EE569 Homework 3
Jimmy Chen

Problem 1: Texture Analysis and Segmentation

Part a) Texture Classification

For problem 1a, a host of different textures are given. These textures are to be analyzed using Laws feature extraction. These features are then grouped together and classified such that textures can be accurately identified and placed into clusters pertaining to their corresponding texture.

Twenty-five 5x5 Laws Filters are generated with the five 1D feature vectors shown in the table below. They are generated by multiplying each kernel with the transpose of every other kernel (including itself).

Name	Kernel
L5 (Level)	`[ 1 4 6 4 1 ]`
E5 (Edge)	`[ -1 -2 0 2 1 ]`
S5 (Spot)	`[ -1 0 2 0 -1 ]`
W5 (Wave)	`[ -1 2 0 -2 1 ]`
R5 (Ripple)	`[ 1 -4 6 -4 1 ]`

The twenty-five 5x5 Laws Filters are shown below

Laws Filter	Kernel
LL5	1 4 6 4 1 4 16 24 16 4 6 24 36 24 6 4 16 24 16 4 1 4 6 4 1
LE5	-1 -2 0 2 1 -4 -8 0 8 4 -6 -12 0 12 6 -4 -8 0 8 4 -1 -2 0 2 1
LS5	-1 0 2 0 -1 -4 0 8 0 -4 -6 0 12 0 -6 -4 0 8 0 -4 -1 0 2 0 -1
LW5	-1 2 0 -2 1 -4 8 0 -8 4 -6 12 0 -12 6 -4 8 0 -8 4 -1 2 0 -2 1

Laws Filter	Kernel
LR5	1 -4 6 -4 1 4 -16 24 -16 4 6 -24 36 -24 6 4 -16 24 -16 4 1 -4 6 -4 1
EL5	-1 -4 -6 -4 -1 -2 -8 -12 -8 -2 0 0 0 0 0 2 8 12 8 2 1 4 6 4 1
EE5	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
ES5	1 -0 -2 -0 1 2 -0 -4 -0 2 -0 0 0 0 -0 -2 0 4 0 -2 -1 0 2 0 -1
EW5	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
ER5	-1 4 -6 4 -1 -2 8 -12 8 -2 0 -0 0 -0 0 2 -8 12 -8 2 1 -4 6 -4 1
SL5	-1 -4 -6 -4 -1 0 0 0 0 0 2 8 12 8 2 0 0 0 0 0 -1 -4 -6 -4 -1
SE5	1 2 -0 -2 -1 -0 -0 0 0 0 -2 -4 0 4 2 -0 -0 0 0 0 1 2 -0 -2 -1
SS5	1 -0 -2 -0 1 -0 0 0 0 -0 -2 0 4 0 -2 -0 0 0 0 -0 1 -0 -2 -0 1

Laws Filter	Kernel
SW5	1 -2 -0 2 -1 -0 0 0 -0 0 -2 4 0 -4 2 -0 0 0 -0 0 1 -2 -0 2 -1
SR5	-1 4 -6 4 -1 0 -0 0 -0 0 2 -8 12 -8 2 0 -0 0 -0 0 -1 4 -6 4 -1
WL5	-1 -4 -6 -4 -1 2 8 12 8 2 0 0 0 0 0 -2 -8 -12 -8 -2 1 4 6 4 1
WE5	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
WS5	1 -0 -2 -0 1 -2 0 4 0 -2 -0 0 0 0 -0 2 -0 -4 -0 2 -1 0 2 0 -1
WW5	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
WR5	-1 4 -6 4 -1 2 -8 12 -8 2 0 -0 0 -0 0 -2 8 -12 8 -2 1 -4 6 -4 1
RL5	1 4 6 4 1 -4 -16 -24 -16 -4 6 24 36 24 6 -4 -16 -24 -16 -4 1 4 6 4 1
RE5	-1 -2 0 2 1 4 8 -0 -8 -4 -6 -12 0 12 6 4 8 -0 -8 -4 -1 -2 0 2 1

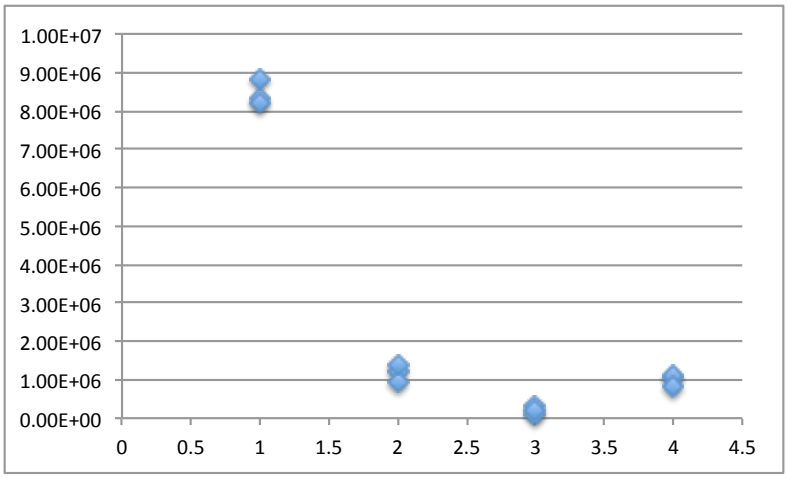
Laws Filter	Kernel
RS5	-1 0 2 0 -1 4 -0 -8 -0 4 -6 0 12 0 -6 4 -0 -8 -0 4 -1 0 2 0 -1
RW5	-1 2 0 -2 1 4 -8 -0 8 -4 -6 12 0 -12 6 4 -8 -0 8 -4 -1 2 0 -2 1
RR5	1 -4 6 -4 1 -4 16 -24 16 -4 6 -24 36 -24 6 -4 16 -24 16 -4 1 -4 6 -4 1

These twenty-five Laws Filters can be used extract features at every pixel from 12 image textures to make a 'feature image.' The average energy of each feature image can be calculated easily, such that each image will end up with 25 feature energy points. These 25 feature energy points can then be combined to form the so-called 'feature vector.' The table below shows the feature vector for each image. The first header corresponds to the image number (specifically, each image is named in the following format: "TextureX.raw"). The second header corresponds to the correct class cluster for each image. The left header corresponds to each Laws filter, in the order of the above table.

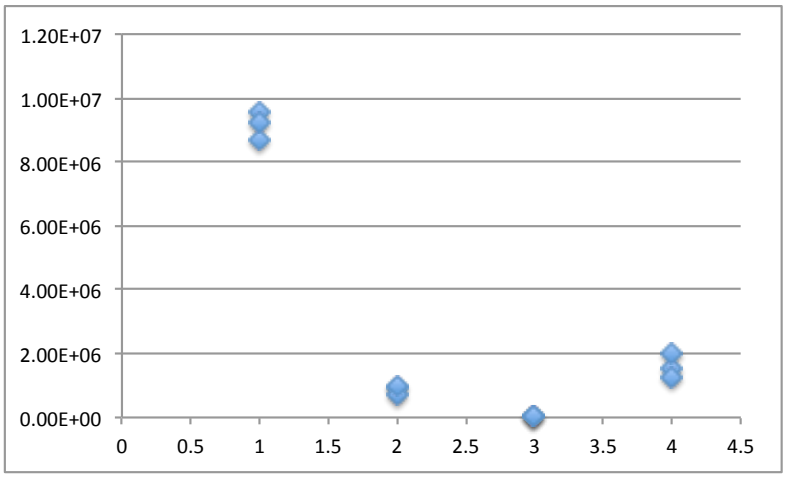
	1	4	6	2	9	12	7	8	10	3	5	11
	1	1	1	2	2	2	3	3	3	4	4	4
1	1.34E+08	1.20E+08	1.32E+08	2.33E+07	2.29E+07	3.02E+07	3.67E+06	1.19E+07	7.13E+06	2.91E+07	3.01E+07	2.18E+07
2	8.31E+06	8.20E+06	8.80E+06	1.20E+06	9.36E+05	1.36E+06	9.68E+04	3.20E+05	2.17E+05	1.08E+06	1.11E+06	8.05E+05
3	1.72E+06	1.90E+06	1.97E+06	2.99E+05	1.85E+05	3.17E+05	1.61E+04	2.89E+04	2.01E+04	1.75E+05	1.85E+05	1.42E+05
4	1.10E+06	1.21E+06	1.24E+06	1.84E+05	9.08E+04	1.95E+05	1.50E+04	1.72E+04	8.88E+03	9.35E+04	9.28E+04	8.04E+04
5	2.91E+06	2.78E+06	3.05E+06	3.45E+05	1.27E+05	3.90E+05	6.26E+04	7.31E+04	2.02E+04	2.28E+05	1.82E+05	2.03E+05
6	8.72E+06	9.54E+06	9.22E+06	9.01E+05	7.21E+05	9.59E+05	4.47E+04	7.24E+04	5.84E+04	1.51E+06	2.02E+06	1.24E+06
7	4.67E+05	5.25E+05	5.01E+05	6.28E+04	3.91E+04	7.10E+04	3.51E+03	4.20E+03	3.44E+03	3.63E+04	4.64E+04	2.53E+04
8	1.13E+05	1.28E+05	1.27E+05	1.85E+04	8.90E+03	2.18E+04	1.37E+03	1.41E+03	9.49E+02	6.85E+03	8.14E+03	4.88E+03
9	8.42E+04	9.09E+04	9.46E+04	1.28E+04	4.85E+03	1.61E+04	1.59E+03	1.64E+03	6.81E+02	4.40E+03	4.49E+03	3.33E+03
10	2.46E+05	2.33E+05	2.58E+05	2.79E+04	7.07E+03	3.57E+04	7.24E+03	7.75E+03	1.27E+03	1.24E+04	1.01E+04	1.03E+04
11	1.77E+06	2.34E+06	1.93E+06	1.82E+05	1.37E+05	1.98E+05	1.52E+04	1.86E+04	1.67E+04	3.12E+05	4.26E+05	2.38E+05
12	1.11E+05	1.36E+05	1.24E+05	1.41E+04	8.44E+03	1.70E+04	1.41E+03	1.50E+03	1.21E+03	8.27E+03	1.08E+04	4.99E+03
13	3.19E+04	3.73E+04	3.64E+04	4.55E+03	2.14E+03	6.28E+03	6.09E+02	6.18E+02	3.87E+02	1.63E+03	1.95E+03	9.66E+02
14	2.67E+04	2.86E+04	3.08E+04	3.57E+03	1.27E+03	5.45E+03	7.42E+02	7.65E+02	2.94E+02	1.10E+03	1.08E+03	6.97E+02
15	8.35E+04	7.95E+04	9.12E+04	9.21E+03	1.92E+03	1.42E+04	3.47E+03	3.76E+03	5.65E+02	3.17E+03	2.44E+03	2.37E+03
16	1.00E+06	1.50E+06	1.13E+06	1.02E+05	7.97E+04	1.33E+05	1.64E+04	1.90E+04	1.65E+04	1.78E+05	2.07E+05	1.04E+05
17	7.30E+04	9.04E+04	8.55E+04	8.28E+03	5.37E+03	1.20E+04	1.62E+03	1.67E+03	1.26E+03	5.91E+03	7.12E+03	2.65E+03
18	2.33E+04	2.77E+04	2.80E+04	2.89E+03	1.47E+03	4.93E+03	7.45E+02	7.50E+02	4.31E+02	1.18E+03	1.34E+03	5.25E+02
19	2.11E+04	2.30E+04	2.55E+04	2.57E+03	9.30E+02	4.95E+03	9.46E+02	9.64E+02	3.49E+02	8.11E+02	7.46E+02	3.91E+02
20	7.21E+04	6.92E+04	8.29E+04	7.99E+03	1.47E+03	1.56E+04	4.63E+03	4.80E+03	7.13E+02	2.37E+03	1.67E+03	1.39E+03
21	2.05E+06	2.97E+06	2.34E+06	2.03E+05	1.82E+05	4.01E+05	7.76E+04	8.35E+04	7.10E+04	3.68E+05	3.22E+05	1.42E+05
22	1.60E+05	1.96E+05	2.00E+05	1.74E+04	1.29E+04	3.52E+04	7.70E+03	7.84E+03	5.22E+03	1.47E+04	1.46E+04	4.26E+03
23	5.52E+04	6.77E+04	7.26E+04	6.71E+03	3.73E+03	1.50E+04	3.68E+03	3.64E+03	1.90E+03	3.08E+03	2.97E+03	8.66E+02
24	5.44E+04	6.16E+04	7.10E+04	6.91E+03	2.55E+03	1.71E+04	4.90E+03	4.83E+03	1.65E+03	2.23E+03	1.71E+03	6.67E+02
25	2.05E+05	2.04E+05	2.55E+05	2.68E+04	4.48E+03	6.83E+04	2.59E+04	2.45E+04	3.55E+03	6.80E+03	4.12E+03	2.58E+03

Looking at the feature vector table, discriminating the images correctly based on a specific features is possible for a few features but not all. For instance, feature 3, 6, 8 are a few specific features that are able to discriminate the images correctly. The energy response for these features for each image is shown in the below graphs, and plotted by their image class.

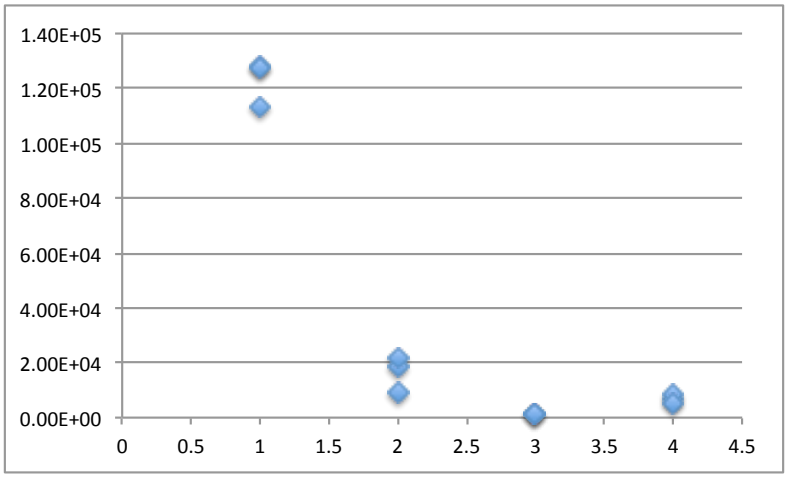
Feature 3 energy response



Feature 6 energy response



Feature 8 energy response



Inspection of the graphs will result in fairly confident discrimination for each class. Values between class 2 and class 4 seem close in distance, and inspection of the data shows that they are indeed separable for these features.

Applying the k-means algorithm on these feature vectors shows that most of the images are correctly.

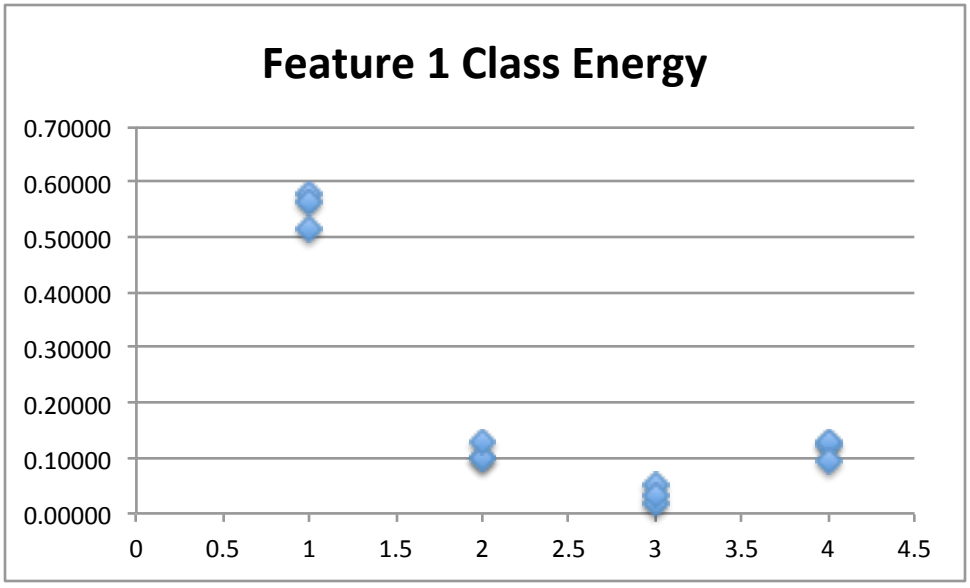
Image #	Actual Class #	Predicted Class #
1	0	0
2	0	0
3	0	0
4	1	1
5	1	1
6	1	3
7	2	2
8	2	2
9	2	2
10	3	3
11	3	3
12	3	1

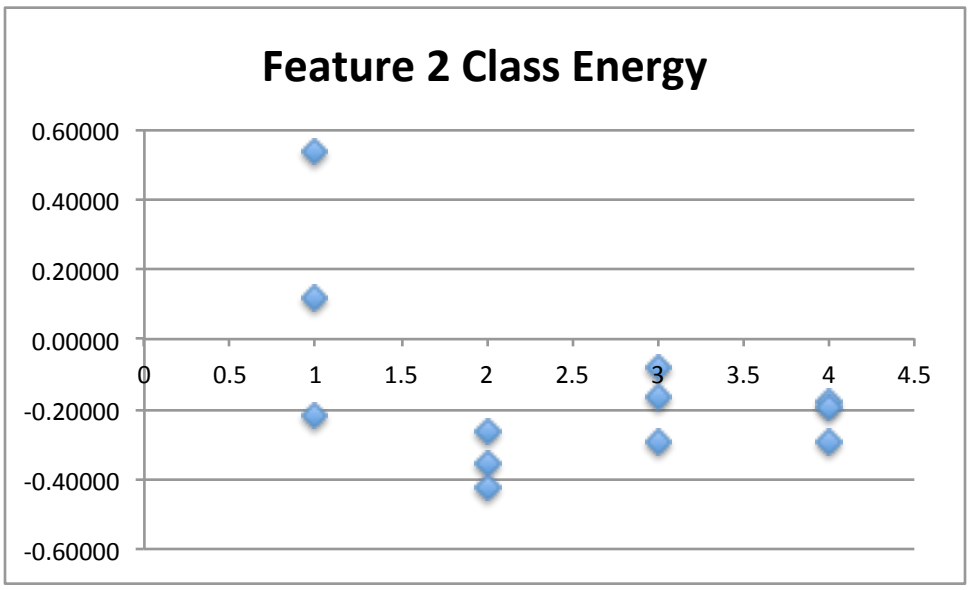
The table shows two classification errors for image 12 and image 6, which gives an accuracy of 83%.

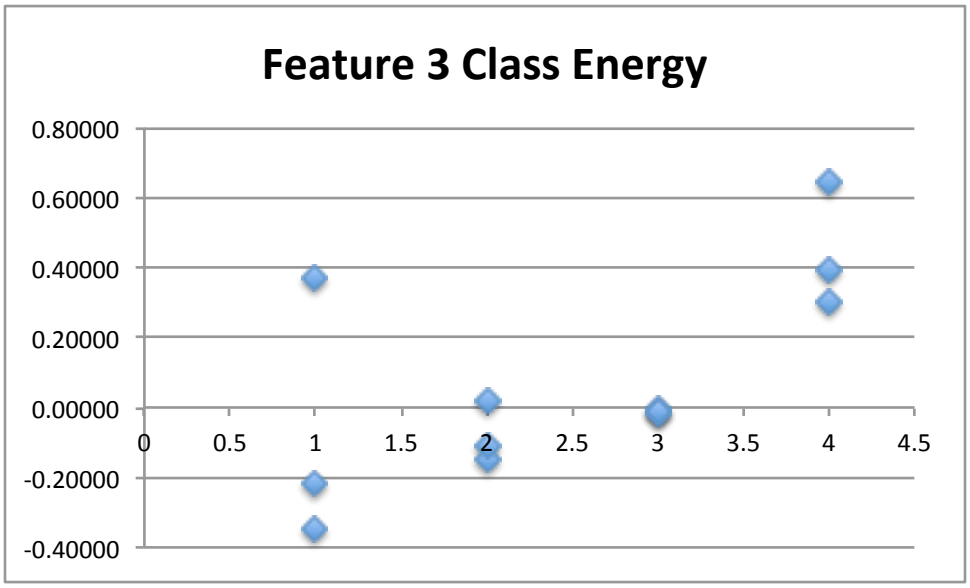
The 25 feature dimension space can be needlessly complex to analyze, and its overall discriminant power could be summarized into a few dimensions. Principal Component Analysis (PCA) is used to reduce the dimensionality from 25 to 3. The resulting 3-D feature vectors are shown in the table below.

Image #	Class #	Feature 1	Feature 2	Feature 3
1	1	0.57777	-0.21869	-0.21823
4	1	0.51507	0.53422	0.37111
6	1	0.56511	0.11912	-0.34812
2	2	0.100242	-0.262624	-0.149581
9	2	0.0982909	-0.352387	0.0212311
12	2	0.129599	-0.422889	-0.113507
7	3	0.0157477	-0.0805802	-0.00067916
8	3	0.051346	-0.29739	-0.0198263
10	3	0.0306707	-0.167873	-0.00749389
3	4	0.125209	-0.292019	0.393744
5	4	0.129407	-0.182628	0.641507
11	4	0.0937031	-0.197144	0.301229

However, the resulting feature vectors don't give a better discriminant representation of the features. Below are the same plots as previously that group the images by their classes and their corresponding energy.







One can see that features 2 and 3 have energies in classes that are not easily discriminable. As a result, the accuracy performance of the k-means clustering algorithm is not so good either.

Image #	Actual Class #	Predicted Class #
1	0	0
2	0	2
3	0	0
4	1	1
5	1	1
6	1	1
7	2	1
8	2	1
9	2	1
10	3	3
11	3	3
12	3	3

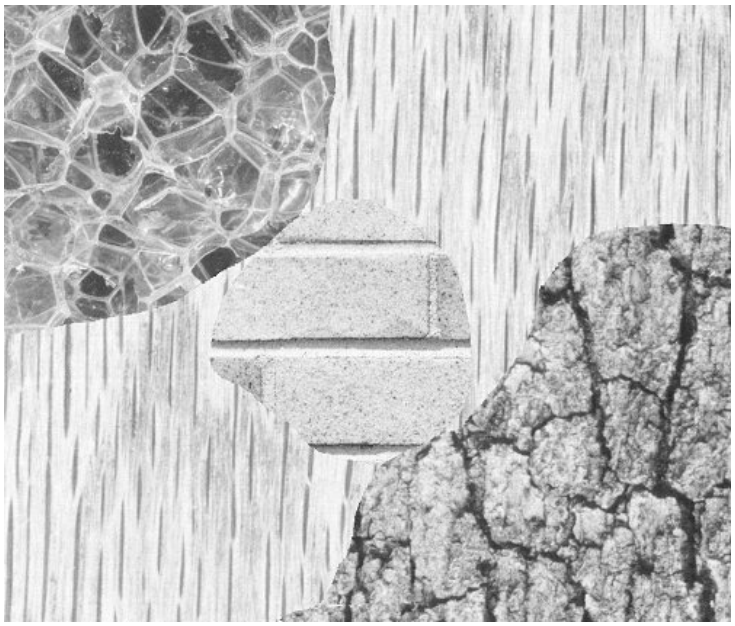
With dimensionality reduction the misclassification rate is doubled to 4 images, giving an accuracy performance of 67%. The misclassification is logical given the result of the feature class energies. The discriminant power of the images are decreased significantly, with the second class and the third class the least discriminable. This observation is reflected in the k-means clustering algorithm. The likely reason for this is due to the uneven energy for each feature vector extracted by the Laws filter. While certain features may have strong discriminant power, their absolute energy maybe dwarfed by others, and therefore will be represented more weakly after computing PCA.

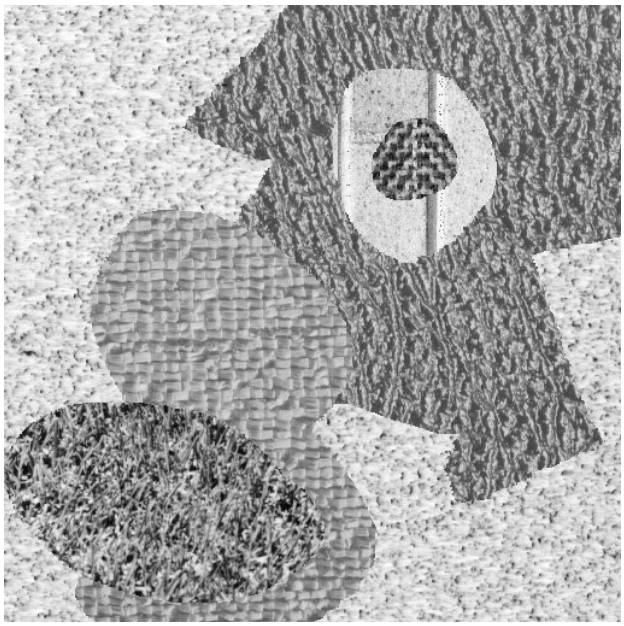
A solution to this problem is to normalize the feature vectors according to the images, so that their relative feature strengths are more accurately represented. In fact, when the same k-means algorithm and PCA was performed after normalization, the results improved dramatically. The class prediction without PCA became 100% accurate and class prediction with PCA only had one classification error.

Image #	Actual Class #	Predicted Class # without PCA	Predicted Class # with PCA
1	0	0	0
2	0	0	0
3	0	0	0
4	1	1	1
5	1	1	2
6	1	1	1
7	2	2	2
8	2	2	2
9	2	2	2
10	3	3	3
11	3	3	3
12	3	3	3

Part b) Texture Segmentation

Texture segmentation is similar to texture classification except on a more granular level. Instead of calculating the energy of each Laws filter response for the entire image, the energy is calculated for every pixel based on its neighborhood pixels. Each pixel will then have a feature vector, which can then be classified into a specific textures. The two images to be segmented are shown below.

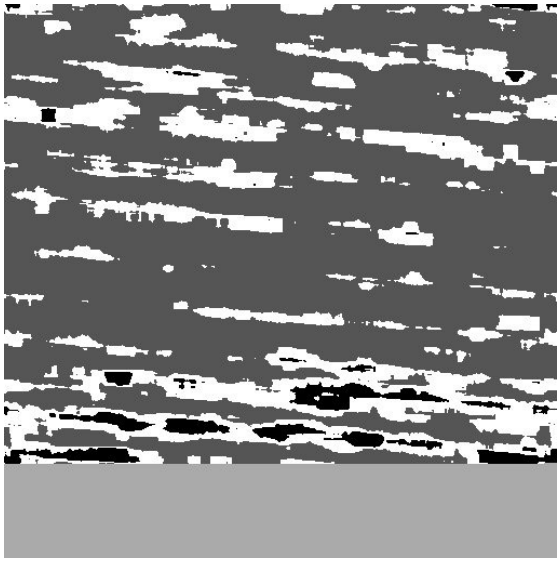




The process of image segmentation is outlined as follows in the homework:

1. Extract the Laws feature through each filter. 25 gray-scale images are generated from a single image.
   - In the code, this is done in the create_energy_mat() function.
2. Compute the neighborhood energy for each pixel in every Laws filter response image.
   - In the create_energy_mat() function, there is a getEnergyOfPixelsInWindow() function that takes in the gray-scale image of the Laws filter response.
3. Normalize every pixel by its $L5^T$ L5 value
   - This is implemented in the normalize_energy() function.
4. Segment the image and cluster the pixels based on the feature_vector at every pixel.
   - This is achieved by the k_means clustering function in the main function.
5. Output an image by writing a grayscale pixel to a file with an intensity that corresponds to the cluster.

For an unknown reason, the written texture segmentation code does not achieve good results. Likely, it is to do with the normalization process, or the way the k_means algorithm is called. Or maybe a crucial step that was not considered that will significantly affect the performance of the algorithm. The results that are achieved with the code are shown below.



As seen from the output images, the results are less than desirable. However, an erroneous output is better than no output and there was no lack in effort from trying very hard to figure out what went wrong :)

Problem 2: Salient Point Descriptors and Image Matching

Part a) Extraction and Description of Salient Points

Since the input images are color images, there are three channels for each image and therefore three different feature descriptions. A fourth exists by loading images as a grayscale and extracting salient points from this image. The SIFT descriptors for the four different channels are shown below for the bus image.

SIFT features for blue channel for bus image.

SIFT features for green channel for bus image.

SIFT features for green channel for bus image.

SIFT features for grayscale bus image.

Below are the SURF descriptors for the bus image for each channel same as above.

SURF features for blue channel for bus image.

SURF features for green channel for bus image.

SURF features for red channel for bus image.

SURF features for grayscale bus image.

The same descriptor images are shown below for the jeep image. First the SIFT features.

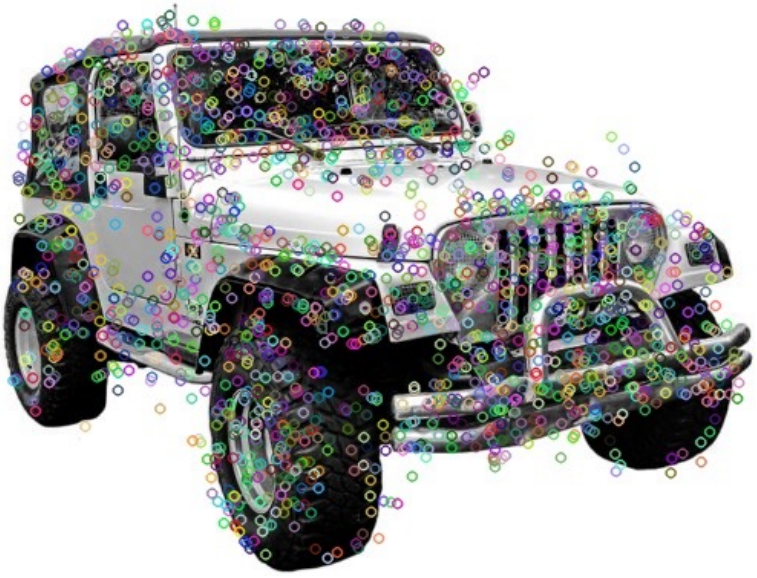
SIFT features for blue channel for jeep image.

SIFT features for green channel for jeep image.

SIFT features for red channel for jeep image.

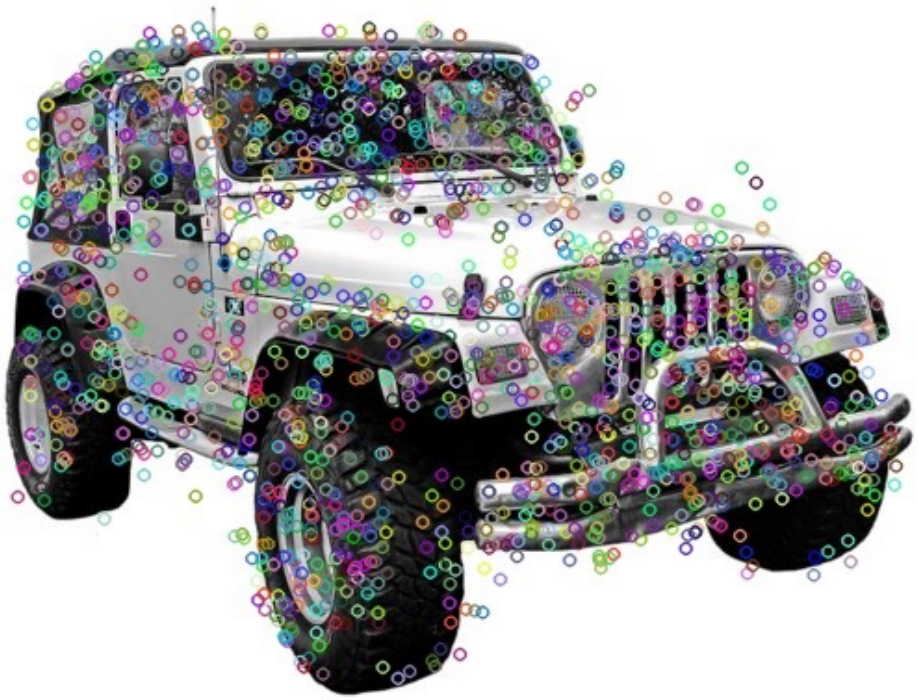
SIFT features for grayscale jeep image.

The following are the jeep SURF features.

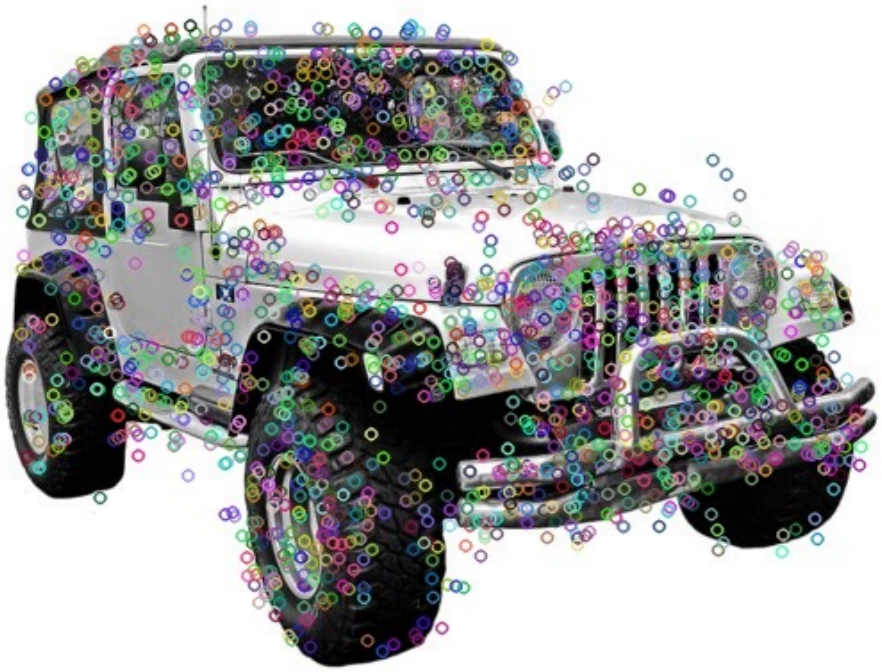


SURF features for blue channel for jeep image.

SURF features for green channel for jeep image.



SURF features for grayscale jeep image.



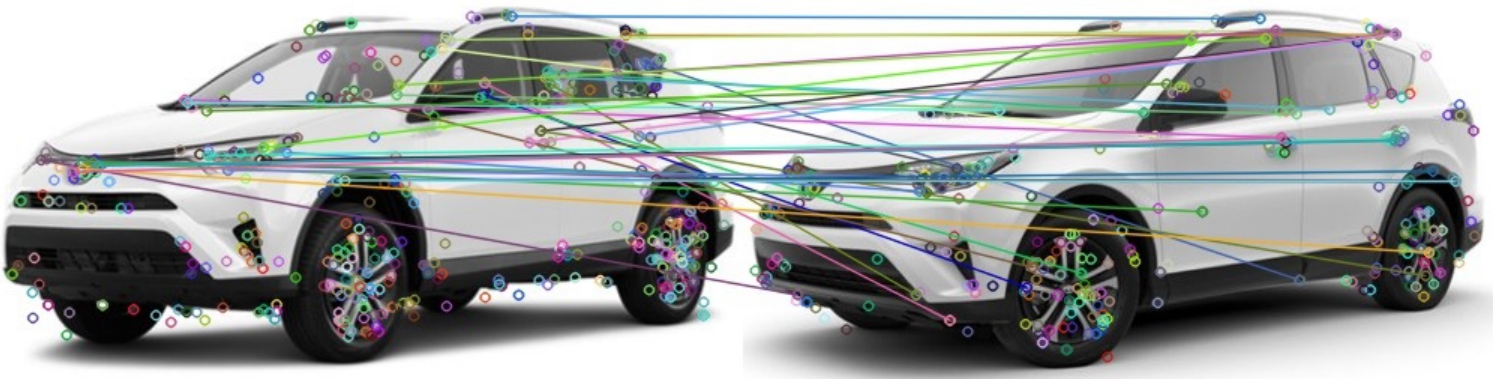
SURF features for red channel for jeep image.

Using the openCV's built in SURF and SIFT functions, the results are quite different. Most noticeably, there is a far larger number of SURF feature descriptors than SIFT feature descriptors. Some of the SURF descriptors also are not descriptive salient points. For instance, the jeep SURF features have some salient points in the background, while this occurs less in SIFT. However, SURF is a faster process and its large number of descriptors may turn out to be an advantage in describing images with richer detail.

Part b) Image Matching

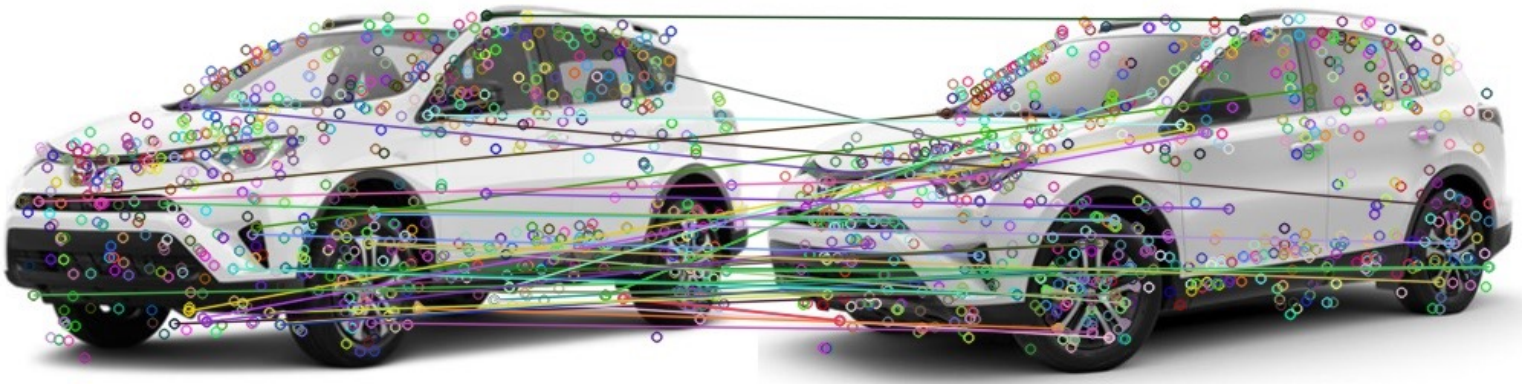
For image matching, SIFT/SURF features are extracted from two separate images and then compared with their neighborhood distances to find the closest salient points. These points are then matched together.

Some of the SIFT features are shown and matched in the picture below between rav4_1.jpg and rav4_2.jpg



SIFT feature matching for rav4_1.jpg and rav4_2.jpg

The same matching procedure is done for the SURF features show below for the same pair of images.

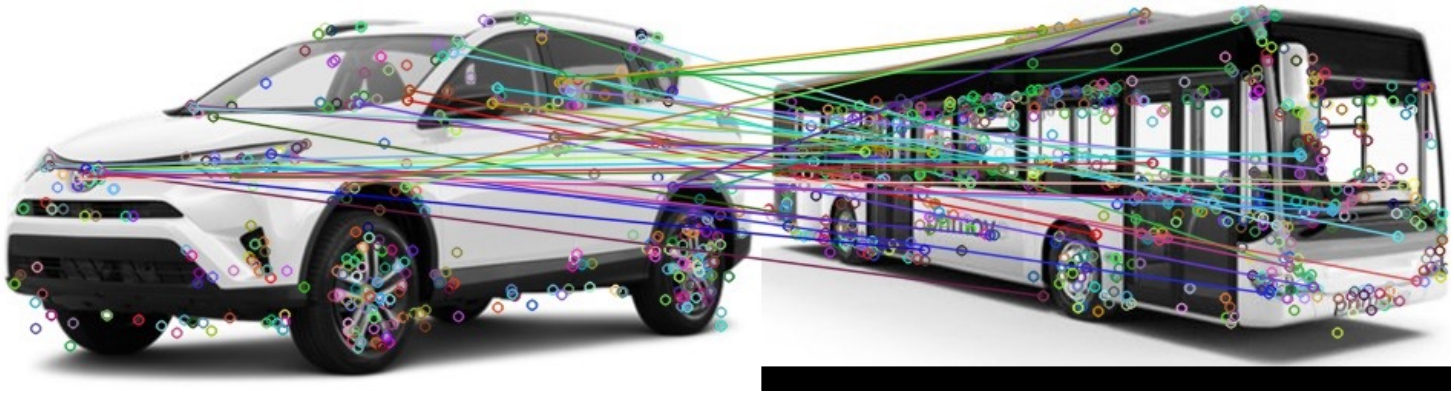


SURF feature matching for rav4_1.jpg and rav4_2.jpg

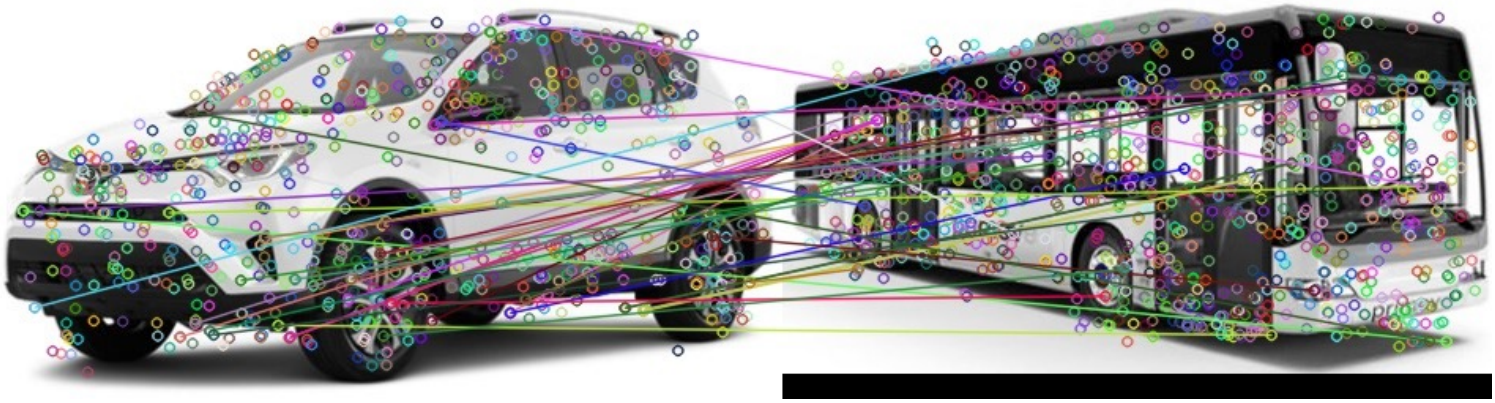
Because there is a large number of features, only a small number of features are actually shown to be matched for visibility. The general matching process is relatively accurate, though there are a number of features that are mismatched. It is difficult to match features that have very precise local information. This is why descriptors on the door handles are misclassified as well as descriptors located on the headlights. These are ambiguous descriptors because their neighborhood information is very similar. Compared with descriptors located on the wheels,

these are more accurately classified (by the SURF, at least) because of its darker color, which is in contrast with the white body of the car. This also causes misclassification as seen in the SIFT descriptors, which matches the wheels to the darker sections of the car windows.

Below is the same matching procedure done for different images; first is for rav4_1.jpg with bus.jpg, second is rav4_1.jpg with jeep.jpg.

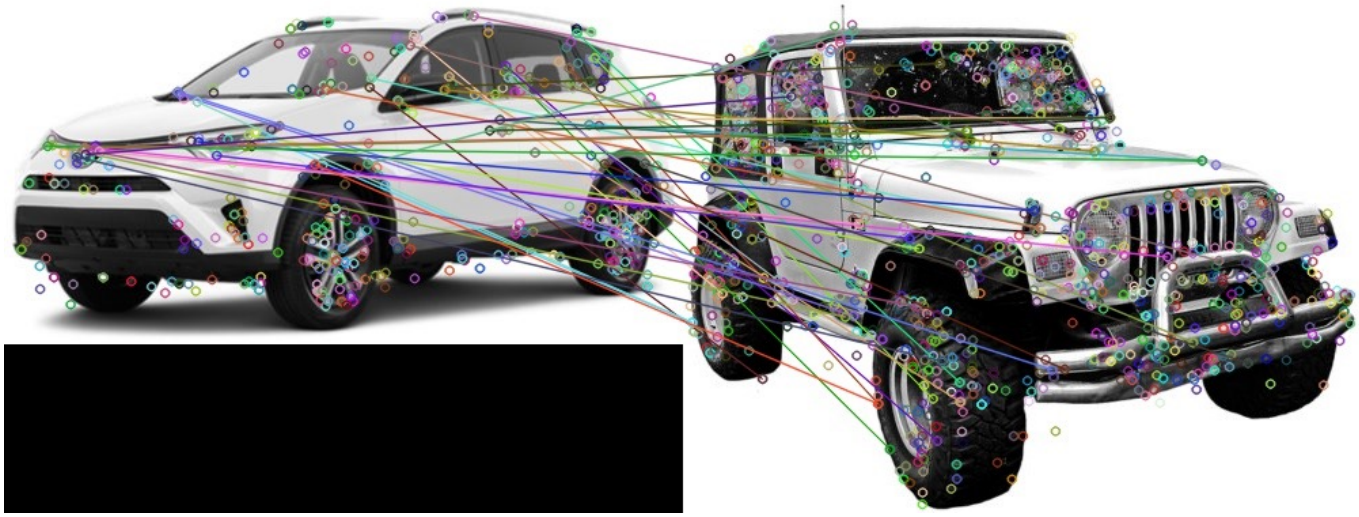


SIFT feature matching for rav4_1.jpg and bus.jpg

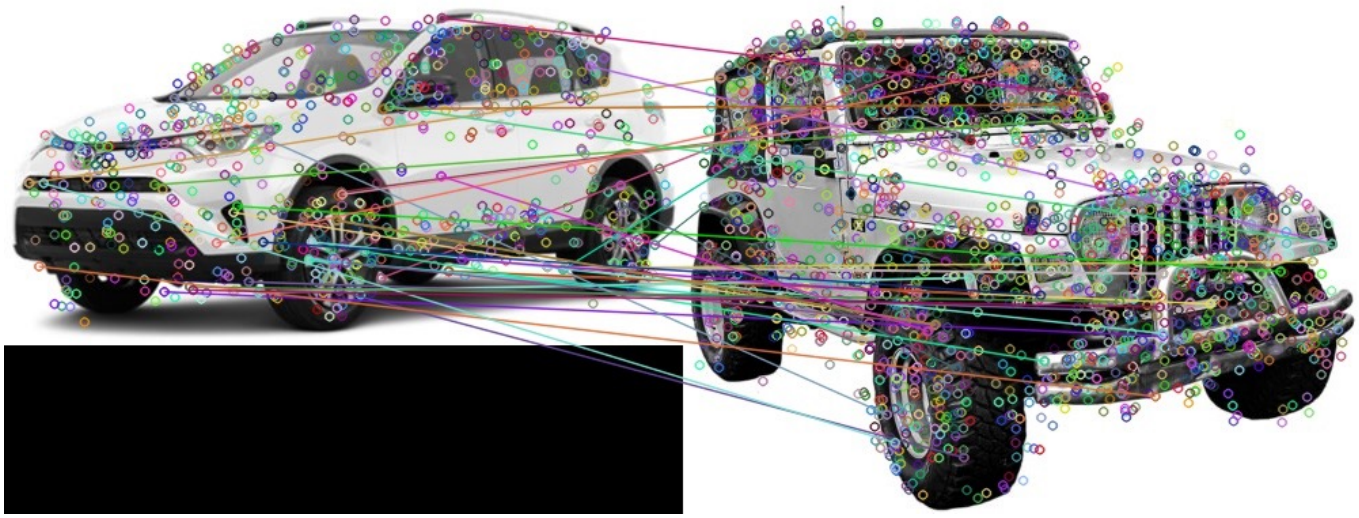


SURF feature matching for rav4_1.jpg and bus.jpg

Needless to say, the feature description matching is unlikely to make much sense because there is simply very little similarities between the two images. This applies for both the bus and jeep images. Although there does appear to be some well-performing matches such as a wheel descriptor matching a wheel descriptor or window to window. It is difficult to say whether to not this is a ‘lucky’ hit that would result from random matching. Further analysis would have to be made. The intuitive understanding is that these matches are simply lucky hits.



SIFT feature matching for rav4_1.jpg and jeep.jpg



SURF feature matching for rav4_1.jpg and jeep.jpg

Part c) Bag of Words

As a way to match test images to a set of training images, one could employ the Bag of Words model. This model is a histogram based SIFT feature clustering method. The model is trained as follows. First, extract the SIFT features for each

training image. Group the SIFT features into 8 clusters. Create a histogram by counting the number of features in each cluster. A histogram is created for each image for specific SIFT feature clusters. The Bag of Words Model is the combination of all these histograms. In this example, the training images that are used are rav4_1.jpg, jeep.jpg and bus.jpg.

Testing an image requires a comparison with every histogram that is created. Create a histogram for the 8 SIFT feature clusters for the test image. In this example the test image is rav4_1.jpg. L2 distance is used to find the closest match in histograms. The table below shows the SIFT feature histograms for each image.

SIFT Feature Cluster #	Jeep.jpg	Bus.jpg	rav4_1.jpg	rav4_2.jpg
1	89	77	43	39
2	156	79	38	74
3	101	112	81	25
4	129	90	54	7
5	88	85	63	28
6	82	88	70	82
7	155	97	53	33

SIFT Feature Cluster #	Jeep.jpg	Bus.jpg	rav4_1.jpg	rav4_2.jpg
8	117	79	17	12

L2 distance (or Euclidean distance) is calculated by summing the square difference for each value. In this case, all the values in the rav4_2.jpg column are compared with the values for every other column and summarized in the table below.

Image	Jeep.jpg	Bus.jpg	rav4_1.jpg
L2 Distance	243.7	166.7	91.9

From the table, it shows that rav4_2 is much closer to rav4_1 than the other images, which is to be expected. This is confirmation that the Bag of Words model works well for these images.

Problem 3: Edge Detection

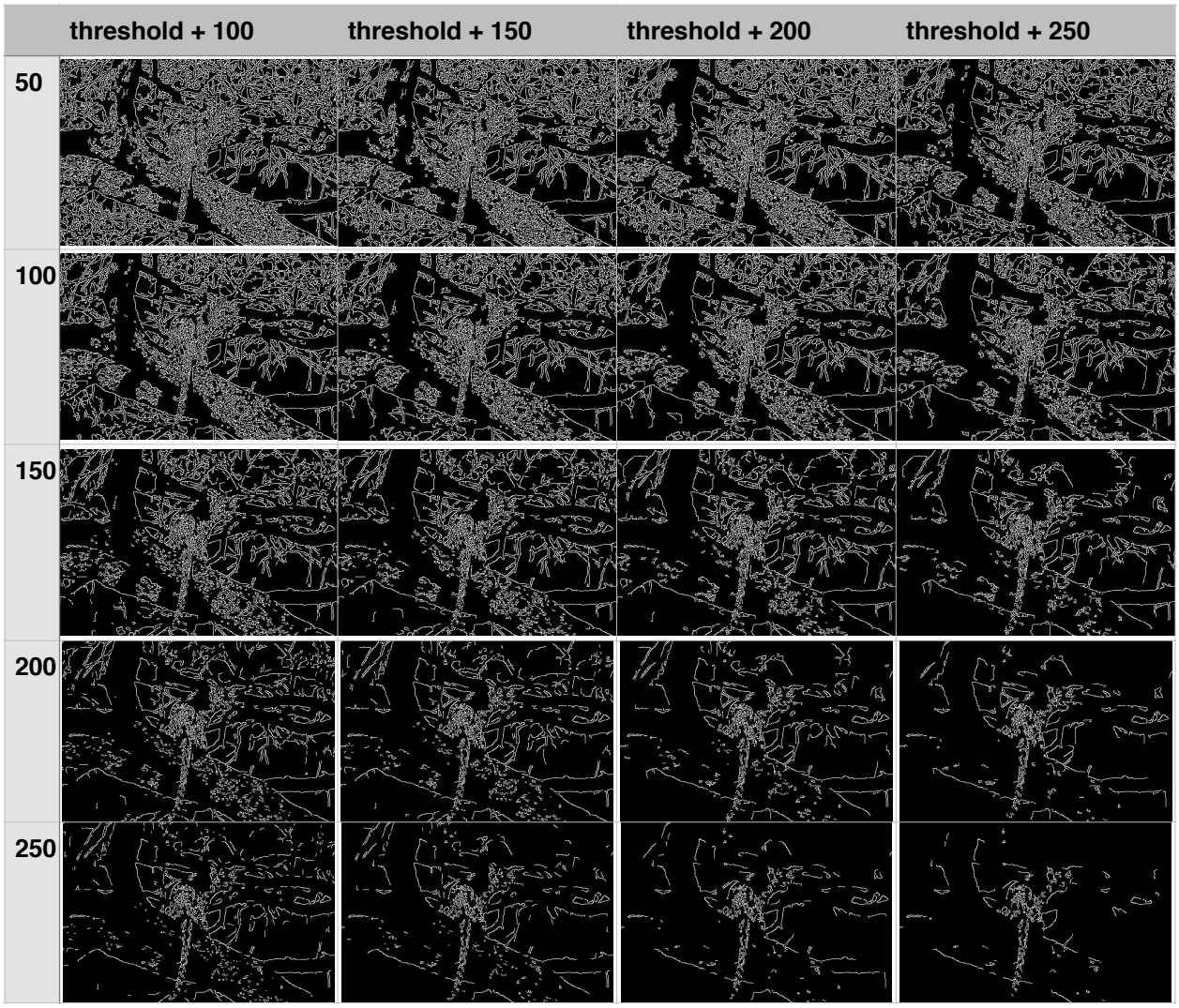
Part a) Canny Edge Detector

For the following problem, the Canny edge detection method is used to find the edges of the following two images. The jaguar image and the zebra image.

Canny edge detection is a multi-step process of edge detection. The process can be broken down in the following steps.

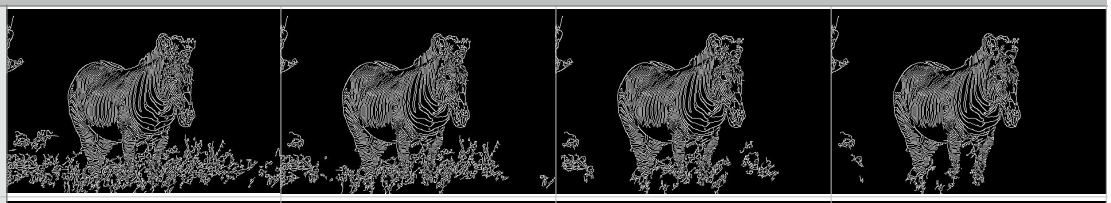
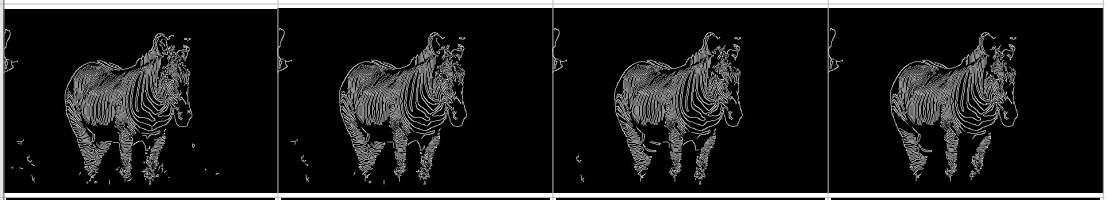
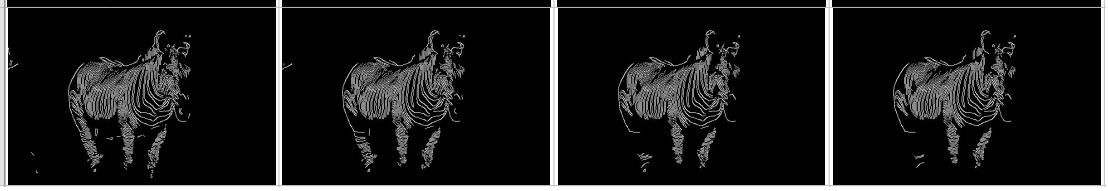
1. Apply Gaussian filtering to smooth the image and remove noise
2. Find gradients of image pixels
3. Use non-maximum suppression
4. Use the double-thresholding technique. The edges are separated into weak and strong edges based on the threshold values.
5. Connect the edges by hysteresis. Suppress weak edges and connect strong edges.

The OpenCV Canny edge detector is used to detect the edges for the Jaguar and Zebra images. The results are summarized in the tables below. Different low and high threshold values are used to explore their effects.



Due to the nature of the image, the Jaguar edge detection performance is quite poor. This is because in the image, there is a lot of gradient intensity differences. There are a lot of edges that separate background and foreground as well as ambiguous backgrounds – there is the sky and then there are the tree branches. That being said, however, the threshold values that give the best Canny edge detection performance is the one with 200 and 400. With these threshold values, the trees and branches can be clearly separated and with careful inspection and comparison, the jaguar itself can also be identified. Higher threshold values suppress details about the jaguar object while lower threshold values detect too many edges associated with the tree branches.

In comparison with the Jaguar image, the Zebra image is much more suitable for edge detection. Even for lower and higher threshold values, the object is clearly distinguishable.

threshold	threshold + 100	threshold + 150	threshold + 200	threshold + 250
50				
100				
150				
200				
250				

It is difficult to tune the parameters to show only the edges of the object. Although it is possible for the Zebra image, it is extremely difficult for the Jaguar image. Even for the Zebra image, if threshold values are chosen to show edges for only the object, there are some details on the object that have been suppressed.

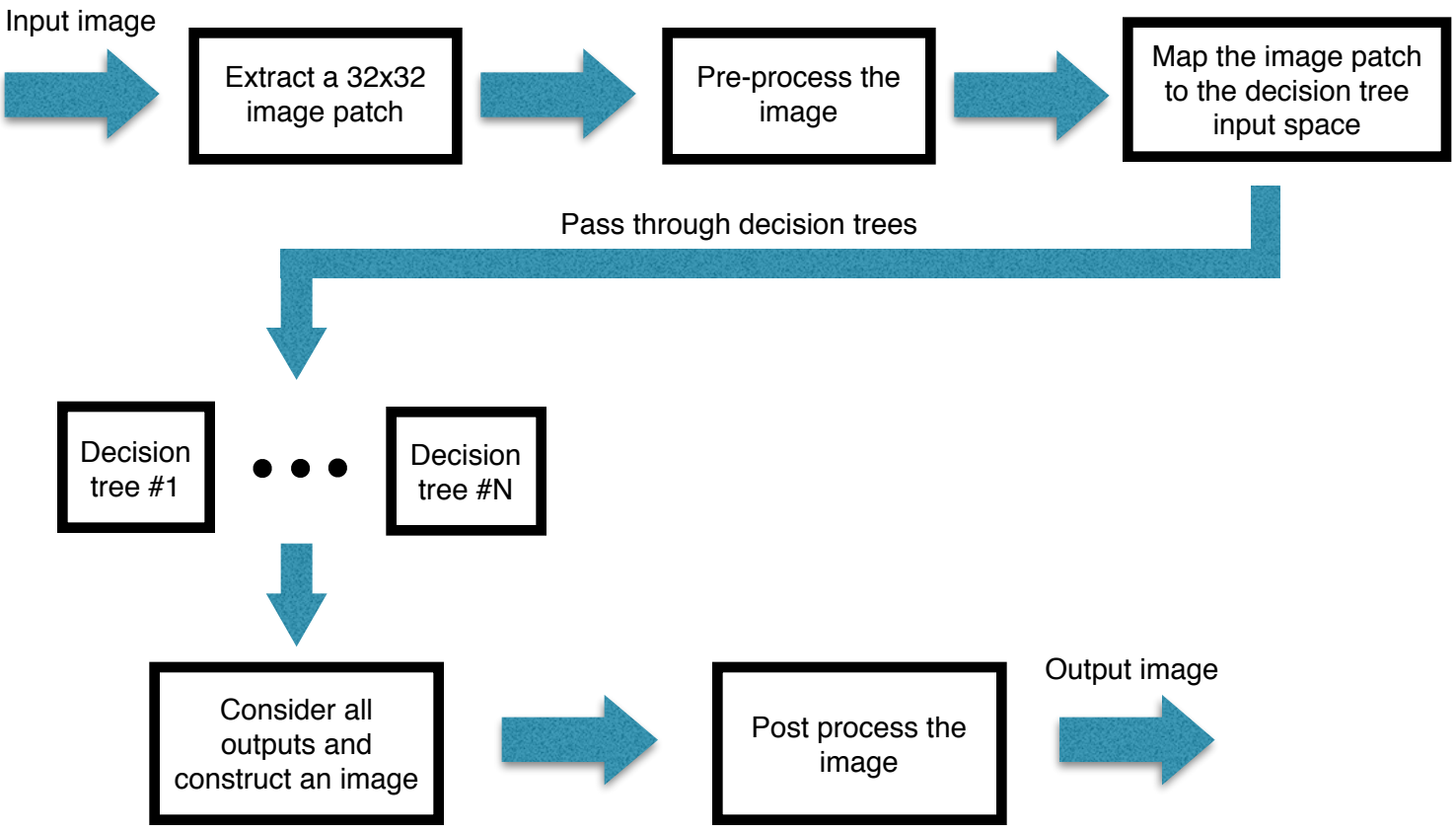
Part b) Structured Edge

Fast Edge Detection using Structured Forests is a new method of edge detection that uses random forests to detect edges in an image. Since this method uses a machine learning modeling technique, training data must be specified and used in order to create a reliable model.

In the paper, the Berkeley BSDS500 segmentation dataset is used to train the model. Random forests is a model that uses numerous separately trained decision trees and considers the aggregate output of each decision tree. The decision trees are 'grown' by choosing a threshold value that can usefully and efficiently separate the training data. The trees are grown until maximum depth is reached or unless information gain meets a threshold or if training set size meets a threshold. This process is repeated for multiple decision trees by randomly subsampling the training set.

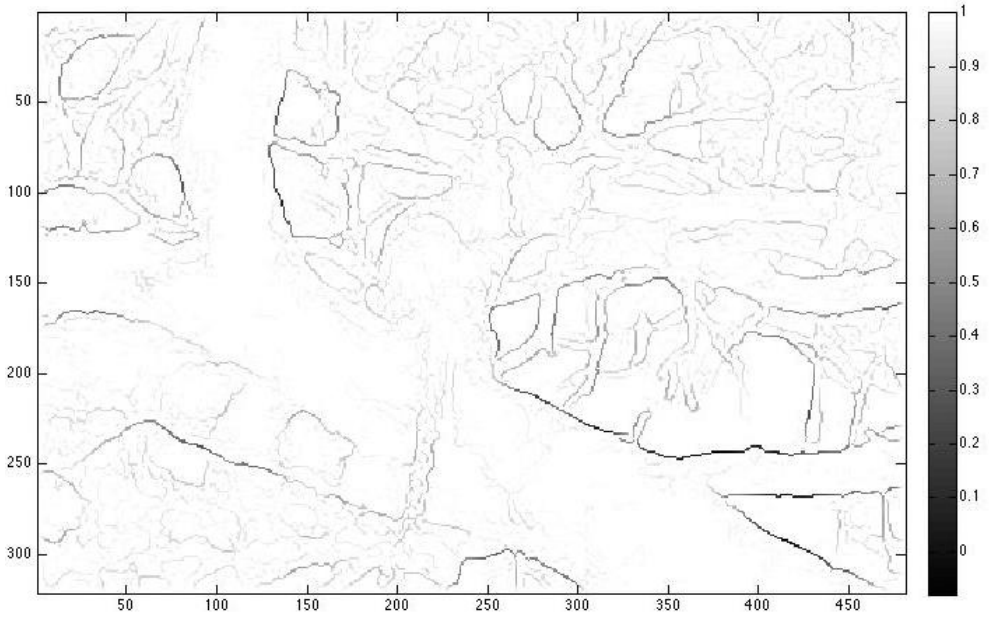
Once the decision trees are trained, the model is able to used for edge detection. An edge is detected in the following process. First, take a 32x32 image patch from the target image. Extract the features by multiple preprocessing techniques. Map the patch to another space that is suitable for the decision tree and has good capability for Euclidean distance. Repeat for all decision trees. Consider and aggregate the output for all the decision trees. Optionally, employ post-processing, namely multiscale detection and/or edge sharpening.

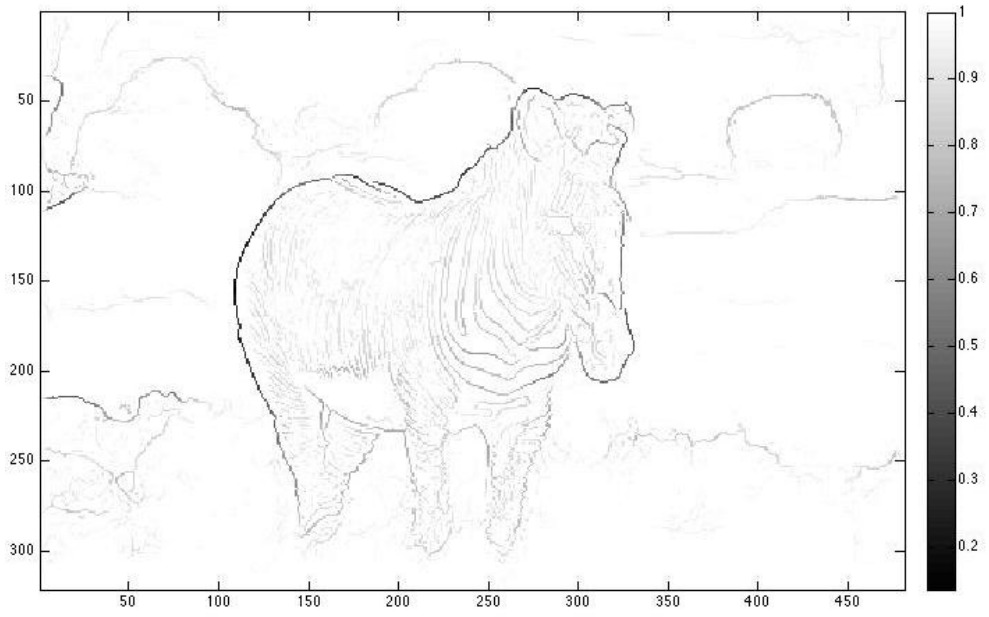
This process is summarized in the below flow chart.



The principle of the random forest classifier is one that avoids overfitting. Training a single decision tree tends to cause overfitting. It has been shown that a decision tree that has lower accuracy tends to give better performance when used with multiple other decision trees. The same idea is used to account for a more general information gain criterion. Approximate edge labels are desired such that ‘similar’ edge labels can then be classified and discriminated. From here, if multiple decision trees give the same ‘similar’ edge label, then one can be more confident that that is a good edge label.

Below are the edge labels generated for the Zebra and Jaguar images using the Structured Edge process as provided by the software package created by Piotr Dollar and C. Lawrence Zitnick.





References

Dollar, Piotr and C. Lawrence Zitnick. "Structured Forests for Fast Edge Detection." ICCV, 2013.