Efficient multi-class objet detection with a hierarchy of classes

Seyed Hamidreza Odabai Fard

Nº d'ordre : D.U. 2623
EDSPIC : 721

Université Blaise Pascal - Clermont II

Ecóle Doctorale
Sciences Pour l'Ingénieur de Clermont-Ferrand

Thèse présentée par :

Seyed Hamidreza Odabai Fard

en vue de l'obtention du grade de

Docteur d'Université

Spécialité : Vision pour la robotique

Efficient Multi-class Object Detection with a Hierarchy of Classes

---

Soutenue le 20 novembre 2015

Composition du Jury :

M. Christophe Garcia	PR - INSA de Lyon	Rapporteur
Mme Catherine Achard	MCF,HDR - UPMC Sorbonne Universités	Rapporteur
M. Matthieu Cord	PR - UPMC Sorbonne Universités	Examinateur
M. Stéphane Canu	PR - INSA de Rouen	Examinateur
M. Thierry Chateau	PR - Université Blaise Pascal	Directeur de thèse
M. Antoine Vacavant	MCF- Université d'Auvergne Clermont 1	Co-directeur de thèse
M. Quoc-Cuong Pham	DR - C.E.A.	Encadrant
M. Mohamed Chaouch	DR - C.E.A.	Encadrant

Abstract

Recent years have witnessed a competition in autonomous navigation for vehicles boosted by the advances in computer vision. The on-board cameras are capable of understanding the semantic content of the environment. A core component of this system is to localize and classify objects in urban scenes. There is a need to have multi-class object detection systems. Designing such an efficient system is a challenging and active research area. The algorithms can be found for applications in autonomous driving, object searches in images or video surveillance. The scale of object classes varies depending on the tasks.

The datasets for object detection started with containing one class only *e.g.* the popular INRIA Person dataset. Nowadays, we witness an expansion of the datasets consisting of more training data or number of object classes. This thesis proposes a solution to efficiently learn a multi-class object detector. The task of such a system is to localize all instances of target object classes in an input image. We distinguish between three major efficiency criteria. First, the detection performance measures the accuracy of detection. Second, we strive low execution times during run-time. Third, we address the scalability of our novel detection framework. The two previous criteria should scale suitably with the number of input classes and the training algorithm has to take a reasonable amount of time when learning with these larger datasets.

Although single-class object detection has seen a considerable improvement over the years, it still remains a challenge to create algorithms that work well with any number of classes. Most works on this subject extent these single-class detectors to work accordingly with multiple classes but remain hardly flexible to new object descriptors. Moreover, they do not consider all these three criteria at the same time. Others use a more traditional approach by iteratively executing a single-class detector for each target class which scales linearly in training time and run-time.

To tackle the challenges, we present a novel framework where for an input patch during detection the closest class is ranked highest. Background labels are rejected as negative samples. The detection goal is to find the highest scoring class. To this end, we derive a convex problem formulation that combines ranking and classification constraints. The accuracy of the system is improved by hierarchically arranging the classes into a tree of classifiers. The leaf nodes represent the individual classes and the intermediate nodes called *super-classes* group recursively these classes together. The super-classes benefit from the shared knowledge of their descending classes. All

these classifiers are learned in a joint optimization problem along with the previously mentioned constraints.

The increased number of classifiers are prohibitive to rapid execution times. The formulation of the detection goal naturally allows to use an adapted *tree traversal* algorithm to progressively search for the best class but reject early in the detection process the background samples and consequently reduce the system's run-time. Our system balances between detection performance and speed-up. We further experimented with feature reduction to decrease the overhead of applying the high-level classifiers in the tree. The framework is transparent to the used object descriptor where we implemented the histogram of orientated gradients and deformable part model both introduced in [Felzenszwalb et al., 2010a].

The capabilities of our system are demonstrated on two challenging datasets containing different object categories not necessarily semantically related. We evaluate both the detection performance with different number of classes and the scalability with respect to run-time. Our experiments show that this framework fulfills the requirements of a multi-class object detector and highlights the advantages of structuring class-level knowledge.

Keywords: multi-class object detection, hierarchical classification, rapid inference, tree of classifiers, tree traversal, hierarchical learning, structured SVM

Acknowledgments

This work was realized in the *CEA LIST/LVIC* team with the academic supervision of *University of Blaise Pascal* and *University of Auvergne*. I am grateful to many people who assisted in the creation of this dissertation.

First, I thank my advisers Thierry Chateau, Antoine Vacavant, Quoc-Cuong Pham and last but not least Mohamed Chaouch for their continuous technical and scientific supervisions. They built a complementary team and I was glad to had the chance working with them.

Next, I would like to thank the thesis committee members Christophe Garcia, Catherine Achard, Matthieu Cord and Stéphane Canu for taking their time, dealing with my work and the interesting discussions.

Furthermore, I am obliged to all my colleges from CEA: Adrien Chan-Hon-Tong, Pierrick Paillet, Boris Meden, Thierry Chesnais, Romain Gauthier, Bertrand Mermet, Olivier Sidibé, Damien Raffard, Loïc Fagot-bouquet, Juliette Bertrand, Solène Chan-long, Florian Chabot, members of the neighboring teams such as the 3D group and as well as every permanent member. A special thanks also goes to Odile Caminondo for her support and advises especially during difficult times of my PhD.

I would like to express my gratitude to my friends for keeping a close connection despite the long geographical distances. Charlotte Merillac and her friends fulfilled me with a great social life. Finally, I am grateful for my father, brother and his family who were always there for me during all these years.

Contents

CHAPTER 1

Introduction

Contents

TODAY 's search engines rely on an input text to deliver search queries. The request to search for content in images and not only the textual content of websites comes more and more to the fore. Current search engines analyze the metadata of the images *e.g.* user-supplied tags or captions to meat the user's request. For a machine to understand these images would require discriminating between approximately 30,000 object categories. This is the number of classes humans are roughly able to distinguish [Biederman, 1987]. Thus, the visual media stays opaque to machines. Recent research studies the challenge large scale object detection of objects in order to parse visual data.

Categorizing this large amount of classes opens new applications for robotics *e.g.* for assisted aid at home. The robots could localize objects and transport them to help elderly people for their daily tasks. Autonomous driving is another vivid research area which requires distinguishing among fewer object classes. Using cameras only, a car guides automatically through streets. For instance, to prevent accidents it is necessary to localize moving objects *e.g.* cars, bicycles, vans, buses, pedestrians or even static ones such as buildings. The scale of the objects is two order of magnitude. An order of magnitude fewer object categories are relevant for video surveillance in parking areas. The targets are intruders to a restricted or controlled zone and can be persons or vehicles. The Fig. 1.1 illustrates some applications possible with a multi-class object detector.

(a) (b) (c)

Figure 1.1 – Multi-class object detection is of high interest for the industry. We show possible applications. The number of target classes depends on the task's requirements. Some need to master thousands of classes while others only localize a dozen of object types. (a) Autonomous driving enables cars to transport a passenger from his position to a target *e.g.* useful to reduce the number of taxi drivers for Uber. (b) Surveillance of parking lots especially in risky areas can help to reduce the number of assaults. The system is able to identify objects in the scene and warn an operator for suspect activities. (c) Users can search on-line for object categories in images *e.g.* apples which is of interest for search engines.

The challenges and requirements vary with the application. We can illustrate this idea based on the execution time of a system. The multi-class object detection framework needs to process the input frames with adequate time limits for the defined categories. Processing can be done on powerful computers with multiple processors or on mobile devices. Ideally, a single system would meat these diverse needs. More challenges are mentioned in the next section. Our work focuses on recognizing instances of multiple object categories in images. We are not limited to any number of classes nor specific image scenes. This keeps the range of applications of our framework flexible to many domains. The studied context is detailed in the next section along with a look on this research field's challenges. We outline the foundations of our own system and close this chapter with the contributions.

1.1 Context for the study

We study the concept of object detection. It is a research field of computer vision. In the early years, an object detector needed to handle a single class. That is given an input image or frame of a video, the result is a set of possible locations of this specific object. In our work, a location is indicated with a bounding box containing the image coordinates of the found instance. Multi-class object detection goes one step further. Such a detector is able to distinguish more than one object class and returns the located instances along with the corresponding class labels. We do not limit us to

certain types of objects nor do we require a prior knowledge about the static image's scene structure. Fig. 1.2a shows an example workflow.

1.1.1 Challenges

Detecting more than one class raises the issues of feasibility during detection but also training. During detection, we consider two major performance criteria which is linked to the criteria of scalability. The first criteria concerns the detection performance of the system when handling multiple classes. How do the errors evolve with the number of classes? For instance, the bicycle and motorcycle detectors could both score positive for a bicycle object. Ideally, the confidence of the bicycle detector is higher than the concurrent detector. The system should be able to rank the correct class higher. The second criteria measures the run-time of the framework. How much time does evaluating a multi-class detector for a fixed number of classes take? How well does the algorithm scale with the number of classes?

A multi-class dataset contains the training images and annotations for all the target classes. Contrary to a single-class detector, the framework needs to master this increase in the training volume. The training phase has to finish in an adequate time. The memory limit should be further respected. This is a non negligible problem as it is not possible to load all the positive training data anymore depending on the object descriptor.

The design of an efficient algorithm comes along with the creation of an appropriate dataset. While annotating datasets used to be expensive, with the rise of Amazon Mechanical Turk it is possible to reduce these costs. Nevertheless, it is costly and time-consuming to annotate a huge number of instances and classes. These difficulties reflect on the efficiency of a multi-class algorithm. It is especially desirable to question the dependence of its performance on the number of training samples. We would like to have an algorithm that can generalize quickly. This is an accessible goal as we have similar classes and the training algorithm could *transfer* the knowledge between classes. The publicly available datasets comprise two parts usually having equal sizes: a training and a test set. Only the training images are allowed to be used in the training phase and the final algorithm is evaluated on the test images to get a comparative performance measure. The training images can be further split into a pure training and a validation set. The validation set is helpful to determine the algorithm's parameters.

The challenges are already numerous for the training of a single-class detector. An object appears with different dimensions and the detector only handles certain quantized sizes. An object's appearance can be changing with its pose or viewpoint. As a simple example, the dimension and appearance of a front car are very different from its side view's shape. The appearance is also dependent on scene's illumination. This factor can hide or emphasize certain contours of an object. The object's contours can be difficult to recognize based on its background. This issue is known as background clutter and can distract the object detector to produce errors. Moreover, the partial occlusion of an object heavily hinders the accuracy of a detector. Certain types of objects are more subject to occlusion in natural circumstances than others. Finally, the training algorithm needs to deal with inter-class and intra-class variability. The former represents the differences between the object categories. How to distinguish between

cars and bicycles? The latter represents the appearance changes in the class itself. We emphasized on this issue before when mentioning the pose and view challenges.

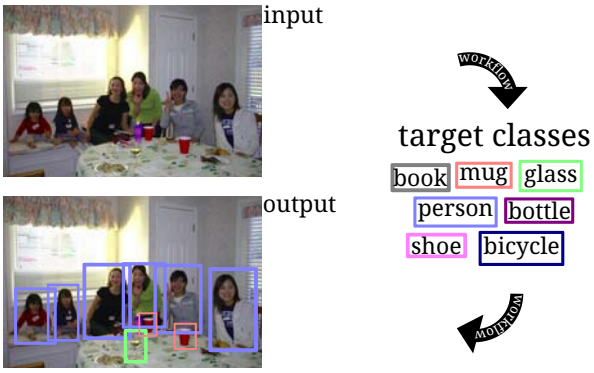
1.1.2 Learning Multiple Classes Through Shared Knowledge

There are multiple ways to tackle the design of a multi-class algorithm. Inspired by the human nature, most techniques take an intuitive approach. We will give concrete examples in Sec. 2. Objects of different categories can have many common characteristics. For instance, a bicycle and motorcycle have a close shape in most poses. These two classes can be easily confused. Both are equipped with a steering wheel or tires. *Part-based* frameworks learn to recognize parts and find evidence for an object class out of these parts. During learning, they extract a dictionary of parts and learn the coherence between parts and object classes. If the object detectors are learned independently, the size of this dictionary increases linearly and thus the run-time. One way to improve the run-time of evaluation is to reduce the number of part localizations. Indeed, the framework does not require to evaluate a tire twice when the shapes for the same parts of a bicycle and car coincide. Learning a joint dictionary keeps the number of entries small.

Other approaches create their own pool of parts upon very simple edge features. The objects are then reproduced with these parts where each part can be divided into smaller parts until the basic entry namely the edge feature is reached. Here again, we note an idea based on sharing features and parts among classes. But these approaches remain very specific to the choice of the descriptor and are hardly modular to adapt to new features. We will review these techniques in Sec. 2.

We follow a different intuition namely that of hierarchical processing of knowledge. [Tenenbaum et al., 2011] show the key role of structured knowledge to learn quickly new words. A sample hierarchy for object recognition is depicted in Fig. 1.2b. This principle is present in different domains where the structure can take a different form. The structure itself is determined by building a similarity matrix between all the object categories. To recognize objects, human beings refer to a hierarchical tree based representation of the object classes. The object classes are placed at the bottom of the tree. Neighboring leaf nodes have more characteristics in common. The intermediate nodes group these leaf node classes together into a more global object category. For instance the bicycle and motorcycle class are a category of a two wheeled vehicle. The depth of a leaf node indicates the distinctiveness of categories: categories that have many similar properties are grouped together increasing the depth of the branch. Their experiments show that this process helps adults to quicker generalize from very few examples. The knowledge about an already established concept helps to rapidly learn a novel one. We find structural relationship also with words as they can be put into a hierarchy based on their meanings as pursued in the WordNet project [Fellbaum and Tengi, 2010].

Generalizing from rare data is central for multi-class object detection *e.g.* to overcome the dataset limitations by reducing the number of necessary annotations. We chose to follow this observation and build a hierarchical multi-class object detector. Another name in this document for a detector or model is *filter*. In our case, a filter takes some input features and produces an output score. More generally, the score does



(a)

(b)

Figure 1.2 – (a) Example result obtained with a multi-class object detector. The target classes are those categories that the user aims to localize in the image. (b) Experiments in the work of [Tenenbaum et al., 2011] show that new objects sharing comparable characteristics with other classes can be ordered hierarchically and this process allows to generalize faster to new shapes.

not need to be discriminative and can constitute a partial element of a final decision function. If the filter's score is partial or represents the final score is depending on the meaning in the text. Every node in the tree represents a filter associated with a linear decision hyperplane. We do not need to make any further assumptions on the object descriptor. One challenge is to handle appropriately the background class which we cannot use as a leaf node. The background offers a too large variability that it is not possible to model it successfully with current features leading to model exclusively the foreground classes. Next, we need to formulate a detection goal that lets the correct class score higher than any other object class. Every node in the tree represents a filter attributing a score to an image region. In this context, the node's score is not necessarily discriminative and helps in the final decision making. This multitude of filters poses a new training difficulty especially when the framework aims to optimize them jointly. Both, the detection and training derivation have to assure an increase in detection performance compared to a simple baseline. We fixed the one-versus-all[1]

[1]More details in Sec. 2.2.1

technique as a baseline which exploits the same features but without a hierarchy. Finally, the execution time of such an approach needs to be considered. This is a major requirement as the number of nodes thus filters increases linearly with the number of classes.

The framework formulated in this work remains complementary to other achievements in the research community. To give an example, the combination with the work of [Sadeghi and Forsyth, 2013] allows to evaluate in a fast way all the filters lying on one branch and level of a tree. Though, this method should not be confused with a cascaded technique *e.g.* [Viola and Jones, 2001]. Each level in a cascade represents a detector which classifies an input. Based on the score, the input is forwarded to deeper cascades. The cascades work independently one from each other. Here, the final score for an input relies on the individual filter scores. Each filter is not an independent detector or entity. It's knowledge is shared with other filters in the tree. The levels are not *cut* into separate stages but score the input jointly.

1.1.3 Evaluation Measure

The performance measure varies depending on the applied dataset. We show the main detection performance measure used within our experiments. A detector can produce four types of outputs: a *true positive* is when the detected instance is correct; a *false positive* is when the detected instance is not the object; a *true negative* and a *false negative* are the same ideas applied to the background. The used datasets penalize multiple detections of the same object. Only the one with the highest score is counted as a true positive.

The user of the dataset delivers a list of bounding boxes with their confidence scores. Recent datasets provide the code to use this as an input and evaluate the performance of the detector. The detections are assigned to the ground truth annotations based on their common overlap ratio. This factor is the ratio between their common area and the union of the two boxes as detailed in [Everingham et al., 2012]:

$$\delta_{\mathrm{o}} = \frac{\mathrm{area}(B_p \cap B_{gt})}{\mathrm{area}(B_p \cup B_{gt})}$$

with B_{gt} being the ground truth bounding box and B_p the detector's bounding box. The detections must be close to the real bounding box and a threshold penalizes all the overlaps with a too small ratio. In our case, the threshold is set to 50%. The detections are sorted by their confidence scores. We need two more definitions. Precision is defined as the percentage of the true positives and all the detected objects. Recall is the fraction of the true positives and all the ground truth positives and measures how successfully the objects are detected. The precision/recall curve is then obtained by computing both values for all detections above a rank.

To have a quantified measure, the *average precision* is the mean precision calculated at eleven equally spaced recall points. For every point, we take the highest precision value above the recall point. Let the precision value for a recall point be given by $p(r)$.

Thus, the average precision defined in [Everingham et al., 2012] is given by:

$$AP = \frac{1}{11} \sum_{r \in 0, 0.1, \ldots, 1} \max_{\tilde{r} \geq r} p(\tilde{r})$$

When dealing with multiple classes, we automatically have multiple precision/recall curves. The *mean average precision* summarizes the results by simply averaging each average precision. The average precision is abbreviated as AP and the multi-class extension by mAP.

1.2 Thesis Outline and Contributions

In this thesis, we designed a multi-class framework that consists of a training and detection module. The training module uses the dataset and learns a model from scratch. The detection part takes this model as input to locate objects in images. Both parts can be called independently. Our system is designed with the goal to respect the two major issues discusses in Sec. 1.1.1 concerning the detection performance and run-time.

To better position our work, we start by reviewing related works. We mention multi-class datasets at the beginning of the chapter 2. In Sec. 2.2, we will have a look on single-class detectors which are fast due to a clever formulation or efficient implementation. These approaches are relevant as chaining multiple single-class detectors is a possible solution to multi-class object detection. The subsequent section Sec. 2.3 is divided into 3 major parts. It discusses other multi-class detection approaches. Multi-class detection dos not only require to limit itself to object's appearances. It can exploit *contextual information* as described in Sec. 2.3.2 to enrich the confidence in the presence of objects. Another interesting feature of multi-class frameworks is the ability of transferring and sharing knowledge which is mentioned in Sec. 2.3.3. This brick is essential for the design of scalable algorithms.

Chapter 3 introduces the concepts of learning with a support vector machine (SVM). This is followed by presenting the extension to multi-class classification using also SVMs. We focus on this machine learning mechanism as it constructs the backbone of our optimization module. We chose a SVM like technique as at the time of this work, the frameworks inspired by SVMs reached the best performances. Even though in recent years neural network based approaches gained significantly in popularity, we will see in the literature review that the features are passed through at the end to a simple one-versus-all technique again based on SVMs. This leaves space to chain a more powerful classifier like ours to the final layer.

The framework is introduced in Chapter 4. We start by formulating the detection objective and derive a **mixture of classification and ranking**. The ranking assures that all the scores for each class is comparative. The classification allows to reject background samples. This detection principle is new where we inspired us from research in image classification. In this field, multiple classes are common but does not have to treat a background label. This formulation can be combined with a hierarchical based approach. It consists in a tree where each node is a filter. The filters produce scores and the score of a class is given by the values of its predecessor filters. It is not possible

to fall back to a simple decision tree as detailed in Sec. 3.2.2 due to the presence of a background class. Afterward, we present an efficient way to learn the filters in the tree. Our method does a **joint optimization** which means that it learns all the filters in one optimization problem. This requires to manage both classification and ranking constraints at the same time. The tree is deduced at the beginning of the framework. At the end of the chapter, we compare our method with our baseline one-versus-all and conduct a thorough analysis. The performance of a tree of classifiers exceeds the baseline. This holds true if the classes are unrelated showing the importance of structured knowledge. We make the similar conclusion as the observations mentioned in Sec. 1.1.2. Hierarchy allows to better **transfer knowledge** and thus generalize quicker from few seen observations. This advantage can have a positive impact on the creation of future datasets.

However evaluating such a tree is time consuming due to the presence of many filters in the tree. We propose to solve this disadvantage by applying an adapted **tree traversal** algorithm as described in Chapter 5. It avoids searching all the paths in the tree and aims to evaluate only relevant filters to find the best scoring class. The first filters at the top of the hierarchy are processed more often. Therefore, we **reduce the dimensions of the filters** from bottom to the top of the tree. We show through empirical results that this method allows to outperform the baseline both in accuracy and speed-up. The algorithm can balance both values depending on the application's requirements. For instance, we show that for similar mean average precision as the baseline, the gain in run-time is approximately an order of magnitude.

The previous chapters used the famous histogram of oriented gradients of Sec. 2.2.1 as the underlying object descriptor for the evaluations. It is simple to implement and fast to extract. Chapter 6 implements a more complicated but **powerful feature extractor** that is dominant in the state-of-the-art. The descriptor models the global object's appearance but also its parts. A deformation cost restrains the possible locations of these parts with respect to an anchor position. The global appearance cost, the parts' appearance costs and the deformation costs are all learned using training examples. Also, the algorithm creates multiple components which can be considered as multiple discriminatory views. The consequence is longer training and detection times. The implementation at hand is a straightforward task but we noted the relevance of minor bricks which have a considerable impact on the detection performance.

Finally, we end our work by discussing future research perspectives. We propose complementary ways to improve the detection performance and run-time. We aim at the same time to make the framework more scalable to high number of training data and classes. We discuss our observations and difficulties. Final notes round up this dissertation.

1.3 List of Publications

The underlying work and contributions lead to the following list of publications:

- [Odabai Fard et al., 2013] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2013). Détection hiérarchique multi-classes d'objets dans les images. In *ORASIS*, Cluny, France

- [Odabai Fard et al., 2014c] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2014c). Joint Learning for Multi-class Object Detection. In *International conference on computer vision theory and applications (VISAPP)*, Lisbon, Portugal

- [Odabai Fard et al., 2014b] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2014b). Joint Hierarchical Learning for Efficient Multi-class Object Detection. In *IEEE Winter Conference on applications of computer vision (WACV)*, Steamboat Springs, Co, USA

- [Odabai Fard et al., 2014a] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2014a). Apprentissage hiérarchique simultané pour la détection efficace d'objets. In *Reconnaissance de Formes et Intelligence Artificielle (RFIA)*, France

- [Gadeski et al., 2014] Etienne Gadeski and Hamidreza Odabai Fard and Hervé Le Borgne (2014). Gpu deformable part model for object recognition. *Journal of Real-Time Image Processing (JRTIP)*

We are currently writing an article to submit for *Pattern Recognition* journal which summarizes our complete work.

CHAPTER 2

Related Work

Contents

In the last four years, multi-class object detection has increasingly gained in popularity in the research community. This comes through its huge range of applicability. It is of importance in fields such as object detection for autonomous driving, traffic sign detection or searching in image collections for object categories. The number of classes k depends on the related application. Recognizing objects in urban scene often only requires distinguishing between a dozen of objects *e.g.* persons, cars, bicycles, motorcycles, vans, trucks and some more. Many more classes up to a scale of thousands are needed when dealing with image understanding with the goal of identifying as many objects as possible in an image. [Biederman, 1987] estimates that humans differentiate about 30,000 categories.

The current research results still do not fulfill consumer satisfaction. A multi-class object detection system needs to face many obstacles:

Performance and run-time trade-off An ultimate goal of any object detection system is to detect with high reliability and as quick as possible. Fulfilling these criteria can be challenging. For instance, the multi-class technique in [Dean et al., 2013] comes with a gain in run-time at the cost of a decreased detection perfor-

mance. Some approaches allow to increase gain in performance while sacrificing detection time [Salakhutdinov et al., 2011]. This balance between detection time and performance is an important design choice and ideally one would have a system which allows to handle this kind of trade-off.

Scalable run-time The testing time has to scale appropriately with the number of classes. This can even be an issue for applications that require today a small number of discernible objects but for which the requirements can change over time.

Scalable training time The popular deformable part based model of [Felzenszwalb et al., 2010a] needs one day to train a pedestrian class having less than 4,000 examples with MATLAB or 4 hours with our C++ coded version. In industry, companies have access to big amount of annotated training images. Training hundred or even thousands of classes one after another would take a considerable amount of time. Therefore, it is of importance to build training algorithms that scale easily with the number of classes k.

Memory usage This is of relevance both at training time and testing time. Annotations of one class are used when training a single-class detector. Training the detector for multiple classes jointly would require to cope with all the annotations which can be memory consuming. It is of general interest to keep the memory footprint small which opens the doors for mobile applications.

Mono-class challenges A multi-class system inherits the challenges of detecting a single class. One of them is occlusion that is the object is seen partially. This changes the appearance shape of the object and the extracted feature vector deviates from the originally learned model. While occlusion happens when an object hides a target class, another issue is the background of the object itself. This difficulty is known as background clutter and can modify too the appearance information of the object. Another problem is the pose of the object which is due to the possible articulations of it. A casual object consists of parts which have a certain degree of deformation. Advanced object descriptors are able to capture these variations. This challenge is augmented with the multiple views upon an object. A frontal view of a car is very different from its side view to just give an illustration. Again some methods try to capture this multi-view diversity in their frameworks. Finally, many systems rely on a fixed size model as we will see in Sec. 2.2.1. Noise due to resolution changes between the ideal model and the extracted features at different sizes of an object impact the efficiency of a system.

In next sections, we will review prior work relevant to a multi-class system. Before starting to review object detectors in this chapter, we mention common datasets in Sec. 2.1. Many such systems rely on algorithms developed for single-class object detection. Therefore, we will review in Sec. 2.2 methods originally developed to locate a single category but which have low run-times. Naive extensions of these methods already allow to have a multi-class framework. The Sec. 2.3 describes systems designed for multi-class detection purposes. The algorithms try to balance the criteria of fast and accurate detections. Further, we mention how previous works address the

challenges of scalability. For instance, this can be achieved by sharing and exploiting knowledge of several classes to learn and detect new object categories.

As we already discussed in Sec. 1.1.2, a *filter* can designate a detector but also a module that scores its input. Usually when speaking of a single-class detector, a filter represents the detector itself. However, sometimes many filters together create the final decision score. In this case, there is a difference between detector and filter.

2.1 Datasets

A dataset is a collection of data. Its content can vary depending on the research domain. Here, we are interested in datasets containing strictly more than one class. The minimal requirement for our framework is to provide annotations which indicate the location of the object instance in the image. We will mention now the most relevant datasets. Some image examples are illustrated in Fig. 2.1.

PASCAL VOC The PASCAL Visual Object Classes (PASCAL VOC) [Everingham et al., 2012] provides data for the field of object detection and other topics such as image segmentation. The dataset is no longer maintained and the dataset changed every year between 2005 and 2012. Depending on the year, the results could be evaluated locally on the personal computer or on an evaluation server. PASCAL VOC 2007, sometimes abbreviated PASCAL VOC07, has a total of 9,963 images which contains 24,640 annotated objects partitioned into $k = 20$ distinct classes. The dataset is split in half for training purposes and the other half for testing the detection models. The training part can be further split into *train* and *validation* groups. The bounding boxes are augmented with additional information *e.g.* viewpoint of the object or if it is a difficult example because of severe occlusion. MATLAB tools are provided to load annotations or to evaluate results.

SUN [Choi et al., 2010] introduced the SUN09 dataset. It contains 12,000 images with more than 200 object categories. A total of 152,000 instances are annotated. Compared to the PASCAL VOC datasets, it has the advantage of having images containing many more annotated objects. This property makes it useful for learning contextual information in scenes.

The previous dataset has not to be confused with the Scene Understanding database "SUN" [Xiao et al., 2010]. It contains a total of 131067 images with 4479 object categories. Again a toolbox is provided for easiness of integrating this dataset to a project. The website [Xiao et al., 2012] allows to browse through instances of objects. This dataset is also suitable for scene categorization.

LabelMe This is an ever evolving dataset [Russell et al., 2008] which can be modified in the web browser online. Every body can contribute to this dataset. There exists even an app for tablets and mobile phones to accelerate this task. It contained a total of 62,197 annotated images with 658,992 labeled objects. This dataset has not been often used in benchmarking object detection systems.

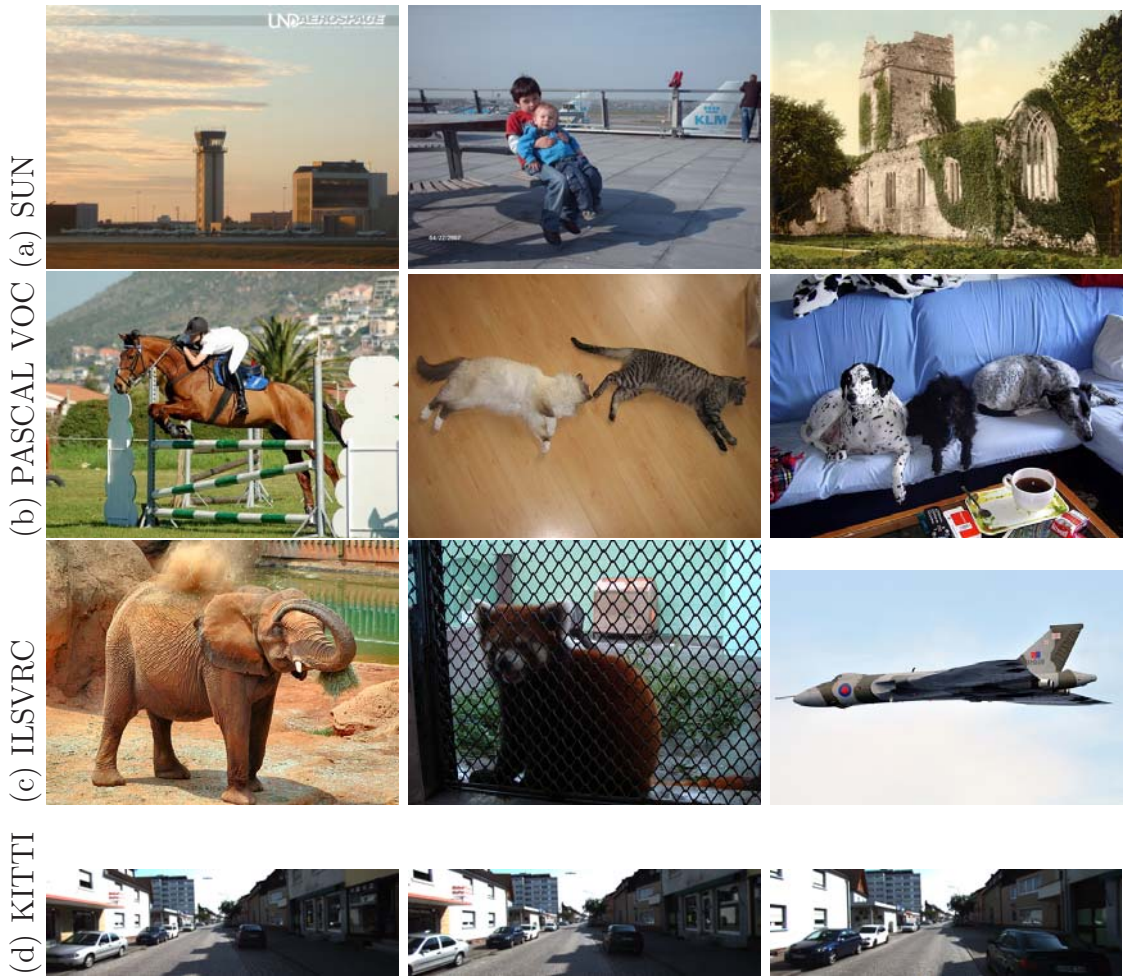


Figure 2.1 – Illustration of images contained in different datasets.

KITTI This dataset was introduced in [Geiger et al., 2013] to improve technologies around autonomous driving. A car recorded long video sequences in urban scenes using various sensors *e.g.* laser scanner, stereo camera and GPS. The dataset with its corresponding website [Geiger et al., 2012] offers a platform for evaluating algorithms in object detection, tracking, odometry, flow estimation and many more domains. The object detection benchmark has 7481 training images and 7518 test images. The images contain a total of 80,256 annotated objects. Each annotation contains the orientation and whether the difficulty of detecting that object. Only a total of three classes participate at the object detection challenge namely the car, bicycle and pedestrian classes.

ILSVRC The Large Scale Visual Recognition Challenge (ILSVRC) [Russakovsky et al., 2015] is an effort to evaluate object detection and classification algorithms on a large scale. It first appeared in 2010 and is improved every year. We will shortly present the 2014 version. The benchmark consists of 200 classes spread over 476688 training and validation images sets and 40,152 test images. For training a total of 534,309 objects are given which is one order of magnitude bigger than PASCAL VOC.

For the evaluations, we exploited only the PASCAL VOC datasets 2007 and 2010. It comes with 20 classes where some subsets of them are semantically close. The KITTI dataset focuses only on 3 categories which is too little to accentuate important properties of our system. The other datasets contain many more classes where the training and evaluation would take more time than on the PASCAL VOC dataset. Moreover, the latter is more popular and strongly used in many research works.

2.2 Single-class Object Detection

Efficient single-class detectors are of special interest. First, many multi-class detectors are an extension of single-class frameworks. Second as we will detail in Sec. 2.2.1, one can chain several single-class detectors to obtain a multi-class system. Each of these single-class detectors are dedicated to only one class. The most confident detector assigns the class label. We will briefly mention some famous concepts of algorithms without going into further details as this is not the focus of this work. It helps us to better understand single-class detectors.

Artificial neural network (ANN) [Bishop, 2006] is a computational model inspired by the central nervous system. An input signal is propagated through a network of nodes called neurons. The neurons are often arranged in layers. Each neuron has several input and output connections. The elements of the input are multiplied by the weights of the connections. These input signals are transformed to generate an output signal. In its easiest case, the output value is the sum of the weighted input values. ANNs and its variants gained more and more in popularity in computer vision since 2009 in applications such as handwriting recognition [Matan et al., 1992], traffic sign recognition [Ciresan et al., 2011] and achieve best results on PASCAL VOC or ILSVRC datasets in object detection [Girshick et al., 2014] which exploit a convolutional neural network to transform image regions into a new feature representation.

Random forest uses an ensemble of decision trees. Each decision tree takes an input and outputs a decision by traversing a tree. Each node in the tree is usually a simple decision function. The outputs of the multiple trees are combined and a final decision is taken. This classification technique resulted in state-of-the-art results in object detection *e.g.* in the works of [Gall and Lempitsky, 2009, Razavi et al., 2012] which are inspired by the work of [Leibe and Schiele, 2003]. The idea illustrated in Fig. 2.2 is to use parts of an object to localize it. The parts must have a spatial coherence and each part votes for the center of an object. For instance, the head of a person votes for the person class at its chest location. The evidence of a object's presence depends on the number of votes concentrated at a position. The more parts vote for an object class at the same center position, the higher is the confidence of it. The parts are extracting during training in a completely unsupervised way. The random forest is applied in this context as to accelerate the matching between the parts and the image regions. The classes are learned independently of each other and the size of their dictionaries regrouping the parts grow fast.

Boosting [Freund and Schapire, 1997] is based on the idea of aggregating several weak classifiers to build a strong one. Often the weak classifiers make use of one feature in the feature vector for taking a decision. During detection, a set of pre-trained weak classifiers is applied where each classifier takes a binary decision. The weighted sum

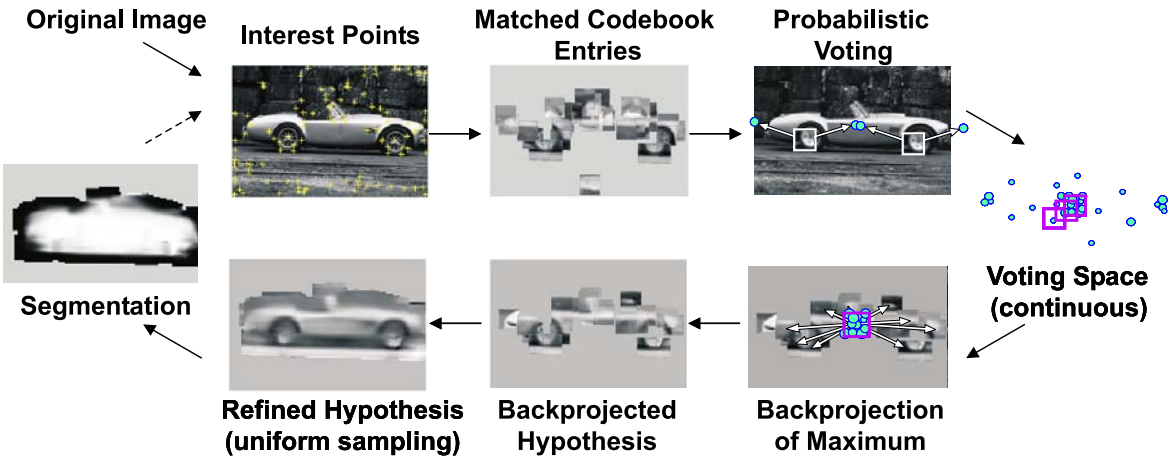


Figure 2.2 – The detection chain for the detector of [Leibe and Schiele, 2003]. Parts from a dictionary are matched to specific image regions. They vote for an object center. The confidence in the detection depends on the density of these votes. An advantage of their method is the obtainment of the rough segmentation of the object (Courtesy of [Leibe and Schiele, 2003]).

of these decisions contribute to the final strong classification. Boosting has found a strong success since the rapid object detection framework of Viola and Jones [Viola and Jones, 2001]. Torralba *et al.* proposed a multi-class boosting technique in [Torralba et al., 2004, Torralba et al., 2007] where the classes share the answers of some weak-classifiers.

Support vector machine (SVM) is another highly exploited learning algorithm. It separates the data into two distinct categories dependent on the distance to a trained hyperplane. We distinguish between linear and non-linear SVMs. The former is suited to classify data that are separable by a linear hyperplane. The latter is more general as it allows to learn a model for many possible forms of data. Due to its success and strong mathematical formulation, SVMs have been studied in theoretical form and have been applied to many domains.

Our work differentiates strongly with chaining single-class detectors. The latter is called one-versus-all and the classes do not share common knowledge and features. The presented framework in this thesis is based on a tree of classifiers and we show the importance of hierarchical knowledge compared to a flat structures as in one-versus-all. In the next section, we describe an object detector based upon SVMs. This helps to better understand our own framework which is built upon similar bricks and introduces some essential notations.

2.2.1 Preliminaries: a Basic Sliding Window Detector

There are many different object detection principles. In this section we given an introduction to sliding window detection systems and its notions. It helps to better understand the remaining sections. We focus on the object detector of [Dalal and Triggs, 2005]. This detection framework was among the top performing systems in

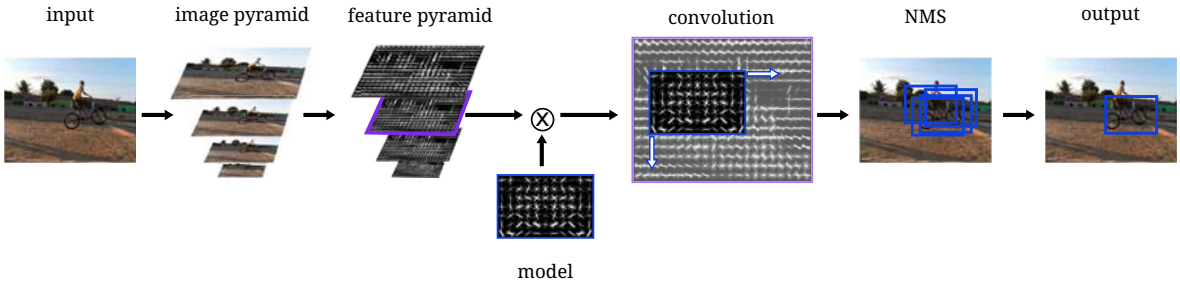


Figure 2.3 – Illustration of a sample detection chain. At first the input image is scaled at various sizes and for each scale the features *e.g.* HOG are extracted. A convolution is computed between the model and the feature pyramid for each location. This produces a score for each position. The best detection is filtered out of the neighboring responses in the non-maximum suppression step.

the early PASCAL VOC competition in 2006. They introduced a new object descriptor which captures the edges, is robust to slight appearance changes and illumination variations. It was widely used in many more frameworks especially between 2005 and 2010. However, it lost in its popularity and was replaced by its slight variant which was introduced in [Felzenszwalb et al., 2008b]. In this work, we applied this latter descriptor as it is more compact and outperforms its original formulation.

An image is stored in pixels and RGB values. A computer is unable to understand a region in an image using only RGB values. One important feature that lets human distinguish among objects is the contour of the object *e.g.* a balloon is round or a house has rectangular form. The first step in the detection chain consists in extracting meaningful features namely the HOG features. The image is grouped into a grid of cells where each cell contains a 31 dimensional feature vector. This procedure is repeated for the different scales of the image to detect objects of various sizes. The next step crops a region of cells. This region has the same width and height than the learned model. The features in this region and the model can be concatenated each into a vector and the dot product of these vectors gives the confidence of seeing the desired object class in the region. The regions are selected by sliding a window over the currently scaled image. Regions scoring higher than a threshold are classified as positive otherwise as a background region. The last step eliminates multiple strongly overlapping positive responses. This principle is depicted in Fig. 2.3. The following paragraphs give more insight into each step.

Histogram of Orientated Gradients

The feature extractor takes an input image and returns a high dimensional feature vector of the image. As we will see, the image is partitioned into cells and each cell contains a f-dimensional vector of floating point values. Each pixel is represented through its red, green and blue intensities. A simple gradient is computed for each pixel and channel. The gradient with the maximum norm for each pixel is kept and its orientation is quantized into 18 bins. The pixels are grouped into cells in our case of size 8x8. Each cell contains two histograms of gradient orientations where the norm

of each pixel's gradient influences the importance of that orientation. There is one histogram for 9 orientation bins and another for 18 bins.

At this stage, we have a 27 dimensional feature vector per cell which is still sensitive to illumination changes. These histograms are normalized by the norm of the neighboring cells' feature vectors. This is repeated 4 times, each time for a different cell group. Consequently, this increases by 4 the feature dimension per block.

In summary, each cell contains two histograms which have together a length of 27 and are normalized in four different ways. This makes a total of 108 values for only one cell. The dimension increases quickly as an image can have several thousands of cells. Therefore, the authors suggested to reduce this dimension of 108 to 31 features by projecting it into a lower dimensional space with PCA. This is done quickly by a modified version of the eigenvectors. The feature map is the matrix or vector containing the histograms of all the cells.

We have the option to accelerate the HOG feature computation in our framework using the techniques presented in [Yan et al., 2014]. It notably uses a hash tables filled off-line for obtaining quickly the orientation and magnitude of the pixels. The computation is exact as the color values of each pixel's channel is restrained between $[0, 255]$.

Feature Pyramid

An image contains objects of various sizes. The learned model has fixed dimensions and only allows detecting pedestrians corresponding to its own dimensions. The image is therefore scaled to be able to localize pedestrians at different dimensions. In this framework, the input image is only downscaled which allows to localize large objects but not small ones. These are neglected due to their imprecise appearances. During detection, we downscale the image with a scaling factor of 1.07. The image is downscaled until reaching at most 5x5 cells after feature extraction. The effect of anti-aliasing is reduced by downscaling the previously scaled image each time. The feature pyramid is a concatenation of all the feature maps for all the scales.

Convolution

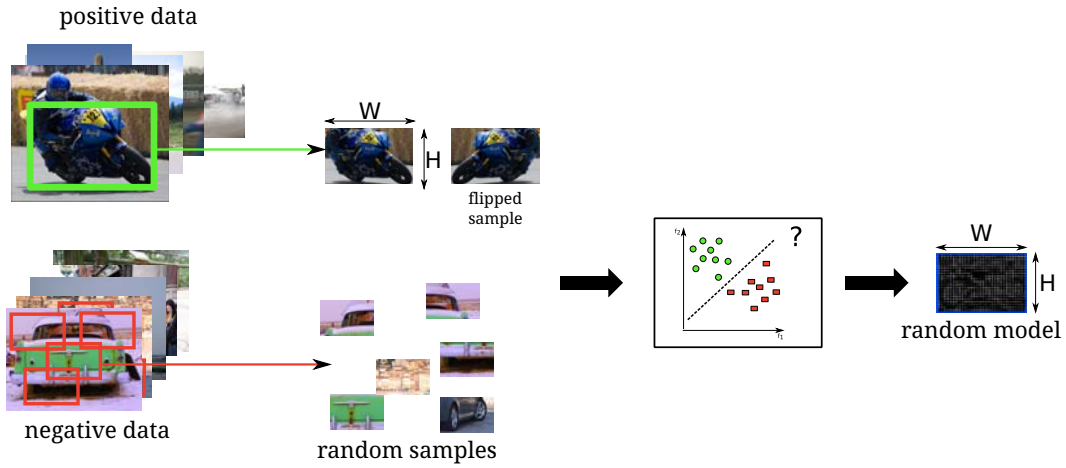
The most time consuming step is the convolution of the model with the feature pyramid. The feature pyramid is processed sequentially. For each scale in the feature pyramid, the learned model is slided over and for each position in it, the dot product between the model weights and the image's features are computed. This results in a score per cell in the feature map:

$$\text{score}(x) = w \cdot x \tag{2.1}$$

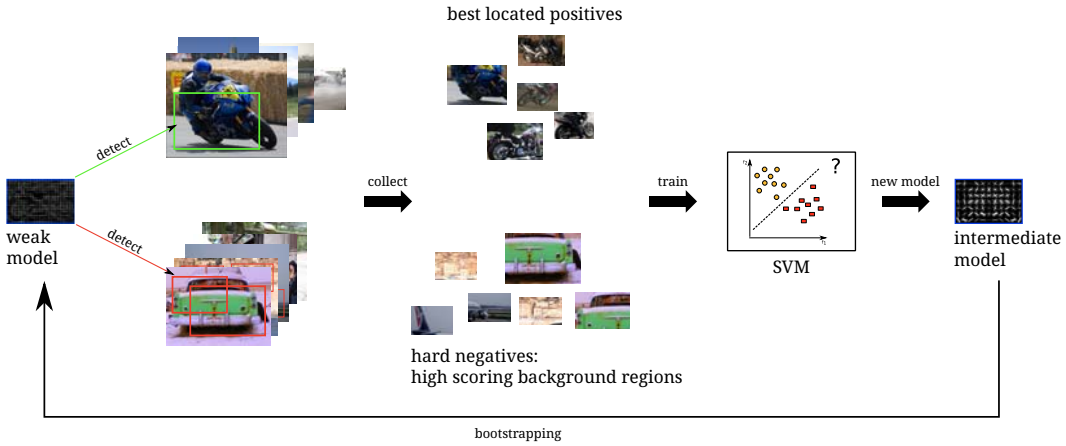
with w being the model. x is the HOG feature extracted around a region at a position in the feature pyramid. This score is called appearance score. Later, we will deal with another type of scores given by the deformation penalty.

Non-maximum Suppression

The non-maximum suppression (NMS) handles overlapping detections. This step can usually be found after having obtained the maps of scores one map for each level in the



(a)



(b)

Figure 2.4 – Training stages for our basic detector using histogram of gradients (HOG) as features. (a) The random phase which builds a basic model by collecting warped positive data and randomly selected image patches. (b) The basic detector is used to create iteratively the final model by improving its detection accuracy and better discriminating hard to classify background samples.

feature pyramid. The locations which score sufficiently high are retained. However, these instances can overlap with each other. The NMS resolves this kind of conflicts as two objects cannot occupy the same location in space. We applied the NMS step of [Felzenszwalb et al., 2010b] which is commonly cited in literature. The scores are sorted in decreasing order. Starting with the highest score, this list is traversed and all the highly overlapping entries in the list are removed. Usually, the overlapping threshold is fixed to 50%. The final result is a list of estimated object locations.

Training

The goal of a training framework is to derive a robust model $w \in \mathbb{R}^{WxHxd}$. The model has width W and height H which makes it a total of WxH cells. Each cell contains a

f-dimensional vector of weights. The dimensions W and H are chosen to be the most common mode in the training set and its surface bigger than 20% of the annotations.

The next step is to learn the weights w. This is done in two phases. Phase one takes positive images and warps them to model dimensions. The examples are also flipped to consider mirrored version of the object. This assumes a vertical symmetrical property of objects and increases artificially the dataset. Another advantage is that we do not learn WxHxd parameters but only optimize the half of it $\frac{W}{2}$xHxd namely the left hand side of the weight vector thus reducing the risk of overfitting. Once the left hand side is optimized, we flipped it and obtain the right hand side of w. The negative data is collected by sampling randomly background regions through all images. With the corresponding extracted features and a learning algorithm such as SVM [Joachims, 1999a] a weak classifier is obtained which is called the *random model*. The process is illustrated in Fig. 2.5a.

The second phase has two objectives. The first one is to optimize the features extracted from positives examples. We do not warp the annotated samples to model dimensions but use the currently learned model to detect the positive example ensuring a sufficient overlap *e.g.* 70% with the annotation. This immediately optimizes the detector based on the levels in the feature pyramid. The second goal is to find more important negative samples. We cannot load all negatives into the memory. This would be time and space consuming but has the advantage that the SVM algorithm knows the support vectors. In the case of SVM, the support vectors are the crucial information needed to obtain the optimal hyperplane. But during the first training phase, it is not ensured to collect these support vector when randomly sampling negative data. Therefore in this second phase, the weak model classifies the background training images and *hard* negatives are retained. These high scoring negatives are close or on the wrong side of the hyperplane and thus necessary examples for the SVM solver. This concept is called *boostrapping* and is relevant to finding more meaningful samples. This phase is repeated several times which drastically increases performance at the cost of an increased training time. An illustration is shown in Fig. 2.5b. The obtained model is now called the *hard model* and can be used as the final result of the framework.

Simple Multi-class Detector

Now that we know how to create a single-class detector, let us see the basic extension to a multi-class detector shown in Fig. 2.5. In the case of multiple categories, the example x can belong to a variety of classes $\mathcal{Y} = \{\mathcal{Y}^+, y_{\text{bg}}\}$ with $\mathcal{Y}^+ = \{y_1, ..., y_k\}$ being the labels of the k positive classes and y_{bg} the negative background label. We have also a multitude of decision hyperplanes w_i with $1 \leq i \leq k$ that is one hyperplane for each positive class. The score of each class is given by the dot product of its weight vector and the extracted object features x:

$$\text{score}_{y_i}(x) = w_i^{\text{T}} x$$

The score can be viewed as a measure of confidence that x is an object of the class y_i. The estimated class y^* is given by the highest scoring class. If the best score is too

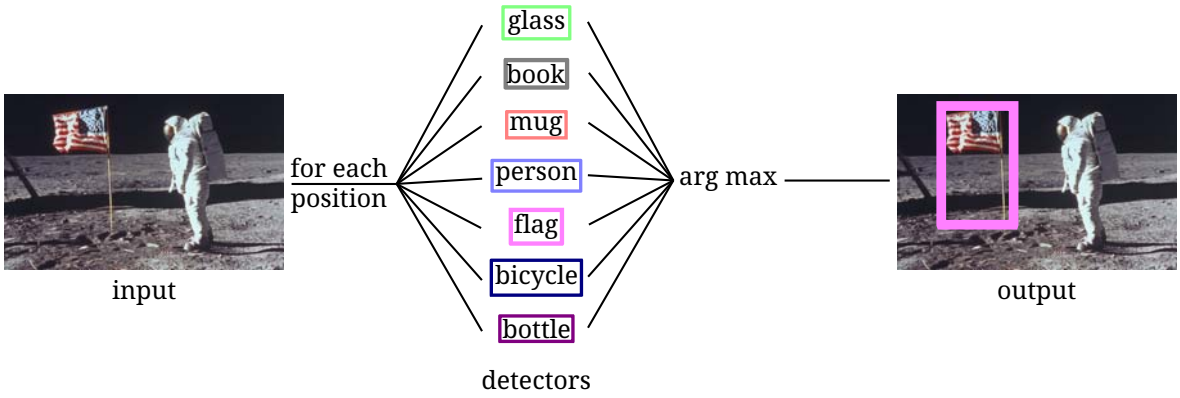


Figure 2.5 – A solution to the multi-class problem is the traditional one-versus-all technique. This figure illustrates the pipeline: At each position in the feature pyramid, the k detectors are evaluated and the label of that position is assigned immediately to the highest scoring class. However, if the confidence in that score is too low, the object is rejected as background.

low, the object is discriminated as background:

$$y^* = \begin{cases} \tilde{y} = \arg\max_{y_i \in \mathcal{Y}^+} \text{score}_{y_i}(x) & \text{, if } \text{score}_{\tilde{y}}(x) \geq 0 \\ y_{\text{bg}} & \text{, otherwise} \end{cases}$$

The label depends on the confidences of all classes and their scores are compared to each other. The weight vectors of the classes have different dimensions and they are optimized independently of each other. Their scores may not be comparable. One detector can produce high scores as another class produces only scores close to zero. In the case of SVM[1], the scores of each class are not a probabilistic measure. Thus, it is convenient to get a degree of certainty about the classification $\text{P}(y = y_i|x)$.

Platt scaling as introduced in [Platt, 1999] gives a solution to achieve this output by fitting an exponential function to the raw score:

$$\text{P}(y_i = 1|x) = \frac{1}{1 + \exp(A \text{ score}(x) + B)}. \tag{2.2}$$

The two parameters A and B are learned during training by applying a Levenberg-Marquardt algorithm. We employed the numerical more stable version of [Lin et al., 2007] which is also employed in LIBSVM [Chang and Lin, 2011]. Now, we simply have to take the maximum of the probability estimates to obtain the most confident label. The training process is straightforward. One class after another is trained as described in Sec. 2.2.1. The order of classes is not relevant for the final multi-class output.

The Deformable Part Model

At the time of this thesis, the most successful detector was the *deformable part model* [Felzenszwalb et al., 2010a] (DPM). It reaches best performances especially on highly deformable datasets *e.g.* PASCAL VOC. Many recent research works inspire their

[1]Please refer to Sec. 3.1 for an introduction to SVM.

framework on this method or extend it in many ways. The model presented before captures the global object appearance. But an object consists also of articulated *parts* which move around. This can simply be a head of an animal. Its location is flexible. The DPM can be seen as an extension to the HOG model described earlier as it models parts and their deformations.

Parts are described by their appearances using HOG features. They have an ideal anchor position. Any deviation from this position is penalized with a quadratic *deformation cost*. The detection score is the sum of the confidence in the root appearance, part appearances and the deformation costs. Each part is placed in a way to maximize the appearance score and minimize the penalty function. It is a single-class detector and therefore one class after another is trained and applied during detection. The parts for each class are not shared nor interact together. To face the multiple view challenge of an object, the authors create multiple *components*. Each component represents one characteristical view of the object.

The training process is completely automatic where one key parameter is the number of components and parts. No parts nor views of the object need to be specified by the annotator. The specific views, also called components, are obtained by clustering the aspect ratios of the object's annotations. After learning a simple HOG model for each component, the parts are added at the most important locations specified by the simple HOG model. A location is determined by a high concentration of SVM weights. Then, the learning algorithm alternates between learning the parts' appearances and positions on the training examples.

The framework is fast and performs well compared to other state-of-the-art methods and the entire code is publicly available. This makes the DPM interesting for the research community as we will see in the next sections.

2.2.2 Algorithmically Accelerated Methods

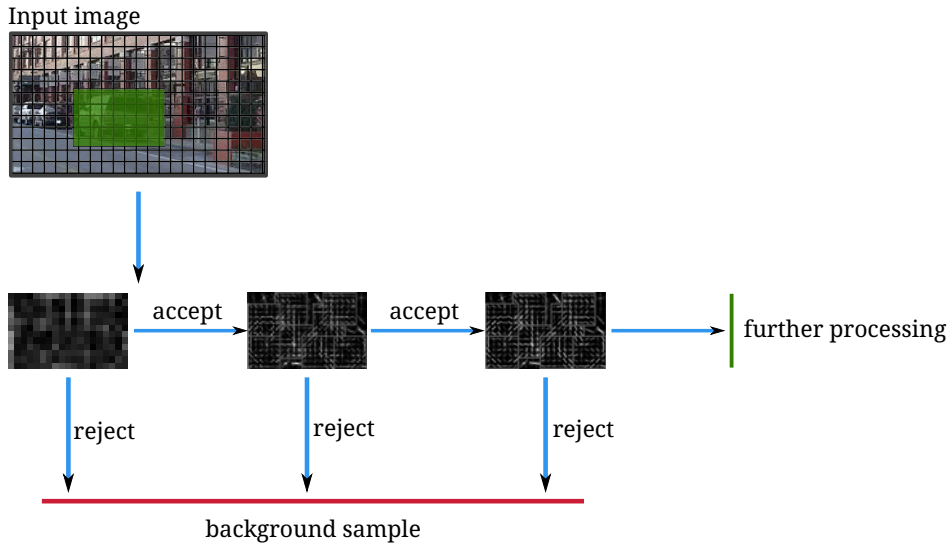
The basic step to have a multi-class object detector is to combine the output of all the individual single-class detectors to what is known as a One-versus-All (OvA) technique as further detailed in Sec. 2.2.1. Each such detector produces a score $\text{score}_y(x)$ for the class y and training sample x. The maximum score defines the detected label y^* of the example.

$$y^* = \arg \max_{y \in \{1,\ldots,k\}} \text{score}_y(x).$$

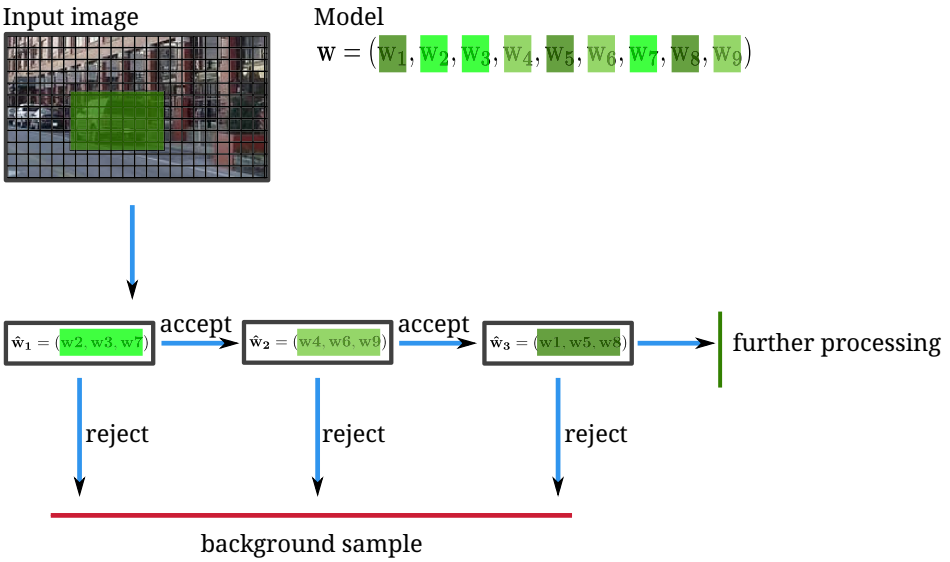
If the score is negative it will be classified as background.

One way to achieve fast multi-class performance is by accelerating single-class detectors. This technique is a more algorithmic approach with the goal to lose as less as possible in detection performance but reduce the detection time per frame. Algorithmic approaches often do not produce the exact solution but are aiming at improving the known systems.

A traditional way of improving speed is to use a coarse-to-fine (Ctf) technique or also called cascade detector. It allows to quickly proceed a considerable amount of hypothesis by exploiting group of attributes level by level instead of treating the complete problem. A Ctf framework handles the input sequentially by transferring it to deeper levels. Each level classifies the input image and transfers it to the next stage



(a)



(b)

Figure 2.6 – (a) An example of a cascade detection system. At the first level the image is scored using a low-resolution model which can be evaluated quickly. Only sufficiently high scoring regions go to the next levels with more precise but time-consuming models. (b) Instead of evaluating the complete model at once and reject samples at the end, another idea is to decompose the computation into pieces. Consequently, a test region is processed by each piece consequently and rejected at each step if its confidence being a foreground object is too low.

or rejects it. Rejected images are immediately classified as background. An illustration of the principle of a Ctf and a cascade detection framework is depicted in Fig. 2.6 A difficult task is decomposed into smaller more tractable sub-problems.

[Viola and Jones, 2001] inspired many researchers to use this approach in order to accelerate various detectors. They adopted a cascade detector where each level consists of a set of weak classifiers. A key concept is which feature attribute should be examined at what level. The work of [Gangaputra and Geman, 2006] formalizes this key issue by measuring the cost of exploring a set of features at one level and its statistical relevance in the final result. This leads to automatically constructed hierarchy of classifiers.

The Ctf method also allowed to accelerate the DPM. The DPM consists of a set of part models. The cascaded version of [Felzenszwalb et al., 2010b] examines sequentially each part filter starting with the root filter. After each filter has been applied, a hypothesis is compared to a learned threshold. Is its score higher, the hypothesis is forwarded to the next stage otherwise it is rejected. The distance transform is also considered as a filter and possible object regions have to "survive" the penalty of the distance transform. Moreover, they use a feature reduced version for each part before applying the more exhaustive part models.

Instead of grouping sets of features into groups and evaluating them one after each other, [Pedersoli et al., 2010] create multi-resolution features. The low resolution features focus more on the global silhouette of the object while the higher resolution features describe the object in more detail. This method leads to a speed-up of up to 12x compared to a baseline detector. An alternative way to explore the concept of multi-resolution cascade is given in [Pedersoli et al., 2011]. The baseline detector is again the DPM model but this time the speed is obtained by varying the image resolution. This is done by recursively searching for best part placements of the DPM filters only on a limited neighborhood mxm instead of the whole image. This makes sense as the parts are usually close to the body. The value of m determines the trade-off between speed and accuracy. They tested their technique on different detectors and can achieve a gain in speed between $10\text{x} - 100\text{x}$. This speed-up is independent of the image content in contrast to the traditional cascades where the speed-up depends on the similarity of the image region to a foreground object.

Another way to look at object localization is by searching for the shortest path to the highest scoring object region as in [Lampert et al., 2009a]. This opens the way to apply search algorithms such as branch and bound. During search, a large region is hierarchically split into smaller subsets along one axis each time. For the real scoring function $f(Y)$ with Y a set of rectangular regions, an approximate upper-bound function $\hat{f}(Y) \geq \max_{y \in Y}(f(y))$ is required. This upper-bound function allows to use best-first search to continue searching for the object in the most promising sub-window. Where sliding window search techniques only find locally maximum responses, this techniques find the globally most promising object hypothesis. The previous approach only achieves interesting speed-ups for linear classifiers. To overcome this issue with non-linear classifiers, [Lampert, 2010] integrate the previous method into a cascade combining search for the best object location and the advantages of a cascade detector. The cascade detector is not applied to every possible location but to regions in the image. These regions are refined recursively at each stage in the cascade resulting in a

divide-and-conquer principle. This combined system yields a speed-up of less than 2x over a cascaded version of a detector.

The methods mentioned earlier reduce the time spent during the convolution of the filters. Another aspect of rapid object detection is the calculation of the feature pyramid that is the features at different scales in order to detect objects of different sizes. The work of [Dollar et al., 2010, Dollár et al., 2014] improve the speed of detecting objects of all scales by re-scaling the features of the learned model and image at the same time. The image is sparsely scaled to all the octaves. Re-scaling the model allows then to detect objects of all the sizes. This technique inspired many future publications to accelerate their detection framework.

We saw before that the speed of cascade detectors strongly depend on the image content. One way to improve scoring the filters with image features independent of the image content is given in the work of [Dubout and Fleuret, 2012]. They apply an old theorem of the theory in signal processing by doing the convolution through the use of a Fourier transform. The cost of a standard convolution between an extracted image feature of size MxN, a model of dimension PxQ and a total of K features is $C_{text\{std\}} = 2KLMNPQ$. However, the cost of calculating the scores through the Fourier transform is $C_{\text{Fourier}} \approx 4KLMN$. This gives a speed-up of $\mathcal{O}(PQ)$ per image and per filter. In other words, the gain in run-time is reduced with higher filter dimensions and number of filters. This makes it especially interesting for multi-class approaches where the amount of filters can get very huge. The speed-up obtained using the DPM framework are up to 8x.

The DPM model applies a small number of parts. On the contrary, the feature synthesis method of [Bar-Hillel et al., 2010] can handle hundreds of parts. This approach is accelerated in [Levi et al., 2013] and achieves real-time performance of 10*fps* which is 4x faster than the cascaded DPM version. The basic idea is that they search for parts efficiently using a KD-Fern which is similar to a KD-tree technique for doing nearest neighbor search.

Another way of accelerating the DPM is shown in [Yan et al., 2014]. The authors made three contributions: (1) Decomposing a 2D filter into several 1D filters, (2) neighborhood aware pruning in the cascade and (3) fast HOG calculation through lookup tables. Decomposing a filter into a linear combination of 1D filters allows for a more efficient hardware implementation. To this end, the filter must have a low rank. This is forced during learning with SVM by minimizing the nuclear norm of the filter. The second contribution is to do more aggressive pruning in each stage of the cascade. Most positive positions arrive at the end of the cascade but are eliminated by the NMS step. The locations can indeed be eliminated much sooner in the cascade chain by suppressing positive hypothesis that are very small in a positive neighborhood. A similar approach can be done for neighborhoods with negative values. If the elements in such a neighborhood are very small, the complete region is classified as background without further processing. The third contribution avoids computing the gradients and orientations necessary to build the HOG features by pre-computing and storing them in a look up table.

All the previously mentioned frameworks report results on average frame rate. [Sadeghi and Forsyth, 2014] guarantee a fixed frame rate. The chosen frame rate allows to trade-off speed and accuracy. With a reasonably small loss in performance,

they do object detection of all 20 categories of Pascal VOC 2007 dataset with a speed of 30Hz. To this end, they use a fast feature pyramid construction similar to [Dollár et al., 2014] by constructing an image and model pyramid. Moreover, they use a vector quantization step of [Sadeghi and Forsyth, 2013] which reduces the dimensionality of the original DPM features using cluster IDs for each cell. As now the cluster IDs are quantized, a lookup table is used to quickly compute an approximate score between the model and the image features. Finally, an object proposal step is integrated which gives a list of the templates with the highest priority to be considered. Their approach has again the advantage to be independent of the considered input image.

Nearly most of these approaches do not improve the performance over their baseline methods. Our developed system allows to ameliorate the performance compared to the traditional OvA method due to its numerous classifiers in the hierarchy. Similar to [Sadeghi and Forsyth, 2014], we are able to trade-off speed and detection performance. This is achieved by our tree traversal algorithm that is fast if it aggressively prunes low scoring regions.

2.2.3 Hardware Accelerated Methods

Compared to the last decade, the price for computing hardware is decreasing while its performance is improving. Already over 40 years ago, Moore [Brock, 2007] stated that the computational capacity is doubling approximately every two years. This statement is known as *Moore's law*. Consequently, the capabilities of electronic devices such as processing capacity or memory capacity gain in performance continuously. Most home users already posses high performance computers with many cores, big storage and graphics processing capacities. Moreover, the hardware architecture of these devices decreases in size facilitating its integration into mobile devices. This motivated the researchers to reduce computational time for object detection as the many computer vision algorithms adapt well to hardware parallelization. Computer vision deals with processing images. Some steps can be optimized by not treating the input sequentially but simultaneously. This holds true for the domain of object detection. We can explore the quick processing using the central processing unit (CPU) or the graphics processing unit (GPU). The GPU consists of thousands of cores while the CPU only possesses a multitude of cores. However, each core on the GPU is slower than those on the CPU side. A known issue with using GPU is also the slow data transfer between GPU and CPU. Both allow to split a time consuming task into multiple smaller tasks which are run in parallel. Afterward, the pieces of results are collected and form the final result. In the following, we will have a look at works that develop code on GPU and CPU often with the aim of not deteriorating the detection performance compared to their baseline CPU implementations.

Currently, the CPU allows more importantly to run multiple resource consuming operations in parallel. The gain in speed depends on the underlying hardware architecture such as number of cores or clock rate. The work of [Cho et al., 2012] optimize the code of the DPM star-cascade [Felzenszwalb et al., 2010b] by multi-threading it. First, they run the original publicly available code on several machines and record the running times for the different parts in the code. The original code is a mixture of MATLAB and C languages. Most of the computation time is spent on extracting the

features for the dense feature pyramid. A further bottleneck is the classification phase where the model scores every candidate window in the feature pyramid. The `pthread` library was used to calculate the feature pyramid in parallel as the processes of extracting features on various levels are independent. The same holds true in the classification phase when each part filter of the DPM model is applied regardless of the other part models. The responses of each thread are summed in the final step to produce the appearance score. They achieve a speed up of a magnitude for the feature extraction phase and a gain of 2x-4x in second phase on a Intel Core i7 2920XM@2.5GHz without loss of performance.

Another concern on using hardware is the balance between energy consumption and the speed enhancement. [Totoni et al., 2013] evaluated the impact of the interaction between detection precision, power and energy consumption and the gain in run-time. The experiments were done using CPU and GPU implementations for face detection. They created 3 different set-ups:

- specialized: Each architecture type processes one task in the detection pipeline and the next frame is only processed once GPU and CPU are idle.
- overlap: Same as specialized with the difference that the next frame is processed as soon as one architecture type is idle.
- split: The image is split in half and treated by the CPU and GPU independent on each other.

The part of the pipeline running on each architecture is chosen according to the utility of the architecture. The experiments showed that the CPU code has higher run-times than the GPU implementation but this comes with the price of a higher energy consumption for the GPU. The best results are achieved for the *overlap* case which is at the same time fast and energy friendly.

The first to port the HOG detector [Dalal and Triggs, 2005] onto GPU were [Wojek et al., 2008]. The authors [Prisacariu and Reid, 2009] exploited the parallel technique too with a more efficient implementation. The GPU port runs $67x - 95x$ faster than a naive CPU implementation. [Sudowe and Leibe, 2011] goes one step further and introduces scene geometry constraints. The detector does not need to search anymore a dense feature pyramid but scans only pedestrians for various sizes on the ground plane.

The more advanced DPM model is implemented by [Gadeski et al., 2014] where I co-authored this work with another research group. I helped to parallelize the original DPM framework using CUDA. We had to find clever solutions to simultaneously multi-thread the code and keep it highly flexible to the input model and image. The challenges using such a model are an increased number of filters and a huge feature pyramid facing space limitations on the GPU side. On standard GPU, we show that one can achieve a speed-up of up to 11x compared to the sequential C++ implementation of the DPM. Compared to a multithreaded C++ version (8 threads), the gain is 5x. The bottleneck is the classification phase and the data transfer between the CPU and GPU. Fig. 2.7 illustrates the workflow of our system. Similarly as before, the authors [Pedersoli et al., 2013] further apply a coarse scanning of the scene by using scene geometry information. The application context is pedestrian detection on roads. Inspired by

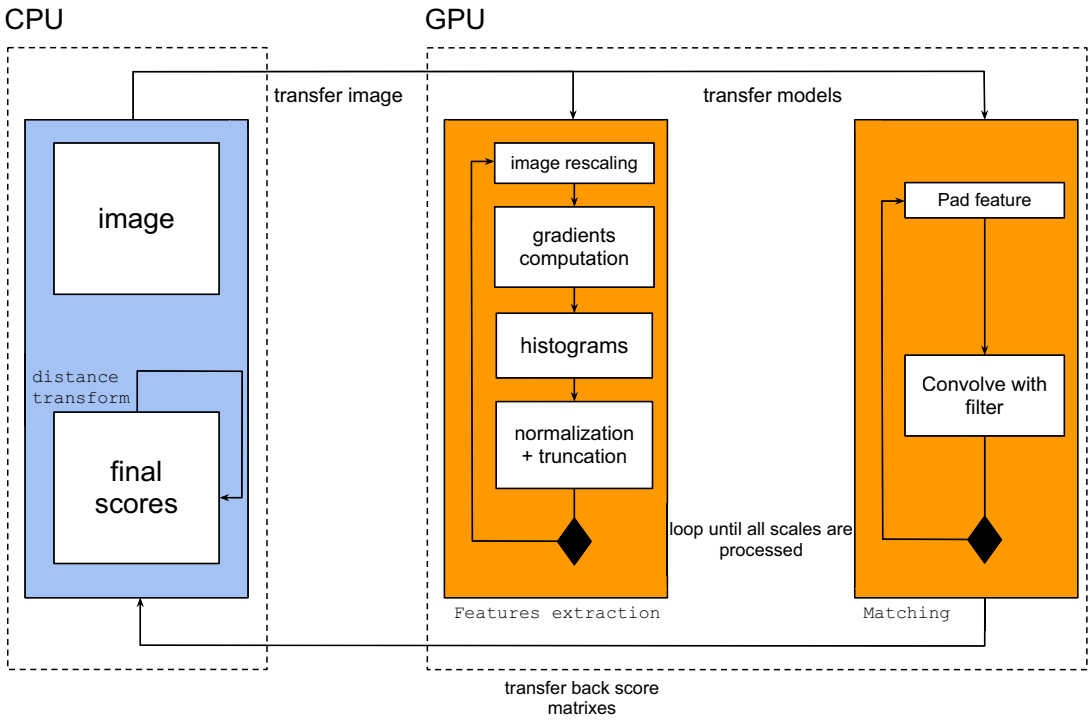


Figure 2.7 – An overview of combining two architecture technologies namely CPU and GPU for object detection. The feature extraction and matching is done on the GPU while the final score is calculated on the CPU (Courtesy of [Gadeski et al., 2014]).

[Felzenszwalb et al., 2010b], they do a coarse-to-fine search. In order to better detect small pedestrians, they neglect resizing the input into bigger images and use a binary feature to normalize the scores between their two models of higher and lower resolution pedestrians. The total speed-up is one order of magnitude.

While the two previous works concentrated on remaining as close as possible to the original DPM code, [Song et al., 2012] represent a part in the DPM model as a combination of dictionary elements. The dictionary is built from the training images of all input classes and are shared among the classes. The filter of one part is the sparse combination of dictionary elements. The speedup factor is given by:

$$\frac{lh^2}{\mathbb{E}\left[\|\alpha_i\|_0\right]}, \tag{2.3}$$

with l the feature dimension, h the part dimension and α_i the sparse vector that activates each dictionary element for a part filter i. The dictionary does very slowly increase in size in the number of classes and makes this approach especially interesting for efficient multi-class object detection. Instead of computing the filter responses with a linearly increasing number of filters in the number of classes, only the dictionary elements need to be convolved with the feature pyramid. The responses of each element are independent of each other. As a consequence, this algorithm is implemented on the GPU. This leads to an algorithm that can trade-off speed and detection performance depending on the size of the dictionary and the sparsity of the reconstruction.

The system of [Benenson et al., 2012] achieves 135 fps when exploiting massively parallel programming. Their baseline detector leans on the work of [Dollar et al., 2009] who use rectangular features with decision trees combined with Adaboost. One of their contributions is to calculate depth information without calculating actually a depth map [Benenson et al., 2011]. This allows to sparsely scan the image for only potential pedestrian regions. Next, traditional systems scale the image several times to obtain the feature pyramid. Their work does no input image resizing. To recognize pedestrians on multiple scales, they learn a model per octave and scale the models. The scaling of all models can be done before starting the detection phase. Executing their system on monocular images already yields a frame-rate of 50fps.

2.3 Multi-class Object Detection

The aim of a multi-class object detector is to localize the instance of various object categories in an image. We will review previous works in the literature in the next section. There is a parallel line of research which consists in making these frameworks more robust and scalable. Sec. 2.3.2 focuses on augmenting the accuracy of a multi-class detector by exploiting additional knowledge such as the coherency of class occurrences in a scene instead of only classifying based on appearance. An interesting property of multi-class learning is the ability to quickly learn a new class. This is helpful is the training algorithm is bound to time constraints or lacks of training data as further specified in Sec. 2.3.3.

2.3.1 Frameworks

This section revisits the most relevant literature subject to this work. The objective of a multi-class framework is to balance speed and accuracy during detection. The traditional idea to improve accuracy is by sharing attributes between classes. It follows that the classes are no longer seen as separate entities but take advantage of the presence of the other classes. Knowledge can be shared on multiple levels in the system. Speed of detection is improved as the scoring of a region in the image is no longer computed separately for each object category. The information gathered during the detection process is shared by all related classes. We structure the state-of-the-art in three categories as illustrated in Fig. 2.8 based on their shared attributes: low-level features, parts and part locations.

Boosting based methods

Features belong to the category of low-level attributes. A feature descriptor can usually be displayed as a high-dimensional vector. It is a key component in the success of an object detector as these describe the objects in the image regions. Intuitively, the description of many classes has similarities and thus can be shared among these classes. The approach depends strongly on the chosen learning algorithm. The work in [Torralba et al., 2007] designated as *JointBoost* were among the first to apply the principle of feature sharing in the domain of object detection. The learning framework is a generalization of the *gentleboost* algorithm [Friedman et al., 2000] which is a variant

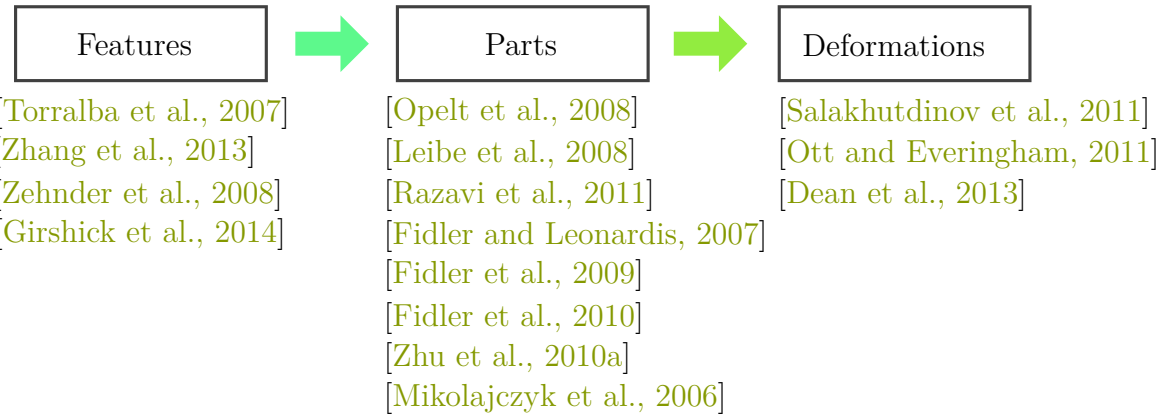


Figure 2.8 – Most state-of-the-art approaches can be grouped depending on the degree of information shared between the target object classes.

of the boosting principle. During training, boosting selects a *weak classifier*. A weak classifier takes an input vector and attributes a label to it. The sum of all these weak classifiers build the final decision function called a *strong classifier*. In the generalized method, each weak decision influences the scores of a subset of classes. This reduces the total number of weak classifiers over all classes. The objective of the training algorithm is to find these weak classifiers that reduce the total multi-class error and not the errors of each class independently. The reduction of the weak classifiers improves the training time and run-time. The latter scales logarithmically to the number of classes. In addition, the performance increases over a OvA method due to the joint learning framework.

[Opelt et al., 2008] share part appearance information and location of that part among classes. A part is described with its contour. Each part votes for an object centroid. The voting space is different for each class. The concentration of votes allows to detect objects in images. During training, a set of words, representing parts, are extracted. The words which match and localize best among positive images and across categories build the final dictionary. In the next step, a boosting framework inspired by [Torralba et al., 2007] is used to select the best parts as weak classifiers. The weights attributed by the boosting algorithm define the voting strength of that part for an object centroid. This joint framework permits to have a sub-linear growth of the dictionary size. Also the performance is increased over learning the classifiers independently.

The JointBoost is also the backbone of [Zhang et al., 2013]. It includes intra-class and inter-class models. Each class is divided into sub-classes with low variation. During training, the sub-classes of a class are not discriminated among each other as the task consists in detecting a class and not the associated sub-class. However, a soft-cascade accelerates the run-time of the detector. A downside of this method is that the run-time and training time increase quickly as each class is divided into sub-categories.

DPM based methods

Upon the success of the DPM, [Ott and Everingham, 2011] enhance this baseline through part sharing. In the original DPM formulation, each mixture component

of each class contains parts which are mutually exclusive to other mixture component and classes. The score of a component is the sum of its own part filters. For the aim of multi-class object detection, it is reasonable to share all the parts between every component and class. This issue is addressed by letting the score of a component be the weighted sum of all part responses. Learning is done exactly as before only that in an inner loop, these combination weights are learned using a SVM problem formulation by fixing all the remaining parameters. The authors conclude that sharing parts among components and classes improves mAP on PASCAL VOC 2010. The paper does not emphasize on whether the multi-class model is scalable to 20 classes as tests were done only with 2 classes.

[Salakhutdinov et al., 2011] tackle the problem of unbalanced training examples in datasets for multi-class object detection. They show that by hierarchically grouping classes into a tree structure, the average detection performance increases. Each node in the tree represents a filter and the total score of a class is given by the sum of the scored filters along its corresponding path. Each filter in the tree is learned independently of the other filters by fixing the other filters. After each iteration, the tree structure is inferred again. A class is assigned to a node in the tree based on the resulting classification performance on the training set but also a CRP prior [Blei et al., 2010] which favors specific tree structures. The training protocol alternates between learning the filters in the tree and optimizing the placement of the classes in the tree. This technique comes with the drawback of increased training time and the increased run-time over OvA as many more nodes are now evaluated. However, the detection performance benefits of the tree structure. The performance of the classes with very few examples but also dominant ones are improved. This work strongly influenced the concepts of this thesis. We differ our work on how we construct our tree and optimized the filters. Our work trains all the filters in a joint problem formulation and trade-offs speed and accuracy of detection. Moreover, our detection goal formulation allows to use a tree traversal algorithm to speed-up the inference time.

Hough transform based methods

The implicit shape model [Leibe et al., 2008] is a detection framework where part appearance and location influence the object hypothesis. During training a dictionary of parts is constructed where each word knows the location to the object's center. During detection, a set of patches from the image are mapped to the dictionary entries which on their turn vote for a hypothesis center. The hypothesis is labeled as positive if the concentration of the votes is high. To improve and fasten the matching from image region to dictionary entries, [Gall and Lempitsky, 2009] proposed to use random forests. The number of leaves in the random forests influence the run-time. The extension to multi-class in [Razavi et al., 2011] has a sub-linear growth in k. The number of votes during detection follows the same behavior. Even though many classes have appearance entries in common, the discrimination among them is made possible through the additional location information. A taxonomy of classes can be used to even more restrain the number of votes as for every feature in the image only a subset of classes vote for a class label. Their system rigid to the star model which is nowadays outperformed. Also, they did not focus their work on the benefits of hierarchical knowledge which is a main subject of this thesis.

In the continuity of the ISM baseline, another example of a generative model is shown in [Polak and Shashua, 2010]. Each class is trained and detected independently. They combine bottom-up and top-down inferences. The former associates the extracted interest points in an image to object parts. Using a Naive-Bayes assumption, this stage infers a preliminary classification of the objects in the image to classes. In the latter stage, the most promising classes only are evaluated by finding best part locations which vote for object centers. The results show that it works reasonably compared to ISM but is still inferior to DPM. Furthermore, no evaluation was done on the scalability.

Compositional models

Compositional models describe objects in a hierarchical way. A part is recursively defined as a composition of parts. The parts belong to a layer and are built based on parts of lower layers. At the top layers of the tree, the nodes represent the totality of the object while towards the leaf nodes, parts of the object are found. The bottom layer consists of basic features that can be seen as words upon which complicated sentences, here objects, are constructed. This principle naturally allows to include part sharing. [Fidler and Leonardis, 2007] describe such a compositional model in a generative approach. A part of the new layer is chosen as to minimize the number of firings of the part in the image, reduce computation for matching that part and it should cover a maximum of points of the object. The framework is evaluated for the multi-class application in [Fidler et al., 2009] when learning the part hierarchies is done jointly, independently or sequentially. Independent learning is the OvA principle and joint learning is building the constellation model for all classes simultaneously. Sequential learning consists of learning the k-th class using the already trained model for $(k-1)$ classes. Sequential and independent learning perform similarly well on detection rate while the overhead of the sequential training is small compared to OvA and scales much better than joint learning. For this detection model, learning sequentially the classes is the optimal choice. It further gives the possibility to add one class after another as the added classes re-use the current codebook. A similar goal is set in [Lim et al., 2011b] which is based on exemplar object detection. Compositional methods are mostly generative models. In our thesis, the focus lies on discriminative techniques which are faster and dominate the best results on recent bechmarks.

Another example of compositional models is given in [Zhu et al., 2010a] called recursive compositional models (RCM). Contrary to the previous approach, not only contour information but further appearance cues such as color information are used. The leaf nodes contain basic oriented segments upon which object specific contours are constructed. The parts are connected by spatial pairwise relationship. Again the levels are built from bottom-to-top. The lowest level representing boundary edges at quantized orientation, the level on top of it picks triplet of the lower level parts, creates prototype triplet clusters and prunes those which do not fit spatially the object class or have a strong overlap. Fig. 2.9 shows several layers of RCM that build 26 classes. The composition of the bottom layer words allows to construct more complex structures shared by all classes. By repeating this process, the algorithm is able to determine automatically the numbers of classes and viewpoints. Compositional methods are nice in formulation but usually long in run-time. [Mikolajczyk et al., 2006] is another example of a generative model. It is similar in its basics to the star model object

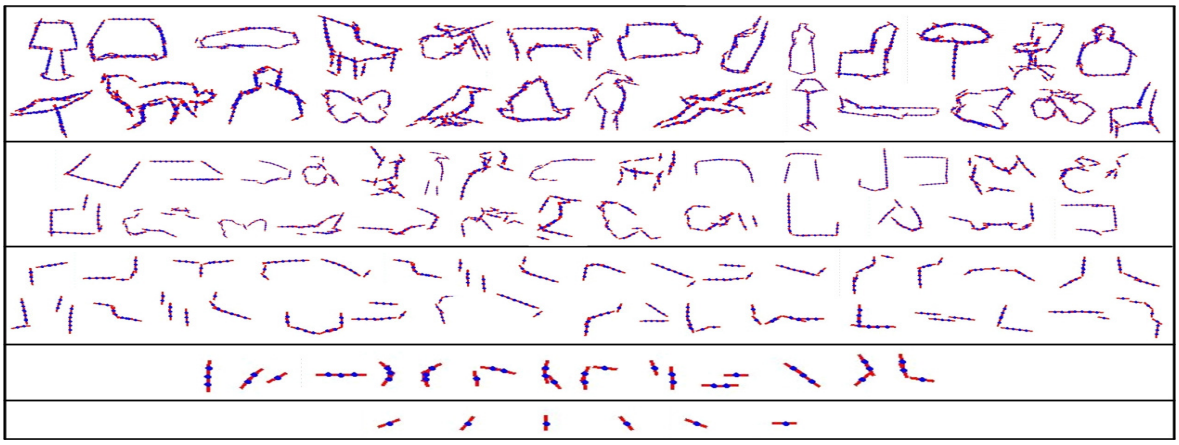


Figure 2.9 – High level object classes can be recursively decomposed into parts. The elements constructing these parts are oriented edges. Objects are represented by a common codebook which grows sub-linear with the number of classes (Courtesy of [Zhu et al., 2010a]).

detector of [Leibe et al., 2008]. A dictionary of features which are shared among object classes vote for an object center. These features are represented in polar coordinates relative to the object center using dominant axes information of the part [Mikolajczyk et al., 2003]. These features are hierarchically clustered based on their location to the center. During detection, the features of the image are clustered similarly and the tree model is matched to the hierarchical structure of the image. Similar to [Razavi et al., 2011], this technique has the advantage of growing the codebook sub-linearly.

Cascade based methods

A natural way of handling multi-class detection is by exporting cascade detectors. It allows to continuously narrow the hypothesis space of classes. In [Fidler et al., 2010] during detection, the tree is traversed from top-to-bottom very similarly to [Amit et al., 2004]. If during the search a node is evaluated to 1, its child nodes are evaluated otherwise the underlying paths are pruned. The algorithm proceeds in a depth-first manner. The intermediate nodes represent coarse models which can be evaluated quickly. Only the likelihood obtained at the leaf node vote for the object's presence. A disadvantage of this search is the hard binary decision taken at the node level instead of creating a table of priority paths. Another drawback is the fact that the final decision function does not use calculated information. Both points are handled in our contributions. The chosen detector for each class is [Fidler and Leonardis, 2007]. The results are relative to this baseline. They report a slightly worse performance for a modest speed-up of around 2x.

[Zehnder et al., 2008] evaluate the benefits of shared stages of a cascade over multiple separate cascade detectors. The different stages of their multi-class cascade are the result of merged stages of a separate cascade. This implies that one stage which can be seen as a node is shared by several classes. Each node is trained using gentleboost. They implement iterative and cascade construction of the cascade. The iterative method constructs the multi-class cascade step by step by recursively measuring and clustering

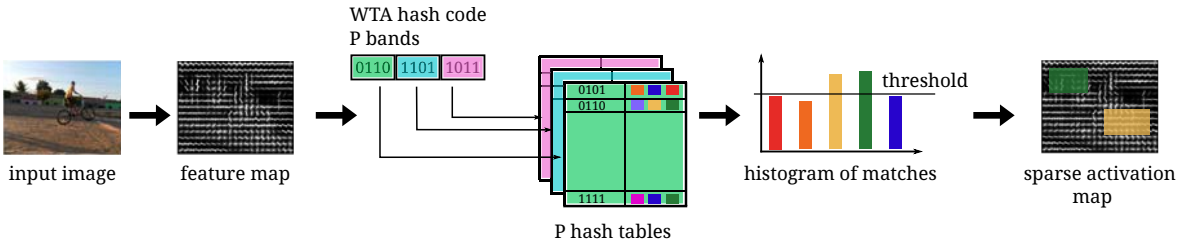


Figure 2.10 – Hashing technique presented in [Dean et al., 2013] makes scalable object detection possible. The detection framework is of [Felzenszwalb et al., 2010a] where the convolution step is heavily accelerated. The filters are no longer evaluated by convolution with the feature map. A WTA hash key of the features allows to vote for parts and object class. Finally, only the most probable part filters are evaluated.

the similarity between the stages of the current cascade. The flash construction method builds first separate cascades for the classes and then merges similar stages together. Both approaches result in a comparable performance. The performance and run-time gain improve with increasing number of classes.

The work of [Lampert and Blaschko, 2008] optimizes the individual models for each class in a joint training framework. This discriminative technique learns simultaneously the relation between object classes. There exists model parameters between every combination of the classes. The decision function for one class depends on the weighted sum of the partial scores attributed by other classes. These weights and model parameters are learned using a multiple kernel learning approach. The visualization of the weights shows that the authors are able to find semantically meaningful relationships between the categories.

Highly scalable methods

Recently two major frameworks achieved impressive results regarding the scalability in the number of classes k. Both methods allow to increase the detector to thousands of categories. The first one that we will mention is very rapid in the execution time with slight performance drops while the second one scales less suitable in k but has stable accuracies.

Accelerating convolution through hashing

One of the most promising steps towards scalable detection of object classes is replacing the heavy convolution for scoring the models and image features in the feature pyramid. In the DPM framework, most of the execution time is spent when computing convolutions of the C model filters with the L locations in the feature pyramid. The complexity is therefore $\mathcal{O}(LC)$. The work of [Dean et al., 2013] investigate the use of hash tables to reduce this complexity to $\mathcal{O}(L)$. In other words, the run-time does not increase with the number of classes respectively filters. The key insight is the transformation of the features into an ordinal space which results into a high-dimensional sparse feature vector. In this space, the similarity function measures the differences between filters making it robust to filter perturbations. The first steps consists in flattening the HOG features in a window or the model. Then, the elements are re-arranged

in a random order giving one new sequence. This procedure is repeated as to obtain N sequences. The first K indices of each sequence are retained. In every sequence the maximum value is replaced by 1 and all the remaining values by 0. The concatenation of all resulting values is referred to as WTA and represents a sparse code of length NK. This new vector is further split into equal-sized parts of length WK with W a parameter. This gives a total of M bands and there exists a different hash table for each band. An entry in a hash table votes for parts. The histogram of these votes gives an ordered list of the most probable filters.

During detection, the WTA hash for each HOG window is computed. Every band accesses the entry in it's respective hash table. The votes are used to identify the most promising part filters for which the final dot products is computed. The algorithm proceeds as usual with the next module in the framework. Besides building the hash tables, there is no modification in the training protocol of the baseline. The approach is depicted in Fig. 2.10.

One drawback is that all filters have to have the same size. No root filter can be used. This results in a negligible performance drop. The technique allows to balance between accuracy and speed by varying K and N. The run-time gain is 20x for 20 classes and at a reasonable performance and four orders of magnitude for $100,000$ classes.

The code is executed on a single machine with at least 20GB of RAM. This acceleration technique can be combined with other methods *e.g.* that reduce the time to construct a feature pyramid or those that reduce the run-time complexity by reducing the locations L to be evaluated.

Convolutional neural networks for efficient feature extraction

Neural networks and more precisely convolutional neural networks (CNN) were often applied and studied in the 90s. In the last 3 years, and especially based on the results of [Krizhevsky et al., 2012] in the field of image classification, the research has again focused on CNNs. To our best knowledge, the best promising work applying CNNs is the work of [Girshick et al., 2014]. We mention their work in this section as their approach has very little overhead to detect many classes. The last few years improved slightly object detection performance on public datasets *e.g.* PASCAL VOC by augmenting the complexity of baseline detectors. However, their work takes a different direction by exploiting features extracted using a CNN. The CNN which takes most of the computation time is shared among all classes.

The detection pipeline is shown in Fig. 2.11. In short, the proposed framework takes the input image, extracts possible regions with objects, computes the CNN features for each of these regions and finally classifies these features using a OvA SVM technique. Instead of exhaustively extracting CNN features for all locations in the image, the first step consists in finding promising regions with objects. This is achieved using selective search [Uijlings et al., 2013]. The image is searched for regions containing the wanted objects but for each selected region we do not know the object class label. Usually this output is forwarded to a multi-class object detector which decides for the class label.

The selected regions are arbitrary shaped. With simple affine image warping, the regions are warped to a standard size and forward propagated through the CNN. The CNN has 5 convolutional layers and 2 fully connected layers which produce for each

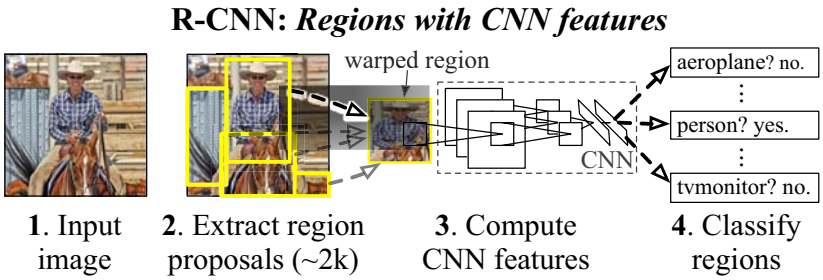


Figure 2.11 – Illustration of the R-CNN detection framework of [Girshick et al., 2014]. Some promising regions are warped to predefined dimensions and the R-CNN allows to compute features for each such region. The new features are then simply forwarded to a OvA linear SVM classifier (Courtesy of [Girshick et al., 2014]).

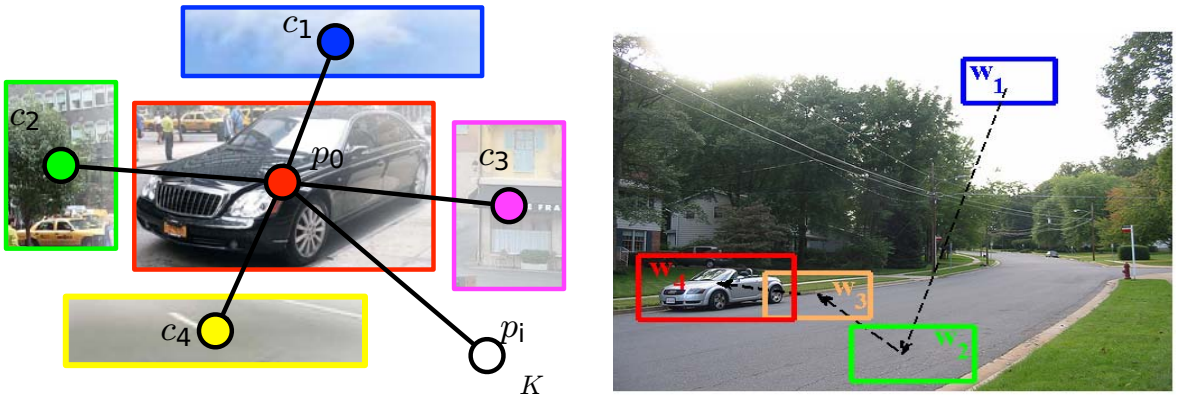
proposed region a feature vector of dimension 4096. This feature vector is class independent. The final step consists in classifying the feature vector into one the possible k classes or the background regions. To this end, the feature vector is the input to a OvA SVM classifier. Each SVM classifier stands for an object class and assigns a score to the feature vector. The class label is defined by the highest scoring class. Finally, the same non-maximum suppression as in the DPM code is applied but for each class separately. The technique does not rely on a feature pyramid nor an exhaustive search method.

During the training phase, they pre-train on a large datasets ILSVRC2012 without bounding box annotations and then fine-tune to the detection task using stochastic gradient descent. The method has shown to be a good choice for learning new features. The run-time is dependent on the number of classes but the associated overhead is very small. The run-time of the CNN for a typical image of the PASCAL VOC dataset is 13s on the GPU and 53s on the CPU. The classification with the OvA SVM classifiers is negligible for small number of classes *e.g.* < 200. For $k = 10$k the estimated time is less than $10s$. One advantage of this framework called R-CNN compared to the hashing method of [Dean et al., 2013] is the strong high detection performance. On the PASCAL VOC 2010 dataset it achieves a mAP of 50.2 compared to 33.4 for the DPM framework. A similar performance boost is observed on the PASCAL VOC 2007 dataset with a mAP of 58.5 for the R-CNN method and 33.7 for the DPM. The R-CNN network transforms an image region to a feature representation. Our network is independent of the region descriptor and the integration of the R-CNN features is left for future work.

Many of these techniques are strongly dependent on the detection procedure and features. They have little flexibility towards the integration of new object descriptors like the compositional models. Our work overcomes these limitations and remains highly flexible to be combined with new ideas.

2.3.2 Exploiting Contextual Knowledge

The role of an object detector is to estimate as accurately as possible the label of an object respectively region. A parallel axis of research is context-based object recognition. Objects in a scene appear in a logical configuration that is the structure of a



(a) (b)

Figure 2.12 – (a) Objects of the same class appear often in a similar environment. The car object class is modeled using the DPM where surrounding patches are added into the model as additional parts (Courtesy of [Mottaghi et al., 2014]). (b) Context information helps to guide the focus of attention. The detector scans the next region based on the past ones. This enables to avoid looking everywhere for objects as in the traditional scanning window technique (Courtesy of [Alexe et al., 2012]).

scene follows configuration rules. These rules determine the interaction between objects, their relative location, sizes and their poses in the image. It has been shown that context is a rich source of information to quickly and reliably determine the nature of an object [Oliva and Torralba, 2007]. A TV set is more likely to appear in front of a sofa constituting a living room than besides an elephant characterizing object pair co-occurrence. Usually a plate appears on a table than below it implying the spatial relationship between objects. It is shown in cognitive science [I Biederman, 1982] that the global structural properties in a scene impact the identification of the object itself.

The multi-class object detection chain can be extended to incorporate such a module. This helps to rectify the outcome of object detectors which can be semantically incorrect. Due to its importance in object detection, we will review in the following some of the works achieved in this domain.

Most approaches in the literature use the contextual information for the task of object priming. It consists in improving object categorization. In the beginning, the rules implied by the scene context were hard coded a priory rules [Hanson and Riseman, 1978, Strat and Fischler, 1991]. This field gained attention in computer vision research through the work in [Torralba, 2003, Torralba and Sinha, 2001]. They model context information for object detection in a probabilistic framework. The objective is to use context features to get priors on class, size and location of objects.

[Desai et al., 2011] focus on the spatial arrangements of objects by learning a weight vector for object appearance and another weight vector for valid geometrical configurations. The relative spatial positioning of the objects are quantized into seven bins. The weight vectors are formulated as a ranking problem where the most likely configuration ranks highest. The results outperform the baseline detection performance. This method learns its own appearance model. The work of [Choi et al., 2012] uses an off-the-shelf object detector. The context information includes a learned spatial location prior between object categories and not the instances, co-occurrence statistics

of object pairs and a global image descriptor gist [Torralba, 2003] that gives information about the scene type. The algorithm learns a tree where each node represents a class and the edge weights indicate the strength of the link between these classes. The sources of context are combined in a probabilistic framework to compute correctness of detections using window locations and scores. This result is again combined with scene information to estimate the object's presence. A crucial context information is the scene type provided by the gist descriptor. The performance does not outperform the DPM on small datasets but works better on datasets with many classes *e.g.* SUN09.

[Peralta et al., 2012] extend this work by conditioning the previous probabilistic model on the scene type. The reason can be explained intuitively: the co-occurrence between object pairs often depends on the scene type. Consequently, the contextual relations depend on the underlying scene. This is modeled by a supplementary latent variable. [Choi et al., 2012] model a single tree while this work uses several trees in a mixture model based on the number of different context types. [Izadinia et al., 2014] also exploit context-specific appearance information by posing object detection as a scene structure discovery problem. The scene layout is modeled through mixture components very similar to the DPM where the appearances of the objects and their relative locations of the objects form a component. [Mottaghi et al., 2014] follow a similar idea and model an object as in the DPM but include a different type of parts where these additional parts, called contextual parts, represent neighboring regions as shown in Fig. 2.13a.

The sources of context are manifold. In [Divvala et al., 2009] various sources are collected to reliably estimate the object's presence. The authors make use of the scene gist and geometrical context estimation from an image to classify the scene. This knowledge is augmented by geographic properties by traversing a separate annotated database and semantic context helping to predict object occurrence. Using the training images, they learn statistics about most probable object location in an image. The object size cue is obtained by estimating the object's depth from the image. All these cues reduce the confusion with background and get more accurate localizations. The context helps most objects having impoverished appearances. The authors of the DPM [Felzenszwalb et al., 2010a] suggested themselves the use of a simple but even until today very efficient context module. The context information is given by a feature vector and aims at rescoring the detections. The feature vector uses global and local knowledge $(\sigma(s), x_1, y_1, x_2, y_2, \mathrm{C}(I))$ where $\sigma(s)$ is the normalized score, (x_1, y_1, x_2, y_2) are the detected bounding box characteristics and $\mathrm{C}(I)$ is the image context grouping the highest scores of each class.

As we have seen so far, most works integrate topological information between objects that can also be exploited in video *e.g.* [Paletta and Greindl, 2003] and not only in still images as the ordering between objects keep a certain temporal logic. On the other hand, [Bachmann and Balthasar, 2008] focus on modeling the interaction of object and scene. They mostly aim to detect cars, bikes and pedestrians on the roads. Three scene categories are defined based on the *opennes* of the space. The context feature is obtained by computing the steerable pyramid [Simoncelli and Freeman, 1995] which emphasizes on the orientation in the image. Differently, the face detector of [Bergboer et al., 2006] rather uses local context features around the image region to help object detection as the background region around an object can be a strong indicator of the

centered object class. This type of context information helped reach in the work of [Zhu et al., 2015] top object detection performance on the PASCAL VOC 2010 dataset. Their method relies on the R-CNN framework [Girshick et al., 2014] which trains a convolutional neural network of object regions obtained by object proposals. They extend this approach among others by increasing the cropped region around the bounding box as background information is a good indicator for the possible objects it may contain.

Scene information is a strong cue for the possible object categories present in the image. The reverse is also true: knowledge from object detectors improves the task of scene classification. An approach that combines both is presented in [Song et al., 2011] where object detectors and image classifiers mutually influence the final decision. The context of one task dynamically modifies the decision hyperplane of the other task. First, a basic hyperplane w_0 for each task is learned. The new hyperplane using context features x_c of the other task is given by: $w = \mathrm{P}x_c + w_0$. The projection matrix P is learned. To reduce the number of parameters, P is constrained by a low rank matrix. The feature vector from classification is a vector where each entry is the output score for each image type. Similarly, the context vector constituted from the object detection is the highest score for each object class obtained in the entire image.

A similar approach in [Espinace et al., 2013] aims at ameliorating the scene classification technique by incrementally adding information of one object class. This next object class is chosen as to increase the mutual information regarding the scene probability distribution. Context can also help to overcome the impoverished appearance related to occlusion. [Ouyang and Wang, 2013] exploit the interaction between 2 persons to better locate pedestrians. They learn a model similar to DPM for 2 persons side-by-side. Each mixture component is obtained by a clustering *e.g.* on the common aspect ratio. The 2-pedestrian model contains root filter, part filters but also the location of the 2 persons modeled as parts. The idea of modeling 2 persons was also exploited in [Tang et al., 2012] with great performance improvements. [Song et al., 2011] rely again on the spatial arrangements between people in a scene. The score labels are assigned by maximizing the score of appearance model but also the relationship between the persons in the image. As before, the DPM is used with mixture components found by clustering features using K-means. [Pepik et al., 2013] also extend the DPM by augmenting the DPM model with additional components handling pairs of persons. To get the different mixture components, they cluster the annotations based on their occlusion properties *e.g.* the position of occluder and occludee. Furthermore, they experiment with a joint root model with the individual objects as parts consisting themselves as parts or the occluder containing the occludee as a part. Their first proposed solution improves precision compared to DPM and is especially powerful when objects suffer from strong occlusion. Again, one drawback with these methods is that they are very specific to the detection model instead of using the detector as a context cue.

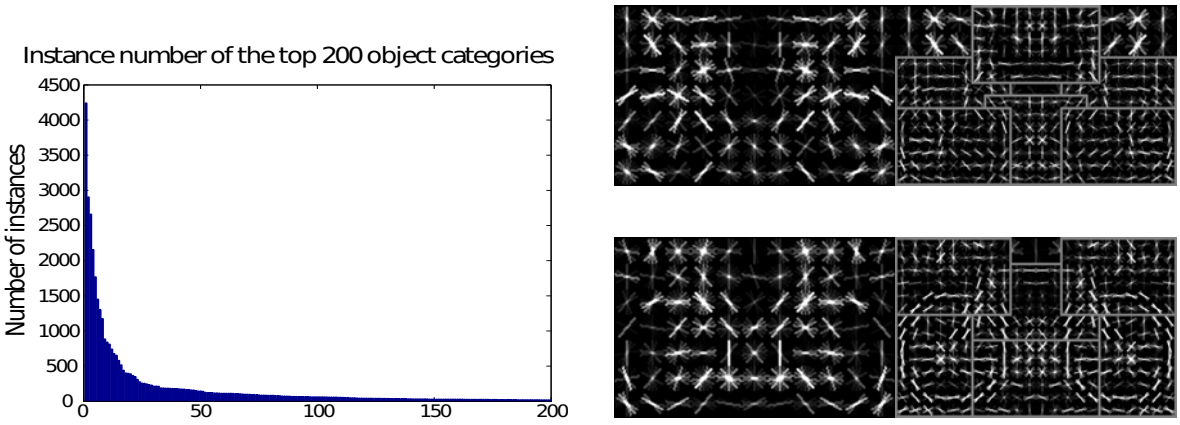
Another useful application consists in guiding the *focus of attention* of the detector depending on the context information. This comes with two advantages: (1) there is no more need to evaluate all the detectors on all positions and (2) by evaluating only a sub-class of classifiers on meaningful regions, we reduce the number of false positives. Using a heat map of likelihood for object locations and scales [Perko and Leonardis, 2007] gains a speed-up of about 7x for keeping a reasonable performance.

Three kinds of sources help to guide the attention namely the geometric context, the image texture and the viewpoint by estimating the horizon. The regions in the image are good indications of possible object locations. [Ristin et al., 2013] extract patches from the training data. At test time, they extract random patches which are matched to the learned one and vote for object locations. The probability map is dependent on the average of the sum of the individual probability estimates. This results in slight performance drop and achieves a speed-up of at least an order of magnitude. Instead of extracting patches in beforehand, [Alexe et al., 2012] search a database of images for similar windows which again vote for possible object locations in the test image. Fig. 2.13b illustrates the search for the next object region dependent on the observed region. Their method not only is able to improve detection performance upon the DPM baseline but is two orders of magnitude faster than the sliding window search.

2.3.3 Scalable Object Detection Through Transfer Learning

Imagine you show to a child the image of a falcon. Even though he might not have seen this species of bird in real, the child might be able to transfer his previous knowledge of similar species to this unknown bird. In computer vision one speaks of the concept of *transfer learning*. It consists of modifying past experiences to generalize to new categories. This concept finds application in object detection and classification. It is of ever growing demand as the datasets increase over time in the number of categories and training samples. As an example, PASCAL VOC2007 consists of 9,963 images and 20 categories where as the now more popular ILSVRC has more than 10 million images and at least two orders of magnitude more classes. This opens doors to scalable object detection and classification where often a reasonable amount of class annotations is not always given. Future algorithms need to handle ideally small amount of annotations for each class and a huge amount of classes. This scarcity of annotations per class makes transfer learning especially important. Intuitively, larger datasets are more suitable for this task due to presence of more similar examples and classes and more generally patterns *e.g.* the most semantic similar class to "car" in PASCAL VOC is bus as where in ILSVRC one can find many more samples such as the "truck" and "van" classes. Moreover, the number of samples per category are not equally distributed and can be approximated by a distribution similar to the Zipf's law [Salakhutdinov et al., 2011] as shown in Fig. 2.14a accentuating the relevance to gain information from other knowledge sources. In the following, we will assume that we have k source classes which can be efficiently learned and k' target classes which lack of sufficient data. Negative transfer occurs if the transfer deteriorates the detection performance of source classes or/and target classes.

[Luo et al., 2011] explore how k source classes help to improve the additional k' target classes. They assume that for the target classes, training data is scarce but available. The technique trains a detection model for all the $k+k'$ classes. The detection models for the target classes remain weak due to the lack of training samples. Therefore during detection, the final decision depends on the sum of its class specific detection score and the weighted combination of the source classes' scores. These weights are optimized in a *multiple kernel learning* [Gönen and Alpaydın, 2011] formulation forcing a class sparsity between the weights.



(a) (b)

Figure 2.13 – (a) Distribution of the number of examples per category in Labelme dataset. Most categories lack of a sufficient amount of annotations (Courtesy of [Wang et al., 2010]). (b) [Deselaers et al., 2010] train an object category by first annotating the training examples. The top model is learned using their approach while the bottom model used the ground truth annotations (Courtesy of [Deselaers et al., 2010]).

[Lim et al., 2011a] do not only consider classes as separate entities but take advantage of the complete labeled training data. Each training example of all the classes can contribute to the target class based on a weight factor. Further, the example of other classes are warped, *e.g.* scaled, to increase its similarity with the target class. The training example weights are learned together with the individual appearance filters for the classes. The weights are regularized with a sparsity norm to prevent noise from other unrelated classes and samples.

Another way of incorporating the source examples in the learning process of the target class is to not only discriminate the objects of the target class but force the target model to rank similar samples of other categories higher than dissimilar ones [Wang et al., 2010].

Many multi-class approaches indirectly exploit the power of transfer learning. This is particularly dominant in frameworks where classes are composed of dictionary entries [Fidler et al., 2009, Opelt et al., 2008, Krempp et al., 2002]. These methods learn sequentially the classes by maximally reusing already known parts. Only when not enough similar parts are found in the dictionary, new parts of the target class are added. Progressively, this method requires less training samples for the new classes as the dictionary gets more complete. A downside is that the system depends on the order of the input classes.

Most transfer learning methods aim at improving the performance of the target classes. [Kuzborskij et al., 2013] go one step further and aim at improving the overall system performance. Its objective is that the k source classes gain in or keep their performance while the next target class benefits from the transfer. This is achieved with two regularization terms in the SVM like problem formulation. The first term aims at learning a target weight vector w_{k+1} close to a weighted combination β_i of the

source weight vectors $\hat{w}_i$:

$$\|w_{k+1} - \sum_{i=1}^{k} \beta_i \hat{w}_i\|^2$$

This term ensures re-using the knowledge of the already learned models. The second term

$$\|(w_1, ..., w_i, ..., w_k) - (\hat{w}_1, ..., \hat{w}_i, ..., \hat{w}_k)\|^2$$

forces the new class models w_i to be close to the previously $\hat{w}_i$ obtained one. It is especially common in the literature to keep the target hyperplane w_t close to at least one of the source models w_s in the form of a regularization term

$$\|w_t - \beta w_s\|^2$$

as investigated in [Aytar and Zisserman, 2011] with β a parameter. High values of β force the target model to be closer to the source model. They learn a target class by choosing manually a source class to transfer from. The weight vector w_s of the source is used to regularize the training process of the new class w_t. They explore several options of regularization one of which is the previously mentioned regularization term. However, one can prove that this implies a trade off between margin maximization and knowledge transfer. Therefore, they modify this term to

$$\|w_t\|^2 + \|w_t\|^2 sin\theta$$

with θ being the angle between w_t and w_s. Their last suggestion is to include the possibility to deform the source template w_s. This makes sense as local deformations make parts of objects more similar *e.g.* the wheel of a motorbike can be stretched to better resemble a bicycle's wheel. Their systems clearly allows to reach faster the final detection performance. One drawback is that one has to choose manually the source and target class.

When selecting two unrelated categories as source and target classes, one might take the risk of a negative transfer when training data is available for the target class. Indeed, [Yao and Doretto, 2010] attenuate this risk by not transferring only from one source class but a set of source classes. The classifier is based on boosting. Boosting progressively selects weak classifiers and increases at each iteration the weights of misclassified training samples. The authors suggest two new extensions to include the knowledge of other classes. First, they extend Adaboost to consider data of other classes. However, these wrongly classified samples of these source classes lose in importance if wrongly classified over the iterations as they might be dissimilar to the target class. Their second technique trains detectors for each source class and uses the obtained weak classifiers as a pool of weak classifiers to choose from for the target class during each iteration. This can be seen as a parameter transfer between classes. The latter has a short training phase while both methods show after experiments that learning from multiple sources is more stable.

Human tests have shown in [Canini and Griffiths, 2010a, Canini and Griffiths, 2010b] that people indeed adapt quickly to new objects by engaging in transfer learning which in return can be technically explained with hierarchical Dirichlet process [Teh,

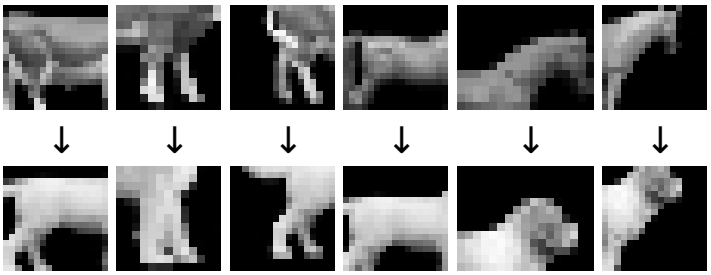
2010] (HDP). In addition, scientists observed that humans tend to generalize quickly from very few samples [Xu and Tenenbaum, 2007]. In the extreme case, only one training sample of the target class is available. In this case, we speak of one-shot learning. These conclusions are the foundations of the work of [Salakhutdinov et al., 2012]. One example is possibly only enough to describe the mean appearance of an object but by far insufficient to have an understanding of its appearance variations. Therefore, they structure the classes into a tree so that classes belong to more general nodes called super-classes. A tree is first trained on the available source classes. An additional target class will be place under a super-class or can create its own based on its similarity to the existing classes and the number of classes attached to a super-class. This new class inherits the model of the super-class based on their degree of resemblance which allows at least to transfer previously collected knowledge of closely looking sibling classes to the target model.

Similarly, [Fei-Fei et al., 2003] emphasize on the importance of a general prior object model to learn new classes. They propose a generative formulation where parts of objects are parametrized by appearance and shape. These parameters are given by an a priori probability density function. When learning a new category, its respective parameters are obtained by updating the previously defined general prior model.

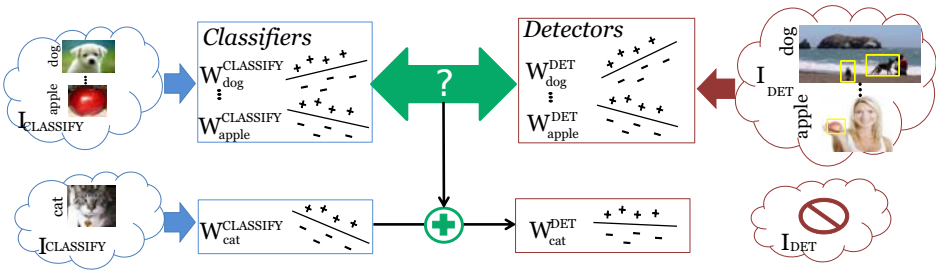
Objects of visually similar classes share common part locations. Therefore, [Bart and Ullman, 2005] learn a database of parts that are known to be important to the source classes. The categories are represented with parts. For the new class, the database is searched for the possible parts that match the learned ones. These located regions are replaced by the fragments in the target sample which form the new parts of the target class. This quickly learns a new class with only one training example. An example is shown in Fig. 2.14c. The cow and horse part fragments are matched to the ones in a dog image. These new extracted dog regions constitute the target's parts.

Another way of sharing information is by transferring transformations within a category. This found practice *e.g.* in [Miller et al., 2000]. Their work aims at letter classification where training samples can be represented as a deformation of a base letter. The authors augment a training dataset of the target class having one example by applying these transformations. A new detector can then be learned having a more complete dataset.

Instead of immediately using the training data or obtained models of the source classes, other approaches populate the training set with annotations. [Deselaers et al., 2010] aim at learning a generic knowledge which allows to annotate the training images of the target class without human interaction. The algorithm iterates between labeling bounding boxes in the images and learning more precisely the appearance model of that class. This is achieved by selecting one bounding box in each training image and by defining an energy function which is minimized when all these bounding boxes represent objects of the same class. The problem is formulated with a CRF. The unary potential measures the similarity with the current object model and the pairwise potential measures the similarity between bounding boxes of two images. After this initialization steps which populates the training set with bounding boxes, any object detector can now be trained on these automatically annotated data. An example model learned with their method is shown in Fig. 2.14b.



(c) [Bart and Ullman, 2005]



(d) [Hoffman et al., 2014]

Figure 2.14 – (a) The first row shows parts of cow or dog images. The second row shows the matching parts of the target class *e.g.* dog. The parts are matched using a cross-generalization algorithm to locate the most similar patches using only one target image (Courtesy of [Bart and Ullman, 2005]). (b) Image labels used for image classifications are easier to gather. This method learns an image classifier for both the source and target classes and object detectors for source classes. Then, it deduces from the source image classifiers and source object detectors a transformation function that converts the target image classifiers to target object detectors (Courtesy of [Hoffman et al., 2014]).

[Guillaumin and Ferrari, 2012] proceed differently. The classes are ordered in a tree structure built manually based on the semantic relationship between classes. The target class has siblings that is neighboring nodes on the same level attached to the same parent. The localization of the target instances uses information of all the target class' ancestors and siblings. Each of these nodes exploits three types of knowledge to model their class or super-class [2]: appearance, location and context. In a training phase, a probability distribution is learned for each source type and node. Using provided bounding boxes of the source categories, they learn weights to combine all these sources and localize correctly the bounding box. Finally, the information of the siblings and ancestors allows to detect the most probable location of the target. This is a very human like approach: the ancestors give general information of the target class while the target class profits from the close properties with its siblings. As an example, bus, van or cars share common properties which can be exchanged between sibling classes.

[2]The concept of a super-class will be introduced in later chapters. In short, a super-class encapsulates several classes.

[Vezhnevets and Ferrari, 2013] pursue the same idea. They describe every window in the images by its similarity to an Exemplar-SVM [Malisiewicz et al., 2011] (E-SVM). The E-SVM is learned on the source training set which contains annotations. Contrary to the previous approach, this method does not create a target model but rather describes the similarity of target windows to source models of the E-SVM. The target windows are sampled from the target image database using the objectness technique [Alexe et al., 2010]. This procedure allows to describe how a target window looks like rather than how it is. The approach produces one bounding box annotation per target image.

Learning object detectors for classes without annotated training data is the focus of [Lampert et al., 2009b]. They make the assumption that attributes can be shared between classes. Attributes can be *e.g.* color, texture or shape. The objects are defined by attributes *e.g.* a tiger has orange and black stripes. An unseen object class can be described through minimal human effort by defining these common properties. Their more powerful method called *direct attribute prediction* learns a first layer of attributes. At test time, these attributes are detected in the image region and both source and target classes are inferred by a weighted combination of these properties.

[Göring et al., 2014] rely on a nonparametric part model. They go through all the training images during test time and find all the images close to the test image using a nearest neighbor search. For each hit, using annotated part locations, features for these parts are extracted in the target image and combined into one feature vector. This feature vector is classified with an SVM which produces a histogram of class memberships. The combination of all the histograms obtained by each hit gives the final class label. It does not explicitly transfer knowledge but rather shares the poses and shapes between categories so that classes with small number of samples can be richly represented.

CNNs have found great success in image classification [Krizhevsky et al., 2012] and object detection [Girshick et al., 2014]. Annotating images for the classification task is a relatively easier job compared with the tedious annotations needed for the detection task. As a consequence, building classifiers even for large datasets *e.g.* ILSVRC is possible given image level annotations. [Hoffman et al., 2014] learn to transform image classifiers to object detectors. Their work requires that bounding box annotations for the k classes are available and that image level annotations for all classes are provided where one can choose the number of target classes much bigger than the number of base classes $k' \gg k$. The objective is depicted in Fig. 2.14d. They produce high quality detectors for both source and target classes where the latter ones have no bounding box information is given. The idea is to train a CNN of all the classes for the classification task and transform these classification models to detection models. More specifically, they first pre-train a CNN for the classification task on all classes and fine tune the layers for the k classes on the detection task. The fine tuning allows them to capture the transformation weights. For each target class, the nearest neighbor classes are selected and the target transformation is given by the average of the transformation weights of these selected classes. The similarity between the categories is defined by the Euclidean distance of the 8-th layer parameters.

Instead of learning additional classes, one can transfer knowledge between datasets and even tasks *e.g.* object classification and action recognition as in [Oquab et al.,

2014]. This is particularly important when dealing with CNNs. These learning algorithms need a huge amount of training data as provided by the ILSVRC2012 dataset. Consequently, learning on smaller datasets *e.g.* PASCAL VOC07 becomes challenging. The authors pre-train a CNN on the bigger dataset using large number of classes and replace the last layer with two additional fully connected layers that is trained on the target dataset. They sample patches in the target training set and use them to adapt these two last layers. The same approach is taken to learn a new task. The results show that this approach is significantly better than learning a CNN from scratch on the target set. Moreover, they achieve or outperform state-of-the-art techniques.

2.4 Conclusion

We have seen many different ways to improve performance and run-time of a multi-class object detector. Both metrics can be improved through the use of intelligent multi-class algorithms, exploiting the potential of fast hardware, better algorithmic designs, use of contextual information and sharing knowledge between object categories.

In our work, we focus on a new multi-class detection framework. We saw that inheriting global knowledge *e.g.* as in [Salakhutdinov et al., 2011, Guillaumin and Ferrari, 2012] plays a crucial role to learn discriminative classifiers. We are able to imagine a falcon in its various views thanks to our general image of its species. Results have shown that these methods outperform the basic OvA method. Many of the previous works ordered classes into a hierarchical structure and inspired strongly our work. But in most works, these models only apply with a specific set of features *e.g.* [Fidler and Leonardis, 2007, Zhu et al., 2010a]. Our objective is to keep the system as flexible as possible considering the choice of the features. We introduce a new optimization method that exploits the training samples of all classes in a joint problem formulation which differentiates us from other works such as [Salakhutdinov et al., 2011]. Moreover, our work naturally allows to have fast run-times and the advantages of transfer learning. This latter property is interesting to be able to learn a new object class even with little training samples.

CHAPTER 3

Learning with Support Vector Machines

Contents

OBJECT detection consists in localizing known object classes in images and videos. The computer has to differentiate between target object categories and background regions. The multitudes of approaches rely on a training stage where the computer learns to understand the characteristics of foreground and background objects. Usually example data of these objects known as *training data* is a priory available. In machine learning, this case is called *supervised learning* as every example in the training data is paired with its desired output values namely its class label and location in the image. If the labels of the training examples are not provided, the machine learning task consists in designing algorithms that can determine hidden structures in these data. This domain is called *unsupervised learning*. This case can for example arise if the class labels are unknown or the images are not annotated. We speak of *semi-supervised* learning if only partial information on the training examples are known in advance. Various algorithms exist for the learning stage that often build the core module in an object detection system. In this report, we mainly focus on supervised learning: The object classes are known and training data is available for both foreground and background objects.

The systems rely on extracting information about the objects put in a form of a vector known as *feature vector*. Each element in this vector carries an information and the ensemble of the elements helps to generalize between an object category and other ones. The feature vectors of the example instances are fed to the learning algorithm which in return produce a model of the underlying object class.

An advanced version of SVM builds the main brick of our detection system and in this chapter, we will give an introduction to SVM. It allows to better understand the next chapters and notations. We will have a look on the binary classification task and how to formalize the task itself. The goal of binary classification is to separate between two categories of objects *e.g.* {'car', 'background'}. Next, we describe two methods to solve the problem formulation. Both methods are essential for the further understanding of this work. The sections allow to get a deeper insight in how the optimization modules look like. Finally, we will review extensions of the SVM principle which are closely related to and inspired our work.

3.1 Introduction to SVM

As mentioned earlier, the samples we want to classify are projected into a hyperspace of dimension d by representing each sample i with a feature vector $x_i \in \mathbb{R}^d$. These samples are associated to a class label $y_i \in \mathcal{Y}$. $\mathcal{Y}$ is the set of all possible classes. The classification function $f(x)$ assigns to an input vector x a target class $y \in \mathcal{Y}$:

$$\begin{aligned} &\mathbb{R}^d \to \mathcal{Y} \\ &x \to f(x). \end{aligned} \tag{3.1}$$

The result of a classification learning algorithm is $f(x)$ which represents output model. The goal is to design a model making the least number of errors on unseen data. We have a total of n data points given.

At first we treat only binary labels $\mathcal{Y} = \{-1, 1\}$ and extend later to arbitrary number of classes. Let's further assume that our samples are linearly separable. That means that samples of both classes can be separated by a hyperplane

$$w \cdot x_i + b = 0, \tag{3.2}$$

$w \in \mathbb{R}^d$ is called weight vector and b is the bias of the hyperplane. Finding quickly the optimal values for (w, b) is the challenge of any SVM solver. We distinguish between positive $y_i = 1$ and negative $y_i = -1$ classes. The samples of the positive class label have a positive distance to the hyperplane and similarly the samples lying under the hyperplane are classified as negative. It follows that for an example i we have:

$$y_i(w \cdot x_i + b) > 0. \tag{3.3}$$

However, every hyperplane given by (w, b) can be expressed by any scaled hyperplane $(\lambda w, \lambda b)$ with $\lambda \mathbb{R}^+$. Therefore, we define the normalized hyperplane where the distance to at least one positive and one negative example is 1. The hyperplane is placed exactly in the middle of both samples producing a maximum margin between both classes. We

can rewrite Eq. (3.3) as:

$$y_i(w \cdot x_i + b) > 1. \tag{3.4}$$

Here the classification function is expressed as:

$$f(x) = \begin{cases} +1 & \text{if } (w \cdot x_t + b) \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{3.5}$$

If the function is positive, we classify the example as a positive one and equivalently if the sign is negative, we give it a negative label.

Constrained Optimization Problem

The geometric distance of a sample to the hyperplane is simply:

$$d_i = \frac{y_i(w \cdot x_i + b)}{\|w\|} \tag{3.6}$$

We want the samples to have the largest geometrical distance to the hyperplane (w, b). In other words, we want to maximize $d_i \geq \frac{1}{\|w\|}$. Finally, we get the constrained SVM optimization problem:

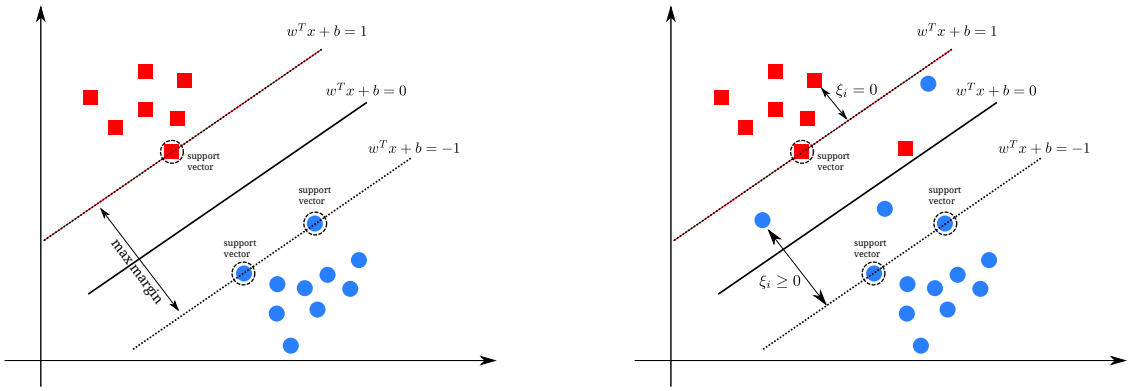
$$\min_{w,b} \ \frac{1}{2}\|w\|^2 \tag{3.7a}$$

$$s.t. \quad y_i(w \cdot x_i + b) > 1 \quad \forall i = 1, \ldots, n. \tag{3.7b}$$

We assume the binary classification task where samples of both classes are linearly separable as depicted in the example Fig. 3.2a. In practice, this is usually not the case (see Fig. 3.2b). Therefore, we introduce non-negative slack variables ξ_i representing an error term we want to minimize:

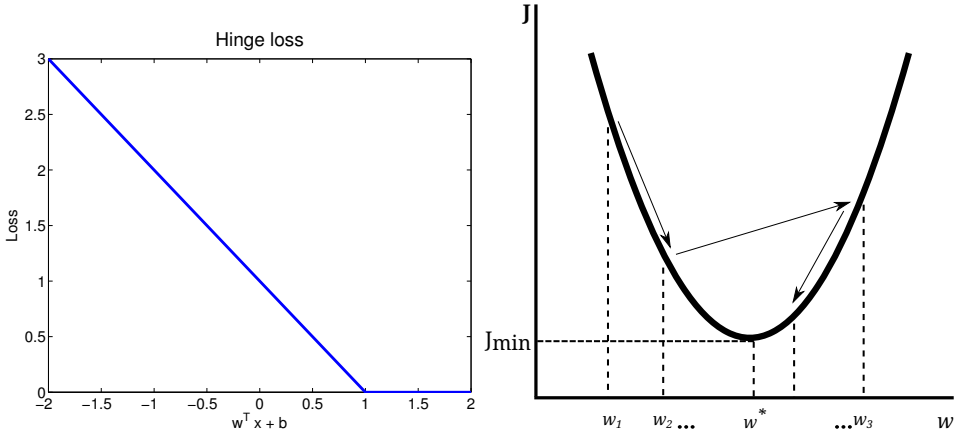
$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ &s.t. \quad y_i(w \cdot x_i + b) \geq 1 - \xi_i \quad \forall i = 1, \ldots, n \\ &\qquad \xi_i \geq 0 \end{aligned} \tag{3.8}$$

There exists a slack variable ξ_i for every example $i = \{1, ..., n\}$. A positive value for ξ_i means that an example is located on the wrong side of the hyperplane and if $\xi_i = 0$ the example is classified correctly. Small values for ξ_i implicate less errors on the training examples. C is a trade-off parameter between minimizing the errors of the data points and finding the minimum margin. If C is chosen large, the SVM solver tries to find the margin with no errors on the training set. This gives poor generalization ability to unseen data as the margin between the samples is small. If C is too small, we will make too many mistakes on the training set and eventually generalize too much instead of finding discriminative characteristics between the labels.



(a) (b)

Figure 3.1 – Illustration of the notions in SVM. The red rectangles represent one object class and the blue circles another one. SVM determines a hyperplane that separates these two classes. It thrives to place the hyperplane as to maximize the margin between both categories. The samples located on the margin are called support vector machines and are crucial for locating the hyperplane. (a) The separable case: the optimal hyperplane classifies correctly all training samples. (b) The non-separable case: it is not possible to separate the classes by a linear hyperplane. This limitation is overcome in the problem formulation by introducing slack variables ξ_i. For samples lying inside the margin or that are wrongly classified, $\xi_i > 0$ and otherwise $\xi_i = 0$.



(a) (b)

Figure 3.2 – (a) Visualization of the hinge loss used when scoring training samples with the current model. All samples that are scored correctly with an absolute value higher than 1 have zero loss. Otherwise the loss depends linearly with the respective value. (b) Work flow of a gradient descent algorithm. Starting at a random point w_1, it iterates by following the negative gradient until reaching ideally the minimum $J_{\min}$ at w^*.

Unconstrained Optimization Problem

The SVM problem can be seen from a different angle which leads to another computational approach to solve the problem. The idea is to put the constraints in Eq. (3.8) into the objective function:

$$\begin{aligned} &\min_{w,b} J(w,b) \\ &J(w,b) = \underbrace{\frac{\lambda}{2}\|w\|^2}_{\text{regularization term}} + \underbrace{\sum_{i=1}^{n} \ell(x_i, y_i)}_{\text{loss term}} . \end{aligned} \tag{3.9}$$

Eq. (3.9) is composed by a regularization term and a loss function. We can use various loss functions. The most commonly used is the hinge loss (see Fig. 3.3a):

$$\ell(x_i, y_i) = \max(0, 1 - y_i(w \cdot x_i + b)). \tag{3.10}$$

It states that there is no loss as long as the samples are correctly classified. For wrongly labeled samples, the penalty term increases linearly. For a discussion of the loss terms see *e.g.* [Chapelle, 2007]. The problem formulation states that we wish to minimize the errors made on the training set. As there may be different values for (w, b) satisfying this condition, we use a regularization term. This additional term avoids overfitting the data. The trade-off is controlled by λ. Again if λ is small, we risk to overfit to the training data and on the other hand, concentrate less to the given samples. There is an equivalence between λ in Eq. (3.9) and C in Eq. (3.8), namely $\lambda = \frac{n}{C}$.

3.1.1 Solving in Primal

The problems in Eq. (3.9) and (3.8) are called primal formulation of SVM. One possible way of solving the primal problem is using a convex minimization technique which we implemented to solve our optimization problem in Sec. 4.3.2. The goal is to find the optimal solution $\tilde{w}^* = (w^*, b^*)$. There exists a variety of techniques. We will shorty review some of them which work iteratively. For a complete description of those approaches, we will refer the reader to [Boyd and Vandenberghe, 2004]. Let t be the current iteration step. These techniques try to find a better solution $\tilde{w}_{t+1}$ using the current solution $\tilde{w}_t$ and a non-negative scalar step size $\eta_t \in \mathbb{R}^+$.

The simplest one is *coordinate descent* which updates the current solution in one of its i-th coordinates. In practice this approach is very slow but it can be applied if approximate solutions are good enough. *Gradient descent* also called *steepest descent* converges in the direction of the gradient of the current solution. While these algorithms can lead to zigzagging, *conjugate descent* tries to avoid this phenomenon by finding linearly independent conjugate directions. These methods require that the function J be differential. If the functions are strictly convex and twice differential, we can apply higher order methods which use the Hessian to discriminate the coordinates differently. Among these approaches count the *Newton's method* using the Hessian and the more applied one *LBFGS* using an iterative estimate of the Hessian matrix.

In the context of this thesis, we formulate a new convex optimization problem that can be solved using one of these methods. We opted for the gradient descent technique

for its popularity [Chapelle, 2007] and its widely use in object detection systems *e.g.* [Felzenszwalb et al., 2010a].

Gradient descent updates the current solution in the direction of its first degree gradient $\nabla J(\tilde{w}_t)$. The update function is:

$$\tilde{w}_{t+1} = \tilde{w}_t - \eta_t \nabla J(\tilde{w}_t). \tag{3.11}$$

The principle is depicted in Fig. 3.3b. Given a starting point, the algorithm moves in the direction of its gradient. We need to carefully choose the step size η_t. A too large step size leads to a zigzagging and a too small step size increases the convergence rate. A good initialization w_0 helps to converge quickly to the final solution $\tilde{w}^*$. An often used technique to determine η_t is called line search. It allows to find optimal values for η_t allowing quicker convergence. Other techniques are the use of a fixed step size or a decaying one. We follow the extensive work of Bottou *et al.* [Bottou and Bousquet, 2007, Bottou, 2010, Bottou, 2012] and use a decaying step size of

$$\eta_t = \frac{1}{t}. \tag{3.12}$$

In order to reach an accuracy of ϵ, Bottou *et al.* prove that the algorithm needs $\log(1/\epsilon)$ iterations. The accuracy is reached when the found solution $\tilde{w}_t^*$ fulfills the following condition: $J(\tilde{w}_t^*) < J(\tilde{w}^*) + \epsilon$. Coming back to the task of the object detection, not all the examples can be loaded into memory.Therefore, we applied *stochastic gradient descent* which is a modified version of gradient descent. Instead of calculating the new direction of update based on all examples, it uses only one example at a time to update the current solution. The number of iterations is given by $1/\epsilon$.

To implement such a scheme, the first thing we need is the gradient of our objective function $J(w, b)$. The sub-gradient at example x_t with respect to w is given by:

$$\frac{\partial J(w,b)}{\partial w}(x_t) = \begin{cases} \lambda w & \text{if } y_t(w \cdot x_t + b) > 1 \\ \lambda w - y_t x_t & \text{otherwise} \end{cases}. \tag{3.13}$$

Similarly, the sub-gradient with respect to b is:

$$\frac{\partial J(w,b)}{\partial b}(x_t) = \begin{cases} 0 & \text{if } y_t(w \cdot x_t + b) > 1 \\ y_t & \text{otherwise} \end{cases}. \tag{3.14}$$

In the literature, the bias b term is often neglected or treated as the last dimension of the feature vector and weight vector. This is achieved by increasing the dimension of the feature vector by 1 and adding b as a further component to w as mentioned in [Duda and Hart, 1973, Shalev-Shwartz et al., 2007]. By doing so, we are not solving anymore exactly the same problem and noticed a loss in performance in practice.

A typical stochastic gradient descent algorithm is shown in Alg. 0. The algorithm iterates over all examples one after another. For each example, it calculates the first degree gradient separately for w and b and updates the best current solution in the direction of the gradient weighted by the step size. Over time, the step size decreases to avoid a zigzagging behavior.

Algorithm 0: Basic stochastic gradient descent

Initialization: Set $t = 0$ and $w_o = 0$
T : number of examples
while $t < T$ **do**
  Calculate gradient for data point t using Eq. (3.13) and Eq. (3.14)
  Compute step size: $\eta_t = 1/t$
  Update model:

$$w_{t+1} = w_t - \eta_t \frac{\partial J(w,b)}{\partial w}(x_t)$$
$$b_{t+1} = b_t - \eta_t \frac{\partial J(w,b)}{\partial b}(x_t)$$

  $t = t + 1$
return (w_T, b_T)

However in practice, we use a slightly different version as shown in Alg. 1 which produced much better results in our experiments. There are several modifications made to the basic version Alg. 0. The first thing to note is the outer loop over the examples. While before we made only one run over all examples to converge, we now go several times over the current training set. Next, we permute the examples to avoid getting stuck in a local minimum. We also use the concept of a waiting list: All the examples that are incorrectly classified in one iteration are again used in the next iteration. However, the correct ones are kept INCACHE iterations and if these are still correctly classified, they end up in the waiting list. They remain in there (WAIT-INCACHE) times and do not influence the gradient descent algorithm. For faster learning times, we use a stopping criteria. There is again a multitude of stopping criteria. We calculate the relative difference in norms of the current solution and the one of the previous iteration:

$$\delta w = \frac{\|w_{t+1}\| - \|w_t\|}{\|w_t\|}. \tag{3.15}$$

The stopping criteria is thus given by $\delta w < 0.0002$ which means that our model did not change significantly over the iteration. We also tune the step size using two parameters $(K, \bar{K})$. This part is especially important when dealing with deformable parts as in Sec. 6 where for different regions of the weight vector, different values for the step size are chosen. Finally, we also lower bound the model by not letting it diverge. In practice, this showed to be especially useful when making use of deformable parts where we lower bound the possible deformations of the model.

We implemented both methods and used them as optimization modules to our problem formulation in Sec. 6. These simple modifications to the basic algorithm Alg. 0 showed significant improvements in practice. The basic gradient descent algorithm yields a too simplistic final solution producing much lower results. We further tested the use of a line search algorithm to determine the optimal values for η_t. We found that this does not help the performance but only increases the learning time.

Algorithm 1: Stochastic gradient descent

Initialization: Set $t = 0$ and $w_o = 0$
T : number of iterations
while $t < T$ **do**
    Permute examples
    foreach *examples* i **do**
        if *wait_time*$[i] >$ *INCACHE* **then**
            wait_time$[i]$ = wait_time$[i] - 1$
            go to 5
        $t = t + 1$
        Calculate gradient for data point i using Eq. (3.13) and Eq. (3.14)
        Compute step size: $\eta_t = K/(t + \tilde{K})$
        Update model:

$$w_{t+1} = w_t - \eta_t \frac{\partial J(w, b)}{\partial w}(x_i)$$
$$b_{t+1} = b_t - \eta_t \frac{\partial J(w, b)}{\partial b}(x_i)$$

        Classify example i using updated model (w_{t+1}, b_{t+1})
        if *incorrectly classified* **then** wait_time $= 0$
        else if *wait_time* == *INCACHE* **then** wait_time$[i]$ = WAIT
        else wait_time$[i]$ = wait_time$[i] + 1$
    Check stopping criteria using 3.15: $\delta w < 0.0002$
    Apply lower bounds

3.1.2 Solving in Dual

Solving a SVM has been especially studied in its dual form. We will mention its advantages and the tool we used to solve it. We will not go into the details and refer the reader to various sources *e.g.* [Hamel, 2009, Bishop, 2006]. In our experiments, we applied both a primal and a dual solver depending on the extracted features.

Again, let us understand the dual form in its simplest linearly separable case. Following the representer theorem [Hamel, 2009], the solution w can be written as:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i, \tag{3.16}$$

with $\alpha_i \in \mathbb{R}$ or in the form of a vector $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$. In other words, the solution vector w is a linear combination of the training data. The dual learning

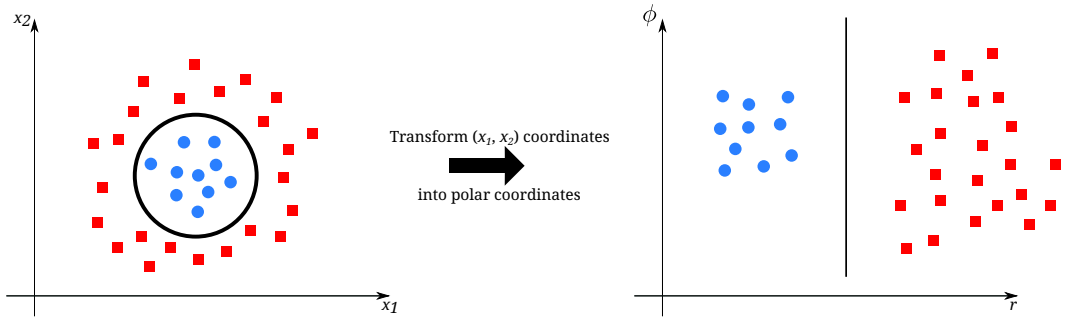


Figure 3.4 – The example data on the left of the figure is not ideally separated by a hyperplane but a circle. However, the data becomes linearly separable when mapping the data to polar coordinates. The SVM problem can be solved in these new dimensions. This is a common approach to handle linearly non-separable data and it is often not even required to explicitly transform the features into a new dimensional space.

problem is stated as follows:

$$\begin{aligned} &\max_{\alpha_i \geq 0} \sum_i^n \alpha_i - \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j (x_i^T x_j) \\ &\text{s.t.} \quad 0 \leq \alpha_i \leq C \\ &\qquad \sum_i \alpha_i y_i = 0. \end{aligned} \tag{3.17}$$

Until now, we considered the simple linearly separable case. In the primal version, we showed a remedy to this problem by introducing slack variables. There is another solution especially interesting in the dual formulation by the mean of a feature map:

$$\begin{aligned} &\phi(x) : x \to \phi(x) \\ &\mathbb{R}^d \to \mathbb{R}^D. \end{aligned} \tag{3.18}$$

The feature map transforms the input feature x into a higher dimensional feature vector $\phi(x)$ with dimension D. The key idea is that, in the new dimensional space with $D \gg d$, the data becomes linearly separable. Then, we only need to find the hyperplane separating the data in this space. We show this in an illustration in Fig. 3.4.

The sign of label is simply given by $sign(w \cdot \phi(x) + b)$ and the learning problem in the primal Eq. (3.9) can be written with the new loss function :

$$\ell(x_i, y_i) = \max(0, 1 - y_i(w \cdot \phi(x_i) + b)). \tag{3.19}$$

Consequently, the classification is more time consuming. Furthermore, the learning problem we need to determine has much more parameters as the dimension of the features and the solution is now higher. This problem is avoided in the dual formulation as the classification task is written as:

$$f(x) = \sum_i^n \alpha_i y_i \underbrace{(\phi(x_i) \cdot \phi(x))}_{k(x_i, x)} + b. \tag{3.20}$$

$k(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ is called the kernel. The learning problem becomes:

$$\begin{aligned} &\max_{\alpha_i \geq 0} \sum_i^n \alpha_i - \frac{1}{2} \sum_i^n \sum_j^n \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ &\text{s.t.} \quad 0 \leq \alpha_i \leq C \\ &\qquad \sum_i \alpha_i y_i = 0. \end{aligned} \tag{3.21}$$

The kernel function depends on the dot product of the mapped features. In practice, $k(x_i, x_j)$ can be computed without the need to calculate $\phi(x_i)$. Often the mapping function is not even known but only the kernel function is given. This is called the *kernel trick* where the data is treated in a high dimensional space without explicitly transforming the features. In the case of $n \ll D$, the solving in the dual is of special interest. We did not code this solver but used the popular SVM^{light}package [Joachims, 1999a] which is available for download under [Joachims, 1999b].

3.2 Extension for Multi-class Classification

The SVM discussed earlier can only handle binary labels $\mathcal{Y} = \{-1, 1\}$. In this work, we focus on multi-class classification where the label set is given by $\mathcal{Y} = \{y_1, \ldots, y_k\}$ with k the number of categories. The classification function $f(x)$ maps an input vector x into one of these possible output classes.

3.2.1 Classical Techniques

A straight forward approach is known as the One-versus-All (OvA) technique. It consists in creating one classifier for each class $\mathcal{Y}^+ = y_i$ and taking all the other classes as the negative set of classes $\mathcal{Y}^- = \{y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_k\}$. It produces a total of k binary classifiers (w_i, b_i). During classification, one applies each classifier separately on the example x. The label is attributed to the highest scoring class:

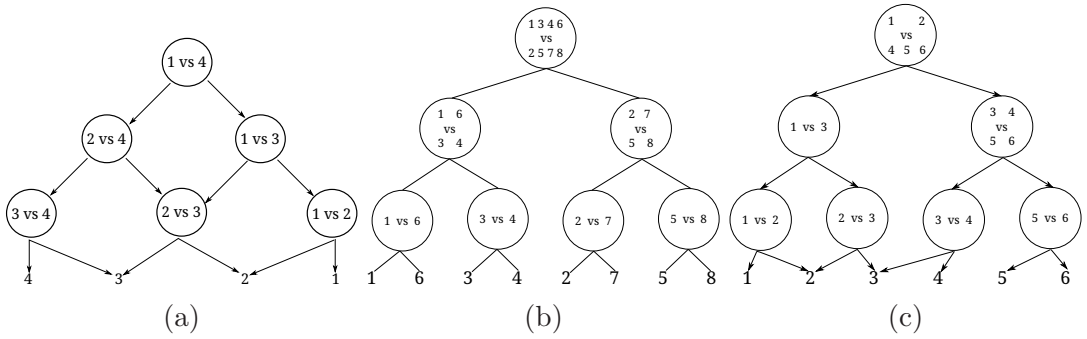
$$y = \arg \max_{i=\{1,\ldots,k\}} w_i x + b_i \tag{3.22}$$

The output scores of each class are not comparable to each other though the filters (w_i, b_i) are not trained simultaneously. One remedy is to normalize the scores using the work of [Platt, 1999]. This scaling approach allows to transform the score to a probabilistic output:

$$\mathrm{P}(y = 1|x) = \frac{1}{1 + \exp(A(wx + b) + B)}. \tag{3.23}$$

The scalar parameters A and B are learned by an algorithm [Platt, 1999] such as a maximum likelihood method. Usually these parameters are estimated on the validation data set.

A different angle of solution is proposed by Crammer *et al.* [Crammer et al., 2001] who cast the multi-class problem into a single optimization problem instead of decom-



(a) (b) (c)

Figure 3.5 – (a) One-versus-One (OvO) method: One class is eliminated after another. (b) A binary decision tree: Only a subset of classes is retained after passing through each node in the hierarchy. (c) Again a binary decision tree but with a relaxed hierarchy: A subset of classes is eliminated at each step but the subsets do not necessarily need to be disjoint. A class can be grouped into several branches in the tree.

posing it into multiple binary problems. They aim to *rank* the best scoring class the highest. This property is learned during the optimization stage:

$$\begin{aligned} \min_{w,b} \; & \frac{1}{2}\|(w_1, \ldots, w_k)\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ s.t. \quad & w_{y_i}^T x_i - w_j^T x_i \geq 1 - \xi_i \\ & i = 1, ..., n, \; \forall j \in \mathcal{Y} \setminus \{y_i\} \\ & \xi_i \geq 0. \end{aligned} \tag{3.24}$$

In this cases, the optimization problem is posed as a ranking problem. The objective is to find filters w_i scoring higher than each other class. Contrary to before the filters are not trained independently. However solving this problem is more difficult as all the dimension of the filters have to be considered simultaneously. The run-time complexity for both previously mentioned techniques is $\mathcal{O}(k)$ as k classifiers are evaluated one after another.

Another approach is the One-versus-one (OvO) [Kressel, 1999] technique. Here, a pair of binary classifiers are built for each possible class combination (y_i, y_j) with $i \neq j$. This gives a total of $k(k-1)/2$ classifiers. For example, for the class y_1 there are all possible classifiers that distinguish between y_1 and y_i, $i \in \mathcal{Y} \setminus \{1\}$. During detection, the pair of classifiers are evaluated and we retain the label of the class getting the most *votes*. The binary filters are trained again independently. Maybe the most important advantage is that each classifier only needs to be able to distinguish between two classes. This reduces also the number of examples each classifier needs to treat during training. The run-time is $\mathcal{O}(k(k-1)/2)$.

Another idea, inspired by the OvO, is the work of [Platt et al., 2000] called DAGSVM. Again all the filters only have to distinguish between two labels. But this time the filters are arranged in a hierarchical order. An example is shown in Fig. 3.6a. The first filter distinguishes between classes y_1 and y_4. Is the sample classified as y_1, one needs to further verify with the remaining classes. This is done again following the same procedure of hierarchical filtering. Knowing that the class is not y_4 avoids

evaluation with the pair of classifiers $(y_i, y_4) \quad i = \{1, 2, 3\}$. This is possible through a hierarchical exclusion system. This technique combines the strength of both OvO and OvA: The filters are trained on only two classes and the run-time is $\mathcal{O}(k)$ though much faster than OvO. A comparison of these methods is given in [Hsu and Lin, 2002].

3.2.2 Decision Trees

Decision trees have been largely used in image classification for reducing the run-time when dealing with a large number of classes k. We will give a review of the principle ideas used in image classification with decision trees. More precisely we focus on trees trained using an SVM-like approach. This allows to better understand our work as it is inspired by these methods and on a more general level to differentiate ourselves from this other research domain. The task of classification consists of distinguishing between k classes that one can model. All the classes are ideally characterized by some features which allows them to be differentiated with other ones. This is a major difference to the detection domain where the background class cannot be modeled due to its huge variability in appearance.

Following the same spirit of hierarchical classification, [Takahashi and Abe, 2003] propose to employ binary decision trees. In [Platt et al., 2000], the right class is found in an elimination process one class after another. Here, the authors suggest to eliminate a set of classes one after another. An example is illustrated in Fig. 3.6b. The first node in the tree distinguishes between two sets of classes $\{1, 3, 4, 6\}$ versus $\{2, 5, 7, 8\}$. The lower nodes repeat the same process by refining the set of possible solutions. At the bottom of the tree, the set consists of one single class determining the class label. Again each node is a simple binary SVM where one set is used as positive examples and the other set as negative ones. A key role plays the structure of the tree. The authors aim at keeping similar classes close and separate the most distinguishable classes early in the tree. This is achieved by hierarchically clustering the examples in the feature space.

The more general idea of decision trees is to build classes in a tree structure with |T| nodes. Let n_i $i \in \{1, ..., |\mathrm{T}|\}$ be the node except a leaf node. The leaf nodes represent individual classes and the intermediary nodes group several classes together. Recursively these classes are combined into larger sets of classes. At the root node handles all the classes. Each node except the leaf node is multi-class SVM discriminating several sets of classes that it represents. In case of a binary decision tree which is the common method in literature, the nodes are simple binary SVMs handling two sets. The child classes are a subset of its parent classes. The set a class y_j at a node n_i belongs to is given by the function $\mathcal{C}_i(y_j)$ which can be in a binary tree ± 1.

During inference, the image traverses the tree from top-to-bottom. At each node, the image is attributed to a child node based on the SVM's decision function. The image continues with the child node which scored highest. Thus, it is relevant to create a hierarchy that reduces the error of the tree. The similar classes are grouped into the same node at a given level. Trees can be build in various ways that we mention further in Sec. 4.3.1. A naive implementation would train all the nodes independently from

each other:

$$\begin{aligned} \min_{w} & \sum_{i=1}^{|\mathrm{T}|} \Big(\frac{1}{2} \|w_i\|^2 + \frac{C}{n} \sum_{j=1}^{n} \xi_j \Big) \\ s.t. \quad & \mathcal{C}_i(y_j) w_{n_i}^T x_j \geq 1 - \xi_j \\ & \xi_j \geq 0. \end{aligned} \tag{3.25}$$

An important advantage of binary balanced decision trees is their run-time. In case of a balanced tree, the run-time becomes $\mathcal{O}(\lceil log(k) \rceil)$ which is much smaller with large number of k compared to a OvA method having a run-time of $\mathcal{O}(k)$.

A similar path was followed in [Bennett and Blue, 1997] using again binary decision trees learning the tree structure and filters simultaneously. [Madzarov et al., 2008] cluster using distance measures in the kernel space to build the tree instead on the pure descriptor level. [Yang and Tsang, 2012] concentrate too on how to build an effective hierarchy by finding sets of classes that produce the maximum margin. It is posed as an integer problem that aims to associate a group label to each example. In a tree, similar classes have a short distance. [Griffin and Perona, 2008] explore tree construction from bottom-to-up and top-to-bottom and conclude that both methods are equally performing. The nodes are learned independently and each node is a binary SVM. This similar learning and inference method is applied in [Binder et al., 2012] but with a semantically predefined tree structure as the authors focus more on discriminating biologically unrelated classes. [Bengio et al., 2010] extent Eq. (3.25) to jointly learn all filters in the tree. The tree itself is build from top-to-bottom using hierarchical spectral clustering. Furthermore, they embed the features into a much smaller dimension that takes into account the overall tree loss by preserving the semantic similarity between classes.

[Dekel et al., 2004] exploit this knowledge to reduce errors between close nodes. They derive an on-line algorithm where not all vertices in the tree are updated. [Hao et al., 2007] apply also a clustering approach inspired by SVM. Kd-tree are exploited in [Cevikalp, 2010]. [Fei and Liu, 2006, Gao and Koller, 2011, Marszalek and Schmid, 2008] leverage the strict separation between classes by allowing several leaf nodes sharing the same class label as depicted in Fig. 3.6c. Assuming a binary tree, the object classes at each node are grouped into 2 categories. The confusing classes unable to be separated by a hyperplane are assigned to both child nodes and are ignored during training for that node. This gives more flexibility to handle intra-class variability. One has further to balance the trade-off between accuracy and run-time. In case of relaxed hierarchies, the number of nodes increases while the number of errors decreases as upper-level nodes are less prone to incorrect decisions. [Fan, 2005] apply a coarse-to-fine search in a tree. Certain features are exploited and evaluated based on the branch of the tree being passed.

Finally, we would like to mention the work of [Sun et al., 2013]. They build a tree using hierarchical clustering and optimize the framework with the structured SVM learning package of [Tsochantaridis et al., 2004] that we introduce later in Sec. 3.2.3. This approach needs to run all the filters in the tree which are not binary filters any more. The final score of each class is given by the sum of the filters' scores lying on its path down to the leaf node. To speed up their framework, they use the A*

algorithm to quickly find the optimal path without the need to evaluate all the filters. This approach comes very close to our system and was published in the same period of time. However, our system relies on an extended optimization algorithm and is able to handle a background label required in object detection tasks.

One major downside of these approaches are the need to model all the k object categories. In the presence of a background class, this raises the challenges on how to model the background knowing that it is subject to high variations. An application of the previous methods is not possible as, currently, there exists no descriptor being able to successfully capture these high intra-class variability. We overcome this problem by not seeing each node as a binary classifier but as a part of a much higher dimensional classification scheme.

3.2.3 Structured Output Classification

Structural support vector machines (S-SVM) generalize the classical SVM to be able to handle more complex output representations such as trees, sequences or sets. In other words, the class labels of the data do not only belong to one of the k classes as in multi-class classification but to a structured output. Also the input data can be structured *e.g.* in form of a tree, which is the case in our work.

The work of Cai *et al.* [Cai and Hofmann, 2004] is, to our best knowledge, among the first ones to propose a structured version of SVMs. First, they get rid of a simple feature representation x but introduce instead the combined feature representation of inputs and outputs $\phi(x, y)$. Depending on the output y and input x, $\phi(x, y)$ represents a different feature vector. An example of this principle is the case of multi-class classification using this combined feature representation. The dimensions of the feature vectors for each class y_i, $i \in \{1, ..., k\}$ do not necessarily need to be of the same length. The combined feature vector would be given by:

$$\phi(x, y) = \begin{pmatrix} \vdots \\ 0 \\ x \\ 0 \\ \vdots \end{pmatrix} \longleftarrow y\text{-th position} \tag{3.26}$$

Neglecting the bias term b of SVM introduced in Sec. 3.1 for simplicity of notation, the scoring function becomes $f(x, y) = w \cdot \phi(x, y)$. The weight vector w is then a concatenation of weight vectors of individual classes w_i $i \in \{1, ..., k\}$. In a basic OvA formulation, the weight vectors w_i are optimized independently from each other. This is not the case with the S-SVM formulation. The high dimensional hyperplane w optimizes simultaneously all the filters w_i. As in multi-class classification, the best output label is assigned to the class y_i resulting in the highest score $\max_{i \in \{1,...,k\}} w \cdot$

$\phi(x, y_i) = w_i \cdot x$. This is a ranking problem very similar to Eq. (3.24):

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i \\ &s.t. \quad w \cdot (\phi(x_i, y_i) - \phi(x_i, y)) \geq 1 - \xi_i \quad \forall i = 1, \ldots, n; y \in \mathcal{Y} \\ &\qquad \xi_i \geq 0. \end{aligned} \tag{3.27}$$

Here again we make use of a ranking formulation of the problem. The major difference is the use of a combined feature descriptor giving much more space to model different kinds of problems such as in natural language modeling, multi-class classification, classification with tree, label sequence learning and context free grammars.

The formulation in Eq. (3.27) is challenging to minimize: Every class y_i is combined with another class y of all possible classes. The growth of the constraints is exponentially. The above problems can be solved in various ways. The work in [Cai and Hofmann, 2004, Tsochantaridis et al., 2004] use the dual of the formulation in Eq. (3.27) to derive a fast solver. They showed that only a subset of the constraints having polynomial size is sufficient for optimization with an accuracy of ϵ. These methods are based on the *cutting plane* algorithm. The idea of cutting plane is not to optimize all the constraints at once as the optimal solution only needs to satisfy only a subset of the constraints. The algorithm iteratively construct a *working set* of constraints $\mathcal{W}$. It does contain some constraints that were violated respectively not fulfilled with the current choice of the weight vector. This reduced working set $\mathcal{W}$ is then optimized and the solution gives the new choice of the weight vector. The iterations stop once the cache of constraints is fulfilled with a precision of ϵ. The authors show that at most $\mathcal{O}(\frac{1}{\epsilon^2})$ constraints are needed.

A major work is done by Joachims *et al.* [Joachims et al., 2009] who also uses a cutting plane approach. At each iteration of their algorithm, not all the constraints given by the possible number of classes is considered. For each sample only the class that violates most the pool of constraints is selected. This gives a number of constraints proportional to the number of samples. Furthermore, all these constraints are summed together. This new formulation is often referred to as a 1-slack formulation of the S-SVM problem:

$$\begin{aligned} &\min_{w,b} \frac{1}{2}\|w\|^2 + C\xi \\ &s.t. \quad \forall(\bar{y}_1, ..., \bar{y}_n) \in \mathcal{Y}^n : \\ &\qquad \frac{1}{n} w^{\mathrm{T}} \sum_{i=1}^{n}[\phi(x_i, y_i) - \phi(x_i, \bar{y}_i)] \geq \frac{1}{n}\sum_{i=1}^{n}\Delta(y, \bar{y}_i) - \xi \quad \xi \geq 0. \end{aligned} \tag{3.28}$$

For the sake of completeness we included the loss function $\Delta(y_i, \bar{y}_i)$ which penalizes the wrong assignment of class y_i to $\bar{y}_i$. Naively, this can be set to 1 if the two classes differ $y_i \neq \bar{y}_i$ and to 0 if both classes equal each other $y_i = \bar{y}_i$. The new constraint is then added to $\mathcal{W}$ and optimized in the dual form. We will detail this idea in more depth in Sec. 4.3.3 where we exploit it to have a fast optimization method. The author proves that the number of constraints converges at most in $\mathcal{O}(\frac{1}{\epsilon})$. The package of this work is publicly available under the authors website [Joachims, 2008].

A similar approach is described in [Uricar et al., 2013]. Their work is inspired by the bundle method of Teo *et al.* [Teo et al., 2007]. The structured output problem is completely solved in its primal formulation where the problem is described in a unconstrained way. A function $J(w)$ is lower-bounded by a sub-gradient at w_0:

$$J(w) \geq J(w_0) + (w - w_0)^{\mathrm{T}} \frac{\delta J}{\delta w}(w_0)$$

At each iteration, these lower bounds are minimized and its optimal point is the new trial point. To avoid zigzag behavior, the authors suggest to add a prox-term in the objective $\|w_{t+1} - w_t\|^2$ which enforces successive solutions to be close. It can be shown that the solution is obtained after $\mathcal{O}(\frac{1}{\epsilon})$ iterations. [Schmidt, 2009] summarize the various methods to reduce the number of constraints.

The work of [Guzmán-Rivera et al., 2013] suggest to use m cutting planes instead of 1. The wide-spread technique of [Joachims et al., 2009] adds one constraint into the working space, while their work adds more constraints to speed up learning. These constraints have to be highly violated and diverse. The first of the m constraints is the standard most violated constraints from its original formulation. The remaining $(m-1)$ constraints are obtained by finding the $(m-1)$-th best solutions to a Markov Random Field (MRF) problem. On modern multi-core computers, the training time can be reduced exploiting parallelization where the authors in [Chang et al., 2013] accelerate the S-SVM dual formulation. The model update phase and finding discriminative classes for each example is subject to parallelization. We implemented a simplified version of this approach. The calculation of the most violated constraint for each class can be done independently and where we use multiple threads to accelerate this time-consuming step. Structured SVMs have been studied in many domains of machine learning. In object detection it has been successfully applied to person layout recognition [Mittal et al., 2012], object detection in the presence of weak annotations [Blaschko et al., 2010] or deeply deformable part models [Zhu et al., 2010b] just to name a very few.

3.3 Conclusion

We have given an introduction to learning with support vector machines. This technique is the core optimization procedure in our framework. The well known extensions to multi-class comes at the price of a reduced run-time compared to OvA or are not naively applicable to object detection. We follow the idea of having a tree of classifiers as the methods in Sec. 3.2.2 show better results with a hierarchical structure than a flat model as in OvA. But these methods are designed for a classification task. In detection, the additional background class needs special attention and cannot be simply treated as any other object class. We suggest in chapter 4 a novel multi-class detection procedure that extents the structured prediction formulation in Sec. 3.2.3. These modifications are one of our contributions.

CHAPTER 4

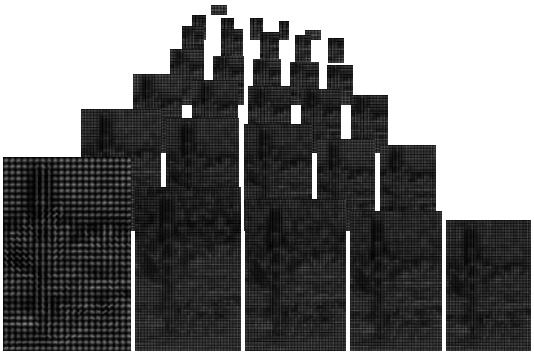
Learning and Detecting Multiple Classes with a Tree of Classifiers

Contents

To detect a certain type of objects in images, we need to learn a model of that object class. This model needs to capture the characteristics of its various aspects and discriminate it to the background objects. We gave an introduction to support vector learning in chapter 3. In this chapter, we present a new model to learn a multi-class object model that captures intra and inter-class variability. The model handles all the classes in a joint framework contrary to a OvA method.

More precisely, we learn a model that consists of a tree of classifiers. These methods are common in image classification as discussed in Sec. 3.2.2 where the classes are ranked among each other. We go one step further and formulate the task of object detection as ranking and classifying an object. To this end, we will extend the structured support vector learning mentioned in Sec. 3.2.3.

(d) input image



(e) feature pyramid

Figure 4.0 – (b) shows the feature pyramid of the image in (a). The interval between two octaves is set to 10.

First, we describe our detection procedure in Sec. 4.1 by introducing the core notations. Next, we introduce our tree of classifiers and how to exactly score an input patch. Afterward, we propose an optimization formulation to learn the filters associated with each classifier in the hierarchy. The structure itself is also learned automatically using only the object annotations. We finish this chapter by showing the results obtained with our proposed framework.

The input image is fragmented into several pieces coming from different resolutions of the image. A *filter* is a function that produces a score to an input image patch. Our framework finds for an input patch the most probable class. The scores of the classes are ranked. In the optimization phase, we assure that the best class gets the highest score. At the same time, if the best confidence is too low, the framework rejects the sample. This mixture of classification and ranking appears both in the detection and hybrid training phase and belongs to our contributions. For k classes, OvA has k detectors respectively filters. When dealing with one class at a time, the term filter equals detector. We augment the number of filters by recursively grouping similar classes into intermediate nodes resulting into a tree of classifiers. The final score of a class is given by the sum of the filter scores lying on the path to its respective leaf node. The detection objective detailed in the next section and the joint optimization of the filters are also new. An advantage is that our system is transparent to the choice of the features.

4.1 Detection Objective

Given an input image, the goal is to associate a label to each position in the image. Let k be the number of object classes to localize. A region can be associated to one of the positive object categories or a background label. The labels belong to a set of k positive classes $\mathcal{Y}^+ \equiv \{y_1, \ldots, y_k\}$ or the background $\mathcal{Y}^- = \{y_{bg}\} = -1$. We re-size the image over various scales. We apply bilinear interpolation for resizing images. One feature map is calculated for each scale. This results in a feature pyramid. An example of it is illustrated in Fig. 4.0. Our approach scores the feature pyramid. That is for each possible position x in the feature pyramid, we calculate an individual score $score_y(x)$

for every class y and an overall score. The letter is defined by the score of the best class:

$$\text{score}(x) = \max_{y \in \{1,\dots,k\}} \text{score}_y(x). \tag{4.1}$$

The predicted object class $\hat{y}$ is given by the final decision function $f : x \to y$ which attributes a label $\hat{y}$ to every x:

$$\hat{y} = f(x) = \begin{cases} -1 & \text{, if score}(x) \leq 0 \\ \arg \text{score}(x) & \text{, otherwise.} \end{cases} \tag{4.2}$$

Our model calculates a score for every class y. If the best score is negative, the hypothesis is classified as negative. Otherwise, we rank the scores and the best score determines its class label. Eq. (4.2) suggests that a region is ranked and classified simultaneously. MCRT unifies *ranking* and *classification* techniques. This is different to previous approaches that only apply one of both techniques.

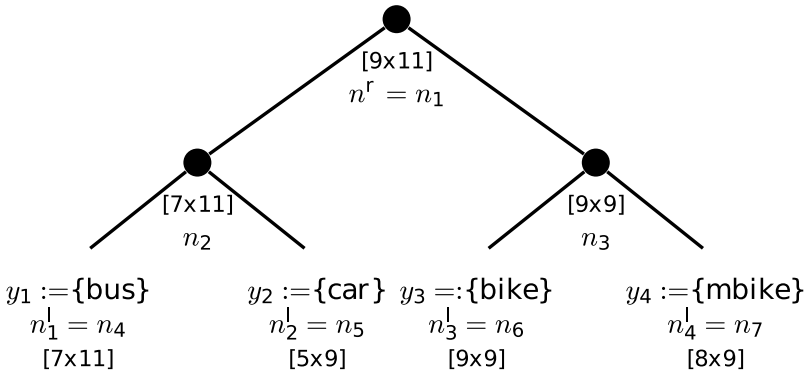
4.2 Detecting Using the Hierarchical Classifier

Our multi-class detection model is defined by a tree of filters. The leaf nodes store the individual classes. An intermediate node is called a *super-class*. It groups several classes associated to its child nodes. These intermediate nodes hierarchically group nearby classes together. We will discuss the tree building process in Section 4.3.1. Classes that are close in the tree are visually more similar than those that are distant. This allows to share visual characteristics among categories in the super-classes.

During detection, each node in the tree scores a region. The sum of these scores lying on one path leading to a class gives the final score of the corresponding class. We retain the best score which determines also its class label. Is this score negative, the background label is assigned to the hypothesis.

We now introduce our notation and detection framework. Formally, a tree $T = \{\mathcal{N}, \mathcal{E}\}$ represents k classes. It consists of $|\mathrm{T}|$ nodes where we designate any node by $n_i,\ i \in \{1, \dots, |\mathrm{T}|\}$. Further, let n_y^l be the leaf node associated to class y. $\text{anc}(n_i)$ contains the set of the ancestors of node n_i including itself and $\text{desc}(n_i)$ is the set of the descendants excluding n_i.

Each node n_i is specifically associated to a subset of classes $\mathcal{Y}_i \subset \mathcal{Y}$. The root filter encapsulates all the classes $\mathcal{Y}^+$ where the leaf nodes n_y^l is associated to a single class $y \in \mathcal{Y}$. Further a child node can only contain a subset of its parent classes. A tree T is further defined by $|\mathrm{T}|$ filters $w = \{w_1, w_2, \dots, w_{|\mathrm{T}|}\}$. The dimensions of the weight vector is mentioned in Sec. 4.3.4. The global weight vector w is the stacked vector of all the node specific weight vectors. These weight vectors are learned during the training process. As we only consider in this work trees with a single parent, the edge $e_{ij} \in \mathcal{E}$ can be written as e_j where i is the parent of j. We can say that a weight vector w_i is associated to an edge e_i or to a node n_i. There is a score $w_i^T \cdot \phi_i(x)$ attributed to an edge e_i with $\phi_i(x)$ being a feature vector extracted by the i-th node. Sec. 4.3.4 gives more detail on how we set $\phi_i(x)$. It exists further an entry score of the tree that is the score $w_1^T \cdot \phi_1(x)$ which is the score given to example x at the root node. We do not speak of weights of the edges as weight often implicate a negative connotation for high



$$\begin{aligned}\text{score}_{y_2}(x) &= w \cdot \Phi^l_{y_2}(x) \\ &= w_1 \cdot \phi_1(x) + w_2 \cdot \phi_2(x) + w_5 \cdot \phi^l_5(x)\end{aligned}$$

Figure 4.1 – Example of 4 classes and the obtained filter dimensions width x height shown in brackets. The process first defines the dimensions of the leaf nodes and iteratively selects the largest dimensions for the super-classes. Further, we see that the score for the car class depends only on its predecessor nodes as defined by Eq. (4.4).

values while scores do not. $\Phi_j(x),\ j \in \{1, \ldots, |\mathrm{T}|\}$ concatenates all the feature vectors where the features of nodes n_i not lying on the path to $n_j,\ n_i \notin \text{anc}(n_j)$, are zeroed. Our notation is visualized in Fig. 4.1. The intermediate score of an example x is the sum of the passed edges up to the i-th node:

$$\text{score}_{n_i}(x) = w \cdot \Phi_i(x) = \sum_{n_j \in \text{anc}(n_i)} w_j^T \cdot \phi_j(x). \tag{4.3}$$

The score depends on the weights of predecessors of node n_i and the corresponding feature vectors. Consequently the final score of a class is defined by the sum of all the scores produced by the edges lying on its path:

$$\text{score}_y(x) = w \cdot \Phi^l_y(x) = \sum_{n_i \in \text{anc}(n^l_y)} w_i^T \cdot \phi_i(x). \tag{4.4}$$

This is depicted in Fig. 4.1. The score of the car class is defined by the {“car”} filter but also its predecessor nodes {“bus, car”} and {“bus, car, bicycle, motorbike”}. Finally, the Eq. (4.2) is applied using Eq. (4.4) to determine the class label. In our formulation, we make use of global and local features. The global features represent several classes while deeper in the tree, the features become more class specific. We focus here on linear classification. We avoid kernel due to its time and memory consuming learning issues and detection rates. We leave these challenging topics for future work. Nevertheless by concatenating features of various levels in the tree, the decision boundary becomes high dimensional. Also, having $|\mathrm{T}|$ linear filters provides a piece-wise linear classification in the feature space. We will show that this leads to more discriminative detection results. Our approach is very similar to OvA as one can imagine to collapse the filters of each path into k class specific detector by adding them together. Also both methods rely on determining the class label based on the maximum score attributed to all

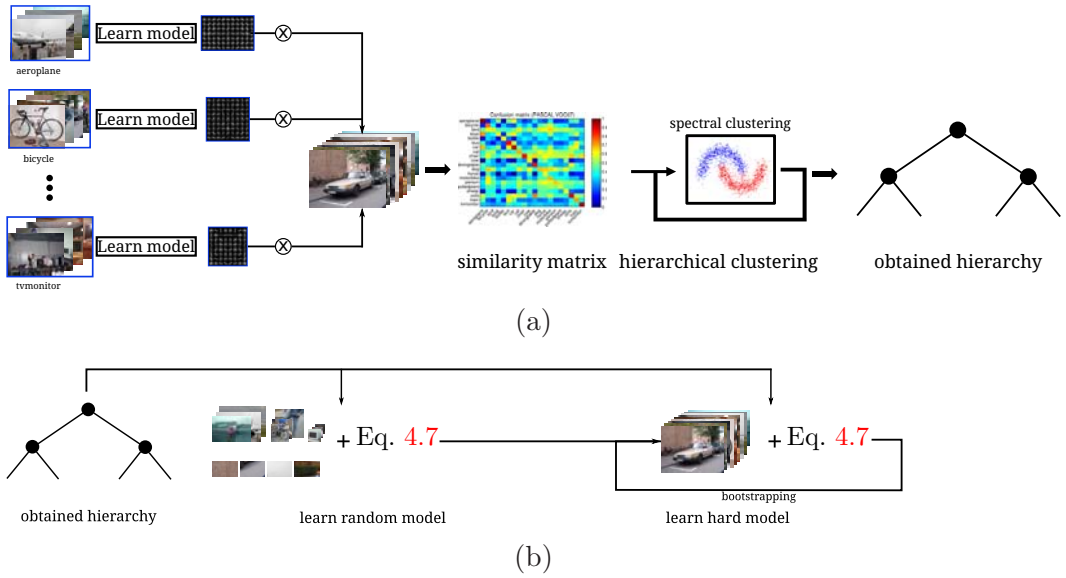


Figure 4.2 – Training pipeline for learning the hierarchical classifier. (a) First k OvA detectors for all the classes are built. They allows to classify the samples of each class and obtain a similarity matrix between object categories. The tree is determined by hierarchically clustering the similarity matrix (Sec. 4.3.1). (b) Knowing the tree, we extract the class features corresponding to the tree and train it using the optimization procedure presented in Sec. 4.3.2. We use the two stages also present in the DPM framework by first building a weak or sometimes called random model and improve it iteratively by relocating the positives and collecting hard negatives.

classes. However, our learning framework allows exactly to learn these intermediate filters which is not possible in a OvA problem formulation where each filter is trained separately or only one weight vector per class is available. In addition, we propose in chapter 5 a novel detection algorithm which improves run-time compared to OvA using our tree model MCRT.

4.3 Learning the Hierarchical Classifier

In this section, we explain the training framework. The first step consists in creating the hierarchy which in our case is a tree. Once the hierarchy is obtained, the learning pipeline follows roughly the same steps as in Sec. 2.2.1 with the difference that we train a hierarchical classifier. Knowing the tree structure, a weak model is trained by extracting positive samples from ground truth data and warping them to model dimensions. The negatives are obtained by randomly sampling the background dataset. Then, we iterate through the dataset again using this weak model and locate positive samples that score high and best match their annotations. This bootstrapping step finds hard negatives support vectors. This is necessary as we use the principles of support vector machines for optimization. The training pipeline is shown in Fig. 4.2. We detail now more the tree creating algorithm and at the heart of the framework, the optimization method.

4.3.1 Creating a Tree

A key influence in the final detection performance is the tree structure. It is essential that filters of the super-classes represent efficiently their underlying classes while discriminating it from all the other classes and the background. To share the most features in a super class, we combine classes having a high confusion. Intuitively, cars and buses or motorbikes and bicycles are easily confused classes. But cars and cows have less characteristics in common.

To this end, we build a detector for every category. These detectors are trained quickly using all positive examples and one iteration of bootstrapping gathering negative examples. Given these detectors, we apply the detector of one class i to the examples of each other class j. The median value of these scores gives the similarity s_{ij}. The examples used for training the detectors are from the training set and the validation set is used for producing the detection scores. The similarity matrix $\mathcal{S} : k \times k$ measures the similarity between all the classes where element is the similarity measure s_{ij} between 2 classes.

We derive the tree structure by hierarchical clustering of the similarity matrix $\mathcal{S}$. We apply spectral clustering [Luxburg, 2007] from top to bottom separating the super-classes into smaller groups of classes until the leaf node is reached. As we want balanced trees for fast run-times (see reason in Section 5.2), we enforce in the k-means step of spectral clustering the number of children at each iteration. The hierarchy is characterized by groups of classes with high intra-class but low inter-class similarity. This technique of creating the tree does also apply to other domains as it makes use of only a similarity matrix. This steps is also part of Fig. 4.2.

There are many other ways of building a taxonomy on the training data. However, it would be out of the scope of this document to deepen this field of clustering. [Razavi et al., 2011] construct the taxonomy based on the sharing matrix. That is once a detection model for each category is learned, they are able to compare these models and calculate their resemblance. It has the convenience of not going anymore through all the training examples while it requires a robust similarity measure between detection models.

[Amit et al., 2004, Fidler et al., 2010] derive a tree structure which improves its cost-to-power ratio by balancing between fast computation of the tree and its detection performance. One can also pursue an approach as in [Salakhutdinov et al., 2011] inspired by cognitive science where a class is attributed to a branch of the tree based on its already attached number of classes and their common similarity.

Inspired by techniques common in text understanding domain where one task consists in assigning a document to a topic, [Sivic et al., 2008, Bart et al., 2008] transfer this knowledge to image classification. They attempt to find a meaningful tree representation of the images using the image appearances and not the semantic knowledge. For example in [Sivic et al., 2008], the images are assigned to topics using hierarchical Latent Dirichlet Allocation introduced in [Blei et al., 2003]. The nodes along a path represent nested topics. These topics can be seen as super-classes.

In general, a hierarchical structures between object categories are obtained by using one of the numerous hierarchical clustering algorithms [Xu and Wunsch, 2005]. In many cases, these greedy algorithms apply recursively a clustering algorithm. One approach is to merge from bottom-up similar instances together which is called agglomerative

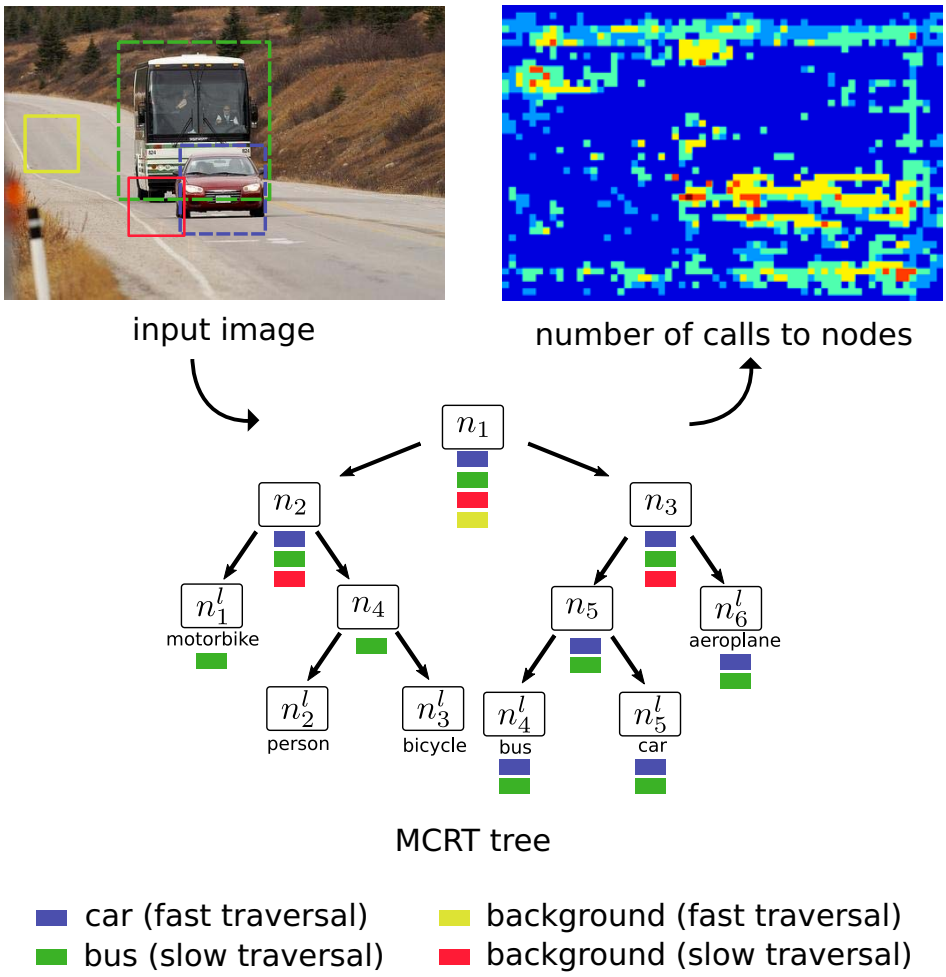


Figure 4.3 – We visualize how different candidate regions shown in the upper left picture traverse our tree. The yellow background region gets rejected early after being scored by filter n_1. The red one uses more filter evaluations to take a final decision. A similar behavior is observed with the red and blue bounding boxes. The heatmap illustrates the relative number of filters evaluated for each position where the blue color means that only one filter was applied.

clustering [Jain and Dubes, 1988]. Another way is to split from top-to-down the instances in different clusters as is the case in our work which is called divisive clustering. Most approaches use either a geometric view of clustering *e.g.* K-means or the spectral method approach. [Blaschko and Gretton, 2009] rely on the statistical view: given the data X, we want to find labels Y such that the statistical dependence between X and Y is maximized. Here, the measure of statistical dependence is HSIC [Gretton et al., 2005]. They obtain a tree structure in an optimization process by constraining the relationship between clusters to be generated by a tree metric.

We opted for the divisive technique based on a similarity matrix first for its simplicity. Second, it makes use of our detection framework to build this matrix. Consequently, the hierarchical clustering algorithm captures the features of the framework.

4.3.2 Optimization Problem

Given the tree hierarchy, we next learn the weight vector w_i of each filter n_i in T. MCRT learns these weights jointly instead of optimizing the filters one after another.

Given only positive classes, our optimization problem is reduced to a ranking problem between positive classes. It can be efficiently solved using S-SVM:

$$\min_{w,\xi_{i(j)}\geq 0} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n^+}\sum_{j=1}^{n^+}\xi_{ij} \tag{4.5a}$$

$$s.t.\ \ \forall y_i \in \mathcal{Y}^+, \forall y_j \in \mathcal{Y}^+\backslash\{y_i\} : w \cdot \delta\Phi_i(y_j) \geq 1 - \xi_{ij} \tag{4.5b}$$

n^+ is the total number of positive examples and $\delta\Phi_i(y) = \Phi^l_{y_i}(x_i) - \Phi^l_y(x_i)$ the difference of feature vectors if region x_i is classified as y_i and y. C and ξ are defined in Sec. 3.1. These constraints rank the score of the correct class and path higher than all the other scores and paths in T. This makes a total of $n^+ \times k$ constraints. For instance, if the region contains a 'motor', the score of the 'motor' class should be higher than those of {'car', 'bikes', 'bus', 'aeroplane', 'person'} given the tree in Fig. 4.3. As the score consists of the sum of the intermediate filter scores, we thus ensure that the right path is given the greatest value.

We now need to distinguish between foreground and background objects. A naive approach would be to use a generic detector [Alexe et al., 2010] to localize object categories of interest. Upon he responses, one could apply a multi-class classification system to score highest the right class. This has the disadvantage of propagating the errors of the first module into the second one incapable of discriminating background samples.

We could only use classification constraints to discriminate the background samples. The scores of the positive classes would be positive and vice versa the scores of the background classes would be negative. The result would be the following optimization problem:

$$\min_{w,\xi_i\geq 0} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n^+ + n^-}\xi_i \tag{4.6a}$$

$$s.t.\ \ \forall i = 1 \ldots n^+ + n^- : \text{sign}(y_i) w \cdot \Phi^l_{y_i}(x_i) \geq 1 - \xi_i \tag{4.6b}$$

with n^- the number of negative training examples. The scores attributed to positive classes are no longer comparable. The inference does not assure that the best path scores the highest value. We conclude the necessity to combine both types of constraints namely classification and ranking.

Considering our detection framework, the background class cannot be modeled due to its huge visual variability and cannot be treated as a leaf node in our tree model.

Algorithm 2: Solving the problem in Eq. (4.7) via its 1-slack formulation in Eq. (4.8). Only one constraint at a time is added to the working set $\mathcal{W}$. The optimization time over the working set is considerably reduced.

$\mathcal{W} \leftarrow \emptyset$
repeat
  foreach *example* $i = 1 \dots n$ **do**
    $\bar{y}_i = \arg\max_{y \in \mathcal{Y}^+} 1 + w \cdot \Phi^l_y(x_i)$
  $\sum_{i=1}^{n^+} w \cdot \Phi^l_{y_i}(x_i) - \max(w \cdot \Phi^l_{\bar{y}_i}(x_i), 0) - \sum_{i=1}^{n^-} w \cdot \Phi^l_{\bar{y}_i}(x_i) \geq 1 - \xi \rightarrow \mathcal{W}$
  Apply quadratic program solver to $\mathcal{W}$
until *$(w, \xi + \epsilon)$ fulfills the constraints in 4.8c*

MCRT further adds classification constraints to the ranking problem in 4.5:

$$\min_{w, \xi_i \geq 0, \xi_{ij} \geq 0} \frac{1}{2}\|w\|^2 + C\left(\sum_{i=1}^{n^+}(\xi_i + \sum_{j=1}^{n^+}\xi_{ij}) + \sum_{i=1}^{n^-}\xi_i\right) \tag{4.7a}$$

$$s.t. \tag{4.7b}$$

$$\text{Ranking:} \tag{4.7c}$$

$$\forall i \text{ with } y_i \in \mathcal{Y}^+, \forall y_j \in \mathcal{Y}^+ \backslash \{y_i\} \quad : w \cdot \delta\Phi_i(y_j) \geq 1 - \xi_{ij} \tag{4.7d}$$

$$\text{Classification:} \tag{4.7e}$$

$$\forall i \text{ with } y_i \in \mathcal{Y}^+ \quad : w \cdot \Phi^l_{y_i}(x_i) \geq 1 - \xi_i \tag{4.7f}$$

$$\forall i \text{ with } y_i \in \mathcal{Y}^-, \forall y_j \in \mathcal{Y}^+ \quad : -w \cdot \Phi^l_{y_j}(x_i) \geq 1 - \xi_i, \tag{4.7g}$$

Our problem formulation uses two types of constraints aiming at learning a stacked weight vector w. The optimization problem ranks a foreground example among all classes using constraint (4.7d). A background example is classified based on negative constraint (4.7g). This leads to $n^+ \times \mathrm{k} + n^+ + n^-$ constraints during optimization.

4.3.3 Practical Implementation

The constraints in the optimization problem 4.7 depend directly on the number of examples and classes. In case of a multi-class dataset, these both values can be large. Our objective is to provide a training algorithm reducing memory consumption and optimization time. We use an approach highly inspired by [Joachims et al., 2009]. We mention two approaches that we have found especially useful depending on the used solver. These approaches depend on the cutting plane method. Instead of optimizing all the constraints, they iteratively refine a *working set* of the constraints.

1-slack Formulation

The idea is to replace all the slack variables with a single one shared across all the constraints. Let $\bar{y}_i \in \mathcal{Y}^{+n}$ be any possible positive label. Maximizing the term $w \cdot \Phi^l_{y_i}(x_i) - \max(w \cdot \Phi^l_{\bar{y}_i}(x_i), 0)$ forces whether the positive example x_i to be clas-

sified as positive or increases the difference between the right and the wrong classes. Simultaneously for a background example x_i, we wish to minimize $w \cdot \Phi^l_{\bar{y}_i}(x_i)$ for all the negative samples. This leads to the following optimization problem:

$$\min_{w,\xi \geq 0} \frac{1}{2}\|w\|^2 + \frac{C}{n}\xi \tag{4.8a}$$

$$s.t. \qquad \forall(\bar{y}_1, \ldots, \bar{y}_n) \in \mathcal{Y}^{+^n} : \tag{4.8b}$$

$$\sum_{i=1}^{n^+} w \cdot \Phi^l_{y_i}(x_i) - \max(w \cdot \Phi^l_{\bar{y}_i}(x_i), 0) - \sum_{i=1}^{n^-} w \cdot \Phi^l_{\bar{y}_i}(x_i) \geq 1 - \xi. \tag{4.8c}$$

For an example (x_i, y_i), all the other positive classes $\bar{y}_i$ are taken into consideration. Now, we have a single slack variable. However, the number of constraints increases exponentially with the number of training examples $n = n^+ + n^-$. The problem in Eq. (4.8) has a sum for every possible combination of the values $\bar{y}_i \in \mathcal{Y}^+$, $i = 1 \ldots n$. [Tsochantaridis et al., 2004] show that Eq. (4.8) and Eq. (4.7) reach the same optimal parameters knowing that $\xi = (1/n)\sum_i \xi_i$ and are thus an equivalent formulation. We have a total of $|\mathcal{Y}^+|^n = |k|^n$ constraints. The problem would be infeasible to handle with standard solvers using ever increasing datasets. Inspired by [Joachims et al., 2009], the optimization problem is solved efficiently using cutting plane algorithms.

We use a subset of the constraints and iteratively refine this working set. At each iteration, a solver is used to optimize the problem. The algorithm is outlined in algorithm 2. We start with an empty list of constraints. Before taking the sum, we consider each example of the sum separately. For each example, instead of taking into account the elements of the sum generated by comparing to all the other classes, we only consider the most similar one. Finally, all the individual parts are summed giving one constraint. This new constraint is added to the working set $\mathcal{W}$. A quadratic program solver is applied on $\mathcal{W}$. If the new solution does not fulfill the constraint of Eq. (4.8c) up to a precision ϵ, the procedure is repeated.

Standard computers possess multiple cores and we use this fact to accelerate our learning phase. A time-consuming step is the calculation of each element in the sum which increases with the number of data n:

$$\sum_{i=1}^{n^+} w \cdot \Phi^l_{y_i}(x_i) - \underbrace{\max(w \cdot \Phi^l_{\bar{y}_i}(x_i), 0)}_{\text{multi-threaded}} \qquad \text{and} \qquad \sum_{i=1}^{n^-} \underbrace{w \cdot \Phi^l_{\bar{y}_i}(x_i)}_{\text{multi-threaded}}$$

The extraction of the feature vector $\Phi_y(x_i)$ takes a considerable amount of time. This is even more important due to the way we save and load features for the structured learning module to reduce the memory consumption. We do not save the features directly in its final representation but crop out regions from a more global feature vector surrounding all the features to compose the features for each class. The calculation of the elements in the sum can be done independently and summed together at the end. We parallelize this calculation using OpenMP and add the n elements once all threads have finished.

Algorithm 3: A coarse-to-tine approach to solve the problem in Eq. (4.7). First, the most violated constraints are added to the working set $\mathcal{W}$ and an intermediate model is learned. This model is iteratively refined by adding less violated strongly constraints to $\mathcal{W}$. The basic idea is to locate support vectors early in the training stage.

```
epsilon = CONSTANT
while epsilon ≥ ϵ do
    constraints = 0
    epsilon = max (epsilon/2, ϵ)
    while no more constraints are violated more than ϵ do
        foreach example i = 1 . . . n do
            ȳ_i = find class that is most similar to y_i
            if constraint 4.7d, 4.7f or 4.7g is violated more than ϵ then
                constraints++
                Add this constraint to working set W
            if constraints ≥ THRESHOLD then
                QP solver ← W
```

Coarse-to-fine Solution

The approach can be seen as a coarse-to-fine manner that iteratively finds the optimal solution. This greedy technique is shown in algorithm 3. We start with a high value for epsilon which allows us later to find the hardest constraints. We refine the value of epsilon iteratively with intermediate solutions for weight vector w. In an inner loop, the algorithm goes over all the examples i and finds the most similar class $\bar{y}_i$ to (x_i, y_i). The resulting constraint is the most violated one. It is only added if the constrained does not fulfill the desired precision up to epsilon. Once enough constraints are gathered, the problem is solved with a quadratic program solver. We use solvers in primal and dual form. These steps are repeated until not enough constraints are violated and the desired precision of ϵ is reached.

4.3.4 Filter Dimensions

A filter is an unit which transforms an input to an output value. Here, a filter associates a score to an input region. In the OvA case, a filter can be the detector itself as it associates a confidence of the class to a position in the image. We have $|\mathrm{T}|$ nodes in the tree. Each node n_i is associated with a weight vector w_i which is a part of the more global weight vector w. We treat w_i as a filter which attributes a *partial score* to some features extracted out of the image region x. This score is only a partial element in the sum of the final score defined by Eq. (4.3) and is the product of $w_i^{\mathrm{T}} \phi_i(x)$.

The filter $w_i \in \mathbb{R}^{W_i \mathrm{x} H_i \mathrm{f}}$ has a width of W_i, height H_i and f number of features per cell. We represent these weight vectors in this way mainly for getting a nicer illustration of the learned weights. The third dimension f is fixed by the object descriptor *e.g.* to $f = 31$. This length corresponds to the HOG feature vector presented in [Felzenszwalb

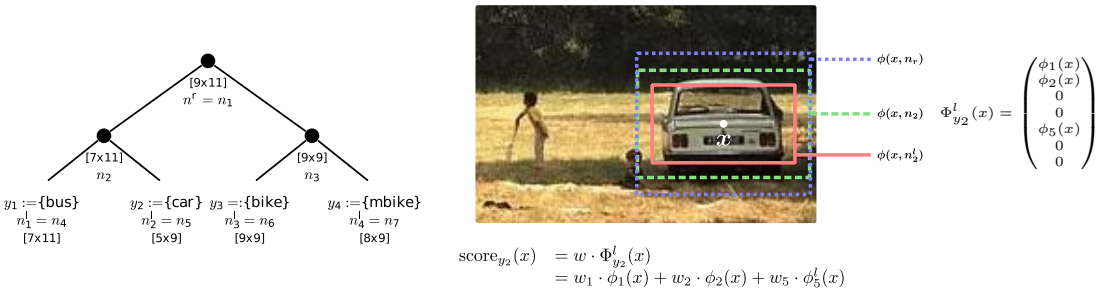


Figure 4.4 – The left part shows an example tree. On the right, the regions where the features are extracted from for the filters n^r in blue, n_2 in green and n_2^l in red are illustrated. The region associated to the root filer encapsulates the regions belonging to its successor nodes as its dimensions are set to the largest values of its own child nodes' dimensions.

et al., 2008a]. The two other parameters capture the dimensions of the object samples from the set of classes modeled by the super-class. We build these parameters from bottom-to-top starting with the leaf nodes. The dimensions of the leaf nodes are constructed differently than those of the super-classes.

The dimensions of the leaf nodes are set as mentioned in Sec. 2.2.1. We compute a histogram of all the aspect ratios $\text{AR}(x_j)$ $j \in \{1, ..., n^+\}$ of an object class and choose the mean value $\text{AR}(w_i)$. Next, we sort all the samples' surfaces and set $\text{m}(w_i) = W_i \text{x} H_i$ so that it is smaller than 80% of the samples' annotations. The final parameters are obtained knowing the aspect ratio and surface:

$$W_i = \sqrt{\frac{\text{m}(w_i)}{\text{AR}(w_i)}}$$
$$H_i = W_i \text{AR}(w_i)$$

This process is not repeated for the super-classes. The dimensions of the super-classes depend on its child nodes' dimensions. A node n_i picks the maximum width and height of its children's dimensions. This ensures that the node is sufficiently large to contain its classes examples:

$$W_i = \max_{n_j \in \text{desc}(n_i)} W_j$$
$$H_i = \max_{n_j \in \text{desc}(n_i)} H_j$$

The filter w_i is multiplied with a feature vector $\phi_i(x)$ extracted around the region x. In most datasets, objects are provided in images and x is the center location of this object. During detection, x stands for the various positions in the feature pyramid and is the center at each point. The features at each node n_i are extracted around the center x and the size of the cropped rectangle region is defined by its filter dimensions W_i and H_i. It is to note that the features extracted by the child nodes are a subset of their parent's features. We can extract many types of features and we experimented with HOG and DPM as object descriptors. Fig. 4.4 shows on an example the determined model dimensions and the notation of the nodes. The right part of the figure shows the selected areas around a region x defined by three nodes.

4.4 Results

We evaluated our approach on two datasets PASCAL VOC'07 and PASCAL VOC'10. We introduced these two datasets in Sec. 2.1. The evaluation protocol is provided within the dataset. The PASCAL VOC'10 dataset does not contain the test annotations and the evaluation protocol forces to evaluate the algorithm via an on-line platform. We intended further to evaluate on VOC'10 dataset offline for reasons that get more clear in Sec. 5. To this end, we used the validation part of the dataset as the test images and split the training part equally into a new training and validation part. We call this new dataset as PASCAL VOC'10*offline*. The used features are the histogram of orientated gradients of [Felzenszwalb et al., 2010a] which we used for its implementation simplicity and popularity. It allows to quickly validate our framework. The quadratic program solver is the one developed in [Joachims, 1999a] and available under [Joachims, 1999b]. It is a SVM solver which optimizes the problem in the dual form introduced in Sec. 3.1.2.

4.4.1 Overall Performance

We selected 6 settings for different values of the number of classes $k = \{2, 4, 6, 8, 10, 20\}$ to get a more representative evaluation depending on k. We fed the classes in the following order into our system where for a specific value of k, we choose the first k entries: {'bus', 'bicycle', 'motorbike', 'car', 'aeroplane', 'person', 'cow', 'horse', 'dog', 'cat', 'bird', 'boat', 'bottle', 'chair', 'diningtable', 'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor'}. For example for $k = 4$, we selected the classes: {'bus','bicycle','motorbike','car'}. Among these 20 classes of the PASCAL VOC datasets are ones with a strong affinity such as {'bicycle','motorbike'}, or expecting common features such as {'bus', 'car'} or simply expecting having little in common *e.g.* {'person','tvmonitor'}. We chose the classes in this order to analyze whether our algorithm performs well in presence of unrelated classes.

We validate different aspects of our algorithm to three designed algorithms. The first method is the One-versus-All technique OvA where the detectors of each class are learned independently from each other The output scores of each class are calibrated using [Platt, 1999] as detailed in Sec. 2.2.1. We consider it as our baseline to which we compare our systems. The MCR model which stands for multi-class classification and ranking. It trains k decision hyperplanes, one for each class, using our framework but without a tree. Contrary to OvA, the weight vectors are trained jointly but then applied sequentially exactly as with OvA. During the simultaneous training process, it mixes classification and ranking constraints. The *e*MCRT model is the one introduced in this chapter. It applies exhaustively a tree of classifiers to determine the optimal object label. It extends MCR in that it uses a tree structure and not a flat hierarchy with no interaction between the classes. It permits to get an understanding on the importance of a hierarchical knowledge during detection. The e stands for exhaustive as every node in the tree is applied. We accelerate the inference time in the chapter 5.

The evaluation criteria are the average precision (AP), mean average precision (mAP) and detection speed. The average precision is a value that describes the precision/recall curve and is the mean precision at equally spaced recall values. The mAP

is simply the average of all AP over the classes. Further details can be found in [Everingham et al., 2010]. We normalize the speed to the run-time of OvA for each setting in k as it constitutes our baseline.

PASCAL VOC 2007

The results are shown in Table 4.1. The MCR method performs better than the OvA for all settings in k. Learning all classes in a joint optimization framework allows to find stable decision hyperplanes for the classes. The improvement is not constant and is best for $k = 20$ and modest for $k = 4$. The hierarchical detector *e*MCRT delivers the best results by outperforming the OvA and MCR for all values of k. The execution

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	21.4	1x	22.3	1x	17.8	1x	16.1	1x	12.9	1x	8.8	1x
MCR	23.7	1x	23.0	1x	20.4	1x	17.4	1x	14.7	1x	12.2	1x
*e*MCRT	**25.7**	0.66x	**25.4**	0.55x	**21.8**	0.51x	**20.3**	0.51x	**17.2**	0.51x	**13.1**	0.4x

Table 4.1 – Performance evaluation for PASCAL VOC 2007 dataset.

time is the same for OvA and MCR. Both methods apply the k detectors one after another and their filter dimensions have the same sizes. This is not true for the MCRT which has many more filters in the tree. The evaluation time takes twice as long. This can be expected using Eq. (5.9) and choosing $b = 2$ as we use balanced binary trees. The detection with the tree takes definitely more time than with the basic OvA method. The improved performance sacrifices the run-time of the current algorithm. This issue is addressed in the chapter 5.

PASCAL VOC 2010

The MCR and OvA methods perform closely or MCR has lower values for mAP than OvA as shown in Table 4.2. Contrary to the VOC'2007 dataset, it did not outperform OvA. This did not affect the good performance of our framework *e*MCRT. It outperforms both methods for all values of k. The best improvement is achieved for $k = 10$ (+1.18%) and the smallest one for $k = 2$ (+1.03%). The run-time follows the

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	20.0	1x	15.2	1x	15.9	1x	12.6	1x	11.9	1x	7.7	1x
MCR	18.1	1x	11.4	1x	14.7	1x	11.8	1x	9.42	1x	7.5	1x
*e*MCRT	**20.6**	0.67x	**15.9**	0.55x	**16.9**	0.59x	**13.6**	0.55x	**14.0**	0.56x	**9.0**	0.48x

Table 4.2 – Performance evaluation for PASCAL VOC 2010 dataset.

same rule as for VOC 2007. We note a loss in speed of MCRT compared to OvA of around 50%.

PASCAL VOC 2010*offline*

Very similar to the VOC 2007 dataset, the experiments summarized in Table 4.2 show that MCR achieves comparable results to OvA. The results are worse for small values of $k = \{2, 4\}$ but superior for $k = \{6, 8, 10, 20\}$. MCRT outperforms again both approaches for every setting. The best relative improvement is for $k = 20$ with $+1.47\%$ and the smallest one is for $k = 4$ with a very small improvement. Here again, the speed

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	30.5	1x	22.7	1x	21.2	1x	17.0	1x	13.4	1x	9.8	1x
MCR	25.8	1x	19.8	1x	23.2	1x	18.6	1x	16.1	1x	13.0	1x
*e*MCRT	**31.4**	0.67x	**22.8**	0.56x	**24.5**	0.48x	**20.8**	0.47x	**19.0**	0.48x	**14.4**	0.48x

Table 4.3 – Performance evaluation for PASCAL VOC 2010*offline* dataset.

of MCRT decreases over the OvA case. We continue in Sec. 5 improving the speed of MCRT. Naturally, the run-time of MCR and OvA are still the same.

Dependence of mAP on k

Does the gain in mean average precision mAP depend on the number of classes k? This relative gain is computed as follows:

$$\text{gain}(k) = \frac{\text{mAP}_{e\text{MCRT}}\,(k) - \text{mAP}_{\text{OvA}}(k)}{\text{mAP}_{\text{OvA}}(k)},$$

where $\text{mAP}_{\text{ALG}}(k)$ stands for the mean average precision value for a setting k and an algorithm ALG *e.g.* *e*MCRT . Fig. 4.5 plots the relative gain in mAP over the number of classes for the three datasets. The gain increases with k for the 2007 and 2010 edition. For the modified dataset 2010*offline* this improvement stagnates with $k \geq 10$. In general, we note that the relative mAP improves with k. The tree benefits from this increased number of filters depending on the depth of the tree. It allows to apply many more linear filters. Further, the super-classes group many more classes together and share their features. Also, the tree building algorithm has more choices to find and cluster similar classes into a branch of the tree and thus learn more meaningful super-classes respectively filters.

Per Class Evaluation

Fig. 4.6 depicts the relative improvement of the *e*MCRT system compared to OvA for the two datasets VOC 2007 and VOC 2010*offline*. The tests were done for $k = 20$ classes. The classes are sorted by their number of training samples in decreasing order. Some classes that had a mean average precision of 0 for OvA, have achieved with *e*MCRT a better score. These classes are not plotted in Fig. 4.6.

For PASCAL VOC 2007, we note that the classes with little number of examples *e.g.* {'cow','sofa'} improve in performance. This is different for VOC 2010*offline* dataset where most classes benefit similarly in performance. Some classes show a decrease in

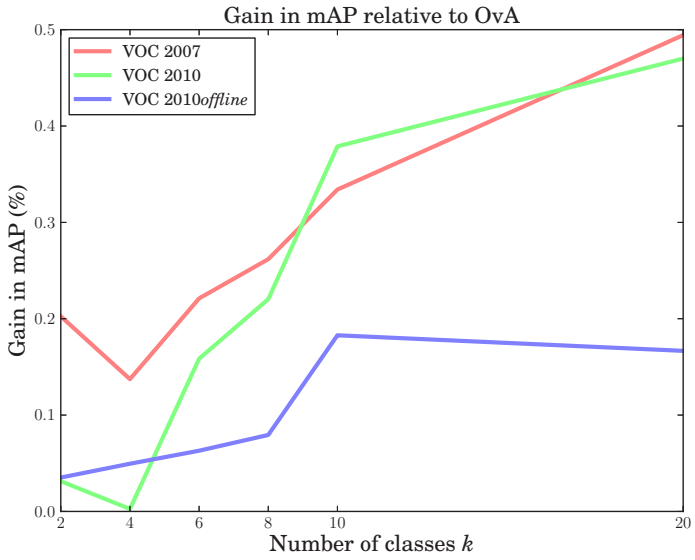
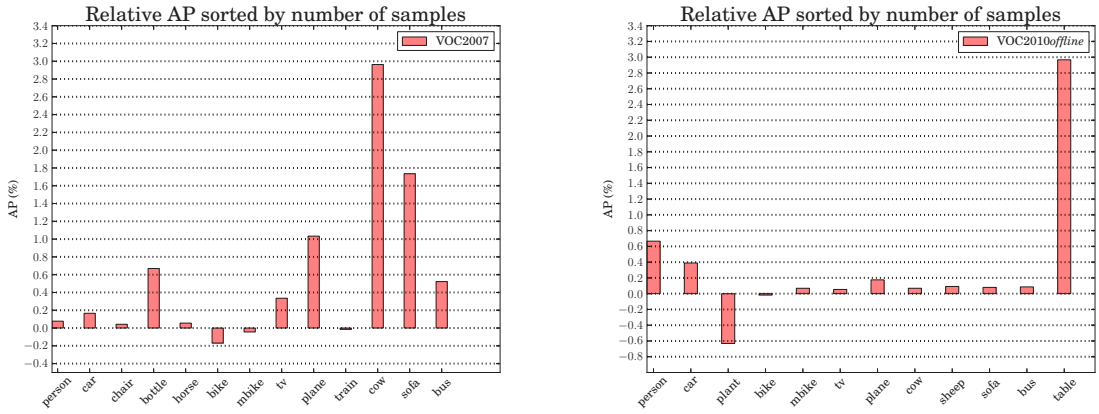


Figure 4.5 – Dependence of the relative gain in mean average precision on the number of classes. The results are reported for the PASCAL VOC datasets. The mAP of the *e*MCRT method is computed relative to the mAP of OvA. The gain increases with the number of classes for VOC 2007, VOC 2010 and VOC 2010*offline* for $k \leq 10$.



(a) PASCAL VOC 2007

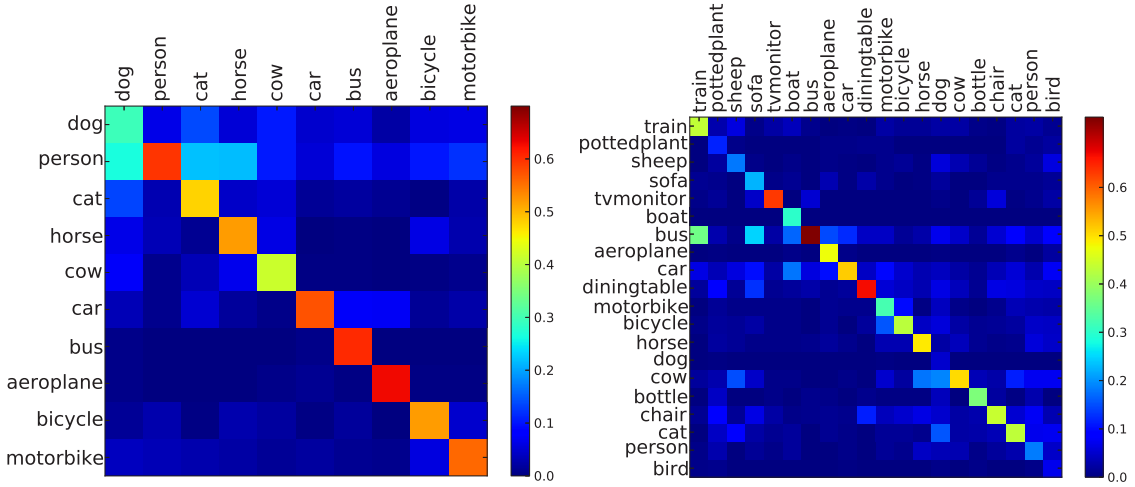
(b) PASCAL VOC 2010*offline*

Figure 4.6 – Relative improvement of detection performance measured in average precision for *e*MCRT and OvA when learning 20 classes. The class names are sorted in decreasing order of number of training samples. Those classes having a mAP of 0 for OvA are not plotted but did in general improve in performance with *e*MCRT .

performance *e.g.* {'bicycle'} class in VOC 2007 or the {'pottedplant'} category in VOC 2010*offline*. The tree improves the accuracy for most classes on both datasets.

Illustration of trees for Pascal VOC 2007

The tree is obtained based on hierarchical clustering of the similarity matrix between class detectors. We show the trees for the PASCAL VOC 2007 dataset in Fig. 4.8 for each value of tested k. For $k = 4$, the classes are grouped as semantically expected.



(c) $k = 10$ (d) $k = 20$

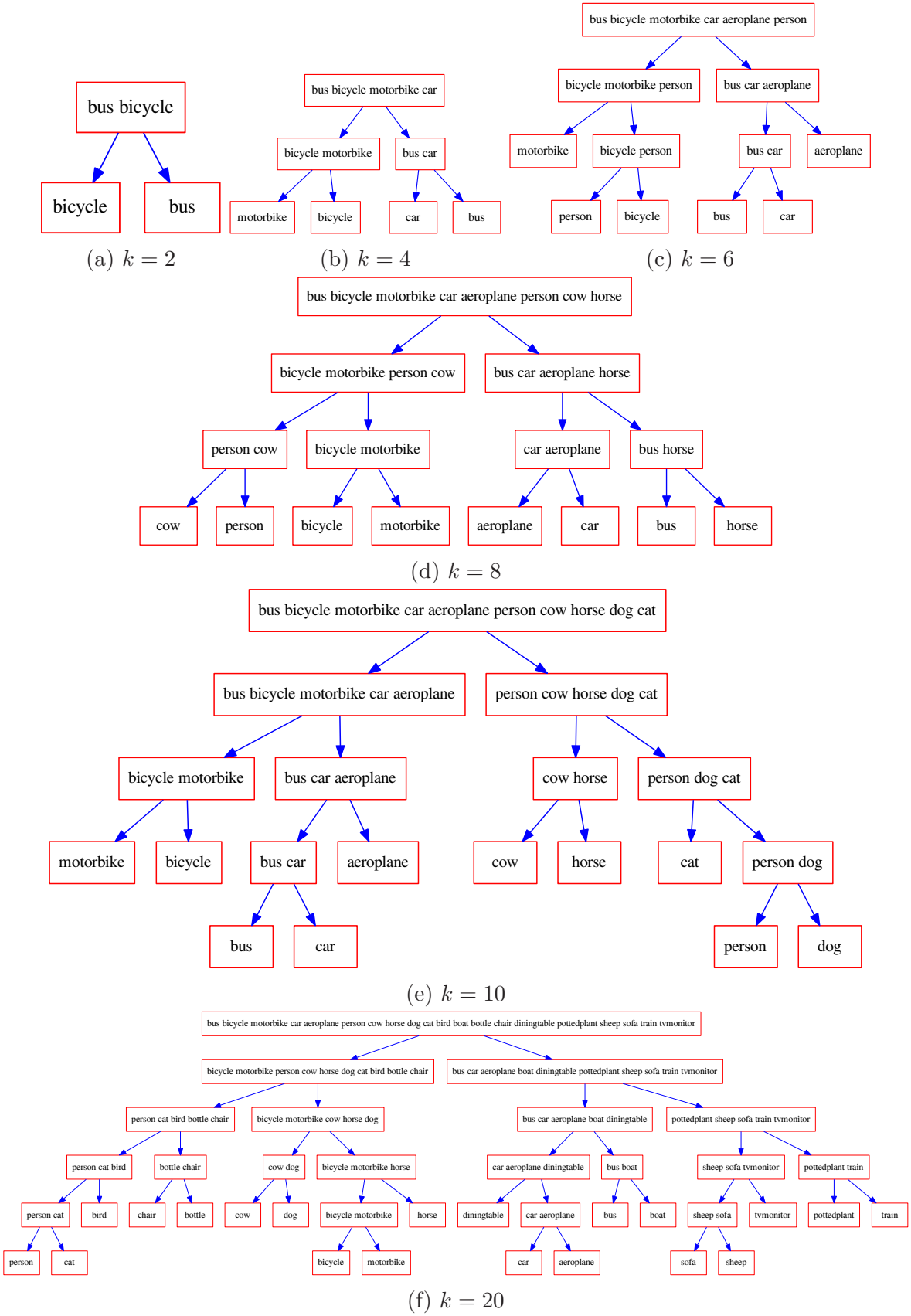
Figure 4.7 – Illustration of the similarity matrix between object categories. The diagonal row shows the accuracy to recognize objects of its own class. Otherwise, the color indicates the confusion between two classes. The ordering of the classes in the similarity matrix corresponds to distances between their respective leaf nodes in the tree.

For the next test $k = 6$, the person and bicycle class are often confused as they have overlapping features. Many images in the dataset show persons standing in front or riding a bicycle. However, in the case of $k = 8$, the classes share sometimes semantically unrelated super-classes mostly due to their resemblance in the feature space. We find more coherent ordering of the classes for $k = 10$. This does not hold again for $k = 20$ where most groups are semantically not related.

We show as well the similarity matrix for $k = \{10, 20\}$ in Fig. 4.7. The classes are ordered based on their distance in the hierarchy. Nearby classes in the illustration are also neighbors in the tree structure. It allows us to better recognize the confusion between the object categories. The diagonal elements show the effectiveness of detecting its own instances. For instance, the bus class detector attains a high score for its own samples however it is easily confused with some other classes such train or sofas. The confusion between a train and a bus is shown in Fig. 4.8 in the category of images surrounded by a light blue box and the overall red box.

Sample results

We show in Fig. 4.9 detection results on some images where the results are satisfying or not working appropriately. We used *e*MCRT learned on PASCAL VOC 2007. The results in the green box are successful detections where as the results shown in the red box are erroneous. We note from these images, that errors are due to amongst others confusion between object categories, contours that look like the resulting object class or the returned bounding box does not meat the overlap requirements. For instance a (tv)monitor can be easily mistaken by many rectangular shaped objects.



(a) $k = 2$ (b) $k = 4$ (c) $k = 6$

(d) $k = 8$

(e) $k = 10$

(f) $k = 20$

Figure 4.8 – Illustrations of the trees for the PASCAL VOC 2007 challenge. The tests were done for different number of classes $k = \{2, 4, 6, 8, 10, 20\}$.

Figure 4.9 – Some detection results for PASCAL VOC 2007 test images using our *e*MCRT framework. We show the results obtained with the models for $k = 4, 6, 8, 20$ as an illustration. The top images surrounded by a green bounding box are correct detections. The bottom ones surrounded by a red bounding box show wrong detections. (best viewed in color and high zoom).

4.4.2 Impact of Related or Unrelated Classes on Performance

In practice, the aimed object classes are defined by the application. In urban scenarios these could be *e.g.* car, bicycle, motorcycle or van to just name a few. Sometimes, these classes are not necessarily semantically close *e.g.* boat and a dolphin in a ocean context. Coherent classes are those that have common features or are semantically close. Incoherent classes on the other hand have less common properties. In this section, we experiment with coherent and incoherent classes to see its relevance in the performance of *e*MCRT .

k	coherency	aeroplane	bicycle	bottle	bus	car	cow	dog	horse	motorbike	person	sheep	sofa	tvmonitor
2	incoherent	✓								✓				
	coherent		✓							✓				
4	incoherent	✓									✓		✓	✓
	coherent						✓		✓		✓	✓		
6	incoherent	✓		✓				✓		✓	✓		✓	
	coherent		✓		✓	✓	✓		✓	✓				

Table 4.4 – We test the influence of the choice of the classes in the improvement of *e*MCRT over OvA. The tests were done with three different number of classes k. For each k, we build a multi-class detector for coherent and incoherent classes. The table shows the classes that are grouped together for each test experiment.

We chose three settings $k = \{2, 4, 6\}$ and two types of coherency {'coherent', 'incoherent'}. This makes a total of 6 experiments. For each setting and coherency type, we trained an *e*MCRT detector and a OvA detector. We report the results in mAP after evaluating the learned detectors on the test set. The chosen classes depend on the setting and the domain of coherency. The selected classes for each task are summarized in Table 4.4. For instance, we chose for the coherent experimental setting with $k = 4$ the following classes: {'cow', 'horse', 'person', 'sheep'}. The dataset is PASCAL VOC 2007. We based our choice on the similarity matrix obtained for this dataset with $k = 20$. We aim to understand how the coherency impacts the improvement of our system in mAP over OvA. Should one favor close classes over unrelated ones when building the hierarchical detector compared to the traditional OvA? Can our system handle incoherent classes? The comparison of the scores between the detectors *e*MCRT and OvA for each experiment allows to understand the influence of the choice of the classes on the final performance of *e*MCRT .

The results are reported in Table 4.5. The values for mAP are obtained by applying each trained detector to the test set. The metric gain is relative to OvA to see the influence of *e*MCRT over the results of OvA. *e*MCRT improves the average detection performance in all setups except for $k = 2$ with a coherent tree where the mAP is slightly inferior. The improvements are much better if the classes have more common characteristics as is the case for $k = \{4, 6\}$. The improvement of *e*MCRT in the incoherent case is smaller than when using coherent classes *e.g.* an increase of +5.2% compared to +8.7% for $k = 6$. The hierarchical classifier improves best if the classes are related. *e*MCRT is superior to OvA for every setting in k when using incoherent classes. Thus, it is possible to use a mixture of unrelated classes such as {'aeroplane','sofa'} depending on the requirements of the application without losing in performance compared to OvA.

coherency	incoherent		coherent	
method	OvA	*e*MCRT	OvA	*e*MCRT
mAP	17.4	20.3	23.4	22.8
gain (%)		**+17**		-2.6

(a) k=2

coherency	incoherent		coherent	
method	OvA	*e*MCRT	OvA	*e*MCRT
mAP	15.6	17.6	10.2	13.5
gain (%)		+12.8		**+32**

(b) k=4

coherency	incoherent		coherent	
method	OvA	*e*MCRT	OvA	*e*MCRT
mAP	13.5	14.2	19.5	21.2
gain (%)		+5.2		**+8.7**

(c) k=6

Table 4.5 – Results obtained for testing the importance of coherency in a tree. Both trained detectors are applied in each experiment to the test set returning a mAP. The gain is calculated relative to OvA.

4.4.3 How Good is the Tree?

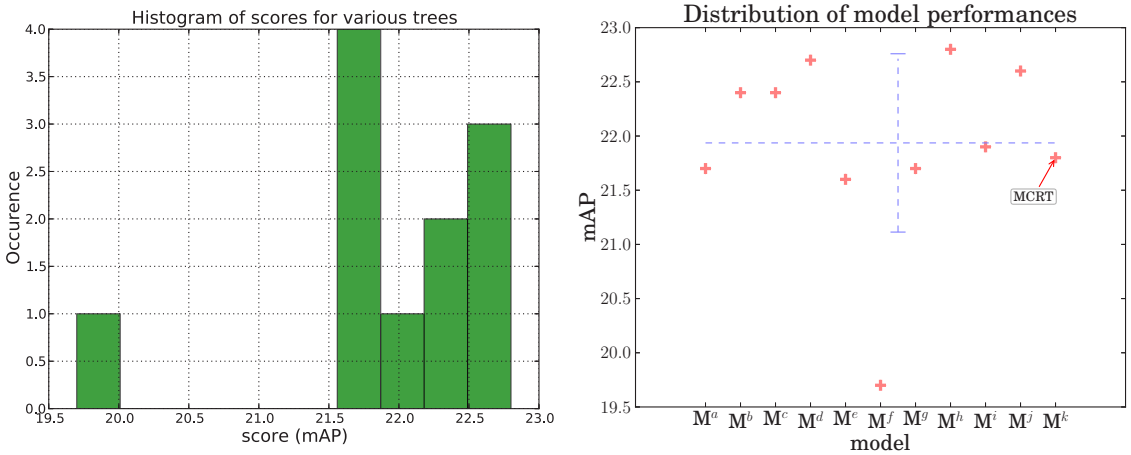
The proposed system builds and learns filters located in a tree structure which is obtained with a hierarchical clustering as detailed in Sec. 4.3.1. The hierarchy is built using a greedy method and influences the average performance of MCRT. We experiment in this section with different types of trees for a fixed value of number of classes k. By varying the tree model, we can compare the detection performance (mAP) of our learned tree with the other hand-designed structures.

We picked the following classes for the evaluation {'aeroplane', 'bicycle', 'bus' , 'car', 'motorbike', 'person'} and fixed $k = 6$. We used our algorithm to build three trees and designed further 10 other tree structures by hard coding it in the program. This makes a total of 11 trees that are trained and evaluated on PASCAL VOC 2007 dataset with the exact same parameters. We tried to build trees where the super-classes group semantically meaningful object categories or unrelated classes. We focused mainly on binary trees as our fast tree traversal algorithm works quickly with these types of hierarchies.

Our learned hierarchy is shown in Fig. 4.12k. It is semantically meaningful and reflects the relationship of objects found in images. The classes {'bus', 'car'} appear in the same context mostly on roads and have a similar shape. The {'aeroplane'} class is combined with the vehicle super-class. On the other side of the branch, we find the classes {'bicycle', 'person'} sharing the same parent. This may seem unreasonable at the first glance. Intuitively, we would cluster {'bicycle'} and {'motorbike'} together. Having a closer look at the images in the dataset, we noticed that many pictures are taken by people photographing each other. Persons can appear standing upright

tree	aeroplane	bicycle	bus	car	motorbike	person	mAP	rank
M^k = MCRT	19.0	25.5	21.6	31.3	18.5	14.8	21.8	(7)
M^a	**21.3**	25.0	18.2	33.5	18.6	13.7	21.7	(8)
M^b	19.9	26.2	21.4	32.2	19.8	14.7	22.4	(5)
M^c	19.4	25.8	18.8	34.2	19.7	16.9	22.4	(4)
M^d	18.2	**28.6**	23.1	33.8	18.6	14.0	22.7	(2)
M^e	19.3	25.7	**24.6**	25.3	18.1	6.8	21.6	(10)
M^f	13.7	23.6	19.8	33.7	15.1	12.4	19.7	(11)
M^g	20.2	26.2	19.0	33.8	20.6	10.4	21.7	(9)
M^h	18.0	24.0	24.3	33.5	**22.2**	14.6	**22.8**	(1)
M^i	18.6	25.8	18.0	**34.3**	20.1	14.5	21.9	(6)
M^j	18.3	27.1	22.9	33.0	19.6	**15.0**	22.6	(3)

Table 4.6 – Results for various tree models. Our result is shown on the line of MCRT. We experimented with 10 other trees which are illustrated in Fig. 4.11. The average performance of the trees is 21.94 ± 0.86. The gap between the best and worst tree is 3.1 which would result in a tree with a mAP of 21.25.



(a) (b)

Figure 4.10 – (a) Histogram of the tree performances. MCRT has a mAP of 21.8 and is located in the higher density region of the histogram. (b) Plot of the performances of the models. We also plotted the mean and variance performance in dotted blue lines. Our model is close to the mean performance of all the models. The model is far away from the worst possible performance given by M^f.

in front of their bicycles heavily occluding it. Therefore, the hierarchical clustering algorithm combined these classes together which is combined on a higher level with the {'motorbike'} class. It resulted in a average detection performance of 21.8.

Let M^m with m ∈ {a,b,c,d,e,f,g,h,i,j,k} be the model associated with the models in Fig. 4.11. The results obtained with these various models are summarized in Table 4.6. We show the individual class scores for each model M^i and their average performance. The last column ranks this respective mAP among all the 11 models.

We created a semantically meaningful tree in Fig. 4.12a nominated as M^a. We combined the two vehicle classes {'bus','car'} into one branch and did the same for {'bicycle', 'motorbike'}. For reasons mentioned before, we decided to put the {'person'}

category together with cycle category and the {'aeroplane'} with the vehicles. In many of other tree structures in the various tests, we found the cycle and/or vehicle super-class. This tree achieved a mAP of 21.7 which is surprisingly performing worse than the automatically deduced MCRT tree. Some of the other manually designed trees are modifications of this basic tree *e.g.* by rearranging some classes slightly differently *e.g.* in M^b where we swapped two classes.

The model M^h has no semantic meaning. The classes were manually put in a meaningless order for the human intuition. Interestingly, the best performance is given by M^h. Following very close are the models M^d and M^j. Right afterward comes M^c where we find a more natural ordering of the classes with the cycle and vehicle super-classes. But we can compare with M^g and note that these super-classes do not provide necessarily good results. There is a gap of 3.1 between the best and worst performing tree resulting in a tree with a mAP of 21.25. The MCRT tree is above the average of this gap which justifies the usage of our tree construction algorithm. Our model is at position 7 with 2.1% better than the worst model M^f and 1% worse than the best model. The performance of the object class depends on the tree model. The {'person'} class has a noticeable difference 8.2% or the {'motorbike'} class with 7.1% between their best and worst model. Although the obtained tree does not have top performance, being able to decouple tree construction and filter optimization has the advantage of reducing the training complexity compared to a joint optimization of tree structure and filters. The mean performance of all our tested trees is 21.94 ± 0.86. A histogram of the performances is shown in Fig. 4.10a. As expected, the automatically learned tree is located in the high density region of the plot. Fig. 4.10b shows the performance of the individual models. Our tree is very close to the mean performance of all the trees by being much better to the worst performance and closer to the best one.

In our case, the tree structure that has little semantic meaning attains the best score. Structuring classes into a semantic tree does perform well but not best. Our tree model contains semantic and feature level information and reaches a reasonable performance compared to the other models. The best tree depends strongly on the feature descriptor capturing the diversity of the images and more generally on the hierarchical framework. We believe that another learning algorithm *e.g.* boosting would result into a different best tree. [Salakhutdinov et al., 2011] use a similar tree construction algorithm and conclude that using an automatically constructed tree leads to better performance than using a predefined tree based on semantics. Our experiments proof this statement as our tree outperforms M^a with a strict semantically ordered structure. Moreover, we go one step further and conclude that the best tree does not necessarily need to be a result of a hierarchically clustering algorithm. The structure learning and optimization is a process which can be optimized jointly instead of fixing it using a similarity matrix and is left for future work in Sec. 7.2.

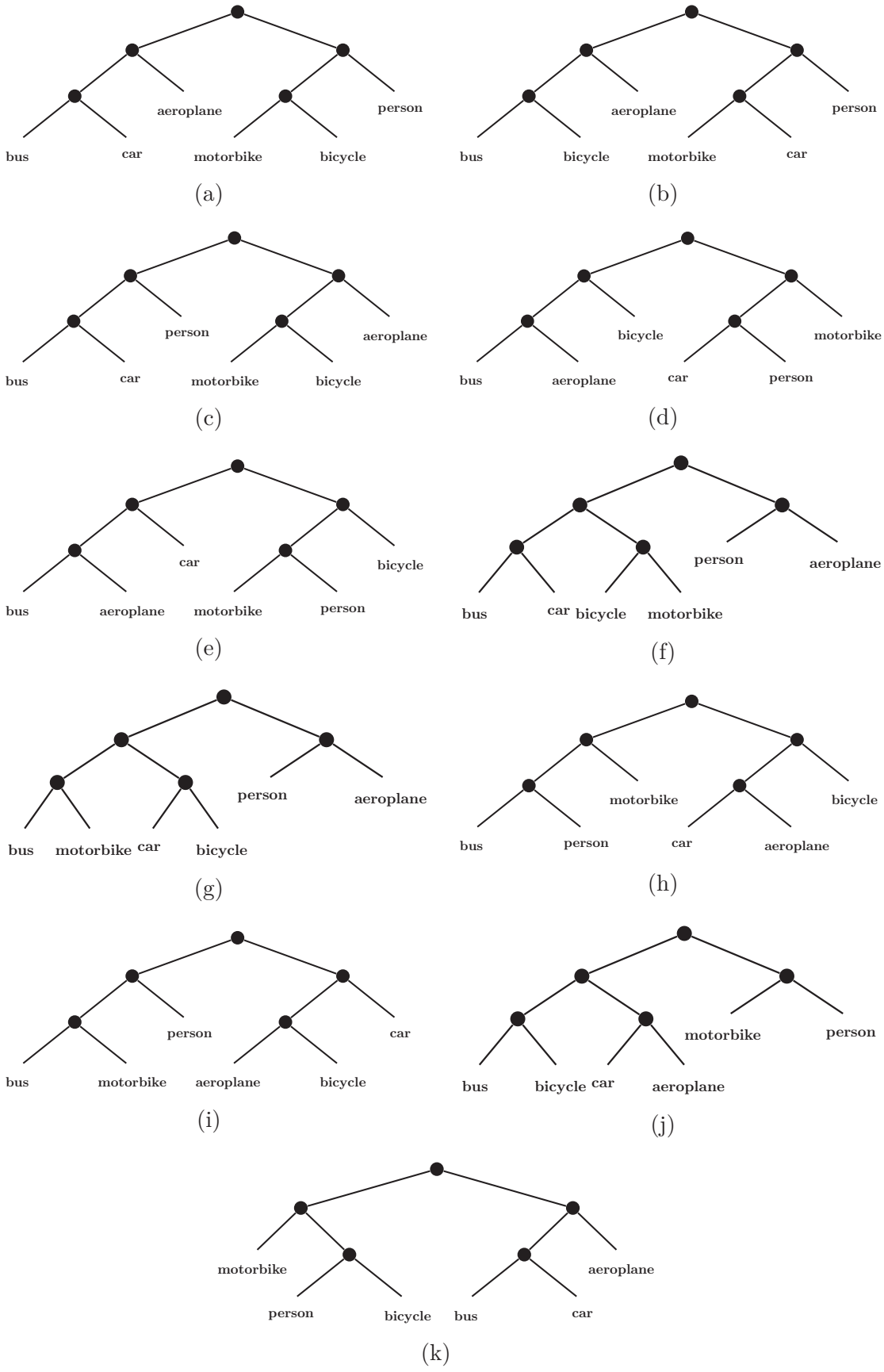


Figure 4.11 – Different trees used for comparison with the MCRT tree shown in (k).

4.4.4 Learning With Missing Training Data

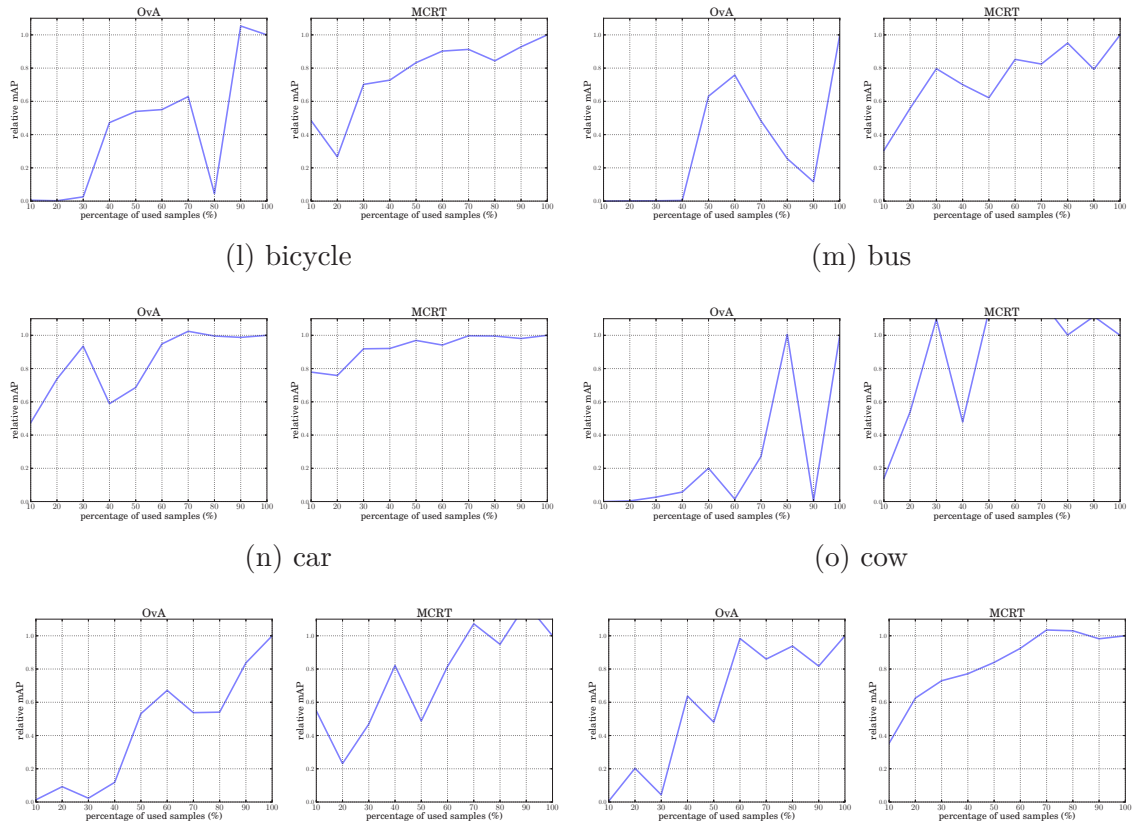
We introduced in Sec. 2.3.3 the concept of transfer learning. It consists in learning a set of classes using knowledge of other classes. This can be realized *e.g.* by sharing examples between classes or transforming the model of one class into another one. We are interested by the idea of sharing examples between classes. This means that one class y_i uses the examples of its own and all the other classes to build its final decision hyper plane. This is naturally the case in our formulation. The ancestor filters associated to each class are shared between many other classes and are optimized during training with the examples of these child classes. All the examples of the child classes contribute to the filters of the intermediate nodes. The final score of a class is the sum of the convolutions between these predecessor filters and the object features. For instance, the root filter is optimized taking into account all the examples and appears in the sum of each class score. The knowledge of all the examples is used to learn the root's weight vector.

We measure the performance of MCRT and OvA if an object class lacks of examples. How well do both methods generalize with missing examples of one class? We take $k = 10$ classes namely {'bus', 'bicycle', 'motorbike', 'car', 'aeroplane', 'person', 'cow', 'horse', 'dog', 'cat'} using the PASCAL VOC 2007 dataset. But due to time constraints, we experimented with 6 of them: {'bicycle', 'bus', 'car', 'cow', 'horse', 'motorbike'}. We chose 3 groups of 2 classes each with common characteristics:{'bus', 'car'}, {'bicycle', 'motorbike'} and {'cow', 'horse'}. The examples of these classes are visually similar and can be of help to learn the class with lacking information. The tree is the same as used in Sec. 4.4.1 for learning the $k = 10$ class model. These groups of classes are clustered together and share each the same immediate parent. This constellation allows us to explore the impact of example sharing as the examples of these groups shape their immediate parents' filters.

The class with lacking training data, called the target class, is alternated one by one in each experiment. The other classes are called base classes. For each class, we start with 10% of the total examples and increase to 100% linearly with a step size of 10. The examples are selected in the order of appearance in the dataset. This makes a total of $10 \times 6 = 60$ experiments. For instance in an experiment, all the classes use the full dataset except {'horse'} which uses first 10% of its examples, then 20% in another test etc. For each test, we train a MCRT and OvA from scratch with the reduced dataset for one class and we record the mean average precision (mAP). The closer the mAP is to the performance using all available annotations, the less dependent is the method on the training data and the faster it can generalize.

Impact on The Target Class

The results are presented in Fig. 4.12 for all the 6 possible target classes and both learning techniques. Each figure shows the relative mAP of the method based on the percentage of used examples. The mAP values are relative to the performance obtained for each method when exploiting all annotations. Intuitively, we would expect the best performance when using 100% of the dataset and see a convergence of the mAPs to this final value. We use relative mAP to compare the dependency on the number of training samples for each method and to not compare the methods to each other. The faster the



(l) bicycle (m) bus

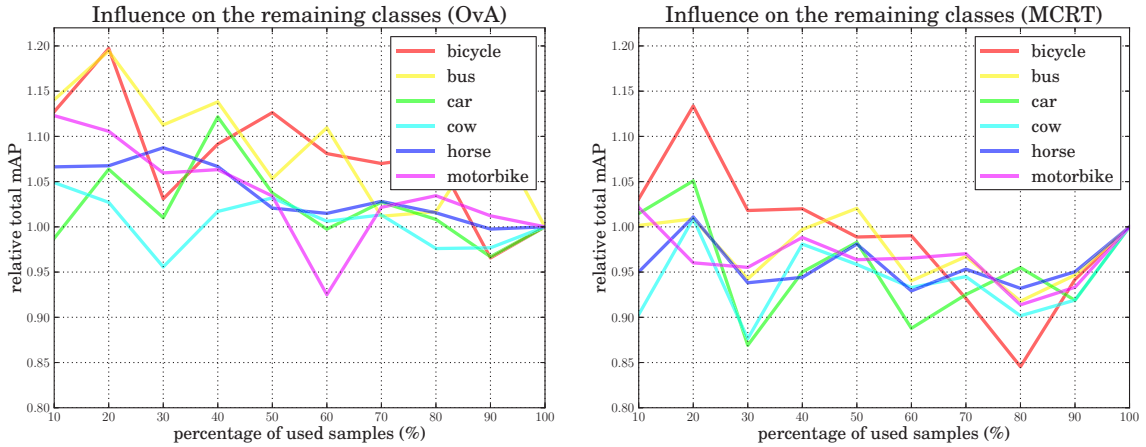
(n) car (o) cow

(p) horse (q) motorbike

Figure 4.12 – Results for one target class with missing training samples when learning a multi-class detection model for 10 classes. We experimented with 6 classes out of 10. The final performance of each method is reached when using 100% of the training data. The mAP values are relative to the final performance for each method and class and show the evolution of this relative mAP with increasing training set.

results converge, the better the learning approach can generalize. The starting relative mAP obtained at 10% can be used to characterize this property. A high value for this point means that the method can learn the new class without needing a large set of annotations. For each class, the left figure plots the relative mAP for OvA method and the right one the relative mAP for MCRT. The results expressed in mAP when using the full dataset is 15.9% for OvA and 17.0% for MCRT. This proves again the superiority of the hierarchical MCRT method over a flat OvA structure as a multi-class object detector.

For each of the six classes, MCRT outperforms OvA in learning from few samples the target class. Most importantly, the starting point of each plot is higher than those of OvA. For instance, the car class learned with MCRT reaches 77.9% of the final score with only 10% of the examples as where the OvA method just attains 47.5% of its full performance. For all the other classes {'bicycle', 'bus', 'cow', 'horse', 'motorbike'}, the starting mAP value for OvA is nearly 0. The detection performance gets better when considering more samples. The staring mAP value for MCRT is always higher than 0 and depends on the considered target class. Its performance increases quickly



(a) (b)

Figure 4.13 – Mean average precision of the base classes depending on the number of samples exploited for the highlighted target class. The mAP values are relative to the final performance when the full dataset is available.

with growing number of samples. This can be nicely observed for the cow object category where MCRT performs very well with little object information but OvA only starts learning the model when having more than 70% of examples delivered. Another example is shown in the horse plot. The MCRT curve is always higher than the OvA curve. The fluctuation can be explained by the selected examples for each experiment. We picked the samples in sorted order where some subsets do not necessarily have key visual representations of the object. Thus for certain subsets the generalization capability suffers.

The car class is little sensible to the size of the training set. It starts at a relative score of 80% and saturates already with 70% of the training data. OvA handles the lack of information well too. This observation can be of help when creating a new dataset. In Pascal VOC dataset, the car class has the second highest number of annotations after the person class. The transfer learning results show that depending on the multi-class method, the focus of the annotators can be shifted to different classes.

We conclude that our method is better able to learn the target class that is missing some samples than OvA. Contrary to the baseline, MCRT uses a tree structure with intermediate super-classes. The filters of these nodes are learned jointly using common samples of the child classes. The missing samples of one class can be compensated through the data of its neighboring classes. The decision of one class depends on the scores of all its ancestor super-classes which use the fully available annotations of the remaining classes to construct discriminative filters. The knowledge of the base classes help the performance of the target class. The motorbike and bicycle classes are visually similar and interact in a similar way with other objects *e.g.* persons. Both classes benefit from this example sharing and have nearly the average final performance with MCRT instead of none.

Impact on The Base Classes

Until now, we considered the performance of the target class in the multi-class approach. Fig. 4.13 shows the mAP for the base classes depending on the number of samples used for the target class and learning method. The score at each experiment is relative to the final score of the base classes. The OvA technique often performs better when the target class uses little examples. While the target class performs worse, the base classes reach a better score. The bus curve is always above 1. When we use little examples for the bus class, the average performance of the base classes increases.

MCRT behaves slightly differently. For most experiments, the performance is worse when little information is used. For instance for the bus experiments, the results are slightly higher or worse than the final performance. All the filters and classes benefit from the presence of many training data. The similarity matrix for each experiment with the bus class is shown in Fig. 4.14, once with OvA and once with MCRT. We note that for the OvA class the bus class remains very weak for the first tests and maintains a high confusion with all the classes. On the contrary, MCRT has a clear diagonal in the similarity matrix and the confusion with other classes decreases when we add more samples while the intra-class similarity increases. We note that the final similarity matrix is much cleaner compared to the final matrix of the traditional method.

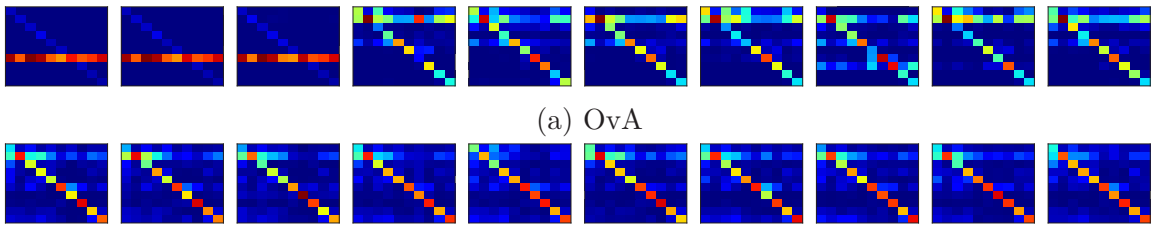
Conclusion

In general, the use of a big dataset helps to improve the overall performance. An exception is for the bicycle experiment where first the average performance of the base classes drops. We believe that this is conditioned on the selected samples which do not represent well the object category. The mean value for the mAP of the base classes and for all experiments with OvA is 1.04 ± 0.046 and MCRT 0.96 ± 0.0431. The former method fluctuates more with the number of training samples while the latter is less dependent on it and delivers more stable results. MCRT can compensate the lack of training samples and is able to immediately learn an object representation. This makes the method interesting to learn many more classes and keeping the cost for annotation low. Finally, when building a dataset, directions for the annotators should consider the difficulty to learn an object class and focus on annotating classes which require large amount of training samples.

4.5 Conclusion

We have presented a detection system which is based on tree of classifiers. Each node in the tree represents a classifier with an associated weight vector. These weight vectors are not binary classifiers and attribute scores to regions of an input image by calculating the convolution between these weight vectors and the region's features. The sums of these values along each path form the final decision scores for each class.

The learning process is completely automatic. The user only specifies the object classes and the parameters of the system *e.g.* the dataset. Our tree learning process is not new and was used in previous works *e.g.* [Bengio et al., 2010]. But we are the first to propose to learn such a tree structure in a joint learning procedure. This is made



(a) OvA

(b) MCRT

Figure 4.14 – Similarity matrix obtained for the bus experiments. Each column represents the a percentage of used training data starting with 10% and ending with 100%. The classes appear in the following order from top-to-bottom and left-to-right: {'dog', 'person', 'cat', 'horse', 'cow', 'car', 'bus', 'aeroplane', 'bicycle', 'motorbike'} which reflects the tree order. The similarity matrices get cleaner With increasing number of training samples. MCRT has from the beginning on a stable similarity matrix and can compensate the lack of data with its discriminative intermediate filters learned with examples of other classes.

possible through the introduction of combined ranking and classification constraints. An input patch is whether classified as background or ranked where the highest ranking class determines the class label. This lead to a structured learning problem that was solved in an efficient implementation inspired by [Joachims et al., 2009].

The traditional method OvA is a flat model. The classes among each other do not share information and do not influence each others scores. We opted for a hierarchical structure instead of this flat model as previous results presented in Sec. 3.2.2 haven shown an improvement of performance compared to OvA. This hierarchical structure formulation is transparent to the selected object descriptor allowing the use of future feature vectors. We described several contributions in this chapter. First, we formulated a hybrid learning and detection procedure consisting of classification and ranking mechanisms. The ranking is necessary to assure comparable scores. The classification dismisses background regions. While using a hierarchy is not new, solving it efficiently and jointly with our constraints is new.

The results validate our assumption on the performance of structured classes: MCRT outperforms our baseline with respect to detection performance on various datasets and combination of classes. These classes do not necessarily need to be similar for MCRT to beat OvA as discussed in Sec. 4.4.2. But the margin is bigger when the classes are better related *e.g.* sharing similar features and context. The automatically learned tree is not ideal. This is not surprising as the tree is fixed in the first step and then the filters are learned. We could have naively iterated over these two steps which would have led to unacceptable learning rates. MCRT's can learn classes with little training examples is another strong point of this method as future large datasets may not consist of a large number of data per class. This ability can be explained by the use of a hierarchy where the super-classes compensate for the lack of data in the individual leaf nodes.

However, MCRT has a higher run-time than the flat OvA method. Thus, we did only fulfill one of the challenges mentioned in the introduction in Chapter 1 namely to improve detection performance. But the run-time of our framework is higher than the

baseline. We need to compute the convolutions for not only the k leaf nodes as in OvA but also for the filters of the super-classes. The size of the tree increases linearly with the number of classes as we will discuss in the next chapter. Therefore, we need to find another inference method to overcome this limitation. Finding a solution to this issue is the subject of the next chapter.

CHAPTER 5

Accelerating the Inference Time with a Tree of Classifiers

Contents

The detection model of Chapter 4 has a slow run-time compared to our baseline OvA detector. Thus, the next section justifies the need to accelerate our detection model. We do this by proposing a tree traversal algorithm which selects best possible paths leading to the right leaf node. At the same time, the Sec. 5.2 will review the most common tree traversal algorithms. The super-class filters capture the appearances of many object types and can represent the child object classes in lower dimensions than the one used for the leaf nodes. Therefore, we will reduce the number of features in most classifiers of the tree as discussed in Sec. 5.3. Both approaches are complementary. Our contribution leads to a trade-off between speed and accuracy. We evaluate our modified algorithm in Sec. 5.4.

This chapter details several contributions. Our problem statement from the previous chapter naturally enables the use of a tree traversal algorithm. We suggest a variant that allows to balance between accuracy of detection and execution time. Background samples are rejected as soon as their optimistic confidence is too small. As we only need to look for the best score, the method tries to search relevant paths in the tree. Moreover, we reduce the number of features which is complementary to the tree traversal method. We also show how to train the parameters of the tree traversal algorithm as not to lose in precision but gain in speed of detection.

5.1 Motivation

In this chapter, we propose a solution to improve the run-time of MCRT. The complexity of a problem influences how fast the algorithm processes an input image. The time analysis here does not represent the complete chain of the detection system because we are rather interested in how long the classifier needs to evaluate one position in the feature pyramid. The analysis is a function of the number of classes k in order to understand the scalability of the system. Based on the number of classes the run-time increases

- sub-linearly : the time complexity grows slower than k that is $\mathcal{O}_{\text{sub}} < \mathcal{O}(k)$
- linearly : the time complexity grows as fast as k that is $\mathcal{O}_{\text{lin}} = \mathcal{O}(k)$
- super-linearly : the time complexity grows faster than k that is $\mathcal{O}_{\text{sup}} > \mathcal{O}(k)$

Usually, one strives to design sub-linear time algorithms. Based on the applications, linear or super-linear complexities are tolerated in practice.

In Chapter 4, we introduced an object detection framework. It scores a position by evaluating the edges of the tree T. The sum of the scores of each individual path up to the leaf nodes produces the final score for each class. We have seen that the detected class label results from the highest ranked path. If the score is smaller than a threshold, ideally -1, we say it is a background sample. This is the exhaustive way of evaluating our tree. Having $|\mathrm{T}|$ weight vectors to score a sample, it leads to a run-time of $t_{\text{MCRT}} = |\mathrm{T}|$ where we divided by the average time for one filter pass. In all cases, the run-time of this exhaustive search method is higher than OvA: $t_{\text{MCRT}} \geq t_{\text{OvA}} = k$. The complexity of both algorithms are nevertheless linear. The complexity of OvA technique grows directly linearly with the number of classes k and the complexity of MCRT depends on $|\mathrm{T}|$ which also grows linearly with k multiplied by a constant factor as presented in Sec. 5.2.4. This constant factor plays a crucial role in real-time applications and therefore we are interested in both the time complexity and the running time.

In practice, our slow detection is undesirable as we would like to keep the run-time as small as possible and it is preferable that the run-time be a magnitude faster than a OvA technique. Chapter 4 presented a hierarchical model with a multitude of weight vectors concatenated into a large weight vector. It also presented many reasons why this novel approach improves detection performance over a OvA baseline. In this chapter, we tackle exactly the execution time of MCRT.

In the literature, tree hierarchies for multi-class object detection are little explored. In multi-class image classification (*e.g.* [Griffin and Perona, 2008]), the use of binary trees with binary classifiers at every node allows immediately to gain a logarithmic complexity $\mathcal{O}(log_2 k)$ as at every leaf we divide by half the number of classes. We propose a tree search algorithm for the case of multi-class object detection with trees as it is the case in our work. We do not evaluate every path and edge in the tree. Thus, the algorithm saves the evaluation of many filters. At the same time, the search path algorithm guarantees to find the best path and simultaneously maintain its detection precision. While the binary trees used in multi-class classification propagate erroneous decisions from the top levels to the bottom ones, the tree traversal algorithm can rectify possible previously wrong decisions.

In addition, we apply feature reduction on the object descriptors based on the level of the tree allowing to further gain in run-time. We can do this, as we will see later, because the tree traversal algorithm spends the most of the processing time on the top filters. Both contributions follow from the way we formulated our detection objective (4.2) coupled with the learning objective of Eq. (4.7).

We assume that the probability of appearance of the k object categories is equal. Even though it is not of relevance here, the background is by far the most dominant class:

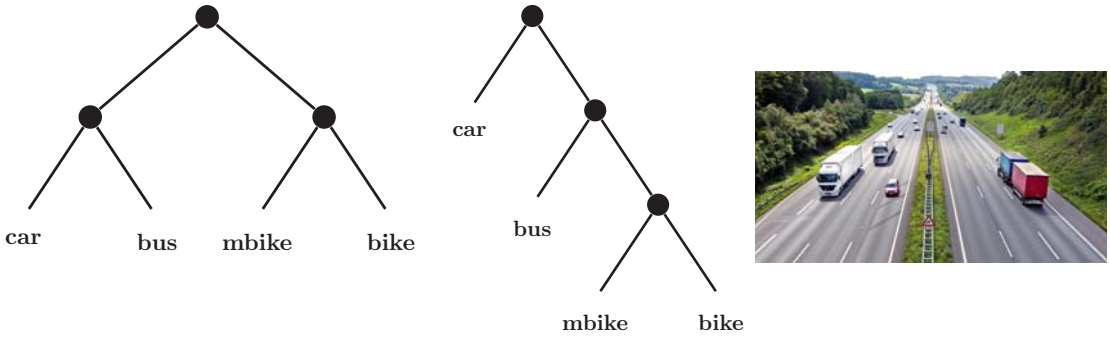
$$\mathrm{P}(y_1|x) = \mathrm{P}(y_2|x) = ... = \mathrm{P}(y_k|x) \ll \mathrm{P}(y_{\mathrm{bg}}|x) \tag{5.1}$$

We keep the number of ancestors equal between the leaf nodes, as every class appears with the same probability in an image. Except mentioned otherwise, we just handle *almost balanced trees*. A balanced tree has equal height, that is depth, on the right and left-hand side of the root filter. An almost balanced tree tolerates a difference of 1 between these both heights. The advantage of this chosen tree structure is that every class is represented by the same amount of filters. In Fig. 5.1a, we give an example of a balanced tree. The path to each leaf node scores 3 filters. But for an unbalanced tree as in Fig. 5.1a, the average number of filters applied would be $14/4 = 3.5$ which is more time consuming as for the balanced case. Further the tree scores 2 filters for the car class but for the bicycle class it scores 4 filters which makes the latter class more robust.

The condition in Eq. (5.1) is not a limitation of the system but an adaptation to the dataset. One could have reasoned for the tree structure differently, if there were a dominant object class *e.g.* on highways as in Fig. 5.1c. The probability to see car is much higher than any other class *e.g.* bicycles or even buses:

$$\mathrm{P}(y_{\mathrm{car}}|x) > \mathrm{P}(y_i|x) \quad , \forall y_i \in \mathcal{Y}^+ \setminus \{y_{\mathrm{car}}\} \tag{5.2}$$

The tree could have looked like in Fig. 5.1b where we end up quickly in the car leaf node and know the final scores for the dominant class. The tree traversal algorithm would not need to go any deeper in the hierarchy for most object positions in the image. The drawback is that the car class score is only the sum of 2 filters instead of the 3 in a balanced tree.



(a) (b) (c)

Figure 5.1 – Example of a balanced (a) and an unbalanced tree (b). The categories in the balanced tree share the same number of super-classes while this is not the case for an unbalanced tree. (c) shows a highway with primarily cars in the image. The tree in (b) allows to rapidly get the final score of the dominant car class. The ordering of the classes influence the rapidity of the detection and depend of the context.

5.2 Tree Search Algorithm

5.2.1 Preliminaries

Searching a graph is linked with the specific goal of finding a path from a start node to a goal node. The complementary condition of finding the shortest path is often desired. The task is even more complicated as certain paths may be occluded. The algorithms must thus be able to move their ways around the obstacles. A well known application of it is in video games. A computer player needs to make its way through a maze in order to reach a certain object or person. Another common application of it is finding the best direction on web mapping websites such as *google maps*.

We have to differentiate between algorithms that only have an objective of connecting two nodes and those algorithms that find the shortest path between two nodes in a graph. The length of a path is determined by the sum of the weights of the edges passed. If all the weights of the edges in the graph have unit length this becomes equivalent to finding the shortest path with the fewest nodes on its way. We say that the graph has equal path cost.

Objective

Let us formalize the objective. We only treat the case of undirected graphs for its ease of formulation. Further it is sufficient to thoroughly understand its use to our multi-class object detection situation. Given an undirected graph G with nodes N and edges E, a path from node n_S to node n_T is defined as $P = (n_S, n_2, \ldots, n_T) \in N, N, \ldots, N$. Let w_{ij} be the weight of the edge e_{ij}. This weight is often represented by a distance function $\text{dist}(n_i, n_j)$. The shortest path can be obtained by minimizing the following

linear programming objective:

$$\begin{aligned} &\min_{x} \sum_{i,j \in E} w_{ij} x_{ij} \\ &s.t.\ \ x_{ij} \geq 0 \\ &\qquad \sum_{j} x_{ij} - \sum_{j} x_{ji} = \begin{cases} 1 & \text{if } i = S \\ -1 & \text{if } i = T \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{5.3}$$

Here x_{ij} is an indicator variable which equals 1 if the shortest path passes through the edge e_{ij}. The minimization problem aims at finding the path with the smallest sum of the weights. The constraints ensure that the paths do not include jumps between unconnected nodes. It states that each outgoing edge ends up in a child node.

In the presence of infinite graphs encountered in some problems such as finding a solution to Rubik's Cube, an algorithm is said to be *complete* if it definitely finds the target node as long as one exists. On the other hand, a search method is said to be *optimal* if the found solution is also the shortest path.

Path Finding Algorithms

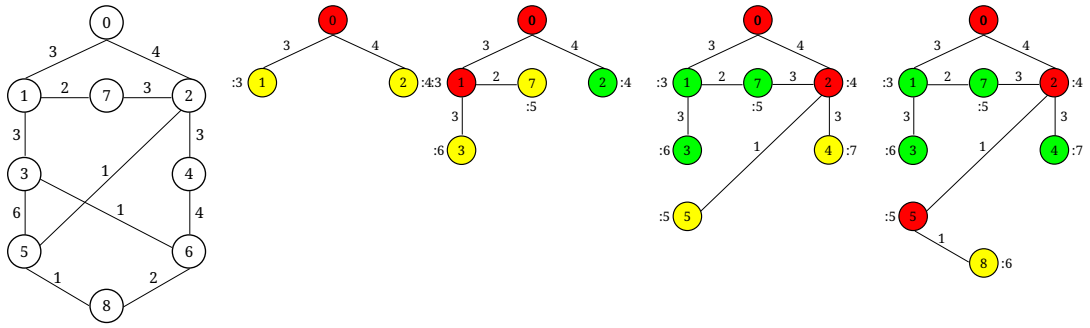
Before going deeper into the shortest path finding algorithms, we review the most known algorithms for finding a path between two nodes. This resulting path does not necessarily need to be the shortest path depending on the algorithm. We mention a few which helps to gain a good overview of the possible methods.

Depth-first search The depth-first search (DFS) algorithm, invented by the french mathematician Charles Pierre Trémaux, traverses the graph from a chosen starting node until it reaches the target node. It is based on the intuition of exploring one branch of the graph before backtracking and looking on another unexplored branch of it. Beginning from a starting node, it explores its first child node. This process is repeated until the target is found or the path ends. Then DFS backtracks to the most recent unexplored node in the graph. It is called depth-first search as the algorithm favors looking first into the depth. If the algorithm is implemented in an iterative manner, one has to use a stack to keep track of the current vertices and those visited.

In the worst-case scenario, the algorithm exhaustively traverses the whole graph and the algorithm runs in $\mathcal{O}(|N| + |E|)$. The depth-first search suffers from incompleteness in infinite graphs where it can get stuck in a branch of the graph with no target node. Also, DFS does not guarantee to find the shortest path.

Breadth-first search Another exhaustive algorithm is the breadth-first search or BFS algorithm created by Edward F. Moore in the 1950s. On the contrary to the DFS method it does not explore first the depth of a graph but proceeds in "circles". Starting from a given node, it explores all its child nodes. If the target class is not among them, BFS goes deeper by choosing the child nodes of its own child nodes. It goes deeper in the graph one level at a time.

Here again, in the worst-case, the algorithms has a run-time of $\mathcal{O}(|N| + |E|)$ as it eventually needs to explore all the edges and nodes. Contrary to the DFS method,



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

Figure 5.2 – Illustration of Dijkstra's algorithm: (a) shows a graph with the respective weights of the edges noted near the connections. The starting node is the one with ID $n_S = 0$ and the target is node number $n_T = 8$. The algorithm starts with node n_S and evaluates its child nodes as shown in (b). The distance to get to node 1 is $dist(0, 1) = 3$ and the distance to get to node 2 is similarly $dist(0, 2) = 4$. We keep a history of all seen distances on a stack and choose the path with the lowest cost. In (c), we proceed by evaluating node number 1. The new distances added to the stack are $dist(0, 7) = 5$ and $dist(0, 3) = 6$. The current minimum is given by the path going from node 0 to node 2. The algorithm corrects its previously chosen path and continue by following node 2 as in (d). This procedure is repeated and the method finally finds the shortest path to node 8 including the cost of the path.

BFS is complete and optimal: It guarantees to find a solution if it is possible to reach the target class starting from a given node and the found path is automatically the shortest one too.

Dijkstra Another greedy algorithm for finding shortest paths is Dijkstras algorithm [Dijkstra, 1959]. It is conceived by the computer scientist Edgar W. Dijkstra in 1956. It became one of the most deployed algorithms if no heuristics are available to estimate the distance to the goal node. The two previous methods neglect the edge weight while Dijkstra's method takes them into consideration. The method is similar to the breadth-first search on unweighted graphs that is graphs having equal edge weights.

The algorithm as depicted in Fig. 5.2 starts as usual with the starting node. Then it evaluates its child nodes by adding their weight of the edges to the current distance to the source. Next, we continue with the node having the shortest distance. We keep a history of all currently seen distances in a stack. At every iteration, we continue with the path having the lowest distance to the source with the help of this stack. The edges are only allowed to have negative weights if the graph degenerates into a tree that is no node has more than one predecessor. The worst case performance is $\mathcal{O}(|E| + |N| \log(|N|))$ as one has to go over all vertices $|N|$, for each iteration find the best current node taking $\mathcal{O}(log(|N|))$ and explore the outgoing edges giving a total of $\mathcal{O}(|E|)$.

Greedy best-first search While Dijkstra's algorithm chooses the next step based on the distance to the source, greedy best-first [Russell and Norvig, 2003] uses the

distance to the target. This distance is expressed through a heuristic function $h(n)$ estimating the distance between current node n and n_T. This search is also sometimes called informed search as it has information about the target. At each iteration, we process next the node which appears to be closest to the target thus having the smallest heuristic $h(n)$. However, this algorithm can get stuck in loops and does not necessarily find the shortest path.

A-star search algorithm We inspired our work on the A* algorithm [Hart et al., 1968]. It is very similar to the Dijkstra's method but uses a further source of information that allows it to outperform Dijkstra. Dijkstra's method calculates at every iteration the current distance from a source vertex to the currently seen nodes. A* further applies heuristics similar to greedy best-first search estimating the distance from these nodes to the target nodes. When no heuristics can be used, A* degenerates to Dijkstra's search method. It guarantees to find the shortest path when the below mentioned conditions are met.

The algorithms described earlier blindly decide which node to process next. A* on the other hand guesses the most promising path. The algorithm is also able to rectify paths once the cost of the best path estimate drops below the previously known estimates. For every node n, it uses a function $g(n)$ which defines the cost from the source n_S to n and a heuristic $h(n)$ estimating the cost respectively distance from node n to node n_T. $g(n)$ determines the already traversed distance and $h(n)$ the possibly remaining cost to reach the goal. Thus, the cost $f(n)$ of a node is given by:

$$f(n) = \underbrace{g(n)}_{\text{source-to-node cost}} + \underbrace{h(n)}_{\text{node-to-target cost}} . \tag{5.4}$$

The A* search algorithm combines the concepts of Dijkstra's algorithm that is choosing nodes with a short distance to the source node given by $g(n)$ and greedy best-first search that is favoring close nodes to the target given by $h(n)$.

The function $h(n)$ must fulfill the condition of *admissibility* that is $h(n)$ must never overestimate the cost from any node to the target:

$$\begin{aligned} &h(n) \leq h^*(n) = \text{dist}(n, n_T) \\ &h(n_T) = 0. \end{aligned} \tag{5.5}$$

The total cost $f(n)$ is a lower bound estimation of the path's actual cost. Thus, this heuristic is an optimistic estimation. A property of A* is that every node with $f(n) < \text{dist}(n_S, n_T)$ is explored. For instance, on a simple square grid where movements are only to go left, right, down and up, one can use the Manhattan distance as a heuristic function.

However by violating the restriction of admissibility one can arise interesting situations. If the function $h(n) = 0$ for any node, then A* acts exactly as Dijkstra's algorithm thus finding the shortest path. This is also the case if $h(n)$ is always lower or equal to the actual cost of reaching the target. The closer the heuristic is to the real cost, the faster A* finds the best path. When the heuristic exactly corresponds to the real costs, A* just expands nodes belonging to the real path. Another way to

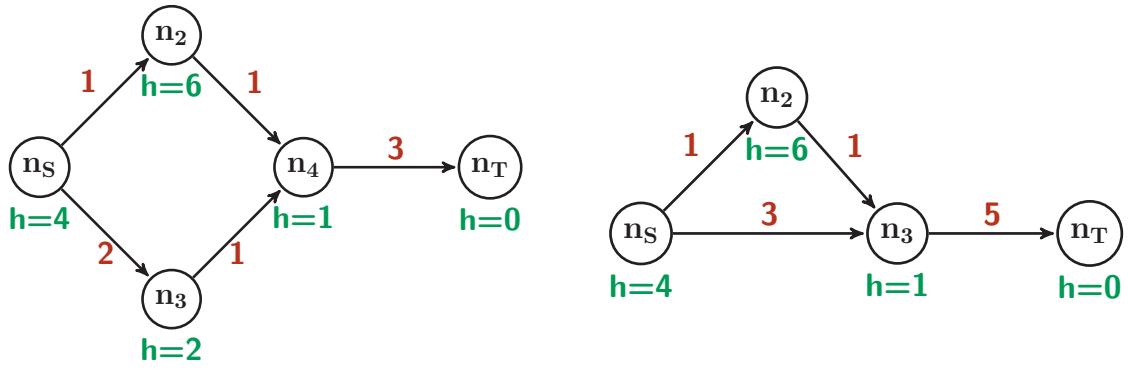


Figure 5.3 – Illustration of the admissibility and consistency properties. The edge weights are marked in red aside the edges and the heuristic is given in green below each node. (a) The heuristic of node n_2 is not admissible as it overestimates the cost to node n_T. The A* algorithm finds the wrong path to the goal node. (b) The heuristic is not consistent which forces A* to expand twice the node n_3.

reach quickly the target node is by sometimes overestimating the actual cost. We exploit these properties in our work to achieve a trade-off between speed and accuracy of detection in Sec. 5.2.2. Finally if $f(n) = h(n)$, A* behaves as a greedy best-first search.

The examples in Fig. 5.4 illustrate the results of Dijkstra's method and A* with three different heuristic functions on a simple maze problem. The objective is to get from the starting position in orange to the goal node in red. The first method does the most number of evaluations as it just considers the distance to the source while Fig. 5.5b and Fig. 5.5c use a further information namely the distance to the target. The examples in Fig. 5.5b and Fig. 5.5c show the impact of the heuristic function. In Fig. 5.5b the Euclidean distance underestimates the right distance to the target, while in Fig. 5.5c the Manhattan distance is a closer heuristic to the real distance between node and target. Underestimating the real distance allows still to find the shortest path but takes much more time. In any case, all three methods guarantee to get the shortest path. Each method found one of the several possible shortest paths. However, this is not anymore the case for Fig. 5.5d. We chose a heuristic function that overestimates the distance to target taking the squared Euclidean distance as a heuristic function. As mentioned earlier, A* does not guarantee to find the shortest path. One advantage that comes with non admissible heuristic is that the target is often found much faster even though the path is now longer.

Another property that influences A* in presence of closed graphs is *consistency* of the heuristic $h(n)$. It is sometimes called monotonicity of $h(n)$. A consistent heuristic has the property that the heuristic of a node $h(n_i)$ is smaller than adding the cost of the path from n_i to n_j and the heuristic of $h(n_j)$. The heuristic is consistent everywhere if for two nodes n_i and n_j, the following condition is fulfilled:

$$\begin{aligned} &n_i \in \text{anc}(n_j) \\ &h(n_i) \leq \text{dist}(n_i, n_j) + h(n_j). \end{aligned} \tag{5.6}$$

This property can accelerate the convergence when dealing with closed graphs. It avoids finding another path to a same node but with a lower cost. Consequently, the method does not explore the same node multiple times. A heuristic that is consistent is automatically admissible. One can prove this by changing in Eq. (5.6) the node n_j with the target node n_T and knowing that $h(n_T) = 0$ using Eq. (5.5). The previous condition states again that the heuristic has to underestimate the real distance to the target which is the admissibility condition. The reverse statement between admissibility and consistency is not true: A heuristic that is admissible is not necessarily consistent. Consistency is as such a stronger criterion. The property of admissibility however is sufficient to make A* optimal in case of closed graphs. For directed trees, consistency does not influence the rapidity of the search algorithm as there is only one way to get from a node to another one.

An intuition on the two properties that is admissibility and consistency is given in Fig. 5.3. The first picture Fig. 5.3a shows that by choosing non admissible heuristic, the wrong path returned. Starting from node n_S, the two nodes n_2 and n_3 are evaluated which gives the following cost functions $f(n_2) = 1 + 6 = 7$ and $f(n_3) = 2 + 2 = 4$. The heuristic $h(n_2) = 6$ of node n_2 overestimates the remaining path cost of $1 + 3 = 4$. Choosing the best predicted path, one moves to vertex n_3 and evaluates n_4 with $f(n_4) = 3 + 1 = 4$. Finally, the target node is reached with a total path cost of $f(n_T) = 6$. This cost is smaller than the first predicted path cost of $f(n_2) = 7$ and consequently, the right path $(n_S \rightarrow n_2 \rightarrow n_4 \rightarrow n_T)$ is not found.

Fig. 5.3b illustrates how inconsistent heuristic forces A* to revisit the node n_3. The algorithm expands the child nodes of n_S and chooses the smallest cost function between $f(n_2) = 7$ and $f(n_3) = 4$. Nevertheless, the estimated path cost of node n_2 is kept in a priority queue. When continuing the path from n_S through node n_3 and reaching the goal node, the total cost is 8 which is not anymore the smallest cost in the priority queue. The search continues from node n_2 by expanding again node n_3 and finding the shortest path $(n_s \rightarrow n_2 \rightarrow n_3 \rightarrow n_T)$ with a cost of 7. Both toy examples do not have equal step cost between two displacements: Moving from node n_S to n_2 is not equally penalized as moving to node n_3 in Fig. 5.3b. Finally, the shortest path is the one with the lowest edge weights and not the number of nodes to traverse.

A* is especially interesting as it is *optimally efficient* for every possible consistent heuristic. This means that there is no other optimal algorithm using the same heuristic that extends fewer nodes during the search for the target than A* . The reason is that A* only evaluates paths whose total estimated cost $f(n)$ is smaller than the shortest path because the heuristic underestimate the real cost. Any other algorithm expanding fewer nodes takes the risk of traversing a suboptimal path as it does not expand all paths with a path cost smaller than the final cost.

The heuristic plays a major role in the time efficiency of the search algorithm. The closer the heuristic function is to the real cost, the fewer nodes are expanded. Let us illustrate this by having two admissible heuristics h_1 and h_2 with $h_2 \geq h_1$. One says that h_2 dominates h_1. The heuristic function h_2 produces total path costs $f_2(n)$ which are closer to the real path cost. In return more nodes are expanded with h_1 as $f_1(n) \leq f_2(n) \leq \text{dist}(n_S, n_T)$ and we saw before that nodes whose estimated costs are smaller than the length of the real shortest path are explored.

A way to quantify the quality of a heuristic is the effective branching factor b^*. It is the average number of children of a node in the tree that A* can traverse. If the number of all possible nodes expanded by A* is N and d_{T} the distance of the shortest path in this tree, than the branching factor is defined as:

$$1 + b^* + (b^*)^2 + \cdots + (b^*)^{d_{\mathrm{T}}} = \frac{1 - b^{*d_{\mathrm{T}}+1}}{1 - b^*} = N. \tag{5.7}$$

The branching factor takes values $b^* \geq 1$. In the ideal case, where a search algorithm takes directly the optimal path, the branching factor is $b^* = 1$. The closer the branching factor is to 1, the faster is the optimal search algorithm. The branching factor can be calculated empirically by applying the search algorithm in various situations and record the number of nodes evaluated by A* and the length of the shortest path. The heuristic producing the smallest branching factor is more favorable to the search problem. In the previous example it would be $h_2(n)$.

Another way to qualify the heuristic is the absolute error $\Delta = h^* - h$ and relative error $\epsilon = \Delta/h^*$ with h^* the perfect heuristic giving the right cost to the target node. The error analysis of the heuristic allows to give a time complexity of the algorithm. In the worst case, A* has to expand all the successors of a node up to the goal state which leads to $\mathcal{O}(b^{d_{\mathrm{T}}})$ where here b is the average number of successors of a node in the tree. Otherwise the complexities can be given in many special cases. If the graph is a tree with a single goal state and the error between h and h^* grows slower than the logarithm of h^* that is: $|h^*(n) - h(n)| = \mathcal{O}(log h^*(n))$, the run-time becomes polynomial. If we drop the assumption on the heuristic but allow to return backwards in the tree, we get $\mathcal{O}(b^{\Delta})$ with Δ the maximum absolute error. A step cost is the cost of taking a step *e.g.* moving around a grid has a constant step cost of 1. By further assuming a constant step cost, the complexity is expressed as $\mathcal{O}(b^{\epsilon d_{\mathrm{T}}}) = \mathcal{O}((b^{\epsilon})^{d_{\mathrm{T}}})$ which leads to the real effective branching factor of b^{ϵ}.

The method starts as usual with the source node n_S. At every iteration at a node, the algorithm evaluates the cost function $f(n)$ for its child nodes and selects the one producing the smallest cost. The most promising path depends on the currently traveled distance from n_S and the expected distance from this child node to n_T. Simultaneously, it keeps a list of already expanded nodes in CLOSED set and a list of possible next moves in OPEN. At the beginning of an iteration, the algorithm selects the node in OPEN with the best cost and adds this node in CLOSED avoiding unnecessary processing same paths twice. On the other hand if a new shorter distance to an already visited node in CLOSED is found, this node is added to OPEN again with the new lower cost. Thus, infinite loops are avoided and it is guaranteed to get the shortest path. We show an example pseudo-code in Alg. 4. We did not show how to record the best path for simplicity but this can be easily achieved by using pointers in nodes' structure pointing to the predecessors that lead to the node.

5.2.2 Application to Multi-class Object Detection

During detection, every path and thus every node is inspected at least once and a convolution between the input features and the node filters is computed. Evaluating exhaustively the tree takes $\mathcal{O}(|\mathrm{T}|)$. Consequently, the run-time grows significantly with

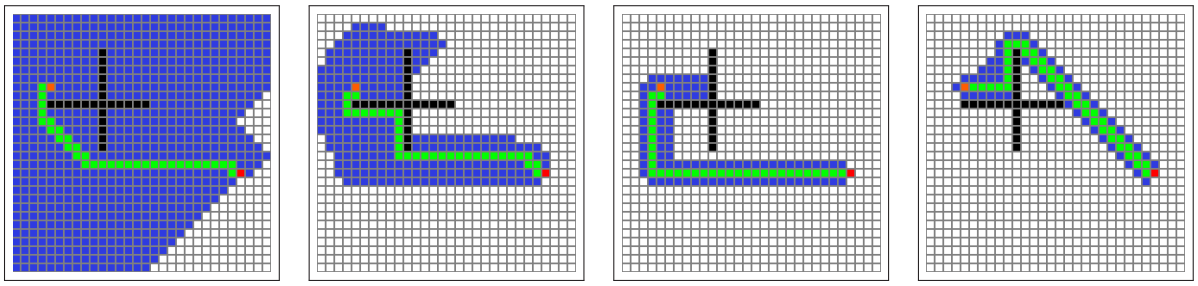
Algorithm 4: A-star Algorithm

Input: Graph G with nodes N
Input: Starting node n_S
Output: Path to target node n_T
S_{open} contains currently best candidates for shortest path: $S_{open} = \{n_S\}$
S_{closed} contains already processed nodes: $S_{open} = \emptyset$

while $S_{open} \neq \emptyset$ **do**
  `// find node with lowest cost`
  u $\leftarrow \arg\min\{S_{open}\}$
  `// treat node as examined`
  $S_{open} = S_{open} \setminus \{u\}$
  $S_{closed} = S_{closed} \cup \{u\}$
  `// return if target node reached`
  if *v equals* n_T **then return**
  foreach *v being a child of node u* **do**
    $f(v) = g(v) + h(v)$
    if $v \notin S_{open}$ ***and*** $v \notin S_{closed}$ **then**
      $S_{open} = S_{open} \cup \{v\}$
    else if $v \in S_{open}$ **then**
      Update current best path for v if necessary
    else
      $S_{open} = S_{open} \cup \{v\}$
      Remove from closed set: $S_{closed} = S_{closed} \setminus \{v\}$

the number of classes k that is the number of nodes $|T| = \mathcal{O}(k)$ avoiding the constant factor. In practice this constant factor which depends on the number of outgoing edges plays an important role for real-time applications. We go one step further and simply want to reduce this linear dependence on the number of classes. To overcome this disadvantage of classifying with trees, we propose the use of a tree search algorithm. Its objective is to evaluate as little as possible the nodes in the tree. Ideally, only nodes lying on the path to the correct leaf node would be applied. When scoring the bus region in the Fig. 5.5, we would like to evaluate the following scoring filters: {{car,bus,mbike,bike}, {car,bus}, {bus}}. We do not know the right path in advance. To find this path, we need to evaluate every child node lying on the path to the desired class label. Taking the previous example, this gives us:{{car,bus,mbike,bike}, {car,bus}, {mbike,bike}, {bus}}. Our algorithm follows from our problem formulation in Eq. (4.7): the right path is ranked highest following the constraint Eq. (4.7d). The objective is to find the path having the highest gain reflected by the class scores. It is also capable of rejecting background samples as soon as possible in the tree which is the consequence of the classification constraint Eq. (4.7g).

The algorithm operates similar to A* : It creates an empty priority queue S_{open} of future scores. Starting from the root vertex, MCRT calculates an entry score for a position in the image. The calculated score reflects an optimistic estimation of the best achievable final score. Therefore, if this best possible score is too small, it is already



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

Figure 5.4 – A simple maze in a two dimensional grid. The movements can only be done in 4 directions: right, left, down and up. Moving in diagonal direction is not possible. The starting node n_S is represented by the orange block while the target node n_T is colored in red. The black blocks are obstacles. The green color shows the path found by each algorithm and the blue color the evaluated blocks. (a) Dijkstra's search algorithm. It evaluates 757 blocks and finds one of the shortest path of length 34. (b) A* method using Euclidean distance as heuristic. It processes 267 regions and finds also one of the shortest paths having a length of 34. (c) A* algorithm using Manhattan distance as heuristic. The number of placements is 119 and the algorithm found too one of the shortest paths of length 34. (d) A* algorithm with the squared Euclidean distance as heuristic which is not admissible. The number of steps taken is 93 however the best path found has a length of 42.

possible to reject the example as background. Next, MCRT evaluates the outgoing edges and re-estimates an optimistic score for the individual paths. The algorithm continues on its path with the best score and records all the other encountered scores in S_{open}. This process is repeated for every newly met nodes. At a given node, the estimated score depends on the sum of edge scores and a heuristic which is an upper-bound of underlying scores. At any stage in the hierarchy, MCRT can return and choose another path if the confidence of the current path is lower than another path in S_{open}. The algorithm stops once it achieves a leaf node with a score higher than any other value in the priority queue. The end score does not contain any heuristic and is the sum of scores of the traversed edges.

This strategy allows us to quickly find the best path in the tree T. While traversing the tree, it is also possible to rectify decisions. As we keep a history of scores in a priority stack, we are able to switch paths as soon as the current path attains a low confidence. Errors made earlier at higher tree level can thus be corrected afterward. Estimating the final score at different nodes in T allows us to reject background samples during detection often in the early stages of the tree.

5.2.3 Formalization

To this end, we introduce a source-to-node function $g(x, n_i) = w \cdot \Phi_i(x)$ and a heuristic $h(n_i) = t_i$. The former represents the sum of the edge weights in the tree starting at the root node to the current node n_i. The edge weight calculated using Eq. (4.3) is the sum of the already calculated scores which represents a partial sum of the final classification score. The heuristic $h(n_i)$ can be seen as a node-to-target function which

estimates the partial sum of the remaining final score. It is independent of the example x. $h(n_i) = t_i$ has constant value and is determined during the learning phase. The gain function $f(x, n_i)$ gives an optimistic estimation of the best possible score when continuing on the outgoing paths of n_i:

$$f(x, n_i) = g(x, n_i) + t_i. \tag{5.8}$$

The heuristic must overestimate the actual remaining score. Consequently, t_i is chosen to be higher or equal than the actual gain of reaching the target class. If this condition is fulfilled , MCRT guarantees to get to the right leaf node. This search technique is inspired by A* able to find the shortest path in a graph. We speak of a gaining function $f(x, n_i)$ and not cost function as in Sec. 5.2.1 since we are looking for the path in the tree with maximum value. We could have instead introduced for example the normalized inverse of the score as a cost function.

It is to note that our heuristic is admissible to the training and validation set and not to the test images which are unknown to the training algorithm. After determining the heuristic on the validation set and testing them on the training set, we observed that MCRT usually found the correct path. This indicates that our learned heuristic overestimates indeed the final scores. In practice, we also use less admissible heuristic that means values for t_i that are less optimistic. This prunes possible paths and rejects more quickly background samples. The drawback is a reduction in precision.

We would like to point the difference to a cascade of detectors [Viola and Jones, 2001]. A cascade eliminates at each stage negative samples using less and less time-consuming features. This is not our case. Further a cascade evaluates each example by passing it through stages. If at one stage the example is rejected, we stop the evaluation. The later stages and thus important features are neglected which is the contrary of our method. We use heuristic which involves the influence of descendant nodes and we only reject an example if its upper-bound score falls below a threshold *e.g.* -1. In a cascade each level is trained independently and does not take into account later stages. Each stage uses only a subset of the training data. This is in contrast to our approach: We propose a joint convex optimization problem where the stacked weight vector $w = \{w_1, w_2, \ldots, w_{|\mathrm{T}|}\}$ is learned using the complete training information.

Fig. 5.5 illustrates how various levels in T avoid evaluations of the local weight vectors on the image region. For the input image, MCRT extracts the features on various levels. The figure shows the processing of the paths in T for one level. First, it calculates the entry score by computing the convolution of the feature map with the root filter w_1 and adding the heuristic t_1. This produces best possible scores for each position in the feature map. The scores already smaller than a threshold are pruned and the corresponding regions are labeled as background. These regions are colored in red while other positions forwarded to the child nodes are marked in green. For these possible object instances, again the upper-bound score $f(x, n_2)$ and $f(x, n_3)$ are examined. Each candidate position keeps a record of its optimistic estimations for each path in the tree. We note for example that at the leaf nodes, the filters w_3^l and w_4^l are only applied to very few positions on the feature map.

Another illustration is shown in Fig. 4.3 where we depict the heat map of applied filters. The red regions show a high use of many filters and the blue ones are immediately filtered out by the root node at the top of the tree. This lower part shows

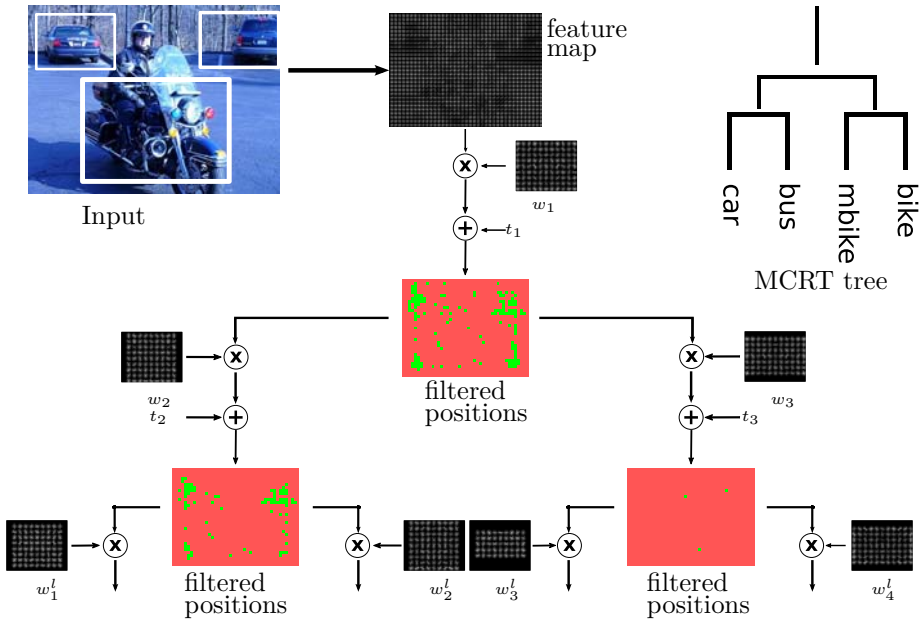


Figure 5.5 – The image illustrates the regions at one scale of the image where the nodes of T are evaluated. The red color indicates positions where no evaluation is done. The leaf nodes are just applied to a reduced number of positions.

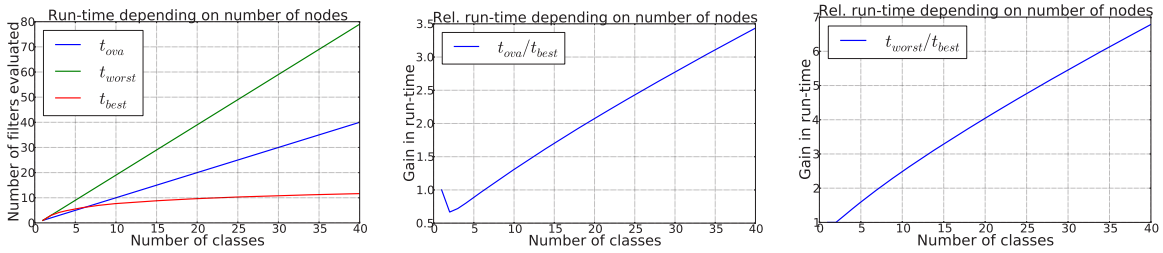
exemplary the different paths run through by 4 selected regions. One is rejected at the top node while some others penetrate deeper into the tree. This is related to the fact that the background region is difficult to classify due to its similarity with a foreground object class or that the region is simply a positive example.

5.2.4 Understanding the run-time

In order to better understand the time complexity, let us assume that the examples treated by the tree belong only to the k object categories. We will include later the background into the analysis. We did not put any prior appearance probability and every class has equal chance to appear $\mathrm{P}(y_1|x) = ... = \mathrm{P}(y_{|\mathrm{T}|}|x)\ , \forall y_i \in \mathcal{Y}^+$. Further, we neglect the time needed to find the best score in priority queue as this is much smaller than the time to compute a convolution between a filter and an image feature region. We give the timing relative to the time spent for computing the convolution of one filter where for simplicity every filter w_i in the tree has equal dimensions. We will first have a look on the run-time itself and later on the time complexity as in practice we also do care of the time constants.

The complexity analysis heavily depends on the heuristic. In the worst case, MCRT passes every edge in the tree T and evaluates $|\mathrm{T}|$ filters. This exhaustive search takes in total $t_{\mathrm{exh}} = |\mathrm{T}|$. Having a balanced tree with b outgoing edges at every node and k leaf nodes, the tree has a depth of $d_{\mathrm{T}} = log_b(k) + 1$ and a total of

$$|\mathrm{T}| = 1 + b + b^2 + \cdots + b^{d_{\mathrm{T}}-1} = \frac{bk - 1}{b - 1} \tag{5.9}$$



(a) (b) (c)

Figure 5.6 – Run-time of different approaches depending on the number of classes. (a) Absolute run-time of OvA, the worst and best case scenario. OvA and the exhaustive search increase linearly while the best case scenario increases logarithmically in k. (b) Relative run-time of the best-case scenario when compared to OvA. One notes that our approach is favorable in presence of many classes. (c) Relative run-time of the best-case scenario when compared to an exhaustive tree search.

nodes. The exhaustive detection time for the binary tree $b = 2$ becomes $t_{\text{exh}} = 2k-1 \geq t_{\text{OvA}} = k$. The run-time of Ova t_{OvA} is as expected smaller which is unfavorable for MCRT.

We have a look at the best case scenario in order to get an intuition behind the tree search algorithm when applied to MCRT. In this particular case, the path from root node to a leaf node has length d_{T}. On the way from top to the bottom, a total of $1 + b(d_{\text{T}} - 1)$ edges is scored: the entry cost plus those of the outgoing edges of every passed node lying on the path to the right class. In consequence by replacing d_{T} with its expression depending on k, the run-time becomes $t_{\text{best}} = blog_b(k) + 1$. We plot the relative run-times of these three times in Fig. 5.6. Comparing the relative run-time between OvA and the best case scenario for MCRT with a binary tree, we note that for very small number of classes $k < 7$, OvA detects faster than MCRT but otherwise is outperformed by our approach.

The time complexity of the exhaustive search and OvA are both the same that is the detection time grows linearly with the number of classes $\mathcal{O}_{\text{OvA}} = \mathcal{O}_{\text{exh}} = \mathcal{O}(k)$. However in the best-case scenario, MCRT only increases logarithmically with k which leads to $\mathcal{O}_{\text{best}} = \mathcal{O}(log(k))$. Compared to the exhaustive technique, using the search algorithm allows to considerably speed-up the frame-rate of the detector. A further observation is that we have the best gain in run-time performance compared to OvA by choosing $b = 2$ in t_{best}. This justifies our choice of binary trees.

In practice, an image is in great part dominated by negative examples. In most scenes, especially from the VOC challenges, only a few object instances are present in the image. The same holds true in many surveillance applications where few object categories need to be recognized *e.g.* parking monitoring or advanced driver assistance systems.

MCRT has the advantage to be able to stop searching for the output label if the best possible scores are already too small for all paths. We can notice this in Fig. 5.5. Many background positions are already classified and labeled as negative after the first gain function $f(x, n_1)$ is calculated using (w_1, t_1). This allows MCRT to considerably speed

Algorithm 5: Determine heuristic t_i for every node

Input: Tree T
Input: Number of classes k
Input: Minimum overlap factor δ_o
Output: heuristic value for each node t_i
$D = \emptyset$
for $y = 1, ..., k$ **do**
  foreach *annotated object* I_i **do**
    `// Launch detector on image` I_i
    determine $D_i \equiv$ `detect` (I_i, δ_o)
    $D = D \cup D_i$
foreach *node* $n_i \in T$ *from top-to-bottom* **do**
  Calculate t_i given by Eq. (5.11)
  `// Prune detections in` D `scoring higher than` t_i
  `prune`(D, n_i, t_i)

up the time needed to process one frame. We show empirically in the experimental in Sec. 5.4.1 the run-time of MCRT on real world images.

5.2.5 Learning Tight Heuristic

Let us first consider admissible heuristic which allows to find the path to the highest scoring class on the *training* set when using the search technique. The heuristic must in this case overestimate the best final score when continuing on all the underlying paths of a node and be as close as possible to the actual gain. The latter property is crucial to avoid propagating negative samples too deep in the hierarchy or trying many wrong paths. With the heuristic kept as small as possible, negative examples are eliminated much sooner in the hierarchy as only a small estimated confidence is added to the current accumulated score. Algorithm 5 depicts how we choose tight values.

The heuristic is determined from top-to-bottom and left-to-right. The annotations in the validation set of all k classes are used to learn these values. Every annotation a_i is associated to a specific location in an image I_i. MCRT is launched on a region around a_i which produces a set of detections D_i. Each element in D_i is a valid instance that is an instance being correctly detected $y_i = \hat{y}_i = arg\max g(x, n_y^l)$ and having a sufficient overlap $\delta_o$with a_i:

$$D_i = \{\forall x \in I_i \mid \hat{y} = y_i \wedge \text{ overlap}(x, a_i) \geq \delta_o\}\,. \tag{5.10}$$

Let $D = \{D_1, \ldots, D_{n^+}\}$ be the collection of all D_i. Then we define t_i as the upper-bound of the sum of the remaining edge scores for every individual path:

$$\forall (x,y) \in D : t_i \geq \max_{\forall D_j \subset D} \min_{\forall (x,y) \in D_j} \sum_{n \in \text{path}(n_i, n_y^l)} \underbrace{w_n \cdot \phi_n(x)}_{\text{individual filter score}}\,, \tag{5.11}$$

where we denote by $\text{path}(n_j, n_y^l)$ the set of nodes being on the path from node n_i to n_y^l. At each node, the multiple instances for an annotation a_i that are strictly bigger than t_i are suppressed. However, at least one detection for every annotation is kept and forwarded to the child nodes. The approach assures that at least one detection ends up in the correct leaf node n_y^l by assuring that the gain function $f(x, n_c)$ of every node n_c on the correct path is higher than the gain function of wrong paths in the tree:

$$\forall i, \exists (x, y) \in D_i \text{ and } n_c \in \text{path}(n_r, n_y^l), n_{\bar{c}} \in \text{T} : \\ f(x, n_c) \geq f(x, n_{\bar{c}}),$$

The heuristic values are learned from experiences given the data available during the learning phase. Admissible heuristics guarantee to end up in the correct leaf node regarding the training/validation set. Among the scores of the positives are outliers producing very high scores. Consequently t_i have large values allowing background samples going deep into the tree or foreground samples evaluating many different paths.

Fig. 5.7 depicts histograms of scores for different number of classes $k = \{2, 4, 6, 8, 10, 20\}$. We build a detector for each setup k. The classes are those used in our experiments as mentioned in Sec. 4.4. The distributions show a high density of scores around a mean value. At the end of the distributions, there are few examples which scored with very high values. In our case, we are interested in the high scoring positive outliers. Eliminating a certain percentage of these positive scores and learning t_i on the new subset of data creates even tighter heuristic. By cutting off more positives, we enforce the heuristic to be smaller and thus be less optimistic and finally gain more in detection time. The drawback is that the search algorithm does not guarantee finding the optimal path. But background samples are filtered more quickly and a region traverses less paths in total.

5.3 Dimensionality Reduction

The filter dimensions of the nodes decrease with the depth of the tree *e.g.* the root filter has the biggest width and height. During detection, the first layer nodes are evaluated the most while the deeper stages remain unevaluated by many background samples and foreground samples choosing the most confident paths. Practically, we can reduce the run-time by using less features in the first weight vectors. Intuitively, the first weight vectors leading to nodes regrouping many different classes do not need the full extent of the features as they are only globally describing the child classes.

We reduce dimensions of the features in the first layers and progressively increase the dimension. The weight vectors of the leaf nodes make use of the full feature size f. Consequently the run-time in the upper layers is reduced. We apply principal component analysis (PCA) for feature dimension reduction of the HOG features. The 31-dimensional feature vector in the [Felzenszwalb et al., 2010a] dataset is already a reduced version of the 108-dimensional object descriptor. However, the authors modified slightly the PCA for fast feature extraction. We experimented with reducing the 108-dimensional feature vector and had similar performances than reducing the 31-dimensional feature vector. We opted for the 31-dimensional vector for simpler seamless integration in our already existing training and detection system.

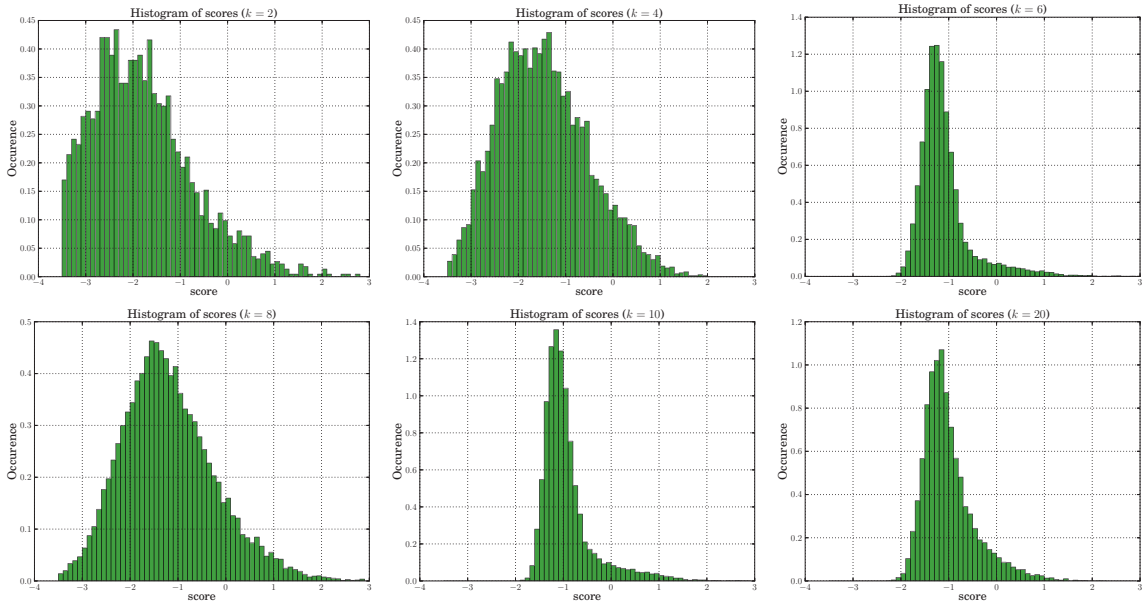


Figure 5.7 – Distribution of scores obtained for different setups when applying the detector to object regions on the validation set. The setups differ in the number of classes with $k = \{2, 4, 6, 8, 10, 20\}$ and a multi-class detector MCRT is built for each setup.

5.3.1 Mathematical Background

A weight vector w_i is defined by its model width W_i, height H_i and feature dimension f. This makes a total of $W_i \times H_i$ cells and $W_i \times H_i \times f$ features. The features of each cell in f dimensions are reduced to $\tilde{f}$ features. As such, a PCA is applied to the $W_i \times H_i$ cells producing a total features of $W_i \times H_i \times \tilde{f}$. We do not apply a PCA to all the features simultaneously. In a sliding window approach, this would require to project the features each time into the new smaller space with the moving windows which is very time consuming. In our case, we once project the cells of the feature pyramid and can apply our detector. The principle is illustrated in Fig. 5.8. PCA is an orthogonal linear transformation that transforms a feature vector $x_i \in \mathbb{R}^f$ to a new coordinate system $\tilde{x_i} = x_i \cdot E \in \mathbb{R}^{\tilde{f}}$. The greatest variance of the data grows with the index of the new coordinates. E contains the eigenvectors E_i obtained by solving $x_i E_i = \lambda_i x_i$. The eigenvectors form a basis for the new coordinate system. λ_i are eigenvalues and $g_i = \sum_{i=1..f} \lambda_i$ is called cumulative energy. The data matrix $X \in \mathbb{R}^{n \times f}$ consists of rows that are data samples and columns are the features. The steps to obtain E first subtract the empirical mean from X, next calculate the respective covariance matrix X_{cov}, obtain the eigenvalues and eigenvectors of X_{cov}. Finally, by rearranging the eigenvalues and eigenvectors in decreasing order we get E.

5.3.2 Application to Our Context

In our case, x is a cell in the image that we want to project into a lower dimensional space. Each cell is a data sample used to find the principal components. We extract HOG features of positive examples annotated with bounding boxes and scaled multiple

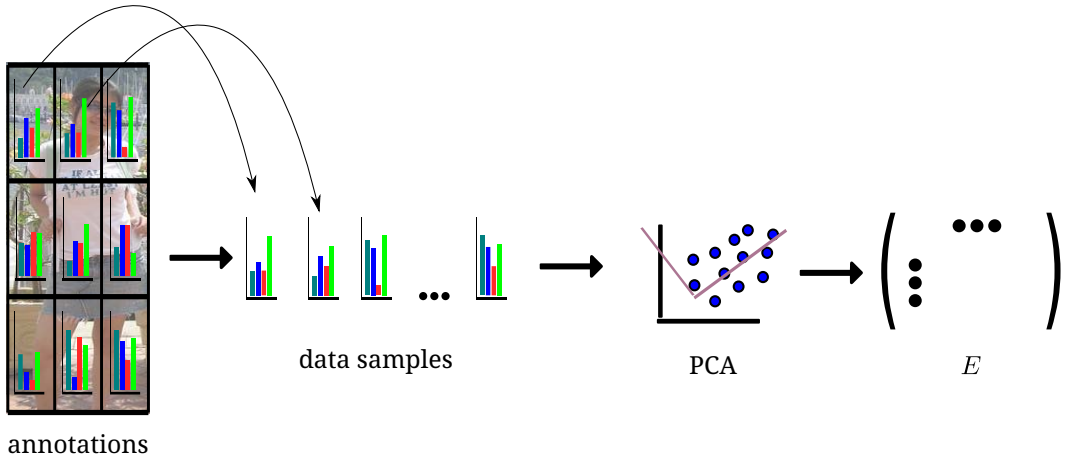


Figure 5.8 – Procedure for obtaining the eigenvectors of principle component analysis using the training examples. Each cell extracted by HOG around an object region is used as a single data. The collection of all HOG cells and all training images is fed to PCA for learning the eigenvectors.

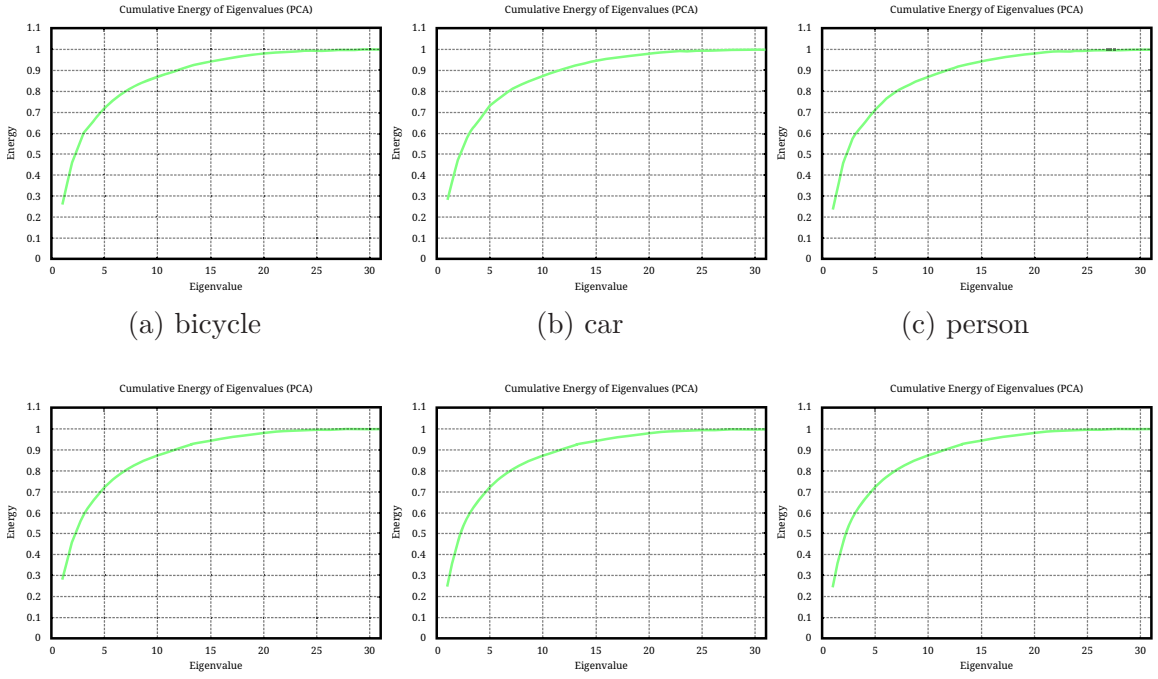
times. No negative samples are used. However loading all the features of every training image into memory simultaneously requires having sufficient memory space particularly with ever growing datasets.

An advantage of PCA over other dimension reduction methods is the possibility to load data sequentially and it uses small amount of memory. This can be done by computing the mean and covariance matrix X_{cov} iteratively for each cell cell_i with a total of $\sum(\text{cell})$ cells:

$$\begin{aligned} \text{cell}_{i,n} &= \text{cell}_{i,n} + \frac{\text{cell}_i}{\sum(\text{cell})} \\ X_{cov_i} &= X_{cov_i} + \text{cell}_i \text{cell}_i^{\mathrm{T}} \\ X_{cov} &= X_{cov} + X_{cov_i} \\ &\quad - \sum(\text{cell}) \text{cell}_{i,n} \text{cell}_{i,n}^{\mathrm{T}} \end{aligned} \tag{5.12}$$

In Fig. 5.9, we plot the relative cumulative energy g_i/g_f for some classes {'bicycle', 'car','person'} and a combination of several {5, 10, 20} classes. To this end, we extract all the HOG vectors for all the cells be it for a single class or a group of classes. The plots are highly similar and we achieve nearly the full energy around 25 features. Already around a feature size of 15 we have a cumulative energy of 95%. Further Fig. 5.10 helps to understand the influence of the feature size on detection performance. We trained a single class detector using our C++ version of the code of [Felzenszwalb et al., 2010a] but with a limited number of features. We note that already around 25 features, the detector obtains fairly optimal results. Using half of the total features decreases the performance around maximum 10%.

The leaf nodes apply all the f features (*e.g.* $f = 31$). The idea is to increase stepwise the feature dimension from the root node to the leaf nodes. Every node in the tree uses the same principal component analysis learned on all classes. For the root node, we fix the number of components to achieve at least an energy level of 95%.



(a) bicycle (b) car (c) person

(d) $k = 5$ (e) $k = 10$ (f) $k = 20$

Figure 5.9 – Cumulative energy of the eigenvalues after PCA reduction. (a,b,c) PCA is applied to only one class. (d,e,f) PCA is applied to a set of classes for three values of k. The classes were chosen in alphabetical order in the VOC 2007 dataset.

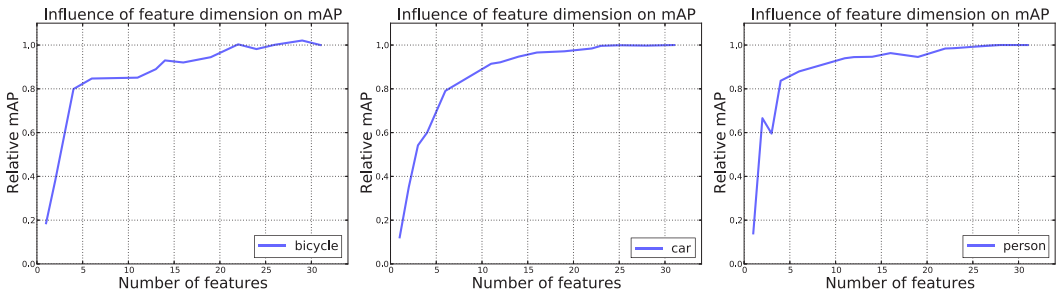


Figure 5.10 – Mean average precision (mAP) depending on the number of features for three different classes. The mAP is normalized to the score obtained when using the complete descriptor . The more features is used the better is the performance. The performance achieves already 90% of the best score when extracting at least 20 features.

Then, the feature dimension is chosen as to increase linearly the energy level up to the final energy level of 1 when reaching the leaf nodes. The precision of features and thus the run-time of applying the weight vectors w_i increases with the depth of the tree.

5.4 Results

We evaluate the methods introduced in this chapter in this section. We are notably interested in understanding the detection ability measured in mAP of the detector to

the run-time. The results are evaluated on the datasets of PASCAL VOC 2007 and 2010 as it was done in Sec. 4.4. The evaluation server for VOC 2010 allows only a limited number of evaluations per week. For the 2010 dataset, we used our modified offline version as many hundred of evaluations are needed for each setting of k.

We introduced the designed models OvA, MCR and eMCRT in Sec. 4.4. The OvA is the baseline and our reported scores throughout the experiments are often relative to its performance. We extend these models and include the following models. MCRT is our system that uses a tree structure to apply many more linear filters compared to OvA but includes admissible heuristics to traverse the tree. Using admissible heuristic, we should not deteriorate in detection performance as the heuristic guide the input to the correct leaf node. The version that does use less admissible heuristic is called fMCRT and achieves the same performance than OvA but has significantly faster detection times. The f stands for fast. We resume the various models in Table 5.1.

Model	Classif.	Ranking	Hierarchy	Inference
OvA	✓			k
MCR	✓	✓		k
MCRT	✓	✓	✓	$< \|\mathrm{T}\|$
fMCRT	✓	✓	✓	$\ll \|\mathrm{T}\|$
eMCRT	✓	✓	✓	$\|\mathrm{T}\|$

Table 5.1 – Properties of detection models.

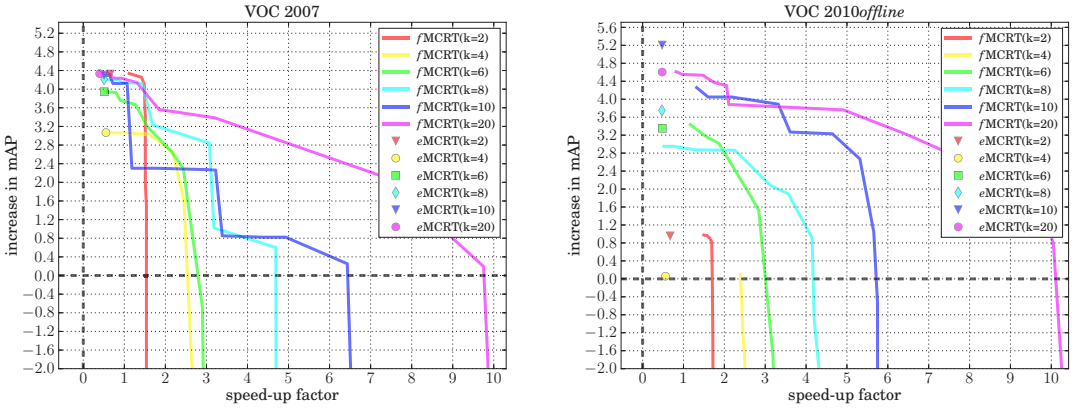
5.4.1 Fast Tree Traversal

We first compare the run-time of MCRT and fMCRT to eMCRT . Intuitively, the eMCRT is very time-consuming during detection as it has to evaluate all filters in the tree. Due to this exhaustive search, it should produce the optimal detection performance. MCRT intends to speed-up the inference time and therefore we would expect it to be faster than eMCRT and perform equally well during detection. The faster fMCRT should run quicker than both methods at the cost of lower performance. We chose the heuristic to reach the performance of OvA.

We experimented with different values for the heuristic as specified in Sec. 5.2.5. We cut-off the positive examples that score higher than a threshold and learned tight heuristic on the remaining examples. The threshold is determined as to erase a certain given percentage of the examples. We use a grid of values for the percentage value. By varying this threshold, we are able to create a trade-off curve between detection performance and run-time. The more positive examples we leave out during heuristic training, the faster the algorithm traverses the tree. Fig. 5.13 shows the detections and heatmaps obtained with MCRT and fMCRT . The former model traverses more nodes compared to the latter model to arrive at a final decision.

PASCAL VOC 2007

The results are depicted in Table 5.2. This exhaustive search over all filters in the tree is very time consuming but yields the best detection performance. MCRT reaches



(a) (b)

Figure 5.11 – Trade-off curve between detection performance and detection run-time. The training process can choose less admissible heuristic and speed-up the inference time. This comes at the cost of detection performance expressed in mAP. The numbers are relative to OvA. For the performance, we subtracted our score in mAP from the score of OvA. The speed-up is a relative value expressed in percentage of gain. (a) Plot for PASCAL VOC 2007 (b) Plot for PASCAL VOC 2010*offline*.

the same performance than *e*MCRT for all the 6 settings. This is an indicator that the admissible heuristic are well chosen as they reproduce the same behavior than a full evaluation would. The speed of MCRT is faster than *e*MCRT where we show the values relative to OvA. The fastest speed-up is attained by *f*MCRT when aiming the same performance as OvA. The trade-off curve is shown in Fig. 5.12a. It shows also

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	21.4	1x	22.3	1x	17.8	1x	16.1	1x	12.9	1x	8.8	1x
*e*MCRT	**25.7**	0.66x	**25.4**	0.55x	**21.8**	0.51x	**20.3**	0.51x	**17.2**	0.51x	**13.1**	0.4x
MCRT	25.7	1.12x	25.4	0.7x	21.8	0.6x	20.3	0.6x	17.2	0.55x	13.1	0.58x
*f*MCRT	21.4	**1.5x**	22.3	**2.6x**	17.8	**2.9x**	16.1	**4.6x**	12.9	**6.4x**	8.8	**9.8x**

Table 5.2 – Performance evaluation for PASCAL VOC 2007 dataset.

the characteristics for *e*MCRT . To improve in run-time, the algorithm needs to lose in detection performance. One can also note for the classes $k = \{2, 4, 6, 8, 20\}$, that it is possible to use less admissible heuristic but achieve the same performance as with admissible heuristic and have faster run-time. This can be seen clearly for $k = 4$, where the curve drops around a relative run-time of about 1.7x but still has the same detection performance as for the exhaustive search.

PASCAL VOC 2010*offline*

We note again that *e*MCRT produces the best detection performance. This method can be accelerated using admissible heuristic which produce here results close to or exactly as the exhaustive search method. This confirms our way to construct admissible heuristic using the training set. Two methods *e*MCRT and MCRT can perform

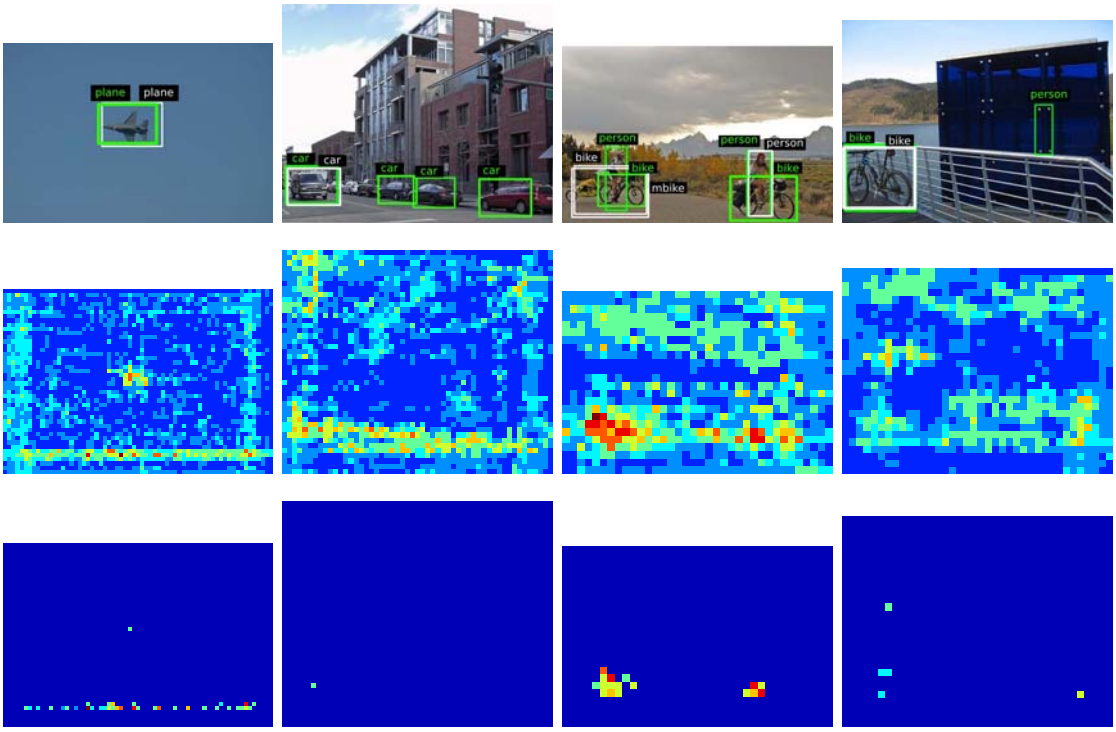


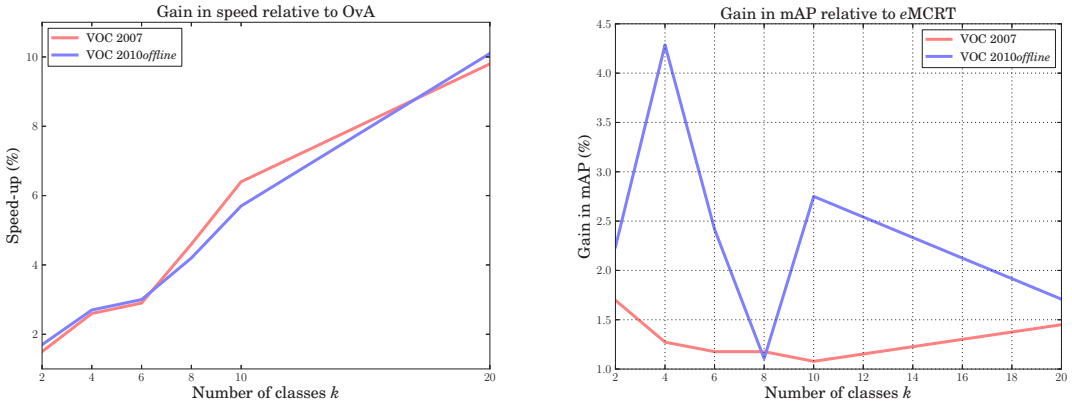
Figure 5.13 – Illustration obtained using MCRT and fMCRT . It shows the heatmaps of these detectors. The red color stands for evaluating all the filters and the blue color means that only the root filter was evaluated. The input images are shown on the first row with their detections. The detections of MCRT are in green and of eMCRT in white. The second row shows the heatmaps of the MCRT method. The third row shows the heatmaps of the fMCRT method which applies fewer filters.

differently as the heuristic are only admissible to the training set and not the test set as it is the case for $k = 10$. MCRT is faster during execution compared to OvA for some settings while being showing better performances. It is always faster compared to eMCRT up to a factor of 4 times.

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	30.5	1x	22.7	1x	21.2	1x	17.0	1x	13.4	1x	9.8	1x
eMCRT	**31.4**	0.67x	**22.8**	0.56x	**24.5**	0.48x	**20.8**	0.47x	**19.0**	0.48x	**14.4**	0.48x
MCRT	31.4	1.5x	22.8	2.4x	24.5	1.16x	20.0	0.52x	18.0	1.32x	14.4	0.82x
fMCRT	30.5	**1.7x**	22.7	**2.7x**	21.2	**3x**	17.0	**4.2x**	13.4	**5.7x**	9.8	**10.1x**

Table 5.2 – Performance evaluation for PASCAL VOC 2010*offline* dataset.

To have a better speed-up, the algorithm fMCRT attains the same performance as OvA but with much higher run-times. The gain in run-time varies with the number of classes and is highest for $k = 20$ with a factor of 10.1x. This shows that our method spends in average less time evaluating filters compared to OvA. The trade-off curve between performance and run-time normalized to OvA is shown in Fig. 5.12b.



(a) (b)

Figure 5.14 – (a) Relative gain in run-time using fMCRT compared to OvA. Both methods have comparable detection performances. The more classes are detected, the faster is fMCRT relative to a sequential evaluation of k detectors. (b) Relative gain in speed of MCRT relative to eMCRT . Both methods achieve comparable mean average precisions but MCRT is slightly faster. We note that the speed-up with admissible heuristic is conditioned on k and the dataset.

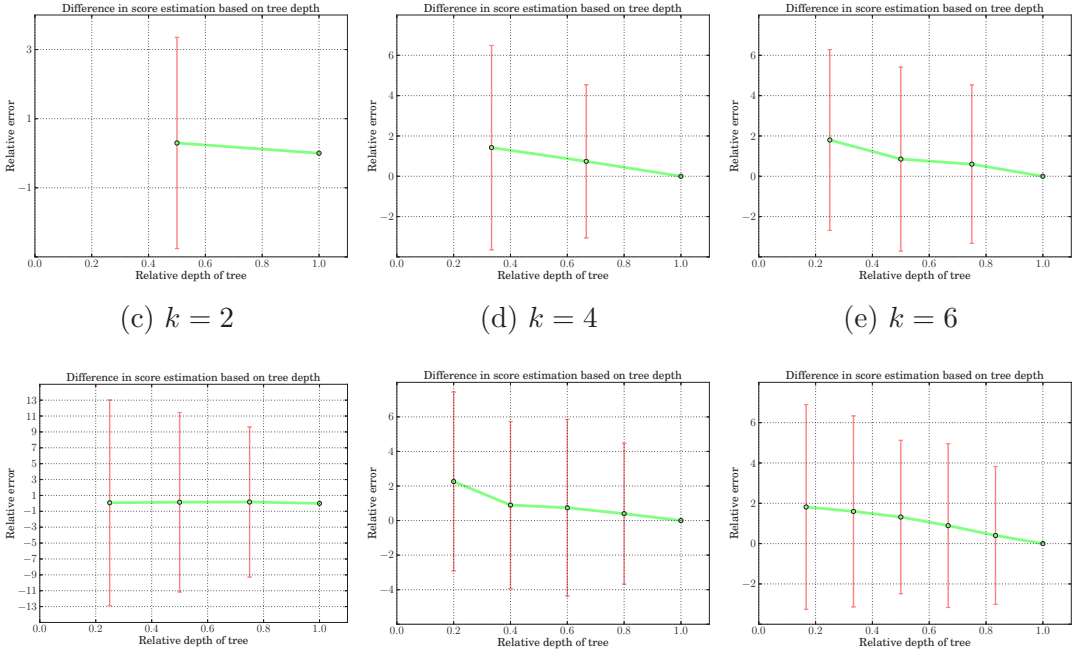
Dependence of Speed-up on k

A core objective of a multi-class classification or detection framework is to have scalable inference times. We consider the fMCRT algorithm. The dependence between the run-time of the detector and the number of classes recognized is wished to be kept small. The OvA algorithm has the basic run-time of $\mathcal{O}(k)$ as highlighted in Sec. 5.2.2 and we normalize the speed-up to the run-time of OvA. The speed-up of fMCRT increases linearly in the number of classes as shown in Fig. 5.15a while having the same detection performance as OvA. Empirically, we conclude that the run-time increases logarithmically with k.

In Fig. 5.15b, we show the speed-up of MCRT compared to eMCRT . Both methods have comparable detection performance. The behavior of the curves depends on the datasets. For the VOC 2010*offline* dataset the gain is generally bigger than for VOC 2007. Relying on admissible heuristic allows to gain in speed as not all the filters in the tree are evaluated. But by more strongly pruning the input through less admissible heuristic, we definitely achieve higher speed-ups.

Error in score estimation based on tree depth

During the tree traversal, each node estimates the final score. This estimation is optimistic and prone to error. Does this error drop the closer the traversal algorithm is to the correct leaf node? To answer this question, we launched the MCRT algorithm with admissible heuristic on test images of PASCAL VOC 07 challenge. The ground truth scores are obtained by evaluating the same test object instances with eMCRT . The relative difference between these two scores of both detectors define the error where as the difference is normalized with the ground truth. The tests were conducted for the diverse values of $k = \{2, 4, 6, 8, 10, 20\}$.



(c) $k = 2$ (d) $k = 4$ (e) $k = 6$

(f) $k = 8$ (g) $k = 10$ (h) $k = 20$

Figure 5.15 – Relative difference in score estimation at each level in the tree. We used admissible heuristic and show the mean difference at each level of the tree and standard deviation. The relative difference is calculated between the estimated final score at a specific node and its true detection value normalized with the ground truth. These ground truth values for all test samples are set by *e*MCRT . The closer the algorithm gets to the leaf node, the better is the estimation of the final score. At the last level of the tree, the obtained scores are very close to their true values.

The results are plotted in Fig. 5.15. The scores at each level are averaged and the standard deviation is calculated. We note a drop in the estimation error with increasing tree depth for all test settings. The estimation of the final score improves the closer the algorithm gets to the leaf nodes. This property allows to refine the correctness of the class estimation with the progress of the tree traversal method. The deeper the method is in the tree, the more confident it is about the estimated final score of a leaf node.

5.4.2 PCA

We extended the previous designs to include the dimensionality reduction discussed in Sec. 5.3. The number of features increases linearly from the root filter to the leaf nodes where all the features are applied. The first method, *pca-e*MCRT does an exhaustive search over all the nodes in this tree and does not use the tree traversal algorithm. The second method *pca-f*MCRT has the similar objective as *f*MCRT . It uses less admissible heuristic. These heuristic are chosen as to achieve the same detection performance than OvA but with faster detection times. It allows to evaluate the speed-up at same detection performance with this new hierarchy that has less overall features.

Detection Performance

The performance for these two designed models are shown in Table 5.3. Using a reduced number of features in the filters of the super-classes is prohibitive to the performance of the tree measured in mAP. In some cases the performances improve *e.g.* $k = \{20\}$ for VOC 2007 or $k = \{2, 4\}$ for VOC 2010*offline* datasets. This effect can be expected as using PCA does not exploit the full extent of the object representation. Nevertheless, the performance drop of *pca-e*MCRT is small and in general achieves 1% less than the *e*MCRT system.

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	21.4	1x	22.3	1x	17.8	1x	16.1	1x	12.9	1x	8.8	1x
*e*MCRT	**25.7**	0.66x	**25.4**	0.55x	**21.8**	0.51x	**20.3**	0.51x	**17.2**	0.51x	13.1	0.4x
MCRT	25.7	1.12x	25.4	0.7x	21.8	0.6x	20.3	0.6x	17.2	0.55x	13.1	0.58x
*f*MCRT	21.4	1.5x	22.3	2.6x	17.8	2.9x	16.1	4.6x	12.9	6.4x	8.8	9.8x
*pca-e*MCRT	23.9	-	24.3	-	21.8	-	19.2	-	16.5	-	**14.0**	-
*pca-f*MCRT	21.4	**2x**	22.3	**3.2x**	17.8	**4.3x**	16.1	**5.7x**	12.9	**7.7x**	8.8	**10.8x**

(a) PASCAL VOC 2007

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	30.5	1x	22.7	1x	21.2	1x	17.0	1x	13.4	1x	9.8	1x
*e*MCRT	31.4	0.67x	22.8	0.56x	**24.5**	0.48x	**20.8**	0.47x	**19.0**	0.48x	**14.4**	0.48x
MCRT	31.4	1.5x	22.8	2.4x	24.5	1.16x	20.0	0.52x	18.0	1.32x	14.4	0.82x
*f*MCRT	30.5	1.7x	22.7	2.7x	21.2	3x	17.0	4.2x	13.4	5.7x	9.8	10.1x
*pca-e*MCRT	**34.9**	-	**23.6**	-	23.6	-	19.6	-	16.2	-	13.3	-
*pca-f*MCRT	30.5	**2.3x**	22.7	**3.6x**	21.2	**3.3x**	17.0	**5.2x**	13.4	**6.2x**	9.8	**10.9x**

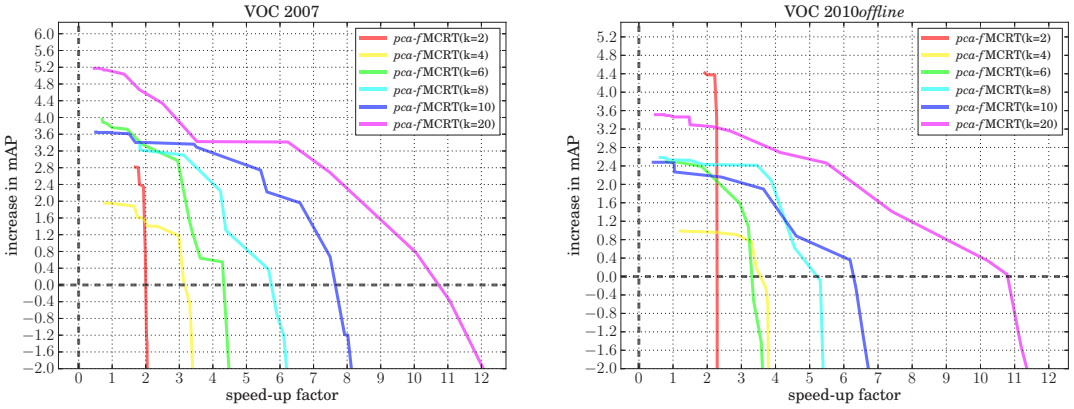
(b) PASCAL VOC 2010*offline*

Table 5.3 – Performance evaluation for the two designed methods *pca-e*MCRT and *pca-f*MCRT that use dimensionality reduction.

Run-time Evaluation

Table 5.3 further summarizes the speed-up for *pca-f*MCRT . The best run-times is obtained using *pca-f*MCRT which combines both ideas of a fast tree traversal technique and increasing the feature dimension depending on the depth of the tree. For a similar mean average precision than OvA, this system can be up to nearly 11x faster than its baseline method. It is also faster than *f*MCRT for both datasets. The relative speed-up to the baseline is more important with increasing number of classes k as pointed out before in Sec. 5.4.1.

Fig. 5.15 visualizes the trade-off between detection performance and speed-up for the settings in k and both datasets. The best detection performance is obtained when choosing admissible heuristic which are the starting point at the left top most point in each curve. But these heuristic do not allow to run the detector faster than the baseline except for $k = 2$. The more the heuristic are discriminative, the faster the run-time gets exactly as in Sec. 5.4.1. The plot for $k = 20$ has the best trade-off between speed and accuracy for both datasets. It shows that the system works better when dealing with many classes. The best detection scores of the PCA based system



(a) (b)

Figure 5.15 – Plot showing the trade-off between the relative mAP and the relative run-times of the various systems. The values are relative to the baseline method of OvA. A OvA system has the operating point at a speed-up of 1x and a relative mAP value of 0. The systems improve in detection times if their speed-up factor is superior to improve 1 and in detection performance if their relative mAP is bigger than 0. (a) PASCAL VOC 2007 (b) PASCAL VOC 2010*offline*.

shown in Fig. 5.15 are similar to the scores obtained with *pca-e*MCRT . This affirms as in Sec. 5.4.1 that the heuristic are admissible not only to the training set but also transfer very well to the test set. Using admissible heuristic allows to gain a modest speed-up for the same performance.

5.5 Conclusion

The hierarchical model described in Chapter 4 had an important disadvantage: Its detection time was linear in the number of classes with a constant factor bigger than the one for OvA. This is due to the additional number of filters of the super-classes in the tree which adds to the k leaf nodes. In this chapter, we have shown that we can use these additional filters to our advantage and achieve better run-times than the baseline algorithm. To this end, we formulated the detection task as the new objective of finding the highest scoring path. This can be done by a tree traversal algorithm detailed in Sec. 5.2.3. This allows to evaluate only certain paths and nodes in the tree. The root filter is always evaluated and the closer the filters are to the root node, the bigger is the chance of being traversed. To gain more in run-time, we reduced the feature dimensions of these super-classes. Also, they do not need to capture the rich characteristics as they model many corresponding child classes simultaneously. But the leaf nodes keep their full feature dimension.

This led to several variants of MCRT. The use of admissible heuristic has shown to slightly improve run-time over the exhaustive search without losing in accuracy. When the heuristic are less admissible, the run-time improves which reduces the mean average precision of the multi-class detector. Thus, we can balance both criteria depending on the application's requirements. We experimented with maximally 20 classes and noted

that the speedup improves with k. Reducing the feature dimension allows to have higher speed-ups for similar performances than OvA. The contributions of this chapter focused on accelerating the detection time. We proposed to do so by using our tree traversal algorithm. It is adapted to our needs: the ranking constraints from chapter 4 force the right class to have the highest score which is exploited in the algorithm. The classification constraints remove background regions as soon as their optimistic confidences drop too low. We also suggested the use of feature reduction which can be combined with the tree traversal mechanism. Finally, the training procedure for determining admissible heuristic showed to be effective as the detection performance keeps similar to an exhaustive evaluation of the tree.

In the next chapter, we would like to experiment with more recent object descriptors namely the deformable part model (DPM) of [Felzenszwalb et al., 2010a]. This descriptor relies on the HOG model to describe global object's appearance but also the local parts of it. It has a higher feature dimension and is definitely more complex to train.

CHAPTER 6

Future Work: Tree of Deformable Part Models

Contents

We used the famous HOG features when we reported the results for detection performance only in Sec. 4.4. The same was true when we evaluated the next contribution concerning the run-time of our system in Sec. 5.4. We used the HOG features because of its popularity especially between 2004-2011. It is a well known feature in literature and used by many research groups. The performance of this object descriptor is outperformed by more elaborated ones. Our complete problem formulation is transparent to the chosen descriptor. We implemented and experimented with the deformable part model again because of its success in the research community.

Before describing in more details our extension to the deformable part model [Felzenszwalb et al., 2008b], we give an introduction into the detection framework upon which we built our own. The deformable part model (DPM) is a top performing object detection system. Many approaches that performed among the best on PASCAL VOC datasets were inspired by this baseline system. Using a relatively simple descriptor, it combines global and local appearance information with deformation costs of part locations. The training time and detection time is small compared to most other systems. Moreover, the code is publicly available under [Felzenszwalb et al., 2008a]. We ported the MATLAB pieces of the code into a C++ implementation which can also run on multiple cores. The DPM defines more generally a new feature descriptor. Our multi-class problem formulation is independent of this descriptor and we used the HOG and DPM descriptors in our implementation.

Our contributions are manifold. The biggest one is the presentation of a hierarchical deformable part model (DPM). Every super-class and class is modeled using global appearances and deformable parts. We show how to optimize the problem statement in chapter 4 in the primal. Also, we illustrate the challenges linked to an increased memory usage. Finally, the integration of the DPM principles leads to a relaxed hierarchy that is one class is associated to several paths in the tree.

Unfortunately, we were not able to create reasonable results on time. The learning phase requires access to powerful computing resources concerning the memory. We leave this part for future work and mention other possible research directions in Sec. 7.2. Before diving into the description of the multi-class DPM, let us first review the HOG detector.

6.1 The Deformable Part Model

The system described in Sec. 2.2.1 covers the complete object region and is best suited for capturing global object appearance. Most objects modify their appearance due to their deformable parts. Naturally, the DPM extended the previous system to capture these part locations and deformations. It combines an appearance model with a deformation cost.

Parts

The global appearance model described by HOG features is called the root model. There exists P parts in total *e.g.* $P = 6$. For training, only weak annotations are necessary that means the system does not require part level annotations nor the number of parts. These parts are located within the root model. Their locations $p_i = (p_x^i, p_y^i)$ at training and test time are not given but inferred. The appearance of the parts are also described by the HOG features. The feature descriptor of a part i is defined by $\phi_{\text{app}}(x, p_i)$. This descriptor extracts a local patch of features around p_i inside the HOG features x. These features are obtained at twice the resolution of the root filter as it has shown to improve performance.

Deformation Cost

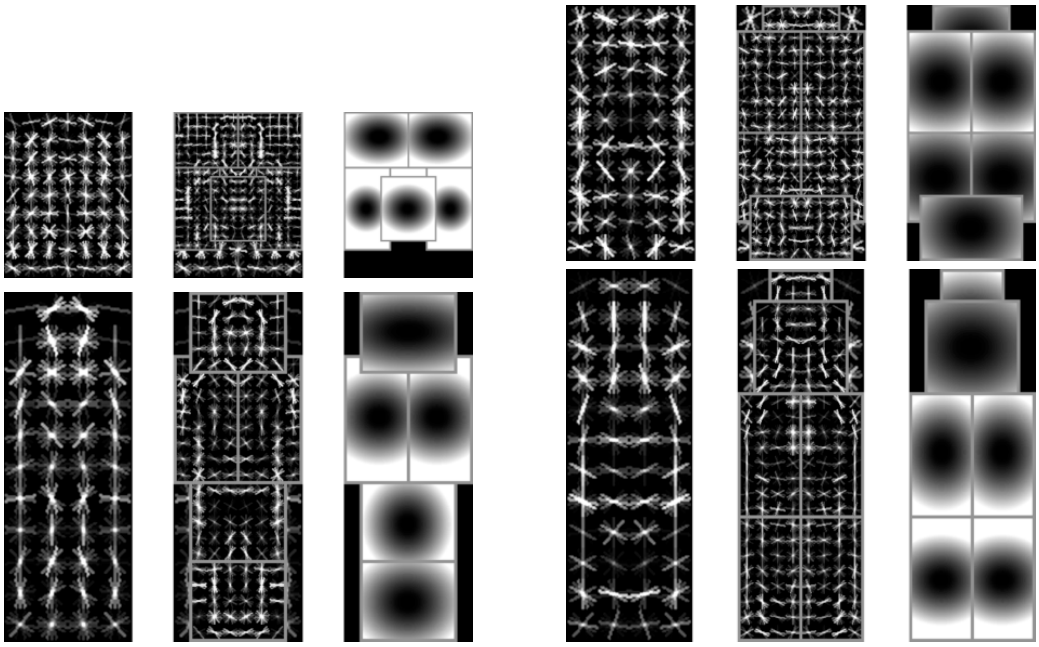
The parts within the root model do not have a static position. The part placements are constrained by a penalty function to get coherent part positions. The ideal position is called the anchor position $a_i = (a_x^i, a_y^i)$. Every deviation from the anchor position comes with a cost. The displacement feature vector is given by:

$$\phi_{\text{disp}}(p^i) = [(a_x^i - p_x^i)^2, (a_x^i - p_x^i), (a_y^i - p_y^i)^2, (a_y^i - p_y^i)].$$

At detection time, the part's location is chosen to maximize the following function:

$$\vartheta_i = \max_{(x,y)\in \mathrm{L}} \{w_i \cdot \phi_{\text{app}}(x, p^i) - v_i \cdot \phi_{\text{disp}}(p^i)\} \tag{6.1}$$

with L the possible part positions, w_i being a weight vector classifying the i-th part appearance features and v_i the learned penalty parameters punishing distant placements



(a) person (b) bottle

Figure 6.0 – Illustration of DPM models for two classes. Each row represents a component. In each component, the left figure represents the positive weights of the root filter, the middle one is the visualization of part appearance weights and the right one visualizes the deformation costs of each part. (a) Model for person class. (b) Model for bottle class (Courtesy of [Felzenszwalb et al., 2010a]).

of parts. Eq. (6.1) is a balance between appearance similarity of a part at a position in the image and its trained deformation. The function ϑ_i is known as the i-th part response.

Scoring Function

The final scoring function gives the confidence that a region describes an object. It is the sum of the individual part responses and the root filter score:

$$\text{score}(x) = w \cdot x + \sum_{i=1}^{P} \vartheta_i. \tag{6.2}$$

We omitted the bias term b for simplicity as this does not change the understanding of the DPM. A naive implementation makes the computation of the part responses unnecessarily long as one would have to check every position of the part in the image with respect to the root filter's position. The cost in terms of a deviation is combined linearly in Eq. (6.1) and is a quadratic function. Given these constraints, the DPM framework uses the distance transform (DT) [Felzenszwalb and Huttenlocher, 2012] that finds in linear time the optimal part placements depending on the root filter's location. The summation term can be treated independent of each other. In our C++ version, we used OpenMP to accelerate this computation.

Mixture Model

Objects appear in multiple views with specific appearances. Learning only one single model that captures the richness of the object's visual characteristics is highly challenging. This framework learns several models to better capture these appearance variations. Each model is called a component and there are a total of C components. The components are obtained by clustering the aspect ratios of the training examples. It is to note that during the hard training phase, the training example is assigned to the component that scores best. The same holds true during detection:

$$\text{score}(x) = \max_{i=1,...,C} \text{score}_i(x).$$

The score of a region is given by the best scoring component. During training, the components are not discriminated among each other as it is more important to get the right label than the component.

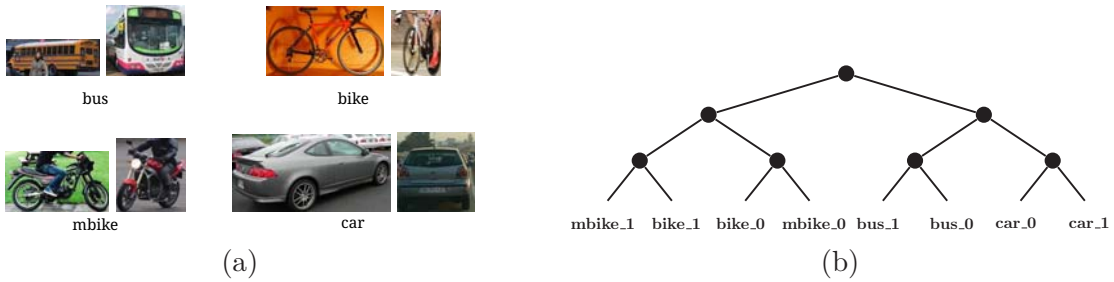
Training

The training protocol is very similar to the one presented in Sec. 2.2.1 which is definitely one advantage of the DPM framework. During the training stage, the examples of the objects are flipped vertically and summed together. This has the convenience to reduce the number of parameters as the right hand side of the annotation is no longer needed and to encounter for the mirrored version of objects.

The system has to face a lack of part annotations. The part locations in the annotations are latent and not available during training. After having obtained a random model, the anchor positions and appearance filters of the parts are given by areas having strong weights in the model's weight vector. The deformation parameters v_i are initialized to default values. The position of the parts in the dataset are now obtained by traversing the images and recording the configuration of the model that scored best. By fixing the part locations, the algorithm now trains all the parameters. It can be shown, that Eq. (6.1) can be written as the dot product of the concatenated weight vector $\bar{w} = (w, w_1, ..., w_P, v_1, ..., v_P)$ and root appearance features, part appearance and deformation features. The weight vector $\bar{w}$ is optimized using the linear SVM in its primal formulation optimized with stochastic gradient descent. The training protocol alternates between these two stages of learning and fixing part locations. The same holds true for the mixture model. The training examples are not assigned to components and are set to the component which yields the best score for that particular example.

6.2 The Hierarchical Deformable Part Model

We now describe the integration of the DPM into our multi-class tree formulation. The core modules of our system remain unchanged. At first a tree model is built only using the HOG features introduced in Sec. 2.2.1. We reduce the number of iterations in the bootstrapping to gain quickly a similarity matrix and the hierarchical classification model but otherwise the steps are exactly the same as with the HOG features.



(a) (b)

Figure 6.1 – Components of a class are treated as independent classes in our system. Thus, components of different classes can be grouped together. (a) We show examples of the training set which scored highest for their respective class. Each class was split into two components. (b) The tree obtained with our system for the 4 classes and 2 components. The left component is attributed the index 0 and the right one the index 1. The components of the motorbike and bicycle class share appearances and belong to the same super-classes.

At this stage, we add parts to this tree model. The parts are added as in the DPM for each node independently by localizing regions of high importance in the weight vector. The number of parts is fixed for each node *e.g.* 6. Defaults values are chosen for the deformation cost and the part appearance filters are cropped out of the tree model. Then, training proceeds by learning the appearance models and deformation costs and the ideal placements of the parts in the training data. The intermediate model is refined by detecting the latent locations which have a sufficient overlap with ground truth annotation of positive samples in the image. The hard negatives are obtained similarly by launching the detector on every image and collecting the wrongly classified samples until a predefined memory limit is reached. These iterations are repeated many times to improve the localization accuracy over positive samples and increase the discriminative power on background regions. In the following, we detail more the specific modifications needed in the learning system.

Multiple components We follow the same idea of creating components by clustering training samples. This is done in the beginning of the framework. Using the aspect ratio of the annotations, C components are created for each class y_i which results in $y_i, ..., y_{i+c}, ..., y_{i+C-1}$ with $0 \leq c < C$. Thus, we end up with kC classes by treating every component as a stand alone class. In other words $\mathcal{Y}^+ \equiv \{y_1, \ldots, y_{k+C}\}$. The training algorithm proceeds as usual. The random phase allows to create weak models for each class and component. There is a slight modification in the hard training stage: we assign the example of each class to the highest scoring component as the component annotation is not available. Once the components are determined for the annotations, we optimize the global filter w. This procedure is repeated several times and alternates between learning the weight vector and the component labels of the training samples. The component association is handled as a latent value in the learning process.

The tree is created without taking into consideration the class membership of the components. Each component is considered as a separate class and a similarity matrix between all these classes is constructed. Objects from similar viewing angles but different classes are thus grouped together if they share similar features *e.g.* the side

view of a bicycle and a motorbike can be grouped into one branch of the tree *e.g.* as shown in Fig. 6.1. Intuitively, one would assume that car and bus components are grouped together too. While this is not the case, the similarity between the right or left components of each class showed a strong resemblance in the similarity matrix.

Components can not be considered everywhere as classes. The ranking constraints in Eq. (4.7) force the decision hyperplane w to create a huge difference between scores of each class. This property is not desired when dealing with components of the same class. It is tolerable if during detection the wrong component is identified but the correct class is detected. To this end, we modify Eq. (4.7d) to avoid penalizing component confusion. This results in

$$\forall i \text{ with } y_i \in \mathcal{Y}^+, \forall y_j \in \mathcal{Y}^+ : \; w \cdot \delta\Phi_i(y_j) \geq \Delta(y_i, y_j) - \xi_{ij}, \tag{6.3}$$

which now includes a loss function $\Delta(y_i, y_j)$. This loss function is equal to zero if the two classes (y_i, y_j) are components that are derived from the same class and otherwise it is 1:

$$\Delta(y_i, y_j) = \begin{cases} 0, & \text{if } y_i \text{ and } y_j \text{ are derived from the same class} \\ 1, & \text{otherwise} \end{cases} \tag{6.4}$$

Parts In Sec. 4.2, we defined filters $w = \{w_1, w_2, ..., w_i, ..., \ldots, w_{|\mathrm{T}|}\}$ where each filter w_i is attributed to a node. The features extracted in the image at node i that is $\phi_i(x)$ are not only limited to the HOG features described in Sec. 2.2.1. We augment these features with part appearance features and deformation costs inspired by the DPM model. Thus, each node's filter becomes a deformable part model. The problem formulation remains unchanged by this higher level descriptor modifications. The new weight vector can be simply written as a concatenation of the additional parameters: $w_i = (w_i^{\text{root}}, w_i^j, v_i^j)$ with $1 \leq j \leq P$ being the part index. Each node has P parts. w_i^j denotes the appearance filter of part j with its deformation values given by v_i^j.

Memory consumption Treating multiple classes in a joint learning framework poses the challenge of memory usage during detection but also during training. The complete scope of the training data is considered in such a framework which means loading into memory their corresponding features. There is a huge amount of negative training data available. The memory usage is limited by using boostrapping that means only considering a certain amount of *hard* negative samples during each iteration. Considering the PASCAL VOC dataset, it is possible to load all the 10,000 positive samples. The DPM framework uses up to 50,000 samples to learn each class and we followed their criteria but experimented too with more negatives.

Naively implementing our framework would quickly require large amount of memory space *e.g.* 10GB even when learning a single class. We illustrate this in the following on a single example using the simpler HOG features. When learning only one model *e.g.* bicycle, we have 353 training samples and in each iteration let us assume that we use 49,647 negative samples making a total of 50,000 samples. The one component bicycle model has dimensions 9x9 on the PASCAL VOC dataset.

We used as an SVM solver the structured SVM package from [Joachims et al., 2009]. The feature vector consists of an 8 byte representation of the feature vector and an

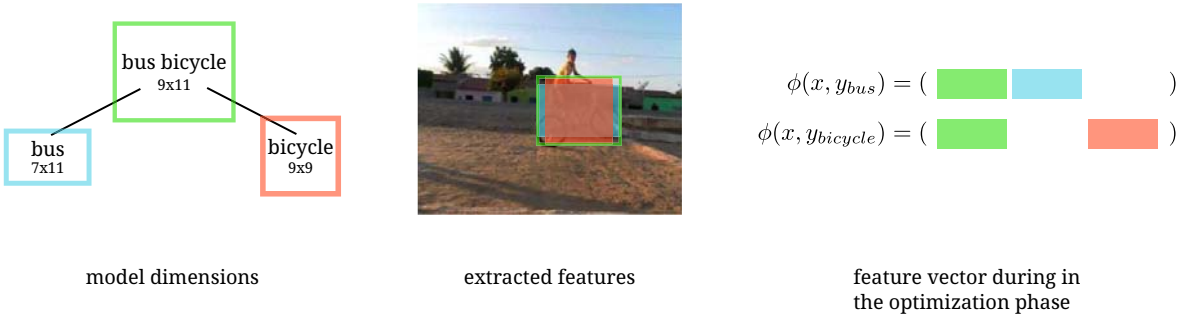


Figure 6.2 – To limit the memory usage, we do not save the features required for all nodes but construct it when needed from the enclosing feature descriptor. The left part shows the model dimensions for each node and class. The root node encapsulates its two child nodes. Therefore, we only save all the features contained by its dimensions instead of saving all the features for each node separately. This is reasonable as the root features, shown in green, contain the features from its child nodes shown in red and blue. As a consequence, it is possible to reconstruct the features $\phi(x, y_i)$ with $i = \{1, 2, \text{bg}\}$ for each node given the saved features during bootstrapping.

int value identifying the feature number. This was done especially for sparse features where we only need to save the values that are unequal to zero. In short, for one feature value we need 10 bytes of memory. Each example requires thus $9x9x31x10 = 25,110$ bytes or 25 kB. For one iteration, this makes a total of $50,000x25 = 1,250,000$kb or 1,25GB. This is very well acceptable for one class and makes our code possible to be trained on a simple machine.

Now let us assume we would like to train a multi-class detector of two classes {“bus”,“bicycle”}. The total number of examples is again limited to 50,000. The model dimensions for bicycle are again 9x9 and for bus are 7x11. The super-class filter which is as the same time the root filter grouping these two classes into a tree, has dimensions 9x11. This makes for $k = 2$ a total of 9x9x31 + 7x11x31 + 9x11x31 = 7967 features and requires 4GB of memory for all the examples. This makes it still possible to run the training process on a simple computer. It is to note that we neglected the space allocated by the feature vector and the constraints in the SVM solver.

The memory usage is dependent on the number of nodes |T| in the tree. We alleviate this dependency by only saving the feature vector extracted for the root filter. By definition, the root filter has the biggest dimensions in both directions and encapsulates all its smaller dimensional classes. The correct feature vector is constructed during the learning phase in the solver when needed by extracting corresponding regions in the saved features. Consequently, the feature size depends on the dimensions of the root filter. This induces a slower training algorithm as the correct feature vector is calculated every time when needed. We notice that the training time does not suffer considerably under this method and is necessary to keep memory usage low. Fig. 6.2 depicts the feature saving and construction stage. The complete green region is loaded by the structured learning algorithm. It is not needed to load separately the red and blue regions. The feature vector ϕ is constructed by extracting the parts of the green region adequately as shown in the right part of the image.

The HOG features are smaller than the DPM features. Every filter is equipped with a root filter, part filters and deformation parameters. For the root filters we proceed as before and only save the enclosing bounding box of all the classes. The part filters of a node can be spread anywhere in the image and not only in the vicinity of the root filter depending on the deformation cost. We save the features of the bounding box that surrounds all the parts of every node. This is efficient as usually the part filters overlap. Moreover, these part features take more in space as their have twice the resolution than the root filter. If we assume that the enclosing bounding box has the same dimensions as the root filter in the case of the single bicycle class, the part appearance and deformation features need (18x18x31 + 4)x10 $\simeq$ 100kB of memory space. The original DPM formulation uses flipped features for parts that are symmetrical. Therefore, we further need to save the enclosing window in its flipped version. For our training set, this requires approximately 11, 2 GB. The correct feature vector is constructed as before when needed out of the saved features in the optimization phase.

The issue here is that the box surrounding all the parts can get very large even for smaller number of classes. This is due to the fact that parts can be spread over the image but also that the top most node can have too large dimensions *e.g.* 15x15. In that case, only loading the features needs 30GB of space and not including all the space allocated for holding the high dimensional constraints. We found that using the DPM in this hierarchical classification context limits the scalability of the framework.

Solving in primal The weight vector w is a concatenation of the filters located in each node w_i, $1 \leq i \leq |\mathrm{T}|$. Each of these w_i is composed of a root filter, appearance filters and deformation filters as mentioned before. The choice of the features is transparent in our formulation which makes it efficient to test new descriptors such as those in [Felzenszwalb et al., 2010b, Girshick et al., 2014]. In the case of the DPM, we first solved our structured optimization problem in the dual form. The results were highly unsatisfying and we experimented with other solvers. In our case, the primal solver of [Felzenszwalb et al., 2010b] produced satisfying results when optimizing the tree's weight vector w.

The primal solver is presented in 3.1.1. It is similar to a simple stochastic gradient descent algorithm. We found it nonetheless very powerful for optimizing w. There are still small modifications compared to a naive implementation. First, it is important to permute the examples as to avoid getting stuck in a local minimum. A cache allows to set aside data that successively gets correctly classified.

Furthermore, the solver uses different values for the regularization term λ and learning rate ν_t, introduced in Sec. 3.1.1, depending on the feature type in w_i and even bias term of the root filter b. In other words, the parameters (λ, ν_t) are set to different values for the root appearance filter w_i^{root}, part appearance filters w_i^j with $1 \leq j \leq P$ and the deformation cost v_i^j. We keep the same parameters between the nodes. The regularization parameter λ is high for deformation cost and low for the bias term of each filter. This implies that we try to keep the deformation flexibility as small as possible. This is useful as to accelerate the distance transform. The parts are no longer spread over the whole image and finding the best part locations in a close neighborhood is often enough for comparable performance as detailed in [Gadeski et al., 2014]. The

learning rate ν_t is chosen higher for the deformation cost than for the appearance parts of the filter. We use Alg. 1 as a primal SVM solver and integrated these feature specific type of parameters.

6.3 Future work

For future work, we need to improve the memory efficiency of the program. But this is not the only limitation why we could not present results. We recoded all the parts written in MATLAB in the original DPM code in C++. Our results for a single class are the same as those obtained by the DPM code. Next, we integrated those parts into our framework. Although the complete code is finished, it takes a considerable amount of time to configure the set of parameters for the multi-class case. It less suitable to automatically configure these parameters through the use of a validation set as learning takes too much time. Finally, we found that simple modules *e.g.* image resizing algorithm or the precision of the features can have a signification influence on the final performance.

CHAPTER 7

Conclusion and Perspectives

Contents

We have presented a new multi-class detection framework along with its training protocol called MCRT. It can balance execution time and accuracy depending on the available resources and the application's constraints. It builds the heart for diverse computer vision applications. But many more modules should be used to improve the two critical requirements namely speed and accuracy such as scene geometry or contextual information. Previous works focused on improving a single-class detector without bothering much about execution time. The multi-class methods are often extensions of a single-class approach and are hardly able to integrate with newly discovered features. Other methods scale impressively with large number of classes *e.g.* [Dean et al., 2013] but require huge memory footprints during detection, have moderate performance similar to OvA and are less suitable for application with little values for k.

We focused our work on designing a new multi-class detection model which works fast even with little values for k. A parallel research direction was to show the relevance of ordered class knowledge in object detection and the advantage that comes with it. We were focused on the real-time capability of multi-class object detection. At the time of starting this work, we found the [Song et al., 2012] method and the [Razavi et al., 2011] suitable for the task of real-time applications. The former requires the use of a reasonable GPU card not always possible in practice and the latter is constrained by a Hough based detection system. This led us to the design of our object detection system. This chapter will conclude our work in Sec. 7.1. We will have a look on the

perspectives of our work, mention various points that merit some future considerations and evaluations that we did not find the time to pursue in Sec. 7.2.

7.1 Conclusion

In the future, computers will have the capacity to understand images and videos. This fascinating feature opens the door for many great domains such as robotics, autonomous driving or video surveillance. One brick towards advanced intelligent systems is identifying all the instances of classes of objects in an input. This consists in returning a set of rectangular bounding boxes and class labels with the target objects in the image. The context of this thesis is to do this in arbitrary types of images and object classes. Multi-class methods are common in image classification where the number of categories are large. In object detection, the methods are highly concentrate to only one mastering one class at a time and are hardly able to adapt to novel descriptors. Other methods scale nearly linearly with the number of classes but require large memory usage and run-time for a small amount of classes as it is possible in our context. This lack of an efficient multi-class detection system inspired us to derive an alternative object detector. This is one module in the chain of a final detection framework which depends on the task.

A tree of classifiers is exploited in many works for image classification. In object detection this is not a common practice as one has to further cope with the background class. We showed a completely automatic way to build and learn such a multi-class object detector. The challenge in the optimization is to reject background samples and find the best class. To this end, we combined multiple constraints consisting of ranking and classification during optimization to obtain our filters in the tree. The detection goal is formulated as finding the best scoring path. The detection with trees suffered from long detection times. An efficient tree traversal method which can be combined with feature reduction allows to remedy this critical problem.

We started this dissertation with an introduction to the context of our work. This is followed by a dense review of related previous works. We introduced the important datasets. The number of classes can range from a few classes $k = 3$ to many more $k = 20$ and to a much larger scale $k \geq 2000$. Fast algorithms can be developed on sophisticated hardware mostly by exploiting the principle of parallelism. Many authors find efficient ways to accelerate algorithms often by calculating a proximate solution to it. The use of hash tables to compute convolutions between filters and image features has found popularity these recent years. Learning huge amount of classes can benefit from transferring the knowledge of a set of base classes to other ones. It notably permits to reduce the number of training data. Finally, image features are not the only cue to exploit for object detection but the use of a context module has shown to be beneficial.

Chapter 3 introduces the basics of training and learning with support vector machines. The classification problem can be solved in primal or dual form using available solvers. The chapter closes with a review of structured support vector machines which builds the foundation of our optimization stage. We noted that the traditional multi-class classification methods are unsuited for object detection as they assume all classes being able to be modeled. This is true for decision trees where the bottom nodes rep-

resent all the possible classes. This lack for handling the background class required the design of a new optimization procedure.

The next chapter introduces our multi-class ranking and classification technique (MCRT) which is built to perform better than OvA. The approach distinguishes between k object classes and one background class. The input image is scored by all the k classes and only if the highest scoring rank is bigger than a threshold (*e.g.* 0), the input is classified as that class label. Otherwise, the input is rejected as a negative region. The scores must be comparable and the right class needs to assign the best score. This formulation alone does not necessarily improve the performance over OvA. The classes are grouped into a tree structure. The nodes are linear filters which score input features. The bottom nodes contain classes. The score of a class is given by the sum of the nodes' scores from the root node to the leaf node lying on classes path. The numerous additional filters help to produce better results compared to the k filters in OvA.

Training this tree structure is a challenging task as it has to cope with all classes and training samples. We optimized the tree by combining the classification and ranking formulations into one optimization problem which is a structured problem. We solved it using a cutting plane algorithm to reduce the large number of constraints. The tree is learned automatically by first constructing quickly a OvA multi-class detector which is used to calculate a similarity matrix between the classes. We proceed by recursively applying a spectral clustering algorithm. The performance affirmation confirms our assumptions: MCRT attains higher accuracy measured in mAP. This holds true over several datasets, number of classes k and even combination of classes. The automatically constructed tree is not optimal but much better than the worse and closer to the best performing one. Finally, MCRT has the convenient feature of transfer learning. The algorithm handles in a stable way when a class has little training data. OvA is unable to learn a stable detector for these classes but MCRT uses the intermediate filters to learn a powerful classifier.

But more filters mean more convolutions. Detection time is as much of a criteria as accuracy. The class label is determined by the object category with the maximum score. The naive approach requires scoring every node in the tree thus traversing every path. Knowing the exact scores of every class is not necessary. Seeing the inference as the goal to find the target leaf node, Chapter 5 describes an adapted tree traversal algorithm to speed-up the detection rates. The algorithm estimates at every node the best possible score it can reach by following a specific path. The scores of the paths are refined every time the algorithm goes one level deeper into the hierarchy. At every iteration the most optimal path is selected and the algorithm goes into that direction. Evidently, the first levels of the tree are traversed more frequently. We opted to reduce the number of features of these nodes by linearly increasing the feature dimensions towards the leaf nodes. The learned heuristic are admissible to the training set. Experiments showed that this holds true in most cases during detection on test images. The accuracy of MCRT using admissible heuristic and the exhaustive algorithm are very close while the former has slightly lower run-times. We obtain a magnitude of speed-up for $k = 20$ by picking less admissible heuristic and aiming the performance of OvA. The gain in run-time increases linearly for the PASCAL VOC dataset with the number of classes. With reduced features, the speed-up is slightly higher, up to nearly 11x. This modest

speed-up can be explained as the accuracy of the tree decreases when having less features.

In chapter 6 of our work, we implemented the deformable part model into our framework. Until then our evaluations relied on the much more simpler HOG features described in [Felzenszwalb et al., 2010a]. The challenges were manifold as we aimed to reproduce the exact same algorithm as with the code of the original authors for $k = 1$ with regard to the training and testing procedure. This necessitated to incorporate the concept of having multiple components. Each component can be seen as a characteristic view of the training samples. To stay close to the original code, we opted for a primal solver which we adapted to work with our optimization module. Every node was augmented with parts and associated deformation costs. We met an unexpected issue with the memory usage. The size of features that need to be loaded for every single annotation is large. In the end, we encountered difficulties already for a small amount of classes and were not in the position to finalize the experiments.

In summary, these concepts led to the following contributions:

Hybrid learning It is common practice to formulate a ranking problem when aiming to find the closest classes to an element. The object detection task is usually formulated as a classification problem recognizing between foreground and background objects. Our problem formulation is a combination of these two types of constraints.

Detection by ranking and classifying MCRT ranks and classifies as we intended to keep the scores between classes comparable and meaningful. The highest scoring category is the closest class to the input.

Hierarchy of classifiers for multi-class object detection Decision trees are found in many research papers. This principle is not naively applicable to object detection due to the highly variable background class. We showed a concept on how to learn a tree of classifiers where the sum of scores of each path builds the individual class scores.

Hierarchy to improve performance A crucial message of our work is that hierarchical methods can be a step forward for better detection performances especially when compared to a flat structured OvA method.

Joint optimization The optimization module simultaneously learns all the filters using information from the complete training data. Instead of iteratively learning one filter at a time, this module solves a structured problem to obtain the global weight vector concatenating the weight vectors of the individual filters.

A framework transparent to feature descriptor Many of the developed multi-class methods are adapted to one specific type of features *e.g.* contour features. Our formulation is independent of the object descriptor as long as a convolution can be computed with a linear filter. This keeps the framework open to future descriptors *e.g.* the region features obtained with R-CNN [Girshick et al., 2014].

Fast inference for tree of classifiers As by definition, a tree of classifiers has a large amount of filters which increases rapidly with the number of classes. Consequently, the detection time suffers. With the proper inference formulation, this

is no longer an inconvenience. Our detection formulation naturally allows the use of a tree traversal algorithm. We proposed one that is able to achieve high speed-ups for comparable performance as OvA and requires a short training time.

Trade-off between accuracy and run-time The algorithm allows to increase the speed-up compared to OvA. Moreover, by choosing appropriate heuristic it is possible to balance between performance and run-time. MCRT does not have a constant speed-up but this property can be adapted based on the application's requirements.

Hierarchical method to learn when having little training data The experiments showed that hierarchical methods attain reasonable performance even when having little training data at disposition for a class. Furthermore, we conclude from the experiments that MCRT converges more quickly to its best possible performance when we increase the training data. This makes hierarchical methods interesting for large datasets where it is difficult to collect large amount of data *e.g.* the $100,000$ class dataset mentioned in [Dean et al., 2013].

Our work allows to understand the advantages of hierarchical structured support vector machines in multi-class object detection. The framework introduced different techniques to overcome the challenges of a hierarchical detection model. In future, we hope to be able to scale these kind of algorithms to more than hundreds of classes with quick learning phases and to introduce the possibility of incrementally adding classes. These changes will make hierarchical knowledge sharing even more attractive due to reduced training and detection times. Finally, the additional knowledge of super-class memberships differentiates these methods from a simple accelerations of many single-class detectors. We hope that it can find application in many various fields independent of the number of classes required such as searching images for all possible objects by Google image search, video-surveillance in parking areas, advanced driver assistance systems or learning incrementally new object categories by autonomous intelligent agents.

7.2 Future Directions

The MCRT framework can be extended in many ways to improve both criteria of accuracy and run-time. The goals for future research are manifold. It is essential to pay attention to the scalability of an algorithm but it should also be efficient for small number of classes. The training time is another critical point for algorithms. Learning one class should be possible in few hours and many more classes in less than one week. Any hierarchical detection system has to outperform the traditional one-versus-all method at least in accuracy because of its additional knowledge.

7.2.1 Accuracy of Detection

Relaxed Hierarchy

We separated the classes at each level in the tree into disjoint sets. Two different nodes do not share one or more similar class labels. Each path leads to a different object

category thus we have k leaf nodes. Another way to structure the tree is to create a relaxed hierarchy. We have introduced the concept in Sec. 3.2.2 and an example is shown in Fig. 3.6c. The leaf nodes at the bottom of the tree are attributed to more than one intermediate nodes. Our framework easily permits to experiment with this concept. We did this in Sec. 6.2b where the classes are partitioned into components. These components create an independent leaf node and can be grouped with components of other classes. Therefore, the bottom layer consists of leaf nodes representing the same classes. One can imagine different other ways to partition the data and classes in the tree *e.g.* as in [Marszalek and Schmid, 2008]. The work of [Aghazadeh et al., 2012] and [Divvala et al., 2012] showed that partitioning the examples into different sets indeed improves the detection accuracy.

Joint Tree Learning and Optimization

Our tree is created by partitioning recursively a similarity matrix between the object classes. This is an intuitive technique to order the classes into a hierarchical structure where similar classes share super-classes. Many other approaches are based on this procedure as we have seen in Sec. 4.3.1. It is convenient in that it uses the very same features that are used during final detection to build the simple detectors necessary to create the similarity matrix.

The downside is that the tree does not directly reduce the classification error during the optimization phase. The optimization problem in Eq. (4.7) assumes a fixed hierarchy. The work of [Salakhutdinov et al., 2011] assumes latent knowledge of the tree and filters. It iterates over fixing the tree and learning the filters and defining a new tree with the learned filters. We avoided this method for its long training times but find it crucial to deduce an optimal hierarchy during training. One could also imagine learning the hierarchy as in [Yang and Tsang, 2012] where finding the optimal tree is solved in a relaxed integer program jointly with finding the most discriminative weight vectors.

Integration With a Context Filter

The objects are organized into a tree structure. Each node in the tree consists of a filter and the leaf nodes represent the final classes. The intermediate nodes are called super-classes. This concept of ordering the classes into bigger groups made us passionate about the method. Contrary to models without a hierarchy, this method gives us the additional information about the semantic class membership of an object. Humans unconsciously identify the semantic class of an object. For instance, an input identified as a van can belong to the group of vehicles. The semantic knowledge of objects should be exploited by a higher level module. This extra information is not immediately of use in object detection but we believe that it can serve as a future source of information *e.g.* in combination with a context module such as augmenting the features of the one presented in [Felzenszwalb et al., 2010a] or as a cue to an image classifier.

Incremental Learning

Learning large number of classes remains a challenge with hierarchical support vector machines for object detection due to time and memory usage. We noted in Sec. 4.4.4 that MCRT is able to learn a model with just a few number of training samples. During training, the knowledge of all the classes helps those classes with little annotated data. We are curious to explore additionally the principle of incremental learning that is adding one class after another to the tree. The idea was already explored in works *e.g.* [Opelt et al., 2008, Fidler et al., 2009]. We could even consider learning a base tree *e.g.* for $k = 200$ most representative classes and incrementally add classes that share properties with those base classes. For the incremental learning we would no longer load examples of the previously added classes. Thus, we hope to be able to facilitate scalable training of this framework.

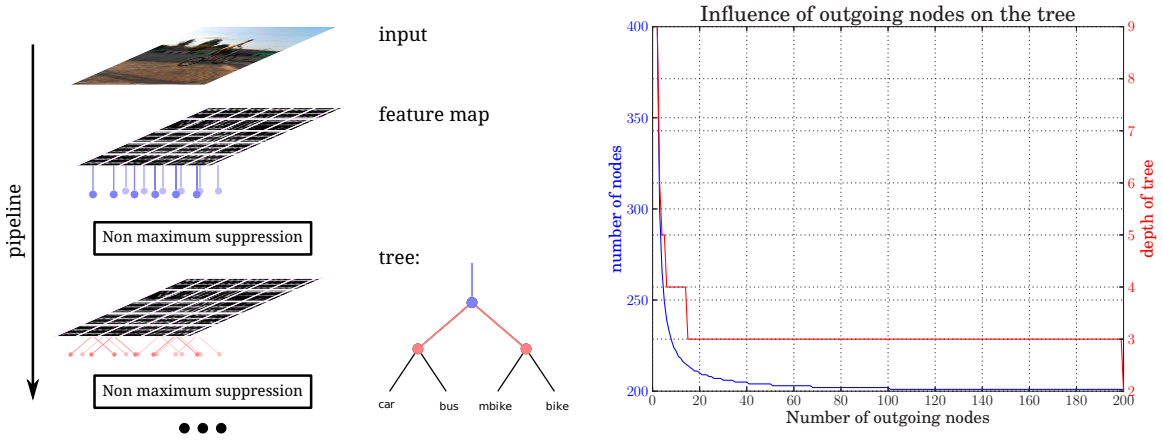
Training MCRT With Sampled Cuts

An orthogonal improvement to the idea of incremental learning is to accelerate the computation for finding the most violated constraint. This function is called for every example and for each example the most similar class is looked for. It is called many times during the optimization phase. With increasing number of classes and training data, this stage becomes very time consuming. We can remedy this limitation by actually sampling the training samples and classes. We do not go over all the training examples but randomly select a subset of those. For each selected example, we can apply the same idea to the number of classes and randomly pick some of them for comparison. [Yu and Joachims, 2008] applied this principle for sampling training examples only. Their approach resulted in a fast optimization procedure that is able to create approximately the error rates close to the exact solvers.

Richer Features

Our core evaluations are done using the histogram of oriented gradients. These features are very fast to obtain and allow us to control the training period. This object descriptor has lost in its popularity and was replaced by more efficient ones. The deformable part model is one of them which greatly outperforms the HOG features for complicated datasets such as PASCAL VOC. HOG features remain powerful for rigid objects. In Chapter 6, we mentioned some of the challenges we had to transfer the deformable part model framework into ours. Controlling the memory consumption is left for future work. One way to do this is to limit severely the distance transform to the neighboring cells of the anchor position of the parts. Parts of all the classes and super-classes would be close and overlap each other. The bounding box enclosing all these parts would be definitely small and require a lower memory usage than with a looser deformation penalty.

Other promising features are the rich features presented in [Girshick et al., 2014]. An image patch traverses a neural network and results in a feature vector of dimension 4096. This feature vector is scored by k filters as with OvA and the class label is determined by the highest scoring filter. We are curious about the possibility to explore our hierarchical classifier on the output of the neural network.



(a) (b)

Figure 7.1 – (a) We could imagine to apply a non-maximum suppression after evaluating each level in the tree. This would save the calculation of partial scores for the suppressed positions. (b) We limited the hierarchical structure to binary trees like in many other research ares. In the future, it could be worth investigating trees with many more outgoing child nodes. This would reduce the depth and total number of nodes and enhance the training time. The detection time can be kept small by evaluating efficiently all child nodes *e.g.* using a hashing based procedure of [Dean et al., 2013].

7.2.2 Execution Time

Use of Collateral Features in the Tree Traversal Algorithm

An input patch is scored by a node and its value is added with a heuristic to build an optimistic estimation of its final score for the node's child classes. This heuristic is an upper-bound of the sum of the scores of the remaining child nodes. This heuristic value is fixed during training as shown in Sec. 5.2.5. It is independent of the input features. Inspired by the branch and bound technique of [Lampert et al., 2009a], this can be modified to use some coarse image features and quickly calculate a tighter upper-bound score.

Another idea to restrict the heuristic is to use already collected information. During detection, an input traverses not only nodes lying on the correct path but is scored by neighboring filters too. Their scores are collateral information and come for free. These values can be put into a novel feature vector. We could learn a regression function estimating a tighter upper-bound of the remaining score depending on the input. Both methods are complementary.

Early Non-maximum Suppression

Currently, we apply a non-maximum suppression (NMS) at the end of the detection chain. We can imagine to do the same at every time we pass a node. This would slightly modify our code. Currently, the algorithm selects a position in the image and passes that input patch through the filters in the tree. The final score for each position is calculated one after another on the feature map. Now, we would have

to calculate the intermediate scores for all the positions before going deeper into the hierarchy. After evaluating each level in the tree, we remove multiple positive responses and reject positions surrounded with negative scores. This principle is analog to the second contribution in the work of [Yan et al., 2014]. We expect this iterative NMS to speed-up the execution time. An example pipeline is shown in Fig. 7.1a.

Increase the Number of Outgoing Nodes

This thesis considered binary trees. That is the number of outgoing child nodes is fixed to 2 for each node. From Sec. 5.2.4, we know that the depth of a tree scales logarithmically with the number of classes. For instance for $k = 20$, the depth can be rounded to 6. For $k = 100,000$ as in [Dean et al., 2013], this would result in a depth of 17. We chose a binary tree as to have fast run-times. At every node, two child nodes are scored. More outgoing nodes would require scoring more filters at each level. Using Eq. (5.9), we note that the total number of nodes $|\mathrm{T}|$ decreases quickly with the number of child nodes. This results in a smaller weight vector as there are less filters. Fig. 7.1b plots the influence of the outgoing child nodes on $|\mathrm{T}|$ and tree depth.

Does this necessarily mean slower execution times? We saw in the literature review in Sec. 2.2.2 two methods [Dean et al., 2013, Sadeghi and Forsyth, 2014] both based on hash tables that evaluate very quickly several filters by finding the most similar filters to a feature vector. We could apply this idea to score all the outgoing nodes nearly instantly and eliminating the dependence on this characteristic. The smaller weight vector facilitates the training process and MCRT and goes one step towards enabling large scale learning with hierarchical support vector machines.

CHAPTER A

Résumé en français

Détection efficace des objets multi-classes avec une hiérarchie des classes

A.1 Résumé

Dans cet article, nous présentons une nouvelle approche de détection multi-classes basée sur un parcours hiérarchique de classifieurs appris simultanément. Pour plus de robustesse et de rapidité, nous proposons d'utiliser un arbre de classes d'objets. Notre modèle de détection est appris en combinant les contraintes de tri et de classification dans un seul problème d'optimisation. Notre formulation convexe permet d'utiliser un algorithme de recherche pour accélérer le temps d'exécution. Nous avons mené des évaluations de notre algorithme sur les benchmarks PASCAL VOC (2007 et 2010). Comparé à l'approche un-contre-tous, notre méthode améliore les performances pour 20 classes et gagne 10x en vitesse.

Mots-clés : détection multi-classes d'objets, classification hiérarchique, inférence rapide, arbre de classifieurs, parcours d'arbre, apprentissage hiérarchique, SVM structuré

A.2 Introduction

La détection d'objet de différentes classes dans les images présente plusieurs difficultés. L'algorithme d'apprentissage doit pouvoir traiter de données avec des variations inter et intra-classe. De plus, le temps d'exécution ne doit pas croître de manière exponentielle avec le nombre de classes.

Dans ce travail, nous proposons une approche nommée *multi-class classification and ranking tree* MCRT améliorant la performance comparée à un-contre-tous SVM (OvA). Le choix du descripteur est transparent pour notre algorithme.

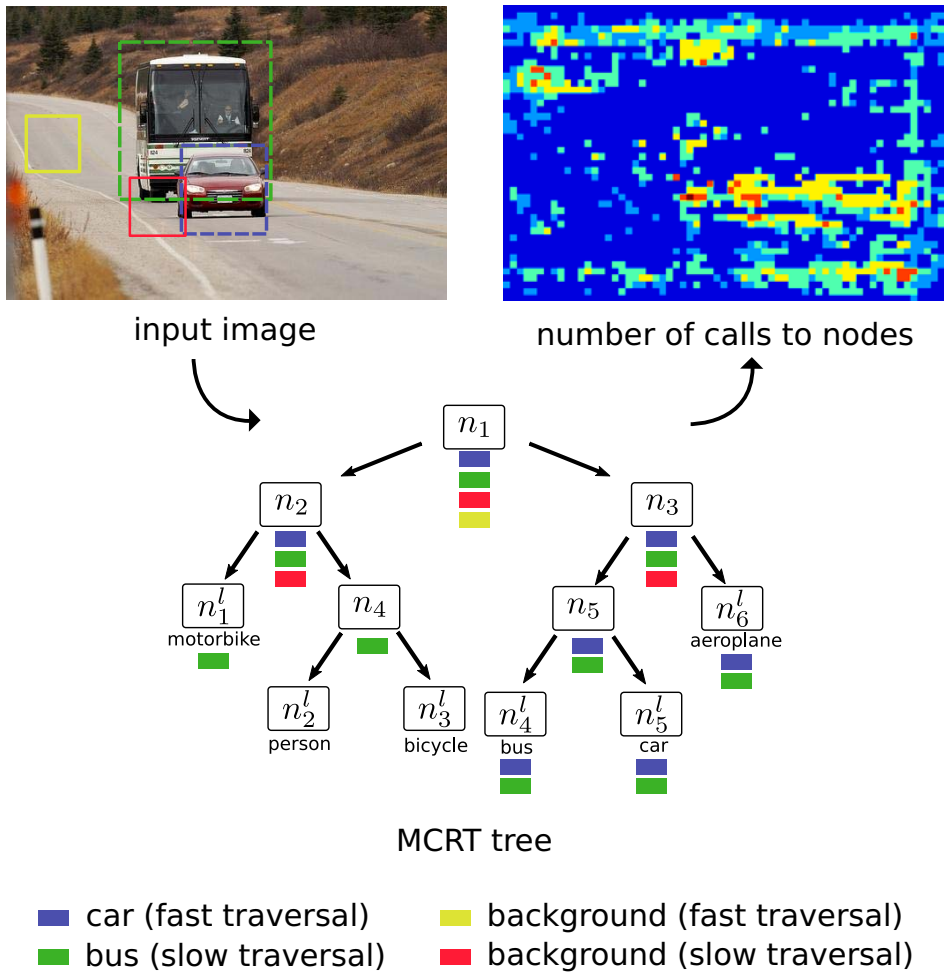


FIGURE A.1 – Cette figure illustre la traversée d'un arbre pour les régions d'une image indiquées en haut-gauche . Le fond en jaune est rejeté après le passage par n_r. Celui en rouge est évaluée par plus de filtres. La carte de chaleur illustre le nombre de filtres évalués pour chaque pixel de l'image en entrée où le bleu signifie qu'un seul filtre est appliqué.

Pour accroître la performance de détection nous regroupons les classes de manière hiérarchique, ce qui permet de partager des exemples entre des classes. La ligne de décision finale est non-linéaire à travers les nombreux classifieurs. Pour éviter de créer un nœud supplémentaire pour la classe *fond*, nous combinons des contraintes de classification et de tri. Pendant la détection, MCRT trie les différentes classes et si le meilleur score est négatif la région est classifiée comme étant du fond. Une autre contribution est l'utilisation d'un algorithme de recherche rapide inspirée de A*, pour trouver la bonne classe dans l'arbre. A chaque niveau de l'arbre, le chemin produisant le plus grand score est calculée grâce à une heuristique dédiée. Cette approche est différente du modèle *coarse-to-fine* car les étages interagissent ensemble.

Cet article est organisé comme suit : la section 2 présente les travaux relatifs à notre problématique. La section 3 introduit la notion et le système de reconnaissance puis la section 4 décrit l'algorithme se basant sur ces éléments. La section 5 nous évaluons notre technique avant de conclure.

A.3 État de l'Art

Ces dernières années, un grand intérêt est prêté à la classification hiérarchique multi-classe. Les algorithmes nécessitent de traiter plus de données et de gérer la complexité d'apprentissage et de détection. Par la suite, nous passons en revue quelques travaux pertinents de l'état de l'art, très vaste sur ce sujet.

Détection d'objet L'idée de partage des exemples, en combinaison avec une approche de type "boosting", a été proposé par Torralba *et al.* [Torralba et al., 2007]. Des classifieurs communs entre classes permettent de contribuer aux scores d'un sous-ensemble de classes. Salakhutdinov *et al.* [Salakhutdinov et al., 2011] a proposé de regrouper les classes en fonction de leurs similarités et des distributions de leurs exemples. Les filtres de l'arbre sont entraînes avec L-SVM [Felzenszwalb et al., 2010a]. Contrairement à notre approche, le temps de détection est proportionnel aux nombre de nœuds dans l'arbre.

Au lieu de travailler au niveau des *caractéristiques* partagées par les exemples, d'autres approches combinent directement les parties communes entre les catégories. Razavi *et al.* [Razavi et al., 2011] créent un dictionnaire commun où les parties votent pour les différentes classes. Les auteurs de [Dean et al., 2013] utilisent le modèle par parties déformables en appliquant une fonction de hachage pour accélérer le calcul du produit scalaire entre les modèles de parties et l'image. La complexité de leur approche est indépendante du nombre de classes mais ne permet pas de choisir entre la performance de détection et rapidité d'exécution.

Classification hiérarchique avec SVM La classification hiérarchique a été notamment appliquée pour la classification des images. Par exemple les travaux [Bengio et al., 2010, Griffin and Perona, 2008] ont explorés les arbres de décisions pour faire une rapide catégorisation des images. L'apprentissage de filtres binaires s'effectue selon une stratégie *top-down*. Dans [Deng et al., 2011], l'arbre et ces filtres sont appris ensemble. Structured SVM [Tsochantaridis et al., 2004] est une approche pour apprendre tous ces filtres simultanément. Ces approchent ne peuvent pas être appliquées dans notre cas car il n'existe pas de solution pour modéliser explicitement la classe fond de la scène (c'est-à-dire les parties de l'image ne contenant aucun objet d'intérêt). Ainsi, nous proposons dans cet article un mode de classification avec une classe négative dominante (le fond) comme c'est le cas dans la détection d'objet.

A.4 Système

Nous voulons classifier chaque position dans une image I appartenant à une des k classes positives $\mathcal{Y}^+ \equiv \{y_1, \ldots, y_k\}$ ou attribuer l'étiquette du fond $y_{bg} = -1$. Nous proposons un modèle combinant les techniques de *tri* et de *classification*. Pour chaque région la bonne classe doit avoir le meilleur rang entre les k catégories. Si le score est négatif, l'hypothèse est classifiée comme *background*. Notre algorithme MCRT attribue à chaque position dans une pyramide de caractéristiques un score et une étiquette. Dans notre système, nous considérons que la pyramide de caractéristiques est l'ensemble de caractéristiques pour différentes échelles de l'image. Le score $score(x)$ pour une position

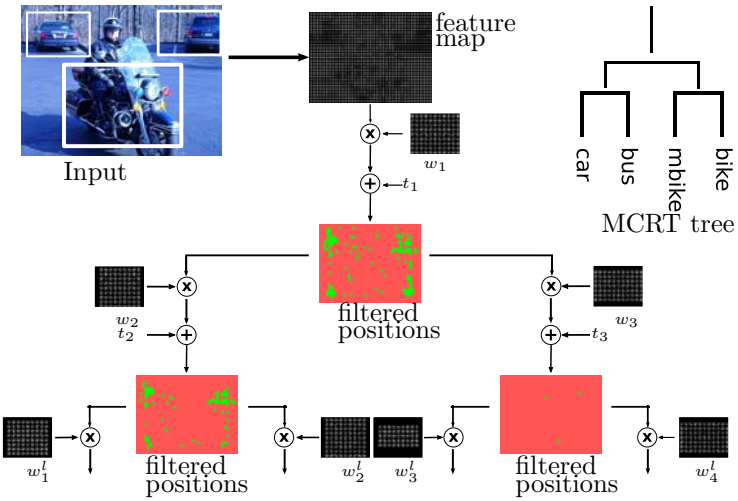


FIGURE A.2 – Illustration des positions où les filtres de T sont appliqués (le rouge indique les positions sans évaluations).

x est calculé par le meilleur score de chaque classe

$$\text{score}(x) = \max_{y \in \{1,\dots,k\}} \text{score}_y(x). \tag{A.1}$$

La classe prédite $\hat{y}$ est donnée par la fonction de décision finale $h : x \to y$ calculant une étiquette $\hat{y}$ pour chaque x :

$$\hat{y} = h(x) = \begin{cases} -1 & \text{, si score}(x) \leq 0 \\ \arg \text{score}(x) & \text{, sinon.} \end{cases} \tag{A.2}$$

Notre modèle de détection multi-classe est défini par un arbre *coarse-to-fine* où les classes sont les feuilles et les nœuds intermédiaires sont déterminés en regroupant les classes similaires de l'étage au-dessous. Chaque nœud intermédiaire est associé aux classes correspondant aux feuilles de l'arbre. Ceci permet le partage de caractéristiques entre les catégories. De plus, la combinaison des classifieurs de chaque noeud donne une ligne de séparation non-linéaire dans l'espace de caractéristiques facilitant la distinction des classes. Nous proposons d'accélérer l'inférence de l'arbre avec un algorithme de recherche pour trouver le chemin avec le plus grand score (*cf.* paragraphe A.4.2).

A.4.1 Notre Modèle de Détection

Dans ce paragraphe, nous donnons des détails sur notre système de détection et introduisons les notations. Un arbre T représentant k classes est formellement défini par $|\text{T}|$ nœuds où n_r est la racine, n_y^l est la feuille associée à la classe y. De plus, n_i $i \in \{1, \dots, |\text{T}|\}$ désigne n'importe quel nœud dans l'arbre. La Fig. A.1 illustre cette notation. De plus, nous notons $\text{anc}(n_i)$ l'ensemble des ancêtres du nœud n_i incluant lui-même et $\text{desc}(n_i)$ est l'ensemble des descendants du noeud n_i (n_i est exclu).

Un modèle de détection pour un arbre T est défini par $|\text{T}|$ filtres $\{w_r, w_2, \dots, w_{|\text{T}|}\}$, w_i étant un filtre pour le nœud i. Le vecteur de décision global w est défini par l'ensemble des w_i. Soit maintenant $\phi_i(x)$ le vecteur de caractéristiques du noeud n_i dans

l'arbre et $\Phi_j(x)$ la concaténation de tous les vecteurs $\phi_i(x)$ des noeuds $n_i \in \text{anc}(n_j)$. Le score pour la classe y à la position x est la somme des scores de tous les filtres individuels sur son chemin :

$$\text{score}_y(x) = w \cdot \Phi_y^l(x). \tag{A.3}$$

La réponse finale définie par l'Eq. (A.2) est calculée par les scores individuels des classes données par l'Eq. (A.3). La combinaison de plusieurs filtres linéaires permet d'avoir une ligne de décision finale non-linéaire. On peut noter que la complexité de ce modèle est $\mathcal{O}(|\text{T}|)$ tandis que le nombre de filtres $|\text{T}|$ augmente de manière significative avec k.

A.4.2 Inférence Rapide

Pendant l'évaluation de l'arbre, notre objectif est d'évaluer uniquement les nœuds se retrouvant sur le chemin vers la feuille de la bonne classe. Par exemple, seuls les filtres $\{w_r, w_2, w_3, w_5, w_6^l, w_4^l, w_5^l\}$ sont appliqués lorsqu'on évalue la région de la voiture dans la Fig. A.1. Aussi, notre algorithme de recherche rejette le *fond* dans les premiers étages. La figure A.2 visualise sur un exemple, les régions (en verts) où les filtres sont utilisés dans T.

Nous procédons de la manière suivante : Chaque nœud évalue ses enfants et additionne le score de leur chemin au score actuel. De plus, notre approche ajoute une estimation de l'importance de tous les poids des chemins restants. Ceci permet d'estimer le meilleur chemin de manière itérative. Nous gardons en mémoire ces estimations et choisissons progressivement le chemin ayant le plus grand score. Même si un chemin n'était pas emprunté dés le début, il se peut qu'il soit traversé plus tard.

Nous introduisons une fonction de classification $g(x, n_i)$:

$$g(x, n_i) = w \cdot \Phi_i(x) \tag{A.4}$$

et une heuristique t_i qui a une valeur constante définie pendant l'apprentissage (cf. paragraphe A.5.4). L'estimation finale à chaque nœud est donnée par la fonction de gain

$$f(x, n_i) = g(x, n_i) + t_i. \tag{A.5}$$

La formulation (A.2) permet implicitement l'utilisation de cet algorithme de recherche pour accélérer l'évaluation car la classe détectée est donnée par le chemin le plus probable. Cette approche est inspirée par la méthode de recherche de plus court chemin A*. L'estimation est choisie pour être toujours plus grande ou égale au vrai score final. Il s'agit d'une estimation optimiste pour atteindre la bonne classe. Ainsi, t_i est admissible et notre algorithme garantit de trouver le chemin optimal.

A.5 Apprentissage Hiérarchique

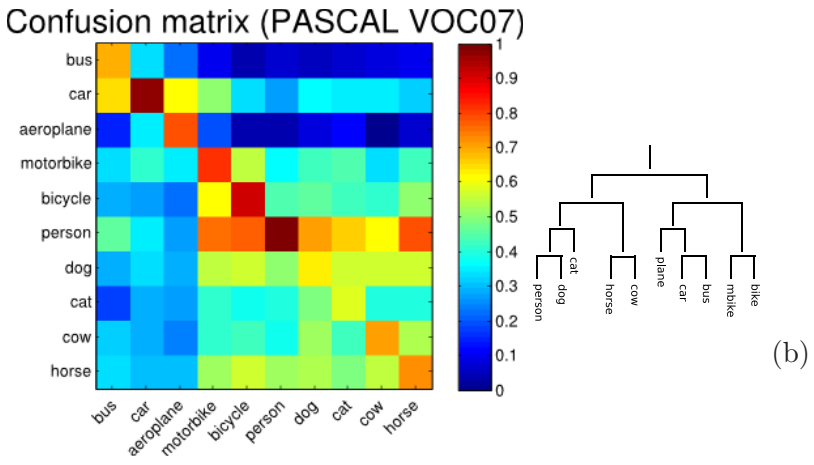
Dans ce paragraphe, nous présentons notre algorithme d'apprentissage hybride. Il apprend de manière automatique la structure de l'arbre, les dimensions des nœuds, leurs vecteurs de poids et les heuristiques.

A.5.1 Construction de la Taxonomie

L'erreur de la classification dépend du pouvoir discriminant des filtres individuels. Les classes qui peuvent être confondues sont regroupées ensemble pour améliorer l'apprentissage de ces filtres. Nous construisons d'abord une matrice de similarité $\mathcal{S} : k \times k$ qui mesure l'affinité s_{ij} entre les paires de classe $(y_i, y_j) \in \mathcal{Y}^+ \times \mathcal{Y}^+$ sur la base de validation. L'affinité s_{ij} est la médiane des valeurs obtenues en classifiant les exemples de la classe y_i avec un détecteur de la classe y_j. Ce détecteur est un simple détecteur HOG, mais nous pourrions employer n'importe quel autre type de filtre dans notre algorithme. De manière plus générale pour un nœud n_i, ses deux enfants (n_{c_1}, n_{c_2}) sont choisis pour minimiser la similarité inter-classe :

$$(n_{c_1}, n_{c_2}) = \arg\min_{\tilde{n}_{c_1}, \tilde{n}_{c_2}} \{\texttt{sim}(\tilde{n}_{c_1}, \tilde{n}_{c_2}) \mid \text{toutes comb. } (\tilde{n}_{c_1}, \tilde{n}_{c_2})\} . \tag{A.6}$$

$\texttt{sim}(n_{c_1}, n_{c_2}) = \sum_{y_i \in n_{c_1}} \sum_{y_j \in n_{c_2}} s_{ij}$ est la similarité entre des super-classes. Ce problème est résolu de manière hiérarchique avec une classification spectrale (spectral clustering) [Shi and Malik, 2000] sur la matrice de similarité. Cette approche a l'avantage de comparer les caractéristiques sur plusieurs échelles et d'être indépendante du domaine d'application. La Fig. A.3 montre un exemple de matrice de similarité et de sa hiérarchie.



(a) (b)

FIGURE A.3 – (a) La matrice de similarité $\mathcal{S}$ pour $k = 10$. (b) L'arbre résultant de la matrice de similarité $\mathcal{S}$.

A.5.2 Formulation du Problème

Nous proposons une approche combinant le tri et la classification regroupés dans un seul problème d'optimisation. Étant données n^+ exemples positifs et n^- exemples négatifs, notre but est de minimiser le risque empirique sur la base d'apprentissage. Ceci conduit

au programme quadratique avec les contraintes linéaires suivantes :

$$\min_{w,\xi_i(j)\geq 0} \frac{1}{2}\|w\|^2 + C\left(\sum_{i=1}^{n^+}(\xi_i + \sum_{j=1}^{n^+}\xi_{ij}) + \sum_{i,j}^{n^-\times n^+}\xi_{ij}\right) \tag{A.7a}$$

$$avec\ \ \forall y_i \in \mathcal{Y}^+, \forall y_j \in \mathcal{Y}^+ : w \cdot \delta\Phi_i(y_j) \geq 1 - \xi_{ij} \tag{A.7b}$$

$$: w \cdot \Phi^l_{y_i}(x_i) \geq 1 - \xi_i \tag{A.7c}$$

$$\forall y_i \in \{y_{bg}\}, \forall y_j \in \mathcal{Y}^+ : -w \cdot \Phi^l_{y_j}(x_i) \geq 1 - \xi_{ij}, \tag{A.7d}$$

où $\delta\Phi_i(y) = \Phi^l_{y_i}(x_i) - \Phi^l_y(x_i)$. $w = (w_r, \cdots, w_{|\mathrm{T}|})$ est la concaténation des vecteurs de poids de chaque nœud dans l'arbre T. La fonction objectif (A.7a) souvent appliquée avec des SVMs est soumise à deux types de contraintes : (i) contraintes de classification (A.7c,A.7d) forçant les poids du filtre global à rejeter les négatifs ; (ii) contraintes de tri (A.7b) entre les exemples de classes positives assurant que le plus grand score soit calculé pour le bon chemin dans l'arbre T.

A.5.3 Optimisation Avec des Plans Sécants

Nous utilisons la méthode des plans sécants [Joachims et al., 2009] pour réduire le temps d'apprentissage. On reformule (A.7) en utilisant une seule variable ξ pour toutes les contraintes donnant le problème d'optimisation suivant :

$$\min_{w,\xi\geq 0} \frac{1}{2}\|w\|^2 + C\xi \tag{A.8a}$$

$$\begin{aligned} &avec\ \ \forall(\bar{y}_1, \ldots, \bar{y}_n) \in \mathcal{Y}^{+n} : \\ &\frac{1}{n}\{\sum_{i=1}^{n^+} w \cdot (\delta\Phi_i(\bar{y}) + \Phi^l_{y_i}(x_i)) \\ &\quad - \sum_{i=1}^{n^-} w \cdot \Phi^l_{\bar{y}}(x_i)\} \geq 1 - \xi. \end{aligned} \tag{A.8b}$$

Le nombre de contraintes augmente de manière exponentielle avec n. Mais l'Eq. (A.8) peut être résolue efficacement en construisant de manière itérative un sous-ensemble de contraintes. Contrairement à la formulation (A.7), seulement une contrainte est ajoutée de manière incrémentale au sous-ensemble. L'algorithme procède de la manière suivante :

1. initialisation du sous-ensemble de contraintes $\mathcal{W} \leftarrow \emptyset$: Au lieu d'ajouter toutes les contraintes, on ajoute une contrainte après l'autre.
2. trouver la contrainte la plus forte : Pour chaque exemple x_i de la classe y_i on cherche une autre classe $\bar{y}_i \in \mathcal{Y}^+ \setminus y_i$ qui lui est le plus proche : $\bar{y}_i = \arg\max_{y\in\mathcal{Y}^+} 1 + w \cdot \Phi^l_y(x_i)$.
3. Ajout de la nouvelle contrainte au sous-ensemble : $\frac{1}{n}\{\sum_{i=1}^{n^+}(w \cdot \Phi_{y_i}(x_i) - \max(w \cdot \Phi_{\bar{y}_i}(x_i), 0) - \sum_{i=1}^{n^-} w \cdot \Phi^l_{\bar{y}_i}(x_i)\} \geq 1 - \xi \rightarrow \mathcal{W}$.
4. Optimisation du problème (A.8) en utilisant un *solver* quadratique sur $\mathcal{W}$.
5. Répétition des étapes (1)-(4) jusqu'à ce que la solution $(w, \xi + \epsilon)$ satisfasse la contrainte A.8b.

L'étape 2 cherche la classe $\bar{y}_i$ qui est la plus proche de l'exemple (x_i, y_i). L'étape 3 ajoute la somme de la contrainte la plus forte au sous-ensemble $\mathcal{W}$ considérant les contraintes de classification et de tri.

A.5.4 Détermination des Heuristiques

Nous cherchons à déterminer les heuristiques les plus strictes tout en évitant les erreurs sur la base d'apprentissage. Chaque heuristique t_i du nœud n_i est déterminée de haut vers le bas et de gauche à droite. Ces valeurs sont des estimations optimistes du vrai gain du score de la classe correcte.

Chaque annotation i des k classes est associée à une image I_i et une position. Soit D_i un ensemble de détections dans I_i où chaque élément x correspond à une instance valide. Une instance valide est une région correctement détectée $y_i = \hat{y}_i = arg\max g(x, n_y^l)$ et ayant un recouvrement δ_o suffisant avec l'annotation de I_i.

$$D_i = \{\forall x \in I_i \mid \hat{y} = y_i \wedge \texttt{intersection}(x, I_i) \geq \delta_o\}\,. \tag{A.9}$$

De plus, soit D l'ensemble de toutes les détections valides $D = \{D_o, \ldots, D_{n^+}\}$. Alors, t_i est donné par :

$$\forall (x, y) \in D : t_j \geq \min \sum_{n_p \in \text{chemin}(n_j, n_y)} \underbrace{w(n_j) \cdot \phi_p(x)}_{\text{score de filtres individuels}}\,, \tag{A.10}$$

où chemin(n_j, n_y) est l'ensemble de nœuds sur le chemin de n_i à n_j. A chaque nœud, nous supprimons les instances multiples ayant un score strictement plus grand que t_i. Toutefois, il reste toujours au moins une détection x pour chaque annotation I_i :

$$\begin{aligned}&\forall i, \exists (x, y) \in D_i \text{ et } n_c \in \text{chemin}(n_r, n_y^l), n_{\bar{c}} \in \mathrm{T} : \\ &f(x, n_c) \geq f(x, n_{\bar{c}}),\end{aligned}$$

avec $f(x, n_i)$ la fonction de gain (*cf.* paragraphe A.4.2).

Dans notre étude, nous avons considéré des heuristiques admissibles par rapport aux données d'apprentissage. Nous pouvons également choisir des heuristiques non admissibles en retirant un certain pourcentage des meilleures détections. Cela produit des heuristiques encore plus strictes accélérant le processus de détection au coût de la performance de la détection.

A.6 Résultats

Dans cette section, nous évaluons notre approche sur les *benchmarks* PASCAL VOC'07 et VOC'10 [Everingham et al., 2010] en suivant leurs protocoles. Chaque base contient 20 classes avec plus de 12000 objets annotés. Elle est partagée de manière égale en base d'apprentissage et de test. Nous avons choisi 6 configurations pour différentes valeurs de $k = \{2, 4, 6, 8, 10, 20\}$ pour nos expériences. Nous avons sélectionné les classes suivantes : {'bus', 'bicycle', 'motorbike', 'car', 'aeroplane', 'person', 'cow', 'horse', 'dog', 'cat'} où pour $k = 8$ nous prenons les 8 premières entrées. Parmi ces classes se

trouvent des classes avec beaucoup de caractéristiques communes (*e.g.*'car' et 'bus') et d'autres moins (*e.g.*'person' et 'aeroplane'). Nous étudions l'influence (1) de mélanger les contraintes de classification et de tri, (2) de faire la détection avec une hiérarchie et (3) d'accélérer la détection avec l'algorithme de recherche dans l'arbre.

Modèles de détection Pour valider les différents aspects de notre méthode, nous comparons la performance de 5 algorithmes de détection (cf. Tab. A.1) : (1) Un-contre-tous (OvA) optimise chaque détecteur indépendemment. Les fonctions de décisions finales sont calibrées [Platt et al., 2000]. Cette stratégie de détection applique k *classifieurs* binaires et conserve le score le plus élevé. (2) Le modèle MCR (multi-class ranking) apprend simultanément tous les classifieurs sans hiérarchie en mixant les contraintes de classification et de tri. (3) Le modèle MCRT apprend une hiérarchie et utilise des heuristiques admissibles pour traverser l'arbre. (4) fMCRT est la version rapide. Contrairement à MCRT, cette approche utilise des heuristiques "moins admissibles" obtenant la même performance de détection que OvA tout en étant considérablement plus rapide. (5) Le modèle eMCRT applique de manière exhaustive tous les filtres de l'arbre. Le choix des caractéristiques ne dépend pas de notre formulation. Nous avons adapté l'outil SVMStruct [Joachims et al., 2009] pour inclure la définition des contraintes et pour apprendre tous les modèles. Nous avons choisi l'histogramme des gradients orientés (HOG [Dalal and Triggs, 2005]) fournis par [Felzenszwalb et al., 2010a] comme *caractéristiques* très reconnus pour leur efficacité. Ils sont extraits autour du centre d'une fenêtre glissante dont les dimensions sont définies par chaque nœud.

Performance de détection Le Tab. A.2a résume les performances de détection en mAP (*mean average precision*) des méthodes pour différents nombres de classe k. MCR atteint des performances similaires à OvA. L'apprentissage simultané des contraintes de *tri* et de classification entre les classes d'objet et le *fond* permet de trouver une ligne de décision stable. Les meilleurs résultats sont obtenus avec eMCRT, une extension de MCR incluant la hiérarchie. L'augmentation du nombre de filtres linéaires et le partage des caractéristiques améliore de manière significative les performances globales de détection.

Complexité de détection Pour VOC'07 les bases d'apprentissage et de validation sont utilisées pour l'apprentissage et la base test pour l'évaluation. Pour VOC'10 la partie apprentissage est utilisée pour apprendre et la partie validation pour l'évaluation [1]. Les Fig. A.4a et A.4b montrent le compromis entre mAP et le gain en vitesse par

[1] La politique de VOC'10 (transférer ses résultats par le site Web du *benchmark*) conduit à des temps d'évaluation trop longs pour cette partie.

Modèle	Classif.	Tri	Hiérarchie	Inférence
OvA	✓			$\|T\|$
MCR	✓	✓		$\|T\|$
MCRT	✓	✓	✓	$< \|T\|$
fMCRT	✓	✓	✓	$\ll \|T\|$
eMCRT	✓	✓	✓	$\|T\|$

Table A.1 – Propriétés des modèles de détection.

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	21.4	1x	22.3	1x	17.8	1x	16.1	1x	12.9	1x	8.8	1x
MCR	23.7	1x	23.0	1x	20.4	1x	17.4	1x	14.7	1x	12.2	1x
*e*MCRT	**25.7**	0.66x	**25.4**	0.55x	**21.8**	0.51x	**20.3**	0.51x	**17.2**	0.51x	**13.1**	0.4x
*f*MCRT	21.4	**1.5x**	22.3	**2.6x**	17.8	**2.9x**	16.1	**4.6x**	12.9	**6.4x**	8.8	**9.8x**

(a) VOC 2007

k	2		4		6		8		10		20	
Evaluation	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed	mAP	Speed
OvA	20.0	1x	15.2	1x	15.9	1x	12.6	1x	11.9	1x	7.7	1x
MCR	18.1	1x	11.4	1x	14.7	1x	11.8	1x	9.42	1x	7.5	1x
*e*MCRT	**20.6**	0.67x	**15.9**	0.55x	**16.9**	0.59x	**13.6**	0.55x	**14.0**	0.56x	**9.0**	0.48x
*f*MCRT	30.5	**1.7x**	22.7	**2.7x**	21.2	**3x**	17.0	**4.2x**	13.4	**5.7x**	9.8	**10.1x**

(b) VOC 2010

TABLE A.2 – Résultats des 4 modèles. Par manque d'annotations de tests pour VOC'10, ces tests ne sont pas fait pour *f*MCRT.

rapport au OvA (méthode de référence). On note tout d'abord que l'utilisation de l'algorithme de recherche permet d'ajuster le compromis entre ces deux critères. Dans presque tous les cas, utiliser l'algorithme de recherche dans l'arbre avec des heuristiques admissibles produit des résultats similaires à l'évaluation de tous les nœuds avec des temps d'exécution plus rapides. Comparé à OvA, nous constatons une amélioration du temps de détection pour le même mAP. Pour $k = 20$ nous avons un facteur de gain 10 pour les 2 jeux de données. La technique de recherche dans l'arbre permet à la fois de rejeter les négatifs tôt dans la hiérarchie et de trouver le chemin vers la bonne classe pour les positifs. Les Fig. A.4c et A.4d présentent le gain relatif en vitesse et mAP pour *f*MCRT ou respectivement *e*MCRT comparé à OvA en fonction du nombre de classes k. Tout d'abord, la performance de rapidité augmente avec le nombre de classe pour les deux jeux de données. Pour *e*MCRT on a toujours un gain en performance de détection qui varie selon k et n'est pas pareil pour les jeux de données. La Fig. A.5 montre le nombre de nœuds appliqués pendant l'exécution par MCRT et *f*MCRT. L'utilisation des heuristiques admissibles permet effectivement de réduire ce nombre de nœuds et réduit le temps de calcul par rapport à une approche exhaustive comme par exemple indiqué sur la Fig. A.4a. Utilisant des heuristiques plus strictes, on réduit le zone sont évaluées plus rapidement et nécessite souvent juste l'utilisant de quelques nœuds. Ceci implique un comportement de détection moins performant.

A.7 Conclusion

Nous avons présenté une méthode permettant d'accélérer la détection hiérarchique multi-classe. Cette tâche est formulée comme un problème de *tri* et de *classification*. Notre formulation de l'apprentissage permet naturellement d'utiliser notre algorithme de parcours d'arbre pour réduire le temps d'exécution. Nous pouvons choisir la performance de détection en fonction de contraintes de temps de calcul. D'autre part, les tests ont montrés que MCRT permet d'avoir des meilleurs résultats que OvA en sacri-

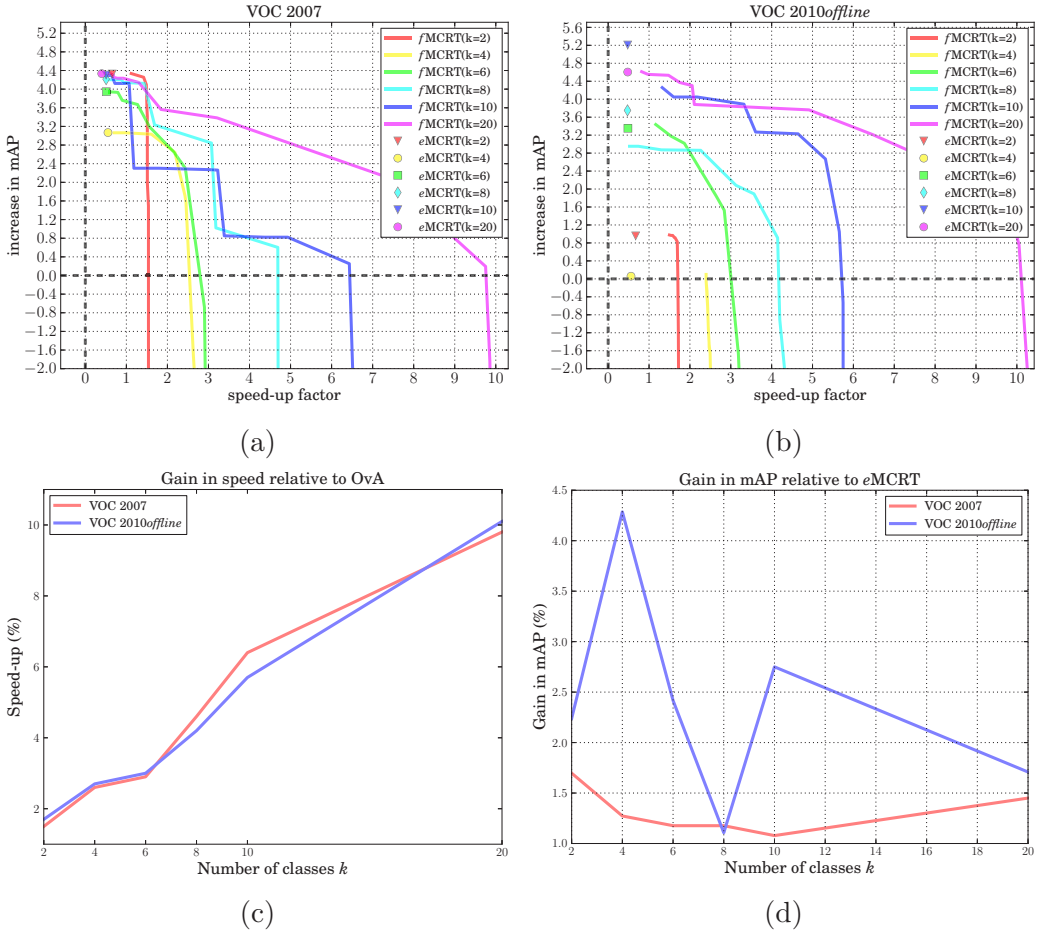


FIGURE A.4 – (a,b) Le compromis entre mAP et gain en vitesse pour les 3 modèles comparé à OvA. (c) Le gain en vitesse de *f*MCRT par rapport à OvA pour différent nombre de classes. Ce gain augmente lorsqu'on ajoute plus de classes. (d) Le gain en mAP par rapport à OvA en utilisant le détecteur *e*MCRT pour diffèrent nombre de classes. On constate que *e*MCRT améliore la performance moyenne indépendant de la valeur de k.

fiant la rapidité. Aussi, MCRT présente un *speed-up* significatif sans compromettre les performances de détection.

Le temps de calcul sur le CPU peut être encore réduit en combinant notre système avec un détecteur d'objet générique [Alexe et al., 2010] pour filtrer des régions dans l'image. Nous sommes convaincus qu'en utilisant des descripteurs plus avancés comme les DPM [Felzenszwalb et al., 2010a], nous pouvons améliorer les résultats de l'état de l'art.

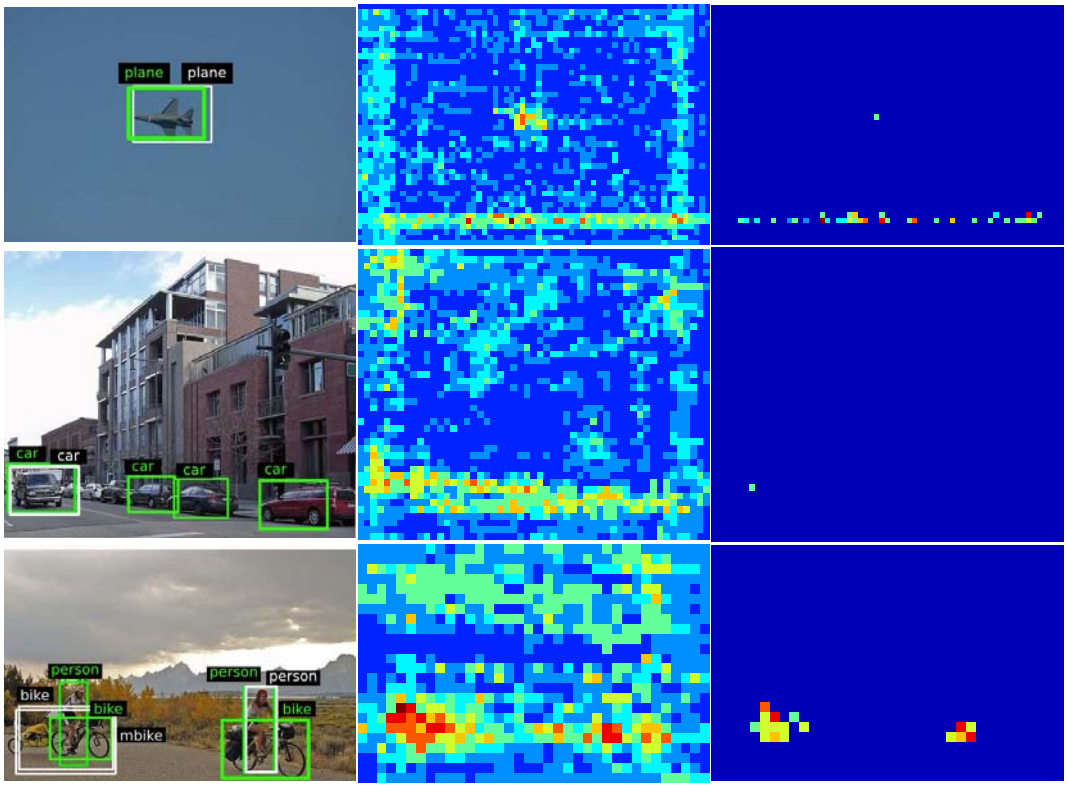


FIGURE A.5 – Résultats obtenus avec MCRT et fMCRT. La colonne de gauche montre les résultats de détection. Les résultats de MCRT sont en vert et pour fMCRT en blanc. Les 2 autres colonnes montrent le nombre relatif de nœuds évalués par MCRT et fMCRT.

Bibliography

[Aghazadeh et al., 2012] Omid Aghazadeh and Hossein Azizpour and Josephine Sullivan and Stefan Carlsson (2012). Mixture component identification and learning for visual recognition. In *European Conference on Computer Vision (ECCV)*.

[Alexe et al., 2010] Bogdan Alexe and Thomas Deselaers and Vittorio Ferrari (2010). What is an object? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Alexe et al., 2012] Bogdan Alexe and Nicolas Heess and Yee Whye Teh and Vittorio Ferrari (2012). Searching for objects driven by context. In *Conference on Neural Information Processing Systems (NIPS)*.

[Amit et al., 2004] Yali Amit and Donald Geman and Xiaodong Fan (2004). A coarse-to-fine strategy for multiclass shape detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Aytar and Zisserman, 2011] Aytar, Y. and Zisserman, A. (2011). Tabula rasa: Model transfer for object category detection. In *International Conference on Computer Vision (ICCV)*.

[Bachmann and Balthasar, 2008] Bachmann, A. and Balthasar, M. (2008). Context-aware object priors. In *Planning, Perception and Navigation for Intelligent Vehicles - Workshop held at IROS*.

[Bar-Hillel et al., 2010] Aharon Bar-Hillel and Dan Levi and Eyal Krupka and Chen Goldberg (2010). Part-based feature synthesis for human detection. In *European Conference on Computer Vision (ECCV)*.

[Bart et al., 2008] Evgeniy Bart and Ian Porteous and Pietro Perona and Max Welling (2008). Unsupervised learning of visual taxonomies. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Bart and Ullman, 2005] Evgeniy Bart and Shimon Ullman (2005). Cross-generalization: learning novel classes from a single example by feature replacement. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Benenson et al., 2012] R. Benenson and M. Mathias and R. Timofte and L. Van Gool (2012). Pedestrian detection at 100 frames per second. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Benenson et al., 2011] Rodrigo Benenson and Radu Timofte and Luc J. Van Gool (2011). Stixels estimation without depth map computation. In *International Conference on Computer Vision (ICCV)*.

[Bengio et al., 2010] Samy Bengio and Jason Weston and David Grangier (2010). Label embedding trees for large multi-class tasks. In *Conference on Neural Information Processing Systems (NIPS)*.

[Bennett and Blue, 1997] K. P. Bennett and J.A. Blue (1997). A support vector machine approach to decision trees. In *Department of Mathematical Sciences Math Report No. 97-100, Rensselaer Polytechnic Institute*.

[Bergboer et al., 2006] Niek Bergboer and Eric O. Postma and H. Jaap van den Herik (2006). Context-based object detection in still images. *Journal of Image and Vision Computing*.

[Biederman, 1987] I Biederman (1987). Recognition-by-components: A theory of human image understanding. *Psychological Review*.

[Binder et al., 2012] Alexander Binder and Klaus-Robert Müller and Motoaki Kawanabe (2012). On taxonomies for multi-class image categorization. *International Journal of Computer Vision*.

[Bishop, 2006] Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc.

[Blaschko and Gretton, 2009] Blaschko, MB. and Gretton, A. (2009). Learning taxonomies by dependence maximization. In *Advances in neural information processing systems 21*, Red Hook, NY, USA. Max-Planck-Gesellschaft, Curran.

[Blaschko et al., 2010] M. B. Blaschko and A. Vedaldi and A. Zisserman (2010). Simultaneous object detection and ranking with weak supervision. In *Conference on Neural Information Processing Systems (NIPS)*.

[Blei et al., 2010] David M. Blei and Thomas L. Griffiths and Michael I. Jordan (2010). The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*.

[Blei et al., 2003] David M. Blei and Thomas L. Griffiths and Michael I. Jordan and Joshua B. Tenenbaum (2003). Hierarchical topic models and the nested chinese restaurant process. In *Conference on Neural Information Processing Systems (NIPS)*.

[Bottou, 2010] Bottou, Léon (2010). Large-scale machine learning with stochastic gradient descent. In *International Conference on Computational Statistics (COMPSTAT)*.

[Bottou, 2012] Léon Bottou (2012). Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade - Second Edition*. Springer.

[Bottou and Bousquet, 2007] Léon Bottou and Olivier Bousquet (2007). The tradeoffs of large scale learning. In *Conference on Neural Information Processing Systems (NIPS)*.

[Boyd and Vandenberghe, 2004] Boyd, Stephen and Vandenberghe, Lieven (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

[Brock, 2007] Brock, D. C., editor (2007). *Understanding Moore's Law: Four Decades of Innovation*, volume 98. The University of Chicago Press.

[Cai and Hofmann, 2004] Cai, Lijuan and Hofmann, Thomas (2004). Hierarchical document categorization with support vector machines. In *ACM International Conference on Information and Knowledge Management (CIKM)*.

[Canini and Griffiths, 2010a] Kevin R. Canini and Thomas L. Griffiths (2010a). Modeling transfer learning and taxonomy induction with the hierarchical Dirichlet process. In *Transfer Learning via Rich Generative Models - Workshop held at NIPS*.

[Canini and Griffiths, 2010b] Kevin R. Canini and Thomas L. Griffiths (2010b). Modeling transfer learning and taxonomy induction with the hierarchical Dirichlet process. In *Transfer Learning via Rich Generative Models - Workshop held at NIPS*.

[Cevikalp, 2010] Hakan Cevikalp (2010). New clustering algorithms for the support vector machine based hierarchical classification. *Pattern Recognition Letters*.

[Chang and Lin, 2011] Chang, Chih-Chung and Lin, Chih-Jen (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*.

[Chang et al., 2013] K.-W. Chang and V. Srikumar and D. Roth (2013). Multicore structural svm training. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*.

[Chapelle, 2007] Olivier Chapelle (2007). Training a support vector machine in the primal. *Neural Computation*.

[Cho et al., 2012] Hyunggi Cho and Paul Rybski and Aharon Bar-Hillel and Wende Zhang (2012). Real-time pedestrian detection with deformable part models. In *IEEE Intelligent Vehicles Symposium (IV)*.

[Choi et al., 2010] Myung Jin Choi and Joseph J. Lim and Antonio Torralba and Alan S. Willsky (2010). Exploiting hierarchical context on a large database of object categories. In *IEEE Conference on Computer VIsion and Pattern Recognition (CVPR)*.

[Choi et al., 2012] Myung Jin Choi and Antonio Torralba and Alan S. Willsky (2012). A tree-based context model for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Ciresan et al., 2011] D. C. Ciresan and U. Meier and J. Masci and J. Schmidhuber (2011). A committee of neural networks for traffic sign classification. In *International Joint Conference on Neural Networks (IJCNN)*.

[Crammer et al., 2001] Koby Crammer and Yoram Singer and Nello Cristianini and John Shawe-taylor and Bob Williamson (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research (JMLR)*.

[Dalal and Triggs, 2005] Navneet Dalal and Bill Triggs (2005). Histograms of oriented gradients for human detection. In *International Conference on Computer Vision & Pattern Recognition (CVPR)*.

[Dean et al., 2013] Dean, Thomas and Ruzon, Mark A. and Segal, Mark and Shlens, Jonathon and Vijayanarasimhan, Sudheendra and Yagnik, Jay (2013). Fast accurate detection of 100000 object classes on a single machine. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Dekel et al., 2004] Ofer Dekel and Joseph Keshet and Yoram Singer (2004). Large margin hierarchical classification. In *International Conference on Machine Learning (ICML)*.

[Deng et al., 2011] Jia Deng and Sanjeev Satheesh and Alex Berg and Li Fei-Fei (2011). Fast and balanced: Efficient label tree learning for large scale object recognition. In *Conference on Neural Information Processing Systems (NIPS)*.

[Desai et al., 2011] Desai, Chaitanya and Ramanan, Deva and Fowlkes, Charless (2011). Discriminative models for multi-class object layout. *International Journal of Computer Vision (IJCV)*.

[Deselaers et al., 2010] Thomas Deselaers and Bogdan Alexe and Vittorio Ferrari (2010). Localizing objects while learning their appearance. In *European Conference on Computer Vision (ECCV)*.

[Dijkstra, 1959] Edsger. W. Dijkstra (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*.

[Divvala et al., 2012] Santosh Kumar Divvala and Alexei A. Efros and Martial Hebert (2012). How important are deformable parts in the deformable parts model? *ACM Computing Research Repository (CoRR)*.

[Divvala et al., 2009] Santosh Kumar Divvala and Derek Hoiem and James Hays and Alexei A. Efros and Martial Hebert (2009). An empirical study of context in object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Dollár et al., 2014] Piotr Dollár and Ron Appel and Serge Belongie and Pietro Perona (2014). Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Dollar et al., 2010] Dollar, Piotr and Belongie, Serge and Perona, Pietro (2010). The fastest pedestrian detector in the west. In *British Machine Vision Conference (BMVC)*. BMVA Press.

[Dollar et al., 2009] P. Dollar and Z. Tu and P. Perona and S. Belongie (2009). Integral channel features. In *British Machine Vision Conference (BMVC)*.

[Dubout and Fleuret, 2012] Dubout, C. and Fleuret, F. (2012). Exact acceleration of linear object detectors. In *European Conference on Computer Vision (ECCV)*.

[Duda and Hart, 1973] Duda, R. O. and Hart, P. E. (1973). *Pattern Classification and Scene Analysis*. John Willey & Sons, New Yotk.

[Espinace et al., 2013] Espinace, P. and Kollar, T. and Roy, N. and Soto, A. (2013). Indoor scene recognition by a mobile robot through adaptive object detection. *Journal of Robotics and Autonomous Systems*.

[Everingham et al., 2010] Everingham, M. and Van Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision (IJCV)*.

[Everingham et al., 2012] Everingham, M. and Van Gool, L. and Williams, C. K. I. and Winn, J. and Zisserman, A. (2012). The PASCAL Visual Object Classes Challenge. http://pascallin.ecs.soton.ac.uk/challenges/VOC/.

[Fan, 2005] Xiaodong Fan (2005). Efficient multiclass object detection by a hierarchy of classifiers. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Fei and Liu, 2006] B. Fei and Jinbai Liu (2006). Binary tree of SVM: a new fast multiclass training and classification algorithm. *IEEE Transactions on Neural Networks*.

[Fei-Fei et al., 2003] Fei-Fei, L. and Fergus, R. and Perona, P. (2003). A Bayesian approach to unsupervised one-shot learning of object categories. In *International Conference on Computer Vision (ICCV)*.

[Fellbaum and Tengi, 2010] Christiane Fellbaum and Randee Tengi (2010). About wordnet. http://wordnet.princeton.edu.

[Felzenszwalb et al., 2008a] P. Felzenszwalb and R. Girshick and D. McAllester and D. Ramanan (2008a). Link to deformable part model. http://cs.brown.edu/ pff/latent-release3/.

[Felzenszwalb et al., 2008b] Felzenszwalb, Pedro and McAllester, David and Ramanan, Deva (2008b). A discriminatively trained, multiscale, deformable part model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Felzenszwalb et al., 2010a] Felzenszwalb, P. F. and Girshick, R. B. and McAllester, D. and Ramanan, D. (2010a). Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Felzenszwalb et al., 2010b] Pedro F. Felzenszwalb and Ross B. Girshick and David A. McAllester (2010b). Cascade object detection with deformable part models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Felzenszwalb and Huttenlocher, 2012] Pedro F. Felzenszwalb and Daniel P. Huttenlocher (2012). Distance transforms of sampled functions. *Theory of Computing*.

[Fidler et al., 2009] Sanja Fidler and Marko Boben and Ales Leonardis (2009). Evaluating multi-class learning strategies in a generative hierarchical framework for object detection. In *Conference on Neural Information Processing Systems (NIPS)*.

[Fidler et al., 2010] Sanja Fidler and Marko Boben and Ales Leonardis (2010). A coarse-to-fine taxonomy of constellations for fast multi-class object detection. In *European Conference on Computer Vision (ECCV)*.

[Fidler and Leonardis, 2007] Sanja Fidler and Ales Leonardis (2007). Towards scalable representations of object categories: Learning a hierarchy of parts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Freund and Schapire, 1997] Freund, Yoav and Schapire, Robert E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*.

[Friedman et al., 2000] J. Friedman and T. Hastie and R. Tibshirani (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*.

[Gadeski et al., 2014] Etienne Gadeski and Hamidreza Odabai Fard and Hervé Le Borgne (2014). Gpu deformable part model for object recognition. *Journal of Real-Time Image Processing (JRTIP)*.

[Gall and Lempitsky, 2009] Juergen Gall and Victor S. Lempitsky (2009). Class-specific hough forests for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Gangaputra and Geman, 2006] Gangaputra, Sachin and Geman, Donald (2006). A design principle for coarse-to-fine classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society.

[Gao and Koller, 2011] Tianshi Gao and Daphne Koller (2011). Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *International Conference on Computer Vision (ICCV)*.

[Geiger et al., 2012] Andreas Geiger and Philip Lenz and Christoph Stiller and Raquel Urtasun (2012). Link to kitti dataset. http://www.cvlibs.net/datasets/kitti/.

[Geiger et al., 2013] Andreas Geiger and Philip Lenz and Christoph Stiller and Raquel Urtasun (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR).*

[Girshick et al., 2014] Ross Girshick and Jeff Donahue and Trevor Darrell and Jitendra Malik (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Gönen and Alpaydın, 2011] Gönen, Mehmet and Alpaydin, Ethem (2011). Multiple kernel learning algorithms. *Journal of Machine Learning Research (JMLR).*

[Göring et al., 2014] Christoph Göring and Erik Rodner and Alexander Freytag and Joachim Denzler (2014). Nonparametric part transfer for fine-grained recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Gretton et al., 2005] Gretton, Arthur and Bousquet, Olivier and Smola, Alex and Schölkopf, Bernhard (2005). Measuring statistical dependence with hilbert-schmidt norms. In *International Conference on Algorithmic Learning Theory (ALT)*, Berlin, Heidelberg. Springer-Verlag.

[Griffin and Perona, 2008] Gregory Griffin and Pietro Perona (2008). Learning and using taxonomies for fast visual categorization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Guillaumin and Ferrari, 2012] Matthieu Guillaumin and Vittorio Ferrari (2012). Large-scale knowledge transfer for object localization in imagenet. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

[Guzmán-Rivera et al., 2013] Abner Guzmán-Rivera and Pushmeet Kohli and Dhruv Batra (2013). Divmcuts: Faster training of structural svms with diverse m-best cutting-planes. In *International Conference on Artificial Intelligence and Statistics (AISTATS).*

[Hamel, 2009] Lutz Hamel (2009). *Knowledge Discovery with: Support Vector Machines.* Wiley.

[Hanson and Riseman, 1978] Hanson, A. R. and Riseman, E. M. (1978). VISIONS: A computer system for interpreting scenes. In Hanson, A. R. and Riseman, E. M., editors, *Computer Vision Systems.* Academic Press, New York.

[Hao et al., 2007] Hao, Pei-Yi and Chiang, Jung-Hsien and Tu, Yi-Kun (2007). Hierarchically svm classification based on support vector clustering method and its application to document categorization. *Expert Systems with Applications.*

[Hart et al., 1968] P. E. Hart and N. J. Nilsson and and B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*.

[Hoffman et al., 2014] Judy Hoffman and Sergio Guadarrama and Eric Tzeng and Ronghang Hu and Jeff Donahue and Ross Girshick and Trevor Darrell and Kate Saenko (2014). LSDA: Large scale detection through adaptation. In *Conference on Neural Information Processing Systems (NIPS)*.

[Hsu and Lin, 2002] Hsu, Chih-Wei and Lin, Chih-Jen (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*.

[I Biederman, 1982] I Biederman, RJ Mezzanotte, JC Rabinowitz (1982). Scene perception: Detecting and judging objects undergoing relational violations. *Cognitive psychology*.

[Izadinia et al., 2014] Hamid Izadinia and Fereshteh Sadeghi and Ali Farhadi (2014). Incorporating scene context and object layout into appearance modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Jain and Dubes, 1988] Jain, Anil K. and Dubes, Richard C. (1988). *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Joachims, 1999a] T. Joachims (1999a). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA.

[Joachims, 1999b] Thorsten. Joachims (1999b). Svmlight. http://svmlight.joachims.orgh/.

[Joachims, 2008] Thorsten. Joachims (2008). Support vector machine for complex outputs. http://www.cs.cornell.edu/people/tj/svm_light/svm_struct.html.

[Joachims et al., 2009] Thorsten Joachims and Thomas Finley and Chun-Nam Yu (2009). Cutting-plane training of structural svms. *Machine Learning*.

[Krempp et al., 2002] S. Krempp and D. Geman and Y. Amit and T E Xsymbols Are Encouraging (2002). Sequential learning of reusable parts for object detection. Technical report, Idiap Research Institute.

[Kressel, 1999] Kressel, Ulrich H.-G. (1999). Pairwise classification and support vector machines. In Schölkopf, B., Burges, C. J. C., and Smola, A. J., editors, *Advances in Kernel Methods*, pages 255–268. MIT Press.

[Krizhevsky et al., 2012] Alex Krizhevsky and Sutskever, Ilya and Geoffrey E. Hinton (2012). Imagenet classification with deep convolutional neural

networks. In Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Conference on Neural Information Processing Systems (NIPS)*, pages 1097–1105. Curran Associates, Inc.

[Kuzborskij et al., 2013] Kuzborskij, Ilja and Orabona, Francesco and Caputo, Barbara (2013). From n to n 1: Multiclass transfer incremental learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Lampert, 2010] Lampert, CH. (2010). An efficient divide-and-conquer cascade for nonlinear object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA. Max-Planck-Gesellschaft, IEEE.

[Lampert et al., 2009a] Lampert, CH. and Blaschko, MB. and Hofmann, T. (2009a). Efficient subwindow search: A branch and bound framework for object localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Lampert and Blaschko, 2008] Christoph H. Lampert and Matthew B. Blaschko (2008). A multiple kernel learning approach to joint multi-class object detection. In *Deutsche Arbeitsgemeinschaft für Mustererkennung (DAGM e.V.)*.

[Lampert et al., 2009b] Christoph H. Lampert and Hannes Nickisch and Stefan Harmeling (2009b). Learning to detect unseen object classes by between-class attribute transfer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Leibe et al., 2008] B. Leibe and A. Leonardis and B. Schiele (2008). Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision (IJCV)*.

[Leibe and Schiele, 2003] Leibe, B. and Schiele, B. (2003). Interleaved object categorization and segmentation. In *British Machine Vision Conference (BMVC)*.

[Levi et al., 2013] Dan Levi and Shai Silberstein and Aharon Bar-Hillel (2013). Fast multiple-part based object detection using kd-ferns. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Lim et al., 2011a] Joseph J. Lim and Ruslan Salakhutdinov and Antonio Torralba (2011a). Transfer learning by borrowing examples for multiclass object detection. In *Conference on Neural Information Processing Systems (NIPS)*.

[Lim et al., 2011b] Ser-Nam Lim and Gianfranco Doretto and Jens Rittscher (2011b). Multi-class object layout with unsupervised image classification and object localization. In *International Symposium on Visual Computing (ISVC)*.

[Lin et al., 2007] Hsuan-Tien Lin and Chih-Jen Lin and Ruby C. Weng (2007). A note on platt's probabilistic outputs for support vector machines. *Machine Learning*.

[Luo et al., 2011] Jie Luo and Tatiana Tommasi and Barbara Caputo (2011). Multiclass transfer learning from unconstrained priors. In *International Conference on Computer Vision (ICCV)*.

[Luxburg, 2007] Luxburg, Ulrike (2007). A tutorial on spectral clustering. *Statistics and Computing*.

[Madzarov et al., 2008] Gjorgji Madzarov and Dejan Gjorgjevikj and Ivan Chorbev (2008). A multi-class svm classifier utilizing binary decision tree. *Informatica*.

[Malisiewicz et al., 2011] Tomasz Malisiewicz and Abhinav Gupta and Alexei A. Efros (2011). Ensemble of exemplar-svms for object detection and beyond. In *International Conference on Computer Vision (ICCV)*.

[Marszalek and Schmid, 2008] Marcin Marszalek and Cordelia Schmid (2008). Constructing category hierarchies for visual recognition. In *European Conference on Computer Vision (ECCV)*.

[Matan et al., 1992] Matan, O. and Bromley, J. and Burges, C. and Denker, J. and Jackel, L. and LeCun, Y. and Pednault, E and Satterfield, W. and Stenard, C and Thompson, T. (1992). Reading handwritten digits: A zip code recognition system. *IEEE Computer*.

[Mikolajczyk et al., 2006] Krystian Mikolajczyk and Bastian Leibe and Bernt Schiele (2006). Multiple object class detection with a generative model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Mikolajczyk et al., 2003] Mikolajczyk, K. and Zisserman, A. and Schmid, C. (2003). Shape recognition with edge-based features. In *British Machine Vision Conference (BMVC)*, volume 2.

[Miller et al., 2000] Eric Miller and N Matsakis and Paul Viola (2000). Learning from one example through shared densities on transforms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1.

[Mittal et al., 2012] Mittal, A. and Blaschko, M. B. and Zisserman, A. and Torr, P. H. S. (2012). Taxonomic multi-class prediction and person layout using efficient structured ranking. In *European Conference on Computer Vision (ECCV)*.

[Mottaghi et al., 2014] Roozbeh Mottaghi and Xianjie Chen and Xiaobai Liu and Sanja Fidler and Raquel Urtasun and Alan Yuille (2014). The role of context for object detection and semantic segmentation in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Odabai Fard et al., 2013] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2013). Détection hiérarchique multi-classes d'objets dans les images. In *ORASIS*, Cluny, France.

[Odabai Fard et al., 2014a] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2014a). Apprentissage hiérarchique simultané pour la détection efficace d'objets. In *Reconnaissance de Formes et Intelligence Artificielle (RFIA)*, France.

[Odabai Fard et al., 2014b] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2014b). Joint Hierarchical Learning for Efficient Multi-class Object Detection. In *IEEE Winter Conference on applications of computer vision (WACV)*, Steamboat Springs, Co, USA.

[Odabai Fard et al., 2014c] Odabai Fard, Hamidreza and Chaouch, Mohamed and Pham, Quoc-Cuong and Vacavant, Antoine and Chateau, Thierry (2014c). Joint Learning for Multi-class Object Detection. In *International conference on computer vision theory and applications (VISAPP)*, Lisbon, Portugal.

[Oliva and Torralba, 2007] Oliva, Aude and Torralba, Antonio (2007). The role of context in object recognition. *Trends in Cognitive Sciences*.

[Opelt et al., 2008] Opelt, Andreas and Pinz, Axel and Zisserman, Andrew (2008). Learning an alphabet of shape and appearance for multi-class object detection. *International Journal of Computer Vision (IJCV)*.

[Oquab et al., 2014] Oquab, Maxime and Bottou, Léon and Laptev, Ivan and Sivic, Josef (2014). Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Columbus, OH, United States.

[Ott and Everingham, 2011] Patrick Ott and Mark Everingham (2011). Shared parts for deformable part-based models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Ouyang and Wang, 2013] Wanli Ouyang and Xiaogang Wang (2013). Single-pedestrian detection aided by multi-pedestrian detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Paletta and Greindl, 2003] Lucas Paletta and Christian Greindl (2003). Context based object detection from video. In *International Conference on Computer Vision Systems (ICVS)*.

[Pedersoli et al., 2010] Pedersoli, Marco and Gonzàlez, Jordi and Bagdanov, Andrew D. and Villanueva, Juan J. (2010). Recursive coarse-to-fine localization for fast object detection. In *European Conference on Computer Vision (ECCV)*, Berlin, Heidelberg. Springer-Verlag.

[Pedersoli et al., 2013] M. Pedersoli and J. Gonzalez and X.Hu and X. Roca (2013). Towards a real-time pedestrian detection based only on vision. *IEEE Transactions on Intelligent Transportation Systems*.

[Pedersoli et al., 2011] Marco Pedersoli and Andrea Vedaldi and Jordi Gonzàlez (2011). A coarse-to-fine approach for fast deformable object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Pepik et al., 2013] Pepik, Bojan and Michael Stark and Peter Gehler and Schiele, Bernt (2013). Occlusion patterns for object class detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Peralta et al., 2012] Billy Peralta and Pablo Espinace and Alvaro Soto (2012). Adaptive hierarchical contexts for object recognition with conditional mixture of trees. In *British Machine Vision Conference (BMVC)*.

[Perko and Leonardis, 2007] Roland Perko and Ales Leonardis (2007). Context driven focus of attention for object detection. In *Attention in Cognitive Systems. Theories and Systems from an Interdisciplinary Viewpoint, 4th International Workshop on Attention in Cognitive Systems, WAPCV 2007, Hyderabad, India, January 8, 2007, Revised Selected Papers*.

[Platt, 1999] John C. Platt (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*. MIT Press.

[Platt et al., 2000] John C. Platt and Nello Cristianini and John Shawe-taylor (2000). Large margin dags for multiclass classification. In *Conference on Neural Information Processing Systems (NIPS)*.

[Polak and Shashua, 2010] Simon Polak and Amnon Shashua (2010). The semi-explicit shape model for multi-object detection and classification. In *European Conference on Computer Vision (ECCV)*.

[Prisacariu and Reid, 2009] Victor Prisacariu and Ian Reid (2009). fasthog - a real-time gpu implementation of hog. Technical Report 2310/09, Department of Engineering Science, Oxford University.

[Razavi et al., 2011] Nima Razavi and Juergen Gall and Luc J. Van Gool (2011). Scalable multi-class object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Razavi et al., 2012] Nima Razavi and Juergen Gall and Pushmeet Kohli and Luc J. Van Gool (2012). Latent hough transform for object detection. In *European Conference on Computer Vision (ECCV)*.

[Ristin et al., 2013] Ristin, M and Gall, J and van Gool, L (2013). Local context priors for object proposal generation. In *Asian Conference on Computer Vision (ACCV)*, volume 7724. Springer.

[Russakovsky et al., 2015] Olga Russakovsky and Jia Deng and Hao Su and Jonathan Krause and Sanjeev Satheesh and Sean Ma and Zhiheng Huang and Andrej Karpathy and Aditya Khosla and Michael Bernstein and Alexander C. Berg and Li Fei-Fei (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*.

[Russell et al., 2008] Russell, Bryan C. and Torralba, Antonio and Murphy, Kevin P. and Freeman, William T. (2008). Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision (IJCV)*.

[Russell and Norvig, 2003] Russell, Stuart J. and Norvig, Peter (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

[Sadeghi and Forsyth, 2013] Mohammad Amin Sadeghi and David Forsyth (2013). Fast template evaluation with vector quantization. In *Conference on Neural Information Processing Systems (NIPS)*.

[Sadeghi and Forsyth, 2014] Mohammad Amin Sadeghi and David A. Forsyth (2014). 30hz object detection with DPM V5. In *European Conference on Computer Vision (ECCV)*.

[Salakhutdinov et al., 2012] Ruslan Salakhutdinov and Joshua B. Tenenbaum and Antonio Torralba (2012). One-shot learning with a hierarchical nonparametric bayesian model. In *Unsupervised and Transfer Learning - Workshop held at ICML*.

[Salakhutdinov et al., 2011] Ruslan Salakhutdinov and Antonio Torralba and Joshua B. Tenenbaum (2011). Learning to share visual appearance for multiclass object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Schmidt, 2009] Mark Schmidt (2009). A note on structural extensions of svms.

[Shalev-Shwartz et al., 2007] Shalev-Shwartz, Shai and Singer, Yoram and Srebro, Nathan (2007). Pegasos: Primal estimated sub-gradient solver for svm. In *International Conference on Machine Learning (ICML)*.

[Shi and Malik, 2000] Jianbo Shi and Jitendra Malik (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Simoncelli and Freeman, 1995] E P Simoncelli and W T Freeman (1995). The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *International Conference on Image Processing (ICIP)*, volume III, Washington, DC. IEEE Sig Proc Society.

[Sivic et al., 2008] Sivic, J. and Russell, B. C. and Zisserman, A. and Freeman, W. T. and Efros, A. A. (2008). Unsupervised discovery of visual object class hierarchies. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Song et al., 2012] H. O. Song and S. Zickler and T. Althoff and R. Girshick and M. Fritz and C. Geyer and P. Felzenszwalb and T. Darrell (2012). Sparselet models for efficient multiclass object detection. In *European Conference on Computer Vision (ECCV)*.

[Song et al., 2011] Zheng Song and Qiang Chen and ZhongYang Huang and Yang Hua and Shuicheng Yan (2011). Contextualizing object detection and classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Strat and Fischler, 1991] Thomas M. Strat and Martin A. Fischler (1991). Context-based vision: Recognizing objects using information from both 2d and 3d imagery. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Sudowe and Leibe, 2011] P. Sudowe and B. Leibe (2011). Efficient Use of Geometric Constraints for Sliding-Window Object Detection in Video. In *International Conference on Computer Vision Systems (ICVS)*.

[Sun et al., 2013] Min Sun and Wan Huang and Silvio Savarese3 (2013). Find the best path: an efficient and accurate classifier for image hierarchies. In *International Conference on Computer Vision (ICCV)*.

[Takahashi and Abe, 2003] F. Takahashi and S. Abe (2003). Decision-tree-based multiclass support vector machines. In *International Conference on Neural Information Processing (ICONIP)*.

[Tang et al., 2012] Siyu Tang and Mykhaylo Andriluka and Bernt Schiele (2012). Detection and tracking of occluded people. In *British Machine Vision Conference (BMVC)*.

[Teh, 2010] Y. W. Teh (2010). Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer.

[Tenenbaum et al., 2011] Joshua B. Tenenbaum and Charles Kemp and Thomas L. Griffiths and Noah D. Goodman (2011). How to grow a mind: Statistics, structure, and abstraction. *Science*.

[Teo et al., 2007] Teo, Choon Hui and Smola, Alex and Vishwanathan, S. V.N. and Le, Quoc Viet (2007). A scalable modular convex solver for regularized risk minimization. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[Torralba, 2003] Antonio Torralba (2003). Contextual priming for object detection. *International Journal of Computer Vision (IJCV)*.

[Torralba et al., 2004] Torralba, A. and Murphy, K. P. and Freeman, W. T. (2004). Sharing features: efficient boosting procedures for multiclass object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Torralba et al., 2007] Antonio Torralba and Kevin P. Murphy and William T. Freeman (2007). Sharing visual features for multiclass and multiview object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*.

[Torralba and Sinha, 2001] Antonio Torralba and Pawan Sinha (2001). Statistical context priming for object detection. In *International Conference on Computer Vision (ICCV)*.

[Totoni et al., 2013] Totoni, Ehsan and Dikmen, Mert and Garzarán, María Jesús (2013). Easy, fast, and energy-efficient object detection on heterogeneous on-chip architectures. *ACM Transactions on Architecture and Code Optimization (TACO)*.

[Tsochantaridis et al., 2004] I. Tsochantaridis and T. Hofmann and T. Joachims and Y. Altun (2004). Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning (ICML)*.

[Uijlings et al., 2013] J.R.R. Uijlings and K.E.A. van de Sande and T. Gevers and A.W.M. Smeulders (2013). Selective search for object recognition. *International Journal of Computer Vision (IJCV)*.

[Uricar et al., 2013] Michal Uricar and Vojtech Franc and Václav Hlavác (2013). Bundle methods for structured output learning - back to the roots. In *Scandinavian Conference on Image Analysis (SCIA)*.

[Vezhnevets and Ferrari, 2013] Vezhnevets, Alexander and Ferrari, Vittorio (2013). Associative embeddings for large-scale knowledge transfer with self-assessment. Technical report, The University of Edinburgh.

[Viola and Jones, 2001] Paul A. Viola and Michael J. Jones (2001). Rapid object detection using a boosted cascade of simple features. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Wang et al., 2010] Gang Wang and David A. Forsyth and Derek Hoiem (2010). Comparative object similarity for improved recognition with few or no examples. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Wojek et al., 2008] Christian Wojek and Gyuri Dorkó and André Schulz and Schiele, Bernt (2008). Sliding-windows for rapid object class localization: A parallel technique. In *Pattern Recognition*.

[Xiao et al., 2012] J. Xiao and J. Hays and K. Ehinger and A. Oliva and and A. Torralba (2012). Link to scnene understanding dataset. http://groups.csail.mit.edu/vision/SUN/.

[Xiao et al., 2010] Jianxiong Xiao and James Hays and Krista A. Ehinger and Aude Oliva and Antonio Torralba (2010). Sun database: Large-scale scene recognition from abbey to zoo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Xu and Tenenbaum, 2007] Fei Xu and Joshua B. Tenenbaum (2007). Word learning as bayesian inference. In *Psychological Review*.

[Xu and Wunsch, 2005] Xu, Rui and Wunsch,II, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*.

[Yan et al., 2014] Junjie Yan and Zhen Lei and Longyin Wen and Stan Z. Li (2014). The fastest deformable part model for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Yang and Tsang, 2012] Jian-Bo Yang and Ivor W. Tsang (2012). Hierarchical maximum margin learning for multi-class classification. *ACM Computing Research Repository (CoRR)*.

[Yao and Doretto, 2010] Yao, Y. and Doretto, G. (2010). Boosting for transfer learning with multiple sources. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Yu and Joachims, 2008] Chun-Nam Yu and T. Joachims (2008). Training structural svms with kernels using sampled cuts. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[Zehnder et al., 2008] P. Zehnder and E. Koller-Meier and L. Van Gool (2008). An efficient shared multi-class detection cascade. In Everingham, M., Needham, C. J., and Fraile, R., editors, *British Machine Vision Conference (BMVC)*. British Machine Vision Association (BMVA).

[Zhang et al., 2013] Suofei Zhang and Nijun Li and Xu Cheng and Zhenyang Wu (2013). Adaptive object detection by implicit sub-class sharing features. *Signal Processing*.

[Zhu et al., 2010a] Long Zhu and Yuanhao Chen and Antonio Torralba and William T. Freeman and Alan L. Yuille (2010a). Part and appearance sharing: Recursive compositional models for multi-view. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Zhu et al., 2010b] Long Zhu and Yuanhao Chen and Alan L. Yuille and William T. Freeman (2010b). Latent hierarchical structural learning for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Zhu et al., 2015] Yukun Zhu and Raquel Urtasun and Ruslan Salakhutdinov and Sanja Fidler (2015). segdeepm: Exploiting segmentation and context in deep neural networks for object detection. *ACM Computing Research Repository (CoRR)*.