Deep-Learning Feature Descriptor for Tree Bark Re-Identification

Mémoire

Martin Robert

Maîtrise en informatique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Deep-Learning Feature Descriptor for Tree Bark Re-Identification

Mémoire

Martin Robert

Sous la direction de:

Philippe Giguère, directeur de recherche
Patrick Dallaire, codirecteur de recherche

Résumé

L'habilité de visuellement ré-identifier des objets est une capacité fondamentale des systèmes de vision. Souvent, ces systèmes s'appuient sur une collection de signatures visuelles basées sur des descripteurs comme SIFT ou SURF. Cependant, ces descripteurs traditionnels ont été conçus pour un certain domaine d'aspects et de géométries de surface (relief limité). Par conséquent, les surfaces très texturées telles que l'écorce des arbres leur posent un défi. Alors, cela rend plus difficile l'utilisation des arbres comme points de repère identifiables à des fins de navigation (robotique) ou le suivi du bois abattu le long d'une chaîne logistique (logistique). Nous proposons donc d'utiliser des descripteurs basés sur les données, qui une fois entraîné avec des images d'écorce, permettront la ré-identification de surfaces d'arbres. À cet effet, nous avons collecté un grand ensemble de données contenant 2 400 images d'écorce présentant de forts changements d'éclairage, annotées par surface et avec la possibilité d'être alignées au pixels près. Nous avons utilisé cet ensemble de données pour échantillonner parmis plus de 2 millions de parcelle d'image de 64x64 pixels afin d'entraîner nos nouveaux descripteurs locaux `DeepBark` et `SqueezeBark`. Notre méthode `DeepBark` a montré un net avantage par rapport aux descripteurs fabriqués à la main SIFT et SURF. Par exemple, nous avons démontré que `DeepBark` peut atteindre une mAP de 87.2% lorsqu'il doit retrouver 11 images d'écorce pertinentes, i.e correspondant à la même surface physique, à une image requête parmis 7,900 images. Notre travail suggère donc qu'il est possible de ré-identifier la surfaces des arbres dans un contexte difficile, tout en rendant public un nouvel ensemble de données.

Abstract

The ability to visually re-identify objects is a fundamental capability in vision systems. Oftentimes, it relies on collections of visual signatures based on descriptors, such as SIFT or SURF. However, these traditional descriptors were designed for a certain domain of surface appearances and geometries (limited relief). Consequently, highly-textured surfaces such as tree bark pose a challenge to them. In turn, this makes it more difficult to use trees as identifiable landmarks for navigational purposes (robotics) or to track felled lumber along a supply chain (logistics). We thus propose to use data-driven descriptors trained on bark images for tree surface re-identification. To this effect, we collected a large dataset containing 2,400 bark images with strong illumination changes, annotated by surface and with the ability to pixel-align them. We used this dataset to sample from more than 2 million 64×64 pixel patches to train our novel local descriptors `DeepBark` and `SqueezeBark`. Our `DeepBark` method has shown a clear advantage against the hand-crafted descriptors SIFT and SURF. For instance, we demonstrated that `DeepBark` can reach a mAP of 87.2% when retrieving 11 relevant bark images, i.e. corresponding to the same physical surface, to a bark query against 7,900 images. Our work thus suggests that re-identifying tree surfaces in a challenging illuminations context is possible. We also make public our dataset, which can be used to benchmark surface re-identification techniques.

Contents

List of Tables

List of Figures

À ma fille Rose, pour qu'elle
sache que tout est possible.

The good thing about science is that it's true whether or not you believe in it.

Neil deGrasse Tyson

Remerciements

L'achèvement de ce travail est le moment idéal pour moi de remercier tous ceux et celles qui ont contribué de près ou de loin à la réussite de ce projet. Aussi, je voudrais dire un merci tout spécial à ma conjointe, Valérie Côté, pour m'avoir supporté durant ce long et ardu projet. De plus, je tiens à remercier mon superviseur, le Professeur Philippe Giguère, pour ses nombreuses corrections et conseils, même si nous n'avons pas eu la chance de se croiser aussi souvent que nous l'aurions souhaité. Puis, j'aimerais dire un sincère merci à Marc-André Fallu, pour son amitié sans failles et son aide pour trouver des arbres bien identifiés pour ajouter à mes données. Il me faut aussi dire merci à Fan Zhou, pour son support essentiel dans mes derniers efforts de collecte de données. Finalement, merci au Fonds de recherche du Québec – Nature et technologies (FRQNT) pour le financement de ce projet.

Introduction

Nowadays, globalization is an undeniable fact, and it comes with lots of challenges. With the rise in the production and consumption of many goods and merchandise, transport logistics became a key challenge of our era. The tracking of objects such as merchandise thus became a necessary concept in many fields, and tracking within the supply chain is now a vital element of the Industry 4.0 philosophy. For example, in Cakici et al. (2011), they look at the radiology industry and evaluate the benefit of switching from an old tracking system using a barcode system and standard inventory counting practice to a new tracking system using Radio Frequency Identification (RFID) technology with a redesign of the business process to fully exploit the technology. They show that the new RFID system would allow a continuous review of the inventory and also reduce shrinkage by tracking expiration. According to Cakici et al. (2011), radiology practice faces difficult operational challenges that new tracking systems could alleviate and help make substantial savings. For Hinkka (2012), it goes without saying that RFID systems show real benefit for object tracking in the supply chain. This is why they focus on the reasons behind the slow adoption of such new tracking technology in supply chain management. Using a literature review, they notice that most published articles concentrated on the technical problems related to RFID, while the problem currently preventing the adoption is more organizational or inter-organizational.

Closer to our subject, tracking objects or products in the forestry industry would consist in monitoring and managing the forest while tracking logged trees from the forest to their entrance in the wood yard. In Sannikov and Pobedinskiy (2018), they proposed a theoretical information system based on RFID to keep a continuous input of the forest state, measure the quantity of harvested wood, and track wood transportation. By building a network allowing communication between RFID device and receiver, they say it would enable the best decisions to be made and help switch from a manual forest management to a remotely automated process. Then in Mtibaa and Chaabane (2014), assuming a forestry company already tags its resource with RFID devices, they propose a software system able to connect to an application layer (XML files and RFID tag). From there, the system can store and process raw data that become available through a web portal that provides both the business and client with easy access to reports and a traceability interface of the resource. According to Mtibaa and Chaabane (2014), such a system would allow companies to engage in product certification

and optimize the distribution of their raw material as well as the best location to keep their inventory.

In the context of mobile robotics, localization is fundamental to enable the robot to move around in his environment in an informed way. In Hellstrom et al. (2009), they use multiple sensory receivers such as a laser scanner, a Global Positioning System (GPS), a gyro, a radar and the vehicle odometry in combination with their new path tracking algorithm developed to follow a learned path based on the steering command to make a 10 tonnes Valmet 830 forwarder follow a precise 160-meter long path. They tested their new algorithm on a Valmet 830 forwarder to simulate as much as possible autonomous wood forwarding in a forest environment. However, the primary sensor they used to re-evaluate their vehicle position was their GPS system that was precise to $2cm$ in their control environment. As they mentioned themselves, GPS systems need a clear view of multiple satellites, have their accuracy sensitive to environmental conditions, and need a stationary receiver to correct recurrent bias. So, forests are a harsh environment where these flaws could prove critical for localization systems. This is why our research aims to provide a reliable localization method based on tree bark to be used as visual landmarks. The importance of landmarks in localization systems comes from their ability to make loop closure in algorithms such as Simultaneous Localization and Mapping (SLAM). In this algorithm, the robot moving in its environment detects interesting landmarks and then stores their descriptions associated with their location together in a database. This enables newly-seen landmarks to be compared with the ones previously seen in the database. When a newly detected landmark (referred to as A here) is matched with a previously-seen one (referred to as B), the assumed location that would be given to landmark A is then compared with the assigned location of the landmark B. This allows an evaluation of the error incurred by the distance travel since having seen the landmark B. That way, these landmarks enable the repositioning of the robot and the correction of the probabilistic map it is simultaneously building. Looking at Figure 0.1, we can see a simplistic example of a robot detecting a previously-seen landmark on his path that will allow him to readjust his map and position hypotheses.

In the work of Smolyanskiy et al. (2017), they present a micro aerial vehicle (MAV) system able to navigate a 1-kilometer forest trail autonomously. Their system uses a 3DR Iris+ quadcopter equipped with sonar and lidar to calculate altitude and velocity but relying on a single forward-facing Microsoft HD Lifecam HD5000 USB camera to calculate the MAV orientation and lateral offset from the forest trail while also detecting objects and obstacles on the trail. Thus, they use monocular SLAM to navigate and follow the current forest path that, in their experiment, did not need any loop closure, since they tested their system on trails without any intersection and 1 kilometer or less in length. Thus, they showed a high capacity to follow forest trails autonomously but would need a better re-localization system for longer trails with many similar path junctions. A great example of the usefulness of landmarks in a

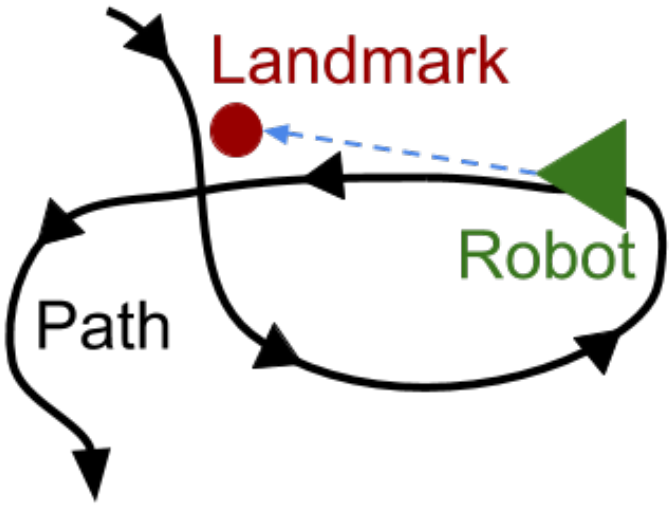


Figure 0.1: Previously seen landmark recognized a second time by a mobile robot.

more complex path is in Ramos et al. (2007). In their paper, they propose a system to detect, describe, and associate landmarks to improve the SLAM algorithm in an outdoor setting. This setting is interesting for us since it is an urban park populated by trees that they use as landmarks, among others. Their approach begins by using laser information to select a possible landmark region. Then, cropping the image from the camera to the proposed region, they classify the crop as a landmark or not. If the image patch is considered as a landmark, they make an appearance model based on the appearance described by a mixture of Gaussians, fused with the position information. Finally, to retrieve previously-seen landmarks, they compared the new appearance model with the ones that were stored using the gated nearest neighbor method. They compared their approach based on an estimated trajectory on a 1.5-kilometer-long path containing several loops using the GPS trajectory collected as ground truth. The two competitors were simple odometry and the standard EKF-SLAM method that both underperform compared to the proposed approach. However, their results still have room for improvement, and their outdoor environment being an urban park has a more sparse distribution of trees than what is expected in a forest. Moreover, using position information in the appearance model would not allow a robot to solve the exploration problem or recover from the wake-up or kidnapping problem common in mobile robotic.

To allow tracking or localization using trees, one must gather information from theses trees that are available all day and all year while displaying distinctive characteristics from one tree to another one. Thus, we chose to use images of tree bark as the source of information for its availability, but also because we suspect an inherent distinctiveness among tree bark, similar to the fingerprint of humans. From this, we designed a system using the standard pipeline of local feature descriptors to extract the distinctive representation inherent to a bark surface. Basing our approach on this typical pipeline will also allow us to use standard comparison

techniques on which we will rely for the data association step of our system. However, looking at Figure 0.2, we can see six images taken from three different bark surfaces that reveal the self-similarity of tree bark while making clear the effect of lighting variation. Indeed, bark texture induces significant changes in appearance when lit from different angles, and it is also important to note that scale and viewpoint variations considerably affect bark appearance too. So, to track trees or use them as landmarks in localization, one must be able to re-identify them and, in our case, make this re-identification based on bark images. In this thesis, we precisely explore this problem by developing a method to compare images of tree bark and determining if they come from the same surface or not.

Our main research objective can be summarized with the following question:

Is it possible to reliably recognize an individual tree with a visual signature extracted from its bark?

Figure 0.2: In this figure, we can see 6 images taken from 3 different bark surfaces identified by the number 1,2,3. Letters a and b make the distinction between two images of a same surface. Comparing images from 1 to 3 shows the self-similarity of bark from different trees while comparing a and b images allows us to see the effect of different illumination angles.

Another assumption we made is that current hand-crafted local feature descriptors are not adapted to give invariant descriptions in the face of significant changes in illumination and viewpoint on bark images. But, new techniques introduced recently have the potential to learn this invariance. Since the advent of the Convolutional Neural Network (CNN) starting when Krizhevsky et al. (2012) won the ImageNet competition in 2012, these new artificial neural networks showed a high capacity to learn a non-linear function enabling unprecedented generalization and invariance. This is why, in this thesis, we propose to replace the description part of the standard local feature descriptor pipeline by a CNN trained on bark images to output invariant descriptions of bark images. A preview of the qualitative performance is

shown in Figure 0.3, which demonstrates the difficulty of dealing with tree bark and descriptors not tailored for it. However, a weakness of these deep learning networks is their tremendous need for useful quality data to learn efficiently. At the beginning of this project, this was indeed another difficulty with the absence of a dataset tailored to this problem. There are already-existing bark datasets such as Fiel and Sablatnig (2010); Svab (2014); Carpentier et al. (2018), for example. Still, these are geared towards tree species classification, and we not only need a bark dataset allowing bark instance retrieval. We need a dataset that enables direct physical correspondence between specific points in multiple images of the same bark surface.

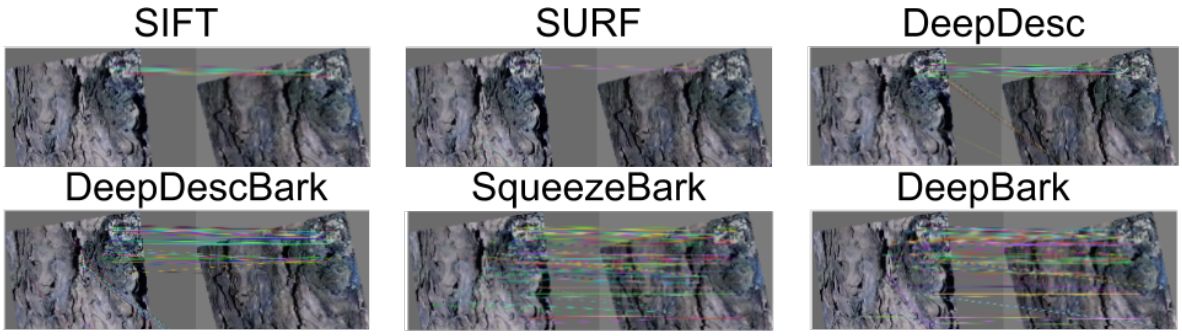


Figure 0.3: Qualitative matching performance of different descriptors, for two images of the same bark surface. Every match shown in the image passed a geometric verification. Some false positive matches remain, due to the high level of self-similarity. Notice the great illumination changes between the image pair, a key difficulty in tree bark re-identification.

To this effect, we collected a dataset of our own, with 200 uniquely-identified bark surface samples, for a total of 2,400 bark images. With these images, we were then able to do bark instance retrieval. This was done by using one of the 12 images of a single bark surface as a query and use our description and data association algorithm to find the other 11 relevant images among all the other bark images used as a test set. The particular fiducial markers we took care to have in every image also provided us with the opportunity to produce a feature-matching dataset enabling the training of deep learning feature descriptors. With all this, we established the first state-of-the-art bark retrieval performance, showing promising results in challenging conditions. In particular, it surpassed by far common local feature descriptors such as Scale Invariant Feature Transform (SIFT) (Lowe (2004)) or Speeded Up Robust Features (SURF) (Bay et al. (2006)), as well as the novel data-driven descriptor `DeepDesc` (Simo-Serra et al. (2015)).

In short, our contributions can be summarized as follows:

- We introduce a novel dataset of tree bark pictures for image retrieval. These pictures contain specific fiducial markers to infer camera plane transformation.
- We use our novel dataset to train a Deep Learning local feature descriptor adapted for

bark images using standard neural network architectures and establish the new state of the art performance for bark re-identification.

- We show that bark surfaces are sufficiently distinct to allow reliable re-identification under a challenging context.
- We compare hand-crafted and deep learning methods on the bark re-identification task and show the necessity of the latter to achieve satisfactory results.

To efficiently understand our approach, the first chapter of this thesis gives the theoretical foundation we used and the current state of the art of the work related to our research. This background information will, for one, cover the subject of computer vision. More precisely, it will discuss the different tasks related to image retrieval along with several ways to describe images in a meaningful way and the useful measures to evaluate this kind of task. Then, an overview of concepts from machine learning about our project will be presented. Among these concepts are the CNN and the objective functions used to train them. We will also look at the latest work on Deep Learning descriptors and the recent papers working on the subject of bark images.

With this knowledge, we will then move toward the second chapter describing our data and methods. These two concepts will be mixed together. That is because we cannot speak of the composition of our dataset without going through the detail of the methodology to collect the said data. Also, any methods discussed in this chapter are closely related to our data, since they are designed to transform, describe or compare bark images from our dataset.

Then, after the theory and methods of the two first chapters, we present the results of our experiments and discuss them in length in chapter 3. To begin, we look at how to choose the optimal hyperparameters for a descriptor and investigate the impact of the size of the data used to train our Deep Learning local feature descriptors. After that, we compare the results of several descriptors being hand-crafted or learned and then examine the capacity of some of these learned descriptors to perform on tree bark species never seen in training. Following this, we produce our most convincing results by testing our best descriptor in a setting composed of bark images reduced in size while adding thousands of similar but unrelated bark images to try to reach the limit of the descriptive ability of our descriptor. Then, before ending this chapter, we evaluate the possibility to speed up the approach along with the possible decrease in performance and explore the distributions of the scores given by different image matching methods to discuss the problem of novel location identification.

Finally, we end this thesis with a conclusion, where we highlight some of the critical results we obtained and give a summary of the work that took us there. Subsequently, we take a moment to reaffirm our contributions before using them as a stepping stone, leading us to the future work it enables, while considering the weakness that should be addressed. Note that

this work can also be found among the published article of the 2020 Conference on Computer and Robot Vision (CRV).

Chapter 1

Background and related work

Before delving deep into our methodology, it is essential to start with the building blocks that led us there. In this chapter, we cover two broad subjects that ultimately take us to an efficient way to compare images of bark together. We begin by exploring the fields related to describing images and how to compare them. This mostly covers issues such as the formulation of image retrieval for different problems in section 1.1, the descriptors projecting whole images to compact descriptions in section 1.2, the descriptors giving invariant descriptions of interesting regions of images in section 1.3 and the Bag of Words (BoW) technique to summarize multiple descriptions of interesting images regions in section 1.4.

Then, we discuss topics in machine learning relevant to our problem. For instance, we start by presenting a brief overview of Convolutional Neural Network (CNN) in section 1.5. Logically following this is metric learning in section 1.6, since it is essential to defined a well-suited learning objective for neural networks. After this, we discuss the multiple ways in which such a network can be trained to give a unique image description in section 1.7. Next, because of the loss of performance by learned descriptors when faced with unseen data from datasets too different, we cover the subject of domain generalization in section 1.8. We also review the latest works about bark and wood images relevant to our approach in section 1.9. Before we conclude this chapter in section 1.11, we go through different metrics useful to evaluating the numerous ways to describe and match images together in section 1.10. Works cited in this chapter have been aggregated in a table that is available at the end of the chapter, in section 1.12.

1.1 Image Retrieval Formulations

Comparing two images directly by their content is useful for many computer vision applications. However, it is generally a highly difficult problem (Wan et al., 2014; Neetu Sharma et al., 2011). This is due in part to images having high dimensionality (the number of pixels times the number of color channels), making the semantic relevant content hard to extract. The

general case where having compact and semantically correct descriptions of images is useful, is the case of image retrieval. However, this task usually is constrained to specific problems, allowing the descriptions of images to focus on particular elements in images related to the constraint problem.

The problem of *image retrieval* can be defined as follow: given an image as a query, the goal is to find other images in a database that look similar to the query one. This was first addressed generically as Content-Based Image Retrieval (CBIR), which appears in conjunction with the consistent growth of images database that became harder to search as they grow, since it led to an increase in wrongly labeled images and that visually going through all images requires more time. In the beginning, searching images for this task was typically accomplished using more simple techniques such as PCA (Sinha and Kangarloo, 2002) or color histogram based on HSV color model (Sural et al., 2002; Kaur and Banga, 2013) and RGB color model (Neetu Sharma et al., 2011). There was also a discussion on the best way of ranking images by similarity to the query Zhang and Ye (2009) and the effect it has on the system performance. But, more recent works have started approaching the CBIR task with deep learning methods as in Wan et al. (2014); Arandjelovic et al. (2018), and there is also Zheng et al. (2016b) that use CBIR as the common ground to compare SIFT based descriptors and deep learning methods. For more in-depth information about CBIR, we strongly suggest to look into the survey made by Datta et al. (2008).

The problem was eventually tailored to specific domains, such as object recognition (Wang et al., 2017b; Sohn, 2016; Wan et al., 2014; Zheng et al., 2016b; Yi et al., 2016; Rocco et al., 2018; Sivic and Zisserman, 2003; Lowe, 2004; Bay et al., 2006). In this formulation, it consists of using a specific object image as a query to find images that contain similar objects, without necessarily showing the searched object exclusively. For example, in the well-known work of Sivic and Zisserman (2003), they showed qualitative results for objects such as a poster, a sign, or an alarm clock. For the poster, they retrieved 20 images, all containing the query object, but with only small variations of the object in the image. Then, for the sign, they retrieved 31/33 correct images and 53/73 for the alarm clock, but here again, it was the same object that needed to be retrieved. Object recognition becomes much harder when you consider retrieving different objects that are relatively similar. For instance, in Sohn (2016), they tried to retrieve cars by model and flowers by species. This resulted in a recognition accuracy for cars of 89.21% and 85.57% for flowers. However, one could make the problem even harder by trying to cluster unseen objects together. For such a task, Sohn (2016) reported the best Harmonic Average of the Precision and Recall (F1) scores of 28.19% for online products, 33.55% for cars, and 27.24% for birds.

Another popular application is face recognition (Liu et al., 2017; Wang et al., 2017a, 2018a,b; Chopra et al., 2005; Schroff et al., 2015; Sohn, 2016; Ming et al., 2018; Sun et al., 2014a; Parkhi et al., 2015; Wang and Deng, 2018; Sun et al., 2016; Zhang et al., 2014; Taigman et al., 2014;

Sun et al., 2015; Taigman et al., 2015; Sun et al., 2014b). In this case, the goal can either be *i)* to correctly classify the identity of a person based on his face image or *ii)*, taking two different images of a face and correctly assigning a label to the two images as being the same person or not. With the recent explosion of data and the easiness of keeping memories of people with images, the field of face recognition made a quick progression toward high performance. We see this phenomenon with companies such as FaceBook that built a dataset with 4.4 million labeled faces from 4,030 people in Taigman et al. (2014). They used a CNN and a face alignment technique to achieve 97.00% accuracy on the known benchmark Labeled Faces in the Wild (LFW) (Huang et al., 2007; Learned-Miller, 2014) and 91.4% accuracy on the YouTube Faces DB (YFD) (Wolf et al., 2011) benchmark. Following this, Google in Schroff et al. (2015) worked with a dataset of 8 million identities, totaling 100-200 million images. With this, they achieve a remarkable 99.63% accuracy on LFW and 95.12% accuracy on YFD, without using anything other than a CNN. However, this dataset size is a challenge to manage on its own. This is why Facebook in Taigman et al. (2015) also worked on how to deal with such large datasets. For this, they made a new dataset containing 10 million different people with around 50 images each, giving roughly 500 million images. They reported a score of 98.0% on the verification task of LFW, which is more difficult than the recognition task. They obtained this score with a single network compared to Sun et al. (2014a) who achieved 99.15% using an ensemble of hundreds of CNN in combination with information from the LFW images. With all the work in face recognition, we added Table 1.1 to give a better overview of the type of datasets available for this task.

Dataset	Available	People	Images
Klare et al. (2015) (IJB-A)	public	500	5712
Huang et al. (2007); Learned-Miller (2014) (LFW)	public	5K	13K
Wolf et al. (2011) (YFD)	public	1595	3425 videos
Sun et al. (2014b) (CelebFaces)	public	10K	202K
Yi et al. (2014) (CASIA-WebFace)	public	10K	500K
Guo et al. (2016) (MS-Celeb-1M)	public	100K	about 10M
Taigman et al. (2014) (Facebook)	private	4K	4.4M
Schroff et al. (2015) (Google)	private	8M	100-200M
Taigman et al. (2015) (Facebook)	private	10M	500M

Table 1.1: Table adapted from Guo et al. (2016). Standard datasets used in face recognition tasks. The number of different identities available in each dataset is the people column, and the total number of images is in the column images. However, the critical aspect is the availability of these datasets provided by the column Available, which depicts the advantage of large companies with their large datasets kept private.

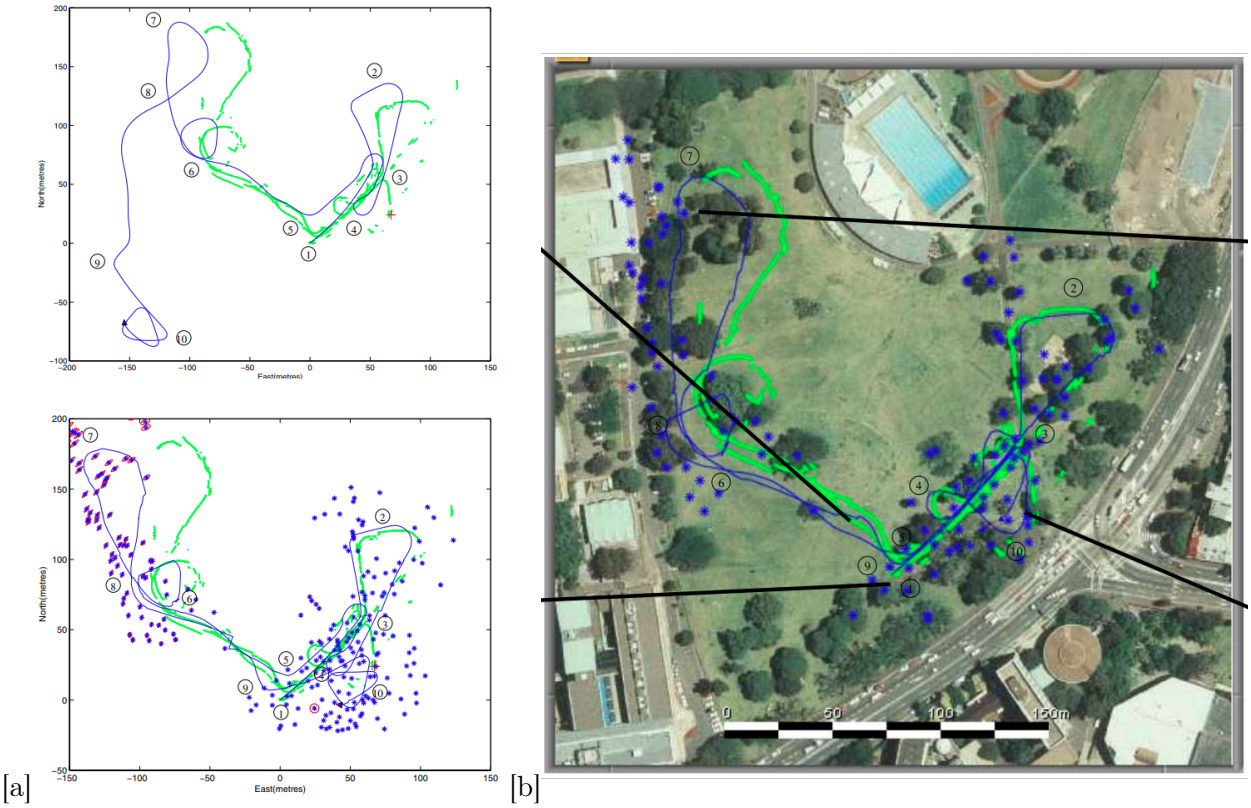
In some other cases, the formulation was used for visual scene recognition (Yi et al., 2016; Rublee et al., 2011; Alcantarilla et al., 2012; Calonder et al., 2010; Arandjelovic, 2012). For the latter, the important aspect is to take an image depicting a certain scene such as a beach, a parking lot, or a forest and retrieve images showing the same kind of scene (but importantly,

not necessarily the same location). In mobile robotics, image retrieval is often used to perform localization. In this field, it is known as Visual Place Recognition (VPR) (Jain et al., 2017; Zheng et al., 2017; Wan et al., 2014; Arandjelovic et al., 2018; Zheng et al., 2016b; Sunderhauf et al., 2015; Cummins and Newman, 2008, 2009; Turcot and Lowe, 2009; Li et al., 2010; Rocco et al., 2018; Ramos et al., 2007). There, the objective is to determine if a location has already been visited, given its visual appearance. The robot could then localize itself by finding previously-seen images which are geo-referenced. This leads us back to the concept of SLAM and the associated problem of loop closure initially discussed in the introduction. However, we did not talk about the difficulty of such a task. An order of magnitude of the problem is provided by the Figure 1.1 from Cummins and Newman (2009), which depicts the loop closure in red on a 1,000 km dataset. By comparing their ground truth in a) with the detected loop closure of their algorithm in b), we see that such a long road will inevitably exhibit difficult places to recognize visually. Also, we can look at Figure 1.2 to evaluate the improvement in localization moving from simple odometry to SLAM algorithm with visual landmarks, passing by standard EKF-SLAM. Nonetheless, on the right side of the figure where visual landmarks are used, we can still see some errors. It is worth noting that the trajectory is only 1.5 kilometers and is located in an urban park, which proves less difficult than other environments such as a dense forest or a snowy landscape where there is a lot of self-similarity in the scenery.

(a) (b)

Figure 1.1: Figure taken from Cummins and Newman (2009). This figure shows a 1,000 kilometers trajectory with loop closure in red. The (a) panel provides the ground truth, and the (b) panel provides the detected loop closures by the Fab-Map 2.0 algorithm. They correctly detected 2,189 loop closures and predicted six false positives. This gave them a 99.8% precision at 5.7% recall. In the original caption, they mentioned that the long section with no loop closure detected is a highway at dusk.

In the area of video surveillance, the problem of Person Re-Identification (Person Re-Id) consists of following an individual through several security camera recordings (Hermans et al., 2017; Zheng et al., 2017, 2016b; Gray et al., 2007; Zheng et al., 2016a, 2015; Li et al., 2014). This technique implies to learn a function that maps multiple images of an individual to the



[a] [b]

Figure 1.2: Figures taken from Ramos et al. (2007). In each case, the blue line shows the estimated trajectory, and the green line represents the estimated ground truth using GPS information, which is not available in some areas of the urban park. In the top part of [a], the blue line is estimated using only odometry, and in the bottom part, we have an estimate using the standard EKF-SLAM. In [b] is the trajectory predicted by the algorithm using visual landmarks proposed by Ramos et al. (2007). We can see that the proposed method improves a lot over the two other methods, but small errors are still visible, and the trajectory used could be more difficult.

same compact description, despite variation of viewpoint, illumination, pose, or even clothes. This way, if the description is unique for each individual and invariant to change in appearance, person re-identification can then retrieve a single person by computing the same description for this person across multiple cameras despite changes in view angle and different people being present. One of the standard datasets tailored for this task is the viewpoint invariant pedestrian recognition (VIPeR) dataset presented in Gray et al. (2007). However, it only contains 632 image pairs of pedestrians with variations only in viewpoint. By today's standard, this dataset is quite small with its total of 1264 images, but since then, larger datasets have appeared, and the most common are shown in Table 1.2. Numerous works have used these datasets as benchmarks. One of these is Zheng et al. (2017), who used a CNN trained with multiple objectives to achieve a rank-1 accuracy of 83.4 in the single-shot setting of the CHUK03 (Li et al., 2014) dataset. On the Market-1501 (Zheng et al., 2015) dataset, they re-

ported a mean Average Precision (mAP) score of 59.87 in the single query setting and a score of 70.33 in multi-query setting. More recently, better results have been reported by Hermans et al. (2017) on the CHUK03 and Market-1501 dataset, but also on the MARS (Zheng et al., 2016a) dataset. For CHUK03, they achieved a rank-1 accuracy of 87.58 in a single-shot setting. On the Market-1501, they obtain a mAP score of 81.07 and 87.18 in single and multi-query respectively, and lastly, they reached a mAP score of 77.43 on the MARS dataset. It is clear that person re-identification is undergoing a rapid progression in performance during recent years, but one can also see that there is still room for improvement when looking at the latest results.

Dataset	pedestrians	images
Gray et al. (2007) (VIPeR)	632	1264
Li et al. (2014) (CUHK03)	1,360	13,164
Zheng et al. (2015) (Market-1501)	1,501	32,668
Zheng et al. (2016a) (MARS)	1,261	1,191,003

Table 1.2: Standard datasets used for the person re-identification task. The column pedestrian shows the number of different identities available in each dataset, and the total number of images is in the column images. These datasets are also in ascending order of apparition from top to bottom. We can see the growth in data following the passage of time.

1.2 Global Descriptor

In the early days of image retrieval systems, the first techniques to describe images relied on the complete image's properties to perform correspondence. Making histograms from colors as in Neetu Sharma et al. (2011), for instance, can give some mildly reliable information about the image content. For example, an image with a predominance of blue might be an image of the sky or the sea, while an image mostly green will probably contain forests or vegetation. However, this information is not enough to precisely characterize an image and can at best allow the association of images with a loosely similar theme. Then, one can also look into Hue-Saturation-Value (HSV) color format (Sural et al., 2002; Kaur and Banga, 2013) to extract information more robust to illumination changes. In the work of Sural et al. (2002), they used the HSV format to categorize pixels as a specific color if the saturation is low or, if it is high, for instance, as a black or white pixel. This is done to allow the decomposition of images using histograms and segmentation in a more relatable way to how humans perceive light. Another way to globally describe images is to view them for what they are in computer science: matrices. Then, using linear algebra, we can characterize images by extracting their eigenvectors. However, this is not readily applicable for images within a dataset, without further processing. For this reason, Sinha and Kangarloo (2002) further constrained their problem by employing dimensionality reduction with Principal Component Analysis (PCA) to compare images based on the variation of their principal components.

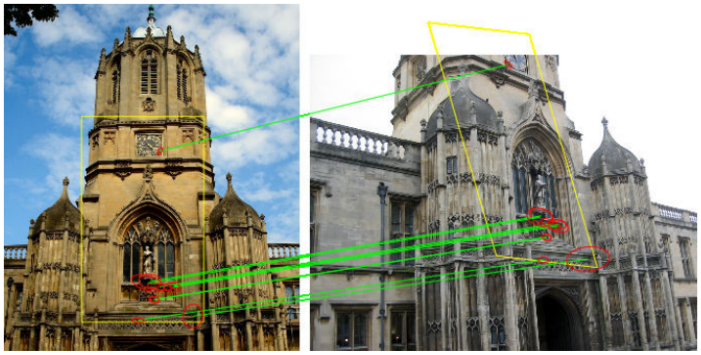
Nonetheless, these techniques may give too much importance to background information, often relating to location and drown small but crucial visual information such as objects or persons, as explained in Datta et al. (2008). Also, they are weak against changes in viewpoint, as mentioned in Lowry et al. (2016).

Since then, other methods have been developed to describe images globally that broadly differ from the one previously mentioned. First, there is the BoW method (Sivic and Zisserman, 2003) that globally describe an image by summarizing the set of all descriptions produced by a local feature descriptor such as SIFT or SURF. There is also a growing interest in using CNN trained on classification tasks to extract a single vector representing a whole image (Wan et al., 2014; Sunderhauf et al., 2015). However, we will not discuss these methods here, because they are the subject of another section.

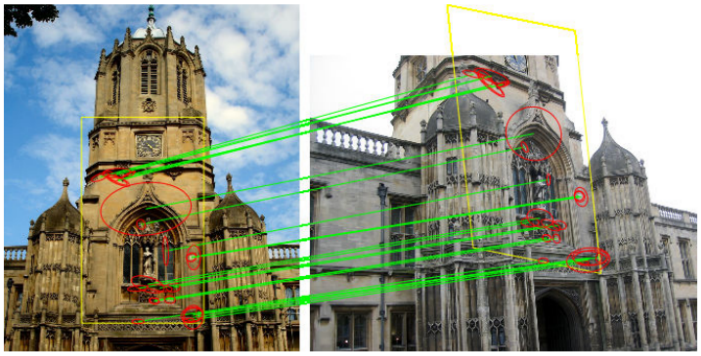
1.3 Hand-Crafted Local Feature Descriptor

Instead of taking the whole image and calculating a global description as a form of summary, local feature descriptors aim at representing an image through a collection of descriptions taken from smaller regions of the image. For these descriptions to be relevant, the small image regions they describe must be a distinctive or meaningful part of the image. The first step in using such a descriptor is to find the location of interesting regions of the image. In the context of local descriptors, these regions are specifically referred to as *keypoints*. These keypoints are found using a keypoint detector, which looks through an image, searching for strong gradients as in Lowe (2004) or searching for corner using pixel intensity as in Viswanathan (2011); Rosten and Drummond (2006). This change in gradient or pixel intensity correlates strongly with edges and corners in an image, which makes them reliable points. The advantage of this technique is that if we can reliably find the same keypoints in an image, we can obtain a nearly invariant representation of the image while mostly including only meaningful parts of it. Once the keypoints are found, the descriptor is then used to describe all of the keypoints surrounding regions following an algorithm devised by an expert. This is the reason why they are called *hand-crafted*.

The goal of a local descriptor is then to summarize the visual content of an image patch taken at the location of a keypoint. This enables a comparison between images based on feature correspondence, as seen in Figure 1.3. The ideal descriptor is *a)* compact (low dimensionality) *b)* fast to compute *c)* distinctive and *d)* robust to illumination, translation and rotations. These are important qualities for local feature descriptors, because the number of descriptions to be made in an image can total over 2000, which can be costly in computation and memory space. A widespread approach of hand-crafted methods to describe image patches is often to rely on histograms of orientation, scale, and magnitude of image gradients, as in SIFT (Lowe, 2004) or SURF (Bay et al., 2006). Different variants have appeared over the years trying to



(a) SIFT (L2 distance): 10 matches



(b) RootSIFT: 26 matches

Figure 1.3: Figure taken from Arandjelovic (2012). It compares the descriptors based on the number of matches retained and the localization quality of the keypoints that have a match. On the left is the query image, and on the right is the image compared with the estimated correspondences. It shows that RootSift offers descriptions that are more distinctive and allow better matching of more locations.

alleviate the computation cost (Calonder et al., 2010; Rublee et al., 2011) or simply trying to increase the performance (Alcantarilla et al., 2012; Arandjelovic, 2012).

1.4 Bag of Words

The Bag of Words (BoW) is a discretized representation of all the information provided by a local feature descriptor. It is calculated from the list of descriptions V generated by the descriptor on an image, which are lists of vectors of reals. The advantage of such a technique against directly using V, is the speed at which two BoW can be compared. Since BoW are discretized summary of a list of descriptions extracted from images, this allows for a fast comparison of these images, which is useful in large datasets. An excellent reference to follow is Manning et al. (2008) in section 6.2, which we explain next.

The idea of BoW in computer vision has been borrowed from the information retrieval domain. As its name suggests, the BoW is essentially a multiset (bag) of every word in a document.

This serves as a summary vector representation of a complete document, using the count of each word present in this document. In computer vision, the idea of words has been extended to the concept of visual word in an image. However, visual words are not defined in advance as words, instead, they represent a cluster composed of similar visual features. These visual features are small parts of an image considered as informative, such as edges, corners, repetitive structures, or any meaningful pattern in an image. This means that visual words must be defined using clustering methods such as the K-mean clustering algorithm from a large set of visual features extracted from available images. Once we clustered our set of visual features into a fixed number k of clusters, we end up with k visual words that become what is known as a visual dictionary *voc* of size k. More formally, the *voc* calculation is done by using a keypoints detector and descriptor to extract all the keypoints descriptions available in a subset of the dataset we want to use for evaluation. Then, these descriptions, which become our visual features, can be clustered to build our *voc*. Finally, we can calculate a BoW representation of unseen images by extracting their visual features and clustering them into our *voc*, giving us the count of visual features for every visual word in an images, as we explain below.

Once we have a relevant *voc* for our dataset, we can calculate the BoW representation of an image I. To do so, the Term Frequency (TF) of each visual feature in I must be determined. The equation for a single TF is written as:

$$TF = \frac{n_z}{|V|}. \tag{1.1}$$

Here, V is the list of descriptions previously computed for the image I using the chosen detector and descriptor pipeline. Then, n_z is the number of descriptions from V being clustered with the visual word z, and $|V|$ is the number of descriptions in our image. Doing this for every visual word in our *voc* gives a normalized BoW representation of I, with the same dimension of the *voc*.

However, some visual words may be present in almost every image of a dataset due to their repetitive nature. For instance, a dataset taken in the street of a city may show street lights in every image, so detecting them will not help to differentiate two images coming from different places. To mitigate this problem, one can calculate the Inverse Document Frequency (IDF) of a visual word from the subset T of our dataset. This enables the adjustment of the BoW by weighting each visual word according to their presence ratio in T. This way, it mostly ignores visual words present everywhere while giving more importance to the less present ones that are more informative. The IDF of a single visual word is defined as:

$$IDF = \log \frac{|T|}{m_z}. \tag{1.2}$$

The numerator term $|T|$ corresponds to the number of images in the subset T of the chosen dataset, which we divided by the total number of images m_z in T having a least one description being clustered with the visual word z. Once every TF in the BoW of an image is weighted by his corresponding IDF, we then obtain a more distinct representation of our image I.

1.5 Convolutional Neural Network

One of the critical developments in computer vision is the Convolutional Neural Network (CNN). The first successful application of CNN was for hand-written digit recognition (Lecun et al., 1998). They began by building a large set of hand-written digits known as the MNIST dataset from the larger NIST dataset. During their experiment, they used their LeNet neural architecture to show the advantage of CNN against other methods such as K-NN classifier, SVM, and fully-connected neural network. However, the usage of CNN did not take on initially, despite its vast potential. The interest for CNN-like architectures only took off when it was first used in the ImageNet (Deng et al., 2009) competition, with AlexNet (Krizhevsky et al., 2012). In this work, they reported a significant winning margin against the best competitor. They succeeded in significantly reducing the current error margin with their AlexNet model trained using a smart GPU implementation of the convolution operation. They also discussed the use of drop-out, non-saturating neurons, and max-pooling that helped achieve their results. It is with this demonstration that the true capability of CNN had been unveiled.

Neuron

This capability comes from a fundamental building block called *artificial neuron*. The connectionism approach of artificial intelligence uses these neurons as their main inference engine. When grouping them in layers and then stacking multiple layers together, it becomes what is referred to as deep neural networks. These networks are part of the subset of machine learning, known as deep learning, which includes CNNs.

Neurons are based first on a linear equation that possess a parametric weight for each input it receives plus a weight bias. This is followed by a non-linear function referred to as an activation function. Being non-linear is a crucial component of the neuron, because it enables multiple representations of an input that would be impossible otherwise. Also, if one would stack several layers of neurons together without non-linearity, the network could then be reduced to a single linear operation. The standard activation function currently used in deep neural networks is the Rectified Linear Unit (ReLU) (Nair and Hinton, 2010). This function is defined as the maximum between the input and 0: $max(x, 0)$. A neuron using a ReLU activation becomes a pattern detector that outputs a positive number only when the input satisfies a particular input pattern. This is demonstrated in Figure 1.4, which shows an example of a neuron receiving five inputs describing a car. The neuron only activates when a car is not too old and shows little mileage. In this toy example, the network might be trying to predict which car a potential client might buy, and this particular neuron learns to respond when a car is relatively new, since this is what a client might want. A latter neuron can discriminate more complex patterns based on the recognized patterns of the preceding network layer and so on so forth.

All Features of a Car

Age	Odometer	Brand	Model	Review
x_1	x_2	x_3	x_4	x_5

Complete Neuron With Weight

$$max(-7.4x_1 - 2.4x_2 + 0x_3 + 0x_4 + 0x_5 + 109, 0)$$

Complete Neuron Equation

$$max(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + b, 0)$$

Simplified Neuron (Without Weight at 0)

$$max(-7.4x_1 - 2.4x_2 + 109, 0)$$

Simplified Cars Sample Table

Cars Label	Age (Year)	Odometer (1000 km)	Neuron Output Before Relu	Neuron Output After Relu
0	4	62	-69.4	0
0	5	33	-7.2	0
0	1	45	-6.4	0
0	9	36	-44.0	0
0	3	51	-35.6	0
0	8	39	-43.8	0
1	3	28	19.6	19.6
1	2	33	15.0	15.0
1	1	18	58.4	58.4
1	3	26	24.4	24.4
1	1	9	80.0	80.0
1	2	37	5.4	5.4

Simplified Neuron Output of Cars Sample

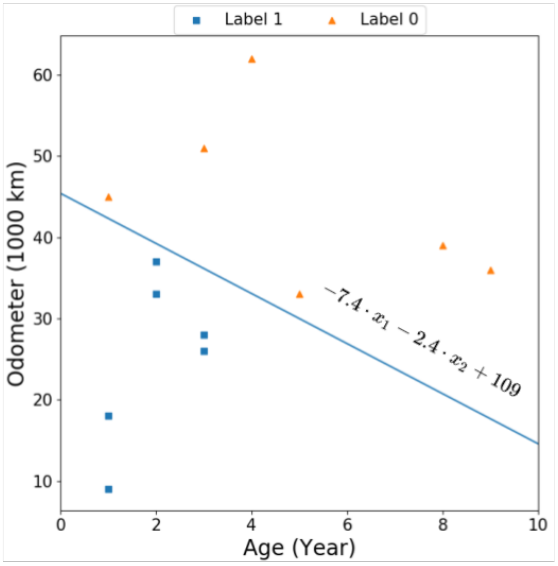


Figure 1.4: Visualization of the neuron discriminating old cars with lots of mileage. Thus, we can see that this neuron will respond to cars that seem relatively new, based on age and mileage. This is based on a toy example where a model tries to predict preferred cars (label 1).

CNN filter

CNN filters are the CNN neuron, but the pattern they search has been specialized for the 2D grid-like arrangement of images. So to find these patterns, CNNs use the convolution operation that essentially multiplies a matrix (the CNN filter) with a corresponding image patch taken at every pixel position. The first two steps of this operation can be seen in Figure 1.5. We can also understand that the filters used in a CNN layer are no more than another linear equation evaluated at a set of pixels. One advantage of this operation is that the matrix format allows us to take into account the spatial structure of an image in which the smaller pattern resides. The other advantage of the convolution is that parameters are shared across image locations, making a single filter capable of finding a recurrent pattern in the image wherever it is. The last significant difference between a fully-connected layer and a CNN one is the output. Where a fully-connected layer gives a new array of features, the CNN

layer will instead output a feature map. When a single filter is convolved against an image, the repeated matrix multiplication produces a result for every pixel position that needs to be kept in the same two-dimensional position to avoid losing the coherence of the image. The resulting matrix visible in Figure 1.5 is an example of such feature maps. The final output of a convolutional layer is then a 3-dimensional tensor that is the concatenation of all the outputted feature maps, since each filter of a CNN layer produces such a feature map.

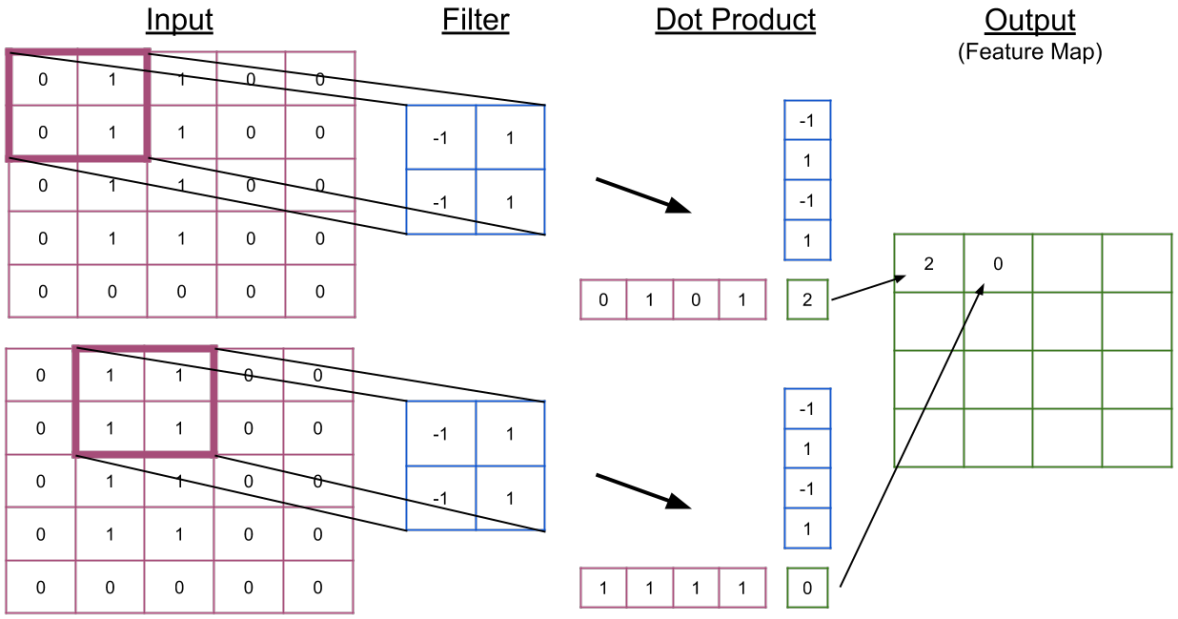


Figure 1.5: Visualization of the convolution operation. In this figure, we can see that the filter is multiplied element-wise at each pixel position, and is equivalent to doing the dot product between two vectors. Note that for brevity, we omit the addition of a weight bias after the dot product operation.

CNN architecture

A network is typically defined by a particular arrangement of blocks, such as convolution layers, number and size of filters, max pooling, etc. Architectures greatly influence the results obtained by deep learning networks. Consequently, they are designed to induce a favorable bias for the task at hand. For example, the simple fact of using convolution operations causes a bias that makes deep networks such as CNN more efficient on images.

One architecture that showed outstanding performance and that is available under different sizes is the ResNet (He et al., 2016). This design intention was to transform the neural network such that it learns the residual of an identity function, at each layer. It comprised a residual block, which is composed of two convolutional layers, which are added to the original input via a skip connection, as displayed in Figure 1.6. This made every subsequent residual block processing a mix of the original input with the features obtained from the preceding blocks. Moreover, the skip connection enables the gradient coming from the final

prediction error to be easily back-propagated to the first layer of the network. This greatly facilitates the training of ResNet-like networks and increases the design flexibility in terms of the number of layers, which in turn gives better results when using more layers, as reported in Figure 1.7. Consequently, many ResNets models are publicly available in common deep learning framework. The complete architecture of a ResNet with 34 layers can be seen in Figure A.2.

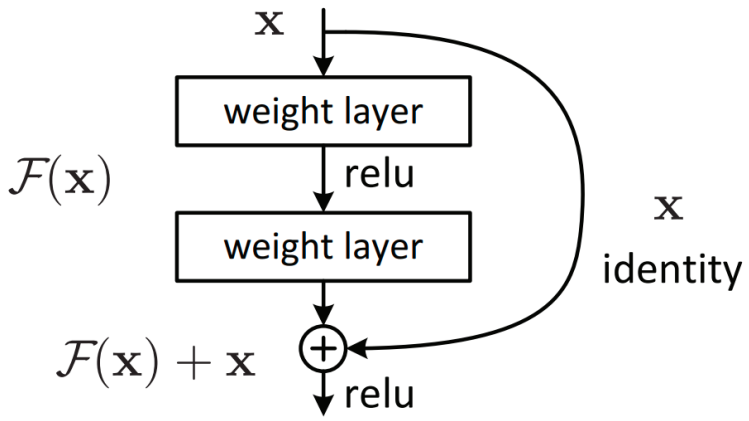


Figure 1.6: Residual block, as presented in He et al. (2016). The $F(x)$ function is defined by the two-weight layers and the ReLU activation function. These weight layers can be any kind of layer, such as convolutional ones. The improvement made compared to traditional networks comes from the addition of the original input x to the output of $F(x)$. This addition between input and output helps to back-propagate the gradient. Also, this forces each block to learn the residual of an identity function.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Figure 1.7: ResNet performance as presented in He et al. (2016). These results were obtained using single-models on the ImageNet validation set and are reported as error rates (%). The number beside ResNet networks indicates the number of layers. From this, we can see a clear improvement following the increase in layers.

Another interesting CNN architecture variation is SqueezeNet (Iandola et al., 2016). In their work, they proposed to focus on the sheer number of parameters in SqueezeNet, instead of trying to improve raw performance. For a quick comparison, the small 18-layers ResNet has 11,689,512 parameters, while the SqueezeNet 1.1 version only has 1,235,496. This gives a nine-fold reduction of the number of parameters while keeping an accuracy similar to AlexNet. Such a feat was achieved with a new CNN building block named *Fire Module*, and is depicted in Figure A.1. It was designed to employ three strategies aiming at reducing the number of parameters needed for convolution, while keeping the highest accuracy possible. Their first strategy was to reduce the number of costly 3x3 filters by replacing a portion of them by smaller 1x1 filters. Then, since filters are designed to be applied on every feature map simultaneously, the *Fire Module* incorporates a layer of 1x1 filters squeezing the number of feature maps before using 3x3 filters on it, thus reducing the depth of the 3x3 filters. Finally, the last strategy was to delay the feature map resolution reduction in the network, considering that He and Sun (2015) have shown that such a strategy can improve performance.

1.6 Metric Learning

As discussed in section 1.1, deep learning is now an essential component in image retrieval. This mostly spurs from the fact that deep learning networks can learn complex and non-linear representations that allow high-dimensional input to be easily classifiable in a linear fashion. This, in turn, means that a deep neural network can project its inputs to a compact representation that is meaningful for the task at hand. For a deep neural network to solve the ImageNet challenge, for example, it had to learn a projection that could take millions of images and send them to a small and meaningful representation to be linearly classifiable in 1000 categories. This hard task generated deep models able to encode a large array of complicated images features from the real world into low dimensional vector representations. These models essentially became global image descriptors and began to be used as such in problems such as VPR (Sunderhauf et al., 2015).

However, certain problems need more than having linearly-separable and meaningful vectors for an arbitrary number of classes. For example, in face recognition, we need to differentiate two distinct persons by their faces. Consequently, a deep model which could accomplish this for two, a hundred or even thousands of faces may not be enough for an application dealing with millions of unseen peoples. This becomes a problem about instance-level representations instead of classes and is useful, among other things, for instance-matching problems. Thus, the objective of a deep learning model used for instance retrieval, should be about making instance representations as distant from each other when different instances are involved, while keeping variation of the same instance as close as possible. This problem is well-adapted to the paradigm known as metric learning. More specifically, metric learning aims at designing a loss function that would allow a metric to be learned in the representation space. Below,

we discuss recent progress made in metric learning in subsection 1.6.1. As classification losses provide good performance and greatly impact the representational power of deep models, we also discuss the use of theses losses in the metric learning setting in subsection 1.6.2.

1.6.1 Metric Loss

Contrastive Loss

Distance metric learning is an approach that tries to learn a representation function to be used with a distance function $d()$ between data points x. For example, we have a dataset S of labelled data $(x, y) \in S$ with x being an example and y its class label. We also have a function f which outputs a vector $f(x) \in R^d$. In metric learning, the goal is to learn this function f to minimizes the distance between the feature vectors, $d(x_1, x_2) = ||f(x_1) - f(x_2)||_2^2$, when both data points x_1 and x_2 belong to the same class $y_1 = y_2$. It must also maximizes the distance when x_1 and x_2 belong to different classes ($y_1 \neq y_2$). This function f can then be used to compare unseen examples similar to the ones in S. The loss doing this base scenario can be found in Hadsell et al. (2006); Chopra et al. (2005); Simo-Serra et al. (2015); DeTone et al. (2018) and is called the contrastive loss, as defined here:

$$L(W, Y, X_1, X_2) = (1 - Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}\{max(0, m - D_w)\}^2. \tag{1.3}$$

In equation 1.3, Y is a label directly stating if a pair of examples (X_1, X_2) are similar ($Y = 0$) or not ($Y = 1$). W are the parameters of D_w that represent the distance function to be learned. This function should minimize the distance when $Y = 0$ and maximize to the extent of a margin m when $Y = 1$.

Triplet Loss

Instead of bringing together similar pairs in the embedding space as much as possible, one can try to make the inter-class variation larger than the intra-class one in the embedding space of the descriptor, as described in Figure 1.8. This is called the *triplet loss* (Schroff et al., 2015; Ming et al., 2018; Arandjelovic et al., 2018; Li et al., 2017a). It is computed from three examples (hence its name), made of an anchor example x, a positive example x^+, and a negative example x^-. They are used to learn the following inequality:

$$||f(x) - f(x^+)||_2^2 < ||f(x) - f(x^-)||_2^2. \tag{1.4}$$

This inequality can then be translated to an actual loss:

$$L(x, x^+, x^-) = m + ||f(x) - f(x^+)||_2^2 - ||f(x) - f(x^-)||_2^2. \tag{1.5}$$

This Equation 1.5 pushes the distance between a positive x^+ and a negative x^- pair further than the selected margin m. In the case of Wang et al. (2017b), they tried to improve the triplet loss by adding constraints based on triangle geometry, to ensure the proper movement of the triplet sample in the embedding space during learning.

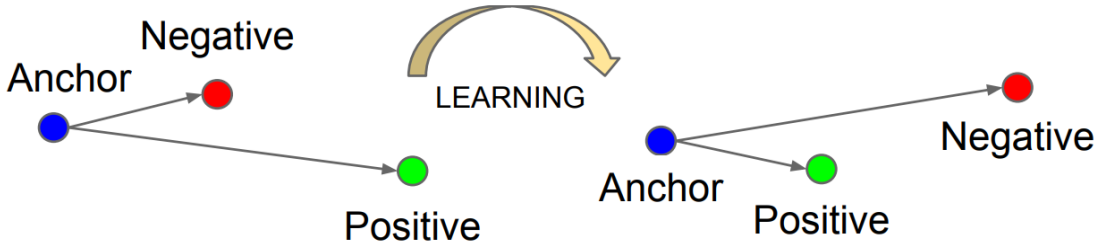


Figure 1.8: This figure shows the result after learning the representation with the triplet loss. The goal of this loss is to make the distance in representation space between two points of the same class A (here the anchor and the positive) smaller than the distance between a point of class A and a point of a different class such as B (here the negative) by a certain margin. It will cluster points of the same class together while assuring a certain margin with other classes. This figure was taken from Schroff et al. (2015).

Triplet Loss Variant

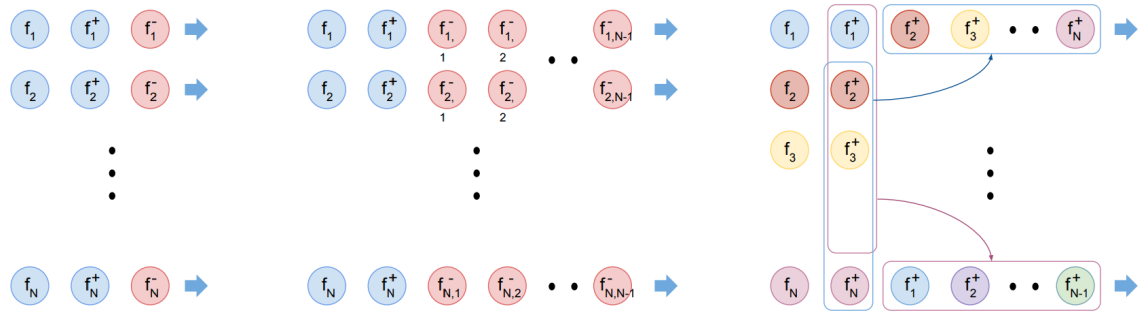
Sohn (2016) created the N-pair-mc loss to improve upon the idea of moving one pair of positive examples away from the negative pair one at the time. There, the goal is to use an ingenious batch construction process, shown in Figure 1.9, to enable the comparison of one positive pair x, x^+ with multiple negatives examples x_i at a time. This loss is expressed as:

$$L(\{x, x^+, \{x_i\}_{i=1}^{N-1}\}; f) = \log(1 + \sum_{i=1}^{N-1} \exp(f^\top f_i - f^\top f^+)), \tag{1.6}$$

with the N-pair-mc loss using the $N-1$ negatives examples x_i to move the positive pair x, x^+ away from every one of them. This has the critical benefit of avoiding that the pair x, x^+ of a triplet (x, x^+, x_i^-) gets closer to an unseen negative example x_j^- during a mini-batch learning update. Also, Hermans et al. (2017) showed that comparing the hardest positive pairs (dissimilar examples that should be similar) with the hardest negative pairs (similar examples that should be dissimilar) in a mini-batch can lead to even better results in some instances.

1.6.2 Classification Loss

With the loss previously explained in subsection 1.6.1, our function $f(x)$ has generally learned to output representation easily comparable in Euclidean distance. The kind of representation learned when classifying with the cross-entropy loss is different, but still relies on a form of distance. A cross-entropy loss is normally used to learn the categorization of instances of a pre-defined problem by transforming the final output of a network into a set of class probabilities. However, this final output given by the network can be seen as the non-normalized angle between the last layer and the input it receives, as explained in Figure 1.10. When considering this angle as a distance, it makes the penultimate layer representation learned with classification useful in metric learning, as we explained below.



(a) Triplet loss (b) $(N+1)$-tuplet loss (c) N-pair-mc loss

Figure 1.9: This figure shows the ingenious batch construction process used by the N-pair-mc loss to reduce the number of evaluations needed to transform each example in their vector representation. In a), we can see the standard Triplet loss that chooses N triplet for a batch, resulting in the need for $3N$ evaluations. In b) is a naive selection of examples for a $(N+1)$-tuplet loss batch, resulting in the need for $(N+1)N$ evaluations. Finally, in c), we can see that to reduce the number of evaluations needed, the N-pair-mc loss selects only N positive pairs of examples. Then, it used half of these pairs as the negative examples to create N $(N+1)$-tuplet, reducing the total number of evaluations to only $2N$. This figure was taken from Sohn (2016).

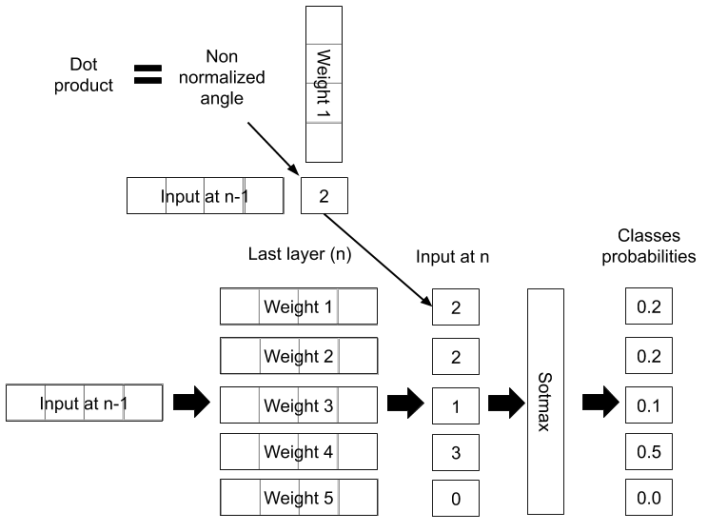


Figure 1.10: The dot product of the input representation at the $n-1$ layer and the weights of the last layer n can be considered as the non-normalized angle between them. This angle can thus serve as a distance and lead us to think of the input representation given by the $n-1$ layer as a representation learned using a distance. The cross-entropy loss indirectly does this by making examples of the same class have similar angles and make classification useful in metric learning.

Cross-entropy Loss

Considering that the objective is to obtain a sufficiently-diverse representation to differentiate

two distinct instances of a dataset, it may not be mandatory to directly enforce a distance between every instance. Instead, Sun et al. (2014b) use a cross-entropy loss to learn the classification of 10,000 different faces. Having many classes forced their CNN model to learn a function that output sufficiently-detailed representation before the last layer to be successfully used in a face verification task.

Similarity Loss

Even if classification can be used for metric learning, it is not the most suitable loss for verification, i.e., to decide if an unseen image pair is considered the same or not. To tackle this problem, a verification loss using binary cross-entropy (Xufeng Han et al., 2015; Sun et al., 2016) was developed to directly output the probability of two images being the same (0 different, 1 same). If we consider this probability as a distance, this loss becomes similar to the contrastive loss by pushing the distance between similar examples to 1 and different examples to 0.

Multi Loss

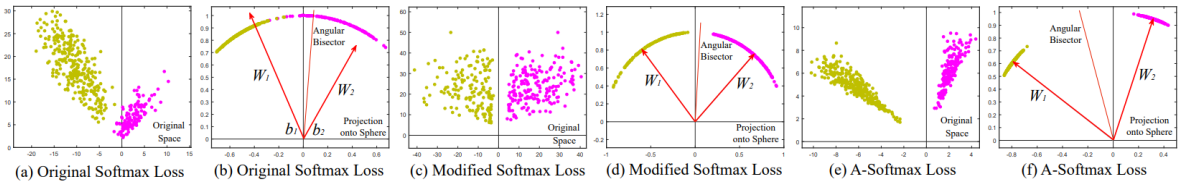
With the great success of losses based on learning distances and the significant power of classification, it is not surprising to see that this led to experiments which combined losses to improve over one another. In Sun et al. (2014a, 2015), they trained their network by carrying out classification *and* using the contrastive loss at the same time. Similarly, Parkhi et al. (2015) used a cross-entropy loss in conjunction with the triplet loss. In the case of Zheng et al. (2017); Taigman et al. (2014); Taigman et al. (2015), they combined classification with a similarity loss objective using binary cross-entropy. With all of these methods, Wan et al. (2014) made a comparison of a pre-trained network on ImageNet with the same network fine-tuned on similar data to the test set using either a cross-entropy loss or a triplet loss. They concluded that a pre-trained network while offering good results, can have its performance significantly boosted by fine-tuning it with either loss.

Angular Loss

Since the performance shown by a cross-entropy loss is still relevant, some people started to address one of its disadvantages, namely the lack of proper decision boundaries. Liu et al. (2017); Wang et al. (2017a, 2018a,b) alleviated this problem by modifying the softmax to incorporate an angular margin that must be overcome by the network to classify an example correctly. This problem can be seen in Figure 1.11, which also depicts the effect of adding a margin in the softmax classification loss in either the non-normalized vector space or the l^2 normalized vector space.

1.7 Learned Local Feature Descriptor

Recently, with the revolutionary appearance of deep learning in computer vision, *data-driven* approaches have dominated the landscape. The assumption is that with the right dataset,



(a) Original Softmax Loss (b) Original Softmax Loss (c) Modified Softmax Loss (d) Modified Softmax Loss (e) A-Softmax Loss (f) A-Softmax Loss

Figure 1.11: Figure taken from Liu et al. (2017). Toy experiment showing 2D representations learned by a CNN on the CASIA face dataset. Three losses were used to train the neural network, which are the softmax loss (which Liu et al. (2017) defined as the combination of softmax and the cross-entropy loss), the modified softmax loss and the A-Softmax loss. For each loss, there is a positional representation and an angular one. Two classes of face features are represented in each graph, one in yellow and one in purple. This figure thus displays the lack of angularity and margin in the class representations of the softmax loss, which made the A-Softmax loss perform better.

a data-driven approach should learn a more meaningful and distinct representation of image patches, while being invariant to changes in lighting and viewpoint conditions. This is in contrast with the hand-crafted approach, in which datasets play a less-direct role. A good example of data-driven approaches is in Brown et al. (2011), where they made a patches dataset using the 3D reconstruction of the Statue of Liberty (New York), Notre Dame (Paris), and Half Dome (Yosemite) from the photos and the multi-view stereo data of Snavely et al. (2008); Goesele et al. (2007). This dataset, now known as Multi-View Stereo (MVS) dataset, is composed of a large set of 64x64 patches, mostly depicting buildings.

Consequently, a descriptor train with such a dataset will be biased toward buildings and may very well fail on other datasets, such as a household objects dataset, to name one. Nevertheless, Brown et al. (2011) showed with their dataset that learning a parametric function on one of the three buildings and testing on another one, made their descriptor offer a better performance than a generic one such as SIFT. Despite the bias induced by the dataset, learned descriptors can thus be useful as long as the data used to train them is appropriately selected for the task.

With encouraging results of machine learning approaches such as Brown et al. (2011), it was expected to see a deep learning approach such as Simo-Serra et al. (2015); Paulin et al. (2015) apply CNN to the local feature descriptor task. In the case of Paulin et al. (2015), they approximate a kernel map in an unsupervised way using the Convolution Kernel Network (CKN) from Mairal et al. (2014) that was developed to be more simple, easy to train and invariant than normal CNN. Then, their network takes image patches as input and produced kernel map approximations, which serve as the descriptions. To train this descriptor, they prepared a new patches dataset using landmark images of Rome taken from Li et al. (2010). They tested their approach on the unseen part of their Rome dataset, as well as other benchmarks, and showed on par or better performance than supervised CNN.

For their part, Simo-Serra et al. (2015) used the MVS dataset to devise a training scheme using

the contrastive loss. Their CNN was then trained to output similar vectors when given two patches of the same region and dissimilar vectors for patches of different regions. A sample of these patches used by Simo-Serra et al. (2015) from the MVS dataset can be seen in Figure 1.12. In this figure, you can observe a pair of similar patches for each different region. When tested on this dataset, they outperformed SIFT for every split used for testing, outperformed the VGG approach (Simonyan et al., 2014) fine-tuned on the data in all splits except one and using different datasets further demonstrated robustness to rotation and generalization.

Instead of outputting vectors that can be compared, Xufeng Han et al. (2015) decided to develop a two-step process that describes two image patches and then directly provides the matching probability between these two patches. To do so, they first designed a deep learning network composed of two CNNs with shared parameters. Each pipeline describes the two image patches at the same time, and then these internal descriptions are fed to a fully-connected network that outputs the similarity probability. They further explain a two-step scheme to rapidly compare images, which is to describe every image patch using their CNN and then compare all pairs using the fully-connected network. When compared on the MVS dataset, they outperformed variant of SIFT and Trzcinski et al. (2012); Simonyan et al. (2014); Brown et al. (2011) approaches. They also showed that they could achieve state-of-the-art results while using quantization to make use of fewer bits per feature.

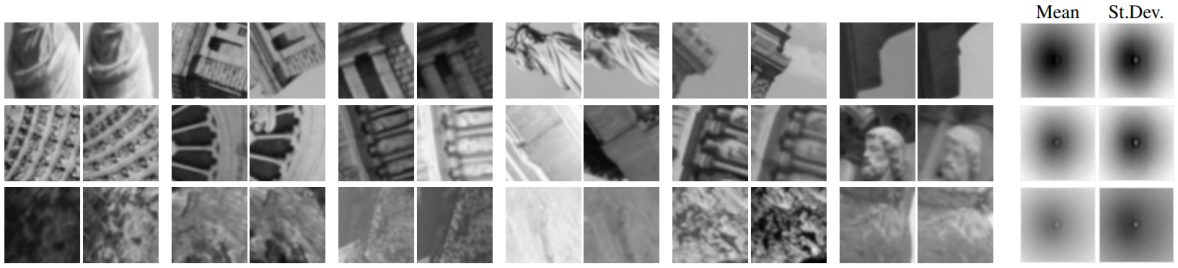


Figure 1.12: This figure shows pairs of images from the Multi-View Stereo (MVS) dataset Snavely et al. (2008); Goesele et al. (2007). Images in the top row come from the New-York Statue of Liberty, and are referred to as 'Liberty' (LY) in the dataset. In the middle row are images from the Notre-Dame de Paris cathedral, dubbed 'Notre Dame' (ND). Finally, in the bottom row, we can see images from the Yosemite National Park in the U.S. that are referred to as 'Yosemite' (YO). Each row is separated by pairs of images representing the same physical location. Figure taken from Simo-Serra et al. (2015).

A more complex approach is to learn the complete description pipeline, which includes both the detection and description of points of interest in an image. Yi et al. (2016) followed this approach by training three CNNs, which respectively: calculate a score map to crop the most interesting region, predict the crop orientation to rotate it accordingly, and describe the image crop. At training time, these three operations are carried out back-to-back to be differentiable in an end-to-end manner. The whole pipeline is trained using quadruplet of 128×128 pixels image patches taken at the location of keypoints calculated by a *Structure-*

from-Motion algorithm. The detector is used to find the most interesting keypoints in each of the 128×128 pixels image patches, and then a 64×64 crop is passed on to the rest of the pipeline. This alleviates the need to train the detector to find the most distinctive points in an image, and then at test time, it can be decoupled from the pipeline to be run on the whole image to find interesting keypoints. They tested their method on three different datasets and outperformed all the other approaches, among which there was Simonyan et al. (2014); Xufeng Han et al. (2015); Simo-Serra et al. (2015); Bay et al. (2006); Alcantarilla et al. (2012); Rublee et al. (2011); Lowe (2004).

In DeTone et al. (2018), they used complete images as input to learn an encoder function using a CNN. This encoder generates a reduced representation of the whole image. It is then sent to an interest point decoder and a descriptor decoder that outputs the keypoints and descriptions respectively, all in a single forward pass. Another pipeline was proposed by Rocco et al. (2018). Their approach begins by calculating the descriptions for every part of the image using a CNN. Then, instead of using keypoint locations to restrain the number of comparisons to the number of interesting points, they directly compare all the descriptors of two images at once. This results in a 4D space of feature matches that they process with a 4D CNN that outputs a 4D map of filtered matches. They assumed that with a 4D CNN they could take advantage of the neighborhood spatial information related to matches to better identify them.

For a good comparison between SIFT- and CNN-based descriptors, the work of Zheng et al. (2016b) offers a good overview of the performance. It also indicates a slight advantage for learned methods when applying fine-tuning. The need to use fine-tuning is leading us back to the problem mentioned earlier: learned descriptors can be biased by their dataset and may underperform on certain tasks. In Schonberger et al. (2017), multiple learned and hand-crafted descriptors are evaluated on different matching tasks as well as the more practical task of image-based reconstruction. They clearly show the high variance of learned descriptors across tasks and the equal or better performance of hand-crafted descriptors on image-based reconstruction. Thus, it is clear that despite their remarkable performance, learned approaches suffer from the generalization problem. However, next in section 1.8, we look into the work done to alleviate specifically this problem.

1.8 Domain Generalization

When tackling a problem with machine learning, data related to the problem at hand is essential to learn something. However, problems where there is no data directly available, may not be easily solvable using learning techniques. This is particularly true for deep learning approaches, as they typically require a significant amount of labeled data. When data is scarce, the way to solve such a problem is by using abundant data coming from similar distributions.

This is known as *Domain Generalization* and aims at using available data from k different problems defined as source domains $\mathcal{D}^s = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_k\}$ to learn invariant features that enable generalization to an unseen target domain $\mathcal{D}^t$. This can be applied to problems such as Visual Place Recognition (VPR), where it is unrealistic to gather images from all around the world with all possible types of variations (illumination, weather, season, etc.). Another example is when real data is hard to collect, but generating synthetic data is cheap and reliable. The same problem appears when trees offer too much variety, and thus collecting sufficient data from every possible type of bark is practically impossible. With the need to investigate *Domain Generalization*, datasets composed of several source domains have been shared, such as the PACS (Li et al., 2017b) dataset and the VLCS (Fang et al., 2013) dataset.

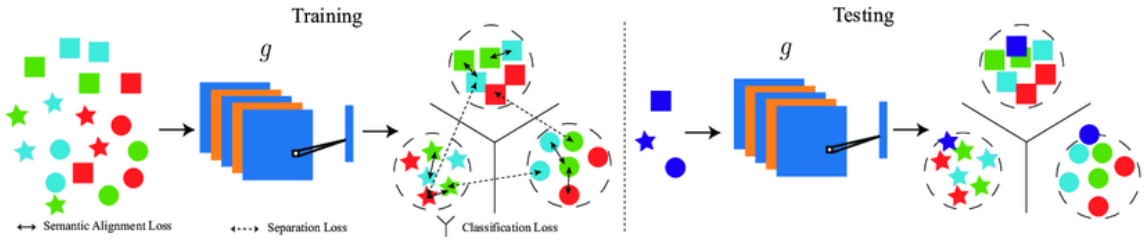


Figure 1.13: This figure was initially designed to show the effect of the different losses proposed in Motiian et al. (2017). However, here we intend to show how well it represents the goal of domain generalization. First, the distinction between the classes to predict is represented by the different shapes. Also, the difference in the domain distributions is expressed by colors. We can then consider the left side as training a network to classify three different shapes, despite the shape coming from different domains. Next, on the right side, we can see the goal of domain generalization, which is to have a classification network being able to classify examples at test time from an unknown domain (purple) in the three same categories with minimum error. This figure was taken from Motiian et al. (2017).

In *Domain Generalization*, all the domains are similar, but each domain possesses its own bias, causing a covariate shift between domains. This, in turn, leads to a reduction of performance when testing the learned model on an unseen domain. Despite this, it is possible to make a model employ the common features shared between domains while lessening the covariate shift effect. This goal is well illustrated in Figure 1.13, where we can see that the model g at test time produces similar representations for the target domain, while still showing a bias compared to the source domains. Common ways of solving domain generalization can range from learning domain-invariant classifiers (Khosla et al., 2012) to learning domain-invariant features (Muandet et al., 2013; Ganin et al., 2016). Also, in Radenovic et al. (2018), they tried to directly transform all source domains to a common representation, in order to learn a single representation for all domains.

1.9 Vision Applied to Bark/Wood Texture

Trees are an essential resource for the economy and the environment. But, with all of the species and variations of appearance, including between the trees of a sole species, recognizing them can be challenging. This motivated research studies on how best to utilize tree characteristics to determine their species and, in particular, bark, since it is easily accessible and present all year round. This led to the first attempts of recognizing tree bark using Local Binary Patterns (LBP) descriptors. This is because bark can be considered as a texture classification problem, for which the LBP descriptors have shown good performance at classifying. This technique and its variants have been used in Sulc and Matas (2013); Svab (2014) to described tree bark textures in images and then classify the bark description using a Support Vector Machine (SVM). Likewise, Huang et al. (2006) described bark images with LBP features, but tried to use a radial basis probabilistic network to carry out species classification instead. A good comparison of LBP descriptors applied to bark texture can be found in Boudra et al. (2015), where they evaluated the performance of such descriptors using instance retrieval based on the bark species. They show moderate results on the Trunk12 (Svab, 2014) dataset and the AFF (Fiel and Sablatnig, 2010) dataset.

Even if bark identification is considered a texture problem, other descriptors not designed with texture in mind have been studied in an attempt to recognize tree species from it. For example, Zheru Chi et al. (2003) used banks of Gabor filter to model texture as a collection of narrowband signals from which they extracted characteristic features of each bark species. Classification was then performed on these extracted features. Closer to our description method, Fiel and Sablatnig (2010) used a SIFT descriptor to process images of bark and calculate a Bag of Words (BoW) description that they classify using a SVM. Their work shows interesting results on the AFF bark dataset. They further evaluated their method for the classification of tree leaves on a small leaf dataset and look at the feasibility of classifying tree needles. Recently, Carpentier et al. (2018) approached tree bark recognition using CNNs, but due to the need for a large amount of data to train deep neural networks, they had to build their own dataset. This dataset, called BarkNet, contains 23,616 images of 23 different tree species. They compared the performance of pre-trained ResNet fine-tuned on bark images and demonstrated high performance, particularly when using a voting scheme based on multiple crops of the bark.

Texture can be found to be similar for the same family of material, thus suggesting that classification is possible between each family, such as maple bark versus pine bark or simply asphalt versus granite. However, texture being randomly generated in nature makes two surfaces of the same family of material similar, but still different. This randomness in texture makes it worthwhile to investigate the possibility of re-identifying a specific texture surface based on the uniqueness of this surface. For instance, one can find interesting research such as that of Zhang et al. (2017); Zhang and Rusinkiewicz (2018) that use ground textures such as

asphalt, wood floor, or other texture surfaces to enable robots to localize themselves. In Zhang et al. (2017), they focused on building a texture map of the floor using the standard SIFT pipeline (detector+descriptor). Once the floor was mapped, they would later take a picture of the ground using their fixed downward camera surrounded by LED that helps generate nearly-invariant images. They then described the image with the SIFT descriptor and compared the features found with the one in their map, using a voting scheme tailored for the randomness of ground texture. In their following work (Zhang and Rusinkiewicz, 2018), they improved by learning a texture feature detector. This detector is based on a CNN, trained on their previous dataset (Zhang et al., 2017). For training, they employed a special "ranking" loss combined with a "peakedness" loss designed to maximize the response map.

1.10 Evaluation Metrics

Image retrieval can be evaluated in multiple ways. However, not every metric gives a good understanding of the performance of any solution. Since the task discussed in this thesis is essentially a binary classification (relevant versus non-relevant) problem, we will review the metrics well-suited for it and their significance. This is important since using a metric not designed for our task could give misleading results. As an example, using accuracy to evaluate the performance of a classification model on a binary problem with a skew ratio of positive/negative examples could give excellent results as long as it labels everything as the majority class. For instance, having 5 positive examples against 1000 negative examples with a model labeling everything as negative would yield a 99.5% accuracy ($\frac{1000}{1005}$ correct predictions), while having mislabelled all positive examples.

Usually, a model predicting a class label will either directly output the predicted label or give a probability (or score) of belonging to each of the labels. In the task of image retrieval, we usually refer to the positive class as a *relevant match* between a query image and an unseen image. Conversely, the negative class will be referred to as a *non-relevant* match. To better adhere to the problem formulation used in this thesis, we will consider that the model gives a matching score for every image of a set compared to a query image, which indicates the relevance of these images to the query. This means that a higher score is related to a better probability of being a relevant image to the query. Therefore, by using a certain threshold, we can transform these scores into a label. This makes us consider the predictions of a model as two separate categories. If the predictions are considered as scores of being classified as relevant, we can use a metric that will estimate performance based on the order of these scores in their set. Otherwise, if we consider the predictions as relevant/non-relevant labels, we will use metrics estimating the performance based on the set of unordered labels. For another reference on the subject of information retrieval evaluation, the reader is referred to section 8 of Manning et al. (2008).

1.10.1 Unordered Set

Starting with the scores given to images by a model predicting relevance, we observe a set of scores as presented in the first row of our toy example of Figure 1.14, where the color blue indicates scores that should be considered relevant and the color red indicates the opposite. When looking for a model that will give an explicit label, we make use of a threshold that will separate the scores given into two distinctive groups that will each receive an explicit label of relevance and non-relevance. In the second row of Figure 1.14, the gray squares indicate possible thresholds that could be used to classify higher score (left) as relevant and lower score (right) as non-relevant. This transforms our set of scores into an unordered set of labels. However, as seen in the figure, a threshold may wrongly classify some elements of the set.

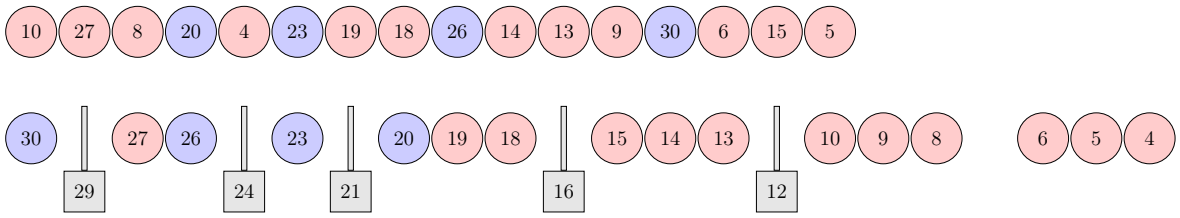


Figure 1.14: Here, we can see two rows depicting the same matching scores. Each number is a score, with blue circles indicating relevant matches, red circles are non-relevant ones, and gray squares represent possible threshold values. First row: an unordered set of matching scores calculated for an arbitrary query. Second row: the same score as row one but in decreasing order to visualize the separation by different thresholds.

		Ground Truth	
		True	False
Predicted	True	True Positive (TP)	False Positive (FP)
	False	False Negative (FN)	True Negative (TN)

Table 1.3: This figure presents a confusion matrix showing the name of the variable between predicted labels and ground-truth labels.

To more easily inspect the count of correctly and incorrectly classified images, we construct a confusion matrix, as shown in Table 1.3. This matrix is needed to count the number of images situated in each intersection of predicted and ground truth labels. These intersections are the correctly predicted relevant images (TP), the correctly predicted non-relevant images (TN), the incorrectly predicted relevant images (FN), and the incorrectly predicted non-relevant images (FP). These four values are essential to evaluate the performance of a prediction model, and multiple metrics may be derived from them, as we will see next.

Precision and Recall

The TP and TN results indicate the number of correct predictions made by our model. While the FP and the FN indicate how many errors our model has made. However, correctly evaluating our model is not that simple. When our test set is largely imbalanced, and we have

few relevant data to compare to the non-relevant data, we need to focus more on our total TP prediction. Also, depending on the application, making more FP errors may be preferable than FN errors, while in other cases, the opposite would be preferable. For a more in-depth explanation of the FP and FN error, see section 9 of Montgomery and Runger (2003). To obtain a more comprehensive overview of the results under these constraints, we commonly use the *Precision*, and *Recall* measures, defined as:

$$Precision = \frac{TP}{TP + FP}, \tag{1.7}$$

$$Recall = \frac{TP}{TP + FN}. \tag{1.8}$$

Equation 1.7 represents how much the model used to make predictions with a certain threshold is conducive of considering non-relevant matches as relevant. Conversely, Equation 1.8 represents the preponderance of the model to wrongly predict relevant matches as non-relevant. While each equation evaluates a different type of error, they are connected by an inversely-proportional relationship. In other words, if we move a threshold to improve one measure, the other will most likely suffer.

Unordered Set Precision and Recall Graph

Looking at *Precision* and *Recall* for a single arbitrary threshold is not meaningful. Thus, one of the common ways to use these two measures is to calculate them for multiple thresholds, spanning the possible scores, and then plot one against the other for each threshold. This results in a Precision Recall (PR) Curve that allows us to see the trade-off between *Precision* and *Recall* for each threshold. If a single threshold could correctly separate every score, it would produce both a *Precision* and *Recall* of 1, and this means that the best point in such a graph is the top right corner at (1,1). However, this is highly improbable except for trivial problems. One should, therefore, look for a point in the graph that represents a threshold where the important metric for their use case respects their need, without penalizing too much the other metric. Table 1.4 shows metrics calculated from the example in Figure 1.14, demonstrating how *Precision* and *Recall* are inversely correlated. To differentiate the PR graph using *Precision* and *Recall* calculated from an unordered set to the PR graph that we present in the ordered set section later, we refer to the former as a Unordered Set Precision Recall (US-PR) graph.

F1 Score

If one is searching for the optimal combination of *Precision* and *Recall* among all thresholds used and considering giving equal importance to both measures, it is a good idea to look at the F1 score. This metric is a single value obtained for a specific threshold that is equally influenced by both *Precision* and *Recall*, while preventing averaged results such as 0.5 for a precision of 0 with a recall of 1 (F1 would yield 0). This F1 score gives a good representation of the expected performance of a method when considering precision as much as recall. The exact calculation of the F1 score takes both measures equally into consideration. It is defined

here as:

$$F1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}. \tag{1.9}$$

Threshold	29		24		21		16		12	
Confusion matrix	1	0	2	1	3	1	4	3	4	6
	3	12	2	11	1	11	0	9	0	6
Precision	1.000		0.667		0.750		0.571		0.400	
Recall	0.250		0.500		0.750		1.000		1.000	
F1	0.400		0.571		0.750		0.727		0.571	
FPR	0.000		0.083		0.083		0.333		0.500	

Table 1.4: Confusion matrix and associated metric calculated for each threshold used in the example of Figure 1.14.

Receiver Operating Characteristic

Another common way to evaluate multiple thresholds is to plot *Recall* against the False Positive Rate (FPR), which is defined here:

$$FPR = \frac{FP}{FP + TN}. \tag{1.10}$$

This gives the Receiver Operating Characteristic (ROC) Curve, which represents the *Recall* as a function of the ratio between the number of relevant matches wrongly classified and everything predicted as non-relevant. The ROC Curve can be seen as the trade-off between the benefit (recall) and the cost (FPR) for every threshold. In such a graph, the diagonal line going from the bottom left to the top right represents random guesses. A line passing by the top left corner would be a perfect classifier (all benefit at no cost).

Area Under the Curve

From the ROC Curve, one can compute a single value, called the Area Under the Curve (AUC), that varies between 0 and 1. This value captures the probability that the model will score a randomly-relevant match higher than a randomly-non-relevant one. Since the AUC refers to the area under the ROC Curve, it should be calculated using an integral. However, since the ROC Curve is the plot of *Recall* against the FPR, we have a series of (x,y) points. Instead of using an integral, the AUC is calculated with the trapezoidal rule shown here:

$$AUC_i = \frac{1}{2}(Recall_i + Recall_{i+1})(FPR_{i+1} - FPR_i). \tag{1.11}$$

In this equation, the subscript i identified a point of the ROC Curve, hence AUC_i is the area between two points of the ROC Curve and repeating Equation 1.11 for $\{i_0, i_1, i_2, ..., i_{n-1}\}$ give an approximation of the AUC. Like the ROC diagonal that represents random guesses, a AUC value of 0.5 is considered uninformative. Conversely, an AUC of 1 is a perfect score. However, a value of 0 is also a perfect score because it can be reversed to 1 by inverting the class labeling. Finally, this metric must be seen as an aggregated representation of the performance since it is a summary of the result of every threshold.

1.10.2 Ordered Set

The other way to evaluate a model performance based on the query scores is to sort them in decreasing order. We took the previous example of Figure 1.14 and changed the thresholds for the ranking system in Figure 1.15. When using an ordered set, the assumption is that relevant matches are given higher scores than non-relevant matches. With this assumption and the fact that we have four relevant images for our example query, we then consider that the first four ranks of our ordered set should contain the relevant matches.

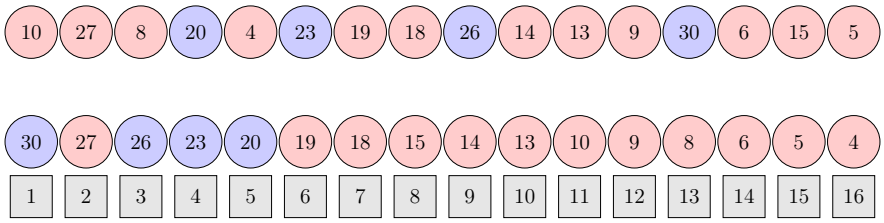


Figure 1.15: Here are two rows depicting the same matching scores. Blue circles are relevant matches, red circles are non-relevant ones, and each number in these circles is a score. The gray squares represent the rank of each score, after putting them in order. First row: an unordered set of matching scores calculated for an arbitrary query. Second row: the same score as row one but in decreasing order, to better visualize the rank of each score.

Precision@K and Recall@K

Just as in the unordered section, *Precision* and *Recall* can be calculated for an ordered set of scores. However, instead of using a threshold, here we need to select a rank K that will be used to separate the ordered scores in relevant and non-relevant classes. The scores ranging from the first rank to the K^{th} rank will be considered as relevant and will enable the calculation of the Precision at rank K (P@K) and Recall at rank K (R@K) as:

$$P@K = \frac{p(K)}{K}, \tag{1.12}$$

$$R@K = \frac{p(K)}{|I|}. \tag{1.13}$$

In both Equation 1.12 and 1.13, the function p returns the number of relevant images ranked between the first rank and the K^{th} rank (K included). For Equation 1.13, I is the set of relevant images and $|I|$ is its cardinality.

Precision Recall Graph

A PR graph can be drawn for ordered sets, just as for unordered sets, by using the P@K for each level of *Recall* available. For this, we calculate the P@K and the R@K at each rank where a relevant image can be found as in Table 1.5. Then, plotting them against each other produce a PR graph. With an ordered set, the PR is then calculated as:

$$PR = \{i \in I \mid (R@i_k, P@i_k)\}. \tag{1.14}$$

Here, $i \in I$ represents one image coming from the set of relevant images, and i_k is the rank where the image i can be found. This builds a set of tuples, with each tuple being (P@K,

R@K). The reason we do not calculate these metrics for every K is that the P@K is bound to decrease if the $K+1$ rank does not contain a relevant match. Instead, we calculate the P@K only when the R@K increases. So, a PR graph allows us to look at the averaged P@K over all queries obtained for each rank where a relevant image is located. To differentiate this PR graph from the US-PR graph, we refer to it as an Ordered Set Precision Recall (OS-PR) graph.

K	1	3	4	5
P@K	1.000	0.667	0.750	0.800
R@K	0.250	0.500	0.750	1.000
Average Precision (AP)	0.804			

Table 1.5: P@K and R@K calculated for each K where there is a relevant image in the example of Figure 1.15. This table shows each measure of an PR graph and the associated AP for our example.

Average Precision

The Average Precision (AP) is the mean of all P@K in an OS-PR graph for a single query, as shown in Table 1.5. Thus, when we make multiple queries in an experiment and average the AP over them, we use the abbreviation mean Average Precision (mAP). What the mAP shows is a single number representing the P@K of every relevant match, which is a summary of the OS-PR graph. Also, a mAP value of 1 corresponds to a graph with a straight line remaining at 1 for any value of R@K.

R-Precision

When K is equal to the number of relevant images (four in our example of Figure 1.15), it is called the R Precision (R-P). Using $K = |I|$ has the advantage that it can reach a value of 1 with a perfect classification and simultaneously give Precision@4 and Recall@4. We can see that this equality occurs in Table 1.6 when $K = 4$. What makes this score different from the mAP is that its evaluation is more arbitrary. For example, if the fourth-best relevant image reaches rank 5 as in the example of Figure 1.15, it will not be counted in the R-P, thus giving a value of 0.75 ($3/4 = 0.75$). While a Precision@5 of 0.8 would be added to the mAP value (for a final mAP of 0.804 as in Table 1.5), making the drop in performance less drastic.

K	1	2	4	5	10
P@K	1.000	0.500	0.750	0.800	0.400
R@K	0.250	0.250	0.750	1.000	1.000

Table 1.6: P@K and R@K calculated from the example in Figure 1.15, using arbitrary K to show the *Precision* and *Recall* equality when $K = |I| = 4$. Note you can also see the drop in *Precision* with an identical *Recall* when going from $K = 1$ to $K = 2$, which would give misleading results if used in a graph.

Precision@1

Even if Precision at rank 1 (P@1) uses an arbitrary rank, this metric can be meaningful. The P@1 checks if the best score per query is a relevant match. It is advantageous if an application needs only one match to confirm the re-identification. However, one must be careful of the number of relevant images used. This is because even if we obtain a high P@1, this does not mean that every relevant images can be retrieved reliably. A high P@1 only indicates the probability we have to always retrieve the more similar relevant image.

1.11 Conclusion

In this chapter, we reviewed the essential elements of image retrieval and deep learning. We first began by organizing the different ways image retrieval is used in different problems such as visual place recognition, face recognition, and person re-identification, among others. With the ability to describe images being essential to image retrieval, we then looked at image description using global descriptor techniques. Apart from global descriptors, we also discussed how hand-crafted local feature descriptors with their more invariant descriptions had become the standard method. However, because these local feature descriptors used computationally expensive algorithms, we also looked at how it is possible to summarize their numerous local descriptions to produce easily comparable Bag of Words (BoW) descriptions.

To understand the fundamentals of Deep Learning, we entered the subject of Convolutional Neural Network (CNN). Despite having a great capacity to represent high dimensional data in meaningful ways, training them to achieve this is still a hard challenge. Following this section, we thus focused our attention on metric learning. We understood how we could give deep neural networks an objective to learn that make unique and invariant representations. Hence, we saw that this combination of CNN and metric learning could produce great descriptors, but in this thesis, we focused on how to use them to create learned local feature descriptors. We explained how these later can achieve excellent performance, but need particular datasets for training and might suffer from the bias present in these datasets. To go more in-depth about this dataset bias problem in machine learning, we transitioned to the subject of domain generalization to explain how this bias problem can be mitigated.

Before reaching this conclusion, we discussed crucial information about the metric that can be used to evaluate image retrieval tasks and how they are all useful in different ways. Finally, before delving into chapter two, a quick review of the literature referenced in this chapter is available in our reference table just below.

1.12 Reference Summary

Author (Year)	CBIR	Object, Scene Reco.	Face Reco.	VPR	Person Re-Id	Global Desc.	Hand Crafted Desc.	CNN	Metric Learning	Learn Desc.	Domain Gen.	Tree Bark
Datta et al. (2008)	x											
Zhang and Ye (2009)	x											
Kaur and Banga (2013)	x											
Sural et al. (2002)	x					x						
Sinha and Kangarloo (2002)	x					x						
Neetu Sharma et al. (2011)	x					x						
Arandjelovic et al. (2018)	x			x					x			
Wan et al. (2014)	x	x		x					x			
Zheng et al. (2016b)	x	x		x	x					x		
Wang et al. (2017b)		x							x			
Sohn (2016)		x	x						x			
Yi et al. (2016)		x								x		
Rocco et al. (2018)		x		x						x		
Sivic and Zisserman (2003)		x										
Lowe (2004)		x					x					
Bay et al. (2006)		x					x					
Liu et al. (2017)			x						x			
Wang et al. (2017a)			x						x			
Wang et al. (2018a)			x						x			
Wang et al. (2018b)			x						x			
Chopra et al. (2005)			x						x			

Schroff et al. (2015)			x						x			
Ming et al. (2018)			x						x			
Sun et al. (2014a)			x						x			
Parkhi et al. (2015)			x						x			
Sun et al. (2016)			x						x			
Wang and Deng (2018)			x									
Zhang et al. (2014)			x									
Taigman et al. (2014)			x						x			
Sun et al. (2015)			x						x			
Taigman et al. (2015)			x						x			
Sun et al. (2014b)			x						x			
Yi et al. (2014)			x						x			
Guo et al. (2016)			x						x			
Klare et al. (2015)			x									
Huang et al. (2007)			x									
Learned-Miller (2014)			x									
Wolf et al. (2011)			x						x			
Rublee et al. (2011)		x					x					
Alcantarilla et al. (2012)		x					x					
Calonder et al. (2010)		x					x					
Lowry et al. (2016)				x								
Ramos et al. (2007)				x								
Jain et al. (2017)				x								
Zheng et al. (2017)				x	x				x			
Sunderhauf et al. (2015)				x					x			

Cummins and Newman (2008)				x								
Cummins and Newman (2009)				x								
Turcot and Lowe (2009)				x								
Li et al. (2010)				x						x		
Hermans et al. (2017)					x				x			
Gray et al. (2007)					x							
Arandjelovic (2012)		x					x					
Viswanathan (2011)							x					
Rosten and Drummond (2006)							x					
Lecun et al. (1998)								x				
Deng et al. (2009)								x				
Krizhevsky et al. (2012)								x				
Nair and Hinton (2010)								x				
He et al. (2016)								x				
Iandola et al. (2016)								x				
He and Sun (2015)								x				
Hadsell et al. (2006)									x			
Li et al. (2017a)									x			
Xufeng Han et al. (2015)									x	x		
Simo-Serra et al. (2015)									x	x		
DeTone et al. (2018)									x	x		
Brown et al. (2011)										x		
Paulin et al. (2015)										x		
Schonberger et al. (2017)										x		
Snavely et al. (2008)										x		

Goesele et al. (2007)										x		
Mairal et al. (2014)										x		
Simonyan et al. (2014)										x		
Trzcinski et al. (2012)										x		
Li et al. (2017b)											x	
Fang et al. (2013)											x	
Motiian et al. (2017)											x	
Khosla et al. (2012)											x	
Muandet et al. (2013)											x	
Radenovic et al. (2018)											x	
Ganin et al. (2016)											x	
Huang et al. (2006)												x
Sulc and Matas (2013)												x
Svab (2014)												x
Boudra et al. (2015)												x
Fiel and Sablatnig (2010)												x
Zheru Chi et al. (2003)												x
Carpentier et al. (2018)												x
Zhang et al. (2017)												x
Zhang and Rusinkiewicz (2018)												x

Table 1.7: Summary of the referenced articles in chapter 1.

Chapter 2

Bark Data and Method

Our project about bark surface re-identification consists of three main components. The first one is the original bark data, which we collected ourselves and named the Bark-Id dataset. The second component is also about data, but more specifically, about a second dataset composed of small images patch tuples representing the same physical locations that we named the Bark-Aligned dataset. Lastly, our third component refers to two descriptors, DeepBark and SqueezeBark, built using Bark-Aligned.

This chapter explains all the details that we have taken into account to obtain these three crucial components. It begins by pointing out the need for a bark dataset tailored for bark re-identification in section 2.1. We then explain all the details taken into account to collect the data of the Bark-Id dataset in section 2.2, followed by the description of Bark-Id in section 2.3. After this, we go through the crucial details of homography in section 2.4. The next section lays out the operations, such as homography and keypoints detection, among others, we needed to be able to apply to our Bark-Id dataset in section 2.5. These operations enabled us to transform the Bark-Id dataset into the Bark-Aligned dataset composed of pixel-aligned image patches, representing identical physical bark location as described in section 2.6.

With the first two components well explained, we then enter the discussion of how we built our deep learned descriptors based on our Bark-Aligned dataset in section 2.7. In addition to our two datasets and two descriptors, we still need to clarify how we made use of them. For this, we first discuss the existing techniques that use local feature descriptors to build visual signatures of images in section 2.8. We also define the three methods we used to compare two bark visual signatures in the context of bark re-identification in section 2.9. Finally, we summarize this chapter in the conclusion of section 2.10.

2.1 Bark Data

All the previously-mentioned methods in section 1.9 are all geared towards tree bark classification. To the best of our knowledge, no work has been done on the problem of bark instance retrieval using images. Consequently, there is no available dataset suited for our bark re-identification objective. Table 2.1 shows the specificity of the datasets used in bark classification works. Moreover, the majority of these datasets are private. In addition, Table 2.1 includes our dataset developed with a careful index annotations to allow the testing of methods to retrieve specific instance (surfaces) of bark images. From this table, one can see that no other datasets allow for pixel-alignment of images of the same bark surface. This is an important point because without such a dataset, we would not have been able to build our Bark-Aligned dataset, without which no deep learning descriptor could have been trained. Finally, to stimulate research in this area, we decided to make our new bark dataset public.

Reference	Number of classes	Number of images	Public	Instance Retrieval	Pixel Aligned
Zheru Chi et al. (2003)	8	200			
Huang et al. (2006)	17	300			
TRUNK12, Svab (2014)	12	393	✓		
Bressane et al. (2015)	5	540			
Blaanco et al. (2016)	23	920			
AFF, Fiel and Sablatnig (2010)	11	1183			
BarkNet, Carpentier et al. (2018)	23	23616	✓		
Ours	2	2400+750	✓	✓	✓

Table 2.1: This table is a comparative of the existing bark datasets based on their size, availability, and applications. All datasets contain tree bark images designed for species classification, except ours. Adapted from Carpentier et al. (2018).

2.2 Description of the Data Gathering Methodology

To collect a dataset enabling instance retrieval and pixel alignment between images of the same bark surface, we designed a proper methodology before beginning to collect bark images. To achieve our tree bark re-identification objective, the dataset we collected needed to allow three different operations on its images. The first one (1) is to associate every image taken with a specific bark surface, enabling a retrieval ground-truth of relevance and non-relevance between images. The second operation (2) was to allow each image to be cropped around a surface of interest. Moreover, besides merely cropping the exceeding information, we needed to specify the contour of the surface of interest to generate a complete correspondence between two images of a sole bark surface. The third operation (3) needed is the homography, which allows the calculation of the spatial transformations between the specified location in multiple images of the same bark surface.

Other than the three previously-mentioned operations, our dataset needed to display certain forms of variations. Without it, our dataset would not present sufficient challenges in terms of retrieval. But more importantly, it would not allow us to train a descriptor to become invariant to these variations. To select them, we took into account that tree bark is a highly-textured surface and then postulated that changes in lighting and viewpoints would drastically perturb the representation of bark in an image. Our methodology thus needed annotated images that can be cropped and spatially registered with one another, while ensuring that changes in viewpoints and lighting were present between images.

Allowing operation 1 was rather straightforward. Since images have to be relevant to each other, we always begin by selecting a distinct physical bark surface. This surface was then photographed multiple times with different lighting and viewpoint variations. Then, when we uploaded the photos to a computer from the camera, we prepared a folder for every set of images corresponding to a specific bark surface. Thus, all images in a folder were relevant to each other. However, even if this is a straightforward operation, it is essential to mention that a distinct physical bark surface is just a certain area of bark on a tree. This means that two bark surfaces with no overlapping area are still considered distinct even if they are part of the same tree. Thus, bark images relevant to each other come from the same physical bark surface area and are not relevant to any other images of a different bark surface located on the same tree.

For operations 2 and 3, a more thoughtful method was necessary. The minimal requirement was to surround a specific surface of bark with fixed markers that would stay visible despite any variations. Using simple thumbtacks of different colors pushed into the bark could have been sufficient, but would have made their precise registration in the image more difficult. For instance, it might have asked of a human to visually locate the center of the thumbtack heads in the image and note the pixel location, which could lead to human error while adding a great cost in annotation time. To avoid such problems, we relied on the mostly-vertical and straight shape of trees to design a rectangular wooden frame that could be attached to most tree trunks. This provided us with a solid support to attach visible fiducial markers around a standardized area. In Figure 2.1, we can see our custom-made wooden frame, which allows a surface of 757.5 cm^2 (rectangle of 50.5 cm by 15 cm) of tree bark to be made visible. This frame was fixed to trunks using elastic cords attached from one side of the frame to the other, with sufficient tension to avoid any movement of the frame while pictures were taken. This frame was employed for every bark surface of our dataset.

As shown in Figure 2.1, we used four ArUco (Ar) fiducial markers to identify the four corners of the rectangle, delimiting the bark surface region. These markers were generated using the OpenCV library available in Python. The advantage of these markers is that the same library that generated them also offers the algorithm to detect them in an image automatically. These markers are defined by four black borders forming a perfect square enclosing a specific

Figure 2.1: This figure shows a sample of zoomed red pine images from our dataset. The wooden frame used to surround every bark surface in our dataset is displayed with its four fiducial markers.

binary pattern in the form of a matrix. This makes it easy to find them using edge detection while being robust to error and uniquely identifiable by their respective pattern. However, the material used to print them was of great importance for two reasons. First, trees being outside, we needed our markers to be resistant to different weather conditions. For instance, when there was too much humidity in the air, simply using paper made all markers subject to curling, reducing their ability to be detected. It also affected the precision to estimate their position. The second reason appeared when we tried waterproof paper: it quickly became apparent that such glossy paper reflected too much light, which completely prevented markers from being detected. The solution we employed was to print our markers on fabric, that we stretched with clips to make them visible on the frame. The fabric was roughly as glossy as regular paper, and them being stretched prevented curling. With this wooden frame and markers, we fulfilled the needs of operation 2 and 3 by being able to surround any bark surface with a clearly-defined rectangle, while having four fixed points allowing to calculate spatial transformations between any pair of images.

Finally, we needed to ensure a good distribution of the desired visual variations across each set of images coming from a single bark surface. Using an LG Q6 cellphone camera with

a resolution of 4160 x 3120 pixels, it was relatively easy to freely move the camera around when taking photos to incorporate changes in viewpoint. For each lighting angle, we used four different viewpoints to add changes in scale, perspective, and rotation. These variations can be seen in Figure 2.2 and with a wider range of variations in Figure 2.4. Changes in lighting angles were the other variation we desired. We felt it was one of the most crucial ones to train and stress our approach properly. This is because, with its numerous ridges and troughs, the appearance of tree bark is highly affected by light direction.

Consequently, hand-crafted descriptors, which tend by design to be less invariant to changes in lighting than viewpoint, are more susceptible to underperform for this condition. Thus, to add significant changes in illumination angles, we collected our images at night with a single 550 lumen LED $Energizer^{TM}$ lamp as the main light source. As only one person collected the pictures, the lamp had to be attached to a tripod. The tripod was moved into one of the three selected angles to light the tree bark, while a person took photos with the hand-held camera. One of the illumination angles was in front ($\sim 0°$) of the bark surface, while the other two were on the left ($\sim 35°$) side and the right ($\sim 35°$) side of the bark surface. When looking at the bottom row of Figure 2.4, we can see twelve images of a single surface, after being pixel aligned. Note that the latter removes the majority of the effect of the viewpoint variation, leaving mostly the difference in illumination, for illustration purposes. It also serves to demonstrate that we can, indeed, register images with our fiducial markers.

2.3 Bark-Id Dataset

Before we could start collecting bark images, we had to take other considerations into account. An important question to answer was the impact of collecting data on multiple tree species. With the results of Carpentier et al. (2018), it was clear that neural networks can easily distinguish between tree species. To avoid biasing our tree bark re-identification results positively, we opted to collect bark images from only two tree species. Otherwise, our descriptors may learn to exploit species as critical information for re-identification, while a realistic deployment scenario could be a forest composed of primarily one tree species. Our focus will thus be on evaluating bark re-identification in experiments containing many bark surfaces, but from one or two different species, to make it more challenging. It will also allow us to test how much training on one species transfers on re-identifying another species.

This requirement on species meant that we had to be able to identify a tree's species correctly. To simplify this error-prone process, we opted to collect Red Pine (RP) (an evergreen) from a tree plantation within a clearly-defined area containing only this single species. This ensured that every image taken from this location was indeed a red pine. For the second species, we chose Elm (EL) (a deciduous tree), since it was present on the university campus in specific locations for which previous forestry reports provided the information. For each of these tree

species, we respectively collected a total of 100 bark surfaces, taken from 50 different trees. A sample of these two kinds of bark taken from our dataset can be seen in Figure 2.2. This meant that for each tree we visited, we attached our wooden frame on two different sides of the trunk, giving two distinct bark surfaces per tree. Then, we took 12 images per distinct surface. We manually verified them after being taken to avoid any issues that would prevent the correct detection of the markers in the image. We also made sure that we had taken 12 images with a good distribution of the variations mentioned above before moving our wooden frame.

Due to the lengthy process of collecting bark surface images, we could not gather enough bark images of either tree species to evaluate our system in a completely realistic scenario that tracks hundreds of thousands of trees. Instead, we decided to focus on augmenting the number of true negative (non-relevant) examples to emulate such conditions. This was possible by adding a faster data collection approach to new bark images for *strictly non-relevant* examples, by forgoing the need to be able to register them pixel-wise. This way, we did not need to attach our wooden frame, speeding up the data gathering process. We thus collected more EL bark pictures with this method because they were physically closer to us, and there were a lot of EL trees that we still had not seen. To keep these new images close to our original appearance distribution, we also took them at night, with three different illumination angles, but with limited changes in point of view. We collected 30 images per tree with some physical overlap, spread nearly uniformly around the trunk. This gave us a total of 750 manually-cropped non-relevant images for any EL query taken at a similar scale of our other images. Moreover, we took these new images with limited changes in viewpoints, because it is easier to artificially alter images to add changes in rotation, perspective or scale than trying to simulate changes in lighting angle. This way, we sped up the data collection process, while still being able to later use data augmentation techniques to get closer to the distribution of appearance variations present in our 200 indexed bark surfaces.

Figure 2.2: Images from our database of Red Pine (RP) and Elm (EL). For each species, we can see three images of the same bark surface, but for different illuminations and camera angles. In each image, there are four fiduciary markers on a custom-made wooden frame, used for pixel-wise registration.

2.4 Homography

To better understand the explanations of section 2.5 coming next, we briefly introduce the concept of homography. This is a well-known concept coming from projective geometry, which became a useful tool for computer vision. This is because computer vision deals with digital images (or videos) that are 2D representations of the 3D environment, as seen in Figure 2.3. This means that an image is a plane in 3D space intersecting straight lines (light rays) going from the point of origin to infinity. When two images represent a common planar surface of the environment, they become related by a homography. This homography is a 3×3 matrix H defining the transformation of a planar surface representation of one image to another.

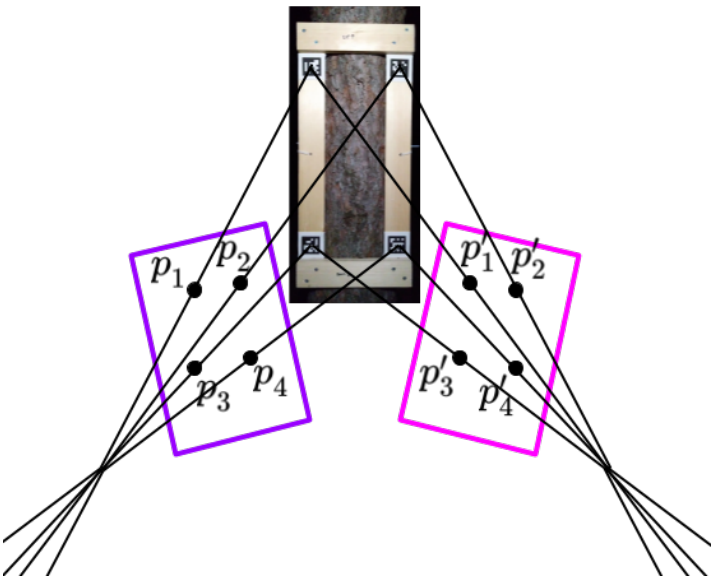


Figure 2.3: This figure shows the spatial relationship between two images of the same bark surface. Considering the bark image as the real physical model in nature, the rectangles containing the points p_k and p'_k represent two different images of the same bark surface. The index k shows the association of each p_k, p'_k with a fiducial marker. These associations come from the light rays bouncing on the bark surface and going through the lens of the camera. The homography is thus the relation between the set of points p_k and p'_k, based on the geometry of straight lines intersecting on common locations.

To calculate this homography, one first needs to select at least four points in each of the two images. These four points identified by the index k must represent the same location on the planar surface in the real 3D space. In other words, in each image, four points defined as $p_k = (x, y)$ will be selected and will have different (x, y) coordinates while representing the location of a unique point in 3D space. This results in a one-to-one correspondence between an image I and another image I', as in Figure 2.3. Once you have these four points correspondence, one can minimize the following equation, to identify the components h_{ij} of

the homography H:

$$\sum_k \left(x'_k - \frac{h_{11}x_k + h_{12}y_k + h_{13}}{h_{31}x_k + h_{32}y_k + h_{33}} \right)^2 + \left(y'_k - \frac{h_{21}x_k + h_{22}y_k + h_{23}}{h_{31}x_k + h_{32}y_k + h_{33}} \right)^2. \tag{2.1}$$

Note that this is exactly the equation used to calculate the homography H during our dataset processing, as specified by the `findHomography` function available in OpenCV. Thus, minimizing Equation 2.1 gives the best values of h_{11} through h_{33} allowing the selection of a single point (x, y) of a starting image I to calculate a destination point (x', y') in another image I':

$$Homogeneous\ p' = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}. \tag{2.2}$$

Lastly, it is crucial to understand that this homography matrix H represents a transformation in 3D space. Consequently, it has to be a 3×3 matrix. However, the coordinates of the images are in two dimensions and cannot be directly transformed. This is why each coordinate from the image I must be represented as a homogeneous coordinate. This is accomplished by adding an extra dimension with a value of 1. This way, we can obtain the homogeneous coordinate representing the point in image I' as visible in the right-hand side of the Equation 2.2. Then finally, to obtain the 2D image coordinate of I' from the homogeneous coordinate, we use the third dimension (z') to scale (x', y') to their correct position, as shown here:

$$p' = (x'/z', y'/z'). \tag{2.3}$$

2.5 Transformation Pipeline

Before applying standard keypoint detection techniques, we had to ensure that images only showed bark to fairly evaluate the different descriptors (hand-crafted vs. data-driven) performance. Also, even when keypoints have been found solely on bark, they still need to be filtered to avoid problems such as extra information given by image borders. Thus, going from original bark images to a clean set of keypoints is a multiple steps process. Luckily, we fully automated this preprocessing. It involved registration between images, image adjustment before keypoints detection, and keypoints filtering.

The first step, image registration, made use of the four visible ArUco (Ar) markers attached to our custom wooden frame. Recall that this frame is entirely visible in every indexed images of our dataset. It was done using the `detectMarkers` function available in the ArUco module of OpenCV. This function returns the four corner positions of each marker with its associated identifier based on its binary matrix. To have a more standardized and straightforward information, we reduced the four corner positions of each marker to their mean position. The original delimitation of our bark surface was then four (x, y) positions with their identifier.

These marker identifiers were needed to make the data association between images, and thus calculate a homography between these two sets of four points, as described in Equation 2.2. However, despite the availability of an Ar library in OpenCV and the robustness of such markers, these markers sometimes exhibited enough specular reflection in certain images so that they could not be easily detected. In these images, we could still clearly see the marker that was undetected. After performing some experiments, we found simple image transformations that would make the marker detectable. The best performing modifications were to either subtract a certain value from every pixel (65 or 100 worked best), use contrast normalization, or to mix the original image with a copy that was passed through an edge sharpener. With this, the Ar markers in every image present in our dataset could be reliably found.

With no assumption on the capacity of hand-crafted and learned descriptors to make use of the bark patterns, we introduced an input image resizing operation as the second step. The idea was that some patterns in a bark image may be more informative than others, but could also be seen only in larger image patches. By allowing an image to be reduced, we let every descriptor decide the optimal downsizing factor that would enable informative patterns to fit in their respective fixed input size. However, this resizing forced us to adjust the markers' position to ensure the proper correspondence for the homography operation. We could have avoided this adjustment by resizing the images before the Ar detection: but the reduced size would have been harmful to the Ar markers detection.

Since all four markers are fixed on the wooden frame, cropping from their original position would leave a large chunk of the frame and the background in the cropped images. We solve this in the third step of preprocessing by moving the four markers' position by a certain margin, towards the frame center. Due to every image having a different rotation, scale, and perspective, adding the margin directly on the image would have affected it in largely different ways. The way around this problem was to adjust the margin specifically for each image. The automatic process to do this involved setting a reference frame R as simply being a rectangle described by four corners. Then, we calculated the spatial transformation going from the frame R to the image I_i through homography H^i. This gave us the means to find four points respecting a fixed margin inside the frame R, which we then transposed in every image using their respective H^i. Adjusting the margin for every bark image provided us with four new marker positions surrounding bark pixels only. After that, the fourth step was to crop the image around these four points.

Because of the rotation and perspective changes, the cropping could still leave artifacts in the image, as seen in the second row of Figure 2.4. To remove this, one could crop inside the four marker locations. However, this would leave almost no pixel left due to the rotation of some images. Moreover, rotating the images to align them with the top of the image would defeat the purpose of training and testing the descriptors' invariance to rotation. Instead, the fifth step was to color every pixel outside of the borders formed by the four marker rectangle in a

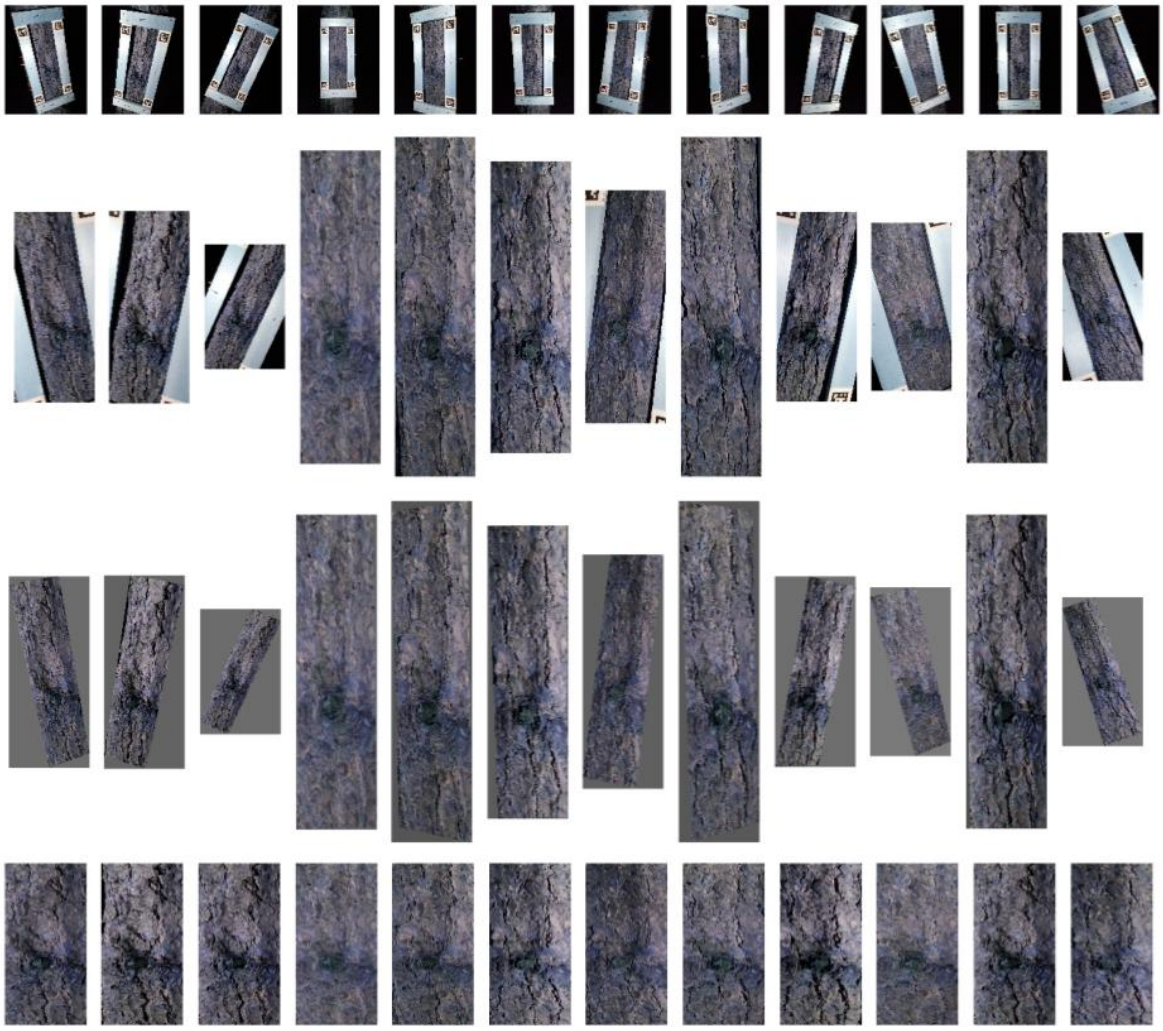


Figure 2.4: Examples of the preprocessing conducted on bark images. First row: original images resized by a chosen factor. Second row: rectangular crops around the wood frame. Third row: removal of any artifact remaining in the crops, using visual marker positions. Fourth row: examples of every image after reprojection onto the reference frame R using homography H^r (from image I_i to R).

unique color. This removed all changes in gradient not inside the specified bark surface and thus prevented the detection of any keypoint on non-bark pixels.

2.6 Bark-Aligned Dataset

Since our descriptors require a dataset composed of 64×64 pixel image patches to be trained with metric learning, we use our transformation pipeline of section 2.5 to create the Bark-Aligned dataset. On top of being properly indexed per bark surface, these patches must also be centered around the exact same physical location. Because patches cannot just be

extracted at random locations but rather at locations of interest, these patches are centered around keypoint locations. Consequently, we detected a maximum number of keypoints for each bark image using the SIFT keypoint detector to have a sufficiently large dataset. It is also essential to respect the fact that the patches' appearance distributions must be similar at training and testing times. Hence, we must use the same detector at both test and training time.

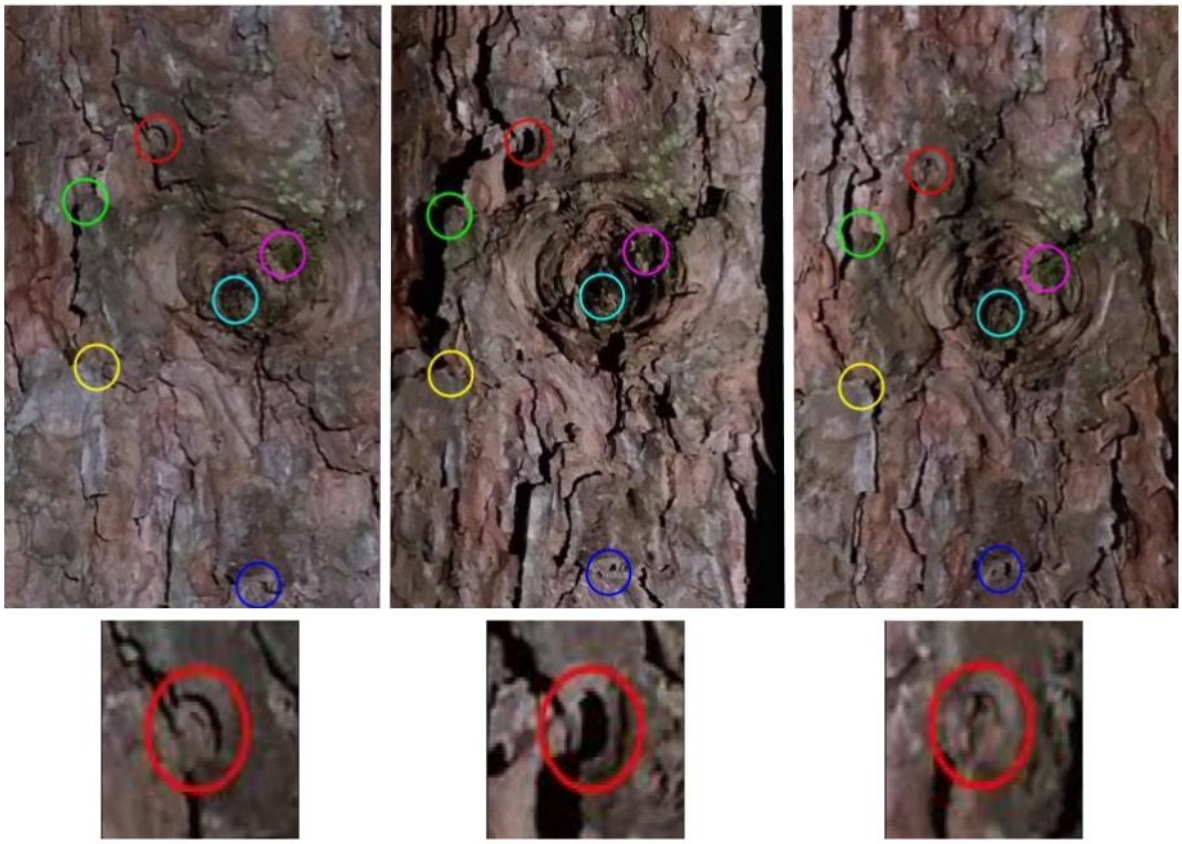


Figure 2.5: Top row: pictures of the same bark surface with strong changes in illumination. Each circle color is a distinct keypoint. Bottom row: close up of the red keypoints from their respective images. This highlights the importance of a descriptor to be as immune as possible to such illumination changes.

However, the input resolution of our descriptor network being 64×64, using a keypoint as its center, thus implies a 32 pixels radius around it. This becomes problematic when a keypoint is detected on the edge of the bark present in an image, because the patch taken for this keypoint will show border artifacts. To fix this problem, we removed every keypoint found at less than 50 pixels of the bark border. We chose 50 pixels since the corner of a square is at a distance from its center equal to the hypotenuse of the right triangle formed by two halves of its side ($\sqrt{32^2 + 32^2} = 45.25$). We also added a supplementary margin of 4.75 pixels ($50 - 45.25 = 4.75$). After verification, there was still some amount of pixels artifact due to

some extreme rotations and perspectives, although this had become insignificant.

After each image had been preprocessed to remove the excess of information (background, frame, shadow, etc.), we performed registration between every image of a bark surface with our reference frame R via homography H^r. We used the estimated positions of the fiducial markers on the frame surrounding the bark surface after preprocessing to estimate these transformations. Then for each bark image of the same surface, we projected all of their keypoints to the reference frame R via the homography H^r. We filtered all of the keypoints in R to require a minimum of 32 pixels between them to minimize overlap. This resulted in around 800-1000 distinct keypoints in R.

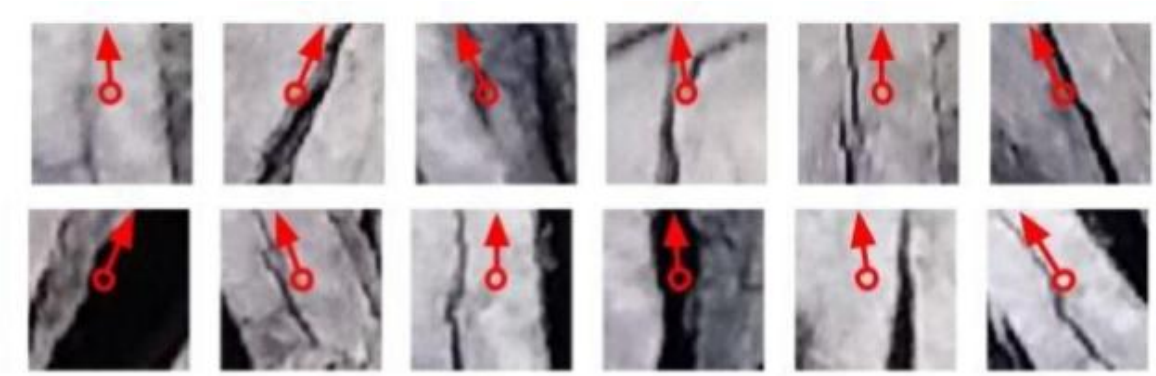


Figure 2.6: Actual example of 64×64 image patches, for a keypoint, projected on all 12 images of the same bark surface. Red arrows indicate the orientation of the original bark images.

For each of these keypoints, we then found the 12 image patches (one per image, see section 2.3) using a homography H^i that gives the transformation from the reference frame R to a specific bark image I_i. This resulted in a collection of 64×64 image patches centered around the exact same physical location on the bark, but with changes in illumination and point of view (rotation, scaling, and perspective). Figure 2.5 shows three images of a unique bark surface, with the manual correspondence between keypoints. Figure 2.6 shows 12 examples of a keypoint extracted according to our algorithm used to create the Bark-Aligned dataset.

2.7 DeepBark and SqueezeBark

To perform the description of image patches, we implemented two different architectures with Pytorch 0.4.1. The first one, `DeepBark`, is based on ResNet-18 developed by He et al. (2016) and pre-trained on ImageNet. We removed the average pooling and the fully-connected layers and replaced them with one fully-connected layer but without any activation function. The full architecture of our `DeepBark` network is described in Table A.1. The second one, `SqueezeBark`, is a smaller network based on SqueezeNet 1.1, presented in the work of Iandola et al. (2016) and also pre-trained on ImageNet. We again removed the average pooling and the fully-connected

layers. We replaced them with a max-pooling layer (to reduce the feature map) and a fully-connected layer (no activation function). The full architecture of our `SqueezeBark` network is described in Table A.2. In both cases, the networks compute a 128-dimensional vector, fed to an l_2 normalization layer. Removing our last fully-connected layer and calculating the number of parameters for the remaining convolutional layers, `DeepBark` is then composed of a total of 10,994,880 parameters and `SqueezeBark` includes 719,552 parameters. Our intention here is to be able to compare a number of network representation powers on the descriptor quality. These networks (`DeepBark` and `SqueezeBark`) were trained with the N-pair-mc loss from Sohn (2016). But our implementation varied slightly, since we did not use $l2$ regularization to avoid degeneracy. Instead, we $l2$-normalized the descriptor vectors v to keep it in a hypersphere as in Schroff et al. (2015).

Our Bark-Aligned dataset, described in section 2.6, is composed of 64×64 patches that we divided into two sets with around 70,000 distinct keypoints for the training set, and 17,000 for the validation set, except for the saturation experiment (section 3.3) where the goal was to vary the training set size. Using 12 patches by keypoint for training and two for validation, this totals 874,000 64×64 bark image patches. At each iteration during training, we only used a pair of examples for every keypoint in the training set. However, to ensure diversity in the (N+1)-tuplet which we used to train with the N-pair-mc loss, we shuffle the order of the keypoints in the training set after each iteration to compare together patches of different keypoints. Since we have 12 relevant patches per keypoint, we also randomly select the pair of examples for each keypoint to ensure an equal probability for every patch to be seen together with every other relevant patch. We added online data augmentation in the form of color, luminosity, and blurriness jitter. Each input patch image was normalized between $(-1, 1)$ by subtracting 127.5 and then dividing by 128. We used the Adam optimizer by Kingma and Ba (2015), starting with a learning rate of $1e^{-4}$ and reducing it by a factor of 0.5 each time the validation plateaued for 20 iterations.

We built the validation set by finding all of the keypoints available in the bark images set aside for validation and randomly selected two patches from the 12 available for each distinct keypoint. This gave us a fixed validation set, where every patch had a corresponding one. This way, during training, we validated our model by selecting 50 keypoints with their two examples at the time and performed a retrieval test to calculate the P@1. The final validation score was the average of every P@1 calculated for every batch of 50 keypoints. After training, we selected the model with the highest validation score. Training was stopped with either early stopping when the validation stagnated for 40 iterations, or when a maximum number of iterations was reached.

2.8 Bark Visual Signature

As explained in chapter 1, numerous descriptor calculation methods have been devised to extract the meaningful content of an image and make a useful description of it. This is necessary to allow an efficient comparison between images, which are too high-dimensional. To tackle our bark re-identification problem, we defined a clear process to describe bark images using local feature descriptors. More specifically, we perform the bark image search via visual signatures, defined as $s_i = (K_i, V_i, b_i)$. These signatures are extracted for each image (database and query I_q), as depicted in Figure 2.7. For this, we mostly follow the approach used in Sivic and Zisserman (2003), summarized below.

First, a keypoint detector selects a collection K_i of keypoints from an image. For each of these keypoints $k \in K_i$, we extract a description v of dimension 128, yielding a list of descriptions V_i. These descriptions can be from standard descriptors, such as SIFT or SURF, or our novel descriptors, which we introduced in section 2.7. The remaining component of an image signature s_i is a BoW representation $b_i \in \mathbb{R}^{1000}$, calculated from the list of descriptions V_i. For the latter, we employ the standard *TF-IDF* technique. In Sivic and Zisserman (2003), the comparison between two BoWs is made using the cosine distance. Instead, we have l_2-normalized once every BoW as a pre-processing step and use the l_2^2 distance to compare them at test time. This way, our distance ranking is equivalent to the pure cosine distance, but without using a dot product.

2.9 Signatures Matching

A typical bark re-identification scenario involves a database of image signatures S and an unseen query image I_q. The search for relevant images to I_q begins by describing the query image into a query signature s_q. A single comparison of two signatures produces a score g, which represents the similarity between them. So, the signature s_q must be compared with every signature of the database S to produce a set of scores $G = \{s_q, s_i \in S \mid compare(s_q, s_i)\}$. Hence, the *compare* function is an important piece of the retrieval process. In our experiments of chapter 3, this function is either a BoW comparison or the matching of descriptions v. After the use of *compare* to produce the set G, the retrieval of the relevant images to I_q is based on the best signature matching scores g.

For the BoW comparison, we simply used the distance between two BoWs $||b_q - b_i||_2^2$ as our score g. The other way to calculate a score between s_q and s_i is the matching of descriptions v and begins by taking the l_2^2 distance between every description of V_q and V_i, in order to obtain a collection M of putative matching pairs of descriptions, $m \in M = (v \in V_q, v \in V_i)$ with $|M| = |V_q|$. The equality $|M| = |V_q|$ appears because M is the result of a non-surjective mapping between each $v \in V_q$ to a $v \in V_i$ and is based on the smallest l_2^2 distance, meaning a single description of V_i can be the closest description to multiple $v \in V_q$.

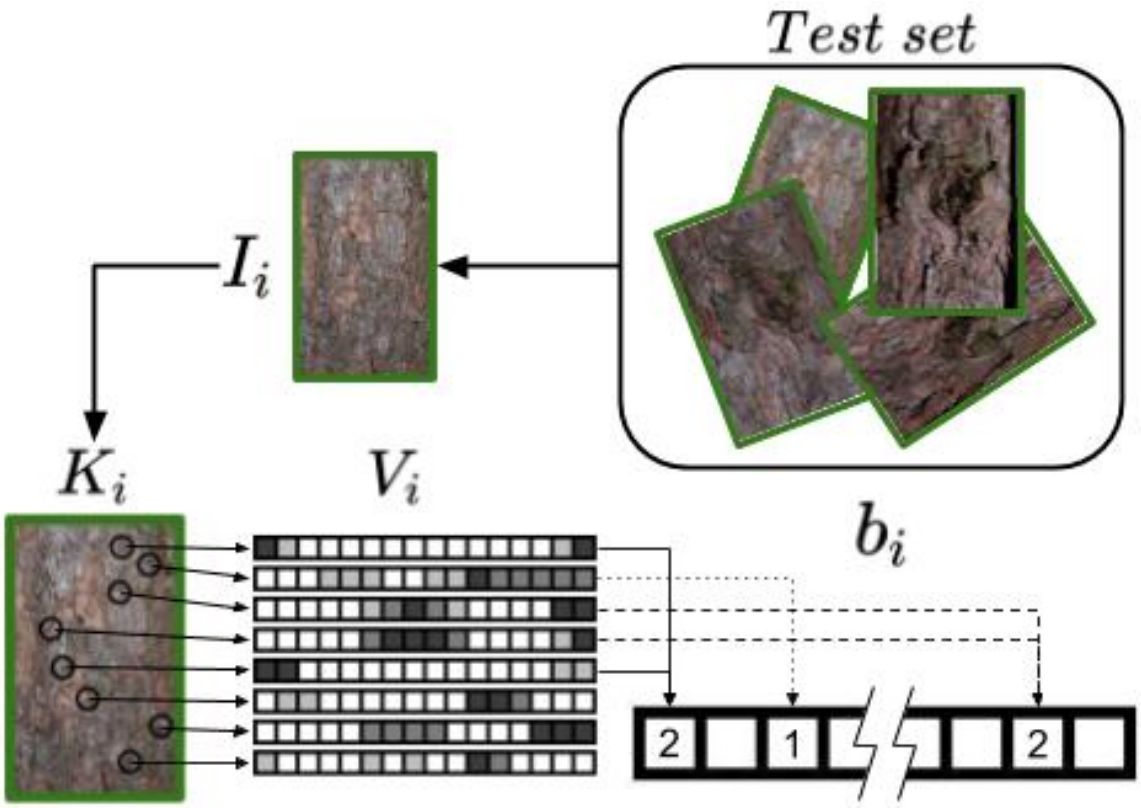


Figure 2.7: Illustration of the signature $s_i = (K_i, V_i, b_i)$ extraction pipeline, for a single image I_i. First, the keypoints K_i are detected. Then, for each keypoint k, a descriptor v is computed, creating the list V_i. Finally, a Bag of Words (BoW) representation b_i of V_i is computed from the quantization of all descriptions v via a visual vocabulary, resulting in a global signature s_i of image I_i.

However, building the set M is not enough to obtain a score g. Hence this method needs to filter out potential false matches in M by adding an extra constraint. In this thesis, we explored two such constraints, that we refer to as scoring methods. The first one is the *Lowe Ratio (LR) test* introduced in Lowe (2004). The second one is a Geometric Verification (GV), which is a simple neighboring consistency check. The GV starts by taking a match $m = (v_x, v_y)$, then retrieving the keypoints k_x and k_y associated with each description v of the match. Following this, we find the α nearest neighbors of each of the keypoints k_x and k_y in their respective images. Finally, the match is accepted if *at least* ρ % of the α neighbors of keypoint k_x have a match $m \in M$ with the α neighbors of keypoint k_y. An example of the GV used to evaluate one match is illustrated in Figure 2.8, with $\alpha = 5$ and $\rho = 0.6$, the match shown would be an accepted match. The number of matches left after filtering with either the LR or the GV scoring methods, is then considered as the matching score g between two bark images.

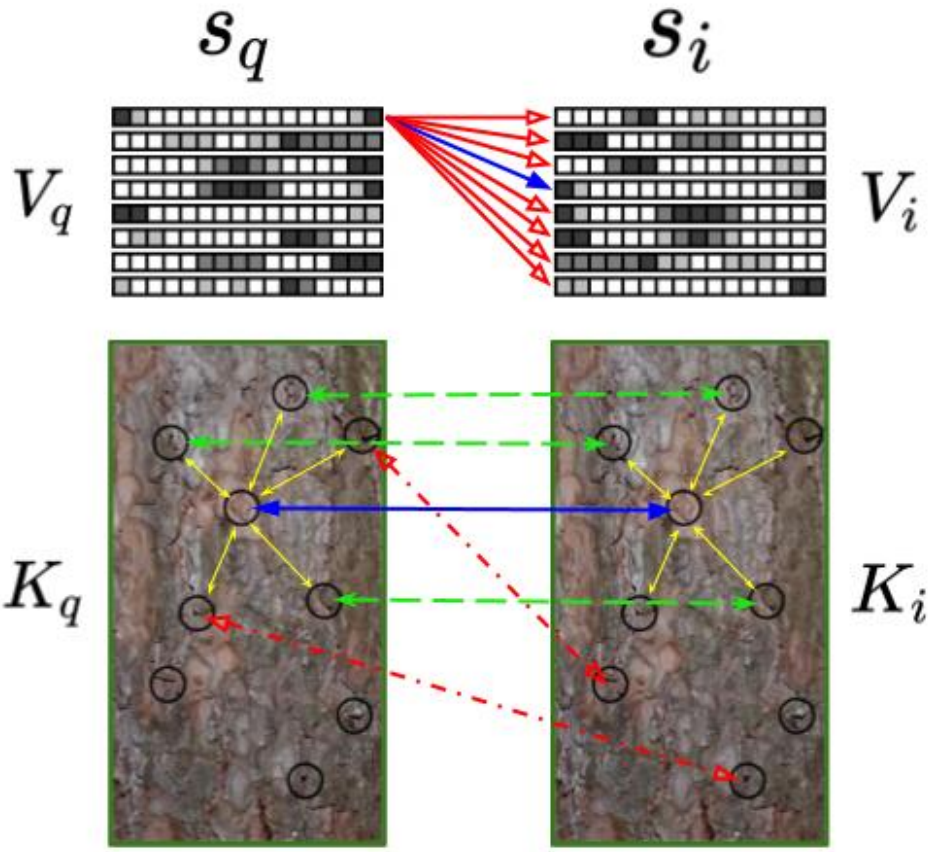


Figure 2.8: First row: example of a match m between a pair of descriptions. Second row: the GV of the match m. Blue arrows show the evaluated match $m \in M$ and green arrows show neighbors matches member of M, while red arrows indicate incorrect matches. With $\alpha = 5$ and $\rho = 0.6$, the match m would be accepted.

2.10 Conclusion

In this chapter, we began by presenting the existing bark dataset available and quickly showed how it was necessary to build our dataset to enable tree bark re-identification. Then, to create this dataset, we discussed the methodology we used to gather the data while incorporating the necessary elements to make this data suited for our re-identification task. We followed this with the composition of the new Bark-Id dataset that was produced using our data gathering methodology. We then had to briefly explain the concept of homography to prepare ourselves for the explanations on our transformation pipeline. This pipeline had two purposes. For one, it was essential to crop and describe images for future comparison. Still, it was also crucial to enable the calculation of the spatial transformation between any images using fiducial markers placed on our wooden frame and homography. By combining our Bark-Id dataset and our transformation pipeline, we built the Bark-Aligned dataset that was vital to allow the training of learned descriptors with bark data. Apart from the dataset for training, we

also reviewed the many details that composed the training procedure we used to build both our learned descriptors `SqueezeBark` and `DeepBark`. Aside from datasets and descriptors, we further needed a methodology to compare our bark images. For this, we explained how we built global visual signatures composed of the keypoints, and descriptions produced with local descriptors and a Bag of Words (BoW) representation. Finally, the last operation we needed to perform was to compare two global visual signatures together, for which we implemented different methods such as the BoW distance or the filtering of a set of matching descriptions M using either the Lowe Ratio (LR) or the Geometric Verification (GV).

Chapter 3

Bark Re-identification Experiments

The problem we are addressing is an instance of re-identification. Given an existing database of bark images and a query image I_q, our goal is to find all images in the database that correspond to the *same physical surface* to re-identify the tree, as seen in Figure 3.1. For instance, if the image compared to I_q is from the same tree *but*, a different bark area, then it is not a valid match, and the tree is not re-identified. For all experiments except in section 3.8, we assume that I_q has a meaningful match in our database, i.e., we are not solving an open-set problem. For more information on novel locations detection, we refer the reader to the work of Cummins and Newman (2008). It is important to note that even if our approach enables trees re-identification, in this chapter, we focused on reporting bark surface re-identification results. The evaluation of tree re-identification is largely influenced by the number of physical bark surfaces used for retrieval, which vary with each specific retrieval scenario and is thus left to future research.

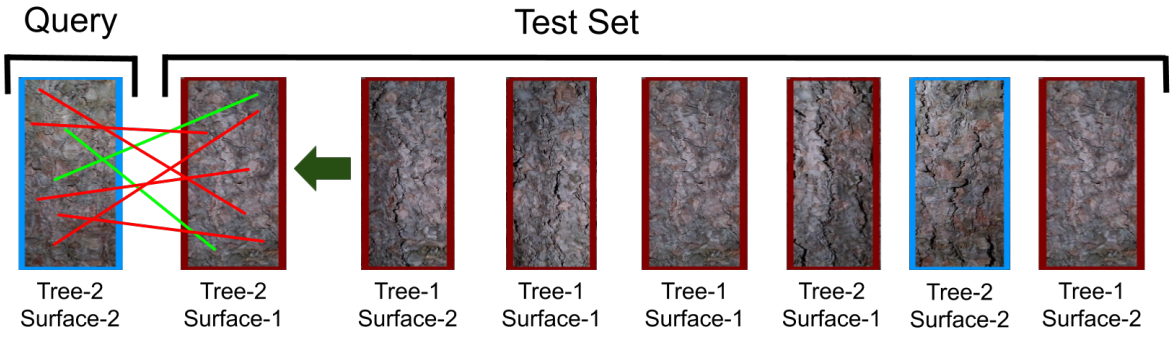


Figure 3.1: Here is a simplified example of bark images from two different trees, on which two different surfaces have been photographed. With two images per surface, this gives us eight images. Our goal is to take one bark image and retrieve the other bark image corresponding to the exact same surface. This way, a previously-seen tree could be recognized using a bark image as a signature.

To make our results unambiguous, we begin this chapter by laying out essential details about

our evaluation methodology in section 3.1. With the numerous hyper-parameters affecting retrieval results, we conducted a grid search of the three most critical hyper-parameters in our opinion and reported the results in section 3.2. Another influential factor, but specific to learned descriptors, is the dataset size used to train these descriptors. Thus, we evaluated our best performing learned descriptor on different training set sizes in section 3.3. With our approach using recent advances in deep learning, it was necessary to compare the performance of the well established hand-crafted descriptors with the newer learned descriptors, which is why we made the experiment of section 3.4. A significant problem of deep learning is the generalization of performance across data with different biases. With the diversity of bias in tree bark, we evaluated the generalization of our descriptors performance between tree species in section 3.5. One of the challenges about our data gathering methodology was the slowness of the data collection. This led to a dataset with less bark images then what more real-world scenarios would have. To mitigate this, we quickly collected unseen negative examples and evaluated the performance of our approach against a larger test set in section 3.6. Because matching the whole sets of descriptions V_q and V_i is time-consuming, we investigated the trade-off between time and performance in section 3.7. To tackle the problem of detecting novel locations, we inspected the score distributions of our approach and showed how a simple threshold could be used for this task in section 3.8. Finally, we resume our results and state the main message of this chapter in the conclusion of section 3.9.

3.1 Evaluation Methodology

Beside `DeepBark` and `SqueezeBark`, we also analysed hand-crafted descriptors, namely SIFT and SURF. We also included `DeepDesc`, a learned descriptor originally trained on the multi-view stereo dataset Brown et al. (2011). We also used our own version of `DeepDesc`, renamed `DeepDescBark`, which we trained on our bark dataset following our training procedure. All descriptors used the SIFT keypoint detector, except for SURF that used its own detector. For all experiments, we used a ratio of 0.8 for the LR test, and set $\alpha = 15$ and $\rho = 0.33$ for the GV filter. As we will see later, these parameters offered good performance, and we did not try to adjust any of them to improve the results further.

Each visual vocabulary *voc* was computed from the training images of each respective experiments while being clustered using the k-mean algorithm. Also, to the best of our knowledge, we did not see previous work using a learned descriptor with the technique of the BoW. Consequently, we do not know the effect of calculating the *voc* using the training set on which we trained our descriptor, instead of having two distinct datasets for these two tasks.

For every experiment (except in section 3.2), we take all indexed bark surface images as a query and search for their 11 respective relevant images, giving, for example, 1200 queries for a single experiment. This means that our unordered set of scores for an experiment is the

score of every query put together. However, such a set of scores comes with a consequence when applying a threshold on it. This is because the calculated score of any query is compared to a threshold without *relativizing* between queries. This problem can be seen in Figure 3.2. Thus, predicting relevance based on a threshold in our context may not give the best retrieval performance compare to predicting relevance on a query basis using a ranked method. Also, the F1 scores we report are always the highest value found among all evaluated thresholds.

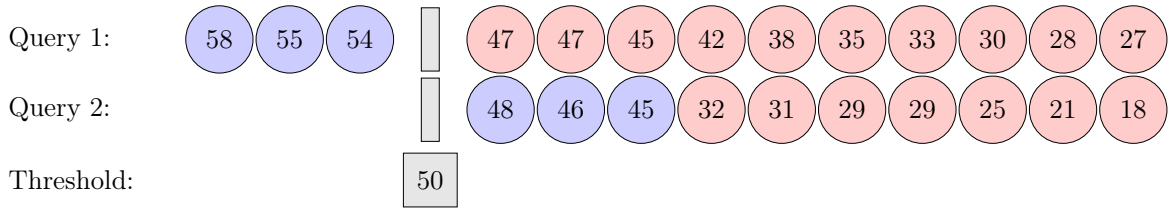


Figure 3.2: Example of the lack of relativization when using a threshold. Each circle node is the score of a bark image compare to a query image. Blue circles are the ground truth relevant scores, and red ones are the non-relevant ground truth scores. The grey square is the threshold used to predict the score relevance. The scores on the left are predicted as relevant and on the right as the opposite. We then clearly see that the threshold correctly predict the relevance for only the query 1, despite the query 2 having a score distribution easily separable.

As opposed to the unordered set method, the ordered set method calculates a metric for each query. This is because the metric needs to be calculated on the ranked score of a single query, which brings the advantage of solving the problem shown in Figure 3.2. However, the mean of 1200 queries, as in most of our experiments, can prevent the detection of problematic performance. To highlight the problem with such a technique, let say we got the following set of AP score: $Scores = \{0.30, 0.40, 0.35, 0.90, 0.95, 0.85\}$. Taking the mean of $Scores$, we obtain the mAP score of 0.625, and we may consider that our approach is successful enough. However, the $Scores$ set show two clear trends ($\{0.30, 0.40, 0.35\}$ and $\{0.90, 0.95, 0.85\}$) in the AP scores that may indicate insidious problems in the approach taken that the mean does not reveal.

3.2 HyperParameters search

Our approach comprises several hyperparameters to select. First is the maximum allowable number γ of keypoints in an image. From experiments, increasing γ beyond 500 keypoints did not significantly improve the performance of any descriptor. The second hyperparameter is the downsizing factor ϕ of the original image. Downsizing an image allowed the receptive field of any method to be increased, without changing its processing pipeline. Our experiments showed that using a downsizing factor of $\phi = 2$ generally helped every descriptor. Our third hyperparameter is the sigma σ used in the blurring performed before passing the image through the keypoint detector. Note that the blur was done for the keypoint detection, but after that we used either the unblurred image for computing the description of learned descriptors

(`DeepBark`, `SqueezeBark`, `DeepDesc` and `DeepDescBark`) or the blurred image for SIFT and SURF. The latter was necessary, as they used the keypoint information found on the blurred image. We found that the best blur filter value σ varied greatly between descriptors. A sample of the results is in Table 3.1 and the chosen values for the section 3.4 experiment are shown in Table 3.2. These values were found by averaging the results of 36 randomly-selected queries run against the validation set for each hyperparameter combination.

Descriptors	ϕ	GV				LR			
		σ				σ			
		0	1	2	3	0	1	2	3
DeepBark	1.0	0.862	0.875	0.881	0.883	0.834	0.841	0.825	0.856
	1.5	0.941	0.947	0.934	0.805	0.901	0.915	0.895	0.907
	2.0	**0.966**	0.954	0.818	0.556	**0.921**	0.917	0.906	0.876
SqueezeBark	1.0	0.109	0.119	0.134	0.139	0.686	0.680	0.675	0.662
	1.5	0.136	0.149	0.148	0.131	0.652	0.663	0.673	0.676
	2.0	0.183	**0.189**	0.159	0.126	**0.732**	0.723	0.706	0.715
SURF	1.0	0.174	0.218	0.321	0.399	0.199	0.235	0.302	0.329
	1.5	0.324	0.359	0.413	0.459	0.272	0.314	0.366	0.418
	2.0	0.315	0.395	0.447	**0.489**	0.322	0.363	0.397	**0.429**
SIFT	1.0	0.120	0.239	0.389	0.441	0.259	0.377	0.460	0.495
	1.5	0.181	0.353	**0.454**	0.406	0.305	0.423	0.535	0.541
	2.0	0.237	0.398	0.376	0.208	0.338	0.461	**0.542**	0.492
DeepDesc	1.0	0.072	0.068	0.062	0.067	0.131	0.125	0.119	0.124
	1.5	0.076	0.081	0.069	0.064	0.161	0.174	0.149	0.127
	2.0	**0.092**	0.091	0.065	0.066	0.169	**0.199**	0.161	0.124
DeepDescBark	1.0	0.058	0.070	0.068	0.069	0.244	0.288	0.276	0.303
	1.5	0.067	0.072	0.074	0.081	0.344	0.341	0.381	0.377
	2.0	0.076	**0.084**	**0.084**	0.073	**0.458**	0.441	**0.458**	0.419

Table 3.1: Results for the hyperparameters grid search, averaged over 36 random queries on the validation set. The downsize factor ϕ is how much the size of the original image was divided, and σ is the sigma used for the gaussian blur on the image before keypoint detection. The values reported for the GV and LR methods are the mAP metric.

Aside from the mAP metric, we also calculated the P@1, R-P, F1 and the AUC metric to find the optimal set of hyper-parameters to use for every method. Thus, some of the hyper-parameters chosen in Table 3.2 may not have the best mAP value as in Table 3.1. We selected those hyper-parameters based on different metrics for either the GV or the LR method. We did this because, as it is shown in Table 3.1, some descriptors performed better with a certain scoring method than with another one. So, we made sure to chose the hyper-parameters

Descriptors	ϕ	σ	GV	LR	Avg. Keypoint Num.
DeepBark	2.0	0	0.966	0.921	492.8 ±18.4
SqueezeBark	2.0	0	0.183	0.732	492.8 ±18.4
SURF	2.0	3	0.489	0.429	499.6 ±4.8
SIFT	1.5	3	0.406	0.541	469.4 ±69.9
DeepDesc	2.0	1	0.091	0.461	497.0 ±17.4
DeepDescBark	2.0	0	0.076	0.458	492.8 ±18.4

Table 3.2: Hyperparameters chosen after careful examination of the grid search. Shown is the mAP metric for the GV and LR methods. We also report the mean number of keypoints found at test time. The number of keypoints was capped to 500.

yielding the best improvement for the scoring method in which each descriptor performed best.

3.3 Impact of training data set size

Data-driven approaches based on Deep Learning tend to be data-hungry. To check the impact of the training data size, we created five training scenarios by tree species, which used 10%, 20%, 30%, 40%, and 50% of the dataset. For this experiment, we chose the learned descriptor `DeepBark`, which we validated and tested on the same sets (10% and 40% respectively) of each species dataset. We stopped training when the validation P@1 stagnated for 40 consecutive iterations.

Table 3.3 shows the performance of `DeepBark`, for each training set size. For each species, the P@1, the R-P, mAP metric with their standard deviation are reported as well as the F1 and AUC metric for the three scoring methods: GV, LR and BoW. What can be seen from this table is that an increase in data improves the results of any metric, any method and also decreases the standard deviation, meaning that more data helps obtain more reliable/stable matching. When looking at the P@1 and the mAP, we can see that finding a single similar bark image is an easy task, but finding every variation of the bark surface necessitates more training data. In the Red Pine case, using 50% of the training generally yielded the best results. However, an interesting observation is that in the case of the Elm, 40% of the training data gave better results than using 50%. But, we believe that this is not statistically significant, given the randomness of the training process and that the difference is minimal.

The US-PR curves for Elm (EL) and Red Pine (RP) are displayed in Figure 3.3 and Figure 3.4, for various training set sizes. Results are reported for all three scoring approaches. The first thing that we notice in them is the lack of smoothness when using the GV method. Nonetheless, the training set size affects the GV curve the same way as for the BoW and the LR scoring methods. This means that this is an inherent problem to the GV method. But, it seems to be slightly mitigated by an increase in the training set size. Other than that, it

Red Pine						
	Metric	10%	20%	30%	40%	50%
BoW	P@1	0.971 ±0.168	0.985 ±0.120	0.985 ±0.120	**0.996 ±0.064**	0.994 ±0.079
	R-P	0.578 ±0.196	0.651 ±0.188	0.705 ±0.174	0.722 ±0.181	**0.751 ±0.171**
	mAP	0.633 ±0.207	0.713 ±0.198	0.769 ±0.178	0.785 ±0.187	**0.812 ±0.173**
	AUC	0.896	0.923	0.940	**0.941**	0.940
	F1	0.523	0.605	0.649	0.663	**0.703**
GV	P@1	0.988 ±0.111	0.990 ±0.102	**0.998 ±0.046**	0.996 ±0.064	**0.998 ±0.046**
	R-P	0.727 ±0.132	0.790 ±0.112	0.828 ±0.097	0.842 ±0.093	**0.857 ±0.087**
	mAP	0.777 ±0.148	0.848 ±0.122	0.892 ±0.098	0.905 ±0.091	**0.922 ±0.080**
	AUC	0.939	0.956	0.958	0.960	**0.968**
	F1	0.645	0.717	0.751	0.769	**0.784**
LR	P@1	**1.000 ±0.000**	**1.000 ±0.000**	**1.000 ±0.000**	**1.000 ±0.000**	**1.000 ±0.000**
	R-P	0.822 ±0.141	0.890 ±0.115	0.921 ±0.099	0.930 ±0.093	**0.938 ±0.086**
	mAP	0.882 ±0.123	0.932 ±0.092	0.956 ±0.075	0.962 ±0.067	**0.967 ±0.062**
	AUC	0.893	0.923	0.944	0.947	**0.955**
	F1	0.724	0.792	0.840	0.859	**0.874**

Elm						
	Metric	10%	20%	30%	40%	50%
BoW	P@1	0.940 ±0.238	0.956 ±0.205	0.971 ±0.168	0.979 ±0.143	**0.983 ±0.128**
	R-P	0.558 ±0.226	0.635 ±0.226	0.662 ±0.218	**0.710 ±0.230**	0.706 ±0.223
	mAP	0.607 ±0.247	0.691 ±0.238	0.721 ±0.226	0.759 ±0.231	**0.764 ±0.223**
	AUC	0.894	0.874	**0.923**	0.893	0.909
	F1	0.532	0.612	0.651	**0.689**	**0.689**
GV	P@1	0.944 ±0.230	0.965 ±0.185	0.977 ±0.150	0.981 ±0.136	**0.983 ±0.128**
	R-P	0.670 ±0.188	0.706 ±0.174	0.742 ±0.172	**0.763 ±0.166**	0.757 ±0.169
	mAP	0.707 ±0.213	0.752 ±0.202	0.791 ±0.189	**0.816 ±0.180**	0.806 ±0.187
	AUC	0.922	0.937	0.942	**0.948**	0.935
	F1	0.640	0.678	0.713	**0.734**	0.722
LR	P@1	0.985 ±0.120	0.996 ±0.064	0.998 ±0.046	0.998 ±0.046	**1.000 ±0.000**
	R-P	0.613 ±0.209	0.689 ±0.214	0.726 ±0.204	**0.748 ±0.196**	0.747 ±0.195
	mAP	0.665 ±0.224	0.740 ±0.216	0.779 ±0.197	**0.800 ±0.190**	0.798 ±0.191
	AUC	0.876	0.903	0.915	**0.924**	0.921
	F1	0.622	0.696	0.729	**0.751**	**0.751**

Table 3.3: This table shows the `DeepBark` descriptor performance, when training with 10%, 20%, 30%, 40%, and 50% of the data from a single tree species. From the remaining data, 10% and 40% have been used for validation and testing, respectively. Hyperparameters were fixed through testing. The best results are in bold for each row.

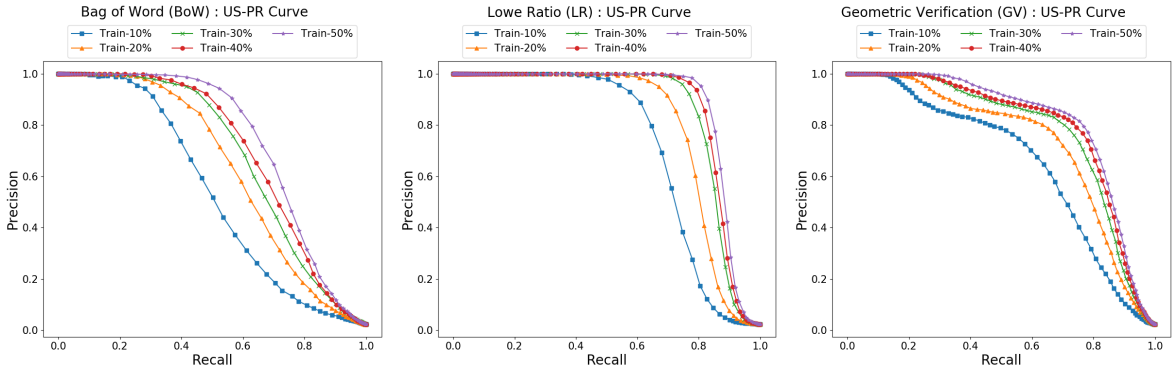


Figure 3.3: US-PR Curve for all training set sizes using RP bark.

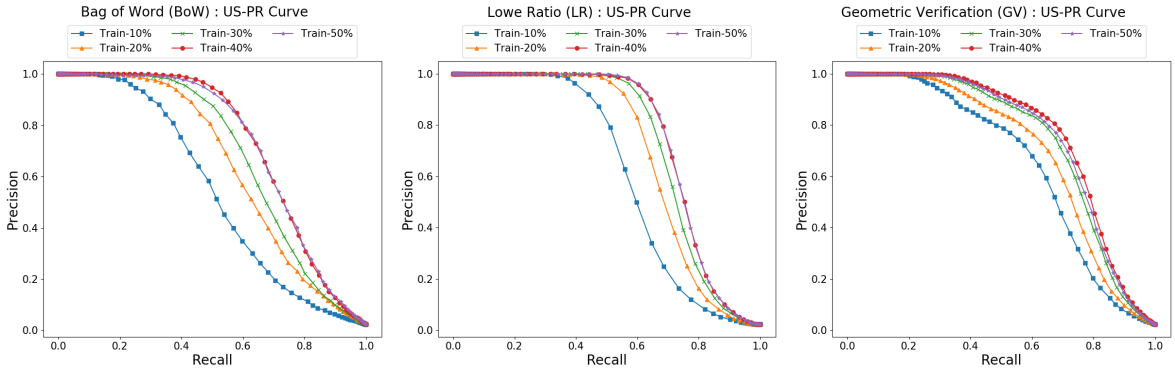


Figure 3.4: US-PR Curve for all training set sizes using EL bark.

is clear that using only 10% of the training set is not enough and that there is almost no difference in performance when using either 40% of 50% of the training data.

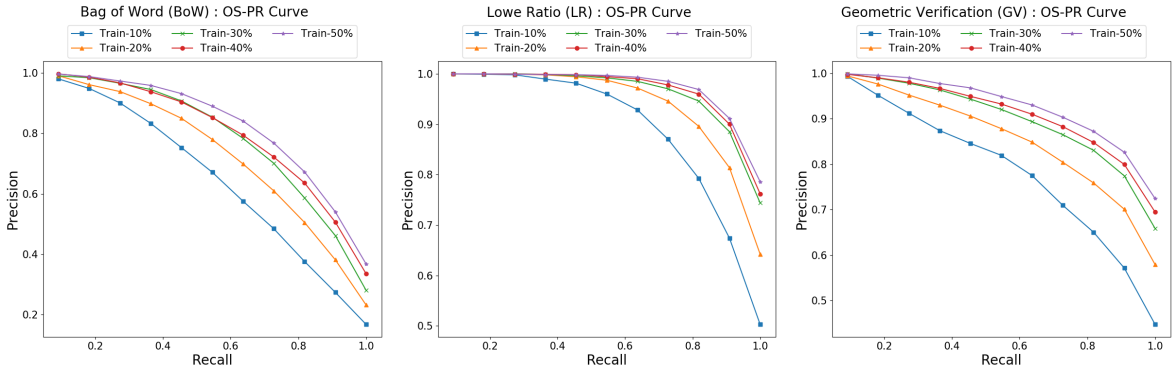


Figure 3.5: OS-PR Curve for all training set sizes using RP bark.

In like manner, the graphs in Figure 3.5 and Figure 3.6 show the OS-PR curves for all three scoring methods. The same lack of smoothness for the GV scoring method appears, but to a

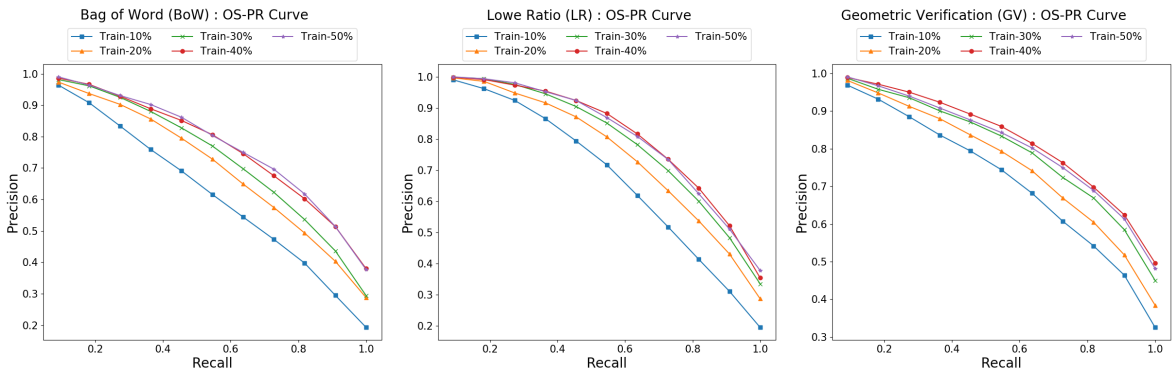


Figure 3.6: OS-PR Curve for all training set sizes using EL bark.

lesser degree. But once again, more data smoothen the GV curve. It gave a visible increase in performance up to 40% of the training set size. These figures also show that the GV and LR methods perform similarly, regardless of the recall or the training dataset size.

But, from the previous Figures 3.3, 3.4, 3.5, 3.6 we can draw two conclusions. First, the BoW performed worst than the two other methods and showed a better improvement from 10% of the training set to 30%. However, this is expected since the *voc* of the BoW is computed from the training set, which changed in size. Thus, the BoW summary representation of the full list of keypoints descriptors V will differ depending on the quality of the *voc*. Second, the increase of the training set size greatly improved the performance of every method up to 30% of the training set size, after which the gains to 40% became less apparent and negligible to 50%.

From the Table 3.3 and Figures 3.3, 3.4, 3.5, 3.6, we concluded that performance gains were minimal beyond 40% and decided to use that much training data in our experiments. This validates that our training database is sufficiently large to obtain good performance. For references, when using 50% of RP as training data, we have access to approximately 42,700 distinct keypoints, giving 512,000 bark image patches of 64x64 pixels. Note that this order of magnitude in the number of training examples is consistent with generally-accepted practices of deep learning.

3.4 Comparing hand-crafted vs. data-driven descriptors

An important objective of this thesis was aimed at evaluating the capacity of different local descriptors to re-identify tree bark images. More specifically, we wanted to learn if traditional hand-crafted descriptors are sufficient for this task or the new learned descriptors would be needed to achieve reasonable performance. To do so, this section compare the performance of multiple descriptors that are listed here: SIFT, SURF, `DeepDesc`, `DeepDescBark`, `SqueezeBark`

and `DeepBark`. The descriptors SIFT, SURF are the hand-crafted category representative, while all the other descriptors are representative of the learned category.

Since many variations could be compared among the learned descriptors, we included more of them. Hence, `DeepDesc` serves to illustrate the expected behavior when no training is done with bark data and can be directly compared with `DeepDescBark` that is our version of `DeepDesc` train on bark data. Also, we used `SqueezeBark` to evaluate the performance of a deep neural network with little weight parameters and `DeepBark` to portray the performance we expect from a deep neural network that obtained the state of the art results on ImageNet.

For this experiment, we selected 50% of red pine bark surfaces and 50% of elm bark surfaces to create a test set, while using the remaining data for the training and validation sets. This corresponded to 80 unique bark surfaces for the training, 20 for the validation, and 100 for testing. We kept the ratio between tree species to 50/50 in each set. The data-driven descriptors `DeepBark`, `SqueezeBark` and `DeepDescBark` were trained for 200 iterations, and we kept the model with the best validation. With 12 images for each bark surface, the test set had a total of 1200 images, with 600 images per tree species. Each of these images was used as a query during the retrieval test. The results were averaged over all queries.

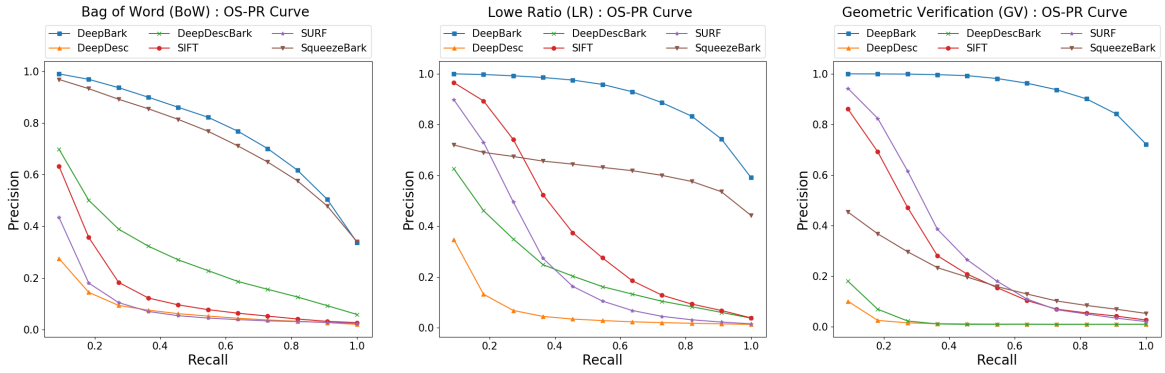


Figure 3.7: OS-PR Curve for all descriptors and scoring methods tested on 50% of RP and EL. Learned descriptors were trained on the remaining 50% of bark. Each of the 1200 images of the test set was used as a query. No extra negative examples were added.

Figure 3.7 and Figure 3.8 show the comparative performance of the six descriptors, in the form of OS-PR and ROC curves, respectively. Based on the Figure 3.7, we can assess to a certain extent, the invariance of the descriptors to changes in illumination and viewpoint. Using the assumption that more similar images are retrieved first, we can look at which recall (and thus rank K) a descriptor loose precision and then evaluate how much variations it can handle before having difficulties to retrieve images. In the case of the hand-crafted descriptors, we can see that they can often successfully retrieve one image, but struggle beyond this. This is visible by their precision over 0.8 for the first image retrieve with LR or GV. Which then quickly drops and reaches a precision of almost 0 at a recall of 1. This indicates that they

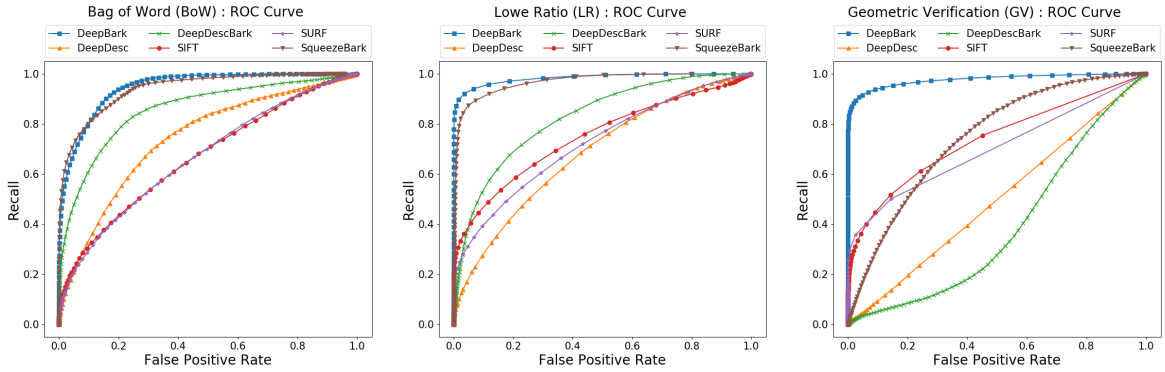


Figure 3.8: ROC Curves for all descriptors and scoring methods tested on 50% of RP and EL. Learned descriptors were trained on the remaining 50% of bark. Each of the 1200 images of the test set was used as a query. No extra negative examples were added.

can retrieve the most similar images relative to the query but then fail to match the relevant images that are more challenging, i.e., have more variations in appearance due to illumination or camera-pose changes. When analyzing ROC Curves, it is common to compare performance to the diagonal going from (0,0) to (1,1), which represents the performance expected by making random choices. Thus, when looking at Figure 3.8, we see that the descriptions generated by SIFT and SURF are distinctive enough to beat random choice, but not by much. From this, we can see that hand-crafted descriptors on our bark retrieval task generally offer a poorer performance over data-driven ones. More precisely, it seems that hand-crafted descriptors can re-identify similar bark images, but when face with illumination and viewpoints variations, they start performing similarly to random choices. However, hand-crafted approaches offer more consistent behavior across all scoring methods.

One of our proposed approach, the *data-driven* descriptor `DeepBark`, obtains overall the best performance. For instance, when looking at Table 3.4, `DeepBark` wins in almost any metric with the GV method, except for the F1 metric where it still dominates, but with the LR method. Even when comparing every descriptor by scoring methods, `DeepBark` performs better than every one. One exception is for the BoW when using AUC and F1, where it has score respectively 0.842 and 0.439. For the BoW with the AUC, there is `DeepDescBark` that reach a score of 0.856. However, it is `SqueezeBark` that show the best performance with an AUC and F1 score of respectively 0.901 and 0.496. Since P@1, R-P and mAP are the average of multiple queries, we calculated their standard deviation and demonstrated once again that `DeepBark` perform better by having smaller standard deviation in general, a sign of more reliable performance.

Interestingly, the results for our second *data-driven* approach, `SqueezeBark`, are mitigated as shown in Figure 3.7 and Figure 3.8. Using the GV method gave worse performances than the two hand-crafted descriptors and a total lack of invariance to illumination and viewpoint

Descriptors	Scoring Methods	Metrics				
		P@1	R-P	mAP	AUC	F1
SIFT	BoW	0.563 ±0.496	0.157 ±0.156	0.153 ±0.157	0.666	0.099
	GV	0.808 ±0.394	0.278 ±0.167	0.270 ±0.170	0.737	0.248
	LR	0.953 ±0.213	0.375 ±0.184	0.390 ±0.197	0.748	0.330
SURF	BoW	0.336 ±0.472	0.101 ±0.122	0.095 ±0.122	0.664	0.085
	GV	0.930 ±0.255	0.311 ±0.175	0.318 ±0.183	0.701	0.354
	LR	0.875 ±0.331	0.252 ±0.141	0.259 ±0.149	0.722	0.248
DeepDesc	BoW	0.170 ±0.376	0.080 ±0.115	0.078 ±0.104	0.739	0.069
	GV	0.053 ±0.225	0.019 ±0.043	0.021 ±0.027	0.497	0.018
	LR	0.249 ±0.433	0.075 ±0.078	0.068 ±0.061	0.683	0.068
DeepDescBark	BoW	0.609 ±0.488	0.268 ±0.216	0.275 ±0.232	0.856	0.236
	GV	0.131 ±0.337	0.035 ±0.067	0.033 ±0.056	0.391	0.018
	LR	0.557 ±0.497	0.217 ±0.198	0.225 ±0.204	0.823	0.131
SqueezeBark	BoW	0.950 ±0.218	0.668 ±0.216	0.726 ±0.222	0.901	0.496
	GV	0.412 ±0.492	0.181 ±0.228	0.196 ±0.244	0.744	0.053
	LR	0.627 ±0.484	0.594 ±0.372	0.617 ±0.374	0.766	0.512
DeepBark	BoW	0.984 ±0.125	0.702 ±0.206	0.764 ±0.212	0.842	0.439
	GV	**1.000 ±0.000**	**0.913 ±0.134**	**0.940 ±0.112**	**0.965**	0.853
	LR	**1.000 ±0.000**	0.864 ±0.163	0.899 ±0.147	0.924	**0.862**

Table 3.4: Different metrics for all descriptors and scoring methods tested on 50% of RP and EL. Learned descriptors were trained on the remaining 50% of bark. Each of the 1200 images of the test set was used as a query. The best result for each metric is in bold. These results clearly show the advantage of learned descriptors trained with bark data. Especially `DeepBark` that dominates every other descriptor in almost any combination of scoring methods and metrics.

changes. However, `SqueezeBark` did largely better with the LR method, as seen in Figure 3.7. There, the precision starts worse than SIFT and SURF. But the same level of precision is almost retained up to the last relevant image. Curiously `SqueezeBark` shows its best performance using the BoW method, which uses a summary representation of the descriptions list V that should, in theory, be less accurate. This might indicate that finding a good descriptor for bark images under strong illumination changes is a difficult problem, requiring a neural architecture with sufficient capacity. This is further supported by `DeepDescBark` performing worse than `SqueezeBark` and `DeepBark`, which are bigger networks.

Importantly, `DeepDescBark` exhibits great improvements over `DeepDesc`. Indeed, the latter displays the worst performance of any descriptor by far, which reiterates the importance of

using the appropriate training data. Nevertheless, the situation in terms of overall performance is still unclear for `DeepDescBark`. Indeed, `DeepDescBark` seems to follow the trend of `SqueezeBark`, but with worse performance. It gives poor performance when used with the GV method, and then it almost catches up with SIFT and SURF using the LR method, as can be seen in Figure 3.7. Finally, while remaining less efficient than `SqueezeBark`, it does beats hand-crafted descriptors with the BoW by a good margin.

3.5 Generalization across species

In the experiments of section 3.4, we reported results on networks trained on both species, instead of training and testing each architecture on a single species. We intended to double the amount of training data, thus benefiting from the potential synergy between species. This kind of synergy is often seen in deep network training, exemplified with the paradigm of multi-task learning. Domain shift is known to be particularly problematic for deep networks. We thus wanted to look at the capability of our networks, `DeepBark` and `SqueezeBark`, to generalize across species. Hence, we devised two experiments to evaluate the generalization from one species to the other, and vice versa. The first one is composed of a training set with 80% of the RP data, using the remaining 20% as the validation set. All of the EL data is, naturally, in the test set (labeled RP→EL). We also performed the converse, labeled EL→RP.

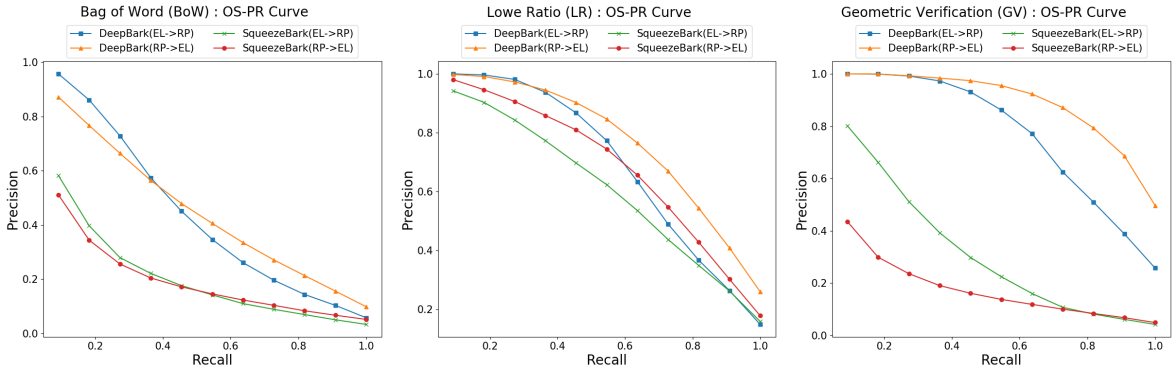


Figure 3.9: OS-PR curves for the generalization test across species. The arrow → indicates the species generalization direction (trained on → tested on).

Figure 3.9 and Figure 3.10 show clearly the domination of `DeepBark` compared to `SqueezeBark`. Also, comparing the two networks based on the generalization direction with Table 3.5 demonstrates the constant superiority of `DeepBark`. Indeed, the latter has better results in every metric and for any method for this generalization test. Moreover, the standard deviation of its performance is smaller in almost every case. The bold numbers in Table 3.5 indicating the best results let us quickly appreciate the perfect scores that `DeepBark` obtained for the P@1, in both species generalization direction.

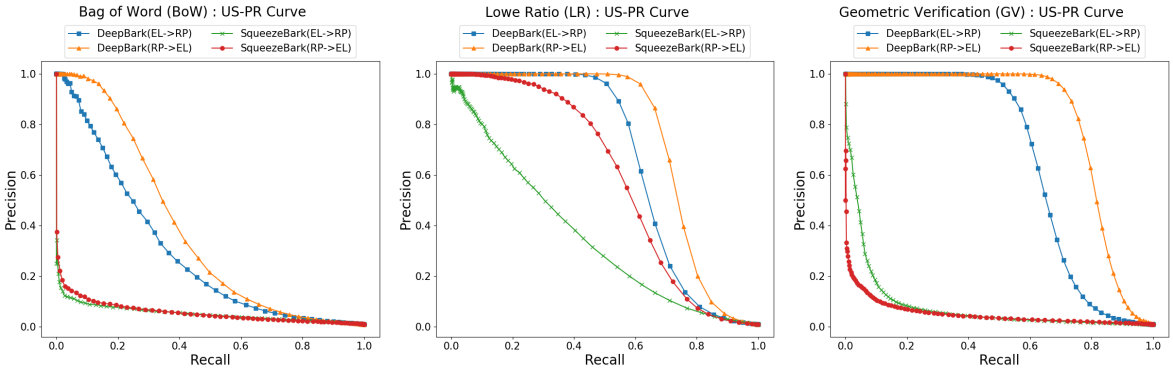


Figure 3.10: US-PR curves for the generalization test across species. The arrow → indicates the species generalization direction (trained on → tested on)

When inspecting which bark species generated the best generalization capacity, it becomes clear that training on RP and testing on EL gave better performance. This appears in Table 3.5, where `DeepBark` (RP→EL) shows the best results across all metrics. The same trend also appears in Figure 3.9 where the two generalization directions are compared for `DeepBark` with the GV method. Finally, the strongest demonstration of this is in Figure 3.10, where (RP→EL) with `DeepBark` always dominates the opposite generalization direction. In the case of `SqueezeBark`, if we put aside the GV method that previously gave questionable results, the two other methods also show an advantage for the (RP→EL) direction. However, it is unclear to us why this trend is present. The first reason could be that EL bark is easier to match correctly, but other reasons such as more precise spatial transformations available or more diverse lighting directions present in the RP bark dataset could cause a situation such as this. Nonetheless, since we took great care to apply the same data collection methodology for any bark species, we suspect more strongly that this generalization direction trend comes from a key difference between tree species.

The important conclusion to reach in this section, is the capacity of our descriptor `DeepBark` to perform well on an unseen tree bark species. This is particularly important since it would be extremely difficult to collect a training dataset containing the exhaustive set of all existing trees species. This is because a real-world application is bound to comprise previously-unseen tree species for the descriptor network. However, looking at Figure 3.9 and Figure 3.10, `DeepBark` shows satisfactory results across species generalization, for the GV and LR methods. If we consider the BoW method as a pre-selection tool (as will be done in section 3.7), the results are acceptable too. Thus, we consider that `DeepBark` has a good generalization capacity.

Descriptors	Scoring Methods	Metrics				
		P@1	R-P	mAP	AUC	F1
DeepBark (RP→EL)	BoW	0.818 ±0.386	0.420 ±0.237	0.439 ±0.264	0.885	0.409
	GV	**1.000 ±0.000**	**0.846 ±0.166**	**0.880 ±0.154**	**0.952**	**0.809**
	LR	0.998 ±0.050	0.713 ±0.204	0.755 ±0.206	0.939	0.751
SqueezeBark (RP→EL)	BoW	0.397 ±0.489	0.188 ±0.193	0.188 ±0.199	0.817	0.119
	GV	0.348 ±0.476	0.163 ±0.194	0.170 ±0.206	0.792	0.108
	LR	0.970 ±0.171	0.637 ±0.229	0.669 ±0.241	0.913	0.589
DeepBark (EL→RP)	BoW	0.939 ±0.239	0.402 ±0.179	0.426 ±0.199	0.864	0.346
	GV	**1.000 ±0.000**	0.707 ±0.177	0.755 ±0.181	0.917	0.685
	LR	**1.000 ±0.000**	0.632 ±0.186	0.678 ±0.189	0.924	0.677
SqueezeBark (EL→RP)	BoW	0.479 ±0.500	0.196 ±0.173	0.196 ±0.176	0.785	0.111
	GV	0.770 ±0.421	0.292 ±0.213	0.303 ±0.230	0.764	0.130
	LR	0.922 ±0.269	0.559 ±0.246	0.593 ±0.266	0.916	0.393

Table 3.5: Results of retrieval test based on generalization across species. The arrow → indicates the generalization direction (trained on → tested on). The best results for each metric are in bold.

3.6 More realistic scenario: adding negative examples

To test how our system would perform on a larger database that would be typical of a real deployment, we added 6,700 true negative elm examples with a crop size similar to query images. Half of them were original images, and the other images were generated via data augmentation by doing either a rotation, scale, or affine transformation. Note that the original 3,350 images contain some physical overlap, as they come from 25 trees.

We reused the `DeepBark` network and the *voc* previously trained in section 3.4. For the test, we removed the red pine images and kept the elm images that we separated into two crops (top and bottom halves), giving us a total of 1,200 images. We thus obtained a database of 7,900 bark images. Again, every query had 11 relevant images. Importantly, this experiment is the only one where we split the bark images into two crops, solely done to increase the database size. This had the side effect of also dropping the performance. Since the visible bark (and thus, the number of visible features) was reduced by half. This can be seen by comparing Figure 3.7 and Figure 3.11.

Inspecting Table 3.6, it is clear that the extra negative data has an impact on the results of our two representative descriptors `DeepBark` and SIFT. Indeed, it is apparent that the different metrics worsen, even with only 600 new negative examples. Among the GV, LR, and the BoW, the most affected one is, without a doubt, the BoW. Looking at Figure 3.11 and

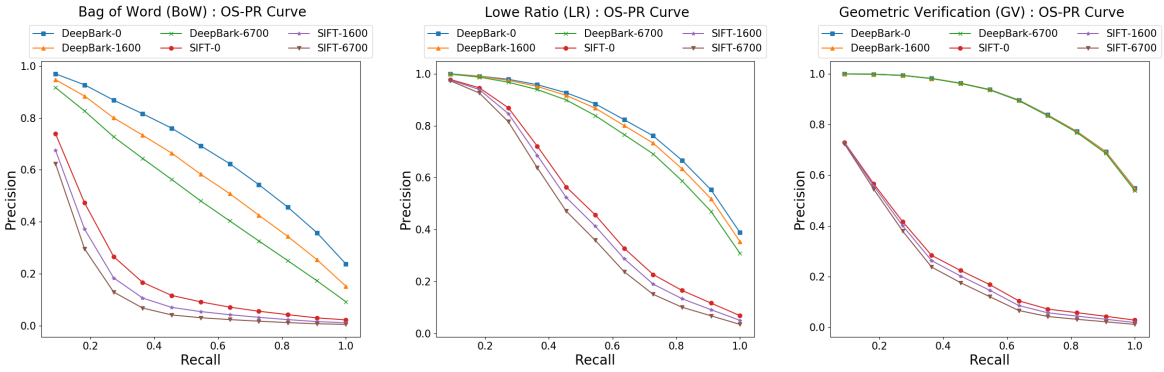


Figure 3.11: OS-PR Curves for the negative examples test on SIFT and `DeepBark`. Numbers in the legend indicate how many negative examples were added.

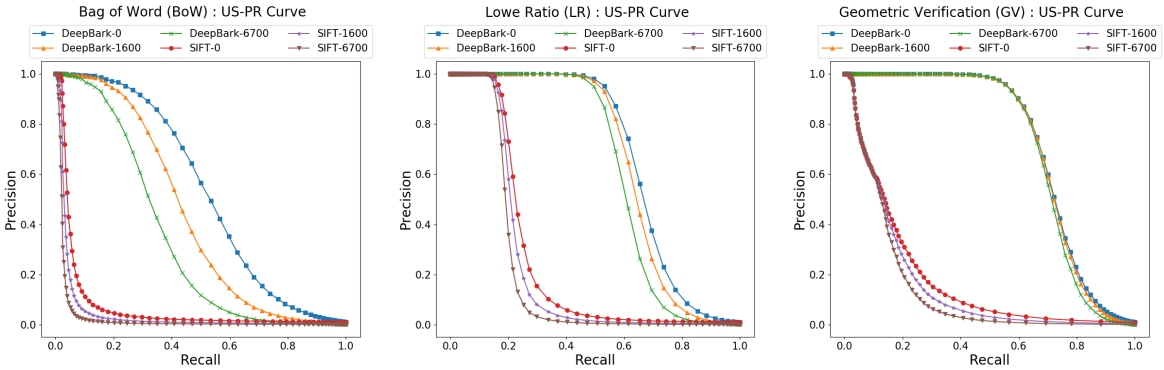


Figure 3.12: US-PR Curves for the negative examples test on SIFT and `DeepBark`. Numbers in the legend indicate how many negative examples were added.

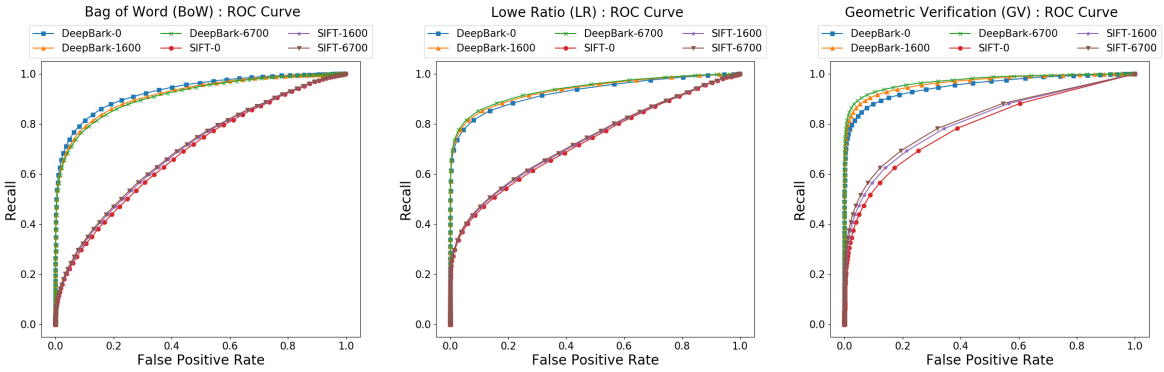


Figure 3.13: ROC Curves for the negative examples test on SIFT and `DeepBark`. Numbers in the legend indicate how many negative examples were added.

Figure 3.12, the effect of negative data on the BoW is quite apparent. We conjecture that this effect is caused by the summary representation that is the BoW and the fact that it does not

use any extra information available from the set of match M.

In comparison, the GV and LR scoring method are more resilient against the augmented number of images and the visible area reduction. However, it seems like the GV scoring method is the clear winner here, with both descriptors showing a smaller loss of performance going from 0 to 6700 added negative examples, which is a surprise since LR was design with SIFT in mind. Carefully analyzing the GV and the LR in the case of SIFT, we can even see that LR initially give better performance than the GV. But as the number of added negative examples increase, the LR performance drop so much that the GV end up with better performance. Thus, it suggests that to correctly re-identify bark images among large database require more than a summary representation of an image and that relying on the geometric consistency of local characteristics in images coming from a unique bark surface is advantageous.

Even if Figure 3.13 could be used to show that `DeepBark` outperform SIFT with smaller crop size and additional negative bark images, we keep it here to demonstrate the pitfall of some metric used in image retrieval. Except for the correctly represented performance gap between the two descriptors, Figure 3.13 is wrong about its representation of the performance when more negative data is added. Looking carefully, we notice that this figure would lead us to believe that the more negative examples are added, the more the performance increase. Examining other metrics show that this is wrong, but taking as an example the AUC in Table 3.6, we notice that the best results are almost always the last one, which is incorrect. This miss-representation is due to how FPR is calculated and reminds us to always stay vigilant of the metric we use.

From Table 3.6, it is clear that `DeepBark` outperforms SIFT either from looking at the highest results or the smaller decrease of performance when adding negative examples. For instance, if we look at the mAP in the column labeled 6700, `DeepBark` always score higher. The closest SIFT could get to `DeepBark`, was when compared with the LR, and it still shows a 0.334 difference between the two scores. On the side of keeping good performance while adding negative examples, `DeepBark` is visibly better with bold results in some metrics starting from 0 added examples to 6700, indicating no drop in performance. However, even if this re-iterate the point that learned descriptors are necessary to allow tree bark re-identification as showed in section 3.4, SIFT is presented here more to show a baseline and the effect that the added negative examples can have. The real conclusion to draw here is that when using the `DeepBark` descriptor in combination with the GV scoring method, we can expect satisfactory performance even with a realistic database of more than 7900 bark images. We expect this, since the combination of `DeepBark` and GV has shown only a negligible drop of performance no matter the number of added negative examples.

DeepBark						
	Metric	0	600	1600	3300	6700
BoW	P@1	**0.952 ±0.214**	0.937 ±0.244	0.924 ±0.265	0.910 ±0.286	0.885 ±0.319
	R-P	**0.611 ±0.238**	0.569 ±0.243	0.537 ±0.248	0.504 ±0.250	0.471 ±0.252
	mAP	**0.659 ±0.258**	0.611 ±0.269	0.572 ±0.276	0.532 ±0.279	0.491 ±0.281
	AUC	**0.918**	0.914	0.911	0.907	0.907
	F1	**0.542**	0.505	0.471	0.431	0.394
GV	P@1	**0.998 ±0.041**	**0.998 ±0.041**	**0.998 ±0.041**	**0.998 ±0.041**	**0.998 ±0.041**
	R-P	**0.832 ±0.179**	**0.832 ±0.178**	**0.832 ±0.178**	**0.832 ±0.178**	0.831 ±0.179
	mAP	**0.874 ±0.165**	**0.874 ±0.165**	**0.874 ±0.166**	0.873 ±0.167	0.872 ±0.168
	AUC	0.951	0.959	0.964	0.968	**0.970**
	F1	**0.723**	**0.723**	0.722	0.721	0.720
LR	P@1	**0.999 ±0.029**	**0.999 ±0.029**	0.998 ±0.050	0.998 ±0.050	0.998 ±0.050
	R-P	**0.769 ±0.203**	0.763 ±0.206	0.756 ±0.209	0.748 ±0.214	0.736 ±0.221
	mAP	**0.812 ±0.198**	0.804 ±0.203	0.794 ±0.209	0.783 ±0.215	0.768 ±0.223
	AUC	0.926	0.930	0.933	0.935	**0.937**
	F1	**0.690**	0.684	0.677	0.671	0.659

SIFT						
	Metric	0	600	1600	3300	6700
BoW	P@1	**0.683 ±0.466**	0.653 ±0.476	0.622 ±0.485	0.601 ±0.490	0.576 ±0.494
	R-P	**0.189 ±0.149**	0.164 ±0.142	0.149 ±0.135	0.136 ±0.130	0.124 ±0.121
	mAP	**0.189 ±0.158**	0.163 ±0.148	0.144 ±0.140	0.129 ±0.131	0.114 ±0.121
	AUC	0.684	0.686	0.694	0.697	**0.698**
	F1	**0.108**	0.093	0.082	0.071	0.059
GV	P@1	**0.586 ±0.493**	**0.586 ±0.493**	**0.586 ±0.493**	**0.586 ±0.493**	0.585 ±0.493
	R-P	**0.271 ±0.161**	0.264 ±0.159	0.263 ±0.159	0.257 ±0.158	0.250 ±0.155
	mAP	**0.245 ±0.154**	0.235 ±0.153	0.230 ±0.153	0.223 ±0.153	0.214 ±0.150
	AUC	0.788	0.799	0.807	0.812	**0.817**
	F1	**0.248**	0.237	0.233	0.225	0.216
LR	P@1	**0.972 ±0.166**	0.971 ±0.168	0.969 ±0.173	0.968 ±0.177	0.967 ±0.180
	R-P	**0.465 ±0.201**	0.458 ±0.201	0.449 ±0.201	0.437 ±0.200	0.425 ±0.200
	mAP	**0.495 ±0.218**	0.481 ±0.219	0.467 ±0.219	0.450 ±0.218	0.434 ±0.217
	AUC	0.727	0.731	0.734	0.735	**0.737**
	F1	**0.316**	0.308	0.299	0.292	0.283

Table 3.6: Results of the negative examples test for `DeepBark` and SIFT. The numbers in the header indicate how many negative examples were added. The best results for each row are in bold.

3.7 Speeding-up classification with Bag of Words (BoW)

Even though the LR and the GV scoring methods perform better in terms of mean precision, it is unrealistic to use them to search a whole database. Indeed, in a realistic scenario, this database might contain over 10,000 images, which would be prohibitively expensive to search with the LR or the GV scoring methods. Instead, the BoW approach can be used as a pre-filter to propose putative candidates to other more computationally-expensive but accurate methods. As shown by Cummins and Newman (2009), a BoW is a fast method to compare and handle large datasets. To obtain a sense of the time that could be saved by such pre-filtering, we report in Table 3.7 the average time for a single comparison of all three scoring methods, on a single thread. Even if we compared the BoW to the second-fastest method (LR), computing a BoW is still 65,750 time faster. However, it is important to note that our implementation of the LR and GV methods are rudimentary and could probably be rewritten to obtain more speed.

Nevertheless, the magnitude of the computational gap would remain, especially considering that our BoW version could also be improved. Using a BoW of size 1000, we calculated that, on average, a BoW has 71.8% of its entries being null. One improvement could thus come from taking advantage of this sparsity. It would also be possible to improve this method by using what is called an inverted index commonly used in information retrieval, as they explain in section 1.2 of Manning et al. (2008).

Scoring Method	BoW	LR	GV
Mean Computation Time (ms)	0.002	131.499	179.387

Table 3.7: Single signature comparison time, averaged over 500 comparisons. Done using our single-thread implementation, running on an Intel Core i7-6700K 4.00GHz.

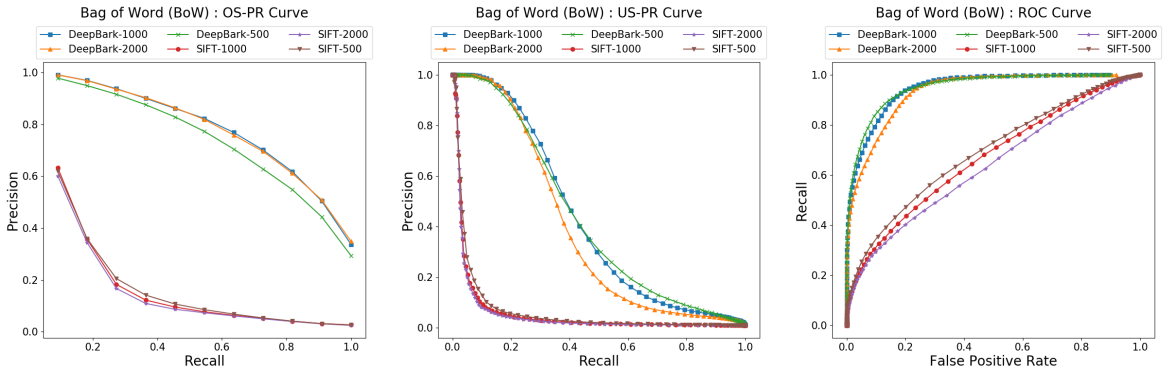


Figure 3.14: Impact of the BoW size on the retrieval performance. From left to right: OS-PR, US-PR and ROC Curves for the SIFT and `DeepBark` descriptors, for three different BoW sizes. The composition and separation of the dataset is the same as in the comparative experiment in section 3.4. Numbers in the legend indicate the BoW size, ranging from 500 to 2000.

Other than time, the precision and recall of the BoW approach should also be considered. Inspecting Figure 3.14, we can understand the effect of the actual BoW size on its performance. This size is, in effect, the number of words in the visual dictionary. For SIFT, the increase in size is slightly detrimental to the performance. However, SIFT is already underperforming, leaving little room for a large degradation. In the case of `DeepBark`, the situation is less clear. In the OS-PR graph, only a size of 500 gives worse performance, while a size of 1000 and 2000 behave similarly.

On the other hand, the US-PR graph shows that using a size of 2000 is detrimental to performance. Note that the three different BoW sizes seems to give equal performance at low recall. Finally, the ROC graph agrees with the US-PR graph, but to a smaller extent. From this, we concluded that our BoW size of 1000 was the most appropriate for all of our experiments.

Descriptors	R@25	R@50	R@100	R@200
SIFT-0	0.248	0.316	0.403	0.520
SIFT-6700	0.150	0.176	0.215	0.268
DeepBark-0	0.728	0.795	0.857	0.908
DeepBark-6700	0.561	0.625	0.681	0.739

Table 3.8: R@K for different values of K using the BoW. Results taken from the experiment with negative examples. The number next to the method name indicates how many negative examples were added.

Nonetheless, Figure 3.14 only let us look at precision to certain recall values, but using the BoW as a pre-filter, we will select an arbitrary number of candidate up to rank K for each query. To this end, we provide Table 3.8, which shows the R@K for various Ks. The first conclusion is that using SIFT and keeping the 200 best image matches in the setting with extra negative examples would only retain 26.8 % of the relevant images, which we judge ineffective for our application. On the other hand, keeping the 200 best matching image pairs using the BoW with `DeepBark` would retain 73.9% of the 11 relevant images among the 7,900 possible matches. The only concern is the loss of performance due to the addition of negative data. For R@25, when going from 0 extra negative data to 6700, it results in a performance loss of 24.3%. For R@200, we acknowledge a performance loss of 18.6%. Still, a recall of 73.9% for R@200 in a dataset of 7900 images is more than satisfactory performance. From this, we can see that if we were to apply the GV with our current algorithm on only the $K = 200$ top matching images given by the BoW pre-filtering, it would take 35.88 s. Note that to select these 200 candidates, we would have to match a query against a dataset of 7,900 images with the BoW, but this would only take 0.016 s, which means that the total time for the whole computation would be of 35.896 s.

3.8 Detecting Novel Tree Surfaces

So far, we have addressed in this thesis the tasks of matching an unseen image to a previously-seen one with sufficient precision. A natural extension, which we have not discussed yet, is determining if a query image is genuinely a *previously-unseen* surface. This determination is necessary because the matching stage will always return the best match, even if it does not correspond to a true positive match. However, we should expect the best matching score to be somewhat unsatisfactory.

A similar issue happens in mobile robotics, where Visual Place Recognition (VPR) is used to identify previously-visited locations. VPR is now part of many mapping solutions. Indeed, an essential task in mobile robotic is to map the environment and enable the robot to localize himself in this environment. This is known as Simultaneous Localization And Mapping (SLAM), which is a challenging problem due to multiple factors such as environmental complexity, sensors unreliability, the fusion of sensors information, and the need for low algorithmic complexity to be realistically used. Thus, algorithms such as particle filters, extended Kalman filters, and GraphSLAM were developed to give good approximated solutions to this problem. These approaches can sometimes refine the localization precision or robustness by leveraging *loop closure*, which is to recognize a previously seen location, using, for instance, image retrieval. *Loop closure* thus enables the localization algorithm to reinforce its hypothesis about its location by detecting a previously-seen location. Visual Place Recognition (VPR) is often the solution of choice to detect these. As such, it has become a powerful tool in mobile robotic to help solve loop closure.

The problem of detecting that a novel location is indeed an unseen one, is properly addressed in Cummins and Newman (2008) using a formal probability framework. In our case, we will be using a simpler approach. We believe that the absence of significant 3D structures, contrary to a scene seen by a robot's camera, allows us to employ a simpler solution. So, instead of a complex framework, we will show that using the scoring results of our `DeepBark` method, a simple threshold is enough to classify unseen images as new or not. We hypothesize that if the score distributions of relevant and non-relevant image matches possess enough distance between them, we can use a threshold discarding almost every non-relevant images, while keeping relevant ones. This means in turns that if an unseen image does not pass the threshold test, it should be considered a new surface, with no valid match in the database. This relies on the assumption that only relevant images can have a high score.

Figure 3.15 shows the *Precision* versus the *Recall* when considering the calculated scores as an unordered set, for all three scoring methods: Bag of Words (BoW), Lowe Ratio (LR) and Geometric Verification (GV). Having done 1,200 queries with 11 relevant images for each query, means that these graphs show the recall for a total of 13,200 relevant images among the 9,480,000 matches scores calculated during the experiment. As the *Recall* gives the proportion

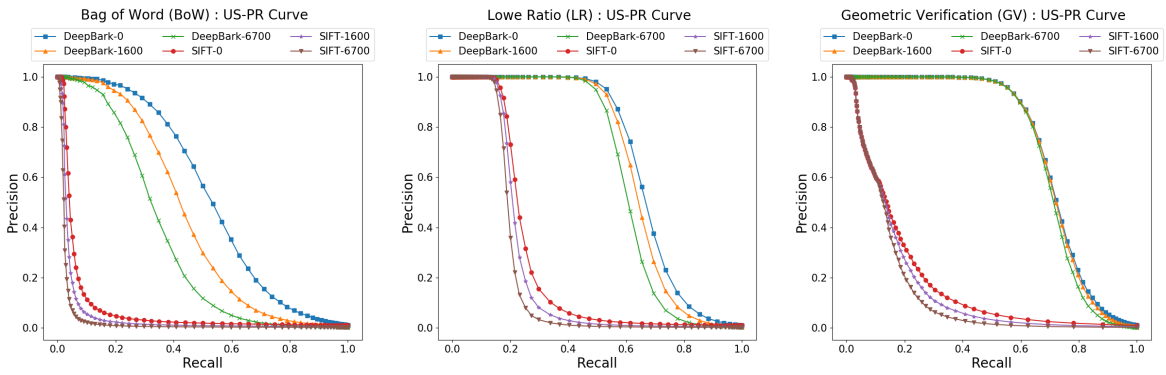


Figure 3.15: US-PR Curves for the negative examples test on SIFT and `DeepBark`. Numbers in the legend indicate how many negative examples were added.

of relevant images retrieved, at 20% recall in the BoW graph we have a *Precision* of nearly 1 for `DeepBark`. Such high precision leads us to believe that having a properly-selected threshold would be sufficiently reliable to classify an image as new with our descriptor. This is also the case for the graph of GV in Figure 3.15 for `DeepBark`, where it shows that we could use a threshold having 50% *Recall* to classify unseen images as not new with almost no error, since the *Precision* is, again, nearly 1. On the contrary, all curves for the SIFT descriptor show a drastic decrease in precision, except for the LR scoring method.

Figure 3.16 is another visualization of the same results for `DeepBark`. But it gives a better idea of the threshold in terms of scores instead of *Recall*. It shows the distributions of matching scores as a histogram for both relevant and non-relevant images. This way, we can see that the mean distribution of the relevant matches score has a significant distance from the mean distribution of the non-relevant matches score. Looking at the subfigure [a], we see that a threshold of 2.02 on the score would allow us to keep more than half the relevant images while discarding almost every non-relevant ones. In addition, subfigure [c] shows that a threshold at 197.40 would keep more than half the relevant images while discarding every non-relevant ones. This indicates that the score distributions of the relevant and non-relevant image matches for `DeepBark` possess enough distance to classify unseen images as new or not, for our tree bark re-identification problem.

For completeness, Figure 3.17 shows the score distributions for the hand-crafted descriptor SIFT. Unlike our descriptor `DeepBark`, we can see that the SIFT descriptor cannot generate sufficient space between the relevant and non-relevant score distributions to classify unseen images effectively, regardless of the scoring method. Even when looking at the best scoring method, the Lowe Ratio (LR) in subfigure [b], we would need to sacrifice approximately 75% of the relevant matches to discard almost every non-relevant match. This would seriously hamper the performance of tree bark re-identification, thus making a SIFT-based approach non-viable in a real scenario.

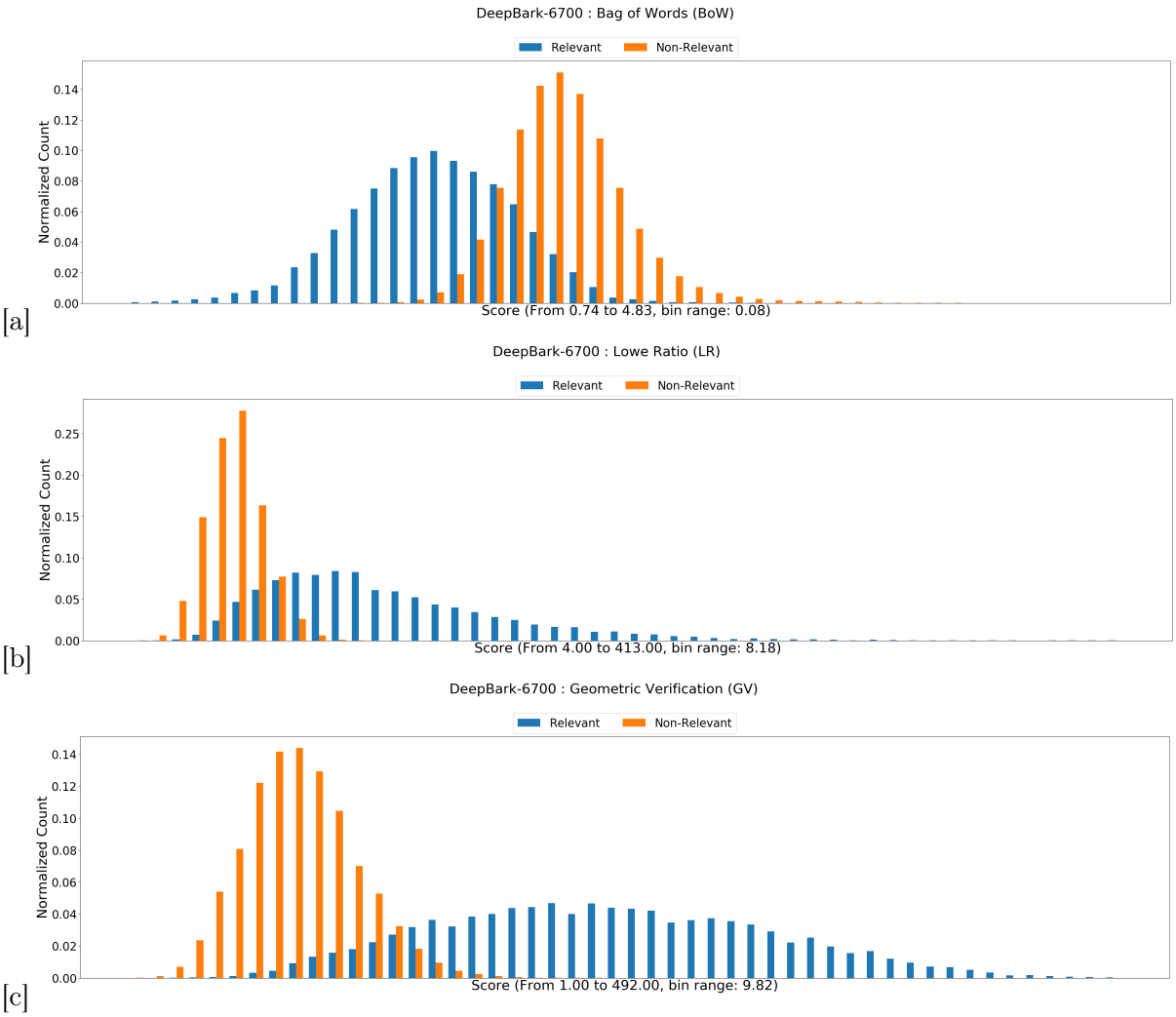


Figure 3.16: Score distributions by relevant and non-relevant matching pairs for `DeepBark` from the negative examples experiment using 6,700 negative examples. Note that in [a], a score refers to a similarity, and thus a lesser score means more similarity. While [b] and [c] refer to the number of feature matches, for which the more, the better.

3.9 Conclusion

With a lot of factors influencing tree bark re-identification experiments, we began this chapter by first laying out details about how we evaluated our experiments. We also made sure to select the best hyper-parameters for every descriptor while using the correct amount of data to train the learned descriptors. With these details and parameters out of the way, we started looking at the efficiency of different types of local descriptors on our tree bark re-identification task. It appeared that using learned descriptors train on bark data was essential to obtain satisfactory results and that hand-crafted descriptors offered poor but constant performance. Then, to obtain a more realistic picture of the performance, we looked at the generalization across tree species to learned that even if it hinders the performance, `DeepBark` showed remarkable

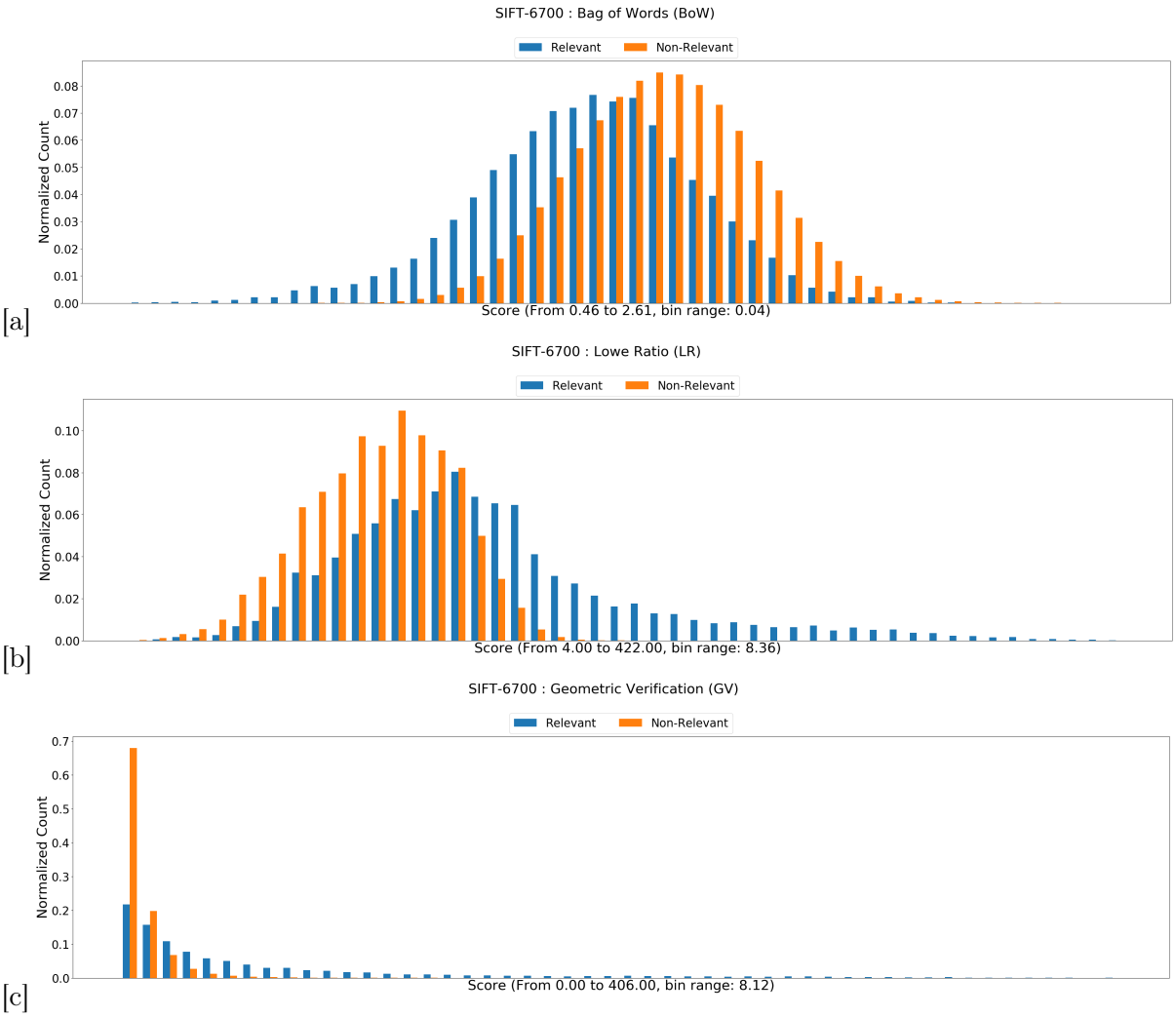


Figure 3.17: Score distributions by relevant and non-relevant matching pairs for SIFT from the negative examples experiment using 6700 negative examples. Note that in [a], a score refers to a similarity, and thus a lesser score means more similarity. While [b] and [c] refer to the number of feature matches, for which the more, the better.

resilience by keeping high precision at more than 50% recall.

Adding to the realistic picture, we did an experiment to evaluate descriptors performance when face with a bigger test set. From this, we determined that the GV scoring method was able to sustain its performance despite adding negative examples and that paired with the excellent performance of `DeepBark`, we could expect satisfactory results against even bigger test sets. However, since the better GV and LR scoring methods are computationally expensive, especially on bigger datasets, we looked to take advantage of the much faster BoW to select potentially matching images. This showed us that we could expect to retain 73.9% of the relevant images by keeping only 200 images among a dataset of 7900 images, in only 0.016 s. Finally, we investigated if we could use a score threshold to decide whether or not an unseen

image possesses a relevant image in our current image database. We discovered that once again, `DeepBark` paired with the GV scoring method showed a great distance between its relevant and non-relevant score distributions, meaning that a single threshold could potentially distinguish between a new unseen or a previously seen image.

Conclusion

In this thesis, we explored bark image re-identification in the challenging context of strong illumination and viewpoint variations. To complete this task, we had to go through essential theories and related works. Then we had to define specific methods, in addition, to collect well-labeled data, to finally design and perform key experiments.

We thus reviewed common techniques such as global and local descriptors to describe images in meaningful ways. In addition to these hand-crafted techniques, we have gone through learned descriptors and the crucial concepts related to them such as convolutional neural networks and metric learning. To put these descriptive methods in context, we looked at the image retrieval task and all the derived formulations. Another essential subject we covered was the multiple metrics available to evaluate image retrieval and their respective advantages and weaknesses. We then concluded this background review with the concept of domain generalization and a thorough survey of the recent research focusing on tree bark images.

After this theoretical review, we started looking at the current state of the available datasets of bark images. Since there were no datasets able to meet our needs, we explained the strict methodology we designed to collect bark data. We then described the resulting Bark-Id dataset, while outlining the pipeline we implemented to transform this Bark-Id dataset into the Bark-Aligned dataset using techniques such as homography. With our data ready, we delved into the training details of our two learned descriptors `DeepBark` and `SqueezeBark`. The last, but no less important part we defined, was how to build a distinctive visual signature for each unique bark surface and the methods we chose to compare such signatures.

Once at this point, we were ready to experiment with our data and methodology to discover crucial knowledge about bark re-identification. To this end, we started by studying the effect of different hyperparameters and the size of the training set. From these two experiments, we asserted that 40% of the data of a single tree species was sufficient to obtain high performance from `DeepBark` and that each descriptor performs differently depending on the image scale and blurriness level. With this knowledge, we were in a position to design an experiment comparing multiple hand-crafted and learned descriptors together. This comparison led us to the conclusion that `DeepBark` was far better than any of the other descriptors with any of the three scoring methods. Analyzing the generalization capacity between tree species of

our two learned descriptors, also showed a clear advantage for `DeepBark` with a small loss of performance compared to the previous experiment. However, to ascertain of the ability of `DeepBark` to perform well under challenging conditions, we planned an experiment where we reduced the size of the available bark images and added true negative bark images of the same size. With 7900 images to compare with every query, `DeepBark` successfully achieves a mAP of 0.872 with a standard deviation of 0.168 using the GV scoring method.

Nonetheless, other considerations about bark re-identification were of vital importance, such as time and open-set problem. In the case of time consideration, we showed that using the BoW method could save a lot of computation without reducing the level of performance by too much. Then, for the open-set problem, we analyzed the results of previous experiments using graphic portraying of the distribution of the score by relevance. Finally, we put forward the idea that a simple threshold could be enough to tackle the problem.

After the collection of our Bark-Id dataset enabling us to tackle the task of bark re-identification and our multiple experiments following a strict methodology, we are confident in re-affirming that:

- We introduced a novel dataset of tree bark pictures with specific markers to infer camera plane transformation, which is also designed for image retrieval.
- We used our novel dataset to train a Deep Learning local feature descriptor adapted for bark images using standard neural network architectures and established the new state of the art performance for bark re-identification.
- We showed that bark surfaces are sufficiently distinct to allow reliable re-identification under a challenging context.
- We demonstrated the necessity to use deep learning methods to achieve satisfactory results, since hand-crafted ones did not.

Also, we can now answer the central question of this thesis, which is:

Is it possible to reliably recognize an individual tree with a visual signature extracted from its bark?

From all of our experiments, the apparent conclusion we have reached is clear: yes, it is possible. However, as usual with research when considering more realistic application scenarios, the more realistic answer lies in a grayer area. This is why further research remains to be done in order to evaluate problems posed by numerous real-world aspects. A glaring problem that realistic scenarios could bear is the encounter of bark belonging to a tree species not present

in the dataset used for training, and this could be worsened by just accumulating too many bark samples to distinguish. Another evident problem real-world applications would bring is the occlusion of tree bark by, for example, snow, leaves, bugs, dirt, etc. Similarly, ripped bark is sure to decrease the available information and thus become a difficulty. Another problem we can foresee is that it will be unlikely that each image taken will either show the same bark surface or not. This means we will have to cope with the fact that two images can partially depict the same bark surface, and that the score indicating a match will vary more.

However, all hope is not lost. Future research can be conducted that should greatly improve on the work done in this thesis. For example, since our descriptor is built using deep learning, it is now common knowledge that more data with more diversity in it improves the performance of deep learning methods. An aspect of our approach that still needs refinement is the choice of a keypoint detector. We chose by default the SIFT difference of Gaussian detector since it is well renown. Yet, it remains unknown which effect other keypoint detectors would have on our approach, and there is likely room for improvement in this aspect. A last crucial point we did not investigate further is the geometric constraint used to remove false matches after the putative matching between two feature lists V. Work on this subject could lead to great improvement over our approach. This is because, in our case, we used the LR and the GV methods, which are now relatively simple compared to the latest state of the art geometric constraint that more efficiently exploits spatial information. Even more ideas could potentially improve the performance of our original approach, but we leave this task to our informed reader.

Appendix A

Appendix A

A.1 ResNet and SqueezeNet

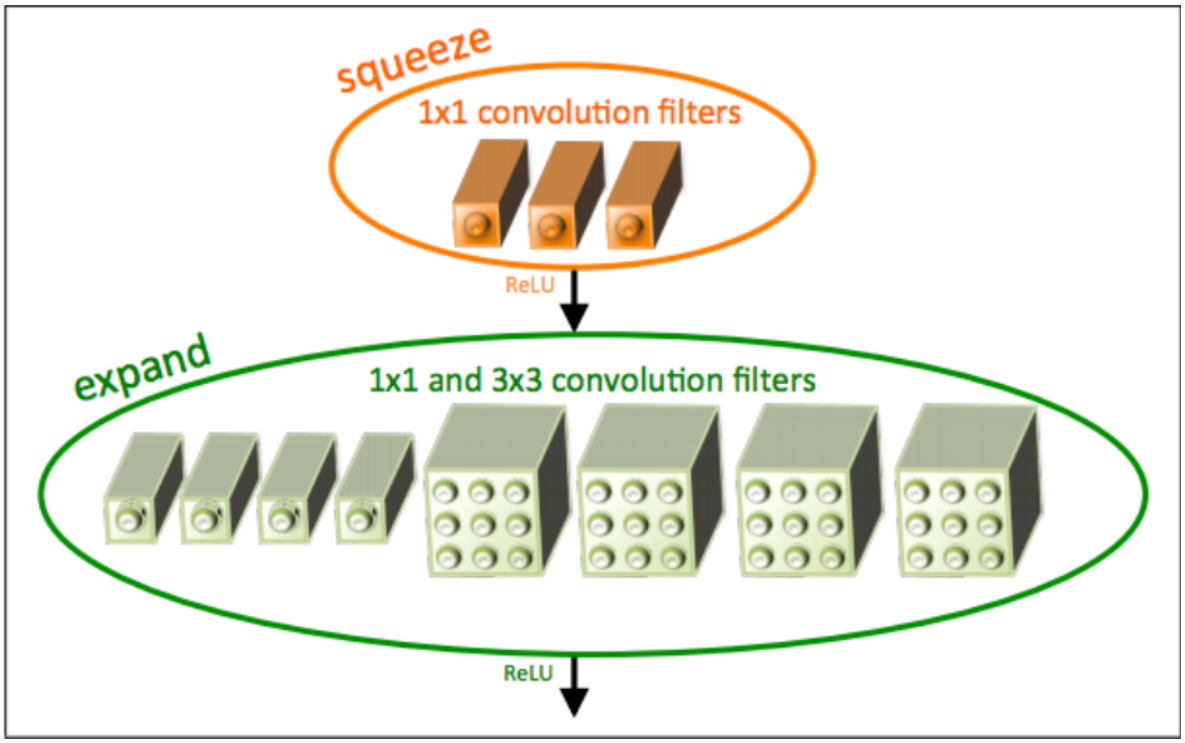


Figure A.1: Fire module architecture, as seen in Iandola et al. (2016) paper. The main idea here is to use the first 1×1 convolution filters to reduce the depth of the feature map before using a combination of 1×1 and more costly 3×3 filters. This depth reduction also reduces the depth of the expand operation filters and thus reduces the number of parameters used.

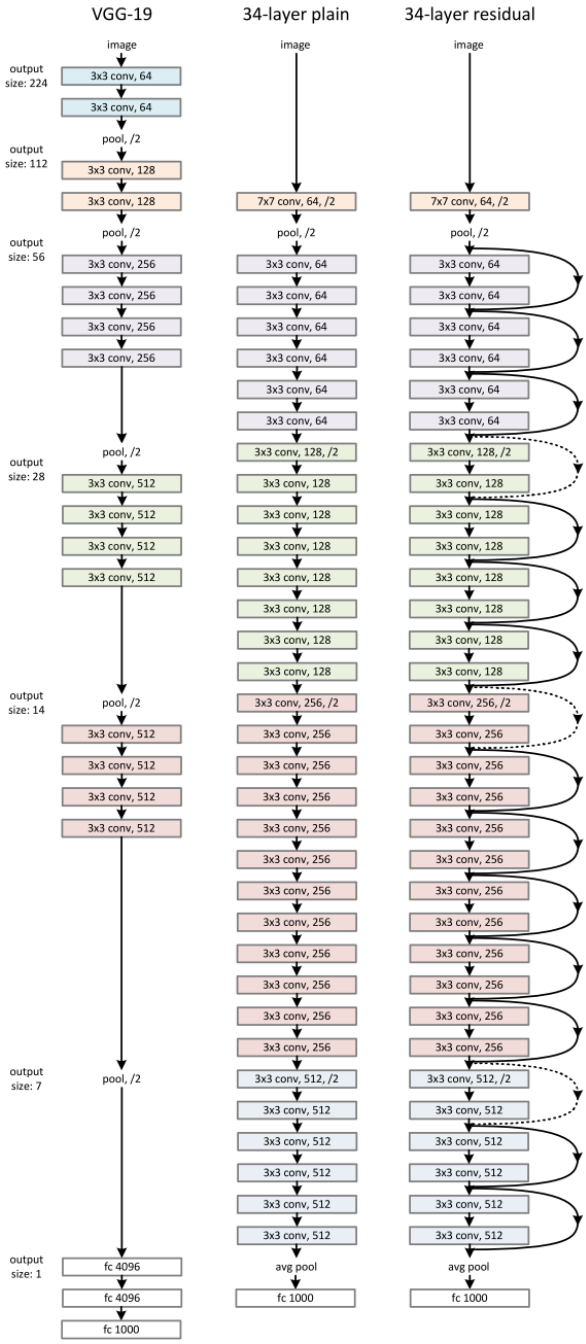


Figure A.2: Figure taken from He et al. (2016) paper. We can see an architectural comparison between the VGG-19 model Simonyan and Zisserman (2015), a plain 34 parameter layers network, and a 34 parameter layers residual network in order from left to right, respectively. The figure also presents the shortcut connections between layers of the residual network. Note that the dotted shortcuts means that the depth dimension is increased from one layer to the other.

Layer Name	Output Size	Params	Stride
Conv1	32x32x64	(7x7x3)x64	2
Maxpool1	16x16x64	None	2
Conv2.1.1	16x16x64	(3x3x64)x64	1
Conv2.1.2	16x16x64	(3x3x64)x64	1
Conv2.2.1	16x16x64	(3x3x64)x64	1
Conv2.2.2	16x16x64	(3x3x64)x64	1
Conv3.1.1	8x8x128	(3x3x64)x128	2
Conv3.1.2	8x8x128	(3x3x128)x128	1
Conv3.2.1	8x8x128	(3x3x128)x128	1
Conv3.2.2	8x8x128	(3x3x128)x128	1
Conv4.1.1	4x4x256	(3x3x128)x256	2
Conv4.1.2	4x4x256	(3x3x256)x256	1
Conv4.2.1	4x4x256	(3x3x256)x256	1
Conv4.2.2	4x4x256	(3x3x256)x256	1
Conv5.1.1	2x2x512	(3x3x256)x512	2
Conv5.1.2	2x2x512	(3x3x512)x512	1
Conv5.2.1	2x2x512	(3x3x512)x512	1
Conv5.2.2	2x2x512	(3x3x512)x512	1
Fully1	128	(2048)x128	None
l_2 norm	128	None	None

Table A.1: DeepBark architecture built with ResNet-18 from Pytorch 0.4.1. Here, the average pool and the fully connected layer of the ResNet-18 are replaced by a fully connected layer followed by l_2 normalization. A batch normalization layer follows each convolution layer. A block of convolution is the detailed architecture of a residual block from ResNet.

Layer Name	Output Size	Params	Stride
Conv1	32x32x64	(3x3x3)x64	2
Maxpool1	16x16x64	None	2
Conv2.1	16x16x16	(1x1x64)x16	1
Conv2.2.1	16x16x64	(1x1x16)x64	1
Conv2.2.2	16x16x64	(3x3x16)x64	1
Conv3.1	16x16x16	(1x1x128)x16	1
Conv3.2.1	16x16x64	(1x1x16)x64	1
Conv3.2.2	16x16x64	(3x3x16)x64	1
Maxpool2	8x8x128	None	2
Conv4.1	8x8x32	(1x1x128)x32	1
Conv4.2.1	8x8x128	(1x1x32)x128	1
Conv4.2.2	8x8x128	(3x3x32)x128	1
Conv5.1	8x8x32	(1x1x256)x32	1
Conv5.2.1	8x8x128	(1x1x32)x128	1
Conv5.2.2	8x8x128	(3x3x32)x128	1
Maxpool3	4x4x256	None	2
Conv6.1	4x4x48	(1x1x256)x48	1
Conv6.2.1	4x4x192	(1x1x48)x192	1
Conv6.2.2	4x4x192	(3x3x48)x192	1
Conv7.1	4x4x48	(1x1x384)x48	1
Conv7.2.1	4x4x192	(1x1x48)x192	1
Conv7.2.2	4x4x192	(3x3x48)x192	1
Conv8.1	4x4x64	(1x1x384)x64	1
Conv8.2.1	4x4x256	(1x1x64)x256	1
Conv8.2.2	4x4x256	(3x3x64)x256	1
Conv9.1	4x4x64	(1x1x512)x64	1
Conv9.2.1	4x4x256	(1x1x64)x256	1
Conv9.2.2	4x4x256	(3x3x64)x256	1
Maxpool4	2x2x512	None	2
Fully1	128	(2048)x128	None
l_2 norm	128	None	None

Table A.2: SqueezeBark architecture built with SqueezeNet 1.1 from Pytorch 0.4.1. Here, the last convolutional layer and the average pooling is replaced by a max pool layer with a fully connected layer and followed by l_2 normalization. A block of convolution is the detailed architecture of a Fire module from SqueezeNet.

A.2 Supplementary Results

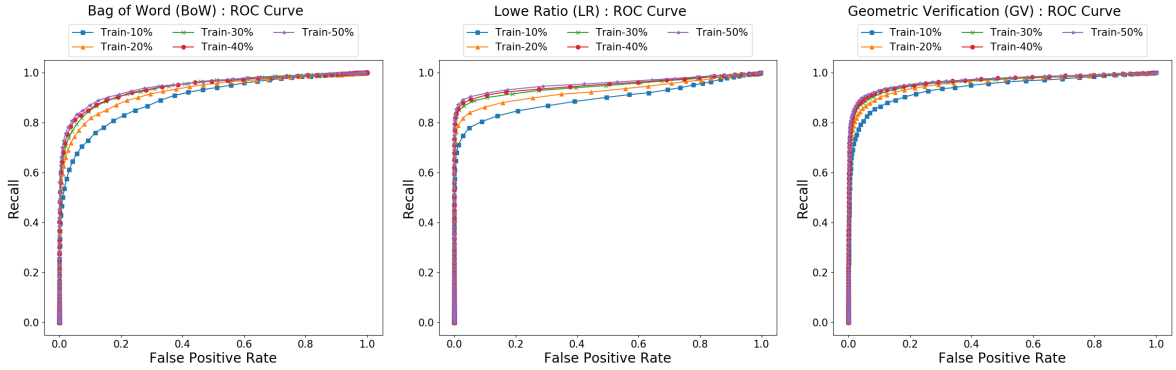


Figure A.3: ROC Curves for all training set sizes using RP bark. Supplementary results of section 3.3.

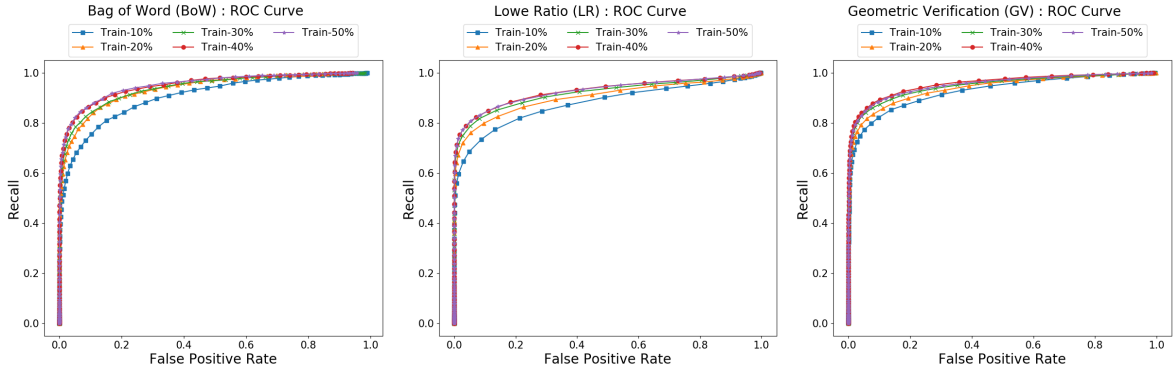


Figure A.4: ROC Curves for all training set sizes using EL bark. Supplementary results of section 3.3.

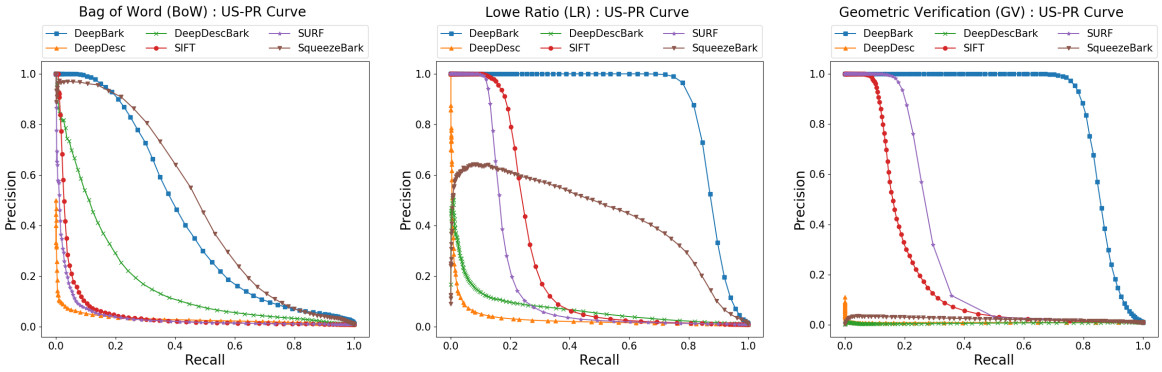


Figure A.5: US-PR Curve for all descriptors and scoring methods tested on 50% of RP and EL. Learned descriptors were trained on the remaining 50% of bark. Each of the 1200 images of the test set is used as a query. No extra negative examples were added. Supplementary results of section 3.4.

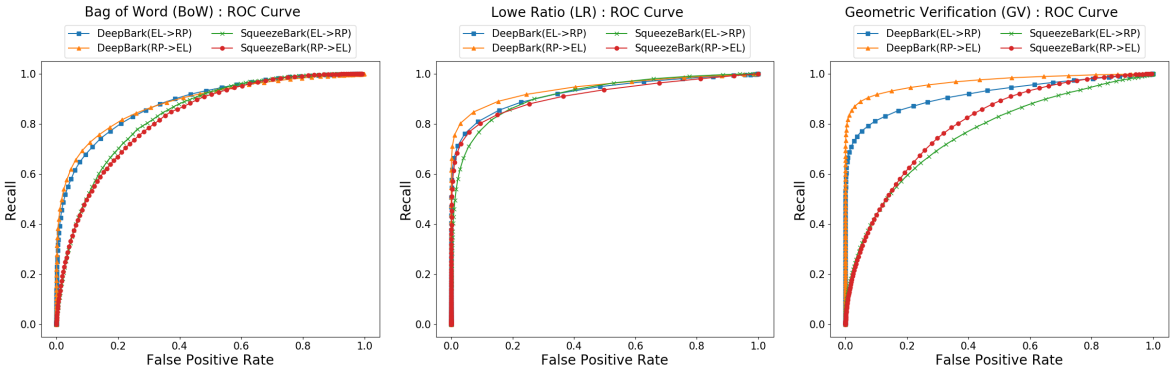


Figure A.6: ROC Curves for the generalization test. The arrow -> indicates the generalization direction (trained on -> tested on). Supplementary results of section 3.5.

Bibliography

Pablo Fernendez Alcantarilla, Adrien Bartoli, and Andrew J. Davison. Kaze features. In *ECCV*, pages 214–227, 2012.

Relja Arandjelovic. Three things everyone should know to improve object retrieval. In *CVPR*, pages 2911–2918, 2012.

Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(6):1437–1451, jun 2018.

Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *CVIU*, volume 110, pages 404–417. 2006.

L. J. Blaanco, C. M. Travieso, J. M. Quinteiro, P. V. Hernandez, M. K. Dutta, and A. Singh. A bark recognition algorithm for plant classification using a least square support vector machine. In *2016 Ninth International Conference on Contemporary Computing (IC3)*, pages 1–5, Aug 2016.

Safia Boudra, Itheri Yahiaoui, and Ali Behloul. A comparison of multi-scale local binary pattern variants for bark image retrieval. In *ACIVS*, pages 764–775, 2015.

Adriano Bressane, Jose Arnaldo Roveda, and Antonio Martins. Statistical analysis of texture in trunk images for biometric identification of tree species. *Environmental monitoring and assessment*, 187:4400, 04 2015.

Matthew Brown, Gang Hua, and Simon Winder. Discriminative learning of local image descriptors. *Transactions on Pattern Analysis and Machine Intelligence*, 33(1):43–57, jan 2011.

Ozden Cakici, Harry Groenevelt, and Abraham Seidmann. Using rfid for the management of pharmaceutical inventory-system optimization and shrinkage control. *Decision Support Systems*, 51:842–852, 11 2011.

Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *ECCV*, pages 778–792, 2010.

Mathieu Carpentier, Philippe Giguère, and Jonathan Gaudreault. Tree species identification from bark images using convolutional neural networks. *IROS*, mar 2018.

Sumit Chopra, Raia Hadsell, and LeCun Y. Learning a similiarty metric discriminatively, with application to face verification. *CVPR*, pages 349–356, 2005.

Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, jun 2008.

Mark Cummins and Paul Newman. Highly scalable appearance-only slam - fab-map 2.0. In *Robotics: Science and Systems*, pages 1828–1833, jun 2009.

Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2), 2008. ISSN 0360-0300. doi: 10.1145/1348246.1348248.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. *CVPRW*, pages 337–349, 2018.

C. Fang, Y. Xu, and D. N. Rockmore. Unbiased metric learning: On the utilization of multiple datasets and web images for softening bias. In *ICCV*, pages 1657–1664, Dec 2013.

Stefan Fiel and Robert Sablatnig. Automated identification of tree species from images of the bark , leaves or needles. pages 67–74, feb 2010.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *J. Mach. Learn. Res.*, 17(1):2096–2030, jan 2016. ISSN 1532-4435.

Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. Multi-view stereo for community photo collections. In *ICCV*, pages 1–8, 2007.

Doug Gray, Shane Brennan, and Hai Tao. Evaluating appearance models for recognition, reacquisition, and tracking. In *International Workshop on Performance Evaluation for Tracking and Surveillance*, 2007.

Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *ECCV*, 2016.

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, pages 1735–1742, 2006.

K. He and J. Sun. Convolutional neural networks at constrained time cost. In *CVPR*, pages 5353–5360, June 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, volume 19, pages 770–778, jun 2016.

Thomas Hellstrom, Par Larkeryd, Tomas Nordfjell, and Ola Ringdahl. Autonomous forest vehicles: Historic, envisioned and state-of-the-art. *International Journal of Forest Engineering*, 20(1):31–38, 2009.

Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *CoRR*, abs/1703.07737, 2017.

Ville Hinkka. Challenges for building rfid tracking systems across the whole supply chain. *International Journal of RF Technologies Research and Applications*, 3:201–218, 05 2012.

Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

Zhi-Kai Huang, Zhong-Hua Quan, and Ji-Xiang Du. Bark classification based on contourlet filter features using rbpnn. In *Intelligent Computing*, pages 1121–1126, 2006.

Forrest Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *CoRR*, abs/1602.07360, 02 2016.

Unnat Jain, Vinay P. Namboodiri, and Gaurav Pandey. Compact environment-invariant codes for robust visual place recognition. In *CRV*, pages 40–47, 2017.

Simardeep Kaur and Dr. Vijay Kumar Banga. Content based image retrieval: Survey and comparison between rgb and hsv model. *IJETT*, 4:575–579, April 2013.

Aditya Khosla, Tinghui Zhou, Tomasz Malisiewicz, Alexei A. Efros, and Antonio Torralba. Undoing the damage of dataset bias. In *ECCV*, pages 158–171, 2012.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Brendan F. Klare, Ben Klein, Emma Taborsky, Austin Blanton, Jordan Cheney, Kristen Allen, Patrick Grother, Alan Mah, and Anil K. Jain. Pushing the frontiers of unconstrained face detection and recognition: Iarpa janus benchmark a. In *CVPR*, June 2015.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105. 2012.

Gary B. Huang Erik Learned-Miller. Labeled faces in the wild: Updates and new reporting procedures. Technical Report UM-CS-2014-003, University of Massachusetts, Amherst, May 2014.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

Chao Li, Xiaokong Ma, Bing Jiang, Xiangang Li, Xuewei Zhang, Xiao Liu, Ying Cao, Ajay Kannan, and Zhenyao Zhu. Deep speaker: an end-to-end neural speaker embedding system. *CoRR*, abs/1705.02304, 2017a.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. Deeper, broader and artier domain generalization. *ICCV*, pages 5543–5551, 2017b.

W. Li, R. Zhao, T. Xiao, and X. Wang. Deepreid: Deep filter pairing neural network for person re-identification. In *CVPR*, pages 152–159, June 2014.

Yunpeng Li, Noah Snavely, and Daniel P Huttenlocher. Location recognition using prioritized feature matching. *ECCV*, pages 791–804, 2010.

Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition. *CVPR*, pages 6738–6746, 2017.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004.

S. Lowry, N. Sunderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford. Visual place recognition: A survey. *IEEE Transactions on Robotics*, 32(1):1–19, Feb 2016. ISSN 1941-0468. doi: 10.1109/TRO.2015.2496823.

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *NIPS*, pages 2627–2635. 2014.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. 2008.

Zuheng Ming, Joseph Chazalon, Muhammad Muzzamil Luqman, Muriel Visani, and Jean Christophe Burie. Simple triplet loss based on intra/inter-class metric learning for face verification. *ICCVW*, pages 1656–1664, 2018.

D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley and Sons, 2003.

Saeid Motiian, Marco Piccirilli, Donald A. Adjeroh, and Gianfranco Doretto. Unified deep supervised domain adaptation and generalization. In *ICCV*, Oct 2017.

Fatma Mtibaa and Amin Chaabane. Forestry wood supply chain information system using rfid technology. *IIE Annual Conference and Expo*, pages 1562–1571, 01 2014.

Krikamol Muandet, David Balduzzi, and Bernhard Scholkopf. Domain generalization via invariant feature representation. In *ICML*, volume 28, pages 10–18, Jun 2013.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

S Neetu Sharma, S Paresh Rawat, and S Jaikaran Singh. Efficient cbir using color histogram processing. *SIPIJ*, 2(1):94–112, mar 2011.

Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. *BMVC*, pages 41.1–41.12, 2015.

M. Paulin, M. Douze, Z. Harchaoui, J. Mairal, F. Perronin, and C. Schmid. Local convolutional features with unsupervised training for image retrieval. In *ICCV*, pages 91–99, 2015.

Filip Radenovic, Giorgos Tolias, and Ondrej Chum. Deep shape matching. In *ECCV*, September 2018.

F. T. Ramos, J. Nieto, and H. F. Durrant-Whyte. Recognising and modelling landmarks to close loops in outdoor slam. In *ICRA*, pages 2036–2041, April 2007.

Ignacio Rocco, Mircea Cimpoi, Relja Arandjelović, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Neighbourhood consensus networks. In *NIPS*, pages 1651–1662. 2018.

Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *ECCV*, pages 430–443, 2006.

Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, pages 2564–2571, 2011.

Sergey P. Sannikov and Vladimir V. Pobedinskiy. Automated system for natural resources management. *CEUR-WS*, May 2018.

Johannes L. Schonberger, Hans Hardmeier, Torsten Sattler, and Marc Pollefeys. Comparative evaluation of hand-crafted and learned local features. In *CVPR*, pages 6959–6968, jul 2017.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, pages 815–823, jun 2015.

Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In *ICCV*, pages 118–126, dec 2015.

K. Simonyan, A. Vedaldi, and A. Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8): 1573–1585, 2014.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Usha Sinha and Hooshang Kangarloo. Principal component analysis for content-based image retrieval. *Radiographics : a review publication of the Radiological Society of North America, Inc*, 22(5):1271–89, 2002.

J. Sivic and A. Zisserman. Video google: a text retrieval approach to object matching in videos. In *ICCV*, number Iccv, pages 1470–1477 vol.2, 2003.

Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stanley T. Birchfield. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. *IROS*, pages 4241–4247, 2017.

Noah Snavely, Steven M. Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, nov 2008.

Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. *NIPS*, (Nips):1857–1865,, 2016.

M. Sulc and J. Matas. Kernel-mapped histograms of multi-scale lbps for tree bark recognition. In *IVCNZ*, pages 82–87, Nov 2013.

Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *NIPS*, pages 1988–1996. 2014a.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10 000 classes. *CVPR*, pages 1891–1898, 2014b.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deeply learned face representations are sparse, selective, and robust. In *CVPR*, pages 2892–2900, 2015.

Yi Sun, Xiaogang Wang, and Xiaoou Tang. Hybrid deep learning for face verification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):1997–2009, 2016.

Niko Sunderhauf, Sareh Shirazi, Adam Jacobson, Feras Dayoub, Edward Pepperell, Ben Upcroft, and Michael Milford. Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. *Robotics: Science and Systems*, 2015.

Shamik Sural, Gang Qian, and Sakti Pramanik. Segmentation and histogram generation using the hsv color space for image retrieval. In *ICIP*, volume 2, pages II–589–II–592, 2002.

Matic Svab. Computer-vision-based tree trunk recognition, 2014.

Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Web-scale training for face identification. In *CVPR*, pages 2746–2754, 2015.

Yaniv Taigman, Ming Yang, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

Tomasz Trzcinski, Mario Christoudias, Vincent Lepetit, and Pascal Fua. Learning image descriptors with the boosting-trick. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 269–277. Curran Associates, Inc., 2012.

Panu Turcot and David G. Lowe. Better matching with fewer features: The selection of useful features in large database recognition problems. *ICCV*, pages 2109–2116, 2009.

Deepak Viswanathan. Features from accelerated segment test (fast). 2011.

Ji Wan, Dayong Wang, Steven Chu Hong Hoi, Pengcheng Wu, Jianke Zhu, Yongdong Zhang, and Jintao Li. Deep learning for content-based image retrieval. In *ACM International Conference on Multimedia*, pages 157–166, 2014.

Feng Wang, Xiang Xiang, Jian Cheng, and Alan Loddon Yuille. Normface: L2 hypersphere embedding for face verification. In *ACM International Conference on Multimedia*, pages 1041–1049, 2017a.

Feng Wang, Jian Cheng, Weiyang Liu, and Haijun Liu. Additive margin softmax for face verification. *Signal Processing Letters*, 25(7):926–930, July 2018a.

Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. *CVPR*, pages 5265–5274, 2018b.

Jian Wang, Feng Zhou, Shilei Wen, Xiao Liu, and Yuanqing Lin. Deep metric learning with angular loss. *ICCV*, pages 2612–2620, 2017b.

Mei Wang and Weihong Deng. Deep face recognition: A survey. *CoRR*, abs/1804.06655, 2018.

Lior Wolf, Tal Hassner, and Itay Maoz. Face recognition in unconstrained videos with matched background similarity. In *CVPR*, pages 529–534. IEEE Computer Society, 2011.

Xufeng Han, Thomas Leung, Yangqing Jia, Rahul Sukthankar, and Alexander C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *CVPR*, volume 07-12-June, pages 3279–3286, jun 2015.

Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. Learning face representation from scratch. *ArXiv*, abs/1411.7923, 2014.

Kwang Moo Yi, Eduard Trulls, Vincent Lepetit, and Pascal Fua. Lift: Learned invariant feature transform. In *ECCV*, 2016.

Jun Zhang and Lei Ye. Ranking method for optimizing precision/recall of content-based image retrieval. *Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing in Conjunction with the UIC'09 and ATC'09 Conferences*, pages 356–361, 2009.

Linguang Zhang and Szymon Rusinkiewicz. Learning to detect features in texture images. In *CVPR*, pages 6325–6333, jun 2018.

Linguang Zhang, Adam Finkelstein, and Szymon Rusinkiewicz. High-precision localization using ground texture. *ICRA*, pages 6381–6387, 2017.

Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. *ECCV*, pages 94–108, 2014.

Liang Zheng, Liyue Shen, Lu Tian, Shengjin Wang, Jingdong Wang, and Qi Tian. Scalable person re-identification: A benchmark. In *Computer Vision, IEEE International Conference on*, 2015.

Liang Zheng, Zhi Bie, Yifan Sun, Jingdong Wang, Chi Su, Shengjin Wang, and Qi Tian. Mars: A video benchmark for large-scale person re-identification. In *ECCV*, 2016a.

Liang Zheng, Yi Yang, and Qi Tian. Sift meets cnn: A decade survey of instance retrieval. *Transactions on Pattern Analysis and Machine Intelligence*, PP, 08 2016b.

Zhedong Zheng, Liang Zheng, and Yi Yang. A discriminatively learned cnn embedding for person reidentification. *ACM Trans. Multimedia Comput. Commun. Appl.*, 14(1):13:1–13:20, dec 2017.

Zheru Chi, Li Houqiang, and Wang Chao. Plant species recognition based on bark patterns using novel gabor filter banks. In *NIPS*, volume 2, pages 1035–1038, Dec 2003.