Learning a Family of Detectors

Yuan, Quan

Boston University Computer Science Department


http://hdl.handle.net/2144/1746

Boston University
LEARNING A FAMILY OF DETECTORS

QUAN YUAN

Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

BOSTON UNIVERSITY
Approved by

First Reader

Stan Sclaroff, PhD
Professor of Computer Science
Boston University

Second Reader

Margrit Betke, PhD
Associate Professor of Computer Science
Boston University

Third Reader

Mike Jones, PhD
Research Scientist
Mitsubishi Electric Research Laboratories
Acknowledgments

First of all I want to thank my advisor, Stan Sclaroff, for guiding me through my entire Ph.D study. I especially appreciate him for giving me the freedom to pursue my own research interests and the encouragement during hard times in my research. What I have learned from his hardworking and his professional style of being a good researcher will be an invaluable treasure in my future. Secondly I want to thank Margrit Betke. She has tolerated my many unexpected requests and guided me along my research path with her deep insights and wisdom. I am also grateful to my committee member, Mike Jones, for taking time to participate my defense and helping improve the thesis with his deep comments.

I feel very proud to be a member of the IVC group, in which I spent very enjoyable six years. I would thank my research collaborator, Ashwin Thangali, for sharing with me his great problem solving skills and key insights in research projects. The key contributions of the thesis have evolved through the numerous discussions that we have had. I thank Rui, Tian, Vitaly, Zheng, Bill, John, Vassilis, Alex, Joni, Liliana, Sam, Jingbin, Murat, Gokberk, Nazli and FeiFei who have been my friends and the closest neighbors in the lab, for acquainting me with this country and improving my language proficiency.

I would thank my internship mentors, Anna and Fatih, who have set me examples of being great industry researchers and gave me strong support to begin my professional career. I would also thank a group of my fellow Chinese students at Boston University, Guangliang, Kebin, Rui Shi, Le Qiu, Xuetao, Jianxun, Yangyang, Jie Lou, Rui Wan, Jia Shao, Dali, Huiquan, Yipin, Qingnan and Ruomin for being supportive during my PhD study and sharing with me a lot of enjoyment at spare time.

Finally, I own many thanks to my whole family. Especially, I thank my wife Xun, for sharing with me all the moments of happiness through the final years in PhD study. I am grateful to my parents to whom I owe everything. Their self-giving love and caring are the most important source I have relied on in my life.
LEARNING A FAMILY OF DETECTORS

(Order No. )

QUAN YUAN

Boston University, Graduate School of Arts and Sciences, 2010

Major Professor: Stan Sclaroff, Department of Computer Science

ABSTRACT

Object detection and recognition are important problems in computer vision. The challenges of these problems come from the presence of noise, background clutter, large within-class variations of the object class and limited training data. In addition, the computational complexity in the recognition process is also a concern in practice. In this thesis, we propose one approach to handle the problem of detecting an object class that exhibits large within-class variations, and a second approach to speed up the classification processes.

In the first approach, we show that foreground-background classification (detection) and within-class classification of the foreground class (pose estimation) can be jointly solved with using a multiplicative form of two kernel functions. One kernel measures similarity for foreground-background classification. The other kernel accounts for latent factors that control within-class variation and implicitly enables feature sharing among foreground training samples. For applications where explicit parameterization of the within-class states is unavailable, a nonparametric formulation of the kernel can be constructed with a proper foreground distance/similarity measure. Detector training is accomplished via standard Support Vector Machine learning. The resulting detectors are tuned to specific variations in the foreground class. They also serve to evaluate hypotheses of the foreground state. When the image masks for foreground objects are provided in training, the detectors can also produce object segmentation. Methods for generating a representative sample set of
detectors are proposed that can enable efficient detection and tracking. In addition, because individual detectors verify hypotheses of foreground state, they can also be incorporated in a tracking-by-detection framework to recover foreground state in image sequences. To run the detectors efficiently at the online stage, an input-sensitive speedup strategy is proposed to select the most relevant detectors quickly. The proposed approach is tested on data sets of human hands, vehicles and human faces. On all data sets, the proposed approach achieves improved detection accuracy over the best competing approaches.

In the second part of the thesis, we formulate a filter-and-refine scheme to speed up recognition processes. The binary outputs of the weak classifiers in a boosted detector are used to identify a small number of candidate foreground state hypotheses quickly via Hamming distance or weighted Hamming distance. The approach is evaluated in three applications: face recognition on the face recognition grand challenge version 2 data set, hand shape detection and parameter estimation on a hand data set, and vehicle detection and estimation of the view angle on a multi-pose vehicle data set. On all data sets, our approach is at least five times faster than simply evaluating all foreground state hypotheses with virtually no loss in classification accuracy.
# Contents

1 Introduction 1
  1.1 Problem Definitions .................................. 2
  1.2 Motivating Applications ............................... 3
  1.3 Main Contributions .................................... 7
  1.4 Plan of the Thesis ..................................... 9

2 Related Work 11
  2.1 Detection Methods for Object Classes of Large Appearance Variations 11
  2.2 Speedup Strategies for Object Recognition ..................................... 20
  2.3 Summary of Related Work ................................ 23

3 Multiplicative Kernel Formulation 25
  3.1 Multiplicative Kernel Construction .......................... 25
  3.2 Nonparametric $k_\theta$ ................................ 28
  3.3 Detector Training ....................................... 30
  3.4 Detection and Foreground State Estimation ....................... 34
  3.5 Tracking with Multiplicative Kernel Detectors ...................... 36

4 Speedup Strategy for the Classification Process 38
  4.1 From Random Binary Weak Classifiers to LSH ............... 39
  4.2 Optimized Hamming Distance Measure .......................... 41
  4.3 Implementation ....................................... 44

5 Experiments 46
  5.1 Experiments on Detectors Trained via Multiplicative Kernels 46
  5.2 Experiments on Speedup Strategy .......................... 68
5.3 Summary of Experimental Results ........................................ 78

6 Discussion and Conclusions .............................................. 80

   6.1 Main Contributions .................................................. 80

   6.2 Limitations and Future Work ...................................... 81

References ............................................................................. 85

Curriculum Vitae ..................................................................... 93
List of Tables

5.1 View angle estimation error in degrees, in eight different test sequences of vehicle tracking. ............................................. 62
5.2 Comparison of filter step with different distance measures. ................. 70
5.3 Mean absolute error (MAE) in degrees, and average filter+refine time spent on each test example, on the hand data set. ...................... 73
5.4 Median of absolute error (Median-AE) in degrees and the total filter+refine time spent on 200 test examples. ............................. 77
List of Figures

1-1 Example result of face detection and face rotation angle estimation in a single image. ............................................. 1
1-2 Example result of vehicle detection and view angle estimation in a sequence. 4
1-3 Example result of hand detection and shape estimation in gesture images collected from sign language sequences. ..................... 5
1-4 Diagram of face detection + face recognition. .......................... 6

3-1 An experiment on synthetic data. ........................................ 29
3-2 Pseudo code for bootstrap training with parametric within-class kernel $k_r$. 32

5-1 Example sign language sequences from which the training and test hand images are obtained. .................................. 47
5-2 Example hand clusters after training with nonparametric multiplicative kernels. ......................................................... 49
5-3 Example detection and segmentation results for the sign language test sequences. ............................................................. 50
5-4 ROC curves of different detectors for hand detection on gesture images collected from sign language sequences. .................... 51
5-5 Examples from the hand data set [92]. .................................... 53
5-6 Comparison of ROC curves on the hand shape data set with two-dimensional parameters [91]. ........................................ 54
5-7 Example images and their binary segmentation masks from the multi-pose vehicle data set used in [38]. ................................. 55
5-8 ROC curves of vehicle detection experiment on the vehicle data set of [38]. 57
5-9 False negative examples and false positive examples of our method in the first task of vehicle detection. .................................................. 57
5-10 Example frames and ground truth annotations of two cameras in test sequence 5 of the PETS 2001 data set. .................................................. 59
5-11 Vehicle detection rate vs false positive rate on sequence 5 of the PETS 2001 data set. .................................................. 60
5-12 Vehicle detection rates for different vehicle sizes on sequence 5 of the PETS 2001 data set. .................................................. 61
5-13 Four example sequences of vehicle tracking. ................................. 63
5-14 13 face view angle subclasses from the Multi-PIE data set that are used in the experiment. .................................................. 64
5-15 Face view angle estimation result on the Multi-PIE data set. .............. 65
5-16 Example face tracking result in two test sequences. .......................... 66
5-17 ROC curves of face detection on two face sequences with our approach. ... 67
5-18 Example face images in the FRGC data set [57]. .............................. 69
5-19 Recognition and retrieval accuracy on the face data set. ..................... 70
5-20 Results of parameter estimation on the hand data set. ....................... 72
5-21 Example images and masks in the data set from [38]. ......................... 74
5-22 Example results of view angle estimation by HOG feature matching. .... 75
5-23 Comparison of different distance measures on car view angle estimation accuracy vs speedup factors. .................................................. 76
5-24 ROC curves of face detection on two face sequences with comparison to the speedup strategy. .................................................. 77
List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>Two-dimensional</td>
</tr>
<tr>
<td>3D</td>
<td>Three-dimensional</td>
</tr>
<tr>
<td>EM</td>
<td>Expectation-Maximization</td>
</tr>
<tr>
<td>SIFT</td>
<td>Scale-invariant Feature Transform</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

A computer vision system to recognize objects typically has two modules: a detection module and a foreground within-class classification module, for example, face detection and face orientation estimation as shown in Figure 1-1. The detection module localizes the object of interest in an image. For example, a face detector can localize human faces in an image and assign bounding boxes for them. The within-class classification module handles within-class variations of a foreground object. For example, face orientation angles are estimated for each detected face.

In this thesis, we propose novel methods to improve the accuracy and efficiency of object detection and foreground within-class classification processes. An important novelty of the proposed methods is that these two tasks can be solved in concert. Although detection
and within-class classification have different problem definitions, our result is not surprising since in each problem the knowledge of the foreground class is exploited.

Before going into details of the approaches, in this chapter, we give formal definitions to object detection and object recognition, describe challenges in these problems, and outline our approaches. We also highlight the main contributions of this thesis.

1.1 Problem Definitions

In this section, definitions of two target problems – object detection and foreground within-class classification are given.

1.1.1 Object Detection

In computer vision, object detection means the localization of instances from an object class of interest in images, e.g., face detection in static images. The detection module is often implemented as a scanning window process where each window location in an image is evaluated by a binary classifier, i.e., foreground class vs. background class. The input to this binary classifier is the image patch within the current window and the output is a binary value 0 or 1, which indicates whether the input represents an instance of the foreground object or not.

Learning a binary classifier is a classic problem in machine learning. There are a number of different techniques that we can apply, e.g., Fisher linear discriminants [22], decision trees [59], artificial neural networks [62], support vector machines [16], boosting [65], etc.

In practice, to improve the efficiency of detection processes, multi-stage cascade detectors [80] have also been developed. Such detectors exploit the fact that the majority of inputs to a detector are background instances. In [80], trivial background instances are rejected quickly from early cascade stages, which just evaluate a small number of features. Only a small portion of background instances, which are similar to foreground objects, are evaluated in the later stages with higher computational expense. In this way, the boosted cascade yields a speedup in the object detection task.
1.1.2 Foreground within-class Classification

A foreground within-class classification module handles classification problems within the foreground class, e.g., hand pose recognition, person ID recognition and face rotation angle estimation. The input to this module is an image patch depicting an instance of the foreground object class, and the output is a label of the foreground within-class state. This module can be implemented in numerous ways. For discrete state spaces – for example, person IDs, hand shape classes, or vehicle types – estimation can be framed as a multi-class classification problem: given an input feature vector, produce an estimate of the class label. For continuous state spaces – for example, face ages, hand joint angles, vehicle orientations – estimation can be formulated in terms of regression: map a given input feature vector to its most likely location in the foreground state space. Another common approach for foreground state estimation is to use nearest neighbor methods. Given a database of annotated objects and a similarity measure, the database object closest to the query is a “matching” object. The annotation of the matching object is assigned to the query as the output.

1.2 Motivating Applications

A system of detection plus recognition can be applied to a wide range of object recognition tasks. Here we give a few example applications that captured our interest and motivated our work.

1.2.1 Object Detection and View Angle Estimation

Object detection and view angle estimation is an important component of automatic computer vision systems. In such applications, we want to detect the locations and scales of all objects (e.g., faces and vehicles) appearing in an input image and simultaneously get estimates of the view angles (the orientations of the object with respect to the camera). The detection result can be rectangular windows that contain detected objects. The view estimation result can be 3D angles. Naturally, the system cannot know in advance what
the view angles are. Thus, in the detection stages, the detector must be able to deal with objects of different view angles. An example result of vehicle detection and view angle estimation in an image sequence is shown in Figure 1-2.

An important issue that arises in this application is that the object appearances undergo large changes from different view angles. Because the background class is defined as everything other than foreground objects, the foreground-background class boundary in the feature space is highly nonlinear and difficult to be modelled by a single classifier.

1.2.2 Hand Detection and Hand Shape Estimation

Accurate detection of hands and estimating hand shapes are challenging problems, with applications in sign language recognition, human-computer interfaces and virtual reality environments. Magnetic trackers can capture hand shape accurately, but they are expensive and intrusive (users have to wear special equipment). Vision systems are less intrusive to human users, but a robust hand detector is usually difficult to obtain, due to the numerous degrees of freedom in joint angles and anthropometric variations across different subjects. When a cluttered background is present, the hand/background classification problem is very challenging. In Figure 1-3, example results of hand detection and shape estimation are shown in images collected from sign language sequences.
Figure 1.3: Example results of hand detection and shape estimation in gesture images collected from sign language sequences. For each detected hand, an estimate of hand shape by a contour that roughly matches the detected hand.

1.2.3 Face Recognition

A face recognition system is a computer vision application for automatically identifying a person from a digital image or a video frame from a video source. Most face recognition systems assume a face detection step, which localizes face instances in an input image or a video frame. The localized faces are then fed into the recognition stage to identify the IDs of the faces. Popular face recognition approaches include multi-class classification schemes and nearest neighbor searches [77, 84] with face similarity measures. The typical flow chart of a face recognition system is shown in Figure 1.4. Although there exist fast detection strategies for faces, the subsequent face recognition stage can be slow due to a large number of hypotheses to verify. For multi-class classification schemes, the number of hypotheses is equivalent to the number of known face IDs. For nearest neighbor approaches, the number of hypotheses is equivalent to the number of database face images. Speedup strategies to improve the efficiency of a face recognition system have great potential impact on practical systems. For instance, in [6], an efficient multi-class classification strategy can improve the speed of face recognition by a factor of three with little loss of accuracy.
1.2.4 Existing Challenges

When object classes exhibit large within-class variations, detection and foreground within-class classification can be chicken-egg problems. Assuming the objects are detected and segmented from the background, foreground within-class classification is relatively straightforward. Assuming specific variations of the foreground class, detection can be achieved as in [60]. However, if neither the foreground state nor detection is given, then challenges arise. For example, it is difficult for a single detector to cope with all variations of the foreground class, while at the same time providing reliable discrimination between the foreground and background – especially in applications where there are widely varying, or even unconstrained backgrounds.

In addition, the speed of the whole system is also important in practice. Although fast methods like cascade detectors [80, 93] have been successfully applied to detection problems, the subsequent foreground classification process can be slow and become a bottleneck. For example, in a face recognition application, a detected face is compared with hundreds or thousands of face IDs. Common methods that employ nearest neighbor search [84] or large margin classifiers [25] can be slow. If the system is in an environment where faces appear faster than they can be recognized, even if the detection time is negligible, the system still must remove some of the detected faces from further consideration to run in real time.
1.3 Main Contributions

In the first part of the thesis, we propose a novel approach to learn a family of detectors that can handle both detection and foreground state estimation problems. This strategy can make use of foreground image masks in training to improve detection accuracy and can be applied in a tracking framework. Sampling and clustering strategies to find a compact set of detectors are also developed. In the second part of this thesis, we propose a speedup strategy for recognition processes. This speedup strategy reuses weak classifiers from the detection stage and can be applied to a broad category of recognition processes.

1.3.1 Multiplicative Kernel Formulation

In the first approach, we propose to learn a family of detectors, where the detectors themselves are parameterized over the space of within-class variations. Our formulation utilizes a product of two kernel functions: a within-class kernel $k_\theta$ to handle foreground state variations and feature sharing, and a between-class kernel $k_x$ to handle foreground-background classification. This kernel formulation is used in a Support Vector Machine (SVM) training algorithm that outputs support vectors and their weights, which can be used to construct a family of detectors that are tuned to foreground variations. After SVM training, a sample set of detectors can be generated, where each detector is associated with a particular foreground state parameter value. All the samples from the detector family share the same support vectors, but the weights of these support vectors vary depending on the within-class state value. A useful side effect of this support vector sharing is that features are implicitly shared across the whole detector family.

The formulation is useful in solving detection, state estimation, and tracking problems. For detection in a scanning window process, an image window can be classified as foreground if at least one of the detectors in the family produces a score that is above a predefined threshold. For a given image window, the foreground state can be estimated simply by examining the state values associated with detectors that produce the highest responses for that input. For particle filter-based tracking methods, like CONDENSAC-
TION [32], importance sampling from the detector family can be driven by a dynamical model at each frame, where the objects are allowed to undergo a range of state variations over time.

With proper nonparametric kernel functions, our formulation can be extended to nonparametric cases, when explicit parameter annotation of the foreground class examples is too expensive to obtain. For instance, it is difficult to annotate joint angles in a large training set of articulated objects like human hand or human body. A mode finding method is also proposed that selects a representative subset of samples from the detector family in the nonparametric case. This generally reduces the number of detectors to be invoked, and thereby makes detection more efficient. If state estimation or tracking is desired, then the user can label the state for each sample in the representative subset. This alleviates the burden of assigning ground truth states for the complete training set, and instead focuses only on labelling the smaller representative subset.

The proposed framework is evaluated in three application areas. The first involves hand detection, segmentation, and shape estimation for images taken from videos of Flemish and American Sign Language. There is a wide range of variations of hand shapes and orientations in these videos. The framework is also tested in estimating index finger angles. The second application involves detection, orientation estimation and tracking of vehicles driving on highways, and the more challenging case of race cars careening on dirt roads. The third application focuses on the problem of detecting and tracking multiple human faces, while simultaneously estimating the left-right rotation angles under illumination variations. The proposed approach compares favorable to existing techniques in theses experiments. For instance, the proposed approach improves detection accuracy by eight percent on the sign language image data set, and by 20 percent on a vehicle detection data set at fixed false positive rates. It also achieves jointly detection and view angle estimation of multi-pose faces with a much smaller training set (one tenth to one fifth compared to existing methods).
1.3.2 Speedup Foreground Within-Class Classification via Reusing Features from Detectors

In the second approach, we propose a filter-refine strategy to address the issue of computational complexity of recognition processes. The proposed strategy reuses weak classifiers outputs from an initial detection stage, where the detector is trained by Adaboost. We employ a distance optimization method to select a subset of weak classifier outputs to compose Hamming codes of detected objects. In the filter step, a detected object is compared with all foreground state hypotheses via a Hamming distance. Implausible foreground state hypotheses can be removed quickly in this filter step. Thus, only a small number of hypotheses must be evaluated via a more expensive but accurate approach in a refine step. This yields a speedup in classification while preserving the overall accuracy.

The speedup approach is evaluated in three applications: face recognition on the Face Recognition Grand Challenge Version 2 (FRGC V2) data set, hand shape detection and parameter estimation on a hand data set, and vehicle detection and view angle estimation on a multi-pose vehicle data set [38]. On all data sets, our approach is about an order of magnitude faster than simply evaluating all hypotheses, with virtually no loss of accuracy. Interestingly, in the face recognition application, the proposed speedup strategy can improve the accuracy by about 0.5 percent.

1.4 Plan of the Thesis

The plan for the rest of the thesis is as following:

Chapter 2 gives a review of existing methods for detecting object classes that exhibit a large range of appearance variations, and existing speedup strategies for object recognition tasks.

Chapter 3 describes the first contribution of the thesis, which is a multiplicative kernel formulation to learn a family of detectors. This family of detectors cover the range of appearance variations in the foreground class. Sampling and clustering strategies are also proposed to obtain a compact set of detectors to be applied at the detection stage. The
family of detectors can also be employed in a tracker to achieve “tracking by detection.”

In Chapter 4, a filter-refine strategy to speed up foreground within-class classification is described. The strategy selects a subset of weak classifiers from detection stage to construct a Hamming distance for a fast filter step for foreground within-class classification. Because the weak classifiers are reused from detection stage, it requires minimum extra calculation in the filter step, which yields a speedup.

In Chapter 5, the proposed methods are evaluated on several different applications: hand shape detection and shape estimation, multi-pose vehicle tracking with view angle estimation, multi-pose face detection with face rotation angle estimation, and face recognition. The competing existing approaches include divide-and-conquer approaches [86], explicit feature sharing approaches [75, 91], and efficient multi-class classification strategies [6]. The proposed methods achieve improved accuracy or efficiency compared to these existing techniques, with quantitative comparison results.
Chapter 2

Related Work

There are two approaches proposed in this thesis. The first one jointly solves detection and foreground within-class classification problems. It can be applied to detecting objects of like human hands, faces, and vehicles. It can also be applied to tracking applications where both object locations and foreground states are of interest. The second approach described in this thesis is applicable to speeding up recognition tasks when an upfront detection step can be assumed. For example, face recognition after face detection.

This chapter presents related work from different areas: face detection, recognition and rotation angle estimation; hand detection and hand shape estimation; multi-pose vehicle detection; pedestrian detection and tracking; speedup strategies for multi-class classification and nearest neighbor search.

2.1 Detection Methods for Object Classes of Large Appearance Variations

In this section, we review related works that handles object classes of large appearance variations.

2.1.1 Subspace Appearance Models

A large amount of work in computer vision has been dedicated to handling the issue of recognizing an object class that exhibits large appearance variations. For instance, generative models [49, 55] were proposed to learn a set of low-dimensional representations that cover a broad range of appearance variations. In the work of Nayar et al. [49], a large set of training images is collected of an object by varying pose and illuminations. The
object is then represented as a manifold in a low-dimensional subspace, which is obtained by principal component analysis (PCA). Given a novel input image, the identity and the pose of a new object is recognized based on the manifold it lies on and its exact position on this manifold.

In the work of Pentland et al. [55], given $N$ individuals under $M$ different views, a “view-based” set of $M$ separate eigenspaces are used to capture the variation of the $N$ individuals in a common view. In a face recognition task, this approach demonstrates superior performance over a universal eigenspace computed from the combination of $NM$ images.

A limitation of the aforementioned two approaches is that they do not address the issue of foreground/background classification, i.e., given an image with background clutter, how the object of interest is localized. In practice, when background clutter is present, the foreground/background classification problem can be quite challenging.

2.1.2 Detection by Skin Color and Background Models

For objects like body parts, a model of skin color can be used for object localization [34]. In cases when the skin color is surprisingly uniform, color-based hand detection is possible [94]. However, this by itself is not a reliable approach. There may exist background objects that bear similar colors to skin. There are also cases when the assumption on illumination is not true, e.g., colored lights or grey images.

Background modelling is an effective way to deal with static or steadily changing background in video surveillance applications. In the work of Stauffer and Grimson [73] an adaptive model is proposed for object tracking with surveillance cameras. In this model, the values of a particular pixel over time are treated as a “pixel process,” which is modelled by a mixture of Gaussians. The prior weight of each Gaussian is also affected by the time elapsed since the last time the pixel value matched this Gaussian. In this way, the Gaussian that matches the most recent pixel values is weighted highly. At the same time, when something becomes part of the background, it does not destroy the existing model of
the background. Background modelling has also been applied to people tracking \cite{85, 70} and vehicle tracking \cite{45}. However, the assumption that the foreground objects have distinct colors over background regions may not always be true. Furthermore, background modelling may not be applicable for detection problems with rapidly moving cameras or in a single image.

2.1.3 Subclass Detectors Obtained via Machine Learning Techniques

Recently, machine learning techniques like boosting \cite{64} and support vector machines \cite{16} have been widely used to learn detectors that tend to be more robust to background clutter. A boosted cascade of detectors using Haar wavelet features is proposed for face detection in the work of Viola and Jones \cite{80}. At each stage in the cascade, a binary classifier decides whether the input is a background patch (to reject) or needs further checks by later stages in the classifier cascade. The initial stages reject trivial background patches quickly. Thus, only a small portion of background patches that are similar to the object class (e.g., human faces) need to be evaluated with more features in deeper stages.

The cascade detector approach is also extended to hand detection in the work of Kolsch and Turk \cite{37}. In this work, cascade detectors are used to detect six different hand postures separately. The hand posture that can be detected with the highest accuracy is chosen as an initialization posture for a hand tracking application.

In the work of Dalal and Triggs \cite{18}, a pedestrian detector is trained using support vector machines with histogram of oriented gradient (HOG) features. HOG features are particularly good at capturing local edge orientations. For pedestrian detection, the edge information on the limbs and body boundaries are important. A SVM classifier with linear kernel can detect upright pedestrians well in scenes where background clutter is present.

Besides background clutter, the appearance variation of the foreground class is another factor that makes detection problems difficult. For example, when viewed from different angles, rigid objects like faces and vehicles may have quite different 2D appearances. Other objects like human body and human hands are articulated objects with large number
of degrees of freedom, therefore the detection tasks for the object classes are even more challenging.

When large appearance variations are taken into account, a detector that can work well for all variations in the foreground class is very difficult to obtain. To deal with large appearance variations in multi-pose faces, recent work in multi-pose face detection builds different detectors for different face rotation ranges. In the work of Jones and Viola [35], subclasses according to different face orientations are created and corresponding detectors are learned for each subclass. At the detection stage, an initial pose estimator decides which subclass detector should be applied on an input.

In the work of Li et al. [40], a coarse to fine pyramid of face orientations is created. The lower the level of the pyramid, the finer partition of the view space is used. For each subclass generated by a partition, a detector is trained using an improved version of Adaboost (FloatBoost), which employs a backtrack mechanism after each iteration of AdaBoost to remove weak classifiers which cause higher error rates. In the detection stage, at each level of the pyramid, the input image patch is rejected as a background patch if none of the subclass detectors gets a positive score. Only those image patches that can go through all levels of the pyramid are classified as faces.

In the work of Huang et al. [29], a coarse to fine tree structure is built for multi-pose faces. Each node in the tree corresponds to a range of 3D angles (a subclass) of face orientations. At each node a detector is learned for this subclass. If an input is classified by the detector of the current node as from the corresponding face subclass, the input is passed to the detector’s children in the tree (one or multiple) to be further examined; otherwise it is rejected as from the background class. An input that passes a sequence of nodes from the root to a leaf is detected as a face.

Similar approaches that partition the foreground class according to foreground state annotation [74] or via unsupervised clustering [23, 52, 66, 86] are employed in pedestrian and human hand detection. In the work of Gavrila [23], to detect pedestrians, a template hierarchy is constructed by clustering similar templates and representing each cluster with
a single prototype template. An estimate of the variance of errors within the cluster is calculated. This estimate is used to define a matching threshold, which decides whether to compare the input image to all templates within the cluster. In the detection stage, an input edge image is compared with templates in the hierarchy from top down. If the input matches one of the templates with a matching score below the threshold, it is accepted as a pedestrian instance.

A template tree is used to obtain a good and efficient approximation to Bayesian filtering for hand tracking [74]. The hand templates are generated from a parametric 3D model. Thus, the transition probabilities between hand shapes have physical meaning, e.g., the rate of change of a joint angle. The posterior distribution of hand shapes is encoded using a piecewise constant distribution over the leaves of the tree. Thus, tracking of hands can be achieved efficiently.

To detect real hands in images, examples of real hands are collected from sign language video sequences and clustered into subclasses [52]. Each subclass has a cascade detector. The cascade detectors are invoked after an initial hand/nonhand classifier. During the detection stage, if an input is accepted by the initial classifier as a hand, the cascade detectors will be evaluated one by one on this input. The detection result also reveals the hand shape subclass label. However, since the hand subclasses are obtained via unsupervised clustering, there can be different hand shapes mixed in one subclass due to imperfectness of the hand similarity measure (shape context [8] in this work).

For pedestrian detection, in the work of Seemann et al. [66], a visual vocabulary (or codebook) of typical pedestrian structures is collected. The spatial distribution of each code book entry is estimated during training. To handle human body articulations and viewpoint variations in detection, viewpoint/articulation clusters are obtained in training data. The recognition process also output the best matching shape cluster for a given input. Local evidence inconsistent with the estimated shape cluster is eliminated to improve the accuracy.

A tree-based detector is proposed in the work of Wu and Navatia [86]. The space of
training samples is progressively divided into a hierarchy by unsupervised clustering (K-means). In training, whether to split the current node is decided by the decrease of training error at a boosting iteration. After splitting, the classifiers at internal nodes are retrained by combining selected features of their children. In the end, subclass detectors can be obtained for each leaf node in the hierarchy.

An interesting effect in these approaches that divide the foreground class into subclasses is that the foreground within-class variations can be revealed simultaneous during detection. For instance, the multi-pose face detectors [29] also outputs estimate of face rotation angles. However, a common issue in training subclass detectors is that the detectors will have limited power when there are too few training samples in each subclass. Thus, a large foreground training set is required to handle a large number of foreground subclasses.

To make the best use of limited training data, feature sharing [75, 91] is important for multi-class detection. In the work of Torralba [75], detectors of different foreground classes are trained jointly via boosting. The selection of each weak classifier takes into account its performance for multiple foreground classes. The calculation of optimal feature sharing during training has a combinatorial complexity. Thus, a greedy strategy is employed to make training efficient. The final detectors improve detection accuracy when the number of training examples in each class is limited, as a result of the sharing mechanism.

Another feature sharing method is proposed in the work of Yuan et al. [91] for object detection. The weak classifiers are also shared among foreground training examples. Overlapping clusters in the foreground class are used to define the supports among training examples for a weak classifier. After training, each foreground training example has a detector for its own, while each feature is shared among multiple detectors.

Explicit feature sharing makes training more expensive due to the combinatorial complexity in choosing classes or training examples to share features. In both approaches [75, 91], greedy strategies to select sharing classes or training examples of each feature are employed as a tradeoff for training speed. This means that only an approximately optimal sharing can be found.
2.1.4 Hybrid Methods

There also exist hybrid methods that unify detection and foreground state estimation. For example, some approaches combine bottom-up part-based detectors with top-down geometric constraints [21, 31, 70, 82, 90].

For human detection, a probabilistic method is proposed in the work of Ioffe et al. [31]. To detect body components, body parts like limbs and torso are defined as segments of parallel lines. Candidates of body parts are detected by a search for parallel lines. Then the best assembly of detected parts is obtained via sampling in the space of possible configurations. The likelihood of an assembly is defined to be proportional to the probability of seeing an assembly in a random view of a person.

A foreground object is represented by a collection of parts arranged in a deformable configuration in the work of Felzenswalb and Huttenlocher [21]. The configuration is represented by spring-like connections between pairs of parts. The connections between parts are restricted to a tree structure and the energy function between two parts has a particular form as a Mahalanobis distance. Thus, an efficient dynamic programming algorithm can be applied to find the optimal configuration.

In another bottom-up approach [70], the 3D human tracking problem is posed as inference in a graphical model. Conditional probabilities relating the 3D pose of connected limbs are learned from motion-capture training data. Human pose and motion estimation is solved with non-parametric belief propagation using a variant of particle filtering that can be applied over a general loopy graph.

The above methods are applied to handle large appearance variations of articulated objects like the human body. A common issue of these approaches is that the search for the optimal configuration can be quite expensive without constraints. One reason is that body part detection can output a large number of false positives in the initial stage, when the body configuration has not been taken into account. Furthermore, exponential complexity in the configuration space makes it difficult to find the optimal solution in a limited time.
Efficient bottom-up search algorithms to find object of variable shape structure are proposed in the work of Wang et al. [82] and extended to tracking in the work of Wu et al. [90]. The term “variable shape structure” is used to characterize object classes in which some shape parts can be repeated an arbitrary number of times, some parts can be optional, and some parts can have several alternative appearances. A generalization of Hidden Markov Models is introduced with a polynomial inference algorithm. The proposed approach determines object location, orientation, scale and structure by finding the globally optimal registration of model states with local image features, in the presence of clutter.

Some other approaches employ a recognition-verification strategy (e.g., [61]), where a one-to-many mapping is used to produce estimates of body pose (bottom-up), and then recognition models are used to verify pose estimates (top-down). Nevertheless, bottom-up recognition from images with background clutter remains difficult, and the verification step cannot correct an error when the recognition is already wrong.

In the work of Bissacco et al. [9], a probabilistic model using Latent Dirichlet Allocation is employed to represent the statistics of images for human pose classification. Quantized HOG features are used as the basic feature for this generative model. A likelihood ratio test determines if an input patch is a person. The human detection accuracy of this approach is slightly lower than a binary SVM classifier [18] as a human detector. The proposed generative model outputs pose estimates.

In the work of Sminchisescu et al. [71], a generative model is used to predict a human body pose and then verify it using a recognition model. The generative model is a mixture of Gaussians with dense SIFT [43] feature descriptors, computed at a regular grid inside the detection window. The recognition model is a conditional mixture of Bayesian experts. The generative model and the recognition model are jointly optimized in a Variational Expectation-Maximization (EM) process.

However, for classification problems, probabilistic models are usually not as robust as discriminative models [78], due to the fact that modelling distributions of classes usually involves estimation of more model variables than discriminative approaches.
2.1.5 Tracking by Detection

Recently, there is increasing interest to employ object detectors in object tracking approaches. One way to employ detectors for tracking is via online learning of foreground/background classifiers [7]. In this approach, an ensemble of weak classifiers is trained online to distinguish between the object and the background. The ensemble is a strong classifier trained by Adaboost. Temporal coherence is maintained by updating the ensemble with new weak classifiers that are obtained online during tracking.

Another strategy to achieve tracking by detection is to maintain a set of detectors during tracking. In a 2D face tracking approach [41], a set of face detectors are trained with different subsets of training examples and different subsets of features. These face detectors are then combined into a multi-stage cascade for importance sampling. During tracking, the sampling process is similar to an annealed particle filter [20]. The difference is that the total number of particles is decreased from early stages to deeper stages. Eventually the particle that passes all stages with the highest weight denotes the face location.

In our approach, the detectors are associated with foreground state annotations. Thus, they can be employed in a particle tracking framework to verify foreground state hypotheses during tracking. The temporal information makes the detector sampling process efficient. Online learning or a cascade sampling process are not required.

2.1.6 Other Kernel Combination Formulations

Two previous approaches [19, 47] also use kernel combinations in formulating classification problems. However, in these works, the kernel combinations are used to combine different feature channels. These methods are designed to only solve a single binary classification problem, not to learn a family of detectors as in our approach, where both foreground-background classification and foreground within-class classification problems are jointly solved.
2.2 Speedup Strategies for Object Recognition

2.2.1 Fast Classification Strategies

Our approach to speed up recognition processes is related to fast multi-class classification strategies [58]. In this work, a multi-class classifier is constructed by combining binary classifiers in a directed acyclic graph. It employs the same number of binary classifiers as a one-versus-all (OVA) approach, but each binary classification is much simpler than OVA; therefore it runs faster. However, for \( n \) classes, the total number of binary classifiers to be trained is on the order of \( n^2 \), which makes the method impractical for problems with large numbers of classes.

The filter-refine strategy has been used in detection and multi-class classification approaches, e.g., [80, 6]. In the work of Viola and Jones [80], a cascade detector is constructed to make object detection much faster. Trivial background instances are rejected early in the cascade. However, for multi-class classification, every hypothesis must be evaluated before it can be rejected. Thus, an input will have to be evaluated with all hypotheses anyway. A cascade structured filter step to reject hypotheses will not have the same advantage as in a detection process.

In the work of Athitsos et al. [6], an embedding-based approach was proposed to speed up multi-class classification. Patterns and classes are mapped to vectors in such a way that patterns and their associated classes tend to get mapped close to each other. Thus, an efficient filter step can be employed in the embedded space to identify a small number of candidate classes. This approach can be applied to a variety of multi-class classification problems. However, extra training is needed to learn the embedding [6], which usually implies a requirement for extra training data. Furthermore, the learned mapping functions need to be calculated using classifiers from the refine stage, which are usually slow in speed.

In another strategy [72], feature reuse has been proposed to make detection processes more efficient. It is shown that reusing features can improve the speed of cascade detectors by 25%. This work speeds up detection, but does not address a subsequent multi-class clas-
sification step. Reusing features has an obvious advantage of minimum extra calculations. In our work, we build the connection between detection and foreground within-class classification, which makes it possible to reuse features from detectors for foreground within-class classification.

In addition, methods [28, 40] mentioned in the previous section integrate detection and foreground classification, whereby the detection result also reveals the foreground state, e.g., face view angle. The divide-and-conquer mechanism achieves great improvement in detection accuracy. However, to achieve accurate foreground state estimation, fine partitioning of the foreground space is needed; this implies the need for a sufficiently large amount of training data with foreground within-class state annotations to use in training a classifier for each foreground subclass, or a feature sharing approach [75] is necessary.

### 2.2.2 Fast Nearest Neighbor Approaches

The nearest neighbor approach is one of the most widely used approaches in pattern recognition. One of the main reasons for its popularity is simplicity. Nearest neighbor approaches can be applied to multi-class classification and continuous parameter estimations problems [67]. There exists a large amount of work for fast nearest neighbor approaches in the literature. Readers can find comprehensive surveys on fast nearest neighbor approaches in [11, 27, 26].

Some of existing speedup strategies [83, 14] for nearest neighbor approach can guarantee the exact nearest neighbor. A simple vector approximation scheme called VA-file is proposed in the work of Weber *et al.* [83] to make the search as fast as possible. With VA-file, the data space is divided into $2^b$ rectangular cells where $b$ denotes a user specified number of bits. The VA-file allocates a unique bit-string of length $b$ for each cell, and approximates data points that fall into a cell by that bit-string. Given this rectangular representation of cells in data space, the upper and lower bounds on the distance to a query can be easily determined during scanning of the approximation file. If the user keeps a smallest upper bound found so far, all those data points that are in cells of larger lower bounds can be
filtered out quickly. Thus, it yields a speedup for the search. In another approach that guarantees exact nearest neighbor [14], correlated clusters in the data are identified, and points are assigned cluster labels according to their distance to cluster centers. Principal components of existing clusters are then calculated. The choice of principal components satisfies that reconstructed distance in each cluster must be bounded by a maximum representation error. Outliers in the data are maintained separately. A disk-based global index structure is then built for efficient search.

Other works proposed use of approximate nearest neighbor search schemes instead of exact search. The locality sensitive hashing (LSH) [30] approach is of particular interest in recent years for its efficiency in nearest neighbor search problems. The basic idea of LSH is to hash the input items so that similar items are mapped to the same buckets with high probability. A valid LSH family is composed of hashing functions that have higher probabilities for similar objects to collide than the probabilities for dissimilar objects to collide. With LSH, a query is mapped to binary strings by hashing functions. Database objects that have at least one binary string identical to the query are retrieved. By appropriately choosing the length of the binary strings and the number of hashing functions, the probability that the query will collide with its near neighbors in the database becomes very high, while the total number of objects to be retrieved remains low.

A LSH family is obtained to recognize body poses from input images in the work of Shakhnarovich et al. [67]. The hashing functions defined on image features are constructed such that images of similar body poses have a high probability to collide and images of dissimilar body poses have a low probability to collide. In this way, an input image can be compared quickly with a large database of annotated body pose images. Only those that collide with the query during hashing are needed to be compared with more expensive image similarity measures. Thus, the body pose recognition can be achieved in an efficient way.

In our approach, the filter step is a fast nearest neighbor search process. However, an important difference from existing techniques is that the feature evaluations are reused
from the initial detection stage, which is usually required to localize the object before recognition.

We should also mention that, for foreground state recognition, regression-based methods [1, 10] can also be used. Although our approach is not applicable to speeding up regression based methods, it can be applied to alternative methods like the nearest neighbor method, which can solve general foreground state estimation problems.

2.3 Summary of Related Work

This chapter reviewed a variety of methods for detecting object classes of large within-class variations and methods to speedup multi-class classification or nearest neighbor search processes. We give summaries of these related works in the following paragraphs.

To detect articulated objects like a human body, bottom-up approaches can become too expensive for real-time applications, particularly when background clutter is present. First, without a global constraint at the component detection stage, the number of candidate components detected in an image can be huge. The usually over-simplified component model (e.g., rectangles for limbs and torso) also adds to false positives in the component detection stage. Second, the search complexity for an optimal configuration is exponential in terms of number of candidate components. Thus, a brute force search is often intractable in practice. While simplified approaches (e.g., tree-like graphical model) do not guarantee the optimal solution.

Existing approaches also handle large appearance variations in the foreground class via divide-and-conquer strategies, i.e., the object class is partitioned into small subclasses and corresponding subclass detectors are obtained. A strong limitation in these approaches is the requirement of large amount of training data to provide enough training examples in each subclass. Although feature sharing techniques [75] can be helpful to improve the detection performance with limited training data, the training process is expensive (quadratic in terms of number of subclasses). Intuitively, it is awkward to first partition the object class but then try to combine them during feature sharing. The proposed approach in this
thesis finds a solution where detectors tuned to specific variations in the foreground class are jointly learned. An explicit partition of the object class is not necessary.

In many practical applications like face recognition and gesture recognition, an initial detection stage is usually required to localize the object or reduce the search space in 2D. However, existing techniques deal with the computational complexity issue of recognition processes in isolation, \textit{i.e.}, the image features or weak classifiers that have been evaluated in the detection stage are completely ignored in the subsequent recognition stage. Consequently, compared to the speedup strategy proposed in this thesis, existing speedup strategies need to evaluate more features or classifiers in the recognition stage.
Chapter 3

Multiplicative Kernel Formulation

In this chapter, we give details of a unified solution for object detection and foreground state estimation (or foreground within-class classification).

3.1 Multiplicative Kernel Construction

Given a feature vector \( x \in \mathbb{R}^n \) computed for an image patch\(^1\), our goal is to decide whether or not the corresponding image patch depicts an instance of the object with parameter \( \theta \in \mathbb{R}^m \), which parameterizes certain variations of the foreground object class, e.g., joint angles, view angles, or latent factors that can be obtained via unsupervised learning. We aim to learn a function \( C(x, \theta) \) that tells whether \( x \) is an instance of the object with parameters \( \theta \),

\[
C(x, \theta) = \begin{cases} 
> 0, & \text{if } x \text{ is an instance of the object with } \theta \\
\leq 0, & \text{otherwise.}
\end{cases} \tag{3.1}
\]

The function \( C(x, \theta) \) is different from a generative model \( P(x, \theta) \) which has been explored in different contexts. Instead of estimating the distribution, \( C(x, \theta) \) only makes a binary decision as in Eq. 3.1. The magnitude of \( C(x, \theta) \) can be interpreted as the confidence of the decision.

Let \( y = [x^T, \theta^T]^T \), then \( C(y) \) is in the standard form of a binary classifier as defined in Eq. 3.1. Given a kernel function \( k_y(\cdot, \cdot) \) and training data, \( C(y) \) can be formulated using a Support Vector Machine (SVM) [16]. The kernel \( k_y(\cdot, \cdot) \) can be constructed as a

\(^1\)In this paper, all vector variables are column vectors.
A combination of kernels defined on $x$ and $\theta$, for example,

$$k_y(y, y') = k_x(x, x') + k_y(\theta, \theta')$$ (3.2)

or

$$k_y(y, y') = k_x(x, x')k_\theta(\theta, \theta'),$$ (3.3)

where $y' = [x'^T, \theta'^T]^T$, and $k_x$ and $k_\theta$ are valid Mercer kernels.

We are particularly interested in the multiplicative form of Eq. 3.3 because it has an explicit interpretation of learning a continuous space of detectors, in which different detectors are tuned to different parameters $\theta$.

Assume $C(x, \theta)$ can be factorized into the product of a feature space mapping $\phi_x(x)$ and a weight vector $w(\theta)$, which is a function of $\theta$,

$$C(x, \theta) = \phi_x(x)^T w(\theta)$$ (3.4)

where $\phi_x(x) = [\phi_0^x(x), \phi_1^x(x), \ldots, \phi_N^x(x)]^T$ is an expansion to a higher-dimensional space, e.g., polynomial expansion. The weight vector $w$ is a function of a continuous variable $\theta$ in our formulation. The benefits of this formulation are twofold. First, we obtain different detectors for different foreground variations encoded in $\theta$, without arbitrarily partitioning $\theta$ space. Second, feature sharing across variations of the foreground class is achieved implicitly.

In this formulation, $w(\theta)$ is approximated by a set of basis functions, where vectors of coefficients are the unknowns to be learned from data,

$$w(\theta) = \sum_{i=0}^{M} v_i \phi_i^\theta(\theta) = V \phi_\theta(\theta)$$ (3.5)

where vectors $v_i \in \mathbb{R}^{N+1}$ are unknowns, and

$$V = [v_0, v_1, \ldots, v_M]$$

$$\phi_\theta(\theta) = [\phi_0^\theta(\theta), \phi_1^\theta(\theta), \ldots, \phi_M^\theta(\theta)]^T.$$
The functions $\phi_x$ and $\phi_\theta$ are given by the user. If we plug Eq. 3.5 into Eq. 3.4,

\[
C(x, \theta) = \phi_x(x)^T V \phi_\theta(\theta)
\]

\[
= \begin{bmatrix}
\phi_0^T(\theta) \phi_x(x) \\
\phi_1^T(\theta) \phi_x(x) \\
\vdots \\
\phi_M^T(\theta) \phi_x(x)
\end{bmatrix}^T
\begin{bmatrix}
v_0 \\
v_1 \\
\vdots \\
v_M
\end{bmatrix}
\]

(3.6)

\[
= \phi_{x, \theta}^T v,
\]

(3.7)

where $\phi_{x, \theta}$ and $v$ are the left and right vectors respectively in Eq. 3.6. Note Eq. 3.7 is a standard binary classification problem, in which $\phi_{x, \theta}$ is the data term and $v$ are unknown weights. These weights can be estimated via a standard kernel based learning method, in our case, the Support Vector Machine (SVM) [16]. The kernel function $k_c(\cdot, \cdot)$ is

\[
k_c(y, y') = \phi_{x, \theta}^T \phi_{x', \theta'}
\]

\[
= [\phi_\theta(\theta)^T \phi_\theta(\theta')] [\phi_x(x)^T \phi_x(x')]
\]

\[
= k_\theta(\theta, \theta') k_x(x, x'),
\]

(3.8)

Learning of $V$ is implicit with this kernel representation. If formulated as a SVM, the classification function becomes,

\[
C(x, \theta) = \sum_{i \in SV} \alpha_i k_\theta(\theta_i, \theta) k_x(x_i, x)
\]

\[
= \sum_{i \in SV} \alpha'_i(\theta) k_x(x_i, x),
\]

(3.9)

where $\alpha_i$ is the weight of the $i^{th}$ support vector, and

\[
\alpha'_i(\theta) = \alpha_i k_\theta(\theta_i, \theta).
\]

(3.10)

Note that original learning of $w(\theta)$ is converted into learning of support vector weights $\alpha_i$.

Feature sharing among different $\theta$ is implicitly achieved by sharing support vectors.
When $k_{\theta}(\cdot, \cdot)$ is strictly non-negative, e.g., a radial basis function (RBF) kernel, Eq. 3.9 can be interpreted as re-weighting the support vectors so that only those having parameters similar to $\theta$ are assigned high weights. Fewer support vectors have to be taken into account in a local subregion in $\theta$ space.

Fig. 3-1 illustrates the basic idea of our approach using a synthetic data set, where the foreground class is parameterized by an angle $\theta$. The goal in training is to obtain local linear decision boundaries, which are parameterized by $\theta$, to separate foreground examples from background examples. Fig. 3-1(b) shows results obtained where $k_{\theta}$ is a Gaussian RBF kernel and $k_{x}$ is a linear kernel. Ideal local boundaries in this case are tangent lines on the class boundary. After training, a local linear boundary is reconstructed as a weighted sum of support vectors in $x_1, x_2$ space. The original data is plotted on the left graph in Fig. 3-1 and two examples of reconstructed local linear decision boundaries are plotted on the right graph in Fig. 3-1.

Once we have obtained all support vectors and corresponding weights $\alpha_i$ after SVM learning, we are able to evaluate $C$ for a given tuple $(x, \theta)$ as defined in Eq. 3.1. If we fix $\theta$, then $C(\cdot, \theta)$ is a detector for a specific $\theta$ value. Conversely, given an $x$ from the foreground class, we can search for the $\theta$ that gives the highest score via $C(x, \theta)$ and use this as the parameter estimate of $x$, i.e., by finding $\hat{\theta} = \arg \max_{\theta} C(x, \theta)$.

### 3.2 Nonparametric $k_{\theta}$

In some problems, parametric forms of foreground within-class variations may not be readily available. For example, there are numerous degrees of freedom in the human hand and the human body. Manual annotation of a large real image data set of hand shapes or body poses can be very expensive, tedious, and prone to errors. For such cases, we propose a nonparametric formulation for the within-class kernel $k_{\theta}$.

To understand the usage of the nonparametric $k_{\theta}$, we need to explain the role of the parametric $k_{\theta}$ in feature sharing as outlined in Sec. 3.1. When $k_{\theta}$ is defined on a continuous $\theta$ space, two training samples with close $\theta$ values should obtain a high $k_{\theta}$ score, and thus
Figure 3.1: An experiment on synthetic data. The foreground class is parameterized by an angle $\theta$. A family of multiplicative kernel classifiers is learned, where $k_\theta$ is a RBF kernel defined on $\theta$, and $k_x$ is a linear kernel defined on $\mathbf{x} = (x_1, x_2)^T$. The linear boundaries for example detectors $C(\mathbf{x}, 23^\circ)$ and $C(\mathbf{x}, -30^\circ)$ are shown on the right as two short black lines. The circled points are the reweighted support vectors (Eq. 3.10). These synthetic “foreground” and “background” classes were chosen to illustrate the idea that local discriminants can be learned jointly via multiplicative kernels, and then reconstructed at a given $\theta$.

are more likely to share features. Conversely, training samples that are far from each other in $\theta$ space are less likely to share features, and should obtain a small $k_\theta$ score. We aim to preserve this similarity behavior in designing a nonparametric kernel.

A straightforward design of a nonparametric kernel $k_\theta$ employs a nonparametric similarity/distance measure, e.g., bidirectional chamfer edge distance [5, 23] or shape context distance [8]. These distance metrics have been used successfully to measure within-class similarities for object classes like hand shape and body pose. They tend to keep similar objects close to each other, and dissimilar objects distant to each other.
Based on a distance measure $D$, a kernel function can be defined \cite{51},

$$k_{\theta}(i, j) = \exp(-\eta D(z_i, z_j)),$$

(3.11)

where $z_i$ and $z_j$ are representations of the foreground training samples indexed by $i$ and $j$ to calculate distance $D$. We note that the representation $z$ is selected to be well-suited for describing within-class variations; therefore, its representation may differ from the feature space of $x$ used for foreground-background discrimination.

After training, in contrast to obtaining a detector of a parameter $\theta$ as in Eq. 3.9, we can obtain a detector for a particular training sample indexed by $i$:

$$C(x, i) = \sum_{j \in SV} \alpha_j k_{\theta}(i, j) k_x(x_j, x)$$

$$= \sum_{j \in SV} \alpha'_j(i) k_x(x_j, x).$$

(3.12)

The kernel $k_{\theta}(i, j)$ gives high weights to support vectors that are similar to $i$. Intuitively, those support vectors that are more similar to $i$ should be weighted more highly when we are constructing a detector for foreground objects similar to the training example indexed by $i$.

It is possible that a distance measure does not guarantee a valid Mercer kernel in Eq. 3.11, which should always yield a positive semi-definite Gram Matrix. However, when the Gram Matrix based on $k_{\theta}$ is positive semi-definite for the training set, we can still apply it, since in detection only $k_x$ is evaluated as in Eq. 3.9 and Eq. 3.12. If the Gram matrix has small negative eigenvalues, we can either adjust $\eta$ in Eq. 3.11, or employ a method to replace the negative eigenvalues with zeros \cite{54}. In our experiments, adjusting $\eta$ in Eq. 3.11 to make the Gram matrix positive definite works in all tests and yields satisfactory results.

### 3.3 Detector Training

In this section, we give details on how to train the model defined in the previous section. A bootstrap training process is proposed first. Then, we describe how to incorporate image
masks in training, if they are available. This can help reduce the influence of background clutter and can also enable foreground object segmentation during detection.

3.3.1 Bootstrap Training

For training we are given a set of foreground and background training samples. The training samples take the form of tuples – \((x, \theta)\) or \((x, i)\). Each foreground sample \(x\) is associated with its corresponding groundtruth \(\theta\) (parametric case) or its sample index \(i\) (nonparametric case). A background training sample \(x\) can be associated with any foreground parameter or index of a foreground training sample to form a valid tuple. The number of such combinations can be huge. We therefore employ an iterative process of bootstrap training to avoid combinatorial complexity while maintaining the desired detection accuracy.

The training process starts with assigning each background feature vector \(x\) a foreground parameter \(\theta\) or index \(i\) of a randomly selected foreground training sample. Then in each iteration the background examples are evaluated by recently trained detectors. The misclassified background samples that yielded highest detection scores are collected in tuple form, \((x, \theta)\) or \((x, i)\), where \(\theta\) or \(i\) is the corresponding parameter or sample index associated with the detector that misclassifies a background sample \(x\). In each iteration, a fixed number \(N_s\) of top-scored background tuples is added to the training set for SVM training in the next iteration. The iterative process ends when the margin between two classes does not increase or the maximum number of iterations has been reached. The pseudo code for this process is given in Fig. 3.2.

3.3.2 Including Object Masks in Training

There are situations in practice when object masks can be obtained during training data acquisition, for instance, when the background is known. In such cases, masks can be exploited to reduce the influence of background regions inside the detection window during both training and testing.
Given
Foreground training samples $X_f = \{x_{f1}, \ldots, x_{fn}\}$,
Background training samples $X_b = \{x_{b1}, \ldots, x_{bn}\}$,
Parameters of foreground training samples $\Theta = \{\theta_{f1}, \ldots, \theta_{fn}\}$.

Initialize
Assign each foreground training sample $x_{fi}$ its actual parameter $\theta_{fi}$, and each background training sample $x_{bi}$ a random value from $\Theta$ to form tuples $(x, \theta)$ for training. If object masks are available in foreground training data, the mask of $\theta$ is applied on $x$.

While
1. Compute Gram matrix $K_g$ of all training samples, where $k_g(i, j) = k_\theta(\theta_i, \theta_j)k_x(x_i, x_j)$.
2. Carry out SVM training using $K_g$, and obtain individual detectors for each $\theta \in \Theta$ according to Eq. 3.9.
3. Apply all the detectors to background training samples $X_b$. The evaluated background training samples are in the form $(x_{pi}, \theta_{pi})$, where $\theta_{pi}$ is the parameter of the individual detector that accepts $x_{pi}$.
4. If the margin between two classes is not increasing or the max number of iterations reached, break.
5. $N_s$ top scored background tuples $(x_{pi}, \theta_{pi})$ comprises a new set of background training tuples. Expand current set of training tuples with this new set of background training tuples.

end

**Figure 3.2:** Pseudo code for bootstrap training with parametric within-class kernel $k_\theta$. For the case of nonparametric $k_\theta$, the set $\Theta$ is replaced by the set of indices of foreground training samples.
When each training sample has a mask, features from outside the mask can be ignored. For instance, to calculate the color histogram of a foreground object, only those pixel colors from inside the mask region should be considered. When the features have local supports and are ordered according to their spatial arrangement, e.g., Histogram of Oriented Gradients (HOG) [18] or Haar wavelet features [80], applying object masks in feature extraction means that the feature components that have supports from outside the object masks have zero values.

To be consistent, masks should also be applied to the background training samples during feature extraction. As mentioned earlier, each background training sample is associated with a randomly chosen foreground parameter $\theta$ or an index $i$ of a foreground training sample. Therefore, the mask of the foreground training sample with $\theta$ or $i$ is applied to this background training sample.

Once a detector is associated with a mask, segmentation can be produced by superimposing the detector’s image mask onto an accepted image patch during detection. The image mask of a detector is first calculated as a weighted sum of image masks from foreground support vectors using the support vector weights $\alpha'_i$, and then obtained by thresholding.

Masks do not have to be explicitly applied in testing when both of the following two conditions are met:

1. The features have local supports like HOG [18] or Haar wavelet features [80].

2. $k(x_1, x_2)$ is based on the dot product $x_1^T x_2$, e.g., the linear or polynomial kernels.

This is true because during detection, when $x_1$ is a support vector and $x_2$ is the input feature vector, calculating $x_1^T x_2$ automatically zeros out the features of $x_2$ from outside of $x_1$’s mask. This is equivalent to excluding features of $x_2$ from outside $x_1$’s mask during detection.

We should also mention that object masks were also used in previous work [13, 87] where image segmentation and detection can be achieved jointly. However, in our method no decomposition of the image mask into local edgelets or image patches is needed.
3.4 Detection and Foreground State Estimation

After training as described in Sec. 3.1, we are able to construct a detector for any parameter $\theta$ or any foreground sample index $i$. However, in a real world application like multi-pose face detection, neither object locations (there could be multiple faces or none in an image) nor object foreground states are known. Thus, during detection, a scanning window process is employed using detectors associated with a predefined representative set of $\theta$ or $i$, which covers typical foreground state variations. The foreground state annotation $\theta$ associated with the detector of the highest detection score is assigned to a detected object as a foreground state estimate. As will be described in the rest of this section, there are a number of ways to determine the representative set of $\theta$ or $i$ that is used in generating the set of detectors. We will focus on two methods: uniform sampling over the training set, and finding representative samples via mode finding (clustering).

3.4.1 Generating a Sample Set of Detectors

Assume that the training set provided a fair and representative sampling of the foreground class. If the foreground states are annotated as parameters $\theta$, e.g., view angles or rotation angles, a representative set of $\theta$ can be obtained by uniformly sampling from the parameters of foreground training examples. In special cases when prior information about the parameter distribution is provided, e.g., in object tracking where temporal information is propagated from frame to frame, importance sampling can be employed instead to draw parameter samples to comprise a representative set of $\theta$. In our experiments, we obtain satisfactory results via uniform sampling for detection and parameter estimation applications, and via importance sampling for tracking applications. In the nonparametric case, uniform sampling over the foreground training samples can also be used to generate the detector family, assuming that the training set provides a fair and representative sample of the foreground class. However, we have found that a mode finding technique is more effective in practice.
3.4.2 Mode Finding for Nonparametric Detectors

In the nonparametric case, uniform sampling can also be used to generate a representative set of $i$. However, we can use mode finding to reduce the chance that similar examples of the foreground class are selected multiple times. Thus, the representative set of detectors can be more compact and the detection process can be more efficient.

Clustering can be employed for mode finding, where a similarity measure $S_\alpha$ is defined on the support vector weights $\alpha'(i)$ of foreground examples,

$$S_\alpha(i, j) = \frac{\alpha'(i)^T \alpha'(j)}{||\alpha'(i)|| \cdot ||\alpha'(j)||},$$  \hspace{1cm} (3.13)

where $\alpha'(i) = [\alpha'_1(i), \alpha'_2(i), \ldots, \alpha'_n(i)]^T$ are the support vector weights for $i$, as defined in Eq. 3.12. Each cluster is regarded as a mode that represents a variation of the foreground class. The proper number of modes is decided via cross-validation to obtain acceptable detection accuracy. The resulting modes can be used to generate a representative sample set for use in the detection stage. This sample set can be defined as the detectors associated with the cluster centers, or it can obtained via importance sampling from the modes.

Interestingly, the modes identified through clustering can also be used in efficient labelling of the foreground state for training samples. At the detection stage, only those detectors associated with representative training examples are used; therefore, only these representatives need be labelled. For example, in a hand detection application, if the user also wants to know whether a detected hand is in an open hand shape, a fist hand shape or a pointing hand shape at detection stage, the user only needs to label foreground training examples in the representative set with these labels. Once a hand is detected, the label associated with the detector of the highest score indicates the hand shape. Furthermore, if continuous parameter values are annotated, then their corresponding detectors can be used in the same way as parametric detectors, i.e., in estimating the foreground state parameters for a detected object, as well as in tracking. An obvious advantage of this strategy is that only a small portion of the foreground training data must be labelled. This can
save a significant amount of effort that might be needed to label all training samples for the same purpose.

3.5 Tracking with Multiplicative Kernel Detectors

Tracking objects that undergo large appearance changes is challenging, e.g., tracking articulated objects like human hands or multi-pose objects like faces and vehicles. Commonly used cylinder models [69] or edge templates [74] usually require strong temporal models and manual initialization to achieve robust tracking, particularly in cluttered scenes.

One way to cope with a cluttered background is to use detectors that are trained against representative background examples. Such a strategy was employed in the in “tracking-by-detection” approaches of [41, 2], where the tracking performance is enhanced by using the detectors that handle cluttered background and variations of the foreground class.

Our detectors trained with multiplicative kernels can be also employed to in a tracking-by-detection framework. A brute force way to implement tracking with parametric detectors that are trained with multiplicative kernels is via frame by frame detection. Although the object locations and foreground states can be recovered in this way, it can be expensive to run a dense scan on each frame with all detectors. We instead propose a tracking approach that incorporates temporal information to make the tracking process more efficient.

We formulate the tracking process in a standard prediction-update framework as in particle filtering and CONDENSATION [32]. For an existing object, given its observations \( Z_t = (z_1, \ldots, z_t) \) up to time \( t \), we estimate the current state \( s_t \) by the following steps:

1. Prediction: \( p(s_t|Z_{t-1}) = \int p(s_t|s_{t-1})p(s_{t-1}|Z_{t-1})ds_{t-1}, \)

2. Update: \( p(s_t|Z_t) \propto p(z_t|s_t)p(s_t|Z_{t-1}). \)

We define \( s_t = (l_t, \theta_t) \), where \( l_t \) is the location (including scale) and \( \theta_t \) is the pose parameter. We assume independence between \( l_t \) and \( \theta_t \). Thus,

\[
p(s_t|s_{t-1}) = p(l_t|l_{t-1})p(\theta_t|\theta_{t-1}).
\]

(3.14)
During importance sampling, \( s_t \) is factorized into \( l_t \) and \( \theta_t \) to reduce the number of dimensions of samples. In practice such factorization is reasonable, since position and \( \theta \) tend to be independent. We also assume zero mean Gaussian distributions for both \( p(l_t|l_{t-1}) \) and \( p(\theta_t|\theta_{t-1}) \), i.e., \( l_t - l_{t-1} \sim N(0, \Sigma_l) \) and \( \theta_t - \theta_{t-1} \sim N(0, \Sigma_\theta) \). The covariance matrices \( \Sigma_l \) and \( \Sigma_\theta \) are chosen according to the typical state changing speed of foreground objects. The Gaussian distribution assumption follows a common choice for the proposal distribution in the particle filtering framework [32].

In the update step, \( p(z_t|s_t) \) is evaluated using our detectors, i.e., given a sample \( \hat{s}_t = (\hat{l}_t, \hat{\theta}_t) \), the detector associated with \( \hat{\theta}_t \) is evaluated at location \( \hat{l}_t \) to give a score \( C(\hat{z}_t, \hat{\theta}_t) \) that determines whether the observation \( \hat{z}_t \) at location \( \hat{l}_t \) should be accepted or rejected as an instance of the object with parameter \( \hat{\theta}_t \). The sample \( \hat{s}_t \) is discarded if the detector classifies it as from the background class. We define \( p(z_t|s_t) = \exp(C(z_t, \theta_t))/w \), where \( w \) is a constant to scale \( \exp(C(z_t, \theta_t)) \) to \([0,1]\). Non-maximum suppression can be applied on locations of accepted samples to produce a set of putative locations for tracked objects in the current frame.

In our tracker implementation, to deal with the entrance of new objects, exhaustive detection is triggered at every \( k \) frames. The parameter \( k \) is selected according to the expected entrance rate for new objects. Once a foreground object is detected during exhaustive detection, a tracking process starts to track it until it exits the scene. Exitance of objects is also automatically handled; once an object exits the scene, samples that are not located on foreground objects in the next frame will be rejected by the detectors.
Chapter 4

Speedup Strategy for the Classification Process

Although running a family of detectors can improve detection accuracy over using a single detector, in practice it may not be as fast as running a single detector. One straightforward way to speed up the detection process is to employ a multi-stage cascade detector structure [80], i.e., adding initial detection stages that reject trivial background patches quickly.

Interestingly, if we use detectors trained by Adaboost in the initial detection stages, the outputs of weak classifiers can be reused in a filter step to eliminate inappropriate detectors from the family of detectors. Thus, the actual number of detectors to run on an input can be further reduced from the sampled set of detectors. The detection-recognition process becomes a “query-sensitive” process, where the selection of detectors depends on the input.

The computational complexity may also be a problem in other recognition tasks where a large number of hypotheses must be examined. For example, consider a face recognition system where each detected face is compared with hundreds or thousands of face IDs. Common methods that employ nearest neighbor search [84] or large margin classifiers [25] can be slow. In our experiments, face identification via one-versus-all (OVA) SVM classifiers of 535 subjects takes more than two seconds per detected face. A nearest neighbor approach could be even slower on this data set, because the total number of faces (18,000) is much larger than the number of subjects (535). If we use this system to recognize terrorists at a train station, the face detector could easily output dozens of faces per second during rush hours. A recognition speed of two seconds/face means a long waiting list of detected faces or dropping detected faces in a real time system.
In the proposed approach, we assume that the initial detector is a boosted cascade detector [80, 81, 93]. In addition, we assume that there are foreground within-class classification strategies to rerank foreground state hypotheses at the refine step, e.g., multiclass classifiers of face IDs, a database of annotated foreground examples for a nearest neighbor approach, etc. Our goal is to design a fast filter step to identify a small number of foreground state hypotheses for a given input. The basic idea is to reuse weak classifier evaluations from the initial detector.

It may seem surprising that a boosted cascade detector’s weak classifiers can also be helpful in foreground within-class classification. Detector training only optimizes accuracy in discriminating foreground vs. background. Yet, as we will soon see, the weak classifier outputs from a boosted cascade detector can be used to construct a Hamming distance that performs well as a filter step for foreground within-class classification.

In the following sections, we will explain our approach in detail. We first show how the cascade detector’s weak classifiers are related to locality sensitive hashing (LSH) [30] functions, which enable approximate nearest neighbor search in the feature space. We then show how to construct a Hamming code using a subset of the cascade detector’s weak classifier outputs that is optimized for foreground within-class classification.

4.1 From Random Binary Weak Classifiers to LSH

In the traditional Adaboost-based method [64], a strong binary classifier $H(x)$ is constructed as a weighted combination of weak classifiers that are selected from a pool of weak classifiers $h_i(x)$, with corresponding weights $\alpha_i$:

$$H(x) = \sum_{i=1}^{n} \alpha_i h_i(x)$$ (4.1)

where $x \in X$ is a feature vector, and $h_i(x) \in \{-1, +1\}$ can be simple decision stumps [80] or linear classifiers [93]. In our approach, each $h_i$ is assumed to be a domain bipartitioning classifier. Therefore, each $h_i$ is equivalent to a hyperplane that divides the feature space into two regions and assigns the input $x$ a binary value $+1$ or $-1$, depending on which side
of the hyperplane \( x \) locates.

We are going to show that those \( h_i(x) \) that are random bipartitioning hyperplanes follow the definition of hashing functions in Locality Sensitive Hashing (LSH) \([30]\). Thus, they can be used to construct a Hamming distance that approximates nearest neighbor search in the Euclidean feature space. In LSH, a family \( \mathcal{H} = \{ h : X \to \pm 1 \} \) of functions over \( X \) is called \((r_1, r_2, p_1, p_2)\)-sensitive for a distance measure \( D_x \), if for any \( x_1, x_2 \in X \)

- if \( D_x(x_1, x_2) \leq r_1 \), then \( \Pr(h(x_1) = h(x_2)) \geq p_1 \),
- if \( D_x(x_1, x_2) > r_2 \), then \( \Pr(h(x_1) = h(x_2)) < p_2 \).

For a locality-sensitive family \( \mathcal{H} \) to be useful, it must satisfy \( r_1 < r_2 \) and \( p_1 > p_2 \).

We use the following observation: in an Euclidean space, the probability \( p \) that two points \( x_1 \) and \( x_2 \) are separated by a random hyperplane increases monotonically with their Euclidean distance \( d = D(x_1, x_2) \). Thus, we have \( p = f(D(x_1, x_2)) \), where \( f \) is a monotonically increasing function and its value is in the range \([0, 1]\).

If we define \( h^*(x) = \pm 1 \) according to which side of the random hyperplane \( x \) is located, we have

\[
\Pr(h^*(x_1) \neq h^*(x_2)) = f(D(x_1, x_2)).
\] (4.2)

Thus, for any \( r_1 < r_2 \), we have

- if \( D(x_1, x_2) \leq r_1 \), then \( \Pr(h^*(x_1) = h^*(x_2)) = 1 - \Pr(h^*(x_1) \neq h^*(x_2)) \geq 1 - f(r_1) \),
- if \( D(x_1, x_2) > r_2 \), then \( \Pr(h^*(x_1) = h^*(x_2)) = 1 - \Pr(h^*(x_1) \neq h^*(x_2)) < 1 - f(r_2) \).

Let \( p_1 = 1 - f(r_1) \) and \( p_2 = 1 - f(r_2) \), then we have \( (r_1, r_2, p_1, p_2) \) that satisfy \( r_1 < r_2 \) and \( p_1 > p_2 \). Therefore, \( h^*(x) \) is a valid hashing function for LSH.

We define a binary string representation \( B(x) \) as the collection of binary outputs of the weak classifiers:

\[
B(x) = \{ h_1(x), h_2(x), \ldots, h_n(x) \}.
\] (4.3)
We define $D_r(x_1, x_2)$ as the Hamming distance between two binary strings $B(x_1)$ and $B(x_2)$ when $h_k$ are random weak classifiers. Retrieval with $D_r$ and a distance threshold $d_H$ is the special case of LSH that approximates the nearest neighbor search in the Euclidean feature space.

Although the weak classifiers collected for a detector are not purely random, it has been noticed [80, 76] that in a bootstrap training process of a cascade detector, the background training samples are more and more similar to the foreground samples, as the cascade stage goes deeper and deeper. The weak classifiers tend to have accuracies close to 50%, similar to random partitions. The Adaboost training process also makes the selected weak classifiers less correlated, because a weak classifier selected in an Adaboost iteration focuses more on training examples that cannot be correctly classified in previous iterations. We define $D_c$ as the Hamming distance that uses weak classifiers from the detection stage. In our experiments, filter-refine with $D_c$ achieves retrieval accuracy close to or even better than the Hamming distance $D_r$ that is based on random partitions.

On the other hand, some $h_k$ included in a cascade detector may not be useful for foreground within-class classification. We therefore propose optimization schemes that extract useful $h_k$ from those in a detector for specific within-class classification tasks.

### 4.2 Optimized Hamming Distance Measure

In this section we propose boosting algorithms to optimize selections of $h_k$ for a specific within-class classification task. The optimized distance measure is a Hamming distance, or a weighted Hamming distance, where each bit is weighted by a real value. Either of these two distance measures can be used in a fast filter step to eliminate implausible foreground state hypotheses quickly.

Intuitively, a good distance measure puts preferable neighboring objects closer to a query than unpreferable ones. For instance, consider continuous parameter estimation problems, like pose estimation [1, 10] or model alignment [48]. These problems can be defined as ranking problems when nearest neighbor approaches [4] or gradient descent
methods [88] are applied. When a nearest neighbor approach is used, the parameter of a more preferable neighbor is closer to the true parameter of the query than a less preferable one. Whereas in discrete classification problems, like face recognition [57], the preferable neighbors of a query are those items that have the same class label.

To optimize a distance measure for ranking problems, previous work [4, 88] proposes using triples $(q, a, b)$ as training examples, where $q$, $a$ and $b$ are samples from the foreground training set. In each triple, $a$ is a more preferable neighbor to $q$ than $b$. In training, a distance function is optimized to always put $a$ closer to $q$ than $b$. Another previous work [46] proposes using pairs $(q, a)$ as training examples for discrete classification problems. Each pair $(q, a)$ is assigned a label $+1$ or $-1$ to indicate whether $a$ is from the same class as $q$ or not. In training, a distance function is optimized to always put pairs of the same class closer than those of different classes. The method of [46] is only relevant to discrete classification. Therefore, we adopt training with triples in our solution, because it can be applied to both parameter estimation and discrete classification problems.

The inputs to our training approach are the following:

1. A training set $S = \{(q_1, a_1, b_1), \ldots, (q_t, a_t, b_t)\}$ of $t$ triples of foreground examples. $q_i$, $a_i$ and $b_i$ are all foreground examples. In each triple, $a_i$ is a more preferable neighbor of $q_i$ than $b_i$.

2. A set of binary functions $B = \{h_1, \ldots, h_n\}$, where $h_k(x) \in \{-1, 1\}$.

Each $h_k$ induces a distance measure

$$d_k(x, y) = |h_k(x) - h_k(y)|/2$$  \hspace{1cm} (4.4)

and a weak classifier $f_k$ (Note $f_k$ is defined on triples, different from $h_k$):

$$f_k(q_i, a_i, b_i) = d_k(q_i, b_i) - d_k(q_i, a_i),$$  \hspace{1cm} (4.5)

where $d_k(x, y) \in \{0, 1\}$ and $f_k(q_i, a_i, b_i) \in \{-1, 0, +1\}$. Our goal in training is to find a
strong classifier
\[
F(q, a, b) = \sum \beta_j f_j(q, a, b),
\]
(4.6)
such that \( F(q, a, b) > 0 \) for all triples \((q, a, b)\). If we define a new distance measure
\[
D_w(x, y) = \sum \beta_j d_j(x, y),
\]
(4.7)
and plug Eqn.(4.5) in Eqn.(4.6), we have
\[
F(q, a, b) = \sum \beta_j f_j(q, a, b) = \sum \beta_j (d_j(q, b) - d_j(q, a)) = D_w(q, b) - D_w(q, a) > 0.
\]
(4.8)

Eqn. (4.8) shows that a \( F \) that always assigns a positive value to a triple \((q, a, b)\) implies a perfect \( D_w \) that always puts a more preferable neighbor \( a \) closer to \( q \) than \( b \). Thus, we can obtain an optimized distance measure \( D_w \) for a specific foreground classification task. The training process to find optimal \( \beta_j \) and \( f_j \) in Eqn. (4.6) follows a standard Adaboost algorithm. The process stops when no more weak classifiers can be added to reduce the training error. If the same \( f_j \) are selected multiple times, their weights are summed to a single \( \beta_j \) to keep all \( f_j \) in \( F \) distinct.

There is a one-to-one correspondence between \( f_k \) and \( h_k \). We call \( \hat{B}(x) \) an optimized binary string representation,
\[
\hat{B}(x) = \{ h_j(x), \text{where } f_j \text{ is selected for } F \} \subseteq B(x).
\]
(4.9)
We call the distance \( D_w \) in Eqn. (4.7) an optimized weighted Hamming distance, since each dimension \( h_j(x) \) is weighted by a real number \( \beta_j \).

We are also able to obtain an optimized Hamming distance without real weights \( \beta_j \). There are only two things that we need to modify in the training process. First, there is a new constraint that \( \beta_j = 1 \). In each iteration, we select an \( f_j \) that reduces the training
error most, but fix its weight $\beta_j = 1$. Second, at the end of each boosting iteration, the selected weak classifier $f_j$ is removed from the pool of all weak classifiers for following iterations. We denote this optimized Hamming distance as $D_h$.

The above distance optimization scheme considers only those weak classifiers that were included in the cascade detector. We could instead construct our optimized distance by selecting weak classifiers from the entire set that was available for training the detector. It would be expected that this distance measure might perform better in filter-and-refine retrieval, since distance construction is not limited only to those classifiers included in the detector. We define $D_a$ to be the weighted Hamming distance obtained by selecting a subset from all weak classifiers.

In our experiments, the training process of $D_a$ is very slow. The bottleneck is weak classifier selection in each iteration, as noted in [56, 89]. Speedup strategies [56, 93] that find the best weak classifier deterministically using statistics of training examples cannot be applied, since the same training example can be $a$ in one triple, but $b$ in another triple. On the face data set, we tried a fast feature selection strategy proposed in [89] that stores weak classifier responses of all training samples in a table, which are reused in each iteration. Furthermore, the feature set was reduced to $1/10$ of its original size by uniform sampling. The training process of $D_a$ still runs for about eight hours, in contrast to 25 seconds if we only consider those weak classifiers that were included in a trained detector.

### 4.3 Implementation

We train a cascade detector of the foreground class by Adaboost. Then, an optimized binary string representation $\hat{B}(x)$ is obtained as described in the previous section.

A table $T$ is constructed to store binary strings $\hat{B}(x)$ of foreground training examples. Each row corresponds to a unique binary string. If multiple foreground training samples have the same binary string, they are stored in the same row, along with the corresponding groundtruth annotations, e.g., face IDs.

During detection, if an input is accepted by the cascade detector, its binary pattern
$\hat{B}(x)$ is compared with all the rows in table $T$ by a fast distance measure ($D_w$ or $D_h$) proposed in Section 4.2. The foreground state hypotheses associated with top $k$ nearest neighbors (or those within a distance threshold) of the input are passed to the refine step.

The following is a summary of the online stage for the example application of face detection and recognition:

1. **Detect**: $x$ is input to the cascade detector, which uses a standard “sliding window” approach.

2. **Filter**: If $x$ is detected as a face, $\hat{B}(x)$ is compared with all rows in table $T$ by a proposed optimized distance measure ($D_w$ or $D_h$). Candidate face IDs are those of top $k$ nearest neighbors of $x$ or those within a certain distance threshold from $x$. These are candidate face IDs for the refine step.

3. **Refine**: Apply OVA classifiers of the candidate face IDs on the corresponding feature representation of $x$. The face ID of the classifier that achieves the highest score is assigned to the input.
Chapter 5

Experiments

In this chapter we describe the evaluation of the two proposed approaches: detector learning with multiplicative kernels and the speedup strategy that reuses features from detection stage. The approaches are implemented in Matlab, and run on a 2.6GHz AMD Opteron 852 processor.

5.1 Experiments on Detectors Trained via Multiplicative Kernels

We evaluate the proposed multiplicative kernel method in three applications: hand detection and shape estimation, multi-pose vehicle detection and tracking, and multi-pose face detection and tracking. For the purpose of these experiments, HOG [18] features are employed for $\mathbf{x}$ on all data sets; while other features could be possible, we chose the HOG feature representation since it is widely used. The detectors of our method are trained using a modified version of SVMlight [33] with multiplicative kernels. The between-class kernel $k_x$ is always a linear kernel, and the within-class kernel $k_\theta$ is a Gaussian RBF kernel or a nonparametric kernel (Eq. 3.11) depending on the data set. Our results are compared with results obtained via methods proposed in [52, 75, 86, 91].

In the detection process, we use a standard scanning window process as used in [80, 18], in which all image sub-windows captured at different image scales are normalized to the size of the detection window.

5.1.1 Hand Detection and Segmentation with Nonparametric $k_\theta$

Hand detection and shape estimation is an important component in Human Computer Interfaces (HCI), e.g., a gesture-based game interface, a sign language recognition system,
etc. Due to large variations of hand appearances, previous work [36, 42] relied heavily on skin color or motion features, which can generate ambiguity between the hands and other moving body parts like the arms and face. Recent work [52] proposed a partition-based approach that yields promising results for hand detection in static images. In this method, the hand class is partitioned via unsupervised clustering based on the shape context similarity measure [8]. A detector is then trained for each cluster. Although such a divide-and-conquer strategy improves detection accuracy, there is no feature sharing between clusters.

In our approach, the detectors are trained in a way that allows features sharing among all hand training examples. Furthermore, segmentation of the detected hand can also be obtained when hand silhouettes are provided in training.

To test our approach in this application setting, we first conduct an experiment in hand detection for sign language data. The hand data set is collected from two sources of sign language video sequences: Flemish Sign Language data [17] and American Sign Language data [50]. In total there are 17 signers. The data set comprises a training set of 3,005 hand images and a test set of 2270 hand images. The test set and training set are disjoint. The hand images are not annotated with hand shape parameters. For the training images, corresponding hand silhouettes are also provided. About 70% of the hand silhouettes are automatically segmented by skin color models or simple background models. The rest are obtained manually. Example frames are shown in Figure 5-1. This data set is available.

**Figure 5-1:** Example sign language sequences from which the training and test hand images are obtained.
The background image set contains images of outdoor and indoor scenes. This set is separated into disjoint training and test sets, which contain 300 images each. 5000 image patches are collected as samples from each image set to be used as training or testing background samples.

HOG features are useful feature representation for hand detection and they are used in this experiment. To extract HOG features, each image is normalized to a 48 by 48 pixel image, which is then divided into overlapping cells of size 6 by 6. Neighboring cells overlap by half. Bins in each cell are normalized with the surrounding 3 by 3 cells using the 2-norm as in [18].

Training of detectors is done as described in chapter 3. As we mentioned earlier, we define $k_x$ as a linear kernel. For the within-class kernel $k_\theta$, the nonparametric form of Eq. 3.11 is used, since parameter annotations are unavailable for the training images. The distance measure $D$ is the bidirectional chamfer edge distance [23] between hand images. With $\eta = 1$, the Gram matrix of $k_\theta$ is positive definite on the training set. The mode finding process as described in chapter 3 is used to generate the detector sample set for 1242 hand modes. The number of modes is determined by the stopping criterion in agglomerative clustering, when the similarity measure of Eq. 3.4.2 between any two clusters is below a threshold value 0.7. The threshold is selected via cross validation of detection accuracy on training examples. The total training time is about 30 minutes on a single 2.6GHz AMD Opteron 852 processor.

Six out of the 1242 hand clusters are illustrated in Figure. 5-2. The figure shows three images for each cluster: the image of the cluster medoid, the positive weights of the detector associated with the cluster medoid, and the mask for the medoid. The positive weights of a detector demonstrate how local edge orientations are weighted. The image mask of a cluster is computed as a weighted sum of image masks of support vectors for the top 50 weights, and then thresholded to obtain a binary image. While there could be different

\footnote{available at http://cs-people.bu.edu/yq/projects/mk.html}
Figure 5.2: Example hand clusters after training with nonparametric multiplicative kernels. Six hand clusters are displayed with their cluster medoids, positive detector weights and hand masks. For each cluster the weights of foreground support vectors are displayed at bottom.

ways to construct an image mask, in our experiment, the resulting masks have appropriate sizes and shapes for this setting.

For each cluster medoid shown in Figure. 5.2, a graph shows the distribution of support vector weights $\alpha'$. Interestingly, although the weights have peaks on a few foreground support vectors, the sum of weights from low weight support vectors is substantial. This indicates that the contributions to the detector of a particular foreground variation come
Figure 5-3: Example detection and segmentation results for the sign language test sequences.

from a broad range of training samples, although each contribution may be small. One explanation is that very different hand shapes may still share segments of finger or palm boundaries.

Examples of the combined detection and segmentation results obtained with our method are shown in Figure. 5-3. The segmentation result is obtained by applying the mask associated with the detector of the highest score on a detected hand. The segmentation obtained in this way is only approximate; nonetheless, the shapes are matched well and the segmentation is obtained at nominal extra cost. The segmentation result from our method can be used to mask the image for a hand shape estimation module in sign language analysis or used as initialization to a method that requires segmented input.

When the detectors are applied on the frames of the Flemish and American sign language sequences [17, 50], they can detect most of the hand shapes. The detectors may fail to detect a hand when there is strong motion blur or it is partially occluded. False positives happen occasionally in regions of strong textures.

Partition-based approaches, e.g., [52], have been applied to hand detection. In [52], the hand class is partitioned into subclasses via k-means clustering using the shape context [8] similarity measure. Then a detector is trained for each subclass. For experimental comparison, a partitioning-based method is formulated and trained as follows: first clustering of hand subclasses is obtained via k-means with Euclidean distance of HOG features, then the detector for each subclass is trained using SVM with a RBF kernel. The $\eta$ of the RBF
The features from outside a subclass mask are ignored during training and testing of the subclass detectors. The accuracy of the partition based method improves as the number of partitions increases to around 50 partitions. Further increases of the number of partitions do not yield significant improvement.

The detection accuracy of the different methods is summarized in the ROC curves of Figure 5-4. As can be seen in the graph, our method outperforms the partition-based methods by a clear margin on this data set. Compared to the best partition-based method (50 partitions), our method improves detection rate from 80% to 90% at a false positive rate of 5%. At the detection rate of 80% our method reduces the false positive rate from 5.3% to 1.7%.

To better understand the accuracy tradeoff in using the representative subset of detec-
tors determined via mode finding, we compared performance against using all detectors. At a fixed false positive rate of 5%, when all detectors are used (3,005 in total), the detection rate is 90.1%. With our mode finding approach, the detection accuracy is 90.0%, 88.4% and 83.5% with 1,242 modes, 938 modes and 300 modes, respectively. The detectors used in our approach (1,242 detectors) achieve the same accuracy, while reducing the number of detector evaluations by about two thirds. In contrast, when we use uniform sampling to obtain 1,242 detectors, the average detection accuracy over ten trials is 88.1% with a standard deviation of 0.49% at the false positive rate of 5%.

As we mentioned in chapter 3.2, a detection score also indicates how well the test instance matches the training example associated with this detector. This means that similar test instances should get similar detector responses and dissimilar test instances should get dissimilar detector responses. For pairs of test hand images, we measure the Pearson product-moment correlation coefficient between the similarity of image features and the similarity of detector responses as following:

- The image feature similarity between two hand images is defined as a dot product between their HOG feature vectors. Note the HOG feature vectors are normalized, so all vectors are on a hypersphere and the dot product indicates the angle between two examples.

- For each hand image, a vector of the detection scores from all 3,005 detectors is constructed and normalized to a length of one. The similarity of detector responses between two hand images is defined as a dot product between two vectors of the normalized detection scores.

The Pearson correlation coefficient between the image feature similarity and the similarity of detector responses is 0.519, which is calculated from all pairs of test examples. Although the exact interpretation of a correlation score may depend on the definition of the problem, a rough guideline for the interpretation of a correlation coefficient $r$ has been given in the work of Cohen [15]:

• Small positive correlation: $0.1 \leq r < 0.3$.

• Median positive correlation: $0.3 \leq r < 0.5$

• Large positive correlation: $0.5 \leq r \leq 1$

As can be seen, the correlation coefficient obtained in our experiment falls on the boundary between median positive correlation and large positive correlation.

5.1.2 Hand Detection and Shape Estimation with Parametric $k_{\theta}$

In the second experiment, we detect instances of a hand shape class that is parameterized by two angles from a cluttered background and estimate the two angles simultaneously. In the hand shape data set of [91], each hand image is given two angles within the range $[0,90]$ - one for the angle of the index finger, the other one for the in-plane rotation. There are 1,605 hand images for training and 925 for testing. There are also 5,500 background training samples and 50,000 background test samples, cropped from real background images or hand images of other hand shapes not included in the target hand shape class. Example hand images are shown in Figure 5-5.

![Figure 5-5: Examples from the hand data set [92].](image)

In the implementation of our method, HOG features are computed as in [91]. The two angle parameters $\theta_1$ and $\theta_2$ are both normalized to $[0,1]$. The between-class kernel $k_x$ is linear as before. The within-class kernel $k_{\theta}$ is a Gaussian RBF kernel in the two-dimensional parameter space, with $\frac{1}{\sigma^2} = 10$. After SVM training, 200 parameter values $\theta$ with corresponding detectors are uniformly sampled from the 1,605 parameter values associated with foreground training examples. This number of detectors is determined to
be adequate via cross-validation using training examples. These 200 detectors are used at the detection stage.

We compare the performance of our formulation with a boosting-based approach [91]. The ROC curves of the detection result (hand vs. background) are shown in Figure. 5-6. As can be seen from the ROC curves, our method consistently outperforms [91], which has already been shown to be superior to partition-based method [52] on the same data set in [91]. At a false positive rate of $2 \times 10^{-4}$, it improves the true positive rate from 94% of to 99%. The partition-based method [52] with 25 subclasses achieves a true positive rate of 91% at the same false positive rate. In terms of speed, both methods apply 200 detectors at the detection stage so the speeds are comparable. However, the training of the multiplicative kernel based method is about 10 times faster than the boosting based method [91].

In our approach, parameter estimation is achieved by assigning the parameter associated with the detector of the highest score. The mean absolute errors on the two finger parameters are 6.7 and 4.6 degrees respectively, in contrast to 9.0 and 5.3 degrees in [91].
The partition-based approach [52] with 25 subclasses does not produce a parameter estimate, but a subclass label that is within a range of 18 degrees.

Detection and angle estimation of this hand shape class is reliable with our method on this data set. False positives happen occasionally in highly textured regions.

5.1.3 Multi-pose Vehicle Detection

In the next experiment we look at a multi-pose vehicle detection problem. We evaluate the performance of the proposed method in two vehicle detection tasks, with comparison to previous approaches [75, 86]. In the first task, we detect vehicles appearing in city scenes. In the second task, we detect vehicles on highways.

For the first task, we use a multi-pose vehicle data set [38], which is a subset of LabelMe [63] database. This subset contains 1,297 vehicles images. Each vehicle image also has a binary segmentation mask converted from the LabelMe annotation polygon. In [38], the data is split up into seven subcategories for car viewpoints approximately 30 degrees apart. Because of vehicle symmetry, the labelled angles cover a half circle from approximately -30 degrees to 180 degrees. Example images from this data set are shown in Figure 5.7. These 1,297 vehicle images are separated into a training set of 866 images and a test set of 431 images. We collected background training and test image sets, which contain 432 and 344 outdoor street scene images, respectively. Most of the background images are from street scene images used in [18]. The rest are downloaded from web. The
background image data sets are available for download.\footnote{available at http://cs-people.bu.edu/yq/projects/mk.html}

In our approach, the nonparametric within-class kernel $k_\theta$ is an RBF kernel defined with Euclidean distance between HOG features. The kernel parameter $\eta = 0.2$. To extract HOG features, each image is normalized to a 90 by 90 pixels image, which is divided into 225 cells of size 6 by 6. Bins in each cell are normalized with the surrounding 3 by 3 cells using the 2-norm as in [18]. $k_x$ is a linear kernel. We implemented two versions of our approach. One is trained with binary image masks and the other one is without image masks. For both versions, the bootstrap training process takes 10 iterations. For both versions, 280 modes are obtained by spectral clustering (normalized cuts [68]) after training. The number of modes is again determined by cross-validation of detection accuracy.

Performance is compared with Torralba’s feature sharing method [75], Wu-Nevatia’s tree based detector [86] and a RBF kernel SVM classifier with $\eta = 0.2$. In each method, the parameter settings were determined so as to optimize detection accuracy. For [75], the view angle subcategory labels of training images are provided in training since it is a multi-class detection method. In training it adds 4,000 weak classifiers in total and outputs seven subclass detectors. Each subclass detector is for a view angle subclass. In [86] the tree structure is mainly controlled by a splitting threshold $\theta_z$. The best $\theta_z$ is found at 0.95 by cross-validation in this experiment, which produces a tree of eight leaf nodes. The weak classifiers collected along the path from the root to a leaf node comprise a detector for the subclass represented by this leaf node. The final numbers of weak classifiers in these eight subclass detectors are between 2032 and 2213.

For fair comparison, a bootstrap method is employed to collect non-trivial background examples for all methods, in the same way as in [18]. First a linear SVM classifier is trained with an initial set of 10,000 training background patches. Then we exhaustively search all the background training images with this linear SVM classifier to collect false positive image patches (“hard examples”). In the scanning process, a total of 2,211 false positive patches are collected. They are added to the initial 10,000 background training
The detection performance of all methods in the first task is shown in the ROC curves of Figure 5.8. Compared with [86], our method with image masks improves the detection rate from 96.7% to 99.0% at the false positive rate of $5 \times 10^{-3}$. At the detection rate of 99.5%, our method reduces the false positive rate from 5% to 0.8%. The speed of our method in this test is the fastest among the three competing methods. On average, it takes $1.85 \times 10^{-4}$ seconds for our method to evaluate a test example, in contrast to $4.40 \times 10^{-4}$ seconds.
seconds and $2.46 \times 10^{-4}$ seconds for [75] and [86], respectively. This is surprising, because in our approach we evaluated more detectors on each test sample than [75] and [86]. Further investigation shows that, in Matlab, the evaluation function via matrix multiplication in our approach runs much faster than the evaluation function which combines weak classifier outputs in [75] and [86]. The overhead of assembling weak classifiers slows down the speed of [75] and [86] in Matlab.

In Figure 5.9, we show false negative examples and false positive examples of our approach on the first task. The false negative examples are collected at a fixed false positive rate of $10^{-3}$. The false positive examples are collected at a fixed detection rate 95%. Because the HOG features are based on gradient orientations and vehicles have symmetric shapes when viewed from certain angles, the false positive examples also show symmetric patterns and strong edges.

In the second part of this experiment, we test our method on detecting vehicles on a highway. Different from the first task where the vehicles are mostly captured in urban scenes, Test Sequence 5 of the PETS 2001 vehicle data set is captured by two moving cameras on a highway, one facing front and the other one facing back. Example frames from two cameras are shown in Figure 5.10. In total there are 2,867 frames for each camera. Each frame is of size 768 by 576 pixels.

In these two test sequences, the vehicles that are moving in the same direction with the cameras (i.e., vehicles bound in same direction) tend to be close to the cameras, and they are imaged at good pixel resolution. It is more challenging to detect the vehicles that are moving in the opposite direction, on the other side of the highway. These vehicles appear at smaller pixel resolutions and are partially occluded by the highway guard rail. For evaluation purposes, we manually annotated vehicles of sizes no smaller than 45 by 45 and occluded by less than one third, in every 10th frame of each camera sequence.

As before, comparison is conducted between our approach and [75, 86]. For the purpose of fair comparison, we compared these methods with our detector without tracking. All methods detect vehicles frame by frame without temporal information. All 1,297 vehicle
images from [38] with their horizontally flipped images are used as training samples. The settings of our method are the same as in the first task. For the tree based method [86], the best splitting threshold is found again at 0.95, which produces a tree that has 19 leaf nodes. For the feature sharing method [75], view categories are provided and 4,000 weak classifiers are collected in training.

For evaluation, we consider a detection window as correct if it overlaps with the ground-truth annotation by more than 50% using the intersection-over-union criterion [44]. The detection performance of the three methods is summarized in Figure 5·11. Compared with Torralba’s method [75], our approach improves the detection rate from 40% to 60% for Camera 1 and 63% to 82% for Camera 2, both at the false positive rate of one per frame. The tree-based method [86] yielded consistently inferior performance to both [75] and our approach.

We also measured the detection rates at different object scales with a fixed false positive rate one per frame. The annotated test examples are put into four categories of sizes (by the side length $L$ of the square bounding box): $45 \leq L < 60$, $60 \leq L < 75$, $75 \leq L < 90$, $L \geq 90$. The detection rates of all three methods at these scale categories are plotted in
Figure 5.11: Vehicle detection rate vs false positive rate on sequence 5 of the PETS 2001 data set. The proposed approach (Multiplicative Kernel) is compared with Wu-Nevatia’s tree based detector [86] and Torralba’s feature sharing method [75].

the Figure. 5.12. As can be seen, for both sequences, the detection rates drop as the object scale goes down. For the proposed approach, the detection rate of the category $L \geq 90$ is 76% in the sequence 1 and 97% in the sequence 2, and the detection rate of the category $45 \leq L < 60$ is around 40% in both sequences. For the best competing approach [75], the detection rate of the category $L \geq 90$ is 58% in the sequence 1 and 80% in the sequence 2, and the detection rate of the category $45 \leq L < 60$ is around 20% in both sequences.

In general, the detection rates of all methods decrease as the scale goes down. One reason is that the HOG features are sensitive to image smoothing, as has been pointed out in [18]. Scaling down the size of an object has the same effect as image smoothing. Thus, the detection rate of objects of small scales is usually worse than that of objects of large scales.

With our approach, most of the mis-detections are due to small object scales and occlusions. False positives happen in textured regions, e.g., along the highway guard rail.

5.1.4 Vehicle Tracking and View Angle Estimation

In this experiment, we measure the vehicle orientation estimation accuracy in tracking. For evaluation, eight test vehicle sequences were downloaded from Google video. The test
sequences are of low frame rate - about 5 to 10 frames per second, with a pixel resolution of 320 by 240. These sequences exhibit strong motion blur artifacts and fast changes in object scale. There are eight distinct vehicles in the sequences, and each vehicle has at least 90 degrees view angle change. Most of the vehicles run on dirt roads and three of them are race cars.

In this experiment, our multiplicative kernel detector used in exhaustive search is the same detector trained for the vehicle detection task on the highway (Figure 5.11 and Figure 5.12). However, the original view angle partition of 30 degrees apart [38] is too coarse for view angle tracking. To get a better measure of view angle estimation accuracy, we need annotations of view angles at smaller intervals. In a similar angle estimation problem of hand pose estimation [5], it has been found that manual estimates by different people varied by 10–30 degrees for hand poses. Vehicles are slightly easier to annotate than human hands because of fewer degrees of freedom. We decide to divide the view angle of vehicles into 5 degrees apart. We annotated 280 vehicles in the training data and all vehicles in the test sequences at this angle interval, by having a user compare vehicles in the video sequences with images of a synthetic car model rotated at different angles.
Table 5.1: View angle estimation error in degrees, in eight different test sequences of vehicle tracking.

<table>
<thead>
<tr>
<th>Sequence ID</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of frames</td>
<td>32</td>
<td>31</td>
<td>39</td>
<td>64</td>
<td>40</td>
<td>16</td>
<td>59</td>
<td>43</td>
</tr>
<tr>
<td>MAE</td>
<td>13°</td>
<td>11°</td>
<td>15°</td>
<td>17°</td>
<td>8°</td>
<td>15°</td>
<td>14°</td>
<td>24°</td>
</tr>
<tr>
<td>Median-AE</td>
<td>10°</td>
<td>10°</td>
<td>15°</td>
<td>15°</td>
<td>5°</td>
<td>15°</td>
<td>15°</td>
<td>15°</td>
</tr>
</tbody>
</table>

We apply the tracking process explained in Chapter 3.5 on this data set. In this test, we assume that there is at most one vehicle in a sequence. Thus, extensive search with detectors is triggered at the first frame, and then triggered again when the tracker loses the target. Furthermore, during tracking the number of samples from a mode is proportional to the weight assigned to the mode. The sample propagation distribution in 2D image coordinates is a Gaussian with an isotropic $\Sigma_l$, which has a standard deviation of 3 pixels in each direction. The sample propagation distribution of the angle is a 1D Gaussian with a standard deviation of 30 degrees. The view angle estimate in the mis-detected frames is linearly interpolated from previous and later view angle estimates.

Example tracking results in four test sequences are shown in Figure 5.13. The resulting angle estimation accuracy during tracking is summarized in Table 5.1. The median of absolute error is in the range from 5 to 15 degrees for the eight test sequences. The mean of absolute errors is in the range from 8 degrees to 24 degrees. Such errors are reasonable and consistent with the reported errors (10 to 30 degrees) in a hand pose estimation system [5].

Two main causes of errors are motion blur and view angles that are not covered in the training examples. Although the tracker may lose the target due to these two reasons, the detectors can recover the target location and view angle automatically in later frames when observations are better presented. The tracking speed is about 2 seconds per frame, including the HOG feature extraction and detector evaluation, which were both implemented in Matlab without special optimization. The HOG feature extraction takes about 70% of the CPU time.
Figure 5-13: Four example sequences of car tracking. Sequences (a)(b)(c)(d) correspond to sequence IDs 3, 8, 1 and 7 respectively in Table 5.1. Synthesized views of tracked cars are displayed at the bottom. Green boxes highlight the errors in these sequences. In sequence (b), the initial detection in the first frame assigns the detected car a rear view, due to the ambiguity between front view and rear view. The error is corrected at subsequent frames when more frames are evaluated during temporal propagation. In sequence (c), the car is missed at frame 25 because the view point elevation is much higher than those in training images. In sequence (d), the car is missed at frame 54 due to motion blur. The view angle estimation in the miss-detected frames are linearly interpolated from previous and later detections.

5.1.5 Face Rotation Angle Estimation and Tracking

Multi-pose face detection is challenging due to the variation of face appearances at different view angles, in addition to other variations due to changes in illumination and facial expressions. A commonly used approach to detecting multi-pose faces is to divide the view angle space into partitions and train a different detector for each partition. In pre-
previous work [29], multi-pose face detection is achieved via partitioning the face class into subclasses according to face view angles. In [53] a face manifold is learned by encoding the face view angles to detect multi-pose faces. Both approaches, however, require a huge amount of training images (30,000 in [53] and 75,000 in [29]). Manual annotation of such a large amount of data is expensive and both face training sets in [29, 53] are not publicly available. In contrast, our multi-pose face detectors can be trained with much fewer training examples (fewer than 5,000) than [29, 53] because of implicit feature sharing.

In this experiment, we train and test our approach with a subset of the recently released CMU Multi-PIE data set [24]. The complete Multi-PIE data set contains face images from 337 subjects, imaged under 15 view points and 19 illumination conditions in up to four recording sessions. In our experiment, we use a subset of 13 views and 10 illuminations of the first 32 subjects. In total there are 8320 face images in the subset. The 13 view points are 30 degrees apart, as shown in Figure. 5.14. Face regions are manually annotated by us. Background training samples are collected from 1000 background images, containing indoor and outdoor images.

Our multiplicative kernel detector is trained with a nonparametric RBF kernel $k_\theta$ and linear kernel $k_x$. For $k_\theta$, the RBF is defined over the Euclidean distance of HOG feature vectors, with $\eta = 0.1$. To extract HOG features, each face region is normalized to the size of 60 pixels by 60 pixels. In HOG feature calculation, each cell is of size 4 by 4. The normalization block size is 3 by 3 cells.

For comparison, subclass detectors for 13 view angle subclasses as in Figure. 5.14 are trained by Torralba’s feature sharing method [75], with 2000 boosting iterations.

We evaluate the performance of face view angle estimation by 4-fold cross-validation.
on 32 subjects. That is, every time we train on 24 subjects and test on the remaining 8 subjects. Mean absolute error (MAE) is used as an evaluation metric for angle estimation accuracy. The comparison result is shown in Figure 5.15. The overall MAE of our approach is 2.1 degrees, in contrast to 3.0 degrees of Torralba’s feature sharing method. With our approach, 0.2% of the test samples have errors greater than or equal to 15 degrees. In contrast, 0.6% of the test samples have errors greater than or equal to 15 degrees with Torralba’s feature sharing method.

To demonstrate our tracking approach in this setting, we collected two video sequences with multiple faces in a lab environment. There are 117 frames in the first sequence and 179 frames in the second sequence. The frame size is 480 by 360 pixels for the first sequence and 648 by 488 pixels for the second sequence. In each sequence there are up to three faces in a frame. The faces make left-right out-of-plane rotations and slight in-plane rotations. For evaluation purposes, we manually annotated all face locations and their left-right rotation

**Figure 5.15:** Face view angle estimation result on the Multi-PIE data set. For each view angle subclass, we plot the mean and standard deviation of the errors on test samples. The overall mean absolute errors are 2.1 degrees and 3.0 degrees for our method and Torralba’s feature sharing method [75], respectively.
angles in every other frame of the test sequences. During annotation, the faces in the test sequences are compared with face images from the Multi-PIE training set to find matching face rotation angles.

We apply the tracking algorithm of Chapter 3.5 with multiplicative detectors on the two sequences. The training set for detectors that are used in tracking are the 4160 face images of first 32 subjects and first 5 illumination variations in the Multi-PIE data set. During tracking, the frames are rotated up to 15 degrees at 5-degree increments to compensate for in-plane rotations. The tracking process is fully automatic. Exhaustive search with all detectors is triggered at every 5 frames in the first sequence and every 10 frames in the second sequence to reset the tracker. The reset rate was determined so as to match roughly the entrance rate of faces. The number of faces is determined by exhaustive search. The sample propagation distribution in 2D image coordinates is a Gaussian with an isotropic $\Sigma_t$, with a standard deviation of 6 pixels in each direction. The sample propagation distribution of the angle is a 1D Gaussian with a standard deviation of 30 degrees. Example frames of the tracking result are shown in Figure 5.16. Most faces are detected correctly when their pitch angle is within the range $[-90^\circ,90^\circ]$. Most of the missed detections are due to large
Figure 5-17: ROC curves of face detection on two test sequences with a family of detectors. The proposed method achieves a detection rate of 80% on the first sequence and 90% on the second sequence. This difference might be attributed to the fact that faces are rotated outside the range $[-90,90]$ in patch angle more frequently in the first sequence.

In the graphs of Figure 5-17, we plot the ROC curves for face detection with our tracking method on the two test sequences. At a false positive rate of 0.1 false positives per frame, our method achieves a detection rate of 80% on the first sequence and 90% on the second sequence. This difference might be attributed to the fact that faces are rotated outside the range $[-90^\circ,90^\circ]$ in pitch angle more frequently in the first sequence. The MAEs of view angle estimation on detected faces are 3.50 degrees on the first sequence and 3.47 degrees on the second sequence.

The tracking speed is about 10 seconds per frame on the first sequence and 17 seconds per frame on the sequence sequence. Extensive search takes about 5 minutes per frame on the first sequence and 14 minutes per frame on the second sequence, using un-optimized Matlab code. About 74% of the total time is spent in HOG feature extraction.
5.2 Experiments on Speedup Strategy

In the second part, we describe the evaluation of the proposed speedup strategy on three data sets: the FRGC version 2 data set [57], the hand image data set [92] and the vehicle data set [38]. Approaches that are compared include: our methods (filter-refine using the optimized weighted Hamming distance $D_w$ and the optimized Hamming distance $D_h$), ClassMap [6], filter-refine with the Hamming distance using random weak classifiers $D_r$, the Hamming distance using all weak classifiers from the detector $D_c$ and the optimized Hamming distance trained with all possible weak classifiers $D_a$, brute force approaches (OVA classifiers or nearest neighbor), and support vector regression (SVR).

5.2.1 FRGC V2 Data Set

In this experiment, we use the same face data set as in [6], which contains all 2D face images in the FRGC version 2 data set. Example face images from this data set are shown in Figure 5.38. 36,817 face images from 535 subjects (i.e., classes) are partitioned into three subsets, half for training, 1/4 for ClassMap embedding (which is not used in our approach), 1/4 for test. The 535 OVA face classifiers are trained using SVMs with RBF kernels as in [6].

We want to mention that the nearest neighbor approaches [77, 84] that use similarity functions are also popular methods in practical face recognition systems. Nearest neighbor approaches are better choices than multi-class classifiers when few examples of a face ID are provided in the database. However, on this FRGC version 2 data set, sufficient training examples are provided for most of the face IDs. Thus, a nearest neighbor method will be slower due to a large number of database face images to compare with given an input. We therefore choose a OVA multi-class classification method as a baseline approach.

For comparison on the face data set, the most related works to speed up multi-class classification are DAGSVM [58] and ClassMap [6]. However, for DAGSVM the total number of binary classifiers is too large to train ($\frac{n(n-1)}{2}$ where $n$ is the number of classes). Thus, we compare following seven approaches, brute force where all 535 OVA face ID classifiers
are applied on an input face, filter-refine with ClassMap [6], filter-refine with $D_w$ which is the optimized weighted Hamming distance, filter-refine with $D_h$ which is the optimized unweighted Hamming distance, filter-refine with $D_c$ which uses outputs of all weak classifiers in the detector cascade, filter-refine with $D_a$, which is trained with all possible weak classifiers, and filter-refine with $D_r$, which is the Hamming distance with random partitions on 50 trials.

The brute force approach takes two steps, face detection followed by face ID recognition. The other approaches take three steps, face detect, face ID filter and face ID refine. In the refine step, only those OVA classifiers for the remaining face IDs from the filter step are applied.

A cascade face detector is trained with 2,500 face images randomly sampled from the training subset. We use the same set of Haar wavelet features as in [80]. The final cascade detector has nine stages and 449 weak classifiers in total. It achieves a detection accuracy of 96% at a false positive rate of $10^{-5}$ on the test set.

The training set for distance optimization comprises 20,000 triples. For all boosting based methods, the boosting processes stops when the reduction of training error in an iteration is less than the threshold $10^{-4}$. In training, 128 weak classifiers are selected for $D_w$, 135 for $D_h$ and 115 for $D_a$. For fair comparison, we use 150 random weak classifiers for $D_r$ in each trial. In the filter step, in which nearest neighbor retrieval is employed, each
Table 5.2: Comparison of filter step with different distance measures. The filter time is the average per test example.

<table>
<thead>
<tr>
<th>Distance measure</th>
<th>( D_w )</th>
<th>( D_h )</th>
<th>( D_c )</th>
<th>( D_a )</th>
<th>( D_r )</th>
</tr>
</thead>
<tbody>
<tr>
<td># weak classifiers</td>
<td>128</td>
<td>135</td>
<td>449</td>
<td>115</td>
<td>150</td>
</tr>
<tr>
<td>filter time (10^{-3} sec)</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>training time</td>
<td>25 s</td>
<td>30 s</td>
<td>N/A</td>
<td>8 h</td>
<td>N/A</td>
</tr>
</tbody>
</table>

test example is compared with all 18,409 training examples. In Table 5.2, the five different distance measures \( D_w, D_h, D_c, D_a \) and \( D_r \) are compared by number of weak classifiers used and the total time spent in the filter step for all 9,076 test examples. The brute force approach applies all 535 OVA classifiers on a test input. On average it takes 2.17 seconds to classify an input with 535 classifiers.

![Recognition and retrieval accuracy on the face data set.](image)

**Figure 5.19:** Recognition and retrieval accuracy on the face data set.

The graph in Figure 5.19 shows the final face recognition results obtained on the face data set. The curve for \( D_r \) is the average over 50 trials. At the cost of 50 OVA classifier evaluations per query, filter-and-refine using \( D_h, D_w \) and \( D_a \) achieves accuracies of 90.5%, 91.8% and 93.0% respectively. In contrast, at the cost of 178 OVA classifier evaluations per query, the ClassMap method achieves an accuracy of 91.6%. The brute
force approach that evaluates all OVA classifiers achieves an accuracy of 92.0%. In terms of speed, the methods $D_w$ and $D_a$ are 3.5 times faster than the ClassMap approach with better classification accuracies, and 10 times faster than the brute force approach.

The optimized Hamming distances $D_w$ and $D_h$ are consistently better than the Hamming distance based on random weak classifiers $D_r$. We also notice that about one third of the random weak classifiers only separate a very small portion of foreground examples from the rest, or do not partition the foreground class at all. This may partially explain why purely random partitions are not as good.

Interestingly, the proposed methods also achieve slightly better accuracies than the brute force approach. For instance, at the cost of 100 OVA classifier evaluations per query, filter-refine using $D_h$, $D_w$ and $D_a$ can achieve accuracies of 92.5%, 93.1% and 93.0% respectively. One plausible explanation is that a face misclassified via brute force can be avoided in our filter-and-refine steps if the OVA classifiers producing false alarms are not considered after the filter step. The same effect was also observed in [6].

Although $D_a$ achieves better accuracy, it does not reuse weak classifiers from the detector, and as noted in Chapter 4.2, training is very slow. Moreover, training $D_a$ is intractable when the potential weak classifiers are too many to enumerate, e.g., linear discriminants in a high dimensional Euclidean space as in the following experiments.

### 5.2.2 Hand Image Data Set

The second application is hand detection and hand shape estimation. We use the same hand image data set as in Sec. 5.1.2. The hand shape is parameterized by two angles: $\theta_1$ is the angle of the index finger with respect to the palm and $\theta_2$ is in-plane orientation. $\theta_1, \theta_2 \in [0^\circ, 90^\circ]$.

We adopted a two step process to recognize the hand shape. First, a boosted cascade is used to detect the hand. Then nearest neighbor retrieval with Euclidean distance in HOG feature space is used to recover two hand parameters. We use the same HOG features as in [92]. The detector is trained with linear discriminants as weak classifiers, as in [93].
The candidate weak linear discriminants are obtained on subsampled (30%) sets of HOG feature components at each iteration, via Fisher linear discriminant analysis [22].

We randomly partition the hand data (1605+925 examples) into training and test sets for 20 trials. In each trial we train a cascade detector and measure the performance of brute force nearest neighbor retrieval, filter-refine with $D_w$, $D_h$, $D_c$ and $D_r$. All approaches are compared by their average accuracy at different speedup factors. Note ClassMap [6] is not included in this experiment; ClassMap is intended for multi-class classification and inappropriate for parameter estimation.

In each trial, we train two distance measures $D_w$ and $D_h$, with 20,000 triples. Each training triple $(q_i, a_i, b_i)$ is constructed such that $b_i$ is farther away from $q_i$ than $a_i$ by Euclidean distance in $(\theta_1, \theta_2)$ space. There is one more constraint that the parameter $(\theta_1, \theta_2)$ of $a_i$ is within 10 degrees difference from $q$ in each dimension, since it is meaningless to maintain an order between $a_i$ and $b_i$ when they are both far from $q_i$. On average, 50 binary weak classifiers are selected for $D_w$ and 53 for $D_h$. For fair comparison, we randomly sample 50 linear boundaries for $D_r$ and 50 linear weak classifiers from the detector for $D_c$ in each trial.

Figure 5.20(a) and Figure 5.20(b) show the comparison of parameter estimation errors.
At a speedup factor of seven, the proposed approach using $D_h$ obtains mean absolute errors (MAE) $4.0^\circ$ and $3.4^\circ$ for $\theta_1$ and $\theta_2$, respectively. The brute force approach using nearest neighbor retrieval achieves MAEs $4.0^\circ$ and $3.2^\circ$ for $\theta_1$ and $\theta_2$, respectively. The MAEs of $D_w$ and $D_h$ are both below $5^\circ$ with speedup factors greater than eight for $\theta_1$ and $\theta_2$.

As we have mentioned in chapter 2, if the foreground within-class classification problem is continuous parameter estimation, a regression based method can be used. For the sake of comparison, we test support vector regression (SVR) [79] from SVMlight [33]. SVR exploits sparsity of the data, so it also has certain advantages in speed. The SVR models use the the same training and test sets as our method. The learning parameters (RBF kernel parameter $\gamma$, cost upper bound $C$) are both searched within the range $[10^{-3}, 100]$ via cross validation to find the best setting.

Table 5.3: Mean absolute error (MAE) in degrees, and average filter+refine time spent on each test example, on the hand data set. “RBF” is radial basis function and “Poly2” is polynomial kernel of degree 2. $D_w$ and $D_r$ report MAEs at a speedup factor of seven.

<table>
<thead>
<tr>
<th>Approach</th>
<th>MAE $\theta_1$ in degree</th>
<th>MAE $\theta_2$ in degree</th>
<th>Time ($10^{-4}$sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVR-RBF, $\gamma = 0.5$</td>
<td>4.4±0.13</td>
<td>3.4±0.09</td>
<td>35±1</td>
</tr>
<tr>
<td>SVR-Poly2</td>
<td>4.5±0.10</td>
<td>3.5±0.09</td>
<td>13±4</td>
</tr>
<tr>
<td>SVR-linear</td>
<td>7.1±0.15</td>
<td>5.0±0.12</td>
<td>4±1</td>
</tr>
<tr>
<td>Brute force NN</td>
<td>4.0±0.13</td>
<td>3.23±0.08</td>
<td>70±0.2</td>
</tr>
<tr>
<td>Filter-Refine $D_w$</td>
<td>4.0±0.11</td>
<td>3.4±0.11</td>
<td>10±1</td>
</tr>
<tr>
<td>Filter-Refine $D_h$</td>
<td>4.0±0.13</td>
<td>3.4±0.11</td>
<td>10±1</td>
</tr>
</tbody>
</table>

Table 5.3 summarizes the performance of all approaches. All approaches except SVR-linear achieve average estimation errors below 5 degrees, which is about the smallest difference of angle values that the human subjects who annotated the data set can tell with confidence. Although the measurement during annotation is pretty accurate (the fingers in the hand images are aligned with straight lines before calculating the angles), two hand examples would be regarded as “almost the same” by the human subjects, when the difference of finger angles between them is less than 5 degrees.

Compared with the lowest error achieved by SVR, the proposed filter-refine method $D_h$
reduces the error of $\theta_1$ by 0.4 and obtains the same error of $\theta_2$, while maintaining a speed only slightly slower than SVR with a linear model.

5.2.3 Vehicle Image Data Set

We also test our method on the multi-pose vehicle data set [38], as shown in Figure 5.7. As has been mentioned in Sec. 5.1.4, to obtain view angle estimation during tracking, we manually labelled view angles of 280 training examples at 5 degrees apart. In this experiment, we further annotated 192 training examples. Thus, in total we have 472 training examples annotated with view angles, out of all 1,297 vehicle images. We random partition all annotated vehicle images into a test set of 200 and a training set of 272 images in 10 trials. In each trial, a random sample of 700 images from the remaining 1,097 unlabelled vehicle images is added into the detector training set (but not for view angle estimation training).

We use the same HOG features as described in Sec. 5.1.3. The length of each feature vector is 2,025.

A cascade detector is trained in the same way as in the hand experiment in each trial, where linear discriminants are used as weak classifiers. On average the cascade detector has 480 weak classifiers in total.

To estimate the view angle of a detected vehicle, we use a simple nearest neighbor approach. The similarity measure is the dot product between HOG feature vectors of two
Figure 5.22: Example results of view angle estimation by HOG feature matching. Test inputs are in the top row, and corresponding nearest neighbors from training images are in the bottom row. The rightmost three pairs are incorrect matches.

Examples. Vehicle masks of training examples are used to zero out feature components outside hypothetical foreground regions. The dot product is normalized by the number of actual vector components that are inside the mask. With this similarity measure, the view angle of the nearest annotated training example is assigned to the test input as an angle estimate. Example matching results are shown in Figure. 5.22.

In our approach, we add a filter step to speed up the view angle estimation process by selecting candidate training examples before HOG feature matching. $D_w$ and $D_h$ are trained with 5,000 triples of annotated training examples. Each triple $(q, a, b)$ is constructed such that $a$ is closer to $q$ on the view angle axis than $b$, and $a$ is within 10 degrees from $q$. During boosting based optimization in 10 trials, on average $D_w$ added 44 weak classifiers and $D_h$ added 47 weak classifiers. For fair comparison, $D_r$ and $D_c$ uses 45 weak classifiers in each trial.

Because there exists strong confusion between frontal and rear views of vehicles, there is a spike around 180 degrees in the distribution of absolute errors, which dominates mean of the absolute errors (MAE). For better understanding of the errors, we measure the median of absolute errors (Median-AE) at different speedup factors in each trial. In Figure. 5.23, distance measures $D_w$, $D_h$, $D_c$ and $D_r$ are compared with brute force nearest neighbor approach on average Median-AE vs speedup factors. Note the results are averages over 10 trials. The brute force approach achieves an average Median-AE of 9.50 degrees. The proposed filter-refine approach using $D_w$ and $D_h$ achieve average Median-AEs of 11.5 and 11.0 respectively, at a speedup factor of 10. In contrast, the filter-refine approach with $D_r$
Figure 5.23: Comparison of different distance measures on car view angle estimation accuracy vs speedup factors.

which uses random partitions achieve an average Median-AE of 25.0, at a speedup factor of 5.

We also test the SVR methods on view angle estimation. Unlike the HOG feature matching approach, regression methods (e.g., SVR) require that all inputs have the same feature dimensions. There is no straightforward way to apply image masks with regression methods. Consequently, the features from background regions outside the image masks are also included during training, which becomes a major disadvantage for regression methods on this data set. In Table 5.4, we summarize the performance of SVR methods, in comparison with the proposed approaches. Filter-refine with $D_w$ and $D_h$ reduce Median-AE by half with a speedup factor of about nine over the SVR approaches.

5.2.4 Multi-pose Face Tracking with MultiPIE Data Set

We also tested the proposed speedup strategy in the face tracking experiment of chapter 5.1.5. To apply the speedup strategy, a Viola-Jones detector is trained with the same 4,160 training face examples used for multiplicative kernel training. There are also 1,900
Table 5.4: Median of absolute error (Median-AE) in degrees and the total filter+refine time spent on 200 test examples. $D_w$ and $D_r$ report Median-AEs at a speedup factor of ten.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Median-AE in degree</th>
<th>Time ($10^{-2}$ sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVR-Poly2</td>
<td>25.7±2.7</td>
<td>11.0±1.6</td>
</tr>
<tr>
<td>SVR-RBF $\gamma = 0.01$</td>
<td>24.2±1.8</td>
<td>10.3±0.7</td>
</tr>
<tr>
<td>Brute force NN</td>
<td>9.5±0.8</td>
<td>12±0.3</td>
</tr>
<tr>
<td>Filter-refine $D_w$</td>
<td>11.5±1.2</td>
<td>1.2±0.1</td>
</tr>
<tr>
<td>Filter-refine $D_h$</td>
<td>11.0±1.1</td>
<td>1.2±0.1</td>
</tr>
</tbody>
</table>

Figure 5·24: ROC curves of face detection on two test sequences with brute force (a family of detectors) and filter-refine speedup strategy. The red solid curves are for our approach with a family of 4,160 detectors. The green curves are results after a filter step which retrieves only 200 detectors out of all 4,160 detectors.

background training images for detector training. After training the cascade detector has 11 stages with a total of 1,066 weak classifiers.

A weighted Hamming distance $D_w$ with 200 bins is trained as described in Chapter 4. With this Hamming distance, the filter step retrieves top 200 training face examples that are most similar to the input. The detectors associated with these 200 training examples are applied on an input, instead of all 4,160 detectors.

The comparison between evaluating all face state hypotheses by a family of detectors and the filter-refine approach is shown in Figure. 5-17. The filter-refine approach has comparable accuracy with the brute force approach on both test sequences. Interestingly, in the
first face test sequence, the filter-refine approach even achieves a slightly better detection rate when the false positive rate is below 0.1 false positives per frame. However, the actual runtime of the filter-refine strategy is still very close to the brute force approach. The reason for this is that the brute force approach can be easily put into a matrix multiplication between a matrix of 4,160 linear classifiers with a matrix of feature vectors collected from sub-windows in an image. In our Matlab implementation, this matrix multiplication runs much faster than evaluating each feature vector with different linear classifiers each time, as in a filter-refine approach.

5.3 Summary of Experimental Results

For the purpose of object detection, the proposed detection approach with multiplicative kernels compares favorably against several existing approaches [86, 75, 91], in applications of hand shape detection, vehicle detection and multi-pose face detection. One of the most impressive result is obtained in vehicle detection on PETS 2001 sequence 5 data set. The proposed approach improves the detection accuracy by 20% at a fixed false positive rate of one per frame. The proposed approach also produces foreground state estimates that match well with groundtruth annotations in vehicle tracking and face tracking.

For efficient foreground within-class classification, the proposed filter-refine approach that reuses features from the detection stage outperforms evaluating all hypotheses and an existing efficient multi-class classification approach [6], on the FRGC V2 data set, a hand data set [91], and a vehicle data set [38]. The proposed approach also significantly reduces the number of detectors to be applied in multi-pose face tracking with a family of detectors. On the FRGC V2 data set, the optimized Hamming distance trained with all possible weak classifiers \(D_a\) achieves slightly higher accuracy than our approach. However, the training of \(D_a\) is far more expensive than our approach, and cannot be applied on the hand shape data set and the vehicle data set due to this computational complexity.

It is interesting to note that, on the FRGC V2 data set, the proposed filter-refine approach can achieve slightly higher accuracy than evaluating all face ID classifiers, with a
speedup factor of about five. A plausible explanation is that some of the face ID classifiers that produce false positives are filtered out in the filter step with our approach. A similar case is also observed in the experiment of the ClassMap approach [6].
Chapter 6

Discussion and Conclusions

In the final chapter of the thesis we summarize the main contributions and open issues in the work that we have described. We also point out potential directions for future work.

6.1 Main Contributions

This section provides a summary of contributions made in this thesis: A multiplicative kernel formulation that jointly solves detection and foreground state estimation problems, and a speedup strategy that reuses features from the detection stage to make foreground within-class classification more efficient.

6.1.1 A Family of Detectors

A key feature in our multiplicative kernel formulation is that the detectors can be associated with continuous state parameters or individual training examples. They are no longer just subclass detectors as in previous work [29, 39, 86, 66]. Thus, each detector can handle very specific foreground state variation. When they are employed at the detection stage, they can provide estimates of foreground state at a finer level than subclass detectors, or estimates of foreground segmentation masks if masks are provided with foreground training examples. The family of detectors can be also employed in a particle filtering frame work to evaluate foreground state hypotheses during object tracking.

Furthermore, the family of detectors are still sharing features during training, which makes the detectors robust with limited amount of training examples. Compared with existing feature sharing techniques [75, 91], the training with multiplicative kernels does not have the issue of combinatorial complexity to find best sharing among training examples.
or subclasses. The training process is just a variant of the standard SVM training process.

6.1.2 A Speedup Strategy that Reuses Features from Detection Stage

The message from the second part of the thesis is that feature evaluations in the initial detection stage can also be useful in the subsequent foreground within-class classification stage. With the proposed filter-refine approach, the recognition time can be reduced by a factor of an order of magnitude in our experiments on face recognition, hand shape recognition and vehicle view angle estimation.

Intuitively, the features or weak classifiers chosen for a detector may not be relevant to the foreground within-class classification problem. However, even with random weak classifiers, we can show that they approximate the nearest neighbor search with an Euclidean distance in the same way as LSH functions do. Furthermore, weak classifiers from the detectors can be selected towards improving state hypotheses retrieval accuracy. Eventually the optimized Hamming codes can be obtained for a fast filtering step using the Hamming distance.

What is also interesting is that the combination of the speedup strategy and a family of detectors makes the detection process input sensitive. The filter step after an initial detector removes implausible foreground state hypotheses of the input. Thus, only those most relevant detectors in the family of detectors are required to be evaluated. The empirical result on face detection and view angle estimation demonstrate the effectiveness of this combined process.

6.2 Limitations and Future Work

The applications that the proposed approaches have been tested on are in the computer vision area. Although the multiplicative kernel formulation may be employed in other areas, e.g., signal detection, we do not have any empirical result to verify it. Thus, we do not make any claim about the applicability of the proposed approaches to other areas. The scope of the proposed approaches in this thesis is limited to the computer vision area.
Naturally, a number of open issues remain to be addressed in the field of object detection and recognition. In this section we briefly discuss some issues that have not been addressed in this thesis and interesting directions for future work.

### 6.2.1 Batch Learning vs. Online Learning for a Family of Detectors

The main motivation of the multiplicative formulation is to improve the performance of detectors to detect a foreground object class that exhibits large within-class variations. When a large training set is provided to cover a large range of foreground state variations, a batch learning algorithm may not be easily applied due to its computational complexity. Intuitively, because different individual detectors are dealing with different foreground within-class variations, the family of detectors may be built incrementally. There exist online SVM learning algorithms like LASVM [12], which has been proposed to improve the speed of SVM learning processes. In our multiplicative kernel formulation, the relevance of a new training example to existing training examples can be measured by the within-class kernel $k_\theta$. The outputs from $k_\theta$ tell which existing training examples are more important to train a detector for this new example. Thus, it will be interesting future work to develop an online SVM learning algorithm that can make use of the within-class kernel outputs to make training even more efficient.

### 6.2.2 Extensions of the Multiplicative Kernel Formulation

Our multiplicative kernel model $C(x, \theta)$ is a general formulation to learn a family of classifiers controlled by a variable $\theta$. Potentially this formulation can be applied to other problems when a flexible set of functions is desirable. For example, it has been noticed in the PhD thesis of Athitsos [3] that the nearest neighbor retrieval problems share certain properties with classification problems. We may extend our model into $C(x, x', \theta)$, which is a $\theta$-sensitive similarity measure for $x$ and $x'$. If $\theta$ depends on $x$, it becomes a query-sensitive similarity measures to compare a query $x$ with data base objects $x'$.

Another potential extension of the multiplicative kernel formulation is to apply it to
regression problems. A family of regressors may be obtained with associated priors encoded in the variable $\theta$. This family of regressors can either output a distribution of estimates, or output a single estimate when the $\theta$ value can be determined at the test stage.

6.2.3 Effect of Object Scales

In our experiments, the detection window is of a fixed size. For objects of different sizes, they are normalized to the size of the detection window before the detectors are applied. Because the details of image features may be lost when an object is captured with a low resolution, the detection accuracy of small objects is usually lower than that of large objects. Our method does not explicitly handle the loss of image details. Thus, the detection accuracy on small objects depends on how much of the image details remain at lower resolutions.

6.2.4 Speed of Running a Family of Detectors

To speed up the detection process with a family of detectors, we added an initial detector to filter out trivial background patches, similar to the initial stages in a cascade detector [80]. To further speedup the detection process, a filter step that reuses features from the initial detector is employed to narrow down the scope of foreground state hypotheses. A filter-refine strategy to improve the classification process has also been used in ClassMap approach [6]. But in our approach the filter step is much faster because of reusing features.

It is possible to further speedup the filter step by hashing functions. It has been shown that a linear scan of all data base examples can be avoided by using hashing functions [67]. A similar strategy may be employed in the filter step to avoid linear scan to search for the nearest neighbors by a Hamming distance.

A hierarchical filtering process [74] may also be applicable to our filter-refine process. It has been shown as an effective way to handle tracking of articulated objects like human hands. It would be interesting to build a multi-stage coarse-to-fine hierarchy to speed up classification processes.
6.2.5 Image Masks in Training and Test

Object masks are used with our detectors to produce coarse segmentation masks of detected objects, *e.g.*, human hand. The use of image masks allow detection windows to have different aspect ratios or shapes for different test instances. A limitation of image masks is that they must be available in training data. For some object like articulated human body, image masks are difficult to obtain in training. Thus, an automatic approach to determine the foreground region inside a detection window will be of great interest for object detection and recognition tasks.

6.2.6 Filter with Hashing Functions

In the proposed filter-refine speedup strategy for foreground within-class classifications, an input is compared with all training examples using a Hamming distance. This exhaustive search can be a limitation in practice when a large number of data base objects must be compared with the input. One way to avoid an exhaustive search is to use LSH functions to approximate this Hamming distance. For this family of LSH functions, the number of bins in each hashing function and the total number of these hashing functions need to be decided. This can be an interesting optimization problem.
References


85


detection, segmentation and pose estimation. In Proceedings of IEEE Conference on

of histograms of oriented gradients. In Proceedings of IEEE Conference on
Computer Vision and Pattern Recognition (CVPR), 2006.

Proceedings of IEEE International Conference on Face and Gesture Recognition
(FGR), 2000.
Curriculum Vitae

Quan Yuan

Address:
27 St. Lukes RD, APT 9
Allston, MA 02134

Email: yq@cs.bu.edu
Phone: 617-763-6395
Web page: http://cs-people.bu.edu/yq

Research Interests
Computer Vision and Machine Learning

Education


September 2001 - July 2003 M.Eng. in Computer Science and Engineering, Computer Science and Engineering Department, Harbin Institute of Technology, P. R. China. Advisor: Prof. Wen Gao.

September 1997 - July 2001 B.S. in Computer Science and Engineering, Harbin Institute of Technology, P. R. China.

Publications

Journals


Conferences and Workshops


**Patents**


**Research Experience**

- Computer Science Department, Boston University, Boston, MA. 09/2003-06/2009. Research Assistant for Prof. Stan Sclaroff.

  - Automatic hand tracking in video sequences: Developed an hand detection method based on motion compensation in video sequences using C++; Implemented a dynamic programming based algorithm for temporal tracking of human hands.
– Object detection and recognition: Implemented the histogram of oriented gradient (HOG) features in C++ and Matlab; Developed a multiplicative kernel learning algorithm to detect object class of large within class variations, e.g., rotated faces, vehicles of different view angles.

– Speedup Strategies for Classification Processes: Designed a filter-refine strategy to speed up classification processes for face recognition, hand shape recognition and vehicle recognition; Implemented the algorithm to achieve speedup factor of about one order of magnitude in aforementioned applications.

• Siemens Medical Solutions, Malvern, PA. 06/2006-08/2006.
  Research Intern, supervised by Anna Jerebko.
  – Designed modules to calculate statistics of image features (textures and moments) for classification.

  Research Intern, supervised by Fatih Porikli.
  – Developed a tracking algorithm to track soft regions in ultrasound images.

Teaching Experience

• Graduate course in machine learning. Undergraduate course in linear algebra. Undergraduate course in algorithms. Introductory courses in C++ programming. Introductory course in Python programming. Introductory course in data structure.

Awards

• Annual CISE First Prize Award at the 2008 Boston University Science and Research Symposium.

Professional Activities

• Student Member of IEEE.

• Reviewer for the following conferences: ICCV’05-07, CVPR’05-09.

Qualifications

• Programming skills: C/C++, Matlab, Python and OpenGL.

• Operating systems: Windows, Linux.

• Language skills: Spoken and written English; spoken and written Chinese.

References

Available upon request.