CAR DETECTION BY CLASSIFICATION OF IMAGE SEGMENTS

By

Robert Morgan Halpenny

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements for

the degree of

MASTER OF SCIENCE

in

Computer Science

December, 2008

Nashville, TN

Approved:

Professor Xenofon D. Koutsoukos

Professor D. Mitchell Wilkes

To my parents, for their years of support and encouragement.

ACKNOWLEDGMENTS

This work would not have been possible without help and support of many people. I would to thank my advisor Dr. Xenofon Koutsoukos, whose guidance and advice was infinitely valuable in allowing me to complete this project. I would also like to thank Dr. Mitchell Wilkes for graciously agreeing to be the second reader for my thesis. Dr. Constantine Aliferis and Dr. Alexander Statnikov provided invaluable assistance in formulating the feature selection and learning problem. Thank you to Hendrik Dahlkamp for providing the image dataset created using Stanford Racing Team's research vehicle "Junior."

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER I

INTRODUCTION

The ability to recognize general classes of objects is one of the most difficult unsolved problems in computer vision. In some areas, such as face recognition, considerable progress has been made; but the face is fairly rigid and faces are fairly similar structure compared to each other. Other objects such as cars, animals, or people are much harder to identify because of the wide variety of forms they may take. Often, recognizing a specific instance of an object may be significantly easier because it has a few particularly distinctive features, e.g. the stripes of a zebra or the tail lights of a corvette. In generalizing to a larger class of objects, such as equines or cars, those distinctive features are not usually available. In addition, it may be difficult to decide if two features belong to the same object, different instances of an object class, or completely different objects.

This paper will set forth a model for classifying segments of an image based on the existence of specific (SIFT) features in and around the segment. Our application is detecting cars in images of city streets. Recognizing cars is an outstanding problem of high importance in the field of automated driving. While range-finding sensors such as lasers and stereo vision can detect obstacles with high precision, it is difficult to anticipate and plan for objects that are potentially dynamic. In a driving situation, cars are among the most highly encountered dynamic objects (along with people) and it would be extremely useful to identify vehicles for route-planning purposes. The images we use for

testing and validation are still frames of video taken from the top of a car driving through San Francisco. This is a particularly difficult area because of the high car density and the background, which is often highly similar to texture of vehicles because of its man-made nature.

In brief, the algorithm will classify segments in the image as car or not-car, according to the presence of SIFT features in and adjacent to that segment. The useful features will be learned from a large dictionary by the HITON-PC algorithm. Segments will be classified by a support-vector machine (SVM) trained on these features. This algorithm is unique in that instead of finding a finding a region-of-interest in the image, it classifies segments directly. This has the benefit of providing more distinct boundaries around the edges of the car, and of excluding nearby regions that are definitely not of interest. Secondly, we use a Bayesian-learning feature selection algorithm to induce a small but powerful set of useful features for our classification problem.

Section 2 of this paper will present a literature review of the main algorithms used in this paper. For each area, we will try to survey the main algorithms and give some rationale as to why particular algorithms were chosen. The details of the four major sub-algorithms will be explained in sections 3-6. Section 7 will describe the architecture of the training procedure and the detection algorithm. Results will be presented in section 8, and the conclusion and areas for future study in section 9. Appendices A and B contain detailed results of training phases.

CHAPTER II

OVERVIEW OF APPLICABLE TECHNOLOGY AND PREVIOUS WORK

In the first four parts of this section, we will briefly discuss the body of work surrounding the four main algorithms used in our experiments. This is provided mainly as a justification for our choice of particular algorithmic solutions for problems we encountered. The last part contains a review of comparable work in the area of car detection.

2.1 Segmentation Algorithms

Segmentation may be used generally to describe any algorithm that groups similar parts of an image. In general, it is unfeasible to consider all of the pixels of an image separately. In practice, it is common to separate the image into regions or segments, and then perform further analysis on those segments as a whole. This reduces computational time and allows for more complex analysis. By dividing our image into segments, we can group features that are highly likely to be related.

Segmentation may be treated as a clustering problem with a few domain specific attributes added to the objective function. The clustering problem can be broken down into two parts: clustering algorithm and objective function.

2.1.1 Clustering Algorithms

There are 3 basic algorithms commonly used for solving clustering problems: k-means, eigenvector and graph-based methods.

K-Means Clustering

For k-means methods, the clustering is initialized by randomly placing k centroids into the feature space. Each piece of data is then clustered to the centroid that it is closest to. The centroids are recalculated to be the at the center of the data points that are assigned to it, and the first step is repeated until no changes occur in the composition of each cluster or some threshold condition is met.

Pseudocode for k-means clustering:

```
k_means(data, k)
//Initialize k centroids at random locations in the feature space
for each i =1:k
        centroids.addcentroid(randomcentroid())
end
//iterate
d_last = {}
until done:
        for each d in data:
                d.centroid = nearest_centroid(d)
        end
        for each c in centroids:
                c.location = average(d where d.centroid == c)
        end
        if d == d_last or timeout:
                done
        end
end
```

K-means clustering is very simple to implement and can give good results for well-defined problems. There are two main difficulties when applying k-means to computer vision problems: choosing an appropriate k and finding an appropriate distance function. In many applications, the appropriate number of clusters may be unknown. The viewing

area may change from simple, where a small number of clusters are appropriate, to complex, where a large number of clusters are appropriate. Using the wrong number of clusters can result in erroneous classification because dissimilar pixels are clustered together if there are too few clusters, or large objects broken up if there are too many. In addition, robustness can be an issue because incorrectly classified pixels can shift the mean of a centroid and cause further disruption in clustering. Secondly, applying an appropriate distance function can be difficult if the clusters are irregularly shaped (long, curved) because the algorithm does not easily lend itself to distance functions that compute distance from classified data (agglomerative link, etc.) instead of from a central point (the centroid.) Still, in problems which are largely similar (for example face recognition), k-means clustering is a simple and computationally efficient clustering algorithm.

Eigenvector Methods

Eigenvector methods (also known as spectral analysis) consider the associations between each pixel in the image. First, an affinity measure is chosen and the affinity between each pixel is computed and the results are placed in an NxN matrix (where N is the number of pixels.) Affinity can be thought of as the dual of a distance metric, in that higher values imply closer relationships between pixels. Then, the eigenvectors/eigenvalues for the matrix are computed. Under this method, each eigenvector/value pair corresponds to a cluster. For each eigenvector, every value indicates the strength of that pixel's affinity for the cluster. Clusters can be found, therefore, by selecting groups of pixels with large weights for all eigenvectors that have a large eigenvalue [10].

The eigenvector approach works very well in most situations. Since the clusters are determined by threshold values, it is resistant to grouping pixels that do not belong to any reasonable cluster. In addition, the number of clusters is not predetermined so this method is applicable to a wider variety of scenes. Unfortunately, the computational cost of computing the eigenvectors/values for the matrix can be prohibitively expensive, especially as the size of the image increases (for example a 1MP image requires solving a 1million x 1 million matrix). Another difficulty is in images where there is a gradient across a cluster. Eigenvector methods work best with clusters that are significantly similar to themselves.

Graph-based Methods

In graph-based clustering methods, the image is treated as a graph, with edges drawn between pixels. Exactly which pixels depends on the details of the algorithm used. The goal of these methods is to select edges in the graph to add (or remove) based on some heuristic search method. There are a number of different methods for creating clusters out of the graph, such as solving the minimum-spanning-tree for the graph, splitting and merging regions based on uniformity of a region, and performing normative-cuts on the graph.

The earliest graph-based methods worked by simply breaking connections between highly dissimilar nodes [35]. While this makes intuitive sense, such methods will poorly classify regions that have internal variation, such as a patch of grass. Another early method involved building histograms of various color features and splitting the image into regions based on peaks in these histograms [25]. Unfortunately, the features to use for such a method are somewhat indeterminate and the edges between the segments were not well-

defined. A more recent approach is aimed at making cuts in the graph that minimize the coherence of parts being split [34][29]. While this approach is highly effective, finding the optimal cuts to make is NP-hard. One of the key results of this method, however, is that it considers segments as a whole when making cutting decisions, as opposed to previously mentioned methods that generally only consider the local area of the arc. The method used in this classification method was developed by Felzenszwalb and Huttenlocher [9]. The image is considered as a graph with each pixel being a node and potential arcs between neighboring pixels. The algorithm then selects arcs for inclusion based on the weight of the edge compared with the internal similarity within the two segments. There are two main benefits to this algorithm. Firstly, it also considers non-local information when segmenting, which provides much more coherent and effective segmentation. Secondly, it is relatively fast, running in NlogN time where N is the number of pixels in the algorithm. This allows for the possibility of applying the segmentation in real-time at video frame rates. This algorithm will be explained in further detail in section 3.

2.1.2 Objective Functions

The simplest measure of similarity is difference in color. For black and white images, this is simply the difference in the intensity. For color (RGB) images, we can use a measure of the difference in intensity in the three color planes. In general, this is not an optimal solution. The intensity of colors changes in a nonlinear manner under different lighting conditions. In addition, RGB does not capture the circular nature of hue. In practice, representations such as hue-saturation-value (HSV) or other normalized color spaces [10].

Improved color representations allow for more accurate comparisons of pixels and improved invariance to lighting conditions. Our implementation uses a simple intensity comparison for each of the three RGB color planes. Although this is not optimal, it works reasonably well. Improving the segmentation algorithm is of primary importance in future improvements to this algorithm.

The spatial proximity of pixels is also very important in grouping them. In some situations, it may be useful to allow gaps of a few pixels to allow for image noise or eliminate fine patterns. It may also be useful to add a measure of how far pixels are from the center of the segment to bias against long or large segments. The segmentation algorithm we used only allows continuous groups of pixels to be considered as one segment, but uses no measurement of distance from the center.

Texture is a measure of frequency in the edges of the image. Finding areas of similar texture can be as useful as finding areas of similar color. The simplest way to find texture is to smooth an edge map of the image, then look at the resulting intensities. Areas of low texture will have a very low average edge intensity, and vice versa. The segmentation algorithm we employ measures the minimum and maximum difference between pixels in the segment to measure texture. This is described in more detail in section 3.

2.2 Image Feature Detection Algorithms

In this context, image features will refer to any representation that can be used to repeatedly identify some point of interest in the image. The simplest features are simply patches taken directly from the image. One could simply simply cut out a section of a

car's headlight and look for that patch in all subsequent images, marking as a match all areas that are sufficiently close. The obvious difficulty is how close a section needs to be to be considered a match, and how that distance is evaluated. In addition, patches are unfortunately very poorly resistant to variations in color and changes in scale and rotation.

Modern feature detection algorithms fall into several classes. The most popular operate on the edges detected within an image. These include Harris corners [13] and SIFT [19]. SIFT features are used in this algorithm because they are reasonably fast to compute and provide good resistance against changes in scale and affine transformations. SIFT achieves scale invariance by sampling the image at a variety of scales, and affine invariance by representing the feature in such a way that reasonably small affine distortions have small effect on the representation.

Another interesting and potentially highly useful feature detection algorithm has been proposed by Mikolajcyzyk [23] which can selectively choose local edges while ignoring others. This could prove highly useful in finding a particular outline on a cluttered background, and further effort should be made to include these features in future versions of this algorithm.

Alternative feature detection algorithms also exist which operate on different parameters of the image, such as matching color and texture histograms over local areas. These were not considered due to the variety of colors present in cars and their unproven nature. Recently, features based on wavelets [22] and improved by Stollnitz, DeRose, and Salesin [30] have been used with good results. Wavelet features are faster to compute in many

circumstances and have been shown to be more useful in some applications [12]. Future experiments should investigate their value compared to SIFT features for our application. Jurie and Triggs [16] have presented results showing that features such as those created by SIFT and others do not perform as well for image classification as simple patches. The computational difficulty in finding informative patches is, unfortunately quite high. In addition to the number of possible features to be selected from, finding the patches in the image to be classified is also considerably more time consuming than SIFT or similar algorithms.

2.3 Feature Selection Algorithms

Feature selection is a large and highly researched area. There are many different approaches to finding the most useful features of a given problem, and many work better or worse depending on the data type.

Filtering

The simplest type of feature selection algorithm is filtering. Each feature is compared to the target variable using some sort of objective function (such as statistical association). Filtering is generally very fast because it requires a small number of computations (linear in time with the number of features) and is easy to implement. Features chosen by filtering may suffer from high redundancy (if many redundant features are highly correlated) and the algorithm will miss features that provide significant information in conjunction with other features.

Wrapping

In wrapping, features are chosen by how they effect the performance of the chosen classifier [17]. Essentially, the performance of the classifier is tested with different sets of features, which are chosen by a heuristic function. In backwards wrapping, features are incrementally removed if they have less than a thresholded penalty on performance. In forwards wrapping, features are incrementally added to the feature set. In general, it can be difficult to provide a good heuristic function that searches effectively in the feature space in a manner much more efficient than exhaustive search. In addition, it can be extremely costly to repeatedly train the classifier if the number of required features is high. In the case of our experiments, the size of the dataset is considerably to large for effective wrapping.

Clustering

In cases of many redundant features, it may be possible to use clustering methods (such as K-means discussed above) to accumulate similar features into super-features. The difficulty of this method is deciding on how to cluster features. Generally, it requires repetitive optimization of the clustering parameters. In some cases, it may not be apparent when features that carry similar but distinct information are clustered together.

Principal Component Analysis

Principal component analysis sequentially creates new features from the dataset by orthogonally rotating the data so that largest variance falls on an axis. The data from features that fall along that axis are then encompassed in the new feature. PCA is relatively fast and extremely effective but has some limitations. It may have problems with discrete data and discrete classifiers in that the transformation will create real-valued

features. In addition, it creates features in a greedy manner, which may not reflect the optimal feature set. PCA is a specific instance of a wider class of linear regression feature selection algorithms, which vary primarily in how they choose to select features. The basic measure, however, is always the variance associated with each feature [14].

Bayesian Network Induction

If each feature in the data is considered a node in a Bayesian network, then finding the connections between the target and the nodes around it will provide useful information about informative, uninformative, and redundant variables [4]. The structure of the network surrounding the target is induced by using statistical tests for independence for all variables. This method is somewhat slow, but valuable in that it produces much smaller feature sets due to its capacity to eliminate redundant features and detect informative variables that are not directly associated with the target. Unlike wrapping methods, it does not require repeated application of the classifier.

2.4 Classification Algorithms

Decision Trees (DT)

DTs sequentially split the data based on the features chosen by a heuristic function. Generally, the heuristic function attempts to minimize the variability in the resulting leaves using a function such as Information Gain:

$$I = -\sum_{i=1}^{N} p_i \log(p_i)$$

Where N is the number of classes and p_i is the proportion of instances which belong to class i. Each node may split on a different feature, which is useful in that groups of

features can be considered separately. Unfortunately, in choosing features greedily it may lose information that becomes split between many different nodes. The models created by DTs are fast to compute and easy to interpret, as they can be easily decomposed into a series of rules. This makes them popular in medical and financial applications where people are uncomfortable using "black box" machine learning algorithms.

K-Nearest-Neighbors (KNN)

In KNN classifiers, each instance is classified according to the K most similar instances in the training dataset. The most similar instances are calculated by a distance metric (euclidean, Manhattan, etc.). The decision among the neighbors may be by simple majority or more complicated mechanisms such as distance-weighting. KNN is very simple to implement and fast to execute. It performs poorly in instances where there are many uncorrelated variables because they tend to dominate the distance metric. Even in datasets with only relevant variables, it can be difficult to decide which variables are relevant in any one particular instance.

Neural Nets (NN)

NN classifiers are extremely flexible and accurate, but very difficult and time-consuming to train. Although many variations exist, the basic idea is described here. Create a number of nodes (perceptrons) that receive inputs from N data features. Create a series of hidden layers behind these inputs, consisting of k hidden nodes per layer, and one (or many in the multi-class case) output node(s) at the end. Each node in the hidden layers is connected to all of the nodes in the layers before and after it, and every connection has a weight. As inputs are fed in, every node generates an output using a function such as a

sigmoid, which is then used in downstream nodes along with the weight of that connection.

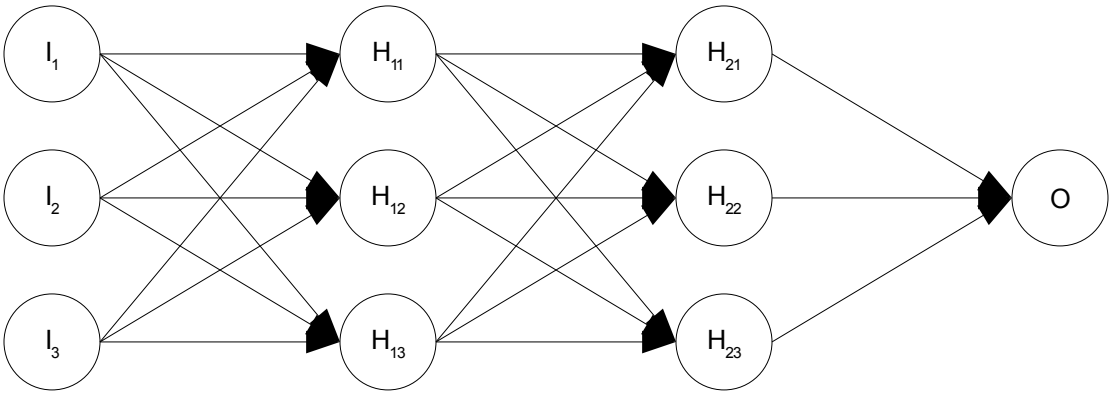

Illustration 1: An example neural net with 3 input nodes, 2 hidden layers of 3 nodes each, and 1 output node.

NNs are generally trained via an algorithm called Back-propagation. Essentially, the weights of each node are modified by:

$$w_n = w_{n-1} + \frac{\delta}{\delta w} L \sum_{n=1}^{k} (D_n - O_n)^2$$

Where w is the weight vector for the inputs of that node, n is the current iteration, L is the learning rate, D_n is the desired output of the node, and O_n is the actual output. This step is repeated for every instance in the training data over several iterations until the accuracy meets a threshold or ceases to change.

NNs are extremely flexible and have been shown to simulate with high accuracy many functions, but at the expense of large hidden layers. Training NNs is very costly and intractable for large numbers of inputs such as presented in this experiment. In addition, NNs are often met with resistance in research because they are very difficult to interpret.

Although their classification accuracy may be high, they are essentially a black box and may not provide the information necessary to better understand the phenomena they are applied to.

Boosting

Boosting consists of repeatedly training many versions of a "weak" classifier and combining them to form a stronger one. Each instance in the training set is given a weight dictating its importance in the classification problem. For each iteration, the weights are changed to reflect the accuracy of the classifier, with incorrectly classified instances getting higher weights so they are more highly considered in the next iteration. The most popular boosting algorithm, AdaBoost, was designed by Freund and Schapire [11], but many variants exist.

Support-Vector Machines (SVM)

SVM classifiers learn a hyperplane separating the classes. To add further discretion, the training data can be projected into arbitrarily higher dimensions using kernel functions [6]. SVMs have recently become popular because they are scalable to large sets of training data. Unlike DTs, SVMs consider all features at every step and therefore will not get stuck in local minima (at least within the restrictions of their kernel function). Unlike KNN, SVMs are also good at using different features to classify different instances. Like NNs, SVMs can be difficult to interpret (especially if a kernel function is used). We decided to use SVMs because the size of the data is very large, and we believe the classifiers will be fairly self-explanatory by which features are used in support vectors. Both SVMs and boosting have been used successfully in image classification systems,

and further investigation using a boosting type classifier is warranted in future research for this application.

2.5 Object Detection Algorithms

There is a tremendous amount of literature and numerous approaches to object detection. In this section, we will restrict our discussion to some of the more recent attempts with comparable methods.

Leung [18] used a combination of SIFT, SVMs, and Gentle AdaBoost to design a develop two car detection systems. The first was a general classifier for images as a whole, but did not pick out the parts of the image that contained cars and did not work well in urban settings. The second defined windows in the image and classified them as likely containing a car or not. This used hand-defined (not SIFT) patches, which were then selected by Gentle AdaBoost for inclusion in the classifier. This had significantly better success, though it only defined a rectangular region that contains a car and required many searches over various window scales and positions. This work was an extension of work by Papageorgiou and Poggio [27], which had attempted to solve a similar problem using Haar wavelets instead of SIFT, and in a much more constrained environment. This work was also extended to detect pedestrians by constraining the features to fit a model [24]. A similar approach was used by Agarwal and Roth [1] except that the model was learned instead of given.

Dorko and Schmid [7][8] have also designed systems for car classification by learning highly-discriminating features by a Gaussian-mixture model (GMM). Objects are localized by essentially drawing rectangles around areas of high feature density.

Our process differs from others in several respects. We learn important features using Bayesian network induction, which we feel provides a more complete and parsimonious set of features without the need to cluster (and potentially obfuscate) features. We do not constrain our classification with models of cars at all. A model may help increase confidence in some classifications and provide information about the pose of the car, but is also not as robust to variation and occlusion. Future work may try incorporating a simple model to help improve decision making on questionable segments. Our approach is considerably faster than the methods which find rectangles of interest because we limit our classification to image segments instead of an arbitrarily large number of possible rectangles. Lastly, because our detection system classifies image segments, it draws much more accurate edges around the edges of cars. This can be invaluable when paired with a motion tracking algorithm or stereo cameras by giving it very specific regions to track.

CHAPTER III

SEGMENTATION

The segmentation algorithm used in this experiment is a simple, graph-based method designed by Felzenszwalb and Huttenlocher [9]. We will use this section to describe the algorithm in a broad sense, more information can be found in the cited source. Segmentation is key to our approach for two reasons. Firstly, it allows us to group SIFT features together with high probability that they belong to the same object. This is critical to the speed of the algorithm because it creates a very small set of groupings for the features, as opposed to other approaches which used windows of vary size to search across the image. Secondly, it allows us to make distinct differentiations between regions of car and non-car.

3.1 Initialization

The function is initialized by creating a graph with a number of nodes equal to the number of pixels in the image. Potential connections are made between every pixel and its 8 neighbors, with the weight of the edge being equal to the absolute intensity difference between the pixels. As the images we use are color, the algorithm is run 3 times, once on each of the different colors planes of the image and the segments are joined as described below. Although this is not an optimal measure of similarity between

pixels, it works well in practice and is simple to compute. Future work should be pursued in using better measures of affinity with this algorithm, though the algorithm itself is independent of the measure of affinity used.

3.2 Comparison Function

The heart of the segmentation is the comparison function, which allows the algorithm to decide whether to join to components or not. In general, the ideal function would link components that are very *similar*, and separate components that are not; unfortunately, *similar* is difficult to define for all contexts. At a basic level, this comparison function seeks to link components that share similar qualities in spatial frequency. This is effective in separating areas of high frequency (such as grass, bricks) from areas of low frequency (asphalt) while maintaining the high frequency areas in a single component. As components are compared, the decision criteria is based on a comparison on the internal difference in each component to the difference between the two components.

For the following formulas, we define:

C– **C**omponent : A set of pixels

E – The set of all edges (e) in the graph

V – The set of all vertices (v) in the graph.

The internal difference within a component is defined to be the maximum weight of any edge on the minimum spanning tree (MST) of the component.

$$Internal\ Difference: Int(C) = \max_{e \in MST(C,E)} w(e) \tag{3.1}$$

The difference between components is defined to be the minimum weight of any edge that joins the two components.

$$Difference: Dif(C_1, C_2) = \min_{v_i \in C1, v_j \in C2, (v_i, v_j) \in E} w(v_i, v_j) \tag{3.2}$$

The minimum internal difference of two components is defined as the minimum of the internal difference of each component plus a threshold function on that component.

$$Min.\ Int.\ Difference: MInt = min(Int(C_1) + \tau(C1), Int(C2) + \tau(C2)) \tag{3.3}$$

Where the threshold function is defined as follows:

$$\begin{aligned} &Threshold: \tau(C) = k / |C| \\ &k: Constant\ Parameter \\ &|C|: Cardinality\ of\ C \end{aligned} \tag{3.4}$$

The decision function to join components of not is:

$$Decision: D(C_1, C_2) = \{ Dif(C_1, C_2) < MInt(C_1, C_2) \} \tag{3.5}$$

Logically, if the difference between components is close (within □(C_i)) to the internal difference of C_i, then the components are reasonably similar. The constant k can be used to bias the segmenting algorithm towards larger or smaller component sizes, with larger and smaller k's respectively. In our experiments, we used a k of 100, which was determined by experiment.

3.3 Algorithm

The algorithm has two steps. The first creates segments using Kruskal's Minimum Spanning Tree algorithm with (3.2) as edge weights between components until no edges exist that satisfy (3.5). The second step post-processes the segments, joining any components smaller than a constant MCS (minimum component size) to their most closely matching neighbor. This step helps to eliminate many small segments and joins components that do not fit any large segment well. The MCS used in our experiments was 600, and was determined by experimentation. Unfortunately, it is difficult to empirically judge the quality of segmentation algorithms, particularly in complex images such as those used by our experiment. For this reason, both constants k and MCS were chosen by human judgment as opposed to empirical optimization.

CHAPTER IV

SCALE-INVARIANT FEATURE TRANSFORMATION (SIFT)

This section will present a basic overview of the Scale Invariant Feature Transformation (SIFT) algorithm. It is intended to familiarize the reader with the way features are selected and therefore the quality of the features, but not as a guide to implementing or improving SIFT. To this end and for the sake of brevity, many implementation details and the rationale for design choices are intentionally omitted. We present the process of the SIFT algorithm as it was presented by Lowe in the following four sections: extrema detection, keypoint localization, orientation assignment, and keypoint descriptor. SIFT was originally presented by David Lowe [19]. For a more detailed reference, see [21]. The software implementation used in experiments presented here was written by Andrea Vedaldi [31].

The purpose of the SIFT algorithm is to find features in an image which are recognizable under varying scales, rotations, and perspective changes. It has been used very successfully in finding the same object in different pictures [20]. More recently, it has been used by many researchers as a feature detection algorithm for object class detection, such as those mentioned in section 2.5.

4.1 Extrema Detection

The first step in creating SIFT features is detecting local extrema in scale space, which correspond to zero crossings of the second derivative of scaled versions of the image. These are the points at which edges and corners are the most prominent in a continuum of scales. For a full discussion of this phenomena, see Witkin [32].

To find these extrema, the image I is first convolved with a series of Gaussian kernels with varying standard deviations □ (referred to as the scale).

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) : for\, \sigma \in S \tag{4.1}$$

to make intermediate images L. S is generally chosen to be a series of □ separated by a factor k of approximately $2^{1/|S|}$. Images which have adjacent □ are then subtracted to create a final image D, which contains the derivative of scale-space information.

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) \tag{4.2}$$

The original image is then down-sampled by a factor of two, and the process is repeated for set number of iterations or octaves. This effectively allows the algorithm to search the image at a wide range of scales.

To locate the extrema, each pixel in every image is compared to its eight neighbors, and the nine neighbors in the images one □ above and below it. If that pixel has a value smaller or larger than all of its neighbors, it is considered an extrema and marked for further investigation. For experiments in this paper, we used 8 octaves, $\sigma_0 = 1.6$ and a $|S| = 6$. This means we end up with six L type images and 5 D type images. To find local

extrema, we can only use D images that have a neighbor, so we eventually take extrema from the 3 center D type images.

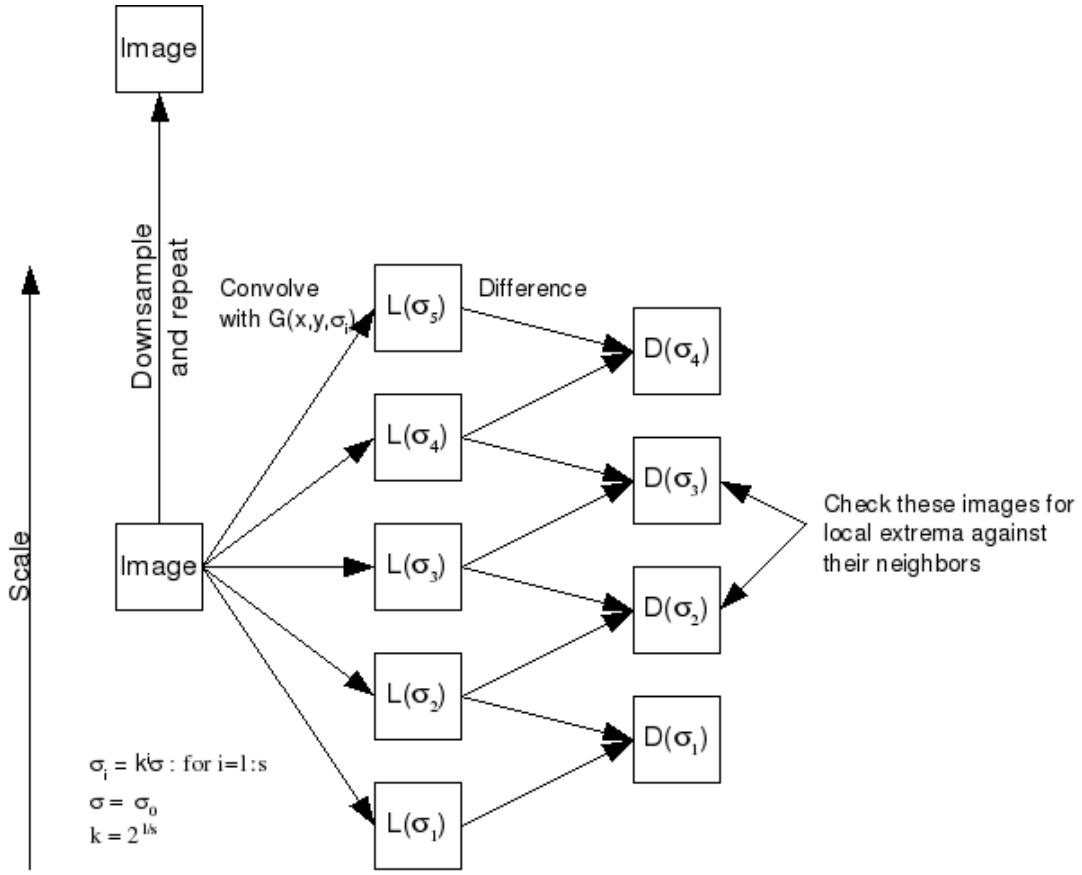

Illustration 2: Detection of SIFT local extrema.

4.2 Keypoint Localization

The purpose of the localization step is to further refine the center of the keypoint and assess its stability. To refine the location of the keypoint, the offset from the sampled position is estimated by solving the equation:

$$\Delta = \frac{-\delta^2 D(x)^{-1}}{\delta x^2} \frac{\delta D(x)}{\delta x} \tag{4.3}$$

where x = (x,y,□)^T.

In addition, the following formula can be used as a metric for defining points with poor contrast. In effect, it shows points where the extrema is not significantly different than it's surrounding points.

$$D_{x\delta} = D(x) + .5 * \frac{\delta D(x)^T}{\delta x}(x + \Delta) \tag{4.4}$$

If the value of metric is sufficiently low, the point is discarded. In our experiments, we used a threshold of $D_{x□} > .0066$. Points below this threshold were discarded.

Keypoints that occur along a straight edge are not unique in that they may occur at multiple places along the same edge. To eliminate such points, SIFT estimates the primary principal curvatures of the point using the Hessian matrix H, which is estimated from neighboring pixels in D. From H, it is easy to estimate the ratio of principal from the following equation:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{4.5}$$

Where r is the ratio of the principal curvatures. In our experiments, we used an r of 10. Points that did not meet the condition of (4.5) were discarded.

4.3 Orientation Assignment

Assigning an orientation to the keypoint allows recognition of that keypoint again even if it has been rotated in the image. In essence, the descriptor (section 4.4) is assigned relative to the orientation of the keypoint.

To assign orientation to the keypoint, each pixel in the image L has the magnitude and the orientation computed:

$$magnitude: m(x,y)=\sqrt{(L(x+1,y)+L(x-1,y))^2+(L(x,y+1)+L(x,y-1))^2} \quad (4.6)$$

$$orientation: \theta(x,y)=\arctan\left(\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}\right) \quad (4.7)$$

For each, keypoint, a histogram is formed from the pixels surrounding it. The histogram has 36 boxes that each occupy 10□ of orientation for pixels. The pixels in the region around the keypoint are added to the histogram, each with each magnitude weighted by a circular Gaussian function centered on the keypoint with standard deviation of 1.5 times the □ of L. The orientation of the keypoint is assigned to be the peak of this histogram. If multiple peaks within 80% of the peak are found, multiple keypoints are created with orientations set to the lesser peaks.

4.4 Keypoint Descriptor

The descriptors for SIFT keypoints are also based on the histograms of local gradients. The area around each keypoint is broken into square sectors, and the orientation and gradients of each pixel in those sectors are used to construct a histogram. In our experiments, a 64x64 pixel area is selected around the center of the keypoint to create the descriptor (oriented according to the keypoint orientation.) This is further subdivided into 16 16x16 pixel sectors. The magnitude of each pixel is weighted by a Gaussian centered on the keypoint with standard deviation of 32, to minimize the effect of far off and possibly noisy pixels. The pixels of each sector are binned into a histogram with 8 bins representing 30□ of orientation. The weights for each bin in each histogram are combined to form a vector and normalized to be between 0 and 256. 16 sectors with 8 bins leads to a descriptor of 128 bytes. The final feature vector includes the x and y locations, the scale, the orientation, and the descriptor. The scale and location are valuable when trying to assign keypoints to an existing model, but are not used in this experiment.

CHAPTER V

HITON-PC

HITON-PC performs feature selection by assuming the data can be modeled as a Bayesian network. Essentially, the algorithm finds all of the variables which are statistically linked to the target. Redundant variables are eliminated by testing for independence conditioned on subsets of other linked variables. The HITON-PC algorithm is due to Aliferis, Tsamardinos, and Statnikov [2]. HITON-PC is part of a larger family of algorithms called GLL, which are widely discussed in [4]. The implementation of HITON-PC we used is part of the Causal Explorer software package, provided by the Discovery Lab at Vanderbilt University [3].

For the following discussion, we will define the following terms:

Network – A directed acyclic graph consisting of vertices (v_i □ V) and edges (e_i □ E). An edge from v_1 to v_2 means that the variable modeled by v_1 is a direct cause of the variable modeled by v_2.

Parent – A node that has a direct causal effect on the target T, i.e. there is a connection from the parent node to T.

Child – A node that is a direct effect of T.

Spouse- A node (other than T) that is the parent of a child of T.

I(X,Y) – Independence: Vertices X and Y are statistically independent.

I(X,Y|S) – Conditional independence: Vertices X and Y are independent given a set of vertices S.

In order to justify the correctness of the algorithm, the following assumptions are made:

Causal sufficiency – There are no hidden variables which cause unmodeled links between variables.

Faithfulness – A network is faithful if all data generated from the network shows the same conditional independence relationships as the network.

Numerical sufficiency – The number of samples available from the network are sufficient to test all independence conditions entailed by the network with high accuracy.

5.1 Markov Blanket and D-Separation

The Markov blanket of a variable is defined to be the minimum set of all variables that when conditioned upon, render the target independent of all other variables. More formally:

$$\min_{|MB(T)|} MB(T) = Z | I(T, X|Z) \forall X \in V \setminus Z \tag{5.1}$$

Qualitatively, the Markov blanket of T consists of all parents, children, and spouses of T. Intuitively, if we can find the Markov blanket of T, it contains all possible information about T, and would be the minimum set of data necessary to feed to an optimal classifier.

A simpler task than finding the Markov blanket is finding the parent-child (PC) set of T. This is simply the set of all parents and children of T. With HITON, the Markov blanket is found by finding the PC set of T, then finding the PC set of the PC set of T and combining the variables. In our experiments, we only used the PC set of T instead of the full Markov blanket, but as will be presented the performance was still quite good.

The concept of D-separation describes the conditions necessary for one variable to be rendered conditionally independent of another [28]. The following definitions apply to the discussion of d-separation:

> **collider** – A node is a collider if it is on path P and it has two input edges on path P.
>
> **Open Path** – A path is open if there is at least one collider which is not instantiated, or one non-collider which is instantiated. If all paths between nodes X and Y are open, then they are conditionally independent.

We will use the following network for illustration of a few examples to clarify d-separation:

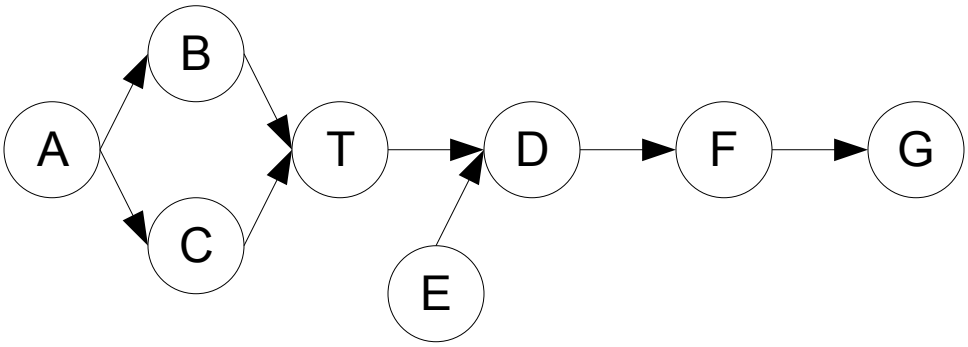

Illustration 3: Sample Bayesian network.

1. I(T,F | D)

If the node D is instantiated (part of the conditioning set), all information transmitted between T and F is blocked. The path T-D-F is open; F contains no information about T (and vice-versa) that is not incorporated in the value of D.

2. ~I(T,F | □)

The node D is not instantiated, nor is it a collider. Therefore, the path T-D-F is closed, and T and F are not d-separated or independent.

3. ~I(A,T | B)

A and T are not d-separated because there still exists a path, A-C-T with no colliders or instantiated nodes.

4. I(A,T | B,C)

By instantiating B and C, all paths between A and T are closed.

5. I(T,E)

The only path between T and E is T-D-E, which is open because the variable D is a collider and is not instantiated.

6. ~I(T,E | D)

Likewise, T and E are not d-separated if D is instantiated. E is a spouse of T, and can provide information about T if the value of D is known.

Difficulties can arise due to size of the conditioning set necessary to render to nodes independent. Consider the following network.

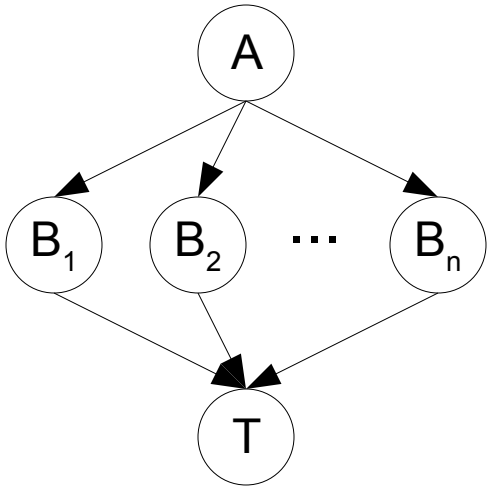

Illustration 4: Bayesian network requiring large conditioning set.

To render I(A,T), the conditioning set must be of size n. In practice, however, statistical tests of independence break down as the size of the conditioning set grows. Experiments done on our data set showed that a conditioning set of size 9 could render any other variable independent of the target, regardless of the true network. For this reason, false positives may occur because the size of the necessary conditioning set is statistically unfeasible. To combat this problem, we limit our tests of independence to conditioning sets of size 3. This biases the error towards including potentially independent features as opposed to excluding useful features.

5.2 Algorithm

In our experiments, we used a version of HITON that only finds the PC set of T, named HITON-PC. The following pseudocode describes the algorithm:

```
HITON-PC(DATA, TARGET, CSIZE)
        TPC = {}  //Initialize the tentative PC set to empty
        //Order all variables in data by their association with T.          //Discard
unassociated variables
        V = order_by_association(DATA, TARGET)
        For v in V:
                Add v to TPC
                for c in TPC:
                        //Condition on all subsets of TPC up to size CSIZE
                        if I(T, c | Z) for any Z □ TPC\c
                                discard c
Return TPC
```

Basically, HITON-PC adds items to the TPC set only if they are still dependent on the target given any subset of the TPC set. To avoid the problem of false exclusions described in the previous section, it only conditions on sets up to size CSIZE. In our experiments, we used CSIZE = 3. As new variables are added to the TPC set, it also checks to see if previously added variables can be removed by conditioning on the new variable or a combination of the new variable and other TPC variables.

In our experiments, we used a the G^2 statistic as our test of independence and association. A variable was considered independent if the association p-value was less than 10^{-5}.

CHAPTER VI

SUPPORT-VECTOR MACHINES (SVM)

Support-Vector Machines are a form of classifier that learns a hyperplane separating the training data. If the data does not lend itself to linear separation, it can be projected into higher dimensions with the use of a kernel function. SVMs are relatively fast to train and highly expressive learning machines.

6.1 Linearly-Separable Case

Consider that we have a set of data, composed of N tuples of a class, $y_i \in \{-1, 1\}$, and a data vector, $x_i \in \{R^n\}$. If this data were linearly-separable, i.e. we could draw a hyperplane that left all tuples of each class on one side or the other, then there would exist a hyperplane H:

$$H = \bar{w} \cdot \bar{x} + \bar{b} \tag{6.1}$$

such that:

$$y_i(\bar{w} \cdot \bar{x}_i + \bar{b}) > 0 \quad for\ i = 1:N \tag{6.2}$$

The best hyperplane (as there are infinitely many that do this in the separable case) is the one that creates the maximum distance between points on either side (called the margin). Note that we can rescale w and b such that for the closest points on either side:

$$y_i(\bar{w} \cdot \bar{x}_i + \bar{b}) = 1 \tag{6.3}$$

We can measure the margin, therefore, as the distance between those points measured perpendicular to the plane H. Alternatively, consider two hyperplanes H1 and H2 that are parallel to H, and pass through the points mentioned previously; the margin is the distance between those hyperplanes.

$$m = \frac{\bar{w}}{\|\bar{w}\|} \cdot (\bar{x}_1 - \bar{x}_2) = \frac{2}{\|\bar{w}\|} \tag{6.4}$$

The margin is what we seek to maximize to find the optimal hyperplane, therefore we must minimize ||w|| subject to the following constraints:

$$y_i(\bar{w} \cdot \bar{x}_i + \bar{b}) \geq 1 \quad for\ i = 1 : N \tag{6.5}$$

Note minimizing ||w|| is identical to minimizing $(1/2)\|w\|^2$, however using the later changes the problem to a quadratic minimization problem which is much easier to solve. To perform this operation, we will reorganize the problem into a Lagrangian formulation.

$$L = \frac{1}{2}\|\bar{w}\|^2 - \sum_{i=1}^{N} \alpha_i y_i (\bar{x}_i \cdot \bar{w} + \bar{b}) + \sum_{i=1}^{N} \alpha_i \tag{6.6}$$

$$\alpha_i \geq 0 \quad for\ i = 1 : N \tag{6.7}$$

Where α_i represents the Lagrange multipliers for the constraints of (6.5). The minimum will occur when the following conditions are met:

$$\frac{\delta L}{\delta \bar{w}} = 0 = \bar{w} - \sum_{i=1}^{N} \alpha_i y_i \bar{x}_i \tag{6.8}$$

$$\frac{\delta L}{\delta \bar{b}} = 0 = \sum_{i=1}^{N} \alpha_i y_i \tag{6.9}$$

$$\alpha_i \geq 0 \quad for\ i = 1:N \tag{6.10}$$

Substituting these into (6.6), we find our task is:

$$\frac{max}{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,\, j=1}^{N} \alpha_i alpha_j y_i y_j \bar{x}_i \cdot \bar{x}_j \tag{6.11}$$

6.2 Linearly Non-Separable

In most practical cases, we find that the data is not linearly separable. There are errors due to noise, unknown information, and other unmodeled behavior. In order to allow for classification errors, Cortes and Vapnick [6] proposed the introduction of slack variables into (6.2), which now becomes:

$$y_i(\bar{w} \cdot \bar{x}_i + \bar{b}) \geq 1 + \xi_i \quad for\ i = 1:N \tag{6.12}$$

$$\xi_i \geq 0 \quad for\ i = 1:N \tag{6.13}$$

This leads to a change in equation (6.10), to:

$$C \geq \alpha_i \geq 0 \tag{6.14}$$

C is the cost, which is chosen as a parameter for the SVM. Higher costs force a more perfect solution to the hyperplane. The linearly-separable case above is equivalent to a SVM with C = □. The optimization problem becomes:

$$L = \frac{1}{2}\|\bar{w}\|^2 - \sum_{i=1}^{N} \alpha_i y_i(\bar{x}_i \cdot \bar{w} + \bar{b}) + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i \tag{6.15}$$

The □i represent the Lagrange multipliers to account for (6.13). Fortunately, the addition of the cost variable does not change the base problem to be solved (6.11). The solution to this is found at the point where (6.8) and (6.9) hold, as well as where the gradient with respect to □i is 0. The full set of constraints are listed below:

$$
\begin{aligned}
&y_i(\bar{x}_i \cdot \bar{w} + b) - 1 + \xi_i \geq 0 \\
&\xi_i \geq 0 \\
&\alpha_i \geq 0 \\
&\mu_i \geq 0 \\
&\alpha_i(y_i(\bar{x}_i \cdot \bar{w} + b) - 1 + \xi_i) = 0 \\
&\mu_i \xi_i = 0
\end{aligned}
\tag{6.16}
$$

6.3 Kernel Methods

In some cases, it may be impossible to create a good separation of the data with a hyperplane. Instead, we project the data into a higher dimensional space, and learn a hyperplane that separates it there. This idea is made tractable by a key observation about (6.11). The only place it is necessary to evaluate xi during the optimization is when dotting it with x_j. This operation can be replaced with a kernel function $k(x_i, x_j)$, which performs the transformation. This allows the SVM to consider mappings to potentially infinite feature spaces without having to optimize potentially infinite □'s.

Many kernel algorithms exist, but the most commonly used are:

$$polynomial: \quad k(\bar{x}_i, \bar{x}_j) = (\gamma \bar{x}_i^T \bar{x}_j + u)^d$$
$$rbf: \quad k(\bar{x}_i, \bar{x}_j) = e^{-\gamma |\bar{x}_i - \bar{x}_j|^2}$$
$$sigmoid: \quad k(\bar{x}_i, \bar{x}_j) = \tanh(\gamma \bar{x}_i^T \bar{x}_j + u)$$

Table 1: Common SVM kernel functions.

In our experiments, we tested all three different types of kernel to choose the one that worked best with the data.

6.4 Unbalanced Data

In some cases, it may be useful to attach different C parameters to data of different classes. The training data may be unbalanced in that one class occurs much more often than the other, or the system penalty to misclassifying one class may be very high (i.e. false positives/negatives). To account for these problems, Osuna, Freund, and Girosi [26] proposed a method of modifying the base SVM equation to allow for different costs. The primal problem becomes:

$$L = \frac{1}{2}\|\bar{w}\|^2 - \sum_{i=1}^{N} \alpha_i y_i (\bar{x}_i \cdot \bar{w} + \bar{b}) + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i + C_{+1} \sum_{y_i=1} \xi_i + C_{-1} \sum_{y_i=-1} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i \quad (6.17)$$

Where C_{+1} and C_{-1} represent the different costs associated with each class. In our experiments, this will be represented by parameter v in the following relationship:

$$C = C_{-1} = \frac{C_{+1}}{v} \quad (6.18)$$

6.5 Other Features

In our experiments, we use the LIBSVM implementation provided by Chang and Lin [5], which includes several other features worth noting. To improve performance of the optimization, shrinking and caching as described by Joachims [15] are implemented. To better quantify the output of the SVM, probability estimates are implemented as described by Wu, Lin, and Weng [33]. This is particularly useful in calculating the performance of the classifier and generating the images presented in section 8.

CHAPTER VII

ARCHITECTURE

7.1 Training

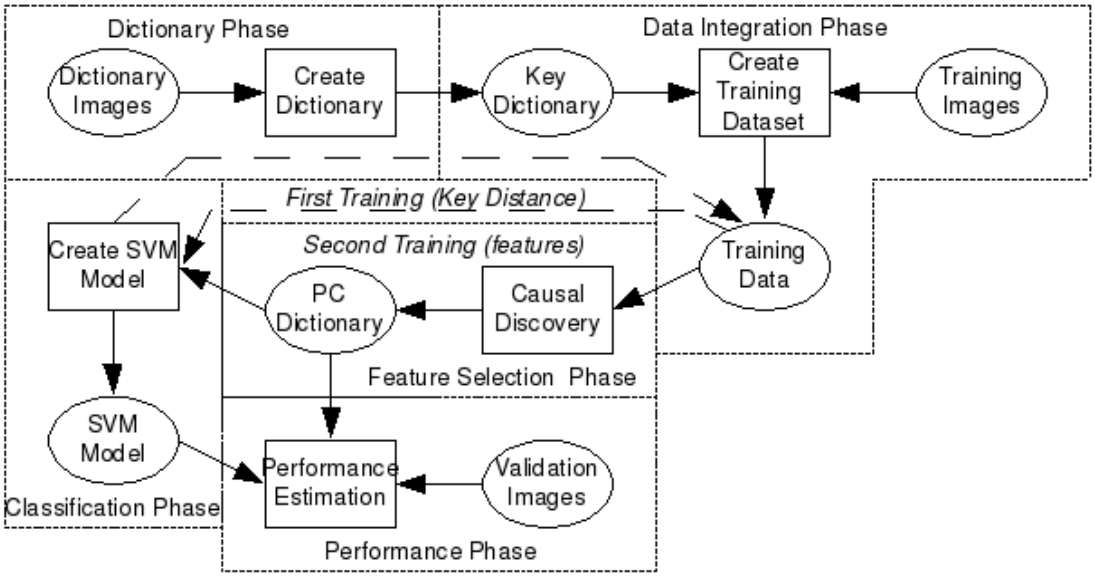

Illustration 5: Overview of training.

The training phase of the algorithm is divided into the following five phases: dictionary, data integration, causal discovery, classification, and performance. The following sections describe each phase in detail, as well as the data grouped with them.

Note that training takes place in two steps. During the first step, a variety of distance thresholds are used during the data integration phase, and evaluated in the classification

phase. This value is returned to the data integration phase, so the causal discovery phase must only be performed once as it is very time consuming.

7.1.1 Dictionary Phase

Illustration 6: Creating the dictionary.

The purpose of the dictionary phase is to create a data dictionary of SIFT keys. This dictionary will provide a reference for all keys found in subsequent images. The input to this step is a set of images with the regions containing cars annotated by hand. To create the dictionary, SIFT is run on every image in the dictionary set. Each key that is centered in an area marked car is added to the dictionary of keys. The dictionary used in experiments for this paper contains 39,257 keys drawn from 43 images.

No attempt was made to cluster or remove similar keys. Primarily, it is unclear that similar keys contain similar information and equally unclear how similarity could be measured without a fully probabilistic treatment, as is performed in the causal discovery phase. Ideally, by using all keys from the dictionary images and assessing their value with a causal discovery step, the maximum information can be extracted.

Keys were only selected from regions that contained cars to keep the size of the dictionary within a tractable size. Although more keys and a wider variety of keys would theoretically improve the quality of the dictionary, a tradeoff had to be made between the

number of keys and the number of images (there were a total of 442,441 keys in the 43 images.) It was determined that this was an appropriate compromise and would focus the algorithm on finding keys that were particularly pertinent to positively identifying cars, so that additional dictionaries could be created later to focus on other objects.

7.1.2 Data Integration Phase

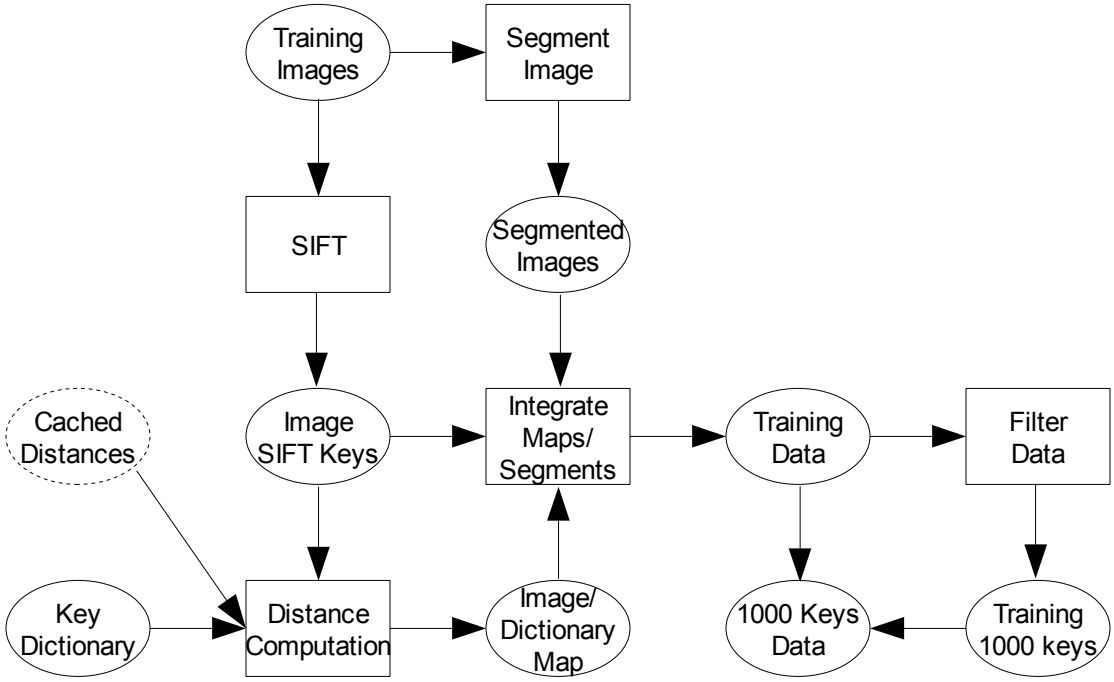

Illustration 7: Integrating the detected SIFT keys with the segmented images

The purpose of the data integration phase is to create a dataset from the training images suitable for use in machine learning applications. The entire process outlined in the above drawing is performed sequentially for each image. The input to this step is a set of 209 training images with the regions containing cars annotated by hand, the key dictionary

from the first phase, and optional cached distance data. First, the image is processed by the SIFT algorithm. The distance (euclidean) from each image key to the dictionary keys is computed and a match is marked if the value is below a threshold (DIST). Each image key may be within the threshold of several dictionary keys so each match is marked. From this data, a map relating image keys to dictionary keys is created and cached off-line in a file. On subsequent runs (to tune parameters related to the segmentation algorithm) this data can be retrieved without the costly process of computing the distance between 5k-10k image keys and 39k dictionary keys. The distance calculation is by far the most time consuming part of this process and would benefit greatly from a parallelized implementation with a graphics card interface such as CUDA.

Second, we segment the image and integrate data between the keys and the segments. Two maps are created: the first relating each segment to the image keys found inside of it, the second relating each segment to the keys in adjacent segments. For each segment, a row is created in the training data file. These rows will be used to perform feature selection, and a subset of each row will be used in training the classifier model. Each segment is of class 1 if 50% of the pixels fall into a region marked car, and -1 if they do not. Each row then contains 39,257 values in range [0,1] indicating if a key which matches the respective dictionary key is present in the segment. This is followed by an additional 39,257 values indicating if the respective dictionary feature is present in any adjacent segment. This creates a row for each segment containing one class value and 78,514 data values. In total, there were 44,450 rows of data from the 209 images. Note that this value is dependent on the parameters we chose for the segmentation algorithm.

Due to the large size of the data, it was necessary to filter the number of features we would attempt to perform later steps with. To do this, we performed a chi-squared correlation between the target and each of the 78,514 features in the training set, and selected the 1000 most highly correlated. These features were paired with their dual in either the segment space (if the feature was from adjacent segments) or the adjacent space (if the feature was within the segment), leading to a data set with 2000 features.

7.1.3 Feature Selection Phase

Illustration 8: Selecting features.

In the feature selection phase, the most important keys are selected for consideration and inclusion in the model. Ideally, these features will contain most if not all of the information pertinent to classifying segments as car or non-car. To find these pertinent features, we use the GLL algorithm HITON-PC. There are several caveats to this implementation. First, we make an assumption that the data conforms to the requirements of the GLL family algorithms, namely causal sufficiency, faithfulness, and numerical sufficiency as defined in section 5. Second, HITON-PC will find only the parent-child set of the target variable. This is a subset of the information contained in the full data set, and only by finding the Markov Blanket of the target variable would we capture all of the possible information. Unfortunately, finding the Markov Blanket requires repeated

applications of the HITON-PC algorithm, and such computations were not tractable given our available computational resources and time. In addition, in many applications of the GLL algorithms it has been observed that the PC set contains nearly all of the information, particularly when the PC is large. Third, HITON-PC may often choose a superset of the true PC because it only uses a fixed size conditioning set. In theory, it would be optimal to test each member of the PC for independence from the target conditioned upon the entire rest of the PC set, but in practice such tests are statistically impossible.

On this data set, we ran the HITON-PC algorithm with the G^2 statistic, a dependence threshold of 10^-5, and conditioning set size of 3. G^2 is the standard statistic used for discrete data. The threshold was determined by testing as a good tradeoff between PC size and computation time. The conditioning set of size 3 was chosen as a standard conditioning set size and small enough to prevent erroneously declaring variables independent but large enough to remove many d-separated variables. 176 of the 1000 keys were found to be significant with these parameters.

Note that during the first training step, this phase is skipped.

7.1.4 Classification Phase

This phase is performed twice with different goals. With the first training step, we seek to optimize the distance parameter used in assigning matches during the data integration phase. With the second, we seek to find the best possible SVM parameters to use for our final classifier. To do this, we train SVMs using a variety of parameters and average the results with 10-fold cross-validation. The set of parameters that gives the best average-

case performance for a single instantiation of SVM parameters is selected as optimal. To perform 10-fold validation, we assigned a random order to the the positive (6,050) and negative (38,400) examples in the training data. We then split each group into 10 equal parts, and combined the respective parts into blocks. The blocks were combined to form 10 training sets of 9 blocks, and 10 testing sets with the block left out of the training set. The purpose of this was to maintain the relative frequency (stratification) of positive and negative examples in each evaluation set. Each set of SVM parameters was run against each of these training sets and performance was evaluated on the respective testing sets. Performance is evaluated by area under the Receiver Operating Characteristics curve (AUC).

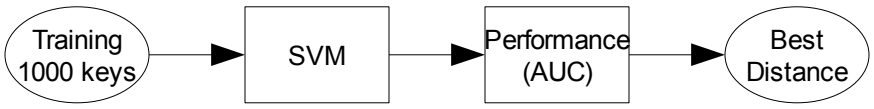

Illustration 9: Performance estimation to optimize distance parameter (DIST).

The input training data to the SVMs for the first training phase is the 2000 feature training data described at the end of the data integration phase. The distance threshold (DIST) was evaluated in multiples of 50 from 300 – 450. For the SVM, we used a polynomial kernel of varying degree (d) and cost weighting parameter (v). The cost weighting parameter was to account for the uneven distribution of the positive examples in the training set (approximately 9 negative examples for every positive example.

The data created with a DIST of 350 was found to have the best performance. Full results of this phase are presented in Appendix A. Note that only trials that completed within 24 hours were considered.

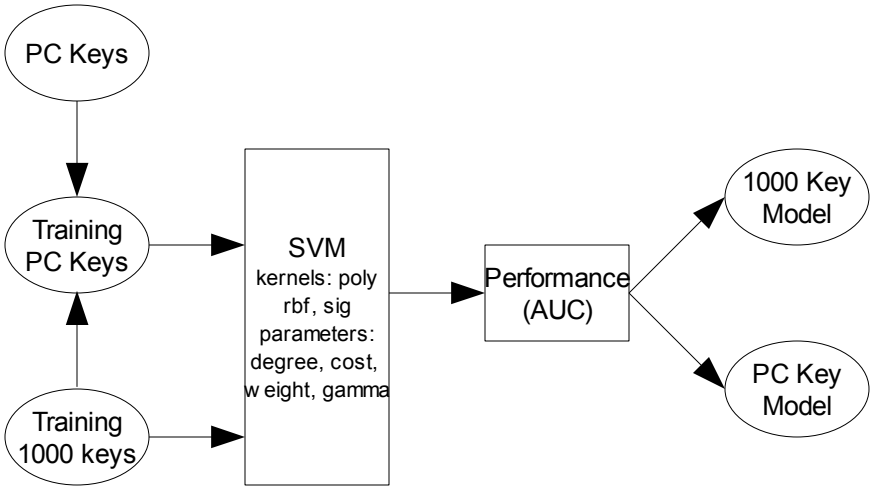

Illustration 10: Optimizing SVM parameters.

In the second phase of training, the training data generated with DIST= 350 was trained using different types of kernel functions (polynomial, radial-basis function (RBF), and sigmoid) and an additional parameter cost (C), as well as the previous parameters. The same training sets were used as in the previous training phase. In addition, "PC" subsets of each training and testing set were created by selecting the features from the PC dictionary discovered in the feature selection phase (7.1.3). The performance of the 1000 feature and PC sets is compared briefly below to show the utility and effectiveness of the HITON-PC algorithm. Full results of this phase are presented in Appendix B. Only trials that finished within 48 hours were considered.

The results of the best 1000 models for each dataset and their complimentary model in the other are presented below.

1000	RBF	0.9010	C : 10	v : 2	G : .000024
PC	RBF	0.8968	C : 10	v : 2	G : .000024
PC	Poly	0.8987	C : 10	v : 2	d : 1
1000	Poly	0.9004	C : 10	v : 2	d : 1

Table 2: Best performing models for 1000 key dataset and PC key dataset

In general, the performance difference between the 1000 key dataset and the PC dataset was negligible. These parameters were used to train SVM models on the entire training dataset, which were then used in the performance estimation phase.

7.1.5 Performance Estimation

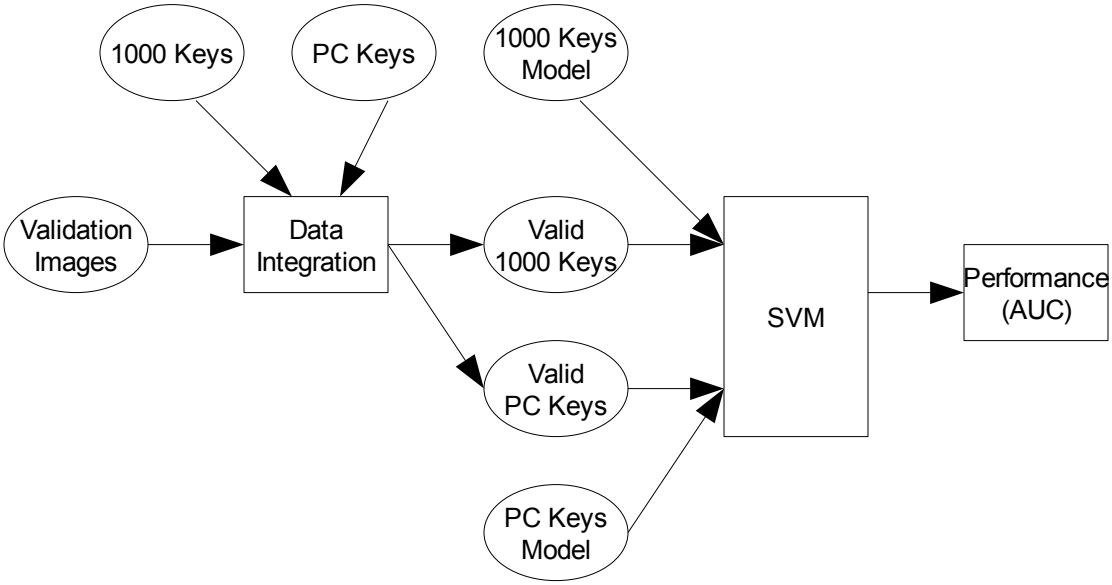

Illustration 11: Final performance estimation.

For the performance estimation phase, the model created in the classification phase is evaluated against an independent set of 55 validation images. Each image has the car

regions hand-annotated to use in evaluating performance. Each image is processed as in the data-integration phase, with the exception that the instead of the full dictionary of 39K features, only the 1000 most highly correlated features (as selected in the data-integration phase) are used. This speeds creation of the classification files considerably as each key must be compared to a much smaller dictionary. A subset of this data is also created selecting only the features from the PC dictionary discovered in the feature selection phase. Each of these data sets is evaluated using the best models from the classification phase. Performance is evaluated by AUC. The results of this step are presented in section 8.

7.2 Practice

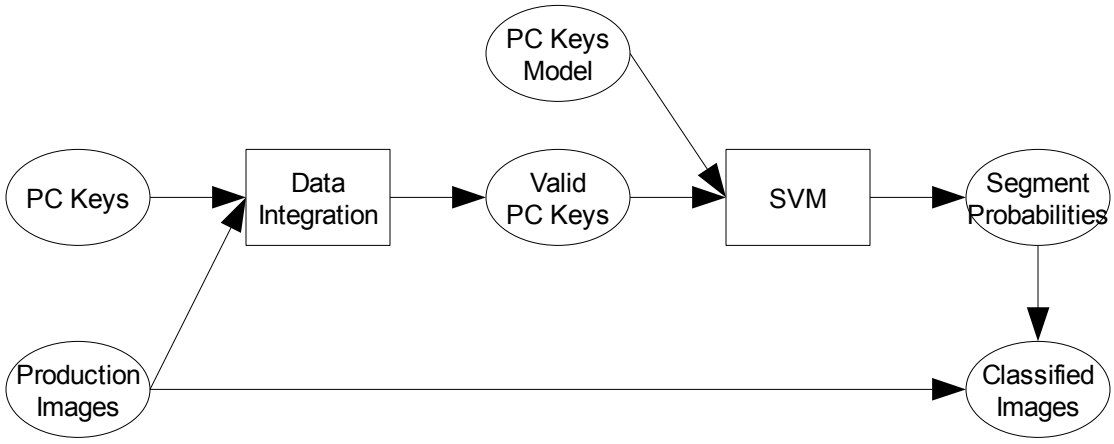

Illustration 12: Overview of detection system as used in practice.

In the production system, the events are much simpler. Each image is processed through the data-integration step, but only the keys from the PC dictionary are searched. The PC dictionary is much, much smaller than the original dictionary (176 vs. ~39K), which

speeds the process considerably. From each image a dataset is created and classified by the best PC model from the performance estimation phase of training. The results of the classification are then applied to the image, with each segment receiving a probability of belonging to a car. The threshold for calculation is chosen by the user to fit some specific point on the ROC curve to correspond to the desired levels specificity vs. sensitivity.

CHAPTER VIII

RESULTS

Two results are significant: our classifier provides significant predictive power, and the difference between the classifier using the 1000 most predictive keys and the PC subset of those keys was negligible. In addition, the processing time for the PC prediction model is faster by approximately one order of magnitude because of the smaller number of keys and simpler SVM mode. For this reason, we will only present detailed results for the PC key dataset.

PC	0.8776

Table 3: Classifier Performance

8.1 PC Key Results

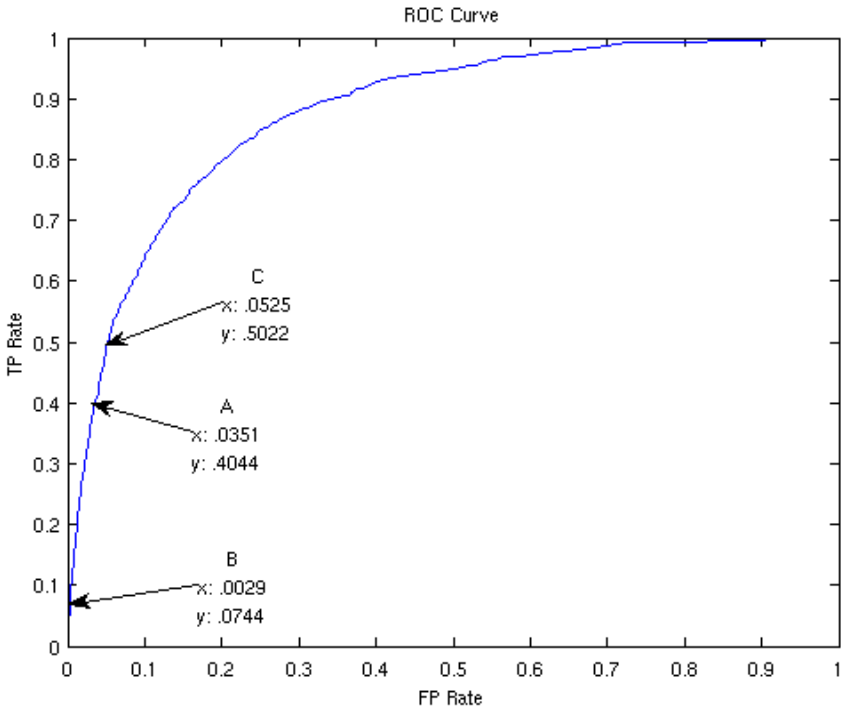

Illustration 13: ROC curve for PC key validation set

The PC key set provided performance of .8776 AUC which is generally considered a fairly good classifier. To interpret the above graph, read that at point A, the classifier correctly categorizes 40.44% of car segments and incorrectly categorizes 3.51% of non-car segments. Unfortunately, due to the heavy disparity in frequency between car and non-car segments, this still leads to a large number of incorrectly classified segments.

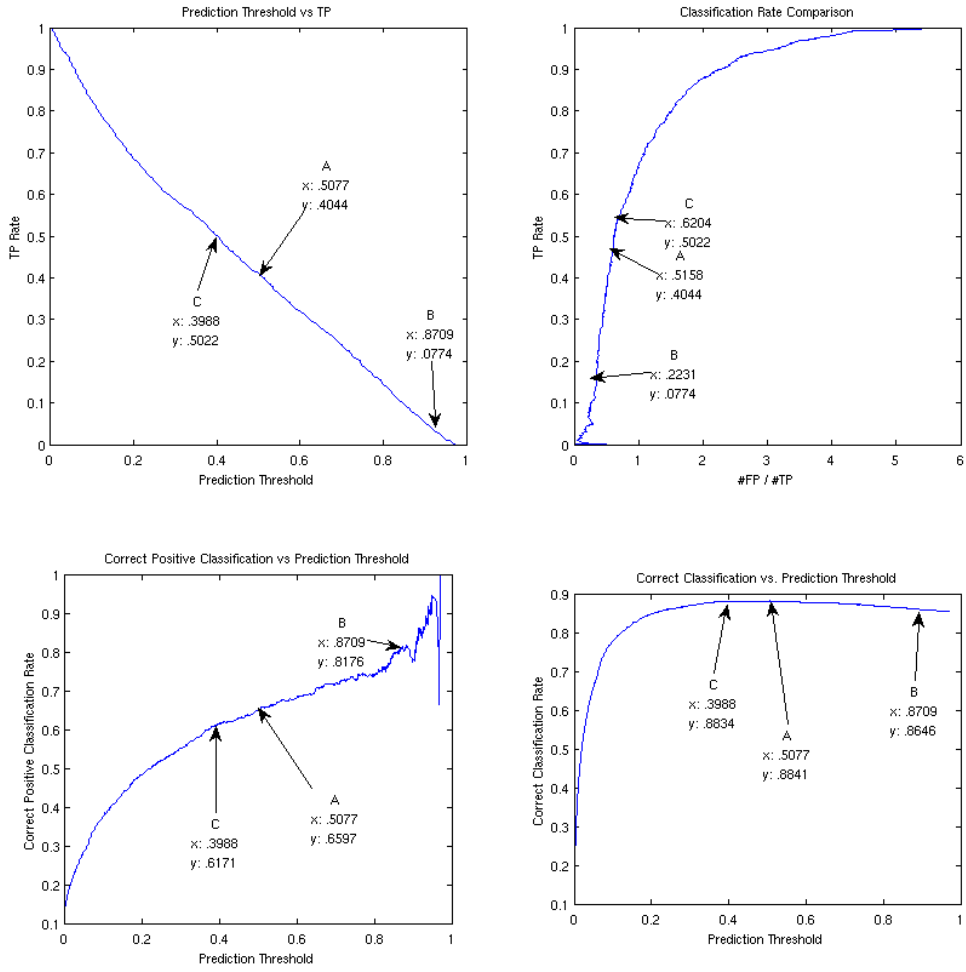

Illustration 14: Graphs illustrating the prediction performance of the PC key set

Maximum total accuracy of 88.41% is achieved at point A, although this point only accurately detects 40.44% of car segments, and 65.97% of detected segments are actually cars. By shifting the prediction threshold to B, we can significantly increase the chance of detected thresholds being car to 81.76%, but at the cost of finding very few (7.74%) of

the true car segments. A shift down the curve to C brings in significantly more true positives with a moderate increase in false positives.

Below we will present a few images to characterize the classifier. For each illustration, the image in the upper left is a black and white version of the original image with white dots for each of the detected keys which matched a key in the PC dictionary. The image in the upper right show the results of the segmentation algorithm on the original image. The image in the lower left shows the classification results of each segment. The lighter the color of the segment, the high probability that it was considered car. The lower right image is the resulting classification using the threshold determined from point A described above.

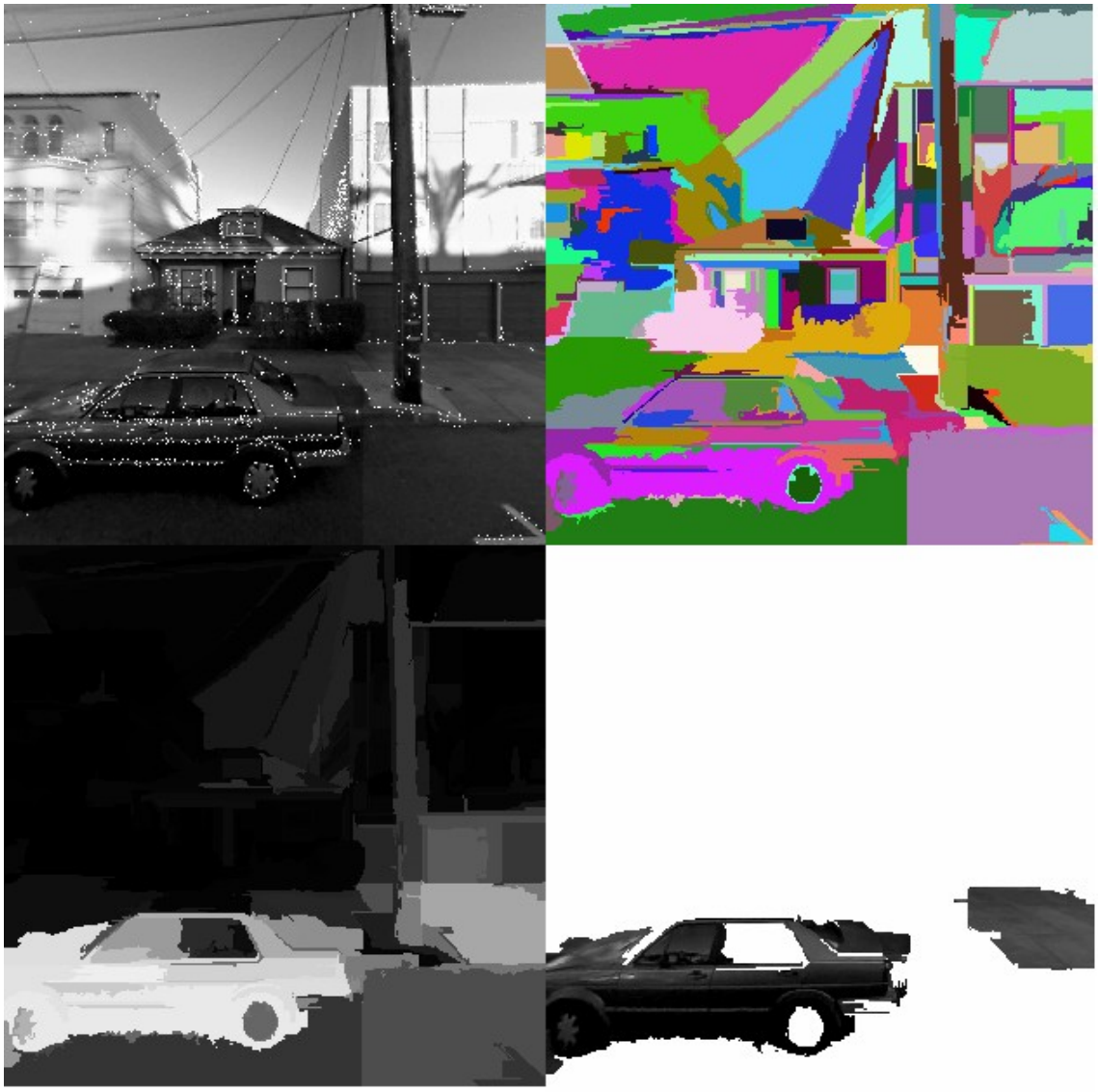

Illustration 15: An example of good classifier performance. The classifier was able to correctly pick most of the car, and incorrectly classified one non car segment. As illustrated in most examples, the classifier often incorrectly classifies sections of pavement and driveway. The rear wheel and rear passenger window of the car was likely misclassified because they have very few known keys.

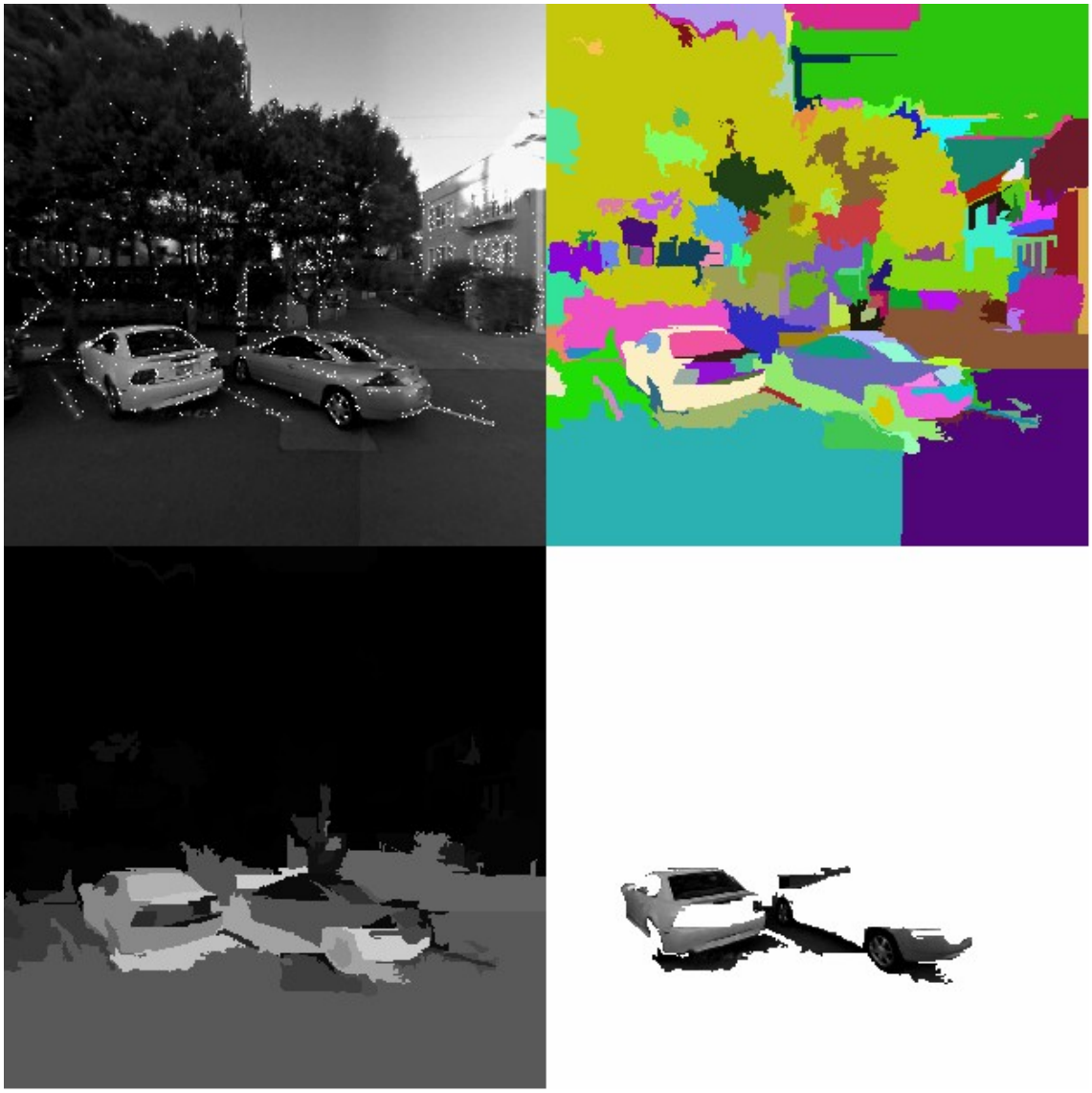

Illustration 16: Another example of good classifier performance. The cars were largely discovered, with little non-car segment misclassification. The classifier has problems classifying the backs of the cars because most training images were side-views.

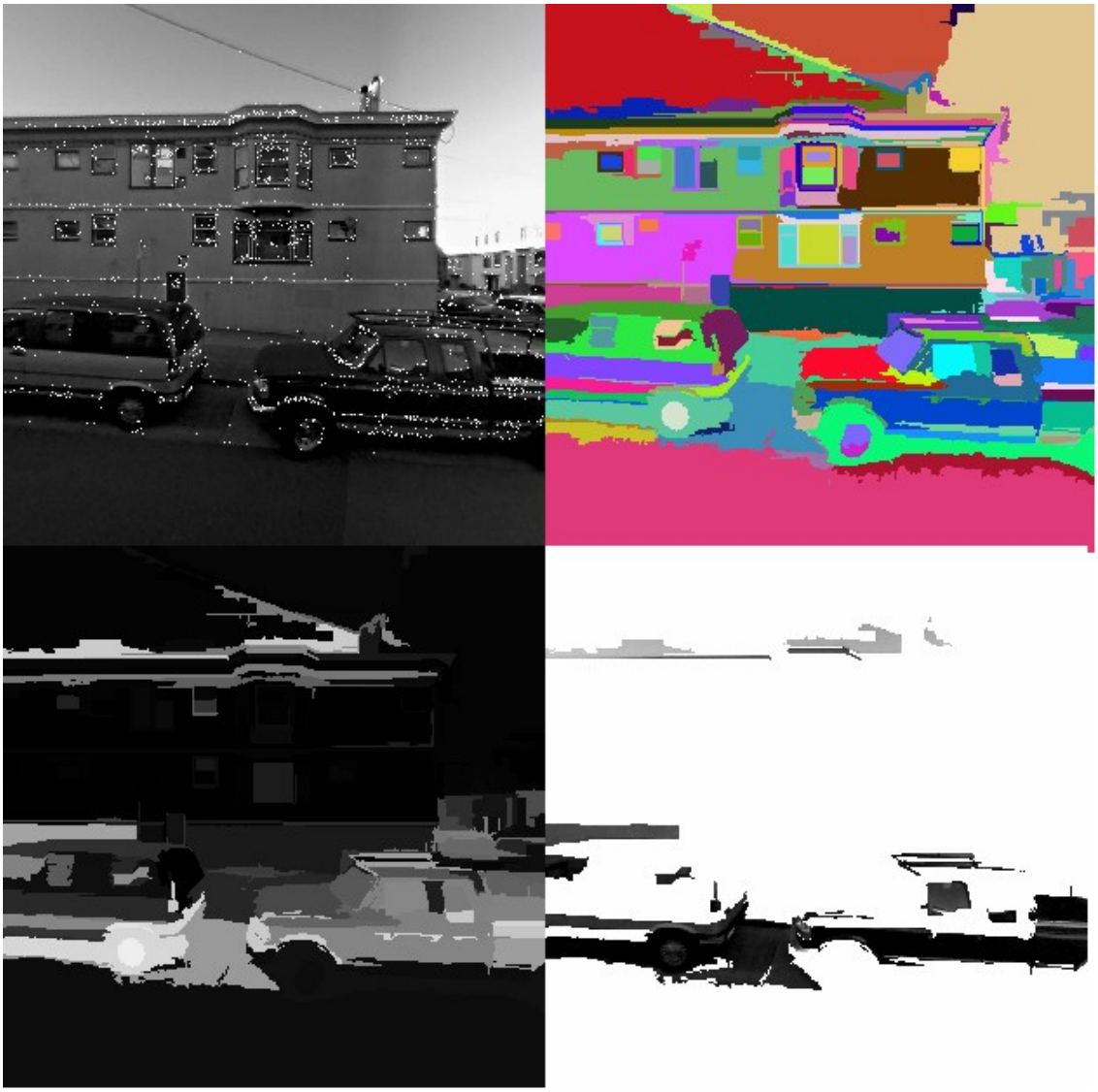

Illustration 17: An example of marginal classifier performance. The cars had reasonable detection, but several misdetections occurred at the top of the building. Additionally, the pavement area between the cars was also misclassified. Using our current scheme, it is difficult to differentiate between segments with few keys within a car and beside a car.

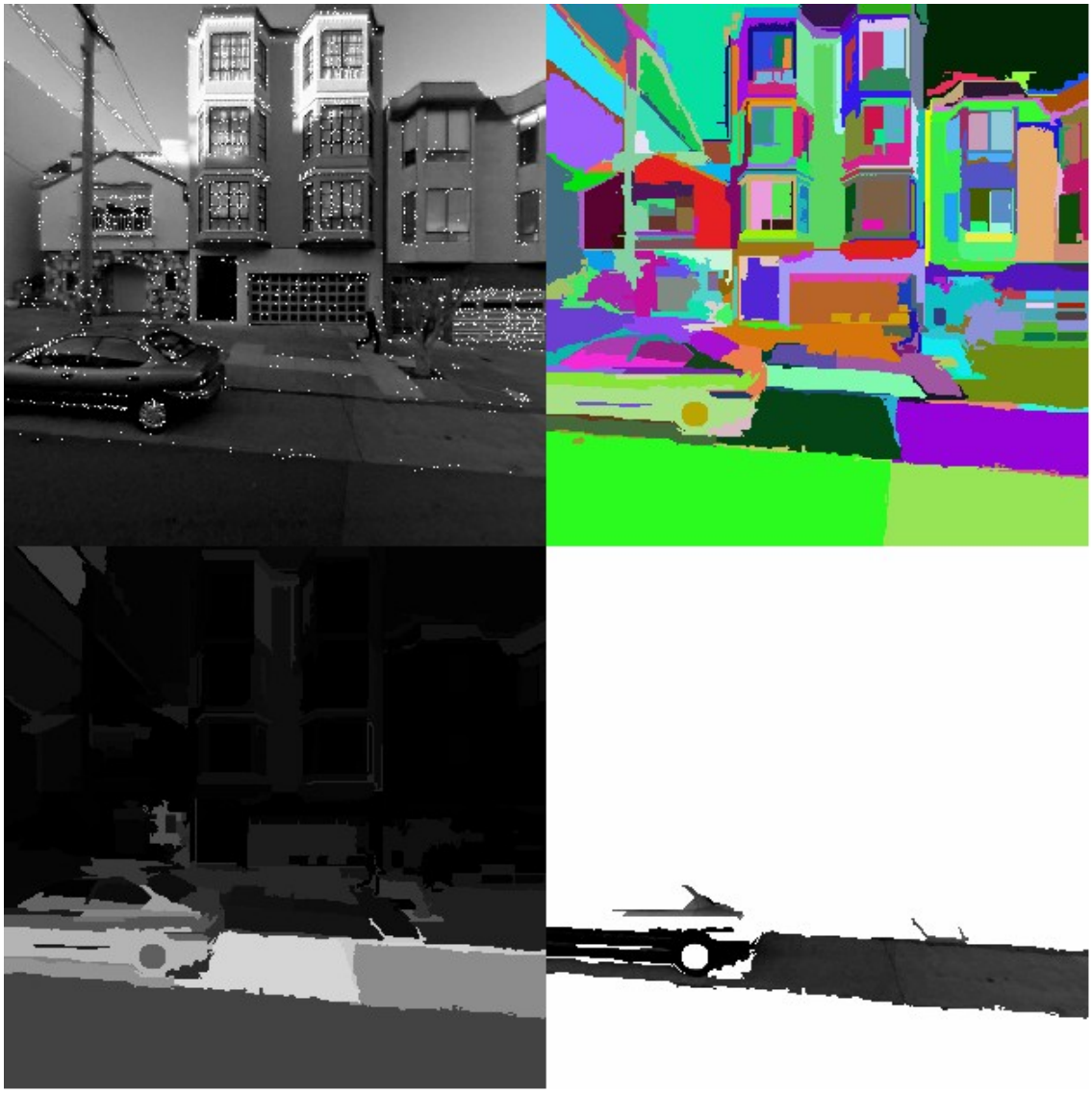

Illustration 18: An example of significant misdetection. It is likely in training that several features were included that are indicative of pavement, because cars are in the same area. This can lead to misdetections such as this one were a significant portion of pavement was considered car. Note how the pavement near the car itself has the highest probability because it is also receiving weight from the keys inside the car.

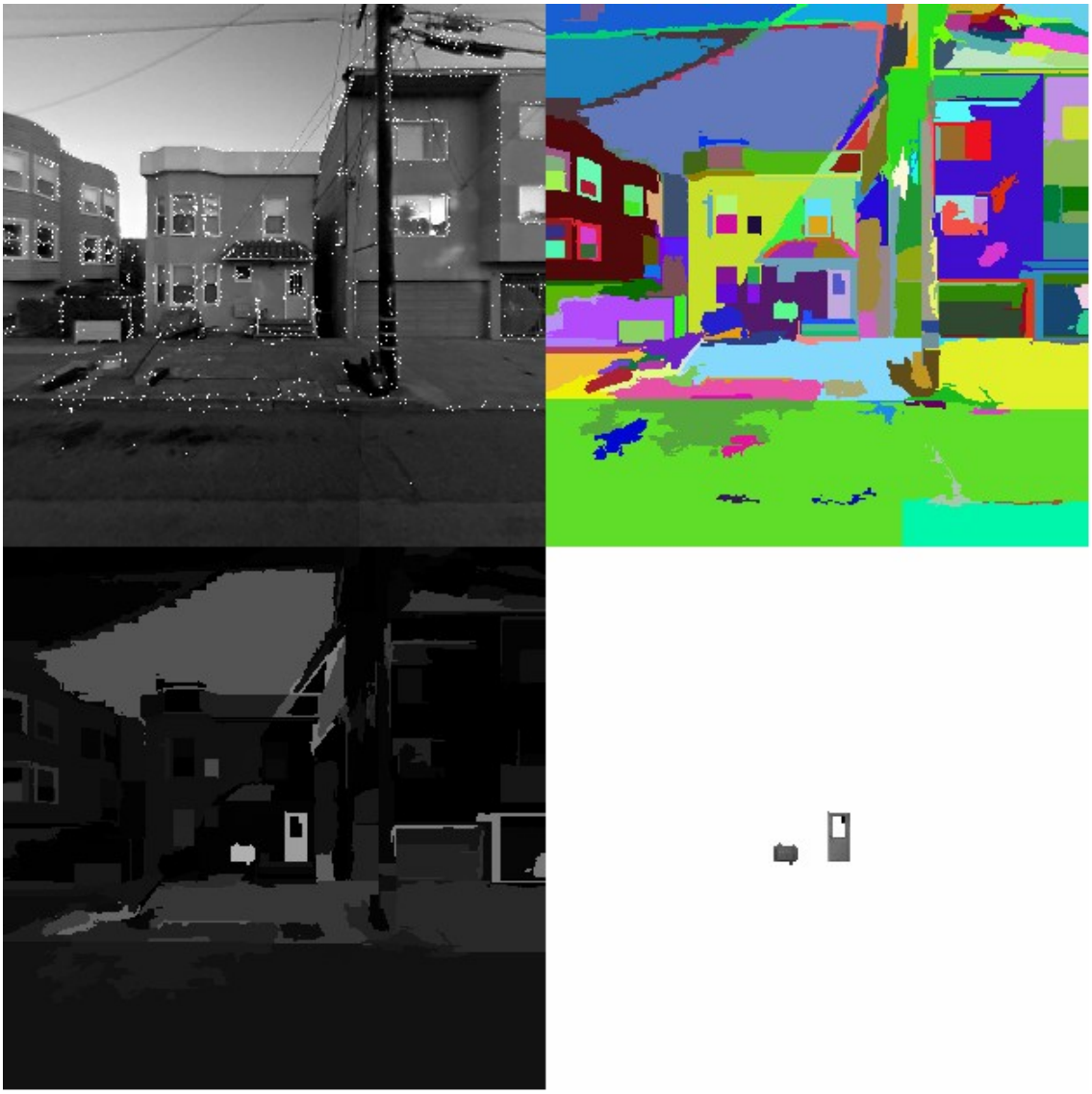

Illustration 19: An example where there is no car to detect. The classifier does a generally good job, only misdetecting two small segments of a door and an air-conditioning unit.

CHAPTER IX

CONCLUSION AND FUTURE WORK

Overall, our detection algorithm provide extremely useful information in detecting cars. The areas that caused the most difficulty were pavement and driveways; areas similar to the environment surrounding cars. In addition, areas without features such as windows were also often misdetected. HITON-PC proved extremely effective as a feature selection tool for this application, resulting in a 5-fold reduction of features with negligible change in accuracy.

Despite these shortcomings, we believe our approach has application good potential for real-time use in car detection. There are a number of ways in which we plan to improve future performance:

Key-Segment Localization

Currently, keys are attached to the segment where their center occurs. The nature of SIFT keys has them occurring on edges, which means that a key could be arbitrarily put into one of several segments that it encodes information for. In future work, it would likely prove valuable to include a key in all segments that fall within a local vicinity determined by the key's scale.

Segmentation

Informal experiments have shown that improving the size of segments improves quality of the classification. Several of the approaches described in section 2.5 use boxes to quantify the edges of car, allowing them to classify a much larger number of keypoints. In general, the more keys that can be included in each segment, the better the quality of the classifier we can build. In the future, steps should be taken to improve our segmentation in two ways: optimizing the parameters of our segmentation algorithm more rigorously, and considering other methods of segmentation. A more optimal algorithm could synergize range data from a stereo camera or laser range-finders to vary segmentation parameters depending on distance (further objects are allowed smaller segments), and to help group items that are geometrically close together. In addition, we should improve the use of texture as a parameter in our segmentation.

Keypoints

Jurie and Triggs [16] have shown that features created by algorithms such as SIFT, while highly recognizable and efficient to compute, are not as informative as image patches. Unfortunately, finding the most informative image patches for classification and matching is difficult; at runtime, our approach is considerably faster. Mikolajcyzyk [23] has proposed features similar to SIFT except that edges can be selectively ignored. These could be particularly useful in discriminating against background noise.

A more immediate gain may be had by selecting from a larger set of keys. In our experiments, we confined our search for keys to areas that contained a car, and then sub-selected among those for the 1000 most highly correlated features. This was done purely for reasons of computational tractability. Future work will be done using the entire possible key dictionary (~400K keys) to see if significant gains can be achieved.

Classifiers

Further improvements may be made to the SVM with further optimization, although experiments suggested they would be slight. Using a different type of classifier such as AdaBoost could also prove beneficial. Adding a post-processing step to add probability to segments surrounded by highly likely car segments could help catch some of the misclassified segments inside the car (such as windows).

Parallelization

Presently, the detector is much too slow to run at video frame-rates; but significant improvements could be had by parallelizing the software to make use of video card hardware, such as with CUDA. The most time-consuming parts of the algorithm are image convolutions (for sift) and large vector calculations (for comparing keys and computing the SVM classification).

A. RESULTS OF FIRST TRAINING PHASE

Results of First Training Phase

The □ parameter was set to the default of 1/k (1/78514 = .012). The u parameter was set to the default of 0. The v and d parameters were varied as per the chart. The charts on the left measure the average AUC for that set of parameters over ten-fold cross-validation. The charts on the right measure the number of trials that completed within 24 hours. DIST=350 had the best performance and was used in subsequent training.

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	0.8244	0.8165	0.8352	**0.8847**	0.8815
2	0.7621	0.8186	0.8252	0.8773	0.8835
3	0.7576	0.8115	0.8146	0.8685	0.8748
4	0.5487	0.8046	0.8116	0.8624	0.8638

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	10	10	10	10	10
2	10	10	10	10	10
3	10	10	10	10	10
4	10	10	10	10	10

DIST=350

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	0.7028	0.7327	0.7901	0.8987	**0.8990**
2	0.7462	0.7933	0.8323	0.8857	0.8954
3	0.5421	0.6324	0.8166	0.8756	0.8920
4	0.3354	0.5704	0.8022	0.8683	0.8900

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	10	10	10	10	10
2	10	10	10	10	10
3	10	10	10	10	10
4	10	10	10	10	10

DIST=400

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	0.7389	0.7752	0.7967	0.8689	0.8636
2	0.8196	0.8415	0.8448	0.8715	**0.8773**
3	0.6826	0.8279	0.8409	0.8682	0.8765
4	0.6646	0.7823	0.8336	0.8628	0.8650

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	10	10	10	10	10
2	10	10	10	10	10
3	10	10	10	10	9
4	10	10	10	10	9

DIST=450

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	0.7665	0.6890	0.6816	0.8318	0.8452
2	0.6973	0.6966	0.7885	0.8392	0.8505
3	0.7107	0.7341	0.8044	0.8412	0.8536
4	0.5000	0.6214	0.6885	0.8414	**0.8556**

d	v				
	0.0010	**0.0100**	**0.1000**	**1.0000**	**10.0000**
1	10	10	10	10	10
2	10	10	10	10	10
3	10	10	10	10	10
4	10	10	10	10	10

B. RESULTS OF SECOND TRAINING PHASE

The charts on the left measure the average AUC for that set of parameters over ten-fold cross-validation. The charts on the right measure the number of trials that completed within 48 hours. The highest performance for each set of parameters is marked in bold.

1	**1**	0.8965	0.8972	0.8971	0.8979	0.8963
	2	0.8809	0.8823	0.8820	0.8848	0.8840
	3	0.8684	0.8690	0.8684	0.8730	0.8762
	4	0.8611	0.8583	0.8578	0.8638	0.8697
2	**1**	0.8845	0.8819	0.8864	0.9004	0.8970
	2	0.8723	0.8718	0.8834	0.8955	0.8925
	3	0.8618	0.8610	0.8737	0.8911	0.8900
	4	0.8532	0.8521	0.8685	0.8860	0.8879
4	**1**	0.7740	0.8074	0.8887	**0.9008**	0.8967
	2	0.7544	0.8176	0.8862	0.8986	0.8946
	3	0.7393	0.8145	0.8806	0.8961	0.8931
	4	0.7327	0.8080	0.8727	0.8931	0.8921
8	**1**	0.7315	0.8338	0.8907	0.9000	0.8920
	2	0.7293	0.8309	0.8881	0.8982	0.8921
	3	0.7254	0.8120	0.8932	0.8973	0.8884
	4	NA	NA	0.8954	0.8935	0.8881

1	**1**	10	10	10	10	10
	2	10	10	10	10	10
	3	10	10	10	10	10
	4	10	10	10	10	10
2	**1**	10	10	10	10	10
	2	10	10	9	10	10
	3	10	10	9	10	10
	4	10	10	10	10	10
4	**1**	10	10	10	10	10
	2	9	10	10	10	10
	3	10	10	10	10	10
	4	10	10	10	10	10
8	**1**	8	10	10	10	10
	2	10	10	10	10	9
	3	10	10	2	9	9
	4	0	0	1	9	6

2	**0.000003**	0.8845	0.8844	0.8843	0.8847	0.8882
	0.000006	0.8847	0.8862	0.8839	0.8836	0.8941
	0.000012	0.8846	0.8853	0.8836	0.8835	0.8981
	0.000024	0.8845	0.8841	0.8834	0.8858	0.8999
4	**0.000003**	0.7603	0.7727	0.7739	0.8191	0.8904
	0.000006	0.7657	0.7737	0.7740	0.8594	0.8957
	0.000012	0.7698	0.7737	0.7739	0.8793	0.8978
	0.000024	0.7718	0.7737	0.8041	0.8881	**0.9010**
8	**0.000003**	0.7250	0.7225	0.7328	0.8458	0.8922
	0.000006	0.7223	0.7226	0.7500	0.8718	0.8967
	0.000012	0.7225	0.7227	0.7788	0.8827	0.8996
	0.000024	0.7227	0.7290	0.8296	0.8903	0.9003

2	**0.000003**	10	10	10	10	10
	0.000006	10	10	10	10	9
	0.000012	10	9	10	10	10
	0.000024	10	10	10	10	9
4	**0.000003**	9	10	10	10	10
	0.000006	10	10	10	10	10
	0.000012	10	10	10	10	9
	0.000024	10	10	10	10	10
8	**0.000003**	9	10	10	10	10
	0.000006	10	10	10	10	10
	0.000012	10	10	10	10	10
	0.000024	10	10	10	10	10

8	**0.000003**	0.7215	0.7224	0.7229	0.7934	0.8856
	0.000006	0.7219	0.7225	0.7328	0.8459	0.8922
	0.000012	0.7222	0.7226	0.7501	0.8719	0.8967
	0.000024	0.7224	0.7222	0.7791	0.8828	**0.8995**

8	**0.000003**	10	10	10	10	10
	0.000006	10	10	10	10	10
	0.000012	10	10	10	10	10
	0.000024	10	9	10	10	10

	3	0.8645	0.8645	0.8808	0.8902	0.8903
	4	0.8559	0.8560	0.8731	0.8861	0.8877
4	**1**	0.8099	0.8690	0.8959	0.8985	0.8968
	2	0.7911	0.8615	0.8924	0.8968	0.8973
	3	0.7806	0.8481	0.8878	0.8943	0.8948
	4	0.7736	0.8337	0.8806	0.8917	0.8926
8	**1**	0.7703	0.8777	0.8963	0.8977	0.8953
	2	0.7579	0.8716	0.8927	0.8973	0.8954
	3	0.7452	0.8568	0.8878	0.8951	0.8929
	4	0.7424	0.8313	0.8823	0.8921	0.8904

	3	10	10	10	10	10
	4	10	10	10	10	10
4	**1**	10	10	10	10	10
	2	10	10	10	10	10
	3	10	10	10	10	10
	4	10	10	10	10	10
8	**1**	10	10	10	10	10
	2	10	10	10	10	10
	3	10	10	10	10	10
	4	10	10	10	10	10

4	**0.000003**	0.8038	0.8057	0.8096	0.8100	0.8844	**4**	**0.000003**	10	10	10	10	10
	0.000006	0.8039	0.8078	0.8098	0.8275	0.8913		**0.000006**	10	10	10	10	10
	0.000012	0.8038	0.8089	0.8099	0.8632	0.8952		**0.000012**	10	10	10	10	10
	0.000024	0.8047	0.8095	0.8099	0.8810	0.8974		**0.000024**	10	10	10	10	10
8	**0.000003**	0.7605	0.7592	0.7590	0.8039	0.8863	**8**	**0.000003**	10	10	10	10	10
	0.000006	0.7598	0.7591	0.7590	0.8526	0.8921		**0.000006**	10	10	10	10	10
	0.000012	0.7595	0.7590	0.7655	0.8750	0.8958		**0.000012**	10	10	10	10	10
	0.000024	0.7593	0.7590	0.7912	0.8841	**0.8977**		**0.000024**	10	10	10	10	10

	0.000012	0.7598	0.7591	0.7590	0.8526	0.8921
	0.000024	0.7595	0.7590	0.7655	0.8750	0.8957

	0.000012	10	10	10	10	10
	0.000024	10	9	10	10	10

REFERENCES

[1] Agarwal, S. and Roth, D., "Learning a Sparse Representation for Object Detection," *Lecture Notes in Computer Science*, issue 2353, pp. 113-130, 2002.

[2] Aliferis, C., Tsamardinos, I., and Stanikov, A., "HITON: A novel blanket algorithm for optimal variable selection," *AMIA 2003 Annual Symposium Proceedings*, pp. 21-25, 2003.

[3] Aliferis, C., Tsamardinos, I., Statnikov, A., and Brown, L.E., "Causal Explorer: A Causal Probabilistic Network Learning Toolkit for Biomedical Discovery," *International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences,* pp. 371-376, 2003.

[4] Aliferis, C., Statnikov, A., Tsamardinos, I., Mani, S., and Koutsoukos, X., "Local Causal and Markov Blanket Induction for Causal Discovery and Feature Selection for Classification," *Unpublished Manuscript,* 2008

[5] Chang, C.-C. and Lin, C.-J., "LIBSVM: A Library for Support Vector Machines," *Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm*, 2001.

[6] Cortes, C. and Vapnick, V., "Support Vector Networks," *Machine Learning,* vol. 20, pp. 273-297, 1995.

[7] Dorko, G., and Schmid, C., "Object Class Recognition Using Discriminative Local Features," Submitted to *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 2004.

[8] Dorko, G., "Selection of Discriminative Regions and Local Descriptors for Generic Object Class Recognition," Ph.D. diss., Institut National Polytechnique de Grenoble, 2006.

[9] Felzenszwalb, P.F. And Huttenlocher, D.P., "Efficient Graph-Based Image Segmentation," *International Journal of Computer Vision,* vol. 59, number 2, pp. 167-181, 2004.

[10] Forsyth, D. and Ponce, J., *Computer Vision: A Modern Approach*, Upper Saddle River: Prentice Hall, 2003, pp. 110-113.

[11] Freund, Y. and Schapire, R.E., "A Decision-Theoretic Generalization of On-line Learning and an Application to Boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119-139, August 1997.

[12] Geronimo, D., Lopez, A., Ponsa, D., and Sappa, A.D., "Haar Wavelets and Edge Orientation Histograms for On-Board Pedestrian Detection," *Lecture Notes in Computer Science*, pp. 418-425, 2007.

[13] Harris, C. and Stephens, M., "A Combined Corner and Edge Detector," *Fourth Alvey Vision Conference*, Manchester, UK, pp. 147-151, 1988.

[14] Hastie, T., Tibshirani, R., Friedman, J., *The Elements of Statistical Learning: Data*

Mining, Inference, and Prediction, New York:Springer, 2001, pp. 41-75.

[15] Joachims, T., "Making Large-Scale SVM Learning Practical," In Scholkopf, B, Burges, C.J.C., and Smola, A.J., editors, *Advances in Kernel Methods – Support Vector Learning*, Cambridge:MIT Press, 1998.

[16] Jurie, F. and Triggs, B., "Creating Efficient Codebooks for Visual Recognition," *Tenth IEEE International Conference on Computer Vision*, vol. 1, pp. 604-610, 2005.

[17] Kohavi, R., and John, G., "Wrappers for Feature Subset Selection," *Artificial Intelligence*, vol. 97, pp. 273-324, 1997.

[18] Leung, B., "Component-based Car Detection in Street Scene Images," Master's thesis, Massachusetts Institute of Technology, 2004.

[19] Lowe, D.G., "Object Recognition from Local Scale-Invariant Features," *International Conference on Computer Vision,* Corfu, Greece, pp. 1150-1157, Sept.1999.

[20] Lowe, D.G. 2001. Local Feature Clustering for 3D Object Recognition. *IEEE Computer Society Conference on Computer vision and Pattern Recognition,* vol 1, 2001. pp. 682-688

[21] Lowe, D.G. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60, 2 (2004), pp. 91-110.

[22] Mallat, S., "A Theory for Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674-693, July 1989.

[23] Mikolajczyk, K., Zizzerman, A., and Schmid, C., "Shape Recognition with Edge-Based Features," *Proceedings of the British Machine Vision Conference*, Norwich, U.K., 2003.

[24] Mohan, A., Papageorgiou, C., and Poggio, T., "Example Based Object Detection in Images by Components," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 4, pp. 349-361, Apr. 2001.

[25] Ohlander, R., Price, K., and Reddy, R., "Picture Segmentation by a Recursive Region Splitting Method," *Computer Graphics Image Processing*, vol. 8, pp. 313-333, 1978.

[26] Osuna, E., Freund, R., and Girosi, F., "Support Vector Machines: Training and Applications," *AI Memo 1602*, Massachusetts Institute of Technology, 1997.

[27] Papageorgiou, C.P., and Poggio, T., "A Trainable Object Detection System: Car Detection in Static Images," *Technical Memo 1673*, Massachusetts Institute of Technology, 1999.

[28] Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, SanMateo:Morgan Kaufmann Publishers, 1988.

[29] Shi, J., Malik, J., "Normalized Cuts and Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, part 8, pp. 888-905, 2000.

[30] Stollnitz, E.J., DeRose, T.D., and Salesin, D.H., "Wavelets for Computer Graphics: A

Primer," *Technical Report 94-09-11*, Department of Computer Science and Engineering, University of Washington, September 1994.

[31] Vedaldi, A., "An Open Implementation of the SIFT Detector and Descriptor," *UCLA CSD Tech. Report 070012, 2006.*

[32] Witkin, A.P., "Scale-Space Filtering," *International Joint Conference on Artificial Intelligence*, Karlsruhe Germany, pp. 1019-1022, 1983.

[33] Wu, T.F. Lin, C.J., and Weng, R.C., "Probability Estimates for Multi-class Classification by Pairwise Coupling," *Journal of Machine Learning Research,* vol. 5, pp. 975-1005, 2004.

[34] Wu, Z. and Leahy, R., "An Optimal Graph Theoretic Approach to Data Clustering: Theory and its Application to Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol 11, pp. 1101-1113, November 1993.

[35] Zahn, C.T., "Graph-Theoretic Methods for Detecting and Describing Gestalt Clusters," *IEEE Transactions on Computing*, vol 20, pp. 68-86, 1971.