Abstract

Current image retrieval techniques, while extremely accurate and effective on a relatively small scale, are unable to scale to very large database conditions due to the limitations imposed by the curse of dimensionality on nearest neighbor searching. In addition to this, the better systems are unsuited to visual browsing of the database. We propose the `HUGINN` framework for a concept-based image retrieval system, which uses state of the art machine learning techniques to constrain searching and browsing through the use of image concept.

This report outlines the current state of the art in image retrieval and outlines a current system using these techniques and a novel technique called probabilistic pose prediction to achieve excellent results for near-duplicate and sub-image retrieval. It then details a novel system for boosting machine learners to aid in the extraction of concept from images, then outlines a plan for future research culminating in a complete image retrieval and browsing system which incorporates machine learning.

Using Machine Learning for Image Retrieval

Matthew Johnson

November 4, 2005

Contents

Chapter 1

Introduction

The first time you visit a new supermarket, finding the groceries you are looking for can be a daunting task. There are thousands of items, and only a relative few of them fit your grocery list. To help with this problem, a supermarket is divided into sections which contain food belonging to a logical group (e.g. dairy, meat, vegetables, etc.). In this way, you can look at the next item on your list, deduce which group it belongs to, and then only search the specific area in the supermarket where that item will be found, thus constricting your search and saving you time. Similarly, if you don't know precisely what you want for dinner yet have a general idea (i.e. meat of some kind and two vegetables) browsing is also made much easier by being able to just walk through a section and see what is available for purchase.

Image retrieval today is like a supermarket where the food is arranged by size, weight or price. Global statistics about images, such as color or texture histograms, and local distinctive features are used to index into databases of images. While this works well for smaller databases, these systems invariably have trouble with distinguishing between similar items in large databases (imagine trying to find an item in the supermarket having only been told its price, as there will most certainly be several items that have the same price). In this report, we will focus first on the content to be found in images and how that content is used by state of the art image retrieval systems. The second focus will be on how to extract concepts from images, after which we propose a method of using these concepts to create better retrieval systems which divide an image database into logical subsections to ease searching and enable effective browsing.

1.1 Current Systems

Current systems for image retrieval take a variety of forms. The first effective systems such as IBM's QBIC system [68] and many others more current [88, 45] use a combination of raw image properties and text, with other systems like Google's image search using a text search on associated text as the sole

indexer. Other systems, such as Ogle et al.'s Chabot system [70] or Carson et al.'s Blobworld [18] allow the user to provide a sample image or images to the system and find similar images based on their features. More recent systems make use of local feature descriptors, such as Lowe's SIFT descriptor [59], to perform difficult tasks such as sub-image and near-duplicate retrieval with high degrees of accuracy [52]. Yet other systems allow the user to browse a database in a completely visual and interactive way to find the desired image [22]. All current systems, however, extract content of some kind from a database of images and compare that content to query content which is provided by the user. This is much like asking someone to find an item by giving its mass, color, smell and specifying that it is "bigger than a bread box". In fact, it is worse than this comparatively simple situation, as the dimensionality of image content (sometimes greater than 100 dimensions) makes finding a near match much more difficult.

1.2 Approach

An approach that has only been tentatively explored [5] is the use of object recognition, specifically the extraction of "concepts" from images, in image retrieval. Instead of extracting simply content, a system able to extract a concept from a query can then apply this concept as an additional constraint on its search, decreasing the sample space and improving search accuracy. Such a system would understand the linking and underlying meaning behind collections of content so that when the user provides a query, it can extrapolate that a search for a furry object with four limbs and white and black stripes is in fact a search for a "zebra", thus eliminating false positives linked to "penguins". This idea is exciting in its possibilities, and we present the `HUGINN` framework for such a system in this report along with a research plan leading to a complete concept based image retrieval system in two years.

1.3 Structure of this Report

This report is structured as follows:

- **Chapter 2** discusses the concept of image content and provides an in depth survey of the various kinds of content used by modern systems. It also contains a necessarily brief discussion of segmentation in Section 2.1.1. Like the raw data from an instrument, image content is meaningless in and of itself yet all computer vision is built upon it. As with all sciences, the world we study is too varied and complex to be modelled in its entirety and we must rely on key features and global statistics in order to interpret it.

- **Chapter 3** examines the state of the art in image retrieval, ending with a study of a fully-functional near-duplicate retrieval system in Section 3.5.

- **Chapter 4** explores the ideas behind image concept. Concept provides models and predictions for content and connections between content and meaning. The chapter concentrates mainly on aspects of machine learning and its applications to computer vision, specifically in the understanding of object classes and their statistical relations to certain types of content. It assumes a basic understanding of probability, mainly Bayesian statistics.

- **Chapter 5** contains the exposition and results of a novel boosting framework for extracting concept as published in [46]. This system improves on state-of-the-art systems and is the initial step in a larger framework of exploration in this area.

- **Chapter 6** concludes the report and summarizes its content. It then details the `HUGINN` concept-based image retrieval framework and presents a timetable for the construction of various prototype systems within the framework, leading to the completion of the PhD project in two years.

All figures were produced by the author's implementations of various algorithms or techniques with the exception of Figures 2.7 and 2.8, which were obtained under the GNU Free Documentation License from online reference materials on color, and where otherwise noted.

Chapter 2

Representing Image Content

As the human brain captures the gigabytes of information that come through the retinas every second, it performs a data reduction of several orders of magnitude such that the information can be processed for use in cognition tasks. The exact reduction is still a subject of some debate, but neurology has yielded a good understanding of its nature, and the field of computer vision has used this description as a guide in solving the visual data reduction problem computationally. This chapter will explore some of the methods used in computer vision to extract meaningful features from images that can then be used for other vision tasks, thus making the formulation and motivation of those tasks clearer.

2.1 Global Features

Global features are gathered over an entire image or image region. Many statistics, such as texture (i.e. repeating or distinctive visual patterns) or shape are only meaningful when looked at in a large area. This section examines various ways to describe color, texture and shape at the level of an image segment and how features can be assembled using these descriptions.

2.1.1 Segmentation

Before we can discuss the extraction of features at the level of an image segment, it is worth taking a brief foray into the techniques used to decide those segments. Also, by studying a sample computer vision problem and its requirements in terms of image data, the need for various kinds of image content will become apparent.

The subject of image segmentation is a deep and abiding problem in computer vision. The ideal way to divide an image into constituent parts is at best

unclear. Typical efforts use texture, color, shape or a combination of these three to determine which regions should be co-extensive. There are several ways of segmenting an image, but this section will only highlight the three main methods in use at the current time.

Expectation Maximization

One method for the segmentation of images is probabilistic in nature. We can think of each pixel in the image as having a probability of being produced by a particular component, c, that represents the image segment to which it belongs. That component has particular properties that describe what kinds of pixels belong to it, and depending on those properties a pixel either belongs or doesn't belong. In the ideal situation the probability for the correct component would be 1 and all others would be 0, however in reality it will be a distribution over the components such that $\sum_c P(c|x) = 1$ where x is the pixel in question. Component membership (for the purpose of segmentation) is determined by the Bayes Decision Rule, namely that the component for which $P(x|c)$ is greatest is chosen.

As we don't know what properties each component has at the start, we derive those properties from the pixels which belong to the component, however we can't assign pixels to each component until we know what properties that component has. More simply, we can't assign pixels to components until we know what pixels are in each component. It seems to be an insurmountable chicken and egg problem, however we can think of it as one of missing data. In essence, there is a missing piece of information, a hidden piece: the assignment of pixels to components. The Expectation Maximization algorithm, first described in [26], was designed to solve precisely this hidden data problem. Though there are a variety of papers that utilize the EM algorithm, as it is colloquially known, a particularly good explanation of the technique can be found in [10].

The way in which various EM-based segmentation algorithms differ depends on the properties used to describe components. They almost all use multivariate normal distributions (Gaussians) to model $P(x|c)$, however some use color, texture or a combination of the two as the property, or feature, vector. The use of Gaussians to model components results in what is effectively a collection of hyper-ellipses in the feature space. A Gaussian can be described by its mean value, μ, and its covariance matrix Σ. If a particular value is chosen to scale Σ, these two quantities describe an ellipse which can help with visualization of the process as shown in Figure 2.1.

By way of example, in [7], Belongie and Malik describe a system for EM Segmentation that would later be used by the Blobworld image retrieval framework [18]. Their features consist of 6 quantities. The first three are HSV color (though they later used CIELab) represented in cartesian cone coordinates: $sv\cos(2\pi h)$, $sv\sin(2\pi h)$, and v. The saturation is multiplied through due to the fact that for lower saturation values hue has very little meaning. Section 2.1.4 has a more detailed description of saturation and an example of a cone-based color representation system.

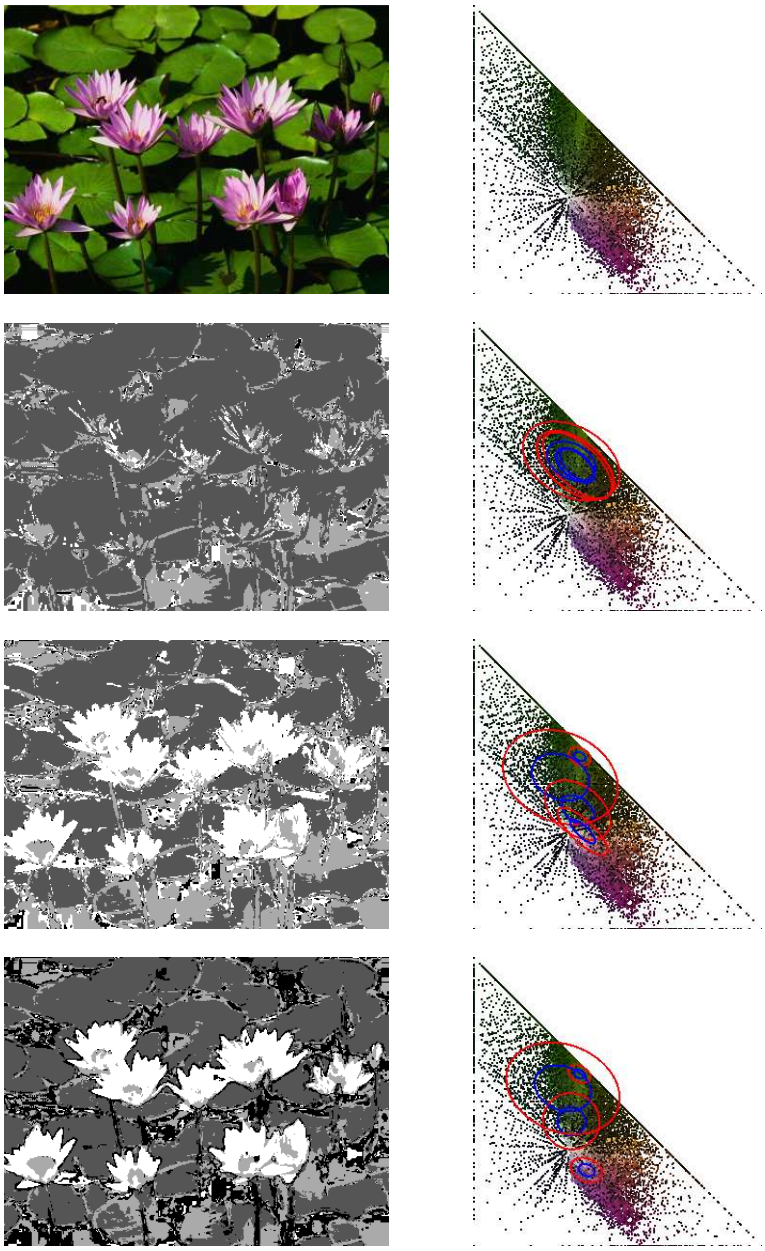

Figure 2.1: *EM Segmentation.* The second, twenty second, and forty-sixth iterations are shown of a bivariate normal distribution mixture model fitting process by EM. On the left are images of the segmentation, and on the right depictions of the original image in rg space, with $r = \frac{R}{R+G+B}$ on the x-axis and $g = \frac{G}{R+G+B}$ on the y-axis. Each component is depicted as two ellipses on the graphs, with the blue ellipse set at σ and the red ellipse at 2σ. The component that a pixel belongs to is denoted by its intensity value.

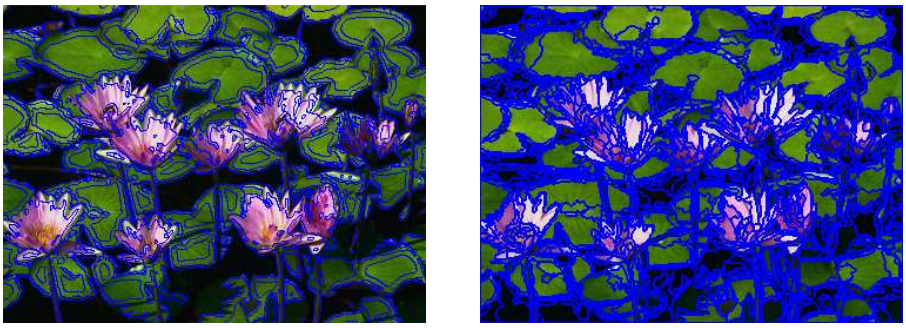

Figure 2.2: *Segmentation Examples* The left image is an EM segmentation from an implementation of the system described in [7]. Notice how the segments delineate between central and boundary regions of the same object, which is suboptimal behavior. The image to the right is output from the Mean Shift texture and color segmenter from [37], using code obtained from the author's website. While these segments are very coherent in color and texture, the algorithm fails to find object-wide characteristics, resulting in considerable over segmentation.

The second three numbers deal with texture, and are slightly more complicated to extract. First, the second moment matrix is estimated from the first difference approximation of the gradient (as described in Section 2.2.1) and its eigensystem calculated. This results in eigenvalues λ_1 and λ_2 with corresponding eigenvectors ϕ_1 and ϕ_2, where ϕ_1 is the first,dominant direction of the gradient with energy λ_1 and ϕ_2 is the second direction with energy λ_2. This is calculated at various scales, and then for every pixel the scale is found at which the polarity stops changing, where polarity is a measure of how many gradients in the neighborhood of the pixel at that scale are going in the direction of x_1. Then three quantities are calculated at that scale: the anisotropy ($a = 1 - \frac{\lambda_2}{\lambda_1}$), the normalized contrast ($c = 2\sqrt{\lambda 1 + \lambda 2}$) and the above mentioned polarity. The numbers which are used in the feature vector are ac, pc and c, as again anisotropy and polarity have little meaning in areas of low contrast. An image and its accompanying segmentation using this algorithm can be seen in Figure 2.2.

EM, though widely used as a technique for a variety of computer vision problems and machine learning problems in general, has a few drawbacks. The main one is related to its formulation as a maximum likelihood estimator, in which while it is guaranteed that with subsequent iterations of the algorithm the likelihood will increase the algorithm itself may be increasing towards a local maximum, not the global one. As such, different initializations of the system (initial values for $P(x|c)$) result in wildly varying segmentations, thus making the proper initialization of these systems a bit of a black art. Another problem which EM shares with K-means clustering (discussed briefly in Section 2.1.5) lies in choosing the number of components used. Again, the choice of the number of components is a difficult one. More components almost certainly means a

better likelihood but will lead to over-segmentation, whereas less components fail to capture the various distinct areas of the image. Finding the right number of components is often application or domain dependent and even then only found after experimentation, with many algorithms trying a range of values and choosing the best based on the minimum description length principle[1]. The final problem is that it assumes (due to the Gaussian mixture model) that all clusters in the space of points assume the same shape, namely that of a hyper-ellipse, an assumption which is most likely unfounded.

Mean Shift

An alternative to the EM method is the mean shift algorithm as presented by Comaniciu [19], which deals in part with some of the shortcomings of that technique. The algorithm performs a transformation from feature space to density space, and uses gradients in density space to find the centers of arbitrary clusters. This density space transformation is achieved using kernel density estimation (also known as the Parzen window technique). Define an image I as a collection of vectors $\{x_1, x_2, \ldots, x_n\}$ computed for each pixel and in R^d, where each dimension of R^d corresponds to a dimension of the chosen feature vector. For every x a multivariate kernel density estimator with kernel $K(x)$ and a symmetric positive definite $d \times d$ bandwidth matrix $\mathbf{H}$, is applied using

$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} K_H(x - x_i) \tag{2.1}$$

where

$$K_H(x) = |H|^{-\frac{1}{2}} K(H^{-\frac{1}{2}} x). \tag{2.2}$$

The gradient of this function, $\hat{\nabla} f(x) \equiv \nabla \hat{f}(x)$, is then calculated giving the gradient of the density at that point. The direction of the gradient is known as the mean shift and is guaranteed to point towards an area of higher density in the space, a property first remarked by Fukunaga and Hostetler [33]. Therefore, at every point x one centers the window at x and estimates the mean shift. The window is then moved in the direction of the mean shift. This process is repeated until the window has stabilized at a peak in the density function. This "mode", though it may not actually be present in the image, can be thought of as the center of a dense pocket of points in feature space and is then used as a label for that point, thus achieving a segmentation.

This technique has some strong advantages, namely that it doesn't need to be initialized with a preset number of clusters and does not make assumptions about the shapes of point clusters in feature space. Mean shift has been used for segmentation by color in [19] and by texture [37], with a combined system's results shown in Figure 2.2. It does make one assumption about feature space,

[1]The minimum description length principle dictates that if there are two descriptions of a situation with similar or equal likelihood, the shorter one should be chosen. In this case, if two models with a different number of components have the same likelihood, the one with fewer components should be chosen.

however, that is often incorrect: that it is Euclidean in nature (a common problem discussed in further detail in Section 3.3). The problem of distance is shared by this technique and all clustering methods, EM or otherwise, and is not easily surmountable.

Graph Based Methods

Another way of approaching image segmentation is to treat the image as a connected graph in which each node is a pixel from the image and the edges are weighted according to the similarity between the two nodes. In this formulation, segments correspond with strongly connected subgraphs and the task is to separate the graph into those subgraphs. The degree of similarity between two subgraphs can be computed as the sum of the weights of all edges connecting them. In graph theory this is known as the "cut", computed for subgraphs A and B as

$$cut(A, B) = \sum_{a \in A, b \in B} w(a, b). \tag{2.3}$$

Ideally we want to divide the graph into subgraphs such that the cut between the two graphs is minimal, meaning that the two graphs are maximally dissimilar from each other. This is known as the *minimum cut* problem, and there exist efficient algorithms to solve it even though there are an exponential number of partitions. The problem with the minimum cut is that too often it simply cuts an outlying node or nodes off from the rest of the graph, as this involves a minimal number of low-weight edges. Shi and Malik introduce in [85] the concept of the normalized cut to provide a different measure of disassociation between two subgraphs. They define association as $assoc(A, B) = \sum_{a \in A, b \in B} w(a, b)$ and the normalized cut as

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \tag{2.4}$$

where $V = A \cup B$. They then find the minimal normalized cut and use that to partition the graph, thus dividing the image into two segments. This process is repeated on each subgraph until a predefined stopping criterion has been reached, in this case that the value of the next cut is below a certain threshold. This produces a segmentation such as that shown in Figure 2.3. While the segmentation does capture coherent image regions, there is significant oversegmentation that occurs due to subgraph divisions. Also, the similarity measure, though it is not specified (a strength of the algorithm) is often simply inverse Euclidean distance which leads to the problem discussed in Section 3.3.

An alternate scheme of graph-based segmentation is introduced by Boykov et al. in [13] as the Graph Cut algorithm. It also tries to find the minimum cut, however unlike Shi and Malik, who defined a new minimum cut condition (the normalized cut) to avoid the standard problems inherent in the pure minimum cut, they redefined the edge weighting process and the graph structure to produce a graph which will be far less likely to have a trivial minimum cut

Figure 2.3: *Result of Normalized Cuts Segmentation* This is an example of a Normalized Cuts segmentation. Note that while it segments out regions which make sense (such as the circle for the sun) it also over-segments smooth areas due to the nature of the stopping condition for the bipartite splitting. Image obtained from Prasad Gabbur.

solution. This graph consists not just of the pixels, but also of two terminal nodes, α and β. The weights are defined using the concept of energy, and the cut is calculated to achieve minimum energy. Each cut is achieved iteratively by way of either assigning a preliminary label to all pixels and swapping a single pixel between the two labels, with each swap reducing energy, or by adding pixels to a kernel from an unlabelled set, with each addition being that which minimally increases the energy. The two algorithms, swapping and additive, are defined so that the weighting function, $V_{p,q}(\alpha, \beta)$, must be either a semi-metric or a metric respectively. Besides that restriction, however, it can be any energy discontinuity preserving function, such as the Markov Random Field (MRF) based function proposed by Geman and Geman [36].

2.1.2 Histograms

By far, the fastest statistic to gather about an image besides its dimensions is an intensity histogram. A histogram, or bar chart, is a means by which the frequency of various values can be quantified. In an image, this means that the intensity of a pixel, or how bright it is to the human eye, is recorded until the total number of occurrences for each value has been tabulated.

There are several problems with this very basic approach. First, in an analog image it is unlikely that many of the real-valued device responses will repeat. In computer vision, however, we tend to deal with digital images which have discrete pixels and intensity values, and as such can ignore this issue for the moment. In Figure 2.4, an image is shown with four accompanying intensity histograms. Each of them are created from the pixel values in that image, but in each the size of the bins is changed, with a total of 255, 127, 63, and 31 bins respectively from left to right. The optimal bin size for an application

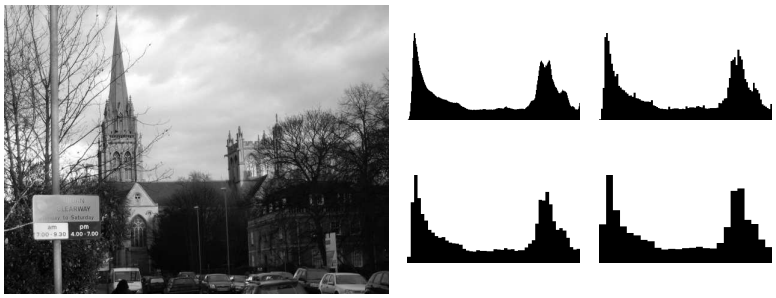

Figure 2.4: *Intensity Histograms at Different Scales* These four histograms are shown with the image they represent. Each is at a different scale, from left to right and top to bottom they have 255, 127, 63 and 31 bins. Notice how as the bin size increases sampling errors occur.

is difficult to determine. A typical task is the matching of two images to see whether they are the same. Shown in Figure 2.5 are three different photographs of the same object in the same place with the same camera, but under different conditions, each with its accompanying histogram. As can be seen, each of these histograms is quite different from the others. Increasing bin size will make histograms more similar over all but reduce match accuracy. Decreasing bin size will make histograms less likely to match but increase match accuracy.

So far we have just looked at intensity histograms, but a general purpose histogram is of more interest. Frequency of color and texture in a region is a statistic that can be used to describe that region for use in a variety of image tasks, namely object matching and recognition. In this general purpose histogram, one has to define "bin centers": points in n-space which are the center of a hyper-rectangle of points which contribute to a particular bin in the histogram. In the intensity histogram case, we are dealing with a constrained example of this system in one dimension. Color will be an entity in (usually) three dimensions, whereas texture can reach a length of up to 48 dimensions. In both of these cases, the general histogram is approximated using a nearest neighbor system, where each bin center is a cluster mean found using a clustering technique (e.g. K-Means, Mixture of Multivariate Gaussians) on a training data set and each point contributes to the nearest neighbor mean's bin. The problem of distance in higher dimensionalities, more fully explored in Section 3.3, rears its ugly head in this situation as the data points for color and texture do not necessarily have Euclidean properties.

2.1.3 Gaussian Feature Description

An alternative to texton and color histograms is the use of a multivariate Gaussian to represent the statistics of a region. Instead of creating a discrete histogram, each pixel in the region is treated as a data point and then a multivariate Gaussian is fit which then models that data. This gives the mean and

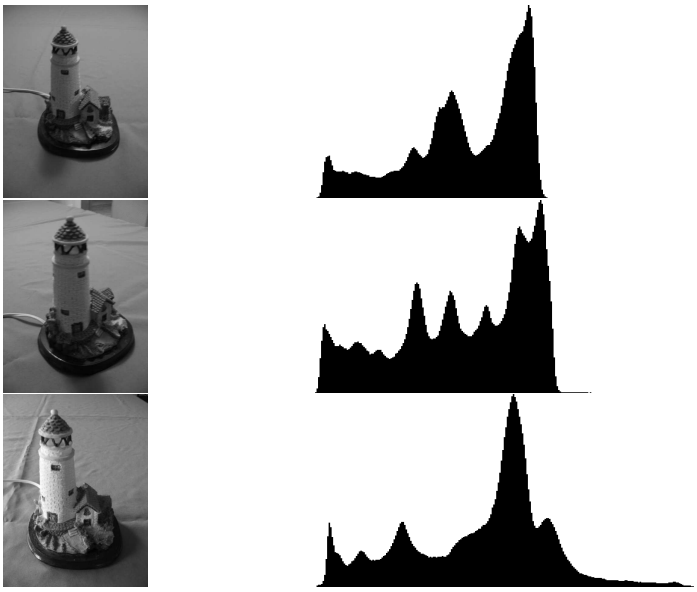

Figure 2.5: *Problems with Histograms* The three images shown are taken with the same camera, and all are of the same object. The top two were taken under artificial light from slightly different angles, and the bottom one was taken in natural light. The first thing to notice is that though these images are of the same object, the histograms do vary. Of particular interest is the way in which the peaks of the top two distributions are just offset enough that the immediate drop off would result in a lower matching score. Finally, note how the change in illumination shifts the weight of the histogram towards the lighter end of the spectrum in the lower image, an effect that can be remarkably pronounced with large changes in illumination.

covariance of the data, of which a full covariance matrix can prove an incredible rich descriptor. A full covariance is rarely used in this paradigm, however, due to the computational and storage costs involved with a diagonal matrix being used instead. This method of representing region data works particularly well in regions which have consistent statistics as it gives a tight distribution around a clear mean. Histograms can prove more useful in other cases, especially those in which there are two or more clear means as a Gaussian will only show an overall mean and indicate that the region is highly varied.

2.1.4 Color

The color of a object is a complex property, and storing it digitally is difficult to do accurately due to the fact that different imaging devices will produce different values for the same color, thus making device-independent storage only possible through calibration. Even if the values are being encoded accurately, however, the wavelength of light emitted from a surface changes based on a variety of environmental conditions such as the viewing angle or the color and strength of the light source(s), though the properties of the surface do not. Thus, it is possible to get two different color readings for the same point on a surface from one moment to the next. In this section we will look at various different methods of storing color in common use and how effective they are at dealing with this problem in relation to their ease of use and understandability.

RGB

The *RGB*, or *R*ed *G*reen *B*lue, color space is the simplest method of color data storage and by far the most widespread in computational applications. The standard method uses 8 bits per color resulting in a 24-bit representation with 0 indicating no amount of a particular color and 255 indicating the maximum amount of that color. For example, (0, 0, 0) is black and (255, 255, 255) is white, with red being (255, 0, 0). The intuition for *RGB* is based somewhat on the human vision system, in which three cones respond to yellow-green, green and blue light to varying degrees, mixing to create the various colors we perceive.

Such a system is said to use additive color, since three or more primary colors are combined to create a gamut of colors. It should be noted that *RGB* doesn't dictate the nature of the red, green and blue that are to be mixed, thus making it possible for the same *RGB* value to result in two completely different colors on two different devices, a significant problem if we are trying to learn color statistics for the purpose of computer vision. For a depiction of the RGB color space, refer to Figure 2.6, a cube in which each of the three dimensions represents values for red, green or blue. As can be seen somewhat in this depiction, the values for a particular color, such as "green", are spread over a large area of the cube as intensity varies. A more useful color space would be one which separates color from intensity.

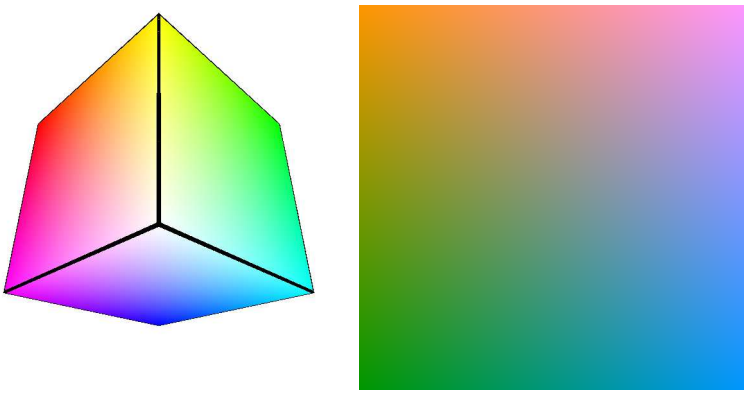

Figure 2.6: *RGB Color Cube and Slice* On the left is a rendering of the *RGB* Color Cube as shown from the perspective of the white corner (black lines added for clarification of edges). On the opposite side of the cube is the same view but with each of the colors darkening to black at the point. The slice is taken at Green = 150, with Red increasing horizontally to the right and blue increasing vertically upwards. Notice how any particular color is spread across large areas of the cube, both on the surface and internally.

HSL

HSL, or *H*ue *S*aturation *L*ightness, is perhaps the most intuitive of the color spaces. Each of the three elements corresponds to one of three aspects of a color: which color it is, how strongly it is that color, and how bright it is respectively. If *RGB* is a cube, then *HSL* is a double-cone as shown in 2.7, with Lightness on the vertical axis, and Hue and Saturation being angle and radius on the circular cross-section. For computer vision *HSL* can be very useful, as it separates out intensity from color, making it possible to store color as a distinct entity and analyze it independent of light source intensity. However, it suffers from the same problem as *RGB* in that the perceived distances between colors are not properly reflected by the geometric properties of the space.

CIELab

The *Commission Internationale d'Eclairage* (International Commission on Illumination) attempted to create a complete, device-independent color space which mimics the perceived distances between colors determined by the human brain. The result was the *Lab* color space, as depicted in Figure 2.8. It is device-independent based on a known white-point (the set color of white under a known illuminant) and is able to encode all colors visible by the human eye (and some which are outside the range as well.) The L represents *L*uminance, a denotes where on a scale from red to green the color lies, and b similarly with blue to yellow. *Lab* is by far the least intuitive of the color spaces, though its completeness, device-independence and Euclidean properties in respect to

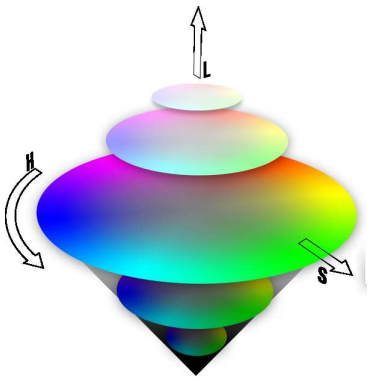

Figure 2.7: *HSL Double Cone* The *HSL* double cone. This structure captures two main human intuitions about color. First, as Lightness increases to its maximum or decreases to zero, color (Hue) and Saturation become decreasingly important and harder to distinguish. Second, color is separated and forms a color wheel in which similar colors are close together.

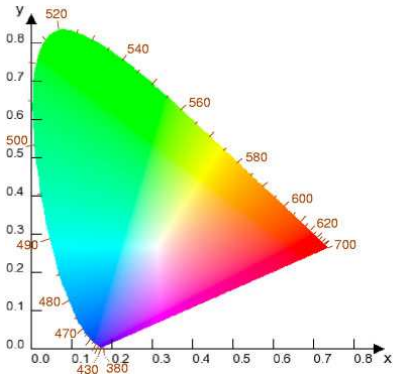

Figure 2.8: *Lab Gamut* Though this representation is sub-optimal (the actual color space can't be represented using printed colors alone, what is shown is an approximation) this shows the range of colors which can be represented using the *Lab* color space, shown here as the hues at maximum luminance.

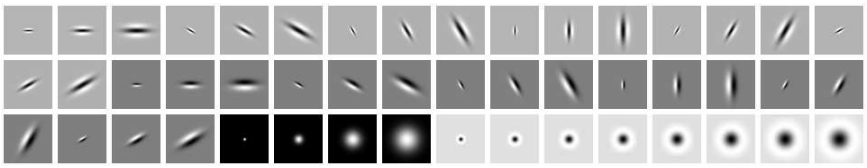

Figure 2.9: *Leung Malik Filter Bank* The Leung Malik filter bank contains an extensive array of filters. First there are edge and bar filters (second derivatives of Gaussians with even and odd phase respectively) at 3 scales and 6 orientations. Secondly, there are 4 Gaussians at 4 scales. Finally, 8 Laplacians of a Gaussian filters at 8 different scales.

human perceived color distance make it very useful for vision tasks.

2.1.5 Texture

Texture is a difficult quantity to measure. In its common sense it denotes the way a surface feels. In vision, however, its tactile sense is eschewed in favor of a visual one, namely a distinctive pattern or patterns which are a result of surface properties. These patterns can repeat, though often they are fractal or random in nature, making their detection and storage problematic. In this section we explore the main systems of storing texture through the use of a representative example of each and how they can all be improved through the innovation of textons.

Filter Banks

In [60], Malik and Perona proposed a method of texture extraction based on biological foundations. There exist cells in the eye known as "simple cells" which have receptive fields that are restricted to small regions of space and are highly structured [9, 62]. These cells were modelled originally using Gabor functions [35] to some success both in the understanding of biological visual systems [49] and in computer vision [92, 31]. Malik and Perona proposed an alternative approximation to the simple cell response field based on the fact they they appeared to come in two different varieties: oriented, and non-oriented. To mimic this, they designed an array of filters (otherwise known as a filter bank) which model the two types of simple cells at varying scales and responding at each pixel in an image. These preliminary filters were modified experimentally until a finalized version was put forth by Leung and Malik in [56]. This filter bank consists of 8 Laplacian of Gaussian filters and 4 Gaussian filters at different scales to provided non-oriented responses, and 36 oriented filters at 6 different angles, 3 different scales, and 2 different phases, as seen in Figure 2.9. The two phases of oriented filters are first and second derivatives of Gaussians in one direction and Gaussians in the other, and thus detect edges or bars respectively along their main axes. These types of filters are called *Gabor-like* filters, as they approximate Gabor functions. Though it is fairly comprehensive in the various

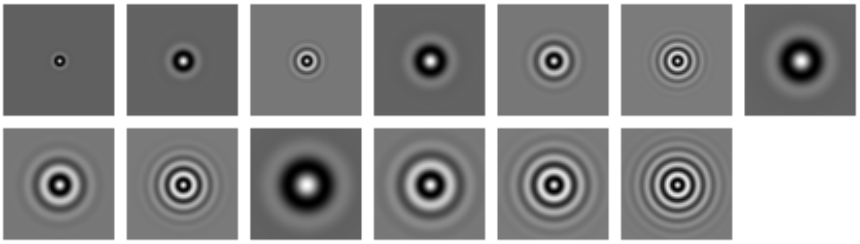

Figure 2.10: *Schmid Filter Bank* This filter bank is rotationally invariant, unlike the Leung Malik filter bank. It consists of 13 isotropic, "Gabor-like" filters, making it substantially smaller than the Leung Malik filter bank as well.

kinds of texture features it can extract, this representation has its limitations. The oriented filters have to be present at various orientations, however since the number of orientations is limited they discretize the rotational space for edges and bars in possibly damaging ways. In addition, with 6 orientations per scale, they limit the total number of scales can be used due to the prohibitive size of the eventual textual descriptor.

Gabor-Like Filters

The Gabor filter function is designed to mimic the response fields of the simple cells using a plane wave restricted by a Gaussian, a model motivated by the results of [49]. The suggested biological reality of paired even and odd symmetry simple cells in the eye [72], is incorporated by a cosine function and sine function as the chosen wave function. The two waves can be combined in complex notation as

$$e^{ikx} = \cos(kx) + i\sin(kx) \tag{2.5}$$

where the real part corresponds to the cell with even symmetry and the imaginary part to the odd-symmetry cell. From this, a biologically motivated filter can be created [24]

$$p_k(x) = \frac{k^2}{\sigma^2} e^{-\frac{k^2}{2\sigma^2}x^2} \left(e^{ikx} - e^{-\frac{\sigma^2}{2}} \right) \parallel p_k(x) \parallel^2 \tilde{k}^2 \tag{2.6}$$

where k is the central frequency of the filter. However, this complex equation is often approximated. One such approximation was suggested by Schmid in [82]. Her filter bank consists of 13 filters of the form

$$F(r, \sigma, \tau) = F_0(\sigma, \tau) + \cos\left(\frac{\pi\tau r}{\sigma}\right) e^{\frac{-r^2}{2\sigma^2}}$$

with $F_0(\sigma, \tau)$ added to obtain a zero DC component [2]. The specific values used for σ and τ in Schmid's proposal are (2, 1), (4, 1), (4, 2), (6, 1), (6, 2), (8, 1),

[2] "DC component" is a term borrowed from electronics and is defined as the average of a signal over time. In this context, the DC component over the filter is found and then $F_0(\sigma, \tau)$ is set to its negative value, the result being that the new signal has a zero DC component.

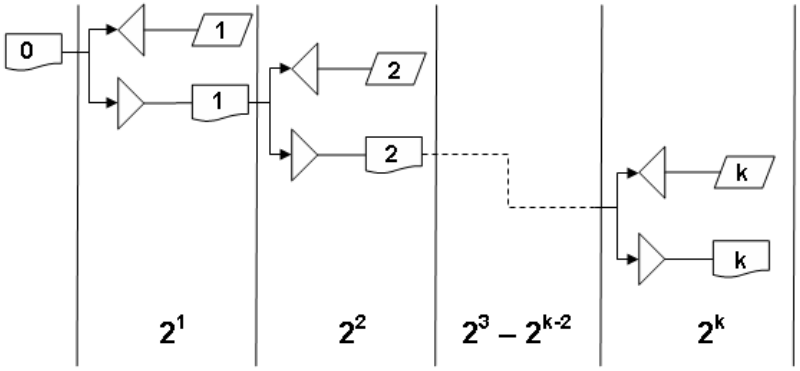

Figure 2.11: *Wavelet Tree* This is a depiction of the iterative discrete wavelet transform process. The highpass filter is denoted by the triangle with its base on the right, the lowpass filter by the triangle with its base on the left. At each iteration, the downsampled result of the low-pass filter is used to create the next level.

(8, 2), (8, 3), (10, 1), (10, 2), (10, 3) and (10, 4), and the filter bank is shown in Figure 2.10. This filter bank has the advantage of being less unwieldy than the Leung and Malik filters, however they have no ability to distinguish between different edge orientations (as they are radially symmetric) though this does give them a certain robustness to rotation. Indeed, Varma and Zisserman found in [95] that rotationally invariant filters did not show reduced performance and even performed better than other filter banks on certain data.

Wavelets

The Gabor filter bank methods and their approximations all share a common problem in that they are irreversible transformations. This, in addition to the fact that they are not mutually orthoganal results in a large amount of shared data between various filters. An alternative filter bank for texture description was proposed by Unser in [93] which avoided these problems through the use of a discrete wavelet frame (DWF) (specifically an over-complete version known as the discrete wavelet frame as detailed in [78, 77].) Wavelets are a class of functions used to localize a given function in space and scaling. A family of wavelets is formed by construction from the "mother wavelet", denoted $\psi(x)$, which is confined in a finite interval. The "daughter wavelets" $\psi^{a,b}(x)$ are then built either by a translation b and/or a contraction a according to the function

$$\psi^{a,b}(x) = \frac{1}{\sqrt{|a|}}\psi(\frac{x-b}{a}) \tag{2.7}$$

A very thorough introduction to wavelets and their uses in signal processing is given by Rioul in [78]. Wavelets have the advantage of being a reversible transformation and that one can construct a family such that it provides an

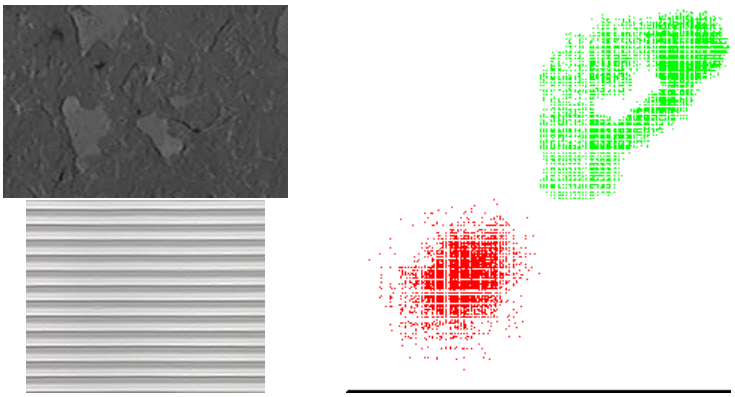

Figure 2.12: *Neighborhood information* Shown here is a demonstration of the fact that there is enough information in the neighborhood of a pixel to separate two textures from each other. Each datapoint is plotted as the value at I(x-1,y-1) on the x-axis against I(x+1,y+1) on the y-axis, with the red points coming from the texture on top and the green points coming from the texture on the bottom. A clear dividing plane obviously exists between these textures in this simple projection.

orthogonal basis for a signal. They essentially give the ability in texture analysis to capture the large scale repetitions of a texture and the inner patterns found within. This is done in the discrete case by way of a highpass filter $h(n)$ and a complementary lowpass filter $g(n)$ related by the equation

$$h(L-1-n) = (-1)^n g(n) \tag{2.8}$$

where L is the length of the filter (necessarily even). The filters are applied and down-sampled by 2 to produce outputs

$$y_0(k) = \sum_n x(n)g(-n+2k) \tag{2.9}$$

$$y_1(k) = \sum_n x(n)h(-n+2k) \tag{2.10}$$

where x is the original signal. The original signal can be retrieved (given filters with perfect reconstruction) by upsampling y_0 and y_1 and filtering by the reverse filters $g'(n)$ and $h'(n)$ respectively. To achieve greater resolution, y_0 is then subsequently passed through $h(n)$ and $g(n)$ and down-sampled again. This process is repeated to the desired level of resolution, with each y_1 in the iteration being the difference between the previous lowband portion and the current one (a passband.) A representation of this process can be seen in Figure 2.11. Wavelets have been used extensively in texture classification and synthesis [30, 66, 73, 93, 25, 100] since their introduction by Mallat's pioneering work in [61].

Patches

The M4 and M8 filter banks were described in [95] by Varma and Zisserman as compact descriptors in a 4 and 8 dimensions respectively. Their aim was to provide descriptors which where as effective as those currently in use but without the curse of dimensionality that using those filter banks induced. They were designed to interact with a larger system for classification of textures and performed better than most current systems at the time. Interestingly, both were soon discarded in favor neighborhood patches, a shockingly simple representation which called into question the dominance of filter banks in the field. The move was motivated by work done by Efros and Leung [28] in which they synthesized new textures using only the local neighborhood of pixels. In [94], Varma and Zisserman published a new paper using their system from [95] with only one change: the $N \times N$ neighborhood of a pixel was used as the texture response (row-sorted in N^2 dimensions) in place of the filter bank response from the M8 descriptor. The results were surprising; with only a 3×3 neighborhood they were able to match their previous results, with improvements as neighborhood size increased. Their explanation for this is that texture can be to a large extent considered realizations of a Markov Random Field (MRF). Formally, for a MRF

$$p\left(I(x_c)|I(x), \forall x \neq x_c\right) = p\left(I(x_c)|I(x), x \in \mathcal{N}(x_c)\right) \tag{2.11}$$

where x_c is a pixel in the image and $\mathcal{N}(x_c)$ is its neighborhood. In words, the value of a pixel is the direct result of the values of its neighbors. It seems unbelievable that the neighborhood of a pixel can provide enough information to separate various classes of texture, however as can be seen in Figure 2.12 where the values of a pixel and its neighbor 2 pixels away diagonally are plotted against each other, neighborhood information is indeed sufficient to create boundaries between two texture classes. While this method doesn't have a biological basis, its results speak for themselves, yet the use of the neighborhood comes with a price. As some textures appear at low frequency, detection at various scales can be necessary, especially when dealing with natural images. Whereas scale is incorporated automatically into filter banks, without modification to the descriptor scale can only be achieved through increased neighborhood sizes, which result in quadratic growth for storage and computation.

Textons

Textons as a concept represent the basic buildings blocks of texture. In much the same way that many different colors can be mixed from a collection of primary colors, texture can be built from the mixing in proportion of various textons. The concept was originally put forth by Julesz in [50], but fell into disuse due to a lack of a concrete description of the entity. It was brought into vogue by Leung and Malik in [55], where they gave such a representation. In that paper, they took the responses from their filter bank (as described in Section 2.1.5) for an image and quantized them using K-means clustering to receive K textons. The K-means clustering algorithm has been extensively studied

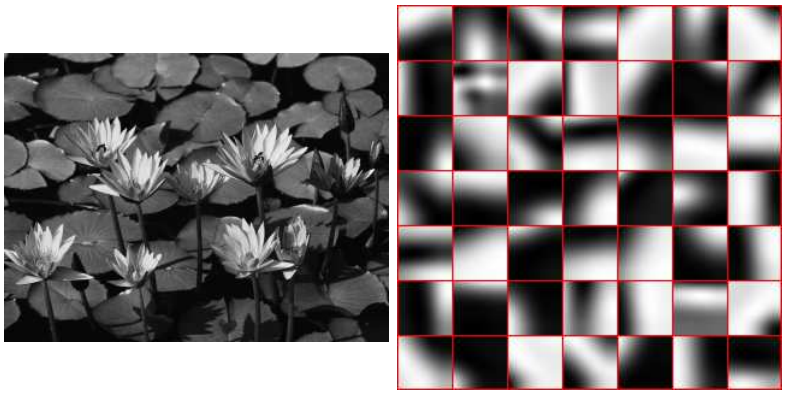

Figure 2.13: *Textons* The patches on the right are textons extracted from the image on the left using K-means clustering over 5x5 ZNCC patches extracted at each pixel. Note how basic image structures, such as oriented edges, bars and corners, are represented.

and many variations exist, however the most commonly used version (known as Lloyd's algorithm [58]) is to associate every vector with the $k \in K$ closest to it, and then to set k equal to the mean centroid of its vectors until a stopping condition has been reached. These centers are then known as the appearance vectors $\{c_0, \ldots, c_K\}$, and function as textons. What the textons encode can be understood more clearly by constructing local image patches P_k from each c_k. By constructing a filter matrix F from concatenating the row vector forms of the filter bank members and finding its pseudo-inverse F^{-1}, one can create a patch a patch filter $P_k = F^{-1}c_k$ as described in [48]. An example of an image and some of its textons represented as patches can be seen in Figure 2.13.

Leung and Malik used K-means as mentioned to extract a group of textons for each image. Then, to define an "universal" set of textons, they took all the texton responses from every image in their dataset and reduced them by means of culling those with little support and merging others until a subset of 100 was reached. K-means as a method can prove troublesome, especially as the distance calculations are being performed in a high-dimensional space (in this case 48 dimensions) and between objects that do not necessarily have Euclidean properties. The inefficiencies due to dimensionality can be greatly reduced through the use of high-dimensional K-means algorithms [51, 1] however the distance problem is not easily surmountable. In [37] Georgescu uses the mean-shift algorithm to extract textons from the filter bank responses from [56], [82] and [95] with encouraging results. Another method is to use principle components analysis to extract textons as done by Cula and Dana in [23].

Once a set of universal textons have been extracted texton histograms can be constructed for an image, where each bin corresponds to a particular texton. The filter bank response at each pixel in the image contributes to the bin of its closest texton, the result being a histogram which describes the image in terms of which textons compose it in which quantities. These texton histograms are

then used to describe a class of texture, allowing then for the classification of new texture through histogram matching, a problem discussed in more detail in Section 3.3.

2.1.6 Shape

Shape in particular only has significance if considered for an image region. It isn't immediately clear how one can describe the shape of a region in a way which is robust to affine transformation (i.e. scaling, shearing, rotation, panning and/or translation.) While no truly effective shape descriptor has been devised there are several techniques which give a workable encoding of shape for specific tasks.

Global Shape Statistics

The simplest shape representations are measurements of global shape characteristics as opposed to representations of any particular shape, and while they cannot be used to reconstruct the original shape they do encode useful information, and have the advantage of being fast and easy to extract. A survey of such techniques is performed in by Gabbur [34], describing in addition to obvious statistics such as the normalized region area quantities such as the first moment, compactness and convexity. The first moment is measured as the normalized (with respect to the image dimensions) variance of the pixel distance to the region centroid, and measures how spread out the region is. Compactness, calculated as $\frac{A}{B^2}$ where A is the region area and B the boundary length, measures how much empty internal space is contained in the region. Convexity, calculated as $\frac{A}{A_{CH}}$ where A_{CH} is the area of the convex hull, measures whether the region is articulated. These rough descriptors are useful to add global shape information to an existing feature set, but do not encode much useful shape information of themselves.

Fourier Outlines

The problem of defining shape consists of the fact that the outline of an object can vary, with many of those variations being affine in nature. Creating a representation for shape that is affine invariant, therefore, was the initial objective of research in this area. The first attempts at a shape descriptor concentrated on the Fourier transform of an outline [71, 98]. Descriptions of the process and its modern uses can be found in [2], with a comparison of various methods performed by Zhang in [99]. For the purpose of illustration I will describe in brief the system detailed by Gabbur in [34]. Regardless of the method used, to extract the Fourier outline from an image of an object a contour for the object region must be determined, usually by a segmentation algorithm to determine a region boundary or from the output of an edge detector. Then, starting at an arbitrary point on the shape the distance from each point on the outline to the region centroid is recorded to create a distance function, $R(s)$, which

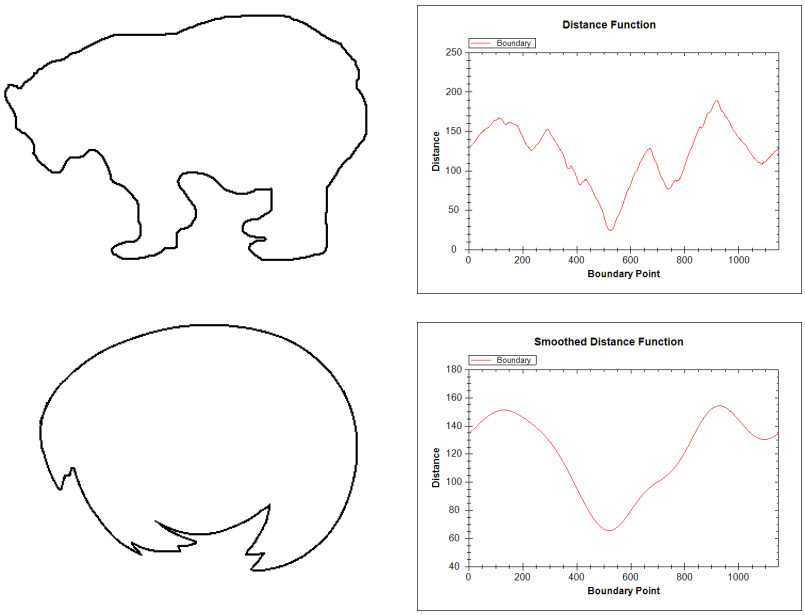

Figure 2.14: *The Distance Function* The top left image shows the original outline and its distance function. The bottom image shows the outline after smoothing and its accompanying distance function.

describes the shape as shown in Figure 2.14. This function, as it is calculated with respect to the centroid, is translation invariant. $R(s)$ is then smoothed using using a Gaussian resulting in the smoothed distance function $R_{smooth}(s)$, also seen in Figure 2.14. This function is then normalized by the mean distance to the center to obtain $R_{smooth,n}(s)$. The final function R_f is achieved by sampling $R_{smooth,n}(s)$ M times (Gabbur uses M=30) to obtain scale invariance. The points used are taken as the center of a N/M sized window of $R_f(s)$. To ensure that the center pixel contains information about the entire window as well as its neighboring windows a value of $\frac{2N}{M}$ is used for the σ of the Gaussian smoothing function. The T-point discrete Fourier transform of $R_F(s)$

$$\mathcal{R}(k) = \sum_{s=0}^{M-1} R_f(s) e^{-j\frac{2\pi ks}{T}} \tag{2.12}$$

$$k = 0, 1, \ldots, T-1 \tag{2.13}$$

$$j = \sqrt{-1} \tag{2.14}$$

where $T \geq M$ to avoid aliasing. If we only use the magnitude of the signal $|\mathcal{R}(k)| = \sqrt{r_{real}^2 + r_{imag}^2}$ we gain several properties important for a shape descriptor. The use of only the magnitude makes the starting point on the bound-

ary unimportant, giving rotational invariance. The descriptor now becomes

$$\mathbf{F} = \left[\frac{|\mathcal{R}(1)|}{|\mathcal{R}(0)|}, \frac{|\mathcal{R}(2)|}{|\mathcal{R}(0)|}, \ldots, \frac{|\mathcal{R}(T/2)|}{|\mathcal{R}(0)|} \right]. \tag{2.15}$$

Normalization by the zero-frequency component makes the descriptor scale-invariant, and due to the fact that the magnitude has the property $\| \mathcal{R}(k) \| = \| \mathcal{R}(T-k) \|$ we need only encode the first $T/2$ members. The Fourier descriptor for the same shape under various transformations is depicted in Figure 2.15, showing its invariance. It should be noted that it does not in fact perform well in the case of non-affine transformations (such as flipping, shown in the middle.) Finally, extracting the outline repeatedly and accurately in real images depends very much on segmentation, with the best results only occuring for perfect segmentations of an object.

Statistical Shape Description

A statistical method for shape description has been one of the ongoing research objectives of Cootes and Taylor. In [20] and [21] they describe a shape descriptor which consists of the mean shape and its variance for an object. The advantage of a their shape descriptor is that new shapes can be assigned a class likelihood, allowing them to be incorporated into a robust statistical framework. It is also a generative model, able to create new shapes by sampling from the shape distribution. The calculation of the mean and variance is achieved by use of a training set of aligned shapes. For every shape in the image a set of training points x is chosen. These points must be placed by hand at distinctive areas on the contour. The images in the training set are then aligned using the following algorithm:

1. Translate each example so that its center of gravity is at the origin.
2. Choose one example as an initial estimate of the mean shape and scale so that $\| \bar{x} \| = 1$.
3. Record the first estimate as $\bar{x}_0$ to define the default reference frame.
4. Align all the shapes with the current estimate of the mean shape.
5. Re-estimate mean from aligned shapes.
6. Apply constraints on the current estimate of the mean by aligning it with $\bar{x}_0$ and scaling so that $\| x \| = 1$.
7. If not converged, return to **4**.

where convergence is declared if the estimate of the mean does not change significantly after an iteration. Once the training set has been aligned, the statistical model is extracted in three steps:

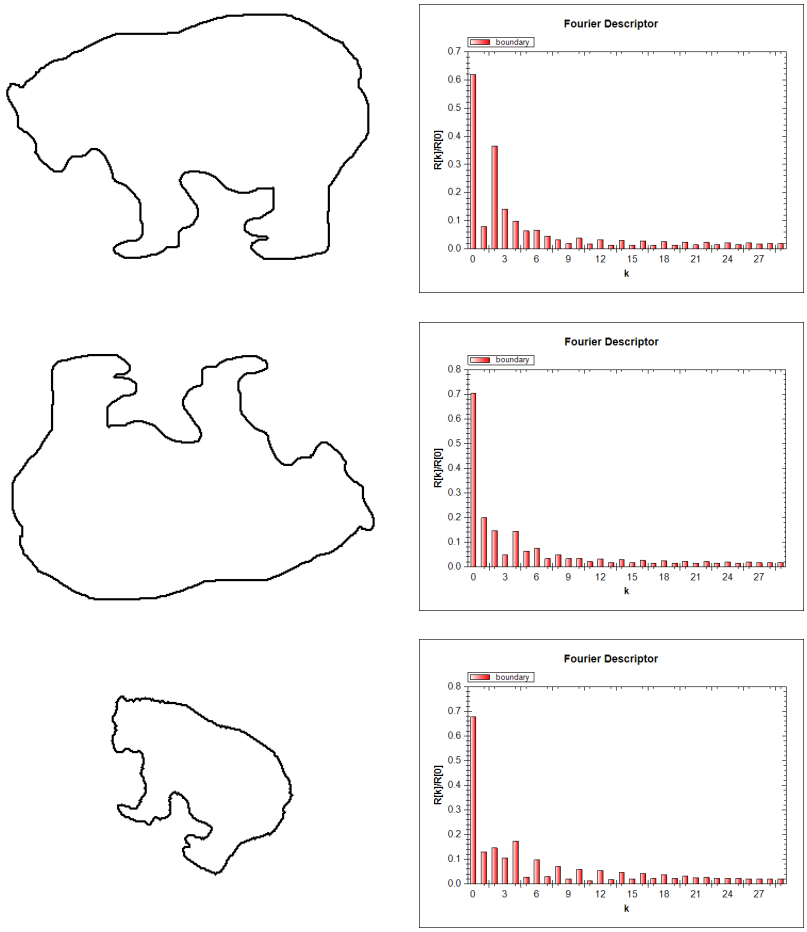

Figure 2.15: *Fourier Outlines* The same outline is shown under a vertical and horizontal flip, and then a scale and rotation in the two images. The Fourier outline does not retain its invariance well under rotation, though the scale and translation in the third image has little effect.

1. Find the mean of the data,

$$\bar{x} = \frac{1}{s}\sum_{i=1}^{s} x_i \tag{2.16}$$

2. Compute the covariance of the data,

$$S = \frac{1}{s-1}\sum_{i=1}^{s}(x_i - \bar{x})(x_i - \bar{s})^T \tag{2.17}$$

3. Compute the eigenvectors ϕ_i and the corresponding eigenvalues λ_i of $\mathbf{S}$ (sorted so that $\lambda_i \geq \lambda_{i+1}$.)

If one creates a set Φ of the most influential eigenvectors then any shape in the training set can be generated using the equation $x_i = \bar{x} + \Phi\mathbf{b}$ where $\mathbf{b} = \Phi^{-1}(x_i - \bar{x}$. New, unseen shapes can be generated by changing the values of $\mathbf{b}$, with λ_i representing the variance of the i^th element of $\mathbf{b}$. To match a candidate shape to the model, it must be first labelled in the same way as those in the training set. Then, a transform between the mean shape and the candidate shape is found, giving values for $\mathbf{b}$. As a last step the probability for the shape belonging to the class can be determined, typically by modelling the class as a multivariate Gaussian over $\mathbf{b}$.

While this method of shape description both generates new shapes by sampling from a shape distribution and provides a statistical framework within which to classify shapes, it has a major drawback. The fact that all shapes, including the test shapes, must be hand-labelled with the feature points makes the algorithm essentially unusable with the current generation of segmentation and feature point systems. While it remains an interesting model, a way to reliably rediscover the same points in varying shapes and match them accordingly must be found before it can be used effectively in real-world situations.

Shape Context

So far a descriptor which describes the shape while being robust to transformation and the contour changes of a real-world object has been elusive. Belongie and Malik provide the proverbial sledgehammer to the problem, however, by way of the shape context descriptor [8]. For each set of points on the contour (the choice of which points does not matter, one of the strengths of the technique) a log-polar histogram is calculated for the vectors between each point and the remainder of the set. In notation $h_i(k) = \#\{q \neq p_i : (q - p_i) \in \text{bin}(k)\}$ where p and q are points on the outline. This histogram, calculated for 5 length bins and 12 orientation bins at each point, is the descriptor. To match two images, a thin plate spline transform is found between the two shapes and then the shape context distance between two images $D_{SC}(P, Q)$ is calculated as

$$D_{SC}(P,Q) = \frac{1}{n}\sum_{p\in P} \arg\min_{q\in Q} C\left(p, T(q)\right) + \frac{1}{m}\sum_{q\in Q} \arg\min_{p\in P} C\left(p, T(q)\right) \tag{2.18}$$

where T is the transformation, $n = |P|$ and $m = |Q|$. The advantages of shape context are that it is robust to transformation and richly descriptive. The obvious disadvantage is that the descriptor for a particular shape is $R \times D \times N$ elements long where R is the number of log length bins, D is the number of directional bins and N is the number of sample points. This is an extravagantly sized descriptor, and while it does solve many shape classification and detection problems very well it is impractical with current storage and processing capabilities.

2.2 Local Features

So far we have just explored global features, and while they do give important information about images and image regions they have many inherent problems. First, finding the region in one image which corresponds with a region in another image is tricky, especially if the two images are views of the same scene from different viewing angles. Secondly, one is unable infer anything about object structure from a global feature histogram or similar descriptor. As such, a separate set of image features, local features, provide a way to localize image data. Once the location of a feature descriptor is known, tasks like finding locations in a new image which share that description or reconstructing an object become possible. Though localized descriptors use the same kinds of elements described in the previous section, they are centered around points in the image which are the direct result of scene structure and as such will appear in many different views of a scene. Before a local feature can be extracted, however, stable points of interest in the image must be found.

2.2.1 Interest Points

The three main kinds of interest points are edges, corners, and blobs. Edges mark the boundaries between different areas in the image, for example areas of different brightness levels, or texture statistics. Corners are found at the areas where several regions intersect, and blobs are found in the stable centers of large regions. Of the three, edges are easiest to extract, corners are the most numerous and blobs the most stable. All three have biological plausibility, i.e. there is evidence that there are structures within mammalian vision systems that perform all three tasks. This section discusses their properties and methods of extracting them in turn.

Edges

Canny edge detection [17] is a efficient process that produces a binary edge image (in which every point is labelled as an edge or otherwise) or an edge list from an input intensity image and is by far the most used edge detection scheme in modern computer vision. Edges are located where intensity values in the two-dimensional image function undergo a sharp change from one state

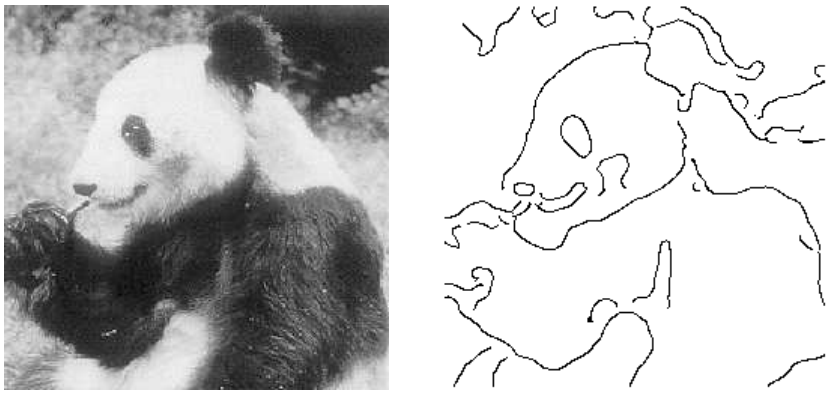

Figure 2.16: *Canny Edge Detection* Shown here is an image and its Canny edge map at $\sigma = 3, \tau_1 = .3, \tau_2 = .7$ where τ_1 and τ_2 are the thresholds used for hysteresis. Notice the clean, connected edges and the complete lack of noisy, unnecessary edges in the image.

to another, such as from a white square to a black background. These points are the local maximums of the derivative of the image, otherwise known as the gradient and denoted by the ∇ symbol. In order to take the derivative of an image, which is a signal and thus prone to noise which results in areas of high frequency that will produce false edges, we must first use a lowpass filter (such as the Gaussian function) to smooth out the image, resulting in only the main gradients being retained in the signal and thus the true edges being detected. The choice of σ for the filter has a large effect on the resulting detection. Large values for σ result in only stronger edges being detected (such as those between foreground objects and the background), while small values retain weaker edges as well (such as interior edges.) Next, the gradient must be taken. While the proper way in which to do this is to convolve the image with the first derivative of a Gaussian in the two primary directions, in practice it is often approximated using a kernel convolution such as the first difference or the Sobel operator. Once the gradients in both directions are known, the magnitude of the edge can be found as the magnitude of the gradient vector.

The gradient magnitude image must be thresholded in some way to be of use for edge detection. The next step in the Canny process, non-maximal suppression, removes all points which aren't the local maximum of their 3×3 neighborhood. These local maxima are set to 1, all other points are set to 0. This new binary image is too sparse, and has removed many valid edge pixels. The next step, hysteresis, is the true contribution of the Canny method. In two-threshold hysteresis two thresholds (usually the higher is three times the lower) are predetermined and then at every pixel, if its gradient is above the higher threshold and it is set to 1 in the binary image, then any pixels along the edge direction [3] which have a value higher than the lower threshold are also

[3]The edge direction is the the normal to the gradient direction

set to 1 in the output image. This has the effect of filling in gaps in edges, a problem known as "streaking" and common to most edge detection schemes. The end result is a binary image showing the clear edges, such as that shown in Figure 2.16.

While the Canny method is effective, it does have problems. Since it only looks at brightness, the edges between a dark object and its shadow or between two objects of similar intensity are weak. A more effective way to detect edges is the Compass operator, first introduced by Ruzon and Tomasi in [80] for use with color and extended for texture by Maxwell and Brubacker [63]. The compass operator looks at the statistics of the entire region as opposed to the local gradient function to find edges. It is composed of a circle of diameter 3σ around a point, where σ is a scale variable. This circle is then bisected N times at regular intervals and a weighted histogram computed for both sides (Ruzon and Tomasi use a Raleigh distribution in polar coordinates to weight the contribution of a pixel, i.e. $f(r,\theta) = r\exp(-r^2/2\sigma^2)$). The distance between these histograms is calculated using EMD as described in Section 3.3.5 with an exponential ground distance between bin centers described by the equation

$$D(x,y) = 1 - e^{\frac{-E(x,y)}{\gamma}} \tag{2.19}$$

where $E(x,y)$ is the Euclidean distance between points x and y. Localization of the exact edge orientation is found by fitting a curve to the maximum angle and its two neighbors and taking its derivative.

Though the compass operator is a powerful new concept, its original implementation using EMD was frightfully slow, with a runtime of 33 minutes on one 769×512 image [80]. Maxwell and Brubacker proposed an alternate formulation in [63] which ran in 30 seconds for a 340×240 image using texture (and thus in higher dimensions) which makes begins to make the compass operator practical for use in edge detection. They achieve this by using the more efficient dynamic time warping [75] as an alternative to EMD, and by jittering the application of the compass operator. Jittering is a practice commonly found in ray tracing, where the image is sampled by only applying the compass operator to one randomly selected pixel in each 4×4 block and then applying a 4-connected box filter that ignores unset pixels. Figure 2.17 shows our implementation of the compass operator using texton histograms and DTW for a particular texture edge, with the DTW graph for each bin shown and the actual edge highlighted in red.

Corners

Corners are areas in the image in which the gradient in all directions is large. At an edge, the gradient in the direction normal to the edge is large but there is no gradient in the direction of the edge itself. At a corner, however no matter what direction the gradient is measured it is high. Similarly, if the gradient shows no change in any direction that point is in a uniform area. In [40], Harris and Stephens describe a method based on the Moravec corner detector [67] which uses this principle to detect corners.

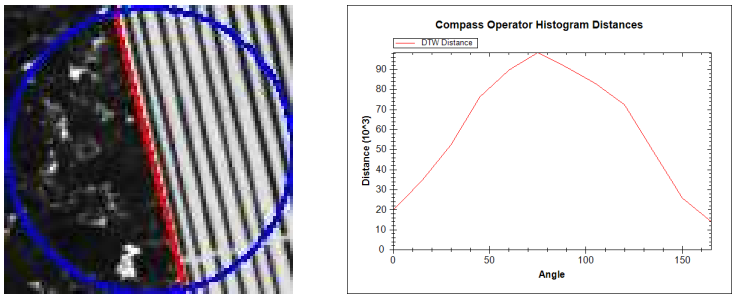

Figure 2.17: *Compass Operator* The graph on the right shows Dynamic Time Warp value for 12 orientations sampled by the compass operator shown on the left, centered on the patch. The blue circle denotes the area sampled, and the red line is drawn at the interpolated edge. Distances on the right are DTW between two texton histograms, with the textons used being those extracted in Figure 2.13.

Denote the smoothed image brightness function as $I(x, y)$. Of interest for corner detection is the matrix

$$H = \nabla I \nabla I^T = \begin{bmatrix} (\frac{\partial I}{\partial x})^2 & (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) \\ (\frac{\partial I}{\partial x})(\frac{\partial I}{\partial y}) & (\frac{\partial I}{\partial y})^2 \end{bmatrix} \tag{2.20}$$

or more specifically, its smoothed version

$$\hat{H} = \begin{bmatrix} \hat{I}_x^2 & \hat{I}_x \hat{I}_y \\ \hat{I}_x \hat{I}_y & \hat{I}_y^2 \end{bmatrix} \tag{2.21}$$

where $\nabla \hat{I}_x = G * \nabla I$, G is a two-dimensional Gaussian smoothing kernel and $\nabla I = [I_x, I_y] = [\frac{\partial I}{\partial x}, \frac{\partial I}{\partial x}]$. This matrix describes the gradient characteristics in the window of the smoothing kernel G. Its eigenvectors ϕ_0 and ϕ_1 describe the main directions of the gradient with the strength of those directions being directly related to their corresponding eigenvalues λ_0 and λ_1. If both λ_0 and λ_1 are large, the area is a corner, if λ_0 dominates the area is an edge, and if both are negligible the area is uniform. As such, this method can be used for both corner and edge detection. The σ used for G determines the scale of the features detected by dictating the size of the neighborhood inspected.

This basic method, though improved by Shi and Tomasi [86] and Noble [69], has remained one of the best corner detectors in use. An example of the corner response on some sample images is shown in Figure 2.18, with detections on the first image showing its theoretical accuracy and on the second showing its performance on real-world images.

The Compass operator from [80], in addition to being able to detect edges, can also be used as a corner detector. Instead of bisecting the circle as for edges, wedges of width β at various orientations θ are cut out of the circle and the difference between the wedge and the remainder of the circle calculated using partial histogram matching with EMD. Once potential corners have been

Figure 2.18: *Harris Corners* This image displays Harris corner detection, with the 200 maximal corners from the image superimposed in red. Notice how the detector fires not only at corners but also at areas of active texture (e.g. the crinkled bag, the tangled wires.)

found they are tested to see whether the edges that would form them exist and are aligned with the angle of the corner. Once these conditions have been met, only plausible corners remain (i.e. ones which connect edges in the image.)

Difference of Gaussians

The Difference of Gaussian (DoG) operator has long been used as an approximation to the Laplacian of a Gaussian to find zero crossings in the gradient of an image for edge detection. It can also be used to find very stable interest points in the center of stable uniform regions [64], and is considered to be a biologically plausible model for the method used by the mammalian brains for this purpose [97]. As shown in Figure 2.19, the difference of two or three Gaussians approximates the Laplacian of a Gaussian, and the response of the operator for an image will have a value of zero at edges. The maxima and minima of the response are the stable points of interest.

Lowe uses the DoG detector as the initial step of his effective SIFT feature point scheme [59]. In his system he first calculates a Gaussian pyramid by convolving an image with a Gaussian of $\sigma = 2^{\frac{1}{L}}\,\mathrm{L} + 3$ times and then subsamples the Lth of these images by 2 to begin the next octave. Repeating this O times, this achieves $L + 3$ images in each of O octaves. Within each octave he then calculates $L + 2$ difference images, subtracting the $i + 1$th image from the ith image for $i = 0$ to $i = L + 2$. In each of these images he tests to see if a maximum or minimum in one scale is also the maximum and minimum in the adjacent scales. Once this has been determined to be true, he localizes the point by finding the maximum of a translated Taylor approximation of the scale space function as described in [15] to obtain a series of stable interest points

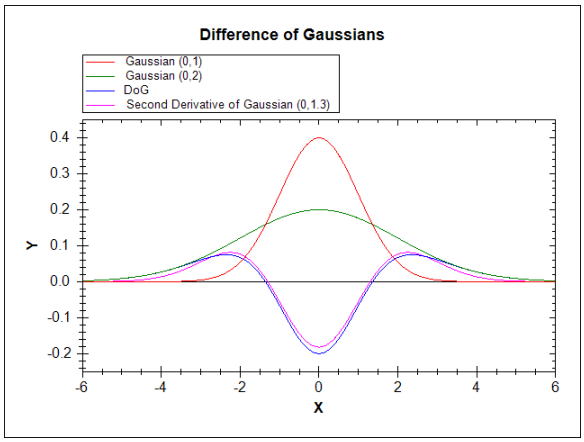

Figure 2.19: *Difference of Gaussians* Here are two Gaussians, their difference, and the second derivative of a Gaussian shown on the same graph. Note how the closely the difference approximates the second derivative.

at different scales. For a more complete description of this process see Section 3.5.1.

2.2.2 The Patch Descriptor

The simplest way to describe the area around an interest point is simply to store the intensity values in a patch around the point and compare them directly against other patches to find a match. This method isn't very robust to changes in intensity, however. We can fix this by centering the values in the patch at 0, meaning that we subtract the mean from every element in the patch to give it a zero mean. While invariant to changes in intensity this representation is still vulnerable to changes in contrast, so we divide each element by the standard deviation. This results in a patch that has zero mean and unit variance and which is the optimal way to use the pixel intensities alone to describe a local feature, being invariant to changes in global intensity and contrast.

2.2.3 The SIFT Descriptor

In a recent evaluation of local descriptors [65] the SIFT descriptor performed significantly better than all other methods. SIFT stands for Scale-Invariant Feature Transform and is described in detail by David Lowe in [59]. It generates a descriptor based on the overall intensity gradient at each pixel. Since the gradient is independent of offset there is automatic invariance to brightness changes, and by binning into a series of histograms some limited invariance to misalignment and warping is also introduced. A large $N \times N$ patch (Lowe uses $N = 16$) of the surrounding image forms the feature neighborhood and is extracted at the scale of the interest point. To gain invariance to rotation, an

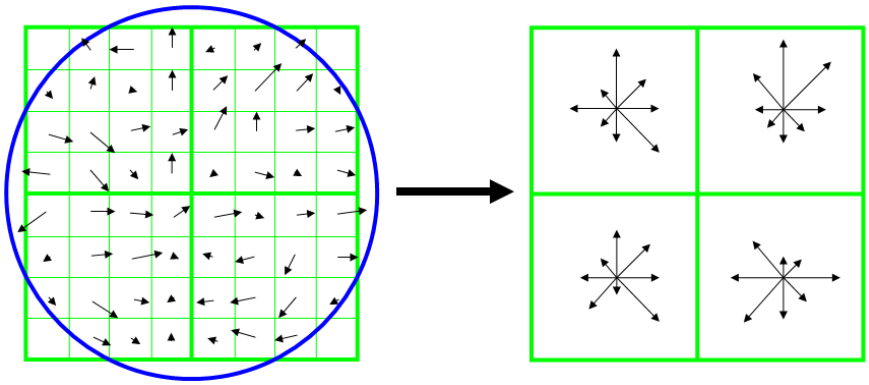

Figure 2.20: *The SIFT Descriptor* This diagram shows how the orientation histogram is assembled for the classic SIFT descriptor. On the left is an 8 by 8 patch of the image, subdivided into 4 sections. The gradient direction and magnitude at each pixel is denoted by an arrow and the weighting Gaussian is denoted by the blue circle (though it would in fact degrade smoothly.) For each quadrant a weighted orientation histogram is built, as depicted on the right, based on the individual orientations. Each orientation contributes to the three adjacent section histograms by way of trilinear interpolation to alleviate border effects. Though a 2 by 2 descriptor is shown here, the actual descriptor consists of a 16 by 16 patch subdivided into 16 sections.

orientation histogram is formed using the gradient directions in the whole patch weighted using a Gaussian with a σ 1.5 times the scale of the feature and the magnitude of the gradient at each point. The maxima of this histogram are taken as dominant orientations for the patch and the descriptor rotated into alignment with these dominant directions to form a new patch. Each maximum greater than or equal to 80% of the true maximum is used to create its own descriptor with the patch rotated to its direction. This can result in several descriptors per interest point, which increases stability of feature detection. The descriptor itself is not rotationally invariant, but aligning to the dominant direction(s) allows for a robustness to orientation changes.

The $N \times N$ patch is split into c cells, and within each cell the intensity gradient at every pixel is calculated and the directions binned into a histogram weighted by their magnitude and a Gaussian window with a σ of .5 times the scale of the feature centered on the patch. If the bins are centered on d directions in each of c cells, the resulting descriptor is a $d \times c$ vector. By dividing the patch into cells, a particular gradient can move around to some degree within the descriptor window and still contribute to the same directional histogram. To provide a smoother transition when a gradient moves from one histogram to the next, however, trilinear interpolation is used to spread its contribution to all histograms by weighted using it with the quantity $1 - d$, where d is the distance from the gradient pixel to the center of a particular histogram as measured in histogram bin units. Once the $d \times c$ vector has been extracted, it is normalized to provide invariance to gradient magnitude (and hence contrast) change. One

final step is performed to help minimize the effects of non-affine lighting changes by thresholding all values in the unit vector to .2 and then renormalizing (the value of .2 was found experimentally.) Figure 2.20 helps with visualization of the descriptor. The SIFT descriptor has proven to be rich and is currently factoring into much vision research in place of other local feature descriptors, especially for localization [83] and image retrieval [52, 59]. However, it is very large (at $4 \times 4 \times 8 = 128$ dimensions) making similarity calculations very expensive.

2.2.4 PCA-SIFT

PCA-SIFT is an alternative patch type designed to be used as an alternative to Lowe's SIFT descriptor. Introduced by Ke et al. in [53], PCA-SIFT uses Principle Components Analysis [47] on oriented gradient patches during an offline process to obtain a gradient patch eigenspace that can be used for projecting extracted patches into lower dimensions to serve as vectors. During the offline step, a large collections of images is processed by Lowe's DoG detector to obtain a feature points and 41×41 patches are extracted at each, oriented to the direction of the feature point. The horizontal and vertical gradients of these patches are concatenated into a $2 \times 39 \times 39 = 3042$ length vector and PCA is performed on the entire collection. The top N eigenvectors of the resulting eigenspace decomposition are then used as the basis of the feature space, where N is typically between 20 and 40. In addition to being significantly smaller than Lowe's descriptor, it is faster to extract and more distinctive, making it a significant improvement to the technique.

Chapter 3

Image Retrieval

One of the major fields of research in Computer Vision is in the retrieval of images from large databases. The basic problem is that given a database of reference photographs an image retrieval system must return candidate objects for a set of query images, which depict objects and scenes that may or may not be in the database. In order to solve this problem, image content must be extracted from the reference and query images using the methods discussed in Chapter 2 and then organized in a manner that results in high precision and recall.

3.1 Precision and Recall

Before we get to the details of image retrieval, it would be prudent to examine the yardsticks by which retrieval systems are measured. Precision and recall are two concepts from data mining that serve to evaluate the quality of a retrieval system. Of interest are four quantities: the total number of correct documents for a query, the total number of documents in the database, the returned number of correct documents, and the total number of returned documents referred to as C_t, P_t, C_r, and P_r respectively. Precision is the quotient $\frac{C_r}{P_r}$, or the percentage of the returned documents which are correct. Recall is $\frac{C_r}{C_t}$, or the percentage of correct documents returned. It is obviously trivial to obtain maximum precision by returning no documents and maximum recall by returned the entire database. However, in each of those cases the other quantity goes to 0. Thus, systems which maximize both precision and recall can be said to be effective retrieval systems, with the perfect retrieval system returning only all the correct documents from the database, achieving a precision and recall of 1.

3.2 Representing an Image

One of the central areas of exploration in image retrieval deals with the way in which an image will be represented in the database. In this section we'll explore

three of the main ways in which this is achieved. There is no absolutely correct way of doing this, rather different applications require different representations of content.

3.2.1 Histograms

The simplest way of representing an image in the database is through a single histogram or collection of histograms. These histograms are calculated over the entire image for color, intensity and/or texture. Histograms can prove quite useful, especially in situations with high-quality reference and query images, or where an exact match is desired. Techniques which achieve partial matching for histograms can even make histograms deal well with matching individual sections of a larger image to smaller images for sub-image retrieval.

Normalized color histograms are used by Jain and Vailaya in [44] over the red, green and blue channels for image retrieval. IBM's QBIC image search framework [68] includes amongst its growing ledger of search criteria the ability to specify a color histogram to narrow a search, as does the Ogle's Chabot system [70]. Another example of robust histogram techniques for color can be found in Swain and Ballard's work [90] where they define histogram intersection and backprojection for efficient retrieval of multicolor objects in clutter.

3.2.2 Blobs

The concept of a "blob" is appropriately very nebulous, however the standard concept is that it consists of a multi-variate Gaussian calculated for a collection of content features from an image region. To create a set of blobs for an image, first a segmentation algorithm is used to divide the image into regions. Then, for each region, content is extracted at every pixel. A standard choice is to use several different color representations and a collection of texture cues with rough shape descriptions (size of the region and normalized position) to create a concatenated feature vector at each point. Then a Gaussian is fit to these points and used as the descriptor from that region, often by using the mean and diagonal covariance as one very large feature vector. This feature vector suffers greatly from dimensionality issues but is a very rich descriptor. If the initial segmentation is good, the feature vectors can be quite distinct and useful for object retrieval.

Blobs have been used exclusively as the representation, though with differing composition, throughout the evolution of the eponymous Blobworld retrieval system [18, 7]. It has also been used by other research from the Berkeley group, notably Barnard et al.'s work with auto-annotation and region labelling [3]. A similar system is used by Cián Shaffrey in his image retrieval work [84], though his representation is the Dual-Tree Complex Wavelet Transform as opposed to the responses of a filter bank.

3.2.3 Collection of Features

The most robust representation currently is to use a collection of local features, such as SIFT or Patch descriptors, to describe the image. The intuition about this is that the most interesting points in an image will be produced by the objects of interest. In the reference photographs, the points will be produced by the object against a matte background and as such create a clean database set of features. Query images may contain several objects producing many features, however if the descriptors are specific and clean enough as few as 3 can be used to perform matching to the database [59].

3.3 Defining Distance

Regardless of the descriptor used, the features from the query image need to be compared against all the database features to find a match. This can be a potentially intractable problem in itself, but it is compounded by the fact that defining similarity is itself difficult problem. In the case of Euclidean data, a simple distance calculation is needed, but the multi-dimensional vectors formed by feature descriptors are almost certainly not Euclidean and must be treated differently.

3.3.1 Defining a metric for non-Euclidean data

A distance metric must have the first three properties to be useful in the comparison of separate entities:

1. $d(x, y) \geq 0$ (non-negativity)
2. $d(x, y) = 0$ if and only if $x = y$ (identity of indiscernible)
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality).

While it is desirable for the triangle inequality to be maintained for all points, it is only required to be accurate for points which are nearby. If a point is equally far away from all points beyond a certain radius, for example, the triangle inequality may not hold for those points but performance as a distance metric will not necessarily be affected.

The way in which a distance metric achieves these properties is unimportant as long as they are followed. In this section we explore various metrics that are used to compare different kinds of non-Euclidean data.

3.3.2 Minkowski-form distance

More commonly known as L_m distance, Minkowski-form distance is a generalized distance metric for comparing two points on a hyperplane. It is described

by

$$L_m(p_1, p_2) = \left(\sum_i^N (p_1[i] - p_2[i])^m \right)^{\frac{1}{m}} \tag{3.1}$$

where N is the dimensionality of the hyperplane. For example, standard Euclidean distance ($d(p_1, p_2) = \sqrt{\sum_i^N (p_1[i] - p_2[i])^2}$ is the L_2 distance. In addition to comparing points, it also used to compare two histograms by treating a N-bin histogram as a point in N dimensions. The L_1 distance is used for compute dissimilarity between color images by Swain and Ballard in [90], however it was shown by Stricker and Orengo [89] that the L_1 distance is sub-optimal for image retrieval due to the number of false positives returned.

3.3.3 Histogram Intersection

The histogram intersection is a measure introduced by Swain and Ballard in [90] and is attractive due to its partial matching abilities. It is defined as

$$d_\cap(H, K) = 1 - \frac{\sum_i \min(h_i, k_i)}{\sum_i k_i}. \tag{3.2}$$

and thus is able handle partial matches when the areas of the two histograms differ. In the case of histograms of equal histogram areas, it is equivalent to the normalized L_1 distance [90].

3.3.4 Kullback-Leibler Divergence and Jeffrey Divergence

The Kullback-Leibler (K-L) divergence [54] is defined as

$$d_\cap(H, K) = \sum_i h_i \log \frac{h_i}{k_i}. \tag{3.3}$$

The K-L divergence is non-symmetric, however, and is sensitive to binning effects. The Jeffry divergence, on the other hand, is an empirically derived extension to the K-L divergence that is numerically stable, symmetric and robust with respect to noise and the size of the histogram bins [74]:

$$d_J(H, K) = \sum_i \left(h_i \log \frac{h_i}{m_i} + k_i \log \frac{k_i}{m_i} \right), \tag{3.4}$$

where $m_i = \frac{h_i + k_i}{2}$.

3.3.5 Earth Mover's Distance

The Earth Mover's Distance, or EMD, is a distance metric used specifically for comparing histograms. It has been used in image retrieval to some success as described by Rubner in [79], as it is able to partially match two histograms

or match histograms of different lengths with different bin centers. It achieves this by posing the problem of matching two histograms as an instance of the transportation problem from linear optimization [41] (also known as the Monge-Kantorovich problem [76]), for which efficient algorithms exist. The problem can be stated in terms of a set of suppliers and consumers with a limited supply of goods and capacity, respectively. The transportation problem is to find the least expensive way to transport the goods from the suppliers to the consumers that satisfies consumer demand. Another way of treating it is to consider one histogram as piles of earth and the other as holes in the ground, with the height or depth of the pile being related to the value for that bin. The distance between each pile of earth (the first histogram) and each of the holes in the ground (the second histogram), the "ground distance", is defined as the Euclidean distance between the bin centers and is used to calculate the cost per movement unit. Once this has been determined, the score is defined as the minimum amount of work required to move the earth into the holes, with low scores for two exact matches and high scores for disparate histograms.

3.3.6 Cross-Correlation

Cross Correlation is a metric of how closely two vectors resemble each other, and is used to compare image patches like the patch descriptor described in Section 2.2.2. In its simplest form, it is calculated by the equation

$$CC(P_1, P_2) = \frac{1}{N} \sum_{i}^{N} P_1[i] P_2[i].$$

However this equation has some problems, namely if the two vectors being compared have different means the metric wouldn't fulfill the requirements stated earlier. So, we must normalize the data, resulting in the equation for normalized cross correlation:

$$NCC(P_1, P_2) = \frac{1}{N} \sum_{i}^{N} (P_1[i] - \mu_1)(P_2[i] - \mu_2).$$

Unfortunately, this equation makes the assumption that the two vectors have the same variance. To truly make cross correlation a metric, we need one final adjustment, resulting in the equation for zero normalized cross correlation:

$$ZNCC(P_1, P_2) = \frac{1}{N} \sum_{i}^{N} \frac{(P_1[i] - \mu_1)}{\sigma_1} \frac{(P_2[i] - \mu_2)}{\sigma_2}.$$

It is for this reason that the patch descriptor described in Section 2.2.2 is modified to have zero mean and unit variance, in essence as a pre-processing step so that one can simply use the cross correlation operator yet it will still be a proper zero normalized cross correlation operation.

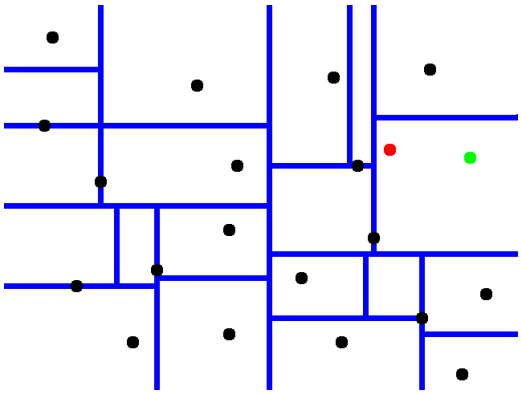

Figure 3.1: *Boundary Problems* This is a depiction of a 2 dimensional KD-Tree in which boundaries are chosen based on the median point. The data points are black, the boundaries are blue, and the query point is red. If a simple $O(\log(n))$ search is performed, the green point is returned, which is quite obviously not the nearest neighbor. In order to find the nearest neighbor in this case, other sub-trees must be searched.

3.4 Retrieval Methods

There are two different philosophies of image retrieval. In one, the retrieval system returns a list of results to the user which are similar to the query which he has made. The system is then evaluated on the composition of those results using precision and recall as described in 3.1. In this task, the goal is to find a neighborhood of results, or nearest neighbors in the data set, to the query images. Another approach to the problem is to guide the user interactively through the collection until he finds the image he is looking for. Though more intensive, it is often a more fruitful search process. We will describe both in this section.

3.4.1 Nearest Neighbor

The nearest neighbor problem is deceivingly simple: given a set of points and a query point, find the point in the set which is closest to the query. The naïve way of doing this is to look at every point in the data set and calculate the distance between it and the query, returning the point with the smallest distance. However, this is a linear search algorithm and we should be able to do better, especially since a linear search is prohibitive with large data sets. Thus, we essentially need a search data structure to speed up the process. For this, we turn to the KD tree.

KD trees

A KD-tree, short for K-dimensional tree, is a way of partitioning K-space into a binary search tree given a set of points in the space which are to be searched. A KD-tree is constructed from a set of points as follows

1. If the number of points in the current hyper-rectangle is less than or equal to the leaf size, finish recursion.
2. For all points in the current hyper-rectangle, find the dimension which has the highest variance. Set the cutting plane for the hyper-rectangle as the mean value in that dimension. [1]
3. Store the cutting plane at the current node.
4. Create two new child nodes with hyper-rectangles corresponding two the two halves of the current hyper-rectangle.
5. Recurse on the child nodes.

with the initial node being set to have a hyper-rectangle equal to the smallest area of K-space that encloses all the data points. The resulting search tree is then descended by seeing what side of the cutting plane a query point is on and going down that branch of the tree until a leaf is found. The resulting hyper-rectangle is going to contain a subset of the data which is highly likely to be nearest to the query point. Boundary issues cause problems, however, as is shown in Figure 3.1, and as such it is not guaranteed to be the case. Thus, we must continue the search to guarantee that the nearest neighbor has been found.

Once a candidate point has been found by descending the tree, this gives a minimum distance against which we can bound whatever continuing process we use. In branch and bound searching, we use the minimum distance to prune a comprehensive search of the tree to find the nearest neighbor. Starting at the root again, we calculate the distance from the query point to the cutting plane using the method:

1. Create a new vector p with dimension K.
2. For every dimension in p, if the query point in that dimension is less than the minimum value for the hyper-rectangle in that dimension, set the value in p to be the minimum value. If the query point lies between the minimum and maximum values for the hyper-rectangle in that dimension, set $p[i] = q[i]$ in dimension i for query q. Otherwise, set the value for p to the maximum value in that dimension.
3. The resulting vector p is the closest possible point in the hyper-rectangle to the query point q. To find the minimum distance simply return $||p-q||$.

[1] There are several ways of performing this step, the one given is a heuristic choice which has been shown to give good performance [51]

resulting in a test distance. If this distance is greater than or equal to the current minimum distance, it is unnecessary to search that entire subtree. It is customary to limit this search to a certain number of nodes however, as it can be prohibitive to search the entire tree with large data sets. If the wrong nodes are looked at first, the chances of finding the true nearest neighbor decreases with smaller search sizes.

A way of modifying branch and bound such that the chance of finding the correct nearest neighbor increases significantly was proposed by Beis and Lowe in [6] as Best Bin First searching. It functions in the same way as branch and bound search except that a priority queue is maintained as the search takes place for all branches not followed. Thus, whenever one branch is chosen over the other in the first descent the branch not chosen is added to the priority queue based on the distance between the query point and the cutting hyperplane. Once the leaf node has been reached and the initial minimum distance found, the nodes in the priority queue are exhaustively searched provided the distance is less than the current minimum. The result is that the most likely subtrees are searched first, which on average results in better performance on retrieval. The search constraint, which it shares with branch and bound searching, has a large effect on its performance and accuracy. This intelligent search method results in a significant reduction in the number of nodes that must be visited to result in an acceptable level of accuracy, but that number still grows with dimensionality and results in an, albeit reduced, curse of dimensionality.

The Curse of Dimensionality

Regardless of the clever technique used to search for the nearest neighbor, the higher the dimension the less useful the technique becomes. For Best Bin First searching, the maximum number of nodes to search must be increased with every added dimension in order to maintain an acceptable level of performance until, eventually, the number of nodes being searched to maintain performance is practically the entire data set, thus resulting in a de facto linear search. This is known as the curse of dimensionality, namely that in higher dimensions the best search technique will cost as much as a linear search if a certain level of quality (i.e. precision and recall) is required. However, there is a technique which alleviates this to some degree by playing a clever computer science trick. [2]

Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a true approximate nearest neighbor algorithm, in that it provides a guaranteed bound ϵ on the error between its result and the correct answer. First proposed by Indyk and Motwani in [43], the algorithm is based on hashtable collision. It is important to note at the start that the algorithm does not make any assumptions about the object similarity or

[2] A wise man once said that all computer science problems can be solved by either hashing, caching or a level of indirection. This is yet another case where he was correct.

dissimilarity measure, which helps to alleviate in some ways the the problems described in Section 3.4.1 as different measures can be used which are appropriate to a particular data set. The basic intuition of the algorithm is that if a range of locality based hash functions place a group of points in the same cell with the query point, then those points are likely to be the nearest neighbors of the query.

An implementation of LSH for use in computer vision was designed by Georgescu et al. in [37]. In the original presentation in [43] and subsequently in [38], Indyk et al. present an algorithm which explores the Hamming [39] and set difference [14] metrics for the purpose of proving the runtime bounds of the algorithm. Georgescu et al. use an extrapolated framework which maintains the same properties while being adapted to high-dimensional data. For LSH, one must define a set of hash functions to be used which are based on two parameters: L, the number of hash functions, and K, the complexity of those hash functions. They tessellate the data space L times with random partitions defined by K inequalities. In each partition, K pairs of random numbers $d_k \in \{1, 2, \ldots, d\}$ where d is the dimensionality of the space and v_k is in the range of the d_k-th dimension. The kth inequality is then $x[d_k] \leq v_k$, allowing a set of L binary strings to be constructed for each data point. This string is the hash code for the point, which is used to place those points in buckets in a hash table. Thus, for every data point there is a list of C_l buckets in L hash tables in which it falls. To find the nearest points to a query point q it is hashed to come up with L buckets C_l and the points in $C_\cup = \bigcup_l^L C_l$ returned. It should be noted that any $q \in C_\cap = \bigcap_l^L C_l$ will return the same result.

The advantages of LSH are that all queries are performed quickly and that the system can be tuned using L and K to deal with different data conditions. In general, increasing K reduces the average cell size in each partition, and L increases the percentage of the true neighborhood approximated by $C_\cup$. For any particular K, there comes a point where increasing L has a negligible effect search accuracy, thus placing a bound on the number of practical (K, L) pairs. A naïve implementation of the algorithm, however, is only efficient if the entire data set is held in memory, which for large data sets can be impractical. An implementation fine-tuned for disk access is provided by Ke et al. in [52] which remedies this problem. Experiments by Gionis et al. in [38] have shown that LSH shows almost no increase in retrieval error as the number of data points increase. Additionally, for each query the algorithm runs in $O(n^{\frac{1}{1+\epsilon}})$, though this upper bound may be unnecessarily high [43, 38] as it depends on values of K and L which are possibly suboptimal.

Spill Trees

A spill tree combines the efficient, tree-based structure of a KD-tree while using techniques from locality sensitive hashing to deal with the curse of dimensionality. It is also an approximate nearest neighbor data structure, but whereas the KD-tree with best bin first searching achieves this by limiting and optimizing the time-intensive back-tracking necessary to increase accuracy, a spill tree

modifies the tree structure itself by allowing points to be present in more than one node.

A standard metric tree, of which a KD-tree is a variant, consists of a binary tree in which the parent contains all the points of its children, and the children split the points between them according to a decision boundary. Ideally, this decision boundary is a hyperplane in the dimensionality of the points which divides the points equally into two parts, but an approximation to this are used in practice [3]. This tree is very similar to a KD-tree and has all of the same problems, namely that the candidate point found in an $O(\log(n))$ search is often incorrect due to boundary issues, as can be seen in 3.1.

A spill tree deals with this problem by replacing the hard decision boundary with a buffer zone which is shared by both children. In this way, points close to the boundary are present in both children and thus a better approximation to the true neighborhood is present at a leaf node. For a more complete discussion on spill trees, the reader is directed to [57], in which they were found to outperform both KD-Tree and LSH methods in accuracy and speed. They are still affected by the curse of dimensionality, however, and as such for dimensionalities greater than 30 a set of dimensionality reductions are performed and an array of spill trees set up, with the approximate neighborhood being the union of the results from all the trees in much the same way as in LSH. This technique can be applied to any data structure as a way of dealing with this problem, and goes a long way to helping do away with the curse, though at the cost of storage and speed.

3.4.2 Browsing

While automatic retrieval of images based on static queries requires a nearest neighbor system of some description, another method of retrieving images is through browsing. Whereas the retrieval task relies on the system to be able to retrieve an accurate subset of the data which reflects a given query, browsing relies on providing accurate and descriptive feedback to a user to aid them in finding what they are looking for as they engage in an interactive process. The difference is akin to visiting a library. Strict retrieval is analogous to asking the librarian to retrieve for you books which deal with a particular issue. The books which are returned will undoubtedly be books which deal with that topic, but it will be based on the librarian's opinions and knowledge of the field. In addition, you will be unable to broaden or tighten your search as necessary during the process (the librarian is away from the desk) or get a sense of what other topics can be found near your intended topic which may pertain to your research. Retrieval through browsing, however, is like asking the librarian to show you to the section where you'd find books on your desired topic. As you move through the library you can change your mind and ask to be sent to other

[3] Typically, a point x is chosen at random and then a point p_1 is found which is farthest from x. Then, another point p_2 is found which is farthest from p_1. All other points are projected onto the line between these two points and the mean or median point on this line used as the decision boundary.

Figure 3.2: *Database Paintings* These are eight sample paintings from the 118 image test database. 1000 to 2000 features were extracted for each and inserted into a KD-tree for nearest neighbor retrieval.

sections, and once you arrive you can delve into a richer field of inquery. There have been a few systems which have provided image browsing of this nature. Of particular note is the work of Cox et al. in [22] on the *PicHunter* Bayesian browsing system. In each iteration of the system N images are shown to the user based on his past history of actions (e.g. choosing one or more of the images he is shown) with each new image T_i being chosen according to whether they are the likely target T as determined by

$$P(T = T_i|H_t) = \frac{P(H_t|T = Ti)P(T = T_i)}{P(H_t} \tag{3.5}$$

where H_t is the current action history for the user. The advantage of the system is that it is a general framework for a Bayesian formulation of the browsing problem. The definition of H_t and the nature of $P(H_t|T = Ti)$ is open to interpretation. In their system, the user is chosen N images and their history consists of the images they were shown at step t, D_t, and the action they took, A_t, which consists of choosing one or more images. As the user chooses images that are "like" the image he desires to find he moves through the system and is able to experience the full range of data available until eventually converging on a subset of the data.

3.5 Study: Building a State-of-the-Art Image Retrieval System

At this point in the report, enough has been covered that we can build a complete, working and state of the art near-duplicate and sub-image retrieval sys-

tem. A near-duplicate and sub-image retrieval system returns all images in the database which are near-duplicates of a query image or contain the query image within them. Our particular system is going to help users find information on a painting based on a query photograph of the painting. For this study, the database used is the 118 photographs of paintings hand-extracted from auction catalogs, 8 of which are shown in Figure 3.2.

The system consists of 4 subsystems:

1. Interest Point Detector
2. Feature Descriptor Extractor
3. Nearest Neighbor Database
4. Match Generator

We will discuss each in turn and in some detail, ending with a look at the Precision/Recall results for a set of query images.

3.5.1 Interest Point Detector

The interest points used in this system are Lowe's Difference of Gaussian (DoG) points as discussed in Section 2.2.1. The reason for this is that the points output by this system are very stable, making them ideal starting points for a matching system. For the purpose of clarifying the method used to extract them, we will discuss it in a bit more detail than was used earlier. The process consists of the following steps [4]:

Prepare the input image

Take an input image I (a monochrome image with intensity values from 0 to 1) and double its size using linear interpolation to receive I_{double}. Then, smooth I_{double} using a Gaussian kernel with $\sigma = 1.5$ to receive $\hat{I}$

Build the Gaussian pyramid

Every octave o has k intervals. At the start of each octave, smooth the initial image with a Gaussian of $\sigma = 1.6$, with the input to the first octave being the prepared input image $\hat{I}$. Then create each interval by smoothing the previous interval with a Gaussian of $\sigma = 2^{\frac{n}{k}}$, where n is the interval. Create $k+3$ intervals (this is important!). To make the next octave, sub-sample the kth interval of the previous octave by two.

[4] This entire tutorial on the DoG detector is a distillation of Lowe's system as described in [59]

Build the DoG pyramid

Create each octave of the DoG pyramid by subtracting the $i+1$th interval of the Gaussian pyramid from the ith interval for $i = 0$ to $k+2$ from the corresponding octave in the Gaussian Pyramid. This gives $k + 2$ DoG images, which again is very important.

Determine candidate points

For each DoG image i from $i = 1$ to k compare each point to see whether it is a local maximum or minimum. If it is, test it against the DoG image above and below ($i - 1$ and $i + 1$) to see whether it is the minimum or maximum of its 26 neighbors in scale space. This is why we need $k + 2$ DoG images in each octave: to allow an accurate comparison with the neighboring intervals. All points which are maximal or minimal in the local scale space are the candidates. So, to say it again, we test to see whether a point (x, y, s) is the maximum of all 8 of its neighbors at scale s, then look at scale $s - 1$ and $s + 1$. An important thing to note here is that while it needs to be greater or less than all its neighbors in scale space, it only needs to be greater than or equal to (or less than or equal to) the points at $(x, y, s - 1)$ and $(x, y, s + 1)$.

Localize each candidate point

To localize the candidate points in scale space, we use the Taylor expansion of the scale space function (localized to a particular point) in three variables: x, y and s. This expansion is as follows:

$$D(\mathrm{x}) = D + \frac{\partial D}{\partial \mathrm{x}}^T \mathrm{x} + \frac{1}{2}\mathrm{x}^T \frac{\partial^2 D}{\partial \mathrm{x}^2}\mathrm{x} \tag{3.6}$$

where D is the DoG image value at $\mathrm{x} = \begin{bmatrix} x \\ y \\ s \end{bmatrix}$. This gives us $H = \frac{\partial^2 D}{\partial \mathrm{X}^2}$ (the Hessian), $G = \frac{\partial D}{\partial \mathrm{X}}$ (the gradient) and $D = D(x, y, s)$ (the scalar). The Hessian is composed as:

$$H = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial x \partial s} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} & \frac{\partial^2 D}{\partial y \partial s} \\ \frac{\partial^2 D}{\partial x \partial s} & \frac{\partial^2 D}{\partial y \partial s} & \frac{\partial^2 D}{\partial s^2} \end{bmatrix} \tag{3.7}$$

where

$$\frac{\partial^2 D}{\partial x^2} = D(x+1,y,s) - 2D(x,y,s) + D(x-1,y,s)$$
$$\frac{\partial^2 D}{\partial y^2} = D(x,y+1,s) - 2D(x,y,s) + D(x,y-1,s)$$
$$\frac{\partial^2 D}{\partial s^2} = D(x,y,s+1) - 2D(x,y,s) + D(x,y,s-1)$$
$$\frac{\partial^2 D}{\partial x \partial y} = D(x-1,y-1,s) + D(x+1,y+1,s) - D(x+1,y-1,s) - D(x-1,y+1,s)$$
$$\frac{\partial^2 D}{\partial x \partial s} = D(x-1,y,s-1) + D(x+1,y,s+1) - D(x+1,y,s-1) - D(x-1,y,s+1)$$
$$\frac{\partial^2 D}{\partial y \partial s} = D(x,y-1,s-1) + D(x,y+1,s+1) - D(x,y+1,s-1) - D(x,y-1,s+1)$$

Then we have the gradient G as:

$$G = \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \\ \frac{\partial D}{\partial s} \end{bmatrix} \tag{3.8}$$

where

$$\frac{\partial D}{\partial x} = D(x+1,y,s) - D(x-1,y,s)$$
$$\frac{\partial D}{\partial y} = D(x,y+1,s) - D(x,y-1,s)$$
$$\frac{\partial D}{\partial s} = D(x,y,s+1) - D(x,y,s-1)$$

We take the derivative of this and set it to zero to find the maximum:

$$\hat{\mathrm{x}} = -\frac{\partial^2 D}{\partial \mathrm{x}^2}^{-1} \frac{\partial D}{\partial \mathrm{x}} \tag{3.9}$$

which localizes the candidate.

If the candidate moves more than .5 in any one direction you need to move the sample point in that direction and re-localize. If this happens more than N times (a good value for N is 2) then the candidate is bad and is rejected. The next culling step is to look at the value of D at the maximum and threshold. Lowe suggests a value of .003, however in practice a lower value (such as .001) can be used to result in more keypoints with very little degradation in performance.

Test the Curvature

Now that the keypoint has been localized and thresholded based on contrast, we look at the local curvature. In order to test the curvature we create another

Hessian, but this time it is only computed for location, not scale, resulting in the matrix

$$H = \begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \frac{\partial^2 D}{\partial x \partial y} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix} \tag{3.10}$$

where all terms are defined as above. If the ratio of the trace of H squared to the determinant of H is greater than $\frac{(r+1)^2}{r}$ (Lowe uses $r = 10$) then that candidate is rejected due to it being an edge.

Detecting Dominant Orientation

We can now disregard the DoG pyramid, it is of no use anymore. All remaining work will be done on the Gaussian pyramid. For each candidate we need to find the dominant orientations. First, we precompute the gradient magnitude and orientation for every image in the pyramid. Then, we sample the orientations weighted by a Gaussian centered at the candidate point with $\sigma = 1.5 * s$ where s is the scale of the point and the magnitude of the gradient. In other words, we create an orientation histogram O with N bins (Lowe uses 36) and then every point within 3σ of the candidate point at (x_c, y_c) contributes a value of $G(x -_x c, y - y_c)m(x, y)$, where G is the aforementioned Gaussian and m is a function of the magnitude of the gradient, to the appropriate bin in O for a value of $\theta(x, y)$, where θ is a function of the orientation of the gradient. We then look for local maxima in this histogram (read: greater than both their neighbors) and fit a parabola to the maximum and its two neighboring bin values to find the actual maximum orientation (by taking the derivative of said parabola and setting it equal to zero). For each of these maxima which are 80% or more of the value at the global maximum we create a separate interest point. This process gives a set of stable DoG points from which we now extract patches.

3.5.2 Feature Descriptor Extractor

The second step of the process is to extract a feature descriptor at each point. For this study, we used Ke et. al's PCA-SIFT descriptor [53]. The PCA-SIFT descriptor is formed by performing a dimensionality reduction on a 3042 dimensional patch descriptor extracted at the interest point. First, a rotated 41×41 patch of the scaled image in which the interest point was detected is calculated so that the orientation of the interest point corresponds to 0 degrees in the patch. The x and y differences are extracted separately to create two 39×39 images which are concatenated into the full 3042 dimensional patch descriptor.

Principle components analysis [47] consists of finding the eigenvalue decomposition of the covariance matrix of a set of data. The resulting set of eigenvectors forms an orthogonal basis for the points, and can be used to reduce the dimensionality of data while retaining much of the variance. In the case of this study system, a precomputed eigenspace based on natural images was obtained from Ke's website.

Figure 3.3: *Sample DoG Interest Points.* Lowe's DoG detector was used on each image to extract interest points. Each red arrow indicates a feature and orientation, with the length corresponding to the scale of the keypoint.

Figure 3.4: *Sample Query.* This is a sample query to the system. There are four images in the query, each occluding a significant portion of the others (with the obvious exclusion of the topmost image). In addition, they have been scaled, rotated and either had noise added or their contrast changed.

3.5.3 Nearest Neighbor Database

Every image in the database was run through the Interest Point Detector and Feature Descriptor Extractor to create a list of keypoints for each image. An image from the database and its features can be seen Figure 3.3. The data structure used for the database was our implementation of the disk-based LSH data structure described in [52].

Upon receiving a query photograph, the first step in the query is to extract the features, as shown in 3.4. Once the features have been extracted, we will perform a nearest neighbor search on each keypoint in the image to find its nearest neighbor in the data set. Some keypoints won't have a good match in the database due to their being background clutter, obstructed in all database images, or unimportant for matching. Thus, we need a rule for discarding these bad points. For our needs, we will record the first and second nearest neighbors in the neighborhood returned by the LSH data structure and if the first nearest neighbor is 80%[5] or more of the distance to the second nearest neighbor, we will discard that keypoint.

3.5.4 Match Generator

Only images in the database who have keypoints which are in the set of nearest neighbors will be examined. For each of these images, the system will take the pairs of nearest neighbors for that image and attempt to find groups which can be explained by the same object pose. Instead of the generalized Hough

[5]This value was determined empirically by Lowe [59]

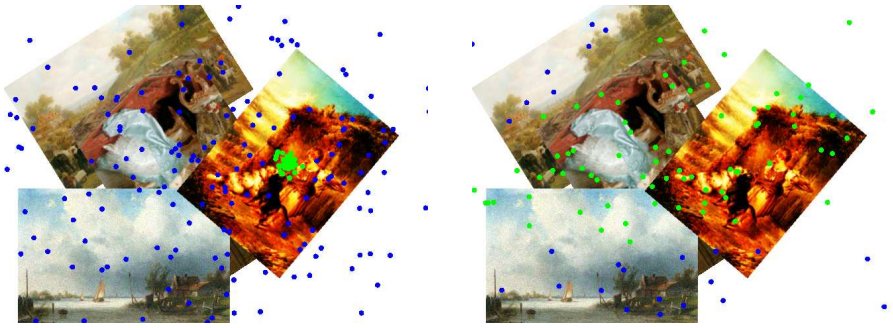

Figure 3.5: *Center Clusters.* Each circle shown is a proposed center from a keypoint based on that keypoint's scale, orientation, location and the relative position from the center of its matched keypoint in the database. As can clearly be seen, there is a clear cluster of correct points in the case of a true positive and just noise for false positives. The colors of the keypoints indicate which mixture component they belong to, again notice that the noise in the true positive image on the left is explained completely, with the correct matches being identified clearly for use in geometric verification.

transform used by Lowe, we will describe here a novel probabilistic approach to give a candidate pose.

Probabilistic Pose Prediction

For every point in the database, we have encoded position as a vector pointing from the center of the database image to the feature, denoted x. For every matching pair, we calculate the difference in orientation, $\nabla\theta = \theta_q - \theta_r$ and the difference in scale $\nabla s = s_q/s_r$ and use them to create a transformation matrix T where

$$T = \begin{bmatrix} \nabla s\cos(\nabla\theta) & -\nabla s\sin(\nabla\theta) \\ \nabla s\sin(\nabla\theta) & \nabla s\cos(\nabla\theta) \end{bmatrix}. \tag{3.11}$$

With T, we can now find a vector $\mathrm{x}' = T\mathrm{x}$, with which we can find the predicted center of the source image in the query image by subtracting it from the query point position. Thus, every match pair predicts a center, and we can analyze this plot to group the pairs. For true positives there will be a mass of center predictions around the actual center of the object and then random predictions, and for false positives there will be a noisy distribution of centers across the image as can be seen in Figure 3.5. Another way of looking at this is that in true positives there are center predictions which are produced by the true keypoints generated by the object and the rest are noise. When posed this way, the most natural way to solve the problem is by fitting a mixture of $K+1$ bivariate Gaussians to the plot given that K instances of an object are expected. In this study system, K is equal to 1, however more complex systems would use the minimum description length principle to choose between different mixture models over a range of values for K. An example of pose prediction for multiple object instances is shown in Figure 3.6. In Figure 3.5, the query image has

different colored center predictions plotted to signify cluster members, and as can clearly be seen one can cleanly separate out the true centers from the noise in the true positive case. For all mixture model components we gather together their constituent points as a basis for an affine transform calculation.

All true positive matching pairs are used to find a true transform from the result image to the query image. Due to the nature of our pose prediction method, the true positive matching pairs have a high probability of geometric correlation. The nature of the geometric constraint in this study system is an affine transform, though a full homography is just as easily computed.

The affine transform is found using least squares. Specifically, if the affine transform is written as

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{3.12}$$

then we can write out a linear system as:

$$\begin{bmatrix} x_1 & y_1 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_1 & y_1 & 0 & 1 \\ x_2 & y_2 & 0 & 0 & 1 & 0 \\ 0 & 0 & x_2 & y_2 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n & y_n & 0 & 0 & 1 & 0 \\ 0 & 0 & x_n & y_n & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_x \\ t_y \end{bmatrix} = \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_n \\ v_n \end{bmatrix} \tag{3.13}$$

Which naturally is of the form $A\mathrm{x} = \mathrm{b}$ and is therefore solvable by means of least squares, namely $\mathrm{x} = [A^T A]^{-1} A^T \mathrm{b}$. Once we have the affine transformation, we remove any matching pairs which don't agree within .25 of the source image dimensions as projected into the new image. We repeat this process N times, where N can be as low as 2 due to the way in which the basis points are chosen, but can be set higher for more accuracy in the transform.

After this process, a set of affine transformations and matching points for a subset of the database has been found, however a means of evaluating these matches is needed. This is done using a novel scoring system. What we are interested in is the joint probability of the transform and the data, $p(A, D) = p(A|D)P(D)$, where A is the affine transformation and D is the set of matching points. To do this, we model the posterior $p(A|D)$ as a bivariate Gaussian, the mean and covariance of which are calculated using the center point predictions of the matching points. The $p(A|D)$ is then calculated by using the affine matrix's center prediction, $\begin{bmatrix} t_x \\ t_y \end{bmatrix}$. For correct matches, the transform's center prediction is at or very near the mean and thus gives a high probability. The prior $P(D)$ is taken as the percentage of total matching points remaining after fitting represented by the basis points D. For incorrect matches the transform's center prediction is far from the mean and given the distribution's large variance for incorrect matches, results in a very low probability for the match, with the prior ensuring that images with a small number of matching points will not be given as much certainty as those with more support.

Figure 3.6: *Pose Prediction.* The three images shown are combined in the test image under a variety of transformations (scale, rotation, noise, contrast, brightness). Below each is shown the center predictions colored according to mixture model membership, where the models used have a $K = 3$. The last row shows the query image and the resulting matches.

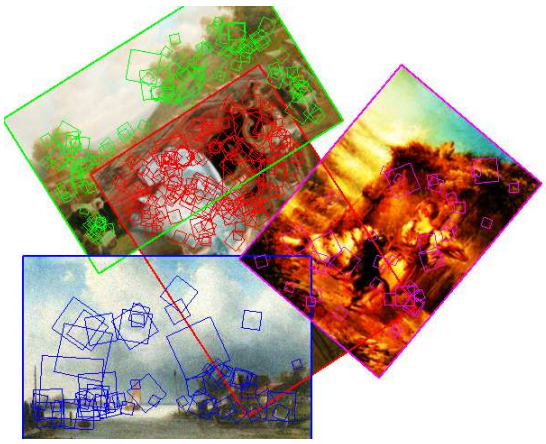

Figure 3.7: *Sample results.* These are the results from the system for the sample query. Each color square indicates a different matching keypoint at a particular location, orientation, and scale. The red, green, blue and magenta squares are the correct matching keypoints and the bold colored rectangles are the position predictions after two iterations of least squares.

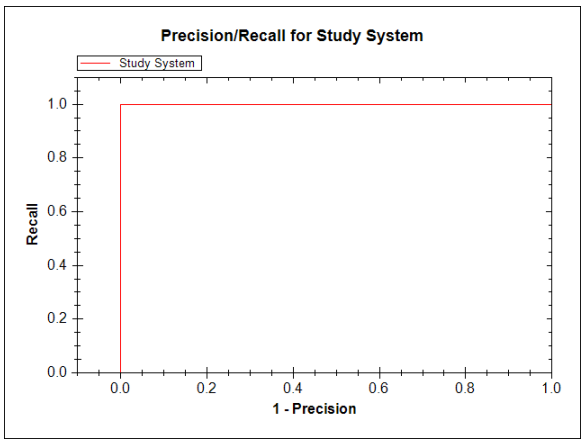

Figure 3.8: *Precision/Recall Curve.* Shown is the combined precision/recall curve for the system. The altered quantity is the certainty value threshold. As can be seen, the system performs perfectly on our database of 118 images, even though the set of query images are in some cases quite difficult. There were 31 queries: 5 occlusion images, 5 contrast-altered images, 11 rotations, 5 noisy images, and 5 images with multiple alterations in combination.

3.5.5 Results

The results for the query image in Figure 3.4 can be seen in Figure 3.7, with the transform onto the query image depicted as a rectangle with the oriented rectangles within being the matching points. A precision/recall curve is given in Figure 3.8.

The problem with this system, and indeed any system based on SIFT or a similar descriptor and using nearest neighbor matches to score images in a database, is that as the number of database images increases the cost of reliably finding the correct nearest neighbors while maintaining performance to the point where it becomes prohibitive. This is an unavoidable result of any technique utilizing nearest neighbor search in a database. Thus, in order to use these techniques on a large scale some method culling the database as a pre-processing step must be found, or alternatively some way of dividing the database into subsets that are reliably and efficiently searched.

Chapter 4

Learning Image Concepts

Image content concentrates on extracting data, in the form of features, which describe an image. Learning image concepts allows for the more difficult task of extracting meaning from an image. Extracting image content is much like reading the mass values on a scale: it gives you information on the substance being measured but does not tell you what that information signifies. Learning the image concept is like taking that measurement and comparing it against other measurements in an attempt to find out an underlying property of the substance, such as its molecular weight. Whereas image content can tell you that an image has a large reddish-orange segment on top of the image with a yellow circle inside of it with a darker segment in the bottom of the image, the image concept is that it depicts ocean sunset.

For the purpose of image retrieval, a concept or concepts (such as sun) extracted from a query can be used to match with images in the database which may not be visually similar by standard content matching techniques, such as a sunset for a query image of the sun in a blue sky. In addition, it can be used to organize the database to enable otherwise limited search techniques (such as the one in Section 3.5) to be more effective, as is proposed for the `HUGINN` system in Section 6.2.

4.1 Image labelling as translation

Of the two problems at hand, learning image concept is by far the most difficult. One method of approaching the problem is through the conceptual framework of concept as translation. To return briefly to the analogy of content as instrument readings, the trick in learning the concept is in finding the underlying law which causes a particular quantity to give its reading, and thus to be able to predict future measurements. The way this law is found is through repeated measurement and recording of results to find a trend, a process which is similar to that used to find the translation between two languages. If enough aligned texts (documents in one language with their correct translation) have been ac-

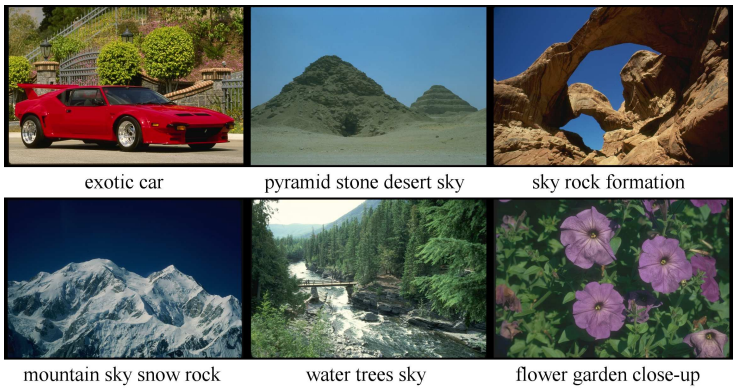

Figure 4.1: *Labelled Images as Aligned Text.* These images were taken from the Corel database, where images have anywhere from 1 to 5 accompanying keywords which describe the image contents. There is no correspondence given between segments and text, however they form two aligned sentences in the image and word languages.

quired then co-occurrence of translated words can be recorded and rules for translation found. All of these rules together define the relationship between the two languages.

If one thinks of image segments, or blobs, as "words" in an image language, then a series of labelled images as shown in Figure 4.1 essentially form an aligned text. Once the problem has been thus formulated, object recognition becomes a matter of finding the translation rules between image portions and the words for the objects they represent. This framework is the focused study of research begun by Barnard et. al in [5] and continued through [4, 27] to a comprehensive study of the idea in [3].

4.2 Learning concepts

Learning concepts as translation rules requires large quantities of training data to be able to perceive all of the co-occurrences that will form the rules and to make sure that the concepts themselves are general enough to apply to new data. For example, if the system is learning the concept of the "sun" but is only shown sunsets, it may be unable to identify the sun in a clear blue sky. The process of learning concepts is achieved through machine learning and there are several different techniques by which a translation model can learn from data trends.

A variety of translation models have been used throughout the nascent stages of the technique, including those used by Brown et al. in [16] for finding translation rules between French and English from the aligned texts of Canadian parliamentary debates, the hierarchical clustering models of Hofmann and Puzicha [42] and latent Direchlet allocations [11, 87]. In [3], Barnard et al. perform a thorough evaluation of various models, with the two best groups being found to be multi-modal hierarchical aspect models and mixtures of multi-modal Latent

Dirichlet Allocation. It is these two groups that we will examine in closer detail in this section.

All methods are designed to work with feature vectors, or "blobs", representing the segments of an image. These are related through annotations to associated words which describe the image as a whole. The accuracy of the segmentation method isn't vital provided that it is consistent, allowing the use of automatic segmentation techniques such as normalized cuts [85] to prepare the data, regardless of their lack of accuracy. Once an image has been segmented, a feature vector is extracted for each segment containing color (mean and variance of RGB sum, RGB, and normalized r/(R+G+B) and g/(R+G+B), color context), texture (mean and variance of odd and even phase filters at 2 sigmas and 6 orientations) and other cues (shape statistics, position, size, etc.) as described in by Gabbur in [34].

4.2.1 Hierarchical Asymmetric Clustering

Of the heirarchical aspect models, the **I-2** model serves to introduce the basic idea and its implementation, and will be discussed here. In the **I-2** model, blobs and words are statistically linked through the assumption that there are hidden factors (concepts) which are each responsible for generating *both* the words and blobs associated with that factor. By generating both words and blobs, the concepts can then be used to link the two, learning how they relate. The observations (image and associated text) are assumed to be generated from multiple draws from the hidden factors, as otherwise all possible combinations of words and blobs would need to be modelled. The joint probability of a particular blob, b, and a word w, is modelled as

$$P(w,b) = \sum_c P(w|c)P(b|c)P(c) \tag{4.1}$$

where c indexes over the concepts, $P(c)$ is the concept prior, $P(w|c)$ is a frequency table, and $P(b|c)$ is a normal distribution over features. The normal distribution over features is assumed to have diagonal covariance to simplify calculation and avoid overfitting. The probability of the observed data, $W \cup B$, given the model, is then:

$$P(W \cup B) = \prod_{b \in B} \left(\sum_c P(b|c)P(c) \right) \prod_{w \in W} \left(\sum_l P(w|c)P(c|B) \right) \tag{4.2}$$

where W is the set of all annotated words, B is the set of blobs and $P(c|B) \propto \sum_b P(c|b)$, normally limited to the N largest blobs (typically 8 or 10).

This model is fit using the Expectation/Maximization technique [26] treating the concepts as hidden data. This results in the following learning equations for

Expectation:

$$P(c|b) \propto P(b|c)P(c) \tag{4.3}$$

$$P(c|B) \propto \sum_{b \in B} P(b|c)P(c) \tag{4.4}$$

$$P(c|w, B) \propto P(w|c)P(c|B) \tag{4.5}$$

and Maximization:

$$P(c) \propto \sum_{d} \left[\sum_{b \in B} P(c, b) + \sum_{w \in W} P(c|w, d) \right] \tag{4.6}$$

$$\mu_c = \frac{\sum_b P(c|b)\mathbf{b}}{\sum_b P(c|b)} \tag{4.7}$$

$$\sigma_c = \frac{\sum_b P(c|b)(\mathbf{b} - \mu_c)(\mathbf{b} - \mu_c)^T}{\sum_b P(c|b)} \tag{4.8}$$

$$P(b|c) \propto \mathcal{N}(\mu_c, \sigma_c, b) \tag{4.9}$$

$$P(w|c) \propto \sum_{d} P(c|w, B) \tag{4.10}$$

where d indexes the training documents and $\mathcal{N}$ is a multivariate normal distribution.

4.2.2 Mixture of Multi-Modal Latent Direchlet Allocation

The other model used is the relatively new, promising and easily misspelt Latent Dirichlet Allocation, first introduced by Blei and Jordan in [11] and then adapted by them to the task of object recognition within a machine translation framework in [3]. In that paper, a mixture of multi-modal LDA's is used to model the complex relationship between images and text. The LDA is a generative probabilistic model like a multivariate normal distribution, but is tailored for collections of data, in the sense that the LDA generates collections intact ex nihilo as opposed to generating each item in the collection independently. For this task, a collection of words and blobs makes up a document and an LDA provides a model of independently generated whole documents.

Since LDA's are generative, they are readily used in a mixture model. That model assumes that each document (blobs and words) was generated by the following process:

1. Choose one of J mixture components $c \sim \text{Multinomial}(\eta)$.
2. Conditioned on the mixture component, choose a mixture over J factors, $\theta \sim \text{Dir}(\alpha_c)$.
3. For each of the N words:
   (a) Choose one of K factors $z_n \sim \text{Multinomial}(\theta)$.

(b) Choose one of V words w_n from $p(w_n|z_n, c, \beta)$, the conditional probability of w_n given the mixture component and latent factor.

4. For each of the M blobs:

   (a) Choose a factor $s_m \sim \text{Multinomial}(\theta)$.

   (b) Choose a blob b_m from $p(b_m|s_m, c, \mu, \Sigma)$, a multivariate normal distribution with diagonal covariance, conditioned on the factor s_m and the mixture component c.

Maximum likelihood estimates of the Dirichlet, word multinomials, and Gaussian parameters can be obtained by the EM algorithm with a variational E step as described in [11].

Given an image and a mixture of multi-modal LDA's (**MoM-LDA**), image based word-prediction is performed by finding the distribution over words based on the computed approximate posterior over mixture components and, for each mixture component, an approximate posterior Dirichlet over factors. Let ϕ denote the approximate posterior over mixture components, and γ_c denote the corresponding approximate posterior Dirichlet. The distribution over words given a collection of blobs B is

$$P(w|B) = \sum_{c=1}^{J} P(c|\phi) \sum_{z=1}^{K} P(w|z) \int P(z|\theta)p(\theta|\gamma_c)d\theta. \tag{4.11}$$

The integral over θ is the expectation of the zth component of $\theta \sim \text{Dir}(\gamma_c)$, and thus easily computed as

$$\int P(z|\theta)p(\theta|\gamma_c)d\theta = \frac{\gamma_{cz}}{\sum_{y=1}^{K} \gamma_{cy}}.. \tag{4.12}$$

The **MoM-LDA** has the advantage of accommodating documents with multiple topics, which is not handled well by the multi-model hierarchical aspect models. Models such as **I-2** try to find a single concept which dominates the document and use that concept to predict words, while the **MoM-LDA** can accommodate topic clusters by conditioning the concept z over the Dirichlet.

4.3 Auto-annotation and Image Retrieval

Once a model for $P(w|b)$ has been trained, a system can be built which can take segmented images as input and generate a distribution over the word vocabulary for the image. The weight of the distribution concentrates around words which were most often associated with blobs of the given type in the training corpus, but through the use of an intermediary concept it is able to predict words for a blob which did not necessarily co-occur but are still appropriate. This model

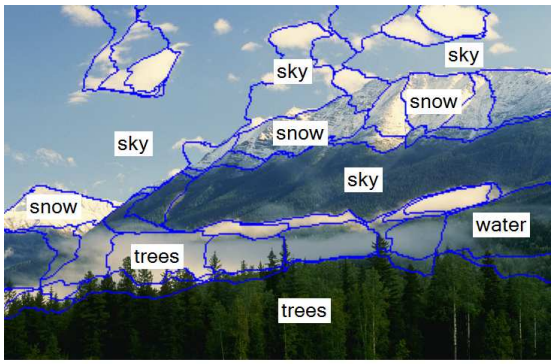

Figure 4.2: *Image Labelling and Annotation.* The word shown is the most probable label for that area of the image, however a full distribution over the vocabulary is produced for each segment. The words below the image are the annotation produced by the algorithm based on the segment labels.

can be used to both label individual blobs with words by assigning the most probable word in the vocabulary and also to label the image as a whole using

$$P(w|B) = \sum_c P(w|c,B) \prod_{b \in B} p(b|c)P(c) \tag{4.13}$$

where B is the set of all blobs in an image (or in practice the N largest) and c indexes concepts. The quantity $P(w|c,B)$ is approximated using its average value, $P(w|c)$ with negligible performance loss to closer approximations [5]. The result of both processes can be seen in Figure 4.2. Equation 4.13 is given using the framework of the **I-2** model, whereas for **MoM-LDA** the same effect can be reached by restricting the collection of blobs to a single blob for the $P(w|b)$ case. The first task is labelling ($p(w|b)$) and is a form of object recognition, however the second task, auto-annotation ($p(w|B)$), has interesting applications to image searching.

This method of extracting image concepts provides an interesting model for document retrieval. Due to the nature of the distributions modelled the user can query on words and/or blobs and the results will be retrieved which best model the concept those words and blobs jointly represent. More precisely, given a target set of documents $D = \{d|d = W \cup B\}$ where W is a set of words and B is a set of blobs, the probability of a user query Q consisting of words and/or blobs being generated by (and thus related to) a document d can be determined using

$$P(Q|d) = \sum_c P(Q|c)P(c|d) \tag{4.14}$$

$$P(Q|c) = \sum_c \left(\prod_{q \in Q} P(q|c) \prod_{i \in d} P(i|c) \right) \tag{4.15}$$

where q is either a word or a blob and $i \in W \cup B$ from document d.

Chapter 5

Multi-label Boosting

The Barnard et al. approach to extracting concepts as presented in Chapter 4 is very effective but it is limited by the fact that it is forced to perform well overall on the entire data set by the training process. As a result, the concepts which are learned tend to relate textually to the words which occur most often in the data (e.g. sky, water, sun) and visually to the easiest to recognize segments (e.g. solid blue, textured green, yellow circles.) This effect can be seen in Figure 5.2, where the model predicts the entire image as "sky". While this produces good results for a large subset of the data and thus is optimal behavior for the system, it doesn't leave any concepts left over to represent the many words and image segments that are less prevalent. A way to somehow account for this forgotten data would significantly improve performance of the algorithm.

5.1 Boosting

Boosting is a process by which several guideable weak learners are combined to create a single system which is better than the sum of its parts. Freund and Schapire, in their seminal paper on Adaboost [32], liken it gambling on horse races. Imagine that you are a terrible gambler, but each of your friends (who are better at betting on horse races than you) gives you a rule of thumb (e.g. bet on the youngest horse, bet on horses named after flowers.) After each race you record the conditions in which each rule succeeded or failed and over time you discover a means by which to combine the rules such that you can successfully bid on the winning horse the majority of the time based on the conditions for the race.

The iterative process consists of telling each subsequent learner to concentrate on a certain subset of the data which has yet to be learned by the previous learners, and thus as a concept it is an ideal solution to the weakness of the concept learning systems in Chapter 4. No boosting system has ever been built, however, which is able to boost multi-label learners of this variety. We present here the **MLBoost** system (first published in [46]) which provides boosting for

multi-label learners. This system provides a framework for improving concept learning by not only improving the performance existing systems but allowing the intelligent combination of different learners working with the various kinds of content described in Chapter 2 such that the best features are used to distinguish between concepts, with each concept being able to use a prime mixture of content types.

5.2 Multilabel and Multiclass Boosting

In its most basic form, Adaboost deals with binary classification (i.e. something is/isn't a member of a class). However, Freund and Schapire provide two multiclass versions of Adaboost in [32]. Our multi-label framework is an extension of the second version, which dealt with learners that were evaluated in terms of pseudoloss. The concept of pseudoloss is quite well suited to dealing with the data ambiguity problem at hand. It was originally defined as

$$\text{ploss}_q \doteq \frac{1}{2}\left(1 - h(x_i, y_i) + \sum_{y \neq y_i} q(i,y)h(x_i,y)\right) \tag{5.1}$$

where q is a label weighting function, x_i is a data input, y_i is the correct label for that data, and h is a hypothesis returned by a weak learner which assigns a certainty value between 0 and 1 for each label given the input. The label weight function, q, deserves some mention. This function measures the degree to which the weak learner mistakes x_i for being a member of another class. It is calculated such that $\sum_{y \neq y_i} q(i,y) = 1$. In this way, if the certainty sum for every label (except the correct label) is 1 then the pseudoloss is 1, and if the classifier is perfect (the correct label has a certainty of 1 with all others 0) the pseudoloss is 0. The place of q in the algorithm is to guide the learner towards learning certain classes more thoroughly than others, which enables the boosting system to concentrate on classes which are hard to classify and underrepresented.

Our situation is slightly different from that of this initial formulation. Whereas the original deals with data items with single labels, we are dealing with data items which have multiple labels. The learner must maximally predict the correct labels and minimally predict the incorrect labels, and we have modified the pseudoloss in a straightforward manner to reflect this:

$$\text{ploss}_q \doteq \frac{1}{2}\left(1 - \frac{\sum_{y \in Y_i} h(x_i, y)}{|Y_i|} + \sum_{y \in Y - Y_i} q(i,y)h(x_i,y)\right). \tag{5.2}$$

5.3 MLBoost

Using this modified pseudoloss as a basis, our modified form of Adaboost, named **MLBoost**, is presented as algorithm 1. It should be noted that there are two

Algorithm 1 MLBoost

1: **Input:**
   1. documents $\langle (x_1, Y_1), \ldots, (x_N, Y_N) \rangle$ with labels $Y_i \subset Y = 1, \ldots, k$
   2. distribution D over the documents
   3. multi-label weak learning algorithm **MultiWeakLearn**
   4. integer T specifying number of iterations
   5. training speed constant K
2: **Initialize** the weight vector: $w_{i,y}^1 = 1$
3: **for** $t = 1$ to T **do**
4: Set $W_i^T = \sum_{y \notin Y} w_{i,y}^t$;

$$q_t(i, y) = \frac{w_{i,y}^t}{W_i^t}$$

for $y \notin Y_i$; and set

$$D_t(i) = \frac{W_i^t}{\sum_{i=1}^N W_i^t}$$

5: Call **MultiWeakLearn**, providing it with the distribution D_t and label weighting function q_t; get back hypothesis $h_t : X \times Y \to [0, 1]$.
6: Set ϵ_t equal to the pseudoloss of h_t (see equation 5.2)
7: Set $\beta_t = \frac{\epsilon_t}{(1-\epsilon_t)}$.
8: Produce an annotation set A_i for each x_i where $|A_i| = |Y_i|$ and contains the most likely labelling elements y with regard to h_t.
9: Set the new weights vector to be

$$w_{i,y}^{t+1} = \begin{cases} K w_{i,y}^t & y \in A_i \ominus Y_i \\ \frac{w_{i,y}^t}{K} & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, N, y \in Y - Y_i$.
10: **end for**
11: **Output** the hypothesis

$$h_f(x, y) = \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t} \right) h_t(x, y)$$

.

novel and key differences between this algorithm and Adaboost.M2. As previously mentioned, the first difference is the modification to the calculation of pseudoloss. The second difference is that the recalculation of the weight vector has been modified to incorporate multi-label output. The weight vector is updated based on whether a particular label is a member of the symmetric difference between the correct annotation for a particular input, Y_i, and the annotation of that input, A_i, which is the set of elements y for which h_t is maximally confident, as done by Schapire and Singer in [81]. For all labels y which are incorrectly given by the hypothesis in the annotation A_i, the document specific weight, $w_{i,y}$ increases by a factor of K, the training speed constant. For all other y, the weight decreases by a factor of K. In this way, documents which are consistently labelled correctly decrease in training significance exponentially while documents which are consistently annotated incorrectly increase in training significance. Also, those which are consistently annotated incorrectly with the same words increase in training significance exponentially. In our experiments we used a value for K of $N^{1/T}$, where N is the number of training documents. Our choice of K was made such that it was difficult for a single document to dominate all training by being annotated incorrectly every time with the same word until the end of the training run.

In keeping with the original form of this algorithm, the learner to be boosted, **MultiWeakLearn**, need only produce hypotheses with pseudoloss consistently below or above 1/2 with respect to the distribution over the documents D and the label weighting function q with which it was presented. However, **MultiWeakLearn** must be able to learn from multi-label data. Because of this, the candidate learning systems tend to be far more complex than in the supervised case, where often simple decision stumps are all that is required [91]. Previous attempts at boosting more complex learners for speech recognition [101] have shown that boosting is still useful in these circumstances, though the resulting array of hypotheses can be time- and space-intensive for calculation and storage, respectively. In our initial exploration of this topic we disregarded this aspect in favor of exploring the usefulness of the technique, however future work will need to concentrate on ways to improve performance and decrease time and space complexity.

5.3.1 MultiWeakLearn

Before we could use this boosting framework to create a vision system, we had to produce a candidate for **MultiWeakLearn**. Our candidate for **MultiWeakLearn** in this paper is a novel modification to the method used by Barnard et. al in [3] and described in section 4.2.1. In the learning equations, we make several key insertions to bias the learning towards subsets of the data and against particular labels as determined by the distribution D and label weighting function q provided by the boosting system as described in algorithm 1.

We use D to influence the clustering of blobs by giving more weight to blobs in emphasized documents during Maximization. $D(i)$, is added as an addition weight to $P(c|b)$ in equations 4.7 and 4.8. We use q, or rather the

value $1 - q_t(w, i)$ to deemphasize incorrect labels when calculating $P(c|w, B)$ in the Expectation (specifically, equation 4.5).

This system provides a hypothesis to the booster which calculates a vocabulary distribution for a document using a novel weighted voting scheme. We introduced this scheme due to problems inherent in the original annotation system provided by Barnard et. al, which would produce a $P(w|B)$ as

$$P(w|B) \propto \sum_b P(w|c)P(c|B). \tag{5.3}$$

Imagine an image of a plane flying in the sky. The blue segments of sky give a large contribution to the solid blue, "sky" concept. The one or two segments depicting the plane give a smaller contribution to the textured gray, "plane" concept. The resulting annotation is heavily biased towards the first concept and contains words like "sky" and "water" but not "plane", even though that segment was correctly labelled.

We propose the following solution: for each blob, the words in the vocabulary are ranked with regard to their likelihood as a labelling for that blob as determined by $P(w, b)$. Then a vote is cast for each word with weight $v_w = \frac{1}{2^{r_w}}$, where r_w is the rank of a particular word (i.e. $r_w = 0$ for the first ranked y and $|Y| - 1$ for the last). The resulting collection of votes is then normalized to produce a distribution $P(w|B)$ over the vocabulary.

5.4 Results

We trained the algorithm using a subset of 1881 images from the Corel database. The state-of-the-art system used for comparison is an unmodified implementation of model **I-2** from [3] with linear topology. Our system, **MLBoost**, was implemented as shown in algorithm 1. The models were evaluated on two tasks: annotation and labelling. Each was trained on roughly 90% of the data, or 1667 images, and a test set of the remaining 214 images was used for evaluation of the model's generalization ability.

5.4.1 Annotation

Annotation provides a straightforward means to determine the effectiveness of the models under study. The baseline for performance is difficult to set; however, in [3] a word frequency distribution extracted from the training data has proven to be a rough equivalent of the "oblivious" algorithms used elsewhere in the literature for a similar purpose. This annotation simply consists of the top N words according to usage, where N is the size of the correct annotation for an image. Due to the nature of the Corel database, the keywords provided as correct annotations in our data contain many of the same words (e.g. "rock", "sky", "close-up"), allowing such a method of annotation to do quite well. To perform better than this, a learner must be gaining knowledge from

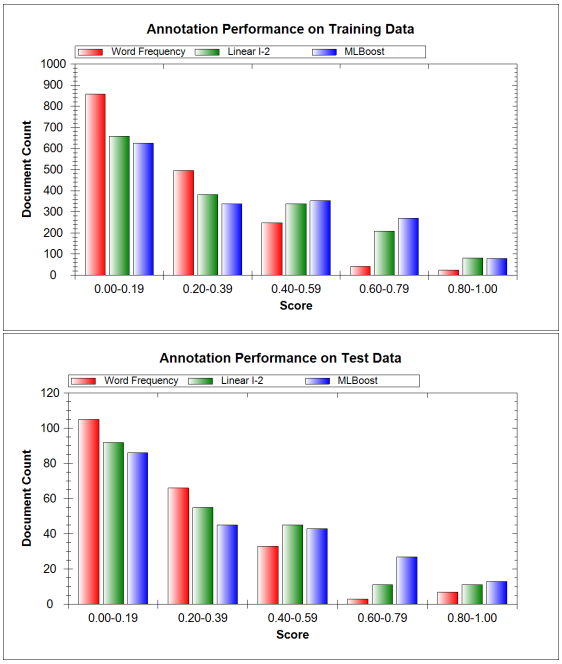

Figure 5.1: *Annotation Performance on training and test data.* The training subset consisted of 1667 images, or 90% of the data, with the test subset consisting of the remaining 10%, or 214 images. The data for word frequency comes from simply producing an annotation of the five most common words in the training data set for each image. The data from **I-2** comes from our implementation of the linear form of the model in [3]. As expected, **MLBoost** has fewer "bad" annotations (i.e. the first two bins), and the stressing of overall performance in the algorithm introduced by taking the mean of the confidence scores for the correct labels results in more "good" annotations (i.e. the last three bins), particularly those which are almost entirely correct. Score is determined as the number of correctly annotated words from an algorithm divided by the number of keywords for an image.

co-occurrence with image data, not simply word usage statistics, and as such it serves well as a baseline.

Annotations are produced as described in section 5.3.1. The annotation performance is shown in figure 5.1 for annotations produced using the word frequency and the two models. As expected, **MLBoost** has fewer "bad" annotations (i.e. the first two bins), and the stressing of overall performance in the algorithm introduced by taking the mean of the confidence scores for the correct labels results in more "good" annotations (i.e. the last three bins, particularly the second to last bin.)

5.4.2 Labelling

Labelling is a difficult task as it is essentially object recognition. The segments produced by normalized cuts tend not to be optimal for the task, though this problem is not endemic to the algorithm as automatic segmentation remains an open problem. That aside, the system produced by Barnard et al. is capable of performing rather well on the task, and as such we can boost that performance. Automatic evaluation of the task is impossible, as the matching between segments and keywords is an unknown quantity. In order to do a thorough evaluation of labelling performance, hand labelling with multiple labels of each segment in every image would have to be performed. While such an evaluation is desirable, it is not within the scope of this paper.

Presented in figure 5.2 are 3 examples each from **I-2** on the left and from **MLBoost** on the right. The automatic segmentations of the images are shown in red, and the top-scoring label for each of the 10 largest segments is displayed. The five words at the bottom of each image are the annotating words produced for the image ordered from left to right in terms of likelihood, with bold font indicating those which are correct. It should be noted that the correct annotations did not always have 5 words, though 5 was the maximum. All five maximally-predicted words are shown to give insight into the nature of the distribution. Note that **I-2** is achieving annotation performance by taking the easiest path in terms of segment labelling, whereas MLBoost is correctly labelling the segments and achieving equal or better annotation performance on the same image input.

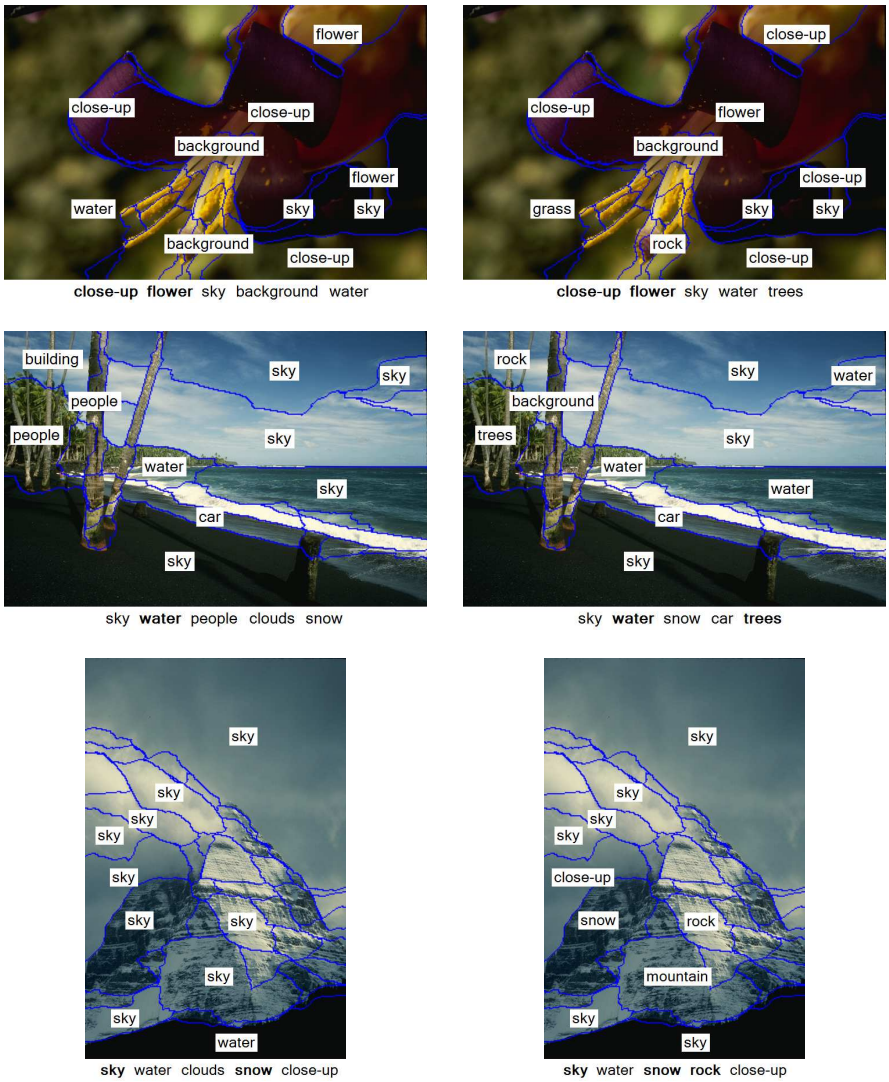

Figure 5.2: *Labelling performance.* The labellings on the left come from model **I-2**, on the right from **MLBoost**. The automatic segmentations of the images are shown in red, and the top-scoring label for each of the 10 largest segments is displayed. The five words at the bottom of each image are the annotating words produced for the image ordered from left to right in terms of likelihood, with bold font indicating those which are correct. Note that **I-2** is achieving annotation performance by taking the easiest path in terms of segment labelling, whereas MLBoost is correctly labelling the segments and achieving equal or better annotation performance on the same image input.

Chapter 6

Conclusion

In this chapter, we will take a look at the contents of the report and what they indicate as a direction for future research in the field of image retrieval. We will look for weaknesses in the current state of the art approaches and display proposals for strengthening current systems that will form the core of the research taking place over the next two years culminating in a completed PhD and a new range of systems for concept-enabled image retrieval.

6.1 Summary

We began this report by pursuing an in depth study of the state of the art in image retrieval. Beginning with various ways to extract image content, we discovered that global color alone can't be depended on for consistency and must be treated carefully. We found that there are many different ways of describing texture but that the innovation of the texton improves all of them and turns texture into an effective descriptor. Finally, we found that shape descriptors are yet to become truly useful representations of image content. We explored different methods of finding reliable local interest points and performed a study of the very successful SIFT descriptor.

Once we were familiar with content, its uses and its properties, we moved on to the problems associated with image retrieval. We found that just as there are many ways of describing image content there are many ways of organizing that content for storage, including histograms, Gaussian distributions and collections of local features. We then explored the problems inherent in defining distance for non-Euclidean entities, looking at a variety of metrics. Once metrics were understood, we moved on to the problem of finding the nearest neighbor in high dimensional spaces, learning of the ways to use KD Trees and the new Locality Sensitive Hashing technique to solve this problem. We looked at browsing as well as a valid alternative to automatic search. Finally, we explored a working image retrieval system built in the vein of Lowe's image retrieval system, which used SIFT and LSH to index into a collection of images and return candidate

matches for a query. We noted that, though the system had good results, it is limited in its scope to relatively small numbers of images due to the curse of dimensionality.

Having reached the state of the art in image retrieval, we began to look at ways of learning image concepts by treating object recognition as machine translation. We looked at different statistical models for use in extracting concepts, and then at ways concept can be used for searching. Given the problems inherent with the current generation of cluster-based concept learning systems, we saw the need to find ways to boost the performance of these algorithms. We learned, however, that there were no boosting frameworks for multi-label learners, motivating the creation of a new boosting framework, **MLBoost**.

6.2 HUGINN: A Concept-Based Image Retrieval Framework

In Norse mythology, Odin's two ravens Huginn and Muninn, "thought" and "memory", searched the world for information which he brought back to Odin at the end of every day. Named in honor of the thought that retrieved information for that monocular deity, the HUGINN framework will integrate the use of concepts to search databases for images. Though the current state-of-the-art systems, specifically those based on SIFT such as [52], are very effective, they are limited in their applicability by an upper bound imposed by the curse of dimensionality, having the effect that with very large databases several indistinguishable results are returned for a query image. This is a situation that is bound to occur with any purely content-based system. Though the descriptors used are rich and particular, they remain very elaborate permutations of "bigger than a bread box"-type qualifications that, however descriptive, are meaningless if no underlying concept is understood.

We propose that we use the major concepts in the database to create "views" of the data space which are biased towards that concept. Each concept is used to weight the features from an image according to the $P(c|I)$ for the purpose of performing a dimensionality reduction as in [12, 29, 96]. These "views" create an embedding in a lower dimensionality which retains accurate nearest neighbor for those entries which are strongly relevant to the particular concept. The use of such a reduction is two-fold: first, it reduces the number of dimensions used for the feature descriptor and thus increases the efficiency and accuracy of distance calculations and it increases the accuracy of approximate nearest neighbor algorithms by creating clearer point position, thus reducing the increase of complexity imposed by the curse of dimensionality.

Aside from practical concerns, this new embedding provides exciting possibilities for image browsing and customization of image database solutions for different clientele requirements. For example, for an art database, the interests of clientele looking to buy art at auction is much different from the interests of school children doing reports on the great artists. As such, the concepts used

to provide views into the database for both client sets could be customized to increase productivity.

For image retrieval, the user provides images and or words which describe the quantity they are searching for. HUGINN can then automatically extract concepts from both the words and the images and search the appropriate spaces for nearest neighbors and text matches, returning a range of results proportional to the concept prior.

Thus, the embedding has a three-fold effect: increasing the speed and efficiency of the searching algorithms, providing customized database portals for browsing, and providing concept-based retrieval. The actual retrieval mechanism used in the framework is open to change, as is the model used to extract concept. With the current state of the art, the retrieval system used would be SIFT based with LSH nearest-neighbor retrieval similar to [52], and the concept learning and extraction system would be the **MLBoost** version of the **I-2** model from [3]. As new feature descriptors and models are developed, however, the HUGINN system can incorporate them freely.

6.3 Future Work

The HUGINN framework and its formalization are the immediate concern of our research. In particular, a proof that a weighted dimensionality reduction will increase the accuracy of approximate nearest neighbor algorithms on searches with a high score for an embedding's concept must be formed. In addition, new ways of extracting concept must be found.

6.3.1 Learning Better Concepts with MLBoost

The main motivation behind **MLBoost**, aside from improving the performance of multi-label learners, was to allow for the intelligent combination of different learners to improve coverage of the feature space. We believe that current systems which perform learning based on large conglomerate vectors which combine color, texture, and shape lose their ability to emphasize particular cues on a concept basis. As such, we want to experiment with combining multiple learners with smaller, more focused feature vectors to learn different object classes and concepts. Another use of the **MLBoost** system is in the combination of different kinds of content. For example, three texture learners based on texton histograms, oriented filter banks, and wavelets could be employed and the booster could combine them to gain the advantages of all three methods.

6.3.2 Stable Feature Groups

One of the problems which plagues image retrieval and matching systems that are based on local feature points is that when adding reference images to the database, it is uncertain which of the detected feature points will be most useful. As such, hundreds to thousands of points are included in the database, in the

hope that the correct ones are among them. This results in large databases in which the majority of the points will be unused and simply clutter up the feature space, resulting in poorer nearest neighbor performance and an inconvenient upper bound on the number of reference images which can be reliable stored and retrieved. In addition, the reference images often need to be of good quality and high resolution to produce the maximum number of points.

We propose that instead of using a single, high quality still image, a sequence of images of a reference object can be used instead, in which the feature points in each frame are detected and registered with the other frames in an attempt to determine which are most stable, i.e. which turn up most often. In addition, instead of storing simply the descriptor, we will store the mean and variance of a multi-variate Gaussian with diagonal covariance over the chosen feature point, thus allowing for a match certainty to be determined instead of a simple Euclidean distance test during matching, which should result in a distinct improvement in nearest neighbor performance.

In addition to finding stable points, we will try to determine groupings of stable and unstable points which occur together and in the same location. In this way, when that stable point is found we can add certainty to the match by looking for the unstable points nearby. It is possible that similar groupings are shared between objects, thus reducing again the number of points which need to be stored in the database and adding increased stability against occlusion and articulation (i.e. objects with moving parts). In addition to noting unstable interest points which co-occur with stable points, segments of silhouette near the grouping could also be stored to incorporate local shape information that could be compared against an Chamfer edge map of the query image.

6.3.3 Provisional Timetable

Naturally, this research is still under way and many aspects may change and evolve as new avenues for exploration come to the fore and the state of the art proceeds in new directions. Table 6.1 is a provisional timetable for the completion of the PhD by September, 2007. This will serve as both a collection of deadlines, a constraint on tangential exploration, and a manner of tracking progress as this research moves towards its conclusion: a state of the art and revolutionary image retrieval framework.

Table 6.1: Provisional Timetable for timely completion of PhD.

Dates	Description of Work
October - December 2006	Stable Feature Groups
January - March 2006	Formalize `HUGINN` Framework
March - May 2006	Build Image Retrieval component for `HUGINN`
May 31st 2006	Deadline: Complete Image Retrieval System
May - October 2006	User studies for Browsing system
October - January 2007	Experiments with Combining Different Learner Types using MLBoost
January - March 2007	Build `HUGINN` Browsing system
March 31st 2007	Deadline: Complete Image Browsing System
March - May 2007	Work on optimization, a clean interface and presentation
May 31st 2007	Deadline: Final System
May - September 2007	Write Thesis

Bibliography

[1] Khaled Alsabti, Sanjay Ranka, and Vineet Singh, *An efficient K-means clustering algorithm*, Proceedings of the IPPS/SPDP Workshop on High Performance Data Mining, 1998.

[2] K. Arbter, W.E. Snyder, H. Burkhardt, and G. Hirzinger, *Application of affine-invariant fourier descriptors to recognition of 3d objects*, IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990), 640–647.

[3] Kobus Barnard, Pinar Duygulu, Nando de Freitas, David Forsyth, David Blei, and Michael I. Jordan, *Matching words and pictures*, Journal of Machine Learning Research **3** (2003), 1107–1135.

[4] Kobus Barnard, Pinar Duygulu, and David Forsyth, *Clustering art*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, 2001, pp. 434–441.

[5] Kobus Barnard and David Forsyth, *Learning the semantics of words and pictures*, Proceedings of the International Conference on Computer Vision, vol. 2, 2001, pp. 408–415.

[6] Jeffrey S. Beis and David G. Lowe, *Shape indexing using approximate nearest-neighbor search in high-dimensional spaces*, Proceedings of the Conference on Computer Vision and Pattern Recognition, 1997, pp. 1000–1006.

[7] Serge Belongie, Chad Carson, Hayit Greenspan, and Jitendra Malik, *Color- and texture-based image segmentation using EM and its application to content-based image retrieval*, Proceedings of the Sixth International Conference on Computer Vision, 1998, p. 675.

[8] Serge Belongie, Jitendra Malik, and Jan Puzicha, *Shape matching and object recognition using shape contexts*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 4, 509–522.

[9] J. Bergen and E. Adelson, *Early vision and texture perception*, Nature (London) **333** (1988), 363–364.

[10] J. Bilmes, *A gentle tutorial on the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models*, Tech. Report ICSI-TR-97-021, University of Berkeley, 1997.

[11] D. Blei, A. Ng, and M. Jordan, *Latent dirichlet allocation*, Journal of Machine Learning Research **3** (2003), 993–1022.

[12] I. Borg and P. J. F. Groenen, *Modern multi-dimensional scaling*, Springer, New York, NY, 1997.

[13] Yuri Boykov, Olga Veksler, and Ramin Zabih, *Fast approximate energy minimization via graph cuts*, Proceedings of ICCV '99, 1999, pp. 377–384.

[14] A. Broder, S. Glassman, M. Manasse, and G. Zweig, *Syntactic cluster of the web*, Proceedings of the Sixth International World Wide Web Conference, 1997, pp. 391–404.

[15] M. Brown and D. G. Lowe, *Invariant features from interest point groups*, Proceedings of BMVC '02 (Cardiff, Wales), 2002, pp. 656–665.

[16] P. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer, *The mathematics of statistical machine translation: parameter estimation*, Computational Linguistics **32** (1993), no. 2, 262–311.

[17] John Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence **8** (1986), no. 6, 679–698.

[18] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein, and Jitendra Malik, *Blobworld: A system for region-based image indexing and retrieval.*, Proceedings of VISUAL 1999, 1999, pp. 509–516.

[19] Dorin Comaniciu and Peter Meer, *Mean shift: A robust approach toward feature space analysis*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 5, 603–619.

[20] T. Cootes and C. J. Taylor, *Statistical models of appearance for computer vision*, Available on author's website: `http://www.isbe.man.ac.uk/`, 2004.

[21] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham, *Training models of shape from sets of examples*, Proc. British Machine Vision Conference (Berlin), Springer, 1992, pp. 266–275.

[22] Ingemar J. Cox, Matt L. Miller, Thomas P. Minka, Thomas V. Papathomas, and Peter N. Yianilos, *The Bayesian image retrieval system, PicHunter: Theory, implementation, and psychophysical experiments*, IEEE Transctions on Image Processing **9** (2000), no. 1, 20–37.

[23] Oana G. Cula and Kristin J. Dana, *Compact representation of bidirectional texture functions*, Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, vol. 1, December 2001, pp. 1041–1047.

[24] John D. Daugman, *Complete discrete 2-d Gabor transforms by neural networks for image analysis and compression*, IEEE Transactions on Acoustics, Speech, and Signal Processing **36** (1988), 1169–1179.

[25] G. Van de Wouwer, P. Scheunders, and D. Van Dyck, *Statistical texture characterization from wavelet representations*, IEEE Transactions on Image Processing **8** (1999), no. 5, 592–598.

[26] A.P. Dempster, N.M. Laird, and D.B. Rubin, *Maximum likelihood from incomplete data via the em algorithm*, Journal of the Royal Statistical Society. Series B (Methodological) **39** (1977), no. 1, 1–38.

[27] Pinar Duygulu, Kobus Barnard, Nando de Freitas, and David Forsyth, *Object recognition as machine translation: Learning a lexicon for a fixed image vocabulary*, Proceedings of ECCV2002 (Copenhagen, Denmark), May 2002.

[28] Alexei A. Efros and Thomas K. Leung, *Texture synthesis by non-parametric sampling*, Proceedings of ICCV '99, 1999.

[29] C. Faloutsos and K. Lin, *Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets*, Tech. Report CS-TR-3383, University of Maryland, January 1994.

[30] G. Fan and X.-G. Xia, *Wavelet-based texture analysis and synthesis using hidden markov models*, IEEE Transactions on Circuits and Systems-Part I **50** (2003), no. 1, 106–120.

[31] I. Fogel and D. Sagi, *Gabor filters as texture discriminators*, Biological Cybernetics **61** (1989), 103–113.

[32] Yoav Freund and Robert E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, Journal of Computer and System Sciences **55** (1997), no. 1, 119–139.

[33] K. Fukunaga and L.D. Hostetler, *The estimation of the gradient of a density function, with applications in pattern recognition*, IEEE Transactions on Information Theory **21** (1975), 32–40.

[34] Prasad Gabbur, *Quantitative evaluation of feature sets, segmentation algorithms and color constancy algorithms using word prediction*, Master's thesis, University of Arizona, 2003.

[35] D. Gabor, *Theory of communication*, Journal of the IEE **93** (1946), 429–457.

[36] S. Geman and D. Geman, *Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images.*, Journal of Pattern Analysis and Machine Intelligence **6** (1984), 721–741.

[37] Bogdan Georgescu, Ilan Shimshoni, and Peter Meer, *Mean shift based clustering in high dimensions: A texture classification example*, Proceedings of ICCV '03, 2003.

[38] A. Gionis, P. Indyk, and R. Motwani, *Similarity search in high dimensions via hashing*, Proceedings of the International Conference on Very Large Databases, 1999.

[39] Richard W. Hamming, *Error-detecting and error-correcting codes*, Bell System Technical Journal **29** (1950), no. 2, 147–160.

[40] Chris Harris and Mike Stephens, *A combined corner and edge detector*, Proceedings of The Fourth Alvey Vision Conference, 1988, pp. 147–151.

[41] F. L. Hitchcock, *The distribution of a product from several sources to numerous localities*, Journal of Mathematical Physics **20** (1941), 224–230.

[42] Thomas Hofmann and Jan Puzicha, *Statistical models for co-occurrence data*, Tech. report, Massachusetts Institute of Technology, 1998.

[43] Piotr Indyk and Rajeev Motwani, *Approximate nearest neighbors: towards removing the curse of dimensionality*, Proceedings of the 30th Symposium on Theory of Computing, 1998, pp. 604–613.

[44] A. Jain and A. Vailaya, *Image retrieval using color and shape*, Pattern Recognition **29** (1996), no. 8, 1233–1244.

[45] S. Jain, A. Joglekar, and D. Bote, *Isearch: A content-based image retrieval (cbir) engine*, Master's thesis, Pune University, 2002.

[46] Matthew Johnson and Roberto Cipolla, *Improved image annotation and labelling through multi-label boosting*, Proceedings of BMVC '05, 2005.

[47] I. T. Joliffe, *Principal component analysis*, Springer-Verlag, 1986.

[48] D. Jones and J. Malik, *Computational framework to determining stereo correspondence from a set of linear spatial filters*, Image and Vision Computing **10** (1992), no. 10, 699–708.

[49] J.P. Jones and L.A. Palmer, *An evaluation of the two-dimensional Gabor filter model of simple receptive fields in cat striate cortex*, Journal of Neurophysiology **58** (1987), no. 6, 1233–1258.

[50] B. Julesz, *Textons, the elements of texture perception, and their interactions*, Nature **290** (1981), no. 5802, 91–97.

[51] Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu, *An efficient k-means clustering algorithm: Analysis and implementation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **24** (2002), no. 7, 881–892.

[52] Y. Ke, R. Sukthankar, and L. Huston, *Efficient near-duplicate detection and sub-image retrieval*, Proceedings of ACM Multimedia, 2004.

[53] Yan Ke and Rahul Sukthankar, *Pca-sift: A more distinctive representation for local image descriptors*, Proceedings of CVPR '04, 2004.

[54] S. Kullback, *Information theory and statistics*, Dover, New York, NY, 1968.

[55] T. Leung and J. Malik, *Recognizing surfaces using three dimensional textons*, Proceedings of IEEE International Conference on Computer Vision '99 (Corfu, Greece), 1999.

[56] Thomas Leung and Jitendra Malik, *Representing and recognizing the visual appearance of materials using three-dimensional textons*, International Journal of Computer Vision **43** (2001), no. 1, 29–44.

[57] Ting Liu, Andrew Moore, Alexander Gray, and Ke Yang, *An investigation of practical approximate nearest neighbor algorithms.*

[58] S.P. Lloyd, *Least squares quantization in PCM*, IEEE Transactions on Information Theory **28** (1982), 129–137.

[59] David G. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision **60** (2004), no. 2, 91–110.

[60] Jitendra Malik and Pietro Perona, *Preattentive texture discrimination with early vision mechanisms*, Journal of Optical Society of America A **7** (1990), no. 5, 923–932.

[61] S.G. Mallat, *A theory of multiresolution signal decomposition: The wavelet representation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **11** (1989), no. 7, 674–693.

[62] S. Marcelja, *Mathematical description of the response of simple cortical cells*, Journal of Optical Society of America, A **70** (1980), no. 11, 1297–1300.

[63] Bruce A. Maxwell and Stephanie J. Brubaker, *Texture edge detection using the compass operator*, Proceedings of BMVC '03, 2003.

[64] K. Mikolajczyk, *Detection of local features invariant to affine transformations*, Ph.D. thesis, Institut National Polytechnique de Grenoble, France, 2002.

[65] K. Mikolajczyk and C. Schmid, *A performance evaluation of local descriptors*, Proceedings of CVPR '03, June 2003.

[66] A. Mojsilovic, M. V. Popovic, and D. M. Rackov, *On the selection of an optimal wavelet basis for texture characterization*, IEEE Transactions on Image Processing **9** (2000), no. 12, 2043–2050.

[67] Hans Moravec, *Obstacle avoidance and navigation in the real world by a seeing robot rover*, Tech. Report CMU-RI-TR-3, Carnegie-Mellon University, September 1980.

[68] Wayne Niblack, Ron Barber, William Equitz, Myron Flickner, Eduardo H. Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin, *The QBIC project: querying images by content using colour, texture and shape*, SPIE Proceedings of Storage and Retrieval for Image and Video Databases, 1993, pp. 173–187.

[69] J. A. Noble, *Descriptions of image surfaces*, Ph.D. thesis, University of Oxford, 1989.

[70] V. Ogle and M. Stonebraker, *Chabot: Retrieval from a relational database of images*, IEEE Computer **28** (1995), 40–48.

[71] E. Persoon and K.S. Fu, *Shape discrimination using fourier descriptors*, IEEE Transactions on Systems, Man and Cybernetics **SMC-7** (1977), no. 3, 170–179.

[72] Daniel A. Pollen and Steven F. Ronner, *Phase relationship between adjacent simple cells in the visual cortex.*, Science **212** (1981), 1409–1411.

[73] J. Portilla and E. P. Simoncelli, *A parametric texture model based on joint statistics of complex wavelet coefficients*, International Journal of Computer Vision **40** (2000), no. 1, 49–71.

[74] J. Puzicha, T. Hofmann, and J. M. Buhmann, *Non-parametric similarity measures for unsupervised texture segmentation and image retrieval*, Proceedings of CVPR '97, June 1997.

[75] L. R. R. Rabiner and B. H. Juang, *Fundamentals of speech recognition*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[76] S. T. Rachev, *The Monge-Kantorovich mass transference problem and its stochastic applications*, Theory of Probability and its Applications **29** (1984), no. 4, 647–676.

[77] O. Rioul, *A discrete-time multiresolution theory*, IEEE Transactions on Signal Processing **41** (1993), no. 8, 2591–2606.

[78] O. Rioul and M. Vetterli, *Wavelets and signal processing*, IEEE Signal Processing Magazine **8** (1991), no. 4, 14–38.

[79] Yossi Rubner, Carlo Tomasi, and Leonidas Guibas, *The earth mover's distance as a metric for image retrieval*, Tech. Report STAN-CS-TN-98-86, Computer Science Department, Stanford University, September 1998.

[80] M. Ruzon and C. Tomasi, *Edge, junction, and corner detection using color distributions*, IEEE Transactions on Pattern Analysis and Machine Intelligence **23** (2001), no. 11, 1281–1295.

[81] Robert Schapire and Yoram Singer, *Improved boosting algorithms using confidence-rated predictions*, Machine Learning **37** (1999), no. 3, 297–336.

[82] Cordelia Schmid, *Constructing models for content-based image retrieval*, Journal of Computer Vision and Pattern Recognition **2** (2001), 39–45.

[83] Stephen Se, David Lowe, and Jim Little, *Global localization using distinctive visual features*, Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems (EPFL, Lausanne, Switzerland), October 2002.

[84] Cián Shaffrey, *Multiscale techniques for image segmentation, classification and retrieval*, Ph.D. thesis, University of Cambridge, 2003.

[85] J. Shi and J. Malik, *Normalized cuts and image segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **22** (2000), 888–905.

[86] Jianbo Shi and Carlo Tomasi, *Good features to track*, Proceedings of CVPR '94 (Seattle), June 1994.

[87] Joset Sivic, Bryan C. Russell, Alexei A. Efros, Andrew Zisserman, and William T. Freeman, *Discovering object categories in image collections*, Tech. Report AIM-2005-005, Massachusetts Institute of Technology, February 2005.

[88] J. R. Smith and S. F. Chang, *VisualSEEk: a fully automated content-based image query system*, Proceedings of the ACM Internation Conference on Multimedia (ACM-MM) (Boston, MA), November 1996, pp. 86–98.

[89] M. Stricker and M. Orengo, *Similarity of color images*, Proceedings of the SPIE Conference on Storage and Retrieval for Image and Video Databases III, vol. 2420, February 1995, pp. 381–392.

[90] M. Swain and D. Ballard, *Color indexing*, International Journal of Computer Vision **7** (1991), no. 1, 11–32.

[91] Antonio Torralba, Kevin P. Murphy, and William T. Freeman, *Sharing features: efficient boosting procedures for multiclass object detection*, Proceedings of CVPR '04, 2004.

[92] M. Turner, *Texture discrimination by Gabor functions*, Biological Cybernetics **55** (1986), 71–82.

[93] M. Unser, *Texture classification and segmentation using wavelet frames*, IEEE Transactions on Image Processing **4** (1995), no. 11, 1549–1560.

[94] M. Varma and A. Zisserman, *Texture classification: Are filter banks necessary?*, Proceedings of the CVPR '03, 2003.

[95] Manik Varma and Andrew Zisserman, *Classifying images of materials: Achieving viewpoint and illumination independence*, Proceedings of ECCV'02, 2002, pp. 255–271.

[96] P. Wu, B. S. Manjunath, and H. D. Shin, *Dimensionality reduction for image retrieval*, Proceedings of the International Conference on Image Processing (Vancouver, Canada), September 2000.

[97] Richard A. Young, Ronald M. Lesperance, and W. Weston Meyer, *The gaussian derivative model for spatial-temporal vision: I. corticla model*, Spatial Vision **14** (2001), no. 3,4, 261–319.

[98] C.T. Zahn and R.Z. Roskies, *Fourier descriptors for plane closed curves*, IEEE Transactions on Computers **21** (1972), no. 3, 269–281.

[99] D. S. Zhang and G. Lu, *A comparative study on shape retrieval using fourier descriptors with different shape signatures*, Proceedings of the International Conference on Intelligent Multimedia and Distance Education (Fargo, North Dakota), June 2001, pp. 1–9.

[100] S. C. Zhu, Y. Wu, and D. Mumford, *Filters, random fields and maximum entropy*, International Journal of Computer Vision **27** (1998), no. 2, 1–20.

[101] Geoffrey Zweig and Mukund Padmanabhan, *Boosting gaussian mixtures in an LVCSR system*, Proceedings of ICASSP 2000, 2000.