EE569 – Homework 3 – Mar 29, 2018

A Report on

Texture Analysis and Segmentation

Edge Detection

Salient Point Descriptors and Image Matching

All the codes were implemented in C++

except for Problem 2 (Parts B and C) where I have used MATLAB for Structured Edge Detection and Performance Evaluation

Problem 3 was solved in C++ and OpenCV - SIFT, SURF, Image Matching and Bag of Words

Submitted by:

ANURAG SYAL

MASTERS IN ELECTRICAL ENGINEERING

MULTIMEDIA AND CREATIVE TECHNOLOGIES

USC ID: 9954-6583-64

Problem 1: Texture Analysis and Segmentation

Part A: Texture Classification
Part B: Texture Segmentation
Part C: Advanced

Abstract and Motivation

In a general sense, "texture Analysis" refers to as assigning a physical image, video or an object into one of a set of predefined categories. *In texture classification, the goal is to assign an unknown sample image to one of a set of known texture classes.* There are 4 problems in the field of texture Analysis:

1. Texture Classification - Assign an unknown sample image to one of a set of known texture classes
2. Texture Segmentation - Partitioning of an image into regions which have homogeneous properties
3. Texture Synthesis - Build a model of image texture, which can then be used for generating the texture
4. Shape from texture - A 2D image is treated as a projection of a 3D scene and apparent texture distortions in the 2D image are used to estimate surface orientations in the 3D scene

Texture analysis is important in many applications of computer image analysis for classification or segmentation of images based on local spatial variations of intensity or color. A successful classification or segmentation requires an efficient description of image texture. Important applications include industrial and biomedical surface inspection, for example for defects and disease, ground classification and segmentation of satellite or aerial imagery, segmentation of textured regions in document analysis, and content-based access to image databases. However, despite many potential areas of application for texture analysis in industry there is only a limited number of successful examples. A major problem is that textures in the real world are often not uniform, due to changes in orientation, scale or other visual appearance. In addition, the degree of computational complexity of many of the proposed texture measures is very high.

Texture classification process involves two phases:

a) Learning phase
b) Recognition phase

In the learning phase, the target is to build a model for the texture content of each texture class present in the training data, which can or cannot comprise of images with known class labels. The texture content of the training images is captured with the chosen texture analysis method, which yields a set of textural features for each image. These features, which can be scalar numbers or discrete histograms or empirical distributions, characterize given textural properties of the images, such as spatial structure, contrast, roughness, orientation, etc.

In the recognition phase the texture content of the unknown sample is first described with the same texture analysis method. Then the textural features of the sample are compared to those of the training images with a classification algorithm, and the sample is assigned to the category with the best match. Optionally, if the best match is not sufficiently good according to some predefined criteria, the unknown sample can be rejected instead.

Texture segmentation is the process of partitioning an image into multiple segments (sets of pixels, also known as super-pixels). *It involves similar process as texture classification followed by labeling.* The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More

precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.
The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (see edge detection). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s). When applied to a stack of images, typical in medical imaging, the resulting contours after image segmentation can be used to create 3D reconstructions with the help of interpolation algorithms like Marching cubes.

Approach and Procedures

Texture Analysis – Using Laws filters and K-Means Clustering

Laws developed a set of two-dimensional masks derived from three simple one-dimensional filters [Laws79] [Laws80]. They are:
L3 = [1, 2, 1] - Level detection
E3 = [-1, 0, 1] - Edge detection
S3 = [-1, 2, -1] - Spot detection

There are 9 2x2 filters that can be generated using the above the 1x3 filters and calculating their tensor product. A few examples are giving below:

-1	2	-1
-2	4	-2
-1	2	-1

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

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

Figure 1: Examples of Laws Filters 3x3, L3S3, S3E3 and E3L3

They can be extended five dimensions in the following way:
Laws convolved these with each other, to provide a set of symmetric and anti-symmetric center-weighted masks with all but the level filters being zero sum. These were convolved in turn with transposes of each other to give various sizes of square masks. He found the most useful to be those shown below. Note that the letters used in the mnemonics stand for Level, Edge, Spot, and Ripple. Sometimes a 5th filter is added called Wave.
L5 = 1/16 * [1, 4, 6, 4, 1]
E5 = 1/6 * [-1, -2, 0, 2, 1]
S5 = 1/4 * [-1, 0, 2, 0, -1]
R5 = 1/16 * [1, -4, 6, -4,1]
W5 = 1/6 * [-1 2 0 -2 1]

0.003906	-0.01563	0.023438	-0.01563	0.003906
0.015625	-0.0625	0.09375	-0.0625	0.015625
0.023438	-0.09375	0.140625	-0.09375	0.023438
0.015625	-0.0625	0.09375	-0.0625	0.015625
0.003906	-0.01563	0.023438	-0.01563	0.003906

0.028	-0.056	0	0.056	-0.028
0.056	-0.111	0	0.111	-0.056
0	0	0	0	0
-0.06	0.111	0	-0.111	0.056
-0.03	0.056	0	-0.056	0.028

Figure 2: Examples of Laws Filters 5x5, E5W5 and L5R5

NOTE: A tensor product is not commutative in nature, i.e. L3E3 ≠ E3L3, but L3E3 = $E3L3^T$

K-Means Initialization in C++:

Initialization is the most important step in clustering. A totally random initialization is bad as the algorithm may settle at a point that is a local minimum indeed but may give a wrong classification. There are several methods for initialization. Three methods were implemented for this:

1. K-Means++ Initiation - David Arthur and Sergei Vassilvitskii
2. **Triangular Initiation – ORIGINAL WORK**
3. Manual initialization - Supervised

Explanation of K-Means initiation methods:

K-Means++

1. Choose one center uniformly at random from among the data points
2. For each data point x, compute D(x), the distance between x and the nearest center that has already been chosen
3. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to $D(x)^2$
4. Repeat Steps 2 and 3 until k centers have been chosen
5. Now that the initial centers have been chosen, proceed using standard k-means clustering

Triangular – This is original work

In this method, I try to find out initializing points using centroid already calculated. The very first centroid is calculated randomly. But after that, I try to find the point that is farthest from the existing centroids. Suppose if we have data on existing 'm' centroids and 'k' centroids must be chosen, then:

-> 1^{st} centroid is selected randomly
-> 2^{nd} centroid is selected which is farthest from the 1^{st} point (selected initial centroid)
-> 3^{rd} centroid is selected which is farthest from the 1^{st} and 2^{nd} points (selected initial centroids)
.
.
.
-> K^{th} centroid is selected which is farthest from the already selected $(K-1)^{th}$ points (selected initial centroids)

1. Let first centroid be the first feature-vector (data point) itself or select a random one (user input is ok)
2. Calculate Euclidean distance of each feature-point from already existing centroids
   a) Store the distances in a vector
   b) Calculate sum of distances of a feature-point from all existing centroids
      1. Find feature-point having maximum distance from the already selected centroids
      2. Find the data point farthest away from all the centroids decided till now
      3. The data point found in above step is an initial centroid so store it

Manual - Supervised

1. Choose fewer than k centers for each of the texture, this is done by a visual inspection
2. Any of the methods above, K-Means or Triangular could be used to carry selection of the further points
3. Now that the initial centers have been chosen, proceed using standard k-means clustering

Texture Classification

Step 1 – Pre-Processing with Balancing (Optional)
The input images are read and DC components is subtracted from each of them. The reason to do so is because we want to eliminate any high feature values that might come in the way of capturing good feature from the image. The DC components have high energy values but very less information content. This is one critical step before feature extraction. The way we do is based on the formula below:

$$Img_{DC_{removed}} = Input_{Img} - (Mean\ of\ all\ pixels\ in\ each\ of\ the\ input\ images)$$

Step 2 – Feature Extraction
The 9 Laws' filters were created as per the methods mentioned above and were applied (convolved) with each of the Texture1-Texture12 images. So now each of the 12 images had 9 images of their own which represented the 9D feature vector for each training image. Thus, each of the training image gave 9 output images that represented the filter response of the entire filter bank in terms of the training data.

Step 3 – Normalizing (Optional)
The above outputs of 12 x 9 were then normalized to find the energy of feature vector. The formula used was as under:

$$Feature_{Vector_{normalized}} = \frac{Feature_{Vector} - (Mean\ of\ all\ the\ data\ in\ the\ feature\ vector)}{Standard\ deviation\ of\ the\ feature\ vector}$$

Step 3 – Clustering using K-Means
To classify and unknown set of textures using the texture types rock, grass, straw and weave, the known textures were first clustered into K points or centroids. So, the feature space is now known to consist of K centroids / clusters/ points. This was then passed into the K-means clustering algorithm as described above to get the best possible cluster centroids based on an iterative process till the Euclidean distance between that of Centroids and Data is the minimum and there is no change on the Centroids even on performing any further iterations.

Implementation in C++:

1. Read all the 12 images and store them in 3D vector of dimensions 128 x 128 x 1
2. Balance the images, i.e. find the Mean of each of the images and subtract the mean from each of the images to get the DC component removed image
3. Create the 9 Laws' filter using the steps mentioned above and apply each of the DC component removed image to get a 12x9 feature vector matrix
4. Normalize the 12x9 matrix obtained in Step 4
5. Perform the K-means clustering over the 12 x 9 data by using either K-Means++ OR Triangular Initialization OR Supervised Means (manually entering means) until several iterations till there is no more change in the value of centroids. The process done using K-Means++ OR Triangular Initialization is unsupervised learning of classifying the data because the initial guess was random.

Texture Segmentation

Step 1 – Windowing
In this step, each pixel is considered as an image. A pixel cannot be an image, because energy can only be calculated for a region and not a pixel, an M x M window is considered around the pixel. M can be taken 25, 35 or 45 for this report. The windows are stored is a vector.

Step 2 – Pre-Processing with Balancing (Optional)
The input images are read and DC components is subtracted from each of them. The reason to do so is because we want to eliminate any high feature values that might come in the way of capturing good feature from the image. The DC components have high energy values but very less information content. This is one critical step before feature extraction. The way we do is based on the formula below:

$$Img_{DC_{removed}} = Input_{Img} - (Mean\ of\ all\ pixels\ in\ each\ of\ the\ input\ images)$$

NOTE: Since the data set involved in this problem in very large number, a memory of this size will give segmentation faults or Process killed error. Because, the data set will become too large to handle.

Calculation of memory required:
At the end of step 1, we have a 4-dimensional vector of the size:
= Number of Pixels x M x M x Bytes Per Pixel
= (Width x Height) x M x M x 1 (Since it's a grey image)
= (600 x 450) x 25 x 25 x 1
= 270,000 x 25 x 25 x 1

So, steps 1-3 are performed of after the other in a single loop itself. In this way, the data is manageable.

Step 3 – Feature Extraction
The 9 Laws' filters were created as per the methods mentioned above and were applied (convolved) with each of the Texture1-Texture12 images. So now each of the 12 images had 9 images of their own which represented the 9D feature vector for each training image. Thus, each of the training image gave 9 output images that represented the filter response of the entire filter bank in terms of the training data.

Step 4 – Normalizing (Optional)
The above outputs of 12 x 9 were then normalized to find the energy of feature vector. The formula used was as under:

$$Feature_{Vector_{normalized}} = \frac{Feature_{Vector} - (Mean\ of\ all\ the\ data\ in\ the\ feature\ vector)}{Standard\ deviation\ of\ the\ feature\ vector}$$

Step 5 – Labelling - Clustering using K-Means
The resulting vector after step 4 is passed into the K-means clustering algorithm as described above to get the best possible cluster. After a cluster is assigned, each cluster is assigned a label. Since there are k-clusters, a constant can be multiplied with the index of each label to give a value between [0-255] and generate the final image.

Implementation in C++:

1. Read the image pixel-by-pixel and generate an M x M size window around it
2. Balance the window and find the Mean of the window and subtract the mean from each of the images to get the DC component removed image
3. Create the 9 Laws' filter using the steps mentioned above and apply each of the DC component removed window to get a 1x9 feature vector matrix
4. Repeat steps 1-3 till all the pixels are done. Final size of the vector will be 270,000 x 9
5. Normalize the 270,000 x 9 matrix obtained in Step 4
6. Perform the K-means clustering over the 270,000 x 9 data by using either K-Means++ OR Triangular Initialization OR Supervised Means (manually entering means) until several iterations till there is no more change in the value of centroids. The process done using K-Means++ OR Triangular Initialization is unsupervised learning of classifying the data because the initial guess was random

Results and Discussion

Texture Segmentation

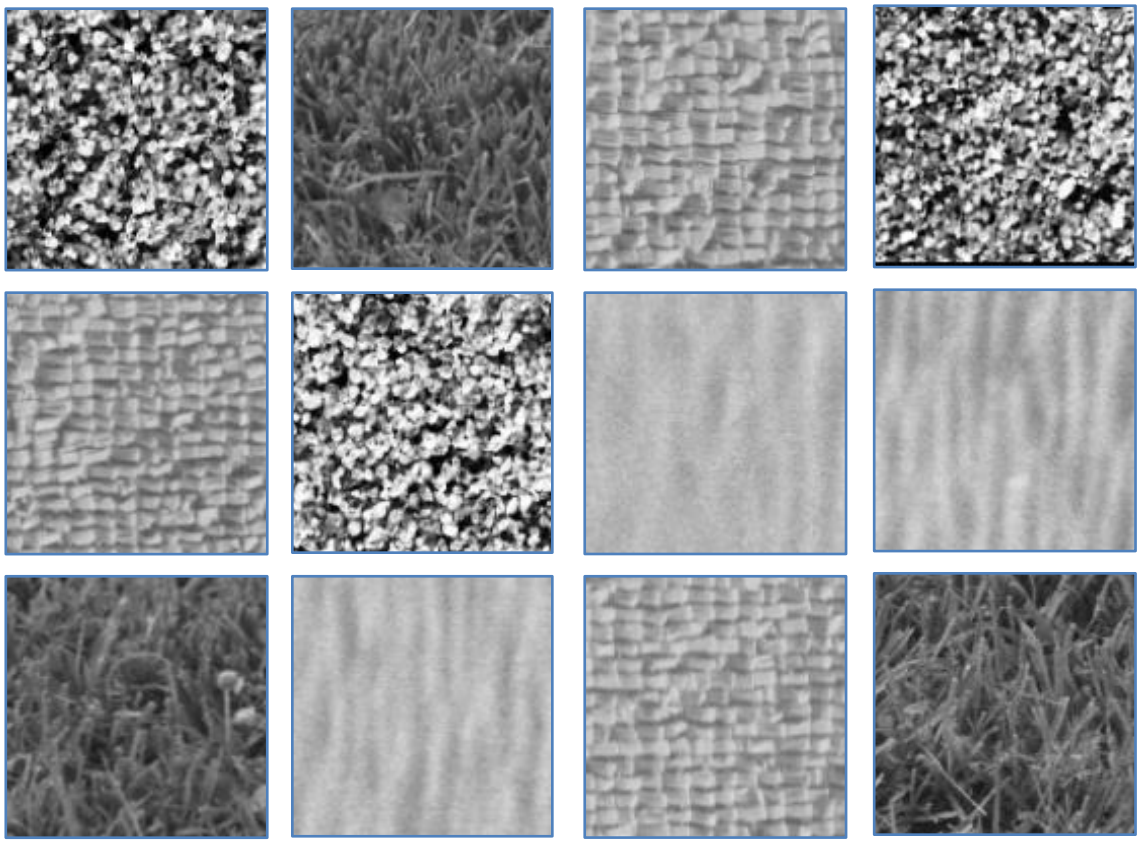


Figure 3: Textures 1-12

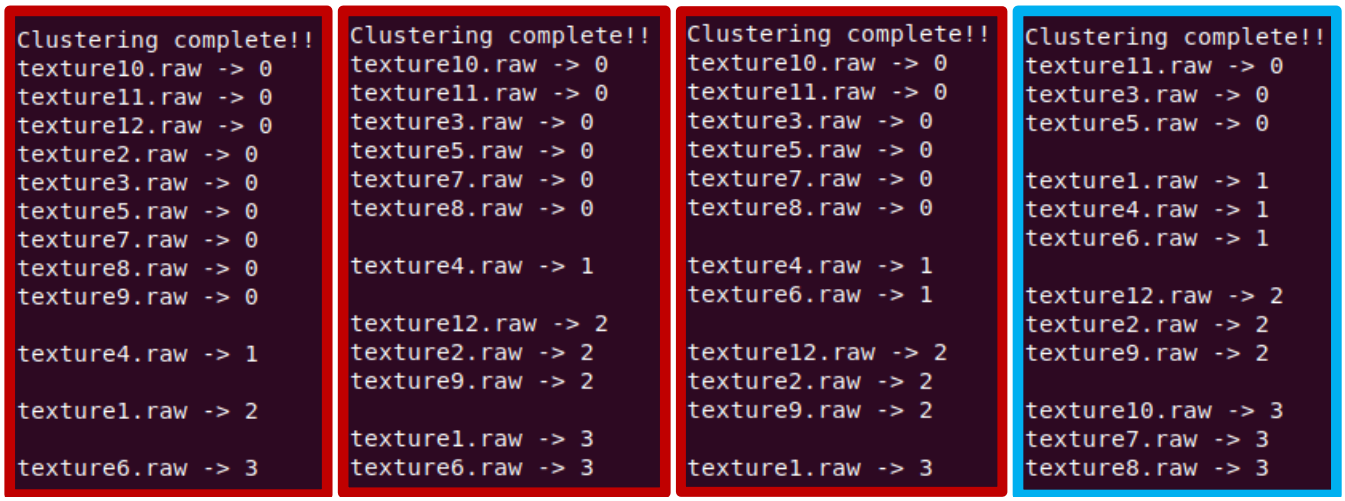


Figure 4: Results using random Initialization

a) Balancing and Normalizing b) Balancing Only c) Normalizing Only d) No Balancing and No Normalizing

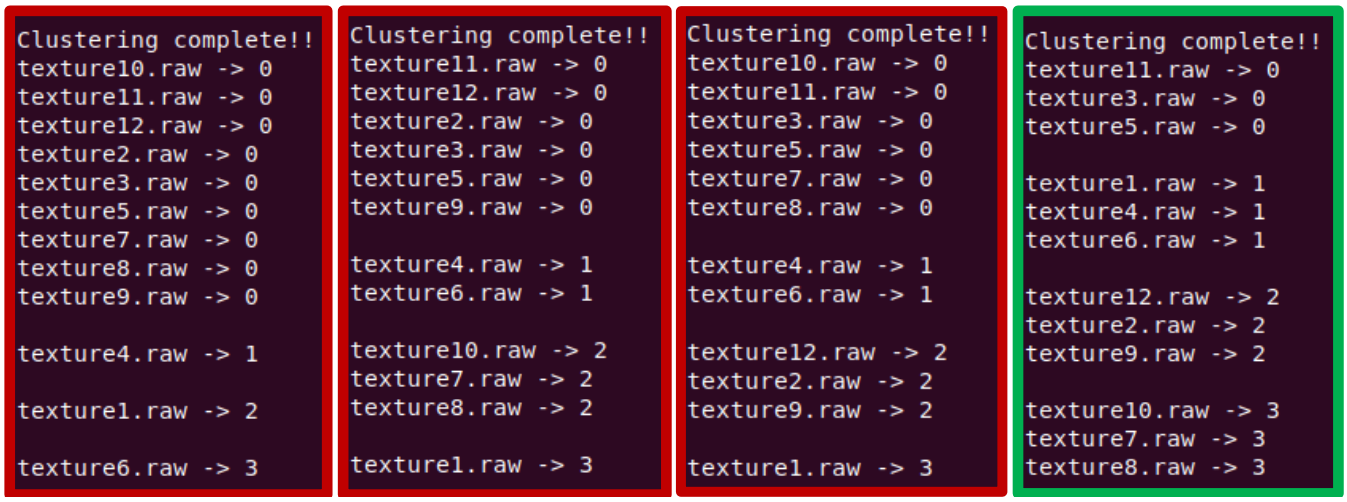


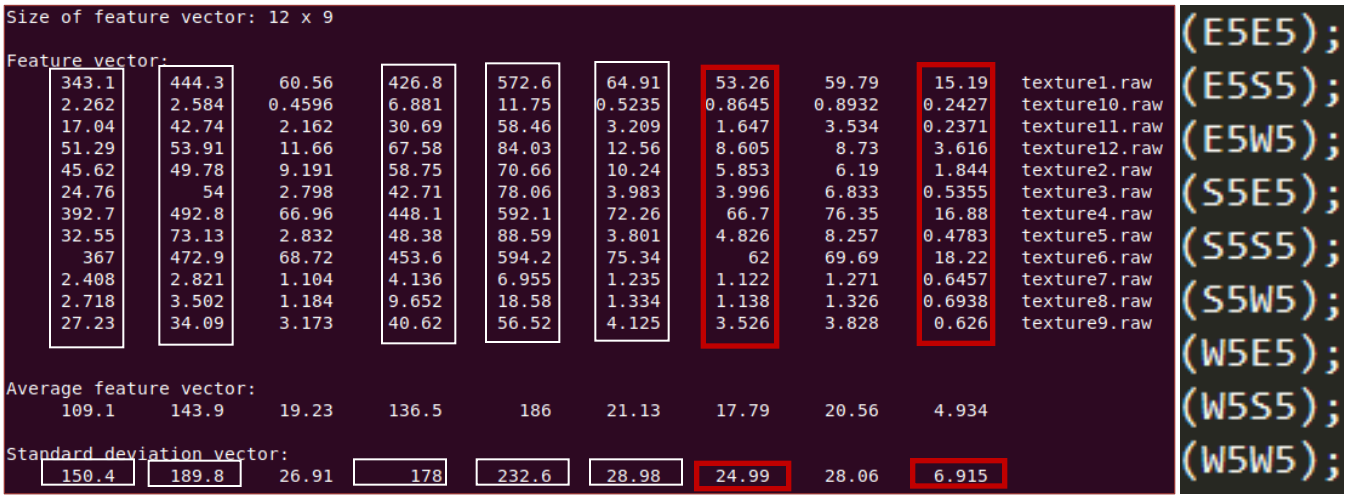
Figure 5: Results using K-Means++ Initialization

a) Balancing and Normalizing b) Balancing Only c) Normalizing Only d) No Balancing and No Normalizing

Figure 6: Results using Triangular Initialization

a) Balancing and Normalizing b) Balancing Only c) Normalizing Only d) No Balancing and No Normalizing

Figure 7: Results under Supervision
a) K-Means++ b) Triangular Initialization



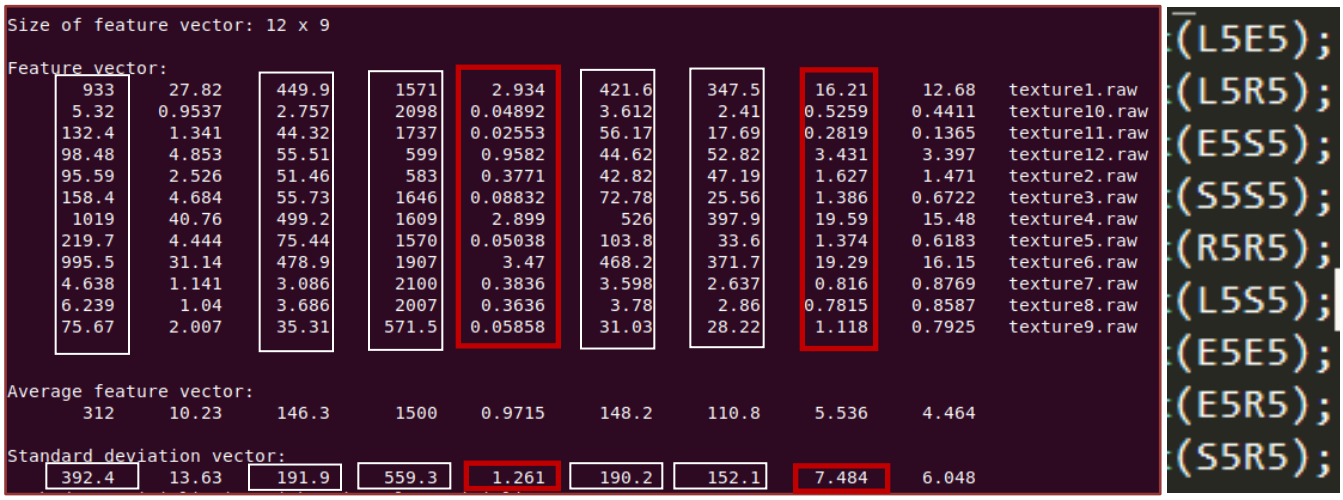


Figure 8: Feature Vector - 12 x 9 - Using Laws' Filter size 5 – a) Configuration 1 b) Configuration 2

Observations

1) Random Initialization is not a proper method for initialization, roughly all the results are **WRONG**.
2) The steps of normalization and balancing swayed the data and gave wrong results in most cases

3) **K-Means++ initialization converges faster than random initialization and gave correct results** when the steps of balancing and normalization were not performed
4) **Triangular initialization converges faster than K-Means++ but gave correct results** when the step of normalization were not performed. **Triangular initialization in general worked better than K-Means++**.
5) **Under supervision, K-Means++ gave wrong results but Triangular Initiation gave correct results**
6) **Good classifying features (descending):**
   a) **Configuration 1:** S5S5(5th Col), S5E5(4th Col), E5S5 (2nd Col), S5W5 (6th Col), E5E5 (1st Col)
   b) **Configuration 2:** L5E5 (1st Col), E5S5 (3rd Col), L5S5 (6th Col), E5E5 (7th Col), S5S5 (4th Col)
7) **Weak classifying features (descending):**
   a) **Configuration 1:** W5W5 (9th Col) and W5E5 (7th Col)
   b) **Configuration 2:** R5R5 (5th Col) and E5R5 (8th Col)
8) Based on the above observation, the decision was taken to move from the 1st Laws' Filter configuration to the second one because the features had better energy compaction and hence better discriminating power
9) The decision about which feature is food and which is not can be taken from the average and standard deviation of that feature. Another metric is Average/Unit Standard Deviation.

Best Results

Under optimum conditions, both K-Means++ and Triangular Initialization gave correct results.
Textures 1, 4 and 6 were correctly grouped together
Textures 2, 9 and 12 were correctly grouped together
Textures 3, 5 and 11 were correctly grouped together
Textures 7, 8 and 10 were correctly grouped together

Texture Segmentation

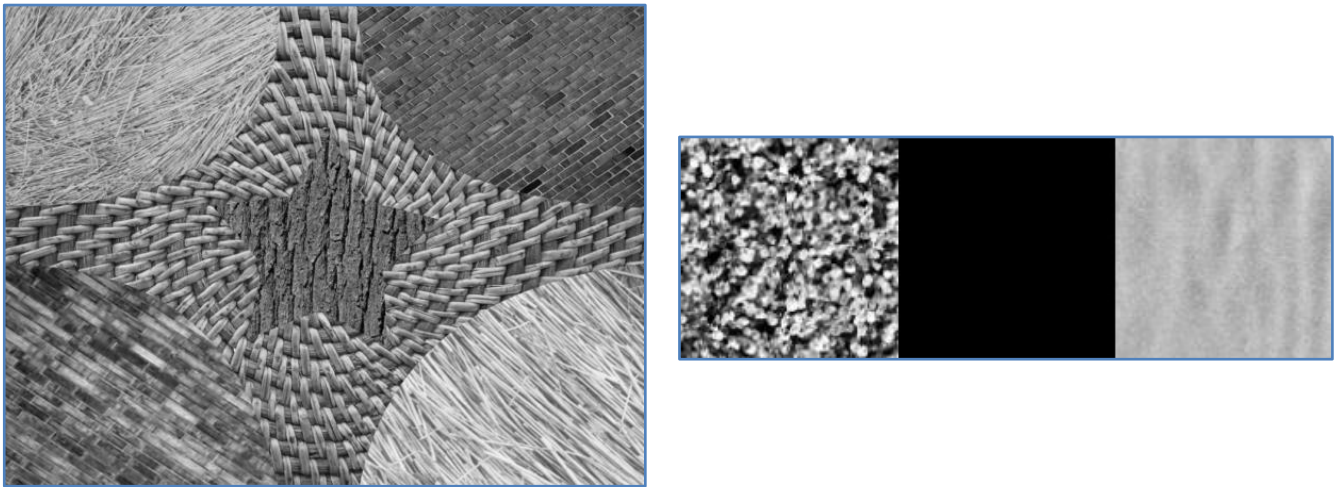


Figure 9: Original Test Images for Texture Segmentation a) Comb.raw (600 x 450) b) Simple.raw (379 x 128)

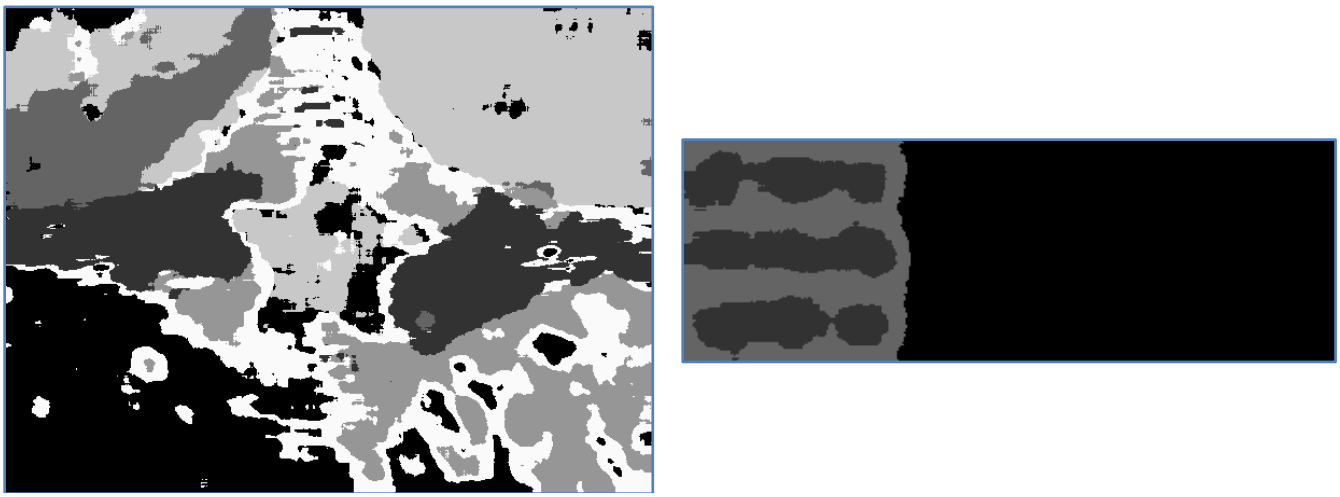


Figure 10: Results: Laws Filter Size 5, K-Means++, Balancing and Normalizing a) Comb.raw b) Simple.raw

Figure 11: Results: Laws Filter Size 5, Supervised, No balancing No normalizing a) Comb.raw b) Simple.raw

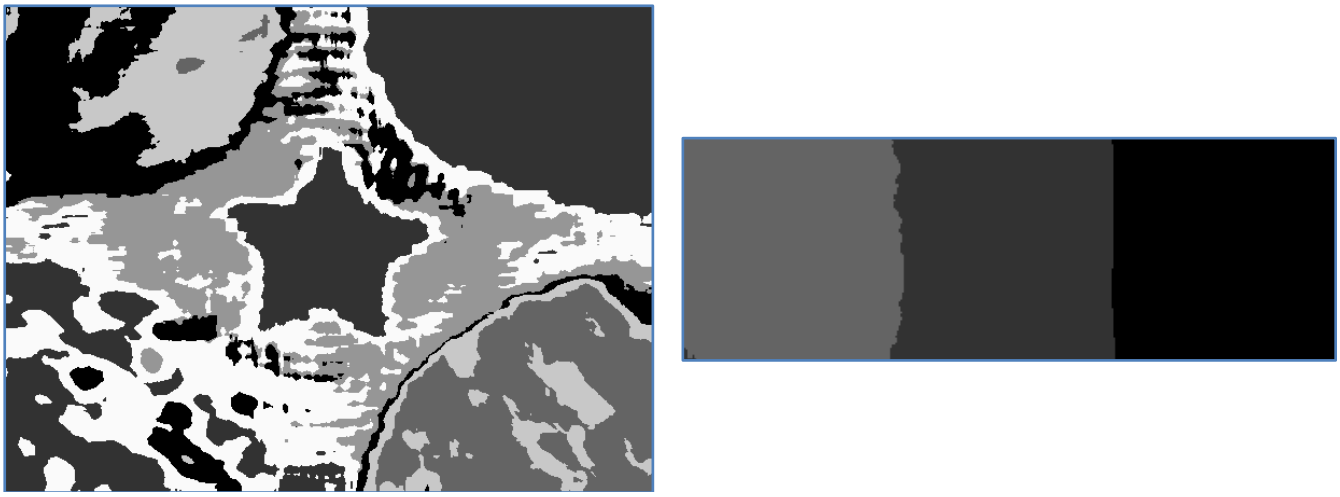


Figure 12: Results: Laws Filter Size 5, Triangular Initialization, No Balancing No Normalizing a) Comb.raw b) Simple.raw

Figure 13: Results: Laws Filter Size 5, Supervised, Without Balancing and Normalizing a) Comb.raw b) Simple.raw

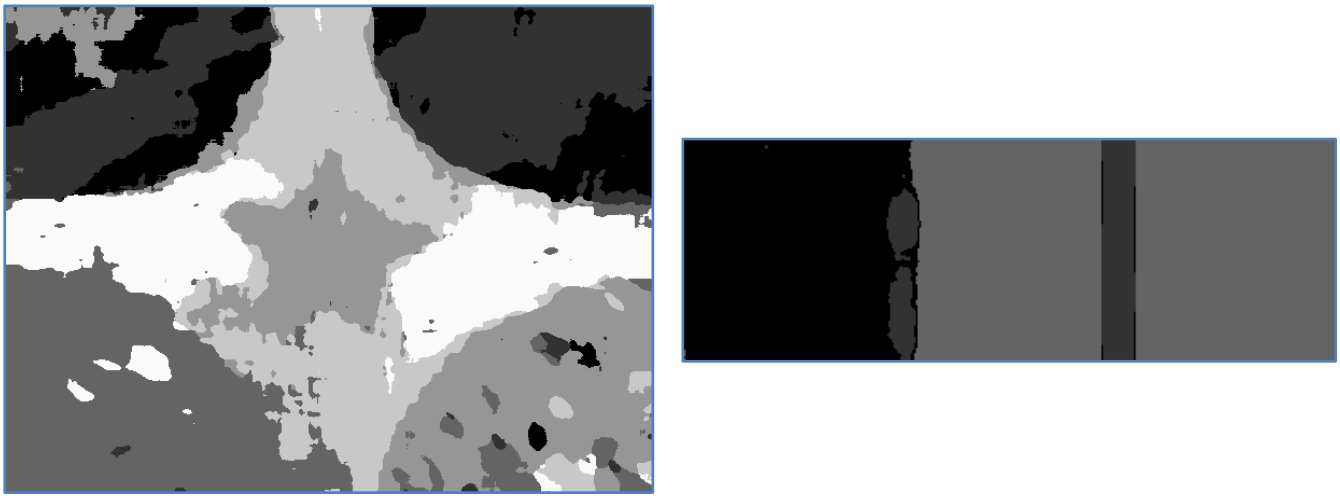


Figure 14: Results: Laws Filter Size 3, Supervised, Balancing and Normalizing a) Comb.raw b) Simple.raw

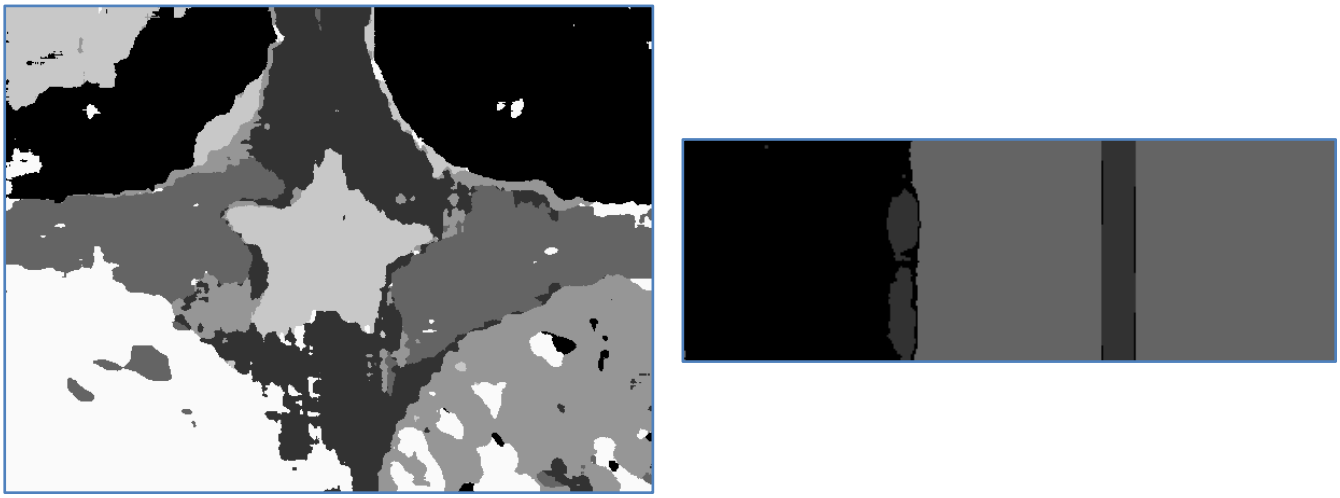


Figure 15: Results: Laws Filter Size 3, K-Means++, Balancing and Normalizing a) Comb.raw b) Simple.raw

Observations

a) Random Initialization is not a proper method for initialization, roughly all the results are **WRONG**.
b) The steps of normalization and balancing swayed the data and gave wrong results in most cases
c) **K-Means++ initialization converges faster than random initialization and gave correct results** when the steps of balancing and normalization were not performed
d) **Triangular initialization converges faster than K-Means++ but gave correct results** when the step of normalization were not performed. **Triangular initialization in general worked better than K-Means++**.
e) **Under supervision, K-Means++ gave wrong results but Triangular Initiation gave correct results**

Experiment with Different Window Sizes

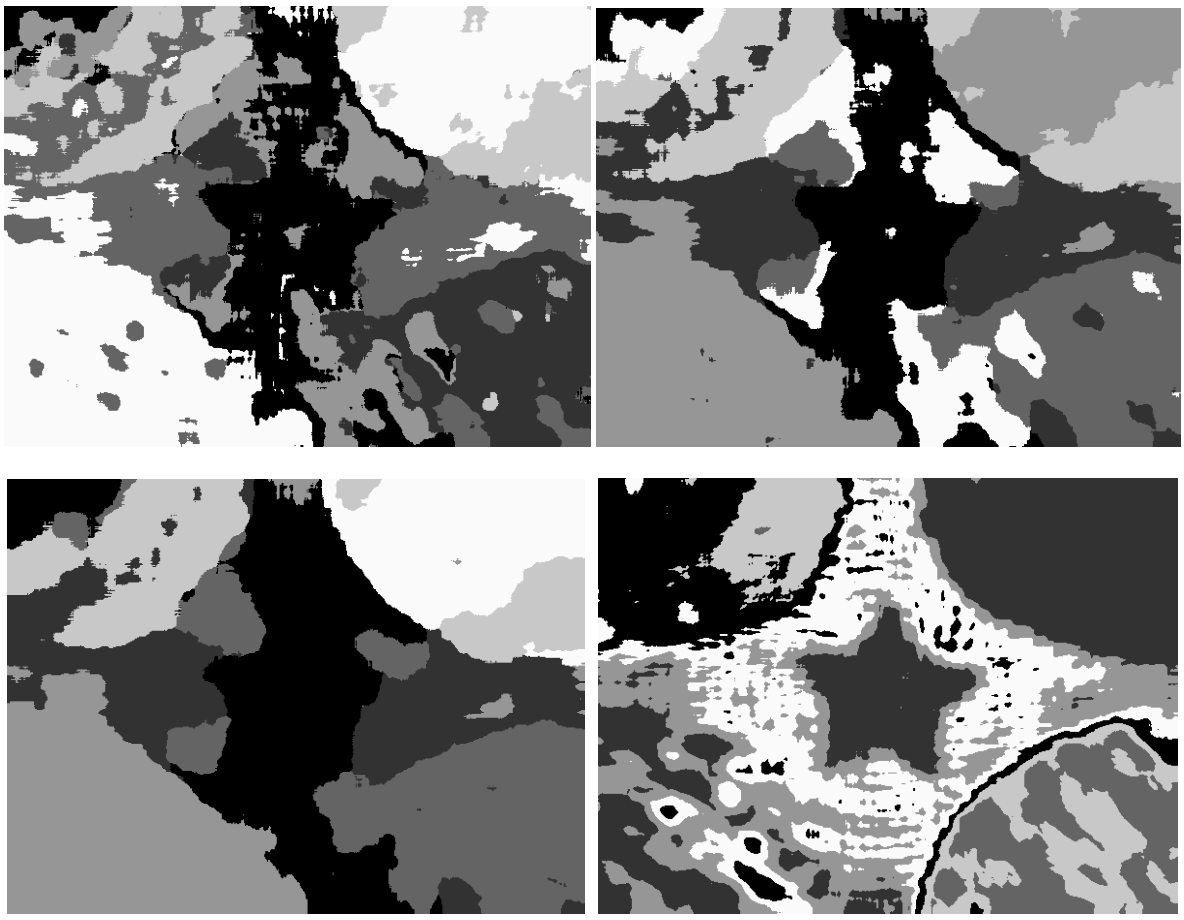


Figure 16: Window size 25, 35, 45 and 55

Observation: Images get smoother and results improves with increase in window size, but time taken is more.

Texture Segmentation using PCA

PCA essentially works on Parseval's theorem or Energy-Compaction theorem in general. This simply means that, all the data can be considered as nothing but energy. Now, in a stream of data, clusters or vectors of components can be formed. For example, in case of sound signals, the various frequencies present in the signal become the components. Number of components determine the dimensionality of the signal. Such that:

Energy of Signal = Energy of frequency 1 + Energy of frequency 2 + …. + Energy of frequency 'n'

In case of a noisy signal, some of the energy of the signal is shared by the noise as well, such that:

Energy of Signal = Energy of Actual Signal + Energy of Noise

Since noise can also be broken down into components we can say that:

Energy of Signal = [Energy of frequency 1 + Energy of frequency 2 + …. + Energy of frequency 'n'] +
[Energy of Noisy frequency 1 + Energy of Noisy frequency 2 + …. + Energy of Noisy frequency 'n']

In real life cases, we generally don't know that which frequency is exactly contributing to the noise. In general, the practical signals have a lot of components, but much of their energy compacted into a very few components:

Energy of Signal = Energy of MAJOR frequency 1 + Energy of MAJOR frequency 2 + …. + Energy of MAJOR 'm'
+ Energy of MINOR frequency (n-2) + Energy of MINOR frequency (n-1) + …. + Energy of MINOR frequency (n)

OR

Energy of Signal = Energy of MOST PROMINENT frequencies + ENERGY of LEAST PROMINENT frequencies

Now, PCA helps in getting rid of the LEAST prominent frequencies, thus reducing dimensionality of the signal. Since most of the energy is concentrated only in the very first few components, the signal can be reconstructed easily.

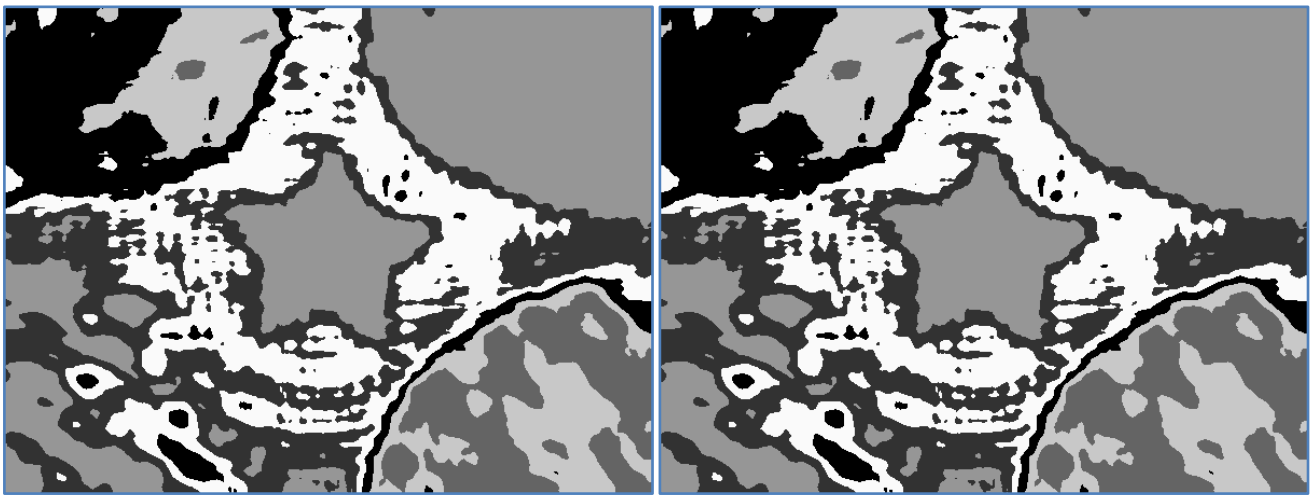


Figure 17: Results: Laws Filter Size 3, Window Size 25, Triangular Initialization a) Without PCA b) With PCA

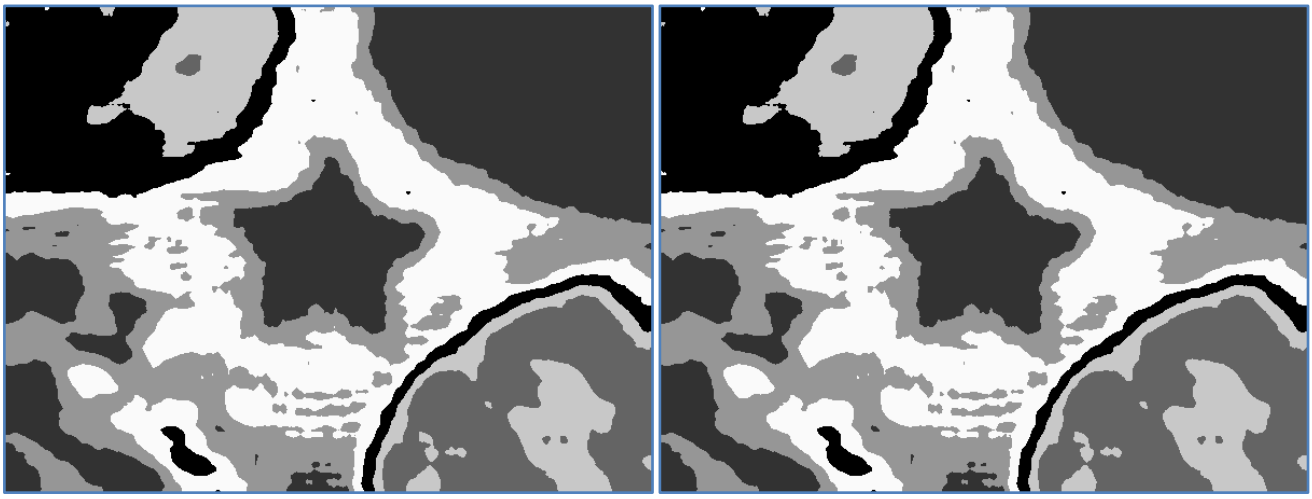


Figure 18: Results: Laws Filter Size 3, Window Size 35, Triangular Initialization a) Without PCA b) With PCA

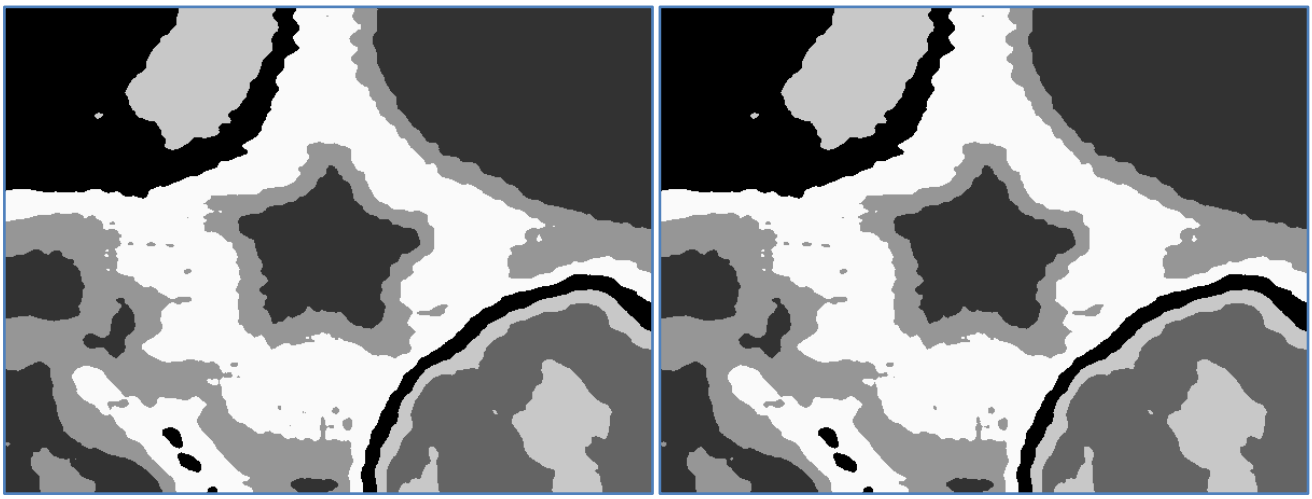


Figure 19: Results: Laws Filter Size 3, Window Size 45, Triangular Initialization a) Without PCA b) With PCA

Observations

a) Number if features for PCA = 3, Total features = 9

b) PCA achieves the best output for each window size (25, 35 and 45)

c) This proves that small number features can help in classifying big data if the discriminant power is good

d) An increase in Window size smoothen the results but window size 35 and 45 give similar results. In fact, output improved from 25 to 30 and 30 to 36. But around window size 38, output almost became constant.

Conclusion

Texture classification was completed and the textures were allotted to their correct categories.

Texture Segmentation is a complex problem, but good results can be obtained by having the correct window size and PCA. PCA doesn't improves the result but help faster calculation and lower run-time.

Problem 2: Edge Detection

Part A: Basic Edge Detection

Abstract and Motivation

Sobel Detector - The Sobel operator is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasizing edges. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. The Sobel operator is based on convolving the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. On the other hand, the gradient approximation that it produces is relatively crude, for high-frequency variations in the image.

Zero-crossing Detector (Using Laplacian of Gaussian - LoG) - The zero-crossing detector looks for places in the Laplacian of an image where the value of the Laplacian passes through zero --- i.e. points where the Laplacian changes sign. Such points often occur at `edges' in images --- i.e. points where the intensity of the image changes rapidly, but they also occur at places that are not as easy to associate with edges. It is best to think of the zero-crossing detector as some sort of feature detector rather than as a specific edge detector. Zero-crossings always lie on closed contours, and so the output from the zero-crossing detector is usually a binary image with single pixel thickness lines showing the positions of the zero-crossing points.

The starting point for the zero-crossing detector is an image which has been filtered using the Laplacian of Gaussian filter. The zero-crossings that we get in the result are strongly influenced by the size of the Gaussian used for the smoothing stage of this operator. As the smoothing is increased then fewer and fewer zero-crossing contours will be found, and those that do remain will correspond to features of larger and larger scale in the image.

Approach and Procedures

There are 2 methods for Edge Detection that have been explored here: Sobel Filter and Zero-Crossing Detector.

`Sobel Edge Detector`

The operator uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives – one for horizontal changes, and one for vertical. If we define A as the source image, and Gx and Gy are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

where * here denotes the 2-dimensional signal processing convolution operation.

Since the Sobel kernels can be decomposed as the products of an averaging and a differentiation kernel, they compute the gradient with smoothing. For example, $\mathbf{G}_x$ can be written as

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

The x-coordinate is defined here as increasing in the "right"-direction, and the y-coordinate is defined as increasing in the "down"-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:

$$\mathbf{G} = \sqrt{{\mathbf{G}_x}^2 + {\mathbf{G}_y}^2}$$

Using this information, we can also calculate the gradient's direction:

$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

where, for example, **Θ** is 0 for a vertical edge which is lighter on the right side.

Comparison with Prewitt Operator

The Sobel operator is very similar to Prewitt operator. It is also a derivate mask and is used for edge detection. Like Prewitt operator Sobel operator is also used to detect two kinds of edges in an image:

Vertical direction

Horizontal direction

Difference with Prewitt Operator - The major difference is that in Sobel operator the coefficients of masks are not fixed and they can be adjusted according to our requirement unless they do not violate any property of derivative masks.

Following are the vertical and horizontal masks for Sobel Operator, respectively:

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

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

Figure 1: Vertical and Horizontal Masks for Sobel

The Sobel masks works exactly same as the Prewitt operator masks. The one difference being, it has “2” and “-2” values in center of first and third columns in vertical masks and center of the first and third rows of the horizontal masks. When applied on an image this mask will highlight the vertical edges and horizontal edges, respectively.

How it works - When we apply this mask on the image it prominent vertical edges. It simply works like as first order derivate and calculates the difference of pixel intensities in an edge region. As the center column is of zero so it

does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also, the center values of both the first and third column is 2 and -2 respectively.

Zero-Crossing Edge detector

In general, a simple differentiation is not enough for finding the edges. Sometimes, we need a better understanding of how rapidly is there is a change in the intensity near a pixel. For that, we need a double-differentiation-kind-of method. That's where Laplacian of Gaussian comes into picture.

Laplacian of Gaussian (LoG) - The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. It is isotropic because we cannot find the direction from the result. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection (see zero crossing edge detectors). The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian smoothing filter to reduce its sensitivity to noise, and hence the two variants will be described together here. The operator normally takes a single gray-level image as input and produces another gray-level image as output.

How It Works - The Laplacian L(x,y) of an image with pixel intensity values I(x,y) is given by:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Since the input image is represented as a set of discrete pixels, we must find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Two commonly used small kernels are shown here:

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Using one of these kernels, the Laplacian can be calculated using standard convolution methods. Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

In fact, since the convolution operation is associative, we can convolve the Gaussian smoothing filter with the Laplacian filter, and then convolve this hybrid filter with the image to achieve the required result. Doing things this way has two advantages:

- Since both the Gaussian and the Laplacian kernels are usually much smaller than the image, this method usually requires far fewer arithmetic operations
- The LoG (`Laplacian of Gaussian') kernel can be pre-calculated in advance so only one convolution needs to be performed at run-time on the image

The 2-D LoG function centered on zero and with Gaussian standard deviation **σ** has the form:

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

Note that as the Gaussian is made increasingly narrow, the LoG kernel becomes the same as the simple Laplacian kernels shown in Figure 1. This is because smoothing with a very narrow Gaussian (σ < 0.5 pixels) on a discrete grid has no effect. Hence on a discrete grid, the simple Laplacian can be viewed as a limiting case of the LoG for narrow Gaussians.

The LoG operator calculates the second spatial derivative of an image. This means that in areas where the image has a constant intensity (i.e. where the intensity gradient is zero), the LoG response will be zero. If there is an intensity change near a pixel, the LoG response will be positive on the darker side, and negative on the lighter side. This means that at a reasonably sharp edge between two regions of uniform but different intensities, the LoG response will be:

- Zero at a long distance from the edge
- Positive just to one side of the edge
- Negative just to the other side of the edge
- Zero at some point in between, on the edge itself.

Figure below illustrates the response of the LoG to a step edge:

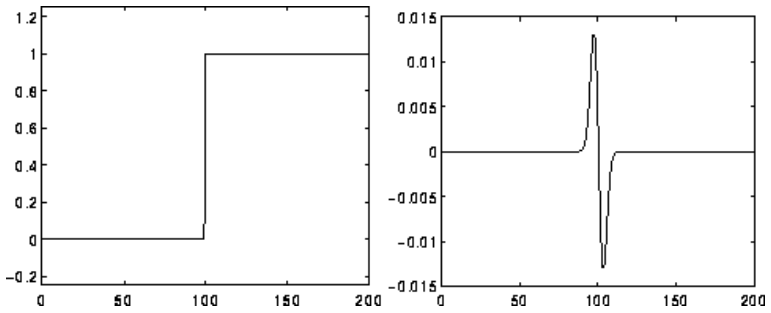


Figure 2: Response of LoG towards a step edge

However, zero crossings also occur at any place where the image intensity gradient starts increasing or starts decreasing, and this may happen at places that are not obviously edges. Often zero crossings are found in regions of very low gradient where the intensity gradient wobbles up and down around zero.

Once the image has been LoG filtered, it only remains to detect the zero crossings. This can be done in several ways. **The simplest is to simply threshold the LoG output at zero**, to produce a binary image where the boundaries between foreground and background regions represent the locations of zero crossing points. These boundaries can then be easily detected and marked in single pass, e.g. using some morphological operator. For instance, to locate all boundary points, we simply must mark each foreground point that has at least one background neighbor.

The problem with this technique is that will tend to bias the location of the zero-crossing edge to either the light side of the edge, or the dark side of the edge, depending on whether it is decided to look for the edges of foreground regions or for the edges of background regions.

A better technique is to consider points on both sides of the threshold boundary, and choose the one with the lowest absolute magnitude of the Laplacian, which will hopefully be closest to the zero crossing. Since the zero crossings generally fall in between two pixels in the LoG filtered image.

But a more superior technique is to check for zero crossing only on a pixel that has recorded a 0-value in a ternary map. If the any of conditions: Left Neighbor ≠ Right Neighbor OR Top Neighbor ≠ Bottom Neighbor, them the pixel is considered to be a pixel ion the edge.

Results and Discussion

Results for Sobel Filter – Boat.raw

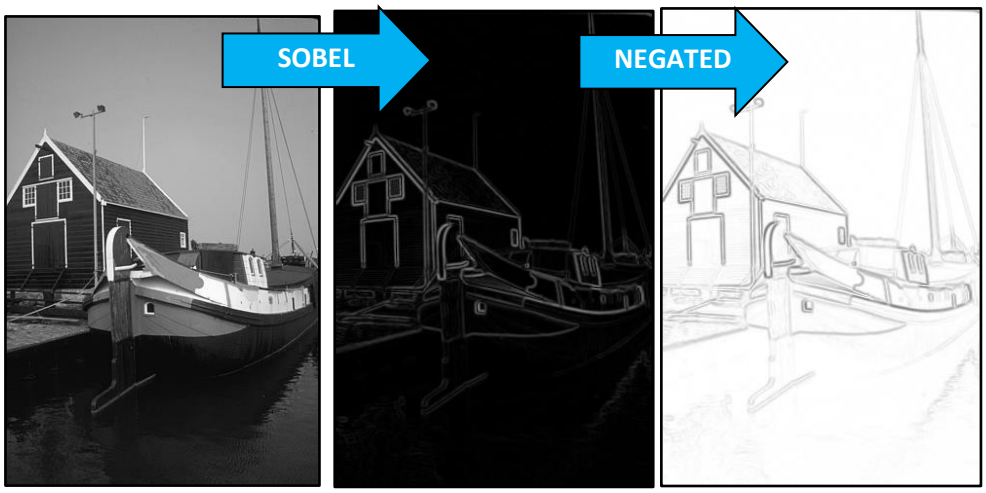


Figure 3: 1) Gray 2) Denoised 3) Sobel filter 4) Negated (Edge map without thresholding)

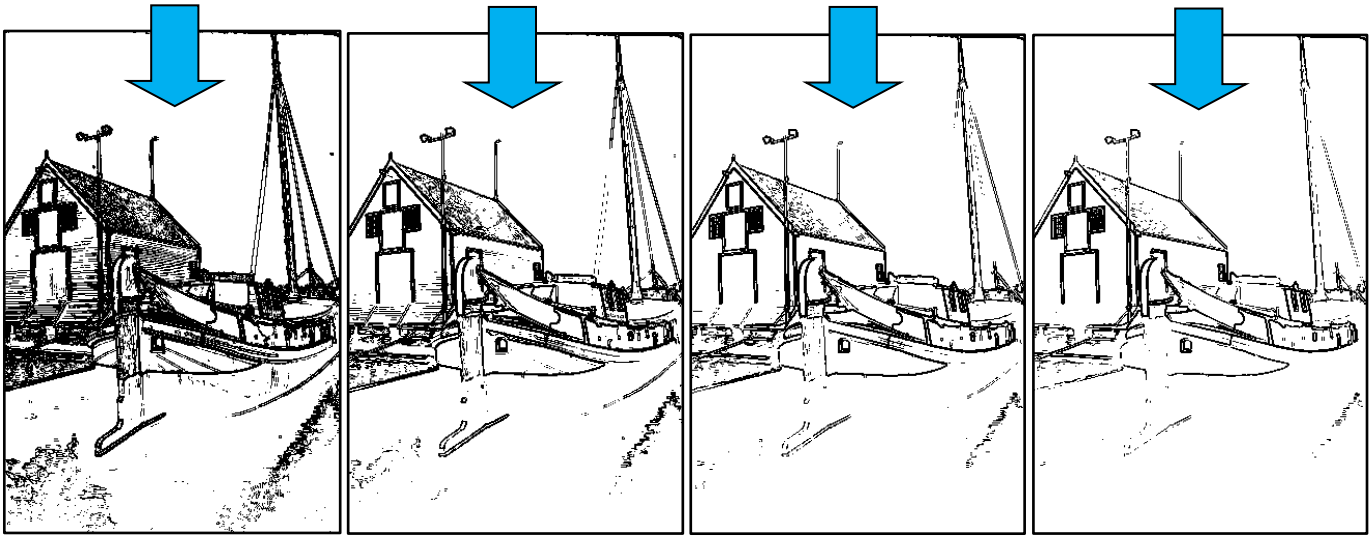


Figure 4: Binary Edge-map at different thresholds 5%, 10%, 15% and 20%

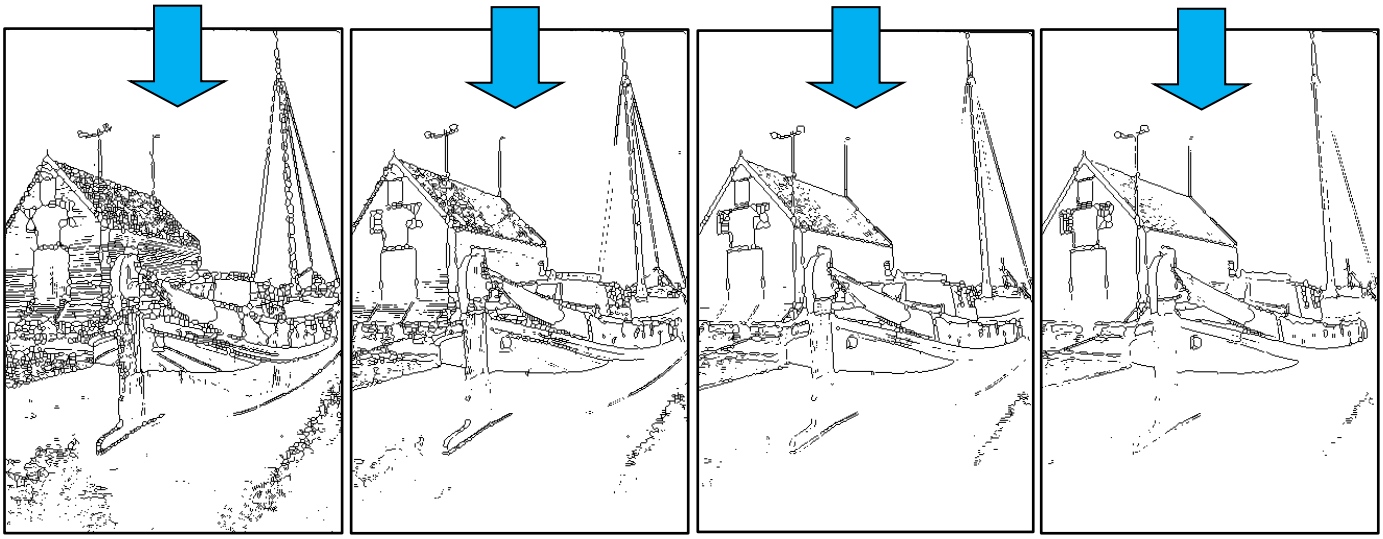


Figure 5: Thinning of edge-maps at different thresholds 5%, 10%, 15% and 20%

Results for Sobel Filter – Boat_noisy.raw

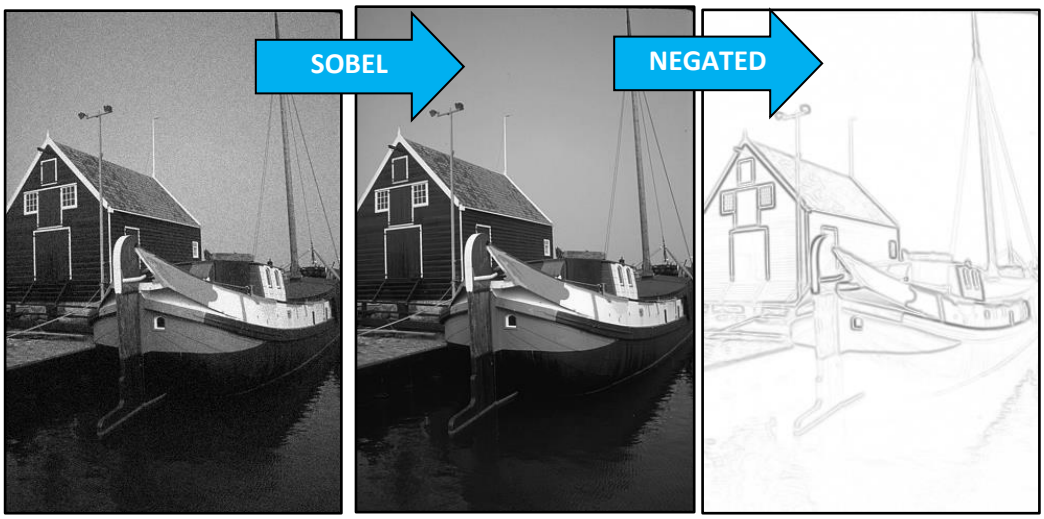


Figure 6: 1) Gray 2) Denoised 3) Sobel filter 4) Negated (Edge map without thresholding)

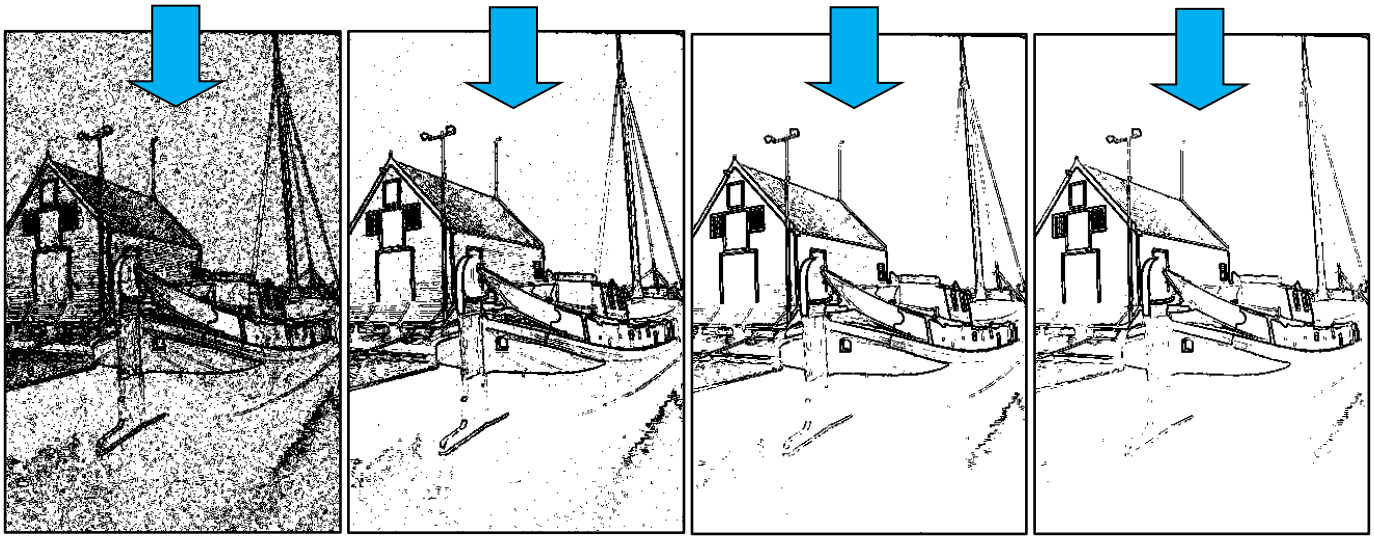


Figure 7: Binary Edge-map at different thresholds 5%, 10%, 15% and 20%

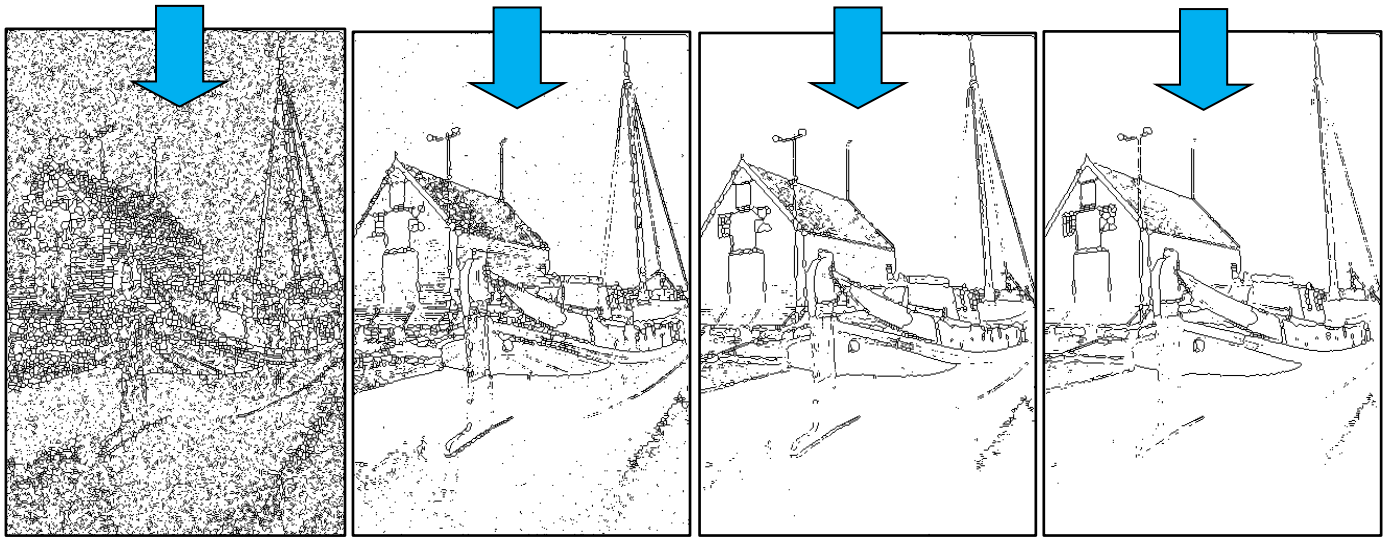


Figure 8: Thinning of edge-maps at different thresholds 5%, 10%, 15% and 20%

Comparison when denoising is not done – Threshold = 15%

Figure 9: Differentiated with Sobel filter – With denoising and Without Denoising

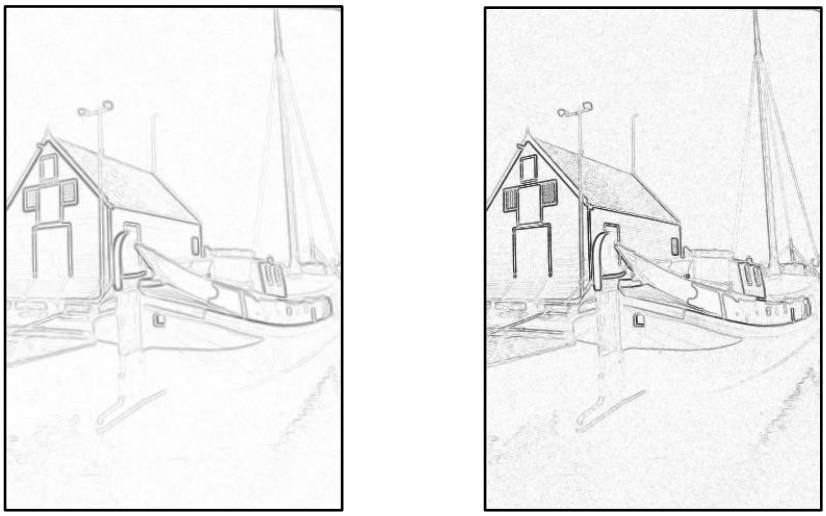


Figure 10: Probability map – without Denoising and with denoising

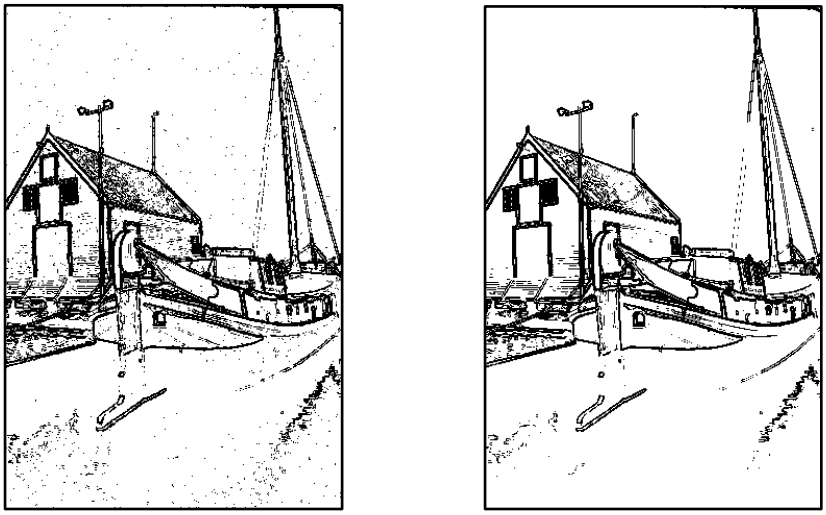


Figure 11: Binary edge map – without Denoising and with denoising

Results for LoG Filter – Boat.raw

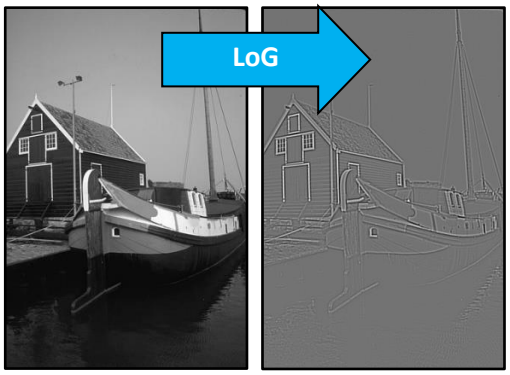


Figure 12: 1) Gray 2) LoG filter

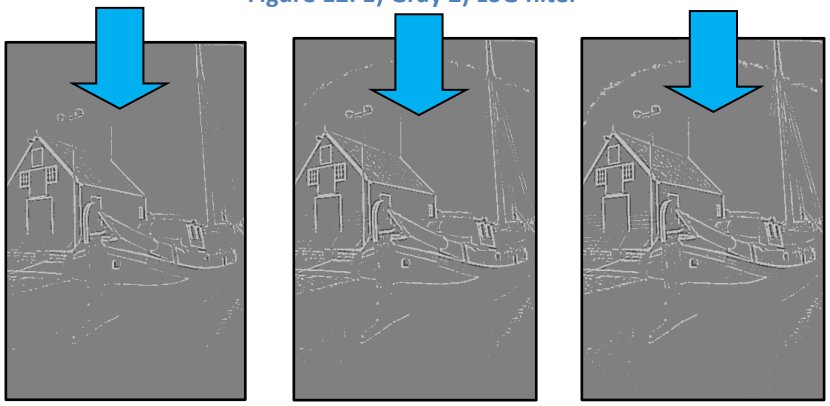


Figure 13: Ternary Edge-maps at different thresholds 5%, 7% and 9%

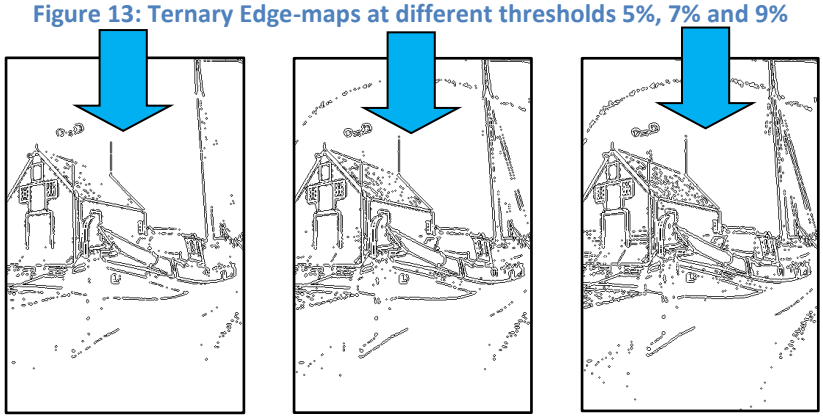


Figure 14: Zero-Crossing at different thresholds 5%, 7% and 9%

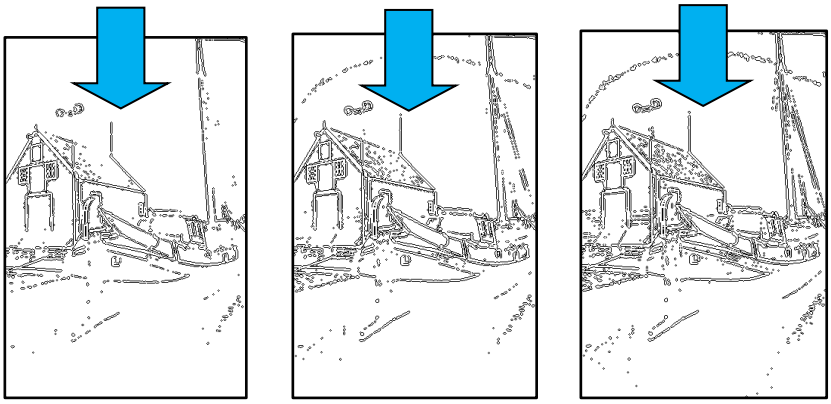


Figure 15: result after Thinning at different thresholds 5%, 7% and 9%

Results for LoG Filter – Boat_noisy.raw

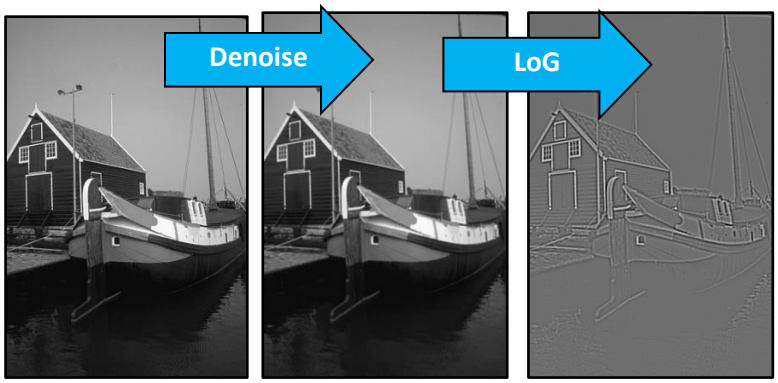


Figure 16: 1) Gray 2) LoG filter

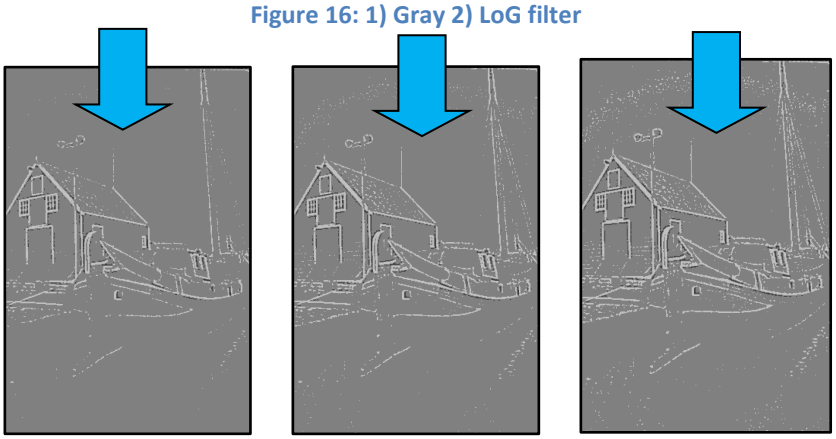


Figure 17: Ternary Edge-maps at different thresholds 5%, 7% and 9%

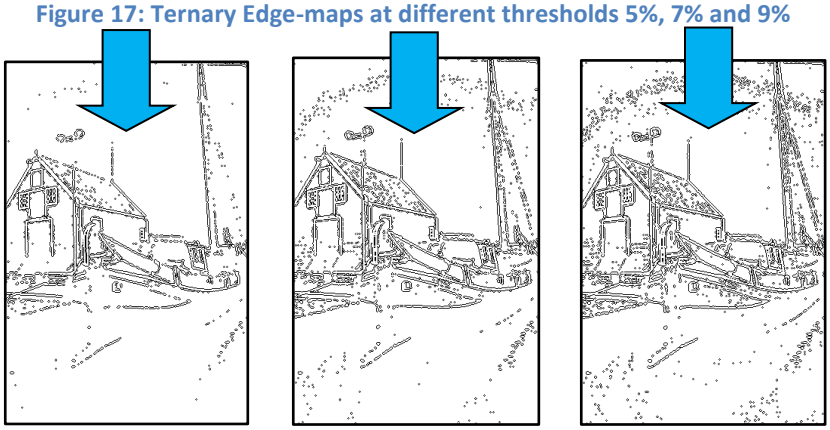


Figure 18: Zero-Crossing at different thresholds 5%, 7% and 9%

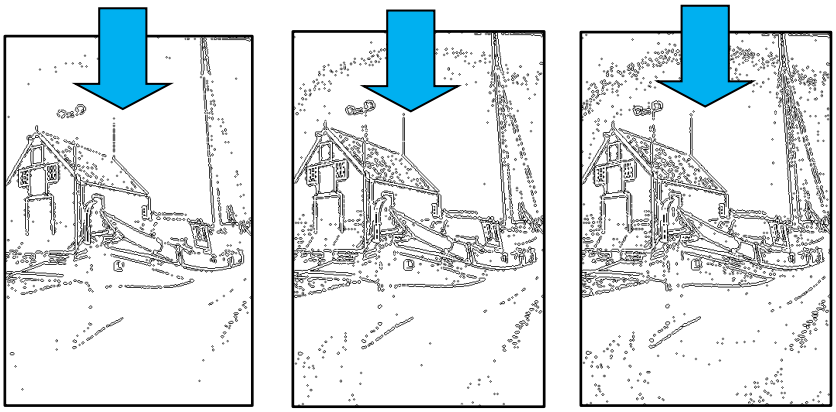


Figure 19: result after Thinning at different thresholds 5%, 7% and 9%

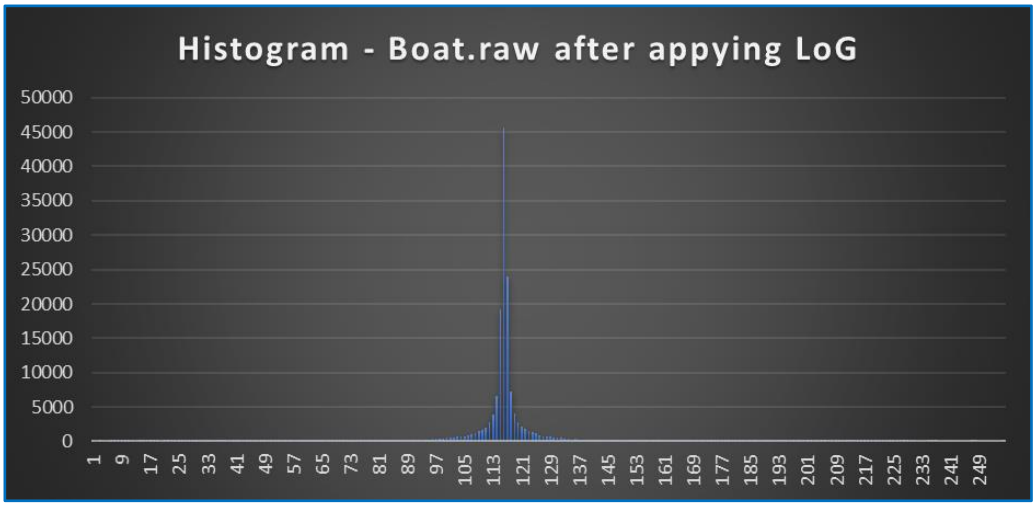


Figure 20: Histogram used for calculation of knee point - For making Ternary Map

Observations – Sobel filter

1) **Threshold plays a major role in detecting images:** There is marked difference between images having lower threshold and higher threshold. Especially in case of noisy images, a lower threshold value results in unnecessary sorts being designated as edges. Histogram was used to calculate the number of pixels between the knee distance. If the number of pixels > threshold (%) * Total pixels, the knee point was marked at that level.
2) **Images post-processed with Morphological Thinning look better overall:** There is significant improvement in images processed using the thinning. This happens because unnecessary think edges are reduced to thin lines.
3) **Denoising using Gaussian filter produces better results for noisy images:** Without using Gaussian, getting the correct output in case of noisy images is kind of impossible.

Observations – LoG filter

1) **Just like Sobel, threshold plays a major role in detecting images:** There is marked difference between images having lower threshold and higher threshold. Especially in case of noisy images, a lower threshold value results in unnecessary sorts being designated as edges.
2) **Ternary maps for different thresholds are different**: The maps provide a rough reasoning about what the result is going to be like. This can be used a good debugging feature as zero-crossing may or may not show edges as good as ternary maps.
3) **Morphological Thinning does not improve the result:** There is no significant improvement in images processed using the thinning. This happens because LoG filter is stable and robust filter. Most of the edges that are detected are single pixel lines where there is little or no scope of thinning
4) Instead of applying Gaussian filter first followed by Sobel or LoG operator. One can convolve the two filters and then use only one filter.

Comparison between Sobel and LoG

1. Sobel produces thick edges but the thinned result of Sobel is visually better than LoG
2. LoG is very sensitive to minute details and can produce very noisy images. Hence it becomes very important to blur the image first and then proceed towards application of LoG
3. LoG's output may look visually worse than Sobel but it more accurate in identifying edge, see the lines along the door of the house. Especially, the windows, Sobel has filled them up while LoG identifies correctly

Part B: Structured Edge

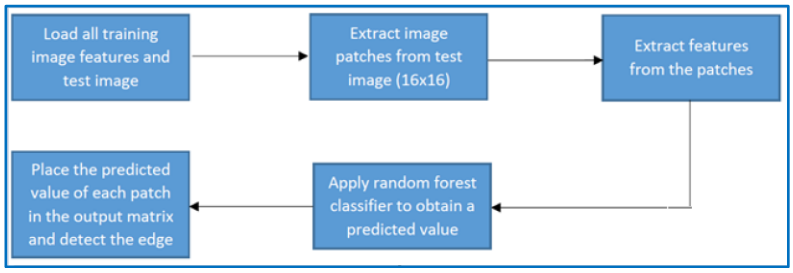
Abstract and Motivation

Contour Detection and Structured Edge are an integral part of image processing and computer vision. They are widely used for detecting fine edges. Structured forest is a state of the art algorithm that is used to compute the edges on a given image. This edge detector considers a patch around the pixel and determines the likelihood of the pixel being an edge point or not. Edge patches are classified into “sketch” tokens using random forest classifiers.

In simple words, Structured edge detector takes the advantage of structure present in the local image patch to learn from the image and perform efficient and accurate edge detector. In learning based edge detection approaches, an image patch is taken and the likelihood that the center pixel contains an edge is computed.

Decision Tree - where a is the sample such that each element/member is classified by recursively branching left or right down the tree until a left node is reached. Generally, decision tree is a flowchart like structure which consists of 3 types of nodes: Decision nodes (Squares), Chance nodes (Circles) and End Nodes (Triangles). At each node j, we must take a binary decision and it is done by a split function. The split function can be complex in certain scenario and hence single feature dimension of is compared to a threshold.

The Structured Edge Detection Algorithm is as under:



Initially the random forest is trained by taking a patch from the input and ground truth images and forming a structured output of whether a certain combination should be classified as an edge or not. After the random forest is trained by an appreciable number of samples, it can be tested. In the testing phase, the test image is given for only 16 segmentations due to the slow speed of random forests. The algorithm it computes the segmented output.

A random forest is a neat approach and very efficient. They run on the divide and conquer logic which helps to improve their performance. The main idea of ensemble approach is to put a set of weak learners to form a strong learner. The basic element of a random forest is a decision tree. The decision trees are weak learners that are put together to form a random forest.

Approach and Procedures

Online available code was used for performing structured edge detection as asked in the question from https://github.com/pdollar/edges. From the list of files, edgesDemo.m file was used to perform the Structured Edge Detection algorithm. The logic used in the MATLAB code is as under:
Step-1: Set the training parameters and the model parameters for an image dataset BSDS500.
Step-2: Obtain the labels for these images.
Step-3: Extract lot of P patches.
Step-4: These patches are the training data.
Step-5: Compute gradient and color to extract features from the patches.
Step-6: Finally train the structured random forest.
Step-7: Then perform the edge detection by extracting patches from the image to be tested on.

Step-8: Then extract features from patches and applying classifiers (structured random forest).
Step-9: Place the predicted value into the output image to get a probability edge map.
Step-10: Set the threshold to the probability edge map to obtain the binary edge map.

Results and Discussion

Results for Structured Edge – House

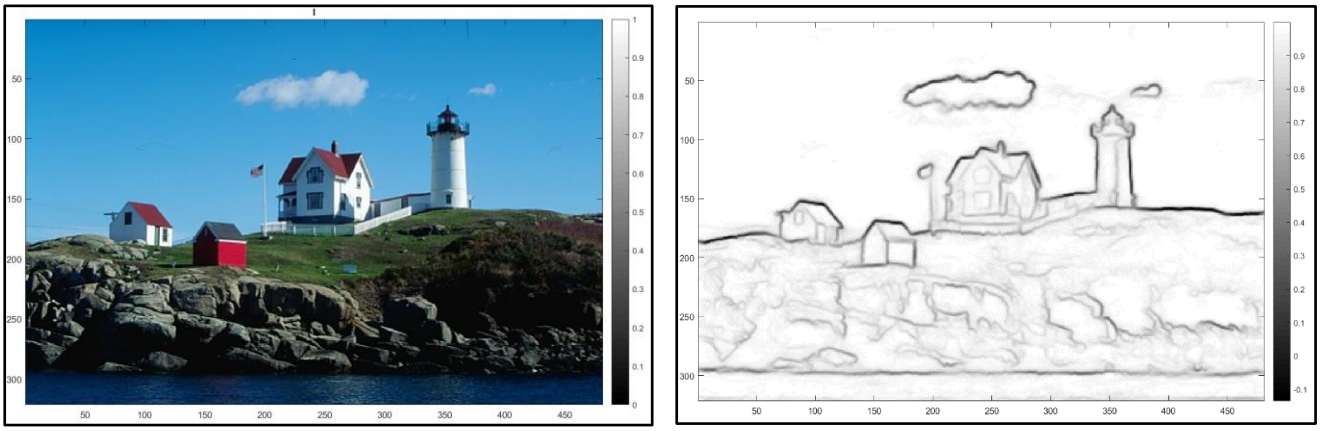


Figure 21: House Image and Probability Map

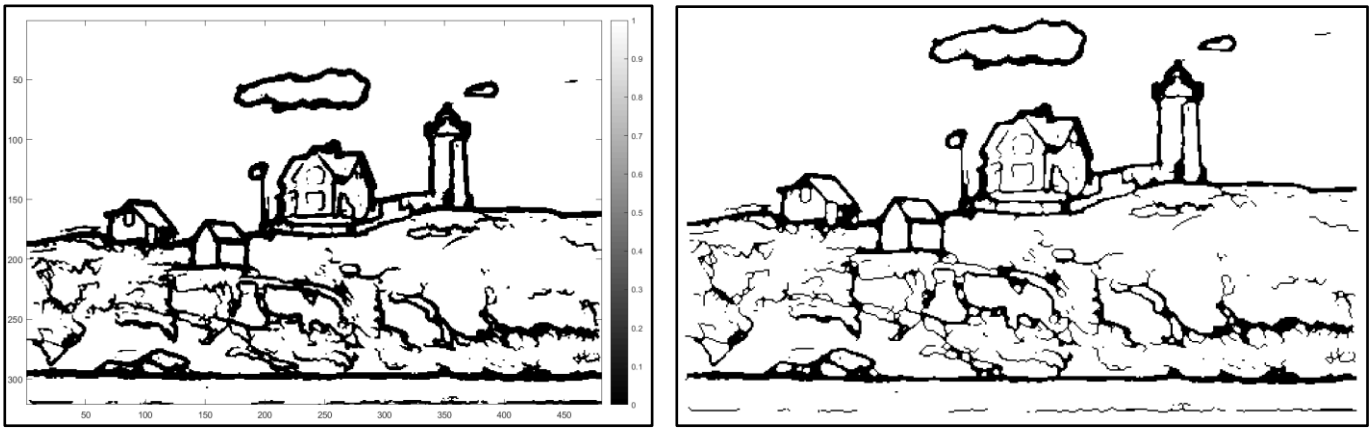


Figure 22: House Image – 1) Structured Edge at 10% threshold 2) Thinned Edges

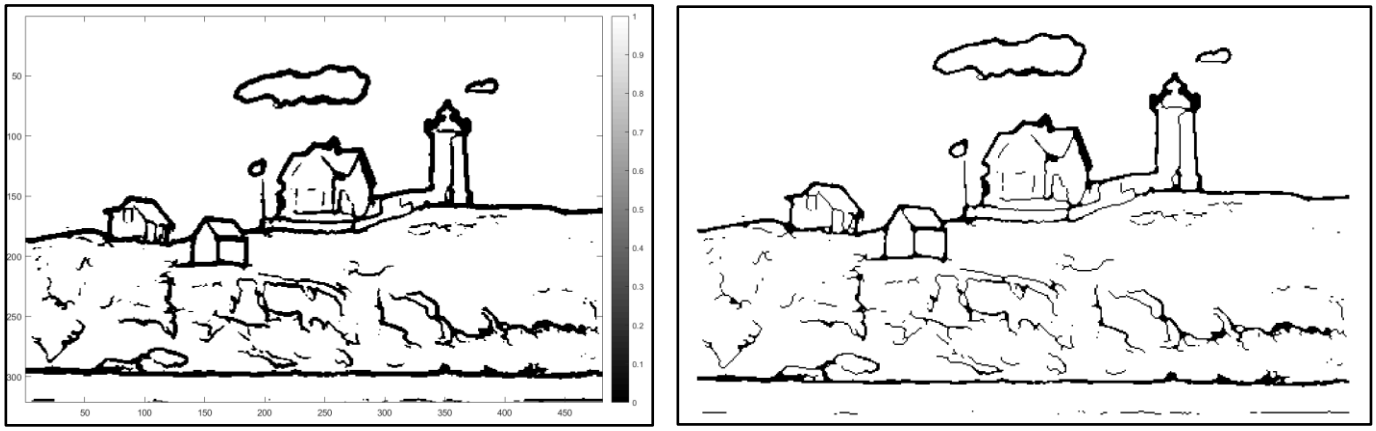


Figure 23: House Image – 1) Structured Edge at 15% threshold 2) Thinned Edges

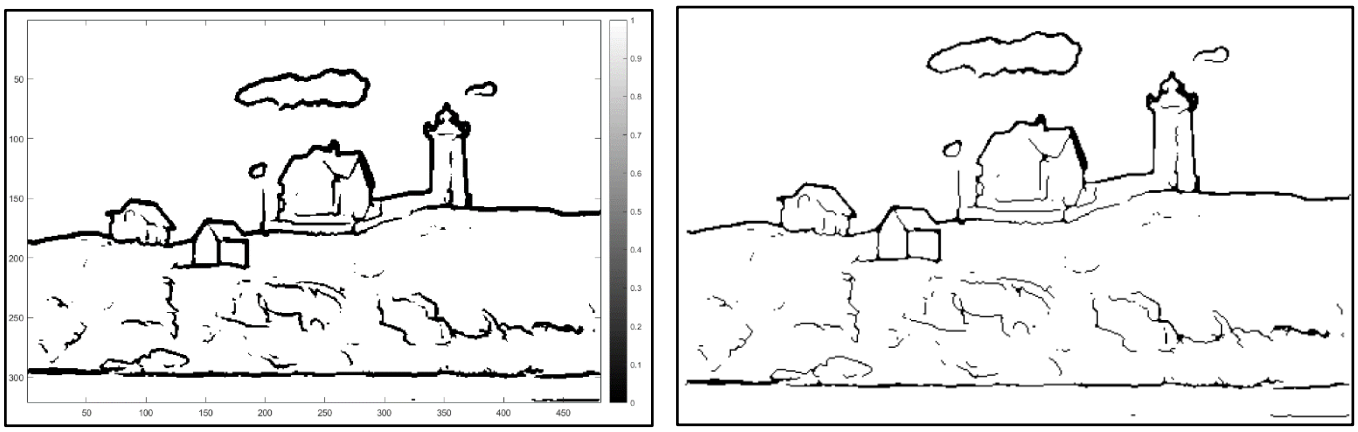


Figure 24: House Image – 1) Structured Edge at 20% threshold 2) Thinned Edges

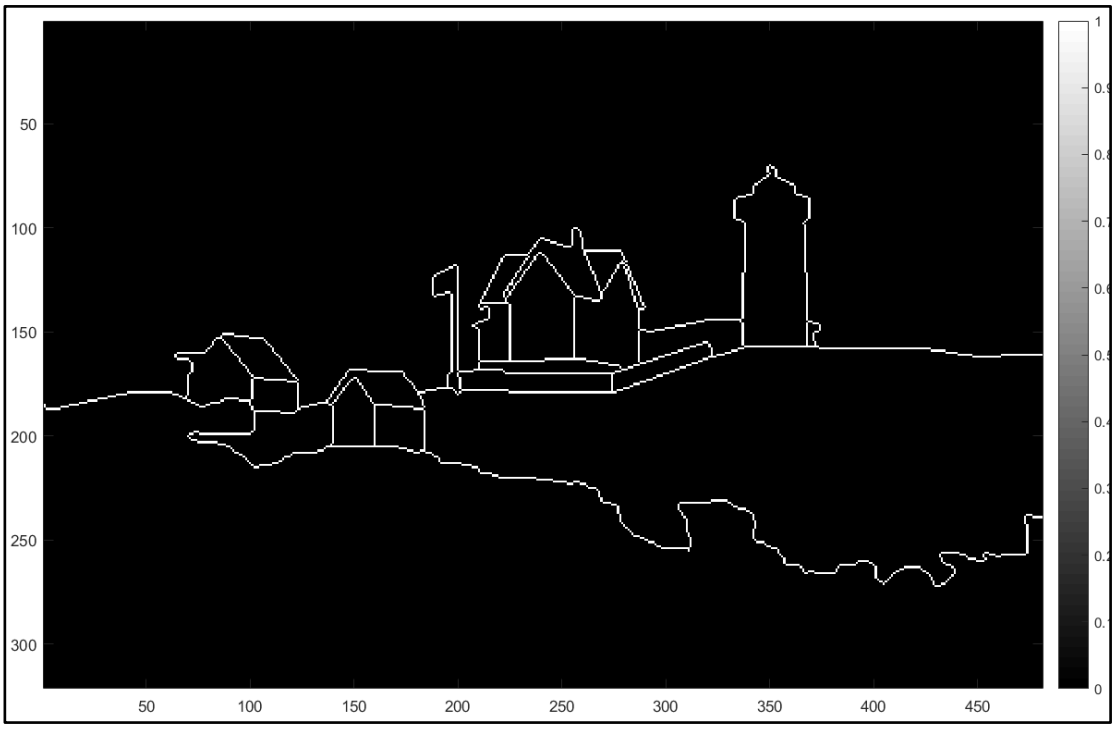


Figure 25: House Image – Ground Truth

Results for Structured Edge – Animal

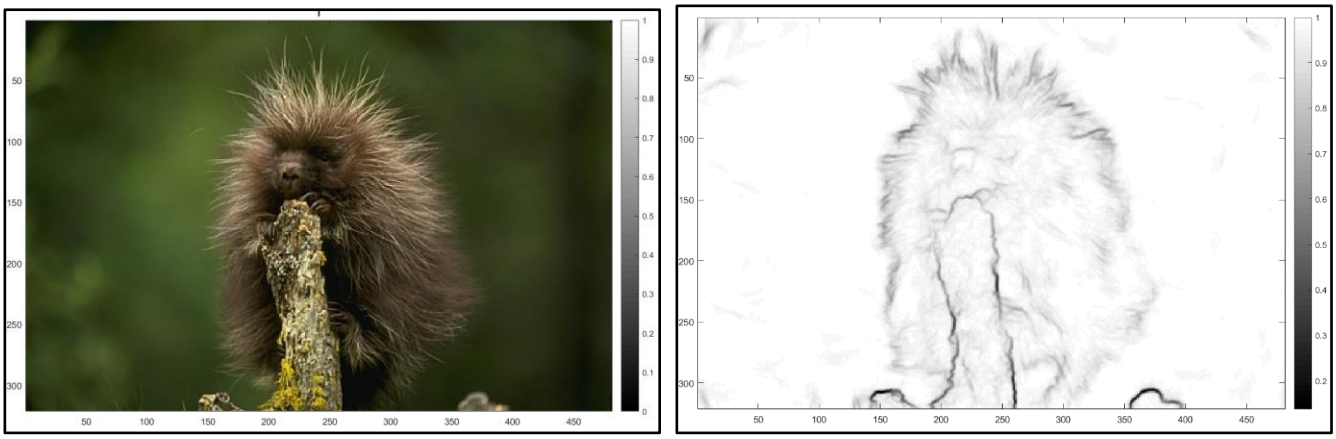


Figure 26: Animal Image and Probability Map

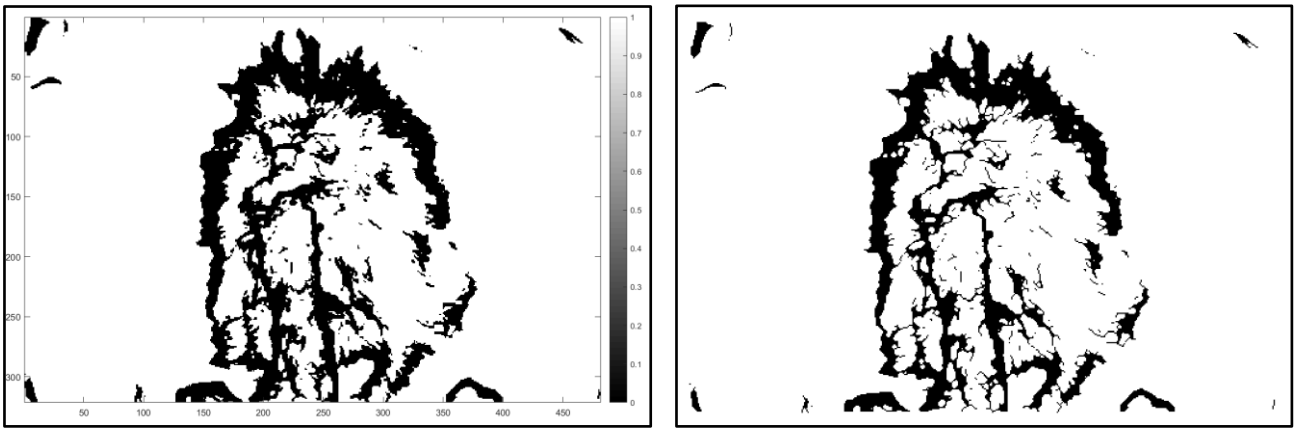


Figure 27: Animal Image – 1) Structured Edge at 5% threshold 2) Thinned Edges

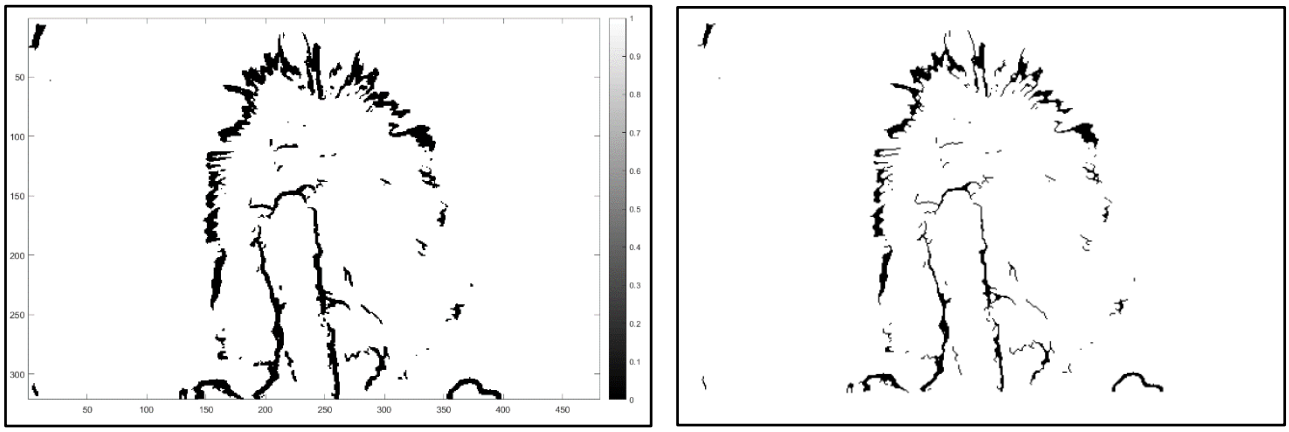


Figure 28: Animal Image – 1) Structured Edge at 12.5% threshold 2) Thinned Edges

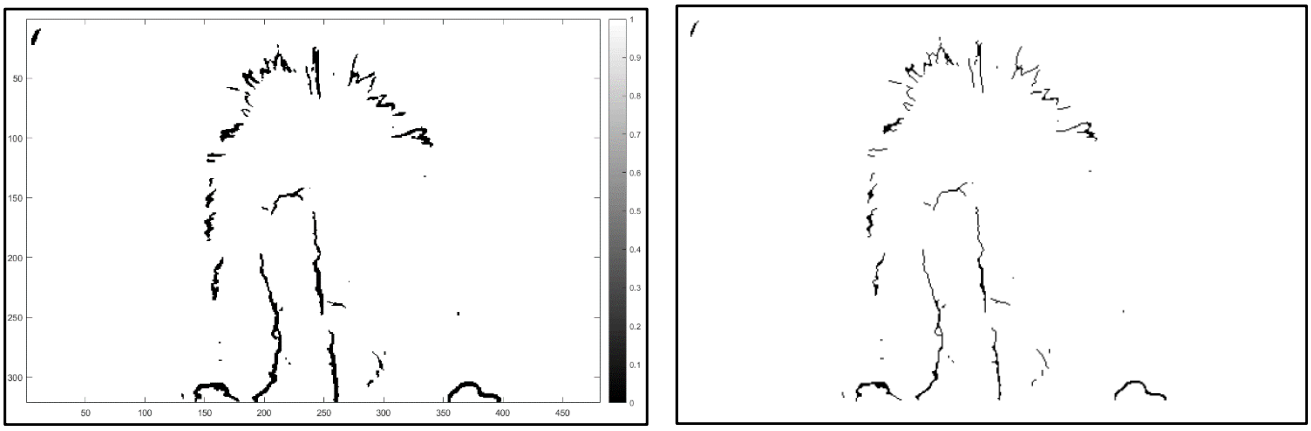


Figure 29: Animal Image – 1) Structured Edge at 20% threshold 2) Thinned Edges

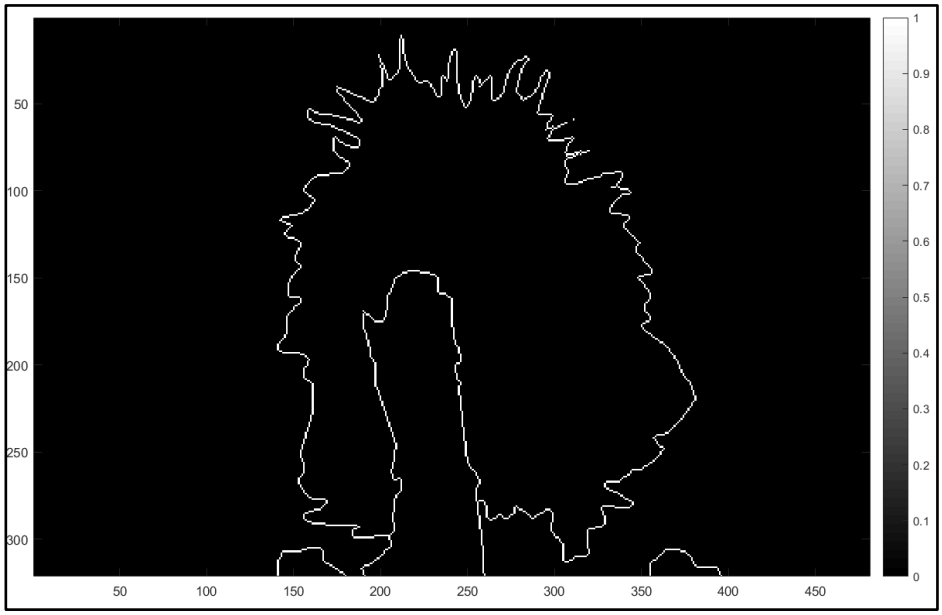


Figure 30: Animal Image – Ground Truth

Observations –

1) Following parameters were observed for running the program
   edgesTrain() at Good Settings
   Model = Random Forest (BSDS) -> Gives fast results
   nPos=5e5 -> Sufficient number for getting enough patches
   nNeg=5e5 -> Sufficient number for getting enough patches
   useParfor=1 -> Because memory was okay on the PC
   Multiscale=0 -> Model better optimized for better results
   model.opts.sharpen=1 -> To avoid errors
   model.opts.nTreesEval=4 -> To avoid errors
   model.opts.nThreads=4 -> For faster results
   model.opts.nms=0
2) Structured gave the best result for House at a threshold of 20%
3) Structured gave the best result for Animal at a threshold of 12.5%

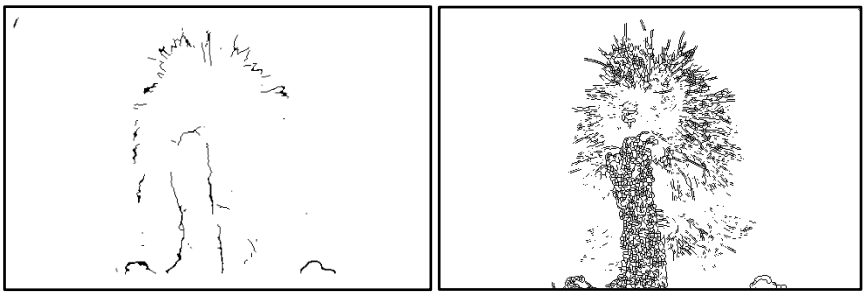


Figure 31: Animal Image – 1) Structured Edge at 12.5% threshold 2) Sobel filter at 15%

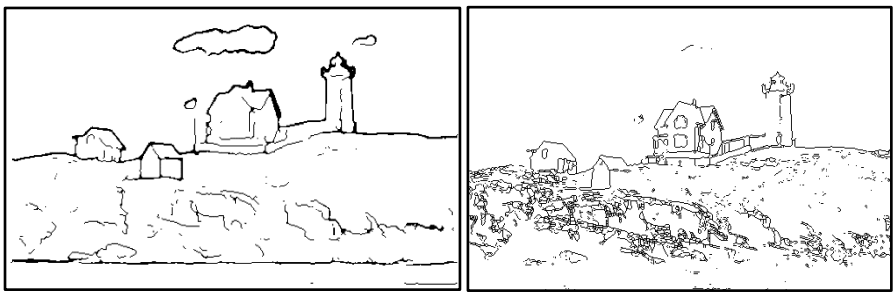


Figure 32: Animal Image – 1) Structured Edge at 20% threshold 2) Sobel filter at 15%

The structed binary edge map was way thicker resulting coverage of less details of the edges as compared to Sobel detector that gave a very fine edge detection which can be seen in the figures above. If we look at the probability edge map on the other hand, for structured edge we can differentiate between objects based on the probability edge map which is not something that is possible through either of Sobel edge detector.

A decision tree classifies an input that belongs to space A as an output that belongs to a space B. The input or output space can be complex in nature. For example, we can have a histogram as the output. In a decision tree, an input is entered at the root node. The input then traverses down the tree in a recursive manner till it reaches the leaf node. Each node in the decision tree has a binary split function with some parameters. This split function decides whether the test sample should progress towards the right child node or the left child node. Often, the split function is complex.

A set of such trees are trained independently to find the parameters in the split function that result in a good split of data. Splitting parameters are chosen to maximize the information gain.

Part C: Performance Evaluation

Approach and Procedures

For evaluating the performance of different edge detectors, we use the parameter called F measure, defined as:

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

where P and R are Precision and Recall respectively. Precision and Recall are calculated as shown below:

$$P = \frac{\#TruePositive}{\#TruePositive + \#FalsePositive}$$
$$R = \frac{\#TruePositive}{\#TruePositive + \#FalseNegative}$$

Now let us define the terminologies that we have used in the formulae above.
#TruePositive are the number of edge pixels in the edge map that coincide with the edge pixels in the ground truth image. These are the edge pixels that SE detector predicts successfully.
#TrueNegative are the number of non-edge pixels in the output edge map that coincide with non-edge pixels in the ground truth image.
#FalsePositive are the number of edge pixels that our algorithms detect but are not the edge pixels in the ground truth image.
#FalseNegative are the number of non-edge pixels in the edge map that our algorithm predicts but are actually edge pixels in the ground truth image.
Groundtruth images are the edge maps provided by the humans. Different people may give different edge maps, thus it is necessary to take the mean of F measure with different edge maps to get the final F measure.

We are provided with 5 ground-truth edge maps for each of the Boat and House inputs. Higher the F measure better is the detector.

From the above two figures we see that the Sobel edge output largely depends on the above-mentioned parameters. Also, we see that the two input images respond differently to the same threshold values. For both the images, a better result is obtained when the Gaussian Filter size is 5 and Sobel kernel size is 3. But the edge map of House.raw is better when the threshold is 20% and the edge map of Animal.raw is better when the threshold is 12.5%. Therefore, we conclude that different images respond differently to the same threshold.

As far as the performance between the two images is concerned, House image gives better edge map than the Animal image. This is because the Animal image has a lot of textures where we get lot of unnecessary details and smooth edges where intensity gradient is nearly equal on both sides of the edge and we lose that edge. This happens because the edges that are lost after smoothing the image to reduce the extra texture in the image from appearing in the edge map. Also, we can see how these marked edges blend into the sky on smoothing and the intensity gradient misses it to detect as an edge. We cannot find the object's location from the edge map because the edge map is not connected and there are other textures as well in the edge map. Also, Sobel is based on point pixel property of the image and thus we cannot locate the objects in the image.

F-measure is a critical tangible qualitative parameter to distinguish and evaluate the performance of the edge detector. F measure is the harmonic mean of Precision (P) and the Recall(R). The square of geometric mean of P and R is divided arithmetic mean of the two. So, F-measure is nothing the Harmonic mean of P and R. Alone the Precision or the Recall cannot not give us the proper parameter to evaluate the performance of the edge detector. Let us visualize F mathematically:

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

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

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

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

It is not possible to get a higher F measure if the precision is higher than recall or vice-versa. When the sum of P and R is constant, when P increases, R decreases which implies when #FP increases #FN decreases. and in the above equation we see that F does not changes when one of the P or R parameter increases or decreases. And when P = R,

$$\#FN = \#FR = 0$$

Hence,

$$F = 2 * \frac{\#TP}{(2 * \#TP)}$$

$$F = 1$$

Results and Discussion

T	P	R	F
0.2	0.46	0.93	0.61
0.2	0.59	0.74	0.65
0.2	0.52	0.75	0.62
0.2	0.52	0.76	0.62
0.2	0.49	0.75	0.59

T	P	R	F
0.1	0.52	0.65	0.58
0.1	0.58	0.63	0.61
0.1	0.29	0.44	0.35
0.1	0.63	0.72	0.67
0.1	0.49	0.58	0.53

T	P	R	F
0.35	0.8	0.73	0.76
0.35	0.94	0.52	0.67
0.35	0.87	0.56	0.68
0.35	0.73	0.48	0.58
0.35	0.66	0.45	0.53

Figure 33: Structured Edge - Animal image at Thresholds: a) 0.1 b) 0.2 and c) 0.35

T	P	R	F
0.1	0.23	0.92	0.37
0.1	0.34	0.83	0.48
0.1	0.3	0.85	0.45
0.1	0.29	0.82	0.43
0.1	0.29	0.86	0.43

T	P	R	F
0.2	0.46	0.93	0.61
0.2	0.59	0.74	0.65
0.2	0.52	0.75	0.62
0.2	0.52	0.76	0.62
0.2	0.49	0.75	0.59

T	P	R	F
0.35	0.74	0.79	0.76
0.35	0.87	0.57	0.69
0.35	0.81	0.61	0.69
0.35	0.69	0.53	0.6
0.35	0.64	0.51	0.57

Figure 34: Structured Edge - For House image at Thresholds: a) 0.1 b) 0.2 and c) 0.35

T	P	R	F
0.15	0.12	0.66	0.2
0.15	0.13	0.64	0.22
0.15	0.07	0.45	0.12
0.15	0.15	0.77	0.25
0.15	0.12	0.62	0.2

T	P	R	F
0.2	0.14	0.55	0.22
0.2	0.16	0.53	0.24
0.2	0.08	0.4	0.14
0.2	0.18	0.63	0.27
0.2	0.14	0.52	0.22

T	P	R	F
0.35	0.15	0.45	0.23
0.35	0.18	0.44	0.25
0.35	0.11	0.37	0.17
0.35	0.2	0.51	0.28
0.35	0.16	0.43	0.23

Figure 35: Sobel - For Animal image at Thresholds: a) 0.15 b) 0.20 and c) 0.25

T	P	R	F
0.15	0.1	0.8	0.18
0.15	0.16	0.79	0.27
0.15	0.15	0.82	0.25
0.15	0.15	0.86	0.26
0.15	0.17	0.98	0.28

T	P	R	F
0.2	0.12	0.71	0.2
0.2	0.19	0.7	0.29
0.2	0.16	0.71	0.27
0.2	0.18	0.78	0.29
0.2	0.2	0.92	0.33

T	P	R	F
0.35	0.13	0.65	0.22
0.35	0.2	0.62	0.31
0.35	0.17	0.62	0.27
0.35	0.2	0.73	0.32
0.35	0.24	0.9	0.38

Figure 36: Sobel - House image at Thresholds: a) 0.1 b) 0.2 and c) 0.35

T	P	R	F
0.05	0.145	0.768	0.244
0.05	0.156	0.71	0.256
0.05	0.085	0.547	0.147
0.05	0.18	0.855	0.297
0.05	0.142	0.701	0.236

T	P	R	F
0.1	0.117	0.914	0.208
0.1	0.131	0.878	0.229
0.1	0.072	0.68	0.13
0.1	0.138	0.961	0.241
0.1	0.118	0.855	0.207

T	P	R	F
0.15	0.097	0.964	0.176
0.15	0.108	0.922	0.193
0.15	0.063	0.76	0.116
0.15	0.11	0.984	0.198
0.15	0.099	0.919	0.179

Figure 37: LoG - Animal image at Thresholds: a) 0.05 b) 0.1 and c) 0.15

T	P	R	F
0.05	0.13	0.66	0.22
0.05	0.2	0.61	0.3
0.05	0.17	0.62	0.27
0.05	0.21	0.75	0.32
0.05	0.24	0.91	0.38

T	P	R	F
0.1	0.1	0.75	0.17
0.1	0.16	0.78	0.27
0.1	0.14	0.79	0.24
0.1	0.15	0.84	0.26
0.1	0.17	0.99	0.29

T	P	R	F
0.15	0.08	0.76	0.15
0.15	0.15	0.82	0.25
0.15	0.13	0.82	0.22
0.15	0.13	0.85	0.23
0.15	0.15	0.99	0.25

Figure 38: LoG - For House image at Thresholds: a) 0.1 b) 0.2 and c) 0.35

Observations:

1. Structured edge has way higher F-score than both Sobel and LoG
2. Highest F-score for House and Animal images are 76% and 76%, respectively - SE
3. Highest F-score for House and Animal images are 38% and 25%, respectively - Sobel
4. Highest F-score for House and Animal images are 29.7% and 38%, respectively – LoG
5. SE out performs Sobel and LoG and Sobel outperforms LoG for the given images
6. F-measure is image dependent. Finding edges for Animal is harder than house for any of the filters, SE, Sobel and LoG. The reason for that could be, it is easier to find edges in some images. In house, the edges can be found very easily, even a human will draw out the same edges as the one found out by SE. But in case of Animal, there are lot of furs, and nobody will practically care about the furs, because there are too many edges.

Conclusion

For the given images, Sobel outperforms LoG at places in terms of visual quality but LoG is better than Sobel in terms of correctness. The structed binary edge map was way thicker resulting coverage of less details of the edges as compared to Sobel edge detector that gave a very fine edge detection which can be seen in the figures above. If we look at the probability edge map on the other hand, for structured edge we can differentiate between objects based on the probability edge map which is not something that is possible through a Sobel edge detector output.

Problem 3: Salient Point Descriptors and Image Matching

Part A: Extraction and Description of Salient Points
Part B: Image Matching

Abstract and Motivation

Image matching is a fundamental aspect of many problems in computer vision, including object or scene recognition, solving for 3D structure from multiple images, stereo correspondence, and motion tracking. This section describes image features that have many properties that make them suitable for matching differing images of an object or scene. The features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint to some extent.

Salient point extraction is very much required in image processing and computer vision tasks specifically such as object recognition, video tracking, navigation, gesture recognition and image stitching, we need a means to extract meaningful features that can describe the image. These kinds of feature extraction requirements are where SIFT and SURF come into the picture. These extracted features are required to be scale invariant and rotation invariant to provide robustness. SIFT and SURF are such algorithms that extract important local features in an image that are scale and rotation invariant.

Task: In this problem, we implement SIFT and SURF algorithms to compare their performance and efficiency. We also utilize these features to perform object matching using Brute Force matching technique. It is not the best kind of matching that can be expected and its output is not the best but the detailed discussion on the same and its pros and cons are discussed below.
Object categorization and localization is sometime required to enhance an image. The biggest challenges in object classification are introduced due to problems such as different camera positions, illumination differences, internal parameters and variation within the object which has not been solved till date and is a big area of research that is still going on.

Approach and Procedures

SIFT is an abbreviation of Scale Invariant Feature Transform and SURF is the abbreviation of Speeded-Up Robust Features.

a) Extraction and Description of Salient Points

The extraction of features using SIFT algorithm is as described below:

- Constructing a scale space: This is the initial preparation. You create internal representations of the original image to ensure scale invariance. This is done by generating a "scale space"
- LoG Approximation: The Laplacian of Gaussian is great for finding interesting points (or key-points) in an image. But it's computationally expensive. So, we cheat and approximate it using the representation created earlier

- Finding Key-points: With the super-fast approximation, we now try to find key points. These are maxima and minima in the Difference of Gaussian image we calculate in the above step
- Get rid of bad Key-points: Edges and low contrast regions are bad key-points. Eliminating these makes the algorithm efficient and robust. A technique similar to the Harris Corner Detector is used here
- Assigning an orientation to the key-points: Orientation is calculated for each key-point. Any further calculations are done relative to this orientation. This effectively cancels out the effect of orientation, making it rotation invariant
- Generate SIFT features: Finally, with scale and rotation invariance in place, one more representation is generated. This helps uniquely identify features. Let's say you can have 50,000 features. With this representation, you can easily identify the features you're looking for

SIFT uses approximated Laplacian of Gaussian with Difference of Gaussian for finding scale-space maxima. SURF approximates LoG with Box Filter. First integral images are calculated then they are convolved with the box filter, which is fast and can be performed in parallel on images with different scales. The scale and location is determined by using Hessian Matrix. It is a square matrix of second order partial derivatives of a scalar valued function/field [13]. The size of the descriptor vector is 64-dimensional vector, which can be extended to 128-D as in SIFT. We can skip the calculation of orientation if we know the two images are upright, which adds to speed up the computation.

In SIFT algorithm, we first extract the features from the image and then compute the descriptor on individual key-point. The dimensions of the descriptor matrix are then Nx128, where N is the number of key-points extracted. Then we plot these points on the image and display the output. We do the similar process with the SURF algorithm, except that we use Hessian threshold for limiting the number of features.

b) Image Matching

We use descriptors for image matching. The image matching is done as described below:

Step 1: The input to image matching algorithm are the two images that are to be matched. These images are converted to grayscale as SIFT and SURF take grayscale inputs.

Step 2: First the SIFT or SURF key-points of both the images are calculated individually. And their corresponding descriptor matrices (Nx128 and Nx64) are generated. The descriptors hold the actual features at each key-point in the respective images.

Step 3: Now we must compare the descriptors of the two input images. There are many different algorithms for descriptor matching like KNN matcher, radius matcher, Brute-Force matcher and Flann Matcher. We have used Flann matching algorithm since it trains on the passed descriptor and class its nearest search methods to find the best matches.

Step 4: We now sort the matched descriptors in the ascending order of their distances. And select the top 30 key-points for display purpose. When we take all the key-points the display image will not be of much use for visual interpretation. Further on this will be discussed in the Discussion Section.

Step 5: We now draw lines between the matched features. For classification problem, we set a threshold on the number of matched key-points to classify an image into certain category or to match the image to some other set of images.

Results and Discussion

Extraction and Description of Salient Points

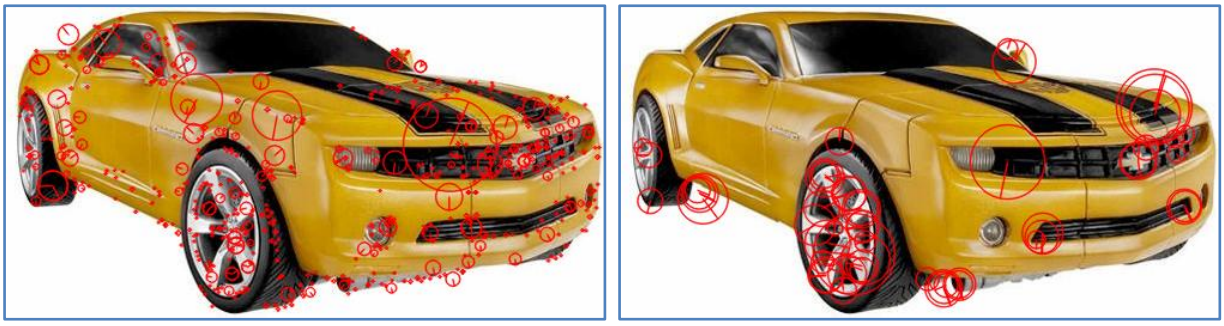


Figure 1: Bumblebee a) SIFT Points a) SURF Points

Figure 2: Ferrari 1 a) SIFT Points a) SURF Points

Figure 3: Ferrari 2 a) SIFT Points a) SURF Points

Figure 4: Optimus Prime a) SIFT Points a) SURF Points

Observations

- It can be observed that more number of important points or interesting points are recognized in SIFT than SURF. This does not necessarily mean that SIFT is better than SURF. The size of the circle shows the most important in the region, where the size region is conveyed by the radius of the circle around the point
- The orientation of the line inside the circle shows the orientation of that region. Orientation is assigned to each key-point to achieve invariance to image rotation. A neighborhood is taken around the key-point location depending on the scale, and the gradient magnitude and direction is calculated in that region.

Image Matching

Figure 5: Bumblebee a) SIFT Points a) SURF Points

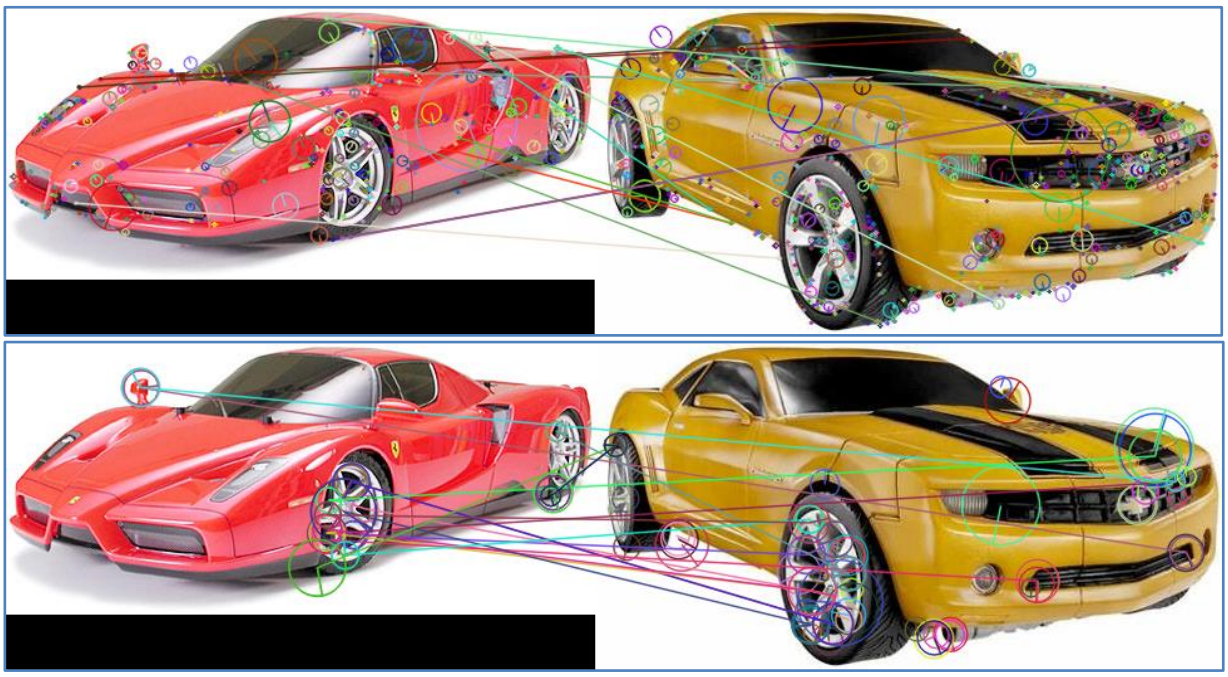


Figure 6: Ferrari 1 and Bumble Bee Car a) SIFT Points a) SURF Points

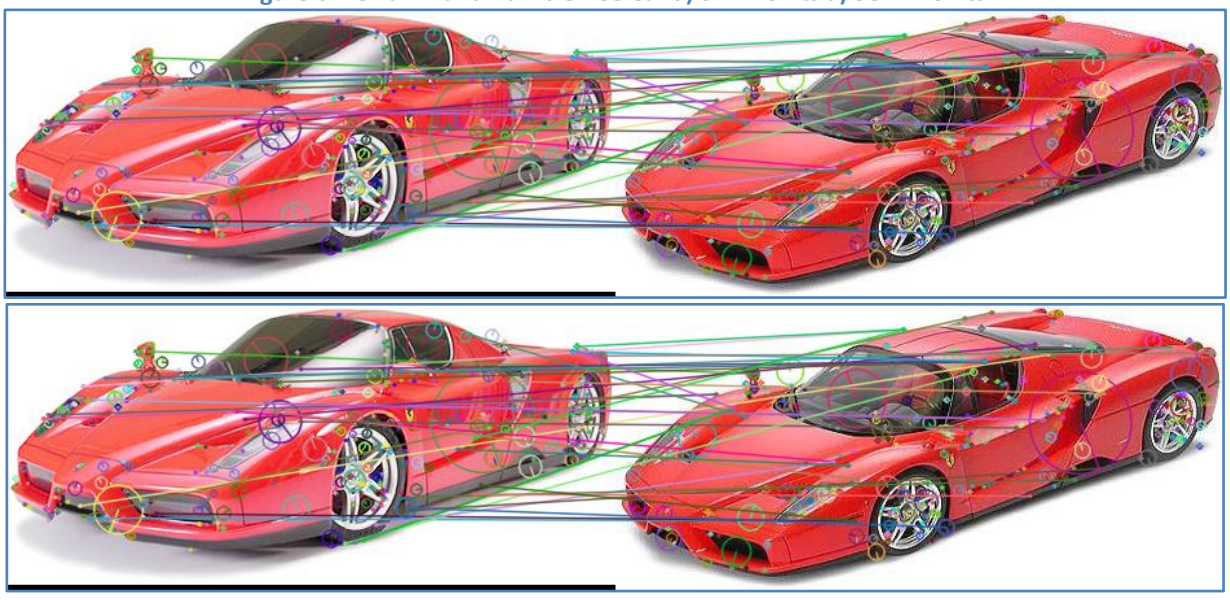
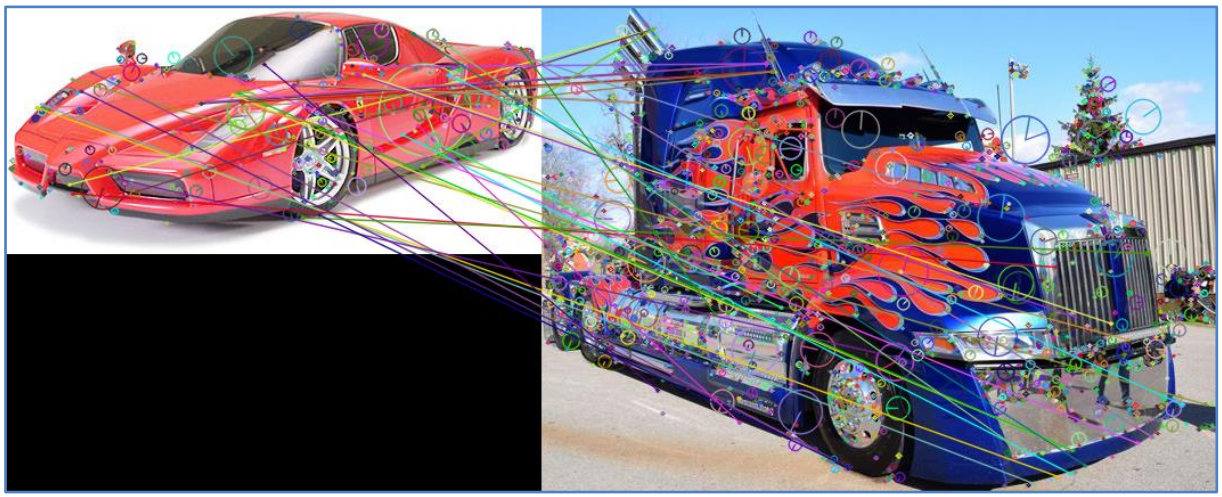


Figure 7: Ferrari 2 a) SIFT Points a) SURF Points



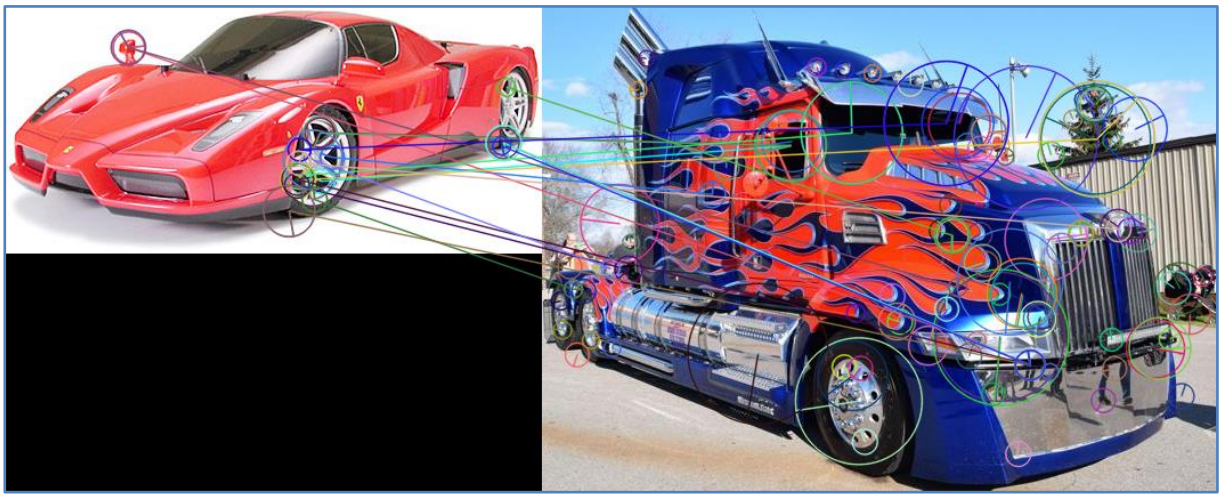


Figure 8: Ferrari 2 a) SIFT Points a) SURF Points

Figure 9: An example showing very large number of Points being detected

Observations

- There are clearly some good matches and some bad matches. **Not every point matched is a correct matching**. For example, wheel portion of Ferrari 1 matches to the bonnet of Bumblebee – bad matching
- **Angle of Cars in the image matters**, Ferrari 1 and Ferrari 2 have more good matches than other pairs

- The orientation of the line inside the circle shows the orientation of that region. Orientation is assigned to each key-point to achieve invariance to image rotation. A neighborhood is taken around the key-point location depending on the scale, and the gradient magnitude and direction is calculated in that region
- **Keeping a bad threshold can generate a lot of points are produced which are not required**. A perfect example of this is the matching between Ferrari 1 and Optimus Prime. This just shows that a lot of applications are possible which can use this data, so we just must know what kind of data is required for that application and accordingly, what can be the appropriate threshold

Part C: Bag of Words

Bag-Of-Words (BoW) or bag of visual words is one of the most popular approaches in the field of image classification. Here we represent an object as a bag of visual words.

Bag of Words algorithm is based on the idea of document classification and OCR use it on a broader scale. It can be used for image classification in small systems wherein user defines the number of clusters. Bag of words is the vector of occurrence of count of words and is expressed as a histogram of each word. Training of Bag of Words is done to cluster the training images into several clusters. The calculated features of test images can be compared to words in a dictionary to get histogram.

These visual words are basically the features of the images. To use Bag of Words for image classification, we need to extract features from images, create a codebook and then generate a histogram. We extract features by using SIFT algorithm. These are 128D feature vectors. There will be many such feature vectors in an image, where the order of each feature is not important. Thus, SIFT features stand a good chance to be the optimum feature extractor in BoW concept. A codebook is basically a dictionary. A codebook consists of codewords. Once the SIFT features are extracted, we use K-means clustering to build our codebook. The codewords here are the centroids of these K clusters. The flowchart below explains the image classification using bag of words.

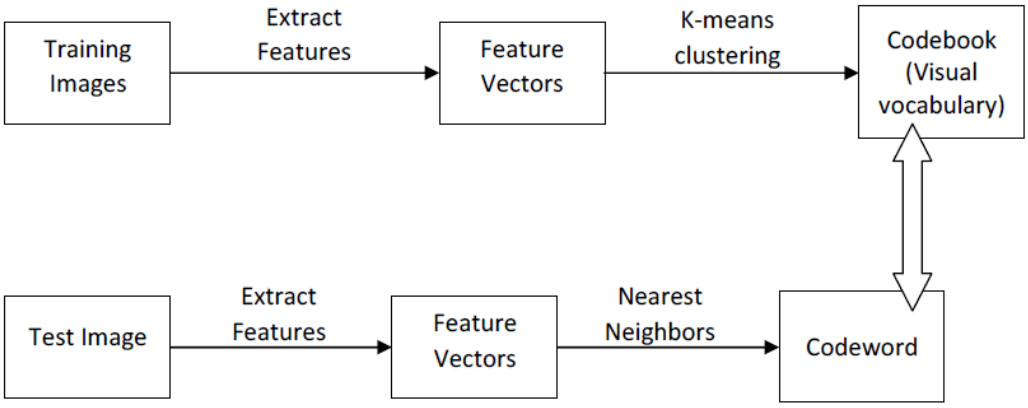


Figure 10: Bag-of-Words Flow Chart

Algorithm for OpenCV for Image Matching:

- We first, extract the local features of the image using SIFT
- Placement of all the features into a single set

- K- means clustering algorithm is applied over the set of the feature vectors we obtained in above step to find centroid co-ordinates and we assign a code word to each centroid obtained. This results in the construction of our vocabulary of length K

Finally, once we obtain feature vectors based on global identities, we classify them. Based on the size of training data we used K- nearest algorithm for smaller training data set.

Results and Discussion

Figure 11: Histograms Plots for 8 bins

Observations

1. For each image a 128x8 feature vector
2. A 369x8 vector is created which is populated by each of the images – This is the BoW
3. The feature vectors for Ferrari 2 (the test image) is matched against the BoW created in 2
4. Histogram plots of Ferrari 1 and Ferrari 2 are almost the same – Hence we know there is a match

Conclusion

OpenCV is a very important tool to learn. The author can appreciate how the building of this platform would have taken place. Finding salient points using SIFT and SURF are techniques which will come handy on a lot of places. Like stitching photos/videos to generate 360° content for Virtual Reality. OpenCV has many applications in Computer Vison and it has many inherent features and tools which can speed up the prototyping process in developing a Computer Vison based software.

Resources

1. Zero Crossing Detector: https://homepages.inf.ed.ac.uk/rbf/HIPR2/zeros.htm
2. Laplacian/Laplacian of Gaussian: https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm
3. PDollar Toolbox for Structured Edge detection: https://github.com/pdollar/toolbox
4. Texture classification, T Ojala: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OJALA1/texclas.htm
5. Supervised Texture Segmentation: A Comparative Study: https://arxiv.org/ftp/arxiv/papers/1601/1601.00212.pdf
6. Code Reference:
   a) https://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT-Features-with-O
   b) https://github.com/bikz05/bag-of-words