ScuDo

Scuola di Dottorato ~ Doctoral School

WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation

Doctoral Program in Electronic and Communications Engineering (28^{th}cycle)

Feature Extraction Using MPEG-CDVS and Deep Learning with Application to Robotic Navigation and Image Classification

By

Pedro Porto Buarque de Gusmão

******

Supervisor(s):

Prof. Enrico Magli

Doctoral Examination Committee:

Prof. Carla Fabiana Chiasserini, Politecnico di Torino

Prof. Matteo Cesana, Politecnico di Milano

Prof. Sergio Saponara, Università degli Studi di Pisa

Politecnico di Torino

2017

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Pedro Porto Buarque de Gusmão
2017

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my loving parents

Acknowledgements

The research presented in this thesis has been supported by TIM, former Telecom Italia. Besides thanking my supervisor Prof. Enrico Magli, I would also like to thank my colleagues at the TIM Visible Lab, Skjalg Lepsøy, Gianluca Francini and Massimo Balestri for all the help they provided me and all the good times we have shared. I must also thank my good friend Stefano Rosa for the help he gave me regarding the mechanical aspects of robotics and for the insightful discussions shared with a cup of coffee.

Finally, this work would not have been possible without the support received from my friends, my family and from my girlfriend, Elisabetta Bichiri.

Thank you so much.

Abstract

The main contributions of this thesis are the evaluation of MPEG Compact Descriptor for Visual Search in the context of indoor robotic navigation and the introduction of a new method for training Convolutional Neural Networks with applications to object classification.

The choice for image descriptor in a visual navigation system is not straightforward. Visual descriptors must be distinctive enough to allow for correct localization while still offering low matching complexity and short descriptor size for real-time applications. MPEG Compact Descriptor for Visual Search is a low complexity image descriptor that offers several levels of compromises between descriptor distinctiveness and size. In this work, we describe how these trade-offs can be used for efficient loop-detection in a typical indoor environment. We first describe a probabilistic approach to loop detection based on the standard's suggested similarity metric. We then evaluate the performance of CDVS compression modes in terms of matching speed, feature extraction, and storage requirements and compare them with the state of the art SIFT descriptor for five different types of indoor floors.

During the second part of this thesis we focus on the new paradigm to machine learning and computer vision called Deep Learning. Under this paradigm visual features are no longer extracted using fine-grained, highly engineered feature extractor, but rather using a Convolutional Neural Networks (CNN) that extracts hierarchical features learned directly from data at the cost of long training periods.

In this context, we propose a method for speeding up the training of Convolutional Neural Networks (CNN) by exploiting the spatial scaling property of convolutions. This is done by first training a *pre-train* CNN of smaller kernel resolutions for a few epochs, followed by properly rescaling its kernels to the *target*'s original dimensions and continuing training at full resolution. We show that the overall training time of a *target* CNN architecture can be reduced by exploiting the spatial scaling property of

convolutions during early stages of learning. Moreover, by rescaling the kernels at different epochs, we identify a trade-off between total training time and maximum obtainable accuracy. Finally, we propose a method for choosing when to rescale kernels and evaluate our approach on recent architectures showing savings in training times of nearly 20% while test set accuracy is preserved.

Contents

List of Figures **xi**

List of Tables **xiv**

1 Introduction **1**

2 Simulataneous Localizantion and Mapping **3**

2.1 Visual Odometry 4

2.1.1 Camera Model and Calibration 6

2.1.2 Motion Model 8

2.2 Loop-Closure 9

2.3 Maximum a Posteriori Optimization 9

3 Visual Feature Descriptors **11**

3.1 Scale Invariant Feature Transform (SIFT) 12

3.1.1 SIFT Keypoint Detection 12

3.1.2 SIFT Feature Extraction 13

3.1.3 SIFT Feature Matching 15

3.2 MPEG Compact Descriptor for Visual Search 16

3.2.1 Image Preprocessing 16

3.2.2 Keypoint Detection 17

3.2.3 Keypoint Selection 18
3.2.4 Local Feature Extraction 19
3.2.5 Local Feature Compression 19
3.2.6 Global Descriptor Generation 20
3.2.7 CDVS Feature Matching 20
3.3 Other Visual Features 21

4 Deep Learning for Object Classification 23
4.1 Artificial Neural Networks 24
4.2 Convolutional Neural Networks 26
4.2.1 Convolutional layers 27
4.2.2 Activation Functions 28
4.2.3 Pooling Layer 29
4.2.4 Fully-connected Layers 30
4.3 Modern Architectures 30

5 Training Convolutional Neural Networks for Object Classification 34
5.1 Gradient-based Learning 35
5.1.1 Backpropagation Algorithm 35
5.1.2 Parameters update 40
5.2 Dataset and Network setup 41
5.2.1 Dataset Division and Preprocessing 41
5.2.2 Regularizers 42
5.2.3 Datasets for Image Classification 43
5.3 Speeding Up CNN Training 44
5.3.1 Convolution Operations 44
5.3.2 Network Architecture 46

5.3.3 Network Reuse 46

6 CDVS in Robotic Visual Navigation 47

6.1 Experimental Setup 47

6.1.1 Software implementations 49

6.2 Preliminary Experiments 51

6.2.1 Effects of Feature Selection and Compression 51

6.2.2 Distinctiveness of CDVS local score 54

6.3 Loop-Closure Detection 56

6.3.1 Loop Definition 56

6.3.2 Loop Probability 57

6.4 Training of Proposed Model 58

6.4.1 Estimating Loop Probability 58

6.5 Experimental Results 60

6.5.1 Visual Odometry for Testing 60

6.5.2 Comparison with laser-scanner 64

6.6 Result Analysis 64

7 Fast Training of Convolutional Neural Networks using Scaled Kernels 67

7.1 Proposed Method 68

7.1.1 Spatially Scaling Convolutions 68

7.1.2 Pre-training Setup 70

7.1.3 Resizing and Continuing Training 71

7.2 Preliminary Experiments 74

7.3 Experiments on Pre-training 76

7.3.1 Resize-and-Continue Scheduled Training 76

7.3.2 Resize-and-Continue with Extra Training 78

7.3.3 Residual Networks 80
7.4 Result Analysis 83

8 Conclusion **84**
8.1 Future work 85

References **86**

List of Figures

2.1 Modern representation of SLAM problem. 4

2.2 Pinhole camera model. 6

3.1 SIFT detection: Keypoints are defined as extrema in the scale-space difference-of-Gaussian function both locally and in scale. 14

3.2 SIFT descriptor generation: A 16x16 pixel grid is center at the detected keypoint. For each 4x4 subregion the descriptor generates an 8-bin orientation histograms. 15

3.3 MPEG Compact Descriptor for Visual Search pipeline. 17

4.1 The first row corresponds to original classic approach to object detection. The second row represents classic approach to object classification. Bottom row represent the Deep Learning approach where each feature level is learned. 25

4.2 Usual representation of a feedforward neural network and its associated artificial neuron model. 26

4.3 Basic structure of a Convolutional Neural Network. A number of aternating convolutional and pooling layers is applied to the input for feature extraction followed by a sequence of one or more fully-connected layers. 27

4.4 Convolutional Layer: Each kernel performs channel-wise 2D convolutions to produce a single channel in the output set of feature-map. 28

4.5 Commonly used activation functions. 29

4.6 Inception module. 32

4.7 Residual block architecture. 33

5.1 2D convolution using matrix-vector multiplication. The concept can be extended for multi-channels inputs and kernels using concatenation. 39

6.1 Robot viewpoint and relative coordinate frame. 48

6.3 ROS Nodes for MPEG-CDVS Visual SLAM system. 50

6.4 Different types of floorings commonly found in indoor environments. Names were assigned according to the flooring's visible attributes. . 51

6.5 Average number of extracted local descriptors per image for each type of flooring. 52

6.6 Visual representation of *local score* for different flooring types. . . . 55

6.7 Visual representation of *local score* for the Printed Wood floor using different compression modes. 56

6.8 Visual representation of SIFT for different floor types 57

6.9 Cumulative loop probability for printed wood floor. 59

6.10 Path comparison using visual odometry. 61

6.11 Paths optimized using LAGO. 63

6.12 Map and path generated with a laser scanner and Gmapping algorithm. 66

7.1 Training starts with a *pre-train* network of smaller convolution kernels and input images. After a number of epochs, kernels are resized to the *target*'s resolution and training continues as scheduled. 68

7.2 Visual representation of the interface between convolutional and fully-connected layers. Feature-maps from a convolutional layer are first vectorized before entering a fully-connected layer, whose weights are usually represented in matrix form. The number of input must be selected according to the new feature-map spatial resolution $(\tilde{W}, \tilde{H})$ and the number of output neurons n_{out} is kept invariant. . . . 72

7.3 Rescaling weights in fully-connected layer back to *target*'s dimensions. Each column in the fully-connected weight matrix is reshaped to match the *pre-train* feature-map dimensions. Rescaling is applied in the same fashion as regular convolutional kernels and weights are then vectorized to the *target*'s new weight matrix. 73

7.4 Accuracy as function of epochs obtained using both original OverFeat-*fast* of input resolution 231×231 and its *pre-train* counterpart having 147×147 input resolution. 76

7.5 Accuracy as function time obtained using both original OverFeat-*fast* of input resolution 231×231 and its *pre-train* counterpart having 147×147 input resolution. 77

7.6 Effects of rescaling kernels at different epochs. Lower and upper horizontal lines define the maximum accuracies obtained with *pre-train* and *target* networks, respectively. 78

7.7 Accuracy as a function of epochs when training is allowed to continue using current learning rules for a few extra epochs. Learning rule is updated as soon as there is a drop in test accuracy. 80

7.8 Accuracy as a function of time when training is allowed to continue using current learning rules for a few extra epochs. Learning rule is updated as soon as there is a drop in test accuracy. 81

7.9 Accuracy curves obtained using ResNet-34 as a function of epochs. Lower and upper horizontal lines define the best accuracies obtained for the new baseline networks. 82

7.10 Accuracy curves obtained using ResNet-34 as a function of time. Lower and upper horizontal lines define the best accuracies obtained for the new baseline networks. 82

List of Tables

3.1 Maximum descriptor length in bytes for each mode of compression. 18

3.2 Number of selected transformed dimensions used by each mode of compression. 19

6.1 Average extraction times per image in milliseconds for each CDVS mode of compression and SIFT. 53

6.2 Average matching times per image in milliseconds for each CDVS mode of compression and SIFT. 54

6.3 Hypothesized values for local score loop detection. 59

6.4 Experiemtal threshold values for local score loop detection. 62

6.5 Relative pose errors between starting and final position for both visual odometry and VSLAM. 64

6.6 Storage requirement for all 7154 images and total matching time between last sequence image and all previous ones. 64

7.1 Suggested kernel resolution conversions with relative resize factors and bounds. 70

7.2 Architecture description of Pre-train network based on Overfeat-*fast*. Values in bold indicate differences with respect to original model. . 75

7.3 Effect of resizing kernels on storage requirements, accuracy and training time. 76

7.4 Final accuracy and training times for resized networks after a total of 55 epochs. Lower and upper bound accuracies are set by *pre-train* and *target* networks, respectively. 79

7.5 Best accuracy and total training times for resized networks with extra training. 80

7.6 Best accuracy and training times for ResNet-34. Training is reduced by 33.7 hours when upscaling two epochs before changing learning rate. 83

Chapter 1

Introduction

Visual features play a fundamental role in all computer vision tasks. They are intended to encode fundamental aspects in an image that are useful for solving specific problems such as face recognition, object classification, robotic localization, etc. Each of these tasks comes with a set of requirements such as response time constraints, accuracy and limited computational resources. Very often a compromise between those three must be attained, which is reflected in the choice of visual feature being used. It is also true, however, that some of these tasks share similar underlying requirements and visual features used for solving one problem could also be used for solving the other. Such similarity can be found between the tasks of large scale object recognition and robotic visual localization, and it is one of the subjects of this thesis.

In robotic navigation, an indoor robot that navigates throughout an environment using images from camera for orientation must compare what it is currently viewing with previously seen landmarks in order to estimate its motion and current position. Such landmarks, commonly referred to as visual features, must be distinctive and of fast comparison for reliable localization and motion estimation. In a seemingly different application, systems that perform image search over based on visual content, known as Content-Based Image Retrieval (CBIR) systems, must also compare a query image of an object against a database and return only the ones that effectively contain the object. Besides being accurate, CBIR response time must also be short not sacrifice the so called user-experience.

The similarity between requirements in these two problems suggests the use of similar solutions. Very recently, the Moving Picture Experts Group (MPEG) has defined a new industry standard for CBIR known as Compact Descriptors for Visual Search (MPEG CDVS) [1]. The standard specifies various modes of compression that offer trade-offs between descriptor distinctiveness and size and also suggests specific metrics to quantify similarity between images. The first part of this thesis is concerned with the use of this new visual descriptor on the context of robotic navigation.

On the one hand, if it is true that similar tasks may share similar solutions, on the other hand, problems that look similar at first might as well have very different requirements. This is the case for the tasks of object recognition and object classification. The former requires the identification of one specific object such as “Mole Antonelliana", while the latter must be able to encode the broader semantic definitions such as “landmark". The difficulty in solving the latter problem lies in fact class “landmark" represents a semantic definition, which encodes countless variations in visual aspects, while the “Mole Antonelliana" is, to some extent, unique.

For many years, however, these two problems were approached using the same visual features which has led to very limited performances in classification tasks. Fortunately enough, in the past few years, the field of computer vision has witnessed a shift of paradigm called Deep Learning, where visual features are no longer designed by computer vision experts but rather they are learned directly from data. This approach has defined new state-of-the-art in image classification and was made possible due to the recent availability of large training datasets and the use of GPUs for massive parallel computation. However, benefits of learning task-oriented features from large datasets come at the cost of long training periods of computationally demanding neural networks, that take weeks to produce desirable results. The second part of this thesis is dedicated to the development of a novel training technique designed to reduce training times of Convolutional Neural Networks (CNN), a family of networks especially developed to learn features for object classification.

Chapter 2

Simulataneous Localizantion and Mapping

In robotic navigation, the problem of generating a representation of the robot's environment while estimating its relative pose is called Simultaneous Localization and Mapping (SLAM). The difficulty in solving SLAM problems lies in its own formulation: a robot's pose must be referred to a map, while the details of a map are measured from relative poses. Moreover, in order to perform measurements and navigation, robots rely on physical sensors and actuators which are limited in precision. Given these inherited uncertainties, modern formulations of SLAM define this problem as a probability function of the robot's pose p_t and map's attribute m conditioned to a sequence of observations $o_{1:t}$, i.e. $P(p_t, m_t | o_{1:t})$.

The definition of pose is usually given by a set of coordinates and direction; however, the precise definition of a map's attributes and observations depend on the particular configuration of environment and sensors being used. When maps are associated to higher-level tasks such as path planning and collision avoidance, it must contain physical information about objects such as position and volume. However, the ultimate use of a map is to provide spatial reference to the robot, so that map attributes require precise positioning and unique identifiers. The observations, on the other hand, are a collection of signals obtained from sensors and signals applied to actuators. Signals applied to an actuator usually have a direct meaning in SLAM, such as "move forward 10 cm". On the other hand, information coming from sensors such as cameras and laser-scanner, must first be analyzed and associated to elements

in the environment, in a process called *data association*, so that it can be useful for estimating pose and the map's attributes.

This current formulation of the SLAM problem is summarized in Figure 2.1 adapted from [2], which divides the SLAM problem into two blocks. The *front-end* is responsible for acquiring data and using it for motion estimation (odometry) and place recognition (loop-detection), while the second block is responsible for generating an optimized representation of the map and robot's path given all observations. This work is mainly concerned with *front-end* block of SLAM that uses a single camera as sensor to produce odometry and loop-closure from visual features. We shall describe these components with detail and briefly describe the back-end. For an extensive and up-to-date overview of SLAM, the interested reader is referred to [2].

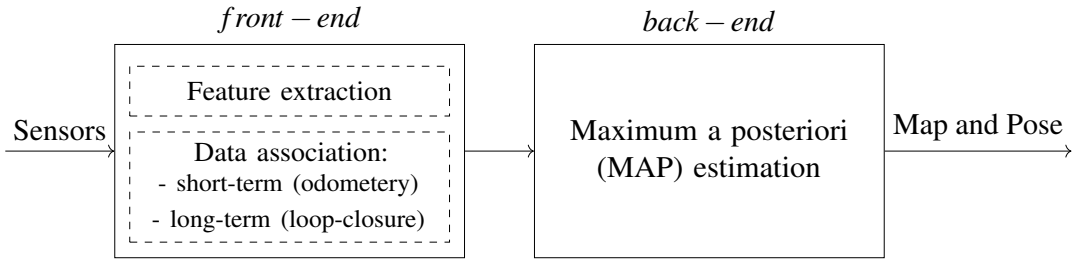


Fig. 2.1 Modern representation of SLAM problem.

2.1 Visual Odometry

Visual Odometry (VO) is the process of estimating a robot's egomotion, i.e. its displacement relative to a static scene, using information provided by one or more cameras attached to it. The term was first used by Srinivasan *et al.* in [3] and derives from wheel encoder odometry, a process commonly used by ground vehicles which counts the number of revolutions a wheel performs in order to estimate a cumulative displacement. Visual Odometry infers motion from sequences of images, which makes it insensitive to wheel drifting, although it does require the environment to have sufficient visual texture and illumination.

Monocular and Stereo VO

VO can be classified as either monocular or stereo depending on the number of cameras being used and on their particular setup. Stereo VO can directly retrieve the 3D positions of points in the environment through triangularisation, which is usually done using multiple cameras having known relative position [4–8], but can also be achieved using a single sliding camera that registers the scene from different viewpoints each time the robot stops [9]. Monocular VO, on the other hand, estimates motion using just one camera that takes just one picture at each position [10–12]. This approach can only estimate displacements up to a scaling factor. This apparent disadvantage can be overcome when using other sources of information such as measuring known objects in the image, or by including additional sensors like laser-scanner.

Visual features

Methods for estimating motion using monocular VO can be appearance-based, feature-based [13, 12, 14] or even a combination of these two. Appearance-based methods use pixel intensity from the entire image in order to estimate motion, while feature-based methods infer displacements from just a few interest points (also known as keypoints) that appear in consecutive images. The process of choosing which points should be used for motion estimation is called keypoint detection, while the process of finding similar keypoints across a sequence of images is called feature matching. The exact procedures with which features are detected and extracted depend on the feature detector and descriptor being used.

Keypoint detectors generally found in literature can be classified as either blob detectors or corner detectors. A corner is defined as an intersection of edges while a blob is small region in the image that differs from immediate neighbors in all directions in terms of illuminate intensity. Corners are usually faster to extract and better localized spatially; however, blobs are usually more distinctive and better localized in scale.

Important characteristics of a feature detectors in the field of robotics include: good spatial and scale localization, which improves motion estimation; repeatability, so that the same points are found in consecutive images; robustness to noise, compression artifacts and blur, so that low cost cameras can be used; and of fast

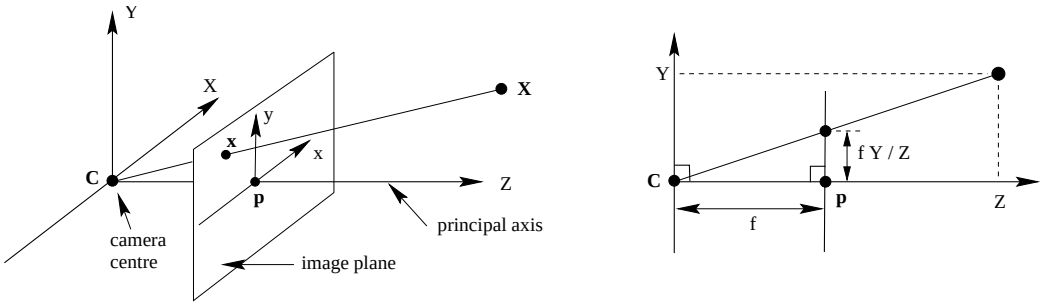


Fig. 2.2 Pinhole camera model.

detection so that will it not limit the speed of the robot. Desired properties for feature descriptors include: robustness to change in viewpoint and illumination so that should these conditions changes the overall feature vector will remain nearly the same; distinctiveness, so that interest point will not mismatch producing wrongfully associated data; small data footprint so that large maps can be generated using limited hardware; and fast extraction and matching times, again not to impose limits to the robot's mobility.

In this work, we are primarily concerned with feature-based Monocular Visual Odometry for indoors, planar environment. We shall we describe the process by which features that have already been matched between consecutive frames can be used for motion estimation. A broader overview on Visual Odometry is found in [15, 16].

2.1.1 Camera Model and Calibration

Estimating 3D motion from just a set of matching pixels requires a mapping between 3D world coordinates to 2D image pixel coordinates. The most commonly used method for doing this is by first using the pinhole camera model [17] followed by a change in coordinate systems.

Under the pinhole model, light emitted from 3D points pass through the *image plane* and meet at the camera center. The $z-$axis in this camera coordinate system is called the *principal axis* and intersects the image plane perpendicularly at the *principal point*. A representation of this model may be seen in Figure 2.2 extracted from [17].

Intrinsic Parameters

The pinhole model makes two strong assumptions that might not always be true. It assumes that the origin of the image plane is at the principal point and that pixels in the camera sensors are squares. The first issue can be corrected by considering an offset of (p_x, p_y), while the second issue can be adapted by considering different values of focal distances α_x and α_y for each axis. These parameters are known as the camera's intrinsic parameters and are used to transform 3D points from camera's coordinate points to 2D points in pixel coordinates as seen in (2.1).

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & 0 & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{cam} \\ y_{cam} \\ z_{cam} \end{bmatrix} \tag{2.1}$$

Extrinsic Parameters

Depending on the camera's position and on the robot's motion constraints, representing 3D points in the camera's coordinates might not be the most appropriate choice. In fact, in our work we extract visual features from the floor plane so that points in our setup will have world coordinate $z_w = 0$, which in turn allows us to reduce our problem to a planar homography as described in [18].

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K\,[R|T] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{2.2}$$

However, for general motion models, a rotation matrix $R \in SO(3)$ and a translation column vector $T \in \mathbb{R}^3$ are usually needed, which make up for the camera's extrinsic parameters. The complete transformation is represented in (2.2) with K being the camera's intrinsic parameters matrix.

Distortion Coefficients

Finally, due to manufacturing processes, lenses may display distortions that are not modeled by the pinhole camera but which can influence the correct 2D-3D mapping.

Radial distortion is generated by the curvature of the lens and it causes straight lines in the edges of the images to appear curved. Tangential distortion, on the other hand, occurs when the image plane is not aligned with the lens.

Fortunately, the problem of camera distortion has been throughly investigated [19–21] and many are the software available that are able correctly estimate these parameters from a sequence of patterned images such as chessboards. In this work we have used the computer vision camera calibration toolbox from MATLAB to retrieve the sets of intrinsic and extrinsic parameters along with distortion coefficients.

2.1.2 Motion Model

In our scenario, a robot carrying a fixed camera moves though an unknown environment acquiring a sequence of pictures at discrete times k. For each pair of consecutive images I_{k-1} and I_k we perform feature extraction and matching, which results into two sets of N matching coordinate pairs. Precise description of how matching is performed is postponed until the next chapter. We combine these pixel coordinates with the camera's intrinsic and extrinsic parameters and produce the sets P_{k-1} and P_k each containing the 3D coordinates for the N matching pairs.

By defining $\widehat{P}_{k-1}$ and $\widehat{P}_k$ to be the centroids of sets P_{k-1} and P_k respectively, we follow the approach for rotation and translation estimation from sets of points described in [22] and apply Singular Vector Decomposition (SVD) on the correlation matrix E.

$$E = \sum_{i=1}^{N} (P_k^i - \widehat{P}_k)(P_{k-1}^i - \widehat{P}_{k-1})^T \tag{2.3}$$

$$[U,S,V] = SVD(E) \tag{2.4}$$

Rotation matrix and translation vector between successive frames are then obtained as follows:

$$R_{k-1,k} = VU^T \tag{2.5}$$

$$T_{k-1,k} = -(R_{k-1,k}\widehat{P}_k) + \widehat{P}_{k-1} \tag{2.6}$$

The set of all $R_{k-1,k}$ and $T_{k-1,k}$ are then used as odometry constraints to the SLAM problem.

2.2 Loop-Closure

Loop-closure detection consists in identifying points in the path that have already been visited by the robot. Loop-closure improves visual odometry in two ways: first, pose estimation obtained by simply accumulating odometry inherently accumulates also measurement errors, which in turn makes the approach unreliable over long trajectories. In this sense, correctly identifying previously seen scenario improves the robot's belief regarding its pose and map attributes. Second, visual odometry by itself is unable to construct globally consistent maps. Since VO only compares features between consecutive frames, it is bound to represent loops as distinct places.

In Visual SLAM, the process of detecting loops is very similar to Visual Odometry with the exception that features being matched are not extracted from consecutive frames but rather they are compared to a larger pool of features from the robot's expected vicinity. In the worse case scenario, known as the kidnapped robot, the robot is moved from its position by an external agent and comparison between features must be done with the entire dataset.

Loop detection also produces relative poses represented by a rotation matrix and a translation vector; however, these loop-closure constraints are relative to past poses as in $R_{k,k-N}$ and $T_{k,k-N}$.

2.3 Maximum a Posteriori Optimization

Algorithmic approaches to solve the optimization part of the SLAM problem are usually divided into three classes: particle-filtering [23], Gaussian filter-based methods [24], and graphical approaches [25–28]. In this work we chose to use a graphical

approach to SLAM known as Linear Approximation for Pose Graph Optimization (LAGO) [28].

LAGO solves the SLAM optimization problem by allowing relative observations obtained from visual odometry and loop-closure to define a graphs of constraints. In this graph, poses are defined as nodes, while relative motion between poses are represented by the graph's edges. LAGO assumes that observations are independent and affected by zero-mean Gaussian noise for both rotation and translation.

For an overview of SLAM optimizers, the interested reader is referred to [2].

Chapter 3

Visual Feature Descriptors

In the previous chapter we have seen how visual features play an important role in Visual SLAM for both motion estimation and loop detection. As a matter of fact, visual features were first used to solve visual navigation [9] and the closely related problem of structure from motion (SfM) [29, 30], i.e. to reconstruct a 3D scene and camera trajectory from a sequence of images.

The problem of recognizing specific objects in a scene came later and became known in computer vision as object recognition. Early attempts to solve this problem would first model the query object using 3D primitives such lines, ellipses and vertex and then try to match those primitives to the candidate image [31, 32]. A second approach, which gained much attention during the 1990s, was to generate a global signature vector from the image based on its luminance such as color [33] and grey scale [34] histograms. The global approach, however, did not perform well when the query object was only partially visible in the dataset, which led to the development of feature based object detection [35]. In this approach, instead of using statistics over the entire image, specific points in an image were chosen and signature vectors were extracted for each one of them. The object recognition problem was then casted as whether or not two sets of features from different images contained enough intersection.

Characteristics inherent to the task of object recognition impose constraints to feature detectors and descriptors similar to those from Visual SLAM. Desired characteristics for feature detector includes: scale invariance for recognizing objects at different distances; repeatability so that pictures taken from different angles

provide the same keypoints; and robustness to noise and image compression so that pictures of the same object taken from different cameras is still able to match. On the other hand, required properties of visual descriptors include robustness to change in viewpoint and illumination and highly distinctive feature vectors. Differently from VSLAM, object recognition cannot rely on any temporal correlation between images, which make the last two requirements even more important. Moreover, although the primary concerns of object recognition systems are precision and recall, the need for both fast extraction and match increases as object recognition is applied to real world cases using large datasets.

In this chapter we describe two visual descriptors that were originally designed for solving the problem of object recognition and whose properties make of them good contenders for Visual SLAM. We first describe the Scale Invariant Feature Transform (SIFT), a hallmark in visual features and is still considered the reference among feature detector and descriptor. Next, we describe the MPEG Compact Descriptor for Visual Search, the recently finished standard for feature detection, description and compression designed for Content Based Image Retrieval. Finally, we give a brief overview of other visual features found in literature which have also been used for Visual SLAM.

3.1 Scale Invariant Feature Transform (SIFT)

The Scale Invariant Feature Transform (SIFT) is an object recognition algorithm developed by David Lowe [36, 37] that describes both a feature detector and a feature descriptor. SIFT has been used in a variety of computer vision applications including panoramic image stitching[38], hand posture recognition [39], and Visual SLAM [40]. It was designed to be invariant to rotation, translation and robust against change in illumination and viewpoint.

3.1.1 SIFT Keypoint Detection

The SIFT feature detector defines a keypoint as a local extremum in a difference-of-Gausssian function (DoG). Theoretical foundation for this approach relies on the scale-space theory [41–43], which represents an image $I(x,y)$ at a scale σ by its convolution with a 2D gaussian kernel of variance σ^2 as in (3.1).

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{3.1}$$

$$D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma) \tag{3.2}$$

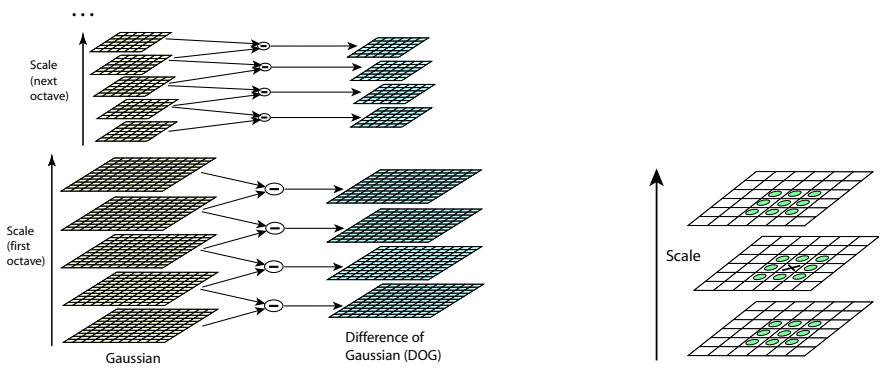
The difference-of-Gaussian function is obtained by successively subtracting adjacent scale-space representations of the same image that differ by a constant factor $k > 1$ in scale as seen in (3.2). In this context, an octave is said to be completed every time the current scale is doubled with respect to the original scale. At the end of each octave the image is downsized and the process is carried on forming a pyramid representation as seen in Figure 3.1a extracted from [37]. Once the pyramid of DoG is available, the algorithm identifies the local maxima and minima in both scale and in space, performing a total of 26 comparison per point as seen in Figure 3.1b, obtained from the original article. In its original formulation, the number of scales used for searching these extrema was set to three, so that a total of five DoG representations were necessary for finding extrema over these central scales, which in turn requires six scale-space representation of images for each octave.

The precise locations of these extrema are further refined by fitting a 3D quadratic surface to the local points, which was shown to improve matching and keypoint stability [44]. Finally, unstable keypoints having low contrast or belonging to edges are eliminated.

It is worth mentioning that the difference-of-Gaussian gives an approximation to the scale-normalized Laplacian of Gaussian $\sigma^2\nabla^2 G$ function whose minima and maxima have been verified experimentally to give more stable features with respect to other detectors such as Hessian and Harris [45].

3.1.2 SIFT Feature Extraction

SIFT descriptor achieves rotation invariance by assigning a dominant orientation for each keypoint. This is done by first extracting magnitude and angle of the gradients in $L(x,y,\sigma)$ closest to the extremum found, as seen in (3.3) and (3.4) respectively. A 36-bin orientation histogram is then generated by quantizing the angles with steps of 10° and weighing each sample by the gradient's magnitude and a gaussian-weighted mask centered around the extremum. The bin containing the maximum weighted



(a) Difference of Gaussians. (b) Extrema detection.

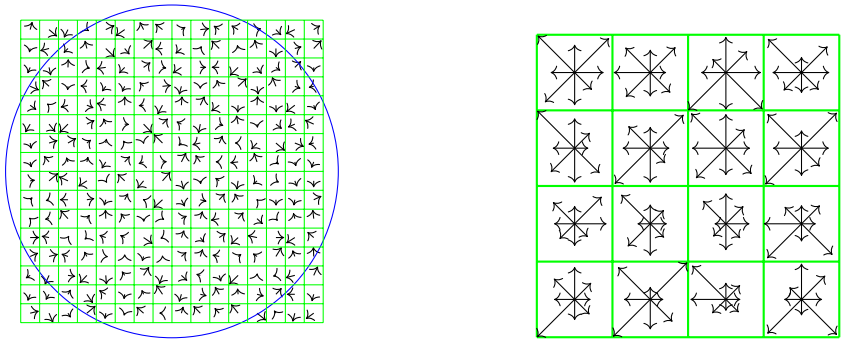
Fig. 3.1 SIFT detection: Keypoints are defined as extrema in the scale-space difference-of-Gaussian function both locally and in scale.

sum of magnitudes defines the keypoint's orientation which, along with images coordinates x, y and scale σ, completely defines a keypoint. However, if the second highest weight sum of gradient magnitude is over 80% of the maximum value, a second keypoint is defined, so that a single extremum can generate more than one keypoint.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{3.3}$$

$$\theta(x,y) = \arctan \frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \tag{3.4}$$

For each one of these detected keypoints, the SIFT descriptor will then generate a 128-dimension vector to be used during matching. This is done by considering image's gradients at a 16x16 pixel neighborhood centered around the keypoint's location and scale and rotated relative to the keypoint's orientation. Much like when defining the keypoint orientation, the gradient's magnitudes are weighted using a circular gaussian-weighted mask centered at the keypoint value as depicted in Figure 3.2a. However, instead of defining a global histogram of orientations, the descriptor divides the 16x16 neighborhood into 4x4 subregions of 4x4 pixels. For each subregion an 8-bin orientation histogram is generated as seen in Figure 3.2b. These 16 histograms are then concatenated into a 128-dimension vector, and finally the descriptor vector is unit normalized after all dimensions have been clamped at



(a) Weighted Gradients (b) Histogram Gradients

Fig. 3.2 SIFT descriptor generation: A 16x16 pixel grid is center at the detected keypoint. For each 4x4 subregion the descriptor generates an 8-bin orientation histograms.

0.2. This is an upper bound empirically found to reduce the effects of non-linear illumination.

3.1.3 SIFT Feature Matching

In the context of object recognition, an object present in a query image q is said to be found in a candidate image d if a certain number of features in q correctly match those in d. Matching of SIFT features is done by evaluating the pairwise ℓ_2 distances between all feature vectors in the query image and all feature vectors in the candidate image. Each keypoint in q is then initially associated to the closest keypoint in d.

The existence of similar feature vectors in either set of features may lead to erroneous feature association. In order reduce the number of incorrect matches generated from similar features vectors in d the distance to the second closest keypoint is also taken under consideration. If the ratio between the distance to the closest and the distance to the second closet keypoint in d is larger than 0.8 these features are no longer considered a match.

As in the case of Visual Odometry, the number of incorrect matches can be further reduced by selecting subsets of matching features that agree on a particular hypothesis of the visual transformation. In its original formulation SIFT uses the generalized Hough Transform [46] to find cluster of matched features that agree on change in scale, pixel coordinates and feature orientation. Other common approaches to robust parameter estimation for object recognition include the already mentioned

Random Sample Consensus (RANSAC) [47] and the Least Median of Squares (LMedS)[48].

3.2 MPEG Compact Descriptor for Visual Search

The ubiquity of digital cameras in devices connected to the Internet such as cellphones, laptops and tablets has made these accessories the ideal platforms for developing augmented reality and visual search applications, such as Google Goggles, Bing Vision, and Amazon Flow.

The existence of such applications, however, imposes new challenges to Content-Based Image Retrieval (CBIR) system, which must search over large datasets and respond with correct results within just a few seconds in order not to compromise the so called user experience. Moreover, visual search applications must also be bandwidth efficient since typical use cases include a user sending a query image or visual descriptors over limited Internet connection. In this scenario, it has been shown that sending locally extracted visual features instead of sending the entire image can significantly reduce the amount of data exchanged with the remote server [49, 50].

In response to these needs the Moving Picture Experts Group has recently completed a new standard for image retrieval known as MPEG Compact Descriptor for Visual Search (CDVS) [1] designed specifically to allow for efficient and interoperable visual search applications. An overview of the process by which features are detected, extracted and compressed according to the standard is represented in Figure 3.3. In this section we will briefly describe the key aspects of each block and highlight the optimization techniques developed for efficient image retrieval. A recent and complete overview of the standard and its history can be found in [51].

3.2.1 Image Preprocessing

In general, high resolution images are not required for correct object recognition and, most of the times, the use of large images just increases processing times. For these reasons, CDVS requires that input images have both horizontal and vertical dimension of at most 640 pixels. If one of the image's dimensions is greater than

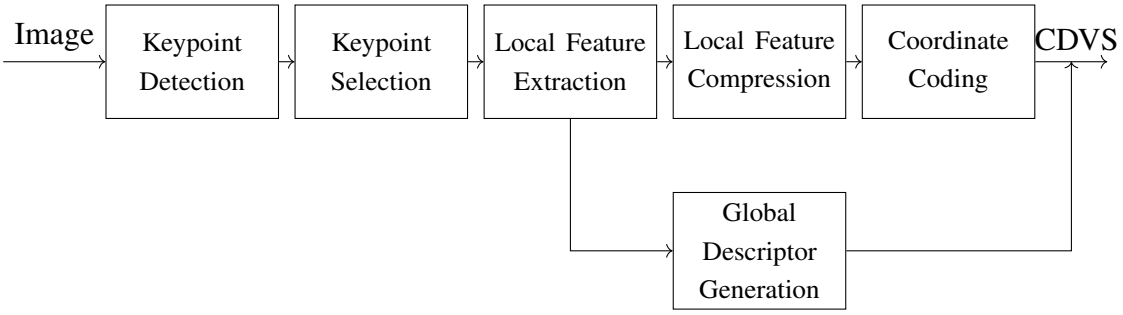


Fig. 3.3 MPEG Compact Descriptor for Visual Search pipeline.

this value, then the image must be resized, keeping the original aspect ratio, so that the largest of the two dimensions be equal to 640 pixels.

3.2.2 Keypoint Detection

Similarly to the SIFT detector, CDVS interest points are found using the scale-space representation of images. However, differently from SIFT, CDVS searches for points of maxima and minima using a Laplacian-of-Gaussian approach instead of difference-of-Gaussians. Moreover, CDVS approximates the Laplacian-of-Gaussian (LoG) function by using a low-degree polynomial known as ALP (A Low-degree Polynomial) as seen in (3.5) and (3.6)

$$ALP(x,y,\sigma) = \sigma^3 \sum_{k=0}^{3} a_k F_k + \sigma^2 \sum_{k=0}^{3} b_k F_k + \sigma \sum_{k=0}^{3} c_k F_k + \sum_{k=0}^{3} d_k F_k \tag{3.5}$$

$$F_k = \sigma_k^2 L(x,y,\sigma_k) * f \tag{3.6}$$

where f is the discrete Laplacian operator matrix and a_k, b_k, c_k and d_k are coefficients defined by the standard. CDVS also defines procedures to refine the keypoint's coordinate position to sub-pixel precision; to remove duplicated keypoints extracted at different octaves; and to assign an orientation to each keypoint.

When compared to SIFT feature detector, ALP detects extrema using a smaller number of pixel-neighborhood comparison. ALP first finds local extrema over the scale-space through the polynomial's first derivative and then it compares each extremum with only 8 spatial neighbors. Also, as seen in (3.5), ALP requires only 4 image filtering operations per octave. Besides being a faster approximation of a

Table 3.1 Maximum descriptor length in bytes for each mode of compression.

Compression	mode 1	mode 2	mode 3	mode 4	mode 5	mode 6
Descriptor Length	512 B	1024 B	2048 B	4096 B	8192 B	16386 B

LoG, the ALP detector has also been shown to retrieve more repeatable key points than the SIFT keypoint detector [52].

3.2.3 Keypoint Selection

Differently from the SIFT descriptor, CDVS does not generate an unbounded set of feature vectors for each detected keypoint. Instead, the CDVS feature extraction generates a bitstream that includes a subset of local features whose total length in bytes is upper-bounded according to one of the modes of compression listed in Table 3.1.

This upperbound implicitly limits the number of descriptors generated by an image and thus poses the question of which subset of keypoints should be used for extracting feature descriptors. Statistical studies on the probability of correctly matching pairs of local features [53] have helped answer this question by defining a measure of relevance for each keypoint. This measure is a function of the following keypoint characteristics:

- Scale where the keypoint was found.
- Keypoint response to the ALP feature detector.
- Keypoint spatial distance to the center of the image.
- Ratio of the squared trace of the Hessian to the determinant of the Hessian, obtained during subpixel refinement.
- Second derivative of the scale-space function with respect to σ.

An indirect benefit of feature selection is that, by limiting the number of local features available in a CDVS bitstream, it reduces the time required for both extraction and matching of visual descriptors.

Table 3.2 Number of selected transformed dimensions used by each mode of compression.

Compression	mode 1	mode 2	mode 3	mode 4	mode 5	mode 6
Number of dimensions	20	20	40	64	80	128

3.2.4 Local Feature Extraction

CDVS uses the SIFT descriptor as a starting point for its local descriptors. For each selected keypoint on the previous step, a 128 dimension vector is generated by computing the histograms of gradients relative to the keypoint's orientation. The standard follows typical SIFT implementations [54] where each component of this vector is quantized to integer values between 0 and 255.

3.2.5 Local Feature Compression

Not all of SIFT dimensions will contribute to the final CDVS local feature. The standard first alternately applies two sets of linear transformations to each histograms of SIFT subregions (Figure 3.2b). According to the compression mode being used, the compression algorithm selects a specific subset of the transformed components to compose the local feature. The number of selected transformed components is reported in Table 3.2.

In order to allow for interoperability between modes, the order with which these components are selected was chosen so that the set of components of a more compressed mode is always a subset of the set of components of a less compressed mode.

Once the transformed components have been selected, each component is quantized to three values namely -1, 0, and 1 and finally encoded into 10,0, and 11. The quantization levels for each component are defined in the standard's normative lookup tables.

Coordinate Coding

The next step in the feature compression pipeline is to efficiently encode the coordinates of each keypoint. Coordinates are usually represented using floating-point

precision, which becomes the bottleneck once the feature vectors have been quantized. CDVS uses a location histogram coding scheme [55] to identify clusters of features and efficiently make use of arithmetic coding.

3.2.6 Global Descriptor Generation

CDVS also defines a *global* descriptors which gives a general representation of the entire image based on the statistics of local features. This is obtained by first selecting up to 250 local features to whom reduce dimensionality using Principal Component Analysis (PCA). It then generates a Fisher Vector [56] representation using a 512 component Gaussian Mixture Model. CDVS further quantizes the global descriptor as to allow for fast Hamming-distance comparison [57].

3.2.7 CDVS Feature Matching

Local Features

Comparison between local features is performed with ℓ_1-norm using XOR and lookup tables, which is much faster than ℓ_2-norm used by SIFT.

CDVS also considers the distance ratio r between the closest match and the second closest one, and defines a matching *score* for each matching pair based on this distance ratio as seen in (3.7).

$$\beta = \cos\frac{\pi r}{2} \tag{3.7}$$

Should more than one point from one image be associated to the same point in the other image, then whichever match scored lowest according to (3.8) is removed from the set of matching pairs. Based on this definition, the standard also suggests a metric of image similarity known as *local score* defined as the sum of the scores of all matching pairs.

$$\text{local score} = \sum_{i=1}^{N} \beta_i \tag{3.8}$$

Geometric Consistency Check

The standard also describes a non-normative geometric consistency check algorithm called DISTRAT. The algorithm defines a goodness-of-fit test whose null hypothesis is based on the spatial distribution of incorrectly matched features. The algorithm has been shown to be many time faster than other robust parameter estimators such as RANSAC [58].

Global Descriptor

The similarity score between two global descriptors is referred to as *global score* and it is a weighted correlation between the these descriptors. The *global score* can be calculated efficiently using XOR and lookup tables. Since the global descriptor ignores the spatial positions of features in an image, its value for metric robotic navigation is very limited, hence it was not used in this work.

3.3 Other Visual Features

A plethora of visual detectors and descriptors has been suggested in the literature in the past 20 years. Here we give a brief overview of the most relevant ones to the field of robotic navigation. For a more complete survey of the subject we refer the reader to the works in [59, 15, 60].

- **Harris:** A popular corner detector that locally analyses the autocorrelation function of the image[61]. It defines the structure tensor in (3.9) where $w(u,v)$ is a circular gaussian window.

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{3.9}$$

  A point in the image is defined to be a corner if the two eigenvalues of A are large. In order to avoid having to calculate these values, Harris suggests using (3.10) with the tunable factor k.

$$M = \det(A) - k\text{trace}^2(A) \tag{3.10}$$

- **Shi-Tomasi:** A corner detector similar to Harris, but whose metric for detection is the lowest among the eigenvalues of A [62]. When compared to Harris, this metric is found to be more robust to affine transformation.

- **FAST:** The Features from Accelerated Segment Test [63] relies finds corners by searching for arcs around a pixel. This test allows for and average of just 3.8 pixel comparisons for each candidate.

- **GLOH:** The Gradient Location and Orientation Histogram [60] is an extension to SIFT, which uses a log-polar grind and PCA to reduce the dimensionality of the final descriptor.

- **SURF:** The Speeded Up Robust Features [64] is both a feature detector and descriptor developed to be a faster alternative to SIFT. It approximates the determinant of Hessian blob detector using Haar wavelet which can be efficiently implemented using the integral image technique described in [65]. The feature descriptor is generated using Haar wavelet responses around the detected points, which can also be computed with integral images.

- **CenSurE:** The Center Surround Extremas [66] is a fast blob detector that computes the extrema of center-surround filters over multiple scales using the image's original resolution for each scale. Center-surround filters are very coarse approximations of the Laplacian of Gaussian operator. The name derives from the high contrast between central and peripheral regions of the filter.

- **BRIEF:** The Binary Robust Independent Elementary Features [67] is a binary string feature descriptor whose individual bits are obtained by comparing the brightness of pairs of points around the keypoint position. Feature matching is fast as it is performed using Hamming distance.

- **BRISK:** The Binary Robust Invariant Scalable Keypoints[68] defines scale-space FAST-based detector which allows for rotation invariance in combination with a binary descriptor.

Chapter 4

Deep Learning for Object Classification

In previous chapters we have described how visual features originally developed for object recognition and later optimized for large scale image retrieval can be used in the context of metric SLAM. This interoperability was possible mainly because these two tasks shared the same underlying assumption where places and objects are unique, rigid entities.

Unfortunately, this same assumption does not hold for the more general task of image classification where an image must be labeled according to a finite set of classes such as *dog*, *human*, *airplane*, etc. The difficulty in solving this task lies in the fact that labels encode semantics not immediately related to isolated image patches. In fact, classification can be interpreted as a complex function which must analyze the image as a whole and discard irrelevant information. For example, an image of a dog should be classified as a dog regardless of the dog's race, pose or color; and a chair should be correctly classified regardless of its precise shape, and material. Early attempts to solve the problems of non-rigid deformations and intra-class variability include the use feature aggregation methods such as bag-of-visual-words [69, 70], and Fisher Vector [71, 72].

The bag-of-visual-words represents each image as a histogram of its quantized visual features. Usually Principal Component Analysis (PCA) is applied on a large amount of feature vectors obtained from a large dataset of images. K-means is then used over the dimensionality reduced features in order to generated a fixed dictionary.

Finally each feature in an image is represented using this dictionary and the entire image is represented as a histogram of occurrences. The Fisher Vector approach, on the other hand, first obtains a Gaussian Mixture Model (GMM) from a large number of feature vectors that will represent a global generative model of the features. Each image is then represented by the gradient of the log-likelihood of its set of features on the GMM, which in turn measure how individual parameters of the GMM should change to better accommodate the image's feature distribution.

At this point we notice that both strategies rely on the extraction of visual features that were developed separately from the aggregation mechanism. In fact, the strategy of building intermediate representations on top of predefined features was so commonly used that famous object classification challenges like the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2010 would provide *"vector quantized SIFT features suitable for a bag of words"* in order to *"facilitate easy participation"*[1].

In this chapter we describe a family of Artificial Neural Networks, known as Convolutional Neural Networks (CNN), which have become the standard approach in object classification in past few years due to their great performance in solving such task. These networks reflect a new paradigm to machine learning known as Deep Learning, whose objective is to *learn* hierarchical set of features directly from data and it is inspired by the early discoveries about the visual cortex [73]. We invite the interested reader to refer to the first chapters of [74] for a more thorough introduction to the Deep Learning, while a collection of important historical events in Deep Learning is reported in [75].

4.1 Artificial Neural Networks

An Artificial Neural Network (ANN) is a computational structure composed of interconnected elementary units called Artificial Neurons originally introduced by McCulloch and Pitts [76]. Their model of a neuron consisted of a set of identical weights, a fixed threshold, binary inputs and output, and an inhibitory signal. Under this model, a neuron would output 1 only if the inhibitor signal were inactive and the

[1] "Features." *Large Scale Visual Recognition Challenge 2010 (ILSVRC2010).* Stanford Vision Lab. Web. 14 Jan. 2017

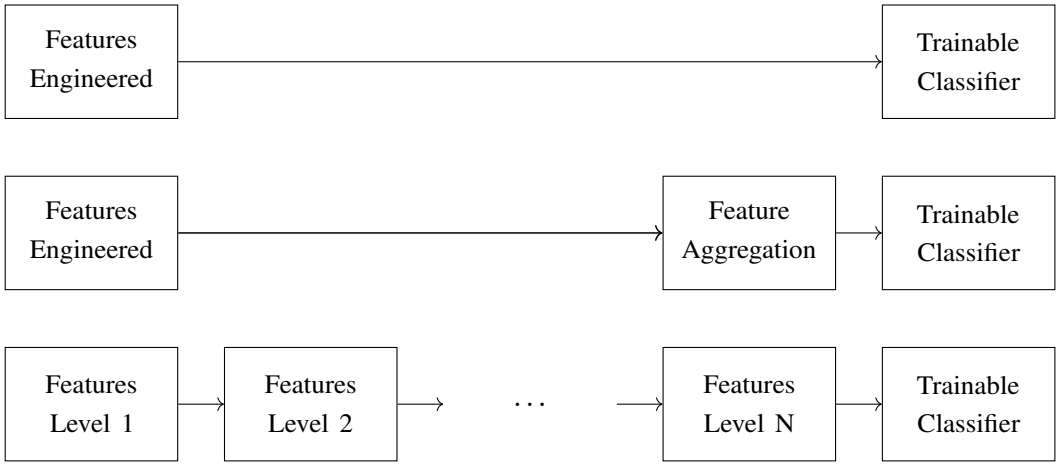


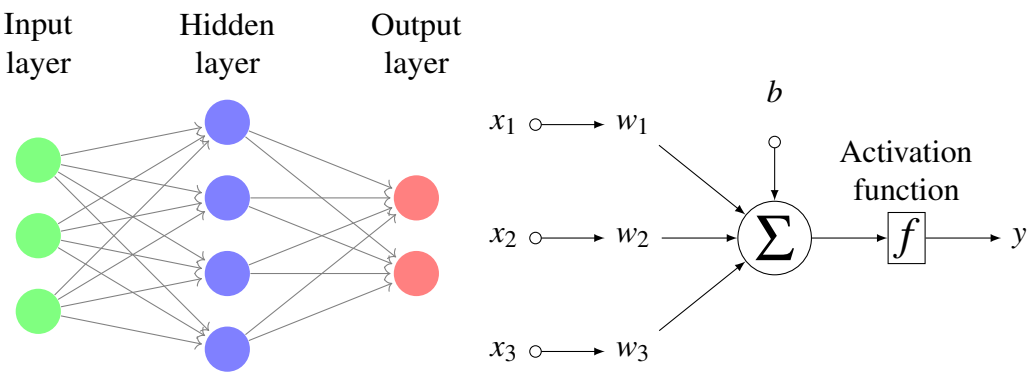
Fig. 4.1 The first row corresponds to original classic approach to object detection. The second row represents classic approach to object classification. Bottom row represent the Deep Learning approach where each feature level is learned.

weighted sum of inputs were greater than the fixed threshold. If either one of these conditions were not met, the neuron would output zero.

Rosenblatt's perceptron [77] later improved this model by removing the inhibitor signal; allowing weights and bias to have different real values for each input; and finally, by providing an algorithm for learning those parameters. Today's model of artificial neuron carries most of the perceptron's characteristics except for its activation function, which no longer needs to be a binary threshold. A modern artificial neuron is depicted in Figure 4.2b, while examples of commonly used activation functions nowadays are seen in Figure 4.5.

The directed graph formed by the connections of neurons defines the network's architecture. According to the presence or absence of cycles in such graph, an ANN architecture can be classified as either *recurrent* or *feedforward* neural network. Cycles in a network create a dependency of the current output on the values of previous inputs, which makes recurrent neural networks most useful when applied to long sequences that have some degree of temporal correlation, such as audio and text. Feedforward neural networks, on the other hand, are more suited for applications where sequences of inputs can be considered to be independent.

As we shall see, Convolutional Neural Networks for object classification are essentially feedforward ANNs, as seen in Figure 4.2a, but whose connections have been constrained to mimic the ones from the optic nerves to the visual cortex.



(a) Feedforward Neural Network. (b) Single neuron.

Fig. 4.2 Usual representation of a feedforward neural network and its associated artificial neuron model.

4.2 Convolutional Neural Networks

Convolution Neural Networks as known today has its roots in the handwritten character recognition system developed by Fukushima [78] known as neocognitron. Inspired by the structure of the mammalian visual cortex [73], the neocognitron was a hierarchical, multi-layered artificial neural network composed of alternating layers of simple cells (S-cells) and complex cells (C-cells). Each neuron in an S-cell layer is responsible for detecting a particular pattern in the previous layer and produce a "cell-plane" map containing the 2D position where the pattern was found. Layers containing C-cells were designed to provide a certain degree of shift invariance and activated if features in its vicinity were active.

Based on these ideas, Yann Lecun developed the LeNet-5 network for handwritten digits classification [79] consisting of two alternating sequences of convolutional layers and pooling layers, followed by three fully-connected layers. The significance of LeNet-5 for the development of CNNs is twofold: First, it serves as the basic structure for most CNNs architectures, i.e. sequences of alternating layers of convolution and pooling followed by a few fully-connected layers, as seen in Figure 4.3, where the depth of a network is defined as the number of these non-linear layers; and second, it showed that CNNs could be trained using the efficient gradient-based learning method known as the backpropagation algorithm. In this section we will focus on describing in depth each of the layers mentioned above, while the description of the backpropagation algorithm is postponed until the next chapter.

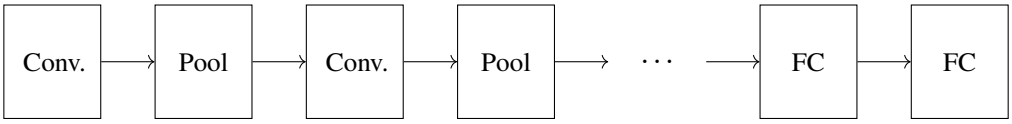


Fig. 4.3 Basic structure of a Convolutional Neural Network. A number of aternating convolutional and pooling layers is applied to the input for feature extraction followed by a sequence of one or more fully-connected layers.

4.2.1 Convolutional layers

Convolutional layers are the essential building-blocks in a CNN that are responsible for extracting features based on the principles of *shared weights* and *locality*.

In a convolutional layer, inputs and outputs are arranged as sequences of 2D maps. Each neuron associated to an output is connected only to a small spatial neighborhood of all input maps. These local connections account for *locality* in CNNs and forces features to represent spatial information.

Oftentimes features that meaningful in one region of the image are also meaningful over all parts of the image. This leads to the idea of shared weight, where neurons associated to the same output 2D map share the same set of weights. In this way, each output map is associated to a feature (set of weights) maps the spatial position where the feature was found, leading to the term *feature-map*.

The number of maps in each sequence is referred to as *channels*, much like the channels in color image. In fact, if the convolutional layer in question is the one immediately connected to input images, the number of input channels of the layer will be three for RGB images and one for gray scale images.

A closer look at the implementation of shared-weights and locality reveals that the output of a convolutional layer can be obtained by convolving the input with the set of shared weights, hence the name convolutional layer. Under this interpretation, each set of local weights is referred to as a *kernel* and each kernel must have the same number of channels as the input signal. Finally we observe that, convolutions are performed only on the *valid* spatial region of input maps, and that they may use strides different from one for computational purposes at the cost of loosing spatial resolution. This means that, considering square input maps of side I, a kernel with spatial resolution K, and a stride of S, the final side of the output maps $O = (I-K)/S+1$. In order to avoid continuous loss of spatial resolution

it is common practice to spatially pad the boarders of input maps. A common representation of Convolutional layers is seen in Figure 4.4.

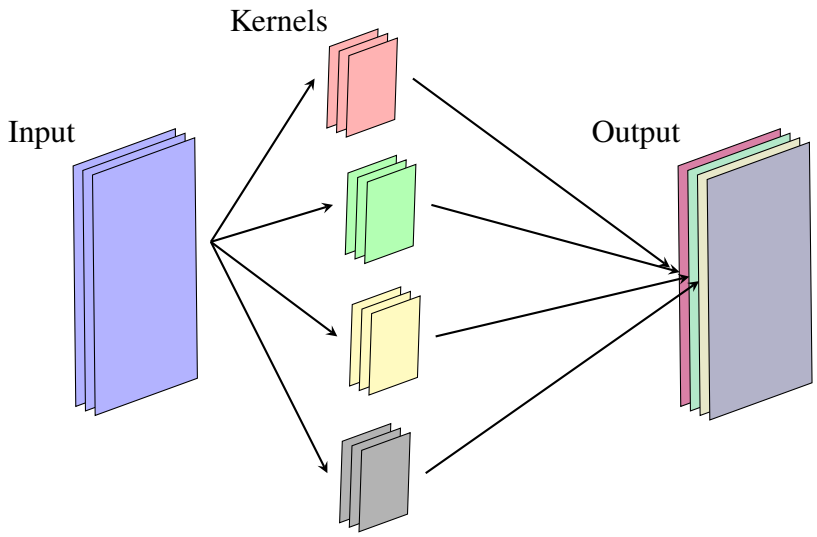
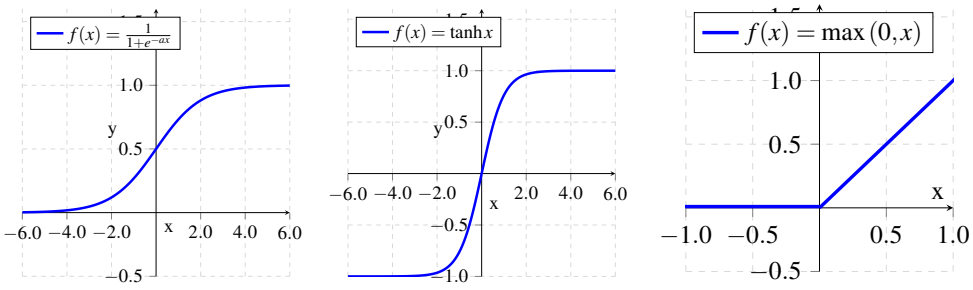


Fig. 4.4 Convolutional Layer: Each kernel performs channel-wise 2D convolutions to produce a single channel in the output set of feature-map.

4.2.2 Activation Functions

Although the modern definition of Artificial Neuron encompasses the use of activation functions, common implementations of CNNs usually define them as separate layers. In Deep Learning, the activation function applied to the pre-activation stage of each neuron (weighted sum of inputs) need be non-linear. This allows for the network to represent complex function such as the “XOR" problem. The choice of non-linear activation function influences the network’s performance and are still subject of research today [80]. Following are the most commonly used activations in literature:

- *Logistic function:* The logistic function belongs to a family of “S" shaped functions known as sigmoid functions and it was the most commonly used activation function until recent years. It outputs values from 0 to 1 and it is a strictly increasing, differentiable function which shows near-linear behavior around zero and saturation over the extremes. As we shall see in the next chapter, linearity is important during training.



(a) Logistic function. (b) Hyperbolic tangent. (c) Rectified Linear Unit.

Fig. 4.5 Commonly used activation functions.

- *Hyperbolic Tangent:* For CNNs having many layers, the fact that the logistic function is not a zero-mean function will eventually cause later layers to work on the saturation region of the function [81]. This can be avoided by using the hyperbolic tangent which has zero mean and also belongs to the sigmoid function family and thus has the same desirable features.
- *Rectified Linear Unit (ReLU):* This is today's most used and recommended activation function. It is a piecewise linear function that outputs 0 for negative values of pre-activation and the pre-activation value itself otherwise [82],[83, 84]. The rectified linear unit is not differentiable at the origin, but in practice this is not a problem.

4.2.3 Pooling Layer

Hubel and Wiesel's notion of "complex cells" where the outputs of spatially close features are combined in order to achieve shift invariance and robustness to noisy input has been incorporated in what today is known as pooling layers. Common pooling layers usually act only on the spatial dimensions of feature maps so that the number of input and output channels in a pooling layer is the same. The importance of pooling operation in computer vision has been thoroughly investigated [85] and current state of the art networks use implement one of the following pooling schemes:

- *Average pooling:* Outputs the average over a small kernel window throughout the entire image. It was very common in the past for its simplicity.

- *Max-pooling:* Outputs the maximum value of a small kernel window throughout the entire image. It is by far today's most used pooling layer.
- *Fractional Max-Pooling:* Much like convolutional layers, pooling-layers can be used with integer strides to produce subsampled feature maps. In order to reduce the loss in resolution cased by integer strides, the work in [86] proposes the use of variable-sized pooling windows, whose dimensions are randomly selected to fit the spatial ratio between input and desired output spatial resolution.

4.2.4 Fully-connected Layers

While neurons in a convolutional layer are just connected to a locally restricted set of neurons in the input activation maps, each neuron in a fully-connected layer receives signals from the entire set of input maps. In practice, this means that fully-connected analyzes the input globally at the cost of loosing spatial information and it is commonly referred to as the "classifier" part of the network.

As we will see in chapter 7, some spatial correlation can still be recovered from a fully-connected layer connected to a convolutional layer. In fact, we exploit this fact to propose a fast training method for CNNs.

4.3 Modern Architectures

In this section we present the recent evolution of CNNs architectures and describe their innovative contributions to the task of image classification.

AlexNet

In 2012, Alex Krizhevsky won the ImageNet Large Scale Vision Challenge (ILSVRC) [87] using a CNN having five convolutional and three fully-connected layers [88]. The network proved the effectiveness of CNNs on classification problems by achieving 15.3% of top-5 classification error, while the second best entry obtained 26.2% while the winner of the previous edition achieved 25% top-5 error.

Overfeat

In 2013, the Overfeat network improved AlexNet's architecture by lowering strides in the first layers yielding to 14.2% top-5 classification error [89]. It also showed that training a convolutional network to simultaneously classify, localize and detect objects in images can improve the accuracy in all these tasks. Given its relative small number of layers, this network is used in chapter 5 during our preliminary studies on improving CNN training speed.

VGG Network

In a complete study authors in [90] showed the importance of depth in classification accuracy by training and evaluating CNNs having from 11 to 19 layers. Their work showed that the use of two consecutive convolutional layers with small 3x3 spatial kernels gave better accuracies than using a single 5x5 convolutional layer and associated this gain to the extra use of non-linearity between layers. Moreover, authors empirically verified that a 19-layer CNN produced similar accuracy as a 16-layer one, exposing the difficulty in training deep CNNs with the techniques available so far. Finally, VGG Net further reduced the top-5 classification error in the ILSVRC-2014 Classification task to 7.3%.

GoogLeNet

Also in 2014, researches at Google presented the GoogLeNet [91], a 22-layer convolutional network based on a composed module known as Inception. The name of the block derives from the fact that the module itself can be considered a network since it is concatenation of parallel and serial convolutional layers, as seen in Figure 4.6. The network achieved top-5 classification error of 6.67%.

Newer incarnations of Inception module have been proposed which decomposed layers having 5x5 kernels into two layers of 3x3, and included the use of Batch Normalization, a technique for improving training of deep CNNs that will be described in the next chapter. The forth and latest version of the Inception module also included the skip connections proposed in the work below, and therefore it is called Inception-ResNet. This network currently achieves 3.08% top-5 classification error in ILSVRC.

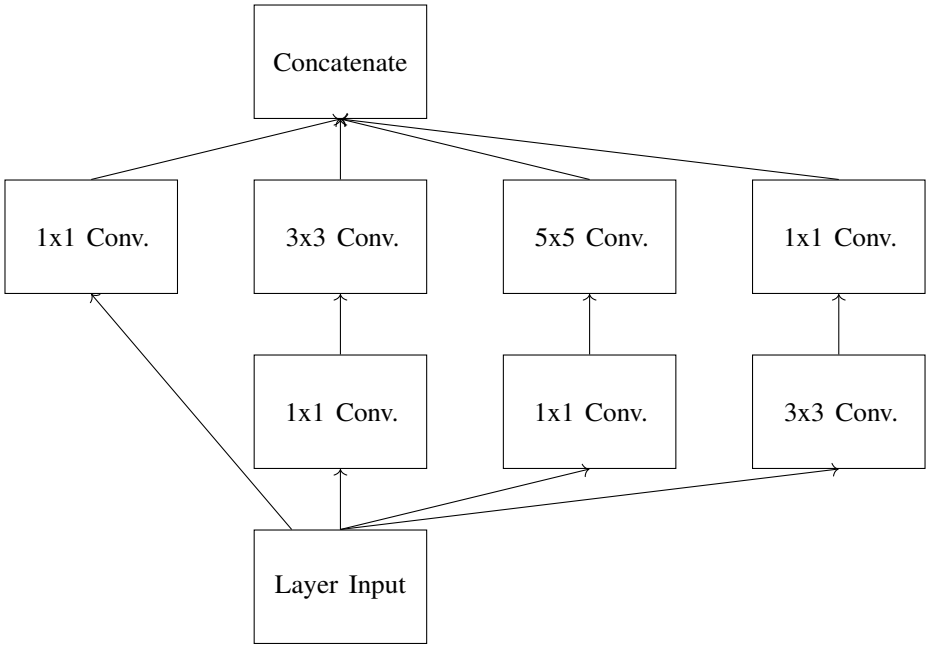


Fig. 4.6 Inception module.

Residual Network

In 2015, authors in [92] proposed a new architecture for CNN called Residual Network (ResNet) based on the concept of skip connections, an idea also developed in independently by [93]. The goal behind a Residual block in ResNet is to learn a difference of representation, as seen in Figure 4.7. This approach allowed authors to achieve 3.57% top-5 error on the ImageNet test set using 110-layer models. This model also serves as validation for our proposed training method.

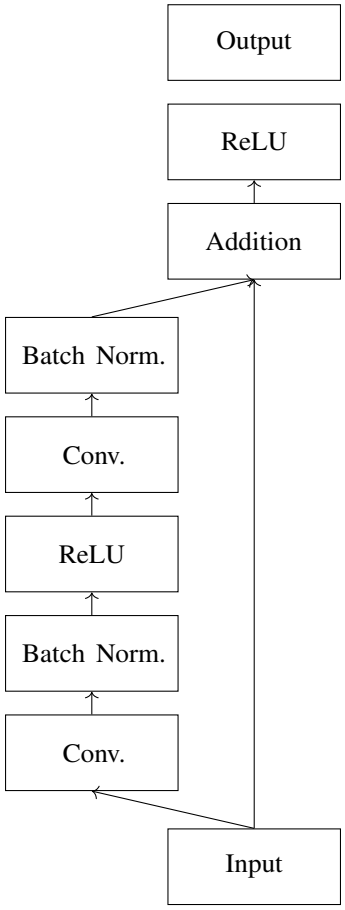


Fig. 4.7 Residual block architecture.

Chapter 5

Training Convolutional Neural Networks for Object Classification

In the previous chapter we have seen how the use of CNNs has helped set new records on international computer vision competitions such as ILSVRC. Learning ad hoc features for specific tasks, however, comes at the cost of long training periods as it has been consistently reported by participants. The AlexNet architecture took between “five and six days to train on two GTX 580 3GB GPUs" [88]; Overfeat alternative models were not fully trained “due to time constraints" [89]; VGG models studied in [90] “took 2–3 weeks" to train using four high-end GPUs; and, while authors in [91] have implemented GoogLeNet using CPU clusters, they also state that their architecture “could be trained to converge using few high-end GPUs within a week".

In this chapter we describe the standard procedure for training Convolutional Neural Networks using the backpropagation algorithm. We also describe how datasets are organized for training and which are the most common datasets used for network benchmarking. Finally, we present the modern approaches for speeding up training found in literature before describing our own method in the next chapter.

5.1 Gradient-based Learning

According to the definition of *learning* given Tom M. Mitchell “a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [94]. When applied to the context of object classification, we can readily identify the task T as being the ability to correctly label images according to a finite set of classes and performance P as a function of the number of correctly classified images. The experience E, on the other hand, depends on the specific choice of learning algorithm.

In general, algorithms for object classification belong to the class of *supervised learning*, a general terminology for algorithms that are trained using sets of labeled data for experience. Gradient-based algorithms are specific supervised learning algorithms that use labeled data and the gradients of the performance measure with respect to classifier's parameters to improve performance itself [95]. As we shall see, these algorithms depend on two main stages namely the *backpropagation* and the *weight update*.

5.1.1 Backpropagation Algorithm

The backpropagation algorithm provides for each labeled sample an estimate on how each weight in the network should be modified in order for the network to correctly classify that sample. It does so by calculating the error the sample produces on a predefined *loss* function, also known as *cost* function or *objective* function.

Loss function

Ideally, the cost function should directly reflect a quantity we want to minimize. In the case of object classification, the most obvious choice for cost function to use during training would be the number of wrongfully classified images in the training set. However, the backpropagation algorithm for training CNNs requires the cost function to be differentiable over the set of weights [95], therefore a surrogate function is needed.

The cost function operates at the network's output, whose discussion has been delayed until now for the sake of simplicity. Usually in the case of CNNs for object classification, the number of output neurons in the last fully-connected layer L is equal to the number of available labels N. The outputs of these neurons are then usually associated to a *softmax* layer defined by the non-linearity $\sigma^S(\cdot)$ in (5.1).

$$y_k^L = \sigma_k^S\left(z_k^L\right) = \frac{e^{z_k^L}}{\sum_{i=1}^{N} e^{z_i^L}} \tag{5.1}$$

As it can be seen from the same equation (5.1), the fact that the outputs of a softmax function are all positive and sum to one helps us interpret the output of a network as a posterior probability of having image x_i being labeled as a particular class k, with $1 < k < N$. Given the probabilistic interpretation of the softmax, a natural choice for cost function is to use the cross-entropy.

The cross-entropy can be defined as a measure of error caused by representing a stochastic phenomenon with the wrong probability distribution. In the case of object classification, each labeled sample is represented as a certain event of its particular class, i.e. each sample is represented as a vector $\mathbf{t} \in \mathbb{R}^N$ made entirely of zeros except for its tth dimension corresponding to its true class t which is set to one. The cross-entropy then measures the difference between this true distribution and the one provided by the output of the softmax layer for each example, as defined in (5.2).

$$C\left(\mathbf{y}^L\right) = -\sum_{k=1}^{N} t_k \log y_k^L \tag{5.2}$$

The cost function, as defined in (5.2), represents the cost for a single example. The total cost for an entire dataset is given the average of the costs of each training sample.

The idea behind the backpropagation algorithm is to identify how learnable parameters such as weights and bias should be changed to reduce the value of the cost function for a given set of training samples. Since CNN are feed-forward networks, each layer in the network can be interpreted as being a function of the previous layers, i.e. the CNN is a composite function of the input image. This interpretation allows the backpropagation algorithm to use the chain rule of calculus

to *propagate backward* the error gradient estimated by the cost function giving the algorithm its name.

In other words, once the loss function has been defined, the backpropagation algorithm allows us to find the partial derivatives of the cost function with respect to the network's trainable parameters, $\frac{\partial C}{\partial w}$ which in principle can be used for finding its minima.

Finding these partial derivatives involves two distinct phases, namely the *forward pass* and the *backward pass*. For ease of understanding, we first describe these two phases for a network containing L fully-connected layers which is applied to a vectorized representation of input image $\mathbf{x}$. A description of how to apply these same concepts to convolutional and pooling layers follows shortly after.

Forward pass

The forward pass is characterized by applying the network to the input. This procedure generates a set of intermediate outputs at the end of each layer. For each neuron j in layer l, it is convenient to store both its *pre-activation* value z_j^l and output a_j^l as defined in (5.3) and (5.4), respectively, where the $\sigma(\cdot)$ is the non-linear activation function associated to the that particular neuron.

$$z_j^l = b_j^l + \sum_i w_{ji}^l a_i^{l-1} \tag{5.3}$$

$$a_j^l = \sigma\left(z_j^l\right) \tag{5.4}$$

Backward pass

We start out backward pass description by defining an intermediate *error* variable δ_j^l for each neuron j at layer l, as seen in (5.5).

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{5.5}$$

For neurons at the last layer L, this error can be directly retrieved as seen in (5.6), assuming that the derivative of the last activation function is well defined over the outputs.

$$\delta_j^L = \sum_{k=1}^{N} \frac{\partial C}{\partial y_k^L} \frac{\partial y_k^L}{\partial z_j^L} \tag{5.6}$$

Considering the common case of using a softmax non-linearity at the output and a cross-entropy cost function, the expression can be reduced to (5.7) where t_j is the label value (one or zero) for class j.

$$\delta_j^L = y_j^L - t_j \tag{5.7}$$

For all other layers $l < L$, the error value can be recursively obtained as follows

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{5.8}$$

$$= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{5.9}$$

$$= \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{5.10}$$

$$= \sum_k \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l) \tag{5.11}$$

Once these errors have been computed, it is possible to find the partial derivatives of the cost function with respect to the weights w_{jk}^l

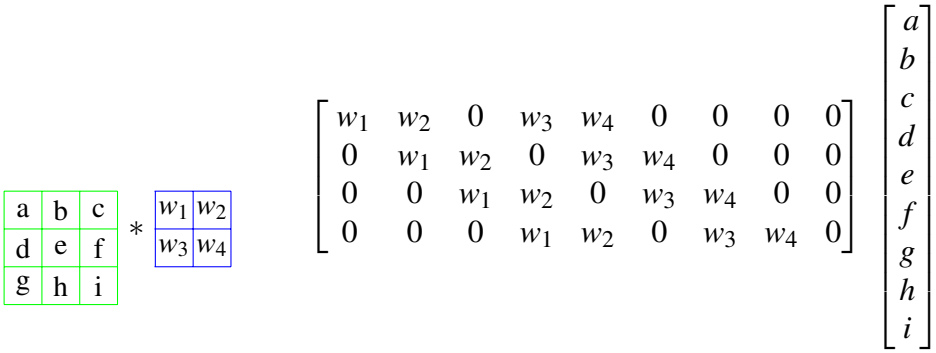
$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{5.12}$$

$$= \delta_j^l a_k^{l-1} \tag{5.13}$$

and biases b_j^l.

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} \tag{5.14}$$

$$= \delta_j^l \tag{5.15}$$



(a) 2D convolution. (b) 2D convolution as matrix-vector multiplication.

Fig. 5.1 2D convolution using matrix-vector multiplication. The concept can be extended for multi-channels inputs and kernels using concatenation.

It is worth mentioning that the partial derivatives stated above are obtained using a single sample in the dataset. In practice, the partial derivatives used during learning are averaged over sets of samples known as *minibatches* usually containing between 16 and 128 sample images.

Convolutional and Pooling Layers

So far we assumed, without loss of generality, that each neuron in layer l was connected to all neurons i in layer $l-1$ with an associated weight w_{ji}. This assumption allows the pre-activations in a layer (weighted sum of inputs) to be represented as the product between a weight matrix $\mathbf{W}$ and a vectorized representation of the layers inputs. In practice, this allows the network to benefit from optimized low-level subroutines such as the Basic Linear Algebra Subprograms (BLAS).

As we can see in Figure 5.1b pre-activations in covolutional layers can also be represented as a matrix-vector multiplication albeit using a very sparse Toeplitz matrix whose efficiency is often compromised. The existence of multiple channels can be accounted through concatenation. Finally, we notice that some pooling layers such as average-pooling, can be interpreted as a convolutional layers. Other layers, such as max-pooling, are however inherently non-linear, and thus cannot be represented using matrix multiplication.

5.1.2 Parameters update

Once the partial derivatives are obtained using the backpropagation algorithm, they are passed to an optimization algorithm that will update the network's parameters and reduce the value of the cost function. In practice, these partial derivatives are represented as a single vector, hence the name gradient-based learning.

Here we present some of the most used approaches for updating parameters using gradient-based methods.

Gradient Descent

Gradient descent is by far the most commonly used optimization technique for training CNNs. The algorithm updates the network's parameters in the opposite direction of the gradient. This is done by simply subtracting from the current vector of parameters the gradient vector weighted by a positive constant α known as *learning-rate*, as seen in (5.16).

$$w^{t+1} = w^t - \alpha \nabla C \tag{5.16}$$

During early stages of training it is common to use large learning-rates and then decreasing it exponentially as the cost function or true accuracy begins to stabilize. In the particular case where the gradient is evaluated over a single sample, the algorithm receives the special name of *stochastic gradient descent* or *online gradient descent*.

Momentum

Gradients evaluated over small minibatches are bound to be noisy and to eventually provide directions outside an optimal path. The idea behind the use of momentum is to update the vector of parameters considering not only the current gradient but also the past ones.

$$v^{t+1} = \mu v^t - \alpha \nabla C \tag{5.17}$$

$$w^{t+1} = w^t + v^{t+1} \tag{5.18}$$

This is done using an intermediate vector v which accumulates previous updates weighted with a *momentum* μ and subtract the current gradient weighted by the learning-rate. Values of μ are typically in the range $[0.8, 0.99]$.

5.2 Dataset and Network setup

In this section we describe how to setup both data and learning algorithms for effective object classification.

5.2.1 Dataset Division and Preprocessing

The final objective of training is to produce a network able to *generalize* well over unseen data. That is, we expect that training using available examples will help the network to correctly classify new images. In order to be able to do so, datasets for object classification are usually divided in the following three subsets:

- **Training Set**: As the name implies, this is the set of labeled data used by the training algorithm to train the network. The accuracy obtained in this set does not reflect the network's ability to generalize since it is measured over the same data that was used for training.
- **Validation Set**: Data in this dataset is used with two objectives. Firstly, accuracy measured over this set is used for verifying the effects of changing hyper-parameters, i.e. the number of layers in the network, the number of kernels per layer and the layer size. Secondly, since data from this set was not used directly for training, accuracy measured over it can be used to verify if the network is learning features useful for generalization or if it is learning only details specific to the training set in a process called *overfitting*. This can be verified in practice by evaluating the accuracy over the validation set after each *epoch*, i.e. after each time the entire training set is presented to the network. If the accuracy evaluated in the training set is much higher than validation, than the network is said to be overfitting.

- **Test Set**: The data in this set should be used only for final evaluation of the algorithm and serves as the best estimate as to how data will effectively generalize in unseen data.

This division allows us to completely isolate the training data from the testing data, which enables us to verify if the network is *generalizing* well on new data. However, it is important to notice computer vision competitions sometimes withholds the labels for the test set, so that it is not uncommon for scientific papers to report accuracies over the validation set.

5.2.2 Regularizers

Increasing the number of layers, channels and kernel sizes of CNN will increase its capacity to learn patterns. What at first might seem a benefit actually has a downside. Networks with high capacity will more likely learn specific patterns of the data instead of learning only relevant information for the task, i.e. they are more likely to overfit. This limits the ability of an network to generalize well on new data. Here we present two methods that help avoid overfitting that are generally known as regularizers since the regulate the networks capacity.

Dropout

During training, neurons in a given layer have their weights updated under the assumption that all neurons are trained together. This leads to the process of co-adaptation where a neuron could be changing its weight just to make up for its neighbor's mistakes, eventually having two or more neurons that learn the same concept. In order to solve this problem, authors in[96] have suggested the use of dropout connections where, during each backpropagation cycle, neurons in a layer are randomly selected and have their output values set to zero, reducing co-adaptation. Usually the percentage of neurons that are turned off during training is set to $p = 50\%$. Once the network is trained, all neurons are turned back on and their outputs are multiplied by p.

Weight Decay

Another method for regularizing the capacity of a neural networks, which is also probably the most common one, is to regularize its weights using a penalty over high values. This is usually done by adding to the cost function an extra term which is the sum of the squares of all weights scaled by a constant λ. This use of ℓ^2 normalization is usually called *weight decay* in neural network literature.

5.2.3 Datasets for Image Classification

Following are some of the most used datasets for object classification. In this section we briefly describe each one of them in term of variability, dataset size and latest obtained classification accuracy when applied.

MNIST

The Mixed NIST (National Institute of Standards and Technology) is a database of 28x28 greyscale images of handwritten digits ranging from 0 to 9. The dataset is made of sixty thousand images for training and ten thousand images for testing. MNIST was very popular in the late 90s and it is still used today for small, proof-of-concept experiments. However, due to its relatively small number of classes, small spatial resolution, and interclass similarity, the MNIST dataset is no longer considered an open problem. As a matter of fact, the classification error in this dataset is currently 0.21% [97].

CIFAR 10-100

Introduced by the University of Toronto, the CIFAR-10 (Canadian Institute For Advanced Research) [98] is a dataset of 32x32 color images of trucks, cars, airplanes, dogs, cats, frogs, deers, ships, horses and birds. Besides being slightly larger in resolution and containing color information, the CIFAR-10 dataset is significantly more challenging than MNIST given the interclass similarity and intraclass variability present in objects and animals. It contains fifty thousand images for training and ten thousand image for testing and current classification error in the CIFAR-10 dataset is 3.47% [86]. CIFAR-100 is similar to CIFAR-10 having one hundred classes with

fewer number of images per class: only five hundred for training and one hundred for testing. The current classification error in this dataset 24.28% [99].

ImageNet

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [87] is an international computer vision competition held each year. One of the challenges in this competition was the object classification task that requires the algorithm to classify 100 thousand images among one thousand classes using training and validation sets of 50 thousand images each. It is currently the preferred dataset for benchmarking classification algorithms given it richness of classes.

Microsoft Coco

Very recently, Microsoft has released the Common Objects in Context (COCO) dataset. The dataset contains per-instance segmented objects to enable precise object localization. It is made of 80 classes consisting of 80k images for training, 40k for validation and 40k for testing. The existence of more than one object instance in each class allows the dataset to have over 500k segmented objects.

5.3 Speeding Up CNN Training

Different attempts to reduce CNN training time using the standard back-propagation technique have been proposed in the literature. Each one of those approaches can be associated to a specific hierarchy of training processes, which in turn allows most of these techniques to be used together.

5.3.1 Convolution Operations

At the lowest level of optimization lies how convolutions are implemented. In literature there are currently three main approaches to speed up convolutions.

Image to Columns (im2col)

A näiv implementation of 2D convolutions requires four for-loops; two for the spatial input dimensions and two for the kernel spatial dimensions. We have seen in section 5.1 how convolutions can instead be implemented using matrix multiplication in order to take advantage of optimized subroutines for linear algebra. Unfortunately, representing a kernel as a Toeplitz can still be inefficient given the matrix's sparsity. In order to solve this problem, authors in [100] have suggested the approach known as *image-to-column* or *im2col*. In this approach the kernel is vectorized into a dense column vector, while each convolved region of the input is vectorized as row vector. This drastically increases the memory consumption due to the data duplication, however speedups of about 2x can be achieved [100].

Fast Fourier Transform

Another possible way to reduce time spent in convolution operations is by working in the frequency domain. Authors in [101] have exploited the fact that kernels applied to an entire minibatch need only be transformed once, which more than makes up for delays caused by Fourier transformations. This approach is also costly in terms of memory usage when the kernels are small compared to the images, however for large minibatch sizes and depending of the architecture it can be twice as fast as common implementations [101].

Minimum Filtering Algorithms

Very recently, the current preference for small 3x3 kernels has revived the interest for minimum filtering algorithms [102]. Authors in [103] have suggested the use of an algorithm similar to im2col that works specifically with 2x2 and 3x3 spatial resolutions. For these sizes of kernels, data can be rearranged as to minimize the number of multiplications required to produce outputs of 2x2 and 3x3 spatial sizes, respectively.

5.3.2 Network Architecture

Each individual layer in a CNN that contains learnable parameters can be seen as learning the statistics of its input. However, after each cycle of backpropagation, parameters are updated, which means that the statistics that a layer l should learn from layer $l-1$ inevitably change, giving rise to the phenomenon known as covariant-shift. In order to solve this problem, authors in [104] have proposed the use of a Batch Normalization layer that normalizes mean and variance of incoming minibatches of feature-maps to speed up training.

5.3.3 Network Reuse

At a highest level of abstraction we find the concept of network reuse. Authors in [105] have suggested the use of function-preserving transformations to train deeper (more layers) and wider (more channels) networks starting from shallower and narrower ones. Their main idea is that training need not always start from scratch instead, new layers and kernels can be added to existing architectures and continue training. Author report a reduced number of data required for the expanded network to achieve its target accuracy of almost 50%.

As we will see in chapter 7, our approach can be seen as a mixture between of highest level of optimization level and the lowest one. It relies on scaling the spatial dimensions of convolution kernels and input images, which allows us to keep the levels of representation that are usually associated with the number layers and kernels per layer.

Chapter 6

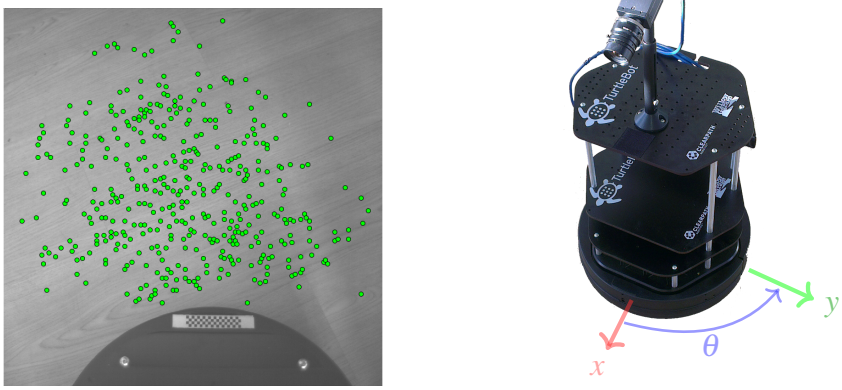
CDVS in Robotic Visual Navigation

In chapter 2 we described in depth the building blocks of a monocular Visual SLAM system, while chapter 3 was dedicated to describing both SIFT visual descriptor and MPEG Compact Descriptor for Visual Search. The current chapter is dedicated to the implementation and analysis of an indoor monocular navigation system using MPEG-CDVS. We analyze the effects of MPEG-CDVS compression modes in terms of extraction times, matching times, and storage requirements and compare the results with the SIFT descriptor for different types of floorings. Next, we verify the effect of compression in visual odometry and propose a probability framework for detecting loops based on the standard's suggested similarity metric referred to as *local score*.

Finally we compare our results with a laser-scanner setup to show that how our visual navigation system produces better results in large indoor spaces.

6.1 Experimental Setup

In this work, a robot carrying a camera navigates through an indoor environment while taking pictures of the floor below it in order to estimate its pose. In particular, our testing robot consists of a Turtlebot 2 from Clearpath Robotics mounting a calibrated, downward-facing, 4.1-megapixel Point Grey Grasshopper 3 camera, as seen in Figure 6.1b.



(a) Detected Keypoints (b) Turtlebot 2

Fig. 6.1 Robot viewpoint and relative coordinate frame.

We choose to use a downward-facing camera for two reasons: Firstly, the setup mitigates the effects of dynamic agents such as humans or other moving objects that can interfere with motion estimation; secondly, the floor surface can be well approximated as a perfect planar surface, which makes motion calculations from pixel simpler.

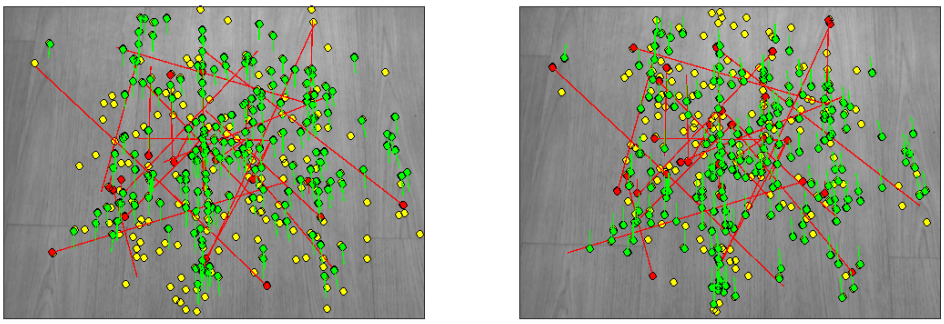
The robot's starting position and heading define both origin and x-axis of a global coordinate frame. This coordinate system then becomes uniquely defined as we choose the z-axis to point upwards, so that points on the ground have all coordinate $z = 0$. We assume the environment floor to be a planar surface so that, for each time step $k > 0$, the robot's pose is given by $\mathbf{x}_k = [x_k, y_k, \theta_k]^T$, where x_k and y_k indicate the robot's coordinates and θ_k is the robot's current heading:

$$[x_k, y_k]^T = R_{(\Delta\theta_{k-1,k})}[x_{k-1}, y_{k-1}]^T + T_{k-1,k} \tag{6.1}$$

$$\theta_k = \theta_{k-1} + \Delta\theta_{k-1,k} \tag{6.2}$$

Final motion between time steps $k-1$ and k can be modeled as a rotation followed by translation. Pose at $t = k$ can be recursively obtained from (6.1), where $\Delta\theta_{k-1,k}$ is the rotation angle estimated between time steps $k-1$ and k, $R_{(\Delta\theta_{k-1,k})}$ is the rotation matrix for that same angle, and $T_{k-1,k}$ is the translation vector. Both parameters are

obtained from the set of matching points between images I_{k-1} and I_k as described earlier in subsection 2.1.2.



(a) $t = k$ (b) $t = k+1$

6.1.1 Software implementations

When implementing software for CDVS-based visual odometry, we followed a modular approach envisioning not only our experiments, but also future use cases such as multi-agent SLAM, where a global map would be constructed based on information retrieved from more than one robot, and other applications not entirely tied to robotic navigation, such as object recognition.

In this context, we chose to use the Robotic Operating System (ROS) [106] development platform to deploy for our robotic navigation system. ROS is the *de facto* standard software development framework for robotics which provides hardware abstraction, low-level device control, message-passing across, and a collection of libraries for navigation. The basic blocks in ROS are called *nodes*, which communicate with each other through *messages* in order to perform specific task.

In this context, we have developed the following nodes in ROS using the C++ programming language:

- `ptGreyImage`: Extracts and resizes images from the Grasshopper camera to VGA resolution. The node publishes these images along with the camera's intrinsic and extrinsic parameters.
- `cdvsExtract`: Receives the sequence of images from the `ptGreyImage` node, extracts and publishes MPEG-CDVS bitstream descriptor for each image. This

node encapsulates the MPEG-CDVS Test Model's [107] implementation of both CDVS Feature Detector and Extractor.

- `cdvsMatch`: Receives a sequence of MPEG-CDVS bitstreams generated by the `cdvsExtract` node and performs pairwise matching including the suggested DISTRAT geometry consistency check. This node encapsulates the MPEG-CDVS Test Model's [107] implementation for feature matching.

- `FeatureOdometry`: Receives a list of matching pixels from `cdvsMatch` along with camera information from `ptGreyImage`. The node further reduces the number of outliers using a limited number of RANSAC iterations to finally publish odometry information to the ROS network.

Nodes were distributed over two computers as depicted in Figure 6.3. The workstation consists of an Asus UX51V with and Intel i-7 3632QM 2.20 GHz CPU and 3.00 GB of RAM, while the Turtlebot 2 carries an Asus F201E laptop with an Intel Celeron 1.50 GHz CPU and 3.00 GB of RAM.

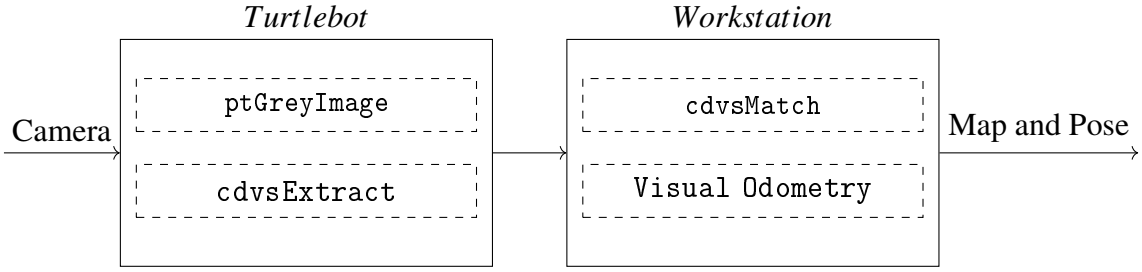
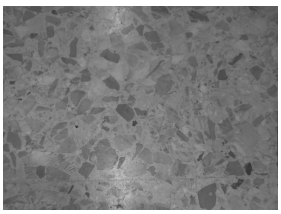
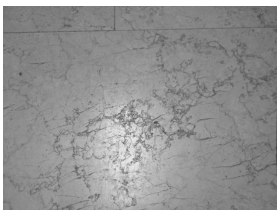


Fig. 6.3 ROS Nodes for MPEG-CDVS Visual SLAM system.

Communication among nodes is subject to network overhead and latency, which would eventually interfere with the estimation of extraction and matching times. Therefore, for experiments that require time evaluation we have used the MATLAB interface of the MPEG-CDVS Test Model [107] for feature extraction and matching and the MATLAB interface for VLFeat SIFT [54]. The MPEG-CDVS Test Module actually uses code from VLFeat for generating the intermediate SIFT features described in chapter 3. This allows for fair comparison in terms of processing speed.



(a) Mosaic



(b) Marble

(c) Red Tiles

(d) Printed Wood

(e) Dotted Tiles

Fig. 6.4 Different types of floorings commonly found in indoor environments. Names were assigned according to the flooring's visible attributes.

6.2 Preliminary Experiments

Our initial experiments serve to analyze the possible benefits in using MPEG-CDVS over the SIFT visual descriptor in terms of both extraction and matching times, and storage requirements. We also evaluate the usage of the standard's suggested *local score* as a possible metric for detecting loops. During these experiments we follow the setup defined in 6.1 and drive the Turtlebot in a straight line for 10 meter while taking pictures of the floor below. We repeat this process for the five different types of floorings depicted in Figure 6.4 in order to verify that the tests generalized well over other environments.

6.2.1 Effects of Feature Selection and Compression

We begin our evaluation by measuring the average number of features extracted in each type of floor and the times required for feature extraction and matching. As we have seen in Section 3.2.3, MPEG-CDVS does not extract local feature from all keypoints. Instead, it limits the number of features in the final CDVS bitstream according to the number of bits allowed by each compression mode. Moreover, the compression mode also establishes the number of dimensions used for each CDVS local feature vector, which further influences matching time.

These characteristics are important when building large maps using limited hardware resources. They allow system designers to indirectly limit motion estimation times between frames (feature extraction and matching times) and map storage requirements (number and size of features) by choosing the an appropriate compression mode.

Storage Requirements

In this experiment we compare CDVS' various modes of compression with the SIFT descriptor in terms of storage requirements. Figure 6.5 shows the effect of feature selection in limiting the number of local features generated for each image.

We notice that for each compression mode the average number of local descriptors extracted is almost independent from the particular type of flooring being used. This, however, is not true for the SIFT descriptor. We see in the same Figure 6.5 that, although some floorings do provide similar number of SIFT descriptors in average, like *dotted* and *red tiles* which generate around 900 interest points in average, other types like *mosaic* might as well double this number.

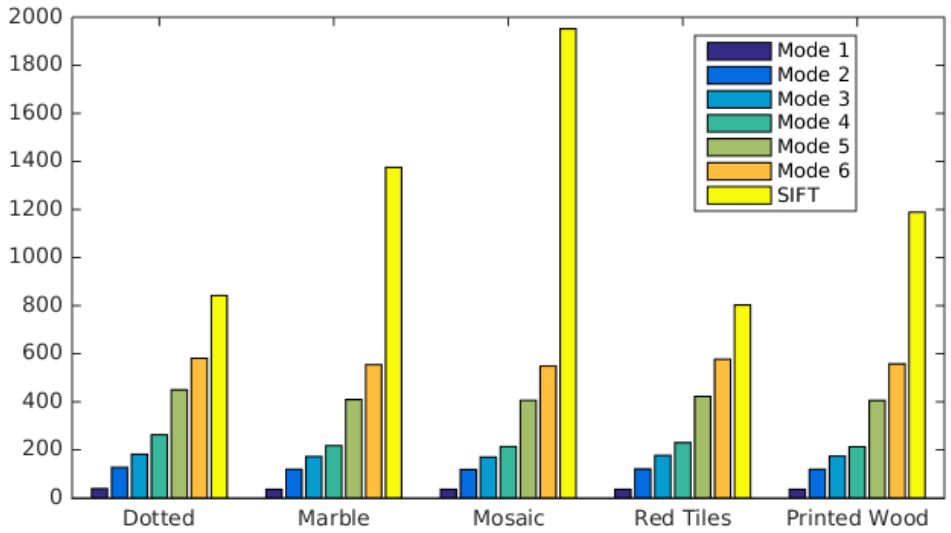


Fig. 6.5 Average number of extracted local descriptors per image for each type of flooring.

In terms of memory efficiency, feature selection has a clear effect on reducing storage requirements. For example, a single image taken from a *mosaic* floor would

in average require almost 300 kB of memory if SIFT descriptor were to be used, while CDVS would require at most 16 kB at its least compressed mode.

Extraction Times

Another positive effect of feature selection is the reduction of extraction time as reported in Table 6.1. Since feature selection is made based on keypoints' characteristics, only small subset of keypoints will be processed. This effect is translated into similar extraction times across the various floorings for a given compression mode. Again this is not seen for the SIFT descriptor, which in cases like the *mosaic* flooring, can be more than an order of magnitude slower than CDVS' least compressed mode 6.

floor types	*mode 1*	*mode 2*	*mode 3*	*mode 4*	*mode 5*	*mode 6*	*SIFT*
Dotted Tiles	16.2	15.4	15.5	16.2	18.9	21.0	217
Marble	15.6	15.3	15.3	16.3	18.9	21.4	295
Mosaic	15.9	15.8	16.0	18.9	22.4	22.3	388
Red Tiles	14.6	14.8	14.7	15.5	18.1	21.0	209
Printed Wood	15.2	15.2	15.3	16.0	18.8	21.0	270

Table 6.1 Average extraction times per image in milliseconds for each CDVS mode of compression and SIFT.

Matching Times

Having a limited number of descriptors per image will also limit the time spent for comparing two images. However, gains in matching speed should be even more pronounced due to CDVS compression scheme. Not only does compression reduce the number of dimensions for each local feature descriptor (modes 1-5) but it also applies ternarisation allowing the use of ℓ_1-norm evaluation instead of the more costly ℓ_2-norm.

In this experiment we have measured the times for matching descriptors from consecutive frames. Table 6.2 confirms this behavior where matching SIFT descriptors between two consecutive pictures of *mosaic* floor takes nearly 20 times longer than matching CDVS descriptors in mode 6.

floor types	*mode 1*	*mode 2*	*mode 3*	*mode 4*	*mode 5*	*mode 6*	*SIFT*
Dotted Tiles	0.26	0.93	1.15	1.97	4.51	7.87	91
Marble	0.18	0.55	0.85	1.29	3.31	6.62	242
Mosaic	0.18	0.54	0.84	1.26	3.31	6.65	490
Red Tiles	0.21	0.53	1.01	1.62	3.84	7.39	84
Printed Wood	0.19	0.67	0.91	1.35	3.51	7.05	182

Table 6.2 Average matching times per image in milliseconds for each CDVS mode of compression and SIFT.

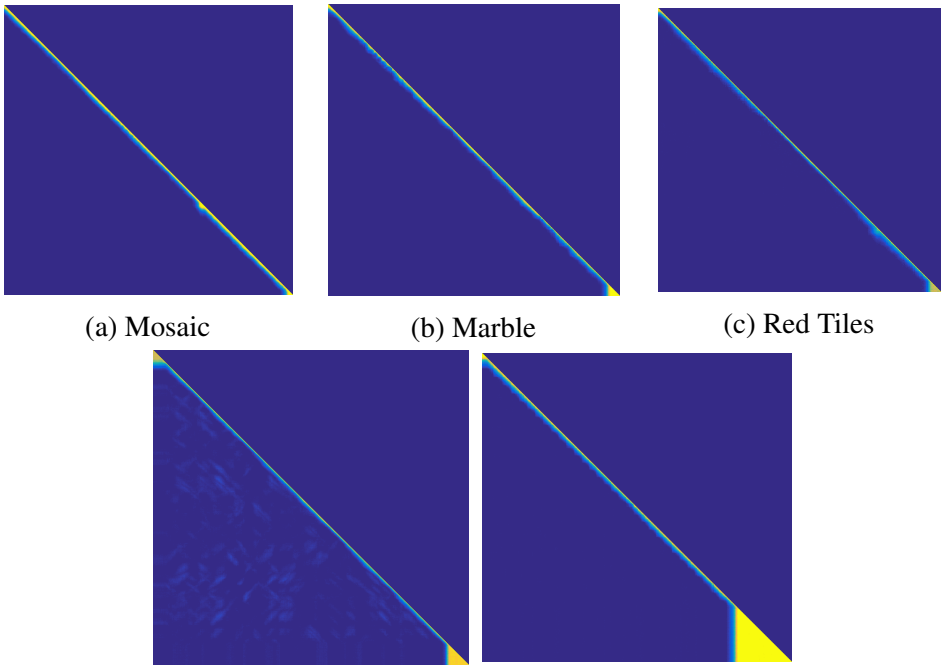
6.2.2 Distinctiveness of CDVS local score

Besides enabling fast extraction and matching and requiring small storage, robotic navigation requires visual features to be distinct enough to correctly detect loops in a path. In this experiment we match each image with all the previous ones in the sequence using CDVS suggested *local score* to measure similarity. We repeat this process for all modes of compression and evaluate the suggested metric as a measure of distinctiveness over short and long displacements.

Distinctiveness in this context means to have high *local score* for pairs of images with much overlapping regions and very low *local score* otherwise. Since images were taken in a sequence during robotic motion, those that are close in the sequence are also spatially next to each other, and thus should have high local score.

A visual representations of these matches using compression mode 6 is given in Figure 6.6 where pixel intensities in position (i, j) represent the local score between current image i and a previously visited image j. Since we only match current images with previous ones, each matrix representing the matches is triangular. In order to allow for a fair visual comparison, the matrices values have been normalized. Yellow pixels mean high local score while dark blue pixels indicate a low score. The presence of small, bright triangles seen at the lower end of each matrix indicates when the robot had stopped.

Ideally, these matching matrices should display increasingly intensity of pixel values (yellow) in regions near each diagonal and very low values (dark blue) everywhere else. The natural randomness intrinsically associated to the production of most of flooring types enables them to have a relatively thick principal diagonal and to display very low matching scores where no overlap occurs. The one noticeable



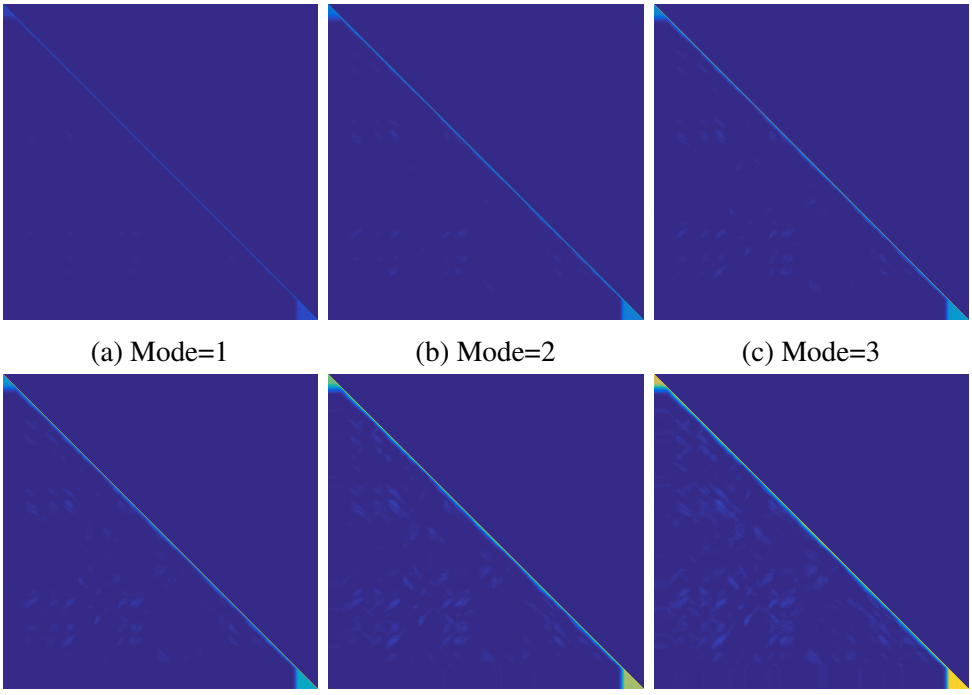
(a) Mosaic (b) Marble (c) Red Tiles

(d) Printed Wood (e) Dotted Tiles

Fig. 6.6 Visual representation of *local score* for different flooring types.

exception occurs for the printed wood floor. This particular artificial type of flooring is made of printed repetitive patterns. The effect of such patterns appears as bright spots on its matching matrix and could be particularly harmful for loop-detection since it leads to erroneously detected loops. We can observe the evolution of these spots and the diagonal thickness in Figure 6.7 as we vary the compression mode. We can also verify in Figure 6.8 that the number of matches produced with SIFT generates a similar pattern.

It is clear that the diagonal thickness decreases for lower modes of compression. This phenomenon happens to all flooring types and it is due to the fact that CDVS will use fewer keypoints with shorter local descriptors to represent each image. This makes it difficult to correctly match images that are even just slightly displaced with respect to one another. Therefore, as expected, lower modes of compression can be considered to offer less distinctive local descriptors. On the other hand and for the same reason, bright spots on the wooden pattern become even more visible as the level of compression increases, which makes this particular kind of flooring the worst case scenario and also our study case to test CDVS for loop detection.



(a) Mode=1 (b) Mode=2 (c) Mode=3

(d) Mode=4 (e) Mode=4 (f) Mode=6

Fig. 6.7 Visual representation of *local score* for the Printed Wood floor using different compression modes.

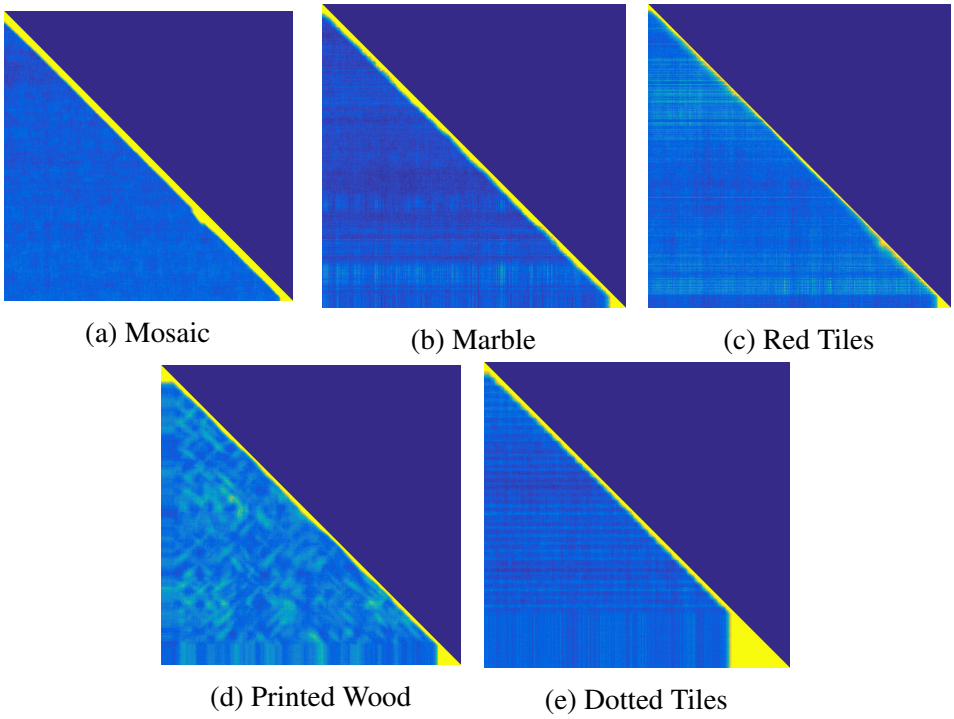
6.3 Loop-Closure Detection

The analysis made in the previous Section led us to try and use MPEG-CDVS *local score* as a metric for detecting loops and thus provide a complete approach to Visual SLAM. But before being able to do so, we need to define precisely what *loop* means in terms of *local score* and how this can adapt to different flooring types.

Given the stochastic nature of the SLAM problem, we decided to use a probabilistic approach for loop detection which allows immediate interpretation on the "belief" of finding a loop, and can also be also integrated with *a posteriori* optimizers.

6.3.1 Loop Definition

The use of a downward-facing camera allows for a natural definition of loop based on the intersection of imaged regions. For images I_a and I_b taken along the robot's path, we define loop as a function of the overlap ratio between the floor areas observed by



(a) Mosaic (b) Marble (c) Red Tiles

(d) Printed Wood (e) Dotted Tiles

Fig. 6.8 Visual representation of SIFT for different floor types .

these two images. So given the area of intersection, area$(I_a \cap I_b)$, and the respective area of union, area$(I_a \cup I_b)$, a loop can be defined as in (6.3)

$$J = \frac{\text{area}(I_a \cap I_b)}{\text{area}(I_a \cup I_b)} \tag{6.3}$$

$$\text{loop}(I_a, I_b, r) = \begin{cases} 1 & \text{if} \quad J \geq r \\ 0 & \text{if} \quad J < r \end{cases} \tag{6.4}$$

where r is the threshold that defines the minimum overlap ratio for which two intersecting images can be considered a loop. In this work we set this threshold to $r = 0.33$, which roughly amounts for an area intersection of 50% when I_a and I_b have the same areas.

6.3.2 Loop Probability

Loop detection as defined in (6.4) requires the knowledge of how much area of intersection there is between the two images. In order to indirectly measure the

probability of having a particular area ratio we use the *local score* given between two images so that

$$P(\text{loop} = 1|\text{score} = s) = P(J \geq r|\text{score} = s) \tag{6.5}$$

$$P(J \geq r|\text{score} = s) = \frac{P(J \geq r, \text{score} = s)}{P(\text{score} = s)} \tag{6.6}$$

The conditional probability in (6.5) can be experimentally estimated through (6.6) by combining the knowledge of the camera's parameters with a source of relative motion estimation. This process will be described in depth during the next section.

6.4 Training of Proposed Model

6.4.1 Estimating Loop Probability

A camera's intrinsic and extrinsic parameters define the camera's pose with respect to the world and also allow us to make real world measurements directly from images. These parameters can also be used to circumscribe observed regions by projecting the camera's field-of-view onto the imaged floor. Once the projected areas of images I_a and I_b are known, it is sufficient to know their relative positions to estimate their area of intersection and thus to be able to evaluate the overlap ratio J.

Relative motion during training was obtained using the robot's odometry, and although odometry suffers from error accumulation after long trajectories, it does provide dependable relative motion estimations over short range distances. Moreover, images that are relatively distant from each other, will have zero overlapping region an therefore error accumulation will constitute a problem. During training phase relative motion was obtained by using a Kalman filter that combined information from both wheel odometry and a robot's internal gyroscope during the experiment described at the beginning of this section.

By combining these pieces of information with the local scores of each analyzed matching pair, we can generate for each compression mode a loop detection probability curve as defined in Eq. 6.6. The resulting curves, as seen in Figure 6.9, show

local score	*mode 1*	*mode 2*	*mode 3*	*mode 4*	*mode 5*	*mode 6*
Hypothesis	10	14	15	18	23	25

Table 6.3 Hypothesized values for local score loop detection.

the probability of two images having more than 50% of intersection for each mode given a local score. Lower compression modes achieve certainty at lower values of local score. This is due to the fact that low compression modes also have fewer descriptors to be used during match.

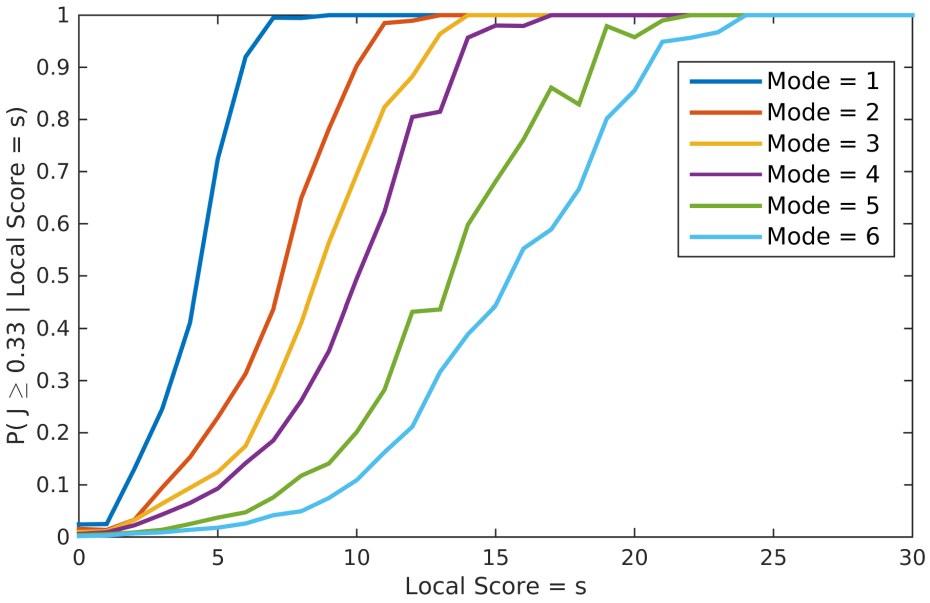


Fig. 6.9 Cumulative loop probability for printed wood floor.

From these curves we select the minimum values values of local score s that guarantee loop detection for each compression mode. These hypothesis values are reported in Table 6.3 and used to define the loops during the final experiments discussed in Section 6.5.

6.5 Experimental Results

Partial results from Section 6.4 have led us to try our loop-detection technique on the most challenging flooring for loop-closure, i.e. the flooring most susceptible to false-loop detection.

In this experiment, the robot navigates through indoor office for about 110 meter while taking a total of 7154 images of its printed wood floor and performing loops before finally going back to its original position.

6.5.1 Visual Odometry for Testing

In order to demonstrate that our approach could be applied to a vision-only navigation system having no other sensors such as gyroscope or wheel encoder, we have decided to also implement VSLAM using visual odometry. Our robot setup follows the one in [108]. However, although we do use a similar approach to obtain odometry, our main concern in this work is the correct detection of loops for VSLAM.

Depending on system requirements, less complex feature descriptors such as Brisk [68] or Harris [61] could be used to generate odometry, while CDVS would be used just for loop detection. However, since local features from each image will already be available, we choose to use CDVS local descriptor to generate visual odometry as well.

As described in chapter 2, for each pair of consecutive images I_{k-1} and I_k we perform feature extraction and match of MPEG CDVS descriptors, which results into two sets of $N > 2$ matching coordinate pairs. We combine these pixel coordinates with the camera's calibration information and produce the sets P_{k-1} and P_k each containing the 3D coordinates for the N matching pairs. From these points we retrieve the estimations for both rotation and translation (6.1).

We first use the sequence of images to generate the path's visual odometry as described in Section 6.4 for all except the first compression mode, which was unable to generate enough matching points between consecutive images. For those modes capable of estimating translation and rotation from consecutive images, we report their respective paths in Figure 6.10 where we use the room's blueprint as reference map.

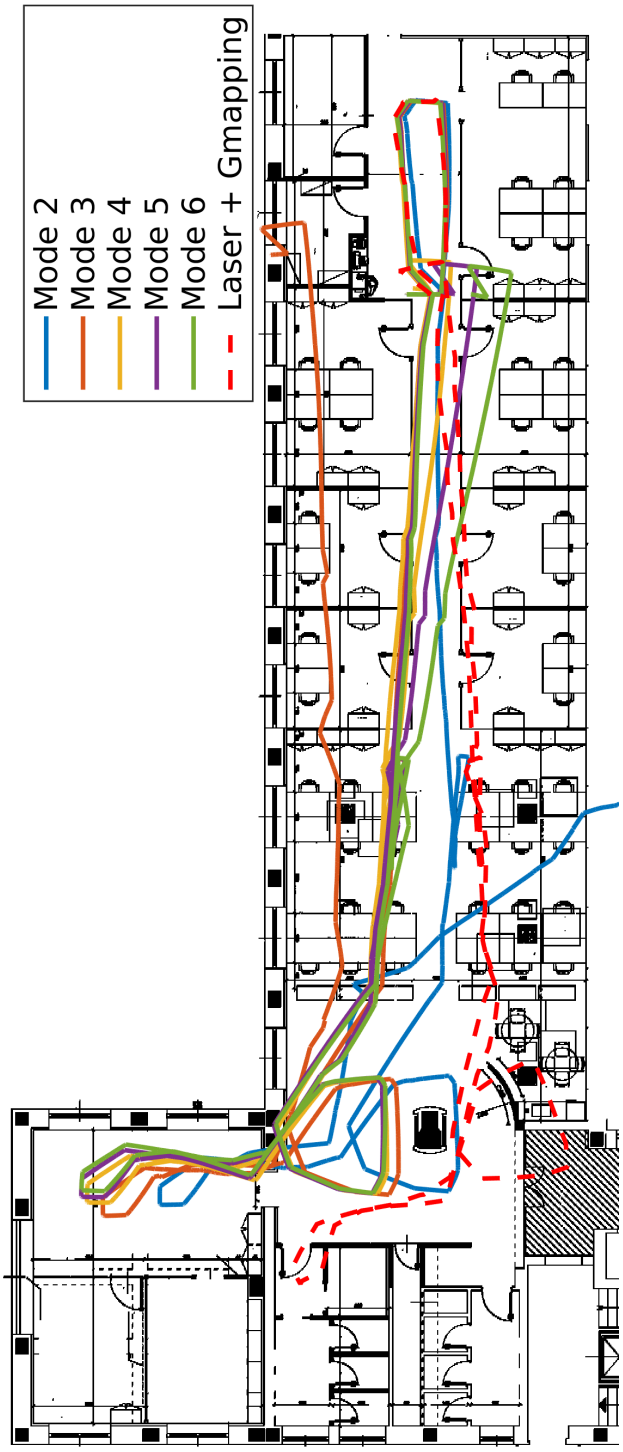


Fig. 6.10 Path comparison using visual odometry.

We then perform loop detection as described in Section 6.4 where for each image pair whose local score was above the hypothesized value in Table 6.3 a loop was declared.

local score	*mode 1*	*mode 2*	*mode 3*	*mode 4*	*mode 5*	*mode 6*
Experimental	–	20	16	18	24	27

Table 6.4 Experiemtal threshold values for local score loop detection.

For each compression mode, we have represented data from visual odometry and loop constraints as a path graph so that the robot's trajectory could be optimized using the LAGO graph optimization software [28], whose purpose is to find a coherent sequence of poses that better describe all loop and odometry constraints, and thus perform VSLAM.

During these experiments, we have observed that the proposed local scores thresholds loop-detection found earlier were slightly too permissive and still allowed for small amount of false-positive loops to be detected. This fact has led us to empirically increase these thresholds until reasonable results were obtained. We report these manually optimized values in Table 6.4 and notice that these values differ very little from the hypothesized ones, which in turn proves that our method is valid. The resulting trajectories for each compression mode using the experimental thresholds can be seen in Figure 6.11.

A visual inspection between the two figures reveals the improvements obtained for all compression modes when loops are correctly detected. Except for compression mode 2, all improved trajectories pass through the hallway, enter and exit the northwest room and respect the physical constraints present in the map. However, in order to have a more quantitative measure of such improvements we report in Table 6.5 the pose difference between starting and ending poses in the trajectory, which ideally should be none.

To highlight the gains in terms of both storage savings and matching times with respect to SIFT, we have compared the amount of memory required to save descriptors for all 7154 images using each compression mode and also report the time necessary to compare the last image in the sequence with all previous one. We report these values in Table 6.6.

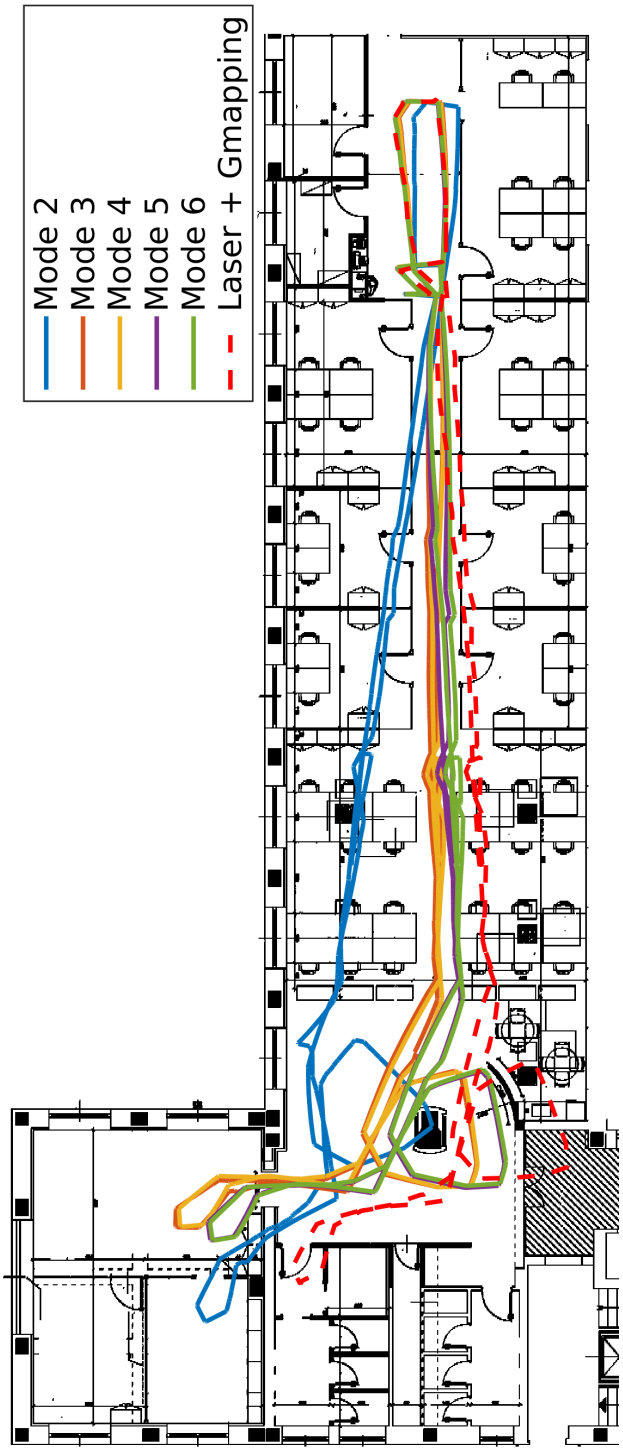


Fig. 6.11 Paths optimized using LAGO.

	Visual Odometry			*Visual SLAM*		
	Δx (m)	Δy (m)	$\Delta\theta$ (rad)	Δx (m)	Δy (m)	$\Delta\theta$ (rad)
Mode 2	17.35	-6.58	-0.86	0.0725	-0.0088	0.0075
Mode 3	-4.36	1.27	0.03	0.0355	-0.0115	0.0001
Mode 4	0.22	0.19	-0.13	0.0359	-0.0149	0.0086
Mode 5	1.01	0.09	-0.17	0.0302	-0.0011	-0.0249
Mode 6	2.10	0.00	-0.23	0.0221	-0.0056	-0.0128

Table 6.5 Relative pose errors between starting and final position for both visual odometry and VSLAM.

property	*mode 2*	*mode 3*	*mode 4*	*mode 5*	*mode 6*	*SIFT*
Storage (MB)	7.67	14.63	28.59	56.55	112.43	1213.84
Time (s)	4.23	6.62	9.62	27.27	58.32	1264.20

Table 6.6 Storage requirement for all 7154 images and total matching time between last sequence image and all previous ones.

6.5.2 Comparison with laser-scanner

Finally, in order to compare our proposed method with existing state-of-the-art frameworks for indoor SLAM, we also report on both figures the path generated using a Hoyuko laser-scanner optimized with the widely used Gmapping algorithm [109].

At first sight, results from laser scanner can be considered incorrect and unreliable. This occurs because laser scanner was unable to create a precise map of environment and thus was unable to reproduce its path correctly on the real world map. This becomes evident in Figure 6.12 where the path generated by the laser seems to be coherent to its self-generated “bent" map. Our method clearly does not suffer from the same issue.

6.6 Result Analysis

In this work we have proposed the use of MPEG-CDVS in a SLAM framework for loop-detection in an indoor environment.

We have shown experimentally that CDVS' feature selection serves not only to reduce the final descriptor size but also to significantly speed up feature extraction and matching. In our practical experiment CDVS's least compressed mode was shown to be over 20 times faster than SIFT during matching time and to require 10 times less storage space and still able to provide for correct loop-detection. Moreover, feature selection also implicitly sets upperbounds for both storage and extraction and matching times, which allows for system designer to make dependable prediction for VSLAM systems requirements regardless of the specific floor type where it will be used.

Finally, when we compared our approach to a laser scanner, we have seen that CDVS has generated far better results. This is because a downward facing camera benefits from making local nearby measurements, while laser-based approaches provide poor results where uncertainty about distant landmarks in large open spaces reduces their overall accuracy.

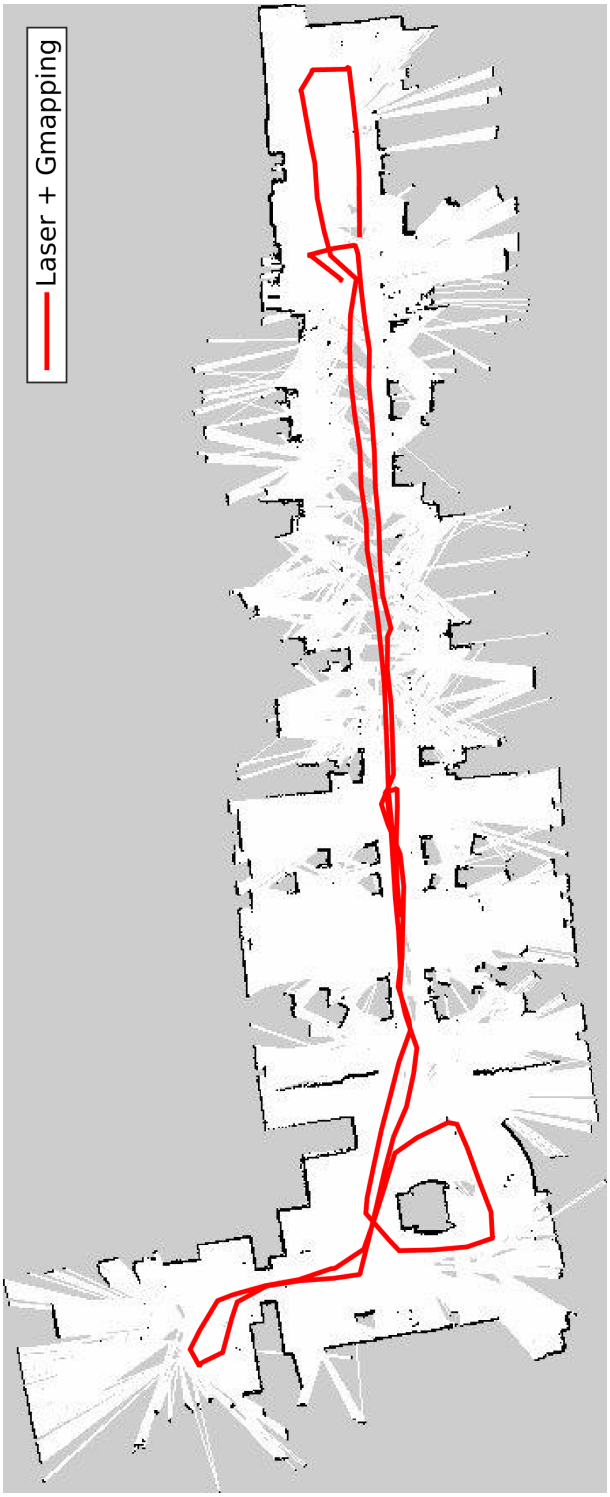


Fig. 6.12 Map and path generated with a laser scanner and Gmapping algorithm.

Chapter 7

Fast Training of Convolutional Neural Networks using Scaled Kernels

We have seen chapter 4 how Convolutional Neural Networks (CNN) have become the ubiquitous tool for solving many computer vision problems such as image classification, and image segmentation [110] at the cost of long training periods. In chapter 5 we saw that there are many approaches that try and mitigate this problem. For early architectures, the FFT approach helped reduce the number of multiplications performed by a convolutional layer in exchange for extra-memory usage [101], however, the current preference for smaller 3×3 kernels has revived the interest for minimal filtering algorithms [102] as seen [103]. Batch Normalization layers also helped reducing training time by reducing the number of iterations necessary for convergence.

In this chapter, we show that the overall training time of a *target* CNN architecture can be reduced by exploiting the spatial scaling property of convolutions during early stages of learning. This is done by first training a *pre-train* CNN of smaller kernel resolutions for a few epochs, followed by properly rescaling its kernels to the *target*'s original dimensions and continuing training at full resolution. Moreover, by rescaling the kernels at different epochs, we identify a trade-off between total training time and maximum obtainable accuracy. Finally, we propose a method for

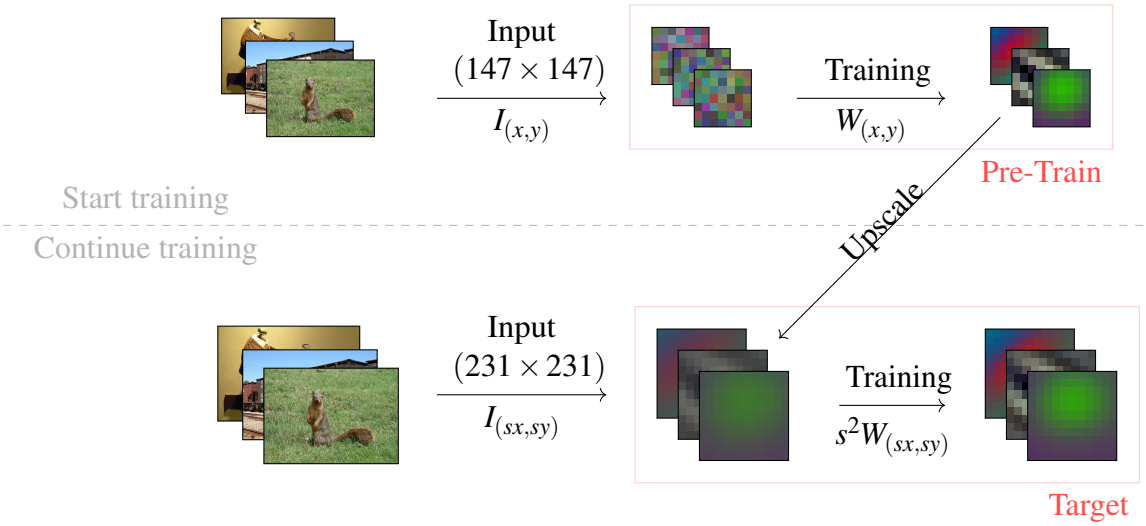


Fig. 7.1 Training starts with a *pre-train* network of smaller convolution kernels and input images. After a number of epochs, kernels are resized to the *target*'s resolution and training continues as scheduled.

choosing when to rescale kernels and evaluate our approach on recent architectures showing savings in training times of nearly 20% while test set accuracy is preserved.

7.1 Proposed Method

This section describes the motivations behind spatially scaling kernels to speed up the overall CNN training procedure.

7.1.1 Spatially Scaling Convolutions

The time scaling property of convolutions states that convolution between two time-scaled signals $I(sx)$ and $W(sx)$ can be obtained by time-scaling the result of convolving the original inputs $I(x)$ and $W(x)$, followed by an amplitude-scaling of $1/|s|$.

This property can be extended to continuous 2D signals (Equations (7.1) and (7.2)) where in this case it is better denoted as the *spatial scaling* property.

$$I(x,y) * W(x,y) = Y(x,y) \tag{7.1}$$

$$I(sx,sy) * W(sx,sy) = \frac{1}{s^2} Y(sx,sy) \tag{7.2}$$

If applied to the context of CNNs, this property would suggest that the output of a convolutional layer could also be obtained from the spatially downsized versions of both the layer's input and its convolution kernels.

Benefits of this possibility can be seen in Equation 7.3, which represents the number of multiplications performed by a convolutional layer l, having C_{l-1} and C_l input and output channels, when both $h_l \times h_l$ input and $k_l \times k_l$ convolution kernels are spatially scaled by a factor s_l.

$$M(l,s_l) = C_{l-1} \left(\frac{k_l}{s_l}\right)^2 \left(\frac{h_l}{s_l} - \frac{k_l}{s_l} + 1\right)^2 C_l \tag{7.3}$$

When compared to its unscaled version, i.e. $s_l = 1$, one can establish the bounds in Equation (7.4) by considering both extremes $h_l = k_l$ and $(h_l - k_l) \gg s_l$, which in turn guarantees a minimum reduction in the number of multiplications proportional to $1/\left|s_l^2\right|$.

$$M(l,1)/s_l^4 < M(l,s_l) \leq M(l,1)/s_l^2 \tag{7.4}$$

The spatial scaling property is, of course, valid only in the continuous domain. Working with downsized versions of inputs will usually result in irreversible loss of spatial resolution and accuracy. However, as shown in the following sections, for moderate values of s_l this property can still be exploited during early stages of training, where the network is still learning the basic structures for its kernels. This can be done by first training an otherwise identical network of smaller kernel resolutions, followed by an upscaling to the target kernel resolution and continuing training.

Target	Pre-train	s_l	s_l^2	s_l^4
1×1	1×1	1.00	1.00	1.00
3×3	2×2	1.50	2.25	5.06
5×5	3×3	1.67	2.77	7.71
7×7	5×5	1.40	1.96	3.84
11×11	7×7	1.57	2.49	6.09

Table 7.1 Suggested kernel resolution conversions with relative resize factors and bounds.

7.1.2 Pre-training Setup

During the pre-training phase, a *pre-train* network of architecture similar to the *target* one having downscaled kernel resolutions shall be trained. Equation (7.3) guarantees that during this phase the training process will run faster.

Generating this *pre-train* network from a *target* network architecture requires choosing new spatial resolutions for each convolutional layer as well as making the necessary adjustments so that fully-connected layers will have compatible input-output dimensions.

Convolution kernels

When deciding on the new kernel resolutions, a trade-off between speed and accuracy must be considered. Selecting kernels much smaller than the originals will cripple the layer's ability to extract and forward high-frequency information, while too conservative downscaling will lead to insignificant improvements in training speeds. Therefore, we suggest the use of Table 7.1 for choosing the pre-train kernel resolution given a target kernel resolution. We have found those values to provide a good compromise between these factors after the complete training process.

Once the new kernel sizes have been chosen, it is necessary to adjust the network's internal parameters so that each convolutional layer closely satisfies Equation 7.2. In order to do so, it is important to observe that a CNN architecture for image classification usually reflects two distinct stages of processing. The first stage contains various layers of convolution and pooling that act as feature extractors. They output *feature-maps* whose spatial resolution depends on that layer's input resolution. The second stage, on the other hand, acts as a classifier and can be identified by the presence of *fully-connected* layers of fixed input and output dimensions.

Solving the input-output dependencies present in the feature extraction phase should be done starting from the input image itself since it has no constraints with any previous layer. Input images also have large spatial resolution so that it should be straightforward to choose a smaller integer length whose ratio with respect to the original image will closely approximate the first chosen scaling factor s_1.

Upper layers will generally not have such flexibility since the following feature-maps will usually have lower spatial resolution. For these layers, the input-output scaling requirement are met by spatially padding or cropping the incoming feature-map.

Fully-connected interface

Special attention must be paid to the interface between convolutional layers and fully-connected ones since the latter require fixed input sizes.

Activations in a fully-connected layer can be represented as a matrix-vector multiplication where each column in the weight matrix is associated to a particular neuron and the number of rows defines the layer's input size. In this interpretation, before entering a fully-connected layer, feature-maps having C channels and spatial resolution of $W \times H$ must be reshaped into a vector representation $\mathbf{x} \in \mathbb{R}^{CHW}$.

Since the new spatial dimensions in the *pre-train* architecture will produce feature-maps of smaller resolutions $\tilde{W} \times \tilde{H}$, the weight matrix in the fully-connected layer must be adapted accordingly. In our representation, this means that the number of inputs (rows) in the weight matrix shall be reduced from CHW to $C\tilde{H}\tilde{W}$, while the number of output neurons (columns) n_{out} is kept invariant. A visual representation of this procedure is seen in Figure 7.2.

Subsequent layers should need no further modification and the *pre-train* network can be trained until a given stopping criterion is met, e.g. classification accuracy on a validation set starts to plateau.

7.1.3 Resizing and Continuing Training

Once the stopping criterion has been reached by the *pre-train* network, its structure must be modified back to the original *target* network.

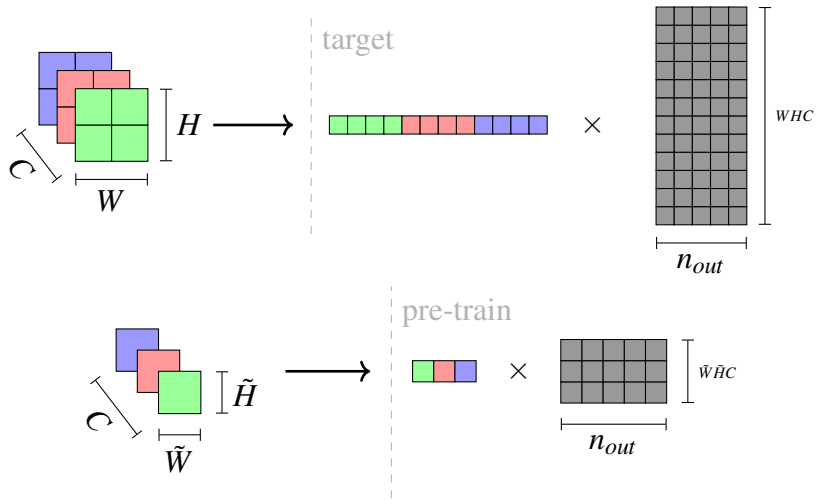


Fig. 7.2 Visual representation of the interface between convolutional and fully-connected layers. Feature-maps from a convolutional layer are first vectorized before entering a fully-connected layer, whose weights are usually represented in matrix form. The number of input must be selected according to the new feature-map spatial resolution $(\tilde{W},\tilde{H})$ and the number of output neurons n_{out} is kept invariant.

Convolution kernels

As seen in Equation 7.2, just spatially resizing both input and kernels would result in an amplitude scaled version of the expected convolution, meaning that the scaling factor would propagate to all subsequent layers. This problem can be avoided simply by scaling the amplitude of the resized kernels by s_l^2. That is, given a pre-trained kernel matrix $W_{l,c,1}(x,y)$ that represents the weights of a convolution kernel c from layer l, the corresponding weights to be used in the target network are $W_{l,c,s_l}(x,y) = s_l^2 W_{l,c,1}(sx,sy)$ so that the amplitude gain caused by the convolution operation cancels out.

Moreover, associated to each convolution operation is a bias component $b_{l,c,1}$ that need not be scaled since it has a constant value. This resizing procedure should be carried out for every channel c in every convolutional layer l as described in Equations (7.5) and (7.6).

$$W_{l,c,s_l}(x,y) = s_l^2 W_{l,c,1}(s_l x, s_l y) \tag{7.5}$$

$$b_{l,c,s_l}(x,y) = b_{l,c,1}(s_l x, s_l y) \tag{7.6}$$

Kernels in our experiments were spatially upscaled using bilinear interpolation. Although other interpolation methods were tested, *pre-train* kernel resolutions were too small to benefit from higher order interpolation such as bicubic.

Fully-connected interface

Again, special attention should be paid to the interface between convolutional and fully-connected layers.

According to the usual interpretation described in 7.1.2, the output of a convolutional layer must be vectorized before serving as input to a fully-connected layer, implying a loss of its explicit spatial representation. However, since the incoming feature-maps do contain intra-channel correlation, such information is still present and it is captured by the weights of the fully-connected layer.

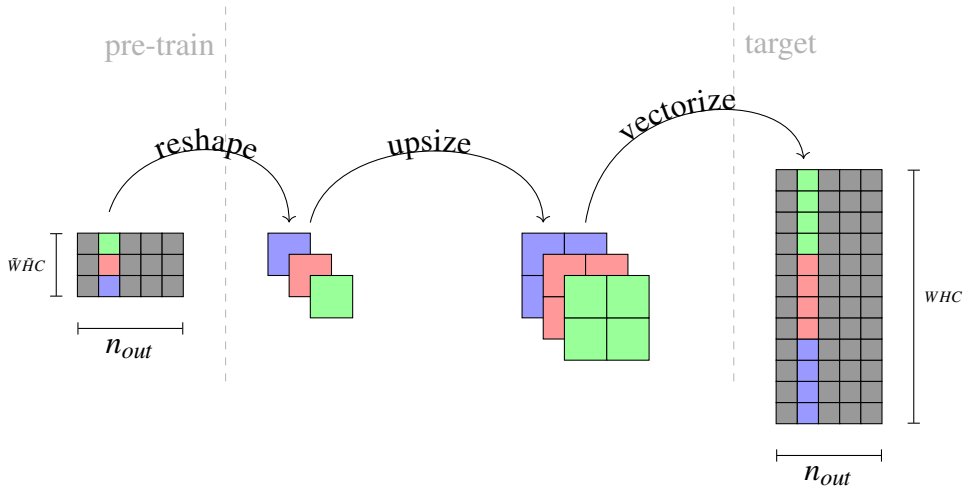


Fig. 7.3 Rescaling weights in fully-connected layer back to *target*'s dimensions. Each column in the fully-connected weight matrix is reshaped to match the *pre-train* feature-map dimensions. Rescaling is applied in the same fashion as regular convolutional kernels and weights are then vectorized to the *target*'s new weight matrix.

In order to exploit this correlation and be able to apply our method, the fully connected layer shall be reinterpreted as a convolutional layer. In other words, each column in the weight matrix must first be reshaped into a third-order tensor $\mathscr{X} \in \mathbb{R}^{C \times \tilde{H} \times \tilde{W}}$ of the same dimensions as the original incoming feature-maps so that we can apply the same resizing rule defined in Equations 7.5 and 7.6. This will produce a new tensor $\mathscr{Y} \in \mathbb{R}^{C \times H \times W}$ that must be then vectorized into the new

weight matrix whose dimensions are consistent with the *target* network. Figure 7.1 illustrates the overall training process.

Finally, weights from successive fully-connected layers must simply be copied to the target model.

7.2 Preliminary Experiments

In order to assess the proposed approach, we must first estimate upper and lower bounds in terms of accuracy and training times set by the *target* network and its *pre-train* counterpart. To do this we use as baseline to our investigation the *fast* variant of 2013 ImageNet localization winner OverFeat [89].

Network: Overfeat-fast

The original OverFeat-*fast* contains five convolutional layers followed by three fully-connected ones that classify 231×231 RGB images among the 1000 classes defined by the ImageNet dataset. By following the steps set in subsection 7.1.2 we generate a *pre-train* model of input resolution 147×147 and do not apply padding at the last convolutional layer. This modified version of the network is described in Table 7.2 using a layout similar to the original article to facilitate comparison. Entries in the table are kept the same except for the convolution input size where we chose to represent the effective convolution input size, accounting for possible paddings and cropping operations.

Network initialization

During our experiments we use an Nvidia Tesla K80 GPU to train and test both networks with ImageNet 2012 CLS-LOC training and validation datasets. We use mini-batches of 128 images and 10k mini-batches per epoch. We use an initial learning rate of 1×10^{-2} and lower it to 5×10^{-3}, 1×10^{-3}, 5×10^{-4}, and 1×10^{-4} at the end of epochs 18, 29, 43, and 52 respectively, until epoch 65 when training is halted. A weight decay of 1×10^{-4} is also applied until the end of epoch 29 and a momentum of 0.9 is used during the entire training. For both networks, weights in

each layer are initialized uniformly at random in the interval $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$, where n is the layer's number of weights.

Reference curves

For the two networks we obtain the train and test accuracies with respect to the number of training epochs and training hours, seen in Figures 7.4 and 7.5. Representing accuracy in terms of epochs allows one to measure how fast the network is learning as data is presented to it, while representation in terms of training time reflects the variable to be optimized.

We notice in Figure 7.4 that test accuracy stops increasing a few epochs after the last change in learning rate. For this reason, we consider both networks to have been fully trained at the end of 55 epochs resulting in best test accuracies of 59.25% (epoch 55) for the *target* network and 55.55% (epoch 53) for the *pre-train* network. We also observe from Figure 7.4 that, during the first epochs, both test and training accuracies follow the same pattern for the two networks, which suggests that information being learnt by the models is both generalizable and adequate to be represented by the smaller, faster network.

On the other hand, Figure 7.5 highlights the effect of using spatially smaller kernels on training time. OverFeat-*fast* took 269.1 hours to perform 55 epochs of training while the *pre-train* network only took 106.8 hours to train on the same amount of data. This reflects a reduction in training time by a factor of 2.51, which largely agrees with the upper-bounds set by Equation 7.3 and the values of s_l^2 in Table 7.1 for the $3 \times 3\,(2.25)$, $5 \times 5\,(2.77)$ and $11 \times 11\,(2.49)$ kernel resolutions found in the original architecture.

Layer	1	2	3	4	5	6	7	8
Stage	conv+max	conv+max	conv	conv	conv+max	full	full	full
Kernel Size (Target)	11x11	5x5	3x3	3x3	3x3	-	-	-
Kernel Size (Pre-train)	**7x7**	**3x3**	**2x2**	**2x2**	**2x2**	-	-	-
Conv. Stride	4x4	1x1	1x1	1x1	1x1	-	-	-
Pooling Size	2x2	2x2	-	-	2x2	-	-	-
Pooling Stride	2x2	2x2	-	-	2x2	-	-	-
Conv. Input Size (Target)	231x231	28x28	14x14	14x14	14x14	6x6	1x1	1x1
Conv. Input Size (Pre-train)	**147x147**	**18x18**	**9x9**	**10x10**	**9x9**	**4x4**	1x1	1x1

Table 7.2 Architecture description of Pre-train network based on Overfeat-*fast*. Values in bold indicate differences with respect to original model.

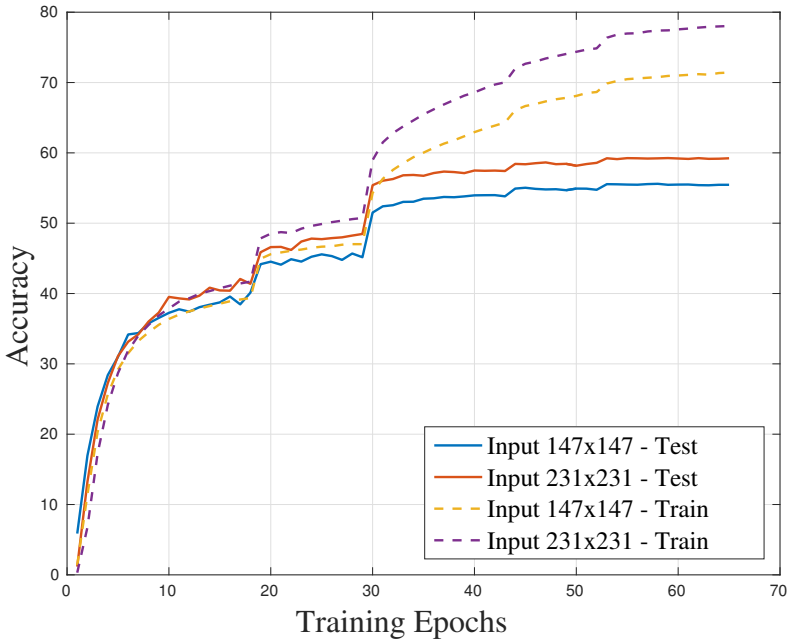


Fig. 7.4 Accuracy as function of epochs obtained using both original OverFeat-*fast* of input resolution 231×231 and its *pre-train* counterpart having 147×147 input resolution.

Input Size	Training Memory	Num. Param.	Best Accuracy (Epoch)	Training Time
231×231	3.88 GiB	130M	59.25%(55)	269.1 h
147×147	2.03 GiB	67M	55.55%(53)	106.8 h

Table 7.3 Effect of resizing kernels on storage requirements, accuracy and training time.

7.3 Experiments on Pre-training

In this Section we evaluate the effects of rescaling the *pre-train* network at different points in time. Our goal is to maximize the number of epochs trained using the smaller network in order to reduce the overall time necessary for training the network.

7.3.1 Resize-and-Continue Scheduled Training

Ideally, one would like to be able to fully train a smaller network, upsize its kernels and immediately obtain the test accuracy of the target network. However, as seen in Figure 7.4, decrease in learning rate and removal of weight decay lead to increase in overfitting, which in turn imposes some constraints to this straightforward approach.

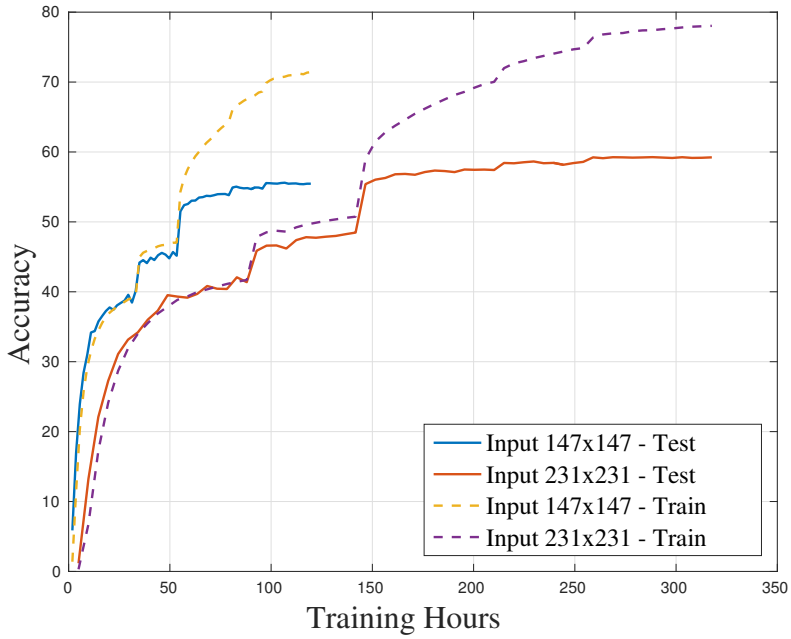


Fig. 7.5 Accuracy as function time obtained using both original OverFeat-*fast* of input resolution 231×231 and its *pre-train* counterpart having 147×147 input resolution.

In this experiment we evaluate the effect of upscaling kernels at different epochs and continuing the scheduled training rule. Since changes in the learning rules had a clear effect on accuracy, we focus on resizing the network before and after these changes. Accuracy curves for each starting epoch are reported in Figure 7.6 including threshold lines for the accuracies obtained by the two baseline networks described in Section 7.2. Although we still consider a 55 epoch training schedule, the process is carried out until epoch 65 in order to verify possible gains due to continuing training.

Curves in Figure 7.6 reveal some interesting behaviour. Each resized model shows a lower starting accuracy when compared to the 147×147 input network test curve. This pattern is expected since interpolation will give an imperfect estimate of the desired kernels. On the other hand, the fact that accuracy does not drop too much indicates that knowledge can, at least partially, be transferred using this method.

The same figure also shows a saturation effect. Networks resized at early stages (epochs 17-20) are able to achieve levels of accuracy similar to the 231×231 input network, while networks resized at late stages (epochs 51-54) can only achieve accuracies below the 147×147 input network threshold. Intermediate values of accuracies were obtained when upsizing the *pre-train* network at epochs from 28

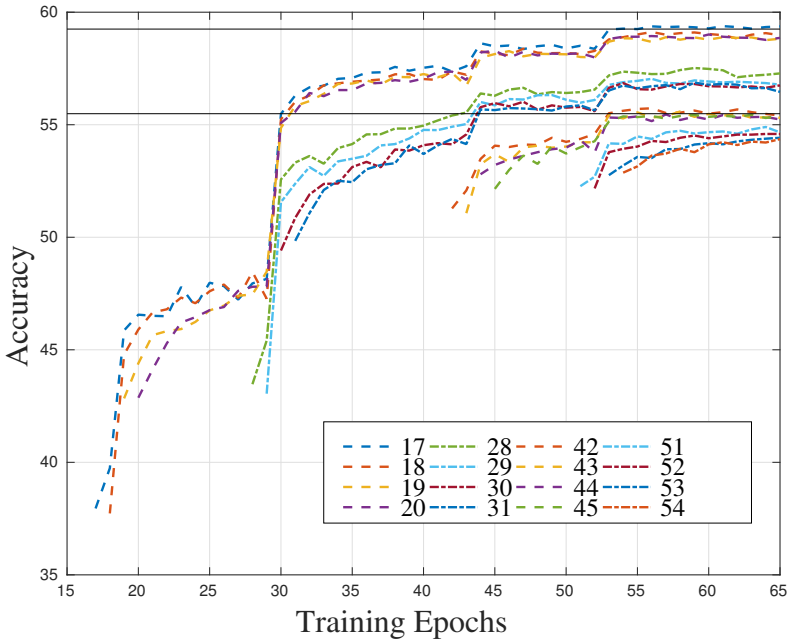


Fig. 7.6 Effects of rescaling kernels at different epochs. Lower and upper horizontal lines define the maximum accuracies obtained with *pre-train* and *target* networks, respectively.

to 31, while accuracies close to the one obtained with the 147×147 input network baseline were obtained by upsizing the pre-train networks at epochs 42-45. Restart training in the vicinity of the last change in learning rate resulted in test accuracies below the *pre-train* network baseline threshold.

Table 7.4 summarizes training times and accuracies for those networks that were able to closely approximate the final accuracy of the *target* network. From this experiment we notice that resizing the *pre-train* network at Epoch 17 produced the same accuracy as the target network even though it takes 49.1 less hours to finish training, a relative gain of 18.25% in training time. These results show the necessity of upscaling early during training in order to achieve the maximum *target's* accuracy.

7.3.2 Resize-and-Continue with Extra Training

As mentioned previously, upscaling weights will produce imprecise kernels that cannot immediately represent the high-frequency details obtained when training with

Network	Best Accuracy (Epoch)	Total Training Time
Pre-train (Input 147×147)	55.55%(53)	106.8 h
Target (Input 231×231)	59.25%(55)	269.1 h
Resized at Epoch 17	**59.25%**(54)	**220.0 h**
Resized at Epoch 18	59.01%(55)	217.0 h
Resized at Epoch 19	58.84%(55)	213.9 h
Resized at Epoch 20	58.91%(54)	210.9 h

Table 7.4 Final accuracy and training times for resized networks after a total of 55 epochs. Lower and upper bound accuracies are set by *pre-train* and *target* networks, respectively.

larger kernels. It is expected that during training these resized kernels will adapt to the new resolution and eventually display the capacity of original target network.

It can be observed in Figure 7.6 that when resizing from epochs 17, 28, and 42, the subsequent epoch still shows relevant increase in accuracy, which does not happen at the same epochs for the baseline networks since, at those points, test accuracy plateaus, raising the need to change learning rate. From this observation we consider maintaining the same learning rule after resizing the networks until there is a drop in test accuracy, from which point on we continue with the predefined learning schedule.

Again we try to maximize the number of epochs run using the *pre-train* network, so we resize and continue the new training procedure at the end of epochs 18, 29 and 43 since these starting points achieved accuracies above the 147×147 input network threshold during the previous experiment. Effects of continuing training using current learning rules for an extra number of epochs are reported in Figures 7.7 and 7.8 along with the curves produced in the previous experiment for the same restarting points.

Accuracies and times for both pre-training approaches are reported in Table 7.5. For each starting point, it can be seen that training for a number of extra epochs does increase the final accuracy. However, this extra training comes at the cost of slowing the overall training procedure.

Moreover, we observe that continuing training from Epoch 19 for 5 extra epochs resulted in a test accuracy slightly above the upper-bound defined by the target network in Section 7.2. Although the difference is too small to be considered as an

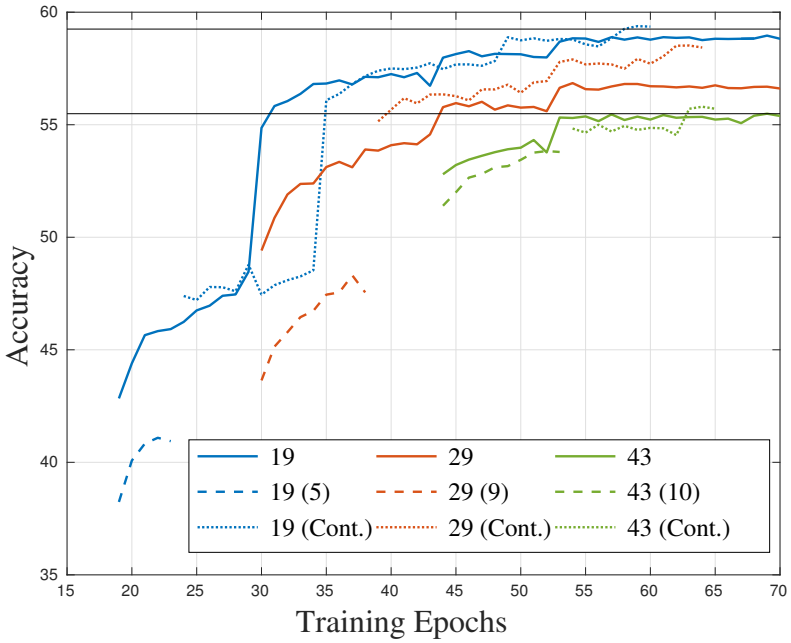


Fig. 7.7 Accuracy as a function of epochs when training is allowed to continue using current learning rules for a few extra epochs. Learning rule is updated as soon as there is a drop in test accuracy.

Network	Extra Epochs	Accuracy (Epoch)	Training Time
Resized at Epoch 19 (Continued)	5	59.36% (59)	238.4 h
Resized at Epoch 30 (Continued)	9	58.52% (64)	224.4 h
Resized at Epoch 43 (Continued)	10	55.80% (64)	186.6 h

Table 7.5 Best accuracy and total training times for resized networks with extra training.

actual improvement (0.11%) it does prove that the upper-bound is achievable using the proposed method while avoiding 30.7 hours of training (11.41% with respect to the original time).

7.3.3 Residual Networks

To prove that our approach can be used on different architectures along with other optimization techniques, we apply our method to the more recent Residual Network [92] architecture having 34 layers. As suggested by previous results, we resize

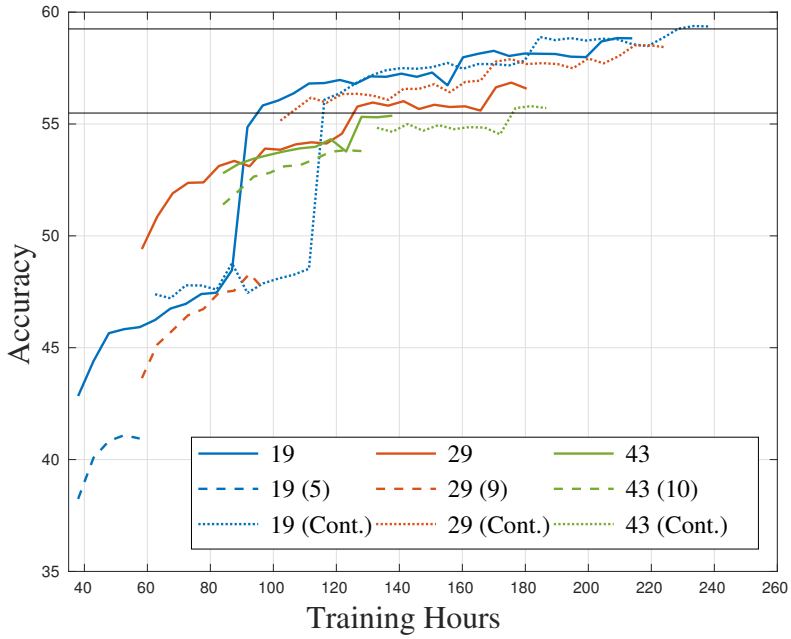


Fig. 7.8 Accuracy as a function of time when training is allowed to continue using current learning rules for a few extra epochs. Learning rule is updated as soon as there is a drop in test accuracy.

the *pre-train* network one and two epochs before changing learning rate and verify possible gains in training times.

For this experiment, training was performed for 90 epochs using mini-batches of 128, weight decay of 1×10^{-4}, and momentum equal to 0.9. Learning rate is initially set to 1×10^{-1} and it is reduced to 1×10^{-2} and 1×10^{-3} before epochs 31 and 61. All experiments were run on a single NVIDIA Titan-X using the CuDNN library for FFT based convolutions. Original images crop resolutions were 224×224 for the *target* network and 160×160 for *pre-train*.

As seen in Figures 7.9 and 7.10, resizing at early epochs (29 and 30) allowed the networks to achieve the expected maximum accuracy, while resizing at late epochs (59 and 60) prevented them from doing so. Moreover, when compared to the original architecture, resizing the *pre-train* ResNet at epoch 29 allowed it avoid 33.7 hours (18.80%) of training and gave in slightly better accuracy. A summary of these results is reported in Table 7.6.

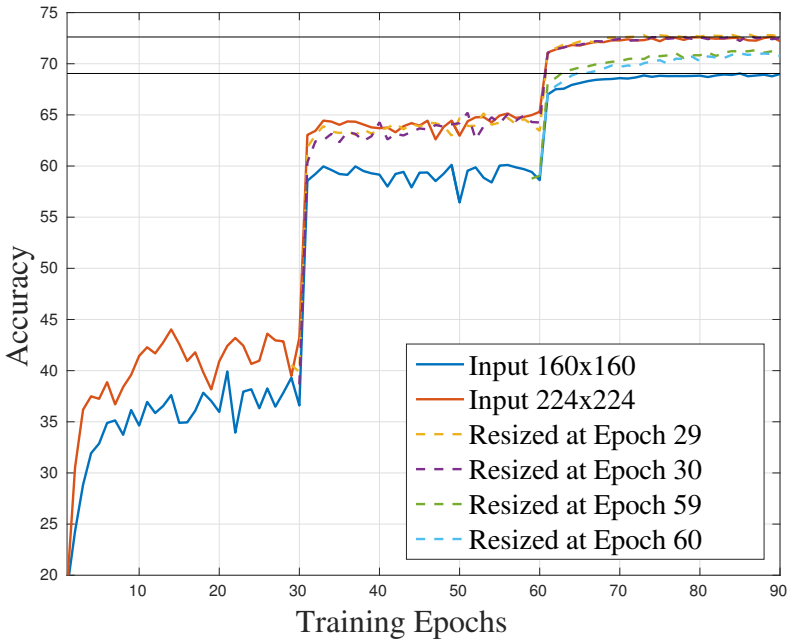


Fig. 7.9 Accuracy curves obtained using ResNet-34 as a function of epochs. Lower and upper horizontal lines define the best accuracies obtained for the new baseline networks.

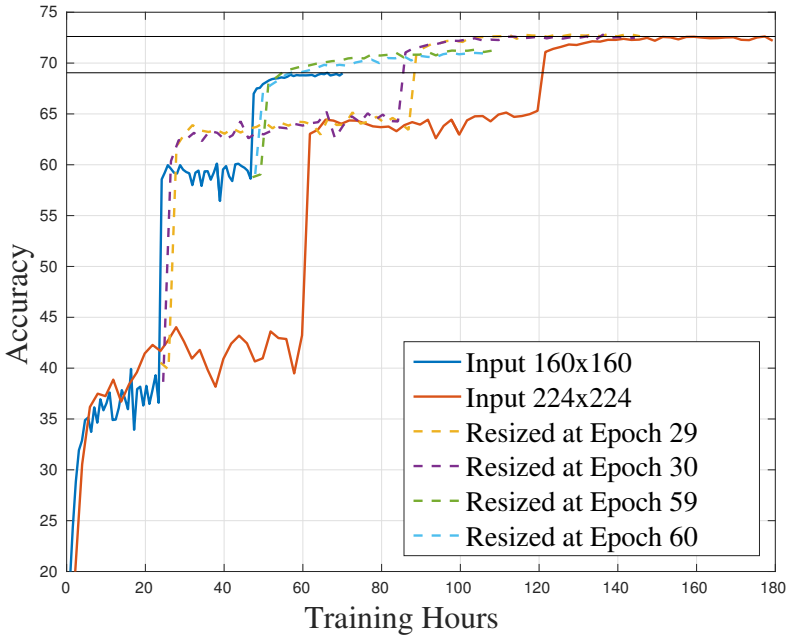


Fig. 7.10 Accuracy curves obtained using ResNet-34 as a function of time. Lower and upper horizontal lines define the best accuracies obtained for the new baseline networks.

Network	Accuracy (Epoch)	Training Time
Pre-train (160×160)	69.05% (85)	70.09 h
Target (224×224)	72.61% (89)	179.33 h
Resized at Epoch 29	**72.91%** (86)	**145.60 h**
Resized at Epoch 30	72.79% (86)	144.32 h
Resized at Epoch 59	71.36% (90)	110.02 h
Resized at Epoch 60	71.01% (85)	107.82 h

Table 7.6 Best accuracy and training times for ResNet-34. Training is reduced by 33.7 hours when upscaling two epochs before changing learning rate.

7.4 Result Analysis

In this work, we have presented a fast way of training CNN that exploits the spatial scaling property of convolutions. Ideally the scaling property would allow a *target* model to be trained from a fully trained *pre-train* network. In practice, however, we have observed that there is an intrinsic saturation process that prevents such näive implementation from succeeding. The longer the *pre-train* network is trained the less likely it is to achieve the performance of the *target* network. Although further investigation is required, to the best of our knowledge this happens because, as the *pre-train* network is trained, the learnt set of weights moves towards a deep local minimum making it difficult to locally find better weights with lower learning-rates.

However, we observe that this effect is mitigated at early stages of learning where testing and training accuracies are similar for both networks. This leads to the conclusion that both networks are learning information that can be generalized, and that can be effectively exploited at both kernel resolutions. This allowed us to use the proposed approach as a pre-training technique where, by resizing the network a couple of epochs before the first scheduled change in learning rate, we were able to obtain the expected *target* accuracy for both OverFeat and ResNet architectures while avoiding 49.1 hours (18.25%) and 33.7 hours (18.80%) of training, respectively.

Chapter 8

Conclusion

In this thesis we have explored two aspects of visual features. We started our work by exploiting the fact that some computer vision tasks such as object recognition and robotic navigation share similar underlying requirements by proposing the use of MPEG CDVS for solving the problem of Visual Simultaneous Navigation and Mapping in the context of indoor monocular navigation. In doing so, we evaluated the standard's various modes of compression in terms of extraction and matching times, distinctiveness over different floorings and storage requirements. We also developed a probabilistic framework for detecting loops based on the standard's suggested similarity metric called "local score". Finally, we proved the effectiveness of our approach for indoor environments by showing superior results against a laser-scanner SLAM setup.

During the second part of this work, we analyzed the long training periods required for learning visual features and proposed a novel, fast method for training Convolutional Neural Networks. In our approach we begin the training process using a *pre-train* network having lower-resolution kernels and input images, and then refine the results at the desired resolution by exploiting the spatial scaling property of convolutions. We applied our method to two ImageNet LSVRC winners and showed a reduction in training time of nearly 20%.

8.1 Future work

Naturally, while performing experiments and evaluating results, new ideas come to mind that could lead to interesting research topics. The following are some future work which should be subject to further investigation.

- *Use MPEG CDVS local score for estimating measurement noise:* During graph optimization using LAGO, we considered all measurements to have equal error estimate. It would be interesting to use feature matching score information to generate more accurate error models.
- *Use MPEG CDVS global descriptor for place recognition:* In this work we did not use MPEG CDVS global descriptor because it does not incorporate feature coordinates. However, the global descriptor could be useful for recognizing different flooring types, which in turn could be used for selecting the most efficient compression mode for navigation.
- *Different optimization methods for CNNs:* In this work we have used the standard Stochastic Gradient Descent approach for training CNNs. It would be interesting to investigate if higher order approximations, such as Newton's method, or adaptive learning rates methods, such as Adagrad, could help avoid accuracy saturation cause by late network upscaling.
- *Use of Adversarial Training:* It has been shown that a technique called Adversarial Training can help networks explore different regions in the weight-space [111]. This is done by backpropagating the gradient of the classification function back to the input image considering a wrongfully associated class. This approach generates wrong, adversarial sample that the network must correct. It would be interesting to try this training approach also to verify if saturation can be avoided.

References

[1] ISO/IEC JTC 1/SC 29/WG 11 (MPEG). Information technology – multimedia content description interface – part 13: Compact descriptors for visual search. Final Draft of International Standard 15938-13, ISO/IEC, Geneva, Switzerland, 2014.

[2] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[3] M Srinivasan, Shaowu Zhang, M Lehrer, and T Collett. Honeybee navigation en route to the goal: visual flight control and odometry. *Journal of Experimental Biology*, 199(1):237–244, 1996.

[4] Larry Matthies and STEVENA Shafer. Error modeling in stereo navigation. *IEEE Journal on Robotics and Automation*, 3(3):239–248, 1987.

[5] Larry Henry Matthies. *Dynamic Stereo Vision*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1989. AAI9023429.

[6] Annalisa Milella and Roland Siegwart. Stereo-based ego-motion estimation using pixel tracking and iterative closest point. In *Fourth IEEE International Conference on Computer Vision Systems (ICVS'06)*, pages 21–21. IEEE, 2006.

[7] Andrew Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3946–3952. IEEE, 2008.

[8] Yang Cheng, Mark Maimone, and Larry Matthies. Visual odometry on the mars exploration rovers. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 1, pages 903–910. IEEE, 2005.

[9] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980.

[10] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[11] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23(1):3–20, 2006.

[12] Jean-Philippe Tardif, Yanis Pavlidis, and Kostas Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2531–2538. IEEE, 2008.

[13] Peter Corke, Dennis Strelow, and Sanjiv Singh. Omnidirectional visual odometry for a planetary rover. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 4, pages 4007–4012. IEEE, 2004.

[14] Etienne Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser, and Patrick Sayd. Real time localization and 3d reconstruction. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 363–370. IEEE, 2006.

[15] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *Robotics & Automation Magazine, IEEE*, 19(2):78–90, 2012.

[16] Davide Scaramuzza and Friedrich Fraundorfer. Visual Odometry [Tutorial]. *IEEE Robotics & Automation Magazine*, 18(4):80–92, December 2011.

[17] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[18] Bojian Liang and Nick Pears. Visual navigation using planar homographies. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 205–210. IEEE, 2002.

[19] Janne Heikkila and Olli Silvén. A four-step camera calibration procedure with implicit image correction. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 1106–1112. IEEE, 1997.

[20] Timothy A Clarke and John G Fryer. The development of camera calibration methods and models. *The Photogrammetric Record*, 16(91):51–66, 1998.

[21] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[22] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, Sept 1987.

[23] Arnaud Doucet, Nando De Freitas, Kevin Murphy, and Stuart Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 176–183. Morgan Kaufmann Publishers Inc., 2000.

[24] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986.

[25] Giorgio Grisetti, Cyrill Stachniss, Slawomir Grzonka, and Wolfram Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems*, pages 27–30, 2007.

[26] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.

[27] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical Report GT-RIM-CP&R-2012-002, GT RIM, Sept 2012.

[28] Robotics Research Group Politecnico di Torino. LAGO: Linear approximation for graph optimization. https://github.com/rrg-polito/lago, 2000–2004.

[29] David Marr and Tomaso Poggio. Cooperative computation of stereo disparity. In *From the Retina to the Neocortex*, pages 239–243. Springer, 1976.

[30] Shimon Ullman. *The interpretation of visual motion.* Massachusetts Inst of Technology Pr, 1979.

[31] Paul J Besl and Ramesh C Jain. Three-dimensional object recognition. *ACM Computing Surveys (CSUR)*, 17(1):75–145, 1985.

[32] Roland T Chin and Charles R Dyer. Model-based recognition in robot vision. *ACM Computing Surveys (CSUR)*, 18(1):67–108, 1986.

[33] Michael J Swain and Dana H Ballard. Color indexing. *International journal of computer vision*, 7(1):11–32, 1991.

[34] Bernt Schiele and James L Crowley. Object recognition using multidimensional receptive field histograms. In *European Conference on Computer Vision*, pages 610–619. Springer, 1996.

[35] Cordelia Schmid and Roger Mohr. Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–534, 1997.

[36] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[37] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[38] Matthew Brown and David G Lowe. Automatic panoramic image stitching using invariant features. *International journal of computer vision*, 74(1):59–73, 2007.

[39] Chieh-Chih Wang and Ko-Chih Wang. Hand posture recognition using adaboost with sift for human robot interaction. In *Recent progress in robotics: viable robotic service to human*, pages 317–329. Springer, 2007.

[40] Robert Sim, Pantelis Elinas, and Matt Griffin. Vision-based slam using the rao-blackwellised particle filter. In *In IJCAI Workshop on Reasoning with Uncertainty in Robotics*, 2005.

[41] Andrew Witkin. Scale-space filtering: A new approach to multi-scale description. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'84.*, volume 9, pages 150–153. IEEE, 1984.

[42] Jan J Koenderink. The structure of images. *Biological cybernetics*, 50(5):363–370, 1984.

[43] Tony Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of applied statistics*, 21(1-2):225–270, 1994.

[44] M. Brown and D. Lowe. Invariant features from interest point groups. In *Proceedings of the British Machine Vision Conference*, pages 23.1–23.10. BMVA Press, 2002. doi:10.5244/C.16.23.

[45] Krystian Mikolajczyk. *Detection of local features invariant to affine transformations*. PhD thesis, Citeseer, 2011.

[46] Dana H Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.

[47] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[48] Peter J Rousseeuw. Least median of squares regression. *Journal of the American statistical association*, 79(388):871–880, 1984.

[49] Bernd Girod, Vijay Chandrasekhar, David M Chen, Ngai-Man Cheung, Radek Grzeszczuk, Yuriy Reznik, Gabriel Takacs, Sam S Tsai, and Ramakrishna Vedantham. Mobile visual search. *IEEE signal processing magazine*, 28(4):61–76, 2011.

[50] Vijay Chandrasekhar, Gabriel Takacs, David M Chen, Sam S Tsai, Yuriy Reznik, Radek Grzeszczuk, and Bernd Girod. Compressed histogram of gradients: A low-bitrate descriptor. *International journal of computer vision*, 96(3):384–399, 2012.

[51] Ling-Yu Duan, Vijay Chandrasekhar, Jie Chen, Jie Lin, Zhe Wang, Tiejun Huang, Bernd Girod, and Wen Gao. Overview of the mpeg-cdvs standard. *IEEE Transactions on Image Processing*, 25(1):179–194, 2016.

[52] Kai Cordes, Bodo Rosenhahn, and Jörn Ostermann. Localization accuracy of interest point detectors with different scale space representations. In *Advanced Video and Signal Based Surveillance (AVSS), 2014 11th IEEE International Conference on*, pages 247–252. IEEE, 2014.

[53] Gianluca Francini, Skjalg Lepsøy, and Massimo Balestri. Selection of local features for visual search. *Signal Processing: Image Communication*, 28(4):311–322, 2013.

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

[55] Sam S Tsai, David Chen, Gabriel Takacs, Vijay Chandrasekhar, Mina Makar, Radek Grzeszczuk, and Bernd Girod. Improved coding for image feature location information. In *SPIE Optical Engineering+ Applications*, pages 84991E–84991E. International Society for Optics and Photonics, 2012.

[56] Florent Perronnin, Yan Liu, Jorge Sánchez, and Hervé Poirier. Large-scale image retrieval with compressed fisher vectors. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3384–3391. IEEE, 2010.

[57] Jie Lin, Ling-Yu Duan, Yaping Huang, Siwei Luo, Tiejun Huang, and Wen Gao. Rate-adaptive compact fisher codes for mobile visual search. *IEEE Signal Processing Letters*, 21(2):195–198, 2014.

[58] Skjalg Lepsoy, Gianluca Francini, Giovanni Cordara, and Pedro PB de Gusmao. Statistical modelling of outliers for fast visual search. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

[59] Adam Schmidt, Marek Kraft, and Andrzej Kasiński. An evaluation of image feature detectors and descriptors for robot navigation. In *International Conference on Computer Vision and Graphics*, pages 251–259. Springer, 2010.

[60] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, 2005.

[61] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.

[62] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[63] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[64] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[65] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[66] Motilal Agrawal, Kurt Konolige, and Morten Rufus Blas. Censure: Center surround extremas for realtime feature detection and matching. In *European Conference on Computer Vision*, pages 102–115. Springer, 2008.

[67] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[68] Stefan Leutenegger, Margarita Chli, and Roland Yves Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[69] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, pages 1–22, 2004.

[70] Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477. IEEE, 2003.

[71] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *European conference on computer vision*, pages 143–156. Springer, 2010.

[72] Tommi S Jaakkola, David Haussler, et al. Exploiting generative models in discriminative classifiers. *Advances in neural information processing systems*, pages 487–493, 1999.

[73] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[74] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[75] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[76] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[77] Frank Rosenblatt. The perceptron a perceiving and recognizing automaton. Technical report, tech. rep., Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957. 2, 1957.

[78] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[79] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[81] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[82] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[83] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[84] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[85] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[86] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[87] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

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

[89] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *International Conference on Learning Representations*. CBLS, apr 2014.

[90] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[91] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[92] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2016.

[93] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[94] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[95] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[96] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[97] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[98] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Department of Computer Science, 04 2009.

[99] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[100] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[101] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. In *International Conference on Learning Representations*. CBLS, April 2014.

[102] Shmuel Winograd. *Arithmetic complexity of computations*, volume 33. Siam, 1980.

[103] Andrew Lavin and Scott Gray. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4013–4021, 2016.

[104] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[105] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*. CBLS, 2016.

[106] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[107] ISO/IEC JTC 1/SC 29/WG 11 (MPEG). Test model 13: Compact descriptors for visual search, 2015.

[108] Hui Wang Hui Wang, Kui Yuan Kui Yuan, Wei Zou Wei Zou, and Qingrui Zhou Qingrui Zhou. Visual odometry based on locally planar ground assumption. *2005 IEEE International Conference on Information Acquisition*, pages 59–64, 2005.

[109] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007.

[110] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[111] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.