

Numéro d'ordre : 2011ISAL0042 Année 2011

Institut National des Sciences Appliquées de Lyon
Laboratoire d'InfoRmatique en Image et Systèmes d'information
École Doctorale Informatique et Mathématiques de Lyon

Thèse de l'Université de Lyon

Présentée en vue d'obtenir le grade de Docteur,
spécialité Informatique

par

Jérôme Revaud

Contributions to a Fast and Robust Object Recognition in Images

---

Thèse soutenue le 27 mai 2011 devant le jury composé de :

M.	Patrick Gros	Directeur de recherche, INRIA Rennes	Rapporteur
M.	Frédéric Jurie	Professeur, Université de Caen	Rapporteur
M.	Vincent Lepetit	Senior Researcher, EPFL	Examinateur
M.	Jean Ponce	Professeur, INRIA Paris	Examinateur
M.	Atilla Baskurt	Professeur, INSA Lyon	Directeur
M.	Yasuo Ariki	Professeur, Université de Kobe	Co-directeur
M.	Guillaume Lavoué	Maître de Conférences, INSA Lyon	Co-encadrant

Laboratoire d'InfoRmatique en Image et Systèmes d'information
UMR 5205 CNRS - INSA de Lyon - Bât. Jules Verne
69621 Villeurbanne cedex - France
Tel: +33 (0)4 72 43 60 97 - Fax: +33 (0)4 72 43 71 17

Abstract

Object recognition in images is a growing field. Since several years, the emergence of invariant interest points such as SIFT [Low01] has enabled rapid and effective systems for the recognition of instances of specific objects as well as classes of objects (e.g. using the bag-of-words model). However, our experiments on the recognition of specific object instances have shown that under realistic conditions of use (e.g. the presence of various noises such as blur, poor lighting, low resolution cameras, etc.) progress remains to be done in terms of recall: despite the low rate of false positives, too few actual instances are detected regardless of the system (RANSAC, votes / Hough ...). In this presentation, we first present a contribution to overcome this problem of robustness for the recognition of object instances, then we straightly extend this contribution to the detection and localization of classes of objects.

Initially, we have developed a method inspired by graph matching to address the problem of fast recognition of instances of specific objects in noisy conditions. This method allows to easily combine any types of local features (eg contours, textures ...) less affected by noise than keypoints, while bypassing the normalization problem and without penalizing too much the detection speed. In this approach, the detection system consists of a set of cascades of micro-classifiers trained beforehand. Each micro-classifier is responsible for comparing the test image locally and from a certain point of view (e.g. as contours, or textures ...) to the same area in the model image. The cascades of micro-classifiers can therefore recognize different parts of the model in a robust manner (only the most effective cascades are selected during learning). Finally, a probabilistic model that combines those partial detections infers global detections. Unlike other methods based on a global rigid transformation, our approach is robust to complex deformations such as those due to perspective or those non-rigid inherent to the model itself (e.g. a face, a flexible magazine).

Our experiments on several datasets have showed the relevance of our approach. It is overall slightly less robust to occlusion than existing approaches, but it produces better performances in noisy conditions.

In a second step, we have developed an approach for detecting classes of objects in the same spirit as the bag-of-visual-words model. For this we use our cascaded micro-classifiers to recognize visual words more distinctive than the classical words simply

based on visual dictionaries (like Csurka et al. [CDF*04] or Lazebnik et al. [LSP05]). Training is divided into two parts: First, we generate cascades of micro-classifiers for recognizing local parts of the model pictures and then in a second step, we use a classifier to model the decision boundary between images of class and those of non-class. This classifier bases its decision on a vector counting the outputs of each binary micro-classifier. This vector is extremely sparse and a simple classifier such as Real-Adaboost manages to produce a system with good performances (this type of classifier is similar in fact to the subgraph membership kernel). In particular, we show that the association of classical visual words (from keypoints patches) and our disctinctive words results in a significant improvement. The computation time is generally quite low, given the structure of the cascades that minimizes the detection time and the form of the classifier is extremely fast to evaluate.

Keywords: Specific object recognition, class object recognition, graph matching, cascades, optimization, mobile robotic.

Contents

List of Figures

List of Tables

Liste des Algorithmes

Chapter 1

Introduction

1.1 A Few Preliminary Words

Automatic object recognition in unconstrained conditions is a challenging task with many potential applications. Despite its seeming simplicity, creating a system capable of understanding the surrounding world from pictures, like we do us humans, is a difficult problem − although it is probably much easier than creating a full artificial intelligence. More pragmatically, this captivating topic has a large number of practical applications in today's world where images are ubiquitous.

Scientifically speaking, object recognition is a whole research topic in itself. It has always interested researchers in computer science and has been a very active topic since the very beginning of computer science (let's say, at least for the past 40 years), when the available techniques were quite poor (see for instance the paper of Fischler and Elschlager from 1973 [FE73] where pictures are rendered using ASCII characters). In comparison, today's techniques can afford complex computations that are several orders of magnitude larger than the ones performed in those pioneer works, thanks to the permanent increase in hardware power. Still, the perfect system is yet to be invented, although important breakthroughs have recently emerged. To give a simple overview, nowadays it is pretty much feasible to detect humans (pedestrians or faces) even in noisy conditions. On the other hand, detecting any kind of objects in realistic conditions is still a challenge for computer vision. In particular, producing detection systems that are both robust to noise (in the general sense: jpeg noise, occlusion, clutter, etc.) and fast is a challenge at stake.

In this dissertation, we present two contributions to object recognition while keeping in mind those two constraints. In the first contribution, we present a generic system

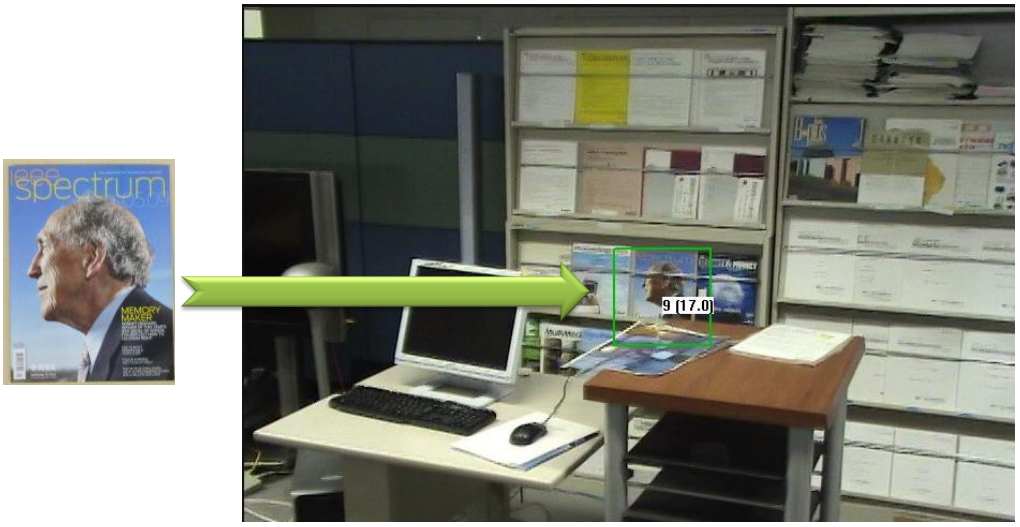


Figure 1.1: An example of recognition. The model object at left, a journal, is recognized and localized by a detection system in the scene picture at right despite some occlusion and a reduced scale.

for the recognition of instances of specific objects (an illustration is shown in Figure 1.1), which is much more robust against realistic noise conditions for mobile robotic than existing methods from the state-of-the-art. In the second contribution, we focus instead on the recognition of classes of objects by re-using parts of the framework of our first contribution, again leading to a system which results in substantial improvement in speed and robustness over existing algorithms.

1.2 Application Field

Contrary to other research fields like mathematics, computer vision and especially object recognition belongs to the field of applied sciences. There is an impressive number of applications directly or indirectly connected to object recognition, from which some examples are illustrated in Figure 1.2. A non-exhaustive list of potential applications includes:

- Robotic vision for:
  - industrial purposes like automatic control of industrial clamps from vision (Figure 1.2, middle column, top row), or automatic counting of elements for non-destructive controls.
  - embedded mobile systems for domestic usage. The purpose for a robot like the ones in the left column of Figure 1.2 is to interact with an indoor environment. It involves different tasks like localization from vision, object recogni-

tion and object pose estimation, face and speech recognition etc. It is crucial to point out that robotic vision in unconstrained environments is especially difficult: it implies that the robot takes decisions in real-time, i.e. requiring a fortiori to detect objects and to understand the scene in real-time, and all of this without making any error. In other words, extreme robustness and detection speed are the key elements of a realistic application. Note that in Japan, an official government program is currently supporting a long-term plan aiming at assisting the elderly with robots.

- Content-Based Image Retrieval (CBIR) systems. Currently, most image search engine (e.g. Google Images) only index images based on the text or legend surrounding them. Because this technique can often be a source of errors, current research moves towards a combination of textual tags, object recognition techniques and propagation of tags between images sharing visual similarities.

- Video surveillance and automatic monitoring of events. This application includes the detection of unusual events as well as their characterization. An example is shown in Figure 1.2 (middle column, bottom row) where an intruder is detected in a parking.

- Augmented reality on smart phones. As the name indicates, the insight in this case is to virtually "augment" the filmed scene by superimposing additional information on it, such as the road and monument names, or the average ranking and critics of a book. Although the final step deals more about information technologies, augmented reality first implies to detect elements in real-time in the filmed scene. An example of automatic landmark detection is shown in top-right corner of Figure 1.2.

- Medical imaging, where the field of applications is vast because of the wide variety of medical image sources (e.g. obtained using magnetic resonance imaging). An example of application involving the automatic recognition of hand bone segments in radiographies is presented in Figure 1.2 (right column, bottom row).

Overall, robustness and/or speed considerations are extremely important for all those mentioned applications. In this dissertation, we mainly focus on the robotic vision application, although other utilizations remain possible as well, so that these two aspects are essential in our contributions.

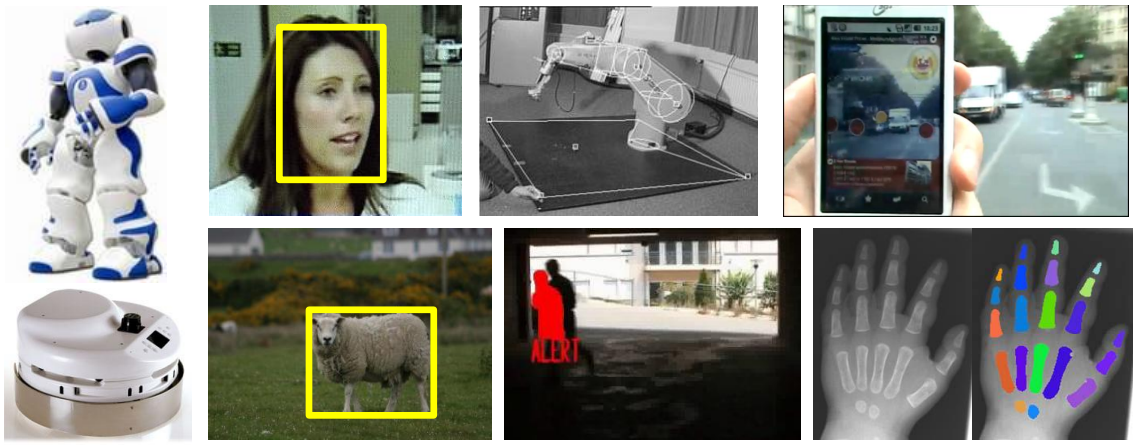


Figure 1.2: Application samples for object recognition. From left to right, top to bottom: mobile robots with elaborate vision processing, face/object detection for content-based image retrieval, visual control of industrial robots, video surveillance (here, intruder detection), augmented reality for mobile phones, medical image processing.

1.3 A Short Definition of Object Recognition Terms

Although the topic may sound intuitive in the ear of the reader, we have to formally define certain terms preliminary to the following of this dissertation.

Object or Class In the formalism of object recognition, an object is defined in its widest sense, i.e., from a specific object (e.g. this book) to a class of objects (e.g. cars, faces). In the first case we talk about *individual* or *specific* objects, while in the second case we talk about *class objects*. We describe this distinction with more details in the paragraph below.

Test/scene image Input image to the recognition system. Unless it is explicitly specified, no particular assumption is made about this image (e.g. each of the model objects can be present or not). In this dissertation, we only consider gray level images defined as matrices of pixels:

$$\mathcal{I} : [1, Tx] \times [1, Ty] \rightarrow [0, 255].$$

Model object An object which is learned by the recognition system from a set of model images in order to be later recognized in test images.

Model Instance A specific exemplar of the model object or model class present in a test image.

Object recognition The task of finding a given model object in a test image, i.e., lo-

calizing every instances of the model object in the test image with a rectangular bounding box. In this dissertation, we put aside the aspect of temporal continuity present in frames when we deal with object recognition in videos. That is, we process all frames independently.

Object detection Although a distinction is sometimes made between *recognition* and *detection*, in this dissertation we consider those two terms to be synonyms.

Localization The task of localizing the position of a model instance, usually in the form of a bounding rectangle (but it may extend up to determining the object pose). As said above, it is a subtask of object recognition.

Classification The task of classifying a test image into one of several pre-defined categories (e.g. sunset, forest, town). Equivalently, if the categories correspond to different model objects or classes, it is the task of deciding if at least one instance of the model object is present in the image. Note that contrary to object recognition, this task does not imply localization and is only applied to classes of objects or classes of backgrounds.

Image Features Set of low-level information extracted from the image. Image pixels are the simplest features (i.e. the lowest level). More complex (and higher-level) features are obtained by rearranging the pixel values according to some pre-defined successions of operations. The next chapter will introduce some frequently used complex features.

Object variations tackled in this dissertation In practice, object recognition has to deal with two kinds of class variations:

- Inter-class variations, between instances of different classes. The more the classes are different in the feature space, the easier it becomes to separate them and classify an instance into the right class.

- Intra-class variations, between instances of the same class. They represent how much instances of the same class can vary with respect to each other. In the case of specific objects, intra-class variations are minor because only caused by noises such as captor noise, movement blur and lighting effects. On the contrary in the case of class objects, intra-class variations are connected to variations of semantic concepts related to the objects. For instance, a face is always composed of two eyes, a mouth etc., but the appearance of each such facial organ varies from one face to another.

As a consequence, a class object recognition system must struggle a lot much to learn the correct class boundaries than a specific object recognition system: because of semantic variations, decision boundaries are much more complex for a class. As a consequence, a lot more images are typically required to train a class object recognition system. Figure (1.3) illustrates those variations for various specific and class objects : as can be seen, the appearance of class objects can largely vary from one instance to another compared to specific objects.

Precisely concerning specific (i.e. individual) object recognition, we choose in this dissertation to consider 3D viewpoint changes and non-rigid object distortions as additional sources of intra-class variations. In fact, we make the choice of not explicitly modeling neither of those variations, that is we consider them as pure noise added on the training instances. All in all, our purpose is to make a generic recognition system robust to a large range of possible disturbances, so that it can bear the unexpected of realistic real-time conditions.

Figure 1.3: Illustration of intra-class variations for the class case (left) and the specific case (right). In the case of specific objects (the stuffed animal and the tea mug), only external variations such as lighting, background or 3D pose affect the appearance of the model object. In the case of classes (the plane and the camera), an additional variation comes from the variety of possible model instances.

1.4 Outlines

This manuscript is organized as follows:

In Chapter 2, we present an overview of the state-of the-art in the field of object recognition with a special focus on specific object detection techniques, as we consider it to be the core of this dissertation.

Then, we present two related contributions to the object recognition framework which both aim at increasing the recognition robustness while maintaining a high detection speed.

Firstly, chapter 3 introduces an approach for specific object recognition. It relies on non-rigid graph matching with a framework designed to enable the integration of different types of local features, contrary to most existing approaches, in order to increase the robustness. Qualitative and quantitative evaluations of this contribution are presented in Chapter 4 on our own dataset for realistic robotic vision and on two other popular datasets. In addition to an in-depth analysis of the detection performance is also included a study of timing performance.

Secondly, we present an extension of the first contribution to the case of class object recognition in Chapter 5. In fact, we use the same feature extraction framework than in the first contribution but we adapt the decision model so as to handle the expected larger intra-class variations of class objects. Again, qualitative and quantitative evaluations of this contribution along with speed considerations are presented in Chapter 6 on single object classes and a popular dataset for image classification.

Finally, Chapter 7 concludes and introduces some perspectives.

Chapter 2

Survey on Object Recognition

Contents

This chapter provides an overview of the current techniques from the state-of-the-art in object recognition for both specific and class object recognition. We begin by

presenting basic concepts related to the feature extraction and description steps. Then, we review existing methods used for specific and class object detection and further examine their machinery with greater details. Finally, we also criticize various aspects of existing methods with respect to our objective in this dissertation of elaborating a fast and robust detection system.

2.1 A Glance at Object Recognition

Even if recognizing an object in any kind of environment is almost immediate and effortless for us, that is still of huge difficulty for computers. After several years of research in the neurocognitive field, what we know so far is that our brain contains several layers of neurons dedicated to different low-level processing of the information coming from the eyes [Sch77]. Those layers contain different neuron types called C1, V1, C2 and V2 which are known to apply some simple fixed preprocessing, such as extracting local edges, the gradient orientations or aggregating those information (in particular, see some detection systems inspired by this cortex organization [KP99, SWB*07]). Afterward those data undergo subsequent processing deeper in the brain. At this very moment, we loose more or less track of what happens, but we can guess that it is complex.

Interestingly enough, object recognition systems roughly follow the same dataflow (see Figure 2.1): in a first step, low-level *image features* are extracted from the images in an automatic way. Examples of low-level features include edges, corners and textures. At this point, not enough information is available to draw any conclusions yet regarding the image content, as each of these features taken individually only owns in the best case a slight correlation with semantic image contents. As a consequence, a more complex decision process, previously trained to distinguish between the model object and clutter, is run in a second step. It relies on a global analysis of all available features and takes a final decision regarding the presence and the location of the object.

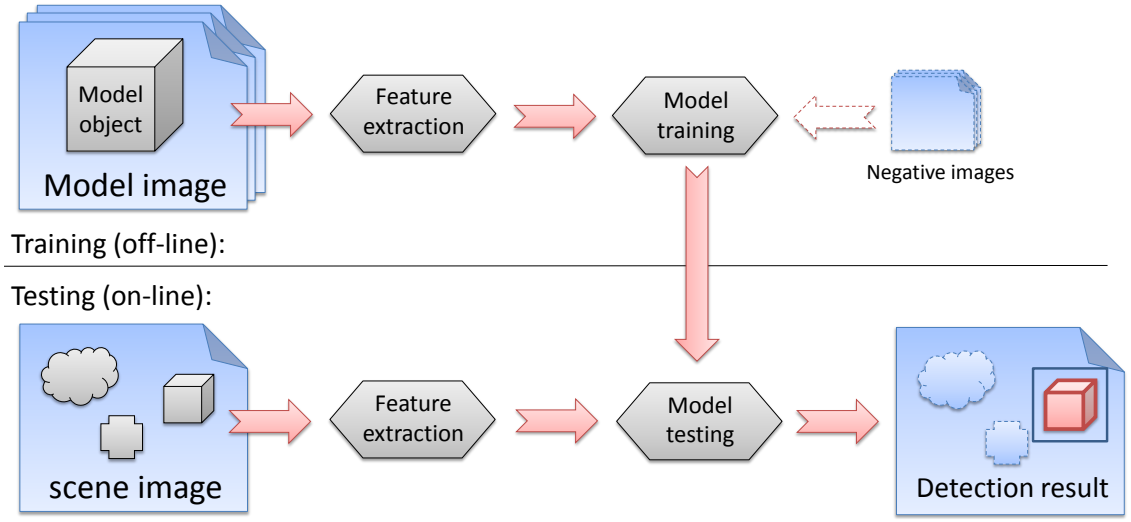


Figure 2.1: General dataflow for object recognition systems.

The very interest of thus decomposing the recognition process in two steps is to simplify the handling of appearance variations. In fact, although the appearance of a same

object may appear consistent over time and environments for us (this illusion comes from the ease with which our brain performs object detection), the situation is completely different for a computer. Small changes in light or viewpoint lead to images in which the same object can appear totally different in terms of image pixels. In order to be usable, a detection scheme thus has to be invariant to the following disturbance sources: noise, illumination, translation, rescaling, rotation and intra-class variations. A two-step decomposition enables an easier sharing of this burden: invariance to illumination, translation, rescaling and rotation are generally handled at the feature level while noise and intra-class variations are dealt with by the decision process.

In the following of this chapter, we begin by presenting most of the popular existing feature detectors and descriptors. Then, we explain how to aggregate these low-level information in order to achieve object detection. As we will see, this implies first to create a model for the object we wish to detect. We firstly dwell on the approaches related to the first contribution of this dissertation, i.e. specific object recognition (Section 2.3), then we also give an overview of existing techniques used for class object recognition (Section 2.4) related to our second contribution.

2.2 Low-level Features

As stated in the previous section, object recognition begins by extracting low-level features from the images as an intermediary step before more complex processing. To put it simply, an image feature is a value computed from the image pixels according to a given formula. The gradient, for instance, is computed as the difference of value between consecutive image pixels. Therefore it is generally said of a feature that it *describes* the image under a certain viewpoint, as it emphasize a given image property (edges for the gradient example).

In practice, a multitude of features are generally extracted from a single image. For simplicity, features stemming from the same type of processing (e.g. texture extraction) are often gathered into *feature vectors*, also called *feature descriptor*s. Sometimes, we will call a "feature vector" simply a "feature" for simplicity. Most often, a descriptor undergoes an additional processing that makes it invariant to some simple variation source (typically, luminance). The interest of using feature descriptors rather than image pixels directly is that they are easier to handle because of their smaller size, their invariance and the fact that they emphasize some image properties useful for the detection task.

There exists several categories of features defined according to the formula used

to compute them. Firstly, we can distinguish between two scopes for computing the features:

- the local scope
- the global scope.

Features computed at a global scope (or more simply, "global features"), as their name suggests, originates from the whole image. An example of a global feature would be the mean luminance of an image. On the contrary, local features are only computed on a limited area of the image. In the following of this dissertation, we will almost only rely on local features as they bring invariance to translation. Indeed, using local features enables to describe only the areas from the image which are straight above the object (i.e. avoiding the background). In comparison, global features are used for tasks that consider the image as a whole, like scene classification (e.g. deciding if a photo was taken in a forest or in a street)[1].

Secondly, we can also categorize local features by the way in which they are extracted:

- sparse features
- dense features.

In the case of sparse features, a preliminary step is necessary to compute the set of image locations where they exist. Those locations are usually selected in a way that is invariant to common transforms (e.g. rotation, translation). The regions corresponding to edges in images are for example invariant to most transforms. On the contrary, dense features are not subject to such constraints and are available anywhere on the image. To summarize, sparse feature integrates an additional aspect of spatial invariance which limits their extraction to a few image locations, whereas dense features do not (see Section 2.2.2). A synthesis of the whole extraction process for local features is summarized in Figure 2.2.

We now review in detail the different types of features that are used in this dissertation and related works from the state-of-the-art. We begin by describing dense features in Section 2.2.1 and then we dwell on sparse feature detectors in Section 2.2.2. Finally, some more elaborate feature descriptors based on gradient histograms are described in Section 2.2.3.

[1]Note that sliding window techniques use global features extracted on sub-images, hence corresponding in reality to local features with respect to the full image (see Section 2.4).

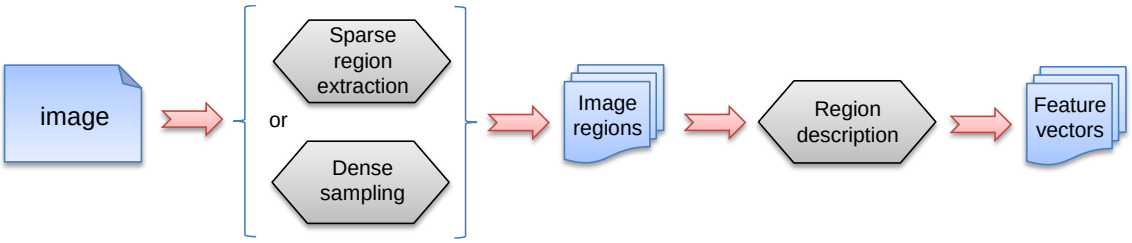


Figure 2.2: The typical dataflow of local feature extraction in an object detection system. In the first step, local image regions are defined either on the base of a dense sampling, or according to a sparse detector. Then, each region is described by a feature vector.

2.2.1 Dense features

2.2.1.1 Convolution-based features

As stated above, dense features are extracted indifferently in every image locations. Most often, they are obtained by convoluting a kernel (i.e. a smaller image) over the image. In this case, the correlation between the image $\mathcal{I}$ and the kernel K translated at the position (x, y) corresponds to the image feature at that location:

$$(\mathcal{I} * K)(x, y) = \sum_{m,n} \mathcal{I}(x - m, y - n) \cdot K(m, n)$$

where $*$ denotes the convolution operator. Since a convolution is highly time consuming for each image pixel, it is common to use the Fourier transform which has a lower computational complexity. The result of a convolution is a response map having the same dimension than the image and where each peak indicates a high correlation between the kernel and the image at the peak location. We now give a non-exhaustive list of popular kernels for extracting dense features.

Template matching

Probably being the most intuitive, template matching consists of convoluting the image with an image patch. It is therefore used to find small model parts (e.g. the patch represents an eye or a wheel) in an image. Because a standard convolution produces biased responses with unnormalized patches (white areas tend to produce higher responses), normalized cross-correlation (NCC) is often used instead. This simple technique is still widely used in recent papers like [UE06] or [TMF07], but overall it has an heavy computational cost.

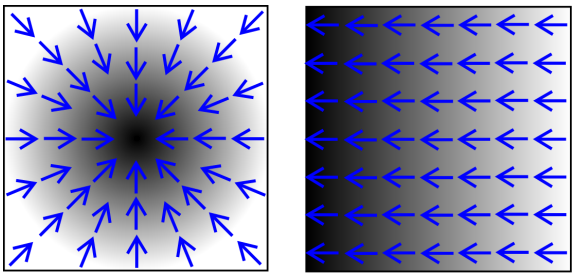


Figure 2.3: Illustration of the gradient field on two simple images. The gradient vectors represent the direction of the local largest change in pixel intensity.

Image Gradient

The gradient $\mathcal{G}$ of an image $\mathcal{I}$ is identical to its mathematical original definition except that its expression is discrete instead of continuous. Two kernels are used corresponding to the x and y image derivatives[2]:

$$\mathcal{G}_x = \mathcal{I} * \begin{bmatrix} +1 & 0 & -1 \end{bmatrix} \text{ and } \mathcal{G}_y = \mathcal{I} * \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}$$

The gradient in a given location (x, y) is thus defined as:

$$\mathcal{G}(x, y) = \left[\mathcal{G}_x(x, y),\, \mathcal{G}_y(x, y)\right].$$

Simply put, the resulting vector field $\mathcal{G}$ indicates the direction $\Theta_{\mathcal{G}}$ of the largest change from light to dark (see Figure 2.3) where $\Theta_{\mathcal{G}} = \arctan\left(\frac{\mathcal{G}_y}{\mathcal{G}_x}\right)$. The rate of change in this direction is encoded by its magnitude $\|\mathcal{G}\| = \sqrt{\mathcal{G}_x^2 + \mathcal{G}_y^2}$.

The main interest of the gradient is that it is fairly insensitive to lighting changes and, contrary to template matching, it is generic and fast to compute. Moreover, the peaks in gradient magnitude indicate the points of sudden change in brightness (i.e., edges) as illustrated in Figure 2.4. To conclude with, the gradient constitutes one of the simplest image feature but its derivative like HOG (Histogram of Oriented Gradients, see Section 2.2.3) are still widely used nowadays.

Apart from template matching and gradient, there exists many other features based on linear convolution of kernels. We can cite for instance the features obtained using the Fourier coefficients, the wavelet transform coefficients or the Gabor filter response

[2]Note that the Sobel filters [SF] are often used instead of the simplistic differential operators presented above in order to be more robust to noise, but the result is essentially the same.

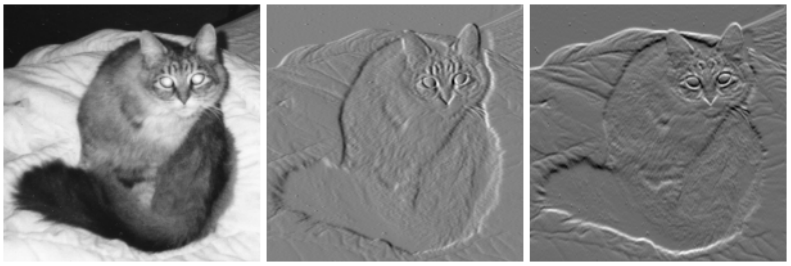


Figure 2.4: Gradient derivatives extracted from the left image. Middle: $\mathcal{G}_x$. Right: $\mathcal{G}_y$. As can be observed, the gradient has a strong magnitude in region with marked contours.

maps.

2.2.1.2 Non-linear features

In this dissertation we do not focus on non-linear dense features. We nevertheless give as example the work of Kruizinga and Petkov [KP99] in which a non-linear texture descriptor is implemented based on the neural processing in the visual cortex (see some texture classification results in Figure 2.5). The texture descriptor is based on the concatenation of simulated neuron output at three different scale levels for each pixel. In chapter 3 we created a texture descriptor inspired from this assembly (see Section 3.3.3) although in our case the three sub-descriptors simply contain a histogram of oriented gradients.

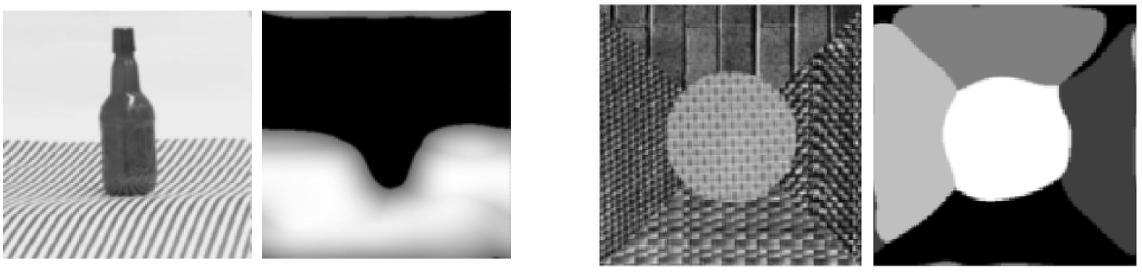


Figure 2.5: Results obtained with the biologically inspired texture descriptor of Kruizinga and Petkov [KP99]. Left pair: dense response map corresponding to a hatched pattern. Right pair: texture classification results (each gray level in the right image stands for a different class).

2.2.2 Sparse features

We saw that dense features are defined for every image pixels, but this is not always useful. Often, an object detection system prefers to focus only on a small set of image regions that are interesting for its purpose. Here, we mean by "region" a set of connected

pixels (e.g. edges) or simply a single point in the image oriented scale-space (those ones are called *keypoints*). For instance, plain areas of an image like a blue sky do not provide valuable information for car detection. An interest region detector thus selects a subset of regions within an image based on a predefined low-level criteria, making this extraction very fast. Then, only the regions selected by the detector are further analyzed in the rest of the detection process.

The criteria used for extraction is generally defined in order to comply to a repeatability constraint. This implies that the detector must yield consistent results despite the presence of usual transforms (namely noise, lighting change, rescaling, in-plane rotation and even sometimes affine transformations). That is, the extraction of interest regions must be invariant to these transformations. Edges or corners for instance are invariant to most of those ones (e.g. the Harris corner detector [HS88]).

In the literature, pairs consisting of a region location and an associated feature descriptor are often addressed as "sparse features" or "invariant local features" due to their limited number, their localized aspect and their invariance to usual transform. Sparse feature detectors have been developed from almost the very beginning of image processing and a non-exhaustive list includes edge detectors (Canny [Can86]), keypoint detectors (SIFT [Low04], MSER [MCUP02], Hessian-Harris corners [HS88]) and region detectors like [AMFM09]. We now give some details about three of the most popular types of sparse features, namely edges, keypoints and regions, some of which being used later in our contributions.

2.2.2.1 Edges

Edge features, sometimes referred as contours or boundaries[3], have long been used by researchers in the field of object detection as they are one of the simplest and more intuitive interest regions. There exists a gap, however, between the contours that a human being would draw in a given image and the edges really detectable in the same image using the gradient magnitude (see Section 2.2.1). This is because humans are influenced by their understanding of the scene. Recent approaches of contour detections like [MAFM08] have nevertheless succeeded to reduce this gap at the cost of complex computations that search for global solutions over the whole image. In this dissertation, for efficiency reasons, we limit to a simpler detector that was designed by Canny in 1986.

The Canny edge detector [Can86] is one of the oldest system for detecting edges, but it is still widely used. It outputs a set of edge pixels based on the gradient magnitude. In

[3]but these words can have a slightly different meaning as they refer to high-level object contours whereas edges are only related to low-level image properties, see [GLAM09].

a first step, the input image is blurred with a Gaussian convolution in order to reduce the amount of white noise. Then, the gradient magnitude and orientation are computed for each image pixel. In a third step, a non-maxima suppression is carried out to eliminate the pixels which are not local maxima of magnitude in the gradient direction. Finally, a threshold with hysteresis is used to select the final set of edges from the remaining pixels: all pixels above a high threshold in term of gradient magnitude are tagged as edge pixels as well as every pixel above a low threshold and neighbor of an edge pixel. This hysteresis technique is more reliable than a simple thresholding, as it is in most cases impossible to specify a global threshold at which a given gradient magnitude switches from being an edge into not being so. Moreover, the process is fast, simple to implement and efficient enough to explain its success until today. An example of edges extracted by this method is given in Figure 2.6.

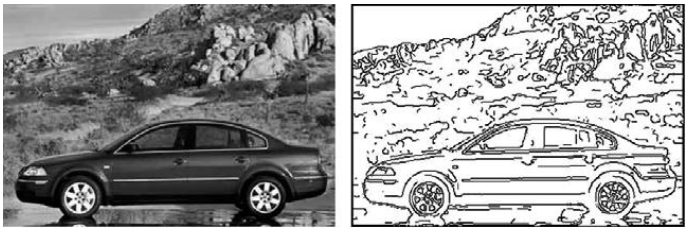


Figure 2.6: Example of edges extracted using the Canny detector.

Additionally, a common finishing stage is to polygonize the set of edge pixels into line segments to simplify their representation. The set of sparse features thus obtained is however not so reliable for matching a same object across different pictures because of the polygonization noise (typically, line segments undergo cuts or on the contrary merge together).

2.2.2.2 Keypoints

The recent emergence of *keypoints,* whose most famous avatar is probably SIFT [Low04], has had a considerable influence on specific object recognition (see Section 2.3). Formally, a keypoint, also called *interest point*, is simply a location $\mathbf{p} = (x, y, \sigma, \theta)$ in the oriented scale-space of the image (in the literature, it often comes implicitly with an associated descriptor). Different techniques have been proposed to extract keypoints in images. We can cite the SIFT detector [Low04], SURF [BTG06] and the Harris-Hessian corner detector [HS88]. Lately, affine region detectors [MTS*05] have been developed to improve keypoint detection by approximating 3D viewpoint changes. Two recent state-of-the-arts about keypoints and affine region detectors can be found in [MP07, MS05].

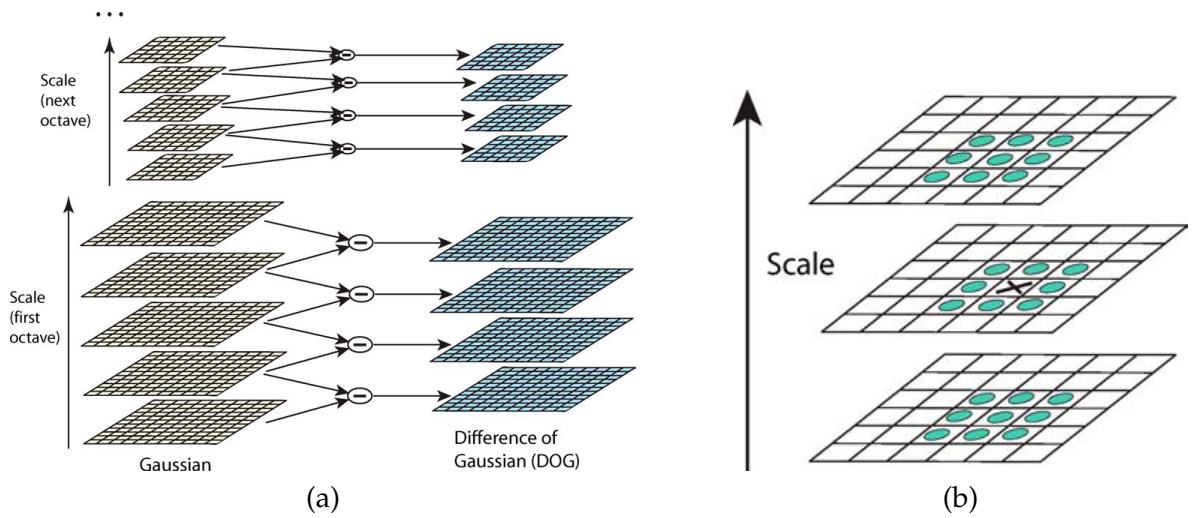


Figure 2.7: (a) Fast computation of the pyramid of difference-of-Gaussian using repeated convolutions with Gaussian (left) and subtraction of adjacent Gaussian images (right). After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeats. (b) Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors at the current and adjacent scales.

We only describe in this section the SIFT detector as it has been proved to be one of the most robust and efficient, as well as the only fully scale invariant method [MP07, MS05]. Introduced by Lowe in 1999 [Low99], the Scale Invariant Feature Transform (SIFT) firstly extracts extrema in the difference-of-Gaussian space (see Figure 2.7). Firstly, repeated convolutions of Gaussian kernels with increasing radius are applied to the input image and the result are stacked in so-called "octaves". Each time that the Gaussian radius exceeds by a factor of 2 the first image of the current octave, the corresponding image is downsampled by a factor of 2 and the process repeats for another octave. Then, adjacent images in octaves are subtracted in order to compute difference-of-Gaussian as a fast approximation to Laplacian (see Figure 2.7.(a)). Then, maxima are searched in the scale-space of difference-of-Gaussian and each point found constitutes a keypoint center at the corresponding scale (see Figure 2.7.(b)). This process especially fits textured objects as strong texture provides a large amount of stable extrema. Finally, an orientation is assigned according to the dominant gradients around the point. The locations thus obtained are invariant to a translation, an in-plane rotation, a rescaling and a illumination change of the input image.

We use SIFT keypoints later in our contributions for their good propensity to specific object recognition [Low04].

2.2.2.3 Regions

Uniform image regions have also been considered to generate sparse features. The Maximally Stable Extremal Region (MSER) detector [MCUP02] is probably the most famous one in this field. It is based on a segmentation of the image using a watershed algorithm and various thresholds. At a high water threshold, all regions merge into a single one but the algorithm is only interested in those regions that resist the watershed the longest. Those extremal regions possess two highly desirable properties: they are invariant to continuous (and thus projective) transformations of image coordinates as well as to monotonic transformations of image intensities. Moreover, an efficient (near linear complexity) and a fast detection algorithm is achieved in practice, making MSER one of the most popular interest region detectors with SIFT (e.g. see [SREZ05, SSSFF09]).

More complex region detectors have been recently developed, like the one of Arbelaez et al. [AMFM09] but unfortunately these detectors are not designed for interest region detections. Instead, they aim at segmenting the image at the highest possible semantic level.

2.2.3 Histogram-based features

Once that a set of sparse image locations have been extracted, each one has to be tagged by a descriptor in order to ease its retrieval and allow its comparison with other descriptors. We saw in Section 2.2.1 how to extract the gradient as a dense vector field from an image. We present here different descriptors that are all based on accumulating the gradient vectors in histograms. Note that those description techniques can be used indifferently for depicting the global image or local patches, depending on the image area on which they are computed. The purpose here is to create robust and distinctive descriptors, both properties being very important to ease subsequent detection schemes.

2.2.3.1 Local descriptors

The SIFT descriptor

We described above the SIFT detector, responsible for choosing a sparse set of invariant points in the input image. The following step consists of building a discriminant descriptor for each such point using the SIFT descriptor.

The SIFT descriptor is a 3D histogram in which two dimensions correspond to image spatial dimensions and the additional dimension to the image gradient direction. It is computed over a local square region of a given radius σ centered on a given point

$\mathbf{p} = (x, y)$ and rotated by a given angle θ (see Figure 2.8). As depicted by Figure 2.8, the histogram consists of 4×4 spatial subdivisions and 8 orientation intervals of 45° each, which makes a total of 128 bins for the final descriptor. During the computation, each gradient vector belonging to the local square region contributes to the histogram in the corresponding bin depending on its location in the local region and on its orientation (the contribution is proportional to the gradient magnitude). In order to avoid boundary effects, the contributions are spread over $2 \times 2 \times 2 = 8$ bins using linear interpolation. Finally, a normalization step is applied to make the 128-dimension descriptor invariant to lighting changes.

The SIFT descriptor has been shown by Mikolajczyk and Schmid [MS05] to be one of the most robust descriptors to perspective and lighting changes with the Shape Context [BM00] descriptor. Moreover, it is robust to small geometric distortions. Due to its popularity, a lot of variants have been proposed: a non-exhaustive list include GLOH [MS05], PCA-SIFT [KS04], SURF [BTG06] and GIST [SI07]. Recently, new keypoint descriptors dedicated to real-time constraints have been developed by Lepetit et al. [LLF05] (later improvements in the same framework include the works of Calonder et al. [CLK*09] and Özuysal et al. [zCLF09] for a fast extraction, description and matching of keypoints). They rely on fast pixel-to-pixel comparisons rather than gradient histograms. As a result, the description step is much faster than with SIFT and the descriptors also seem to better handle perspective distortions.

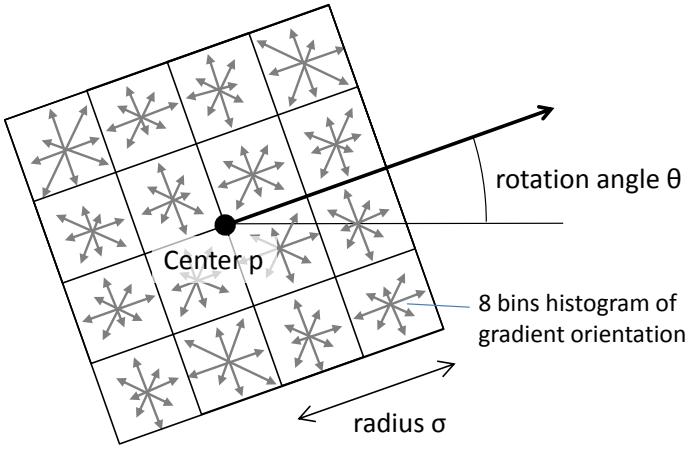


Figure 2.8: The SIFT descriptor consists of a 3D histogram in which two dimensions correspond to image spatial dimensions (4×4 bins) and the additional dimension to the image gradient direction (8 bins). The histogram covers a square region of the image parametrized by a radius, a center and a rotation angle.

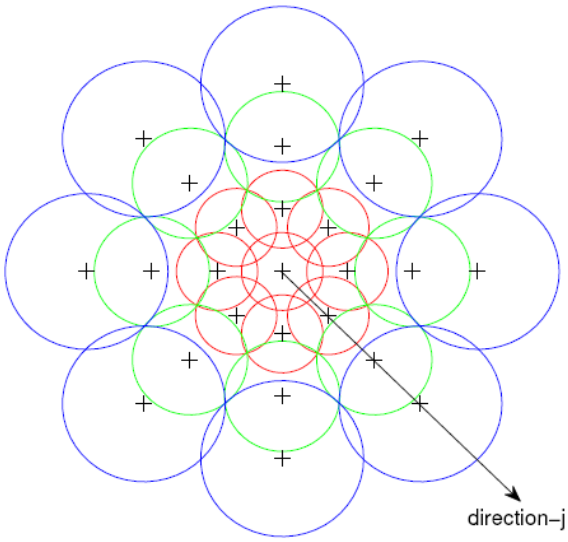


Figure 2.9: The DAISY descriptor [TLF08]. Each circle represents a region where the radius is proportional to the standard deviations of the Gaussian kernels and the '+' sign represents the locations where the convoluted orientation maps are sampled. The radius of the outer regions are increased to have an equal sampling of the rotational axis which is necessary for robustness against rotation.

DAISY

Tola et al. [TLF08] have introduced in 2008 a feature descriptor named DAISY which is similar in many respects to SIFT at the difference that it is designed for a fast dense extraction. It was shown to achieve better results than SIFT for wide-baseline matching applied to stereoscopic images.

Specifically it also consists of several histograms of oriented gradients which are not positioned on a square grid like SIFT but on a daisy-shaped grid (see Figure 2.9). The key insight of DAISY is that computational efficiency can be achieved without performance loss by convoluting orientation maps to compute the bin values. In other words, the original gradient map of the image is divided into height maps based on the gradient orientation (i.e. each map only takes care of a 45° bin), and a Gaussian blurring at several scale levels for each map achieves a pre-computation of the histogram bins at every image location and scale. Histograms picked up at the locations shown in Figure 2.9 are finally concatenated into the final feature descriptor for a given center and scale.

In Chapter 3, we use a related descriptor where the extraction part is strongly inspired from the work of Tola et al. [TLF08]: the difference is that we used a Fourier transform in the orientation space to compute the orientation maps in order to obtain oriented descriptors without interpolating the bin values.

2.3 Specific Object Recognition

Now that low-level features have been presented, we study how to combine them in order to effectively take a decision about the presence of a given model object in a test image. As our first contribution focuses on specific object recognition, we begin by a summary of the existing methods for specific object recognition, yet including a few references to related class object detection methods when necessary. Existing systems for specific object recognition can be classified as follows:

- methods using global features,
- methods using sparse local features, from which we can distinguish:
  - the ones relying on a rigid matching, and
  - the ones relying on a non-rigid matching.

2.3.1 Using global features

Techniques using global features for specific object recognition are quite anecdotic in the state-of-the-art. Indeed, the advantages of using local features compared with global features are huge as we will see below. This is essentially why global techniques have been investigated *before* the emergence of reliable invariant local features.

To put it simply, techniques using global features aim at recognizing the object in its whole. To achieve this result one generally has to learn, from a set of images, the object to recognize. Nayar et al. [NWN96] have presented in 1996 a fast method which can handle one hundred objects while still being effective. They conducted a principal component analysis of the model pictures in order to extract eigen-views that eliminate the lighting noise. Then, an optimized scheme of nearest neighbor search was used to quickly match a test image with a model object. A lot of other works relying on global features have been proposed for class object recognition, like the one of Viola and Jones for face detection with boosted cascade of simple classifiers [VJ04].

However, using global features has several drawbacks: first of all, the object has to fill the whole test image in order to match the model. To overcome this issue, sliding window techniques are generally used to enable invariance to translation, scaling and rotation. This solution nevertheless has a large computational cost (thousands of windows must be examined [GLAM09]) whereas specific object recognition usually implies real-time constraints. Precisely retrieving the 3D model pose using global features also appears very difficult. Thirdly, the amount of data needed for training is usually huge,

as well as the training time. A last problem is that these approaches have difficulties dealing with partial occlusions. Those issues are admissible for class object recognition since a lot of model pictures are necessary anyway to precisely learn the intra-class variations, and the task is difficult enough to afford to avoid the occlusion problem. On the contrary, we expect more from a simpler system dealing with specific objects: i.e., training the model from only a few pictures and bearing occlusions.

2.3.2 Using local features

A wide variety of specific object detection methods relies on sparse local features. Since the properties used for extracting these features are invariant to most real-world transforms, a common technique is to describe the model object by a constellation of these local features in the training stage; and to search the same spatial arrangement of features in the test image during the detection stage. To summarize, the general scheme usually implies three steps:

1. The first one is the extraction and description of sparse invariant local features, in both test and model images.

2. The next step consists of selecting test image features that match the model ones (i.e. pairwise matches between keypoints, lines or regions).

3. The final step elects the best subset of test image features based on their spatial consistency with respect to the geometrical arrangement of the model features.

In this way, the object position can be precisely computed as well as the occlusion map, provided that the model object is covered by a sufficient number of local features (e.g. like in [FT04]). A fourth additional step is also often performed to assert a detection using a probabilistic model which depends on the method.

Using keypoints as local features

Among all different types of local features, one type holds more attention than others. Indeed, the emergence of keypoints has significantly improved the state-of-the-art in various domains of computer vision and more particularly in the detection of specific objects (e.g. see the pioneer work of Schmid and Mohr [SM96]). Clearly, most methods presented in the following are based on keypoints.

In fact, recognition methods using keypoints present numerous advantages: they inherit the invariant properties of keypoints (namely to translation, scale and rotation) and

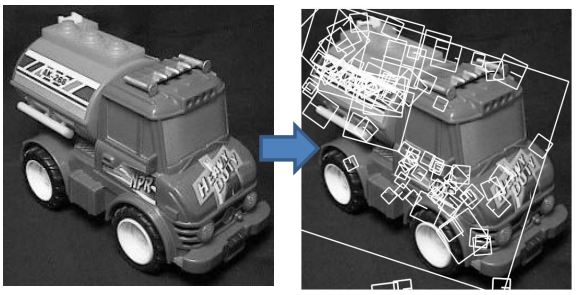


Figure 2.10: Representation of an object as a constellation of keypoints. Each keypoint is associated to a local square patch where the keypoint descriptor is extracted. In the method of Lowe [Low04], the position of all keypoints is constrained by a global affine transform of their coordinates in the model image, thus enabling to filter out most incorrect detections.

the localized aspect of the features makes them robust to occlusion without significant increase in complexity. Moreover, thanks to the high descriptive power of the keypoint descriptors (see Section 2.2.3), any training is quite unnecessary. Finally, those methods are generally simple to carry out and they can perform close to real-time (in particular, see [CLF08]). Another point that could explain why keypoints have become so popular these last few years is that the concept of decomposing an object into a constellation of small interest patches is somehow familiar with the human visual system. An example of an object described by a constellation of keypoints is shown in Figure 2.10. We distinguish in the following between two different ways of verifying the geometric consistency of a constellation: namely, rigid and non-rigid techniques.

2.3.2.1 Rigid matching

Methods that rely on a rigid transform (e.g. a projective transform) to constrain the local feature positions can be classified into two categories: RANSAC-based methods and Hough-based methods.

Hough-based

The Hough transform was first patented in 1962 and later adapted for the computer vision community by Duda and Hart [DH72]. Briefly, the Hough transform involves two stages: in the first one, votes are accumulated in the parameter space based on an examination of the available features in the test image (because of imprecision, votes are usually cast on intervals in the parameter space, or equivalently, the parameter space is quantized into several bins); in the second stage the votes are clustered and the position of the largest cluster yields the optimal transform parameters.

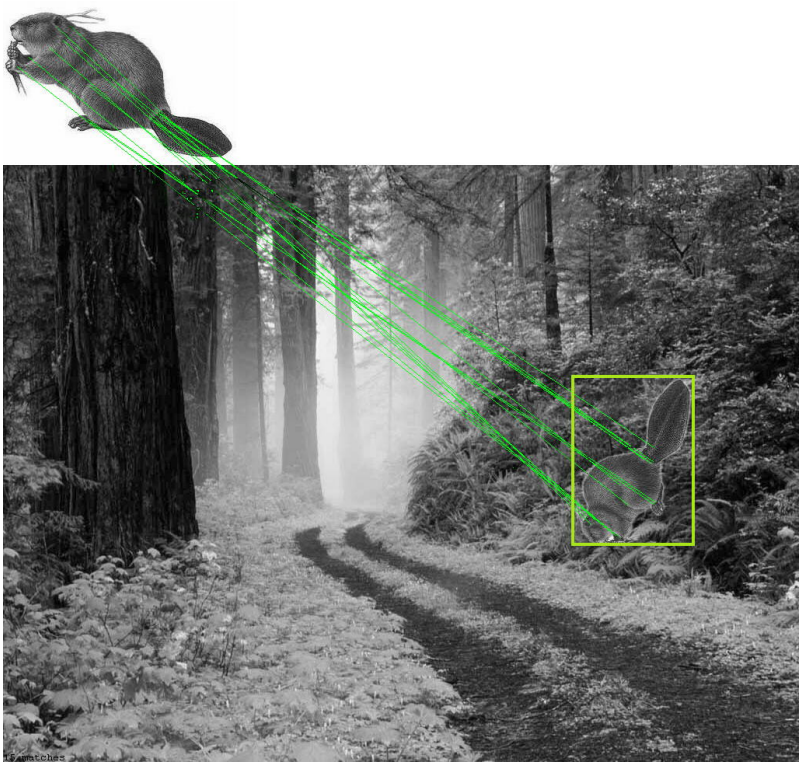


Figure 2.11: Example of robust detection of specific instances using Lowe's method [Low04]. In spite of a projective transform of the model image (top) in the test image (bottom, yellow box) and a large amount of clutter, Lowe's method is still able to correctly detect the beaver thanks to a spatial verification of the matched keypoint configuration. In this image, the search of the beaver model object results in 14 detections, the best one being correct with a probability score of 100% while the false positive ones have much lower scores (i.e. all below 23%, average score is 8.9%).

From all specific object detection methods using the Hough transform, the method of Lowe [Low04] is probably the most famous and popular. During the training stage, multiple views of the same object are combined in order to compute a set of characteristic views. In the same time, keypoints belonging to the model views are indexed in a k-d tree in order to enable a fast pairwise matching between model and scene keypoints (this technique is scalable to a large number of model objects and thus has been replicated in many other works, e.g. see [BTG06]). During detection, keypoints in the scene image are extracted and matched with the model keypoints using the k-d tree. Then, the Hough transform is performed: each matched scene keypoint votes for an approximate model position in the parameter space (assuming a simple similarity transform, the keypoint's position, scale and orientation suffice for the extrapolation). Finally, peaks of votes in the parameter space are further verified with an affine transform and a probabilistic decision determines whether the object is really there or not, based on the amount of spurious matches in the concerned area. An example of detection using Lowe's method is presented in Figure 2.11. The drawback of such an approach is that it does not take into account the real 3D shape and the 3D transformations of the object and therefore is unable to recover its precise spatial pose. Moreover, Moreels and Perona [MP08] have shown that the choice of the bin size in the Hough space is problematic (smaller bins cause fewer true positives, while larger bins cause more spurious detections). They have proposed instead a cascaded procedure which adds an additional RANSAC stage (see below) after the Hough transform, and have also improved the final probabilistic decision in order to reduce the false alarm rate. However, their probabilistic model relies on correct and incorrect keypoint match densities which are rather hard to obtain (they used a mechanical rotating table with different objects placed on it to obtain ground truth feature matches [MP07]).

Finally, the method of Gu et al. [GLAM09] is also related to the Hough transform. By representing the model objects as bags of regions (each one weighted during the training using a machine learning technique similar to a support vector machine) and then applying a similar voting scheme in the scale-space of possible instance locations, they manage to detect textureless objects unfit to be depicted by keypoints. The used region detector is unfortunately very complex and not suitable for fast applications.

ransac-based techniques

The RANSAC algorithm was introduced by Fishler and Bolles in 1981 [FB81]. It is possibly the most widely used robust estimator in the field of computer vision. Figure 2.12

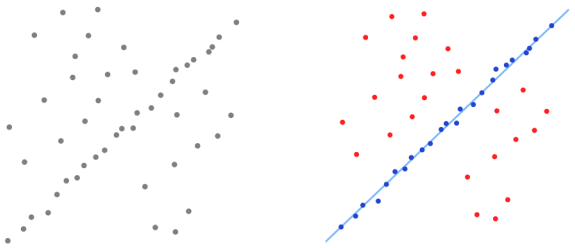


Figure 2.12: RANSAC can extrapolate the correct line parameters despite the presence of many outliers (source: Wikipedia).

illustrates how the line estimated by RANSAC from a noisy set of points effectively recovers the optimal parameters. The RANSAC algorithm can be summarized as follows: assuming a noisy set of samples and a given spatial transform, the algorithm iteratively picks a small number of input samples and estimate the transform parameters of the associated fitting problem. Then, a score is given to this trial to measure its quality, usually by counting the number of *inliers*, i.e. the number of other samples that comply with this parametrization. Finally, the transform parameters corresponding to the best trial are returned. Because RANSAC relies on a succession of random trials, it is not guaranteed to find the optimal solution. A probabilistic formula is used in practice to determine the number of iterations necessary to output the optimal solution with some confidence.

Numerous papers related to the matching of specific objects or even whole scenes, like short and wide baseline stereo matching [CM02, MCUP02], motion segmentation [Tor95] and of course specific object detection [LLF05, RLSP06] have used RANSAC coupled with keypoints as robust estimator. Lepetit et al. [LLF05], for instance, have presented a real-time system based on randomized trees for keypoint matching which have been later improved by Özuysal et al. [zCLF09] into *ferns*. Their solution is notably robust against changes in view point and illumination. In a different fashion, Rothganger et al. [RLSP06] have considered affine invariant keypoints to recover more efficiently the object pose from the matched feature patches. Even if those methods give good results, a common drawback is that the 3D shape of the model objects has to be learned beforehand.

Finally, note that the original RANSAC algorithm has been adapted into several variants by Chum et al. [CMK03, CM05, CM08]. We have implemented in Chapter 4 for comparison purpose the variant called "Locally Optimal RANSAC" (LO-RANSAC) [CMK03] which assumes two transforms to fasten the matching process (the first transform being an approximation of the second one). In the main loop, the simplified transform is used as it requires less samples to estimate the transform parameters, hence reducing the number of iterations. During the verification step, a secondary RANSAC using the full

transform is launched only on the set of inliers discovered by the simplified transform. Chum et al. have shown that this way of processing gives better results than using a standard RANSAC and is also faster. Our experiments (see Chapter 4) have confirmed this statement with the setting proposed by Philbin et al. [PCI*07] in which a similarity transform is used for the main RANSAC loop (only requiring one keypoint match to estimate the transform parameters) and a projective transform for the verification step (requiring four matches).

Other techniques

Rosenhahn and Sommer [RS05a, RS05b] have proposed a technique for 3D object tracking. To that aim, the conformal space is used to embed feature points, lines and circles. Nevertheless, their method assumes that the matching step is performed externally to their method (i.e. it can only be used to track an object after a manual matching initialization). To our knowledge, no full detection scheme relying on this theory yet exists. In a different style the older system of Jurie [Jur01] also use edge features to represent objects. Indexing techniques are used to achieve fast 2D and 3D object recognition while additional optimizations are used to recursively prune the space of hypothesis poses at matching time. Yet, the general shortcomings of such edge-based approaches is that edge features alone carry low distinctiveness and that the quality of the segmentation (edge extraction) is variable (it is known to be not robust to noise in general).

2.3.2.2 Non-rigid matching

As we saw rigid matching is efficient, but obviously it can not handle distortions like what happens to a bent magazine or to a yawning face, for instance. Non-rigid matching, on the contrary, assumes that the model object can be decomposed in a set of different independent parts that can move on their own (with some limits, of course). This strategy has been shown to give more flexibility to the model [FTG06] and to increase performances thanks to the fact that distant features are disconnected [CPM09]. The matching cost is however often superior compared to the case of a rigid matching, but this is expected as the number of parameters that govern a non-rigid transform is by far superior to the number of parameters for a rigid transform.

Non-rigid matching can be roughly categorized into two kinds of techniques: those relying on graph matching and those denoted as part-based models (note that both categories are strongly related, as we will see).

Graph matching

Graph matching seems to be a straightforward way to resolve specific object detection. Indeed, after having extracted some sparse local features, both model object and scene can be represented as graphs (see Figure 2.13). Moreover, graph matching operates at a local scale by comparing pairs of nodes or pairs of edges, thus avoiding the need of a global (rigid) transform. Formally, the graph matching problem can be formulated as the maximization of the following objective function [GR96, BBM05, CSS07]:

$$E(\mathbf{M}) = \sum_{\alpha,i,\beta,j} \mathbf{H}_{\alpha,i,\beta,j}\mathbf{M}_{\alpha,i}\mathbf{M}_{\beta,j}$$

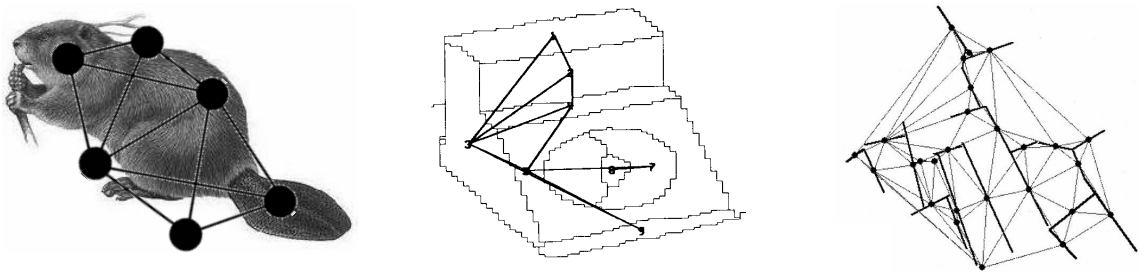
where

- $\mathbf{M}$ is the desired match matrix (i.e. $\mathbf{M}_{\alpha,i} = 1$ means that node α from the first graph is matched to node i from the second graph, otherwise $\mathbf{M}_{\alpha,i} = 0$). $\mathbf{M}$ is usually subject to an additional constraint: a many-to-one matching scheme is often allowed (i.e. $\forall i, \sum_{\alpha} \mathbf{M}_{\alpha,i} = 1$ or conversely by interchanging i with α), and

- $\mathbf{H}$ is a matrix which describes the compatibilities between edges of the two graphs. $\mathbf{H}_{\alpha,i,\beta,j}$ thus measures how much the edge (α, β) (from the first graph) and the edge (i, j) (from the second graph) are compatible. In the case where $\alpha = \beta$ and $i = j$, $\mathbf{H}_{\alpha,i,\beta,j}$ simply measures the compatibility between node α and node i.

Driven by this straightforward formulation, researchers have long proposed graph matching as a powerful tool for classifying structured patterns (see [CFSV04] for details). More specifically, a large amount of studies have tackled the recognition problem using graph matching, for instance applied to the detection of faces [WFKvdM97], indoor objects [GR96] or mechanical parts [KK91].

The main drawback of this kind of approaches however lies in the computational power needed to match two graphs. In fact, it has been shown that the subgraph isomorphism problem (i.e. what we practically call graph matching) is NP-hard. As a consequence, researchers have either focused on sub-problems easier to solve (e.g. the matching of bipartite graph [KK91]), either proposed heuristics and optimizations so as to efficiently reach or approximate the global solution [BDBV01, SBV01, MB98]. For instance, Messmer and Bunke [MB98] have shown that the subgraph isomorphism resolution using a model graph decomposition and a set of graph edit operations can be very robust compared to classical A*-like algorithms that were developed first. On the contrary, recent researches have focused on global approximations (graph-cuts [TKR08],



(a) (b) (c)

Figure 2.13: Representing objects as graphs (nodes are figured with black dots). (a) Using keypoints; (b) using regions (here, surfaces); (c) using line segments.

tensor-based [DBKP09] and/or spectral method [CSS07]), but the timing performances remain disappointing for large graphs. In comparison, the historically older relaxation methods perform faster and stay competitive in practice [FSGD08, MGMR02, TKR08], although no theoretical guarantee ensures their convergence.

Among all applications of graph matching to specific object detection, we can cite the system of Kim et al. [KHP07] which is dedicated to recognize indoor objects. In their approach, edge segments are firstly extracted and described in term of their neighborhood (i.e. luminance and color). Then, they are matched between the scene and the model using logistic classifiers. Finally, a spectral method [CSS07] is used to solve the global assignment problem. The method shows superior results compared to a SIFT based approach, but this is expected as the problem setup is dedicated to the detection of textureless indoor objects. The works of Christmas et al. [CKP94] and Wilson and Hancock [WH99] for matching road segments in maps are also interesting. In the first approach, a two-levels hierarchy based on the size of the line segments yields good matching results despite its apparent simplicity.

However, the main problem of graph matching techniques in our opinion lies in the discretization necessary to convert an image into a graph through a selection of some image spots (i.e. each one being transformed into a graph node). Indeed, this step inevitably results in a loss of relevant information. We will see in Chapter 3 how this issue can be addressed by introducing the notion of *continuous graph* (i.e. a graph in which the number of nodes is infinite) thanks to the use of dense or semi-sparse features (namely textures and edges).

Part-based models for specific object recognition

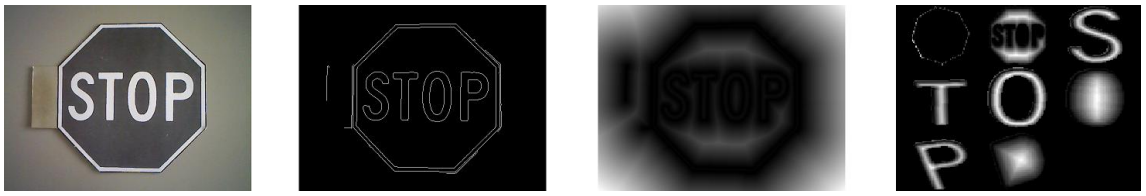
Apart from graph matching, a closely related field is the class of *part-based* object recognition methods. Although the term "parts" may refer to semantic parts (especially for

class object recognition methods, see next section), we restrict here to the case of specific objects. In this context parts thus only mean local patches of the object surface, most often derived from sparse feature detectors. Part-based models for specific object recognition address the problem in a similar fashion than graph matching (i.e. decomposing objects in parts loosely connected) but use different techniques to solve the part assignment problem.

First of them, the work of Ferrari et al. [FT04, FTG06] deals with the object recognition problem in a greedy fashion. Firstly, local patches are densely sampled on the model objects in order to learn their entire surface. During detection, the method of Ferrari et al. gradually explores the areas surrounding some initial matches obtained using sparse affine features, recursively constructing more and more matching regions, increasingly farther from the initial ones. To eliminate wrong matches, the process alternates between contraction phases and expansion phases, hence achieving object segmentation at the same time. A similar approach have been proposed by Kushal and Ponce [KP06] specifically for the detection and 3D pose recovery of 3D rigid objects. The problem of those method is that they only fit strongly textured objects preferably viewed in close-up and that they are very slow (4-5 minutes to process a pair of model and scene images on a 2.4 Ghz computer). Moreover, the segmentation aspect (dense coverage) of the method makes the model very heavy and is not always desirable for practical applications.

On the contrary, the approach of Detry et al. [DPP08] is centered on edge features connected by a hierarchy. Their method allows to infer the position and the 3D pose of a model instance, but the detection time is also slow because of the probabilistic handling of the resolution: belief propagation is performed in moderately high dimensional spaces to enable the invariance to translation, scale and 3D rotation. Even if their optimization using a density estimation technique enables an important speed-up, it still takes one minute to detect the object and its pose. A similar work was done previously by Scalzo and Piater [SP05] where an expectation-maximization scheme was used to identify and code spatial correlations between features/parts.

Recently, an other approach using edge features has been proposed by Holzer et al. [HHIN09]. Their technique relies on a depiction of the model object as a set of closed contours. For each contour template, a distance map is computed during training to store the minimal distance between each template pixel and the closest edge pixel (see Figure 2.14), which is robust to segmentation noise. By training a classifier for various template poses, they could obtain robustness against perspective effects. In addition, spatial relations between multiple contours on the object are learned and later used for outlier removal. At run time, the classifier provides the identity and a rough 3D pose



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

Figure 2.14: An illustration of the distance transform used in [HHIN09]. (a) A stop sign picture; (b) Edges extracted with the Canny detector [Can86]; (c) Distance maps computed from (b): the closer we are to an edge, the smaller is the distance (dark pixels correspond to small values); (d) the eight templates extracted from closed contours of the model object.

of the Distance Transform Template, which is further refined by a modified template matching algorithm that is also based on the distance transform. Of course, this method is only relevant for the objects presenting planar contours on their textured surface.

2.4 Class Object Recognition

Finally, we give in this section an overview of current class object recognition techniques. Basically, the main difference between specific object recognition and class object recognition is that in the latter case, intra-class variations are larger (in particular, beyond 3D pose changes) and more complex to model. Practically, this means that the boundary between positive and negative instances in the feature space has a potentially complicated shape, in particular because the semantic definition of an object class differs from the feature-based definition. As a consequence, a widely used solution is to transfer the burden of modeling this complex boundary to machine learning algorithms. Those ones are indeed dedicated to handle this kind of problem and can efficiently learn a decision surface from samples[4] (generally in an optimal way regarding a certain formulation of the problem). Those techniques are either discriminative (i.e. existing machine learning techniques like Support Vector Machine (SVM [BGV92]), boosting (e.g. AdaBoost [FS95] or its variants) or generative (probabilistic Bayesian models, e.g. the naive model [CDF*04]). The result is called a classifier, as its task is simply to decide whether a given sample (expressed in the feature space) belongs to the model class or not. To summarize, the main trend in class object recognition is thus to express the model images as vectors in a relevant feature space, and to train a classifier with negative and positive sample vectors so as to learn the class distribution (in other words, we talk about statistical learning). We will now review in more details some of the most efficient feature spaces

[4]We only talk about supervised learning in this dissertation.

found so far as well as frequent schemes used for detection.

2.4.1 Feature spaces for class object recognition

Why not using simple feature types ?

As we saw previously, simple features such as keypoints are enough for specific object recognition. Although using the same features as well for classes of objects could appear to be a good idea, it is not that simple. The main problem lies in the fact that simple features are often too much specific, enabling few generalization regarding the larger intra-class variations occurring for classes. To overcome this issue, class methods have to add additional steps (e.g. creating histograms of features, see below) leading to higher-level features which are more invariant to class variations.

Bag-of-words

The Bag-of-Words (BoW) features have been firstly proposed by the natural language processing community. The original approach was aiming at representing a textual document as a histogram of the words composing it. (The term "bag" originates from the fact that the position information of the words in the document is lost in the histogram bining process). This feature space is known to be extremely effective for textual documents (e.g. that's how Google indexes web pages), so several researchers have proposed an application of the same principle to images.

In computer vision, the solution which has been proposed by several groups is to replace textual words by visual words [CDF*04, FFP05, LSP05]: first, local features having a high descriptive power (typically SIFT descriptors) are extracted from the image (using either dense sampling or salient detector); then, each local feature is quantized, i.e. associated to the nearest word in a predefined codebook (we assume that the codebook, or "visual dictionary", has been preliminary built using clustering techniques like k-means in the descriptor space); and finally, the descriptor for that image is computed as a histogram of the visual words present in that image. After that, the image is represented as a point in the space of histograms. In this feature space, two images are compared based on the distances between their histograms. Popular distances include the chi-squared distance and the minimum intersection between histograms [ZBMM06, BZ07]. Note that the computer vision community still benefits nowadays from techniques used in the textual document field (e.g. see Tirilly et al. [TCG10]).

Surprisingly, bag-of-features performs very well for various tasks despite its simplicity (e.g. object recognition [BZ07] or image classification [ZBMM06]). In fact, the

loss of spatial information seems to be rather an advantage for handling class variations as it provides invariance to pose/viewpoint and geometric variations (in fact, the bag-of-feature representation amounts to consider images as textures having no spatial organization by definition [ZBMM06]). On the other hand, the lack of spatial information is also one of the most frequent criticism against BoW. In fact, there are applications which require to take into account the geometric configuration of the local features (at least partially). As a consequence, an additional spatial verification step is sometimes performed after the histogram comparison (e.g. see Chum et al. [CPM09]) or spatial information are directly incorporated into the histogram (e.g. the spatial pyramid of Lazebnik et al.[LSP06]).

Histogram of oriented gradient (HOG)

Now that we have presented histograms of visual words, we present the Histograms of Oriented Gradients (HOG). Introduced by Dalal and Triggs [DT05], the insight is as the name suggests to accumulate image gradients into histogram bins corresponding to different gradient locations and orientations. More specifically, the image is divided into a dense grid of uniformly spaced cells. Each cell then contains a single histogram with several orientation bins that receives the contributions of the underlying gradient vectors. Contrary to the SIFT descriptor presented above, the HOG feature is intended to describe the image in its entirety (or the sub-image in the case of a sliding window, see below) without any rotation or scale invariance.

Dalal and Triggs [DT05] have studied the influence of each stage of the feature computation process regarding the performance of a pedestrian detection application. They have concluded that fine-scale gradients, fine orientation binning, relatively coarse spatial binning, and high-quality local contrast normalization in overlapping descriptor blocks are all important for good results.

High-level local features

While the two previous features are global features, we now present local features which are specially designed to handle the case of class objects. In other words, those local features are tolerant to some variations. An excellent example is the biologically-inspired features presented by Serre et al. [SWB*07]. In their work, the extraction of features follows the process explained at the beginning of this chapter: the image is convoluted by Gabor filters (corresponding to C1 cells in the visual cortex), then the response maps are sub-sampled and max-pooled in a local frame (corresponding to C2 cells); after that

the maps are again convoluted (V1 cells) and a final sub-sampling followed by max-pooling yields the feature vector (V2 cells). Although the feature extraction process is costly computationally speaking, the scene classification results are very good [SWB*07], and subsequent works have also proved that those features can be very efficient for recognizing class objects [ML06].

In this dissertation, we also present high-level features designed to detect model parts. Our features are somehow related to the features of Serre et al. [SWB*07] as they are composed of several local features loosely connected so as to get a maximum response with respect to a model groups of feature in a local frame (i.e. some sort of max-pooling). Finally, note that other types of high-level features have also been developed, most of them being inspired by biological processes in the human visual cortex [KP99, JWXD10, SI07].

Comparison to specific object detection systems

To conclude this subsection, in general the feature types used in the case of class objects are either global or dense. Compared to the simple sparse features used for specific object detection (i.e. structural methods), those types generate more data and hence multiply the overall computational cost by a large factor. As a result, many class object recognition systems are not real-time *at all*. To conclude with, specific object detection requires fast machinery with respect to its range of applications, and hence cannot afford the complex features and processing used in the case of class object recognition.

2.4.2 Detection schemes

Sliding windows

The simplest and yet widely used strategy to detect object in images is to use a sliding window. In order to enable invariance against translation and scale, the recognition process follows the following steps:

1. A window scans the input image at various locations and scales.
2. For each window:
   (a) A global feature vector is extracted.
   (b) The classifier decides the presence or absence of the model object in the window based on the feature vector.

The second step is summarized in Figure 2.15.a. Although this scheme can seem simplistic, it gives good results as it allows the utilization of statistical learning techniques to solve the recognition problem: the recognition problem reduces to classifying feature vectors into class and non-class categories. Typically, discriminative classifiers like SVM [HJS09] or AdaBoost [VJ01] are used with this scheme and are trained using bootstrapping, i.e. iteratively adding to the classifier training set windows wrongly classified by the classifier learned in the previous iteration. Because the number of windows to examine in an image is potentially very large, several optimization schemes have been presented (see the next paragraph). Overall, sliding windows remains extremely used for generic class object detection and face detection [ML06, TMF07, HALL05, FG08].

Optimization using cascades

As said above, the sliding window scheme involves the examination of tens of thousands of windows (often even more, see Gu et al. [GLAM09]), which is generally very slow. In order to overcome this limitation, Viola and Jones [VJ01] have first proposed to use a cascaded detection scheme in order to speed up the detection. The insight of cascades is to save as much energy as possible during the detection process: as soon as a negative outcome becomes evident, the computations stop for the current window. Recent approaches complying to this methodology include the works of Vedaldi et al. [VGVZ09] and Harzallah et al. [HJS09]. In this dissertation, we will also make use of cascades although they will only act at the feature extraction level (i.e. we extract "smart" features) and not at the classifier level (Chapter 4).

To draw a parallel of cascades with the human vision, one immediately "knows" which spots of an image to focus on to get a fast understanding of it. This intuition has a lot to do with the pre-processing done automatically by the pre-cognitive system in our brain in order to predict interest areas in the scene. This behavior allows to save body resources by sensing only small parts of the scene with a greater resolution. For instance, flat areas like the sky are of low interest, so almost no time is spent analyzing them. Interestingly enough, the structure of cascaded detection systems is closely related to the human visual system.

The origin of cascades arises from the fact that in a classical sliding window scheme the same amount of computations is spent whether the considered area is plain blue sky or not. Intuitively, one can understand that this approach is far from the optimum computationally speaking and that many time that could be reinvested into more complex tasks is lost. More generally, such an approach becomes dramatically costly for detect-

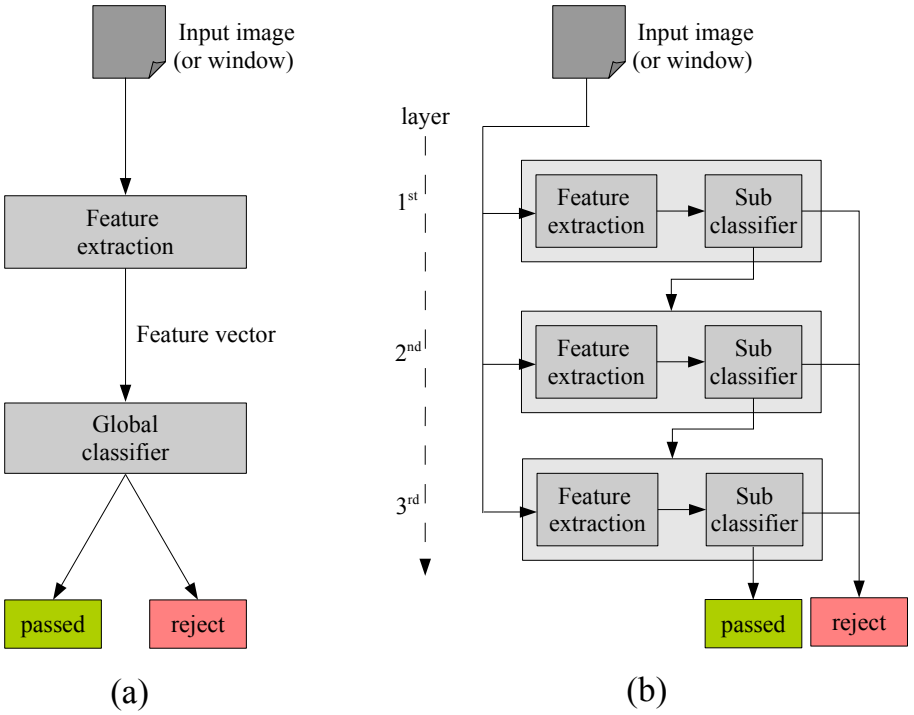


Figure 2.15: Comparison between a classical detection process and a cascaded detection process. (a) a standard process; (b) a cascade with 3 layers. Each layer is activated if the previous layer returns a positive response.

ing more than one type of objects provided that the window aspect ratio, the features or the classifiers used are different.

The cascade framework thus proposes to decompose the recognition process into several successive steps of increasing complexity [VJ01, EHOK01]. The key idea is to enable an ending as early as possible: instead of taking a decision on the full available knowledge, like is done typically in image classification with Support Vector Machine for instance, the global decision function $F : \mathbb{R}^n \rightarrow \{0,1\}$ is fragmented into smaller functions $f_i : \mathbb{R}^{q_i} \rightarrow \{0,1\}$ that are evaluated sequentially:

$$F(\mathbf{x}) \equiv \bigotimes_i f_i(\mathbf{x}_i) \text{ with } \forall i \, q_i < n, \text{ i.e. } \mathbf{x}_i \subset \mathbf{x}$$

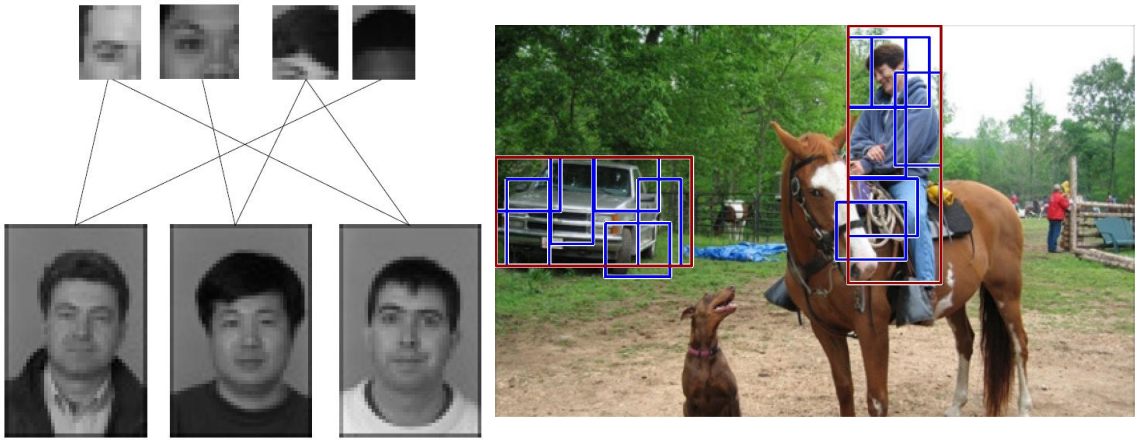
where $\mathbf{x}$ represents the full feature vector for a given window and $\otimes$ is a generic sequence operator that can take various forms. Here, each *subclassifier* f_i is dedicated to clutter detection rather than true positive labelling. A single negative decision thus suffices to abort the rest of the detection process for the current window (see Figure 2.15.b). As long as the vast majority of input vectors are clutters, an admissible hypothesis for real-world object recognition systems [EHOK01, ESPM05], the approach becomes extremely efficient: millions of windows can be examined in a matter of seconds (espe-

cially when associated to a fast feature extraction process like in Viola and Jones [VJ04]). Additionally, one can purposely set up overly simple subclassifiers for the first cascade layers (for instance, a single feature is used in the first subclassifier of [VJ04]), whereas the subclassifiers of the last stages are more complex in order to better represent the ideal decision surface. Some examples of cascaded strategies include the work of Vedaldi et al. [VGVZ09] where a decrease of processing time per image from 27 hours to 67 seconds was reported compared with a brute force approach. Similarly, Felzenszwalb et al. [FGM10] improved the detection time by a factor of 20 using cascades with respect to the same approach using dynamic programming and generalized distance transforms.

Part-based models

A drawback of sliding window is that their global consideration of the sub-image makes them unsuitable to bear occlusion; moreover they are generally not invariant to rotation (in order to save computations). On the other hand, the fact that many class objects can be intuitively decomposed into parts has led to the development of part-based models. Similarly to some previously mentioned methods for specific object recognition, in a part-based model the model object is represented as a collection of parts (each one provided with a corresponding local appearance) along with their spatial configuration. An illustration of such possible decomposition is shown in Figure 2.16 for faces, cars and humans. Part-based models thus belong to the field of structural methods, in contrast to sliding window techniques (although some crossovers have been recently developed). Note that generative classifiers (i.e. Bayesian instead of discriminative) are generally used as classifiers for this class of methods because they can straightforwardly describe the generation of structural models. Contrary to the rigid model for specific objects, the representation in a part-based model is tolerant to class variations both in the part appearance (i.e. appearance variations are handled in the descriptor) and in their spatial configuration (i.e. parts are loosely connected). The insight is that class objects are more different globally than locally: a car and a truck may be globally dissimilar, but they both have rather similar parts (e.g. wheels, headlights, handles) and the spatial arrangement of the parts is only slightly variable.

In the literature, probably the oldest part-based model was developed by Fischler and Elschlager [FE73] for face detection. In their pioneer approach, they considered faces as collection of facial organs connected by spring-like links. More recently, several detectors and descriptors have been proposed to detect and describe the parts (e.g. the Kadir-Bradir detector [ZCY07], descriptor with Gaussian model [FPZ03]). Likewise,



(a) (b)

Figure 2.16: Illustration of part-based models. (a) Decomposition of faces into local rectangular patches [BBU04]; (b) "cars" and "human" recognition results where the positions of the detected parts are highlighted in blue [FGM10].

there exist several options for learning the spatial arrangement of the different parts with respect to each other: usually, the spatial configuration is expressed as a set of pairwise interactions between parts [FPZ03, LHS07, SP05] for which different organizations have been proposed: constellation of parts [LLS04, AAR04], star-shaped models [CFH06, FPZ05, FGMR09], graph-based models [FPZ03, ZC06], hierarchies of parts [EU05, BT05, SP06] etc. In this dissertation we also model the class objects as collections of parts, although we do not explicitly compute the spatial arrangement of parts for the class case.

To conclude with, class object detection schemes can be either based on statistical learning (sliding windows) coupled with discriminative classifiers; or based on structural (i.e. part-based) models coupled with generative Bayesian classifiers (at least this is the general trend). In this dissertation, the originality of our contribution for class object recognition (Chapter 5) is to combine every type of features (sparse, dense and higher-level features) with every above-mentioned detection scheme (sliding windows, cascades and part-based model) altogether in a unified consistent graph-matching framework. Our purpose is to take the best of each schemes: efficiency of statical learning techniques concerning sliding windows, detection speed concerning cascades and smart representation of parts-based models.

Chapter 3

Cascaded Multi-feature Incomplete Graph Matching For 3D Specific Object Recognition

Contents

In this chapter, we present an approach for the recognition of instances of specific 3D objects. The proposed approach builds upon the graph matching framework and enables the joint utilization of different types of local features (namely keypoints, edges and textures) in a unified manner so as to improve robustness. The combination of different feature types, either sparse or dense, is made possible through a cascaded detection scheme. Contrary to standard graph matching methods, we do not convert the test images into finite graphs (i.e. no discretization nor quantization). Instead, we explore the continuous space of graphs in the test image at detection time. For that purpose, we define local kernels compatible with an efficient indexing of the image features in order to enable a fast detection. During training, the mutual information is used to select the most discriminative model subgraphs; then at detection time those ones are detected using a cascaded process.

This work has been partially published in the International Conference for Pattern Recognition (IEEE ICPR 2010) [RLAB10].

(a) Lowe's Method (b) The proposed method

Figure 3.1: (a) Recognition failure of a keypoint-based method on an image with motion blur because the SIFT keypoint detector performs poorly in these conditions; (b) on the contrary, the method presented in this chapter is able to properly detect the object thanks to the utilization of dense image features.

3.1 Introduction and Motivations

To our knowledge, almost every method for specific object recognition from the state-of-the-art is based solely on keypoint features (see Chapter 2). On one hand, it is certain that keypoints enable an elegant and convenient handling of the problem thanks to their ability to accurate extraction and pairwise matching. On the other hand, they perform poorly on textureless objects because keypoint detectors tend to find salient points only inside well-textured regions (see Section 2.2.2). Indeed our experiments on a home-made dataset have highlighted the fact that the repeatability of keypoints can highly deteriorate in noisy conditions of use. In particular, we noticed that in an indoor environment, using a low quality camera suffices to significantly degrade the good performance of keypoint-based methods (see an example of this in Figure 3.1). As most practical utilizations of specific object detection concern embedded systems equipped with low-quality video cameras, we believe it to be a serious matter.

In the state-of-the-art, exceptions are the method of Ferrari et al. [FTG06] and Kushal and Ponce [KP06] which in addition to keypoints also use densely sampled patches so as to improve robustness and enable a precise segmentation of the retrieved instances. As a result, their methods are extremely robust to large occlusions and distortions. In fact, using different types of local features is known to enhance the detection performance for various tasks (e.g. in image classification or class object recognition [LZL*05, MHK06, MSHvdW07, GN09, GLAM09]). Different feature types can indeed complement each others well by describing different aspects of the image (texture, edges, colors, etc.). For instance, edge features have been widely used for specific object de-

tection as well (see [Jur01, KHP07, DPP08, HHIN09]), so it might probably be a good idea to use them in our system. Moreover, it has been demonstrated that using dense features like HOG [DT05] (see Section 2.2.3) or densely sampled patches [JT05, FTG06] was a good idea for various applications. The interest of using dense features in our case is to rely on other feature sources than saliency-based detectors: as said earlier, those detectors (e.g. the SIFT detector as mentioned above) experience difficulties when they deal with blurred images or important scale changes. The flip side of the coin is that systems which use dense features are often extremely slow (e.g. the system of Ferrari mentioned earlier). Moreover, it is a delicate problem to combine different types of features (especially sparse and dense features) in a same framework. To solve both issues, we turned ourselves toward the graph matching framework coupled with cascades.

3.1.1 The feature combination problem

As pointed out above, we show in this chapter how a cascade-oriented graph matching framework can help to solve the delicate problem of combining together heterogeneous types of features (i.e. sparse and dense features). Generally speaking, this is not a trivial matter in computer vision and especially in the object recognition field. It often raises several well-known issues, such as the normalization problem, the increase of computational complexity due to feature extractions and the inherent difficulties to combine sparse and dense types of features.

Normalization issues Different types of features involve different ranges of values and it generally gets bothersome when such heterogeneous values are gathered in a same feature vector. In the literature, normalization is generally achieved by assuming that each component of the global feature vector follows a Gaussian distribution (meaning, subtracting the mean and dividing by the standard deviation) or a χ^2 distribution in the case of histograms [VGVZ09]. In practice however, such hypotheses are not always realistic. Recent works on Multiple Kernel Learning (MKL) have contributed to partially solve some of these issues (combining heterogeneous types by using a linear combination of dedicated kernels), but the results can still be disappointing compared to a simple averaging for instance [GN09, VGVZ09].

A first benefit in a cascade-oriented framework is that in a cascade, the different subclassifiers $\{f_i\}$ use different subsets φ_i of the whole feature set φ: $f_i : \varphi_i \rightarrow \{0, 1\}$ (see Section 2.4.2). Assuming that each φ_i only contains scalars picked out from a particular feature type, each decision function then combines comparable features, which shrugs off most of the problem. The method presented in this chapter is among those ones.

Namely, dense textures, sparse keypoints and semi-sparse edges are used separately in the subclassifiers.

Computational issues Feature extraction is a time-consuming process which can even become a bottleneck in a standard object detection application. For instance, Vedaldi et al. [VGVZ09] have evaluated that, in the perspective of a classical approach (see Figure 2.15.(a)), just computing the feature vector for all possible windows is prohibitively slow. On the contrary, a cascade-oriented framework offers efficient ways to reduce the computational burden. As the decisions are taken temporally (i.e. one after the others), it becomes possible to prune every unnecessary feature extraction work. Ideally, a cascaded system is expected to extract the features at run-time, i.e., just before they are required for evaluation by the subclassifier (see Figure 2.15.(b)). This way, only a few spots in the image get closely examined, saving important amounts of computational power as demonstrated by Felzenszwalb et al. [FGM10] for instance.

In particular, it can be interesting to limit the number of feature types used in the first cascade layers (i.e. the part of the cascade which is evaluated the most frequently). Since feature types are generally independent, each type requires its own machinery to be extracted from the image. By retaining a subset or even a single feature type to feed the subclassifier of the first layer, the time spent to extract all the other types will be saved. Such a strategy was used independently by Harzallah et al. [HJS09] and Vedaldi et al. [VGVZ09]. In the first case, the feature type that is the least expensive to compute (namely, HOG features optimized using integral images) was used alone by the first level subclassifier without significant loss of performance compared to using all types. In the second case, a jumping window technique relying again on a single feature type (namely SIFT keypoints) was used to generate candidate windows sent to the second cascade layer. Our method strongly relates to this latter work.

Heterogeneity issues The last problem concerns the combination of feature types differently stored in terms of data structures: sparse features are stored in lists of variable length whereas dense features are stored in vectors of fixed length. Because machine learning techniques generally prefer to deal with fixed-length vectors, sparse features have to undergo some preprocessing (typically they are quantized before an histogram binning [CDF*04], see Section 2.2.3). This process is not always desirable as it makes lose some valuable information contained in the sparse features (for instance, the keypoint positions).

In our cascade-oriented framework, we adopt instead an approach where sparse

and global features are not directly used together. The local kernels that we define in Section 3.3 are specific to each feature type and perform sparse-to-sparse or dense-to-dense comparisons separately. For instance, the kernel associated to a dense feature type operates a local search in the dense feature space in order to find optimal local matching.

3.1.2 Outlines of the proposed method

To summarize, the proposed algorithm takes as input a collection of model images (the model position in each image is supposed to be known) as well as a collection of non-class (background) images. It automatically extracts a large collection of local features of various types from the model images (Section 3.3). Then, each training image is converted into a *prototype graph* by considering features as graph nodes and connecting neighboring nodes in the scale-space (Section 3.4.1). The last step of the training procedure consists of building a *detection lattice* from a selection of the most discriminative subgraphs from the prototype graphs. The lattice is composed of cascaded micro-classifiers aiming at successively recognizing neighboring model features in a region growing scheme.

During the recognition stage, graph matching is efficiently performed based on an iterative scheme which picks one scene keypoint each time and feeds it to the detection lattice to initiate the search of a model part around the keypoint location. The detection lattice thus checks the area surrounding the input keypoint, searching for features consistent with the model graph. Since we focus on realistic object recognition, we tackle the occlusion problem by considering the recognition to be successful when a sufficiently big subgraph of the prototype graph is discovered in the test image (Section 3.4.2). This is optimally done by computing during training the posterior probability of finding the whole model given a model subgraph. We also introduce a new texture descriptor, both descriptive and fast to compute in section 3.3.3 which highly contributes to explain our good results (see Chapter 4).

The different parts of our method are represented in Figure 3.2 and detailed in the Section 3.4. The lattice construction procedure is detailed in Section 3.5. Finally, Section 3.6 concludes.

3.1.3 Related works

There are several related works with respect to ours in the state-of-the-art. To begin with, the approaches that we found to be the closest to ours belong to the field of part-based

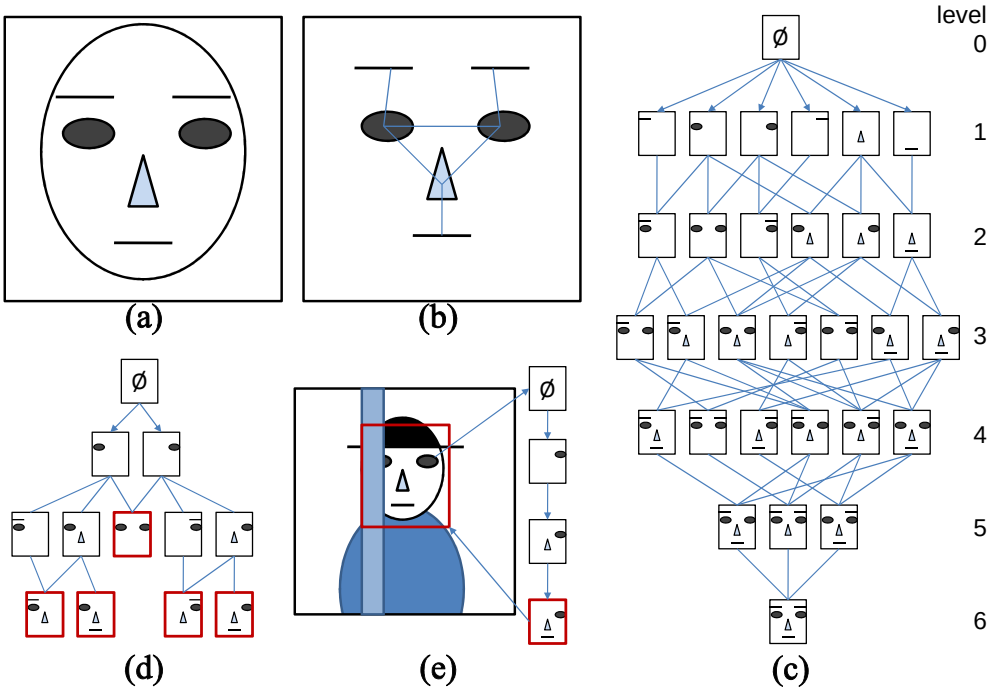


Figure 3.2: Summary of the method presented in this chapter. (a) A set of model images (i.e. the model is a face, here only one image shown). Local features are extracted using either a detector (i.e. keypoints) or densely sampled. For simplification, they are represented using lines (for edges), ellipses (for keypoints) and triangles (for textures), i.e. 3 feature types (only a small number of them is drawn for clarity). (b) Construction of a *prototype graph* for each model image from the local features; (c) Complete *detection lattice* for the prototype graph shown in (b). The lattice contains cascades of micro-classifiers aiming at detecting the prototype graph by checking local feature one by one in any possible order; (d) Pruned detection lattice: it now aims at detecting subgraphs (red squares) of the prototype graphs; (e) example of recognition from a randomly picked scene keypoint (top blue arrow): the keypoint is fed into the lattice, each lattice path is evaluated, a successful path is found leading to a model subgraph (small red square), and a vote is cast in the test image (large red square).

methods for class object recognition. As described in Chapter 2, this family of methods considers the recognition problem under a structural viewpoint: the model object is viewed as a graph in which the different parts corresponds to graph nodes, which are connected by pairwise spatial interactions.

The method of Zhu et al. [ZCY07], for instance, considers triplets of keypoints as basic features and learns grammar rules based on "AND" and "OR" operations to detect the objects. The resulting grammar resembles our lattice in the fact that it can be viewed as a mixture of trees, allowing the detection of different model subgraphs using inference with Markov Random Fields. Contrary to us however, neither cascades nor dense features are used; moreover the tree shape is learned using an EM algorithm while in our case we rely on mutual information to build our lattice. Finally, their method can only afford to extract a small number of features per image (i.e. graphs nodes) and has to limit the number of edges (for instance, they constrain edges to not cross) so that the run-time complexity does not explode. Similar shortcomings also hold for the method of Fergus et al. [FPZ03] and Zhang et al. [ZBMM06]. More generally, the feature types and decision schemes used in the case of class object detection are more complex than for our specific object detection approach. Traditional cascades are indeed built using high level classifiers (e.g. AdaBoost in [VJ01, FG08]), each of them handling hundreds of features. In our case, each classifier is extremely simple as it takes its decision from a single feature. Although there exists one part-based approach by Felzenszwalb [FH05] that uses simple texture features similar to ours, our system differs from this one in the matching scheme essentially different, and also the fact that our parts are not manually landmarked before training. On the contrary, our parts are automatically gathered based on the mutual information that they provide to the model. This latter part selection scheme is similar to the one initiated by Vidal-Naquet and Ullman [VNU03] and continued by Epshtein and Ullman [EU07] except that in our case the list of parts is not explicitly computed *before* training; instead, the appearance and geometrical models are learned at the same time to select the most discriminant parts with respect to their close neighbors in the model images. Finally, the recent part-based approach of Felzenszwalb et al. [FGM10] uses cascades like us to speed up the detection, but here again, the features used, the matching scheme and the invariance set are different.

In the field of specific object detection, the methods of Lazebnik et al. [LSP04] and Ferrari et al. [FTG06] are probably the most similar to ours. In the first case [LSP04], model parts constituted by connected keypoints are learned from training images; and then they are detected in test images using a region growing scheme with a pruning based on the spatial arrangement and descriptor consistency of the features matched.

Again a single type of salient feature is used (i.e. affine keypoints). Moreover, the pruning thresholds for checking spatial consistency at growing time are defined manually whereas in our approach, all thresholds are learned automatically. In the second case [FTG06], the recognition of specific instances is performed in two steps: first keypoint matches are used to raise hypothesis, and then dense features are extracted to verify the hypothesis with an iterative expansion/contraction process seeking to discover the entire visible object surface. Likewise, we also use keypoints as a first step for recognition and other features (in particular dense features) for growing regions, but this latter process is much faster thanks to the utilization of cascades. Moreover, we do not seek to entirely detect the instance surface, instead we only try to detect parts which are distinctive enough (our expansion process stops as soon as the distinctiveness is sufficient according to a prior training). Finally, the method of Moreels et al. [MP08] is essentially different to ours, although the title of their paper may suggest it as they also rely on a cascade-oriented framework for specific object recognition. Contrary to us, the system of Moreels et al. [MP08] only uses a single type of feature (i.e. keypoint). Furthermore, their cascade consists of a succession of several rigid recognition schemes (i.e. Hough transform followed by RANSAC), which is completely different in the principle and in practice from our incomplete graph matching strategy.

3.2 Useful notation

Pre-emptively to the following of this chapter, we introduce for clarity the table of useful symbols in Table 3.1.

3.3 Used Features

At the bottom of the recognition process, low-level features are used to locally describe the model object. In order to get a recognition system as fast as possible, we selected a subset of three complementary feature types prone to fast extraction:

- Keypoints, denoted by φ_K.
- Edges, denoted by φ_E.
- Textures, denoted by φ_T.

For each of these three types, we outline below its properties and define a **kernel** function $K_t : \varphi_t \times \varphi_t \rightarrow \mathbb{R}$. We refer to this kernel as a *local kernel* as it takes into account

symbol	meaning
$\mathcal{O}$	the model object
$\mathcal{I}$	an image
$\mathbb{I}$	a list of images $\mathbb{I}$.
$\mathbb{I}^+$	the list of positive training images (i.e. model views).
$\mathbf{c} = (x, y)$	2D center
σ	scale. By convention we use radius = scale.
θ	orientation ($\theta \in [-\pi, \pi]$)
$\mathbf{h} = (\sigma\cos\theta, \sigma\sin\theta)$	radial vector.
$\mathbf{p} = (\mathbf{c}, \mathbf{h})$	position = center and radial vector.
φ_t	type of local feature (e.g. SIFT keypoint)
$\mathbf{z}$	descriptor (e.g. 128-dimensional vector for SIFT).
$\phi^t = (\mathbf{p}, \mathbf{z})$	local feature of type φ_t = a position and a descriptor.
$K_t : \varphi_t \times \varphi_t \to \mathbb{R}$	kernel of type φ_t.
$A = \{\phi\}$	model aggregate (collection of connected local features).
$A' = \{\phi'\}$	detected aggregate (collection of detected local features).
$e_{ij} = (A_i \to A_j)$	lattice branch connecting aggregate A_i to aggregate A_j.
d_{ij}^{max}	threshold of the micro-classifier associated to e_{ij}.

Table 3.1: Useful symbols.

both the positions $\mathbf{p} = (\mathbf{c}, \mathbf{h})$ of the two features (respectively, their center $\mathbf{c}$ and their radial vector $\mathbf{h}$) and their descriptor $\mathbf{z}$, in contrast with standard kernels as in Multiple Kernel Learning (see [GN09]) which act at a global scale. The kernel output is some sort of distance between the two features and is used later in the recognition process to check the presence in an image of a specified local feature (see Subsection 3.4.2).

3.3.1 Keypoints

We use SIFT keypoints for their good propensity to specific object recognition [Low04, MP07]. In our system, the SIFT detector acts as a saliency detector, and only salient regions are further analyzed. In other words, the search of the model object always starts from SIFT keypoints. In order to overcome the robustness issues mentionned in the introduction, we use an absolute distance between SIFT descriptor (i.e. the noise is thus seen as constant and additive) instead of the traditional distance ratio between the first and second best neighbors.

Formally, each image keypoint $\phi^K \in \varphi_K$ is defined by a center $\mathbf{c} = (x, y)$, a radial vector $\mathbf{h} = (\sigma\cos\theta, \sigma\sin\theta)$ (where σ is the patch radius and θ its orientation), and a descriptor $\mathbf{z}$ of 128 dimensions. We define two kernels for this feature type:

- The first kernel is a standard comparison between descriptors:

$$K_K^z(\phi_i^K, \phi_j^K) = \left\|\mathbf{z}_j - \mathbf{z}_i\right\| \tag{3.1}$$

- The second kernel is a spatial distance between two "compatible" keypoints ϕ_i^K and ϕ_j^K:

$$K_K(\phi_i^K, \phi_j^K) = \begin{cases} \left\|\mathbf{c}_j - \mathbf{c}_i\right\|^2 + \alpha_K^2 \left\|\mathbf{h}_j - \mathbf{h}_i\right\|^2 & \text{if } K_K^z(\phi_i^K, \phi_j^K) \leq \zeta_K, \\ \infty & \text{otherwise.} \end{cases} \tag{3.2}$$

  where ζ_K is a threshold that specifies the acceptable amount of noise for a SIFT descriptor (see §4.2.2). Since the system will need in the following to quickly compare a given keypoint versus all keypoints present in the test image, we index the scene keypoints in a k-d tree. Contrary to [Low04], this indexing is based on the keypoint position rather than on its descriptor.

3.3.2 Edges

We use the Canny edge detector [Can86] followed by a step of polygonization to obtain a bunch of line segments. A line segment $\phi^E \in \varphi_E$ is only defined by its center and its radial vector (no descriptor) such that the boundaries of the segments are $\mathbf{c} + \mathbf{h}$ and $\mathbf{c} - \mathbf{h}$. The local kernel K_E between two edges ϕ_i^E and ϕ_j^E is the maximum of the minimum distance between each pair of pixels lying on both segments:

$$K_E(\phi_i^E, \phi_j^E) = \begin{cases} \max\limits_{p\in[-1,1]} \min\limits_{q\in[-1,1]} \left\|(\mathbf{c}_j + p\mathbf{h}_i) - (\mathbf{c}_j + q\mathbf{h}_i)\right\| & \text{if } \left|\theta_j - \theta_i\right| \leq \zeta_E, \\ \infty & \text{otherwise.} \end{cases} \tag{3.3}$$

(since no visual descriptor comes with a line segment, we simply check the orientation). Again, we reduce the search time of a given line segment against all existing segments in the test image using 6 distance maps (i.e. we use 6 orientation bins, so that $\zeta_E = 30°$, and each distance map only considers the edges which are roughly oriented in the map orientation). This technique is robust to a noisy polygonization since the distance does not vary much if the existing line undergoes cuts or oversize. Moreover, it enables to quickly create at run-time new segments superimposed on existing ones but having different boundary locations, in order to fit in the best possible way the position of a query line segment. An example of a fitting between a request line segment and all the edges contained in a sample image is illustrated in Figure 3.3. A new line segment is created at run-time according to the projection of the request line segment onto the nearest image edges having similar orientations (Figure 3.3.d). Thanks to this operation, the set of existing segment features is virtually infinite. One can thus think of edge features as semi-sparse features in the sense that they can adapt to the request (within

some limits). This behavior would be clearly impossible to implement in a classical graph matching application where the set of features is finite and well defined *before* proceeding to the matching.

3.3.3 Textures

We derive a new texture descriptor from the work of Tola et al. about the DAISY feature [TLF08]. Since textures are dense features, they exist for every pixel of the image scale-space. In our case, the descriptor of a texture feature $\phi^T \in \varphi_T$ located at $\mathbf{p} = (x, y)$ and $\mathbf{h} = (\sigma\cos\theta, \sigma\sin\theta)$ is defined as the concatenation of three sub-descriptors extracted at the same position $\mathbf{p}$ but at three different scales $\{\sigma/1.54, \sigma, 1.26\sigma\}$ (we followed a similar definition by Kruizinga and Petkov [KP99]). Each sub-descriptor is an 8-bins histogram of oriented gradient extracted at the corresponding position. The local kernel is simply defined as the Euclidean distance between the two descriptors, provided that the two locations are not too far away in the scale-space:

$$K_T(\phi_i^T, \phi_j^T) = \begin{cases} \left\|\mathbf{z}_i - \mathbf{z}_j\right\| & \text{if } \left\|\mathbf{c}_i - \mathbf{c}_j\right\|^2 + \alpha_T^2 \left\|\mathbf{h}_i - \mathbf{h}_j\right\|^2 \leq \zeta_T, \\ \infty & \text{otherwise.} \end{cases} \tag{3.4}$$

As in the original paper of Tola et al. [TLF08], we precomputed eight gradient maps (one for each orientation) at the finest scale and spanned the rest of the scale-space with a pyramid of Gaussian to enable a fast descriptor extraction.

Finally, we introduce here the shorthanded notation

$$\min_{\mathcal{I}} K_t(\phi_i^t) \equiv \min_{\phi_j^t \in \mathcal{I}(\varphi_t)} K_t(\phi_i^t, \phi_j^t)$$

in which a given request feature ϕ_i^t is compared to all image features of the same type t, and the minimum distance is returned (hence the notation of the minimum of K_t on the whole image $\mathcal{I}$). Thanks to the indexing of each feature type, this comparison is extremely fast and is a key component for our system.

3.4 Algorithm Description

Now that the feature types used in our approach have been presented, we describe in this section the core of our approach. Our method for the detection of specific object instances is as follows: first, we assume that a few images of the model object $\mathcal{O}$ are pro-

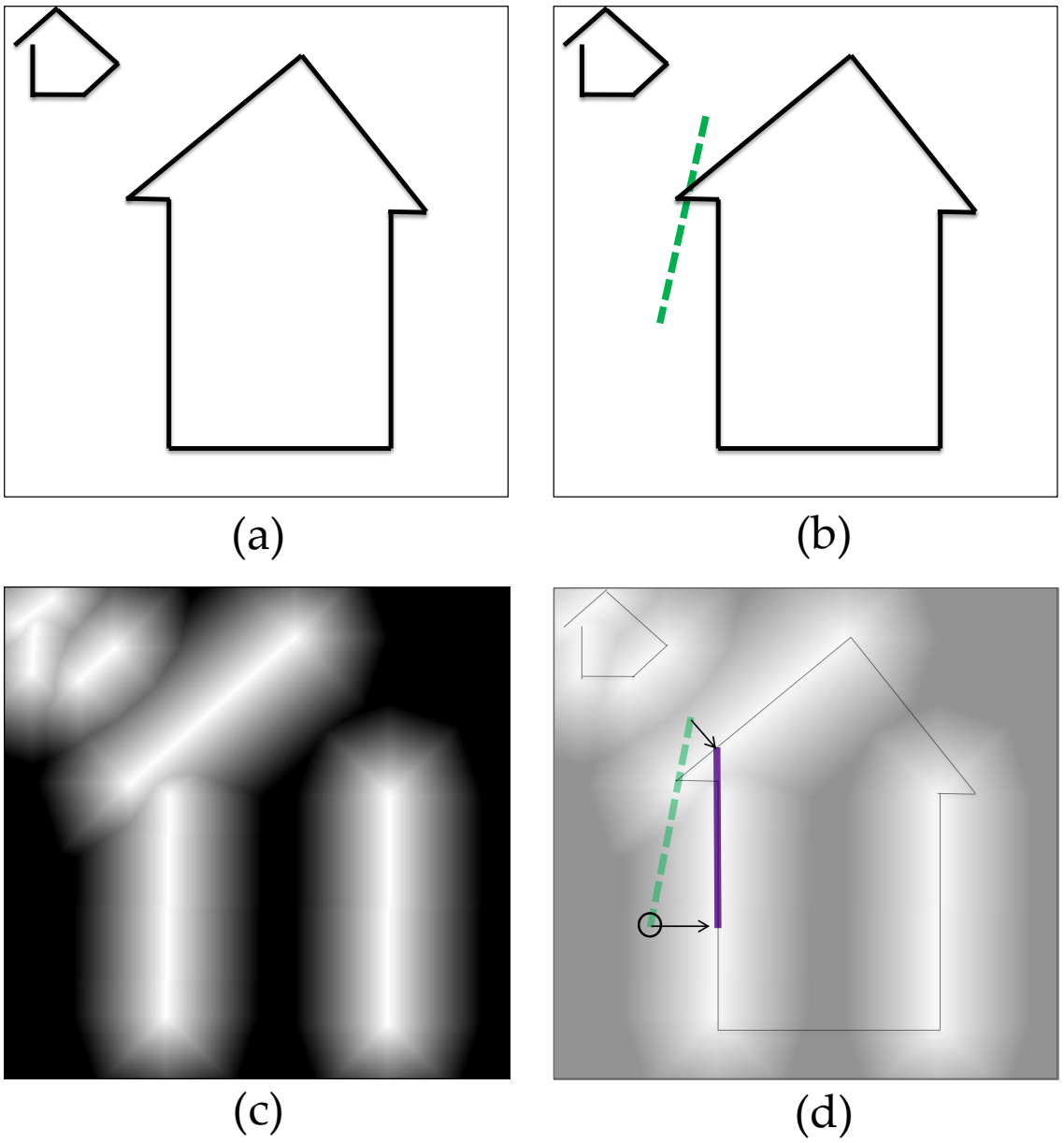


Figure 3.3: (a) Sample image. (b) Request line segment (dashed green line). (c) Distance map corresponding to the closest orientation bin $\theta = 75° \pm 30°$. (d) Projection of the request line segment onto the nearest existing edges using the distance map (the resulting line segment is shown as a bold line). The circle indicates the point of maximum distance between the request segment and any existing image edges of the same orientation. This distance is returned by the kernel K_E and corresponds to the fitting error.

vided (i.e. the object is viewed under different viewpoints and/or lighting conditions) (in Figure 3.2.a, only one simplified training image is shown for clarity). Theoretically, the more numerous the training pictures are, the better the recognition will be (redundancy between them does not pose a problem). Then, a prototype graph is extracted from each model image. The prototype graphs are constituted of local features (either extracted using a detector or densely sampled) which are connected when close in the scale-space (Figure 3.2.b). A detection lattice is constructed from those prototype graphs: its aim is to recognize the prototype graphs (i.e. the model views) using a region growing scheme. It is composed of cascaded micro-classifiers that verify the presence of local features one by one (Figure 3.2.c). Because the complete lattice has an exponential size, only a small part is used so as to enable a fast detection (Figure 3.2.d). Finally, the lattice is used to detect objects in test images (Figure 3.2.e). The resulting system is robust to occlusion and has a low computational complexity.

3.4.1 The prototype graphs

First, we extract a prototype graph $\mathcal{G}_n$ from each model image $\mathcal{I}_n \in \mathbb{I}^+$ of the model object $\mathcal{O}$. The aim of this step is to transform the input model pictures from matrices of pixels into structured objects. This is necessary as our method belongs to the family of structural methods, i.e. methods that decompose the model object into a finite set of "parts" (although the term "part" may not be well-chosen in our case, as the parts that we extract do not necessarily refer to semantic parts). In our case, the definition of a part is just a connected subset of graph nodes, and this decomposition is redundant (parts can overlap).

The procedure for converting a model image $\mathcal{I}_n$ into a prototype graph $\mathcal{G}_n$ is the following:

1. Firstly, the picture is aligned in a reference frame $\mathbf{p}_{ref}$ using a similarity transform (i.e. normalizing coordinates in $[-1, 1]$).

2. Secondly, local features of each type are extracted from the image. For keypoints and line segments, the SIFT detector and the Canny edge detector are used. For textures, we sample them densely and uniformly. The aim here is to cover the image with a large number of local features, each of them constituting a weak classifier potentially selected later in the detection lattice (in the same spirit as the work of Viola and Jones [VJ01]).

3. Each local feature becomes a graph node with center $\mathbf{c}$, scale σ and angle θ (and

$\mathbf{h} = (\sigma \cos\theta, \sigma \sin\theta)$).

4. Two nodes are linked if their distance in the scale-space is small enough. Typically, we use the following criteria for connecting two nodes with centers $\mathbf{c}_i$, $\mathbf{c}_j$ and scale σ_i, σ_j:

$$0.5 < \frac{\sigma_i}{\sigma_j} < 2 \quad \text{and} \quad \left\|\mathbf{c}_i - \mathbf{c}_j\right\|^2 < \sigma_i \sigma_j \tag{3.5}$$

The first inequality constrains the two nodes to have about the same scale (up to a factor 2), while the second one ensures their centers are not too far apart in the scale-space. Hence, each graph edge ideally stands for a stable neighborhood relationship as we assume the correlation between two model features to decrease when their distance augments. Note finally that two features (i.e. nodes) with different types can be connected, in fact the graph nodes are linked regardless of the feature types.

This procedure is repeated for every model image $\mathcal{I}_n \in \mathbb{I}^+$. Note that the construction of the prototype graphs is fully automatic; Figure 3.2.(b) presents an illustration of a simplified prototype graph (a realistic graph would be too complex to be displayed here as it typically contains thousands of features and edges). In the following, a detection lattice will be straightly derived from the prototype graphs in order to recognize subgraphs (i.e. model parts) in a cascaded manner (problem also known as *incomplete graph matching*).

The prototype graph $\mathcal{G}$ For convenience, we also define a unified prototype graph $\mathcal{G} = \bigcup \mathcal{G}_n$. So in the following when we speak of "the prototype graph" in the singular, we mean the gathering of all prototype graphs in a single graph (note that $\mathcal{G}$ has $|\mathbb{I}^+|$ connected components, one for each model image).

3.4.2 The detection lattice

Concerning the detection, we use a sort of degenerate tree formally called a "lattice". Mathematically, a lattice $\mathcal{L}$ is a set with a partial order relation between elements. A simple example of lattice is shown in Figure 3.4. Because of this order relation, a lattice resembles a tree except the fact that there can be more than one path between two nodes, but still excluding cycles (edges are oriented). The depth of a node is defined as in a tree, i.e. by the number of edges from which it is separated from the root, and all nodes with the same depth constitutes a *level*. [1]

[1] There exists another definition of a lattice – a squared grid – but this has nothing to do with our method.

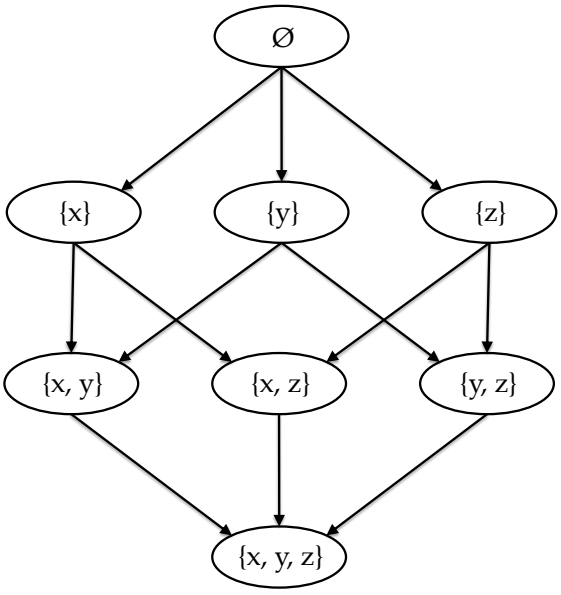


Figure 3.4: Hasse diagram of a lattice formed from the set $\{x, y, z\}$ with subset inclusion as order relation (4 levels).

In our framework, the detection lattice $\mathcal{L}$ stores some possible ways of building the prototype graph by adding nodes one by one. More precisely, each lattice node represents a *connected* subgraph of the prototype graph $\mathcal{G}$, and each branch between two lattice nodes represents an atomic addition of a *single* prototype graph node (i.e. a feature) from the first subgraph to the second one. The order relation between elements is thus the subset inclusion.

Aggregates For clarity, we denote in the following the lattice nodes by the term "**aggregate**". An aggregate is a connected subgraph of the prototype graph $\mathcal{G}$, and as said earlier it corresponds to a *model part* in our approach. Depending on the number of atomic features composing it, the part will be smaller or larger (but is is interesting to note that contrary to [LSP04] all features contained in an aggregate will be close in the scale-space due to the connection constraint of (3.5)). To sum up, each level l of the lattice contains aggregates of cardinality equal to l. For example, the root level ($l = 0$) contains only one empty aggregate which stands for the empty graph, level 1 is composed of aggregates containing single features, and so on. An illustration is given in Figure 3.2.c.

Obviously, the cardinality of each level grows exponentially with l. For this reason, it is not tractable to compute the entire lattice. Fortunately, storing it entirely is useless for our purpose and in the following we will confine ourselves to using an incomplete lattice (see Figure 3.2.d).

3.4.3 Aggregate position

As we saw above, the purpose of the recognition step is to discover model parts, i.e. aggregates, in the test image. In other words, we want to detect in the test image groups of features that are consistent with a model aggregate in terms of feature descriptors and spatial arrangement. Concerning the latter point, we use a 2D similarity transform to align two aggregates, so we need to define a unique center and a radial vector for an aggregate from its features. The purpose here is that two matching aggregates have roughly the same centers and radial vectors up to some noise. The averaging of the spatial positions of the composing features appears to be a valid choice, as noise is in random directions and hence canceled by the averaging. Moreover, the result of the averaging is independent of the order in which the features are added in the aggregate. This is important as an aggregate can be constructed from different lattice paths, i.e. by adding its composing features in different orders, see Figure 3.2.d for an illustration.

Formally, let an aggregate A contain a set of l features, then its center $\mathbf{c}_A$ and radial vector $\mathbf{h}_A$ are defined as:

$$\mathbf{c}_A = \frac{1}{l} \sum_{n=1}^{l} \mathbf{c}_n \tag{3.6}$$

$$\mathbf{h}_A = \frac{\sum_{n=1}^{l} \mathbf{h}_n}{\left\| \sum_{n=1}^{l} \mathbf{h}_n \right\|} \sqrt{\frac{1}{l} \left(\sum_{n=1}^{l} \|\mathbf{c}_n\|^2 + \|\mathbf{h}_n\|^2 \right) - \|\mathbf{c}_A\|^2} \tag{3.7}$$

Formulas (3.6) and (3.7) represent the averaged center and the average orientation normalized by the standard deviation around the global center and the composing centers, respectively. Those formulas experimentally showed up to give stable results even in case of important deformations and can deal with slight 3D viewpoint changes (see Figure 4.14). Moreover $\mathbf{c}_A$ and $\mathbf{h}_A$ can be computed in constant time using the center and radial vector of A's father, at the cost of updating a few hidden variables, making the aggregate growth an efficient operation.

3.4.4 Aggregate recognition

We now explain the recognition process assuming first that the detection lattice $\mathcal{L}$ is already constructed. As mentioned earlier, the purpose of the recognition step is to discover model aggregates (i.e. model parts) in the test image, and then to draw a final decision about the presence of the model object $\mathcal{O}$ from those partial detections.

To begin with, aggregates are detected in the test images using an expansion process where features are added one at a time. The expansion process is dictated by the lattice

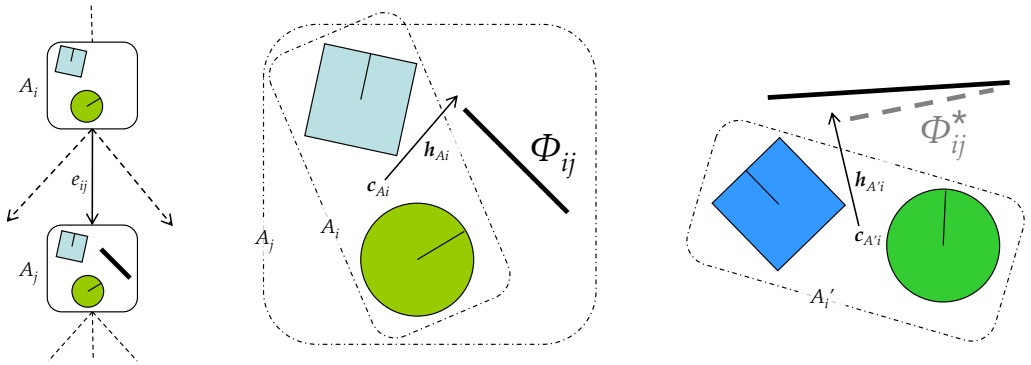
shape: every discovered aggregate corresponds to a lattice node (i.e. a model aggregate) and attempts to go further in the lattice, i.e. to grow according to the available lattice branches starting from the lattice node. For this purpose, we use cascades in order to enable a fast detection of the aggregates. Practically, a classifier is associated to each lattice branch in order to take a decision about letting the aggregate grows or not. Namely, each classifier is responsible for ensuring that the feature associated to its lattice branch is also present in the test image, relatively to the aggregate position. In contrast with the work of Viola and Jones [VJ01], our classifiers are extremely simple as they simply consist of single decision stumps. For this reason, we denote them as "**micro-classifiers**" in the following. A corollary is that an aggregate can be seen as a *weak classifier*: when it is detected in the test image, it indicates that a model part is probably present, but more aggregates are required to ensure the detection of the model object.

Formally, the aggregate detection procedure is as follows: first, a loop iteratively picks a random *seed* feature in the test image. A **seed feature** is a salient feature which acts as an entry point in the image for the search of model aggregates, similarly to the jumping windows of Chum and Zisserman [CZ07]. In our case, we use SIFT keypoints to initiate the search. The picked feature is then fed to all **seed branches**, i.e. lattice branches starting from the root. If the associated micro-classifier of each seed branch returns a positive response, then the branch is traversed. Likewise, all children branches are recursively checked and traversed in case of positive responses of the associated micro-classifiers. When the seed feature truly belongs to a learned part visible in the test image, then at least one terminal aggregate will be detected at that position (we mean by **terminal aggregate** an aggregate which has no children). An example of this discovery process is shown in Figure 3.2.e. In the following, we distinguish the notation between a *model* aggregate A, which contains original model features, and a *detected* aggregate A' which is constituted of similar features detected in the test image.

More formally, let A_i be a model aggregate containing l features and A'_i the corresponding detected aggregate found in the test image (thus it also contains l matched features). The micro-classifier condition for reaching level $l+1$ through the branch $e_{ij} = (A_i \rightarrow A_j)$ (i.e. connecting aggregate A_i (l features) to aggregate A_j ($l+1$ features) $\mathcal{L}$), which corresponds to the addition of the model feature ϕ^t_{ij} of type t, is:

$$d_{ij} = \min_{\mathcal{I}} K_t(\phi^{t*}_{ij}) \leq d^{max}_{ij} \tag{3.8}$$

where $\phi^*_{ij} = \left(\mathbf{p}^*_{ij}, \mathbf{z}_{ij}\right)$ is the predicted model feature in the test image (i.e. same descriptor but different position), d_{ij} the kernel distance between ϕ^*_{ij} and what can actually be



Model aggregates A_i and A_j (Model image)

Detected aggregate A_i' (Test image)

Figure 3.5: Predicted position of the edge feature ϕ^*_{ij} in the test image (dashed line) with respect to the model aggregate A_i and the two features already detected (a square and a circle for simplification). In our approach, this is done using a 2D similarity. As can be seen, the test image contains an edge near the predicted position (strong black line), meaning that the micro-classifier of the edge $e_{ij} = (A_i \rightarrow A_j)$ will most likely output a positive decision (i.e. $d_{ij} < d^{max}_{ij}$).

found in the test image, and d^{max}_{ij} is a constant learned during training (see section 3.5). Here, $\mathbf{p}^*_{ij}$ is the predicted position of ϕ^t_{ij} relatively to the position of A'_i, which in our case is obtained by a simple 2D similarity:

$$\mathbf{p}^*_{ij} = \text{sim2D}(\mathbf{p}_{ij}|A_i, A'_i) = (\mathbf{R}\mathbf{c}_{ij} + \mathbf{t}, \mathbf{R}\mathbf{h}_{ij})$$

with $\mathbf{R}$ a 2×2 matrix and $\mathbf{t}$ a translation vector defined such that

$$\begin{cases} \mathbf{c}_{A'_i} = & \mathbf{R}\mathbf{c}_{A_i} + \mathbf{t} \\ \mathbf{h}_{A'_i} = & \mathbf{R}\mathbf{h}_{A_i} \end{cases}.$$

An illustration of this growing process is given in Figure 3.5: here, the aggregate A_i composed of two features has been already detected in the test image (A'_i). The next step is to reach children aggregate A_j through edge e_{ij} which corresponds to the addition of an edge feature ϕ_{ij} (strong solid line in the left figure). As a result, the ideal position of ϕ_{ij} is computed in the test image with respect to A'_i, resulting in ϕ^*_{ij} (gray dashed line). The local kernel $\min_{\mathcal{I}} K_E(\phi^*_{ij})$ is evaluated and since an edge is also present in the test image at approximately the same position, the kernel returns a distance inferior to d^{max}_{ij}. In other words, the branch e_{ij} is traversed. The process then repeats for subsequent children aggregates (not shown).

Although the features are of different types, the fact that they are added one at a time enables to bypass the problem of combining different feature types together. If a feature

is not found (i.e. $d_e > d_e^{max}$), the progression in the path is simply abandoned. When a terminal aggregate A_{term} is reached, a position hypothesis is cast for the model presence at sim2D($\mathbf{p}_{ref}|A_{term}, A'_{term}$). The pseudo-code for the aggregate detection procedure is given in Algorithm 3.1. Finally, hypothesis are clustered in the oriented scale space using a greedy algorithm (Section 3.4.5) and a probability formula is applied to weight each cluster (Section 3.4.6).

Continuous graph matching

It should be noted that new test image features are computed at each micro-classifier evaluation (those features are composing the detected aggregates). This comes from the formulation of the decision function (eq. (3.8)) which asks for a minimum over all possible test image features. Since this latter set is almost infinite (for dense feature at least) and thus intractable to compute, instead only a few features are extracted at plausible positions and compared. For instance, the texture micro-classifier first collects a few texture features in the neighborhood of the request position; then it returns the one minimizing the distance criteria. Another example is the case of edge features where a small line segment can be created onto a bigger one to minimize the distance to the request (or conversely two contiguous line segments can be united into a bigger one).

This is in contrast with classical graph matching where the test image is first discretized into a finite graph of limited size (usually small) before proceeding to the matching itself. In our case, everything happens as if the test graph has an infinite number of nodes. Thus, the classical decrease of robustness caused by discretization does not affect our approach (this is illustrated in the next Chapter).

3.4.5 Clustering of detected aggregates

After having detected aggregates in the test images, those aggregates cast hypothesis which are clustered in the four-dimensional scale-space of locations (x, y) and poses (σ, θ) (i.e. Hough transform) using a greedy process. Formally, let us assume two detected aggregates A'_i and A'_j. Two hypothesis, one for each, are then cast at position $\mathbf{p}_i^{hyp}$ and $\mathbf{p}_j^{hyp}$ with

$$\mathbf{p}_i^{hyp} = \text{sim2D}(\mathbf{p}_{ref}|A_i, A'_i)$$

and

$$\mathbf{p}_j^{hyp} = \text{sim2D}(\mathbf{p}_{ref}|A_j, A'_j).$$

Then, hypothesis are merged if they are not too distant in the scale-space. Namely,

Input:

- test image $\mathcal{I}$
- detection lattice $\mathcal{L}$

Output:

- List of weighted hypothesis H

MAIN:
$H := \emptyset$
$\mathcal{I}(\varphi_K)$:=EXTRACTKEYPOINTS($\mathcal{I}$)
$A_0 := root(\mathcal{L})$
for each $\phi_k^K \in \mathcal{I}(\varphi_K)$ **do**
  for each edge $e_{0i} = (A_0 \rightarrow A_i)$ **do**
    if $K_K^z(\phi_k^K, \phi_{0i}^K) < d_{0i}^{max}$ **then**
      $H := H \cup$ LATTICEFROMNODE($A_i, \{\phi_k^K\}, \mathcal{I}, \mathcal{L}$)
    end if
  end for
end for
CLUSTERHYPOTHESIS(H) // see Section 3.4.5
WEIGHTHYPOTHESIS(H) // see Section 3.4.6
return H

LATTICEFROMNODE($A_i, A_i', \mathcal{I}, \mathcal{L}$):
if is_terminal(A_i) **then**
  $\mathbf{p}_i^{hyp} :=$ sim2D($\mathbf{p}_{ref} | A_i, A_i'$)
  return $\mathbf{p}_i^{hyp}$
end if
$H := \emptyset$
for each edge $e_{ij} = (A_i \rightarrow A_j)$ **do**
  $\mathbf{p}_{ij}^* :=$ sim2D($\mathbf{p}_{ij} | A_i, A_i'$)
  $\phi_{ij}^* := (\mathbf{p}_{ij}^*, \mathbf{z}_{ij})$
  if $\min_{\mathcal{I}} K(\phi_{ij}^*) < d_{ij}^{max}$ **then**
    $\phi_{ij}' := \arg\min_I K(\phi_{ij}^*)$
    H:= $H \cup$ LATTICEFROMNODE($A_j, A_i' \cup \phi_{ij}', \mathcal{I}, \mathcal{L}$)
  end if
end for
return H

Algorithme 3.1: Pseudo-code for the detection.

for two hypothesis $\mathbf{p}_i^{hyp}$ and $\mathbf{p}_j^{hyp}$, they merge if

$$\left\|\mathbf{c}_i^{hyp} - \mathbf{c}_j^{hyp}\right\|^2 + \left\|\mathbf{h}_i^{hyp} - \mathbf{h}_j^{hyp}\right\|^2 < \frac{\sigma_i^2 + \sigma_j^2 + 3\sigma_i^{hyp}\sigma_j^{hyp}}{8} \tag{3.9}$$

This criterion is more permissive than the criterion of eq. (3.17) used during training (see below), as we expect more distortion in the test images than in the training images. Intuitively, it roughly corresponds to a maximum factor 2 in scale ratio, a maximal angle difference of 40° and to a maximal distance of 0.8σ between the two centers (note that in practice, it is impossible to reach all three limits together as the sum balances the conditions in eq. (3.9)).

The merging criterion enables robustness to non-rigid distortions, as it is illustrated in Chapter 4 (e.g. Figure 4.10). Note that hypothesis are clustered independently for each model view (i.e. two hypothesis belonging to different training views cannot merge). Finally, each cluster defines a **detection** D, with a center $\mathbf{c}_D$ and a radial vector $\mathbf{h}_D$ computed as the average of the clustered hypothesis. Finally, a probability is assigned to each detection (next Section).

3.4.6 Probabilistic model for clusters of hypothesis

Let a test image $\mathcal{I}$ be processed with the lattice $\mathcal{L}$ designed to detect the model object $\mathcal{O}$. As a result, a set of detections are output. Without loss of generality, let us consider the case of a single detection D in the following. More particularly, we will assume that the detected aggregates in the set $\{A'\}$ have all voted for this detection (i.e. ignoring the other aggregates detected in $\mathcal{I}$). The general probabilistic formula of finding object $\mathcal{O}$ at location $\mathbf{c}_D$ and pose $\mathbf{h}_D$ knowing the detected aggregates $\{A'\}$ is the following:

$$p(\mathcal{O}_D|\{A'\}) = \max_{\mathcal{V}_n \in \mathcal{O}} p(\mathcal{V}_{n,D}|\{A'\}^{\mathcal{V}_{n,D}})$$

with

$$p(\mathcal{V}_{n,D}|\{A'\}^{\mathcal{V}_{n,D}}) = p(\mathcal{V}_{n,D}) \frac{p(\{A'\}^{\mathcal{V}_{n,D}}|\mathcal{V}_{n,D})}{p(\{A'\}^{\mathcal{V}_{n,D}})} \tag{3.10}$$

(from Bayes' rule). That is, the object $\mathcal{O}$ is detected at position D if one of its view $\mathcal{V}_n$ is detected at this position ($\{A'\}^{\mathcal{V}_{n,D}} \subset \{A'\}$ denotes the set of detected aggregates voting for view $\mathcal{V}_n$ at position D). In our framework, each view $\mathcal{V}_n \in \mathcal{O}$ simply corresponds to a single training image $\mathcal{I}_n \in \mathbb{I}^+$. In the following we will drop the subscript n, D for clarity and focus on a single view $\mathcal{V} \in \mathcal{O}$.

The aim is now to make eq. (3.10) computable. As in similar works [Low01, RLSP06],

one can consider that $p(\mathcal{V}_D)$ is independent of the instance location and pose, thus $p(\mathcal{V}_D) = p(\mathcal{V})$, which is in turn assumed independent of the considered view so that $p(\mathcal{V})$ becomes constant. Then, one can develop the right-hand part of eq. (3.10) into

$$\frac{p(\{A'\}|\mathcal{V})}{p(\{A'\})} = \frac{p(A'_1|\{A'_{2+}\},\mathcal{V})p(\{A'_{2+}\},\mathcal{V})}{p(A'_1|\{A'_{2+}\})p(\{A'_{2+}\})} = \ldots = \prod_i \frac{p(A'_i|\{A'_{i+}\},\mathcal{V})}{p(A'_i|\{A'_{i+}\})}$$

with $\{A'\} = A'_1 \cup \{A'_2, \ldots, A'_n\} = A'_1 \cup \{A'_{1+}\}$. However, a new problem arises: all those conditional distributions $p(A_i|\{A'_{i+}\})$ are hard to learn in practice, as the number of training images is very small with respect to the huge space of function parameters (*i.e.* 2^n possible combinations for n Boolean parameters representing the presence or absence of each $A'_i \in \{A'_i\}_{1\leq i\leq n}$). (As we will see in Chapter 5, those conditional probabilities can be partly estimated in the case of class object detection, as the number of training images is by far superior to the one in the instance detection case.)

In order to make the evaluation of eq. (3.10) possible, we must then inject a-priori knowledge in the model. Typically, this requires the addition of some independence hypothesis between the conditional distributions. We can for instance consider $p(A'_i)$ and $p(A'_j)$ to be independent for $i \neq j$, which is roughly true unless A'_i and A'_j are overlapping in $\mathcal{I}$. This would yield:

$$\begin{aligned}\frac{p(\{A'\}|\mathcal{V})}{p(\{A'\})} &= \prod_i \frac{p(A'_i|\{A'_{i+}\},\mathcal{V})}{p(A'_i|\{A'_{i+}\})} \\ &\simeq \prod_i \frac{p(A'_i|\mathcal{V})}{p(A'_i)}\end{aligned} \tag{3.11}$$

In our case however, the clustering of the detected aggregates in the scale-space of hypothesis introduces a bias in the independence assumption. Indeed, we experimentally observe that for a same detection size, false positive detections are more likely to be generated by smaller aggregates in proportion (see Figure 3.6). This is expected as this straightly relates to the ratio of the surface recognized as model parts to the full object surface. To compensate this, we introduce a correction that lowers the weights of aggregates being small with respect to the detection size:

$$\begin{aligned}\frac{p(\{A'\}|\mathcal{V})}{p(\{A'\})} &= \prod_i \frac{p(A'_i|\{A'_{i+}\},\mathcal{V})}{p(A'_i|\{A'_{i+}\})} \\ &\simeq \prod_i \left[\frac{p(A'_i|\mathcal{V})}{p(A'_i)}\right]^{\eta(A'_i,D)}\end{aligned} \tag{3.12}$$

Figure 3.6: Noise affects more clusters of hypothesis having a large support (outer purple circle), as their surface absorbs more false aggregates (small blue circles). Left: a positive detection (the toy car); right: a negative detection for the same object.

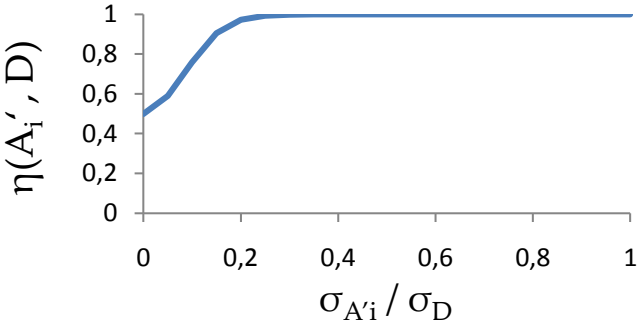


Figure 3.7: Plot of $\eta(A'_i, D)$ as learned based on positive and negative detections. Small aggregates with respect with the object size are purposely disadvantaged because they are more likely to arise from noise.

This correction takes the form of an exponentiation with the exponent lying in the range $0 < \eta(A'_i, D) < 1$ (thus canceling A'_i if $\eta(A'_i, D) = 0$ or having no effect if $\eta(A'_i, D) = 1$), since the ratio $p(A'_i|\mathcal{V})/p(A'_i)$ is necessarily superior to 1. One could explicitly derive $\eta(A'_i, D)$ from the stacking tolerance defined in eq. (3.9) but this appears difficult in practice. Instead, we use a logistic function learned on the basis of example aggregates belonging to true and false detections:

$$\eta(A'_i, D) = \frac{1}{Z}\left(\frac{1}{1+\exp[a+b\frac{\sigma_{A'_i}}{\sigma_D}]} + c\right)$$

where $Z = (1+\exp[a+b])^{-1} + c$ is a normalization factor. A plot of $\eta(A'_i, D)$ is shown in Figure 3.7 on which we can see that small aggregates in proportion are disadvantaged.

To sum up, until now our model makes a rough independence assumption between aggregates. (Notice that we consider $p(A'_i|\mathcal{V})$ and $p(A'_j|\mathcal{V})$ to be independent as well in eq. 3.12. This is mostly incorrect as knowing the model presence, two aggregates are

often correlated, but we use it for simplicity and because of the detection noise and the possibility of occlusion.) What happens however if two detected aggregates share some common branches (i.e. lattice edges) in their respective lattice path ? As a result, they would be highly correlated, thus undermining our initial assumption. A solution to this issue consists of considering the set of traversed branches $\{e'\}$ instead of $\{A'\}$. In fact, the Independence assumption is more acceptable for branches than for aggregates, as branches correspond to the atomic features composing the aggregates. Let us first define the set of lattice branches $\{e'\}$ traversed by the detected aggregates $\{A'\}$:

$$\{e'\} = \bigcup_{A'_i \in \{A'\}} \{e'\}^i$$

(by a slight abuse of notation, $\{e'\}^i$ is the set of branches lying on the lattice path leading to A'_i). In this definition the redundancy between aggregates is canceled by the union operator on sets of branches. Indeed, if two aggregates share common branches, each shared branch appears only once in $\{e'\}$.

As we saw in Section 3.4.4, each aggregate is detected after a cascade of positive decisions in the lattice, each one corresponding to traversing a branch. According to this definition, we can replace aggregates by branches in eq. (3.12). Following the same rationale, eq. (3.12) can be rewritten into:

$$\frac{p(\{A'\}|\mathcal{V})}{p(\{A'\})} \simeq \prod_{e_{jk} \in \{e'\}} \left[\frac{p(e'_{jk}|\{e'\}^{jk-}, \mathcal{V})}{p(e'_{jk}|\{e'\}^{jk-})} \right]^{\eta(A_k, D)} \tag{3.13}$$

where $\{e'\}^{jk-}$ is the set of branches already traversed before e'_{jk}. Hence from 3.10 we have the final formula

$$p(\mathcal{V}_D|\{A'\}) = p(\mathcal{V}_D) \prod_{e_{jk} \in \{e'\}} \left[\frac{p(e'_{jk}|\{e'\}^{jk-}, \mathcal{V}_D)}{p(e'_{jk}|\{e'\}^{jk-})} \right]^{\eta(A_k, D)} .$$

which can also be expressed as a detection score by taking the logarithm:

$$\text{score}(\mathcal{V}_D|\{A'\}^D) = \log(p(\mathcal{V}_D)) + \sum_{e_{jk} \in \{e'\}} \eta(A_k, D) \log \left[\frac{p(e'_{jk}|\{e'\}^{jk-}, \mathcal{V}_D)}{p(e'_{jk}|\{e'\}^{jk-})} \right] \tag{3.14}$$

Note that the ratio $p(e'_{jk}|\{e'\}^{jk-}, \mathcal{V}) / p(e'_{jk}|\{e'\}^{jk-})$ are constants estimated during training, and the term $\log(p(\mathcal{V}))$ is constant for every pair of model-view so it disappears.

Discussion Our probabilistic model for detection is simple and fast to compute. Basically, the detection score is a weighted sum of the weights associated to each traversed branches. In other words, the more numerous are the positive decisions taken by the micro-classifiers, the higher is the score. This appears appropriate as it means that the confidence of the object being detected directly relates to the number of its parts being recognized.

Moreover, another consequence is that there is a theoretical independence between the detection performance and the number of terminal aggregates in the lattice. In fact, eq (3.14) shows that the average detection score is proportional to the number of branches in the lattice. In other words, the ratio of true detection scores to false detection scores remains constant on average. Of course, the variance of this ratio might change, and the less terminal aggregates, the more unstable the performances might be. This is confirmed by our experiments in Chapter 4. (Note that we also tried a different probabilistic model: the one of Lowe [Low01], whose major difference is to take into account the number of keypoints in the hypothesis area, but results were only worse).

Besides, our probabilistic model only takes into account the conditional dependency between branches if they lie on the same lattice path. That is, two branches lying on distinct lattice paths (i.e. paths that are connected only by the lattice root) are considered independent. All in all, this results in modeling the joint conditional distribution only for small groups of branches, each group being independent with other groups. In our opinion, this is similar to the work of Özuysal et al. [zCLF09] where ferns are used to classify keypoint patches and achieve excellent performance. In their work, a single "fern" is a set of micro-classifiers small enough so that the joint probability distribution can be modeled, while different ferns are considered as independent. If the reader thinks of a lattice path as a fern, both models look similar (although the distributions used in their work are different).

3.5 How to build the detection lattice

As we saw earlier, each path in the lattice represents the growth of an aggregate – i.e. the gradual addition of new model features. As a consequence, the distinctiveness of the aggregate grows as well. We can imagine that when the aggregate ultimately contains all the model features (of a single training image), its distinctiveness is maximum. On the other hand, it becomes too much distinctive to allow some tolerance to noise. The key point of the training is then to find out at which point the aggregates become distinctive enough and still maintain some robustness to noise.

One solution would be to first build the complete lattice, then measure the distinctiveness of all aggregates, and finally prune the lattice so as to maximize this trade-off. However, because it is not tractable to build the complete lattice, we opt for a sub-optimal solution consisting of iteratively adding a new level to the lattice, measuring the distinctiveness of each new aggregate and stopping their growth when they become distinctive enough.

3.5.1 Algorithm inputs

The training algorithm takes as inputs:

- n_{pos} training images $\mathcal{I}_n$ of a single 3D model object (e.g. different viewpoints, or same viewpoints with different lighting conditions). In the following, we denote the list of positive images as $\mathbb{I}^+ = [\mathcal{I}_n]_{n=1}^{n_{pos}}$.
- a vector $\mathbf{P}^+ = [\mathbf{p}_n^+]_{n=1}^{n_{pos}}$ containing the positions of the model object in the previous images. The object center, scale and orientation in each image have thus to be known (at least approximately). [2]
- n_{neg} of negative training images. The model object must not appear in those images as they are used to estimate background distributions. In the following, we denote the list of negative images as $\mathbb{I}^-$.
- an integer parameter n_{term} controlling the trade-off between robustness and detection time of our method (see Section 3.4.2).

3.5.2 Iterative pruning of the lattice

Initially, the detection lattice only contains the empty aggregate ($l = 0$). Then, starting from the first level ($l = 1$), the lattice $\mathcal{L}$ is built in an iterative fashion. For each level $l \in [1, \infty]$, the following operations are executed:

1. All possible children aggregates are added to level l:

$$\mathcal{L}(l) = \bigcup_{A_i \in \mathcal{L}(l-1)} \left\{ A_j \in \mathcal{S}(\mathcal{G}) | \neg \text{is\_terminal}(A_i) \text{ and } \text{card}(A_j) = l \text{ and } A_i \subset A_j \right\}$$

where $\mathcal{G}$ is the unified prototype graph, $\mathcal{S}(\mathcal{G})$ is the set of all subgraphs of $\mathcal{G}$ and $\mathcal{L}(l) = \{A_j \in \mathcal{L} | \text{card}(A_j) = l\}$. In the following, we call those new aggregates as

[2] In the case of 3D objects with many viewpoints, the orientation can be hard to define. We personally choose the projection of the vertical axis onto the camera plane. In any case, this is not a big issue for our training algorithm: the lattice size will just augment in case of ill-defined orientations as less redundancy will be found between different viewpoints.

"**candidate aggregates**". (In fact, we arbitrarily limit their number to 8 times the number of non-terminal aggregates in level $l-1$[3] otherwise it would explode from $l \approx 4$ because of the exponential growth of the number of subgraphs).

2. Each candidate aggregate $A_j \in \mathcal{L}(l)$ is connected to its parents $\{A_i \in \mathcal{L}(l-1) | A_i \subset A_j\}$ of the previous level by branches $e_{ij} = (A_i \rightarrow A_j)$. For each such branch, the associated micro-classifier is initialized with a fixed threshold d_{ij}^{max}, learned separately (see Section 3.5.3).

3. If $l = 1$: go back to step 1. (We prevent aggregates of only one feature to become terminal.)

4. Loop until all candidates have been picked:

   (a) Pick the best candidate $A_k \in \{A_j \in \mathcal{L}(l) | \neg\text{picked}(A_j)\}$ according to the mutual information (see Section 3.5.4).

   (b) Set picked(A_k) to true.

   (c) Detect model aggregate A_k in each training images, leading to a set of detection $\{A'_k\}$.

   (d) If A_k is no more detected in negative images, then A_k becomes terminal.

   (e) If $\frac{n_{det}^{\mathcal{L}}}{n_{pos}} \geq n_{term}$, the training algorithm **stops** and **returns** the current lattice $\mathcal{L}$ cleaned from all remaining candidates ($n_{det}^{\mathcal{L}}$ is the total number of detected parts in the positive training images with the current lattice, n_{pos} is the number of positive training images and n_{term} is a parameter specifying the desired number of detected parts per model image).

5. Go back to step 1.

Thus, aggregates may become terminal at different levels depending on their distinctiveness (step 4.d). This can be connected to the human cognitive way of recognizing occluded objects: an object can be identified even if a very small but very characteristic part of it is visible and *vice versa*.

The stopping criterion in step 4.e aims at controlling the number of detectable model parts (i.e. the number of terminal aggregates). When at least n_{term} model parts are detected in each training image (on average), the lattice construction stops. The ranking system of step 4.a, detailed below, ensures that every images is roughly covered by the same number of parts. Note that a similar strategy is used in the MMRFS algorithm of

[3] That is, we randomly delete some candidates.

Cheng et al. [CYHH07]. In the following we describe in more detail the steps 2 and 4.a of the loop in Sections 3.5.3 and Section 3.5.4.

3.5.3 Learning the micro-classifier thresholds

As we saw previously, each lattice branch is associated to a micro-classifier that produces a binary decision by comparing a kernel distance to a threshold (see eq. (3.8)). We have tried in a previous version of this algorithm to learn those thresholds individually for each branch, by maximizing the mutual information on the training set. However, we have found that this method was providing too unstable results and was computationally costly. Instead, we propose now to learn by advance the thresholds "once and for all", regardless of the model object by using an independent training set.

As with traditional cascades, our strategy is to minimize the false negative rate for each micro-classifier. In other words, we do not want to miss a true detection. Since our lattice paths are dedicated to recognizing small parts of the model object and since we expect some noise, we thus set the thresholds to the maximal amount of noise expected after usual image transformations (jpeg noise, blur, viewpoint change...). Specifically, we have set the thresholds such that on average 95% of true matches are accepted (indifferently of the noise types). Thus, the threshold is only function of the type of the associated feature: for an branch e_{ij} adding the model feature ϕ_{ij}^t of type t, then we set

$$d_{ij}^{max} := d_t^{max}.$$

Detailed experiments on the calculation of these thresholds are provided in the next chapter.

3.5.4 Ranking of the aggregates

During the training loop, for each level a large number of candidate aggregates are generated. However, in the end only a few of the candidates are kept in the lattice (the ones that lie on the path of a terminal aggregate), whereas all the other ones are abandoned. The problem is then to select the best candidates so as to maximize both following criteria:

- The individual efficiency. Efficient candidates generate more true detections and less false detections.
- The coverage of every possible model parts. Because of the possibility of occlusion, we need well spread aggregates to detect every areas of the training images, even

though some areas are easier to detect than others (e.g. for a face: an eye versus a chin).

In order to satisfy both criteria, we need an adapted metric to rank the candidates. For that purpose, we have formalized this problem in the information theory framework. To begin with, the mutual information measures the correlation between two random variables X and Y:

$$\mathrm{MI}(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

In our case, we define the following variables:

- Let $\mathbf{p}_m^n$ be the m-th part of training image $\mathcal{I}_n$. We will define later more precisely what is an image part, for now let just consider that it is a local patch of the image: it has a center $\mathbf{c}_m^n$ and a radius σ_m^n.

- Let $\mathbf{A}_i(\mathbf{p}_m^n)$ be a binary random variable representing the fact that aggregate A_i is detected at position $\mathbf{p}_m^n$:

$$\mathbf{A}_i(\mathbf{p}_m^n) = \begin{cases} 1 & \text{if } \exists A_i' | \mathbf{p}_m^n = \mathbf{p}_{A_i'} \\ 0 & \text{otherwise.} \end{cases}.$$

- Let $\mathbf{O}(\mathbf{p}_m^n)$ be a binary random variable symbolizing the ground truth. As we wish to detect aggregates in parts belonging to the model object and not to the background, it is defined as:

$$\mathbf{O}(\mathbf{p}_m^n) = \begin{cases} 1 & \text{if } \mathcal{I}_n \in \mathbb{I}^+ \\ 0 & \text{otherwise.} \end{cases}$$

(We do here the simplifying assumption that positive images do not contain background.)

Obviously, one can use the mutual information to measure the efficiency of a candidate aggregate with respect to the detection task. In fact, the mutual information between $\mathbf{O}$ and $\mathbf{A}_i$ tells us how much the aggregate A_i is useful for detecting the model object. If A_i generates a lots of detections, both in positive and negative images, then the mutual information will be low, likewise if A_i is detected neither in negative nor positive images. The maximal mutual information will be reached when A_i is detected in positive images and not in negative images.

However, the mutual information used as a metric only satisfies the first criterion. In order to satisfy both criteria (individual efficiency and good coverage), we use the *gain* in mutual information. This is a measure of the amount of independent information delivered by a third variable Z in addition to X:

$$\text{gain}(Z|X;Y) = \text{MI}(X,Z;Y) - \text{MI}(X;Y) \tag{3.15}$$

where

$$\text{MI}(X,Z;Y) \quad = \quad \sum_{x,y,z} p(x,y,z) \log \frac{p(x,y,z)}{p(x,z)p(y)}.$$

In our case, $\text{gain}(\mathbf{A}_j|\mathbf{A}_i;\mathbf{O})$ straightly gives us the amount of information provided by a *second* candidate A_j, in addition to the first candidate A_i. If A_i and A_j detect exactly the same model part then the gain is null; on the contrary, if A_i and A_j are complementary (i.e. A_i and A_j detecting different model parts), the gain will be important. Furthermore, the gain will be even more important if A_i and A_j are not only complementary but also efficient *separately*. Likewise, we can measure the gain brought by a third aggregate A_k with respect to A_i and A_j using $\text{gain}(\mathbf{A}_k|\mathbf{A}_i,\mathbf{A}_j;\mathbf{O})$ (and so on with any numbers of aggregates).

As a consequence, we see here how the gain in mutual information can be a used as a metric to rank the candidates in a way that optimizes both (a) the selection of efficient candidates and (b) the detection of all model parts. However, computing the gain for several random variables has an exponential complexity and in our case, we have to deal with a large number of aggregates. The solution is then to use a new random variable, $\mathbf{M}(\mathbf{p}_m^n)$ defined as the probability that a given part $\mathbf{p}_m^n$ is detected with the current lattice:

$$\mathbf{M}(\mathbf{p}_m^n) = p(\exists A_i \in \mathcal{L}(l+) \text{ and } \exists A_i' | \mathbf{p}_m^n = \mathbf{p}_{A_i'}),$$

where $\mathcal{L}(l+)$ is the set of already picked or terminated aggregates:

$$\mathcal{L}(l+) = \{A_i \in \mathcal{L} | \text{terminated}(A_i)\} \cup \{A_i \in \mathcal{L}(l) | \text{picked}(A_i)\}.$$

Since every picked aggregate will terminate sooner or later anyway, they are considered equally with terminal aggregates. In other words, $\mathbf{M}$ acts as a memory of all potential terminal aggregates such that $\text{gain}(\mathbf{A}_k|\mathcal{L}(l+);\mathbf{O}) \approx \text{gain}(\mathbf{A}_k|\mathbf{M};\mathbf{O})$. The advantage is that the computation of $\text{gain}(\mathbf{A}_k|\mathbf{M};\mathbf{O})$ is in constant time instead of exponential time. This simplification is possible because our process for accepting candidates is iterative:

candidates are accepted one by one (see step 4), making it possible to "update" $\mathbf{M}$ each time that a candidate is picked or terminated. Note that Vidal-Naquet and Ullman [VNU03] have also used the mutual information as a feature selection process; however, their optimization was with a max-min procedure different from ours (and slower as the complexity in their case is squared with the number of parts).

3.5.5 Discretization of the training image into parts

Until now we have used undefined model parts $\mathbf{p}_m^n$ for the calculation of the information gain. Depending on the definition of what is a part, this computation can take different forms and be slower or faster. Obviously, it is not tractable to compute the information gain for an infinite number of parts. As a result, we have chosen to discretize the training images into a small, finite set of parts.

More precisely, we have clustered the scale-space in a similar manner to the spatial pyramid of Lazebnik et al. [LSP06]. We have used one pyramid of 4 levels for each training image, and a single "virtual" location for all negative images. This makes a total of $1^2 + 2^2 + 4^2 + 8^2 = 85$ parts for each training image (each level is separated from others by a factor 2 of scale). Hence, we need to store $85n_{pos} + 1$ probabilities in $\mathbf{M}$. Those are respectively the probabilities for each model part to be detected with the current lattice, and the probability to detect the background (see Figure 3.8).

Note that we have not used false positive images in the computation of the information gain for the seed branches (first iteration in our training loop). This is because we want to select seed features that are robust to initiate the detection of true model parts regardless of their detection performance on background images. This is in the same philosophy as with traditional cascades: the first micro-classifiers, more than any others, should generate as few false negative as possible in order not to impair the rest of the detection process (the next micro-classifier purpose being then to evacuate false positives).

Explicitness of the coverage map M As we saw earlier, $\mathbf{M}(\mathbf{p}_m^n)$ stores the probabilities of detecting parts with the current lattice. We define those probabilities as:

$$\mathbf{M}(\mathbf{p}_m^n) = \begin{cases} 1 - \prod_{n_{det}^{\mathcal{L}(l+)}(\mathbf{p}_m^n)} (1 - p_{rep}^K) & \text{if } \mathcal{I}_n \in \mathbb{I}^+ \\ 0 & \text{otherwise.} \end{cases} \tag{3.16}$$

where $n_{det}^{\mathcal{L}(l+)}(\mathbf{p}_m^n)$ denotes the number of detection of the part $\mathbf{p}_m^n$ by the terminal and picked aggregates in the current lattice, and p_{rep}^K is the keypoint repeatability: it is the

probability that a keypoint is detected at the same place after some noisy transformation. In $\mathbf{M}(\mathbf{p}_m^n)$, we consider for simplification that the probability that an aggregate is detected is equal to the probability p_{rep}^K that its seed feature is detected. Making the assumption that all seed features are detected independently, we get the above formula (upper row of eq. (3.16)). (Note that this formula makes it possible to update $\mathbf{M}$ in constant time each time that $n_{det}^{\mathcal{L}(l+)}$ is incremented.) Practically, we use linear interpolation to dispatch a detection into several contiguous cells $\mathbf{p}_m^n$ of the spatial pyramid. This is done based on the center $\mathbf{c}_{A_i'}$ and radius $\sigma_{A_i'}$ of the detected aggregates. Moreover, we only count as positive a aggregate A_i' detected in model image $\mathcal{I}_n$ if it accurately extrapolate the ground truth position:

$$\left\|\mathbf{c}_n^{hyp} - \mathbf{c}_n^+\right\|^2 + \left\|\mathbf{h}_n^{hyp} - \mathbf{h}_n^+\right\|^2 < \frac{\sigma_n^{hyp}\sigma_n^+}{4} \tag{3.17}$$

where $\mathbf{p}_n^+$ is the known position of the model object in $\mathcal{I}_n$ and $\mathbf{p}_n^{hyp} = \text{sim2D}(\mathbf{p}_{ref}; A_i, A_i')$ (this formula allows for some tolerance: a factor 1.5 in scale difference is accepted if the two centers $\mathbf{c}_n^{hyp}$ and $\mathbf{c}_n^+$ are equal, for instance). Finally, we do not store the probability of false detections in $\mathbf{M}$. It would be pointless, as there are a very large number of potential parts in the negative images and we can not reduce all those probabilities in a single value. An example of the evolution of $\mathbf{M}$ during training is shown in Figure 3.8.

3.6 Conclusion

We have presented a novel approach for the detection of instances of specific objects. This system enables a fast recognition robust to noise and occlusions. The introduction of local kernels allowed to combine different types of features while still preserving the invariance to a large set of transformations. The cascaded structure of the detection lattice enables a smart handling of the graph matching problem in a continuous space (i.e. the test image is not discretized into a graph) contrary to classical graph matching, in order to improve both speed and robustness.

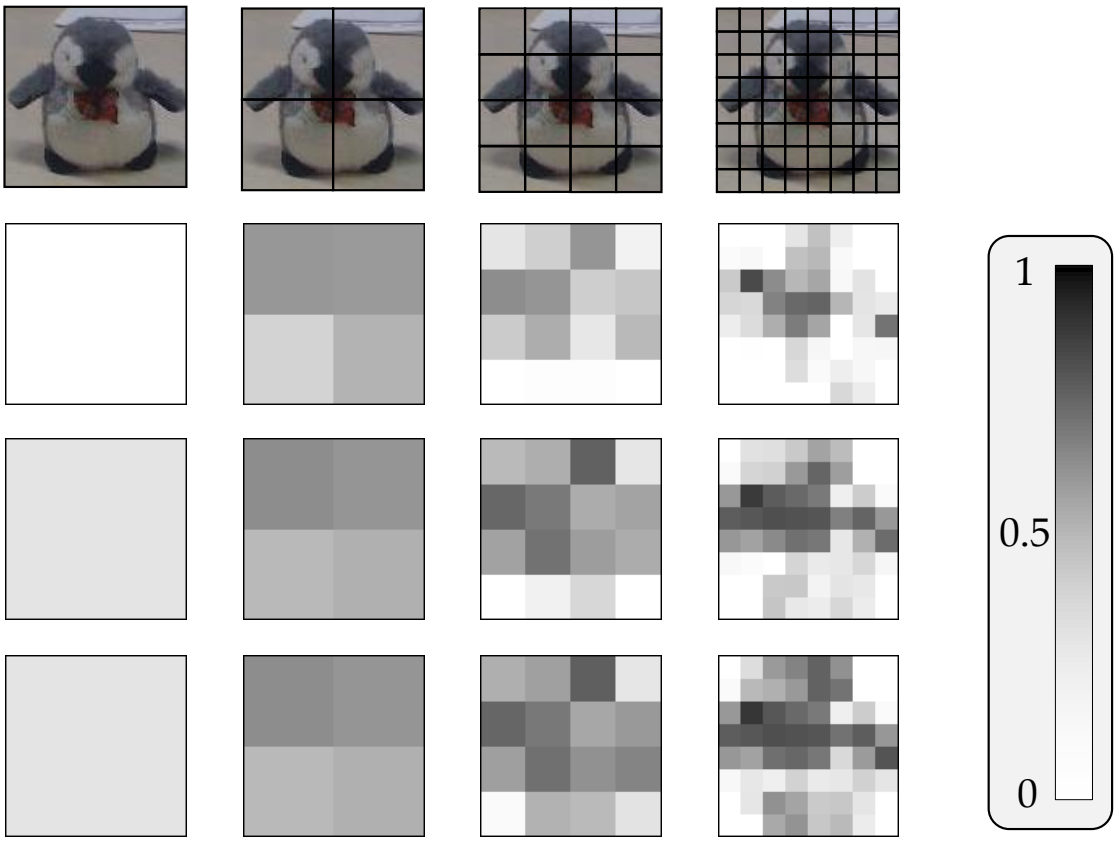


Figure 3.8: Top row: model parts according to the scale-space pyramid decomposition. Subsequent rows: evolution of the probabilities stored in **M** to detect each model part while building the detection lattice. Second row: $l = 2$; third row: $l = 3$; fourth row: $l = 6$.

Chapter 4

Evaluation of Our Contribution For Specific Object Detection

Contents

This chapter presents a quantitative evaluation of the contribution presented in the previous chapter. Firstly, we detail the preliminary training step for determining the micro-classifier's thresholds independently of any model object. Then, an in-depth examination of the performance is conducted for different datasets: our own dataset for mobile robotic, the dataset of Ferrari et al. [FTG06] and the dataset of Rothganger et al.

[RLSP06]. We also make a quantitative comparison against some of the most popular algorithms from the state of the art (a baseline RANSAC, the improved LO-RANSAC from Chum et al. [CMK03] and Lowe's method [Low04]). Results shows that our approach outperforms those algorithms in realistic conditions of mobile robotic. On the two other datasets, our method performs slightly less well than the best existing methods but still holds out well while providing very fast detection.

4.1 Discussion about the evaluation

There exist several methods and datasets for evaluating the performance of a specific object detection system. In the following of this section, we give an overview of the existing datasets and metrics in the state-of-the-art and we justify a selection of the most relevant ones with respect to our method and our objectives.

4.1.1 Test datasets

4.1.1.0.1 Existing datasets There are several available datasets to evaluate algorithms for specific object detection. Among them, we can cite:

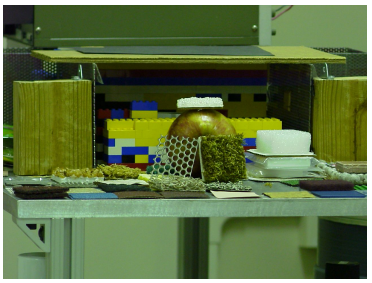
- The dataset of Ferrari et al. [FT04] (9 model objects, 23 test images). It focuses on 3D object recognition (4 objects have one view, the other 5 have 6~8 views) in heavily cluttered conditions. Strong occlusions make this dataset quite challenging. Moreover, large distortions of the model instances are often visible because of (1) extreme perspective effects and (2) non-rigid distortions (e.g. bended magazine).

- The dataset of Rothganger et al. [RLSP06] (8 model objects, 51 test images) for 3D object reconstruction and recognition. The particularity of this dataset is that a large number of training views are provided for each object (between 20 and 28), so that a full 3D reconstruction is possible. concerning the test images, they show the objects in varying scale, pose and level of occlusion. The amount of clutter also varies depending on the images. Finally, note that the test images are very large in surface (2~3 MPixels) which leads to a large number of keypoints per image and hence a long detection time.

- The dataset of Moreels et al. [MP05, MP08]. It consists of close-up views of home objects (101 objects, 123 test images) and toys (31 objects, 141 test images). Similarly to the previous datasets, the test images show close-up views of the model objects in cluttered background with possible occlusion. Note that only two methods have been evaluated on this dataset (Moreels' and Lowe's method).

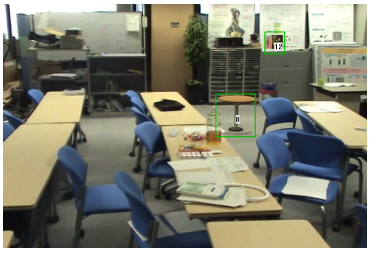
- The dataset of Kootstra et al. [KYdB08]. It consists of close-up views of seven objects under 36 different viewpoints (images are taken by a mobile robot equipped with a CCD camera). The dataset is not publicly available to our knowledge, but appears easy because the sample pictures shown in the original paper [KYdB08]

Ferrari et al. [FT04]



Rothganger et al. [RLSP06]

Moreels et al.
[MP05, MP08]



Our dataset
[RLAB10, Rev09]

Figure 4.1: Sample test images from different datasets for the evaluation of specific object recognition systems.

display no occlusion and few clutter. No real evaluation has been performed on this dataset (only pairwise keypoint matching).

- The dataset of Kushal and Ponce [KP06] (9 model objects, 80 test images). It emphasizes the 3D aspect of object recognition: each model object comes with 7~12 training viewpoints and appears in various 3D poses in the test images with possibly strong occlusions. This dataset mostly resembles the dataset of Rothganger et al. [RLSP06]. Furthermore, only the method of Kushal and Ponce [KP06] and two baselines have been evaluated on this dataset.

To sum up, in all those datasets test images consist of close-up views of the model objects, often strongly occluded (see some examples of test images in Figure 4.1). Although this is challenging, we do not find this aspect really realistic regarding scenarii of mobile robotic. Usually, a mobile robot has to detect an object from a certain distance despite a low-quality acquisition device, which is an essentially different task. As a result we decided, in addition to evaluating our system with the first two datasets presented above for comparison purpose, to create a new dataset providing more realistic conditions for robotic vision. We describe our dataset and the associated experiments in Section 4.3.

4.1.2 Evaluation metrics

In most similar works, results are presented in the form of Receiver Operating Characteristic (ROC) curves. This is a clever choice as it enables one to see in one glance the efficiency of the method: the more the curve follows the left and top boundaries, the better the performance is. In a classical ROC curve, the abscissa axis represents the False Positive Rate (FPR) and the ordinate axis represents the True Positive Rate (TPR). The TPR and FPR are respectively defined as the ratio of true (resp. false) detections to the total number of positive (resp. negative) boxes defined in the ground truth. A true detection occurs when the detection system hypothesizes a box fitting enough the ground truth (in our case, a minimum in mutual overlap of 25% is required) and having the correct label. Since for each detection is associated a confidence by the detection system, by varying the threshold of minimum confidence one can generate all ROC curve points.

Nevertheless, it is difficult to define the FPR in the case of object detection, because the number of negative boxes is potentially infinite when we consider every possible combinations of location, orientation and scale not fitting positive boxes. As a consequence, instead of the FPR we prefer to use the average number of False Positive detections Per test Image (FPPI) as in [FFJS07]. Another alternative chosen by Ferrari [FTG06] is to use the $1 - precision$ value, where the precision is defined as the ratio of the number of positive detections to the number of detections (both above a given threshold). This choice appears to us less pertinent as it is less meaningful to the reader.

Averaging issues In the literature, results are generally presented in term of a single ROC curve, although the dataset is composed of several model objects varying in ground truth box count and difficulty. Because in our dataset the number of ground truth boxes for each model object can considerably vary, we present both the standard single ROC curve (all model objects combined) and the averaged ROC curve obtained by vertically averaging the individual ROC curves corresponding to each model object.

In other words, two different scenarii are used to compute the ROC curves for our dataset. In the first scenario, the ROC curve is generated by only taking into account the detections boxes, indifferently of the object identity (i.e. the same thresholds are used for all objects); in the second scenario, the ROC curve is obtained by vertically averaging the individual ROC curves corresponding to each object (i.e. different thresholds may correspond to similar FPPI values). We believe that the first scenario is more realistic as in a practical utilization individual thresholds per object may be unknown, although

it is less fair because in this case all objects are not equally represented in the curves. For completeness, we show both curves each time believing that each curve should be considered with an equal importance.

Area Under Curve (AUC) In order to measure in a single real value the performance of a detection system, we use the Area Under ROC Curve (AUC) metric. As the name indicates, it simply corresponds to the area under the ROC curve (i.e. for the range of FPPI shown in the plot, usually $0 \leq \text{FPPI} \leq 1$). This is a robust measure which ranges between 0 and 1 (note that it is closely related to the average Precision).

4.2 Preliminary training

Preliminarily to the experiments, we have to train the lattice parameters that are independent of the model objects. The following parameters are to be trained:

- The thresholds d_{ij}^{max} of each branch micro-classifier (Section 4.2.1),
- The local kernel parameters (Section 4.2.2).

4.2.1 Learning the subclassifier thresholds

As we saw previously, each lattice edge is associated to a simple micro-classifier that produces a binary decision by comparing a kernel distance to a threshold (see eq. (3.8)). Our goal in this section is to learn the classifier thresholds in a automated manner. We have tried in a previous version of this algorithm to learn those thresholds individually for each edge, by maximizing the mutual information on the training set. However, we have found that this method was providing too instable results and was computationally costly. Instead, we propose now to learn by advance the thresholds "once and for all", regardless of the model object by using an independent training set.

As with traditional cascades, our strategy is to minimize the false negative rate for each subclassifier. In other words, we do not want to miss a true detection. Since our lattice paths are dedicated to recognizing small parts of the model object and that we expect some noise, we thus set the thresholds to the maximal amount of noise expected after usual image transformations (jpeg noise, blur, viewpoint change...).

For this purpose, we have used the dataset made available by Mikolajczyk and Schmid [MS05]. This dataset was originally proposed for the evaluation of feature detectors+descriptors and is composed of 48 pictures belonging to eight distinct sets (6 pictures in each set). Each set represents the same scene purposely affected by a type of

noise, such as JPEG noise or slight 3D viewpoint change. What makes the dataset interesting for our purpose is that the pairwise pixel correspondences between each image of a set are provided as ground truth. As a result, it makes it possible to measure the amount of noise caused by usual transforms on the kernel distances defined in Section (3.3).

Namely, we have extracted a large amount of random aggregates in all 48 images. Then, knowing aggregate correspondences (based on the adequacy between ground truth and their position) we have measured the distribution of kernel distances for true matches. Note that contrary to other authors like Lowe [Low04], we are not interested by the case of false matches. Indeed our unique purpose is to build lattice paths that are robust to noise; the distinctiveness of the aggregates being later handled by the succession of subclassifiers – eliminating more and more false detections, as with traditional cascades.

The resulting histograms of distances are presented in Figure 4.2 for each kernel. We have set the thresholds such that on average 95% of true matches are accepted (indifferently of the noise types). Specifically, the threshold is only function of the type of the associated feature: for an edge e_{ij} adding the model feature ϕ_{ij}^t of type t, then we set

$$d_{ij}^{max} = d_t^{max}.$$

The retained thresholds are presented in Table 4.1. For the keypoint type, there are two kernels: the first one is used to compare SIFT descriptors of seed features (first row of Table 4.1) while the second one is used in the subsequent levels of the lattice (second row of Table 4.1). For the seed branches, we only authorize matches if the descriptor distance is less than $d_{K^z}^{max} = 0.251$, which corresponds to 75% retrieval ($d_{K^z}^{max}$ is the micro-classifier threshold of the seed branches and appears in Algorithm 3.1). We reduced this proportion with respect to 95% in order to fasten the matching. Indeed, it makes a big difference in speed because the detection time is somehow proportional to the number of matched seed features. According to our experiments, restraining the threshold to 75% of retrieval divides by 10 the number of matches compared to a threshold of 95% and does not impair much the performance. This optimization is used only for the first level; for the subsequent levels we set $\zeta_K = 0.5$ for the second kernel (representing 95% retrieval). Finally, we have also measured the repeatability of keypoints p_K^{rep} needed in eq. (3.16), see rightmost column of Table 4.1

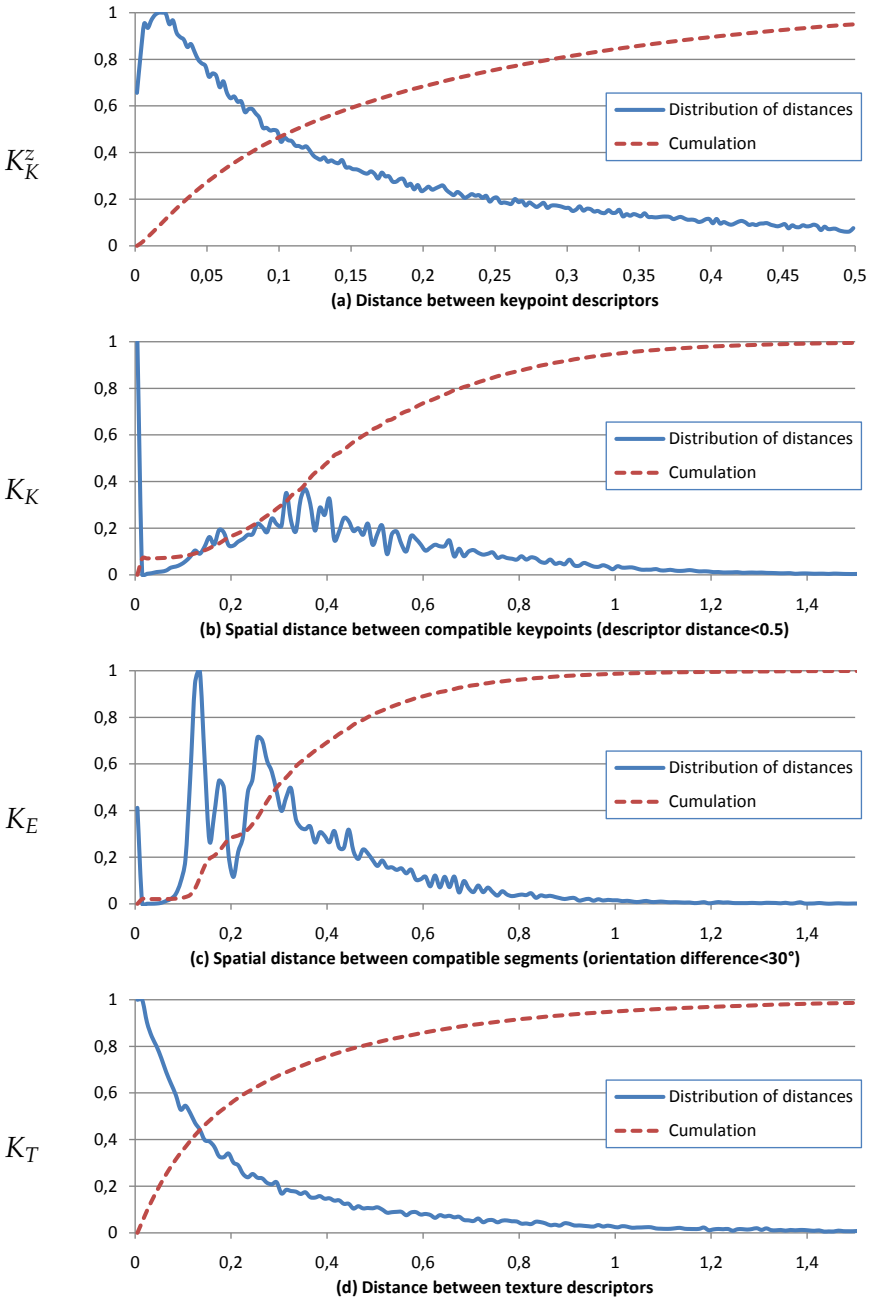


Figure 4.2: Distance distributions corresponding to true matches for each kernel. The irregular curves for K_K and K_E are due to normalization effects on integer distances (typically for small line segments or keypoints).

Table 4.1: Retained thresholds and standard-deviation for each feature types.

Type	Kernel	Branch level	d_t^{max}	repeatability
Keypoint (desc)	K_K^z	$0 \rightarrow 1$	0.251	0.345882
Keypoint (pos)	K_K	$n \rightarrow (n+1), n \geq 1$	1.004	
Segment line	K_E	$n \rightarrow (n+1), n \geq 1$	0.793	
Texture	K_T	$n \rightarrow (n+1), n \geq 1$	0.944	

4.2.2 Other kernel parameters

A few other parameters have to be fixed *a priori* for the local kernels. We arbitrarily set $\alpha_K^2 = 8$ for the second keypoint local kernel K_K (see eq. (3.2)) to balance location against scale and orientation. Concerning textures, since their descriptor varies smoothly along the image, they are robust to slight positioning errors so we considered a unique test as sufficient, i.e. we set $\alpha_T^2 = 1$ and $\zeta_T = 0$ (see eq. (3.4)). This minimum setting produces excellent results, see below.

4.3 The CS17 dataset

As stated above, we have created a new dataset for the sake of better simulating classical robotic vision conditions. The name "CS-17" originates from the laboratory name directed by Professor Ariki (Kobe University) where it was made.

Dataset description

We have manually shot a dozen of indoor videos with a standard SONY handycam. The image resolution is willingly smaller (720×480) than in the existing datasets where high-quality photos are used. As stated above, our choice is motivated by the aim of better simulating the realistic conditions of robotic vision, which are generally much more difficult because of the poor luminosity conditions of the indoor environment, the variety of noises (captor noise, movement blur, video interlace) and also because of the objects themselves which are not always heavily textured. The videos were sampled at 10 fps (resulting in 2837 frames, a much higher number than in the existing datasets) where the ground truth was manually labeled. All training and test images can be found at http://liris.cnrs.fr/jerome.revaud/CVIU.

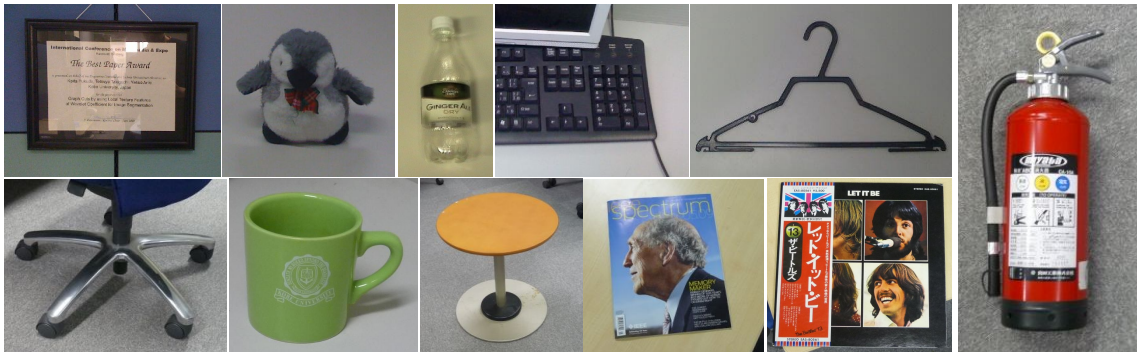


Figure 4.3: Model objects used in the experiments. The amount of texture and the shape dramatically differ depending on the object.

Table 4.2: Dataset statistics. The third and fourth columns respectively denote the number of training images and the number of positive boxes in the ground truth.

#	model name	train	test
1	frame	1	784
2	stuffed animal	4	319
3	50cl bottle	3	130
4	keyboard	1	121
5	cloth hanger	2	352
6	desk chair	4	477
7	tea mug	4	224
8	coffee table	4	329
9	journal	3	201
10	vinyl record	3	207
11	fire extinguisher	3	235

Model objects Eleven objects at our disposal, such as a cloth hanger or a fire extinguisher, were used to test our system. The detailed list of model objects is presented in Table 4.2 and their pictures can be seen in Figure 4.3. The objects were chosen based on their amount of texture and their shape in order to cover a large range of possible indoor objects: the bottle, the journal and the vinyl models are heavily textured contrary to the mug and the small table models; the frame, the journal and the vinyl models are flat and rectangular whereas the stuffed animal, the bottle, the chair legs, the mug and the small table have complex 3D shapes; the hanger, the chair legs and the mug contain holes and/or sharp edges; the frame, the bottle, the chair legs and the journal are prone to specular reflections; etc.

As in similar datasets [FTG06, MP08], we have used a small number of training images per model object (between 1 and 4). However, all model objects are only shown from a single 3D viewpoint (in other words, this is a 2D dataset). When several training images are available for a single object, it simply means that they are taken in different viewing conditions (e.g. scale, lighting) under the same viewpoint with possible redundancy. From the 2837 frames, 2272 frames contain at least one model object (i.e. 565 frames only show background) and the total number of true boxes is 3502.

4.3.1 Parameter Tuning

Before proceeding to the experiments, we need to tune two parameters of our approach:

- The average number n_{term} of detected parts per model image (Section 3.5.2).
- The number n_{neg} of negative images (Section 3.5.1). For simplicity, we assume in the following that all negative images are independent and have roughly the same size.

In order to study how the performance of our approach vary depending on these two parameters, we have divided our dataset between a held-out validation set (all the im-

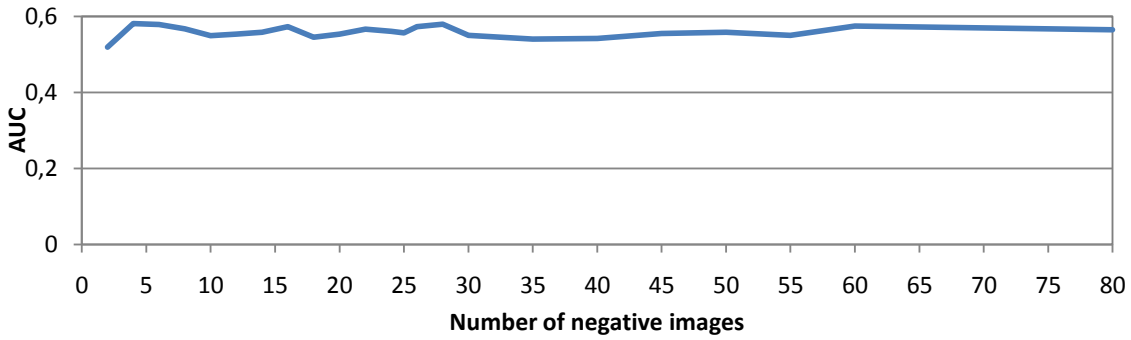


Figure 4.4: Impact of the number of negative training images on the performance in term of AUC value (in fact: almost no impact).

ages having the suffix "0.jpg", hence $1/10^{\text{th}}$ of all test images) and a test set (the rest of the dataset). All tuning experiments presented in the following of this subsection are done on the validation dataset.

Number of negative images We have first investigated the number of negative images in the training set. To that aim, we have launched the training procedure for values of n_{neg} comprised between 2 and 80, having fixed beforehand n_{term} to 200. The first 19 negative images were shot by us in the laboratory, the next images are sampled from the "Background" category of the Caltech-101 dataset [FFFP06]. (The average number of seed features per negative image is about 1000.) Then, we have measured the performance in term of AUC metric. Results are presented in Figure 4.4.

As can be observed, results are stable for any number of negative images (only a small decay is observed for $n_{neg} = 2$), in accordance with our theoretical model (i.e. the number of negative images is not involved in our probabilistic model). However, it should be noted that the number of negative images have an influence on the average depth of the lattice paths (i.e. on the distinctiveness of the terminal aggregates). This is logical insofar as aggregates become terminal when they are no more detected in negative images (see Section 3.5.2). So for a large number of training images, aggregates must be very distinctive to terminate their path. Because their distinctiveness grows along with their depth, it encourages long paths in this case.

In Figure 4.5, this effect is illustrated by the fact that the number of detections is much larger for low values of n_{neg}. This in turn impacts the detection time: a larger number of hypothesis involves a larger clustering time, and because our algorithm is greedy it becomes typically quite slow. The difference of time between $n_{neg} = 5$ and $n_{neg} = 80$ is about 0.5s according to our experiments. As a consequence, we choose in

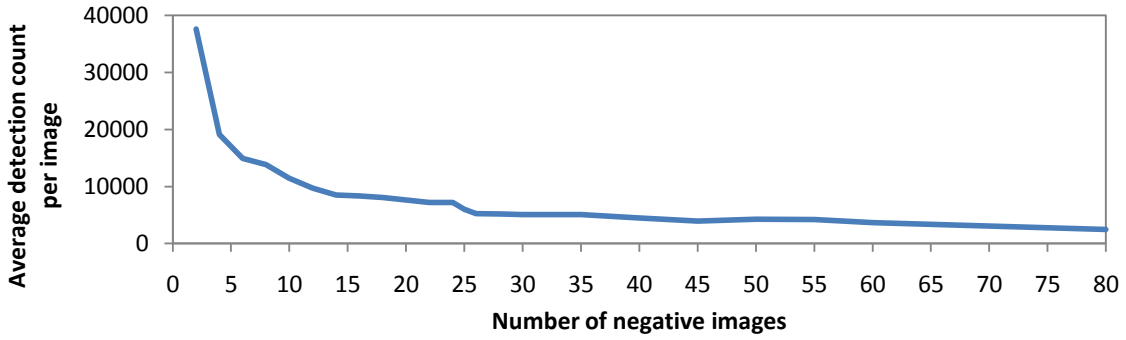


Figure 4.5: Impact of the number of negative training images in term of the average number of detections per image. Because large numbers of negative images encourage distinctive aggregate, there are less detections per image for similar performance.

the following experiments an intermediate value of $n_{neg} = 30$ because the number of average detections per image stabilizes from this value. This corresponds in practice to a cumulation of about 30,000 seed features in the negative images, hence a probability for termination of $p(A_i|\neg\mathcal{O}) < 1/30000 \simeq 3.10^{-5}$.

Number of model parts Now that we have studied the influence of the number of negative training images, we turn now to the number of model parts detected by the lattice. We recall that the parameter n_{term} corresponds to the minimum average number of detected aggregates per model image during the lattice construction (Section 3.5.2). For instance, a value $n_{term} = 1$ means that the lattice detects at least a single aggregate (i.e. part) in each model image, on average. Theoretically, this parameter has no influence on the detection performance (see the discussion in Section 3.4.6), but this is without considering the possibility of occlusion or noise. In practice, we have to trade-off between a small value of n_{term} (favoring a small lattice hence a fast detection) and large value of n_{term} which brings more robustness to occlusion and noise (because more model parts are likely to be detected).

We have experimented different values of n_{term} to verify this effect. The results are presented in terms of AUC in Figure 4.6 averaged for $n_{neg} \in [2, 30]$. As can be seen, the performance does not vary significantly for $n_{term} \in [40, 200]$ (especially, no significant improvement is observed for $n_{term} \geq 135$). However, the variance of the results significantly decreases from $n_{term} \geq 60$, confirming our guess that small numbers of model parts produce less stable results. On the other hand, the detection time is significantly inferior for a smaller number of parts: Figure 4.6 shows that the detection time is roughly linear with the value of n_{term} (remember that the preliminary feature extraction time is constant and independent of n_{term}; it is denoted by a red area in Figure 4.6). This makes

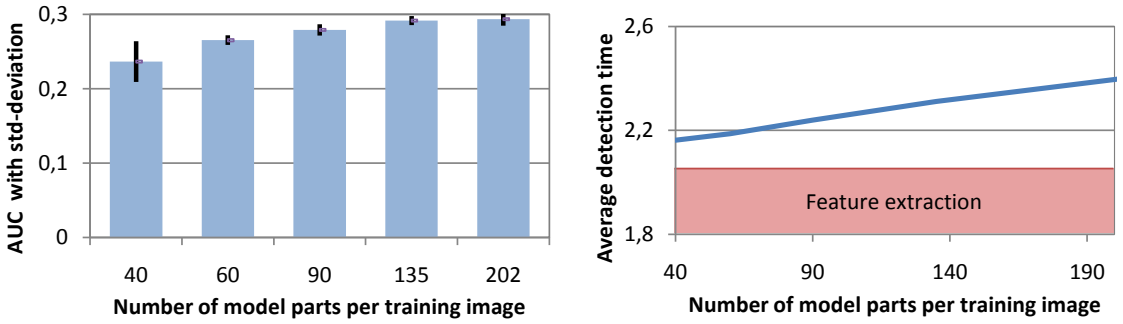


Figure 4.6: Influence of the parameter n_{term} on the detection performance (left) and on the detection time per image for searching all 11 objects (right). n_{term} corresponds to the average number of detected parts per model images.

sense because n_{term} straightly relates to the lattice breadth (see the paragraph "detection time" below). As a consequence, we use the adequate value of n_{term} corresponding to the best AUC to detection time ratio in Section 4.4 and 4.5. In this section, we just use the value producing the best results ($n_{term} = 200$); but remember that almost similar results are achieved from $n_{term} \geq 60$ as shown in Figure 4.6.

4.3.2 Comparative experiments

Experimental settings

We have trained our detection lattice with all available positive images (knowing the bounding boxes and orientations of the objects), and $n_{neg} = 30$ negative images, constituted by 19 background photos of the indoor environment plus the 11 first images of the "Background" category of Caltech-101. We have set the number of model parts to $n_{term} = 200$, but similar results can be achieved with lower values as illustrated in Figure 4.6.

Concurrent approaches We compare our contribution against the most commonly used methods for this task from the state of the art:

- A baseline RANSAC [FB81] with a homography. Keypoints from all model images are stored into a k-d tree to enable their fast retrieval (matches are based on the first-to-second best neighbor distance ratio being less than 0.8). Since a homography needs at minimum 4 matches to infer a pose hypothesis, the number of iterations excessively increases when the ratio of inliers is small. As a consequence, we used an over-estimated probability of 10% (the true effective rate being much smaller). The tolerance for the localization of inliers was set to 5% of the diameter

of the detected objects (this value gives the best results on our dataset). Finally, the stopping criterion and the probabilistic formula asserting a detection are the same as in PROSAC [CM05].

- The locally optimized RANSAC (LO-RANSAC) as theorized by Chum et al. [CMK03] and practically adapted by Philbin et al. [PCI*07]. It is similar to RANSAC except that a similarity transform is used in the main loop and a homography in the local optimization step (called every time that a new maximum of inlier count is found). This solution enables a much lower probability for inliers ratio as the similarity transform only requires a single match: we set a worst-case ratio value of 1% without noticeable slowdown. As with the standard RANSAC, the probability of finding the object is moved to generate the ROC curve, however contrary to RANSAC an inlier tolerance of 5% for the outer loop and 2.5% for the inner loop produces the best results.

- The object recognition system from Lowe [Low04]: SIFT keypoints are used for local feature matching using a kd-tree search. Each match casts a vote for an approximate model pose, the votes are then accumulated into a hash map and all clusters of 3 votes or more are verified using an affine transform. Remaining hypothesis undergo a probability decision and the threshold on the acceptance probability is used to generate the ROC curve.

Experimental results

The detection performance is presented for each method in terms of ROC curves in Figure 4.7. As stated in Section (4.1.2), we display two plots in order to reflect both averaged performance (left plot, all model objects have an equal weight) and overall performance (right plot, all instances are considered indifferently of their model object). This distinction appears important to us because some model objects are largely more present in the test set than others (e.g. the frame object, see Table 4.2), making the right plot imbalanced. Nevertheless, the two plots look rather similar for all methods meaning that the performance on each model object *alone* are approximately equivalent. Sample detections for our method are also shown in Figure 4.8.

Our contribution significantly outperforms all concurrent methods. Among them, Lowe's method is producing the best results, but is still far behind ours. In order to check whether this superiority was coming from our implementation of Lowe's method or not, we have tested it on several separate images. An example of such test is shown in Figure 2.11 (Chapter 2). In this image, the beaver is correctly detected with a score of

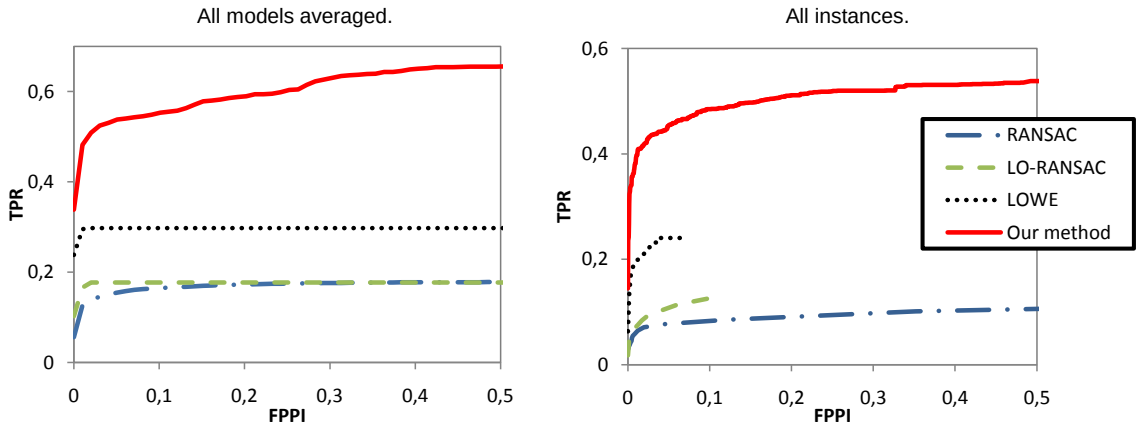


Figure 4.7: Comparative results for the CS-17 dataset in term of ROC curves.

$(1 - 10^{-12})$ (15 matches) despite a projective transform and a lot of clutter, whereas the second best detection (out of fourteen) only has a score of 0.23 (6 matches). In addition to those sample tests, we have also tested our implementation on the Rothganger dataset (see Section 4.5 below) and we have obtained a similar result to the one reported, which has reinforced the idea that our implementation is trustable.

In fact, the explanation of the failure of traditional methods arises instead from the fact that our test images are extremely noisy. An example of such noise is shown in Figure 4.9 where the leftmost image are the training images and the other image corresponds to detected instances, rescaled such that all objects have the same scale. As can be observed, serious distortions affect the instance pictures (note that those images are rendered using the PNG format in this document, so that no artificial noise is added compared to the original test images). Those distortions are caused by scale changes (instances often appear very small because they are seen from a large distance), MPEG noise, movement blur and light reflections. As a result, a lot less keypoints are detected on the instances, or similarly, the repeatability of keypoints is reduced.

Extracting more keypoints To verify that our keypoint detector was not the cause of this detection failure, we have tried to generate more keypoints: instead of using Lowe's executable (which is not parametrizable), we have used the SIFT implementation made available by Andrew Vedaldi [VF08]. We have lowered the extraction threshold, which resulted in 3720 keypoints per test image on average, instead of 949 before. The curves displayed for Lowe's method and LO-RANSAC method in Figure 4.7 are actually produced with this setting. Note that this setting does not really change the outcome: extracting more keypoints only slightly extends Lowe's curve to the right from the point where it reach its maximum TPR (thus no improvement is achieved) at the price of a

Figure 4.8: Examples of detections on the CS-17 dataset using the proposed method (FPPI is set to 0.01 for each model). This method is robust to occlusions, rescaling, viewpoint changes and various captor noises like movement blur. Correct detections are represented with green squares, incorrect detections with red squares, and missed objects with dashed blue squares.

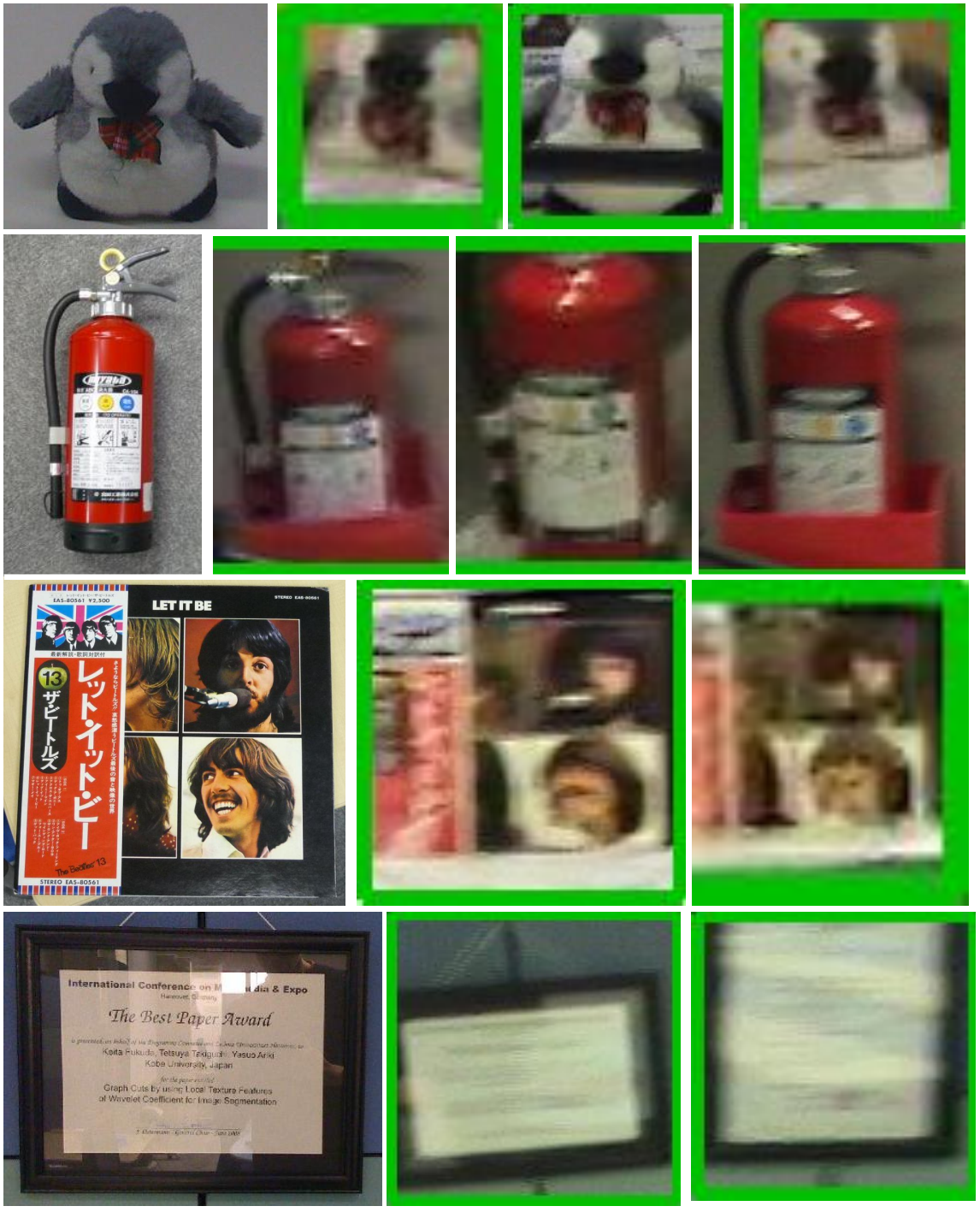


Figure 4.9: Sample detections for our method on the CS-17 dataset for four model objects: the stuffed animal, the fire extinguisher, the vinyl and the frame (leftmost images). It should be noticed that the instances suffers from serious noise sources such as rescaling, MPEG noise, movement blur and light reflections. For this reason, traditional methods fails at detecting most instances.

Table 4.3: Average processing times with a 2.3 GHz computer on a 720x480 test image for 11 model objects. Our detection time depends on the parameter n_{term} (e.g. 400 ms for the curves reported earlier with $n_{term} = 200$).
[1] We use Lowe's executable to extract the SIFT keypoints, which is rather slow because the image first has to be converted to the PGM format, and the extraction output is made through a text file. With more efficient implementations of the SIFT detector, the detection time is about 1s.

Operation		time (ms)
Feature extraction	SIFT keypoints	1648[1]
	Edges	310
	Textures	210
Object detection	RANSAC	225
	LO-RANSAC	108
	LOWE	81
	Ours	80~400

considerably slower detection speed. On the contrary, for the standard RANSAC, the best performance was obtained with Lowe's executable because for RANSAC with an homography, the chance of hitting four correct matches is higher when the number of image keypoints is small.

Detection time

We summarize in Table 4.3 the average detection time necessary for searching the 11 models objects in each test image. All tests were processed on a 2.6 Ghz computer without coding any particular optimizations. Globally, the feature extraction step on a 720x480 image takes about 2s which three-quarters are spent solely for keypoint extraction (note that Lowe's executable used to extract SIFT keypoints is probably not optimal, e.g. other keypoint detectors like the one in [zCLF09] are almost instantaneous). Moreover, edge and texture features are only used by our contribution although their extraction is rather fast. An efficient implementation could be obtained by parallelizing all these processes on different cores.

Our contribution remains relatively fast, depending on the value of n_{term} (see Figure 4.6). Indeed, the small number of features in terminal aggregates (2.6 on average, to relate to the thousands of features initially present in the prototype graphs) enables almost instantaneous verifications. For a value of $n_{term} \geq 60$, the detection time of our system is comparable to that of Lowe's method or LO-RANSAC and the detection performance is still largely superior to theirs. Still, we believe that our system could be accelerated several times with an appropriate optimization. In fact, the detection time is linear with the number of branches starting from the root node, because each keypoint in the test image is compared to each first-level seed feature. So by interposing a tree-

shaped indexing structure between the root and the first-level aggregates, one should be able to achieve a significant speed-up. The work of Beretti et al. [BDV01] for indexing graph using nested spheres, for instance, seems to be a promising track to achieve such a boost. This is however beyond the scope of this dissertation.

4.3.3 Discussion

As stated earlier, our contribution significantly outperforms the methods which we compared to. Although it is difficult to give an exhaustive list of the causes of this superiority, here are the main trends:

- Contrary to the others, our method generates a lot of detections from which the correct ones tend to pile up whereas the negative ones behaves as a random uniform noise. In comparison, even a very low probability threshold for Lowe's method and LO-RANSAC surprisingly does not produce a lot more true positives (i.e. ROC curves stop very soon in Figure 4.7). This is due to the fact that these methods require respectively at least 3 and 4 correctly matched keypoints (but often more) to assert a single detection, which is quite a difficult prerequisite in our noisy conditions (see how dirty object instances are in Figure 4.9).

- When the model object is poorly textured, the keypoint descriptors are very unspecific, decreasing the probability that the best-to-second distance ratio is accepted. Since our method uses absolute distance between keypoint descriptors, it is not affected by this issue.

- Scale is very different between model images and instances in videos. Usually, model photos are taken in close-up while detection needs to recognize the objects at a much smaller scale. This is a problem for keypoints, since their theoretical invariance to scale owns some limits, whereas texture features which are not salient are readily extractable everywhere in the image. On the contrary, our method specifically addresses this issue by dedicating aggregates to large scaled model parts (see Section 3.5.5).

- When the object surface is small or when the object contains holes (e.g. the cloth hanger), SIFT keypoints describe most of the time the background instead of the object. Our method can use line segments which are less prone to background clutter.

Moreover, our approach demonstrates that an efficient detection scheme can be built in a two-steps manner: one first part for fast and rough hypothesis generation using a

Used features	AUC	Difference with K+E+T
Keypoints	0.44	**-0.10**
Edges	0.47	**-0.07**
Textures	0.55	**+0.01**
Keypoints + Textures	0.51	**-0.03**
Keypoints + Edges	0.54	**+0.01**
Edges + Textures	0.54	**+0.00**
K+E+T	0.54	

Table 4.4: Contributions of each feature type on the CS-17 dataset for our method. The leftmost column specifies which subset of the 3 basic feature types were used to construct the lattice from the second level (the first lattice level always remains composed with keypoints). The two other columns show the performance thus obtained in term of ROC AUC and the difference with the normal case where all three feature types are used.

small fraction of the model information, and a second part for verification using the rest of the model information. In our case, the cascaded structure of the lattice intrinsically includes the two parts, because the first part for hypothesis generation is constituted by the seed branches, and the second part for verification is performed by the rest of the paths (from level $l \geq 2$).

Importance of each feature type We have taken a look at the importance of each feature type regarding the detection performance. To do so, we have constructed the detection lattice using a subset of the three proposed feature types (i.e. keypoints, edges and textures). For instance, we have constructed detection lattices using only edges. (Note however that the first level of the lattice is an exception to this constraint as it remains always exclusively composed of keypoints.) Then, we have measured the performance of such lattices on our dataset in terms of AUC value; results are presented in Table 4.4.

In general, the combination of two feature types performs better than one type used alone but clearly the best performance is obtained when the "Texture" feature type is present. The explanation to the superiority of the texture features is that they are not affected by repeatability issues: they are readily extractable everywhere. In contrast, the keypoint and edge types are salient and thus their extraction is subject to the presence of noise (typically: rescaling, motion blur), although the edge feature is more flexible than the keypoint feature (see Section (3.3.2)). To conclude with, this confirms our assumption that salient features, still important for reducing the search space and for readjusting the aggregate positions during their growth (texture feature does not have any anchorage), should be used in association with non-salient features to gain robustness.

For completeness, we also show in Table 4.5 the proportion of each feature type automatically chosen in each detection lattice for this dataset. The texture type is logi-

Table 4.5: Lattice statistics for each model object. The three columns respectively denotes the percentage of keypoints, edges and textures automatically chosen in the recognition lattice.

#	model name	% kpt	% edge	% tex
1	frame	22.2	14.4	63.3
2	stuffed animal	14.1	8.9	77.0
3	50cl bottle	19.5	8.6	71.9
4	keyboard	20.7	8.7	70.7
5	cloth hanger	22.4	19.1	58.9
6	chair leg	12.1	18.4	69.5
7	tea mug	10.2	22.2	67.7
8	coffee table	8.9	14.8	76.3
9	journal	18.3	14.8	66.9
10	vinyl record	35.8	22.8	41.4
11	fire extinguisher	21.5	11.7	66.8
	average	18.5	15.0	66.5

cally dominating for two reasons: first of them, it is an efficient feature as said earlier; and secondly texture features are extracted in large number at the construction of the prototype graphs, making them statistically more often connected to the other features.

4.4 The ETHZ toys dataset

For comparative purpose, we have evaluated our approach on two existing datasets. The first of them is the ETHZ-toys dataset as made available by Ferrari et al [FTG06].

Dataset description The ETHZ-toys dataset [FTG06] is composed of 9 model objects and 23 test images containing 42 instances in total. The number of training images per model depends on the model object: some objects are covered by (up to) 8 views forming 360° viewpoints, while some others have only a single training image (e.g. two magazines). All model objects are strongly textured, either with textual patterns or with printed images, making them an easy target for keypoint-based detection systems. However, the main challenge of this dataset lies in three points: (a) the heavy amount of clutter in the test images; (b) the important amount of occlusions; and (c) the strong distortions with which the model objects appear in the test set. The distortions can be either caused by a close-up, causing the perspective projection to be quite extreme (e.g. the fourth image in Figure 4.10 for instance), or the consequence of intended non-rigid deformations (e.g. first and fifth images in Figure 4.10).

Experimental settings As mentioned above the distortions of the instances in the test images are extremely important and cause the failure of most pairwise matching between SIFT descriptors, causing in turn extremely low performance for our method. Probably for this reason, Ferrari et al. have used affine invariant features instead of only scale invariant features. This supplementary invariance indeed provides better pairwise

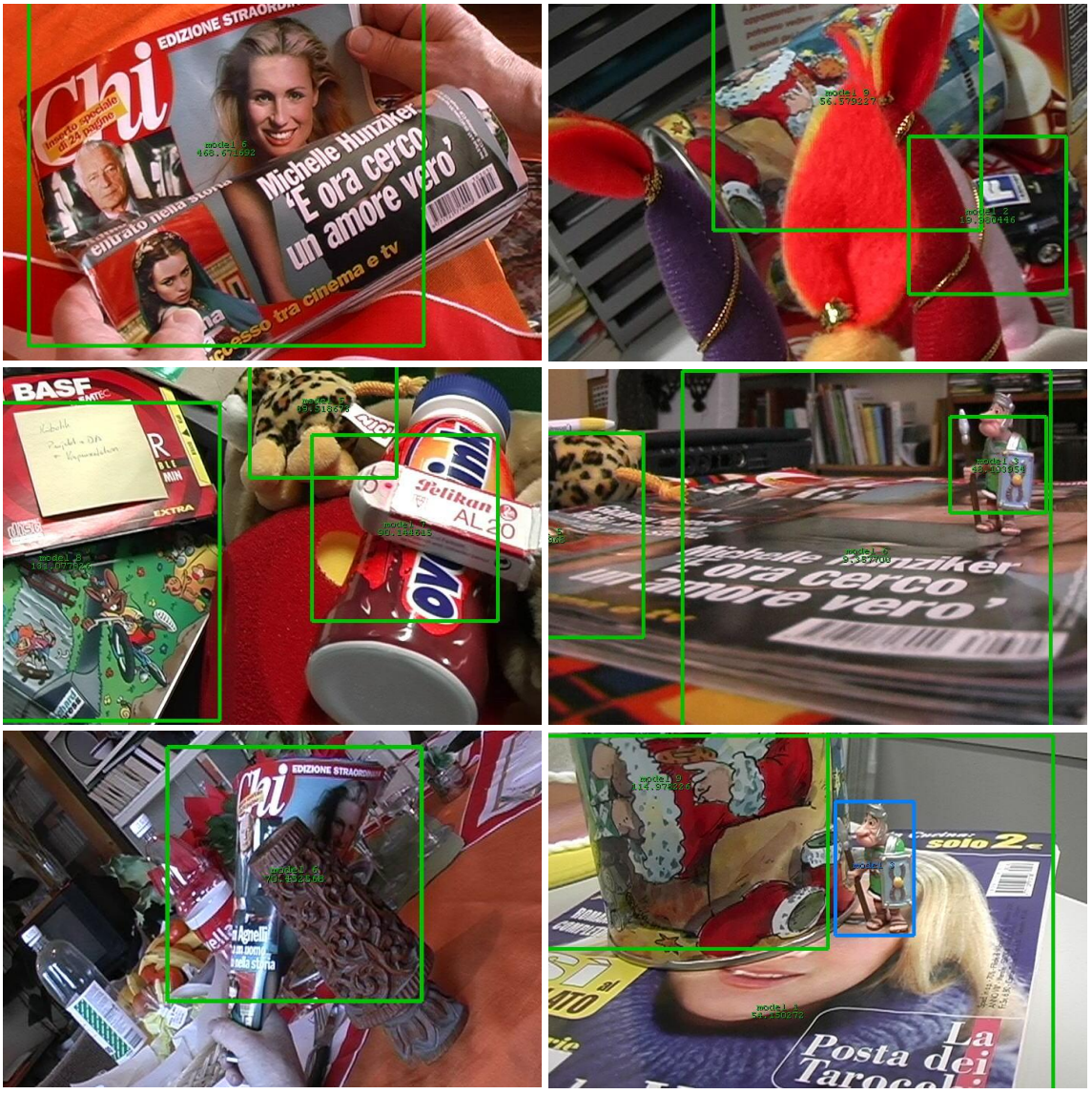


Figure 4.10: Sample images from the ETHZ-toys dataset [FTG06]. Instances suffer from strong distortions (non rigid distortions in the first and fourth image, extreme perspective effects in other images) or strong occlusions (all 6 images). The correct detections for our method are bounded by green rectangles, the roman soldier toy inside a blue rectangle in the last image was not detected by our method.

matching performance for strong viewpoint changes. We could have done the same, but our method is not designed to use affine invariant features as seed features (i.e. there would be two half vectors per feature instead of one).

So instead, we have used a simpler trick inspired to us by Yu and Morel on their work with ASIFT [YM09]. ASIFT is an extension of SIFT to the affine invariance. In this work, affine SIFT features are generated by simply extracting SIFT features on affinely deformed versions of the input image. In our case, we have simply generated additional views using a projective transform and we have added those views to the training set. Various distances of the objects to the camera were used (i.e. assuming flat objects), resulting in 9 times more pictures than in the original training set. Concerning the negative training set, we have used the first $n_{neg} = 30$ images of the "Background" category of the Caltech-101 dataset as no negative images were provided in the original dataset.

Experimental results we have tried different values of n_{term} and we only display here the results for $n_{term} = 135$, as no significant improvement was obtained beyond. The corresponding ROC plot is shown in Figure 4.11. Our method produces good results at null FPR (equivalent to the one of Ferrari's method), however no real improvement is obtained for increasing FPR - contrary to Ferrari's method. Note however that color information is used in Ferrari's method but not in ours, which might be an important handicap for us on such a difficult dataset. Still, our method is well beyond all other methods to which Ferrari et al. compare to for decent values of FPR (the abscissa axis on the ROC plot corresponds to $1 - precision$, meaning that after 0.5 there are more erroneous detections than correct ones). As a consequence, we consider those results as very satisfying regarding the fact that our method was not designed to be robust against such strong occlusions and distortions. The fact that 73% of the test instances are correctly detected without any false detection shows that our method resists quite well and is still able to discriminate between objects and clutter. The rest of the instances are not detected because of huge occlusion.

Detection time The detection time for our method is several orders of magnitude smaller to the one of Ferrari's method. On average, our method requires 3 seconds per image to search all 9 objects (feature extraction step included) whereas an hour is spent for each image with the method of Ferrari (in both case, a 2.4 GHz computer is used). This long time is easily explained by the fact that Ferrari's method iterates several steps of an expansion-contraction procedure in order to recognize the maximum

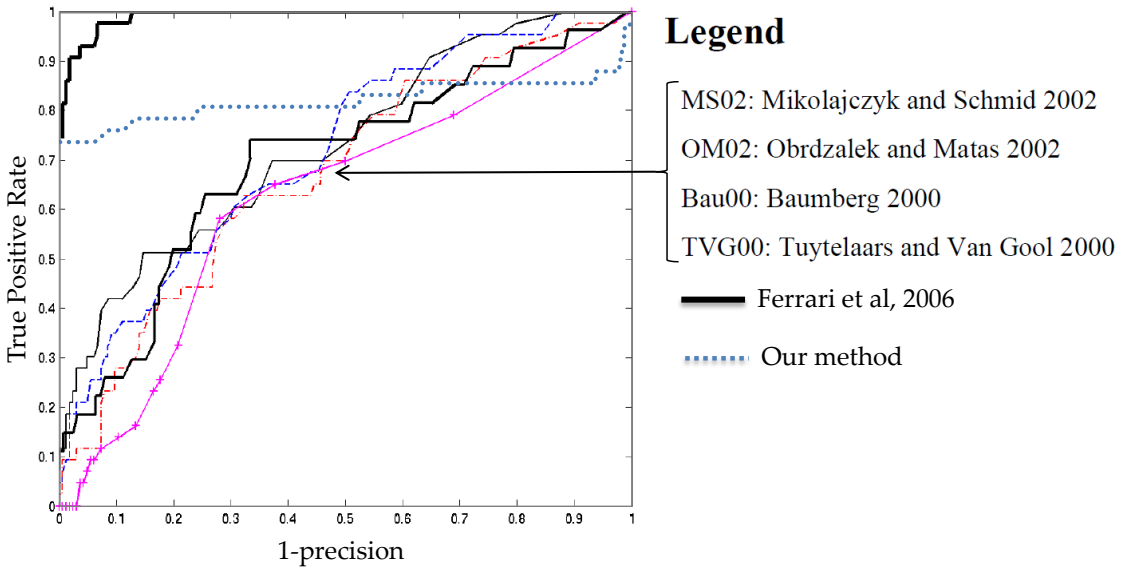


Figure 4.11: ROC plot for the ETHZ-toys dataset. The two best curves are for Ferrari et al.'s method and our method. See text for details.

surface for each object. Indeed, in the work of Ferrari et al. [FTG06] the training images are densely sampled with local textures to exhaustively learn the surface of each object. This is in contrast with our method where all training images are also densely sampled, but where only a tiny fraction of them is retained in the detection lattice.

4.5 The Rothganger dataset

The second existing dataset to which we compare is the Ponce-Rothganger dataset [RLSP06].

Dataset description The Rothganger dataset is composed of 8 model objects and 51 test images containing 78 instances in total. Its purpose is the evaluation of systems for 3D recognition of specific instances. As a result, each model object is purposely shot under an important number of 3D viewpoints (between 16 and 29) in the training set. Likewise, object instances appear viewed from various 3D viewpoints in the test set.

More globally, this dataset does not present major difficulties, as most of the model objects are well textured and thus easily recognizable using standard keypoints-based methods. Its main advantage is that many existing systems [RLSP06, Low04, MP05, FTG06] have already been evaluated with it, making the comparison possible.

Experimental settings As usual, we have trained a single lattice for each model object, setting the number of negative images to $n_{neg} = 30$ (using again the first 30 images of

the Background folder of the Caltech-101 dataset [FFFP06]) and trying different number of terminal nodes in the set $n_{term} \in \{40, 60, 90, 135, 200\}$.

Due to the large size of the training images, we have been forced to downsize each of them by a factor 2. In fact, the original training images are up to 4 MPixels and each of them contains thousands of keypoints. During training, this creates multitudes of candidate aggregates, making the learning extremely slow. Reducing the size up to 1 MPixels allows our algorithm to *relatively* quickly construct the detection lattice (about 4 hours per model object on our 2.6 GHz machine) – and more importantly, avoids the computer to crash because of memory overflow.

Besides, it seems impossible with our current technical configuration to build the index maps for the edge features when the image size exceeds some threshold (remember that we need 6 distance maps as well as 6 pointer-to-nearest-segment maps to index the edges in a single training image). As a result, using the edge features makes our computer consistently crash because of memory overflow (2 GB of RAM apparently do not suffice). In the end, we excluded the edge features from our feature set. We also thought about generating additional perspective views for training like we did on the ETHZ datasets, but because of the good results without this trick and the multiplication of training time and memory required, we did not do that either.

To sum up, our experimental setting was to downsize the training images to a factor 4 (in surface) and to only use SIFT keypoints and textures (no edge), without generating additional training images.

Experimental results We compare in Figure 4.12 our results for the value $n_{term} = 90$ in the form of a ROC plot (no real improvement was obtained for higher values of n_{term} while the training+testing time was increasing). This value is thus the most interesting trade-off between performance and speed in our opinion.

As can be seen, our results are approaching the best results obtained on this dataset. More specifically, our method produces slightly superior results to the Rothganger's black and white setting (our method does not use color information neither) for a detection time again several orders of magnitude faster (Rothganger et al. reports one hour spent for each image on a 3 GHz computer). Lowe's method produces the best performance on this dataset probably because of the great number of pairwise matches of SIFT descriptors for each instance, both thanks to the great deal of local textures on the model objects and the high-resolution nature of the test images. Because our method selects in the final lattice only a tiny proportion of all SIFT keypoints found in the training images, a logical consequence is a slight loss of performance against occluded and/or

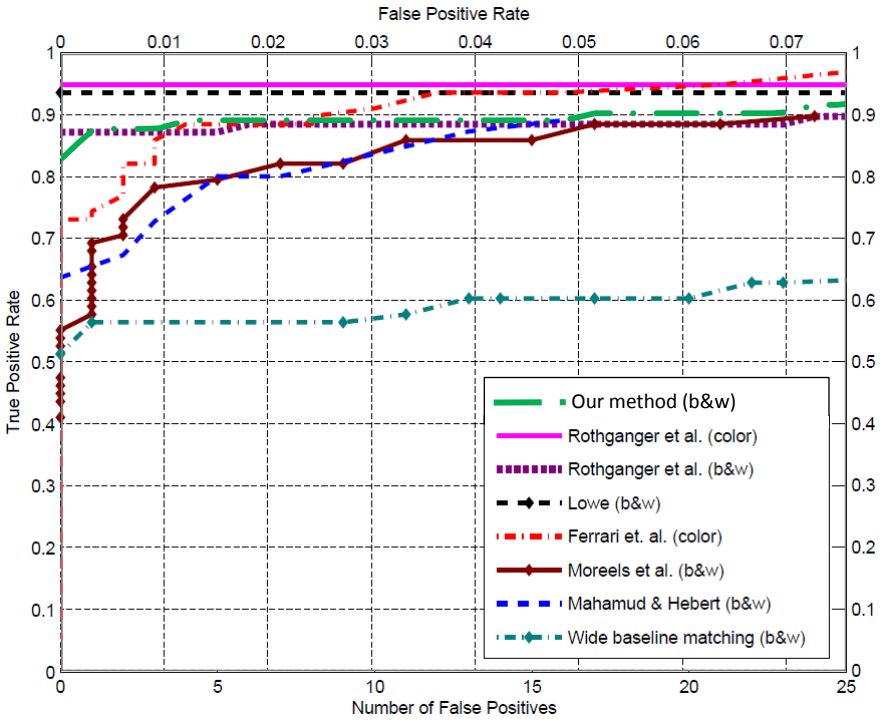
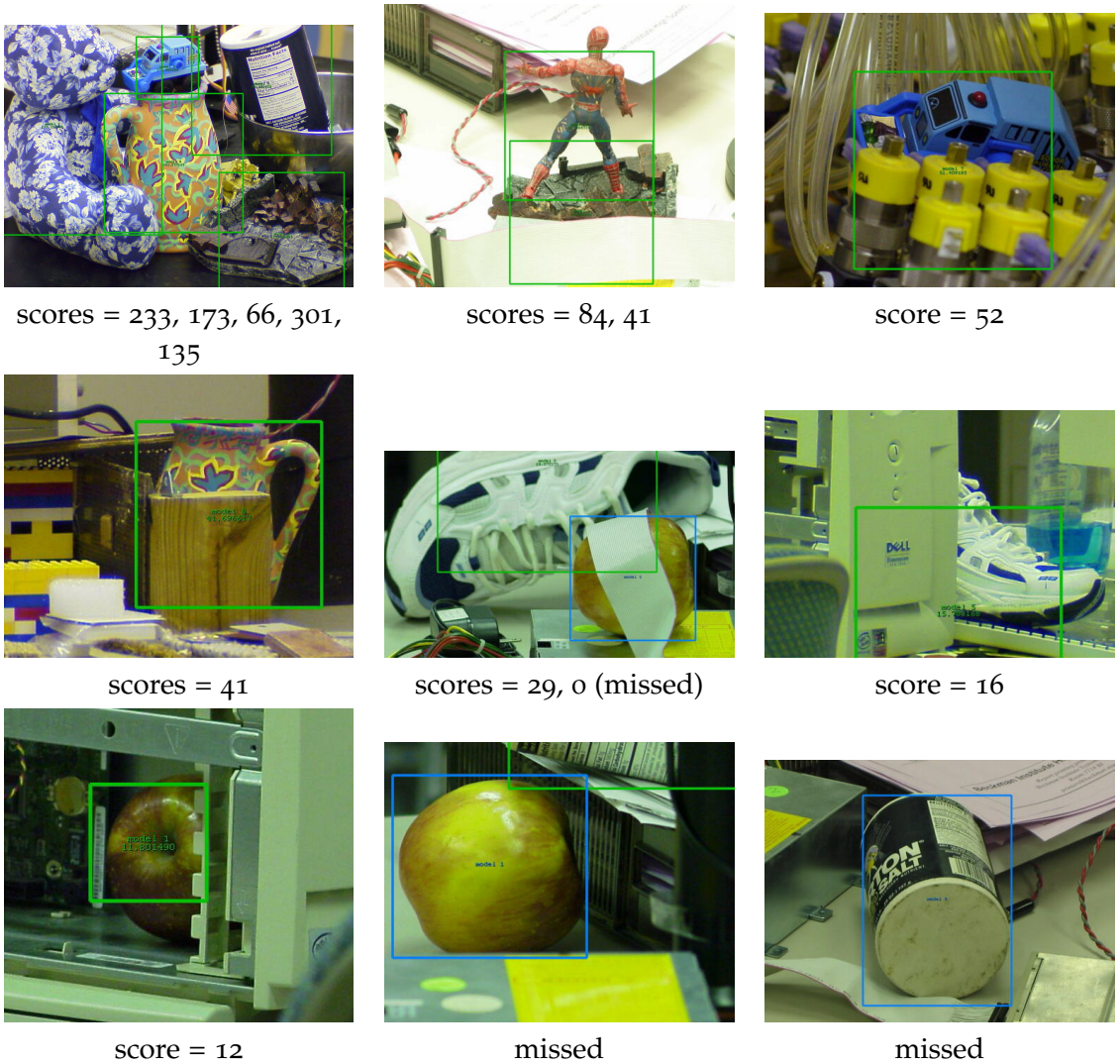


Figure 4.12: Comparison of performance of our method in terms of in ROC curves on the Rothganger dataset.

poorly textured objects.

The apple object, for instance, has a rather uniform texture (or, at least, a weak texture) which does not produce many SIFT keypoints. In fact, it is clearly the object which is the most difficult to detect using our method. Figure 4.13 illustrates some of its instances completely missed during the tests. On the contrary, the strong occlusions undergone by some instances (e.g. the vase in second row, first column of Figure 4.13) are not a problem for our method because many aggregates are still detected from the large number of keypoints still visible (e.g. see the high score of the occluded teddy-bear model object in Figure 4.14, second row, third column). Finally, a few instances which are viewed from viewpoints that are too much different from the viewpoints provided in the training set, are also missed as seed keypoints do not correspond. This problem was handled by using affine invariant features in the original paper of Rothganger et al. [RLSP06], but our experiments show that except 2 instances of the cylinder box model, all other instances can be well recognized using classical SIFT features.

Robustness to 3D viewpoint change We now give a short example of how our method is robust to viewpoint change. For this purpose, we have trained a lattice on a small subset of the training images available for the teddy bear model. Namely, we have



scores = 233, 173, 66, 301, 135 | scores = 84, 41 | score = 52

scores = 41 | scores = 29, 0 (missed) | score = 16

score = 12 | missed | missed

Figure 4.13: Some correct detections and the worst failures of our method on the Rothganger dataset (images are cropped around the bounding box for visual clarity). When there are more than one object per image, scores are given in left-to-right and top-down order. For information, the average scores on this dataset are 174.0 and 12.9 for the true positives and the noise (FP), respectively.

selected three images showing a frontal view of the teddy bear (see Figure 4.14). Then, we launched the recognition on the test dataset to check to which extent our method was robust to viewpoint change. The teddy bear is a good example as it appears under many viewpoints more or less different from the frontal view. Results are shown in Figure 4.14. Although it is hard to generalize from a single example, results still show that our method is able to recognize viewpoints that are $\simeq 80°$ apart from what was learned, although the detection score obviously decreases proportionally to the angle difference.

4.6 Conclusion

We have presented a quantitative evaluation of our contribution for specific recognition on several datasets:

- On our own dataset (the CS-17 dataset), specifically fitted to model the realistic conditions of robotic vision, our contribution yields performance largely superior to the state of the art in terms of ROC plots.
- On the two existing datasets, our method gives correct results although it appears slightly less robust to occlusion that existing methods. In this regard, it should be observed that each system behaves differently depending on the dataset: Lowe's method, for instance, performs excellently on the Rothganger dataset but poorly on the CS-17 dataset; similar observation can be made with Ferrari's method, which is inferior to our method on the Rothganger dataset but performs better on its own dataset.

In any case, we have shown that our strategy of selecting only a small fraction of the model features (the most efficient ones) in the detection lattice is valid. Besides, it seems extremely important to add other feature types in the recognition process. It indeed highly increases the robustness and improve the performance in noisy conditions.

Figure 4.14: Illustration of the robustness of our method to viewpoint change. (Top) The three training images of the teddy bear used to train the lattice (orientation is defined vertically for the three images, $n_{term} = 90$). (Bottom) Top scoring detections on the Rothganger dataset using the lattice thus created (small green circles indicate the position of detected aggregates, large purple circles indicate the hypothesized object location and pose). Our method shows an impressive tolerance to viewpoint change (the first negative detection comes with a score of 16, after the 11^{th} positive image). The quality of the 10^{th} detection is questionable due to bounding box misplacement. The next detections (with scores inferior to 16) are all true negatives.

Chapter 5

Extension of the Multi-feature Incomplete Graph Matching to Recognition of Class Objects

Contents

In this chapter, we present an extension of the method described in Chapter 3 for detecting classes of object instead of instances of specific objects. This development is motivated by the fact that, in our opinion, the part decomposition and detection

achieved respectively by the lattice construction and detection procedures of Chapter 3 also hold for object classes.

We begin by presenting the general principles of our method and similar methods from the state-of-the-art. Then, we introduce the modifications made to the original approach of Chapter 3. These modifications include the way in which the lattice is trained and the way in which the objects are detected. More specifically for the latter part we use a discriminative classifier (real-AdaBoost) instead of the probabilistic model originally proposed in Chapter 3. We also describe an optimization to fasten the training.

5.1 Introduction

Although the method presented in Chapter 3 is designed for specific object recognition, we have reason to believe that it can also apply to class object recognition. In fact, many works have highlighted the fact that class instances share *at least* local similarities [FPZ03, VNU03, JT05]. This appears to be consistent with the intuition for object classes like cars or faces, for which semantic parts like car wheels or eyes are shared between all class instances – despite some minor variations in the appearance of these parts.

Because (i) our detection lattice is designed so as to detect *similar parts* to the ones found on training images and that (ii) the final constraint on the spatial arrangement of these different parts is rather loose, it is reasonable to think that it could also well apply to the case of class object recognition. To check this up, we have trained our lattice with a few images from the Caltech-101 dataset [FFFP06] which, as the names indicates, contains sample images of objects belonging to 101 different categories (i.e. object classes). Examples of training images and qualitative recognition results with the unmodified lattice of Chapter 3 are shown in Figure 5.1. As can be observed, the detection lattice is well able to cope with small shape variations and various noises. However, additional tests (not shown here) have also demonstrated that the method of Chapter 3 *unchanged* produces poor results on more complicated cases. As a result, we have decided to *adapt* the original method of Chapter 3 to the case of class objects. The corresponding set of modifications and the resulting recognition algorithm are presented in this chapter.

In the rest of this section, we first briefly introduce the outline of the proposed method. Then, we discuss about related works and justify the proposed algorithm based on solid evidences found in other papers.

5.1.1 Method overview

Globally, the method presented in this chapter takes advantage of two existing general frameworks for class object recognition:

- Concerning the feature part, we lean upon the frequent pattern mining framework.
- Concerning the structure of the detection algorithm, we adopt the bag-of-features framework.

A description of those frameworks along with a review of related papers is performed in the next subsection.

Figure 5.1: Test of the method described in Chapter 3 with class objects instead of specific objects. A few object classes with relatively low intra-class variations were picked up from the Caltech-101 dataset [FFFP06]. Top: training images for each class (for the negative training set, the 20 first images of the Caltech-101 "background" folder are used); bottom: recognition examples. We only show here detections supported by an important number of votes (i.e. more than 10).

5.1.2 Related works

Frequent Patterns Mining

Recently, a number of papers [CYHH07, NTU*07, QFLG07] have shown that frequent patterns can be powerful structures for complex classification tasks. In a general context, a *pattern* is defined as a higher-level combination of low-level features. For instance, in Cheng et al. [CYHH07], each sample to classify is represented by a set of binary attributes; in this case, a pattern is the union of several binary attributes, i.e., a subset of attributes. Then, a *frequent pattern* is a pattern which is found in a large number of training samples.

More formally, the (relative) support of a pattern α on a sample set $\mathcal{S}$ is defined as $D_\alpha = |\{s \in \mathcal{S} | \alpha \subseteq s\}| \,/\, |\mathcal{S}|$, where $\alpha \subseteq s$ denotes the fact that the pattern α is present in sample s (note that $0 \leq D_\alpha \leq 1$). Interestingly, Cheng et al. [CYHH07] have demonstrated using both mutual information and Fisher's Score metrics that frequent patterns provide on average more information for the classification task than low-level features do, as long as the pattern frequency is in a given range (i.e. $\theta_{min} \leq D_\alpha \leq \theta^{max}$). In fact, patterns that are too much frequent can not have highly discriminative power as they are indistinctly found both in class and non-class samples. Symmetrically, rare patterns are also of low interest for classification (in fact, using them could even harm the classification due to over-fitting effects). Note that similar observations were made before by other authors like Jurie and Triggs [JT05] although no in-depth study was provided at the time.

Subgraph as a pattern

In the object recognition context, low-level features often take the form of local regions in the broad sense, e.g. keypoints, edges or image regions. As noticed by a large number of researchers [KK91, LSP04, ZBMM06, CFJV06, FSGD08, TWLB10], it appears natural to organize those features in the shape of a graph, as a graph provides an effective and compact representation of the image based on both local appearances (using graph nodes) and spatial relationship (using graph edges). In this paradigm, image patterns can be then simply defined as subgraphs.

Frequent subgraph mining has already been investigated by several authors. For the general case of attributed relational graph, we can cite the pioneer work of Kuramochi and Karypis [KK01]. In the object recognition field, we can cite for instance Nowozin et al. [NTU*07], Quack et al. [QFLG07] or more recently Ozdemir and Aksoy [OA10] and Zhang et al. [ZYW*10]. In all those works, although the graph framework may not be

explicitly mentioned, frequent patterns take the form of combination of nearby features and are extracted using data-mining techniques, i.e. after discretizing and quantizing the training images into discrete structures so as to fit traditional database algorithms.

The two latter works are the most similar to the method presented in this chapter. In their approaches, the authors convert the training images into graphs of quantized keypoints, and frequent subgraphs are used for classification purpose (for instance, histogram of subgraphs are computed in the bag-of-features fashion). In our opinion, the problem in those approaches lies in the discretization and quantization steps required to convert images into finite, small graphs. This conversion indeed implies losses in terms of non-salient information, which is known to be valuable for classification [JT05, VGVZ09]. On the other hand, it is intractable to create graphs based on densely-sampled features, because they come in great number and the resulting graph size and complexity would prevent any practical utilization. In this perspective, our detection lattice appears as the right trade-off between the two options: in our framework every subgraph is ensured to contain at least a single salient feature (i.e. a keypoint), so as to achieve fast detection, but the other subgraph features can be densely sampled in the image.

Using frequent subgraphs for classification

So in all above mentioned papers as well as in this chapter, subgraphs are used for classification. However, it is well known that the number of existing subgraphs for a given graph is exponential with the graph size. As a result, it is intractable to use the multitude of them for classification. One solution is to purposely reduce the graph size, like in [ZYW*10] where graphs with only 80 nodes describe single images, but this may harm the performance as explained above. A more widely used solution is thus to select a subset of all possible subgraphs based on a heuristic procedure. Such procedures derive generally from the Sequential Forward Selection (SFS) algorithm [CYHH07, OA10] and/or Depth First Search (DFS) procedure [KMM05]. In both case, the insight is to iteratively select those subgraphs that bring the most information with respect to the classification task, while avoiding redundancy inside the set of selected subgraphs. For this purpose, subgraphs are generally enumerated from smallest to biggest while in the same time different pruning strategies based on minimum information gain are employed [KMM05, NTU*07, CYHH07].

In this chapter, we use a slightly modified version of the lattice construction procedure of Chapter 3 as subgraph selection algorithm. It is in fact similar in its structure to

the kind of existing algorithms mentioned above: the space of all possible subgraphs is explored and pruned at the same time; and mutual information enables simultaneously to find efficient subgraphs and to avoid redundancy. The fact that more positive training images are available in the class case than for the specific object case indeed makes our algorithm to select frequent subgraphs: remember that the negative images are modeled with a single value in the coverage map (Section 3.5.5).

Finally, note for completeness that there exists other strategies for classification with frequent subgraphs such as the marginalized kernel of Kashima and Tsuda [KTI03]. In their work, two graphs are compared by an infinite pairwise comparison of random walks, which amounts in fact to compare them in terms of common subgraphs. This appears unfortunately hardly applicable for our continuous graph framework.

The bag-of-features model

Finally, we use the bag-of-features as classification model. Since the bag-of-features model has already been presented in Chapter 2, we just sum up here that it consists in representing an image as a set (or an histogram) of quantized visual words. Despite the fact that spatial relationship between visual words are lost in the binning process, the bag-of-features model performs surprisingly well. In addition, this model is easy to train, easy to use, and works well in practice for a large number of object categories. So we decided to use it as well in our class object recognition system; however, instead of coupling it with a SVM classifier as is generally done [JT05, ZBMM06], we prefer AdaBoost as it inherently gets rid of over-fitting problems by selecting a sparse subset of features (so that there are two successive feature selections in our method: the first one during the lattice construction and the second one done by AdaBoost). In fact, the association subgraph + AdaBoost has already been proved to be extremely powerful in term of classification accuracy in Kudo et al. [KMM05]. Another choice would have been to choose a linear SVM, as it also somehow performs a feature selection through the weight attributed to each feature (e.g. see Cheng et al. [CYHH07], Jurie et al. [JT05]). However, as noticed by [ZYW*10], SVM with graph kernel are less interpretable easily and usually cost more computations.

5.1.3 Chapter outline

To sum up, we introduce in this chapter an extension of the method described in Chapter 3 to the case of class object recognition. The chapter content is as follows: firstly, we describe in Section 5.2 the general frame of the proposed method, which follows the

classical scheme in this domain (i.e. sliding window with feature extraction and classifier). Then in Section 5.3 we detail and justify modifications to the original method in order to detect classes of objects instead of instances of specific objects. Those modifications include changes in the lattice training and detection procedures as well as in the final decision model. Finally, we conclude in Section 5.4.

5.2 Method Description

We describe in this chapter a method for the joint detection and localization of class objects in natural images. The details of our approach are best explained by considering the recognition step first. The next section will introduce how to build a detection lattice for the class case, but for now assume that the lattice already exists. Following the literature mainstream, we divide the detection problem into two nested tasks:

- The first one is to decide if an object instance belonging to the learned class is present in the input test image or not (i.e. classification task). That is, we explicitly require the object size to roughly match the image size.
- The second task is to achieve object detection and localization using the first system. Classically, we scan the input image at several positions and several scales, and each sub-image (i.e. *window*) is classified by the first system.

We now describe in more details the resulting system as a whole.

Sliding window

Without loss of generality, let us consider in the following a single window $\mathcal{W}$ extracted from an input image $\mathcal{I}$ at a given position and scale, which have no importance (we assume that the window aspect ratio is known by advance). At detection time, visual words and aggregates are detected in $\mathcal{W}$ using respectively a standard keypoint detector followed by nearest neighbor search in a pre-computed visual codebook, and a pre-computed detection lattice. Then, features are extracted from those detected words and aggregates and are gathered into a feature vector. Finally, the feature vector is examined by a classifier which evaluates the likelihood of the object presence in the window. Note that we do not use contextual information based on global image features combined with the window position and scale like [MTEF06, HJS09]. In other words, all windows are processed independently. A final non-maxima suppression step is applied after that all windows of image $\mathcal{I}$ have been processed, in order to remove redundant overlapping detections.

Classification and Training procedure

Our general approach for training follows a classical scheme in the literature: first we generate a large number of features; then we use a feature selection process and finally we train a classifier from those selected features. The feature selection step in our case is combined with the classifier training step as we use AdaBoost coupled with decision stumps. Hence our purpose for the first step is to generate weak classifiers, i.e., features that are at least *weakly correlated* with the ground truth. In our case, the features correspond either to the histogram bins of the bag-of-features model or to the binary presence or absence of learned model parts. That is, for the latter case our lattice is used to detect frequent aggregates (i.e. model subgraphs) distinctive to the class. Hence, our method is related to the graph boosting of [ZYW*10] or classification using frequent pattern of [CYHH07], although our detection lattice searches for frequent subgraphs in the continuous image space instead of a discretized space.

In the following, we first describe the features extracted for a window $\mathcal{W}$, keeping the same notation as in Chapter 3. Then, we detail the classifier used and its training procedure.

5.2.1 Features used

Basic features As stated above, we consider two sets of basic features to describe the content of a window $\mathcal{W}$:

- Real-valued features $\{f_n^W\}_{n=1}^{N_{words}}$ derived from visual words. Those features correspond to the bin sizes of the normalized histogram of visual words in the bag-of-features framework. Namely, keypoints are initially extracted in the image at salient locations along with their descriptor, then a code is associated to each of them according to their nearest neighbor in a pre-computed visual codebook of size N_{words}. (The codebook is computed beforehand using k-means on a set of descriptors extracted in natural images.) In our case, the feature f_n^W corresponding to code n is then defined as

$$f_n^W(\mathcal{W}) = \frac{\#\{\phi_i^W | \mathbf{c}_i^W \in \mathcal{W} \text{ and } \mathbf{z}_i^W = n\}}{\sum_j \#\{\phi_j^W | \mathbf{c}_j^W \in \mathcal{W} \text{ and } \mathbf{z}_j^W = n\}}$$

  where $\#\{\ldots\}$ is the cardinality of a set and $\phi_i^W \in \varphi_W$ is an instantiated visual word (i.e. $\phi_i^W = (\mathbf{p}_i^W, \mathbf{z}_i^W) \in \varphi_W$ with φ_W the visual word feature type, $\mathbf{p}_i^W = (\mathbf{c}_i^W, \mathbf{h}_i^W)$ the visual word position and $\mathbf{z}_i^W \in [1, N_{words}]$ the associated code with respect to

the codebook).

- Binary features $\{f_i^A\}_{i=1}^{N_{Aggr}}$ derived from aggregates. Each such feature is a binary value indicating the presence or absence of an aggregate A_i (i.e. a model part) inside $\mathcal{W}$. Contrary to visual words, aggregate features are composed of several types of local features (namely keypoints, edges and textures). Formally, we have

$$f_i^A(\mathcal{W}) = \begin{cases} 1 & \text{if } \exists A'_i \in \mathcal{W} \\ 0 & \text{otherwise.} \end{cases}$$

  In order to achieve scale invariance, we restrict the detected aggregates to the set of aggregates whose scale and position fit the current window $\mathcal{W}$. Formally, it amounts to define the property $A'_i \in \mathcal{W}$ as follows:

$$A'_i \in \mathcal{W} := \mathbf{c}_{A'_i} \in \mathcal{W} \text{ and } 0.5 < \frac{\sigma_{A'_i}^{hyp}}{\sigma_{\mathcal{W}}} < 2 \tag{5.1}$$

  where $\mathbf{c}_{A'_i}$ is the aggregate center and $\sigma_{A'_i}^{hyp} = \sigma_{ref}\frac{\sigma_{A'_i}}{\sigma_{A_i}}$ is the hypothesized object scale in the test image with respect to the object scale in the training images.

Complex features Next, we derive complex features from the previous basic ones:

- We define doublets of aggregates $\left\{f_{ij}^D(\mathcal{W})\right\}_{i,j}^{N_{doublets}}$ as complex features indicating the joint presence of two given aggregates A'_i and A'_j in the window. Doublet features are formally defined as

$$f_{ij}^D(\mathcal{W}) = \begin{cases} 1 & \text{if } \exists A'_i \in \mathcal{W} \text{ and } \exists A'_j \in \mathcal{W} \\ 0 & \text{otherwise.} \end{cases},$$

  where the property $\exists A'_i \in \mathcal{W}$ is defined as above. Doublets features have been shown to produce good results thanks to their increased distinctiveness in several works (see [SREZ05] and [TWL*10]). Moreover, they are cheap to compute in our framework as their definition is closely related to the definition of simple aggregate features. The only shortcoming is that their number is squared with the number of aggregates, so that we select only a limited number of them ($N_{doublets} \ll N_{aggr}^2$) based on a random sampling over existing doublets in the positive training images.

- Three real-valued features f_0^C, f_1^C and f_2^C derived from aggregate features and corresponding to the number of aggregates detected in the window normalized by

different values. Firstly, we define the unnormalized aggregate count $f_0^S(\mathcal{W})$ as

$$f_0^C(\mathcal{W}) = \sum_i f_i^A(\mathcal{W})$$

Then, we define $f_1^C = f_0^C / N_A(\mathcal{W})$ and $f_2^C = f_0^C / N_K(\mathcal{W})$ as the number of detected aggregates normalized by, respectively, the number of aggregates $N_A(\mathcal{W}) = \#\{A_i' | A_i' \in \mathcal{W}\}$ and the number of keypoints $N_K(\mathcal{W}) = \#\{\phi_i^K | \mathbf{c}_i^K \in \mathcal{W}\}$ detected in the window $\mathcal{W}$. Note that according to this definition, we currently have $f_1^C = 1$. However, when we divide the window into different cells $\mathcal{C} \subseteq \mathcal{W}$ using the spatial pyramid layout (see below), this eventually leads to separate instantiation of $\{f_0^C, f_1^C, f_2^C\}$ for each cell where $f_1^C(\mathcal{C}) \leq 1$.

The list of all features used is summarized in Table 5.1.

Spatial pyramid

We again derive new features inspired by the spatial pyramid framework from the features defined above. The spatial pyramid was proposed simultaneously by Lazebnik et al. [LSP06] and Grauman and Darrell [GD05] as an improvement of the bag-of-features model. It enables to capture, to some extent, the spatial layout of the features inside an image or a window (recall that the original bag-of-features model ignores the location of the features) by subdividing the window into overlapping cells and computing a separate histogram for each cell. It has been shown to significantly improve the performance while only bringing minor modifications to the model. In the original paper [LSP06], different weights are assigned to the cells due to their SVM classification scheme, but in our case (i.e. boosting) those weights become unnecessary.

As a result, we follows the same guideline and recursively subdivide the original window by a factor 2, leading to a set of overlapping cells $\mathcal{C} \subseteq \mathcal{W}$ (with 2^{2l} cells per level $l \geq 0$). Each feature described above is thus computed independently in every cell. This is in order to better model the spatial arrangement of the features (remember that each feature alone do not contain information about its position in the window). For three levels of the spatial pyramid, we then have 21 cells (one at the first level, 4 at the second level and 16 at the third level). As a consequence, the number of features would also be multiplied by 21 for three levels.

5.2.2 Window classification

Now that a set of features have been extracted for a given window, we must feed those features to a classifier that will decide if the window contains the learned class object or not. In Chapter 3, we have used a probabilistic generative model to take this decision (Section 3.4.6). However, we now turn towards a discriminative classifier for the class object case.

The reason behind this change is that the probabilistic model proposed in Chapter 3 is over-simplified - which is fine for a simple task like instance detection, where intra-class variance is small. On the contrary in the case of class objects, one has to find a way to precisely learn the complex decision boundaries between class and non-class in the feature space. In our case, we choose AdaBoost combined with decision stumps as classifier. This choice is motivated by several reasons: AdaBoost produces an interpretable classifier well suited to our task: selected features basically corresponds to model parts; a boosted classifier is extremely fast to evaluate because its evaluation simply amounts to a weighted sum of binary variables[1]; and finally it only selects the features that are useful for discriminating class and non-class windows. Note that this is important as the set of features extracted earlier may be over-complete, especially concerning the aggregate features. (We will see later that the detection lattice is purposely constructed so as to extract a great deal of aggregates.) We now give a formal description of the classifier and its training.

AdaBoost

AdaBoost is a classifier originally proposed by Freund and Schapire [FS95]. It has been shown to yield excellent classification accuracy, generally equivalent to support vector machines (SVM). As said above, its main advantage with respect to SVM is that it can select a subset of features from the huge feature pool available during training. Formally, the classifier takes the form of a real-valued decision function expressed itself as a sum of weak classifiers:

$$H(\mathbf{x}) = \sum_t h_t^*(\mathbf{x})$$

where $\mathbf{x}$ is a feature vector to classify and $h_t^*(\mathbf{x})$ is a weak classifier selected at training iteration t. In our case, $\mathbf{x}$ corresponds to the features extracted in a window $\mathcal{W}$ and in the following we will replace $\mathbf{x}$ by $\mathcal{W}$ in a slight abuse of notation. In addition to the reasons mentioned above, we choose to use decision stumps as weak classifiers for their

[1] It can be further accelerated at almost no cost using a soft-cascade like Bourdev and Brandt[BB05]

Table 5.1: Summary of the features used in our classifier (Real-AdaBoost).

Description	Type	total number	really used	number of stumps
histogram of visual words f_n^W	array of real	$N_{words}N_{cells}$	$N_{words}N_{cells}$	$10N_{words}N_{cells}$
single aggregates f_i^A	binary	$N_{cells}\,\lvert\mathcal{L}_E\rvert$	$N_{cells}\,\lvert\mathcal{L}_E\rvert$	$N_{cells}\,\lvert\mathcal{L}_E\rvert$
doublets of aggregates f_{ij}^D	binary	$(N_{cells}\,\lvert\mathcal{L}_E\rvert)^2$	$N_{doublets}$	$N_{doublets}$
# aggregate per cell f_0^C	real	N_{cells}	N_{cells}	$100N_{cells}$
# aggregate per cell / $N_A(\mathcal{W}) = f_1^C$	real	N_{cells}	N_{cells}	$100N_{cells}$
# aggregate per cell / $N_K(\mathcal{W}) = f_2^C$	real	N_{cells}	N_{cells}	$100N_{cells}$

simplicity and the fact that they are compatible with our binary features. Note that we also tried to use a SVM classifier instead of AdaBoost, but the results were not satisfying.

Decision stumps

A decision stump is a minimal decision tree composed of only one root and two leaf nodes. In Real-AdaBoost, a widely used variant of AdaBoost, decision stumps are defined as follows:

$$h_i(\mathcal{W}) = \frac{1}{2}\log\frac{p_{\mathbf{w}}(\mathcal{W}\in\mathbb{W}^+|f_{h_i}(\mathcal{W}))}{p_{\mathbf{w}}(\mathcal{W}\in\mathbb{W}^-|f_{h_i}(\mathcal{W}))}$$

where $p_{\mathbf{w}}$ is a probability subject to weight vector $\mathbf{w}$ (see below), $\mathbb{W} = \mathbb{W}^+ \cup \mathbb{W}^-$ is the set of training windows, respectively positive windows and negative windows and f_{h_i} is the binary feature corresponding to the weak classifier h_i. Because we also deal with non-binary features (e.g. f_0^C), we integrate those ones in the training pool using additional binary features which are generated by comparisons of the original real-valued feature against fixed thresholds. The thresholds are sampled on the distribution of values observed in the training set for each real-valued features, as is classically done. The number of thresholds (i.e. resulting binary features) chosen for each real-valued feature is summarized in the rightmost column of Table 5.1.

5.2.3 Optimization for training the classifier

Training procedure During training, a subset of the stumps extracted previously is chosen to form the final classifier. The size of the subset is determined by validation on a separate training set. Empirically, the number of selected stumps is always much lower than the number of available stumps, resulting in a compact and fast classifier.

More precisely, in each iteration t of boosting, the best stump $h_t^* \in \{h_i\}_{i=1}^{n_{stumps}}$ is selected as the one minimizing the score $Z_{i,\mathbf{w}}$ on the training set:

$$Z_{i,\mathbf{w}} = 2\left(\sqrt{p_{\mathbf{w}}\left(\mathcal{W} \in \mathbb{W}^{+} | f_{h_i}(\mathcal{W}) = 1\right) p_{\mathbf{w}}\left(\mathcal{W} \in \mathbb{W}^{-} | f_{h_i}(\mathcal{W}) = 0\right)}\right.$$

$$\left. + \sqrt{p_{\mathbf{w}}\left(\mathcal{W} \in \mathbb{W}^{-} | f_{h_i}(\mathcal{W}) = 1\right) p_{\mathbf{w}}\left(\mathcal{W} \in \mathbb{W}^{+} | f_{h_i}(\mathcal{W}) = 0\right)}\right) \tag{5.2}$$

The weighted probability $p_{\mathbf{w}}(\mathcal{S})$ mentioned earlier is formally defined as

$$p_{\mathbf{w}}(\mathcal{S}) = \frac{\sum_{n=1}^{n_{samples}} \delta(\mathcal{W}_n \in \mathcal{S}) w_n}{\sum_{n=1}^{n_{samples}} w_n}$$

where $\mathcal{S} \subseteq \mathbb{W}$ is a subset of training windows and $\delta : \mathbb{B} \rightarrow \{0, 1\}$ is the Kronecker delta. Once the best stump h_t^* is selected, the weights in $\mathbf{w}$ are updated for each training sample according to the following formula:

$$w_n = w_n \exp\left[h_t^*(\mathcal{W}_n)\left(2\delta(\mathcal{W}_n \in \mathbb{W}^{+}) - 1\right)\right]$$

More details for the training procedure of real-AdaBoost can be found in [SS98, WZ01].

Optimization During our training, the number of decision stumps n_{stumps} in the training pool typically ranges between 40,000 and 100,000 (i.e. it corresponds to about 1,500 aggregates in the lattice with three levels of the spatial pyramid). Because in each boosting iteration, every stump is evaluated with respect to every training window (5.2), it is important to use a data structure making the evaluation of a stump on a given window as fast as possible. Initially, we implemented a tree map structure to store the detected aggregates and visual words for each window. This makes sense, since the set of detected aggregates and visual words is sparse: a tree map structure saves memory by allocating memory only for the detected features. Testing the presence of an aggregate or a visual word amounts then to search for its hash-code in the tree map, which is efficient because in $O(\log n)$. (The other complex features are small in number, so we will neglect them in this analysis.) While this intuitive choice appears valid, it becomes problematic regarding the training time: in practice using this scheme causes hours of training because of the huge number of such atomic searches.

A simple solution, way faster, exists: to pre-compute by advance the binary matrix $\mathbf{F}$ telling about the output of each stump for each sample (hence $\mathbf{F}$ has $n_{samples}$ rows and n_{stumps} columns) in order to avoid looking each time in the tree map (i.e. $O(1)$ instead of $O(\log n)$). Because this matrix is huge, it is not feasible to store it entirely. Instead, we decompose it into a set of lists, where each list L_i ($1 \leq i \leq n_{stumps}$) stores non-null values

in the i-th column of $\mathbf{F}$. Because $\mathbf{F}$ is sparse, the lists are lite in memory. Using this new configuration, it takes about one minute to learn the classifier for 5000 training examples in total and 50,000 stumps (including 2-fold cross-validation). In comparison, the SVM training time for the same task is approximately equivalent (although additional training iterations are required for a SVM in order to find the optimal constant C).

5.2.4 Optimization for detection speed

As we saw, AdaBoost performs feature selection when building the classifier. In other words, only a few features from the training pool are kept in the final classifier $H(\mathcal{W})$, and the other ones are discarded. According to our features (see Table 5.1), the situation is the following:

- If at least a single feature $h_t^*(\mathcal{W})$ related to f_0^C, f_1^C or f_2^C is selected in $H(\mathcal{W})$, then every aggregates A_i in the lattice is necessary at detection time. Indeed, those three features count the number of all detected aggregates fitting the current window $\mathcal{W}$ indifferently of their identity.

- Conversely, if neither f_0^C, f_1^C nor f_2^C are selected in $H(\mathcal{W})$, then every aggregate which is not on the path of a useful aggregate becomes useless at detection time. We call a useful aggregate A_i an aggregate which is selected in the final classifier (either because a related single f_i^A or doublet f_{ij}^D feature is selected in $H(\mathcal{W})$).

As a consequence, we straightly see how the lattice can be pruned when the second situation is true. Because most aggregates in the training lattice are not selected in the final classifier, the pruning reduces the lattice size by a large factor, which results in a significant gain in detection speed. In the next chapter we experiment pruning the lattice after purposely discarding the three aggregate count features f_0^C, f_1^C and f_2^C in order to study the effect in terms of detection performance and speed. In particular we show that a significant boost can be obtained without losing performance.

5.3 Modifications to the original lattice

We detail and justify in this section the modifications applied to the framework of Chapter 3 in order to detect aggregates in the case of class object detection instead of specific instance detection. Those modifications include minor changes in the lattice training and detection procedures.

5.3.1 Rotation variance

Several works have shown that the rotational invariance property of keypoints is, at best, useless or, more often, harms the performance in the case of class object recognition. Indeed class instances are most of the time all in the same orientation in test or train images (e.g. pedestrians walk vertically, feet on the ground). Besides, the way in which keypoint rotational invariance is achieved demands an important amount of local similarity. For object classes, where the intra-class variation is high, this invariance is unstable.

As a consequence, we drop the rotational invariance for SIFT keypoints in our recognition lattice (i.e. both for training and testing). Namely, we define the radial vector for any keypoint ϕ_i^K as being equal to $\mathbf{h}_{\phi_i^K} = (\sigma_{\phi_i^K}, 0)$ (the aggregate position and growth remain computed in the same way as before). Our experiments have shown a significant improvement of the performances following this simple modification. Of course, visual word features are also extracted from SIFT keypoints without using the orientation.

5.3.2 Recognition procedure for the lattice

In Chapter 3, we have set up a detection lattice for localizing in the test images exactly the same object parts (up to some noise) than the ones learned in the training images. Each terminal aggregate was responsible for detecting a model part with a high distinctiveness, and this distinctiveness was achieved by a succession of micro-classifiers until termination of the corresponding lattice path.

In the class case, however, larger differences between parts can be expected since the intra-class variation is known to be more important. For this reason, it becomes less clear which level of distinctiveness is necessary to detect each part: some parts might remain extremely similar between class instances while other parts might be only loosely similar. As a consequence, we decide to consider all possible levels of distinctiveness for each lattice path, letting the classifier decide which one is the most useful at training time.

In other words, we output detections at all levels of the lattice (and not only at terminal levels): since we know that the distinctiveness of a lattice path gradually augments as the path grows (i.e. as micro-classifiers are added), we output one detection per traversed branch. An illustration of this behavior is shown in Figure 5.2. In this example, 3 detections are output corresponding to 3 aggregates of intermediate levels (note in particular that the terminal aggregate is not detected because the last feature is missing in the test image). A direct consequence of this modification is that a lot more aggre-

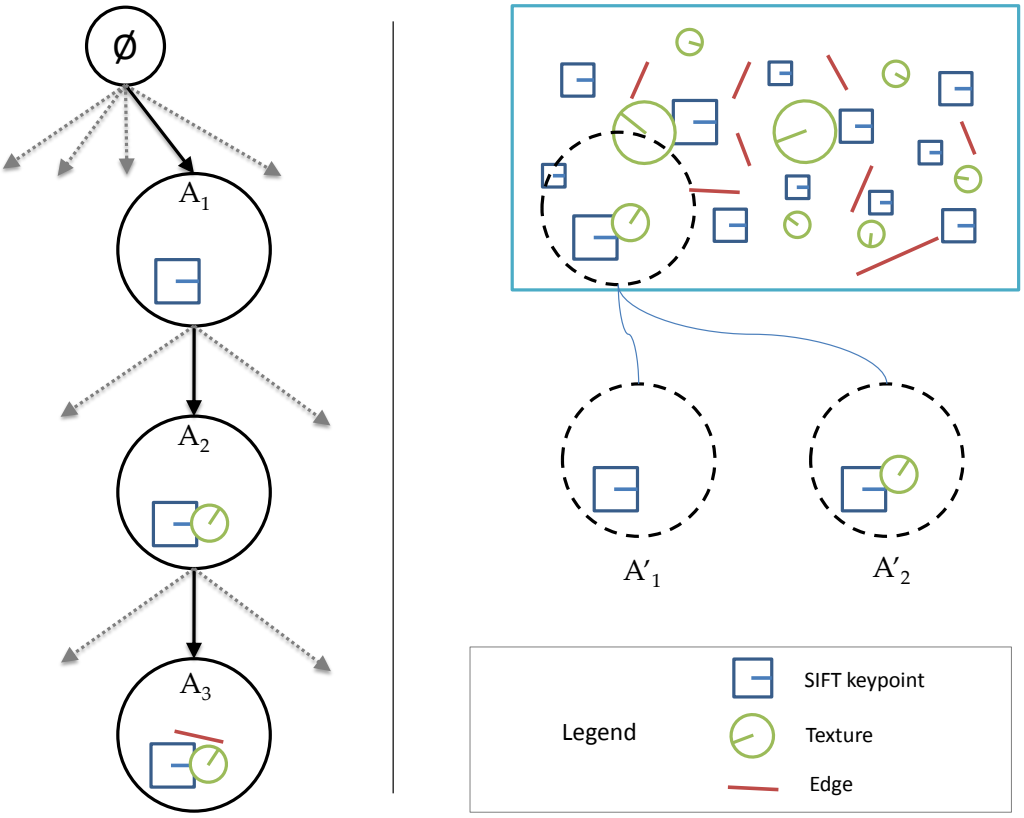


Figure 5.2: Illustration of incomplete aggregate detections. Left column: example of one lattice path terminated at level 3 and composed of the model aggregates A_1, A_2 and A_3. Right column: at top, a test image viewed as a collection of local features (here keypoints, textures and edges). Two aggregates are detected in the bottom-left corner of the image corresponding to the incomplete model aggregates A_1 and A_2. The terminal aggregate A_3 is not detected because the third feature (an horizontal edge) is absent from the test image.

gates are detected in each image. This is not a problem insofar as AdaBoost is able to drop useless aggregates and can take a fast decision based on a sum of extremely simple decisions (presence or absence of each such detected aggregate).

In the same line of thinking, we purposely raise the micro-classifier threshold $d_{K^z}^{max}$ of the seed branches (see Section 4.2.1). Originally, this threshold is set to a low value so as to eliminate most candidate keypoints in the test image, and thus achieve fast detection. The value chosen in Section 4.2.1 technically amounts to prune 99.9% of the pairs (image keypoint, seed branch). Because this selectivity does not fit the intra-class variations and also because we want to generate lattice paths with more uniformly distributed selectivity of micro-classifiers, we raise the threshold so that only 95~99% of the pair (test keypoint, seed branch) are pruned. Experiments in the next chapter are dedicated to studying the impact of the variation of $d_{K^z}^{max}$ within such range. Note that

the rest of the thresholds for the other kernels remains unchanged. We indeed make the simplifying assumption that intra-class variations of local model parts are equivalent in terms of effects to the various noises used for computing those thresholds (see Section 3.5.3).

Comparison with the probabilistic model of Chapter 3 In the probabilistic model described in Section 3.4.6, the probabilities $p(e'_{jk}|\{e'\}^{jk-}, \mathcal{V}) / p(e'_{jk}|\{e'\}^{jk-})$ were estimated independently for each lattice path. Morevoer, an additional independence assumption between lattice branches lying on different paths was made to simplify the evaluation of $p(\mathcal{O}|\{A'_i\})$. On the contrary, in our case AdaBoost is able to implicitly learn the joint probability distribution $p(\mathcal{O}|\{A'_i\})$ conditioned on the presence of every model aggregates in the lattice (although this is not true strictly speaking, as AdaBoost only estimates a discriminative function and not a probability distribution). This is made possible by the fact that a lot more training images are available, so that the classifier does not overfit. Morevoer, the fact that *every* lattice aggregate can be selected in the classifier (and not only terminal ones) allows AdaBoost to tune the adequate level of disctinctiveness corresponding to each model part (for the same lattice path, i.e. model part, a longer path is more distinctive that a shorter one). This choice has been shown to improve the detection performance by a large amount in our experiments. Morevoer, to our knowledge no similar mechanism has yet been proposed in the computer vision community (i.e. selecting the distinctiveness dynamically when building the final classifier, and not only statically beforehand).

5.3.3 Training procedure for the lattice

As we saw above, the recognition lattice is used to detect aggregates in the test images. What we want is, as before, to manage to build the lattice in such a way that the detected aggregates are correlated as much as possible with the ground truth. So similarly to Chapter 3 we train our lattice from a set of class images (rather than several pictures of the same object) and a set of negative images (background images). However, the fact that the training images can not be precisely aligned when building the prototype graphs (Section 3.4.1), and our new objective of generating a large pool of training features (i.e. lattice aggregates) rather than a fixed number of terminal aggregates imply a few modifications of the training algorithm. To summarize, we basically used almost the same procedure, except for the following steps:

Hypothesis Extrapolation During training, recall that the position of the detected aggregates is checked to ensure that they enable a correct extrapolation of the known object bounding box. If the extrapolation is erroneous[2], then the detection is ignored in the calculation of the information gain. In Section 3.4.6, we used criterion (3.9) to ensure that the extrapolation is correct up to some extent. For a similar reason than exposed before, the aggregate orientation and position become unreliable in the class case. So instead of extrapolating the bounding box using a 2D similarity, we simply check that the detected aggregate center remains in the known bounding box and that the extrapolated scale is correct. In other words, we just require the model aggregates to detect locally similar model parts at similar scales (we do not care about their position or orientation). As a consequence, we use the criterion (5.1) already defined above in Section 5.2.1 instead of the original criterion (3.9).

Lattice path termination In Chapter 3, the condition for a model aggregate to become terminal was based on the fact that it does not generate any detection in the set of negative training images. In the class case, this can be a problem as the number of negative training images is fixed and generally large (typically, hundred images), which can prevent lattice paths from terminating before a high level. Although this is not a problem in theory, as every path will eventually terminate with a large number of features (giving extremely high distinctiveness to the path), in practice it makes the training last very long. Our solution to this issue is to change the termination criterion: to terminate a path, we simply require that the number of false detections generated by its terminal aggregate A_k is small in proportion:

$$p(A_k|\neg\mathcal{O}) = \frac{\#\{A'_k|A'_k \in \mathbb{I}^-\}}{\#\{\phi_i^K|\phi_i^K \in \mathbb{I}^-\}} < p^{max}$$

This proportion is calculated by dividing the number of false detections by the number of seed features (i.e. the number of trials) in the negative images. This parameter is also subject to investigations in the experiments in the next chapter. In practice, we found out that $p^{max} \approx 10^{-4}$ was producing good results (it corresponds to 5 features per aggregate on average).

End of lattice construction In Chapter 3, the condition for stopping the lattice construction was based on a threshold on the number of detected aggregates in the positive images (i.e. minimum coverage). In the case of classes, the objective is slightly different

[2]This typically happens for an aggregate detected on an image where the object have a sufficiently different pose from the image from which the aggregate was created.

as we want to generate as many training features (i.e. aggregates) as possible, letting AdaBoost decide which ones are pertinent. Because an infinity of aggregates is not desirable either, we must define some criterion to generate as many aggregates as possible but avoiding too much redundancy between them.

For this reason, we limit the number of candidates retained in each level based on a minimum gain in mutual information. Moreover, we ignore false detections in this calculation of the added mutual information because we reckon that what is important at this point is to properly cover the model images with detected aggregates, rather than finding aggregates with a good correlation (those ones with best correlation are picked first in the loop anyway). Formally, steps (4.d-4.e) in the training loop of Section 3.5.2 are replaced as follows:

[4] Loop until all candidates have been picked:

(a) Pick the best candidate $A_k \in \{A_j \in \mathcal{L}(l) | \neg \text{picked}(A_j)\}$ according to the mutual information (see Section 3.5.4).

(b) Set picked(A_k) to true.

(c) Detect model aggregate A_k in each training images, leading to a set of detection $\{A'_k\}$.

(d) if A_k is detected in strictly less than two training images: remove A_k from the candidates and return to step (4.a).

(e) if $\text{gain}^+(\mathbf{A}_k|\mathbf{M};\mathbf{O}) < gain^+_{min}$: remove A_k from the candidates and return to step (4.a).

(f) If $p(A_k|\neg\mathcal{O}) < p^{max}$, then A_k becomes terminal.

Step (4.d) is added to prune non-frequent aggregates, whose information gain for the detection task has been shown to be bounded by a low value and are most likely useless for training [CYHH07]. Step (4.e) purpose is to evacuate redundant aggregates. This is again similar to heuristic algorithm of feature selection as the MMMRFS algorithm of [CYHH07].

The loop stops when the maximal level is reached (we typically use a maximum of 6 levels) or when there are no more candidates. gain^+ measures the maximum gain in mutual information only computed from positive detections for a single training image:

$$gain^+(\mathbf{A}_k|\mathbf{M};\mathbf{O}) = \max_{\mathcal{I}\in\mathbb{I}^+} \text{MI}^+_{\mathcal{I}}(\mathbf{M},\mathbf{A}_k;\mathbf{O}) - \text{MI}^+_{\mathcal{I}}(\mathbf{M};\mathbf{O})$$

with

$$\mathrm{MI}^{+}_{\mathcal{I}}(\mathbf{M};\mathbf{O}) = \sum_{m=\{0,1\}} p(\mathbf{M}_{\mathcal{I}} = m, \mathbf{O}_{\mathcal{I}} = 1) \log \frac{p(\mathbf{M}_{\mathcal{I}} = m, \mathbf{O}_{\mathcal{I}} = 1)}{p(\mathbf{M}_{\mathcal{I}} = m)p(\mathbf{O}_{\mathcal{I}} = 1)}$$

where $\mathbf{O}_{\mathcal{I}}$ and $\mathbf{M}_{\mathcal{I}}$ denotes the random variable previously defined restricted to a single training image $\mathcal{I}$. The reason behind why we restrict the computation of the gain of information to the maximum over all training images is two-fold: firstly, it makes the threshold $gain^{+}_{min}$ independent of the number of model images; secondly, it avoids to support aggregates that are of little good for a lot of training images. In other words, we advantage aggregates that are very good on a small subset of training images, because they are generally more distinctive and meaningful than the ones discovered in all training images (and usually discovered as well in a large number in negative images).

5.4 Conclusion

We have presented a novel framework for class object detection and localization derived from the lattice described in Chapter 3. This approach combines the advantages of several previous works, i.e. the bag-of-features and the part-based models. A simple but robust (sliding window plus classifier) scheme is used for the detection. The originality of our method is that different levels of distinctiveness for each model part are proposed to the classifier (i.e. every aggregates in the lattice). To sum-up, we rely both on generic features suited to detect any classes (including the background) and on class specific features dedicated to a single class. Dedicated features are generally slower to compute, but in our case the cascaded lattice enables a relatively fast detection. A simple way to further accelerate the detection speed would be to use soft-cascades in the boosted classifier.

Chapter 6

Evaluation of Our Contribution For Class Object Detection

Contents

This chapter presents a qualitative and quantitative evaluation of the contribution presented in the previous chapter, i.e. an approach for class object recognition using subgraphs. We begin by presenting existing datasets available for that purpose and we select some of them with respect to our objectives. On the first dataset composed of four single classes we tune and evaluate our approach in terms of object recognition and image classification. On this occasion we quantitatively compare our results against a baseline and a part-based model from the state-of-the-art. Secondly, we answer the image classification challenge proposed for the Pascal VOC 2005 dataset. We show that our method is able to achieve results slightly inferior, similar or even superior to other state-of-the-art approaches for a smaller detection time. Moreover, the comparison with

related approaches of subgraph boosting clearly turns in our favour with improvements of 10% in term of EER measure.

6.1 Introduction

6.1.1 Existing datasets

There is a large variety of datasets available to evaluate class object detection systems. Among the most popular ones, we can cite:

- Whole series of Pascal Visual Object Challenges [EGWZ05, EVGW*] (Pascal VOC 2005, 2006, 2007 and 2008). The goal of these challenges is to recognize objects from a number of visual object classes in realistic scenes (i.e. not pre-segmented objects). Evaluation is either performed in terms of image classification (i.e. classify a test image to a given class means that at least one instance of the class is present in the image) or object recognition (i.e. detect and localize every instances in the test images) performance.
- The Graz dataset [OPFA06] and the Xerox dataset [WAC*04], which have essentially similar characteristics.
- The Caltech-101 and Caltech-256 datasets [FFFP06, GHP07] concern the image classification task only (only one object per image filling most of the image frame).
- Some separate datasets concerning single object classes like "airplanes" or "car rears" are available on the web page of the Computational Vision Group of California Institute of Technology [Com].

Note that this list is not exhaustive and that there are many other datasets, in particular in the field of Content-Based Image Retrieval (CBIR) where the topics are closely related to object recognition and image classification.

6.1.2 Purpose of this chapter

There are two purposes for this chapter. First of all, we wish to study the influence of the different parameters listed in Chapter 5. In particular, we want to inspect the parameter effects in terms of detection performance and speed. Then, we also want to compare our results against the baseline and comparable approaches from the state-of-the-art.

Single object classes

As a first step, we need to evaluate the effect of the three free parameters $d_{K^z}^{max}$ (eq. (3.8), Section 4.2.1), p^{max} and $gain_{min}^{+}$ (Section 5.3.3). Because there are a large number of combinations of their possible instantiations, we purposely select a few datasets which

are limited in size and thus fast to evaluate. We choose four separated datasets consisting of single object classes: we use the "horses", "car rears", "motorbikes" and "airplanes" datasets from [BU02] and [Com]. These four categories are also well representative of standard categories in other datasets like Pascal [EVGW*].

Image classification

In a second step, we evaluate our approach on the Pascal 2005 dataset in term of classification performance, in order to compare our results to state-of-the-art results. In particular, we want to compare to the system of Zhang et al. [ZYW*10] as their method is closely related to ours (i.e., boosting with subgraph decision stumps). It would have also been interesting to evaluate our contribution in term of object recognition performance (i.e. instead of classification) on a larger dataset like Pascal VOC 2007, but due to lack of time we must report this evaluation for later[1].

6.2 Experiments on Single Classes

As stated above, we evaluate our system using four object classes as a first experimental step. In more details, the four classes are the following:

- The Weizmann horses dataset (327 class images, 900 non-class images) from [BU02].
- Three classes from the Caltech Computational Vision Group web page [Com]:
  - The "airplanes" dataset (1074 images)
  - The "car rears 127" dataset (127 images)
  - The "motorbikes" dataset (798 images)

For each of the three latter classes, we use the "background" dataset from [Com] as non-class pictures. Every image is downsized such that its size fits the size reported in Epshtein and Ullman [EU07], i.e., about 200 pixels wide. Sample images from those datasets are shown in Figure 6.1.

[1] It indeed takes a long time to train and test on this kind of dataset, as several classifiers must be trained for each pair of class object and window aspect ratio (e.g. they are different for different viewpoints like rear view or side view). Also, the large number of training images and the parameter tuning do not help matters.

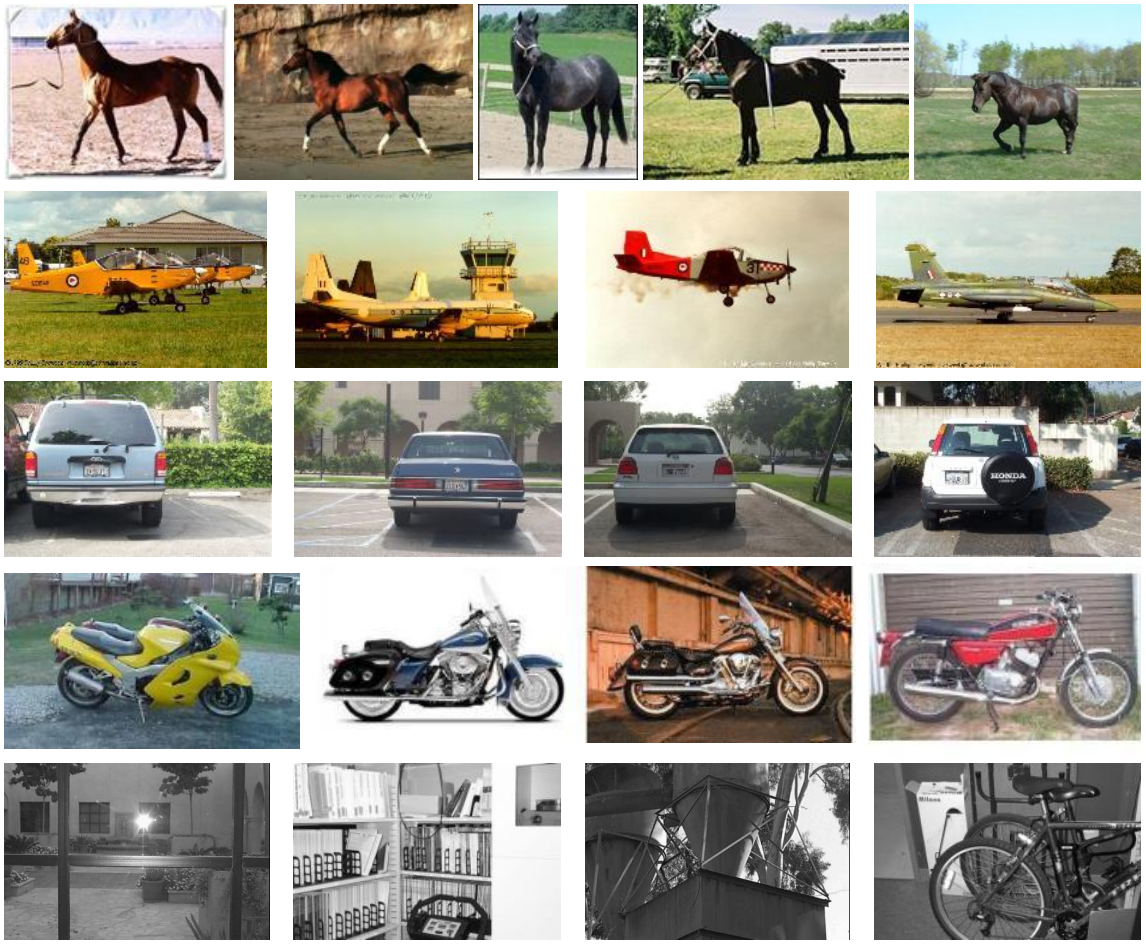


Figure 6.1: Sample images from the "horses", "airplanes", "car rears 127" and "motorbikes" datasets (each row shows instances from a single category in this order). The last row shows images from the "background" dataset used in our tests as non-class images.

Comments about the four datasets

In addition to the fact that running experiments on those four datasets is relatively fast because of the small sizes of images, hence allowing us to extensively study parameter effects, another advantage is that the datasets only contain instances viewed from a single viewpoint. With respect to our lattice training procedure, this is the perfect setting: training images are easily alignable hence enabling the construction procedure to learn all model parts from a large number of examples for each one (remember that a coverage map is used to divide each model image into a set of parts according to a pyramidal division the scale-space, see Section 3.5.5). Moreover, this evacuates the problem of scanning the image at different aspect-ratios as exists in more realistic datasets like the Pascal VOC.

Finally, one last advantage is that results from the method of Epshtein and Ullman [EU07] have been reported for two out of these four datasets (the third dataset used by Epshtein and Ullman is not publicly available to our knowledge[2]). It is interesting to compare to them as their method is also based on using local multi-scale model parts learned using mutual information, but the features and decision model are different from our system (respectively, they use template-based features and a hierarchical probabilistic model).

6.2.1 Parameter tuning

As a first experimental step, we evaluate in this section the influence of the following three parameters:

- The seed branch threshold $d_{K^z}^{max}$ (eq. (3.8), Section 4.2.1). Depending on the value of $d_{K^z}^{max}$, seed branches will let a various proportion of test keypoints traverse them. In particular, we suspect that a small value of $d_{K^z}^{max}$ would harm the performance, as too much similarity between train and test seed keypoints will be required in spite of the intra-class variations.

- The path termination threshold p^{max} (Section 5.3.3). When the probability $p(A_k|\neg\mathcal{O})$ of lattice aggregate A_k being detected in non-class images goes below p^{max}, then the corresponding lattice path is terminated. In other words, the higher is p^{max}, the shorter are lattice paths (on average).

- The minimum gain in mutual information gain_{min}^{+} (Section 5.3.3). When the gain of information from a candidate aggregate is inferior to gain_{min}^{+}, then it is pruned.

[2] We wrote to them to get the last dataset but we did not receive any answer.

A small value of gain^{+}_{min} thus involves less pruning and more terminal aggregates (i.e. a broader lattice).

To sum up, the first parameter controls the distinctiveness of the seed branches and the other two influence the lattice depth and breadth. (Note that $d^{max}_{K^z}$ also influence the lattice depth because $p(A_k|\neg\mathcal{O})$ depends on $d^{max}_{K^z}$ and the distinctiveness of the following branches on A_k's path.)

Experimental settings The following operations are done for each dataset independently.

Test/train data We randomly split in half the class and non-class images between training and test. The training set is one more time split: one half for learning the lattice and the classifier, and the other one for validation.

Lattice training The lattice construction is based on a subset of 32 class images for efficiency. For the same reason, we use random sampling during the candidate picking loop (step 4.a, Section 5.3.3) to speed-up the training. In other words, we only pick the best candidate from a sub-sample of all candidates in each iteration (an exhaustive search would be too slow). A drawback of randomization is that the training procedure can produce different lattices from the same training images and parameters.

Classifier training Concerning the classifier training, we use 3 levels of spatial pyramid (standard value in the literature [LSP06, GD05, HJS09]), $N_{doublets} = 10000$ doublet features randomly sampled from positive training windows and a codebook of 100 visual words learned from background images using k-means (superior values did not yield superior results due to the small image sizes, identical codebook size was reported in Harzallah et al. [HJS09] for Pascal 2007). We generate additional training windows by shifting the original bounding boxes in position and scale (two shifts of $\pm 12.5\%$ along x and y and one shift of $\pm 10\%$ along scale), hence multiplying the number of positive training windows by 7. Note that this operation is very important to improve performance because it forces AdaBoost to be robust to a noisy window positioning. This is in contrast with a traditional bag-of-features + χ^2-SVM [ZBMM06] where this operation is useless and even harms the performance according to our experiments. In fact, this problem does not arise for a SVM because the kernel returns real values directly related to the distances between samples; on the contrary for AdaBoost decision stumps only return binary

decisions. As a consequence, robustness to a noisy positioning must be explicitly learned from the training set. Finally, we use 4 iterations of bootstrapping to train real-AdaBoost.

Detection During detection, we scan each test image with a window of a fixed aspect-ratio at multiple scales (we use the average aspect-ratio of the training bounding boxes, the window shifts are 25% in position and 20% in scale). Each of the windows is evaluated independently by the classifier and a final non-maxima suppression step is performed on overlapping windows. Finally, ROC curves are generated from the detected boxes (we take 50% minimum mutual overlap with the ground truth to consider a box as correct as in [EVGW*]).

6.2.1.1 Influence of the parameters

We now perform an exhaustive search over the space of parameters. To reduce the search space, we quantize each parameter into the following values: $d_{K^z}^{max} \in \{7 \times 10^4, 8 \times 10^4, 9 \times 10^4\}$, $p^{max} \in \{5.1 \times 10^{-2}, 6.4 \times 10^{-3}, 8 \times 10^{-4}, 10^{-4}\}$ and $\text{gain}_{min}^{+} \in \{0.05, 0.07, 0.09, 0.11, 0.13\}$. The values for $d_{K^z}^{max}$ correspond respectively to a matching probability of 2.3%, 3.9% and 6.2% between two keypoints randomly sampled in natural images. The values for p^{max} correspond to average path lengths of 2.01, 2.49, 3.23 and 3.51 respectively. Finally, the values for gain_{min}^{+} respectively correspond to an average number of terminal aggregates per lattice of 1264, 1015, 833, 702 and 594.

Mean AUC

First of all, we investigate the parameter influence on the recognition performance in term of the Area Under ROC Curve (AUC) measure. To that aim, we measure the average AUC when one parameter is fixed and the two other ones are varying. Results are shown for each parameter separately in Figures 6.2, 6.3 and 6.4[3].

As can be noticed on the averaged charts, performance is substantially equivalent whatever the value of each parameter. About the parameter $d_{K^z}^{max}$, slightly better results seem to be reached for $d_{K^z}^{max} = 80000$. About the minimum information gain gain_{min}^{+}, slightly better results are achieved for low values. This appears logical as low values imply larger detection lattices and hence larger feature pools, yielding more choice to AdaBoost to build the classifier. Still, there is not a big AUC difference between

[3]Due to a different threshold involved in the non-maxima suppression step between experiments of this section and the next section, the AUC values obtained in the charts do not exactly corresponds to subsequent AUC curves below (which are correct). Please excuse us for this awkwardness which fortunately does not invalidate the results in the charts (only a constant offset is added).

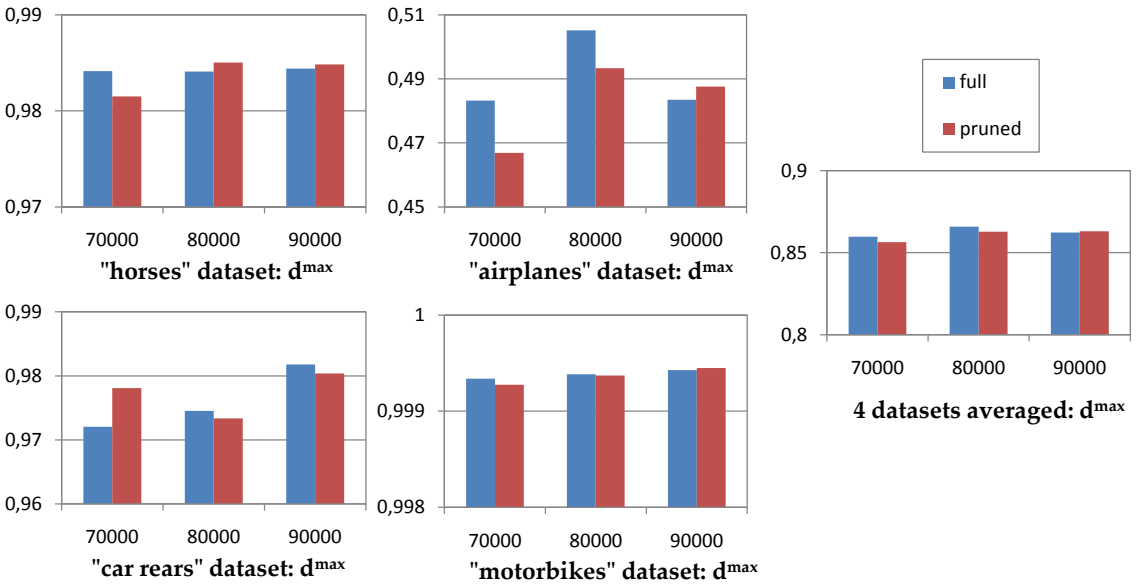


Figure 6.2: Influence of the parameter $d_{K^z}^{max}$ on the performance in term of AUC measure on the four datasets. Left bars correspond to using f_0^C, f_1^C and f_2^C and right bars to not using them in order to prune the detection lattice. The last chart shows averaged AUC over all datasets.

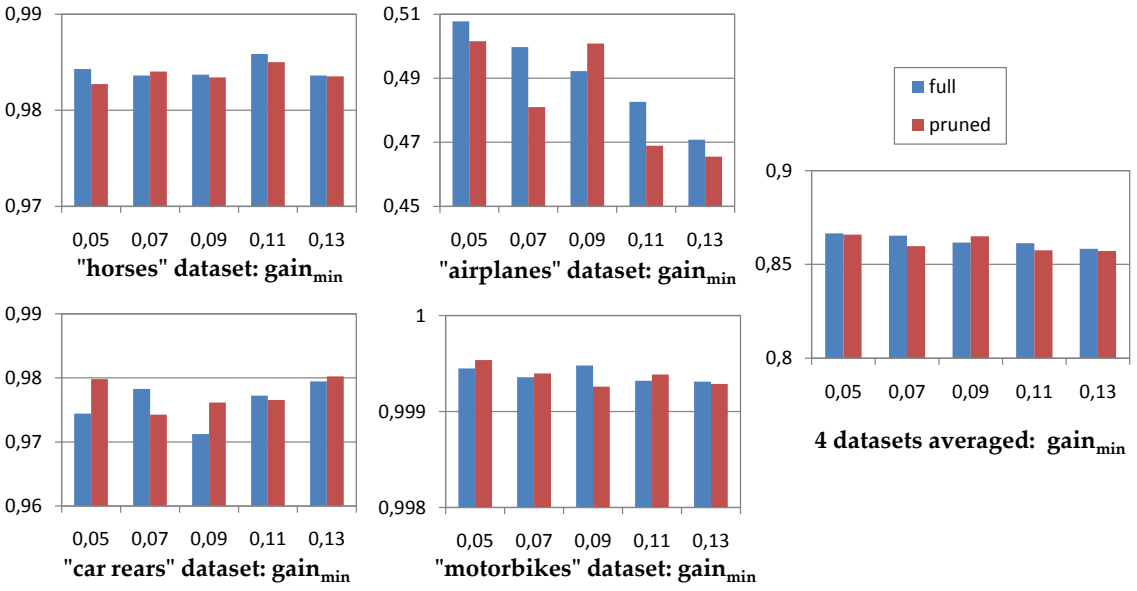


Figure 6.3: Influence of the parameter gain_{min}^{+} on the performance in term of AUC measure on the four datasets. Left bars correspond to using f_0^C, f_1^C and f_2^C and right bars to not using them in order to prune the detection lattice. The last chart shows averaged AUC over all datasets.

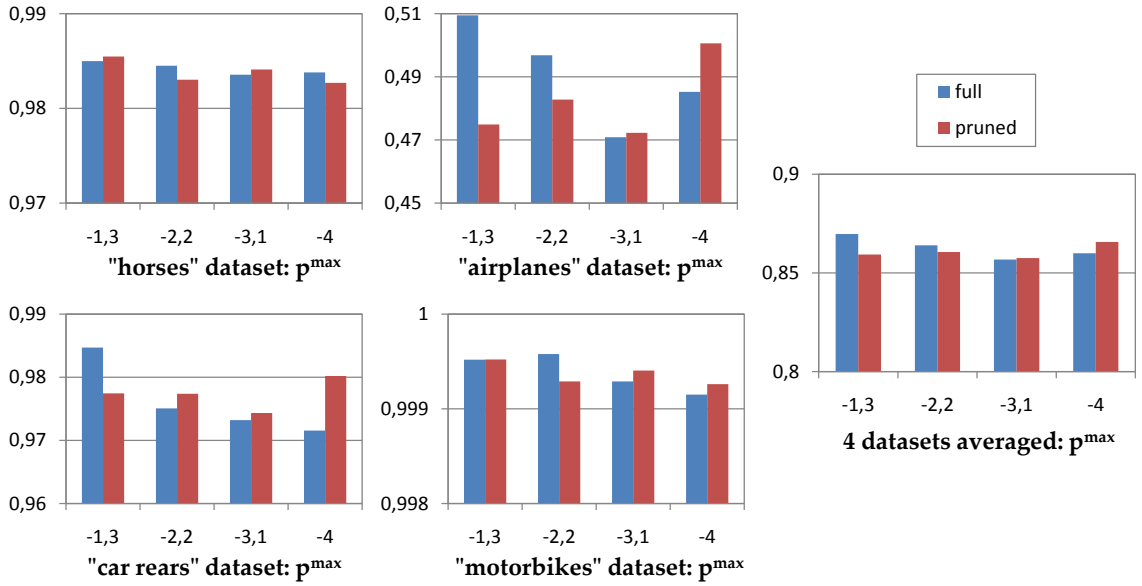


Figure 6.4: Influence of the parameter p^{max} on the performance in term of AUC measure on the four datasets (labels are expressed as decimal logarithms of p^{max}). Left bars correspond to using f_0^C, f_1^C and f_2^C and right bars to not using them in order to prune the detection lattice. The last chart shows averaged AUC over all datasets.

$\text{gain}^+_{min} = 0.05$ and $\text{gain}^+_{min} = 0.13$, meaning that our approach succeeds in picking first the most discriminative candidates during the lattice construction loop. About the parameter p^{max}, for some reason the best results depend whether the features f_0^C, f_1^C and f_2^C are used or not (although the change is not so important). Namely, when the $\{f_n^C\}_{n=0..2}$ are used (i.e. the detection lattice is not pruned) then setting $p^{max} = 10^{-1.3}$ performs best. On the contrary, when the $\{f_n^C\}_{n=0..2}$ are not used, setting $p^{max} = 10^{-4}$ performs best. It is interesting to note however that the $\{f_n^C\}_{n=0..2}$ are not so useful: essentially similar performance is reached whether they are used or not. This is important since it means that the detection lattice can be pruned so as to reduce the detection time (Section 5.2.4) without sacrifying the performance.

Detection Time

We investigate now the effect of the different parameters on the detection time. It is indeed expected that a larger detection lattice implies more detected aggregates, hence more time spent per image. We display in Figure 6.5 the detection time averaged over all datasets as a function of the three different parameters (i.e. excluding the feature extraction step: we only measure the time of aggregate extraction followed by sliding window classification). Interestingly, one can see that pruning the detection lattice after constructing the classifier (Section 5.2.4) significantly boosts the detection speed by a

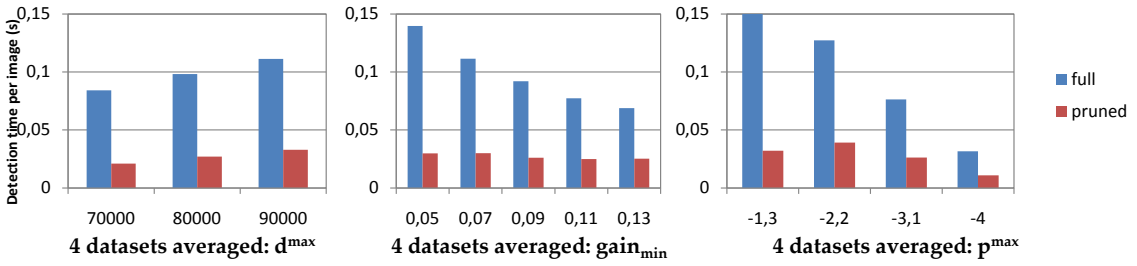


Figure 6.5: Average detection time per image (excluding the feature extraction step) over the different datasets in function of the parameters. The detection time is almost constant and independent of the parameters when the detection lattice is pruned.

factor 4~5. In particular, it is crucial to point out that after pruning the detection time does not depend on the original lattice width anymore (see middle and left charts in Figure 6.5). As a consequence, we systematically discard the $\{f_n^C\}_{n=0..2}$ features in order to prune the lattice in the following experiments.

6.2.2 Comparison against other approaches

We now quantitatively evaluate our approach on the four datasets versus the following settings:

- A baseline bag-of-features with $N_{words} = 100$ (superior values did not yield superior results). We use a SVM with a χ^2 kernel as suggested by [BZ07] and determine the slack parameter on the validation set.
- A baseline bag-of-features with $N_{words} = 100$, but using real-AdaBoost instead of a SVM.
- The method of Epshtein et al. [EU07] which is related to ours in the fact that discriminative model parts are also learned during training using mutual information gain. However contrary to us, their features are dense image patches matched by normalized cross-correlation (NCC), and they use a hierarchical probabilistic model to infer the object presence from the detected parts (Figure 6.10). (Note that their results are only reported for two out of our four datasets.)

Concerning our approach, we try different values of d_{Kz}^{max}, gain_{min}^{+} and p^{max} and retain the combination that produce the best performance on the validation dataset. This is crucial in order to cancel the effect of randomization in the lattice construction procedure (a probably more efficient solution would be to merge several lattices constructed with different or identical parameters, and to use this larger lattice for training). Moreover, we use two settings for our method in order to investigate the complementarity

of visual words and aggregate features: in the first setting we use both visual words and aggregates, while in the second setting we use only aggregates. Finally, we evaluate the performance using either one level of spatial pyramid [LSP06, BZ07] (i.e. pure bag-of-features without any spatial information) or 3 levels (21 cells per window, standard setting in the literature).

Experimental results

Sample detections for the "horses" and "car-rears" datasets are illustrated in Figures 6.6 and 6.7 respectively. Each image corresponds to the window which was ranked highest by the classifier (i.e. images are cropped). Aggregates used by the classifier are drawn as green squares. As can be observed, most aggregates are positioned on object parts rather than on the background, demonstrating that our lattice construction procedure is well able to learn discriminative aggregates.

Then, quantitative results for one level are shown in Figure 6.8 and results for three levels are shown in Figure 6.9. The acronyms in the figure legends are the following: "BoF" means bag-of-features (i.e. classical visual words), "Aggr" means that we use aggregates detected with the lattice, and "SVM" or "realAB" (real-AdaBoost) specify the type of the classifier.

In the first situation (one level) our methods constantly outperform all others. The performance for the two baselines (BoF+SVM and BoF+realAB) is roughly similar, while the performance using only aggregates is slightly inferior to the combination of visual words and aggregates.

On the contrary for three pyramid levels we observe a good complementarity between visual words and aggregates (in particular for the "horses", "motorbikes" and "car rears" datasets). This is interesting as it means that aggregates alone are too much specific to the learned class and are insufficient to achieve high performance. This is because they do not allow a good modeling of the negative windows; on the contrary visual word features are more generic as the visual codebook is constructed from a collection of natural images: the combination of the two feature groups substantially improves the performance, demonstrating their good complementarity. Apart from this, our methods still outperform the baselines for three pyramid levels except for the "motorbikes" dataset where the SVM performs slightly better. This is due to the fact that the localization is less precise using AdaBoost that using a SVM for the above mentioned reason (i.e. real-valued kernel distance vs binary stumps). Indeed, we will see in the next paragraph that when we classify the images our system performs better than the

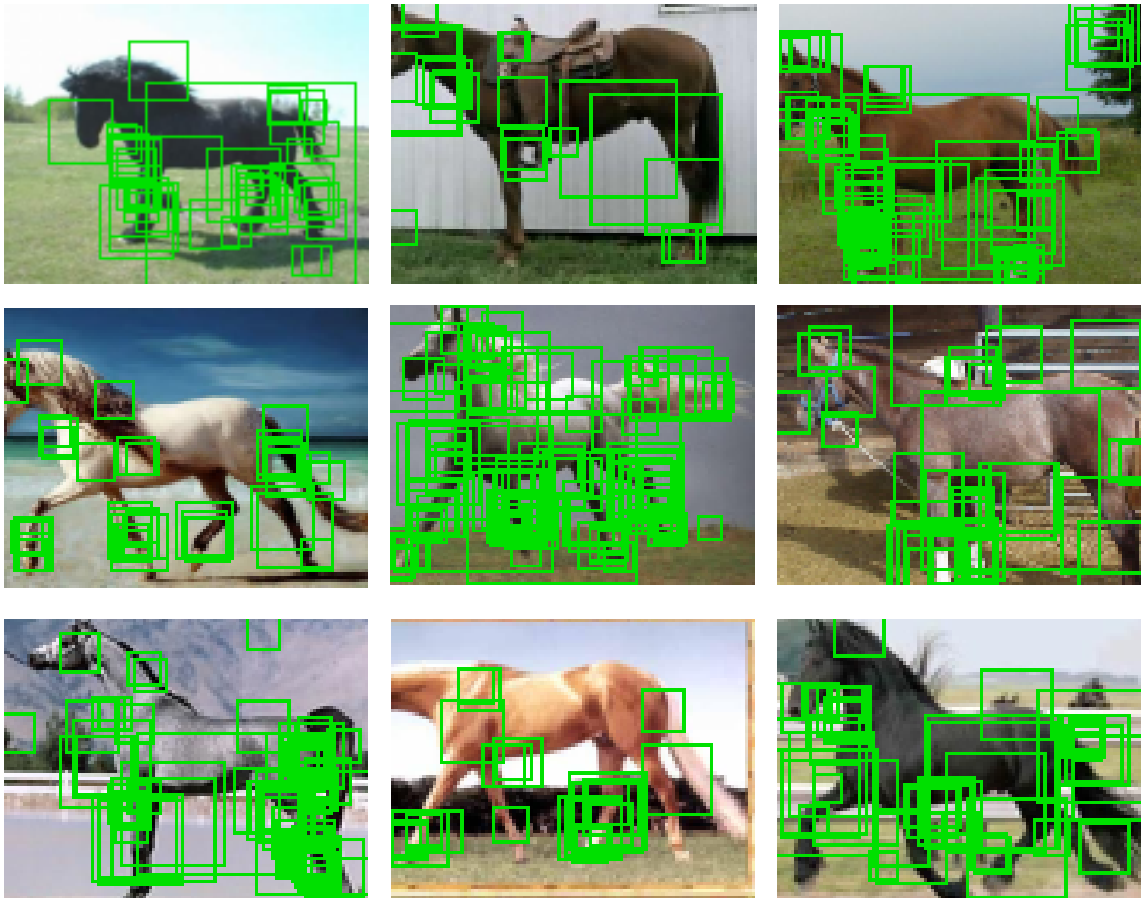


Figure 6.6: Sample detections on the "horses" dataset (only the highest-ranked window per image is shown). Aggregates used by the classifier to rank the window are highlighted in green squares. As can be observed, most squares are positioned on object parts rather than on the background, demonstrating that our lattice construction procedure is well able to learn discriminative aggregates.

Figure 6.7: Sample detections on the "car-rears" dataset (only the highest-ranked window per image is shown). Aggregates used by the classifier to rank the window are highlighted in green squares. As can be observed, most squares are positioned on object parts rather than on the background, demonstrating that our lattice construction procedure is well able to learn discriminative aggregates.

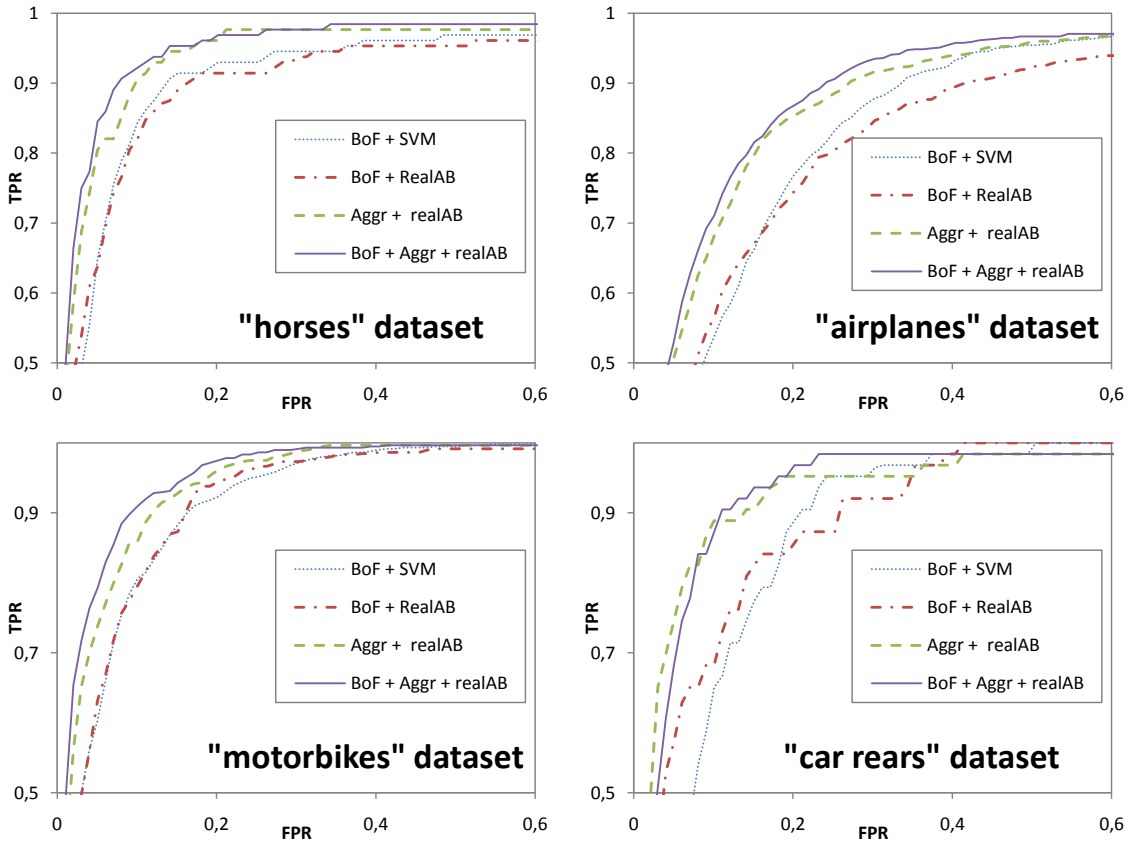


Figure 6.8: ROC plots on the four datasets for our methods versus several baselines. For all methods we use one level of spatial pyramid, i.e. this is a pure bag-of-features setting. The acronyms in the legends are the following: BoF means bag-of-features (i.e. classical visual words), Aggr means that we use aggregates detected with the lattice, and SVM / realAB (real-AdaBoost) specifies the used classifier type.

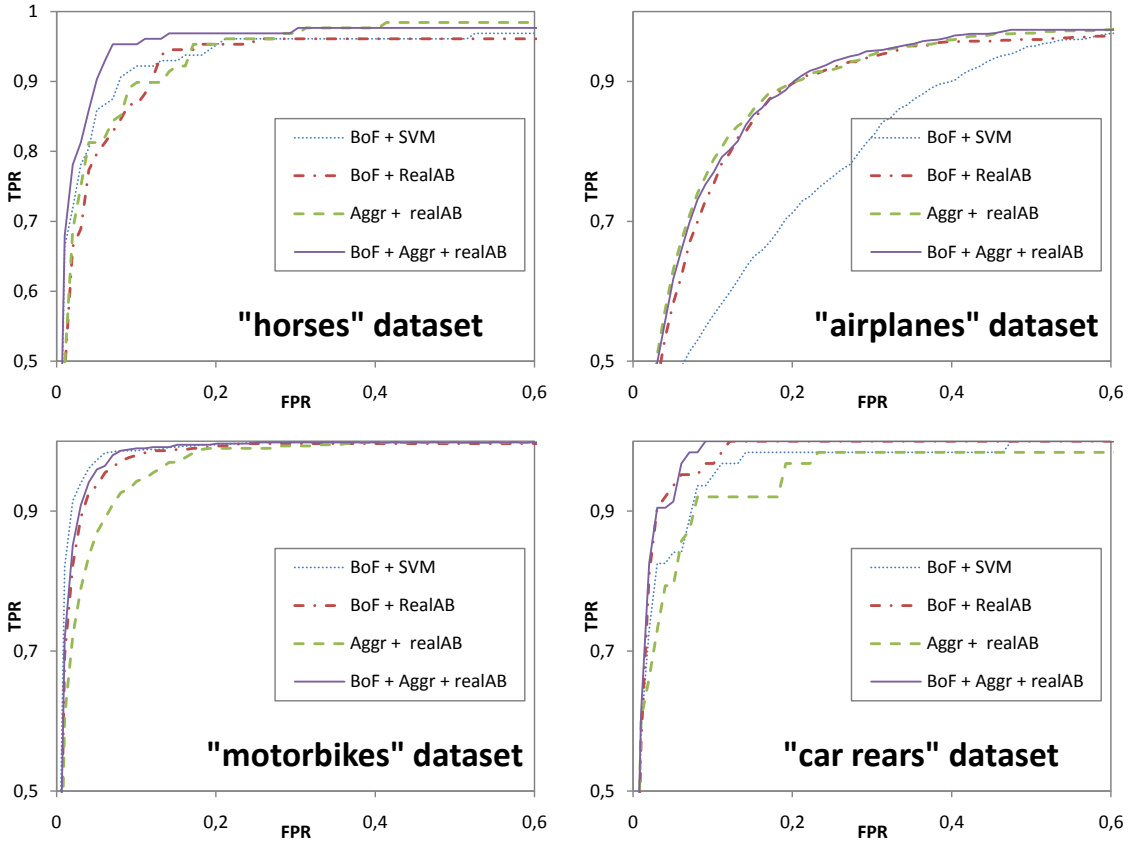


Figure 6.9: ROC plots on the four datasets for our methods versus several baselines. For all methods we use three levels of spatial pyramid (a standard setting). The acronyms in the legends are the following: BoF means bag-of-features (i.e. classical visual words), Aggr means that we use aggregates detected with the lattice, and SVM / realAB (real-AdaBoost) specifies the used classifier type.

same baseline SVM for this dataset although exactly the same lattice and classifier are used.

Finally, note that the baseline SVM performance significantly degrades for three pyramid levels compared with one level on the "airplanes" dataset. As noted by Bosch and Zisserman [BZ07], the importance of each pyramid level varies from one class to another, and more complex models are required to learn those distributions. In contrast to SVMs, AdaBoost coupled with decision stumps selects a sparse set of features to construct the classifier and hence is able to avoid inefficient features, independently of their level in the pyramid. This is in contrast with the sample-based selection made by a SVM where single features can not be discarded. Because our approach also uses AdaBoost, we can theoretically set the number of levels to a high value without fearing performance losses (although the training time may dramatically increase). This is important as the less parameters to tune, the more simple and convenient is a method.

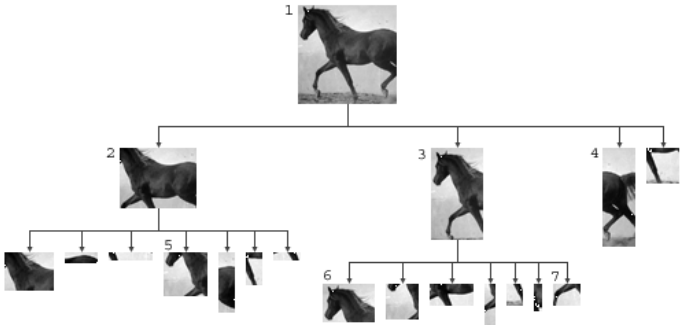


Figure 6.10: The hierarchical decomposition of Epshtein and Ullman [EU07]: an example of simple hierarchy.

Classifying images

In order to compare to Epshtein and Ullman [EU07] we conform to their experimental settings: for each image, we perform object recognition using a sliding window, then we retain the label of the box with the highest score to determine the image label (i.e. class or non-class image). Unfortunately, only two of the three datasets that Epshtein and Ullman have used are available to us (motorbikes and horses). Results in term of ROC plots for those two datasets are shown in Figure 6.11. Again, our approach outperforms theirs. It is important to note that the method of Epshtein and Ullman is probably much slower than ours, although they did not report any running time: in fact, they have used dense patch features which are searched in the test image using Normalized Cross Correlation (NCC). This operation is typically very slow, and probably for this reason they have only evaluated their system on a small number of classes with small sized images. On the contrary, our method classifies each image in a matter of milliseconds, if we exclude the feature extraction step (keypoint extraction is the longest part, it requires about half a second for an image, the segment and texture extraction only last about 100 milliseconds[4]).

6.3 Pascal 2005 Classification experiments

We present in this section an evaluation of our approach in term of classification performance on the Pascal VOC 2005 dataset [EGWZ05]. Our purpose is to compare to related approaches like the methods of Nowozin et al. [NTU*07] and Zhang et al. [ZYW*10] as well as to the best results from the state-of-the-art.

[4]Note however than the feature extraction step has to be done once per image, in the case where multiple objects are searched in an image

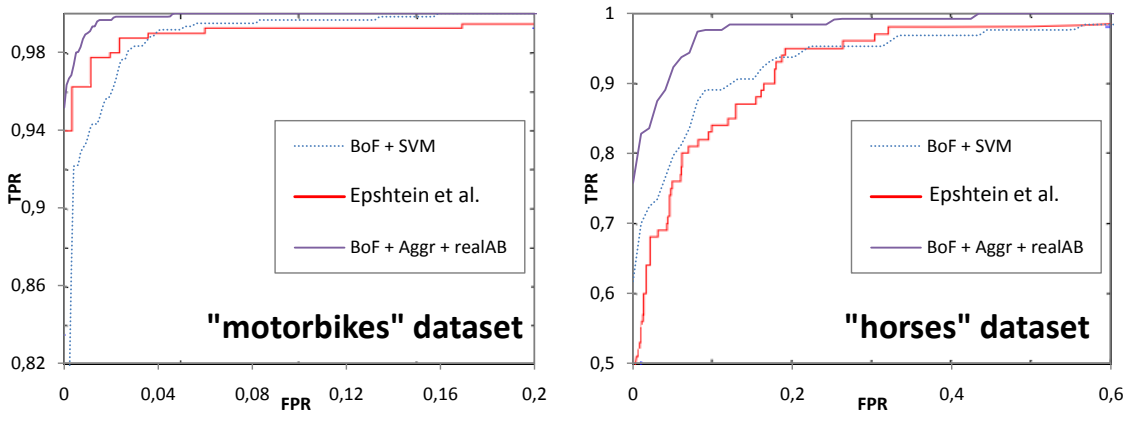


Figure 6.11: Comparison of our method with the method of Epshtein and Ullman [EU07] and a baseline bag-of-features + SVM.

Dataset description The Pascal VOC 2005 dataset aims at evaluating the performance of classification and object recognition approaches in unconstrained natural images. The dataset comprises four categories: bicycles, cars, motorbikes and people. For each category, a set of training and test images are provided (a various number of instances from only one category are present in each image). Sample training images are shown in Figure 6.12. As can be observed, instances can appear at various scales, locations, viewpoints and levels of occlusion, making the dataset extremely challenging. Note however that the category "motorbikes" is easier than the other ones as instances most often appear under the same viewpoint, fit the image size and are in front of an empty background (e.g. first and second pictures in the third row of Figure 6.12).

Finally, it is important to note that two different test sets are provided by the dataset creators: the first set follows the same distribution than the training set (i.e. images originate from the same sources), while the second set is composed of images gathered from different sources. As a result, the second test set is harder but enables to evaluate the generalization power of an approach.

Experimental settings

Because object recognition and localization is unrealistic on the Pascal VOC 2005 dataset with our current approach (particularly for the people category where the variability is huge, see last row of Figure 6.12), we turned toward the image classification challenge. Contrary to the protocol presented in the previous section where the best window was used to determine the image score, we simply make a single evaluation per image to assign its score, that is, we only evaluate a single window fitting the whole image. This way of doing is standard in the literature and shrugs off aspect ratio and viewpoint

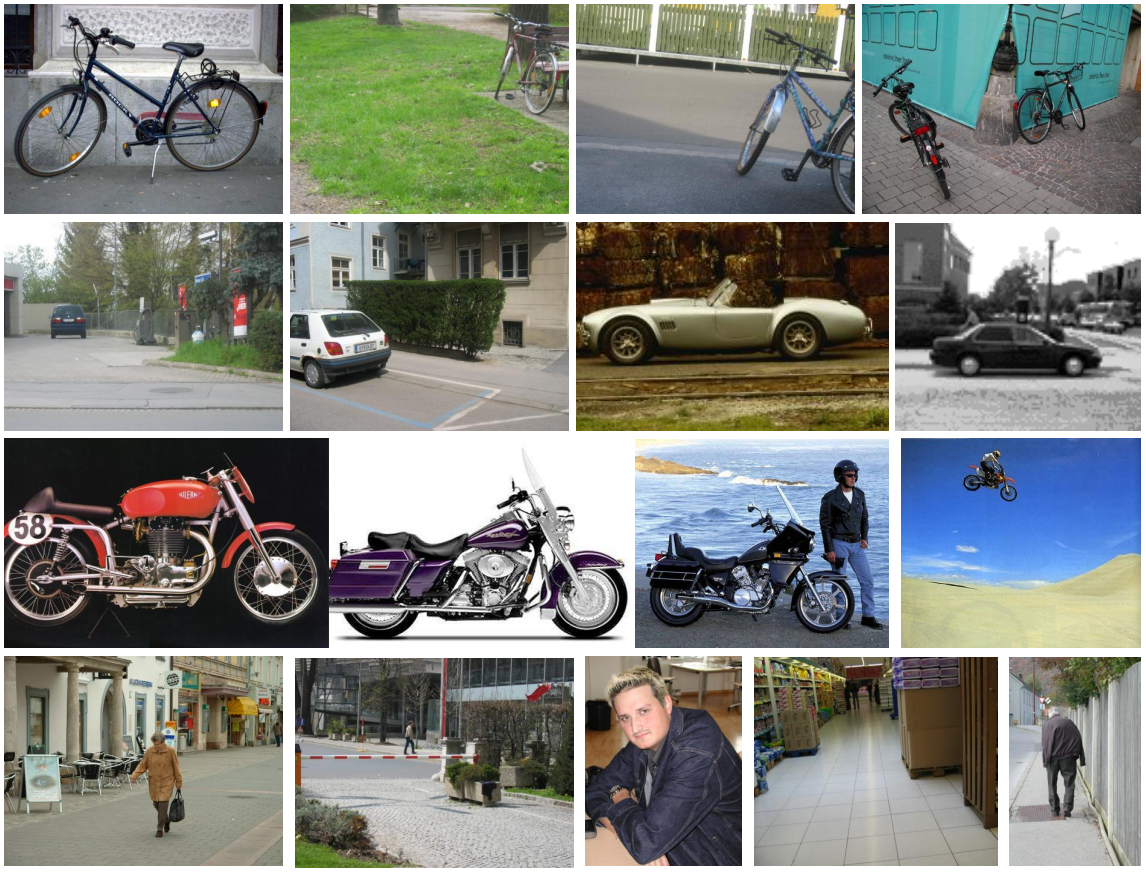


Figure 6.12: Sample images from Pascal VOC 2005 [EGWZ05] (each row shows images from a single category). The intra-class variations are much greater for the "people" category than for other classes.

issues raised by sliding window methods when object classes have great variability. Moreover, it has the advantage to take into account the image *context*, i.e. to take into account any information in the image other than the object itself that may be useful for classifying it.

Concerning our training procedure, we build beforehand one lattice per category using 32 positives images and every available negative images from the other categories. We set the thresholds to the following values: $p^{max} = 10^{-4}$, $\text{gain}^{+}_{min} = 0.05$ and $d^{max}_{K^z} = \{60000, 70000, 80000, 90000\}$. We set gain^{+}_{min} to a low value to generate a large number of aggregates, since it tends to increase the performance as we saw previously (Figure 6.3). We also set p^{max} to a low value for the same reason (see Figure 6.4) and because we expect more keypoints per window (since images are larger), hence requiring more distinctive aggregates. Finally, we select several values for the seed branch threshold $d^{max}_{K^z}$ since the tuning of Section 6.2.1 did not clearly demonstrate any preference[5]. We thus obtain a total of 16 lattices (4 per category) which we merge altogether using a simple concatenation. The resulting lattice is used to collect aggregates in the training and test images. Then, we train a binary classifier for each category with respect to the provided validation set. Although we could use several pyramid levels to improve the performance, we only use one pyramid level (i.e. pure bag-of-aggregates) for a fair comparison with other existing approaches (see below). For the same reason, we do not use doublet features (i.e. $N_{doublets} = 0$). We also experiment our method with and without using visual words (we use a codebook of $N_{words} = 250$). Finally, after learning the classifier we prune the original training lattice so as to save only the selected aggregates. Whereas the original lattice contains 40,638 aggregates (6684 seed branches, 5.2 features per terminal aggregate on average), the pruned lattices only contain 785 (249 seed branches), 935 (286 seed branches), 1013 (305 seed branches) and 349 aggregates (102 seed branches) respectively for the bicycle, cars, motorbikes and people categories.

Experimental results

Following the standard procedure, results are expressed in terms of Equal Error Rate (EER) measured on separate ROC curves (one for each category). Table 6.1 shows the performance of our method as well as results of existing methods from the state-of-the-art. Our method is clearly inferior to the best results of Larlus et al. [LDJ06] and Zhang et al. [ZBMM06], but on the other hand our aggregates only grow from salient keypoints (i.e. sparse SIFT) whereas million of dense vectors are extracted in Larlus

[5] in fact, the optimal value may depend on each seed features

	test set 1				test set 2			
method	Motorbikes	Bicycles	People	Cars	Motorbikes	Bicycles	People	Cars
bag-of-features (sparse) [ZBMM06]	95.8	89.4	90.4	92.3	79.7	68.4	72.8	72.3
bag-of-features (dense) [LDJ06]	97.7	93.0	97.7	96.1	-	-	-	-
subgraph boosting [NTU*07]	80.6	69.0	66.8	80.7	66.2	62.1	62.1	63.4
Subgraph sharing [ZYW*10]	81.5	71.2	71.1	83.8	66.3	62.2	62.5	68.7
Our method (Aggregates only)	95.0	82.6	85.5	91.6	70.7	70.0	72.0	72.5
Our method (Bof+Aggr)	95.6	85.1	85.3	92.0	72.4	72.0	72.5	74.8

Table 6.1: Equal Error Rate (EER) for each category on Pascal VOC 2005.

et al. algorithm, making it probably very slow. Concerning the work of Zhang et al. [ZBMM06] we report in Table 6.1 their results obtained using only keypoint extracted with the SIFT detector (like we do) for a fair comparison. Note however that they use two types of descriptors for each keypoint (SIFT and SPIN). Again, our results are about 5% inferior on the first test set. However, on the second test set we obtain similar and even superior results on the "cars" and "bicycles" categories, meaning that our method is well able to generalize.

On the contrary, our method largely outperforms the related approaches of Nowozin et al. [NTU*07] and Zhang et al. [ZYW*10] for all categories and test sets. The improvement in term of EER ranges between 5% and 14% for both test sets. This is not a surprise insofar as both Nowozin et al. [NTU*07] and Zhang et al. [ZYW*10] extract small 80-node graphs per image for computational reasons. In comparison, our method can afford to look at thousands of subgraphs per image without noticeable slowdown. Indeed our classification time per image is 3.28s on average, from which 2.78s s are spent for extracting features (80% of this time is just dedicated to keypoint extraction and description) and the rest (0.5s) is spent on detecting aggregates and evaluating all four classifiers (note that the classification time is the same whether we use the visual words or not). This latter time is well below the time of ~1s per image and per category reported by Zhang et al [ZBMM06] to classify an image using either a EMD or a χ^2 SVM kernel.

Discussion about normalization issues

More generally, we explain our inferior results with respect to bag-of-features methods on the first test set by the fact that our decision scheme only considers the binary presence or absence of aggregates, which may contain less information for representing the image than histograms do. In fact, there is no explicit normalization in our method with respect to the image size except for the fact that all aggregates incorrectly extrapolating

the image size are discarded (i.e. this is the same strategy than in Section 6.2 with the sliding window). Yet, this scheme is not robust against scale changes of instances in test images (a common situation in the Pascal VOC 2005 dataset). On the contrary, in the classical bag-of-features robust normalization is naturally achieved by computing the relative proportion of each visual word in the image (i.e. histogram bins sum to one). To conclude with, it would be interesting to investigate other ways of normalizing our feature vector, e.g. to use an histogram of subgraphs like Ozdemir and Aksoy [OA10] for which spatial information is not entirely lost in the bining process (local spatial information is encoded into the subgraphs).

6.4 Conclusion

We have presented a quantitative evaluation of the contribution of Chapter 5. We have first evaluated our approach on the tasks of recognizing and localizing instances belonging to 4 separate object classes. Our results are superior to different baselines as well as to the hierarchy of parts of Epshtein et al. [EU07]. Besides, pruning the lattice appears to be a winning strategy as it implies similar performance while the detection time is divided by 5 in the same time.

Secondly, we have compared our approach to other works on the Pascal VOC 2005 dataset in the classification challenge. We have obtained results that are slightly inferior or equivalent to the state-of-the-art with a faster classification scheme; moreover our method largely outperforms related approaches based on subgraph boosting. As noted by Zhang et al. [ZYW*10], our moderately good results against bag-of-features models may be explained by the fact that our lattice construction process is strongly suboptimal due to the best-forward aggregate selection procedure (training loop of Section 5.3.3). As a result, perspectives include discovering better heuristics for aggregate selection as well as finding more robust normalization schemes in the classification case.

Chapter 7

Conclusion

7.1 Summary of Contributions

In this dissertation, we have presented two contributions to the field of object recognition. The first contribution leans upon to the case of specific objects while the second one applies to class objects. We now summarize each of our contributions to their respective framework.

Specific Object Recognition

In Chapter 3, we have presented a method of specific 3D object recognition inspired from graph matching. Specific object recognition is a problem that may appear simpler to handle than class object recognition, but as our experiments have shown, there is still a long way before reaching a 100% detection rate in realistic (noisy) conditions of use. The insight of our approach is to decompose the model object into a set of parts and to detect those parts in the scene image to infer the object presence (note that we do not explicitely model the object 3D shape, instead we tackle viewpoint changes by allowing some tolerance to distortions). Contrary to keypoint-based approaches [Low04, LLF05, MP08], only a small number of model parts are selected during training according to a mutual information based criterion. Another difference with respect to classical graph matching, where the scene images are beforehand converted into graphs, is that our system is able to generate new graph nodes at detection time in a dense fashion. The only constraint is that the search of plausible parts in the scene image has to begin from salient spots (keypoints). Contrary to similar methods relying on slow expansion/contraction iterations [FTG06, KP06], our matching scheme relies on cascades to speed up the detection. The use of cascades enables in addition to combine different feature channels (both

sparse and dense) which results in a significant improvement in term of robustness, as demonstrated in Chapter 4.

Indeed, our experiments on our own dataset fitted to model the realistic conditions of mobile robotic (11 model objects and more than 2800 test images) have shown that our detection scheme outperforms other standard methods. Other comparisons against state-of-the-art algorithms on existing datasets emphasizing occlusion and clutter rather than scale changes show that our method performs slightly less well than other approaches; however our method remains generally faster. All in all, we demonstrated that despite a small number of training images cascades are adequate for specific object recognition, and that it is extremely important to use several feature types (especially dense features) in the recognition process.

Class Object Recognition

In chapter 5, we have introduced an extension of the graph matching method described in Chapter 3 to the detection of class objects. The basic reason for this extension was the fact that class objects can also be efficiently represented as collection of parts [AAR04, BT05], hence allowing to reuse our previous part detection system. However, we have used a discriminative classifier (real-AdaBoost) instead of the probabilistic model originally proposed in Chapter 3 in order to better handle class variations. Globally, the strength of our approach comes from that it combines both statistical discriminative learning through sliding window and bag-of-features frameworks, and the representational power of structural methods through the graph matching framework. To sum up our method, we achieve object recognition using a sliding window in which the model presence is evaluated from visual words as well as detected model parts. Moreover, contrary to classical approaches where the parts are learned before training the classifier (e.g. [VNU03]), in our method different levels of distinctiveness for each model part are dynamically available to the classifier at training time. As a result our detection process relies both on generic features suited to detect any classes (including the background) and on class specific features (model parts) with finely tuned distinctiveness dedicated to the model class. Although dedicated features are generally slower to compute (e.g. [SWB*07]), in our case the cascaded lattice enables a relatively fast detection. In addition, we have also introduced two optimizations to fasten both the training and the detection steps.

We have then experimented the resulting system in Chapter 6 on several existing datasets. On the first dataset composed of four single classes we have evaluated our ap-

proach in terms of object recognition and image classification. On this occasion we have demonstrated superior performance against a baseline bag-of-words and a part-based model from the state-of-the-art. Secondly, we have answered the image classification challenge proposed for the Pascal VOC 2005 dataset. We have shown that our method is able to achieve results slightly inferior, similar or even superior to other state-of-the-art approaches for a smaller detection time. Moreover, the comparison with related approaches of subgraph boosting has clearly turned in our favour with improvements of 10% in term of EER measure, hence validating the soundness of our approach.

7.2 Perspectives

Given the extent of the works presented in this dissertation, there are many possible perspectives. We give here a non-exhaustive list of the most important tracks in our opinion.

Scalability issues

The detection technique that we have used for our first contribution is not yet scalable to a large number of model objects in the database. The reason is that we use an absolute distance between keypoint descriptors instead of a relative distance using a kd-tree like other methods generally do (e.g. [Low04]). Yet, we believe it possible to tackle this issue in two complementary ways: first is to gather all detection lattices (one per model object) into one single lattice in order to share aggregates accross model objects. A similar approach for subgraph sharing applied to pure graph matching was already demonstrated to be feasible by Messmer and Bunke [MB98]. Secondly we noticed that the detection time is linear with the number of branches starting from the lattice root node, because each keypoint in the test image is compared to each first-level seed feature. So by interposing a tree-shaped indexing structure between the root and the first-level aggregates, one should be able to achieve a significant speed-up. The work of Beretti et al. [BDV01] for indexing graph using nested spheres, for instance, seems to be a promising track to achieve such a boost.

Real-time object detection

Although both contributions are already fast, there is still a gap before achieving real-time detection. One general solution to increase the speed would be to parallelize the feature extraction process. A promising track seems also the use of faster keypoint

extraction and description schemes (e.g. the ferns proposed by Özuysal et al. [zCLF09]). Finally, a simple way to further accelerate the detection speed for the second contribution would be to use soft cascades [BB05].

Improvement of the lattice construction procedure

As noted by Zhang et al. [ZYW*10], the fact that bag-of-subgraphs perform less well than classical bag-of-features for the classification task may be explained by the fact that the subgraph generation process is strongly suboptimal due to the best-forward aggregate selection procedure (training loop of Section 5.3.3). Indeed, it seems impossible to exhaustively explore the huge space of all possible subgraphs. As a result, additional researches are necessary to discover better heuristics for subgraph selection during the lattice construction process.

Improvement of the bag-of-subgraphs model

Finally, we believe that the binary representation of the bag-of-subgraphs model (i.e. only considering the presence or absence of each aggregate) is suboptimal for image classification. In fact, there is the lack of a normalization compared to classical bag-of-features where robust normalization is naturally achieved by computing the relative proportion of each visual word in the image (i.e. histogram bins sum to one). To conclude with, it would be interesting to investigate other ways of normalizing our feature vector, e.g. to use an histogram of subgraphs like Ozdemir and Aksoy [OA10].

Bibliography

[AAR04] Agarwal S., Awan A., Roth D.: Learning to detect objects in images via a sparse, part-based representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 26* (2004), 2004.

[AMFM09] Arbelaez P., Maire M., Fowlkes C., Malik J.: From contours to regions: An empirical evaluation. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[BB05] Bourdev L., Brandt J.: Robust object detection via soft cascade. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2005).

[BBM05] Berg A. C., Berg T. L., Malik J.: Shape matching and object recognition using low distortion correspondences. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 26–33.

[BBU04] Bart E., Byvatov E., Ullman S.: View-invariant recognition using corresponding object fragments. In *European Conference on Computer Vision (ECCV)* (2004), vol. 0, pp. 152–165.

[BDBV01] Berretti S., Del Bimbo A., Vicario E.: Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 23*, 10 (2001), 1089–1105.

[BDV01] Beretti S., Del Bimbo A., Vicario E.: Efficient matching and indexing of graph models in content-based retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 23*, 10 (2001), 1089–1105.

[BGV92] Boser B. E., Guyon I. M., Vapnik V. N.: A training algorithm for optimal margin classifiers. In *Conference on Learning Theory (COLT)* (1992), pp. 144–152.

[BM00] Belongie S., Malik J.: Matching with shape contexts. In *Content Based Access of Image and Video Libraries (CBAIVL)* (2000), p. 20.

[BT05] Bouchard G., Triggs B.: Hierarchical part-based visual object categorization. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 710–715.

[BTG06] Bay H., Tuytelaars T., Gool L. V.: Surf: Speeded up robust features. *Lecture Notes in Computer Science Volume 3951* (2006), 404–417.

[BU02] Borenstein E., Ullman S.: Class-specific, top-down segmentation. In *European Conference on Computer Vision (ECCV)* (2002), pp. 109–124.

[BZ07] Bosch A., Zisserman A.: Representing shape with a spatial pyramid kernel. In *Conference for Image and Video Retrieval (CIVR)* (2007), pp. 401–408.

[Can86] Canny J.: A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 8*, 6 (1986), 679–698.

[CDF*04] Csurka G., Dance C. R., Fan L., Willamowski J., Bray C.: Visual categorization with bags of keypoints. In *Workshop on Statistical Learning in Computer Vision* (2004).

[CFH06] Crandall D., Felzenszwalb P., Huttenlocher D.: *Toward Category-Level Object Recognition*. 2006, ch. Object Recognition by Combining Appearance and Geometry, pp. 462–482.

[CFJV06] Conte D., Foggia P., Jolion J., Vento M.: A graph-based, multi-resolution algorithm for tracking objects in presence of occlusions. *Pattern Recognition (PR) 39*, 4 (Apr. 2006), 562–572.

[CFSV04] Conte D., Foggia P., Sansone C., Vento M.: Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence 18* (2004), 265–298.

[CKP94] Christmas W. J., Kittler J., Petrou M.: Matching of road segments using probabilistic relaxation: Reducing the computational requirements. In *Sensing, Imaging and Vision for control and guidance of aerospace vehicles* (1994), pp. 169–179.

[CLF08] Calonder M., Lepetit V., Fua P.: Keypoint signatures for fast learning and recognition. In *European Conference on Computer Vision (ECCV)* (2008), pp. 58–71.

[CLK*09] Calonder M., Lepetit V., Konolige K., Mihelich P., Bowman J., Fua P.: Compact signatures for high-speed interest point description and matching. In *International Conference on Computer Vision (ICCV)* (2009).

[CM02] Chetverikov D., Matas J.: Periodic textures as distinguished regions for wide-baseline stereo correspondence. In *Textureo2* (2002), pp. 25–30.

[CM05] Chum O., Matas J.: Matching with prosac " progressive sample consensus. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 220–226.

[CM08] Chum O., Matas J.: Optimal randomized ransac. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 30*, 8 (2008), 1472–1482.

[CMK03] Chum O., Matas J., Kittler J.: Locally optimized ransac. *Pattern Recognition (PR)* (2003), 236–243.

[Com] Computational Vision Group: Caltech Computer Vision Datasets.

[CPM09] Chum O., Perdoch M., Matas J.: Geometric min-hashing: Finding a (thick) needle in a haystack. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2009), pp. 17–24.

[CSS07] Cour T., Srinivasan P., Shi J.: Balanced graph matching. In *Advances in Neural Information Processing System (NIPS)* (2007).

[CYHH07] Cheng H., Yan X., Han J., Hsu C.: Discriminative frequent pattern analysis for effective classification. In *International Conference on Data Engineering (ICDE)* (2007), pp. 716–725.

[CZ07] Chum O., Zisserman A.: An exemplar model for learning object classes. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2007).

[DBKP09] Duchenne O., Bach F., Kweon I., Ponce J.: A tensor-based algorithm for high-order graph matching. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[DH72] Duda R. O., Hart P. E.: Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM 15*, 1 (1972), 11–15.

[DPP08] Detry R., Pugeault N., Piater J. H.: Probabilistic pose recovery using learned hierarchical object models. In *International Conference on Computer Vision (ICCV)* (2008).

[DT05] Dalal N., Triggs B.: Histograms of oriented gradients for human detection. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2005), Schmid C., Soatto S., Tomasi C., (Eds.), vol. 2, pp. 886–893.

[EGWZ05] Everingham M., Gool L., Williams C., Zisserman A.: *Pascal visual object classes challenge results*. Tech. rep., 2005.

[EHOK01] Elad M., Hel-Or Y., Keshet R.: Pattern detection using a maximal rejection classifier. *Pattern Recognition Letters (PRL) 23* (2001), 1459–1471.

[ESPM05] Eveland C. K., Socolinsky D. A., Priebe C. E., Marchette D. J.: A hierarchical methodology for class detection problems with skewed priors. *Journal of Classification 22*, 1 (2005), 17–48.

[EU05] Epshtein B., Ullman S.: Feature hierarchies for object classification. In *International Conference on Computer Vision (ICCV)* (2005), pp. 220–227.

[EU07] Epshtein B., Ullman S.: Semantic hierarchies for recognizing objects and parts. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2007).

[EVGW*] Everingham M., Van Gool L., Williams C. K. I., Winn J., Zisserman A.: The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop.

[FB81] Fischler M. A., Bolles R. C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*, 6 (1981), 381–395.

[FE73] Fischler M. A., Elschlager R. A.: The representation and matching of pictorial structures. *IEEE Transactions on Computers 22*, 1 (1973), 67–92.

[FFFP06] Fei-Fei L., Fergus R., Perona P.: One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 28*, 4 (2006), 594.

[FFJS07] Ferrari V., Fevrier L., Jurie F., Schmid C.: Groups of adjacent contour segments for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (2007), 36–51.

[FFP05] Fei-Fei L., Perona P.: A bayesian hierarchical model for learning natural scene categories. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 524–531.

[FG08] Fleuret F., Geman D.: Stationary features and cat detection. *Journal of Machine Learning Research (JMLR) 9* (November 2008), 2549–2578.

[FGM10] Felzenszwalb P. F., Girshick R. B., McAllester D.: Cascade object detection with deformable part models. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2010).

[FGMR09] Felzenszwalb P. F., Girshick R. B., McAllester D., Ramanan D.: Object detection with discriminatively trained part based models. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence (PAMI) 99, PrePrints (2009).

[FH05] Felzenszwalb P. F., Huttenlocher D. P.: Pictorial structures for object recognition. *International Journal of Computer Vision (IJCV) 61*, 1 (2005), 55–79.

[FPZ03] Fergus R., Perona P., Zisserman A.: Object class recognition by unsupervised scale-invariant learning. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2003), vol. 2, pp. II–264–II–271 vol.2.

[FPZ05] Fergus R., Perona P., Zisserman A.: A sparse object category model for efficient learning and exhaustive recognition. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 380–387.

[FS95] Freund Y., Schapire R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory* (1995), pp. 23–37.

[FSGD08] Fischer B., Sauren M., Güld M. O., Deserno T. M.: Scene analysis with structural prototypes for content-based image retrieval in medicine. In *Medical Imaging* (2008), Reinhardt J. M., Pluim J. P. W., (Eds.), vol. 6914, SPIE.

[FT04] Ferrari V., Tuytelaars T.: Simultaneous object recognition and segmentation by image exploration. In *European Conference on Computer Vision (ECCV)* (2004), pp. 40–54.

[FTG06] Ferrari V., Tuytelaars T., Gool L.: Simultaneous object recognition and segmentation from single or multiple model views. *International Journal of Computer Vision (IJCV) 67*, 2 (2006), 159–188.

[GD05] Grauman K., Darrell T.: The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features. In *International Conference on Computer Vision (ICCV)* (2005), no. October, pp. 1458–1465.

[GHP07] Griffin G., Holub A., Perona P.: *Caltech-256 Object Category Dataset*. Tech. rep., California Institute of Technology, 2007.

[GLAM09] Gu C., Lim J. J., Arbelaez P., Malik J.: Recognition using regions. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[GN09] Gehler P., Nowozin S.: On feature combination for multiclass object classification. In *International Conference on Computer Vision (ICCV)* (October 2009).

[GR96] Gold S., Rangarajan A.: A graduated assignment algorithm for graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 18* (1996), 377 – 388.

[HALL05] Huang C., Ai H., Li Y., Lao S.: Vector boosting for rotation invariant multi-view face detection. In *International Conference on Computer Vision (ICCV)* (2005), pp. 446–453.

[HHIN09] Holzer S., Hinterstoisser S., Ilic S., Navab N.: Distance transform templates for object detection and pose estimation. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[HJS09] Harzallah H., Jurie F., Schmid C.: Combining efficient object localization and image classification. In *International Conference on Computer Vision (ICCV)* (sep 2009).

[HS88] Harris C., Stephens M.: A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference* (1988), pp. 147–151.

[JT05] Jurie F., Triggs B.: Creating efficient codebooks for visual recognition. In *International Conference on Computer Vision (ICCV)* (2005), pp. 604–610.

[Jur01] Jurie F.: Object recognition: solution of the simultaneous pose and correspondence problem. *Traitement du signal 18*, 5-6 (2001), 321–344.

[JWXD10] Jiang A., Wang C., Xiao B., Dai R.: A New Biologically Inspired Feature for Scene Image Classification. In *International Conference on Pattern Recognition (ICPR)* (Aug. 2010), pp. 758–761.

[KHP07] Kim G., Hebert M., Park S.-K.: Preliminary development of a line feature-based object recognition system for textureless indoor objects. In *Recent Progress in Robotics: Viable Robotic Service to Human*, Springer-LNCIS, (Ed.). Springer, 2007, ch. Perception Guided Navigation and Manipulation, pp. 255–268.

[KK91] Kim W.-Y., Kak A. C.: 3-d object recognition using bipartite matching embedded in discrete relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 13*, 3 (1991), 224–251.

[KK01] Kuramochi M., Karypis G.: Frequent subgraph discovery. In *International Conference on Data Mining (ICDM)* (2001), no. Icdm, pp. 313–320.

[KMM05] Kudo T., Maeda E., Matsumoto Y.: An application of boosting to graph classification. *Advances in neural information processing systems 17* (2005), 729–736.

[KP99] Kruizinga P., Petkov N.: Nonlinear operator for oriented texture. *Image Processing, IEEE Transactions on 8*, 10 (1999), 1395–1407.

[KP06] Kushal A., Ponce J.: Modeling 3d objects from stereo views and recognizing them in photographs. In *European Conference on Computer Vision (ECCV)* (2006), pp. 563–574.

[KS04] Ke Y., Sukthankar R.: Pca-sift: A more distinctive representation for local image descriptors. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2004), vol. 02, pp. 506–513.

[KTI03] Kashima H., Tsuda K., Inokuchi A.: Marginalized kernels between labeled graphs. In *Machine Learning* (2003), vol. 20, p. 321.

[KYdB08] Kootstra G., Ypma J., de Boer B.: Active exploration and keypoint clustering for object recognition. *2008 IEEE International Conference on Robotics and Automation* (May 2008), 1005–1010.

[LDJ06] Larlus D., Dorko G., Jurie F.: Creation de vocabulaires visuels efficaces pour la categorisation d'images. In *Reconnaissance des Formes et Intelligence Artificielle (RFIA)* (2006).

[LHS07] Leordeanu M., Hebert M., Sukthankar R.: Beyond Local Appearance: Category Recognition from Pairwise Interactions of Simple Features. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2007), pp. 1–8.

[LLF05] Lepetit V., Lagger P., Fua P.: Randomized trees for real-time keypoint recognition. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2005).

[LLS04] Leibe B., Leonardis A., Schiele B.: Combined object categorization and segmentation with an implicit shape model. In *European Conference on Computer Vision (ECCV)* (May 2004), pp. 17–32.

[Low99] Lowe D. G.: Object recognition from local scale-invariant features. In *International Conference on Computer Vision (ICCV)* (1999), pp. 1150–1157.

[Low01] Lowe D. G.: Local feature view clustering for 3d object recognition. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2001).

[Low04] Lowe D. G.: Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV) 60*, 2 (2004), 91–110.

[LSP04] Lazebnik S., Schmid C., Ponce J.: Semi-local affine parts for object recognition. In *British Machine Vision Conference (BMVC)* (2004), vol. 2, pp. 959–968.

[LSP05] Lazebnik S., Schmid C., Ponce J.: A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 27*, 8 (Aug. 2005), 1265–78.

[LSP06] Lazebnik S., Schmid C., Ponce J.: Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2006), vol. 2, pp. 2169–2178.

[LZL*05] Liu X., Zhang L., Li M., Zhang H., Wang D.: Boosting image classification with lda-based feature combination for digital photograph management. *Pattern Recognition (PR) 38*, 6 (2005), 887 – 901.

[MAFM08] Maire M., Arbelaez P., Fowlkes C., Malik J.: Using contours to detect and localize junctions in natural images. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2008).

[MB98] Messmer B. T., Bunke H.: A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 20*, 5 (1998), 493–504.

[MCUP02] Matas J., Chum O., Urban M., Pajdla T.: Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference (BMVC)* (2002), vol. 1, pp. 384–393.

[MGMR02] Melnik S., Garcia-Molina H., Rahm E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *International Conference on Data Engineering (ICDE)* (2002).

[MHK06] Mansur A., Hossain M., Kuno Y.: Integration of multiple methods for class and specific object recognition. In *International Symposium on Visual Computing (ISVC)* (2006), pp. I: 841–849.

[ML06] Mutch J., Lowe D. G.: Multiclass object recognition with sparse, localized features. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2006), pp. 11–18.

[MP05] Moreels P., Perona P.: *Probabilistic Coarse-To-Fine Object Recognition*. Tech. rep., California Institute of Technology, 2005.

[MP07] Moreels P., Perona P.: Evaluation of features detectors and descriptors based on 3d objects. *International Journal of Computer Vision (IJCV) 73*, 3 (2007), 263–284.

[MP08] Moreels P., Perona P.: A probabilistic cascade of detectors for individual object recognition. In *European Conference on Computer Vision (ECCV)* (2008), pp. 426–439.

[MS05] Mikolajczyk K., Schmid C.: A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 27*, 10 (2005), 1615–1630.

[MSHvdW07] Marszałek M., Schmid C., Harzallah H., van de Weijer J.: Learning object representations for visual object class recognition, oct 2007.

[MTEF06] Murphy K., Torralba A., Eaton D., Freeman W.: Object detection and localization using local and global features. In *Toward Category-Level Object Recognition*, Ponce J., Hebert M., Schmid C., Zisserman A., (Eds.), vol. 4170 of *Lecture Notes in Computer Science*. Springer, 2006, pp. 382–400.

[MTS*05] Mikolajczyk K., Tuytelaars T., Schmid C., Zisserman A., Matas J., Schaffalitzky F., Kadir T., Gool L. V.: A comparison of affine region detectors. *International Journal of Computer Vision 65*, 1/2 (2005), 43–72.

[NTU*07] Nowozin S., Tsuda K., Uno T., Kudo T., Bakir G.: Weighted Substructure Mining for Image Analysis. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2007), pp. 1–8.

[NWN96] Nayar S. K., Watanabe M., Noguchi M.: Real-time 100 object recognition system. *IEEE Transactions on Pattern Analysis and Machine Intelligence 18*, 12 (1996), 1186–1198.

[OA10] Ozdemir B., Aksoy S.: Image Classification Using Subgraph Histogram Representation. In *International Conference on Pattern Recognition (ICPR)* (Aug. 2010), vol. 1, pp. 1112–1115.

[OPFA06] Opelt A., Pinz A., Fussenegger M., Auer P.: Generic object recognition with boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 28*, 3 (Mar. 2006), 416–31.

[PCI*07] Philbin J., Chum O., Isard M., Sivic J., Zisserman A.: Object retrieval with large vocabularies and fast spatial matching. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2007), pp. 1–8.

[QFLG07] Quack T., Ferrari V., Leibe B., Gool L. V.: Efficient mining of frequent and distinctive feature configurations. In *International Conference on Computer Vision (ICCV)* (October 2007).

[Rev09] Revaud J.: The cs-17 dataset. `http://liris.cnrs.fr/jerome.revaud/datasets.shtml`, 2009.

[RLAB10] Revaud J., Lavoué G., Ariki Y., Baskurt A.: Learning an efficient and robust graph matching procedure for specific object recognition. In *International Conference on Pattern Recognition (ICPR)* (Aug. 2010).

[RLSP06] Rothganger F., Lazebnik S., Schmid C., Ponce J.: 3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International Journal of Computer Vision (IJCV) 66*, 3 (2006), 231–259.

[RS05a] Rosenhahn B., Sommer G.: Pose estimation in conformal geometric algebra part i: The stratification of mathematical spaces. *Journal of Mathematical Imaging and Vision 22*, 1 (2005), 27–48.

[RS05b] Rosenhahn B., Sommer G.: Pose estimation in conformal geometric algebra part ii: Real-time pose estimation using extended feature concepts. *Journal of Mathematical Imaging and Vision 22* (2005), 49–70.

[SBV01] Shearer K., Bunke H., Venkatesh S.: Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition (PR) 34*, 5 (May 2001), 1075–1091.

[Sch77] Schwartz E. L.: Spatial mapping in the primate sensory projection: analytic structure and relevance to perception. *Biological cybernetics 25*, 4 (February 1977), 181–194.

[SF] Sobel I., Feldman G.: A 3x3 isotropic gradient operator for image processing. presented at a talk at the Stanford Artificial Project in 1968.

[SI07] Siagian C., Itti L.: Rapid biologically-inspired scene classification using features shared with visual attention. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 29*, 2 (2007), 300–312.

[SM96] Schmid C., Mohr R.: Combining greyvalue invariants with local constraints for object recognition. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (1996).

[SP05] Scalzo F., Piater J. H.: Statistical learning of visual feature hierarchies. *Computer Vision and Pattern Recognition Workshops (CVPRW) 0* (2005), 44.

[SP06] Scalzo F., Piater J. H.: Unsupervised learning of dense hierarchical appearance represe. In *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition* (2006), pp. 395–398.

[SREZ05] Sivic J., Russell B., Efros A. A., Zisserman A.: Discovering objects and their location in images. In *International Conference on Computer Vision (ICCV)* (October 2005).

[SS98] Schapire R. E., Singer Y.: Improved boosting algorithms using confidence-rated predictions. In *Conference on Learning Theory (COLT)* (1998), pp. 80–91.

[SSSFF09] Sun M., Su H., Savarese S., Fei-Fei L.: A multi-view probabilistic model for 3d object classes. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2009).

[SWB*07] Serre T., Wolf L., Bileschi S., Riesenhuber M., Poggio T.: Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 29*, 3 (2007), 411–426.

[TCG10] Tirilly P., Claveau V., Gros P.: Distances and weighting schemes for bag of visual words image retrieval. In *Conference on Multimedia Information Retrieval (MIR)* (2010), pp. 323–332.

[TKR08] Torresani L., Kolmogorov V., Rother C.: Feature correspondence via graph matching: Models and global optimization. In *European Conference on Computer Vision (ECCV)* (2008), pp. 596–609.

[TLF08] Tola E., Lepetit V., Fua P.: A fast local descriptor for dense matching. *International Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2008), 1–8.

[TMF07] Torralba A., Murphy K. P., Freeman W. T.: Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 29*, 5 (2007), 854–869.

[Tor95] Torr P. H. S.: *Outlier detection and motion segmentation*. PhD thesis, Dept. of Engineering Science, University of Oxford, 1995.

[TWL*10] Ta A. P., Wolf C., Lavoué G., Baskurt A., Jolion J.-M.: Pairwise features for human action recognition . In *International Conference on Pattern Recognition (ICPR)* (Aug. 2010), IEEE, (Ed.).

[TWLB10] Ta A. P., Wolf C., Lavoué G., Baskurt A.: Recognizing and localizing individual activities through graph matching. In *International Conference on Advanced Video and Signal-Based Surveillance (AVSS)* (2010), IEEE, (Ed.).

[UE06] Ullman S., Epshtein B.: Visual classification by a hierarchy of extended fragments. *Lecture Notes in Computer Science 4170* (2006), 321–344.

[VF08] Vedaldi A., Fulkerson B.: VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[VGVZ09] Vedaldi A., Gulshan V., Varma M., Zisserman A.: Multiple kernels for object detection. In *International Conference on Computer Vision (ICCV)* (September 2009).

[VJ01] Viola P., Jones M.: Rapid object detection using a boosted cascade of simple features. In *International Conference on Computer Vision and Pattern Recognition (CVPR)* (2001), pp. 511–518.

[VJ04] Viola P., Jones M. J.: Robust real-time face detection. *International Journal of Computer Vision (IJCV) 57*, 2 (2004), 137–154.

[VNU03] Vidal-Naquet M., Ullman S.: Object recognition with informative features and linear classification. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision* (2003), p. 281.

[WAC*04] Willamowski J., Arregui D., Csurka G., Dance C., Fan L.: Categorizing nine visual classes using local appearance descriptors. In *ICPR Workshop on Learning for Adaptable Visual Systems* (2004), vol. 17, p. 21.

[WFKvdM97] Wiskott L., Fellous J.-M., Krüger N., von der Malsburg C.: Face recognition by elastic bunch graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 19*, 7 (1997), 775–779.

[WH99] Wilson R. C., Hancock E. R.: Graph matching with hierarchical discrete relaxation. *Pattern Recognition Letters (PRL) 20*, 10 (1999), 1041–1052.

[WZ01] Wang Y., Zhang H.: Content-based image orientation detection with support vector machines. In *CBAIVL '01: Proceedings of the IEEE Workshop on Content-based Access of Image and Video Libraries (CBAIVL'01)* (2001), p. 17.

[YM09] Yu G., Morel J.-M.: A fully affine invariant image comparison method. In *International Conference on Acoustics Speech and Signal Processing (ICASSP)* (2009), no. 1, pp. 597–1600.

[ZBMM06] Zhang H., Berg A., Maire M., Malik J.: Svm-knn: Discriminative nearest neighbor classification for visual category recognition. *International Conference on Computer Vision and Pattern Recognition (CVPR) 2* (2006), 2126–2136.

[ZC06] Zhang D.-q., Chang S.-F.: A Generative-Discriminative Hybrid Method for Multi-View Object Detection. pp. 2017–2024.

[zCLF09] Özuysal M., Calonder M., Lepetit V., Fua P.: Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 99*, 1 (2009).

[ZCY07] Zhu L., Chen Y., Yuille A.: Unsupervised learning of a probabilistic grammar for object detection and parsing. *Advances in neural information processing systems 19* (2007), 1617.

[ZYW*10] Zhang B., Ye G., Wang Y., Wang W., Xu J., Herman G., Yang Y.: Multi-class Graph Boosting with Subgraph Sharing for Object Recognition. In *International Conference on Pattern Recognition (ICPR)* (Aug. 2010), pp. 1541–1544.

Author's Publications

International Journals

- Improving Zernike Moments Comparison for Optimal Similarity and Rotation Angle Retrieval, **J. Revaud**, G. Lavoué, and A. Baskurt. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 31(4):627-636, 2009.

- Human-Robot Interface Using System Request Utterance Detection Based on Acoustic Features, T. Takiguchi, T. Yamagata, A. Sako, N. Miyake, **J. Revaud**, and Y. Ariki. In International Journal of Hybrid Information Technology, Vol. 1, No. 3, pp. 81-90, 2008.

International Conferences

- Fast and cheap object recognition by linear combination of views, **J. Revaud**, G. Lavoué, Y. Ariki, and A. Baskurt. In *Conference on Image and Video Retrieval (CIVR)*, pages 194–201, ACM, 2007.

- Scale-Invariant Proximity Graph for Fast Probabilistic Object Recognition, **J. Revaud**, G. Lavoué, Y. Ariki, and A. Baskurt. In *Conference on Image and Video Retrievals 2010 (CIVR)*, ACM.

- Learning an efficient and robust graph matching procedure for specific object recognition, **J. Revaud**, G. Lavoué, Y. Ariki, and A. Baskurt. In *International Conference on Pattern Recognition 2010 (ICPR)*.

- Human-Robot Interface Using System Request Utterance Detection Based on Acoustic Features, T. Takiguchi, T. Yamagata, A. Sako, N. Miyake, **J. Revaud**, and Y. Ariki.

In The 2nd International Conference on Multimedia and Ubiquitous Engineering (MUE2008), pp. 304-309, 2008-04.

Book Chapter

- Task-specific salience for object recognition, **J. Revaud**, G. Lavoué, Y. Ariki, and A. Baskurt. Chapter accepted for the book '*Innovations in Intelligent Image Analysis*', Springer-Verlag, 2011 (to be published).

Local Conferences:

- Combinaison de caractéristiques pour la reconnaissance rapide, robuste et invariante d'objets spécifiques, **J. Revaud**, G. Lavoué, Y. Ariki, and A. Baskurt. In *Reconnaissance des Formes et Intelligence Artificielle (RFIA)*, 2010.
- Une nouvelle mesure de distance entre descripteurs de moments de Zernike pour une similarité optimale et un angle de rotation entre les images, **J. Revaud**, G. Lavoué, and A. Baskurt. In *Compression et Representation des Signaux Audiovisuels (CORESA)*, 2009.