Probabilistic Local Variation Segmentation

Michael Baltaxe
Probabilistic Local Variation Segmentation

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Michael Baltaxe

Submitted to the Senate
of the Technion — Israel Institute of Technology
Cheshvan 5774  Haifa  October 2013
This research was carried out under the supervision of Prof. Michael Lindenbaum, in the Faculty of Computer Science.

**Acknowledgements**

I would like to express my gratitude to my thesis advisor, Prof. Michael Lindenbaum. His guidance and supervision were crucial for the successful completion of this work. I was lucky enough to have the chance to learn from him about computer vision and, maybe more importantly, about scientific research and critical thinking.

I also want to thank my family for their encouragement through this journey. My father and mother, Ronald and Sara, together with my brother Erik were an endless source of support.

The Technion’s funding of this research is hereby acknowledged.
Contents

List of Figures

Abstract

Abbreviations and Notations

1 Introduction

2 Background
  2.1 Image Oversegmentation
     2.1.1 Local Variation
     2.1.2 Watershed
     2.1.3 Mean Shift
     2.1.4 Turbopixels
     2.1.5 SLIC Superpixels
     2.1.6 Normalized Cut (NCut)
  2.2 Machine Learning Algorithms
     2.2.1 Naive Bayes Classifier
     2.2.2 Random Forest Classifier
  2.3 LUV Color space

3 The Local Variation Algorithm

4 Empirical Study of the Local Variation Algorithm
  4.1 Greedy Merging - Distribution and Size Independent Version
  4.2 Local Variation with a Constant Threshold - Distribution Dependent,
     Size Independent Version I
  4.3 Greedy Merging with Distribution-Adaptive Weights - Distribution De-
     pendent, Size Independent Version II
  4.4 Area Based Merging - Distribution Independent, Size Dependent Version
  4.5 LV without Removing Small Segments
  4.6 Experimental Results
  4.7 Conclusion
5 Local Variation As a Statistical Hypothesis Test
  5.1 Interpreting Local Variation as Maximum Estimation ................. 31
    5.1.1 Estimating the Maximum of a Uniform Distribution .............. 31
    5.1.2 Interpreting Local Variation as Maximum Estimation ............ 32
    5.1.3 A Controllable Segmentation Based on Maximum Estimation ...... 33
  5.2 Local Variation as a Hypothesis Testing Problem .................... 33
    5.2.1 Natural Image Statistics ..................................... 33
    5.2.2 Merge Decisions from Hypothesis Testing ....................... 37
    5.2.3 Parameter Estimation under Bias Sampling ....................... 39
    5.2.4 Properties of the Segmentation ................................ 43
    5.2.5 Computational Complexity of Hypothesis Testing ................. 43
  5.3 Experimental Results .............................................. 43
    5.3.1 Quality Measures and Data ..................................... 43
    5.3.2 Testing Probabilistic Versions of the Local Variation Algorithm 44
  5.4 Conclusion .......................................................... 47

6 Multistage Learning Based Oversegmentation ............................ 49
  6.1 Single-Linkage Agglomerative Oversegmentation ....................... 49
    6.1.1 Our Contribution .................................................. 51
  6.2 Oversegmentation Algorithm ......................................... 52
    6.2.1 Initialization - Weighted Graph Definition ...................... 53
    6.2.2 Iterative Merges .................................................. 54
    6.2.3 Postprocessing I: Removing Small Segments ...................... 54
    6.2.4 Postprocessing II: Backtracking ................................ 55
    6.2.5 Multistage Segmentation ......................................... 57
  6.3 Classifiers ........................................................... 59
    6.3.1 Features .......................................................... 59
    6.3.2 Training Process .................................................. 61
  6.4 Experimental Results ................................................. 62
    6.4.1 Implementation Details .......................................... 62
    6.4.2 Results ......................................................... 62
  6.5 Conclusion ........................................................... 63

7 Discussion ............................................................. 67

A Fast Approximation of $\chi^2_{p,\nu}$ ................................ 69

Hebrew Abstract .......................................................... i
List of Figures

1.1 Example of images divided into superpixels by an oversegmentation method. The number of superpixels depends on the characteristics of the image and the needs of the end user. .......................... 6

3.1 The local variation method against other oversegmentation methods described in the literature. (a) Recall and (b) undersegmentation error curves as a function of the number of segments. .......................... 22

4.1 Experiments performed on the local variation algorithm. .......................... 27

4.2 The local variation method against its reduced versions. No reduced version could achieve the performance of the original algorithm. .......................... 28

5.1 Segmentations obtained with the methods described in this chapter. Compare to figure 4.1. .......................... 34

5.2 (a-d) Four natural images, (e-h) semi-log plot with histogram of weights of the following edge sets for each image: all edges in an image, all edges within human marked segments (i.e., excluding edges crossing borders), edges in an MST of an image, edges in an MST of each segment. .......................... 36

5.3 Right: visualization of $\hat{\lambda}$ for each segment. The brighter the segment, the larger $\hat{\lambda}$. Thus white represents smooth segments and black represents textured ones. Left: corresponding oversegmentation. .......................... 42

5.4 (a) Recall and (b) undersegmentation error of the local variation method against its probabilistic versions proposed in this chapter. While the algorithms behave similarly, the proposed hypothesis test (with censored estimation) performs best for medium to large segment counts. .......................... 46

5.5 Four images segmented with the pLV algorithm (censored version) with parameters $m = 350, \delta = 0.05$. The number of segments depends on the characteristics of the image, yielding 344 segments for image a, 77 for b, 244 for c, and 261 for d. .......................... 47

6.1 Recall for several oversegmentation methods in the literature. .......................... 51
6.2 Post-processing to remove small segments: (a) without post-processing, (b) with post-processing. Note how the recall is maintained while the number of segments is effectively reduced.

6.3 Example of backtracking: $S^0$ is the initial segmentation with each segment being a vertex of the graph. $S^1$ is the segmentation obtained by greedy merging when stopping with two segments, the edge with largest weight \((20)\) has been removed defining to connected components: $SP^1_1$ and $SP^1_2$. The thick lines in $S^1$ are edges in an MST of each of the components. Finally, the optimal cutting edge is found to be $e_{cut} = (SP^0_7, SP^0_10)$. This is the edge from those in the MSTs that yields the largest value for equation (6.4) (yielding exactly the value $15 + 15 + 1 = 31$ which is the sum of the edges between $T_{12e_{cut}}$ and $T_{22e_{cut}}$). Once this edge is removed three trees remain, defining the final three components of segmentation $S^2$.

6.4 Correcting a segmentation using backtracking: original segmentation (left) and segmentation after backtracking (right).

6.5 A scheme describing the evolution of SP number through the algorithm.

6.6 Distances between superpixels for several features used in the classifiers. $D_{LUV}$ and $D_{AbsoluteGrad}$ complement each other, note for example the rightmost edge of the statue to the right (visible only in $D_{AbsoluteGrad}$) and the bottom edges in all three statues (visible only in $D_{LUV}$). $D_{MaxSize}$ differentiates SP pairs containing big segments.

6.7 Recall results for the methods described in the chapter.

6.8 First two rows: example of images segmented at different levels of detail using both the lean and rich classifiers, four stages and backtracking. From left to right, 2000 SPs, 600 SPs, 100 SPs. Third and fourth rows: comparison of the segmentations obtained with our method to hand-made segmentations before (third row) and after BT (fourth row). Blue edges represent edges found only by the algorithm, red edges represent edges drawn only by humans, green edges represent edges defined by both human and our algorithm. Noted how the edges drawn by humans are consistently recalled by the algorithm.

6.9 Precision vs. recall curve. Note how our method improves the basic pLV used to create initial superpixels. Additionally, note that methods that calculate a soft edge decision are more successful than algorithms that create closed contours.

A.1 The value $\chi^2_{p,\nu}$ as a function of its degrees of freedom.
Abstract

The goal of image oversegmentation is to divide an image into several pieces or “segments”, such that each segment is part of an object present in the scene. Contrary to image segmentation algorithms, an oversegmentation algorithm is allowed to output more segments than the number of objects that appear in the image. Oversegmentation is a very common preprocessing step for several common computer vision tasks. In this work we study image oversegmentation and develop new algorithms.

In the first part of our work, we analyze the local variation (LV) algorithm, which is one of the most common algorithms for image oversegmentation. We show that all the components in LV are essential to achieve high performance and then show that algorithms similar to LV can be devised by applying different statistical decisions. This leads us to introduce probabilistic local variation (pLV), a new algorithm based on statistics of natural images and on a hypothesis testing decision. pLV presents state-of-the-art results (for fine oversegmentation) while keeping the same computational complexity of the LV algorithm, and is in practice one of the fastest oversegmentation methods in the literature.

The LV and pLV algorithms are, in essence, single linkage algorithms. In the second part of our work we restrict ourselves to this type of algorithm and propose three modifications that improve their accuracy. First, we use machine learning methods to learn dissimilarities between superpixels and use these dissimilarities as distances between clusters. Then, we introduce a multistage approach to compute robust features. Finally, we add a correction mechanism which includes global information to overcome mistakes introduced by the greedy decisions. The resulting algorithms are more accurate than the pLV / LV algorithms but also slower. Therefore, the choice of a particular algorithm depends on the desired speed / accuracy tradeoff.
## Abbreviations and Notations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT</td>
<td>backtracking algorithm</td>
</tr>
<tr>
<td>CI</td>
<td>confidence interval</td>
</tr>
<tr>
<td>HypThr(C)</td>
<td>pLV’s adaptive threshold</td>
</tr>
<tr>
<td>Int(C)</td>
<td>internal dissimilarity of component $C$</td>
</tr>
<tr>
<td>LUV</td>
<td>CIE 1976 ($L^*u^<em>v^</em>$) color space</td>
</tr>
<tr>
<td>LV</td>
<td>local variation</td>
</tr>
<tr>
<td>Mint(C, D)</td>
<td>minimum internal difference of components $C$ and $D$</td>
</tr>
<tr>
<td>ML</td>
<td>maximum likelihood</td>
</tr>
<tr>
<td>MST</td>
<td>minimum spanning tree</td>
</tr>
<tr>
<td>$P_G(S)$</td>
<td>precision of segmentation $S$ w.r.t. ground truth $G$</td>
</tr>
<tr>
<td>PDF</td>
<td>probability density function</td>
</tr>
<tr>
<td>pLV</td>
<td>probabilistic local variation</td>
</tr>
<tr>
<td>$R_G(S)$</td>
<td>recall of segmentation $S$ w.r.t. ground truth $G$</td>
</tr>
<tr>
<td>SP</td>
<td>superpixel</td>
</tr>
<tr>
<td>$T(x)$</td>
<td>LV’s threshold function</td>
</tr>
<tr>
<td>$UE_G(S)$</td>
<td>undersegmentation error of segmentation $S$ w.r.t. ground truth $G$</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>$\chi^2$ distribution with $\nu$ degrees of freedom</td>
</tr>
<tr>
<td>$</td>
<td>C</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Image segmentation is the procedure of partitioning an input image into several meaningful pieces or segments, each of which should be semantically complete (i.e., an item or structure by itself). A segmented image can be considered as an intermediate representation of the world, which makes it useful for later higher level processes. This makes image segmentation a major focus of – and a challenge to – the computer vision community.

Oversegmentation is a less demanding type of segmentation that relies on the notion of superpixels, introduced by Ren and Malik in [RM03]. The aim is to group several pixels in an image into a single unit called a superpixel so that it is fully contained within an object; it represents a fragment of a conceptually meaningful structure. An image divided into superpixels in which there are more superpixels than objects is called an oversegmentation, so called because each superpixel by itself can be though of as a segment, and there are many more segments than objects in the image.

Oversegmentation is an attractive way to compact an image into a more succinct representation. Thus, it could be used as a preprocessing step to improve the performance of algorithms that deal with higher level computer vision tasks. For example, superpixels have been used for discovering the support of objects in images [RW11], describing the 3D geometry of a scene [HEH07], multiclass object segmentation [FVS09, YHRF10, GRC+08], and image segmentation [LWC12]. Figure 1.1 shows examples of superpixels created on images, obtained with an oversegmentation method.

In this work we treat the task of image over segmentation. Several different approaches towards this task exist, some of the most popular being [CM02, LSK+09, ASS+12], among others. Of special importance for our work is the local variation (LV) algorithm [FH04]. This is a fast an accurate method for image over segmentation on which our methods build.
The local variation algorithm is a graph based greedy algorithm. It merges iteratively two neighboring segments according to a segment similarity criterion (the gradient between segments) and an adaptive threshold combining properties of both segments. The greedy approach yields low computational complexity, and the algorithm presents state-of-the-art results in terms of standard oversegmentation quality measures, however mathematical theory behind it is little.

Initially, we will develop some insight into the LV algorithm by empirical experimentation. Reduced versions of the algorithms are subjected to test in order to reveal the importance of each one of the components in the algorithm. We will eventually conclude that both components present in the adaptive threshold (segment size and gradient distribution within the segments) are essential in order to obtain good image segmentations.

Building on this observation, in this work we consider the local variation method as a process making local decisions based on statistical considerations. In a first stage, we show that LV can be interpreted as a process estimating the value of the strongest gradient within a segment, under the assumption that the gradients within image segments are distributed uniformly. Furthermore, we expand this probabilistic approach and introduce the probabilistic local variation algorithm (pLV) which casts LV as a hypothesis testing problem. Towards this end, we treat the gradients within a segment as distributed according to an exponential PDF, and for each pair of neighboring segments we calculate the probability that a gradient with each segment is larger than the gradient between the two of them. The assumption that the gradients are distributed exponentially is treated in this work (see section 5.2.1), and we keep the same graph based implementation used by LV in order to keep low computational complexity. Actually, the computational complexities of LV and pLV are the same.

In a second stage, we focus on single-linkage agglomerative algorithms for image oversegmentation. This family of algorithms have been proposed for image segmentation by performing iterative greedy merges within pixels, nevertheless, limited results are
obtained due to its greedy decision process. Image oversegmentation, however, in which a smaller number of greedy decisions are made can benefit from the single-linkage approach. We note that both LV and pLV are in essence single-linkage algorithms, and in this work we present three modifications to the general framework which improve the performance of the algorithms.

The first improvement comes from the world of machine learning. Instead of using gradients within segments as similarity measure, we use supervised learning to learn a same / not same rule between neighboring segments. Instead of checking the classifier’s output class, we use the classifier’s decision level as the similarity measure between segments. It is this similarity measure which we eventually use to sort segments within the single-linkage framework. The similarity measure might change over time, as more and more merges are performed. This leads us to the second improvement of the framework: recalculating the feature vector and the decision level multiple times throughout the process, thus yielding a multistage learning based oversegmentation method.

In a way, single-linkage methods are shortsighted: only local information is considered for making a merge decision. The third improvement we propose to single-linkage intends to overcome this issue. We introduce a correction mechanism which uses global information to fix mistakes introduced in the greedy process. The method, denoted as backtracking, finds the partition that maximizes a cost function subject to a reduced search domain which includes only partitions of the MST induced during the greedy merging process.

The structure of this document is as follows. Previous work on image oversegmentation, together with some machine learning algorithms are presented in chapter 2. The local variation method is introduced in chapter 3. An empirical study of the local variation algorithm is presented in chapter 4. Then, our probabilistic interpretation of local variation and the pLV oversegmentation method are introduced in chapter 5. Chapter 6 presents our modified single-linkage approach to image oversegmentation. Finally, discussion follows in chapter 7.
Chapter 2

Background

2.1 Image Oversegmentation

Following is a description of some common algorithms used for image oversegmentation. These algorithms were tested and used throughout our work.

2.1.1 Local Variation

Local variation, by Felzenszwalb and Huttenlocher [FH04], is a common algorithm used to create superpixels. It is an accurate and fast method with complexity $O(n \log n)$. This method is the base of our work in chapters 4 and 5 and is presented in detail in chapter 3.

2.1.2 Watershed

The watershed method is a morphological approach to image segmentation. Initially, a gradient magnitude image of the input is created and its local minima are found and labeled with unique identifiers. This gradient magnitude image is interpreted as a topographical map: the higher the gray level, the higher the terrain at that point. For each pixel in the image, a “drop of water” is poured and tracked all the way in this “gradient terrain” until it is deposited in one of the previously identified local minima. Each pixel is assigned the label associated with the local minimum in which its water drop was deposited. Finally, each set of pixels with the same label defines a segment. From a topographical point of view, a segment can be interpreted as the catchment basin of the local minimum associated with the corresponding label.

Meyer [Mey94] introduced an efficient $O(n \log n)$ algorithm to simulate this process. The algorithm is as follows:
1. Create the gradient magnitude of the input image, $G(x,y)$.

2. Give a different label to each local minima of $G(x,y)$.

3. Insert the neighbors of all labeled pixels into priority queue $Q$, pixel $(x,y)$ is inserted with priority $1/G(x,y)$.

4. While $Q$ is not empty:
   (a) Extract the pixel with highest priority from $Q$, call it $x$.
   (b) If all labeled neighbors of $x$ have the same label, assign this label to $x$, and add to $Q$ all its non-labeled neighbors that are not already in the queue.

5. The unlabeled pixels define the watershed lines, and the segmentation.

This algorithm is fast, although it is very susceptible to noise. If there is noise in the image, the gradient image, $G(x,y)$, will be even noisier, thus defining a large number of local minima in step 2 of the algorithm. The final outcome is a large fragmentation: many small artificial segments appear in the segmentation. Likewise, noise might reduce the intensity of an edge, yielding a merge of two perceptually separate regions. The quality of the gradient created in step 1 of the algorithm has strong influence in the segmentation output.

2.1.3 Mean Shift

The mean shift is an algorithm widely used for the task of data clustering. The basic ideas were originally developed in the mid 1970’s by Fukunaga and Hostetler [FH75], however it became popular in the computer vision community only after the works by Cheng [Che95] and specially by Comaniciu and Meer [CM02].

A set of input points are regarded as random samples drawn from a probability density function (PDF) and the procedure finds the peaks of the PDF, which correspond to the modes of the data. Then, the data is divided into clusters by associating each data point to its mode. The complexity of this algorithm is $O(n^2)$, but it is quite accurate (has relatively good recall).

The procedure finds the corresponding mode to each data point $x_k$ by performing the following steps:

1. Set $y = x_k$.

2. Open a search window around point $y$.

3. Shift $y$ to the mean position of the points within the search window.

4. Repeat steps 2 and 3 until convergence.
The search window defined in step 2 is called a “kernel”. It is a continuous function defined over the full domain of the input data, and, eventually, gives a weight to each point as a function of its distance from \( y \).

Formally, let \( g(x) \) be the kernel function, \( h \) be a user controlled parameter, and \( \{x_i\}_{i=1,...,n} \) be the set of input points. For each point \( x_k \) (\( 1 \leq k \leq n \)), the sequence of successive locations of the kernel, \( \{y_j\}_{j=0,1,2,...} \), is defined as follows:

\[
y_0 = x_k
\]

\[
y_{j+1} = \frac{\sum_{i=1}^{n} x_i g \left( \frac{y_j - x_i}{h} \right)^2}{\sum_{i=1}^{n} g \left( \frac{y_j - x_i}{h} \right)^2}
\]

Comaniciu and Meer showed in [CM02] that the procedure converges if the kernel “has a convex and monotonically decreasing profile”, where the profile of \( g(x) \) is \( k(x) = -\int g(x) dx \). Furthermore, they showed that mean shift performs gradient ascent on the underlying PDF, hence the sequence \( y_j \) converges to a maximum (mode) of this function.

For image segmentation, each pixel is an input point for the procedure, and the feature space is usually a five element vector \( x = [x, y, L, U, V]^T \) where \( (x, y) \) are the \( x \) and \( y \) coordinates of the pixel and \( (L, U, V) \) are the color components of the pixel in the LUV color space (see section 2.3). The parameter \( h \) can be set to control the number of segments in the output oversegmentation. Some commonly used kernels are the following:

- \( g(x) = \begin{cases} \ 1, & |x| \leq 1 \\ \ 0, & \text{otherwise} \end{cases} \)

- \( g(x) = \frac{1}{2} e^{-\frac{x^2}{2}} \)

2.1.4 Turbopixels

Levinstein et al. [LSK+09] presents a level sets approach towards the task of image oversegmentation. The main idea is to evolve a set of curves such that they never cross each other and also attach to edges in the image. By the end of the procedure, the evolved curves define the borders of superpixels. The numerical framework used for curve evolution is level sets, and narrow band methods are adopted to accelerate the procedure. The main drawback in this method is its low recall. The computational complexity of the method is \( O(n) \), although in practice it is slower than watershed or local variation [ASS+12].
The algorithm is as follows:

1. Create a grid of seed points throughout the image.

2. Create the level set function $\Psi^0$ as the signed Euclidean distance from the seed points.

3. While the boundaries defined by the zero level set of $\Psi$ can evolve:
   
   (a) Evolve $\Psi$ for $T$ steps according to the equation $\Psi^{n+1} = \Psi^n - S_I S_B \|\Psi^n\| \Delta t$.

   (b) Create the skeleton of the zero level set of $\Psi^{n+1}$, this defines the set of superpixels.

   (c) Recalculate the speed functions $S_I$ and $S_B$.

The parameter $\Delta t$ in step 3a controls the rate of evolution of the curve. The terms $S_I$ and $S_B$ control the speed of the front at each point in the domain of the image. The boundary speed, $S_B$, prevents the curves from crossing each other and is defined as follows:

$$S_B(x, y) = \begin{cases} 
0, & \text{if } (x, y) \text{ on the skeleton} \\
1, & \text{otherwise}
\end{cases}$$

On the other hand, the image speed, $S_I$, is driven by boundary curvature and image similarity: boundary curvature favors smooth edges, while image similarity makes the contour stick to image edges. The function used is the following:

$$S_I(x, y) = (1 - \alpha \kappa(x, y)) \phi(x, y) - \beta (N(x, y) \cdot \nabla \phi(x, y))$$

where $\kappa(x, y)$ is the boundary curvature at point $(x, y)$, $\phi(x, y)$ is the “local affinity” function, giving low values near edges and high values elsewhere, and $N(x, y)$ is the normal of the curve at point $(x, y)$. The parameters $\alpha$ and $\beta$ control the weight of each term in $S_I$. It is important to note that the left term in this function is a reaction-diffusion term that makes the curve slow down when it gets close to an image edge, and the right term is the well known doublet term that attracts the curve to edges when it is close to one. In the original paper, the local affinity function is defined as

$$\phi(x, y) = e^{-E(x, y)/\nu}$$

$$E(x, y) = \frac{\|\nabla I(x, y)\|}{G_\sigma * \|\nabla I(x, y)\| + \gamma}$$

Here, $I(x, y)$ is the input image, while $\nu, \sigma, \gamma$ are parameters.
2.1.5 SLIC Superpixels

Radhakrishna et al. [ASS+12] proposed the SLIC superpixel algorithm, which is based on the popular k-means clustering algorithm. The complexity is reduced by defining a restricted search area for each cluster center during the assignment step. To cope with this reduced search space, a distance function is specifically designed, taking into account both spatial proximity and color proximity. The algorithm has recall comparable to that of local variation. Its complexity is $O(n)$, and in practice its runtime is very similar to local variation, with a small advantage for SLIC. The algorithm, however, uses an additional parameter that introduces a trade-off between superpixel compactness and adhesion to image boundaries.

The distance function between pixels $p_i$ and $p_j$ used in [ASS+12] is the following

$$d_c(p_i, p_j) = \sqrt{(L_i - L_j)^2 + (U_i - U_j)^2 + (V_i - V_j)^2}$$

(2.1)

$$d_s(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

(2.2)

$$D(p_i, p_j) = \sqrt{\left(\frac{d_c(p_i, p_j)}{N_c}\right)^2 + \left(\frac{d_s(p_i, p_j)}{N_s}\right)^2}$$

(2.3)

where $L_a, U_a, V_a$ are the color components of pixel $a$ in LUV color space, $x_a, y_a$ are its spacial coordinates, $N_c$ is a constant defining the maximum color distance, and $N_s$ is a constant defining the maximum spacial distance.

2.1.6 Normalized Cut (NCut)

Normalized Cut (NCut) was proposed by Shi and Malik in [SM00], and it has been used by Ren and Malik [RM03] to perform segmentation by training a classifier that differentiates between correct and incorrect superpixel groupings. This is a graph cut formulation of the problem, in which a cost function is minimized in order to divide the graph into two disjoint sets. Once the two sets yielding the optimal value of the cost function have been found (the optimal cut), the procedure can be repeated iteratively until a selected number of segments is reached.

Let $G = (V, E)$ be a graph defined by the image such that each pixel is a vertex $v \in V$, and each pair of neighboring pixels define an edge $e = (u, v) \in E$. Let $w(e)$ be the weight of edge $e = (u, v)$ corresponding to the similarity between pixels $u$ and $v$. This similarity can be defined in terms of color, position, texture, etc. The sets $A, B \subseteq V$ are are said to be a cut of $G$ if $A \cup B = V$ and $A \cap B = \emptyset$, and the value of the cut is given by:

$$\text{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v)$$

(2.4)
We would like to find the optimal cut minimizing equation (2.4), however, it has been noticed that this function is usually minimized by simply disconnecting one pixel from the rest of the image. In contrast, Shi and Malik [SM00] propose to minimize the normalized cut function:

\[
N_{\text{cut}}(A, B) = \frac{\text{cut}(A, B)}{\text{assoc}(A, V)} + \frac{\text{cut}(A, B)}{\text{assoc}(B, V)}
\]

(2.5)

where \(\text{assoc}(A, V) = \sum_{u \in A, v \in V} w(u, v)\). In this case, the cut that separates a single point would yield a large value for \(\text{cut}(A, B)/\text{assoc}(A, V)\) since all the edges between A and V are edges traversing the cut, thus yielding a large value for the Ncut function.

Finding the minimizer of equation (2.5) is an NP-Complete problem. An approximation, based on spectral theory, to finding the optimal cut is devised: let \(D\) be a diagonal matrix of size \(|V| \times |V|\) with \(D(i, i) = \sum_{j \in V} w(i, j)\), and let \(W\) be a \(|V| \times |V|\) symmetric matrix such that \(W(i, j) = w(i, j)\). Solve the following generalized eigenvalue problem:

\[
(D - W)y = \lambda Dy
\]

(2.6)

Take the eigenvector corresponding to the second smallest eigenvalue, \(y^*\), and think of it as an indicator vector. A threshold, \(\epsilon\), on this vector defines a partition:

\[
P(y^*, \epsilon) = \begin{cases} 
A = \{v \mid y^*_v > \epsilon\}, \\
B = \{v \mid y^*_v \leq \epsilon\}
\end{cases}
\]

(2.7)

Iterate the value of \(\epsilon\) in uniform steps in the range \([\min(y^*), \max(y^*)]\) and find the threshold that yields the minimum value of equation (2.5), \(\epsilon^*\). The optimal partition of the graph (defining the partition of the image) is \(P(y^*, \epsilon^*)\).

To summarize, the complete segmentation procedure is as follows:

1. Create graph \(G = (V, E)\) with edge weights \(w(e)\) from the input image.
2. Create the matrices \(D\) and \(W\).
3. Solve the generalized eigenvalue problem in equation (2.6) and find the second smallest eigenvalue, \(y^*\).
4. Find the value \(\epsilon^*\) so that the partition \(P(y^*, \epsilon^*)\) yields the smallest value for equation (2.5).
5. The final partition is \(P(y^*, \epsilon^*)\). Iterate the process until the number of desired segments is reached.

While this is an interesting approach to image segmentation, the drawback of this algorithm is its runtime. The (approximate) complexity of this method is \(O(sn^{3/2})\),
with \( s \) being the number of segments in the output, and in real life problems, the matrices involved are very big, thus rendering this method prohibitive.

2.2 Machine Learning Algorithms

In what follows, we present two machine learning algorithms used in our work: naïve Bayes and random forest. The algorithms are used for two-class classification problems.

2.2.1 Naïve Bayes Classifier

Let \( A_1, ..., A_k \) be attributes used to determine the class \( C \) of a sample. Then, the naïve Bayes classifier [Elk97] selects the class with the highest probability given the measures on the attributes, namely the class \( c \) that maximizes \( P(C = c \mid A_1 = a_1 \land A_2 = a_2 \land ... \land A_k = a_k) \).

According to Bayes’ rule, this expression can be calculated as

\[
\frac{P(A_1 = a_1 \land A_2 = a_2 \land ... \land A_k = a_k \mid C = c)}{P(A_1 = a_1 \land A_2 = a_2 \land ... \land A_k = a_k)} P(C = c)
\] (2.8)

The output to our two class problem needs to be a monotonic decision value (see chapter 6). We obtain this value by calculating the log-likelihood of the two classes, that is

\[
d = \log \left( \frac{P(C = +1 \mid A_1 = a_1 \land A_2 = a_2 \land ... \land A_k = a_k)}{P(C = -1 \mid A_1 = a_1 \land A_2 = a_2 \land ... \land A_k = a_k)} \right)
\] (2.9)

Note that when substituting equation (2.8) into equation (2.9), the denominator in equation (2.8) cancels out. Thus training reduces to creating histograms of the distributions of each attribute for each class, and storing the distributions of the classes themselves.

2.2.2 Random Forest Classifier

Training a random forest [Bre01] consists in growing a set of trees. When testing a new sample, each tree outputs a decision determining a class. Each tree provides a “vote”, and the most voted class is the final decision of the classifier. Every one of the trees in the ensemble is grown as follows:

1. Sample a subset of subjects from the training data with replacement. This is the training set for the tree being built.

2. Let \( M \) be the number of attributes in the data. Define \( m \ll M \) and grow a classification tree such that at each node, a set of \( m \) attributes is selected
randomly and the best split is calculated within this smaller set of attributes.

Just like in the case for the naïve Bayes classifier, we need a monotonic decision value. In our two class problem, the output of the random forest provides votes for merge and not merge, thus we calculate the probability of merge, \( P(C = -1) \), as the fraction of merge votes in the forest. For a given test sample, let \( N_{-1} \) be the number of votes for merge and \( N_{+1} \) be the number of votes for not merge. The decision value is once again calculated as a log-likelihood ratio as follows:

\[
P(C = -1) = \frac{N_{-1}}{N_{-1} + N_{+1}}
\]

\[
d = \log \left( \frac{1 - P(C = -1))}{P(C = -1)} \right)
\]

(2.10)

### 2.3 LUV Color space

The CIE 1976 (L* u* v*) color space (here abbreviated LUV color space) is a transformation of the CIE XYZ color space with the aim of achieving perceptual uniformity. It is commonly used in computer vision and graphics applications since perceptual color distances are approximately euclidean distances in this space. In our algorithms (chapters 5 and 6), we use the LUV color space for quantifying dissimilarity between segments.

The \( L^* \), \( u^* \), and \( v^* \) quantities are defined by the following equations:

\[
L^* = \begin{cases} 
\left( \frac{2Y}{3} \right)^3 \frac{Y}{Y_n}, & Y/Y_n \leq \left( \frac{6}{29} \right)^3 \\
116 \left( \frac{Y}{Y_n} \right)^{1/3} - 16, & Y/Y_n > \left( \frac{6}{29} \right)^3 
\end{cases}
\]

(2.11)

\[
u^* = 13L^* (u' - u'_n)
\]

(2.12)

\[
v^* = 13L^* (v' - v'_n)
\]

(2.13)

where \( u'_n \) and \( v'_n \) are the chromaticity coordinates of the white color stimulus, and \( Y_n \) the \( Y \) coordinate of the white color in the XYZ space. The values for \( u' \) and \( v' \) are given by

\[
u' = \frac{4X}{X + 15Y + 3Z}
\]

(2.14)

\[
v' = \frac{9Y}{X + 15Y + 3Z}
\]

(2.15)

As stated above, the distances between colors in the LUV color space are approximately
euclidean, thus, the distance between two colors, $D$, is given by

$$D = \sqrt{(\Delta L^*)^2 + (\Delta u^*)^2 + (\Delta v^*)^2} \quad (2.16)$$

In [Poi81], Pointer develops an extensive analysis of the LUV color space and compares it to the CIE 1976 ($L^* a^* b^*$) color space. He concludes that both color spaces are close to euclidean in color distance perception and that neither color space is significantly better than the other, while some anomalies exist in the CIELAB space. This justifies our selection of the LUV color space.
Chapter 3

The Local Variation Algorithm

The local variation algorithm (LV) by Felzenszwalb and Huttenlocher [FH04] is a widely used, fast and accurate oversegmentation method. The method uses a graph representation of the image to iteratively perform greedy merge decisions by evaluating the evidence for an edge between two segments. Even though all the decisions are greedy, it has been shown in [FH04] that, in a sense, the final segmentation has the global properties of being neither too coarse nor too fine. The decision criterion is partially heuristic and yet the algorithm provides accurate results. The algorithm is efficient, with complexity $O(n \log n)$ ($n$ being the number of pixels in the image), and in practice runs in a fraction of a second (for $321 \times 481$ images).

The local variation algorithm is a graph based method. It merges regions iteratively one by one. In principle, a merge is decided upon by comparing the dissimilarity between the regions to the internal difference within each region.

Before presenting the algorithm, some definitions are introduced. Let $G = (V, E)$ be a graph created from the input image, with vertices on the pixels and edges between each pair of neighboring pixels. Define a weight function on the edges, $w : E \rightarrow \mathbb{R}$, representing pixel dissimilarity (for example, the LUV color distance between pixels). Likewise, let a component (segment) $C_i$ be a set of connected pixels. The components change throughout the segmentation process, and, initially, the set of components $\{C_i\}$ is the set of pixels.

Additionally, the internal dissimilarity of component $C_i$, denoted $Int(C_i)$, and the minimum internal difference of two components $C_i, C_j$, denoted $MInt(C_i, C_j)$, are defined as follows:

\[
Int(C_i) = \max_{e \in MS(C_i)} w(e) \tag{3.1}
\]

\[
MInt(C_i, C_j) = \min_{x \in \{i,j\}} (Int(C_x) + T(C_x)) \tag{3.2}
\]
where \( MST(C_i) \) is a minimum spanning tree of \( C_i \), and \( T(C_i) \) is a component dependent “threshold function” defined as

\[
T(C_i) = \frac{K}{|C_i|}
\]

(3.3)

in which \( K \) is a user controlled parameter and \( |C_i| \) denotes the number of vertices in component \( C_i \).

Then, the local variation algorithm for image oversegmentation is presented in algorithm 3.1.

**Algorithm 3.1 Local Variation Algorithm**

**Input:** A weighted graph \( G = (V, E) \) with weights \( w(e) \), \( e \in E \), defined by an image.

**Output:** A set of components \( C_1, ..., C_n \) defining segments

1. Sort \( E \) by non-decreasing edge weight \((e_1, e_2, ..., e_m)\)
2. Initialize segmentation \( S^0 \) with each vertex being a component
3. for all \( q = 1, ..., m \) do
4. \( e_q = (v_i, v_j) \) ← the edge with the \( q \)th lightest weight
5. \( C^q_{i-1} \) ← the component of \( S^{q-1} \) containing \( v_i \)
6. \( C^q_{j-1} \) ← the component of \( S^{q-1} \) containing \( v_j \)
7. if \( (C^q_{i-1} \neq C^q_{j-1}) \land (w(e_q) \leq MInt(C^q_{i-1}, C^q_{j-1})) \) then
8. \( S^q = S^{q-1} \cup \{C^q_{i-1} \cup C^q_{j-1}\} \setminus \{C^q_{i-1}, C^q_{j-1}\} \)
9. else
10. \( S^q = S^{q-1} \)
11. end if
12. end for
13. **Postprocessing:** Merge all small segments to the neighbor with closest color.

Two components are merged only if the lightest edge that connects them is lighter than the heaviest edge in the MST of the components plus a margin. Since the edges are sorted in step 1, the edges causing merges are exactly those that would be selected by Kruskal’s algorithm [Kru56] for creating an MST of the subgraph corresponding to each segment.

The parameter \( K \) can be used to control the number of segments in the output segmentation. The larger the value of \( K \), the fewer the number of segments: increasing \( K \) increases the value of the threshold function, which implies that the number of edges with weight holding the condition in line 7 of the algorithm is larger, and more merges are performed.

An interesting property is proven for the obtained segmentation. Let

\[
Dif(C_i, C_j) = \min_{v_i \in C_i, v_j \in C_j, (v_i, v_j) \in E} w((v_i, v_j)),
\]

(3.4)
and let \( D(C_i, C_j) \) be a predicate for the components \( C_i \) and \( C_j \)

\[
D(C_i, C_j) = \begin{cases} 
  \text{true,} & \text{if } \text{Diff}(C_i, C_j) > M\text{Int}(C_i, C_j) \\
  \text{false,} & \text{otherwise}
\end{cases}
\]  

(3.5)

It is shown in [FH04] that algorithm 3.1 yields a segmentation that is neither too fine nor too coarse according to predicate \( D \). Segmentation \( S \) is considered too fine if there exists a pair of components \( C_i, C_j \in S \) for which \( D(C_i, C_j) = \text{false} \). \( S \) is considered too coarse if there exists a proper refinement of \( S \) that is not too fine: segmentation \( T \) is a proper refinement of \( S \) if each component of \( T \) is contained in a component of \( S \), and additionally \( S \neq T \) (i.e., \( T \) can be obtained from \( S \) by dividing one or more segments into several parts).

It is common practice in oversegmentation algorithms to add a post-processing stage in which small segments are removed. This is implemented in line 13 of the algorithm, by merging a small segment to its spatial neighbor with the smallest color difference.

Recall is commonly used to quantify the ability of an oversegmentation method to produce segmentations in which the edges between segments attach to object borders. Recall is defined as the fraction of the ground truth boundary that lies close to the boundary of a superpixel found by the algorithm (see section 5.3.1 for a more precise definition). Figure 3.1a presents the recall of common oversegmentation methods, as a function of superpixel density (the measurements are performed on the Berkeley dataset BSDS300 [MFTM01]). Note how the local variation algorithm consistently yields the best recall results. Additionally, as stated before, it is among the fastest oversegmentation methods, matched only by SLIC superpixels.

The undersegmentation error (to be defined in section 5.3.1) of the LV algorithm is larger than the error yielded by some other oversegmentation algorithms; see figure 3.1b. For oversegmentation tasks, however, the undersegmentation error is considered less important. This fact, together with its high recall and short runtime, often make the LV algorithm the method of choice.

The high accuracy, obtained by the greedy LV algorithm, is impressive. It is mainly due to the adaptive threshold \( \text{Int}(C_i) + T(C_i) \), which depends on two components: the distribution within the segments and their size. The particular combination of these two components match the criterion used to decide whether a segmentation is too fine/coarse but is not theoretically supported otherwise. In this work we offer empirical analysis (section 4) and statistical interpretations (two versions in sections 5.1 and 5.2) of the LV algorithm. The interpretations lead, eventually, to an improved LV-like algorithm, denoted as probabilistic local variation (pLV). This new algorithm follows a statistical decision procedure, maintains LV’s desirable properties, and improves its recall and undersegmentation error.
Figure 3.1. The local variation method against other oversegmentation methods described in the literature. (a) Recall and (b) undersegmentation error curves as a function of the number of segments.
Chapter 4

Empirical Study of the Local Variation Algorithm

Understanding the remarkably good performance of the local variation algorithm was the motivation for our work in this chapter. We empirically study its properties using variations that include only some of its principles. We show that only the combination of all of them is able to achieve the same performance as the original algorithm.

The accuracy of the local variation algorithm is mainly due to the adaptive threshold used as a merging criterion. This threshold, $\text{Int}(C_i) + T(C_i)$, is the sum of two components:

1. Distribution dependent component. The first component, $\text{Int}(C_i)$, expresses the typical range, or distribution, of weight values in the segment $C_i$ to be merged. A merge decision is made when the tested weight is not too far from the weights connecting all pixels in the existing segment $C_i$.

2. Size dependent component. The second component, $T(C_i) = \frac{K}{|C_i|}$, biases the merging decision so that merging small components is preferred over merging large ones.

In this chapter, we ask ourselves which of these components and their particular implementation are crucial to maintaining highly accurate oversegmentation results. Our approach is to test reduced versions of the LV algorithm. Every reduced version is different in some particular detail from the original LV algorithm. If this detail is not important, we expect to get the same performance as the original.

All the tests are performed by starting with single pixels and use, unless stated otherwise, the weight function $w(e = (u, v)) = ||\text{LUV}(u) - \text{LUV}(v)||_2$, where $\text{LUV}(x)$ is a vector with the color components of pixel $x$ in LUV color space.
4.1  Greedy Merging - Distribution and Size Independent Version

In our first experiment, we ran the simplest reduced version of the original LV algorithm. This version uses no threshold and therefore depends neither on the distribution nor on the segment sizes. We made two changes to the original algorithm in listing 3.1:

1. For all pairs of components $C_i, C_j$ we set $MInt(C_i, C_j) = \infty$.
2. The loop in lines 3-12 of the algorithm is stopped when a given number of segments is reached.

By setting $MInt(C_i, C_j) = \infty$ for all pairs of components, line 7 of the algorithm is always evaluated to true and all pairs of segments are merged. The segments are merged by non-decreasing edge weight (as defined in line 1 of the algorithm); the oversegmentation can hence be viewed as greedy.

Figure 4.1a presents the results of this test and, as can be seen, incorrect segment merges may occur due to weak LUV color differences at local scales (i.e., neighboring pixels have very similar color, yielding small LUV differences). Mistakes due to locally weak differences are evident, for example, in the mountains, which are merged with the sky. Clearly, it is enough that a single edge in $E$ connecting two separate regions has small absolute weight in order that the two regions will be merged. This may eliminate even perceptually clear discontinuities. Thus the method is not at all robust and performs much worse than LV; see the recall curves in figure 4.2.

4.2  Local Variation with a Constant Threshold - Distribution Dependent, Size Independent Version I

In our second experiment we ran the local variation algorithm with the function $T(C_i)$ in equation (3.3) set to be a constant value:

$$T(C_i) = K.$$  \hspace{1cm} (4.1)

The result of this experiment is presented in figure 4.1b. We see that very large segments are created before small, visually similar segments are merged. For example, the grassy area in the lower right part of the image is fairly homogeneous but divided into several segments, while the water body is markedly heterogeneous due to specular reflections, yet it is extracted as a single segment. Note that in the original LV method, the function $T(C_j) = \frac{K}{|C_j|}$ accounts for some regularization of segment size: smaller segments are
easier to merge than larger ones. The removal of this regularization eventually leads to lower recall.

4.3 Greedy Merging with Distribution-Adaptive Weights - Distribution Dependent, Size Independent Version

The original LV algorithm calculates the distribution dependent part, \( \text{Int}(C_i) \), by maintaining an MST and using its heaviest edge. This way it can adapt the decision to the local segment variability. The local variability can also be taken into account by using a normalized weight that normalizes the raw LUV distance by the variability. Note that by working with the MST, the LV methods ignores most of the edges associated with the heaviest weights. A similar implementation of the same principle would be to take some percentile of the set of weights inside the segment or in its vicinity. Here we tested a version which uses a weight normalized by the median LUV difference in a fixed neighborhood. This may be regarded as a coarse, approximate version of the LV algorithm.

In this experiment we ran algorithm 3.1 with the two changes introduced in section 4.1. Additionally, we changed the definition of \( w(e) \) to account for normalization. Let \( e = (u, v) \in E \) be an edge between pixels \( u \) and \( v \), and \( x_p \) be a vector with the spatial coordinates of pixel \( p \). We define the k-edge neighborhood of an edge to be

\[
\text{EdgeNeigh}_k(u,v) = \{(a,b) \in E \mid \max (\|x_a - x_u\|, \|x_b - x_u\|, \|x_a - x_v\|, \|x_b - x_v\|) < k\}.
\] (4.2)

Then, the weight used to define the merging order in line 1 of algorithm 3.1 is

\[
w_N(u,v) = \frac{w(u,v)}{\text{median} \{w(e) \mid e \in \text{EdgeNeigh}_k(u,v)\}}.
\] (4.3)

Figure 4.1c presents the results of this experiment. The recall of the segmentation obtained is notably higher than the one obtained in the two previous experiments. (Note, in particular, how most of the mountains are separated from the sky.) This is because for every merge we take into account more information than we did before. Note, however, that we still get regions of very different sizes.

Computationally, this method is as expensive as LV. In [FH04] it is shown that when using quantiles instead of the minimum for calculating the difference between two segments, the segmentation problem becomes NP-hard. Our approach, however, is different. The normalized weights, \( w_N \), can be pre-calculated efficiently by running a
median filter on the input image, which can be done in $O(n)$ by the method in [PH07].

4.4 Area Based Merging - Distribution Independent, Size Dependent Version

In the following experiment, we explore the constraint on the size of the segments. This reduced version merges segments greedily while they are smaller than a given minimum size $M$.

The algorithm is just like algorithm 3.1, but the condition in line 7 is replaced by the following one:

\[(C_{q}^{n-1} \neq C_{j}^{n-1}) \land \left( \min_{x \in \{i,j\}} |C_{x}^{n-1}| < M \right) \, . \tag{4.4}\]

Figure 4.1d shows a segmentation obtained with this algorithm for $M = 16$. We can appreciate that in this case the superpixels are more or less regular with roughly the same area. It is important to note that by using the greedy merging strategy, we are in effect merging small components with the neighbor with the closest color, until a minimum size is reached.

4.5 LV without Removing Small Segments

The LV algorithm ends with a post-processing step, in which small components are merged with the neighbor with the closest color (line 13 in algorithm 3.1). Post-processing is common practice, not even mentioned in the description of many algorithms. We found, however, that it is crucial to achieving high recall. To show this, we ran a version of the LV algorithm where the post-processing stage is discarded.

Discarding the post-processing stage means that segments of all sizes (even one pixel) are accepted. This results in segmentations containing a large number of very small segments even though they do not represent a meaningful object part, which increases the total number of segments in the output. Thus, to obtain a prespecified number of segments, other, very large and erroneous segments must be created. The result is low recall.

4.6 Experimental Results

In this chapter, we defined several reduced versions of the LV algorithm. We now test the recall achieved by each one and compare to the original LV method (the experiments
Figure 4.1. Experiments performed on the local variation algorithm.
where performed on the Berkeley dataset [MFTM01]). Our main comparison tool is a function describing the recall against the number of segments (see section 5.3.1 for a definition of recall). Clearly, the recall is reduced monotonically with the number of segments. In each one of the algorithms the trade-off level is specified by a free parameter.

The methods under test are (a) the original LV algorithm, with $K$ being the free parameter (section 3); (b) greedy merging, selecting the number of segments (section 4.1); (c) local variation with a constant threshold (section 4.2); (d) greedy merging with distribution-adaptive weights (section 4.3); (e) area based merging, with $M$ being the free parameter (section 4.4); and (f) LV without small segment removal (section 4.5).

The recall curves for the reduced versions of the local variation algorithm are presented in figure 4.2. We can see that all the reduced versions yield lower recall than that of the original algorithm. Thus, we conclude that all the characteristics of the LV algorithm are required to achieve good oversegmentation results. Note however that for high segment counts, the area based merging method is comparable to local variation. Additionally, note that very poor results are achieved for the LV algorithm without the post-processing step where small components are removed, thus illustrating the importance of this stage.

![Figure 4.2. The local variation method against its reduced versions. No reduced version could achieve the performance of the original algorithm.](image)

**4.7 Conclusion**

We have presented a set of experiments that reveal the main characteristics of the well-known local variation algorithm. The main attribute of this algorithm is the setting
of an adaptive threshold for each segment, used to determine whether it should be merged with one of its neighboring segments. This threshold is composed of the following two equally important components:

1. A distribution dependent component, expressed as the weight of the heaviest edge in an MST of the segments.

2. A size dependent component, expressed as the number of pixels in the segment.

The experiments show that both the distribution and size dependent terms are crucial to the performance of the LV algorithm. Moreover, for high segment counts, the area-based criterion is sufficient for obtaining very good results, contrary to the case for large superpixels.
Chapter 5

Local Variation As a Statistical Hypothesis Test

In this chapter we propose two algorithms that employ statistical tools and share the same basic principles present in LV, and in this sense justify the original version of the algorithm. The first variation involves the estimation of the maximum value of a uniform distribution. The second is based on the common exponential model for the statistics of derivatives in natural images [SLSZ03]. For the latter algorithm (the probabilistic local variation algorithm), the decision to merge image segments is a hypothesis testing decision. This new algorithm has the same computational complexity and behavior as the local variation algorithm, it justifies its principles from a statistical point of view, and even yields moderately better oversegmentation results.

Thus, our contribution in this chapter is twofold:

1. We examine several probabilistic models that explain the LV algorithm.

2. We provide two new algorithms, derived using probabilistic methods and natural image statistics, which are similar to LV but are both better argued mathematically and achieve better performance. In fact, to the best of our knowledge, one of these algorithms achieves the best performance among the methods in the literature, at least for oversegmentation to many segments.

5.1 Interpreting Local Variation as Maximum Estimation

5.1.1 Estimating the Maximum of a Uniform Distribution

The threshold used in local variation has several possible statistical interpretations. In this section we consider one interpretation, which explains the local variation rule as a
test against the estimated maximum value.

Consider a set of samples drawn from a uniform distribution specified by an unknown interval \([\text{min}_U, \text{max}_U]\). We may want to estimate the parameters of the uniform distribution, \(\text{min}_U, \text{max}_U\), from the samples \(^1\). For the special case where \(\text{min}_U = 0\), let \(m\) be the sample maximum and \(S\) be the set size. Then, the minimum variance unbiased estimator (MVUE) for the maximum value of the distribution is given by

\[
\hat{\text{max}}_U = m + \frac{m}{S}.
\] (5.1)

An MVUE estimator is unbiased in the sense that the difference between the expected value of the estimation and the true value of the parameter equals zero.

5.1.2 Interpreting Local Variation as Maximum Estimation

The estimate (5.1) seems similar to the threshold expression used in [FH04]. Both expressions contain two terms. The first is a distribution related term, \(m\), which is the maximal observed value. The second term in both cases is size dependent. The difference between the two expressions is that in the maximum estimation expression (5.1), the size dependent term depends also on the maximal observed value, \(m\). Thus, we hypothesize that an LV-like process would be obtained by considering the weight values in each segment, estimating their maximum under the assumption that they come from a uniform distribution, and testing whether a new weight falls below the estimated maximum and therefore belongs to the same distribution. If the weight satisfies this test for the two segments, then it implies that merging them is justified.

Thus the algorithm is exactly like algorithm 3.1 (local variation), except that the minimum internal difference, \(M\text{Int}(C_i, C_j)\), between two components is replaced by:

\[
M\text{Int}(C_i, C_j) = \min_{x \in \{i,j\}} \left( \text{Int}(C_x) + \frac{\text{Int}(C_x)}{|C_x|} + c \right).
\] (5.2)

In theory the constant \(c\) should be 0 as it is in eq. (5.1). However, the weights we calculate suffer from severe quantization. It may often be that the maximum observed value in a small segment is simply 0, which prevents any further merges. Note also that the difference between two continuous grey levels may be almost 2, and yet, the difference between their quantized values is zero. To allow for these quantization errors, we set the constant \(c = 1\).

\(^1\)This is the continuous version of the problem known in the statistical theory literature as the German tank problem [RB47], because its solution was used by the Allies in WW2 to estimate the number of tanks produced by the Germans from the serial numbers of captured tanks. Using the continuous version accounts for sampling with replacement.
Additionally, note that the expression for the maximum estimator is parameterless. This implies that the LV-like process which uses this estimate, (5.2), is parameterless as well, and yields a particular segmentation which we cannot control. Figure 5.1a presents the results of this method. Do we obtain a segmentation which is close to the natural one? Some segments, like the sky, look natural, but overall, the number of segments seems to large. One reason could indeed be the uniform distribution assumption, which is not justified. See section 5.2 for an alternative, more successful model.

5.1.3 A Controllable Segmentation Based on Maximum Estimation

The decision rule based on the maximum estimation is parameterless and therefore does not allow us to control the oversegmentation level. A simple extension, following the LV approach, would be to change $MInt(C_i, C_j)$ to:

$$MInt(C_i, C_j) = \min_{x \in \{i,j\}} \left( Int(C_x) + \frac{k \cdot Int(C_x)}{|C_x|} + c \right),$$

(5.3)

where $k$ is a constant controlling the number of superpixels created.

Figure 5.1b shows the segmentation obtained from this algorithm with $k = 22$, chosen so that the number of superpixels is roughly the same as that obtained by the other methods shown in the figure (compare also to figure 4.1).

The maximum estimation based method (section 5.1.2) and its extension described here come relatively close to the original LV compared to other methods, both in the actual segmentations obtained and in the recall curve; see figure 5.4.

5.2 Local Variation as a Hypothesis Testing Problem

The statistical interpretation in section 5.1 is closely related to the original local variation formulation. The recall curves presented in section 5.3 show that the results are comparable as well. However, the assumption of a uniform distribution for edge weights within a segment seems unjustified. In this section, we present an alternative statistical model associated with natural image statistics.

5.2.1 Natural Image Statistics

The field of natural image statistics has been a focus of much interest over the last two decades. The general goal is to produce distribution models that will allow us to use statistical tools for compression [Sha93, BS99, BM10], noise reduction [SA96, ZW09], inference [SSD04], texture synthesis [HB95, ZWM97], colorization [BM10], etc. In order
Figure 5.1. Segmentations obtained with the methods described in this chapter. Compare to figure 4.1.
to define such models, images are characterized using different descriptors and statistical models are derived for describing them mathematically. Common descriptors include wavelet coefficients \([DBV97]\), sparse coding coefficients \([OF97]\), projections of the image on an independent component basis (Independent Component Analysis - ICA) \([Com94]\), and intensity difference between adjacent pixels \([GS01]\).

An accepted way to model the behavior of these image descriptors, initially proposed by Mallat \([Mal89]\), is by means of the generalized Gaussian distribution (also called generalized Laplacian distribution)

\[
P(x) = \frac{\beta}{2\alpha \Gamma(1/\beta)} e^{-(x/\alpha)^\beta}
\]

where \(\alpha \in (0, \infty), \beta \in [0, \infty)\) and common values for \(\beta\) are in the range [0.5, 0.8] while \(\alpha\) varies depending on the descriptor used \([SLSZ03]\). This model has been successfully used for image denoising \([ML06]\), image segmentation \([HS03]\), and for separating reflections from the base content in an image \([LW07]\).

We are interested in the weights of the graph edges which are either the absolute differences between LUV vectors or simply intensity differences. The intensity differences are closely related to some wavelet coefficient and to gradient strengths, both of which were modeled with the generalized Gaussian distribution \([Mal89, HM99]\).

The population we consider is somewhat different, however. We are interested only in weights that are part of the MST and are inside segments (and not between them). We checked the validity of the exponential model, a particular case of the generalized Gaussian distribution with \(\beta = 1\), on several images and found that the exponential assumption is reasonable; figure 5.2 describes 4 plots of the LUV edge weight statistics:

1. Weights of all edges in the image.
2. Weights of all edges within image segments, as marked by a human (as given in BSDS300 \([MFTM01]\)).
3. Weights of edges in an MST of the full image.
4. Weights of edges in an MST of each segment, as marked by a human.

Initially, note how the distributions of the edges in the MSTs are biased towards lower weights compared to the distributions obtained from the edges in the complete image (this is because the MST discards most of the heavier weights). Additionally, note that the edge weight distributions within the segments and throughout the entire image are very similar. This is because the number of edges crossing segments is very small compared to the total number of edges in the image. Thus the probabilities do not change much.
Figure 5.2. (a-d) Four natural images, (e-h) semi-log plot with histogram of weights of the following edge sets for each image: all edges in an image, all edges within human marked segments (i.e., excluding edges crossing borders), edges in an MST of an image, edges in an MST of each segment.
Furthermore, as can be seen, all distributions are close to an exponential law (observed as a straight line in the semilog plot of the figure). In this work, we shall therefore assume that the model underlying the distribution of edge weights in the MST of a (true, or manually specified) segment is the exponential distribution:

\[ P(x) = \lambda e^{-\lambda x}. \]  

(5.5)

5.2.2 Merge Decisions from Hypothesis Testing

Consider the local merging context where we need to decide whether two segments \( S_i, S_j \) merge along edge \( e \). We would like to decide whether the edge \( e \) is part of the segment \( S_i \). We propose to make this decision by testing\(^2\) the hypothesis that the weight \( w(e) \) of the edge \( e \) belongs to the distribution \( P_i(x) = \lambda_i e^{-\lambda_i x} \). If this hypothesis is rejected, the segments are not merged. Otherwise the segments are merged. We refer to this method as the probabilistic local variation approach (or pLV).

To test the hypothesis that \( w(e) \) belongs to \( S_i \), we consider the probability:

\[ P_i(x > w(e)) = \int_{w(e)}^{\infty} \lambda_i e^{-\lambda_i x} dx = e^{-\lambda_i w(e)}. \]  

(5.6)

The hypothesis is rejected with a level of significance \( \delta \) whenever \( P_i(x > w(e)) < \delta \).

Thus, the probabilistic local variation approach uses the following alternative rule for deciding whether two segments should be merged:

1. Let \( e^* \) be the edge with the minimum weight connecting two segments, \( S_i, S_j \).

2. For each segment \( S_a \in \{S_i, S_j\} \), estimate the distribution of the weights in it, \( P_a(x) \), as an exponential distribution.

3. For each segment \( S_a \in \{S_i, S_j\} \), test the hypothesis that \( e^* \) belongs to the corresponding distribution using the hypothesis test (5.6), \( P_a(x > w(e^*)) < \delta \).

4. If the hypothesis is rejected in at least one of the tests, do not merge. Otherwise merge.

In the rest of this section we describe several ways to estimate the parameters \( \lambda_i \) and the implied distribution.

A straightforward estimate for \( \lambda \) would be the maximum likelihood (ML) estimator. Given a population of \( n \) samples, \( \{x_1, x_2, ..., x_n\} \), drawn i.i.d. (independent and identically distributed), we do not test whether the two segments have the same distribution because (a) most tests of this type are nonparametric and therefore weaker, and (b) we are interested mostly in the association of the particular edge: it could be that the segments are of the same type but not of the same structure in the sense that there is a clear edge between them, causing a large LUV difference between them.
cally distributed) from the exponential distribution (5.5), then the maximum likelihood estimator for the parameter, \( \hat{\lambda}_{ML} \), is [Ros09] (section 7.6)

\[
\hat{\lambda}_{ML} = \frac{1}{\bar{x}} = \frac{n}{\sum_{i=1}^{n} x_i}. \tag{5.7}
\]

The ML estimator is, however, noise prone and highly unstable when the sample size is small, as is the case when the merged segments are small. One way to make a robust decision is by using confidence intervals (CI) for \( \hat{\lambda}_{ML} \). According to [Ros09] (section 7.6), the symmetric 100(1 − \( \alpha \))% CI for \( \hat{\lambda}_{ML} \), for a population of \( n \) samples drawn from the exponential distribution is given by:

\[
\left( \frac{\chi^2_{1-\alpha/2,2n}}{2n} \hat{\lambda}_{ML}, \frac{\chi^2_{\alpha/2,2n}}{2n} \hat{\lambda}_{ML} \right), \tag{5.8}
\]

where \( \chi^2_{p,\nu} \) is defined as follows: if \( X \) is a random variable following a \( \chi^2 \) distribution with \( \nu \) degrees of freedom, then \( \chi^2_{p,\nu} \) is the quantity \( P\{X \geq \chi^2_{p,\nu}\} = p \). A convenient and fast way to approximate this value is available [Lin88], and is described in appendix A. The expression grows almost linearly with the number of degrees of freedom, \( \nu = 2n \), for larger \( n \). For small \( n \) values, it grows slower than linear for the lower limit (\( p = 1 - \alpha/2 \)) and faster for the upper limit (\( p = \alpha/2 \)). This is apparent from figure A.1 in the appendix and also from the approximation expression in eq. (A.2). Therefore, the relative size of the confidence interval (relative to \( \hat{\lambda}_{ML} \)) decreases with the segment size, as intuitively expected.

As explained in section 5.2.3, the edge weights used for estimating \( \lambda \) are biased towards smaller values. Thus, we choose the lower limit of the CI in equation (5.8), and will reject a merge whenever

\[
P(x > w(e)) = e^{-w(e)\hat{\lambda}_{ML} \frac{\chi^2_{1-\alpha/2,2n}}{2n}} < \delta. \tag{5.9}
\]

Equivalently, we reject a merge whenever

\[
w(e) > \frac{\ln (1/\delta)}{\hat{\lambda}_{ML} \frac{\chi^2_{1-\alpha/2,2n}}{2n}}. \tag{5.10}
\]

Figure 5.4a presents the recall curve for this process. As seen, replacing \( \hat{\lambda}_{ML} \) by the lower limit of the CI for \( \lambda \) indeed yields better recall, very similar to that of LV.

Which \( \delta \) should we choose? The role of this parameter is analogous to that of \( K \) in the original local variation algorithm [FH04]: it can be used to control the number of segments in the output. Specifically, making \( \delta \) larger results in a segmentation with
more segments.

Note also that qualitatively, the algorithm behaves according to the LV principle and gives priority to the merging of smaller superpixels. This is not immediately clear, however, because changes in $\lambda$ influence other parameters. Suppose you are interested in a final segmentation with $N$ superpixels. Reducing $\lambda$ to the lower limit of the confidence interval must be compensated for by increasing $\delta$. Otherwise, the threshold will be higher, more merges will be performed, and the number of superpixels in the final segmentation will be smaller than $N$. This compensation, however, is non-uniform. While the level of significance $\delta$ is uniform and applies to all merging decisions, smaller segments are associated with larger confidence intervals, which makes the lower limit of their confidence intervals lower on the average. Therefore, the threshold on the edge weight (5.10) is higher for smaller superpixels, which gives them priority to merge.

While this model achieves nice segmentations and sheds some new light on the LV algorithm, we are not completely satisfied with it. This is because the statistical assumption underlying the estimation of $\lambda$ – that the measurements are obtained by i.i.d. sampling – seems inaccurate. First, the samples we have are sampled in nearby locations, which makes them correlated. Moreover, they are sampled with strong preference for lower weight edges. The first problem seems not to make a big difference because the expected value, used for estimating lambda (5.7), is the same even if the samples are correlated. The second problem, preferring smaller values, is considered in the next section.

### 5.2.3 Parameter Estimation under Bias Sampling

Contrary to the assumption used in section 5.2.2, our samples are not drawn i.i.d. Following the local variation algorithm, we first sort all the edges in the image according to their weight, and then iterate through them from the lightest to the heaviest, performing our test. The implication, is that by the time we test edge $e$ with weight $w(e)$, all the edges $e_{S_1} \in S_1$ hold the property $w(e_{S_1}) \leq w(e)$. In other words, at each step the population inside a segment is biased towards lower values. In what follows we aim to correct this bias.

Suppose the segment has $m$ pixels, and at some point in time our method of sampling provides us with $n$ measurements, which, as stated before, are not i.i.d. Specifically, we have seen the lower $n$ samples in the population. Then, the task can be stated as follows: let $\{x_1, x_2, ..., x_m\}$ be $m$ samples drawn i.i.d. from the exponential distribution $\lambda e^{-\lambda x}$, such that for every $i < j$ we have $x_i \leq x_j$. From this set of $m$ samples we have only seen the smallest $n$ ones, $\{x_1, ..., x_n\}$ ($n < m$), and we want to estimate the value of $\lambda$. 

39
The structure of this problem is the one defined by type II censoring, in which \( m \) random variables are drawn but only the smallest \( n < m \) values are observed. This type of sampling is common in reliability estimation, where one tries to study the failure rate of some process/machine by analyzing only a subset composed of the samples which failed first [ES53]. Under these conditions, and following [ES53], the maximum likelihood for \( \lambda \) can be derived as follows:

A sequence of size \( m \) is called partially \( n \)-ordered if the first \( n \) elements in it are non-decreasing (or non-increasing), and not larger (or not smaller) than any of the remaining \( m - n \) elements. Consider a sequence, \( Y \), of \( m \) samples drawn i.i.d. from some probability distribution. A partially \( n \)-ordered sequence, \( X \), may be generated from it by choosing the \( n \) smallest elements from \( Y \), and making them the first \( n \) elements of \( X \), in non-decreasing order. The remaining \( m - n \) elements in \( X \) are identical to the remaining \( m - n \) elements in \( Y \). Their order is the same order as in \( Y \).

Note that many sequences \( Y \) may correspond to the same partially \( n \)-ordered \( X \). The first \( n \) elements may come from \( H = \frac{m!}{(m-n)!} \) different sets of locations, implying that the probability of getting a particular partially \( n \)-ordered sequence is \( H \) times the probability of getting the original sequence \( Y \) of i.i.d. drawn samples. Therefore, the joint probability density of observing the values \( x_1, \ldots, x_n \) is

\[
\begin{align*}
f(x_1, \ldots, x_n) &= H \cdot P(x_1, \ldots, x_n) \cdot P(x_{n+1} > x_n, \ldots, x_m > x_n) \\
&= H \cdot P(x_1) \cdots P(x_n) \cdot P(x_{n+1} > x_n) \cdots P(x_m > x_n) \\
&= H \cdot P(x_1) \cdots P(x_n) \cdot \left( \int_{x_n}^{\infty} \lambda e^{-\lambda x} \, dx \right) \cdots \left( \int_{x_n}^{\infty} \lambda e^{-\lambda x} \, dx \right) \\
&= H \cdot e^{-\lambda x_1} \cdots e^{-\lambda x_n} \left( e^{-\lambda x_n} \cdots e^{-\lambda x_n} \right) \\
&= H \lambda^n \cdot e^{-\lambda \sum_{i=1}^{n} x_i} \cdot e^{-\lambda (m-n)x_n} \\
&= H \lambda^n \cdot e^{-\lambda \left( \sum_{i=1}^{n} x_i + (m-n)x_n \right)}.
\end{align*}
\]

(5.11)

The value of \( \lambda \) yielding the maximum likelihood for this function can be found by differentiation with respect to \( \lambda \) and equating to zero. Note that the maximum of \( f(x_1, \ldots, x_n) \) is independent of \( H \) and reached at the same point as the logarithm of this function. Hence,

\[
\hat{\lambda}_{C-ML} = \arg\max_\lambda \left( \ln \left( \frac{f(x_1, \ldots, x_n)}{H} \right) \right) \\
= \arg\max_\lambda \left( \ln(\lambda^n) - \lambda \left( \sum_{i=1}^{n} x_i + (m-n)x_n \right) \right),
\]

(5.12)

where the notation C-ML is for censored maximum likelihood.
Differentiating and equating to zero, we get the ML estimation of $\lambda$

$$\hat{\lambda}_{C-ML} = \frac{n}{\sum_{i=1}^{n} x_i + (m-n)x_n}. \quad (5.13)$$

Finally, consider a segment $C$ with edge weights being $\{x_1, ..., x_n, ..., x_m\}$, and the merging decision when the edge under study is $e$. Then, combining equations (5.10) and (5.13), the condition for merge rejection becomes:

$$w(e) > HypThr(C) = \frac{2 \ln(1/\delta) (\sum_{i=1}^{n} x_i + (m-n)x_n)}{\chi^2_{1-\alpha/2,2n}}, \quad (5.14)$$

where $m$ is a user specified parameter reflecting the expected size of the true segment. In the case $m = n$, $\hat{\lambda}_{C-ML}$ converges to $\hat{\lambda}_{ML}$, which we use also for $n > m$.

In summary, our hypothesis testing algorithm for image oversegmentation is exactly as algorithm 3.1, but the minimum internal difference between two components is set to:

$$MInt(C_i, C_j) = \min_{x \in \{i,j\}} HypThr(C_x). \quad (5.15)$$

The value of $\lambda$ for a given segment characterizes the distribution of its edges. The larger the variability within a segment, the smaller the value of $\lambda$. Figure 5.3 depicts this fact. Each segment is given a color according to its $\hat{\lambda}$: the brighter the segment, the larger $\hat{\lambda}$. Thus smooth segments are presented white, while textured ones are black. Figure 5.1c presents an example segmentation of the pLV method with correction for biased sampling.

Probabilistic local variation with censoring is based on different principles and yet follows the same basic behavior as the local variation algorithm. We consider the decision condition (5.14) and observe the following three properties:

1. As discussed above, the denominator $\chi^2_{1-\alpha/2,2n}$ grows with the size of the segment, $n$, while the numerator decreases with $n$. Thus, the threshold decreases with $n$ and gives preference for merges of small segments.

2. The weight of the heaviest edge in the MST of a segment, $x_n$, appears in the numerator of the threshold function. Therefore, a heavier $x_n$ leads to a higher threshold and to a more likely merge decision.

3. The behavior is actually more complex: for small segments, where $n \ll m$, the heaviest edge, $x_n$, is indeed the most important, and its importance is amplified by a factor linear in $(m-n)/n$. For larger segments, though, the amplification factor is smaller, making the heaviest edge less important and the average weight of the segment edges more important.
Figure 5.3. Right: visualization of \( \hat{\lambda} \) for each segment. The brighter the segment, the larger \( \hat{\lambda} \). Thus white represents smooth segments and black represents textured ones. Left: corresponding oversegmentation.
5.2.4 Properties of the Segmentation

As stated in section 3, it is shown in [FH04] that the segmentation obtained with the LV algorithm is neither too fine nor too coarse.

For the pLV algorithm, the segmentation obtained holds the property of not being too fine. This can be demonstrated exactly as it was for the LV algorithm and follows from the definition of a too-fine segmentation, and the predicate defining the evidence of an edge between two components in equation (3.5).

If the condition \( \text{HypThr}(C) > \max_{e \in \text{MST}(C)} w(e) \) holds for every segment, then the segmentation is also not too coarse. In this case, the demonstration for the LV case is also valid for pLV. Let us examine this condition:

The factor \( \sum_{i=1}^{n} x_i + (m - n)x_n \) in equation (5.14) is always larger than \( \max_{e \in \text{MST}(C)} w(e) \) since by definition \( x_n = \max_{e \in \text{MST}(C)} w(e) \). Thus we require

\[
\frac{\ln(1/\delta)}{\chi^2_{1-\alpha/2,2n}} > 1
\]

\[
\delta > e^{-\chi^2_{1-\alpha/2,2n}/2}
\]

(5.16)

for the segmentation to be not too coarse.

5.2.5 Computational Complexity of Hypothesis Testing

In order to calculate the value of equation (5.14), all we need is to use the approximation for \( \chi^2_{p,\nu} \) (eq. (A.2)) and keep three values for each segment: the sum of its elements, the value of the last (heaviest) edge added to it, and its number of elements. All these values are updated in \( O(1) \), and thus the complexity of the hypothesis testing method is exactly the same as that of the local variation algorithm, namely \( O(n \log n) \).

5.3 Experimental Results

5.3.1 Quality Measures and Data

Following common practice, we use the boundary recall and the undersegmentation error as quantitative measures for the performance of several algorithms. For oversegmentation algorithms, which, by definition, allow segments to split, the recall is the more important criterion.
The recall [MFM04] is the fraction of ground truth boundary pixels that are matched by the boundaries defined by the algorithm. Let $\delta G$ be the set of ground truth boundary pixels, $\delta S$ be the set of boundary pixels found by the algorithm, and $1(.)$ be the indicator function. Then the recall is defined as

$$R_G(S) = \frac{\sum_{p \in \delta G} 1\left(\min_{q \in \delta S} \|p - q\| < \epsilon\right)}{|\delta G|}.$$  \hfill (5.17)

In our tests we use $\epsilon = 2$.

The undersegmentation error [LSK+09] is a measure of segment “bleeding.” If $G = \{G_i\}_{i=1}^M$ is a ground truth segmentation with $M$ segments, and $S = \{S_i\}_{i=1}^N$ is the segmentation created by an algorithm, then the undersegmentation error is defined as follows:

$$UE_G(S) = \frac{\sum_{i=1}^M \sum_{j : |S_j \cap G_i| \neq 0} |S_j \setminus G_i|}{\sum_{i=1}^M |G_i|}. \hfill (5.18)$$

The experiments presented in this section were performed on the Berkeley dataset (test) [MFTM01]. To accommodate for the multiple ground truth segmentations available from this dataset, we average the recall and undersegmentation error over all ground truth versions for a given segmented image.

In all our experiments (except when explicitly specified), we perform a post-processing step to remove segments smaller than 10% the average requested segment size (this is also performed for the basic local variation algorithm). In other words, if we want a segmentation with $S$ segments, and the input image is of size $ImageSize = M \times N$, then all segments of size $0.1 \times ImageSize / S$ are removed.

### 5.3.2 Testing Probabilistic Versions of the Local Variation Algorithm

Here, we experiment with the probabilistic interpretations of the local variation algorithm developed through sections 5.1 and 5.2 and compare them to the original version.

The first probabilistic version uses a uniform distribution model for the edge weights and estimates the interval specifying this uniform distribution. This method gives a single segmentation (see figure 5.1a). It is difficult to compare its average recall to that of local variation, because the segmentation of different images results in different numbers of segments (and different recall). Therefore, to facilitate the comparison, we generalized the method so that it depends on a parameter, similarly to LV, by giving control on the additive term of the function $MInt$ (see section 5.1.3). The recall curve associated with this approach is shown in figure 5.4a. It is clearly inferior to that of local variation but in a sense it behaves similarly and is better than several of the other methods in the literature.
The probabilistic local variation approach, which relies on the exponential distribution characterizing natural image statistics and uses hypothesis testing as its decision mechanism, is more successful. One should be careful when estimating the exponential model parameter, $\lambda$. The simplest estimation, based on maximum likelihood, overestimates $\lambda$, which yields poor recall; see figure 5.4a. Correcting it using the confidence interval improves the results significantly and results in the same recall as local variation. Observing that the $\lambda$ estimation is biased to be larger than the real value (see section section 5.2.3), we chose the lower limit of the confidence interval. The bias also implies that there is still room for improvement.

Indeed, the censored estimation, which takes the bias into account, gives a more accurate estimate of $\lambda$ and yields excellent, state of the art, results, which are even somewhat better than those of LV.

The pLV algorithm (in its censored estimation version) depends on two parameters: $m$, the expected size of the segment, and $\delta$, the level of significance of the decision (see eq. (5.14)). In principle, the number of segments may be controlled by any combination of them. We experimented with various combinations and suggest that the parameters be set as follows:

1. For situations where the number of segments is prespecified as $S$, we propose to set $m = \text{ImageSize}/S$ and to tune $\delta$, so that the required number of segments is achieved. For example, for obtaining, say, 500 segments in an image from the Berkley Dataset, $m = \frac{321 \times 481}{500} = 308.8$. We found that this value works well for all numbers of segments in the range $S \in [200, 2000]$.

2. For situations where the accurate number of segments, $S$, is less important, and where faster operation is required, we suggest to find $m, \delta$ by empirically maximizing the average performance, which depends on the set of images and on the task. Then, using these parameters for a new image will provide adaptive segmentation, yielding more segments on “busy” images and fewer segments on smooth ones. See figure 5.5, where several images were segmented with the same set of parameters ($m = 350, \delta = 0.05$).

To obtain the recall and undersegmentation error plots presented in figure 5.4, we use the first option. Additionally, for the cases where a confidence interval is taken into account, we use a 95% CI (i.e., $\alpha = 0.05$).

In [ASS12], SLIC superpixels are compared to local variation [FH04], NCut [SM00], Turbopixels [LSK09], Quick Shift [VS08], and an energy optimization method [VBM10]. Their results in terms of recall and undersegmentation error are similar to our review presented in figure 3.1, where local variation and SLIC superpixels present the best recall and undersegmentation error respectively. Additionally, their results show that LV has the second best running time – almost the same as that of the best method.
Figure 5.4. (a) Recall and (b) undersegmentation error of the local variation method against its probabilistic versions proposed in this chapter. While the algorithms behave similarly, the proposed hypothesis test (with censored estimation) performs best for medium to large segment counts.
Figure 5.5. Four images segmented with the pLV algorithm (censored version) with parameters $m = 350, \delta = 0.05$. The number of segments depends on the characteristics of the image, yielding 344 segments for image a, 77 for b, 244 for c, and 261 for d.

The running time of our method is exactly the same as LV, and the recall is better for large numbers of segments. The undersegmentation error is also better than that of LV, yielding results similar to those of SLIC superpixels, which, as stated before, performs best for this measure; see figure 5.4b.

5.4 Conclusion

In this chapter we presented insight on the LV algorithm by building on mathematical tools from the fields of probability and statistics. We developed a hypothesis testing method called probabilistic local variation which follows the same methodology as LV. Furthermore, pLV uses an adaptive threshold including the same two components present in LV’s threshold (a size dependent component and a distribution dependent component), but it is derived by probabilistic analysis based on the properties of natural image statistics.

Our experiments show that pLV outperforms local variation, while maintaining the same computational complexity, making it a good choice for the task of general image oversegmentation.
Chapter 6

Multistage Learning Based Oversegmentation

Single-linkage clustering has been proposed as an algorithm for image segmentation, however, its greedy decisions result in poor performance. Oversegmentation, on the other hand, which aims to divide an image into more pieces than objects, can benefit from the single-linkage framework due to its simplicity and quick computation. In this chapter, we present three modifications to single-linkage which result in state-of-the-art performance for the oversegmentation task. Specifically, we use machine learning methods to learn dissimilarities between superpixels and use these dissimilarities as distances between clusters, introduce a multistage approach to compute robust features, and add a correction mechanism which includes global information to overcome mistakes introduced by the greedy decisions. We show that using this modified single-linkage method on top of an existing oversegmentation algorithm improves its results.

6.1 Single-Linkage Agglomerative Oversegmentation

We focus here on a particular type of oversegmentation algorithms: single-linkage agglomerative algorithms. Like many segmentation methods, these algorithms are originated from clustering algorithms. Clustering algorithms consider a set of elements and aim to divide it into subsets so that, in some sense, the similarity within each subset is larger than the similarity between subsets. Single-linkage clustering is perhaps the simplest clustering method. It builds on a dissimilarity (distance) \(d(x, y)\) specified between every two elements, and specify the distance between two clusters \(D(X,Y)\) as the minimal distance between the two closest elements, one from each cluster:

\[
D(X,Y) = \min_{x \in X, y \in Y} d(x, y),
\]

(6.1)
where \(d(x, y)\) denotes the distance between elements \(x\) and \(y\). A simple agglomerative single-linkage algorithm starts with each node being a subset, and performs iterative merges between the two clusters with the smallest inter-cluster distance until a stopping criterion is reached [DHS01, section 10.9]. It is common to describe the clustering task using a weighted graph \(G = (V, E)\) with a vertex for each element, edges defining a neighborhood system, and a weight function \(w : E \rightarrow \mathbb{R}\) being the distance. It is easy to see that merges occur only on the edges of the graph’s minimum spanning tree (MST), and the process is similar to Kruskal’s algorithm [Kru56].

This process was adapted to image segmentation [FP02, section 15.3]. One may, for example, specify the initial elements as pixels, and use the gray level difference as the distance. Then, running the iterative merging process leads to a clustering which specifies an image segmentation. This approach may succeed with simple images containing uniform intensity regions but usually fails for real images. The reason is that the method is not robust: two very different regions may erroneously merge if there is a single path of pixels with similar pairwise intensities. Surprisingly, even simple modifications may increase robustness and lead to excellent algorithms; e.g. [FH04], see below.

The general structure of modified single-linkage agglomerative oversegmentation algorithms may be specified by three main stages:

**Initialization:** Specifying the weighted graph - Here the nodes \(V = \{v\}\), the edges \(E = \{(u, v)\}\), and the weights \(\{w(u, v) | (u, v) \in E\}\) are specified. Node attributes \(\{a(v) | v \in V\}\) may be specified as well.

**Iterative merging** - Here the edges \((u, v)\) are scanned in nondecreasing weight order. If the edge is between two clusters, then they are merged depending either only on \(w(u, v)\), or, in the more general modified version on the the graph properties related to \((u, v)\) (e.g. \(w(u, v), a(u), a(v)\)). The merge may include an update of the graph properties. This process continues until some stopping criterion is satisfied.

**Postprocessing** - Here the segmentation is further modified by a different process. It is common for example, to merge all very small segments with their neighbors.

The LV algorithm [FH04] is an example of a modified single-linkage agglomerative oversegmentation. It implements the initialization step by defining each pixel in the image as a segment on its own, and setting the dissimilarity between elements as the absolute intensity difference between pixels; local merges are performed by comparing the dissimilarity between the segments to an adaptive threshold which depends on the segments’ sizes and on the dissimilarities within them; and a post-processing step removes small segments by merging them to the neighboring segment with the smallest color difference. Using this adaptive threshold makes the algorithm very fast (complexity \(O(n \log n)\)) and yet much better than single-linkage agglomeration or modified agglomeration algorithms that use simpler thresholds.
Figure 6.1. Recall for several oversegmentation methods in the literature.

The pLV algorithm (section 5.2) is based on the modified single-linkage structure as well, but replaces the threshold used in LV by a hypothesis test procedure which relies on natural image statistics. Both algorithms achieve state-of-the-art results using the single-linkage agglomerative structure. They are fast due to the simplicity of the algorithm, but also accurate thanks to the non-trivial merging decision; see figure 6.1.

6.1.1 Our Contribution

In this chapter we propose several variations on the modified single-linkage oversegmentation algorithms. These variations use three main principles:

1. **Learn the non-similarities** - We propose to replace the color distance dissimilarity function by a learned function, which may be either based just on a few features or on a richer feature set. The advantage of learning the dissimilarity is twofold. First, the learning procedure can find a function that discriminates true boundary better than simple color difference. In addition, learning the dissimilarity makes the merging decision and process simpler because, for example, the graph update becomes much less frequent, making most of the algorithm equivalent to Kruskal’s MST algorithm.

2. **Multistage** - We propose to run the oversegmentation process in stages, everyone of which recalculates the graph using the merged regions as nodes. Two main advantages result from this multistage approach. First, training the dissimilarity function only on one set of superpixels is not optimal because superpixels of different sizes have different useful properties. In particular, the learning does not help much for the first stage where the nodes correspond to single pixels. The
second advantage is that properties of larger superpixels can be made more robust, which is especially important for the greedy algorithm.

3. **Correct bad connections** - Single-linkage agglomerative algorithms suffer from non-robustness: a single wrong connection suffice for two full segments to merge incorrectly. Therefore, we propose a correction mechanism in the postprocessing stage. The mechanism, termed backtracking (BT), tests the segments obtained by the previous agglomerative stage and decomposes them into subsegments according to a non-single-linkage criterion. This compensates for the greedy, non-robust, mechanism, and yields in a significant improvement over learning based algorithms.

The algorithms either match the performance of the state-of-the-art algorithms or improve it. In particular we provide an algorithm that is almost as fast as the LV / pLV approaches but is more accurate, and a not-so-fast algorithm that is significantly better than both.

These principles were used before in other algorithms, but not in a way that leads to efficient and fast implementation, as required in oversegmentation. Edge detection decision functions were studied in many papers but they usually result in relatively slow decisions [MFM04, DTB06]. A multistage approach with new features added as the segments grow was considered for example in [AGBB12]. Interestingly, this work uses soft, delayed, decisions, which, in a sense, are related to our correction approach which allows to reverse a merge decision.

The remaining of this chapter is organized as follows. In section 6.2 we present the general structure of our single-linkage oversegmentation algorithm, in section 6.3 we elaborate on the features we use and the classifier, section 6.4 presents experimental results, and finally, we conclude in section 6.5.

### 6.2 Oversegmentation Algorithm

The proposed oversegmentation algorithm is composed of stages. Every stage starts with a set of superpixels (SPs) and returns another set, containing a smaller number of superpixels. Every stage is composed of 4 steps: graph initialization, SPs merging, and two types of postprocessing. These steps are described below and afterwards the multistage structure is discussed. The actual classifiers that we use and the associated training process are discussed in the next section.

In principle, we could start from a superpixel set where every superpixel is simply a pixel, but we found that the proposed process works only when the initial superpixels are larger. Any oversegmentation algorithm can be used to generate the first superpixel set. We use the probabilistic local variation algorithm (section 5.2.2) which is both fast
and accurate for fine oversegmentation. In our experiments we set the number of initial superpixels to $N_0 \approx 2000$.

### 6.2.1 Initialization - Weighted Graph Definition

Let $S^{in}_i$ be the initial segmentation for the $i$-th stage. The number of superpixels in $S^{in}_i$ is denoted $N_i$. The graph $G = (V, E)$, describing the initial oversegmentation is constructed. Its nodes correspond to the initial superpixels, $|V| = N_i$. The edges of the graph are defined between adjacent superpixels.

The weights of the edges in the graph are determined by the response (decision value) of a trained classifier. A classifier is used because it can combine different features and yield an effective estimate indicating whether the SPs are parts of a single true segment or not. The classifier takes a set of features characterizing the edge and the corresponding pair of adjacent superpixels. These features are calculated as part of the initialization. We experimented with different features, and found that for small superpixels it is best to use only two features, which we now describe.

1. Absolute LUV Difference: Let $LUV_i$ be the mean color description (in the perceptual L*u*v* color space) within the $i$-th superpixel. Then,

$$D_{LUV}(i, j) = \|LUV_i - LUV_j\|_2. \quad (6.2)$$

2. Relative Gradient: Let $G(x)$ be the gradient magnitude of the gray level version of the input image at pixel $x$, and $BorderSet(i, j)$ be the set of pixels in the border between superpixels $i$ and $j$. Then,

$$G_{Segment}(i) = \text{median}\{G(x) | x \in SP_i\}$$
$$G_{Border}(i, j) = \text{median}\{G(x) | x \in BorderSet(i, j)\}$$
$$D_{RelativeGrad}(i, j) = \frac{G_{border}(i, j)}{\max(G_{segment}(i), G_{segment}(j))}. \quad (6.3)$$

Note that this feature relies on a comparison between the gradients on the boundary and the typical gradients within the segments; this is similar, in principle, to the approach taken in the LV (and pLV) algorithm. The similarity does not stop here: thinking about the gradients as differences between gray levels in adjacent pixel, implies that there are roughly $2n$ such values for a SP containing $n$ pixels. The median, or the $n$-th largest value, roughly corresponds to the highest value in the MST, used in [FH04].

Richer sets of features could be used and are indeed used when the oversegmentation process progresses and the superpixels become large; see section 6.3. We found how-
ever that the classifier based on these two simple features goes a long way, and the improvement with richer feature sets is not very big.

For each edge the two features are calculated and fed into a naïve Bayes classifier [Elk97] and the resulting decision value is used as the edge weight. Note that the weight is used only for ordering the edges and therefore transforming the decision value into an estimated probability (as done, e.g. in [Pla99]), is not needed.

The graph does not change throughout the stage. However, at the end of the stage a new graph is defined in which neither the number of nodes, the edges, nor the weights, stay as specified in the first, initialization step.

6.2.2 Iterative Merges

This process is a single-linkage agglomerative clustering. It start by defining every node (superpixel) as a separate component. That is, $N_i$ components are defined. Then, iteratively, choose the lowest weight edge corresponding to two SPs, merge the components associated with the two SPs, unconditionally, and repeat until a prespecified number of components, $N_{i2} < N_i$, is achieved. Note that every component corresponds to a set of nodes and to a minimum spanning tree (MST) $T_j = (V_{T_j}, E_{T_j})$, containing the edges on which the merges occurred.

6.2.3 Postprocessing I: Removing Small Segments

The previous process often leaves many very small segments that correspond to small details in the scene, to highlights on object boundary, etc. These segments may significantly differ in color from their neighbors and are therefore not merged with them in the iterative merging phase. Being small, they are not significant parts of important segments that are usually relatively large. Interestingly, it is not possible to learn the size effect from human annotation because the later refers to true segments which are much larger than both these small segment and the normal superpixels. Therefore, like most oversegmentation algorithms, we propose to remove these small segments by merging them with the most (color-wise) similar segment. We consider a segment small when its size is a small fraction, $\alpha$, of the average segment size or smaller. After these segments are removed the number of components is further reduced to $N_{i3}$. Figure 6.2 presents an example of small component removal. Note how the recall is practically the same, while the number of segments is strongly reduced.
6.2.4 Postprocessing II: Backtracking

Single-linkage agglomerative algorithms suffer from non-robustness: a single path of pairwise similar nodes (superpixels) suffice for two full segments to incorrectly merge. Therefore, we propose to add a correction mechanism. The proposed correction mechanism examines the segments obtained by the agglomerative merging stage, one by one, go over all merged edges and examines the merging decision with respect to a more robust criterion. If, for a particular segment, a merge that is not justified is found, the segment is decomposed. This step increases the number of SPs and compensates for the greedy, non-robust, merges.

Let $A = \{T_j = (V_{T_j}, E_{T_j})\}, 1 \leq j \leq N_{i3}$ be the set of MSTs of the connected components of $G$. Then, we search for the edge in all the MSTs which specifies the worst cut within the component. Consider an edge $e$ in one of the trees $T_j$. Let $T_{1je}, T_{2je}$ be the two trees obtained by removing edge $e$ from tree $T_j$. The edge specifying the worst cut is:

$$e_{\text{worst-cut}} = \arg\max_{e \in \bigcup_{j=1}^{N_{i3}} E_{T_j}} \sum_{v_1 \in T_{1je}}^{v_2 \in T_{2je}} w((v_1, v_2)) \quad (6.4)$$

The edge specifying the worst cut is removed, increasing the number of components by one. The process is repeated iteratively until the number of components grows to the prespecified $N_{i4}$. A straightforward implementation of this algorithm has complexity $O(n^2)$. The rational of this process is to remove unjustified merges due to accidentally weak edges. The cut cost is much more robust than the single linkage criterion simply because it relies on more edges.

Figure 6.3 presents a schematic example of the backtracking procedure. Figure 6.4 shows an example of segmentation correction by using backtracking.
Figure 6.3. Example of backtracking: \( S^0 \) is the initial segmentation with each segment being a vertex of the graph. \( S^1 \) is the segmentation obtained by greedy merging when stopping with two segments, the edge with largest weight \( 20 \) has been removed defining to connected components: \( SP^1_1 \) and \( SP^1_2 \). The thick lines in \( S^1 \) are edges in an MST of each of the components. Finally, the optimal cutting edge is found to be \( e_{cut} = (SP^0_1, SP^0_{10}) \). This is the edge from those in the MSTs that yields the largest value for equation (6.4) (yielding exactly the value \( 15 + 15 + 1 = 31 \) which is the sum of the edges between \( T_{12e_{cut}} \) and \( T_{22e_{cut}} \)). Once this edge is removed three trees remain, defining the final three components of segmentation \( S^2 \).
6.2.5 Multistage Segmentation

The process described in sections 6.2.1-6.2.4 is repeated iteratively, in several stages. Within each stage, the actual operation is specified by the number of components resulting from every step, \( N_{i2}, N_{i3}, N_{i4} \). The components produced by the last step in the \( i \)-th stage are the initial superpixels set for the \((i + 1)\)-th stage.

We found that a multistage algorithm performs better than a single stage. In fact, without backtracking, a single stage algorithm performed worse than the reference pLV. A four stage algorithm performed much better, with backtracking or without it. We actually found that performance increase monotonically with the number of stages and that say, a 10 stage algorithm performs slightly better than the 4 stage algorithm; see figure 6.5 for a schematic diagram.

Figure 6.5. A scheme describing the evolution of SP number through the algorithm.
Several operations can take place when starting a new stage. The benefits of each operation are discussed next:

1. We may recalculate the features that are used by the classifier to specify the weights. This is the most basic change and it is always done. The advantage in feature recalculation is that the features become more stable and more robust. They become more stable because averages are calculated on larger regions. This is the case, for example, with the LUV distance. They become more robust because as the SPs become larger, the median used in edge dependent features (e.g. the relative gradient feature) is calculated over a larger set. With this minimal change, no change in the features or in the classifier takes place. Yet, the effect is still significant.

2. In addition to recalculating the features, we can also change the classifier. A relatively small change would be to use the same features but to train the classifier on another set of SPs, corresponding to the increased SPs average size.

3. Finally, we can also change the features themselves. This makes sense because larger SPs allow to calculate more informative features. With this option we need to train several classifiers corresponding to different SP sizes and different features.

The final selection of number of stages, the parameters of every stage and the changes between the stages affects the performance of the algorithm. Usually, adding features or retraining does not decrease the classifier’s accuracy and the corresponding oversegmentation performance. Therefore, choosing operations to perform at the end of each stage, the number of features, etc. is a compromise between accuracy and complexity. We experimented with many options and came to several conclusions: first, having several stages and recalculating the features in each of them is better than having a single one, but the performance improvement with more than 3-4 stages is small. We set the number of superpixels in the end of each stage \((N_{i4})\) so that it is roughly halved, \(N_{i4} \approx 0.5N_i\), and the performance is not sensitive to the exact value. Thus, \(N_1 = 2000, N_2 = 1000, N_3 = 600, N_4 = 300\). On the other hand, training the same classifier with the same features over different sizes of SPs does not improve performance. Finally using a rich set of features improves performance but only at latter stages where the SPs are relatively large and their number is small.

As for the inner parameters of each stage, \(N_{i2}\) and \(\alpha\), we optimize them on a grid \((N_{i2} \in \{0.8, 0.9, 1.0, 1.1, 1.2, 1.3\}N_{i4}) ; \alpha \in \{0.1, 0.2, 0.3\}\) for the different stages and found that the best values are roughly \(N_{i2} = 1.1 \cdot N_{i4} ; \alpha = 0.2\). Note that the number of removed small SPs is not directly specified as it depends on the size distribution. Note also that the amount of BT is also indirectly specified by the difference between the final number of segments \(N_{i4}\) and the number of SPs left after small SP removal. This simple rule surprisingly yielded the optimal (or close to optimal) values for all
6.3 Classifiers

The classifiers are used to set the weights of the edges in the segmentation graph. They should discriminate between graph edges corresponding to superpixel pairs where the two superpixels belong to the same object, and edges corresponding to SP pairs that correspond to different objects. Therefore, they are actually edge detectors. We could use, in principle, existing, learning based, edge detectors, such as the Pb detectors [MFM04] or the BEL detector [DTB06]. These edge detectors however are relatively slow. Therefore we prefer to developed our own classifiers.

In contrast to common edge detectors which test every image point in all orientations, the classifiers considered here classify pairs of superpixels of the initial oversegmentation. Note that the number of objects to classify is much lower.

We experimented with many features and classifiers. Intuitively, using many features and a complex classifier is computational costly. We eventually concluded that for fine oversegmentation (many superpixels), using many features does not substantially improve the accuracy and that two features suffice to achieve very good results. Working with two features we can work with the naïve Bayes classifier [Elk97], which performed slightly better than other choices. We refer to this classifier as the lean (feature) set classifier. For coarser oversegmentation, a richer set of features improves the accuracy. We experimented with many features and eventually suggest to use a set of 7 features. In this case, the naïve Bayes classifier is no longer practical and the random forest classifier [Bre01] is used. We refer to this classifier as the rich set classifier.

6.3.1 Features

The features used in our classifiers are described next. The first two features are used for the lean set classifier. All the features are used for the rich set classifier.

1. Absolute LUV Difference - see section 6.2.1 for details.
2. Relative Gradient - see section 6.2.1 for details.

3. Relative LUV difference - Here we calculate the difference relative to the neighbors minimum similarity. Let \( \text{neigh}(i) \) be the set of neighboring SPs of superpixel \( i \),
then
\[
R(i) = \text{mean}(\{D_{LUV}(i, j) \mid j \in \text{neigh}(i)\})
\]
\[
D_{RelativeLUV}(i, j) = \frac{D_{LUV}(i, j)}{\max\{R(i), R(j)\}}.
\] (6.5)

\(D_{LUV}(i, j)\) is specified in eq. (6.2). Variations, such as normalizing the distance by \(\min, \text{median}, \text{mean}(R(i), R(j))\) are possible but were found to be less effective.

4. Absolute Gradient - specified simply as
\[
D_{AbsoluteGrad}(i, j) = \text{median}(\{G(x) \mid x \in \text{BorderSet}(i, j)\})
\] (6.6)

where \(\text{BorderSet}(i, j)\) the set of pixels in the border between superpixels \(i\) and \(j\). The median is used to enhance robustness.

5. Maximum Segment Size: Let \(|SP_i|\) denote the number of pixels in segment \(SP_i\). Then,
\[
D_{MaxSize}(i, j) = \max(|SP_i|, |SP_j|)
\] (6.7)
\[
D_{MinSize}(i, j) = \min(|SP_i|, |SP_j|).
\] (6.8)

The maximum SP size \(D_{MaxSize}(i, j)\) is one feature, and

6. The size ratio is another feature
\[
D_{SizeRatio}(i, j) = \frac{D_{MinSize}(i, j)}{D_{MaxSize}(i, j)}.
\] (6.9)

7. Texture: We use the TPLBP filter [WHT11] to characterize texture in the image. This filter is applied on the full image with parameters \(r = 2, \alpha = 4\) (selected empirically), and for each SP we create a histogram with 256 bins - one bin for each possible output of the filter. Then, the dissimilarity between a pair of neighboring SPs is calculated as the Jensen-Shannon divergence between the corresponding histograms. Let \(a, b\) be two discrete distributions with \(n\) bins. Then
\[
JS(a, b) = \frac{1}{2} KL(a, (a + b)/2) + \frac{1}{2} KL(b, (a + b)/2).
\] (6.10)

where \(KL(p, q)\) is the Kullback-Leibler divergence.

Figure 6.6 shows some of the distances that have been consider as classifier features.
Figure 6.6. Distances between superpixels for several features used in the classifiers. $D_{LUV}$ and $D_{AbsoluteGrad}$ complement each other, note for example the rightmost edge of the statue to the right (visible only in $D_{AbsoluteGrad}$) and the bottom edges in all three statues (visible only in $D_{LUV}$). $D_{MaxSize}$ differentiates SP pairs containing big segments.

6.3.2 Training Process

For training, we use adjacent superpixel pairs that are obtained from the same oversegmentation algorithm used to obtain the initial segmentation (specifically, we use pLV). Ideally, the training should be done on SPs created by the proposed process and should match the size of the SPs in the training set to those that are classified. Instead we used the small SPs of the size available as initial SPs set to the agglomerative process.

The examples were annotated with the help of manual segmentation annotation [MFTM01]. A SP was assigned to the human marked segment with the largest overlapping area. Then, a pair of two adjacent SPs was considered a positive (+1, do-not-merge) example if both SPs were assigned to different human marked segments. Otherwise the SP pair is considered a negative example (-1, merge). When several observers provided different manual segmentation, the examples were weighted proportionally.

We note that due to the large number of SPs, there are much more negative examples than positive examples. However the single linkage agglomerative segmentation penalizes incorrect merges much more heavily than incorrect non-merge decision. The later do not cause much harm because the connection is made through another path. In principle we could assign higher cost to incorrect merges, which is equivalent to changing the prior. In practice however, as we only use the ranking of the classifier, the actual prior used does not make a difference.
6.4 Experimental Results

6.4.1 Implementation Details

The naïve Bayes classifier was implemented as follows: we used the two features specified in eq. (6.2) and (6.3), and the histograms of each feature contains 100 equal intervals. The distribution for positive and negative examples was learned from about 40000 examples. Laplacian correction was applied.

The random forest uses all the features described in section 6.3.1. The classifier has 50 trees. At each node the number of features considered for determining the split is the square root of the total number of features used, selected randomly with uniform distribution. The criterion used for selecting the feature and corresponding threshold splitting a node was the information gain. The number of samples in the training set was about 10000.

6.4.2 Results

In this section we present the performance of the methods described throughout the chapter. To compare different algorithms we use the recall [MFM04]. This is a common measure used to quantify the ability of an oversegmentation method to produce segmentations in which the edges between segments attach to object borders, and is defined as the fraction of the ground truth boundary that lies close to the boundary of a superpixel found by the algorithm; see [MFM04, LSK+09], section 5.3.1 for implementation details. Figure 6.7 presents the results, note how the recall is better than the one obtained with pLV which was used to create the initial superpixels.
Creating an oversegmentation with 200 SPs in an image of size 321 × 481 takes 12.3 seconds when using two classifiers (lean and rich) and backtracking. If only the lean classifier is used and backtracking is not included in the process, the method takes 3.2 seconds to run. Thus, depending on the characteristics of the application, it is possible to select high accuracy and longer runtime, or reduced accuracy and high speed.

Figure 6.8 presents two examples of images segmented at different levels of detail. The segmentations presented were obtained using both the lean and rich classifiers, four stages (see section 6.2.5) and backtracking. The last two rows of the figure compare the segmentations obtained with lowest level of detail (ie. 100 SPs) to hand-made segmentations created by humans [MFTM01]. Note how the edges drawn by humans coincide consistently with edges found by our method (green edges). Being an oversegmentation method, our algorithm yields more edges than the ones appearing in human segmentations (blue edges), however very few hand-made edges are missed (red edges), thus yielding high recall.

Another measure of quality usually used for boundary detection algorithms is the precision [MFM04], it is defined as follows: let $\delta G$ be the set of ground truth boundary pixels, $\delta S$ be the set of pixels in segmentation $S$ found by the automatic algorithm, and $\mathbb{1}(.)$ be the indicator function, then the precision of segmentation $S$ is

$$P_G(S) = \frac{\sum_{p \in \delta G} \mathbb{1}\left( \min_{q \in \delta S} \| p - q \| < \epsilon \right)}{|\delta S|}. \quad (6.11)$$

Figure 6.9 presents the precision vs. recall curve for the methods in this chapter, together with other algorithms in the literature. As can be appreciated, algorithms that calculate a soft edge decision (i.e. sketch tokens [LZD13] ) are more successful than algorithms that calculate closed contours (i.e. Local Variation, pLV, Learning multistage). Additionally, note how the method presented in the chapter improves the basic pLV algorithm.

### 6.5 Conclusion

We have shown how single-linkage algorithms can be used for image oversegmentation. In particular, we have presented a single-linkage agglomerative algorithm that uses machine learning methods and correction techniques that include global information. We have shown that even though single-linkage algorithms are relatively simple, this approach can be used to improve the performance of existing oversegmentation algorithms such as probabilistic local variation.
Figure 6.8. First two rows: example of images segmented at different levels of detail using both the lean and rich classifiers, four stages and backtracking. From left to right, 2000 SPs, 600 SPs, 100 SPs. Third and fourth rows: comparison of the segmentations obtained with our method to hand-made segmentations before (third row) and after BT (fourth row). Blue edges represent edges found only by the algorithm, red edges represent edges drawn only by humans, green edges represent edges defined by both human and our algorithm. Noted how the edges drawn by humans are consistently recalled by the algorithm.
Figure 6.9. Precision vs. recall curve. Note how our method improves the basic pLV used to create initial superpixels. Additionally, note that methods that calculate a soft edge decision are more successful than algorithms that create closed contours.
Chapter 7

Discussion

In this work we have presented insights into the task of image oversegmentation. In particular, our work is based on the well known local variation algorithm [FH04]. We introduced theoretical justification to this algorithm and further developed it into new algorithms that yield state-of-the-art results.

Thus, the main contributions of this work are the following:

1. We have shown that the two components present in the LV adaptive threshold, segment size and segment distribution, are essential to obtain a good segmentation.

2. We provided theoretical insight to the LV algorithm by showing that its adaptive threshold has a statistical interpretation. Specifically, it can be understood as the estimation of the maximum value of the gradients in the MST of a segment, assuming that the gradients are distributed uniformly.

3. By understanding that the uniform distribution is not correct, we casted the segmentation task as a hypothesis testing problem. In this framework, the gradients are assumed to be distributed exponentially, according to statistics of natural images. The probability that an edge connecting two segments belongs to each of them is then calculated and used to accept or reject the merge of two neighboring segments throughout an iterative process. This approach resulted into the $O(n \log n)$ algorithm denoted pLV.

4. We introduced several modifications to the basic single-linkage agglomerative algorithm to yield an improvement in existing segmentation methods like LV or pLV. Specifically, we presented a method for learning similarities between segments, adopted a multistage process, and introduced an optimization method for correcting mistakes called backtracking.

These contributions are weaved together in this work by building one on top of the other.
Nevertheless, some of the contributions in this work can be relevant in other fields. The study of natural image statistics presented in section 5.2.1 presents the distribution of the gradients in the MST of the image and we show that it differs from the distribution of the gradients in the full image. This fact can be exploited for image compression. Additionally, we introduced a machine learning approach for defining similarity between segments. The use of the decision level of the classifier as a confidence indicator instead of a hard decision rule can be further developed in other visual tasks.

Further developments on topics addressed by this work would of course be beneficial for our segmentation method, but they could also prove beneficial for understanding image segmentation or maybe other vision related tasks. Next we discuss further explorations based on our developments.

In the pLV method we use the edges in the MST of each segment to create a representative statistic of each segment. Using the edges in the MST yields an efficient algorithm, however the statistic might not be as representative as desired. Studying different sampling methods is an interesting problem by itself and could yield improved segmentation results. Likewise, we used the assumption that the distribution of gradients in the MST follow an exponential rule. Although this assumption is supported empirically, as presented through our work, it might only be an approximation to the real distribution. Studying the statistics of natural images and developing more accurate models will affect not only our work but several other fields such as image compression, as stated before.

Regarding the machine learning approach to differentiate if neighboring superpixels belong to the same or different segments, further work is needed. We introduced several cues including relative color differences, absolute color differences, texture and gradients, however these are far from exhaustive. Similar to the work in \cite{AMFM11}, using cues that include directional information can improve the segmentation result. Unlike \cite{AMFM11}, in which many directions are tested at each pixel, in our method a single direction could be used which is uniquely defined by the direction of the edge between two superpixels.

The \textit{backtracking} method could also be further studied. Two straightforward extensions would be to develop a different cost function and an efficient calculation method to find the global maximum of the cost function. Finding fast convergence methods would minimize the need to compromise between runtime and accuracy for single-linkage like algorithms.

To conclude, we would like to notice that image segmentation is a basic building block towards the more demanding task of computer vision. It is an open subject in the research community and further developments in the field would greatly improve human-computer interaction.
Appendix A

Fast Approximation of $\chi^2_{p,\nu}$

The value $\chi^2_{p,\nu}$ is defined as follows: let $X$ be a random variable following a $\chi^2$ distribution with $\nu$ degrees of freedom. Then $\chi^2_{p,\nu}$ is the quantity that holds $P\{X \geq \chi^2_{p,\nu}\} = p$.

A fast approximation [Lin88] of the area under the upper tail of the $\chi^2$ distribution, $p$, and its inverse, $\chi^2_{p,\nu}$, is presented below.

$$ p \approx \begin{cases} 1 - \frac{1}{2} \exp \left( b_1 z + a_1 z^2 \right), & z \leq 0 \\ \frac{1}{2} \exp \left( b_2 z + a_2 z^2 \right), & z > 0 \end{cases} \quad (A.1) $$

where $z = \sqrt{\chi^2_{p,\nu}} - \sqrt{\nu}$.

Values for $a_1, b_1, a_2, b_2$ are found by fitting $b_1 z + a_1 z^2 = \ln(2(1-p))$ and $b_2 z + a_2 z^2 = \ln(2p)$ by the least squares method, obtaining

$$ a_1 = -0.9911, b_1 = -0.8055 \quad a_2 = -0.6763, b_2 = -1.2451. $$

The approximation for $\chi^2_{p,\nu}$ is obtained by solving equation (A.1) for $\chi^2_{p,\nu}$:

$$ \chi^2_{p,\nu} \approx (z + \sqrt{\nu})^2 \quad (A.2) $$

where

$$ z = \begin{cases} -\frac{b_1 + \sqrt{b_1^2 - 4a_1 c_1}}{2a_1}, & p \geq 0.5 \\ -\frac{b_2 - \sqrt{b_2^2 - 4a_2 c_2}}{2a_2}, & p > 0.5 \end{cases} \quad (A.3) $$

$$ c_1 = -\ln(2(1-p)), c_2 = -\ln(2p). \quad (A.4) $$
Figure A.1. The value $\chi^2_{p,\nu}$ as a function of its degrees of freedom.

The behavior of $\chi^2_{p,\nu}$ as a function of its degrees of freedom is presented in figure A.1.
Bibliography


undersegmentation error, which MBG achieved with a higher truth, while the algorithms achieved a lower segmented truth. The algorithm succeeded in segmenting and undersegmentation error was lower than the ground truth. The BSDS300 dataset was used for the experiment [MFTM01] (BSDS300). The motivation is the accuracy of the algorithm, which is measured on the basis of the number of segments and the number of pixels in the image. The undersegmentation error is calculated as a function of the number of segments in the segmentation. As the number of segments decreases, the undersegmentation error increases, and as the number of segments increases, the undersegmentation error decreases.

Reiko Shirono, Masahiro Matsuda, and Yukihiro Hasegawa. Undersegmentation error of the algorithm, which is lower than the state-of-the-art. The algorithm achieved a recall of 0.98 on 2000 segments, 0.945 on 1000 segments, and 0.9 on 500 segments. The recall for the BSDS300 dataset is greater than 0.95 for all algorithms. The SLIC algorithm achieved a recall of 0.98 on 2000 segments, 0.945 on 1000 segments, and 0.9 on 500 segments. The undersegmentation error of the algorithm is lower than the state-of-the-art.
local variation (LV) to measure local variation within a segment. The algorithm is based on a probabilistic model, where the decision is made based on statistical criteria. The task of this algorithm is to explain the decisions made by the algorithm as statistically accepted and used in theoretical assessments. In the first step, significant segments are identified, and the gradient is strengthened within these segments, under the assumption that the gradient increases.

The model is based on a single-linkage clustering method (pLV) and uses a probabilistic local variation (pLV) as a measure of similarity. The algorithm then uses an exponential model for the probability of a pixel belonging to a segment, which is consistent with the statistical model. In addition, the model uses a probabilistic approach for the estimation of the improvement in the segmentation.

The key to this approach is to use a single-linkage agglomeration algorithm that can merge segments based on the decision value. The decision value is calculated as the number of pixels within a segment, and the algorithm uses a supervised learning approach to improve the decision value.

The algorithm uses a multistage learning approach to iteratively improve the segmentation. In each stage, the algorithm uses a decision value to determine the optimal segmentation, and the segmentation is refined iteratively to improve the decision value. The final segmentation is used to evaluate the performance of the algorithm, and the algorithm is improved iteratively to achieve better results.

The evaluation is performed using the standard metrics of mIoU and precision, recall, and f1-score. The results show that the proposed algorithm achieves better performance compared to existing algorithms, making it a useful tool for image segmentation tasks.
**Resume**

Segmentation is a divided into segments process to divide an image into a number of segments called superpixels, each of which represents an object. The result of the image that has been segmented can be described as a bin image of the world, and thus segmentation is a process primarily used in the research of computer vision, with the highest level of segmentation being that which has been refined and improved.

In this thesis, researchers in the field of computer vision are making efforts to develop image segmentation algorithms. In particular, superpixel segmentation is more popular in recent years.

The reason for this popularity is the use of superpixels in computer vision tasks that are more complex than the traditional segmentation tasks.

Successful segmentation of an image (oversegmentation) is a process that divides an image into a number of segments called superpixels, each of which represents a region of an object. The result of the segmentation process is a bin image of the world, which is described as an image of objects in the world.

One of the most popular algorithms for image segmentation is the superpixel algorithm, which is based on the concept of superpixel. The superpixel algorithm is a method that divides an image into a number of segments called superpixels, each of which represents a region of an object. The result of this process is a bin image of the world, which is described as an image of objects in the world.

One of the most important parameters in superpixel segmentation is the local variation (LV) of the image. The LV is a measure of the variation in the image, which is calculated using the formula:

$LV = \frac{1}{n} \sum_{i=1}^{n} \left( \frac{X_i - \bar{X}}{\sigma} \right)^2$

Where $X_i$ is the intensity of the pixel, $\bar{X}$ is the mean intensity, and $\sigma$ is the standard deviation.

The superpixel algorithm is a method that divides an image into a number of segments called superpixels, each of which represents a region of an object. The result of this process is a bin image of the world, which is described as an image of objects in the world. The LV is an important parameter in superpixel segmentation, as it is used to determine the number of superpixels that are needed to represent the image.
המחקר בוצע בהנחיית של פרופסור מיכאל לינדנבוים, בפקולטה למדעי המחשב.

תודה

ברצוני להודות לפרסומ' מיכאל לינדנבוים, היועץ لمкер זה. הדרכהו וفكוחו והחוניים לשלמה, אשר עזרו לי לאברה את ההדמונת הלומד ממוון על נושא לי אימוני מחשבה וגו.

אולים אפייל יותר שאבי, על מחקר מדויק והשימור ביברוית.

אני גות התודה למשתתף של על עידוד לאורח כל הדרכ היה. אמא ואמא, רחל ורחל,

בוחד אם אתי אירק לי מקורי או סופי לתמיכתי בכל רגע.

הכרת תודה מסורה לטעינו עלモノ המחקר זה.
סגמנטציה בעזרת הערכה סטטיסטית של
שינוי מוקומי

תורם על מחקר

לשם מלי הלך של הדרישות לעבורה
מניסים菩提ימ בדועים המתחבש

מיכאל בלייקסה

הנה לולשיט היסנין - מכון טכנולוגי לישראל
חושן תש"ע'ה חיפה אוקטובר 2013
סגמנטציה בעזרת הערכה סטטיסטית של שיווי מיקומי

מיכאל בלאנקה