Superpixel Benchmark and Comparison

Peer Neubert and Peter Protzel

Chemnitz University of Technology
Department of Electrical Engineering and Information Technology
{*peer.neubert, protzel*}*@etit-tu-chemnitz.de*

Abstract Superpixel segmentation showed to be a useful preprocessing step in many computer vision applications. This led to a variety of algorithms to compute superpixel segmentations, each with individual strengths and weaknesses. We discuss the need for a standardized evaluation scheme of such algorithms and propose a benchmark including data sets, error metrics, usage guidelines and an open source implementation of the benchmark in a Matlab toolbox. The benchmark evaluates the quality of the superpixel segmentation with respect to human ground truth segmentation and the segmentation robustness to affine image transformations, which is crucial for application on image sequences. To our knowledge, this is the first benchmark considering the segmentation robustness to such image transformations. To better consider the characteristics of an image oversegmentation, we provide a new formulation of the undersegmentation error. Using this benchmark, we evaluate eight algorithms with available open source implementations and discuss the results with respect to requirements of further applications and runtime.

1 Introduction

Image pixels are the base unit in most image processing tasks. However, they are a consequence of the discrete representation of images and not natural entities. Superpixels are the result of perceptual grouping of pixels, or seen the other way around, the results of an image oversegmentation. Superpixels carry more information than pixels and align better with image edges than rectangular image patches (see fig. 1.7). Superpixel can cause substantial speed-up of subsequent processing since the number of superpixels of an image varies from 25 to 2500, in contrast to hundreds of thousands of pixels. A

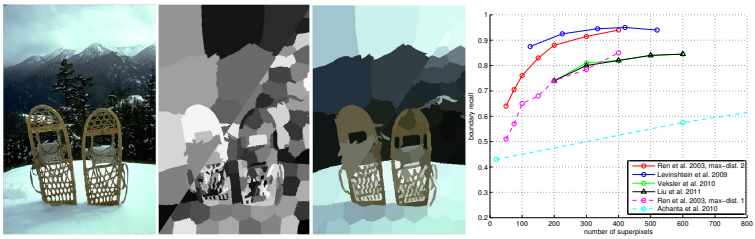


Figure 1.1: *(left)* Superpixels examples. The input image (left part) is segmented into 25 (middle part, top-left corner) and 250 (middle part, bottom-right) superixels and shown with mean segment colors (right part). *(right)* The varying results on the established normalized cuts superpixel segmentation algorithm using the same implementation, data set and error metric, demonstrate the need for a standardized benchmark.

superpixel segmentation of an input image is illustrated in fig. 1.1. Superpixels are becoming increasingly popular in many computer vision applications. Their benefit is analyzed in [1] and shown in applications like object recognition [2], segmentation [3] and automatic photo pop-up [4]. Downsides of using superpixel segmentation as preprocessing step are the computational effort for the computation of superpixels and more importantly the risk of loosing meaningful image edges by placing them inside a superpixel. Depending on the application and the used superpixel algorithm, subsequent processing steps can struggle with a non lattice arrangement of the superpixels. Therefore, the careful choice of the superpixel algorithm and its parameters for the particular application are crucial. In this paper, we provide a benchmark framework for superpixel segmentation algorithms and compare various existing algorithms. We focus on algorithms with an open source implementation, that are ready for application on a variety of computer vision tasks. We consider both, the quality of the segmentation compared to human ground truth segmentations and the robustness to affine image transformations. The following section gives an overview of the state of the art in superpixel segmentation and figures out, why there is still a need for a comparison of the quality of these algorithms. Section 3 describes our evaluation scheme, followed by an overview of the compared algorithms in section 4. Comparison results are given in section 5.

2 Related Work and Why We Need a New Benchmark

There has been a lot of research on superpixels since the term has been established in [5]. In particular, various algorithms for superpixel segmentations have been proposed, e.g. [5] [6] [7] [8] [9] [10] [11] [12] (these algorithms are compared in this work). Most of the existing work includes comparisons to

a couple of established algorithms, supported by publicly available implementations. Due to its broad publicity and its free implementation, superpixel segmentation based on normalized cuts [5] is one of the commonly used algorithms for comparison. Together with the Berkeley Segmentation Data Set [13] and the two error metrics "boundary recall" and "undersegmentation error", this is a repetitive comparison framework in the literature. However, there are small variations in the usage of the dataset or implementation of the error metric that cause major differences in the benchmark results. This is illustrated in fig. 1.1 *(right)*. Each curve comes from a paper where the authors report to use the free implementation of the normalized cuts superpixel algorithm, together with the Berkeley Segmentation Data Set and the boundary recall error metric. Nevertheless, the results are apparently different. A closer look reveals small differences in the usage of the dataset (e.g. Is the full dataset used or just a part? How is dealt with multiple ground truth?) and the implementations of the error metrics (e.g. How are boundary recall and undersegmentation error implemented? What is the threshold on boundary recall?) Furthermore, to apply superpixel segmentation on image sequences or video, the stability of the segmentation under transformations of the image content is important. To our knowledge, there exists no benchmark considering such transformations.

3 The Proposed Superpixel Benchmark

3.1 Error Metrics

For evaluation of segmentation quality, we focus on two error metrics: boundary recall and undersegmentation error. Boundary recall is an established measurement to evaluate segmentation algorithms. However, using boundary recall alone, favors segments with long boundaries. E.g. think of a segment with an anfractuous boundary that assigns each image pixel as boundary, this segment would achieve perfect boundary recall. Thus, in figure-ground segmentation, boundary recall is used together with boundary precision to take care of the length of boundaries. However, since for superpixel segmentation a low precision is inherent, undersegmentation error is more appropriate. Because there exist various implementations of these error metrics, we describe and discuss our choices in the following.

Boundary Recall

is the fraction of ground truth edges that fall within a certain distance d of a least one superpixel boundary. We use $d = 2$. Given a ground truth boundary image G and the algorithms boundary image B, the computation of boundary recall is straight forward:

1. **True Positives (TP)** Number of boundary pixels in G for whose exist a boundary pixel in B in range d.

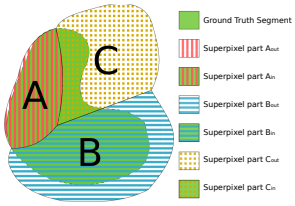


Figure 1.2: Illustration of undersegmentation error. A ground truth segment (green) is covered by three superpixels (A,B,C), that can flood over the ground truth segment border.

2. **False Negatives (FN)** Number of boundary pixels in G for whose does not exist a boundary pixel in B in range d.
3. **Boundary Recall** $R = \frac{TP}{TP+FN}$

Multiple ground truth boundary images are combined using OR operation. Thus, the resulting boundary map answers for each pixel the question: Is there a ground truth segmentation with a boundary at this pixel?

Undersegmentation Error

compares segment areas to measure to what extend superpixels flood over the ground truth segment borders. A ground truth segment divides a superpixel P in an *in* and an *out* part. This is illustrated in fig. 1.2. There exist various implementations of undersegmentation error metrics. In [11], for each segment S it is summed over the out-parts of all superpixels P that overlap the segment:

$$UndersegmentationError_{TP} = \sum_{S \in GT} \frac{\sum_{P:P \cap S \neq \emptyset} |P_{out}|}{|S|} \tag{1.1}$$

For the example of fig. 1.2, this is: $\frac{|A_{out}|+|B_{out}|+|C_{out}|}{|S|}$ However, there is a serious penalty for large superpixels that have only a small overlap with the ground truth segment. Thus, [12] uses a similar model, but only superpixels with an overlap with the segment of at least 5% of the superpixel size are regarded. To overcome this free parameter, we propose a new formulation of the oversegmentation error and define the remaining error as the smaller error introduced by either appending the *out*-part to the segment or by omitting the *in*-part of the superpixel. Being N the total number of pixels, this results in:

$$UndersegmentationError = \frac{1}{N} \left[\sum_{S \in GT} \left(\sum_{P:P \cap S \neq \emptyset} min(P_{in}, P_{out}) \right) \right] \tag{1.2}$$

The inner sum is the error introduced by this specific combination of ground truth segment and superpixel. For the example of fig. 1.2, this is: $\frac{|A_{out}|+|B_{out}|+|C_{in}|}{|S|}$

Precision-Recall on Boundary Images

For evaluation of robustness to affine image transformations, we compare boundary images of superpixel segmentations directly. The concrete usage is described in section 3.2. The evaluation metric is precision-recall represented by the F1-score. Computation of TP, FN and Recall follows computation of boundary recall, FP and Precision are computed as follows (d is set to 2 again):

1. **False Positives (FP)** Number of boundary pixels in B for whose does not exist a boundary pixel in G in range d
2. **Boundary Precision** $P = \frac{TP}{TP+FP}$

3.2 Benchmarking Robustness to Affine Transformations

For application of superpixel segmentation on image sequences or video, the stability of the segmentation under image changes is crucial. To evaluate the robustness of an algorithm A against an affine image transformation T, we compute two boundary images using this algorithm and compare. The first boundary image $A(I)$ is computed on the origin image directly. For the second boundary image, the image is first transformed by T to $T(I)$ using inverse mapping, followed by the computation of the boundary image $A(T(I))$. Back-transformation yields $T^{-1}(A(T(I)))$ which is compared to the boundary map $A(I)$ obtained from the segmentation of the origin image. The transformation scheme is illustrated in figure 1.3 together with some example transformations. The comparison of the boundary maps follows section 3.1 and yields an F-score for each transformation. Due to their non rectangular shape, a black border can appear at some transformed images $T(I)$. To minimize their effects on the segmentation, all origin images are padded with a black border at the very beginning.

3.3 Image Data, Usage Guidelines and Matlab Toolbox

The implementations of the error metrics are part of an open source Matlab toolbox that is available from our website[1]. The toolbox also provides functions for automatic benchmarking and evaluation of new superpixel algorithms, including easy parameter evaluation. The image data base is BSDS500 [14], the extended version of the established Berkeley Segmentation Data Set. To ensure comparable results, the usage of the partitions of the BSDS 500 dataset is intended as follows:

[1] http://www.tu-chemnitz.de/etit/proaut/forschung/superpixel.html

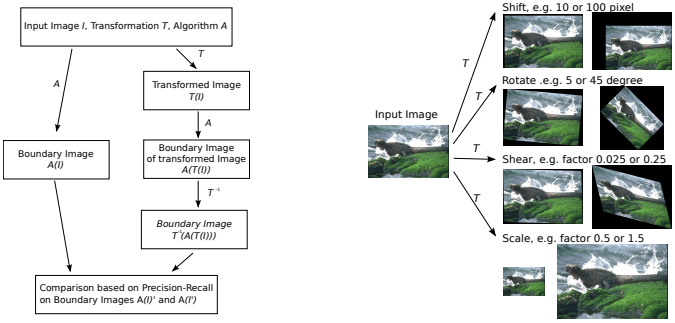


Figure 1.3: Image transformations *(left)* To evaluate the robustness to an image transformation T, we compute two boundary images, $A(I)$ and $T^{-1}(A(T(I)))$, and compare. For more details, see text. *(right)* Illustration of example transformations.

1. **Train** There are 200 images for learning purposes.
2. **Val** The cross-validation dataset contains 100 images for adjusting the parameters of an algorithm.
3. **Test** The 200 test set images are only meant for the final benchmark.

4 Algorithms

We compare various existing superpixel segmentation algorithms. Requirements for an algorithm to be presented here is an available open source implementation. The results are going to be presented on our web site, and thought to be extended with the results of newly available algorithms. Algorithms tested are Normalized Cuts (NC) [5][2], Felzenszwalb-Huttenlocher Segmentation (FH) [6][3], Edge Augmented Mean Shift (EAMS) [7] [15][4], Quickshift (QS) [9][5], Marker-Controlled Watershed Segmentation (WS) [10][6], Entropy Rate Superpixel Segmentation (ERS) [8][7], Turbopixel Segmentation (TP) [11][8] and two implementations of Simple Linear Iterative Clustering [12] (oriSLIC[9] and vlSLIC[10]). The oriSLIC implementation does not strictly follow the de-

[2] http://www.timotheecour.com/software/ncut/ncut.html
[3] http://www.cs.brown.edu/~pff/segment/
[4] http://www.wisdom.weizmann.ac.il/~bagon/matlab.html
[5] http://www.vlfeat.org/
[6] We use the OpenCV implementation with uniformly distributed markers http://opencv.willowgarage.com/wiki/
[7] http://www.umiacs.umd.edu/~mingyliu/research.html#ers
[8] http://www.cs.toronto.edu/~babalex/turbopixels_supplementary.tar.gz
[9] http://ivrg.epfl.ch/supplementary_material/RK_SLICSuperpixels/index.html
[10] http://www.vlfeat.org/

scription in [12] but incorporates some simplifications for speedup. The baseline comparison algorithm (BOX) simply divides the image to a regular grid with the given number of segments. Resulting segmentations are shown in figure 1.7. Due to its runtime, the normalized cuts algorithm is often applied on resized images. Thus we include a "NC resized" algorithm in the benchmark, where the images are scaled to 160 pixel on the longer side before segmentation. We further want to point out, that we show results of Turbopixel segmentation, although the performance for small numbers of superpixels is much worse than in the origin paper [11]. Turbopixel segmentation is based on growing of initial small segments. For small numbers (e.g. 100 or 1000) of initial segments, these initial segments do not grow enough to cover the complete image. We consider this implementation as broken for small numbers of segments.

5 Comparison Results

Segmentation Quality

The properties of the resulting superpixel segmentations strongly depend on the algorithm. **ERS** produces similar sized superpixels, that carefully follow image gradients. Increasing the number of segments results in a refinement of the prior segmentation (i.e. only few superpixel borders get lost, but some superpixels are divided). Further, **ERS** allows to catch an exact number of segments. Superpixel borders of **FH** strictly follow image gradients. The resulting segments vary strongly in shape and size. Sometimes, long, thin segments occur along strong image gradients. There are problems with very small segment numbers (e.g. 30) on the regarded image size (481x321). **EAMS**

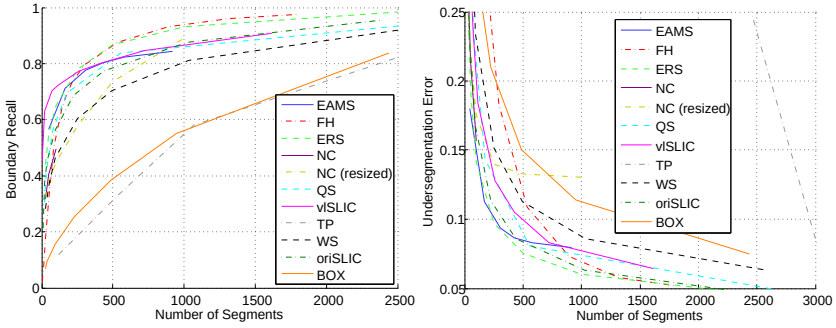
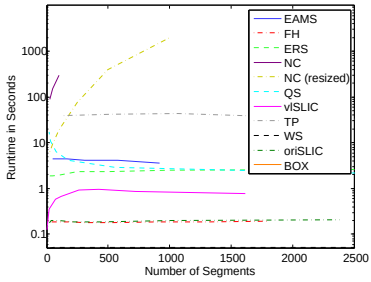


Figure 1.4: Results of segmentation algorithms on the proposed benchmark. At boundary recall top-left is better, at oversegmentation error, bottom left is better.

segments also vary in size and shape, but are more regular than with **FH**. Further, increasing the number of segments gives a good refinement of the prior segmentation. **NC** does not give such an refinement, but the superpixels are regularly sized, shaped and distributed. The **resized NC** does not give the exact same results, but the segmentation has same properties, including an exactly controllable number of segments. **QS** produces more irregular shapes and there occur small artifact segments. The regularity of segment borders of **oriSLIC** and **vlSLIC** strongly depend on the compactness parameter. Tuning this parameter is crucial. Moreover both implementations handle the compactness parameter differently. Values that create comparable smooth results for large numbers of segments produce much smoother borders at **oriSLIC** for small segment numbers. However, the compactness parameter enables adjusting the regularity of the superpixels at the cost of loosing some object borders. At **WS** the initial marker distribution influences the segmentation result. On one hand it supports a regular distribution, on the other hand does it cause small artifact segments. The resulting segment size and shape are not regulated. The quantitative benchmark results on segmentation quality are shown in figure 1.4. On boundary recall, **vlSLIC** performs very well on small numbers of segments, however, for larger numbers, **FH** and **ERS** achieve higher recall. **oriSLIC** also performs worse on small numbers, but equally well for larger numbers. This also follows the intuition, that segmentations with more irregular, longer border achieve higher boundary recall. On undersegmentation error **ERS**, **EAMS** and **oriSLIC** perform best. On very small numbers of segments, **vlSLIC** performs comparable but gets higher errors on larger numbers. As expected **TP** performs badly on small numbers of superpixels and does not achieve the performance from the origin paper [11]. The visual inspection of **BOX** "segmentations" and its quantitative performance emphasizes the benefit of superpixel segmentations.



t in sec	n=50	n=100	n=500	n=1000
EAMS	4.4	4.4	4.1	4
FH	0.13	0.13	0.13	0.13
ERS	1.85	1.93	2.37	2.52
NC	150	295	-	-
NC (resized)	9.51	19.34	402.44	-
QS	9.25	5.86	3.05	2.71
VL SLIC	0.49	0.64	0.94	0.83
TP	-	38.7	41.10	43.71
WS	0.01	0.01	0.01	0.01
ori SLIC	0.19	0.20	0.19	0.20

Figure 1.5: *(left)* Runtime of segmentation algorithms (log-scale). *(right)* Runtime in seconds for different number of superpixels.

Runtime

The runtime comparison given in figure 1.5 was carried out on an Intel Core 2 Quad Q9400 (2.66 GHz) processor with 4 GB RAM. The measurements only show tendencies, since no special efforts were made to make exact runtime measurements. **WS** is by far the fastest algorithm followed by **FH** and **oriSLIC**. **oriSLIC** is faster than **vlSLIC** since the latter follows more closely the description in [12] while the former uses some simplifications for speedup. If runtime considerations play any role in the application of an superpixel algorithm, it is important to notice, that the range of runtimes of the different algorithms covers five orders of magnitudes.

Segmentation Robustness

Figure 1.6 shows results of all algorithms on the affine transformations shifting, rotation, scaling and shearing. The number of segments is 250. **FH** and **EAMS** are invariant to image shifts, **QS** is at least robust to shifts. Results of **ERS** are inferior, but good-natured. **SLIC** is has problems with shifts, however, **vlSLIC** can better handle certain step widths. Due to the lattice arrangement of superpixels in NC, this algorithm also has serious problems

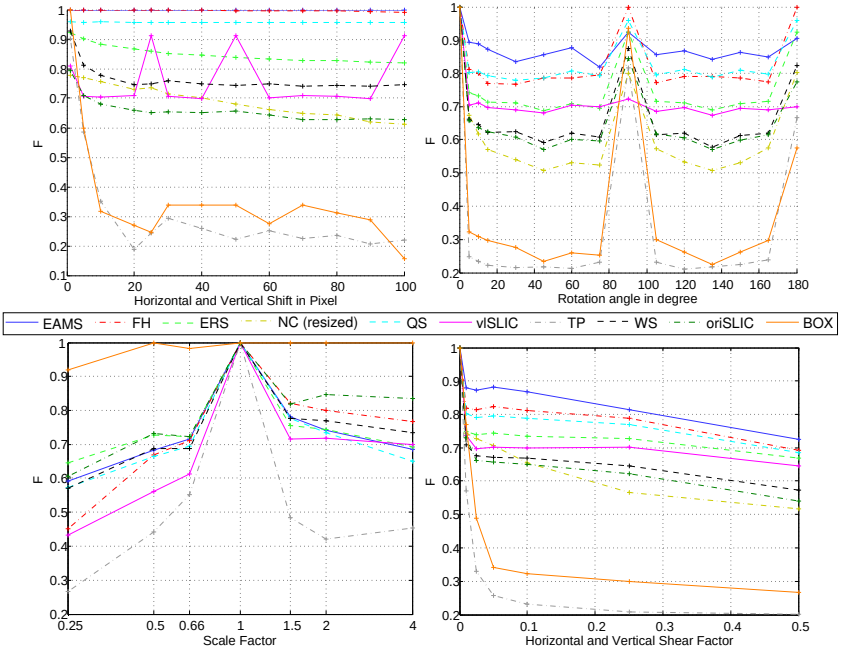


Figure 1.6: Robustness of algorithms towards shifting, rotation, scaling and shearing.

with shifts. The same happens for **WS**, due to the regular distributed markers. On rotation a periodic behavior of the performance of most algorithms can be seen. Rotation of 90 or 180 degree are less hard than arbitrary rotations. EAMS performs best on rotations, followed by **FH** and **QS**. Again, the lattice initial segment arrangements of **NC**, **WS** and **oriSLIC** cause problems. On changes on scale, all algorithms perform comparable for small changes. As expected, there is an asymmetry in favor of enlarging the image content. **ERS** deals best with decreasing image content, **oriSLIC** deals best with enlarging. On shearing, the behavior of all algorithms is similar, **EAMS** performs best.

6 Conclusions

We designed and applied a benchmark for superpixel segmentation algorithms. The implementation of all necessary functions to run the benchmark on new algorithms is available from our website[11]. We compared open source implementations of state of the art algorithms regarding quality and robustness of the segmentations. Left for future work is the evaluation of robustness with respect to variations of the intensity or color values. Algorithms that strongly depend on image data and not on a compactness constraints or lattice distribution of segments, showed to be more robust towards affine transformations. However, dependent on the application, compactness constraints or lattice distribution can be crucial. **WS** is the fastest algorithm, however, neither the segmentation quality nor the robustness is among the best. **FH** seems to be a good alternative if runtime plays a role, however, the resulting segments vary strongly in shape and size. **EAMS** showed to be the most robust against affine transformations, however, **EAMS** is inferior with respect to boundary recall. **QS** is faster than **EAMS** but yielded inferior segmentation results. **ERS** has good balanced results on segmentation properties, quality and robustness. If the runtime of 2s per image is not a problem, this could be an interesting base for many computer vision tasks. The two implementations of **SLIC** showed good, but diverging properties. Very recent, the authors of [12] announced a new zero-parameter version (SLIC0) on their website that overcomes the compactness parameter. Dependent on the application, using this parameter to adjust for more regular or more gradient aligned superpixel could be preferable. Further, on Windows machines, the closed source implementation of Lattice Cut [16] could be interesting. The implementation of **TP** seems to be broken, otherwise, this could have been a faster alternative for **NC**, which is too slow for many real world application. In general, there is a lack for faster superpixel algorithms with high segmentation quality and robustness.[12]

[11] http://www.tu-chemnitz.de/etit/proaut/forschung/superpixel.html

[12] This work has been funded by the European Union with the European Social Fund (ESF) and by the state of Saxony.

Figure 1.7: Superpixel segmentations. *(top-row)* Different segmentations by humans taken as ground truth. *(other rows)* Each row shows results of one algorithm and 3 parameter settings, visualized as overlayed boundary image and superpixels with average image colors. The algorithms parameters where chosen to produce about 25, 100 and 500 superpixels (execpt for TP and NC). The algorithm name and individual number of superpixels are given on the very left.

References

1. T. Malisiewicz and A. A. Efros, "Improving spatial support for objects via multiple segmentations," in *BMVC*, 2007.
2. C. Pantofaru, C. Schmid, and M. Hebert, "Object recognition by integrating multiple image segmentations," in *ECCV*, 2008.
3. P. Mehrani and O. Veksler, "Saliency segmentation based on learning and graph cut refinement," in *BMVC*, 2010.
4. D. Hoiem, A. A. Efros, and M. Hebert, "Automatic photo pop-up," *ACM Trans. Graph.*, vol. 24, 2005.
5. X. Ren and J. Malik, "Learning a classification model for segmentation," in *ICCV*, 2003.
6. P. F. Felzenszwalb and D. P. Huttenlocher, "Efficiently computing a good segmentation," 1998.
7. D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *TPAMI*, vol. 24, 2002.
8. M.-Y. Liu, O. Tuzell, S. Ramalingam, and R. Chellappa, "Entropy rate superpixel segmentation." in *CVPR*, 2011.
9. A. Vedaldi and S. Soatto, "Quick shift and kernel methods for mode seeking," in *ECCV*, 2008.
10. F. Meyer, "Color image segmentation," in *ICIP92*, 1992.
11. A. Levinshtein, A. Stere, K. Kutulakos, D. Fleet, S. Dickinson, and K. Siddiqi, "Turbopixels: Fast superpixels using geometric flows," *TPAMI*, 2009.
12. R. Achanta, A. Shaji, K.Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC Superpixels," Tech. Rep., 2010.
13. D. 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," in *ICCV*, 2001.
14. P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *TPAMI*, 2011.
15. P. Meer and B. Georgescu, "Edge detection with embedded confidence," *TPAMI*, vol. 23, 2001.
16. A. P. Moore, S. J. D. Prince, and J. Warrell, "Lattice cut - constructing superpixels using layer constraints," *CVPR*, 2010.