Mean Shift Method for Image Segmentation

盛凯枫*

2018 年 5 月 14 日

Abstract

As one of the most powerful clustering technique, mean shift has been used in various classification tasks including image segmentation, clustering, visual tracking, space analysis, mode seeking, etc. In the current solving of image segmentation problem, a Python program written from scratch utilizing the mean shift method is used to segment two randomly chosen pictures which are down sampled to the size of 200*150 pixels. Experiments adopting different parameters are carried out to test their influences on the segmentation process and the results are being compared.

Method

Kernel Function; Shifting Step

Choosing a proper kernel function is one of the most important step for the implementation of the mean shift method. Two kernels, the flat kernel and the Gaussian kernel, have been tried in the experiments with a same set of parameters, respectively. Specifically speaking, for each step of the shifting process, a weight matrix with the same size as the input picture is computed. In the case of flat kernel:

$$W_i = \begin{cases} 0 & \text{if } \| p_i - p \| > r^2 \\ 1 & \text{if } \| p_i - p \| \leq r^2 \end{cases} \tag{1}$$

$$\| p_i - p \| = \sum_k (p_{ik} - p_k)^2 \tag{2}$$

While in the case of Gaussian kernel:

$$W_i = e^{-\frac{\|p_i - p\|}{r^2}} \tag{3}$$

With the weight matrix shown above, p is shift to a new point $p' = \frac{\sum_i W_i p_i}{\sum_i W_i}$ in one shifting step. Details are shown in the code *shifting step*.

Spacial information is utilized in the mean shift procedure by concatenating the normalized coordinate(pixel coordinate rescaled to the range of 0-255 and multiplied by a coefficient α) at the end of the three elements list of the RGB information for each pixel, yielding a new picture with each pixel being a five elements list. Details are shown in the code *binding coordinate*.

Image Segmentation

The shifting process for one pixel composes of a row of shifting steps, stopping when $\| p' - p \| < 0.1^2$ or when the max step number is arrived. Max step number is fixed at 100 for the whole report. And regarding to the entire picture, the same shifting process is repeated for each pixel, generating a new picture with each pixel being the final point of the mean shift process. Details are shown in the code *image segmentation*.

To make better use of the Intel Core i7-7700K CPU on which the program runs and shorten the time

*Peking University, School of Physics. Email:1500011404@pku.edu.cn

of the segmentation procedure, a multi-threading feature is adopted, computing each pixel column of the picture parallelly in a separate thread.

Result

There are three parameters, the kernel type, the kernel radius and the alpha coefficient, to be determined in the mean shift method. Experiments employing sets of different parameters are carried out to illustrate their influence on the outcome of the mean shift procedure. The kernel type is either Gaussian kernel or flat kernel, the kernel radius takes a value in the set of 5, 10, 20, 30, 50, 100 and alpha in the set of 0, 0.1, 0.3, 0.5, 1.0. Image Segmentation results are compared in 2,3,5,6. The code is shown in *parameter experiment*.

References

[1] Dorin Comaniciu, Peter Meer, "Mean Shift: A Robust Approach Toward Feature Space Analysis" IEEE Transactions on Pattern Analysis and Machine Intelligence, May 2002.

---

shifting step

```
def shift(point, pic_array, kernel_type):
    global radius
    if kernel_type:
        w = np.exp((-np.sum((pic_array-point)**2, 2)/radius**2))
    else:
        w = (np.sum((pic_array-point)**2, 2) < radius**2)
        w = w.astype(np.int32)
    w = w.reshape((w.shape[0], w.shape[1], 1))/np.sum(w)
    s = np.sum(np.sum(w*pic_array, 0), 0)
    return s
```

binding coordinate

```
def binding_coordinate(pic):
    pic_data = pic.load()
    pic_array = []
    for x in range(pic.size[0]):
        line = []
        for y in range(pic.size[1]):
            line.append(
                np.array(list(pic_data[x, y]) + [x / pic.size[0] * 256 * alpha] + [y
                    / pic.size[1] * 256 * alpha]))
        pic_array.append(line)
    return np.array(pic_array)
```

image segmentation

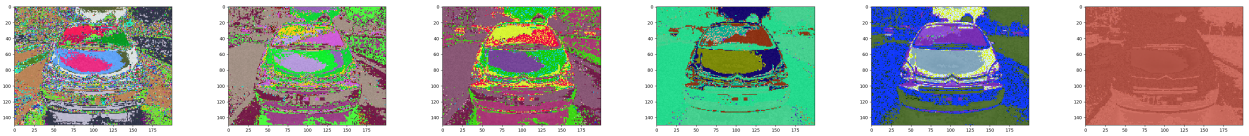
```
def segment(pic_array):
    peak_set = [np.zeros(5)]
    peak_distribution = np.zeros(pic.size) - 1
    stime = time.time()
    thread_array = []
    lock = threading.Lock()

    for x in range(0, pic.size[0]):
        def compute_column(x):
            for y in range(0, pic.size[1]):
                p = pic_array[x, y]
                for step in range(max_step):
                    next_p = shift(p, pic_array, kernel_type)
                    if np.sum((p-next_p)**2) < 0.01:
                        break
                    p = next_p
                lock.acquire()
                peak_set.append(p)
                c = len(peak_set) - 1
                global counter
                counter += 1
                if counter % pic.size[1] == 0:
                    print(counter // pic.size[1])
                    print(time.time() - stime)
                lock.release()
                peak_distribution[x, y] = c

        thread_array.append(threading.Thread(target=compute_column, args=(x,)))
        thread_array[-1].start()
    for t in thread_array:
        t.join()
    np.save("set", peak_set)
    np.save("dis", peak_distribution)
    peak_dis = peak_distribution.T
    new_dis = np.zeros((peak_dis.shape[0], peak_dis.shape[1], 3))
    for x in range(peak_dis.shape[0]):
        for y in range(peak_dis.shape[1]):
            new_dis[x, y, 2] = peak_set[int(peak_dis[x, y])][2]
            new_dis[x, y, :2] = peak_set[int(peak_dis[x, y])][:2]
    plt.imshow(256-new_dis)
    plt.savefig("results/2_k\%s_r\%s_alpha\%s.png"\%(kernel_type, radius, alpha))
```

parameter experiment

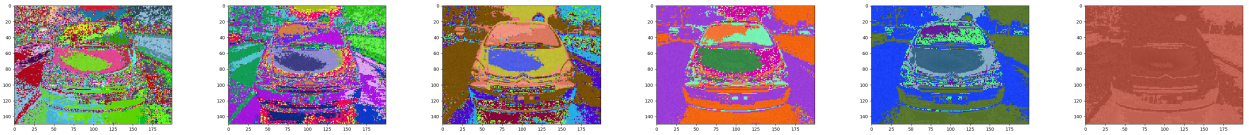
```
pic = Image.open("pic2.jpg").resize((200, 150), Image.ANTIALIAS)
```

```
max_step = 100
for r in [5, 10, 20, 30, 50, 100]:
    for a in [0.0, 0.1, 0.3, 0.5, 1.0]:
        for k in [1, 0]:
            counter = 0
            radius = r
            alpha = a
            kernel_type = k
            segment(pic)
```

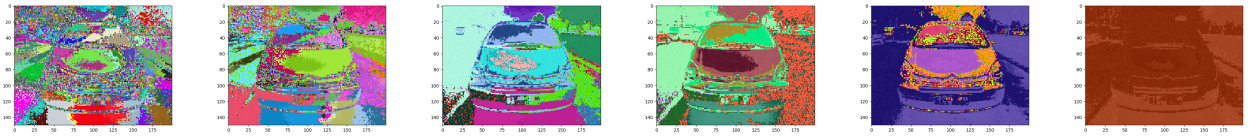
Figure 1: The origin figure of figure1.



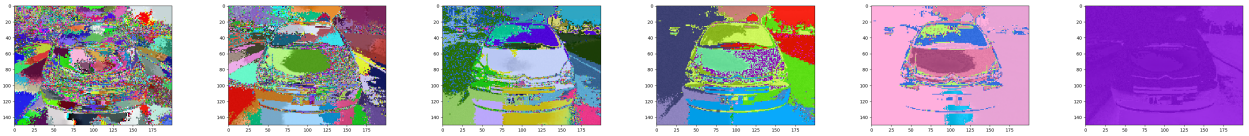
2.1:r=5,α=0.0 2.2:r=10,α=0.0 2.3:r=20,α=0.0 2.4:r=30,α=0.0 2.5:r=50,α=0.0 2.6:r=100,α=0.0



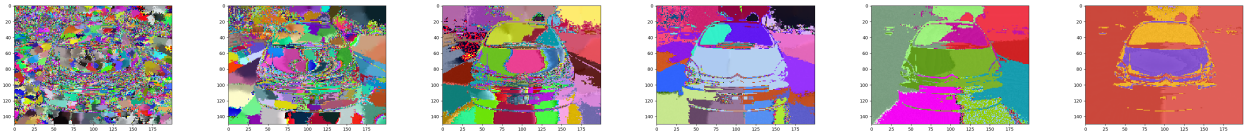
2.7:r=5,α=0.1 2.8:r=10,α=0.1 2.9:r=20,α=0.1 2.10:r=30,α=0.1 2.11:r=50,α=0.1 2.12:r=100,α=0.1



2.13:r=5,α=0.3 2.14:r=10,α=0.3 2.15:r=20,α=0.3 2.16:r=30,α=0.3 2.17:r=50,α=0.3 2.18:r=100,α=0.3

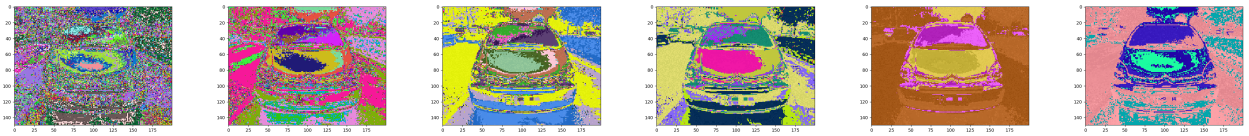


2.19:r=5,α=0.5 2.20:r=10,α=0.5 2.21:r=20,α=0.5 2.22:r=30,α=0.5 2.23:r=50,α=0.5 2.24:r=100,α=0.5

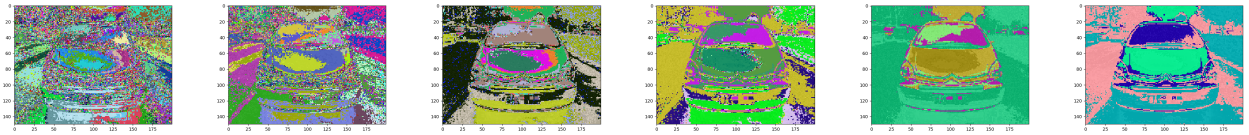


2.25:r=5,α=1.0 2.26:r=10,α=1.0 2.27:r=20,α=1.0 2.28:r=30,α=1.0 2.29:r=50,α=1.0 2.30:r=100,α=1.0

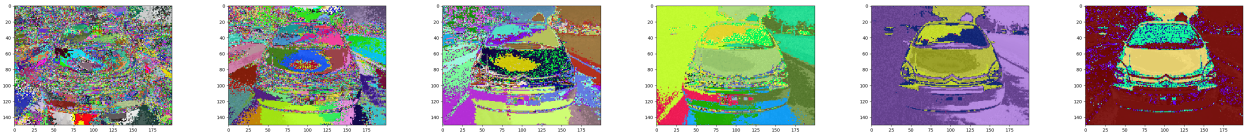
Figure 2: Segmentation results of figure1 with Gaussian kernel.



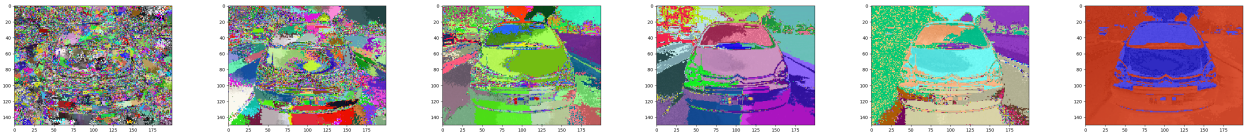
3.1:r=5,α=0.0 3.2:r=10,α=0.0 3.3:r=20,α=0.0 3.4:r=30,α=0.0 3.5:r=50,α=0.0 3.6:r=100,α=0.0



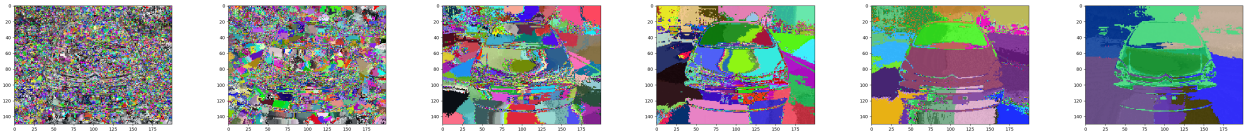
3.7:r=5,α=0.1 3.8:r=10,α=0.1 3.9:r=20,α=0.1 3.10:r=30,α=0.1 3.11:r=50,α=0.1 3.12:r=100,α=0.1



3.13:r=5,α=0.3 3.14:r=10,α=0.3 3.15:r=20,α=0.3 3.16:r=30,α=0.3 3.17:r=50,α=0.3 3.18:r=100,α=0.3



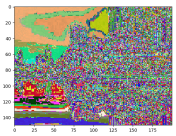
3.19:r=5,α=0.5 3.20:r=10,α=0.5 3.21:r=20,α=0.5 3.22:r=30,α=0.5 3.23:r=50,α=0.5 3.24:r=100,α=0.5



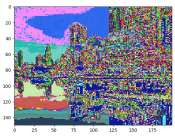
3.25:r=5,α=1.0 3.26:r=10,α=1.0 3.27:r=20,α=1.0 3.28:r=30,α=1.0 3.29:r=50,α=1.0 3.30:r=100,α=1.0

Figure 3: Segmentation results of figure1 with flat kernel.

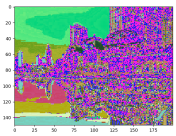
Figure 4: The origin figure of figure2.



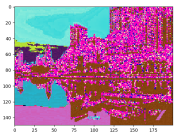
5.1:r=5,α=0.0



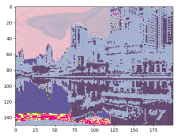
5.2:r=10,α=0.0



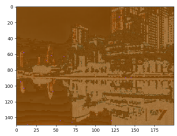
5.3:r=20,α=0.0



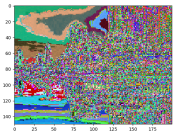
5.4:r=30,α=0.0



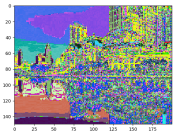
5.5:r=50,α=0.0



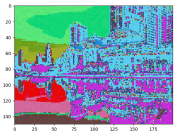
5.6:r=100,α=0.0



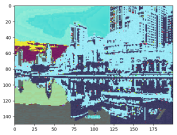
5.7:r=5,α=0.1



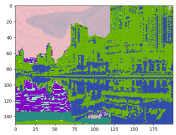
5.8:r=10,α=0.1



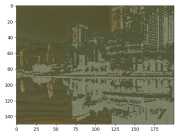
5.9:r=20,α=0.1



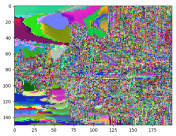
5.10:r=30,α=0.1



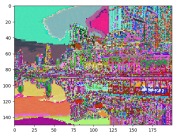
5.11:r=50,α=0.1



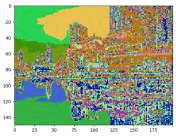
5.12:r=100,α=0.1



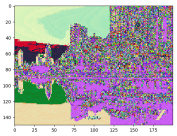
5.13:r=5,α=0.3



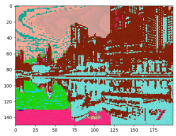
5.14:r=10,α=0.3



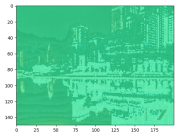
5.15:r=20,α=0.3



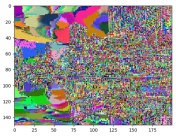
5.16:r=30,α=0.3



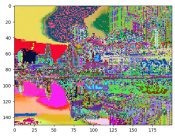
5.17:r=50,α=0.3



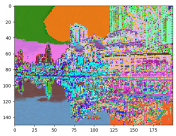
5.18:r=100,α=0.3



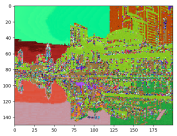
5.19:r=5,α=0.5



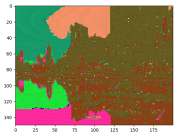
5.20:r=10,α=0.5



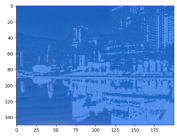
5.21:r=20,α=0.5



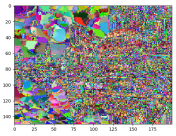
5.22:r=30,α=0.5



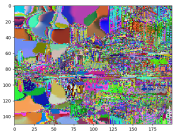
5.23:r=50,α=0.5



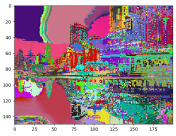
5.24:r=100,α=0.5



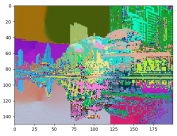
5.25:r=5,α=1.0



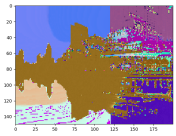
5.26:r=10,α=1.0



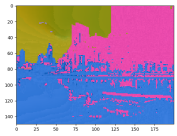
5.27:r=20,α=1.0



5.28:r=30,α=1.0

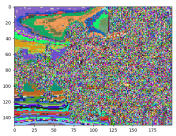


5.29:r=50,α=1.0

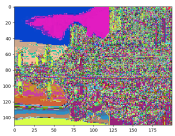


5.30:r=100,α=1.0

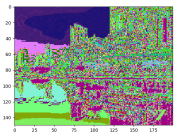
Figure 5: Segmentation results of figure2 with Gaussian kernel.



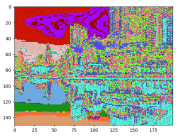
6.1:r=5,α=0.0



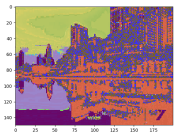
6.2:r=10,α=0.0



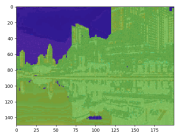
6.3:r=20,α=0.0



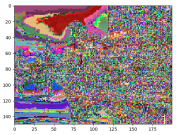
6.4:r=30,α=0.0



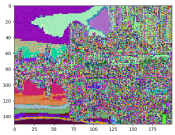
6.5:r=50,α=0.0



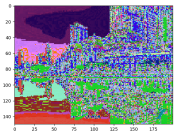
6.6:r=100,α=0.0



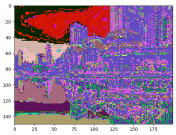
6.7:r=5,α=0.1



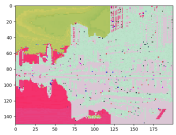
6.8:r=10,α=0.1



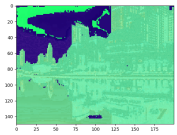
6.9:r=20,α=0.1



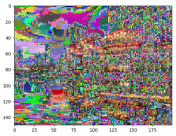
6.10:r=30,α=0.1



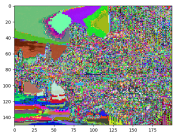
6.11:r=50,α=0.1



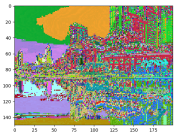
6.12:r=100,α=0.1



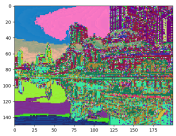
6.13:r=5,α=0.3



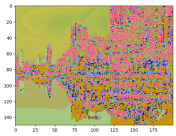
6.14:r=10,α=0.3



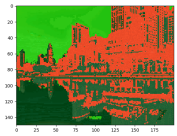
6.15:r=20,α=0.3



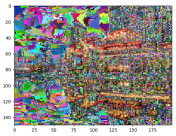
6.16:r=30,α=0.3



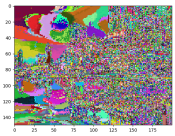
6.17:r=50,α=0.3



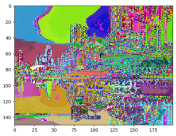
6.18:r=100,α=0.3



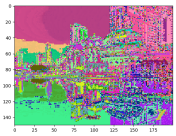
6.19:r=5,α=0.5



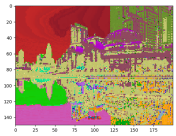
6.20:r=10,α=0.5



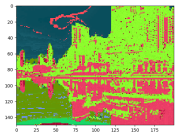
6.21:r=20,α=0.5



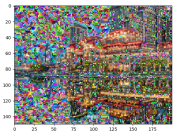
6.22:r=30,α=0.5



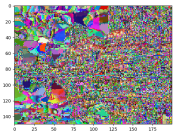
6.23:r=50,α=0.5



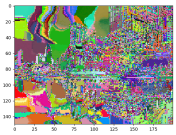
6.24:r=100,α=0.5



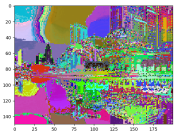
6.25:r=5,α=1.0



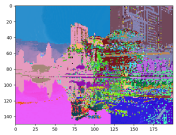
6.26:r=10,α=1.0



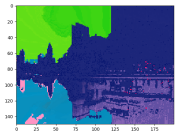
6.27:r=20,α=1.0



6.28:r=30,α=1.0



6.29:r=50,α=1.0



6.30:r=100,α=1.0

Figure 6: Segmentation results of figure2 with flat kernel.