Universitat Autònoma
de Barcelona

Geometric Computer Vision Techniques for Scene Reconstruction

A dissertation submitted by **Edgar Riba Pi** at Universitat Autònoma de Barcelonato fulfil the degree of **Doctor of Philosophy**.

Bellaterra, January 14, 2021

Director	**Dr. Daniel Ponsa** Universitat Autònoma de Barcelona Dept. Ciències de la computació & Centre de Visió per Computador
Thesis committee	**Dr. Cecilio Angulo** Universitat Politècnica de Catalunya Dept. d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial
	Dr. Felipe Lumbreras Universitat Autònoma de Barcelona Dept. Ciències de la computació & Centre de Visió per Computador
	Dr. Xavier Binefa Universitat Pompeu Fabra Departament de Tecnologies de la Informació i les Comunicacions

This document was typeset by the author using LaTeX 2ε.

ISBN: 978-84-945373-1-8

Printed by Ediciones Gráficas Rey, S.L.

Acknowledgements

Everything started while I was doing my internship for my masters thesis in Paris at Aldebaran Robotics and a colleague sent to me the link of opening PhD positions at the CVC. I initially doubted whether to apply since I always considered myself to have a more engineer background and at that time was not really aware about the consequences of starting a PhD and how would affect to my career in the future.

I am an outgoing person which like challenges. This is the reason why I decided to send an email to just ask information about the different open positions and arrange a call meeting with Dani. That was my first contact with Dani, which later has become my PhD supervisor during this last 5 years. I would like to thank him for giving me this opportunity and the support for all the collaborations and internships these years.

My stay at the CVC was very productive and gave me the opportunity to meet amazing people such as Felipe, to whom I want to thank for the enormous number of hours of discussion about redundant questions for geometry. The two Paus, Pep, Albert, Xavi and Arnau for the good times spent in our cluster office in the basement; Lorena, Iris, Xesco, Marc, Felipe, Bojana and the rest of the CARLA team for the nice breakfasts and barbeque; Alexandra, Txell and the administration people to help and listen me every time I wanted to organise hackathons or promote any crazy open source initiative.

Probably I am missing a long list of people that I met during these years since the number of collaborations and contacts I did has been remarkable. However, I would specially like to thank Vassilis and Krystian to give me the opportunity to start collaborating with them and guide me during my initial stage of the PhD. The rest of the Imperial College people, Axel, Adri, Joan, Marina, Irene, Pablo, Yurun for such good times in London and China.

My tight collaborations with the open source community brought me also the opportunity to meet brilliant people like Gary and Ethan; Vincent, Vadim, Stefano, Reza, Mona, Wes, Prassanna from OpenCV and Arraiy. During my stay in the US, Camilo, Karsten, Ali, Matilda and Anguelos for such good times and never ending discussions about open source and computer science. I would also specially thank to Francesc, Luis, Antonio, Albert, Jordi and the rest of people from Kognia Sports and IRI for helping me during my last stage of the PhD.

Finally, to Gore and Daga for the initial support to this long trip and bring inspiration and meaning to my work. A strong mention to my family for trying to understand my going and coming back trips, and not to forgive the ones that could not make it to see the final result of this thesis.

Abstract

From the early stages of Computer Vision, scene reconstruction has been one of the most studied topics leading to a wide variety of new discoveries and applications. Object grasping and manipulation, localization and mapping, or even visual effect generation are different examples of applications in which scene reconstruction has taken an important role for industries such as robotics, factory automation, or audio visual production. However, scene reconstruction is an extensive topic that can be approached in many different ways with already existing solutions that effectively work in controlled environments. Formally, the problem of scene reconstruction can be formulated as a sequence of independent processes which compose a pipeline. In this thesis, we analyse some parts of the reconstruction pipeline from which we contribute with novel methods using Convolutional Neural Networks (CNN) proposing innovative solutions that consider the optimisation of the methods in an end-to-end fashion. First, we review the state of the art of classical local features detectors and descriptors and contribute with two novel methods that inherently improve pre-existing solutions in the scene reconstruction pipeline.

It is a fact that computer science and software engineering are two fields that usually go hand in hand and evolve according to mutual needs making easier the design of complex and efficient algorithms. For this reason, we contribute with Kornia, a library specifically designed to work with classical computer vision techniques along with deep neural networks. In essence, we created a framework that eases the design of complex pipelines for computer vision algorithms so that can be included within neural networks and be used to backpropagate gradients throw a common optimisation framework. Finally, in the last chapter of this thesis we develop the aforementioned concept of designing end-to-end systems with classical projective geometry. Thus, we contribute with a solution to the problem of synthetic view generation by hallucinating novel views from high deformable cloths objects using a geometry aware end-to-end system. To summarize, in this thesis we demonstrate that with a proper design that combine classical geometric computer vision methods with deep learning techniques can lead to improve pre-existing solutions for the problem of scene reconstruction.

Key words: *Computer Vision, Scene Reconstruction, Local Features, Differentiable Operators, Views Synthesis Generation*

Resumen

Desde los inicios de la Visión por Computador, la reconstrucción de escenas ha sido uno de los temas más estudiados que ha llevado a una amplia variedad de nuevos descubrimientos y aplicaciones. La manipulación de objetos, la localización y mapeo, o incluso la generación de efectos visuales son diferentes ejemplos de aplicaciones en las que la reconstrucción de escenas ha tomado un papel importante para industrias como la robótica, la automatización de fábricas o la producción audiovisual. Sin embargo, la reconstrucción de escenas es un tema extenso que se puede abordar de muchas formas diferentes con soluciones ya existentes que funcionan de manera efectiva en entornos controlados. Formalmente, el problema de la reconstrucción de escenas puede formularse como una secuencia de procesos independientes. En esta tesis, analizamos algunas partes del pipeline de reconstrucción a partir de las cuales contribuimos con métodos novedosos utilizando Redes Neuronales Convolucionales (CNN) proponiendo soluciones innovadoras que consideran la optimización de los métodos de forma end-to-end. En primer lugar, revisamos el estado del arte de los detectores y descriptores de características locales clásicas y contribuimos con dos métodos novedosos que mejoran las soluciones preexistentes en el problema de reconstrucción de escenas.

Es un hecho que la informática y la ingeniería de software son dos campos que suelen ir de la mano y evolucionan según necesidades mutuas facilitando el diseño de algoritmos complejos y eficientes. Por esta razón, contribuimos con Kornia, una libreria diseñada específicamente para trabajar con técnicas clásicas de visión por computadora conjuntamente con redes neuronales profundas. En esencia, creamos un marco que facilita el diseño de procesos complejos para algoritmos de visión por computadora para que puedan incluirse dentro de las redes neuronales y usarse para propagar gradientes dentro de un marco de optimización común. Finalmente, en el último capítulo de esta tesis desarrollamos el concepto antes mencionado de diseñar sistemas de forma conjunta con geometría proyectiva clásica. Por lo tanto, proponemos una solución al problema de la generación de vistas sintéticas mediante la alucinación de vistas novedosas de objetos altamente deformables utilizando un sistema conjunto con la geometría de la escena. En resumen, en esta tesis demostramos que con un diseño adecuado que combine los métodos clásicos de visión geométrica por computador con técnicas de aprendizaje profundo puede conducir a mejores soluciones para el problema de la reconstrucción de escenas.

Palabras clave: *Visión por Computador, Reconstrucción de Escenas, Características Locales, Operadores Diferenciables, Generación de Vistas Sintéticas*

Resum

Des dels inicis de la Visió per Computador, la reconstrucció d'escenes ha estat un dels temes més estudiats que ha portat a una àmplia varietat de nous descobriments i aplicacions. La manipulació d'objectes, la localització i mapeig, o fins i tot la generació d'efectes visuals són diferents exemples d'aplicacions en les que la reconstrucció d'escenes ha pres un paper important per a indústries com la robòtica, l'automatització de fàbriques o la producció audiovisual. No obstant això, la reconstrucció d'escenes és un tema extens que es pot abordar de moltes formes diferents amb solucions ja existents que funcionen de manera efectiva en entorns controlats. Formalment, el problema de la reconstrucció d'escenes pot formular-se com una seqüència de processos independents. En aquesta tesi, analitzem algunes parts de la seqüència de reconstrucció a partir de les quals contribuïm amb nous mètodes que fan servir Convolutional Neural Networks (CNN), proposant solucions innovadores que consideren l'optimització dels mètodes de forma conjunta. En primer lloc, revisem l'estat de l'art dels detectors i descriptors de característiques local clàssiques i contribuïm amb dos mètodes nous que milloren intrínsecament les solucions preexistents al problema de reconstrucció d'escenes.

És un fet que la informàtica i l'enginyeria del software són dos camps que solen anar de la mà i evolucionen segons necessitats mútues facilitant el disseny d'algoritmes complexos i eficients. Per aquesta raó, contribuïm amb Kornia, un llibreria dissenyada específicament per treballar amb tècniques clàssiques de visió per computador conjuntament amb xarxes neuronals profundes. En essència, hem creat un marc que facilita el disseny de processos complexes per algoritmes de visió per computador perquè es puguin incloure dins les xarxes neuronals i usar-se per propagar gradients dins d'un marc d'optimització comú. Finalment, en l'últim capítol d'aquesta tesi desenvolupem el concepte abans esmentat de dissenyar sistemes de forma conjunta amb geometria projectiva clàssica. Per tant, proposem una solució a el problema de la generació de vistes sintètiques mitjançant l'al·lucinació de vistes noves d'objectes altament deformables utilitzant un sistema conjunt amb la geometria de l'escena. En resum, en aquesta tesi demostrem que amb un disseny adequat que combini els mètodes clàssics de visió geomètrica per computador amb tècniques d'aprenentatge profund pot conduir a la millora de solucions per al problema de la reconstrucció d'escenes.

Paraules clau: *Visió per Computador, Reconstrucció d'Escenes, Característiques Locals, Operadors Diferenciables, Generació de Vistes Sintètiques*

Contents

List of Figures

List of Tables

1 Introduction

Every single organism in an ecosystem has the property to have a life-form. Humans, as many of the other species organisms, are part of the biological evolution. Darwin states in [1] that all species of organisms arise and develop through the natural selection of small, inherited variations that increase the individuals ability to compete, survive, and reproduce. As individual organisms, we are responsible to observe, plan and execute a series of actions that will determine the interaction with our surrounding environment. The capability of communicating with other individuals, move from one place to the other, or search and grasp objects are just examples of primitive actions that help us to interact with our environment. However, in order to perform many of those primitive actions it is needed to have and use a perception system in order to gather and interpret sensory information. As humans, our perception system can be extremely complex and composed by several layers of sensory cells that respond to a specific type of physical stimulus which eventually will be processed by our brains and used to perform specific actions.

The human perception system evolved such in a way that reconstructs our environment and intrinsically creates an internal representation to assure this interaction and eventually make us as specie to survive. In this direction, the main motivation of this work is to study the different mechanisms that the human perception system uses to reconstruct the environment and bridge it with computer vision. Precisely, in this thesis we first study the problem of scene reconstruction - understood as a sequential pipeline, we review the existing solutions and propose new methods for two of the main tasks in classical multi-view scene reconstruction: key-point detection and key-point description. With this aim, in this thesis we have develop a new Python library (Kornia) that fills the gap between classical computer vision systems and current deep learning based approaches. Using this library a multi-camera scene reconstruction systems has been built, which is the last contribution in this thesis.

Figure 1.1 – **Result obtained from a complete scene reconstruction pipeline.** Reconstruction obtained using *Open Drone Maps* an open source photogrammetry toolkit to process aerial imagery into maps and 3D models. As can be seen, the global structure of the scene is very accurate after the combination of several computer vision techniques and multi-view geometry.

1.1 Scene Reconstruction

In the previous section we introduced the concept of Computer Vision and we associated it to the process to reconstruct a scene similarly as the human perception system does. However, we would like first ask ourselves the following question - *How much related is Computer Vision to the task of scene reconstruction ?* To answer this question, we should first define the problem. Formally, in computer vision and computer graphics, scene reconstruction or 3D reconstruction is the process of capturing the shape and appearance of real object, accomplished either by active or passive methods [2]. In figure 1.1 we can appreciate the resulting reconstruction of a scene combining several computer vision techniques and multi-view geometry.

There is an extensive literature about 3D reconstruction describing the different methodologies and algorithms to solve the entire problem. Nevertheless, the scope of this thesis is to not a give an exhaustive review of the state of the art for all the existing approaches, yet we emphasize specific tasks in chapters 2, 3 and 4. In the next sections, we aim to describe some of the most known approximations for

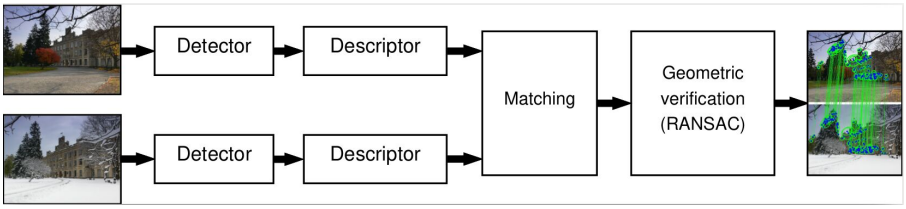


Figure 1.2 – **Wide baseline stereo matching algorithm.** The diagram of the commonly used wide baseline stereo matching algorithm [3] that shows the different sub-tasks in the entire pipeline.

solving the 3D reconstruction problem based on the following approaches: camera pose estimation for two-view image matching, stereo systems, range imaging with RGB-D cameras, and monocular depth estimation for multi-view geometry.

1.1.1 Camera Pose Estimation

Choosing the right features to compute the relative pose between two cameras is a crucial task to estimate the depth information of a scene. This can be achieved by the classical process of establishing correspondences between pixels and/or between images and estimating the geometric relation between the cameras. Formally, this process is described as the Wide Baseline Stereo (WBS) algorithm [4] that later can be used as a building block for the application of 3D reconstruction. In figure 1.2 we can see a visual example with the result of the WBS algorithm for the case to align two-images from a different view using the described classical features matching pipeline.

As we will see in chapter 2, in this thesis we give a strong focus on the detection and description of local features in the context of the WBS algorithm. Thus, in the first place - *What is a local feature?* As stated in [5] a Local Feature can be described as an image pattern which differs from its intermediate neighborhood. Considering that the most common image properties are intensity, color and texture; Local Features can take different forms such as points, edgels or small image patches which can be accompanied by a descriptor vector. Traditionally, local features have been defined using hand-crafted methods [6, 7] that allowed to detect singular elements from a single scene across different views in order robustly align the images. However, similar to [8, 9] in this thesis in chapter 2 we review and propose new methods for detecting and describing local features based on Convolutional Neural Networks (CNN).

Figure 1.3 – **Structure from Motion example.** This figure shows an example of the result of a Structure from Motion pipeline which reconstructs an entire scene given a set of unordered images. The image is from *Bundler* [10].

Structure from Motion

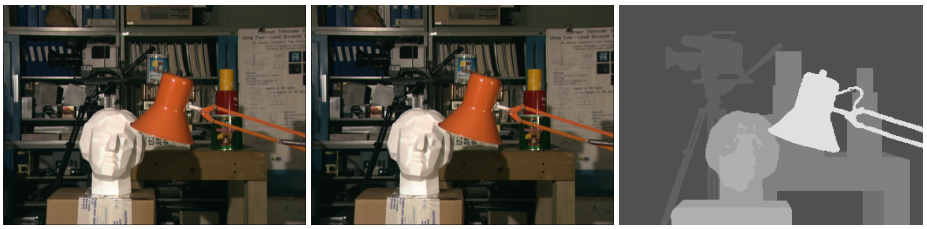
One of the most popular applications where the camera pose estimation approach can be directly applied is Structure from Motion (SfM) [11]. SfM can be described as the technique associated to the field of photogrammetry that tries to estimate a three dimensional representation from two or more images based on the local motion between the different view information (see figure 1.3). The main principle for SfM is based on the motion parallax theory that using the difference between the displacement of the different views and from the depth information will be used to estimate an accurate 3D representation of the world. Finding the structure from motion it is directly related to stereo vision since in both cases it is needed to have a geometric relation between the different image views.

1.1.2 Depth Maps Estimation

In the previous section we introduced how to solve the 3D reconstruction of a scene using as a starting point the estimation of the relative pose between the cameras in order to obtain a sparse reconstruction of the scene. In this section we give a general overview for estimating dense depth maps using stereo systems, range imaging for RGB-D cameras and monocular depth estimation using deep learning methods for the case of multi-camera views.

Stereo systems

In classical computer vision exist alternatives for estimating the depth information from a two different views with a known calibration and relative pose which sim-



(a) (b) (c)

Figure 1.4 – **Stereo vision system example.** The figure shows a sample from a stereo camera setup from Tsukuba dataset [12]. The first two images (a) and (b) correspond to the left and right camera of the setup. The image (c) represents the ground truth depth map of this stereo pair.

plifies the problem. These are the stereo systems, which similarly to the human binocular vision system, have two equidistant cameras, displaced horizontally to obtain two different views on a scene. In order to produce the depth information, the stereo systems compare the two images and estimate a disparity, which encodes the difference in horizontal coordinates of the corresponding image points. The values in this disparity map are inversely proportional to the scene depth at the corresponding pixel location (see figure 1.4). Several works have proposed solutions for stereo matching using classical methods [13] that are effective in controlled environments. On the other hand, the stereo matching minimization problem is NP-complete leading sometimes to expensive computational solutions. For this reason, novel approaches based on CNN [14, 15] methods have been proposed that improve the quality and convergence speed of the stereo vision matching algorithm.

Range Imaging

A different approach to estimate the depth information of a scene is using range imaging techniques to produce a 2D image containing the distance to points in a scene from a specific 3D point. The resulting image, commonly called range image or depth map contains a distance value for every pixel in the image which with a proper calibration can be directly related to a physical unit (see figure 1.5).

There are several devices to produce depth maps, usually referred as range cameras or RGB-D cameras that directly produce the depth information of the scene. These type of sensors project a known pattern on the scene by illuminating the scene with a specially designed pattern, structured light, that permit to determine the depth information from the reflected light. These type of devices are extensively

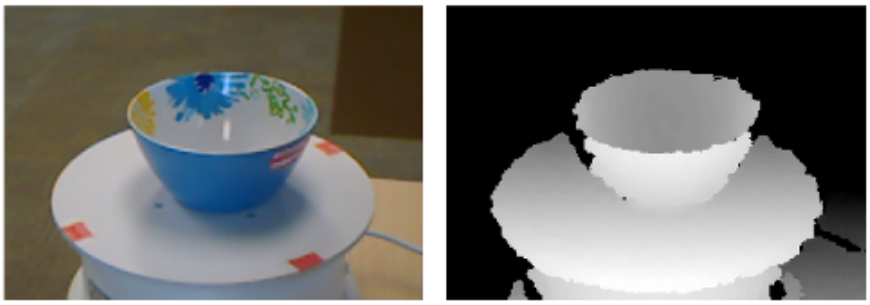


Figure 1.5 – **Range Imaging example.** The figure shows an RGB image (left) and the ground truth depth (right) obtained from a Kinect sensor. This sample is from the RGB-D Object Dataset [16].

used in robotics applications, or in controlled environments since they suffer from the limited measurement range and outdoor sunlight sensitivity [17].

Monocular depth estimation

With the raise of deep neural networks showing their outstanding performance over classical computer vision methods [18, 19], recent approaches have been proposed also for the task of depth estimation using convolutional neural networks (CNNs) [20]. Additionally, a variety of methods with different architectures have shown that a pixel-level depth can be recovered from a single image in an end-to-end manner based on recurrent neural networks (RNNs) [21], variational auto-encoders (VAEs) [22] and generative adversarial networks (GANs) [23]. Following this same direction, in chapter 4 we propose a novel end-to-end method to estimate the depth information of a scene from different views combining classical geometric computer vision techniques and deep learning.

1.2 Thesis contributions

In this thesis, we analyze the problem of scene reconstruction and we contribute with novel methods for detecting and describing keypoints, a framework that eases the transition from classical to end-to-end deep learning methods, and finally, an end-to-end solution to estimate depth maps using a multi-view camera approach. The rest of this thesis is organised as follows:

- In chapter 2, we study the problem of scene reconstruction in the context of the described classical features matching pipeline focusing on the specific tasks to detect and describe local features. The contributions made in this chapter consist of two novel CNN-based modules to be integrated in the traditional scene reconstruction pipeline: a keypoint detector and a keypoint descriptor.
- In chapter 3, we study the integration of classical image processing algorithms within the computational graph of a deep learning system in order to simplify the design of end-to-end pipelines. We contribute with Kornia, a framework that combines classical computer vision with modern auto-differentiable deep learning technologies which makes use of different hardware acceleration capabilities to run the algorithms in the GPUs and TPUs.
- In chapter 4, we study the problem of scene reconstruction in the context of a multi-camera environment in which we transition from the classic methods to an end-to-end approach. The contribution in this chapters consists in a novel end-to-end CNN-based method that estimates depth maps of deformable objects from an arbitrary camera view exploiting the geometry of the scene.

1.3 First Published Appearances contributions

The work described in this thesis has been submitted and/or published in different conferences and journals. In the following, the publications related with each chapter are listed:

- Chapter 2: Vassileios Balntas, **Edgar Riba**, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *BMVC*, 2016.
- Chapter 2: Axel Barroso-Laguna, **Edgar Riba**, Daniel Ponsa, and Krystian Mikolajczyk. Key.Net: Keypoint Detection by Handcrafted and Learned CNN Filters. In *ICCV*, 2019.
- Chapter 3: **Edgar Riba**, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an Open Source Differentiable Computer Vision Library for PyTorch. In *Winter Conference on Applications of Computer Vision (WACV)*, 2020.
- Chapter 3: **Edgar Riba**, Dmytro Mishkin, Jian Shi, Daniel Ponsa, Francesc Moreno-Noguer, and Gary Bradski. A survey on Kornia: an Open Source Differentiable Computer Vision Library for PyTorch. In *Journal of Engineering Applications of Artificial Intelligence* (under review).
- Chapter 3: Jian Shi, **Edgar Riba**, Dmytro Mishkin, and Francesc Moreno-Noguer. Differentiable Data Augmentation with Kornia. In *Neurips 2020 Workshop: Workshop on differentiable computer vision, graphics, and physics applied to machine learning*, 2020.
- Chapter 4: **Edgar Riba**, Jordi Sanchez-Riera, Yurun Tian, Fan Zhang, Albert Pumarola, Yiannis Demiris, Krystian Mikolajczyk, and Francesc Moreno-Noguer. Novel View Synthesis of Depth Maps for Cloth Manipulation. In *ICRA* 2021 (under review).
- Chapter 4: **Edgar Riba**, Jordi Sanchez-Riera, Albert Pumarola, Fan Zhang, Yurun Tian, Yiannis Demiris, Krystian Mikolajczyk, and Francesc Moreno-Noguer. Depth Map Synthesis for Deformable Clothes. In *CVPR* 2021 (under review).

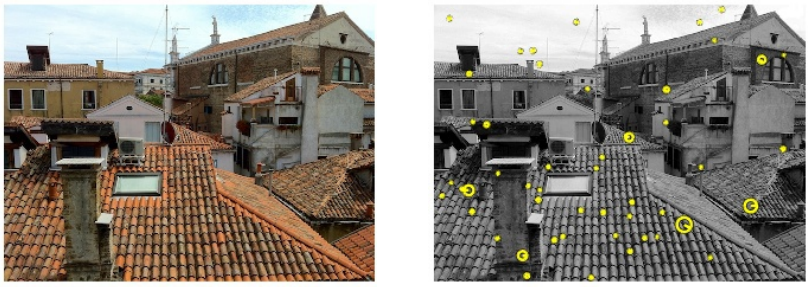
2 Local Features Detection and Description

2.1 Introduction

In this chapter we focus on the concept of local features and its usage within a 3d reconstruction pipeline. Precisely, as we introduced in the previous chapter we empathize on the classical features matching pipeline, or Wide Baseline Stereo (WBS) algorithm which can be decomposed in several sub-tasks to be processed independently. First, in section 2.2 we analyze the problem of the detection of points of interest in the scene, then, in section 2.3 how to describe the points in order to match them between different views robustly.

Local features is one of the most studied topics by the Computer Vision community which resulted to a wide list of different applications such as image stitching, camera calibration and obviously for scene reconstruction, which is the topic of this thesis. Local features detection can be formally described as the problem of finding discriminative regions in an image from which several properties as the location, orientation or the shape. On the other hand, the task to extract local features descriptors consist in computing embeddings from the small image patches which later are used for the matching task as seen in the WBS pipeline figure 1.2. Thus, the scope of this thesis is just to cover and propose novel solutions for the two initial steps of the classic pipeline within the scene reconstruction context.

Traditionally, several of the proposed methods to solve features detection use low level image processing techniques that brought to a wide number of well founded algorithms based on robust and handcrafted methods [6, 7], as can be seen in figure 2.1. However, in this thesis we propose a novel approach for keypoints detection task that combines handcrafted and learned CNN filters within a shallow multi-scale architecture. Handcrafted filters provide anchor structures for learned filters, which localize, score and rank repeatable features. Scale-space representation is used within the network to extract keypoints at different levels. We design a loss function to detect robust features that exist across a range of scales and to maximize the repeatability score. The proposed method, named Key.Net model is trained on data synthetically created from ImageNet [24] and evaluated on HPatches benchmark [25]. Results are provided to show that the approach outperforms state-of-the-art detectors in terms of repeatability, matching performance



(a) (b)

Figure 2.1 – **Local feature detection using VLFeat.** Example showing the result of a local features detection algorithm using the VLFeat library [26] with the SIFT [6] detector. **Left:** the original RGB image. **Right:** the image in grayscale containing the found keypoints represented as circles, where the radius show the estimated size of the keypoint. Image is courtesy of the VLFeat examples website.

and complexity.

Additionally, it has recently been demonstrated that local feature descriptors based on convolutional neural networks (CNN) can significantly improve the matching performance and with it the final 3d reconstruction of a scene. Previous work on learning such descriptors has focused on exploiting pairs of positive and negative patches to learn discriminative CNN representations. In this thesis, we propose a new method that utilizes triplets of training samples, together with in-triplet mining of hard negatives. We show that the method achieves state of the art results, without the computational overhead typically associated with mining of negatives and with lower complexity of the network architecture. The approach is compared to recently introduced convolutional local feature descriptors, and demonstrate the advantages of the proposed methods in terms of performance and speed.

2.2 Feature Detection

Recent advances in local feature detectors and descriptors has led to remarkable improvements in areas such as image matching, object recognition, self-guided navigation or 3D reconstruction. Although the general direction of image matching methods is moving towards learned based systems, the advantage of learning methods over handcrafted ones has not been clearly demonstrated in keypoint detection

[27] bringing initial approximations less efficient and with a high computational cost. However, the growing popularity of augmented reality (AR) headsets, as well as AR smartphone apps, has drawn more attention to reliable and efficient local feature detectors that could be used for surface estimation, sparse 3D reconstruction, 3D model acquisition or objects alignment, among others.

Traditionally, local feature detectors were based on engineered filters. For instance, approaches such as Difference of Gaussians [6], Harris-Laplace or Hessian-Affine [28] use combinations of image derivatives to compute feature maps, which is remarkably similar to the operations in trained CNN's layers. Intuitively, with just a few layers, a network could mimic the behavior of traditional detectors by learning the appropriate values in its convolutional filters. However, the improvements upon handcrafted detectors offered by recently proposed fully CNN based methods [29, 30, 31, 32, 33] are limited in terms of widely accepted metrics such as repeatability which measures the closeness between the detected keypoint location and its ground truth. In contrast to the method that we propose later, in addition to the keypoint location, in some cases the estimation of the scale (or surrounding region) and even the affine transformation is required. However, those methods usually lack in terms of accuracy when estimating the affine parameters of the feature regions. Robustness to scale variations seems particularly problematic while other parameters such as dominant orientation can be regressed well by CNNs [29, 34].

These problems in previous approaches motivates our novel architecture, termed Key.Net, that makes use of handcrafted and learned filters as well as a multi-scale representation to identify keypoints at different scales. The Key.Net architecture is illustrated in figure 2.2. Introducing handcrafted filters, which act as soft anchors, makes possible to generate a model with fewer parameters that state of the art detectors while maintaining the performance in terms of repeatability. The model operates on a multi-scale representation of full-size images and returns a response map containing a keypoint score for every pixel. The multi-scale input allows the network to propose stable keypoints across scales thus providing robustness to scale changes considering that the network does not explicitly predict the affine parameters for the scale.

Ideally, a robust detector is able to propose the same keypoints for images that undergo different geometric or photometric transformations. A number of related works have focused their objective function to address this issue, although they were based either on local patches [31, 32] or global map regression loss [9, 33, 35]. In contrast, we extend the covariant constraint loss used in previous works to a new objective function that combines local and global information in order to predict more stable keypoints across the different scales. The Key.Net architecture produces as an output a response map which needs an operator to extract discrete keypoint locations to evaluate the geometric loss function. For this reason, we design a

fully differentiable operator, Multi-scale Index Proposal, that proposes keypoints at multi-scale regions. We extensively evaluate the method in recently introduced HPatches benchmark [25] in terms of accuracy and repeatability according to the protocol from [36].

In summary, the contribution presented in this section is a multi-scale feature detection method with a shallow architecture based on:

1. the Key.Net network that generates a response map for those potential regions in the input image that might contain a keypoint.
2. a differentiable operator to extract and rank the stable keypoints across the scales.

The rest of the section is organized as follows. A revision in keypoint detection is shown in section 2.2.1. Section 2.2.2 presents the proposed hybrid Key.Net architecture of handcrafted and learned CNNs filters and section 2.2.3 introduces the loss. Implementation and experimental details are given in section 2.2.4 and the results are presented in section 2.2.5.

2.2.1 Related Work in Feature Detection

There are many surveys that extensively discuss feature detection methods [27, 37]. In this section related works are presented and organized in two main categories: handcrafted and learned based detectors.

Handcrafted Detectors

Traditional feature detectors localize geometric structures through engineered algorithms, which are often referred to as handcrafted. Harris [38] and Hessian [39] detectors used first and second order image derivatives to find corners or blobs in images. Those detectors were further extended to handle multi-scale and affine transformations [28, 40]. Region based methods such as SIFT [6] looked for blobs over multiple scale levels, and MSER [41] segmented and selected stable regions as keypoints. Later, SURF [42] accelerated the detection process by using integral images and an approximation of the Hessian matrix; and ORB [7] proposed an alternative to SIFT by proposing an oriented version the FAST [43] detector. Multi-scale improvements were proposed in KAZE [44] and its extension, A-KAZE [45], where Hessian detector was applied to a non-linear diffusion scale space in contrast to widely used Gaussian pyramid. Although corner detectors proved to be robust and efficient, other methods seek alternative structures within images.

Learned Detectors

The success of learned methods in general object detection and feature descriptors motivated the research community to explore similar techniques for feature detectors. FAST [43] was one of the first attempts to use machine learning to derive a corner keypoint detector. Further works extended FAST by optimizing it [46], adding a descriptor [47] or orientation estimation [7].

Latest advances in CNNs also made an impact on feature detection. TILDE [35] trained multiple piece-wise linear regression models to identify interest points that are robust under severe weather and illumination changes. DNet [31] introduced a new formulation to train a CNN based on feature covariant constraints. Previous detector was extended in [32] (TCDET) by adding predefined detector anchors, showing improved stability in training. In [30] were presented two networks, MagicPoint, and MagicWarp, which first extracted salient points and then a parameterized transformation between pairs of images. MagicPoint was extended in [9] to SuperPoint, which included a salient detector and descriptor. LIFT [29] implemented an end-to-end feature detection and description pipeline, including the orientation estimation for every feature. Quadruple image patches and a ranking scheme of point responses as cost function were used in [48] to train a neural network. In [49], authors proposed a pipeline to automatically sample positive and negative pairs of patches from a region proposal network to optimize jointly point detections and their representations. Recently, LF-Net [33] estimated position, scale and orientation of features by optimizing jointly the detector and descriptor.

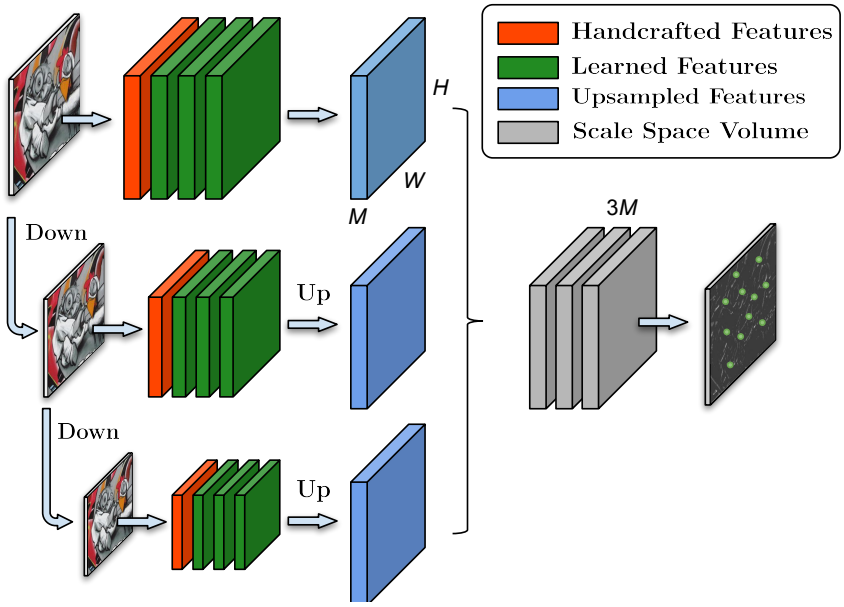


Figure 2.2 – **Features detection architecture.** The proposed Key.Net architecture combines handcrafted and learned filters to extract features at different scale levels. Feature maps are upsampled and concatenated. Last learned filter combines the Scale Space Volume to obtain the final response map.

2.2.2 Key.Net Architecture

In this section we present a method to detect keypoints which attempts to combine the strong points of both handcrafted and learned-based approaches with the aim to obtain results comparable to the best learned-based detectors with the efficiency and low cost of hand-crafted solutions. The proposed architecture, named Key.Ney, can be seen in figure 2.2.

Handcrafted and Learned Filters

The design of the handcrafted filters is inspired by the success of Harris [38] and Hessian [39] detectors, which used first and second order derivatives to compute the salient corner responses. A complete set of derivatives is called *LocalJet* [50] and they approximate the signal in the local neighborhood as known from Taylor

expansion:

$$I_{i_1,\ldots,i_n} = I_0 * \partial_{i_1,\ldots,i_n} g_\sigma(\vec{x}), \tag{2.1}$$

where g_σ denotes a Gaussian of width σ centered at $\vec{x} = \vec{0}$, and i_n denotes the derivative direction and order. Since higher order derivatives i.e., $n > 2$ are sensitive to noise and require large kernels, the subset of *LocalJet* that we have considered, includes derivatives and their combinations up to the second order only:

- **First Order**. From image I we derive 1st order gradients I_x and I_y. In addition, we compute $I_x * I_y$, ${I_x}^2$ and ${I_y}^2$ as in the second moment matrix of Harris detector [38].
- **Second Order**. From image I, 2nd order derivatives I_{xx}, I_{yy} and I_{xy} are also included as in the Hessian matrix used in Hessian and DoG detectors [6, 51]. We also add $I_{xx} * I_{yy}$ and I_{xy}^2 because of the performance boost that induce in Hessian detectors.

In addition we include a convolutional layer with M learnable filters, a batch normalization layer and a ReLU activation function in order to complement the handcrafted features. The hardcoded filters reduce the number of total learnable parameters to train the architecture, improving the stability and convergence during backpropagation.

Multi-scale Pyramid

The design of the proposed architecture has been done to be robust to small scale changes without the need for computing several forward passes and iterate over the scale space. As illustrated in figure 2.2, the network includes three scale levels of the input image which is blurred and downsampled by a factor of 1.2. All the feature maps resulting from the handcrafted filters are concatenated to feed the stack of learned filters in each of the scale levels. All three streams share the weights, such that the same type of anchors result from different levels and form the set of candidates for final keypoints. Feature maps from all scale levels are then upsampled, concatenated and fed to the last convolutional filter to obtain the final response map.

2.2.3 Geometric Loss Function

In the proposed architecture, given an image it produces a response map that will be used later to extract the keypoints needed by the classical scene reconstruction pipeline. The learning process of the parameters in our architecture will be based on a geometric loss function which needs to be very carefully designed in order to backpropagate the gradients through the entire pipeline.

In supervised training, the loss function relies on the ground truth. In the case of keypoints, ground truth is not well defined as keypoint locations are useful as long as they can be accurately detected regardless of geometric or photometric image transformation. Some learned detectors [31, 33, 48] train the network to identify keypoints without constraining their locations, where only the homography transformation between images is used as ground truth to calculate the loss as a function of keypoints repeatability.

Other works [9, 32, 35] show the benefits of using anchors to guide their training which work as a reference respect to the potential neighbours keypoints. Although anchors make the training more stable and lead to better results, they prevent the network from proposing new keypoints in case there is no anchor in the proximity. In contrast, the handcrafted filters in Key.Net provide a weak constraint with the benefit of the anchor-based methods while allowing the detector to propose new stable keypoints. The proposed approach, only takes the geometric transformation between images is required to guide the loss.

Index Proposal Layer

This section introduces the Index Proposal (IP) layer, the differentiable operator used as base to extract the sub-pixel coordinates of the keypoints from the produced response maps by the network. The operator is later extended to its multi-scale version in section 2.2.3 to enforce the detection across the different scale in the features domain.

Extracting coordinates for training keypoint detectors has been widely studied and showed great improvements: [29, 31, 32] extracted coordinates in reference to the patch size, SuperPoint [9] used a channel-wise softmax to get maxima belonging to fix grids of $8x8$, and [52] used a spatial softmax layer to compute the global maxima of a feature map, obtaining one keypoint candidate per feature map. In contrast to previous methods, our IP layer is able to return multiple global keypoint coordinates centered on local maxima from a single image without constraining the number of keypoints to the depth of the feature map [52] or the size of the grid [9]. This condition is beneficial in the sense that the network is not constrained by the internal design and later makes very practical to extract an arbitrary number of keypoints.

Similarly to handcrafted techniques, keypoint locations are indicated by local maxima of the filter response map $\mathbf{R}$ output by Key.Net. Spatial softmax operator is an effective method for extracting the location of a soft maximum within a window [9, 29, 33, 52]. Therefore, to ensure that the IP layer is fully differentiable, we rely on spatial softmax operator to obtain the coordinates of a single keypoint per window. Consider a window w_i of size $N \times N$ in $\mathbf{R}$, with the score value at each

coordinate $[u, v]$ within the window, then we compute m_i by exponentially scaling and normalizing w_i:

$$m_i(u,v) = \frac{e^{w_i(u,v)}}{\sum_{j,k}^{N} e^{w_i(j,k)}}. \tag{2.2}$$

Due to the exponential scaling the maximum value in w_i dominates and its coordinate is the expected location calculated as the weighted average $[\bar{u}_i, \bar{v}_i]$ gives an approximation of the maximum coordinates m_i as follows:

$$[x_i, y_i]^T = [\bar{u}_i, \bar{v}_i]^T = \sum_{u,v}^{N} [W \odot m_i, W^T \odot m_i]^T + c_w, \tag{2.3}$$

where W is a kernel of size $N \times N$ with index values $j = 1 : N$ along its columns, pointwise product $\odot$, and c_w is the top-left corner coordinates of window w_i. This is similar to non-maxima suppression (NMS) but unlike NMS, the IP layer is differentiable and it is a weighted average of the global maxima of the window rather than the exact location of it. Depending on the base of the power expression in equation 2.2, multiple local maxima may have a more or less significant effect on the resulting coordinates.

A detector is covariant if same features are detected under varying image transformations. Covariant constraint was formulated as a regression problem in [31]. Given images I_a and I_b, and ground truth homography $H_{b,a}$ between them, the loss $\mathbf{L}$ is based on the squared difference between the coordinates extracted by IP layer and its ground truth coordinates extracted by a non-differentaible NMS in the corresponding windows from I_a and I_b:

$$\mathbf{L}_{IP}(I_a, I_b, H_{a,b}, N) = \sum_{i}^{NxN} \alpha_i \| [x_i, y_i]_a^T - H_{b,a}[\hat{x}_i, \hat{y}_i]_b^T \|^2,$$

$$\text{and} \quad \alpha_i = \mathbf{R}_a(x_i, y_i)_a + \mathbf{R}_b(\hat{x}_i, \hat{y}_i)_b, \tag{2.4}$$

where $\mathbf{R}_a$ and $\mathbf{R}_b$ are the response map of I_a and I_b with coordinates related by the homography $H_{b,a}$. We skip homogeneous coordinates for simplicity. $\hat{x}$ and $\hat{y}$ are obtained from applying $H_{b,a}$ to (x, y). The information in α_i is the accumulation of the response map of each image once they are put in correspondence with the $H_{a,b}$ which controls the contribution of each location based on its score value. Gradients are only back-propagated where IP layer was applied, therefore, we switch I_a and I_b and combine both losses to enforce consistency.

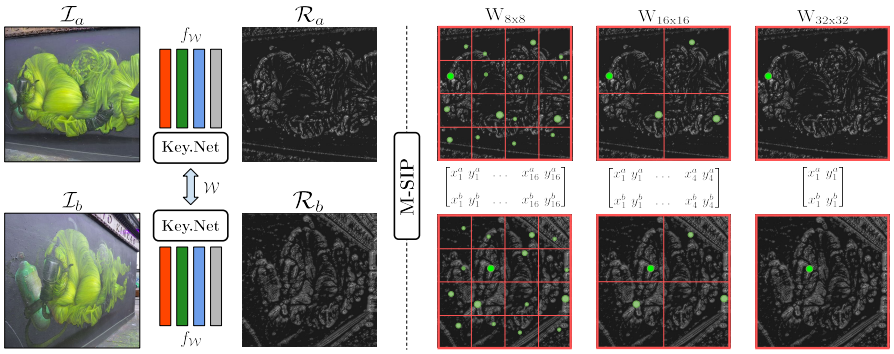


Figure 2.3 – **Siamese training process.** Image I_a and I_b go through Key.Net to generate their response maps, R_a and R_b. M-SIP proposes interest point coordinates for each one of the windows at multi-scale regions. The final loss function is computed as a regression of coordinate indexes from I_a and transformed coordinate indexes from I_b (see in equation 2.4).

Multi-scale Index Proposal Layer

IP layer returns one location per window, therefore, the number of keypoints per image strongly depends on the predefined window size N. In particular, if N is big with an increasing size only a few dominant keypoints survive in the image. In [53], authors demonstrated improved performance of local features by accumulating image features not only within a spatial window but also within the neighboring scales. We propose to extend IP layer loss by incorporating multi-scale representation of a local neighborhood. Multiple window sizes encourage the network to find keypoints that exist across a range of scales. The additional benefit of including larger windows is that other keypoints within the window can act as anchors for the estimated location of the dominant keypoint. Similar idea proved successful in [54], where stable region boundaries are used.

We, therefore, propose the Multi-Scale Index Proposal (M-SIP) layer. M-SIP splits multiple times the response map into grids, each with a window size of $N_s \times N_s$ and computes the candidate keypoint position for each window as shown in figure 2.3. Our proposed loss function is the average of covariant constraint losses from all scale levels:

$$\mathbf{L}_{MSIP}(I_a, I_b, H_{a,b}) = \sum_s \lambda_s \mathbf{L}_{IP}(I_a, I_b, H_{a,b}, N_s), \tag{2.5}$$

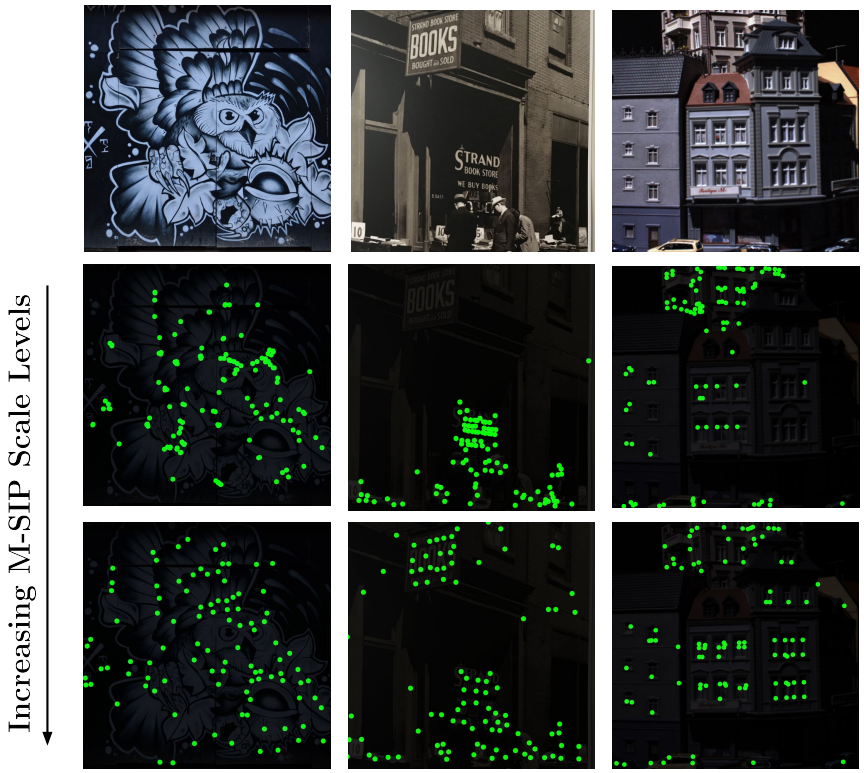


Figure 2.4 – **Key.Net qualitative results with different windows sizes.** Keypoints obtained after adding larger context windows to M-SIP operator. The points that are more stable remain as the M-SIP operator increases its window size. While the feature maps in the middle row contain points around edges or non discriminative areas from the initial scales, the bottom row shows detections from the lower scales which shows that are more robust under geometric transformations.

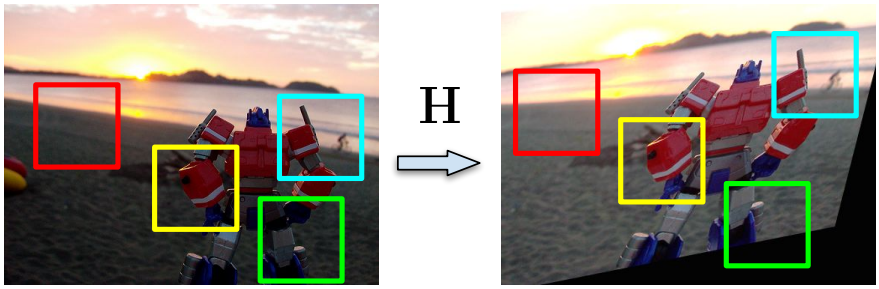


Figure 2.5 – **Homography synthetic data generation.** We apply random geometric and photometric transformations to images and extract pairs of corresponding regions as the training set. Red crop is discarded by checking the response of the handcrafted filters.

where s is the index of the scale level with N_s as window size, $\mathbf{L}_{IP}$ is the covariant constraint loss and λ_s is the control parameter at scale level s, that decreases proportionally to the increasing window area as larger windows lead to a larger loss, which is somewhat similar to the scale-space normalisation [28].

The combination of different scales imposes an intrinsic process of simultaneous scoring and ranking of keypoints within the network. In order to minimize the loss, the network will learn to give higher scores to the pixel locations centered on image features that remain dominant across a range of scales. Figure 2.4 shows different response maps for increasing window size.

2.2.4 Experimental Evaluation

In this section, we present implementation details, metrics and the dataset used for evaluating the method.

Training Data

We generate a synthetic training set from ImageNet ILSVRC 2012 [55] dataset. We apply random geometric transformations to images and extract pairs of corresponding regions as our training set. The process is illustrated in figure 2.5. The parameters of the transformations are: scale $[0.5, 3.5]$, skew $[-0.8, 0.8]$ and rotation $[-60°, 60°]$ which are common transformations seen across scene reconstruction datasets. Textureless regions are discarded by checking if the mean response of any of the handcrafted filters is lower than a threshold. We modify contrast, brightness

and hue value in HSV space to one of the images to improve network's robustness against illumination changes. In addition, for each pair, we generate a binary mask that indicates the common area between images. Mask is used in training to avoid regressing indexes of keypoints that are not present in the common region. There are 12,000 image pairs of size 192 × 192. We use 9,000 of them as the training data and 3,000 as validation set.

Evaluation Metrics

We follow the evaluation protocol proposed in [36] and improved in the follow up works [27, 29, 31, 32] which is based on the repeatability scores. The repeatability score for a pair of images is computed as the ratio between the number of corresponding keypoints and the lower number of keypoints detected in one of the two images. We take the top-k extracted keypoints to compare across methods and allow each keypoint to match only once as in [35, 46]. In addition, as exposed by [27], we address the bias from the magnification factor that was applied to accelerate the computation of the overlap error between multi-scale keypoints.

Keypoints are identified by spatial coordinates and scales at which the features were detected, the last defined by a squared around the center. To identify corresponding keypoints we compute the Intersection-over-Union error, ϵ_{IoU}, between the areas of the two candidates. To evaluate the accuracy of keypoint location and scale independently, we perform two sets of experiments. One is based on the scales extracted from the scale space and the other assumes the scales are correctly detected by using the ground truth parameters. In our benchmark, we use top 1,000 interest points that belong to the common region between images and a match is considered correct when ϵ_{IoU} is smaller than 0.4 i.e., the overlap between corresponding regions is more than 60%. The scales are normalized as in [27], which sets the larger size in a pair of points to 30 pixels, and rescales the other one accordingly. Non-maxima suppression of 15×15 is performed at inference time during evaluation.

HPatches [25] dataset is used for testing. HPatches contains 116 sequences, which are split between viewpoint and illumination transformations, 59 and 57 sequences respectively. HPatches offers predefined image patches for evaluating descriptors, instead, we use full images for evaluating keypoint detectors.

Experimental Setup

Training is performed in a siamese pipeline, with two instances of Key.Net that share the weights and are updated at the same time. Each convolutional layer has $M = 8$ filters of size 5×5, with He [56] weights initialization and L2 kernel regularizer. We compute the covariant constraint loss $\mathbf{L}_{M\text{-}SIP}$ for five scale levels, with the size of

Figure 2.6 – **Key.Net quantitative results for different learnable blocks.** Repeatability results for different combinations of handcrafted filters and a number of learnable layers ($M = 8$ filters each). A higher number of layers leads to better results. All repeatability scores are computed on synthetic validation set from ImageNet.

the M-SIP windows $N_s \in [8, 16, 24, 32, 40]$ and loss term $\lambda_s \in [256, 64, 16, 4, 1]$, that were determined by performing a hyperparameter search on the validation set. Larger candidate window sizes have greater mean errors between coordinate points since the maximum distance is proportional to the window size. Thus, λ_s has the largest value for the smallest window. We use a batch size of 32, an Adam Optimizer with a learning rate of 10^{-3} and a decay factor of 0.5 after 30 epochs. On average, the architecture converges in 20 epochs, 2h on a machine with an i7-7700 CPU running at 3.60GHz and a NVIDIA GeForce GTX 1080 Ti. Evaluation benchmark, synthetic data generator, Key.Net network, and loss are implemented using TensorFlow and are available on GitHub[1].

2.2.5 Detection Results

In this section, we present the experiments and discuss the results. We first show results on validation data for several variants of the proposed architecture. Next, Key.Net repeatability scores in single-scale and multi-scale are presented along with the state-of-the-art detectors on HPatches. Moreover, we evaluate the matching per-

[1] https://github.com/axelBarroso/Key.Net

M-SIP Region Sizes					
W_{8x8}	W_{16x16}	W_{24x24}	W_{32x32}	W_{40x40}	Repeatability
✓	-	-	-	-	70.5
✓	✓	-	-	-	74.6
✓	✓	✓	-	-	76.8
✓	✓	✓	✓	-	77.6
-	-	-	-	✓	65.7
-	-	-	✓	✓	71.4
-	-	✓	✓	✓	73.2
-	✓	✓	✓	✓	74.9
✓	✓	✓	✓	✓	**79.1**

Table 2.1 – **Key.Net quantitative results for M-SIP regions sizes.** Comparison of repeatability results for the various number of levels in M-SIP operator. We show different combinations of context losses as the final loss, from smaller to larger regions. The best result is obtained when using five window sizes from 8×8 up to 40×40.

formance, the number of learnable parameters and inference time of our proposed detector and compare to other techniques.

Preliminary Analysis

We study several combinations of loss terms, different handcrafted filters and the effects of the number of learnable layers or pyramid levels within the architecture.

- **Filter Combinations** are analyzed in figure 2.6. We show results for 1^{st} and 2^{nd} order filters as well as their combination. All networks have the same number of filters, however, we either freeze first layer of 10 filters with handcrafted kernels (c.f. section 2.2.2) or learn them depending on the variant of our network, e.g, in Fully Learnable Key.Net there are no handcrafted filters as all are randomly initialized and learned. The results show that the information provided by handcrafted filters is essential when the number of learnable layers is small. Handcrafted filters act as soft constraints, which directly discard areas without gradients, i.e. non-discriminative with low repeatability. However, as we add more learnable blocks, repeatability scores for combined and fully learnable networks become comparable. Naturally, gradient-based handcrafted filters are simple, and architectures with enough complexity can learn them.
- **Multiple Pyramid Levels** at the input to the network also affect the detection performance as shown in table 2.1. For a single pyramid level, only the original

image is used as input. Adding pyramid levels is similar to increasing the size of the receptive fields in the architecture. Our experiment suggests that using more than three levels does not lead to significantly improved results. On the validation set, we obtain a repeatability score of 72.5% for one level, an increase of 6.6% for three, and 7.0% for five levels. We, therefore, use three levels, which achieve good performance while keeping the computational cost low.

Detection algorithm comparison

This section presents the results for state-of-the-art local feature detectors along with our proposed method. Table 2.2 shows the repeatability score, average intersection-over-union error $\bar{\epsilon}_{IoU}$. Suffixes -TI and -SI, refer to translation (detection at a single scale only) and scale invariance (detection at multiple scales), respectively. Key-point location is only evaluated under L by assuming correct scale detection, while scale and location (SL) use the actual detected scale and location for computing the overlap error.

In addition to Key.Net, we evaluate in these experiment the performance of Tiny-Key.Net, which is a reduced size architecture with all handcrafted filters but only one learnable layer with one filter ($M = 1$) and a single downscaling factor in the input. The idea behind Tiny-Key.Net is to demonstrate how far the complexity can be reduced while keeping good performance. Key.Net and Tiny-Key.Net are extended to scale invariance by evaluating the detector on several scaled images, similar to [32]. We also show results on single scale input Key.Net-TI, to compare it directly with other TI detectors such as SuperPoint or TILDE. We manually set the threshold of algorithms to return at least 1,000 points per image. As MSER proposes regions without scoring or ranking, we randomly pick 1,000 points to compute the results. We repeat this experiment ten times and average the results for MSER. Key.Net has the best results on viewpoint sequences, in terms of both, location and scale. Tiny-Key.Net does not perform as well as Key.Net but it is within the top three repeatability scores, after Key.Net-TI and Key.Net-SI.

On illumination sequences, Key.Net-TI performs the best among TI detectors, which are not affected by scale estimation errors. TCDET, which uses points detected by TILDE as anchors, is the most accurate in location estimation compared to other SI detectors. Note that TILDE based detectors were specifically designed and trained for illumination sequences. LF-Net is the best SI detector according to SL overlap, not suffering much from incorrect scale estimations.

	Viewpoint					Illumination				
	Repeatability		$\bar{\epsilon}_{IoU}$		S_{range}	Repeatability		$\bar{\epsilon}_{IoU}$		S_{range}
	SL	L	SL	L	SL	SL	L	SL	L	SL
SIFT-SI [6]	43.1	57.6	**0.18**	0.12	78.6	47.8	60.4	0.18	0.12	84.5
SURF-SI [42]	46.7	60.3	**0.18**	0.18	24.8	53.0	64.0	0.15	0.11	27.4
FAST-TI [43]	30.4	63.1	0.21	**0.10**	-	63.6	63.6	**0.09**	**0.09**	-
MSER-SI [41]	56.4	62.8	**0.12**	**0.08**	**503.7**	46.5	54.5	0.12	0.10	**524.8**
Harris-Laplace-SI [51]	45.1	62.0	0.20	0.13	**95.9**	52.7	62.0	0.17	**0.08**	**90.4**
KAZE-SI [44]	53.3	65.7	0.20	0.11	12.5	56.9	65.7	0.12	0.10	12.7
AKAZE-SI [45]	54.0	65.6	0.19	**0.10**	13.5	64.9	69.1	0.11	**0.09**	13.6
TILDE-TI [35]	31.0	65.1	0.20	0.15	-	**70.4**	70.4	0.11	0.11	-
LIFT-SI [29]	43.4	59.4	0.20	0.13	13.3	51.6	65.4	0.18	0.12	13.8
DNet-SI [31]	49.4	62.2	0.21	0.14	11.4	59.1	65.1	0.14	0.13	17.1
TCDET-SI [32]	49.6	61.6	0.23	0.16	6.7	66.9	**71.0**	0.16	0.15	11.4
SuperPoint-TI [9]	33.3	67.1	0.20	0.17	-	69.9	69.9	**0.10**	0.10	-
LF-Net-SI [33]	32.3	62.2	0.23	0.12	2.00	68.6	69.1	**0.10**	0.10	2.0
Tiny-Key.Net-SI	**57.8**	70.3	0.20	0.12	7.6	56.1	62.8	0.14	0.11	7.6
Key.Net-TI	34.2	**71.5**	0.20	0.11	-	**72.0**	**72.0**	**0.10**	0.10	-
Key.Net-SI	**59.6**	**72.6**	0.19	0.14	7.6	61.3	66.2	0.12	0.10	7.6

Table 2.2 – **Key.Net repeteability results.** Repeatability results (%) for translation (TI) and scale (SI) invariant detectors on HPatches. We also report average overlap error $\bar{\epsilon}_{IoU}$ and ratio of maximum to minimum extracted scale S_{Range}. In SL, scales and locations are used to compute overlap error, meanwhile, in L, only locations are used and scales are assumed to be correctly estimated. Key.Net and Tiny-Key.Net are the best algorithms on viewpoint, for both L and SL. On illumination sequences, translation invariant Key.Net-TI obtains the best accuracy.

	Matching Score	
	View	Illum
MSER [41] + HardNet [57]	11.7	18.8
SIFT [6] + HardNet [57]	23.2	24.8
HarrisLaplace [51] + HardNet [57]	30.0	31.7
AKAZE [45] + HardNet [57]	36.4	41.4
TILDE [35] + HardNet [57]	32.3	40.3
LIFT [29] + HardNet [57]	30.3	32.8
DNet [31] + HardNet [57]	33.5	34.7
TCDET [32] + HardNet [57]	27.6	36.3
SuperPoint [9] + HardNet [57]	37.4	**43.0**
LF-Net [33] + HardNet [57]	26.9	**43.8**
LIFT [29]	21.8	26.5
SuperPoint [9]	**38.0**	41.5
LF-Net [33]	23.0	29.1
Tiny-Key.Net + HardNet [57]	37.9	37.3
Key.Net + HardNet [57]	**38.4**	40.7

Table 2.3 – **Key.Net matching scores results.** Matching score (%) of best detectors together with HardNet and state-of-the-art detector/descriptors. Results on HPatches sequences, both viewpoint, and illumination. Key.Net architecture get the best matching score for viewpoint, while LF-Net+HardNet for illumination sequences.

Keypoint matching analysis

In order to demonstrate that the detected features are useful for matching, table 2.3 shows the percentage of keypoints that have been put in correspondence matching scores for the different detectors combined with HardNet descriptor [57]. As our method only focuses on the detection part, and for a fair comparison, we used the same descriptor and discard the orientation for all methods that provide it since our method do not estimate the keypoint orientation. In addition, we include in the table LIFT[29], SuperPoint[9] and LF-Net[33] with their descriptors, but ignoring their orientation estimation. The matching score is computed as the ratio between features matched and detected top 1k best ranked points. Best matching percentages matching scores are obtained by Key.Net on viewpoint, and LF-Net+HardNet on illumination. Feature detectors that were optimized jointly with

a descriptor [9, 29, 33] have better matching score than regular learned detectors on illumination sequences, but not on viewpoint. Handcrafted AKAZE performs close to the top learned methods.

Efficiency

We also compare the number of learnable parameters, indicating then the complexity of the predictor, therefore, an increased risk of overfitting and a need for large training data. Table 2.4 shows the approximate number of parameters for different architectures. Learnable parameters that are not used during inference in the detector part are not counted for SuperPoint and LF-Net detectors. The highest complexity is from SuperPoint with $940k$ learnable parameters. Key.Net has nearly 160 times fewer parameters and Tiny-Key.Net has 3,100 times fewer parameters than SuperPoint with better repeatability for viewpoint scenes. The inference time of an image of 600×600 is 5.7ms (175 FPS) and 31ms (32.25 FPS) for Tiny-Key.Net and Key.Net, respectively.

Number of Learnable Parameters				
TCDET	SuperPoint	LF-Net	Key.Net	Tiny-Key.Net
548k	940k	39k	**5.9k**	**280**

Table 2.4 – **Key.Net number of parameters comparison.** Comparison of the number of learnable parameters for state-of-the-art architectures. Tiny-Key.Net has only one learnable block with one filter.

2.2.6 Conclusions

In this section we introduced an approach to detect local features that combines handcrafted and learned CNN filters that can be easily replaced by any existing detection method in the reconstruction pipeline. We have proposed a multi-scale index proposal layer that finds keypoints across a range of scales, with a loss function that optimizes the robustness and discriminating properties of the detections. We demonstrated how to compute and combine differentiable keypoint detection loss for multi-scale representation. Evaluation results on large benchmark show that combining handcrafted and learned features as well as multi-scale analysis at different stages of the network improves the repeatability scores compared to other state-of-the-art keypoint detection methods. We further show that using handcrafted filters significantly reduce the complexity of the architecture leading to a detector with 280 learnable parameters and inference of 175 frames per second.

2.3 Feature description

In the previous section we described the process to detect interesting points in the image using classical and CNN based algorithms. We also reviewed the state of the art and proposed a new method based on CNNs that improves the pre-existing algorithms. Thus, in order to complete a reconstruction pipeline using local features (see again figure 1.2), as we have discussed during the introduction of this thesis it is also necessary to not only detect but also describe the detected points. As seen in figure 2.7 the classical approach for describing local regions is to extract small first image patches and compute a descriptor based on the properties obtained from the detection algorithm. There is a wide variety of methods to compute descriptors from the image patches using classical methods such as the very well known SIFT [6] that bases on computing histograms of gradients and has inspired a whole generation of researchers in this field.

However, it has recently been demonstrated that local feature descriptors based on convolutional neural networks (CNN) can significantly improve the matching performance and with it the final 3d reconstruction of a scene. Previous work on learning such descriptors has focused on exploiting pairs of positive and negative patches to learn discriminative CNN representations. In this thesis, we propose a new method that utilizes triplets of training samples, together with in-triplet mining of hard negatives. We show that the method achieves state of the art results, without the computational overhead typically associated with mining of negatives and with lower complexity of the network architecture. The approach is compared to recently introduced convolutional local feature descriptors, and demonstrate the advantages of the proposed methods in terms of performance and speed. In addition, different loss functions associated with triplets are examined using basic geometric constraints.

In this section, we investigate the use of triplets in learning local feature descriptors with convolutional neural networks including the following contributions:

1. we examine different different loss functions for triplet based-learning.
2. we investigate the performance of these methods in terms of patch matching, and patch pairs classification in widely used benchmarks.
3. we show that in-triplet hard negative mining can lead to improved results.
4. we demonstrate that excellent descriptor performance can be obtained with a shallow network thus avoiding computationally complex architectures and expensive mini-batch hard negative mining.

(a) (b)

Figure 2.7 – **Local feature description using VLFeat.** Example showing the result of a local features detection algorithm using the VLFeat library [26] with the SIFT [6] descriptor. **Left:** the original RGB image. **Right:** the image in grayscale containing the detected keypoints including the region from where descriptor will be extracted according to the keypoint size and orientation. This image has been obained from the VLFeat examples website.

2.3.1 Related Work in Feature Description

Finding correspondences between images via local descriptors is one of the most extensively studied problems in computer vision due to the wide range of applications. The field has witnessed several breakthroughs in this area such as SIFT [58], invariant region detectors [28], fast binary descriptors [59], optimised descriptor parameters [60, 61] which have made a significant and wide impact in various computer vision tasks. Recently end-to-end learnt descriptors [8, 62, 63, 64] based on CNN architectures and training on a large dataset of positive and negative sample pairs, were demonstrated to significantly outperform state of the art features. This was a natural adoption of CNN to local descriptors as deep learning had already been shown to significantly improve in many computer vision areas [65].

Recent work on deep learning for learning feature embeddings examines the use of triplets of samples instead of solely focusing on pairs [66, 67, 68]. Different loss functions are proposed in these works, but a systematic study of their characteristics is yet to be done. In addition, these works are focused on more general embeddings (e.g. product similarity, 3D description of objects, MNIST classification).

The design and implementation of local descriptors has undergone a remarkable evolution over the past two decades ranging from differential or moment

invariants, correlations, PCA projected patches, histograms of gradients or other measurements, etc. An overview of pre-2005 descriptors with SIFT [58] identified as the top performer can be found in [69]. Its benchmark data accelerated the progress in this field and there have been a number of notable contributions, including recent DSP-SIFT [70], falling into the same category of descriptors as SIFT but the improvements were not sufficient to supersede SIFT in general. The research focus shifted to improve the speed and memory footprint e.g. as in BRIEF [59] and the follow up efforts. Introduction of datasets with correspondence ground truth [60] stimulated development of learning based descriptors which try to optimise descriptor parameters and learn projections or distance metrics [61, 71] for better matching.

End-to-end learning of patch descriptors using CNN has been attempted in several works [8, 62, 63, 64] and consistent improvements were reported over the state of the art descriptors. Interest in the field started from results shown in [62] that the features from the last layer of a convolutional deep network trained on ImageNet [72] can outperform SIFT. This was a significant result, since the convolutional features from ImageNet were not specifically learnt for such local representations. Learning a CNN from local patches extracted from local features only, based on a siamese architecture with hinge contrastive loss [73] was demonstrated in [8, 63, 64] to significantly improve the matching performance. This approach was originally proposed in [74], however due to the limited evaluation this work was not immediately followed.

Note that in [8, 64] both feature layers and metric layers are learnt in the same network. Thus, the final contrastive loss is optimised in terms of the abstract metric learned in the last layer of the network. On the contrary, [63] directly uses the features extracted after the convolutional layers of the CNN, without training a specialised distance layer. This allows the extracted descriptors to be used in traditional pipelines. However, the experiments from [8] show that metric learning performs better than generic *L2* matching. Another important observation from [8] is that multiscale architectures perform better than the single scale ones. However, this is not unique to the CNNs, since previous works have shown that aggregating descriptors from multiple scales, improves the results.

2.3.2 Learning patch descriptors

In this section, we first discuss the two most commonly used loss functions when learning with triplets, and we then investigate their characteristics. A patch descriptor is considered as a non-linear encoding resulting from a final layer of a convolutional neural network. Let $\boldsymbol{x} \in \mathbb{R}^{n \times n}$ represent the patch given as input to the network and $f(\boldsymbol{x}) \in \mathbb{R}^D$ represent the D features given as output from the

network. For all of the methods below, the goal is to learn the embedding $f(\boldsymbol{x})$ s.t. $||f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)||_2$ is low if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are extracted from the same physical point location i.e. positive match, and high otherwise.

Learning with pairs

Learning with pairs involves training from samples of the form $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ell\}$, with ℓ being a label for the patch pair, which is -1 for negative pairs, and 1 for positive pairs. The contrastive loss is defined as

$$l(\boldsymbol{x}_1, \boldsymbol{x}_2; \ell) = \begin{cases} ||f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)||_2 & \text{if } \ell = 1 \\ max(0, \mu - ||f(\boldsymbol{x}_1) - f(\boldsymbol{x}_2)||_2) & \text{if } \ell = -1 \end{cases} \tag{2.6}$$

where μ is an arbitrarily set margin. Note that the weights of the CNN in $f(\cdot)$ need to be regularised, otherwise the margin would have no effect. Intuitively the hinge embedding loss penalizes positive pairs that have large distance and negative pairs that have small distance (less than μ).

Note that learning local feature descriptors is a more specific problem than general image classification such as in ImageNet, since the transformations a local patch can undergo are limited compared to different objects of the same visual category. In addition, patches in pairs representing negative examples are usually very different, thus make it easy for the learning process to optimize the distances. This issue is identified in [63], where the majority of the negative patch pairs ($\ell = -1$) do not contribute to the update of the gradients in the optimization process as their distance is already larger than μ parameter in Eq. (2.6). To address this issue hard negative mining was proposed [63] to include more negative pairs in the training. The hard negative training pairs were identified by their distance and a subset of these examples were re-fed to the network for gradient update in each iteration. Note that while this process leads to more discriminative convolutional features, it also comes at a very high computational cost, since in each epoch, a subset of the training data need to be backpropagated again through the network. Specifically, the best performing architecture from [63], required 67% of the computational cost to be spent for mining hard negatives.

Learning with triplets

Recent work in [67] shows that learning representations with triplets of examples, gives much better results than learning with pairs using the same network. Inspired by this, we focus on learning feature descriptors based on triplets of patches.

Learning with triplets involves training from samples of the form $\{\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{n}\}$, where a is the *anchor*, p *positive*, which is a different sample of the same class as a, and n

negative is a sample belonging to a different class. In our case, a and p are different viewpoints of the same physical point, and n comes from a different keypoint. Furthermore, optimising the parameters of the network brings a and p close in the feature space, and pushes a and n far apart. For brevity, we shall write that $\delta_+ = ||f(\boldsymbol{a}) - f(\boldsymbol{p})||_2$ and $\delta_- = ||f(\boldsymbol{a}) - f(\boldsymbol{n})||_2$. We can categorise the loss functions that have been proposed in the literature for learning convolutional embeddings with triplets into two groups, the *ranking-based losses* and the *ratio-based losses* [66, 67, 68]. Below we give a brief review of both categories, and discuss their differences.

Margin ranking loss. This ranking loss that was first proposed for learning embeddings using convolutional neural networks in [66] is defined as

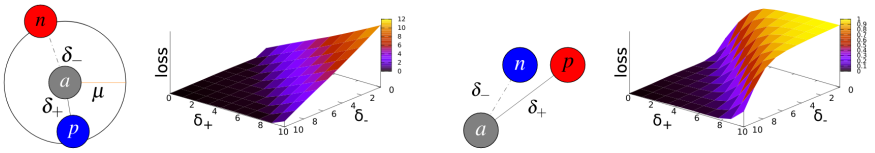
$$\lambda(\delta_+, \delta_-) = max(0, \mu + \delta_+ - \delta_-) \tag{2.7}$$

where μ is a margin parameter. The margin ranking loss is a convex approximation to the $0-1$ ranking error loss, which measures the violation of the ranking order of the embedded features inside in the triplet. The correct order should be $\delta_- > \delta_+ + \mu$. If that is not the case, then the network adjusts its weights to achieve this result. As it can be seen the formulation also involves a margin, similarly to Eq.(2.6). Note that if this marginal distance difference is respected, the loss is 0, and thus the weights are not updated. Fig. 2.8 (b) illustrates the loss surface of $\lambda(\delta_+, \delta_-)$. The loss remains 0 until the margin is violated, and after that, there is a linear increase. Also note that the loss in not upper bounded, only lower bounded to 0.

Ratio loss. In contrast to the ranking loss that forces the embeddings to be learned such that they satisfy ranking of the form $\delta_- > \delta_+ + \mu$, a ratio loss is investigated in [67] which optimises the ratio distances within triplets. This loss learns embeddings such that $\frac{\delta_-}{\delta_+} \to \infty$.

$$\hat{\lambda}(\delta_+, \delta_-) = (\frac{e^{\delta_+}}{e^{\delta_+} + e^{\delta_-}})^2 + (1 - \frac{e^{\delta_-}}{e^{\delta_+} + e^{\delta_-}})^2 \tag{2.8}$$

As one can examine from Eq. 2.8, the goal of this loss function is to force $(\frac{e^{\delta_+}}{e^{\delta_+} + e^{\delta_-}})^2$ to 0, and $(\frac{e^{\delta_-}}{e^{\delta_+} + e^{\delta_-}})^2$ to 1. Note that both are achieved by the first term of the equation, but we report here the original formulation from [67]. There is no margin associated with this loss, and by definition we have $0 \leq \hat{\lambda} \leq 1$ for all values of δ_-, δ_+. Note that unlike the margin-ranking loss, where $\lambda = 0$ is possible, every training sample in this case is associated with some non-negative loss value. Figure. 2.8 (d) shows the loss surface of $\hat{\lambda}(\delta_+, \delta_-)$, which compared to the ranking based loss has a clear slope between the two loss levels, and the loss reaches a plateau quickly when $\delta_- > \delta_+$. Also note that this loss is upper bounded to 1.



(a) (b) (c) (d)

Figure 2.8 – **Proposed triplet loss functions.** (a) Margin ranking loss. It seeks to push n outside the circle defined by the margin μ, and pull p inside. (b) Margin ranking loss values in function of δ_-, δ_+ (c) Ratio loss. It seeks to force δ_+ to be much smaller than δ_-. (d) Ratio loss values in function of δ_-, δ_+

In-triplet hard negative mining with anchor swap

All previous works that exploit the idea of triplet based learning use only two of the possible three distances within each triplet w.r.t. one sample used as an *anchor*, thus ignoring the third distance $\delta_-^{'} = ||f(\boldsymbol{p}) - f(\boldsymbol{n})||_2$. Note that since the feature embedding network already computes the representations for $f(\boldsymbol{a}), f(\boldsymbol{p}), f(\boldsymbol{n})$, there is no need for extra convolutional overhead to compute $\delta_-^{'}$ except evaluating the L_2 distance.

We define the *in-triplet hard negative* as $\delta_* = min(\delta_-, \delta_-^{'})$. If $\delta_* = \delta_-^{'}$, we swap $\{a, p\}$, and thus p becomes the *anchor*, and a becomes the *positive* sample. This ensures that the hardest negative inside the triplet is used for backpropagation. Subsequently, the margin ranking loss becomes $\lambda(\delta_+, \delta_*) = max(0, \mu + \delta_+ - \delta_*)$. A similar expression can be devised for the ratio loss. This simple technique can lead to improved results without computational overhead, as we experimentally show in section 2.3.4.

2.3.3 Experimental Evaluation

To demonstrate the impact that triplet based training has on the performance of CNN descriptors we use a simple network architecture : {Conv(7,7)-Tanh-Pool(2,2)-Conv(6,6)-Tanh-FC(128)} implemented in `Torch` [75] with the following simplified training process. CNN is trained from $5M$ triplets sampled on-the-fly using patches from [71]. We do not use data augmentation unlike in typical CNNs for general classification or convolutional feature descriptors from [8][64]. When forming a triplet for training we choose randomly a positive pair of patches that originate from the same physical point and a randomly sampled patch from another keypoint. This is in contrast to other works where carefully designed schemes of choosing

the training data are used in order to enhance the performance [64, 66]. For the optimization the Stochastic Gradient Descend [76] is used, and the training is done in batches of 128 items, with a learning rate of 0.1 which is temporally annealed, momentum of 0.9 and weight decay of 10^{-6}. We also reduce the learning rate every epoch. The convolution methods are from the NVIDIA cuDNN library [77]. The training of a single epoch with $5M$ training triplets takes approximately 10 minutes in an NVIDIA Titan X GPU.

It is worth noting that the CNN used in our experiments consists of only two convolutional layers, while all of the other state-of-the art deep feature descriptors consist of four or more layers [8, 63, 64]. Our motivation for such shallow network is to develop a descriptor for practical applications including those requiring real time processing. This is a challenging goal given that all previously introduced descriptors are computationally very intensive, thus impractical for most applications. This design is also inspired by the approach introduced in [61], where pooling of the responses of Gaussian filters and a simple linear projection produced very good results. Thus, we build a simple hierarchical network that is based on 100 convolutional filters, followed by a linear transformation that projects the responses of the filters to the desired output dimensionality. Several other implementation variants are possible such as different non-linearity layers (e.g. ReLU as in [8, 64]), extra normalization layers, or multiscale architectures but these are likely to further improve the results and are beyond the scope of this work. We provide all the learned models and the training code for all the variants in GitHub[2].

2.3.4 Description Results

In this section we evaluate the proposed local feature descriptor within the two most popular benchmarks in the field of local descriptor matching (patch pair classification and nearest neighbour patch matching) and we test on different datasets to show that it can generalise well. We compare our method to SIFT [58], Convex optimization [61] and the recently introduced convolutional feature descriptors MatchNet [64], DeepCompare [8] and DeepDesc [63], which are currently the state of the art in terms of matching accuracy. The original code was used in all the experiments. More details can be found in the supplementary materials. We name our four variants `TFeat-ranking` for the networks learnt with the ranking loss, `TFeat-ranking*` for the networks learnt with the ranking loss with anchor swap, `TFeat-ratio` for the ratio loss, and `TFeat-ratio*` for the ratio loss with anchor swap.

[2] https://github.com/vbalnt/tfeat

Note that for a fair comparison, we do not use the multi-scale *2ch architectures* from [8]. Multi-scale approaches use multiple patches from each example, with extra inputs in form of cropped sub-patches around the center of each patch. This introduces information from different samples in the scale-space and it has been shown to lead to significant improvements in terms of matching accuracy [70]. Such approach can be used for various descriptors (e.g. MatchNet-2ch, TFeat-2ch, DeepDesc-2ch). The evaluation is done with two different evaluation metrics frequently found in the literature, patch pair classification success in terms of ROC curves [60], and mean average precision in terms of correct matching of feature points between pairs of images [69]. Note that these two metrics are of very different nature, the former measures how succesfull a classification of positive and negative patch pairs is, and the latter is evaluating the performance of a descriptor in nearest neighbour matching scenario where the task is to find correspondences in two large sets of descriptors.

Patch pair classification benchmark

The evaluation procedure designed in this benchmark measures the ability of a descriptor to discriminate positive patch pairs from negative ones in the Photo Tour dataset [71]. This dataset consists of three subsets *Liberty*,*Yosemite* & *Notredame*, with each containing more than 500k patch pairs extracted around keypoints. We follow the protocol proposed in [71] where the ROC curve is generated by theshold-ing the distance scores between patch pairs. The number reported here is the false positive rate at 95% true positive rate (FPR95), as used in many influencial works in the field. For the evaluation we use the $100K$ patch pairs proposed as defined in the benchmark. For the training we use two out of the three subsets for training and the remaining for testing. Note that some methods such as DeepDesc [63], does not report performance with training based on a single dataset, therefore for each test set, the training is performed on the other two datasets.

The results for each of the combinations of training and testing using the three subsets of the Photo Tour dataset are shown in Table 2.5 including the FPR95 average across all possible combinations. Our networks outperform all the previously intro-duced single-scale convolutional feature descriptors, and in some cases with large margins except from one training-test combination where the 4096-dimensional version of MatchNet outperforms our TFeat variants. However, even in this case, the version of MatchNet with comparable dimensionality to our descriptors is out-performed by three of our variants. Also note that MatchNet is specifically designed for patch pair classification, since it also includes a similarity metric layer trained on top of the feature layer.

Training		Not	Lib	Not	Yos	Yos	Lib	
Testing		Yos		Lib		Not		
Descriptor	#							mean
SIFT [58]	128	27.29		29.84		22.53		26.55
ImageNet$_{4conv}$ [62]	128	30.22		14.26		9.64		18.04
ConvexOpt [61]	80	10.08	11.63	11.42	14.58	7.22	6.17	10.28
DeepCompare siam [8]	256	15.89	19.91	13.24	17.25	8.38	6.01	13.45
Deepcompare *siam2stream*	512	13.02	13.24	8.79	12.84	5.58	4.54	9.67
DeepDesc [63]	128	16.19		8.82		4.54		9.85
MatchNet [64]	512	11	13.58	8.84	13.02	7.7	4.75	9.82
MatchNet [64]	4096	8.39	10.88	**6.90**	10.77	5.76	3.87	7.75
TFeat-ratio	128	**8.32**	**10.25**	8.93	**10.13**	**4.12**	**3.79**	**7.59**
*TFeat-ratio**	128	**7.24**	**8.53**	8.07	**9.53**	**4.23**	**3.47**	**6.84**
TFeat-margin	128	**7.95**	**8.10**	7.64	**9.88**	**3.83**	**3.39**	**6.79**
*TFeat-margin**	128	**7.08**	**7.82**	7.22	**9.79**	**3.85**	**3.12**	**6.47**

Table 2.5 – **Patch pair classification results.** Results from the Photo-Tour dataset [71]. Numbers are reported in terms of *FPR95* following state of the art in this field (see text for more details). *Italics* indicate the descriptors introduced here, and **bold** numbers indicate the top performing descriptor. Yos:Yosemite, Lib:Liberty, Not:Notredame.

Nearest neighbour patch matching benchmark

In this benchmark we perform an evaluation in the task of matching two images by putting in correspondence the detected keypoints which is the main problem where descriptors are used to perform scene reconstruction. To measure the nearest neighbour matching performance, we establish correspondence ground truth using the homographies and the overlap error from [69]. We consider two feature points between the two images in correspondence if the overlap error between the detected regions is less than 50%. Note that a region from one image can be in correspondence with several regions from the other image. Each image has an associated set of approximately 1K patches that have been extracted from the keypoints detected using two classic detectors (DoG and HarrisAffine).

The results are presented with precision-recall curves as it was originally proposed in [69]. More specifically, for each patch from the left image we find its nearest neighbour in the right image. Based on the ground truth overlap we identify the false positives and true positives, and generate precision-recall curves. The area under the precision-recall curve is the reported mean-average precision [8, 60, 70]. For this experiment, we use the `vl_benchmarks` [26] library (`vl_covdet` function),

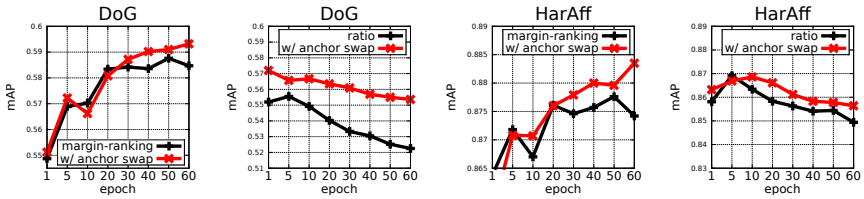


Figure 2.9 – **Descriptor network comparing the different loss function.** Ratio based loss function overfits in the process of separating the positive and negative pairs within a triplet, and does not perform well in the nearest neighbour matching experiment. On the contrary, learning with triplets and margin ranking does not suffer from this problem which shows that ranking methods are more suitable for nearest neighbor matching scenarios.

with some minor modifications to limit the descriptors extracted from an image to 1K, which is important to avoid bias by different numbers of features in different images. For all the experiments below, the descriptors are trained on the Liberty dataset with patches extracted using the DoG detector [71].

For the testing the nearest neighbor matching benchmark two datasets are mainly used in the literature, *Oxford matching dataset* [69], which is of small size, but include images acquired by a camera, and the *generated matching dataset* [62] which is much larger in volume but created synthetically. In the following sections, we discuss our testing results in those two datasets.

Ratio loss vs. margin loss. In this experiment we want to ascertain which triplet loss funtion lead to a better image matching performance. Fig 2.9 shows the performance of the same network trained for the same number of epochs on the Liberty dataset. We report the mAP of image matching in the Oxford dataset. It can be observed that the margin based loss increases the performance as more epochs are used in the training process. No over-fitting is noticed when training and testing patch classification (e.g. training with ratio loss on Liberty and testing on Yosemite or Notredame). Interestingly, the ratio loss seems to decrease the patch matching performance as the network is trained for more epochs. This also hints that other methods from the literature that were only tested in the patch classification scenario, may not perform well in matching. In our view, this shows that evaluating descriptors only in terms of ROC curves is not representative for realistic matching scenarios. Finally, the results show that the loss functions with anchor swapping perform better than without swapping. Note that this simple technique can lead to improved results with no additional computational overhead both in the foward and backward passes.

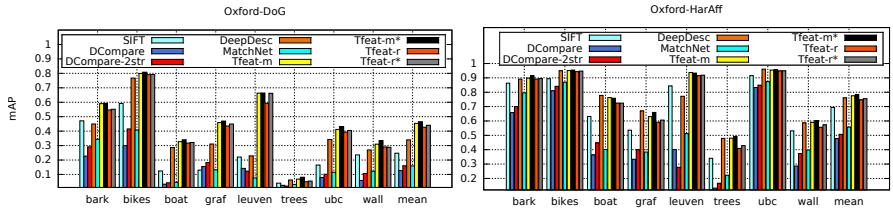


Figure 2.10 – **Keypoint image matching quantitative results on real dataset**. Evaluation on the Oxford image matching dataset [69], for two different types of feature extractors, DoG and HarrisAffine.

Keypoint matching. Figure 2.10 presents the mAP results for Oxford benchmark, across all image sequences from the Oxford dataset, for two different keypoint detectors, DoG and Harris-Affine. However, all networks are trained just on DoG keypoints. In the case of our ratio loss, we use the networks from the first epoch since the performance in general exhibits better results (cf. Fig 2.9). In the case of the DoG keypoints, our networks outperform all the others in terms of mAP. The second best performing descriptor is the DeepDesc descriptor from [63]. We stress again here, that this descriptor was not one of the best performing ones in the pair classification benchmark as shown in Table 2.5. This confirms our findings that the classification benchmark is not a representative measure for the common real-world application of descriptors which often relies on nearest neighbor matching. When using Harris-Affine keypoints our descriptor still outperforms the others, although with a smaller margin.

Figure 2.11 shows the results across various synthetic transformations of image pairs. Our descriptor gives the top scores in most sequences. It is also worth noting, that even though this dataset has some severe deformations as well as nonlinear filtering, the overall performance for both types of feature extractors is higher than for the Oxford dataset. This shows that synthetic deformations are less challenging for descriptors than some real-world changes as the ones found in Oxford dataset.

Efficiency

One of the main motivations behind this work, was the need for a fast and practical feature descriptor based on CNN. The small network trained with triplets that we used in our experiments, is very efficient in terms of descriptor extraction time. We compare the extraction time per patch, averaged over $20K$ patches, of recently introduced convolutional feature descriptors. The extraction is done with NVIDIA Titan X GPU. Our descriptor is 10 times faster than DeepCompare [8], and 50 times faster than MatchNet [64] and DeepDesc [63]. In fact, when running on GPU, we reach speeds of $10\mu s$ per patch which is comparable with the CPU speeds of the fast

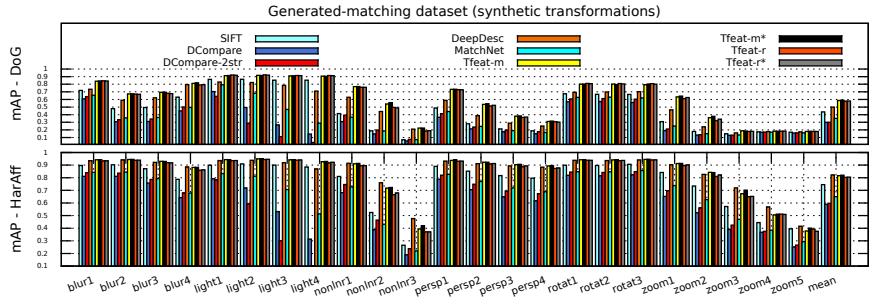


Figure 2.11 – **Keypoint image matching results with synthetic dataset**. Evaluation on the generated-matching dataset [62], for two different types of feature extractors, DoG and HarrisAffine.

binary descriptors[59]. This is a significant advantage over the previously proposed descriptors and makes CNN based descriptors applicable to practical problems with large datasets.

2.3.5 Conclusion

In this section we have introduced a new approach to train a CNN architecture for extracting local image descriptors in the context of patch matching. Similar to the detection algorithm presented in section 2.2, our proposal can be easily included within a scene reconstruction pipeline. The results show that using triplets for training results in a better descriptor and faster learning. Also the descriptor dimensionality is significantly smaller than other CNN based descriptors. We show that due to these properties the proposed network is less prone to over-fitting and has good generalisation properties. In addition, the high computational cost of hard negative mining has been successfully replaced by the very efficient triplet based loss. We also demonstrate that ratio-loss based methods are more suitable for patch pair classification, and margin-loss based methods work better in nearest neighbour matching applications. This indicates that a good performance on patch classification does not necessarily generalise to a good performance in nearest neighbour based frameworks.

3 Differentiable Computer Vision

In the previous chapter we discussed the usage of classical and CNN methods for the tasks to detect and describe local features in a scene reconstruction pipeline. Those tasks can be understood as black boxes inside a pipeline making them trivial to replace in terms of design or executed as separated processes. Thus, having independent processes in a pipeline can be beneficial in some aspects but it is known that deep learning systems work better when the entire pipeline is jointly optimised. To do that, there is a need for adapting pre-existing methods so that can be included inside the same computational graph as the learnable blocks and allow to back-propagate the gradients through the entire pipeline. For this reason, in this chapter we propose a framework that bridges classical computer vision and deep learning under the paradigm of differentiable programming. In addition, the framework eases the transition to implement jointly methods such as end to end pipelines for scene reconstruction as we will see in the last chapter of this thesis.

We present *Kornia*, an open source computer vision library built upon a set of differentiable routines and modules that aims to solve generic computer vision problems. The package uses *PyTorch* as its main backend, not only for efficiency but also to take advantage of the reverse auto-differentiation engine to define and compute the gradient of complex functions. Inspired by *OpenCV*, *Kornia* is composed of a set of modules containing operators that can be integrated into neural networks to train models to perform a wide range of operations including image transformations, camera calibration, epipolar geometry, and low level image processing techniques, such as filtering and edge detection that operate directly on high dimensional tensor representations on graphical processing units, generating faster systems.

3.1 Motivation

Computer vision has driven a lot of technological advances in modern society for many different industries such as Automotion to improve the perception algorithms for self-driving cars; Factory Automation precisely in robotics field; or Audio Visual Production for visual effects generation. One of the key components of this achievements has been thanks to open source software that provided to the community free and accessible implementations of the main computer vision and machine learning algorithms.

There exist several open-source libraries widely used by the computer vision community which are tailored to process images using Central Processing Units (CPUs) such as OpenCV [78], scikit-image [79], or Pillow [80] and many others that are optimised for specific use cases. However, nowadays many of the top performing computer vision algorithms rely on deep learning models, with the huge need to process images in parallel using Graphical Processing Units (GPUs) in order to achieve high-performance requirements. Within that context, during the last couple of years many frameworks for deep learning have gained a lot popularity; to mention some of them: PyTorch [81], Tensorflow [82], Caffe [83], MXNet [84], or MatConvNet [85]. In concrete, PyTorch [81] due to its reverse-mode automatic differentiation engine, dynamic computation graph, distributed learning, eager/script execution modes and its intuitive API introduces a different paradigm within the community. PyTorch and its ecosystem provide a few packages to work with images such as its most popular toolkit, *torchvision*, which is mainly designed to perform data augmentation, read popular datasets and implementations of state-of-the-art models for tasks such as detection, segmentation, image generation, and landmark detection. Despite all these virtues, PyTorch is still lacking in implementations for classical vision algorithms using their native tensor data structures and making them efficient to be used on GPUs or any high-permanence device supported by PyTorch.

This thesis introduces *Kornia*, an open source computer vision library built on top of PyTorch that is intended to help students, researchers, companies and entrepreneurs to implement computer vision applications oriented towards deep learning. Our library, in contrast to standard vision frameworks, provides classical and advanced image processing algorithms implemented such that they can be embedded into deep networks. *Kornia* is designed to fill the gap between PyTorch and computer vision communities and it is based on some of the pre-existing open source solutions for computer vision (Pillow Image (PIL), skimage, torchvision, Tensorflow.image), but with a strong inspiration on OpenCV [78]. As shown in Table 3.1, *Kornia*, in contrast to other existing libraries, which are limited to its usage

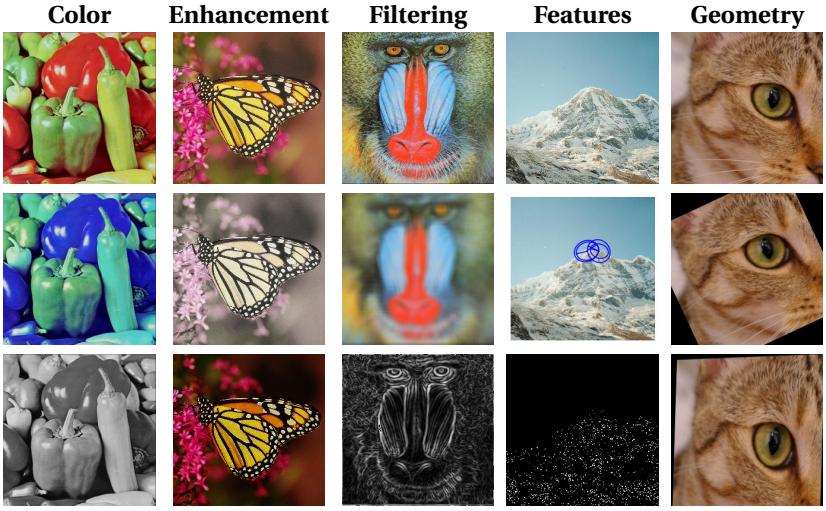


Figure 3.1 – **Kornia computer vision topics overview.** *Kornia* implements routines for low level image processing tasks using native *PyTorch* operators and taking advantage of its high-performance optimizations. The purpose of the library is to be used as a base for large-scale vision projects, data augmentation frameworks, or for creating computer vision layers inside of neural network layers that allow for back-propagating the error through them. The results in this figure are obtained from a given batch of image tensor using data parallelism in the GPU. More examples showing the usage of the library on this specific tasks plus other related to vision are provided in the kornia-examples repository.

in CPU and similar to tf.image making use of differentiability on the GPU, tries to combine the simplicity of OpenCV and PyTorch in order to leverage differentiable programming for computer vision borrowing some properties from PyTorch such as differentiability, GPU acceleration, distributed data-flow and production quality code. Figure 3.1 shows different examples of using the library in several classical computer vision tasks.

In addition to introducing *Kornia*, in this chapter we contribute with some demos showcasing how *Kornia* and its components eases the implementation of several common computer vision tasks like image reconstruction, image registration, depth estimation or local features detection. During the different sections, it is also included an extensive explanation of the different capabilities and algorithms

	CPU	GPU	Batched	Differentiable	ND-Array
scikit-image	✓	×	×	×	×
Numpy and scipy	✓	×	×	×	×
Albumentations	✓	×	×	×	×
torchvision	✓	×	×	×	×
OpenCV	✓	✓	×	×	×
Nvidia Dali	✓	✓	✓	×	×
tf.image	✓	✓	✓	✓	✓
Kornia	✓	✓	✓	✓	✓

Table 3.1 – **Comparison between different computer vision libraries.** *Kornia* and tensorflow-image are the only frameworks that fully support batched operators and differentiable on the GPU.

that can be found in each of the different modules of the library, including short coding examples and links with fully working examples; we also include some additional experiments to benchmark against other frameworks and evaluate its usage depending on the batch size; and finally, we showcase an example about how easily can *Kornia* be integrated within an end to end training system.

The rest of the chapter is organized as follows: we review the state of the art in terms of open source software for computer vision and machine learning in Section 3.2; Section 3.3 describes the design principles of the proposed library and all its components, and Section 3.5 introduces use cases that can be implemented using the library's main features. *Kornia* is public available in GitHub[1] with an Apache License 2.0 and can be installed in any Linux, MacOS or Windows operating system, having PyTorch as a single dependency, through the Python Package Index (PyPI) using the following command:

```
pip install kornia
```

3.2 Related work

In this section we review the state of the art for computer vision software libraries. Related works will be divided in two main categories: classical computer vision and deep learning oriented frameworks. The former are focused on the very first

[1] https://github.com/kornia/kornia

libraries that implement mostly algorithms optimized for the CPU, and the second category targets solutions for GPU.

3.2.1 Classical computer vision libraries

Nowadays, there is a wide variety of options for frameworks that implement computer vision algorithms. However, during the early days of computer vision, it was difficult to find any available and free accessible software for image processing algorithms. All existing software for computer vision was mostly developed within universities or at small teams in companies, and it was not shipped in any form nor released to the public domain. An example of this type of proprietary software specific for Machine Vision, was the Matrox Imaging Library (MIL) [86].

It was not until Intel released the first version of the Open Source Computer Vision Library (OpenCV) that changed the paradigm for the existing Computer Vision software by that time making it accessible for everyone. OpenCV [78] initially implemented computer vision algorithms for real-time ray tracing, visual interfaces and 3D display walls. All the algorithms were made available with a permissive library not only for research, but also for production and commercial usage. OpenCV changed the paradigm within the computer vision community given the fact that most state of art algorithms in computer vision were now put in an common framework written very efficiently in C, becoming in that way a reference within the community.

The computer vision community shifted to improving or besting existing algorithms and started sharing their code with the community. This resulted in new code optimized mostly for CPU. Vedaldi et al. introduced *VLFeat* [26], an open source library that implements popular computer vision algorithms specializing in image understanding and local features extraction and matching. VLFeat was written in C for efficiency and compatibility, with interfaces in MATLAB. For ease of use, it supported Windows, Mac OS X, and Linux, and has been a reference due to their efficient implementations for local features extraction and matching such as Fisher Vector [87], VLAD [88], SIFT [89], and MSER [90].

MathWorks released a proprietary Computer Vision Toolbox inside one of its famous product MATLAB [91] that covered many of the main computer vision, 3D vision, and video processing algorithms which has been used by many computer vision students and researchers becoming quite standard within the researcher community. The computer vision community have been using MATLAB for some decades, and many still use it.

Over time, new projects such as Numpy [92] which includes powerful N-dimensional array objects manipulation and very optimised linear algebra support, started to position themselves as the angular stone for scientific computing within the Python

language community. Another example is Scikit-learn [93] that with same philosophy as Numpy, partially implements machine learning algorithms for classification, regression and clustering including support vector machines, random forests, gradient boosting and k-means. A similar project, Scikit-image [79] implements open source collections of algorithms for image processing making it compatible with this new group of Python scientific packages.

3.2.2 Deep learning and computer vision

Computer vision frameworks have been optimized for CPU to fulfill realtime requirements for different applications, however, the recent success of deep learning changed the way how computer vision system are designed. A. Krizhevsky et al. introduced to the community *Alexnet* [24] continuing the old ideas from Yann LeCun's Convolutional Neural Networks (CNNs) [94] paper with an architecture similar to LeNet-5 and achieved the best results by far in The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 [72] image classification task. This was a breakthrough moment for the computer vision community, and changed the way computer vision was understood.

In terms of deep learning software, new frameworks such *Caffe* [83], Torch [95], MXNet [84], Chainer [96], Theano [97], MatConvNet [85], PyTorch [81], and Tensorflow [82] appeared on the scene with efficient implementations of classical operators for computer vision such as convolutions. All these frameworks included optimisations using the GPU and parallel programming [98] as an approach to handle the need for large amounts of data processing in order to train large deep learning models.

With the rise of deep learning, most standard computer vision frameworks are now being used to perform pre-processing, or data augmentation on the CPU, which for some use cases like volumetric data in medical imaging or multi-spectral data is quite limited due to the need of parallelism to not decrease performance in both, training and inference time. Examples of libraries that are currently used to perform pre- and post-processing on the CPU within the deep learning frameworks are OpenCV or Pillow.

Given that most deep learning frameworks still use standard vision libraries to perform the pre- and post-processing on CPU and similar to Tensorflow.image as Table 3.1 shows, *Kornia* fill this gap within the PyTorch ecosystem by introducing a computer vision library that implements classical computer vision algorithms taking advantage of the different properties that modern frameworks for deep learning like PyTorch can provide.

3.3 *Kornia*: Computer Vision for PyTorch

Kornia[2] can be defined as a computer vision library for PyTorch, inspired by OpenCV and with strong GPU support. *Kornia* allows users to write code as if they were using native PyTorch providing high level interfaces to vision algorithms computed directly on tensors. In addition, some of the main PyTorch features are inherited by *Kornia* such as a high performance environment with easy access to automatic differentiation, executing models on different devices (CPU, GPU or Tensor Porcessing Unit – TPU), parallel programming by default, communication primitives for multiprocess parallelism across several computation nodes and code ready for production. In the following, we elaborate on these properties.

Differentiable. Any image processing algorithm that can be defined as a Direct Acyclic Graph (DAG) structure can be incorporated in a neural network and can be optimized during training, making use of the reverse-mode [99] auto-differentiation [100], compute gradients via backpropagation [101]. In practice, this means that computer vision functions are operators that can be placed as layers within the neural networks for training via backpropagating through them.

Transparent API. A key component in the library design is its easy way to seamlessly add hardware acceleration to your program with a minimum effort. The library API is agnostic to the input source device, meaning that the algorithms can either be executed in several device type such as CPU, GPU or the recently introduced TPU.

Parallel programming. Batch processing is another important feature that enables to run vision operators using data parallelism by default. The assumption for the operators is to receive N-channel image tensors as input batches, contrary to standard vision libraries with single 1-3 channel images. Hence, working with multispectral, hyperspectral or volumetric image can be done in a straight-forward manner using *Kornia*.

Distributed. It provides support for communication primitives for multi-process parallelism across several computation nodes running on one or more group of local or cloud based machines. The library design allows users to run their applications in different distributed systems, or even able to process large vision pipelines in an efficient way.

Production. Since version v1.0.0, PyTorch has the feature to serialize and op-

[2] https://kornia.org

timize models for production purposes. Based on its just-in-time (JIT) compiler, PyTorch traces the models, creating *TorchScript* programs at runtime in order to be run in a standalone C++ program using kernel fusion to do faster inference. This makes our library a perfect fit also for built-in vision products.

What *Kornia* is NOT

Kornia aims to be a reimplementation for OpenCV for research purposes in the sense that mimics some of the main functionalities adding the ability to backpropagate through the different operators. However, note that Kornia does not seek to be a replacement for OpenCV since it is not optimised for production purposes or to achieve high-performance in embedded devices. Even though the project is backed up by the OpenCV.org, there is no intention to merge in any form both projects in the mid-term. Kornia can be understood as a set of tools for training neural networks to be later used in production using other optimised frameworks.

3.3.1 Library structure

The internal structure of the library is designed to cover different computer vision areas, including color conversions, low level image processing, geometric transformations and some utilities for training such as specific loss functions, conversions between data layouts for different frameworks, or functionalities to easily visualise images and debug models during training. Similar to other frameworks, the library is composed of several sub-modules grouped by generic computer vision topics:

kornia.augmentations: The library provides a fully functional set of routines that can be used to perform data augmentation for training deep networks. This module implements in a high level logic several functionalities found across the other modules. The main feature of this module, and similar to the rest of the library, is that it can perform data augmentation routines in a batch mode, using any supported device, and can be used for backpropagation. Some of the available functionalities which are worth to mention are the following: random rotations; affine and perspective transformations; several random color intensities transformations, image noise distortion, motion blurring, and many of the different differentiable data augmentation policies proposed in [102, 103]. In addition, we include a novel feature which is not found in other augmentations frameworks, which allows the user to retrieve the applied transformation or chained transformations after each cal. For instance, the generated random rotation matrix which can be used later to undo the image transformation itself, or to be applied to additional metadata such as the label images for semantic segmentation, or to the bounding boxes or landmark keypoints

for object detection tasks. This is very valuable for research purposes since it gives the user the flexibility to perform complex data augmentations pipelines. A snippet showcasing a small example is shown in Example 1. In addition, in section 3.4.1 we will review performance benchmarks for this module.

Example 1: Data augmentation pipeline

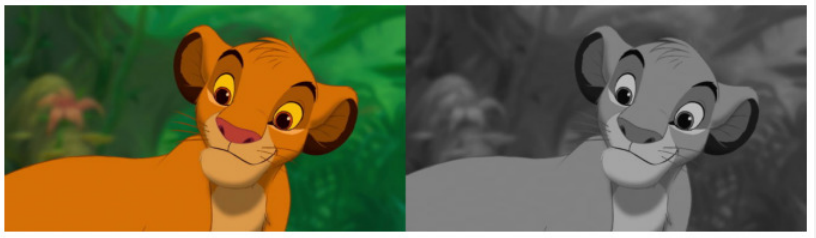
Example showing how flexible is *Kornia* to define a data augmentation pipeline using other PyTorch components. Concretely, this pipeline takes a batch of tensor images and randomly applies a vertical flip, retrieves the applied affine transformation, and finally applies a color jitter transformation based on the user defined preference. The code for this example can be found in the following link.

Code

```
class DataAugmentationPipeline(torch.nn.Module):
    """Module to perform data augmentation using Kornia."""
    def __init__(self, apply_color_jitter: bool = True) -> None:
        super().__init__()
        self._apply_color_jitter = apply_color_jitter
        self.transforms = torch.nn.Sequential(
            K.augmentation.RandomVerticalFlip(
                p=0.5, return_transform=True
            ),
        )
        self.jitter = K.augmentation.ColorJitter(
            brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1
        )

    @torch.no_grad()  # disable gradients for efficiency
    def forward(self, x: torch.Tensor):
        x_out, trans = self.transforms(x)
        if self._apply_color_jitter:
            x_out = self.jitter(x_out)
```

kornia.color: Conversions between color spaces are useful when working with 3-band color images. For this purpose, we provide several functionalities to perform operations that have a similar behaviour as those found in the existing libraries such as OpenCV [78] or Scikit-image [79]. We have introduced small modifications in order to support floating point precision. The functionality found in this module covers operations to map back and forth between the most common color space representations, including Grayscale, RGB, RGBA, BGR, HSV, YCbCr, CIE-XYZ, CIE-Luv or CIE-Lab. In addition, we provide high level interfaces to manipulate color properties to perform intensities normalisation, compute color histograms, or adjust color properties like the brightness, contrast, hue, gamma spectrum, saturation and blending operations to combine different images. We next show in Example 2, an example of how color conversion can be done.

Example 2: Color Space Conversion



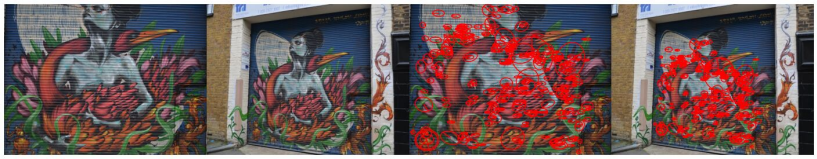
Example showing how to load and decode an image using OpenCV and apply a color space conversion using *Kornia*, and with a torch tensor image representation. The code for this example can be found in the following link.

Code

```
# load image in numpy using OpenCV
img: np.ndarray = cv2.imread("simba.png", cv2.IMREAD_COLOR)

# load image in torch.Tensor
img_bgr: torch.Tensor = K.image_to_tensor(img)
img_rgb = K.bgr_to_rgb(img_bgr)
img_rgb = K.normalize(img_rgb, 0., 255.)

# apply color transforms
img_gray = K.rgb_to_grayscale(img_rgb)
```

kornia.features: Local features detection and description are a key ingredient in a wide range of computer vision algorithms, for e.g. image stitching, structure from motion, or image retrieval. *Kornia* provides operators to detect local features, compute descriptors, and perform feature matching. The module contains differentiable versions of the Harris corner detector [104], Shi-Tomasi corner detector detector [105], Hessian detector [106], their scale and affine covariant versions [107], DoG [89], patch dominant gradient orientation [89] and the SIFT descriptor [89] and recent deep learning based methods such as HardNet [108] or SOSNet [109].

Example 3: Feature detection and Matching



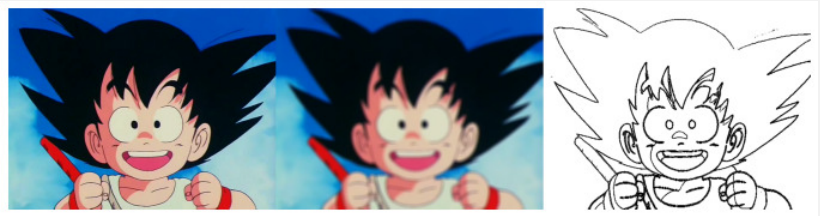
Example showing the use of several components for local feature detection, matching and geometry verification. The code for this example can be found in the following link.

Code

```
# Define local deature detector and descriptor
PS = 32  # patch size
detector = K.ScaleSpaceDetector(num_features=2000)

descriptor = K.HardNet(pretrained=True)

# detect and extract patches
lafs, resps = detector(timg_gray)
patches =  K.extract_patches_from_pyramid(timg_gray, lafs, 32)
descs = descriptor(patches.view(B * N, CH, H, W)).view(B, N, -1)

# Matching
tentatives, scores = K.match_smnn(descs[0], descs[1], 0.95)
kps = KF.laf.get_laf_center(lafs)
kps_tent1 = kps[0:1,tentatives[:,0]]
kps_tent2 = kps[1:2,tentatives[:,1]]

# Finding homography
H = K.find_homography_dlt_iterated(
    kps_tent1, kps_tent2, 1-scores.view(1,-1)
)
```

kornia.filters: Image filtering is another traditional operation in image processing and computer vision used in a number applications, from noise removal to fancy work-arts creation. This module provides operators to perform linear and non-linear filtering operations on tensor images. High level functions to convolve tensors with hand-crafted kernels; for computing first and second order image or n-dimensional tensor derivatives; high level differentiable implementations for blurring algorithms such as Gaussian, Box, Median or Motion blurs; Laplace, and Sobel[110] edges detector. The functionalities found in this module can be either used for creating accelerated computer pipelines or, as we will see in section 3.5.1, to compute loss functions to maintain image properties during reconstructions processes. Example 4 shows a use-case of this functionality.

Example 4: Image filtering and Edge detection



Example about how to load an image using OpenCV, apply 2D Gaussian blur filtering and computing the Sobel edges on a RGB torch tensor image. The code for this example can be found in the following link.

Code

```
# load image in numpy using OpenCV
img: np.ndarray = cv2.imread("goku.png", cv2.IMREAD_COLOR)

# load image in torch.Tensor
img_bgr: torch.Tensor = K.image_to_tensor(img, keepdim=False)
img_rgb = K.bgr_to_rgb(img_bgr).float() / 255.
img_gray = K.rgb_to_grayscale(img_rgb)

# apply a gaussian blur
img_edge = K.sobel(img_gray)
img_blur = K.gaussian_blur2d(img_rgb, (11, 11), (10.5, 10.5))
```

kornia.geometry: Geometric image transformations is another key ingredient in computer vision to manipulate images. Since geometry operations are typically performed in 2D or 3D, we provide several algorithms to work with both cases. This module, the original core of the library, consists of the following submodules: **camera**, **conversions**, **depth**, **epipolar**, **homography**, **linalg**, **subpix**, **transform** and **warp**. We next describe each of them and followed by Example 5 that show a short snippet code to perform a simple image perspective transformation.

- `camera`: A set of routines specific to different types of camera representations such as pinhole or orthographic models containing functionalities such as projecting and unprojecting points from the camera to a world frame.
- `conversions`: Routines to perform conversions between angle representation, pixel coordinates, rotation matrices and quaternions.
- `depth`: A set of layers to manipulate depth maps such as how to compute 3d point clouds given depth maps and calibrated cameras; compute surface normals per pixel and warp tensor frames given calibrated cameras setup.
- `epipolar`: A set of functionalities to work with epipolar geometry and utilites that can be used as a base for Structure from Motion (SfM) or SLAM problems.
- `homography`: An API to work with homography estimation including the Direct Linear Transform (DLT) algorithm with its iterated version towards differentiable RANSAC.
- `linalg`: Functions to perform general rigid-body homogeneous transformations. We include implementations to transform points between frames and for homogeneous transformations, manipulation such as composition, inverse and to compute relative poses.
- `subpix`: A set of operators to perform differentiable sub-pixel coordinates extraction. Useful to work or implement along with geometric based loss functions to regress 2d or 3d coordinates.
- `transforms`: The module provides low level interfaces to manipulate 2d images, with routines for rotating, scaling, translating, shearing; cropping functions in several modalities such as central crops, crop and resize; flipping transformations in the vertical and horizontal axis; resizing operations (see Example 5 below).
- `warp`: An API to perform several types of tensor warping operations, including 2d and 3d affine transformations and utilities to compute such transformations.

Example 5: Geometric Image transformation

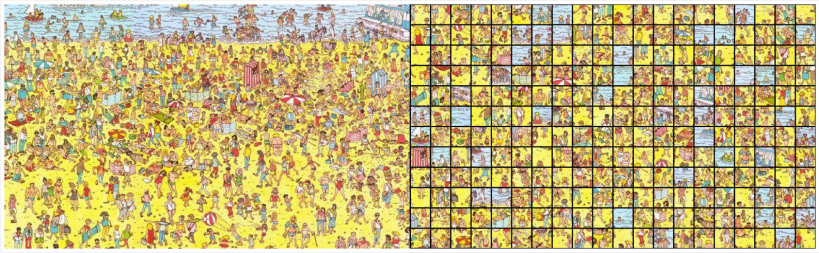
Example showing how to compute the perspective transformation matrix to warp one image into another given four control points. The code for this example can be found in the following link.

Code

```
# the source points are the region to crop corners
points_src = torch.tensor([[
    [125., 150.], [562., 40.], [562., 282.], [54., 328.],
]])

# the destination points are the image vertexes
h, w = img_rgb.shape[-2:]  # destination size
points_dst = torch.tensor([[
    [0., 0.], [w - 1., 0.], [w - 1., h - 1.], [0., h - 1.],
]])

# compute perspective transform
M: torch.tensor = K.get_perspective_transform(
    points_src, points_dst)

# warp the original image by the found transform
img_warp: torch.tensor = K.warp_perspective(
    img_rgb, M, dsize=(h, w))
```

kornia.contrib: Inspired by other libraries, we also include a contrib module to collect experimental operators and user contributions. Currently it contains routines for splitting tensors in blocks (see Example 6), or recent paper implementations such as modules combining learned features with gaussian blurring that can be inserted within the networks to improve stability and robustness against image shifting [111].

Example 6: Extract image patches



Example showing how to split an image tensor into patches, sort the patches in order to create a mosaic. The code for this example can be found in the following link.

Code

```
# load image in numpy using OpenCV
img: np.ndarray = cv2.imread("wally.jpg", cv2.IMREAD_COLOR)

# load image in torch.Tensor
img_bgr: torch.Tensor = K.image_to_tensor(img, keepdim=False)
img_rgb = K.bgr_to_rgb(img_bgr)
img_rgb = K.normalize(img_rgb.float(), 0., 255.)

# extract tensor patches
patches = K.extract_tensor_patches(
    img_rgb, window_size=32, stride=32
)
```

3.4 Performance comparative

In this section we show quantitative and qualitative results on experiments comparing our image processing API compared to other existing image processing libraries. In order to remark the benefits of using Kornia with respect to other computer vision libraries we have measured the cost of processing a batch of images with classical image processing algorithms.

3.4.1 Batched image processing

As stated in section 3.3.1, *Kornia* provides implementations for low level image processing, e.g. color conversions, filtering and geometric transformations that

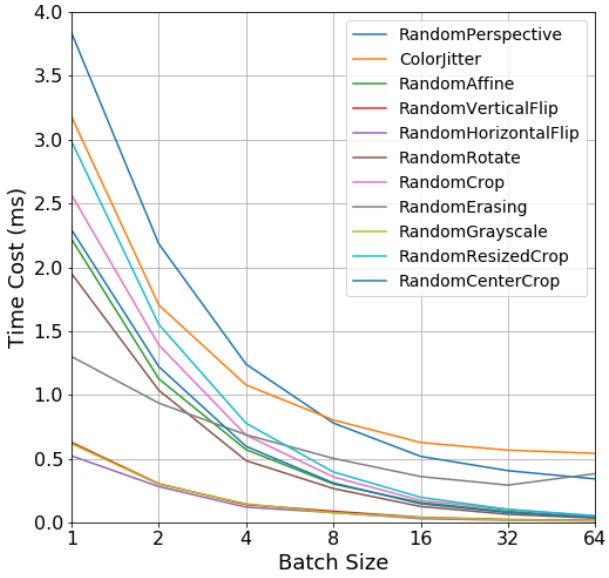


Figure 3.2 – **Operation-wise benchmark respect to other vision libraries.** Operation-wise per-sample timing benchmark with different batch sizes with fixed image size 224x224.

implicitly use native PyTorch operators such as 2D convolutions and simple matrix multiplications, all optimized for different hardware devices. Our API can be combined with other PyTorch components allowing to run vision algorithms via parallel programming, or even sending composed functions to distributed environments.

Although the scope of this library is not to provide explicit optimized code for computer vision we want to show an experiment comparing the performance of our library with respect to other existing vision libraries, namely OpenCV [78], PIL, skimage [79] and scipy [93]. The purpose of this experiment is to give a brief idea of how our implementations compare to libraries that are very well optimized for specific computer vision algorithms. The setup of the experiment assumes as input an RGB tensor of images with a fixed resolution of (256x256), and varying the size of the batch. In this experiment, we first compared different operator across some of the most used vision packages (see figure 3.2), and second, we compute Sobel edges

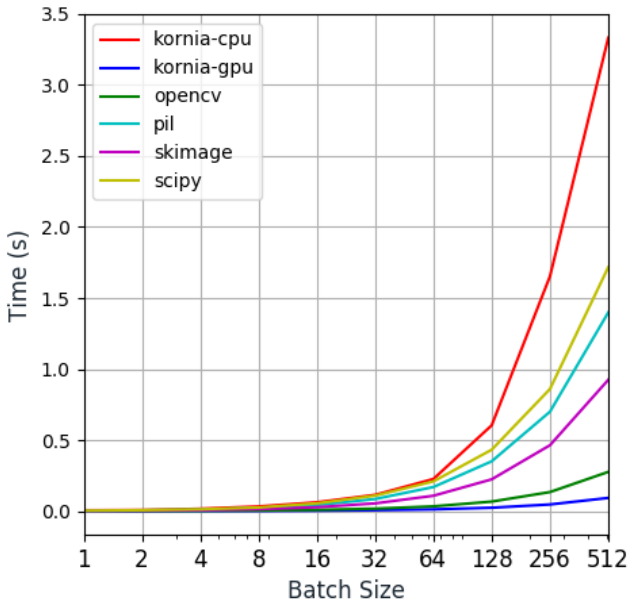


Figure 3.3 – **Sobel edges benchmark compared to other vision libraries.** Results of the benchmark comparing *Kornia* to other state-of-the-art vision libraries. We measure the elapsed time for computing Sobel edges (lower is better).

500 times measuring the median elapsed time between samples (see figure 3.3. The results show that for small batches, *Kornia*'s performance is similar to those obtained using other libraries. It is worth noting that when we use a large batch size, the performance for our CPU implementation is the lowest, but when using the GPU we get the best timing performance. The machine used for this experiment was an Intel(R) Xeon(R) CPU E5-1620 v3 @ 3.50GHz and a Nvidia Geforce GTX 1080 Ti.

In a second experiment we have evaluated the Kornia augmentation module, which is designed and optimized for batch accelerated augmentations. We firstly assessed the operation-level time-efficiency of Kornia and TorchVision with batch size 1, under image sizes ranging from 224x224 to 600x600. As shown in Table 3.2, the time perfomance of Kornia and TorchVision are comparable when process single RGB image.

	Image Size	224	240	260	300	380	456	528	600
Operation	Library								
ColorJitter	Kornia	12.73	**11.33**	**12.13**	13.30	**15.13**	17.46	**19.87**	**22.94**
	TorchVision	**11.36**	12.04	12.64	13.30	15.31	**17.30**	20.18	23.03
RandomAffine	Kornia	**2.22**	**2.14**	**2.22**	**2.35**	**2.27**	**2.21**	2.32	**2.19**
	TorchVision	2.74	2.51	2.61	2.55	2.35	2.22	**2.22**	2.26
RandomCenterCrop	Kornia	**2.29**	**2.26**	**2.25**	**2.38**	**2.31**	2.40	**2.31**	2.40
	TorchVision	2.48	2.48	2.37	2.49	2.44	**2.38**	2.45	**2.34**
RandomCrop	Kornia	**2.57**	**2.57**	**2.57**	2.72	**2.59**	2.64	**2.52**	2.56
	TorchVision	2.90	2.72	2.59	**2.65**	2.60	**2.59**	2.65	2.56
RandomErasing	Kornia	**1.30**	**1.33**	1.46	**1.48**	**1.65**	1.99	2.36	2.68
	TorchVision	1.51	1.35	**1.41**	1.52	1.93	**1.94**	**2.25**	**2.58**
RandomGrayscale	Kornia	**0.62**	**0.56**	**0.60**	**0.56**	0.59	**0.57**	0.74	**0.57**
	TorchVision	0.68	0.59	0.61	0.59	0.59	0.59	**0.72**	0.60
RandomHorizontalFlip	Kornia	**0.52**	**0.54**	0.52	0.58	**0.47**	**0.48**	0.64	**0.48**
	TorchVision	0.56	0.58	**0.47**	**0.48**	0.49	0.49	**0.48**	0.51
RandomPerspective	Kornia	**3.84**	4.02	**4.02**	4.27	**4.34**	**5.05**	5.55	6.25
	TorchVision	4.70	**3.88**	4.08	**3.99**	4.44	5.14	**5.44**	**5.73**
RandomResizedCrop	Kornia	**2.99**	**2.92**	**2.98**	**2.88**	3.05	**3.02**	**2.89**	**2.88**
	TorchVision	3.27	2.97	3.15	2.96	**3.04**	3.04	2.96	2.97
RandomRotate	Kornia	**1.95**	**1.84**	**1.87**	2.01	**1.95**	**1.90**	**1.94**	2.02
	TorchVision	2.08	1.93	2.08	**1.92**	2.01	1.94	2.07	**1.93**
RandomVerticalFlip	Kornia	**0.63**	0.71	0.62	**0.55**	**0.59**	0.57	**0.62**	0.65
	TorchVision	0.70	**0.62**	**0.55**	0.56	0.60	**0.56**	0.65	**0.61**

Table 3.2 – **Performance time comparison of Kornia and TorchVision using different image sizes.** The results are computed as the average time cost (milliseconds) of 10 runs for each operation. Note that both libraries have very similar computation time performances.

We have performed a third experiment aimed to evaluate both from the practical view and in terms of performance the usage of Kornia for data auguemtation purposes. Data augmentation (DA) is a widely used technique to increase the variance of a dataset by applying random transformations to data examples during the training stage of a learning system. Generally, image augmentations can be divided in two groups: color space transformations that modify pixel intensity values (e.g. brightness, contrast adjustment) and geometric transformations that change the spatial locations of pixels (e.g. rotation, flipping, affine transformations). Whilst training a neural network, DA is an important ingredient for regularization that alleviates overfitting problems [112]. An inherent limitation of most current

Color Space Augmentations		
Normalize	Denormalize	ColorJitter
Solarize	Equalize	Sharpness
MixUp	CutMix	Grayscale
2D Spatial Augmentations (on 4d tensor)		
CenterCrop	Affine	ResizedCrop
Perspective	HorizontalFlip	VerticalFlip
Erasing	Rotation	Crop
3D Volumetric Augmentations (on 5d tensor)		
CenterCrop3D	Crop3D	Perspective3D
HorizontalFlip3D	VerticalFlip3D	DepthicalFlip3D

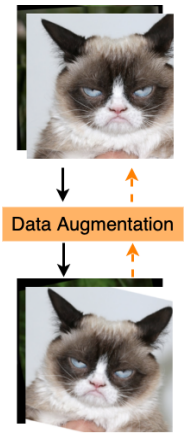


Figure 3.4 – **Left**: Subset of supported differentiable augmentations under Kornia 0.4.1. **Right**: Our proposed scheme to represent differentiable data augmentation showing the gradients flow of the different transformations go forth and back throw the augmentations pipeline. Black arrow represents the forward pass while orange arrow represents backpropagation.

augmentation frameworks is that they mostly rely on non-differentiable functions executed outside the computation graphs.

In order to optimize the augmentation parameters (e.g. degree of rotation) by a specific objective function, differentiable data augmentation (DDA) is used. Earlier works like spatial Transformers [113] formulated spatial image transformations in a differentiable manner, allowing backpropagation through pixel coordinates by using weighted average of the pixel intensities. Recent works proposed to use DDA to improve GAN's training [114], and to optimize augmentation policies [115, 116].

Example 7 shows how to integrate data augmentation as a layer in the computational graph of a deep learning architecture. This approach approach offers the following advantages:

- **Automatic differentiation**. Gradients of augmentation layers could be computed whilst forward pass by taking the advantage of PyTorch autograd engine.
- **Higher reproducibilities**. Augmentation randomness is controlled by PyTorch random state for the reproducible DA under the same random seed. In addition, DA pipeline can be serialized along with any neural networks by simply `torch.save` and `torch.load`.

Example 7: DDA Pipeline

```
import kornia.augmentation as K

class MyAugmentationPipeline(nn.Module):
    def __init__(self):
        super(MyAugmentationPipeline, self).__init__()
        self.mixup = K.RandomMixUp(p=1.)
        self.aff = K.RandomAffine(360, p=0.5)
        self.jitter = K.ColorJitter(0.2, 0.3, 0.2, 0.3, p=0.5)
        self.crp = K.RandomCrop((200, 200))

    def forward(self, input, label):
        input, label = self.mixup(input, label)
        input = self.crp(self.jitter(self.aff(input)))
        return input, label

aug = MyAugmentationPipeline()
```

On-device Computations

```
augmented = aug(images.to('cuda:0'))  # in device cuda:0
augmented = aug(images.to('cuda:1'))  # in device cuda:1
```

Save and Load

```
torch.save(aug, "./saved_da.pt")
aug_restored = torch.load("./saved_da.pt")
```

Our framework provides an easy and intuitive solution to backpropagate the gradients through augmentation layers using the native PyTorch workflow. In any augmentations, `kornia.augmentation` takes `nn.Parameter` as differentiable parameters while `torch.tensor` as static parameters. The following Example 8 shows how to optimize the differentiable parameters (including brightness, contrast, satu-

ration) of `kornia.augmentation.ColorJitter` and backpropagate the gradients based on the computed error from a loss function.

Example 8: Optimizable DA

```
import kornia.augmentation as K
import torch; import torch.nn as nn

t = lambda x: torch.tensor(x); p = lambda x: nn.Parameter(t(x))
torch.manual_seed(42);

images = torch.tensor(img, requires_grad=True)

jitter = K.ColorJitter(
    p([0.8, 0.8]), p([0.7, 0.7]), p([0.6, 0.6]), t([0.1, 0.1]))

out = jitter(images)

loss = nn.MSELoss()(out, images)
optimizer_img = torch.optim.SGD([images], lr=1e+5)
optimizer_param = torch.optim.SGD(jitter.parameters(), lr=0.1)

loss.backward()
optimizer_img.step()
optimizer_param.step()
```

Updated Image

From left to right: the original input, augmented image and gradient-updated image.

Updated Parameters

```
brightness -> [0.8048, 0.8363]      contrast -> [0.7030, 0.7323]
saturation -> [0.5999, 0.5976]      hue      -> [0.1000, 0.1000]
```

Existing libraries such as TorchVision (based on PIL) and Albumentations [117] (based on OpenCV) are optimized for CPU processing taking the advantage of multi-threading. However, our framework is optimized for GPU batch processing that runs in a synchronous manner as a precedent module for neural networks. The different design choices among those libraries determined the difference in the performance

under different circumstances (e.g. hardware, batch sizes, image sizes). As we state in Table 3.3, TorchVision/Albumentations show a better performance when lower computational resources are required (e.g. small image size, less images), while Kornia DDA gives a better performance when there is a CPU overhead. For the performance experiments, we used Intel Xeon E5-2698 v4 2.2 GHz (20-Core) and 4 Nvidia Tesla V100 GPUs. The code for the experiments will be publicly provided to compare against other hardware.

Num. GPUs for Data Parallelism	Comparison Among Different Image Sizes (Kornia / Albumentations / TorchVision) 32x32	224x224	512x512
1	14.28 / **12.07** / 12.33	14.23 / **12.10** / 12.48	14.22 / **12.08** / 12.77
2	15.99 / 16.47 / **14.06**	**12.85** / 12.93 / 13.91	**12.93** / 13.34 / 14.03
3	16.61 / 17.88 / **15.21**	**12.97** / 14.46 / 15.00	**13.08** / 13.96 / 15.36
4	16.87 / 18.99 / **15.66**	**13.32** / 15.38 / 15.94	**13.44** / 15.84 / 16.12

Table 3.3 – **Speed benchmark among DA libraries.** The results are computed as the time cost (seconds) of training 1 epoch of ResNet18 using 2560 random generated faked data. Specifically, DA methods compared are RandomAffine, ColorJitter and Normalize. Batch size is 512 in all the experiments. The add-on GPU memory cost from `kornia.augmentation` is negligible.

3.5 Use cases

This section presents practical examples of the library use for well known classical vision problems demonstrating its easiness for computing the derivatives of complex loss functions and releasing the user of that part. We then provide an end to end training example for low-dimensional embedding application showcasing the usage of image processing functions as loss functions. Next, we describe an example of image registration leveraging on our differentiable warpers. Finally, we provide an example demonstrating the applicability of our differentiable local features implementations to solve a classical wide baseline stereo matching problem.

3.5.1 End to end Low-dimensional embedding

Encoding images into low-dimensional spaces is a well know topic in computer vision that has been studied for many years. The current trend addresses the

Figure 3.5 – **End to end Low-dimensional embedding qualitative results.** Results obtained in the experiment for learning low-dimensional embeddings and using them to decode the original image using the equation loss (3.1). *Row 1:* The original images used as input for the network. *Row 2:* the reconstructed images using $\alpha = 1$ which means that the network optimises for the L1 distance. *Row 3:* the reconstructed images using $\alpha = 0.5$ where the network optimises for the L1 distance at the same time as for the Sobel Edges. We can be appreciate that the images in the last row keep the structure of the edges in the central part of the faces.

problem using Convolutional Autoencoder architectures and its variants, such as Variational Auto Encoders (VAE). However, many of the proposed methods do not take into account image properties in the decoding phase. In this example, we showcase how easily *Kornia* components could allow to introduce these properties while training the networks, and exploit the differentiability property to enforce e.g. robustness in the edges reconstruction, or even make use of the color properties to go from one color space to another and backpropagate the gradients using those constraints.

Implementation. In order to showcase the explained problem, we have taken the architecture from one of the state of the art methods for image encoding *Deep Feature Consistent Variational Autoencoder* [118] slightly modifiying the network to get rid of the variational part. For training, we have chosen a popular dataset for this task - *CelebA* dataset [119] which is made of a set of images with aligned and cropped faces of famous celebrities. The experiment consists in evaluating the network qualitatively in a validation set composed by 30% of the images from the original set. Our baseline to compare with consists of a loss function that optimises

the L1 distance between the input image and the reconstructed by the network in RGB color space. We further evaluate the performance by adding constraints to the network during the learning process. Concretely, in this toy example we propose to add extra losses that aim to keep the Sobel edges between the original and the reconstructed image. As we can see in Figure 3.5, when the network is trained using a single loss, the reconstructed images look more smooth respect to when we train using a composed loss function including the Sobel edges detector as a variable to optimize during the reconstruction. The final loss we used has the following shape:

$$Loss = \alpha | \ I - \hat{I} \ | + (1 - \alpha) * | \ \text{Sobel}(I) - \text{Sobel}(\hat{I}) \ | \tag{3.1}$$

This example opens a full research path to explore all these possibilities towards using very known image properties constraints within our neural network guiding the gradients towards a more realistic solution. The intention of this experiment is not to provide a solution to the problem since it might require a complex grid search parameters search, but instead, to illustrate researchers how to build sophisticated loss functions based on *Kornia* differentiable components for image processing.

3.5.2 Image registration by Gradient Descent

In the following, we show the potential of the library for tasks reasoning about the 2D planar geometry (e.g. marker-based camera pose estimation or spatial transformer networks [120]). *Kornia* provides a set of differentiable operators to perform geometric image transformations such as rotations, translations, scalings, shearings, as well as affine and homography transformation. At the core of the geometry module, we have implemented an operator `kornia.HomographyWarper`, which, based on the homography matric, warps a tensor in the reference frame A to a reference frame B in a very efficient way.

Implementation. The task to solve is image registration using a multi-scale version of the Lucas-Kanade [121] strategy. Given a pair of images I_a and I_b, it optimizes the parameters of the homography H_a^b that minimizes the photometric error between I_b and the transformation of $\hat{I}_b$ denoted as $\omega(I_a, H_a^b)$. Thanks to the Pytorch *Autograd* engine, this can be implemented without explicitly computing the derivatives of the loss function from equation 3.2, resulting in a very compact and intuitive code.

$$\text{Loss} = \sum_{u,v}^{N} \| I_b - \omega(I_a, H_a^b) \|_1 \tag{3.2}$$

The loss function is optimized at each level of a multi-resolution pyramid, from the lower to the upper resolution levels. Figure 3.6 shows the original images,

warped images and the error per pixel with respect to the ground truth warp at each of the scale levels. We use the Adam [122] optimizer with a learning rate of $1e-3$, iterating 200 times at each scale level. As a side note, pixel coordinates are normalized in the range of $[-1,1]$, meaning that there is no need to re-scale the optimized parameters between pyramid levels. The code for this example is provided in the following link.

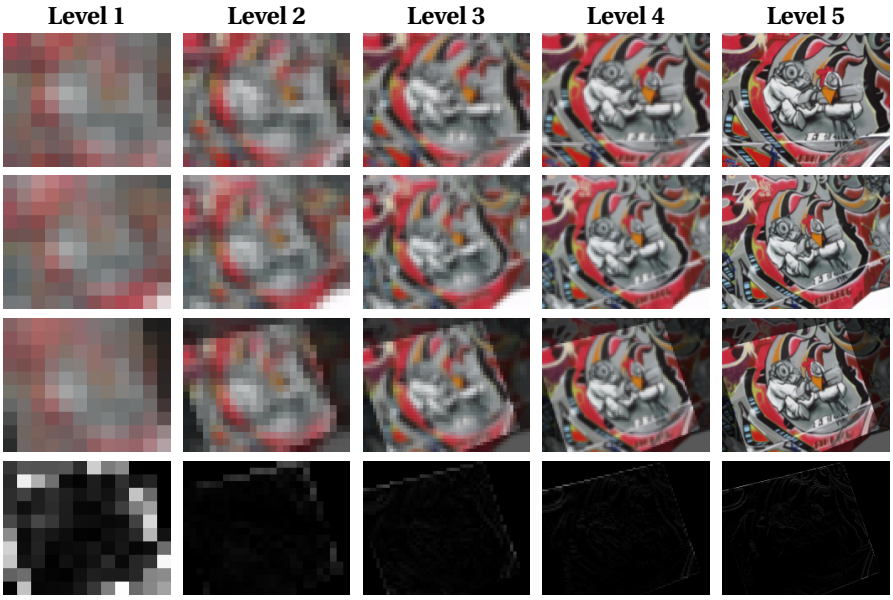


Figure 3.6 – **Results of the image registration by gradient descent.** Each column represents a different level of the image pyramid used to optimize the loss function. *Row 1:* original source image; *Row 2:* original destination image; *Row 3:* source image warped to destination at the end of the optimization loop at that specific scale level. *Row 4:* photometric error between the warped image using the estimated homography and the warped image using the ground truth homography. The algorithm starts to converge in the lower scales refining the solution as it goes to the upper levels of the pyramid.

3.5.3 Multi-View Depth Estimation by Gradient Descent

In this use case we have implemented a fully differential generic multi-view pipeline, using our framework, for machine learning research and applications. For this purpose, *Kornia* provides the `kornia.DepthWarper` operator that takes an arbitrary number of calibrated camera views and warps them to a reference camera frame given the depth in the reference frame.

Multi-view reconstruction is a well understood problem with a good geometric model [123], and many approaches for matching and optimization [124, 125, 126] in addition to recent promising deep learning approaches [127] making use of CNN's to extract robust features in order to create more accurate 3D reconstructions. We have found current machine learning approaches [128, 129] to be limiting, in the sense that they have not been generalized to arbitrary numbers of views (spatial or temporal) being developed for datasets which are only stereo and with low resolution. Moreover, most machine learning approaches assume that there is high quality ground truth depth provided as commonly available datasets, which limits their applicability to new datasets or new camera configurations. Classical approaches such as planesweep, patch match or DTAM [130] have not been implemented with deep learning in mind, and do not fit easily into existing deep learning frameworks.

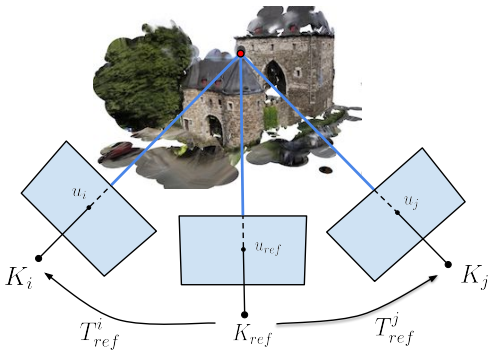


Figure 3.7 – **Classical multi-view stereo cameras setup.** Setup with three cameras and the 2D point projections of a 3D point in the world reference frame. In the setup we describe in the text, we assume the internal parameters K of the cameras are known, as well as their relative homogeneous transformations T^i_j. The subscript $_{ref}$ denotes the reference camera.

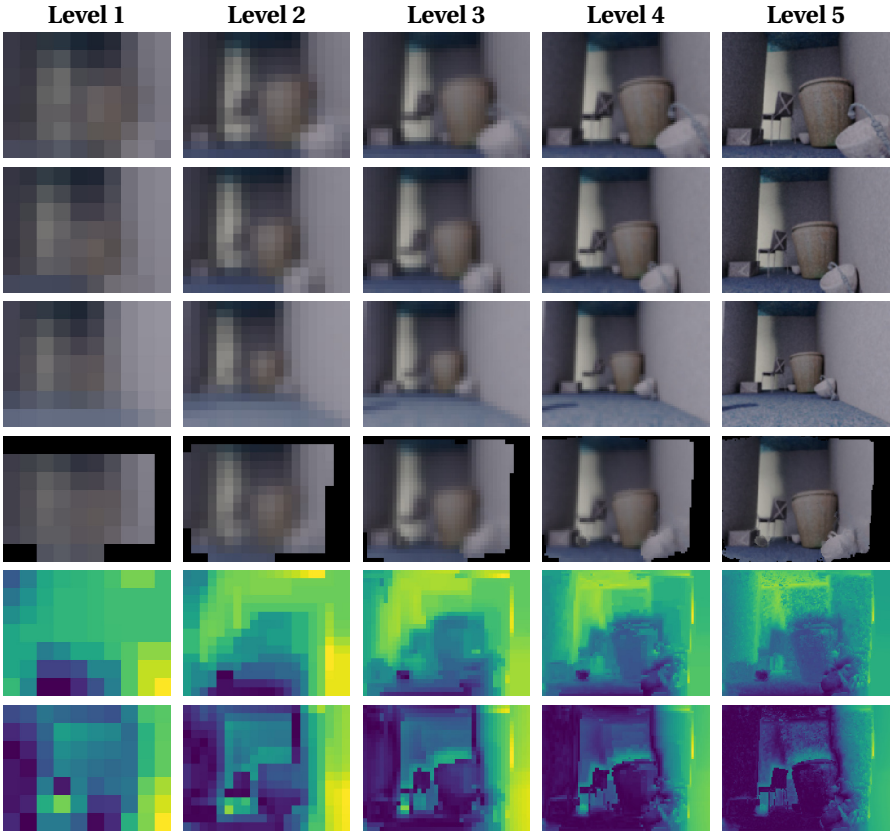


Figure 3.8 – **Results of the depth estimation by gradient descent**. The results show the depth map produced by the given set of calibrated camera images over different scales. Each column represents a level of a multi-resolution image pyramid. *Row 1 to 3:* source images, where the 2*nd* row is the reference view; *Row 4:* images from row 1 and 3 warped to the reference camera given the depth at that particular scale level. *Rows 5 & 6:* estimated depth map and the error per pixel compared to the ground truth depth map in the reference camera. The data used for these experiments was extracted from SceneNet RGB-D dataset [131], containing photorealistic indoor image trajectories.

Implementation. We start with a simple formulation to estimate depth images using gradient descent on a variety of losses based on state of the art classical approaches (photometric, depth consistency, depth piece-wise smoothness, and multi-scale pyramids). The multi-view reconstruction pipeline receives as input a set of views, including RGB images and calibrated intrinsic camera models K_i and relative pose estimates T^i_{ref}. Then, the system estimates the depth image $\boldsymbol{d}_{\text{ref}}$ for a reference view. Since we assume a calibrated setup as shows figure 3.7, the depth value of a given pixel $\boldsymbol{u}_{\text{ref}} = [u_{\text{ref}}, v_{\text{ref}}]$ in the reference view, $\boldsymbol{d}_{\text{ref}}$, can be used to compute the corresponding pixel location $\boldsymbol{u}_i = [u_i, v_i]$ in any of the other views through simple projective geometry $H^i_{\text{ref}} = K_i \cdot T^i_{\text{ref}} \cdot K^{-1}_{\text{ref}}$. Given this, we can warp views onto each other using a differentiable bilinear sampling, as proposed in [132], $\tilde{I_{\text{ref}}} = \omega(I_i, H^i_{\text{ref}}, \boldsymbol{d}_{\text{ref}})$.

Similar to [124, 133, 134], depth is estimated by minimizing a photometric error between the views warped to the reference view:

$$L_{\text{photo1}} = \frac{1}{n}\sum^{n} \frac{1 - \text{SSIM}(I_{\text{ref}}, \tilde{I_{\text{ref}}})}{2} \tag{3.3}$$

$$L_{\text{photo2}} = \frac{1}{n}\sum^{n} |I_{\text{ref}} - \tilde{I_{\text{ref}}}| \tag{3.4}$$

where SSIM is the Structural Similarity Index Measurement used as a loss function.

We compute an additional loss to encourage disparities to be locally smooth with a penalty on the disparity gradients weighted by image gradients as

$$L_{\text{smooth}} = \frac{1}{n}\sum^{n} |\partial_x d| e^{-\|\partial_x I_i\|} + |\partial_y d| e^{-\|\partial_y I_i\|} \tag{3.5}$$

Finally, losses are combined with a weighted sum:

$$L_{\text{total}} = \alpha L_{\text{photo1}} + (1 - \alpha) L_{\text{photo2}} + \lambda L_{\text{smooth}} \tag{3.6}$$

These losses could be easily modified or extended, depending on how well their inherent assumption fit the data, e.g. photometric consistency only holds for small view displacements.

Figure 3.8 shows partial results obtained by the depth algorithm implemented using *Kornia*. The algorithm receives as input 3 calibrated RGB images (320x240). We used Stochastic Gradient Descent (SGD) with momentum and compute the depth at 7 different scales by blurring the image and down-sampling the resolution by a factor of 2 from the previous size. To compute the loss, we up-sample to the original size using bilinear interpolation. The refinement at each level was done for 500 iterations starting from the lowest resolution. The initial values for depth were

obtained by a random uniform sampling in a range between 0 and 1. The code for this example is provided in the following link.

3.5.4 Targeted adversarial attack on SIFT-matching

In this final use case we show how to implement a fully differential wide baseline stereo matching with local feature detectors and descriptors using `kornia.features`. We demonstrate the differentiability by making a targeted adversarial attack on the wide baseline matching pipeline.

Local feature detectors and descriptors are the workhorses of 3d reconstruction [135, 136], visual localization [137] and image retrieval [138]. Although learning-based methods now seem to dominate [139], recent benchmark top-performers still use Difference-of-Gaussians aka SIFT detector [140]. SIFT descriptor is still one of the best for 3d reconstruction tasks [141]. Thus, we believe that community would benefit from having GPU-accelerated and differentiable version of the classical tools provided under `kornia.features`.

Adversarial attacks. Adversarial attacks is an area of research which recently gained popularity after the seminal work of Szegedy et al. [142], showing that small perturbations in the input image can switch the neural network prediction outcome. There exist several works showing that CNN-based solutions for classification [143], segmentation [144], object detection [145] and image retrieval [146] are all prone to such attacks. The only paper about attack on local feature matching is [147], which proposed to place special noisy patches on response peak locations, killing the matching process for matching pairs. Yet, the authors do not know of any paper devoted to targeted adversarial attacks on local features-based image matching. Most attack methods are "white-box" [143], which means they require access to the model gradients w.r.t the input. This makes them an excellent choice for a `kornia.features` differentiability demonstration.

Implementation. The two view matching task is posed as follows [3]: given two images I_a and I_b of the same scene, find the correspondences between pixels in images. This is typically solved by detecting local features, describing the local patches with a descriptor and then matching by minimum descriptor distance with some filtering. *Kornia* provides all these ingredients.

We consider the following adversarial attack: given a non-matching image pair I_a, I_b, and the desired homography H_a^b, modify images so that the correspondence finding algorithm will output a non-negligible number of matches consistent with the homography H_a^b. This means that both local detectors should fire in specific locations and the local patches around that location should be matchable by a given loss function such as:

Figure 3.9 – **Targeted adversarial attack on image matching.** From left to right: original images, which do not match; images, optimized by gradient descent to have local features that match; the result of the attack: matching features (Hessian detector + SIFT descriptor), which survived RANSAC geometric verification

$$L_{\text{total}} = L_{\text{loc}} + \alpha L_{\text{desc}} + \beta L_{\text{reg}} \tag{3.7}$$

$$L_{\text{loc}} = \frac{1}{n}\sum^{n}(p_1 - Hp_2)^2 \tag{3.8}$$

$$L_{\text{desc}} = \frac{1}{n}\sum^{n}(1 + d(D_1, D_2) - d(D_1, D_{2neg})) \tag{3.9}$$

$$L_{\text{reg}} = \frac{1}{n}\sum^{n}(I - I_{\text{init}})^2 \tag{3.10}$$

where p_1 is keypoint detected in I_a, p_2 is closest reprojected by the H_a^b keypoint detected in image I_b, σ_1 and σ_2 are their scales, D_1 and D_2 – their descriptors, D_{2neg} - hard negative in batch, $d(\cdot,\cdot)$ – L2 distance, and I_{init} is original unmodified version of I_a and I_b.

The detector used in this example is the Hessian blob detector [106]; the descriptor is the SIFT [89]. We keep the top-2500 keypoints and use the Adam [122] optimizer with a learning rate of 0.003. Figure 3.9 shows the original images, optimized images and optimized images with matching features visualized. The perturbations are not quite imperceptible, but that it is not the goal of the current

example. The code for this example is provided in the following link.

3.6 Conclusions

In this chapter we have introduced *Kornia*, a library for computer vision in PyTorch that implements traditional vision algorithms in a differentiable manner making use of hardware acceleration to improve performance. We demonstrated that using our library, classical vision problems such as image registration by homography, depth estimation, or local features matching can be very easily solved with a high performance similar to existing libraries making use of GPU acceleration and batch parallelism. By leveraging this project, we believe that classical computer vision libraries can take a different role within the deep learning environments as components of layers of the networks as well as pre- and post-processing of the results. This has been demonstrated in the chapter in a varied set of use cases. In addition, one more reason why combining traditional and deep learning methods through *Kornia* is the fact that it can help to minimize the time-cost of having engineers tuning hyper-parameters for classical vision methods where instead those parameters can be easily optimised for specific tasks making more powerful the use of differentiability. The proposed library has been well received by the community in particular the computer vision researcher that use PyTorch. At the time of the submission of this thesis, *Kornia* has more than 3400 GitHub stars, 350 forks and up to 75 contributors and it's been used by more than 175 research projects.

4 View Synthesis Generation

In chapter 2 we discussed how to approach scene reconstruction using classical and CNN based methods. We first reviewed specific parts of the pipeline that uses local features to detect and describe regions of interest later used to perform the 3d reconstruction. Having a pipeline where you can switch and replace some components eases those cases where you want to parallelize processes, detect errors or even perform offline optimisation of those specific parts. However, as we remarked in chapter 3, the trend for deep learning is to design the entire pipeline in an end to end fashion and optimise all the components of the pipeline under the same optimisation framework. For this reason, we introduced Kornia a framework that provides a wide variety of tools to combine classical vision with deep neural networks. In this chapter we are going focus on the design of a system to perform scene reconstruction trained following the mentioned end-to-end trend, and combining classical computer vision functions of projective geometry inside the system computational graph.

In the next sections we present a novel approach for synthesizing realistic depth maps of deformable clothes from arbitrary views. Given a depth map of a hanging cloth captured from a camera, our model renders the depth map of how this cloth would be seen under a new camera pose, realistically hallucinating the occluded parts and their most likely creases and folds. While view synthesis is a well known problem in the RGB domain, it is still quite unexplored for inferring novel geometry, and specially for highly deformable objects. In this scenario the problem becomes specially challenging due to the large amount of ambiguities and possible shapes. In order to tackle this problem we propose a novel architecture that performs geometry and occlusion reasoning in the feature space and combines it with adversarial learning. Extensive evaluation shows that our approach, while being relatively simple, is able to generate novel views of complex cloth configurations in both simulated and real scenarios.

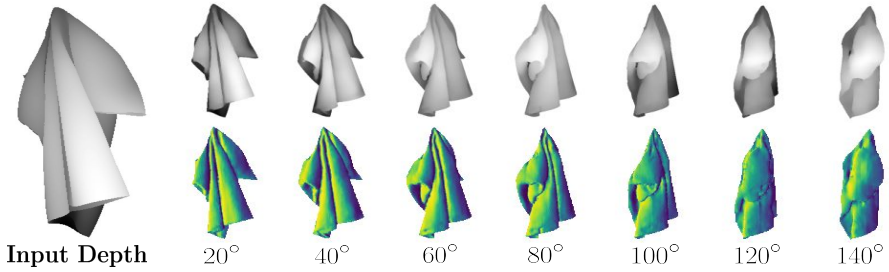


Figure 4.1 – **View Synthesis Generation from an input depth map.** Given a depth map of a deformable cloth (left) and a desired pose defined by a SE(3) rigid transformation, our model generates novel view synthesis preserving the geometry of the input in a high resolution output depth map. In the top row can be seen the depth map estimated by the network while in the bottom is shown the normals map obtained from the predicted depth.

4.1 Motivation

Being able to generate novel depth maps of a deformable cloth under an arbitrary camera view and from a single depth image, can open the door to a number of new exciting applications in different areas, including augmented reality, virtual reality, 3D content production and robotics.

While recent advances in convolutional neural networks [148, 149, 150, 151] and in particular in those architectures based on the implicit function [152, 153] have addressed this problem for rigid objects, articulated objects and clothes under mild deformations, there exist no work in doing so for highly deformable and wrinkled clothes, such as the one shown in Fig. 4.1. This example corresponds to a T-shirt hanging from a point. Observe that the folds and creases seen from the input depth map rapidly disappear when moving away from the frontal view, while new folds appear. The complexity of this problem also invalidates other geometric approaches that have been used for cloth modelling, like [154, 155, 156, 157], which typically use triangular meshes to represent the underlying shape. In order to keep computational complexity bounded, the resolution of these meshes is quite limited, typically a few tens of vertices, preventing to correctly model complex deformations.

In this chapter we propose a novel and efficient network to represent depth maps of complex clothing deformations and generating novel views of it. The main novelty of the network we propose is in its capacity to perform geometry and

occlusion reasoning in the feature space. Concretely, given an input depth map, we initially compute features with an off-the-shelf image encoder-decoder. We then perform projective feature warping according to the desired novel viewpoint we aim to render, and occlusion reasoning, still in the feature domain. We show that these operations, together with an adversarial loss in the spatial domain, allow generating very accurate depth maps, realistically hallucinating the occluded creases and folds.

We train and evaluate the proposed network on synthetic and real data, and in both cases we demonstrate its ability to hallucinate unseen and intricate details of deformed cloth. Additionally, the proposed architecture builds upon convolutional blocks, allowing to be executed very efficiently providing the output in a few milliseconds.

In summary, our main contributions are the following:

- A novel approach to synthesize depth maps of deformable clothes from arbitrary viewpoints.
- A novel network that leverages on geometry and occlusion reasoning in the feature space.
- A new synthetic dataset of highly deformed cloth with ground truth depth maps.

4.2 Related Work

Inferring multiple viewpoints of an object given an arbitrary image of that object is a highly complex task, especially in the case of non rigid surfaces such as clothes. Many techniques estimate accurate 3D triangular meshes of deformable surfaces from either single images [155, 158, 159] (shape-from-template) or video sequences [160, 161, 162, 163] (non-rigid shape from motion). These approaches typically rely on the fact that 2D point correspondences between images can be readily established, which is a strong assumption for highly wrinkled and folded clothes.

Other approaches are more focused on identifying semantically meaningful structures using handcrafted features, to extract generic keypoints [164], grasping points [165, 166], wrinkles [167, 168], edges and corners [169], volumetric features [170] and cloth parts [171]. Unfortunately, in the case of severe deformed clothes these handcrafted features are not always valid or easy to obtain. For instance, given an arbitrary configuration of a deformed cloth, it could be possible that the semantic features that we are searching for lie in the back side of the cloth, hence, would not be visible and our method would fail.

More recent methods, that are based on neural networks [150, 172, 173] or GANs [174] try to take advantage of object symmetries to obtain very good view

estimations or directly reconstruct a whole object using RGB or depth images [148, 149, 151] or depth [175]. However, these methods assume rigid objects or just minor deformations of non rigid objects [156, 157], where a cloth is represented by a low resolution triangular mesh. This is a limited representation for real life clothes. Many clothes, such as shirts, pants, dresses among others have complicated structures with several seams, that can produce very complex deformations, with strong occlusions. Most of the literature, however, tackle a simplified problem and focus on reconstructing folds and wrinkles from clothes worn by a human body model [176, 177, 178, 179], which helps to constrain in great measure the type of folds that are formed on the cloth.

These are not the cases we are interested in this chapter, as we aim to reason about cloth objects that are severely deformed. For instance, as shown in Fig. 4.1, we assume a cloth garment hung from a single point, which generates many wrinkles, folds and self-occlusions. In order to tackle these type of scenarios, we need to go beyond state-of-the-art.

4.3 Our approach

Our framework for estimating the depth of a deformable object from an input depth map and desired rigid transformation is shown in Fig. 4.2. We have designed our architecture based on a *Deep Generative Adversarial Neural Network* scheme which intrinsically uses projective geometry theory to reason about the global geometric aspect of the problem. The network is composed of a primary branch, the *Generator*, that regresses the depth map viewed from a desired pose from which we analytically estimate a normal map and a segmentation mask to separate the object from the background. This principal branch is responsible to learn geometric-aware features using a classic projective geometry pipeline to project learnt features from one view to another. In this same branch, we include another network that combines the original and projected features with and encoding of the applied rigid transformation that is responsible to reason about the potential occlusions and projection order. In addition, we consider a second branch, the *Critic*, that takes the produced depth map and learns to discriminate whether is realistic enough to solve the task and produce more robust solutions to the problem. In the results section we will show that after adapting the projective geometry at the features level, our architecture is able to generate novel synthetic views of deformable cloth depth maps under large camera view changes.

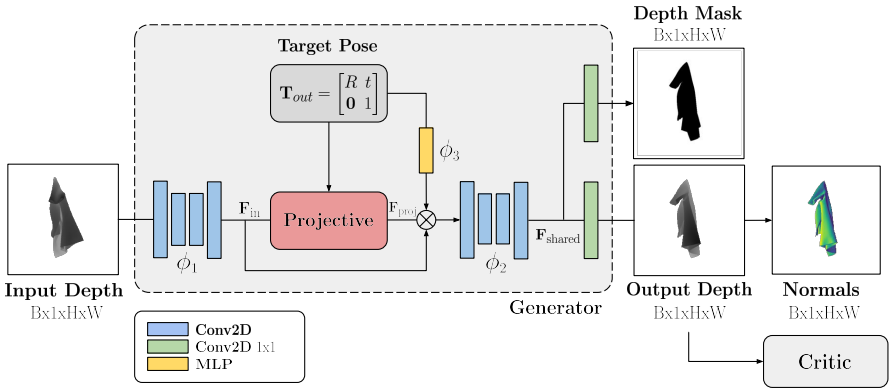


Figure 4.2 – **Proposed architecture for view synthesis generation.** The proposed architecture consists of two main branches that fulfils in a classic *Deep Generative Adversarial Network* scheme. The first, is the *Generator* that has two prediction heads: foreground/background respect to the object and depth regression from which we perform an analytical estimation of the normals. The second, the *Critic* takes the estimated depth map and classifies as a real or fake sample to help for a better generalisation during the depth regression.

4.4 Projective Geometry Network

In this section we give the formulation of the problem that we want to solve and describe the proposed network architecture, which is composed by different components from classical projective geometry. In addition, we also define the different loss functions, including the adversarial training scheme used only during the training of the model.

4.4.1 Problem Formulation

Let $\mathbf{X_w}$ be the 3D world coordinates of a deformed cloth, and $\mathbf{D}_{\text{in}} \in \mathbb{R}^{H \times W}$ the depth map representation when seen from a particular perspective defined by a camera pose $\mathbf{T}_{\text{w}}^{\text{in}} \in \text{SE}(3)$ that relates the mapping of the object 3D coordinates from a source camera to a world coordinates system. We aim at designing a *Deep Generative Neural Network* framework that given an input depth map $\mathbf{D}_{\text{in}}$ and a desired output pose $\mathbf{T}_{\text{w}}^{\text{out}}$, synthesizes a new depth map $\mathbf{D}_{\text{out}} \in \mathbb{R}^{H \times W}$ by inherently projecting the latent but unavailable $\mathbf{X_w}$ to the desired camera reference frame which can be formally described by the following mapping $(\mathbf{D}_{\text{in}}, \mathbf{T}_{\text{out}}) \rightarrow \mathbf{D}_{\text{out}}$.

We assume that the camera calibration parameters are known, namely $\mathbf{K}$ that describes the focal lengths (f_u, f_v), and principal point (c_u, c_v) which are later used to project the object coordinates from the camera frame to the world coordinate system. Our approach is solved in a supervised manner sampling tuples defined by $\{\mathbf{D}^i_{\text{in}}, \mathbf{T}^i_{\text{out}}, \mathbf{D}^i_{\text{out}}\}^N_{i=1}$ from the dataset used both for training and evaluation (different splits). With our method we do not intend to do an explicit reconstruction of the object shape, but instead learn geometric invariant features through a process of features projection from one view to another for later combining and reason about the occlusions.

4.4.2 Network architecture

Given an input depth image $\mathbf{D}_{\text{in}}$, of size $H \times W$, the initial step consists in extracting a set of dense local features per pixel. Following [180] we feed $\mathbf{D}_{\text{in}}$ into a first encoder-decoder network which is configured in such a way that the bottleneck has small downsampling factor of 2. Let us denote these features as $\phi_1(\mathbf{D}_{\text{in}}) \rightarrow \mathbf{F}_{\text{in}}$ of size $F \times H \times W$. The intuition is that these local features will keep information of the 3D local structures in order to later reason about the depth ordering to perform the view projection.

Projective features warping. In order to make the features robust to viewpoint changes, we have designed a differential operator that using simple projective geometry warps the learnt features $\mathbf{F}_{\text{in}}$ to the desired output pose $\mathbf{T}^{\text{out}}_{\text{w}}$. The operator assumes to have a calibrated setup, which using the following equation:

$$\mathbf{H}^{\text{out}}_{\text{in}} = \mathbf{K} \cdot \mathbf{T}^{\text{out}}_{\text{w}} \cdot (\mathbf{T}^{\text{in}}_{\text{w}})^{-1} \cdot \mathbf{K}^{-1} \tag{4.1}$$

This expression will give us the transformation to project features from one view to the desired output pose. Following a similar approach as in [181, 182] we can later perform a differentiable bilinear sampling such that:

$$\omega(\mathbf{F}_{\text{in}}, \mathbf{H}^{\text{out}}_{\text{in}}, \mathbf{D}_{\text{in}}) \rightarrow \mathbf{F}_{\text{proj}} \tag{4.2}$$

That is, projects $\mathbf{F}_{\text{in}}$ according to $\mathbf{H}^{\text{out}}_{\text{in}}$, obtaining $\mathbf{F}_{\text{proj}}$. This operator will produce a stack of features of size $F \times H \times W$ that will help later the system to be invariant to viewpoint changes.

Occlusion comprehension. One of the challenges for our method is the way occlusions are solved when we project features between two different views. Initially, our system assumes no knowledge about the explicit shape of the object, in the sense that during the features projection there is no notion about the depth ordering, or in other terms, what is front or behind. Since we want to let our system to reason

about how to handle occlusions, we will take the original and warped features along with an embedding of the desired pose and feed them to a second network:

$$\phi_2(\mathbf{F}_{\text{in}}, \mathbf{F}_{\text{proj}}, \phi_3(\mathbf{T}_{\text{in}}^{\text{out}})) \rightarrow \mathbf{F}_{\text{shared}} \tag{4.3}$$

This expression generates $\mathbf{F}_{\text{shared}}$, which will encode the combined information from the input depth map and its projection. We denote this second network as an auto-encoder with a downsampling factor of 8. A large factor, will force the network to have also a large receptive field so that the captured information comes from the whole scene and make easy to resolve the occluded information.

Task aware prediction. Our system has a pre-final step where $\mathbf{F}_{\text{shared}}$ is fed to the two different network heads. The final heads of the network are small learnt kernels of 1x1 that will specialise for each of the sub-tasks. The first, produces $\mathbf{D}_{\text{out}}$ with the actual regression values of the depth map and from which using analytical methods, we estimate the normal maps. Finally, the second head produces $\mathbf{D}_{\text{mask}}$ that represents a segmentation mask of foreground and background of the object respect to the scene.

Depth critic. In our method we implement a critic network D as in PatchGan [183] that maps from an input depth map to a matrix $\mathbf{Y}_I \in \mathbb{R}^{26\text{x}26}$, where $\mathbf{Y}_I[i, j]$ represents a probability for each patch overlapping $i\,j$ to be a real sample. This critic network enforces high frequency correctness in order to reduce the blurriness of the generated depth maps and make them more close to the ones from the training distribution.

4.4.3 Loss functions

The training of our model is a composition of losses that tries to enforce different aspects of the cloth configuration and preserve implicit consistency on the 3D shape. As described in the previous section, our architecture has two different output heads, the first to predict a depth map and the second a mask. We next describe in detail the different losses used for optimising the model parameters:

Weighted depth regression loss. The main loss function for our model optimises the depth pixels values produced by our model with respect to the ground truth data. We define this cost function in terms of a weighted *Mean Square Error* that will be evaluated on the foreground object:

$$L_{\text{D}} = \frac{1}{n} \sum_t \lambda_t \cdot \|\mathbf{D}_{\text{out}}(t) - \mathbf{D}_{\text{GT}}(t)\|_2^2 , \tag{4.4}$$

where $\mathbf{D}_{\text{GT}}$ is the ground truth depth map and n are the number of foreground

pixels t, i.e. pixels with a depth value. We define λ_t as weighting term which is computed by projecting the original input depth map $\mathbf{D}_{\text{in}}$ to produce a coarse depth map $\mathbf{D}_{\text{coarse}}$ and compare against $\mathbf{D}_{\text{GT}}$ to obtain a mask of the occluded regions. λ_t can be seen as an attention mechanism during the training to make the model to optimise better the parts of the cloth with missing or occluded information.

Binary mask loss. The second loss we consider is a binary classification loss function which accounts for the number of valid pixels in the output depth image. This loss is defined in terms of a *Cross Entropy* using the depth values to the infinity to define the ground truth mask:

$$L_{\text{M}} = -\frac{1}{n}\sum_{t} \mathbf{D}_{\text{mask}}(t) \cdot \log(\hat{\mathbf{D}}_{\text{mask}}(t)) \tag{4.5}$$

where $\hat{\mathbf{D}}_{\text{mask}}$ is the ground truth for the valid depth map, and n the total number of foreground pixels t in the depth images.

Depth gradients regularisation. Predicting the folds in the cloth is a difficult task for the network without an extra supervision. For this reason, we add an extra term loss that will try to constrain the network to preserve the predicted gradients on the depth image with respect to the ground truth. This loss is computed as follows:

$$L_{\text{S}} = \mid \mathit{Sobel}(\mathbf{D}_{\text{out}}) - \mathit{Sobel}(\mathbf{D}_{\text{GT}}) \mid \tag{4.6}$$

where *Sobel* is a differentiable Sobel edge operator [182] that computes the normalised edge image gradients within the depth domain.

Normal maps regularisation. We want our method to be robust under geometric transformations. Similar to the depth gradients regularisation method, we force our loss function to optimise the normal maps computed from the output depth map $\mathbf{D}_{\text{out}}$. Our final loss function, contains an extra term so that the computed normals from the predicted depth are forced to be close from the ground truth ones. The normal loss can be written in terms of a cosine similarity:

$$L_{\text{N}} = \frac{1}{n}\sum_{t} \frac{\mathbf{N}_{\text{out}}(t) \cdot \mathbf{N}_{\text{GT}}(t)}{\max(\|\mathbf{N}_{\text{out}}(t)\|_2 \cdot \|\mathbf{N}_{\text{GT}}(t)\|_2, \epsilon)}\,, \tag{4.7}$$

where $\mathbf{N}_{\text{out}}$ and $\mathbf{N}_{\text{GT}}$ are the Normal maps computed from the depth map prediction and ground truth, respectively. ϵ is a small value to prevent division by zero, and again, n are the number of foreground pixels t. In order to compute the normal maps, we use a differentiable operator [182, 184] that using the camera calibration matrix (referred as $\mathbf{K}$), maps the depth map to a point cloud and from neighbour points computes the 3D spatial gradients. Then, we can compute an approximation

of the normal maps with the cross product of the derivatives at each 3D point.

Adversarial loss. In order to optimize our main network branch, namely the generator G, and force it to produce depth maps that follow the same distribution as the training data, we use a modified version of the standard GAN algorithm [185] proposed by WGAN-GP [186]. Specifically, the original GAN formulation is based on the Jensen-Shannon (JS) divergence loss function and follows a *min-max strategy game* between the generator G and the critic D. The last tries to correctly classify real and fake depth images while the generator will try to foul the critic. This loss is extremely unstable and not continuous with respect to the parameters coming from the generator and can easily saturate the system leading to vanishing gradients in the critic. This issue is solved in WGAN [187] which proposes to replace the JS with the continuous Earth Mover Distance. In order to maintain the Lipschitz constraint, WGAN-GP [186] proposes to add a penalty term for the critic network computed as the norm of the gradients with respect to the critic input. In our work we formulate the loss as follows:

$$L_{\mathrm{A}} = \mathbb{E}_{\mathbf{I}\sim\mathbb{P}_o}[D(G(\mathbf{D}_{\mathrm{in}}))] - \mathbb{E}_{\mathbf{I}\sim\mathbb{P}_o}[D(\mathbf{D}_{\mathrm{out}})] + \lambda_{gp}\mathbb{E}_{\tilde{\mathbf{I}}\sim\mathbb{P}_{\tilde{\mathbf{I}}}}[(||\nabla_{\tilde{I}}D(\tilde{I})||_2 - 1)^2] \tag{4.8}$$

where λ_{gp} is a penalty coefficient.

Total loss. Our final loss is the average of the different losses described before:

$$\mathrm{Loss} = \frac{1}{5}(L_{\mathrm{D}} + L_{\mathrm{M}} + L_{\mathrm{S}} + L_{\mathrm{N}} + L_{\mathrm{A}}) \tag{4.9}$$

4.5 Experimental setup

The model is trained with a synthetically generated dataset, using depth images of size $H \times W = 512 \times 512$. The image features ϕ_1 and ϕ_2 are obtained from an auto-encoder with skipped connections [180] and a downscaling factor of 2 and 8, respectively. The resulting feature maps from $\phi_1(\mathbf{D}_{\mathrm{in}})$ are of size of $F \times H \times W = 32 \times 512 \times 512$.The computed embedding for $\phi_3(\mathbf{T}_{\mathrm{in}}^{\mathrm{out}})$ has also size of 32, which is later expanded and concatenated to every pixel of $\mathbf{F}_{\mathrm{in}}$ and $\mathbf{F}_{\mathrm{out}}$. The head for predicting the output depth map $\mathbf{D}_{\mathrm{out}}$ has a non-linearity TanH to keep the distribution normalized. We use Adam solver [188] with a learning rate of 0.0001 for both the generator and discriminator, $\beta 1 = 0$, $\beta 2 = 0.999$ and a Cosine Annealing learning rate scheduling [189]. The model is trained with a batch size of 8 in a single Nvidia® GTX 1080 for 1 day.

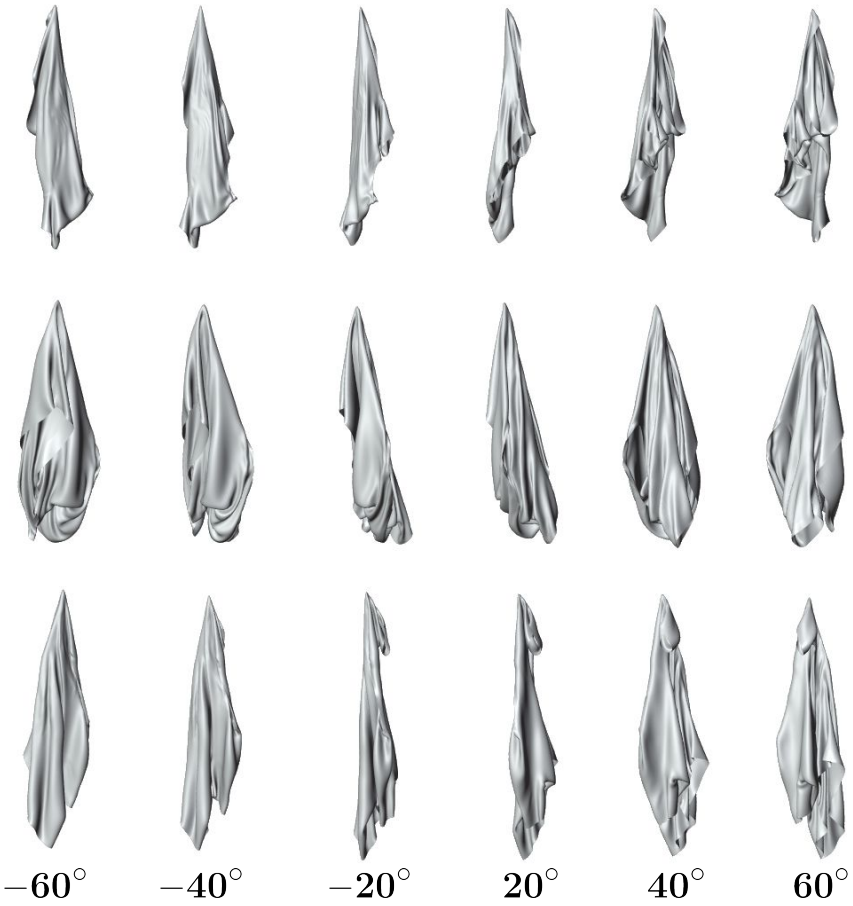


Figure 4.3 – **Sample images of the synthetic dataset.** We build a large dataset of deforming clothes viewed by a ring of 36 depth cameras. The cloth model is hung from a different position (rows) and captured from several cameras (columns).

4.6 Experimental Results

To evaluate the proposed method, we experiment with both synthetic and real datasets. In the following, we first describe our configuration of the two datasets, then explain in detail the training strategy, and finally provide both quantitative and qualitative results.

4.6.1 Dataset

Synthetic data generation. We show in Fig. 4.3 three different examples of the sequences (sampled at 20deg) we produce using the physics cloth engine of Blender [190]. The setup for generating these depth maps consists in a deformed t-shirt model surrounded by a rig of 36 cameras in a circle separated by steps of 10 degrees. Specifically, the bounding box defined by the deformed mesh lies at the center of the circle, and we set the radius to 120cm to ensure the whole t-shirt mesh is completely visible by all cameras.

A 3D human body design suite [191] is used to obtain the t-shirt model. This model is defined by a quad mesh with 3.500 vertices, which is the best topology for the cloth physics engine simulator. The cloth physics engine is based on a spring mass model, which contains several cloth fabric presets as well as several parameters that are tunable for adjusting the behaviour of the simulation. We use the *cotton* preset in the case of the t-shirt, and just modify the bending and stiffness parameters to achieve more realistic deformations. The t-shirt mesh is hung from a point and let deform by gravity. The deformation process is run for 250 steps on each physics simulation to ensure a rest position is achieved. Before running each simulation, the mesh is randomly rotated, and a vertex is also randomly chosen as a hanging point.

The complete dataset consists of 1.000 simulations and for each simulation the depth image, normal image, background segmentation, and intrinsic and extrinsic camera matrices are recorded. We pick 100 simulations for training the network, 30 for validation and 30 for test. There is no overlap between the three splits.

Real data generation. A t-shirt is grasped by a Baxter robot and naturally hung under gravity. We manually adjusted real-world setup to roughly match the appearance and dynamics of the simulation. Specifically, an Intel RealSense L515 camera is placed 120cm away from the grasping point. The robot continuously rotates the t-shirt while depth images are captured in the meanwhile.

The whole real dataset consists of 100 sequences, with 36 depth images (every 10 rotation degree) in each sequence. The clothes are grasped from different points in different sequences. The entire acquisition process takes approximately 16 hours.

We segment the garment from the background by thresholding the depth between 110cm and 130cm. Then, the acquired depth images are cropped to remove the non-garment parts of the image and resized to 512×512 pixels. We use 50 out of 100 sequences for training the network, 15 for validation and 15 for test.

4.6.2 Incremental training

During the study to determine the maximum rotation angle by our system we found that training our model from scratch does not converge to an optimal solution. We experimentally found that the best approach to tackle this problem is using the so well known *Curriculum Learning* training technique and train our system in a incremental manner starting with "easy" data samples and increase the complexity of the task by showing to our model more "difficult" samples with coming views with larger angle rotations.

The process we adopt to sample our training data is as follows: First, a subset of simulations is randomly sampled from the total number of simulations in the dataset. Second, we randomly pick two depth maps for each of the simulation, one shall be the input depth map $\mathbf{D}_{\text{in}}$ and the other the output $\mathbf{D}_{\text{out}}$. We will also annotate the training sample with the rotational angle between the two depth maps.

To train the network, we need several interactions. To be more specific, in the first iteration, we sample source and destination depth which are are most 10 degrees apart. The network is trained until converge and ensuring a validation loss below a certain threshold. Then, in the next iteration, we increase the training dataset by including more samples from rotations up to 20 degrees, and we start the training again initialising the weights with the ones obtained from the previous stage. We repeat this incremental process for 40, 60, 80 ... 180 degrees. We observed that by following this training procedure we can obtain a faster training convergence for large rotation angles.

4.6.3 Quantitative Evaluation

We next evaluate the accuracy of our trained model under different rotation angles using both the synthetically generated and real datasets. In order to perform the evaluation, we compare against *Baseline* which is a coarse depth map obtained by analytically rotating the 3D points of the input depth map to the desired view.

We report the evaluation on the synthetic data using the best performing model. The model has been trained using up to 180 degrees of rotation following the described *Curriculum Learning* approach. For the real data, we have taken the model trained on the synthetic dataset and finetuned using the real training data.

In order to report the evaluation, we use two different metrics: the bidirectional

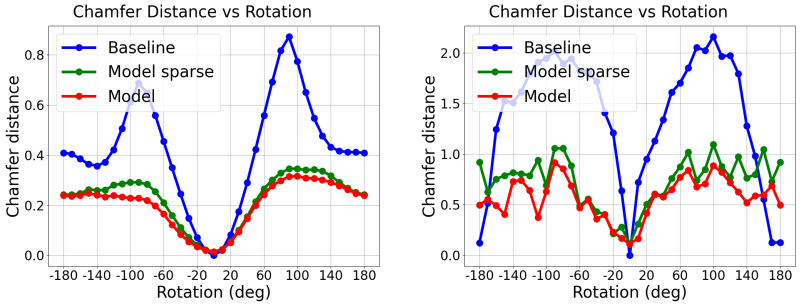


Figure 4.4 – **Evaluation on synthetic and real data for the Chamfer distance.** Evaluation of the network performance when generating novel views from the same input depth map under different rotation angles. *Baseline* is the analytic geometric rotation of the input depth map. *Model* is the result obtained from our network. *Model sparse* is the result of the proposed model, evaluated only on the image pixels that have a valid depth after the projection between the two views. Chamfer distance vs the rotation for synthetic (left) and real (right) data. The lower the better.

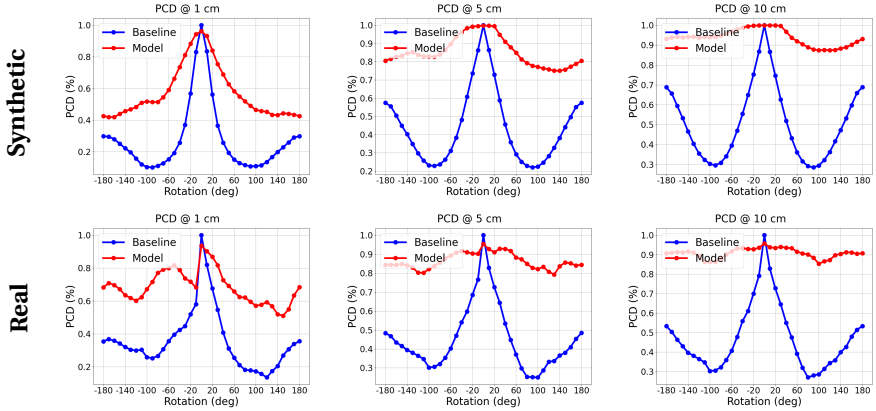


Figure 4.5 – **Evaluation on synthetic and real data for the Percentage Correct Depth.** Similar to the figure 4.4, we show the performance of our network in terms of Percentage Correct Depth (PCD) at different accuracy: [1, 5, 10] cm. The higher the better.

Chamfer Distance and the *Percentage of Correct Depth (PCD)*, which are defined as follows.

Let $\hat{\mathbf{D}}$ be the estimated depth map and $\mathbf{D}_{\text{GT}}$ the ground truth depth map. The bidirectional Chamfer Distance is computed as follows:

$$\text{Chamfer}(\hat{\mathbf{D}}, \mathbf{D}_{\text{GT}}) = \frac{1}{2}(\text{KNN}(\hat{\mathbf{D}} \rightarrow \mathbf{D}_{\text{GT}}) + \text{KNN}(\mathbf{D}_{\text{GT}} \rightarrow \hat{\mathbf{D}})) \tag{4.10}$$

where $\text{KNN}(\hat{\mathbf{D}} \rightarrow \mathbf{D}_{\text{GT}})$ represents the average Euclidean distance for all points of $\hat{\mathbf{D}}$ to their nearest neighbor in $\mathbf{D}_{\text{GT}}$. Note that $\text{KNN}(\cdot,\cdot)$ is not a true distance measure because it is not symmetric. This is why we compute it bidirectionally.

We also asses the accuracy of our method under different error thresholds using the *Percentage of Correct Depth (PCD)*, which represents the percentage of predicted depth pixels with a depth error below a certain threshold. For instance, PCD@k represents the per- centage of predicted depth pixels with a depth error below k.

In Fig. 4.4 and 4.5 we report the described error metrics for both the synthetic and real data experiments. As can be appreciated, our method is able to generate results with a higher accuracy compared to the baseline in all the rotation angles. In the first column experiments we demonstrate that how our model is always below the baseline in terms of the Chamfer Distance error. In this particular case, we also report the error of "Model sparse", in which the error is only evaluated on the (u, v) pixels of the depth image for which the baseline has valid depth values after the projection from the original view. The reported error for "Model sparse", is very similar to the full model performance, which indicates that the model is able to correctly predict both the already seen points, as those that needs to hallucinate.

When considering the PCD metric, our model also consistently imporved the baseline. It is worth noticing that for our model PCD@5cm surpasses always 0.8, that is, at least 80% of the estimated depth points have an error of 5cm or less. The magnitude of this error would give to us enough precision to use the method in some the applications mentioned before such as augmented or virtual reality.

Fig. 4.6 and 4.7 show visual results of the generated data using our best performing model under a range of $[-180, 180]$ degrees of rotation. Note that for large rotations (± 180 degrees) the baseline is affected by gross errors and displays erroneous folds. Our approach, although a bit blurry, is able to recover the main structures of folds and creases.

4.6.4 Ablation study

In the following we evaluate the performance of the network under different configurations (with and without the adversarial loss). The study, similar to the experiments

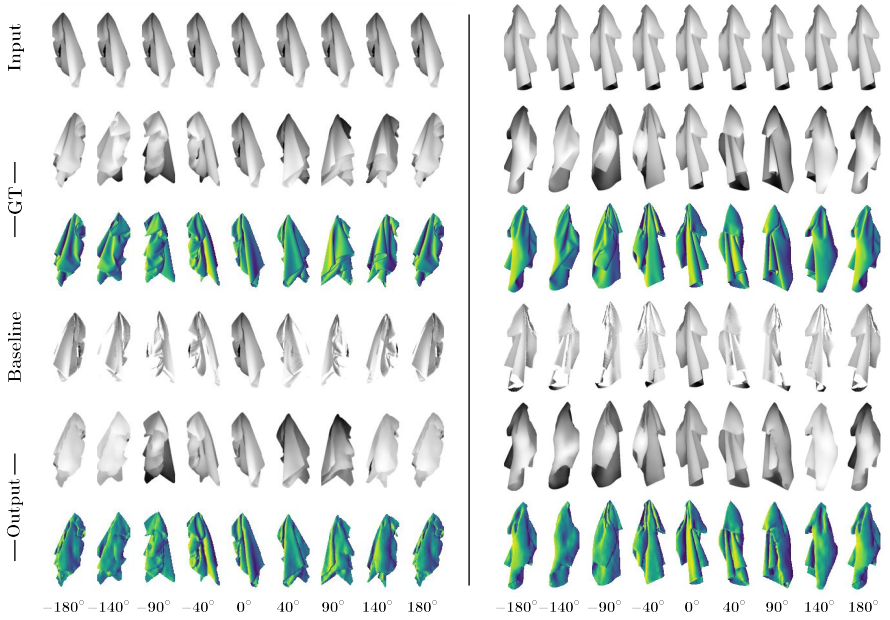


Figure 4.6 – **Qualitative results on synthetic data.** Outputs produced by our model under two different simulations (Left to Right blocks) showing several rotations from -180 to 180 degrees in the Z-axis (Left to Right columns). **Row 1:** Input depth map used to hallucinate the novel views. **Row 2-3:** Ground truth depth and normals maps from the evaluation set. **Row 4:** Baseline computed from the input depth map. **Row 5-6:** Predicted Depth map and the normals computed analytically from the predicted Depth.

discussed in Figure 4.4 reports the *Chamfer distance* metric between ground truth and estimated point clouds and the *Cosine Distance* between normal maps. For further comparison, we also include the performance of the model *Baseline*. As observed in the Table 4.1, the use of the GAN loss brings certain improvement both in the real and synthetic experiments. Our model, with and without GAN, significantly improves the Baseline.

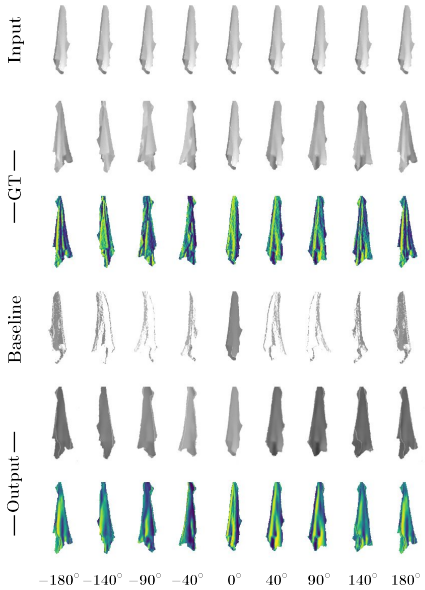


Figure 4.7 – **Qualitative results on real data.** Results obtained with our network on a sequence captured from a real camera setup. Similar to Figure. 4.6, the network produce new views from different rotation in the Z-axis from -180 to 180 degrees. **Row 1:** Input depth map. **Row 2-3:** Ground truth depth and normals maps. **Row 4:** Baseline computed from the input depth map. **Row 5-6:** Predicted Depth map and the normals computed analytically from the predicted Depth.

4.7 Conclusions

In this chapter we have proposed an approach to synthesize novel views of depth maps for deforming clothes. We have shown that encoding the input depth map using spatial-aware features, and geometrically transforming these features, allows hallucinating complex details, like folds and creases, which were not visible in the input depth map. Moreover, we have shown that our method is capable to recuperate many details in very extreme viewpoints where the geometric information is very little, since it is lost by the self-occlusions. We have presented several quantitative results on a synthetic dataset and proof that the proposed method also can work with real data captured by a depth sensor. Finally, the system is based on

Method	Synthetic Data		Real Data	
	Chamfer↓	Cosine↓	Chamfer↓	Cosine↓
Baseline	0.2147 ± 0.11	-	1.4500 ± 0.56	-
Network w/o GAN	0.0568 ± 0.02	0.8320	2.0092 ± 0.23	0.6528
Network with GAN	0.0348 ± 0.01	0.8314	0.5530 ± 0.17	0.6216

Table 4.1 – **Quantitative results under different network configurations.** The table shows the results in terms of *Chamfer distance* mean and standard deviation and the *Cosine Distance* between the normals maps for the Baseline and the model trained with and without the Adversarial loss. The evaluation has been done from -180 to 180 degrees of rotation.

purely convolutional blocks, allowing to perform inference in a few milliseconds and uses the library proposed in chapter 3 to include classical projective geometry techniques within the computational graph.

5 Conclusions and Future work

5.1 Conclusions

In this PhD dissertation we have addressed the problem of scene reconstruction using local features and later as an end to end pipeline including some of the ideas from projective geometry. During this thesis we have reviewed the different approaches proposed during the last decade and contributed with methods that empirically show an improvement over the state of the art. Even though scene reconstruction can be considered as a solved problem in controlled conditions, we have seen during the different chapters of this thesis that there is still space for improvements in each step of the pipeline, or even in the pipeline itself.

In chapter 2, we have faced the reconstruction problem using local features to detect and describe regions of interest to find the image correspondences needed by classical scene reconstruction pipelines. In addition, we have contributed with two different methods that use CNNs for detecting keypoints and compute descriptors from image patches. We have demonstrated through experimental results that both methods can improve the results over pre-existing methods for the specific task of image matching. In addition, we have proposed two differentiable operators to extract the keypoints localization and a triplet loss function based on geometry constraints. Both methods can easily replace keypoint detection and description modules of classical scene reconstruction pipelines, and can be also used in other related problems like SLAM or image mosaicing. The outcome in terms of scientific publications for the work presented in this chapter have been two conference papers [192, 193] in collaboration with the *Imperial College London*.

In chapter 3, we have introduced *Kornia*, our PyTorch based library that eases the design of end-to-end pipelines by implementing classical computer vision algorithms into a differentiable programming paradigm. We have showed different uses cases where *Kornia* has been used to recast classical computer vision algorithms in a differentiable manner. We also provided benchmarks showing the potential of the library to be used as a tool to implement efficiently existing algorithms or data augmentation in high performance devices. The so well adoption of the framework

by the community it is an indicator that having the ability to include the ideas of traditional methods inside deep learning architectures has a lot of potential and incentives the development of new computer vision solutions. The work of this chapter have produced a conference paper [184] in collaboration with the company *Arraiy, Inc.*

In chapter 4, we have contributed with an end-to-end system based on Kornia, devoted to estimate the depth maps of unseen views of clothing. We have experimented with synthetically generated and real data and showed the performance of the method over different degrees of rotation. The obtained results show that including geometric constraints within the end to end network is very beneficial in front of new data and helps with the generalization of the problem. The work on this chapter have been done in collaboration with the *Institut de Robotica Industrial de Barcelona* leading to two different paper submissions [194, 195].

5.2 Discussion and Futures Perspectives

Along the different chapters in this thesis we have presented the different contributions for the specific problem of scene reconstruction. We presented the works showing the transition from methods using classical computer vision methods to smoothly include deep learning components. We would like to describe our experience and findings in a chronological order to give a more reasonable understanding about the *How* and *Why* of the work presented in this thesis.

The aim of this thesis was to understand the scene reconstruction pipeline and for this reason we first analyzed the different solutions that the community was proposing by that time in a more practical point of view. We first got involved and contributed to the *OpenDroneMaps* project which had already implemented an end-to-end pipeline with the classical approach described in chapter 2. The different contributions in this project helped us to have a good understanding of what was happening internally in the scene reconstruction pipeline. At this point, our initial intuition was to take one of the main sub-tasks in the pipeline and use it as an entry point to explore novel solutions for improving the entire pipeline.

As we described in section 2.3, local features descriptors was the first and a relatively easy task where we could completely replace pre-existing solutions in the local features domain by a CNN and improve upon the state the art methods [196]. Our next step was following the same direction to solve small tasks in the reconstruction pipeline which as described in section 2.2, led to the study of local features detectors. However, was at this moment where we started to think about this idea of combining deep learning components with the classical ideas of computer vision which led later to mix handcrafted and learnt deep features. In addition, in the

publication of Key.Net [192] we also contributed with a differentiable operator that could be used to chain the two tasks of detection and descriptor, and potentially be used to design other high level tasks such as SfM or SLAM systems.

With all this ideas in mind of mixing classical computer vision methods with deep learning and combine them in a single framework was one of the main reasons why we started the Kornia [184]project. The use of classical algorithms which already had years of maturity and the fact that can be included within the networks and serve as constraints opens a new field toward differentiable programming or also known as Software 2.0. This same fact also opens the door to Computer Vision 2.0. and helps with the design to improve the most known algorithms in the classical computer vision community. Given that our interest was also in the study of the geometry in the scene reconstruction pipeline, we wanted to prove our hypothesis and marry the ideas of designing a jointly optimizable end-to-end pipeline to combine projective geometry with deep learnt features to reconstruct part of the scene [195].

As a final remark, along this thesis we have seen that deep learning have taken over most of the existing works in the computer vision community. It can be also noticed that the computer vision community has completely switched from classical methods to pure deep learned networks in order to solve tasks with well founded methods. However, through the different contributions in this thesis we have showed that there is still room for the classical ideas to be used along with new methods and technologies. For this reason, we believe that in the long run the combination of classical computer vision and deep learning can be very beneficial for the community and can open a wide variety of research lines and opportunities.

5.3 Scientific Articles

5.3.1 International Conferences and Workshops

The work developed during this thesis has been presented in several international conferences and submitted to two journals.

- **Edgar Riba**, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an Open Source Differentiable Computer Vision Library for PyTorch. In *Winter Conference on Applications of Computer Vision (WACV)*, 2020.
- Vassileios Balntas, **Edgar Riba**, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *BMVC*, 2016.
- Axel Barroso-Laguna, **Edgar Riba**, Daniel Ponsa, and Krystian Mikolajczyk. Key.Net: Keypoint Detection by Handcrafted and Learned CNN Filters. In

ICCV, 2019.

- **Edgar Riba**, Jordi Sanchez-Riera, Yurun Tian, Fan Zhang, Albert Pumarola, Yiannis Demiris, Krystian Mikolajczyk, and Francesc Moreno-Noguer. Novel View Synthesis of Depth Maps for Cloth Manipulation. In *ICRA* 2021 (under review).
- **Edgar Riba**, Jordi Sanchez-Riera, Albert Pumarola, Fan Zhang, Yurun Tian, Yiannis Demiris, Krystian Mikolajczyk, and Francesc Moreno-Noguer. Depth Map Synthesis for Deformable Clothes. In *CVPR* 2021 (under review).
- Jian Shi, **Edgar Riba**, Dmytro Mishkin, and Francesc Moreno-Noguer. Differentiable Data Augmentation with Kornia. In *Neurips 2020 Workshop DiffCVGP*, 2020.

5.3.2 Journals

- **Edgar Riba**, Dmytro Mishkin, Jian Shi, Daniel Ponsa, Francesc Moreno-Noguer, and Gary Bradski. A survey on Kornia: an Open Source Differentiable Computer Vision Library for PyTorch. In *EEAI*, 2020. (under review).
- **Edgar Riba**, Jordi Sanchez-Riera, Yurun Tian, Fan Zhang, Albert Pumarola, Yiannis Demiris, Krystian Mikolajczyk, and Francesc Moreno-Noguer. Novel View Synthesis of Depth Maps for Cloth Manipulation. In *RAL*, 2021 (under review).

5.4 Contributed Code

Among the different activities performed during this thesis, part of the work has been involved in the development of software in different open source initiatives. In the follow I list the main open source projects that I have been involved during the thesis that helped me to get a good understanding about computer vision and deep learning:

- **Kornia**: differentiable computer vision library for PyTorch. I am the project leader and core maintainer of the project and community leader.
  URL: https://github.com/kornia/kornia.
- **tiny-dnn:** a header only, dependency-free deep learning framework in C++14. I maintained this small library that implements deep learning functionalities in C++ and later integrated into OpenCV as part of the Google Summer of Code.
  URL: https://github.com/tiny-dnn/tiny-dnn.
- **TFeat:** Python repository to reproduce the results presented in [193]. I co-maintain the repository with the other authors.

URL: https://github.com/vbalnt/tfeat.
- **TripletLoss:** Python code that implement the margin and the ratio triplet losses in PyTorch from [193]. I originally integrated the triplet loss function in the main PyTorch repository.
- **KeyNet:** Python repository to reproduce the results presented in [192]. I co-maintain the repository with the other authors.
  URL: https://github.com/axelBarroso/Key.Net.

5.5 Scientific Dissemination

In the following there is a list the different talks at conferences and tutorials, appearances in the media, internships and research stays that I have achieved during the duration of this thesis:

5.5.1 Invited Talks and Tutorials

- **Edgar Riba**, Mona Fathollahi, Wesley Chaney, Ethan Rublee and Gary Bradski. torchgeometry: when PyTorch meets geometry. In *PyTorch Developer Conference Poster Session*, 2018
- Vassileios Balntas, Dmytro Mishkin, **Edgar Riba**. Local Features: From SIFT to Differentiable Methods. In, *CVPR 2020 Tutorial*.
- Vassileios Balntas, Dmytro Mishkin, **Edgar Riba**. Local Features: From SIFT to Differentiable Methods. In, *WACV 2020 Tutorial*
- **Edgar Riba** (Organizer). Kornia Hackathon 2019. *Satellite event of Deep Learning Symposium BCN 2019*.
- **Edgar Riba**. Differentiable Computer Vision: an introduction to Kornia. *WACV 2020 Tutorial*.
- **Edgar Riba**. Differentiable Computer Vision: an introduction to Kornia. *GDG Spain 2020, Youtube podcast*.
- **Edgar Riba**. Differentiable Computer Vision: an introduction to Kornia. *Nvidia GTC 2020*.
- **Edgar Riba**. Differentiable Computer Vision: an introduction to Kornia. *IRI Robotics and AI Summer School 2020*.
- **Edgar Riba**. Differentiable Computer Vision: an introduction to Kornia. *PyBCN 2020*.

5.5.2 In the Media

The appearances in the social media and news to disseminate part of the work done in this thesis:

- *How a research scientist built Kornia: an open source differentiable library for PyTorch.* Medium, PyTorch, 2019.
- *Kornia: an Open Source Differentiable Computer Vision Library for PyTorch.* OpenCV Blog, 2020.
- *OpenCV 20th Anniversary Series.* Video2-min6. https://youtu.be/w69BQYgM7xI

5.5.3 Internships

The internships and research stays carried out that led part of the publications presented during the thesis:

- Arraiy, Inc., Mountain View, CA, USA.
  Host: Dr. Gary Bradski
  Date: April 2017-November 2017

- Imperial College London, UK.
  Host: Dr. Krystian Mikolajzcyk
  Date: June 2018-November 2018

- Institut de Robotica Industrial de Barcelona, Barcelona, ES.
  Host: Dr. Francesc Moreno-Noguer.
  Date: September 2019-Present

5.5.4 Community

List of the different open source communities, affiliations and side projects I contributed during the duration of the thesis:

- Kornia.org Project Leader, 2018-Present.
- Active member of the PyTorch community, 2016-Present.
- Technical Committee Member at OpenCV.org, 2018-Present.
- Google Summer of Code Mentor at OpenCV, Summer [2017, 2018, 2019, 2020].
- Google Summer of Code Student at OpenCV, Summer 2016.

Bibliography

[1] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, 1859.

[2] Theo Moons, Luc Van Gool, and Maarten Vergauwen. 3d reconstruction from multiple images: Part 1 - principles. *Foundations and Trends in Computer Graphics and Vision*, 4:287–404, 01 2009.

[3] Philip Pritchett and Andrew Zisserman. Wide baseline stereo matching. In *ICCV*, pages 754–, 1998.

[4] Cordelia Schmid and Roger Mohr. Matching by local invariants, 1995.

[5] Tinne Tuytelaars and Krystian Mikolajczyk. *Local invariant feature detectors: a survey*. Now Publishers Inc, 2008.

[6] D. G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 2004.

[7] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. *ICCV*, 2011.

[8] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[9] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. *CVPR Workshop*, 2018.

[10] Noah Snavely, Steven Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. acm trans graph 25(3):835-846. *ACM Trans. Graph.*, 25:835–846, 07 2006.

[11] S. Ullman. *The Interpretation of Visual Motion*. Artificial Intelligence. MIT Press, 1979.

[12] Martin Peris, Sara Martull, Atsuto Maki, Yasuhiro Ohkawa, and Kazuhiro Fukui. Towards a simulation driven stereo vision system. In *Proceedings of the 21st International Conference on Pattern Recognition, ICPR 2012, Tsukuba, Japan, November 11-15, 2012*, pages 1038–1042. IEEE Computer Society, 2012.

[13] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[14] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks, 2015.

[15] A. Ahmadi and I. Patras. Unsupervised convolutional neural networks for motion estimation. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 1629–1633, 2016.

[16] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *2011 IEEE International Conference on Robotics and Automation*, pages 1817–1824, 2011.

[17] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction, 2017.

[18] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[19] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[20] Ravi Garg, Vijay Kumar B. G, and Ian D. Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. *CoRR*, abs/1603.04992, 2016.

[21] Rui Wang, Stephen M. Pizer, and Jan-Michael Frahm. Recurrent neural network for (un-)supervised learning of monocular videovisual odometry and depth. *CoRR*, abs/1904.07087, 2019.

[22] Punarjay Chakravarty, Praveen Narayanan, and Tom Roussel. GEN-SLAM: generative modeling for monocular simultaneous localization and mapping. *CoRR*, abs/1902.02086, 2019.

[23] Filippo Aleotti, Fabio Tosi, Matteo Poggi, and Stefano Mattoccia. *Generative Adversarial Networks for Unsupervised Monocular Depth Prediction*, pages 337–354. Springer International Publishing, Cham, 2019.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *NIPS*, pages 1097–1105, 2012.

[25] V. Balntas, K. Lenc, A. Vedaldi, and K. Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. *CVPR*, 2017.

[26] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008.

[27] K. Lenc and A. Vedaldi. Large scale evaluation of local image feature detectors on homography datasets. *BMVC*, 2018.

[28] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *ICCV*, 2004.

[29] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. Lift: Learned invariant feature transform. *ECCV*, 2016.

[30] D. DeTone, T. Malisiewicz, and A. Rabinovich. Toward geometric deep slam. *arXiv preprint arXiv:1707.07410*, 2017.

[31] K. Lenc and A. Vedaldi. Learning covariant feature detectors. *ECCV*, 2016.

[32] X. Zhang, X.Y. Felix, S. Karaman, and S. F. Chang. Learning discriminative and transformation covariant local feature detectors. *CVPR*, 2017.

[33] Y. Ono, E. Trulls, P. Fua, and K. Moo Yi. LF-Net: Learning Local Features from Images. *NIPS*, 2018.

[34] K. Moo Yi, Y. Verdie, P. Fua, and V. Lepetit. Learning to assign orientations to feature points. *CVPR*, 2016.

[35] Y. Verdie, K. Yi, P. Fua, and V. Lepetit. Tilde: a temporally invariant learned detector. *CVPR*, 2015.

[36] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *TPAMI*, 2005.

[37] T. Tuytelaars, K. Mikolajczyk, and others. Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, 2008.

[38] C. Harris and M. Stephens. A combined corner and edge detector. *Alvey Vision Conference*, 1988.

[39] P. Beaudet. Rotationally invariant image operators. *ICPR*, 1978.

[40] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *IJCV*, 2005.

[41] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 2004.

[42] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 2008.

[43] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. *ECCV*, 2006.

[44] P. Alcantarilla, A. Bartoli, and A. J Davison. Kaze features. *ECCV*, 2012.

[45] P. Alcantarilla, J. Nuevo, and A. Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *BMVC*, 2013.

[46] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *TPAMI*, 2010.

[47] S. Leutenegger, M. Chli, and R. Y Siegwart. Brisk: Binary robust invariant scalable keypoints. *ICCV*, 2011.

[48] Nikolay Savinov and Akihito Seki. Quad-networks: unsupervised learning to rank for interest point detection. *CVPR*, 2017.

[49] G. Georgakis, S. Karanam, Z. Wu, J. Ernst, and J. Košecká. End-to-end learning of keypoint detector and descriptor for pose invariant 3d matching. *CVPR*, 2018.

[50] L. Florack, B.T.H. Romeny, M. Viergever, and J. Koenderink. The gaussian scale-space paradigm and the multiscale local jet. *IJCV*, 2002.

[51] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. *ICCV*, 2001.

[52] S. Suwajanakorn, N. Snavely, J. Tompson, and M. Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. *NIPS*, 2018.

[53] J. Dong and S. Soatto. Domain-size pooling in local descriptors: Dsp-sift. *CVPR*, 2017.

[54] O. Stepan and J Matas. Object recognition using local affine frames on distinguished regions. *BMVC*, 2002.

[55] O. Russakovsky, J. Deng, Hao Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, 2014.

[56] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *ICCV*, 2015.

[57] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. *NIPS*, 2017.

[58] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[59] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

[60] S. Winder, G. Hua, and M. Brown. Picking the best daisy. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR09)*, Miami, June 2009.

[61] K. Simonyan, A. Vedaldi, and A. Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2014.

[62] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift, 2015.

[63] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. *International Conference on Computer Vision*, 2015.

[64] Xufeng Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3279–3286, 2015.

[65] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[66] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. *CoRR*, abs/1404.4661, 2014.

[67] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. *CoRR*, abs/1412.6622, 2014.

[68] Paul Wohlhart and Vincent Lepetit. Learning Descriptors for Object Recognition and 3D Pose Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[69] K Mikolajczyk and C Schmid. A performance evaluation of local descriptors. *Proceedings of the International Conference on Pattern Recognition*, pages 257–263, 2003.

[70] Jingming Dong and Stefano Soatto. Domain-size pooling in local descriptors: DSP-SIFT. *CoRR*, abs/1412.8556, 2014.

[71] G. Hua M. Brown and S. Winder. Discriminative learning of local image descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[72] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, pages 1–42, April 2015.

[73] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality reduction by learning an invariant mapping. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1735–1742, 2006.

[74] M. Jahrer, Michael Grabner, and Horst Bischof. Learned local descriptors for recognition and matching. In *Proceedings of the Computer Vision Winter Workshop 2008*, pages 39–46, 2008.

[75] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[76] Léon Bottou. Stochastic gradient tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks, Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 430–445. Springer, 2012.

[77] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

[78] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[79] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[80] Fredrik Lundh. The python imaging library (pil), 1995.

[81] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

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

[83] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA, 2014. ACM.

[84] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

[85] A. Vedaldi and K. Lenc. Matconvnet: Convolutional neural networks for matlab. In *ACM International Conference on Multimedia*, 2015.

[86] Matrox. Matrox imaging library (mil), 1976.

[87] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *IJCV*, 105(3):222–245, December 2013.

[88] Herve Jegou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *CVPR*, 2010.

[89] David G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV 2004*, 60(2):91–110, 2004.

[90] Jiri Matas, Ondrej Chum, Martin Urban, and Tomás Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *BMVC*, 2002.

[91] MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[92] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[93] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[94] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[95] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[96] Seiya Tokui, Ryosuke Okuta, Takuya Akiba, Yusuke Niitani, Toru Ogawa, Shunta Saito, Shuji Suzuki, Kota Uenishi, Brian Vogel, and Hiroyuki Yamazaki Vincent. Chainer: A deep learning framework for accelerating the research cycle. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2002–2011, New York, NY, USA, 2019. ACM.

[97] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proceedings of the 9th Python in Science Conference*, pages 3–10, 2010.

[98] Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[99] B. Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana-Champaign, IL, January 1980.

[100] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2008.

[101] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[102] Gaurav Fotedar, Nima Tajbakhsh, Shilpa P. Ananth, and Xiaowei Ding. Extreme consistency: Overcoming annotation scarcity and domain shifts. *CoRR*, abs/2004.11966, 2020.

[103] B. Huang and H. Ling. Deltra: Deep light transport for projector-camera systems. *arXiv:2003.03040*, 2020.

[104] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[105] Jianbo Shi and Carlo Tomasi. Good features to track, 1994.

[106] P. R. Beaudet. Rotationally invariant image operators. In *IJCPR*, pages 579–583, Kyoto, Japan, November 1978.

[107] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *IJCV 2004*, 60(1):63–86, 2004.

[108] Anastasiya Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins: Local descriptor learning loss, 2017.

[109] Yurun Tian, Xin Yu, Bin Fan, Fuchao Wu, Huub Heijnen, and Vassileios Balntas. Sosnet: Second order similarity regularization for local descriptor learning, 2019.

[110] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23(2):358–367, 1988.

[111] Richard Zhang. Making convolutional networks shift-invariant again. In *ICML*, 2019.

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

[113] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015.

[114] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. *arXiv preprint arXiv:2006.10738*, 2020.

[115] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Meta approach to data augmentation optimization. *arXiv preprint arXiv:2006.07965*, 2020.

[116] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. *arXiv preprint arXiv:1911.06987*, 2019.

[117] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[118] Xianxu Hou, LinLin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. *CoRR*, abs/1610.00291, 2016.

[119] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[120] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[121] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *Int. J. Comput. Vision*, 56(3):221–255, February 2004.

[122] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, Dec 2015.

[123] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2 edition, 2003.

[124] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *ICCV*, pages 2320–2327, 2011.

[125] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo - stereo matching with slanted support windows. In *BMVC*, January 2011.

[126] Laura Sevilla-Lara, Deqing Sun, Varun Jampani, and Michael J. Black. Optical flow with semantic segmentation and localized layers. *CoRR*, abs/1603.03911, 2016.

[127] Wenjie Luo, Alexander Schwing, and Raquel Urtasun. Efficient deep learning for stereo matching. In *CVPR*, pages 5695–5703, 2016.

[128] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015.

[129] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, abs/1612.01925, 2016.

[130] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2320–2327, Washington, DC, USA, 2011. IEEE Computer Society.

[131] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J.Davison. Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation. *ICCV*, 2017.

[132] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *NIPS*, pages 2017–2025, 2015.

[133] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, 2017.

[134] Sudeep Pillai, Rares Ambrus, and Adrien Gaidon. Superdepth: Self-supervised, super-resolved monocular depth estimation. *CoRR*, abs/1810.01849, 2018.

[135] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, pages 4104–4113, 2016.

[136] Aji Resindra, Akihiko Torii, and Masatoshi Okutomi. Structure from motion using dense cnn features with keypoint relocalization. *IPSJ Transactions on Computer Vision and Applications*, 10, Dec 2018.

[137] P.-E. Sarlin, C. Cadena, R. Siegwart, and M. Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *CVPR*, 2019.

[138] Tianwei Shen, Zixin Luo, Lei Zhou, Runze Zhang, Siyu Zhu, Tian Fang, and Long Quan. Matchable image retrieval by learning from surface reconstruction, 2018.

[139] Gabriela Csurka and Martin Humenberger. From handcrafted to deep local invariant features. *CoRR*, abs/1807.10254, 2018.

[140] Eduard Trulls. CVPR 2019 Workshop. Image Matching: Local Features & Beyond., 2019.

[141] Johannes Lutz Schönberger, Hans Hardmeier, Torsten Sattler, and Marc Pollefeys. Comparative Evaluation of Hand-Crafted and Learned Local Features. In *CVPR*, 2017.

[142] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

[143] Wieland Brendel, Jonas Rauber, Alexey Kurakin, Nicolas Papernot, Behar Veliqi, Marcel Salathé, Sharada P Mohanty, and Matthias Bethge. Adversarial vision challenge. In *32nd Conference on Neural Information Processing Systems (NIPS 2018) Competition Track*, 2018.

[144] Anurag Arnab, Ondrej Miksik, and Philip H. S. Torr. On the robustness of semantic segmentation models to adversarial attacks. In *CVPR*, 2018.

[145] Shang-Tse Chen, Cory Cornelius, Jason Martin, and Duen Horng (Polo) Chau. Shapeshifter: Robust physical adversarial attack on faster R-CNN object detector. In *ECML-PKDD*, pages 52–68, 2018.

[146] Jie Li, Rongrong Ji, Hong Liu, Xiaopeng Hong, Yue Gao, and Qi Tian. Universal perturbation attack against image retrieval. In *ICCV*, 2019.

[147] Muhammad Latif Anjum, Zohaib Ali, and Wajahat Hussain. Adversarial examples for handcrafted features. In *BMVC*, 2019.

[148] Jacob Varley, Chad DeChant, Adam Richardson, Avinash Nair, Joaquín Ruales, and Peter Allen. Shape completion enabled robotic grasping. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017.

[149] Yichao Zhou, Shichen Liu, and Yi Ma. Learning to detect 3d reflection symmetry for single-view reconstruction, 2020.

[150] Nicolai Häni, Selim Engin, Jun-Jee Chao, and Volkan Isler. Continuous object representation networks: Novel view synthesis without target view supervision. In *Proc. NeurIPS*, 2020.

[151] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems*, 2016.

[152] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. In *Advances in Neural Information Processing Systems 32*, 2019.

[153] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[154] Mathieu Salzmann and Pascal Fua. *Deformable Surface 3D Reconstruction from Monocular Images*, volume 2. Morgan and Claypool, 09 2010.

[155] Albert Pumarola, Antonio Agudo, Lorenzo Porzi, Alberto Sanfeliu, Vincent Lepetit, and Francesc Moreno-Noguer. Geometry-aware network for non-rigid shape prediction from a single view. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[156] Vladislav Golyanik, Soshi Shimada, Kiran Varanasi, and Didier Stricker. Hdm-net: Monocular non-rigid 3d reconstruction with learned deformation model. In *International Conference on Virtual Reality and Augmented Reality*, 2018.

[157] Aggeliki Tsoli and Antonis. A. Argyros. Patch-based reconstruction of a textureless deformable 3d surface from a single rgb image. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, 2019.

[158] Shaifali Parashar, Daniel Pizarro, and Adrien Bartoli. Local deformable 3d reconstruction with cartan's connections. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, PP:1–1, 06 2019.

[159] F. Moreno-Noguer and P. Fua. Stochastic exploration of ambiguities for non-rigid shape recovery. *IEEE transactions on pattern analysis and machine intelligence*, 35(2):463–475, Jan 2013.

[160] Antonio Agudo, Francesc Moreno-Noguer, Begoña Calvo, and J. M. M. Montiel. Sequential non-rigid structure from motion using physical priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:979–994, 2016.

[161] Antonio Agudo and Francesc Moreno-Noguer. Combining local-physical and global-statistical models for sequential deformable shape from motion. *International Journal of Computer Vision (IJCV)*, 122(2):371–387, 2017.

[162] A. Chhatkuli, D. Pizarro, T. Collins, and A. Bartoli. Inextensible non-rigid shape-from-motion by second-order cone programming. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[163] F. Moreno-Noguer and J. M. Porta. Probabilistic simultaneous pose and non-rigid shape recovery. In *CVPR 2011*, pages 1289–1296, 2011.

[164] Edgar Simo-Serra, C. Torras, and F. Moreno-Noguer. Dali: Deformation and light invariant descriptor. *International Journal of Computer Vision*, 115:136–154, 2015.

[165] A. Ramisa, G. Alenya, F. Moreno-Noguer, and C. Torras. A 3d descriptor to detect task-oriented grasping points in clothing. *Pattern Recognition*, 60(C):936–948, 2016.

[166] Andreas Doumanoglou, Tae-Kyun Kim, Xiaowei Zhao, and Sotiris Malassiotis. Active random forests: An application to autonomous unfolding of clothes. In *European Conference on Computer Vision (ECCV)*, 2014.

[167] Ariel Kapusta, Zackory Erickson, Henry M Clever, Wenhao Yu, C Karen Liu, Greg Turk, and Charles C Kemp. Personalized collaborative plans for robot-assisted dressing via optimization and simulation. *Autonomous Robots*, 43(8):2183–2207, 2019.

[168] Luz María Martínez and Javier Ruiz-del Solar. Recognition of grasp points for clothes manipulation under unconstrained conditions. *arXiv*, pages arXiv–1706, 2017.

[169] Christos Kampouris, Ioannis Mariolis, Georgia Peleka, Evangelos Skartados, Andreas Kargakos, Dimitra Triantafyllou, and Sotiris Malassiotis. Multi-sensorial and explorative recognition of garments and their material properties in unconstrained environment. In *IEEE international conference on robotics and automation (ICRA)*, 2016.

[170] Yinxiao Li, Yonghao Yue, Danfei Xu, Eitan Grinspun, and Peter K Allen. Folding deformable objects using predictive simulation and trajectory optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[171] Arnau Ramisa, Guillem Alenya, Francesc Moreno-Noguer, and Carme Torras. Using depth and appearance features for informed robot grasping of highly wrinkled clothes. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[172] Albert Pumarola, Stefan Popov, Francesc Moreno-Noguer, and Vittorio Ferrari. C-flow: Conditional generative flow models for images and 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[173] Haozhe Xie, Hongxun Yao, Xiaoshuai Sun, Shangchen Zhou, and Shengping Zhang. Pix2vox: Context-aware 3d reconstruction from single and multi-view images. In *ICCV*, 2019.

[174] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *The IEEE International Conference on Computer Vision (ICCV)*, Nov 2019.

[175] B. Yang, H. Wen, S. Wang, R. Clark, A. Markham, and N. Trigoni. 3d object reconstruction from a single depth view with adversarial learning. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017.

[176] Erhan Gundogdu, Victor Constantin, Amrollah Seifoddini, Minh Dang, Mathieu Salzmann, and Pascal Fua. Garnet: A two-stream network for fast and accurate 3d cloth draping. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[177] Igor Santesteban, Miguel A. Otaduy, and Dan Casas. Learning-Based Animation of Clothing for Virtual Try-On. *Computer Graphics Forum*, 2019.

[178] Garvita Tiwari, Bharat Bhatnagar, Tony Tung, and Gerard Pons-Moll. Sizer: A dataset and model for parsing 3d clothing and learning size sensitive 3d clothing. In *European Conference on Computer Vision (ECCV)*, 2020.

[179] Chaitanya Patel, Zhouyingcheng Liao, and Gerard Pons-Moll. Tailornet: Predicting clothing in 3d as a function of human pose, shape and garment style. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[180] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[181] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2017–2025. Curran Associates, Inc., 2015.

[182] E. Riba, D. Mishkin, J. Shi, D. Ponsa, F. Moreno-Noguer, and G. Bradski. A survey on kornia: an open source differentiable computer vision library for pytorch, 2020.

[183] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[184] Edgar Riba, Dmytro Mishkin, Dani Ponsa, Rublee Ethan, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Winter Conference on Applications of Computer Vision*, 2020.

[185] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 2672–2680. Curran Associates, Inc., 2014.

[186] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5767–5777. Curran Associates, Inc., 2017.

[187] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[188] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[189] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016.

[190] Blender. https://www.blender.org. Accessed: 2020-10-16.

[191] Makehuman. http://www.makehumancommunity.org. Accessed: 2020-10-16.

[192] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Key.net: Keypoint detection by handcrafted and learned cnn filters. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5835–5843, 2019.

[193] V. Balntas, E. Riba, D. Ponsa, and K. Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. *BMVC*, 2016.

[194] Edgar Riba, Jordi Sanchez-Riera, Yurun Tian, Fan Zhang, Albert Pumarola, Yiannis Demiris, Krystian Mikolajczyk, and Francesc Moreno-Noguer. Novel view synthesis of depth maps for cloth manipulation. In *ICRA (under review)*, 2021.

[195] Edgar Riba, Jordi Sanchez-Riera, Albert Pumarola, Yurun Tian, Fan Zhang, Yiannis Demiris, Krystian Mikolajczyk, and Francesc Moreno-Noguer. Depthmap synthesis for deformable clothes. In *CVPR (under review)*, 2021.

[196] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016.