Image Segmentation using Mean Shift Algorithm

Letian Chen, School of Psychological and Cognitive Sciences

1 Introduction

From image classification, detection to segmentation, the task is getting harder and harder. Image classification only requires a label for the entire image, while detection requires the program to detect several objects with bounding box and label them separately. However, segmentation requires more precisely segment objects in the pixel level, i.e. to divide the pixels of image to several classes. It is clear the segmentation is much harder and has more computational burden.

1.1 Task

The task for this homework is image segmentation. As is said before, image segmentation requires the program to correctly divide image in the pixel level, labeling every pixel a class. For example, in the Fig. 1, the human is separated from sheep and sheep is separated from each other.

1.2 Algorithm Review

Researchers have proposed a lot of algorithms to tackle the hard image segmentation problem. We could divide them into two categories: classic and deep-learning method.

The classic algorithm includes unsupervised learning and interactive learning. In unsupervised learning algorithms, the "mean shift" [1] and "Graph Cut" [2] algorithm is the most widely accepted algorithm. Mean shift is the main algorithm in this report and we will introduce it in the Section 2. "Graph Cut", however, is to model the image as an graph and use minimal flow in graph theory to solve a minimum graph cut in the image. Interactive learning means that user could interact with the program somehow, for example, draw a rough outline for image matting and then run segmentation algorithms using the sketch as a guide. The representative algorithm in interactive learning includes "Grab Cut" [3] and lazy-snapping [4].

As deep learning develops, deep-learning based image segmentation algorithms have been brought forward and their performance has exceeded classic methods. The representative work includes FCN [5], SegNet [6] and DeepLab [7]. As the goal of this homework is to implement a classic algorithm, we won't explain deep algorithms here.

1.3 This Report

In this report, we introduced the task and some classic algorithms in Section 1. Then, we will explain the method we use, the difficulty we face and the solution to tackle these problems in Section 2. After that, we will show our algorithm's result. Finally we will have a discussion in Section 4.

2 Method

2.1 Mean Shift

Mean Shift algorithm is proposed by Comaniciu, Dorin and Meer, Peter in 2002 [1]. It is a unsupervised algorithm, which means that it do not need training to find the structure of data, in this case, several clusters of pixels. The main idea of "Mean Shift" algorithm is to find several mode in the image, and points that converge to the same mode are considered as the same cluster.

Specifically, the algorithm contains two steps:

1. Mode Search: In this step, we will try searching modes from several start points using mean shift.
2. Mode Cluster: Combine near modes according to distance measure.

Mode search is described as follows:

1. Initialize $y_{i,0} = x_i$, x_i is the i-th start point. The start point is chosen every h_s row and h_s column. Note that x_i does not only contain location information but also the RGB

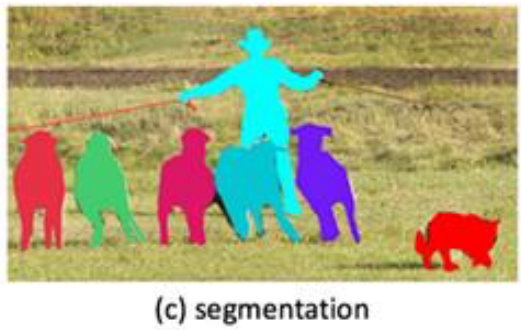


Figure 1. Segmentation Demonstration

value of that location. It is because mean shift should be performed on feature space, while in this task the feature space consists of location (x, y) and color (RGB). Thus, x_i is a five length vector.

2. Find all data points that satisfy the condition that $\|\frac{z_j - y_{i,k}}{h}\| <= bandwidth$, in which *bandwidth* is a free parameter to control the bandwidth of "near" definition. It is worth noting that h in the denominator indicates that adjustment for different dimensions in norm operation. For example, if the feature space contains coordination and color, it is obvious that they have different scales. Therefore, h is a vector to make every dimension has the same scale.

3. After filtering z_j, we need to calculate the new mean by $y_{i,k+1} = \frac{\sum_{n=1}^{N} g(z_n)}{N}$, i.e. regard the mean of all "nearby" z_j as the new mean. The $g()$ function is a kernel function controlling the weight according to how far it is to the current mean when summed.

4. If the new mean $y_{i,k+1}$ is very similar to the previous one $|y_{i,k+1} - y_{i,k}| <= bandwidth$, it means the iteration has converged, go to step 5, otherwise, go to step 2 and iterates again.

5. Record the converged mean $y_{i,c}$.

Mode Cluster is described as follows: for every converged mean m_i that are not deleted, iterate all mean points m_j and check their distance to m_i. The distance definition is the same as mode search: $\|\frac{z_j - y_{i,k}}{h}\|$. If the distance is lower than *bandwidth*, then delete m_j.

2.2 Implementation

My code implementation is based on the code of https://github.com/Pranshu258/meanshift. I mainly modified the calculation method of it (from serial to parallel), which will be explained in detail after.

To use my code,

1. Choose one of *run_uniform.sh* or *run_Gaussian.sh*. The former one's $g()$ is a uniform function while the latter one's $g()$ is a Gaussian function.

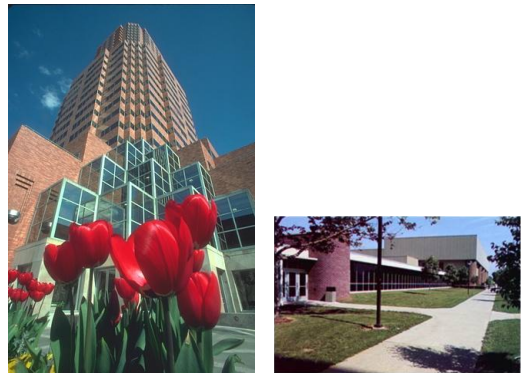
2. Edit the $image = "XXX"$ line in either *run_uniform.sh* or *run_Gaussian.sh* to adjust the path of the original image.

3. Use linux style run *./run_uniform.sh* or *./run_Gaussian.sh* to run the code.

4. It will automatically generate 8 images using bandwidth of 10, 20, 30, 40, 50, 60, 70, 80.

The mean shift module and mean cluster module's method is the same as described in Section 2.1. The scale adjuster h is $(1, 1, 1/5, 1/5, 1/5)$, in correspondence to (x, y, R, G, B) in feature space. We will see the different *bandwidth*'s impact in Section 3. The starting point stride h_s we use is 40 since the image size is approximately 400×400, and in this way we could generate 100 initial start points. The way we generate starting point is: for every 40×40 pixels, we will generate a random pixel as one of starting points.

To generate the segmented image, we will calculate the nearest converged mode for every pixel, and then fill that pixel's RGB for them. It is worth noting that this is not the best method. The best method should be calculate the converged modes



(a) Input 1 (b) Input 2

Figure 2. Inputs

for every pixel, but the computational burden that way is quite too much and I can't just get the result of it using my laptop. Therefore, the result may not that good compared with the correct method, but it is still showing significant segmentation success.

2.3 Difficulty and Solution

Although it seems easy, I encounter many difficulties during the implementation and debugging.

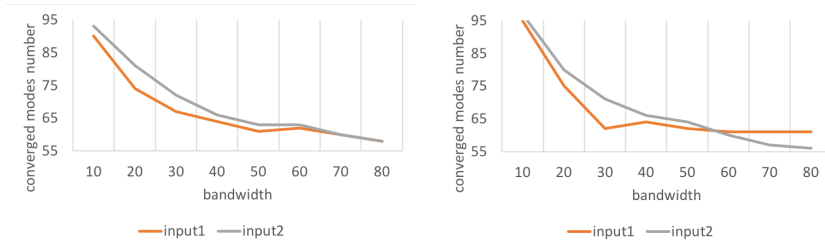
1. Parallelization: The maximum iteration times we set for each start point is 15, which means we may have to do 15 iterations for about 100 start points in the worst case. In fact, is takes me about 90s to get the result. What's worse, we will need to calculate the value for every pixel in the final segmented image drawing module, though we have done some simplification as mentioned in Section 2.2 to use nearest modes rather than the true converged modes. However, the program is well-parallelizable. We could easily use separate CPU core to do iterations for each 100 start points. We could also easily calculate the value for every pixel separately on separate CPU cores. After transforming the serial program into a parallel one, the mode search time decreases from 90s to 10s on a 12-core CPU. The drawing part's time decreases from I-don't-know to 70s. Since I want to do the comparison in different bandwidth, the decreased calculation time is really essential.

2. Choose of image: because of the difficulty of the hard, choose of image is very important if you want to see good results. As a matter of fact, mean shift is just an unsupervised algorithm, which means that the data need to demonstrate a clear pattern for the algorithm to find because the algorithm does not know the exact answer. After trying, the rule of thumb I think is to choose images that has clear color separation is a better choice since we only use location and color in our feature space. Fig. 1 is a good example since human, sheep and background's color are all clearly different. If the difference if mainly on texture, we may use a convolution filter to firstly get texture features and add them in the feature space.

3. Bandwidth: bandwidth strongly influence the performance of the algorithm. In fact, larger bandwidth will cause unrelated modes to be combined, while smaller bandwidth will cause too much modes to have a good segmentation results. Thus, we tried different bandwidth in two images to see how bandwidth influence the segmentation result and what's the best bandwidth generally.

3 Result

We use two images to demonstrate the segmentation result. They are shown in Fig. 2. We could see that the first image is harder than the second one because the second one's color separation is obvious while the first one's color on flowers and leaves is strongly influenced by light.



(a) Uniform (b) Gaussian

Figure 3. Converged modes number in different Bandwidths

(a) Bandwidth=10 (b) Bandwidth=30

Figure 4. Segmentation Result for Input 2 in Different Bandwidth

Firstly, we'd like to see how bandwidth will influence the segmentation result. Different converged modes numbers in different bandwidths is shown in Fig. 3. Fig. 3(a) uses uniform $g()$ in calculating the new mass mean, while Fig. 3(b) uses Gaussian $g()$. We could see that in uniform $g()$, the bandwidth of 40 seems to be a turning point that after bigger bandwidth will result in no change of converged modes number. Nonetheless, in the Gaussian $g()$, the turning point is at the bandwidth of 30. Therefore, we regard the 40 is the best bandwidth for uniform $g()$ while 30 is the best bandwidth for Gaussian $g()$. We could also see the effect of different bandwidth in Fig 4.

We demonstrate our segmentation result for input1 in Fig. 5. It is obvious that both uniform and gaussian $g()$ successfully segment the image into 60+ classes. But as we said before, this image's color is highly influenced by light, which makes that flowers and leaves being separated into several classes.

In the segmentation result for input 2 (See Fig. 6), we could clearly see the kermesinus cylindrical building is successfully divided into three classes, from most bright to most dim. The grass is almost separated into the same class. Therefore, in order to make mean shift algorithm do its job better, we should pick images that different objects have different colors.

We could also see that gaussian segmentation is a little bit better than uniform segmentation, since it seems more smooth in both input1 and input2 figures.

4 Discussion

In this project, we learned the task of image segmentation, the idea of mean shift and implement our own version of mean shift algorithm. The algorithm works well in two representative images, though the result seems better in the second image because it is relatively easy because of the obvious color separation in different segments. Also, we demonstrate the difference when using different bandwidths in the separation. We conclude that for uniform mean calculation, bandwidth of 40 is better while for Gaussian mean calculation, bandwidth of 30 is better. We also parallelize the program to make it at least 10x faster than the original implementation.

However, our implementation still have several problems. For example, as said in Section 2.2, we do not use every pixel's true convergence point to be its class, but use the nearest modes as its class to decrease the computational burden. In fact, the calculation could be further parallelized

(a) Original (b) Uniform (c) Gaussian

Figure 5. Segmentation Result for Input 1

(a) Original (b) Uniform (c) Gaussian

Figure 6. Segmentation Result for Input 2

to make exact calculation possible. Moreover, we could also transform many calculation into matrix calculation to put them run on GPU to gain an even faster performance. Besides, mean shift is still only an unsupervised algorithm. Hence, the segmentation result is not very good compared to interactive algorithms or trained deep-learning algorithms. To make it better, the choose of feature space is essential.

5 Reference

[1] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 5, pp. 603–619, 2002.

[2] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[3] C. Rother, V. Kolmogorov, and A. Blake, "Grabcut: Interactive foreground extraction using iterated graph cuts," in *ACM transactions on graphics (TOG)*, vol. 23, pp. 309–314, ACM, 2004.

[4] Y. Li, J. Sun, C.-K. Tang, and H.-Y. Shum, "Lazy snapping," in *ACM Transactions on Graphics (ToG)*, vol. 23, pp. 303–308, ACM, 2004.

[5] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[6] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2018.