Muthulakshmi Chandrasekaran

4486-1802-42

muthulac@usc.edu

EE569 Homework #3

March 28 ,2018

Problem 1 – Texture Analysis and Segmentation

Part A – Texture Classification

A.1 Motivation

In many computer vision problems, the major area of interest is the analysis of textures. A texture can be defined as a group of pixels that are almost in the same energy band. Various operations can be performed on textures like texture classification, texture segmentation etc, which are collectively called as texture analysis. There are numerous algorithms for texture analysis. Most of these algorithms involve extraction of the key descriptors of the image – also called as features. The most commonly used methods of texture analysis can be classified as spatial domain methods, or frequency domain methods. In texture classification, various textures are differentiated from each other using proper algorithms.

A.2 Approach

Texture Classification is a method where in the different textures are classified into various classes. There are two approaches to texture classification – supervised and unsupervised classification.

- In Supervised Classification, the classifier is trained using a pre – given dataset. After training, the classifier can be used to classify the textures.
- In Unsupervised Classification, there is no need of prior training for the classifier, and the classifier automatically classifies the textures.

The given problem consists of 12 input images – that are to be classified into four given textures.

The classification methods involve the following steps:

i. Feature Extraction:

For each input image, a 9 – dimensional feature vector is obtained, using the LAWS filters.

There is a total of nine $5 * 5$ LAWS masks, obtained using the following three LAWS filters.

$$E5 = \frac{1}{6}[-1 \quad -2 \quad 0 \quad 2 \quad 1]$$

$$S5 = \frac{1}{4}[-1 \quad 0 \quad 2 \quad 0 \quad -1]$$

$$W5 = \frac{1}{6}[-1 \quad 2 \quad 0 \quad -2 \quad 1]$$

The nine LAWS masks are E5E5, E5S5, E5W5, S5E5, S5S5, S5W5, W5E5, W5S5 and W5W5.

For the first image, each of these masks is applied on the image. So, there will be a nine – dimensional feature vector for each pixel in the input image.

ii. Feature Averaging:

The average energy of the resultant image is calculated as

$$Average\ Energy = \frac{\sum(pixel\ value)^2}{total\ number\ of\ pixels}.$$

So, there will be a vector of nine energies (one for each mask), for one image.

This is repeated for the rest of 11 images. Thus, because of feature extraction, there will be one nine – dimensional feature vector corresponding to each image.

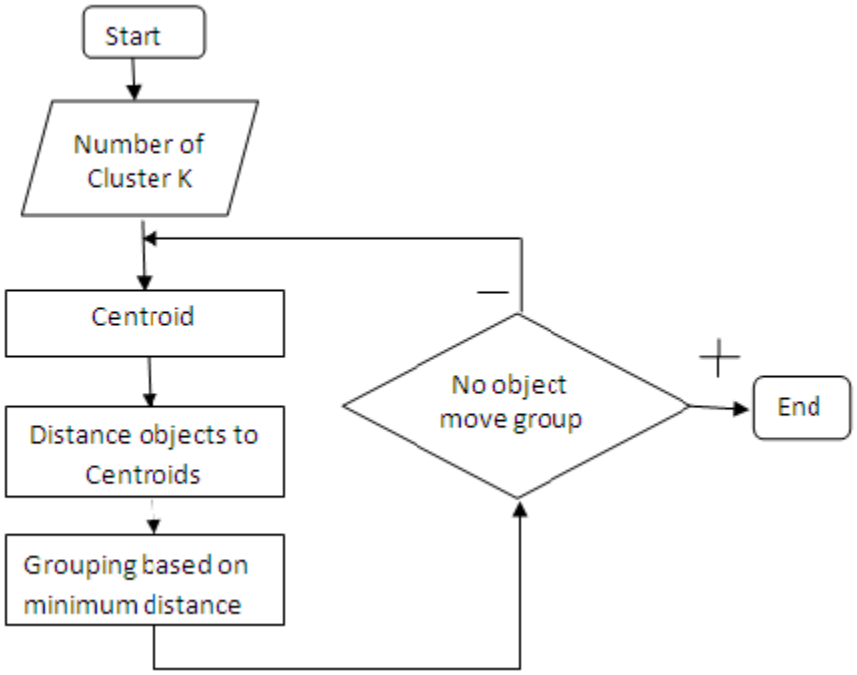
iii. Clustering:

Once the features are extracted, the textures are classified using the K – means classifier.

The K – Means Clustering Algorithm can be explained as follows:

1. If there are K classes, initialize K centroids, or simply centers. These centers have the same dimensions as the number of features. These initial centers could be randomly chosen,
2. For a given feature vector at a pixel location in an image, find the distance of each center to the point.
3. Find the corresponding class label for that pixel by taking the minimum of the distance values.,
4. Now, update the cluster centroids by taking the average of the points that lie in the corresponding cluster.
5. Repeat the steps 2 to 4 until the change in the centers of the clusters are negligible.
6. Now, give the corresponding label.

A simple flowchart of the K – means classifier is shown below [2].



The approach in C++ is as follows.

1. The input image is read into an array using the input file operations.

2. For this input image (say image1), the nine masks are calculated using the three basic masks – E5, S5 and W5. These nine masks are labelled as (say) $m1, m2, \dots, m9$.
3. Each mask is applied on the image1. The result will be a total of 9 images – one for each mask.
4. For every resulting image, the average energy of the image is calculated using the formula

$$Average\ energy = \frac{\sum(pixel\ value)^2}{total\ number\ of\ pixels}$$

   Hence there will be one energy value associated with each image. The nine energy values are put in an array – it is a feature vector that contains nine elements.
5. Steps 1 – 4 are repeated for twelve images. Hence the final feature array contains the energy values for twelve images, with each value containing nine features. In short, a two – dimensional array of 12 rows and 9 columns.
6. Then K – means clustering method is coded. There are four textures – hence four centroids or centers are defined, with each point taking a random value.
7. The distance of each of the feature vector with each of the four centers is found. The minimum distance is calculated, and each class is assigned a unique class label.
8. After every iteration, the centers are updated – the new centroid of a class is the average of the datapoints belonging to that class.
9. This updating is done until the difference between old and new centres is negligible.

A.3 Results

The twelve input images are shown in Figure 1.1.

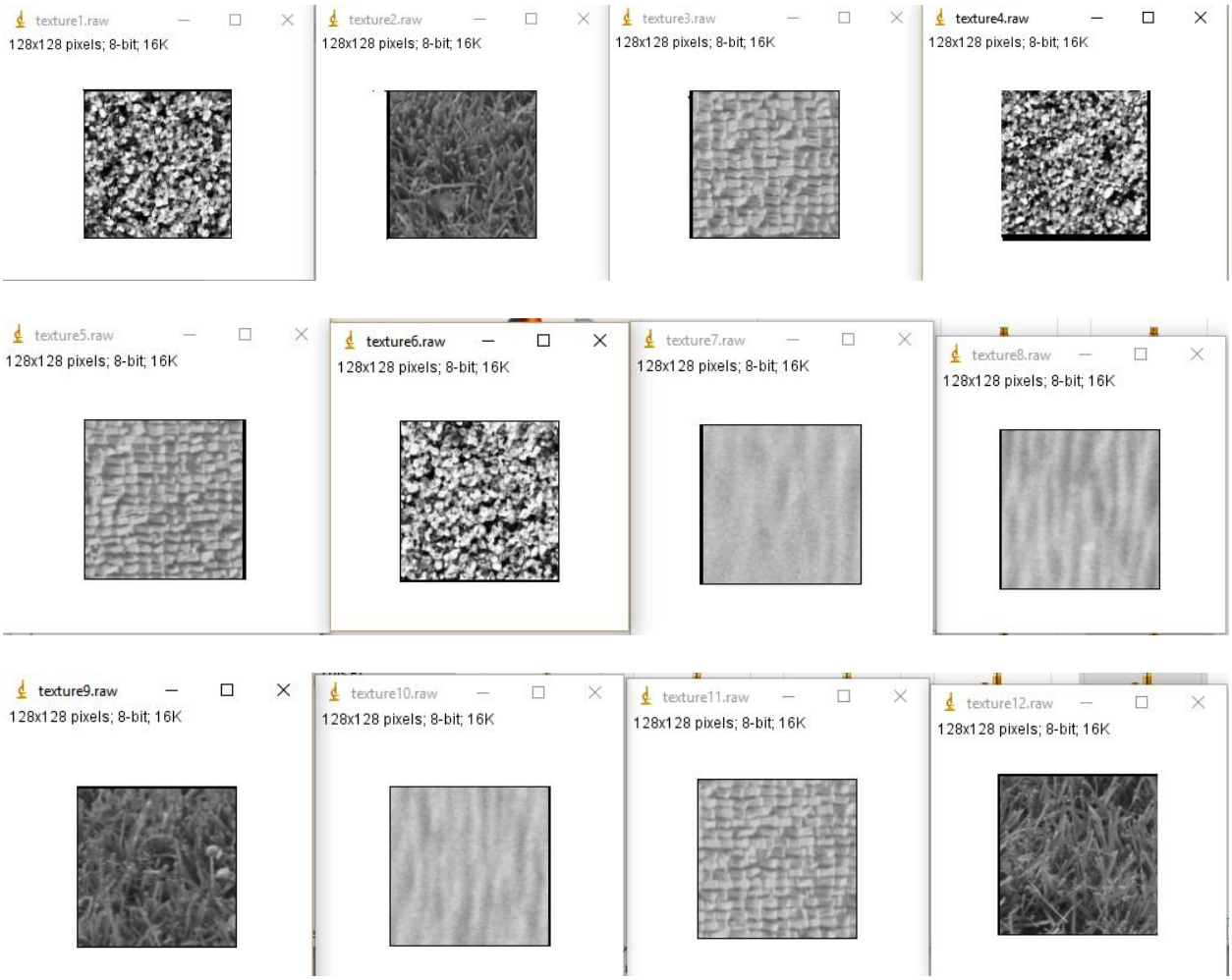


Figure 1.1. Input images for texture analysis and segmentation

These images are to be classified into four classes described as in Figure 1.2.

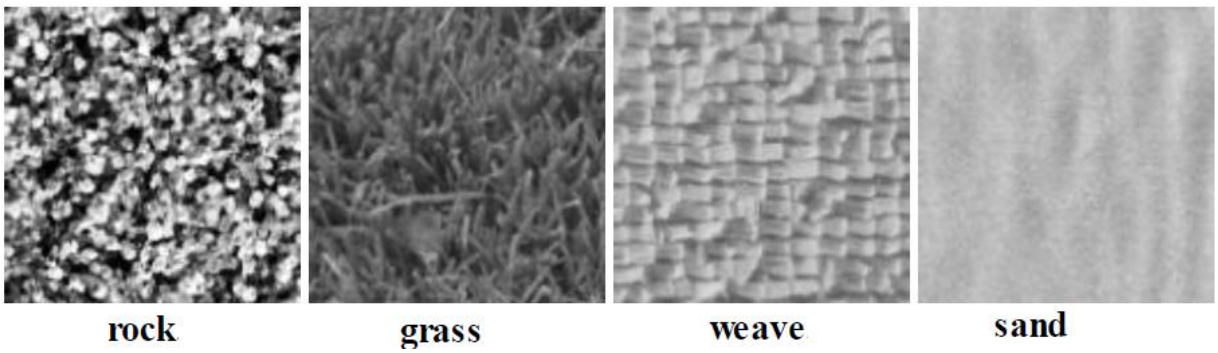


Figure 1.2 Four types of textures

The nine LAWS filters are applied to the image to get the feature vectors of the twelve images as shown in Figure 1.3.

```
Image - Features :

 Image 1 : 346.806 187.429 61.3248 185.743 119.11 43.825 53.936 38.6919 15.4197

 Image 2 : 0.53691 0.739194 0.32782 0.518149 0.713365 0.316365 0.23027 0.317026 0.140596

 Image 3 : 25.5061 10.6254 2.88787 13.1105 5.77346 1.67207 4.1343 1.87301 0.557259

 Image 4 : 397.064 217.721 67.8413 232.046 142.295 48.23 67.6174 46.7429 17.1597

 Image 5 : 33.5296 12.8416 2.93347 17.1328 6.8336 1.62237 5.01304 2.09863 0.500623

 Image 6 : 370.988 211.596 69.6187 207.815 135.716 50.3176 62.8354 46.1049 18.5089

 Image 7 : 2.63196 2.33063 1.20139 2.38337 2.30478 1.24136 1.21872 1.25387 0.701701

 Image 8 : 2.85445 2.36298 1.24122 2.39597 2.3506 1.30405 1.1919 1.26193 0.727152

 Image 9 : 28.1618 14.229 3.28561 13.3343 7.61957 1.94687 3.64796 2.30515 0.645901

 Image 10 : 2.40568 1.55743 0.487189 1.9736 1.44827 0.482768 0.917963 0.717202 0.254904

 Image 11 : 17.657 7.61467 2.23987 7.65389 3.28783 1.02466 1.72015 0.761089 0.247638

 Image 12 : 52.7186 36.8178 12.0007 28.5384 24.1175 9.22364 8.83249 8.3716 3.71547
```

Figure 1.3 Feature Vectors of the twelve images

The final centroids obtained for the four classes are shown in Figure 1.4.

```
 Final centres :
Centroid for class 1 :371.619  205.582  66.2616  208.535  132.374  47.4575  61.4629  43.8466  17.0294
Centroid for class 2 :2.10725  1.74756  0.814406  1.81777  1.70425  0.836137  0.889713  0.887508  0.456088
Centroid for class 3 :52.7186  36.8178  12.0007  28.5384  24.1175  9.22364  8.83249  8.3716  3.71547
Centroid for class 4 :26.2136  11.3276  2.8367  12.8079  5.87861  1.56649  3.62886  1.75947  0.487855
 Number of loops : 20
```

Figure 1.4 Final centres obtained

The classification of textures based on K – means clustering is done. Figure 1.5 shows the output when the input images are texture1.raw, texture2.raw, texture12.raw.

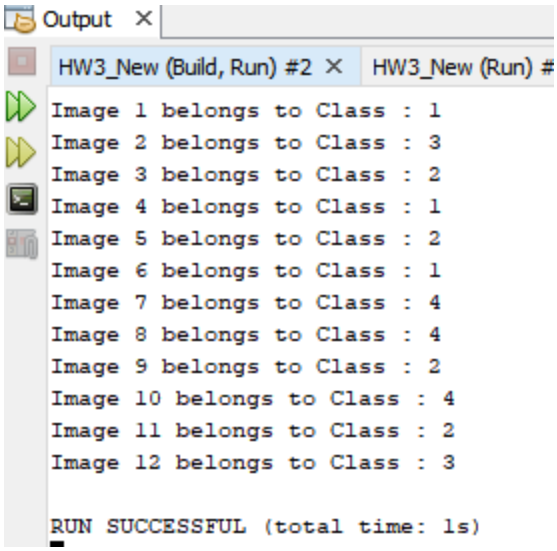


Figure 1.5 Classification based on Input images texture1.raw, texture2.raw, .., texture12.raw.

Classification of textures based on K – means based on a separate set of input images.

Image texture14.raw is included in the set of input images except for image texture9.raw, which is wrongly classified. Figure 1.6(a) shows texture14.raw. Figure 1.6(b) shows the output obtained for the new set of input images.

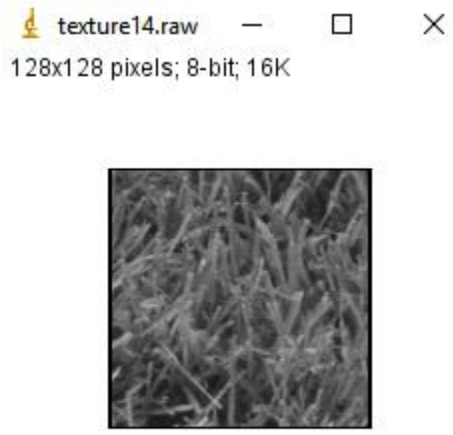


Figure 1.6(a) Texture14.raw input image

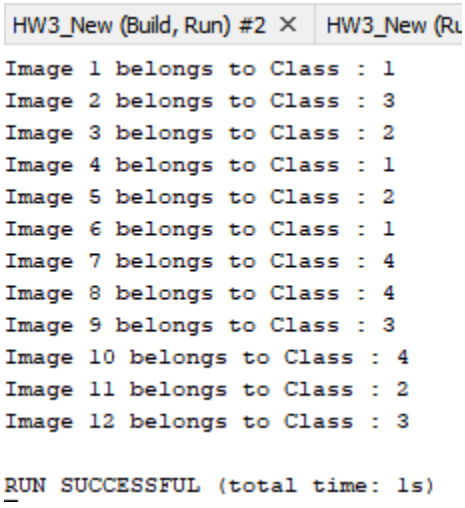


Figure 1.6(b) Classification for the new set of input images.

The interclass variance is obtained as shown below.

```
Intraclass variances :
11068.3  0.990404  0.909872  0.112135  1377.4  1289.7  0.529215  65.8458  67.7384
```

A.4 Discussion

The result of this program is just the class labels, which are integers. Upon visual inspection, for the set of input images 1 to 12, Images 1,4 and 6 belong to the same class, Images 2,9,12 belong to the same class; Images 3,5,11 belong to the same class and Images 7,8,10 belong to the same class. But the results are slightly misclassified.

If a separate set of input images are used, the classification is found to be right.

To determine which measure has the strongest discriminant power, variance of each feature with respect to each other is computed. For a strong discriminant power, the interclass variance should be lower, and the interclass variance should be higher. The intraclass variance is calculated using the formula.

The interclass variance can be computed by the covariance matrix. The principal diagonal elements are found to be the variance of each feature. The feature which has the largest variance has the weakest discriminant power, since it varies too much, and has high chances of occurrences.

From the calculations, it is observed that feature 1 has the highest discriminant power or the highest intraclass variance, and feature 4 has the least discriminant power or highest variance. Hence, the feature $L3^T L3$, corresponding to the filter has less significance, and is not a useful feature.

Part B – Texture Segmentation

B.1 Motivation

A texture can be defined as a group of pixels that are almost in the same energy band. Various operations can be performed on textures like texture classification, texture segmentation etc., which are collectively called as texture analysis. There are numerous algorithms for texture analysis. Most of these algorithms involve extraction of the key descriptors of the image – also called as features. The most commonly used methods of texture analysis can be classified as spatial domain methods, or frequency domain methods. In statistical approach to texture analysis is done using the spatial distribution of image pixels. A structural approach analyses textures as hierarchies of spatial arrangements. A probabilistic approach makes use of the stochastic models.

In texture classification, various textures are differentiated from each other using proper algorithms. There are two basic steps to any texture analysis – feature extraction and feature classification. There are different algorithms for texture analysis – most of them differ in the methods of either feature extraction or classification.

B.2 Approach

There are two basic methods for classification – supervised approach and unsupervised approach. The supervised classifier is trained using a pre – given dataset, and then used to classify data. In unsupervised classifier, no prior training is needed for the classifier. The basic steps used in Texture Segmentation are as follows.

i. LAWS Feature Extraction

The LAWS filters are used to get the filtered images. Here $3 * 3$ LAWS filters are used. The three basic laws filters are

L3 = {0.1666*1 , 0.1666*2 , 0.1666*1}

E3 = {0.5*1 , 0, 0.5*-1}

S3 = {0.5*1, 0.5*-2 , 0.5*1}

Using these three basic filters, nine filters are obtained.

ii. Energy Computation

As in the previous part, nine different images are obtained by using each mask. Here one image contains different textures. Hence energy for each pixel, for each mask, energy values are calculated. Hence each pixel has nine energy values – or simply a feature vector of dimension nine.

iii. Normalization

On computing the mean of the kernels (or the masks), it is observed that the L3L3 kernel only has a non – zero mean. Hence this mask can be used for normalizing the other energy values.

For example, if the feature vector is

$$[L3L3\ L3E3\ L3S3\ E3L3\ E3E3\ E3S3\ S3L3\ S3E3\ S3S3]$$

After normalization, this becomes

$$\left[1\ \frac{L3E3}{L3L3}\ \frac{L3S3}{L3L3}\ \frac{E3L3}{L3L3}\ \frac{E3E3}{L3L3}\ \frac{E3S3}{L3L3}\ \frac{S3L3}{L3L3}\ \frac{S3E3}{L3L3}\ \frac{S3S3}{L3L3}\right]$$

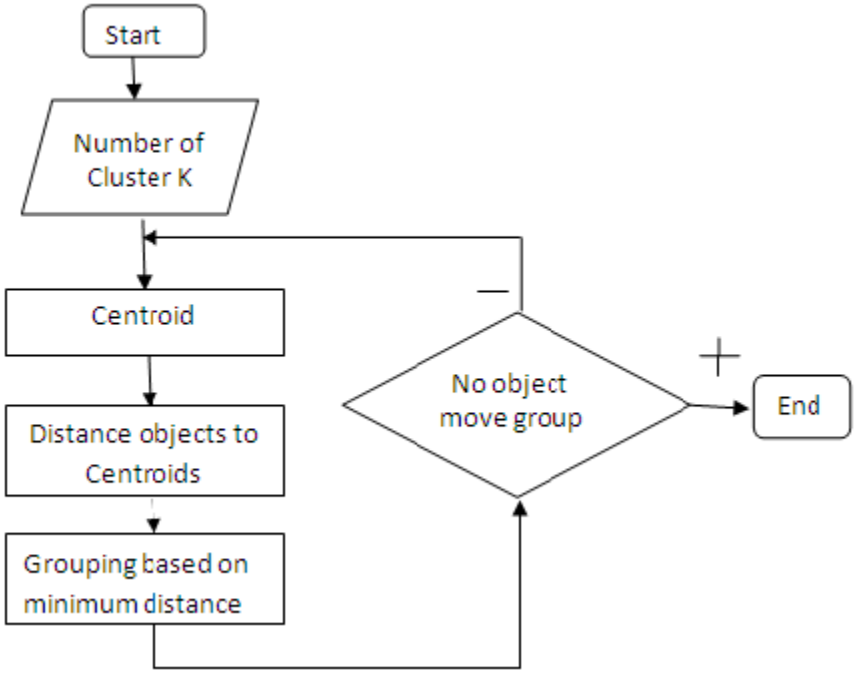
iv. Segmentation

K – Means algorithm is used to classify the textures. The K – means algorithm can be explained as follows.

1. If there are K classes, initialize K centroids, or simply centres. These centres have the same dimensions as the number of features. These initial centres could be randomly chosen,
2. For a given feature vector at a pixel location in an image, find the distance of each center to the point.
3. Find the corresponding class label for that pixel by taking the minimum of the distance values.

4. Now, update the cluster centroids by taking the average of the points that lie in the corresponding cluster.
5. Repeat the steps 2 to 4 until the change in the centres of the clusters are negligible.
6. Now, give the corresponding label.

A simple flowchart of the K – means classifier is shown below [2].



Here there are six classes. Hence the six textures can be represented by six gray levels in the output image.

The approach in C++ is as follows:

1. The input image is read into an array using the input file operations.
2. For this input image, the nine LAWS masks are calculated using the three basic $3 * 3$ masks – L3, E3 and S3. These nine masks are labelled as (say) $m1, m2, \dots, m9$.
3. Each mask is applied on the input image. The result will be a total of 9 images – one for each mask.

4. For every pixel in the resulting image, the average energy of the pixel is calculated using different window sizes, say $N = 11\, and\, N = 13$, This is done for each mask. Average is calculated using the formula

$$Average\ energy = \frac{\sum(pixel\ value)^2}{total\ number\ of\ pixels}$$

   Hence there will be one energy value associated with each mask for each pixel. The nine energy values of one pixel are put in an array – it is a feature vector that contains nine elements.
5. The feature vector is then normalized using the energy value for $L3L3$.
6. Then K – means clustering method is coded. There are six textures – hence six centroids or centres are defined, with each point taking a random value.
7. The distance of each of the feature vector with each of the six centres is found. The minimum distance is calculated, and each class is assigned a unique class label.
8. After every iteration, the centers are updated – the new centroid of a class is the average of the datapoints belonging to that class.
9. This updating is done until the difference between old and new centres is negligible.
10. Once the K – means algorithm is completed, each pixel with belong to either of the six classes – labelled either as 1 or 2 or 3 or 4 or 5 or 6.
11. Each of the class label is assigned a unique gray scale value – 0, 51, 102, 153, 204 and 255 respectively to denote the six different textures.

B.3 Results

In this problem, the textures are present in a single image. There are 6 different textures in the image, which is classified into each class. The nine gray – scale images obtained by LAWS filters are shown in Figure 1.7(a) to Figure 1.7(i).

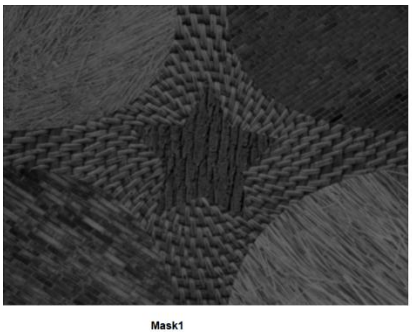


Figure 1.7(a) Output obtained by using $L3^T L3$

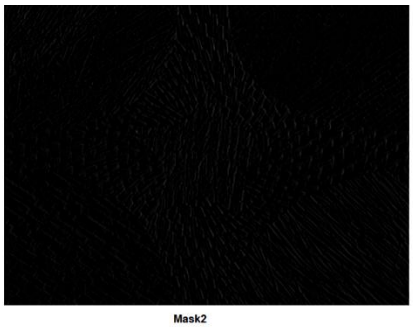


Figure 1.7(b) Output obtained by using $L3^T E3$

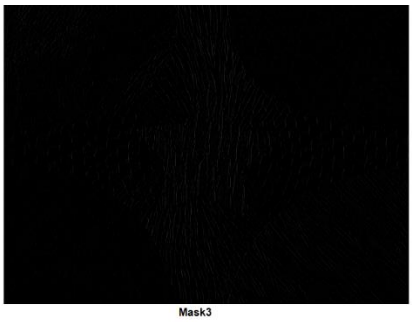


Figure 1.7(c) Output obtained by using $L3^T S3$

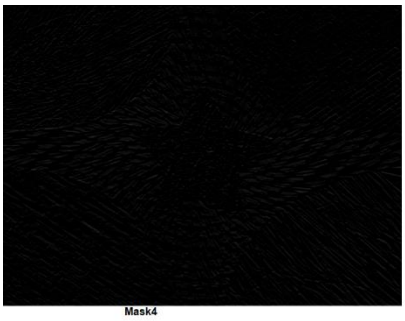


Figure 1.7(d) Output obtained by using $E3^T L3$

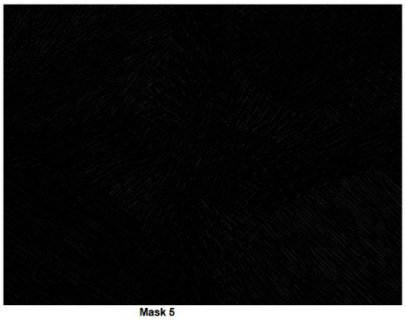


Figure 1.7(e) Output obtained by using $E3^T E3$

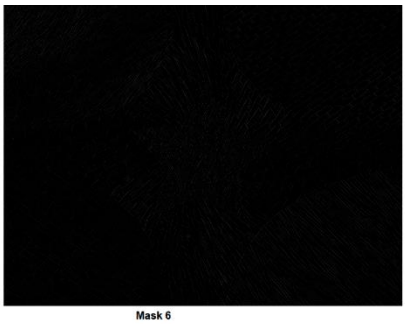


Figure 1.7(f) Output obtained by using $E3^T S3$

Figure 1.7(g) Output obtained by using $S3^T L3$

Figure 1.7(h) Output obtained by using $S3^T E3$

Figure 1.7(i) Output obtained by using $S3^T S3$

The resulting images for different window sizes are shown in Figure 1.8.

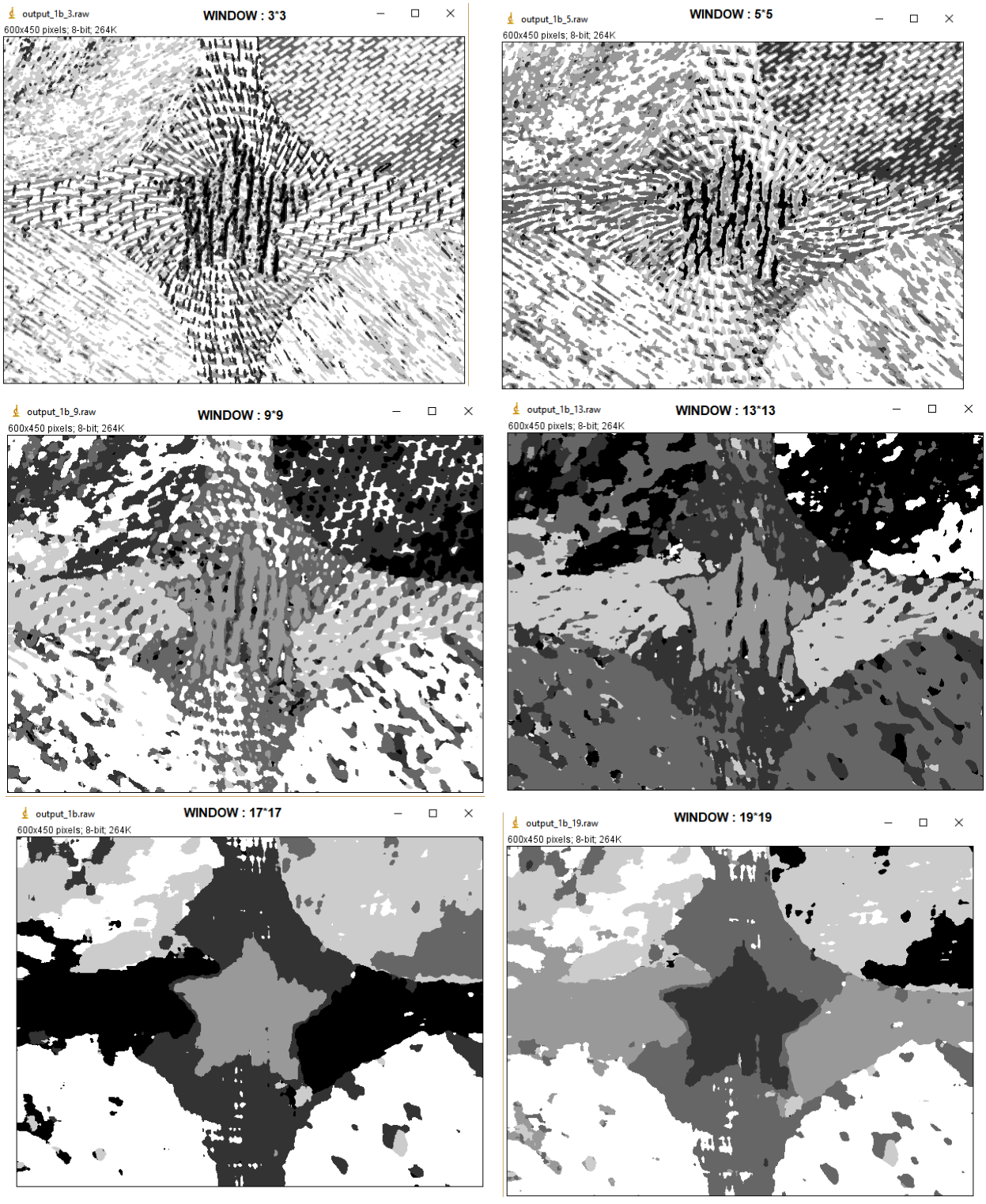


Figure 1.8 Output images for different window sizes.

The cluster size is kept constant at 6. For a window size of 17, the result is found to be better as shown in Figure 1.9. For each class from 1 to 6, it is represented by a gray scale value – one of 0,51,102,153,204,255.

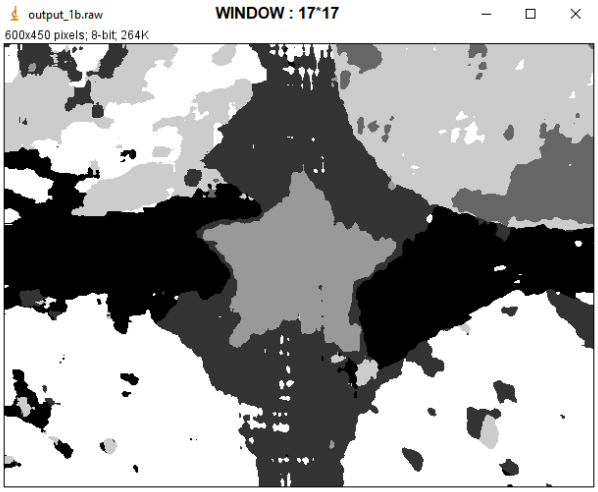


Figure 1.9 Texture Classification

Figure 1.10 shows the histogram plot of the output image, indicating the presence of six gray scale values.

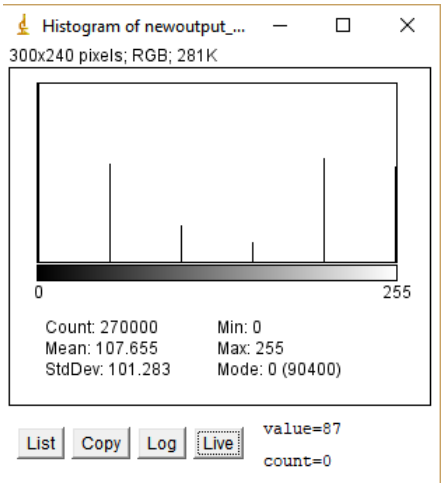


Figure 1.10 Histogram showing the six gray scale values

B.4 Discussion

The energy is computed using different window sizes. It is observed that for a window of smaller size, say 3*3, the output texture segmentation is not good. As the window size is increased, the texture segmentation is found to improve.

But as the window size increases, smoothness is also found to increase, which misses out the major details in the image. Hence a tradeoff is observed between smoothness and accuracy.

It can also be observed that as the LAWS filter dimensions become 3*3 from 5*5 (compared to Part A), the variance of each feature with respect to one another varies. But though the feature vector changes, the feature $L3^T L3$ is found to have the least discriminant power, and hence it is not a useful feature.

K – means clustering is used to classify the textures into different classes. The accuracy of K – means clustering is largely dependent on the choice of cluster centres and the number of clusters. In the algorithm, the centres are chosen at random, and hence there are variations in accuracy.

Thus, the given image is segmented into different classes using K – means clustering.

Part C – Bonus: Texture Segmentation using PCA

C.1 Motivation

Principal Component Analysis(PCA) is used for converting a set of possibly correlated variables into uncorrelated variables using orthogonal transformations. PCA is a dimensionality reduction algorithm. PCA is extensively used in Image processing in various fields. It can be used for denoising an image, for image segmentation to classify the textures in an image etc. There are different Texture Segmentation methods that give approximate results to classifying textures. To improve the results of segmentation, one possible suggestion is using PCA.

C.2 Approach

In this problem, the input image is of size 600*450. In accordance with the steps in classifying the textures rightly, the first step is the application of LAWS filters. If a $5 * 5$ LAWS filter is used, there are 25 masks. Hence each image is operated upon 25 times – one for each mask. PCA can be used in dimension reduction – reducing it to a lesser number of significant masks or features – and operating upon them.

The approach to PCA using SVD is shown below [ref Discussion Notes].

- Consider the feature matrix X of size n*m, n = #features and m = # datapoints
- X_i → i[th] column of the matrix X; Compute mean $mX(of\ size\ n * 1) = \frac{\sum_{i=1}^{i=m} X_i}{m}$
- Compute zero mean feature matrix $ZX\ (of\ size\ n * m)$ as $Zxi\ =\ Xi\ -\ mX$
- Compute SVD of ZX as $[U, S, V'\ ] = svd(ZX); + U \rightarrow n * n\ ;\ S \rightarrow n * m\ and\ V \rightarrow m * m$
- Dimensionality Reduction: First n' significant features are chosen i.e. retaining first n' columns of U and S.
- Compute the reduced transformation matrix UR, of size $n' * n$.
- If RX is the reduced dimensionality matrix, then $RX\ =\ U * R'\ *\ ZX$.

This RX now contains the reduced set of feature vectors.

This is implemented in C++, with PCA function alone used in Matlab.

i. The input image is obtained using file operations.
ii. 5*5 LAWS filters are used as operators on the input image to get 25 masks.
iii. For each mask, the average energy of each pixel is computed using the formula $\frac{\sum(pixel\ value)^2}{total\ number\ of\ pixels}$.
iv. Now, the feature vector is obtained. This vector is written to a file.
v. This file is read in Matlab, and PCA is done.
vi. The results of the file are stored in another text file, in Matlab.
vii. This file is again read in C++, and the K – means clustering algorithm is applied.

Thus, PCA could be used for texture segmentation.

C.3 Results

The better output from Part 1(b) is taken as the specifications. Here the window size is taken to be, and the LAWS filter computation is for 5*% matrices.

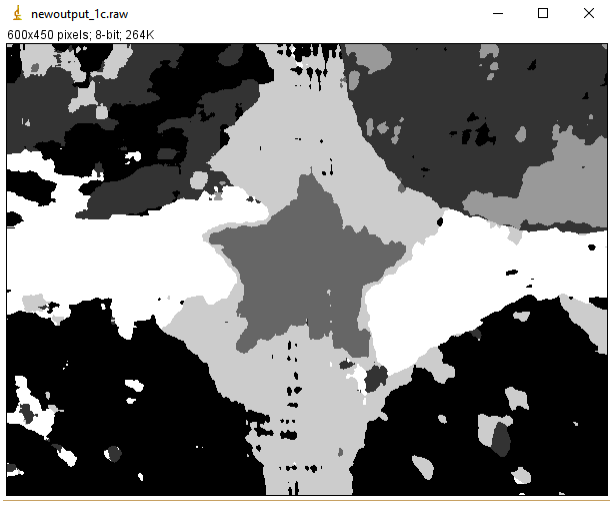


Figure 1.11 PCA Output

The output image obtained is as in Figure 1.11.

C.4 Discussion

There are many post processing methods to improve the texture segmentation results.

i. Comparing the results of Part 1(b) and Part 1(c), the images are almost visually alike, since the dimensions are reduce to only 2. Observing carefully, PCA is found to give a slightly poor result. This is because of the extensive use of LAWS filters – wherein the processing data becomes too huge.
   Finding the right threshold for significant and less significant features is not known. For that a proper algorithm is needed. The number of reduced features is taken here as 5. Hence there could be more loss of data, even if there is a dimensionality reduction.

ii. Another method that could be used for improving the results of image segmentation is the use of Gabor filters – that are like human eye. The Gabor filter is a harmonic function multiplied by Gaussian function, and can provide optimal

resolution in both space and spatial domains. These could be used for solving more complex textures [3].

iii. Here K – means clustering is used for thresholding, instead the other methods could be tried – like Otsu's threshold (or maximum variance) method, Markov's models etc., Histogram – based methods etc.

References:

1. William K Pratt, "Digital Image Processing ", Edition 4.

2. Muhammad Sohaib, Ihsan – ul – Haq and Qaisar Mushtaq, "Dimensionality Reduction of HyperSpectral Image Data Using Band Clustering and selection through K – Means based on Statistical Characteristics of Band Images ", International Journal of Advanced Computer Science, Vol 2, No.4, PP 146 – 151, April 2012.

3. Vaijinath V. Bhosle, Vrushsen P. Pawar," Texture Segmentation: Different Methods ", International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-3, Issue-5, November 2013

Problem 2 – Edge Detection

Part A – Basic Edge Detectors

A.1 Motivation

Edges are a crucial factor that helps to distinguish structure and properties of an image. They mostly signify the local changes in an image. An edge can be defined as a local change in the image, particularly in terms of intensity of the image pixels. This could be a discontinuity in the intensity of the image, or in the first derivative of intensity. These discontinuities are abrupt changes in the intensity, that often represent boundaries in an image. Edge detection algorithms are used in extracting the edges from the image.

A.2 Approach

There are various kinds of edge detection algorithms – Classical, Zero Crossing, Laplacian of Gaussian etc.

Sobel Edge Detector:

In classical methods of edge detection, $N * N$ operators (or masks) are used on the image. These filters are sensitive to large gradients at the edges, and returns zero values in the non – uniform regions. There are different operators that can be used to detect vertical, horizontal or diagonal edges. The primary component used here for detection of edges is the gradient operator. An image is converted to its grayscale values, and the changes in the adjacent grayscale values are approximated to gradient. The gradient is defined as

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

The most commonly used gradient – based operator is the Sobel Operator. Consider a pixel $a(i,j)$ as shown below.

a_0	a_1	a_2
a_7	$a(i,j)$	a_3
a_6	a_5	a_4

The Sobel operator consists of two $3 * 3$ operators as shown below, which operates on the above pixel neighborhood.

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

Vertical Mask

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

Horizontal Mask

By convolving the above masks in two directions $x \; and \; y$, the gradient is obtained as $G_x \; and \; G_y$ respectively.

The magnitude and the direction of the gradient is calculated as

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

The approach in C++ is as follows:

1. The input images Boat. Raw and Boat_noisy.raw are obtained as input using file operations.
2. For each of the images, the horizontal and vertical masks are applied to obtain the gradient images.
3. The gradient images can contain both positive and negative values, that correspond to edge detection. To view the gradients as images, these values are normalized to the range $[0,255]$.
4. The final image obtained by the Sobel filter is the magnitude of the gradient operation. For each pixel, the resultant value is given by

$$G = \sqrt{G_x^2 + G_y^2}$$

5. Various thresholds are applied to the Sobel filter and compared. Using the threshold values, the edge map is obtained. An edge map contains either 0 or 255. This edge map is written to the output raw image.

Laplacian of Gaussian Edge Detector:

Laplacian operator is a measure of the second spatial derivative of an image. The output of Laplacian operator gives the intensity changes and hence edges are detected using Laplacian operator. The Gaussian operator is applied before the Laplacian operator. An image is smoothened using Gaussian operator, and this reduces the high – frequency noise components in the image. Simply, Gaussian for smoothening and Laplacian for edge detection together is the Laplacian of Gaussian operator.

This is a second derivative filter. An edge is detected if there is a zero crossing, and a corresponding peak in the first derivative.

- The actual input image is first smoothened using Gaussian operator.
- Smoothening reduces the noise, and spreads the image. Hence, locally maximum gradient is taken into consideration – zero crossings of the second derivative achieves this.
- The edges are detected when there is a zero crossing and a corresponding first derivative above a certain threshold. The output of the LOG operator is obtained by the convolution operation as

$$h(x,y) = \nabla^2[\, g(x,y) * f(x,y)\,]$$

Using the derivative rule,

$$h(x,y) = \nabla^2[\, g(x,y)] * f(x,y)$$

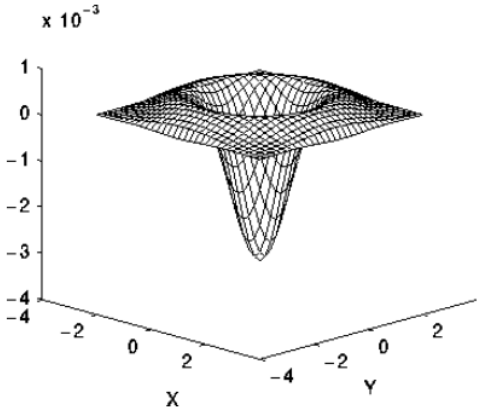
Or simply

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

An example of a $5 * 5$ Laplacian of Gaussian Mask could be

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

The LOG function looks as shown on the right.



The approach in C++ is as follows.

1. The input images Boat.raw and Boat_noisy.raw are obtained using file operations.
2. The LOG operator co – efficients of a window size N and σ is obtained from Matlab using the `fspecial` command.
3. Then the LOG operator is obtained on each input image.
4. Since the resultant images contain both positive and negative values, the values are normalized to [0,255] to obtain the resultant images.
5. Then the LOG – filtered image is divided into three thresholds using a suitable algorithm. Each region is given a different value.
6. Then edge map of each of the images is obtained – by zero crossing operation.

A.3 Results

The input images for the edge detection are Boat.raw and Boat_noisy.raw, as shown in Figure 2.1(a) and Figure 2.1(b).

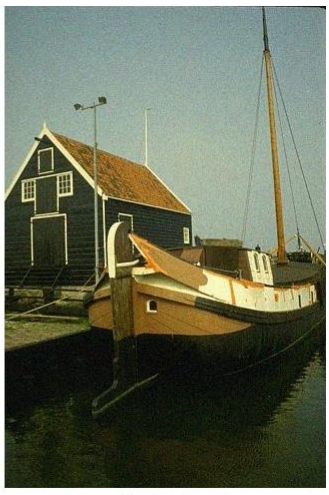


Figure 2.1(a) Boat.raw image **Figure 2.1(b) Boat_noisy.raw**.

The gray – scale images are found using the luminosity method, and are shown in Figure 2.2(a) and Figure 2.2(b) respectively.

Figure 2.2(a) Gray scale image of Boat.raw

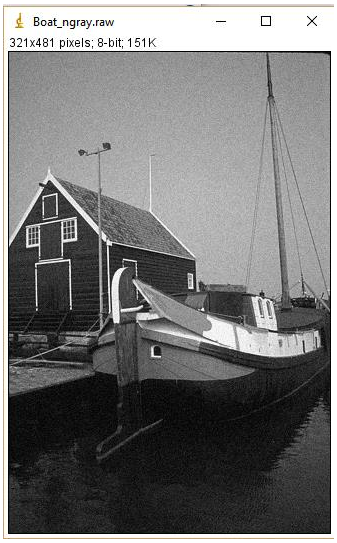


Figure 2.2(b) Gray scale image of Boat_noisy.raw

The results for the Sobel Edge Detector is shown below. The x – gradient and y – gradient images, that are normalized to [0,255] are shown in Figure 2.3 and Figure 2.4 for the two input images respectively.

Figure 2.3(a) x – gradient of Boat.raw

Figure 2.3(b) y – gradient of Boat.raw

Figure 2.4(a) x – gradient of Boat_noisy.raw

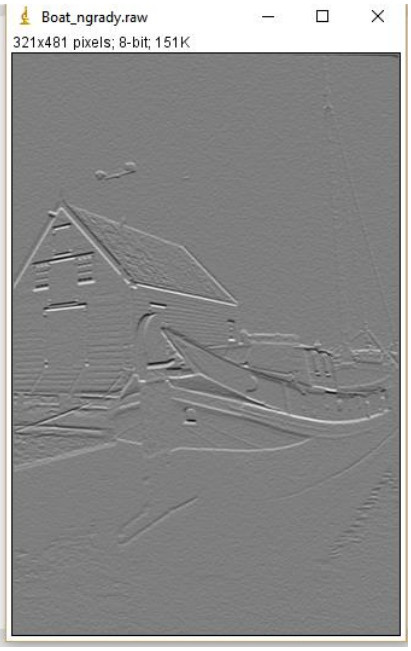


Figure 2.4(b) y – gradient of Boat_noisy.raw

The output gradient map is obtained as shown in Figure 2.5 and 2.6 respectively.

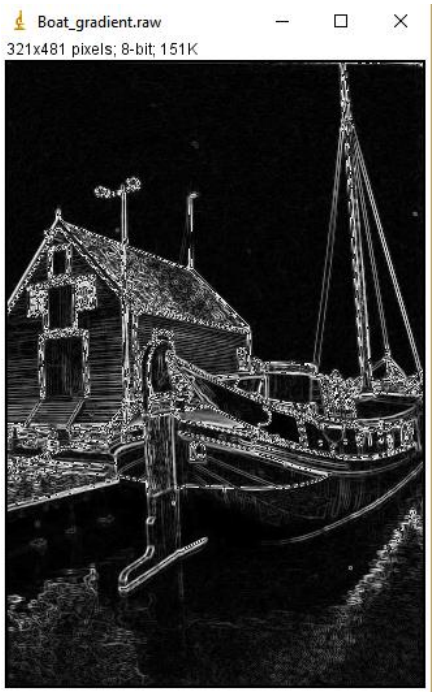


Figure 2.5 Output Gradient Map for Sobel edge detection for Boat.raw

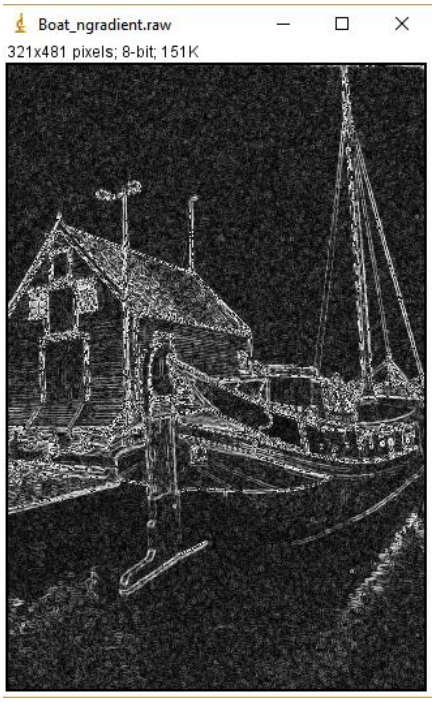
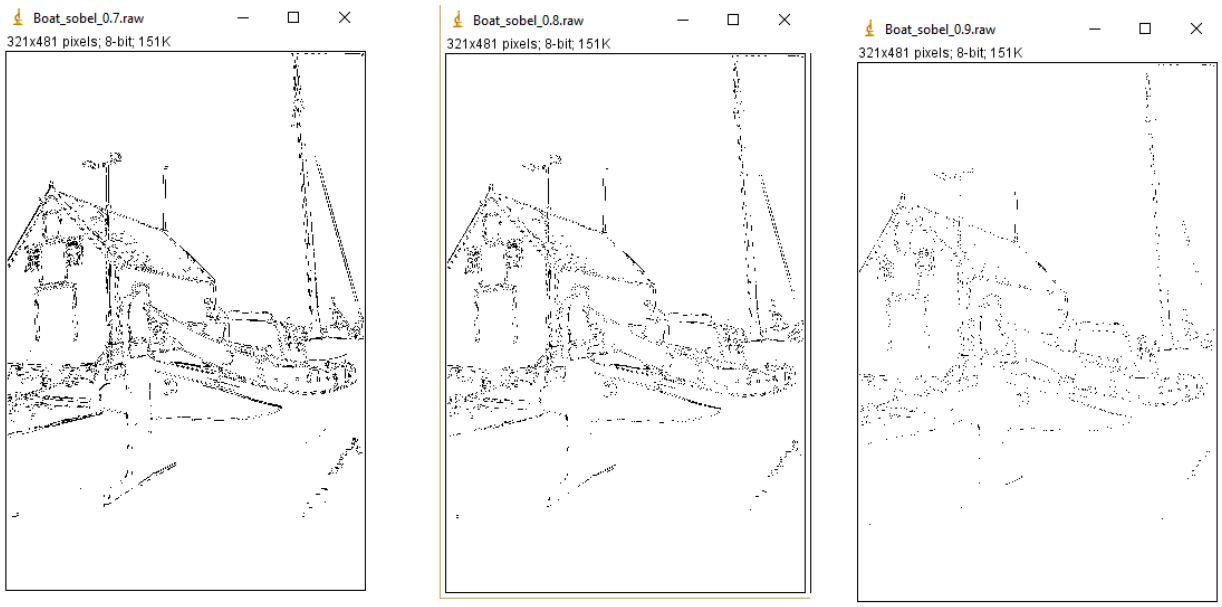


Figure 2.6 Output Gradient Map for Sobel edge detection for Boat_noisy.raw

For different values of threshold, the edge maps for Boat.raw are obtained as in Figure 2.7. The edge maps are made on a white background, which makes it easy to compare the outputs of all edge detectors. The edge maps are based on the condition that:

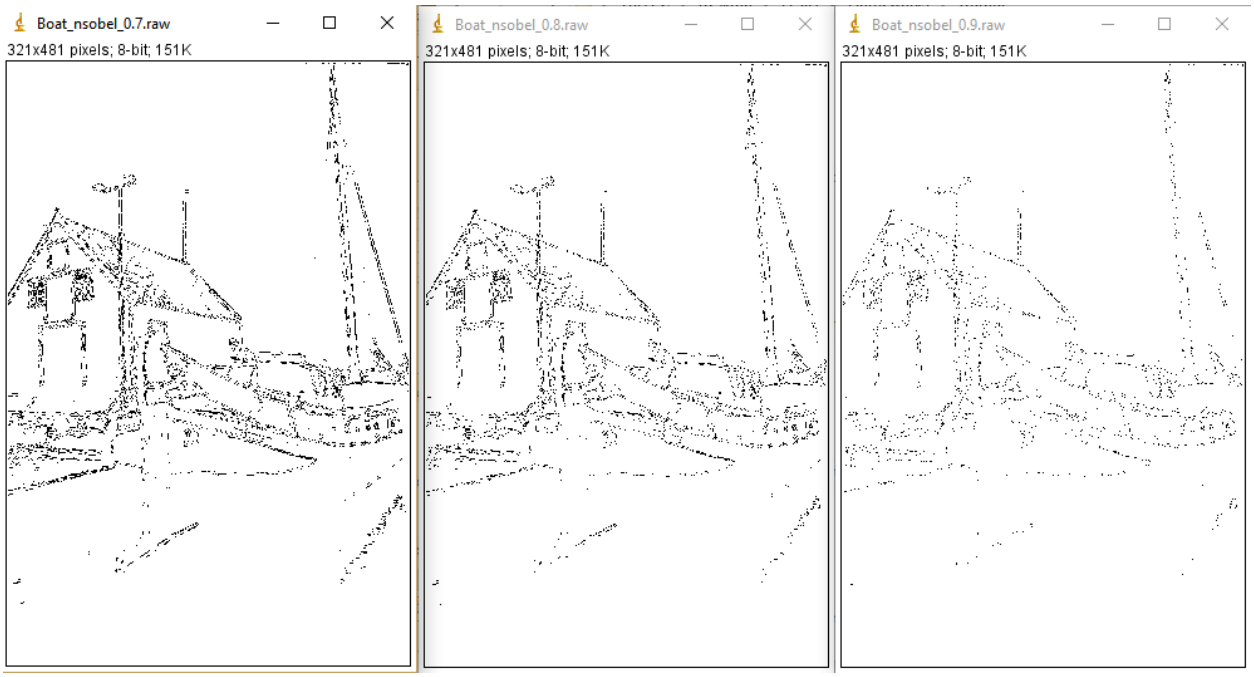
```
For all Pixels < (Percentage * 255)
     Assign a value 255;
Else
     Assign a value 0;
```



Threshold = 70 % **Threshold = 80%** **Threshold = 90%**

Figure 2.7 Edge maps for different thresholds for Boat.raw

For different values of threshold, the edge maps for Boat_noisy.raw are obtained as in Figure 2.8.



Threshold = 70 % **Threshold = 80%** **Threshold = 90%**

Figure 2.8 Edge maps for different thresholds for Boat_noisy.raw

The output for the LOG filter are as follows.

The normalized LOG responses are shown in Figure 2.9 for the two images.

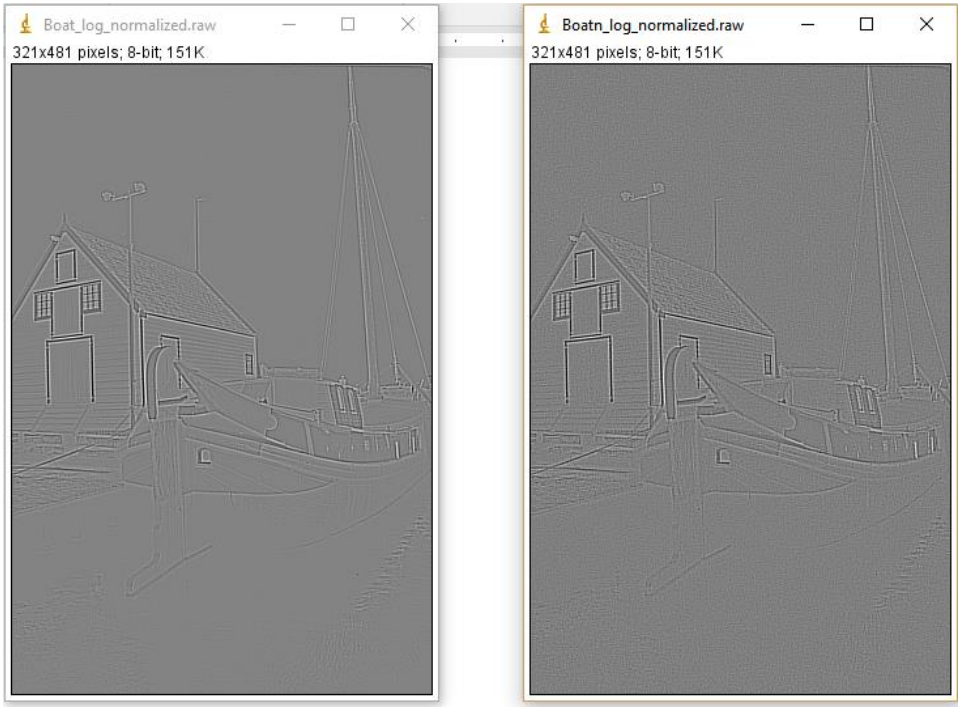


Figure 2.9(a) Normalized LOG response for Boat.raw

Figure 2.9(b) Normalized LOG response for Boat_noisy.raw

The ternary maps for the images are shown in Figure 2.10.

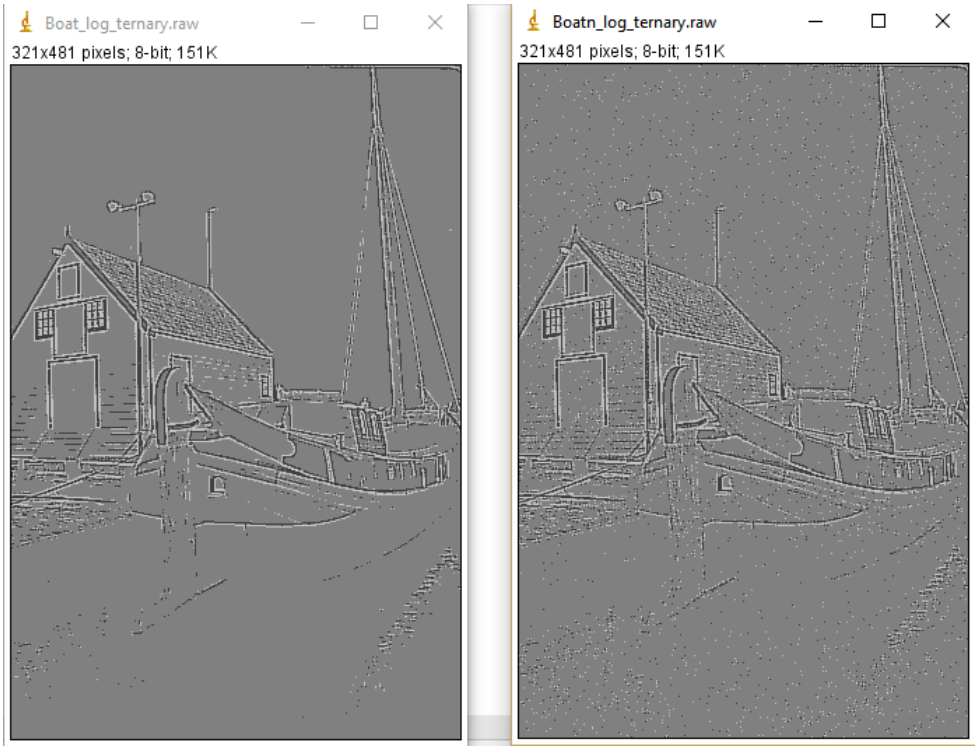


Figure 2.10(a) Ternary Map obtained for LOG filter for Boat.raw

Figure 2.10(b) Ternary Map obtained for LOG filter for Boat_noisy.raw

The edge map for the images obtained are shown in the Figure 2.11.

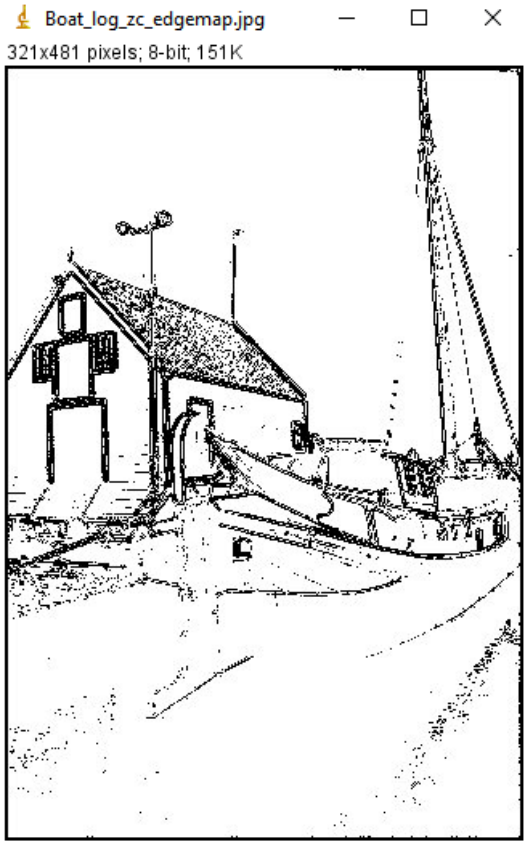


Figure 2.11(a) Edge Map obtained for LOG filter for Boat.raw

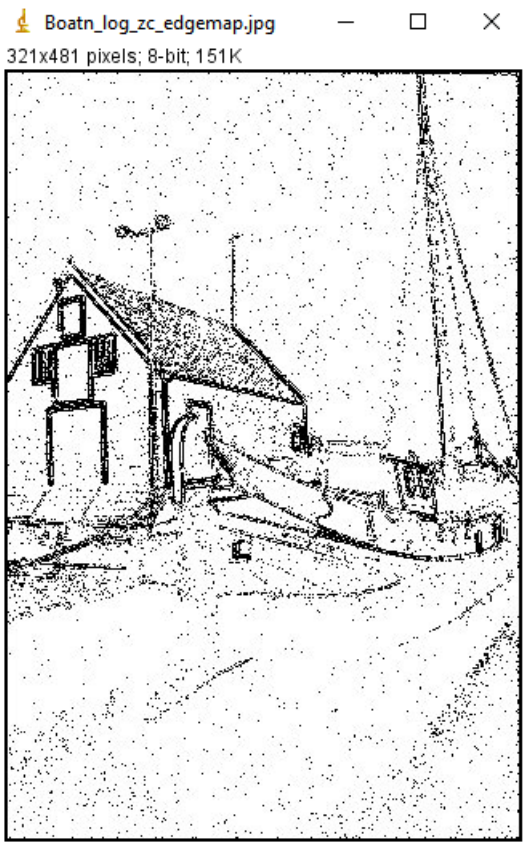


Figure 2.11(b) Edge Map obtained for LOG filter for Boat_noisy.raw

A.4 Discussion

Comparing the output for Sobel Edge detector, it is observed that the Sobel detector gives better edge detection results for higher inverse thresholds. Comparing the Sobel outputs for the original image and the noisy image, it is observed that the Sobel detector works better for the image without noise. There is always granular noise present in the operations on Boat_noisy.raw image. Even in the final gradient map, and in the edge map, there is always granular noise present, which is a disadvantage. Though Sobel filter is simple, it provides the advantage of identifying the edges with their orientations. Sobel filters are inaccurate in terms of edge detections, especially in the presence of noise. The presence of noise degrades the edges, as observed in the results above. This is an example of a classical edge detector, which is based on gradient values.

Comparing the LOG operator, it is observed that the sensitivity to noise is slightly improved. The algorithm for detecting the knee points is as follows:

- The histogram of the LOG – operated image is obtained.
- The mean of the resulting image is obtained.
- The standard deviation of the distribution is obtained.
- By varying the value of the standard deviation towards both the extreme values, the point where there is a maximum change of values between the current and previous values is taken as the knee points.
- If the knee points are obtained at a and b, assuming $a < b$, the entire histogram is divided into three regions – $[0, a), [a, b], and\ (b, 255]$.
- Each region is assigned a value: -1, 0 and 1 respectively. The image is viewed by mapping these three ternary values to 64, 128 and 192.
- After this edge detection is applied. The following conditions are considered for a zero crossing.

	1	
	0	
	-1	

	-1	
	0	
	1	

-1	0	1

1	0	-1

If there is a zero crossing, the edge is assigned a 1, else a zero.

The results of LOG operator are better compared to the results of Sobel operator, since LOG first smoothens out the noise, and then the edge is detected. The original Boat.raw image gives better edge detection results, compared to Boat_noisy.raw using the LOG operator. For noise smoothening, the results are better compared to the Sobel operator. The second derivative operator used here gives better results in terms of edge detection, since an edge is decided based on the second derivative – zero crossing and the peak of the first derivative. A disadvantage is that this operator doesn't detect edges where the gradient intensity function varies.

Comparing the Sobel and LOG operators,

- Sobel gives very poor results for noisy images, and the LOG gives better performance for noisy images.
- Sobel is simple to implement, and LOG requires more complex computations.
- Sobel easily detects the edges and their orientations, and LOG does easy zero – crossings.
- Sobel doesn't consider wider area around pixels, and LOG considers wider area around pixels.
- LOG doesn't detect edges where the gradient intensity function varies.
- The overall performance of LOG is better compared to Sobel.

Part B – Structured Edge Detection

B.1 Motivation

An edge corresponds to a local change in an image, particularly in terms of pixel intensity. There are different edge detection algorithms available. The traditional methods of edge detection include using Sobel operator, zero – crossing detector, LOG operator etc. These operators compute the color gradient magnitudes, and suppress the non – peak points in the image. This approach is fine, if there are no texture edges in the image. If there are visually salient edges (like texture – edges), these do not fit into the color – gradient magnitudes, and hence the traditional methods fail here. A better approach could be the use of a neighborhood (or a patch of pixels), and compute the chances that the patch contains an edge. This method could be feasible, since the patches are local, and contain inter – dependent pixels.

B.2 Approach

The conventional edge detectors like Sobel use the gradient based techniques for edge detection. Structured Edge detector makes use of the structure that is present in the images. Structure here refers to the edges present (with reference to edge detection).

A pixel is computed based on a local patch. The likelihood of an edge present in the local patch is computed. The structured forests used here operate on a standard input space. SE Detector uses a Random Forest(RF) Classifier. RF Classifier consists of multiple decision trees, and the results are integrated to one polynomial function.

Decision Trees:

A decision tree is used to classify a node by binary split function, until a leaf node is reached. Based on a decision function, the incoming sample is either sent to the left or right side of the tree. The output of a decision tree is the value on the leaf node that is reached by recursively branching through the tree. Consider an incoming sample $x \epsilon X$. The binary split function can be given as $h(x, \theta_j) \epsilon \{0,1\}$. The decision rule is that

$if\ h(x, \theta_j) = 0, node\ j\ sends\ x\ to\ the\ left, otherwise\ to\ the\ right.$ The split function can be arbitrarily chosen – it is usually based on a threshold like

$\theta = (k, \tau)\ and\ h(x, \theta) = [x(k) < \tau]$, where [.] is an indicator function.

Or $\theta = (k1, k2, \tau)\ and\ h(x, \theta) = [x(k1) - x(k2) < \tau]$.

A decision forest is an ensemble of T independent trees. The choice of ensemble model depends on the choice of the problem. Each decision tree is trained recursively. This training is dependent on the Information Gain Criterion and the choice of the split parameters. Training is done recursively until the maximum depth of the node is reached, or if training set is below the thresholds. From this it is observed that independent decision trees are highly variant, and it is overcome by combining multiple de – correlated trees. Higher diversity of the tress by subsampling and randomness.

These decision forests might run into a problem of overfitting, which can be overcome using Random Decision Forests. Some important parameters in the ransom decision trees include the information gain criterion and the ensemble model. The prediction of an input sample x is dependent on these parameters and as well as on the prediction of multiple uncorrelated decision trees.

Structured Edge Detection Algorithm:

Goal: Label each pixel of the input image denoting if it has an edge or not.

Given a set of training images, wherein boundaries between segments correspond to contours. An image patch of size N*N is chosen – and specified either as a segmentation mask y or binary edge map y'. The following steps are followed in SE detection.

i. Computation of input features:

The input image patch predicts a $16 * 16$ segmentation mask from a $32 * 32$ input patch. The input image patch is augmented with K channels, with resulting feature vector $x \epsilon \mathbb{R}^{32*32*K}$. The two feature types used here are pixel lookups $x(i, j, k)$ and pairwise differences $x(i1, j1, k) - x(i2, j2, k)$. Using the set of color channels for detection, there are a total of 13 channels (3 color, 2 magnitudes and 8 orientation channels). The channels are down sampled by a factor 2, resulting in $\frac{32*32*13}{4} = 3328$ candidate features $x(i, j, k)$. Then a large triangle blur is applied and down sampled to a resolution of 5*5. Hence, there will be 300 candidate features per channel, or 7228 features per patch.

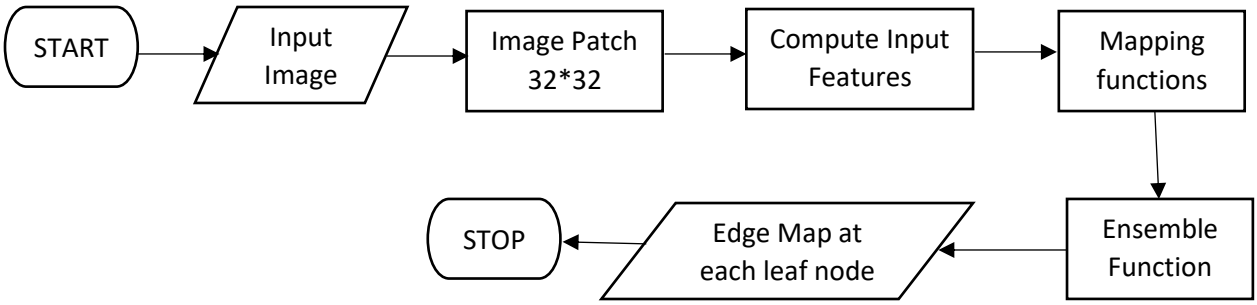
ii. Mapping Functions:

This is used for training the decision trees. The structured labels are 16*16 segmentation masks. If $\pi : Y \rightarrow Z$ is the mapping to be used, Euclidean distance gives a poor measure, hence a different mapping is used. Let y(j) denote j th pixel of mask y, for j from 1 to 256. Sample a pair of locations $j1 \neq j2$ and check if $y(j1) = y(j2)$. This defines the mapping as a binary vector. With reference to the paper, a subset of 256 dimensions gives better results.

iii. Ensemble model:

This is used to combine multiple input predictions. Multiple decorrelated decision trees are combined to achieve robust results. Predictions are combined through averaging. Thus, each leaf node stores an edge map.

The advantage of this method is that the use of structured labels reduces the number of decision trees to be evaluated for each pixel. The flow chart is below.



B.3 Results

The Structured Edge detector is applied to Animal and House Images. The output obtained for Animal and House Images are shown as follows.

The probability maps and the corresponding binary edge maps for the Animal.jpg image is shown in Figure 2.12.

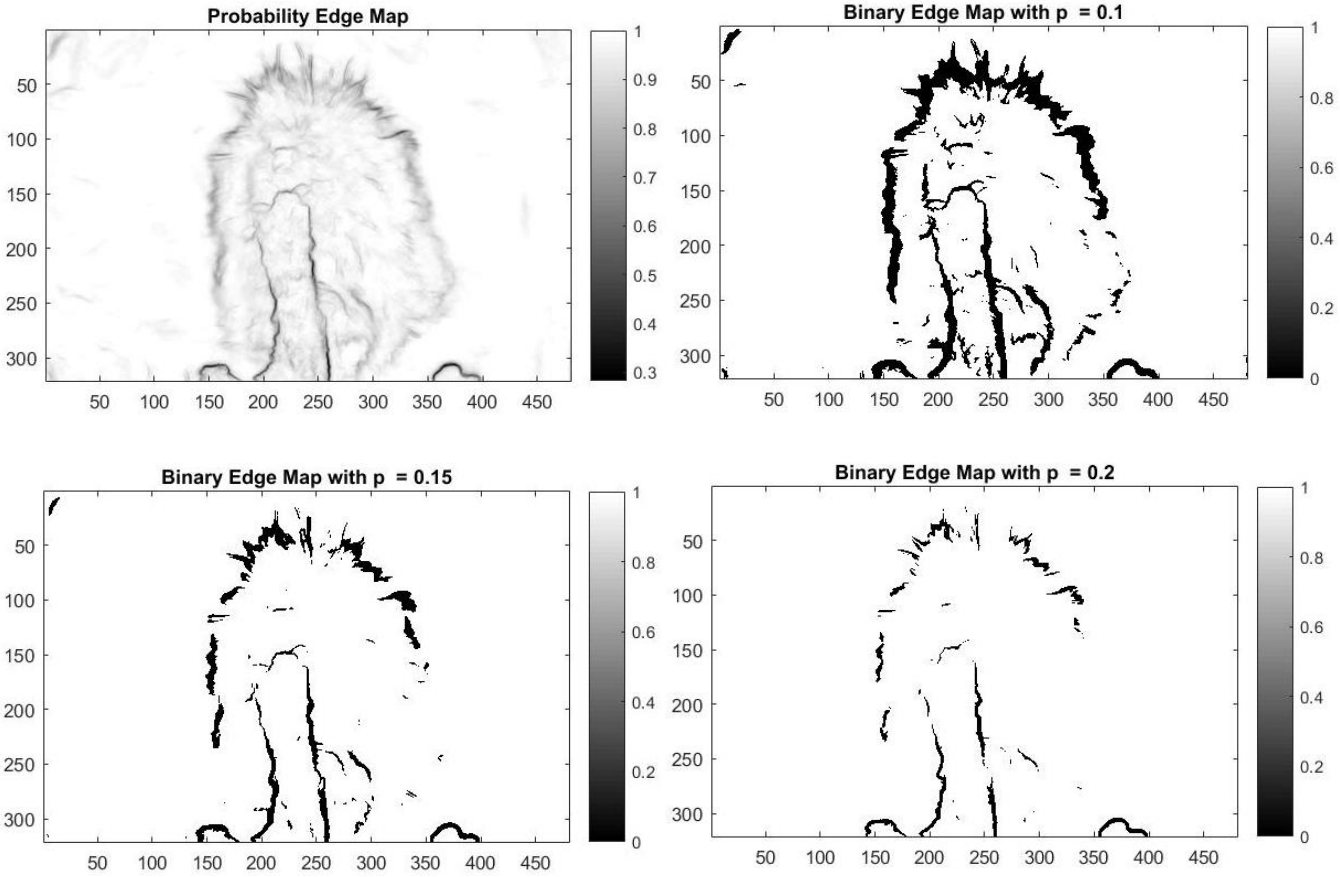


Figure 2.12 Outputs for Animal.jpg image without NMS

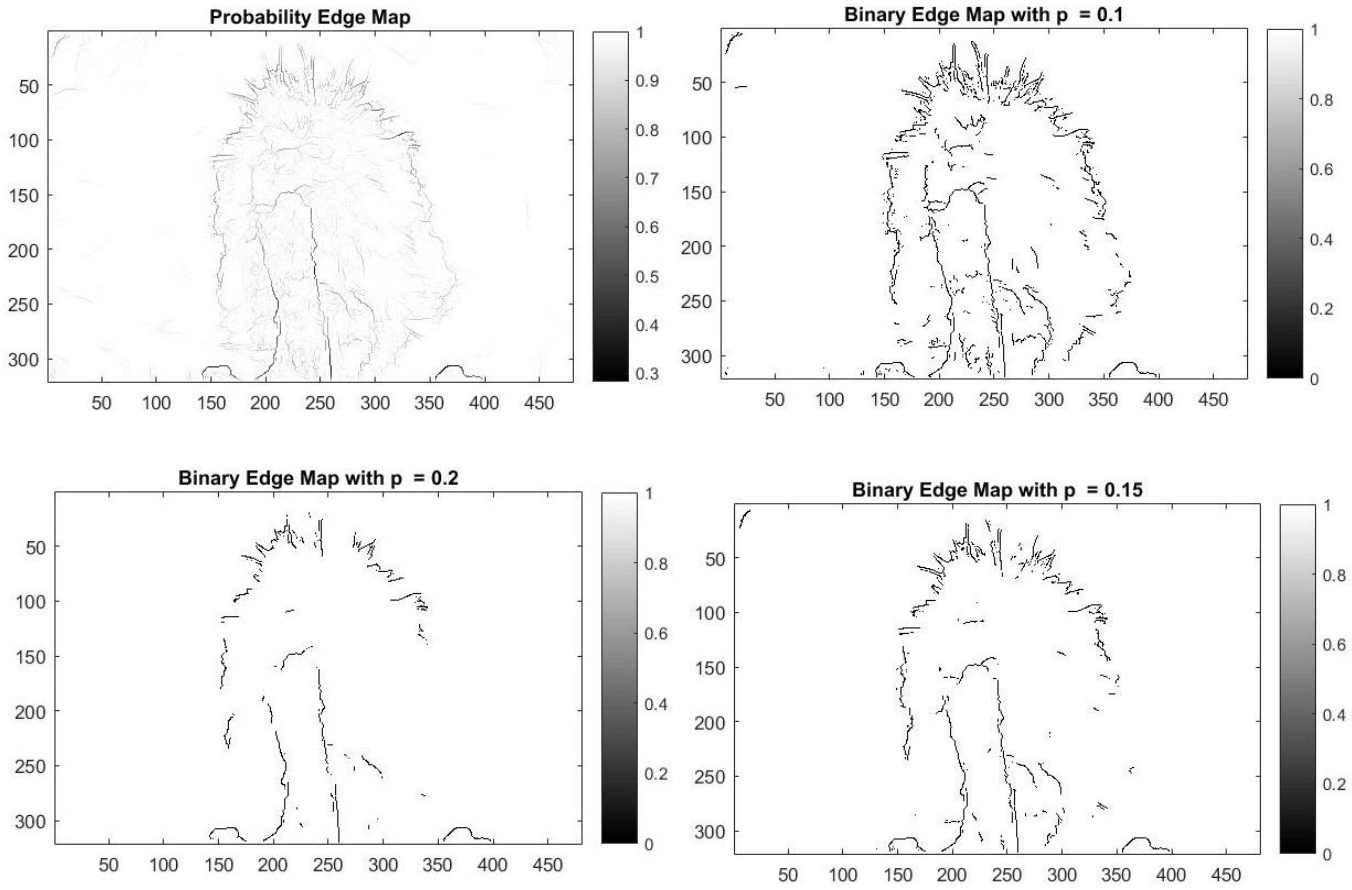


Figure 2.13 shows the results with NMS as the post – processing step.

The probability maps and the corresponding binary edge maps for House.jpg image is shown in Figure 2.14.

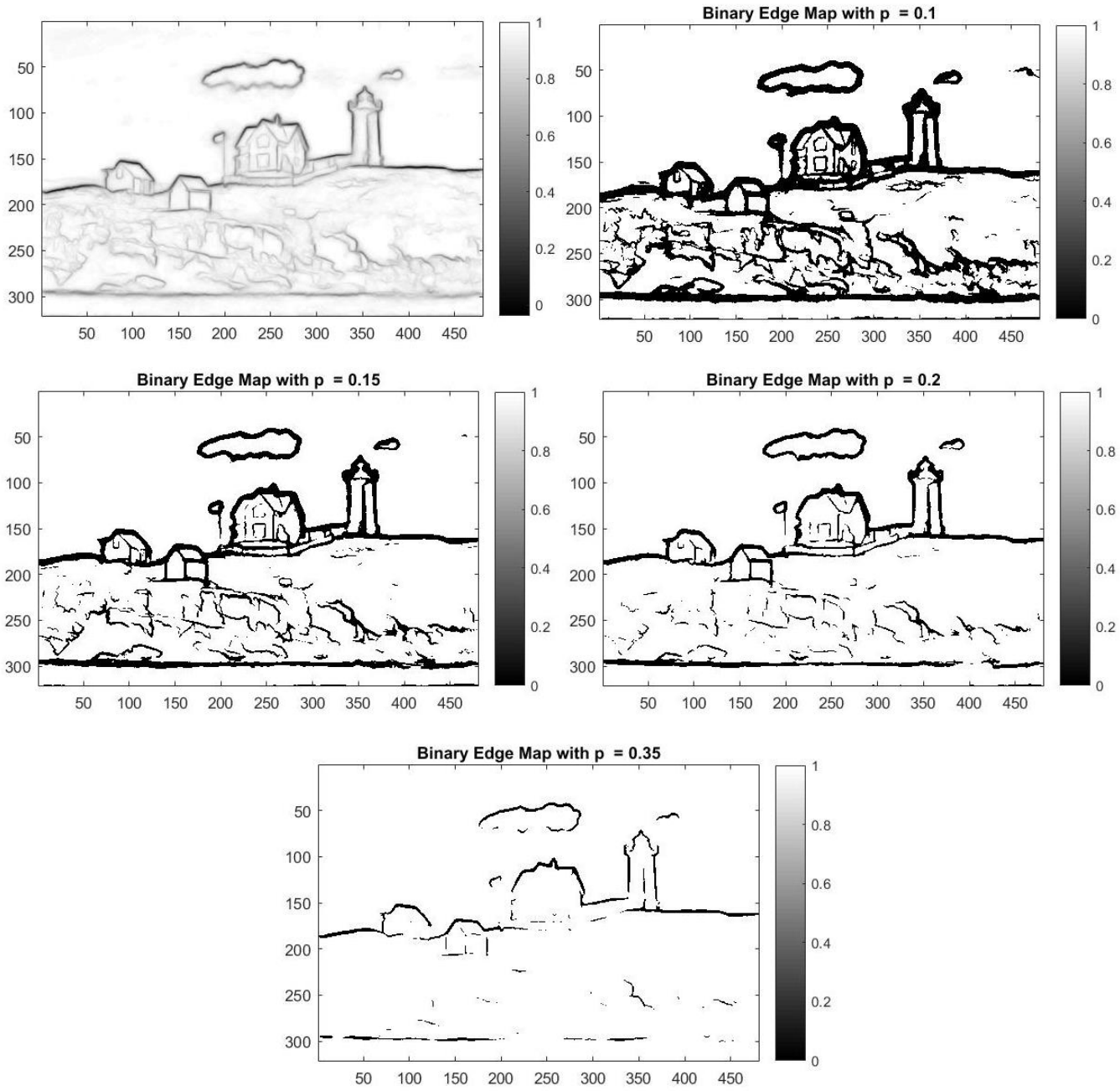


Figure 2.14 Outputs for House.jpg image without NMS

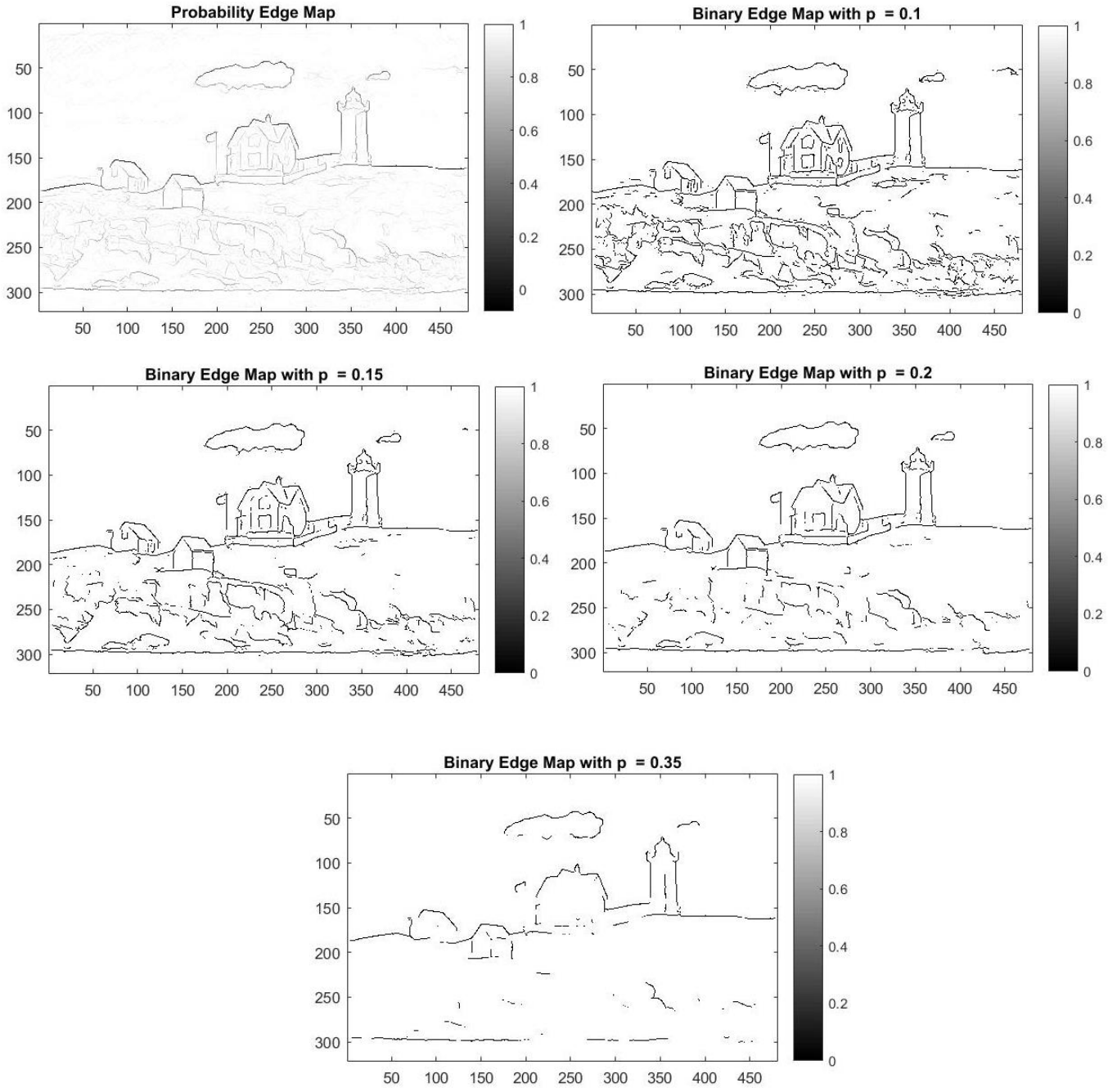


Figure 2.15 shows the results with NMS as the post – processing step

Comparing Sobel and SE detector, based on the visual results, Sobel detector gives more deformed output. There are many noisy features. Edges are better detected in SE because it used random decision trees, wherein all factors are considered, unlike Sobel, where only the neighborhood pixels are considered.

B.4 Discussion

The Structured Edge Detector is found to give better results compared to Sobel and LOG edge detection algorithms. The threshold for the Edge Detection algorithms are chosen in such a way that maximum features are retained in the image. Comparing the visual results of the Sobel and the Structured Edge detector, SE detector gives better results.

Part C – Performance Evaluation

C.1 Motivation

In computer vision, detection of edges is a primary operation. There are different methods of edge detection employed. Some examples include the use of conventional edge detectors like Sobel, Prewitt, Roberts operators, or the use of zero – crossing edge detectors, or Laplacian of Gaussian edge detectors (that employ second derivative features). There are also learning – based methods like Structured Edge Detection methods (as in the previous part). To compare and analyze the various edge detectors, a performance measure is needed. This performance evaluation is done in this problem using a measure called F – measure.

C.2 Approach

The objective is to compare the performance of different edge detectors. The algorithm should be able to produce edges better than humans do. So, the performance measure involves edges drawn by humans too – these are called ground truth images. The Berkeley Segmentation Data Set 500 (BSDS 500) is used here, that provides some ground truth images for the data. The performance measure used here is the F – measure, that is calculated by the harmonic mean of Precision and Recall.

F – measure (F) is given in terms of Precision (P) and Recall (R) as

$$F = 2 * \frac{P * R}{P + R}$$

P and R are calculated as

$$Precision\ P = \frac{\#True\ Positives}{\#\ True\ Positives + \#\ False\ Positives}$$

$$Recall\ R = \frac{\#True\ Positives}{\#\ True\ Positives + \#\ False\ Negatives}$$

True Positive – edge pixels in the edge map coincide with edge pixels in ground truth images

True Negative – Non – edge pixels in edge map coincide with non – edge pixels in the ground truth

False Positive – Edge pixels in edge map correspond to non – edge pixels in ground truth

False Negative – Non – edge pixels in edge map correspond to edge pixels in ground truth.

Hence True Positive and False Positive measures are correct detection of edges.

For every ground truth image, the F – measure is computed. The mean of all F – measures gives the final F – measure for the image.

The precision parameter could be made higher by decreasing the threshold used for obtaining the binary edge map. But this results in a lower recall. Hence using the combined parameter F – measure gives a better result. A higher F – measure implies a better edge.

C.3 Results

The probability thresholds could be chosen based on the F – measure, for better results. To calculate F – measure, the following test parameters are changed, the training parameters are left unchanged.

```
%% set detection parameters (can set after training)
model.opts.multiscale=0;          % for top accuracy set multiscale=1
model.opts.sharpen=2;             % for top speed set sharpen=0
model.opts.nTreesEval=4;          % for top speed set nTreesEval=1
model.opts.nThreads=4;            % max number threads for evaluation
model.opts.nms=0;                 % set to true to enable nms
```

These are the default parameters. These are changed as follows.

```
model.opts.multiscale=1;          % for top accuracy set multiscale=1
model.opts.sharpen=2;             % for top speed set sharpen=0
model.opts.nTreesEval=4;          % for top speed set nTreesEval=1
model.opts.nThreads=4;            % max number threads for evaluation
model.opts.nms=0 or 1 ;                % set to true to enable nms
```

NMS is set to both TRUE and FALSE and then the code is evaluated.

The Precision, Recall and F – measure parameters that are calculated for both the images are obtained as follows.

Input Image: Animal. jpg

Parameter	GT1	GT2	GT3	GT4	GT5	Mean
P	0.6954	0.6324	0.4947	0.6786	0.5457	0.7379
R	0.6151	0.6560	0.4036	0.7357	0.6886	0.6917
F	0.6528	0.6440	0.4445	0.7060	0.6089	0.7141
Average F	0.70124					

The overall Precision, Recall and F – measure on running all the five GT images together is 0.7379, 0.6917 and 0.7141 respectively.

Input Image: House. jpg

Parameter	GT1	GT2	GT3	GT4	GT5	Mean
P	0.6477	0.7704	0.7123	0.6075	0.5413	0.8223
R	0.8576	0.6594	0.6975	0.7149	0.7499	0.7784
F	0.7380	0.7106	0.7048	0.6568	0.6287	0.7995
Average F	0.74158					

The overall Precision, Recall and F – measure on running all the five GT images together is 0.8223, 0.7784 and 0.7995 respectively.

The Sobel and LOG detectors are applied on the House.jpg and Animal.jpg and the results re tabulated as follows.

		Animal.jpg	House.jpg
Sobel	**Precision**	0.1855	0.236
	Recall	0.8590	0.8861
	F - Measure	0.308	0.3658
LOG	**Precision**	0.1770	0.1875
	Recall	0.9702	0.9512
	F – Measure	0.3210	0.313
SE Detection	**Precision**	0.737	0.8223
	Recall	0.6917	0.7784
	F - Measure	0.7141	0.7995

C.4 Discussion

The F – measures for different edge detectors – Sobel, LOG and Structured Edge Detectors are calculated using the given Matlab codes. A better F – measure indicates a better edge detection. It is observed that the Structured Edge Detector gives better F – measure compared to Sobel and LOG edge detectors. Comparing the performance of the different edge detectors,

- F – measure for SE detector is higher, irrespective of the images
- Sobel has a poor F – measure because it involves a simple gradient operation that considers only the changes in intensity values.
- LOG gives better values compared to Sobel, because it uses the second derivative for edge detection, and that the orientation matters, But LOG fails at the boundaries of variant intensity changes. Hence F – measure is lesser than that of SE detector.
- There are some discrepancies observed in calculating the F – measure. There are two methods of calculating F – measure. One: F – measure is done separately for each image, and then the mean is calculated. The second: All images are taken

as input at once, and then the mean is calculated. The F – measure values are found to be different for both. F – measure is better when the second method is used.

F – measure is image dependent. This is observed from the F – measure values obtained for Animal.jpg and House.jpg images for different Edge detectors. F – measure majorly depends on the Ground Truth images, which are human drawn. The images that have more objects in them, are better drawn for edges, with the motive of positive edge detection. Also, there is a proper intensity change observed in the House. jpg compared to Animal.jpg image. The features in the House image are clearly defined, and hence edges can be easily drawn. In Animal image, the features are more difficult to recognize and draw the edges properly.

F – measure is usually considered as the harmonic mean of Precision and Recall. In simple words, for any classification problem, Precision is the number of elements correctly labeled belonging to the positive class (called true positives) divided by the total number of elements that are labelled as belonging to the positive class. Recall is defined as the number of true positives divided by the total number of elements that belong to the same class. If Precision = 1.0 for any class C denotes every item labeled as class C belongs to class C and Recall = 1.0 denotes every item from class C was labeled as belonging to class C. Solving for F – Measure mathematically,

$$F = \frac{2 * PR}{P+R}$$

Solving,

$$F = 2 * \frac{\frac{TP}{TP+FP} * \frac{TP}{TP+FN}}{\frac{TP}{TP+FP} + \frac{TP}{TP+FN}}$$

$$F = 2 * \frac{TP}{(2TP) + FN + FP}$$

From the above equation, it is not possible to obtain a higher F measure with higher precision or higher recall. When P+R is constant, either P increases, this means FP increases, and since P + R is a constant, R decreases. When P = R,

$$F = 2 * \frac{R}{R + R} = 1$$

Hence a maximum value of F – measure (= 1) can be obtained when P = R.

This calculation of F – measure hence serves as a descriptor to evaluate the performance of different edge detection algorithms.

References:

1. William Pratt, “Digital Image Processing”
2. Piotr Doll´ar, C. Lawrence Zitnick, “Structured Forests for Fast Edge Detection”
3. https://github.com/pdollar/edges
4. BSDS 500 Dataset

Problem 3 – Salient Point Descriptors and Image Matching

Part A – Extraction and Description of Salient Points

A.1 Motivation

In Computer Vision, Image Matching problems are important in finding the structure and orientation of any object. Image Matching is widely used in many applications like Object Retrieval, Motion Tracking etc. The major steps in Image Matching include Feature Extraction, Mapping of Features and Application.

In Feature Detection and Extraction, algorithms are used to detect the point of interest in the images (or simply the keypoints) and the descriptors. Keypoints and descriptors are vital in identifying the similarity between objects in the image. Some features that integrate

both keypoints and descriptors like SIFT, SURF, ORB etc. are widely used. In matching, the similarity of each point with the most similar one in other image is computed. For example, some techniques like FLANN used Euclidean distance to find the similar features. Hence it is used for matching non – binary features like SIFT and SURF.

A.2 Approach

In this problem, the salient points in the image must be identified, and the features must be extracted. Here images that have similar properties are taken into consideration. These features are invariant to rotation and scaling.

SIFT Algorithm:

Scale Invariant Feature Transform(SIFT) is an algorithm used to identify the keypoints and descriptors in the image. The main steps in the algorithm – Scale space Extrema Detection, Keypoints Localization, Orientation Assignment, Keypoint Descriptor and Keypoint Matching.

1. Scale – space Extrema Detection:

Since SIFT should detect larger corners, there is a need for larger windows. For this, scale – space filtering is used. Hence, a scale – space is constructed, which is invariant to scaling. Using this scale – space, potential keypoints are identified.

SIFT algorithm uses Difference of Gaussians (DOG) which approximates the Laplacian of Gaussian(LOG). DOG is obtained as the difference of Gaussian with two different values of the standard deviation, which is done for different octaves of the image. Once DOG is done, local extrema is found across the scale – space i.e. keypoints are represented the best in that scale. The keypoints are usually considered the maxima and minima (i.e. local extrema).

2. Keypoint Localization:

The keypoints obtained from Step (1) are localized to get more accurate keypoints. In this step, keypoints of low – contrast and those at the edges are eliminated. Hence, the resultant will be the desired keypoints of interest.

3. Orientation Assignment:

Each Keypoint is now assigned an orientation to make it invariant to rotation operations. A histogram of the orientation of pixels about a Keypoint is calculated, and using threshold of 80%, orientation of a Keypoint is calculated. This makes the algorithm invariant to rotation.

4. Keypoint Descriptor:

Knowing the orientation assignment, a Keypoint descriptor is constructed. A 16*16 neighborhood is taken, and divided into 16 blocks of 4*4. For each block, 8 bin histograms of orientation are created. Hence there are a total of 128 bins for a keypoint,

5. Keypoint Matching:

The nearest neighbors of a keypoint in two images are identified, and compared.

After Step 4, the SIFT features are created. Here the keypoints are extracted, and descriptors are found for each keypoint. Hence if there are K – Keypoints, there will be a total of $K * 128$ descriptors.

SURF Algorithm:

Speeded – Up Robust Features(SURF) algorithm is an improved version of the SIFT algorithm. Basically, it is a fast version of SIFT. In SIFT, LOG was approximated by a DOG filter. In SURF, LoG is approximated by a Box Filter. This provides the advantage of easy computation of convolution (using integral images) and parallel computations of different scale – spaces.

SURF uses wavelet responses for orientation assignment. These responses can be found from the integral images; hence this computation is faster. This method also uses signed Laplacian operator, which is helpful in stages of Keypoint Matching wherein the contrast is just compared for same type. This minimal description allows faster computation. The number of features are limited by the Hessian Threshold.

Extraction of features is done using OpenCV and Python. The input images that are to be used are Optimus_prime.jpg and Bumblebee.jpg.

A.3 Results

The actual input images are Optimus_Prime.jpg and Bumblebee.jpg, which is shown in Figure 3.1.

Figure 3.1(a) Optimus_prime.jpg

Figure 3.1(b) Bumblebee.jpg

SIFT and SURF algorithm is applied to get the desired keypoints.

Figure 3.2 shows the output of the above images with the default parameters (128 descriptors, and the number of keypoints determined by the code).

Figure 3.2(a) SIFT_Optimus_prime.jpg – Keypoints: 1765

Figure 3.2(b) SIFT_Bumblebee, jpg – Keypoints: 605

The number of keypoints for SIFT are varied and the results are obtained as in Figure 3.3.

Bumblebee_ Keypoints :200

Bumblebee_ Keypoints :500

Figure 3.3(a) SIFT_ Bumblebee.jpg – varying the number of keypoints

Keypoints : 5000

Keypoints : 500

Figure 3.3(b) SIFT_Optimus_prime.jpg – varying the number of Keypoints

The SURF outputs using the default parameters are shown in Figure 3.4.

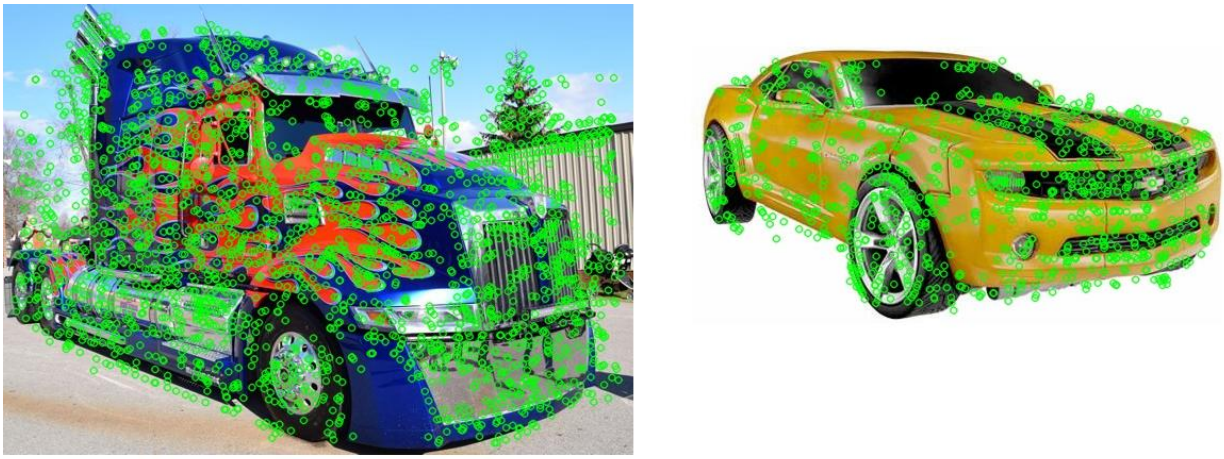


Figure 3.4(a) SURF_Optimus_prime - Keypoints : 2564

Figure 3.4(b) SURF_Bumblebee – Keypoints : 1053

The Hessian parameter threshold can be used in SURF to limit the number of keypoints. The result with a different Hessian parameter is shown in Figure 3.5.

Figure 3.5 SURF _Outputs with different Hessian Thresholds

To know the orientation of the keypoint, only one parameter in the function has to be changed. For example, consider the Bumblebee image, wherein the different orientations are shown as in Figure 3.6.

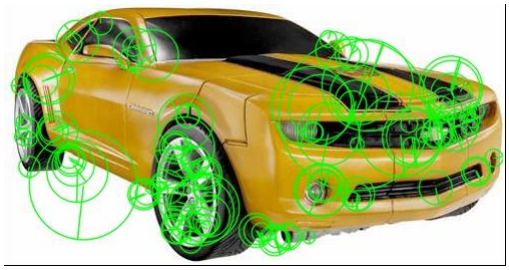


Figure 3.6 SURF_Matcher with orientation parameter = 4

A.4 Discussion

Both SIFT and SURF are used in this problem to extract the keypoints (or the salient features) in an image. Comparing SIFT and SURF

- SIFT assigns the keypoints based on the local image gradient variations. SURF uses wavelet orientations for keypoint assignments.
- SIFT approximates the scale space using Difference of Gaussian Filters, while SURF used Box filters.
- For finding keypoints, SURF uses a Blob detector that uses Hessian matrix. Hence Hessian threshold determines the number of keypoints.
- Changes to illumination, noise and blur are significant when SIFT is used. SURF is better in these cases. Scaling and viewpoint invariance are better in SIFT.
- SURF algorithm is found to be faster, compared to SIFT because of the above features.
- The number of keypoints can be determined by the Hessian Threshold in SURF and a parameter change in SIFT.
- The different orientations can be also obtained, though this is not particularly useful in some applications.

As a concluding note, SIFT gives slightly better features for matching, though SURF is much faster.

Part B – Image Matching

B.1 Motivation

In Computer Vision, Image Matching problems are important in finding the structure and orientation of any object. Image Matching is widely used in many applications like Object Retrieval, Motion Tracking etc. The major steps in Image Matching include Feature Extraction, Mapping of Features and Application. In Image matching algorithms, keypoints and descriptors are vital in identifying the similarity between objects in the image. Some features that integrate both keypoints and descriptors like SIFT, SURF,

ORB etc. are widely used. In matching, the similarity of each point with the most similar one in other image is computed. This is done with the help of descriptors.

B.2 Approach

The two input images are matched for features using SIFT and SURF algorithms. The steps in image matching using SIFT algorithm are as follows.

1. The two input images needed for matching are obtained.
2. The input RGB images are converted into Grayscale images.
3. For each input image, the keypoints and descriptors of the image using SIFT algorithm is obtained. The keypoints are the actual pixel values that describe the salient features of the image. Descriptors contain the actual features. The descriptor matrix for SIFT is of size $N * 128$.
4. Image matching is done using comparison of descriptors. Algorithm like Brute – Force Matcher, FLANN (Fast Library for Approximate Nearest Neighbors) Matcher etc. can be used for feature – matching.
5. The matched features are displayed on the output images using lines.
6. For matching using SURF Algorithm, the steps 1 – 5 are followed, except that SURF is used instead of SIFT algorithm.

Here, Image matching is done using OpenCV and Python. The approach in OpenCV – Python is as follows.

i. The two input images are read using appropriate IMREAD commands.
ii. The input RGB images are converted into gray images using cvtColor function.
iii. SIFT or SURF algorithm is applied using the appropriate function call on both the images. For each image, the result will be an array consisting of keypoints and descriptors.
iv. Image matching is done using the in – built algorithms. Here, Brute Force Matching Algorithm is used because it gives better results.
v. Then the obtained matched keypoints are sorted, and the top K matches are used for display.
vi. Lines are drawn between similar features to know the matched components.

B.3 Results

First SIFT and SURF features are extracted from both the images, as in Figure 3.7 and Figure 3.8 respectively.

Figure 3.7(a) SIFT_Ferrari1

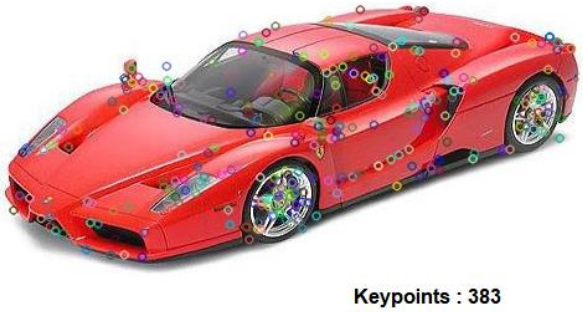


Figure 3.7(b) SIFT_Ferrari2

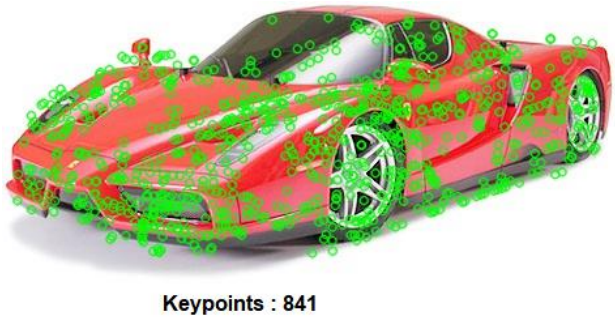


Figure 3.8(a) SURF_Ferrari1

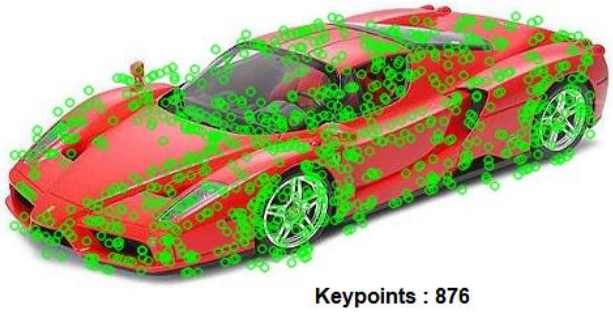


Figure 3.8(b) SURF_Ferrari2

The matcher used here are Brute Force and FLANN Based Matcher.

Figure 3.9 shows the output for Brute Force and FLANN Matchers for SIFT.

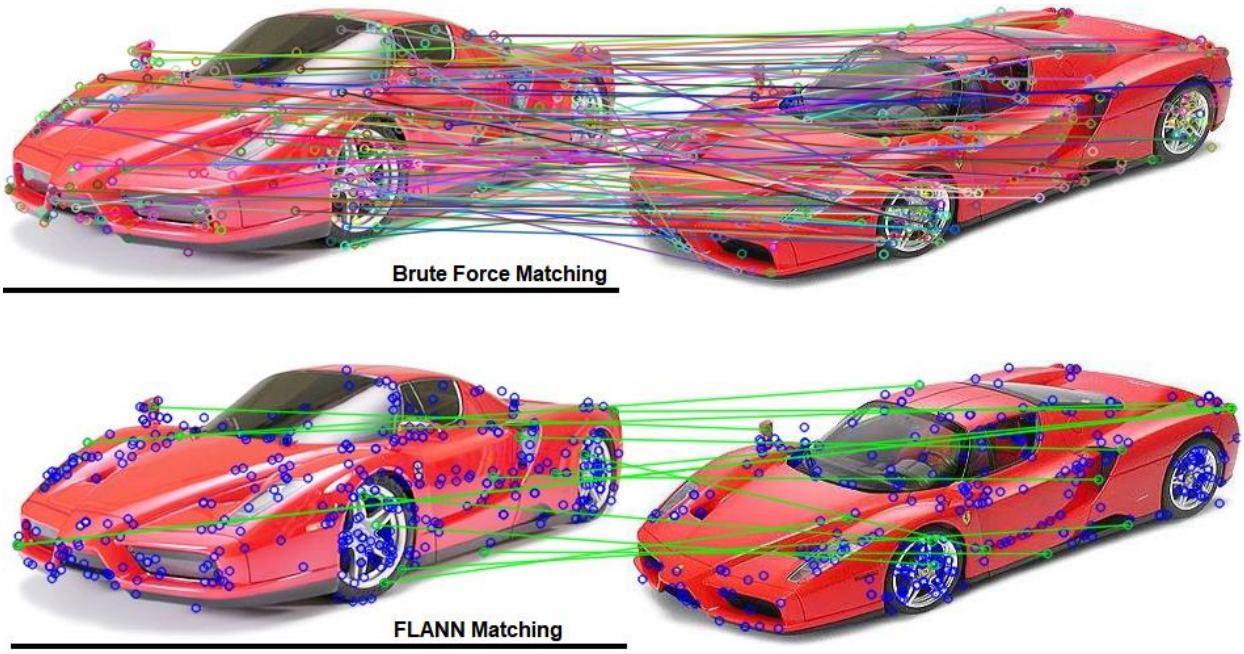


Figure 3.9 Matchers for SIFT

Figure 3.10 shows the output for Brute Force and FLANN Matchers for SURF.

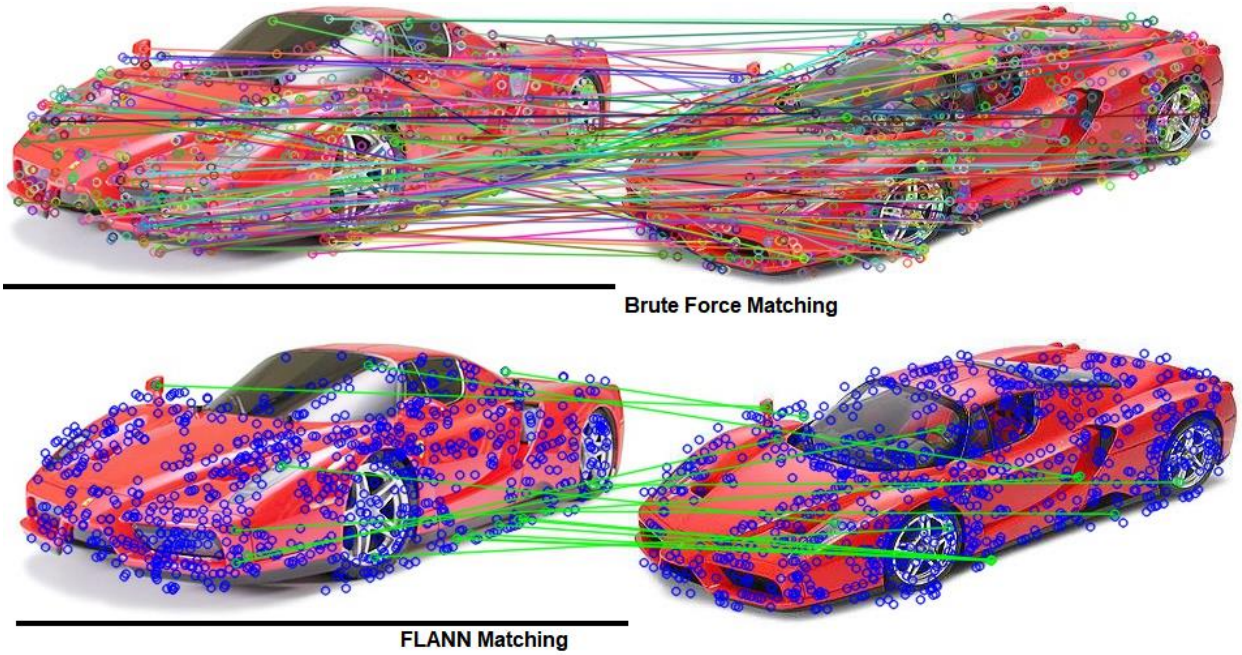


Figure 3.10 Matchers for SURF

This matching is done for different pairs of objects as well.

Figure 3.11 shows the matching between Ferrari1 and Bumblebee for SIFT and SURF

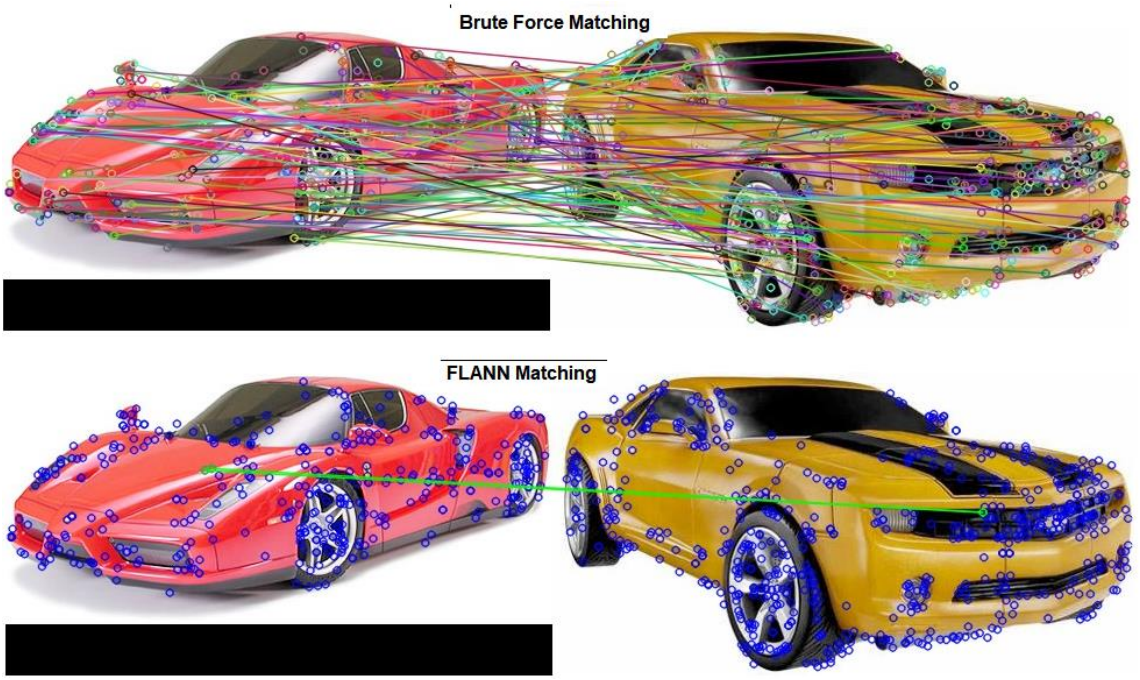


Figure 3.11(a) SIFT Matching for Ferrari1 and Bumblebee

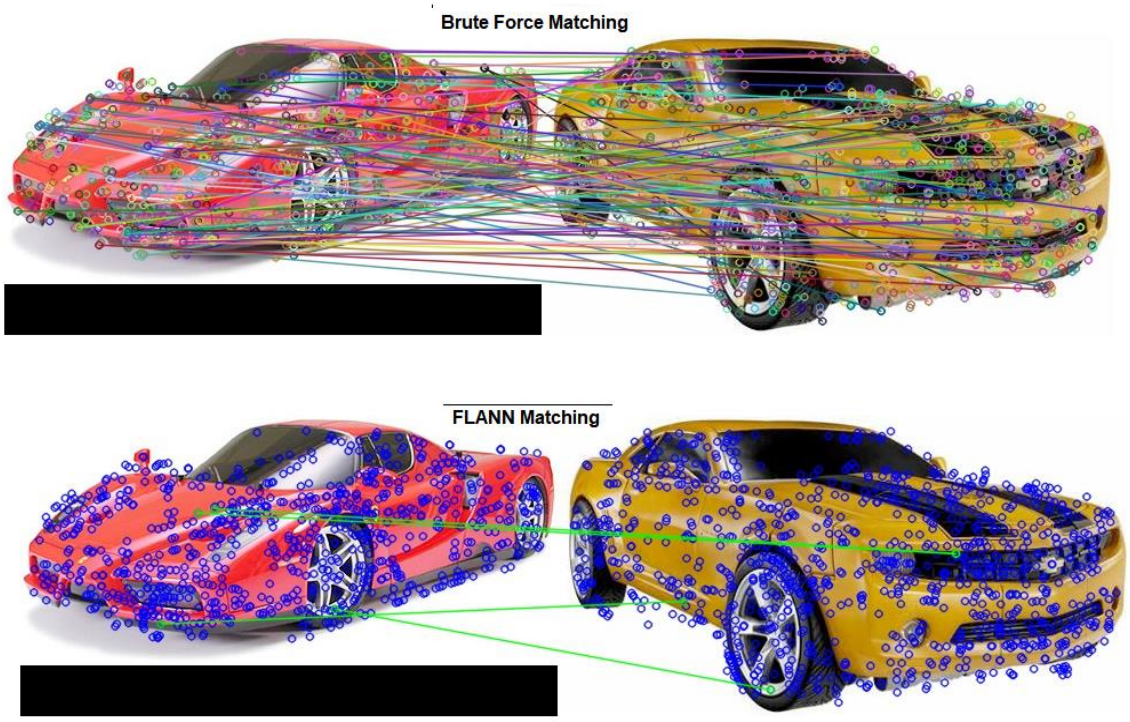


Figure 3.11(b) SURF Matching for Ferrari1 and Bumblebee

Figure 3.12 shows the matching between Ferrari1 and Optimus_Prime for SIFT and SURF

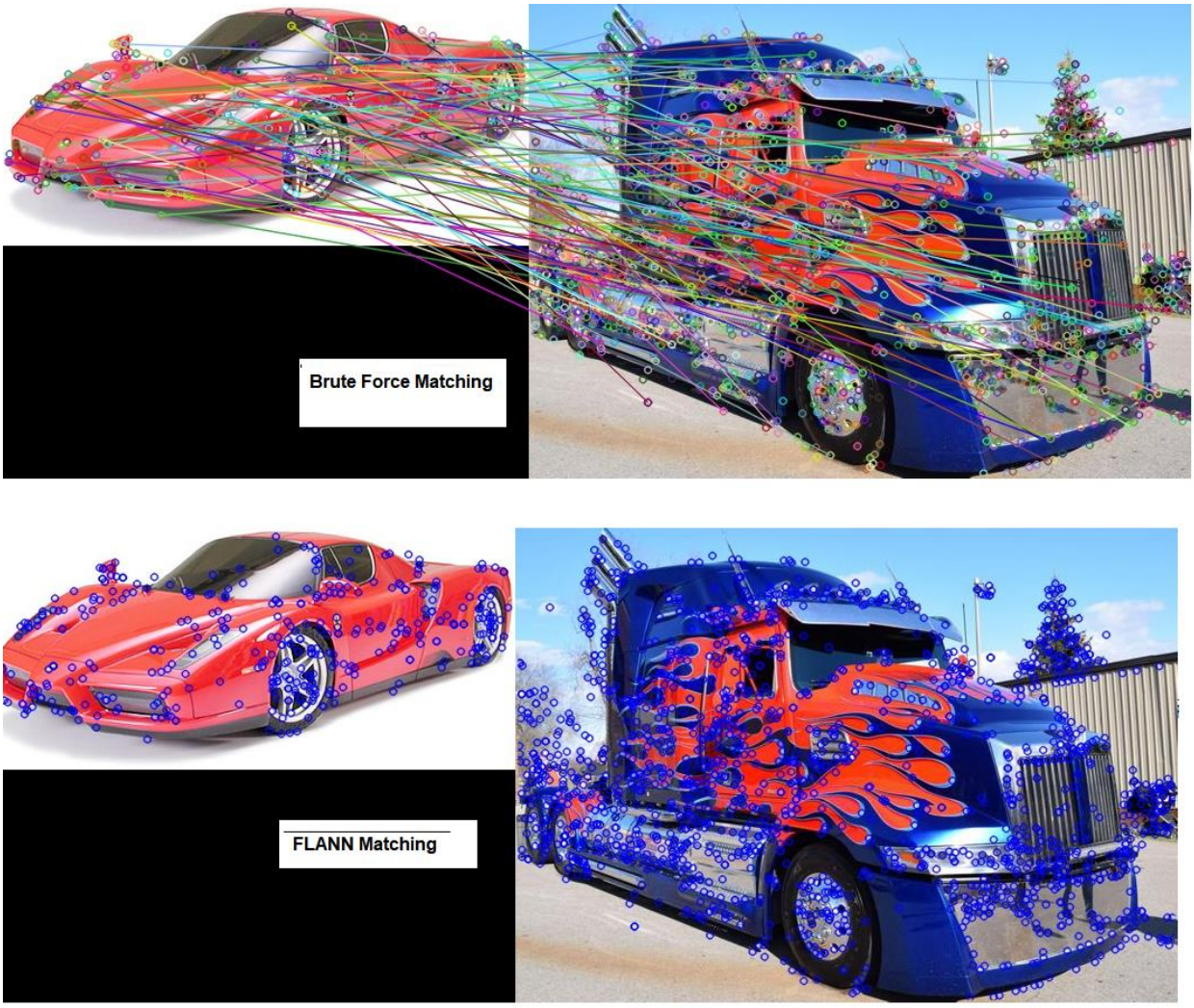


Figure 3.12(a) SIFT Matching for Ferrari1 and Optimus_Prime

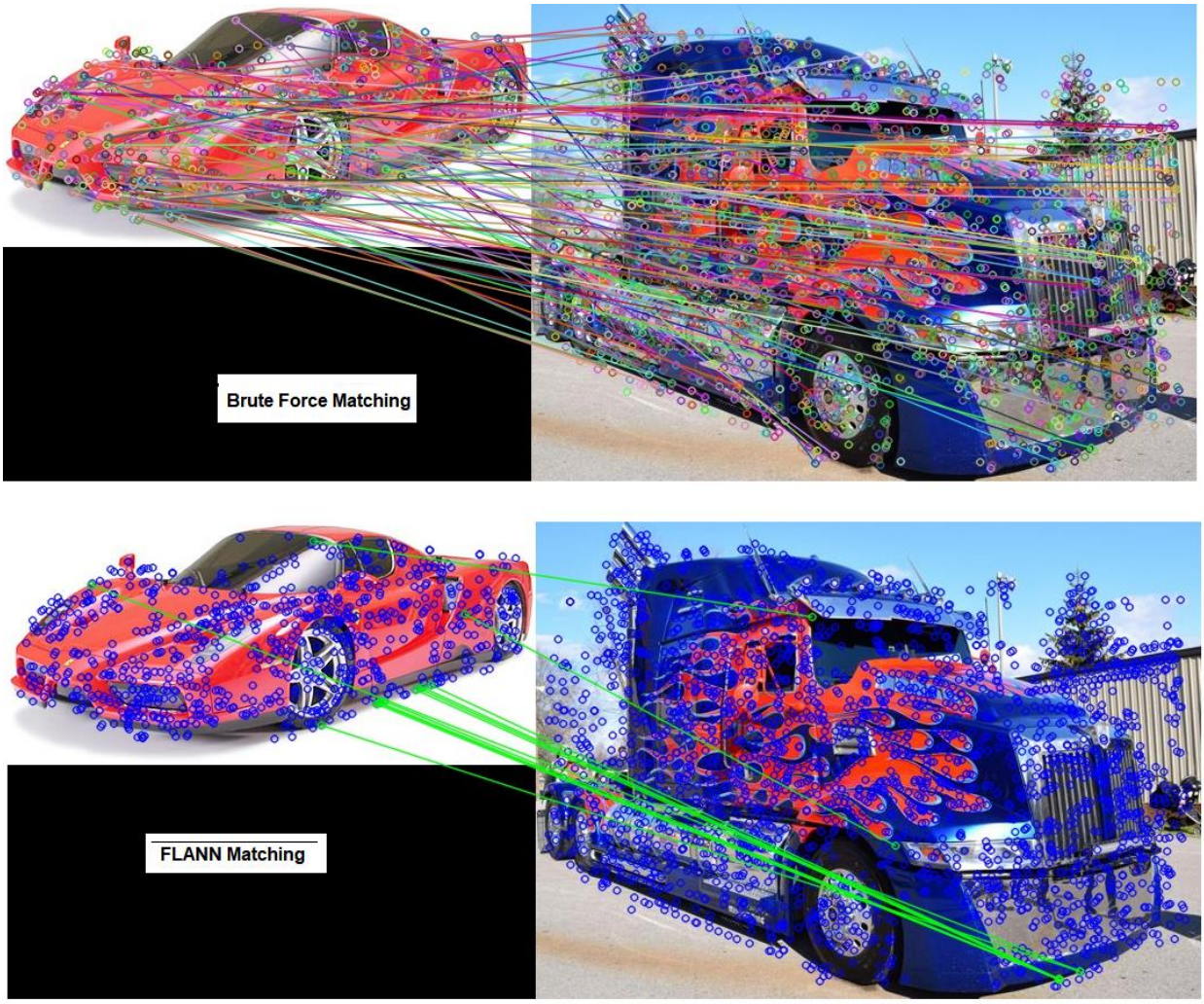


Figure 3.12(b) SURF Matching for Ferrari1 and Optimus_Prime

B.4 Discussion

In this problem, keypoints are extracted using SIFT and SURF methods, as in Figure 3.7 and Figure 3.8 respectively. Image Matching could be done with all the features or the top N features. For the conventional Image Matching Algorithms, there is a threshold on the number of matching features. In this problem, two matching methods are used – Brute Force (or BF Method) and the FLANN Based Method. BF Matcher takes the descriptor of a feature and compares it with some distance (either L2 Norm or Hamming Distance), and the closest match is returned. FLANN denotes Fast Library for Approximate Nearest

Neighbors. It is suitable for features of large dimensions and contains the set of algorithms for finding the nearest neighbors in an efficient and fast way. This is faster than BF Matcher.

FLANN is widely used because it gives better results. Comparing the outputs in Figure 3.9 and 3.10, BF Matcher connects almost all features, which is not desirable. Only the vital features need to be connected, as done by the FLANN matcher. In these matchings, it is observed that the points are matched based on the keypoints only – irrespective of location. SIFT Matching (Figure 3.9) seems to contain less matches compared to the SURF matching (Figure 3.10). SURF is preferred when the images are almost similar. BF matching is messy, and hence FLANN is a better option for Image Matching.

Figure 3.11 and Figure 3.12 give the matching results for different objects. In Figure 3.11, it can be observed that BF matching just matches the closest matching points, irrespective of feature checks. FLANN gives only very few lines of matching between the two different objects, for both SURF and SIFT, indicating that the objects are different. Hence FLANN outperforms BF Matcher. Similar is the case for Figure 3.12, where in there is a zero match.

For similar objects, the extracted keypoints are similar in terms of luminance, intensity and color. The keypoints also depend on the neighbors. Hence more matching is obtained for similar images. For dissimilar images, the keypoints are extracted, but they do not match in terms of the basic decision functions (or algorithms – as in FLANN).

Hence FLANN matcher is found to be better for image matching, as it rightly does image matching for similar and dissimilar images. This could be improved by using FLANN with Knn matcher.

Part C – Bag of Words

C.1 Motivation

In Computer Vision, Bag of Words(BoW) is a classification model that uses features as words, and is widely used for classifying images. The primary idea of this classification model is to group each keypoint into one of the visual words, and then use a histogram to represent them. The keypoints are usually generated by the SIFT algorithm, and the classification is usually done by the K – means algorithm. Bag of Words serves an important object classification approach.

C.2 Approach

Bag of Words model, used for image classification has three stages – Feature detection, feature representation and codebook generation.

Stage 1 – Feature Detection

The salient features in an image are detected by means of suitable algorithms.

Stage 2 – Feature Representation

After feature detection, each image can be represented using the smaller features. These smaller feature vectors are called descriptors, that contain the salient features of the image. These are called descriptors. Algorithms like SIFT give the keypoints and descriptors in an image. The descriptor using SIFT is of 128 dimensions.

Stage 3 – Codebook generation

After feature Representation, the feature descriptors are converted into vectors (or words). From this a codebook (or a dictionary) is also produced. After feature extraction using SIFT, codewords are generated using K – means clustering algorithm. These codewords are defined as the center of the learned clusters. The total number of clusters is the size of the code book. Hence each part of the image is mapped to some codeword.

The image can be represented by a histogram of codewords.

This Bag of Words is coded in C++ using the available source code.

The approach to Bag of Words can be described as follows.

i. The input training images are obtained.
ii. Features (Keypoints and Descriptors) are extracted from each image using the SIFT algorithm.
iii. Using the K – means clustering, the codebook or the dictionary is created.
iv. Now, the test images are obtained.
v. Features are extracted from the test images as well using SIFT.
vi. Then the nearest neighbor is obtained using suitable matching algorithm, and is classified to belong to that codeword.

C.3 Results

The output is a codebook, which is a .yml file, that contains the data for the training images. There is also a descriptor.yml file generated, that contains the code word for the test image – Ferrari_2.jpg.

The codewords of the four images are obtained, and they are plotted as below.

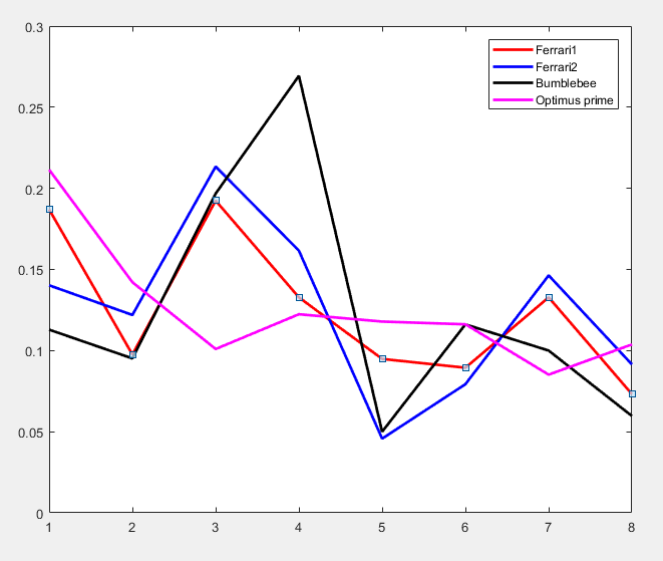


Figure 3.13 Plot of Code Words

They could also be classified using K – Means clustering into 8 Groups, where in the histogram plots are shown below.

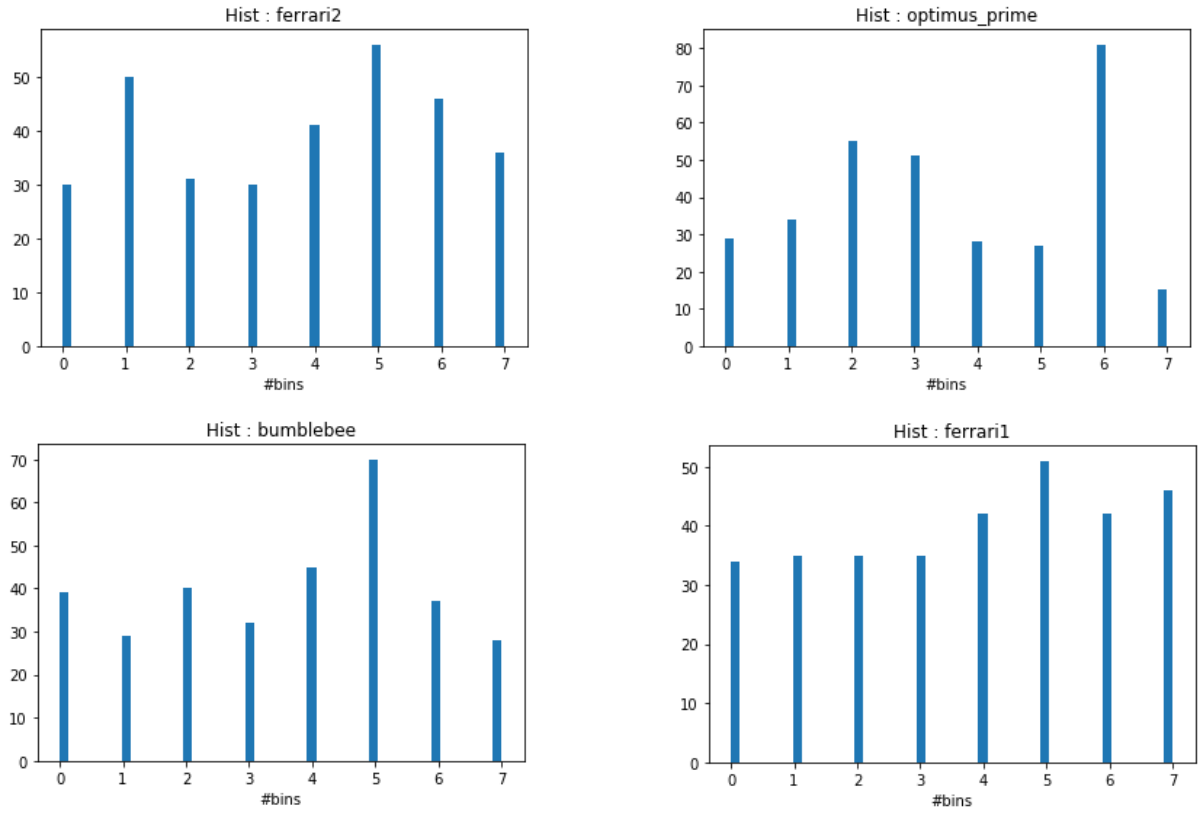


Figure 3.14 Histogram of Code words

The histogram of all the four images – three training and one test image is found.

C.4 Discussion

The training set consists of three images – Optimus_prime, Bumblebee and Ferrari_1. SIFT algorithm is applied on the three images to obtain the code book or a dictionary. The codebook has 8 bins. Each bin is characterized by the center of the SIFT vector, and each image can be represented by the 8 bins, called Bag of Words. SIFT is 128 dimensional, and hence the output dictionary contains 8 * 128 matrix. The descriptor.yml file contains the codeword that the test image generates. From the plot of the codewords, it is observed that the test image Ferrari2 matches closely with the plot of Ferrari1, which indicates these objects are similar.

The same inference is obtained from the histogram plots. K – means is used because the feature vector is 128 dimensional, and there is a relatively less chances of information loss.

References:

1. William Pratt, “Digital Image Processing”

2. http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html

3.http://docs.opencv.org/modules/features2d/doc/object_categorization.html

4.https://www.codeproject.com/articles/619039/bag-of-features-descriptor-on-sift-features

---