EE569 Digital Image Processing
Name: Shuo Wang
USC ID: 8749390300
Email: wang133@usc.edu
Date: Mar. 24, 2017

EE 569: Homework #3

Issued: 2/27/2017 Due: 11:59PM, 3/26/2017

Problem 1: Texture Analysis and Segmentation (30 %)

(a) Texture Classification (Basic: 10%)

Motivation

We should do two kinds of classification methods to tell the class of the texture A to F: unsupervised classification and classification.

Unsupervised classification is that we should find the centroids of all four clusters for the training data while we do not know which cluster the training data belongs to. That is, we have to form the cluster by identifying the difference among the values of data. The classification method we use is K-mean.
Then, the completed K-mean model can be applied to identify which class the test data point belongs to. If it is closest to one of the centroids representing one cluster, we can regard the point belongs to this cluster.

Supervised classification is that we have the idea about which training data point belongs to one of the clusters when we get the training images. We can regard the average of points in one cluster as the centroid of one cluster. Then, those centroids can be applied to identify which class the test data point belongs to. If it is closest to one of the centroids representing one cluster, we can regard the point belongs to this cluster.

Approach and Procedures

At first, we will discuss the unsupervised method.
For the unsupervised, I have two kinds of vectors with distinct dimensions to do the classification: the 25-D vector and the 3-D vector.

To get the 25-D feature vector for each image, we can do such steps:
1 Get the average pixel value for a training image. Then, we subtract pixel value in each points by the average value, which is the input matrix.
2 Use the Law filter to process input matrix above. The Law matrix has five kinds of value: L5, E5, S5, W5 and R5, which only represents only one direction. Therefore, for the two-direction matrix, we should have 25 kinds of combination situation, which means that there are 25 kinds of Laws filters. For example, we can get one of the filter as follow (in the horizontal direction, it is E5; in the vertical direction, it is W5):

$$W5^T E5 = \begin{bmatrix} -1 \\ 2 \\ 0 \\ -2 \\ 1 \end{bmatrix} [-1 \quad -2 \quad 0 \quad 2 \quad 1] = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ -2 & -4 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 4 & 0 & -4 & -2 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

3 Do the energy average for those feature vectors for all pixel points, which can get a 25-D vector

representing a specific image. The formula can be shown below (the MN is the size of the image):

$$\boldsymbol{R} = \sqrt{\frac{1}{MN} \sum_{for\ all\ pixel\ point(i,j)} \boldsymbol{r}_{(i,j)}{}^2}$$

To get the 3-D feature vector for each image, we can do such steps:

The first and second step is same as the one for the 25-D feature vector.

3 Having got the 25-D feature vector, we can use the PCA function in the MATLAB to calculate the PCA matrix which can transform the 25-D feature vector in the second step to 3-D feature vector. Since the input of K-mean function I constructed is 25-D feature vector, I have to do zero-padding for the 3-D feature vector.

4 Do the energy average for those feature vectors for all pixel points, which can get a 3-D vector representing a specific image. The formula can be shown below (the MN is the size of the image):

$$\boldsymbol{R} = \sqrt{\frac{1}{MN} \sum_{for\ all\ pixel\ point(i,j)} \boldsymbol{r}_{(i,j)}{}^2}$$

Having get the feature vectors for the training images, we should build the K-mean function. The K-mean should be implemented as follows:

1 Initialize the K points as the centroids of K clusters

2 Do the Euclidean distance between a point in the set and each centroids. If the distance is the smallest, we can regard the point belonging to this class.

3 for each class, do the average to update the centroid for each class

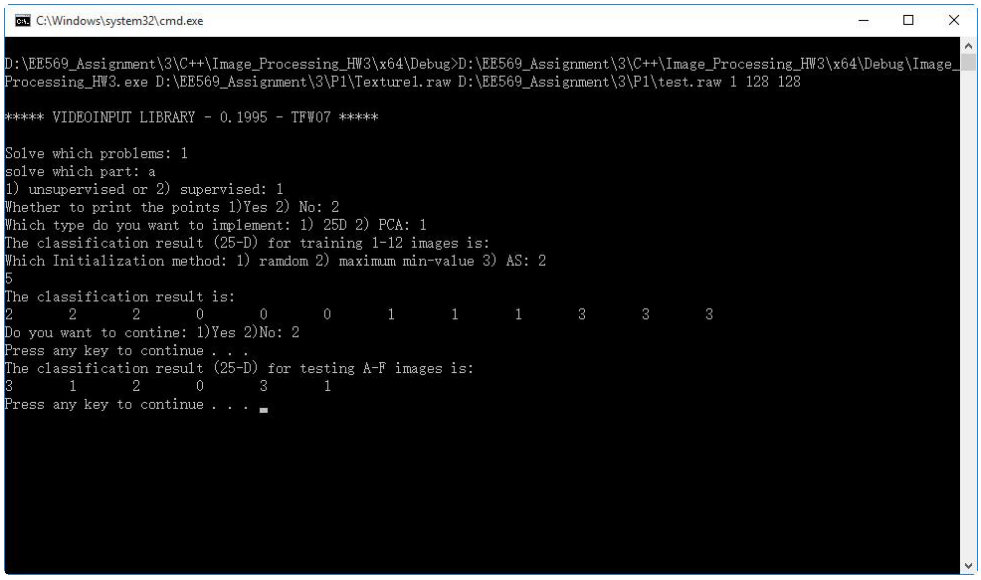
4 Do 2 and 3 steps until the centroids can change little.

Experimental Results

Unsupervised

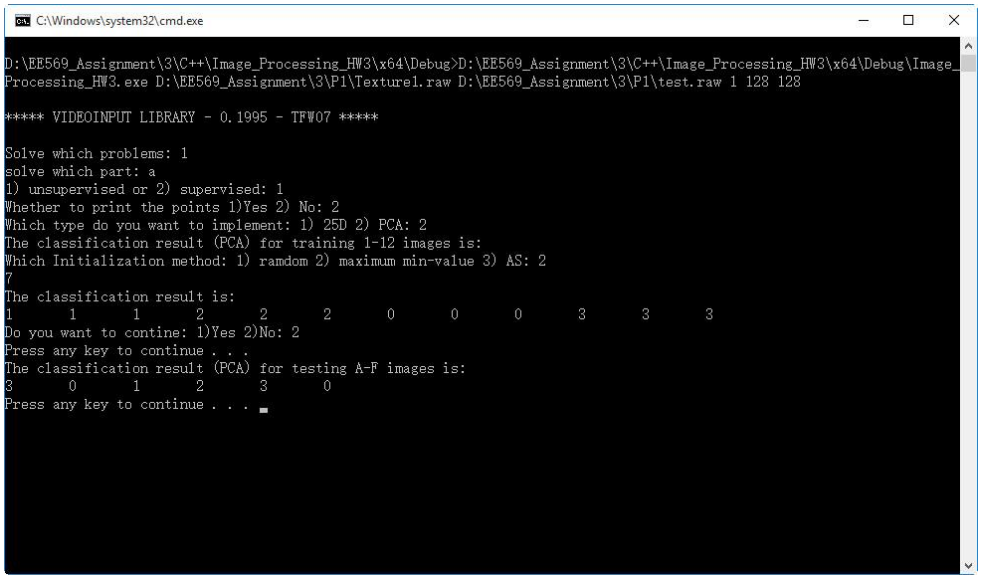
The result represents the class number for Image 1 to 12 (training) or A to F (Testing)

The best result of the classification for the training image and testing image by 25-D vector can be shown as follow:



The classification result of the testing and training by the 25-D vector

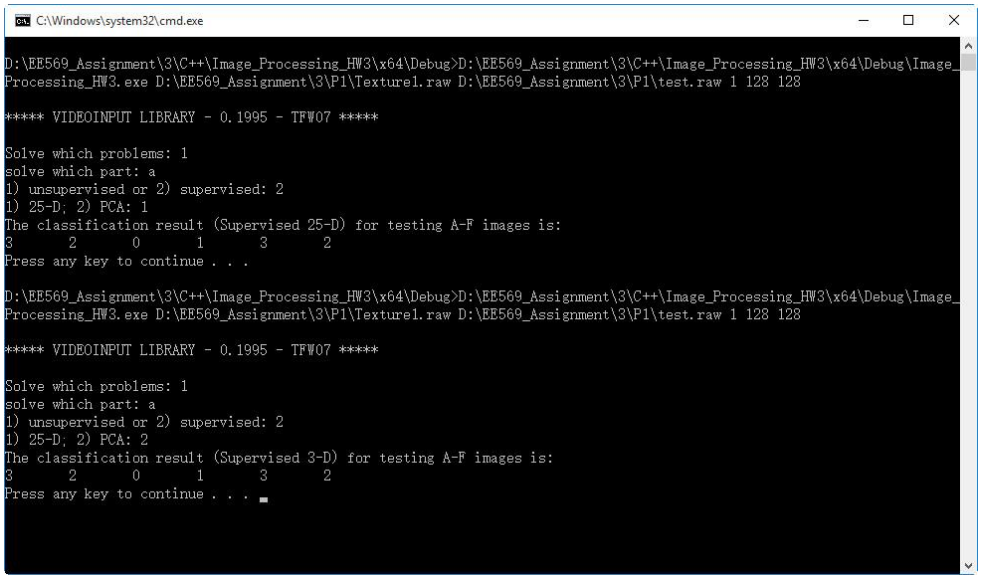
The best result of the classification for the training image and testing image by 3-D vector can be shown as follow:



The classification result of the testing and training by the 25-D vector

Supervised

The best result of the classification for the testing image by both 25-D and 3-D vectors can be shown as follow:



The classification result of the testing by the 25-D and 3-D vectors

Discussion

In this section, I will firstly discuss the power of the elements in the image. Then, I will discuss two issues that change the accuracy of the k-means significantly in the question: the k-means initialization method and the value of the first dimension. In addition, I will talk about the effect of the PCA.

At first, we can discuss the power of the elements in the vector. As a matter of fact, the law filter divides the frequency area into 25 parts. The value of the twelve 25-D vectors can be shown below:

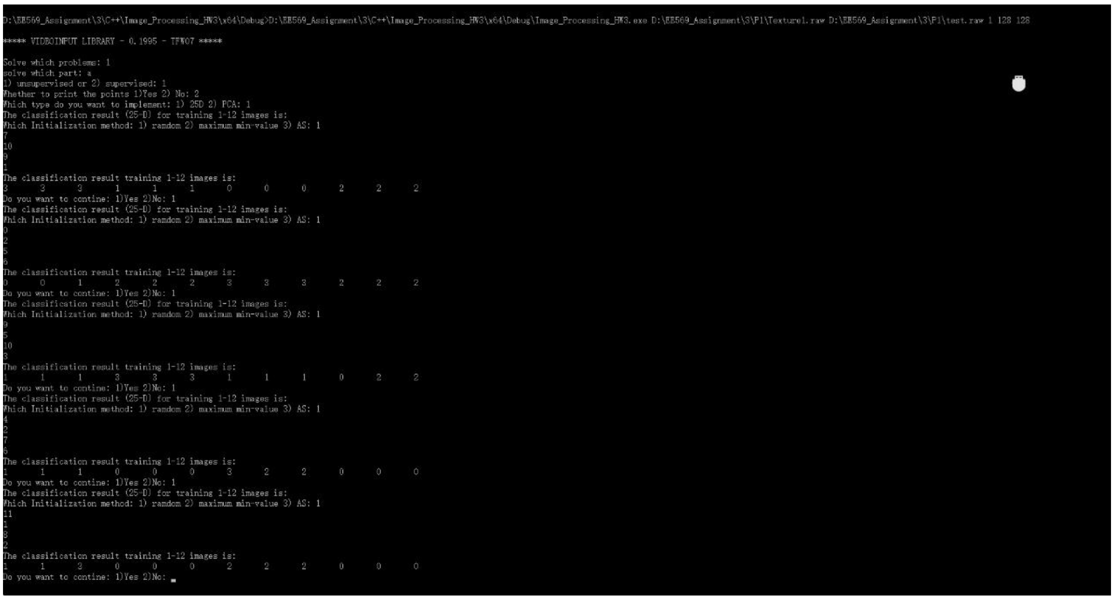
aS9

25x12 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	40.0923	37.1348	30.0886	13.6470	14.1218	16.2839	47.7129	47.7376	45.9965	21.0590	20.1637	21.2251
2	8.5077	8.6350	6.9574	3.9510	4.0658	4.4346	10.0994	10.5007	10.6306	4.8780	4.8851	4.7294
3	3.7348	4.3116	3.6868	2.3090	2.4095	2.4494	4.6637	4.6470	5.0044	2.3933	2.4075	2.3663
4	2.9054	3.6153	3.2199	2.1616	2.2618	2.1970	3.2241	3.1265	3.3346	1.8826	1.9047	1.9342
5	4.4043	5.6238	5.1277	3.6743	3.8469	3.6983	3.2200	3.0632	3.1744	2.5123	2.5943	2.6853
6	6.3111	6.2069	5.5544	4.0284	4.1005	4.3647	7.6128	8.3799	8.2123	3.1280	3.0293	3.1844
7	1.6147	1.6711	1.5053	1.2012	1.2424	1.2542	2.1349	2.3771	2.1812	0.9360	0.9194	0.9674
8	0.8404	0.9536	0.8499	0.7060	0.7354	0.7163	1.0941	1.1710	1.1043	0.5359	0.5247	0.5613
9	0.7309	0.8848	0.7915	0.6537	0.6929	0.6588	0.7972	0.8423	0.8043	0.4696	0.4628	0.4961
10	1.1895	1.5291	1.3845	1.1009	1.1814	1.1420	0.8379	0.8719	0.8298	0.6693	0.6860	0.7174
11	2.9646	3.1123	2.8094	2.4065	2.4360	2.3568	3.1526	3.3617	3.7656	1.3234	1.3227	1.3760
12	0.8220	0.9038	0.8341	0.7263	0.7502	0.7103	0.9514	1.0431	1.0247	0.4209	0.4179	0.4430
13	0.4572	0.5319	0.4868	0.4246	0.4428	0.4185	0.5076	0.5565	0.5290	0.2578	0.2523	0.2729
14	0.4090	0.5025	0.4548	0.3868	0.4139	0.3865	0.3837	0.4208	0.3987	0.2378	0.2366	0.2527
15	0.6715	0.8870	0.8013	0.6429	0.7024	0.6664	0.4186	0.4589	0.4298	0.3572	0.3768	0.3822
16	2.4273	2.7153	2.4916	2.2422	2.2670	2.0695	2.0521	2.2126	2.5782	0.9882	0.9996	1.0357
17	0.7082	0.8299	0.7677	0.6847	0.7106	0.6449	0.6298	0.7034	0.7255	0.3163	0.3179	0.3374
18	0.4065	0.4992	0.4562	0.3983	0.4203	0.3846	0.3420	0.3899	0.3797	0.1972	0.1963	0.2126
19	0.3695	0.4761	0.4257	0.3610	0.3901	0.3551	0.2664	0.3036	0.2887	0.1875	0.1906	0.2021
20	0.6223	0.8422	0.7465	0.6047	0.6688	0.6196	0.3115	0.3581	0.3254	0.3031	0.3249	0.3238
21	3.6056	4.3452	4.0096	3.5301	3.5763	3.1824	1.9643	2.2269	2.5289	1.2839	1.2912	1.3403
22	1.0698	1.3628	1.2277	1.1011	1.1401	1.0083	0.6131	0.7127	0.7405	0.4116	0.4170	0.4424
23	0.6328	0.8279	0.7372	0.6545	0.6917	0.6109	0.3389	0.4053	0.3985	0.2622	0.2647	0.2880
24	0.5980	0.8117	0.7130	0.6145	0.6578	0.5845	0.2801	0.3369	0.3111	0.2648	0.2708	0.2896
25	1.0875	1.5266	1.3311	1.1017	1.1949	1.0907	0.4061	0.4987	0.4071	0.4891	0.5148	0.5188

From the picture, we can find that the first element, which is the lowest frequency in both the horizontal and vertical direction, has the strongest power; while the 24^{th} element, which is the high frequency in in both the horizontal and vertical direction, has the lowest power. The reason is that the Law's filter represents different frequency interval and power in the image will concentrate on the low frequency area which represents the first dimension, while the high frequency pixel is really rare, which represents the 24^{th} dimension.

And then, I did not care about the initialization method. However, when I used the random number method (That is, I just choose the K (cluster number) points in the all sample points) to initialize the K-mean. However, I get the result as follow (take the 25-D vector as example):



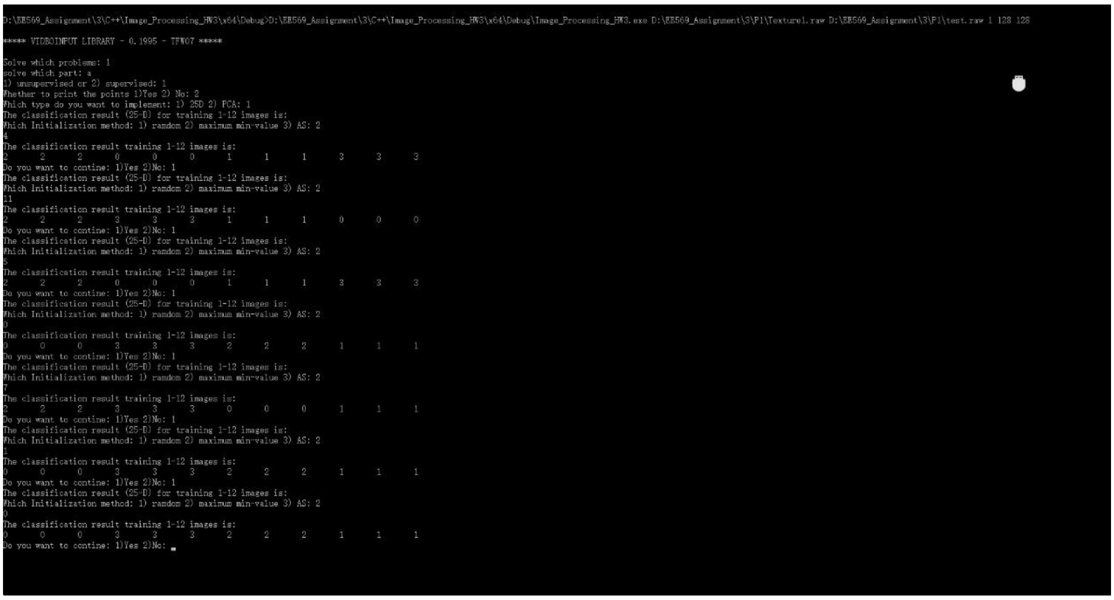
The classification result of the training by the 25-D vector (random number)

Compared with the original image (1-3 images are one group; 4-6 images are one group; 7-9 images

are one group; 10-12 images are one group), we can find that for some choices, the result of classification is incorrect if the initialization points are different. As a result, we can use another method to initialize the points: maximum minimum-value solution.

For the maximum minimum-value solution, we can choose the first initial centroid randomly. Then, for each points which has not been chosen, we should find the minimum distance between this point and the chosen initialized centroids. Then, for those unselected points and those minimum distances, we will decide the points with the maximum minimum-distance as the next centroids. We will repeat the procedure until we get all K centroids for K clusters.

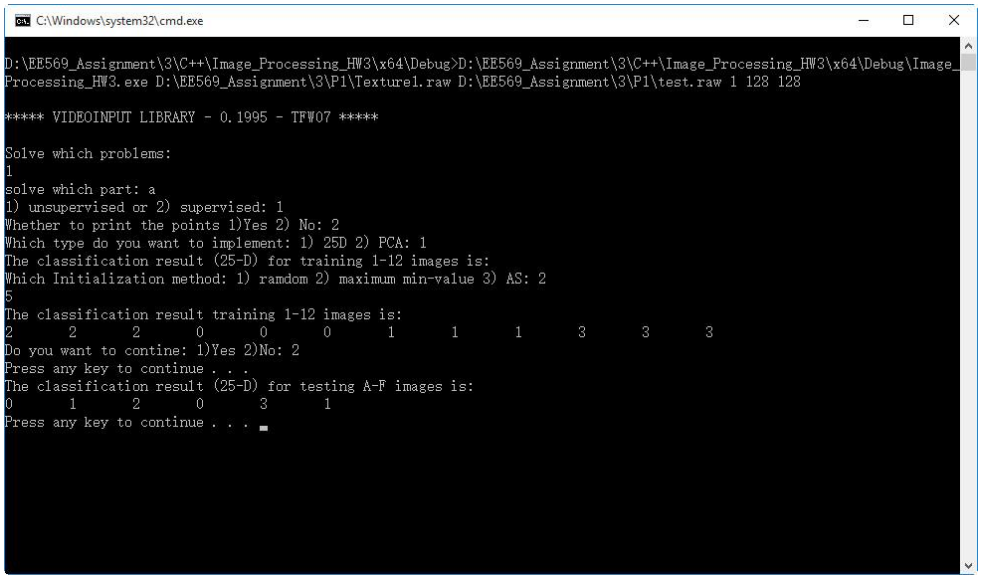
The result can be shown as follow (do not change other steps and only change the initialization method):



The classification result of the training by the 25-D vector (maximum minimum-value)

From the result, we can find that the new initialization method has a much better performance than the random value initialization method.

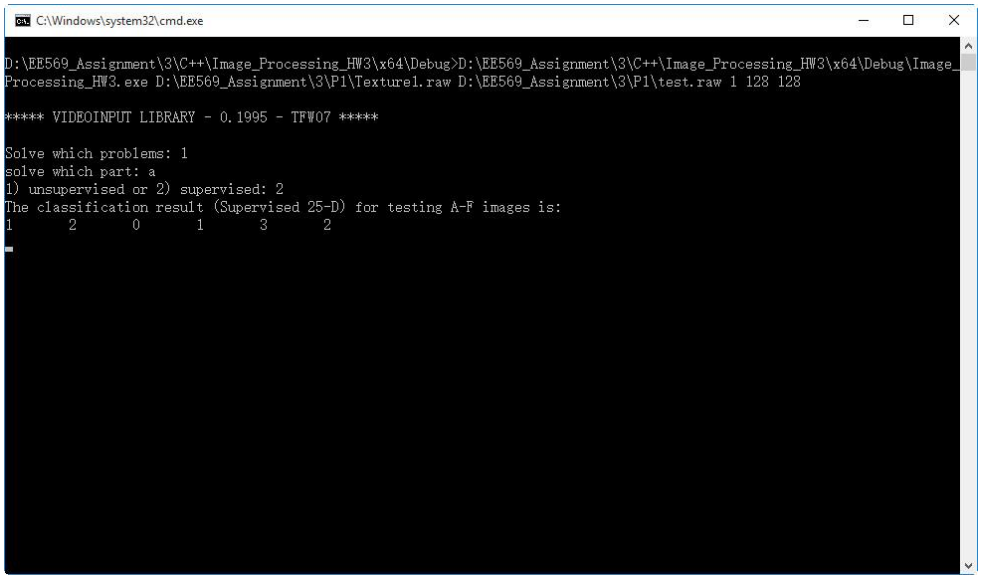
For the first-dimension value, I have some issues to think about. At first, I just use all the dimensions in the vector to do the classification. However, I found that the result is not good especially for testing. One of the situation can be shown as follow:



The classification result of the training and testing by all elements in the 25-D vector

Compared with the original test images (1 and 5 are same, 2 and 6 are same, 3 is in one class, 4 is in one class), we can find that there is one element whose classification result is wrong even though the training result is good.

Even we use the supervised method, we can get the similar results:



The classification result of the testing by all elements in the 25-D vector (Supervise)

Therefore, I began to consider if the data has some problems. Then, I searched the result of the 25-D vector data and get the result as follow:

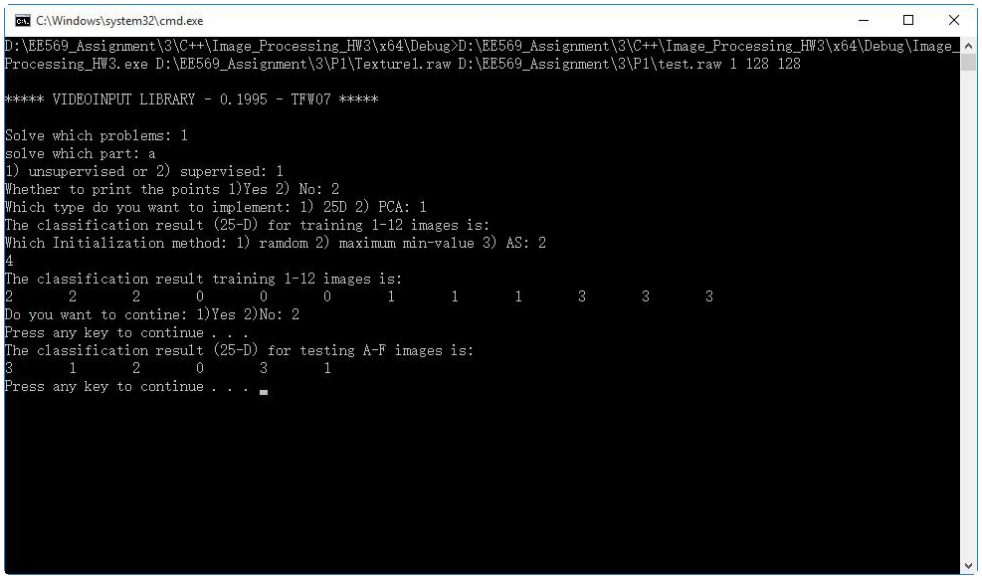
coeff | coeff1 | A1 | a59

25x12 double

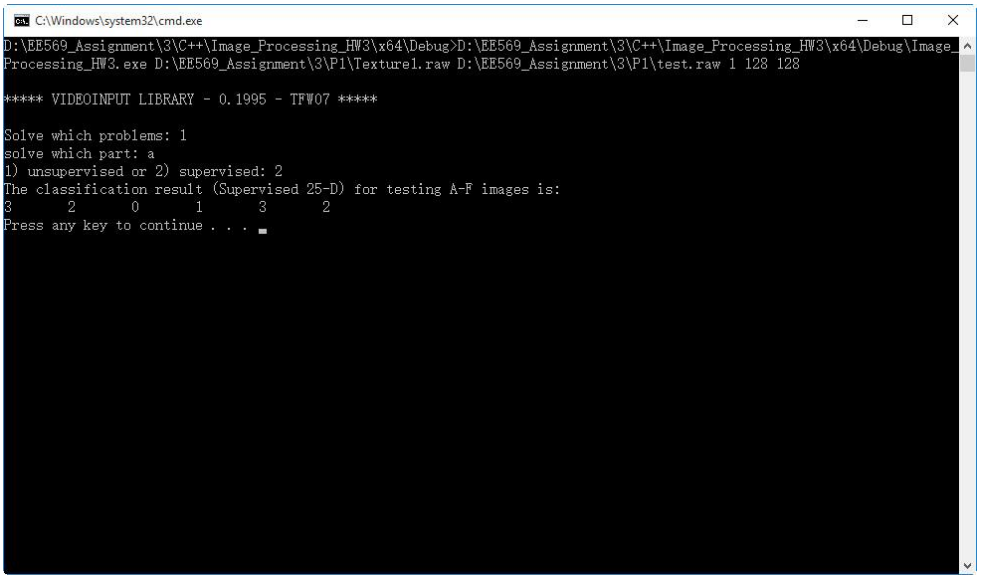
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	40.0923	37.1348	30.0886	13.6470	14.1218	16.2839	47.7129	47.7376	45.9965	21.0590	20.1637	21.2251	
2	8.5077	8.6350	6.9574	3.9510	4.0658	4.4346	10.0994	10.5007	10.6306	4.8780	4.8851	4.7294	
3	3.7348	4.3116	3.6868	2.3090	2.4095	2.4494	4.6637	4.6470	5.0044	2.3933	2.4075	2.3663	
4	2.9054	3.6153	3.2199	2.1616	2.2618	2.1970	3.2241	3.1265	3.3346	1.8826	1.9047	1.9342	
5	4.4043	5.6238	5.1277	3.6743	3.8469	3.6983	3.2200	3.0632	3.1744	2.5123	2.5943	2.6853	
6	6.3111	6.2069	5.5544	4.0284	4.1005	4.3647	7.6128	8.3799	8.2123	3.1280	3.0293	3.1844	
7	1.6147	1.6711	1.5053	1.2012	1.2424	1.2542	2.1349	2.3771	2.1812	0.9360	0.9194	0.9674	
8	0.8404	0.9536	0.8499	0.7060	0.7354	0.7163	1.0941	1.1710	1.1043	0.5359	0.5247	0.5613	
9	0.7309	0.8848	0.7915	0.6537	0.6929	0.6588	0.7972	0.8423	0.8043	0.4696	0.4628	0.4961	
10	1.1895	1.5291	1.3845	1.1009	1.1814	1.1420	0.8379	0.8719	0.8298	0.6693	0.6860	0.7174	
11	2.9646	3.1123	2.8094	2.4065	2.4360	2.3568	3.1526	3.3617	3.7656	1.3234	1.3227	1.3760	
12	0.8220	0.9038	0.8341	0.7263	0.7502	0.7103	0.9514	1.0431	1.0247	0.4209	0.4179	0.4430	
13	0.4572	0.5319	0.4868	0.4246	0.4428	0.4185	0.5076	0.5565	0.5290	0.2578	0.2523	0.2729	
14	0.4090	0.5025	0.4548	0.3868	0.4139	0.3865	0.3837	0.4208	0.3987	0.2378	0.2366	0.2527	
15	0.6715	0.8870	0.8013	0.6429	0.7024	0.6664	0.4186	0.4589	0.4298	0.3572	0.3768	0.3822	
16	2.4273	2.7153	2.4916	2.2422	2.2670	2.0695	2.0521	2.2126	2.5782	0.9882	0.9996	1.0357	
17	0.7082	0.8299	0.7677	0.6847	0.7106	0.6449	0.6298	0.7034	0.7255	0.3163	0.3179	0.3374	
18	0.4065	0.4992	0.4562	0.3983	0.4203	0.3846	0.3420	0.3899	0.3797	0.1972	0.1963	0.2126	
19	0.3695	0.4761	0.4257	0.3610	0.3901	0.3551	0.2664	0.3036	0.2887	0.1875	0.1906	0.2021	
20	0.6223	0.8422	0.7465	0.6047	0.6688	0.6196	0.3115	0.3581	0.3254	0.3031	0.3249	0.3238	
21	3.6056	4.3452	4.0096	3.5301	3.5763	3.1824	1.9643	2.2269	2.5289	1.2839	1.2912	1.3403	
22	1.0698	1.3628	1.2277	1.1011	1.1401	1.0083	0.6131	0.7127	0.7405	0.4116	0.4170	0.4424	
23	0.6328	0.8279	0.7372	0.6545	0.6917	0.6109	0.3389	0.4053	0.3985	0.2622	0.2647	0.2880	

The 25-D vector for 12 training images

From the data, we can find that the power of first dimension is huge, which means that it will mainly decide the cluster. However, we can find that the 1-3 is really close to the 7-9 while they belong to different clusters. Therefore, we can get the conclusion that this dimension message seems to be useless. As a result, I began to think that we do not need to consider the first dimension and use the rest dimension elements to do the classification. The result can be shown below (just delete the first element in the vector) (the first is the unsupervised result and the second is the supervised result):

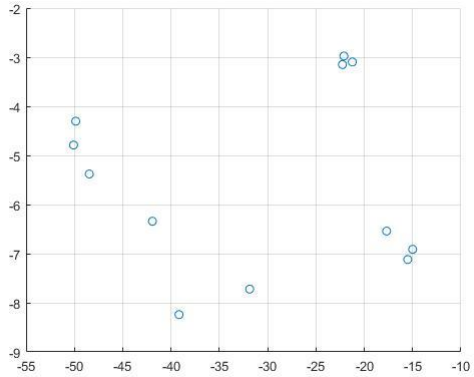


The classification result of the training and testing by 2nd-25th elements in the 25-D vector



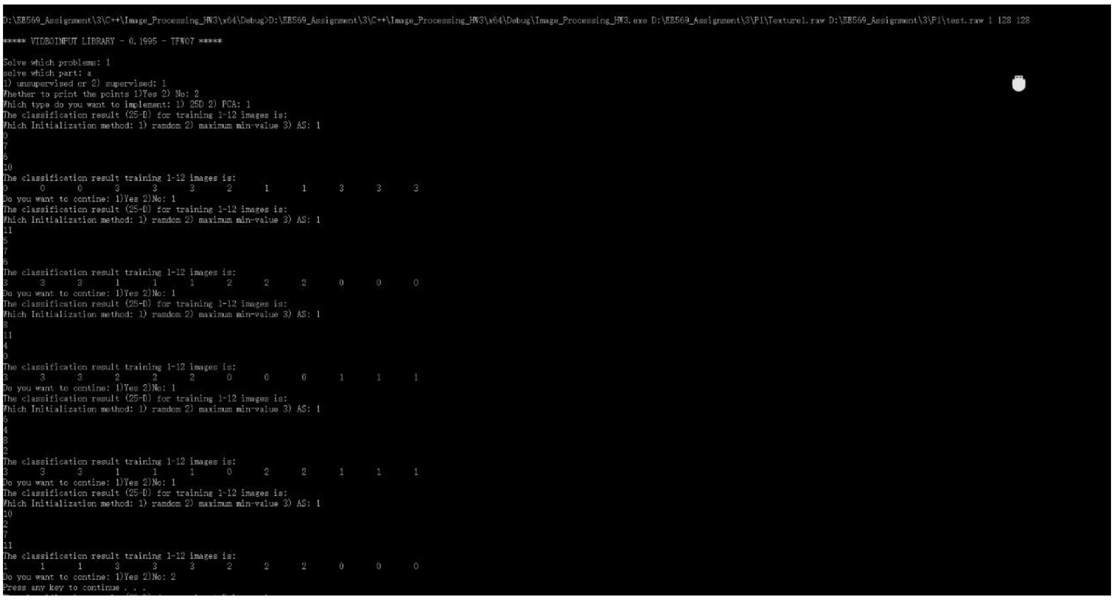
The classification result of the testing by 2nd-25th elements in the 25-D vector (Supervise)

Compared with the results by different elements, we can get that deleting the first dimension can get a good performance for the testing image.

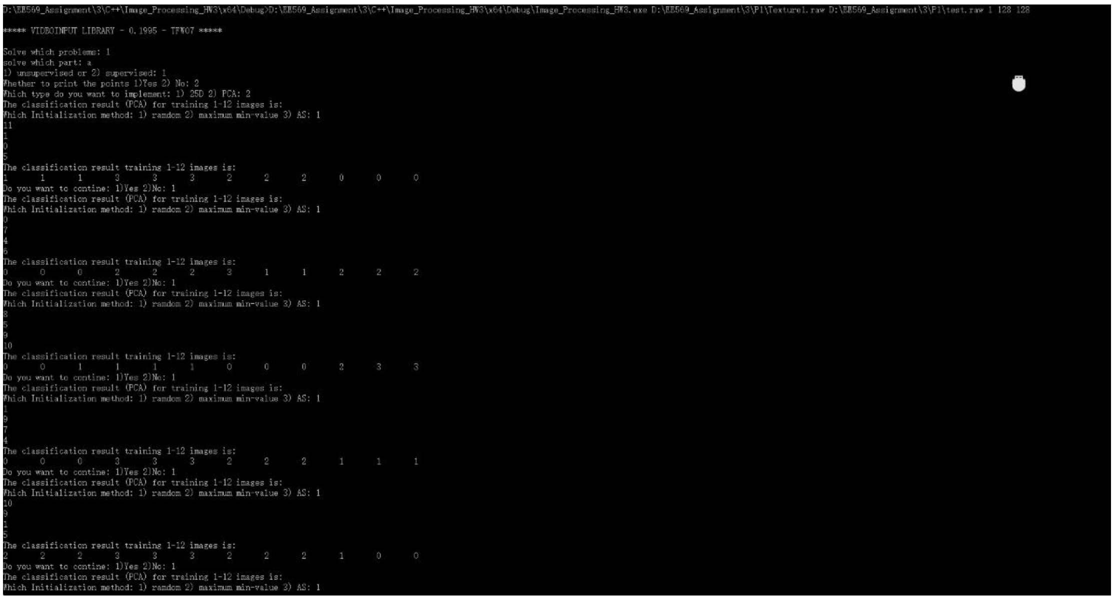


Then, we can consider the effects of the PCA. By the PCA, we can decrease the dimension of the vector from 25 to 3. We can consider the accuracy of classification between the 25-D and the 3-D. The plot of the 12 images represented by 3-D can be shown as follow:

The result can be shown as follow (in this question, I use the random number solution to get the result):



The classification result of the testing by 2nd-25th elements in the 25-D vector



The classification result of the testing by 2nd-25th elements in the PCA vector

From the result, we can get that the 25-D result will have at most 5 errors, while the PCA result has also at most 5 errors, which means that the accuracy of the classification cannot be decreased when the number of dimension decreased.

Also, we can consider the speed of the k-mean when using the 25-D and the PCA vectors. From the experiment, we can find that the running time of k-mean by 25-D is around 0.4 second while the running time of k-mean by PCA is around 0.35 seconds. As a result, we can find that the PCA vector can make the k-mean a little faster than the 25-D vector.

(b) Texture Segmentation (Basic: 10%)

Motivation

Use the K-mean algorithm to classify the image into several parts according to its texture. Then, use the result centroids obtained from K-means to classify the texture for each pixels. If the pixel

belongs to a specific texture, we can set the pixel into a color representing the texture, which can help us to classify the objects.

Approach and Procedures

At first, we will convert the color image into gray image by the formula below (RGB is the pixel value of R channel, G channel and B channel):

$$Gray = 0.2989 \times \mathrm{R} \ + \ 0.5870 \times \mathrm{G} \ + \ 0.1140 \times \mathrm{B}$$

Then, we will get the average pixel value for the image that we use to do segmentation and subtract pixel value in each points by the average value, which is the input matrix.

And then, we will use the Law filter to process input matrix above. The Law matrix has five kinds of value: L5, E5, S5, W5 and R5, which only represents only one direction. Therefore, for the two-direction matrix, we should have 25 kinds of combination situation, which means that there are 25 kinds of Laws filters. For example, we can get one of the filter as follow (in the horizontal direction, it is E5; in the vertical direction, it is W5):

$$W5^T E5 = \begin{bmatrix} -1 \\ 2 \\ 0 \\ -2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ -2 & -4 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 4 & 0 & -4 & -2 \\ -1 & -2 & 0 & 2 & 1 \end{bmatrix}$$

Having got the 25-D vectors for each pixel, we should use the window approach to decrease the fluctuations in the image. For each pixel, we should consider the neighbor pixel values (the neighbor pixels can be located at a 15x15 or 13x13 average window whose center is the pixel we want to solve).

Then, we should calculate the energy average to get the center pixel value, which can be shown below (N is the size of the average window):

$$\boldsymbol{R} = \sqrt{\frac{1}{N \times N} \sum_{for\ all\ pixel\ point(i,j)\ in\ the\ average\ window} \boldsymbol{r}_{(i,j)}{}^2}$$

Finally, I began to do the normalization to change the value in each vectors. We can get the feature vectors we want.

Having get the feature vectors for the image, we should build the K-mean function. The K-mean should be implemented as follows:

1 Initialize the K points as the centroids of K clusters

2 Do the Euclidean distance between a point in the set and each centroids. If the distance is the smallest, we can regard the point belonging to this class.

3 for each class, do the average to update the centroid for each class

4 Do 2 and 3 steps until the centroids can change little.

Having done the K-means and got the centroids for each class, we can use different color to represent different textures (classes). As a result, we will get the segmentation picture we want.

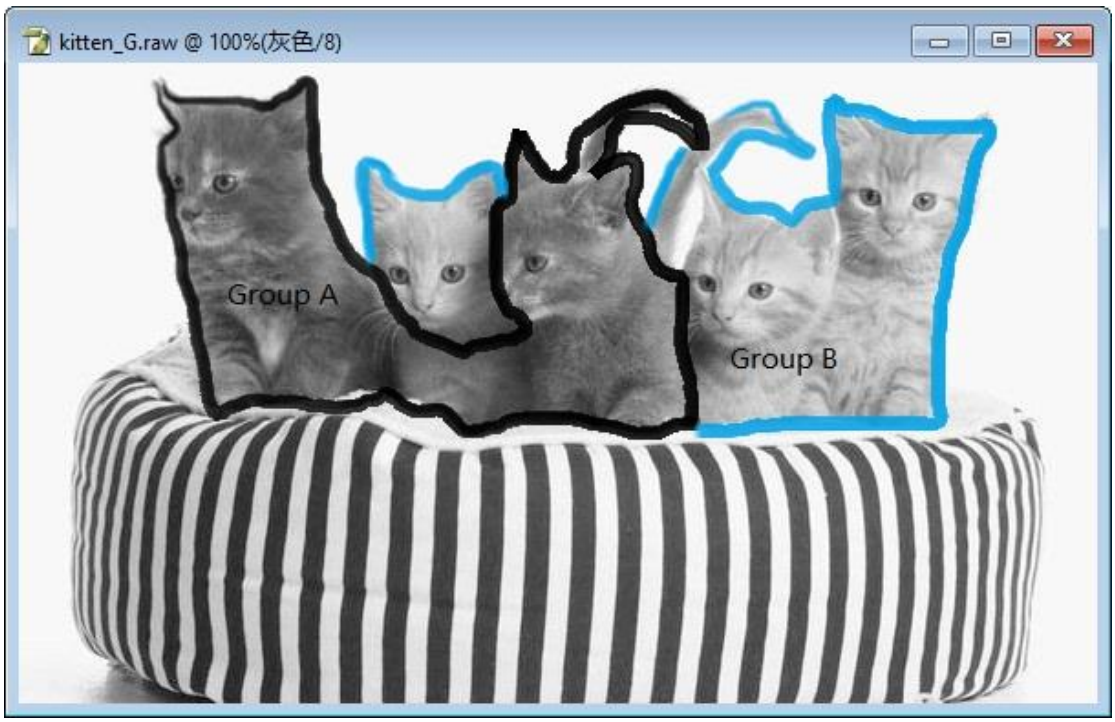
Experimental Results

When converting the color image to gray image, we can do the analysis about which will be segmented. At first, we will show the gray image:

The gray picture of Kitten.raw

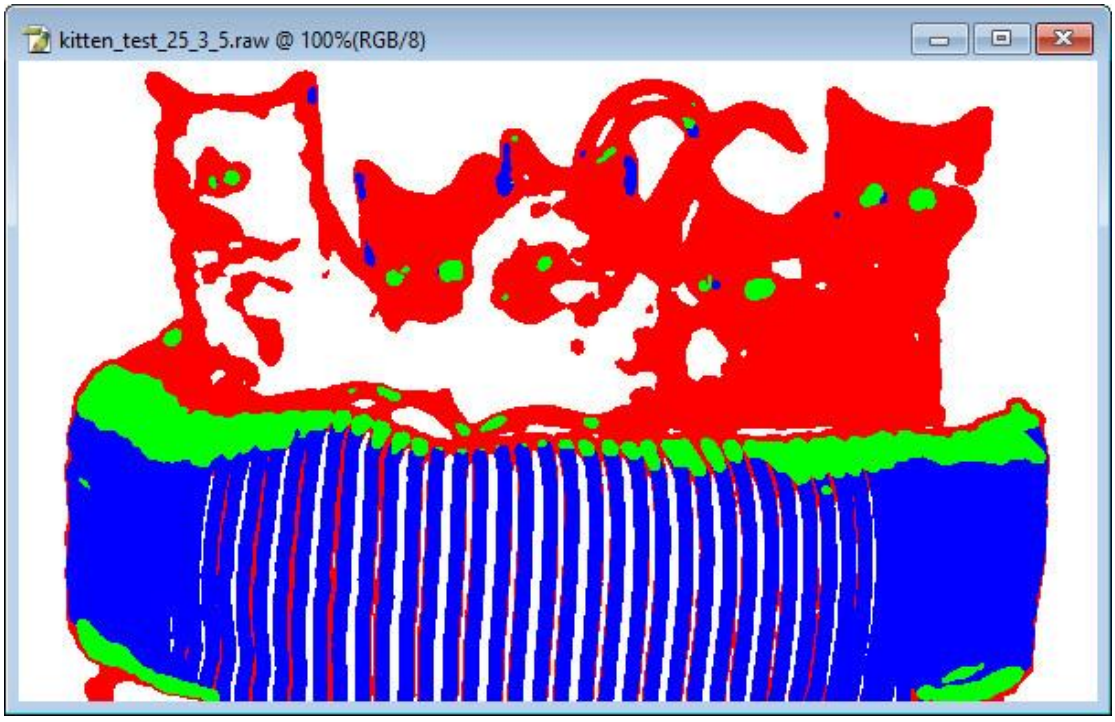
From the gray image, we can easily divide the right two cats into one group (group B) and the first left and the middle into another group (group A). For the second left cat, we can evaluate the deviation between the groups.

According to the observation, we can find that the pixel value of group B is around 200; the pixel value of group A is around 80. In addition, we can gain that the body of second left cat is around 85 and the head of second left cat is around 200. Therefore, we can put the head of cat into group A and the body of cat into group B. The division prediction can be shown below:



The division of the cat in the kitten.raw

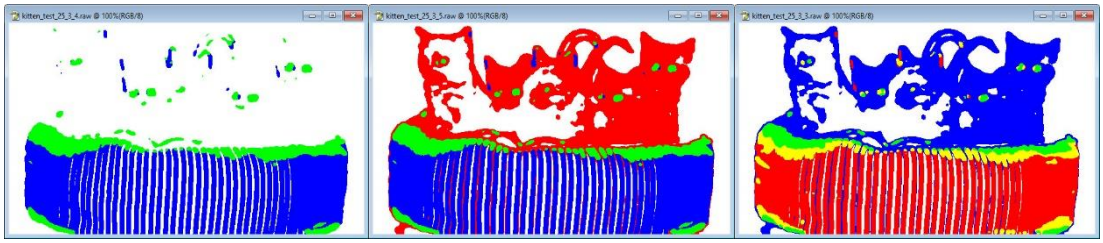
The best result of the segmentation can be shown below (I use 4 clusters to do the segmentation):



Use the K = 4 to do the classification

Discussion

Since we have no idea about the number of the clusters in this section, we have to choose the least number of the clusters to represent the segmentation as complete as possible. As a result, I try K = 3, 4 and 5 to get the segmentation, which can be shown as follow:



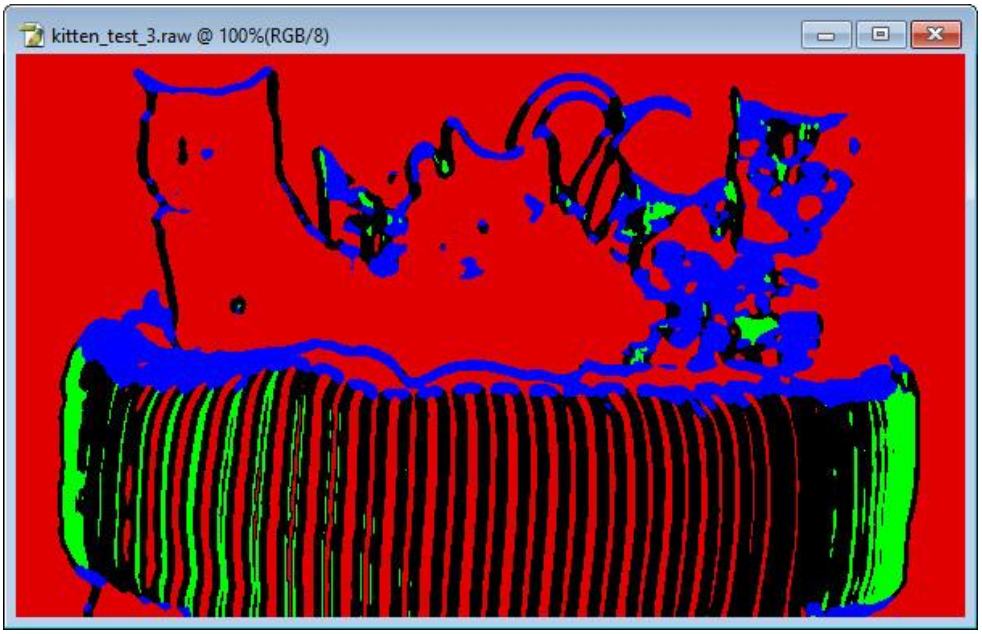
Use the K = 3 (Left), 4 (Middle) and 5 (Right) to do the classification

Then, we can find that we cannot get the shape of cat if we do the classification less than or equal to 3 because the texture of cat is much closer to the background than the mat. As a result, we should use the cluster more than four.

When searching the vectors, we can find that the first dimension is much larger than any other dimensions just like the Problem 1 (a). In the question, we are asked to use the energy of the first dimension to normalize the other dimension, which can be shown as follow:

$$P_{in\ (i,j)\ out}(s) = P_{in\ (i,j)\ input}(s) \times P_{in\ (i,j)\ input}(s) \div P_{in\ (i,j)\ input}(1)$$

However, when I use the four cluster to do the k-mean classification, the result can be shown as follow:

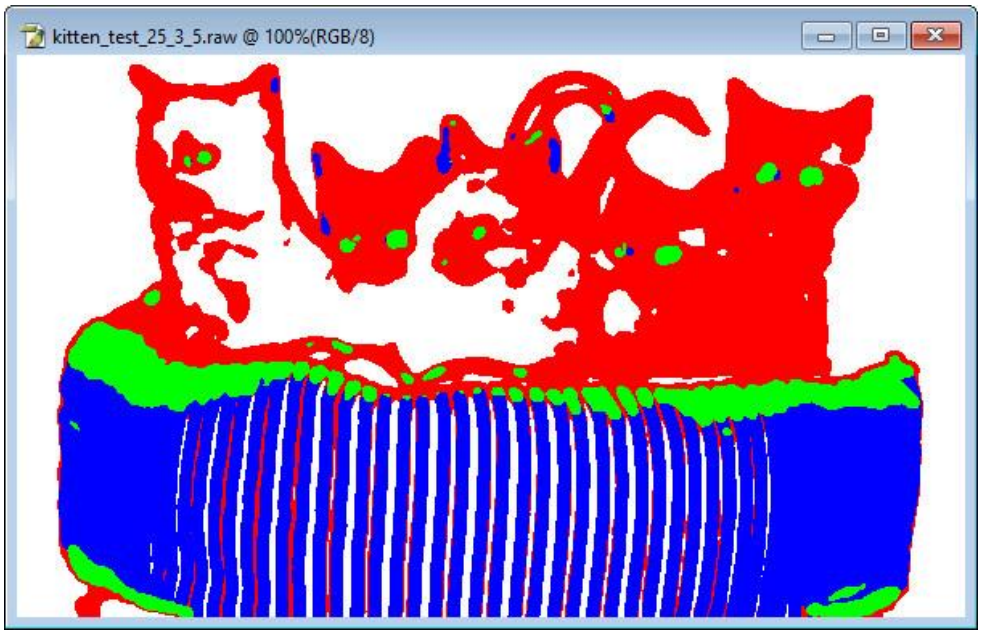


We can find that the second and forth cat are not classified clearly even though the first and second cat can be identified clearly. Therefore, we have to propose another solution to modify the number. Having utilized some methods, I use normalization instead of deleting this dimension to process the vector. The procedure is as follow:

1. For each dimension, find the maximum value among all pictures.
2. For a point in each dimension, we do the calculation as follow:

$$P_{in\ s\ dimension\ out}(i,j) = P_{in\ s\ dimension\ input}(i,j) \times max_{first\ dimension} \div max_{s\ dimension}$$

Use the calculation, we can make the contribution for each dimension to the classification equally, which can decrease the affect by the first dimension, the result can be shown as follow:



However, we can find that the classification of the cat has some inconsistent points, especially on the cat's faces. In addition, for the cat in the middle, we have a bit difficulty in classifying it. As a

result, we should do some revision.

(c) Further Improvement (Advanced: 10%)

Motivation

According to the question found in the Problem 1 (b), some approaches have to be proposed to improve the segmentation result. Here are some methods: 1) use the support vector machine to do fine-tuned segmentation having done the K-means; 2) use the PCA to decrease the dimension of the feature vectors; 3) Use the post-processing to fill the black hole in each sequence.

Approach and Procedures

PCA Method

At first, we should do the PCA to decrease the dimension of the vector in order to represent the vector effectively just like the Problem 1 (a). Since the value of dimension should be defined on my own, I have to find how many dimensions the output should be. Then, according to the result in MATLAB, I use the 10-12 dimension as the PCA result.

SVM Method

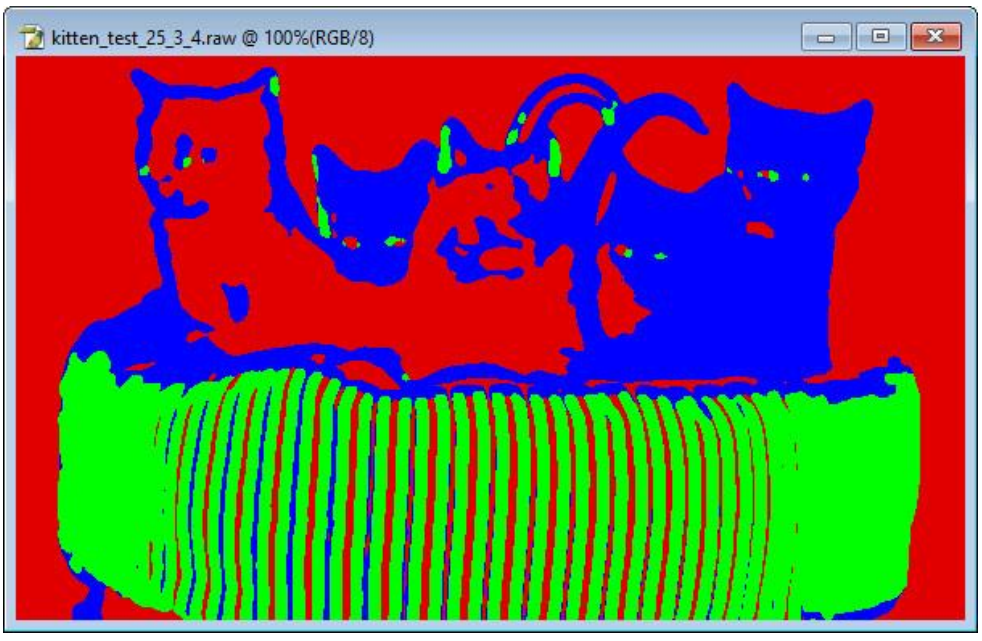
At first, we should also do the K-means to do the coarse segmentation. Then, we can choose some of the points as the training points to do the train the SVM model. Then, we can use the SVM to do the fine-tuned segmentation.

Post-Processing

At first, we will do the K-means to do the coarse segmentation. Then, for each channel, we consider filling the holes or inconsistent points by looking into the relation of the white and the black in each area. If there is an inconsistent area, we can use the background color to fill in.

Experimental Results

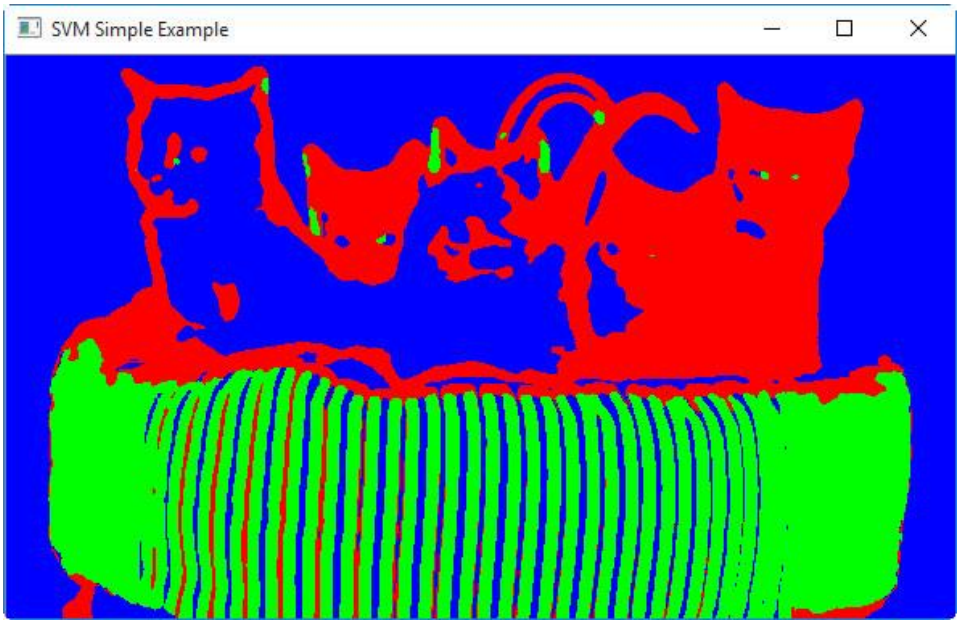
The Result of PCA can be shown as follow (in this question, the dimension is 11):



Use the PCA to do the classification

Compared with the result of the 25-D vector in the (b), we can find that the inconsistent point in the 11-D image is much less and smaller than the ones in the 25-D.

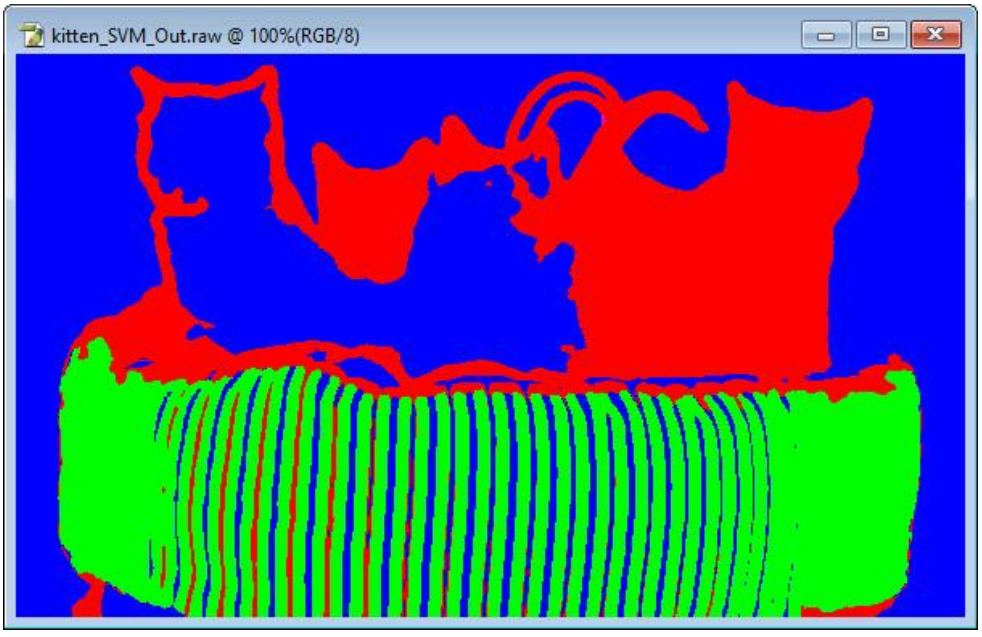
The result of the SVM can be shown as follow (the dimension is 11-D)



Use the PCA and SVM to do the classification

Compared with the solution that only use the PCA, we can find that the inconsistent area has decreased a little.

The result of Post-Processing can be shown as follow:



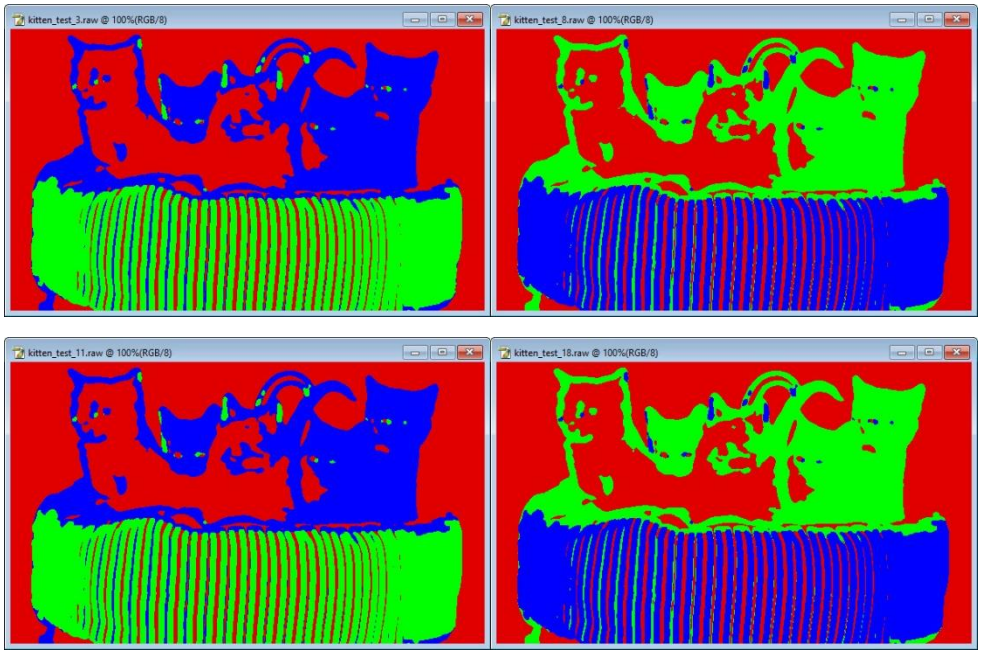
The result of the post processing

From the result, we can see that the post-processing can decrease the uncertainty of the image, which can make people tell the object in the area effectively.

Discussion

Use the PCA, we should consider how many dimensions we should choose. For this question, I decide to choose four dimensions: 3-D, 8-D, 11-D and 18-D.

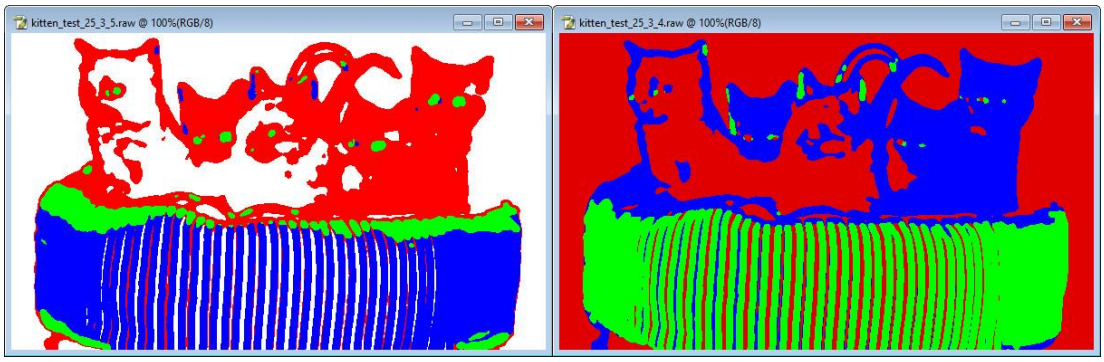
The result can be shown as follow:



The result for different PCA dimensions

As a result, we can see that if the dimension is larger than 13, there will be little difference among the output.

In addition, we can compare the result of PCA by 3 clusters with the 25-D by 4 clusters. The result can be shown as follow:

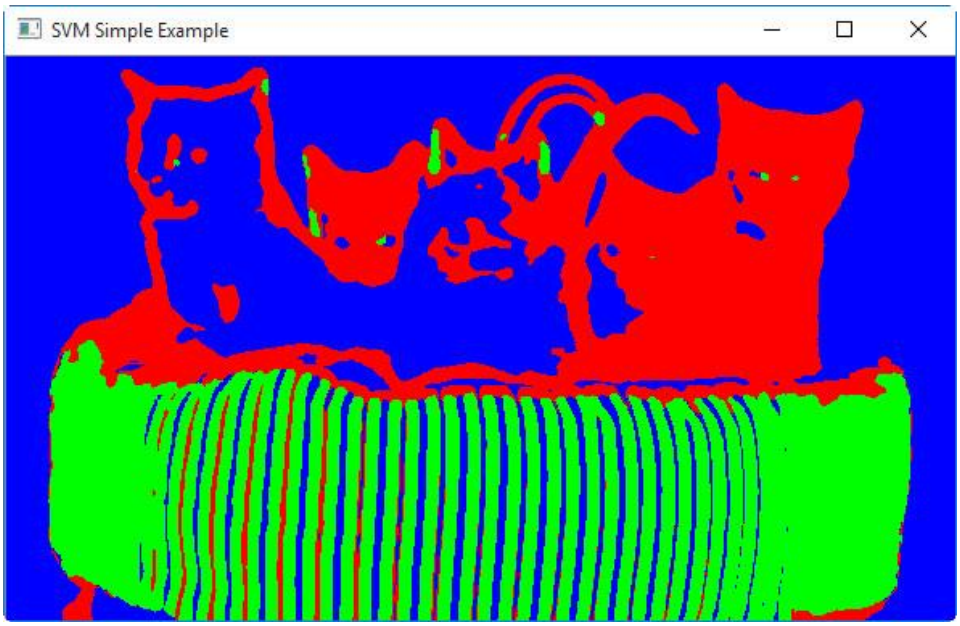


The comparison between the 25-D (Left) and PCA (Right)

As a result, we can find that the PCA can make a better classification compared with the 25-D solution.

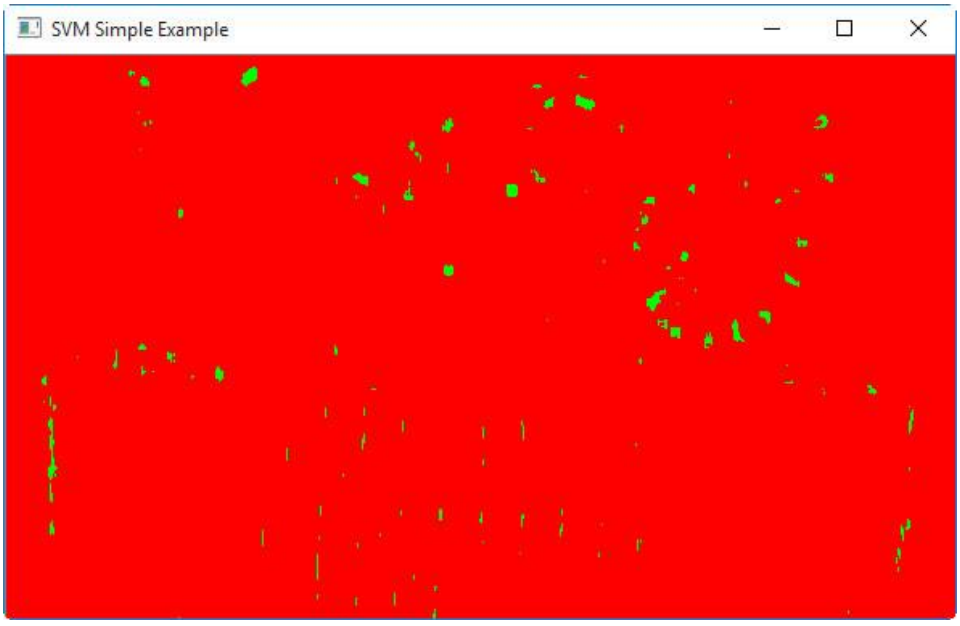
In addition, we can consider the Support Vector Machine. In the SVM, the most important thing is the SVM kernel because it can affect the boundary of each cluster significantly. There are different types of kernel: Linear, Polynomial, Radial basis function, Sigmoid, and so on. In this question, I will make the comparison among Linear, Sigmoid and Exponential Chi2 kernel.

The result of Linear is:



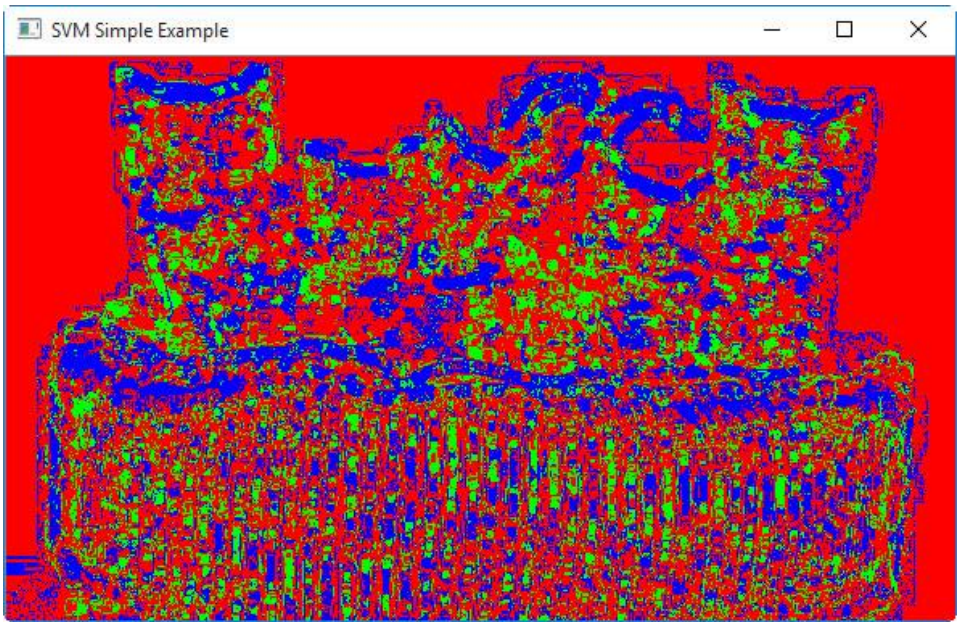
The result for Linear SVM Kernel

The result of Sigmoid is:



The result for Sigmoid SVM Kernel

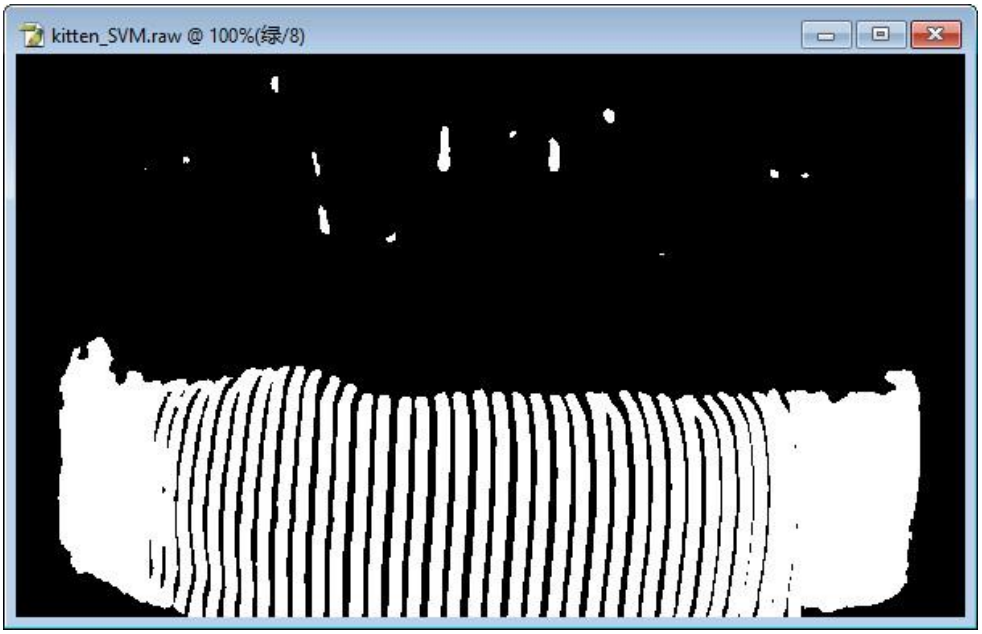
The result of Exponential Chi2 kernel is:



The result for Chi2 SVM Kernel

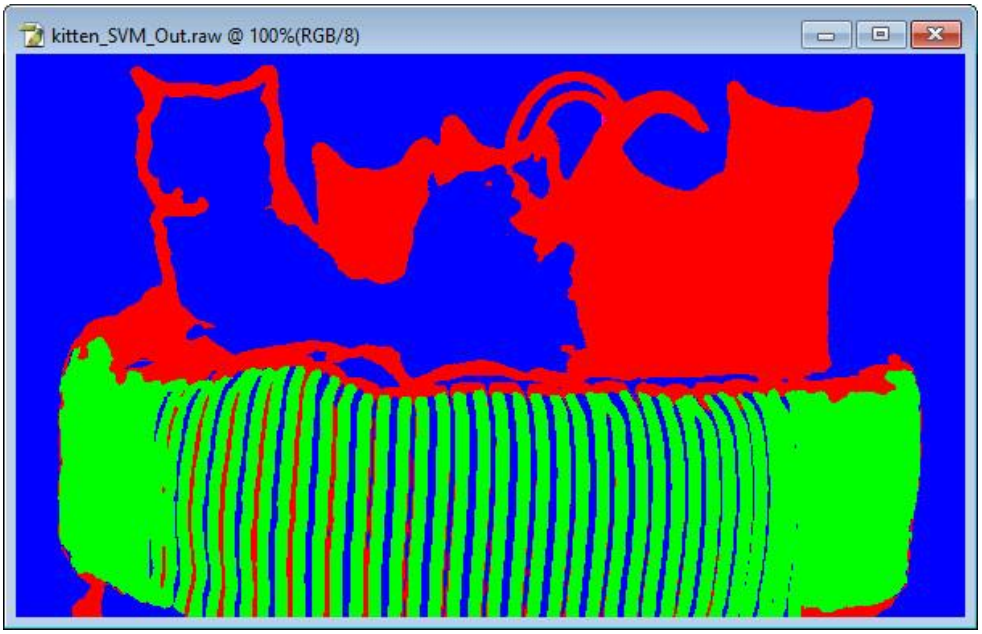
Among three kernels, we can find that the Linear Kernel is a good choice for the texture segmentation.

For the Post-processing, I decide to use the observation to fill the hole or delete the small contour in each RGB channel. For example, when looking into the green channel in the area, we can find that there are some small contour areas in the upper part of the image.



The channel of the green color relevant to the result for Linear SVM Kernel

Therefore, we just care about the specific area and turn the area into the background color. For the other channels, I will do the same procedure: check the hole area or small contour area and turn it to be the color around it. In this question, I just care about the area of the cat instead of the mat. Therefore, we can finally get the result below:



The Final result of post processing

From the picture, we can clearly find that there are some cats and there are two kinds of cats. As a result, the task of segmentation is satisfied.

Problem 2: Edge and Contour Detection (40 %)

(a) Canny Edge Detector (Basic: 15%)

Motivation

Use the Canny edge detector in Opencv to extract the edges in the image. Then try different the thresholds to test the effect of the edge detector.

Approach and Procedures

In this question, I will use the canny edge detector defined in the `imgproc.hpp` head file. The function of Canny can be shown as follow:

```
Canny(edge, edge, (double)g_nCurrValue, (double)g_nP, 3);
```

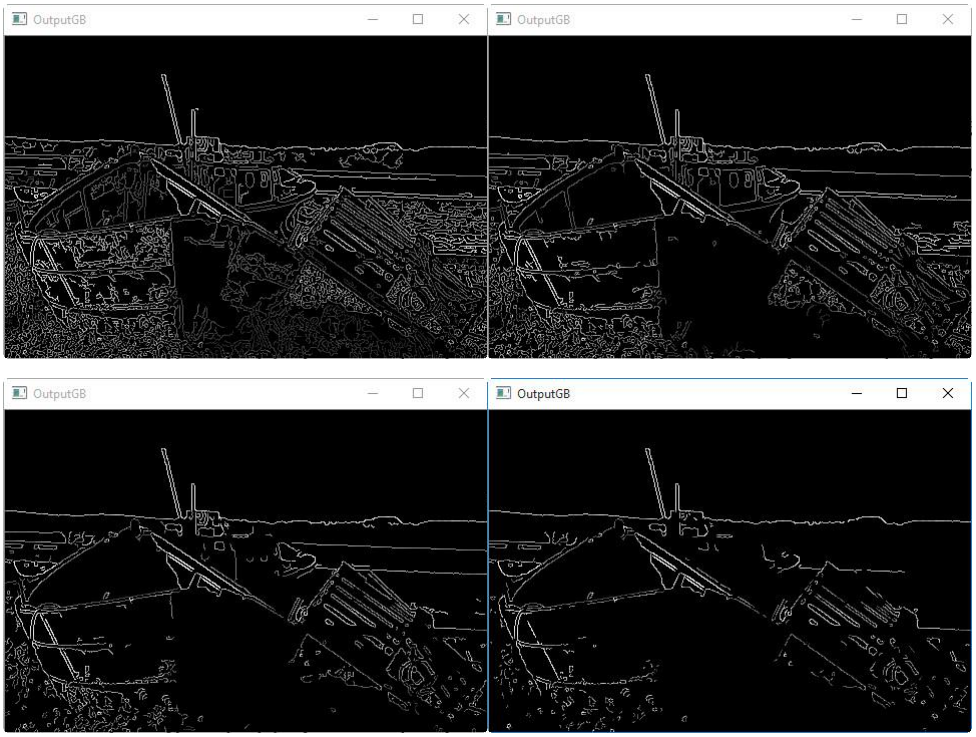
The first element is the input picture, which should be the gray image. The second one is the output edge image. The third and fourth elements are the two thresholds of the canny detector. And the final is the aperture size for the Sobel operator whose default value is 3.

Because we should consider different thresholds. It will be difficult and complicated to change those values in the cpp code. Therefore, I decide to use the track bar to change the threshold. Since there are two thresholds to coordinate, we should use two track bars to adjust the threshold. And because the thresholds are the magnitude of gradient, we can define the range of two bars are from 0 to 255. Then, the edge image will change when the bars representing the thresholds change.

Experimental Results

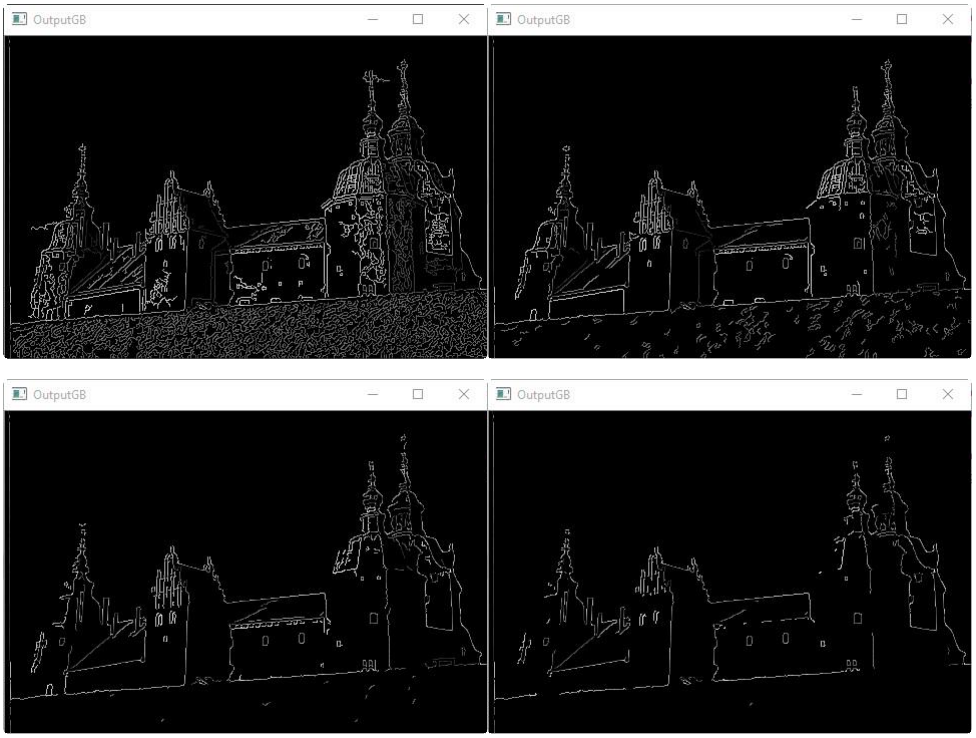
For each image, we will use four pairs of thresholds to see the effect of the edge detector when changing the thresholds: $(64, 0)$, $(128, 64)$, $(192, 128)$ and $(255, 192)$.

For the boat image, the result of detection can be shown as follow:



The result of canny detection by $(64, 0)$, $(128, 64)$, $(192, 128)$ and $(255, 192)$ Boat.raw

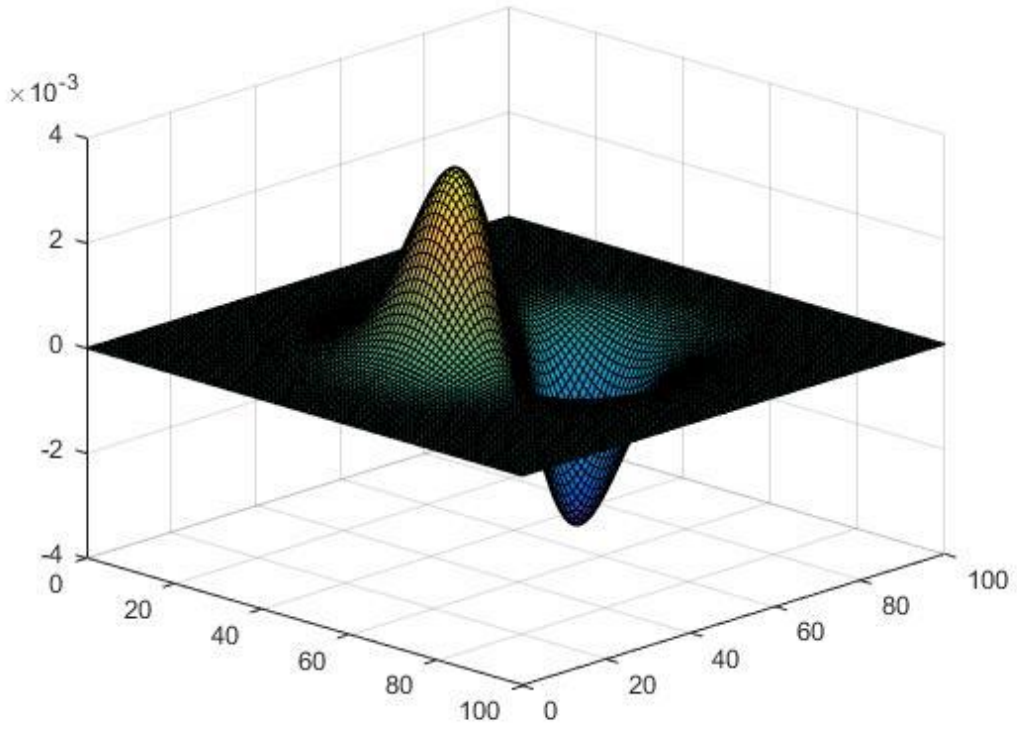
For the castle image, the result of detection can be shown as follow:



The result of canny detection by $(64, 0)$, $(128, 64)$, $(192, 128)$ and $(255, 192)$ Castle.raw

Discussion

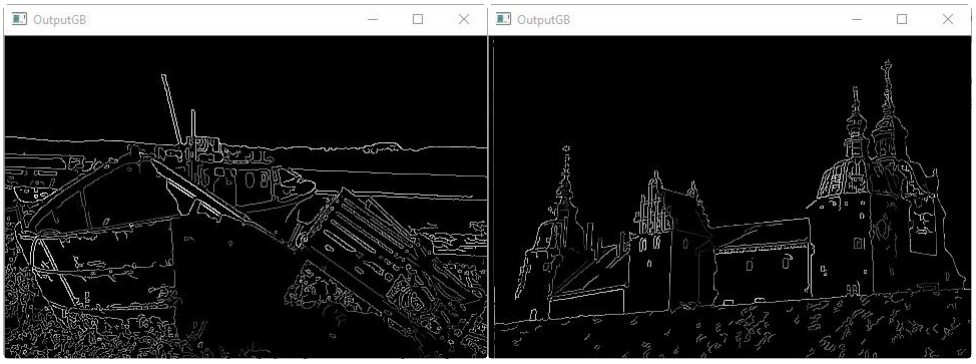
At first, we should discuss the procedure of the edge detection about the Canny. At first, the Gradient Gaussian function is applied to acquire the gradient in the image, which can be shown as follow:



Then, we use the two thresholds in the canny function to justify if the pixel is edge. If the magnitude of gradient is more than the larger threshold, we can regard it as the edge; if the magnitude of gradient is less than the smaller threshold, we do not regard it as the edge; for the gradient between two thresholds, we regard it as edge if it connects to the edge.

Based on the theory of the canny edge detection, we can compare two images again. Then, we can find that the castle image performs better when using the Canny. The reason is that there is a huge distinction about the pixel value between the background and the castle and the gradient of the background is much small. When we use the Canny, we can easily filter the background and keep castle by coordinating the thresholds. However, in the boat image, the gradient between the boat and the background is relatively small and the gradient of the image is large. As a result, it will be difficult to filter the background and keep castle by coordinating the thresholds.

However, if we can choose the threshold value carefully, we can get the location of the objects. The example for the boat and castle can be shown below:



Since the gradient between the boat and the background is relatively small and the gradient of the image is large for the boat image, we cannot only show the edges on the object by tuning the parameters. The reason is that if we just tune the parameter to delete the background gradients, we can also delete some of the edges on the object since the gradients is less than the background one. As a result, we cannot show the edges only on the object.

The according to the theory of the Canny edge detection, we can find that the relationship between the threshold value and image contents are really complicated.
For example, for the texture, if the texture is really rough, if we choose the larger threshold value really low, the gradient in the texture will be regarded as the edge, while gradient in the texture will be filtered surely only if the smaller threshold value is really high. Therefore, for filtering the rough texture, we can only choose the threshold value high enough.
However, for the smooth edge, the situation will be different. If we choose the smaller threshold value really high, the smooth edge cannot be regarded as the edge. Therefore, for keeping the smooth edge, we can only choose the threshold value low enough.
For the weak object boundary regions, the gradient between two objects is not clear. As a result, the situation is really similar with the smooth edge. If we choose the smaller threshold value really high, the boundary cannot be regarded as the edge. Therefore, for keeping the weak object boundary, we can only choose the threshold value low enough.

(b) Contour Detection with Structured Edge (Basic: 15%)

Motivation

Use the Structured edge and the random forest model to detect the edge in the picture. The output is the probability edge map which are generated in the gray picture. Having get the probability edge map, we can set the threshold to get the final binary edge map.

Approach and Procedures

In this section, the Structured edge detector and the Random forest will be discussed.
For Structured edge detection, the Random forest is the basic model to decide if it is an edge. Therefore, we can explain the process of the random forest construction and the Principle of the Random forest classification.

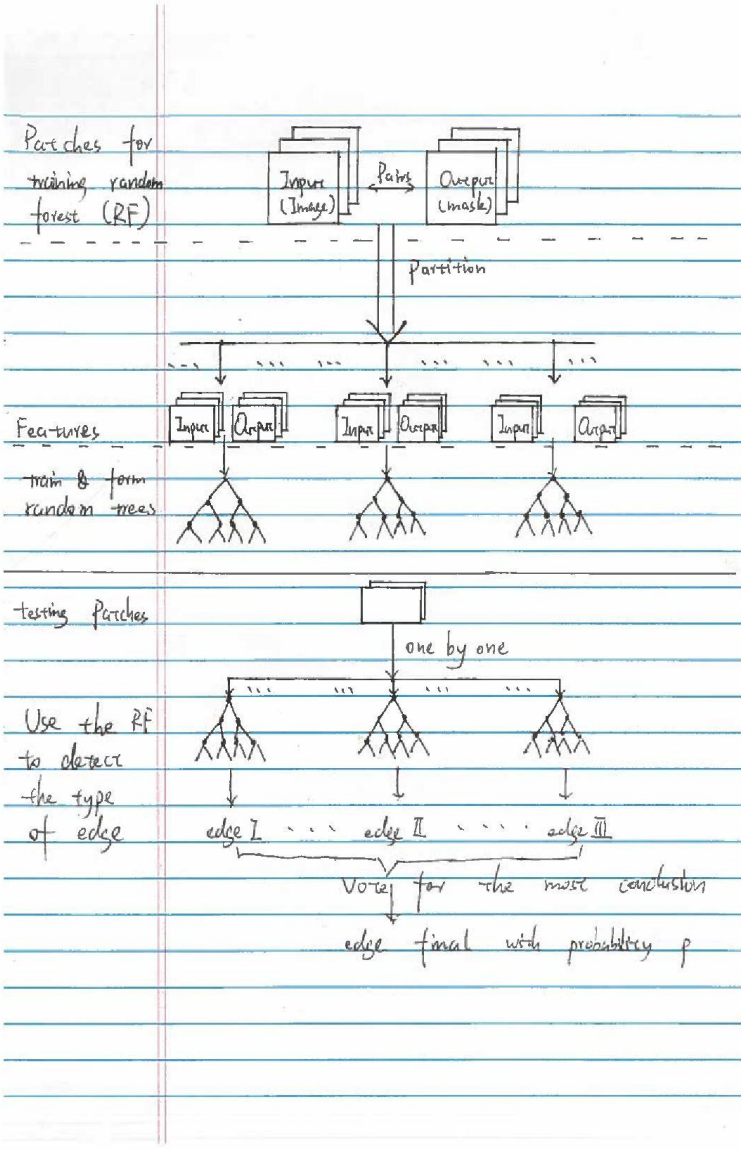
For building the random forest, we should do the steps as follow:

1. From the training data, which is a series of pairs relevant to the input (origin image patch) and the output (edge image), we choose some of the data pairs.

2. By a group of the data pairs, we should train the random tree. For each node of the random tree, it should classify one of the feature correctly.
3. We just choose another group of the training data pairs from the rest group of the data. Then, we train another new random forest. Finally, we can get the random forest to classify the edge.

For using the Random forest, we should put one of the testing patches into every N random trees in the RF. For each tree, a decision will be proposed. Then, we will obtain the final decision by comparing the relative number of the decisions and picking up the one with the highest rate.

Having got the idea about the building Random forest classifier, we can introduce the algorithm of the structured edge detection. The flow chart can be shown as follows:



For each training data, we have two image patches which form a pair: one is the origin image patch, the other is the corresponding mask or edge patch, which represent the input and the desired output. Then, we divide those training data into N groups.

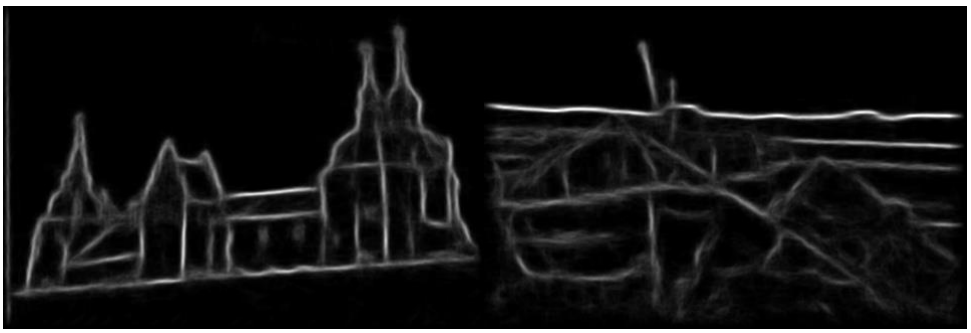
For each group of data, we at first convert each patch into a feature matrix. Then, we use the training data to train and form a random tree. When training and getting to a node of the tree, we will try to acquire the proper mapping function to spilt the training data into two groups, which means the classification. As a result, for this group data, we form a structure tree.

If we do such conduction for other groups of data, we can get N structured trees, which forms the random forest. As a result, the training process is finished.

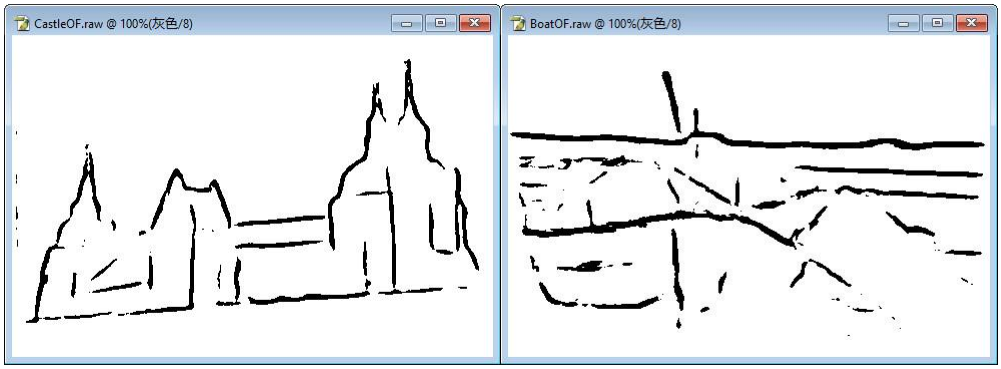
Then, for each patches from the testing images, we firstly convert the patch into a feature matrix. Then, we input the feature into each trees in the random forest. For each tree, we can get a judgement about which type of edge it is. Then, for all N judgements, we vote for the most agreed type of edges with a specific probability.

Experimental Results

In this question, I use the Opencv and the random forest in the Internet to implement the Structured edge Method. The probability of Castle and Boat can be shown as follow (Left is the Castle and Right is Boat):



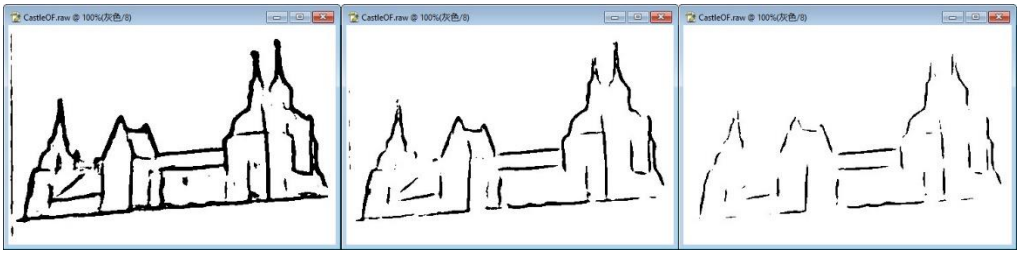
The best result of Castle and Boat in vision can be shown as follow:



Discussion

In the flow chart, we can get that we will admit the type of edge with the most probability and the output image represent the probability message for each pixel. In the picture, the bright the white is, the more possible it is the edge. Therefore, if the pixel value is x (the range is from 0 to 255), the point has the $x/255$ possibilities to be the edge. As a result, we can set the probability of threshold from 0 to 1 to get the binary edge map. The result can be shown as follow (in this question, we choose 0.2, 0.3 and 0.4 as the example):

For the castle image, the result is as follow:



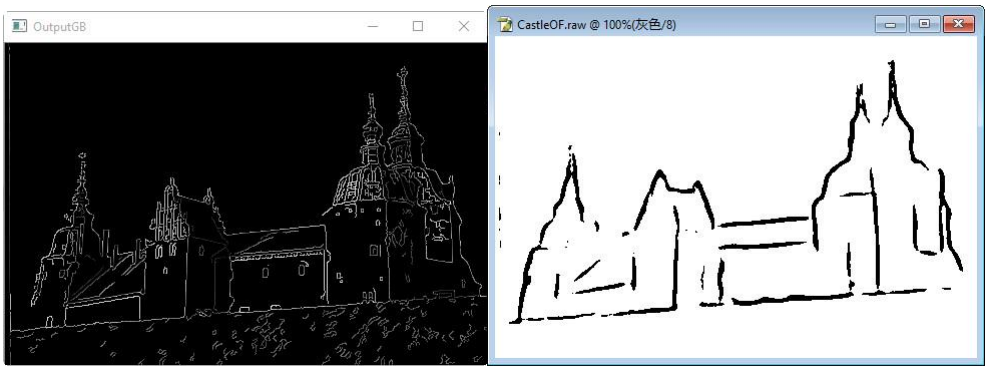
The result of binary edge image when $p = 0.2$ (Left), $p = 0.3$ (Middle), $p = 0.4$ (Right)

For the boat image, the result is as follow:

The result of binary edge image when $p = 0.2$ (Left), $p = 0.3$ (Middle), $p = 0.4$ (Right)

From the different results, we can find that we have to consider the probability threshold carefully. If the value is too large, we cannot get the proper edge of the image, which makes it difficult to clarify the objects; if the value is too small, we will receive a series of irrelevant message about the edge.

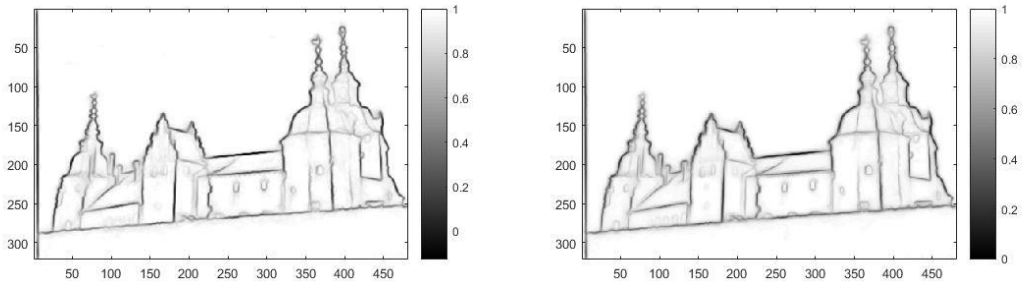
In addition, we can compare the Canny result with the Structured edge result. Take the Castle for example.



Then, we can find that even though the edge of the castle in structured edge method is wide, we can get the basic shape of the castle with little irrelevant message while we cannot filter the irrelevant message easily when we use the Canny method.

Next, I will discuss the effect of some parameters in the structured edge detector. We will use the MATLAB to evaluate them since there is no parameters to coordinate in the opencv.

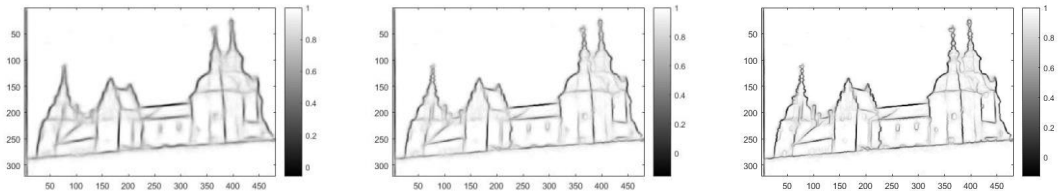
The first parameter is the multiscale. In the instruction, it has only two value: 0 and 1. I will use the castle to test its effect. The result can be shown as follow:



The output when multiscale is 0 (Left) and 1 (Right)

From the result, the multiscale can make the detector judge more pixels as the edge since there are more dark area around the edge. Since we find that the effect without the multiscale is enough, we can use both parameters. However, from the running time, we will choose the 0 multiscale because the running time of 1 multiscale is 0.798 second, which is greatly larger than the one of 0 multiscale.

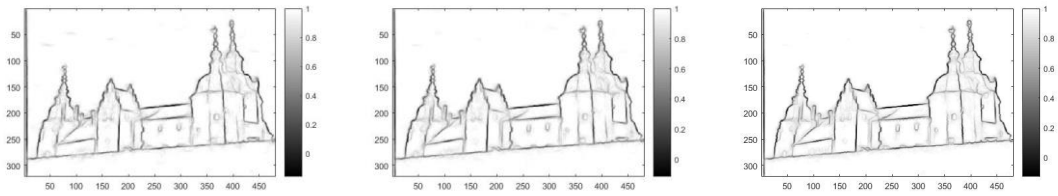
The second parameter is the sharpen. In the question, there are only three values: 0, 1 and 2. When changing the sharpen parameter, we can get the result as follow:



The output when the sharpen is 0 (Left), 1 (Middle) and 2(Right)

Even though the running time is larger when the sharpen value is higher (0.058s, 0.123s and 0.162s), we can find that the width of edge is thinner when the sharpen value is higher, which will improve the precision. Therefore, I will choose the 2 as the sharpen parameter.

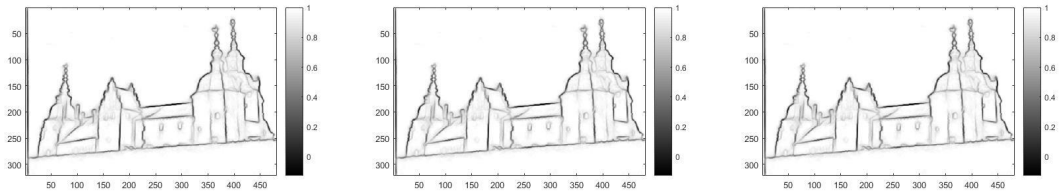
The third parameter is the nTreesEval. It can be set as integer larger than 1. Therefore, I will choose three parameters: 1, 2 and 4 to test the effect of the parameter, the result can be shown as follow:



The output when the nTreesEval is 1 (Left), 2 (Middle) and 4 (Right)

From the picture, we can find that even though the running time is larger when the nTreesEval value is higher (0.046s, 0.083s and 0.152s), we can find that the irrelevant messages are fewer when the nTreesEval value is higher. Therefore, I will choose the 4 as the nTreesEval parameter.

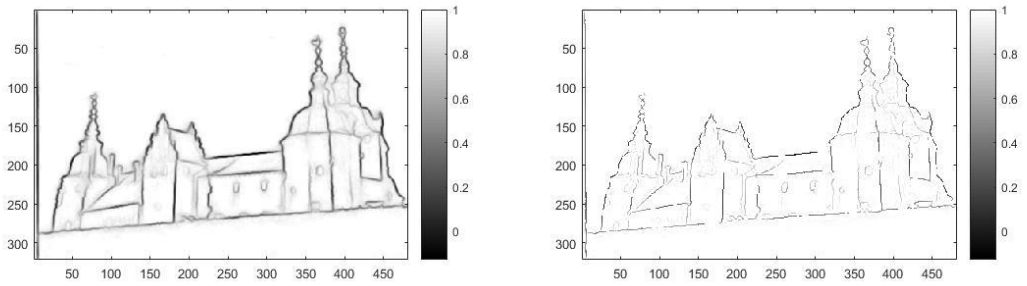
The forth parameter is the max number threads. It can be set as integer larger than 1. Therefore, I will choose three parameters: 1, 2 and 4 to test the effect of the parameter, the result can be shown as follow:



The output when the max number thread is 1 (Left), 2 (Middle) and 4 (Right)

From the picture, we can find that there is no difference among three images in quality. However, from the running time, we can find that the time is smaller when the max number threads is larger (0.163s, 0.161s and 0.157s). Therefore, I will choose the 4 as the nTreesEval parameter.

The last parameter is the nms. In the instruction, it has only two value: 0 and 1. I will use the castle to test its effect. The result can be shown as follow:



The output when nms is 0 (Left) and 1 (Right)

From the result, the nms can make the edge thinner, but it cannot make the edge connected, which makes people difficult to do the segmentation and make the recall decrease. Therefore, in this question, I will use the 0 as the nms parameter.

(c) Performance Evaluation (Advanced: 10%)

Motivation

In this section, I will use the MATLAB to do the structured edge and canny and use the F measure tool to evaluate the precision and recall. Then, we can use the precision can recall value to compute the F value.

Approach and Procedures

To evaluate the result of Canny detector, I just want to use some samples to conduct the evaluation because the edge detection result of Canny detector varies among the choice of two thresholds, which means that there are too many uncertainties. However, for the structured edge, we can consider many possibilities because the final output result can only be determined by the threshold of probability. Also, I will compare the measures between the result of Canny and the one of structured edge.

For an output edge image, we will get the F measure by comparing with ground truth image. When we compare those two images pixel by pixel, we will have four situations (in this table exist means the pixel is an edge pixel in the image):

	Output image	Ground truth
True Positive	Exist	Exist
True Negative	Not Exist	Not Exist
False Positive	Exist	Not Exist
False Negative	Not Exist	Exist

Then, we can evaluate the precision and recall parameters for the output image.

The precision is to justify whether all the candidate edge pixels in the output image are really the edge pixel in the ground truth image. The formula can be shown as follow:

$$\text{Precision} = \frac{\#\ \text{of True Positive}}{\#\ \text{of True Positive} + \#\ \text{of False Positive}}$$

The recall can be used to justify whether some of the ground truth edge pixels are not regarded as the edge pixels in the output image. The formula can be shown as follow:

$$\text{Recall} = \frac{\#\ \text{of True Positive}}{\#\ \text{of True Positive} + \#\ \text{of False Negative}}$$

Then, we can calculate the F measure. The formula can be shown as follow:

$$F = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Experimental Results

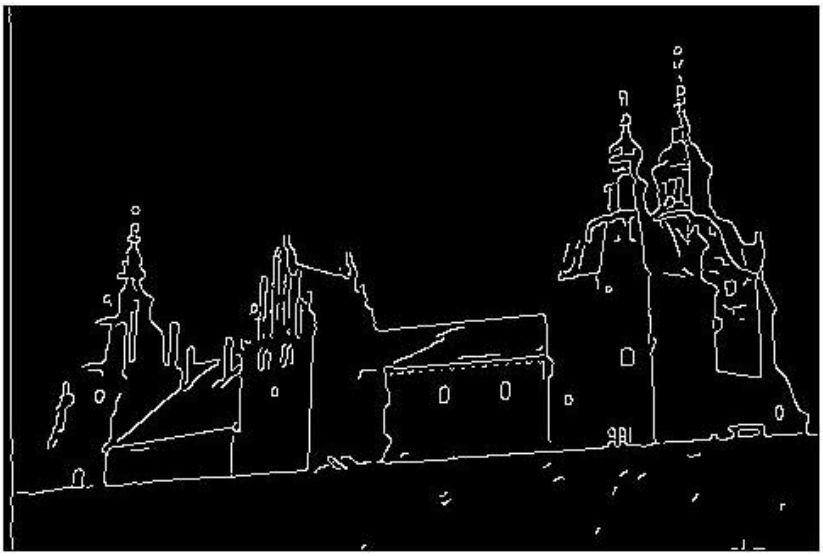
In this section, we can compare the effect of the Canny and structured edge. The best result of the Structured edge and the good result of Canny can be shown as follow:

For the castle, the result can be shown as follow:

Method	P of ground truth	R of Ground truth	Mean P	Mean R	Final F
Canny	0.3604	0.8900	0.4920	0.8643	0.63
	0.3570	0.8926			
	0.5061	0.8339			
	0.6306	0.8519			
	0.5083	0.8634			
	0.5894	0.8537			
Structured edge	0.5202	0.9316	0.6753	0.8686	0.76
	0.5220	0.9463			
	0.7129	0.8518			
	0.8210	0.8043			
	0.7025	0.8653			
	0.7731	0.8120			

And the output of the edge image can be shown as follow:

Canny



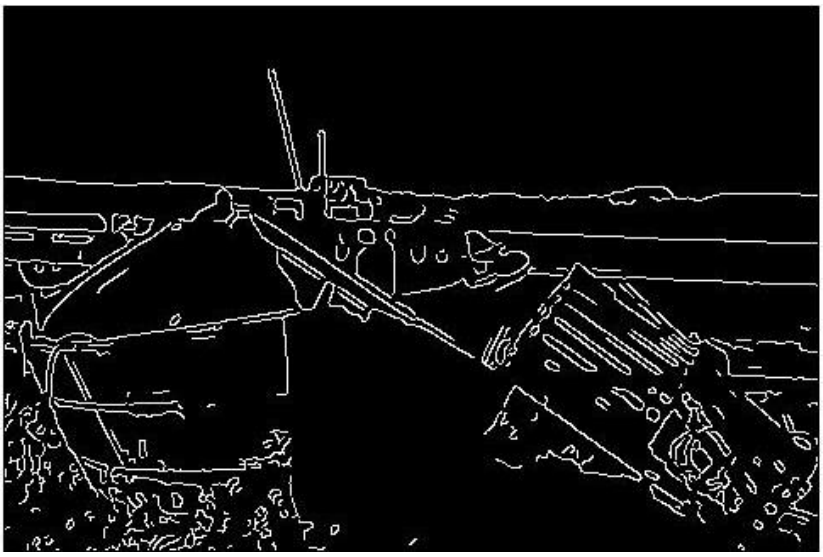
Structured edge

For the Boat image, the result can be shown below:

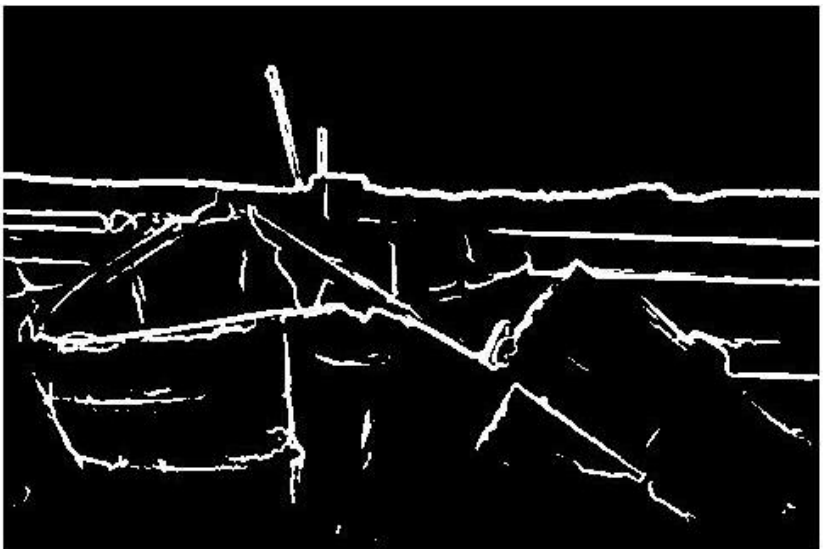
Method	P of ground truth	R of Ground truth	Mean P	Mean R	Final F
Canny	0.5220	0.7762	0.4278	0.8306	0.56
	0.3490	0.8544			
	0.5488	0.7939			
	0.4235	0.8713			
	0.3009	0.8340			
	0.4225	0.8536			
Structured edge	0.5760	0.4383	0.5741	0.5595	0.57
	0.5302	0.6644			
	0.5874	0.4349			
	0.6043	0.6363			
	0.4567	0.6477			
	0.6026	0.6231			

And the output of the edge image can be shown as follow:

Canny



Structured edge



Discussion

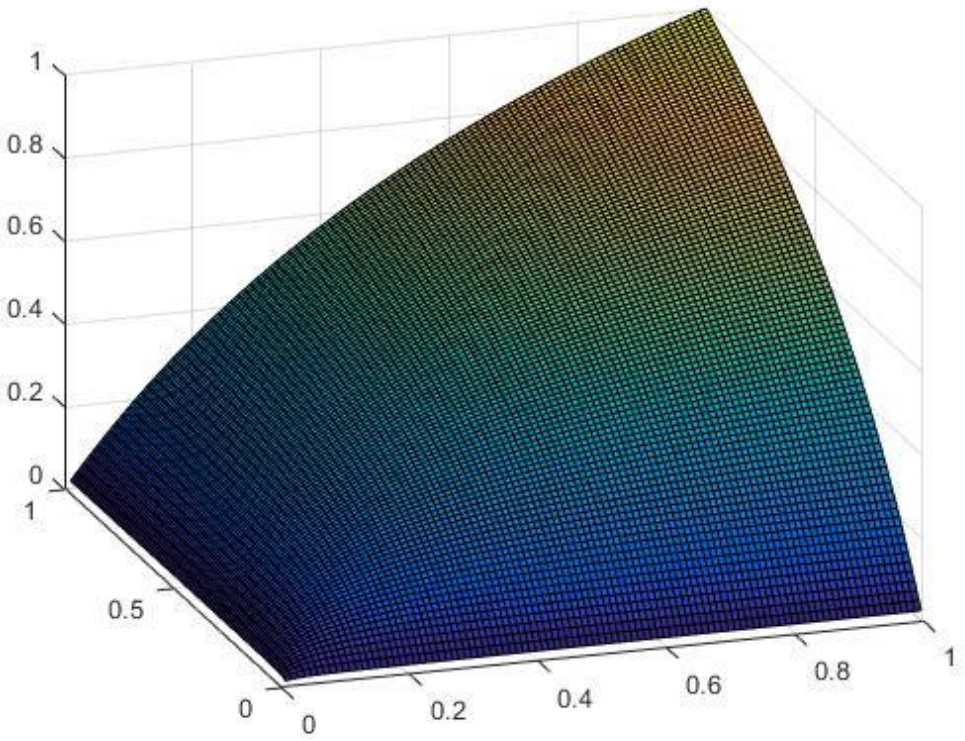
From the statistics above, we can find that the effect of the different edge detectors can be varied among the different type of the image.

For the first image, we can find that the gradient at the edge of castle is much clearer than the one in the background. Therefore, both the structured edge detector and canny detector can get good result. However, from the result images of both detectors, we can find some difference: the edge of SE is so bold that the F measure can be affected significantly. However, even though the edge of canny is not bold, many irrelevant messages are regarded as the edge. Therefore, both from the parameter and the vision, we can find that the structured edge can do well in the castle image.

There will be some issues to be considered in the Boat image. we can find that the gradient at the edge of castle is not clear compared with the one in the background. Therefore, both the structured edge

detector and canny detector is not likely to get a good result. However, if we can evaluate the output image, we can see some difference: for the canny detector, the output image has a lot of details, which make it impossible to be judged as boat. For the Structured edge image, those details are suppressed and it is more likely to see the fact that there is a boat. Therefore, even though the f measures for both images are same, the vision result is different.

In addition, we can find that the meaning of the f measure. From the formula, the f feature is the harmonic average of the precision and the recall. As a result, we can find that if one value is too large and the another is really small, we will get the f feature which is close the small value. Therefore, if the situation above is valid, we cannot get the maximum value. The relation can be shown as follow (the z is the final output of the f measure):



For another question: if the precision and recall are constant, what will happen? As a matter of fact, we can rewrite the formula above:

If $\text{Precision} + \text{Recall} = \text{C}$

$$F = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \times \frac{\text{Precision} \times (\text{C} - \text{Precision})}{\text{C}} = 2 \times \frac{\text{C} \times \text{Precision} - \text{Precision}^2}{\text{C}}$$

It is quadratic function, and when $\text{Precision} = \text{Recall} = \text{C}/2$, we can get the maximum value 0.5.

In this section, I use the function in MATLAB edge detection toolbox: `edgesEvalImg` to evaluate the F measure. However, there is a problem: we should use the specific ground truth file which is defined in BSDS-500. Therefore, for the ground truth image, I use the function below to convert the image file to the file we want.

```
namepart1 = 'D:/EE569_Assignment/3/P3/Castle_gt';namepart10 = 'Castle_gt';
```

```
namepart2 = 'D:/EE569_Assignment/3/P3/Boat_gt';namepart20 = 'Boat_gt';
num = ['1' '2' '3' '4' '5' '6'];
for k = 1:size(num,2)
    Name1 = strcat(namepart1, num(k), '.jpg');
    image = imread(Name1);
    image1 = im2double(image);
    for i = 1:size(image1,1)
      for j = 1:size(image1,2)
          if(image1(i,j) <= 0.5)
              image1(i,j) = 0;
          else
              image1(i,j) = 1;
          end
       end
    end
    imshow(image1);
    groundTruth = cell(1);
    groundTruth{1,1}.Boundaries = logical(1 - image1);
    Name2 = strcat(namepart10, num(k), '.mat');
    save(Name2,'groundTruth');
end
```

Therefore, we can get the ground truth file we want and use it to calculate the precision and recall.

Problem 3: Salient Point Descriptors and Image Matching (30 %)

(a) Extraction and Description of Salient Points (Basic: 10%)

Motivation

In this section, we will use both the SIFT and SURF to detect the key points in a series of images.

Approach and Procedures

In this question, we use the SIFT and SURF to do the key-point detection.

For the SIFT method, we can do the processing as follow:

1. For a specific Size of the image, we use Gaussian filter window with different standard deviation to do the convolution with the origin image. The Gaussian filter window can be shown as follow (σ is the standard deviation):

$$\frac{1}{\left(4\times e^{-\left(\frac{2}{\sigma^2}\right)}+4\times e^{-\left(\frac{1}{\sigma^2}\right)}+1\right)}\times\begin{bmatrix} e^{-\left(\frac{2}{\sigma^2}\right)} & e^{-\left(\frac{1}{\sigma^2}\right)} & e^{-\left(\frac{2}{\sigma^2}\right)} \\ e^{-\left(\frac{1}{\sigma^2}\right)} & 1 & e^{-\left(\frac{1}{\sigma^2}\right)} \\ e^{-\left(\frac{2}{\sigma^2}\right)} & e^{-\left(\frac{1}{\sigma^2}\right)} & e^{-\left(\frac{2}{\sigma^2}\right)} \end{bmatrix}$$

If the number of the picture with the same size is n, the relative deviation of the Gaussian can be shown as follow:

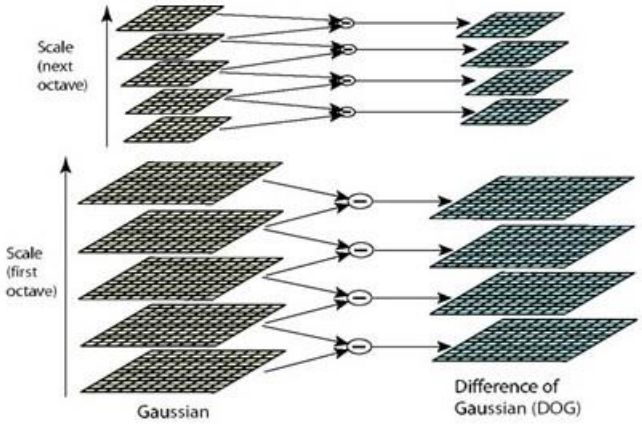
$$\sigma, k\sigma, k^2\sigma, k^3\sigma, \cdots, k^{n-1}\sigma$$

2. We can sample the image and get the image whose size is half of the previous image. Therefore, for the different size, we call a kind of size as an octave. In different octave, the value of σ is different (The Size of the Gaussian filter is same in all kinds of octaves). If for a specific image, the number of octave is m (the origin image size is in the first octave), and the number of pictures in the same

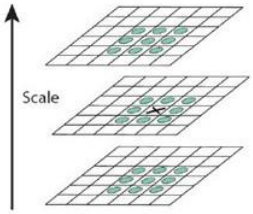
size is n, the deviation for each image can be shown as follow (in an octave):

$$2^{m-1}(\sigma, k\sigma, k^2\sigma, k^3\sigma, \cdots, k^{n-1}\sigma)$$

3. Having done the Gaussian filtration, we will do the deviation between two images in the same octave. As a result, we can get the Difference of Gaussian image. The process can be shown as the flow chart:



4. Then, for each pixel, we can compare it with the adjacent pixels. In this process, the adjacent pixels have 26 points: around it in the same image (8); direct above the image with same pixel locations (9) and direct below the image with same pixel locations (9). The processing flow chart can be shown as follow:



If the pixel is larger or smaller than any other pixels around it, we can regard it as a candidate key point and save the point.

For those candidate points, we can use the Hessian matrix to justify if it is real key points, which is shown as follows:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix}$$

Then, we will compute the trace and the determinant of the Hessian Matrix marked as $Tr(H)$ and $Det(H)$. Then we can compare the value below (in this part, r is 10):

$$\frac{Tr(H)^2}{Det(H)} \quad and \quad \frac{(r+1)^2}{r}$$

If the $(Tr(H)^2)/Det(H)$ is more than $(r+1)^2/r$, we delete the candidate key points.

For the SURF method, the procedure is different.

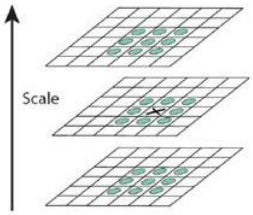
1. For a specific Size of the image, we use one Size of Gaussian filter window with different standard deviation to do the convolution with the origin image. If the number of the picture with the same size is n, the relative deviation of the Gaussian can be shown as follow:

$$\sigma, k\sigma, k^2\sigma, k^3\sigma, \cdots, k^{n-1}\sigma$$

2. In the SURF, the different octave has the same size of the image. The only difference is the size of

Gaussian filter. Therefore, we can get a group of the images varying among the difference octave and scale.

3. For each pixel, we calculate the determinant of Hessian Matrix and compare those values with the adjacent pixels. In this process, the adjacent pixels have 26 points: around it in the same image (8); direct above the image with same pixel locations (9) and direct below the image with same pixel locations (9). The processing flow chart can be shown as follow:

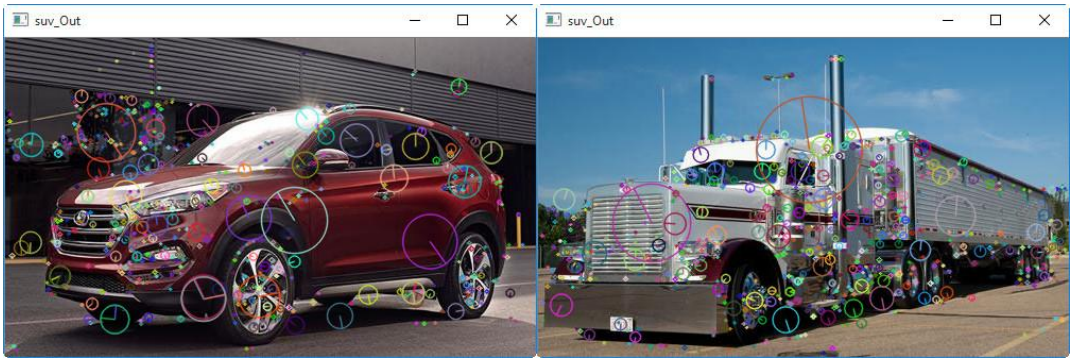


If the pixel is larger or smaller than any other pixels around it, we can regard it as a candidate key point and save the point.

4. We use the procedure similar with the SIFT to delete the bad candidate points.

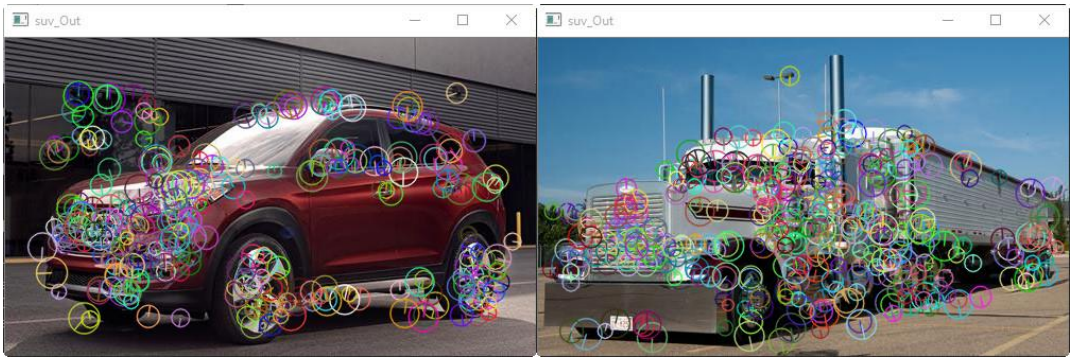
Experimental Results

The result of the SIFT key point detection can be shown as follow (use the default parameters):



The Key point in SUV (Left) and truck (Right)

The result of SURF key point detection can be shown as follow (use the default parameters):



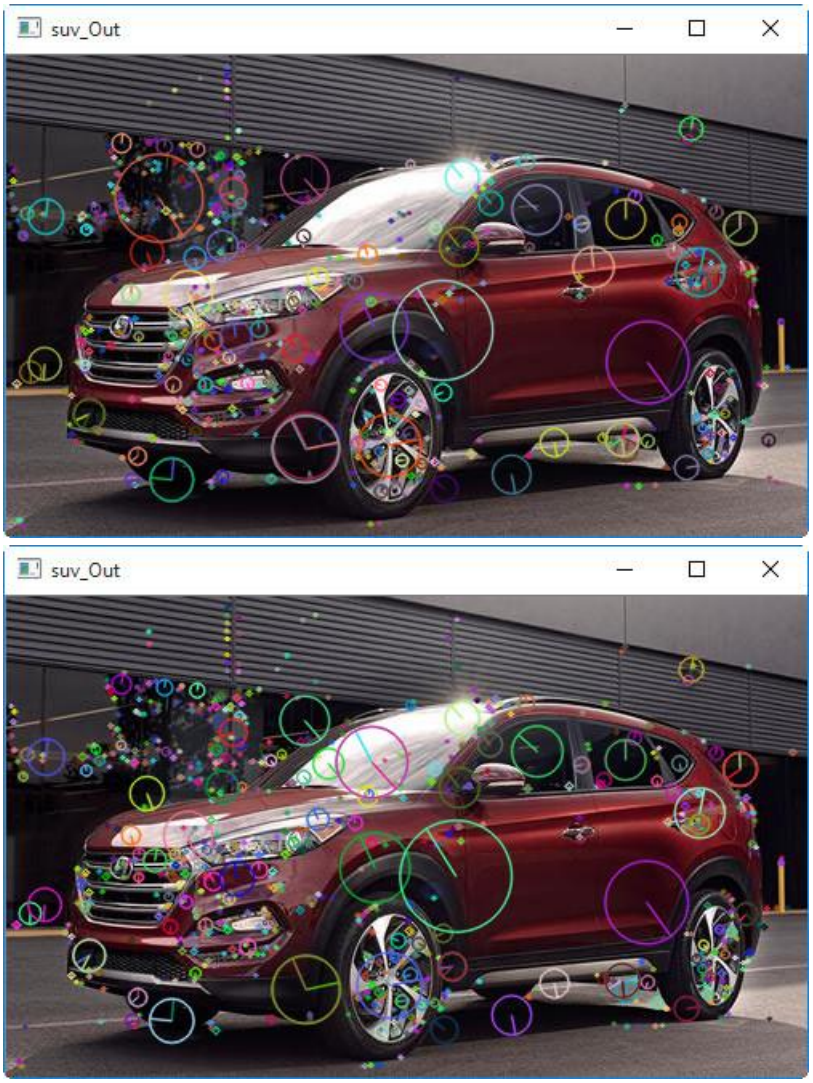
The Key point in SUV (Left) and truck (Right)

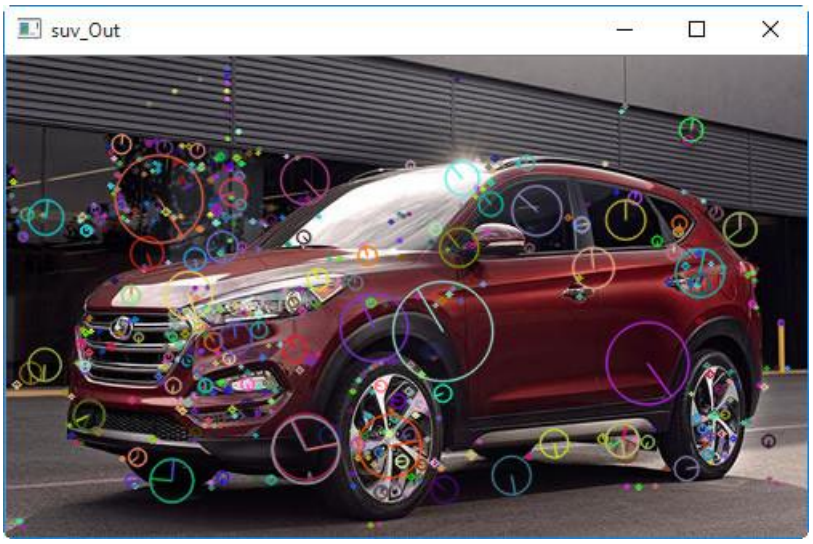
Discussion

In this section, I will tell the difference between the SURF and SIFT first. From the theory of the SIFT and SURF above, we can find that the speed of SIFT is significantly slower than the SURF because the dimension of the SIFT descriptors is much less than the one of the SURF descriptors. It can be verified by the time of SURF (1.226s) and SIFT (1.312s). However, we will get more key points in the SURF. The reason is that use the SURF, we will only change the area of the Gaussian window instead of the

size of the image. As a result, there will be more possible points in the SURF can be regarded as the local minimum or maximum points, which means that there will be much more points regarded as the key points by the method of SURF. In addition, in the SIFT, we will suppress the no maximum points then do the Hessian Process, which will filter many candidate key points. However, in the SURF, the Hessian process will be conducted then the non-maximum points will be suppressed. Therefore, the SURF will have more key points with faster calculation speed.

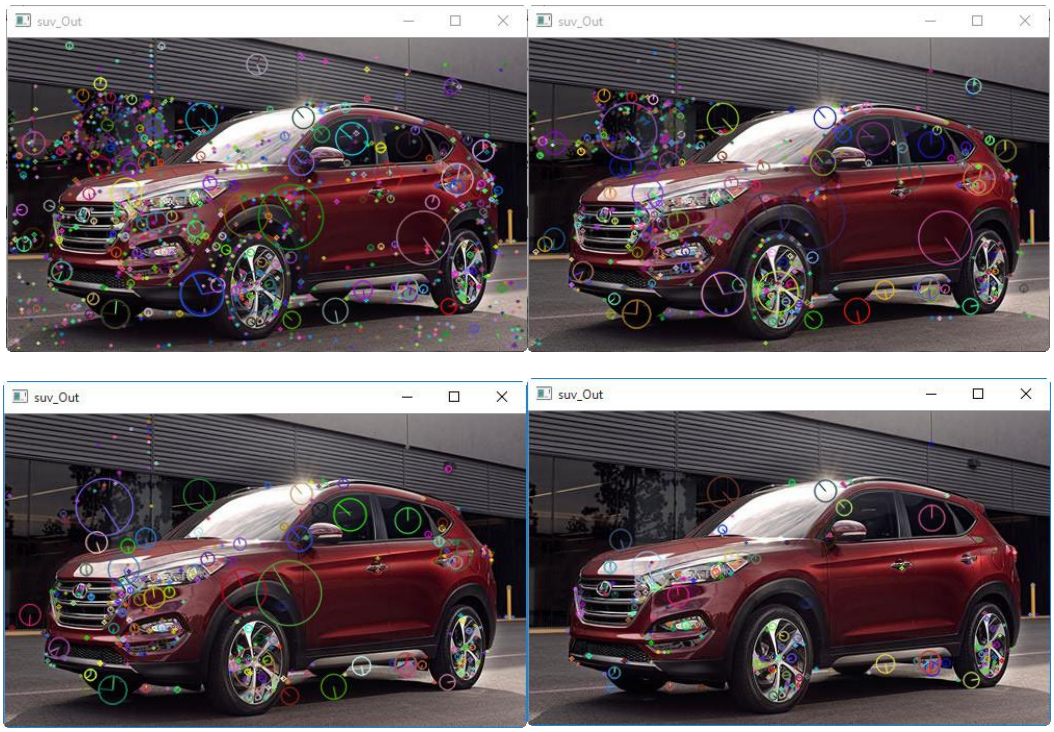
Then, I want to change some parameters to see the detection result.
For the SIFT, I want to change two parameters: the number of layers in each octave and the contrast threshold. The contrast threshold can be used to filter out weak features in semi-uniform (low-contrast) regions. The larger the threshold, the less features are produced by the detector. We can change the number of layers in each octave from 1 to 6 and the contrast threshold from 0 to 0.1.
The result of choose different number of layers can be shown as follow (I choose 3, 4 and 5 and the threshold is 0.04). In the picture, I take the SUV as example:





From the result, we can find that the change of the number of layers in each octave can make little different about the number and location of the key points.

The result of choose contrast thresholds can be shown as follow (I choose 0.01, 0.04, 0.07 and 0.1). In the picture, I take the SUV as example:



From the result, we can find that the larger the threshold is, the less the number of key points is. In the definition of the contrast threshold, we can find that it can filter the points least likely to be the key points. Therefore, if we can make it higher, we can preserve the most-likely key points and filter irrelevant points. However, this value cannot be set too large since it will filter some important key points.

(b) Image Matching (Basic: 10%)

Motivation

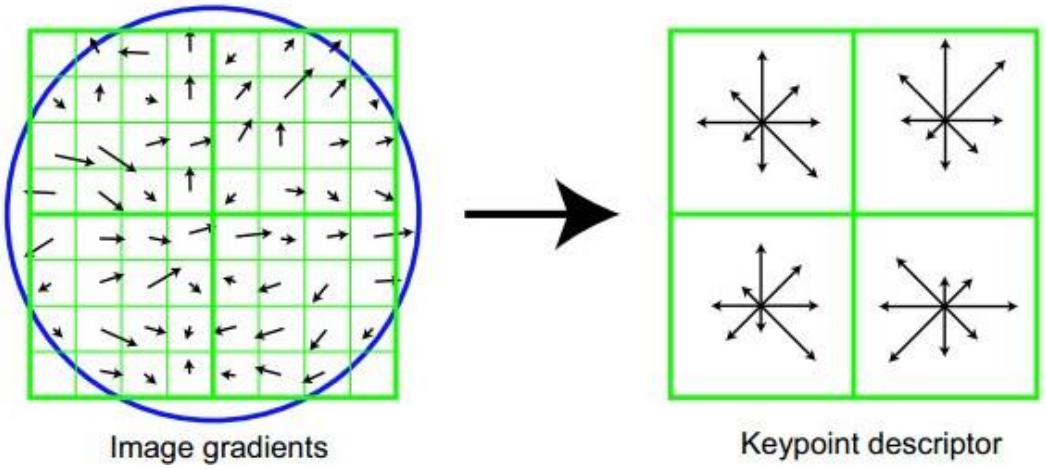
In this section, we will use both the SIFT and SURF to detect the key points and match the relevant key

points in a pair of images.

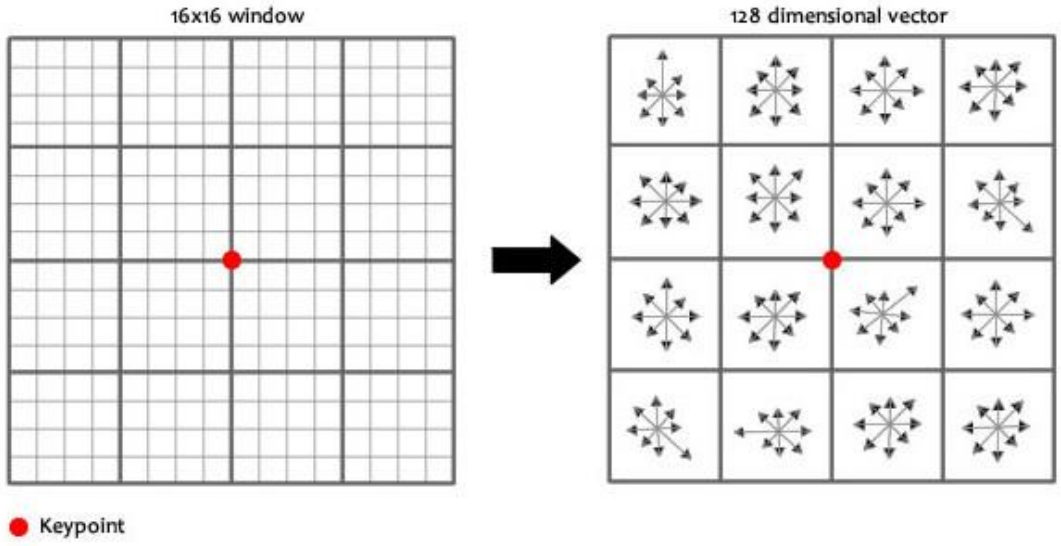
Approach and Procedures

For the SIFT, we can use the coordinates of the key points in the Problem 3 (a) to conduct the matching processing.

For a key point, we can draw 8x8 squares around it and find the gradient directions for each pixels in the window. Then, we use the Gaussian kernel to normalize the affection on the direction according to the relative location to the central points. Then, for each 4x4 square, we will do the histogram on the direction of gradient. Then, we can get the key point descriptor for the key point. The chart can be shown as follow:



In practice, we will use the 16x16 square window to form the key point descriptor. In this situation, we will also use the 4x4 square to do the histogram on the direction of gradient. Then, we can get $4 \times 4 = 16$ descriptor and there are eight directions for each descriptor. Therefore, there are $16 \times 8 = 128$ elements in the key point descriptor.

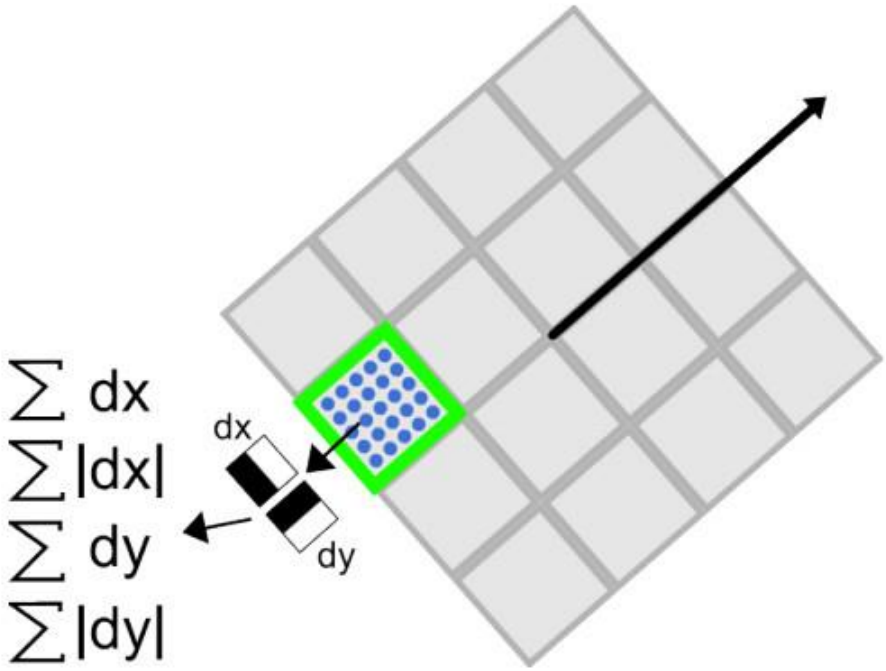


Then, for the pairs of image, we will compare the descriptor by calculating the Euclidean metric between two key points in two images. Then, for each point in the input, we will accept two nearest points in the output image. Then, we will find the ration of first nearest point distance to second nearest point distance. If the ratio is less than a threshold, we will regard first nearest point as a pair with the input point.

For the SURF, there are some differences in the matching procedure, especially in the construction of the

key point descriptor.

We form a square around the key points and divide the square into 16 parts. In each subpart, we calculate the wavelet features in horizontal and vertical direction. So in each subpart, there will be four values: sum of the wavelet features in horizontal; sum of the wavelet features in vertical; absolute sum of the wavelet features in horizontal; absolute sum of the wavelet features in vertical. As a result, for each key points, there are $4 \times 16 = 64$ dimension vector to represent the descriptor.



The matching process in SURF is similar with the one in the SIFT.

Experimental Results

In this section, I just use the default parameter to do the key value detection. The result can be shown as follow: (the first is SIFT, and the second is the SURF)

For the Convertible_1 and Convertible_2, the matching result is:

For the Convertible_1 and SUV, the matching result is:

For the Convertible_1 and Truck, the matching result is:

Discussion

As a matter of fact, when we use the SURF and SIFT to do the matching without filtering some pairs, I find that the pairs are so many that I cannot see the connections clearly. As a result, I decide to filter some pairs which are really distant each other. In the question, we use the variance "distance" in the `BFMatcher` to acquire the distance. Then, we get the maximum distance and minimum distance in those pairs and filter pairs with large distance. Then, we can get the Experimental result above.

For the Convertible_1 and Convertible_2, which are the same cars, since the main direction of two objects are similar, we can just see whether there are so many lines that are really steep. According to the observation, we can find that there are some matching pairs that is not consistent with the human's feeling while many pairs is correct from the people's perspective. In the matching picture, we can find that there are some pairs that cannot be considered by human beings as the pair since the background is different and some key points in the background are similar with the points in the objective (vehicle). As a result, we can find some pairs which is wrong from people's perspective.

For the Convertible_1 and SUV or Convertible_1 and Truck, which are totally different cars, there should not be so many pairs can be regarded as matching. However, from the result, we can find that there are too many points are considered as the same pairs even though they are completely wrong. The reason is that those points has similar main directions after conducting the SIFT or SURF progress. As a result, machine will combine those key points into a series of pairs.

(c) Bag of Words (Advanced: 10%)

Motivation

Use the k-means to divide the points into 8 groups, which will form a codebook or a bag of words. Then, do the histogram for key points in each picture and clarify the relation between each pictures.

Approach and Procedures

In this question, I will use the pictures SUV, Truck and Convertible_1 to form the codebook. The procedure can be shown as follow:

1. Use the SIFT or SURF to find out the key points in the SUV, Truck and Convertible_1 pictures and combine them into a group of key points.
2. For the group of key points, use the k-means to divide those points into eight groups according to the descriptor vector for each key point.
3. Each centroid which is the descriptor vector will represent one kind of the key points.

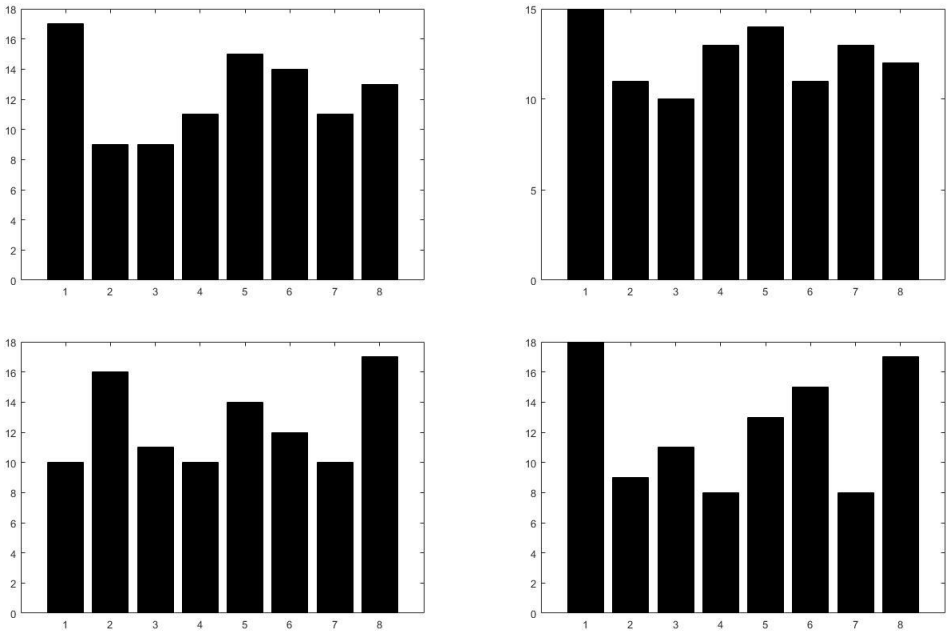
Having formed the codebook, we can use it to get the histogram for all SUV, Truck, Convertible_1 and Convertible_2 images. The process is as follow:

1. Use the Euclidean metric to calculate the distance between the descriptor vector of one key point and eight centroids. If it is closest to one of the centroids, we can conclude that it belongs to the group.
2. For a histogram, we can regard it as an 8-D vector. In order to calculate its distance, we should normalize it. At first, we should calculate the sum for each element in the vector. Then, we divide each element with the sum.
3. Finally, we can use the Euclidean metric to get the distance between the Convertible_2 and other images.

Experimental Results

Use the approach, we can get the both the histogram and distance of codebooks between the

Convertible_2 and other three images. The normalized histograms can be shown as follow (the first is SUV, second is truck, third is Convertible_1, forth is Convertible_2):



The distance can be shown as follow (for one result):

The codebook	The image	The distance
Convertible_2	SUV	0.069847
Convertible_2	Truck	0.101113
Convertible_2	Convertible_1	0.105167

As a result, we can find that the difference between the SUV and Convertible_2 is small, which means that they are closest.

Discussion

As a matter of fact, we will divide less than 1000 points into 8 groups. As a result, we may get some situation where the SUV and Convertible_2 is not closest. The reason is that the number of eight groups may not be completely comparable. That is, the classification is not complete and there is something wrong with the K-mean classification.

In addition, we assume that the Convertible_1 and Convertible_2 are closest, while the result shows that the Convertible_2 and SUV are closest. The reason is that the background between two images are different, which will have some interfere points when finding the key points. If the background can be removed, the result will be consistent with what we assumed.