Homework 3 Report

// Date : 20 March 2018
// Name : Tamoghna Chattopadhyay
// ID : 8541324935
// email : tchattop@usc.edu

Problem 1.(a)

I. Abstract and Motivation

In this problem we are to perform texture classification using nine 5x5 Laws Filter. Textures are basically patterns which repeat periodically and can be classified by extracting features from the image. There are two main tasks in this problem: Feature Extraction and Classification. For classification, K means algorithm is utilized.

II. Approach and Procedures

Textures usually follow statistical patterns which can be extracted as feature vectors. Here, in this problem we are given 12 textures which we need to classify. The first task is that the image is padded with extra rows and columns so that filter can be properly applied to it. Then the tensor product of the 1D arrays are done to get the Law Filters. Here we use 9 of the Law filters formed by

$$E5 = (1/6)*[-1\ -2\ 0\ 2\ 1],\ \ S5 = (1/4)*[-1\ 0\ 2\ 0\ -1],\ \ W5 = (1/6)*[-1\ 2\ 0\ -2\ 1]$$

After calculating the tensor products, we get the following 9 filters:

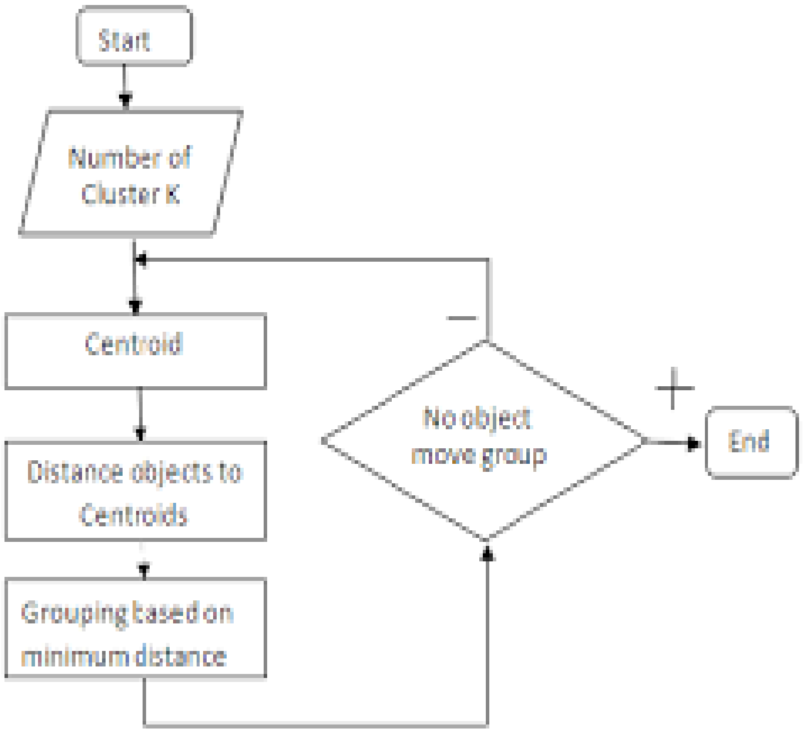
```
E5E5[25]=  {-1,-2,0,2,1,0,0,0,0,0,2,4,0,-4,-2,0,0,0,0,0,-1,-2,0,2,1} * 1/36
E5S5[25]=  {-1, 0,2,0,-1,-2,0,4,0,-2,0,0,0,0,0,2,0,-4,0,2,1,0,-2,0,1} * 1/24
E5W5[25]= {1,-2,0,2,-1,2,-4,0,4,-2,0,0,0,0,0,-2,4,0,-4,2,-1,2,0,-2,1} * 1/36
S5E5[25]=  {-1,-2,0,2,1,0,0,0,0,0,2,4,0,-4,-2,0,0,0,0,0,-1,-2,0,2,1} * 1/24
S5S5[25]=  {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} * 1/16
S5W5[25]= {-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} * 1/24
W5E5[25]= {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} * 1/36
W5S5[25]= {-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} * 1/24
W5W5[25]= {-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} * 1/36
```

Then the DC component in the image is removed from the entire image as it hardly provides any important information. Only the AC components contribute to the feature vector calculation. Then the above nine filters formed are convolved with the padded image formed and a 9D feature set is created. The total energy of each of these 9 vectors are calculated using the formula:

$$Energy\ per\ image = \frac{1}{Width * Height} * \sum_{i=0}^{height} \sum_{j=0}^{width} I(i,j) * I(i,j)$$

Thus we get 9 different energies for each image which comprises the feature vector. So we generate a 12x9 feature vector set for all images combined. The feature vector is then normalized to between 0 and 1. This process is called feature extraction.

The next step is K-means clustering. It is carried out with k=4 as we have 4 centroids related to the 4 classes of textures. The K-means algorithm is given as follows:


Start
Number of Cluster K
Centroid
Distance objects to Centroids
Grouping based on minimum distance
No object move group
−
+
End

So basically we initialize 4 random centroids. Then we calculate the distance of each feature vector from these centroid and group the textures based on the minimum distance from the centroid. We continue iteration through the loop till no texture changes it's class in a loop. This way, the textures are classified into four separate classes.

III. Results and Discussion

The feature vectors obtained for the 12 images are:

0.87409	0.47652	0.15645	0.46488	0.30055	0.11066	0.13473	0.09731	0.03857
0.11846	0.07882	0.02367	0.05825	0.04250	0.01429	0.01434	0.01133	0.00412
0.06393	0.02636	0.00674	0.03216	0.01392	0.00360	0.00976	0.00411	0.00078
1	0.55628	0.17445	0.57913	0.35876	0.12188	0.16866	0.11671	0.04262
0.08397	0.03225	0.00703	0.04196	0.01643	0.00347	0.01181	0.00459	0.00063
0.93768	0.53822	0.17750	0.51882	0.3402	0.12611	0.15649	0.11449	0.04560
0.00601	0.0053	0.00244	0.005335	0.005169	0.002503	0.002431	0.002538	0.001149
0.00660	0.00536	0.00253	0.00537	0.00527	0.00265	0.00236	0.00255	0.00121
0.07025	0.03591	0.00801	0.03253	0.01872	0.00443	0.00847	0.00522	0.00104
0.00545	0.00331	0.00061	0.00431	0.00299	0.00058	0.00167	0.00117	1.86E-05
0.04400	0.01868	0.00509	0.01854	0.00763	0.00196	0.00368	0.00129	0
0.13319	0.09246	0.02986	0.07083	0.05923	0.02232	0.02143	0.02007	0.00858

Looking at the frequency response of the different Laws Filters, we can understand the contribution of each feature.

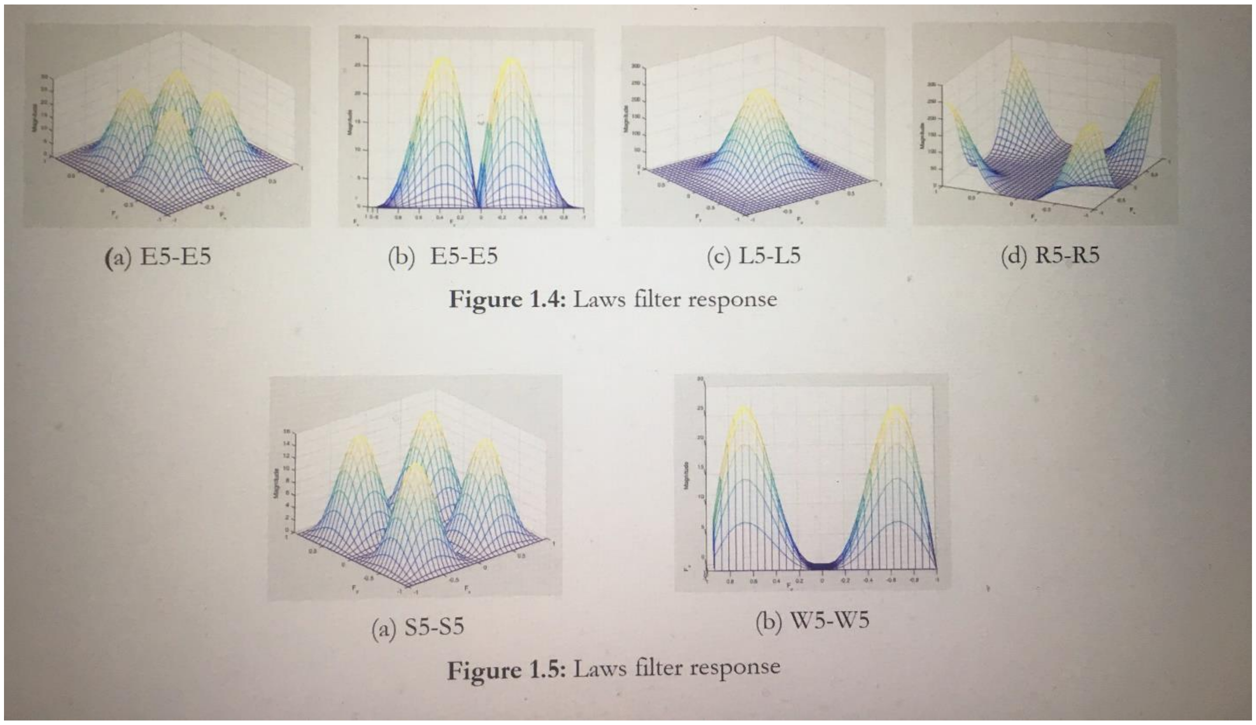

(a) E5-E5 (b) E5-E5 (c) L5-L5 (d) R5-R5

Figure 1.4: Laws filter response

(a) S5-S5 (b) W5-W5

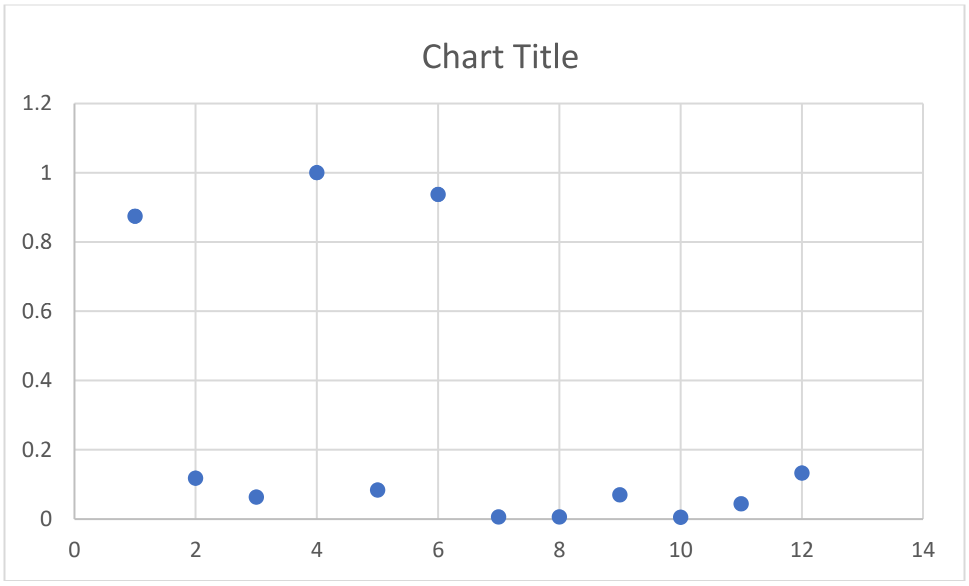
Figure 1.5: Laws filter response

From the filter responses, we can say that:

- L5 – Low Pass Filter with cut off frequency at 0.5
- E5 – Band Pass Filter with lower cut off at 0.1 and upper cut off at 0.7
- S5 – Band Pass Filter with lower cut off at 0.2 and upper cut off at 0.8
- W5 – Band Pass Filter with lower cut off at 0.3 and upper cut off at 0.9
- R5 – High Pass Filter with cut off frequency at 0.5

When these filters are convolved with the input image, different frequency components from the different axis are captured. The features are calculated based on the filter response over the entire image. The energy value will be high for a filter response whose frequencies are suitably comparable to the frequencies present in the texture image.

For comparing the discriminant power of features, we can plot each feature for every image and check whether the values overlap, i.e., if they are same in value for different classes. The more the feature can separate out the classes, the more discriminant power is of that feature. And vice versa. So for example,



The above chart shows variation for feature 1 for the 12 images. As we can see, the feature can easily separate Class 1 from the rest but it may find it difficult to separate the other classes as they are almost same value. So it has less discriminant power than maybe a feature which can classify all the other classes correctly. In the given dataset of texture images, it is very difficult to find any single feature which can perfectly classify all images into classes correctly. Most of the features, when I went through their scatter plot, seem to classify class 1 perfectly but will face a problem when classifying the other classes as their values are quite close to each other.

The K-Means classification gives the following result:

Final Class Labels:
Images belong to Class 1: 1 4 6
Images belong to Class 2: 2 12
Images belong to Class 3: 3 5 9 11
Images belong to Class 4: 7 8 10

When comparing with test with reality, I saw that one image – texture9 has been misclassified into class 3 when it belongs with class 2. The reason behind this is that the values of the feature vector for it lie very close to the centroid of class 3 despite it belonging to class 2. So the K-Means classifier does one misclassification on the dataset of texture images.

Problem 1.(b)

I. Abstract and Motivation

In this problem we are to perform texture segmentation using nine 3x3 Laws Filter. Textures are basically patterns which repeat periodically and can be classified by extracting features from the image. There are three main tasks in this problem: Feature Extraction, Classification and then Segmentation. For classification, K means algorithm is utilized.

II. Approach and Procedures

Textures usually follow statistical patterns which can be extracted as feature vectors. Here, in this problem we are given 12 textures which we need to classify. The first task is that the image is padded with extra rows and columns so that filter can be properly applied to it. Then the tensor product of the 1D arrays are done to get the Law Filters. Here we use 9 of the Law filters formed by

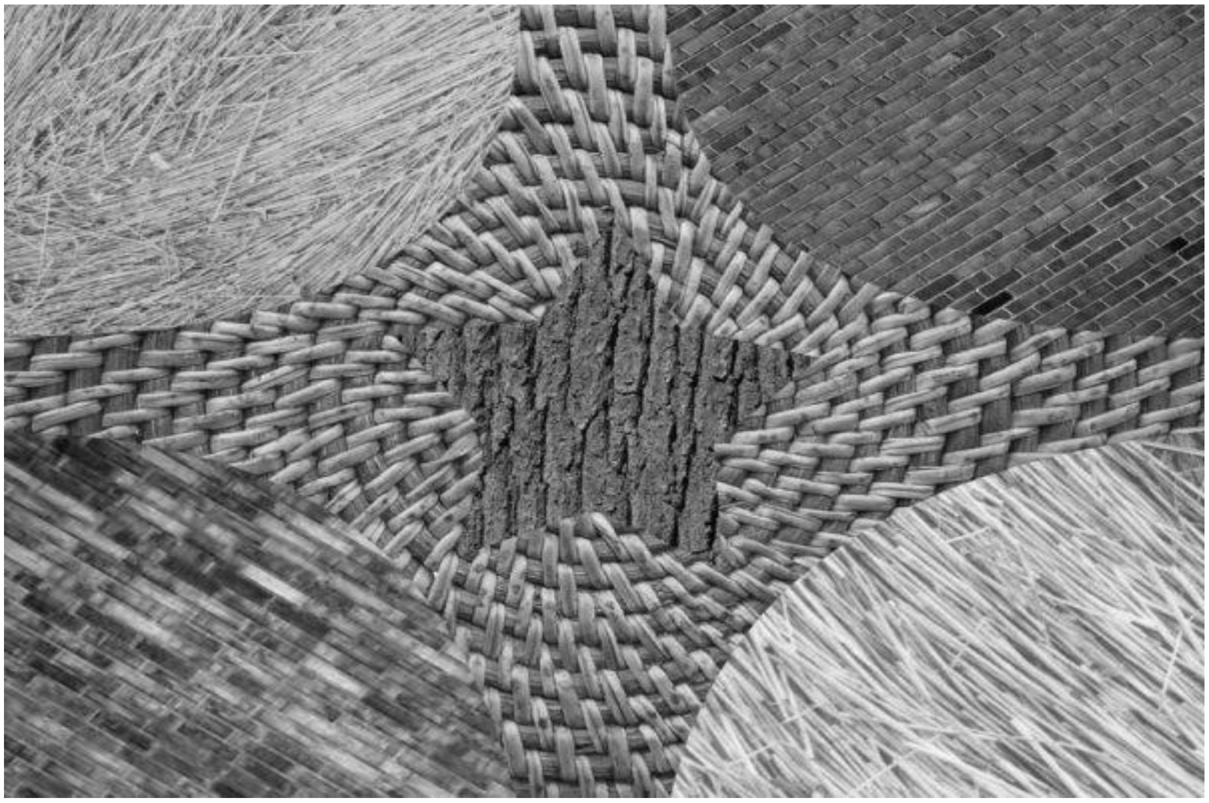
L3 = 1/6 * [1,2,1]   E3 = 1/2 * [-1,0,1]   S3 = 1/2 * [-1,2,-1]

The process for texture segmentation is very similar to texture classification. But instead of calculating global means of the entire image, we calculate the mean over small windows. Since texture is a local property, we obtain the energy per pixel and thus find the 9-D feature vector for each pixel. In this process we have to normalize by L3L3 filter. Then we apply the K-means clustering algorithm as discussed in previous question. Here, due to presence of six different textures in the image, we use 6 different centroids for the k-means algorithm. Once each pixel is given a label after the clustering algorithm is done, we can set a different color for each label. As there are six textures in image, we use six gray levels (0, 51, 102, 153, 204, 255) to denote six segmented regions in the output image.

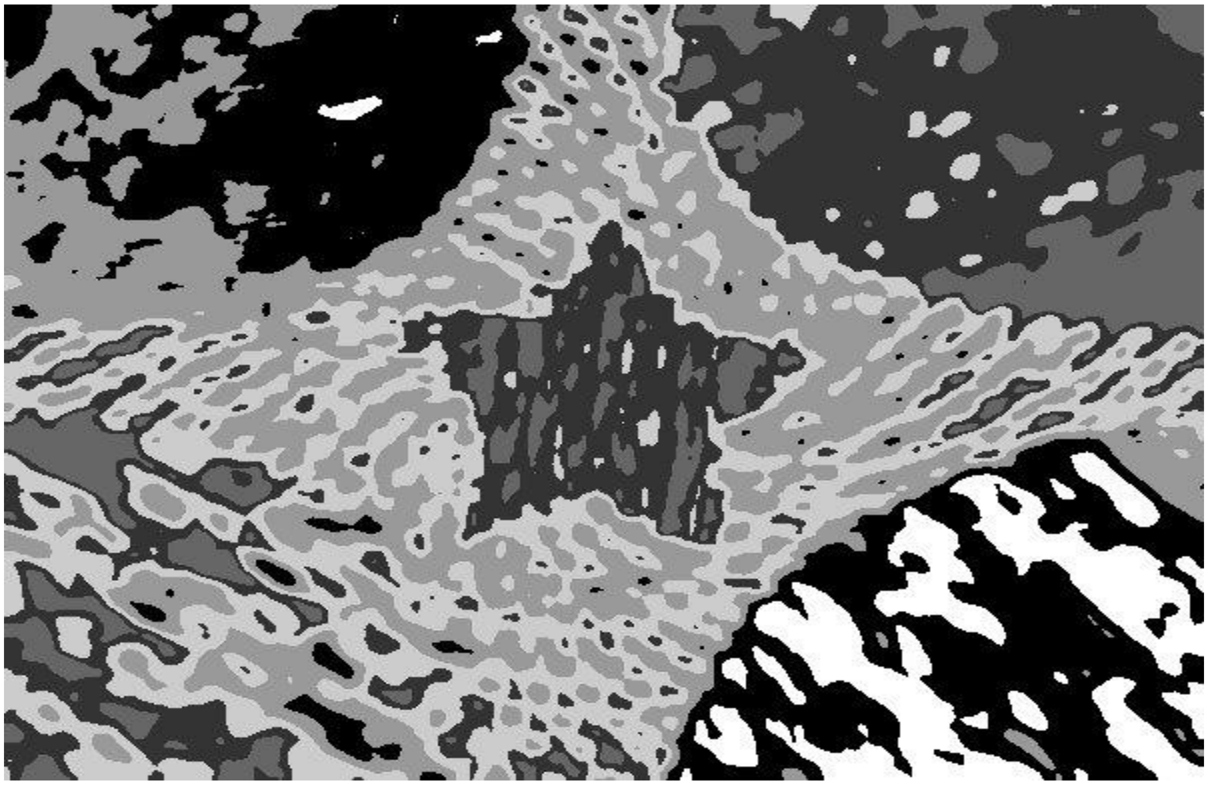
This way, the texture segmentation process takes place.

III. Results and Discussion

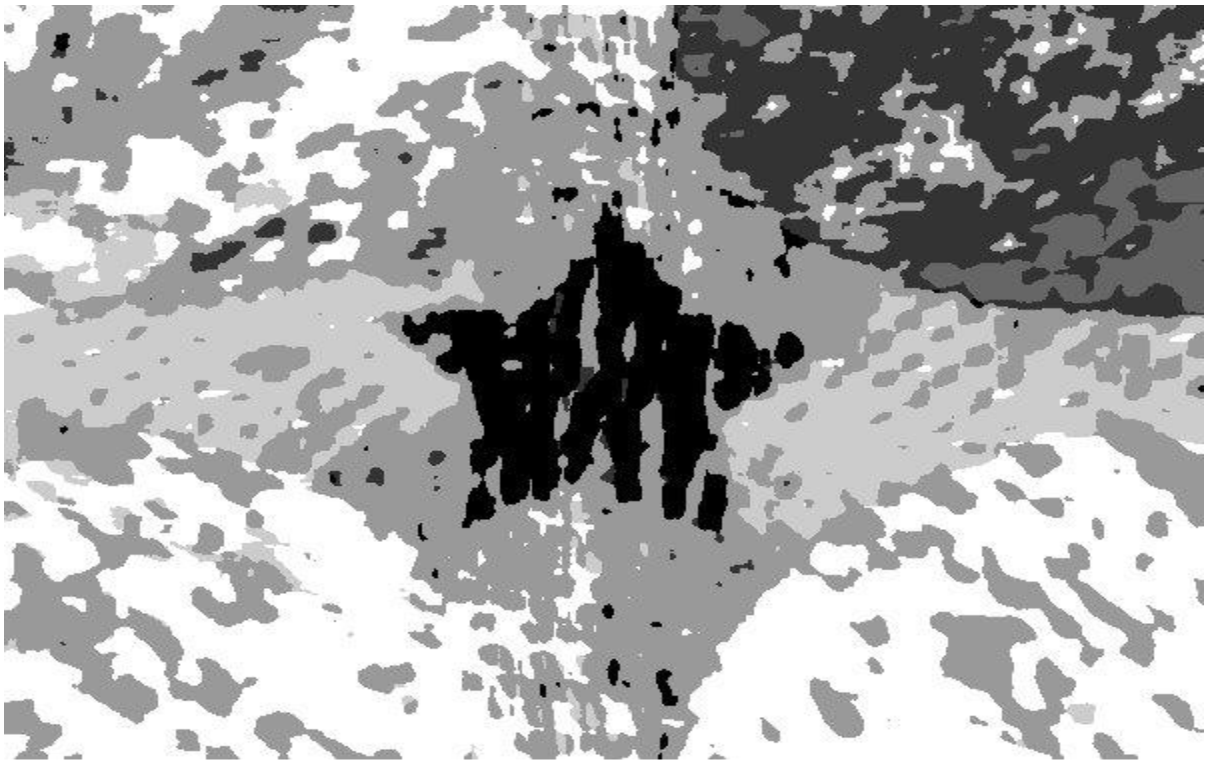
Original Image:



The image after segmentation process:



The segmented image after normalization with L3-L3 is done:



As can be seen from the output, though texture segmentation has been done, the result is not very clear. Textures which appear similar have been segmented in the same manner. Also as the window size while calculating the local means and energy is increased, the segmentation result gets somewhat better until an optimal point. Beyond that point, the image becomes blocky.

Problem 1.(c)

I. Abstract and Motivation

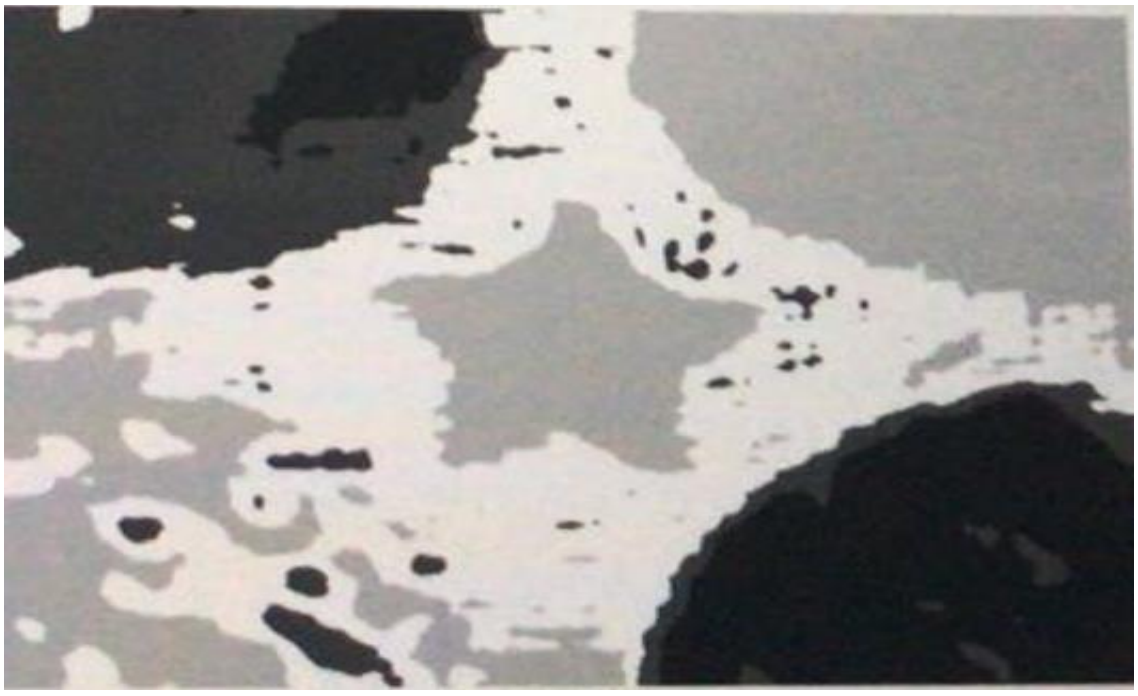
In this problem, we are to apply PCA for feature reduction and then do segmentation of the given image.

II. Approach and Procedures

To improve the segmentation output, PCA (Principal Component Analysis) is done. PCA can reduce the 25D feature space to nD space by considering only the most important Eigen Values by the Singular Value Decomposition technique of decomposition. PCA calculates the Eigen values and Eigen vectors of higher dimensional data space. Then it reduces it to n dimensions whose basis are the Eigen vectors corresponding to the n largest Eigen values. This is done as they contain the maximum information. It is important to calculate the PCA on a window size large enough so that local regions of textures are taken under consideration. PCA transforms all samples in the feature space, irrespective of the class labels.

Another method which can be used to reduce the dimensions of the feature vector is LDA. LDA, while converting to a lower dimension space, takes the class labels into consideration. This is the main difference between PCA and LDA.

III. Results and Discussion



Increasing the window size and reducing the dimension of feature vectors down to 5, it does improve the segmentation image a lot. Also, the k means algorithm runs a bit faster due to the smaller feature vector dimension, due to which the calculations are shortened.

Problem 2.(a).1

I. Abstract and Motivation

In this problem we were asked to implement the basic Edge detection technique on the images provided by using Sobel Filter. The RGB image is first converted to grayscale image and then Sobel filter is applied on the image to get the edges present in the images. In images, edges are defined by lines comprising points where the brightness changes sharply. They form the boundaries of different objects present in the image and present valuable information.

II. Approach and Procedures

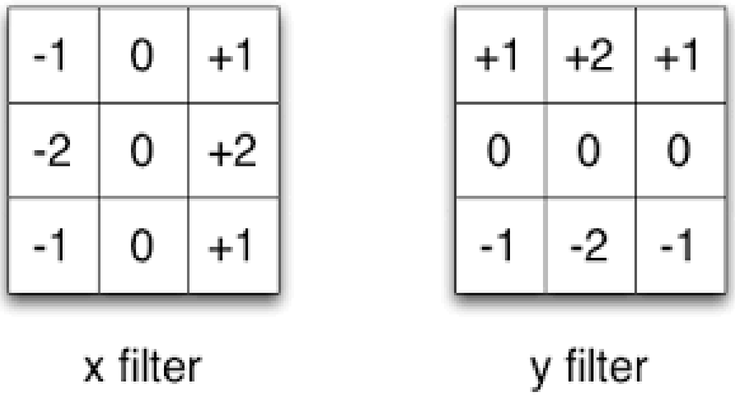
First, the RGB image is converted to Grayscale by using the formula:

$$\text{Gray} = 0.21*\text{Red} + 0.72*\text{Green} + 0.07*\text{Blue};$$

Then extra columns and rows are added or padded to the original image. Then we apply the Sobel filters which are 1st order derivatives. We calculate the derivative of each pixel in both the x and y direction. The formula is:

$$\text{Grad\_x} = 0.25*(A2 + 2*A3 + A4 - A0 – 2*A7 – A6)$$
$$\text{Grad\_y} = 0.25*(A0 + 2*A1 + A2 – A4 – 2*A5 – A6)$$

Where A are the values of the neighboring pixels. Then the gradient magnitude is given by taking the square root of the sum of squares of the above two sums. After getting the magnitude, the 10% of them with largest magnitude are picked as edges while the rest are discarded as background.



-1	0	+1
-2	0	+2
-1	0	+1

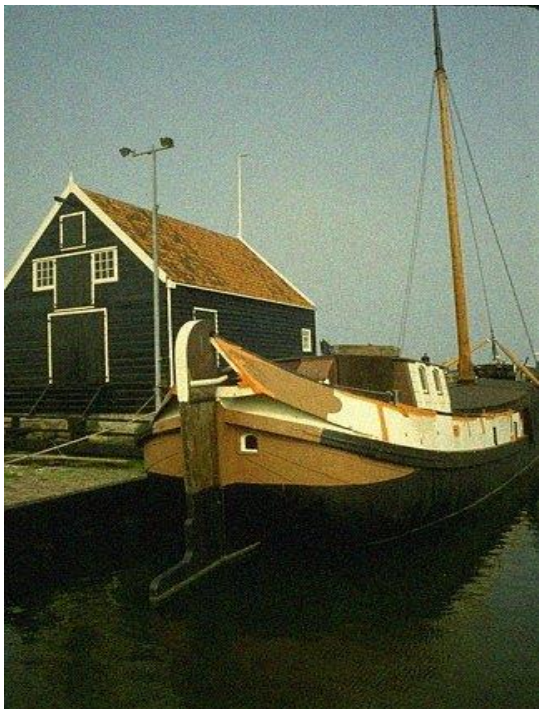
x filter

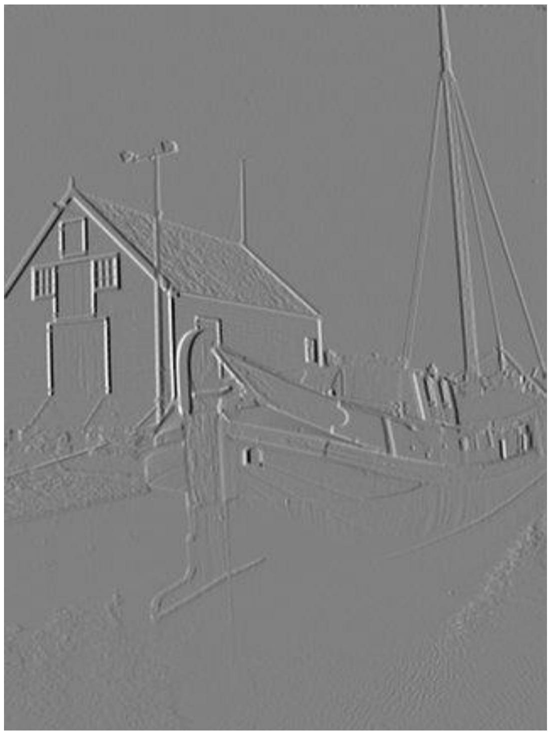
+1	+2	+1
0	0	0
-1	-2	-1

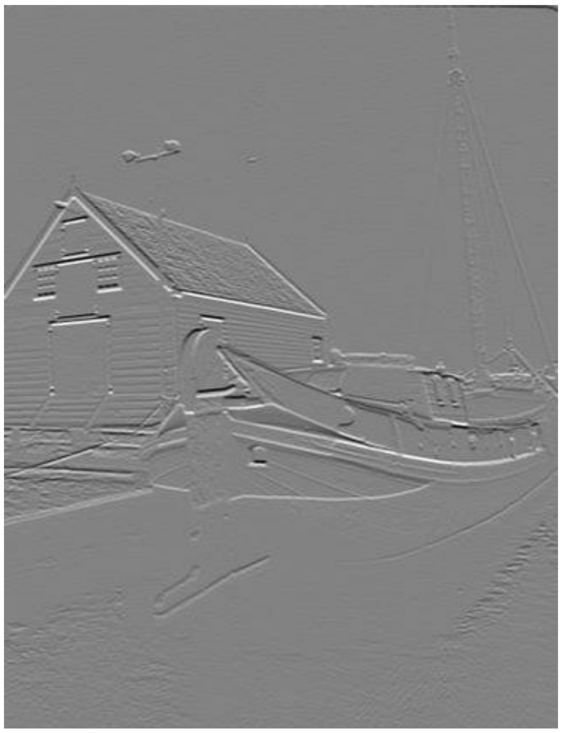
y filter

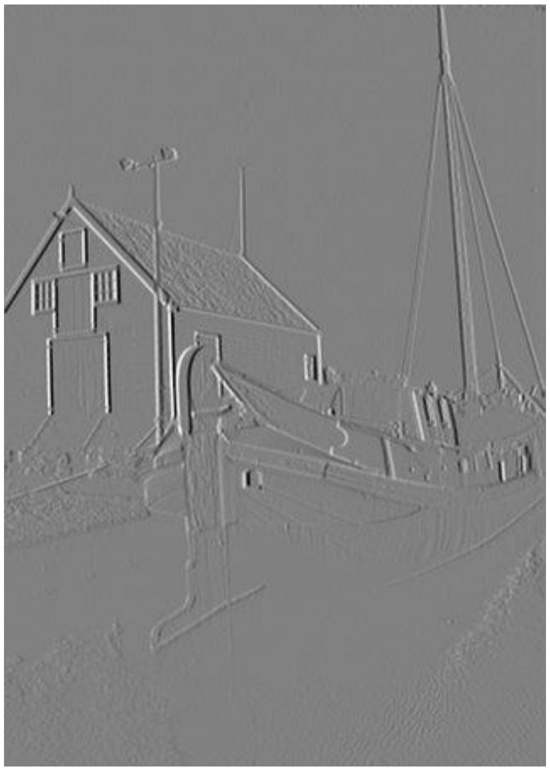
III. Results and Discussion

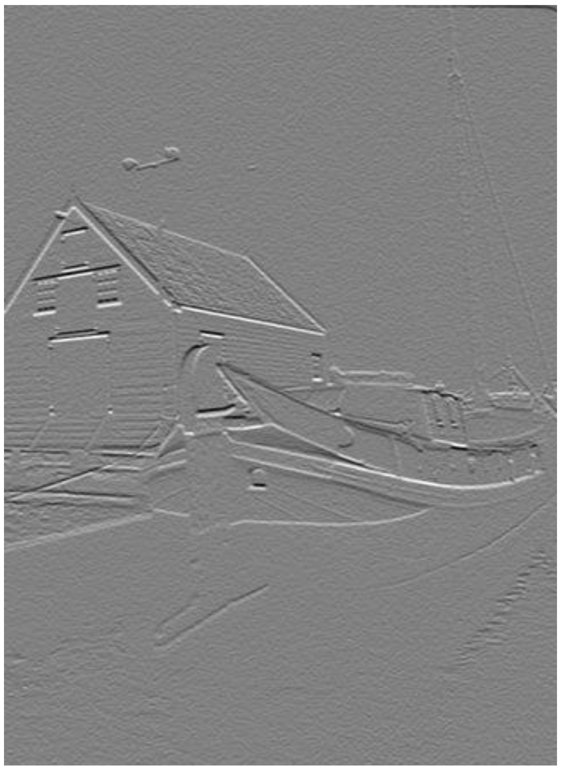
Main Image : Boat

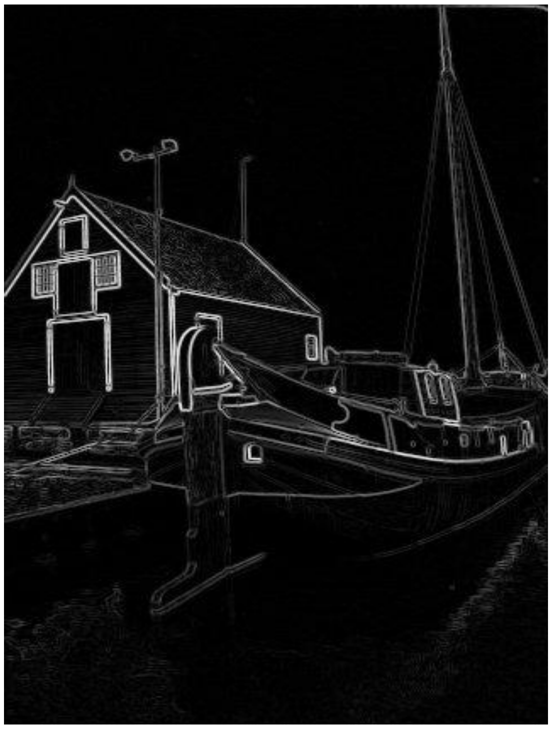

Main Image : Noisy Boat

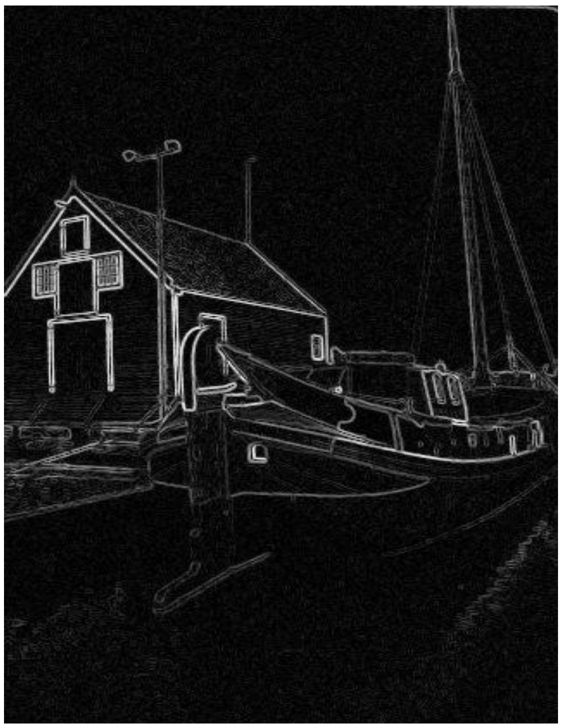

X gradient image after applying Sobel to Boat image

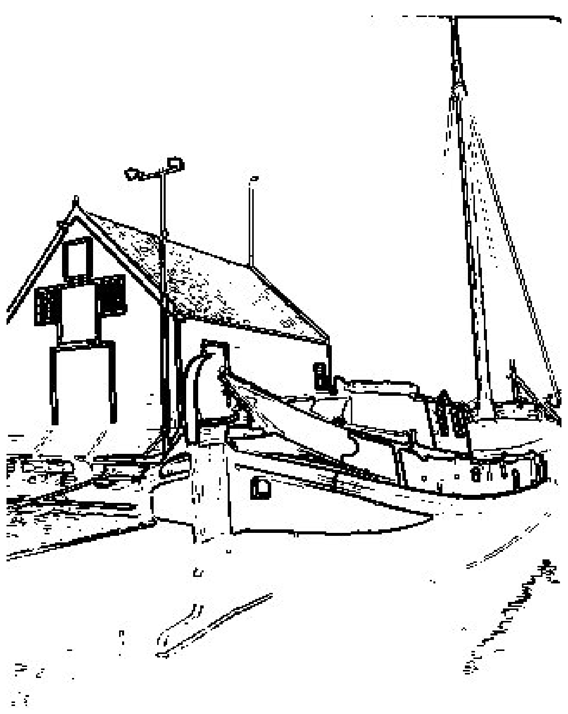

Y gradient image after applying Sobel to Boat image

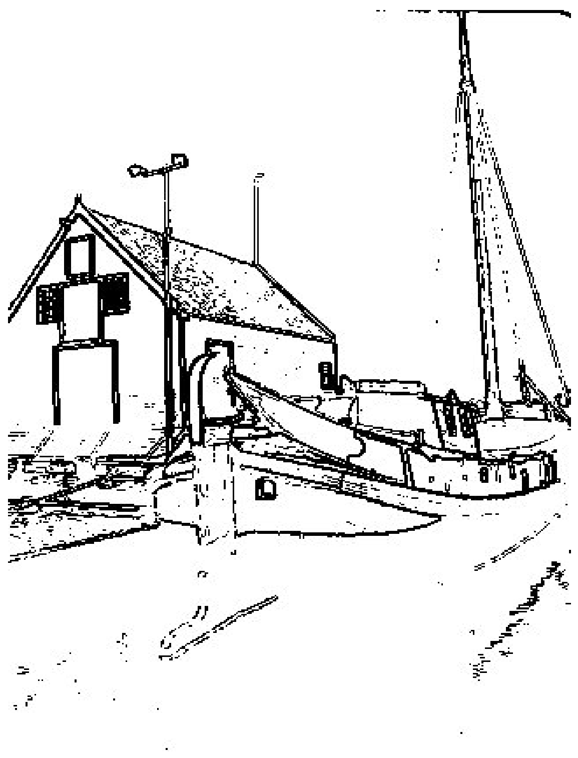

X gradient image after applying Sobel to Boat_Noisy image


Y gradient image after applying Sobel to Boat_Noisy image


Application of Sobel Filter to Boat


Application of Sobel Filter to Boat_Noisy


Edge map for Boat Image


Edge map for Boat_Noisy image

Depending upon the value of threshold set, we get more number of edges in the image if we set the threshold value more. I have set threshold to be 0.10, i.e., 10%. Also, in my observations, the edge map obtained from the Boat image is better than that obtained from the Boat_Noisy image, as more true edge lines are shown.

Problem 2.(a).2

I. Abstract and Motivation

In this problem we were asked to implement the basic Edge detection technique on the images provided by using LoG Filter. The RGB image is first converted to grayscale image and then LoG filter is applied on the image to get the edges present in the images. In images, edges are defined by lines comprising points where the brightness changes sharply. They form the boundaries of different objects present in the image and present valuable information.

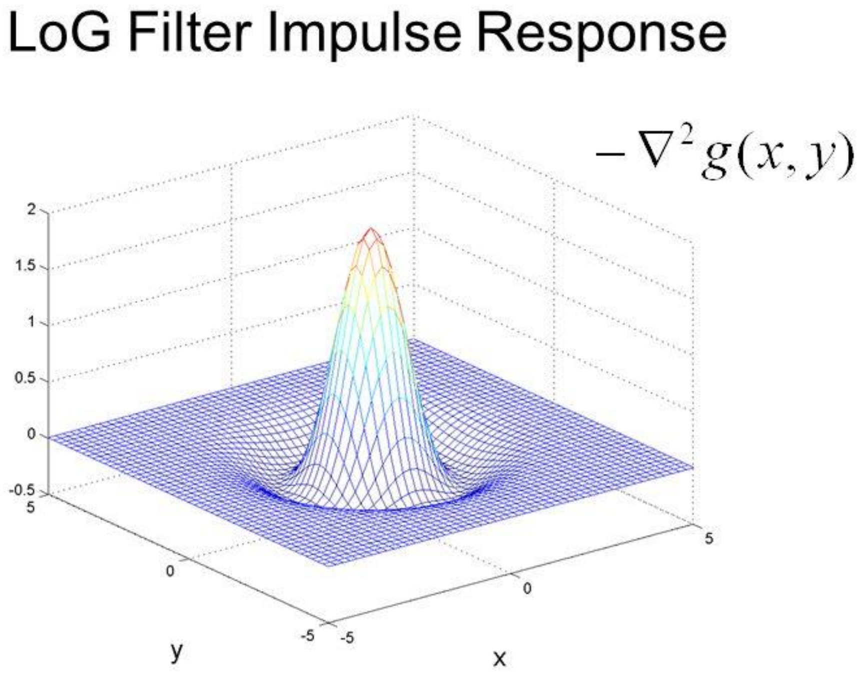
II. Approach and Procedures

First, the RGB image is converted to Grayscale by using the formula:

$$Gray = 0.21*Red + 0.72*Green + 0.07*Blue;$$

Then extra columns and rows are added or padded to the original image. Then we apply the LoG filter. Which is a representation of the second derivative of the image. Here, theoretically a Gaussian filter is used to smoothen the image and reduce noise, and then a Laplacian is used to detect the edges present in the image. The formula for the filter is given by:

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}}$$



The 2-D LoG can be approximated by a 5 by 5 convolution kernel such as :

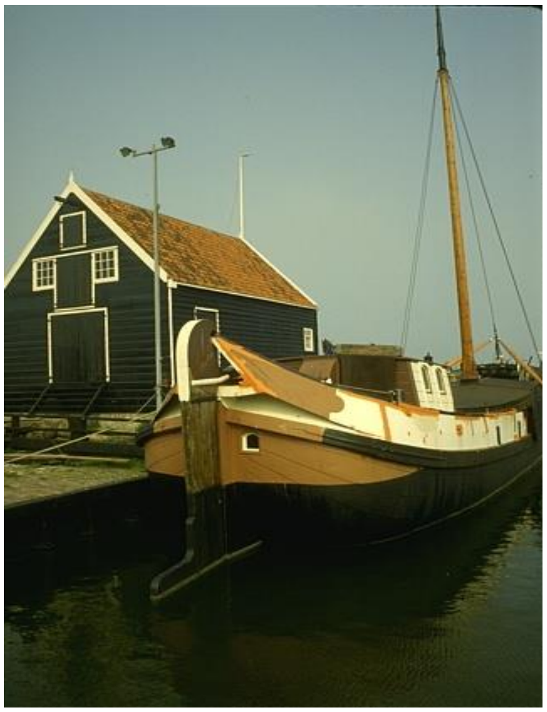
$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & -16 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

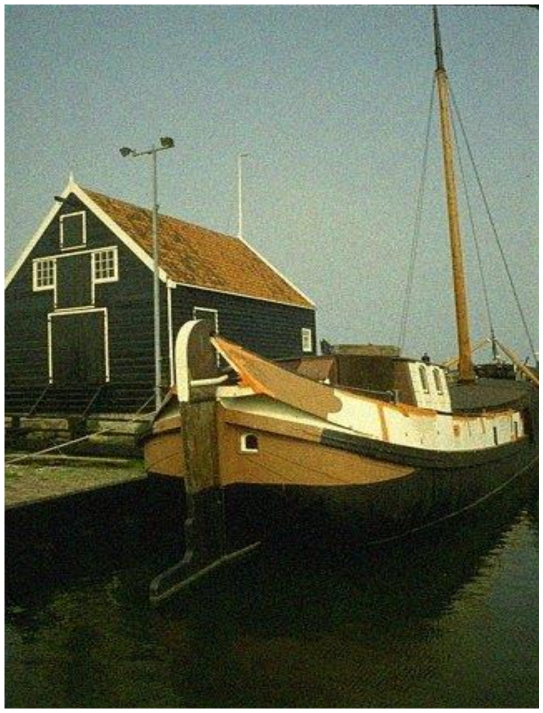
Here, the value of sigma is 1.4. This filter is applied to the image to get the LoG response of the image. For finding the kneepoint from the image, we first make the histogram of the image and then the cumulative histogram. For the first kneepoint, I set the condition that it is the value of intensity for which the cumulative intensity becomes 0.05*(size of image). Similarly, for the second kneepoint, the condition that it is the value of intensity for which the cumulative intensity becomes 0.95*(size of image). After getting the two kneepoint, we can divide the image into three parts: before first kneepoint(which is set as -1), between the two kneepoint(which is set as 0) and after the second kneepoint(which is set as 1).

To show the ternary map, we can map -1, 0, 1 to gray-level 64, 128, 192. This gives the ternary image.

For zero crossing detection, we have to first roam in a 3x3 window over the entire image. For each window, if the middle pixel is 0, then zero crossing is possible. Then we check that given middle pixel to be zero, whether there's the presence of 1 and -1 in directly opposite positions of the middle position. If such a case arises then zero crossing is present. There are 8 such possible cases. So to check, I checked whether the multiplication of opposite pixels result in negative number, which is easier to implement than implementing eight filters. Through this method, I got the zero crossing edge map.

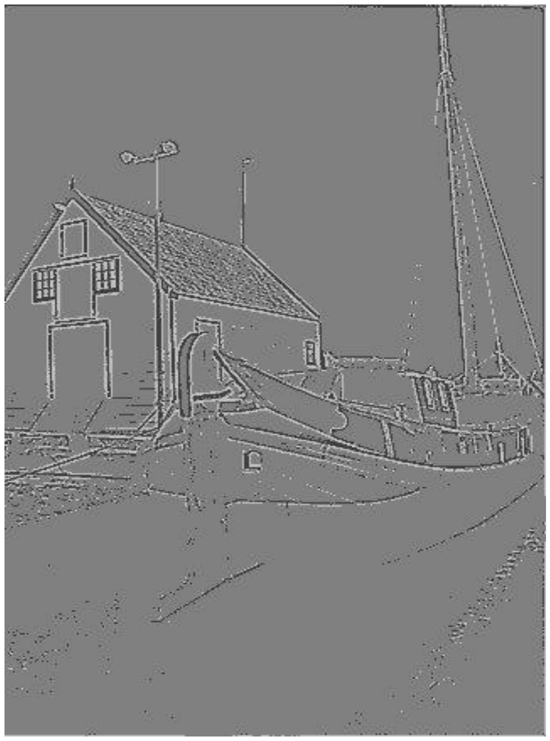
III. Results and Discussion

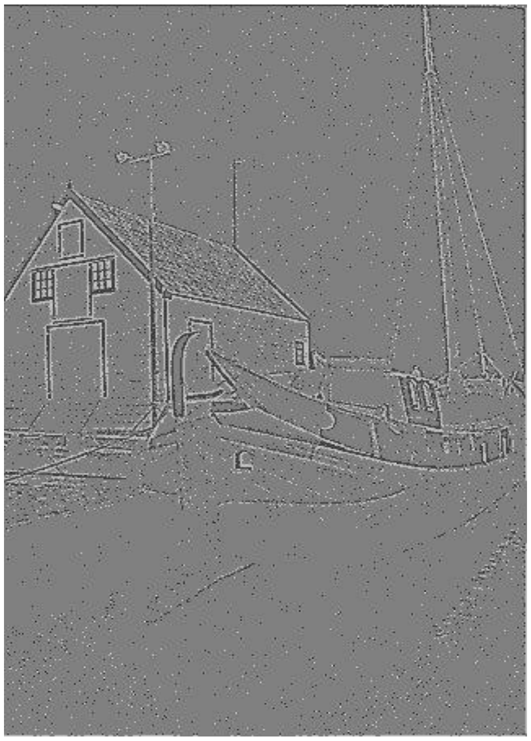

Main Image : Boat

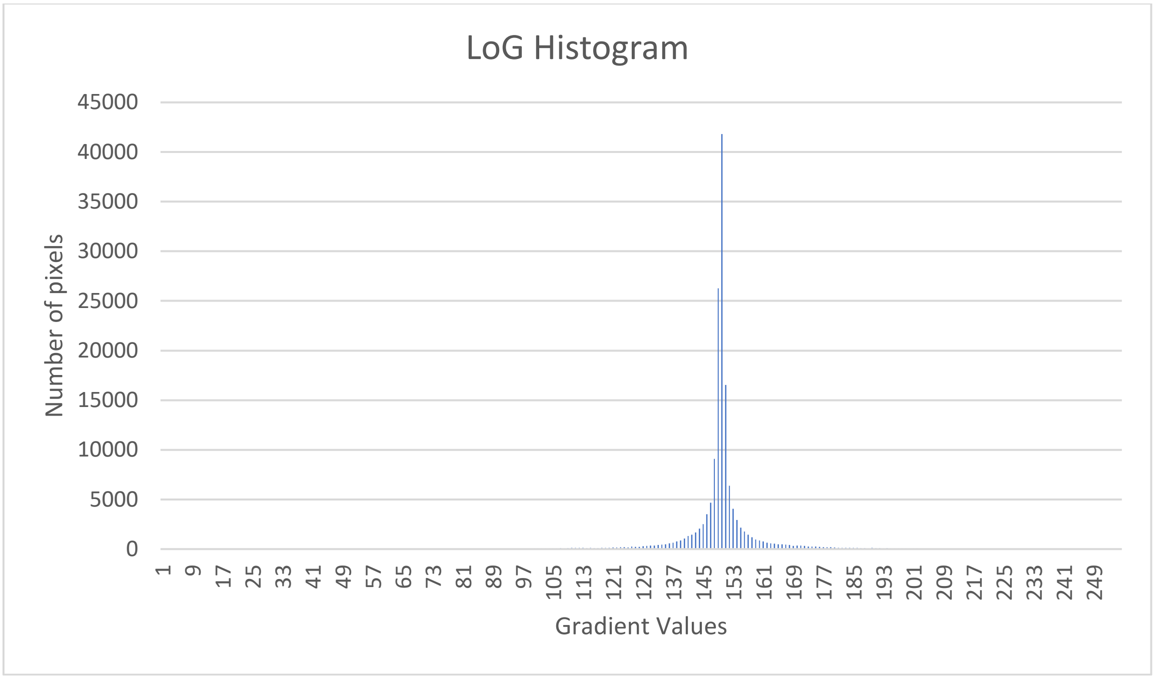

Main Image : Noisy Boat

Application of LoG filter to Boat

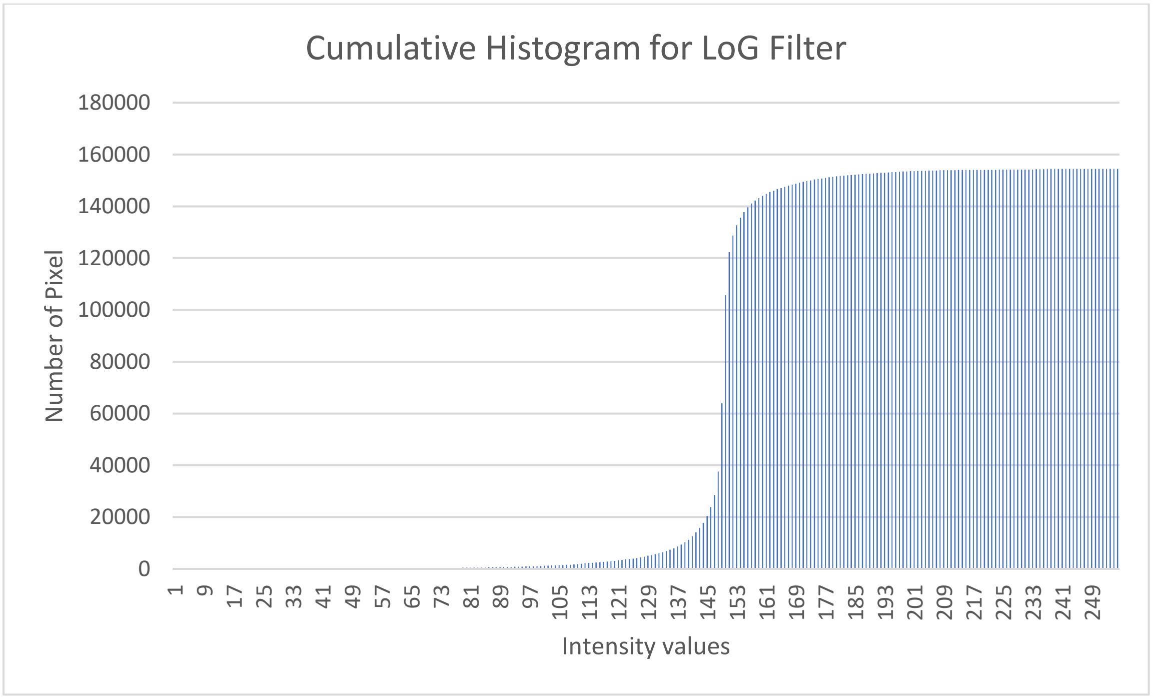
Application of LoG filter to Boat_Noisy


Ternary Image of Boat

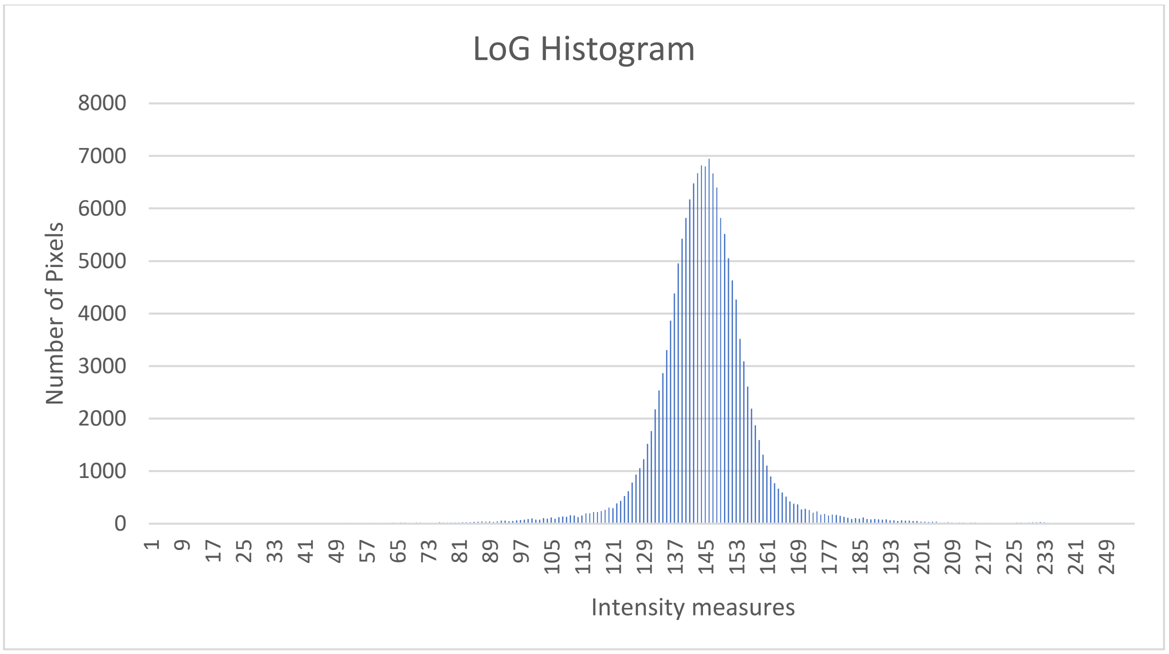

Ternary Image of Boat_Noisy



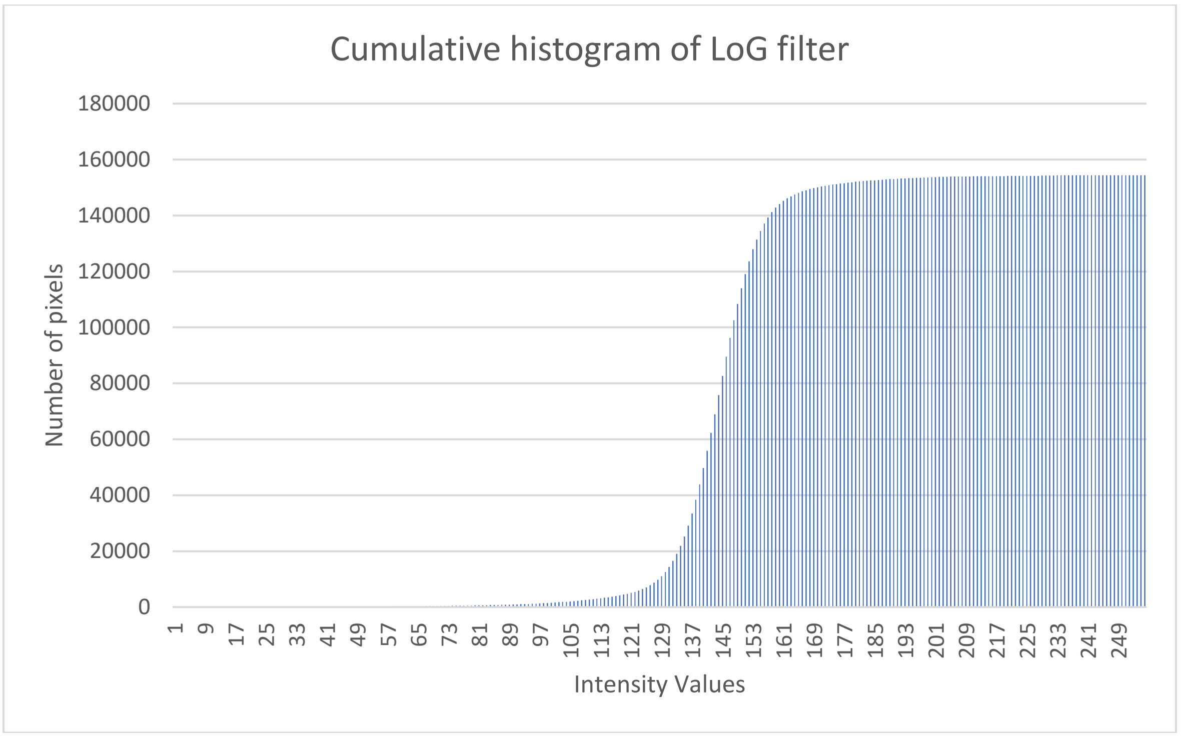
Histogram for Boat image



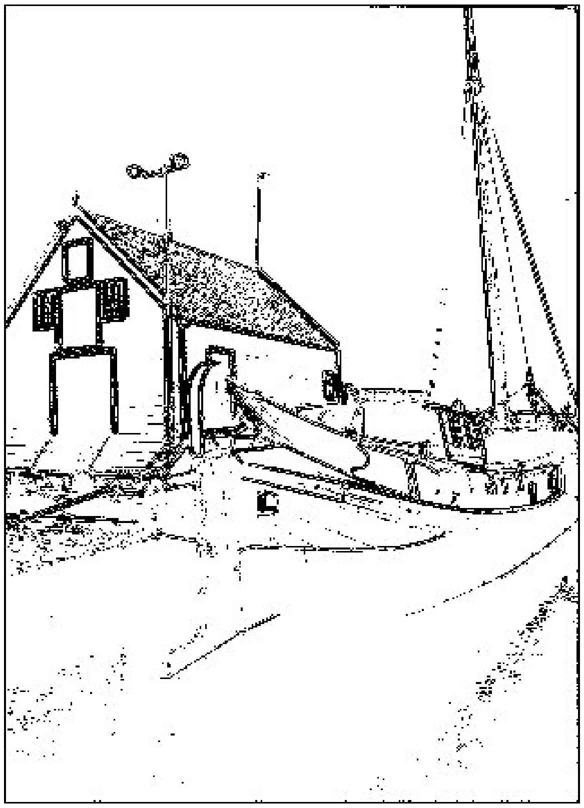
Cumulative histogram for Boat image to find knee points

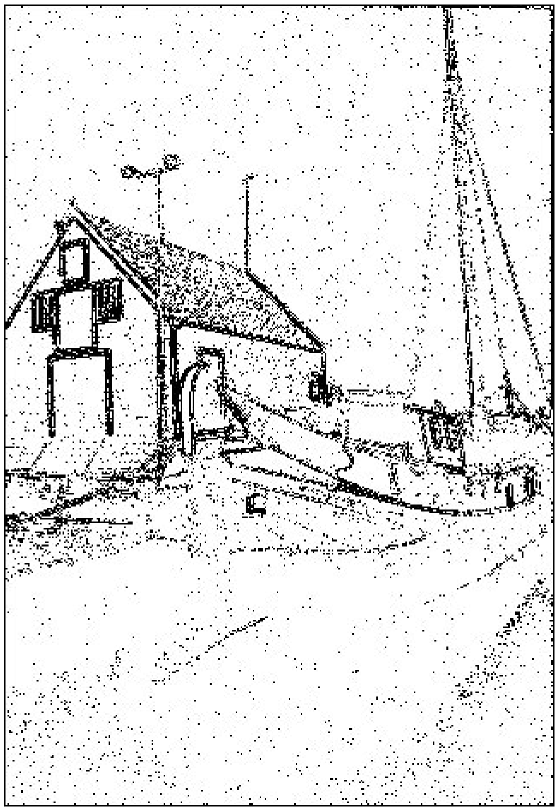


Histogram for Boat_Noisy image



Cumulative histogram for Boat_Noisy image to find knee points





Zero Crossing for Boat Image Zero Crossing for Boat_Noisy Image

The zero crossing edge map for the Boat image is much better than the Boat_Noisy image, as can be seen from above. The Sobel filter output is susceptible to noise interference but LoG having the Gaussian inbuilt, can remove the noise. LoG filter also gives thinner edges than Sobel filter. But, LoG filters often detect the edge of double pixel wide hence it is rarely used for direct edge detection. Hence for best and optimal edge detection, we choose better filters like Canny edge filter or Gabor filters.

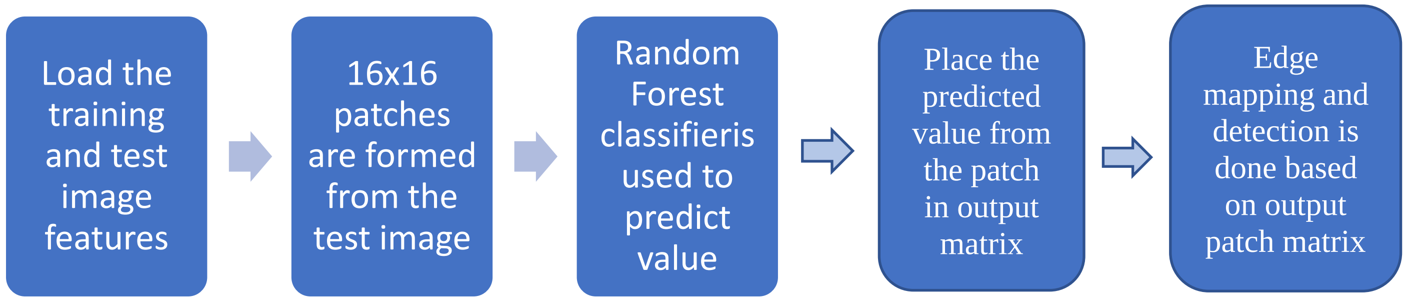
Problem 2.(b)

I. Abstract and Motivation

In this problem we were asked to implement the Structured Edge (SE) detector to extract edge segments from a color image with availaible matlab source code. The SE detector can be directly applied to the RGB image without converting it into a grayscale image. Also, the SE detector will generate a probability edge map. To obtain a binary edge map, we need to binarize the probability edge map with a threshold. The Structured Forest method was developed by Piotr and Zitnik. It gives good performance on edge detection with comparably less computation time to other algorithms.

II. Approach and Procedures

The Structured Edge algorithm uses patches of image as input with which it predicts local segmentation masks. These structured labels are then used to train decision trees and form random forests. The edge detector considers a patch and determines whether it is an edge point or not as patches of edges normally exhibit well known structures like straight line or T-joints. These edge patches are classified using random forest classifiers. The flow chart is as follows:


Load the training and test image features
16x16 patches are formed from the test image
Random Forest classifieris used to predict value
Place the predicted value from the patch in output matrix
Edge mapping and detection is done based on output patch matrix

Initial step is the training of the random forest classifier. This is done by taking a patch from the input images and the ground truth images. Then a structured output is formed which states whether a certain combination can be classified as an edge or not. The input features have two types: pixel lookups and pairwise pixel differences. It predicts a structured 16x16 segmentation mask to find the pixel lookups. The total number of channels is 13, which includes 3 color channels, 2 magnitude and 8 orientation. All channels are downsampled by 4. But when the decision tree is built, not all the features are used. As if all features are used, it might lead to over-fitting problem.

After the random forest is trained, it can be tested on the test image which is segmented and given as input. A random forest is a composite approach and runs on the divide and conquer formula. The main idea behind this is to construct a number of decision trees to classify samples. The principle is that normally each decision tree has very low classification rate and may overfit for new data. But once we ensemble the various decision trees, due to the law of large numbers, It gives better classification rate and less overfitting chances.

The input or output space can be complex and have high dimensions. In the decision tree, an input is entered in the root node. The input then traverses through the tree till it reaches the leaf node. Each node on the tree is a binary splitting function which decides whether the test sample goes to the right or left child nodes based on certain parameters. Often, complex split functions are utilised in the decision trees. A set of such trees are trained independently to find the parameters to be used in the split functions. The parameters are chosen to provide maximum information gain.

Every decision tree has two types of nodes: leaf node and internal leaf node. Internal leaf nodes have splitting functions which include features and thresholds. For the input training sample from image, the internal leaf node classifies it into different child nodes. Iteratio of this process multiple times leads to sample finally reaching the leaf node, where labels are set. The main task is to choose the optimal features and thresholds for the splitting function to work on.

The toolbox provided was used in Matlab for the process and all the Mex Files were compiled through the toolbox.

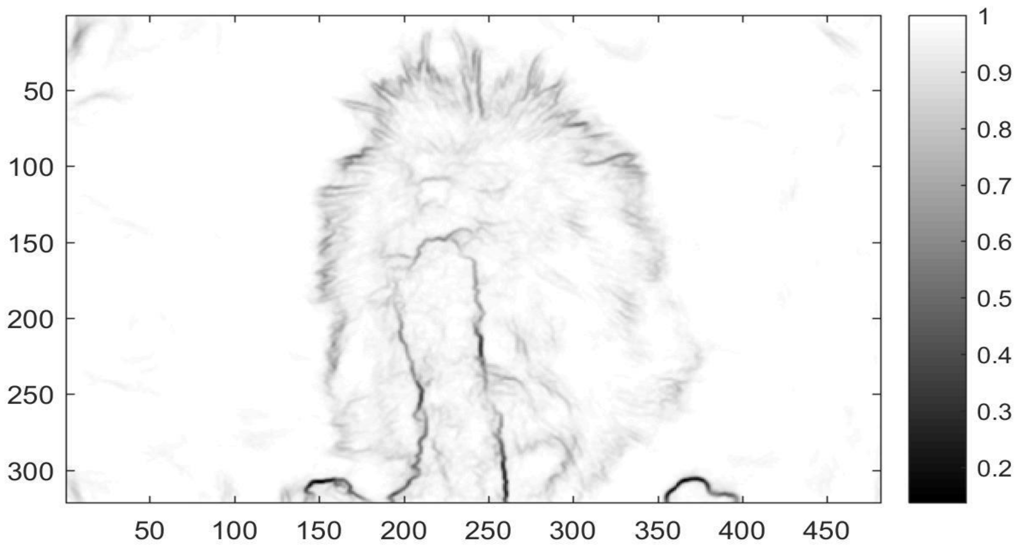
The parameters for SE are as follows:

model.opts.multiscale	Determines the accuracy of edge map, I used 1
model.opts.sharpen	Determines the sharpness of edge map, I used 2
model.opts.nTreesEval	Dtermines the number of trees for evaluation, affect the running time, I used 4
model.opts.nThreads	Set the maximum threshold for evaluation, I used 4
model.opts.nms	Determines whether to use NMS or not, I used 0, i.e. NMS not used

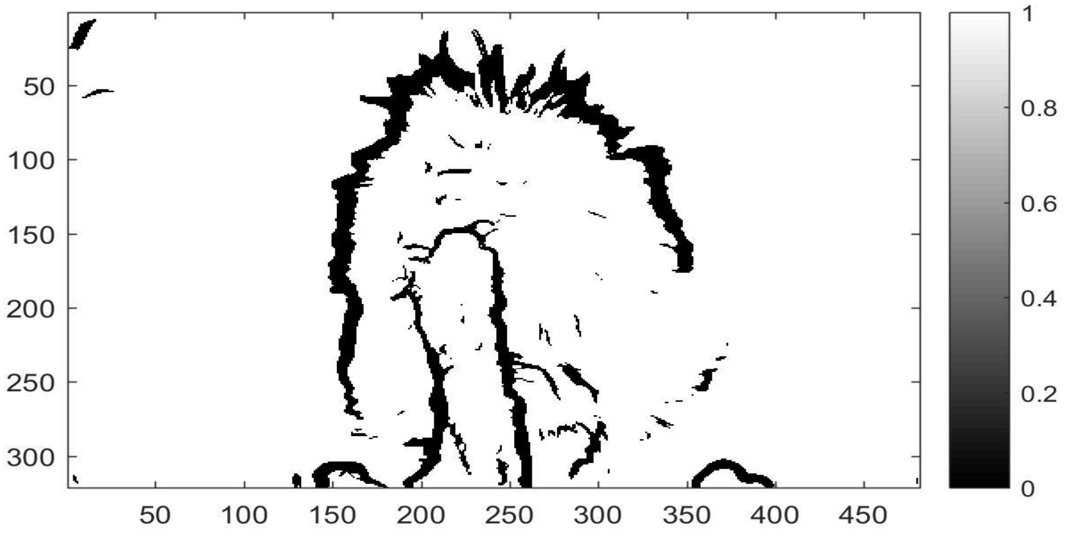
The second parameter determines the sharpness of the edges in the images. With increase in value, the edges become sharper. The third parameter can reduce the running and compilation time. The fourth parameter determines the threshold for evaluation. The pixel values of edge map we get are not 0 and 1, this threshold needs to be set to determine 0 and 1. This parameter doesn't seem to have an effect on the output edge map.

III. Results and Discussion

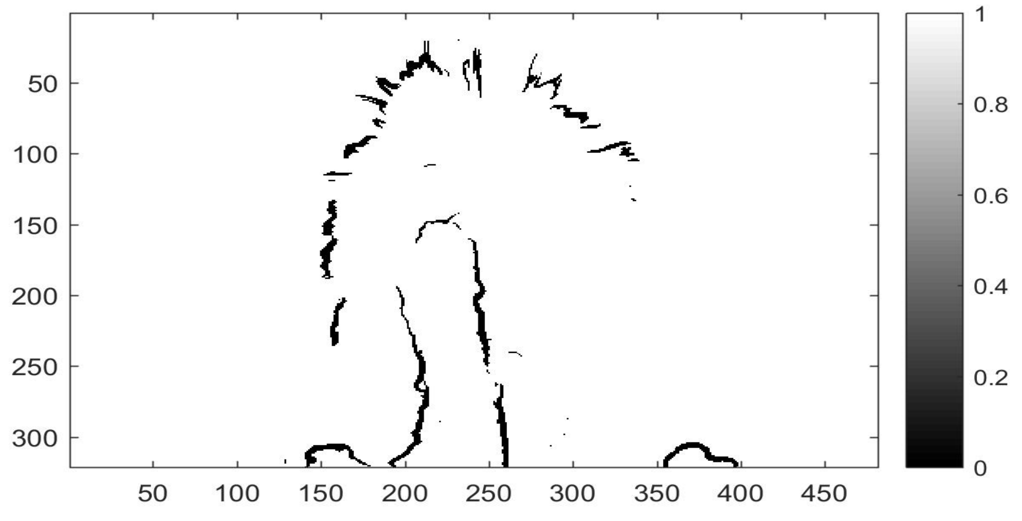
Original Image – Animal



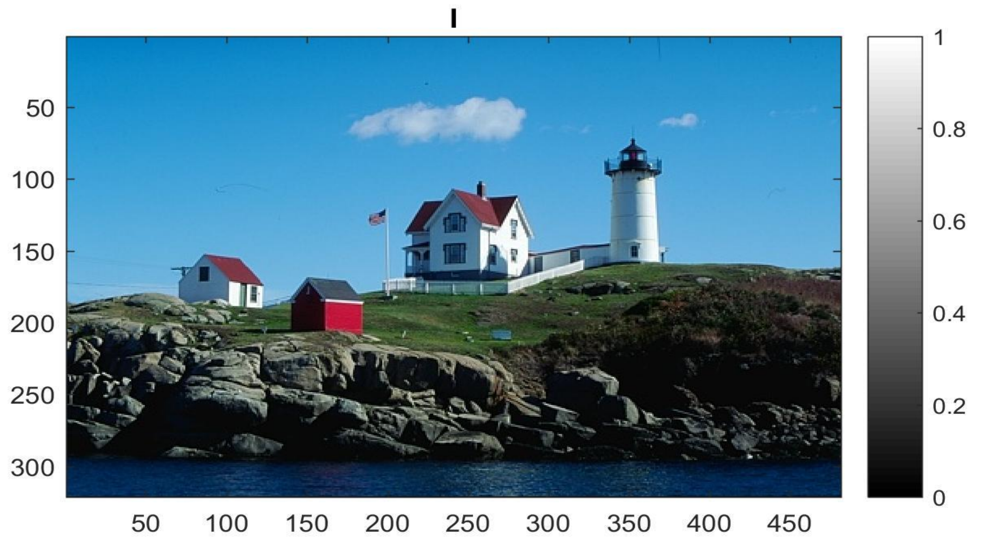
Output of Structured Edge algorithm for Animal image



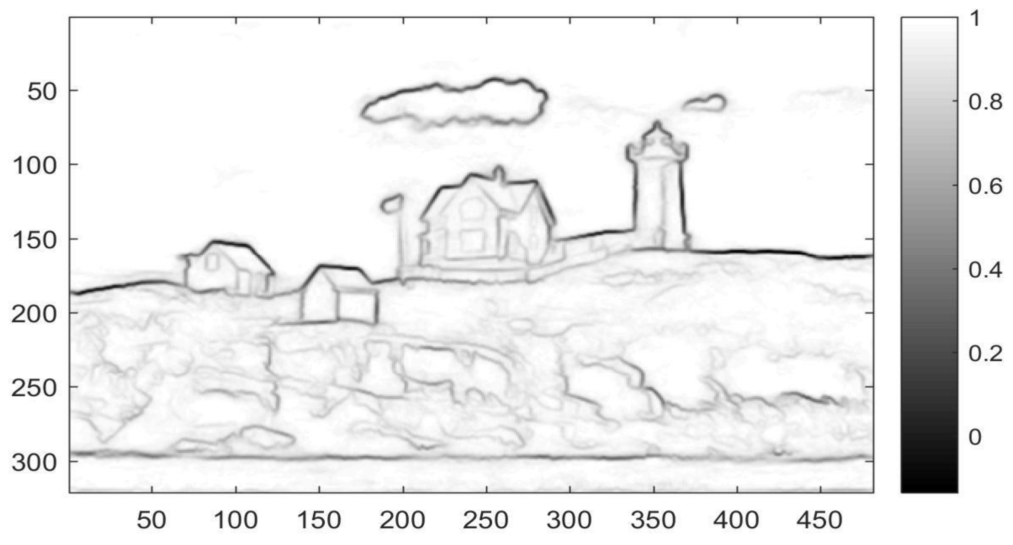
Binarized image with threshold = 0.1 for Animal



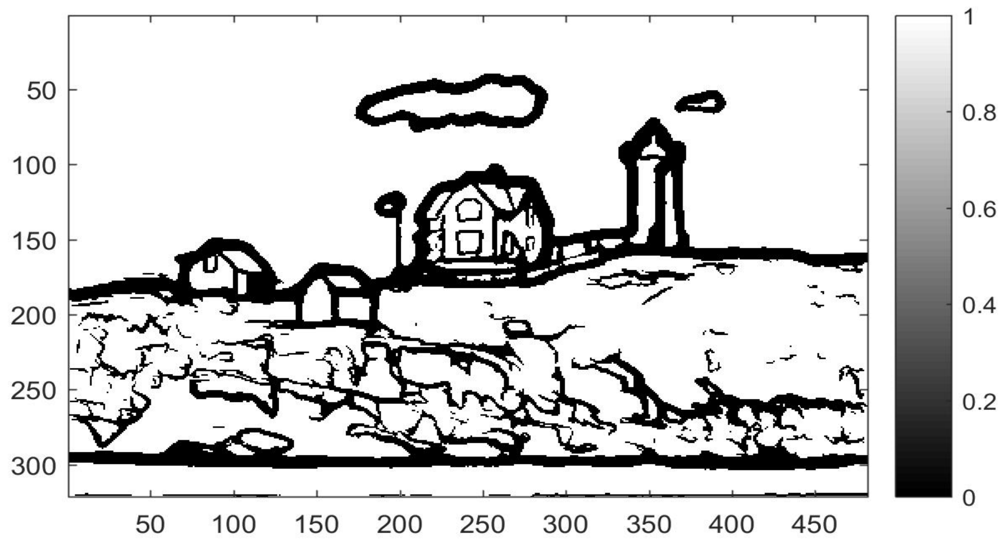
Binarized image with threshold = 0.2 for Animal



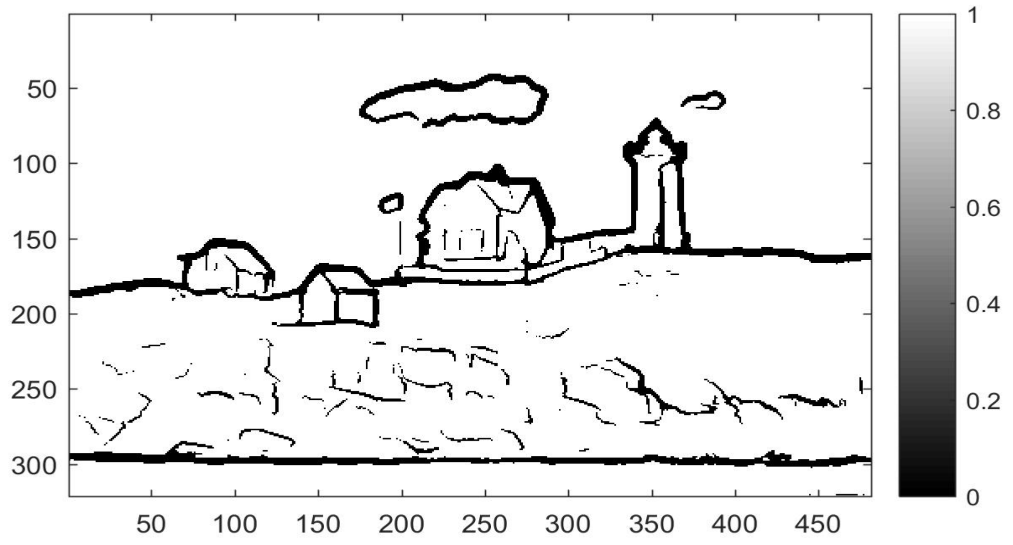
Original Image – House



Output of Structured Edge algorithm for House image



Binarized image with threshold = 0.1 for House



Binarized image with threshold = 0.2 for House

SE detector overcomes the shortcomings of the Sobel Edge detectors and LoG detectors. It has thin edges of objects in images and also preserves the complete edge information. Also the running time for the SE detector was very fast compared to the LoG edge detection filter. And we can set different parameters for the splitting functions for creating the decision trees. This helps in better edge detection results.

Problem 2.(c)

I. Abstract and Motivation

In this question, we have to perform performance evaluation, which is a key component in checking whether the estimates generated by the SE algorithm actually have similarity with the ground truths drawn by humans. To evaluate the performance of an edge map, we need to identify the error.

II. Approach and Procedures

All pixels in an edge map correspond to one of the four classes:

(1) True positive: Edge pixels in the edge map coincide with edge pixels in the ground truth. These are edge pixels the algorithm successfully identifies.
(2) True negative: Non-edge pixels in the edge map coincide with non-edge pixels in the ground truth. These are non-edge pixels the algorithm successfully identifies.

(3) False positive: Edge pixels in the edge map correspond to the non-edge pixels in the ground truth. These are fake edge pixels the algorithm wrongly identifies.
(4) False negative: Non-edge pixels in the edge map correspond to the true edge pixels in the ground truth. These are edge pixels the algorithm misses.

The pixels in (1) and (2) are correct ones while those in (3) and (4) are error pixels of two different types to be evaluated. The performance of an edge detection algorithm can be measured using the F measure, which is a function of the precision and the recall.

Precision(P) = #True Positive / (#True Positive + #False Positive)
Recall(R) = #True Positive / (#True Positive + #False Negative)

F = 2*P*R / (P+R)

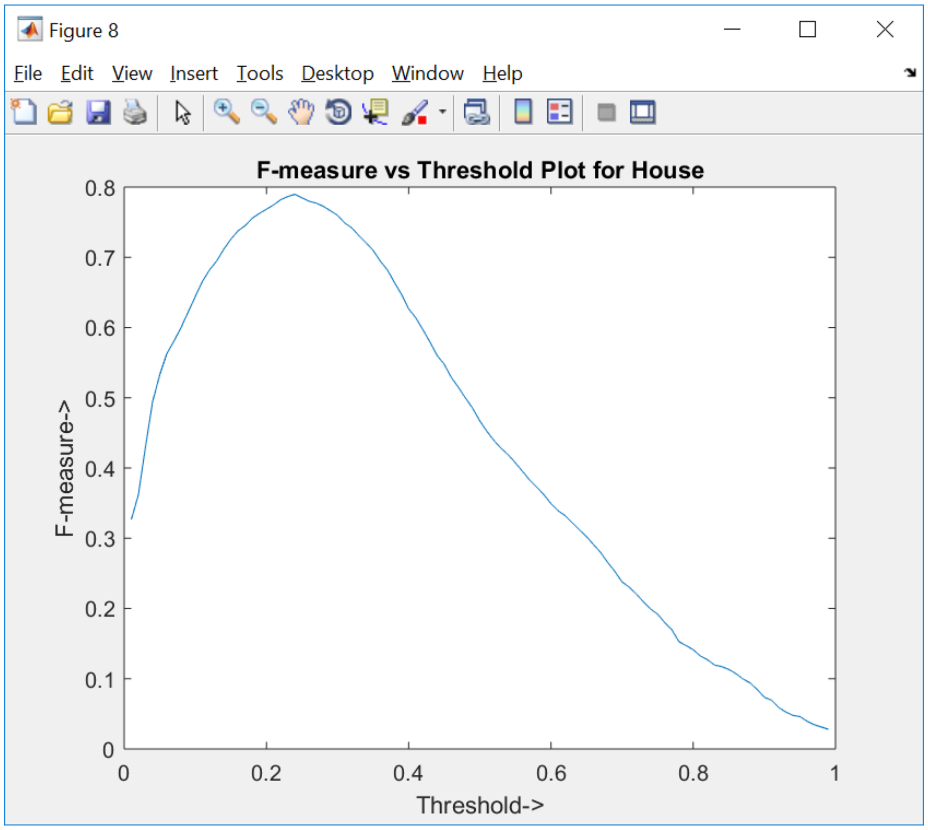
We can make the precision higher by decreasing the threshold in deriving the binary edge map. However, this will result in a lower recall. Generally, we need to consider both precision and recall at the same time and a metric called the F measure is developed for this purpose. A higher F measure implies a better edge detector.

III. Results and Discussion

For the house image, for threshold = 0.1,

Ground Truths	Precision	Recall	F-Measure
GT1	0.246961	0.932597	-------
GT2	0.360825	0.838099	-------
GT3	0.320511	0.851594	-------
GT4	0.308817	0.839047	-------
GT5	0.306509	0.869489	-------
Mean Value	0.3087246	0.8661652	0.4552027

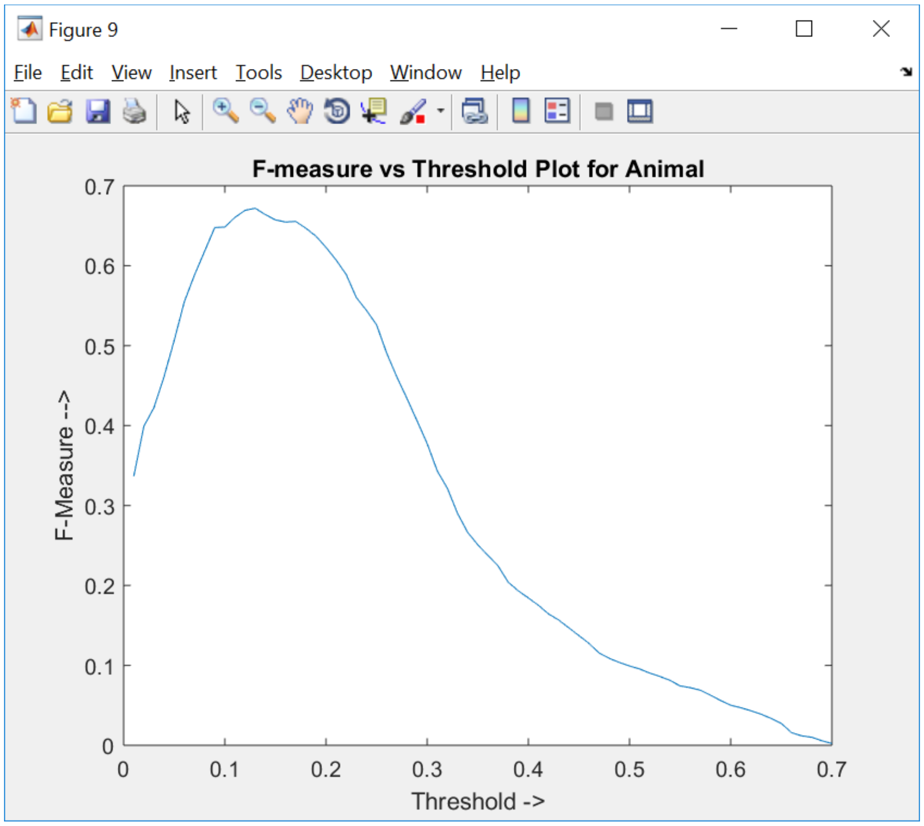
Similarly, we can calculate the F measure value by taking the mean of all the precision and recall values for all the ground truths for all the threshold values. If we plot a curve of F measure value against threshold, it looks as follows:



For the animal image, for threshold = 0.1,

Ground Truths	Precision	Recall	F-Measure
GT1	0.51716	0.654372	-------
GT2	0.58204	0.630989	-------
GT3	0.286789	0.438849	-------
GT4	0.633286	0.717635	-------
GT5	0.492713	0.580609	-------
Mean Value	0.5023976	0.6044908	0.54873595

Similarly, we can calculate the F measure value by taking the mean of all the precision and recall values for all the ground truths for all the threshold values. If we plot a curve of F measure value against threshold, it looks as follows:



From the two plots, it can be seen that the maximum possible F-measure value for the House image is more than that of the animal image.

Thus, we can see that Sobel filter has the worst result from the three methods used. The edges are very wide in the case of Sobel and there is a possibility of the F-measure being low due to the presence of false positives in the image. The LoG filter performs better than Sobel as it removes the noise and then does filtering. Though the Sobel Filter is easier to implement but the LoG filter gives better result as it find the correct places of edges and we can decide the sigma of the Function to be used. The best method though is the SE detection method which trains a model to detect the edges in an image. It gives much better precision and recall values at the same time. The random forest approach used in SE detection reduces the probability of error in assignment of the edge.

For the House image, we get a maximum F-measure of 0.7897 at a threshold of 0.24. For the Animal image, we get a maximum F-measure at 0.6719 at a threshold of 0.13. It's easier to get higher F measure for the House image because the image has simpler edge structure than the Animal image. Also the objects are quite monotonous for the House image, whereas for the Animal image, the edges are more complicated for the animal's body. Most humans won't mark the edges formed by the fur on the animal's body in their ground truths. So the number of False positive and false negative points increases for the Animal image. In the case of the House, it's easier for humans to draw the edges and boundary of the house and rocks. Hence, intuitively, this might be the reason behind getting better F-measure for the House image.

As F-measure is the harmonic mean of the precision and recall, when Precision is significantly higher than recall in value, we can simply approximate F-measure value to be equal to twice of Recall. Then it's not possible to get a high value of F-measure. This holds true for the vice versa case too.

When sum of Precision and Recall is constant, let P+R = C, a constant. Then, R = C-P. Then, we can write,

$$F = 2*(P*R) / (P+R) = 2*P*(C-P) / (P+C-P) = 2*P*(C-P) / C$$

To get the highest F, P = argmax(P*(C-P)) for P. Solving, P = R = C/2.
Thus, if sum of precision and recall is constant, then F-measure reaches maximum value when precision is equal to recall.

Problem 3.(a)

I. Abstract and Motivation

Features with meaningful information are the basic requirement for all types of machine learning and computer vision tasks. The robustness and strength of features extracted from an image helps in description of an image in the form of it's attributes. SIFT and SURF are two algorithms that can help extract features from an image that are scale and rotation invariant. Sift and Surf are used in varied applications like object recognition, motion tracking and navigation. In this problem, we are asked to use the SIFT and SURF algorithms to identify the features from given images. We also do object matching using the key points and descriptors found from the images using the algorithms.

II. Approach and Procedures

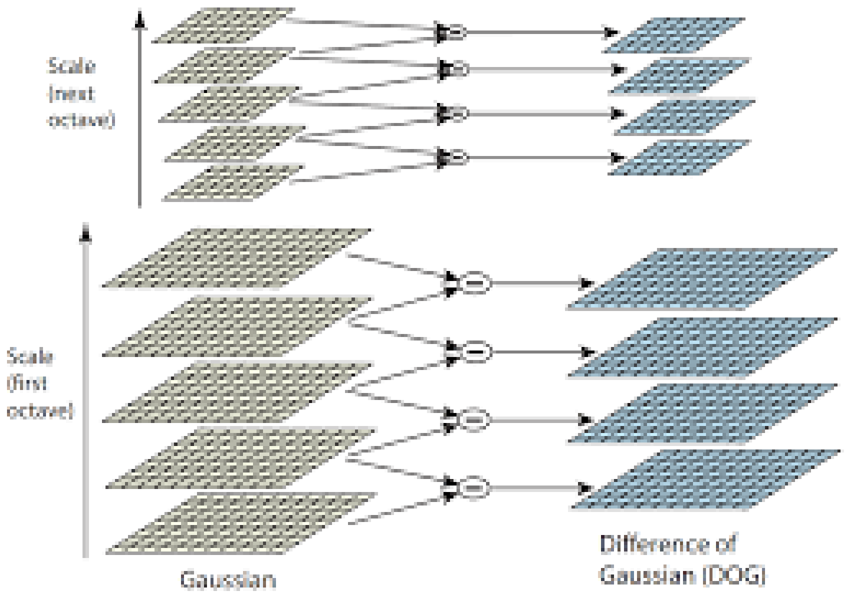
SIFT Algorithm:

SIFT stands for Scale Invariant Feature Transform. It is an algorithm to extract feature descriptors which are invariant to rotation, scaling, noise, small changes in point of view and change in illumination. It consists of four parts:

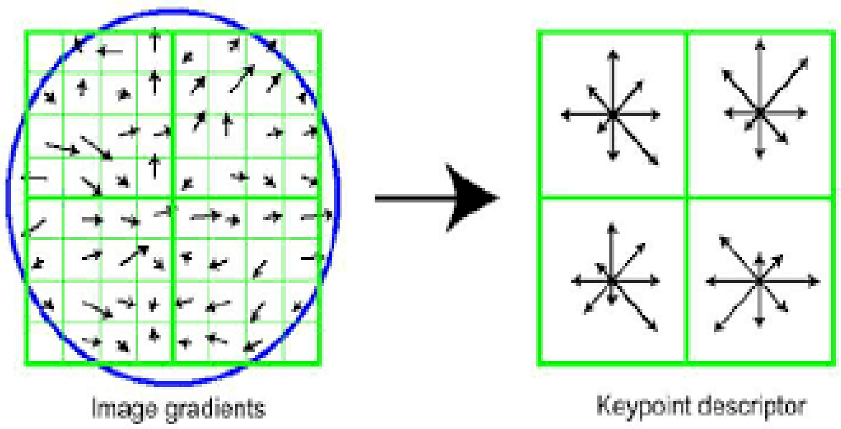
- Scale Space Extrema Detection
- Key Point Localization
- Orientation Assignment
- Generation of Key Point Descriptors

To make the key points scale invariant, the input image is considered on different scales by applying Gaussian smoothening filter with different amounts of sigma. The scale space consists of m octaves with 3 images in each octave. Each octave is then down sampled by a factor of two. Also, each image in an octave is smoothened by a sigma value that is multiple of k. David Lowe, the author of the paper on SIFT, found the value of sigma as 1.6 and value of k as 0.707. After Gaussian smoothening, the difference of each Gaussian is taken. As LoG (Laplacian of Gaussian) takes a large time to compute, DoG is used. The DoG gives an edge map. From this edge map, point of interest is selected by iterating through all the possible points and then selecting the point of interest which lies in the local minima and maxima at a given scale. This is done by comparing a given point at different scales.

In the next step of key point localization, we use the points of interests from the previous step to compare with the hessian threshold to eliminate noisy and weak points. First the Taylor series expansion of DoG is calculated and points lying below the threshold of 0.3 are eliminated. We compute the hessian matrix and it's eigen values. If the ratio of the first and second eigen value of the hessian matrix is greater than 10, then the point of interest is an edge point. So we discard that point. Hessian matrix decides the number of points to be removed. It takes the ratio of the trace of the matrix over it's determinant and according to heuristic computation decides the max values.



Then to achieve rotational invariance, we assign an orientation value to the key points that are obtained from the second step based on the neighborhood of the points of interest. The size of the neighborhood depends on the octave at which the key point is found. Then 36 bins are considered representing 360 degrees, with each bin having a difference of 10 degrees. Orientation is then found by the gradient magnitude of the pixel that we are calculating. The orientation is weighted by the gradient magnitude and a Gaussian weighted circular window with sigma value equal to 1.5 times the scaling factor. The weighted orientation is then plotted on an histogram and all the orientations which lie within eighty percent of the maximum histogram value are said to increase the robustness.



Now, to get the keypoint descriptors, we need to consider 16x16 neighborhood from the point of interest which is divided into 4x4 units from which histograms of the pixels are calculated. The Resolution of the histograms are taken as 8. This results in 128 bins in the 16x16 neighborhood, which are then considered as the feature vectors for a keypoint.

SURF Algorithm:
SURF stands for Speeded Up Robust Features. It is partly inspired by SIFT descriptor. But it is faster and more robust against image transformation. The SURF algorithm includes the integration of image for speeding up filtering. Then it uses a blob detector to find the interest points based on the Hessian matrix. In SURF, the mask convolutions are independent of the size of the filter and the integral images created are of the same size as the image that needs to be analyzed. The value of the integral image at any point is given by the sum of the intensity values lesser than or equal to where we evaluate the integral of the image. The process is similar to SIFT algorithm until we find the determinant of the Hessian matrix.

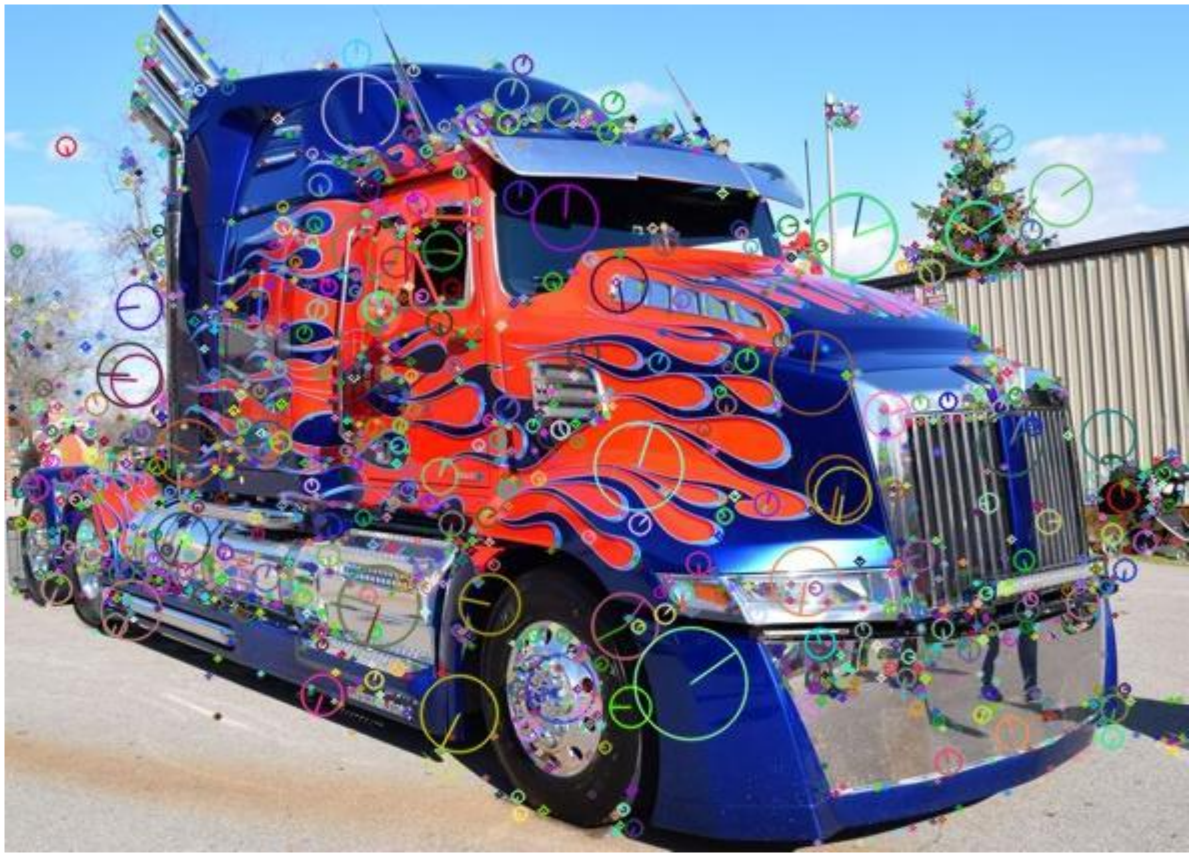
The next part of the process is finding the major interest points in which 3x3 non-maximal suppression below and above the scale of each octave is used. As the scale space is large and irregular shaped, the interest points are interpolated to obtain correct scales. Feature vector direction and angle is found by the Haar transform. Thus, 64 dimensional feature vectors for 16 sub regions are generated by the square description window. SURF is computationally more efficient.

Both the algorithms are directly implemented from the OpenCV library using python.

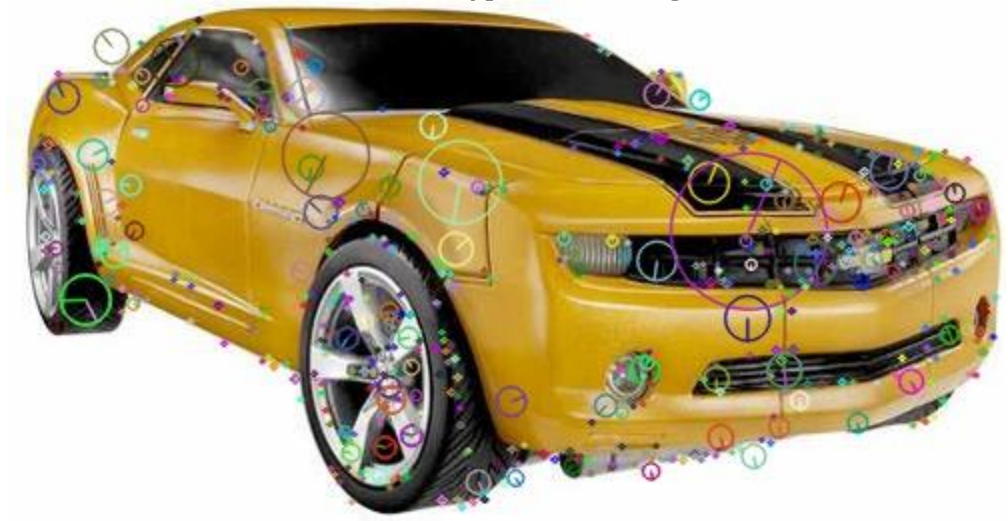
III. Results and Discussion

Image 1 – Optimus Prime

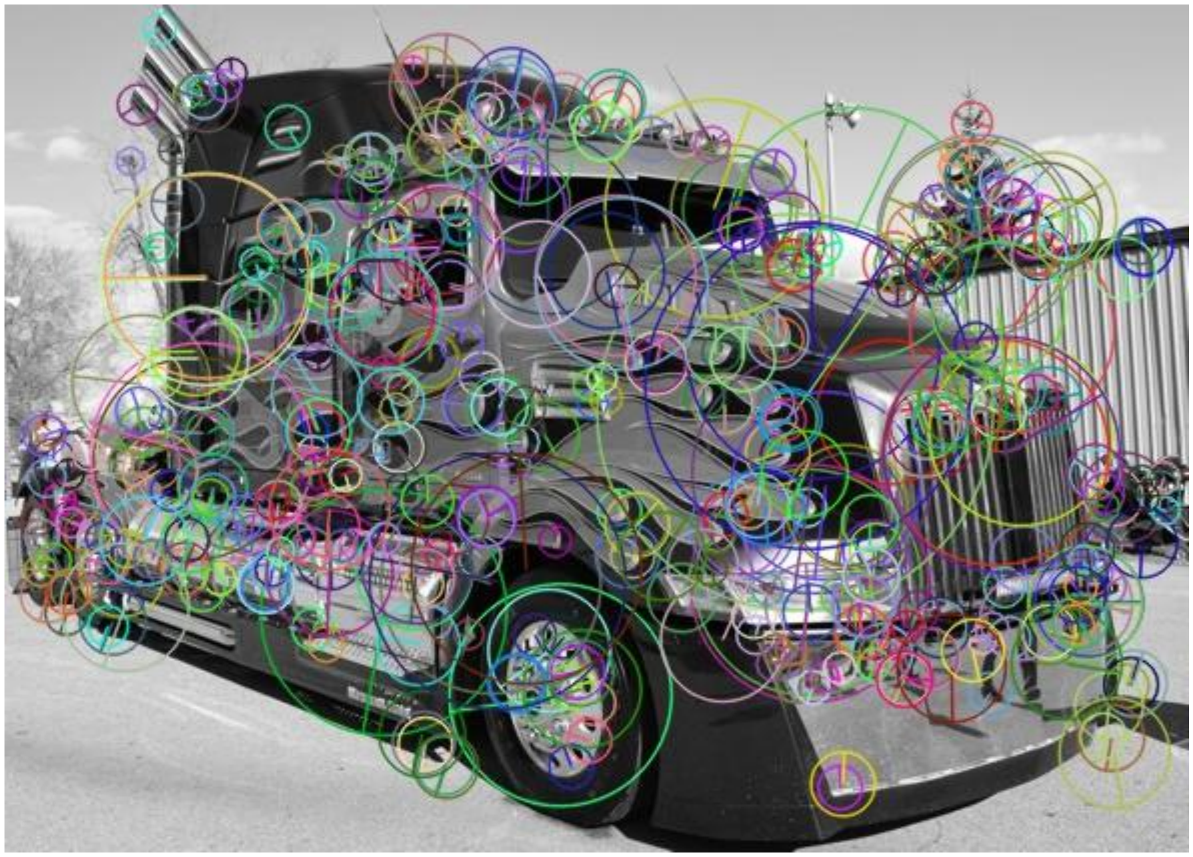
Image 2 – Bumblebee



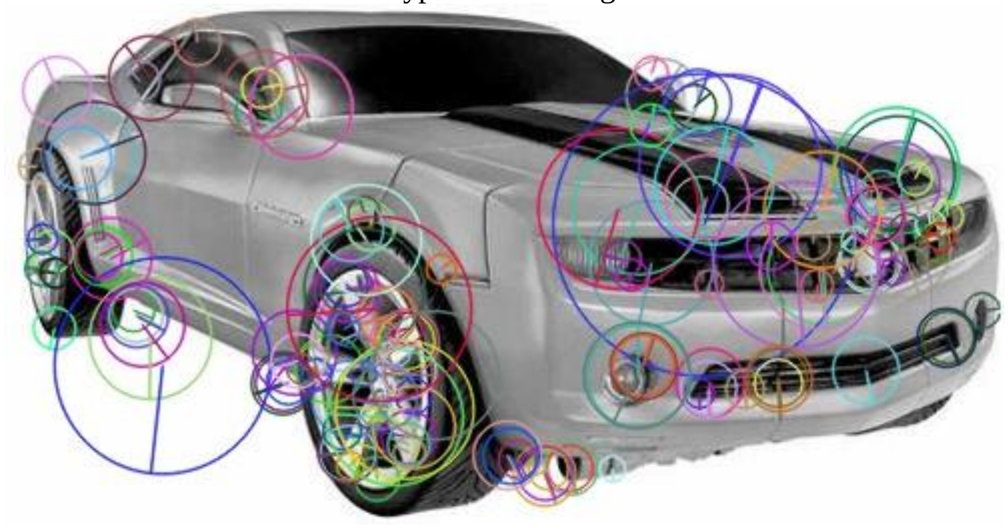
SIFT keypoints for Image1



SIFT keypoints for Image 2



SURF keypoints for Image 1



SURF keypoints for Image 2

Comparing SIFT and SURF, we find that SURF outclasses SIFT as it takes very little computational time when compared to SIFT. This time complexity arises as SURF uses integral images. Also, SURF has more keypoints than SIFT, though we can decrease the number of keypoints to be selected. Also as pyramidal structure of DoG is used in SIFT, the algorithm may falter in cases with similar backgrounds or blurring.

Problem 3.(b)

I. Abstract and Motivation

In this problem we are asked to use SIFT and SURF algorithms for the process of object matching. We are to extract the SIFT features from the given images and then match between the two images. Similarly, we have to work with the SURF features.

II. Approach and Procedures

After extraction of key points and descriptors using the SIFT and SURF algorithms, we can do Image Matching. We use Brute Force Matching algorithm to match the extracted features. The Brute Force matcher computes a match for every point and may contain noisy matches too. Another matching technique which can be used is the FLANN based matching technique. This technique uses the nearest neighbor approach and calculates the ratio of the first and second best guess. If the ratio is greater than 0.8, it cancels that match thus eliminating redundancy and increasing efficiency.
All the algorithms are directly implemented from the OpenCV library using python.

III. Results and Discussion

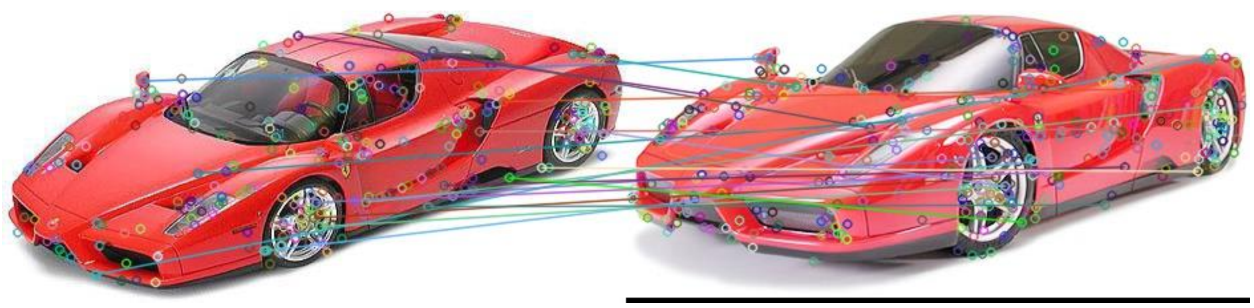


Image Matching using SIFT between Ferrari 1 and Ferrari 2

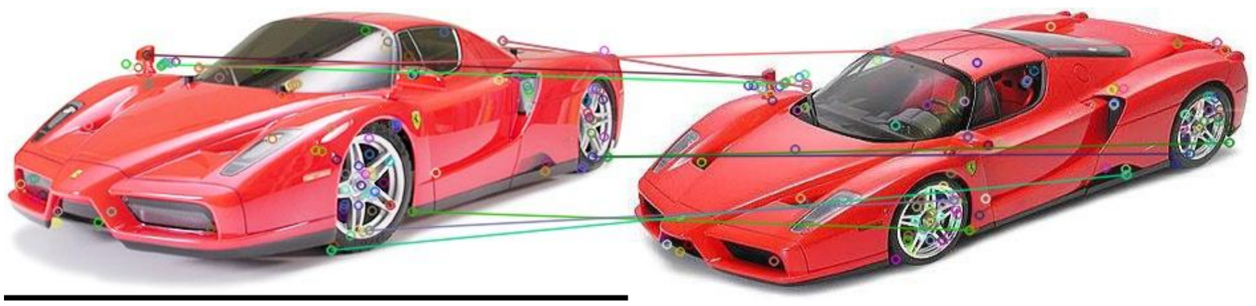


Image Matching using SURF between Ferrari 1 and Ferrari 2

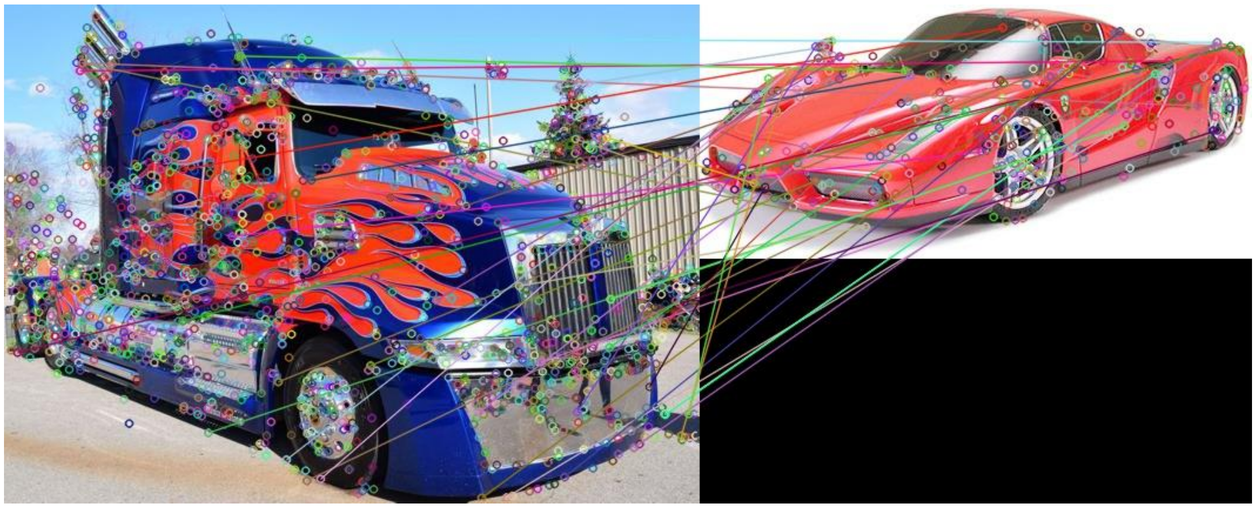


Image Matching using SIFT between Ferrari 1 and Optimus Prime

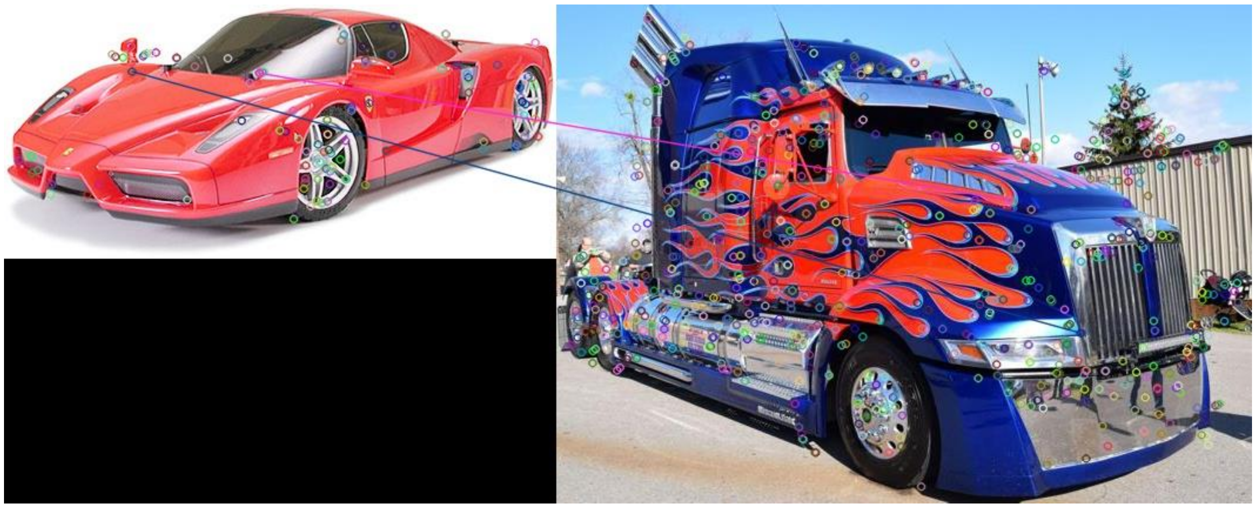


Image Matching using SURF between Ferrari 1 and Optimus Prime

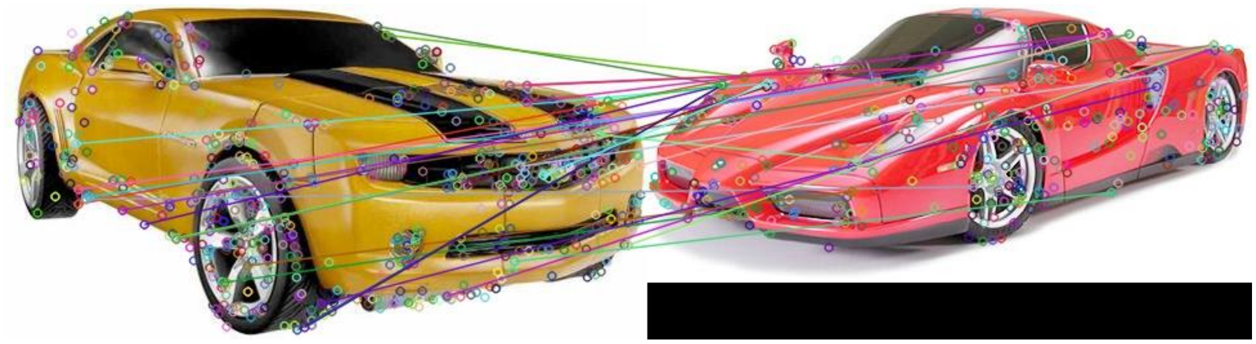


Image Matching using SIFT between Ferrari 1 and Bumblebee

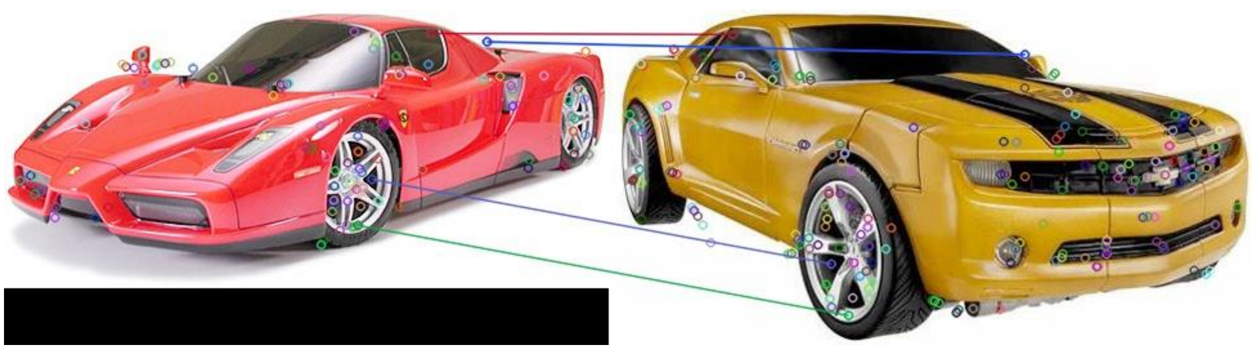


Image Matching using SURF between Ferrari 1 and Bumblebee

The Brute Force matcher algorithm of the Open CV can cause some noisy and redundant points to be selected. But overall, when two similar images are considered, in our case Ferrari 1 and Ferrari 2, the SIFT and SURF algorithms work well enough. But when two dissimilar images are matched, like, Ferrari 1 and Optimus Prime or Bumblebee, we see problems in the matching algorithm. When we see the matching of SIFT for Ferrari 1 and Optimus Prime, we see lots of matching being done, but most of these are noisy points. SURF performs better as it doesn't contain noisy points, but still it matches too few key points. But when the images are somewhat similar, like Ferrari 1 and Bumblebee, we see that the algorithms perform better than the previous case. This is because the cars in the images are similarly built and thus have more common keypoints which can be matched.

Problem 3.(c)

I. Abstract and Motivation

In this problem, we are to perform Bag of Words algorithm to form a codebook. We are to match the codewords for all four images and match Ferrari_2's codewords with other images.

II. Approach and Procedures

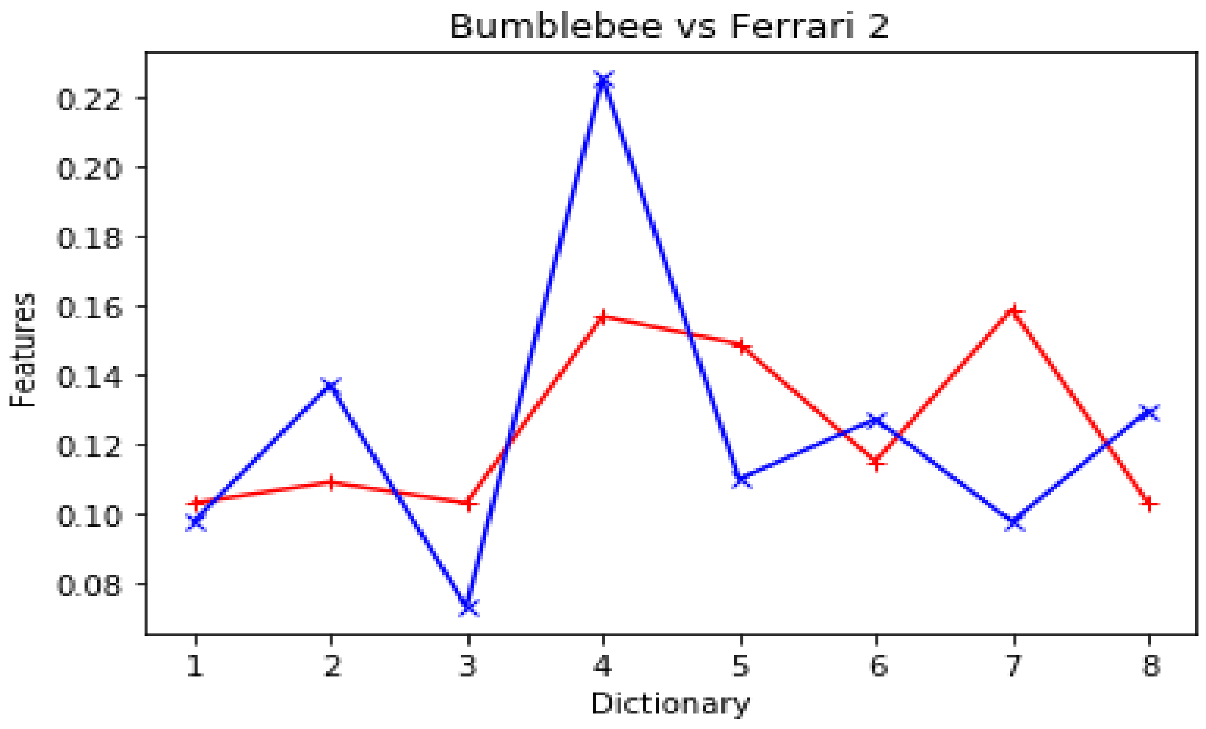
The Bag of Words algorithm is used for image classification in small systems where we can define the number of clusters to be formed. It is basically a vector of occurrence of the count of words and is expressed as a histogram of each word. The images are clustered into several clusters using k-means algorithm. The calculated features of test images can be compared to the codewords stored in the dictionary to get the histograms.

In the first step of the process, we calculate the SIFT descriptors of all the images. Then the descriptors of the training image are kept together in a bag and we choose K=8 from this bag of descriptors of the training image. Then the K-means algorithm is applied after choosing the 8 feature descriptors from the bag. Each image descriptor is thus classified on the basis of this 8 descriptors. The nearest neighbor algorithm is used for the test image descriptors. Once this is carried out, each of the image descriptors are classified into 8 bins on the basis of the codeword generated and the histogram of each of the 4 image descriptors is calculated. Once all the histograms are got, the one with the largest similarity with that of the test image descriptor histogram is considered as the best match to the test image. Then we print the class on the image which we find it closely belongs to based on this comparison.

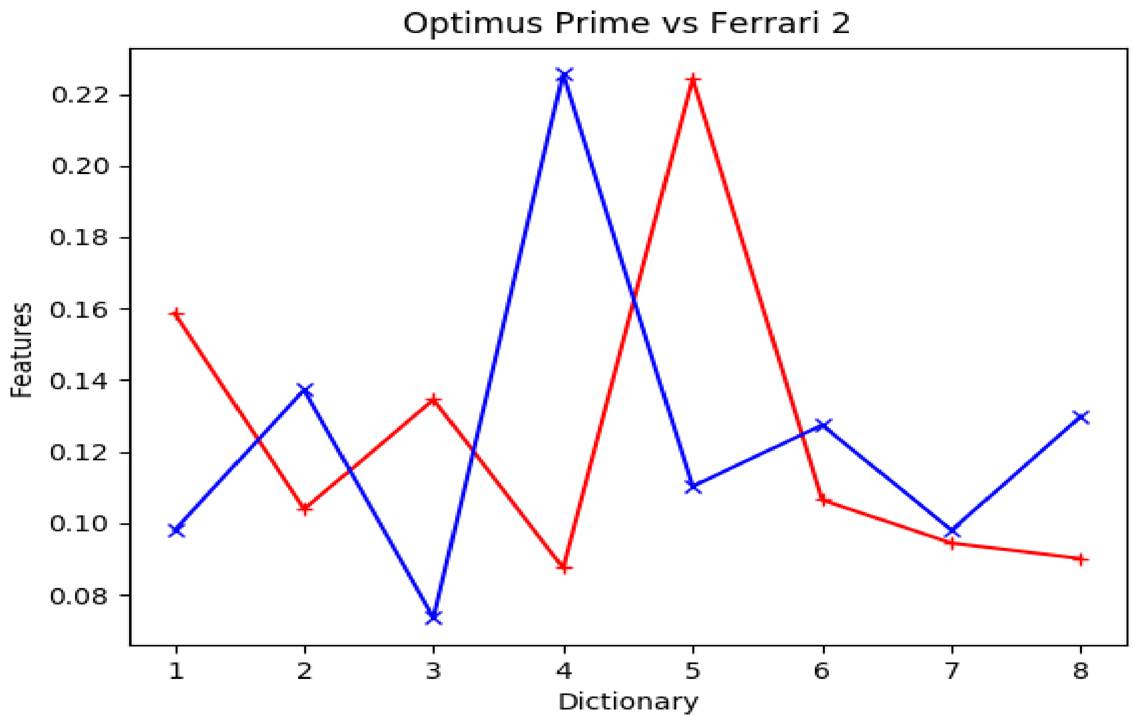
I used OpenCV functions and python for the question. The codewords were printed out, stored in excel and the histograms are formed.

III. Results and Discussion

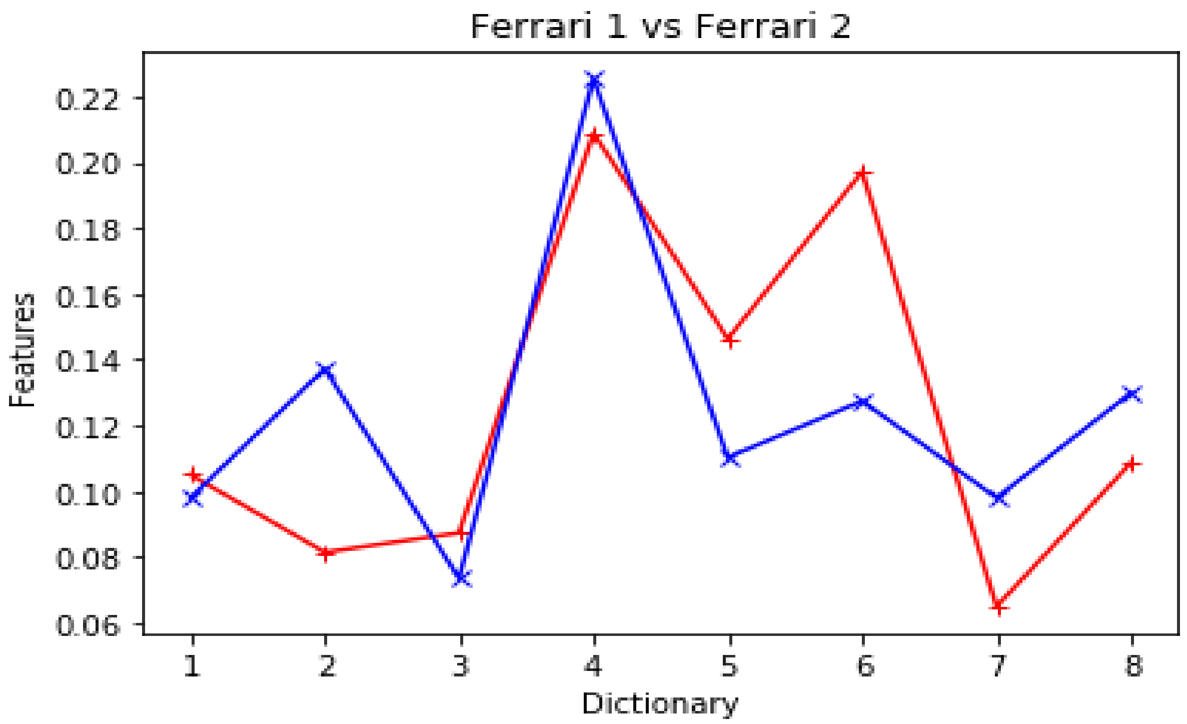
The similarity between the codewords of two images were plotted in graphs as follows. We can check for similarity by looking at how close the graphs are.



Comparison between Bumblebee and Ferrari 2

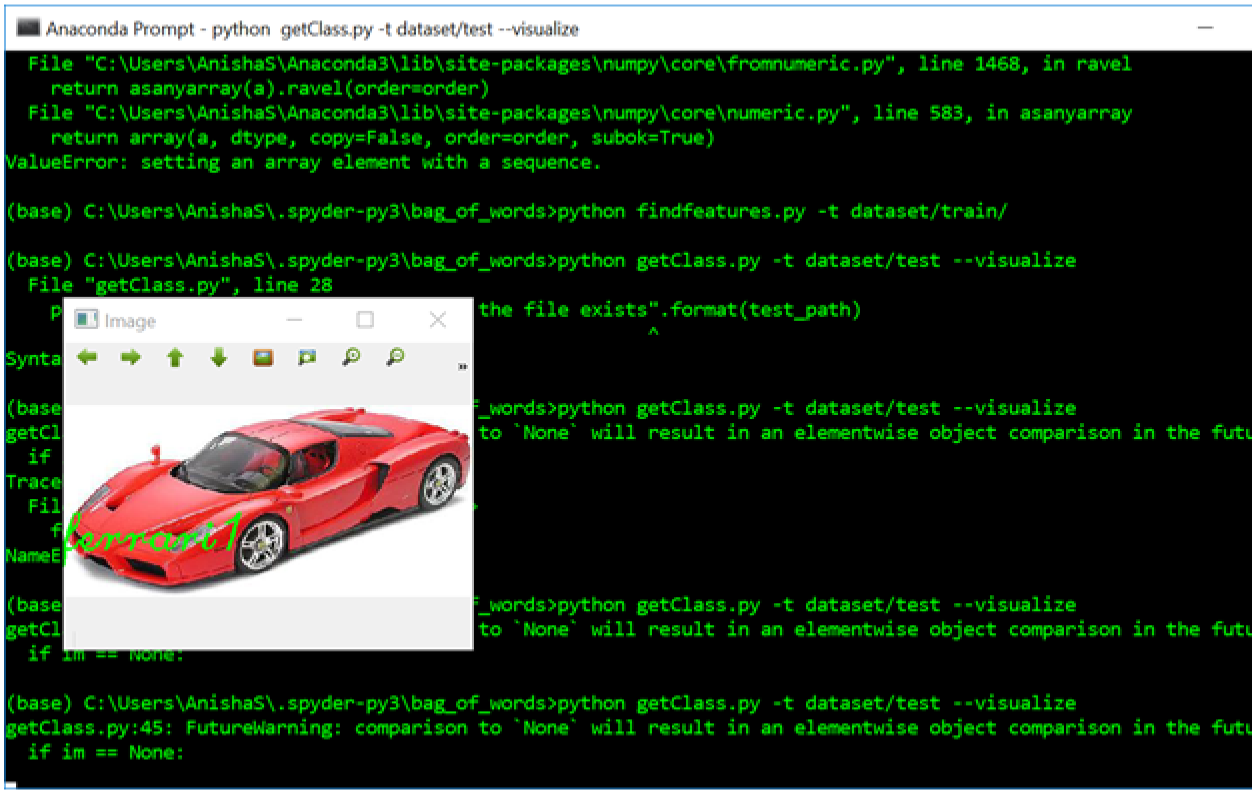


Comparison between Optimus Prime and Ferrari 2



Comparison between Ferrari1 and Ferrari 2

As we can see, the graphs of codewords for Ferrari 1 and Ferrari 2 are most similar. Hence the image Ferrari 2 is classified into class Ferrari 1.


Anaconda Prompt - python getClass.py -t dataset/test --visualize
Image
Ferrari1

```
  File "C:\Users\AnishaS\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py", line 1468, in ravel
    return asanyarray(a).ravel(order=order)
  File "C:\Users\AnishaS\Anaconda3\lib\site-packages\numpy\core\numeric.py", line 583, in asanyarray
    return array(a, dtype, copy=False, order=order, subok=True)
ValueError: setting an array element with a sequence.

(base) C:\Users\AnishaS\.spyder-py3\bag_of_words>python findfeatures.py -t dataset/train/

(base) C:\Users\AnishaS\.spyder-py3\bag_of_words>python getClass.py -t dataset/test --visualize
  File "getClass.py", line 28
    p                                        the file exists".format(test_path)
                                                                   ^
Synta

(base                                        _words>python getClass.py -t dataset/test --visualize
getCl                                        to `None` will result in an elementwise object comparison in the futu
  if
Trace
  Fil
    f
NameE

(base                                        _words>python getClass.py -t dataset/test --visualize
getCl                                        to `None` will result in an elementwise object comparison in the futu
  if im == None:

(base) C:\Users\AnishaS\.spyder-py3\bag_of_words>python getClass.py -t dataset/test --visualize
getClass.py:45: FutureWarning: comparison to `None` will result in an elementwise object comparison in the futu
  if im == None:
```

As can be seen by the resulting image, the Ferrari 2 car has been classified as Ferrari 1 as it shares the maximum similarity of histogram of codewords with Ferrari 1 image.

References:

1. Image Processing – William Pratt textbook
2. http://docs.opencv.org/3.2/doc/tutorials.html
3. https://github.com/pdollar/edges
4. David G. Lowe – “Distinctive image features from Scale Invariant Key Points” - University of British Columbia, Vancouver, Canada
5. P M Panchal, S R Panchal, S K Shah, “A comparison of SIFT and SURF”, IJIRCCE, Vol 1, Issue 2, April 2013
6. R. Unnikrishnan, C. Pantofaru, and M. Hebert, “Toward objective evaluation of image segmentation algorithms”, IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 6, pp. 929-944, Jun. 2007