Efficient Object Detection via Structured Learning and Local Classifiers

Ziming Zhang

Thesis submitted in partial fulfillment of the requirements of the award of

Doctor of Philosophy

Oxford Brookes University

2013

THE FOLLOWING FIGURES HAVE BEEN EXCLUDED AT THE REQUEST OF THE UNIVERSITY:

FIGURE 1.1 (PAGE 2)

FIGURE 1.2 (PAGE 3)

FIGURE 1.4 (PAGE 7)

FIGURE 1.5 (PAGE 9)

Abstract

Object detection has made great strides recently. However, it is still facing two big challenges: detection accuracy and computational efficiency. In this thesis, we present an automatic efficient object detection framework to detect object instances in images using bounding boxes, which can be trained and tested easily on current personal computers. Our framework is a sliding-window based approach, and consists of two major components: (1) efficient object proposal generation, predicting possible object bounding boxes, and (2) efficient object proposal verification, classifying each bounding box in a multiclass manner.

For object proposal generation, we formulate this problem as a structured learning problem and investigate structural support vector machines (SSVMs) with our proposed scale/aspect-ratio quantization scheme and ranking constraints. A general ranking-order decomposition algorithm is developed for solving the formulation efficiently, and applied to generate proposals using a two-stage cascade. Using image gradients as features, our object proposal generation method achieves state-of-the-art results in terms of object recall at a low cost in computation.

For object proposal verification, we propose two locally linear and one locally nonlinear classifiers to approximate the nonlinear decision boundaries in the feature space efficiently. Inspired by the kernel trick, these classifiers map the original features into another feature space explicitly where linear classifiers are employed for classification, and thus have linear computational complexity in both training and testing, similar to that of linear classifiers. Therefore, in general, our classifiers can achieve comparable accuracy to kernel based classifiers at the cost of lower computational time.

To demonstrate its efficiency and generality, our framework is applied to four different object detection tasks: VOC detection challenges, traffic sign detection, pedestrian detection, and face detection. In each task, it can perform reasonably well with acceptable detection accuracy and good computational efficiency. For instance, on VOC datasets with 20 object classes, our method achieved about 0.1 mean average precision (AP) within 2 hours of training and 0.05 second of testing a 500×300 pixel image using a mixture of MATLAB and C++ code on a current personal computer.

Acknowledgements

I would like to take this opportunity to thank my supervisor, Prof. Philip H.S. Torr, for guiding me through my PhD studies at Oxford Brookes University and always being there with useful advices and suggestions.

Meanwhile, I would like to thank my colleagues Mr. Paul Sturgess, Mr. Sunando Sengupta, Mr. Morten Lidegaard, Mr. Ondrej Miksik, Dr. Ľubor Ladický, Dr. Amir Saffari, Dr. Mingming Cheng, and many others in the Oxford Brookes Vision Group for their great discussion and care.

Also, I would like to thank Prof. Andrew Zisserman and all the members in his Visual Geometry Group, University of Oxford, from whom I have learned a lot.

It is my honor to have Dr. Mark Bishop (University of London), Dr. Andrea Vedaldi (University of Oxford), and Dr. David Duce (Oxford Brookes University) as my examiners. Thank them for their very useful suggestions, encouraging words, thoughtful criticism, and time on my thesis.

Special thanks to the IST Programme of the European Community, under the PASCAL2 Network of Excellence, for the financial support.

Finally, I would like to thank my family for their understanding and support, especially my wife, Dr. Cong Geng.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

(a) Original image (b) Image with some object detetion results

Figure 1.1: An example of object detection in an image.

Object detection in computer vision aims to provide a general method to localize and recognize object instances of interest within different categories simultaneously in images or videos [32, 38, 99, 104, 115, 129, 145]. For instance, given Fig. 1.1(a), we would like to find the light pole, buses, pedestrians, and even buildings. An object detection method may return some results like Fig. 1.1(b).

Since object detection is a huge research area, in this thesis we would like to restrict our object detection problem to the following task:

We focus on developing a sliding window based efficient object detection framework, which can be applied to different detection tasks with little modification and effort, and run easily on current personal computers, so that it can detect object instances of interest within different categories automatically and simultaneously in 2D images, and output bounding boxes (i.e. rectangles) surrounding each object instance. In our framework, computational efficiency in both training and testing and detection generality are considered as important as detection accuracy.

(a) Automatic navigation (b) Surveillance (c) Robotics (d) Medical applications

Figure 1.2: Illustration of some real-world applications involving object detection.

1.1 Motivation

The ultimate goal of computer vision is to build an automatic system which can duplicate the ability of human vision to fully understand the contents of images, *e.g.* things and stuff [64], and reason about the high-level relations between them, *e.g.* object geometry and activity [65, 79], with the aid of geometry, physics, statistics, and learning theory [55]. Towards this goal, object detection plays a very important role, because it answers two essential questions in computer vision: object localization, answering where the object instances of interest are with respect to the images, and object recognition, answering what categories the object instances of interest belong to.

With the help of localization and recognition, object detection can benefit many other research areas in computer vision, such as image classification [111], image segmentation [139], object tracking [62], etc. For instance, in the recent Visual Object Classes Challenge 2012 (VOC2012) [47], for the image classification competition (*i.e.* "comp1"), the winning method introduced object detection techniques into their recognition framework to improve performance. Not only for research purposes, many real-world applications involve object detection as one of the key techniques as well, such as automatic navigation [41], surveillance [100], robotics [60, 69], and medical applications [152]. Fig. 1.2 illustrates these applications using object detection.

Obviously detection accuracy is very important as every detection method wishes to achieve as high accuracy as possible. In an automatic navigation sys-

tem, for instance, it is desired that every person in front of the car should be detected. What other factors should really matter in real-world applications?

Computational Efficiency. Following the example of automatic navigation, the systems are expected to detect all possible persons as fast as possible by allowing incorrect detections to a certain degree, so that the drivers (or other parts of the systems) have sufficient time to react. Take the surveillance application in Fig. 1.2(b) for another example. As the shop owner, it is definitely expected that any movement of human beings involving violence can be detected, or even predicted, as early as possible, so that he can better protect himself and his property. With the development of IT hardware, mobile computing has become more and more popular. It is reported by Silicon India that the number of active cell phones will reach 7.3 billion by 2014. What a huge potential market for object detection applications! However, whether a detection system can be applied successfully on these mobile phones is highly dependent on its computational efficiency and requirement on the hardware, since mobile phones can only offer very limited computing power due to their usage.

All such applications suggest that for many real-world applications, the computational efficiency of detection is another major concern, which may be as important as detection accuracy.

Detection Generality. Recently the commercial product, Kinect [119,148], from Microsoft has achieved a big success, and Kinect itself can be considered as a milestone in the history of computer vision. Besides the hardware, the algorithm [119] used in Kinect makes it possible to recognize hundreds of different human poses in real-time with high accuracy, which contributes significantly to its success. Besides Kinect, many other applications require to handle object instances of interest within different categories (in Kinect, each human pose is categorized to a class for recognition). As we illustrate in Fig. 1.2(a), an automatic navigation system should detect multiple objects with different categories.

Therefore, a good object detection system should be able to handle multiclass object detection problems inherently at a low cost of computation (otherwise, in

tem, for instance, it is desired that every person in front of the car should be detected. What other factors should really matter in real-world applications?

Computational Efficiency. Following the example of automatic navigation, the systems are expected to detect all possible persons as fast as possible by allowing incorrect detections to a certain degree, so that the drivers (or other parts of the systems) have sufficient time to react. Take the surveillance application in Fig. 1.2(b) for another example. As the shop owner, it is definitely expected that any movement of human beings involving violence can be detected, or even predicted, as early as possible, so that he can better protect himself and his property. With the development of IT hardware, mobile computing has become more and more popular. It is reported by Silicon India that the number of active cell phones will reach 7.3 billion by 2014. What a huge potential market for object detection applications! However, whether a detection system can be applied successfully on these mobile phones is highly dependent on its computational efficiency and requirement on the hardware, since mobile phones can only offer very limited computing power due to their usage.

All such applications suggest that for many real-world applications, the computational efficiency of detection is another major concern, which may be as important as detection accuracy.

Detection Generality. Recently the commercial product, Kinect [119,148], from Microsoft has achieved a big success, and Kinect itself can be considered as a milestone in the history of computer vision. Besides the hardware, the algorithm [119] used in Kinect makes it possible to recognize hundreds of different human poses in real-time with high accuracy, which contributes significantly to its success. Besides Kinect, many other applications require to handle object instances of interest within different categories (in Kinect, each human pose is categorized to a class for recognition). As we illustrate in Fig. 1.2(a), an automatic navigation system should detect multiple objects with different categories.

Therefore, a good object detection system should be able to handle multiclass object detection problems inherently at a low cost of computation (otherwise, in

contradiction to the computational efficiency requirement). Meanwhile, it should be applicable for different detection tasks with little modification and effort.

In this thesis we aim to develop an efficient object detection framework, which performs multiclass object detection simultaneously and efficiently for different detection tasks, and can run easily on current personal computers for real-world applications.

1.2 Challenges

The essence of every object detection method is to localize and recognize object instances of interest. *A sliding window based object detector takes every pixel in images as a potential location for an object, and every patch centered at the pixel as a potential object which needs to be recognized or discarded.* Therefore, we here explain the challenges in object localization and object recognition separately for sliding window based object detection methods in general.

1.2.1 Localization

For the bounding box based detection methods, the goal of object localization is to put a bounding box around each object instance as close to the object's boundary as possible. To better explain the challenges in localization, we first introduce some definitions for measuring the localization quality.

Definition 1.1 (Bounding Box Overlap Score). *The* overlap score *between a bounding box s and a ground-truth bounding box of an object t, $o(s,t)$, is defined as their intersection area, $A_3(s,t)$, divided by their union area, $A_1(s,t)+A_2(s,t)+A_3(s,t)$, as illustrated in Fig. 1.3, and calculated using Eq. 1.2.1 below:*

$$o(s,t) = \frac{s \bigcap t}{s \bigcup t} = \frac{A_3(s,t)}{A_1(s,t) + A_2(s,t) + A_3(s,t)}. \tag{1.2.1}$$

Clearly, $0 \leq o(s,t) \leq 1$, and the higher $o(s,t)$ is, the better the detection with the bounding box s is.

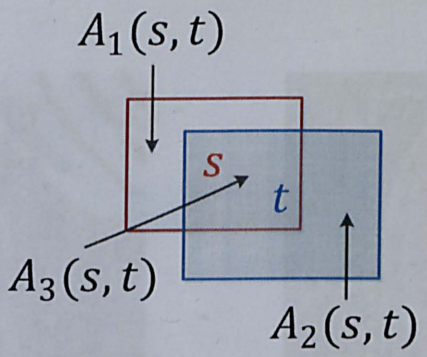


Figure 1.3: Explanation of how to calculate the overlap score between two bounding boxes s and t using Eq. 1.2.1, where A_1, A_2 and A_3 denote three areas, respectively.

Definition 1.2 (η-**Accuracy**). *A window $s \in S$ can be localized by another window $t \in T$ to* η-accuracy *if $o(s,t) \geq \eta$ $(0 \leq \eta \leq 1)$.*

Definition 1.3 (**Correct Detection**). *Given an overlap threshold η, a detection bounding box s is considered as a* correct detection *for detecting object t if and only if s can localize t to η-accuracy.*

Then we list some challenges in localization as follows:

1. **The parameter search space for finding correct bounding boxes is huge.** In an image, each window can be represented by 4 parameters: the coordinate of its top-left corner (x, y) and its width and height (w, h). Suppose there are no spatial and scale/aspect-ratio priors for objects, which means that objects can be uniformly distributed at any position in an image with any reasonable scale/aspect-ratio, then given an image with width and height (W, H), the number of possible bounding boxes is:

$$\sum_{x=1}^{W}\sum_{y=1}^{H}\sum_{w=0}^{W-x}\sum_{h=0}^{H-y} = \frac{1}{4}WH(W-1)(H-1) = O(W^2H^2) \tag{1.2.2}$$

That is, the number of possible bounding boxes is quadratically proportional to the size of the image. Thus, given an image with a common resolution 256×256 pixels, this number will be about $2^{30} \approx 10^9$! Therefore, how to

Figure 1.4: Illustration of some cases that non-maximum suppression (NMS) fails.

search for the correct bounding boxes in this huge space efficiently becomes challenging.

2. **The correct detection for a single object instance of interest may be multiple.** For an arbitrary object instance, any detection satisfying Def. 1.3 is a correct detection. However, since there is only one object, more than one correct detections become redundant, and harm the precision-recall score, which is used to measure the detection methods. A well-known method to handle this problem is non-maximum suppression (NMS) [22]. For instance, in the VOC detection challenges [47], given two correct detection bounding boxes s_1 and s_2 *w.r.t.* an object, with detection confidence scores c_1 and c_2 (indicating the possibility of a bounding box containing an object, and higher scores, higher possibilities), respectively, NMS will remove s_2 if $c_1 > c_2$ and $\tilde{o}(s_1, s_2) = \frac{A_3(s_1,s_2)}{A_2(s_1,s_2)+A_3(s_1,s_2)} \geq 0.5$; otherwise, s_2 will be kept and considered as another detection. However, using this NMS rule there are still many cases that cannot be handled correctly, as illustrated in Fig. 1.4. Therefore, how to remove the redundancy of multiple correct detections to a single object still remains challenging.

1.2.2 Recognition

Generic object recognition (also called object categorization) in images has a long history in computer vision, and it is still unsolved [33]. The goal of object recognition is to classify each object instance using certain features and models. Features are used to represent each object instance based on low level image information like pixels, and they can be hand-crafted such as Scale-Invariant Feature Transform (SIFT) [91] and Histogram of Oriented Gradients (HOG) [32], or learned using, for instance, deep learning [81]. Object models are used to verify the category of each object instance. These models typically represent high level properties of object categories, which are presumed to be shared by all the instances belonging to each category, such as statistical information of visual words (*e.g.* the Bag-of-Words model (BoW) [31] and its derivatives such as the spatial pyramid BoW model [80]), attribute based models [50], and part based models [53]. Usually some machine learning techniques are involved to learn these object models, such as support vector machines (SVMs) [39], multiple kernel learning [136], and graphical models [122].

In object categorization, there are still many challenges such as:

1. **Imaging factors, *e.g.* lighting, occlusion, truncation, clutter, pose.** These factors affect the quality and content complexity of images a lot, and make it very difficult to represent each object instance properly using low level information.

2. **Intra-class variation *v.s.* inter-class variation.** Intra-class variation in object instances makes the learning of object models difficult, because the models need to deal with "all" possible variabilities in objects. And inter-class variation determines the discriminability between different object classes. Therefore, a good object representation should have low intra-class variation and high inter-class variation.

3. **Many object categories.** There are approximately 10^4 — 3×10^4 object categories [11] in the world that humans can recognize. Considering the variation challenge above, in order to make computer vision comparable

Figure 1.5: Some sample images from the categories of sofa (top), bicycle (middle), and motorbike (bottom) in VOC2007 [42] to illustrate the challenges in object recognition.

with the visual capability of human being, we still need to do massive work to help machines recognize them.

Fig. 1.5 gives some examples of these challenges above. For object detection, there exist some other extra challenges in recognition such as:

4. **Very large-scale and extremely imbalanced correct and wrong detection bounding boxes for both learning and testing.** A natural image would usually only contain a few objects of interest. However, as we describe above, the search space for correct bounding boxes in images is huge. This makes the training windows extremely imbalanced, with very

small portion of positives and massive negatives, which is problematic for learning in both computer vision and machine learning.

1.3 Contributions

The major contributions introduced within this thesis for object detection are concerned with the development of efficient methods for:

1. Searching the possibly correct bounding boxes for object instances (*i.e.* object proposals). This problem can be denoted as the object proposal generation problem;

2. Learning multiclass nonlinear classifiers (*i.e.* object models) for recognition, which can handle large-scale imbalanced data. This problem can be denoted as the object proposal verification problem;

and

3. Presenting an efficient object detection framework using the developed methods in object proposal generation and verification, as shown in Fig. 1.6, with good computational efficiency, detection generality, and acceptable detection accuracy.

For object proposal generation, we formulate this problem as a structured learning problem and investigate structural support vector machines (SSVMs) with our proposed scale/aspect-ratio quantization scheme and ranking constraints. A general ranking-order decomposition algorithm is developed for solving the formulation efficiently, and applied to generate proposals using a two-stage cascade. Using image gradients as features, our object proposal generation method achieves state-of-the-art results in terms of object recall at a low cost in computation.

For object proposal verification, we propose two locally linear and one locally nonlinear classifiers to approximate the nonlinear decision boundaries in the feature space efficiently. Inspired by the kernel trick, these classifiers map the original features into another feature space explicitly where linear classifiers are employed for classification, and thus have linear computational complexity in

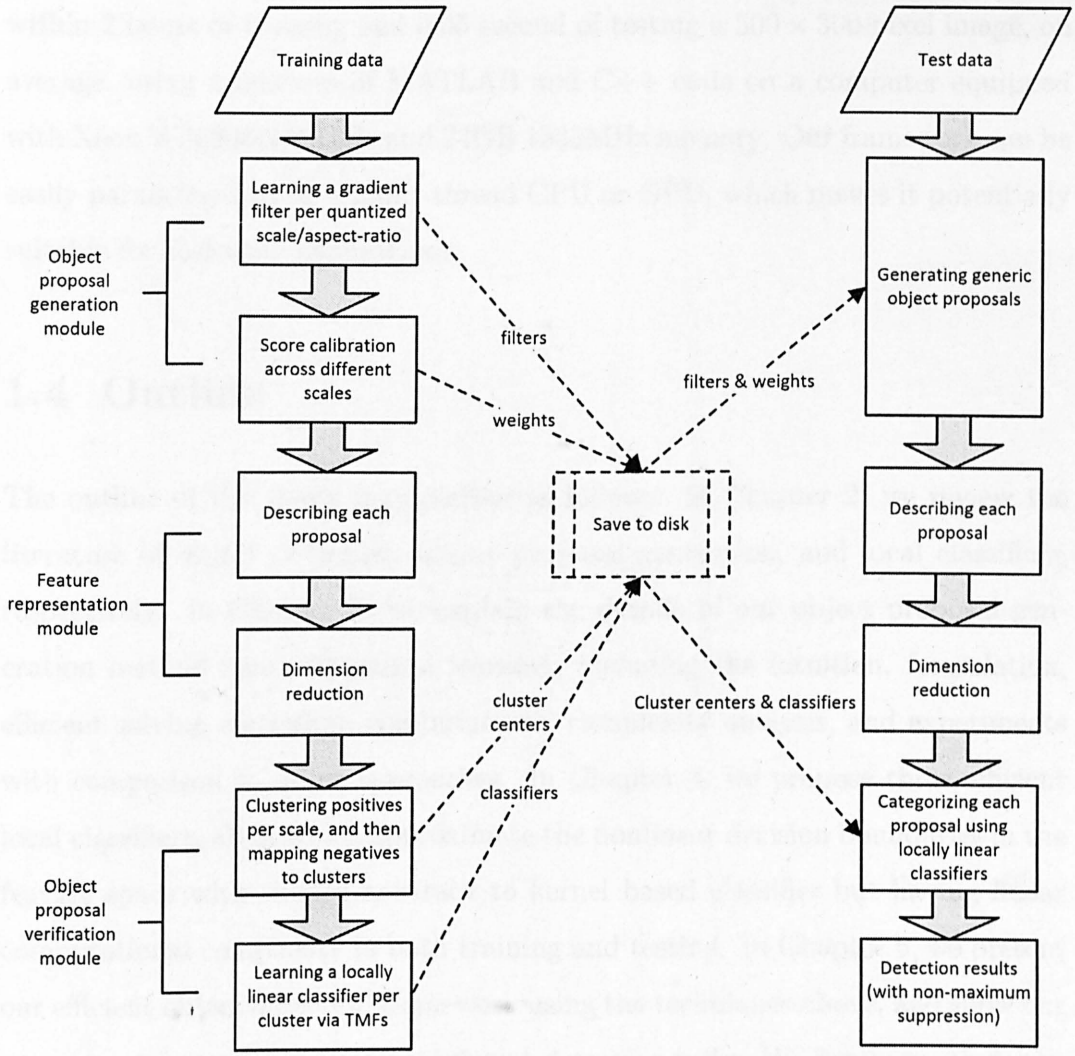


Figure 1.6: Work flow of our efficient object detection framework in both training and testing.

both training and testing, similar to that of linear classifiers. Therefore, in general, our classifiers can achieve comparable accuracy to kernel based classifiers at the cost of lower computational time.

We demonstrated the efficiency and generality of our detection framework by applying it to four different object detection tasks, that is, VOC detection challenges, traffic sign detection, pedestrian detection, and face detection. HOG features are used for representing each object proposal in our framework. In each task, our method can perform reasonably well with acceptable detection accuracy and good computational efficiency. For instance, on VOC datasets with 20 object classes, our method achieved about 0.1 mean average precision (AP)

within 2 hours of training and 0.05 second of testing a 500 × 300-pixel image, on average, using a mixture of MATLAB and C++ code on a computer equipped with Xeon W3680@3.33GHz and 24GB 1333MHz memory. Our framework can be easily parallelized using a multi-thread CPU or GPU, which makes it potentially suitable for real-time applications.

1.4 Outline

The outline of the thesis is organized as follows: In Chapter 2, we review the literature of object detection, object proposal generation, and local classifiers, respectively. In Chapter 3, we explain the details of our object proposal generation method using structural learning, including the intuition, formulation, efficient solving algorithm, computational complexity analysis, and experiments with comparison to other approaches. In Chapter 4, we propose three efficient local classifiers, all of which approximate the nonlinear decision boundaries in the feature space with similar accuracy to kernel based classifier but having linear computational complexity in both training and testing. In Chapter 5, we present our efficient object detection framework using the techniques above, and show our experimental results on the four different detection tasks. We finally conclude the thesis in Chapter 6 and give some perspectives for future work.

1.5 Publications

Part of the work described here has previously appeared as the following publications.

- Chapter 3
  - [151] Ziming Zhang, Jonathan Warrell, and Philip H. S. Torr. Proposal Generation for Object Detection using Cascaded Ranking SVMs. In proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1497-1504, 2011.
- Chapter 4

— [149] Ziming Zhang, Lubor Ladicky, Philip H. S. Torr, and Amir Saffari. Learning Anchor Planes for Classification. In proceedings of Advances in Neural Information Processing Systems (NIPS), pages 1611-1619, 2011.

— [150] Ziming Zhang, Paul Sturgess, Sunando Sengupta, Nigel Crook, and Philip H. S. Torr. Efficient Discriminative Learning of Parametric Nearest Neighbor Classifiers. In proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2232-2239, 2012.

For each of the above papers, I contributed the key ideas and wrote the first drafts.

Chapter 2

Literature Review

Sliding window based object detection has a long history in computer vision [8, 32, 37, 51, 56, 85, 86, 95, 114, 120, 127, 129]. In this chapter, we first review several types of sliding window based object detection methods. In general all of these methods contain two major modules, object proposal generation and verification, respectively, which is also the case with ours. Therefore, we further review some related work on object proposal generation and verification in our detection framework.

2.1 Sliding Window Based Object Detection

In this section, we review several types of sliding window based object detection methods chronologically from shape-based to appearance-based methods. Notice that a method may contain some techniques from different types of approaches, and we simply categorize it to one type without repeating it in the others.

Interest point matching [90,91]. Geometric information of interest points on objects is a good cue to recognize rigid objects because it has a certain tolerance to image translation, rotation, and scaling. In this context, object detection problems can be considered as an alignment problem [98] where two sets of points from the template and target, respectively, are aligned based on certain measures. A famous example of such approaches is matching Scale Invariant Feature Transform (SIFT) [90,91] points for object detection. The computational time of such methods is relatively low, and can be applied for *specific* rigid object detection with good performance.

Shape/Contour matching using distance measures [4,56,89,92,120] or shape descriptors [7,8,9]. The shape or contour of an object is a set of points that enclose the object as tightly as possible. Typically, these shapes are represented using edges, on top of which different distance measures can be used for detection, such as Chamfer distance [89,92], Hausdorff distance [120], or even self-defined distance [4]. Calculating the distance between a template shape and a patch in a target image is very fast, especially using distance transforms [56], and

for some objects with distinct shapes the detection accuracy is good. However, as a global shape, such methods suffer from being sensitive to noisy points, highly dependent on detected edges, and hardly handling occlusion.

To resolve such problems, point distributions along the edges are captured to summarize geometric information of shapes. These points can be quantized like Shape Context [7], or can be soft-assigned like Geometric Blur [8, 9]. Then matching can be done among these shape descriptors. Such methods are less sensitive to noisy points and can handle occlusion partially, but still they are heavily dependent on edge detection algorithms.

Implicit Shape Models (ISMs) [85,86,103,118]. An ISM for a given object category learns a class-specific codebook, where each codeword can be used to predict the existence of an object instance with a spatial probability distribution. Codewords can be local appearance features detected by interest point detectors [85, 86], or contour fragments [103, 118], which are learned discriminatively using some machine learning techniques like boosting. Usually, the spatial probability distributions are assumed to be Gaussian, and each object instance is represented as a star model. That is, the prediction of each codeword is independent with each other, and all the codewords lying on an object instance are connected with the center of the object. Final detections can be made by voting [85, 86] or classification [103, 118]. Compared with those shape matching methods, ISMs allow the shape models deformable to better fit the object instances. However, the computational time of such methods may be much longer, and their performances are highly dependent on the existence of the codewords.

Viola-Jones object detection framework (*i.e.* simple features with fast cascaded classifiers) [37,87,129,147]. The basic idea behind the Viola-Jones framework [129] is to use Haar-like features to capture the mean of the pixel intensities in images at different locations with varying scales/aspect-ratios, and evaluate these distributions with fast cascaded classifiers, which are learned supervisedly using boosting. Because Haar-like features are translation and scale invariant, integral images can be utilized for fast calculation of the responses of

Haar-like features without constructing image pyramids. An *integral image* for a given image is a matrix whose entry at an arbitrary location (x, y) is the sum of the pixel values above and to the left of (x, y), inclusive, in the original image. This framework has been demonstrated to succeed in face detection [129] and some other kinds of rigid object detection such as cars and pedestrians [97].

Inspired by the nice property of the translation and scale invariance of Haar-like features, Dollar *et. al.* [37] proposed approximating the feature responses in an image pyramid using a feature pyramid as long as the features satisfy the translation and scale invariance property. In this way, the computational time is reduced dramatically, and they showed good performance of such a method on pedestrian detection.

Meanwhile, fast cascaded classifiers began to be widely used in object detection to prune the window searching space [147]. Recently Li *et. al.* [87] proposed a Speeded Up Robust Features (SURF) cascaded classifier with much less weak classifiers and stages in the cascade, which leads to faster training and testing speed as well as better performance on face detection. Vedaldi *et. al.* [127] proposed another cascaded classifier for object detection and demonstrated its power on VOC2009 [44]. In this method, the computational complexity of the classifier is still very high, even in a cascaded manner from the bottom stage (linear classifiers) to the top stage (kernel SVMs). Also many different features are employed to build multiple kernels for final decision, which increases the computational time as well.

HOG features with linear classifiers [32, 63, 95, 96]. In [32], Dalal and Triggs proposed Histogram of Oriented Gradients (HOG) features with linear SVMs as classifiers for pedestrian detection, and achieved very good performance in terms of accuracy and computational efficiency. The idea behind HOG is to divide images (or windows) into smaller cells, compute a histogram of oriented gradient in each cell, and then renormalize the histograms in the cells by looking into adjacent larger blocks. To better understand how and why HOG works, recently Vondrick *et. al.* [130] proposed several feature visualization algorithms that help humans see the visual world as a computer might see it.

Recently, exemplar SVMs [95] were proposed for object detection based on HOG and linear SVMs. The basic idea is to train a linear SVM for every individual positive data using massive negative data to produce a rank list, where higher ranks are given to the data points which are more likely objects, and objects are detected using the calibrated scores of the classifier responses by logistic regression. Obviously the computational complexity of this method is very high. To reduce the computational time in training, Hariharan *et. al.* [63] demonstrated that by replacing linear SVMs in exemplar SVMs with Linear Discriminative Analysis (LDA), it can achieve very similar accuracies but much faster training speed. Gharbi *et. al.* [96] proposed a more general method to train exemplar SVMs, which approximates the high dimensional feature space using a simple Gaussian distribution, and computes the normal to the Gaussian at each positive data. However, the long computational time in testing for such methods is not reduced, as we can see in [96]. Also, as claimed in [61], training a linear SVM per positive data may lead to poor generalization of classifiers. Instead, [61] proposed learning a classifier using a subset of positive data, and demonstrated that the classifiers learned in this way are better than exemplar SVMs with faster training and testing speed.

In such methods, linear classifiers such as linear SVMs are employed because of their good computational efficiency, and good accuracy when the feature dimension is relatively high. To extend this type of detection methods, other features can be used with linear classifiers rather than HOG, such as contour fragments [54].

Deformable Part Models (DPMs) [51]. Similar to ISMs, DPMs [51] allow the parts of the object models to be movable within a local region so that the object models fit the data better than rigid HOG features. In other words, DPMs learn the local appearance (*i.e.* parts) and pairwise geometry among the parts to model objects. Typically, in such models at coarse scales, HOG features with linear classifiers is applied to shrink the search space for objects, and at finer scales, DPMs are applied just to the local regions that may contain objects. Limited by the computational complexity, DPMs usually predefine only

Table 2.1: Module comparison in some popular detection methods.

Method	Proposal generation module	Proposal verification module
Shape matching [89]	all possible bounding boxes	distance comparison
ISMs [86]	interest point detection and codeword matching	probabilistic voting
Viola–Jones [129]	all possible bounding boxes	cascaded classifiers with boosting
HOG with linear SVMs [32]	all possible bounding boxes	linear SVMs
Exemplar SVMs [95]	all possible bounding boxes	linear SVMs
MKL [127]	jumping windows	cascaded kernel SVMs with MKL
DPMs [51]	linear filters	latent SVMs

several models per object category, each of which consists of a few parts (*e.g.* 6 parts). The object parts are usually represented using HOG, and their corresponding part models are learned supervisedly and carefully using clustering and linear SVMs, because the quality of learned part models has a great impact on the detection accuracy. Using the learned part models, latent SVMs (*i.e.* linear SVMs with multiple instance learning) are utilized to learn the object models with the star spatial configurations of different part models on the objects by fitting them to the positive data, similar to ISMs. Some work has been done to improve the computational complexity of DPMs, such as using cascaded detection [52], coarse-to-fine search [107], sparselet models [121], and steerable part models [108]. Meanwhile, there are some other work on enriching the flexibility of DPMs, such as mixtures of parts [140], tree-structure DPMs [154], and visual phrases [112] or relational phraselets [34].

Discussions. To summarize, all of these object detection models contain two major modules, object proposal generation and verification, respectively. Table 2.1 lists the module comparison between some popular object detection methods.

From the aspect of computational complexity, however, all the methods in Table 2.1 are *category-dependent*, which means that the computational complexity of each module in each method above will be linearly proportional to the number of object categories. Therefore, with the increase of the number of object categories, the computational time will be longer and longer for all the listed methods.

Currently, linear SVMs [95], kernel SVMs [127] and latent SVMs [51] are the

most popular classifiers in object detection. During testing, the computational complexity of linear SVMs is only linear to the number of object categories, but its accuracy is lower than kernel SVMs and latent SVMs in general. The complexity of kernel SVMs in testing is linear to the number of object categories, and in some cases especially for large-scale datasets, it may increase linearly with the size of training data [28]. However, the support vectors in kernel SVMs can capture the intra-class and inter-class variability of objects implicitly, which are learned by kernels. The complexity of latent SVMs in DPMs is linear to the total number of parts in all the object models, but not the size of training data. However, it is still an open question that how many object models per object category, and how many parts per model will be sufficient for detection.

In order to achieve good computational efficiency and detection generality with acceptable detection accuracy, we would like to make our object models shared with each other as much as possible to reduce the computational complexity. We believe that this is a correct way for object detection, just as suggested by Torralba *et. al.* [123]:

> We believe the computation of shared features will be an essential component of object recognition algorithms as we scale up to large numbers of objects.

In the rest of this chapter, we will review some work on object proposal generation and verification, respectively.

2.2 Localization using Object Proposals

In object detection, we are interested in localizing instances of an object within an image, typically providing as output a set of windows (*i.e.* bounding boxes) containing object instances. Object detection can be treated directly as a regression problem [13], where the task is to predict the location and scale of a single object from an image (or its absence), or a classification problem [32, 51, 95, 127, 129], where the task is to classify every window in an image as either containing an object or not. With the help of nonlinear kernels, more training data, more

features, *etc.*, these methods have achieved better and better detection performances on the public datasets (*e.g.* the detection tasks in the PASCAL VOC challenges), but unfortunately with longer and longer computational time. For instance, DPMs requires computational time linearly proportional to the numbers of object categories and bounding boxes that the classifiers need to verify.

Therefore, the need to accelerate the evaluation process without hurting detection accuracy is thus becoming more important for a successful object detection system. Typically, we do not want to evaluate a complex classifier at all possible positions, scales and aspect ratios in an image, but only a limited number. *We specifically address this problem as the problem of generating proposals of bounding boxes for object localization.* Recently the object proposal generation problem has attracted much attention [1, 27, 52, 77, 78, 84, 109].

Various methods have been proposed to handle the *specific* object proposal generation problem for object detection. Branch-and-bound techniques [77, 78] for instance limit the number of windows that must be evaluated by pruning sets of windows whose response can be bounded. The efficiency of such methods is highly dependent on the strength of the bound, and the ease with which it can be evaluated, which can cause the method to offer limited speed up for nonlinear classifiers, even for kernelized branch-and-bound algorithms [2, 3]. To relax the bound constraint, Lehmann *et. al.* [84] proposed a branch-and-rank algorithm, which introduced the ranking constraint into branch-and-bound to replace the bound constraint.

Alternatively, cascaded approaches use weaker but faster classifiers in the initial stages to prune out negative examples. With the increase of the number of stages in the cascade, more powerful but slower nonlinear classifiers can be used for final decision. In [127] the jumping window approach [27] was utilized to build an initial linear classifier by selecting pairs of discriminative visual words from their associated rectangle regions, which indicate the existence of an object instance, then followed by quasi-linear SVMs and nonlinear SVMs.

Felzenszwalb *et. al.* [51] proposed a deformable part model (DPM) based on latent SVMs in which part filters are only evaluated if a sufficient response is obtained from a global "root" filter, and further [52] proposed a cascaded algorithm

for DPMs. Such approaches have been proved to be efficient, and have generated state-of-the-art results [51]. However, the fact that in [52] the decision scores for detections must be compared across the training data may limit the efficiency of the early cascade stages, where we only need to compare the scores of a classifier at any level of the cascade locally within a single image. Further, such approaches learn a single model which is applied at varying resolutions, while [106] strongly suggests that we should explicitly learn different detectors for different scales. Also, some other work has been done to improve the computational complexity of DPMs, such as coarse-to-fine search [107] and branch-and-bound for DPMs with Dual-Trees data structure [72].

In contrast with specific object proposal generation, there is some recent interesting work [1,40,109] targeting at *generic* object proposal generation, that is, generating object proposals regardless of the object categories (*i.e.* the object/non-object binary setting). Though for object segmentation initially, the method proposed in [40] can be also used for object detection, which produces a bag of category-independent regions (not bounding boxes) using many different visual cues and structured learning for ranking the regions. The candidate regions for ranking are generated by certain segmentation algorithms. Objectness measure [1] combined multiple visual cues to score the windows, and then produced the object proposals by sampling windows with high scores. Based on [1], Rahtu *et. al.* [109] proposed another category-independent cascaded method for proposal generation, where the proposal candidates are sampled from superpixels, which are generated using a segmentation method, according to a prior object localization distribution and then ranked using structured learning with learned features.

2.3 Recognition using Local Classifiers

As we discussed in Section 2.1, all the three widely used SVM based classifiers have their own advantages and drawbacks. To balance classification accuracy and computational efficiency of the classifiers for object proposal verification, we are particularly interested in local classifiers as an alternative, which in general

can have low computational complexity and high classification accuracy for many recognition tasks (*e.g.* the winning method for image classification in VOC2009 [44]). In [71] Kecman and Brooks proved that the stability bounds for local SVMs are tighter than the ones for traditional, global, SVMs. In order to develop an efficient object detection framework, local classifiers are hence suitable.

Before introducing local classifiers, let us revisit binary SVMs first. The basic idea in binary SVMs is to maximize the margin between positive data and negative data. Given a set of training data $\{\mathbf{x}_i, y_i\}_{i=1,\cdots,N}$, where $\forall i, \mathbf{x}_i \in \mathbb{R}^d$ denotes a d-dimensional feature vector and $y_i \in \{\pm 1\}$ denotes its associated label, with the help of slack variables, a binary SVM is mathematically defined as follows:

$$\begin{aligned} &\min_{\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_i \xi_i \\ &\text{s.t.} \ \ \forall i,\ y_i\left(\mathbf{w}^T\mathbf{x}_i + \mathbf{b}\right) \geq 1 - \xi_i, \\ &\qquad \xi_i \geq 0, \end{aligned} \tag{2.3.1}$$

where $(\mathbf{w}, \mathbf{b})$ denote the model parameters for the SVM, $\boldsymbol{\xi}$ denotes the slack variable, $C \geq 0$ denotes a predefined regularization parameter, and $\cdot^T$ denotes the vector transpose operator.

By introducing the kernel tricks into Eq. 2.3.1, we can derive the dual form of kernel based binary SVMs as below:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \ \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j) - \sum_{i=1}^{N}\alpha_i \\ &\text{s.t.} \ \ \forall i,\ 0 \leq \alpha_i \leq C,\ \textstyle\sum_{i=1}^{N} \alpha_i y_i = 1, \end{aligned} \tag{2.3.2}$$

where $\boldsymbol{\alpha}$ denotes the Lagrange multipliers and $\phi(\cdot)$ denotes the kernel mapping function. A kernel SVM tries to select a subset of data points (*i.e.* support vectors) from the *entire* training data and give them positive weights to construct its decision function

$$f(\hat{\mathbf{x}}) = \sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i)^T\phi(\hat{\mathbf{x}}) \tag{2.3.3}$$

for a given test data point $\hat{\mathbf{x}}$.

On the contrary, the local classifiers we mentioned here are learned using local

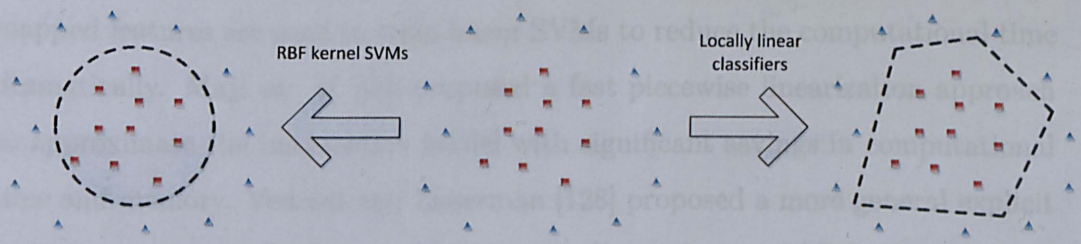


Figure 2.1: Illustration of comparison between RBF-kernel SVMs and locally linear classifiers, where red rectangles and blue triangles represent two classes of data, and the dashed circle and lines denote the decision boundaries of RBF kernel SVMs and the locally linear classifiers, respectively.

context information of each training data point, and classify each test data point based on its local context as well. Formally, we define our binary local classifiers as follows:

Definition 2.1 (Binary Local Classifiers). *We consider a rich family of classifiers as binary local classifiers, whose decision functions for a given data point* $\hat{\mathbf{x}}$ *satisfy the following formula:*

$$f(\hat{\mathbf{x}}) = \sum_{i=1}^{N_z} \alpha_i \psi(\mathbf{z}_i, \hat{\mathbf{x}}) \tag{2.3.4}$$

where $\forall i = 1, \cdots, N_z$, $\mathbf{z}_i \in \mathbb{R}^d$ *is an arbitrary point in the same feature space as* $\hat{\mathbf{x}}$ *lies in,* $\alpha_i \in \mathbb{R}$ *is its associated weight, and* $\psi(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *is a data-dependent similarity measure function.*

By comparing Eq. 2.3.4 with Eq. 2.3.3, we can see that kernel SVMs can be considered as special cases of our local classifiers, where $N_z = N$, $\forall i, \mathbf{z}_i = \mathbf{x}_i$, and $\psi(\mathbf{z}_i, \hat{\mathbf{x}}) = y_i \phi(\mathbf{x}_i)^T \phi(\hat{\mathbf{x}})$. Fig. 2.1 shows an example of comparing RBF kernel SVMs with locally linear classifiers, a type of widely used local classifiers, in a 2D case. As we see, the decision boundaries of RBF kernel SVMs form a regular circle shape in contrast to the irregular shape for locally linear classifiers, because kernel SVMs have prior assumptions on the shapes of decision boundaries, while locally linear classifiers are data-driven.

One of the major problems in kernel SVMs is that they are too computationally intensive for applications with large-scale data sets. With the help of local classifiers, some kernels can be approximated very well using explicit feature maps (*i.e.* constructing the function $\phi(\cdot)$ in Eq. 2.3.3 directly), and the

mapped features are used to train linear SVMs to reduce the computational time dramatically. Maji *et. al.* [94] proposed a fast piecewise linearization approach to approximate the intersection kernel with significant savings in computational time and memory. Vedaldi and Zisserman [128] proposed a more general explicit feature mapping method to approximate the homogeneous additive kernels (*e.g.* the intersection kernel and the χ^2 kernel) based on the Fourier sampling theorem.

However, the optimal decision boundary for a classification problem does not necessarily behave as defined by a kernel. In fact, it could be an arbitrary nonlinear function in the feature space. In such cases, it is still possible to approximate an optimal decision boundary locally using linear functions according to Taylor's theorem. Following this thought, several locally linear classifiers have recently been proposed and successfully applied to object recognition. Zhang *et. al.* [146] proposed an SVM-KNN classifier, where for each test data point, its K nearest neighbors in the training data are found and used to train multi-class linear SVMs. Similar ideas appeared in [137]. In SVM-KNN, searching for K nearest neighbors for each point can be considered as a coding process, and a few coding methods (*e.g.* local coordinate coding (LCC) [143], improved LCC [142], locality-constrained linear coding (LLC) [131], deep coding network [88]) have been proposed for approximating the nonlinear optimal decision boundaries, which are assumed to be Lipschitz smooth functions in order to guarantee theoretical upper bounds of the approximation error. Ladicky and Torr [76] proposed a method for learning locally linear SVMs using encoded data.

Nearest neighbor classifiers [6,15,125,134] have been widely used in computer vision, and they fall in the definition of our local classifiers as well. In [15] Boiman *et. al.* proposed a Naive Bayes Nearest Neighbor classifier (NBNN) based on the nonparametric nearest neighbors for image classification, and achieved good results on some benchmark datasets. Behmo *et. al.* [6] parameterized NBNN by max-margin methods for both image classification and object detection. Tuytelaars *et. al.* [125] built a kernel for SVMs by generating image representations using NBNN. Weinberger and Saul [134] proposed a large-margin nearest neighbor classifier (LMNN) which improves the K-nearest-neighbor (KNN) classifier by learning the Mahalanobis distance metric from labeled examples.

Chapter 3

Object Proposal Generation: Structured Learning

In this chapter, we formulate the object proposal generation problem as a structured learning problem, since we need to predict the object proposals with locations and scales/aspect-ratios (*i.e.* structural information). Particularly, we investigate structural support vector machines (SSVMs) [124] with our proposed scale/aspect-ratio quantization scheme and ranking constraints. Considering the computational efficiency of training and testing our method, we propose a ranking-order decomposition algorithm to solve our SSVMs formulation approximately and efficiently, which can bound the total loss of our formulation under certain conditions. Further we propose a specific two-stage cascade approach for the object proposal generation problem, which we allow to utilize different regularizers, constraints, even features in each stage.

The major differences between our method and the previous related work on object proposal generation [1, 40, 109] are:

- From the view of features, our method only takes simple image gradients as features for learning and testing, while all of the related work above utilize multiple visual cues in images;
- From the view of ranking for proposals, our method utilizes the classification scores (*i.e.* margins) generated by the learned linear classifiers for ranking, rather than some scores from superpixels [40], prior object localization distributions [109], or the combination of multiple visual cues [1, 109];
- From the view of learning, our method formulates the problem into a supervised structured learning framework, while the others involve much more heuristics.

As a result, our method can run much faster than all the other three methods with similar or even better performance.

3.1 Formulation: Structural SVMs

We start with introducing the normal structural SVMs (SSVMs) and one of its variants in Section 3.1.1, then we introduce our scale/aspect-ratio quantization

scheme into SSVMs in Section 3.1.2, and finally by further adding ranking constraints into SSVMs, we propose our own SSVMs for the object proposal generation problem in Section 3.1.3. In contrast to previous work on object detection using structural learning such as [13], where each image associates one loss, our formulation allows every pair of compared patches in an image to have a loss, which essentially makes our method suitable for multi-scale multi-object detection. Besides, an efficient optimization algorithm has been developed for our formulation to handle large-scale data easily (see Section 3.2 for details).

3.1.1 SSVMs and LSE-SSVMs

Structural SVMs (SSVMs) [124] are tools for predicting structured outputs which extend the traditional SVMs. Given a set of training data $(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\cdots,N}$ where $\mathbf{y}_i \in \mathcal{Y}$ denotes a structure and $\Psi(\mathbf{x}_i, \mathbf{y}_i)$ denotes the feature vector associated with data $\mathbf{x}_i$ under the structure $\mathbf{y}_i$, SSVMs [1] can be formulated in the following way [124]:

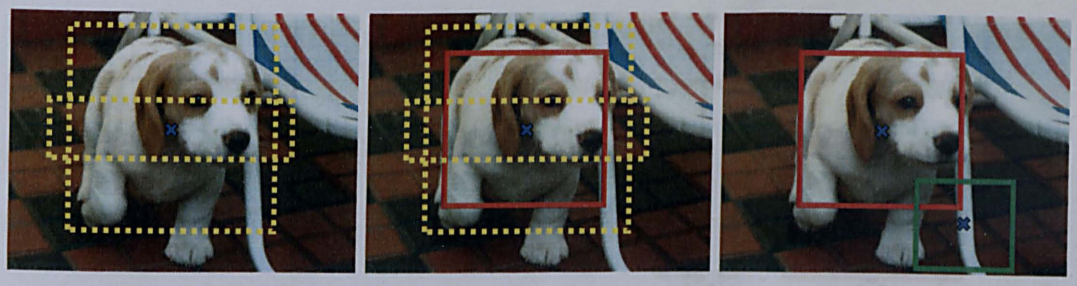
$$\min_{\mathbf{w}} \ \frac{1}{p}\|\mathbf{w}\|_p^p + C\sum_i \max_{\mathbf{y}}\{\ell\{\mathbf{w}\cdot(\Psi(\mathbf{x}_i,\mathbf{y}_i)-\Psi(\mathbf{x}_i,\mathbf{y}))-\Delta(\mathbf{y}_i,\mathbf{y})\}\}, \qquad (3.1.1)$$

where $\mathbf{w}$ denotes the model parameters, $\ell\{\cdot\}$ denotes an arbitrary loss function (*e.g.* 0/1 loss, hinge loss), $\Delta(\mathbf{y}_i, \mathbf{y})$ denotes a loss measuring the difference between an arbitrary structure $\mathbf{y} \in \mathcal{Y}$ and the structure $\mathbf{y}_i$, and $\Delta(\mathbf{y}_i, \mathbf{y}) = 0$ if $\mathbf{y}_i = \mathbf{y}$, otherwise $\Delta(\mathbf{y}_i, \mathbf{y}) \geq 0$; $C \geq 0$ is the predefined regularization parameter; "$\cdot$" denotes the dot product operator between two vectors; $p \in \{1, 2\}$. During testing, the structure of a test data point $\mathbf{x}^*$ is predicted as:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}\in\mathcal{Y}} \{\mathbf{w}\cdot\Psi(\mathbf{x}^*, \mathbf{y})\}. \qquad (3.1.2)$$

To explain how to use SSVMs to formulate the object proposal generation problem, let us take Fig. 3.1(a) for example. Each pixel in the image, denoted by the blue cross, can be considered as an arbitrary data point $\mathbf{x}_i$ in Eq. 3.1.1, an arbitrary window centered at the pixel, denoted by the yellow dashed rectangles,

[1] In the thesis, we consider both ℓ_1- and ℓ_2-norm SSVMs without specific explanation.



(a) SSVMs and LSE-SSVMs (b) Q-LSE-SSVMs (c) QR-LSE-SSVMs

Figure 3.1: Illustration of differences of Structural SVMs (SSVMs), LSE-SSVMs, Q-LSE-SSVMs, and QR-LSE-SSVMs, respectively. Given a pixel, denoted as a blue cross, the windows (*i.e.* yellow dashed rectangles) centered at the pixel in (a) could be many. Using our scale/aspect-ratio quantization scheme, those windows in (a) can be represented by the red solid window centered at the pixel in (b). By relaxation, in (c) windows at different pixels with different quantized scales/aspect-ratios can be compared.

can be taken as the structure variable $\mathbf{y}$, and $\Psi(\mathbf{x}_i, \mathbf{y})$ is the feature within the window $\mathbf{y}$ (*e.g.* the gradient patch, HOG or SIFT). Letting the slack variable $\xi_i = \max_{\mathbf{y}}\{\ell\{\mathbf{w}\cdot(\Psi(\mathbf{x}_i,\mathbf{y}_i)-\Psi(\mathbf{x}_i,\mathbf{y}))-\Delta(\mathbf{y}_i,\mathbf{y})\}\}$, it is only associated with the pixel $\mathbf{x}_i$. Therefore, in our case SSVMs aim to learn a model $\mathbf{w}$ so that at any pixel $\mathbf{x}_i$ in a given image, the margin between the feature defined by the window $\mathbf{y}_i$ and the feature defined by any arbitrary window $\mathbf{y}$ should be no less than the loss $\Delta(\mathbf{y}_i, \mathbf{y})$ with the help of slack variables. When generating object proposals, SSVMs search for the bounding box $\mathbf{y}^*$ at a pixel $\mathbf{x}^*$ which maximizes the margin.

In [105], another type of SSVMs with linear summed error (LSE-SSVMs) was proposed. Different from [124], in LSE-SSVMs the slack variables are dependent on not only data points but also structures, that is,

$$\min_{\mathbf{w}} \ \frac{1}{p}\|\mathbf{w}\|_p^p + C\sum_{i,\mathbf{y}\in\mathcal{Y}} \ell\{\mathbf{w}\cdot(\Psi(\mathbf{x}_i,\mathbf{y}_i)-\Psi(\mathbf{x}_i,\mathbf{y}))-\Delta(\mathbf{y}_i,\mathbf{y})\}. \qquad (3.1.3)$$

To understand LSE-SSVMs in the context of proposal generation, letting the slack variable $\xi_i(\mathbf{y}) = \ell\{\mathbf{w}\cdot(\Psi(\mathbf{x}_i,\mathbf{y}_i)-\Psi(\mathbf{x}_i,\mathbf{y}))-\Delta(\mathbf{y}_i,\mathbf{y})\}$, now it is associated with not only the pixel $\mathbf{x}_i$ but also the structure $\mathbf{y}$. In other words, a slack variable is assigned to each window in an image for optimization, which gives us more flexibility for learning models.

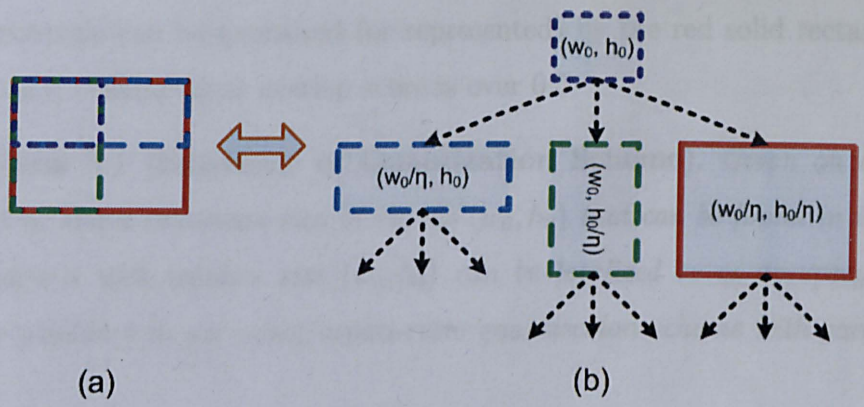


Figure 3.2: Our scale/aspect-ratio quantization scheme can be represented hierarchically. (a) superimposes the four window scales in a mini-quantization scheme with $\eta = 0.5$, and (b) unfolds the scales into a tree structure. The relative widths and heights of the windows are represented by the (w, h) pairs. Such a hierarchy can represent all windows to 0.5-accuracy.

3.1.2 LSE-SSVMs with Quantized Scales/Aspect-ratios (Q-LSE-SSVMs)

The major difficulty of applying SSVMs to our proposal generation problem is the huge structural search space for localizing bounding boxes, which is quadratically proportional to image sizes as addressed in Section 1.2.1. To tackle this challenge, we propose quantizing the scales/aspect-ratios of all the windows in images into several discrete scales/aspect-ratios to shrink the search space for localization. Based on Definition 1.2 in Section 1.2, which is the definition of *η-accuracy*, we propose the following scale/aspect-ratio quantization scheme:

Definition 3.1 (Quantized Scale/Aspect-ratio). *Given an overlap threshold $\eta \geq 0$, a window s in an image can be quantized into a quantized scale/aspect-ratio $\mathcal{T}$ if and only if $\exists t \in \mathcal{T}$ such that s can be localized to η-accuracy, where t is a window with the quantized scale/aspect-ratio.*

Fig. 3.2 illustrates our scale/aspect-ratio quantization scheme for a 0.5-accuracy case in a hierarchy. Given a minimum size of objects (w_0, h_0) that can be found in images, our quantization scheme can be easily represented by a ternary tree. Another example is shown in Fig. 3.1(b), where the bigger dotted

yellow rectangle can be quantized (or represented) by the red solid rectangle to 0.5-accuracy, because their overlap score is over 0.5.

Proposition 3.1 (Existence of Quantization Scheme). *Given an overlap threshold η_0 and a minimum size of objects (w_0, h_0) that can be found in images, any window s with window size (w_s, h_s) can be localized to η_0-accuracy by at least one window t in our scale/aspect-ratio quantization scheme with parameter $\eta \geq \eta_0$.*

Proof. According to Fig. 3.2, we can construct a subset of windows in our quantized scheme by computing $a \in \left\{ \lfloor \log_\eta \frac{w_s}{w_0} \rfloor, \lceil \log_\eta \frac{w_s}{w_0} \rceil \right\}$ and $b \in \left\{ \lfloor \log_\eta \frac{h_s}{h_0} \rfloor, \lceil \log_\eta \frac{h_s}{h_0} \rceil \right\}$, where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote the floor and ceiling operations, respectively. Letting $t(w_t, h_t)$ be a window with quantized window size (w_t, h_t), the overlap between s and t can be calculated as follows:

$$\begin{aligned} \exists a, b,\ o\left(s, t\left(w_0\eta^a, h_0\eta^b\right)\right) &= \frac{\min\left\{w_s, w_0\eta^a\right\} \cdot \min\left\{h_s, h_0\eta^b\right\}}{\max\left\{w_s, w_0\eta^a\right\} \cdot \max\left\{h_s, h_0\eta^b\right\}} \\ &= \eta^{\left|a - \log_\eta \frac{w_s}{w_0}\right| + \left|b - \log_\eta \frac{h_s}{h_0}\right|} \geq \eta^{0.5+0.5} = \eta \geq \eta_0. \end{aligned} \tag{3.1.4}$$

That is, s can be localized to η_0-accuracy by t. □

Proposition 3.2 (Minimum Number of Quantized Scales/Aspect-ratios). *Given an overlap threshold $\eta \geq 0$, a minimum size (w_0, h_0) and a maximum size (w, h) of objects that can be found in images, the minimum number of quantized scales/aspect-ratios that is sufficient to localize any object is bounded by $\lceil \log_\eta \frac{w_0}{w} \rceil \lceil \log_\eta \frac{h_0}{h} \rceil$.*

Proof. According to the construction of our scale/aspect-ratio quantization scheme, the numbers of quantized scales that will cover the maximum size of objects w and h are $\lceil \log_\eta \frac{w_0}{w} \rceil$ and $\lceil \log_\eta \frac{h_0}{h} \rceil$, respectively. Therefore, the maximum number of quantized scales that is needed to cover all possible objects in images is $\lceil \log_\eta \frac{w_0}{w} \rceil \lceil \log_\eta \frac{h_0}{h} \rceil$. □

Proposition 3.3 (Search Space for Localization using Quantization Scheme). *Given an overlap threshold $0 < \eta < 1$, the minimum size of quantized scales/aspect-ratio (w_0, h_0), and the maximum image size (W, H), the search space for localizing bounding boxes using our quantization scheme is $O\left(W \cdot \lceil \log_{\frac{1}{\eta}} \frac{W}{w_0} \rceil \cdot H \cdot \lceil \log_{\frac{1}{\eta}} \frac{H}{h_0} \rceil\right)$.*

Proof. According to Proposition 3.2, the search space for quantized scales is reduced to $\lceil \log_{\frac{1}{\eta}} \frac{W}{w_0} \rceil \lceil \log_{\frac{1}{\eta}} \frac{H}{h_0} \rceil$, while the search space for positions of proposals keeps the same $O(W \cdot H)$. Therefore, the search space for localization using quantization scheme is $O\left(W \cdot \lceil \log_{\frac{1}{\eta}} \frac{W}{w_0} \rceil \cdot H \cdot \lceil \log_{\frac{1}{\eta}} \frac{H}{h_0} \rceil\right)$. □

Now by introducing our scale/aspect-ratio quantization scheme into LSE-SSVMs, we can formulate our Q-LSE-SSVMs as follows:

$$\min_{\mathbf{w}} \ \frac{1}{p}\|\mathbf{w}\|_p^p + C \sum_{i,\bar{\mathbf{y}} \in \bar{\mathcal{Y}}} \ell\{\mathbf{w} \cdot (\Psi(\mathbf{x}_i, \bar{\mathbf{y}}_i) - \Psi(\mathbf{x}_i, \bar{\mathbf{y}})) - \Delta(\bar{\mathbf{y}}_i, \bar{\mathbf{y}})\} \qquad (3.1.5)$$

where $\bar{\mathcal{Y}}$, $\bar{\mathbf{y}}$, and $\bar{\mathbf{y}}_i$ denote the quantized scale/aspect-ratio space, a quantized scale/aspect-ratio in this space, and the quantized ground-truth scale/aspect-ratio for the pixel $\mathbf{x}_i$, respectively. As illustrated in Fig. 3.1(b), the yellow dashed windows can be quantized into a red solid window centered at the pixel.

3.1.3 Q-LSE-SSVMs with Ranking Constraints (QR-LSE-SSVMs)

Notice that in Q-LSE-SSVMs, only the windows centered at the same location in an image can be compared, which is the same case in LSE-SVMs. However, for proposal generation, all the windows within an image should be comparable with each other without any limit on location or scale/aspect-ratio. Therefore, in order to gain more flexibility on window comparison, we introduce the ranking constraints into Q-LSE-SSVMs, as illustrated in Fig. 3.1(c), and rewrite it in Eq. 3.1.6 below:

$$\min_{\mathbf{w}} \ \frac{1}{p}\|\mathbf{w}\|_p^p + C \sum_{i,j,m_i,m_j} \ell\{\mathbf{w} \cdot (\Psi(\mathbf{x}_i, \bar{\mathbf{y}}_{m_i}) - \Psi(\mathbf{x}_j, \bar{\mathbf{y}}_{m_j})) - \Delta_{ij}(\bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j})\} \qquad (3.1.6)$$

$$\text{s.t.} \ \ \forall i, j, m_i, m_j, \quad r(\mathbf{x}_i, \bar{\mathbf{y}}_{m_i}) \geq r(\mathbf{x}_j, \bar{\mathbf{y}}_{m_j}), \quad \bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j} \in \bar{\mathcal{Y}},$$

where $\Delta_{ij}(\bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j}) \geq 0$ denotes a loss measuring the difference between a window centered at $\mathbf{x}_i$ with size $\bar{\mathbf{y}}_{m_i}$ and a window centered at $\mathbf{x}_j$ with size $\bar{\mathbf{y}}_{m_j}$, and $r(\mathbf{x}_i, \bar{\mathbf{y}}_{m_i})$ and $r(\mathbf{x}_j, \bar{\mathbf{y}}_{m_j})$ denote their corresponding ranks, respectively. Ideally

the data with higher ranks should have larger margins. Clearly, if the ranking list is defined as $\forall i, \bar{\mathbf{y}} \in \bar{\mathcal{Y}}, r(\mathbf{x}_i, \bar{\mathbf{y}}_i) \geq r(\mathbf{x}_i, \bar{\mathbf{y}})$, Eq. 3.1.6 turns into Eq. 3.1.5. Letting the slack variable $\xi_{ij}(\bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j}) = \ell\{\mathbf{w} \cdot (\Psi(\mathbf{x}_i, \bar{\mathbf{y}}_{m_i}) - \Psi(\mathbf{x}_j, \bar{\mathbf{y}}_{m_j})) - \Delta_{ij}(\bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j})\}$, now it is associated with any pair of location and window size in images, which leads to much more variables for learning.

Definition 3.2 (Maximum Overlap). *Given an image I and the ground-truth bounding boxes of multiple objects $g_{1 \cdots m_I}$ in I, the* maximum overlap *of a window s in I is defined as*

$$o_s = \max_{i \in \{1, \cdots, m_I\}} o(s, g_i). \tag{3.1.7}$$

where $o(s, g_i)$ denotes the overlap score between s and g_i.

Definition 3.3 (Correct Object Proposals). *Given an overlap threshold $\eta \geq 0$, a window s is considered as a* correct object proposal *in an image if and only if $o_s \geq \eta$.*

For object proposal generation, we can construct the ranking list for a training image based on the maximum overlap scores of windows for learning models, the higher the scores, the higher the ranks. During testing, the proposals with larger margins should have better chances of localizing object instances to η-accuracy.

Notice that in Eq. 3.1.6, the model $\mathbf{w}$ is enforced to be the same for all the quantized scales/aspect-ratios, which limits the discriminative power of the model. By relaxing this condition, eventually we propose our SSVMs formulation for proposal generation, QR-LSE-SSVMs, as follows:

$$\begin{aligned} \min_{\mathbf{w}} \ & \frac{1}{p} \sum_{m=1}^{|\bar{\mathcal{Y}}|} \|\mathbf{w}_m\|_p^p + C \sum_{i,j,m_i,m_j} \ell\{\mathbf{w}_{m_i} \cdot \Psi(\mathbf{x}_i, \bar{\mathbf{y}}_{m_i}) - \mathbf{w}_{m_j} \cdot \Psi(\mathbf{x}_j, \bar{\mathbf{y}}_{m_j}) - \Delta_{ij}(\bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j})\} \\ \text{s.t.} \ & \forall i, j, m_i, m_j, \quad r(\mathbf{x}_i, \bar{\mathbf{y}}_{m_i}) \geq r(\mathbf{x}_j, \bar{\mathbf{y}}_{m_j}), \quad \bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j} \in \bar{\mathcal{Y}}, \end{aligned} \tag{3.1.8}$$

where $|\mathcal{Y}|$ denotes the number of quantized scales/aspect-ratios.

3.2 Optimization: Ranking-Order Decomposition

Basically Eq. 3.1.8 can be considered as a multi-task learning problem which contains millions of constraints and thousands of model parameters $\mathbf{w}$'s for optimization. This will be a big challenge for solving this problem efficiently using general optimization algorithms, such as cutting-plane methods [70, 105, 153], projected sub-gradient methods [153] and even stochastic gradient descent methods [105]. Therefore, we propose a ranking-order decomposition algorithm for solving Eq. 3.1.8 approximately and efficiently.

3.2.1 General Algorithm

Our algorithm is inspired by the dual decomposition method for learning Markov Random Fields (MRF) [73, 74], where an MRF (*i.e.* master problem) is decomposed into smaller sub-graphs (*i.e.* slave problems) using dual decomposition, which can be solved more efficiently based on the current master model and later their solutions are used for updating the master model. This process is repeated until it converges.

The basic idea of our algorithm is to decompose the original problem in Eq. 3.1.8 into slave problems with much fewer constraints and model parameters inside, each of which can be solved independently and much more efficiently. Then the solutions of these slave problems are passed to the master problem for updating its parameters. This process is repeated. Fig. 3.3 illustrates our ranking-order decomposition algorithm using data points associated with two structures.

In order to apply this algorithm to solve the original problem in Eq. 3.1.8, we introduce two variables $\bar{\mathbf{w}}$ and $\mathbf{z}$ such that $\forall m, \mathbf{w}_m = \bar{\mathbf{w}}_m z_m$. Then the

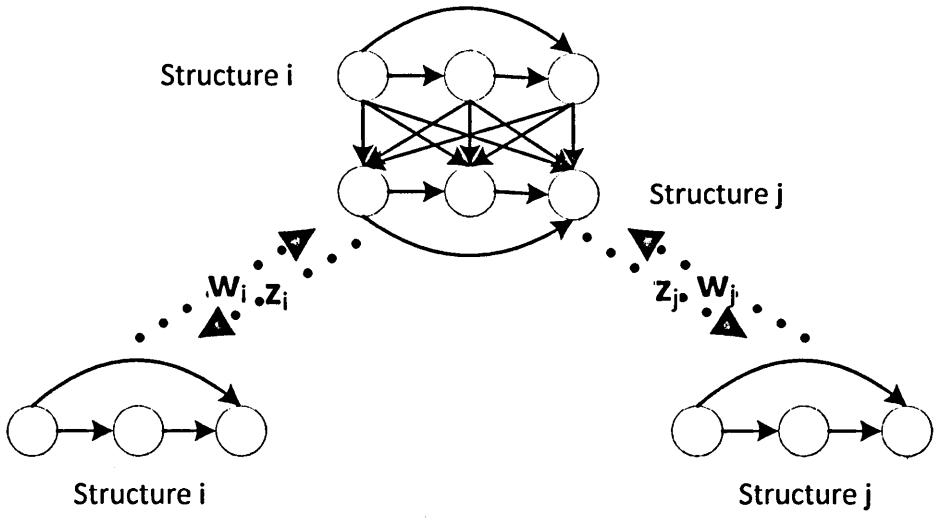


Figure 3.3: Illustration of ranking-order decomposition for optimizing QR-LSE-SSVMs given two structures i and j, where each circle denotes a data point, each directed solid edge denotes a ranking order from data with a higher rank to data with a lower rank, each directed dotted edge denotes the message (parameter) passed along the direction, and the circles in each row share the same structure. Using the terminology in Dual Decomposition, the top model is the master problem, and the two bottom models are the slave problems.

optimization problem in Eq. 3.1.8 can be rewritten as follows:

$$\min_{\bar{\mathbf{w}},\mathbf{z}} \frac{1}{p}\sum_{m=1}^{|\bar{\mathcal{Y}}|} |z_m|^p \|\bar{\mathbf{w}}_m\|_p^p + C \sum_{i,j,m_i,m_j} \ell\{z_{m_i}\left[\bar{\mathbf{w}}_m \cdot \Psi(\mathbf{x}_i, \bar{\mathbf{y}}_m)\right] - z_{m_j}\left[\bar{\mathbf{w}}_m \cdot \Psi(\mathbf{x}_j, \bar{\mathbf{y}}_m)\right] - \Delta_{ij}(\bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j})\} \tag{3.2.1}$$

$$\text{s.t.} \quad \forall i, j, m_i, m_j,\ r(\mathbf{x}_i, \bar{\mathbf{y}}_{m_i}) \geq r(\mathbf{x}_j, \bar{\mathbf{y}}_{m_j}),\ \bar{\mathbf{y}}_{m_i}, \bar{\mathbf{y}}_{m_j} \in \bar{\mathcal{Y}}.$$

It turns out that the optimization problem in Eq. 3.2.1 can be considered as a biconvex optimization problem. To solve it using our ranking-order decomposition algorithm, we can decompose it into the following optimization problem, where Eq. 3.2.2 defines the slave problems, and Eq. 3.2.3 defines the master problem:

$$\min_{\bar{\mathbf{w}}} \frac{1}{p}\sum_{m=1}^{|\bar{\mathcal{Y}}|} |z_m|^p \|\bar{\mathbf{w}}_m\|_p^p + C \sum_{i,j,m} \ell\{z_m\left[\bar{\mathbf{w}}_m \cdot \Psi(\mathbf{x}_i, \bar{\mathbf{y}}_m) - \bar{\mathbf{w}}_m \cdot \Psi(\mathbf{x}_j, \bar{\mathbf{y}}_m)\right] - \Delta_{ij}(\bar{\mathbf{y}}_m)\}$$

$$\text{s.t.} \quad \forall i, j, m, \quad r(\mathbf{x}_i, \bar{\mathbf{y}}_m) \geq r(\mathbf{x}_j, \bar{\mathbf{y}}_m), \quad \bar{\mathbf{y}}_m \in \bar{\mathcal{Y}}; \tag{3.2.2}$$

$$\min_{\mathbf{z}} \frac{1}{p}\sum_{m=1}^{|\bar{\mathcal{Y}}|}|z_m|^p\|\bar{\mathbf{w}}_m\|_p^p + C\sum_{i,j,m_i,m_j}\ell\{z_{m_i}[\bar{\mathbf{w}}_m\cdot\Psi(\mathbf{x}_i,\bar{\mathbf{y}}_m)] - z_{m_j}[\bar{\mathbf{w}}_m\cdot\Psi(\mathbf{x}_j,\bar{\mathbf{y}}_m)] - \Delta_{ij}(\bar{\mathbf{y}}_{m_i},\bar{\mathbf{y}}_{m_j})\} \tag{3.2.3}$$

$$\text{s.t. } \forall i,j,m_i,m_j,\ r(\mathbf{x}_i,\bar{\mathbf{y}}_{m_i}) \geq r(\mathbf{x}_j,\bar{\mathbf{y}}_{m_j}),\ \bar{\mathbf{y}}_{m_i},\bar{\mathbf{y}}_{m_j}\in\bar{\mathcal{Y}}.$$

Solving Eq. 3.2.2 is equivalent to solving $|\mathcal{Y}|$ independent much smaller minimization problems, each of which learns only a model $\bar{\mathbf{w}}_m$ using the data with structure $\bar{\mathbf{y}}_m$. This can be done efficiently using cutting-plane methods [70, 105, 153] or stochastic gradient descent methods [105] in a parallelized manner. The optimization in Eq. 3.2.2 enforces the learned ranking orders of data within each structure to fit the predefined ranking list in Eq. 3.2.2 as well as possible, regardless of the ranking orders between the data with different structures.

Solving Eq. 3.2.3 is rather easy using cutting-plane methods or stochastic gradient descent methods as well, with very small number of parameters in $\mathbf{z}$ though the number of constraints in Eq. 3.2.3 is the same as that in Eq. 3.2.1. The slack variables will disappear in the dual form of Eq. 3.2.3. This optimization enforces the learned ranking orders among data points to fit the predefined ranking list in the original problem in Eq. 3.2.1 as well as possible.

Overall, solving Eq. 3.2.2 and Eq. 3.2.3 sequentially and repeatedly gives us the solution which can be used to solve QR-LSE-SSVMs approximately. Unfortunately, unlike Dual Decomposition, our ranking-order decomposition algorithm cannot guarantee the convergence in each iteration. However, under some conditions, our algorithm can provide the lower and upper bounds of the minimum loss in the original problem.

Theorem 3.1 (Minimum Loss Bounds for Solving QR-LSE-SSVMs). *Let ℓ be an arbitrary loss function and $C = +\infty$ in Eq. 3.1.8, Eq. 3.2.2 and Eq. 3.2.3, respectively. Suppose in each iteration the minimum losses in Eq. 3.2.2 and Eq. 3.2.3 are equal to L_1 and L_2, then the minimum loss in Eq. 3.1.8 is lower-bounded by L_1 and upper-bounded by L_2.*

Proof. Let $\mathbf{w}$, $\bar{\mathbf{w}}$, and $\mathbf{z}$ be the optimal solutions in Eq. 3.1.8, Eq. 3.2.2, and Eq. 3.2.3, respectively, and $\forall m, \tilde{\mathbf{w}}_m = z_m\bar{\mathbf{w}}_m$, denoted by $\tilde{\mathbf{w}}$ for the corresponding solution. Because of $C = +\infty$, the objective function in each optimization

problem above is equivalent to computing the total loss in the data. We denote the objective functions above as f_1, f_2, and f_3, respectively, then $f_2(\bar{\mathbf{w}}) = L_1$ and $f_3(\tilde{\mathbf{w}}) = L_2$.

$\because$ $\bar{\mathbf{w}}$ minimizes Eq. 3.2.2,

$\therefore$ $f_2(\bar{\mathbf{w}}) \leq f_2(\mathbf{w})$.

$\because$ $\mathbf{w}$ minimizes Eq. 3.1.8,

$\therefore$ Based on our ranking-order decomposition algorithm, we have $f_1(\mathbf{w}) \leq f_1(\tilde{\mathbf{w}}) = f_3(\tilde{\mathbf{w}})$.

$\because$ The constraint set in Eq. 3.2.2 is a subset of that in Eq. 3.1.8,

$\therefore$ $f_2(\mathbf{w}) \leq f_1(\mathbf{w})$.

$\therefore$ $L_1 = f_2(\bar{\mathbf{w}}) \leq f_2(\mathbf{w}) \leq f_1(\mathbf{w}) \leq f_1(\tilde{\mathbf{w}}) = f_3(\tilde{\mathbf{w}}) = L_2$. □

3.2.2 A Two-Stage Cascaded Model for Object Proposal Generation

In this section, we explain how to apply our ranking-order decomposition algorithm to generate object proposals. We will use the hinge loss function in the optimization, and repeat the iteration between the master problem and the slave problems only once.

Cascaded classifiers are good tools for handling extremely imbalanced data, that is, too many negatives and too few positives. Object detection is one of the applications with extremely imbalanced data, where the objects of interest in an image are very few but the non-object are many, considering the huge structural search space of windows. In the cascade, only "positives" are passed on as outputs of each stage, which have higher ranks than those "negatives".

In our training data, each image is annotated with the bounding boxes of objects of interest. Our goal is to give higher ranks to the correct object proposals than the wrong ones within each image in a very efficient way, such that the windows at the top of the ranking list can be taken as our final object proposals. Fig. 3.4 summarizes our cascaded method for generating proposals.

For ease of explanation of our cascaded approach, we list the main notations used in the following sections in Table 3.1.

Table 3.1: Some notations used in the explanation of our cascaded model for object proposal generation.

Notation	Definition
T	The set of all possible windows in an image.
S	The set of all possible windows in our window quantization scheme.
$S(w,h)$	The set of all the windows in an image with width w and height h.
$o(t,s)$	The overlap between window $t \in T$ and window $s \in S$.
o_t	The maximum overlap for window $t \in T$ in an image.
$\eta \in [0,1]$	Overlap threshold for proposal generation.
k	A given scale/aspect-ratio in our quantization scheme.
S_k	The set of all the windows which can be represented to η-accuracy at quantized scale/aspect-ratio k.
$\mathbf{w}_k$, $\mathbf{z}_k$	Learned linear classifiers for quantized scale/aspect-ratio k at Stage I and II, respectively.
$\mathbf{v}$	A channel response feature vector used in Stage II for learning $\mathbf{z}$.

3.2.2.1 Stage I: Scale/Aspect-ratio Specific Ranking

The first stage of our cascade aims to pass on a number of object proposals based on different sliding windows at each of a set of quantized scales and aspect ratios to the next stage. This is done by learning a linear classifier for each quantized scale/aspect-ratio separately.

Individual Classifier Learning

Given η and a set of quantized scales/aspect-ratios, for each scale k [2] we wish to learn a linear classifier $f_1(\mathbf{x}_s;\mathbf{w}_k) = \mathbf{w}_k \cdot \mathbf{x}_s$, as suggested in [106], to rank a window $s \in S_k$, where $\mathbf{x}_s$ denotes its feature vector, among all the windows in S_k.

Ideally, we expect that the ranking score for any window $s_i \in S_k \bigcap T_I$ with $o_{s_i} \geq \eta$ is always higher than that of any window $s_j \in T$ with $o_{s_j} < \eta$, where $\bigcap$ denotes the intersection between two sets. That is, for $\mathbf{w}_k$ we require that within the image I all the corresponding positive training windows $I_k^+ = \{s_i \in S_k \bigcap T_I | o_{s_i} \geq \eta\}$ should be ranked above all the training negatives $I^- = \{s_j \in T_I | o_{s_j} < \eta\}$. This leads us to formulate the problem as a ranking SVM, which

[2] In the following sections, we refer to scale k as quantized scale/aspect-ratio k for short.

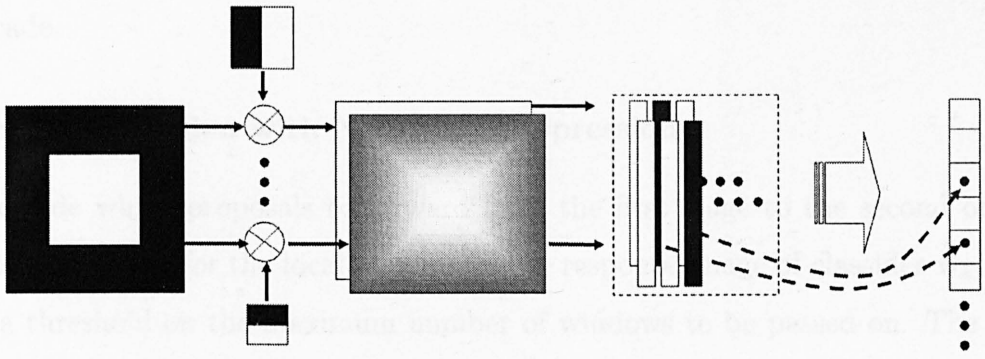


Figure 3.4: Summary of our cascaded method for generating proposals. An image (a) is first convolved with a set of linear classifiers at varying scales/aspect-ratios (b) producing response images (c). Local maxima are extracted from each response image, and the corresponding windows with top ranking scores are forwarded to the second stage of the cascade. Each proposed window is associated with a feature vector (d), and a second round of ranking orders these proposals (e) so that the true positives (marked as black) are pushed towards the top. Our method outputs the top ranking windows in this final ordering.

can be expressed as below [3]:

$$\begin{aligned} \min_{\mathbf{w}_k,\boldsymbol{\xi}} \quad & \frac{1}{p}\|\mathbf{w}_k\|_p^p + C\sum_{i,j,n}\xi_{ij}^n \\ \text{s.t.} \quad & \forall n, i \in I_{kn}^+, j \in I_n^-,\ \mathbf{w}_k \cdot (\mathbf{x}_i^n - \mathbf{x}_j^n) \geq 1 - \xi_{ij}^n, \\ & \xi_{ij}^n \geq 0. \end{aligned} \tag{3.2.4}$$

Here, $\mathbf{x}_i^n$ and $\mathbf{x}_j^n$ are the feature vectors associated with positive window i and negative window j in training image I_n, respectively, $\boldsymbol{\xi}$ are the slack variables, and $C \geq 0$ is a predefined regularization parameter. We set the loss Δ in Eq. 3.2.2 to 1.

Recall that the purpose of learning the individual classifiers is to build the proposal pool for further usage, so the constraints in Eq. 3.2.4 are restricted to *one* quantized scale in *one* image. Therefore, the ranking scores from each classifier are incompatible across scales/aspect-ratios, necessitating the second stage in the

[3] If taking $\mathbf{0}$ as the dummy feature whose rank is higher than negatives but lower than positives, then only comparing data with the dummy feature turns Eq. 3.2.4 into a standard SVM. We denote the solution of Eq. 3.2.4 as "ℓ_p-w/r", and the solution of Eq. 3.2.4 with the dummy feature as "ℓ_p-o/r".

cascade.

Proposal Selection with Non-Max Suppression

To decide which proposals to forward from the first stage to the second of the cascade, we look for the local maxima in the response image of classifier $\mathbf{w}_k$, and set a threshold on the maximum number of windows to be passed on. The first stage thus has two controlling parameters. The first, $\gamma \in [0, 2]$, specifies the ratio between the size of the neighborhood over which we search for the local maxima, and the reference window size for each classifier. This is the non-max suppression parameter. The second, d_1, specifies the maximum number of windows, which are the top d_1 ranked local maxima, that can be passed on from any scale.

3.2.2.2 Stage II: Ranking Score Calibration

The first stage of the cascade generates a number of proposal windows at each scale k for image I. The second stage then re-ranks these globally, so that the best proposals across scales are forwarded. To achieve this, we introduce a new feature vector for each window, $\mathbf{v}$, which consists of the channel responses of the classifier at the first stage. This is a relaxation of linear scalability in Eq. 3.2.3 in the general algorithm. For instance, $\mathbf{v}$ could be a 4-dimensional feature vector if feature $\mathbf{x}$ is divided into 4 segments without overlaps, each of which gives a response to the corresponding classifier. The reason for splitting $\mathbf{x}$ into different segments is that we could make full use of information in different segments to improve the calibration performance.

Based on $\mathbf{v}$, we can re-rank each window i by the decision function $f(\mathbf{v}_i) = \mathbf{z}_{k_i} \cdot \mathbf{v}_i + e_{k_i}$, where k_i denotes the quantized scale/aspect-ratio associated with window i, $\mathbf{z}_{k_i}$ is a set of coefficients for scale k_i that we would like to learn, and e_{k_i} is the corresponding bias term. Similarly, we formulate this learning problem

as a multiclass ranking SVM as shown in Eq. 3.2.5 [4]:

$$\begin{aligned} \min_{\mathbf{z},\mathbf{e},\boldsymbol{\xi}} \ & \frac{1}{p}\|\mathbf{z}\|_p^p + C\sum_{i,j,n}\xi_{ij}^n \\ \text{s.t.} \ & \forall n, i \in \hat{I}_n^+, j \in \hat{I}_n^-,\ \mathbf{z}_{k_i}\cdot\mathbf{v}_i^n - \mathbf{z}_{k_j}\cdot\mathbf{v}_j^n + e_{k_i} - e_{k_j} \geq 1 - \xi_{ij}^n, \\ & \xi_{ij}^n \geq 0. \end{aligned} \tag{3.2.5}$$

Here, $\hat{I}_n^+$ and $\hat{I}_n^-$ denote the positive and negative windows in image I_n forwarded from the first stage of the cascade across different quantized scales/aspect-ratios. Similarly, the loss Δ in Eq. 3.2.3 is set to 1. We ignore the weights for $\mathbf{z}$ in the original formula in Eq. 3.2.3 and just focus on re-ranking the object proposals, since the ranking error will dominate the objective function in Eq. 3.2.3.

In this way, all the proposal candidates in each image can be ranked, and the top d_2 windows are then considered as the final object proposals.

3.2.3 Computational Complexity

Our method involves the application of simple linear classifiers to the images, and as such is dominated by the complexity of 2D convolution with linear kernel which must be applied to each image. The complexity can thus be approximated as $O(K \times R \times (W \times H) \times (W_I \times H_I))$, where K denotes the number of individual classifiers learned in Stage I, R denotes the number of segments used in Stage II, (W, H) denotes the filter size, and (W_I, H_I) denotes the resized image size. We note that ***our complexity is therefore (largely) independent of the number of potential proposals let through at each stage*** (d_1, d_2)***, unlike methods which include non-linear classifiers [77, 127], and the number of object categories for generating object proposals.*** Also, our algorithm is quite suitable for parallel computing, which will reduce the running time dramatically.

[4] Similar to Eq. 3.2.4 with the dummy feature, we continue to use the same notations for the solution of Eq. 3.2.5.

3.3 Experiments

We test our method in two tasks: *specific* object proposal generation and *generic* object proposal generation, respectively.

3.3.1 Specific Object Proposal Generation

We design a comprehensive set of experiments to assess the impact of various parameters and design choices in our model. We also compare our performance against a state-of-the-art method [77] and show substantial improvement. We measure our performance in terms of *recall-overlap* curves [77, 127], which provides a means of assessing the potential information preserved for further processing, and the speed of our method. We test on PASCAL VOC2006 [48] and VOC2010 [45] datasets. VOC2006 consists of 10 object categories, 5304 images of natural scenes, with object labels and their corresponding ground-truth bounding boxes released for training, validation and test sets. VOC2010 consists of 20 object categories, 21738 natural images, and object labels and their corresponding ground-truth bounding boxes are available for training and validation sets only. For training and testing, we split VOC2006 into train/validation and (train+validation)/test, and VOC2010 into train/validation, respectively.

3.3.1.1 Implementation

For this task, we simply employ "$\ell_2 - w/r$" and "$\ell_1 - w/r$" for Eq. 3.2.4 and Eq. 3.2.5, respectively, to explore the impacts of different parameters on the performance in terms of object recall and running time. We assume that in images the minimum and maximum sizes of objects can be localized by 16×16 and 512×512 pixel patches to η-accuracy, respectively. These two sizes are decided based on the statistics of object and image sizes in VOC challenges.

We use simple zero-mean gradient features to learn each classifier $\mathbf{w}_k$ at the first stage. In detail, we first convert all the images into gray scale, and represent all the object ground-truth bounding boxes to η-accuracy using our scale/aspect-ratio quantization scheme to provide positive windows. After randomly selecting

negatives across scales, all windows are resized to a fixed feature window size (W, H), and then for each pixel, the magnitude and orientation of its gradient is calculated. Orientation weights are then calculated in a fixed set of R orientation channels for assigning the gradients to build sub-features $\mathbf{x}_r$ ($r \in \{1, \cdots, R\}$) separately. Finally, by concatenating all $\mathbf{x}_r$, a $(W \times H \times R)$-dimensional vector is generated consisting of spatial and gradient information. To handle the different illumination contrasts in images, we subtract the mean value to produce a feature vector $\mathbf{x}_i$ for window i, and the learned classifiers are thus guaranteed to be zero-mean vectors (avoiding the need for a bias in Eq. 3.2.4). The features used at the second stage, $\mathbf{v}$, are produced by concatenating the classifier responses from each orientation channel at the first stage, producing an R-dimensional vector where $v_r = \mathbf{w}_{k,r} \cdot \mathbf{x}_r$. At test time, to generate features $\mathbf{x}$, we simply resize the image for each scale k by the ratio of its reference window to (W, H), and then apply the learned classifier $\mathbf{w}_k$ by 2D convolution.

The remaining global parameters of the cascade are γ, d_1 and d_2, which affect the trade-off between the number of positive windows we retain at each stage, and the amount of noise we allow through. We investigate the effects of these parameters in the following sections.

3.3.1.2 VOC2006

Cascade Design: γ, d_1, d_2

We first evaluate the effects of the following cascade parameters: the neighborhood size for finding local maxima γ in the first stage, and the number of windows to be passed on at the first and second stages, d_1 and d_2. Fig. 3.5 shows the performance of various parameter settings in terms of the *area under curve* (AUC) (*i.e.* recall-overlap curve) for the class bicycle in VOC2006. We can see that as we move from left to right (increasing d_2) the area with highest AUC scores shifts from bottom right to top left. This implies that more candidates selected from the first cascade stage (high d_1) and a higher γ are appropriate for low-recall regimes (low d_2), while the opposite is true for high-recall (high d_2). For further experiments we choose values $d_1 = 50$ and $\gamma = 0.6$, which work well across the d_2 settings.

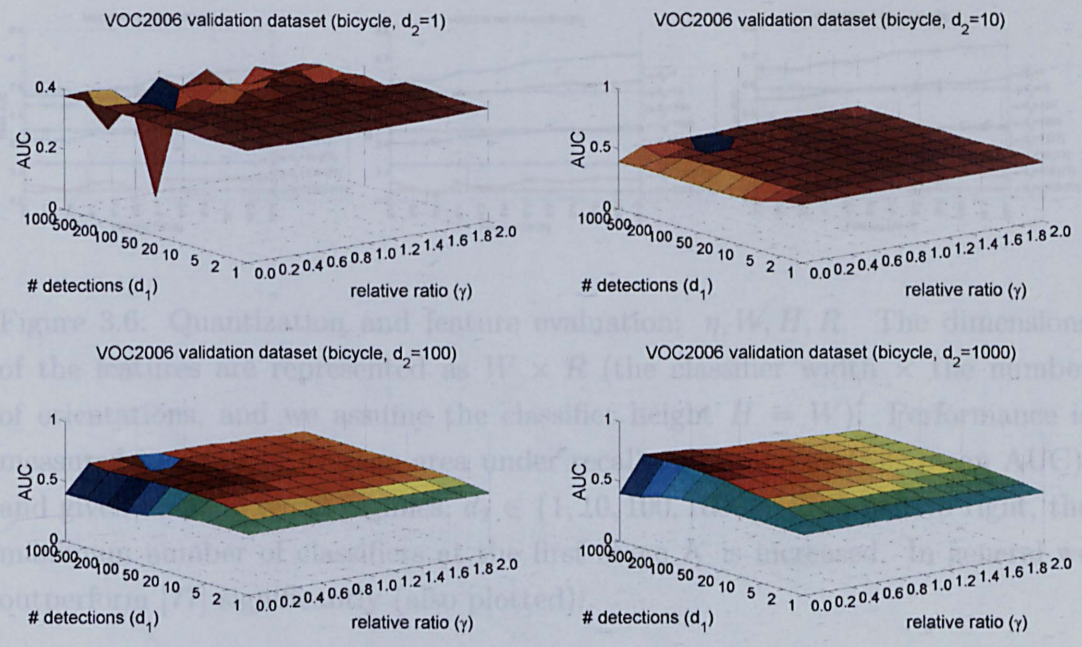


Figure 3.5: Cascade design evaluation: γ, d_1, d_2. Higher *area under curve* (AUC) scores are represented by warmer color. The effects of varying γ (neighborhood size) and d_1 (number of candidates selected from the first cascade stage) are tested under various recall regimes by varying d_2 (number of candidates selected from the second cascade stage). See Section 3.3.1.2 for commentary.

Quantization and Features: K, W, H, R

We next assess the effects on the performance of the features we use (*i.e.* the size of the classifiers (W, H) and the number of orientations R), and the maximum number of classifiers learned at the first stage K (determined by the overlap threshold η). Fig. 3.6 summarizes the results, considering 4 different recall regimes by varying the number of output proposals $d_2 \in \{1, 10, 100, 1000\}$, and comparing them against the best results of [77] in these regimes. Performance is measured again in terms of AUC (averaged across classes). We can see that, as expected, performance increases both as the size of the classifiers and number of orientations increase (W, H, R), and as K increases. However, both of these factors imply longer computational time as discussed in Section 3.2.3. We see though that even with the smallest feature size, 2×2 with 1 orientation (*i.e.* only 4-dimensional features), we improve substantially on [77] in most cases and achieve comparable performance otherwise. We will offer further comparison which takes computational time into account.

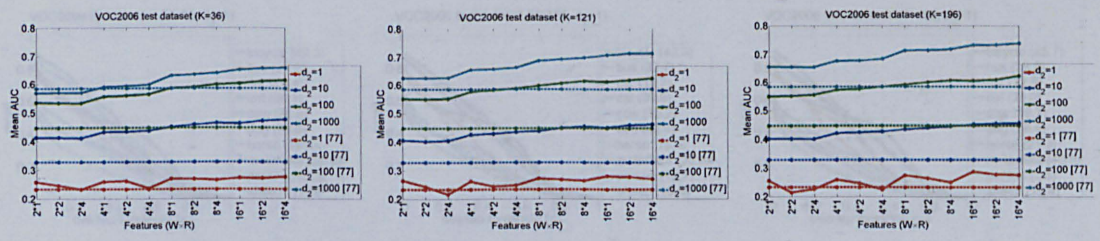


Figure 3.6: Quantization and feature evaluation: η, W, H, R. The dimensions of the features are represented as $W \times R$ (the classifier width $\times$ the number of orientations, and we assume the classifier height $H = W$). Performance is measured in terms of average area under recall-overlap curve (*i.e.* mean AUC), and given under 4 recall regimes, $d_2 \in \{1, 10, 100, 1000\}$. From left to right, the maximum number of classifiers at the first stage K is increased. In general we outperform [77] significantly (also plotted).

Recall-Overlap Evaluation

Fig. 3.7 breaks the VOC2006 results down by class, and displays the recall-overlap curves that were used to calculate Fig. 3.6 for the case of $(W, H, R) = (16, 16, 4)$. We can see here the movement of the curves towards the top-right both as we allow more output proposals ($d_2 \in \{1, 10, 100, 1000\}$) and as we increase $K = \{36, 121, 196\}$. We recall that our quantized scales/aspect-ratios are designed to cover bounding boxes to a particular overlap threshold of η, so $K \in \{36, 121, 196\}$ corresponds to $\eta \in \{0.5, 0.67, 0.75\}$ respectively. This affects the performance observed, and on the $K = 36$ graph for instance, we see that the curves are high for $\eta \leq 0.5$, but then drop quickly. Similar drops can be observed in the $K = 121$ and $K = 196$ graphs for the corresponding later points in the curves, $\eta = 0.67$ and 0.75, implying our quantization is capturing the desired information. The average recalls when $d_2 = 1000$ and $\eta = 0.5$ are **95.8%, 97.1%, 96.0%** for $K \in \{36, 121, 196\}$ respectively.

Recall-Proposal Evaluation

As a pre-process step for object detection, the object recall with a certain η using a fixed number of proposals is more important, because this recall determines the best performance that an object detection system can achieve.

In Fig. 3.8(a)-(b) we show how the recall is effected as we increase the number

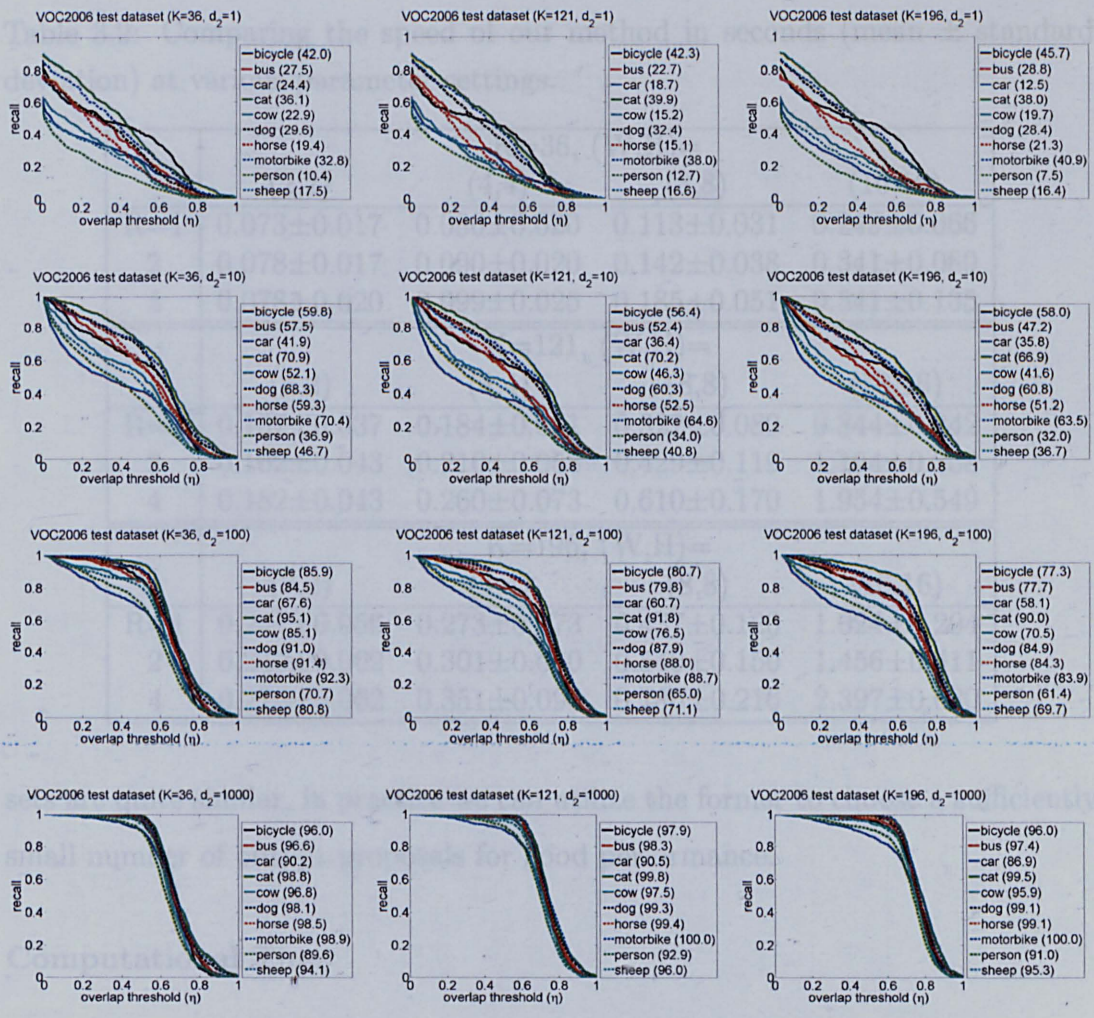


Figure 3.7: Recall-overlap evaluation for VOC2006. Recall-overlap curves are plotted for individual classes using $d_2 \in \{1, 10, 100, 1000\}$ from left to right, and $K \in \{36, 121, 196\}$ from top to bottom. All curves are plotted using $(W, H, R) = (16, 16, 4)$. The numbers shown in the legends are the recall percentages when the overlap threshold η is set to 0.5.

of output proposals d_2 from 1 to 1000 on the validation and test sets of VOC2006. We fix $(W, H, R, K) = (16, 16, 4, 36)$. We can see that on both validation and test datasets when d_2 is beyond 400, the curves hardly change, which means the AUC for $d_2 = 400$ and $d_2 = 1000$ will be very similar. We believe that this property of our approach is useful for detection tasks, because it narrows down significantly the total number of windows that classifiers need to check while losing few correct detections. In fact, some categories need far fewer proposals to achieve good performance. For instance, for the cat category, 100 output proposals saturates performance. Since the behaviors of our approach on both validation and test

Table 3.2: Comparing the speed of our method in seconds (mean ± standard deviation) at various parameter settings.

	K=36, (W,H)=			
	(2,2)	(4,4)	(8,8)	(16,16)
R=1	0.073±0.017	0.080±0.020	0.113±0.031	0.249±0.066
2	0.078±0.017	0.090±0.020	0.142±0.038	0.341±0.089
4	0.078±0.020	0.099±0.025	0.185±0.051	0.541±0.135
	K=121, (W,H)=			
	(2,2)	(4,4)	(8,8)	(16,16)
R=1	0.157±0.037	0.184±0.051	0.322±0.089	0.844±0.242
2	0.162±0.043	0.210±0.058	0.429±0.119	1.194±0.338
4	0.182±0.043	0.260±0.073	0.610±0.170	1.954±0.549
	K=196, (W,H)=			
	(2,2)	(4,4)	(8,8)	(16,16)
R=1	0.241±0.056	0.273±0.073	0.437±0.120	1.024±0.294
2	0.247±0.062	0.301±0.080	0.549±0.150	1.456±0.411
4	0.270±0.062	0.351±0.090	0.786±0.216	2.397±0.680

sets are quite similar, in practice we can utilize the former to choose a sufficiently small number of output proposals for good performance.

Computational Time

Details of our computational time are shown in Table 3.2. Our implementation is a mixture of Matlab and C++ code, and is run on a single thread of Intel Xeon W3680@3.33GHz. The computational time shown here includes all the steps at the test stage, *i.e.* calculating features, 2D convolution, proposal selection, and ranking score calibration. The computational time in [77] is 0.47 ± 0.01 [5].

As we see, with increase in the size of the feature windows (W, H), the number of orientation channels R, and the maximum number of classifiers learned at the first stage K, computational time grows roughly linearly in the log-scale. This demonstrates that the computational complexity of our approach can be approximated by the complexity of 2D convolution.

[5] In [77], the computational time is only for training models without considering the time for feature extraction based on a 2.8 GHz PC.

Table 3.3: Comparing the performance of our method in terms of AUC (%) with that of [77]. We show our performance at two settings, the fastest setting $(W, H, R, K) = (2, 2, 1, 36)$ and the best setting $(W, H, R, K) = (16, 16, 4, 121)$. Both settings improve on [77] substantially. On average, for testing time per image, the fastest setting needs about 0.07(s) and the best setting needs about 2.0(s).

Method	bicycle				bus			
	d_2 =1	10	100	1000	d_2 =1	10	100	1000
best in [77]	25.0	38.5	50.7	62.4	19.4	28.0	41.9	58.8
(W,H,R,K)=(2,2,1,36)	34.7	46.7	56.3	58.8	20.0	35.8	54.9	57.9
(W,H,R,K)=(16,16,4,121)	**35.0**	**51.0**	**65.4**	**71.4**	**29.7**	**49.2**	**65.5**	**72.0**

	car				cat				cow			
d_2 =1	10	100	1000	d_2 =1	10	100	1000	d_2 =1	10	100	1000
25.2	31.6	39.4	49.6	**44.7**	56.7	67.9	**76.7**	15.8	24.6	36.9	52.5
10.1	23.0	44.8	53.1	40.2	54.5	61.4	62.1	19.4	34.1	48.8	53.9
16.8	**33.4**	**51.2**	**67.4**	39.8	**58.1**	**70.3**	73.9	**21.6**	**40.6**	**59.1**	**69.9**

	dog				horse				motorbike			
d_2 =1	10	100	1000	d_2 =1	10	100	1000	d_2 =1	10	100	1000
37.7	49.0	61.2	71.8	21.5	31.7	47.5	63.7	24.7	36.1	49.8	63.4
33.1	51.1	58.9	60.6	25.8	46.4	56.4	58.1	26.4	36.2	60.5	61.8
33.2	**53.4**	**67.4**	**72.3**	**27.8**	**49.8**	**65.9**	**72.4**	**35.8**	**55.2**	**69.3**	**74.3**

	person				sheep				average			
d_2 =1	10	100	1000	d_2 =1	10	100	1000	d_2 =1	10	100	1000
7.9	14.4	24.7	41.7	12.0	18.4	28.4	44.2	23.4	32.9	44.8	58.5
11.9	30.3	46.3	53.5	12.0	28.6	46.6	52.6	23.4	38.7	53.5	57.2
13.4	**32.9**	**52.5**	**67.4**	**17.6**	**37.5**	**57.6**	**69.7**	**27.1**	**46.1**	**62.4**	**71.1**

AUC Comparison

We can see on Fig. 3.6 that these all offer further substantial improvements, and we make a closer comparison in Table 3.3 by comparing AUC values of [77] with our results at our fastest or best settings. Averaging across the four output settings ($d_2 \in \{1, 10, 100, 1000\}$), [77] achieves 39.9%, while we achieve **43.2%** using our fastest setting $(W, H, R, K) = (2, 2, 1, 36)$, and **51.7%** using our best setting $(W, H, R, K) = (16, 16, 4, 121)$. Our approach is thus quicker, and offers a substantial improvement in output quality to [77].

Contribution of Scale and Aspect Ratio

To verify our two-stage cascade, involving separate ranking of scales and aspect ratios followed by a calibration, is contributing to our performance, we give further results in Table 3.4 where during learning the individual classifiers we compare our full system against restricted cases where we (a) use only one quantization

Table 3.4: Comparing the performance of our method in terms of AUC (%) when no scale/aspect-ratio information is included during learning the classifiers (*i.e.* single classifier), when only aspect ratio information is included, and when both scale and aspect ratio are included.

Method	bicycle				bus			
	d_2 =1	10	100	1000	d_2 =1	10	100	1000
single	29.1	**50.3**	62.7	**66.5**	20.6	43.2	58.8	65.3
ratio	31.7	50.2	61.7	65.0	22.2	44.6	59.8	65.6
scale & ratio	**35.5**	49.9	**62.8**	65.7	**30.5**	**50.3**	**62.9**	**66.8**

car				cat				cow			
d_2 =1	10	100	1000	d_2 =1	10	100	1000	d_2 =1	10	100	1000
23.9	38.0	52.7	63.2	**40.1**	**60.0**	67.0	69.1	**26.8**	48.0	61.6	65.8
23.0	**40.1**	**55.5**	**64.7**	38.6	59.8	67.2	68.6	**26.8**	**49.1**	**61.9**	**66.2**
22.5	36.9	53.9	63.8	39.4	59.2	**68.1**	**69.5**	22.3	43.1	59.1	64.9

dog				horse				motorbike			
d_2 =1	10	100	1000	d_2 =1	10	100	1000	d_2 =1	10	100	1000
32.4	**55.8**	**65.8**	67.2	24.7	47.0	62.8	65.3	34.5	52.3	65.3	67.9
32.1	54.9	64.9	66.9	22.0	46.0	63.0	66.1	32.4	52.5	66.0	67.7
30.8	54.6	65.4	**68.1**	**27.1**	**51.2**	**64.0**	**66.6**	**35.8**	**55.7**	**66.4**	**68.2**

person				sheep				average			
d_2 =1	10	100	1000	d_2 =1	10	100	1000	d_2 =1	10	100	1000
15.2	**37.0**	**55.4**	61.9	19.3	43.5	58.8	64.4	26.7	47.5	61.1	65.7
13.8	35.2	54.9	61.9	**22.1**	**44.3**	**59.5**	**65.0**	26.5	**47.7**	61.4	65.8
14.4	36.0	54.2	**62.1**	19.0	40.0	57.8	64.5	**27.7**	**47.7**	**61.5**	**66.0**

level, and so do not use scale and aspect ratio information (thus learning only one classifier), and (b) use only aspect ratio information (learning one classifier per aspect ratio). In each case, the feature size is set to $(W, H, R) = (16, 16, 4)$ and $K = 36$. As shown, we have an average gain in performance as scale and aspect ratio information is added (although in certain classes the effect is less pronounced, and aspect ratio plays a more important role than scale in some).

3.3.1.3 VOC2010

We repeat our recall-overlap and recall-proposal evaluations on VOC2010. In Fig. 3.8(c) we see a similar pattern across classes to the VOC2006 validation and test sets, implying that thresholds can be generalized (even for individual classes) across these datasets. In Fig. 3.9 we see a similar pattern of results to Fig. 3.7 (also using the setting $(W, H, R) = (16, 16, 4)$). The average recalls when $d_2 = 1000$ and $\eta = 0.5$ are **86.2%**, **92.7%**, **91.0%** for $K \in \{36, 121, 196\}$ respectively, which are comparable to those in Section 3.3.1.2. We therefore believe that our approach is robust and efficient across datasets.

3.3.2 Generic Object Proposal Generation

For this task, we learn only one object model per quantized scale/aspect-ratio by using all the object instances in the training data as positives to train a single binary object/non-object filter and output object proposals per image during testing, no matter what classes the object instances belong to. This is our default learn and testing procedure without specific mention.

We test our method on PASCAL VOC2007 [42] and VOC2012 [47], respectively. Both VOC2007 and VOC2012 contain 20 object categories. VOC2007 consists of 9963 natural images with object labels and their corresponding ground-truth bounding boxes released for training, validation and test sets. VOC2012 consists of 17125 natural images, and object labels and their corresponding ground-truth bounding boxes are available for training and validation sets only.

Based on the experimental results in Section 3.3.1 and considering the computational efficiency as well, we set $\gamma = 0.6$, $d_1 = 50$, $(W, H, R) = (16, 16, 1)$, $\eta = 0.5$ without specific explanation.

3.3.2.1 VOC2007

We first test different cascade settings on this data using different combinations of K and ℓ in Eq. 3.2.4 and Eq. 3.2.5, and then compare our method with [1] [6] and [109] [7], respectively. We use the training/validation dataset, consisting of 5011 images, to train our model, and test it on the test dataset, comprising 4952 images.

Cascade Setting Comparison

Fig. 3.10 summarizes the comparison results, where we run our implementation for three times, and report the mean and standard deviation of our results. From the top 3 settings in each sub-figure, we can see that (1) In general, the performances using different settings are close to each other; (2) In Stage I, the method $\ell_1 - o/r$ seems to work best, which trains ℓ_1-norm SVMs, rather than ranking

[6] We downloaded their public code and precomputed windows for VOC2007 from `http://groups.inf.ed.ac.uk/calvin/objectness/`.

[7] We downloaded their public code and precomputed windows for VOC2007 from `http://www.cse.oulu.fi/CMV/Downloads/ObjectDetection`.

Table 3.5: AUC comparison on VOC2007 using 1000 proposals in Fig. 3.11 and Fig. 3.13.

Method	AUC (objects)	AUC (classes)
Ours ($\ell_1 - o/r + \ell_1 - o/r$)	64.5%	(65.1±2.2)%
[1]	64.9%	(66.8±4.2)%
[109]	**67.4%**	**(70.8±8.3)%**

SVMs; (3) In Stage II, both methods $\ell_1 - o/r$ and $\ell_2 - w/r$ seem to work better than others, the first training ℓ_1-norm SVMs and the second training ranking SVMs; (4) With a larger K, the AUC under 1000 proposals becomes larger, while the AUC under a fewer proposals (*i.e.* 1, 10, 100) becomes smaller.

It surprises us that the $\ell_1 - o/r$ SVMs work so well in our cascade, because usually ℓ_2-norm SVMs work better than ℓ_1-norm SVMs for the classification tasks [49]. We believe that ℓ_1-norm SVMs actually select the discriminant features and suppress non-discriminant ones between objects and non-objects.

Recall-Overlap Evaluation

Fig. 3.11 shows our comparison results on VOC2007 (top) and VOC2012 (bottom), respectively. From the curves, we can see that our method has a similar behavior to [1], and their AUC values are close to each other, and at $\eta = 0.5$, in most cases our method and [1] achieve higher object recall than [109]. However, in terms of quantity of proposals, [109] is the best among these methods, because its curves drop quickly when η is larger than around 0.75, while the curves of ours and [109] drop when η is larger than around 0.55. This observation indicates that compared to our method and [1], the correct detection proposals outputted by [109] are closer to the ground-truth bounding boxes of objects.

Fig. 3.13 breaks the VOC2007 results in Fig. 3.11 down by class using 1000 proposals, and displays the recall-overlap curves. Similar observation to Fig. 3.11 can be made. Table 3.5 summarizes the AUC comparison on VOC2007.

Recall-Proposal Evaluation

In Fig. 3.12(left) we show how the recalls of different methods are effected as we increase the number of output proposals d_2 from 1 to 1000 on VOC2007. We

Table 3.6: Object recall comparison on VOC2007 using 1000 proposals as shown in Fig. 3.12 and Fig. 3.14.

Method	Recall (objects)	Recall (classes)
Ours ($\ell_1 - o/r + \ell_1 - o/r$)	**93.8%**	**(95.1±3.5)%**
[1]	88.6%	(92.0±6.7)%
[109]	77.7%	(82.8±12.8)%

Table 3.7: Object recall comparison on VOC2007 using different numbers of proposals.

Method	10 Prop.	100 Prop.	1000 Prop.
Ours ($\ell_1 - o/r + \ell_1 - o/r$)	**46.4%**	**78.7%**	**93.1%**
[1]	41.0%	71.0%	91.0%

can see that when d_2 is beyond 200, the curves become flatter and flatter, similar to Fig. 3.8. From the comparison of the 4 cascade settings, $\ell_1 - o/r + \ell_1 - o/r$ performs best. Compared with [1,109], our method has a similar behavior to [1], and both are better than [109] significantly.

Similarly, Fig. 3.14 breaks the VOC2007 results in Fig. 3.12 down by class and displays the recall-proposal curves. As we see, again some categories need far fewer proposals to achieve good performance. For instance, for the dog category, 100 output proposals saturate performance. Table 3.6 lists the object recall comparison on VOC2007.

Especially, here we also perform a same experiment used in objectness [1]. We divide the 20 object categories into two sets. Same as [1], we use the 14 categories (*i.e.* aeroplane, bicycle, boat, bottle, bus, chair, diningtable, horse, motorbike, person, pottedplant, sofa, train, tvmonitor) as testing categories, and the rest as training categories, which means that these 14 categories are unseen during training. The images containing objects within the training categories in the training/validation dataset are utilized as the training data for learning our models, and the images containing objects within the testing categories in the test dataset are utilized as the test data for evaluating our models. This experiment is designed for exploring the generality of the proposal methods. Table 3.7 lists our comparison results between ours and objectness [1]. Still our method outperforms [1] in terms of object recall given the number of proposals.

Table 3.8: Computational time comparison on VOC2007 in second per image with 1000 proposals.

Method	Computational time
Ours	**0.25±0.07**
[1]	3.58±0.25
[109]	2.22±0.42

Table 3.9: AUC comparison on VOC2012 using 1000 proposals as shown in Fig. 3.11 and Fig. 3.15.

Method	AUC (objects)	AUC (classes)
Ours ($\ell_1 - o/r + \ell_1 - o/r$)	64.2%	(64.9±3.4)%
[1]	**64.6%**	**(65.9±5.2)%**
[109]	60.5%	(63.1±10.4)%

Computational Time

The computational time comparison of the three methods is listed in Table 3.8. Our implementation is a mixture of Matlab and C++ code, just like [1, 109], and all the codes are run on a single thread of Intel Xeon W3680@3.33GHz. The computational time shown here includes all the steps during testing, starting from loading images. Clearly, our method is 10 times faster than [1, 109].

3.3.2.2 VOC2012

We repeat our recall-overlap and recall-proposal evaluations on the VOC2012 training/validation dataset, as shown in Fig. 3.11, Fig. 3.12(right), Fig. 3.15 and Fig. 3.16. We did not re-train the model for each method. Instead, we simply applied the models learned from VOC2007 training/validation dataset directly to the training/validation dataset in VOC2012. Similar patterns can be observed to VOC2007.

Particularly, by comparing Table 3.5, Table 3.6, Table 3.9 and Table 3.10, we can see that our method is quite robust across the datasets, and the most stable among the three methods across different categories with the minimum standard deviation.

Table 3.10: Object recall comparison on VOC2012 using 1000 proposals as shown in Fig. 3.12 and Fig. 3.16.

Method	Recall (objects)	Recall (classes)
Ours ($\ell_1 - o/r + \ell_1 - o/r$)	**91.7%**	**(92.5±5.6)%**
[1]	87.8%	(89.9±9.1)%
[109]	68.3%	(71.7±15.3)%

3.4 Conclusion

We have introduced structural SVMs with our proposed scale/aspect-ratio quantization scheme and ranking constraints (QR-LSE-SSVMs) for the object proposal generation problem. A general ranking-order decomposition algorithm was proposed to solve QR-LSE-SSVMs efficiently and approximately with theoretically guaranteed lower and upper bounds of the total loss in QR-LSE-SSVMs.

Particularly, for our proposal problem, we further proposed a two-stage cascaded model, whose computational complexity per image is largely dependent on only the sizes of images and filters, but not the numbers of proposals and object categories. We envisage that the cascaded model can be used as the initial stages of a complete object detection pipeline, even for real-time object detection. Our method naturally incorporates scale and aspect ratio information about objects, which are treated separately in the first stage of the cascade, and we emphasize the flexibility of the method, where different types of features could easily be incorporated at this stage. At the second stage, all the proposal candidates are re-ranked by calibrating their ranking scores from the first stage to generate our final object proposals, which are among the top of the ranking list.

Our method is both fast and efficient, and we have shown a substantial improvement in speed and recall over recent related work [1,77,109]. Besides object detection, we believe that our method will contribute to many other research areas, such as segmentation [23] and stereo matching [14].

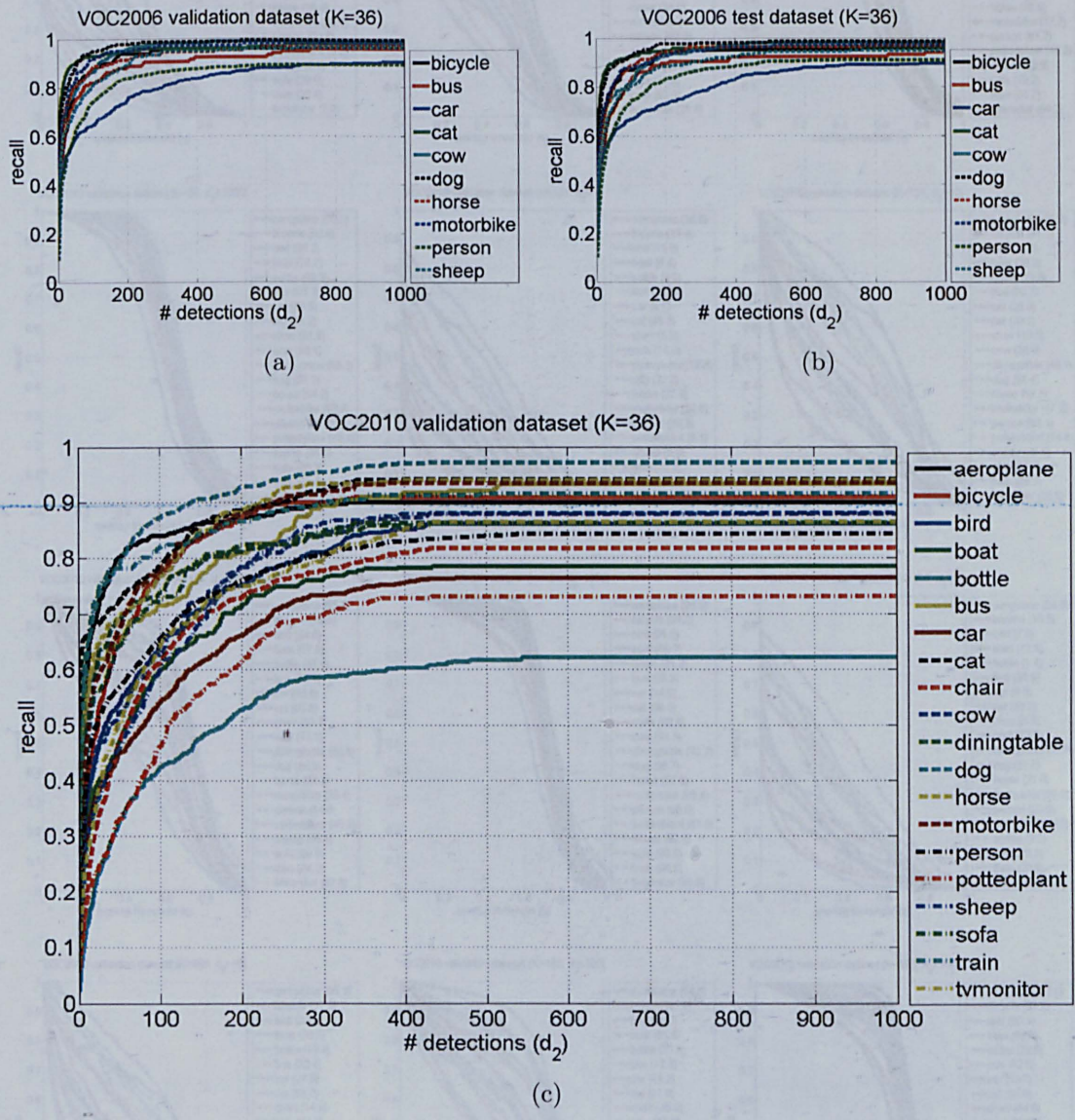


Figure 3.8: Recall-proposal evaluation. (a) VOC2006 validation set, (b) VOC2006 test set, (c) VOC2010 validation set. Recall is measured against increasing numbers of output proposals, d_2. Other parameters are fixed at $(W, H, R, K) = (16, 16, 4, 36)$. Notice that the curves are similar for different classes in all cases, implying we can generalize thresholds from one case to another.

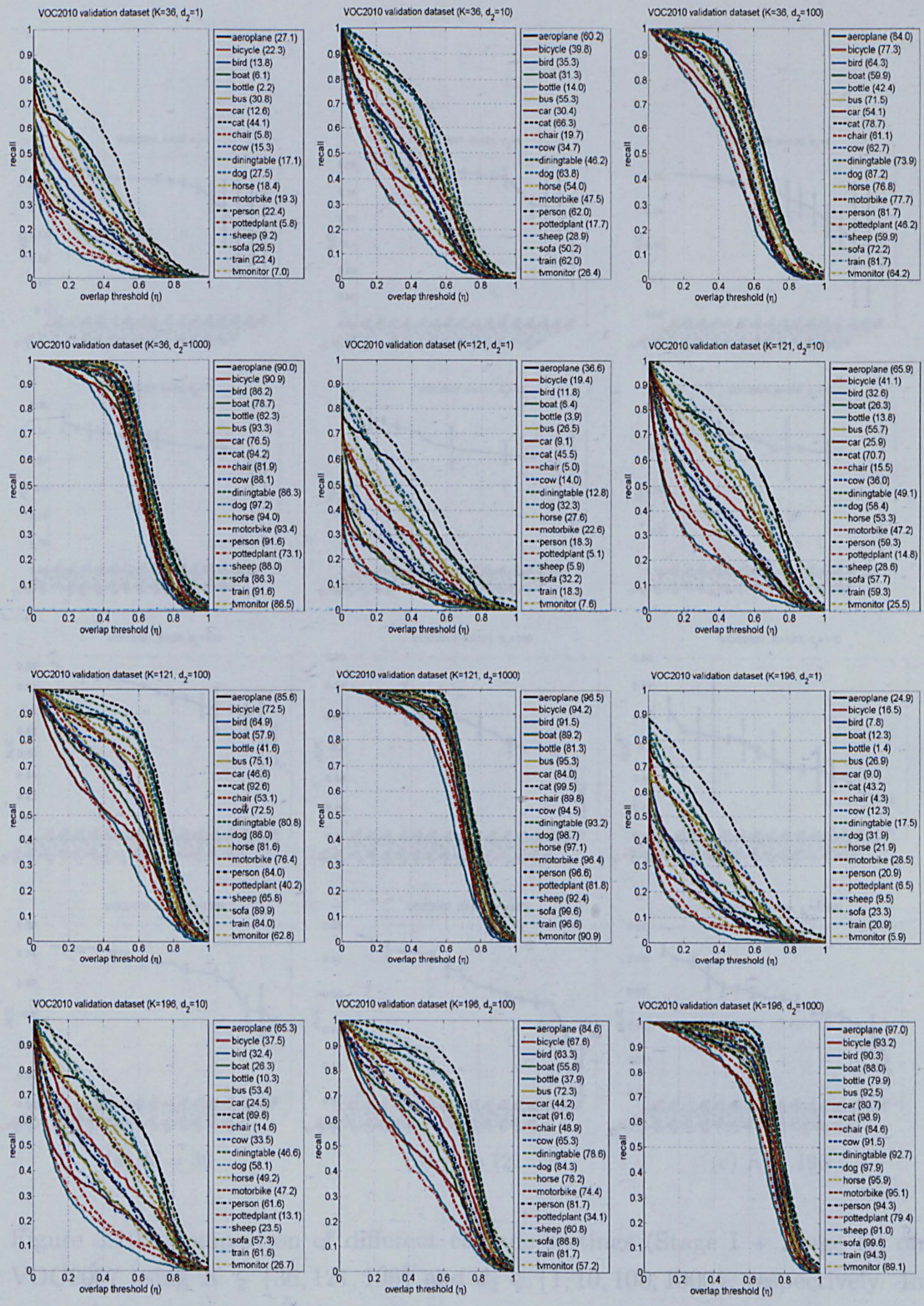
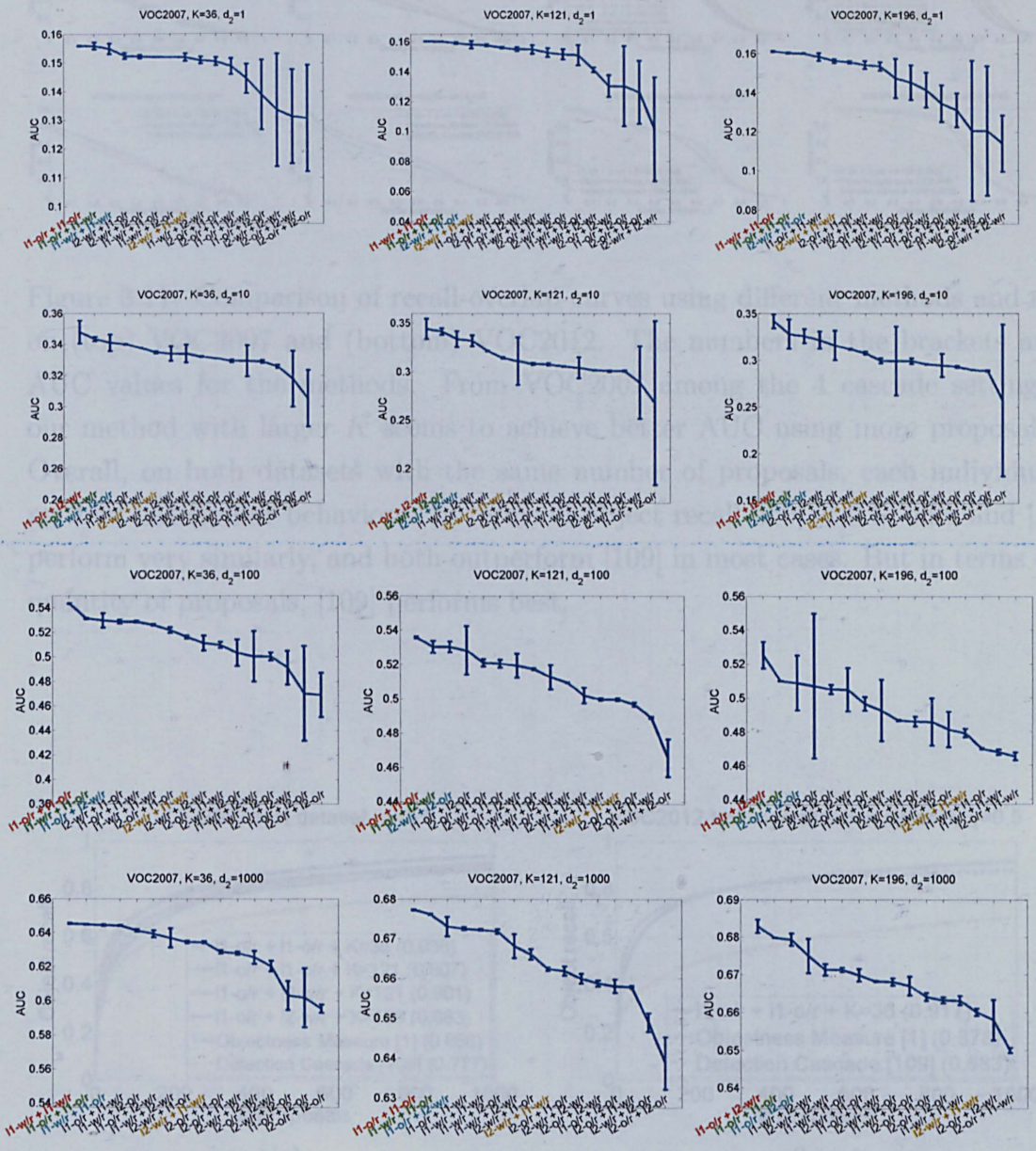


Figure 3.9: Recall-overlap evaluation for VOC2010. Recall-overlap curves are plotted for individual classes using $d_2 \in \{1, 10, 100, 1000\}$ from left to right, and $K \in \{36, 121, 196\}$ from top to bottom. All curves are plotted using $(W, H, R) = (16, 16, 4)$. The numbers shown in the legends are the recall percentages when the overlap threshold η is set to 0.5.



(a) $K = 36$ (b) $K = 121$ (c) $K = 196$

Figure 3.10: Comparison of different cascade settings (Stage I + Stage II) on VOC2007 using $K \in \{36, 121, 196\}$ and $d_2 \in \{1, 10, 100, 1000\}$, respectively. In each sub-figure, the cascade settings are sorted in descending order based on the mean of *area under object recall-overlap curves* (AUC), the top 3 settings are colored by red, green, and cyan, respectively.

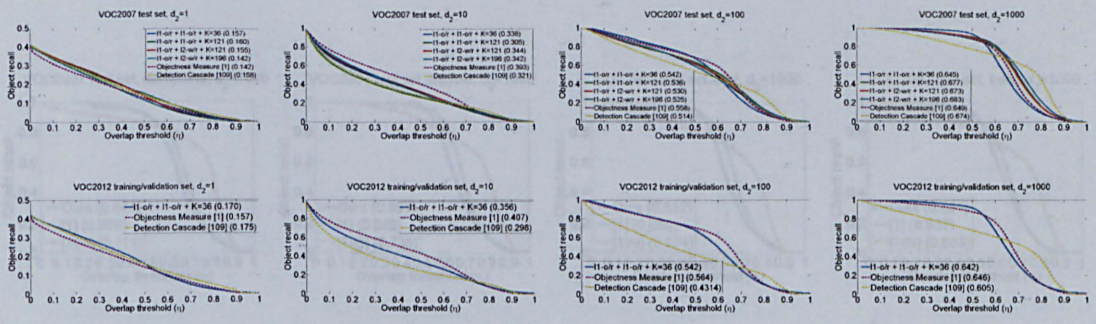
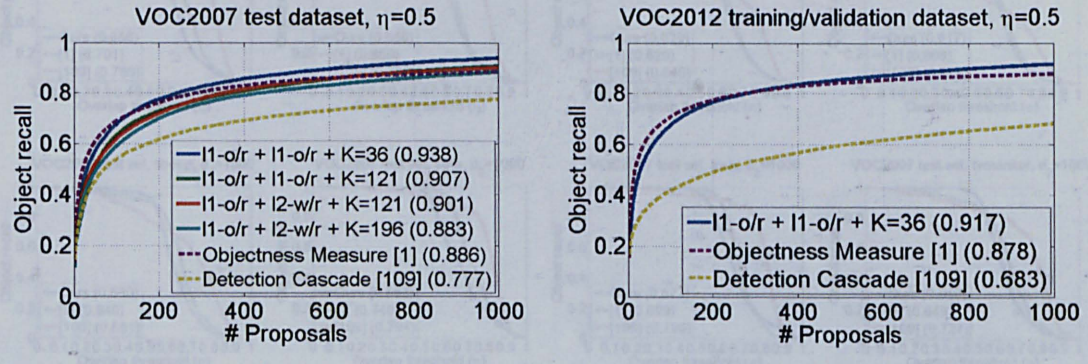


Figure 3.11: Comparison of recall-overlap curves using different methods and d_2 on (top) VOC2007 and (bottom) VOC2012. The numbers in the brackets are AUC values for the methods. From VOC2007 among the 4 cascade settings, our method with larger K seems to achieve better AUC using more proposals. Overall, on both datasets with the same number of proposals, each individual method has similar behaviors. In terms of object recall at $\eta = 0.5$, ours and [1] perform very similarly, and both outperform [109] in most cases. But in terms of quantity of proposals, [109] performs best.



(a) (b)

Figure 3.12: Comparison of recall-proposal curves using different methods on (a) VOC2007 and (b) VOC2012, respectively. The numbers in the brackets are the object recalls using 1000 proposals. In (a) among the 4 cascade settings, $\ell_1 - o/r + \ell_1 - o/r$ performs best. Still our method and [1] have similar behaviors on both datasets, and both outperform [109] significantly. Using 1000 proposals, our method outperforms [1, 109] by 5.2% and 16.1% on VOC2007, and 3.9% and 23.4% on VOC2012, respectively.

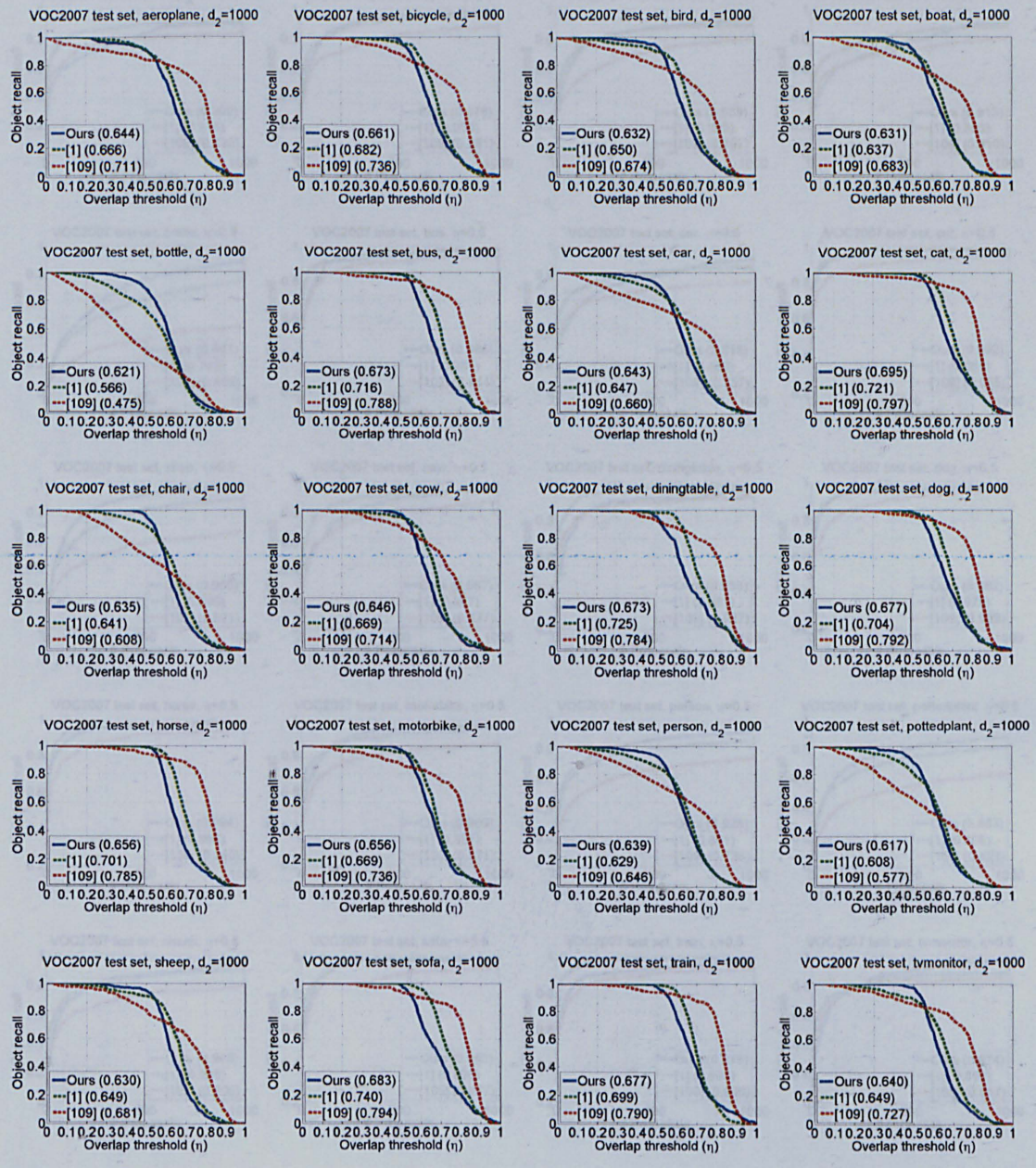


Figure 3.13: Comparison of recall-overlap curves using different methods on each class in the test dataset of VOC2007. The numbers in brackets are the AUC values for each method. In general, our method (*i.e.* $\ell_1 - o/r + \ell_1 - o/r$) performs similarly to [1], and when $\eta > 0.5$ [109] seems better than ours and [1] in terms of quantity of proposals. The mean and standard deviation of AUC's for our method, [1,109] are $(65.1 \pm 2.2)\%$, $(66.8 \pm 4.2)\%$, and $(70.8 \pm 8.3)\%$, respectively.

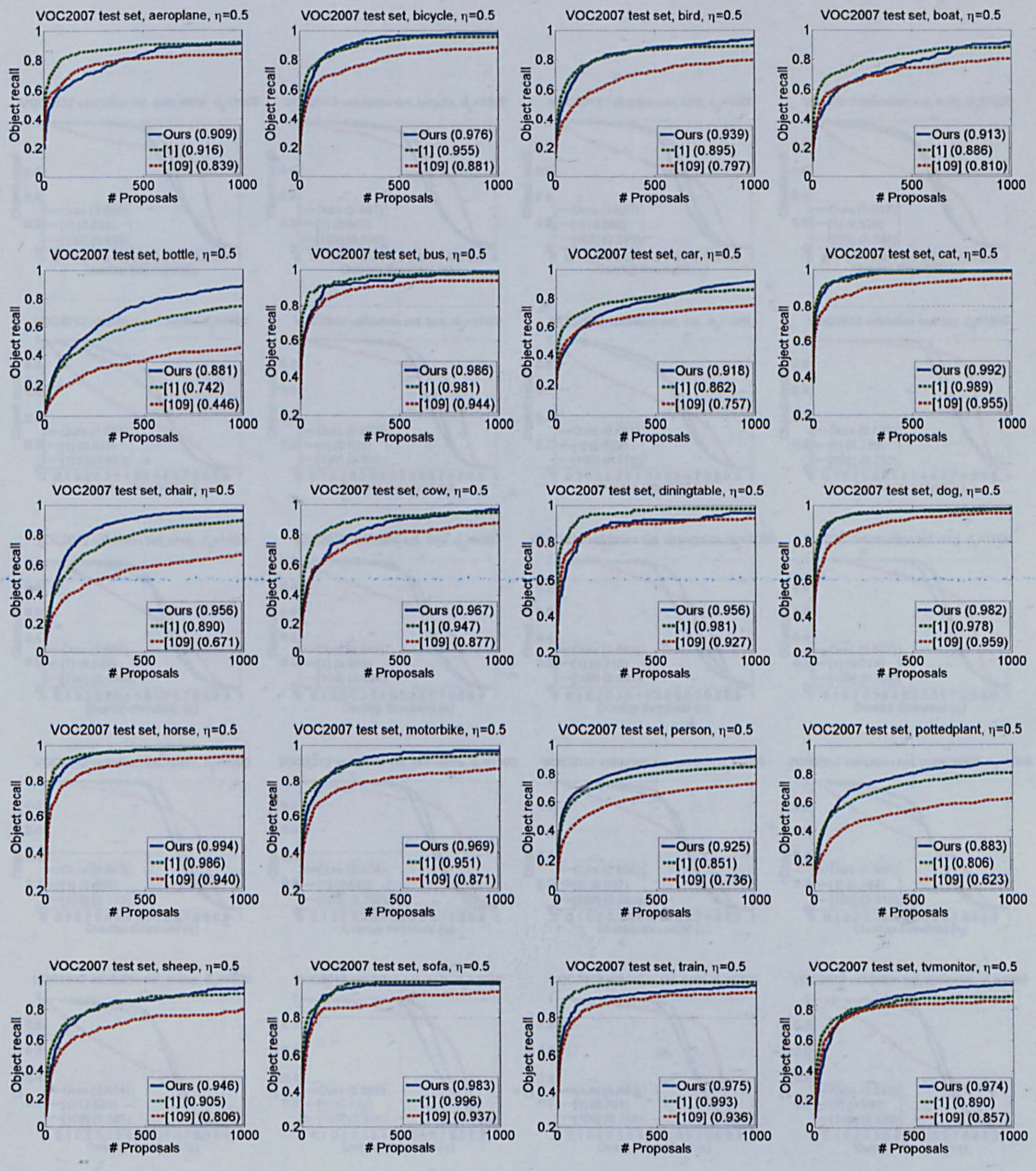


Figure 3.14: Comparison of recall-proposal curves using different methods and $\eta = 0.5$ on each class in the test dataset of VOC2007. The numbers in brackets are the object recall values for each method using 1000 proposals. Still, in general our method (*i.e.* $\ell_1 - o/r + \ell_1 - o/r$) and [1] have similar behaviors, and both outperform [109] using 1000 proposals. The mean and standard deviation of the object recall values for our method, [1, 109] are $(95.1 \pm 3.5)\%$, $(92.0 \pm 6.7)\%$, and $(82.8 \pm 12.8)\%$, respectively.

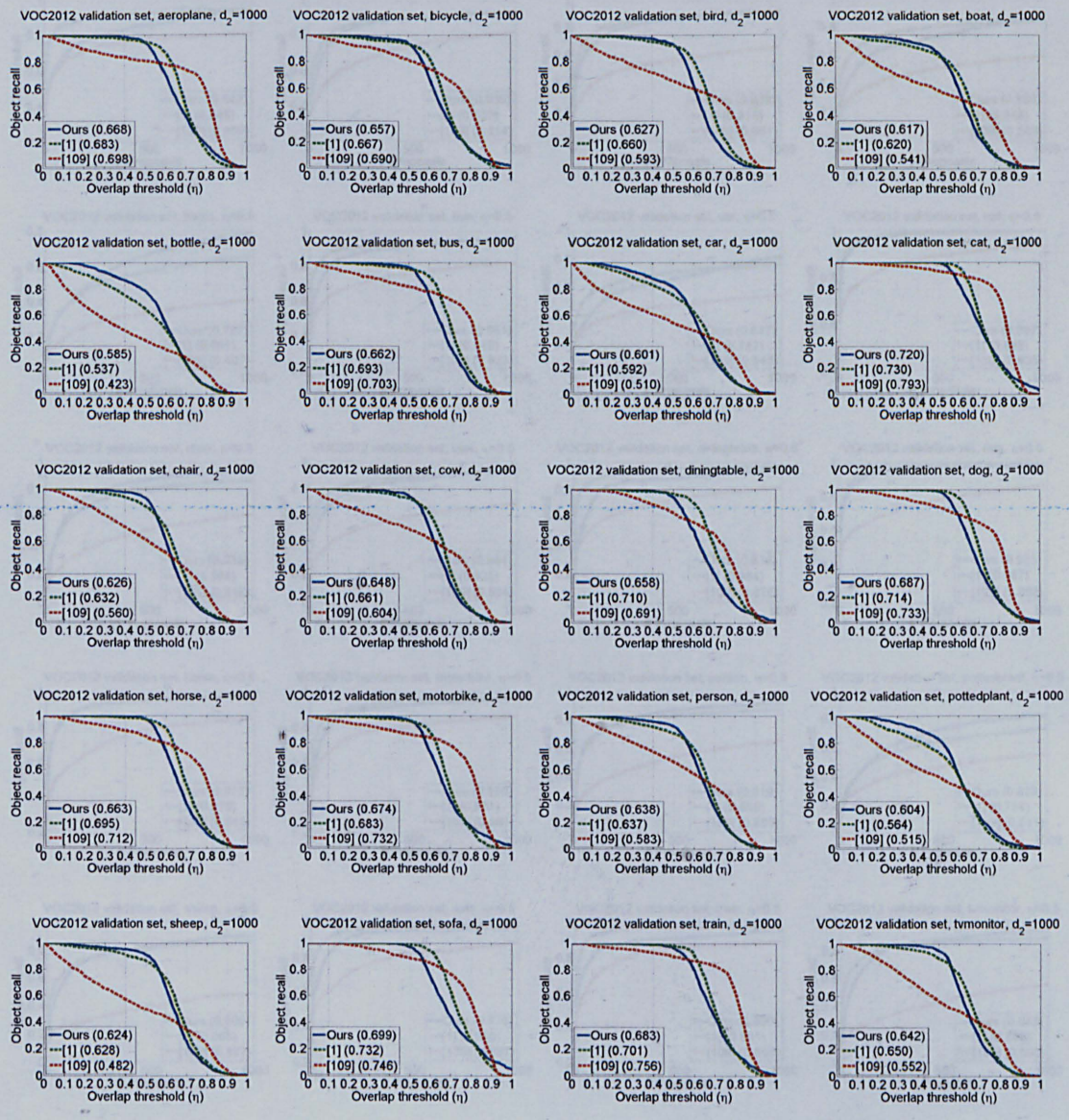


Figure 3.15: Comparison of recall-overlap curves using different methods on each class in the training/validation dataset of VOC2012. The numbers in brackets are the AUC values for each method. Similar observations can be made to those in Fig. 3.13. The mean and standard deviation of AUC's for our method, [1, 109] are $(64.9 \pm 3.4)\%$, $(65.9 \pm 5.2)\%$, and $(63.1 \pm 10.4)\%$, respectively.

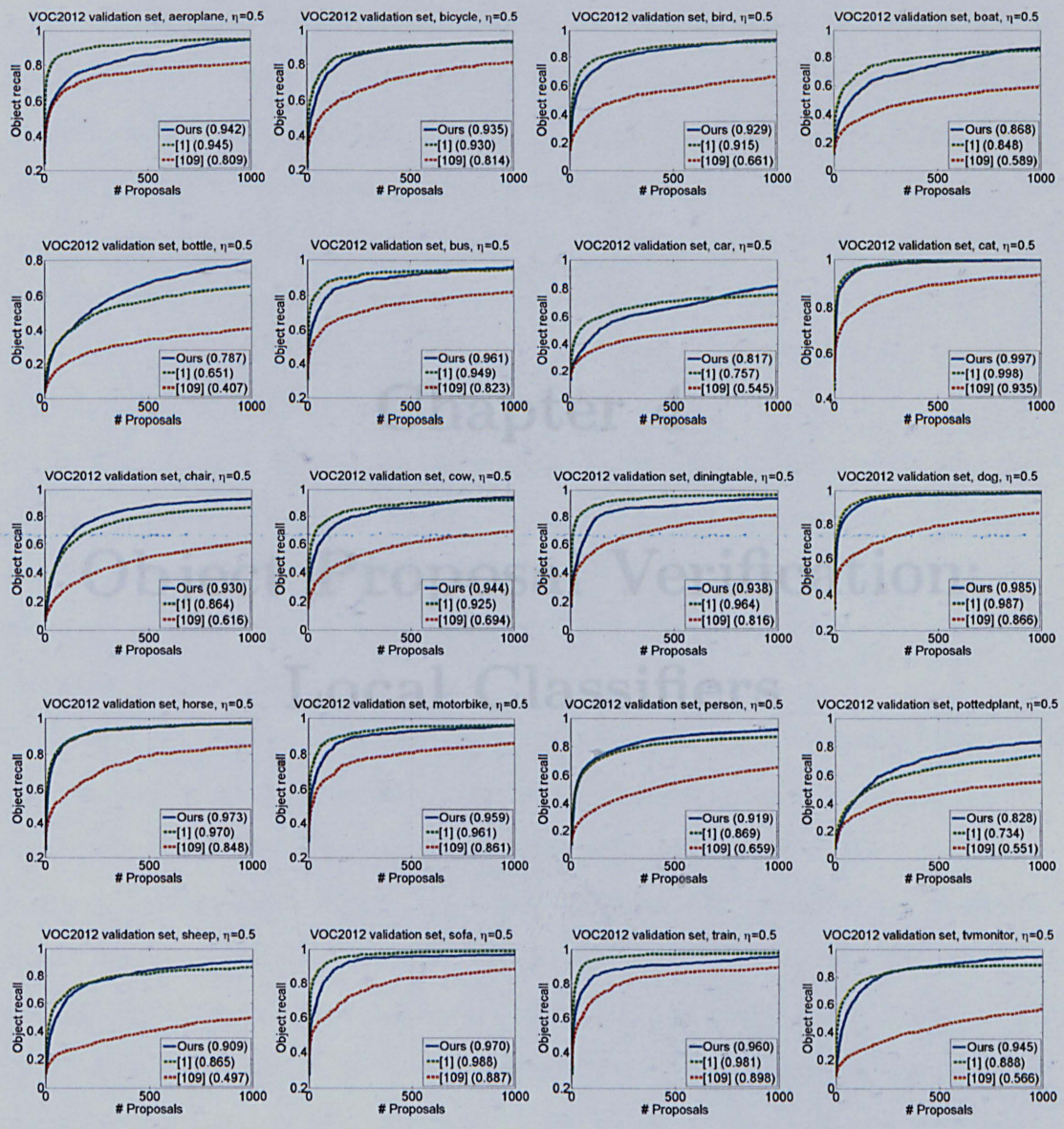


Figure 3.16: Comparison of recall-proposal curves using different methods and $\eta = 0.5$ on each class in the training/validation dataset of VOC2012. The numbers in brackets are the object recall values for each method using 1000 proposals. The mean and standard deviation of the object recall values for our method, $[1, 109]$ are $(92.5 \pm 5.6)\%$, $(89.9 \pm 9.1)\%$, and $(71.7 \pm 15.3)\%$, respectively. Similar observations can be made to those in Fig. 3.14.

Chapter 4

Object Proposal Verification: Local Classifiers

Local classifiers have attracted more and more attention recently [76,131,143,150]. The basic idea is to discover the data structures in low-dimensional manifolds, constructed by the local neighbors surrounding the data in the original feature space, where linear classifiers are trained. In general, due to the sparseness of features, these methods have low memory and storage requirement and fast speed of training and testing SVMs. However, such methods suffer from high computational complexity in learning sparse features in cases of using high dimensional data, large-scale data, or over-complete codebooks. For instance, sparse coding is widely used in such methods, which imposes some special regularizers (*e.g.* ℓ_1 norm [82], mixed norm [75], KL-divergence [20]) on the features in the optimization during both training and testing.

In this chapter, we propose three methods for efficient learning of local classifiers from locally linear to locally nonlinear. Each of our classifiers has similar computational complexity to that of linear SVMs, and comparable classification accuracy to kernel SVMs, which gives us a powerful tool for our efficient object detection framework.

Firstly, we propose an orthogonal coordinate coding (OCC) based locally linear classifier where each anchor point in the feature space is represented parametrically by an anchor plane so that every point in the feature space can be potentially localized by a few anchor planes. However, the orthogonality constraint limits the number of anchor planes, which reduces the discriminative power of the coding. By relaxing the orthogonality constraint, we propose our second locally linear classifier based on truncated marginal features (TMFs). In this classifier, a similar truncated function to that in OCC for encoding is utilized. In contrast though, the parameters of the affine transformation in the truncated function and the classifier are learned jointly using a biconvex formulation. Although locally linear classifiers are very powerful to approximate the decision boundaries among data, in some cases we may need local nonlinearity in the classifier so that the approximation could be done better. Therefore, we propose our third locally nonlinear classifier by parameterizing the traditional nearest neighbor (NN) classifier, which is actually a nonparametric locally nonlinear classifier. In this classifier, we learn to capture the prior knowledge of data distribution in each class, and

add this information into the NN classifier. We explain the details of our local classifiers in the following sections.

4.1 Learning Orthogonal Coordinate Coding

4.1.1 Introduction

Our orthogonal coordinate coding (OCC) is inspired by local coordinate coding (LCC) [143]. LCC is a coding scheme that encodes the data locally so that any nonlinear (α, β, p)-Lipschitz smooth function (see Definition 4.1 in Section 4.1.2 for details) over the data manifold can be approximated using linear functions. There are two components in LCC: (1) a set of anchor points which decide the local coordinates, and (2) the coding for each data based on the local coordinates given the anchor points. Theoretically [143] suggests that under certain assumptions, locality is more essential than sparsity for nonlinear function approximation. LCC has been successfully applied to many applications such as object recognition (*e.g.* locality-constraint linear coding (LLC) [131]) in the VOC 2009 challenge [44].

One big issue in LCC is that its classification performance is highly dependent on the number of anchor points, as observed in Yu and Zhang [142], because these points should be "local enough" to encode surrounding data on the data manifold accurately, which sometimes means that in real applications the number of anchor points explodes to a surprisingly huge number. This has been demonstrated in [143] where LCC has been tested on the MNIST dataset, using from 512 to 4096 anchor points learned from sparse coding, the error rate decreased from 2.64% to 1.90%. This situation could become a serious problem when the distribution of the data points is sparse in the feature space, *i.e.* there are many "holes" between data points (*e.g.* regions of feature space that are sparsely populated by data). As a result of this, many redundant anchor points will be distributed in the holes with little information. By using many anchor points, the computational complexity of the classifier at both training and test time increases significantly, defeating the original purpose of using LCC.

So far several approaches have been proposed for problems closely related to anchor point learning such as dictionary learning or codebook learning. For instance, Lee *et. al.* [83] proposed learning the anchor points for sparse coding with the ℓ_1 norm regularizer using the Lagrange dual, while Bradley and Bagnell [20] advocated KL-divergence as the regularizer. Mairal *et. al.* [93] proposed an online dictionary learning algorithm using stochastic approximations. Wang *et. al.* [131] proposed locality-constraint linear coding (LLC), which is a fast implementation of LCC, and an online incremental codebook learning algorithm using coordinate descent method, whose performance is very close to that using K-Means. However, none of these algorithms can deal with the holes of sparse data as they need many anchor points.

Alternatively, we propose a method to approximate any nonlinear (α, β, p)-Lipschitz smooth function using an orthogonal coordinate coding (OCC) scheme on a set of orthogonal basis vectors. Each basis vector $\mathbf{v} \in \mathbb{R}^d$ defines an **anchor plane** through the origin of the feature space, which can be considered as consisting of an infinite number of anchor points, and the nearest point on each anchor plane to a data point $\mathbf{x} \in \mathbb{R}^d$ is used for coding. The data point $\mathbf{x}$ will be encoded based on the margin, $\mathbf{x}^T\mathbf{v}$, where $(\cdot)^T$ denotes the matrix transpose operator, between $\mathbf{x}$ and an anchor plane defined by $\mathbf{v}$. Using anchor planes, many anchor points can be replaced by only a few anchor planes while preserving similar locality of anchor points. This sparsity may lead to a better generalization since many anchor points will overfit the data easily. Therefore, it can deal with the hole problem in LCC.

Meanwhile, the learned orthogonal basis vectors can fit naturally into locally linear SVMs (LL-SVMs) proposed by Ladicky and Torr [76]. They show how the functions defining the classifiers can be approximated using local codings and show how this model can be optimized in an online fashion by performing stochastic gradient descent with the same convergence guarantees as standard gradient descent method for linear SVMs. Mathematically LL-SVMs are formulated as

follows:

$$\begin{aligned} \arg\min_{\mathbf{W},\mathbf{b}} \ & \frac{\lambda}{2}\|\mathbf{W}\|_F^2 + \frac{1}{|\mathcal{S}|}\sum_{k\in\mathcal{S}}\xi_k \\ \text{s.t.} \ & \forall k \in \mathcal{S},\ y_k\left[\gamma_{x_k}^T\mathbf{W}\mathbf{x}_k + \gamma_{x_k}^T\mathbf{b}\right] \geq 1-\xi_k, \\ & \xi_k \geq 0, \end{aligned} \tag{4.1.1}$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix, $|\mathcal{S}|$ denotes the number of training data in the training set $\mathcal{S}$, $\forall k$, $\mathbf{x}_k \in \mathbb{R}^d$ is a training vector, $y_k \in \{-1, 1\}$ is its label, $\gamma_{x_k} \in \mathbb{R}^N$ is its local coding, $\lambda \geq 0$ is a pre-defined regularization parameter, and $\mathbf{W} \in \mathbb{R}^{N\times d}$ and $\mathbf{b} \in \mathbb{R}^N$ are the model parameters. As demonstrated in our experiments, the choices of the local coding methods are very important for LL-SVMs, and an improper choice will hurt its performance.

4.1.2 Orthogonal Coordinate Coding

For clarification, we summarize some notations in Table 4.1 which are used in LCC and OCC.

Table 4.1: Some notation used in LCC and OCC.

Notation	Definition
$\mathbf{v} \in \mathbb{R}^d$	An anchor point in LCC; a basis vector which defines an anchor plane through the origin of the feature space $\mathbb{R}^d$ in OCC.
$\mathcal{C} \subset \mathbb{R}^d$	A subset in the feature space containing all the anchor points (or basis vectors) ($\forall \mathbf{v}, \mathbf{v} \in \mathcal{C}$) in LCC (or OCC).
$\gamma_v(\mathbf{x}) \in \mathbb{R}$	The local coding of a data point $\mathbf{x} \in \mathbb{R}^d$ using the anchor point (or basis vector) $\mathbf{v}$.
$\gamma(\mathbf{x}) \in \mathbb{R}^d$	The physical approximation vector of a data point $\mathbf{x}$.
γ	A map of $\mathbf{x} \in \mathbb{R}^d$ to the coding vector.
$(\gamma, \mathcal{C})$	A coordinate coding.

4.1.2.1 Preliminary

We first recall some definitions and lemmas in LCC based on which we develop our method. Notice that in the following sections, $\|\cdot\|$ denotes the ℓ_2-norm without explicit explanation.

Definition 4.1 (Lipschitz Smoothness [143]). *A function $f(\mathbf{x})$ on $\mathbb{R}^d$ is (α, β, p)-Lipschitz smooth with respect to a norm $\|\cdot\|$ if $|f(\mathbf{x}') - f(\mathbf{x})| \leq \alpha\|\mathbf{x} - \mathbf{x}'\|$ and $|f(\mathbf{x}') - f(\mathbf{x}) - \nabla f(\mathbf{x})^T(\mathbf{x}' - \mathbf{x})| \leq \beta\|\mathbf{x} - \mathbf{x}'\|^{1+p}$, where we assume $\alpha, \beta > 0$ and $p \in (0, 1]$.*

Definition 4.2 (Coordinate Coding [143]). *A coordinate coding is a pair $(\gamma, \mathcal{C})$, where $\mathcal{C} \subset \mathbb{R}^d$ is a set of anchor points, and γ is a map of $\mathbf{x} \in \mathbb{R}^d$ to $[\gamma_v(\mathbf{x})]_{v \in \mathcal{C}} \in \mathbb{R}^{|\mathcal{C}|}$ such that $\sum_{\mathbf{v}} \gamma_v(\mathbf{x}) = 1$. It induces the following physical approximation of $\mathbf{x}$ in $\mathbb{R}^d$: $\gamma(\mathbf{x}) = \sum_{v \in \mathcal{C}} \gamma_v(\mathbf{x})\mathbf{v}$.*

Lemma 4.1 (Linearization [143]). *Let $(\gamma, \mathcal{C})$ be an arbitrary coordinate coding on $\mathbb{R}^d$. Let f be an (α, β, p)-Lipschitz smooth function. We have for all $\mathbf{x} \in \mathbb{R}^d$:*

$$\left| f(\mathbf{x}) - \sum_{v \in \mathcal{C}} \gamma_v(\mathbf{x}) f(\mathbf{v}) \right| \leq \alpha\|\mathbf{x} - \gamma(\mathbf{x})\| + \beta \sum_{v \in \mathcal{C}} |\gamma_v(\mathbf{x})| \, \|\mathbf{v} - \gamma(\mathbf{x})\|^{1+p} \quad (4.1.2)$$

As explained in [18], a good coding scheme for nonlinear function approximation should make $\mathbf{x}$ close to its physical approximation $\gamma(\mathbf{x})$ (*i.e.* smaller data reconstruction error $\|\mathbf{x} - \gamma(\mathbf{x})\|$) and should be localized (*i.e.* smaller localization error $\sum_{v \in \mathcal{C}} |\gamma_v(\mathbf{x})| \, \|\mathbf{v} - \gamma(\mathbf{x})\|^{1+p}$). This is the basic idea of LCC.

Definition 4.3 (Localization Measure [143]). *Given α, β, p, and coding $(\gamma, \mathcal{C})$, we define*

$$Q_{\alpha,\beta,p}(\gamma, \mathcal{C}) = \mathbb{E}_x \left[\alpha\|\mathbf{x} - \gamma(\mathbf{x})\| + \beta \sum_{v \in \mathcal{C}} |\gamma_v(\mathbf{x})| \, \|\mathbf{v} - \gamma(\mathbf{x})\|^{1+p} \right] \quad (4.1.3)$$

4.1.2.2 Orthogonal Coordinate Coding

We follow the notations in Table 4.1, and define our orthogonal coordinate coding (OCC) as below.

Definition 4.4 (Orthogonal Coordinate Coding). *An orthogonal coordinate coding is a pair $(\gamma, \mathcal{C})$, where $\mathcal{C} \subset \mathbb{R}^d$ contains $|\mathcal{C}|$ basis vectors and coding γ is a map of $\mathbf{x} \in \mathbb{R}^d$ to $[\gamma_v(\mathbf{x})]_{v \in \mathcal{C}} \in \mathbb{R}^{|\mathcal{C}|}$, so that a subset of orthogonal basis vectors $\mathcal{C}_\perp$ can be selected from $\mathcal{C}$ for encoding $\mathbf{x}$ and $\forall \mathbf{v} \in \mathcal{C}_\perp$, $\gamma_v(\mathbf{x}) \geq 0$, $\sum_{v \in \mathcal{C}_\perp} \gamma_v(\mathbf{x}) = 1$, $\forall \mathbf{v} \notin \mathcal{C}_\perp$, $\gamma_v(\mathbf{x}) = 0$.*

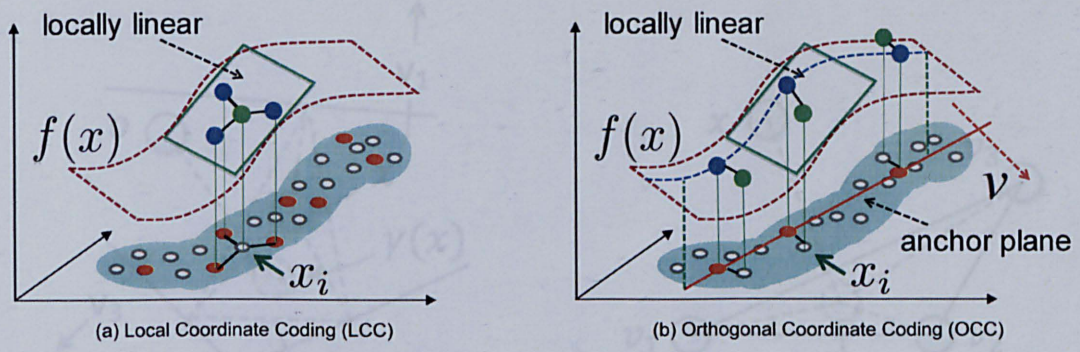


Figure 4.1: Comparison of the geometric views on (a) LCC and (b) OCC, where the white and red dots denote the data and anchor points, respectively. In LCC, the anchor points are distributed among the data space and several nearest neighbors around the data are selected for data reconstruction, while in OCC the anchor points are located on the anchor plane defined by the normal vector (*i.e.* coordinate, basis vector) $\mathbf{v}$ and only the closest point to each data point (*i.e.* the red dots) on the anchor plane is selected for coding. The figures are borrowed from the slides of [141].

Compared to Definition 4.2, we can see that OCC is a special case of coordinate coding where the selected coordinates are orthogonal. Figure 4.1 illustrates the geometric views on LCC and OCC respectively. Intuitively, in both methods anchor points try to encode data locally. However, the ways of their arrangement are quite different. In LCC anchor points are distributed among the whole data space so that each data can be covered by certain anchor points in a local region, and their distribution cannot be described using regular shapes. On the contrary, anchor points in OCC are located on anchor planes defined by basis vectors. In fact, each anchor plane can be considered as an infinite number of anchor points, and for each data point only its closest point on each anchor plane is utilized for reconstruction and localization. Therefore, intuitively the number of anchor planes in OCC should be much smaller than the number of anchor points in LCC. Since we do not define the anchor points explicitly in the feature space, any point could be encoded using OCC potentially, which makes this method handle the "hole" problem seamlessly.

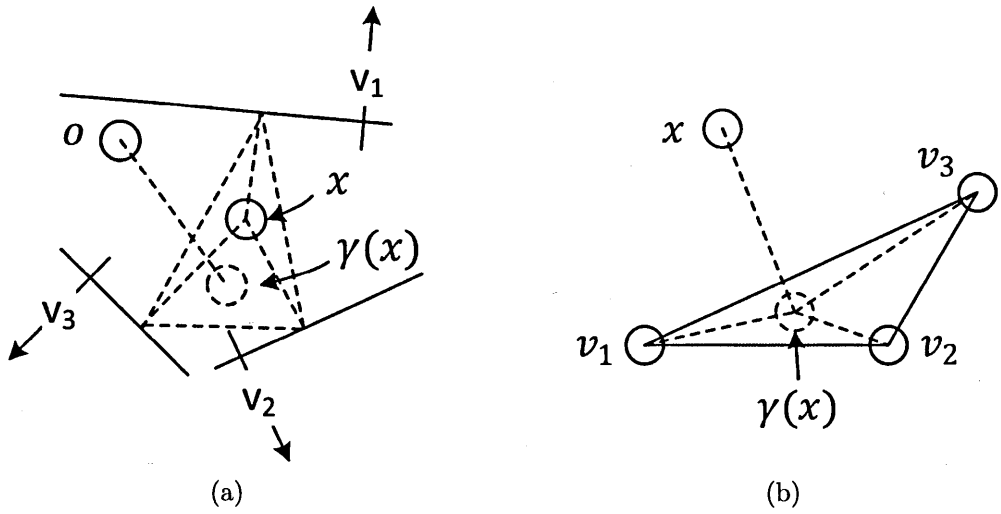


Figure 4.2: Illustration of learning OCC using (a) the closest point to the data on each anchor plane, or equivalently (b) basis vectors as anchor points in the feature space. Here $\mathbf{v}_1$, $\mathbf{v}_2$, and $\mathbf{v}_3$ denote the orthogonal basis vectors for defining anchor planes, $\mathbf{x}$ denotes a data point, $\gamma(\mathbf{x})$ denotes the physical approximation of $\mathbf{x}$, and $\mathbf{o}$ denotes the origin of the feature space.

4.1.2.3 Learning and Encoding

Instead of optimizing Definition 4.3, LCC simplifies the localization error term by assuming $\gamma(\mathbf{x}) = \mathbf{x}$ and $p = 1$. Mathematically LCC solves the following optimization problem:

$$\min_{(\gamma,\mathcal{C})} \sum_{\mathbf{x}\in\mathcal{X}} \left\{ \frac{1}{2}\|\mathbf{x}-\gamma(\mathbf{x})\|^2 + \mu \sum_{\mathbf{v}\in\mathcal{C}} |\gamma_v(\mathbf{x})| \|\mathbf{v}-\mathbf{x}\|^2 + \lambda \sum_{\mathbf{v}\in\mathcal{C}} \|\mathbf{v}\|^2 \right\} \tag{4.1.4}$$
$$\text{s.t.} \quad \forall \mathbf{x}, \quad \textstyle\sum_{\mathbf{v}\in\mathcal{C}} \gamma_v(\mathbf{x}) = 1.$$

They update $\mathcal{C}$ and γ via alternating optimization. The step of updating γ can be transformed into a canonical LASSO problem, and the step of updating $\mathcal{C}$ is a least squares problem.

Differently, in OCC we would like to minimize the total square Euclidean distances between the reconstructed data points and the origin, as illustrated in Fig. 4.2(a). Given a data point $\mathbf{x}$ and a orthogonal basis vector set $\mathcal{C}$, the data point is encoded using the closest points on the anchor planes, that is $\forall \mathbf{v} \in \mathcal{C}, \mathbf{x} - \frac{\mathbf{x}^T\mathbf{v}}{\|\mathbf{v}\|^2}\mathbf{v}$. Notice that we restrict the reconstructed point in a simplex constructed by the selected closest points. Intuitively the formulation for learning OCC can

be shown as follows:

$$\min_{\mathbf{z},\boldsymbol{\theta},\mathcal{C}} \sum_{\mathbf{x}\in\mathcal{X}} \left\| \sum_{\mathbf{v}\in\mathcal{C}} \left(\mathbf{x} - \frac{\mathbf{x}^T\mathbf{v}}{\|\mathbf{v}\|^2}\mathbf{v} \right) z_{x,v}\theta_{x,v} \right\|^2 + \lambda|\mathcal{C}| \tag{4.1.5}$$
$$\text{s.t.} \quad \forall \mathbf{x} \in \mathcal{X}, \mathbf{u}, \mathbf{v} \in \mathcal{C},\ \mathbf{u} \neq \mathbf{v} \Rightarrow z_{x,u} z_{x,v} \mathbf{u}^T\mathbf{v} = 0,$$
$$z_{x,v} \in \{0,1\},\ \textstyle\sum_{\mathbf{v}\in\mathcal{C}} z_{x,v} = M,$$
$$\theta_{x,v} \geq 0,\ \textstyle\sum_{\mathbf{v}\in\mathcal{C}} z_{x,v}\theta_{x,v} = 1,$$

where $|\mathcal{C}|$ denotes the number of basis vectors, $\mathbf{z} \subset \mathbb{R}^{|\mathcal{X}|\times|\mathcal{C}|}$ denotes a binary matrix indicating which orthogonal basis vectors are selected to encode a data point, $\boldsymbol{\theta} \subset \mathbb{R}^{|\mathcal{X}|\times|\mathcal{C}|}$ denotes another matrix indicating the weight associated with each closest point for constructing the simplex, $M \geq 0$ is a predefined constant controlling the number of orthogonal basis vectors that are used for coding, and $\lambda \geq 0$ controls the trade-off between the total distance and the number of basis vectors.

Since the term $\frac{\mathbf{x}^T\mathbf{v}}{\|\mathbf{v}\|^2}\mathbf{v}z_{x,v}\theta_{x,v}$ in Eq. 4.1.5 has a product of three unknown variables, we introduce a single coordinate coding variable $\gamma_v(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{v}}{\|\mathbf{v}\|^2}\theta_{x,v}$. Then Eq. 4.1.5 can be rewritten as follows, as illustrated in Fig. 4.2(b):

$$\min_{\mathbf{z},(\boldsymbol{\gamma},\mathcal{C})} \sum_{\mathbf{x}\in\mathcal{X}} \left\| \mathbf{x} - \sum_{\mathbf{v}\in\mathcal{C}} \mathbf{v} z_{x,v}\gamma_v(\mathbf{x}) \right\|^2 + \lambda|\mathcal{C}| \tag{4.1.6}$$
$$\text{s.t.} \quad \forall \mathbf{x} \in \mathcal{X}, \mathbf{u}, \mathbf{v} \in \mathcal{C},\ \mathbf{u} \neq \mathbf{v} \Rightarrow z_{x,u} z_{x,v} \mathbf{u}^T\mathbf{v} = 0,$$
$$z_{x,v} \in \{0,1\},\ \textstyle\sum_{\mathbf{v}\in\mathcal{C}} z_{x,v} = M,$$
$$\gamma_v(\mathbf{x}) \geq 0,\ \textstyle\sum_{\mathbf{v}\in\mathcal{C}} z_{x,v}\gamma_v(\mathbf{x}) = 1,$$

Proposition 4.1 (Minimum Number of Basis Vectors). *Assuming that data points are distributed uniformly in the feature space, in order to satisfy the constraints in Eq. 4.1.6 for any arbitrary data point* $\mathbf{x}$ *using orthogonal basis vectors, the number of basis vectors in* $\mathcal{C}$ *should satisfy* $|\mathcal{C}| \geq 2M$.

Proof. Under the assumption, for any hyper-plane defined by $\mathbf{v}$, which goes through the origin of the feature space, the probability of $\mathbf{x}^T\mathbf{v} \geq 0$ is equal to 0.5. Letting $\mathcal{C}_\perp$ contain M orthogonal basis vectors, and letting $\mathcal{C} = \mathcal{C}_\perp \bigcup \{-\mathcal{C}_\perp\}$ where $\forall \mathbf{v} \in \mathcal{C}_\perp$, $-\mathbf{v} \in \{-\mathcal{C}_\perp\}$, and $\bigcup$ denotes the set union operator, then each data point can be encoded by the orthogonal basis vectors either from $\mathcal{C}_\perp$ or from

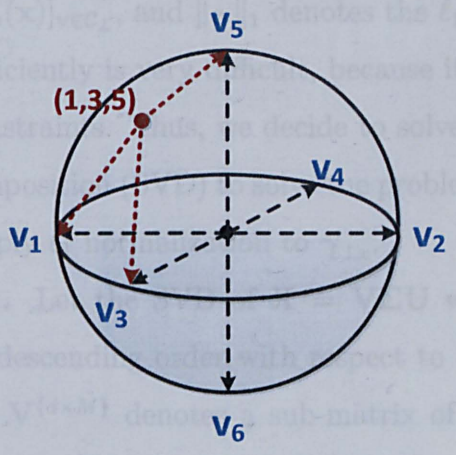


Figure 4.3: Example of encoding a data point in a 3D space using 6 anchor points, denoted by the blue color, where $\mathbf{v}_1 = -\mathbf{v}_2$, $\mathbf{v}_3 = -\mathbf{v}_4$, $\mathbf{v}_5 = -\mathbf{v}_6$, and $\mathbf{v}_1 \perp \mathbf{v}_3 \perp \mathbf{v}_5$. The given data, denoted by the red dot, is encoded by $\mathbf{v}_1$, $\mathbf{v}_3$, and $\mathbf{v}_5$, respectively, since their codes are positive, and the codes using the rest anchor points are negative.

$-\mathcal{C}_\perp$ to satisfy the constraints in Eq. 4.1.6. Therefore, the minimum number of basis vectors in $\mathcal{C}$ is $2M$. □

Fig. 4.3 gives an example of using 6 anchor points to encode a given data point in a 3D feature space, where $\mathcal{C} = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5, \mathbf{v}_6\} = \{\pm\mathbf{v}_1, \pm\mathbf{v}_3, \pm\mathbf{v}_5\}$. An arbitrary data point in this 3D feature space can be encoded by $\mathcal{C}$ so that the constraints in Eq. 4.1.6 can be satisfied for OCC.

We assume that in Eq. 4.1.6 $\lambda|\mathcal{C}|$ will dominate the objective function, and thus $|\mathcal{C}|$ should take its minimum value, as stated in Proposition 4.1. By constructing $\mathcal{C} = \mathcal{C}_\perp \bigcup \{-\mathcal{C}_\perp\}$, we can see that the functionality of $\mathbf{z}$ is a sign indicator. Therefore, again we can rewrite Eq. 4.1.6 as follows:

$$\begin{aligned} \min_{(\gamma_\perp, \mathcal{C}_\perp)} & \sum_{\mathbf{x}\in\mathcal{X}} \|\mathbf{x} - \mathbf{C}_\perp \gamma_{\perp x}\|^2 \\ \text{s.t.} \quad & \forall \mathbf{u}, \mathbf{v} \in \mathcal{C}_\perp,\ \mathbf{u} \neq \mathbf{v} \Rightarrow \mathbf{u}^T\mathbf{v} = 0, \\ & |\mathcal{C}_\perp| = M, \\ & \forall \mathbf{x},\ \|\gamma_{\perp x}\|_1 = 1, \end{aligned} \tag{4.1.7}$$

where $\mathbf{C}_\perp \in \mathbb{R}^{d\times|\mathcal{C}_\perp|}$ denotes the orthogonal basis vector matrix with $\mathbf{v} \in \mathcal{C}_\perp$ as columns, $\gamma_{\perp x} \in \mathbb{R}^{|\mathcal{C}_\perp|}$ denotes the coding vector of data point $\mathbf{x}$ containing all

$\gamma_v(\mathbf{x})$ in order $\boldsymbol{\gamma}_{\perp x} = [\gamma_v(\mathbf{x})]_{\mathbf{v}\in\mathcal{C}_\perp}$, and $\|\cdot\|_1$ denotes the ℓ_1 norm of a vector.

Solving Eq. 4.1.7 efficiently is very difficult, because it involves both orthogonality and ℓ_1 norm constraints. Thus, we decide to solve it approximately. We use singular value decomposition (SVD) to solve the problem without the ℓ_1 norm constraint first, then apply ℓ_1 normalization to $\boldsymbol{\gamma}_{\perp x}$.

(I) Solving for $\mathcal{C}_\perp$. Let the SVD of $\mathbf{X} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}$ where the singular values are positive and in descending order with respect to $\boldsymbol{\Sigma}$. Then we set $\mathbf{C}_\perp = \mathbf{V}^{\{d\times M\}}\boldsymbol{\Sigma}^{\{M\times M\}}$, where $\mathbf{V}^{\{d\times M\}}$ denotes a sub-matrix of $\mathbf{V}$ containing the elements within rows from 1 to d and columns from 1 to M, similarly for $\boldsymbol{\Sigma}^{\{M\times M\}}$. We need only to use a few top eigenvectors as our orthogonal basis vectors for coding, and the search space is far smaller than generating anchor points.

Notice that SVD can be applied in two different ways: (1) directly to the entire training data matrix, or (2) separately to the data within each category. We denote these two types of OCC as ***G-OCC*** (i.e. Generic OCC) and ***C-OCC*** (i.e. Class-specific OCC), respectively.

(II) Solving for $\boldsymbol{\gamma}_{\perp x}$. Since we have the orthogonal basis vectors in $\mathcal{C}_\perp$, we can easily derive the formulation for calculating $\tilde{\boldsymbol{\gamma}}_{\perp x}$, the values of $\boldsymbol{\gamma}_{\perp x}$ before normalization, that is, $\tilde{\boldsymbol{\gamma}}_{\perp x} = (\mathbf{C}_\perp^T\mathbf{C}_\perp)^{-1}\mathbf{C}_\perp^T\mathbf{x}$. Specifically, $\forall \mathbf{v} \in \mathcal{C}_\perp$, $\tilde{\gamma}_v(\mathbf{x}) = \frac{\mathbf{v}^T\mathbf{x}}{\|\mathbf{v}\|^2}$. Finally, we can calculate $\boldsymbol{\gamma}_{\perp x}$ by normalizing $\tilde{\boldsymbol{\gamma}}_{\perp x}$ using ℓ_1 norm.

Similarly, after constructing $\mathcal{C} = \mathcal{C}_\perp \bigcup \{-\mathcal{C}_\perp\}$, we can easily encode the data using Eq. 4.1.8, followed by ℓ_1 normalization:

$$\forall \mathbf{v} \in \mathcal{C}, \mathbf{x},\ \gamma_v(\mathbf{x}) \propto \max\left\{\frac{\mathbf{v}^T\mathbf{x}}{\|\mathbf{v}\|^2}, 0\right\}. \tag{4.1.8}$$

In our experimental section, we simply take the codes in Eq. 4.1.7 as our OCC by assuming that the weights learned latter in the classifiers are the same for the counterparts (*e.g.* $\mathbf{v}$ and $-\mathbf{v}$) in the original set of basis vectors $\mathcal{C}$.

4.1.3 Modeling Classification Decision Boundary in Locally Linear SVMs

Given a set of data $\{(\mathbf{x}_i, y_i)\}$ where $y_i \in \{-1, 1\}$ is the label of $\mathbf{x}_i$, the decision boundary for a binary linear SVM is $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ where $\mathbf{w}$ is the SVM coefficients defining the decision hyperplane and b is a bias term. Here, we assume that the decision boundary is an (α, β, p)-Lipschitz smooth function. Since in LCC each data is encoded by some anchor points on the data manifold, it can model the decision boundary of an SVM directly using $f(\mathbf{x}) \approx \sum_{\mathbf{v}\in\mathcal{C}} \gamma_v(\mathbf{x}) f(\mathbf{v})$. Then by taking γ_x as the input data of a linear SVM, $f(\mathbf{v})$'s can be learned to approximate the decision boundary f.

However, OCC learns a set of orthogonal basis vectors, which define anchor planes rather than anchor points, and are used for coding. This makes OCC suitable to model the decision hyperplanes with LL-SVMs. Given data $\mathbf{x}$ and an OCC $(\gamma, \mathcal{C})$, the decision boundary in LL-SVMs can be formulated as follows [1].

$$f(\mathbf{x}) = \mathbf{w}(\mathbf{x})^T\mathbf{x} + b = \sum_{\mathbf{v}\in\mathcal{C}} \gamma_v(\mathbf{x})\mathbf{w}(\mathbf{v})^T\mathbf{x} + b = \gamma_x^T\mathbf{W}\mathbf{x} + b \tag{4.1.9}$$

where $\mathbf{W} \in \mathbb{R}^{M\times d}$ is a matrix which needs to be learned for LL-SVMs. In the view of kernel SVMs, we actually define another kernel $\mathbf{K}$ based on $\mathbf{x}$ and γ_x as shown below,

$$\forall i, j,\ \mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) =< \gamma_{x_i}\mathbf{x}_i^T, \gamma_{x_j}\mathbf{x}_j^T > \tag{4.1.10}$$

where $< \cdot, \cdot >$ denotes the Frobenius inner product.

Then given a test data point $\mathbf{x} \in \mathbb{R}^d$, its class label is assigned by

$$y(\mathbf{x}) = \arg\max_y \gamma_{x,y}^T\mathbf{W}_y\mathbf{x} + b_y, \tag{4.1.11}$$

where $\gamma_{x,y}$ denotes the code for $\mathbf{x}$ using C-OCC, while using G-OCC, $\gamma_{x,y} = \gamma_x$.

[1] Notice that Eq. 4.1.9 is slightly different from the original formulation in [76] by ignoring the different bias term for each orthogonal basis vector.

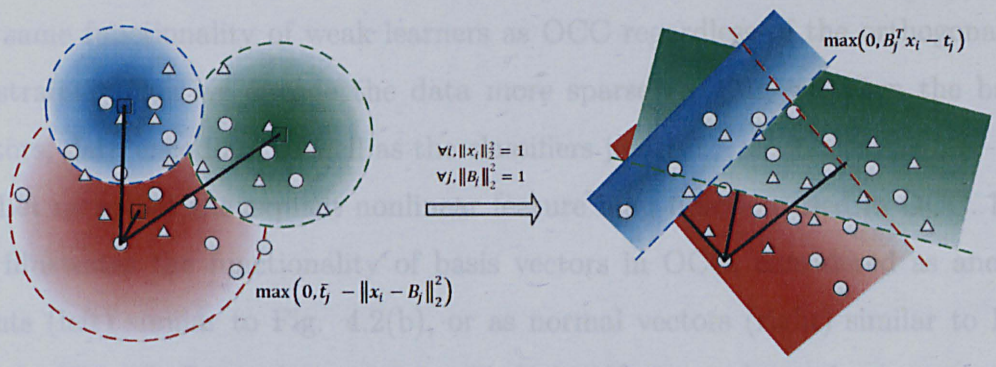


Figure 4.4: Illustration of our truncated marginal features (TMFs) for learning locally linear classifiers. In both figures, circle and triangle denote 2 classes, different color regions denote the weight regions, and the darker the color is, the higher the weight is. On the left, rectangles denote anchor points and the dashed circles denote the boundaries of their local regions, outside which weights are zeros and inside which weights decrease linearly with square Euclidean distances between data and anchor points. Assuming data and anchor points are localized on the unit hypersphere, the anchor points with their local regions in the left figure can be represented by the hyperplanes, denoted by the dashed lines, with bias terms in the right figure. Accordingly, weights in the left figure (*i.e.* solid lines) can be represented as margins in the right figure (*i.e.* solid lines).

4.2 Learning Locally Linear Classifiers via Truncated Marginal Features

4.2.1 Introduction

As we see from Eq. 4.1.8, in OCC each anchor plane splits the entire feature space into two parts, where all the data points above the plane have the positive codes, indicating how far the data points are from the anchor plane, otherwise 0. This coding process defines an explicit nonlinear feature map function, and the functionality of anchor planes is actually performed as weak learners which collect the geometric information of data as the input for classifiers. However, the orthogonality constraint limits the number of learned basis vectors and non-zero elements in the coding, and the learned codes from OCC may not be optimal for the classification purpose because the coding process is independent of learning classifiers.

Then the questions come out: Can we learn more basis vectors to perform

the same functionality of weak learners as OCC regardless of the orthogonality constraint? Can we encode the data more sparsely? Can we learn the basis vectors, data encoding, as well as the classifiers jointly?

Let us revisit the explicit nonlinear feature map function used in OCC. Fig. 4.4 illustrates the functionality of basis vectors in OCC, considered as anchor points (left) similar to Fig. 4.2(b), or as normal vectors (right) similar to Fig. 4.2(a). The left figure shows a very simple localization scheme for constructing local classifiers, where each data point can be represented based on the squared Euclidean distances between it and the anchor points. Particularly, when the data and anchor points are localized on the unit hypersphere, this distance-based representation can be rewritten as the truncated margin representation in the right figure, which is used in Eq. 4.1.8. We call the features in the right figure *truncated marginal features* (TMFs).

To learn our locally linear classifier as well as TMFs jointly, we formulate it as a biconvex minimization problem, and solve it efficiently using Adaptive Convex Search (ACS), where the explicit nonlinear map function is learned using random projection and stochastic sub-gradient descent, and the classifiers are learned using multiclass linear SVM solvers. We further prove that our locally linear classifiers with TMFs can be used to approximate the decision boundaries of kernel SVMs using the summation of multiple nonlinear arc-cosine kernels [25] with equal weights.

4.2.2 Joint Learning of Classifiers and Features

4.2.2.1 Preliminaries

Definition 4.5 (Truncated Functions). *Given a data point $x \in \mathbb{R}$ and a threshold $t \in \mathbb{R}$, we define a truncated function ψ as follows:*

$$\psi(x;t) = \max(x - t, 0). \tag{4.2.1}$$

Definition 4.6 (Truncated Marginal Features (TMFs)). *We define the*

TMF of a data point $\mathbf{x} \in \mathbb{R}^d$, $\phi(\mathbf{x}; \mathbf{B}, \mathbf{t})$, *as follows:*

$$\begin{aligned} \phi(\mathbf{x}; \mathbf{B}, \mathbf{t}) &= \max\left(\mathbf{B}^T\mathbf{x} - \mathbf{t}, \mathbf{0}\right) \\ \text{s.t.} \quad \mathbf{t}_{lb} &\preceq \mathbf{t} \preceq \mathbf{t}_{ub}, \ \forall i, \ \|\mathbf{B}_i\|_2^2 \leq 1, \end{aligned} \tag{4.2.2}$$

where $\boldsymbol{B} \in \mathbb{R}^{d \times D}$ *is the transformation matrix,* $\boldsymbol{B}_i \in \mathbb{R}^d$ *is the* i^{th} *column in* $\boldsymbol{B}$, $\mathbf{t} \in \mathbb{R}^D$ *is a threshold vector bounded by the predefined lower bound* $\mathbf{t}_{lb} \in \mathbb{R}^D$ *and predefined upper bound* $\mathbf{t}_{ub} \in \mathbb{R}^D$, **max** *and* $\preceq$ *are the element-wise operators of* max *and* $\leq$, *respectively, and* $\|\cdot\|_2$ *is the* ℓ_2 *norm.*

Remarks: In our TMFs, the bound constraint on the threshold vector $\mathbf{t}$ helps to control the sparseness of TMFs, because the lower and upper bounds can be set manually and data-dependently. The other constraint on the norm of each column in $\mathbf{B}$ guarantees that the TMFs are scale invariant. Note that our TMFs will be equivalent to a single-layer threshold network [25] when $\mathbf{t} = 0$.

Based on the definition of TMFs, we address our learning task as below.

Problem. *Given a set of training data* $(\mathbf{x}_i, y_i)_{i=1,\cdots,N}$ *where* $\forall i, \mathbf{x}_i \in \mathbb{R}^d$ *is a feature vector and* $y_i \in \{1, \cdots, C\} (C \in \mathbb{N})$ *is its class label, we would like to learn a multiclass classifier, so that*

$$\begin{aligned} &\min_{\mathbf{B},\mathbf{t},\mathbf{W},\mathbf{b}} \left\{ \frac{1}{2}\sum_{c=1}^{C}\|\mathbf{w}_c\|_2^2 + \frac{\theta}{2}\|\mathbf{t}\|_2^2 + \eta\sum_{i=1}^{N}\sum_{c=1}^{C}\max\left(1 - \mathbb{I}_{y_i,c}\left(\mathbf{w}_c^T\phi(\mathbf{x}_i; \mathbf{B}, \mathbf{t}) + b_c\right), 0\right) \right\} \\ &\text{s.t.} \quad \mathbf{t}_{lb} \preceq \mathbf{t} \preceq \mathbf{t}_{ub}, \ \forall j, \ \|\mathbf{B}_j\|_2^2 \leq 1, \end{aligned} \tag{4.2.3}$$

where $(\mathbf{B}, \mathbf{t})$ *is the TMF parameters shared by all the classes,* $(\mathbf{w}_c, b_c)$ *are the classifier parameters for class* $c \in \{1, \cdots, C\}$ *(*$\mathbf{w}_c \in \mathbb{R}^D$ *is the* c^{th} *column in* $\mathbf{W} \in \mathbb{R}^{D \times C}$, *and* $b_c \in \mathbb{R}$ *is the* c^{th} *element in* $\mathbf{b} \in \mathbb{R}^C$*),* $\theta \geq 0$ *and* $\eta \geq 0$ *are predefined regularizer parameters,* $\mathbb{I}_{y_i,c}$ *is an indicator function and* $\mathbb{I}_{y_i,c} = 1$ *if* $y_i = c$, *otherwise 0.*

The class label of a test data point $\hat{\mathbf{x}}$ is predicted as

$$y_{\hat{\mathbf{x}}} = \arg\max_c f_c(\hat{\mathbf{x}}; \mathbf{B}, \mathbf{t}, \mathbf{w}_c, b_c) = \arg\max_c \left\{\mathbf{w}_c^T\phi(\hat{\mathbf{x}}; \mathbf{B}, \mathbf{t}) + b_c\right\}, \tag{4.2.4}$$

where $f_c(\hat{\mathbf{x}}; \mathbf{B}, \mathbf{t}, \mathbf{w}_c, b_c)$ denotes the decision function for class c.

4.2.2.2 Analysis

Eq. 4.2.3 is equivalent to inserting TMFs in the formulation of multiclass linear SVMs. However, this problem is never convex in general, but biconvex.

Definition 4.7 (Biconvex functions [57]). *Let $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^m$ be two non-empty, convex sets, and let f be a real-valued function on $\mathcal{X} \times \mathcal{Y}$. f is biconvex if and only if for all quadruples $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_1, \mathbf{y}_2), (\mathbf{x}_2, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2) \in \mathcal{X} \times \mathcal{Y}$ it holds, that for every $(\lambda, \mu) \in [0,1] \times [0,1]$,*

$$\begin{aligned} f(\mathbf{x}_\lambda, \mathbf{y}_\mu) \leq (1-\lambda)(1-\mu) f(\mathbf{x}_1, \mathbf{y}_1) + (1-\lambda)\mu f(\mathbf{x}_1, \mathbf{y}_2) \\ + \lambda(1-\mu) f(\mathbf{x}_2, \mathbf{y}_1) + \lambda\mu f(\mathbf{x}_2, \mathbf{y}_2), \end{aligned} \tag{4.2.5}$$

where $(\mathbf{x}_\lambda, \mathbf{y}_\mu) := ((1-\lambda)\mathbf{x}_1 + \lambda\mathbf{x}_2, (1-\mu)\mathbf{y}_1 + \mu\mathbf{y}_2)$.

Theorem 4.1. *The minimization problem in Eq. 4.2.3 is a biconvex minimization problem.*

Proof. By introducing slack variables into Eq. 4.2.3, it is shown that solving Eq. 4.2.3 is equivalent to solving the following biconvex minimization problem:

$$\begin{aligned} \min_{\mathbf{B},\mathbf{t},\mathbf{W},\mathbf{b},\boldsymbol{\xi},\boldsymbol{\rho}} \quad & \frac{1}{2}\sum_c \|\mathbf{w}_c\|_2^2 + \frac{\theta}{2}\|\mathbf{t}\|_2^2 + \eta \sum_{i,c} \xi_{i,c} + \tau \sum_i \|\boldsymbol{\rho}_i\|_1 \\ \text{s.t.} \quad & \forall i, c,\ \mathbb{I}_{y_i,c}\left[\mathbf{w}_c^T\left(\mathbf{B}^T\mathbf{x}_i - \mathbf{t} + \boldsymbol{\rho}_i\right) + b_c\right] \geq 1 - \xi_{i,c}, \\ & \forall i, c,\ \xi_{i,c} \geq 0, \\ & \forall i,\ \mathbf{B}^T\mathbf{x}_i - \mathbf{t} + \boldsymbol{\rho}_i \succeq 0,\ \boldsymbol{\rho}_i \succeq 0,\ \mathbf{t}_{lb} \preceq \mathbf{t} \preceq \mathbf{t}_{ub}, \\ & \forall j,\ \|\mathbf{B}_j\|_2^2 \leq 1, \end{aligned} \tag{4.2.6}$$

where $\boldsymbol{\xi}$ and $\boldsymbol{\rho}$ are slack variables, $\forall i, \boldsymbol{\rho}_i \in \mathbb{R}^D$ is a vector, $\|\cdot\|_1$ is the ℓ_1 norm, $\succeq$ is the element-wise operator of $\geq$, and $\tau (0 \leq \tau \ll \theta)$ is a predefined constant. From the first constraint in Eq. 4.2.6, we can see that this optimization problem is biconvex. □

We can adopt Alternating Convex Search (ACS) [5] to solve Eq. 4.2.6:

(1) $(\mathbf{W}, \mathbf{b}, \boldsymbol{\xi})$ can be learned using any linear multiclass SVM solver such as LIBLINEAR [49] while fixing $(\mathbf{B}, \mathbf{t}, \boldsymbol{\rho})$;

(2) $(\mathbf{B}, \mathbf{t}, \boldsymbol{\rho}, \boldsymbol{\xi})$ can be learned using quadratically constrained quadratic programming (QCQP) [19] while fixing $(\mathbf{W}, \mathbf{b})$, because all the quadratic terms in the objective function and the constraints are positive semidefinite;

(3) Repeat (1) and (2) above until converge.

Theorem 4.2 (Convergence [59]). *Given $\mathcal{B} \subseteq \mathbb{R}^n \times \mathbb{R}^m$, let $f : \mathbb{B} \to \mathbb{R}$ be a biconvex function bounded from below. If the optimization problem in each update of ACS is solvable, then the sequence of function values generated by ACS converges monotonically.*

Clearly, the optimization of Eq. 4.2.6 using ACS will converge to a local minimum. Since learning classifiers while fixing $\mathbf{B}$ and $\mathbf{t}$ is simply applying linear SVM solvers to Eq. 4.2.6, in the following sections, we will focus on how to learn $\mathbf{B}$ and $\mathbf{t}$ efficiently.

4.2.2.3 Supervised Learning of B

We learn $\mathbf{B}$ while fixing $(\mathbf{W}, \mathbf{b})$ and $\mathbf{t}$. The main difficulty of learning $\mathbf{B}$ is to satisfy the quadratic constraint $\|\mathbf{B}\|_2^2 \leq 1$, as it makes the problem NP-hard. Unfortunately, considering the huge number of variables in Eq. 4.2.6 that need to be learned, semidefinite programming (SDP), a widely used relaxation of QCQP, is inapplicable for solving our problem.

Inspired by some recent work on random projection for classification [68,117], we propose an algorithm to search for a sub-optimal solution of Eq. 4.2.6 using random projection and screening test [135], which is shown in Alg. 4.1.

As we can see, Alg. 4.1 guarantees to converge to a local solution monotonically while fixing $(\mathbf{W}, \mathbf{b})$ and $\mathbf{t}$, and has relatively low computational complexity. Sampling from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$ guarantees that the new entries satisfy the norm condition of each column in $\mathbf{B}$.

4.2.2.4 Supervised Learning of t

Given $(\mathbf{W}, \mathbf{b})$ and $\mathbf{B}$, we utilize the stochastic sub-gradient descent method to learn $\mathbf{t}$, similar to PEGASOS [116], due to its high learning efficiency.

Algorithm 4.1: Random projection screening test algorithm for learning $\mathbf{B}$ in TMFs.

Input : $(\mathbf{W}, \mathbf{b})$, $\mathbf{t}$
Output: $\mathbf{B}$

repeat
  Randomly sample $J (J \in \mathbb{N})$ columns in $\mathbf{B}$, denoted as $\mathbf{B}_s$;
  Randomly sample J i.i.d. vectors from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$, normalized using ℓ_2 norm and denoted as $\mathbf{B}_r$;
  Replace $\mathbf{B}_s$ with $\mathbf{B}_r$ in $\mathbf{B}$, denoted as $\mathbf{B}'$;
  if *The objective value in Eq. 4.2.3 is smaller using* $\mathbf{B}'$ *than using* $\mathbf{B}$ **then**
    $\mathbf{B} \leftarrow \mathbf{B}'$;
  end
until *Converge*;
return $\mathbf{B}$;

Letting $g(z) = \max(z, 0) (z \in \mathbb{R})$, the problem of learning $\mathbf{t}$ in Eq. 4.2.3 can be reformulated as follows:

$$\begin{aligned} \mathbf{t}^* &= \arg\min_{\mathbf{t}} L(\mathbf{x}, \mathbf{y}; \theta, \eta, \mathbf{W}, \mathbf{b}, \mathbf{B}, \mathbf{t}_{lb}, \mathbf{t}_{ub}) \\ &= \arg\min_{\mathbf{t}} \left\{ \frac{\theta}{2\eta} \|\mathbf{t}\|_2^2 + \sum_{c,i} g(1 - \mathbb{I}_{y_i,c} f_c(\mathbf{x}_i; \mathbf{B}, \mathbf{t}, \mathbf{w}_c, b_c)) \right\} \\ &\text{s.t.} \quad \mathbf{t}_{lb} \preceq \mathbf{t} \preceq \mathbf{t}_{ub}, \end{aligned} \tag{4.2.7}$$

where $L(\mathbf{x}, \mathbf{y}; \theta, \eta, \mathbf{W}, \mathbf{b}, \mathbf{B}, \mathbf{t}_{lb}, \mathbf{t}_{ub})$ denotes the regularized loss function in Eq. 4.2.3.

Further, letting $j \in \{1, \cdots, D\}$ denote the j^{th} dimension in $\mathbf{t}$, the sub-gradient of L over $\mathbf{t}^{(j)}$ given a data point $(\mathbf{x}_i, y_i)$ can be calculated as follows:

$$\frac{\partial L}{\partial \mathbf{t}^{(j)}} = \frac{\partial L}{\partial g} \cdot \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial \mathbf{t}^{(j)}} = \frac{\theta}{\eta} \mathbf{t}^{(j)} + \sum_c \left. \frac{\partial g}{\partial z} \right|_{z = 1 - \mathbb{I}_{y_i,c} f_c(\mathbf{x}_i; \mathbf{B}, \mathbf{t}, \mathbf{w}_c, b_c)} \left. \frac{\partial g}{\partial z} \right|_{z = \mathbf{B}_j^T \mathbf{x}_i - \mathbf{t}^{(j)}} \mathbb{I}_{y_i,c} \mathbf{w}_c^{(j)}, \tag{4.2.8}$$

where $\frac{\partial g}{\partial z}$ denotes the sub-gradient of g over z, and $\frac{\partial g}{\partial z} = 1$ if $z > 0$; otherwise, $\frac{\partial g}{\partial z} = 0$. We show our learning algorithm for $\mathbf{t}$ in Alg. 4.2.

Algorithm 4.2: Stochastic sub-gradient descent method for learning $\mathbf{t}$ in TMFs.

Input : $\{(\mathbf{x}_i, y_i)\}$, $(\mathbf{W}, \mathbf{b})$, θ, η, $\mathbf{t}_0$, $\mathbf{t}_{lb}$, $\mathbf{t}_{ub}$
Output: $\mathbf{t}$

for $k \leftarrow 1$ *to* K **do**
  Choose a subset $\mathcal{A}_k \subseteq \{(\mathbf{x}_i, y_i)\}$, where $|A_k| = s_k$;
  $\gamma_k \leftarrow \frac{\eta}{\theta k}$;
  foreach $j \in \{1, \cdots, D\}$ **do**

$$\mathbf{t}^{(j)}_{k-\frac{1}{2}} \leftarrow \mathbf{t}^{(j)}_{k-1} - \frac{\gamma_k}{s_k} \sum_{\mathcal{A}_k} \frac{\partial L}{\partial \mathbf{t}^{(j)}}\bigg|_{\mathbf{t}=\mathbf{t}_{k-1}};$$

$$\mathbf{t}^{(j)}_{k} \leftarrow \min\left\{\max\left\{\mathbf{t}^{(j)}_{k-\frac{1}{2}}, \mathbf{t}^{(j)}_{lb}\right\}, \mathbf{t}^{(j)}_{ub}\right\};$$

  end
end
return $\mathbf{t}$;

4.2.3 Nonlinear Kernel Approximation

We start with some introduction of arc-cosine kernels [25, 26], and then explain why the linear kernel of TMFs can be used to approximate the summation of different arc-cosine kernels. Here, we follow the terminology in [25] for a better explanation.

4.2.3.1 Arc-cosine Kernels

In [25, 26], given two data points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, the n^{th} order arc-cosine kernel function is defined as:

$$K_n(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{\pi}\|\mathbf{x}_1\|_2^n \|\mathbf{x}_2\|_2^n J_n(\theta) = 2\int \mathbf{dw} \frac{e^{-\frac{\|\mathbf{w}\|_2^2}{2}}}{(2\pi)^{\frac{d}{2}}} \Theta(\mathbf{w}^T\mathbf{x}_1)\Theta(\mathbf{w}^T\mathbf{x}_2)(\mathbf{w}^T\mathbf{x}_1)^n(\mathbf{w}^T\mathbf{x}_2)^n, \tag{4.2.9}$$

where $\theta = \cos^{-1}\left(\frac{\mathbf{x}_1^T\mathbf{x}_2}{\|\mathbf{x}_1\|_2^2\|\mathbf{x}_2\|_2^2}\right)$, $J_n(\theta) = (-1)^n(\sin\theta)^{2n+1}\left(\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}\right)^n\left(\frac{\pi-\theta}{\sin\theta}\right)$, and $\Theta(\cdot) = \frac{1}{2}(1 + \text{sign}(\cdot))$ denotes the element-wise Heaviside step function. Note that the $n = 1$ arc-cosine kernel, $K_1(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{\pi}\|\mathbf{x}_1\|_2\|\mathbf{x}_2\|_2\{\sin\theta + (\pi - \theta)\cos\theta\}$, is not translationally invariant.

4.2.3.2 Kernel Approximation

By comparing Eq. 4.2.2 with Eq. 4.2.9, we have the following proposition on the relationship between TMFs and arc-cosine kernels.

Proposition 4.2. *(1) Given a data point* $\mathbf{x} \in \mathbb{R}^d$, *then*

$$\forall j = 1, \cdots, J,\ \phi(\mathbf{x}; \mathbf{B}_j, 0) = \Theta(\mathbf{B}_j^T\mathbf{x})\mathbf{B}_j^T\mathbf{x}. \tag{4.2.10}$$

(2) [25] Given two data points $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$, *if each column in* $\mathbf{B}$ *is sampled i.i.d. from a normal distribution with zero mean and unit variance, then*

$$\lim_{J\to+\infty} \frac{2}{J}\phi(\mathbf{x}_1; \mathbf{B}, 0)^T\phi(\mathbf{x}_2; \mathbf{B}, 0) = K_1(\mathbf{x}_1, \mathbf{x}_2). \tag{4.2.11}$$

Theorem 4.3 (Nonlinear kernel approximation). *Given TMF parameters* $(\mathbf{B}, \mathbf{t})$, *suppose* $\mathbf{B}$ *and* $\mathbf{t}$ *can be divided into* M *segments without replacement, so that* $\forall m \in \{1, \cdots, M\}, \exists \mu_m, \mathbf{B}_m^T\mu_m = \mathbf{t}_m$. *Letting the number of columns in each segment* $\mathbf{B}_m$ *be* J_m, *for two data points* $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^d$ *we have*

$$\sum_{m=1}^{M}\left\{\lim_{J_m\to+\infty}\frac{2}{J_m}\phi(\mathbf{x}_1; \mathbf{B}_m, \mathbf{t}_m)^T\phi(\mathbf{x}_2; \mathbf{B}_m, \mathbf{t}_m)\right\} = \sum_{m=1}^{M} K_1(\mathbf{x}_1 - \mu_m, \mathbf{x}_2 - \mu_m). \tag{4.2.12}$$

Proof.

$$\begin{aligned} \forall \mathbf{x},\ \forall m,\ \phi(\mathbf{x}; \mathbf{B}_m, \mathbf{t}_m) &= \max\left(\mathbf{B}_m^T\mathbf{x} - \mathbf{t}_m, 0\right) \\ &= \max\left(\mathbf{B}_m^T[\mathbf{x} - \mu_m], 0\right) = \phi\left(\mathbf{x} - \mu_m; \mathbf{B}_m, 0\right). \end{aligned} \tag{4.2.13}$$

Then by applying Proposition (2) into each segment, the theorem is proven. □

Theorem 4.3 indicates that our TMFs can be considered as the combination of sample dimensions from different infinite dimensional features which can be used to approximate arc-cosine kernels.

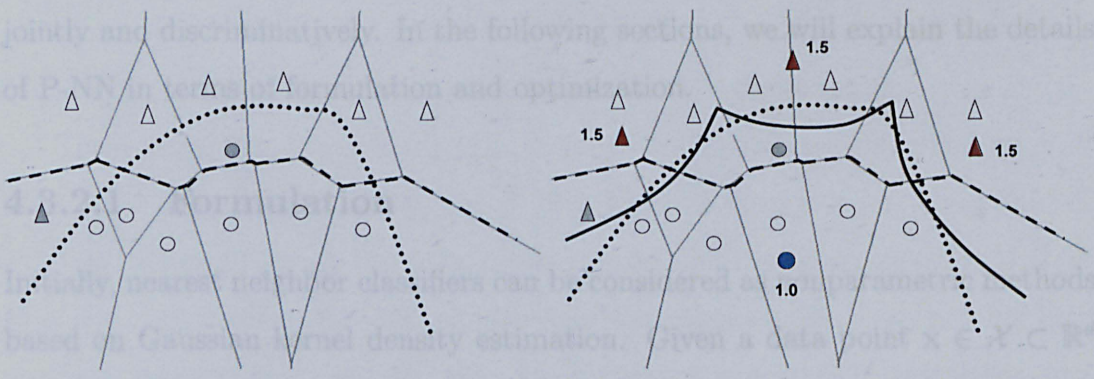
4.3 From Linear to Nonlinear: Parametric Nearest Neighbor Classifiers

4.3.1 Introduction

Now we have shown two methods of training locally linear classifiers: orthogonal coordinate coding (OCC) with LL-SVMs, or truncated marginal features (TMFs) with linear SVMs. Since the data distribution in the feature space is so complex that in some situations, even locally linear classifiers cannot separate data properly, or there will be too many locally linear fragments (one fragment, one classifier) that are needed for classification. This will lead to either low classification accuracy or high computational complexity, which violates our goal of learning local classifiers. Therefore, to overcome these situations, the next question comes to us: Can we go beyond locally linear classifiers to locally nonlinear classifiers?

Let us look at the learning process of locally linear classifiers. Basically whether a classifier is locally linear or locally nonlinear depends on the decision function, which is embedded in the margin constraint of the formulation for learning, such as Eq. 4.1.1 and Eq. 4.2.3. If the decision function is defined nonlinearly, then in a similar way we may learn a locally nonlinear classifier. To learn locally nonlinear classifiers, in this section, we will revisit the nearest neighbor classifiers, whose decision boundaries are defined based on the square Euclidean distances.

In fact, the vanilla nearest neighbor (NN) algorithm is one of the simplest locally linear classifiers, because all the prototypes for classification share the equal weight, making the decision boundary between any pair of prototypes become a line. However, the vanilla NN classifier lacks robustness due to the noise often present in real-world data. Therefore, we propose a novel max-margin based *Parametric Nearest Neighbor* classifier (P-NN), and its extension Ensemble of P-NN (EP-NN). Our method extends the nonparametric kernel estimation [12], and jointly learns the prototypes and their associated weights for classification, which are not the same for all the prototypes any more. Each learned prototype



(a) Nearest neighbor classifier (b) Parametric nearest neighbor classifier

Figure 4.5: Illustration of the differences between (a) the nonparametric nearest neighbor classifier (*i.e.* 1-NN) and (b) our parametric nearest neighbor classifier (P-NN), where 2 classes are represented by △ and ◯, the 3 red triangles are the prototypes in class △, the blue circle is the prototype in class ◯, the numbers close to the prototypes are their weights, the dashed curve denotes the decision boundary of 1-NN, the solid hyperbolas in (b) denote the decision boundary of P-NN, and the dotted curve denotes the optimal decision boundary which needs approximation. Clearly, 1-NN makes no attempt to approximate the optimal decision boundary. However, our P-NN learns not only the prototypes in each class but also the classifier parameters (*i.e.* the nonnegative weights of the prototypes and the bias terms for different classes), which approximates the optimal decision boundary locally using hyperbolas based on the weighted squared Euclidean distances.

is represented by a locally linear combination of some data points. The functionality of the prototypes in our method is similar to the support vectors in kernel-based SVMs, but fully controllable. The classification decision boundaries in our classifiers are built based on the minimum weighted squared Euclidean distances between the data points and the prototypes, which is locally nonlinear.

4.3.2 Parametric Nearest Neighbor Classifiers

As illustrated in Fig. 4.5, in general the decision boundary of a nearest neighbor classifier is locally linear, but it makes no attempt to approximate the optimal decision boundary for classification. On the contrary, our parametric nearest neighbor classifier (P-NN) aims to approximate the optimal decision boundary locally by learning both the prototypes for each class and the classifier parameters

jointly and discriminatively. In the following sections, we will explain the details of P-NN in terms of formulation and optimization.

4.3.2.1 Formulation

Initially, nearest neighbor classifiers can be considered as nonparametric methods based on Gaussian kernel density estimation. Given a data point $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^d$ with class label $c \in \mathcal{C}$, where $\mathcal{C}$ denotes the class set, and a set of prototypes $\mathcal{U}_c$ within the same class, suppose that the window sizes, which are unknown, in the Gaussian kernels for the prototypes in each class are the same, denoted as $h_c \geq 0$, then the probability of the data point $\mathbf{x}$ belonging to a class c can be formulated as follows:

$$p(c|\mathbf{x}) \propto p(\mathbf{x}|c) = \frac{1}{z_c} \sum_{\mathbf{u}_j \in \mathcal{U}_c} \exp\left\{-\frac{\|\mathbf{x} - \mathbf{u}_j\|^2}{2(h_c)^2}\right\} \tag{4.3.1}$$

where $\|\cdot\|$ denotes the ℓ_2-norm and $z_c = |\mathcal{U}_c|(2\pi)^{\frac{d}{2}}(h_c)^d$ is a normalization factor of the density function where $|\mathcal{U}_c|$ denotes the number of prototypes in class c. Following [6], $p(\mathbf{x}|c)$ can be approximated by the largest term in the summation. Then by taking the log-likelihood, Eq. 4.3.1 can be rewritten as:

$$-\log p(c|\mathbf{x}) \approx w_c \left\{\min_{\mathbf{u}_j \in \mathcal{U}_c} \|\mathbf{x} - \mathbf{u}_j\|^2\right\} + b_c \tag{4.3.2}$$

where $w_c = \frac{1}{2(h_c)^2} \geq 0$, $b_c = \log z_c + \log p(\mathbf{x}) - \log p(c)$, $p(\mathbf{x})$ and $p(c)$ are the fixed prior probabilities of data point $\mathbf{x}$ and class c, respectively.

Nonparametric nearest neighbor classifiers assume that given a test data point $\mathbf{x} \in \mathbb{R}^d$, all the w's and b's for all the classes are the same. This leads to the class label prediction rule in nearest neighbor classifiers as follows:

$$c^* = \arg\min_{c \in \mathcal{C}, \mathbf{u}_j \in \mathcal{U}_c} \|\mathbf{x} - \mathbf{u}_j\|^2. \tag{4.3.3}$$

However, as argued in [6], because both the window size h_c and the class prior probability $p(c)$ could vary a lot for different classes, the assumption in the nonparametric nearest neighbor classifiers hardly holds for most cases. On the

contrary, our P-NN estimates w's and b's for all the training classes as well as learning the prototypes for each class by maximizing the margins.

Given a training data set $(\mathbf{x}_i, y_i)_{i=1,\cdots,|\mathcal{X}|}$ with $|\mathcal{X}|$ data points, where $\forall i, \mathbf{x}_i \in \mathcal{X} \subset \mathbb{R}^d$ is a data point and $y_i \in \mathcal{C} \subset \mathbb{N}$ is its class label, for any class $\forall c \in \mathcal{C}$, P-NN attempts to jointly learn the prototypes $\mathcal{U}_c$ and the class model $(\mathbf{w}_c, \mathbf{b}_c)$, so that the minimum weighted Euclidean distance between each data point $\mathbf{x}_i$ and the prototypes in $\mathcal{U}_{y_i}$ is smaller than the minimum weighted Euclidean distance between $\mathbf{x}_i$ and any prototype in $\mathcal{U}_{\bar{c}} = \bigcup_{c \in \mathcal{C} \setminus \{y_i\}} \mathcal{U}_c$, where $\setminus$ denotes the set complement operator. Therefore, based on the hinge loss, P-NN can be formulated as the following optimization problem:

$$
\begin{aligned}
&\min_{\mathbf{u},\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_i \xi_i \qquad (4.3.4)\\
&\text{s.t.} \quad \forall i, c_i \in \mathcal{C} \setminus \{y_i\}, \ \min_{c_i} \left\{ w_{c_i} \min_{\mathbf{u}_k \in \mathcal{U}_{c_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2 + b_{c_i} \right\} \\
&\qquad\qquad\qquad\qquad \geq w_{y_i} \min_{\mathbf{u}_j \in \mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2 + b_{y_i} + 1 - \xi_i, \\
&\qquad \forall i, \ \xi_i \geq 0, \\
&\qquad \forall c \in \mathcal{C}, \ w_c \geq 0,
\end{aligned}
$$

where $\lambda \geq 0$ is a pre-defined regularization parameter, $\boldsymbol{\xi}$ denotes the set of slack variables, $\mathbf{u}_j \in \mathcal{U}_{y_i}$ (*resp.* $\mathbf{u}_k \in \mathcal{U}_{c_i}$) denotes a prototype in $\mathcal{U}_{y_i}$ (*resp.* $\mathcal{U}_{c_i}$), and w_{y_i} and b_{y_i} (*resp.* w_{c_i} and b_{c_i}) are the class model parameters for class y_i (*resp.* c_i). We denote $(\mathbf{w}, \mathbf{b})$ as the classifier parameters, which are vectors consisting of all w's and b's respectively. Finally, a test data point $\mathbf{x}$ is labeled as:

$$
c^* = \arg\min_{c \in \mathcal{C}} \left\{ w_c \min_{\mathbf{u}_j \in \mathcal{U}_c} \|\mathbf{x} - \mathbf{u}_j\|^2 + b_c \right\}. \qquad (4.3.5)
$$

4.3.2.2 Optimization

We adopt an alternating optimization method between learning prototypes and learning classifier parameters to solve the non-convex problem in Eq. 4.3.4.

Learning Prototypes

We update $\mathbf{u}$ and $\boldsymbol{\xi}$ in Eq. 4.3.4 while fixing $\mathbf{w}$ and $\mathbf{b}$ using stochastic gradient descent, similar to the *online-loss-minimization* algorithm in [29]. We say that $\hat{c}_i$

Algorithm 4.3: Initialization of the prototypes for each class: $\mathcal{U} = \mathit{InitializePrototypes}(\mathcal{X}, \mathcal{Y}, \{|\mathcal{U}_c|\})$

Input : training data $(\mathcal{X}, \mathcal{Y})$, number of prototypes per class $\{|\mathcal{U}_c|\}_{c \in \mathcal{C}}$
Output: prototypes $\mathcal{U} = \bigcup_{c \in \mathcal{C}} \mathcal{U}_c$

foreach $c \in \mathcal{C}$ **do**
  $\mathcal{U}_c \leftarrow \emptyset$;
  repeat
    Randomly select data $(\mathbf{x}, y) \in (\mathcal{X}, \mathcal{Y})$ so that $\mathbf{x} \notin \mathcal{U}_c$ and $y = c$;
    $\mathcal{U}_c \leftarrow \mathcal{U}_c \bigcup \{\mathbf{x}\}$;
  until $|\mathcal{U}_c|$ *data points has been added*;
end
return $\mathcal{U} = \bigcup_{c \in \mathcal{C}} \mathcal{U}_c$;

Algorithm 4.4: Stochastic gradient descent for learning prototypes: $\mathcal{U} = \mathit{LearnPrototypes}(\mathcal{X}, \mathcal{Y}, \{\eta_i\}, \mathcal{U}, \mathbf{w}, \mathbf{b})$

Input : training data $(\mathcal{X}, \mathcal{Y})$, learning rate $\{\eta_i\}$, prototypes $\mathcal{U}$, classifier parameters $(\mathbf{w}, \mathbf{b})$
Output: prototypes $\mathcal{U} = \bigcup_{c \in \mathcal{C}} \mathcal{U}_c$

foreach $(\mathbf{x}_i, y_i) \in (\mathcal{X}, \mathcal{Y})$ **do**
  if $\min_{c_i \in \mathcal{C} \setminus \{y_i\}} \left\{ w_{c_i} \min_{\mathbf{u}_k \in \mathcal{U}_{c_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2 + b_{c_i} \right\} <$
  $w_{y_i} \min_{\mathbf{u}_j \in \mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2 + b_{y_i} + 1$ **then**
    $\mathbf{u}_j^* = \arg\min_{\mathbf{u}_j \in \mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2$;
    $\mathbf{u}_k^* = \arg\min_{\mathbf{u}_k \in \mathcal{U}_{\hat{c}_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2$;
    $\mathbf{u}_j^* \leftarrow \mathbf{u}_j^* + \eta_i w_{y_i} (\mathbf{x}_i - \mathbf{u}_j^*)$;
    $\mathbf{u}_k^* \leftarrow \mathbf{u}_k^* - \eta_i w_{\hat{c}_i} (\mathbf{x}_i - \mathbf{u}_k^*)$;
  end
end
return $\mathcal{U} = \bigcup_{c \in \mathcal{C}} \mathcal{U}_c$;

is the *closest class label* to y_i for $\mathbf{x}_i$ if

$$\hat{c}_i = \arg\min_{c_i \in \mathcal{C} \setminus \{y_i\}} \left\{ w_{c_i} \min_{\mathbf{u}_k \in \mathcal{U}_{c_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2 + b_{c_i} \right\}. \tag{4.3.6}$$

Letting $g(\mathbf{x}_i, \mathbf{u}; \mathbf{w}, \mathbf{b}) = \xi_i$ be the hinge loss given a data point $\mathbf{x}_i$, and $\hat{c}_i$ be the closest class label to y_i for $\mathbf{x}_i$, then the sub-gradient of g w.r.t. an arbitrary prototype $\mathbf{u}$, denoted as $\frac{\partial g}{\partial \mathbf{u}}$, is: if $\xi_i > 0$, then $\frac{\partial g}{\partial \mathbf{u}_j^*} = 2w_{y_i} \left(\mathbf{u}_j^* - \mathbf{x}_i\right)$ and $\frac{\partial g}{\partial \mathbf{u}_k^*} = 2w_{\hat{c}_i} \left(\mathbf{x}_i - \mathbf{u}_k^*\right)$, where $\mathbf{u}_j^* = \arg\min_{\mathbf{u}_j \in \mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2$ and $\mathbf{u}_k^* = \arg\min_{\mathbf{u}_k \in \mathcal{U}_{\hat{c}_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2$; otherwise, $\frac{\partial g}{\partial \mathbf{u}} = \mathbf{0}$.

Then we can use the following equation to update $\mathbf{u}$ given a data point $\mathbf{x}_i$:

$$\forall \mathbf{u} \in \left\{ \mathcal{U} = \bigcup_{c \in \mathcal{C}} \mathcal{U}_c \right\}, \quad \mathbf{u}^{(t+1)} = \mathbf{u}^{(t)} - \eta_t \frac{\partial g}{\partial \mathbf{u}^{(t)}}, \tag{4.3.7}$$

where η_t and $\frac{\partial g}{\partial \mathbf{u}^{(t)}}$ denote the learning rate parameter and the sub-gradient for $\mathbf{u}$ at iteration $t \in \mathbb{N}$, respectively, and $\bigcup$ denotes the set union operator.

Alg. 4.3 and Alg. 4.4 show our learning algorithms, where we use some training points as the initial prototypes, because at the beginning we want to guarantee that the data points and the prototypes are definitely in the same class, or definitely not. Other clustering algorithms such as K-Means could be used as well to initialize the prototypes.

Learning Classifier Parameters

We update $\mathbf{w}$, $\mathbf{b}$ and $\boldsymbol{\xi}$ in Eq. 4.3.4 while fixing $\mathbf{u}$. Then given data $(\mathbf{x}_i, y_i)$, letting $\mathbf{v}_i$ be a $|\mathcal{C}|$-dimensional vector consisting of 0's, where $|\mathcal{C}|$ is the number of classes, and $\hat{c}_i$ be the closest class label to y_i for $\mathbf{x}_i$, we set $\mathbf{v}_i(\hat{c}_i) = \min_{\mathbf{u}_k \in \mathcal{U}_{\hat{c}_i}} \|\mathbf{x}_i - \mathbf{u}_k\|^2$ and $\mathbf{v}_i(y_i) = -\min_{\mathbf{u}_j \in \mathcal{U}_{y_i}} \|\mathbf{x}_i - \mathbf{u}_j\|^2$, where $\mathbf{v}_i(\cdot)$ denotes the value at a particular bin of vector $\mathbf{v}_i$. Therefore, Eq. 4.3.4 can be rewritten as follows:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ & \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_i \xi_i \\ \text{s.t.} \ & \forall i,\ \mathbf{w}^T \mathbf{v}_i + b_{\hat{c}_i} - b_{y_i} \geq 1 - \xi_i, \\ & \forall i,\ \xi_i \geq 0, \\ & \forall c \in \mathcal{C},\ w_c \geq 0, \end{aligned} \tag{4.3.8}$$

where $(\cdot)^T$ denotes the matrix transpose operator. Notice that both $\hat{c}_i$ and $\mathbf{v}_i$ are dependent on the classifier parameters $(\mathbf{w}, \mathbf{b})$. So if the classifier parameters are updated, $\hat{c}_i$ and $\mathbf{v}_i$ should be updated as well. Thus, we present an iterative optimization algorithm to solve Eq. 4.3.8 as shown in Alg. 4.5, where Ω denotes a set of triplets.

Finally, based on Alg. 4.3-4.5, we can jointly learn the prototypes and the classifier parameters by maximizing the margin in an alternating manner, as presented in Alg. 4.6, where FLT_MAX denotes the max value that we can set

Algorithm 4.5: Iterative optimization for solving Eq. 4.3.8: $(\mathbf{w}, \mathbf{b}) = LearnClassifiers(\mathcal{X}, \mathcal{Y}, \mathcal{U}, \mathbf{w}, \mathbf{b})$

Input : training data $(\mathcal{X}, \mathcal{Y})$, prototypes $\mathcal{U}$, classifier parameters $(\mathbf{w}, \mathbf{b})$
Output: classifier parameters $(\mathbf{w}, \mathbf{b})$

$\Omega \leftarrow \emptyset$;
repeat
  foreach $(\mathbf{x}_i, y_i) \in (\mathcal{X}, \mathcal{Y})$ **do**
    Calculate $\hat{c}_i$ using Eq. 4.3.6 and $\mathbf{v}_i \in \mathbb{R}^{|\mathcal{C}|}$;
    $\Omega \leftarrow \Omega \bigcup \{(\mathbf{v}_i, y_i, \hat{c}_i)\}$;
  end
  Update $\mathbf{w}, \mathbf{b}$ based on Ω using Eq. 4.3.8;
until *Classifier parameters converged*;
return $\mathbf{w}, \mathbf{b}$;

Algorithm 4.6: Alternating optimization for solving Eq. 4.3.4

Input : training data $(\mathcal{X}, \mathcal{Y})$, learning rate $\{\eta_i\}$, number of prototypes per class $\{|\mathcal{U}_c|\}_{c \in \mathcal{C}}$
Output: prototypes $\mathcal{U}$, classifier parameters $(\mathbf{w}, \mathbf{b})$

foreach $c \in \mathcal{C}$ **do**
  $w_c \leftarrow$ FLT_MAX, $b_c \leftarrow 0$;
end
$\mathcal{U} = InitializePrototypes(\mathcal{X}, \mathcal{Y}, \{|\mathcal{U}_c|\})$;
repeat
  $\mathcal{U} = LearnPrototypes(\mathcal{X}, \mathcal{Y}, \{\eta_i\}, \mathcal{U}, \mathbf{w}, \mathbf{b})$;
  $(\mathbf{w}, \mathbf{b}) = LearnClassifiers(\mathcal{X}, \mathcal{Y}, \mathcal{U}, \mathbf{w}, \mathbf{b})$;
until *Converged*;
return $\mathcal{U}, \mathbf{w}, \mathbf{b}$;

to w's so that Eq. 4.3.4 can be optimized from its largest value.

4.3.3 Ensemble of Parametric Nearest Neighbor Classifiers

P-NN assumes that the window sizes in the Gaussian kernel density estimation are the same for all the prototypes in the same class, while varying for different classes. However, this assumption is quite strong, because even for the prototypes in the same class, the window sizes may vary individually.

In order to relax this assumption, we take advantage of the random initialization of the prototypes in P-NN due to the non-convexity of Eq. 4.3.4, similar

to random forest [21]. In this way, we further introduce an *Ensemble of P-NN* (EP-NN) classifier to boost the classification accuracy. We call the set of learned prototypes in each P-NN a *base learner*. Rather than learning one base learner with many prototypes for each class, which risks overfitting the training data, EP-NN jointly learns multiple base learners with reasonable numbers of prototypes per class.

Given a training data set $(\mathbf{x}_i, y_i)_{i=1,\cdots,|\mathcal{X}|}$, EP-NN is formulated as below to jointly learn $|\mathcal{L}|$ base learners and the classifier parameters, where $l \in \mathcal{L}$ denotes the l^{th} base learner in the set $\mathcal{L}$:

$$\begin{aligned}
&\min_{\mathbf{u},\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ \frac{\lambda}{2}\sum_{c\in\mathcal{C}}\|\mathbf{w}_c\|^2 + \sum_i \xi_i \qquad (4.3.9)\\
&\text{s.t.} \quad \forall i, c_i \in \mathcal{C}\setminus\{y_i\},\ \min_{c_i}\left\{\textstyle\sum_{l\in\mathcal{L}} w_{c_i}^l \min_{\mathbf{u}_k\in\mathcal{U}_{c_i}^l} \|\mathbf{x}_i-\mathbf{u}_k\|^2 + b_{c_i}\right\}\\
&\qquad\qquad \geq \textstyle\sum_{l\in\mathcal{L}} w_{y_i}^l \min_{\mathbf{u}_j\in\mathcal{U}_{y_i}^l} \|\mathbf{x}_i-\mathbf{u}_j\|^2 + b_{y_i} + 1 - \xi_i,\\
&\qquad \forall i,\ \xi_i \geq 0,\\
&\qquad \forall c\in\mathcal{C}, \forall l\in\mathcal{L},\ w_c^l \geq 0.
\end{aligned}$$

We can easily modify Alg. 4.3-4.6 to solve Eq. 4.3.9 by considering all the base learners together for each update. In the same way, we can easily extend EP-NN by taking multi-source information into account.

4.3.4 Implementation

In order to compare P-NN and EP-NN with other locally linear methods easily, especially the coding based locally linear methods, as well as making a fast implementation, in practice we followed the stacked generalization framework and implemented our classifiers approximately in a two-stage way: first encoding data and then training multiclass linear SVMs. Empirically the classification accuracies of this implementation are very close to those of P-NN and EP-NN based on Alg. 4.6, with much faster training speed and less care of parameter tuning.

(I) Encoding data. We learn each base learner independently so that this process can be parallelized. After the first update of the prototypes in Alg. 4.6, we stop updating prototypes, because empirically we find that these prototypes

are good enough for classification.

To encode data, we map each data point into a distance based sparse vector. Given a data point $\mathbf{x} \in \mathcal{X}$ and $|\mathcal{L}|$ base learners, letting $\forall l \in \mathcal{L}, \mathbf{v}_i^l \in \mathbb{R}^{|\mathcal{C}|}$ be a vector, where $|\mathcal{C}|$ is the number of classes, we set the c^{th} bin in $\mathbf{v}_i^l$ as $\mathbf{v}_i^l(c) = \min_{\mathbf{u}_j \in \mathcal{U}_c^l} \|\mathbf{x}_i - \mathbf{u}_j\|^2$, where $c = \arg\min_{c \in \mathcal{C}} \left\{\min_{\mathbf{u}_j \in \mathcal{U}_c^l} \|\mathbf{x}_i - \mathbf{u}_j\|^2\right\}$, and 0 to the rest bins. Further, we denote $\mathbf{v}_i$ as our encoded feature vector by concatenating all $\mathbf{v}_i^l$'s and normalizing it using ℓ_1-norm. Notice that our distance based feature vectors are $|\mathcal{C}| \times |\mathcal{L}|$ dimensional, but in each vector only $|\mathcal{L}|$ bins are non-zeros.

(II) Training multiclass linear SVMs. By taking the encoded data as the input, we can train the following standard multiclass linear SVMs [30] for classification:

$$\begin{aligned} \min_{\mathbf{w},\mathbf{b},\boldsymbol{\xi}} \ & \frac{\lambda}{2}\sum_c \|\mathbf{w}_c\|^2 + \sum_{i,c_i} \xi_{i,c_i} \\ \text{s.t.} \ & \forall i, \forall c_i \in \mathcal{C} \setminus \{y_i\}, \ \left[\mathbf{w}_{c_i}^T \mathbf{v}_i + b_{c_i}\right] - \left[\mathbf{w}_{y_i}^T \mathbf{v}_i + b_{y_i}\right] \geq 1 - \xi_{i,c_i}, \\ & \xi_{i,c_i} \geq 0. \end{aligned} \tag{4.3.10}$$

Here we relax Eq. 4.3.8 by (1) removing the nonnegative constraints on $\mathbf{w}$, and (2) allowing that the weights of the prototypes in the same class can be changed in different base learners, rather than fixed values.

4.4 Computational Complexity

We denote the data dimension, and the numbers of basis vectors, training data points, categories, and iterations in training as d, D, N, C, and K, respectively, and assume that the computational complexities of the unit operations $+$, $-$, $*$, $\leq$, and $\geq$ are the same, denoted as $O(1)$. We measure the computational complexity by counting how many unit operations involved in training or testing.

4.4.1 Orthogonal Coordinate Coding with LL-SVMs

For this method, the computational complexity during training can be divided into three parts:

(1) The computational complexity of learning OCC is equivalent to that of SVD, which is $O(kd^2N + k'N^3)$ where k and k' are constants [58];

(2) Encoding the training data requires $O(dDN)$;

(3) As stated in [144], the computational complexity of training a binary linear SVM solver can be as low as $O(d)$ per iteration, such as PEGASOS. Therefore, training multiclass LL-SVMs needs $O(CdDK)$.

Overall, the computational complexity of training a classifier using G-OCC and LL-SVMs is $O(kd^2N + k'N^3) + O(dDN) + O(CdDK)$, while using C-OCC and LL-SVMs it requires $O(Ckd^2N + Ck'N^3) + O(CdDN) + O(CdDK)$.

During testing, the computational complexity using G-OCC and LL-SVMs per data is $O((3 + 2C)dD)$, while using C-OCC and LL-SVMs per data it is $O(5CdD)$.

4.4.2 Truncated Marginal Features with Linear SVMs

For this method, the computational complexity during training can be divided into three parts as follows:

(1) For learning $\mathbf{B}$, the computational complexity of Alg. 4.1 is $O(dDNCK)$;

(2) For learning $\mathbf{t}$, the computational complexity of Alg. 4.2 is $O(DCK)$;

(3) Based on [144], training multiclass linear SVMs requires $O(DCK)$.

Therefore, the overall computational complexity during test is $O(dDNCK) + O(DCK) + O(DCK)$. During testing, the computational complexity of this method is $O(2D(d + C))$ per data.

4.4.3 Parametric Nearest Neighbor Classifiers

In general the computational complexity of the min operator [2] is $O(d)$. The distance between a data point $\mathbf{x}$ and a prototype $\mathbf{u}$ is $\|\mathbf{x}-\mathbf{u}\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x}^T\mathbf{u} + \|\mathbf{u}\|^2$, where $\|\mathbf{x}\|^2$, $\|\mathbf{u}\|^2$ and $2\mathbf{u}$ can be pre-calculated. Therefore, the computational complexity of calculating distances is $(2d+2)\cdot O(1)$.

Letting $\forall c \in \mathcal{C}, |\mathcal{U}_c|$ be the number of prototypes for class c, during training the computational complexity of this method can be divided into two parts as follows:

(1) Learning prototypes using Alg. 4.4 needs $O(K((2d+3)\sum_c |\mathcal{U}_c| + 6d)) = O(K((2d+3)D + 6d))$, where $D = \sum_c |\mathcal{U}_c|$;

(2) Based on [144], training multiclass linear SVMs requires $O(DCK)$.

So the training computational complexity of P-NN is $O(K((2d+3)D+6d)) + O(DCK)$. During testing it is $O((2d+3)D + 3C)$ per data. Notice that the ensemble of P-NN classifiers (EP-NN) shares similar computational complexity to that of P-NN.

4.5 Experiments

4.5.1 Datasets

We test our local classifiers on three optical character recognition (OCR) benchmark datasets for machine learning: MNIST, USPS and LETTER. We use the raw features provided in each dataset so that we can compare our results fairly with others.

MNIST contains 40000 training and 10000 testing gray-scale images with resolution 28×28 pixels, which are normalized directly into 784 dimensional vectors. The label of each image is one of the 10 digits from 0 to 9. USPS contains 7291 training and 2007 testing gray-scale images with resolution 16×16 pixels, directly stored as 256 dimensional vectors, and the label of each image still corresponds

[2] In practice, the complexity of the min operator depends on the data structure. At most, it is $O(d)$.

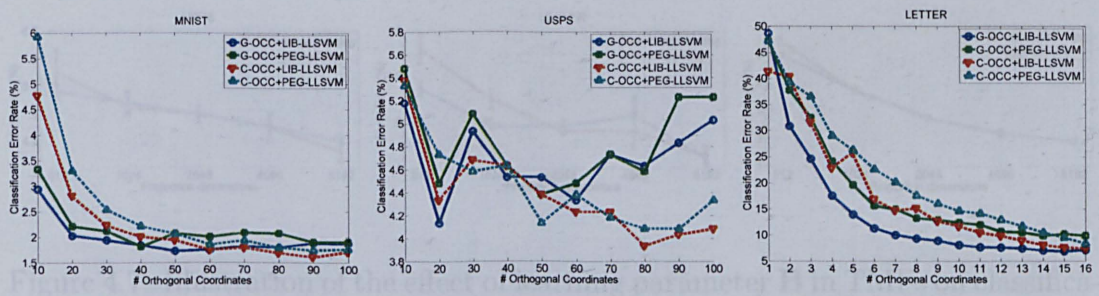


Figure 4.6: Performance comparison among the four different settings of OCC with LL-SVM on MNIST (left), USPS (middle), and LETTER (right) using different numbers of orthogonal basis vectors.

to one of the 10 digits from 0 to 9. LETTER contains 16000 training and 4000 testing images, each of which is represented as a relatively short 16 dimensional vector, and the label of each image corresponds to one of the 26 letters from A to Z.

4.5.2 Tuning Parameters in Local Classifiers

4.5.2.1 Orthogonal Coordinate Coding with LL-SVMs

We re-implement LL-SVM based on LIBLINEAR [49] [3] and PEGASOS [116] [4], respectively, and perform multiclass classification using the one-vs-all strategy. This aims to test the effect of either quadratic programming or stochastic gradient based SVM solver on both accuracy and computational time. We denote these two implementations of LL-SVM as *LIB-LLSVM* and *PEG-LLSVM* for short. Together with our G-OCC and C-OCC (see Section 4.1.2.3), there are four locally linear classifiers in total, namely, *G-OCC + LIB-LLSVM*, *G-OCC + PEG-LLSVM*, *C-OCC + LIB-LLSVM*, and *C-OCC + PEG-LLSVM*. The regularizer in each classifier is determined using cross validation.

Figure 4.6 shows the comparison of classification error rates among the four classifiers on MNIST (left), USPS (middle), and LETTER (right), respectively, using different numbers of orthogonal basis vectors. With the same OCC, LIB-LLSVM performs slightly better than PEG-LLSVM in terms of accuracy, and

[3] Using LIBLINEAR, we implement LL-SVM based on Eq. 4.1.9.
[4] Using PEGASOS, we implement LL-SVM based on the original formulation in [76].

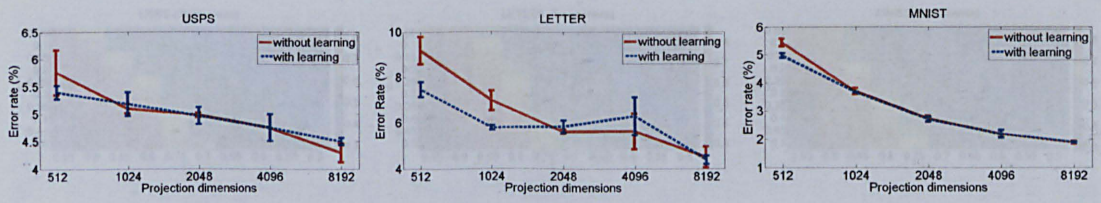


Figure 4.7: Illustration of the effect of learning parameter $\mathbf{B}$ in TMFs on classification with different projection dimensions and $\mathbf{t} = \mathbf{0}$ using USPS (left), LETTER (middle), and MNIST (right), respectively.

both behaves similarly with the increase of the number of orthogonal basis vectors. It seems that in general C-OCC is better than G-OCC.

4.5.2.2 Truncated Marginal Features with Linear SVMs

For this classifier, we set the parameters in Eq. 4.2.3 as follows: $\theta = 10^{-4}\eta$ without further tuning, η is decided using cross validation, and the number of iterations in Alg. 4.2 is fixed to 10^5 without tuning. Classification performance is measured using mean classification error rate across all the classes.

Effect of Learning B in TMFs

To explore the effect of learning parameter $\mathbf{B}$ in TMFs on classification, we design the following experiments: we fix the threshold vector $\mathbf{t} = \mathbf{0}$, and randomly sample the projection matrix $\mathbf{B}$ from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$ with ℓ_2 normalization as initialization. The numbers of columns in $\mathbf{B} \in \mathbb{R}^{d \times D}$ vary from $2^9 = 512$ to $2^{13} = 8192$. The learning process follows Alg. 4.1.

As we can see in Fig. 4.7, with lower dimensions (*e.g.* 512-dim), the learning of projection matrix $\mathbf{B}$ does help to improve the classification performance. However, with the increase of dimensions, the improvement becomes marginal, or even counteractive. This observation is reasonable, because with higher dimensions, the supervised learning of $\mathbf{B}$ is more likely to overfit data. Therefore, in the following experiments we will sample $\mathbf{B}$ with high dimensions once without further learning.

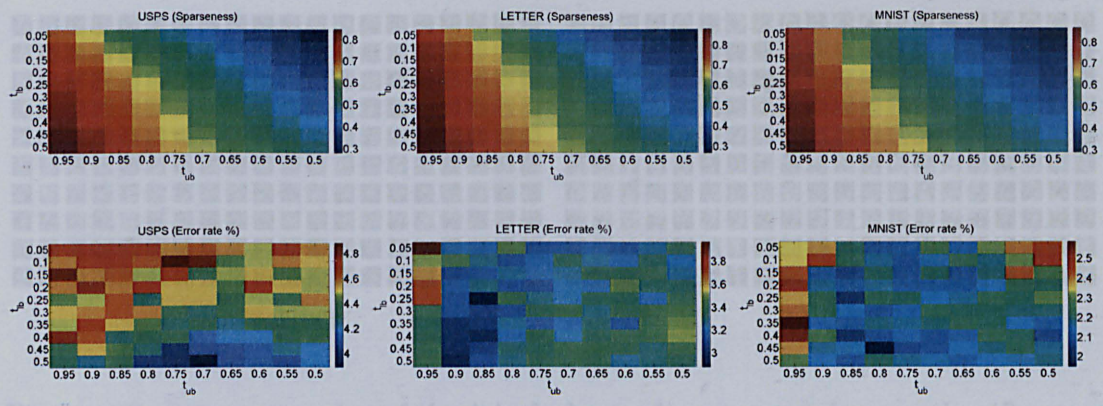


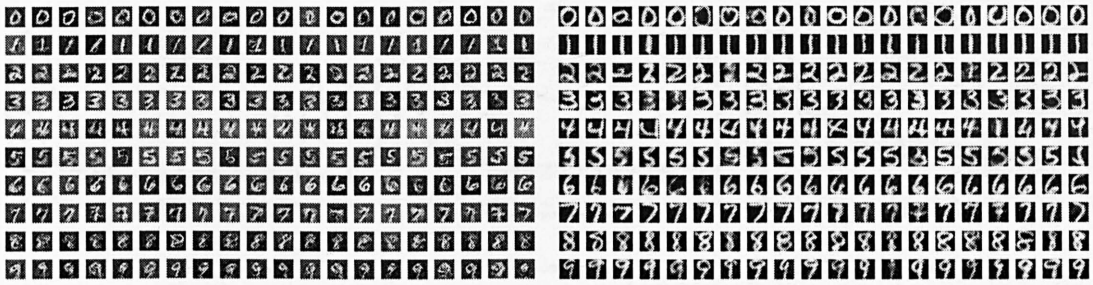
Figure 4.8: Illustration of the effect of learning parameter $\mathbf{t}$ in TMFs on sparseness (top) and error rate (bottom) with different lower bounds (y-axis: $\mathbf{t}_{lb}$) and upper bounds (x-axis: $\mathbf{t}_{ub}$) using USPS (left), LETTER (middle), and MNIST (right), respectively. The projection dimension of TMFs is fixed to 4096.

Effect of Learning t in TMFs

In our optimization problem, the threshold vector $\mathbf{t}$ is bounded by the lower bound $\mathbf{t}_{lb}$ and the upper bound $\mathbf{t}_{ub}$. Therefore, in order to explore the effect of learning $\mathbf{t}$ on classification, we need to explore the effects of different settings of its bounds.

We design the experiments on classification error rate and sparseness in TMFs of test data using different ranges of lower bounds and upper bounds. *Sparseness* of a matrix is defined as the percentage of the number of zeros in the matrix. So the higher the sparseness is, the more percentage of zeros the matrix has. To set the lower bounds and the upper bounds, we assume that the margins of the data follow a Gaussian distribution based on the Central Limit Theorem [110], and use the cumulative distribution function (CDF) to calculate the bounds. The CDF is normalized from 0 to 1, and we use percentages to indicate the lower bounds and upper bounds. In practice, we sample the values from the training data rather than calculating them. Manually we set the lower bounds from 0.05 to 0.5 in CDF, step by 0.05, and the upper bounds from 0.95 to 0.5 in CDF, step by -0.05. The learning process follows Alg. 4.2.

Fig. 4.8 shows our results. As we can see, the distributions of sparseness on USPS, LETTER and MNIST are very similar: the value at the bottom left corner is highest, the value at the top right corner is lowest, and the values between



(a) MNIST (b) USPS

Figure 4.9: Some examples of the jointly learned prototypes by our classifiers on (a) MNIST and (b) USPS, 20 prototypes per class.

these two corners decrease gradually. This demonstrates that the sparseness of the margin matrix is controlled by the lower bound and upper bound via learning $\mathbf{t}$. In terms of classification error rate, the distributions are not so similar to each other. However, the ranges of error rates are quite small, varying about 1%. More interestingly, all of the best performances on these three datasets occur with sparseness equal to around 0.65. Therefore, in the following experiments, we manually set the lower bound to 0.6, and the upper bound to 0.9, respectively, without further tuning.

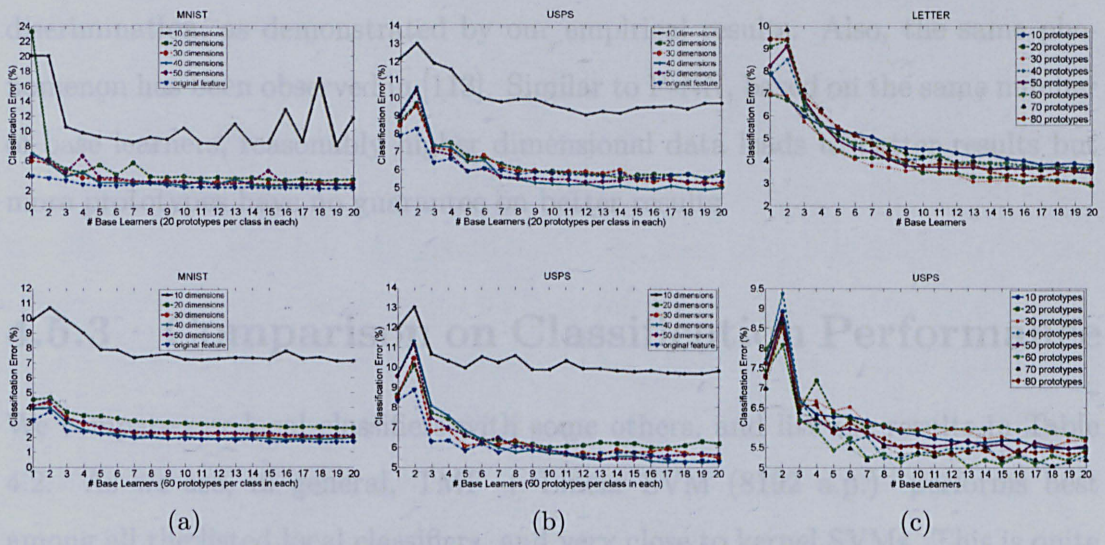
In summary, this locally linear classifier is quite stable *w.r.t.* the changes of initialization points and the bounds, though its formulation is biconvex.

4.5.2.3 Parametric Nearest Neighbor Classifiers

In this method, in order to learn the prototypes in each base learner, we randomly select at most 10^5 data points from the training set, where each data point is allowed to be selected repeatedly, and fix the learning rate to 0.1. LIBLINEAR [49] is employed as our multiclass SVM solver.

We first visualize some of the learned prototypes for MNIST and USPS in Fig. 4.9 (a) and (b), respectively. Each prototype is represented as a linear combination of different training data points and plays a role of a weak classifier. We can roughly see the digit represented by each prototype, which demonstrates the good discriminability of the learned prototypes.

Then we test the robustness of our classifiers *w.r.t.* dimensions of features, numbers of prototypes per class in each base learner, and numbers of base learners.



(a) (b) (c)

Figure 4.10: Performance of EP-NN: classification error *v.s.* the number of base learners on (a) MNIST and (b) USPS with different dimensions of data using 20 (top) and 60 (bottom) prototypes per class; (c) LETTER (top) and USPS (bottom) with different numbers of the prototypes per class in each base learner from 10 to 80, step by 10, using original features. When the number of base learners is equal to 1, EP-NN turns into P-NN. Clearly, EP-NN boosts the performance of P-NN significantly.

To build low-dimensional features, we directly apply singular value decomposition (SVD) to the original data in MNIST and USPS and take the top-K values in the coefficient vector of each data point. Notice that when the number of base learners is equal to 1, EP-NN actually turns into P-NN. Fig. 4.10 summarizes the comparison results among the three factors:

(I) P-NN: From Fig. 4.10(a) and (b), P-NN seems a little sensitive to very low dimensional data (*e.g.* 10 or 20). However, when the feature dimension is higher, P-NN behaves stably within 2% difference, and performs best using the original features. From Fig. 4.10(c), we can see clearly that more prototypes per class does not guarantee a better performance using P-NN, as we expected, but its performance is still reasonably stable within 3% difference.

(II) EP-NN: From Fig. 4.10, we can see that EP-NN really boosts the classification accuracy of P-NN significantly. With only 2 base learners, EP-NN performs worse than P-NN, because sometimes the prototypes will disagree with each other, leading to weak discrimination between classes. However as we increase the number of base learners, the majority will tend to agree giving better

discrimination, as demonstrated by our empirical results. Also, the same phenomenon has been observed in [113]. Similar to P-NN, based on the same number of base learners, reasonably higher dimensional data leads to better results but more prototypes have no guarantee on better results.

4.5.3 Comparison on Classification Performance

We compare our local classifiers with some others, and list the results in Table 4.2. As we see, in general, TMF + Linear SVM (8192 a.p.) performs best among all the listed local classifiers, and very close to kernel SVMs. This is quite reasonable, because this method will localize the data better. In terms of kernel approximation, higher projection dimension will lead to better approximation of summation of multiple kernels. Interestingly, LMNN performs significantly better than kernel SVMs on USPS, leading to a better average performance over the three datasets. This suggests that maybe we should introduce metric learning into local classifier learning as well. We will explore this in our future work.

4.5.4 Comparison on Computational Time

We compare the training time and testing time of our local classifiers with some other methods in Table 4.3 and Table 4.4, respectively. All of our methods are implemented based on a mixture of MATLAB and C++ code, and run on a single thread of Xeon X5550@2.67GHz CPU. The timing of our methods listed in the tables are including every step for learning classifiers, such as data encoding.

In Table 4.3, we can see that the training time for different local classifiers varies a lot, but in general training them is much faster than training kernel SVMs, and for some cases the speed-up is around 10^4. Similar trends can be observed in Table 4.4 for testing as well. Overall, these numbers are consistent with our computational complexity analysis in Section 4.4.

4.6 Conclusion

In this section, we propose two locally linear and one locally nonlinear classifiers.

We propose orthogonal coordinate coding (OCC) to encode high dimensional data based on a set of anchor planes defined by a set of orthogonal basis vectors, which can be easily learned using SVD to minimize the data reconstruction error. Each basis vector can be considered as a weak learner, and by feeding the codes of data into locally linear SVMs (LL-SVMs), OCC can help LL-SVMs to approximate the nonlinear decision boundary in the feature space better.

By extending the idea of weak learners in OCC, we propose a very efficient algorithm for learning locally linear classifiers using truncated marginal features (TMFs), which are generated by an explicit nonlinear map function. This map function performs the data localization in a supervised manner for locally linear classifiers. We formulate this problem as a biconvex minimization problem. Alternating convex search is utilized for solving the problem efficiently and locally, where random projection and stochastic sub-gradient descent are used to learn the parameters in TMFs, and a linear multiclass SVM solver is used to learn the locally linear classifiers. Our method is in favor of sparse features while improving the classification performance, and it can be used to approximate the summation of nonlinear kernels generated by arc-cosine kernels.

Beyond the locally linear classifiers, we propose a locally nonlinear classifier, Parametric Nearest Neighbor (P-NN), and its extension Ensemble of P-NN (EP-NN). These classifiers extend the analysis of the Gaussian kernel density estimation, and attempt to learn the prototypes for nearest neighbor search and the classifier parameters jointly and discriminatively. The decision boundary of our classifiers consists of a set of nonlinear functions, since we use the minimum weighted squared Euclidean distances between the data and the prototypes as the classification criterion. We implement P-NN and EP-NN by following the stacked generalization framework, where each data point is mapped into a very sparse vector based on the minimum distances across the classes in each base learner, and as the inputs multiclass linear SVMs are trained for classification.

We analyze the computational complexity of our local classifiers, and com-

pare them in our experiments with some other methods in terms of classification error rate and running time in both training and testing, respectively. Overall, the performance of our local classifiers are stable and not sensitive to parameter changes within a wide range, and their accuracies are close to those of kernel SVMs, but running much faster, which is consistent with our computational complexity analysis. Considering both accuracy and running time, our locally linear classifier $TMF + LinearSVM$ performs best among all the local classifiers listed in the tables, especially for large-scale datasets. Therefore, this classifier gives us a better chance to handle the extremely large set of data efficiently for object detection in proposal verification.

Table 4.2: Classification error rate comparison (%) between our methods and others on MNIST, USPS, and LETTER. In general, our TMF + Linear SVM (8192 a.p.) performs best among all the listed local classifiers, and very close to kernel SVMs.

Method		MNIST	USPS	LETTER	AVE.
Ours	G-OCC+LIB-LLSVM (# bas. vec.)	1.72 (50)	4.14 (20)	6.85 (15)	4.24
	G-OCC+PEG-LLSVM (# bas. vec.)	1.81 (40)	4.38 (50)	9.83 (14)	5.34
	C-OCC+LIB-LLSVM (# bas. vec.)	1.61 (90)	3.94 (80)	7.35 (16)	4.30
	C-OCC+PEG-LLSVM (# bas. vec.)	1.74 (90)	4.09 (80)	8.30 (16)	4.71
	TMF + Linear SVM (4096 a.p.)	2.19	4.77	3.12	3.36
	TMF + Linear SVM (8192 a.p.)	1.77	***3.89***	***2.73***	***2.80***
	P-NN (40 prototypes per class)	3.13	7.87	6.95	5.98
	EP-NN (40 p.p.c, 20 base learners)	***1.65***	4.88	2.90	3.14
Linear SVMs	Linear SVM (10 passes) [16]	12.00	9.57	41.77	21.11
	LIBLINEAR [49]	8.18	8.32	30.60	15.70
Other local classifiers	LCC + Linear SVM (512 a.p.) [143]	2.64	-	-	-
	LCC + Linear SVM (4096 a.p.) [143]	1.90	-	-	-
	improved LCC + Linear SVM (512 a.p.) [142]	1.95	-	-	-
	improved LCC + Linear SVM (4096 a.p.) [142]	1.64	-	-	-
	LLC + Linear SVM (512 a.p.) [131]	3.69	5.78	9.02	6.16
	LLC + Linear SVM (4096 a.p.) [131]	2.28	4.38	4.12	3.59
	DCN + Linear SVM ($L_1 = 64, L_2 = 512$) [88]	1.51	-	-	-
	LL-SVM (100 a.p., 10 passes) [76]	1.85	5.78	5.32	4.32
	Adaptive Local Hyperplane (ALH) [137]	2.15	4.19	2.95	3.10
Kernel SVMs	LIBSVM (RBF kernel) [24]	**1.36**	-	-	-
	LIBSVM (arc-cosine kernel) [25]	5.57	6.53	2.75	4.95
	LA-SVM (RBF kernel, 1 pass) [18]	1.42	-	-	-
	LA-SVM (RBF kernel, 2 passes) [18]	**1.36**	-	-	-
	MCSVM (RBF kernel) [30]	1.44	4.24	2.42	2.70
	SVM_{struct} (RBF kernel) [124]	1.40	4.38	**2.40**	2.73
	LA-RANK (RBF kernel, 1 pass) [17]	1.41	4.25	2.80	2.82
Others	BP^M+MRG (Budget learning) [133]	-	6.10	10.50	-
	EFM + Linear SVM (Intersection kernel) [126, 128]	9.11	8.12	8.22	8.48
	Nearest Neighbor (1-NN)	3.09	5.08	4.35	4.17
	K Nearest Neighbors (KNN)	2.92	4.88	4.35	4.05
	LMNN [134]	1.70	**0.91**	3.60	**2.07**

"a.p." denotes anchor points.

Table 4.3: Training time (/s) comparison between our methods and others on MNIST, USPS, and LETTER. The numbers in Row 8-15 are copied from [76].

Method		MNIST	USPS	LETTER
Ours	G-OCC + LIB-LLSVM	113.38	5.78	4.14
	G-OCC + PEG-LLSVM	125.03	14.50	2.02
	C-OCC + LIB-LLSVM	224.09	25.61	1.66
	C-OCC + PEG-LLSVM	273.70	23.31	0.85
	TMF + Linear SVM (4096 a.p.)	54.73	7.81	59.92
	TMF + Linear SVM (8192 a.p.)	119.59	14.01	156.75
	EP-NN (40 p.p.c, 20 base learners)	921.11	209.67	42.64
Others	Linear SVM (10 passes) [16]	1.50	0.26	0.18
	LL-SVM (100 a.p., 10 passes) [76]	81.70	6.20	4.20
	LIBSVM (RBF kernel) [24]	1.75×10^4	-	-
	LA-SVM (RBF kernel, 1 pass) [18]	4.90×10^3	-	-
	LA-SVM (RBF kernel, 2 passes) [18]	1.22×10^4	-	-
	MCSVM (RBF kernel) [30]	2.50×10^4	60.00	1.20×10^3
	SVM_{struct} (RBF kernel) [124]	2.65×10^5	6.30×10^3	2.40×10^4
	LA-RANK (RBF kernel, 1 pass) [17]	3.00×10^4	85.00	9.40×10^2

Table 4.4: Testing time (/μs) comparison per data between our methods and others on MNIST, USPS, and LETTER. The numbers in Row 8-12 are copied from [76].

Method		MNIST	USPS	LETTER
Ours	G-OCC + LIB-LLSVM	5.51×10^3	19.23	4.09
	G-OCC + PEG-LLSVM	302.28	23.25	3.33
	C-OCC + LIB-LLSVM	9.57×10^3	547.60	63.13
	C-OCC + PEG-LLSVM	503.18	50.63	28.94
	TMF + Linear SVM (4096 a.p.)	288.93	279.13	227.27
	TMF + Linear SVM (8192 a.p.)	536.30	493.52	433.70
	EP-NN (40 p.p.c, 20 base learners)	336.00	174.00	124.00
Others	Linear SVM (10 passes) [16]	8.75	-	-
	LL-SVM (100 a.p., 10 passes) [76]	470.00	-	-
	LIBSVM (RBF kernel) [24]	4.60×10^4	-	-
	LA-SVM (RBF kernel, 1 pass) [18]	4.06×10^4	-	-
	LA-SVM (RBF kernel, 2 passes) [18]	4.28×10^4	-	-

Chapter 5

Efficient Object Detection Framework

In this chapter, we present our efficient object detection framework on current personal computers. Our framework simply integrates the techniques for object proposal generation in Chapter 3 and proposal verification in Chapter 4 using HOG features, implemented using a mixture of MATLAB and C++ code. We test this framework in four applications: (1) VOC challenges, (2) traffic sign detection, (3) pedestrian detection, and (4) face detection. We report our performance in terms of accuracy and computational time in both training and testing, and compare them with other methods as well.

5.1 System Design

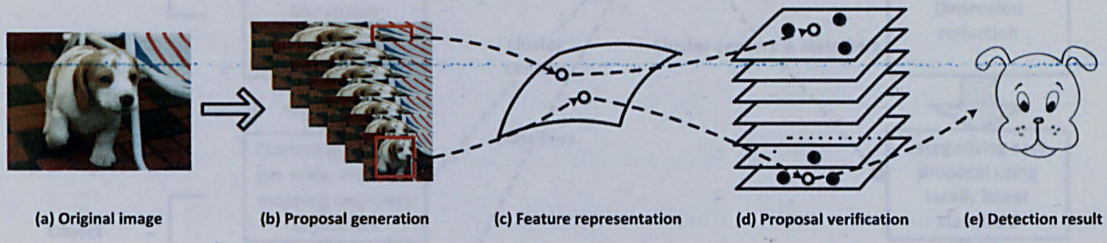


Figure 5.1: Illustration of our efficient object detection framework.

Fig. 5.1 illustrates our efficient object detection framework. Basically there are three modules inside: (1) object proposal generation module (PGM), (2) feature representation module (FRM), and (3) object proposal verification module (PVM). Fig. 5.2 shows all the steps in the implementation of our framework during training and testing.

5.1.1 Object Proposal Generation Module

In PGM our object proposal generation method explained in Chapter 3 is applied to learn the model parameters during training, which are used for generating proposals in both training and testing.

The major parameters in this module which impact the performance of the framework greatly are the overlap threshold parameter η (its equivalent parameter is the maximum number of quantized scales/aspect-ratios K, which is used in the

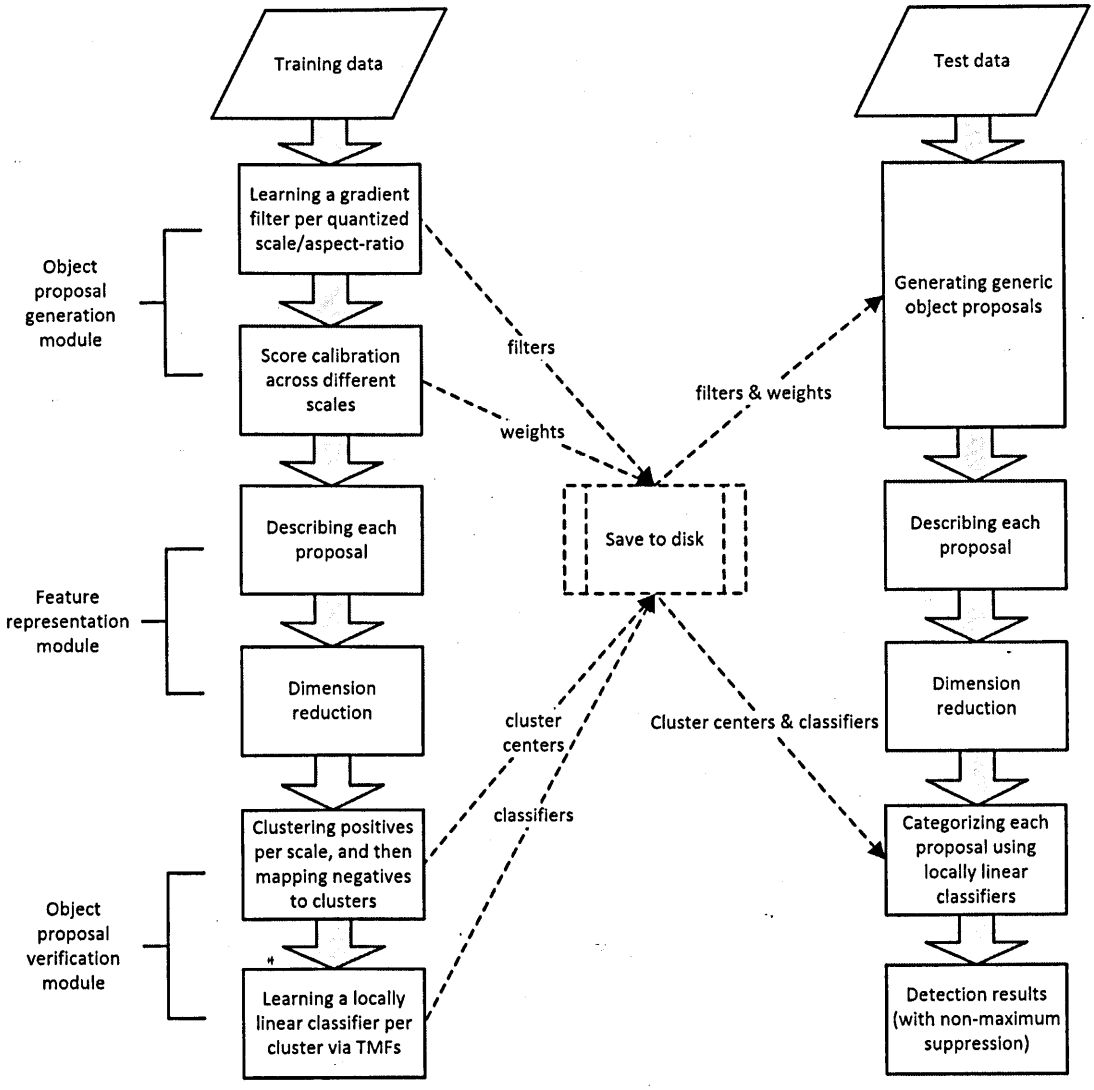


Figure 5.2: Work flow of our efficient object detection framework in both training and testing.

experiments) and the number of final proposals d_2, which are passed down to the next module. Other parameters are kept the same as described in Section 3.3.1.3 without specific explanation. We will investigate these two parameters further for the performance of our framework in the experiments.

5.1.2 Feature Representation Module

In FRM, each proposal is represented by certain feature descriptors, such as HOG, local binary pattern (LBP) [102], *etc.*. In our experiments we prefer HOG since it is widely used in object detection. Thus, there are two parameters for computing

HOG descriptors, that is, the size and number of cells in HOG. We fix the size of cells to 12 pixels without further tuning, and will investigate the number of cells in our experiments.

In order to handle the data noise as well as keeping the data compact, we further apply SVD to the feature descriptors for dimension reduction. We set the reduced dimension to 128 without further tuning because in [96] the effect of the dimension of the HOG space for separating data has been investigated. It turns out that this dimension reduction can also accelerate the computation of truncated marginal features (TMFs) for classification.

5.1.3 Object Proposal Verification Module

In PVM, during training we first cluster the positive data in each quantized scale/aspect-ratio and map negative data to clusters based on the minimum Euclidean distances between the negative data and the cluster centers in the 128-dim feature space. Then for each cluster a locally linear classifier is learned using TMFs as described in Section 4.2. The reason for clustering before learning classifiers is to generate better positive data for training classifiers so that the learned classifiers have better generalization. Similar technique has been used and demonstrated well for object detection in [61].

We utilize K-Means as our clustering method, and control the average number of positive data in each cluster, which is investigated later in the experiments. Another parameter that needs to be investigated is the number of anchor points (*i.e.* columns in the projection matrix $\mathbf{B}$) for constructing TMFs. We simplify the learning process for TMFs by setting the threshold vector $\mathbf{t}$ to the inner product between the projection matrix and the mean vector of the training data points.

5.2 Applications

In order to demonstrate that our framework is efficient and sufficiently general for different detection tasks with reasonable detection accuracies, we apply our framework to four detection tasks: (1) VOC object detection challenges, (2) traffic

sign detection, (3) pedestrian detection, and (4) face detection, where Task (1) and (2) are multiclass object detection, while Task (3) and (4) are specific object detection (*i.e.* binary classification problems). Our framework is implemented using a mixture of MATLAB and C++ code.

Our default computer is:

- Dell T3500 workstation, equipped with Xeon W3680@3.33GHz (6 cores, 12 threads) and 24GB DDR3 1333MHz memory.

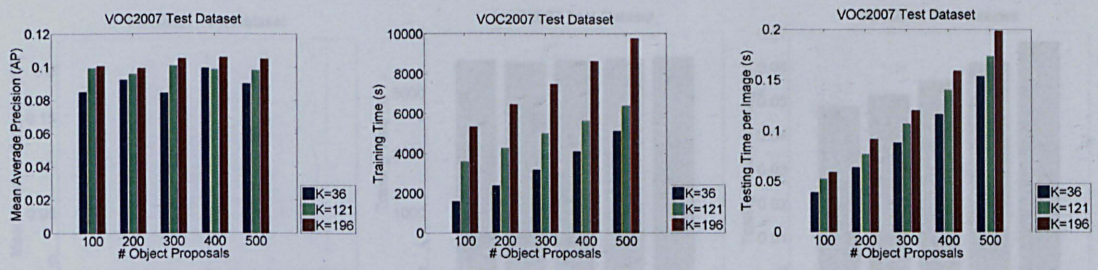
The default parameters for investigation are:

- In PGM, $K = 121$ and $d_2 = 100$;
- In FRM, the number of cells in HOG is set to 5;
- In PVM, the average number of positives per cluster is set to 2000, and the number of anchor points for TMFs is set to 2048;
- As our linear SVM solver, we employ LIBLINEAR as usual and set the parameter $C = 10$ for all the experiments without further tuning;
- We assume that the smallest and biggest object instances of interest in each dataset can be localized correctly by 16×16 and 512×512 pixel patches, respectively.

When we investigate one parameter, the rest are kept the same as the default values. We report our performance in terms of average precision (AP), training time, and testing time per image, respectively. The reported timing counts for all the steps for detection in both training and testing, starting from loading images, using all the threads in the CPU (12 threads).

5.2.1 VOC Challenges

From VOC2007 to VOC2012, each dataset contains 20 object categories, and the size of each image is around 300×500 (or 500×300) pixels. Except VOC2007, which consists of train/validation/test data with public annotation available, the rest have train/validation data with public annotation available only. In this



(a) (b) (c)

Figure 5.3: Performance comparison using different K and numbers of proposals on VOC2007 test dataset in terms of (a) mean average precision (AP), (b) training time, and (c) testing time per image.

task, first we will investigate the effects of different parameters on performance using VOC2007 and compare our results with some other methods, then list the results on the rest datasets from VOC2008 to VOC2012 with analysis.

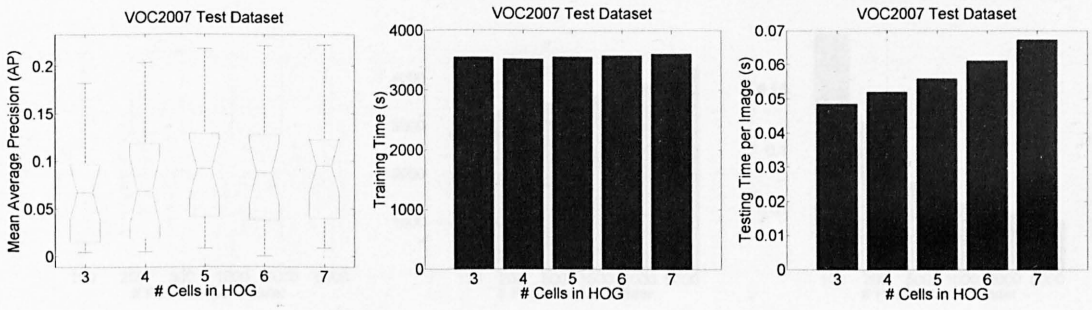
5.2.1.1 Object Proposal Generation Module

Fig. 5.3 shows the performance comparison using different maximum numbers of filters K learned at the first stage of the cascaded model, and numbers of final object proposals d_2 on VOC2007 test dataset. We can observe that:

- Increasing K does help improve mean AP. However, when K is beyond 121, the improvement is marginal. Moreover, it takes more time in both training and testing. We believe that with larger K, the overlap scores in correct proposals become larger, leading to better classifiers in the end due to better quality of positives.

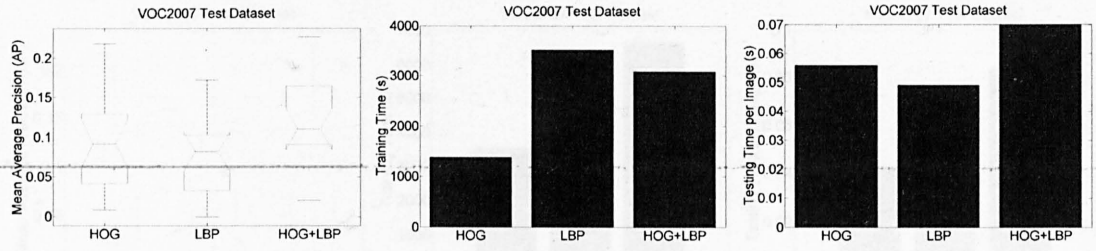
- Increasing d_2 does little help to improve mean AP, while it costs much more computation in both training and testing. Recall that in Fig. 3.12 our curves grow sub-linearly, which means the growing speed of negatives is larger than that of positives, when the number of proposals is beyond 100. Taking the error rates of classifiers as constants, we expect that the AP values should be performed like this:

- With increase of K or d_2, the computational time in both training and testing is growing linearly proportional to either K or d_2, which is consistent with our computational complexity analysis in Section 3.2.3.



(a) (b) (c)

Figure 5.4: Performance comparison using different numbers of cells in HOG on VOC2007 test dataset in terms of (a) mean average precision (AP), (b) training time, and (c) testing time per image.



(a) (b) (c)

Figure 5.5: Performance comparison using HOG, LBP, and HOG+LBP on VOC2007 test dataset in terms of (a) mean average precision (AP), (b) training time, and (c) testing time per image.

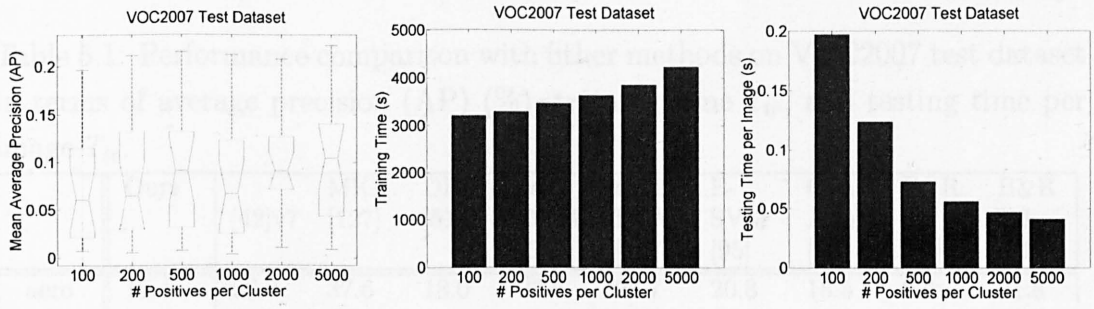
Notice that using $K = 36$ and $d_2 = 100$, which is the fastest setting in Fig. 5.3, our method can process a test image within ***0.039*** second on average and achieve 8.5% mean AP.

5.2.1.2 Feature Representation Module

Fig. 5.4 shows the effect of different numbers of cells in HOG on our performance. When this number is beyond 5, mean AP and training time change little, while testing time increases noticeably, since computing HOG features needs more time.

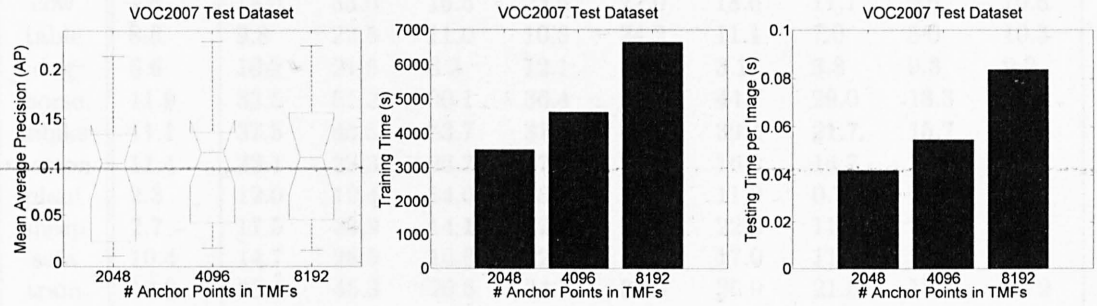
We also test our method using HOG, LBP [1], and HOG+LBP, as shown in Fig. 5.5. As we see, in terms of mean AP, HOG is slightly better than LBP, and their combination is better than both. For training time, LBP needs much more time than HOG, and slightly more than HOG+LBP. For testing time, LBP needs

[1]We employ the VLFeat library [126] to compute HOG and LBP, respectively.



(a) (b) (c)

Figure 5.6: Performance comparison by varying the average number of positive data points per cluster on VOC2007 test dataset in terms of (a) mean average precision (AP), (b) training time, and (c) testing time per image.



(a) (b) (c)

Figure 5.7: Performance comparison using different numbers of anchor points in truncated marginal features (TMFs) on VOC2007 test dataset in terms of (a) mean average precision (AP), (b) training time, and (c) testing time per image.

the least, while HOG+LBP needs the most, which is very reasonable considering their feature calculation time. This experiment suggests that we can potentially improve mean AP by adding more different types of features into FRM, similar to [127]. Consequently it requires more computational time during testing.

5.2.1.3 Object Proposal Verification Module

Fig. 5.6 shows the performance comparison on VOC2007 by varying the average number of positive data points per cluster. In general, with the increase of the number, mean AP is changing a little, training time is growing slowly, and testing time is dropping significantly due to the reduction of the number of clusters.

Fig. 5.7 shows the performance comparison using different numbers of anchor

Table 5.1: Performance comparison with other methods on VOC2007 test dataset in terms of average precision (AP) (%), training time T_{tr}, and testing time per image T_{te}.

	Ours	[42]v7	MKL [127]	DPM [51]	cas. DPM [52]	c.t.f. DPM [107]	E-SVM [95]	Gau. App. [96]	B&R T1 [84]	B&R T6 [84]
aero	15.4	26.2	37.6	18.0	22.8	27.7	20.8	18.5	9.6	21.8
bike	11.5	40.9	47.8	41.1	49.4	54.0	48.0	38.0	12.8	23.2
bird	9.1	9.8	15.3	9.2	10.6	6.6	7.7	1.06	2.3	2.9
boat	0.5	9.4	15.3	9.8	12.9	15.1	14.3	10.5	3.1	9.8
bottle	9.1	21.4	21.9	24.9	27.1	14.8	13.1	12.7	1.1	9.1
bus	12.8	39.3	50.7	34.9	47.4	44.2	39.7	37.0	11.4	20.3
car	20.8	43.2	50.6	39.6	50.2	47.3	41.1	37.4	16.3	23.0
cat	8.3	24.0	30.0	11.0	18.8	14.6	5.2	11.4	11.7	18.1
chair	4.5	12.8	17.3	15.5	15.7	12.5	11.6	10.3	9.1	9.4
cow	1.5	14.0	33.0	16.5	23.6	22.0	18.6	11.7	9.5	10.8
table	8.6	9.8	22.5	11.0	10.3	24.2	11.1	7.0	5.0	10.3
dog	5.6	16.2	21.5	6.2	12.1	12.0	3.1	3.8	9.3	9.2
horse	11.9	33.5	51.2	30.1	36.4	52.0	44.7	29.0	18.3	30.0
mbike	11.1	37.5	45.5	33.7	37.1	42.0	39.4	21.7	15.7	28.9
person	11.4	22.1	23.3	26.7	37.2	31.2	16.9	14.7	10.0	11.6
plant	2.3	12.0	12.4	14.0	13.2	10.6	11.2	0.7	0.1	1.5
sheep	2.7	17.5	23.9	14.1	22.6	22.9	22.6	11.3	2.3	10.3
sofa	10.4	14.7	28.5	15.6	22.9	18.8	17.0	11.8	5.5	13.6
train	19.3	33.4	45.3	20.6	34.7	35.3	36.9	21.5	15.4	24.9
tv	13.1	28.9	48.5	33.6	40.0	31.1	30.0	27.9	10.7	15.6
AVE.	9.5	23.3	31.2	21.3	27.3	26.9	22.7	17.2	9.0	15.2
T_{tr}	1.0(h)	-	-	-	-	-	-	≈4(h)	-	-
T_{te}	0.05(s)	-	67(s)	2(s)	<1(s)	<1(s)	-	≈1(h)	≈20(s)	≈20(s)

points in TMFs on VOC2007 [2]. Increasing this number has little impact on mean AP, but incurs more time in both training and testing. The growing speed in computational time is roughly linear to the differences between the numbers, which is consistent with the computational complexity analysis in Section 4.4.2.

5.2.1.4 Performance Comparison

We compare our performance using the default computer and the default parameters with some other methods. As listed in Table 5.1, compared with other methods our method achieves reasonable AP on each class in VOC2007, with much faster training and test speed. This is exactly the purpose that our method would like to accomplish. However, our AP is still lower than most of the methods

[2]Due to the memory limit of our default computer, we ran this experiment on a server equipped with 2×Xeon X5560@2.80GHz (8 cores, 8 threads) and 96GB DDR3 1333MHz memory. Therefore, the timing shown in Fig. 5.7 cannot be compared with others directly.

Table 5.2: Performance comparison on different VOC train/validation datasets in terms of average precision (AP) (%), training time T_{tr}, and testing time per image T_{te}.

	VOC2008 [43]	VOC2009	VOC2010	VOC2011 [46]	VOC2012
aero	4.6	18.2	22.5	21.5	17.2
bike	3.0	12.0	13.9	12.1	13.1
bird	3.3	1.6	2.0	4.5	9.1
boat	3.0	0.8	4.5	6.1	6.1
bottle	9.1	9.1	9.1	9.1	4.5
bus	13.4	26.4	26.1	33.4	28.5
car	9.1	11.5	13.6	13.0	12.9
cat	4.8	12.6	20.1	17.9	17.1
chair	0.2	0.8	3.0	9.1	9.1
cow	0.60	0.5	3.6	1.5	1.5
table	0.8	1.8	9.1	9.1	0.7
dog	2.8	10.6	13.0	7.7	9.2
horse	2.3	5.3	4.2	10.2	3.9
mbike	9.1	7.3	13.8	14.0	14.1
person	10.8	10.9	13.5	10.0	13.7
plant	0.3	0.5	6.1	1.8	1.3
sheep	0.0	1.5	9.1	9.1	9.1
sofa	3.0	4.0	3.8	4.5	9.1
train	8.3	16.3	15.7	15.9	13.4
tv	9.1	12.0	12.0	13.6	13.4
AVE.	4.9	8.2	10.9	11.2	10.4
T_{tr}	0.7(h)	0.9(h)	1.1(h)	1.2(h)	1.2(h)
T_{te}	0.04(s)	0.05(s)	0.05(s)	0.05(s)	0.05(s)
# Train img.	2111	3473	4998	5717	5717
# Test img.	2221	3581	5105	5823	5823

listed in Table 5.1, which suggests that the object proposal verification module still has a lot of room for improvement. The major reason for low AP is the large-scale extremely unbalanced data generated from the object proposal generation module. The ratio between negatives (*i.e.* wrong object detection proposals) and positives (*i.e.* correct object detection proposals) in the proposal is roughly 100:1. In this situation, our classifier tends to misclassify the positives, which results in the low AP. We also tested our method by replacing the whole proposal verification module with simple linear SVMs, and we found that our current AP is about 1% better. Therefore, how to handle this very challenging large-scale extremely unbalanced data is still a big problem, and it will be in our future work.

We also ran our code on the rest VOC datasets, except VOC2006, using the

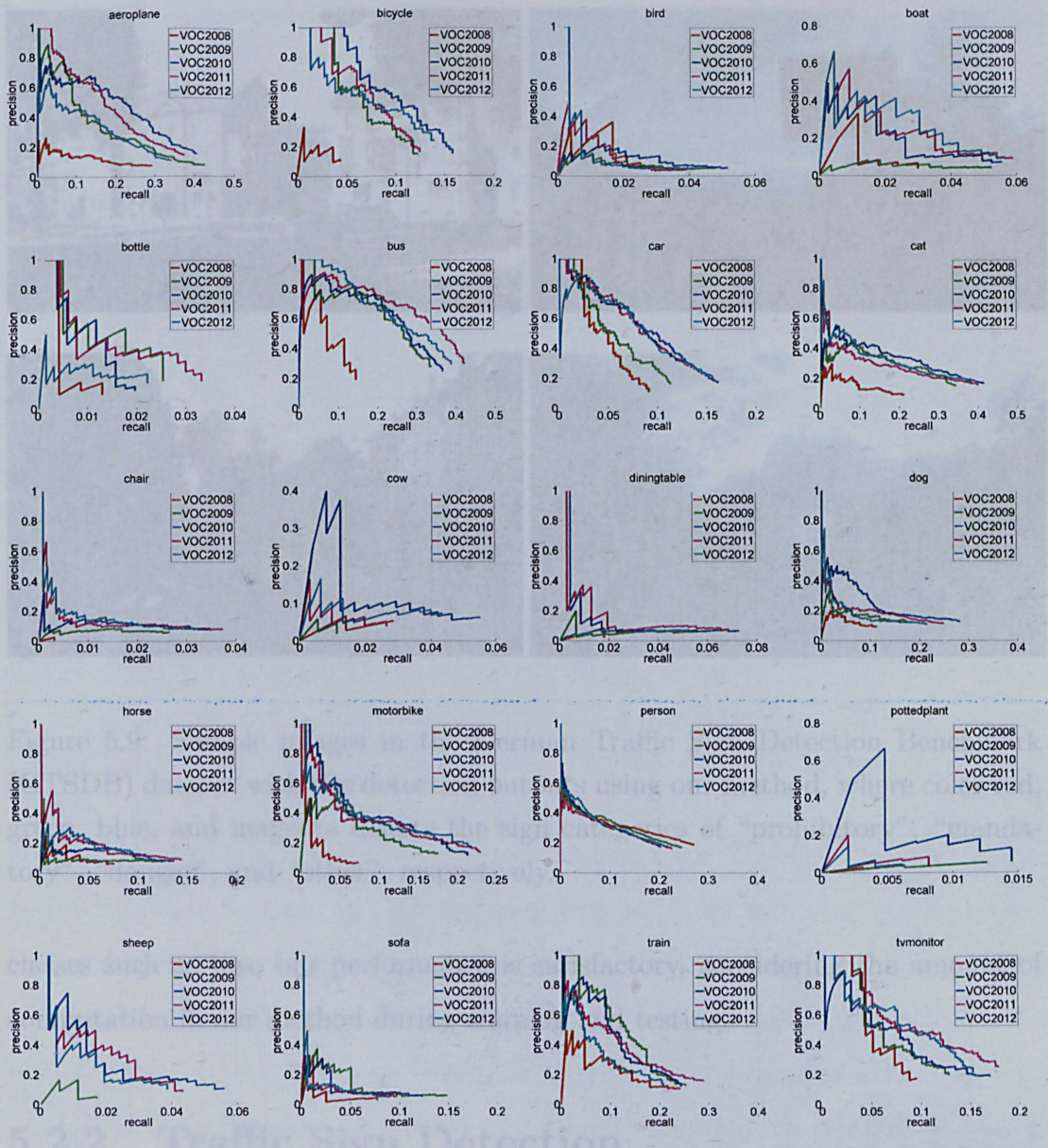


Figure 5.8: AP comparison per class on different VOC train/validation datasets.

training data as the training set and the validation data as the test set, and list our results in Table 5.2. With more training images, it seems that our method performs better with more training time as well. However, the testing time per image changes little. In Fig. 5.8, we show the AP comparison per class between different VOC train/validation datasets. As we see, for each class, the curves have similar behavior. For the difficult classes such as bird, boat and bottle, our recall is very low, which means that either our proposal method fails to localize the object instances or our classifier fails to verify the proposals correctly, or both. This leaves us large room for improvement. For those relatively simple

Figure 5.9: Sample images in the German Traffic Sign Detection Benchmark (GTSDB) dataset with the detection outputs using our method, where color red, green, blue, and magenta denote the sign categories of "prohibitory", "mandatory", "danger", and "other", respectively.

classes such as bus, our performance is satisfactory, considering the amount of computation in our method during training and testing.

5.2.2 Traffic Sign Detection

For this task, we utilize the German Traffic Sign Detection Benchmark (GTSDB) dataset [66]. This dataset has 600 training images with annotations available, and 300 test images without public annotations yet. The size of each image is 1360 × 800 pixels, and the sizes of traffic signs in the images vary from 16 × 16 to 128 × 128. There are four different sign categories, *i.e.* "prohibitory", "mandatory", "danger", and "other". Notice that in this dataset, some images do not contain any traffic sign.

To evaluate our method, we adopt 10-fold cross validation using the training data only. That is, we manually divide the training data into 10 folds in order, step by 10, each of which contains 60 images (*i.e.* Image 1, 11, 21, $\cdots$, 591 are

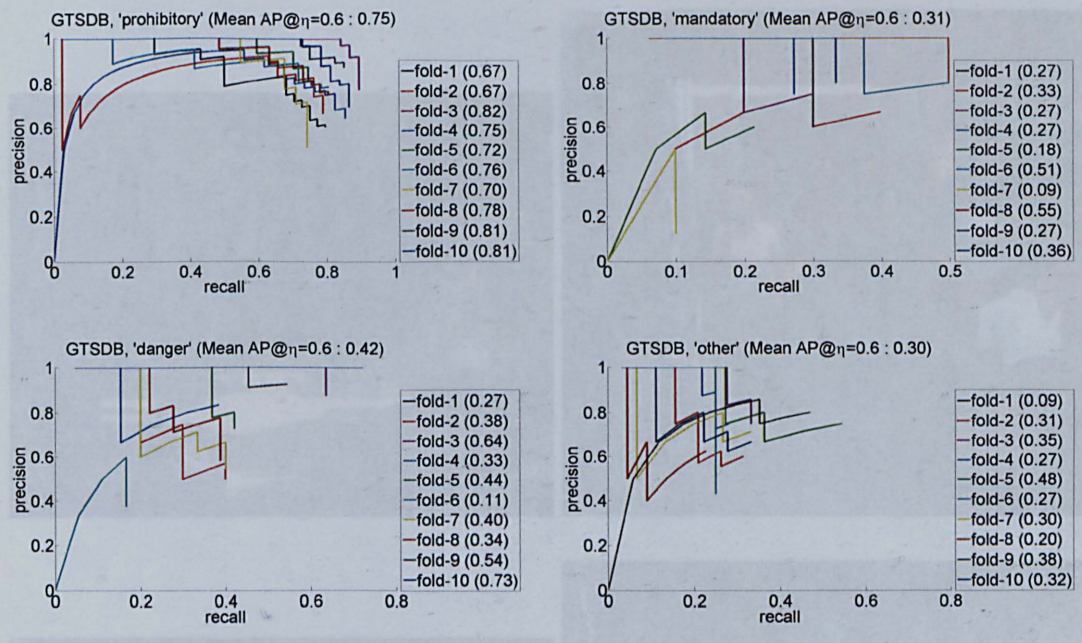


Figure 5.10: Precision-recall curves of the 10-fold cross validation on the categories of "prohibitory", "mandatory", "danger", and "other" in GTSDB, respectively. The numbers in the brackets are average precision (AP) values.

in Fold 1, and so on). We leave one fold as test data, and use the rest 9 folds as training data to train the multiclass object detectors using our method. We report the 10 precision-recall curves and AP's for evaluation.

Fig. 5.9 shows some sample images from the dataset, associated with the detection outputs of our method. Due to the large sizes of images and the relatively small sizes of traffic signs, we modify the default setting of our method: For each image, we allow PGM to generate 5000 object proposals per image at most. As we see, the lighting condition in the images varies a lot, and the background is very noisy, making the detection task difficult.

We show the precision-recall curves over the 10-fold cross validation for each category in Fig. 5.10. From this figure, we can see that the images containing the traffic signs in category "mandatory", "danger", and "other" are very few based on the behaviors of their corresponding sub-figures, which may result in training classifiers improperly. To give a rough view of how well our method works on this dataset, we simply compare each mean AP with the results on the website `http://benchmark.ini.rub.de/?section=gtsdb&subsection=results`. For the category "prohibitory", "mandatory", and "danger", our results outperform 24, 2, 8

Figure 5.11: Sample images in the Penn-Fudan Pedestrian Detection dataset with the detection outputs using our method, where the dashed bounding boxes with red color denote the ground-truth of pedestrians, and the solid bounding boxes with green color denote our detection outputs.

out of 52, 34, 28 results in the lists, respectively. Notice that the overlap threshold for the test is set to $\eta = 0.6$. As for computational time, our code finishes training within 1.4 hours, and processes a test image within 20 seconds, on average.

5.2.3 Pedestrian Detection

For this task, we utilize the Penn-Fudan Pedestrian Detection database [132]. The images in the dataset are taken from scenes around campus and urban streets, and each image will contain at least one pedestrian. The sizes of images are around 500×500 pixels, and the heights of labeled pedestrians in this database fall into [180,390] pixels. All labeled pedestrians are straight up. In total there

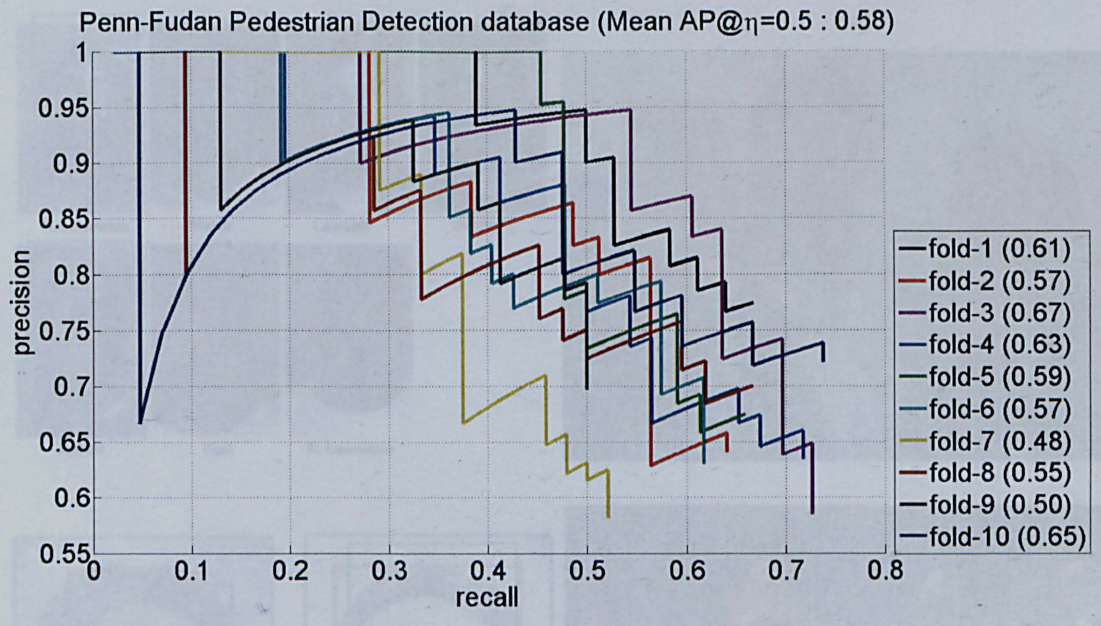


Figure 5.12: Precision-recall curves over the 10-fold cross validation on the Penn-Fudan Pedestrian Detection dataset. The numbers in the brackets are average precision (AP) values.

are 170 images with 345 labeled pedestrians, among which 96 images are taken from around University of Pennsylvania, and the remaining 74 are taken from around Fudan University.

Similar to the traffic sign detection task, we adopt 10-fold cross validation to test our method. In detail, we manually sample the images in order, step by 17, to create 10 different folds, 9 of which are used as training data, and the rest as test data. We perform specific object detection (*i.e.* binary classification) using our method, and report the 10 precision-recall curves for evaluation.

Fig. 5.11 shows some sample images from the dataset with the detection outputs using our method. The pedestrians in the dataset appear in different scenarios, face different directions, have partial occlusion, *etc.* We use the default setting of our code for detection.

Fig. 5.12 shows the precision-recall curves over the 10-fold cross validation. The behavior of each curve is similar to each other in general, producing similar AP's. Training our method on this dataset needs 2 minutes, and processing a test image requires 0.15 second, on average.

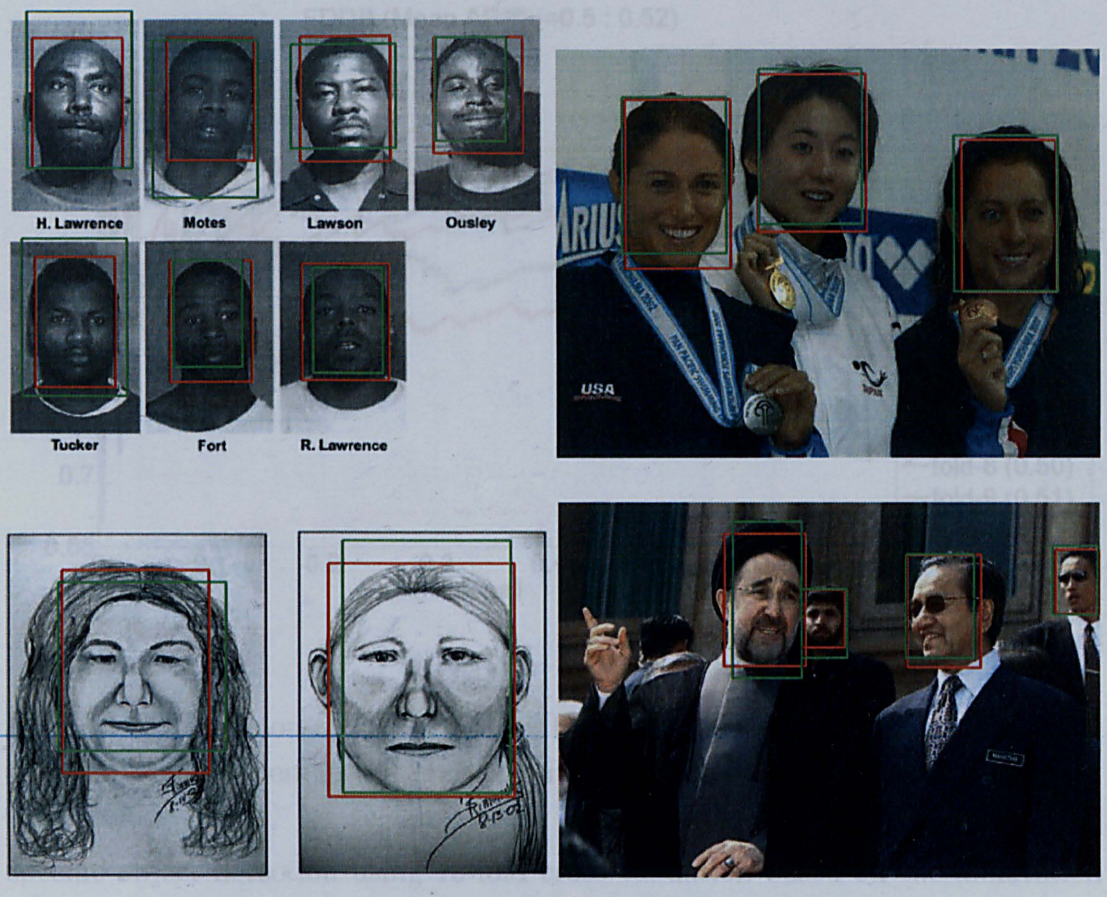


Figure 5.13: Sample images in the Face Detection Data Set and Benchmark (FDDB) dataset with the detection outputs using our method, where the red bounding boxes denote the ground-truth of faces, and the green bounding boxes denote our detection outputs.

5.2.4 Face Detection

For this task, we utilize the Face Detection Data Set and Benchmark (FDDB) [67] from University of Massachusetts, a dataset of face regions designed for studying the problem of unconstrained face detection. This dataset contains the annotations for 5171 faces in a set of 2845 images taken from the Faces in the Wild [10] dataset. The sizes of images are roughly 300×450 (or 450×300) pixels.

Each face in the dataset is annotated as an ellipse with 5 parameters. In order to make it suitable for our bounding box based detection method, we sample 300 points on each ellipse uniformly, and localize the leftmost, rightmost, topmost, and bottommost points among these points to draw a ground-truth bounding box. Our evaluation is based on these ground-truth bounding boxes. Still we perform

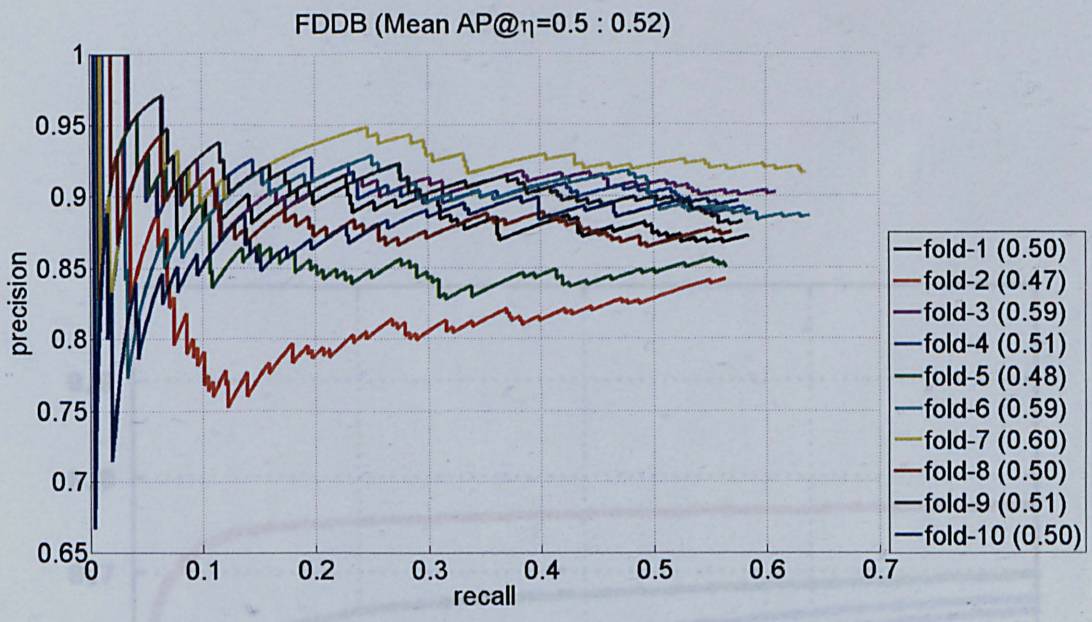


Figure 5.14: Precision-recall curves over the 10-fold cross validation on the FDDB dataset. The numbers in the brackets are average precision (AP) values.

specific object detection using 10-fold cross validation based on the annotation files in the dataset, and report the 10 precision-recall curves for evaluation. To get better accuracy, we allow PGM to generate 2000 proposals per image, and the rest parameters are set to the default values.

Fig. 5.13 shows some sample images from the dataset with detection outputs using our method. These faces may have different lighting conditions, expressions, views, sizes, angles, truncation, and occlusion, *etc.* Even some sketch faces are included.

We show the precision-recall curves over the 10-fold cross validation using this dataset in Fig. 5.14. Like in the other tasks, the behaviors of the curves perform similarly, which demonstrates again that our method is quite robust. Further, we compare our performance with some other face detection methods and show the results in Fig. 5.15. As we see, although our method is not designed particularly for face detection, it still can achieve comparable performance to the rest. Training our method using this dataset needs 1 hour, and processing a test image requires 6 seconds, on average.

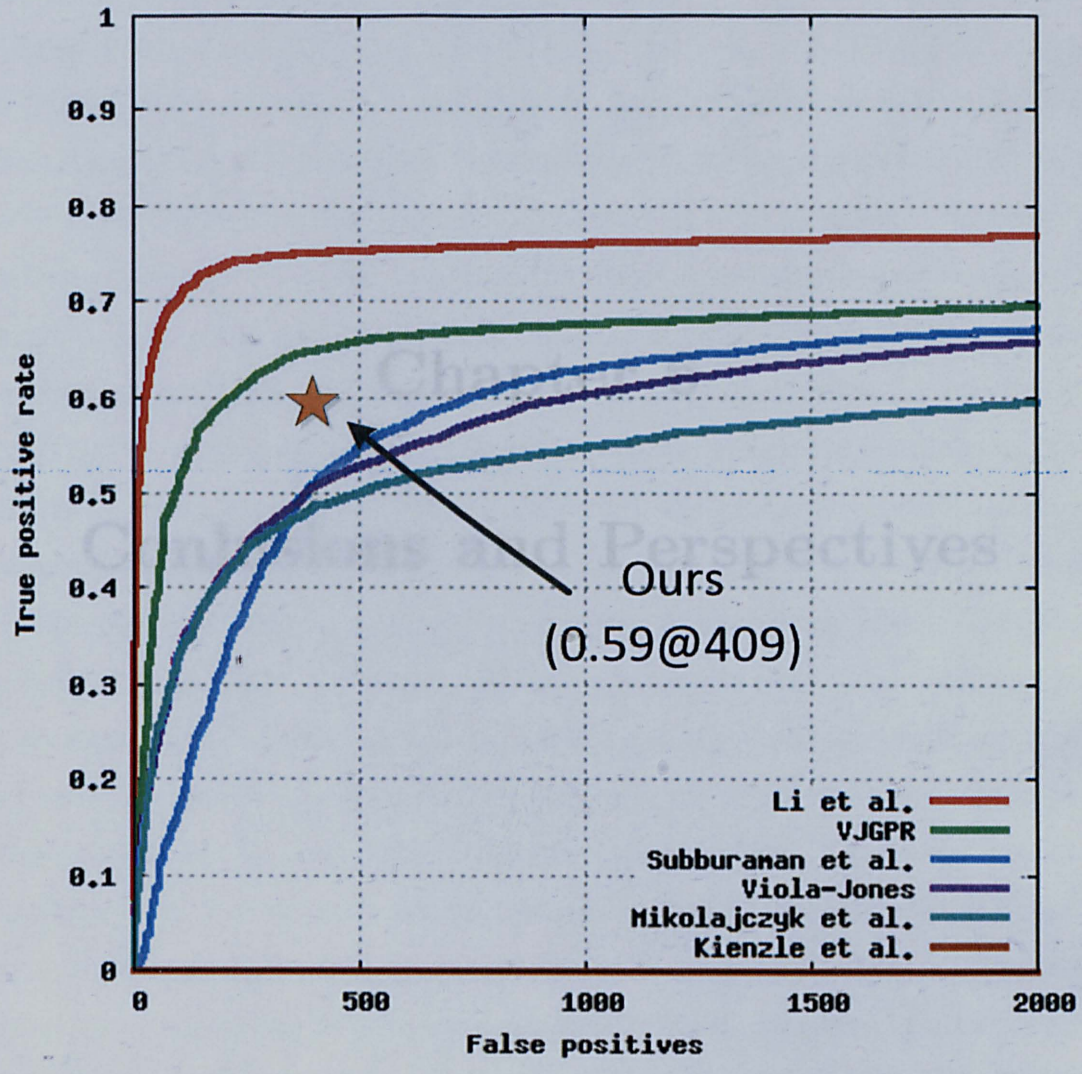


Figure 5.15: Performance comparison with some other methods on the FDDB dataset. Over the 10-fold cross validation, our method covers 59% faces in the dataset, and has 409 false positives in total. The other results in the figure are taken from `http://vis-www.cs.umass.edu/fddb/results.html`.

Chapter 6

Conlusions and Perspectives

In this thesis, we presented an efficient object detection framework based on the sliding window strategy, which consists of three modules: object proposal generation module (PGM), feature representation module (FRM), and object proposal verification module (PVM). We studied the influence of PGM and PVM on the detection performance in terms of detection accuracy and computational efficiency, respectively.

First, we formulated the object proposal generation problem as a structured learning problem and investigated structural support vector machines (SSVMs) with linear kernels for this problem. In particular, a scale/aspect-ratio quantization scheme was proposed to reduce the search space of bounding boxes, and introduced into SSVMs with ranking constraints as well, so that the proposal ranking orders based on the margins can fit the expected ranking orders predefined by the ranking constraints. In order to solve our structured learning problem efficiently with controllable classification errors, we further proposed a ranking-order decomposition algorithm, which decomposes the original problem into some sub-problems, which can be solved more easily and efficiently with much fewer constraints, and utilizes their solutions to approximate the solution of the original problem. In such way, we proved that the loss of the original problem can be upper-bounded and lower-bounded by the losses of the sub-problems. To apply this general algorithm to our proposal generation problem, a two-stage cascade method was proposed, whose computational complexity is linearly proportional to the sizes of images and the sizes and number of filters, in general, but independent of the numbers of output object proposals and object categories. We tested our proposal generation method on VOC object detection challenges for specific and generic object proposal generation tasks. Our experiments demonstrated that with simple image gradients as features, our object proposal generation method achieves state-of-the-art results at a very low cost in computation. Compared with some other methods, ours can achieve slightly better object recall given a number of object proposals, but run 10 times faster.

Second, we proposed two locally linear and one locally nonlinear classifiers for object proposal verification. Our first locally linear classifier came from orthogonal coordinate coding (OCC) with locally linear SVMs (LL-SVMs). By

encoding high dimensional data using a few anchor planes defined by orthogonal basis vectors, OCC can help LL-SVMs to approximate the nonlinear decision boundary in the feature space better. The functionality of basis vectors in OCC can be considered as weak learners, based on which we proposed a second locally linear classifier using truncated marginal features (TMFs) and traditional multiclass linear SVMs. TMFs are generated by an explicit nonlinear map function which performs the data localization in a supervised manner. Learning the map function as well as the classifiers based on the large-margin criterion can be considered as a biconvex minimization problem, which can be solved efficiently and locally by alternating convex search where random projection and stochastic sub-gradient descend are used to learn the parameters in TMFs, and a linear multiclass SVM solver is used to learn the locally linear classifiers. This method is in favor of sparse features while improving the classification performance, and it can be used to approximate the summation of nonlinear arc-cosine kernels. Beyond the locally linear classifiers, we proposed a third locally nonlinear classifier, Parametric Nearest Neighbor (P-NN), and its extension Ensemble of P-NN (EP-NN). This method extends the analysis of Gaussian kernel density estimation, and attempts to learn the prototypes for nearest neighbor search and the classifier parameters jointly and discriminatively. Eventually the minimum weighted squared Euclidean distances between the data and the prototypes are utilized to construct the nonlinear decision boundaries locally in the original feature space. The computational complexity of these three local classifiers is similar to that of linear SVMs in both training and testing. Therefore, they can be trained and tested very efficiently. Our experiments demonstrated that all of these classifiers can achieve classification accuracies very close to those of kernel SVMs with good stability and insensitivity to parameter changes within a wide range, but run much faster than kernel SVMs in both training and testing, which is consistent with our computational complexity analysis.

Based on our proposal generation method and local classifiers, we presented our efficient object detection framework. Specifically we chose the locally linear classifier of TMFs with multiclass linear SVMs for PVM, due to its good performance in terms of accuracy and efficiency. We demonstrated the efficiency

and generality of our framework by applying it to four different object detection tasks, that is, VOC detection challenges, traffic sign detection, pedestrian detection, and face detection. In each task, our method can perform reasonably well with acceptable detection accuracy and very good computational efficiency. For instance, on VOC detection challenges, our method can achieve about 0.1 mean AP within 2 hours of training and 0.05 seconds of testing a 500×300 pixel image using a mixture of MATLAB and C++ code on a computer equipped with an Xeon W3680@3.33GHz processor and 24GB 1333MHz memory.

Compared with state-of-the-art detection methods, our framework still has large room for improvement in terms of detection accuracy. In order to improve our framework from the view of learning models, we would like to consider the following aspects as our future work:

Relational Dependency Networks (RDNs) [101]. In object proposal generation, we learned SSVMs with ranking constraints, and proposed a ranking-order decomposition algorithm to solve it efficiently and approximately with guaranteed error bounds. The ranking list is actually a special and simple relational dependency network (RDN), and in our proposal generation problem, it can be replaced with any tree-structured RDN (without loop inside). Then similar decomposition techniques can be used to solve the structured learning problems efficiently and approximately by solving the master problems with much fewer parameters and slave problems with much fewer constraints and parameters, and still the losses of the original problems can be easily bounded by the losses of these sub-problems. Using RDNs, we can describe the contents of images with much richer information such as locations [35] and contexts [36] of objects, relations between parts of objects [34], *etc.*

Deep Learning in Feature Representation [81]. In both PGM and PVM, data points are needed to be represented well. In PGM, the features should be general enough to cover "all" possible objects for object/non-object ranking or classification. In PVM, the features should be discriminative enough for distinguishing different object classes and background. In our framework, we represent

the patches in each image independently with each other, which is not always true (at least not true for some neighbor patches between different layers in the image pyramid). Using deep learning, features can be generated automatically based on the learning requirement in a hierarchical way to incorporate information in images, and this may be very useful in our framework for both ranking and classification. However, to preserve the efficiency of our framework in both training and testing, it will be a challenge to integrate deep learning, and how to apply deep learning for object detection properly is still an open issue.

Local Classifiers with Multiple Instance Learning. Latent SVMs can be considered as linear SVMs with multiple instance learning. Kernelized latent SVMs [138] have recently been successfully applied in object recognition and localization, and their performance is better than latent SVMs. Since our local classifiers can approximate kernel SVMs using explicit nonlinear map functions and linear SVMs, intuitively we can extend our methods to learn local classifiers in the context of multiple instance learning to "kernelize" latent SVMs approximately. We expect that the computational complexity of the new local classifiers should be similar to that of latent SVMs. The major difficulties in the extension are how to learn the anchor points, which should be sets of feature vectors, and how to design the explicit map functions.

Bibliography

[1] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2189–2202, 2012.

[2] Senjian An, Patrick Peursum, Wanquan Liu, and Svetha Venkatesh. Efficient subwindow search with submodular score functions. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1409–1416, 2011.

[3] Senjian An, Patrick Peursum, Wanquan Liu, Svetha Venkatesh, and Xiaoming Chen. Exploiting monge structures in optimum subwindow search. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 926–933, 2010.

[4] Xiang Bai, Quannan Li, Longin Jan Latecki, Wenyu Liu, and Zhuowen Tu. Shape band: A deformable object detection approach. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1335–1342, 2009.

[5] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming Theory and Algorithms.* John Wiley, New York, 1993.

[6] Régis Behmo, Paul Marcombes, Arnak Dalalyan, and Véronique Prinet. Towards optimal naive bayes nearest neighbor. In *Proceedings of European Conference on Computer Vision*, pages 171–184, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:509–522, 2001.

[8] Alexander C. Berg, Tamara L. Berg, and Jitendra Malik. Shape matching and object recognition using low distortion correspondences. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 26–33, 2005.

[9] Alexander C. Berg and Jitendra Malik. Geometric blur for template matching. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 607–614, 2001.

[10] Tamara L. Berg, Alexander C. Berg, Jaety Edwards, and David A. Forsyth. Who's in the picture. In *Proceedings of Advances in Neural Information Processing Systems*, pages 137–144, 2005.

[11] I. Biederman. Visual object recognition. *An Invitation to Cognitive Science*, 2:112–165, 1995.

[12] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[13] M.B. Blaschko and C.H. Lampert. Learning to localize objects with structured output regression. In *Proceedings of European Conference on Computer Vision*, pages I: 2–15, 2008.

[14] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Extracting 3d scene-consistent object proposals and depth from stereo images. In *Proceedings of the 12th European conference on Computer Vision - Volume Part V*, Proceedings of European Conference on Computer Vision, pages 467–481, Berlin, Heidelberg, 2012. Springer-Verlag.

[15] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[16] Antoine Bordes, Léon Bottou, and Patrick Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10:1737–1754, December 2009.

[17] Antoine Bordes, Léon Bottou, Patrick Gallinari, and Jason Weston. Solving multiclass support vector machines with larank. In *Proceedings of International Conference on Machine Learning*, pages 89–96, 2007.

[18] Antoine Bordes, Seyda Ertekin, Jason Weston, and Léon Bottou. Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619, December 2005.

[19] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[20] David M. Bradley and J. Andrew Bagnell. Differential Sparse Coding. In *Proceedings of Advances in Neural Information Processing Systems*, pages 113–120, 2009.

[21] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, October 2001.

[22] John Canny. Finding edges and lines in images. Technical report, Cambridge, MA, USA, 1983.

[23] J. Carreira and C. Sminchisescu. CPMC: Automatic Object Segmentation Using Constrained Parametric Min-Cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1312–1328, July 2012.

[24] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[25] Youngmin Cho and Lawrence Saul. Kernel methods for deep learning. In *Proceedings of Advances in Neural Information Processing Systems*, pages 342–350. 2009.

[26] Youngmin Cho and Lawrence K. Saul. Analysis and extension of arc-cosine kernels for large margin classification. *ACM Computing Research Repository*, abs/1112.3712, 2011.

[27] O. Chum and A. Zisserman. An exemplar model for learning object classes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[28] Andrew Cotter, Shai Shalev-Shwartz, and Nathan Srebro. Learning optimally sparse support vector machines. In *Proceedings of International Conference on Machine Learning*, 2013.

[29] Koby Crammer, Ran Gilad-Bachrach, Amir Navot, and Naftali Tishby. Margin analysis of the LVQ algorithm. In *Proceedings of Advances in Neural Information Processing Systems*, pages 462–469, 2002.

[30] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, March 2002.

[31] G. Csurka, C. Bray, C. Dance, and L. Fan. Visual categorization with bags of keypoints. In *Proceedings of European Conference on Computer Vision Workshop on Statistical Learning in Computer Vision*, pages 1–22, 2004.

[32] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition - Volume 1 - Volume 01*, pages 886–893, 2005.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[34] Chaitanya Desai and Deva Ramanan. Detecting actions, poses, and objects with relational phraselets. In *Proceedings of European Conference on Computer Vision*, pages 158–172, 2012.

[35] Chaitanya Desai, Deva Ramanan, and Charless Fowlkes. Discriminative models for multi-class object layout. In *Proceedings of International Conference on Computer Vision*, pages 229–236, 2009.

[36] Santosh Kumar Divvala, Derek Hoiem, James Hays, Alexei A. Efros, and Martial Hebert. An empirical study of context in object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1271–1278, 2009.

[37] Piotr Dollár, Serge Belongie, and Pietro Perona. The fastest pedestrian detector in the west. In *Proceedings of British Machine Vision Conference*, pages 1–11, 2010.

[38] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743–761, 2012.

[39] J Eichhorn and O Chapelle. Object categorization with SVM: kernels for local features. Technical Report 137, Max Planck Institute for Biological Cybernetics, Tübingen, Germany, 7 2004.

[40] Ian Endres and Derek Hoiem. Category independent object proposals. In *Proceedings of European Conference on Computer Vision*, pages 575–588, 2010.

[41] Andreas Ess, Konrad Schindler, Bastian Leibe, and Luc Van Gool. Object detection and tracking for autonomous navigation in dynamic environments. *International Journal of Robotics Research*, 29(14):1707–1725, December 2010.

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

[43] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 (VOC2008) Results. http://www.pascal-network.org/challenges/VOC/voc2008/workshop/index.html.

[44] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2009 (VOC2009) Results. http://www.pascal-network.org/challenges/VOC/voc2009/workshop/index.html.

[45] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2010 (VOC2010) Results. http://www.pascal-network.org/challenges/VOC/voc2010/workshop/index.html.

[46] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html.

[47] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[48] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

[49] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[50] Ali Farhadi, Ian Endres, Derek Hoiem, and David A. Forsyth. Describing objects by their attributes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1778–1785, 2009.

[51] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[52] Pedro F. Felzenszwalb, Ross B. Girshick, and David A. McAllester. Cascade object detection with deformable part models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2241–2248, 2010.

[53] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Pictorial structures for object recognition. *International Journal of Computer Vision*, 61(1):55–79, January 2005.

[54] V. Ferrari, L. Fevrier, F. Jurie, and C. Schmid. Groups of adjacent contour segments for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(1):36–51, January 2008.

[55] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach.* Prentice Hall Professional Technical Reference, 2002.

[56] D. M. Gavrila and V. Philomin. Real-time object detection using distance transforms. In *Proceedings of Intelligent Vehicles Conference*, page 998, 1998.

[57] K. Goh, L. Turan, M. Safonov, G. Papavassilopoulos, and J. Ly. Biaffine matrix inequality properties and computational methods. In *Proceedings of the American Control Conference*, pages 850–855, Albuquerque, NM, June 1994.

[58] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.).* Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[59] Jochen Gorski, Frank Pfeuffer, and Kathrin Klamroth. Biconvex sets and optimization with biconvex functions - a survey and extensions. *Mathematical Methods of Operations Research*, 66(3):373–407, 2007.

[60] Stephen Gould, Paul Baumstarck, Morgan Quigley, Andrew Y. Ng, and Daphne Koller. Integrating visual and range data for robotic object detection. In *Proceedings of European Conference on Computer Vision Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications (M2SFA2)*, 2008.

[61] Chunhui Gu, Pablo Andrés Arbeláez, Yuanqing Lin, Kai Yu, and Jitendra Malik. Multi-component models for object detection. In *Proceedings of European Conference on Computer Vision*, pages 445–458, 2012.

[62] Sam Hare, Amir Saffari, and Philip H. S. Torr. Struck: Structured output tracking with kernels. In *Proceedings of International Conference on Computer Vision*, pages 263–270, 2011.

[63] Bharath Hariharan, Jitendra Malik, and Deva Ramanan. Discriminative decorrelation for clustering and classification. In *Proceedings of European Conference on Computer Vision*, 2012.

[64] Geremy Heitz and Daphne Koller. Learning spatial context: Using stuff to find things. In *Proceedings of European Conference on Computer Vision*, pages 30–43, 2008.

[65] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Putting objects in perspective. *International Journal of Computer Vision*, 80(1):3–15, 2008.

[66] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks (submitted)*, 2013.

[67] Vidit Jain and Erik Learned-Miller. FDDB: A benchmark for face detection in unconstrained settings. Technical Report UM-CS-2010-009, University of Massachusetts, Amherst, 2010.

[68] Vinay Jethava, Krishnan Suresh, Chiranjib Bhattacharyya, and Ramesh Hariharan. Randomized algorithms for large scale SVMs. *ACM Computing Research Repository*, abs/0909.3609, 2009.

[69] Zhaoyin Jia, Ashutosh Saxena, and Tsuhan Chen. Robotic object detection: Learning to improve the classifiers using sparse graphs for path planning. In *Proceedings of International Joint Conferences on Artificial Intelligence*, pages 2072–2078, 2011.

[70] Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59, October 2009.

[71] Vojislav Kecman and J. Paul Brooks. Locally linear support vector machines and other local models. In *Proceedings of International Joint Conference on Neural Networks*, pages 1–6, 2010.

[72] Iasonas Kokkinos. Rapid deformable object detection using dual-tree branch-and-bound. In *Proceedings of Advances in Neural Information Processing Systems*, pages 2681–2689, 2011.

[73] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. Mrf optimization via dual decomposition: Message-passing revisited. In *Proceedings of International Conference on Computer Vision*, pages 1–8, 2007.

[74] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. Mrf energy minimization and beyond via dual decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):531–552, 2011.

[75] Matthieu Kowalski. Sparse regression using mixed norms. *Applied and Computational Harmonic Analysis*, 27(3):303–324, 2009.

[76] Lubor Ladicky and Philip H. S. Torr. Locally linear support vector machines. In *Proceedings of International Conference on Machine Learning*, pages 985–992, 2011.

[77] C.H. Lampert. An efficient divide-and-conquer cascade for nonlinear object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1022–1029, 2010.

[78] C.H. Lampert, M.B. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2129–2142, December 2009.

[79] Tian Lan, Leonid Sigal, and Greg Mori. Social roles in hierarchical models for human activity recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[80] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories.

In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition - Volume 2*, pages 2169–2178, 2006.

[81] Quoc V. Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Greg Corrado, Kai Chen, Jeffrey Dean, and Andrew Y. Ng. Building high-level features using large scale unsupervised learning. In *Proceedings of International Conference on Machine Learning*, 2012.

[82] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *Proceedings of Advances in Neural Information Processing Systems*, pages 801–808. 2007.

[83] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Proceedings of Advances in Neural Information Processing Systems*, pages 801–808. MIT Press, Cambridge, MA, 2007.

[84] Alain Lehmann, Peter V. Gehler, and Luc J. Van Gool. Branch&rank: Non-linear object detection. In *Proceedings of British Machine Vision Conference*, pages 1–11, 2011.

[85] Alain Lehmann, Bastian Leibe, and Luc van Gool. Prism: Principled implicit shape model. In *Proceedings of British Machine Vision Conference*, pages 64.1–64.11, 2009. doi:10.5244/C.23.64.

[86] B. Leibe, A. Leonardis, and B. Schiele. Combined object categorization and segmentation with an implicit shape model. In *Proceedings of the Workshop on Statistical Learning in Computer Vision*, Prague, Czech Republic, May 2004.

[87] Jianguo Li, Tao Wang, and Yimin Zhang. Face detection using surf cascade. In *International Conference on Computer Vision Workshops*, pages 2183–2190, 2011.

[88] Yuanqing Lin, Tong Zhang, Shenghuo Zhu, and Kai Yu. Deep coding networks. In *Proceedings of Advances in Neural Information Processing Systems*, Cambridge, MA, 2010. MIT Press.

[89] Ming-Yu Liu, Oncel Tuzel, Ashok Veeraraghavan, and Rama Chellappa. Fast directional chamfer matching. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1696–1703, 2010.

[90] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of International Conference on Computer Vision*, pages 1150–1157, 1999.

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

[92] Tianyang Ma, Xingwei Yang, and Longin Jan Latecki. Boosting chamfer matching by learning chamfer distance normalization. In *Proceedings of European Conference on Computer Vision: Part V*, pages 450–463, Berlin, Heidelberg, 2010. Springer-Verlag.

[93] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 689–696, 2009.

[94] S. Maji, A.C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[95] Tomasz Malisiewicz, Abhinav Gupta, and Alexei A. Efros. Ensemble of exemplar-SVMs for object detection and beyond. In *Proceedings of International Conference on Computer Vision*, pages 89–96, 2011.

[96] Sylvain Paris Michael Gharbi, Tomasz Malisiewicz and Frédo Durand. A gaussian approximation of feature space for fast image similarity. Technical report, MIT-CSAIL-TR-2012-032, 10 2012.

[97] Fabien Moutarde, Bogdan Stanciulescu, and Amaury Breheret. Real-time visual detection of vehicles and pedestrians with new efficient adaBoost features. In *2nd Workshop on Planning, Perception and Navigation for Intelligent Vehicles (PPNIV), at 2008 IEEE International Conference on Intelligent RObots Systems (IROS 2008)*, Nice, France, 2008.

[98] Joseph L. Mundy. Object recognition in the geometric era: A retrospective. In Jean Ponce, Martial Hebert, Cordelia Schmid, and Andrew Zisserman, editors, *Toward Category-Level Object Recognition*, volume 4170 of *Lecture Notes in Computer Science*, pages 3–28. Springer, 2006.

[99] Kevin P. Murphy, Antonio Torralba, Daniel Eaton, and William T. Freeman. Object detection and localization using local and global features. In *Toward Category-Level Object Recognition*, pages 382–400, 2006.

[100] J. C. Nascimento and J. S. Marques. Performance evaluation of object detection algorithms for video surveillance. *IEEE Transactions on Multimedia*, 8(4):761–774, August 2006.

[101] Jennifer Neville and David Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 8:653–692, May 2007.

[102] T. Ojala, M. Pietikainen, and D. Harwood. Performance evaluation of texture measures with classification based on kullback discrimination of distributions. In *Proceedings of International Conference on Pattern Recognition*, pages A:582–585, 1994.

[103] Andreas Opelt, Axel Pinz, and Andrew Zisserman. A boundary-fragment-model for object detection. In *Proceedings of European Conference on Computer Vision*, pages 575–588, Berlin, Heidelberg, 2006. Springer-Verlag.

[104] Patrick Ott and M. Everingham. Shared parts for deformable part-based models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1513–1520, 2011.

[105] Balamurugan P., Shirish Krishnaj Shevade, and T. Ravindra Babu. Efficient algorithms for linear summed error structural SVMs. In *Proceedings of International Joint Conference on Neural Networks*, pages 1–8, 2012.

[106] Dennis Park, Deva Ramanan, and Charless Fowlkes. Multiresolution models for object detection. In *Proceedings of European Conference on Computer Vision*, pages 241–254, 2010.

[107] Marco Pedersoli, Andrea Vedaldi, and Jordi Gonzàlez. A coarse-to-fine approach for fast deformable object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1353–1360, 2011.

[108] Hamed Pirsiavash and Deva Ramanan. Steerable part models. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3226–3233, 2012.

[109] Esa Rahtu, Juho Kannala, and Matthew Blaschko. Learning a category independent object detection cascade. In *Proceedings of International Conference on Computer Vision*, 2011.

[110] John A. Rice. *Mathematical Statistics and Data Analysis.* Duxbury Press, Belmont, CA, 2nd edition, 1995.

[111] Olga Russakovsky, Yuanqing Lin, Kai Yu, and Fei-Fei Li. Object-centric spatial pooling for image classification. In *Proceedings of European Conference on Computer Vision*, pages 1–15, 2012.

[112] Mohammad Amin Sadeghi and Ali Farhadi. Recognition using visual phrases. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1745–1752, 2011.

[113] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651–1686, 1998.

[114] Henry Schneiderman and Takeo Kanade. Object detection using the statistics of parts. *International Journal of Computer Vision*, 56(3):151–177, 2004.

[115] Hae Jong Seo and Peyman Milanfar. Training-free, generic object detection using locally adaptive regression kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1688–1704, September 2010.

[116] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30, 2011.

[117] Qinfeng Shi, Chunhua Shen, Rhys Hill, and Anton van den Hengel. Is margin preserved after random projection? In *Proceedings of International Conference on Machine Learning*, 2012.

[118] Jamie Shotton, Andrew Blake, and Roberto Cipolla. Contour-based learning for object detection. In *Proceedings of IEEE International Conference on Computer Vision*, pages 503–510, Washington, DC, USA, 2005. IEEE Computer Society.

[119] Jamie Shotton, Andrew W. Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1297–1304, 2011.

[120] Dong-Gyu Sim, Oh-Kyu Kwon, and Rae-Hong Park. Object matching algorithms using robust hausdorff distance measures. *IEEE Transactions on Image Processing*, 8(3):425–429, 1999.

[121] Hyun Oh Song, Stefan Zickler, Tim Althoff, Ross B. Girshick, Mario Fritz, Christopher Geyer, Pedro F. Felzenszwalb, and Trevor Darrell. Sparselet models for efficient multiclass object detection. In *Proceedings of European Conference on Computer Vision*, pages 802–815, 2012.

[122] Erik B. Sudderth. *Graphical models for visual object recognition and tracking.* PhD thesis, Cambridge, MA, USA, 2006. AAI0809973.

[123] Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Sharing features: Efficient boosting procedures for multiclass object detection. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 762–769, 2004.

[124] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, December 2005.

[125] T. Tuytelaars, M. Fritz, K. Saenko, and T. Darrell. The NBNN kernel. In *Proceedings of International Conference on Computer Vision*, 2011.

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

[127] Andrea Vedaldi, Varun Gulshan, Manik Varma, and Andrew Zisserman. Multiple kernels for object detection. In *Proceedings of International Conference on Computer Vision*, pages 606–613, 2009.

[128] Andrea Vedaldi and Andrew Zisserman. Efficient additive kernels via explicit feature maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):480–492, 2012.

[129] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[130] Carl Vondrick, Aditya Khosla, Tomasz Malisiewicz, and Antonio Torralba. Inverting and visualizing features for object detection. *ACM Computing Research Repository*, abs/1212.2278, 2012.

[131] Jinjun Wang, Jianchao Yang, Kai Yu, Fengjun Lv, Thomas S. Huang, and Yihong Gong. Locality-constrained linear coding for image classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3360–3367, 2010.

[132] Liming Wang, Jianbo Shi, Gang Song, and I fan Shen. Object detection combining recognition and segmentation. In *Proceedings of Asian Conference of Computer Vision*, pages 189–199, 2007.

[133] Zhuang Wang, Koby Crammer, and Slobodan Vucetic. Multi-class pegasos on a budget. In *Proceedings of International Conference on Machine Learning*, pages 1143–1150. Omnipress, 2010.

[134] K.Q. Weinberger and L.K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[135] Zhen James Xiang, Hao Xu, and Peter J. Ramadge. Learning sparse representations of high dimensional data on large scale dictionaries. In *Proceedings of Advances in Neural Information Processing Systems*, pages 900–908, 2011.

[136] Jingjing Yang, YongHong Tian, Ling-Yu Duan, Tiejun Huang, and Wen Gao. Group-sensitive multiple kernel learning for object recognition. *IEEE Transactions on Image Processing*, 21(5):2838–2852, 2012.

[137] Tao Yang and Vojislav Kecman. Adaptive local hyperplane classification. *Neurocomputing*, 71(1315):3001 – 3004, 2008.

[138] Weilong Yang, Yang Wang, Arash Vahdat, and Greg Mori. Kernel latent SVM for visual recognition. In *Proceedings of Advances in Neural Information Processing Systems*, pages 818–826, 2012.

[139] Yi Yang, Sam Hallman, Deva Ramanan, and Charless C. Fowlkes. Layered object models for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1731–1743, 2012.

[140] Yi Yang and Deva Ramanan. Articulated pose estimation with flexible mixtures-of-parts. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1385–1392, 2011.

[141] K. Yu and A. Ng. ECCV-2010 Tutorial: Feature Learning for Image Classification. http://ufldl.stanford.edu/eccv10-tutorial/.

[142] Kai Yu and Tong Zhang. Improved local coordinate coding using local tangents. In *Proceedings of International Conference on Machine Learning*, pages 1215–1222, 2010.

[143] Kai Yu, Tong Zhang, and Yihong Gong. Nonlinear learning using local coordinate coding. In *Proceedings of Advances in Neural Information Processing Systems*, pages 2223–2231, 2009.

[144] Guo-Xun Yuan, Chia-Hua Ho, and Chih-Jen Lin. Recent advances of large-scale linear classification. *Proceedings of the IEEE*, 100(9):2584–2603, 2012.

[145] Cha Zhang and Zhengyou Zhang. A survey of recent advances in face detection. Technical report, Microsoft Research, 06 2010.

[146] H. Zhang, A.C. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages II: 2126–2136, 2006.

[147] Wei Zhang, G. Zelinsky, and D. Samaras. Real-time Accurate Object Detection using Multiple Resolutions. In *Proceedings of International Conference on Computer Vision*, pages 1–8, 2007.

[148] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19(2):4–10, April 2012.

[149] Ziming Zhang, Lubor Ladicky, Philip H. S. Torr, and Amir Saffari. Learning anchor planes for classification. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1611–1619, 2011.

[150] Ziming Zhang, Paul Sturgess, Sunando Sengupta, Nigel Crook, and Philip H. S. Torr. Efficient discriminative learning of parametric nearest neighbor classifiers. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2232–2239, 2012.

[151] Ziming Zhang, Jonathan Warrell, and Philip H. S. Torr. Proposal generation for object detection using cascaded ranking SVMs. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1497–1504, 2011.

[152] Yefeng Zheng, Xiaoguang Lu, Bogdan Georgescu, Arne Littmann, Edgar Mueller, and Dorin Comaniciu. Robust object detection using marginal

space learning and ranking-based multi-detector aggregation: Application to left ventricle detection in 2D MRI images. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 1343–1350, 2009.

[153] Jun Zhu, Eric P. Xing, and Bo Zhang. Primal sparse max-margin markov networks. In *Proceedings of ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1047–1056, New York, NY, USA, 2009. ACM.

[154] Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2879–2886, 2012.