Published in Image Processing On Line on 2022–04–21.
Submitted on 2021–07–30, accepted on 2022–04–05.
ISSN 2105–1232 © 2022 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2022.373

Bilateral K-Means for Superpixel Computation (the SLIC Method)

Robin Gay[1], Jérémie Lecoutre[1], Nicolas Menouret[1], Arthur Morillon[1], Pascal Monasse[2]

[1]École des Ponts ParisTech, F-77455 Marne-la-Vallée, France
{robin.gay,jeremie.lecoutre,nicolas.menouret,arthur.morillon}@eleves.enpc.fr
[2]Université Paris-Est, LIGM (UMR CNRS 8049), ENPC, F-77455 Marne-la-Vallée, France
pascal.monasse@enpc.fr

Communicated by Gregory Randall *Demo edited by* Pascal Monasse

Abstract

As a substitute to a full segmentation of a digital image, or as preprocessing to a segmentation algorithm, superpixels provide an over-segmentation that offers several benefits: good adherence to edges, uniformity of color inside superpixels, a richer adjacency structure than the regular grid of pixels, and the fact that each node of the graph of superpixels has a shape, which can be used in subsequent processing. Moreover, their evaluation is less subjective than a full segmentation, which somehow always involves a semantic interpretation of the scene. The SLIC method (Simple Linear Iterative Clustering) has been a very popular algorithm to compute superpixels since its introduction. Its advantage is due to its simplicity and to its computing time performance. In essence, it consists in a K-means clustering in bilateral domain, involving both position and color. We study in detail this algorithm and propose a fast, simple post-processing that ensures that superpixels are connected, a property not ensured by the original method.

Source Code

The commented C++ source code for SLIC and its documentation are available on the web page of this article[1]. Usage instructions are detailed in the `README.md` file of the archive.

Keywords: superpixel; image segmentation; SLIC (Simple Linear Iterative Clustering)

[1]https://doi.org/10.5201/ipol.2022.373

1 Introduction

A superpixel is a set of pixels that share some common features: they are close to each other and they have also close colors. It is interesting to have a full segmentation of the image into superpixels. This usually provides an over-segmentation, without any semantic inference. Many algorithms can then be applied to the superpixel graph instead of pixel-level graph: this provides an accelerated processing because superpixels are much fewer than pixels. Additionally, the shape of the superpixel can be useful. For instance, superpixels are used directly for tracking [19], stereo [18] and 3D reconstruction [3], and can also be at the basis of object detection [2] or semantic segmentation [5].

Prior to the publication of the SLIC algorithm by Achanta et al. [1], all superpixel algorithms had super-linear complexity, at best $O(N \log N)$ and more frequently $O(N^2)$ or higher, with N the number of pixels of the image. An exception was TurboPixels [7], which has linear complexity but with a high constant factor (the average time for a 3 Mpx. image being around 800 seconds [1]). In contrast, SLIC's complexity is linear $O(N)$. The number of superpixels can also be decided beforehand in contrast to some other methods, but what makes it especially attractive is its simplicity. In essence, it relies on K-means applied with an appropriate distance mixing space proximity and color closeness. For this reason, it is applied in bilateral space, in the same manner as the bilateral filter [12]. Moreover, since spatial and color proximity are mixed in a composite distance, the weight attributed to each can be user-defined. This parameter, referred as compactness, favors either the spatial proximity or the color similarity, and it can be adjusted to favor round or irregular superpixels.

One drawback of the algorithm is that the connectivity of superpixels is not ensured. This can be inconvenient for posterior processing based on superpixels. To remedy this problem, we propose a simple and fast solution that ensures that final superpixels are connected. It simply keeps the largest connected component of the SLIC superpixels, making pixels of other connected components orphans[2]. Then a second pass in a specific order reassigns the orphans to an adjacent superpixel.

Section 2 gives some background. Section 3 explains how the SLIC method proceeds and the post-processing repairing connectivity is exposed in Section 4. Experiments and discussion follow in Section 5 and we finish with a conclusion.

2 Background

This section presents the background about K-means, the main procedure underlying the SLIC algorithm, and explain how the graph or pixels in the regular grid can be transformed into a higher level graph of superpixels, provided the map from pixels to superpixels is known.

2.1 K-Means

The K-means algorithm is a simple method to approximate the solution of a hard problem. Let us consider points $\{x_i\}$, $i \in \{1, \dots, n\}$ in $\mathbb{R}^d$ and the problem of clustering the points in K clusters, K being a positive integer, which can be formulated as

$$\arg \min_{\{X_k\},\{\epsilon_{ik}\}} \sum_{k=1}^{K} \sum_{i=1}^{n} \epsilon_{ik}\, D(x_i, X_k), \tag{1}$$

$$\text{s.t. } \forall i\ \epsilon_{ik} \in \{0, 1\},\ \sum_k \epsilon_{ik} = 1. \tag{2}$$

[2]We call *parent* of a pixel the superpixel that contains it, and following this terminology, *orphan* a pixel losing its parent before a subsequent *adoption*.

Points $X_k \in \mathbb{R}^d$ are called the superpixel centers. D is a distance or an increasing function of a distance. Condition (2) means that each x_i must be assigned to a single superpixel X_k that is the closest one. This defines a partition of the points x_i. A superpixel is then defined as its center X_k and its pixels $P(X_k) = \{x_i : \epsilon_{ik} = 1\}$.

An approximate solution can be found by block coordinate descent, a popular optimization strategy [17], separating the problems of finding X_k and ϵ_{ik}

$$\forall i, k \quad \epsilon_{ik} = \mathbf{1}\left(k = \arg\min_j D(x_i, X_j)\right), \tag{3}$$

$$\forall k \quad X_k = \arg\min_X \sum_{i=1}^{n} \epsilon_{ik}\, D(x_i, X). \tag{4}$$

Both steps are alternated until a fixed point is reached. The first one is called the *assignment step* and the second one the *update step*. The former is trivial to solve for any D: select for x_i the closest X_k, in contrast the latter is a hard problem whose solution strongly depends on properties of the choice of D. If D is the Euclidean norm, no closed form formula solves the update step, though the Weiszfeld's iterative algorithm is guaranteed to converge to the solution [16]. However, if D is the squared Euclidean norm, the solution is

$$X_k = \frac{\sum_i \epsilon_{i,k} x_i}{\sum_i \epsilon_{ik}}. \tag{5}$$

Equation (5) assumes that at least one point x_i is assigned to X_k.

The complexity of the assignment in step (3) is $O(K\,N)$. At first sight, it is also the case of the update step (4), but using the fact that most ϵ_{ik} are null, it is convenient to record as output of (3) a map ℓ indicating the index k for which $\epsilon_{ik} = 1$

$$\ell(i) = \arg\min_j D(x_i, X_j), \tag{6}$$

and we can just write

$$\epsilon_{ik} = \begin{cases} 1 & \text{if } k = \ell(i) \\ 0 & \text{otherwise} \end{cases}. \tag{7}$$

Then (5) can be computed in a single scan of the image, as shown by Algorithm 1, in complexity $O(N)$. The bottleneck of each iteration is therefore in the assignment step. Fortunately, it can be easily parallelized.

Algorithm 1: Applying the update formula (5) in $O(N)$

Input: Map ℓ associating to each point i its cluster index $k = \ell(i)$
Output: Vectors X_k according to (5)
$\forall k$, $X_k \leftarrow 0$, $n_k \leftarrow 0$;
foreach i **do**
  $X_{\ell(i)} \leftarrow X_{\ell(i)} + x_i$; `// Numerator of (5)`
  $n_{l(i)} \leftarrow n_{l(i)} + 1$; `// Denominator of (5)`
$\forall k$, $X_k \leftarrow X_k / n_k$;

2.2 Graph of Superpixels

The K superpixels form a partition of the image: each pixel is assigned to a single superpixel, called its parent. We have the map ℓ associating to each pixel p an index in $\{1, \ldots, K\}$ of the assigned superpixel, where each superpixel can be made explicit by more than its index. Its centroid can be computed, so as its average color. Direct access to its pixels without a full scan of the map ℓ can be achieved with a sorted array of pixels, as shown in Algorithm 2.

Algorithm 2: Direct access from a superpixel to its pixels

Input: Map ℓ associating to each pixel index its superpixel index
Output: Array of pixels S, for each k an interval $[b_k, e_k]$ in S of the pixels of the superpixel X_k
$S \leftarrow$ Sort $\{(x_i, \ell(i))\}$ by increasing value of ℓ
$\ell_- \leftarrow -1$ `// Previous label`
for *index* i *of* S **do**
  if $S[i].\ell \neq \ell_-$ **then**
    $b_{S[i].\ell} \leftarrow i$
  else
    $e_{S[i].\ell} \leftarrow i$
  $\ell_- \leftarrow S[i].\ell$

We can lift the adjacency of pixels to superpixels: two superpixels X_j and X_k are adjacent if

$$\exists x \in P(X_j), \exists y \in P(X_k), \text{such that } x \text{ and } y \text{ are adjacent.} \tag{8}$$

We choose 4-connectivity, but 8-connectivity may also be considered. When an efficient data structure for a mathematical set (data collection without repetition) is available[3], the way to recover the adjacency is straightforward, see Algorithm 3. Since the graph is undirected, we can consider the full neighborhood of pixel p and put a single link, or, as in our implementation, a half neighborhood and put both links between neighbor superpixels. The latter is more efficient as it involves fewer cache misses when reading ℓ. In 4-connectivity, we consider East and South neighbors of p, and in 8-connectivity East, South-East, South and South-West neighbors.

Algorithm 3: Recovering the adjacency structure of superpixels

Input: Map ℓ associating to each pixel index its superpixel index
Output: For each superpixel X_k, the set of indexes of its neighbors N_k
$\forall k$, $N_k \leftarrow \emptyset$
foreach *pixel* p *of* ℓ **do**
  foreach *Neighbor* q *of* p **do** `// Half-neighborhood is sufficient`
    if $\ell(p) \neq \ell(q)$ **then**
      $N_{\ell(p)} \leftarrow \ell(q)$ `// Insert in set`
      $N_{\ell(q)} \leftarrow \ell(p)$

[3]The type `std::set<int>` of the C++ Standard Template Library fits this requirement

3 The SLIC Method

3.1 Overview

The algorithm is initialized by putting initial superpixel positions on a regular grid inside the image, according to the number K chosen by the user. The spatial distance between nearest superpixels is then a value S. Each one is assigned the color (r, g, b) of its nearest pixel. Concatenating position and color, we get points $\mathbf{X}_k = (x_k, y_k, r_k, g_k, b_k) \in \mathbb{R}^5$ as clusters and points $\mathbf{x}_i = (u_i, v_i, r_i, g_i, b_i) \in \mathbb{R}^5$ to assign to them. A loop of assignment/update iterations is performed until convergence according to the standard K-means algorithm in $\mathbb{R}^5$.

3.2 Initialization

Given K the prescribed number of superpixels, each will have an "influence" zone of radius S pixels around it, hence the step size

$$S = \left\lfloor \sqrt{N/K} \right\rfloor . \tag{9}$$

A regular grid of $\lfloor w/S \rfloor \times \lfloor h/S \rfloor$ places the spatial position of the superpixels, $w \times h$ being the dimension of the image. It is centered in the image through the computation of the total padding $pad = (w - S\lfloor w/S \rfloor, h - S\lfloor h/S \rfloor)$, the remainder of the Euclidean division of each dimension by S. The positions are thus at

$$(i\,S + S/2 + pad_x/2, j\,S + S/2 + pad_y/2), \tag{10}$$

with integer values i and j, as illustrated in Figure 1.

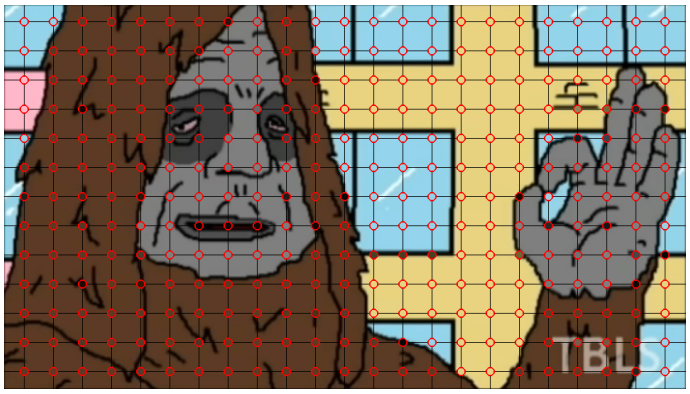


Figure 1: Superpixel centers (red circles) on the initial grid (black lines).

Notice that the resulting number of superpixels may be different than requested: apart from low or large values of K, $K \ll N$ or $K \sim N$, in which the rounding effects are noticeable, images of size $W \times H$ with $W \ll H$ or $H \ll W$ may distort K because superpixels are placed on a regular grid. For instance, for an image 16×16 with requested $K = 64$, (9) yields $S = 2$ and the exact number of superpixels $K' = K$ are created. For the same number of pixels N in an image of dimension 256×1, $K' = 128 = 2K$.

3.3 Gradient Correction

Superpixels should be placed in homogeneous regions. The blind initialization may place some on an edge, characterized by a strong gradient. An optional parameter g is considered to shift to the

Algorithm 4: Minimal gradient shift of superpixel position

Input: Image I, initial position $X = (x, y)$ of a superpixel, radius of search g
Output: Corrected position X' with minimal gradient
$m \leftarrow +\infty$
for $p \in [x-g, x+g] \times [y-g, y+g]$ **do**
  $G \leftarrow \|I(p) - I(p+(1,0))\|^2 + \|I(p) - I(p+(0,1))\|^2$
  if $m > G$ **then**
    $m \leftarrow G$
    $X' \leftarrow p$

position with lowest gradient in the window $[-g, g] \times [-g, g]$ centered at the initial position, this is done following Algorithm 4.

$I(p)$ is a 3-vector (r, g, b), and the squared Euclidean distance between $I(p)$ and $I(q) = (r', g', b')$ is $(r-r')^2+(g-g')^2+(b-b')^2$. Care must be taken that neighbors $p+\delta$ do not fall outside the image. When that happens, $p - \delta$ should be used instead. It may also happen that some positions p in the search square are out of bounds. Such positions must be ignored in the minimum computation.

3.4 The Bilateral Distance

Superpixel centers are characterized by position and color, a vector of $\mathbb{R}^5$ in the form $\mathbf{X} = (x, y, r, g, b)$. To compare to a pixel $\mathbf{x} = (x', y', r', g', b')$ we shall consider both spatial and color distances

$$D_s(\mathbf{x}, \mathbf{X}) = \sqrt{(x - x')^2 + (y - y')^2}, \tag{11}$$

$$D_c(\mathbf{x}, \mathbf{X}) = \sqrt{(r - r')^2 + (g - g')^2 + (b - b')^2}. \tag{12}$$

The Euclidean distance in $\mathbb{R}^5$ is $\sqrt{D_s^2 + D_c^2}$ but position and color have different scales, hence a weighting factor m should be used

$$D(\mathbf{x}, \mathbf{X})^2 = m^2 \frac{D_s(\mathbf{x}, \mathbf{X})^2}{S^2} + D_c(\mathbf{x}, \mathbf{X})^2. \tag{13}$$

The normalizing factor S^2 is meant to give an interpretation of m independent of K: a zoom-in by a factor 2 of the image with the same numbers K and m results in the same zoomed-in superpixels. The weight m can be interpreted as a compactness parameter: the higher, the more weight is given to spatial distance and superpixels are more compact. When $m \gg 1$, only spatial distance counts and the initial regular grid of superpixels is optimal.

3.5 Pseudo-code

The pseudo-code of SLIC, Algorithm 5, is simple and follows the K-means algorithm almost unmodified. The difference is that the influence region of a superpixel is limited spatially to the neighborhood $[-S, S]^2$ around its center. This is sensible since strongly elongated superpixels are not desired. It also accelerates the assignment step: whereas it is normally of complexity $O(K\,N)$, the search area is reduced to $4S^2$ pixels instead of N. Since $S \approx \sqrt{N/K}$, $4S^2 \approx 4N/K$ and the step becomes $O(N)$. The strict convergence condition $E = 0$ is relaxed to E being small. Various stopping criteria are discussed in Section 5.5.3.

Algorithm 5: Bilateral K-means applied to $SLIC$

Input: Image I, number of superpixels K, compactness parameter m, radius of minimal gradient search g (optional)

Output: Map ℓ giving for each pixel the index k of its assigned superpixels, superpixels centers $X_k \in \mathbb{R}^5$

Initialize centers X_k and compute radius S // See Section 3.2

Gradient-driven shift of each X_k // Algorithm 4 if $g > 0$

$E \leftarrow +\infty$

while $E > 0.5$ **do**

  $\forall p,\ d(p) \leftarrow \infty$ // Min distance to each pixel

  for $k = 1 \dots K,\ pixel\ p \in [x_k - S, x_k + S] \times [y_k - S, y_k + S]$ **do** // Assignment step

    $e(p) \leftarrow D\left((p, I(p)), X_k\right)^2$ // (13) depends on m

    if $d(p) > e(p)$ **then**

      $d(p) \leftarrow e(p)$

      $\ell(p) \leftarrow k$

  $X'_k \leftarrow$ Update step // Algorithm 1

  $E \leftarrow \sqrt{\sum_k D_s(X_k, X'_k)^2 / K}$ // Compute shift between iterations

  $X_k \leftarrow X'_k$

4 Connectivity Enforcement

The main weakness of the SLIC algorithm is the lack of a component that enforces the connectivity of the superpixels. For a low compactness parameter m in (13), color homogeneity is favored over spatial proximity. Even for a fairly high value of m, the connectivity is not guaranteed as a hard constraint. See Figure 2 for an example of disconnected superpixel. For practical reasons, it is best to work with connected superpixels, and we propose here an algorithm ensuring this property.

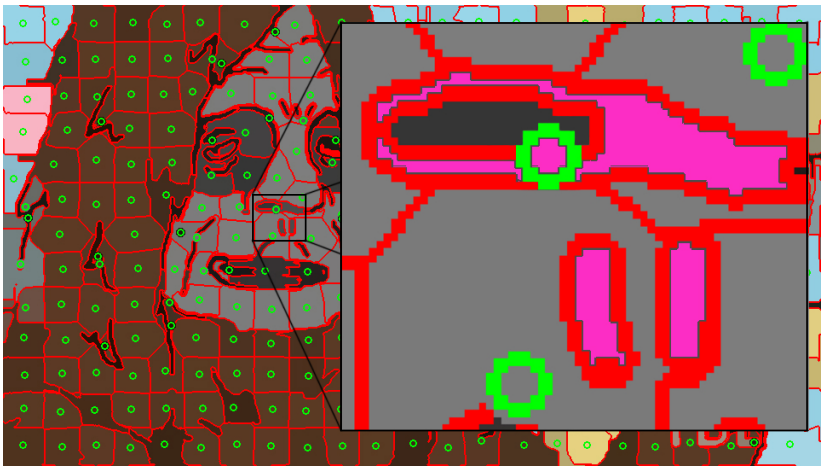


Figure 2: This image was treated by SLIC without enforcing connectivity. The superpixel with pink color has three connected components (green dots represent superpixel centers).

The principle is to keep only the connected component with largest area of each superpixel. The pixels of the other minor connected components are temporarily declared orphans and are adopted in a second phase. The adoption proceeds by looking at neighbors of these pixels: if all labeled neighbors are from the same superpixel, the orphan pixel is assigned to it; if several labels appear among neighbors, the one closest in color is selected. Without precaution, this procedure is too

greedy, as a newly adopted pixel can propagate directly its assignment to its neighbors that had initially no labeled neighbors. To mitigate this effect, a distance-guided propagation is performed as described in Section 4.3.

4.1 Labeling the Connected Components

The map ℓ stores for each pixel the index of its parent superpixel, an integer. The iso-levels of this map must be labeled by connected component. Many such algorithms exist [6] and we choose an elementary one. Two main categories of algorithms exist: label-propagation and label-equivalence-resolving. The method we propose is in the first category (see Algorithm 6).

Pixels are scanned in raster scan order. All bear initially the label -1 in map cc. When the current pixel p is not labeled, a new label is created and $cc(p)$ receives this label. It is then put in a stack (any dynamic data collection is adequate) and the algorithm iterates until the stack gets empty, at which point the connected component of iso-level of ℓ containing the pixel is labeled in cc. A pixel is popped from the stack, and all its unlabeled neighbors with the same value in ℓ get the current label and are stored in the stack for subsequent propagation. The latter condition is crucial so that a pixel can never enter the stack more than once and the algorithm finishes. The raster scan can then proceed with the pixel next to p. Pixels having four neighbors, each one can be reached only four times by propagation and the number of reads in ℓ is about $4N$, therefore Algorithm 6 has linear complexity $O(N)$. At the same time, the number of pixels in each connected components is counted for later usage, that is, the histogram of cc.

Algorithm 6: Connected component labeling

```
Input: Image ℓ
Output: Image of labels cc, histogram H of cc
cc(.) ← −1                                                  // Invalid label
n ← 0                                                       // Current label
for pixel p do
    if cc(p) == −1 then
        cc(p) ← n
        C ← p                          // C is any dynamic array, stack or queue
        H(n) ← 0
        while C ≠ ∅ do
            q ← C                                           // Pop a pixel from C
            H(n) ← H(n) + 1
            foreach r ∼ q do                                // Neighbors of q
                if ℓ(r) = ℓ(p) and cc(r) = −1 then
                    cc(r) ← n
                    C ← r                                   // Push in C
        n ← n + 1
```

4.2 Discard Minor Connected Components

The next step's goal is to make orphans (label value -1) the pixels of ℓ that are not in the major connected component of their parent superpixel. Notice that the label of this component in cc is unknown, so it has to be computed first. ℓ having K labels, an array of K elements M_i is filled with

the label of the running maximum area of the superpixel's components. The image is scanned pixel by pixel p, and if $H(M_{\ell(p)}) > H(cc(p))$ then $M_{\ell(p)}$ is changed to $cc(p)$. Second, each superpixel weeds out the pixels of its minor connected components, which become orphans. For that, we scan the pixels and each one p whose label in cc is not $M_{\ell(p)}$ becomes orphan. The whole procedure in Algorithm 7 makes two scans of the image, so has complexity $O(N)$. An example of orphan superpixels is in Figure 3. Since the superpixels were modified, their average color may be recomputed.

Algorithm 7: Make orphans the pixels in minor connected components of their superpixel

Input: Map ℓ giving for each pixel the index k of its assigned superpixels, map cc of connected components of ℓ, histogram H of cc

Output: ℓ is modified with some pixels getting an invalid label -1

$\forall k = 1 \dots K$, $M_k \leftarrow -1$ `// Invalid label`

foreach *pixel* p **do**
  if $M_{\ell(p)}$ *is a valid label and* $H(M_{\ell(p)}) > H_{cc(p)}$ **then**
    $M_{\ell(p)} \leftarrow cc(p)$

foreach *pixel* p **do**
  if $cc(p) \neq M_{l(p)}$ **then**
    $cc(p) \leftarrow -1$

(a) Original image

(b) Orphan pixels in color and superpixels' major connected components

Figure 3: An image before and after SLIC and the first two steps of the connectivity enforcement algorithm.

4.3 Adoption of Orphans

Orphans are processed in order of their distance to labeled pixels. This avoids having an orphan adjacent to a superpixel and propagating its label to a large region of orphans. We use the Manhattan distance $D_M(p,q) = |x_p - x_q| + |y_p - y_q|$, ensuring that distances are integers and that a shortest path algorithm gives the solution. The chosen distance coincides with the 4-connectivity graph of pixels and any shortest path algorithm on graph can be used. The problem is simple here since each edge has a weight one. A good candidate is the Dijkstra algorithm [4]. However, this requires the use of a priority queue and then sorting the pixels by their distance.

To compute the distance map to labeled pixels, we prefer a simpler method implemented in just a few lines of code: perform successive dilations of set of pixels, Algorithm 8. First, pixels at distance 0 (labeled pixels) are dilated, the orphan pixels affected have distance 1; perform a new dilation and affected orphan pixels are at distance 2 and so on until all pixels get a distance. We store the distance directly in ℓ with negative integers: regular labeled pixels (at distance 0) have a positive label, while orphans have as label the opposite of their distance. Each dilation requires a scan of the image. At worst, we need up to N dilations before it finishes and therefore the algorithm has worst case complexity $O(N^2)$. Actually, the distance of an orphan is $O(S)$ yielding at most $O(S^2) = O(N)$ dilations. In practice, distances do not become large and the algorithm stops much earlier. When pixels have their distance computed, they can be queued, so that the second stage does not need a sort procedure.

Algorithm 8: Distance of orphans to labeled pixels

Input: Map ℓ, with negative labels for orphans
Output: Pixels of ℓ with negative labels are replaced by the opposite of their distance to labeled pixels, queue Q of orphans ordered by distance

```
foreach pixel p do
    if ℓ(p) < 0 then ℓ(p) ← −∞          // Actually smallest representable integer
for d = −1, −2, ... do                   // Stops when no pixel has changed
    foreach pixel p do
        if ℓ(p) > d then
            foreach q ∼ p do             // Neighbors of p
                if l(q) < d then
                    l(q) ← d
                    Q ← q                // Push pixel in queue
```

The second stage proceeds by propagating the label of the superpixel closest in color to orphans, following Algorithm 9. The minimum in the formula involves the neighbors of p and it is guaranteed that one of them at least has a valid label. This is due to the fact that Q is popped in increasing order of distance. The procedure requires a single pass through the queue, which has less than N pixels, and the minimum involves four neighbors for p, hence a complexity of $O(N)$.

Algorithm 9: Adoption of orphans

Input: Image I, map ℓ, queue Q of orphans ordered by distance, superixels X_k
Output: Map ℓ without orphans

```
while Q not empty do
    p ← Q                                          // Pop from queue
    q ← arg min_{q∼p, ℓ(q)≥0} D_c((p, I(p)), X_{ℓ(q)})^2   // Neighbors of p with valid parent
    superpixel
    l(p) ← l(q)
```

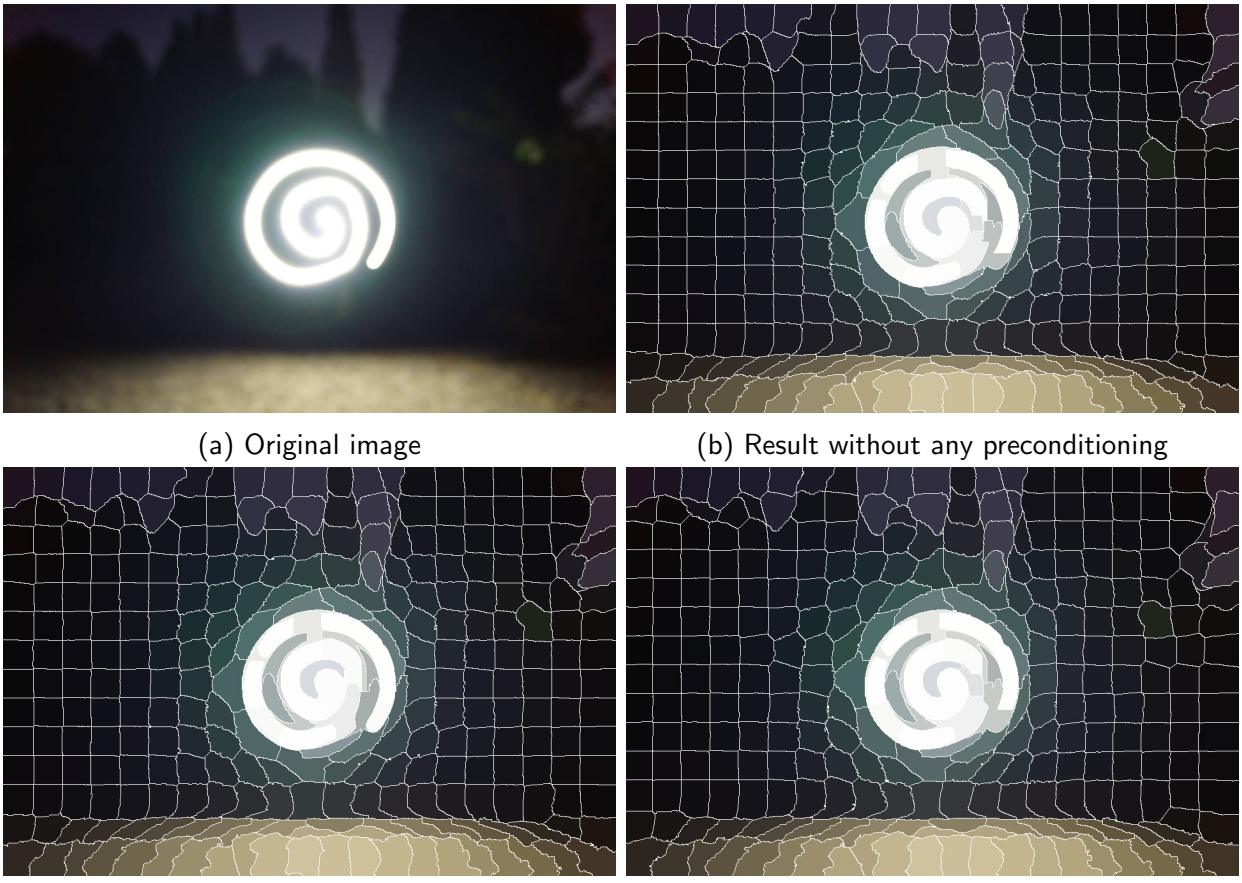
5 Experiments

5.1 Implementation

The input of SLIC is the color or gray-scale image and the parameters K, m, g. Its output is the full graph of the superpixels along with the map ℓ giving for each pixel the index of its parent superpixel. Contrary to the original implementation by the authors of [1], the map ℓ is not stored to a file[4]. The program outputs only an image representing the superpixels: pixels get the color of their parent superpixel, but pixels at the boundary of a superpixel are displayed in white (user-configurable). To this end, pixels having a neighbor with a different superpixel are marked. To avoid thick borders of 2 pixels, only half the neighborhood is examined, namely East and North neighbors.

5.2 The Minimum Gradient Preconditioning

The authors of SLIC [1] propose to initialize the superpixels by relocating the centers to the pixel of minimum gradient in a 3×3 square centered at their initial grid position, as explained in Section 3.3. This is meant to avoid initializing superpixel centers on an edge of the image. This pre-processing is supposed to improve the results of the algorithm in terms of contour adherence.



(a) Original image

(b) Result without any preconditioning

(c) Result with a 3×3 minimum gradient search zone

(d) Result with a 10×10 minimum gradient search zone

Figure 4: Results with $K = 300$ and $m = 30$ for various gradient preconditionings.

Figure 4 illustrates different sizes of correction window, from none to a large one, through the

[4]They output a binary file with extension `.dat`: most standard image formats do not support 32-bit integers for pixels, except TIFF. However, very few image viewers support TIFF 32-bit.

prescribed 3×3 size. Little difference is noticeable; on the tail of the spiral, the 3×3 initialization shows better color fidelity. However the 10×10 initialization is even less precise than the standard initialization in this zone. The adverse effect of a large radius can be explained as follows: as soon as $g > S/2$, the potential shift regions of adjacent superpixels intersect. A low pixel gradient in this intersection can attract both superpixels to the same position, which is inefficient. The default choice in our implementation is a value $g = 0$, meaning no gradient-guided shift, though it can be changed by the user.

5.3 Performance

5.3.1 Runtime

The runtime of a superpixel algorithm is of utmost importance, as it is a pre-processing algorithm. Figure 5 shows the runtime for the SLIC algorithm itself and the connectivity enforcement procedure. These are measured on an Intel Core i7 CPU @ 2.60 GHz, the C++ code being compiled with GCC 7.5 under Linux. It can be observed that the connectivity step is always a fraction of the SLIC step. The general trend is a decreasing processing time with respect to K. For large images with proportionally few superpixels, the runtime is too large as many iterations are necessary before the superpixels stabilize. However, even for 16 Mpixels, the runtime goes below 10 s, which is probably small compared to further processing.

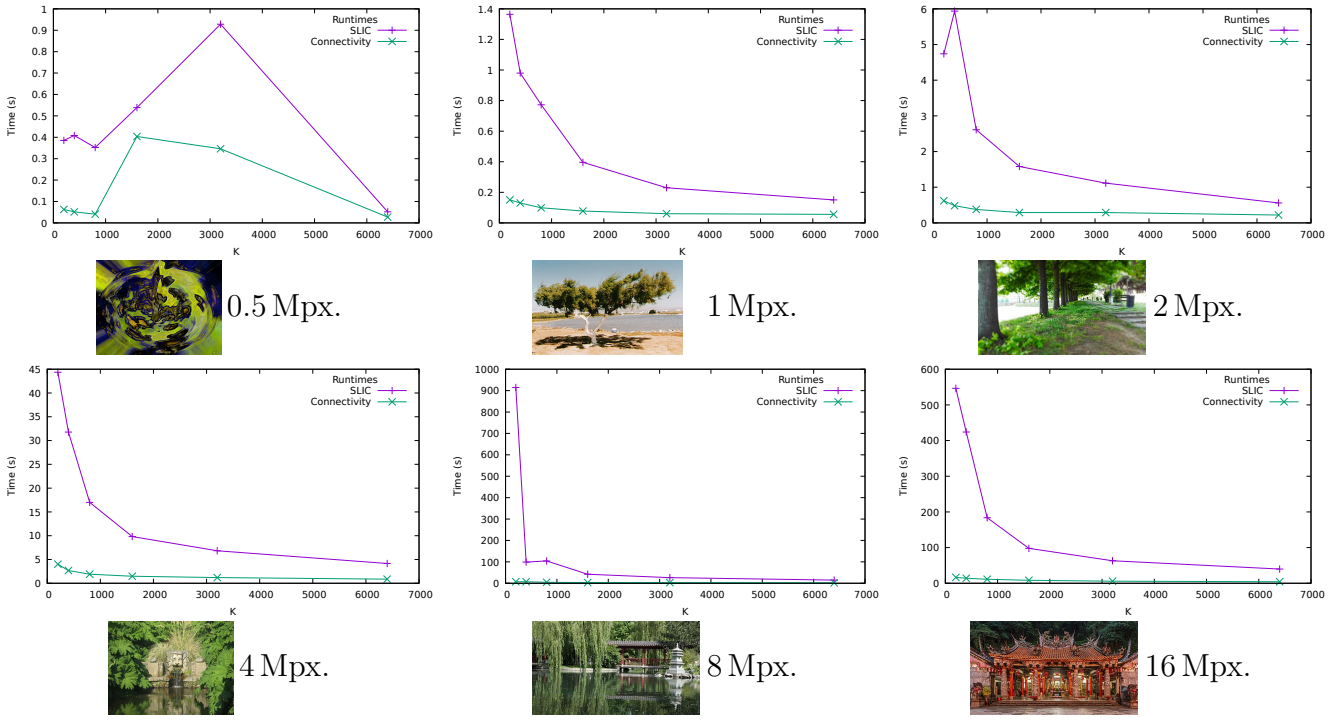


Figure 5: Runtime as a function of K, for different image resolutions ($0.5 \ldots 16$ Mpx). Compactness parameter $m = 40$.

5.3.2 Convergence

There is no theoretical guarantee of convergence of the K-means procedure, though in practice a few iterations are sufficient (precise measures appear below). Still, we ran over a pathological case (one among several thousands of images) where the SLIC repeats the same periodic pattern indefinitely after 354 iterations, see Figure 6. To have a finite runtime, our code limits to 1000 iterations. If the algorithm stops because of this limit, a warning message is displayed.

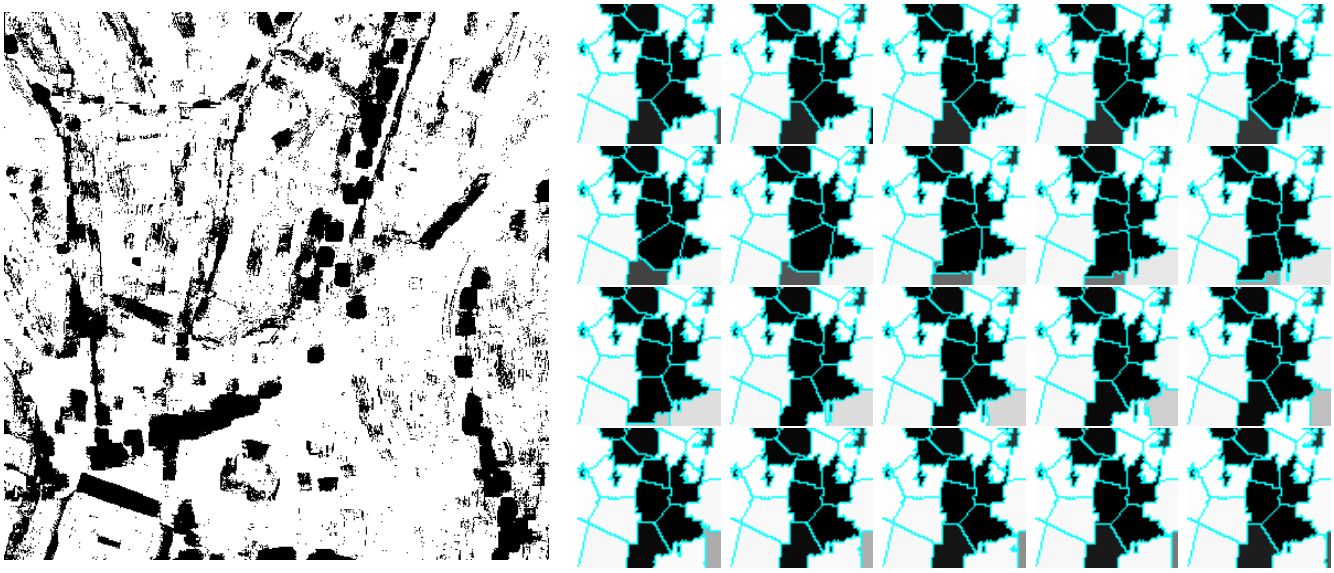


Figure 6: A pathological binary image sending the K-means into a periodic loop of 19 images after the first 354 iterations. Left: original image. Right, from left to right and top to bottom: the superpixels at bottom-right corner after 355 to 374 iterations. At iteration 374, the superpixels are the same as at iteration 355. The segmentation remains identical in the rest of the image.

5.4 Comparison with the Original Implementation

The code from the authors of [1] is generally faster than ours, since it systematically stops after ten loops. This allows it to be fast whatever the size of the image. For example, a 16 Mpx image can take over a minute of treatment with our program, against a few seconds with theirs. Regarding precision, we noticed a few shortcomings in their program. As can be seen on Figure 7, our superpixels adapt better to the contours of the guitar head's top and body.

Images treated by both algorithms are presented in Figure 8 for comparison. Note that the original version displays the original image in the background of the output image, while our version has been programmed to return an image in which the superpixels are displayed in their average color. Their implementation computes the Euclidean color distance in Lab color space instead of RGB for ours. This color space is closer to human perception, but it is questionable to what extent it improves the results.

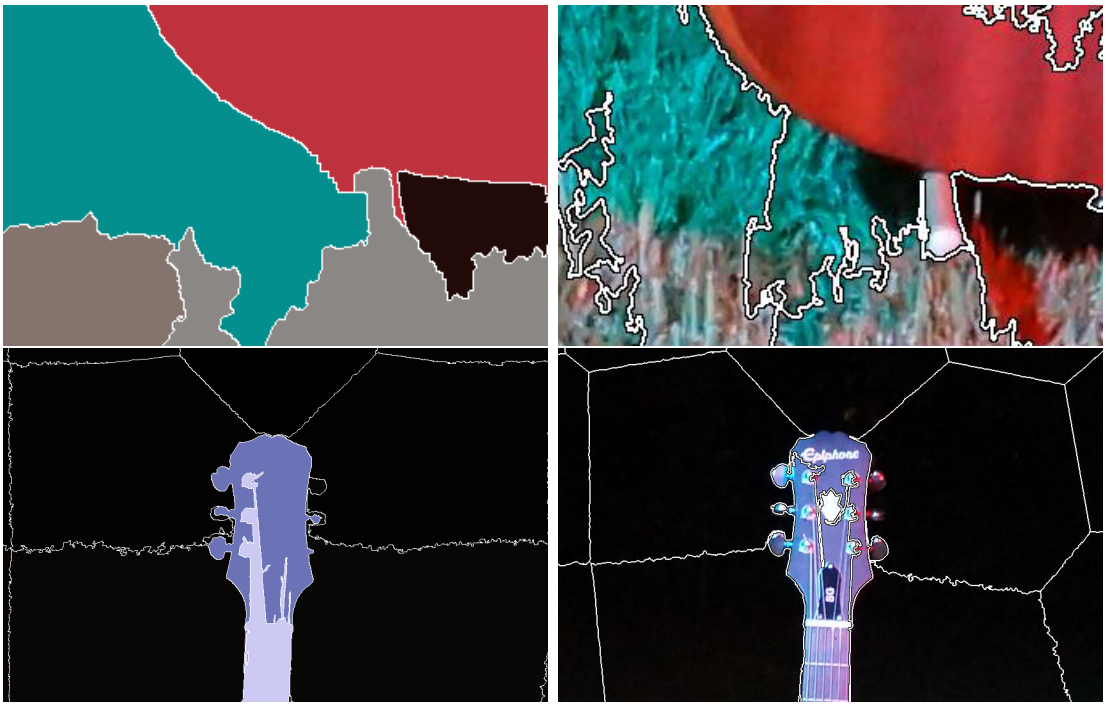
5.5 Benchmark

Quantitative results can be computed and they are at the basis of superpixel benchmarks, such as [15] and [11]. Superpixels are supposed to constitute an over-segmentation of an image and among the multiple metrics used to measure its quality, two stand out: boundary recall and under-segmentation error. They are relative to a ground truth segmentation $G = \{G_i\}$, a partition of the set of pixels. The computed superpixels are in the partition $S = \{S_j\}$. Ideally, each G_i is a union of some S_j and we measure the deviation from this.

5.5.1 Metrics

The first measure, boundary recall, measures the proportion of boundary pixels of G_i being boundary pixels of some S_j, within a tolerance of one pixel. A boundary pixel $p \in G_i$ is such that one of its

(a) Original image



(b) Our version of SLIC (c) Original SLIC

Figure 7: Zooms on an image treated by the two versions of SLIC with the same parameters.

four neighbors $q \in G_k$ with $i \neq k$. Noting ∂G_i the boundary pixels of G_i, we get

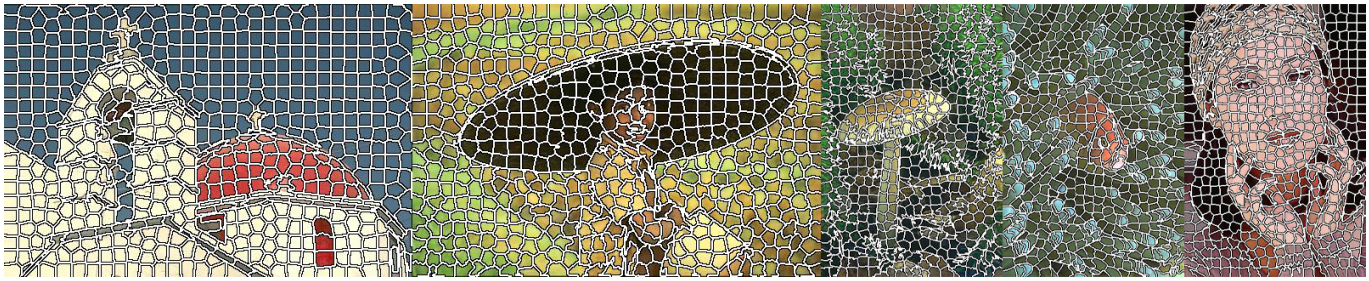
$$BR = \frac{\sum_i \sum_{p \in \partial G_i} \mathbf{1}[\exists j, \exists q \in \partial S_j : \|q - p\|_\infty \leq 1]}{\sum_i |\partial G_i|}. \tag{14}$$

We have $0 \leq BR \leq 1$ and higher is better.

The second measure, under-segmentation error, indicates the amount of leakage of superpixels with respect to G. The principle is to "assign" each S_j to one G_i and count the pixels of S_j outside G_i. The principle used by Van den Bergh et al. [13] is to assign S_j to the G_i with maximal overlap, leading to the formula

$$UE' = \frac{1}{w\,h} \sum_j \left| S_j \setminus \arg\max_{G_i \in G} |S_j \cap G_i| \right|. \tag{15}$$

(a) Original pictures



(b) Original authors' SLIC program

(c) Our SLIC program

Figure 8: Results of both versions of SLIC on images from [8], ($K = 600$, $m = 30$).

An alternative formula due to Neubert and Protzel [9], and the one we use, is

$$UE = \frac{1}{w\,h} \sum_{i,j} \min(|S_j \cap G_i|, |S_j \setminus G_i|). \tag{16}$$

To interpret such formula, notice that summing the two terms of the minimum yields $|S_j|$, hence one is at least $|S_j|/2$. Suppose there is some G_i such that $|S_j \cap G_i| > |S_j|/2$. Then S_j is assigned to G_i and the contribution of S_j to the sum is

$$|S_j \setminus G_i| + \sum_{k \neq i} |S_j \cap G_k| = 2\,|S_j \setminus G_i| < |S_j|. \tag{17}$$

Each pixel leaking out of G_i is counted twice. On the contrary, if no G_i contains more than half the pixels of S_j, the result of the minimum is $|S_j \cap G_i|$, which yields $|S_j|$ when summed over all i. This means S_j has no assignment G_i and all its pixels leak. Notice that $0 \leq UE \leq 1$ and lower is better.

Finally, a third measure independent of the ground truth is the explained variation, the relative variance change when each pixel is replaced by the mean value of its superpixel

$$EV = \frac{\sum_j |S_j|\,\|\bar{S}_j - \bar{I}\|^2}{\sum_P \|I(p) - \bar{I}\|^2}, \tag{18}$$

where $\bar{X}$ denotes the average of the image over the set of pixels in X. We put a norm here because the image may be in color. We have $0 \leq EV \leq 1$ and higher is better. The upper bound of 1 can be proved by a convexity argument.

5.5.2 Benchmark Protocol

We use the Berkeley Segmentation Dataset [8] (BSDS), a collection of 500 small size images (481×321), among which 200 are in the test section. These come with user-annotated segmentations, at least five for each image (one image has only four segmentations). The score is computed following [11]. For each image in the test set, the worst measures, that is, lowest BR, higher UE, and lowest EV, when comparing the superpixel segmentation to the five ground truth segmentations is recorded. This score is averaged over all images of the test set. This is performed for 18 values of K, from 200 to 5200, yielding a curve for BR, UE, and EV. The area under the curve (AUC) for $1-$BR, UE and $1-$EV gives finally a compound score for each criterion, where lower is better.

5.5.3 Comparison of Implementations

The code associated to [11] includes the evaluation routines along with wrappers for 28 superpixel algorithms. Among them, three implementations of SLIC are included: the original one from [1], the reimplementation in VLFeat [14] and a fast variant, preemptive SLIC [10]. This allows a quantitative comparison of the implementations. The original implementation contains a connectivity enforcement function, not mentioned in the article, that relabels pixels following Algorithm 10. Notice that, contrary to our method, there is no guarantee that the number of superpixels cannot increase after relabeling. For instance, consider initially $K = 2$ superpixels, with two connected components of $5/16 \times w\,h$ pixels for the first one, and two connected components of respective size $5/16 \times w\,h$ and $1/16 \times w\,h$ pixels for the second one. Since $5/16 \times w\,h > 4/16 \times w\,h = 1/2 \times w\,h/K$, we get after relabeling $K' = 3 > K$ superpixels.

Algorithm 10: Connectivity enforcement in the code of [1].

Input: Map ℓ giving for each pixel the index k of its assigned superpixels
Output: Corrected map ℓ' ensuring connected superpixels
$\ell'(.) \leftarrow -1$ `// Invalid label`
foreach *pixel* p **do**
  if $\ell'(p) < 0$ **then**
    $C \leftarrow cc(\ell = \ell(p), p)$ `// Connected component of iso-level containing` p
    if $|C| > \frac{1}{2}\frac{w\,h}{K}$ **then**
      $k \leftarrow$ new label
    else
      $k \leftarrow \min(0, \max_{q \sim p} \ell'(q))$ `// Existing label` $k \geq 0$ `of 4-neighbor of` p
    $\ell'(C) \leftarrow$ k `// Put label to all pixels of` C
return ℓ'

The implementation of VLFeat has a similar function but the threshold for the number of pixels is user-defined and defaults to 1, which does not enforce connectivity. It is important to set it as the same value as the original SLIC: by default, the benchmark code of [11] relabels each connected component with a unique label, which increases greatly the number of superpixels and the metrics do not make sense any more.

To perform our tests, we use a fork of D. Stutz's code[5], which allows the build with recent versions of OpenCV and fixes a few bugs: measure of AUC was faulty and led even to NaN ("Not a Number") in some situations; the results on the last image of the dataset were not incorporated in the statistics.

[5]Original: https://github.com/davidstutz/superpixel-benchmark, fork: https://github.com/pmonasse/superpixel-benchmark

Table 1: Metrics (see Section 5.5.1) for different SLIC implementations.

Implementation	AUC(1−BR)	AUC(UE)	AUC(1−EV)
Achanta et al. [1]	10.9	7.88	10.1
preSLIC [10]	13.0	8.22	10.6
vlSLIC [14]	11.7	9.09	12.8
Ours	**10.8**	**7.64**	**8.85**

Table 1 and Figure 9 show the metrics for the different implementations. In terms of BR, vlSLIC and especially preSLIC have worse performance compared to the other two, which are on par. Concerning UE, the best is our implementation and the worst vlSLIC. Finally, concerning EV we get a significant best performance for our implementation and the worst for vlSLIC, original implementation and preSLIC yielding intermediate results.

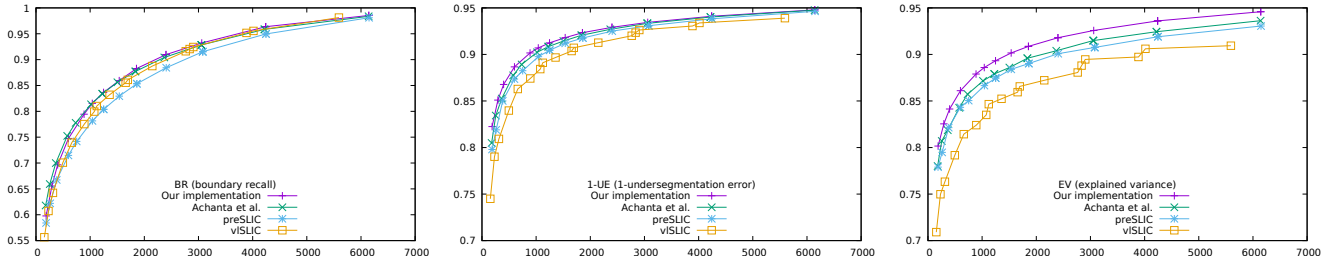


Figure 9: Quantitative comparison of four implementations of SLIC in terms of BR, 1−UE and EV (higher is better). We took the default value of the original implementation $m = 40$.

Table 2: Metrics (see Section 5.5.1) for different stopping criteria. X_k^i is the center of superpixel k at iteration number i.

Stopping criterion	AUC(1−BR)	AUC(UE)	AUC(1−EV)
$\max_k \|X_k^i - X_k^{i-1}\| = 0$	10.9	**7.44**	**8.32**
$\max_k \|X_k^i - X_k^{i-1}\| < 0.5$	10.9	7.45	8.34
$\sqrt{\sum_k \|X_k^i - X_k^{i-1}\|^2/K} < 0.5$	**10.8**	7.64	8.85
10 iterations	10.9	7.52	8.49

We also tested several stopping criteria in our implementation, depending on the relative motion of superpixels compared to previous iteration: we note X_k^i the position of superpixel of index k at iteration number i.

- No change ($\max_k \|X_k^i - X_k^{i-1}\| = 0$), capped at 1000 iterations.
- Maximum motion below 0.5 pixel ($\max_k \|X_k^i - X_k^{i-1}\| < 0.5$).
- Root mean square motion below 0.5 pixel ($\sqrt{\sum_k \|X_k^i - X_k^{i-1}\|^2/K} < 0.5$).
- 10 iterations (like the original implementation).

The actual number of iterations for the first three cases appears in Figure 10 and the metrics in Table 2. The first effect to note is that the number of iterations decreases with the number of superpixels. Normally, the algorithm would need to run until convergence, that is, there is no motion of superpixels anymore. Actually, the cap of 1000 iterations is reached in a few cases. Even discarding these pathological images, the number of iterations can be several hundreds for a low

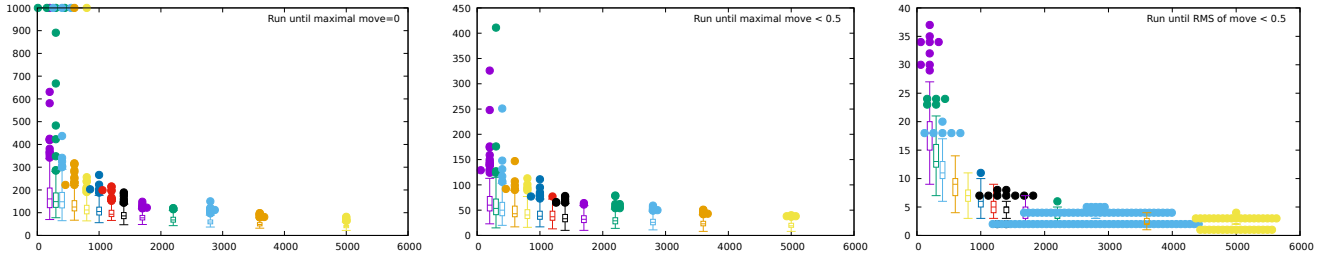


Figure 10: Number of iterations on BSDS test set, as a function of K, according to the stopping criterion.

number of superpixels, which is too long: Remember that superpixel extraction is assumed to be a fast preprocessing task. Relaxing this condition to imposing that the maximum move of a superpixel is below 0.5 pixel still leads to good metrics but does not prevent a high number of iterations. Finally, imposing the root mean square of superpixel motions being less than 0.5 pixel is an attractive target: it always stops quickly, and in less than 10 iterations for a high number of superpixels. This is actually the fastest alternative, but at the cost of a slightly lower quality. The choice of the original implementation to stop at 10 iterations by default is a good compromise between speed and quality.

6 Conclusion

Despite the numerous alternative algorithms proposed since the publication of SLIC (first preprint appearing in 2010) [15], it remains the reference due to its simplicity, good adherence to contours and fast runtime performance. Fast processing is a prime requirement for superpixels, but several metrics are also used to compare the results. The requirement of connected superpixels is not ensured by the SLIC in itself, but results from a post-processing, not described (though implemented) in the original article. We proposed an alternative one and we showed that it yields better quantitative results.

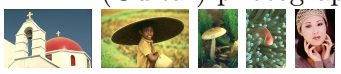
Image Credits

 (Sassy) The Big Lez Show by Jarrad Wright CC-BY-SA 3.0 (2012)

 (Spiral) photograph by Thomas Williamson and Robin Gay, CC-BY-SA 4.0 (2017)

 (Pond) photograph by Robin Gay, CC-BY-SA 4.0 (2017)

 (Guitar) photograph by Thomas Williamson, CC-BY-SA 4.0 (2017)

 (Castle, Boy, Mushroom, Fish, Woman) Berkeley Segmentation Dataset [8][6]

 (Green background) by Ribaisu, CC-BY-SA 4.0 (2004)[7]

 (Naxos tree) by ERWEH, CC-BY-SA 2.0 (2004)[8]

[6] https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html. Images #118035, #189011, #208001, #210088, #302003.

[7] https://commons.wikimedia.org/wiki/File:Green_Background.png

[8] https://commons.wikimedia.org/wiki/File:Naxos-Tree.imp-1.jpg

 (Background) by Pasqualino Ubaldini, CC-BY-SA 4.0 (2016)[9]

 (Jardin botanique Vauville) by Suzelfe, CC-BY-SA 3.0 (2008)[10]

 (Chinagarten) by Burkhard Mücke, CC-BY-SA 4.0 (2017)[11]

 (Quanhua temple, Taiwan) by CEphoto, Uwe Aranas (2015)[12]

References

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, *SLIC Superpixels Compared to State-of-the-art Superpixel Methods*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34 (2011), pp. 2274–2282. https://doi.org/10.1109/TPAMI.2012.120.

[2] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik, *Multiscale combinatorial grouping*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 328–335. https://doi.org/10.1109/CVPR.2014.49.

[3] A. Bódis-Szomorú, H. Riemenschneider, and L. Van Gool, *Superpixel meshes for fast edge-preserving surface reconstruction*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 2011–2020. https://doi.org/10.1109/CVPR.2015.7298812.

[4] E.W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische mathematik, 1 (1959), pp. 269–271. https://doi.org/10.1007/BF01386390.

[5] S. Gould, J. Rodgers, D. Cohen, G. Elidan, and D. Koller, *Multi-class segmentation with relative location prior*, International Journal of Computer Vision, 80 (2008), pp. 300–316. https://doi.org/10.1007/s11263-008-0140-x.

[6] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, *The connected-component labeling problem: A review of state-of-the-art algorithms*, Pattern Recognition, 70 (2017), pp. 25–43. https://doi.org/10.1016/j.patcog.2017.04.018.

[7] A. Levinshtein, A. Stere, K. Kutulakos, D. Fleet, S. Dickinson, and K. Siddiqi, *Turbopixels: Fast superpixels using geometric flows*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2009), pp. 2290–2297. https://doi.org/10.1109/TPAMI.2009.96.

[8] D. Martin, C. Fowlkes, D. Tal, and J. Malik, *A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics*, in International Conference on Computer Vision (ICCV), vol. 2, July 2001, pp. 416–423. https://doi.org/10.1109/ICCV.2001.937655.

[9] P. Neubert and P. Protzel, *Superpixel benchmark and comparison*, in Forum Bildverarbeitung, vol. 6, 2012, pp. 205–218.

[9] https://commons.wikimedia.org/wiki/File:Background_1.jpg
[10] https://commons.wikimedia.org/wiki/File:JardinBotaniqueVauville-T%C3%AAteDeLion.JPG
[11] https://commons.wikimedia.org/wiki/File:Chinagarten_IGA_2017_03.jpg
[12] https://commons.wikimedia.org/wiki/File:Miaoli-County_Taiwan_Quanhua-Temple-03.jpg

[10] ——, *Compact watershed and preemptive SLIC: On improving trade-offs of superpixel segmentation algorithms*, in International Conference on Pattern Recognition (ICPR), IEEE, 2014, pp. 996–1001. https://doi.org/10.1109/ICPR.2014.181.

[11] D. Stutz, A. Hermans, and B. Leibe, *Superpixels: An evaluation of the state-of-the-art*, Computer Vision and Image Understanding, 166 (2018), pp. 1–27. https://doi.org/10.1016/j.cviu.2017.03.007.

[12] C. Tomasi and R. Manduchi, *Bilateral filtering for gray and color images*, in International Conference on Computer Vision (ICCV), vol. 1, IEEE, 1998, pp. 839–846. https://doi.org/10.1109/ICCV.1998.710815.

[13] M. Van den Bergh, X. Boix, G. Roig, and L. Van Gool, *SEEDS: Superpixels extracted via energy-driven sampling*, International Journal of Computer Vision, 111 (2015), pp. 298–314. https://dx.doi.org/10.1007/s11263-014-0744-2.

[14] A. Vedaldi and B. Fulkerson, *VLFeat: An open and portable library of computer vision algorithms*. https://www.vlfeat.org/, 2008.

[15] M. Wang, X. Liu, Y. Gao, X. Ma, and N.Q. Soomro, *Superpixel segmentation: A benchmark*, Signal Processing: Image Communication, 56 (2017), pp. 28–39. https://dx.doi.org/10.1016/j.image.2017.04.007.

[16] E. Weiszfeld, *Sur le point pour lequel la somme des distances de n points donnés est minimum*, Tohoku Mathematical Journal, First Series, 43 (1937), pp. 355–386.

[17] S.J. Wright, *Coordinate descent algorithms*, Mathematical Programming, 151 (2015), pp. 3–34. https://doi.org/10.1007/s10107-015-0892-3.

[18] K. Yamaguchi, D. McAllester, and R. Urtasun, *Efficient joint segmentation, occlusion labeling, stereo and flow estimation*, in European Conference on Computer Vision (ECCV), Springer, 2014, pp. 756–771. https://doi.org/10.1007/978-3-319-10602-1_49.

[19] F. Yang, H. Lu, and M-H. Yang, *Robust superpixel tracking*, IEEE Transactions on Image Processing, 23 (2014), pp. 1639–1651. https://doi.org/10.1109/TIP.2014.2300823.