Exploiting Low Level Image Segmentation for Object Recognition

Thomas Oskam

September 13, 2007

Abstract

There exist many approaches to object recognition of image data. Most of the methods use a top-down approach to classify the content, as low-level information is often considered inapplicable or insufficient for this task. The method presented in this work, however, shows a way to exploit low-level image segmentation for the purpose of categorizing different object classes of still images. The key idea is to not only use one single segmentation, but a whole set of different parameterized segmentations of an image as basis for feature extraction. Segment boundaries are used as paths along which strings of feature vectors are drawn. These strings are pairwise aligned to create a scoring matrix, which can be transformed into a Mercer kernel for a standard SVM based classifier.
Despite the inherit problems of low-level segmentations, this method performs very well on standard benchmark image databases and shows that segmentation indeed can be used for object recognition.

The first goal of this semester thesis was to create a Matlab framework that implements the proposed method for object recognition. With this framework as basis, several approaches to improve scoring results of pairwise alignments, and thus improve the categorization rate, need to be tested. Also the parameters used for the string alignment can have an impact on the performance as well. Therefore, the second goal was to examine different combinations of methods for feature extraction and string alignment with different parameter settings in order to find out which configurations lead to the highest retrieval rates.

Contents

List of Figures

1

Introduction

The goal of image segmentation is to divide a cluttered scene into regions with similar characteristics. Current segmentation techniques mostly use bottom-up approaches on the basis of local image properties. Attributes like texture, brightness, color, or motion are locally evaluated to detect coherent units. This locality, however, is also the biggest weakness of the approach: Segmentation depends essentially on the quality of these attributes. Poor data conditions like shadows, occlusions, or noise directly influence the performance of the segmentation. It is often the case, that the detected segments are not coherent with human perception of the image content. Figure 1.1 shows an example of a poor quality segmentation. Because of these inherent difficulties, the usefulness of low-level indentifying of objects in real world images is questionable, and the current best approaches indeed do not employ low-level image segmentation.

It also has been proposed to treat segmentation and recognition in an interleaved manner to avoid the above problems (see e.g. [Yu02]). Such approaches typically mix top-down and bottom-up

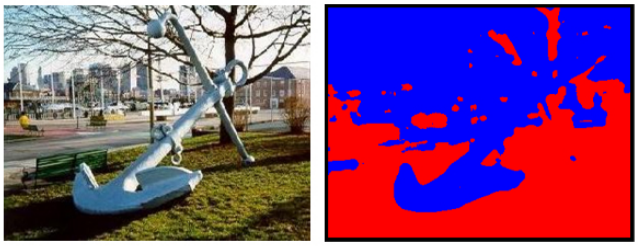

Figure 1.1: Problems of low-level segmentation. The white anchor in the foreground has a similar color and similar texture properties as a large part of the background. This leads to a wrong segmentation where most of the background is associated with the anchor.

strategies, based on back propagating hypotheses about the objects, down to the segmentation level. If the initial segmentation is of reasonable quality, which for some applications is true, these methods seem to work quite well. These meaningful segmentations can often be produced if other side information can be factored into the process. This is for example the case in video sequences, where the movement information can be used to determine areas that are coherent with objects.

For the task of segmenting still images, however, such additional information is usually not available, which limits the performance of finding coherent areas, in particular if there are many potential object classes that have to be distinguished from an image. Despite the generally poor quality of bottom-up approaches in real-world images, we show that it is possible to exploit segmentation for object recognition. The main idea is to use a whole collection of different segmentations of the same image to extract meaningful feature vectors to characterize the displayed object. A segmentation on poor image data, as stated above, will most likely not capture the complete silhouette of the displayed object. Some parts of the object boundary, however, will typically be found. A scoring matrix is computed by way of pairwise alignment of low-level feature strings, which are extracted along these different segment boundaries of a training image set. This matrix is then transformed into a Mercer kernel and used to train a standard SVM-based classifier.

The foundation for our work is laid by the principle of *compositionality* by [Ger98]. As observed in cognition, and especially in human vision, complex entities are perceived as compositions of comparably few, simple, and widely usable parts. Objects are represented based on their components and the relations between them. Using a variety of different segmentations of an image, we expect some of these parts to be captured per segmentation. By consistent multiple segmentations, we expect most of the object silhouette to be captured.

The main goals of this thesis are twofold: on the one hand, to provide a Matlab implementation of the object recognition approach in order to perform tests with different configurations and parameter settings. Both the feature extraction and the string alignment allow different strategies that impact the categorization result. On the other hand, this thesis focusses on testing the object recognition method with different approaches of feature extraction as well as with different alignment methods for the creation of a kernel matrix in order to improve retrieval rates.

The, remainder of this work is organized as follows: in chapter 2, the computation of a series of segmentations per image is discussed. This lays the foundation for the feature extraction, which is discussed in chapter 3. Chapter 4 covers the process of string alignment in order to derive kernel matrices. Several useful alignment methods are discussed in detail as well as the division of the data into training and test kernels. In chapter 5, a method is shown to transform the previously computed kernels into valid Mercer kernels that can be fed into a standard black-box kernel-classifier. Several tests are shown in chapter 6, where different approaches to extract features in combination with different parameter for the string alignment methods are compared. The results for these different test settings are presented and several observations are discussed in this chapter as well. Chapter 7 completes this thesis with an overall summary and conclusion.

2

Ensembles of Low-Level Segmentations

The general problem with image segmentation is to divide the image into connected regions that represent semantically equivalent parts. In most cases, this equivalence is not modeled explicitly, but rather dependent on several locally evaluated low-level features, such as color or texture. Therefore, segmentation methods have to deal with combining many different feature vectors in a meaningful way to construct these regions.

One popular solution, which is often used for this problem, is to simply stack the different feature vectors into one high-dimensional vector, as described in [Aga04]. Two different types of problems can arise with this approach: (i) Such grouping algorithms, on a technical site, can become increasingly unstable as the dimension of the stacked feature vector grows. This problem is usually related to the steep increase of local minima of the objective functions. (ii) On the other hand, the problem of relevance of different features for the segmentation task arises. A feature that is highly relevant for one particular setting (e.g. the texture of a structured surface like textile) may only contain useless noise in another setting (e.g. a lambertian surface) and vice versa.

Our approach to create an ensemble of different segmentations per image uses the solution proposed in [Rot04]. This method combines both ideas of partitioning and feature combination and selection.
The core idea of this algorithm is the automatic selection of features for a given image. It consists of a Gaussian mixture model with built-in relevance detection which selects the most important features by maximizing a constrained likelihood criterion. Model selection is performed by drawing *resamples* of the object set, training the segmentation model on the individual *resamples* and comparing the resulting solutions. See Figure 2.1 for a schematic demonstration.

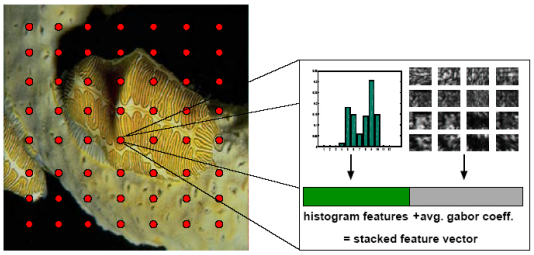

Figure 2.1: This figure shows a schematic overview of the image sites and the stacked feature vector.

Adapted to our image segmentation problem, this strategy translates into sampling different image sites, inferring a segmentation on the basis of these sites, and identifying stable segmentations (i.e. those which can be reproduced on many different random samples of image sites). This procedure is repeated for different numbers of mixture modes, and finally a stability-ranked list of prototypical segmentations is received.

Additionally to selecting stable segmentations, all individual segmentations are superposed to calculate a probabilistic boundary map, which encodes the probability for each pixel to be part of the object boundary. This boundary map is later used as one of the image features in the information extraction process. Chapter 3.2 discusses this in more depth.

3

Feature Extraction

The segmentation images S_i of an image I produced by the method proposed in [Rot04] are the foundation for the feature extraction. The parts of interest for the feature extraction are the patch boundaries $B_j(S_i) = \{(x_1, y_1), \ldots, (x_k, y_k)\}$. Information in form of different feature vectors is drawn from equidistant positions along the postprocessed boundary paths $B'_j(S_i)$ of the different segments and forms a set of strings that represent the object's properties for the given image.

3.1 Computation of meaningful paths

The extraction of the boundaries along each patch of a segmented image gives a set of paths B_j along which features are extracted. These paths, however, typically contain a lot of redundant and undesired positions. Therefore, the boundaries need to be post processed to improve the feature extraction.

The most undesired parts within the extracted boundaries are the positions along the image border. These regions are always considered as a part of a patch boundary as the segmentation step does not explicitly handle these cases. These pixels at image borders can lead to random feature vectors drawn inside a displayed object or background. In most cases, his information does not represent data from the object boundary, and therefore need to be removed from the boundary path B_j, see Figure 3.1.
The removal of a series of positions from a path leads to gaps which may induce further problems. The fact that alignment algorithms can handle such gaps, however, allows us to circumvent this problem in the future processing pipeline. The alignment of extracted feature strings is discussed in chapter 4.

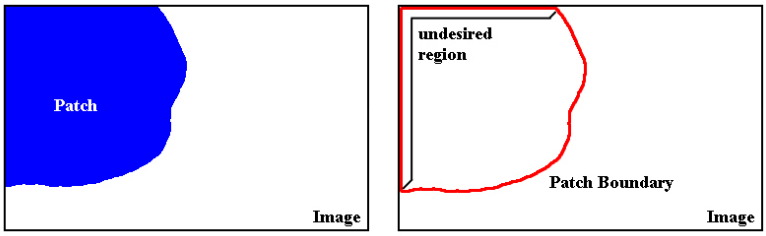

Figure 3.1: On the left side, a patch that is reaching over the image border is displayed. The region that the patch is representing is very likely to go beyond the image boundary. This leads to undesired positions in the patch boundary, which will lead to random feature information.

In images of poor quality, namely images with high texture and lighting differences, the segmentation process can lead to a lot of small cluttered regions. These contain, in most cases, noisy information of the background or small parts of the object. The smaller these patches are, the less relevant they are to capture the properties of the displayed object class. These paths can easily be removed by rejecting all boundaries, whose lengths are below a certain threshold.
A source of redundancy arises due to the fact that segment patches are disjoint. This leads to a lot of paths that follow the same way and therefore should to be removed as well. This, however, is not crucial to the recognition itself, and will only decrease the workload of the feature extraction.

As a result of the first post-processing, we obtain a set of boundary paths B_j^* in which redundant boundary paths or parts thereof have been sorted out. They are now further optimized such that the drawn feature strings are better suited for the later alignment process. The start point of a boundary path for example, which is random due to the boundary extraction algorithm, will have a certain impact on the alignment [1]. Therefore it is often helpful to rotate the position vectors of the path such that an extrema point (i.e. the position with the highest y value) is at the beginning. The rotated boundaries B_j^{*rot} are computed as follows:

$$B_j^* = \{(x_j, y_j), \ldots, (x_t, y_t), (x_{t+1}, y_{t+1}), \ldots, (x_k, y_k)\} \tag{3.1}$$

with

$$y_{t+1} = \underset{i}{\operatorname{argmax}} \{y_i\}, \tag{3.2}$$

$$B_j^{*rot} = \{(x_{t+1}, y_{t+1}), \ldots, (x_k, y_k), (x_1, y_1), \ldots, (x_t, y_t)\}. \tag{3.3}$$

Another problem is the only partial capture of boundaries. It can happen that a specific boundary is split due to the patch reaching the image boundary (See Figure 3.2 for a schematic demonstration) resulting in a too low score of the alignment against a path representing the same

[1] This, in fact, also depends on the alignment method. The start point will have a bigger impact on the result of a global alignment than a local alignment.

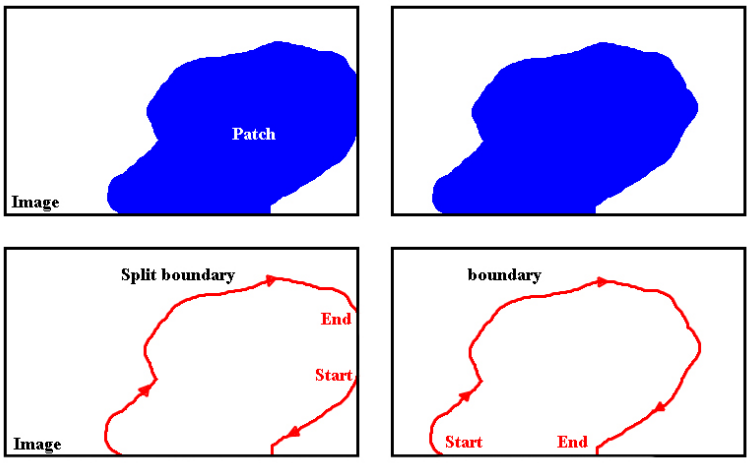

Figure 3.2: The top row shows two different segments of the same object. On the left side the patch reaches to the image border. The second row shows the resulting post processed boundary paths of these segments. The problem that arises is due to the left patch boundary, which is split once more than the right boundary. This can lead to a wrong order, meaning that the part of the path that is apart from the rest may be interpreted on the wrong end of the path and thus may lead to a too low score when aligned against the other.
This problem can be resolved by extending each boundary along itself, creating a region in the middle containing the correct order.

object silhouette, because such a partition does not fit for general alignments. The boundaries thus need to be extended along themselves to improve the stability:

$$B'_j = \{B_j^{*rot}, B_j^{*rot}\}. \tag{3.4}$$

After these steps, the boundary paths B_j are reduced to a set of positions B'_j that are suitable for the feature extraction. Figure 3.3 shows a schematic of the boundary post processing on a set of segmentations of a stop sign image.

3.2 Extracting Features

The post-processed boundaries B'_j are now ready to be used as paths for feature extraction. For the purpose of feature extraction, a polar scale P is created that divides a circular space into different segments with respect to a certain origin (Chapter 3.2.1 discusses polar scales in more depth). Each segment represents an area of interest over which a feature is averaged. The combination of all segments in a meaningful[2] way leads to a local feature vector. The polar scale is applied at equidistant points along each path resulting in a feature string for each boundary. Figure 3.4 shows an illustration of the feature extraction using a polar scale.

[2]In our implementation the feature vector is created by stacking the segments from the out most layer counter-clockwise to the innermost layer.

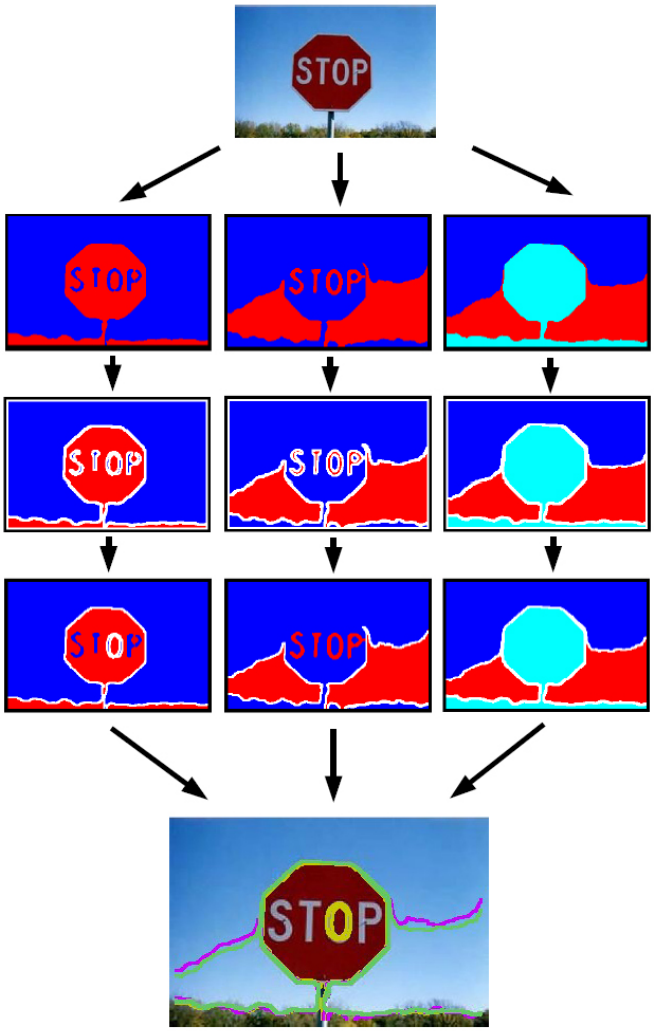

Figure 3.3: In this figure, you can see the different stages of the boundary extraction and post processing. The image on the top is the target. The three images of the second row are some the results of the segmentation step. The next row shows the image boundaries that arise before the post processing step. The fourth row shows the corresponding patch boundaries after the processing. The image on the bottom shows a superposition of the post-processed boundaries.

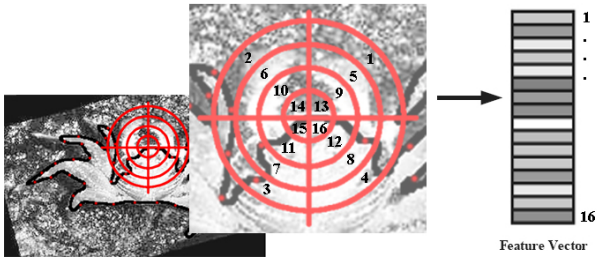

Figure 3.4: This is an example of extracting gray values from an image. The polar scale is applied along a (post processed) patch boundary, where the gray values are summed up for each segment. The resulting feature vector is used together with the other vectors along the boundary to form a feature string.

$$P = \{p_1, \ldots, p_l\} \text{ with } p_i = \{(x_{i1}, y_{i1}), \ldots, (x_{in}, y_{in}))\}, \tag{3.5}$$

$$v_k = \sum_{t=k1}^{kn} (F(b_{jx} + x_t, b_{jy} + y_t)) \text{ for } k = 1, \ldots, l. \tag{3.6}$$

P represents the polar scale, which contains a list of offset positions p_i for each segment i. The feature vector $V = [v_1, \ldots, v_l]^T$ is created by summing up the values in a feature matrix F at a chosen boundary position $b = (b_{jx}, b_{jy})$ from B'_j offset with the positions of p_i. This results in a feature vector V for each evaluated position on B'_j. Through the extraction of a feature vector at different equidistant positions b_i on B'_j a feature string $S_j = \{V_1, \ldots, V_n\}$ for each boundary is formed.

3.2.1 Polar Scales

As stated above, a polar scale divides a circular space into a set of segments with respect to an origin. A polar scale in practice can have two forms; Equidistant and logarithmic. The equidistant polar scale is defined as a radial scale whose sectors are equally distant with respect to the origin. In contrast, the logarithmic polar scale uses a logarithmic function to define the distance to the origin for each segment circle. Figure 3.5 shows each an example of an equidistant and a logarithmic polar scale.

We have tested the use of either polar scale to determine which one gives better results. Intuitively, the log polar scale seems to fit better for the problem as the information taken is correlated to the distance from the point of on the boundary. However it turned out that both performed almost equally well for the task of extracting feature vectors.

3.2.2 Features

Features which are extracted along each boundary path B'_j are stored in a matrix F. In our approach, we use two different features: (i) First, the color information of the original image

Figure 3.5: Different kinds of polar scales. On the left: equidistant 4-4 polar scale. On the right: logarithmic 4-4 polar scale.

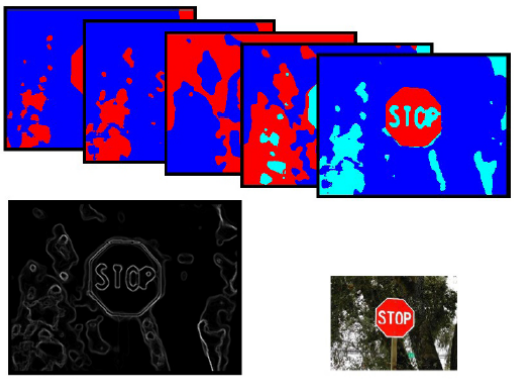

Figure 3.6: The probabilistic boundary map (bottom left image) is created by superposition of the boundaries of up to 100 different segmentations of the image.

is used. For this purpose the color image is transformed into an intensity matrix with gray values in the range of [0; 255]. The transformation of the image also includes an equalization, which is needed to downscale fluctuations in brightness between different images. (ii) The second feature is a probabilistic boundary map. This feature image is created additionally to the segmentations of an image (See chapter 2). The probability matrix is built from superpositions of all boundaries of a variety of segmentations, and represents for each pixel the probability to be part of a boundary in one of the segmentations. Figure 3.6 shows an example of such a probability map.

3.3 Implementation

The Matlab implementation, which was an integral part of this thesis, provides several functions that process the tasks described in this chapter.
The class *SA_Image* encapsulates an image with its properties into an object. Additional to functions like *setColor*, *getColor*, etc. that allows local manipulation or requests to the image, the function *getBoundaries* is given. This function returns a list of boundaries, which are automatically extracted along image patches with the same color value.
To store the boundaries, another class is provided: *SA_VectorArray*. This class allows to store vectors with equal m-Dimension and different n-Dimensions into the same data structure. This is needed as boundary paths can have different lengths, but are composed of position vectors of

the same dimension.

The post processing of the boundaries is split up into two functions. First, *SA_prepareBoundaries* takes a list of boundaries and performs the pre-processing operations discussed in section 3.1 in order to remove redundant and undesired positions. Second, the function *SA_extendBoundaries* is provided, which extends all boundaries in a list along themselves by a given factor between [0;1].

In order to extract features along the extracted and post processed boundaries, polar scales are needed. The two functions *SA_getPolarScale* and *SA_getLogPolarScale* return a list of positions that divide the space according to the passed parameter. Polar scales are also stored in an instance of *SA_VectorArray*, and are used as parameter of the function *SA_extract* to extract features from a passed feature image. The result of this function is a list of feature strings, which are also stored in an instance of *SA_VectorArray*.

4

String Alignment

After the features have been extracted as described in chapter 3, the information content of an image I is now represented in form of a list of feature strings $L_{feature} = \{S_1, \ldots, S_k\}$ for each feature matrix F. The next step in the object recognition approach is to compute a similarity value between two strings in order to create a kernel matrix for a standard black-box classifier. Alignment in general is a way of arranging two information sequences to identify regions of similarity that may be a consequence of functional or structural relationships. The process of boundary extraction and the ensuing feature extraction along the post-processed boundaries B'_j ensures that the similarity of two strings is indeed a consequence of structural analogy. Therefore, alignments represent a valuable method for the purpose of calculating a resemblance score of two feature strings.

4.1 Alignment Algorithms

In order to align two feature strings S_1 and S_2, several alignment methods can be used. One thing they all have in common is, that a score function $c(U, V)$ is needed, which gives a similarity value for two elements U of S_1 and V of S_2. For alignments used in e.g. DNA sequence alignment, a similarity matrix is given for a discrete and finite set of elements. In our case, however, the string elements are vectors from R^d. The sample correlation between two vectors, where each vector index is interpreted as a sample, provides a viable way to compute such a similarity score.

$$c(U,V) = \frac{l\sum_{i=1}^{l} U_i V_i - \sum_{i=1}^{l} U_i \sum_{i=1}^{l} V_i}{\sqrt{l\sum_{i=1}^{l} U_i^2 - (\sum_{i=1}^{l} U_i)^2}\sqrt{l\sum_{i=1}^{l} V_i^2 - (\sum_{i=1}^{l} V_i)^2}}, \tag{4.1}$$

$$\text{where } l = dim(U) = dim(V).$$

In the following sub sections, four different kinds of alignment methods are discussed. As the feature stings, drawn along post processed patch boundaries, also incorporate topological properties of the boundaries (like gaps), several approaches may be viable to ensure a good alignment score between two strings that represent the same object.

4.1.1 Global Alignment

Global alignment is an approach to align two sequences of different length from the beginning to the end, allowing gaps. We assume a linear gap penalty d to be applied whenever a gap is inserted. The dynamic programming algorithm, that solves this problem, is known in biological sequence analysis as the Needleman-Wunsch [1970] algorithm. The approach we use is an adapted version of Gotoh [1982].
The motivation to use global alignment for the purpose of aligning feature strings is the assumption, that two images of the same object also may have similar boundary paths that represent the object. This may be true for images that show the object from a similar viewing angle, like frontal pictures of a stop sign. In these cases, global alignment can be applied very well as the boundaries should be almost equal and the introduction of gaps can handle missing parts in one of the strings.

The idea is to build a global optimal alignment by using previously computed alignments of optimal solutions of a subset of the problem. A scoring matrix T is built where $T(i, j)$ is the score for the optimal alignment of the first i elements of U aligned with the first j elements of V. The following relations are used to fill the matrix from the top left to the bottom right:

$$T(i,j) = \max \begin{cases} T(i-1, j-1) + c(U_i, V_j) \\ T(i-1, j) - d \\ T(i, j-1) - d. \end{cases} \tag{4.2}$$

The final score of the global alignment is $s(S_1, S_2) = T(m, n)$, which is the bottom right value of the scoring matrix T, with m being the size of S_1, and n the size of S_2.

If not only the alignment score $s(S_1, S_2)$ is needed, but also the alignment itself, a second matrix A is needed. For each entry $T(i, j)$ computed in the alignment matrix the corresponding entry $A(i, j)$ is filled with a token, indicating which decision was made in equation 4.2 to compute the value of $T(i, j)$. To reconstruct the optimal alignment, the matrix A can be traced back from the bottom right entry $A(m, n)$, reproducing the alignment by means of the local decisions.

4.1.2 Local Alignment

In contrast to global alignment, which tries to align two sequences from the beginning to the end, local alignment is looking at the best alignment between subsequences of the two given strings. In the context of object recognition this may lead to better results because objects may

be captured only partially due to the inherent problems of low-level image segmentation. Local alignment provides an excellent method to avoid too low scores caused by partial segmentation errors, which lead to feature strings that mix up image and background information.

The algorithm for finding an optimal local alignment is closely related to the global algorithm discussed in the previous section. There are two differences, however. First, an additional possibility for each cell in the scoring matrix is added, allowing $T(i, j)$ to take the value 0. This allows the algorithm to start a new alignment, as soon as the previous alignment score falls below zero:

$$T(i,j) = \max \begin{cases} T(i-1, j-1) + c(U_i, V_j) \\ T(i-1, j) - d \\ T(i, j-1) - d \\ 0. \end{cases} \tag{4.3}$$

The second change is the possibility of the alignment to end at any place. This implies a difference in finding the alignment score of the optimal local alignment:

$$s(S_1, S_2) = \max_{i,j} \{T(i,j)\}. \tag{4.4}$$

The final alignment score $s(S_1, S_2)$ is the overall highest entry in the alignment matrix T.
To reconstruct the alignment itself, a second matrix A is needed. In the same manner, as for global alignment, the decision that was made in 4.3 for each entry $T(i, j)$ is stored in $A(i, j)$. The reconstruction of the alignment begins at $A(i_s, j_s)$, where i_s, j_s are the indices of the maximal entry in T and continues by way of backtracking.

4.1.3 Repeated Matches

The alignment in the previous section gave the best single local alignment score. For long sequences, however, it is quite possible that there exist many different local alignments with significant score. This property of the alignment can be viable for several image types. A displayed object may have multiple alike parts. This is true, for example, in images of an animal. The legs of the animal have similar silhouettes as well as similar feature properties. This can lead to cases, in which one string captures the features along one leg, and therefore can be repeatedly matched to a string representing the the whole animal with all its legs.

Let us assume that we are only interested in alignment scores higher than a threshold value t. This will be true in general, because there are always small local alignments that arise even in completely unrelated sequences. To build up the alignment matrix T, first $T(0, 0) = 0$ has to be initialized. The matrix T can now be filled using the following relations:

$$T(i,0) = \max \begin{cases} T(i-1, 0) \\ T(i-1, j) - t \quad \text{for } j = 1 \ldots m, \end{cases} \tag{4.5}$$

with $m = length(S_1)$, and

$$T(i,j) = \max \begin{cases} T(i,0) \\ T(i-1,j-1) + c(U_i, V_j) \\ T(i-1,j) - d \\ T(i,j-1) - d. \end{cases} \tag{4.6}$$

The relation (4.5) handles unmatched regions and ends of matches only allowing matches to end when they exceed the threshold value T, whereas (4.6) handles starts of matches and extensions. Note that this method is asymmetric: it finds multiple local alignments of one sequence in the other.

The individual scores $s_i(S_1, S_2)$ can be found by tracing back the first row of the alignment matrix. The overall score can be computed by summing up the individual scores s_i:

$$s(S_1, S_2) = \sum_i s_i(S_1, S_2). \tag{4.7}$$

To find the alignment itself similar to the reconstruction for local alignments. The matrix A again stores the decisions made at the corresponding entries of T. The difference is, that in repeated matches several alignments may occur. The last alignment is found by starting at the entry $A(i_{s_k}, j_{s_k})$, where the indices correspond to $T(i_{s_k}, j_{s_k})$; the score entry of the last alignment. The backtracking of this alignment ends, when i reaches zero. The bottom entry of the previous column then is the position, where the previous alignment begins. Repeating this leads to the individual alignments of S_1 to S_2.

4.1.4 Overlap Matches

Another type of search is appropriate when it is expected that one sequence is contained in the other, or if they overlap. This method is, in fact, similar to the global alignment, but, in contrast, it does not penalize overhanging ends. Figure 4.1 shows this schematically.
The motivation to use overlap-matches may be described as follows: boundaries, coming from segmentations of the same object, may emerge with different parts missing. Overlap matches alignment provides a method that optimally aligns the features strings, regardless of missing start-, interior-, or end parts of the strings.

In order to perform an overlap-matches alignment, the alignment matrix needs to be initialized in a first step using the following equations:

$$F(i,0) = 0 \text{ for } i = 1 \ldots m \text{ with } m = length(S_1), \tag{4.8}$$
$$F(0,j) = 0 \text{ for } j = 1 \ldots n \text{ with } n = length(S_2). \tag{4.9}$$

The recurrence relations within the matrix is now identical to the one used in global alignments:

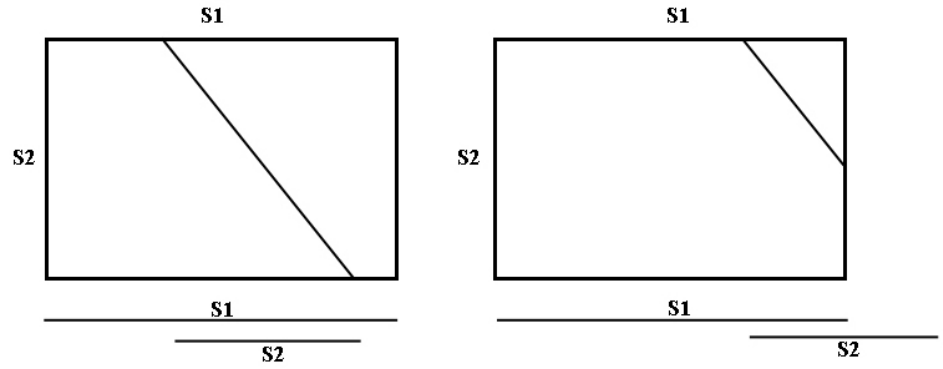

Figure 4.1: This figure schematically shows possible solutions for an overlap alignment. On the left side, a usual local match is shown. The image on the right side shows an overlap match of $S1$ and $S2$.

$$T(i,j) = \max \begin{cases} T(i-1, j-1) + c(U_i, V_j) \\ T(i-1, j) - d \\ T(i, j-1) - d. \end{cases} \tag{4.10}$$

The initialization of the top row and the left column of T with zeros allows the algorithm to start subsequences at any point without a penalty. The final alignment score is the highest entry of the right most column and the bottom row of T, due to the possibility of overlapping:

$$s(S_1, V_1) = \max \begin{cases} \max_{i=1..n} \{T(m,i)\} \\ \max_{j=1..m} \{T(j,n)\}. \end{cases} \tag{4.11}$$

The reconstruction of the alignment is again very similar to the reconstruction of global alignments. The decisions made in 4.10 and 4.11 are again stored in the corresponding cells of a second matrix A. The trace back starts at $A(i_s, j_s)$, where the indices i_s and j_s denote the scoring entry $T(i_s, j_s)$. The alignment is now found, like for global alignment, by tracing the decisions back to the first entry where $i = 0$.

4.2 Alignment Matrices

Now that a similarity score $s(S_1, S_2)$ for two feature vector strings S_1 and S_2 can be calculated, the kernel matrices for the categorization step can be computed. The kernel is built as a pairwise alignment of the strings. The feature extraction step provides a list L_i of strings for each image I_i. All L_i from the training image set are concatenated to one list L^{train}. Pairwise alignments of the elements L_i^{train} from this list lead to the training kernel M:

$$L^{train} = \bigcup_{i=1}^{t} L_i \text{ with } L_i = \{V_{i1}, \ldots, V_{ik}\} \text{ from } I_1 \ldots I_t, \tag{4.12}$$

$$M(i,j) = s(L_i^{train}, L_j^{train}). \tag{4.13}$$

The entry (i, j) in the alignment matrix M simply is the score of the the ith string aligned with the jth string of L^{train}. The matrix M represents the training kernel used in the classifyer.

A second matrix N is also created using a list L^{test}, which contains all feature strings from the test image set $J_1, \ldots . J_u$. The test feature strings L_i^{test} are aligned against the training feature L_i^{train} strings to form the test score matrix N;

$$L^{test} = \bigcup_{i=1}^{u} L_i' \text{ with } L_i' = \{V_{i1}', \ldots, V_{ik}'\} \text{ from } J_1 \ldots J_u, \tag{4.14}$$

$$N(i,j) = s(L_i^{train}, L_j^{test}). \tag{4.15}$$

4.3 Implementation

The Matlab framework provides several functions that accomplish the tasks discussed in this chapter. The four presented alignment methods are implemented as functions, that take two feature stings and the according parameter as input. *SA_localAlignment* implements local alignment, *SA_globalAlignment* implements global alignment, *SA_repeatedMatchesAlignment* implements repeated matches and *SA_overlapMatchesAlignment* implements overlap matches alignment.

The function *SA_alignmentMatrix* combines the four implementations of alignment methods into one function, which computes an alignment matrix. One of the four alignment methods is used to create the score matrix dependent on passed parameter.

To model the whole process, beginning with boundary and feature extraction and resulting in a score matrix of pairwise aligned feature strings, several functions are provided. *D_parseFileName* is a help function that parses content files in order to create a list of strings that encode the names of available images from an image category. This function is used by *D_processTrainingData* to process a given list of images, together with several of the lower level functions presented here and in chapter 3.3. Boundaries are extracted from a set of training images along which then feature strings are drawn. The extracted strings are stored, and the training score matrix is computed as pairwise alignment of all feature strings. The function *D_processTestData* has similar functionality as it stores the extracted feature strings from a set of test images and computes the test score matrix as alignment of all training strings with all test strings.

5

Categorization

The next step in the object recognition approach is to train a classifier to be able to categorize the different images classes. For this purpose a standard black-box classifier can be used, where only a valid Mercer kernel is needed. The *kernel PCA* algorithm [SSM98] describes a method to recover the basis vectors from a kernel matrix by way of eigenvalue decomposition. This requires the kernel matrix to be positive semi-definite, which is not guaranteed by the alignment methods discussed in chapter 4. The matrix M typically contains several small negative eigenvalues. To produce a Mercer kernel for the classifier, however, the *kernel PCA* algorithm can still be applied with the small change that all negative eigenvalues $\lambda_i < 0$ have to be set to zero. In detail, the adapted algorithm to recover the basis vectors $\{x_i\}_{i=1}^m$ for the matrix M proceeds as follows:

1. Calculate the centralized dot product matrix $M^c = QMQ$ with $Q = I_m - \frac{1}{m} e_m e_m^\top$ and $e_m = (1, 1, \dots\ 1)^\top$ being a m-vector of ones.
2. Express M^c in its eigenbasis: $M^c = V \Lambda^* V^\top$, where $V = (v_1, \dots\ v_m)$ contains the eigenvectors v_i and $\Lambda^* = \mathrm{diag}(\lambda_1, \dots\ \lambda_p\, \lambda_{p+1}, \dots, \lambda_m)$ is a diagonal matrix of eigenvalues with $\lambda_{p+1}, \dots, \lambda_m \leqslant 0$. The negative eigenvalues have to be set equal to zero to form the matrix $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_p\, 0, \dots, 0)$.
3. Calculate the $m \times p$ map matrix

$$X_p = V_p (\Lambda_p)^{1/2}, \ \text{ with } \ V_p = (v_1, \dots\ v_p) \ \text{ and } \ \Lambda_p = \mathrm{diag}(\lambda_1, \dots\ \lambda_p). \tag{5.1}$$

   The rows of X_p contain the vectors $\{x_i\}_{i=1}^m$ in p dimensional space, whose mutual dot products are given by M.

Predicting the cluster membership of new data. First notice that due to the eigenvalue equation $M^c V_p = V_p \Lambda_p$, we can rewrite (5.1) in the form:

$$X_p = M^c V_p (\Lambda_p)^{-1/2}. \tag{5.2}$$

Consider now the situation where we are given n new objects and the corresponding $n \times m$ matrix of pairwise alignment scores N_{ij} between these new objects and all m original objects. In order to predict the cluster membership of the new objects, we first have to project them into the Euclidean space spanned by the eigenvectors V_p of the centered dot product matrix M^c. Then, we assign each new object to the cluster with the closest centroid. For the projection itself, two steps are required:

1. re-express the matrix N in the centered coordinate system:

$$N^c_{ij} = N_{ij} - \frac{1}{m}\sum_{k=1}^{m} N_{ik} - \frac{1}{m}\sum_{k=1}^{m} M^c_{kj} + \frac{1}{m^2}\sum_{k,l=1}^{m} M^c_{kl} \tag{5.3}$$

2. project the objects represented by N^c into the coordinate system spanned by the eigenvectors V_p of the matrix M^c:

$$X_p^{\text{new}} = N^c \, V_p (\Lambda_p)^{-1/2}. \tag{5.4}$$

6

Experiments

As one of the main goals of this thesis is to find out which combinations of polar scales and alignment methods lead to the best retrieval rates, we conducted a series of tests. The images used were taken from the caltech 101 database.

6.1 Experimental Setup

We performed different test runs on 10 test and 10 training images from three different categories: crab, stop sign, and Windsor chair. Per test run, one of three different linear polar scales was used to extract feature-strings along the boundaries of of all test- and training images. The three polar scales can be seen in Figure 6.1 in relation to an average-sized image.

Two feature images where used to extract feature strings. First, the gray value of the image, and, second, the probabilistic boundary map produced in an additional step of the segmentation (See chapter 2). An example of these feature images is shown in Figure 6.2.

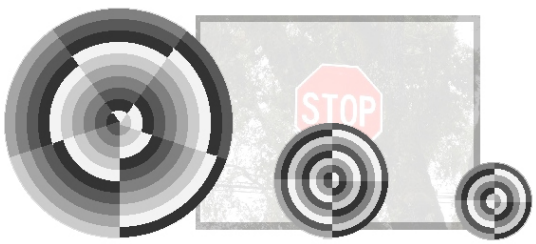

Figure 6.1: The three different sized polar scales used for information extraction.

Figure 6.2: The feature images that were used in the testing. The left picture is the original image. The picture in the middle shows the gray value image. On the right, the probabilistic boundary map is shown.

For all three sets of feature strings resulting from the feature extraction using one of the three polar scales four different alignment methods were used to compute the score matrices of pairwise alignment of all training strings against each other and all training strings against the test strings; Local alignment, global alignment, repeated matches alignment, and overlap matches alignment. Kernel matrices were produced by three different sets of parameter for each of the four alignment methods; Gap penalty $d = 0.3$ and score threshold $T = 5$, $d = 0.6$ and $T = 10$, $d = 0.9$ and $T = 15$. This resulted in 12 different training and test score matrices per image category for test run 1 and 2. The third test run was performed on local and global alignment only, resulting in 6 different training and test score matrices per image category.

The computed score matrices were transformed according to the method described in chapter 5 to create valid Mercer kernels. A standard SVM based classifier was then trained with each of the training kernels and retrieval rates were computed by evaluating the test kernels.

6.2 Results and Conclusion

The Figures 6.3, 6.4, and 6.5 show the test results for the three sets of feature strings. In the first test run the feature strings resulting from the extraction with a large polar scale were used. The second test run was based on the data coming from a medium-sized polar scale, and the last test run used the feature strings from a small polar scale. The retrieval rates of each method and parameter set are shown on the y-axis of the charts, where class 0 is crab, class 1 is stop sign and class 2 is Windsor chair. The retrieval rates are values between [0;1] and display the percentage of correct classified test images of the corresponding class. The right most bar for each alignment shows the average retrieval rate of all three classes together.

As can be seen in 6.3 and 6.4, the average retrieval decreases for more complicated alignments; Repeated matches alignment and overlap matches alignment. This shows, that there is obviously no need to use complicated methods, since the simple alignments (global- and local alignment) lead to better results, regardless of the size of the polar scale used to extract the feature strings.

Another observation, which becomes clear if the retrieval rates of the three test runs are compared, is that the average retrieval rate decreases along with the size of the polar scale. Figure 6.3 shows the results of the test where the largest polar scale was used. In comparison to Figure 6.4 and 6.5, the retrieval rates are significantly higher for all tested alignment methods.

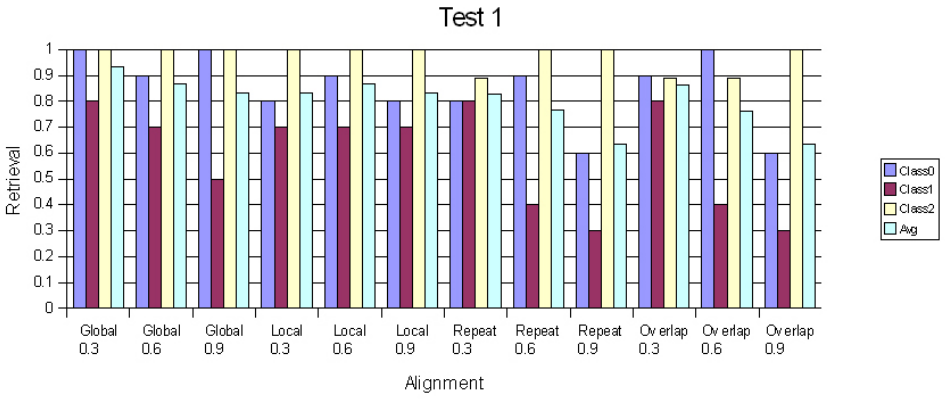

Figure 6.3: Test results using a large polar scale.

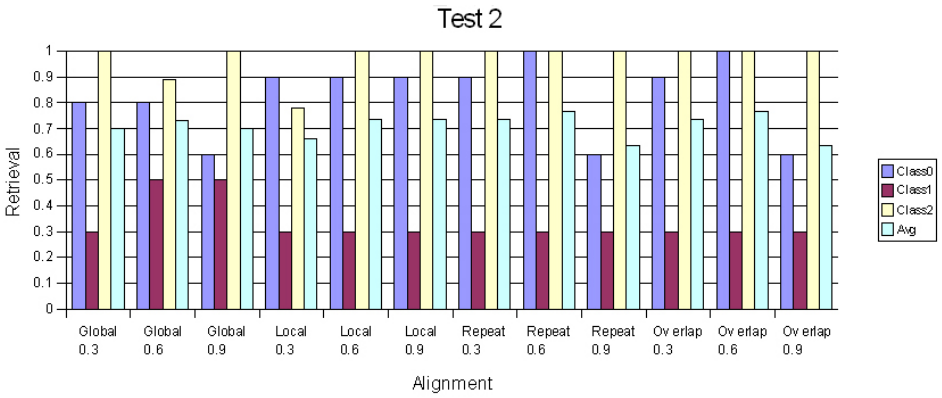

Figure 6.4: Test results using a medium-sized polar scale.

This leads to the conclusion that bigger polar scales, which capture more information of objects than smaller ones, work better.

The different parameter sets used for each alignment were also target of testing. However, the experiments don't allow a clear answer to the question which gap penalty and threshold values lead to the best results.
In the first test run (Figure 6.3), global alignment performs best for small gap penalties. This observation is understandable as this method may need to introduce a lot of gaps to align two strings. Local alignment, in contrast, seems to be quite independent of the gap penalty size, which is a direct consequence of the local method.
Repeated and overlap matches alignment show for class 0 (crab) a tendency to lower retrieval rates for larger gap penalties.
These different observations, however, could not clearly be verified in the other two test runs (Figure 6.4 and Figure 6.5). Also the impact of the threshold parameter t, which was used in overlap and repeated matches alignment, shows no clear trend. Therefore, no concluding statement about the impact of alignment parameter can be made at this point. More experiments need to be performed to answer this question.

The retrieval rates of class 1 (stop sign) are in all three test runs significantly lower than the

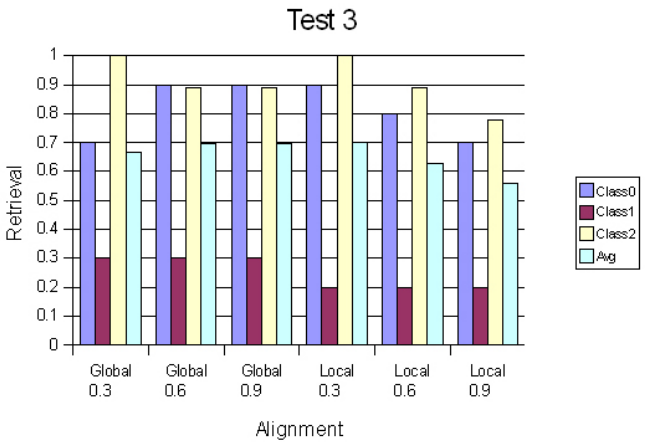

Figure 6.5: Test results using a small polar scale.

rates of the other classes. This may be explained by the fact, that the images used in the stop sign class have partly large differences in the scale of the displayed object. In some cases, the stop sign covers the whole image. In others, the image was taken from fairly far away, resulting in a lot of background region compared to the stop sign itself. This shows, that the method depends a lot on the scale of the displayed objects. Large differences can lead to an unstable classification.

The test results in general show, however, that the method is indeed capable of performing a state-of-the-art object recognition. Retrieval rates that are mostly above 0.75 encourage further examination of the approach. Training and test images have admittedly been chosen to be well separable, but the method seems to deliver good results nonetheless.

7

Conclusion

In this thesis, a method was presented which allows to exploit low-level image segmentation for object recognition. A series of segmentations per image are used to extract features along the segment boundaries. Several post processing steps were shown that optimize the segment boundaries for the ensuing alignment. Polar scales were introduced as a method to spatially divide the image along the boundaries in order to extract feature vectors. Different alignment methods have been discussed to create a kernel score matrix using the extracted feature strings. With an adapted version of the kernel PCA algorithm [SSM98], the score matrices can be transformed into valid Mercer kernels. These kernels can be used to train a standard classifier.

Experiments show that this method is indeed capable of performing a state-of-the-art object recognition. It seems to work best when large polar scales are used to extract feature strings. The size of the polar scale, which lead to the highest retrieval rates, was roughly the same as the dimension of the image. This shows that it is necessary to consider long-range interactions in images.

Tests also show that average retrieval rates decrease for more complicated alignments like repeated matches alignment and overlap matches alignment. This observation indicates, that there is no need to use complicated methods as the simple alignments (global- and local alignment) lead to better results, regardless of the size of the polar scale used to extract the feature strings.

The different parameter sets used for each alignment were also target of testing. However, the tests do not allow a clear answer to the question which gap penalty and threshold values lead to the highest retrieval rates.
Global alignments seem to perform best for small gap penalties. Local alignments, in contrast, seem to be quite independent of the gap penalty. For repeated and overlap matches alignment, a tendency to lower retrieval rates for larger gap penalties can also be observed for particular

image classes.
These tendencies, however, are limited to some configurations, and could not be verified for all test cases. Therefore, no concluding statement can be made at this point regarding the parametrization of the alignment.

Tests have also uncovered another problem: Large differences in the scale of the displayed object lead to unstable classifications. This effect needs further examination to ensure that the method can be used on much larger sets of possibly low-quality images.

Bibliography

[Aga04] Awan A. Roth D. Agarwal, S. Learning to detect objects in images via a sparse, part-based representation. *IEEE Trans. Pattern Anal. Machine Intell.*, 26(11), 2004.

[Ger98] Potter D.F. Chi Z. German, S. Composition systems. *Technical report, Division of Applied Mathematics, Brown University, Providence, RI*, 1998.

[Rot04] Lange T. Roth, V. Adaptive feature selection in image segmentation. *Pattern RecognitionDAGM04*, 2004.

[SSM98] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[Yu02] Gross R. Shi j. Yu, S.X. Concurrent object recognition and segmentation by graph partitioning. *NIPS, MIT Press*, 2002.