

Homework3

1552746 崔鹤洁

Problem1

Problems:

Please prove that the eigen vector α_2 associated with the second largest eigen-value of C has the following two properties:

- (1) it is orthogonal to α_1 ;
- (2) among all the orientations orthogonal to α_1 , the variance of the data projects along α_2 is the largest one.

Answers:

(1) We assume that λ_1, λ_2 are two different eigen values, and α_1, α_1 are the corresponding eigen vectors, we have:

$$A\alpha_1 = \lambda_1\alpha_1 \quad (1)$$

$$A\alpha_2 = \lambda_2\alpha_2 \quad (2)$$

Transpose these two equations, and multiply α_2 on the right side of equation (1), multiply α_1 on the right side of equation (2), we can get:

$$\alpha_1^T A^T \alpha_2 = \lambda_1 \alpha_1^T \alpha_2 \quad (3)$$

$$\alpha_2^T A^T \alpha_1 = \lambda_2 \alpha_2^T \alpha_1 \quad (4)$$

We know that A is a covariance matrix, therefore, A is a symmetric matrix.

We have noticed that $(\alpha_2^T A^T \alpha_1)^T = \alpha_1^T A^T \alpha_2$ (A is a symmetric matrix, so $A^T = A$)

Together with (3) and (4), we know that $(\lambda_2 \alpha_2^T \alpha_1)^T = \lambda_1 \alpha_1^T \alpha_2$,

that is $\lambda_2 \alpha_1^T \alpha_2 = \lambda_1 \alpha_1^T \alpha_2$

So, $(\lambda_2 - \lambda_1) \alpha_1^T \alpha_2 = 0$

Since $\lambda_1 \neq \lambda_2$, we get $\alpha_1^T \alpha_2 = 0$

So, α_2 is orthogonal to α_1 .

(2) From what is driven from the problem, what we need to prove is that:

$$\alpha_2 = \operatorname{argmax}(\operatorname{var}(\alpha^T X)) = \operatorname{argmax}(\alpha^T C \alpha),$$

in the condition that

$$\alpha^T \alpha = 1(1)$$

$$(\alpha, \alpha_1) = 0(2),$$

From Lagrangian multiplier method, we can get that:

$$F = \alpha^T C \alpha - \lambda_2 (\alpha^T \alpha - 1) - \lambda_1 \alpha^T \alpha_1(3)$$

Calculate the first derivative of F , we can get:

$$\frac{dF}{d\alpha} = 2C\alpha - 2\lambda_2 \alpha - \lambda_1 \alpha_1 = 0(4)$$

Then multiply α_1^T on the leftside of the equation, we can get:

$$2\alpha_1^T C \alpha - 2\lambda_2 \alpha_1^T \alpha - \lambda_1 \alpha_1^T \alpha_1 = 0(5)$$

since $\alpha_1^T \alpha_1 = 1, \alpha_1^T \alpha = 0$, the equation can be simplified to:

$$\lambda_1 = 2\alpha_1^T C \alpha(6)$$

transpose both the left and right side of the equation above, we get:

$$\lambda_1 = 2\alpha^T C \alpha_1(7)$$

Since α_1 is a eigen vector of C , the corresponding eigen value of α_1 is λ_1

Therefore, $C\alpha_1 = \lambda_1 \alpha_1$

$$\text{So, } \lambda_1 = 2\lambda_1 \alpha^T \alpha_1 = 0$$

Together with equation (4), we can get

$$\frac{dF}{d\alpha} = 2C\alpha - 2\lambda_2 \alpha = 0$$

Finally, we know that $C\alpha = \lambda_2 \alpha$

That is to say, the solution of the original equation is the eigenvector of the covariance matrix C ,

since $(\alpha, \alpha_1) \neq 0$, α_1 is not a solution of the original equation

In the condition of $\text{argmax}(\text{var}(\alpha^T X)) = \text{argmax} \lambda_2$

So, λ_2 take the second biggest eigenvalue of covariance C , α is the corresponding eigenvector of λ_2 , that is α_2

So the solution of original equation is $\alpha = \alpha_2$, therefore, in all the vector that orthogonal to α_1 , the one which have the largest variance in X projection is α_2 .

That is to say, we have come to what the problem says, "Among all the orientations orthogonal to α_1 , the variance of the data projects along α_2 is the largest one. "

Problem2

Problems:

1. Description of the parameters used for superpixel segmentation and Gabor Texture feature extraction.

- parameters used for superpixel segmentation:

```
[labels, numlabels] = slicomex(img,numOfSuperpixel);
```

Parameters:

- `img`: the image being segmented.
- `numOfSuperPixel`: the appropriate value of the number of superpixels after segmenting.

Return Values:

- `labels`: a matrix which have the same size of the image, it shows the superpixel index of each single pixel.
 - `numlabels`: the accurate number of superpixels.
- parameters used for Gabor Texture feature:

```
[EO, BP] = gaborconvolve(im, nscale, norient, minWaveLength, mult, sigmaOnf, dThetaOnSigma, Lnorm, feedback)
```

Parameters:

- `im`: Image to be convolved.
- `nscale`: Number of wavelet scales. Here we test 4 scales.
- `norient`: Number of filter orientations. Here we test 4 norient.
- `minWaveLength`: Wavelength of smallest scale filter.
- `mult`: Scaling factor between successive filters.
- `sigmaOnf`: Ratio of the standard deviation of the Gaussian describing the log Gabor filter's transfer function in the frequency domain to the filter center frequency.
- `dThetaOnSigma`: Ratio of angular interval between filter orientations and the standard deviation of the angular Gaussian function used to construct filters in the freq plane.
- `Lnorm`: Optional integer indicating what norm the filters should be normalized to. A value of 1 will produce filters with the same L1 norm, 2 will produce filters with matching L2 norm. the default value of 0 results in no normalization (the filters have unit height Gaussian transfer functions on a log frequency scale)
- `feedback`: Optional parameter. If set to 1 a message indicating which orientation is being processed is printed on the screen.

Return Values:

- `EO`: 2D cell array of complex valued convolution results, $EO\{s,o\}$ = convolution result for scale s and orientation o .
The real part is the result of convolving with the even symmetric filter, the imaginary part is the result from convolution with the odd symmetric filter.

Hence:

$\text{abs}(\text{EO}\{s,o\})$ returns the magnitude of the convolution over the image at scale s and orientation o .

$\text{angle}(\text{EO}\{s,o\})$ returns the phase angles.

- `BP`: Cell array of bandpass images corresponding to each scale s .

2. Gabor texture feature examples (output of the `gaborconvolve` function) for several scales and orientations for at least five different images.

Image 1: bee.jpg


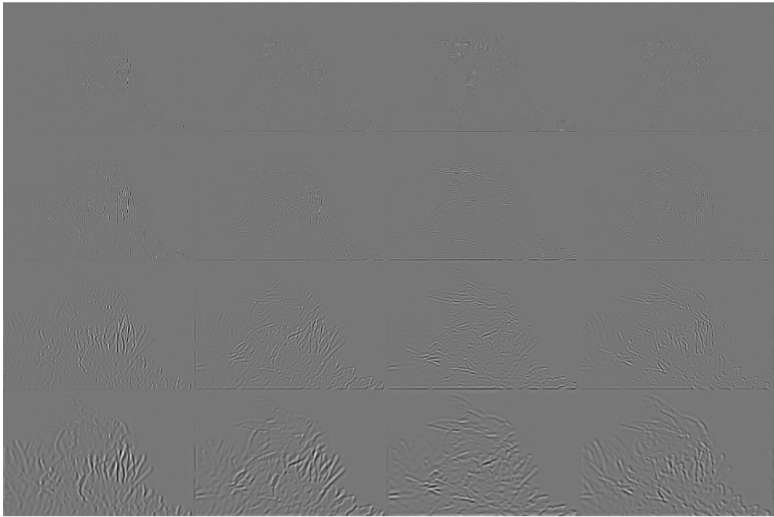
Original	Gabor convolve (4 scales, 4 orientation)
	

image 2: bear.png

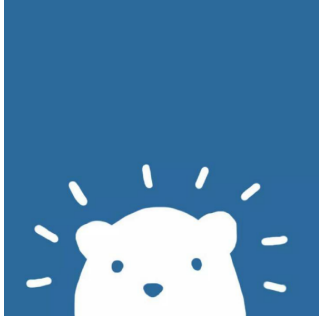
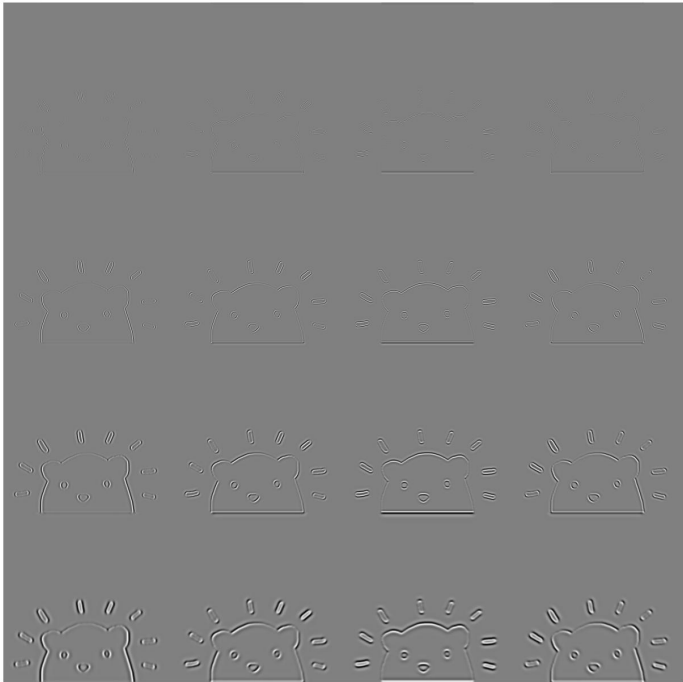
Original	Gabor convolve (4 scales, 4 orientation)
	

image 3: cat.jpg

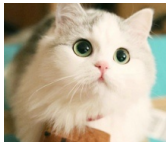
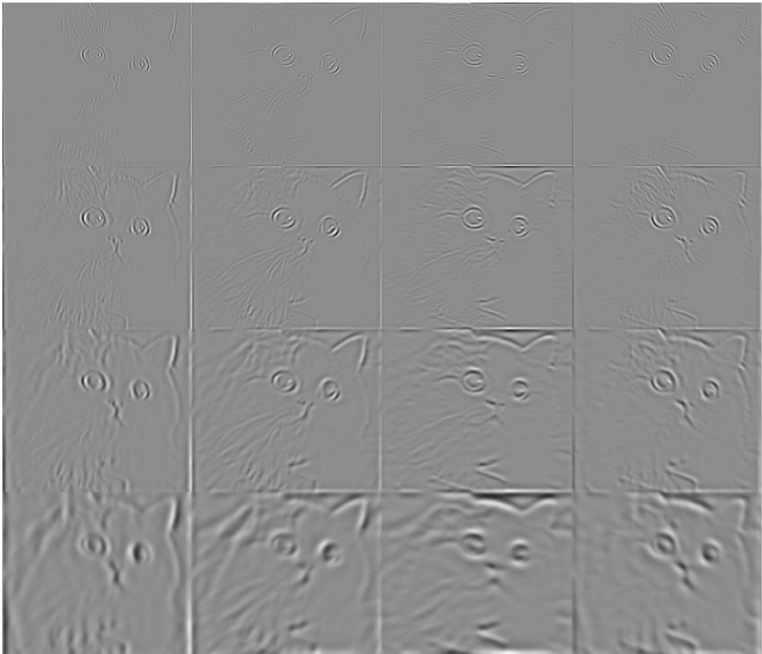
Original	Gabor convolve (4 scales, 4 orientation)
	

image 4: pig.jpg


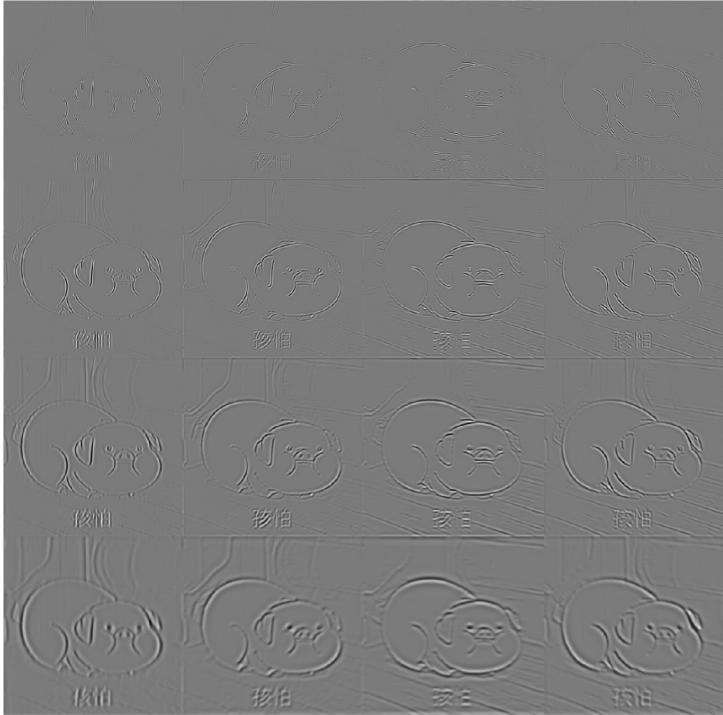

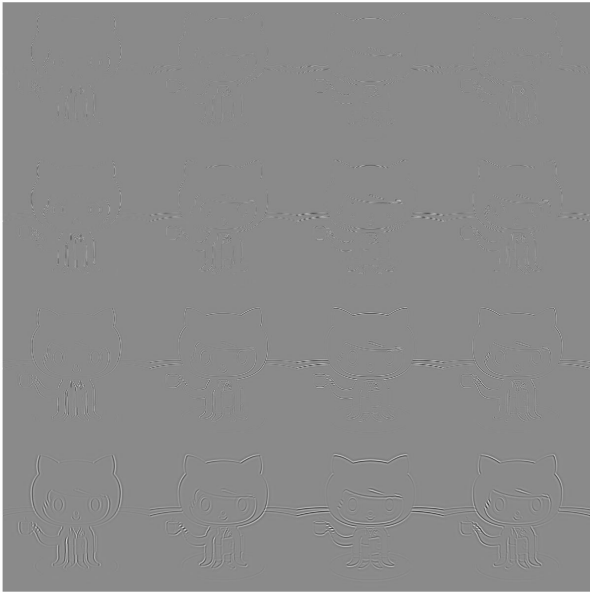
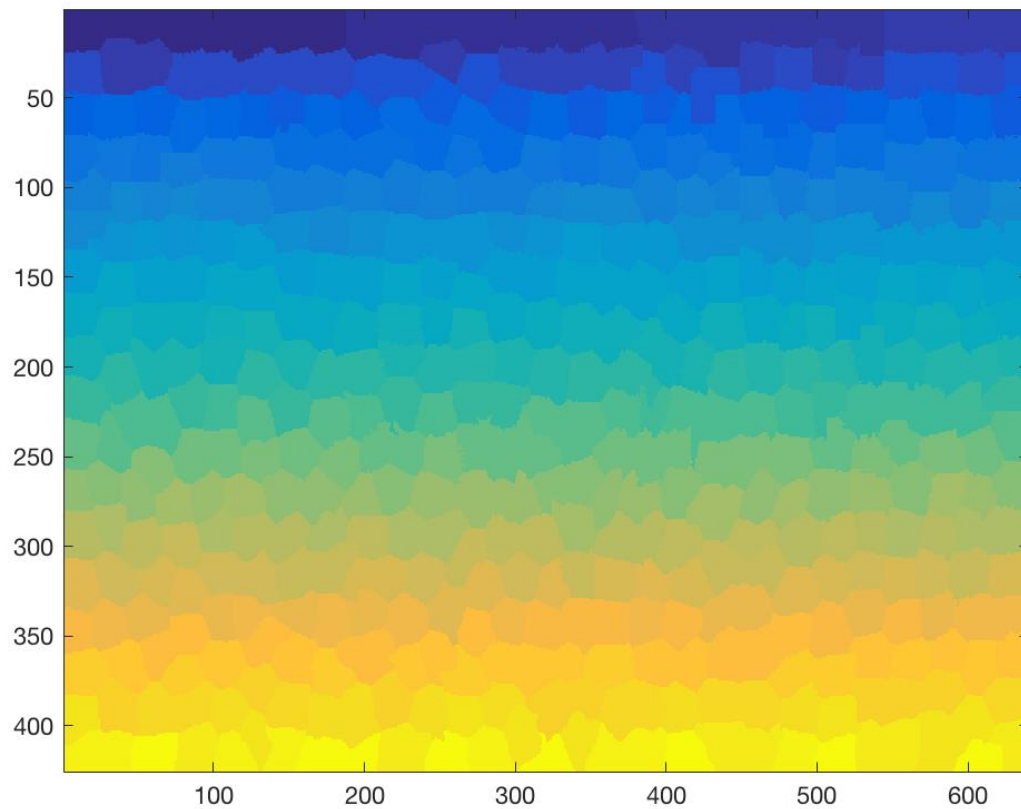
Original	Gabor convolve (4 scales, 4 orientation)
	

image5: codercat.jpg

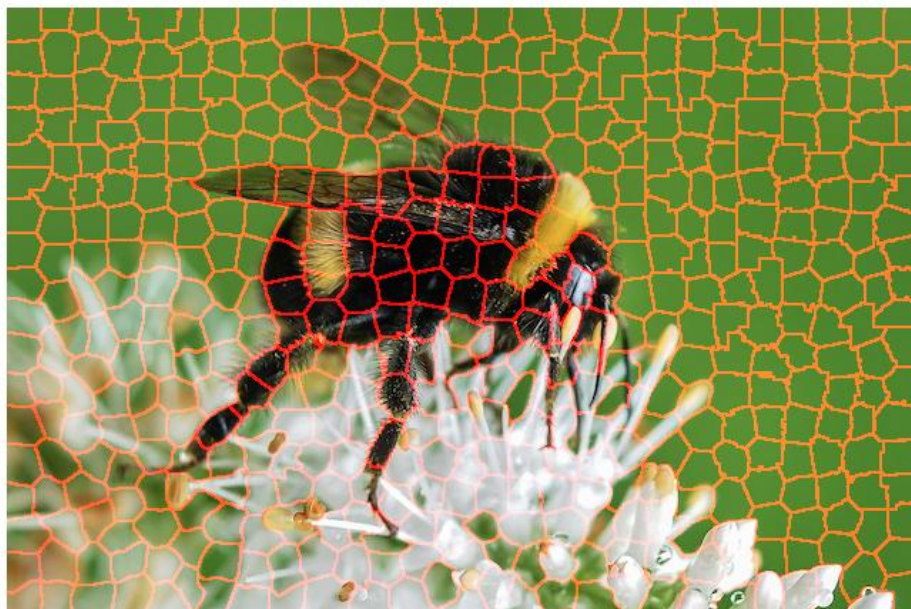
Original	Gabor convolve (4 scales, 4 orientation)
	

3. Clustering/segmentation results in Part 3 and Part 4. You must show the obtained regions by overlaying segment boundaries on the image as in Figure 1.

Segmentation Result, using the SLIC0 version:



Overlaying segment boundaries on the image as in Figure:



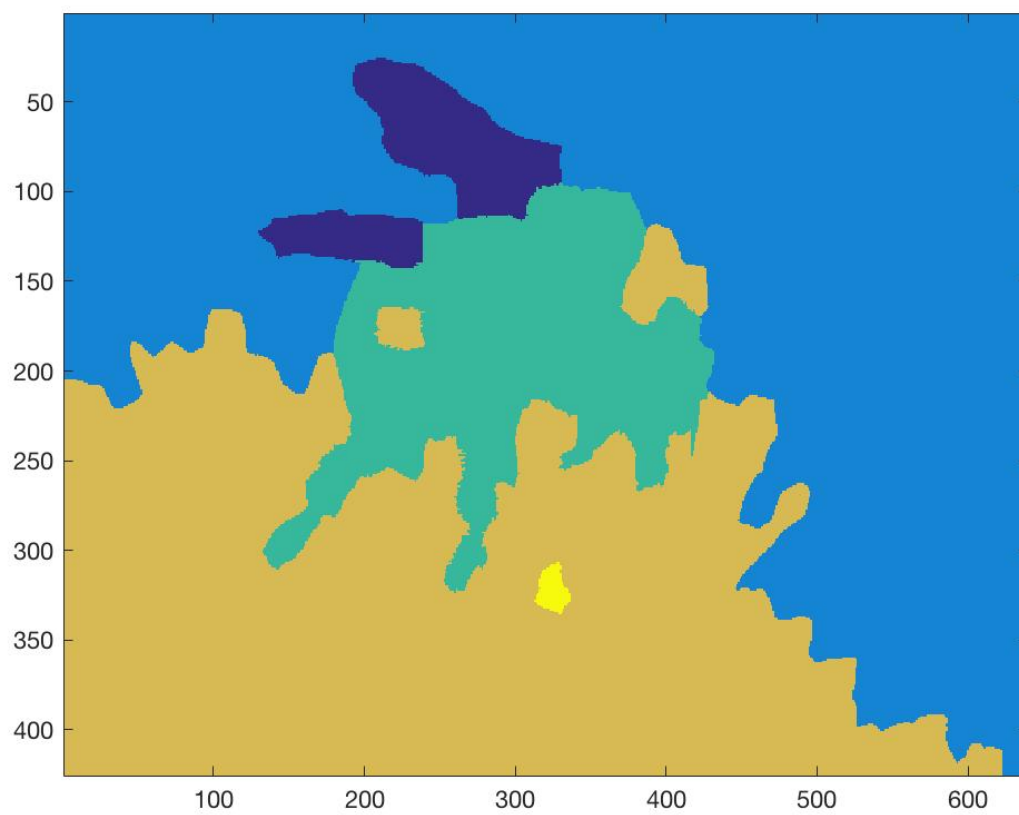
Clustering result based on superpixel:

Note: Here I used 2 methods to get the color feature vector and texture feature, then pick a better one to show. The detailed description of the two methods can be seen in point 5. I also put part of the code there.



Clustering label result based on superpixel:

Here I show the five labels result after clustering.

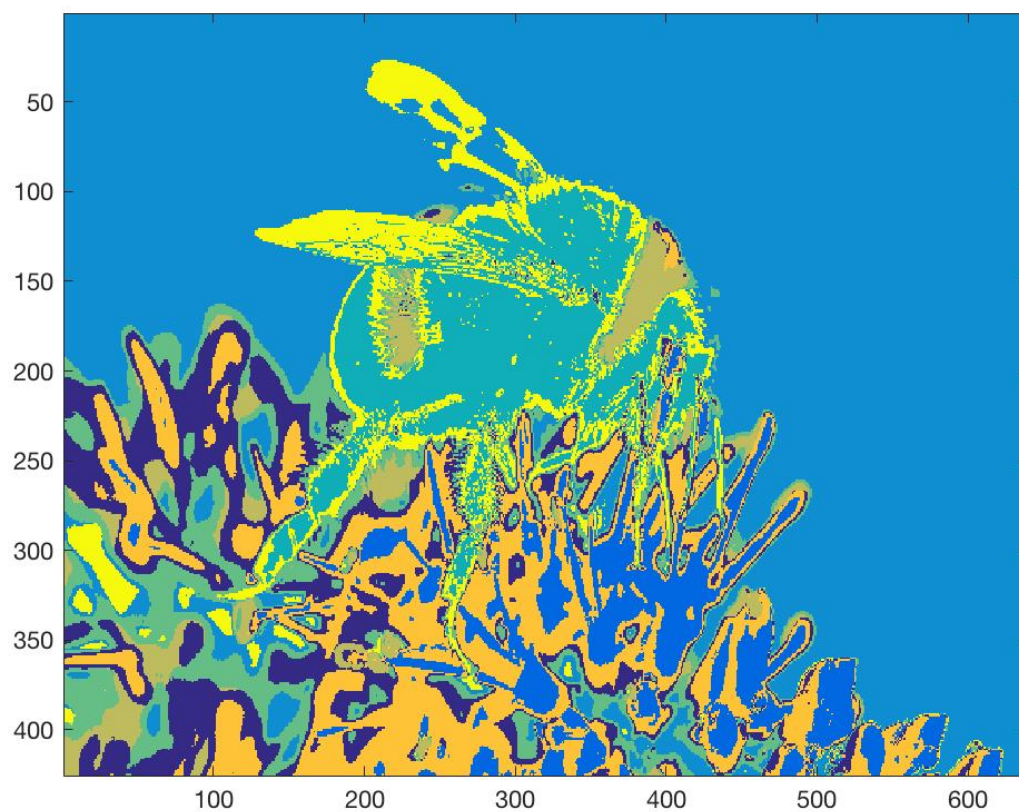


Clustering result based on single pixels:

Note, clustering based on single pixel only uses the color information.



Clustering label result based on single pixel:



4. Citation for any external code used.

- I downloaded the **SLICdemo.m** and **slicomex.c**, and compile the **slicomex.c** to get **slicomex.mexmaci64**.

- When I run the **gaborconvolve.m** function, I found that it used two another, so I also include them into the workspace.

(1) lowpassfilter.m

(2) filtergrid.m

5. Detailed discussion of the results with respect to different features, different superpixel extraction parameters, different clustering parameters, and clustering using superpixel based and pixel-based information.

- **Result with different features:**

Color features:

At first, I used a very simple methods:

Calculate the average values of 3 channel (RGB) for each superpixel and get a **3-D color feature vector** for each superpixel and normalize them.

Then, with the inspiration of **Similarity computing** method in **Selective Search** for Object Recognition, I reconstruct another color feature vector extract method.

The idea and theorem of this method:

Use L1-norm normalization to get 25 bins of the histogram for each color channel of the image, so that each superpixel can get a **30-D color feature vector** $C_i = c_i^1, \dots, c_i^n$, each element of the color vector is a histogram value of the interval. This is because we have divide 0~255 into 10 intervals and for each interval there are RGB 3 channels.

You can see it in *GetColorFeatureVector.m*:

```

% get the color feature vector of a label's area
function colorFeatureVector = GetColorFeatureVector(im, intervalNum)
[m,n,k] = size(im);
intervalLength = 256/intervalNum;
colorFeatureVector = zeros(k, intervalNum);
for i = 1:m
    for j = 1:n
        for q = 1:k
            temp= ceil(im(i,j,q)/intervalLength);
            if temp==0
                temp=1;
            end
            colorFeatureVector(q, temp) = colorFeatureVector(q, temp)+1;
        end
    end
end
s = sum(colorFeatureVector,2)';
for j = 1:k
    for i =1:intervalNum
        colorFeatureVector(j,i) =
double(colorFeatureVector(j,i))/double(s(j));
    end
end
colorFeatureVector = reshape(colorFeatureVector', 1, k*intervalNum);

```

Texture features:

For texture features, I have also tried 2 methods:

At first, I calculate the average grayscale value under each scale and orientation. Since I have set the parameters for **gaborconvolve.m** as below:

- nscale: Number of wavelet scales. Here we test 4 scales.
- norient: Number of filter orientations. Here we test 4 norient.

For each superpixel, I calculate the average grayscale value of it's correspond image after gaborconvolve. Therefore, there is a **16-D texture feature vector** for each superpixel.

However, when I show the result image only with the texture feature, the cluster result is not that satisfying. So I change the methods as below and get a **30-D texture feature vector** for each superpixel:

The idea of this method:

- (1) Choose 3 out of 16 result images of gaborconvolve, trying to contain different scales and orientations.
- (2) Make them overlap to get a more completable texture feature.
- (3) normalize
- (4) Dividing into 10 intervals, using histogram for statistics

Since I use 3 result images from gaborconvolve as 3 layers, for each layers, we divide the pixels into 10 groups. Therefore, the result texture feature vector is also **30-D**.

You can see it in *GetTextureFeatureVector.m*:

```
% use gabor filter to get the texture feature vector of a label's area
function textureFeatureVector = GetTextureFeatureVector(img)
[m, ~] = gaborconvolve(img, 4, 6, 3, 1.7, 0.65, 1.3, 0, 1);
[imgx, imgy, ~] = size(img);
featureMatric = cell2mat([m(2,2), m(3,4), m(4,6)]);
featureMatric = reshape(featureMatric, [imgx, imgy, 3]);
featureMatric = abs(featureMatric);
% normalize
[x,y,~] = size(featureMatric);
for k = 1:3
    mi = min(min(featureMatric(:,:,k)));
    ma = max(max(featureMatric(:,:,k)));
    for i = 1:x
        for j = 1:y
            featureMatric(i,j,k) = 255*featureMatric(i,j,k)/(ma-mi);
        end
    end
end





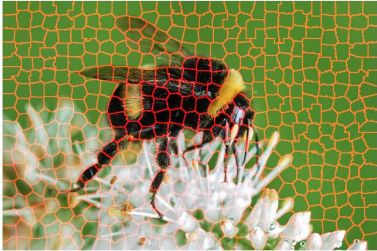



featureMatric = uint8(featureMatric);
textureFeatureVector = GetColorFeatureVector(featureMatric, 10);
```

- **Result with different superpixel extraction parameters:**

For changing the different superpixels parameters, the following statements is needed:

```
[labels, numlabels] = slicomex(img,100);
[labels, numlabels] = slicomex(img,300);
[labels, numlabels] = slicomex(img,500);
[labels, numlabels] = slicomex(img,700);
```

They slice the original picture into 100, 300, 500, 700 parts:

100		
300		
500		
700		

Analysis:

From the result of clustering based on superpixels, we can see that when the superpixels parameters (dividing parts) is low, the results after clustering are rough and have some error classification.

In general, with the growth of parts number, from 100 to 700, the cluster results become more accurate.

Therefore, we can come to the conclusion that the detailed information can be preserved after clustering with more slicing parts.

However, it doesn't mean the more parts the better. If divided into too many parts, it loses the lots of advantages of superpixel. For example, improving the complexity of image processing, and all other advantages discussed about the comparison between clustering based on superpixel and clustering based on pixel.

- **Different Clustering parameters:**

- For cluster based on superpixel, I use **hierarchical cluster** method:

```
Z = linkage(featureVector, 'average', 'euclidean');
T = cluster(Z, 'maxclust', num);
[x,y] = size(labels);
clusterLabels = zeros(x, y);
for i = 1:x
    for j = 1:y
        clusterLabels(i,j) = T(labels(i,j));
    end
end
img = ShowEdge(img, clusterLabels);
```

`Z = linkage(X)` returns a matrix Z that encodes a tree of hierarchical clusters of the rows of the real matrix X.

`Z = linkage(X, method, metric)` performs clustering using the distance measure metric to compute distances between the rows of X.

`method`: Algorithm for computing distance between clusters.

here, I choose 'average', it described as "Unweighted average distance (UPGMA)"

`metric`: Any distance metric that the `pdist` function accepts.





here, I choose 'euclidean', it described as "Euclidean distance (default)."

`T = cluster(Z, 'maxclust', n)` constructs a maximum of n clusters using the `distance` criterion. `cluster` finds the smallest height at which a horizontal cut through the tree leaves n or fewer clusters.

Since the choice of parameters "method", "metric", "distance" is related to the pixel distribution, I have tried many other choices and compare the result image after clustering, then choose the parameters setting "average", "euclidean", "maxclust", which shows the best result.

I have also uses k-means clustering method here. However, the result after clustering is not that satisfying. So I used hierarchical clustering at last.

Here are some of the comparison results from different methods and parameter settings:

<p>K-means</p>	
<p>Hierarchical Clustering Parameters:'average', 'euclidean'</p>	
<p>Hierarchical Clustering Parameters:'weighted', 'euclidean'</p>	
<p>Hierarchical Clustering Parameters:'weighted', 'correlation'</p>	

For cluster based on single pixel, I use **K-means cluster** method:


```

T = kmeans(colorVector, 5);
clusterLabelsBasedOnSinglePixel = zeros(x, y);
for i = 0:x-1
    for j = 1:y
        clusterLabelsBasedOnSinglePixel(i+1,j) = T(i*y+j);
    end
end
img = ShowEdge(img, clusterLabelsBasedOnSinglePixel);

```

Here I choose **K-means** because when dealing with clustering based on single-pixel, the computational complexity of **Hierarchical Clustering** is too high, so it is not applied to clustering based on single pixel.

`kmeans(X, k)` performs *k*-means clustering to partition the observations of the *n*-by-*p* data matrix *X* into *k* clusters, and returns an *n*-by-1 vector *idx* containing cluster indices of each observation. Rows of *X* correspond to points and columns correspond to variables.

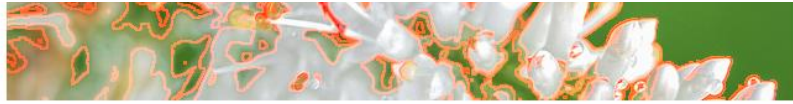
By default, `kmeans` uses the squared Euclidean distance measure and the *k*-means++ algorithm for cluster center initialization.

k=3



k=5




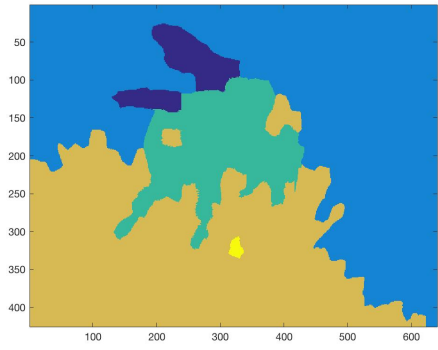

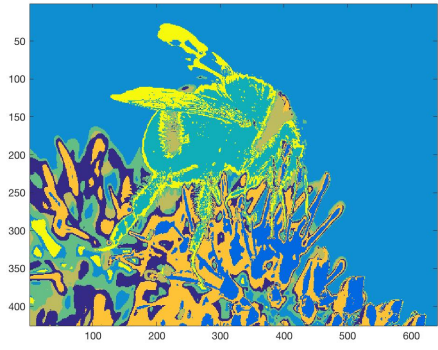


k=7



- **Comparison between Clustering using superpixel based and pixel based information:**

result constraction:

<p>superpixel based clustering</p>		
<p>single pixel based clustering</p>		

Analysis:

- The concept of superpixel is a pixel block with certain visual meaning, which is composed of neighboring pixels with similar texture, color and brightness. It uses the similarity between pixels to group pixels, uses a small amount of pixels to replace a large number of pixels to express the image features, greatly reduces the complexity of the image processing, so it is usually used as the preprocessing step of the segmentation algorithm.
- If the size of the image is large, the segmentation on the graph defined on the pixel grid is tricky, while the slic super-pixel reduces the complexity of the graph and makes the segmentation easy to handle.
- The segmentation based on single pixel can only be segmented according to color information, since there is no texture feature for single pixel. As long as there is a little color difference, it will be separated in two parts. Therefore, the image of the whole object is fragmented after clustering. We know that, the object extracted after image segmentation can be used in the fields of image semantic recognition, image searching and so on. So, segmentation based on single pixel is not convenient for computer to do further processing.
- On the contrary, segmentation based on super pixel combined color feature vector and texture feature vector, it is clustered based on superpixels, so the particle size is not that small. The whole object is preserved and easy to be got for further processing.

Reference:

- [1] <http://blog.csdn.net/surgewong/article/details/39316931>
- [2] http://blog.csdn.net/qq_16365849/article/details/50646679

