EECE 680D PROJECT – A Python/OpenCV Implementation of Manifold SLIC: A fast Method to Compute Content-Sensitive Super-pixels

Souleymane Sow

Thomas J. Watson School of Engineering and Applied Science
Binghamton University - State University of New York

Abstract

Superpixels are an important part in image processing tasks such as segmentation. It helps make the whole process more efficient. Superpixel is a group of connected pixels that can perceptually highlight image features. A widely used method to compute uniform superpixels known as Simple Linear Iterative Clustering (SLIC) has proven to be efficient. In this study, Manifold SLIC[1] method an extension to SLIC that compute content-sensitive superpixels to yield small superpixels in content-dense region and large superpixels in content-sparse region, has been implemented in Python/OpenCV using standard dataset. Comparison has been made between this method and SLIC using Lena and BSDS benchmark. Computation results and source code have been shared on GitHub. Results have shown that Manifold SLIC is efficient, content sensitive, fast comparable to SLIC. Therefore, we can argue that Manifold SLIC is 10 times faster than current state-of-the-art content sensitive superpixels computation methods.

1. Introduction

Computer vision is an emerging field driven by advancement in machine learning and data science. Superpixels, a group of connected pixels that can perceptually capture image features, is an important preprocessing step in most computer vision process such as image segmentation, contour closure, object tracking, etc…

Two most important groups of algorithms used to compute superpixels are graph-based methods and clustering-based methods. In graph-based methods, images are represented by graph with pixels at nodes and solution is found by minimizing a cost function on the graph. Normalized Cuts(NC), Felzenszwalb-Huttenlocher(FH), Superpixel lattices(SL), GraphCut are most known graph-based methods. Moreover, clustering-based methods group pixels into clusters and iteratively refine them until a convergence criteria is met. The algorithm start by initializing with a set of evenly distributed seeds $\{s_i\}_{i=1}^{K}$ on image domain I. Turbopixels, Simple Linear Iterative Clustering(SLIC), Structure Sensitive Superpixels(SSS) are the best known.

Turbopixels generates a geometric flow for each seed and propagate them using level set method. Superpixels set boundary is the points of intersection of two flows. SLIC use an adaption of K-means and cluster pixels in a 5-D Euclidean Space with combined location and color. Each pixel is assigned to nearest seed and iteratively update cluster center using Centroidal Voronoi tessellation (CVT). This algorithm yields uniform superpixels. Content Sensitive Superpixel SSS algorithm use geometric flow method into a CVT optimization framework. This methods use geodesic distance instead of Euclidean distance to effectively capture the non-homogenous feature in images to have small superpixels in content-dense region and large superpixels in content-sparse regions.

However, most algorithms have some limitations. NC, SL does not adhere to image boundaries and NC has high computational cost. FH produce superpixels with irregular size and shapes. Turbopixel run slow on real-world datasets even with a theoretical linear time complexity O(N). Even though it is content-sensitive, SSS is computationally expensive while SLIC produce uniform superpixel everywhere with efficiency. This is due to the fact with SLIC distance is measured in Euclidean space and use local search on to improve performance while with SSS geodesic distance is used to measure distance to effectively capture non-homogenous feature in image. The need to improve SLIC algorithm to be content-sensitive was the idea behind Manifold SLIC.

Manifold SLIC support fast computation of content-sensitive superpixels compared to SSS. It combines simplicity and high performance of SLIC and the content-aware nature of SSS. Clustering are computed on 2D manifold M embedded in the combined color and image space $\mathbb{R}^5$ and area elements on M are good measure of content density in I. An efficient algorithm has been developed to compute Restricted CVT, a uniform tesselation on M to induce content-sensitive superpixels characterized by measuring areas of Voronoi cells on M. This makes Manifold SLIC low cost and 10 times faster than state-of-the-art content-sensitive superpixel algorithm [1]. In addition to restricted CVT, locally updating voronoi cells with Splitting and Merging operators help meet K requirement. This will split content rich regions and merge content sparse regions. In this study, we will implement Manifold SLIC algorithm using Python/OpenCV and compute Manifold SLIC for Lena and BSDS benchmark datasets, contrast content-sensitivity and compare runtime performance to that of SLIC.

2. SLIC preliminaries

In this part, we will attempt to summarize SLIC and SSS concept however detailed descriptions are available in [1].

2.1. SLIC

Given an input image I with N pixels, for any pixel p ∈ I, c(p) is the representation of the color of p in the CIELAB color space and c(p)=(l(p), a(p), b(p)).

Given two pixels p1, p2 such that p1=(u1, v1) and p2=(u2, v2), the normalized Euclidean distance between p1 and p2 in the combined color and location space $\mathbb{R}^5$ is:

$$D(p1,p2) = \sqrt{\left(\frac{ds}{Ns}\right)^2 + \left(\frac{dc}{Nc}\right)^2} \qquad (1)$$

where N_c *and* N_s are two constants,

$$ds = ||p1 - p2||_2 = \sqrt{(u1-u2)^2 + (v1-v2)^2}$$

$$dc = ||c(p1) - c(p2)||_2$$
$$= \sqrt{\left(l(p1)-l(p2)\right)^2 + \left(a(p1)-a(p2)\right)^2 + \left(b(p1)-b(p2)\right)^2}$$

Given a set $\{s_i\}_{i=1}^K$ of evenly distributed seeds in the image, SLIC partitions I using Voronoi diagram $\{\vartheta(s_i)\}_{i=1}^K$ and then iteratively improves the Voronoi cells by moving the seeds to their centers of mass to become the so-called centroidal Voronoi tessellation(CVT).

Mathematical definition of CVT follows: Let ρ: I → ℝ+ be a density function defined on I. For a Voronoi cell $\{\vartheta(s_i)\}_{i=1}^K$, its center of mass c_i is given by:

$$c_i = \frac{\int_{x\in\vartheta(s_i)} x\rho(x)dx}{\int_{x\in\vartheta(s_i)} \rho(x)dx} \qquad (2)$$

The Voronoi tessellation $\{\vartheta(s_i)\}_{i=1}^K$ is a CVT if, for each Voronoi cell, the center of mass coincides with the seed that is $c_i = s_i, 1 < i \leq K.$ Du et al. [5] showed that a CVT is the minimizer of the following energy E

$$E(\{s_i\}_{i=1}^K, \{\vartheta(s_i)\}_{i=1}^K) = \sum_{i=1}^K \int_{x\in\vartheta(s_i)} \rho(x)d(x,s_i)dx \qquad (3)$$

where $d(x,s_i)$ measures the distance between x and y. They also showed that the solution always exists.

In SLIC algorithm, $d(x,g_i)$ = $D(x,g_i)$ defined in Eqn. (1) and computes CVT using the Lloyd method, which iteratively moves each seed to the corresponding center of mass. We outline SLIC in Algorithm 1 which adopts two heuristic strategies. First, rather than explicitly constructing the Voronoi cells, for each seed s_i, it computes distances from s_i to the pixels within a 2S× 2S window centered at s_i (lines 6-13) and then assigns a pixel to the seed with the least distance. This local search also distinguishes SLIC from the conventional K-means method, which has to compute the distances between s_i and every pixel in the image. Second, it does not compute the center of mass using the area integral in Eqn. (2). Instead, it finds the centroid c_i as the mean vector of all the pixels belonging to the Voronoi cell (line 16). Computational results show that SLIC converges in only 10 iterations for real-world images [6]. As a result, SLIC has an $O(N)$ time complexity, which is independent of the number of superpixels K. In contrast, the conventional K-means method runs in $O(KNN_{iter})$, where N_{iter} is the number of iterations.

Algorithm 1 SLIC [6]

Input: An image I of N pixels, the desired number of superpixels K, the maximal number of iterations $iter_{max}$ and the convergence threshold ε.
Output: K superpixels of similar sizes.

```
1: Initialize seeds $\{s_i\}_{i=1}^{K}$ at regular grids with step length $S = \sqrt{\frac{N}{K}}$.
2: Move each seed to the lowest gradient position in a 3x3 neighborhood.
3: Initialize label $l(p) = -1$ and distance $d(p) = \infty$ for each pixel $p$.
4: Initialize the residual error $err = \infty$ and $iter = 0$.
5: while $err > \varepsilon$ and $iter \leq iter_{max}$ do
6:     for each seed $s_i$ do
7:         for each pixel $p$ in a 2S × 2S region around $s_i$ do
8:             Compute the distance D= D($s_i$, p) between $s_i$ and $p$.
9:             if $D < D(p)$ then
10:                $d(p) = D;\ l(p) = i$
11:            end if
12:        end for
13:    end for
14:    $err = 0$
15:    for each $seed\ s_i$ do
16:        Compute average $c_i$ of pixels in the cluster of $s_i$.
17:        $err$ += ‖ $c_i$ — $s_i$ ‖$^2$
18:        $s_i = c_i$
19:    end for
20:    $iter++$
21: end while
```

2.2. SSS

Even though SLIC is conceptually simple and highly efficient due to the constant CVT density function $\rho(x) \equiv 1\ for\ any\ x \in I$, it produces uniform superpixels where size remain same everywhere. In order to highlight content change in real-world images, SSS algorithm computes content-sensitive superpixels that are small superpixels in content-dense regions and large superpixels in content-sparse regions. Content- sensitive superpixels were computed by Wang et al. [3] using geometric flow [9] into the CVT optimization framework. In contrast to SLIC that uses Euclidean distances, SSS adopts geodesic distances defined as follows:

$$d(p_1, p_2) = \min_{P_{p_1,p_2}} \int_0^1 D(P_{p_1,p_2}(t)||\dot{P}_{p_1,p_2}(t)||dt \qquad (4)$$

where $P_{p_1,p_2}(t)$ is a parameterized path connecting pixels $p_1(t = 0)\ and\ p_2\ (t = 1)$. The weight function D($z$) penalizes image boundaries [9],

$$D(z) = e^{\frac{edge(z)}{v}}, \qquad edge(z) = \frac{||\nabla I||}{G_\sigma||\nabla I|| + \gamma} \qquad (5)$$

where v is a scaling parameter, $edge(z)$ is a normalized edge measurement, G_σ is the Gaussian functions with the standard deviation σ, and γ is a small constant to alleviate the effect of weak intensity boundary. Compared with Euclidean distances in Eqn. (1), geodesic distances in Eqn. (4) penalize image boundary, hereby are an effective measure of content-sensitivity. Computational results show that Wang et al.'s method [3] can compute high-quality content-sensitive superpixels that exhibit high boundary adherence. However, since it is computationally expensive to measure geodesic distances, their method is among the slowest algorithms in our experiments.

3. Overview

Manifold SLIC was extended to SLIC to make it an efficient content-sensitive superpixels computation algorithm.

Similar to SLIC [6], the color is represented in the CIELAB color space and denote by $c(u,v) = (l(u,v), a(u,v), b(u,v))$ the color at the pixel (u,v) in the image I [1]. Then we define a stretching map $\Phi : \rightarrow \mathbb{R}^5$ to map pixels to a 2-manifold $\mathcal{M}$ embedded in the 5-dimensional combined image and color space,

$$\begin{aligned}\Phi(u,v) &= (\lambda_1 p, \lambda_2 c) \\ &= (\lambda_1 u, \lambda_1 v, \lambda_2 l, \lambda_2 a, \lambda_2 b), \qquad (6)\end{aligned}$$

where $\lambda_1\ and\ \lambda_2$ are global stretching factors. We set $\lambda_1 = \frac{1}{N_s}, \lambda_2 = \frac{1}{N_c}$ and adopt the SLIC metric D in Eqn. (1) to measure the distance between points on manifold $\mathcal{M}$,

$$D(\Phi(p_1), \Phi(p_2)) = \sqrt{\lambda_1^2 d_s^2 + \lambda_2^2 d_c^2} \qquad (7)$$

For a pixel $p = (u,v)$, we denote by $\square_p$ the unit square 1 × 1 centered at p. $Let\ p_1, p_2, p_3, p_4$ be the four corners of $\square_p$, each of which is determined by the average of the four neighboring pixels. The square $\square_p$ consists of two triangles, $\square_p = \triangle\ p_1p_2p_3 \cup \triangle\ p_3p_4p_1$. Then we approximate the area of the curved triangle $\Phi(\triangle\ p_1p_2p_3)$ by the area of planar triangle $\triangle\ \Phi(p_1)\Phi(p_2)\Phi(p_3)$,

$$Area(\Phi(\triangle\ p_1p_2p_3)) \approx \frac{1}{2} \| \overrightarrow{\Phi(p_2)\Phi(p_1)} \| * \| \overrightarrow{\Phi(p_2)\Phi(p_3)} * \sin(\theta), \qquad (8)$$

where θ is the angle between vectors $\overrightarrow{\Phi(p_2)\Phi(p_1)}$ and $\overrightarrow{\Phi(p_2)\Phi(p_3)}$. Note that the length $||\overrightarrow{\Phi(x)\Phi(y)}||$ is measured using the metric D in Eqn. (7). $Area(\Phi(\triangle\ p_3p_4p_1))$ can be computed in a similar way. See Figure 2 in [1]. The area of a region $\Phi(\Omega) \subset \mathcal{M}$ is simply the sum $of\ Area(\Phi(\square_{p_i}))$ for all pixels $p_i \in \Omega$.

This method is based on an important observation: for a region $\Phi(\Omega) \subset I \subset \mathbb{R}^2$, the area of the corresponding region $\Phi(\Omega) \subset M$ on $\mathcal{M}$ depends on both the area of Ω and the intensity or color variation in Ω. The higher the variation of colors in Ω, the larger the area of $\Phi(\Omega)$, and vice versa. If we can compute a uniform tessellation on $\mathcal{M}$, the inverse mapping Φ^{-1} will induce the content-sensitive tessellation on I [1]. See Figure 3 in [1]. Towards this goal, authors in [1] propose two simple yet effective techniques: computing restricted CVT (Section 4) and locally updating bad-shaped Voronoi cells (Section 5). We also outline the manifold SLIC in Algorithm 2 and highlight its differences to SLIC in blue [1].

Algorithm 2 Manifold SLIC [1]

Input: An image I of N pixels, the desired number of superpixels K, the maximal number of iterations $iter_{max}$ and the convergence threshold ε.
Output: K superpixels of similar sizes.

```
1: Initialize seeds $\{s_i\}_{i=1}^{K}$ at regular grids with step length $S = \sqrt{\frac{N}{K}}$.
2: Move each seed to the lowest gradient position in a 3x3 neighborhood.
3: Initialize label $l(p) = -1$ and distance $d(p) = \infty$ for each pixel $p$.
4: Initialize the residual seed dual error $err = \infty$ and $iter = 0$.
5: Compute Area($\Phi(\square_p)$) for each pixel $\in I$
6: Compute a local search range $\Xi = \frac{4\sum_{i=1}^{K} Area(\Phi(\square_{p_i}))}{K}$
7: while $err > \varepsilon$ and $iter \le iter_{max}$ do
8:     for each seed $s_i$ do
9:         Compute $Area(\Phi(\Omega(s_i)))$ of a $2S \times 2S$ region $\Omega(s_i)$ centered at $s_i$.
10:        Compute a scaling factor $\lambda_i = \sqrt{\frac{\Xi}{Area(\Phi(\Omega(s_i)))}}$.
11:        if iter > 0 and $\lambda_i < \tau$ and $Area(\vartheta_{\mathcal{M}}(\Phi(s_i))) > \Xi/4$ then
12:            Split Voronoi cell $\vartheta_{\mathcal{M}}(\Phi(s_i))$. (Section 5.1)
13:        end if
14:    end for
15:    Merge the Voronoi cells whose areas are small and whose seeds are close to each other. (Section 5.2)
16:    for each seed $s_i$ do
17:        for each pixel $p$ in a $2\lambda_i$ S × $2\lambda_i$ S region around $s_i$ do
18:            Compute the distance $D = D(\Phi(s_i), \Phi(p))$ using Eqn.(7).
19:            if $D < D(p)$ then
20:                $d(p) = D;\ l(p) = i$
21:            end if
22:        end for
23:    end for
24:    $err = 0$
25:    for each seed $s_i$ do
26:        Compute $Area(\vartheta_{\mathcal{M}}(\Phi(s_i)))$
27:        Compute the mass center $m_i$ of $\vartheta_{\mathcal{M}}(\Phi(s_i))$ using Eqn. (11) and set $c_i = P(m_i)$ using Eqn. (19) in [1]
28:        $err \mathrel{+}= \| c_i - s_i \|^2$
29:        $s_i = c_i$
30:    end for
31:    $iter++$
32: end while
```

4. Computing Restricted CVT

For best result, a trade-off between the quality of the content-sensitivity of superpixels and CVT computation complexity has been applied. Since it is expensive to compute exact CVTs on curved manifolds [7, 8, 9], the restricted centroidal Voronoi tessellation (RCVT) has been computed, which approximates the exact CVT well for K in a proper range, K >200 in images of various resolution [1].

4.1. Restricted Voronoi Diagram and RCVT

Let $S = \{s_i | s_i \in I, 1 \le i \le K\}$ be the set of seeds in the image I and $G = \{\Phi(s_i)\}_{i=1}^{K}$ =1 the corresponding generators on the manifold $\mathcal{M}$. The Euclidean Voronoi cell of a generator $\Phi(s_i)$, denoted by $\vartheta_{\mathbb{R}_5}$, is

$$\vartheta_{\mathbb{R}_5}(\Phi(s_i)) = \left\{ x \in \mathbb{R}_5 \,\middle|\, \|x - \Phi(s_i)\|_2 \le \left\|x - \Phi(s_j)\right\|_2, \forall j \ne i, \Phi(s_j) \in G \right\}$$

Restricting $\vartheta_{\mathbb{R}_5}$ to the manifold $\mathcal{M}$, we obtain the restricted Voronoi cell

$\mathcal{M}(\Phi(s_i)) \triangleq \mathcal{M} \cap \vartheta_{\mathbb{R}_5}(\Phi(s_i))$. The restricted Voronoi diagram $RVD(G, \mathcal{M})$ [4] is the collection of restricted Voronoi cells satisfying

$$RVD(G,\mathcal{M}) = \{\vartheta_{\mathcal{M}}(\Phi(s_i)) \ne \emptyset \mid \forall \Phi(s_i) \in G\} \qquad (10)$$

Edelsbrunner and Shah [4] showed that $RV\,D(G,\mathcal{M})$ is a finite closed covering of $\mathcal{M}$. For a restricted Voronoi cell $\vartheta_{\mathcal{M}}(\Phi(s_i))$, its mass centroid m_i is

$$m_i = \frac{\int_{r\in\vartheta_{\mathcal{M}}(\Phi(s_i))} r dr}{\int_{r\in\vartheta_{\mathcal{M}}(\Phi(s_i))} 1 dr} \qquad (11)$$

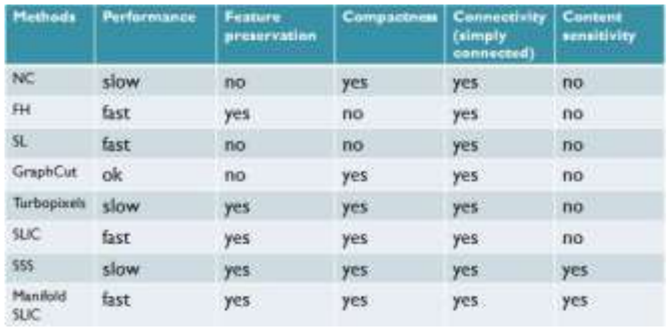
Then we define restricted CVT as follows:

Definition 1. Let $X = \{x_i \mid x_i \in \mathbb{R}^n, 1 \leq i \leq K\}$be a set of points in $\mathbb{R}^n$ and $\mathcal{M}$ be a 2-manifold embedded in $\mathbb{R}^n$. A restricted Voronoi diagram RV D(X,$\mathcal{M}$) is a restricted CVT (or RCVT) if each generator $x_i \in X$ is the mass centroid of its restricted Voronoi cell.

Theorem 1. Let $\mathcal{M}$ be a 2-manifold embedded in $\mathbb{R}^n\ and\ K \in \mathbb{Z}_+$ a positive integer. For an arbitrary set X of points $\{x_i\}_{i=1}^{K}\ in\ \mathbb{R}$ and an arbitrary $\{V_i\}_{i=1}^{K}$ on $\mathcal{M}$, $\cup_{i=1}^{K} V_i = \mathcal{M}$, $V_i \cap V_j = \emptyset$, $\forall\, i \neq j$, define the CVT energy functional as follows:

$$\mathcal{E}(\{(x_i,V_i)\}_{i=1}^{K}) = \sum_{i=1}^{K} \int_{r\in V_i} d^2(r,x_i)dr \qquad (12)$$

Then the necessary condition for $\mathcal{E}$ being minimized is that $\{(x_i,V_i)\}_{i=1}^{K}\ is\ a\ RCVT\ of \mathcal{M}$. Proof and further discussion on efficient and simple method to compute RCVT, locally updating Voronoi cells are in [1]. Methods discussed above are summarized in table 1.



Methods	Performance	Feature preservation	Compactness	Connectivity (simply connected)	Content sensitivity
NC	slow	no	yes	yes	no
FH	fast	yes	no	yes	no
SL	fast	no	no	yes	no
GraphCut	ok	no	yes	yes	no
Turbopixels	slow	yes	yes	yes	no
SLIC	fast	yes	yes	yes	no
SSS	slow	yes	yes	yes	yes
Manifold SLIC	fast	yes	yes	yes	yes

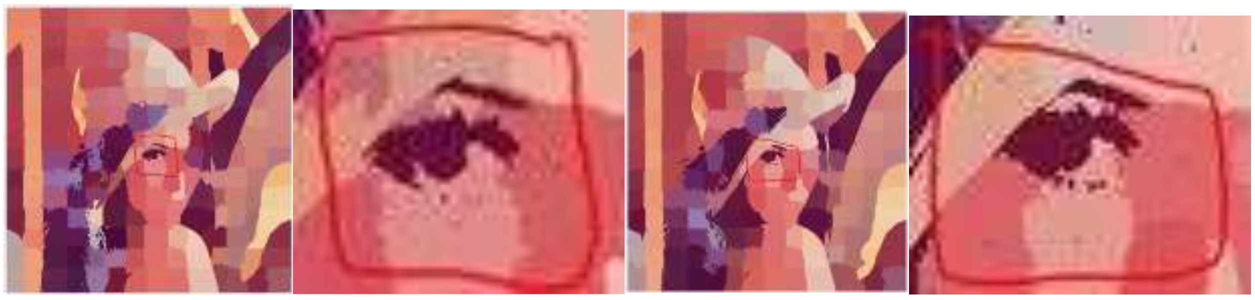
Table 1: Summary of different methods

5. Experimental results

5.1. Lena dataset: SLIC: K=200, M=40, Iter=10

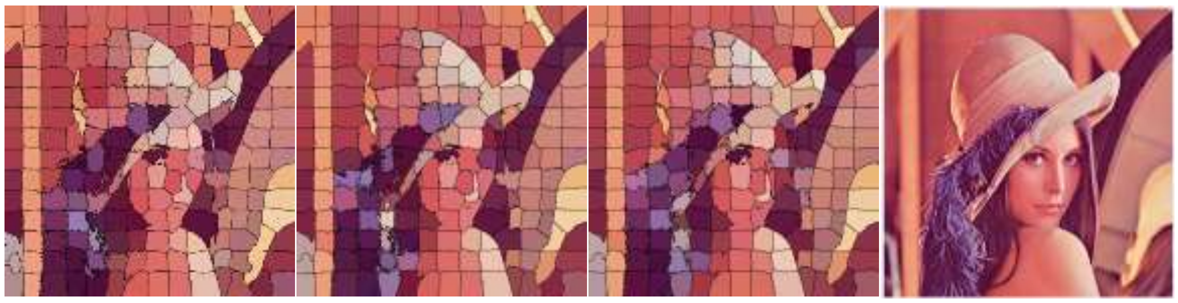
We implemented Manifold SLIC in Python/OpenCV on Spyder environment with Anaconda platform. We tests result

on standard datasets such as Lena and BSDS on GitHub see figures 1~5. Comparison and contrast between SLIC superpixels and Manifold SLIC superpixels showed that Manifold SLIC is not only efficient but content-sensitive.



(a) MSLIC (b) MSLIC (c) SLIC (d) SLIC

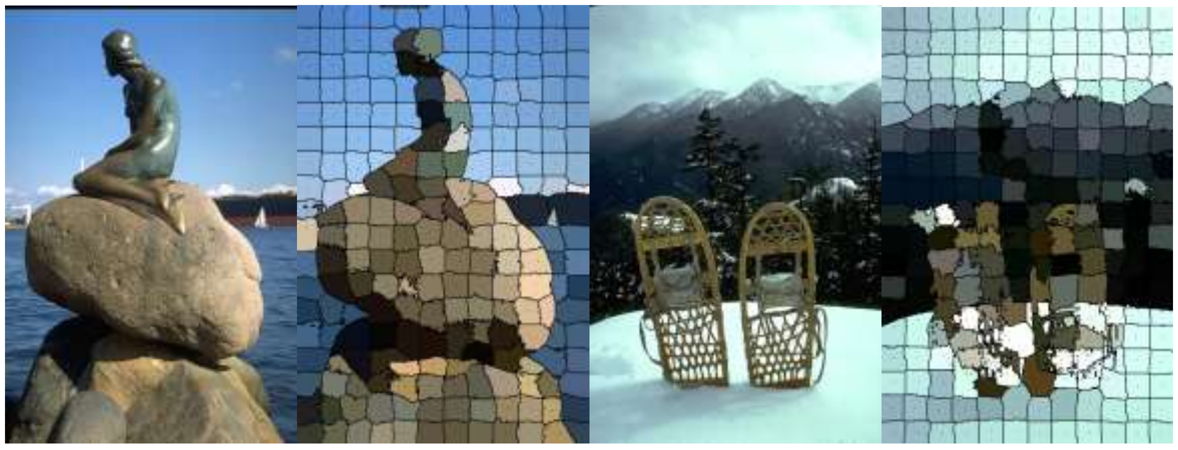
Figure 1. (a) and (b) show MSLIC is content sensitive compared to SLIC in (c) and (d)



(a) SLIC (b) MSLIC (c) S&M MSLIC (d) Original

Figure 2. Segmentation comparison between SLIC, MSLIC, and MSLIC split and merge applied. Compared to (a), (b) shows that Manifold SLIC is content-sensitive with smaller superpixels in content dense regions see left of the eye. Also comparing (c) to (b), we see that smaller area are being merged to neighboring cell, see top right.

5.2. BSDS500 dataset: SLIC: K=200, M=40, Iter=10



(a) Original (b) MSLIC (c) Original (d) MSLIC

Figure3. Show MSLIC performance in terms of content-sensitivity, segmentation error, and boundary recall.

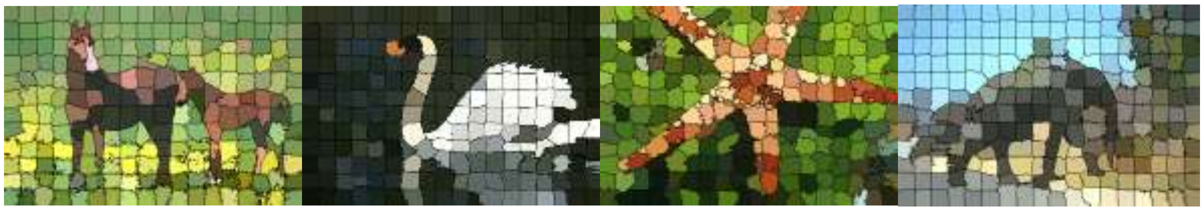


Figure4.MSLIC superpixels of BSDS500 dataset with K=200

5.3. BSDS500 dataset: SLIC: K=300, M=40, Iter=10

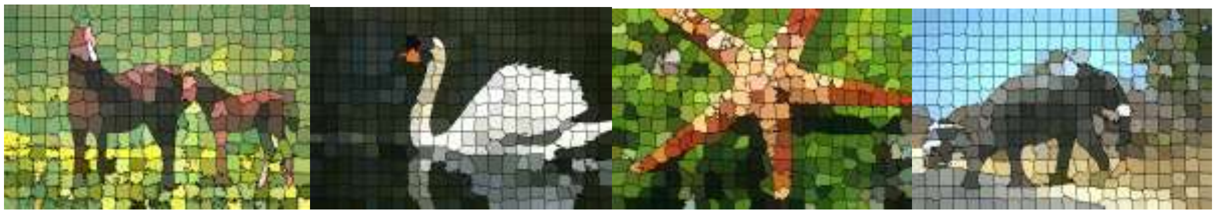


Figure5. MSLIC superpixels of BSDS500 dataset with K=300, compared to figure4, we can see that superpixels under-segmentation error and boundary recall performance improves when the number of superpixel increases at the cost of runtime performance.

6. Conclusions

Manifold SLIC is an extension of SLIC that make it possible to efficiently compute content-sensitive superpixels. This is accomplished by mapping image I to 2-manifold M embedded in combined image and color space $\mathbb{R}^5$ whose area elements are a good measure of content sensitivity. Computing SLIC method on M would induce non-uniform tessellation, thus content-sensitive when projected back to image domain. This process is called restricted Centroidal Voronoi Tesselation (RCVT). Computed area elements on manifold reflect density of image content. Efficient computation method of restricted CVT on M induces content-sensitive superpixels in I. Results of test on Lena and Berkeley benchmark show superior performance. In terms of under-segmentation error, boundary recall, content sensitivity, and runtime performance, Manifold SLIC is 10 times faster than the-state-of-the-art content-sensitive superpixel algorithm [1]. It is also based on the time it took to run tests on dataset.

7. References

[1] Y.-J. Liu, C. Yu, M. Yu, and Y. He, "Manifold SLIC: A fast method to compute content-sensitive superpixels," in Proc. IEEE Conf. Comput. Vis. Pattern Recog., 2016, pp. 651 659.https://fccdata.org/?facid=74039

[2] A. Levinshtein, A. Stere, K. N. Kutulakos, D. J. Fleet, S. J. Dickinson, and K. Siddiqi. Turbopixels: Fast superpixels using geometric flows. IEEE Trans. Pattern Analysis and Machine Intelligence, 31(12):2290 2297, 2009.

[3] P. Wang, G. Zeng, R. Gan, J. Wang, and H. Zha. Structure-sensitive superpixels via geodesic distance. International Journal of Computer Vision, 103(1):1– 21, 2013.

[4] H. Edelsbrunner and N. R. Shah. Triangulating topological spaces. International Journal of Computational Geometry & Applications, 7(4):365–378, 1997.

[5] Q. Du, V. Faber, and M. Gunzburger. Centroidal Voronoi tessellations: Applications and algorithms.SIAM Review, 41(4):637676, 1999.

[6] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. S¨usstrunk. SLIC superpixels compared to state-of-the-art superpixel methods. IEEE Trans. Pattern Analysis and Machine Intelligence, 34(11):2012–2281, 2012.

[7] Y.-J. Liu. Semi-continuity of skeletons in 2-manifold and discrete Voronoi approximation. IEEE Trans. Pattern Analysis and Machine Intelligence, 37(9):1938–1944, 2015.

[8] Y.-J. Liu, Z.-Q. Chen, and K. Tang. Construction of iso-contours, bisectors and Voronoi diagrams on triangulated surfaces. IEEE Trans. Pattern Analysis and Machine Intelligence, 33(8):1502–1517, 2011.

[9] X. Wang, X. Ying, Y.-J. Liu, S.-Q. Xin, W. Wang, X. Gu, W. Mueller-Wittig, and Y. He. Intrinsic computation of centroidal Voronoi tessellation (CVT) on meshes. Computer-Aided Design, 58:51–61, 2015.

8. Appendix

Dataset, codes and sample tests result for this project are shared on GitHub at https://github.com/sou1248/Image-Processing-Project.