Image Processing - Assignment 2

Image Segmentation

Beryl Li

1 Introduction

In assignment 2, I implemented Simple Linear Iterative Clustering (SLIC) algorithm for superpixel segmentation. For each input image, different settings are applied, such as LAB/YCbCr/HSV/RGB color spaces, region size from small to large, and smoothing or sharpening techniques in image enhancement. Furthermore, the visual result comparison with OpenCV is also provided.

2 Methods

SLIC algorithm clusters the pixels in an image based on k-means. Given an image and the expected average segment size, the first step is to initialize the cluster centers. Next, each pixel will be assigned a cluster by comparing the similarity with its nearby cluster centers. After cluster assignment is complete, for each cluster, the center will be updated by averaging the 5-dimensional feature of each pixel. The process of cluster assignment and center update will be repeated until the algorithm converges.

2.1 Initialization of cluster centers

Each cluster center is placed at the center of its grid first. However, it is possible that the initialized center is on an edge, which is not a good starting point. Therefore, a cluster center will be move to a near position which has lowest gradient. Fig 1 shows cluster centers before moving to nearby lowest gradient position, and Fig 2 shows cluster centers after moving to nearby lowest gradient position.

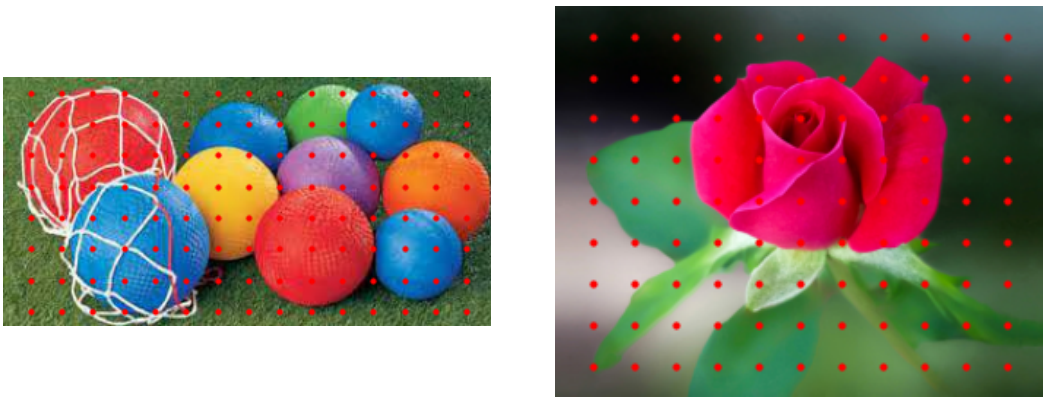


Figure 1: Initialization of cluster centers
(before moving to lowest gradient position)

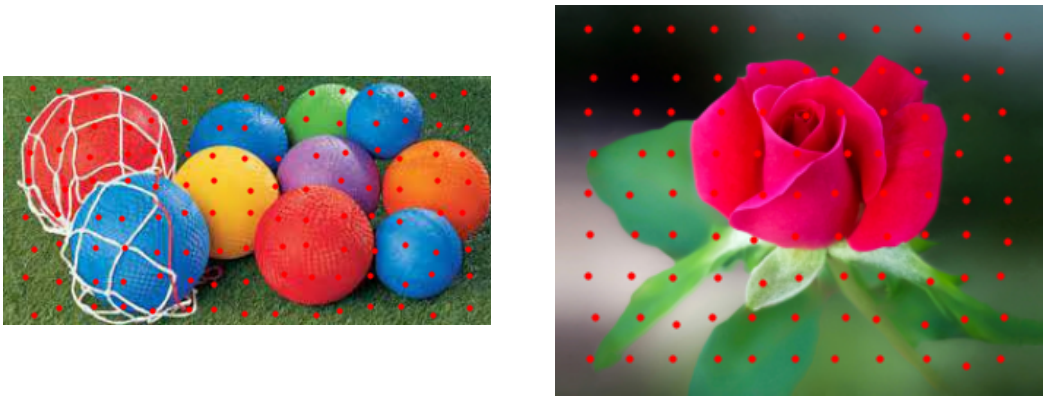


Figure 2: Initialization of cluster centers
(after moving to lowest gradient position)

2.2 Cluster assignment for each pixel

The distance between a cluster center and its neighboring pixels will be calculated to determine which cluster a pixel should belong to. There are two factors affecting the distance. One is the pixel value, which can be expressed in different color spaces, such as RGB, LAB, YCbCr, etc. The other factor is location, which is the x-y coordinate of a given pixel. The more similar the colors and the closer the pixel is to the cluster center, the shorter the distance between the pixel and the cluster center. Since there are two factors, a weight is added to set the ratio of the two factors. Fig 3 demonstrates some coloring of the cluster assignment segments.

Figure 3: Cluster assignment segments

2.3 Center update

Once all the pixels are assigned a cluster, a new center can be generated by averaging the features of all pixels in the cluster. Firstly, I calculate the center position of the target cluster to get the location feature. Next, I obtain the new color feature by averaging the pixel values in a small area centered at the new cluster center. By doing so, the cluster color feature will be more representative than referring the sole pixel of the cluster center.

2.4 Test for convergence

The steps of 'cluster assignment for each pixel' and 'center update' will be repeated many times until the new features of the clusters are almost the same as the previous iteration, which means this algorithm has converged. The contours of segments will be displayed after the algorithm is finished. Fig 4 demonstrates some segment contours of resulting images.

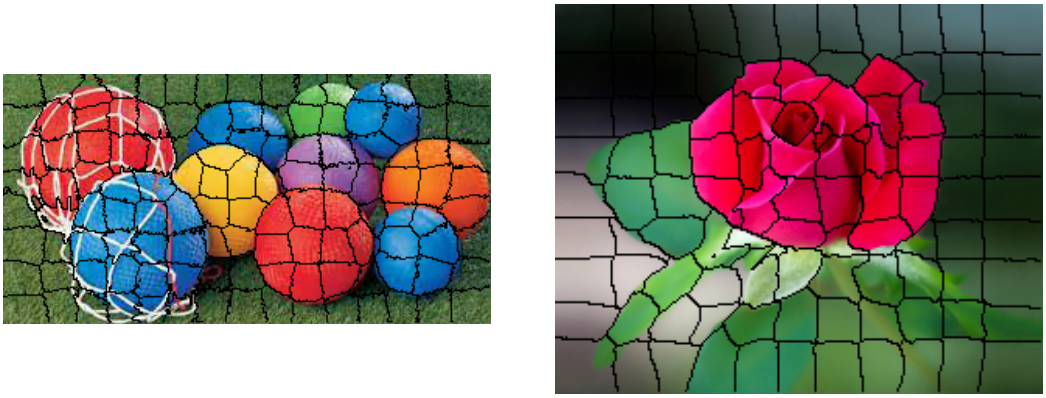


Figure 4: Segmentation contours

3 Experiments

3.1 Experiment 1: color space

In experiment 1, the image Coins (Fig 5a) is converted to LAB/YCbCr/HSV/RGB color space before running the SLIC algorithm. The region size is set to 60 pixels, and the ratio of value distance and location distance is 1:5. Before SLIC algorithm starts, the initialized cluster centers will move to position with lowest gradient in the area of 1/9 grid region. Fig 6 shows the clustering results by visualizing contours and segments, and Fig 5b shows the clustering contour produced by OpenCV.

(a) Original coins image (b) OpenCV clustering result

Figure 5: Coins

In fig 6, the result generated by the RGB color space is the best of all results because the shape of all the coin segments in the RGB color space is closest to a circle compared to other counterparts. For the segments produced in the LAB/YCbCr/HSV color space, they are not as good as the result of the RGB color space since the contour of coins is not round enough. For the result produced by OpenCV in fig 5b, the contour does not fit the outline of coins well, either.

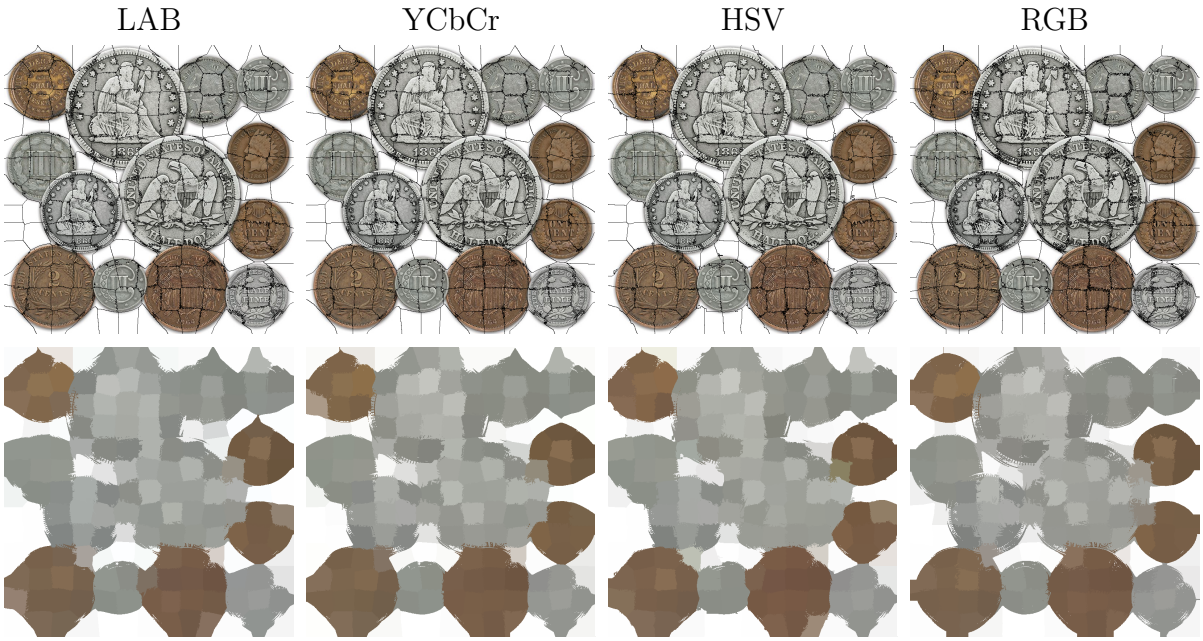


Figure 6: Results produced by LAB/YCbCr/HSV/RGB color space

3.2 Experiment 2 - region size

In experiment 2, different region size is adopted to generate different number of segments. The setting of this experiment is as follows. The ratio of value distance and location distance is 1:5. Before SLIC algorithm starts, the initialized cluster centers will move to position with lowest gradient in the area of 1/9 grid region. For the original image Leaves (Fig 7), fig 8 is the clustering result generated by OpenCV with region size = 40/30/20.

Figure 7: Original leaves image

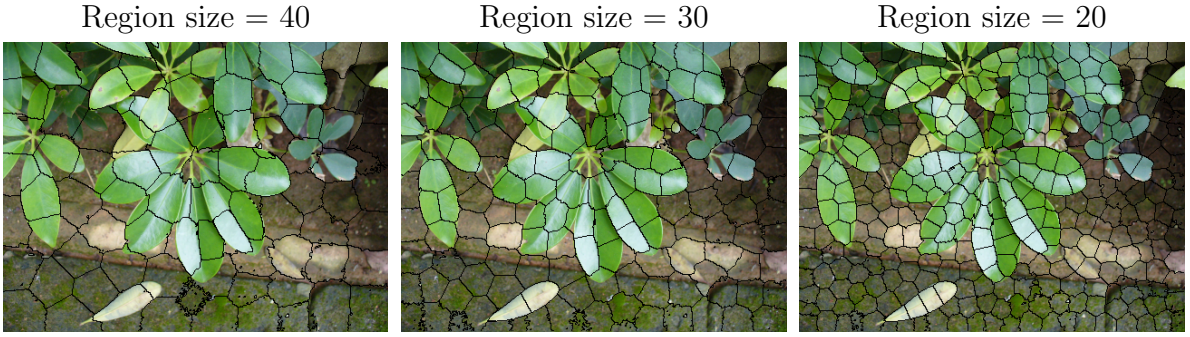


Figure 8: OpenCV clustering results

In fig 9, the smaller region size, the better the visualization results look. The small region size means more segments, which can preserve more image details. If we take a closer look at the stem, it is obvious that smaller region size provides a sharper outline of the stem while the large region size classifies the stem and background into the same cluster.

Comparing my results (fig 9) with the results of OpenCV (fig 8), the segments shape of my results is relative irregular, while the segments shape of OpenCV is more circular. The irregular shape of my results makes the produced segments more suitable for the contours of the image.

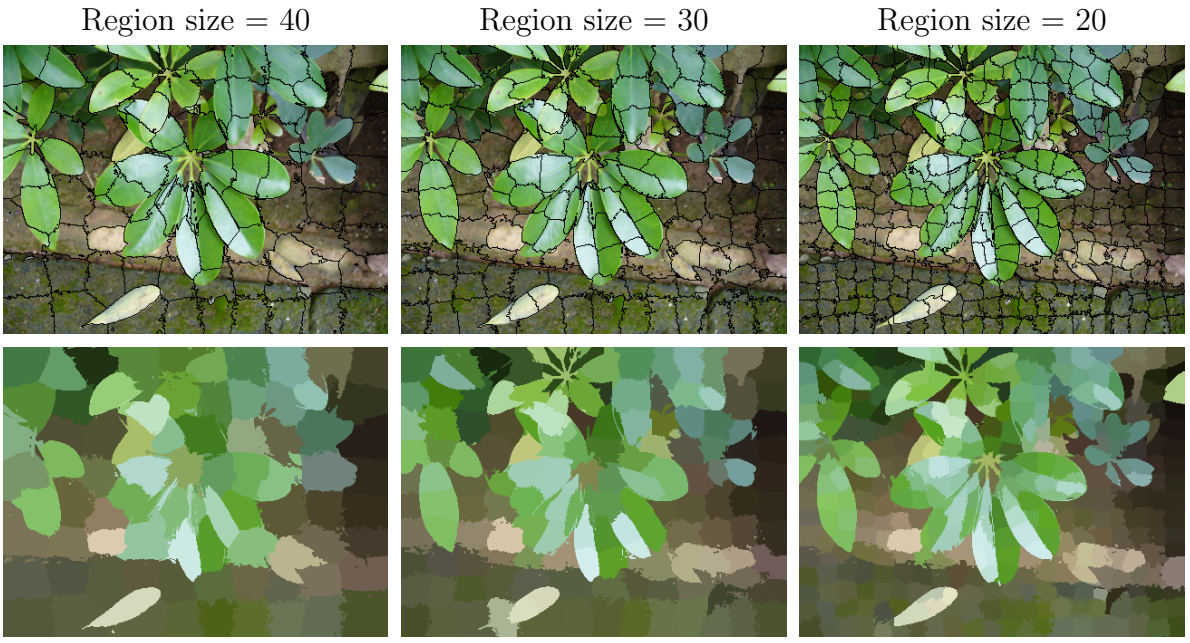
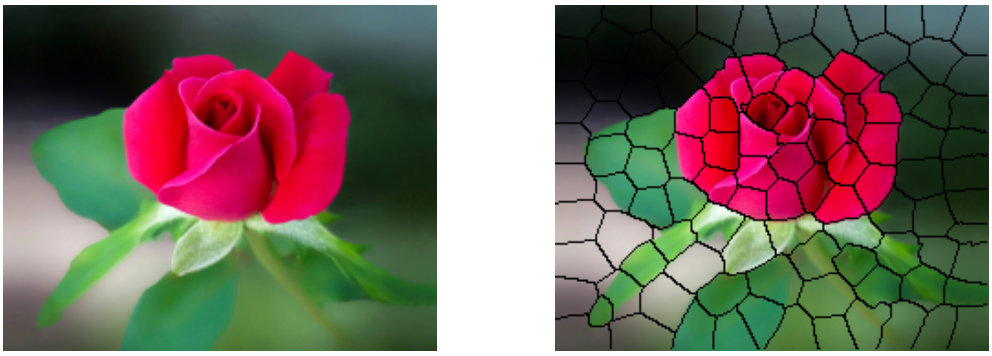


Figure 9: Results with region size = 40/30/20

3.3 Experiment 3 - weights of color and location features

In experiment 3, different weights are used to observe the effect of different ratio of color and location features on the clustering result. The region size are all the same in this experiment, and the initialized cluster centers will move to position with lowest gradient in the area of 1/9 grid region as well. Fig 10a) is the original rose image, and fig 10b) is the clustering result produced by OpenCV.



(a) Original rose image (b) OpenCV clustering result

Figure 10: Rose

The higher the weight, the more important the location features. That is why the shape of the segments is more regular when the weight is set to 10. In other words, a weight as small as 3 allows the shape of the segments to be more flexible. These phenomenon appear obviously in fig 11, which shows the irregular shape of segments retains more information of the original image.

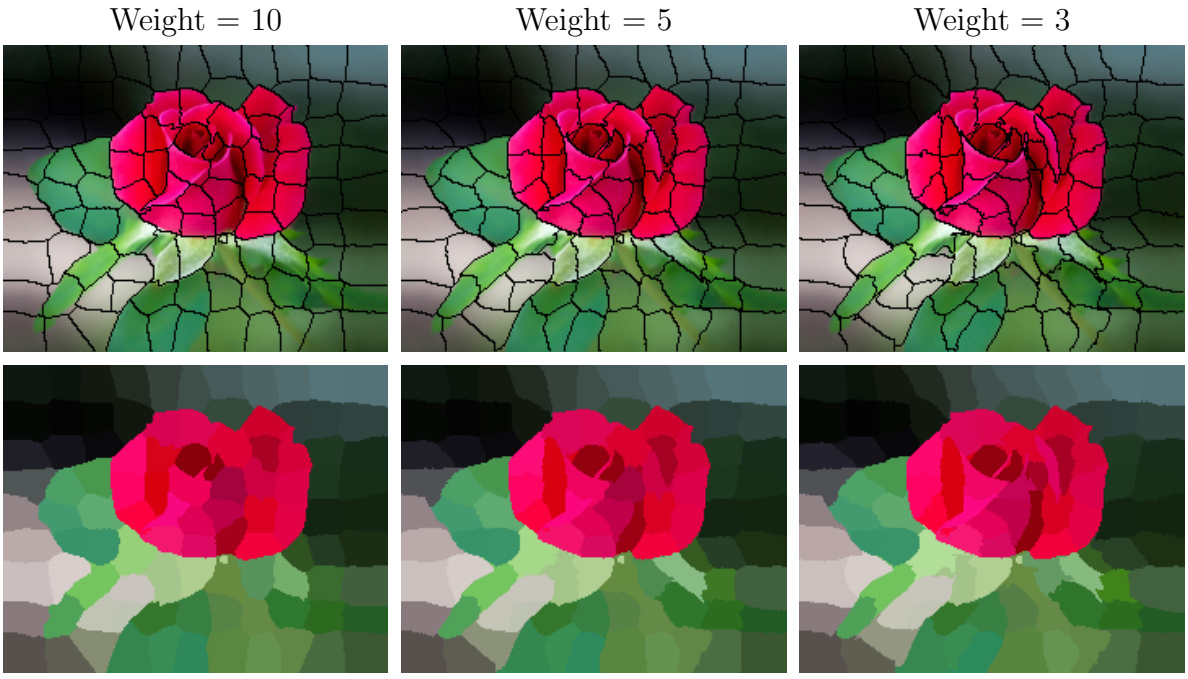


Figure 11: Results with weight = 10/5/3

3.4 Experiment 4 - initialization of cluster centers

I have mentioned that some adjustment are made to the location of the cluster centers right after initialization. In experiment 4, different adjustments will be applied so that we can see their effect on the generated clusters. The region size is 30 pixels, and the weight of color and location features is 10. Fig 12a is the original basketball image, and fig 12b is the corresponding clustering result produced by OpenCV.

(a) Original basketball image (b) OpenCV clustering result

Figure 12: Basketball

In fig 13, $\frac{1}{n} \times \frac{1}{n}$ grid means we will find the lowest gradient position within $\frac{1}{n} \times \frac{1}{n}$ size of the grid. Images from left to right in fig 13, the cluster centers are fixed in the leftmost column, while in the rightmost column, the cluster centers are relocated after a large-scale search. The reason why we need to adjust the position of cluster centers is to avoid the initial center at the edge pixel.

The comparison of initial cluster centers from different searching area and the corresponding results is shown in fig 13. If no adjustment is adopted, a initial center may be at the edge pixel, which will not produce a good enough segmentation. On the other hand, if a proper-sized region is searched to find the lowest gradient, better clusters can be created, as the rightmost column of fig 13 shows.

Figure 13: Results with different cluster center adjustment

3.5 Experiment 5 - smoothing first

In experiment 5, some image enhancement techniques used in HW1 will be applied before the SLIC algorithm. For the SLIC algorithm, the region size is 40 pixels, and the weight of color and location features is 5.

Figure 14: Pets

Figure 15: OpenCV clustering result

Fig 14 includes the original pets image, image processed by the bilateral filter (kernel = 5, $\sigma_s = 3, \sigma_v = 3$) and image processed by the median filter (kernel = 5). Fig 15 shows the corresponding clustering results generated by OpenCV. When either a bilateral or a median filter is applied, the back contour of the dog becomes better, which means reducing the furry level of the image really helps.

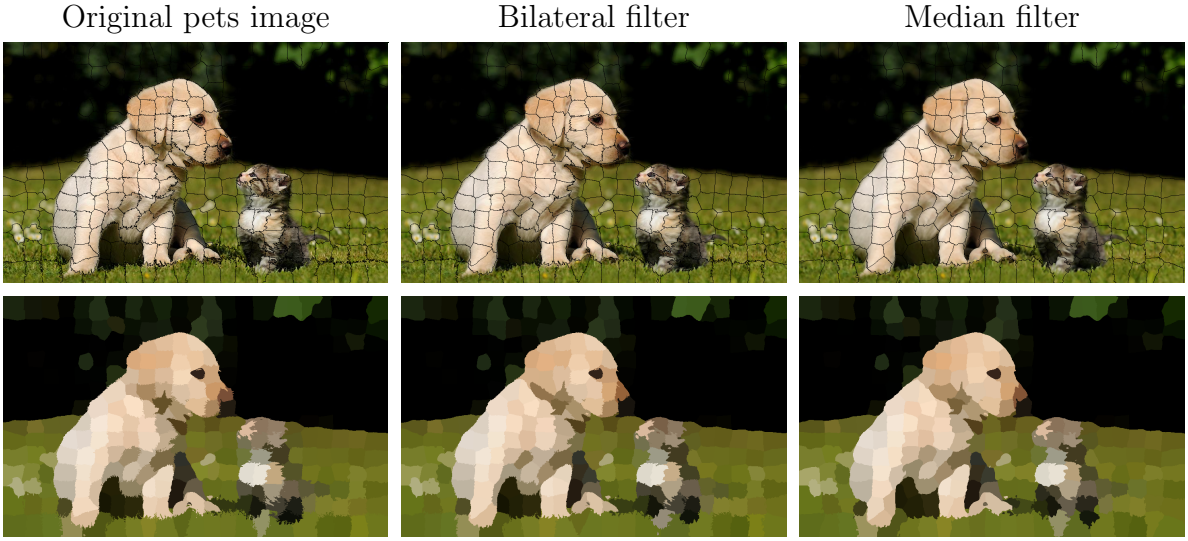


Figure 16: Results without/with bilateral/median filter

Fig 16 shows the corresponding clustering results visualized by contours and segments The furry pets in this test case make the pixel values not smooth. Without smoothing, the boundaries of segments are quite jagged, as shown in the leftmost column. As for the image preprocessed by bilateral filtering, since the edge attributes will be preserved by the bilateral filter, the segment boundaries are still a little jagged. With the help of the median filter, the rightmost column shows the smoothest segment boundaries.

4 Discussions

In this assignment, a lot of experiments have been done. Different color spaces were used to convert the image pixel values before SLIC algorithm; When performing SLIC algorithm, different region size and weights of color and location features were tried to have a look at their influence on the resulting clustering; I also tried to initialize the cluster centers by using grids of different sizes to find the lowest gradient; Some techniques for image enhancement were applied, too, so the clustering quality can be further improved by these preprocessing techniques.

With this implementation experience, I have a deeper understanding of superpixel generation and its corresponding factors. Region size and weights of color and location features essentially determine what the resulting superpixels will be looked like. The smoothing and sharpening preprocessing or cluster centers adjustment right after initialization will slightly modify the generated clusters.

Meanwhile, I studied one recent superpixel algorithm called Superpixel Sampling Networks [1]. Instead of using hand-crafted features, they used a deep convolutional network to extract the features and then feed the features into differentiable SLIC algorithm. This will be my next step to improve the current SLIC algorithm I have implemented.

References

[1] Varun Jampani, Deqing Sun, Ming Yu Liu, Ming Hsuan Yang, and Jan Kautz. Superpixel sampling networks. In *ECCV*, 2018.

5 Codes

5.1 slic.py

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
from img_io import *
from functools import partial
from tqdm import tqdm as std_tqdm
tqdm = partial(std_tqdm, dynamic_ncols=True)


def slic_cv2(img, iter=50, region_size=20, ruler=10.0):
    slic = cv2.ximgproc.createSuperpixelSLIC(img, region_size, ruler)
    slic.iterate(iter)
    label_slic = slic.getLabels()

    mask_slic = np.zeros(img.shape[:2], dtype='uint8')
    for h in range(img.shape[0]-1):
        for w in range(img.shape[1]-1):
            if label_slic[h+1][w] != label_slic[h][w] or label_slic[h][w+1] !=
                label_slic[h][w]:
                mask_slic[h][w] = 255
            else:
                mask_slic[h][w] = 0

    number_slic = slic.getNumberOfSuperpixels()
    print(f'Total number of superpixels: {number_slic:<5d}')
    mask_inv_slic = cv2.bitwise_not(mask_slic)
    img_slic = cv2.bitwise_and(img, img, mask=mask_inv_slic)

    return img_slic


class SLIC():
    def __init__(self, img, region_size, max_iter, const, threshold):
        self.img = img
        self.height, self.width = img.shape[:2]
        self.region_size = region_size
        self.max_iter = max_iter
        self.const = const
        self.threshold = threshold

    def init(self):
        ''' initialize superpixel cluster centers
        centers (ndarray): (l, a, b, x, y)
        labels (ndarray): (h, w) full of -1
        distances (ndarray): (h, w) full of inf
        '''
```

```
46          self.centers = []
47          h_init = round((self.height % self.region_size) / 2 + self.region_size
                / 2)
48          w_init = round((self.width % self.region_size) / 2 + self.region_size
                / 2)
49          for h in range(h_init, self.height, self.region_size):
50              for w in range(w_init, self.width, self.region_size):
51                  self.centers.append((*self.img[h, w], h, w))
52          self.centers = np.array(self.centers)
53
54          self.labels = np.full((self.height, self.width), -1)
55          self.distances = np.full((self.height, self.width), np.inf)
56
57          print(f'Total number of superpixels: {len(self.centers):<5d}')
58
59      def _get_grad(self, patch):
60          grad_1 = abs(np.sum([patch[:, :, i] * np.array([[-1, -2, -1], [0, 0,
                0], [1, 2, 1]]) for i in range(3)]))
61          grad_2 = abs(np.sum([patch[:, :, i] * np.array([[-1, 0, 1], [-2, 0,
                2], [-1, 0, 1]]) for i in range(3)]))
62          return grad_1 + grad_2
63
64      def adjust_centers(self, ratio=3):
65          patch_size = self.region_size // 2 // ratio
66          for c in range(len(self.centers)):
67              h, w = self.centers[c][3], self.centers[c][4]
68              grad = self._get_grad(self.img[h-1:h+2, w-1:w+2, :])
69              for dh in range(-patch_size, patch_size + 1):
70                  for dw in range(-patch_size, patch_size + 1):
71                      new_grad = self._get_grad(self.img[h+dh-1:h+dh+2,
                            w+dw-1:w+dw+2, :])
72                      if new_grad < grad:
73                          self.centers[c] = np.array((*self.img[h+dh, w+dw, :],
                                h+dh, w+dw))
74                          grad = new_grad
75
76      def show_centers(self, output_path=''):
77          plt.scatter(x=self.centers[:, 4], y=self.centers[:, 3], c='r', s=10)
78          img = cv2.cvtColor(self.img, cv2.COLOR_BGR2RGB)
79          plt.imshow(img)
80          if output_path != '':
81              plt.savefig(output_path, bbox_inches='tight', pad_inches=0)
82          plt.show()
83          plt.close()
84
85      def _get_dist(self, p, c):
86          d_color = np.sqrt(np.sum((p[:3] - c[:3])**2))
87          d_spatial = np.sqrt(np.sum((p[3:] - c[3:])**2))
88          return  (d_color + self.const * d_spatial)
89
90      def assign_labels(self):
```

```
91          for c in range(len(self.centers)):
92              for h in range(self.centers[c][3] - self.region_size,
                    self.centers[c][3] + self.region_size):
93                  for w in range(self.centers[c][4] - self.region_size,
                        self.centers[c][4] + self.region_size):
94                      if h < 0 or h >= self.height or w < 0 or w >= self.width:
95                          continue
96                      d = self._get_dist(np.array((*self.img[h, w, :], h, w)),
                            self.centers[c])
97                      if d < self.distances[h][w]:
98                          self.distances[h][w] = d
99                          self.labels[h][w] = c
100
101     def update_centers_v0(self):
102         denoms = np.zeros(self.centers.shape[0])
103         num = np.zeros(self.centers.shape)
104         for h in range(self.height):
105             for w in range(self.width):
106                 denoms[self.labels[h][w]] += 1
107                 num[self.labels[h][w]] += np.array((*self.img[h, w, :], h, w))
108
109         new_centers = np.zeros(self.centers.shape, dtype='int')
110         diff = 0
111         for c in range(len(new_centers)):
112             new_centers[c] = num[c] / denoms[c]
113             diff += np.sum(abs(self.centers[c] - new_centers[c]))
114
115         return diff
116
117     def update_centers(self, n=2):
118         denoms_pos = np.zeros(self.centers.shape[0])
119         num_pos = np.zeros(self.centers[:, 3:].shape)
120         for h in range(self.height):
121             for w in range(self.width):
122                 denoms_pos[self.labels[h][w]] += 1
123                 num_pos[self.labels[h][w]] += np.array((h, w))
124
125         new_centers = np.zeros(self.centers.shape, dtype='int')
126         diff = 0
127         for c in range(len(new_centers)):
128             if denoms_pos[c] != 0:
129                 new_centers[c, 3:] = num_pos[c] / denoms_pos[c]
130                 diff += np.sum(abs(self.centers[c, 3:] - new_centers[c, 3:]))
131
132         denoms_color = np.zeros(self.centers.shape[0])
133         num_color = np.zeros(self.centers[:, :3].shape)
134         patch_size = self.region_size // n // 2
135         for c in range(len(new_centers)):
136             for h in range(new_centers[c, 3] - patch_size, new_centers[c, 3] +
                    patch_size):
```

```
137                 for w in range(new_centers[c, 4] - patch_size, new_centers[c,
                        4] + patch_size):
138                     if self.labels[h][w] == c:
139                         denoms_color[c] += 1
140                         num_color[c] += np.array(self.img[h, w, :])
141             if denoms_color[c] != 0:
142                 new_centers[c, :3] = num_color[c] / denoms_color[c]
143 
144         self.centers = new_centers
145         return diff
146 
147     def display(self, output_path='', tag=-1):
148         img = np.zeros(self.img.shape, dtype='uint8')
149         for h in range(self.height):
150             for w in range(self.width):
151                 img[h][w] = np.array(self.centers[self.labels[h][w]][:3],
                        dtype='uint8')
152         if tag != -1:
153             img = cv2.cvtColor(img, tag)
154         display_img(img, False)
155         if output_path != '':
156             save_img(img, output_path)
157 
158     def display_contour(self, output_path=''):
159         mask = np.zeros(self.img.shape[:2], dtype='uint8')
160         for h in range(self.height-1):
161             for w in range(self.width-1):
162                 if self.labels[h+1][w] != self.labels[h][w] or
                        self.labels[h][w+1] != self.labels[h][w]:
163                     mask[h][w] = 255
164                 else:
165                     mask[h][w] = 0
166         mask = cv2.bitwise_not(mask)
167         img_slic = cv2.bitwise_and(self.img, self.img, mask=mask)
168         display_img(img_slic, False)
169         if output_path != '':
170             save_img(img_slic, output_path)
171 
172     def process(self):
173         with tqdm(total=self.max_iter) as pbar:
174             for i in range(self.max_iter):
175                 self.assign_labels()
176                 diff = self.update_centers()
177                 if diff < self.threshold:
178                     break
179                 pbar.set_postfix({'diff': diff})
180                 pbar.update(1)
```

Code 1: Simple Linear Iterative Clustering

5.2 Experiment 1: Coins - color space

```python
parser = argparse.ArgumentParser()
parser.add_argument('--img_file', type=str, default='pic4PR2/coins.jpg')
parser.add_argument('--is_gray', type=bool, default=False)
parser.add_argument('--output_file', type=str, default='output_coins/')
args, unknown = parser.parse_known_args()

def experiment_cv2(args, output_path, region_size):
    img = read_img(args.img_file, args.is_gray)
    img_slic = slic_cv2(img, iter=50, region_size=region_size)
    display_img(img_slic, False)
    if output_path != '':
        save_img(img_slic, output_path + f'_{region_size}')
experiment_cv2(args, args.output_file+'cv2', 60)

def experiment_color_space(args):
    img = read_img(args.img_file, args.is_gray)

    ''' HSV '''
    slic = SLIC(img, region_size=60, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.img = cv2.cvtColor(slic.img, cv2.COLOR_BGR2HSV)
    slic.adjust_centers()
    slic.process()
    slic.img = cv2.cvtColor(slic.img, cv2.COLOR_HSV2BGR)
    slic.display(args.output_file+'segmentation_HSV', cv2.COLOR_HSV2BGR)
    slic.display_contour(args.output_file+'contour_HSV')

    ''' LAB '''
    slic = SLIC(img, region_size=60, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.img = cv2.cvtColor(slic.img, cv2.COLOR_BGR2LAB)
    slic.adjust_centers()
    slic.process()
    slic.img = cv2.cvtColor(slic.img, cv2.COLOR_LAB2BGR)
    slic.display(args.output_file+'segmentation_LAB', cv2.COLOR_LAB2BGR)
    slic.display_contour(args.output_file+'contour_LAB')

    ''' YCbCr '''
    slic = SLIC(img, region_size=60, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.img = cv2.cvtColor(slic.img, cv2.COLOR_BGR2YCR_CB)
    slic.adjust_centers()
    slic.process()
    slic.img = cv2.cvtColor(slic.img, cv2.COLOR_YCR_CB2BGR)
    slic.display(args.output_file+'segmentation_YCbCr', cv2.COLOR_YCR_CB2BGR)
    slic.display_contour(args.output_file+'contour_YCbCr')

    ''' RGB '''
    slic = SLIC(img, region_size=60, max_iter=10, const=5, threshold=1)
```

```
    slic.init()
    slic.adjust_centers()
    slic.process()
    slic.display(args.output_file+'segmentation_RGB')
    slic.display_contour(args.output_file+'contour_RGB')

experiment_color_space(args)
```

Code 2: Experiment 1: Coins - color space

5.3 Experiment 2: Leaves - region size

```
parser = argparse.ArgumentParser()
parser.add_argument('--img_file', type=str, default='pic4PR2/leaves.jpg')
parser.add_argument('--is_gray', type=bool, default=False)
parser.add_argument('--output_file', type=str, default='output_leaves/')
args, unknown = parser.parse_known_args()

def experiment_cv2(args, output_path, region_size):
    img = read_img(args.img_file, args.is_gray)
    img = cv2.resize(img, (img.shape[1] // 3, img.shape[0] // 3),
        interpolation = cv2.INTER_AREA)
    img_slic = slic_cv2(img, iter=250, region_size=region_size)
    display_img(img_slic, False)
    if output_path != '':
        save_img(img_slic, output_path + f'_{region_size}')

experiment_cv2(args, args.output_file+'cv2', 20)
experiment_cv2(args, args.output_file+'cv2', 30)
experiment_cv2(args, args.output_file+'cv2', 40)

def experiment_region_size(args):
    img = read_img(args.img_file, args.is_gray)
    img = cv2.resize(img, (img.shape[1] // 3, img.shape[0] // 3),
        interpolation = cv2.INTER_AREA)

    ''' region_size=20 '''
    slic = SLIC(img, region_size=20, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.process()
    slic.display(args.output_file+'segment_region_size_20')
    slic.display_contour(args.output_file+'contour_region_size_20')

    ''' region_size=30 '''
    slic = SLIC(img, region_size=30, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.process()
```

```
    slic.display(args.output_file+'segment_region_size_30')
    slic.display_contour(args.output_file+'contour_region_size_30')

    ''' region_size=40 '''
    slic = SLIC(img, region_size=40, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.process()
    slic.display(args.output_file+'segment_region_size_40')
    slic.display_contour(args.output_file+'contour_region_size_40')

experiment_region_size(args)
```

Code 3: Experiment 2: Leaves - region size

5.4 Experiment 3: Rose - weights of color and location features

```
parser = argparse.ArgumentParser()

parser.add_argument('--img_file', type=str, default='pic4PR2/rose.jpg')
parser.add_argument('--is_gray', type=bool, default=False)
parser.add_argument('--output_file', type=str, default='output_rose/')

args, unknown = parser.parse_known_args()

def experiment_cv2(args, output_path, region_size):
    img = read_img(args.img_file, args.is_gray)
    img = cv2.resize(img, (img.shape[1] // 10, img.shape[0] // 10),
        interpolation = cv2.INTER_AREA)
    img_slic = slic_cv2(img, iter=250, region_size=region_size)
    display_img(img_slic, False)
    if output_path != '':
        save_img(img_slic, output_path + f'_{region_size}')
experiment_cv2(args, args.output_file+'cv2', 20)

def experiment_weight(args):
    img = read_img(args.img_file, args.is_gray)
    img = cv2.resize(img, (img.shape[1] // 10, img.shape[0] // 10),
        interpolation = cv2.INTER_AREA)

    ''' const=3 '''
    slic = SLIC(img, region_size=20, max_iter=15, const=3, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.process()
    slic.display(args.output_file+'segment_const_3')
    slic.display_contour(args.output_file+'contour_const_3')

    ''' const=5 '''
```

```
    slic = SLIC(img, region_size=20, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.process()
    slic.display(args.output_file+'segment_const_5')
    slic.display_contour(args.output_file+'contour_const_5')

    ''' const=10 '''
    slic = SLIC(img, region_size=20, max_iter=15, const=10, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.process()
    slic.display(args.output_file+'segment_const_10')
    slic.display_contour(args.output_file+'contour_const_10')

experiment_weight(args)
```

Code 4: Experiment 3: Rose - weights of color and location features

5.5 Experiment 4: Basketball - initialization of cluster centers

```
parser = argparse.ArgumentParser()

parser.add_argument('--img_file', type=str, default='pic4PR2/basketball.jpg')
parser.add_argument('--is_gray', type=bool, default=False)
parser.add_argument('--output_file', type=str, default='output_basketball/')

args, unknown = parser.parse_known_args()

def get_init_centers():
    img = read_img(args.img_file, args.is_gray)
    slic = SLIC(img, region_size=60, max_iter=100, const=20, threshold=1)
    slic.init()
    slic.show_centers(args.output_file+'centers_axisoff')

    slic.init()
    slic.adjust_centers(ratio=4)
    slic.show_centers(args.output_file+'centers_adjust_4_axisoff')

    slic.init()
    slic.adjust_centers(ratio=3)
    slic.show_centers(args.output_file+'centers_adjust_3_axisoff')
get_init_centers()

def experiment_cv2(args, output_path, region_size, ruler):
    img = read_img(args.img_file, args.is_gray)
    img_slic = slic_cv2(img, iter=250, region_size=region_size, ruler=ruler)
    display_img(img_slic, False)
    if output_path != '':
```

```
        save_img(img_slic, output_path + f'_{region_size}')
experiment_cv2(args, args.output_file+'cv2', 30, 10)

def experiment_move_center_or_not(args):
    img = read_img(args.img_file, args.is_gray)

    ''' Not adjust_centers '''
    print('Not adjust_centers...')
    slic = SLIC(img, region_size=30, max_iter=15, const=10, threshold=1)
    slic.init()
    slic.process()
    slic.display(args.output_file+'segment_not_adjust_centers')
    slic.display_contour(args.output_file+'contour_not_adjust_centers')

    ''' adjust_centers 4 '''
    slic = SLIC(img, region_size=30, max_iter=15, const=10, threshold=1)
    slic.init()
    slic.adjust_centers(ratio=4)
    slic.process()
    slic.display(args.output_file+'segment_adjust_centers_4')
    slic.display_contour(args.output_file+'contour_adjust_centers_4')

    ''' adjust_centers 3 '''
    slic = SLIC(img, region_size=30, max_iter=15, const=10, threshold=1)
    slic.init()
    slic.adjust_centers(ratio=3)
    slic.process()
    slic.display(args.output_file+'segment_adjust_centers_3')
    slic.display_contour(args.output_file+'contour_adjust_centers_3')
experiment_move_center_or_not(args)
```

Code 5: Experiment 4: Basketball - initialization of cluster centers

5.6 Experiment 5: Pets - smoothing first

```
parser = argparse.ArgumentParser()

parser.add_argument('--img_file', type=str, default='pic4PR2/pets.jpg')
parser.add_argument('--is_gray', type=bool, default=False)
parser.add_argument('--output_file', type=str, default='output_pets/')

args, unknown = parser.parse_known_args()

def experiment_none(args):
    img = read_img(args.img_file, args.is_gray)
    save_img(img, args.output_file + f'original_none')

    img_slic = slic_cv2(img, iter=250, region_size=40, ruler=10)
    display_img(img_slic, False)
```

```
    save_img(img_slic, args.output_file + f'cv2_none')

    slic = SLIC(img, region_size=40, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.show_centers(args.output_file+'centers_none')
    slic.process()
    slic.display(args.output_file+'segment_none')
    slic.display_contour(args.output_file+'contour_none')
experiment_none(args)

def experiment_median(args):
    img = read_img(args.img_file, args.is_gray)
    display_img(img, args.is_gray)

    img_med = np.empty(img.shape)
    img_med[:, :, 0] = median_filter(img[:, :, 0], 5)
    img_med[:, :, 1] = median_filter(img[:, :, 1], 5)
    img_med[:, :, 2] = median_filter(img[:, :, 2], 5)
    img_med = img_med.astype('uint8')
    img = img_med
    display_img(img, args.is_gray)
    save_img(img, args.output_file + f'original_median')

    img_slic = slic_cv2(img, iter=250, region_size=40, ruler=10)
    display_img(img_slic, False)
    save_img(img_slic, args.output_file + f'cv2_median')

    slic = SLIC(img, region_size=40, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.show_centers(args.output_file+'centers_median')
    slic.process()
    slic.display(args.output_file+'segment_median')
    slic.display_contour(args.output_file+'contour_median')
experiment_median(args)

def experiment_bilateral(args):
    img = read_img(args.img_file, args.is_gray)
    display_img(img, args.is_gray)

    img_bi = np.empty(img.shape, dtype='uint8')
    img_bi[:, :, 0] = bilateral_filter(img[:, :, 0], 5, 3, 3)
    img_bi[:, :, 1] = bilateral_filter(img[:, :, 1], 5, 3, 3)
    img_bi[:, :, 2] = bilateral_filter(img[:, :, 2], 5, 3, 3)
    img_bi = img_bi.astype('uint8')
    img = img_bi
    display_img(img, args.is_gray)
    save_img(img, args.output_file + f'original_bilateral')

    img_slic = slic_cv2(img, iter=250, region_size=40, ruler=10)
```

```
    display_img(img_slic, False)
    save_img(img_slic, args.output_file + f'cv2_bilateral')

    slic = SLIC(img, region_size=40, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.show_centers(args.output_file+'centers_bilateral')
    slic.process()
    slic.display(args.output_file+'segment_bilateral')
    slic.display_contour(args.output_file+'contour_bilateral')
experiment_bilateral(args)
```

Code 6: Experiment 5: Pets - smoothing first

5.7 Experiment 6: Scene - sharpening first

```
parser = argparse.ArgumentParser()

parser.add_argument('--img_file', type=str, default='pic4PR2/scene.jpg')
parser.add_argument('--is_gray', type=bool, default=False)
parser.add_argument('--output_file', type=str, default='output_scene/')

args, unknown = parser.parse_known_args()

def experiment_none(args):
    img = read_img(args.img_file, args.is_gray)
    save_img(img, args.output_file + f'original_none')

    img_slic = slic_cv2(img, iter=250, region_size=30, ruler=10)
    display_img(img_slic, False)
    save_img(img_slic, args.output_file + f'cv2_none')

    slic = SLIC(img, region_size=30, max_iter=15, const=5, threshold=1)
    slic.init()
    slic.adjust_centers()
    slic.show_centers(args.output_file+'centers_none')
    slic.process()
    slic.display(args.output_file+'segment_none')
    slic.display_contour(args.output_file+'contour_none')
experiment_none(args)

def experiment_laplacian(args):
    img = read_img(args.img_file, args.is_gray)
    display_img(img, args.is_gray)

    ''' smoothing filter: gaussian '''
    img_gaussian = np.empty(img.shape)
    kernel = get_kernel1d(kernel_size=3, type='gaussian')
    for i in range(3):
```

```
34          img_gaussian[:, :, i] = conv_separable(img[:, :, i], kernel)
35      img_gaussian = img_gaussian.astype('uint8')
36      display_img(img_gaussian, args.is_gray)
37      save_img(img_gaussian, args.output_file + f'original_gaussian')
38
39      ''' sharpening filter: laplacian '''
40      img_sharp = np.zeros(img.shape, dtype='uint8')
41      img_new = np.zeros(img.shape)
42      for i in range(3):
43          kernel = get_kernel2d(type='laplacian_2')
44          img_sharp[:, :, i] = conv_nonseparable(img_gaussian[:, :, i],
                kernel).astype('int64')
45          img_new[:, :, i] = img_gaussian[:, :, i] + 0.5 * img_sharp[:, :, i]
46          img_new[:, :, i][img_new[:, :, i] > 255] = 255; img_new[:, :,
                i][img_new[:, :, i] < 0] = 0
47      img_new = img_new.astype('uint8')
48      display_img(img_sharp, False)
49      save_img(img_sharp, args.output_file + f'original_laplacian')
50      display_img(img_new, False)
51      save_img(img_new, args.output_file + f'original_laplacian_enhanced')
52      img = img_new
53
54      img_slic = slic_cv2(img, iter=250, region_size=30, ruler=10)
55      display_img(img_slic, False)
56      save_img(img_slic, args.output_file + f'cv2_laplacian')
57
58      slic = SLIC(img, region_size=30, max_iter=15, const=5, threshold=1)
59      slic.init()
60      slic.adjust_centers()
61      slic.show_centers(args.output_file+'centers_laplacian')
62      slic.process()
63      slic.display(args.output_file+'segment_laplacian')
64      slic.display_contour(args.output_file+'contour_laplacian')
65  experiment_laplacian(args)
```

Code 7: Experiment 6: Scene - sharpening first