DIP: Final project report

Image segmentation based on the normalized cut framework

Yu-Ning Liu    Chung-Han Huang    Wei-Lun Chao
R98942125      R98942117        R98942073

Motivation

Image segmentation is an important image processing, and it seems everywhere if we want to analyze what inside the image. For example, if we seek to find if there is a chair or person inside an indoor image, we may need image segmentation to separate objects and analyze each object individually to check what it is. Image segmentation usually serves as the pre-processing before image pattern recognition, image feature extraction and image compression. Researches of it started around 1970, while there is still no robust solution, so we want to find the reason and see what we can do to improve it.

Our final project title is a little bit different from the proposal. The title of the proposal is “Photo Labeling Based on Texture Feature and Image Segmentation”, while during the execution, we change it into “Image segmentation based on the normalized cut framework”. The main reason is that we found there are many kinds of existed image segmentation techniques and methods, in order to gain enough background, we went through several surveys and decided to change the title into a deep view of image segmentation.

1. Introduction

Image segmentation is used to separate an image into several “meaningful” parts. It is an old research topic, which started around 1970, but there is still no robust solution toward it. There are two main reasons, the first is that the content variety of images is too large, and the second one is that there is no benchmark standard to judge the performance. For example, in figure 1.1, we show an original image and two segmented images based on different kinds of image segmentation methods. The one of figure 1.1 (b) separates the sky into several parts while the figure 1.1 (c) misses some detail in the original image. Every technique has its own advantages also disadvantages, so it’s hard to tell which one is better.

There are tons of previous works about image segmentation, great survey resources could be found in [1, 2, 3]. From these surveys, we could simply separate the image segmentation techniques into three different classes (1) feature-space based method,
(2) **image-domain based method**, and (3) **edge-based method**. The feature-space based method is composed of two steps, feature extraction and clustering. Feature extraction is the process to find some characteristics of each pixel or of the region around each pixel, for example, pixel value, pixel color component, windowed average pixel value, windowed variance, Law’s filter feature, Tamura feature, and Gabor wavelet feature, etc.. After we get some symbolic properties around each pixel, clustering process is executed to separate the image into several “meaningful” parts based on these properties. This is just like what we have tried from DIP homework 4, where we used Law’s feature combined with K-means clustering algorithm. There are also many kinds of clustering algorithms, for example, Gaussian mixture model, mean shift, and the one of our project, “normalized cut”.

![Figure 1.1: (a) is the original image, (b) is the segmentation result based on [6], and (c) is the result from [7].](image)

Image-domain based method goes through the image and finds the boundary between segments by some rules. The main consideration to separate two pixels into different segments is the pixel value difference, so this kind of methods couldn’t deal with textures very well. Split and merge, region growing, and watershed are the most popular methods in this class. The third class is edge-based image segmentation method, which consists of edge detection and edge linking.

Although there have been many kinds of existed methods, some common problem still can’t be solved. For class (1), the accurate boundaries between segments are still hard to determine because features take properties around but not exactly on each pixel. Class (2) only uses the pixel value information, which may result in over-segmentation on texture regions. Finally the edge detection process makes class (3) always suffer the over-segmentation problem. In our project, we adopt the “normalized cut framework” for image segmentation, which finds the best cutting path from the global view (the whole image view) rather than by local thresholds and is expected to have better segmentation results than other methods. In section 2, the basic idea of normalized cut framework and its mathematical derivation is presented,
and in section 3, we talk about the features we adopt for similarity measurement. In section 4, we perform our image segmentation methods on several kinds of image and show the results. And finally in section 5, we’ll give discussion and conclusion about our project, and also list some future works that we can keep going for the advanced research purposes.

2. Normalized cut framework

The normalized cut framework is proposed by J. Malik and J. Shi [8]. In their opinion, the image segmentation problem can be seen as a graph theory problem. Graph theory is an interesting math topic which models math problems into arcs (edges) and nodes. Although it’s hard to explain graph theory in this project report, we give two practical examples to give readers more ideas about what it can do. In figure 2.1, a graph model of Taiwan map is presented, where we models each county as a node, and the edge between two nodes means these two counties are connected in the original map. This model could be used for coloring problems (give each county a color, while connected county should have different colors), or transportation flow problems in advanced. Each edge in the model could contain a value (weight), which could be used as flow or importance of it. This kind of graph is called “weighted graph”, and is frequently adopted by internet researchers.

![Figure 2.1: (a) is a simplified Taiwan map, and (2) is the graph model of (a), which models each county as a node, and if two counties are connected, an edge is drawn between them.](image)

2.1 Introduction

In the normalized cut framework, we also model the image into a graph. We model each pixel of the image as a node in the graph, and set an edge between two nodes if there are similarities between them. The normalized cut framework is composed of
two steps: similarity measurement and normalized cut process. The first step should be combined with feature extraction and we’ll talk about this step in section 3. The purpose of this step is to compute the similarity between pixels and this value is set as the weight on the edge. In order to model all the similarities of an image, all pair of pixels will contain an edge, which means if an image contains \( N \) pixels, there will be totally \((N - 1)N/2\) edges in the corresponding graph. This kind of graph is called “complete graph” and needs a large memory space. To simplify the problem, sometimes we set edges between two nodes only when their distance is smaller than a specific threshold. For example, in figure 2.2, we show an example about modeling an image into a graph. Edges with blue color mean weak similarities, while edges with red color mean strong similarities.

![Figure 2.2](image)

Figure 2.2: (a) is the original image, and in (b) this image has been modeled as a graph: each pixel as a node, and a pair of nodes have an edge only if their distance is equal to 1. Edges with blue color mean weak similarities, while edges with red color mean strong similarities.

This is a connected graph because each pixel could go through the edges to reach all other pixels else. The term “cut” means eliminating a set of edges to make the graph “unconnected”, and the cut value is the total weights on this set of edges. For example, if we eliminate all the blue edges in figure 2.2, then the nodes with white color will be “unconnected” to the nodes with dark color, and now we say the graph has been separate into two connected graph (the outside dark group and the inner white group). So, from the graph theory, the image segmentation problem is modeled as graph cut problem. But there are many kinds of cutting path we can adopt to separate the image into two part, we must have to follow some criterions. Remember that the weights on edges have the similarity meaning between pixels, so if we want to separate two pixels into two different groups, their similarity is expected to be small.

Three kinds of cutting criterions have been proposed in recent years: (1) minimum cut, (b) minimum ratio cut, and (c) minimum normalized cut, and the normalized cut has
been proved to maintain both high difference between two segments and high similarities inside each segments. So in our project, we adopt the normalized cut framework.

2.2 The formula for finding normalized cut

In these two subsections, we’ll present the mathematical derivation and algorithm implementation about how to find the normalized cut in a given image. The original derivation is presented in [6], and here I just give a short summary.

A graph $G = (V, E)$ can be partitioned into two disjoint sets, $A, B, A \cup B = V, A \cap B = \emptyset$, by simply removing edges connecting the two parts. The degree of dissimilarity between these two pieces can be computed as total weight of the edges that have been removed. In graph theoretic language, it is called the cut: [6]

$$\text{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v) \quad (2.1)$$

The normalized cut then could be defined as:

$$N_{\text{cut}}(A, B) = \frac{\text{cut}(A, B)}{\text{asso}(A, V)} + \frac{\text{cut}(A, B)}{\text{asso}(B, V)} \quad (2.2)$$

where

$$\text{asso}(A, V) = \sum_{u \in A, t \in V} w(u, t)$$

is the total connection from nodes in $A$ to all nodes in the graph, and $\text{asso}(B, V)$ is similarly defined. In the same spirit, we can define a measure for total normalized association within groups (a measure for similarities inside each group) for a given partition: [6]

$$N_{\text{asso}}(A, B) = \frac{\text{asso}(A, A)}{\text{asso}(A, V)} + \frac{\text{asso}(B, B)}{\text{asso}(B, V)} \quad (2.3)$$

Here an important relation between $N_{\text{cut}}(A, B)$ and $N_{\text{asso}}(A, B)$ could be derived as followed:

$$N_{\text{cut}}(A, B) = \frac{\text{cut}(A, B)}{\text{asso}(A, V)} + \frac{\text{cut}(A, B)}{\text{asso}(B, V)}$$

$$= \frac{\text{asso}(A, V) - \text{asso}(A, A)}{\text{asso}(A, V)} + \frac{\text{asso}(B, V) - \text{asso}(B, B)}{\text{asso}(B, V)}$$

$$= 2 - \left( \frac{\text{asso}(A, A)}{\text{asso}(A, V)} + \frac{\text{asso}(B, B)}{\text{asso}(B, V)} \right)$$

$$= 2 - N_{\text{asso}}(A, B) \quad (2.4)$$

From this equation, we see that minimizing the disassociation between groups is identical to maximizing the association within each group.

2.3 Implementation algorithm
In [6], the normalized cut problem has been derived into a general eigenvector problem. In this subsection, we just list the most important equations, and readers who are interested in the total process could get more in [6].

Assume now we want to separate an image \( V \) with size \( M \times N \) into two parts, we need to define two matrices: \( W \) and \( D \), both of size \((MN)\times(MN)\). The matrix \( W \) is the similarity matrix with element \( w_{ij} \) as the similarity between the \( i^{th} \) pixel and the \( j^{th} \) pixel. The matrix \( D \) is a diagonal matrix and each diagonal element \( d_i \) contains the sum of all the elements in the \( i^{th} \) row in \( W \). With these two matrices, finding the minimum normalized cut of image \( V \) into two parts \( A \) and \( B \) is equal to solve the equation as followed:

\[
\min N_{\text{cut}} = \min_y \frac{y^T(D-W)y}{y^TDy}
\]  

(2.5)

where \( y \) is an \((MN)\times1\) vector with each element indicating the attribute of each pixel into the two groups. Equation (2.5) could be further simplified into a general eigenvector problem as followed:

\[
(D - W)y = \lambda Dy
\]  

(2.6)

*The eigenvector \( y \) with the second smallest eigenvalue is selected for image segmentation.* The element values in \( y \) can contain all real numbers, so a threshold should be defined to separate pixels into two groups.

**2.4 Determine the vector \( y \)**

There are several kinds of ways to define this threshold, for example, **zero, mean value**, and **medium value** among all the elements in \( y \). In our project, we use these three kinds of thresholds to get three different \( y \). If an element in \( y \) is larger than the threshold, we set the element as 1, otherwise as \(-b\). The value \( b \) is defined as:

\[
b = \frac{\sum_{y > \text{threshold}} d_i}{\sum_{y < \text{threshold}} d_i}
\]  

(2.7)

We substitute the rebuilt \( y \) into equation (2.5) and choose the \( y \) with the minimum normalized cut value. Based on the two element values, we can separate pixels into two groups.

**2.5 Summary of the algorithms**

Figure 2.3 is the flowchart of the normalized cut framework. First we model the image into a graph and get the matrices \( W \) and \( D \). Then we solve equation (2.6) and (2.5) to get the rebuilt \( y \) and separate the image into two segments. The normalized
cut could only separate a segment into two parts in each iteration, so we need to recursively do the separation process to each segment. There is a diamond-shape block in the flowchart which serves the stopping mechanism of the recursive operation. For each segment, we check its area (number of pixels inside) and the further minimum normalized cut value, if the area is smaller than a defined threshold or the further minimum normalized cut value is larger than another defined threshold, the further separation process for this segment stops. The W and D for each segment could directly be extracted from the original W, so we don’t have to rebuild it at each iteration. With this flowchart, we could solve the minimum normalized cut problem by Matlab programs and implement the image segmentation operation.

Figure 2.3: The flowchart of the normalized cut framework

3. Feature extraction and similarity measurement

Since we want to segment different objects into different regions, the first step we need is to compute the feature of each pixel and compute the similarity of each pair of pixels before we separate them. We have several methods to calculate image features including luminance for non-texture images and texton [10], a powerful tool we use for texture images. We also find some papers using texton and contour information [9] and get better performance while resulting in complicated computation. In our approach, we adopt luminance (RGB) based, texton based, and the adaptive method combining luminance and texton for feature computation. After extracting features of
each pixel, we used some general distance measure methods to count the similarity between each pair of pixels. In this section, we’ll discuss these procedures in detail.

3.1 Luminance

3.1.1 Gray level image

For gray level images, we can simply use pixel value or the averaged pixel value in a window as the feature of each pixel.

3.2.2 Color image

For color images, we have many methods to identify the pixel including RGB (red, green, blue), HSV (hue, saturation, value) and HSL (hue, saturation, lightness) …etc. In our approach, we compute the mean square difference of the RGB values as the similarity between two pixels. Figure 3.1 and 3.2 show the color component basis and their responses on a natural image.

![Image](a) (b)

Figure 3.1: (a) is the color representation based on RGB color space, and (b) is based on HSV color space.

3.2 Texton

We have three steps to realize texton as their feature. (1) filter bank, (2) K-means algorithm, and (3) Histogram as Texton. We have 62 filters totally in our filter bank, including edge filters, bar filters, Gabor wavelet filters, Gaussian filter, and Difference of Gaussian (DOG) filter, and we perform several scales and orientations on each group of filters. After building the filter bank, we have a 62-dimension vector for each pixel as their feature. Then we perform the K-means clustering algorithm to catalog these feature representations into a fixed number of groups (Each group is called a texton) and assign a group index to each pixel. Later we compute the texton-index histogram in a sliding window (containing several pixels
inside) around each pixel as the final feature representation of this pixel. So finally we have a histogram for each pixel in the image with each bin as the occurring probability of each texton inside the sliding window, and the number of bins of the histogram is equal to the number of groups after the K-means clustering operation.

Figure 3.2: (a) is the HSV decomposition result of a natural image, and the order from top down is the original image, the hue, saturation, and the luminance component. (b) is the RGB decomposition result of the same natural image, and the order from top down is the original image, the red, green, and the blue component.

3.2.1 Filter bank
a. bar filter

Figure 3.3: The bar filters. Each column is of different orientations, and each row means different scales (from top down: 5x5, 7x7, and 9x9).

b. edge filter

Figure 3.4: The result of bar filters on the Lena image. There are 18 sub-images inside and each ones is the response after the filter in the corresponding location in figure 3.3.

Figure 3.5: The edge filters. Each column is of different orientations, and each row means different scales (from top down: 3x3, 5x5, and 9x9).
Figure 3.6: The result of edge filters on the Lena image. There are 18 sub-images inside and each one is the response after the filter in the corresponding location in figure 3.5.

c. **Gabor wavelet filter**

Figure 3.7: The Gabor wavelet filters. Each row is of different orientations, and each column means different scales.

Figure 3.8: The result of Gabor wavelet filters on the Lena image with 3 scales and 8 orientations.

d. **Gaussian filter**

Figure 3.9: The shape of the Gaussian filter (left) and its result on the Lena image (right).
e. Difference of Gaussian

![DOG filter](image)

Figure 3.10: The shape of the DOG filter (left) and its result on the Lena image (right).

After these five kinds of filters, we totally have 62 filters response on the input image, which means the feature of each pixel is a 62-dimensional vector.

### 3.2.2 K-means clustering

It is a famous tool to handle unsupervised classification problem. We show an example of \( k=3 \) groups in figure 3.11. The green points on the left graph mean we have a lot of data points in the space domain. After K-means we can separate these data points into three groups.

![K-means clustering](image)

Figure 3.11: The left-hand side is the original data points in the 3-dimensional space, and the right-hand side is the k-means clustering result (3 groups).

We have two goals to achieve: maintain large distance among data points in different clusters and small distance among data points in the same cluster. Inter-cluster distance must be a large value because we seek that each group has large distance from other groups so that we can clearly observe their boundaries. And if a new data point comes in, it is more obvious that which group the new point should be classified into. On the other side, the intra-cluster distance must be as small as possible so that the points inside are of nearly the same properties and represent the same feature information.
There are two issues in K3means clustering algorithm: (1) How to choose the number of groups K, and (2) Given K, what is the criterion to separate each cluster? For supervised K3means clustering, we already have the value of K, but not the case for un-supervised K3means. To solve this problem, we set K=2 at first, then set the threshold of inter-cluster and intra-cluster distance. If the distance of inter-cluster is bigger than our threshold, we let K=K+1, and redo the same K3means clustering procedure described below.

The algorithm for K3means clustering:

(1) Initialization: Select K vectors randomly as the initial centroids, and do the following iterations.

(2) Step 1, from k cluster centers, use the NN (nearest neighbor) rule to classify each data points into one cluster

(3) Step 2, re-compute the centroid of each cluster.

Figure 3.12 to 3.15 show an example with 2 centroids and 10 data points.

Figure 3.12: At first we form 2 initial centroids, red and green, all of data points are near to the green 1, so all of them will be the cluster 1.

Figure 3.13: The green centroid is a re-computed centroid as the average of all the data points classified into the green group in figure 3.12, and then we calculate the distance from each data to each centroid and do the clustering by NN rule again.
Figure 3.14: Re do the same procedure from figure 3.12 and 3.13, get the new classification result, and find the new set of cluster centroids.

Figure 3.15: The iterations stop when the new set of centroids is nearly the same (under a specific threshold) as the centroids of the iteration at the before iteration.

But we may get the different clustering results with different sets of the initial centroids. Figure 3.16 to figure 3.18 show three examples with 3 clusters.

Figure 3.16: left-hand side is the setting of initial centroids, and the right-hand side is the final clustering result.

Figure 3.17: left-hand side is the setting of initial centroids, and the right-hand side is the final clustering result. As you can see, one centroid has been ignored.
We can know the result will change with different initial centroids. And even we have 3 centroids at first, we may only have 2 groups, the physical meaning is that the centroid didn’t have clustering because it is really far away with these data. We already have the 62-dimensional features for each pixel, then we can execute the K-means clustering algorithm to separate dissimilar features of pixels into different groups, and assign a group index to each pixel.

### 3.2.3 Texton and feature representation

The K centroids after the K-means clustering operation in the before subsection are called the K textons of the image. We compute the texton-index histogram in a sliding window (containing several pixels inside) around each pixel as the final feature representation of this pixel. So finally we have a histogram for each pixel in the image with each bin as the occurring probability of each texton inside the sliding window, and the number of bins of the histogram is equal to the number of groups after the K-means clustering operation. Figure 3.19 shows the representation method.
After all, we have histogram as the feature representation for each pixel. We can use this information to compute the similarity matrix and then segment this image.

### 3.3 Similarity measurement

To calculate the similarity between each pixel, we combine two measurements: feature similarity and spatial similarity.

1. **Feature similarity:**
   At first we use X2 distance to compute the distance between two histograms:
   \[
   
   x^2(h_i, h_j) = \frac{1}{2} \sum_{l=1}^{L} \frac{[h_i(l) - h_j(l)]^2}{h_i(l) + h_j(l)}
   \]
   (3.1)
   Then we use exponential function as the distance measurement. As the distance getting far away, \( W_f \), which means the similarity will be smaller, where the \( \delta_f \) is a constant.
   \[
   W_f = \exp\left(-\frac{x^2(h_i, h_j)}{\delta_f}\right)
   \]
   (3.2)

2. **Spatial similarity:**
   But it is not sufficient if we care only the feature similarity, we also need to consider spatial similarity because general speaking if the pixels are far away, the similarity will be smaller:
   \[
   W_p = \exp\left(-\frac{(x_i - x_j)^2}{\delta_p}\right), \text{ if } \|(x_i - x_j)\| < r
   \]
   (3.3)
   \( \delta_p \) is a constant and \( r \) is a threshold if the distance too far away, we don’t have to consider the similarity. Finally we have a similarity matrix. If the size of image is \( M \)-by-\( N \), the size of similarity matrix will be \( (MN) \)-by-\( (MN) \).

### 4. Experimental result

We perform our image segmentation method on several kinds of images, including the cartoon images, landscape images, and the texture images, etc., and in this section we’ll show these results. Here the distance threshold for building an edge between two pixels (mentioned in subsection 2.1) is set as 3.
4.1 Tests on simple images

In figure 4.1 to 4.3, we test on three simple images with the RGB based similarity measurement, and as you can see, the segmentation results are really good. The boundaries of the eagle, the human shape, and the bear could be detected accurately, and these objects are all segmented into one group in the corresponding images. The only drawback is that the background has been over-segmented, while we can do some post-processing to merge them into fewer groups.

![Figure 4.1](image1.png)

Figure 4.1: (a) is the original eagle image, and the seven sub-images in (b) show the segmentation results, where the nonblack region in each sub-image means each segment. The complete eagle shape is contained in the sub-image at the first row and the first column.

![Figure 4.2](image2.png)

Figure 4.2: (a) is the original human-shaped image, and the five sub-images in (b) show the segmentation results, where the nonblack region in each sub-image means each segment. The complete human shape is contained in the sub-image at the first row and the third column.
Figure 4.3: (a) is the original “Wini poor bear” image, and the six sub-images in (b) show the segmentation results, where the nonblack region in each sub-image means each segment. The complete eagle shape is contained in the sub-image at the first row and the first column.

4.2 Tests on landscape and natural images

Besides these three simple images, we also try our method on landscape images and natural images. Figure 4.4 is a test on a landscape image with mountain, lake, and sky inside, and our method could segment different natural objects into different groups, especially the sky with cloud inside. Here the over segmentation problem occurs again on the lake region. Figure 4.5 is another test on an insect image, and the body of image could be segmented out exactly. These experiments are based on RGB based similarity measurement.

Figure 4.4: (a) is the original landscape image, and the seven sub-images in (b) show the segmentation results, where the nonblack region in each sub-image means each segment. The complete sky with cloud inside is contained in the sub-image at the first row and the first column.
Figure 4.5: (a) is the original firefly image, and the eight sub-images in (b) show the segmentation results, where the white region in each sub-image means each segment. The complete firefly body is contained in the sub-image at the second row and the second column.

4.3 Test on the Lena image

We also test on the popular “Lena” image and figure 4.6 shows the result. The face region and the shoulder are segmented into the same group because they have the same skin color. The segmented hat region has some overlap with the background region due to obscure boundary between them. This experiment is based on luminance based similarity measurement.

Figure 4.6: (a) is the original Lena image, and the nine sub-images in (b) show the segmentation results, where the white region in each sub-image means each segment.
4.4 Tests on texture images

In the DIP homework 4, we tried to segment an image containing four different texture regions into four groups, and in this project we also test on these kinds of texture images. We first perform our method on the three-textured image shown in figure 4.7 with the RGB based similarity measurement, and the result is shown in figure 4.8. As you can see, the result is not good, different textures have been segmented into the same group due to the large variance of RGB values inside each texture region.

![Figure 4.7: a three-texture image for segmentation testing.](image)

![Figure 4.8: The segmentation result of figure 4.7 with the RGB based similarity measurement.](image)

In order to solve this problem, we try on the texton based similarity measurement, and the result is shown in figure 4.9. The performance is much better than the one with
RGB base, where each feature has been classified into different segments accurately. But we still find that the boundaries between different feature groups can’t be cut at the right position, and this is because the feature of each pixel is computed around in a sliding window, which performs like the averaging operation and make the boundary contrast smoothly.

![Figure 4.9](image)

Figure 4.9: The segmentation result of figure 4.7 with the texton based similarity measurement.

We further try another similarity measurement method which adaptively selected using RGB or texton based method based on the amount of textures around each pixel. At this step, we use the edge density and the RGB value variance around each pixel to define the amount of textures and generate a probability for selecting RGB based similarity or texton based similarity, the idea is shown below:

\[
\text{Similarity} = (1 - P) \times S_{\text{RGB}} + P \times S_{\text{texton}}
\]  

(4.1)

where \( S_{\text{RGB}} \) means the RGB based similarity and \( S_{\text{texton}} \) means the texton based similarity. The \( P \) is the amount of texture around the two pixels for similarity measurement, and this value is constrained between 0 and 1. The result is shown in figure 4.10, and it performs better than the before two methods.

![Figure 4.10](image)

Figure 4.10: The segmentation result of figure 4.7 with the adaptive based similarity measurement.
Finally we try our method on the same image we used in DIP homework 4, and the result is shown in figure 4.11. This result comes from the texton based similarity measurement and it’s really good.

![Image](image.png)

Figure 4.11: (a) is the original “small-texture” image, and the five sub-images in (b) show the segmentation results, where the nonblack region in each sub-image means each segment.

5. Discussion and conclusion

From our experiments, we see that our methods could achieve the image segmentation purpose. For simple images with just a little texture inside, the result is quite good, and this method could also performs well on natural and landscape images. The above tests are based on the luminance or RGB based similarity measurement. When we test our algorithm on the texture images with the same similarity measurement, we found the result become worse. And the main reason is that the pixel value variance is really large in the texture region, which means pixels in the same texture region may have small similarities and are segmented into different groups.

This problem could be solved by using texton based similarity measurement, while the boundary position between segments are not so accurate determined due to the averaging operation by the sliding window. From [9], we knew that estimating the region properties is really an importance process. The Luminance-based method performs well on simple images with just a little texture inside. On the contrary, the
Texton-based method is suitable to process picture with more texture component inside. Therefore, it’s a good issue to use an adaptive method to fit on all pictures.

Another issue in our project implementation is that there exists some uncertainty in K-means clustering and the eigenvector computing functions. We usually get different results when performing the same algorithm on the same image, although the results are still very good, but we may seek more stable way to execute these two steps.

We also find that the computation time of the eigenvectors is very slow. For a 64-by-64 image, it costs 30 seconds to get the final result, while it costs near 5 minutes for a 256-by-256 image. To solve this problem, we go through some papers but there is still no effective solution towards this problem. The idea in [4], which uses a hierarchical structure for computing the similarity may be an worthy method to try, but the algorithm is too complicated and we can’t finish it until the final project presentation.

6. Future works

There are still some things we can do for future works. At first, we will improve the stability of program. We want to modify the code of function to let programs more stable. Secondly, it’s a chance to get a better adaptive method for image segmentation even that we have an adaptive method already. The third one is to generate the post-processing mechanism for region merging. We can write a code about merging groups with the same texture into a single group.

7. Division of Labor

(1) Yu-Ning Liu: PPT, final report, paper survey, similarity measurement and implementation.
(2) Chung-Han Huang: PPT, final report, paper survey, experiment and parameter tuning.
(3) Wei-Lun Chao: PPT, final report, paper survey, normalized cut and implementation.

Reference


