# Probabilistic Image Segmentation

Michael Kihm*, Hafiz Hamza**, Simone Schopperth**
*Università di Bologna, **Berlin University of Technology
michael.kihm@studio.unibo.it, hamza@campus.tu-berlin.de,simone.schopperth@gmx.de

## Abstract

*In this paper we propose an image segmentation method based on a probabilistic model and the use of different image features. Our region growing algorithm combines several cues to subdivide the image into meaningful parts. The different cues are combined and weighted in a probabilistic model based on a fast kernel density estimation. The probabilistic model uses the naïve Bayes theorem to combine the cues. It is therefore very flexible and allows an unrestricted selection of preferred image features one to freely choose the preferred image features to adapt the algorithm to individual needs. An evaluation on the Berkeley Segmentation Dataset (BSD) shows the consistency of our approach.*

***Keywords:*** *Image segmentation, region growing, probabilistic model, naïve Bayes, fast kernel density estimation*

## 1. Introduction

Semantic image segmentation is an important part of image processing. To date, different approaches for image segmentation have been published. A good overview is given in [1]. However, many of the approaches focus on a single image feature like texture, color or edges to separate the image. We propose a region growing algorithm which combines several cues to grow the regions to a segmented image.
Our region growing algorithm starts with a set of SLIC[1] superpixels [4] and grows the regions until every image pixel is assigned to just one segment. Our approach uses several cues, combined in a probabilistic manner, to decide whether regions should be merged or not. The calculation of the features of a region are based on basic feature descriptors to allow a fast computation time. The probability model assumes independence of the features and therefore multiplies their likelihoods based on the naïve Bayes theorem. The likelihoods of each feature get calculated by a fast and approximated Gaussian kernel density estimator based on the approach from Raykar *et al.* [2].
In this paper we will discuss the work related to the project (chapter 2) and explain the calculation of the chosen image features (chapter 3). Furthermore, we will describe the region growing approach and the training of the algorithm in chapters 4 and 5. The probabilistic model will be explained in chapter 6. The results of the segmentations are shown in chapter 7. Chapter 8 summarizes the work.

[1] Simple Linear Iterative Clustering

## 2. Related Works

This projects mainly builds on the segmentation algorithm from Stanski *et al.* [3] and the fast kernel estimation from Raykar *et. al.* [2]. Stanski *et al.* developed a region growing algorithm and combined three different features to calculate the merging probability of regions. The approach from Stanski *et al.* starts from single pixels, where each pixel is considered a single region. The algorithm calculates the merging probability from arbitrarily chosen pairs of regions and the probabilstic model then decides whether regions should be merged or not. The calculation of the merging probability is carried out in two steps. Beginning with an estimation of the posterior $\hat{P}(merge|x_{1...n})$ from the features and second the calculation of $\hat{P}(reliablex|size)$ increases the stability of the probability estimation. His method was evaluated on the Berkeley Segmentation Dataset 300 [6] and achieved a remarkable score of 0,65 for grayscale images. In this project the constructive probabilistic learning CEPEL model of Stanski *et al.* was not used. Furthermore, a slightly different region growing approach was performed.
Our probabilistic model relies on the fast kernel estimation from Raykar, *et al*, which will be described in detail in chapter 6.

## 3. Feature Calculation

We use simple and basic feature descriptors to maintain a fast computation time.

### 3.1. Texture

The texture of one region is described by its standard deviation (stddev) and is calculated in the grayscale colorspace. The stddev is calculated wtih the following equation

$$\sigma = \sum_{i=1}^{L} \sqrt{((i-\mu)^2 \cdot p(i))} \quad (1)$$

where $i....L$ are the gray-levels and $p(i)$ is the probability of gray-level i. The probability is calculated using the normalized histogram of the region. Furthermore, entropy was implemented as an additional texture descriptor which, however, did not lead to a better segmentation result.

### 3.2. Color

The feature color is computed in the L*a*b colorspace. The L*a*b* colorspace has the advantage of easily approximating the difference of two colors by assuming that each color represents a point in a three-dimensional space [10]. The color difference $\Delta C$ is therefore obtained from the euclidean distance of the mean values from the L, a and b channels of two regions.

$$\Delta C = \sqrt{(\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2} \quad (2)$$

### 3.3. Arrangement

The arrangement of two regions relative to each other is a percentage value, depending on the number of pixels on the region borders. The calculation is done by dividing the number of border pixels (not touching the other region) over the number of all border pixels of the two regions.

## 4. Region Growing

Our segmentation approach is based on the region growing principle described in [1]. Region growing is an iteratively segmentation method which starts from single pixels or regions to grow segments. The algorithm stops as soon as every image pixel is assigned to one segment. The starting point of our region growing algorithm is a set of 1500 superpixels. Superpixels are a combined set of pixels, calculated in an iterative clustering, which provide a coarse segmentation of the image. We used the implemented SLIC code from Achanta *et al.* [4] to obtain the superpixels.
The following steps summarize our region growing algorithm:

1. Run SLIC to calculate superpixels.
2. Find adjacent regions.
3. Read and store training data (results that were found and stored during training).
4. Iterate through all regions and calculate for the adjacent regions:
   - The difference in texture and color
   - The arrangement to each other
5. If merging probability is higher than a defined threshold, merge the two regions, otherwise not.
6. Repeat 4-5 until all merging probabilities are less than the defined threshold.
7. Save the results as segmented image and as boundary map

We also tried to implement the approach from [3]. Their approach is to treat every single pixel as a region in the beginning and calculate their merging probabilities. The two regions with the highest merging probabilities will be merged. Afterwards, the merging probabilities of the adjacent regions of the just created new region have to be calculated and the regions with the highest probabilities will be merged. Our implementation idea was to store the merging probabilities for each two regions $A$ and $B$ and both regions in a maximum heap with respect to their merging probabilities, i.e. the root of the heap stores the two regions with the maximum merging probability and the probability itself. This implementation ensures fast access to the two regions that have the highest merging probability, hence, it ensures fast merging. However, it does not ensure fast re-computation of the merging probabilities after the most probable regions were merged. Therefore, this approach was not competitive and we chose to pursue the presented approach.

### 4.1. Details about finding adjacent regions

After running SLIC, we get some super pixels, say N, and we label each of the super pixel in the range 0-N. A trivial approach for finding adjacent regions is that we compare every region with every other region but this would take a lot of computational time. So after observing the performance of the *FindAdjacentRegions()* function, we devised an equation to limit this range (i.e. 0-N). Now, for a region labelled X, instead of checking all 0-N regions, we just check regions labelled X-Y to X+Y where Y is calculated from our devised equation

## 5. Training

The training data set was taken from the Berkeley Segmentation Dataset [6]. We downloaded the training images and their respective ground truth, containing information about the partial position of segment borders. There are various data files for a single image as many users have submitted the segmentation for a single image. Each of this data

file, contains, for instance, some information about the image like height, width, etc. and the complete information about the segments.
A simple pseudo-code for training using a single image is stated below:

1. Load the training data from the file and store it.
2. Run SLIC and get superpixels.
3. Find adjacent regions.
4. For every pair of adjacent regions, check whether they belong to the same segment.
5. If yes, merge the two regions and store YES (true) else store NO (false), along with all other parameters.
6. Repeat 4-5 until no more regions get merged.
7. Save the training statistic in a text file.

## 6. Probabilistic model

The probability model decides on the basis of the feature vector x and the training data if two regions should be merged or not. This decision is mathematically described by the calculation of the posterior probability $\hat{P}(merge|x)$. From the law of conditional probabilities the Bayes theorem can then be derived [5]. The Bayes theorem equation for our segmentation decision is given by:

$$\hat{P}(merge|x_{1..n}) = \frac{\hat{p}(x_{1....n}|merge)\hat{P}(merge)}{\hat{p}(x_{1....n}|merge)\hat{P}(merge)+\hat{p}(x_{1....n}|\overline{merge})\cdot\hat{P}(\overline{merge})} \tag{3}$$

where $\hat{P}(merge)$, $\hat{P}(\overline{merge})$ are the prior probabilities of the possible decisions merged or not merged and $\hat{p}(x_{1....n}|merge)$ , $\hat{p}(x_{1....n}|\overline{merge})$ are the probability densities of the likelihood. The calculated features are assumed to be independent from each other, validating the applicability of the naive Bayes and allowing the multiplication of the likelihoods.

$$\begin{aligned}\hat{p}(x_{1...n}|merge) &= \prod_{i=1}^{n}\hat{p}(x_i|merge)\\ \hat{p}(x_{1...n}|\overline{merge}) &= \prod_{i=1}^{n}\hat{p}(x_i|\overline{merge})\end{aligned} \tag{4}$$

The prior probabilities $\hat{P}(merge)$, $\hat{P}(\overline{merge})$ are calculated once in an initialization phase of the program. The prior $\hat{P}(merge)$ is obtained from the trainings data vector by dividing the number of merged events over the total size of the trainings data vector:

$$\hat{P}(merge) = \frac{\sum\hat{P}(merge)}{dataVector.size()} \tag{5}$$

$\hat{P}(\overline{merge})$ can then easily be calculated by $1-\hat{P}(merge)$. The likelihood needs to be calculated for each feature at every iteration step. To obtain a reliable result it is necessary to find an appropriate approach to calculate the likelihoods. A nonparametric estimation was chosen since no information on the probability distribution of the data was available. Furthermore a nonparametric estimation offers the possibility to freely add other feature descriptors without knowing any shape of the probability density function. There are many approaches for a nonparametric probability estimation, as summarized in [5]. The easiest approach is the probability estimation with histograms. The advantage of this approach is that the training data can be discarded after the probability estimation. This has especially advantages for a large set of trainings data, which is required for a meaningful nonparametric probability estimation. The drawback of the histogram is the huge discretization of the data which can lead to inexact estimations. Our segmentation algorithm was first implemented with a probability estimation based on histograms. Since the resulting segmentation was not very exact, this approach was discarded and the likelihood estimation is now based on a *kernel density estimator* (KDE) with a Gaussian function as kernel function.

$$\hat{p}(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\sqrt{2\pi}\cdot h^2}e^{\frac{|x-x_e|^2}{h^2}} \tag{6}$$

where N is the number of data and h the bandwidth of the kernel. The PDF[2] estimated by a KDE results to be more continuous as a PDF from a histogram approach. However a KDE estimation needs to take each training sample into account to estimate the probability. The computational cost for M evaluation points and N sample points is O(NM), which causes a high computational complexity when using large datasets. In our case six (M=6) probability densities need to be calculated for each region growing iteration step. The training of one image results in around 4800 to 5200 sample points. This high amount of data and the large number of probability densities caused a computation time of around 30 minutes by using the training data from around 30 images. Therefore it was necessary to find a faster, but still precise KDE technique. A suitable KDE was found by the Fast Univariate Density Derivative Estimator from Raykar *et al.* [2], which is a fast algorithm to calculate an $\epsilon$ exact approximation of the r-th derivative of a Gaussian kernel. The Gaussian can be easily r-times differentiated by:

$$\hat{p}(x_e)^{(r)} = \frac{(-1)^r}{\sqrt{2*\pi}n\cdot h^{r+1}}\sum_{i=1}^{N}H_r(\frac{(x_e-x_i)^2}{h})e^{\frac{-(x_e-x_i)^2}{2h^2}} \tag{7}$$

[2]probability density function

The term $H_r$ represents the r-th Hermite polynomial. Hermite polynomials are a set of orthogonal polynomials in the range of $(-\infty, +\infty)$ [8]. For a normal KDE only the computation of the derivative $r = 0$ required. The in [2] proposed kernel estimator separates $x_e$ and $x_i$ of the e-function of equation 7 via Taylor series and calculates only the first terms of the series. With the freely choosable error $\epsilon$ one can decide how many series terms get included into the calculation.

$$e^{-\frac{(x_e-x_i)^2}{h_2^2}} = \sum_{k=0}^{p-1} \frac{2^k}{k!} \left( e^{\frac{-|x_i-x_*|^2}{h_2^2}} \left( \frac{x_i - x_*}{h_2} \right)^k \right) * \left( e^{\frac{-|x_e-x_*|^2}{h_2^2}} \left( \frac{x_e - x_*}{h_2} \right)^k \right) + error_p \tag{8}$$

The Hermite polynomial $H_r$ from equation 7 can be factorized by the binomial theorem:

$$H_r\left(\tfrac{x_e-x_i}{h_1}\right) = \textstyle\sum_{s=0}^{\lfloor r/2 \rfloor} \sum_{t=0}^{r-2s} a_{st} \left(\tfrac{x_i-x_*}{h_1}\right)^t \left(\tfrac{x_e-x_*}{h_1}\right)^{r-2s-t}$$

$$a_{st} = \frac{(-1)^{s+t} \cdot r!}{2^s s! t! (r-2s-t)!} \tag{9}$$

Combining equations 8 and 9 then yield:

$$\hat{p}(x_e)^{(r)} = \textstyle\sum_{k=0}^{p-1} \sum_{s=0}^{\lfloor r/2 \rfloor} \sum_{t=0}^{r-2s} a_{st} B_{kt} e^{\frac{-|x_e-x_*|^2}{h_2^2}} \left(\frac{x_e-x_*}{h_2}\right)^k \left(\frac{x_e-x_*}{h_1}\right)^{r-2s-t}$$

$$B_{kt} = \tfrac{2^k}{k!} \textstyle\sum_{i=1}^{N} q_i e^{\frac{|x_i-x_*|^2}{h_2^2}} \left(\frac{x_i-x_*}{h_2}\right)^k \left(\frac{x_i-x_*}{h_1}\right)^t \quad k = 0...p-1, t = 0....r$$

$$q_i = (-1)^r / \sqrt{2\pi} n h^{r+1} \quad n\text{: size of training data} \tag{10}$$

The Taylor series is expanded about the point $x_*$ and gives only a good approximation in a small interval around $x_*$. Therefore, the data space gets separated into L equal intervals of a certain length $r_x$ and $x_*$ is substituted by the cluster centers $c_l$. This results in:

$$\hat{p}(x_e)^{(r)} = \textstyle\sum_{l=1}^{L} \sum_{k=0}^{p-1} \sum_{s=0}^{\lfloor r/2 \rfloor} \sum_{t=0}^{r-2s} a_{st} B_{kt}^l e^{\frac{-|x_e-c_l|^2}{h_2^2}} \left(\frac{x_e-c_l}{h_2}\right)^k \left(\frac{x_e-c_l}{h_1}\right)^{r-2s-t}$$

$$B_{kt}^l = \tfrac{2^k}{k!} \textstyle\sum_{x_i \epsilon S_l} q_i e^{\frac{-|x_i-x_l|^2}{h_2^2}} \left(\frac{x_i-c_l}{h_2}\right)^k \left(\frac{x_i-c_l}{h_1}\right)^t \quad k = 0...p-1, t = 0....r \tag{11}$$

The $B$ coefficients are calculated for each bin and do not depend on the evaluation point $x_e$. Therefore they can be calculated once in an initialization step. Also the $a$ terms of equation 9 can be calculated once. The computation time can even be more reduced by calculating only the bins inside a certain cutoff radius $r_{ex}$. This cutoff radius also depends on the desired error $\epsilon$. Considering $r_{ex}$ and substituting $h_1 = h$ and $h_2 = \sqrt{2}h$ leads to the final equation:

$$\hat{p}(x_e)^{(r)} = \sum_{\forall\, s.t\ |x_e-c_l| \le r_{ex}} \sum_{k=0}^{p-1} \sum_{s=0}^{\lfloor r/2 \rfloor} \sum_{t=0}^{r-2s} a_{st} B_{kt}^l e^{\frac{-|x_e-c_l|^2}{2h^2}} \cdot \left(\frac{x_e - c_l}{h_1}\right)^{k+r-2s-t} \tag{12}$$

A more detailed explanation is given in [2]. The above described KDE brought an impressive performance enhancement which is shown in figure 1. The plot shows the computation time comparison of the normal KDE using a Gaussian kernel, the Gaussian KDE implemented with openMP and the Fast Gaussian Kernel approximation. The plotted computation time represents the average computation time of three test images. The algorithm was implemented and benchmarked with a training data set of 15 images and an error $\epsilon$ of $4.4 \cdot 10^{-6}$. The adequate bandwidth h was estimated by a Matlab script.

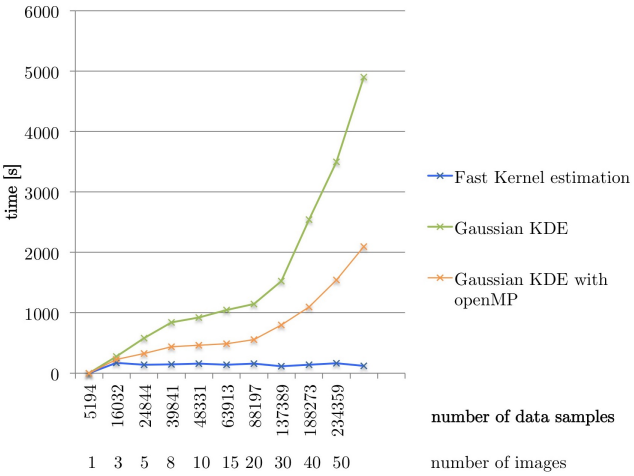


Figure 1. Plot of the computation time of the region growing algorithm for different sizes of training samples. It can be seen that the computation time drastically increases with a rising size of training data, when using the normal Gaussian kernel (green line). The performance of the Gaussian KDE can be improved with the openMP library, which subdivides the probability estimation in parallel executable threads (orange line). The computation time still increases with enlarging training data. However, the computation time of the fast Kernel estimator ($\epsilon = 4.4 \cdot 10^{-6}$) stays, also with a large training set, almost constant (blue line).

## 7. Evaluation

Our segmentation algorithm was evaluated on the Berkeley Segmentation Dataset BSD500 [7], which evaluates an algorithm by comparing the estimated segments to human made segmentations. This comparison is done in terms of precision R, recall R and F-score calculations. Precision is the noise influence of the segmentation algorithm and is mathematically described by the probability that an estimated segmentation decision is valid. Recall states the probability whether the ground truth was correctly detected or not. The F-score provides the harmonic mean of recall and precision [9].

$$F_{score} = \frac{2 \cdot P \cdot R}{P + R} \tag{13}$$

| Threshold | F-Score |
|---|---|
| 0.5 | ODS: F(0.86, 0.34) = 0.48<br>OIS: F(0.86, 0.34) = 0.48 |
| 0.4 | ODS: F(0.71, 0.41) = 0.52<br>OIS: F(0.71, 0.41) = 0.52 |

Table 1. Segmentation quality of our algorithm, benchmarked on the validation images of the BSD500. A good algorithm has a F-score value.

The algorithm was benchmarked on the validation dataset of BSD500 which contains 100 images of natural scenes. The method was benchmarked with merging thresholds 0.4 and 0.5. The obtained scores are shown in table 1.

It can be seen that our approach didn't reach the top 10 of the BSD, which starts with a F-score of 0.57. The score of a human segmentation is 0.79 and the best benchmarked segmentation algorithm from Ren *et. al* [13] reached a score form 0.71. By observing the segmentations from figure 2, one can see that the results especially for complex scenes are quite poor. The algorithm assigns segments not only to the object of interest, but also to single image details like little stones (figure 2 (6)) or hair (figure 2 (3)) and oversegments the image.

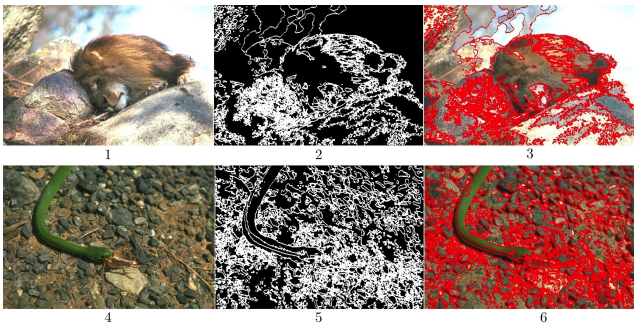


Figure 2. Segmentation of our segmentation approach for complex images. The first column shows the input images. Columns 2 and 3 are the resulting boundary maps and segmentation images.

A meaningful segmentation is only obtained in relatively simple images without complex textures. Figure 3 shows some example output images of simple scenes.

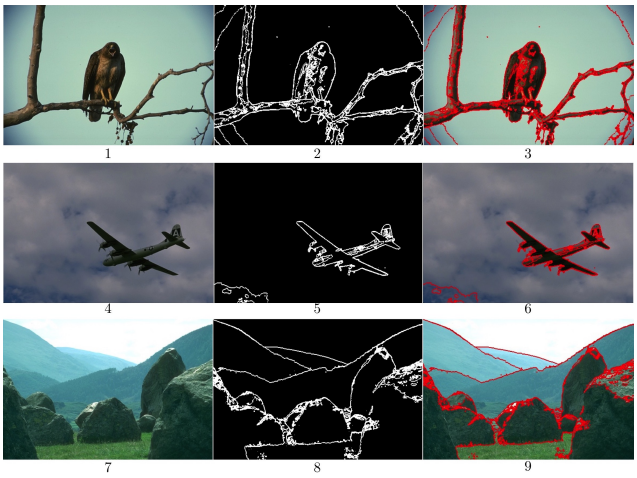


Figure 3. Segmentation of our segmentation approach for simple natural scenes. The first column shows the input images. Columns 2 and 3 show the resulting boundary maps and segmentation images.

## 8. Conclusion

Although our segmentation algorithm didn't achieve a high score on the BSD, it is still a useful and flexible tool with high potential to improve its score on the BSD. An enhancement could be achieved by substituting our very basic feature descriptors with more complex and exact descriptors. A possibility would be the use of Textons [11] or covariance matrices [12] as feature descriptors. Furthermore, the region growing approach of Stanski *et al.* could be applied to improve the image segmentation. This algorithm calculates the merging probability $\hat{P}(merge|x)$ from randomly chosen region pairs. This region growing method has the advantage of merging regions with a high merging probability first. Difficult merging decisions are performed only after more information is available, which eventually leads to an increased reliability of the merging probability $\hat{P}(merge|x)$ for complex regions. Our region growing approach estimates the merging probability of the adjacent regions from each region or superpixel and merges every region which has a higher merging probability then a defined threshold. This allows a fast computation but merging decisions for complex regions have to be performed already in early iteration steps of the region growing. This could be the reason why we obtain an oversegmentation for complex images. An idea could be to implement our working approach with the heap method. For each superpixel, we could have a heap that stores the regions and their merging probabilities that can be merged with the superpixel. The root of each heap would store the region that has the highest merging probability with the superpixel. Beside the region growing also the likelihood estimation could be designed differently to allow an increased flexibility to changes in the training

data. At the moment, a separate Matlab script needs to be run to calculate the correct bandwidth of the KDE kernel. It would be more convenient, if the algorithm estimated the correct bandwidth at an initialization step of the program.

## References


[1] R. Gonzales, R. Woods. *Digital Image Processing*. Addison Wesley, 2008.

[2] Raykar, V. C. and Duraiswami, R. and Zhao, L. H.*Fast Computation of Kernel Estimators*.Journal of Computational and Graphical Statistics, volume 19, number 1, pp. 205-220. 2010

[3] A. Stanski, O. Hellwich. *A Projection and Density Estimation Method for Knowledge Discovery*. Plos ONE. 2012.

[4] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Ssstrunk. *SLIC Superpixels Compared to State-of-the-Art Superpixel Methods*. in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2274-2282. Nov. 2012.

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Buisiness Media LLC. 2006.

[6] D. R. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. ICCV, 2001.

[7] P. Arbelaez, M. MaireContour. *Detection and Hierarchical Image Segmentation*. IEEE TPAMI, vol. 33, no 5, pp. 898-916. May 011

[8] J. J. Sakurai. Modern Quantum Mechanics. Addison-Weslex Publishing Company, 1994

[9] D. R. Martin. *Learning to Detect Natural Image Boundaries Using Local Brightness, Color, and Texture Cues*. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 1, January 2004

[10] *Lab color space*. In Wikipedia. Retrieved August 20, 2017, from https://en.wikipedia.org/wiki/Lab_color_space.

[11] S. Zhu, C. Guo, Y. Wand and Z. Xu. *What are Textons?*. International Journal of Computer Vision 62(1/2), 121143, 2005

[12] M. Donoser, H. Bischof. *Using covariance matrices for unsupervised texture segmentation*. 2008 19th International Conference on Pattern Recognition, Tampa, FL. 2008. pp. 1-4. doi: 10.1109/ICPR.2008.4761350.

[13] Xiaofeng Ren, Liefeng Bo. *Discriminatively Trained Sparse Code Gradients for Contour Detection*. NIPS 2012.