# Superpixels: An Evaluation of the State-of-the-Art

David Stutz, Alexander Hermans, Bastian Leibe

*Visual Computing Institute, RWTH Aachen University, Germany*



**Abstract**

Superpixels group perceptually similar pixels to create visually meaningful entities while heavily reducing the number of primitives. As of these properties, superpixel algorithms have received much attention since their naming in 2003. By today, publicly available and well-understood superpixel algorithms have turned into standard tools in low-level vision. As such, and due to their quick adoption in a wide range of applications, appropriate benchmarks are crucial for algorithm selection and comparison. Until now, the rapidly growing number of algorithms as well as varying experimental setups hindered the development of a unifying benchmark. We present a comprehensive evaluation of 28 state-of-the-art superpixel algorithms utilizing a benchmark focussing on fair comparison and designed to provide new and relevant insights. To this end, we explicitly discuss parameter optimization and the importance of strictly enforcing connectivity. Furthermore, by extending well-known metrics, we are able to summarize algorithm performance independent of the number of generated superpixels, thereby overcoming a major limitation of available benchmarks. Furthermore, we discuss runtime, robustness against noise, blur and affine transformations, implementation details as well as aspects of visual quality. Finally, we present an overall ranking of superpixel algorithms which redefines the state-of-the-art and enables researchers to easily select appropriate algorithms and the corresponding implementations which themselves are made publicly available as part of our benchmark at `davidstutz.de/projects/superpixel-benchmark/`.

*Keywords:* superpixels; superpixel segmentation; image segmentation; perceptual grouping; benchmark; evaluation

## 1. Introduction

Introduced by Ren and Malik in 2003 [1], superpixels group pixels similar in color and other low-level properties. In this respect, superpixels address two problems inherent to the processing of digital images [1]: firstly, pixels are merely a result of discretization; and secondly, the high number of pixels in large images prevents many algorithms from being computationally feasible. Ren and Malik introduce superpixels as more natural entities – grouping pixels which perceptually belong together while heavily reducing the number of primitives.

Superpixels have been been used in a wide range of applications – even before the term "superpixel" was coined. As early as 1988, Mester and Franke [2] present segmentation results similar to superpixels. Later, in 1997, early versions of the watershed algorithm were known to produce superpixel-like segments [3]. In the early 2000s, Hoiem et al. [4, 5] used the segmentation algorithms of [6] and [7] to generate oversegmentations for 3D reconstruction and occlusion boundaries. Similarly, the normalized cuts algorithm was early adopted for oversegmentation [1] and semantic segmentation [8]. In [4, 5] and [9], superpixels have been used to extract meaningful features for subsequent tasks – extensive lists of used features are included. Since the introduction of the first superpixel algorithms around 2009, they have been applied to many important problems in computer vision: tracking [10, 11], stereo and occlusion [12, 13], 3D-reconstruction [14], saliency [15], object detection [16, 17], depth recovery [18] and depth estimation [19], indoor scene understanding [20], scene flow [21], and as basis for convolutional neural networks [22] to name just a few. Superpixels have also been adopted in domain specific applications such as medical image segmentation [23, 24, 25] or medical image retrieval [26]. Moreover, superpixels have been found useful for dataset annotation [27, 28]. Finally, several superpixel algorithms (among others [29], [30] and [31]) have been adapted to videos and image volumes – a survey and comparison of some of these so-called supervoxel algorithms can be found in [32].

In view of this background, most authors do not make an explicit difference between superpixel algorithms and oversegmentation algorithms, i.e. superpixel algorithms are usually compared with oversegmentation algorithms and the terms have been used interchangeably (e.g. [33, 34, 35]). Veksler et al. [36] distinguish superpixel algorithms from segmentation algorithms running in "oversegmentation mode". More recently, Neubert and Protzel [37] distinguish superpixel algorithms from oversegmentation algorithms with respect to their behavior on video sequences. In general, it is very difficult to draw a clear line between superpixel algorithms and oversegmentation algo-

rithms. Several oversegmentation algorithms were not intended to generate superpixels, nevertheless, some of them share many characteristics with superpixel algorithms. We use the convention that superpixel algorithms offer control over the number of generated superpixels while segmentation algorithms running in "oversegmentation mode" do not. This covers the observations made by Veksler et al. and Neubert and Protzel.

In general, most authors (e.g. [33, 38, 30, 34]) agree on the following requirements for superpixels:

- Partition. Superpixels should define a partition of the image, i.e. superpixels should be disjoint and assign a label to every pixel.
- Connectivity. Superpixels are expected to represent connected sets of pixels.
- Boundary Adherence. Superpixels should preserve image boundaries. Here, the appropriate definition of image boundaries may depend on the application.
- Compactness, Regularity and Smoothness. In the absence of image boundaries, superpixels should be compact, placed regularly and exhibit smooth boundaries.
- Efficiency. Superpixels should be generated efficiently.
- Controllable Number of Superpixels. The number of generated superpixels should be controllable.

Some of these requirements may be formulated implicitly, e.g. Liu et al. [38] require that superpixels may not lower the achievable performance of subsequent processing steps. Achanta et al. [30] even require superpixels to increase the performance of subsequent processing steps. Furthermore, the above requirements should be fulfilled with as few superpixels as possible [38].

**Contributions.** We present an extensive evaluation of 28 algorithms on 5 datasets regarding visual quality, performance, runtime, implementation details and robustness to noise, blur and affine transformations. In particular, we demonstrate the applicability of superpixel algorithms to indoor, outdoor and person images. To ensure a fair comparison, parameters have been optimized on separate training sets; as the number of generated superpixels heavily influences parameter optimization, we additionally enforced connectivity. Furthermore, to evaluate superpixel algorithms independent of the number of superpixels, we propose to integrate over commonly used metrics such as Boundary Recall [39], Undersegmentation Error [33, 30, 35] and Explained Variation [40]. Finally, we present a ranking of the superpixel algorithms considering multiple metrics and independent of the number of generated superpixels.

**Outline.** In Section 2 we discuss important related work regarding the comparison of superpixel algorithms and subsequently, in Section 3, we present the evaluated superpixel algorithms. In Section 4 we discuss relevant datasets and introduce the used metrics in Section 5. Then, Section 6 briefly discusses problems related to parameter optimization before we present experimental results in Section 7. We conclude with a short summary in Section 8.

## 2. Related Work

Our efforts towards a comprehensive comparison of available superpixel algorithms is motivated by the lack thereof within the literature. Notable publications in this regard are [34], [30], [35], and [37]. Schick et al. [34] introduce a metric for evaluating the compactness of superpixels, while Achanta et al. [30] as well as Neubert and Protzel [35] concentrate on using known metrics. Furthermore, Neubert and Protzel evaluate the robustness of superpixel algorithms with respect to affine transformations such as scaling, rotation, shear and translation. However, they do not consider ground truth for evaluating robustness. More recently, Neubert and Protzel [37] used the Sintel dataset [41] to evaluate superpixel algorithms based on optical flow in order to assess the stability of superpixel algorithms in video sequences.

Instead of relying on a general evaluation of superpixel algrorithms, some authors compared the use of superpixel algorithms for specific computer vision tasks. Achanta et al. [30] use the approaches of [8] and [42] to assess superpixel algorithms as pre-processing step for semantic segmentation. Similarly, Strassburg et al. [43] evaluate superpixel algorithms based on the semantic segmentation approach described in [9]. Weikersdorfer et al. [44] use superpixels as basis for the normalized cuts algorithm [45] applied to classical segmentation and compare the results with the well-known segmentation algorithm by Arbeláez et al. [46]. Koniusz and Mikolajczyk [47], in contrast, evaluate superpixel algorithms for interest point extraction.

In addition to the above publications, authors of superpixel algorithms usually compare their proposed approaches to existing superpixel algorithms. Usually, the goal is to demonstrate superiority with regard to specific aspects. However, used parameter settings are usually not reported, or default parameters are used, and implementations of metrics differ. Therefore, these experiments are not comparable across publications.

Complementing the discussion of superpixel algorithms in the literature so far, and similar to [34], [30] and [35], we concentrate on known metrics to give a general, application independent evaluation of superpixel algorithms. However, we consider minimum/maximum as well as standard deviation in addition to metric averages in order assess the stability of superpixel algorithms as also considered by Neubert and Protzel [35, 37]. Furthermore, we explicitly document parameter optimization and strictly enforce connectivity to ensure fair comparison. In contrast to [35], our robustness experiments additionally consider noise and blur and make use of ground truth for evaluation. Finally, we render three well-known metrics independent of the number of generated superpixels allowing us to present a final ranking of superpixel algorithms.

## 3. Algorithms

In our comparison, we aim to discuss well-known algorithms with publicly available implementations alongside lesser-known and more recent algorithms for which implementations were partly provided by the authors. To address the large number of superpixel algorithms, and inspired by Achanta et al. [30], we find a rough categorization of the discussed algorithms helpful. For each algorithm, we present the used acronym, the reference and its number of citations[1]. In addition, we provide implementation details such as the programming lanugage, the used color space, the number of parameters as well as whether the number of superpixels, the compactness and the number of iterations (if applicable) are controllable.

**Watershed-based.** These algorithms are based on the waterhed algorithm (**W**) and usually differ in how the image is pre-processed and how markers are set. The number of superpixels is determined by the number of markers, and some watershed-based superpixel algorithms offer control over the compactness, for example **WP** or **CW**.

Name
**W** – Watershed
Reference (Google Scholar Citations) | Color
Meyer [7], 1992 (234)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB; 1 Parameter | ✓ | – | – |

Name
**CW** – Compact Watershed
Reference (Google Scholar Citations) | Color
Neubert and Protzel [48], 2014 (11)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB; 2 Parameters | ✓ | ✓ | – |

Name
**MSS** – Morphological Superpixel Segmentation
Reference (Google Scholar Citations) | Color
Benesova and Kottman [49], 2014 (4)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB; 5 Parameters | ✓ | – | – |

Name
**WP** – Water Pixels
Reference (Google Scholar Citations) | Color
Machairas et al. [50, 51], 2014 (5 + 8)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| Python; RGB; 2 Parameters | ✓ | ✓ | – |

**Density-based.** Popular density-based algorithms are Edge-Augmented Mean Shift (**EAMS**) and Quick Shift (**QS**). Both perform mode-seeking in a computed density image; each pixel is assigned to the corresponding mode it falls into. Density-based algorithms usually cannot offer control over the number of superpixels or their compactness and are, therefore, also categorized as oversegmentation algorithms.

Name
**EAMS** – Edge-Augmented Mean Shift
Reference (Google Scholar Citations) | Color
Comaniciu and Meer [52], 2002 (9631)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; RGB; 2 Parameters | – | – | – |

[1]Google Scholar citations as of October 13, 2016.

Name
**QS** – Quick Shift
Reference (Google Scholar Citations) | Color
Vedaldi and Soatto [53], 2002 (376)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; Lab; 3 Parameters | – | – | – |

**Graph-based.** Graph-based algorithms treat the image as undirected graph and partition this graph based on edge-weights which are often computed as color differences or similarities. The algorithms differ in the partitioning algorithm, for example **FH**, **ERS** and **POISE** exhibit a bottom-up merging of pixels into superpixels, while **NC** and **CIS** use cuts and **PB** uses elimination [54].

Name
**NC** – Normalized Cuts
Reference (Google Scholar Citations) | Color
Ren and Malik [1], 2002 (996)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; RGB; 3 Parameters | ✓ | – | – |

Name
**FH** – Felzenswalb and Huttenlocher
Reference (Google Scholar Citations) | Color
Felzenswalb et al. [6], 2004 (4144)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB; 3 Parameters | – | – | – |

Name
**RW** – Random Walks
Reference (Google Scholar Citations) | Color
Grady et al. [55, 56], 2004 (189 + 1587)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; RGB; 2 Parameters | ✓ | – | – |

Name
**CIS** – Constant Intensity Superpixels
Reference (Google Scholar Citations) | Color
Veksler et al. [36], 2010 (223)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Gray; 4 Parameters | ✓ | – | ✓ |

Name
**ERS**– Entropy Rate Superpixels
Reference (Google Scholar Citations) | Color
Liu et al.et al. [38], 2011 (216)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB; 3 Parameters | ✓ | – | – |

Name
**PB** – Boolean Optimization Superpixels
Reference (Google Scholar Citations) | Color
Zhang et al. [57], 2011 (36)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB; 3 Parameters | ✓ | – | – |

Name
**POISE** – Proposals for Objects from Improved Seeds and Energies
Reference (Google Scholar Citations) | Color
Humayun et al. [58], 2015 (3)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; RGB; 5 Parameters | ✓ | – | – |

**Contour evolution.** These algorithms represent superpixels as evolving contours starting from inital seed pixels.

Name
**TP** – Turbo Pixels
Reference (Google Scholar Citations) | Color
Levinshtein et al. [33], 2009 (559)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; RGB; 4 Parameters | ✓ | – | – |

Name
**ERGC** – Eikonal Region Growing Clustering
Reference (Google Scholar Citations) Color
Buyssens et al. [59, 60], 2014 (2 + 1)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Lab; 3 Parameters | ✓ | ✓ | – |

**Path-based.** Path-based approaches partition an image into superpixels by connecting seed points through pixel paths following specific criteria. The number of superpixels is easily controllable, however, compactness usually is not. Often, these algorithms use edge information: **PF** uses discrete image gradients and **TPS** uses edge detection as proposed in [61].

Name
**PF** – Path Finder
Reference (Google Scholar Citations) Color
Drucker et al. [62], 2009 (18)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| Java; RGB; 2 Parameters | ✓ | – | – |

Name
**TPS** – Topology Preserving Superpixels
Reference (Google Scholar Citations) Color
Dai et al. [63, 64], 2012 (8 + 1)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; RGB; 4 Parameters | ✓ | – | – |

**Clustering-based.** These superpixel algorithms are inspired by clustering algorithms such as $k$-means initialized by seed pixels and using color information, spatial information and additional information such as depth (as for example done by **DASP**). Intuitively, the number of generated superpixels and their compactness is controllable. Although these algorithms are iterative, post-processing is required in order to enforce connectivity.

Name
**SLIC** – Simple Linear Iterative Clustering
Reference (Google Scholar Citations) Color
Achanta et al. [65, 30], 2010 (438 + 1843)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Lab; 4 Parameters | ✓ | ✓ | ✓ |

Name
**DASP** – Depth-Adaptive Superpixels
Reference (Google Scholar Citations) Color
Weikersdorfer et al. [44], 2012 (22)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB**D**; 5 Parameters | ✓ | ✓ | ✓ |

Name
**VC** – VCells
Reference (Google Scholar Citations) Color
Wang and Wang [66], 2012 (36)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Lab; 6 Parameters | ✓ | ✓ | – |

Name
**VCCS** – Voxel-Cloud Connectivity Segmentation
Reference (Google Scholar Citations) Color
Papon et al. [67], 2013 (87)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB**D**; 4 Parameters | – | ✓ | – |

Name
**preSLIC** – Preemptive SLIC
Reference (Google Scholar Citations) Color
Neubert and Protzel [48], 2014 (11)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Lab; 4 Parameters | ✓ | ✓ | ✓ |

Name
**LSC** – Linear Spectral Clustering
Reference (Google Scholar Citations) Color
Li and Chen [68], 2015 (2)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Lab; 4 Parameters | ✓ | ✓ | ✓ |

We note that **VCCS** directly operates within a point cloud and we, therefore, backproject the generated supervoxels onto the image plane. Thus, the number of generated superpixels is harder to control.

**Energy optimization.** These algorithms iteratively optimize a formulated energy. The image is partitioned into a regular grid as initial superpixel segmentation, and pixels are exchanged between neighboring superpixels with regard to the energy. The number of superpixels is controllable, compactness can be controlled and the iterations can usually be aborted at any point.

Name
**CRS** – Contour Relaxed Superpixels
Reference (Google Scholar Citations) Color
Conrad et al. [69, 70], 2011 (14 + 4)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; YCrCb; 4 Parameters | ✓ | ✓ | ✓ |

Name
**SEEDS** – Superpixels Extracted via Energy-Driven Sampling
Reference (Google Scholar Citations) Color
Van den Bergh et al. [71], 2012 (98)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Lab; 6 Parameters | ✓ | – | ✓ |

Name
**CCS** – Convexity Constrained Superpixels
Reference (Google Scholar Citations) Color
Tasli et al. [72, 73], 2013 (6 + 4)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; Lab; 3 Parameters | ✓ | ✓ | ✓ |

Name
**ETPS** – Extended Topology Preserving Segmentation
Reference (Google Scholar Citations) Color
Yao et al. [74], 2015 (6)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| C/C++; RGB; 5 Parameters | ✓ | ✓ | ✓ |

**Wavelet-based.** We found that Superpixels from Edge-Avoiding Wavelets (**SEAW**) [43] is not yet captured in the discussed categories. In particular, it is not comparable to the algorithms discussed so far.

Name
**SEAW** – Superpixels from Edge-Avoiding Wavelets
Reference (Google Scholar Citations) Color
Strassburg et al. [43], 2015 (0)

| Implementation | Superpixels | Compactness | Iterations |
|---|---|---|---|
| MatLab/C; RGB; 3 Parameters | ✓ | – | – |

While the above algorithms represent a large part of the proposed superpixel algorithms, some algorithms are not included due to missing, unnoticed or only recently published implementations[2]. These include [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 84, 88, 89].

[2]Visit `davidstutz.de/projects/superpixel-benchmark/` to integrate your algorithm into the comparison. We are currently working on integrating [84] and [88].

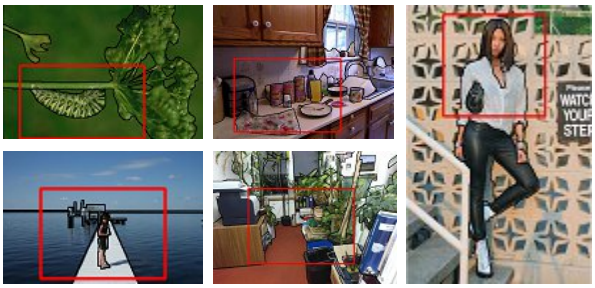


Figure 1: Example images from the used datasets. From left to right: BSDS500, SBD, NYUV2, SUNRGBD, and Fash. Black contours represent ground truth and red rectangles indicate excerpts used for qualitative comparison in Figures 5 and 6. **Best viewed in color.**

| | | BSDS500 | SBD | NYUV2 | SUNRGBD | Fash |
|---|---|---|---|---|---|---|
| Images | Train | 100 | 238 | 199 | 200 | 222 |
| | Test | 200 | 477 | 399 | 400 | 463 |
| Size | Train | 481 × 321 | 316 × 240 | 608 × 448 | 658 × 486 | 400 × 600 |
| | Test | 481 × 321 | 314 × 242 | 608 × 448 | 660 × 488 | 400 × 600 |

Table 1: Basic statistics of the used datasets: the total number of images, the number of training and test images and the size of the images (averaged per dimension). The number of images for the SUNRGBD dataset excludes the images from the NYUV2 dataset. For the NYUV2 and SUNRGBD datasets, training and test images have been chosen uniformly at random if necessary. Note that the odd numbers used for the NYUV2 dataset are for no special reason.

## 4. Datasets

We chose five different datasets to evaluate superpixel algorithms: two indoor datasets, two outdoor datasets and one person dataset. We found that these datasets realisticly reflect the setting of common applications, while leveraging the availability of large, pixel-level annotated datasets. In addition, these datasets enable us to draw a more complete picture of algorithm performance going beyond the datasets commonly used within the literature. Furthermore, both indoor datasets provide depth information, allowing us to evaluate superpixel algorithms requiring depth information as additional cue. In the following we briefly discuss the main aspects of these datasets; Figure 1 shows example images and Table 1 summarizes key statistics.

BSDS500 [46]. The Berkeley Segmentation Dataset 500 (BSDS500) was the first to be used for superpixel algorithm evaluation (e.g. [1, 33]). It contains 500 images and provides at least 5 high-quality ground truth segmentations per image. Therefore, we evaluate algorithms on all ground truth segmentations and, for each image and a given metric, choose the the ground truth segmentation resulting in the worst score for averaging. The images represent simple outdoor scenes, showing landscape, buildings, animals and humans, where foreground and background are usually easily identified. Nevertheless, natural scenes where segment boundaries are not clearly identifiable, contribute to the difficulty of the dataset.

SBD [90]. The Stanford Background Dataset (SBD) combines 715 images from several datasets [91, 92, 93, 94]. As result, the dataset contains images of varying size, quality and scenes. The images show outdoor scenes such as landscape, animals or street scenes. In contrast to the BSDS500 dataset the scenes tend to be more complex, often containing multiple foreground objects or scenes without clearly identifiable foreground. The semantic ground truth has been pre-processed to ensure connected segments.

NYUV2 [95]. The NYU Depth Dataset V2 (NYUV2) contains 1449 images including pre-processed depth. Silberman et al. provide instance labels which are used to ensure connected segments. Furthermore, following Ren and Bo [96], we pre-processed the ground truth to remove small unlabeled regions. The provided ground truth is of lower quality compared to the BSDS500 dataset. The images show varying indoor scenes of private apartments and commercial accomodations which are often cluttered and badly lit. The images were taken using Microsoft's Kinect.

SUNRGBD [97]. The Sun RGB-D dataset (SUNRGBD) contains 10335 images including pre-processed depth. The dataset combines images from the NYUV2 dataset and other datasets [98, 99] with newly acquired images. In contrast to the NYUV2 dataset, the SUNRGBD dataset combines images from the following devices: Intel RealSense, Asus Xtion and Microsoft Kinect v1 and v2 – we refer to [97] for details. We removed the images taken from the NYUV2 dataset. The images show cluttered indoor scenes with bad lighting taken from private apartments as well as commercial accomodations. The provided semantic ground truth has been pre-processed similarly to those of the NYUV2 dataset.

Fash [27]. The Fashionista dataset (Fash) contains 685 images which have previously been used for clothes parsing. The images show the full body of fashion bloggers in front of various backgrounds. Yamaguchi et al. leveraged Amazon Mechanical Turk to acquire semantic ground truth based on pre-computed segments ([27] suggests that the algorithm in [46] has been used). The ground truth has been pre-processed to ensure connected segments.

## 5. Benchmark

Our benchmark aims to score the requirements for superpixels discussed in Section 1; in particular boundary adherence and compactness (note that connectivity is enforced during parameter optimization, see Section 6.2). As these metrics inherently depend on the number of generated superpixels, we further extend these metrics to allow the assessment of superpixel algorithms independent of the number of generated superpixels. Therefore, let $S = \{S_j\}_{j=1}^{\mathrm{K}}$ and $G = \{G_i\}$ be partitions of the same image $I : x_n \mapsto I(x_n)$, $1 \leq n \leq N$, where $S$ represents a superpixel segmentation and $G$ a ground truth segmentation.

Boundary Recall (Rec) [39] is the most commonly used metric to asses boundary adherence given ground truth. Let $\mathrm{FN}(G, S)$ and $\mathrm{TP}(G, S)$ be the number of false negative and true positive boundary pixels in $S$ with respect

to $G$. Then Rec is defined as

$$\mathrm{Rec}(G,S) = \frac{\mathrm{TP}(G,S)}{\mathrm{TP}(G,S) + \mathrm{FN}(G,S)}. \quad (1)$$

Overall, high Rec represents better boundary adherence with respect to to the ground truth boundaries, i.e. higher is better. In practice, a boundary pixel in $S$ is matched to an arbitrary boundary pixel in $G$ within a local neighborhood of size $(2r+1)\times(2r+1)$, with $r$ being 0.0025 times the image diagonal rounded to the next integer (e.g. $r=1$ for the BSDS500 dataset).

Undersegmentation Error (UE) [33, 30, 35] measures the "leakage" of superpixels with respect to $G$ and, therefore, implicitly also measures boundary adherence. The original formulation by Levinshtein et al. [33] can be written as

$$\mathrm{UE}_{\mathrm{Levin}}(G,S) = \frac{1}{|G|}\sum_{G_i}\frac{\left(\sum_{S_j\cap G_i\neq\emptyset}|S_j|\right)-|G_i|}{|G_i|} \quad (2)$$

where the inner term represents the "leakage" of superpixel $S_j$ with respect to $G$. However, some authors [30, 35] argue that Equation (2) penalizes superpixels overlapping only slightly with neighboring ground truth segments and is not constrained to lie in $[0,1]$. Achanta et al. [30] suggest to threshold the "leakage" term of Equation (2) and only consider those superpixels $S_j$ with a minimum overlap of $\frac{5}{100}\cdot|S_j|$. In contrast, Neubert and Protzel [35] propose a new formulation not suffering from the above drawbacks:

$$\mathrm{UE}_{\mathrm{NP}}(G,S) = \frac{1}{N}\sum_{G_i}\sum_{S_j\cap G_i\neq\emptyset}\min\{|S_j\cap G_i|, |S_j - G_i|\}. \quad (3)$$

All formulations have in common that lower UE refers to less leakage with respect to the ground truth, i.e. lower is better. In the following we use $\mathrm{UE}\equiv\mathrm{UE}_{\mathrm{NP}}$.

Explained Variation (EV) [40] quantifies the quality of a superpixel segmentation without relying on ground truth. As image boundaries tend to exhibit strong change in color and structure, EV assesses boundary adherence independent of human annotions. EV is defined as

$$\mathrm{EV}(S) = \frac{\sum_{S_j}|S_j|(\mu(S_j)-\mu(I))^2}{\sum_{x_n}(I(x_n)-\mu(I))^2} \quad (4)$$

where $\mu(S_j)$ and $\mu(I)$ are the mean color of superpixel $S_j$ and the image $I$, respectively. As result, higher EV quantifies the variation of the image explained by the superpixels, i.e. higher is better.

Compactness (CO) [34] has been introduced by Schick et al. [34] to evaluate the compactness of superpixels:

$$\mathrm{CO}(G,S) = \frac{1}{N}\sum_{S_j}|S_j|\frac{4\pi A(S_j)}{P(S_j)}. \quad (5)$$

CO compares the area $A(S_j)$ of each superpixel $S_j$ with the area of a circle (the most compact 2-dimensional shape) with same perimeter $P(S_j)$, i.e. higher is better.

While we will focus on Rec, UE, EV and CO, further notable metrics include: Achievable Segmentation Accuracy (ASA) [38] quantifies the achievable accuracy for segmentation using superpixels as pre-processing step:

$$\mathrm{ASA}(G,S) = \frac{1}{N}\sum_{S_j}\max_{G_i}\{|S_j\cap G_i|\}; \quad (6)$$

Intra-Cluster Variation (ICV) [49] computes the average variation within each superpixel:

$$\mathrm{ICV}(S) = \frac{1}{|S|}\sum_{S_j}\frac{\sqrt{\sum_{x_n\in S_j}(I(x_n)-\mu(S_j))^2}}{|S_j|}; \quad (7)$$

Mean Distance to Edge (MDE) [49] refines Rec by also considering the distance to the nearest boundary pixel within the ground truth segmentation:

$$\mathrm{MDE}(G,S) = \frac{1}{N}\sum_{x_n\in B(G)}\mathrm{dist}_S(x_n) \quad (8)$$

where $B(G)$ is the set of boundary pixels in $G$, and $\mathrm{dist}_S$ is a distance transform of $S$.

### *5.1. Expressiveness and Chosen Metrics*

Due to the large number of available metrics, we examined their expressiveness in order to systematically concentrate on few relevant metrics. We found that UE tends to correlate strongly with ASA which can be explained by Equations (3) and (6), respectively. In particular, simple calculation shows that ASA strongly resembles $(1-\mathrm{UE})$. Surprisingly, $\mathrm{UE}_{\mathrm{NP}}$ does not correlate with $\mathrm{UE}_{\mathrm{Levin}}$ suggesting that either both metrics reflect different aspects of superpixels, or $\mathrm{UE}_{\mathrm{Levin}}$ unfairly penalizes some superpixels as suggested in [30] and [35]. Unsurprisingly, MDE correlates strongly with Rec which can also be explained by their respective definitions. In this sense, MDE does not provide additional information. Finally, ICV does not correlate with EV which may be attributed to the missing normalization in Equation (7) when compared to EV. This also results in ICV not begin comparable across images as the intra-cluster variation is not related to the overall variation within the image. As of these considerations, we concentrate on Rec, UE and EV for the presented experiments. Details can be found in Appendix C.1.

### *5.2. Average Recall, Average Undersegmentation Error and Average Explained Variation*

As the chosen metrics inherently depend on the number of superpixels, we seek a way of summarizing the performance with respect to Rec, UE and EV independent of K. To this end, we use the area above the Rec, $(1-\mathrm{UE})$ and EV curves in the interval $[\mathrm{K}_{\min},\mathrm{K}_{\max}] = [200, 5200]$

to quantify performance independent of K. In Section 7.2, we will see that these metrics appropriately summarize the performance of superpixel algorithms. To avoid confusion, we denote these metrics by Average Recall (ARec), Average Undersegmentation Error (AUE) and Average Explained Variation (AEV) – note that this refers to an average over K. By construction, lower ARec, AUE and AEV is better.

## 6. Parameter Optimization

For the sake of fair comparison, we optimized parameters on the training sets depicted in Table 1. Unfortunately, parameter optimization is not explicitly discussed in related work (e.g. [34, 30, 35, 34]) and used parameters are not reported in most publications. In addition, varying runtimes as well as categorical and integer parameters render parameter optimization difficult such that we had to rely on discrete grid search, jointly optimizing Rec and UE, i.e. minimizing $(1 - \text{Rec}) + \text{UE}$. In the following, we briefly discuss the main difficulties encountered during parameter optimization, namely controlling the number of generated superpixels and ensuring connectivity.

### *6.1. Controlling the Number of Generated Superpixels*

As discussed in Section 1, superpixel algorithms are expected to offer control over the number of generated superpixels. We further expect the algorithms to meet the desired number of superpixels within acceptable bounds. For several algorithms, however, the number of generated superpixels is strongly dependent on other parameters. Figure 2 demonstrates the influence of specific parameters on the number of generated superpixels (before ensuring connectivity as in Section 6.2) for **LSC**, **CIS**, **VC**, **CRS** and **PB**. For some of the algorithms, such parameters needed to be constrained to an appropriate value range even after enforcing connectivity.

For oversegmentation algorithms such as **FH**, **EAMS** and **QS** not providing control over the number of generated superpixels, we attempted to exploit simple relationships between the provided parameters and the number of generated superpixels. For **EAMS** and **QS** this allows to control the number of generated superpixels at least roughly. **FH**, in contrast, does not allow to control the number of generated superpixels as easily. Therefore, we evaluated **FH** for a large set of parameter combinations and chose the parameters resulting in approximately the desired number of superpixels.

### *6.2. Ensuring Connectivity*

Unfortunately, many implementations (note the difference between implementation and algorithm) cannot ensure the connectivity of the generated superpixels as required in Section 1. Therefore, we decided to strictly enforce connectivity using a connected components algorithm, i.e. after computing superpixels, each connected component is relabeled as separate superpixel. For some implementations, this results in many unintended superpixels comprising few pixels. In these cases we additionally merge the newly generated superpixels into larger neighboring ones. However, even with these post-processing steps, the evaluated implementations of **CIS**, **CRS**, **PB**, **DASP**, **VC**, **VCCS** or **LSC** generate highly varying numbers of superpixels across different images.

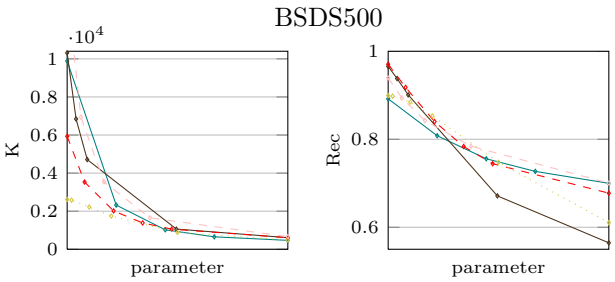



Figure 2: K and Rec on the training set of the BSDS500 dataset when varying parameters strongly influencing the number of generated superpixels of: **LSC**; **CIS**; **VC**; **CRS**; and **PB**. The parameters have been omitted and scaled for clarity. A higher number of superpixels results in increased Rec. Therefore, unnoticed superpixels inherently complicate fair comparison. **Best viewed in color.**

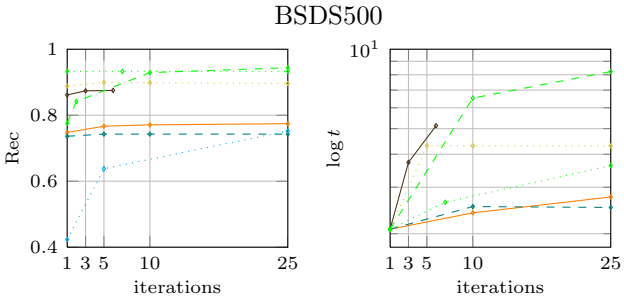



Figure 3: Rec and runtime in seconds $t$ on the training set of the BSDS500 dataset when varying the number of iterations of: **SLIC**; **CRS**; **SEEDS**; **preSLIC**; **LSC**; and **ETPS**. Most algorithms achieve reasonable Rec with about $3 - 10$ iterations. Still, parameter optimization with respect to Rec and UE favors more iterations. **Best viewed in color.**

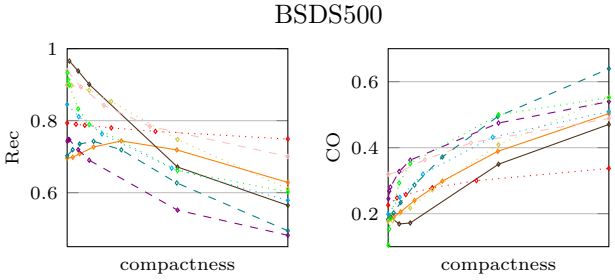



Figure 4: Rec and CO on the training set of the BSDS500 dataset when varying the compactness parameter of: **SLIC**; **CRS**; **VC**; **preSLIC**; **CW**; **ERGC**; **LSC**; and **ETPS**. The parameters have been omitted and scaled for clarity. High CO comes at the cost of reduced Rec and parameter optimization with respect to Rec and UE results in less compact superpixels. **Best viewed in color.**

### 6.3. Common Trade-Offs: Runtime and Compactness

Two other types of parameters deserve detailed discussion: the number of iterations and the compactness parameter. The former controls the trade-off between runtime and performance, exemplarily demonstrated in Figure 3 showing that more iterations usually result in higher Rec and higher runtime in seconds $t$. The latter controls the trade-off between compactness and performance and Figure 4 shows that higher CO usually results in lower Rec. Overall, parameter optimization with respect to Rec and UE results in higher runtime and lower compactness.

## 7. Experiments

Our experiments include visual quality, performance with respect to Rec, UE and EV as well as runtime. In contrast to existing work [34, 30, 35, 34], we consider minimum/maximum and standard deviation of Rec, UE and EV (in relation to the number of generated superpixels K) and present results for the introduced metrics ARec, AUE and AEV. Furthermore, we present experiments regarding implementation details as well as robustness against noise, blur and affine transformations. Finally, we give an overall ranking based on ARec and AUE.

### 7.1. Qualitative

Visual quality is best determined by considering compactness, regularity and smoothness on the one hand and boundary adherence on the other. Here, compactness refers to the area covered by individual superpixels (as captured in Equation (5)); regularity corresponds to both the superpixels' sizes and their arrangement; and smootness refers to the superpixels' boundaries. Figures 5 and 6 show results on all datasets. We begin by discussing boundary adherence, in particular, with regard to the difference between superpixel and oversegmentation algorithms, before considering compactness, smoothness and regularity.

The majority of algorithms provides solid adherence to important image boundaries, especially for large K. We consider the woman image – in particular, the background – and the caterpillar image in Figure 5. Algorithms with inferior boundary adherence are easily identified as those not capturing the pattern in the background or the silhouette of the caterpillar: **FH**, **QS**, **CIS**, **PF**, **PB**, **TPS**, **TP** and **SEAW**. The remaining algorithms do not necessarily capture all image details, as for example the woman's face, but important image boundaries are consistently captured. We note that of the three evaluated oversegmentation algorithms, i.e. **EAMS**, **FH** and **QS**, only **EAMS** demonstrates adequate boundary adherence. Furthermore, we observe that increasing K results in more details being captured by all algorithms. Notable algorithms regarding boundary adherence include **CRS**, **ERS**, **SEEDS**, **ERGC** and **ETPS**. These algorithms are able to capture even smaller details such as the coloring of the caterpillar or elements of the woman's face.

Compactness strongly varies across algorithms and a compactness parameter is beneficial to control the degree of compactness as it allows to gradually trade boundary adherence for compactness. We consider the caterpillar image in Figure 5. **TP**, **RW**, **W**, and **PF** are examples for algorithms not providing a compactness parameter. While **TP** generates very compact superpixels and **RW** tends to resemble grid-like superpixels, **W** and **PF** generate highly non-compact superpixels. In this regard, compactness depends on algorithm and implementation details (e.g. grid-like initialization) and varies across algorithms. For algorithms providing control over the compactness of the generated superpixels, we find that parameter optimization has strong impact on compactness. Examples are **CRS**, **LSC**, **ETPS** and **ERGC** showing highly irregular superpixels, while **SLIC**, **CCS**, **VC** and **WP** generate more compact superpixels. For **DASP** and **VCCS**, requiring depth information, similar observations can be made on the kitchen image in Figure 6. Inspite of the influence of parameter optimization, we find that a compactness parameter is beneficial. This can best be observed in Figure 8, showing superpixels generated by **SLIC** and **CRS** for different degrees of compactness. We observe that compactness can be increased while only gradually sacrificing boundary adherence.

We find that compactness does not necessarily induce regularity and smoothness; some algorithms, however, are able to unite compactness, regularity and smoothness. Considering the sea image in Figure 5 for **CIS** and **TP**, we observe that compact superpixels are not necessarily arranged regularly. Similarly, compact superpixels do not need to exhibit smooth boundaries, as can be seen for **PB**. On the other hand, compact superpixels are often generated in a regular fashion, as can be seen for many algorithms providing a compactness parameter such as **SLIC**, **VC** and **CCS**. In such cases, compactness also induces smoother and more regular superpixels. We also observe that many algorithms exhibiting excellent boundary adherence such as **CRS**, **SEEDS** or **ETPS** generate highly irregular and non-smooth superpixels. These observations also justify the separate consideration of compactness, regularity and smoothness to judge visual quality. While the importance of compactness, regularity and smoothness may depend on the application at hand, these properties represent the trade-off between abstraction from and sensitivity to low-level image content which is inherent to all superpixel algorithms.

In conclusion, we find that the evaluated path-based and density-based algorithms as well as oversegmentation algorithms show inferior visual quality. On the other hand, clustering-based, contour evolution and iterative energy optimization algorithms mostly show good boundary adherence and some provide a compactness parameter, e.g. **SLIC**, **ERGC** and **ETPS**. Graph-based algorithms show mixed results – algorithms such as **FH**, **CIS** and **PB** showing inferior boundary adherence, while **ERS**, **RW**, **NC** and **POISE** exhibit better boundary adherence. However,

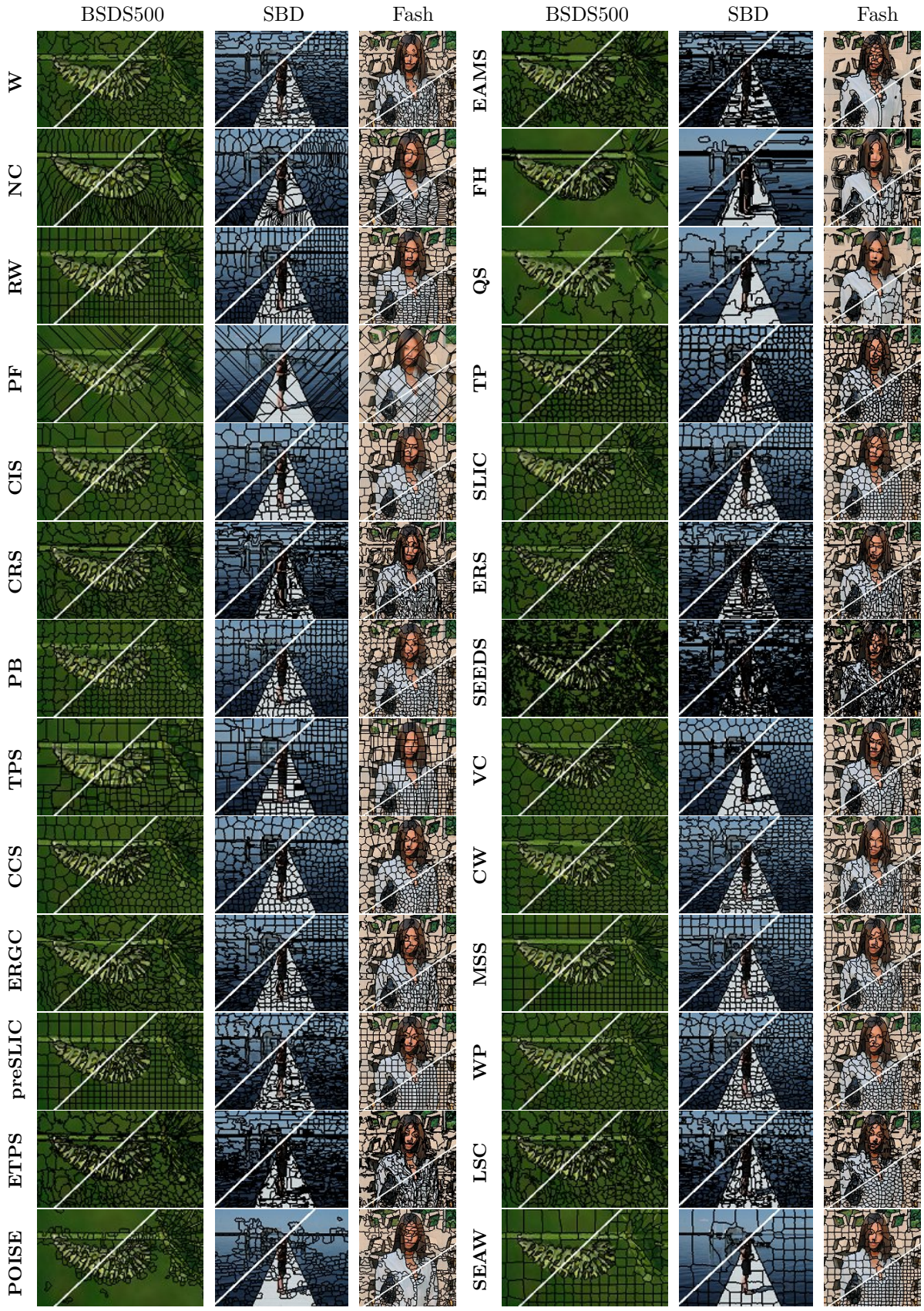



Figure 5: Qualitative results on the BSDS500, SBD and Fash datasets. Excerpts from the images in Figure 1 are shown for $K \approx 400$ in the upper left corner and $K \approx 1200$ in the lower right corner. Superpixel boundaries are depicted in black; best viewed in color. We judge visual quality on the basis of boundary adherence, compactness, smoothness and regularity. Boundary adherence can be judged both on the caterpillar image as well as on the woman image – the caterpillar's boundaries are hard to detect and the woman's face exhibits small details. In contrast, compactness, regularity and smoothness can be evaluated considering the background in the caterpillar and see images. **Best viewed in color.**

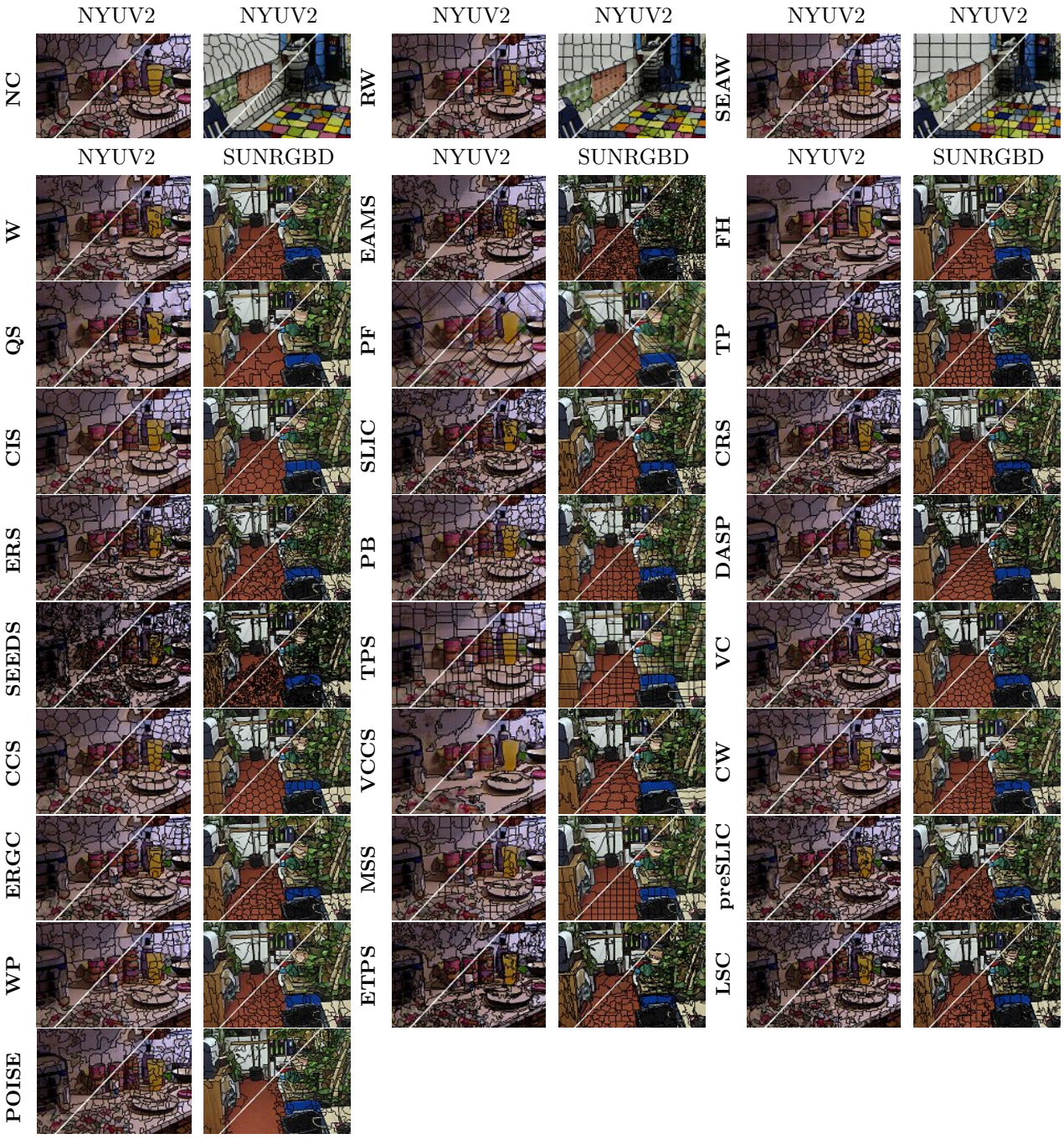



Figure 6: Qualitative results on the NYUV2 and SUNRGBD datasets. Excerpts from the images in Figure 1 are shown for $K \approx 400$ in the upper left corner and $K \approx 1200$ in the lower right corner. Superpixel boundaries are depicted in black; best viewed in color. **NC**, **RW** and **SEAW** could not be evaluated on the SUNRGBD dataset due to exhaustive memory usage of the corresponding MatLab implementations. Therefore, results for the NYUV2 dataset are shown. Visual quality is judged regarding boundary adherence, compactness, smoothness and regularity. We also find that depth information, as used in **DASP** and **VCCS**, may help resemble the underlying 3D-structure. **Best viewed in color.**


Figure 7: The influence of a low, on the left, and high, on the right, compactness parameter demonstrated on the caterpillar image from the BSDS500 datasets using **SLIC** and **CRS** for K $\approx$ 400. Superpixel boundaries are depicted in black; best viewed in color. Superpixel algorithms providing a compactness parameter allow to trade boundary adherence for compactness. **Best viewed in color.**

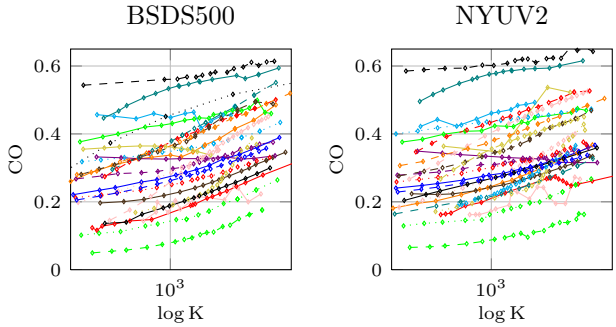



Figure 8: CO on the BSDS500 and NYUV2 datasets. Considering Figures 5 and 6, CO appropriately reflects compactness. However, it does not take into account other aspects of visual quality such as regularity and smoothness. Therefore, we find that CO is of limited use in a quantitative assessment of visual quality. **Best viewed in color.**

good boundary adherence, especially regarding details in the image, often comes at the price of lower compactness, regularity and/or smoothness as can be seen for **ETPS** and **SEEDS**. Furthermore, compactness, smoothness and regularity are not necessarily linked and should be discussed separately.

#### *7.1.1. Compactness*

CO measures compactness, however, does not reflect regularity or smoothness; therefore, CO is not sufficient to objectively judge visual quality. We consider Figure 8, showing CO on the BSDS500 and NYUV2 datasets, and we observe that CO correctly measures compactness. For example, **WP**, **TP** and **CIS**, exhibiting high CO, also present very compact superpixels in Figures 5 and 6. However, these superpixels do not necessarily have to be visually appealing, i.e. may lack regularity and/or smoothness. This can exemplarily be seen for **TPS**, exhibiting high compactness bur poor regularity, or **PB** showing high compactness but inferior smoothness. Overall, we find that CO should not be considered isolated from a qualitative evaluation.

#### *7.1.2. Depth*

Depth information helps superpixels resemble the 3D-structure within the image. Considering Figure 6, in particular both images for **DASP** and **VCCS**, we deduce that depth information may be beneficial for superpixels to resemble the 3D-structure of a scene. For example, when considering planar surfaces (e.g. the table) in both images from Figure 6 for **DASP**, we clearly see that the superpixels easily align with the floor in a way perceived as 3-dimensional. For **VCCS**, this effect is less observable which may be due to the compactness parameter.

### *7.2. Quantitative*

Performance is determined by Rec, UE and EV. In contrast to most authors, we will look beyond metric averages. In particular, we consider the minimum/maximum as well as the standard deviation to get an impression of the behavior of superpixel algorithms. Furthermore, this allows to quantize the stability of superpixel algorithms as also considered by Neubert and Protzel in [37].

Rec and UE offer a ground truth dependent overview to assess the performance of superpixel algorithms. We consider Figures 9a and 9b, showing Rec and UE on the BSDS500 dataset. With respect to Rec, we can easily identify top performing algorithms, such as **ETPS** and **SEEDS**, as well as low performing algorithms, such as **FH**, **QS** and **PF**. However, the remaining algorithms lie closely together in between these two extremes, showing (apart from some exceptions) similar performance especially for large K. Still, some algorithms perform consistently better than others, as for example **ERGC**, **SLIC**, **ERS** and **CRS**. For UE, low performing algorithms, such as **PF** or **QS**, are still easily identified while the remaining algorithms tend to lie more closely together. Nevertheless, we can identify algorithms consistently demonstrating good performance, such as **ERGC**, **ETPS**, **CRS**, **SLIC** and **ERS**. On the NYUV2 dataset, considering Figures 10a and 10b, these observations can be confirmed except for minor differences as for example the excellent performance of **ERS** regarding UE or the better performance of **QS** regarding UE. Overall, Rec and UE provide a quick overview of superpixel algorithm performance but might not be sufficient to reliably discriminate superpixel algorithms.

In contrast to Rec and UE, EV offers a ground truth independent assessment of superpixel algorithms. Considering Figure 9c, showing EV on the BSDS500 dataset, we observe that algorithms are dragged apart and even for large K significantly different EV values are attained. This suggests, that considering ground truth independent metrics may be beneficial for comparison. However, EV cannot replace Rec or UE, as we can observe when comparing to Figures 9a and 9b, showing Rec and UE on the BSDS500 dataset; in particular **QS**, **FH** and **CIS** are performing significantly better with respect to EV than regarding Rec and UE. This suggests that EV may be used to identify poorly performing algorithms, such as **TPS**, **PF**, **PB** or **NC**. On the other hand, EV is not necessarily suited to identify well-performing algorithms due to the lack of underlying ground truth. Overall, EV is suitable to complement the view provided by Rec and UE, however, should not be considered in isolation.

The stability of superpixel algorithms can be quantified by min Rec, max UE and min EV considering the behavior for increasing K. We consider Figures 9d, 9e and 9f, showing min Rec, max UE and min EV on the BSDS500 dataset.

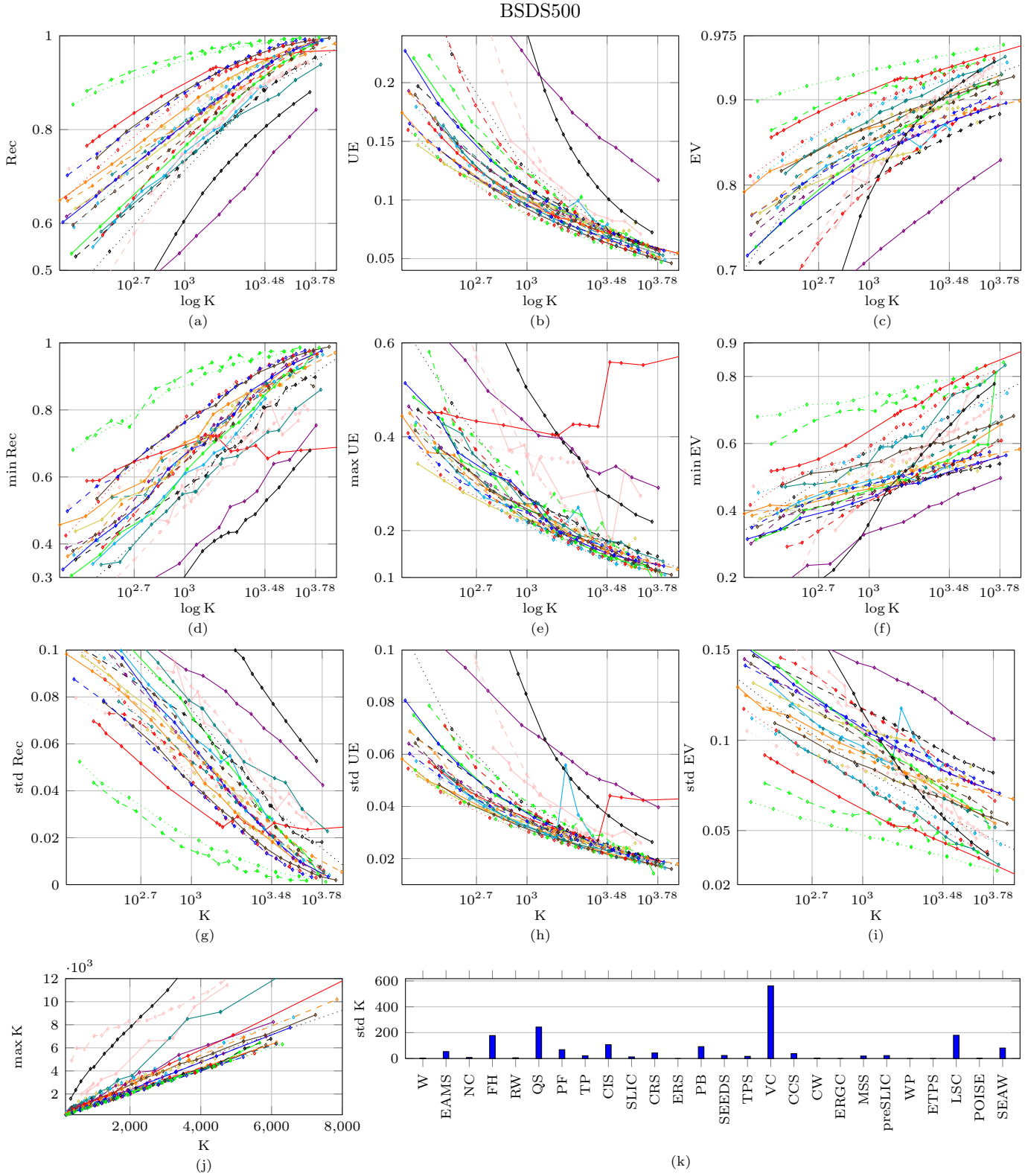



Figure 9: Quantitative experiments on the BSDS500 dataset; remember that K denotes the number of generated superpixels. Rec (higher is better) and UE (lower is better) give a concise overview of the performance with respect to ground truth. In contrast, EV (higher is better) gives a ground truth independent view on performance. While top-performers as well as poorly performing algorithms are easily identified, we provide more find-grained experimental results by considering min Rec, max UE and min EV. These statistics additionally can be used to quantity the stability of superpixel algorithms. In particular, stable algorithms are expected to exhibit monotonically improving min Rec, max UE and min EV. The corresponding std Rec, std UE and std EV as well as max K and std K help to identify stable algorithms. **Best viewed in color.**

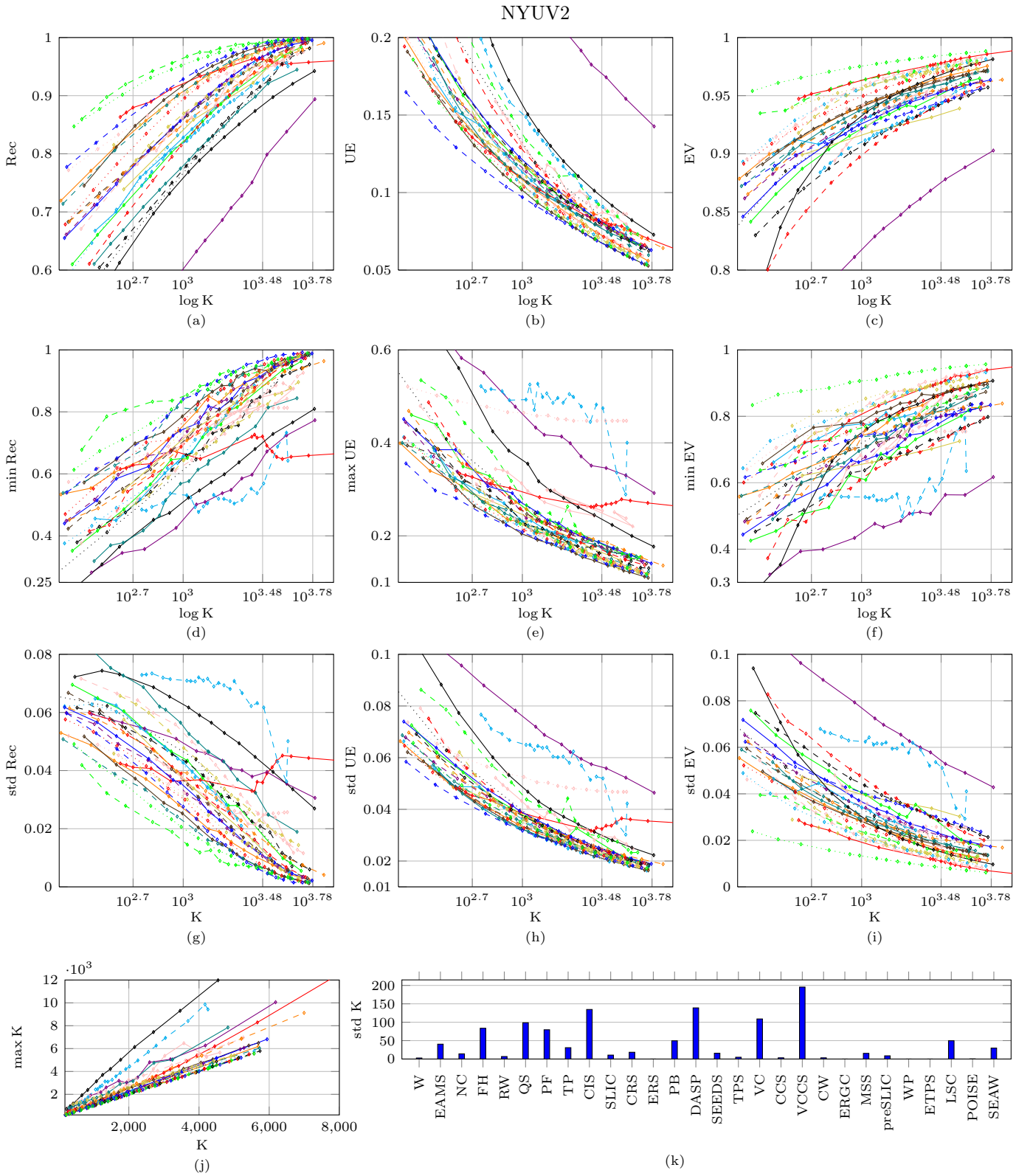



Figure 10: Quantitative results on the NYUV2 dataset; remember that K denotes the number of generated superpixels. The presented experimental results complement the discussion in Figure 9 and show that most observations can be confirmed across datasets. Furthermore, **DASP** and **VCCS** show inferior performance suggesting that depth information does not necessarily improve performance. **Best viewed in color.**

**W** **EAMS** **NC** **FH** **RW** **QS** **PF** **TP** **CIS**
**SLIC** **CRS** **ERS** **PB** **DASP** **SEEDS** **TPS** **VC** **CCS**
**VCCS** **CW** **ERGC** **MSS** **preSLIC** **WP** **ETPS** **LSC** **POISE**
**SEAW**

We define the stability of superpixel algorithms as follows: an algorithm is considered stable if performance monotonically increases with K (i.e. monotonically increasing Rec and EV and monotonically decreasing UE). Furthermore, these experiments can be interpreted as empirical bounds on the performance. For example algorithms such as **ETPS**, **ERGC**, **ERS**, **CRS** and **SLIC** can be considered stable and provide good bounds. In contrast, algorithms such as **EAMS**, **FH**, **VC** or **POISE** are punished by considering min Rec, max UE and min EV and cannot be described as stable. Especially oversegmentation algorithms show poor stability. Most strikingly, **EAMS** seems to perform especially poorly on at least one image from the BSDS500 dataset. Overall, we find that min Rec, max UE and min EV appropriately reflect the stability of superpixel algorithms.

The minimum/maximum of Rec, UE and EV captures lower/upper bounds on performance. In contrast, the corresponding standard deviation can be thought of as the expected deviation from the average performance. We consider Figures 9g, 9h and 9i showing the standard deviation of Rec, UE and EV on the BSDS500 dataset. We can observe that in many cases good performing algorithms such as **ETPS**, **CRS**, **SLIC** or **ERS** also demonstrate low standard deviation. Oversegmentation algorithms, on the other hand, show higher standard deviation – together with algorithms such as **PF**, **TPS**, **VC**, **CIS** and **SEAW**. In this sense, stable algorithms can also be identified by low and monotonically decreasing standard deviation.

The variation in the number of generated superpixels is an important aspect for many superpixel algorithms. In particular, High standard deviation in the number of generated superpixels can be related to poor performance regarding Rec, UE and EV. We find that superpixel algorithms ensuring that the desired number of superpixels is met within appropriate bounds are preferrable. We consider Figures 9j and 9k, showing max K and std K for $K \approx 400$ on the BSDS500 dataset. Even after enforcing connectivity as described in Section 6.2, we observe that several implementations are not always able to meet the desired number of superpixels within acceptable bounds. Among these algorithms are **QS**, **VC**, **FH**, **CIS** and **LSC**. Except for the latter case, this can be related to poor performance with respect to Rec, UE and EV. Conversely, considering algorithms such as **ETPS**, **ERGC** or **ERS** which guarantee that the desired number of superpixels is met exactly, this can be related to good performance regarding these metrics. To draw similar conclusions for algorithms utilizing depth information, i.e. **DASP** and **VCCS**, the reader is encouraged to consider Figures 10j and 10k, showing max K and std K for $K \approx 400$ on the NYUV2 dataset. We can conclude that superpixel algorithms with low standard deviation in the number of generated superpixels are showing better performance in many cases.

Finally, we discuss the proposed metrics ARec, AUE and AEV (computed as the area above the Rec, (1 − UE) and EV curves within the interval $[K_{min}, K_{max}] = [200, 5200]$). We find that these metrics appropriately reflect and summarize the performance of superpixel algorithms independent of K. As can be seen in Figure 11a, showing ▮▮ ARec, ▮▮ AUE and ▮▮ AEV on the BSDS500 dataset, most of the previous observations can be confirmed. For example, we exemplarily consider **SEEDS** and observe low ARec and AEV which is confirmed by Figures 9a and 9c, showing Rec and EV on the BSDS500 dataset, where **SEEDS** consistently outperforms all algorithms except for **ETPS**. However, we can also observe higher AUE compared to algorithms such as **ETPS**, **ERS** or **CRS** wich is also consistent with Figure 9b, showing UE on the BSDS500 dataset. We conclude, that ARec, AUE and AEV give an easy-to-understand summary of algorithm performance. Furthermore, ARec, AUE and AEV can be used to rank the different algorithms according to the corresponding metrics; we will follow up on this idea in Section 7.7.

The observed ARec, AUE and AEV also properly reflect the difficulty of the different datasets. We consider Figure 11 showing ▮▮ ARec, ▮▮ AUE and ▮▮ AEV for all five datasets. Concentrating on **SEEDS** and **ETPS**, we see that the relative performance (i.e. the performance of **SEEDS** compared to **ETPS**) is consistent across datasets; **SEEDS** usually showing higher AUE while ARec and AEV are usually similar. Therefore, we observe that these metrics can be used to characterize the difficulty and ground truth of the datasets. For example, considering the Fash dataset, we observe very high AEV compared to the other datasets, while ARec and AUE are usually very low. This can be explained by the ground truth shown in Figure B.17e, i.e. the ground truth is limited to the foreground (in the case of Figure B.17e, the woman), leaving even complicated background unannotated. Similar arguments can be developed for the consistently lower ARec, AUE and AEV for the NYUV2 and SUNRGBD datasets compared to the BSDS500 dataset. For the SBD dataset, lower ARec, AUE and AEV can also be explained by the smaller average image size.

In conclusion, ARec, AUE and AEV accurately reflect the performance of superpixel algorithms and can be used to judge datasets. Across the different datasets, path-based and density-based algorithms perform poorly, while the remaining classes show mixed performance. However, some iterative energy optimization, clustering-based and graph-based algorithms such as **ETPS**, **SEEDS**, **CRS**, **ERS** and **SLIC** show favorable performance.

#### *7.2.1. Depth*

Depth information does not necessarily improve performance regarding Rec, UE and EV. We consider Figures 10a, 10b and 10c presenting Rec, UE and EV on the NYUV2 dataset. In particular, we consider **DASP** and **VCCS**. We observe, that **DASP** consistently outperforms **VCCS**. Therefore, we consider the performance of **DASP** and investigate whether depth information improves per-


Figure 11: ARec, AUE and AEV (lower is better) on the used datasets. We find that ARec, AUE and AEV appropriately summarize performance independent of the number of generated superpixels. Plausible examples to consider are top-performing algorithms such as **ETPS**, **ERS**, **SLIC** or **CRS** as well as poorly performing ones such as **QS** and **PF**. **Best viewed in color.**

formance. Note that **DASP** performs similar to **SLIC**, exhibiting slightly worse Rec and slightly better UE and EV for large K. However, **DASP** does not clearly outperform **SLIC**. As indicated in Section 3, **DASP** and **SLIC** are both clustering-based algorithms. In particular, both algorithms are based on $k$-means using color and spatial information and **DASP** additionally utilizes depth information. This suggests that the clustering approach does not benefit from depth information. We note that a similar line of thought can be applied to **VCCS** except that **VCCS** directly operates within a point cloud, rendering the comparison problematic. Still we conclude that depth information used in the form of **DASP** does not improve performance. However, we note that evaluation is carried out in the 2D image plain only.

### *7.3. Runtime*

Considering runtime, we report CPU time[3] excluding connected components but including color space conversions if applicable. We assured that no multi-threading or GPU computation were used. We begin by considering runtime in general, with a glimpse on realtime applications, before considering iterative algorithms.

We find that some well performing algorithms can be run at (near) realtime. We consider Figure 12 showing runtime in seconds $t$ on the BSDS500 (image size $481 \times 321$) and NYUV2 (image size $608 \times 448$) datasets. Concretely, considering the watershed-based algorithms **W** and **CW**, we can report runtimes below 10ms on both datasets, corresponding to roughly 100fps. Similarly, **PF** runs at below 10ms. Furthermore, several algorithms, such as **SLIC**, **ERGC**, **FH**, **PB**, **MSS** and **preSLIC** provide runtimes below 80ms and some of them are iterative, i.e. reducing the number of iterations may further reduce runtime. However, using the convention that realtime corresponds to roughly 30fps, this leaves **preSLIC** and **MSS** on the larger images of the NYUV2 dataset. However, even without explicit runtime requirements, we find runtimes below 1s per image to be beneficial for using superpixel algorithms as pre-processing tool, ruling out **TPS**, **CIS**, **SEAW**, **RW** and **NC**. Overall, several superpixel algorithms provide runtimes appropriate for pre-processing tools; realtime applications are still dependent on a few fast algorithms.

Iterative algorithms offer to reduce runtime while gradually lowering performance. Considering Figure 13, showing Rec, UE and runtime in seconds $t$ for all iterative algorithms on the BSDS500 dataset, we observe that the number of iterations can safely be reduced to decrease runtime while lowering Rec and increasing UE only slightly. In the best case, for example considering **ETPS**, reducing the number of iterations from 25 to 1 reduces the runtime from 680ms to 58ms, while keeping Rec und UE nearly

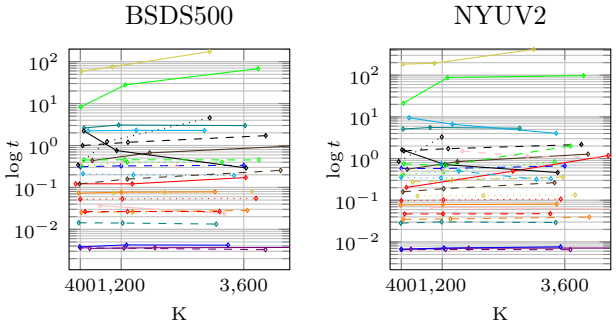



Figure 12: Runtime in seconds on the BSDS500 and NYUV2 datasets. Watershed-based, some clustering-based algorithms as well as **PF** offer runtimes below $100ms$. In the light of realtime applications, **CW**, **W** and **PF** even provide runtimes below $10ms$. However, independent of the application at hand, we find runtimes below $1s$ beneficial for using superpixel algorithms as pre-processing tools. **Best viewed in color.**

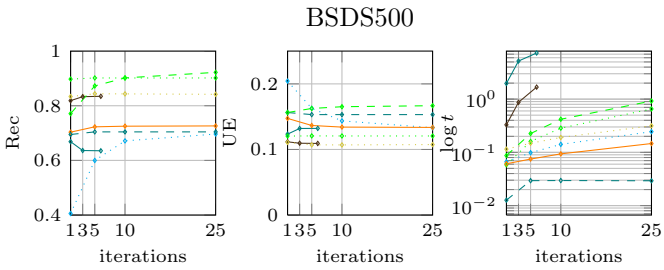
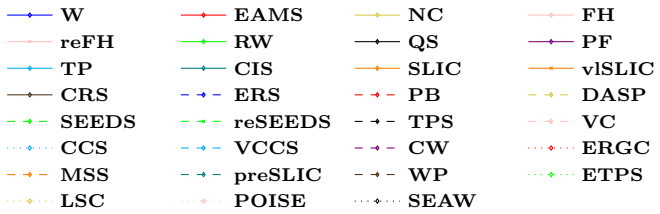



Figure 13: Rec, UE and runtime in seconds $t$ for iterative algorithms with K $\approx$ 400 on the BSDS500 dataset. Some algorithms allow to gradually trade performance for runtime, reducing runtime by several $100ms$ in some cases. **Best viewed in color.**




constant. For other cases, such as **SEEDS**, Rec decreases abruptly when using less than 5 iterations. Still, runtime can be reduced from 920ms to 220ms. For **CRS** and **CIS**, runtime reduction is similarly significant, but both algorithms still exhibit higher runtimes. If post-processing is necessary, for example for **SLIC** and **preSLIC**, the number of iterations has to be fixed in advance. However, for other iterative algorithms, the number of iterations may be adapted at runtime depending on the available time. Overall, iterative algorithms are beneficial as they are able to gradually trade performance for runtime.

We conclude that watershed-based as well as some path-based and clustering-based algorithms are candidates for realtime applications. Iterative algorithms, in particular many clustering-based and iterative energy optimization algorithms, may further be speeded up by reducing the number of iterations and trading performance for runtime. On a final note, we want to remind the reader that the

[3] Runtimes have been taken on an Intel® Core™ i7-3770 @ $3.4GHz$, 64bit with 32GB RAM.

image sizes of all used datasets may be relatively small compared to today's applications. However, the relative runtime comparison is still valuable.

### *7.4. Influence of Implementations*

We discuss the influence of implementation details on performance and runtime for different implementations of **SEEDS**, **SLIC** and **FH**: **reSEEDS** is a revised implementation of **SEEDS**, **vlSLIC** is an implementation of **SLIC** as part of the VLFeat library [100], and **reFH** is a revised implementation of **FH**. We also include **preSLIC**. In particular, **reSEEDS** and **reFH** use different data structures to represent superpixels and, thus, reach better connectivity. **vlSLIC**, in contrast, is very close to **SLIC**, but implemented in C instead of C++. Finally, **preSLIC** is based on **SLIC** while stopping the iterative clustering for each superpixel individually.

We find that revisiting implementation details may be beneficial for both performance and runtime. We consider Figure 14 showing Rec, UE and runtime in seconds $t$ for the introduced implementations of **SLIC**, **SEEDS** and **FH** on the BSDS500 dataset. For **reSEEDS** and **reFH**, we observe improved performance which can be related to the improved connectivity. However, even very similar implementations such as **SLIC** and **vlSLIC** differ slightly in performance; note the lower Rec and higher UE of **vlSLIC** compared to **SLIC**. Overall, the difference in runtime is most striking, for example **reSEEDS** and **preSLIC** show significantly lower runtime compared to **SEEDS** and **SLIC**. **reFH**, in contrast, shows higher runtime compared to **FH** due to a more complex data structure.

As expected, implementation details effect runtime, however, in the presented cases, i.e. for **SLIC**, **SEEDS** and **FH**, performance is also affected. Nevertheless, it still needs to be examined whether this holds true for the remaining algorithms, as well. Furthermore, the experiments suggest that improving connectivity helps performance.

### *7.5. Robustness*

Similar to Neubert and Protzel [35], we investigate the influence of noise, blur and affine transformations. We evaluated all algorithms for K $\approx$ 400 on the BSDS500 dataset. In the following we exemplarily discuss salt and pepper noise and average blurring.

Most algorithms are robust to salt and pepper noise; blurring, in contrast, tends to reduce performance. We consider Figure 15 showing Rec, UE and K for $p \in \{0, 0.04, 0.08, 0.12, 0.16\}$ being the probability of a pixel being salt or pepper. Note that Figure 15 shows the number of superpixels $K$ before enforcing connectivity as described in Section 6.2. As we can deduce, salt and pepper noise only slightly reduces Rec and UE for most algorithms. Some algorithms compensate the noise by generating more superpixels such as **VC** or **SEAW** while only slightly reducing performance. In contrast, for **QS** the performance

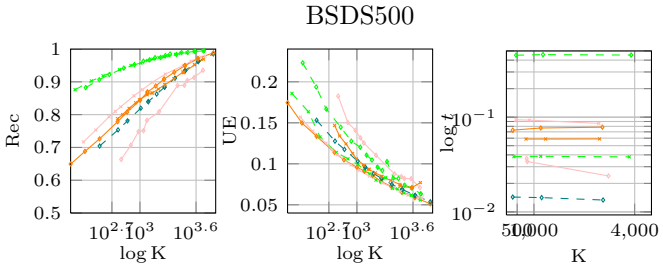



Figure 14: Rec, UE and runtime in seconds $t$ on the BSDS500 dataset for different implementations of **SLIC**, **SEEDS** and **FH**. In particular, **reSEEDS** and **reFH** show slightly better performance which may be explained by improved connectivity. **vlSLIC**, in contrast, shows similar performance to **SLIC** and, indeed, the implementations are very similar. Finally, **preSLIC** reduces runtime by reducing the number of iterations spend on individual superpixels. **Best viewed in color.**

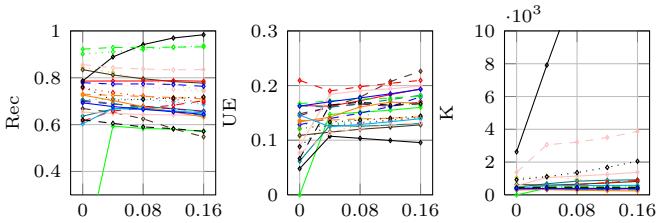



Figure 15: The influence of salt and pepper noise for $p \in \{0, 0.04, 0.08, 0.12, 0.16\}$ being the probability of salt or pepper. Regarding Rec and UE, most algorithms are not significantly influence by salt and pepper noise. Algorithms such as **QS** and **VC** compensate the noise by generating additional superpixels. **Best viewed in color.**

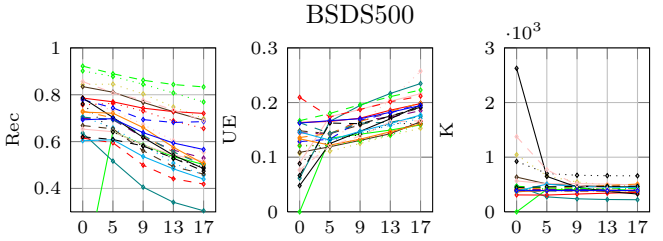
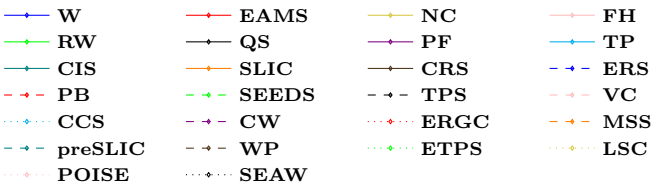



Figure 16: The influence of average blur for $k \in \{0, 5, 9, 13, 17\}$ being the filter size. As can be seen, blurring gradually reduces performance – which may be explained by vanishing image boundaries. In addition, for algorithms such as **VC** and **QS**, blurring also leads to fewer superpixels being generated. **Best viewed in color.**




even increases – a result of the strongly increasing number of superpixels. Similar results can be obtained for Gaussian additive noise. Turning to Figure 16 showing Rec, UE and K for $k \in \{0, 5, 9, 13, 17\}$ being the size of a box filter used for average blurring. As expected, blurring leads to

| | Rec | UE | EV | K | $t$ |
|---|---|---|---|---|---|
| **W** | 0.9998 | 0.0377 | 0.9238 | 20078.2 | 0.0090 |
| **PF** | 0.9512 | 0.0763 | 0.8834 | 15171.5 | 0.0105 |
| **CIS** | 0.9959 | 0.0353 | 0.9792 | 15988.6 | 7.617 |
| **SLIC** | 0.9997 | 0.0339 | 0.9570 | 17029.7 | 0.1633 |
| **CRS** | 0.9999 | 0.0265 | 0.9587 | 24615.8 | 4.7014 |
| **ERS** | 0.9999 | 0.0297 | 0.9564 | 20000 | 0.5935 |
| **PB** | 0.9988 | 0.0398 | 0.9477 | 15058.9 | 0.0531 |
| **SEEDS** | 0.9997 | 0.0340 | 0.9729 | 18982.5 | 0.2267 |
| **VC** | 0.9482 | 0.04621 | 0.9631 | 10487.5 | 2.3174 |
| **CCS** | 0.9999 | 0.0223 | 0.9637 | 17676.9 | 0.227 |
| **CW** | 0.9999 | 0.0322 | 0.9362 | 26319.8 | 0.0049 |
| **ERGC** | 0.9999 | 0.0316 | 0.9744 | 21312 | 0.1217 |
| **MSS** | 0.9989 | 0.0372 | 0.9171 | 25890.5 | 0.0894 |
| **preSLIC** | 0.9999 | 0.0359 | 0.958 | 17088.3 | 0.021 |
| **WP** | 0.9980 | 0.0411 | 0.9463 | 15502.7 | 0.6510 |
| **ETPS** | 0.9999 | 0.0311 | 0.9793 | 17227 | 1.1657 |

Table 2: Rec, UE, EV, K and runtime in seconds $t$ for K $\approx$ 20000 on the BSDS500 dataset including all algorithms able to generate K $\gg$ 5000 superpixels. The experiments demonstrate that nearly all superpixel algorithms are able to capture the image content without loss of information – with Rec $\approx 0.99$ and UE $\approx 0.03$ – while reducing the number of primitives from $481 \cdot 321 = 154401$ to K $\approx$ 20000.

reduced performance with respect to both Rec and UE. Furthermore, it leads to a reduced number of generated superpixels for algorithms such as **QS** or **VC**. Similar observations can be made for motion blur as well as Gaussian blur.

Overall, most superpixel algorithms are robust to the considered noise models, while blurring tends to reduce performance. Although the corresponding experiments are omitted for brevity, we found that affine transformations do not influence performance.

### *7.6. More Superpixels*

Up to now, we used ARec, AUE and AEV to summarize experimental results for K $\in [200, 5200]$. However, for some applications, generating K $\gg$ 5200 superpixels may be interesting.

For K $\approx$ 20000, superpixel algorithms can be used to dramatically reduce the number of primitives with negligible loss of information. We consider Table 2 presenting Rec, UE, EV, runtime in seconds $t$ and K for K $\approx 20,000$ on the BSDS500 dataset. We note that some algorithms were not able to generate K $\gg$ 5200 superpixels and are, therefore, excluded. Similarly, we excluded algorithms not offering control over the number of generated superpixels. We observe that except for **VC** and **PF** all algorithms achive Rec $\geq 0.99$, UE $\approx 0.3$, and EV $> 0.9$. Furthermore, the runtime of many algorithms is preserved. For example **W** and **CW** still run in below $10ms$ and the runtime for **preSLIC** and **SLIC** increases only slightly. Obviously, the number of generated superpixels varies more strongly for large K. Overall, most algorithms are able to capture the image content nearly perfectly while reducing the number of primitives from $321 \times 481 = 154401$ to K $\approx$ 20000.

### *7.7. Ranking*

We conclude the experimental part of this paper with a ranking with respect to ARec and AUE– reflecting the objective used for parameter optimization. Unfortunately, the high number of algorithms as well as the low number of datasets prohibits using statistical tests to extract rankings, as done in other benchmarks (e.g. [101, 102]). Therefore, Table 3 presents the computed average ranks and the corresponding rank matrix. We find that the presented average ranks provide a founded overview of the evaluated algorithms, summarizing many of the observations discussed before. In the absense of additional constraints, Table 3 may be used to select suitable superpixel algorithms.

## 8. Conclusion

In this paper, we presented a large-scale comparison of superpixel algorithms taking into account visual quality, ground truth dependent and independent metrics, runtime, implementation details as well as robustness to noise, blur and affine transformations. For fairness, we systematically optimized parameters while strictly enforcing connectivity. Based on the obtained parameters, we presented experiments based on five different datasets including indoor and outdoor scenes as well as persons. In contrast to existing work [34, 30, 35, 37], we considered minimum/maximum as well as the standard deviation in addition to simple metric averages. We further proposed Average Recall, Average Undersegmentation Error and Average Explained Variation to summarize algorithm performance independent of the number of generated superpixels. This enabled us to present an overall ranking of superpixel algorithms aimed to simplify and guide algorithm selection.

Regarding the mentioned aspects of superpixel algorithms, we made several observations relevant for applications and future research. Considering visual quality, we found that the majority of algorithms provides solid boundary adherence; some algorithms are able to capture even small details. However, better boundary adherence may influence compactness, regularity and smoothness. While regularity and smoothness strongly depends on the individual algorithms, a compactness parameter is beneficial to trade-off boundary adherence for compactness. Regarding performance, Boundary Recall [39], Undersegmentation Error [33, 30, 35] and Explained Variation [40] provide a solid overview but are not sufficient to discriminate algorithms reliably. Therefore, we used the minimum/maximum as well as the standard deviation of these metrics to identify stable algorithms, i.e. algoritms providing monotonically increasing performance with regard to the number of generated superpixels. Furthermore, we were able to relate poor performance to a high standard deviation in the number of generated superpixels, justifying the need to strictly control connectivity. Concerning runtime, we identified several algorithms providing realtime

| ARec | AUE | | Rank | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2.05 | 6.07 | **ETPS** | 1 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1.57 | 7.98 | **SEEDS** | 3.8 | 0 | 2 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3.38 | 6.28 | **ERS** | 3.8 | 0 | 1 | 1 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4.16 | 6.29 | **CRS** | 4.8 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4.18 | 6.77 | **EAMS** | 5.4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4.55 | 6.34 | **ERGC** | 5.8 | 0 | 0 | 0 | 0 | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4.91 | 6.50 | **SLIC** | 6.2 | 0 | 0 | 0 | 0 | 2 | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5.40 | 7.24 | **LSC** | 9.2 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5.49 | 7.02 | **preSLIC** | 9.2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6.60 | 6.41 | **CCS** | 10.6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6.49 | 7.19 | **CW** | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8.24 | 7.83 | **DASP** | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6.51 | 7.44 | **W** | 12.8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7.44 | 6.97 | **WP** | 13.4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7.85 | 7.58 | **POISE** | 13.8 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 8.13 | 6.90 | **NC** | 15.25 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7.66 | 7.43 | **MSS** | 15.8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8.90 | 7.67 | **VC** | 17.8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 8.31 | 7.85 | **PB** | 18.4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8.53 | 8.15 | **FH** | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8.35 | 7.32 | **RW** | 19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 11.45 | 6.70 | **CIS** | 20.8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11.45 | 7.19 | **TPS** | 21 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11.05 | 7.93 | **TP** | 21.6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 14.22 | 13.54 | **VCCS** | 23 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 11.97 | 8.36 | **SEAW** | 24 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | 1 | 0 | 0 |
| 15.72 | 10.75 | **QS** | 24.8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 0 | 1 | 0 |
| 24.64 | 14.91 | **PF** | 26.2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 0 | 1 |

Table 3: Average ranks and rank matrix of all evaluated algorithms. We note that we could not evaluate **RW**, **NC** and **SEAW** on the SUNRGBD dataset and **DASP** and **VCCS** cannot be evaluated on the BSDS500, SBD and Fash datasets. The ranking considers both ARec and AUE to reflect the objective used for parameter optimization.

capabilities, i.e. roughly 30fps, and showed that iterative algorithms allow to reduce runtime while only gradually reducing performance. Implementation details are rarely discussed in the literature; on three examples, we highlighted the advantage of ensuring connectivity and showed that revisiting implementations may benefit performance and runtime. We further demonstrated that generating a higher number of superpixels, e.g. roughly 20000, results in nearly no loss of information while still greatly reducing the number of primitives. Finally, we experimentally argued that superpixel algorithms are robust against noise and affine transformations before providing a final ranking of the algorithms based on the proposed metrics Average Recall and Average Undersegmentation Error.

From the ranking in Table 3, we recommend 6 algorithms for use in practice, thereby covering a wide range of application scenarios: **ETPS** [74], **SEEDS** [71], **ERS** [38], **CRS** [69, 70], **ERGC** [59] and **SLIC** [65]. These algorithms show superior performance regarding Boundary Recall, Undersegmentation Error and Explained Variation and can be considered stable. Furthermore, they are iterative (except for **ERGC** and **ERS**) and provide a compactness parameter (except for **SEEDS** and **ERS**). Except for **ERS** and **CRS**, they provide runtimes below 100ms – depending on the implementation – and **preSLIC** [48], which we see as a variant of **SLIC**, provides realtime capabilities. Finally, the algorithms provide control over the number of generated superpixels (therefore, **EAMS**, ranked 5th in Table 3, is not recommended), are able to generate mostly connected superpixels and exhibit a very low standard deviation in the number of generated superpixels.

**Software.** The individual implementations, together with the used benchmark, are made publicly available at `davidstutz.de/projects/superpixel-benchmark/`.

**Acknowledgements.** The work in this paper was funded by the EU project STRANDS (ICT-2011-600623). We are also grateful for the implementations provided by many authors.

## References

## References

[1] X. Ren, J. Malik, Learning a classification model for segmentation, in: International Conference on Computer Vision, 2003, pp. 10–17.

[2] R. Mester, U. Franke, Statistical model based image segmentation using region growing, contour relaxation and classification, in: SPIE Symposium on Visual Communications and Image Processing, 1988, pp. 616–624.

[3] B. Marcotegui, F. Meyer, Bottom-up segmentation of image sequences for coding, Annales des Télécommunications 52 (7) (1997) 397–407.

[4] D. Hoiem, A. A. Efros, M. Hebert, Automatic photo pop-up, ACM Transactions on Graphics 24 (3) (2005) 577–584.

[5] D. Hoiem, A. N. Stein, A. A. Efros, M. Hebert, Recovering occlusion boundaries from a single image, in: International Conference on Computer Vision, 2007, pp. 1–8.

[6] P. F. Felzenswalb, D. P. Huttenlocher, Efficient graph-based image segmentation, International Journal of Computer Vision 59 (2) (2004) 167–181.
[7] F. Meyer, Color image segmentation, in: International Conference on Image Processing and its Applications, 1992, pp. 303–306.
[8] S. Gould, J. Rodgers, D. Cohen, G. Elidan, D. Koller, Multi-class segmentation with relative location prior, International Journal of Computer Vision 80 (3) (2008) 300–316.
[9] J. Tighe, S. Lazebnik, SuperParsing: Scalable nonparametric image parsing with superpixels, in: European Conference on Computer Vision, 2010, pp. 352–365.
[10] S. Wang, H. Lu, F. Yang, M.-H. Yang, Superpixel tracking, in: International Conference on Computer Vision, 2011, pp. 1323–1330.
[11] F. Yang, H. Lu, M.-H. Yang, Robust superpixel tracking, Transactions on Image Processing 23 (4) (2014) 1639–1651.
[12] Y. Zhang, R. Hartley, J. Mashford, S. Burn, Superpixels, occlusion and stereo, in: International Conference on Digital Image Computing Techniques and Applications, 2011, pp. 84–91.
[13] K. Yamaguchi, D. A. McAllester, R. Urtasun, Efficient joint segmentation, occlusion labeling, stereo and flow estimation, in: European Conference on Computer Vision, 2014, pp. 756–771.
[14] A. Bódis-Szomorú, H. Riemenschneider, L. van Gool, Superpixel meshes for fast edge-preserving surface reconstruction, in: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2011–2020.
[15] F. Perazzi, P. Krähenbühl, Y. Pritch, A. Hornung, Saliency filters: Contrast based filtering for salient region detection, in: IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 733–740.
[16] G. Shu, A. Dehghan, M. Shah, Improving an object detector and extracting regions using superpixels, in: IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 3721–3727.
[17] J. Yan, Y. Yu, X. Zhu, Z. Lei, S. Z. Li, Object detection by labeling superpixels, in: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5107–5116.
[18] M. V. den Bergh, D. Carton, L. van Gool, Depth SEEDS: recovering incomplete depth data using superpixels, in: Winter Conference on Applications of Computer Vision, 2013, pp. 363–368.
[19] F. Liu, C. Shen, G. Lin, Deep convolutional neural fields for depth estimation from a single image, Computing Research Repository abs/1411.6387.
[20] A. Geiger, C. Wang, Joint 3d object and layout inference from a single RGB-D image, in: German Conference on Pattern Recognition, 2015, pp. 183–195.
[21] M. Menze, A. Geiger, Object scene flow for autonomous vehicles, in: IEEE Conference on Computer Vision and Pattern Recognition, Boston, Massachusetts, 2015, pp. 3061–3070.
[22] R. Gadde, V. Jampani, M. Kiefel, P. V. Gehler, Superpixel convolutional networks using bilateral inceptions, Computing Research Repository abs/1511.06739.
[23] B. Andres, U. Köthe, M. Helmstaedter, W. Denk, F. A. Hamprecht, Segmentation of SBFSEM volume data of neural tissue by hierarchical classification, in: DAGM Annual Pattern Recognition Symposium, 2008, pp. 142–152.
[24] A. Lucchi, K. Smith, R. Achanta, V. Lepetit, P. Fua, A fully automated approach to segmentation of irregularly shaped cellular structures in EM images, in: International Conference onMedical Image Computing and Computer Assisted Interventions, 2010, pp. 463–471.
[25] A. Lucchi, K. Smith, R. Achanta, G. Knott, P. Fua, Supervoxel-based segmentation of mitochondria in EM image stacks with learned shape features, IEEE Transactions on Medical Imaging 31 (2) (2012) 474–486.
[26] S. Haas, R. Donner, A. Burner, M. Holzer, G. Langs, Superpixel-based interest points for effective bags of visual words medical image retrieval, in: International Conference onMedical Image Computing and Computer Assisted Interventions, 2011, pp. 58–68.
[27] K. Yamaguchi, K. M. H, L. E. Ortiz, T. L. Berg, Parsing clothing in fashion photographs, in: IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 3570–3577.
[28] S. Liu, J. Feng, C. Domokos, H. Xu, J. Huang, Z. Hu, S. Yan, Fashion parsing with weak color-category labels, IEEE Transactions on Multimedia 16 (1) (2014) 253–265.
[29] M. van den Bergh, G. Roig, X. Boix, S. Manen, L. van Gool, Online video seeds for temporal window objectness, in: International Conference on Computer Vision, 2013, pp. 377–384.
[30] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, SLIC superpixels compared to state-of-the-art superpixel methods, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (11) (2012) 2274–2281.
[31] M. Grundmann, V. Kwatra, M. Han, I. A. Essa, Efficient hierarchical graph-based video segmentation, in: IEEE Conference on Computer Vision and Pattern Recognition, 2010, pp. 2141–2148.
[32] C. Xu, J. J. Corso, Evaluation of super-voxel methods for early video processing, in: IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 1202–1209.
[33] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, K. Siddiqi, TurboPixels: Fast superpixels using geometric flows, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (12) (2009) 2290–2297.
[34] A. Schick, M. Fischer, R. Stiefelhagen, Measuring and evaluating the compactness of superpixels, in: International Conference on Pattern Recognition, 2012, pp. 930–934.
[35] P. Neubert, P. Protzel, Superpixel benchmark and comparison, in: Forum Bildverarbeitung, 2012.
[36] O. Veksler, Y. Boykov, P. Mehrani, Superpixels and supervoxels in an energy optimization framework, in: European Conference on Computer Vision, Vol. 6315, 2010, pp. 211–224.
[37] P. Neubert, P. Protzel, Evaluating superpixels in video: Metrics beyond figure-ground segmentation, in: British Machine Vision Conference, 2013.
[38] M. Y. Lui, O. Tuzel, S. Ramalingam, R. Chellappa, Entropy rate superpixel segmentation, in: IEEE Conference on Computer Vision and Pattern Recognition, 2011, pp. 2097–2104.
[39] D. Martin, C. Fowlkes, J. Malik, Learning to detect natural image boundaries using local brightness, color, and texture cues, IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (5) (2004) 530–549.
[40] A. P. Moore, S. J. D. Prince, J. Warrell, U. Mohammed, G. Jones, Superpixel lattices, in: IEEE Conference on Computer Vision and Pattern Recognition, 2008, pp. 1–8.
[41] D. J. Butler, J. Wulff, G. B. Stanley, M. J. Black, A naturalistic open source movie for optical flow evaluation, in: European Conference on Computer Vision, 2012, pp. 611–625.
[42] J. M. Gonfaus, X. B. Bosch, J. van de Weijer, A. D. Bagdanov, J. Serrat, J. Gonzàlez, Harmony potentials for joint classification and segmentation, in: IEEE Conference on Computer Vision and Pattern Recognition, 2010, pp. 3280–3287.
[43] J. Strassburg, R. Grzeszick, L. Rothacker, G. A. Fink, On the influence of superpixel methods for image parsing, in: International Conference on Computer Vision Theory and Application, 2015, pp. 518–527.
[44] D. Weikersdorfer, D. Gossow, M. Beetz, Depth-adaptive superpixels, in: International Conference on Pattern Recognition, 2012, pp. 2087–2090.
[45] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 22 (8) (2000) 888–905.
[46] P. Arbeláez, M. Maire, C. Fowlkes, J. Malik, Contour detection and hierarchical image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (5) (2011) 898–916.
[47] P. Koniusz, K. Mikolajczyk, Segmentation based interest points and evaluation of unsupervised image segmentation methods, in: British Machine Vision Conference, 2009, pp.

1–11.
[48] P. Neubert, P. Protzel, Compact watershed and preemptive SLIC: on improving trade-offs of superpixel segmentation algorithms, in: International Conference on Pattern Recognition, 2014, pp. 996–1001.
[49] W. Benesova, M. Kottman, Fast superpixel segmentation using morphological processing, in: Conference on Machine Vision and Machine Learning, 2014.
[50] V. Machairas, E. Decencière, T. Walter, Waterpixels: Superpixels based on the watershed transformation, in: International Conference on Image Processing, 2014, pp. 4343–4347.
[51] V. Machairas, M. Faessel, D. Cardenas-Pena, T. Chabardes, T. Walter, E. Decencière, Waterpixels, Transactions on Image Processing 24 (11) (2015) 3707–3716.
[52] D. Comaniciu, P. Meer, Mean shift: A robust approach toward feature space analysis, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (5) (2002) 603–619.
[53] A. Vedaldi, S. Soatto, Quick shift and kernel methods for mode seeking, in: European Conference on Computer Vision, Vol. 5305, 2008, pp. 705–718.
[54] P. Carr, R. I. Hartley, Minimizing energy functions on 4-connected lattices using elimination, in: International Conference on Computer Vision, 2009, pp. 2042–2049.
[55] L. Grady, G. Funka-Lea, Multi-label image segmentation for medical applications based on graph-theoretic electrical potentials, in: ECCV Workshops on Computer Vision and Mathematical Methods in Medical and Biomedical Image Analysis, 2004, pp. 230–245.
[56] L. Grady, Random walks for image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (11) (2006) 1768–1783.
[57] Y. Zhang, R. Hartley, J. Mashford, S. Burn, Superpixels via pseudo-boolean optimization, in: International Conference on Computer Vision, 2011, pp. 1387–1394.
[58] J. M. R. A. Humayun, F. Li, The middle child problem: Revisiting parametric min-cut and seeds for object proposals, in: International Conference on Computer Vision, 2015, pp. 1600 – 1608.
[59] P. Buyssens, I. Gardin, S. Ruan, Eikonal based region growing for superpixels generation: Application to semi-supervised real time organ segmentation in CT images, Innovation and Research in BioMedical Engineering 35 (1) (2014) 20–26.
[60] P. Buyssens, M. Toutain, A. Elmoataz, O. Lézoray, Eikonal-based vertices growing and iterative seeding for efficient graph-based segmentation, in: International Conference on Image Processing, 2014, pp. 4368–4372.
[61] P. Dollár, C. L. Zitnick, Structured forests for fast edge detection, in: International Conference on Computer Vision, 2013, pp. 1841–1848.
[62] F. Drucker, J. MacCormick, Fast superpixels for video analysis, in: Workshop on Motion and Video Computing, 2009, pp. 1–8.
[63] D. Tang, H. Fu, X. Cao, Topology preserved regular superpixel, in: IEEE International Conference on Multimedia and Expo, 2012, pp. 765–768.
[64] H. Fu, X. Cao, D. Tang, Y. Han, D. Xu, Regularity preserved superpixels and supervoxels, IEEE Transactions on Multimedia 16 (4) (2014) 1165–1175.
[65] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, SLIC superpixels, Tech. rep., École Polytechnique Fédérale de Lausanne (2010).
[66] J. Wang, X. Wang, VCells: Simple and efficient superpixels using edge-weighted centroidal voronoi tessellations, IEEE Transactions on Pattern Analysis and Machine Intelligence 34 (6) (2012) 1241–1247.
[67] J. Papon, A. Abramov, M. Schoeler, F. Wörgötter, Voxel cloud connectivity segmentation - supervoxels for point clouds, in: IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 2027–2034.
[68] Z. Li, J. Chen, Superpixel segmentation using linear spectral clustering, in: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1356–1363.
[69] R. Mester, C. Conrad, A. Guevara, Multichannel segmentation using contour relaxation: Fast super-pixels and temporal propagation, in: Scandinavian Conference Image Analysis, 2011, pp. 250–261.
[70] C. Conrad, M. Mertz, R. Mester, Contour-relaxed superpixels, in: Energy Minimization Methods in Computer Vision and Pattern Recognition, 2013, pp. 280–293.
[71] M. van den Bergh, X. Boix, G. Roig, B. de Capitani, L. van Gool, SEEDS: Superpixels extracted via energy-driven sampling, in: European Conference on Computer Vision, Vol. 7578, 2012, pp. 13–26.
[72] H. E. Tasli, C. Çigla, T. Gevers, A. A. Alatan, Super pixel extraction via convexity induced boundary adaptation, in: IEEE International Conference on Multimedia and Expo, 2013, pp. 1–6.
[73] H. E. Tasli, C. Cigla, A. A. Alatan, Convexity constrained efficient superpixel and supervoxel extraction, Signal Processing: Image Communication 33 (2015) 71–85.
[74] J. Yao, M. Boben, S. Fidler, R. Urtasun, Real-time coarse-to-fine topologically preserving segmentation, in: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2947–2955.
[75] C. Rohkohl, K. Engel, Efficient image segmentation using pairwise pixel similarities, in: DAGM Annual Pattern Recognition Symposium, 2007, pp. 254–263.
[76] D. Engel, L. Spinello, R. Triebel, R. Siegwart, H. H. Bülthoff, C. Curio, Medial features for superpixel segmentation, in: IAPR International Conference on Machine Vision Applications, 2009, pp. 248–252.
[77] A. Ducournau, S. Rital, A. Bretto, B. Laget, Hypergraph coarsening for image superpixelization, in: International Symposium on I/V Communications and Mobile Network, 2010, pp. 1–4.
[78] G. Zeng, P. Wang, J. Wang, R. Gan, H. Zha, Structure-sensitive superpixels via geodesic distance, in: International Conference on Computer Vision, 2011, pp. 447–454.
[79] F. Perbet, A. Maki, Homogeneous superpixels from random walks, in: Machine Vision and Applications, Conference on, 2011, pp. 26–30.
[80] Y. Du, J. Shen, X. Yu, D. Wang, Superpixels using random walker, in: IEEE Global High Tech Congress on Electronics, 2012, pp. 62–63.
[81] J. Yang, Z. Gan, X. Gui, K. Li, C. Hou, 3-d geometry enhanced superpixels for RGB-D data, in: Pacific-Rim Conference on Multimedia, 2013, pp. 35–46.
[82] Z. Ren, G. Shakhnarovich, Image segmentation by cascaded region agglomeration, in: IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 2011–2018.
[83] K.-S. Kim, D. Zhang, M.-C. Kang, S.-J. Ko, Improved simple linear iterative clustering superpixels, in: International Symposium on Consumer Electronics, 2013, pp. 259–260.
[84] J. Shen, Y. Du, W. Wang, X. Li, Lazy random walks for superpixel segmentation, Transactions on Image Processing 23 (4) (2014) 1451–1462.
[85] P. Morerio, L. Marcenaro, C. S. Regazzoni, A generative superpixel method, in: International Conference on Information Fusion, 2014, pp. 1–7.
[86] P. Siva, A. Wong, Grid seams: A fast superpixel algorithm for real-time applications, in: Computer and Robot Vision, Conference on, 2014, pp. 127–134.
[87] S. R. S. P. Malladi, S. Ram, J. J. Rodriguez, Superpixels using morphology for rock image segmentation, in: IEEE Southwest Symposium on Image Analysis and Interpretation, 2014, pp. 145–148.
[88] O. Freifeld, Y. Li, J. W. Fisher, A fast method for inferring high-quality simply-connected superpixels, in: International Conference on Image Processing, 2015, pp. 2184–2188.
[89] J. Lv, An improved slic superpixels using reciprocal nearest neighbor clustering, International Journal of Signal Processing, Image Processing and Pattern Recognition 8 (5) (2015) 239–248.

[90] S. Gould, R. Fulton, D. Koller, Decomposing a scene into geometric and semantically consistent regions, in: International Conference on Computer Vision, 2009, pp. 1–8.
[91] B. C. Russell, A. Torralba, K. P. Murphy, W. T. Freeman, Labelme: A database and web-based tool for image annotation, International Journal of Computer Vision 77 (1-3) (2008) 157–173.
[92] A. Criminisi, Microsoft research cambridge object recognition image database, http://research.microsoft.com/en-us/projects/objectclassrecognition/ (2004).
[93] L. M. Everingham, L. van Gool, C. K. I. Williams, J. Winn, A. Zisserman, The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results, http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.
[94] D. Hoiem, A. A. Efros, M. Hebert, Recovering surface layout from an image, International Journal of Computer Vision 75 (1) (2007) 151–172.
[95] N. Silberman, D. Hoiem, P. Kohli, R. Fergus, Indoor segmentation and support inference from RGBD images, in: European Conference on Computer Vision, 2012, pp. 746–760.
[96] X. Ren, L. Bo, Discriminatively trained sparse code gradients for contour detection, in: Neural Information Processing Systems, 2012, pp. 593–601.
[97] S. Song, S. P. Lichtenberg, J. Xiao, SUN RGB-D: A RGB-D scene understanding benchmark suite, in: IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 567–576.
[98] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, T. Darrell, A category-level 3-d object dataset: Putting the kinect to work, in: International Conference on Computer Vision Workshops, 2011, pp. 1168–1174.
[99] J. Xiao, A. Owens, A. Torralba, SUN3D: A database of big spaces reconstructed using sfm and object labels, in: International Conference on Computer Vision, 2013, pp. 1625–1632.
[100] A. Vedaldi, B. Fulkerson, VLFeat: An open and portable library of computer vision algorithms, http://www.vlfeat.org/ (2008).
[101] J. Demsar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine Learning Research 7 (2006) 1–30.
[102] P. Dollár, C. Wojek, B. Schiele, P. Perona, Pedestrian detection: A benchmark, in: IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 304–311.

## Appendix A. Algorithms

Complementing the information presented in Section 3, Table A.4 gives a complete overview of all algorithms.

## Appendix B. Datasets

The BSDS500 dataset is the only dataset providing several ground truth segmentations per image. Therefore, we briefly discuss evaluation on the BSDS500 dataset in detail. Furthermore, additional example images from all used datasets are shown in Figure B.17 and are used for qualitative results in Appendix E.1.

Assuming at least two ground truth segmentations per image, Arbeláez et al. [46] consider two methodologies of computing average Rec: computing the average Rec over all ground truth segmentations per image, and subsequently taking the worst average (i.e. the lowest Rec); or taking the lowest Rec over all ground truth segmentations per image and averaging these. We follow Arbeláez et al. and pick the latter approach. The same methodology is then applied to UE, EV, ASA and $\mathrm{UE}_{\mathrm{Levin}}$ . In this sense, we never overestimate performance with respect to the provided ground truth segmentations.

## Appendix C. Benchmark

In the following section we discuss the expressiveness of the used metrics and details regarding the computation of ARec, AUE and AEV.

### *Appendix C.1. Expressiveness and Correlation*

Complementing Section 5.1, we exemplarily discuss the correlation computed for **SEEDS** with $\mathrm{K} \approx 400$ on the BSDS500 dataset as shown in Table C.5. We note that the following observations can be confirmed when considering different algorithms. Still, **SEEDS** has the advantage of showing good overall performance (see the ranking in Section 7.7) and low standard deviation in Rec, UE and EV. We observe a correlation of $-0.47$ between Rec and UE reflecting that **SEEDS** exhibits high Rec but comparably lower UE on the BSDS500 dataset. This justifies the choice of using both Rec and UE for quantitative comparison. The correlation of UE and ASA is 1, which we explained with the respective definitions. More interestingly, the correlation of UE and $\mathrm{UE}_{\mathrm{Levin}}$ is $-0.7$. Therefore, we decided to discuss results regarding $\mathrm{UE}_{\mathrm{Levin}}$ in more detail in Appendix E. Nevertheless, this may also confirm the observations by Neubert and Protzel [35] as well as Achanta et al. [30] that $\mathrm{UE}_{\mathrm{Levin}}$ unjustly penalizes large superpixels. The high correlation of $-0.97$ between MDE and Rec has also been explained using the respective definitions. Interestingly, the correlation decreases with rising K. This can, however, be explained by the implementation of Rec, allowing a fixed tolerance of 0.0025 times the image diagonal. The correlation of $-0.42$ between ICV and EV was explained by the missing normalization of ICV compared to EV. This observation is confirmed by the decreasing correlation for larger K as, on average, superpixels get smaller thereby reducing the influence of normalization.

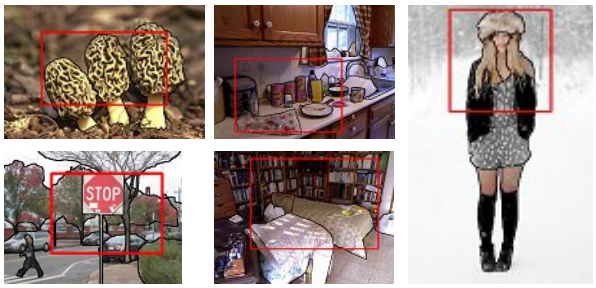


Figure B.17: Example images from the used datasets. From left ro right: BSDS500; SBD; NYUV2; SUNRGBD; and Fash. Black contours represent ground truth and red rectangles indicate excerpts used for qualitative comparison in Figures E.18 and E.19. **Best viewed in color.**

| K = 400 | Rec | UE | $\mathrm{UE}_{\mathrm{Levin}}$ | ASA | CO | EV | MDE | ICV |
|---|---|---|---|---|---|---|---|---|
| Rec | 1 | -0.47 | -0.07 | 0.46 | -0.08 | 0.11 | -0.97 | -0.04 |
| UE | -0.47 | 1 | 0.11 | -1 | 0.07 | -0.19 | 0.47 | 0.25 |
| $\mathrm{UE}_{\mathrm{Levin}}$ | -0.07 | 0.11 | 1 | -0.1 | -0.01 | -0.03 | 0.09 | 0.05 |
| ASA | 0.46 | -1 | -0.1 | 1 | -0.07 | 0.18 | -0.47 | -0.24 |
| CO | -0.08 | 0.07 | -0.01 | -0.07 | 1 | 0.34 | 0.06 | -0.05 |
| EV | 0.11 | -0.19 | -0.03 | 0.18 | 0.34 | 1 | -0.16 | -0.42 |
| MDE | -0.97 | 0.47 | 0.09 | -0.47 | 0.06 | -0.16 | 1 | 0.06 |
| ICV | -0.04 | 0.25 | 0.05 | -0.24 | -0.05 | -0.42 | 0.06 | 1 |
| CD | 0.05 | 0 | 0 | 0 | -0.86 | -0.45 | -0.03 | 0.21 |

| K = 400 | Rec | UE | $\mathrm{UE}_{\mathrm{Levin}}$ | ASA | CO | EV | MDE | ICV |
|---|---|---|---|---|---|---|---|---|
| Rec | 1 | -0.4 | -0.06 | 0.4 | -0.11 | 0.16 | -0.92 | -0.09 |
| UE | -0.4 | 1 | 0.14 | -1 | -0.01 | -0.22 | 0.45 | 0.27 |
| $\mathrm{UE}_{\mathrm{Levin}}$ | -0.06 | 0.14 | 1 | -0.14 | -0.07 | -0.07 | 0.08 | 0.09 |
| ASA | 0.4 | -1 | -0.14 | 1 | 0.01 | 0.21 | -0.45 | -0.27 |
| CO | -0.11 | -0.01 | -0.07 | 0.01 | 1 | 0.24 | 0.12 | -0.15 |
| EV | 0.16 | -0.22 | -0.07 | 0.21 | 0.24 | 1 | -0.3 | -0.52 |
| MDE | -0.92 | 0.45 | 0.08 | -0.45 | 0.12 | -0.3 | 1 | 0.15 |
| ICV | -0.09 | 0.27 | 0.09 | -0.27 | -0.15 | -0.52 | 0.15 | 1 |
| CD | 0.13 | 0 | 0.03 | -0.01 | -0.91 | -0.31 | -0.13 | 0.15 |

| K = 400 | Rec | UE | $\mathrm{UE}_{\mathrm{Levin}}$ | ASA | CO | EV | MDE | ICV |
|---|---|---|---|---|---|---|---|---|
| Rec | 1 | -0.26 | -0.05 | 0.28 | -0.08 | 0.12 | -0.66 | -0.1 |
| UE | -0.26 | 1 | 0.16 | -1 | 0.02 | -0.2 | 0.41 | 0.28 |
| $\mathrm{UE}_{\mathrm{Levin}}$ | -0.05 | 0.16 | 1 | -0.16 | -0.02 | -0.07 | 0.1 | 0.07 |
| ASA | 0.28 | -1 | -0.16 | 1 | -0.02 | 0.2 | -0.41 | -0.28 |
| CO | -0.08 | 0.02 | -0.02 | -0.02 | 1 | 0.19 | 0.13 | -0.17 |
| EV | 0.12 | -0.2 | -0.07 | 0.2 | 0.19 | 1 | -0.36 | -0.61 |
| MDE | -0.66 | 0.41 | 0.1 | -0.41 | 0.13 | -0.36 | 1 | 0.22 |
| ICV | -0.1 | 0.28 | 0.07 | -0.28 | -0.17 | -0.61 | 0.22 | 1 |

Table C.5: Pearson correlation coefficient of all discussed metrics exemplarily shown for **SEEDS** with $\mathrm{K} \approx 400$, $\mathrm{K} \approx 1200$ and $\mathrm{K} \approx 3600$.

### *Appendix C.2. Average Recall, Average Undersegmentation Error and Average Explained Variation*

As introduced in Section 5.2, ARec, AUE and AEV are intended to summarize algorithm performance independent of K. To this end, we compute the area above the Rec, $(1-\mathrm{UE})$ and EV curves within the interval $[\mathrm{K}_{\mathrm{min}}, \mathrm{K}_{\mathrm{max}}] = [200, 5200]$. In particular, we use the trapezoidal rule for integration. As the algorithms do not necessarily generate

| | Reference | Year | Citations | Categorization | Implementation | GRAY | RGB | Lab | Luv | HSV | YCrCb | #Parameters | #Superpixels | #Iterations | Compactness | Depth | Edges | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **W** | [7] | 1992 | 234 | watershed | C/C++ | – | ☑ | ✓ | ✓ | ✓ | ✓ | 1 | ✓ | – | – | – | – | |
| **EAMS** | [52] | 2002 | 9631 | density | MatLab/C | – | ☑ | – | ✓ | – | – | 2 | – | – | – | – | ✓ | |
| **NC** | [1] | 2003 | 996 | graph | MatLab/C | – | ☑ | – | – | – | – | 3 | ✓ | – | – | – | – | |
| **FH** | [6] | 2004 | 4144 | graph | C/C++ | – | ☑ | – | – | – | – | 3 | – | – | – | – | – | |
| **reFH** | ———"——— | | | ———"——— | C/C++ | – | ☑ | – | – | – | – | 3 | – | – | – | – | – | |
| **RW** | [55, 56] | 2004 | 189 + 1587 | graph | MatLab/C | – | ☑ | – | – | – | – | 2 | ✓ | – | – | – | – | |
| **QS** | [53] | 2008 | 376 | density | MatLab/C | – | ✓ | ☑ | – | – | – | 3 | – | – | – | – | – | |
| **PF** | [62] | 2009 | 18 | path | Java | – | ☑ | – | – | – | – | 2 | ✓ | – | – | – | – | |
| **TP** | [33] | 2009 | 559 | contour evolution | MatLab/C | – | ☑ | – | – | – | – | 4 | ✓ | – | – | – | – | |
| **CIS** | [36] | 2010 | 223 | graph | C/C++ | ☑ | ✓ | – | – | – | – | 4 | ✓ | ✓ | – | – | – | |
| **SLIC** | [65, 30] | 2010 | 438 + 1843 | clustering | C/C++ | – | ✓ | ☑ | – | – | – | 4 | ✓ | ✓ | ✓ | – | – | |
| **vlSLIC** | ———"——— | | | ———"——— | C/C++ | – | ☑ | – | – | – | – | 4 | ✓ | ✓ | ✓ | – | – | |
| **CRS** | [69, 70] | 2011 | 14 + 4 | energy optimization | C/C++ | ✓ | ✓ | – | – | – | ☑ | 4 | ✓ | ✓ | ✓ | – | – | |
| **ERS** | [38] | 2011 | 216 | graph | C/C++ | – | ☑ | – | – | – | – | 3 | ✓ | – | – | – | – | |
| **PB** | [57] | 2011 | 36 | graph | C/C++ | – | ☑ | – | – | – | – | 3 | ✓ | – | – | – | – | |
| **DASP** | [44] | 2012 | 22 | clustering | C/C++ | – | ☑ | – | – | – | – | 5 | ✓ | ✓ | ✓ | – | ✓ | |
| **SEEDS** | [71] | 2012 | 98 | energy optimization | C/C++ | – | ✓ | ☑ | – | ✓ | – | 6 | ✓ | ✓ | – | – | – | |
| **reSEEDS** | ———"——— | | | ———"——— | C/C++ | – | ☑ | ✓ | – | ✓ | – | 6 | ✓ | ✓ | ✓ | – | – | |
| **TPS** | [63, 64] | 2012 | 8 + 1 | path | MatLab/C | – | ☑ | – | – | – | – | 4 | ✓ | – | – | – | ✓ | |
| **VC** | [66] | 2012 | 36 | clustering | C/C++ | – | ✓ | ☑ | – | – | – | 6 | ✓ | – | ✓ | – | – | |
| **CCS** | [72, 73] | 2013 | 6 + 4 | energy optimization | C/C++ | – | ✓ | ☑ | – | – | – | 3 | ✓ | ✓ | ✓ | – | – | |
| **VCCS** | [67] | 2013 | 87 | clustering | C/C++ | – | ☑ | – | – | – | – | 4 | – | – | ✓ | ✓ | – | |
| **CW** | [48] | 2014 | 11 | watershed | C/C++ | – | ☑ | – | – | – | – | 2 | ✓ | – | ✓ | – | – | |
| **ERGC** | [59, 60] | 2014 | 2 + 1 | contour evolution | C/C++ | – | ✓ | ☑ | – | – | – | 3 | ✓ | – | ✓ | – | – | |
| **MSS** | [49] | 2014 | 4 | watershed | C/C++ | – | ☑ | – | – | – | – | 5 | ✓ | – | – | – | – | |
| **preSLIC** | [48] | 2014 | 11 | clustering | C/C++ | – | ✓ | ☑ | – | – | – | 4 | ✓ | ✓ | ✓ | – | – | |
| **WP** | [50, 51] | 2014 | 5 + 8 | watershed | Python | – | ☑ | – | – | – | – | 2 | ✓ | – | ✓ | – | – | |
| **ETPS** | [74] | 2015 | 6 | energy optimization | C/C++ | – | ☑ | – | – | – | – | 5 | ✓ | ✓ | ✓ | – | – | |
| **LSC** | [68] | 2015 | 2 | clustering | C/C++ | – | ✓ | ☑ | – | – | – | 4 | ✓ | ✓ | ✓ | – | – | |
| **POISE** | [58] | 2015 | 3 | graph | MatLab/C | – | ☑ | – | – | – | – | 5 | ✓ | – | – | – | ✓ | |
| **SEAW** | [43] | 2015 | 0 | wavelet | MatLab/C | – | ☑ | – | – | – | – | 3 | – | – | – | – | – | |

Table A.4: List of evaluated superpixel algorithms. First of all we present the used acronym (in parts consistent with [30] and [35]), the corresponding publication, the year of publication and the number of Google Scholar citations as of October 13, 2016. We present a coarse categorization which is discussed in Section 3. We additionally present implementation details such as the programming language, supported color spaces and provided parameters. The color space used for evaluation is marked by a square. **Best viewed in color.**

the desired number of superpixels, we additionally considered the following two cases for special treatment. First, if an algorithm generates more that $K_{max}$ superpixels (or less than $K_{min}$), we interpolate linearly to determine the value for $K_{max}$ ($K_{min}$). Second, if an algorithm consistently generates less that $K_{max}$ (or more than $K_{min}$) superpixels, we take the value lower or equal (greater or equal) and closest to $K_{max}$ ($K_{min}$). In the second case, a superpixel algorithm is penalized if it is not able to generate very few (i.e. $K_{min}$) or very many (i.e. $K_{max}$) superpixels.

## Appendix D. Parameter Optimization

We discuss the following two topics concerning parameter optimization in more detail: color spaces and controlling the number of superpixels in a consistent manner. Overall, we find that together with Section 6, the described parameter optimization procedure ensures fair comparison as far as possible.

### *Appendix D.1. Color Spaces*

The used color space inherently influences the performance of superpixel algorithms as the majority of superpixel algorithms depend on comparing pixels within this color space. To ensure fair comparison, we included the color space in parameter optimization. In particular, we ensured that all algorithms support RGB color space and considered different color spaces only if reported in the corresponding publications or supported by the respective implementation. While some algorithms may benefit from different color spaces not mentioned in the corresponding publications, we decided to not consider additional color spaces for simplicity and to avoid additional overhead during parameter optimization. Parameter optimization yielded the color spaces highlighted in Table A.4.

### *Appendix D.2. Controlling the Number of Generated Superpixels*

Some implementations, for example **ERS** and **POISE**, control the number of superpixels directly – for example by

stopping the merging of pixels as soon as the desired number of superpixels is met. In contrast, clustering-based algorithms (except for **DASP**), contour evolution algorithms, watershed-based algorithms as well as path-based algorithms utilize a regular grid to initialize superpixels. Some algorithms allow to adapt the grid in both horizontal and vertical direction, while others require a Cartesian grid. We expected this difference to be reflected in the experimental results, however, this is not the case. We standardized initialization in both cases.

## Appendix E. Experiments

We complement Section 7 with additional experimental results. In particular, we provide additional qualitative results to better judge the visual quality of individual superpixel algorithms. Furthermore, we explicitly present ASA and $UE_{Levin}$ on the BSDS500 and NYUV2 datasets as well as Rec, UE and EV on the SBD, SUNRGBD and Fash datasets.

### *Appendix E.1. Qualitative*

We briefly discuss visual quality on additional examples provided in Figures E.18 and E.19. Additionally, Figure E.20 shows the influence of the compactness parameter on superpixel algorithms not discussed in Section 7.1.

Most algorithms exhibit good boundary adherence, especially for large K. In contrast to the discussion in Section 7.1 focussing on qualitative results with K $\approx$ 400 and K $\approx$ 1200, Figures E.18 and E.19 also show results for K $\approx$ 3600. We observe that with rising K, most algorithms exhibit better boundary adherence. Exceptions are, again, easily identified: **FH**, **QS**, **CIS**, **PF**, **PB**, **TPS** and **SEAW**. Still, due to higher K, the effect of missed image boundaries is not as serious as with less superpixels. Overall, the remaining algorithms show good boundary adherence, especially for high K.

Compactness increases with higher K; still, a compactness parameter is beneficial. While for higher K, superpixels tend to be more compact in general, the influence of parameter optimization with respect to Rec and UE is still visible – also for algorithms providing a compactness parameter. For example, **ERGC** or **ETPS** exhibit more irregular compared to **SLIC** or **CCS**. Complementing this discussion, Figure E.20 shows the influence of the compactness parameter for the algorithms with compactness parameter not discussed in detail in Section 7.1. It can be seen, that a compactness parameter allows to gradually trade boundary adherence for compactness in all of the presented cases. However, higher K also induces higher compactness for algorithms not providing a compactness parameter such as **CIS**, **RW**, **W** or **MSS** to name only a few examples. Overall, compactness benefits from higher K.

Overall, higher K induces both better boundary adherence and higher compactness independent of a compactness parameter being involved.

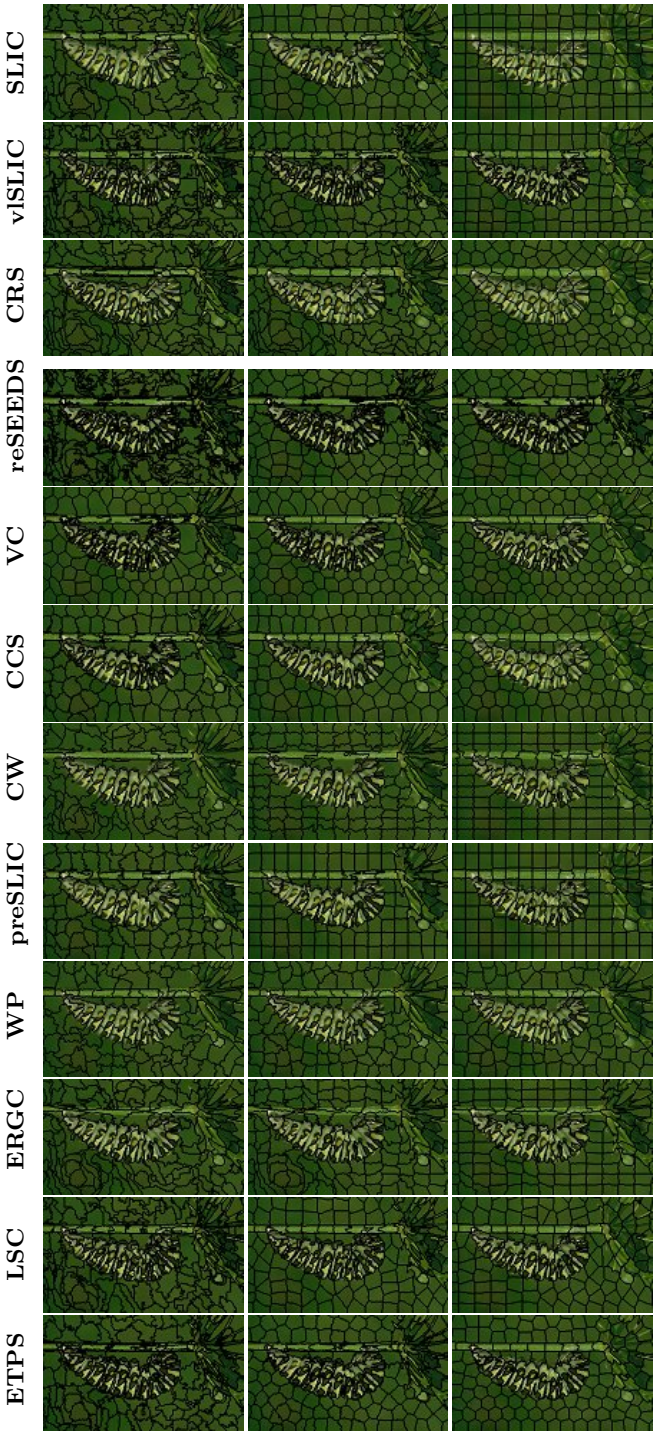



Figure E.20: The influence of a low, on the left, and high, on the right, compactness parameter demonstrated on the caterpillar image from the BSDS500 dataset for K $\approx$ 400. Superpixel boundaries are depicted in black; best viewed in color. For all shown algorithms, the compactness parameter allows to gradually trade boundary adherence for compactness. **Best viewed in color.**

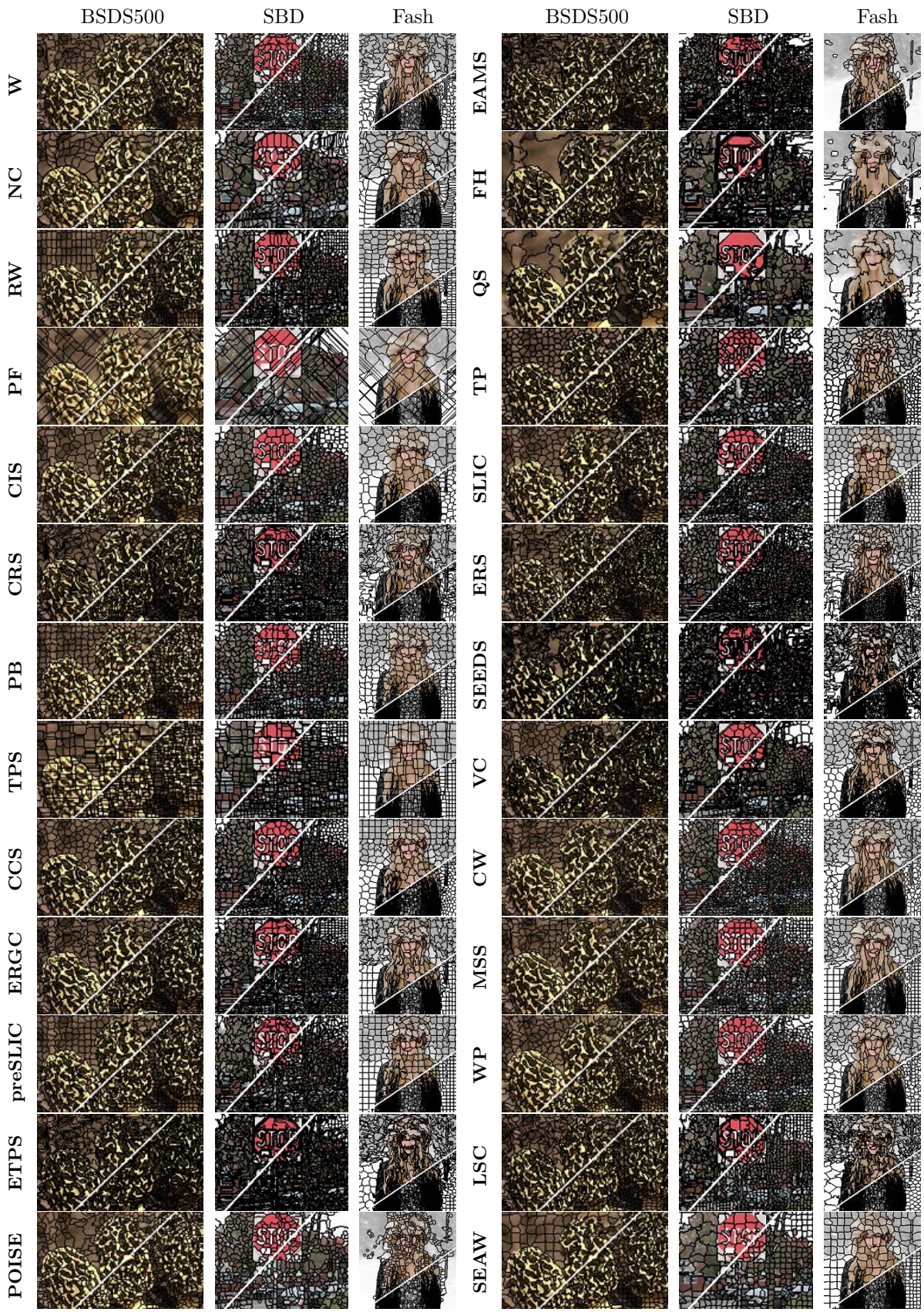



Figure E.18: Qualitative results on the BSDS500, SBD and Fash datasets; excerpts from the images in Figure 1 are shown for $K \approx 1200$, in the upper left corner, and $K \approx 3600$, in the lower right corner. Superpixel boundaries are depicted in black; best viewed in color. We observe that with higher K both boundary adherence and compactness increases, even for algorithms not offering a compactness parameter. **Best viewed in color.**

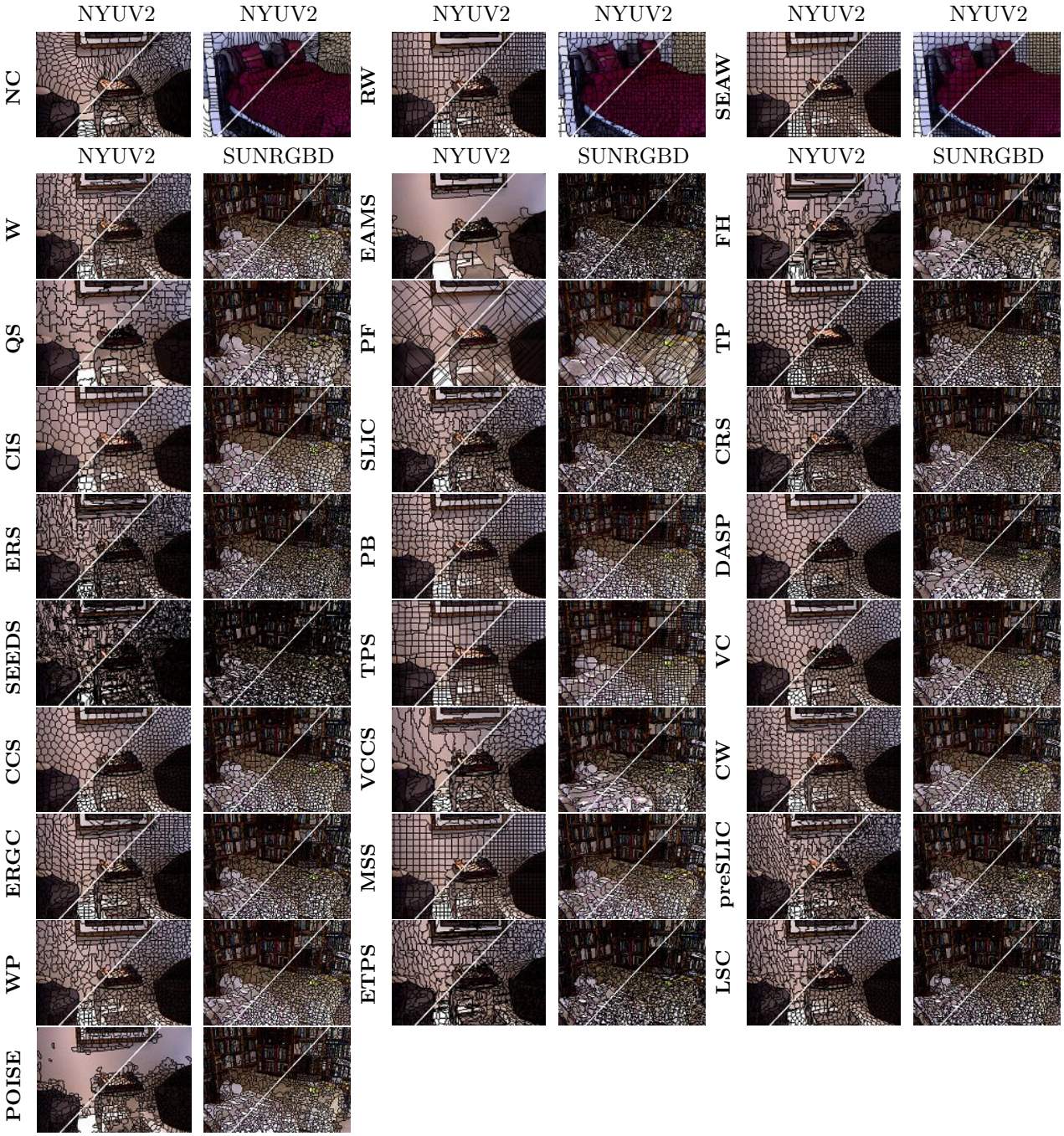



Figure E.19: Qualitative results on the NYUV2 and SUNRGBD datasets; excerpts from the images in Figure 1 are shown for $K \approx 1200$, in the upper left corner, and $K \approx 3600$, in the lower right corner. Superpixel boundaries are depicted in black; best viewed in color. **NC**, **RW** and **SEAW** could not be evaluated on the SUNRGBD dataset due to exhaustive memory usage of the corresponding MatLab implementations. Therefore, results on the NYUV2 dataset are shown. **Best viewed in color.**

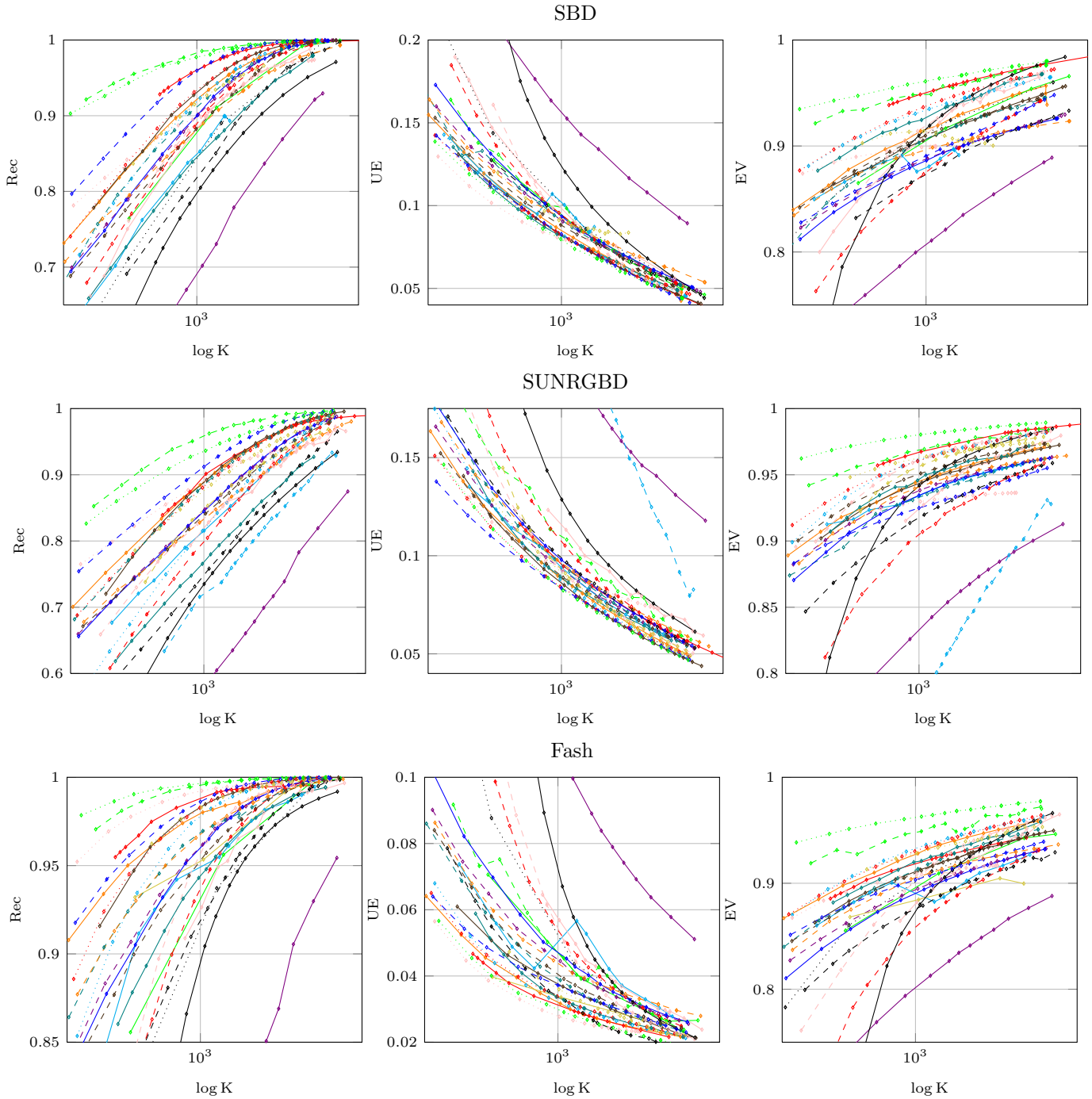



Figure E.21: Rec, UE and EV on the SBD, SUNRGBD and Fash datasets. Similar to the results presented for the BSDS500 and NYUV2 datasets (compare Figures 9 and 10), Rec and UE give a roguh overview of algorithm performance with respect to ground truth. Concerning Rec, we observe similar performance across the three datasets, while algorithms may show different behavior with respect to UE. Similarly, EV gives a ground truth independent overview of algorithm performance where algorithms show similar performance across datasets. **Best viewed in color.**

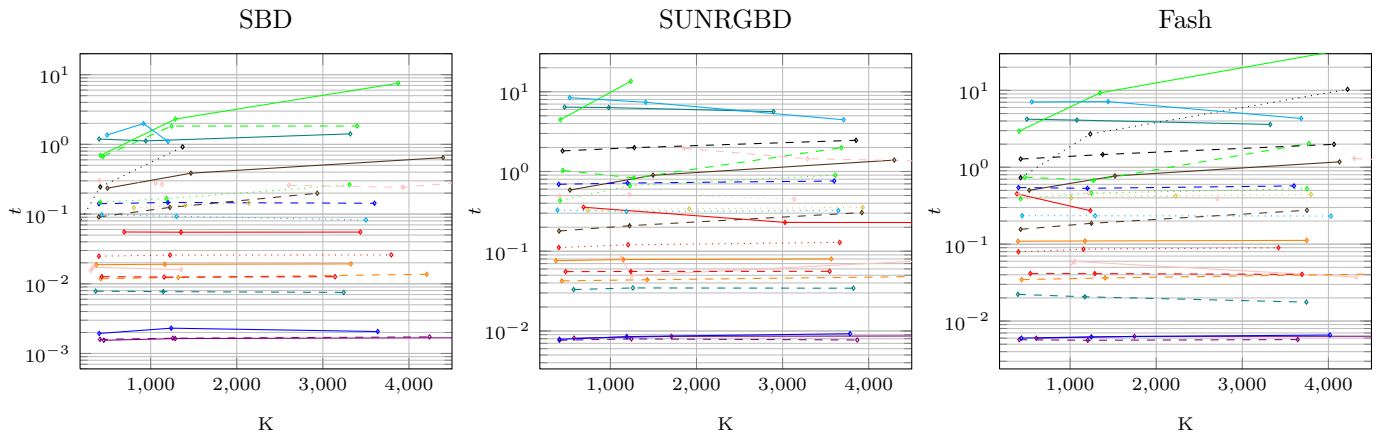



Figure E.22: Runtime in seconds $t$ on the SBD, SUNRGBD and Fash datasets. The results allow to get an impression of how runtime of individual algorithms scales with the size of the image. In particular, we deduce that most algorithm's runtime scales linear in the input size, while the number of generated superpixels does have little influence. **Best viewed in color.**

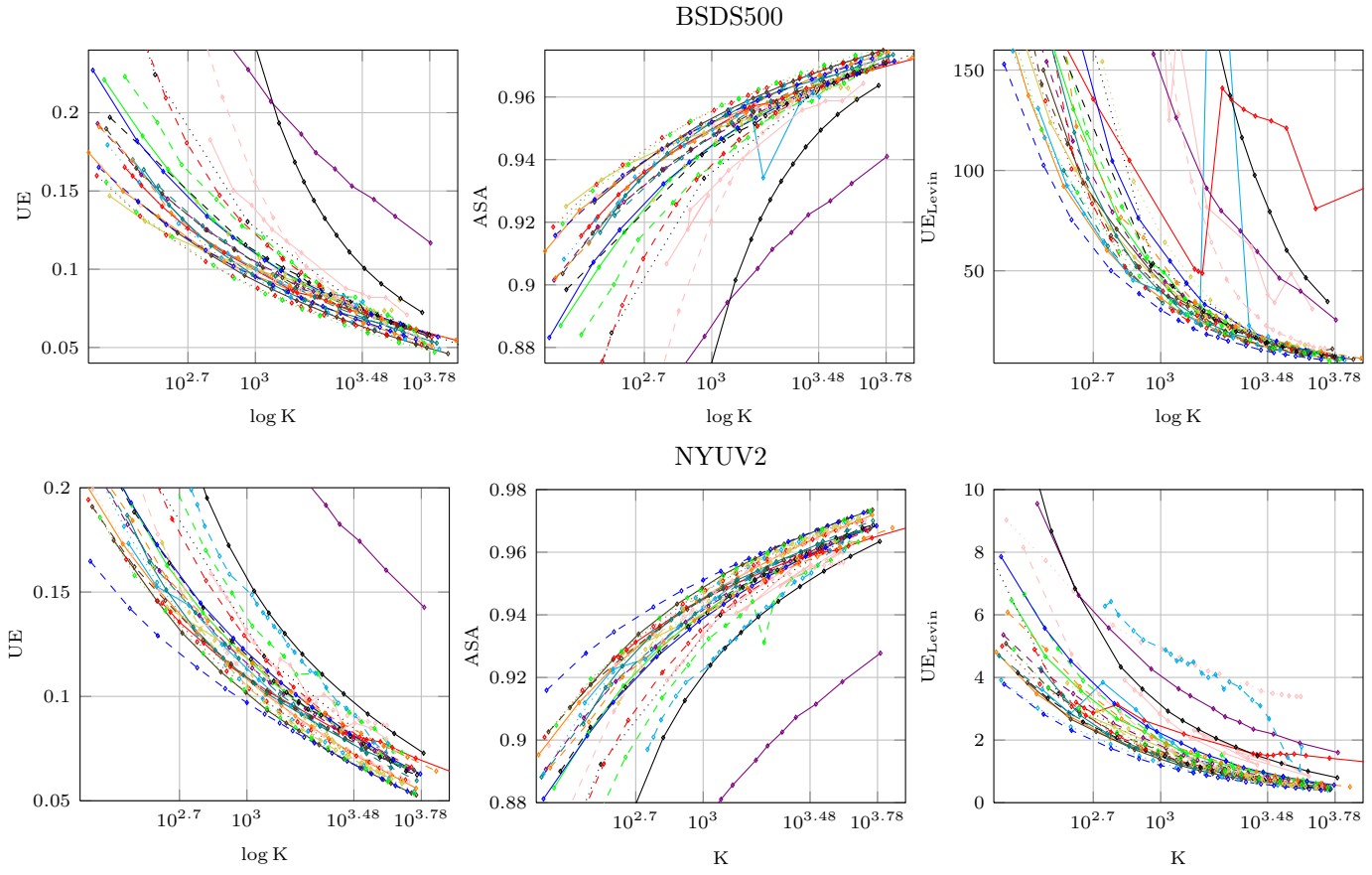



Figure E.23: UE, ASA and $\text{UE}_{\text{Levin}}$ on the BSDS500 and NYUV2 datasets. We find that ASA does not provide new insights compared to UE, as it closely reflects $(1 - \text{UE})$ except for a minor absolute offset. $\text{UE}_{\text{Levin}}$ , in contrast, provides a different point view compared to UE. However, $\text{UE}_{\text{Levin}}$ is harder to interpret and strongly varies across datasets. **Best viewed in color.**

**W** **EAMS** **NC** **FH** **RW** **QS** **PF** **TP** **CIS**
**SLIC** **CRS** **ERS** **PB** **DASP** **SEEDS** **TPS** **VC** **CCS**
**VCCS** **CW** **ERGC** **MSS** **preSLIC** **WP** **ETPS** **LSC** **POISE**
**SEAW**

*Appendix E.2. Quantitative*

The following experiments complement the discussion in Section 7.2 in two regards. First, we present additional experiments considering both ASA and $\mathrm{UE}_{\mathrm{Levin}}$ on the BSDS500 and NYUV2 datasets. Then, we consider Rec, UE and EV in more details for the remaining datasets, i.e. the SBD, SUNRGBD and Fash datasets. We begin by discussing ASA and $\mathrm{UE}_{\mathrm{Levin}}$ , also in regard to the observations made in Sections 5.1 and Appendix C.

As observed on the BSDS500 and NYUV2 datasets in Section 7.2, Rec and UE can be used to roughly asses superpixel algorithms based on ground truth. However, for large K, these metrics are not necessarily sufficient to discriminate between the superpixel algorithms. Considering Figure E.21, in particular with regard to Rec, we can identify algorithms showing above-average performance such as **ETPS** and **SEEDS**. These algorithms perform well on all three datasets. Similarly, **PF**, **QS**, **SEAW** and **TPS** perform poorly on all three datasets. Regarding UE, in contrast, top-performer across all three algorithms are not identified as easily. For example, **POISE** demonstrates low UE on the SBD and Fash datasets, while performing poorly on the SUNRGBD dataset. Similarly, **ERS** shows excellent performance on the SUNRGBD dataset, while being outperformed by **POISE** as well as **ETPS** on the SBD and Fash datasets. Overall, Rec and UE do not necessarily give a consistent view on the performance of the superpixel algorithms across datasets. This may also be explained by the ground truth quality as already discussed in Section 7.2.

The above observations also justify the use of EV to judge superpixel algorithms independent of ground truth. Considering Figure E.21, in particular, with regard to EV, we can observe a more consistent view across the datasets. Both, top-performing algorithms such as **ETPS** and **SEEDS**, as well as poorly performing algorithms such as **PF**, **PB** or **TPS** can easily be identified. In between these two extremes, superpixel algorithms are easier to discriminate compared to Rec and UE. Furthermore, some superpixel algorithms such as **QS**, **FH** or **CIS** are performing better compared to Rec or UE. This confirms the observations that ground truth independent assessment is beneficial but cannot replace Rec or UE.

We find that ASA closely mimicks the behavior of $(1 - \mathrm{UE})$ while $\mathrm{UE}_{\mathrm{Levin}}$ may complement our discussion with an additional viewpoint which is, however, hard to interpret. We consider Figure E.23 showing UE, ASA and $\mathrm{UE}_{\mathrm{Levin}}$ for both the BSDS500 and NYUV2 datasets. Focussing on UE and ASA, we easily see that ASA nearly reflects $(1 - \mathrm{UE})$ while being a small constant off. In particular, all algorithms exhibit nearly the same behavior, while absolutely the algorithms show higher ASA compared to $(1 - \mathrm{UE})$. This demonstrates that ASA does not give new insights with respect to the quantitative comparison of superpixel algorithms. In contrast, the algorithms show different behavior considering $\mathrm{UE}_{\mathrm{Levin}}$ . This is mainly due to the unconstrained range of $\mathrm{UE}_{\mathrm{Levin}}$ (compared to $\mathrm{UE} \in [0, 1]$). In particular, for algorithms such as **EAMS** and **FH**, $\mathrm{UE}_{\mathrm{Levin}}$ reflects the behavior of max UE as shown in Figure 9e. The remaining algorithms lie more closely together. Still, algorithms such as **ERS**, **SEEDS** or **PB** show better $\mathrm{UE}_{\mathrm{Levin}}$ than UE (seen relatively to the remaining algorithms). In the case of **EAMS** and **FH**, high $\mathrm{UE}_{\mathrm{Levin}}$ may indeed be explained by the considerations of Neubert and Protzel [35] arguing that $\mathrm{UE}_{\mathrm{Levin}}$ unjustly penalizes large superpixels. For the remaining algorithms, the same argument can only be applied in smaller scale as these algorithms usually do not generate large superpixels. In this line of throught, the excellent performance of **ERS** may be explained by the employed regularizer for enforcing uniform superpixel size. Overall, ASA does not contribute to an insightful discussion, while $\mathrm{UE}_{\mathrm{Levin}}$ may be considered in addition to UE to complete the picture of algorithm performance.

*Appendix E.3. Runtime*

We briefly discuss runtime on the SBD, SUNRGBD and Fash datasets allowing to get more insights on how the algorithms scale with respect to image size and the number of generated superpixels.

We find that the runtime of most algorithms scales roughly linear in the input size, while the number of generated superpixels has little influence. We first remember that the average image size of the SBD, SUNRGBD and Fash datasets is: $314 \times 242 = 75988$, $660 \times 488 = 322080$ and $400 \times 600 = 240000$. For $\mathrm{K} \approx 400$, **W** runs in roughly 1.9ms and 7.9ms on the SBD and SUNRGBD datasets, respectively. As the input size for the SUNRGBD dataset is roughly 4.24 times larger compared to the SBD dataset, this results in roughly linear scaling of runtime with respect to the input size. Similar reasoning can be applied to most of the remaining algorithms, especially fast algorithms such as **CW**, **PF**, **preSLIC**, **MSS** or **SLIC**. Except for **RW**, **QS** and **SEAW** we also notice that the number of generated superpixels does not influence runtime significantly. Overall, the results confirm the claim of many authors that algorithms scale linear in the input size, while the number of generated superpixels has little influence.