Hierarchical SegmentBoost

A segment level classification approach to object class recognition

by

Jordan M. Reynolds

B.Eng. with Distinction., The University of Victoria, 2004

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

Dec, 2006

Abstract

In this work we address the problem of object recognition and localization within cluttered, natural scenes. Specifically we present a new approach to recognizing deformable objects that uses only local information. We suggest a new method of computing labels for arbitrary regions within an image using only local color and texture information. The results demonstrate our success in both identifying and localizing several classes of objects within cluttered scenes. We make two primary contributions to the field of deformable object recognition. First we present a new technique for labeling arbitrary regions within an image using texture and color features. Second we introduce a hierarchical approach to combining the classification results from segmentations of different granularity. Since the field of deformable object recognition is still in its infancy, we hope the work presented here will be used as a stepping stone to developing more complex, multi-object detection systems.

Contents

Abstract ii

Contents iii

List of Tables vi

List of Figures vii

Acknowledgements xii

1 Introduction 1

1.1 Motivation 1

1.2 Problem Definition 2

1.2.1 Presence vs. Localization 3

1.2.2 Instance vs. Class Recognition 4

1.2.3 Pixel Labeling vs. Bounding Box Annotation 4

1.3 Challenges 5

1.3.1 Environmental Changes 6

1.3.2 Intra-class Variations 6

1.4 Outline of Thesis 7

2 Related work 8

2.1 Single class object detection 8

2.2 Multi-class object detection 10

2.3 Non-rigid object detection 12

2.3.1 Generative models 12

2.3.2 Discriminative models 15

3 Theory 19

3.1 Boosting 19

3.1.1 AdaBoost 20

3.2 Belief Propagation 22

4 Segment Boost 24

4.1 System Overview 24

4.2 Pre-processing 25

4.3 Segmentation 26

4.3.1 Regular Griding 27

4.3.2 Irregular Region Segmentation 28

4.4 Feature Extraction 30

4.4.1 Color Histograms 30

4.4.2 Texture Histograms 32

4.5 Training and Classification 36

4.5.1 Combining Features 37

4.5.2 Training 37

4.5.3 Classification of new images 38

4.5.4 Logistic Fitting 39

5 Heirarchical Segment Boost 40

5.1 Motivation 40

5.2 Segmentation Tree 42

5.3 Belief Propagation on the Segmentation Tree 44

6 Results 47

6.1 Receiver Operating Characteristics 47

6.2 Dataset 49

6.3 Base classifier 50

6.4 Hierarchical model 51

6.5 Cow classifier results 55
6.6 Other class results 61
6.7 Run time 69

7 Conclusion and Future Work 70
7.1 Conclusion 70
7.2 Future work 71
7.2.1 Base classifier performance 71
7.2.2 Hierarchical model 71

Bibliography 73

A RGB to HSV conversion 77

B Dataset 78

List of Tables

6.1 Base classifier performance using only color, only texture, and combined color and texture features. 51
6.2 Base classifier performance using only color, only texture, and combined color and texture features. Cow BP shows the results after running belief propagation over the segment tree. 58
6.3 A comparison of our results with the PASCAL VOC challenge entries. Segment boost is our base classifier, while Hierarchical segment boost is after applying our hierarchical model. Please refer to [8] for the details of the various techniques. 61
6.4 Combined feature performance before and after applying the hierarchical model for the cat and car classes. 63
6.5 Comparison of our results to other entries in the 2006 PASCAL VOC challenge. Ranking is where our results would be ranked on similar data within the 20 entries. 63

List of Figures

1.1 Expected system behaviour 3
1.2 Instance vs. class level labeling 5
1.3 Bounding box vs. pixel level labeling 5

2.1 Example of the failure of rigid bound box labeling on rigid objects 10
2.2 Example of the failure of rigid bound box labeling on non-rigid objects 11

4.1 Overall system flow. 24
4.2 Effects of normalization on images of low contrast. 26
4.3 Results of applying regular grid and freeform segmentation techniques to an image. 27
4.4 Regular grid scale problem. 28
4.5 Example of color histogram computations. 32
4.6 Representation of a single scale, single orientation Gabor filter. . 34
4.7 Family of gabor filters, spanning both scale and orientation. . . . 35
4.8 Gabor filters applied to a sample image. 36

5.1 An illustration of the aperture problem. 41
5.2 Segmentation heirarchy. 42
5.3 Segmentation tree produced from a sample segmentation heirarchy. 43

6.1 Perfect and random guess ROC curves. 49
6.2 Silhouette annotations for a sample cow and cat image. 50
6.3 Cow boosted classifier details. 52

6.4 Example of a subset of segmentation trees, showing belifs before and after applying BP. 54
6.5 ROC curves produced by our cow classifier using color, texture, and combined feature sets. 57
6.6 ROC curve for color only features, showing all levels in the heirarchy seperately. 58
6.7 Sample cow detector outputs. 59
6.8 Sample failures produced by the cow classifier. 60
6.9 ROC curve for the cat object detector, using the color and texture features 64
6.10 Sample cat detector outputs. 65
6.11 Sample cat detector failures. 66
6.12 ROC curve for the car object detector 67
6.13 Sample car detector outputs. 68

B.1 A sample of images from the PASCAL cow dataset. 79
B.2 A sample of images from the PASCAL cat dataset. 80

Acknowledgements

I would have never achieved the completion of this work without the guidance and help from a large number of individuals. I would especially like to extend my thanks to my colleagues in the Lab for Computational Intelligence namely Scott Helmir, Kenji Okuma, Tristram Southey, and Mike Yurick. Most importantly, I'd like to extend my special thanks to my supervisors Dr. Kevin Murphy and Dr. Jim Little for keeping me on the right path to complete my research.

Jordan M. Reynolds

University of British Columbia

Dec 2006

Chapter 1

Introduction

1.1 Motivation

One of the most fundamental challenges in computer vision is the problem of object recognition. With a system capable of identifying objects within a scene we would be able to simplify many problems in both vision and robotics. Having a system able to identify and label images based on content would facilitate the development of a much more powerful image search engine, where, rather then searching based on the context in which an image appears, one could search based on the actual content of the image. Robotics could benefit as well, by integrating recognition systems into autonomous agents one could build much more powerful and useful robotic systems. Rather then building detailed, fragile static maps, the agent could localize and navigate by determining its location based on the objects it is able to identify within its surroundings. For example, if we are able to identify objects such as televisions and sofas within our current sourroundings, we can make a strong guess that we are in a living room, if we identify dishes, a stove, and a sink we are more likely to be in a kitchen. If a robot is able to identify a large number of objects within its environment, the task of localisation then becomes one of inferring location through context. The work of [20] shows the feasibility of this approach to navigation.

Much work has been done in the field of object detection, and strong results have been shown for the problem of identifying certain classes of objects, such as cars, that do not deform[1] [2, 20, 27, 31, 32]. Less work has been done on identi-

[1] While cars may rotate in the plain, relative positions between components do not change.

fying classes of completely deformable objects such as animals [1, 5, 16, 29]. We choose to focus our work on the problem of identifying these deformable objects.

The main difficulty in developing a detector that is capable of identifying the class of fully deformable objects stems from our inability to use positional information as a cue. When identifying rigid or semi rigid objects, we can use spatial cues as strong evidence for the class membership of any object. For example we would expect a computer screen to have four corners roughly aligned, and we would expect a car to have two wheels below the main body. We cannot rely on these cues for deformable objects, as the position of any semi rigid component of the object can change relative to any other component as the object deforms. Because of this, we need to develop new features that are both descriptive enough to allow classification with only local information, and general enough to allow for intra-class variations[2].

1.2 Problem Definition

The problem we are attempting to solve can be stated as :

> We wish to construct a classifier that, given a novel image[3] is able to label all pixels belonging to a particular object class. At the same time, we need to also be able to recognize those pixels that do not belong to the object class, and instead belong to the generic class "background".

The problem becomes one of choosing a method of describing pixels or regions within the image, and constructing a classifier, based on these descriptors is able to produce a binary label for the class membership of the pixels or region. Figure

For example we would always expect to see the tires lower then then body.

[2]Intra-class variations stem from visual differences between objects of the same class, for example no two cows will look visually identical.

[3]That is, an image we have never seen before

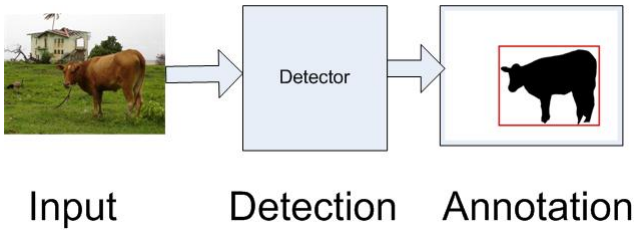


Figure 1.1: Given an input image, we wish to design a classifier that can produce a correct object level annotation as output.

1.1 shows how we expect the system to behave. The resulting system should produce pixel level, localized class labels for object we have trained it to detect. We will now more carefully define what we mean by pixel level annotation, and class labels.

1.2.1 Presence vs. Localization

We can consider two similar problems in the field of object identification. The first is the problem of presence detection, the second localization. To solve the presence problem, one need only annotate each image as to containing the object of interest or not. For example, if a presence based system were to be asked to annotate the input image of Figure 1.1, simply indicating that there is a cow present in the image would be sufficient to be considered a correct detection. For problems like image searching and indexing, these simple annotations are sufficient to complete the task of searching for all images that contain a cow. The more interesting problem is that of localisation. Rather then simply indicating whether an object is present or absent in an image, one might be interested in locating the object. For many tasks, it is imperative that we know where things are within our environment rather then just what exists within our view. In this thesis we present a solution to the problem of localization of objects within an image.

1.2.2 Instance vs. Class Recognition

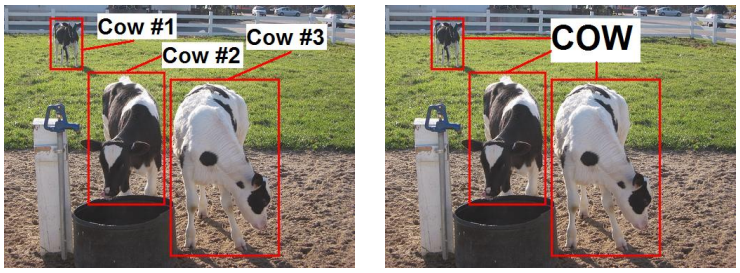
We can also sub-divide object recognition problem into the problems of class and instance level recognition. The problem of instance level recognition is the problem of identifying and applying a unique label, in a repeatable way to individual members of a class. Figure 1.2(a) illustrates this, each cow is given a unique label "cow #1", "cow #2" etc. The repeatability requirement means, if we see the same object in a different image, possibly from a different viewpoint, we need to apply the same label. Cow #1 will always be labeled as such, and no other instance of "cow" will be given the same label. Simpler techniques may work well to solve these instance level problems. As we can consider each instance a unique class, we expect, barring environmental changes, very little intra-class variation. The problem is still not an easy one, but we need not consider the effects of intra-class variations.

Class level recognition deals with the problem of applying a higher level label to a collection of objects. Rather then applying a unique label to each instance of "cow", we wish to apply the label "cow" to all instances as illustrated in Figure 1.2(b). We could attempt to learn a family of individual instance level classifiers, then apply a semantic hierarchy to the results[4]. However we are much more interested in constructing a classifier that is able to naturally generalize over a class of objects, and is able to handle a large amount of intra-class variation. In our research we tackle the problem of class level recognition.

1.2.3 Pixel Labeling vs. Bounding Box Annotation

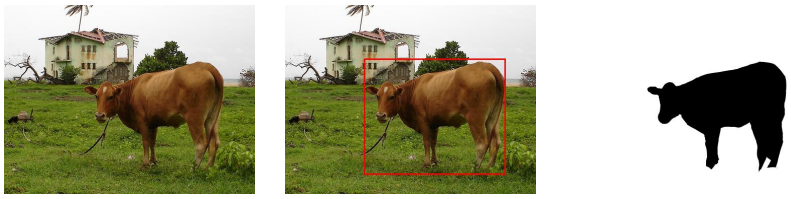
There are several possible methods for annotating the final results. Figure 1.3 shows the primary two. In bounding box annotation, we simply wish to draw a rectangular boundary around the region we believe the object exists. This method of annotation works quite well for objects that do not deform greatly,

[4]In this case, an is-a relationship, "Cow #1" is-a "Cow".



(a). (b).

Figure 1.2: (a). Instance level labeling for 3 distinct cows, note the number just represents a unique label applied to that cow, in the way humans apply names. (b). Class level labeling, all "cow" objects are assigned the same label.



(a). (b). (c).

Figure 1.3: (a). Input image, (b). Bounding box annotation, (c). Pixel level labeling, here the dark regions are pixels that have been labeled as "cow".

and objects we wish to detect from a single viewpoint[5]. Pixel labeling however attempts to apply a label to each pixel in the image, producing a much tighter annotation. The remainder of our work deals only with pixel level labeling.

1.3 Challenges

There are several challenges we need to overcome in developing a successful object recognition system. Most of these stem from the deformable nature of objects, intra-class variations, and changes in environmental conditions such as

[5]Single viewpoint here refers to objects we do not expect to see any rotation within the object class, "Sides of Cars" would be an example

lighting. We will now examine these challenges in more detail.

1.3.1 Environmental Changes

Images can come from a number of places, digital cameras, scans of photographs, cellular telephones. With these variability in sources also comes variability in the quality of the image. While some digital cameras have high quality optical elements and sensors, cell phones tend to have lower quality elements. One of the largest problems caused by low quality sensors is a lack of contrast under low lighting conditions. As the lighting conditions change, the color, and amount of detail we can see within the image also changes. In Section 4.2 we discuss some pre-processing steps that will help reduce the impact caused by these changing lighting conditions.

1.3.2 Intra-class Variations

The largest problem we need to address in building a class level recognition system is that of intra-class variation. We can break these variations down into three primary types.

- Normal visual differences that stem from individualization. No two cows look exactly alike, so we need to account for this by building some generalization into our system.
- Object deformation, as a non-rigid object deforms it changes in visual appearance. Not only do parts of the object that may have been visible before become occluded and new ones become visible, but the relative position between parts of the object changes as well.
- Viewpoint changes, even a rigid object looks different depending on our viewpoint. A car looks visually different from the front then it does from the side.

We must carefully design our detector to handle these intra-class variations. While some are easier to design around, such as training multiple detectors based

on viewpoint, the problems caused by object deformation are much harder. Our system attempts to handle the variations caused by deformation by only examining local information, and ignoring spactial constraints. Because of this, we are able to construct a classifier that is independent of deformations as well as viewpoint changes[6]

1.4 Outline of Thesis

Developing a class level object recognition system for deformable objects is a challenging task. In the following chapters we outline one approach to solving this problem. Our system is comprised of two primary components. The first is a patch level classifier that is able to classify each patch in a segmented image as either belonging to the class of interest, or being background. The second component takes the output of several classifiers, all run at different segmentation granularities, and applies belief propagation to combine the results into a single, more accurate classification.

In the next chapter we outline previous work in the field of both rigid and deformable object recognition. Chapter 3 presents some background material to the reader who may not be familiar with the field of image processing or machine learning. In Chapter 4 we present our approach to solving the problem of object recognition and localization by applying hierarchical boosted classifiers. Chapter 6 shows how our system performs on a set of real world data. The final chapter summarizes our work, and presents some possible future extensions that may further improve our results.

[6] Viewpoint changes can be considered large scale deformations.

Chapter 2

Related work

2.1 Single class object detection

First we focus on the problem of class level recognition of rigid objects, that is objects for which we can draw a fixed aspect ratio bounding box. A fixed aspect ratio bounding box in this context is one that we are allowed to change the scale of, but not the relative x and y dimensions.

Viola and Jones [32] developed a system for class level recognition of faces in natural images. They approached the problem by computing a pyramid of images across multiple scales. Features were then computed by correlating all of the images within the pyramid with a collection of hand chosen patches. The integral of the pixels that fell within a fixed dimension region (or window) were then computed at fixed spacing within the image, essentially producing an integral response to each filter at each scale within the pyramid. The single valued integral responses for each region were then concatenated to produce a feature vector for each of the windows at each scale. A discriminative classifier, namely boosted decision stumps, was then trained using a collection of hand labeled data. The resulting detector was then used to label each window at each scale as to being in the object class or not. To reduce computational complexity, the detector was run at each scale in turn, starting with the largest. If a region at the largest scale was classified as not containing the object, windows falling within the same region in smaller scales were not examined, and were summarily considered to not contain the object of interest.

Schneiderman [27] extends the idea of the cascade by not only reducing computational complexity across scales, but across features within a scale. Early stages in the feature cascade use simple to compute features, while later stages use more complex models. By using simple detectors early in the cascade, it is possible to eliminate regions within the image that we can be sure do not contain the object of interest, while only computing the more expensive features in areas which we have some belief the object may exist.

Murphy, Torralba, and Freeman [20] further extend the ideas of [32] by including the concept of spacial masks. Features in the Viola detector were computed as the sum of filter responses over whole windows. Spacial masks not only consider the filter responses, but the location within the window the response occurs. Essentially each element in the feature vector consists of the sum of value of the filter responses within a sub-region inside the window. This allows for better object localization within the image due to the fact that the feature vector will change even when the same object is found, in whole or part within the window. Consider a square box, smaller then the window. Without the use of spactial masks the feature vector would remain the same as the window moved around, provided the box remained fully contained within the window, as we are computing the sum of filter responses over the window as a whole. By applying spacial masks, the feature vector will change as the window moves around, as edges and corners will move between different mask regions. This leads to high detector responses in a fewer number of windows, and better overall object localization.

For object classes that do not deform, or in situations we are not expecting viewpoint changes, these techniques work quite well. They begin to fail however when object undergo out of plane rotations, or deformations. Even fully rigid object, such as cars, have a visually different appearance when viewed from the front or side. Additionally, the aspect ratio of the object begins to change

(a). (b).

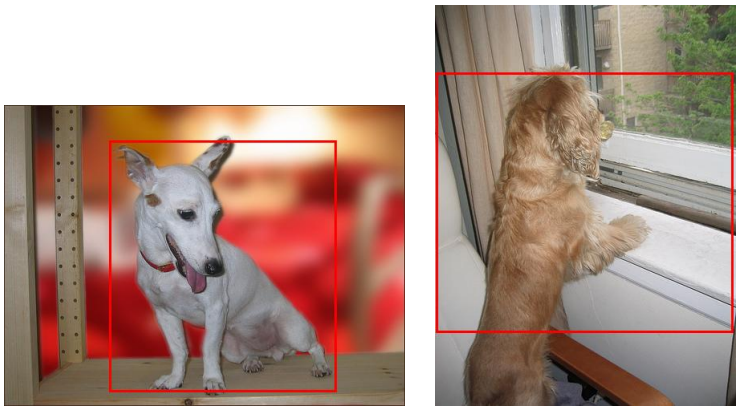
Figure 2.1: Two images containing the object "car". The box is of equal aspect ratio in both images. In (a), the box tightly constrains the object of interest. Due to the affine viewpoint change, (b) contains as much background as it does positive class.

as it undergoes these transformations. Figure 2.1 illustrates the problems that occur when a rigid object undergoes out of plane rotation, a bounding box that correctly annotates the object class "car" in (a), does not correctly annotate the same object class in (b). We will discus a technique to overcome this problem in section 2.2. Figure 2.2 illustrates the problems caused by deformable classes of objects. In both images the object we are interested in is "dog", due to the deformable nature of animals, it would be impossible to define a fixed dimension bounding box that would correctly annotate all instances.

2.2 Multi-class object detection

One possible solution for the out of plane rotation problems discussed in section 2.1 is to train multiple detectors, one for each expected view[1]. In its simplest form, we would train N independent detectors, one per expected viewpoint, and simply run each detector over a query image in turn. We could then take

[1]Though there are an infinite number of possible view points, each detector has some invariance to rotational changes, and we need only train a few detectors who's invariance windows overlap to cover all possible viewpoints.



(a). (b).

Figure 2.2: Two images containing the object "dog". The box is of equal aspect ratio in both images. Due to the deformable nature of the object class, the bounding box that correctly annotates the object in (a), fails to do so in (b).

the most probable detection as our final result. This approach suffers from computational complexity issues, as the time required to run the process grows with the number of expected viewpoints N. An additional problem we could encounter with such a naive system is that it would require a vast amount of annotated data, as each classifier would require its own unique set of labeled data.

Torralba, Murphy, and Freeman [30] have developed a system that jointly trains the detectors for each viewpoint, making use of common features when possible. The joint boosting approach presented in [30], attempts to learn single decision boundaries at each step that septate a single class from the rest, thus casting the multi-class detection problem into a collection of one vs. rest single class problems. By applying Joint boosting, they are able to reduce the number of features required for N classifications problems from linear in N to roughly logarithmic. As Torralba suggests, we can use this joint approach to train our family of multi-view detectors simultaneously, and reduce both the number of

features we need to learn, as well as the application time.

Using joint boosting it would be possible to train a detector capable of identifying an object that we would expect to go though a countable number of viewpoint changes. That is, we know a priori what rotational changes we expect the object to undergo. In the case of cars for example, we can reasonably expect it to rotate within the plane, and would normally not account for inversions and such. For freely rotating objects, while it is certainly possible to collect and label data for each expected view, it quickly becomes impractical to do so. Such a system is also incapable of dealing with completely deformable objects for the same reason. We cannot expect to collect and uniquely label images containing all possible positions and views for "dog", or similar objects.

2.3 Non-rigid object detection

As we have discussed in the previous section, several techniques exist to solve the problem of viewpoint changes for rigid objects. We will now examine some existing solutions for the identification and localization of deformable objects. These techniques can be broadly divided into two categories. First, generative models attempt to learn a statistical model for a class. We can then pose the question what is the probability that a new object was generated using our model? Discriminative models on the other hand, try to find decision boundaries within the data (be it features or raw pixel values) that separate the class of interest from the distractor or background data. The method discussed in Sections 2.2 and 2.1 are examples of discriminative classifiers.

2.3.1 Generative models

The one advantage to generative models is that they do not require any negative class data to train. The problem of choosing a fully representative set of

non-class exemplars is very hard, and would require large amounts of data[2]. Generative models however, only look at statistical relationships within purely positive class data, and therefore do not require the notion of negative class exemplars.

Rikert, Jones, and Viola [24] developed a statistical clustering generative model for object detection. In their model they compute texture based features using steerable pyramids to provide an affine invariant feature set. The steerable pyramid can be considered a local texture response calculated across a family of directional filters. These directional filters are applied to an image pyramid[3]. A feature vector, or "parent vector" is then computed by integrating the filter responses within a regular grid over both scale and directions. These parent vectors are then clustered using a greedy algorithm in a bottom up fashion. Each cluster represents a collection of repeatedly occurring features within the class of interest. The resulting clusters are then used to generate a Gaussian mixture model. An additional discriminative step was added by computing the the same descriptors and applying clustering to a set of negative examples, to remove elements from the mixture model that occurred in both positive and negative class data[4]. The final classification of a novel test image is achieved by computing the probability as to the image having been generated from the positive class model, as opposed to the negative class model.

This system was only designed and tested on pre-extracted patches, classifying the whole patch as to being either positive or negative class. Due to the computational cost of querying the class of a patch, applying this technique in a sliding window framework would be very costly, and therefore was unable to

[2]If for example we wish to train a "car" classifier, to fully train the discriminative model we would need to provide the training algorithm with at least one example of every instance of "not-car". In practice we make due with a subsample of negative class data.

[3]An image pyramid is a collection of blurred and re-sampled images, effectively creating a hierarchy of scales.

[4]Thus making this approach more of a hybrid generative/discriminative classifier

be used to both classify and localize an object within an image.

In [16] Leibe, Leonardis, and Schiele learn a generative model that is able to overcome the inability of the method of [24] to localize the object of interest within a cluttered scene. They proceed by running an interest point detector, such as the Difference of Gaussian operator [18] to find salient points within the image. A patchlet is then extracted from around each of these interest points, these patchelets are of a size depending on the scale they were detected at. The patchlets are then resized to be of a fixed dimension (in the case of their experiments 25×25 pixels). A clustering algorithm is then run on these patches to to group visually similar patches. Each cluster can be considered as a codebook entry, with the whole codebook representing the collection of possible patches we may see being generated in an image by the object of interest. In the next step, each of these codebook entries are compared against the images used to generate them, and their position relative to the annotation center is recorded with the entry. Consider a car tire as a codebook entry, this patch may occur in a visually similar way at the front and back of a car. We would annotate the codebook entry with the information that the center of the object could exist above and forward of the patch (in the case of the rear tire) or above and behind (in the case of the front tire). To detect the object of interest in a novel image, the interest point detector is once again run, and patches extracted. These patches are then compared against the existing codebook entries, and either assigned as a match to an existing entry, or discarded if they are not visually similar to any entry. In the next step, each of the patches that match a codebook entry use their positional information to vote in Hough space as to where it believes the center of the object is. This additional voting step allows for the localization of the object within a cluttered scene. A final back projection step is then applied, where codebook entries that did not appear in the image, but are supported by the hough estimate of the center are invoked to segment the object from the background.

While this technique produces very good results for deformable objects, it suffers from a few drawbacks. The first is the problem of choosing a fully representative codebook. While most objects have several similar features, large intra class variations can only be modeled by insuring that images representing the entire range of dis-similarities is available at training time. In addition if we wish to train a multi view detector in this way, we need a fully representative set of views. The amount of data to achieve this for an object class with high intra-class visual variation quickly becomes large. Additionally such an approach is unable to reject false positives with elements similar to the class of interest. That is, if two object classes share visual similarities, it is hard for such a model to distinguish between them.

Fritz, Leibe, Caputo and Schiele [10] extend their work in [16] by introducing an additional discriminative step to reduce the impact of the problems previously mentioned. In this approach they compute their hypotheses in the same way as they did in [16], where it is possible for each matching codebook entry to have come from a different training example. In this approach, the initial implicit shape model is used to produce a set of hypotheses. A SVM based discriminative model is then used to verify the correctness of each of these hypotheses. The resulting system, due to the use of negative class data is able to eliminate a large number of false positive matches. However the problem of choosing a fully representative set of data still remains.

2.3.2 Discriminative models

Our work, along with a few new techniques take a purely discriminative approach to the problem of deformable object recognition.

In [29] Shotton, Winn, Rother and Criminisi develop a technique for solving the deformable object recognition problem using a mix of conditional random fields, and boosted discriminative classifiers. Using a collection of hand anno-

tated images they proceed to learn a conditional random field model of class labellings within the training set. This model attempts to learn contextual relationships between objects and their co-locations within an image. The local potentials are produced by the output of joint boosting [30], and provided a likelihood that each pixel is a member of a given object class. The joint boosting detector is trained using features generated by a learned texton model [17]. Textons here are used in a similar way to the hand chosen features of [20] and as they are learned from data, provide a much stronger feature set. These filter responses are combined with spatial masks for better localization. In addition a color model is learned as a component of the local potential, as color can provide a strong cue to class membership. The resulting system is able to both segment the image into regions, and provide object labels for each region. Training is done using a small set of hand labeled data, with the result being both a segmentation of the image and a per-segment label

A lot of the power of Shotton's approach comes from learning the joint appearance models, that is having high confidence in one region label, and a strong correlation between the label for that region and another region, serves to boost the confidence in the label of the second region. Some of this power however can be lost when objects appear in scenes out of context from the norm, a car appearing inside perhaps at a car show, or a person swimming for example. Additionally there is the requirement to learn a detector for all region labels that occur within the training set, that is, if our training set has a large number of cluttered images, with high per-image object count, the number of classifiers that need to be learned quickly becomes large. In their paper, Shotton states that it takes up-wards of 42 hours of computational time to learn a 21 object detector.

The work of [4] is similar to our own. In this work they present a method for combining bottom up segmentations using global image features, and the result of a top down classifier. The top down classifier in the case is a parts based model, while the bottom up classifier is a region growth algorithm similar to the method of [9]. In their approach they proceed to learn a series of represen-

tative fragments for each class of interest. These fragments are very similar to the patchlets of [16]. They then proceed to build a hierarchy of segmentations, and compute a saliency measure for each node in the hierarchy. This saliency measure is a rank ordering of the distinctiveness of the each region. They then proceed to minimize the global energy of the system using a two part cost function. We need to note here that the structure of the tree depends only on the segmentation information, and not on the classifier output. The first part of the cost function penalises leaf nodes for which the top down labeling differ from the bottom up labels (this penalty can only be applied at the leaf nodes as they only have classification information at one scale). The second part of the cost function penalises low saliency regions whose labels differ from their parents, while not penalizing high saliency segments. This allows segments with distinct appearance to have labels dis-similar to their parents.

In [3] Borenstein extends this work by building a full Bayesian model. First a prior over all possible foreground background segmentations is computed by constructing a soft segmentation hierarchy similar to that used in [4]. Next this prior is used by a top down segmenter, based on shape templates to construct a global approximation of the segmentation. Finally inference is used to combine the top down and bottom up segmentations. The shape templates used by the top down segmenter do not look at any underlying image features, but rely purely on the boundaries produced by the segmentation. These templates consist of a simple binary mask of fixed size, that label all pixes as being foreground or background and do not inherently contain any information about the underlying image appearance (for example, they do not consider internal color or texture information). Because of this, these shape templates have a natural invariance to intra-class appearance variations, but it is still required to learn a fully representative set of templates for all possible object deformations and rotations.

These work differs from our own in several fundamental ways. First they

makes no attempt to overcome the problems caused by learning a representative set of fragments for a deformable object class. The underlying patch based model still needs exemplars from all possible object deformations to provide strong base classifications. Secondly the nodes within the hierarchy only contain information as to their global saliency, and not to their direct belief in being either positive or negative class. Because of this it is unable to correct errors in classification due to too little local information. While their approach attempts to solve for a single best global labeling, we are integrating the belief at each level of the hierarchy to either strengthen or weaken our belief in the labels at the leaf nodes.

Our approach attempts to solve several of these problems. By pre-segmenting the image, we are able to produce results similar to [29] and [4] without the need to learn classifiers for all objects that may occur in the scene. In addition, objects appearing out of context pose little problem to our system, as we have no notion of co-occurrence. We combine elements of the sliding window approach, by computing fixed length descriptors over regions, with the power of segmentation, to allow the windows to change as the object deforms. By introducing a hierarchical segmentation, we can help ensure correct segmentation at one or more scales, without the need to explicitly search across scales as is required in sliding window approaches [32]. While the work of [4] uses a similar hierarchical segmentation it does not attempt to classify these segments, and rather opts for a single global classification. Because of this we are able to correct errors in the classification at lower scales by introducing the classification results from higher up in the segmentation tree. Our features are similar to those used in [29], but we choose Gabor filters rather then requiring the need to learn a representative family of Textons. We are able to learn a strong classifier without the need for large amounts of data as may be need in the multi view approaches, and can do so computationally faster then most Generative models are able to. Our resulting system is able to produce results similar to those of [29], with considerably less training time.

Chapter 3

Theory

3.1 Boosting

Given a set of features that describe some underlying data, and a set of labels for these features, we wish to learn a classifier. This predictor should, given a new set of features, be able to produce the correct label. The task of constructing this predictor is one of the fundamental problems in machine learning.

Rather then attempt to construct a single, monolithic predictor that is able to separate the label classes, we choose to approach the problem slightly differently. If we can build a family of simple predictors, who are able to separate the classes correctly better then chance[1] and have a mechanism to combine these predictions, we could produce a predictor with an error rate much better then any of the inidividual predictors. Adaboost attempts to construct such a predictor from a family of weak predictors, or weak learners. It does so by choosing a new predictor at each round of an iterative process, that minimizes the classification error, then re-weighting the training data and repeating. Effectively we are boosting the performance of many weak classifiers by allowing them all to vote on their belief of the final label.

[1] That is, they are able to predict the correct label for the data greater then 50% of the time.

3.1.1 AdaBoost

One of the most popular boosting algorithms is AdaBoost [25] [26]. Consider the following problem ; given a collection of features F, and an associated set of labels $Y \in \{0, 1\}$ we wish to construct a predictor $G(F)$ that, given a set of input vectors F' returns a set of labels $Y' \in \{0, 1\}$. The error rate of this predictor is given by [12]:

$$\epsilon = \frac{1}{N} \sum_{i=1}^{N} I(y'_i \neq G(f'_i)) \tag{3.1}$$

Where $I(y'_i \neq G(f'_i))$ is the indicator function, which takes the value one when $y'_i \neq G(f'_i)$ which is true when the predictor returns the wrong label, and zero otherwise.

As was briefly mentioned in the previous section, boosting works by applying a sequence of weak predictors and taking as the label, the weighted majority vote :

$$G(F) = \sum_{i=1}^{I} \alpha_i G_i(F) \tag{3.2}$$

Here $G_i(F)$ is the i'th weak predictor and α_i is its associated weight. The problem of training an AdaBoost classifier then becomes one of choosing the family of weak predictors, and their associated weights.

Algorithm 1 outlines the AdaBoost procedure. At each iteration, we choose the current best weak classifier[2], compute the classification error, and re-weight the data. It is the re-weighting step that gives AdaBoost most of its power. Intuitively we would like to guide the AdaBoost algorithm to focus more on choosing weak predictors that can separate previously mis-classified examples, and reduce its focus on example that have already been correctly classified in a previous step. We achieve this by weighting each of the training examples, at

[2]It is possible to choose the same classifier in multiple iterations, though each will have a different weight assigned

each iteration t using the following re-weighting scheme as introduced in [26]:

$$w_i^t = w_i^{t-1} \cdot \exp\left[log\left[\frac{1-\epsilon^{t-1}}{\epsilon^{t-1}}\right] \cdot I(y_j \neq G^{t-1}(x_i))\right], i = 1, ..., N \tag{3.3}$$

Algorithm 1 AdaBoost

1: Set all the initial weights equal $w_j = \frac{1}{N}, j = 1, ..., N$
2: **for** $i = 1$ *to* I **do**
3: Fit a weak learner $G_i(x)$ to the currently weighted data.
4: Compute the error $\epsilon_i = \frac{\sum_{j=1}^{N} w_j \cdot I(y_j \neq G_i(x_j))}{\sum_{j=1}^{N} w_j}$
5: Compute the classifier weight $\alpha_i = \log \frac{(1-\epsilon_i)}{\epsilon_i}$
6: Re-compute the data weights $w_j \leftarrow w_j \cdot e^{\alpha_j \cdot I(y_j \neq G_j(x_j))}, j = 1, ..., N$
7: **end for**
8: Return the strong classifier $G(x) = \sum_{i=1}^{I} \alpha_i \cdot G_i(x)$

We assign the weight w_i^t to the same value it took in the previous iteration if the weak classifier G^{t-1} we chose previously correctly classifies the example. Otherwise we increase the weight by the factor $\frac{1-\epsilon^{t-1}}{\epsilon^{t-1}}$. Because of this, the predictor we fit in iteration t will lend more weight to examples that were misclassified in the previous iteration. We continue to iterate until we have selected the desired number of weak learners. There does not, at this time seem to be a good theoretical stopping condition for this algorithm. Continuing to iterate after the error rate has stabilised can, in some cases continue to increase test set performance of the final strong predictor.

We have not addressed the problem of choosing a weak predictor here, as any predictor able to classify the data with greater then 50% success will work in the boosting framework. Several such weak predictors include decision stumps, and trees [7].

3.2 Belief Propagation

We are interested in examining the inference problem in graphical models. Our goal is to compute a marginal probability $P(x_i)$ for each node x_i in a graph G. This belief should take into account not only its local evidence, but information from its surroundings, in this case from its neighboring nodes. Computing of the full marginal for each node would require marginalizing out all other variables in the graph :

$$P(x_i) = \sum_{x_j, \forall j \in (G - x_i)} P(x_1, x_2, ..., x_N) \tag{3.4}$$

For large complicated graph structures, the time to compute the marginal for each node grows with the graph size. We desire instead to find a way to re-use information, as summation is a commutable operator, rather then re-computing the sum of Equation (3.4) for each node x_i we can re-use summations that do not include the current query node x_i. This simple operation gives rise to the belief propagating algorithm [13]. We will only consider the case where the graph G is singly connected, and loop free. We can decompose any such graph into a factor graph [34] containing local evidence $\phi(x_i)$, and clique potentials $\psi(x_i, x_j)$. Our marginal probability at node x_i can be expressed in terms of its local evidence, and clique potentials as :

$$P(x_i) = \frac{1}{Z} \sum_{x_j, \forall j \in (G - x_i)} \prod_{i,j} \psi(x_i, x_j) \prod_i \phi(x_i) \tag{3.5}$$

We can now define the concept of a message $m_{i \to j}$ as the re-usable computation stored at each node. If we define the message as :

$$m_{i \to j}(x_j) \leftarrow \sum b(x_i) \cdot \psi(x_i, x_j) \tag{3.6}$$

Which is the product of the local belief and the clique potential. The local belief can be defined as :

$$b(x_i) = \frac{1}{Z} \phi_i(x_i) \prod_{j \in N(i)} m_{j \to i}(x_i) \tag{3.7}$$

Where Z is a normalization constant, to ensure our local belief is a strict probability. We can see that, by expanding Equation (3.5), we have the marginalisation over a product of all local evidences and clique potentials. Iteratively Expanding Equation (3.7), will yield the same result for any given loop-free graph. The advantage we gain in using Equation (3.7) is we can compute the message $m_{i \rightarrow j}(x_j)$ once for each pair of nodes, store the result, and re-compute our local belief $b(x_i)$ for each node in the graph with no need for further messages to be passed.

Algorithm 2 lays out the basic framework for performing message passing belief propagation on a loop free graph. We start with an arbitrary leaf node, and begin passing messages to all of its immediate neighbors. We then begin working inward toward the root, passing messages forward through the graph. When we reach the root node of the graph, we begin working backward, passing messages from the root node back toward the leafs. Once we again reach the leaf, every node in the graph has received messages from its immediate neighbors, and is ready to compute its local belief.

Algorithm 2 Belief Propagation

1: let U be the set of all nodes that have yet received messages from its neighbors $N(x_i)$.
2: **while** U is not empty **do**
3: Choose a node x_j from U that has not sent messages to its neighbors $N(x_j)$.
4: Compute a message $m_{j->n}$, $\forall n \in N(x_j)$ using Equation (3.6)
5: **end while**
6: Compute the local belief $b(x_i)$ for all x_i in G using Equation (3.7)

Chapter 4

Segment Boost

4.1 System Overview

Figure 4.1 provides a visual outline of how our system operates. An input image undergoes a pre-processing step to lessen the impact of environmental lighting conditions as well as poor contrast caused by low grade optical elements in the capture device. Next the image is segmented using a fast over-segmentation method. A fixed length descriptor is then computed for each region which is then labeled using a learned boosted classifier. The process is then repeated at several granularities of segmentation. The results are then combined using a hierarchical belief propagation technique. The final result is a label for each segment in the image, as well as an associated confidence.

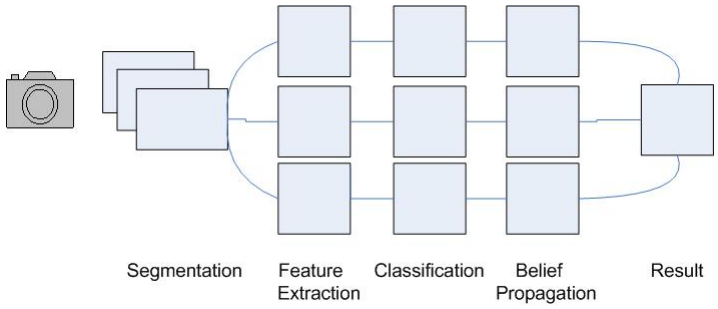


Figure 4.1: Overall system flow.

We will now examine each component of the system outlined in Figure 4.1 in detail.

4.2 Pre-processing

Due to poor environmental and lighting conditions or poor quality equipment, we often find images that are of very low contrast. It is difficult to extract feature information from such images, as much of the data is compressed into a small brightness range. We choose histogram equalisation as our method for dealing with such problems. Histogram equalisation attempts to increase overall image contrast by stretching the intensity values within an image across the entire available dynamic range. The desired result is a monotonically increasing cumulative intensity histogram.

Let $p(i)$ be the probability of observing a pixel with intensity value i in an image $\mathfrak{I}$, we can write this probability as :

$$p(i) = \frac{\sum_{j=1}^{N} \delta(v_j - i)}{N}, i \in 0, .., M \tag{4.1}$$

Where N is the total number of pixels in the image, v_j is the intensity value of pixel j, and M is the maximal intensity value. We can define the cumulative density at i as :

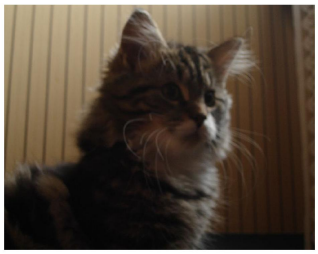
$$c(i) = \sum_{j=1}^{i} p(j) \tag{4.2}$$

Our desire is for $c(i)$ to be linearly increasing. To achieve this, we fit a transfer function $v_i' = \tau(v_i)$ to map current intensity values v_i into new values v_i' in our normalized image, where $c(i)'$. The function is fit by minimizing the residual between the cumulative density of the desired histogram $c(j)$[1], and the histogram produced by the transfer function $c(\tau(j))$ for all intensities M:

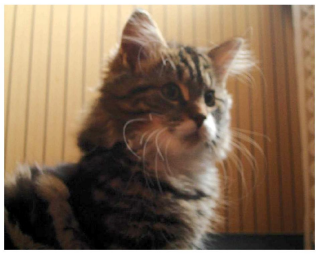
$$Minimize \ \|c(\tau(j)) - c(j)\|_{\forall j \in M} \tag{4.3}$$

Figure 4.2 shows the result of applying histogram equalisation to a low contrast image. One can see much more detail in the body of the cat in the normalized image.

[1] In this case the desired histogram is flat.



(a)



(b)

Figure 4.2: (a). Un-normalized image, (b). Image normalized using histogram equalization. Note the extra detail visible on the body

4.3 Segmentation

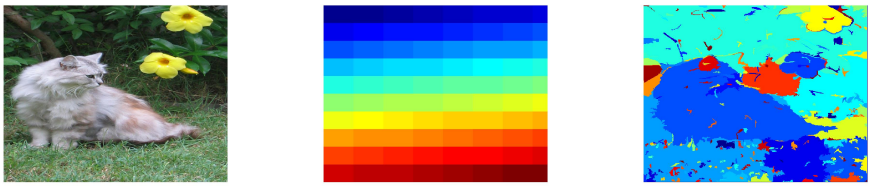
We wish to decompose the image into a number of discrete regions. Ideally the boundary of these regions would exist along the boundaries of objects within the image. We will consider a good segmentation one that exhibits the following properties :

1. Does not merge dis-similar regions.
2. Does not heavily over-segment the image[2].

Rule (1) asks that the segmenter does not return a segment that contains both a large portion of the object of interest and background. Rule (2) ensures that the regions are large enough that we can compute meaningful descriptors.

We will consider two primary segmentation techniques. The first, regular griding, does not look at the underlying image data, rather it chooses to subdivide the image into fixed size regions. This method satisfies rule (2), but will fail to satisfy rule (1) in many cases. The second method we consider is the graph based image segmentation technique of [9]. This technique is not

[2]Some over segmentation is acceptable, but segmentation into single pixel regions is not.



(a). (b). (c).

Figure 4.3: (a). Input image, (b). Regular grid segmentation of the cat image, (c). output produced by applying the segmentation technique of [9]. In both (b), and (c) areas of similar color represent regions contained within the same segment.

guaranteed to satisfy either rule (1) nor rule (2), but does so in most practical cases.

4.3.1 Regular Griding

We first consider the naive technique of regular grid segmentation. This technique sub divides the image into a number of $N \times M$ rectangular regions. Figure 4.3(b) shows the result of this type of segmentation. As we do not examine any of the underlying image structure when performing the segmentation, we need to ensure that we choose region sizes that do not combine too much positive and negative class in the same segment. We also have to consider the problem of small positive class objects in the image. These may be smaller then a single rectangular region, and possibly only contribute a small fraction to each of several segments. Figure 4.4 represents this problem, the positive class circle, while the same size as the rectangular grid, spans 4 segments rather then being centered in a single one. This serves to pollute the information we can extract from each of these regions regarding the positive class with a great deal of negative class information.

Early experimentation showed that this technique was sufficient for posi-

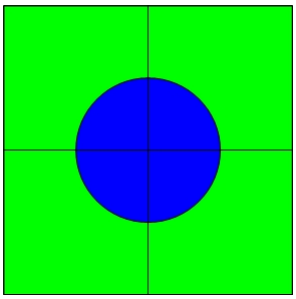


Figure 4.4: Regular grid segmentation. Here the circle is the positive class object.

tive class objects that are large relative to the image size, but fail due to the aforementioned problem for smaller positive class objects.

4.3.2 Irregular Region Segmentation

Rather than use fixed grid segmentation that does not exploit any underlying image structure, we now examine a technique that is able to dynamically choose segment sizes and shape based on underlying image information. One such approach is that proposed in [9] which views the image as a graph, and attempts to build segments from the bottom up.

We will consider a graph G representation of the image, where each pixel is represented by a vertex $v_i \in V$, and the connection between neighboring pixels as edges $e_{i,j} \in E$. We will also define a weight $w(e)$, as a measure of the visual difference between two vertexes[3]. For simplicity we can use the absolute intensity difference as a measure of $w(e) =\mid v_i - v_j \mid$. We can now compute a metric measuring the strength of the internal connections within a component (or segment) C:

$$INT(C) = \max_{e \in MST(C,E)} w(e) \tag{4.4}$$

[3]This can be viewed as the weight of the edge in the graph

We are computing here the cost of the most expensive edge in the minimum spanning tree ($MST(C, E)$) in component C.

Algorithm 3 lays out the basic procedure for segmenting the image graph. At each iteration we consider the pairwise weight between two components, and the internal weight within the components. If by merging the independent components C_i and C_j we do not increase the measure $INT(C_i \cup C_j)$ then we merge, otherwise we leave them as independent components.

Algorithm 3 Efficient tree based segmentation.

1: Sort E into $\pi = (o_1, ...o_M)$ by non-decreasing edge weight.
2: Start with segmentation S^0 where each vertex v_i is its own segment
3: **for** $q = 1$ *to* M **do**
4: let v_i, v_j be the vertexes connected by e_q, the q-th edge in the ordering π
5: **if** v_i, v_j are dis-joint components of S^{q-1} **then**
6: **if** w_q is small compared to $INT(v_i)$ *and* $INT(v_j)$ **then**
7: Merge v_i and v_j in S^q
8: **else**
9: $S^q = S^{q-1}$
10: **end if**
11: **end if**
12: **end for**
13: Assign each pixel to a segment in S to $R(x)$, the collection of regions.

In practice we first smooth the image before applying the procedure outlined in Algorithm 3 to reduce large changes in $w(e)$ due to image noise. The amount of smoothing applied in the pre-processing step is controlled by a parameter σ_{seg}. We also define a runtime parameter that controls our measure of "small" used in step 6 of the segmentation algorithm. This parameter k ultimately controls the level of over-segmentation produced by the algorithm. The result is a segmentation similar to that shown in Figure 4.3 (c).

4.4 Feature Extraction

Now that we have divided the image up into a number of unique segments, we need to compute a descriptor over each region. This summary over regions should uniquely describe the information within the patch, in a way that is independent of patch size and shape. Without this requirement, it would be impossible to compare the descriptor from two un-equal sized regions.
We choose to describe the image information in a region using two sets of features, color and texture histograms.

4.4.1 Color Histograms

Color can provide a very strong separator between objects of a particular class, and background. Rarely do we see blue cows, or red dogs. Because of this we choose to include color information, in the form of Hue, Saturation and Intensity histograms in our feature set. We choose to use histograms as a representation, as it allows us to construct a vector of fixed size from a region containing an arbitrary number of pixels.

We desire to construct a color feature vector that is as robust to intensity changes as possible, to achieve this we construct the histogram over the HSV space image rather then the RGB space image. The HSV space naturally decouples the intensity information from the inherent color[4], and therefore provides the robustness we seek.

Histogram Binning

To maintain this HS and V separation we will construct two separate histograms, the first being of size $N_h \cdot N_s$ will contain the HS channel information, while the second of size N_v will be constructed over the Value space. The resulting

[4]The color information is contained in the Hue and saturation components, while the intensity is represented in the Value component. Refer to Appendix A for details on conversion from RGB space to HSV.

histogram will therefore be of size $N = N_h \cdot N_s + N_v$. For our experiments we have set $N_h = H_s = N_v = 10$. We chose 10 bin histograms, as this seems to be a standard value within the literature [**?**]

For each pixel in the region of interest $R(x)$ we can compute a color vector $y_{hs}(i) = [H_i, S_i]$ consisting of the HS channel information, and a second vector $y_v(i) = [V_i]$ containing the V channel information. We can then denote $b_h(i) \in (1, ..., N_h), b_s(i) \in (1, ..., N_s)$ as the bin indexes for the color vector $y_{hs}(i)$ and $b_v(i) \in (1, ..., N_v)$ as the bin index for the intensity vector $y_v(i)$. We can now construct a kernel density estimate over the region, $q_{hs}(x) = \{q(m, n; x)\}_{m=1...N_h, n=1...N_s}$ is given by [22]:

$$q_{hs}(m, n; x) = Z_{hs} \sum_{i \in R(x)} \delta[(b_h(i) - m)(b_s(i) - n)] \tag{4.5}$$

And a second density estimator $q_v(x) = \{q(l; x)\}_{l=1...N_v}$ as :

$$q_v(l; x) = Z_v \sum_{i \in R(x)} \delta[(b_v - l)] \tag{4.6}$$

where δ is the Kronecker delta function, and Z_{hs}, Z_v are normalizing constants to ensure the total bin weights sum to 1. It should be noted that the HS and V space histograms are normalized separately, to ensure that we do not over count. That is, every pixel is represented once in the HS space histogram, and again in the V space. As the total number of bins in the V space histogram is smaller then the HS space, normalizing the histograms after combining will reduce the power in the HS space, and lend more weight to the V space.

Figure 4.5 shows a sample color histogram computed across the given image, note that the HS and V histograms are shown separately. In our implementation we compute the HS histogram as a two dimensional grid of bins which we then convert to a single linear index vector, the V histogram is then concatenated to achieve our final color feature vector.

We choose to compute this color histogram over the un-normalized image, as the normalization process tends to introduce some hue changes into the image.

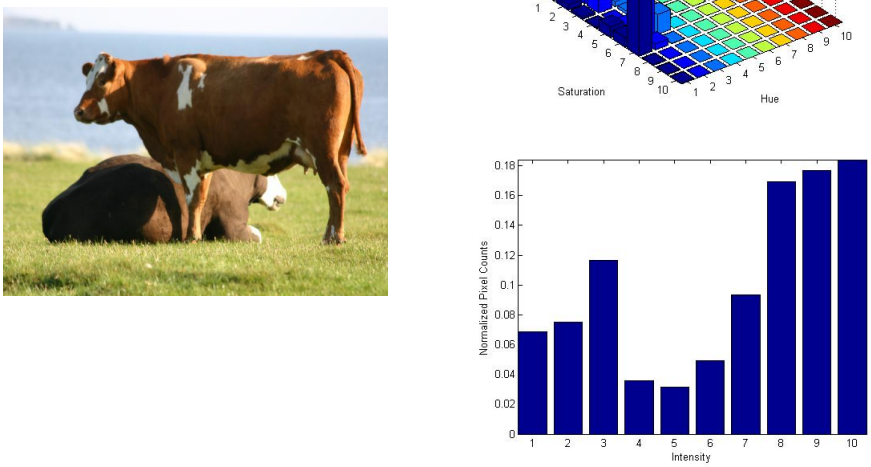


Figure 4.5: Color Histograms : (left). Input image, (top right). Two dimensional hue and saturation histogram computed over the whole image, (bottom right). Intensity histogram computed over the same region

4.4.2 Texture Histograms

Rather then simply look at the color in each region, it makes sense to also examine the texture information. This will help us distinguish a brown horse from the trunk of a tree, or the side of a barn. Many techniques exist to compute the textural information within an image including Fourier Transform [14], Textons [17], and Gabor filters [33]. Textons require some notion of examplars, either learned or hand chosen, we wish to avoid the problem associated with choosing a fully representative set of basis patches and opt instead to compute frequency

domain information over regions. Both the Fourier Transform, and Gabor filters provide tools to extract frequency domain information, the Fourier transform however computes the frequency domain content as a function of the image as a whole, thus discarding any locational information. The Gabor filters however allow us to retain locational information, allowing us to isolate texture information produced by a smaller region in the image rather then the image as a whole.

Gabor Filter

The use of Gabor filters in texture classification is quite widespread [15, 33]. Gabor filters partially owe their popularity to their similarity to processes that occur in the primary visual cortex [19]. As well these filters have been shown to posses optimal localization properties in both spatial and frequency domain [33].

A single Gabor filter $F(x, y)$ can be viewed as the product of a complex sinusoidal $s(x, y)$of particular angle and frequency and a Gaussian $\mathfrak{G}(x, y)$ as Equation (4.7) shows. Figure 4.6 shows a single scale and orientation Gabor filter.

This complex filter is then convolved with a query signal $\mathfrak{I}(x, y)$ to produce a complex response. The real component of this response represents the amount of power $\mathfrak{I}(x, y)$ has in the band represented by the filter, at location (x,y). While the angular component represents the phase of the response.

$$F(x, y) = s(x, y) \cdot \mathfrak{G}(x, y) \tag{4.7}$$

Where:

$$\begin{aligned} s(x, y) &= e^{-j2\pi(u_0 x + v_0 y)} \\ \mathfrak{G}(x, y) &= e^{\frac{-(x^2+y^2)}{2\sigma^2}} \end{aligned} \tag{4.8}$$

We can view this complex filter as a symmetric filter (Equation (4.9)) and

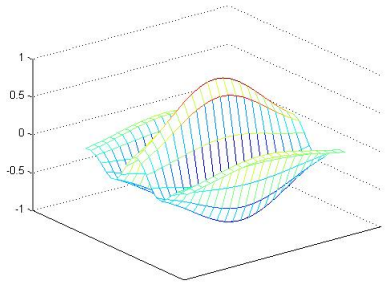


Figure 4.6: A two dimensional, single orientation and frequency Gabor filter. The Gaussian envelope can be clearly seen in the exponential decay of the sinusoidal function.

an asymmetric filter (Equation (4.10)) using the Euler identity[5]. This decomposition is done for reasons of simplicity in implementation.

$$F_{sym}(x,y) = \cos\left(u_0 x + v_0 y\right) \cdot e^{-\left[\frac{x^2+y^2}{2\sigma^2}\right]} \tag{4.9}$$

$$F_{asym}(x,y) = \sin\left(u_0 x + v_0 y\right) \cdot e^{-\left[\frac{x^2+y^2}{2\sigma^2}\right]} \tag{4.10}$$

Rather then performing a single complex convolution between the filter and target signal, we can now perform two simpler real convolutions:

$$\begin{aligned} R_{sym}(x,y) &= \mathfrak{I}(x,y) * F_{sym}(x,y) \\ R_{asym}(x,y) &= \mathfrak{I}(x,y) * F_{asym}(x,y) \end{aligned} \tag{4.11}$$

The resulting magnitude can be calculated as Equation (4.12). This magnitude is referred to as the Gabor energy [28]. We can also calculate the phase response as in Equation (4.13).

$$R_{mag}(x,y) = \cdot\sqrt{R_{sym}(x,y)^2 + R_{asym}(x,y)^2} \tag{4.12}$$

[5]Euler's identity can be expressed as : $e^{j\pi\theta} = \cos\theta + j \cdot \sin\theta$

$$R_{phase}(x, y) = \arctan\left[\frac{R_{asym}}{R_{sym}}\right] \tag{4.13}$$

Filter Banks

A single Gabor filter will most likely not provide enough information to perform any sort of texture classification, as it is only capable of analysing the texture at a single scale and orientation. To overcome this we build a family of Gabor filters, of varying orientation and scales.

Figure 4.7 shows such a family of filters. In our system, we choose to apply Gabor filters of 4 scales and 6 orientations. This is primarily motivated by wishing to capture as much textural information as possible, while avoiding examining responses from large scale image features, such as edges and object boundaries. Figure 4.8 shows the Gabor energies for a collection of Gabor filters of a single orientation, but varying frequencies.

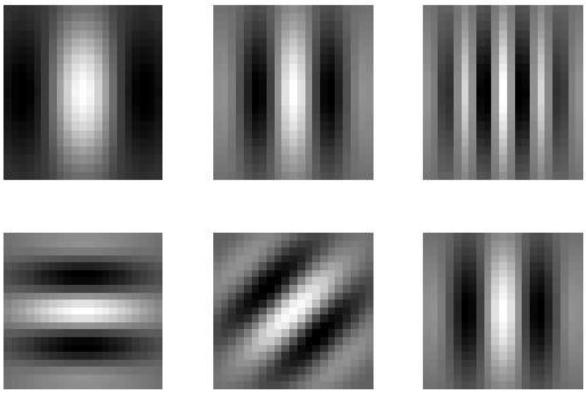


Figure 4.7: Gabor filters showing (top) a progression across scales, and (bottom) progression across orientations.

Given a family of Gabor filters, $F_i(x, y)$, we compute the response of the image to each filter in turn, $R_i(x, y) = F_i(x, y) * \mathfrak{I}(x, y)$.

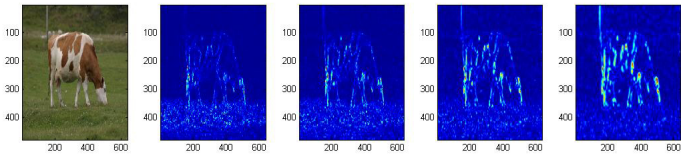


Figure 4.8: A progression of Gabor filters of increasing frequency applied to an image. The left most response shows the highest frequency band (smallest texture), the right most image shows the lower frequency (largest texture).

Histogram Binning

As we discussed in Section 4.5, we desire to construct a vector of fixed length from regions of arbitrary size. Once again we choose to use histograms as our mechanism. We compute these histograms over the Gabor energy responses, one per scale and orientation.

In our experiments we choose a collection of Gabor filters spanning 6 orientations, and 4 scales, resulting in $6 \cdot 4 = 24$ Gabor energy fields. We chose to quantize the responses into 10 discrete bins. As in the previous section, we will denote $b_{(s,\theta)}(i) \in (1, ..., N)$ as the bin index for pixel i, in scale s and orientation θ. We again construct a density estimator $r_{(s,\theta)}(x) = \{q(m; x)\}_{m=1...N}$ over their region $R_{(s,\theta)}(x)$.

$$q(m; x) = Z \sum_{i \in R_{(s,\theta)}(x)} \delta[(b_{(s,\theta)}(i) - m)] \tag{4.14}$$

The factor Z is a normalizing constant to insure that the histogram bins sum to one.

4.5 Training and Classification

In the previous section we discussed how to compute a set of fixed length descriptor for regions of variable size within the image. We will now discuss how

these descriptor can be used to classify each region as either coming from the object of interest or not.

4.5.1 Combining Features

We have computed a family of histograms for each region, one in HS space $q_{hs}(x)$, on in V $q_v(x)$ space, as well as one per scale and orientation in Gabor energy space $r_{(s,\theta)}(x)$. We compute our region descriptor by concatenating these locally normalized histograms into a single vector. This process also includes flattening the two dimensional $q_{hs}(x)$ histogram into a single dimensional vector.

$$\mathfrak{D}(x) = [q_{hs}(x), q_v(x), r_{(s,\theta)}(x)]_{s=1\ldots S, \theta \in \Theta} \tag{4.15}$$

Where S is the total number of scales for which we have computed Gabor energies, and Θ is the family of orientations. We do not further normalize the resulting histogram, as this would tend to cause numerical underflow, and information could potentially be lost. The length of the resulting region descriptor $\mathfrak{D}(x)$ is independent of the region size.

4.5.2 Training

Given a set of labeled training images, we wish to learn a general classification rule. We choose to use boosting as discussed in Section 3.1. For each image in the training set, we first perform the segmentation, then compute a a descriptor $\mathfrak{D}(x)$ for each region. We additionally compute a label $y(x)$ where :

$$y(x) = \begin{cases} 1 & \text{if } \mu(x) \geq \mathrm{T} \\ 0 & \text{otherwise} \end{cases}$$

As our ground truth labels are in-dependant of the segmentation, it is possible for a segment to span both positive and negative class regions. To overcome this, we calculate an overlap factor $\mu(x)$ as the fraction of pixels within a region

labeled as positive class:

$$\mu(x) = \frac{\sum_{i \in R(x)} I(p_i = 1)}{\sum_{i \in R(x)} 1} \tag{4.16}$$

For our experiments, we choose $T = .75$, that is for a region $R(x)$ to be labeled as positive class, at least 75% of its pixels must be labeled as positive class.

Boosted Stumps

As we discussed in Section 3.1, the boosting framework can use any weak learner to achieve a high classification rate. In our system we choose to use decision stumps, as they are computationally simple, and very powerful in practice.

Equation (4.17) gives the basic form of the decision stump. Here F_k is the k'th element of the feature vector F, it should be noted that the decision stump only considers a single element of the feature vector rather then the feature vector as a whole. Effectively we are choosing the linear decision boundary that best separates the underlying data in a single dimension.

$$\begin{aligned} g(F, \rho) &= sign[w_1 \cdot F_k - w_0]_x \\ where\ \rho &= \{k, w_0, w_1\} \end{aligned} \tag{4.17}$$

To apply the classifier to novel inputs, we need only store a collection of ρ parameters, one per boosting round.

We now use the boosting procedure outlined in 3.1 to fit a family of weak learners $g_i(F, \rho_i), i = 1..N$ to the training data that can be used to classify new feature vectors.

4.5.3 Classification of new images

Given a query image $\mathfrak{I}'$, we can compute a fixed length descriptor $\mathfrak{D}'(x)$ using the process outlined in section 4.4 for each region $R(x)$. We now wish to apply

a class label $L(x)$ and a score $S(x)$ to each region using the classifier we learned in the previous section.

Equation (4.18) outlines the process of applying a score and label to the boosted output. Effectively the score is computed as the weighted sum of votes from the individual weak learners.

$$\begin{aligned} S(x) &= \sum_{i=1}^{I} \alpha_i g_i(\mathfrak{D}'(x), \rho_i) \\ L(x) &= sign(S(x)) \end{aligned} \tag{4.18}$$

The final label is taken as the sign of the resulting score. It should be noted that the boosting classifier does not use all of the feature elements of $\mathfrak{D}'$, it is indeed possible for the boosting classifier to only look at a small number of the elements of $\mathfrak{D}'$.

4.5.4 Logistic Fitting

While the labels produced by boosting are somewhat useful for classification, we are much more interested in the scores. These can be viewed as a confidence in our prediction. As the score is simply a total of the class label votes, we need to apply some extra machinery to convert it to a usable confidence value. To do this we reserve a portion of our test data, and fit a logistic function to the resulting scores, using the ground truth labels as calculated in section 4.5.2.

We fit the regression parameters $\theta = [\alpha, \beta]$ to a small subset of the data using a standard logistic regression package. We can then predict the confidence that a new segment label is in the positive class $P(L(x) = 1|\mathfrak{D}'(x), \theta)$ using Equation 4.19.

$$\begin{aligned} P(L(x) = 1|\mathfrak{D}'(x), \theta) &= \frac{e^{\phi}}{1 - e^{\phi}} \\ where\ \phi &= \alpha + \beta \cdot S(x) \end{aligned} \tag{4.19}$$

Chapter 5

Heirarchical Segment Boost

5.1 Motivation

We noticed in early experiments that combining or splitting segments could change the probability that the segment would receive a particular label. We can explain this change by reconsidering our feature set. Because our features are histograms, they combine the information from an entire region, effectively averaging over whole region. The segmenter may choose segmentation boundaries that span areas of subtle texture and color changes, or regions where these changes are smooth. The histograms therefore average over these changes in color and texture, effectively diluting the information contained in the descriptor. While the classifier is still able to apply the correct label to the segment, it does so with a lesser confidence.

To overcome the problem of choosing a single best segmentation level we build a hierarchy of segmentations, and combine the results from each layer in the segmentation tree using belief propagation. Figure 5.1 provides us with an example that motivates our choice of this hierarchical model. In Figure 5.1(a). we see one possible segment within an image, it would be hard as a human to apply the label "cow" to this segment as there simply is not enough information to do so. Our detector has the same problem, while it may label the region as cow, it will do so at a much lower certainly then it may if it were given more information. In Figure 5.1(b). we see a second possible segmentation, it would not be surprising that a human would apply the label "cow" given the

additional information, as well we would expect our detector to be much more certain that this is a cow. Figure 5.1(c). shows the other extreme, with the segment comprising the whole image. In this case there is too much information to apply a consistent label. We would expect a human to maybe apply the label "cow in field" or "field" rather then labeling it "cow" and ignoring the background information. We would expect our feature vector computed over this under segmentation to be polluted with both positive and negative class data, once again leading to a high level of uncertainty produced by our detector.

(a). (b). (c).

Figure 5.1: An illustration of the aperture problem. (a) Is a small segment of a cow, it would be hard for a human to apply a label given only that much information. (b) the right scale segmentation, it would be easy for a human to apply the label "cow". (c) indiscriminating, again it would be hard to apply the unique label "cow" while ignoring the background.

As we wish to build a classifier that is able to identify objects at any scale within an image, it would be impossible to tweak a segmenter that would produce a best segmentation for all objects we expect to see within the data set. So we are motivated to run our system with different segmentation parameters, in the hopes that we will at some scale, get close to the best possible segmentation for all objects. We then face the problem of combining the resulting information. Simply taking the highest certainty detection for each pixel would possibly lead to inclusion of mis-classifications at the smallest scale that have no support at higher scales. Belief propagation gives us a much cleaner mechanism to combine

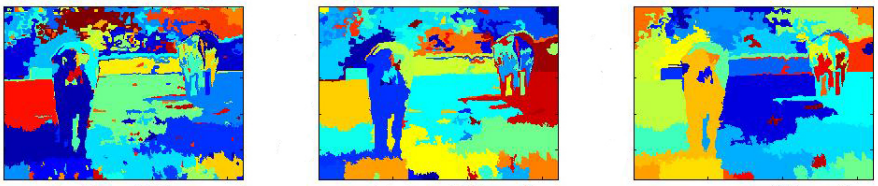


Figure 5.2: Hierarchy of segmentations produced by varying the parameters $\varphi = \{0.5, 500, 50\}, \{0.75, 500, 200\}, \{0.75, 500, 500\}$.

the information across scales. In the next section we present our approach to applying belief propagation to our problem.

5.2 Segmentation Tree

As we discussed in Section 4.3, the segmenter has three degrees of freedom $\varphi = \{\sigma, k, min\}$. By controlling these parameters we can produce segmentations at finer or coarser levels of granularity. Figure 5.2 shows such a hierarchy, the strong object boundaries remain very similar, only the internal segmentation changes over scales.

It is intuitive to view these segmentation hierarchies in tree form, where the root node represents a segment in the highest level of pyramid. And children, segments that occupy the same space in the lower levels of the pyramid. Figure 5.3 shows a simplified tree constructed from the given segmentation. In practice the lower level segments may overlap one or more segments in the level above. We choose to assign the child node to the parent it most overlaps by computing the pixel-wise overlap between the parent $R(p)$ and the child $R(c)$:

$$O(p, c) = \frac{\sum R(p) \cap R(c)}{\sum R(c)} \tag{5.1}$$

Let $\mathbb{P}$ be the set of all parent candidates, that is, all the segments that $R(c)$

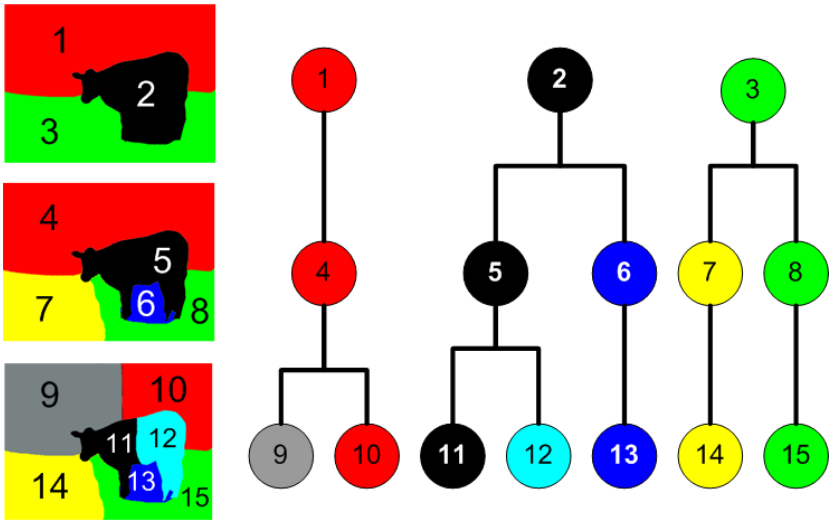


Figure 5.3: The tree (right) built from the corresponding hierarchy of segmentations (left). Here we present a pesudo segmentation.

overlap by at least one pixel. We assign the parent of $R(c)$ to be :

$$Parent(R(c)) = R(p), \ where \ p = \arg\max_{q \in \mathbb{P}} \ O(q, c) \tag{5.2}$$

The resulting structure is guaranteed to be a n-ary tree, as each child is assigned only a single parent, and each parent is allowed to have multiple children. The reason for choosing this structure will become apparent in the next section. Rather then building a single monolithic tree we build a family of trees, each rooted at a segment in the highest level of the segmentation hierarchy.

We now apply the methods we have previously discussed to compute a probability $P(L(x) = 1|\mathfrak{D}'(x))$ for each region $R(x)$ in each level of the segmentation hierarchy independently. The result is a unique probability that each segment takes on the positive class label, for each segment in the hierarchy. It should be noted that, by decomposing the larger segments, we can drastically change our belief in the label for the same underlying pixels at different levels in the

hierarchy. This is due to the fact that the features produced by a small segment lower in the hierarchy may be drowned out when it is merged with a larger segment higher up in the tree.

5.3 Belief Propagation on the Segmentation Tree

We now wish to merge the results from each level in the segmentation hierarchy. Simply averaging beliefs for each pixel would not lead to a consistent label for every element in a segment. Rather we choose to use belief propagation to allow the nodes in the segment tree to share their belief as to the label that each should take.

The local belief $\phi(x)$ represents our belief that node x takes the positive class label. We compute the local belief as shown in Equation (5.3), where we simply take the logistic thresholded value from the detector.

$$\phi(x) = P(L(x) = 1|\mathfrak{D}(x)) \tag{5.3}$$

We also store the full patch descriptor $\mathfrak{D}(x)$ at each node, we require the vector to facilitate comparing two nodes in the tree. We can now compute an inter-node comparability measure $\lambda_{i,j}$ for all connected pairs of nodes i, j in the tree. As $\mathfrak{D}(x)$ is a histogram, we choose to compute the χ^2 distance (see Equation 5.4 as a measure of similarity. We additionally exponentiate the χ^2 measure to both constrain the result in the range $(0, 1)$, and invert the result. This inversion is required as the χ^2 measure is small for two similar $\mathfrak{D}(x)$ and large for two dis-similar $\mathfrak{D}(x)$, we desire our compatibility score to be large for similar descriptors, and small otherwise.

$$\begin{aligned} \chi^2_{i,j} &= \sum_{m=1}^{k} \frac{(\mathfrak{D}_m(i) - \mathfrak{D}_m(j))^2}{(\mathfrak{D}_m(i) + \mathfrak{D}_m(j))} \\ \lambda_{i,j} &= e^{-\chi^2_{i,j}} \end{aligned} \tag{5.4}$$

Our local potential function $\psi_{i,j}$ should reflect our intuition as to how the probabilities should flow through the segmentation tree. If a node i is similar to is neighbor j, that is, the χ^2 distance between $\mathfrak{D}(i)$ and $\mathfrak{D}(j)$ is small, we want i and j to have similar beliefs. If $\mathfrak{D}(i)$ is dis-similar to $\mathfrak{D}(j)$ we don't care whether i and j have similar beliefs or not. To achieve this, we adopt the local potential function of Equation (5.5), as $\lambda_{i,j} \in (0, 1)$ we can control the growth of the exponentials.

$$\psi_{i,j} = \begin{bmatrix} e^{\lambda_{i,j} \cdot \gamma} & e^{-\lambda_{i,j} \cdot \gamma} \\ e^{-\lambda_{i,j} \cdot \gamma} & e^{\lambda_{i,j} \cdot \gamma} \end{bmatrix} \tag{5.5}$$

Where γ is a free parameter that controls the rate of growth of the exponential. In our experiments, setting $\gamma = 100$ was determined empirically as the best value for all object classes.

Examining Equation (5.5) more closely, we can see that when $\lambda_{i,j} \to 0$ the exponentials approach 1, and the $\psi_{i,j}$ matrix becomes constant. This occurs when the χ^2 measure is large[1]. This basically serves to sever the link between i and j in the tree. When the similarity measure $\lambda_{i,j}$ is large, $\psi_{i,j}$ approaches a diagonal matrix, enforcing our desire for i and j to have the same belief.

As the underlying graph structure is strictly an N-ary tree, we can achieve convergence in our beliefs using a single forward-backward pass, making the computational time linear in the number of nodes. Let $C(i)$ be the i'th node at the current level, and $P(i) = Parent(C(i))$ be its parent in the tree. We can make our first pass by computing the message $m_{(C(i) \to P(i))}$ for each $C(i)$ in the lowest level of the segmentation tree, we then proceed upward through the tree. We can effectively ignore the tree structure in this computation, and simply scan across all nodes at each level linearly, propagating, and storing the message at the next highest level as we go. We proceed in this way until we have passed a message to the root node in each segmentation tree.

[1] $\mathfrak{D}(i)$ is very dis-similar to $\mathfrak{D}(j)$

After completing the forward pass, the root node has received messages from all of its neighbors, and is ready to compute its final message. Let $P(i)$ be the i'th node at the current level, and $C_j(i)$ be the j'th child of $P(i)$. We now compute the message $m_{(P(i)\rightarrow C_j(i))j\in J}$, for each child in the set of all children J. Each message will be different, as the $\psi_{P(i),C_j(i)}$ matrix will differ for each child in J. We once again proceed down the tree in this fashion, scanning linearly across a level in the tree, and propagating messages down to the child nodes.

Upon completion of the backward pass, all nodes have received messages from all neighbors, and are ready to compute their final beliefs using Equation (3.7). The final probability at each node is equal to its belief after receiving all messages from its neighbors $N(x)$ in the tree:

$$P(L(x) = 1|\mathfrak{D}(x), b(y))_{y\in N(x)} = \frac{1}{Z}\phi(x) \prod_{j\in N(x)} m_{j\rightarrow x} \tag{5.6}$$

This final probability value can be thresholded to produce a final label for the patch.

Chapter 6

Results

We will now examine the performance of our classifier using a standard quantitative metric. Our classifier is trained using a standardized dataset, and therefore allows us to compare our results to other pre-existing systems. First we will discuss our comparative metric, then we will examine our dataset, finally we will examine the results of training our classifier to detect several object classes.

6.1 Receiver Operating Characteristics

To allow for comparison between different system configurations, we need to choose a metric that is independent of any underlying threshold values we may choose. For example, if we choose to simply summarise our results as a detection rate figure[1] we are loosing a great deal of information about the underlying performance of our classifier. We do not know where the majority of classifications fall relative to the threshold, or how changing the threshold affects our classification rate.

To counter these limitations of simple detection rate statistics, we choose to use the Receiver Operating Characteristic (ROC) curve as our main qualitative metric. This metric has been shown [6, 11] to be a very good measure of performance in classification tasks, and is used in both the vision and medical communities. The ROC curve is constructed by plotting the correct detection

[1] Detection rate can be calculated by thresholding the beliefs at some value τ, then computing the number of correctly classified regions divided by the total number of ground truth positive regions

rate versus the false alarm rate, as we sweep the detection threshold through all possible values. We can then summarize the curve into a single statistic by integrating the area under the curve, to compute the AROC (area under receiver operating curve) statistic.

To construct a ROC curve we use the following process. First let R represent all regions within the image $\mathfrak{I}$, next let N represent all regions in R that have ground truth negative labels, and P represent all regions with ground truth positive labels. We must ensure $N \cup P = R$ which ensures every region is given a ground truth label. We now concatenate all of our belief values into a list $T(i) = P(L(i) = 1|\mathfrak{D}(i)), i \in R$, this will be the set of thresholds we will sweep over to construct the curve, we then sort this list such that $T(i) > T(i+1), \forall i$.

For each $\tau \in T$ we compute a correct detection rate using Equation (6.1), and a false alarm rate using Equation (6.2). Here I is the indicator function, and takes the value 1 when the regions belief is equal to or greater then the current threshold value.

$$CD = \frac{\sum_{i \in P} I(P(L(i) = 1|\mathfrak{D}(i)) \geq \tau)}{\sum_{i \in P} 1} \tag{6.1}$$

$$FA = \frac{\sum_{i \in N} I(P(L(i) = 1|\mathfrak{D}(i)) \geq \tau)}{\sum_{i \in N} 1} \tag{6.2}$$

Finally we plot the correct detection rate versus the false alarm rate for each threshold value in their sorted order. Figure 6.1 shows the ROC curves we would expect from a perfect classifier and a random guess classifier. The perfect classifier shows a 100% correct detection rate with a 0% false alarm rate, while the random guess classifiers correct detecton and false alarm rate should grow in synch with each other. The AROC values for the perfect and random guess classifiers should be 1 and .5 respectivly, and we would expect any classifier to fall within these bounds[2].

[2] An AROC value of less then .5 indicates the classifier is performing worse then random, and therefore is of no use

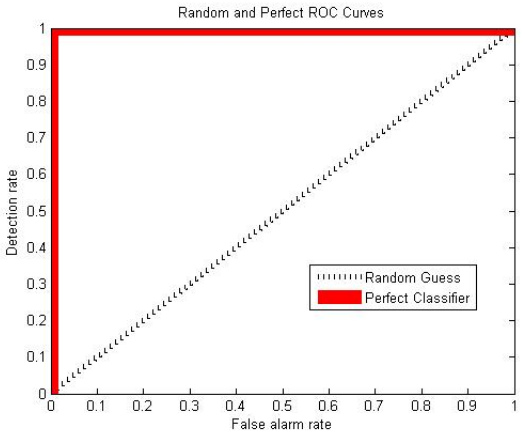


Figure 6.1: ROC curves showing a perfect classifier, as well as the curve we would expect to be generated using a random guess classifier.

6.2 Dataset

We use the PASCAL visual object challenge (PASCAL VOC) dataset [21] for both training and evaluating the performance of our detector. We make this choice as it allows us to compare our results against the published performance of several other state of the art detectors. Please refer to Appendix B for some sample images from the data set.

We have chosen to re-annotate the images in the dataset with silhouettes. We discovered early in the development of our detector that bounding box annotations were insufficient to train our classifier, as many regions were given incorrect ground truth labels. This is due to the fact that we are applying a ground truth label to each region based on its overlap to the annotation. With a base rectangular bounding box, we would apply a positive ground truth label to regions between the a cows legs for example. The result of these mis-annotations is a weaker base classifier, as the positive class training set presented to boosting is polluted with negative class segments. The silhouette annotations as shown

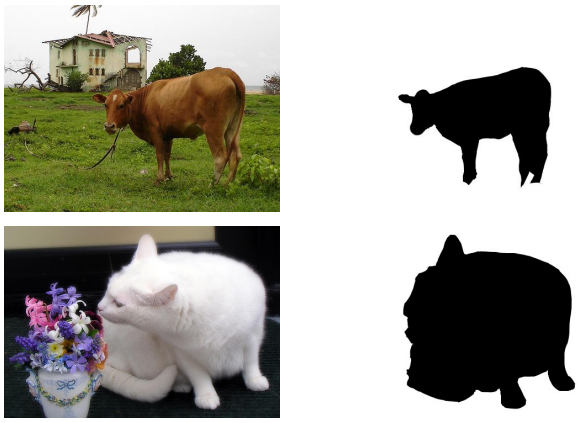


Figure 6.2: Silhouette annotations for a sample cow and cat image.

in Figure 6.2, eliminate this dataset pollution.

6.3 Base classifier

We begin to examine the results of our system at the base classifier level. For the cow class data we can examine the features chosen by boosting. The first ten features chosen alternate between color and texture $[C, T, C, T, C, T, T, C, T]$, which shows that boosting does not focus on a specific feature type, rather chooses the most discriminative from all possible features.

Figure 6.3 shows the details of the features chosen by boosting, as well as their associated thresholds. We can see by examining the feature vectors and thresholds that not all features are discriminative or even correct, but when combined in the boosting framework return higher scores in segments that are of the positive class. Table 6.1 shows the performance of the base classifier on several object classes. We have trained classifiers using the color and texture features independently, as well as a final classifier that combines both. This is done to ensure that we are not over fitting, or only using color or texture information. It is interesting to note that the color features are a stronger base

classifier for the cow objects, while the texture serves as a stronger individual classifier for the cat images. This is not surprising as cows tend to exist in outdoor environments, and learning that green and blue never belong to the object class serves to quickly label large regions as background. While cats tend to exist in a mix of both indoor and outdoor environments, where a larger variance in background color exists. In both cases the combined features provide a stronger base classification then either of the individual feature sets.

Object class	Base classifier AROC		
	Color Features	Texture Features	Combined Features
Cow	0.818	0.809	0.894
Cat	0.734	0.792	0.816
Car	N/A	N/A	.736

Table 6.1: Base classifier performance using only color, only texture, and combined color and texture features.

6.4 Hierarchical model

We will now examine the behaviour of the hierarchical model in more detail. For all of our experiments we use a 3 level hierarchy, this was chosen as it provides a good trade off between a range of reasonable segmentations and overall memory usage. Forcing the segmenter to choose patches larger then our coarsest grain segmentation leads to strong under-segmentation, which in turn leads to poor classification results[3]. Choosing segmentations finer grained then our minimum segmentation leads to a large number of very small, meaningless patches.

Figure 6.4 shows the segmentation trees generated by three segments in the example image. We can see many interesting interactions in these three small

[3]Strong under-segmentation leads to regions that are a mix of both positive and negative class.

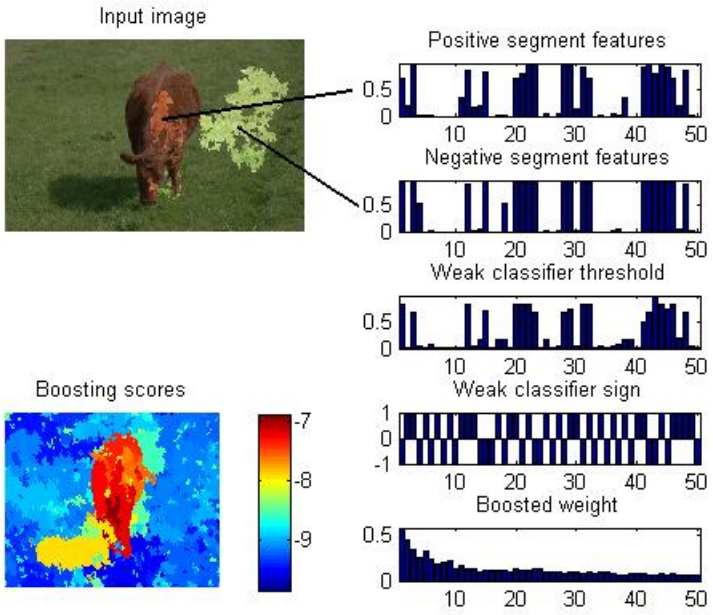


Figure 6.3: The details of the first 50 features chosen by boosting. The top right bar graph shows the raw histogram bin values of the first 50 features extracted from a positive patch within the image on the left. The second graph shows the features extracted from a negative patch. The third bar graph is the threshold value chosen by boosting for that feature, while the fourth graph indicates the sign (in this case which side of the threshold we expect the positive class to be on). The final graph is the α thresholds chosen by the boosting algorithm. The bottom left image is the resulting score assigned to each segment by the boosting algorithm.

trees. The first is in the small positive class subtree (nodes ii and d in Figure 6.4(d).) that corresponds to the cows ear. In the coarsest segmentation it is lumped in with the large background segment, and given a very low probability of being positive class. When it is segmented separately in the next level of the hierarchy, it is assigned a much higher belief. Due to the dis-similarity between the large background segment (Node 1) and the cows ear(Node ii) in feature space, our belief propagation algorithm assigns a low compatibility score to the connection between the nodes in the hierarchy. Consequently Node 1 does not force its belief on Node ii, and our belief in the label of the ear segment remains high as we would expect.

We can see another interesting behavior in Node c of Figure 6.4(d). We see that it is initially assigned a relatively large probability of taking on the positive class label, it is also given a fairly high weight connecting it to its parent. Consequently the belief propagation algorithm pushes the belief in it being a positive class node down, and we become more certain that it is a negative class node, as ground truth indicates it is.

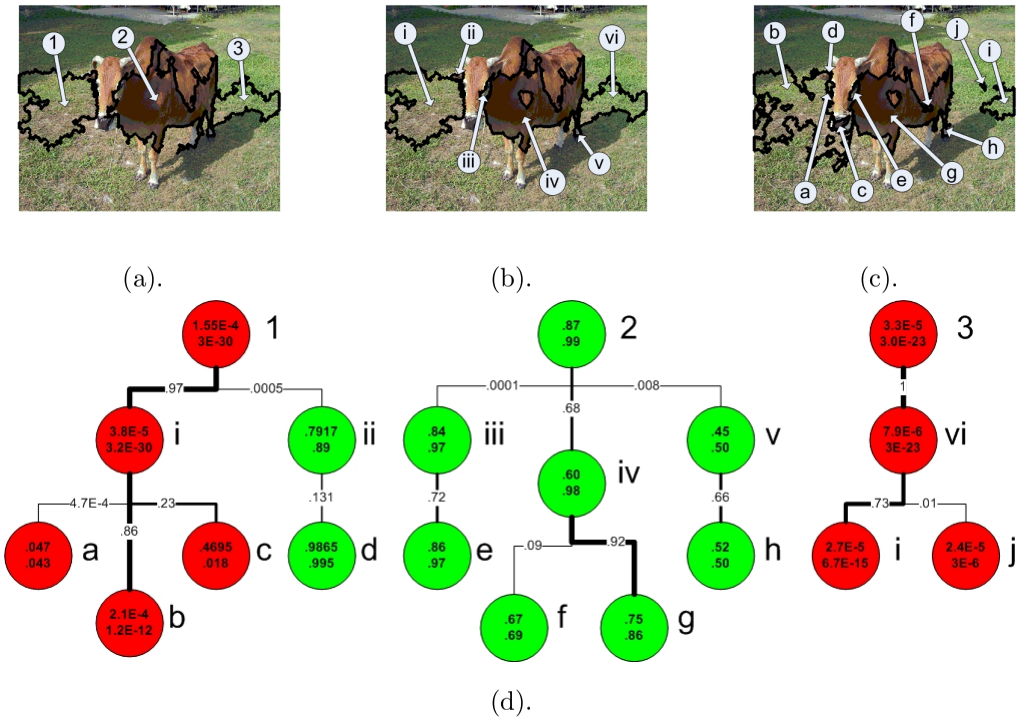


Figure 6.4: The tree in (d). is the CRF induced by the segmentations of (a), (b), and (c). The dark nodes represent ground truth negative nodes, while the lighter nodes represent ground truth positive. The top value in each node is our belief $P(L(i) = 1|\mathfrak{D}(i))$, while the bottom node is our belief after running belief propagation over the tree. The thickness of the connecting lines represents the compatibility score λ_{i-j} between the two nodes.

6.5 Cow classifier results

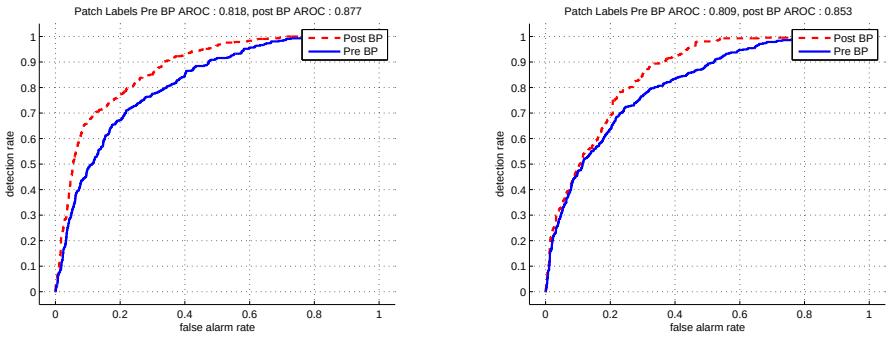
Figure 6.5 presents the ROC curves for our cow classifier. Each graph presents both the base classifier performance, and the results after running belief propagation on the segment tree. Table 6.2 summarizes the AROC values for these curves. We can see that the hierarchical approach provides a larger improvement in ROC performance for the color and texture only classifier cases then it does for the combined feature classifier. This is due mainly to the fact that the underlying classifier performance improves to the point that the is not much room left for the hierarchical model to further improve results. Patches that the underlying classifier mis-classify in the combined model are assigned such low confidence in all scales of the hierarchy that the belief propagation has no information to be able to correct the errors. While an improvement of two percent in classifier performance does not, on the surface seem to be large, a recent object detection competition[4] shows that the difference in performance between many current leading techniques is in the same order of magnitude. The leading classifier submitted during the 2006 PASCAL VOC challenge was the work of Perronnin [23], which reports AROC performance of 0.940 (as reported in [8]). Table 6.3 compares our technique to several PASCAL VOC entries, the results show that we are competitive on similar data with several other leading techniques.

Figure 6.6 plots the ROC curves at each level of the hierarchy before running belief propagation, as well as the final ROC curve at the finest grain segmentation after applying our hierarchical model. We can see that in this case, the base classifier performance at the fine grain segmentation is quite poor in comparison to the coarser grain segmentations. However our hierarchical model is able to boost the overall classifier performance up to match the best base classifier, while maintaining the fine grain segmentation boundaries.

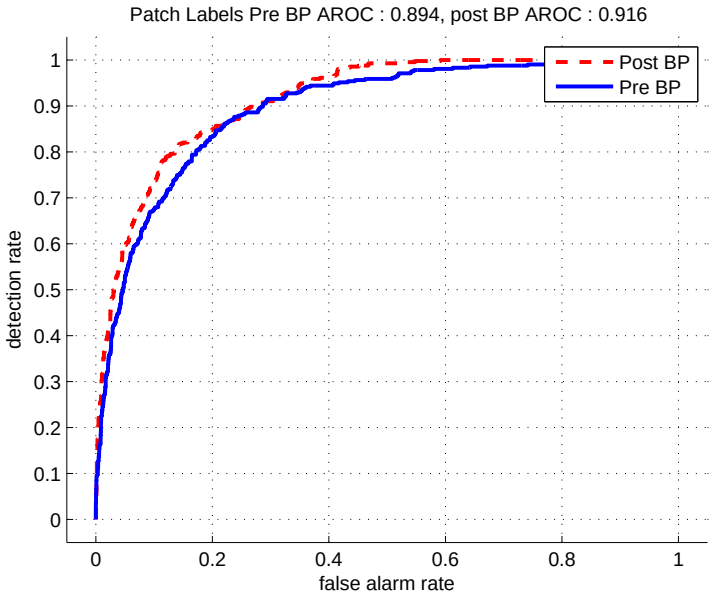
[4]The PASCAL Visual Object Challenge 2006

Figure 6.7 shows a sample of detector outputs. The left most image is the raw input image, the middle image shows the confidence in our detections after running belief propagation over the resulting tree. We are able to detect single and multiple cows in a variety of natural scenes, at a large variety of scales with very high confidence.

Figure 6.8 shows two examples where our classifier fails to correctly label the regions. In the first image we label a large portion of the ground as being cow. In feature space, the large patch of ground looks cow like, having both a similar color and texture. Due to this, boosting assigns the region a high base probability of having a positive class label. Our hierarchical model is unable to correct this error as at no level of the hierarchy does the region not look cow like, and our belief in the label is re-enforced. In the second image of Figure 6.8, we fail to correctly label the positive class cow. In this particular example the cow has an unusual color and very fine hair. It is not unexpected that we would have regions of dead grass in many of our training images with similar coloration, and therefore our underlying classifier has learned to label such regions as belonging to the non-object class. The majority of the errors being made are not of such a large scale, and primarily consist of small regions that are mis-labeled, or regions in very heavy shadows where little color or texture information exists. In the case of areas in heavy shade we would expect our segmenter to choose the same boundaries in all levels of the hierarchy, as they tend to have very strong boundaries. Therefore our hierarchical model is unable to improve our belief in the label, as all nodes in the resulting tree have low belief.



(a). (b).



(c).

Figure 6.5: ROC curves for (a). Color only Features, (b). Texture only features, and (c). combined color and texture features. Both base classifier and post belief propagation curves are plotted. Generated using (a). RunCowDetectorColor.m, (b). RunCowDetectorTexture.m, (c). RunCowDetectorCT.m

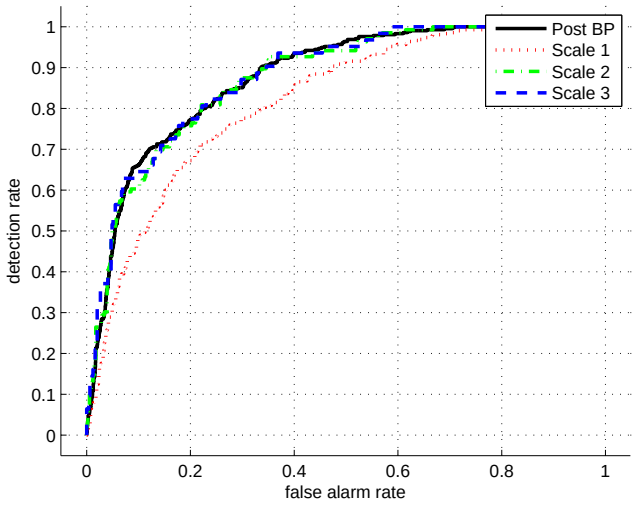


Figure 6.6: ROC curve for the color feature only base classifier plotted for each level independently in the hierarchy. The ROC curve after running belief propagation lives within the maximal hull of all of the segmentation granularity performances. Generated using RunCowDetectorColor.m

Object class	classifier AROC		
	Color Features	Texture Features	Combined Features
Cow base classifier	0.818	0.809	0.894
Cow BP	0.877	0.853	0.916
Improvement	0.059	0.044	0.022

Table 6.2: Base classifier performance using only color, only texture, and combined color and texture features. Cow BP shows the results after running belief propagation over the segment tree.

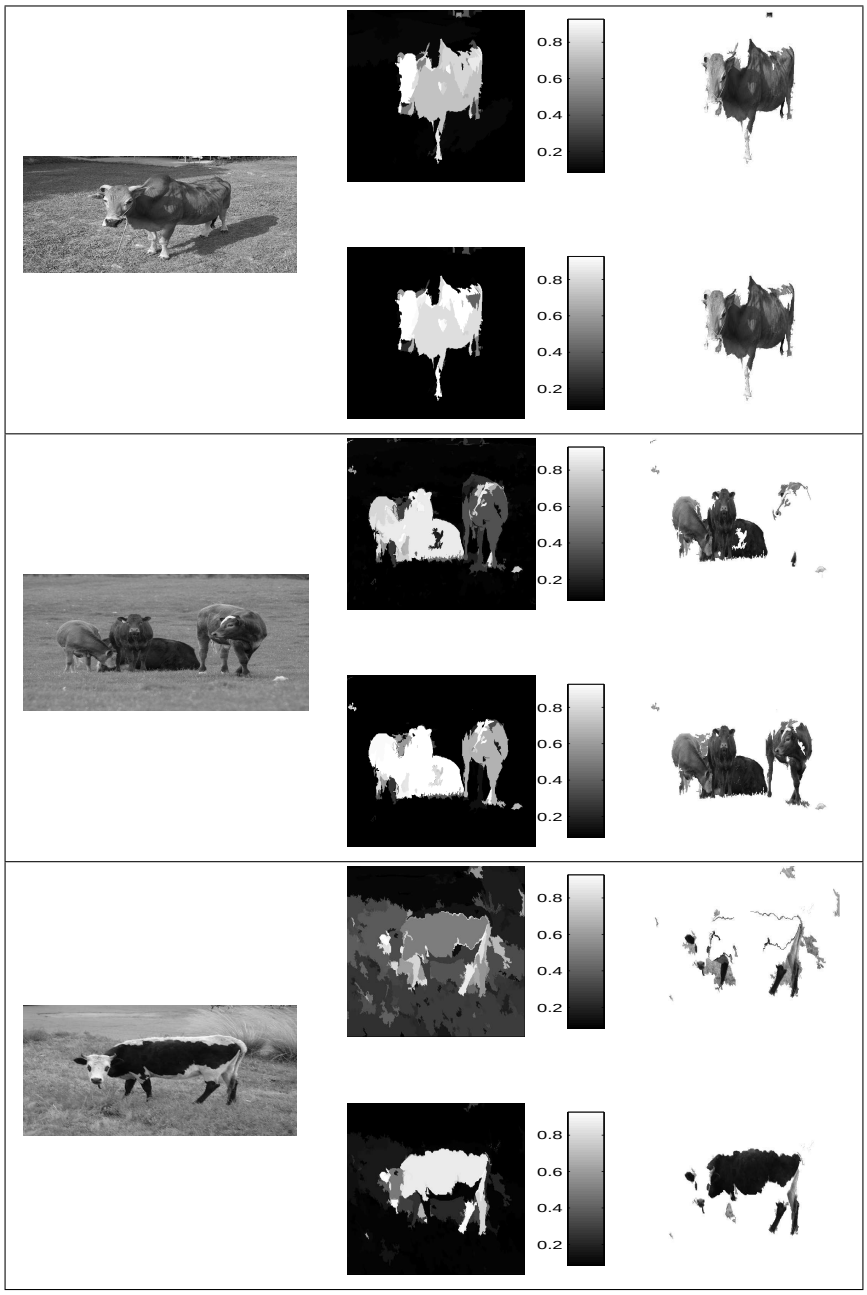


Figure 6.7: Sample detector outputs. The left most image is the input to the detector, the middle image represents our positive class label beliefs. The right most image shows those pixels that have a greater then 50% confidence in positive class labels. The top row shows our belif before applying our heirarchical model, while the bottom row shows the results after running BP. Generated using RunCowDetectorCT.m

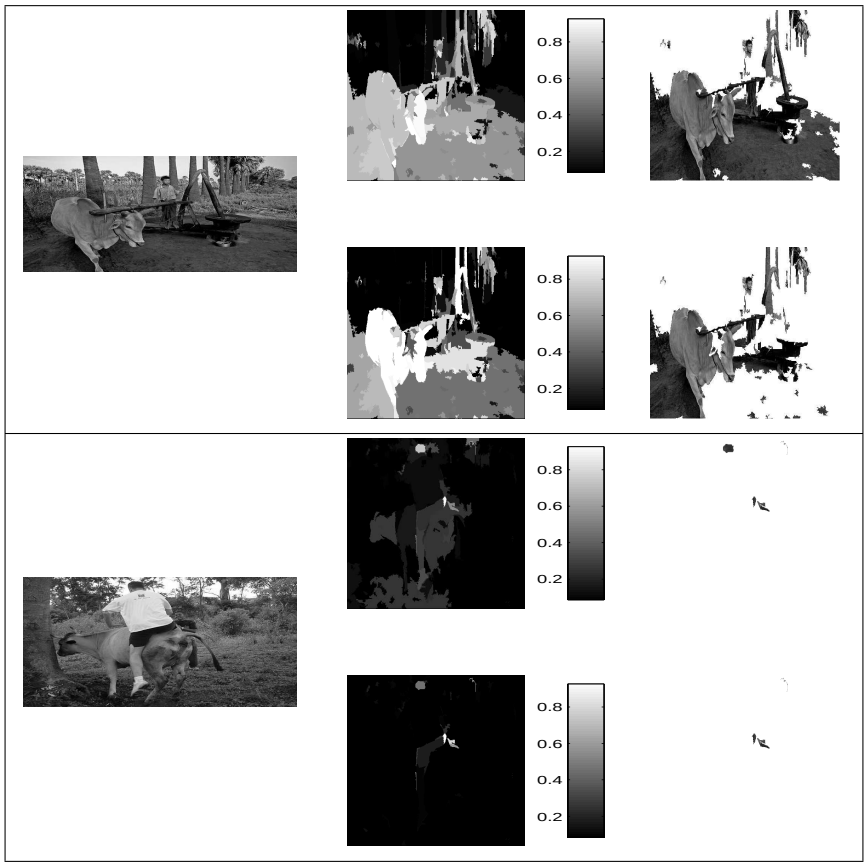


Figure 6.8: Failures in detection. The first labels a large portion of the background as belonging to the positive class, while the second fails to correctly label the positive class region. Generated using RunCowDetectorCT.m

Method	AROC
XRCE	0.94
QMUL LSPCH	0.938
INRIA Marszalek	0.922
* Hierarchical Segment Boost	0.916
RWTH Dischist	0.910
INRIA Moosmann	0.895
* Segment Boost	0.894

Table 6.3: A comparison of our results with the PASCAL VOC challenge entries. Segment boost is our base classifier, while Hierarchical segment boost is after applying our hierarchical model. Please refer to [8] for the details of the various techniques.

6.6 Other class results

We trained our classifier on two additional object classes available in the PASCAL VOC dataset; Cat and Car. Table 6.4 summarizes these results. Table 6.5 compares our results to other techniques applied on similar data as part of the 2006 PASCAL VOC. Our performance gain by applying the hierarchical model on the cat class data is similar to that we see in the cow class. Figure 6.9 shows the ROC curve produced by our cat detector, while Figure 6.10 shows some sample detections. We can see in this case, some of our gains are coming in the form of removing false positive patches. Additionally, the classifier performance while not as great as some approaches presented in [8], falls within the mean of performances reported on similar PASCAL data. However our base classifier model does not perform as well as the cow detector in this case. The weaker base classifier performance can be attributed to two main factors:

- Larger intra class variation. Cats have a larger coloration and textural variability then cows.

- Larger non-object class appearance variations. While the environments in which we expect to encounter cows is somewhat limited[5] the environments in which we may encounter cats is unlimited.

We will now look at each of these factors in more detail.

The larger intra class variation in appearance of cats makes the task of learning a discriminative classifier more difficult. As boosting chooses features and weights that best separate data within the training set, encountering novel color and texture combinations in the test set decreases our overall confidence in our positive class label. While the boosted classifier is able to generalize over some changes by having different sets of weak classifiers activate, it does so at the cost of confidence in the final label. For example, if we encounter a long haired cat with a coloration that we have not seen in our training data, the texture features will strongly tell us that the region is cat, however the color features will tell us that were unsure of the label for the region. As we sum these resulting weak classifier outputs, our overall belief in the label moves toward uncertainty.

The larger variability in non-object class appearance can lead to a higher false positive rate. As cats can appear in a large variety of environments, it is not inconceivable that we will encounter an image with a rug or upholstery with similar local appearance features as a cat. These cat like local descriptors lead to high base probability assigned to negative ground truth region. Our hierarchical model should be able to correct some of these, at least those that at a larger scale include enough information to dis-ambiguate the true label of the region. Figure 6.11 shows some sample failures in detection.

We additionally trained our detector on car objects. While cars are not deformable objects, we were interested in evaluating our performance on objects for which the system was not designed. Figure 6.12 shows the resulting ROC

[5] We would not expect to find a cow inside a house, or in a city

Object class	classifier AROC		
	Base classifier	post BP	Improvement
Cat	0.799	0.819	0.02
Car	0.736	0.744	0.008

Table 6.4: Combined feature performance before and after applying the hierarchical model for the cat and car classes.

Class	Our result	Best VOC	Ranking
Cat	0.819	0.937	12
Car	0.744	0.977	19

Table 6.5: Comparison of our results to other entries in the 2006 PASCAL VOC challenge. Ranking is where our results would be ranked on similar data within the 20 entries.

curve for our car detector, while Figure 6.13 shows some sample detections. We knew a priori that cars would present a different problem to our approach. As we are computing descriptors over local regions, we could predict that our base classifier would be very weak due to two primary factors :

- Locally car objects have very little texture. As we know our segmenter will choose strong image boundaries (for example it will segment a cars windshield, and the hood separately) we will not see any edge texture information.
- Cars tend to be of similar color to many objects in the environments in which we expect to encounter them.

The first factor limits the usefulness of textural information in assigning a positive class label to cars. It does however allow us to label regions as not car, that is if we encounter areas of high texture (for example trees, the road surface, grass) we can be quite sure that it is not a car. The second limits the usefulness

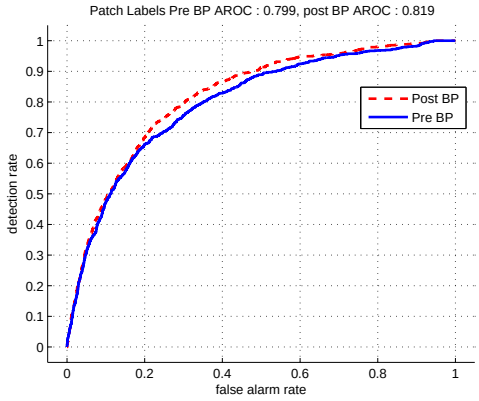


Figure 6.9: ROC curve for the cat object detector, using the color and texture features. Generated using RunCatDetectorCT.m

of color features. It would not be surprising to find many buildings or signs that have coloration similar to cars. Our hierarchical model fails to provide a large boost in this case due to the fact that our segmenter will always choose boundaries on strong image boundaries. We can see in the examples presented in Figure 6.13 that the regions we tend to correctly label are the ones that contain some textural information, while the large flat surfaces such as the hood, windshield, and side panels are missed.

The outline of a car and the surrounding environment will, in most cases, be a very strong image boundary, as will the outline of a cars windows or doors. Because of this the segmenter will choose to segment the image on these strong image bounds regardless of the parameter settings (for example we cannot easily force our segmenter to segment a car as a single region). This leads to building chains rather then trees in the segmentation hierarchy in which each node will have a similar or the same belief. In the belief propagation step, our belief in the labels will not change, as the parent has nearly the same belief sate as the child.

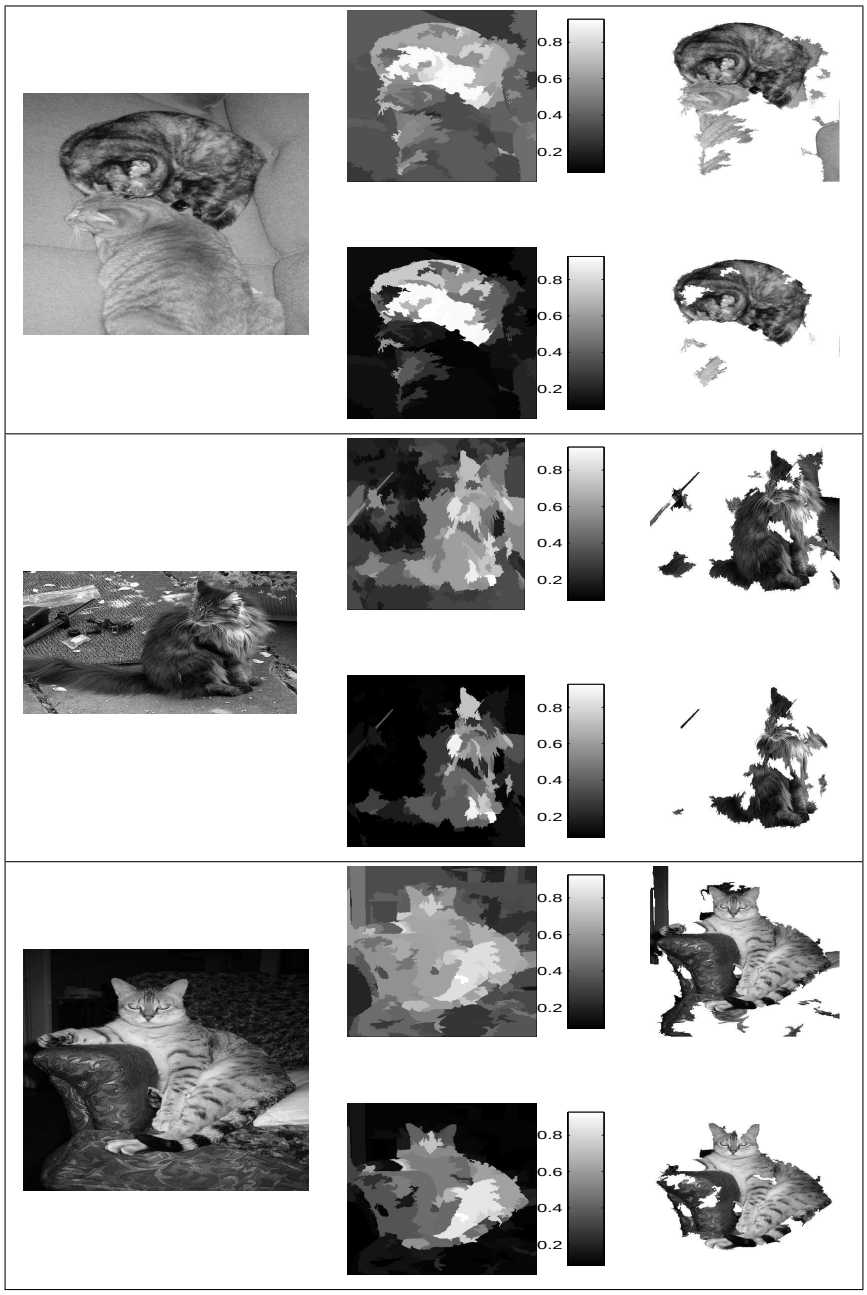


Figure 6.10: Sample detector outputs. The left most image is the input to the detector, the middle image represents our positive class label beliefs. The right most image shows those pixels that have a greater then 30% confidence in positive class labels. The top row shows our belief before applying our heirarchical model, while the bottom row shows the results after running BP. Generated using RunCatDetectorCT.m

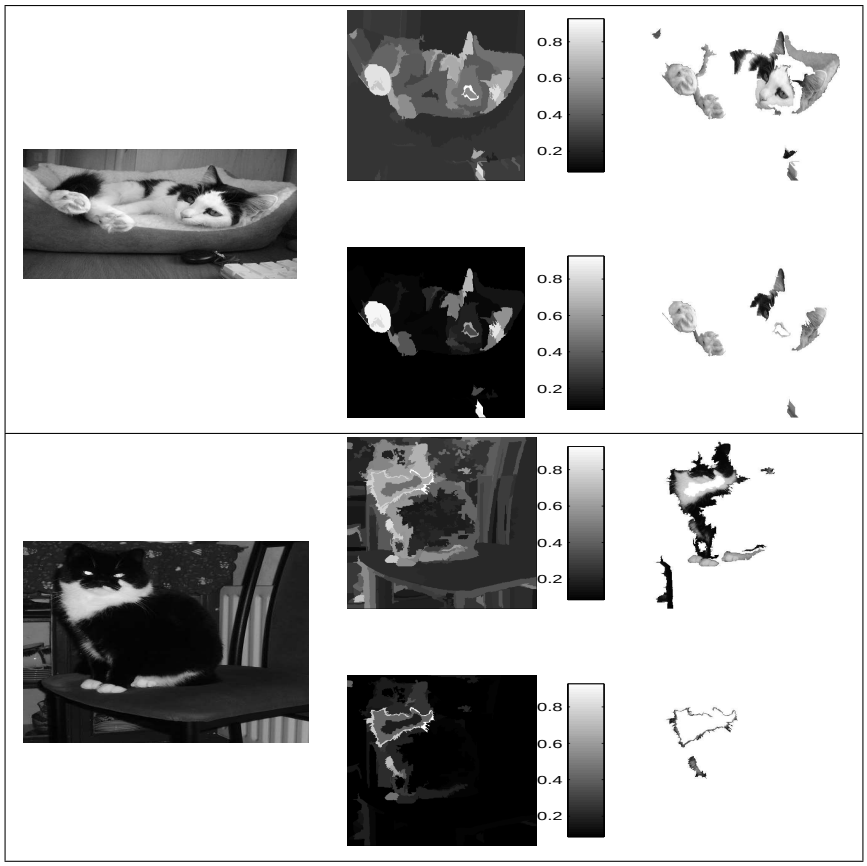


Figure 6.11: Sample failed detector outputs for our cat class detector. Generated using RunCatDetectorCT.m

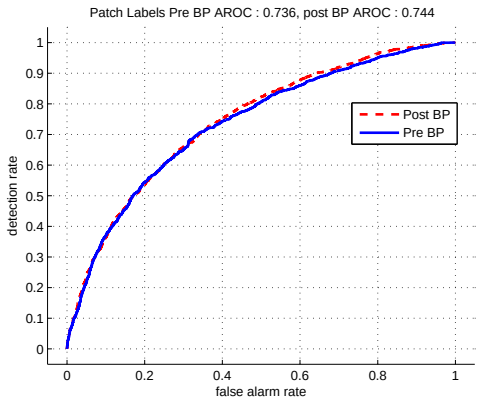


Figure 6.12: ROC curve for the car object detector, using the color + texture features. Generated using RunCarDetectorCT.m

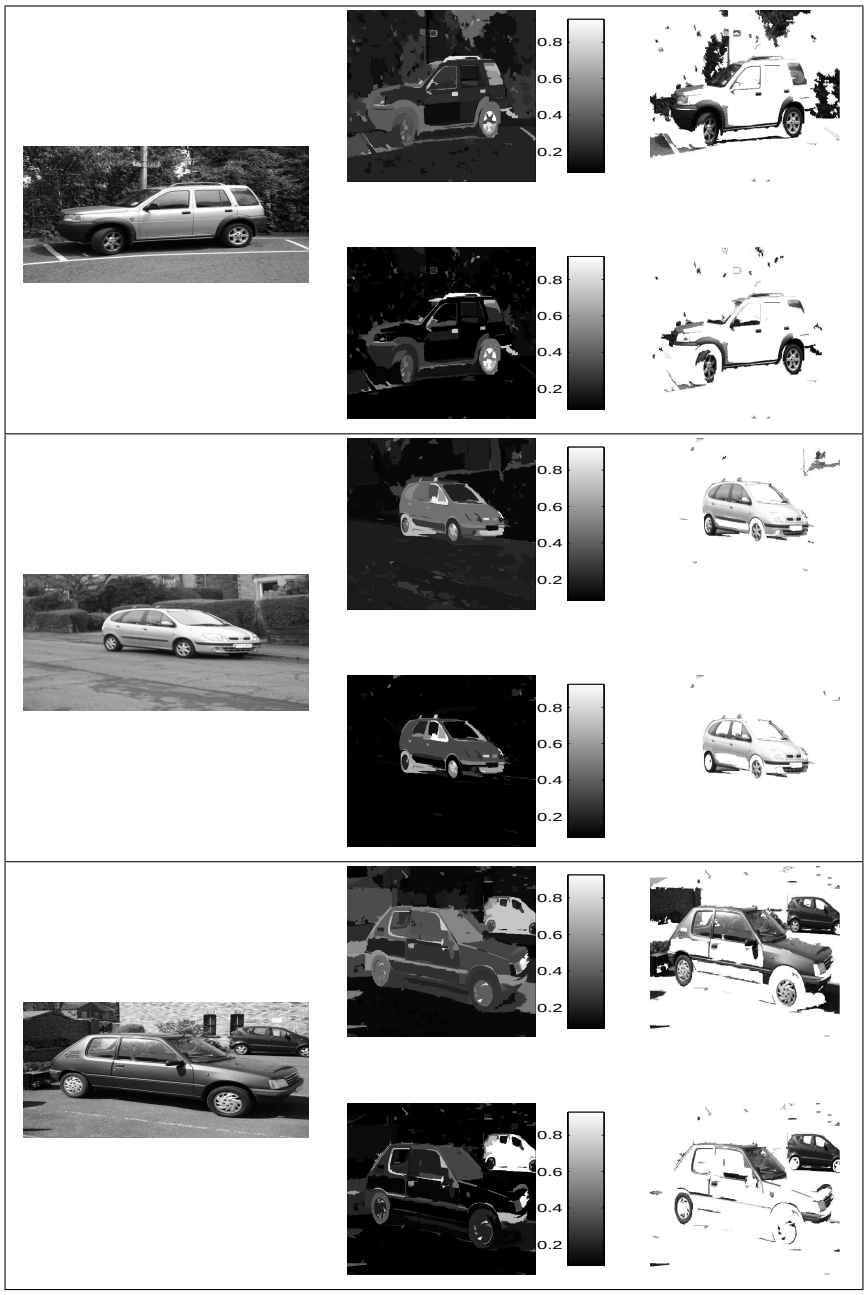


Figure 6.13: Sample detector outputs from our car detector. The left most image is the input to the detector, the middle image represents our positive class label beliefs. The right most image shows those pixels that have a greater then 30% confidence in positive class labels. Generated using RunCarDetectorCT.m

6.7 Run time

We have implemented our system entirely in the Matlab environment under Windows. Computing the training set features and training the classifier with 100 images takes approximately 3 hours total computation time on a modern Pentium 4 2.66GHz desktop PC with 1 GB of RAM. Processing and classifying a new image takes approximately 2 min per image on the same machine. Most of this computational cost comes from the Gabor filter operations. As our Gabor filter function is implemented in Matlab, it creates a large computational bottleneck. We are confident that, with a proper C implementation we can get the total processing time into the order of twenty to thirty seconds per image.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

We have presented a new approach to identifying non rigid objects in natural images. This new approach uses only local information computed over segments to apply a base label to each pixel. Additionally we construct a hierarchy over several granularities of segmentation in order to solve the aperture problem. We then introduced a method employing belief propagation to allow information to flow through this hierarchy. Our final detections integrate the information from all levels of the hierarchy, while maintaining the object bounds we find at the finest grain level of segmentation.

We have shown that our hierarchical model improves the performance of our base classifier for several object classes. And that the overall system performance is close to many current techniques. While there is still room for improvement in the final classification results, we have shown that our approach is a viable method for identifying completely deformable objects within natural images of varying resolution and quality.

We have further shown that the hierarchical model we present does boost the performance of a fairly weak base classifier. It is quite possible that this technique, when integrated with a better base classifier, will produce even better

classification performance on a larger range of objects.

7.2 Future work

We have presented here an approach that is capable of identifying deformable objects in natural images. There is still however room for improvement in our current approach. Here we present some still open problems, and some possible extensions to build on this work.

7.2.1 Base classifier performance

While we have shown that our hierarchical model is able to improve the performance of our classifier, the overall system performance is still heavily dependant on the underlying base classifier. We chose to use simple color and texture histograms as region descriptors, but it is quite possible that a different set of base features would provide stronger base classifier performance. For example, textons or patch shape information may provide stronger base classification for semi-ridgid object classes.

Several other approaches may also be adopted to train the classifier, we have chosen boosting as it provides a reasonable level of performance with only a small computational overhead. It also produces a fairly robust classifier, that is able to handle moderate amounts of intra class variation. Improvement in both the feature set and base classifier performance will reflect directly in the final overall system performance.

7.2.2 Hierarchical model

We have chosen to build a simple disconnected forest within our segmentation hierarchy, we can suggest two possible extensions on this model.

First we could form a single tree, with a super node who's child set contains

all nodes in the coarsest grain segmentation. This super node would have a feature descriptor computed across the whole image, and would provide a prior over our belief that the object exists within the image. This prior could be trained independent of the classifier.

Secondly we could choose to construct a planar CRF over each level of the segmentation hierarchy, and perform label smoothing. We chose not to do this in our implementation for two reasons :

- We assume that the boundaries at which the CRF would choose not to propagate labels would be the same boundaries our segmentation algorithm would cut on, and therefore we would never propagate labels between connected regions[1].
- We would need to devise some mechanism to either alternate between tree based belief propagation and planar operations, or solve the whole system as a three dimensional CRF. The first option may never converge, while the second would drastically increase our computational cost.

While it is not clear that these modifications will improve the performance improvements we see by applying the hierarchical model, they are certainly open areas to explore.

We have shown the power of our simple hierarchical model in improving the performance of a simple classifier. Integrating further information into the model will also increase the performance boost we are achieving. While we have simply integrated the information across a hierarchy of segmenters, it is quite possible to apply this same model using different classifier techniques at each level of the hierarchy, or building a more complete CRF model that is able to integrate information across both granularities, and techniques.

[1] That is, if two regions are similar enough that the planar CRF would choose to smooth the labels, they should have been segmented as a single region.

Bibliography

[1] S. Agarwal, A. Awan, and D. Roth. Learning to detect objects in images via a sparse, part-based representation. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 26(11):1475–1490, 2004.

[2] A. Berg, T. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondence. Technical report, UC Berkeley, 2004.

[3] E. Borenstein and J. Malik. Shape guided object segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 969–976, 2006.

[4] E. Borenstein, E. Sharon, and S. Ullman. Combining top-down and bottom-up segmentation. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 46–46, 2004.

[5] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift, 2000.

[6] D. Dorfman and E. Alf. Maximum likelihood estimation of parameters of signal detection theorya direct solution. *Psychometrika*, 33(1):117–124, 1968.

[7] G. Eibl and K.P. Pfeiffer. Multiclass boosting for weak classifiers. *J. of Machine Learning Research*, 6:189–210, 2005.

[8] M. Everingham, A. Zisserman, C. Williams, and L. Van Gool. The pascal visual object classes challenege 2006 (voc2006) results, Accessed : Aug 2006.

[9] P. Felzenszwalb and D. Huttenlocher. Efficient graph-based image segmentation. *Intl. J. Computer Vision*, 59(2):167–181, 2004.

[10] M. Fritz, B. Leibe, B. Caputo, , and B. Schiele. Integrating representative and discriminant models for object category detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 1363–1370, October 2005.

[11] J. Hanley and B. McNeil. The meaning and use of the area under the receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of Statistical Learning*. Springer, 2001.

[13] D. Heckerman. A tutorial on learning with bayesian networks. *http://research.microsoft.com/research/pubs/view.aspx?msr_tr_id=MSR-TR-95-06 : Accessed June 2006*, 1995.

[14] T.I. Hsu, A. Calway, and R. Wilson. Texture analysis using the multiresolution fourier transform. In *Proc 8th Scandinavian Conference on Image Analysis*, pages 823–830, 1993.

[15] A. Jain, N. Ratha, and S. Lakshmanan. Object detection using gabor filters. *Pattern Recognition*, 30:295–309, 1997.

[16] B. Leibe, A. Leonardis, , and B. Schiele. Combined object categorization and segmentation with an implicit shape model. In *ECCV'04 Workshop on Statistical Learning in Computer Vision*, May 2004.

[17] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. In *Intl. J. Computer Vision*, pages 29–44, 2001.

[18] D. Lowe. Distinctive image features from scale-invariant keypoints. *Intl. J. Computer Vision*, 60(2):91–110, 2004.

[19] T. N. Mundhenk and L. Itti. A model of contour integration in early visual cortex. In *Lecture Notes in Computer Science*, volume 2525, pages 80–89, Nov 2002.

[20] K. Murphy, A. Torralba, and W. Freeman. Using the forest to see the trees:a graphical model relating features, objects and scenes. *Advances in Neural info. Proc. sys.*, 16, 2003.

[21] PASCAL. Pascal voc 2006 dataset : http://www.pascal-network.org/challenges/voc/databases.html, Accessed : May 2006.

[22] P. Perez, C. Hue, J. Vermaak, and M. Ganget. Color-based probabilistic tracking. In *ECCV*, volume 2350, pages 661–675, 2002.

[23] F. Perronnin, C. Dance, and G. Csurka. Adapted vocabularies for generic visual categorization. In *ECCV*, 2006.

[24] T. Rikert, M. Jones, and P. Viola. A cluster-based statistical model for object detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1046–1053, 1999.

[25] R. Schapire. The boosting approach to machine learning: An overview. In *RSRI Workshop on Nonlinear Estimation and Classification.*, 2001.

[26] R. Schapire, Y. Freund, P. Bartlett, and W.S. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Intl. Conf. on Machine Learning*, pages 322–330, 1997.

[27] H. Schneiderman. Feature-centric evaluation for efficient cascaded object detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 29–36, 2004.

[28] N. Petkov S.E. Grigorescu and P. Kruizinga. Comparison of texture features based on gabor filters. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 11(10):1160–1167, 2002.

[29] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV*, 2006.

[30] A. Torralba, K. Murphy, and W. Freeman. Sharing visual features for multiclass and multiview object detection. *Proc. IEEE Intl. Conf. on Image Processing*, 2006 To appear.

[31] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 511–525, 2001.

[32] P. Viola and M. Jones. Robust real-time object detection. *Intl. J. Computer Vision*, 57(2):137–154, 2004.

[33] T. Weldon, W. Higgins, and D. Dunn. Efficient Gabor filter design for texture segmentation. *Pattern Recognition*, 29(12):2005–2015, 1996.

[34] J. Yedidia, W. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *Intl. Joint Conf. on AI*, 2001.

Appendix A

RGB to HSV conversion

If we consider the RGB color space as a rectangular co-ordinate system, with red along one axis, blue along a second and green along the third, we can in a similar fashion consider the HSV space as a cylindrical co-ordinate system with hue as the radial distance from the center, saturation as the angular displacement, and intensity as the vertical locaiton on the cylinder. It is the nature of this co-ordinate system that makes the HSV space useful to us. If we look at a single slice through the cylinder, one would see all possible colors, at a fixed intensity. Each slice through the cylinder would produce the same colors, but at a different intensity. Because of this examining only the HS values within an HSV image provides an intensity insensitive way to examine the color content. The conversion between RGB and HSV is a simple co-ordinate mapping. Equation (A.1) outlines the required transformations.

$$
\begin{aligned}
H_{(x,y)} &= \arccos\left[\frac{.5\cdot[(R_{(x,y)}-G_{(x,y)})+(R_{(x,y)}-B_{(x,y)})]}{\sqrt{(R_{(x,y)}-G_{(x,y)})^2+(R_{(x,y)}-B_{(x,y)})\cdot(G_{(x,y)}-B_{(x,y)})}}\right] \\
S_{(x,y)} &= 1-\frac{3\cdot min(R_{(x,y)},G_{(x,y)},B_{(x,y)})}{R_{(x,y)}+G_{(x,y)}+B_{(x,y)}} \\
V_{(x,y)} &= \frac{R_{(x,y)}+G_{(x,y)}+B_{(x,y)}}{3}
\end{aligned}
\tag{A.1}
$$

Appendix B

Dataset

Figures B.1 and B.2 show sample images from the PASCAL VOC Cow and Cat datasets respectivly.

Figure B.1: A sample of images from the PASCAL cow dataset.

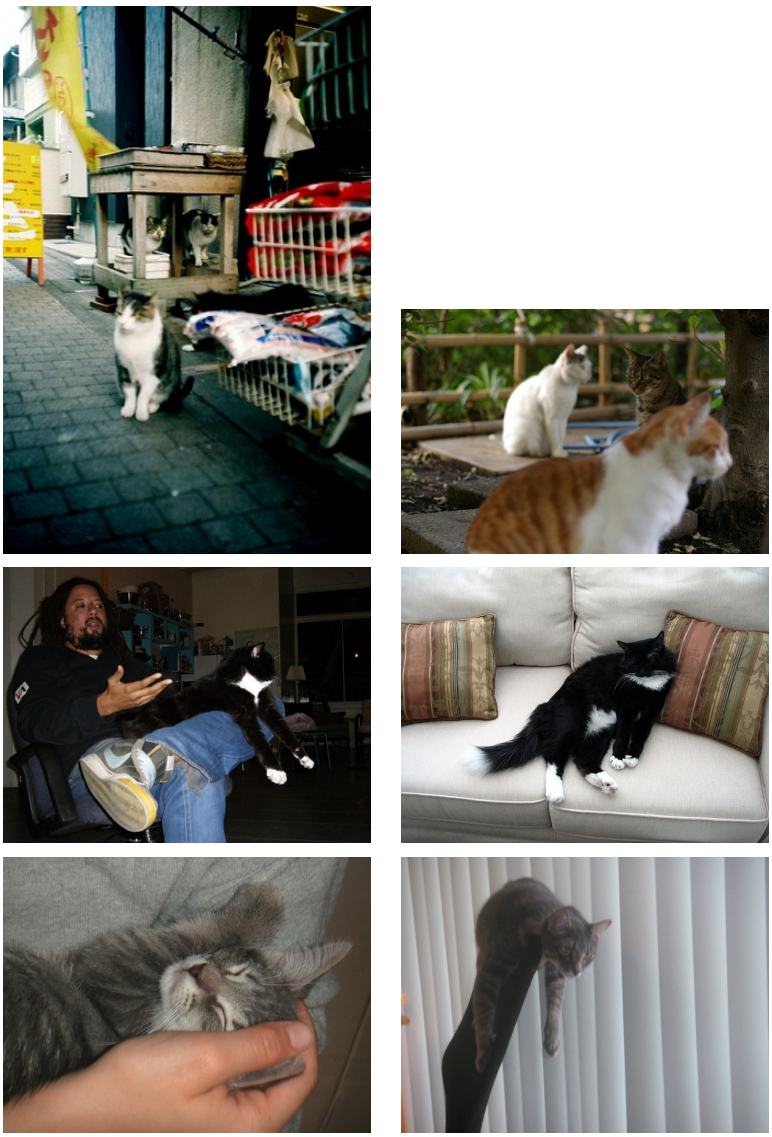


Figure B.2: A sample of images from the PASCAL cat dataset.