Homework 3 Report

Name: Akshata Bhat
USCID: 9560895350
Mail: akshatab@usc.edu

INDEX

PROBLEM 1: TEXTURE ANALYSIS AND SEGMENTATION

ABSTRACT AND MOTIVATION

The field of texture analysis has four main problems or applications to solve - texture classification, texture segmentation, texture synthesis and shape from texture. A good and efficient texture classification task involves a good description of the input image. Some of the applications of texture classification are inspection of surfaces in the biomedical industry and mechanical industry for defects and disease, satellite imagery, content classification is huge databases. Despite vast applications, there are very few successful applications due to the non-uniform nature of the texture images, varied orientation and scaling in real life applications. In order to analyze the contents of various parts of an image, texture segmentation is used.

APPROACH AND PROCEDURE:

1 A. TEXTURE CLASSIFICATION

Texture classification involves two different stages: learning and recognition phases

a) Learning phase:

In the learning phase the aim is to build a model containing the texture content of every texture class which is present in the training data. These textures are generally just annotated images. A suitable texture analysis algorithm is chosen and the texture content is extracted from the images either in the form of vectors or histograms which convey information about the image contents such as brightness or contrast or contours in the image.

b) Recognition phase:

In this phase the texture information of the image to be classified is extracted by using the texture analysis method. These texture features are then compared with the textures of the images comprising the training dataset and are assigned labels according to the class to which the closest match feature belongs to. In case the closest match or the error rate is very high, then the target image which was supposed to be classified is rejected as not belonging to any of the classes in the available training set.

TEXTURE SEGMENTATION

In texture segmentation which is a harder problem when compared to texture classification, the various textures in the images are identified and then partitioned based on the boundary locations in the image. This method is similar to that of texture classification that is, it involves extracting the features in the image and then identifying that as a texture and assign a unique color or intensity to each unique partition.

Texture segmentation can be either a supervised or unsupervised process depending on whether we have any prior knowledge about the textures in the image. In supervised texture

segmentation, we identify the parts of the image, which are similar to the textures in the available training set and separate the textures, if there are matches.

In the process of unsupervised segmentation, the textures in the image are located by using algorithms which give the change in the edges or contents and then isolating the different textures for future processing. In real world applications, unsupervised segmentation has been found to be more useful but it has huge number of computations involved to get the final segmented outputs.

Procedure for Image Classification:

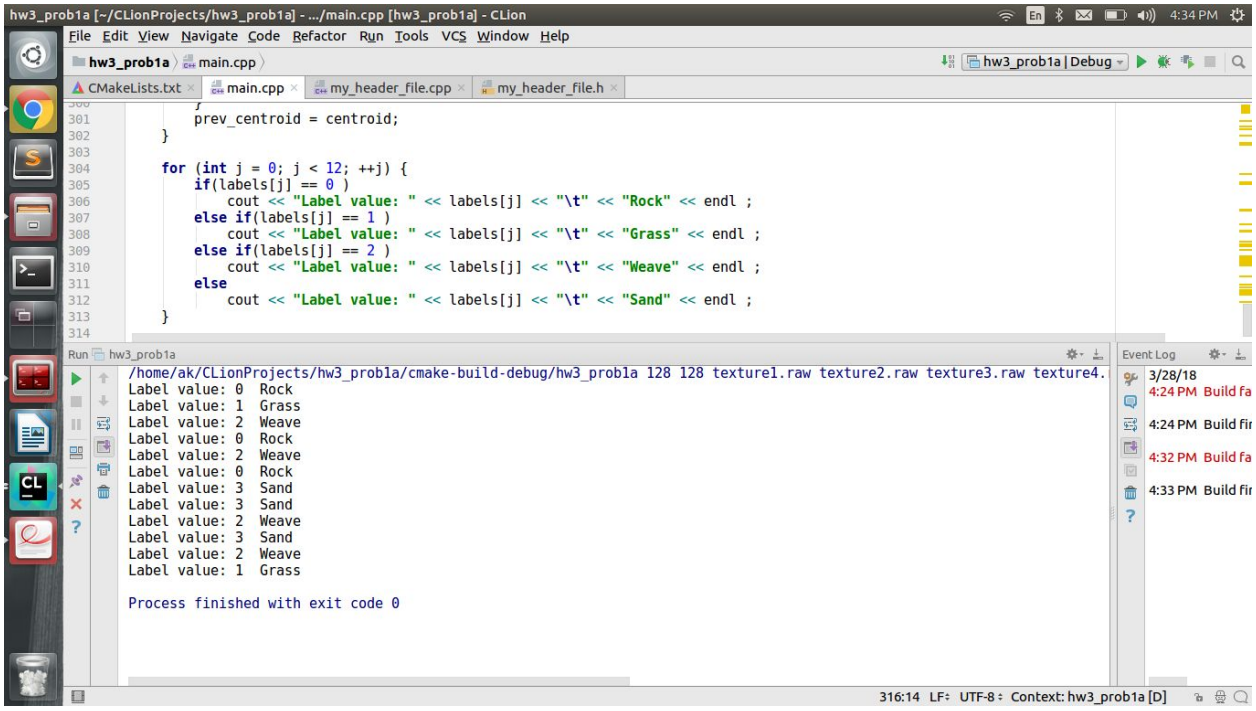
1) Read the 12 input files into 1D arrays by passing the corresponding width and height as parameters
2) Normalize each of the 12 texture images,by subtracting the mean of the image from each pixel intensity value to generate the normalized image
2) Save the filters L5, S5 and W5 as 1D arrays so that the tensor product of each Laws filter can be taken with another to generate the 5x5 masks.
3) These 5x5 masks are then applied to the normalized images to generate the feature vectors from the images for each pixel by performing the pixel replication when the corner pixels were the ones being processed.
4) The normalized feature vectors are then passed to function which generates the 9 dimensional feature vectors corresponding to the Energy of each image after applying the Laws filters.
5) The function which generates the 9D vectors has the following steps:
    a) Perform feature extraction for each input texture image, but applying the 9 Laws filters.
    b) The energy of each filtered image corresponding a certain filter is computed to generate one dimension of the feature vector corresponding to that texture image.
    c) The above step is repeated for every Laws filter and every texture image
6) At the end of the above steps we get twelve 9D feature vectors using which we decide which texture image belongs to which class
7) Initialize any of the 4 feature vectors from the total 12 feature vectors, corresponding to the 4 classes into which the texture images are to classified as the initial centroids
8) Take one 9D vector at a time, find the Euclidean distance between each feature vector and centroid to assign labels to every feature vector based on which class the centroid it is closest to.
9) Once all the 9D feature vectors have been assigned one among the 4 labels, recompute all the centroids
10) The centroids are calculated by taking an average of all the feature vectors which have been assigned to that class.
11) Repeat the steps 8) , 9) and 10) until – the centroids of each class are no longer changing this means that the k-means algorithm has converged and the texture images have been assigned to the correct classes and have been given the correct labels.

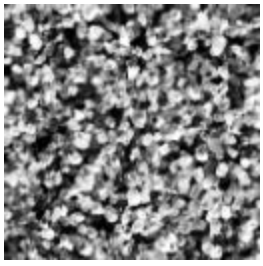
Procedure for Image Segmentation:

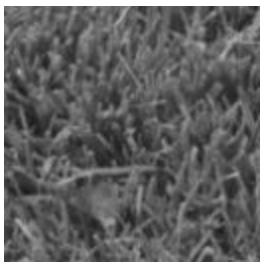
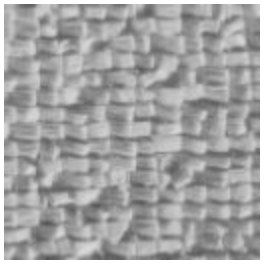
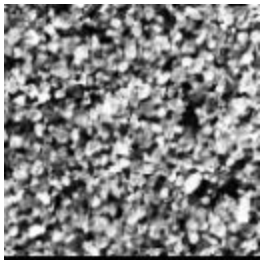
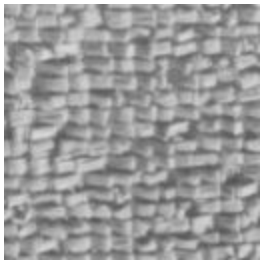
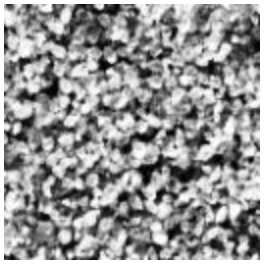
1) Read the input file to be segmented into a 1D array by passing the corresponding width and height as parameters
2) Normalize the composite texture image,by taking a window of N size and subtracting the mean of the pixels in that window from the corresponding center pixel in that window. Perform boundary extensions by replicating the pixels for the corners.
3) Repeat the above step for every pixel in the input image to get the normalized image for further processing
4) Save the filters E3, L3 and S3 as 1D arrays so that the tensor product of each Laws filter can be taken with another to generate the 3x3 filter masks
5) Pass the 9 Laws filters as argument to the function used for generating 9D feature vectors for each pixel in the input image
6) In the function to generate 9D feature vectors perform the following steps:
    a) Take one of the Laws filters at a time, apply it on the input image by taking 3x3 window of image pixels and taking a product of the normalized pixel intensities with the filter values
    b) At the end of the above step you will have 2D feature image for that filter
    c) Compute the energy of the 2D feature image obtained in the above step by taking a suitable window size and computing the energy for the center pixel in that window.
    d) Repeat the above step for the entire 2D filtered image to get the 2D energy image which actually one of the dimensions of the feature vector
    e) Store the above 2D Image as column vector in a bigger 2D array of Width*height number of rows and 9 columns
7) Repeat the step 6) for all the 9 Laws filters to generate the training dataset for performing k-means clustering.
9) Initialize any of the 6 feature vectors from the regions belonging to the 6 regions from the image as the initial centroids for the 6 classes
10) Take one 9D vector at a time from the width*height number of vectors, find the Euclidean distance between each feature vector and centroid to assign labels to every feature vector based on which class the centroid it is closest to.
11) Once all the 9D feature vectors have been assigned one among the 6 labels, recompute all the centroids
12) The centroids are calculated by taking an average of all the feature vectors which have been assigned to that class, considering one feature at a time.
13) Repeat the steps 8) , 9) and 10) until – the centroids of each class are no longer changing this means that the k-means algorithm has converged and the texture images have been assigned to the correct classes and have been given the correct labels.
14) Once all the textures are assigned the correct labels, assign six different pixel intensities to the each pixel value of a particular class.
15) Write the newly assigned intensity values to a file. This final output image should have the 6 differently colored images which indicates the textures have been correctly identified.

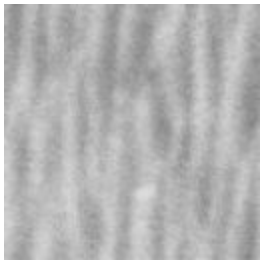
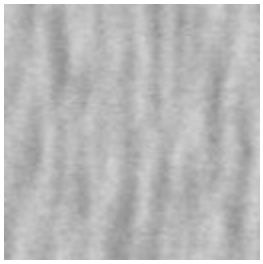
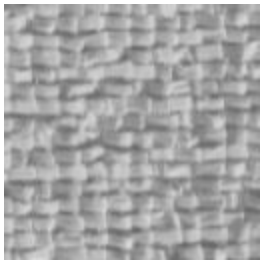
EXPERIMENTAL RESULTS:

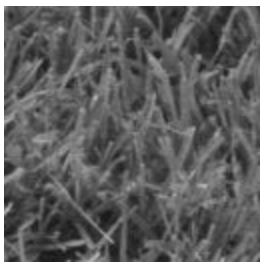
IMAGE CLASSIFICATION:



TEXTURE TYPE	IMAGE	EXPECTED RESULT	PREDICTED RESULT	LABEL ASSIGNED	PREDICTION ACCURACY
Texture 1		ROCK	ROCK	0	**CORRECT**

Texture 2		GRASS	GRASS	1	**CORRECT**
Texture 3		WEAVE	WEAVE	2	**CORRECT**
Texture 4		ROCK	ROCK	0	**CORRECT**
Texture 5		WEAVE	WEAVE	2	**CORRECT**
Texture 6		ROCK	ROCK	0	**CORRECT**

Texture 7		SAND	SAND	3	**CORRECT**
Texture 8		SAND	SAND	3	**CORRECT**
Texture 9		GRASS	WEAVE	2	**WRONG**
Texture 10		SAND	SAND	3	**CORRECT**
Texture 11		WEAVE	WEAVE	2	**CORRECT**

Texture 12		GRASS	GRASS	1	**CORRECT**

DISCUSSION:

I performed the texture classification of the twelve textures by applying the nine 5x5 Laws filters corresponding to the 1D kernels for Edge, Spot and Wave.

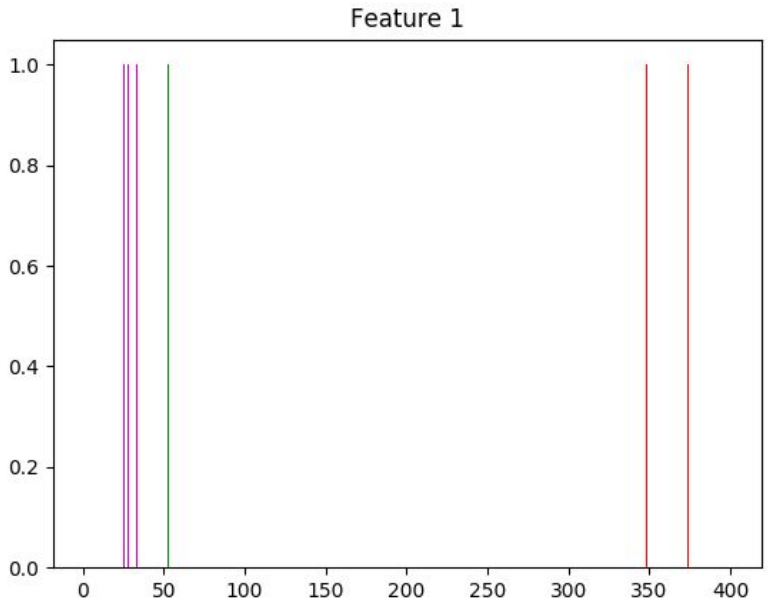
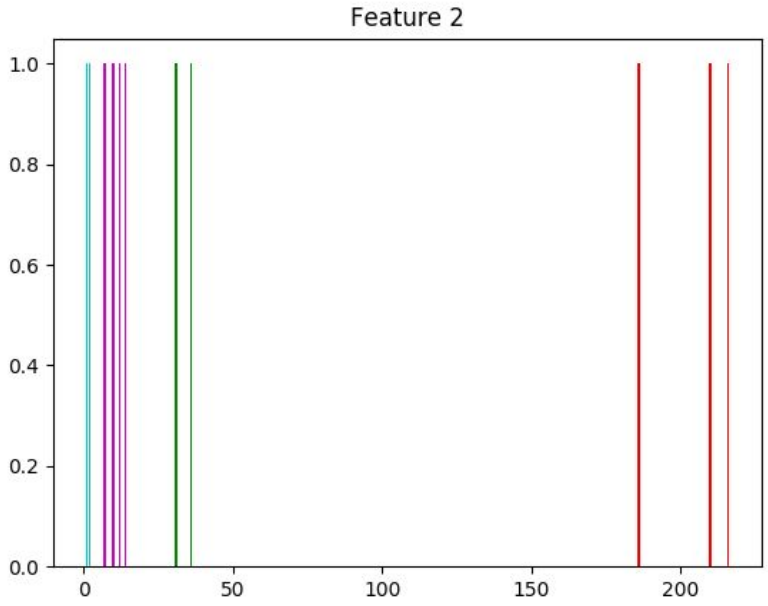
Name	Kernel
L5 (Level)	[1 4 6 4 1]
E5 (Edge)	[-1 -2 0 2 1]
S5 (Spot)	[-1 0 2 0 -1]
W5(Wave)	[-1 2 0 -2 1]
R5 (Ripple)	[1 -4 6 -4 1]

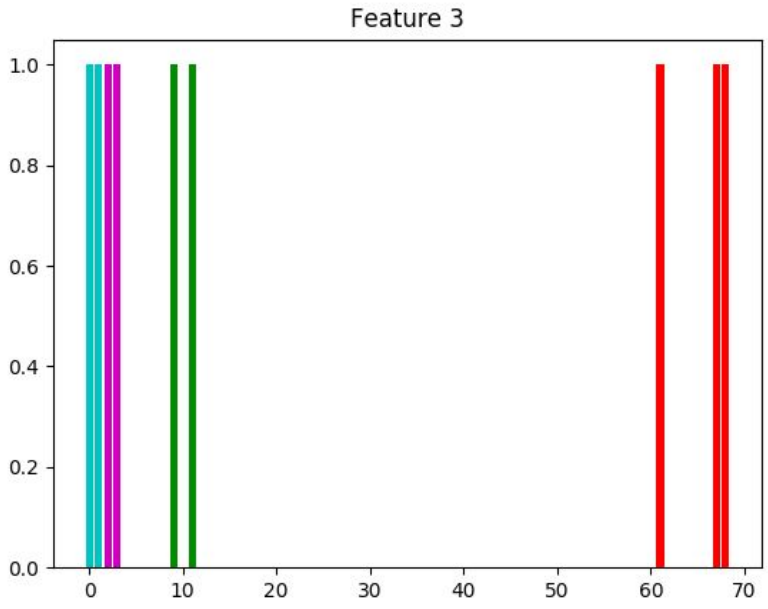
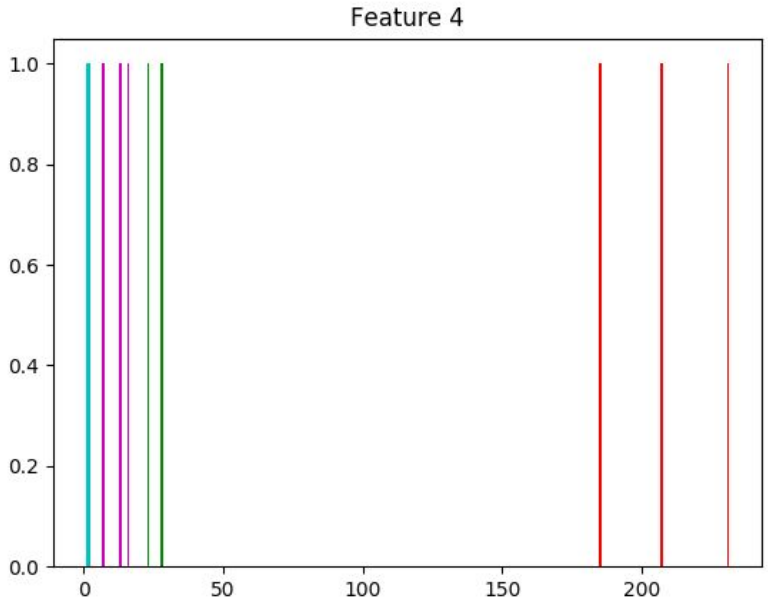
4 feature vectors belonging to the four classes were chosen as centroids and k-means clustering was performed and at the end of the k-means cluster it was found that all the texture were correctly classified except for one - a texture belonging to the grass class was classified as a weave - texture 9 was misclassified.

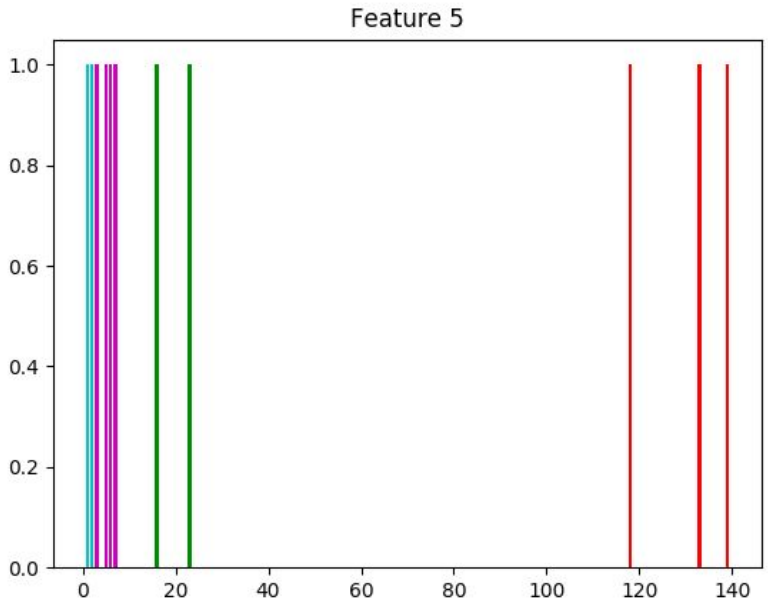
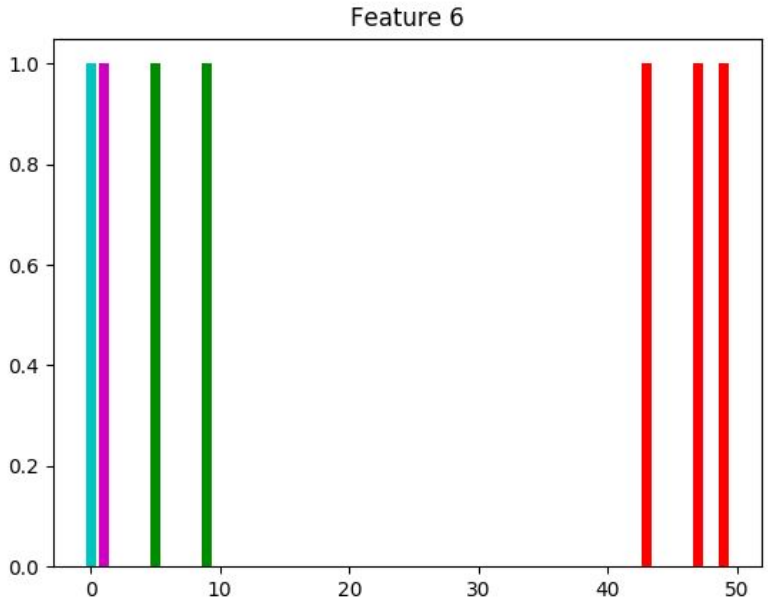
When applying the filters, pixel were replicated when the corners were being multiplied by the masks, in order not to pick up any junk values. The 9D feature vector obtained for each image is used to perform classification on the image. Hence the feature vectors convey information about the various distinct properties. When the bar plots of each of the feature vectors for the images were obtained, it was found that the feature 2 extracted by applying the Laws filter formed by taking the tensor product of filters - Edge and Sand, performed the best compared to the rest of the filter to discriminate the images from one another. Hence the E5S5 could be called the feature with the highest discriminant power.

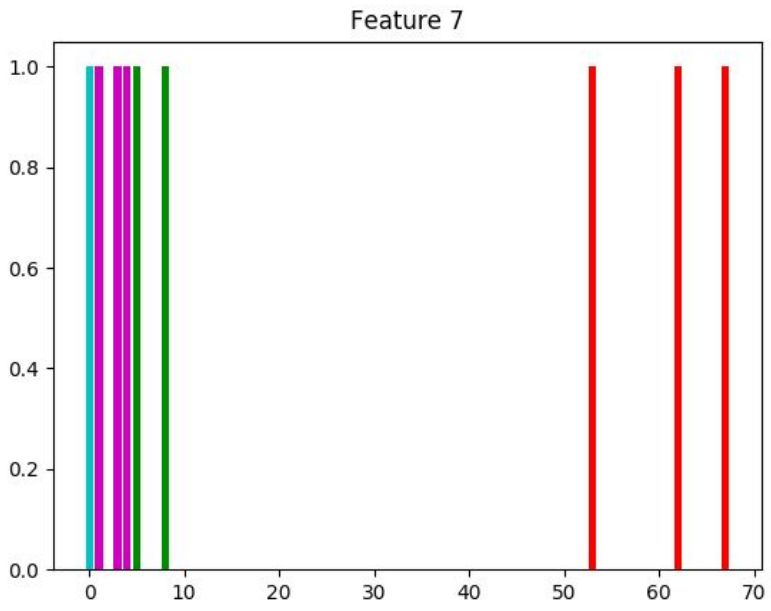
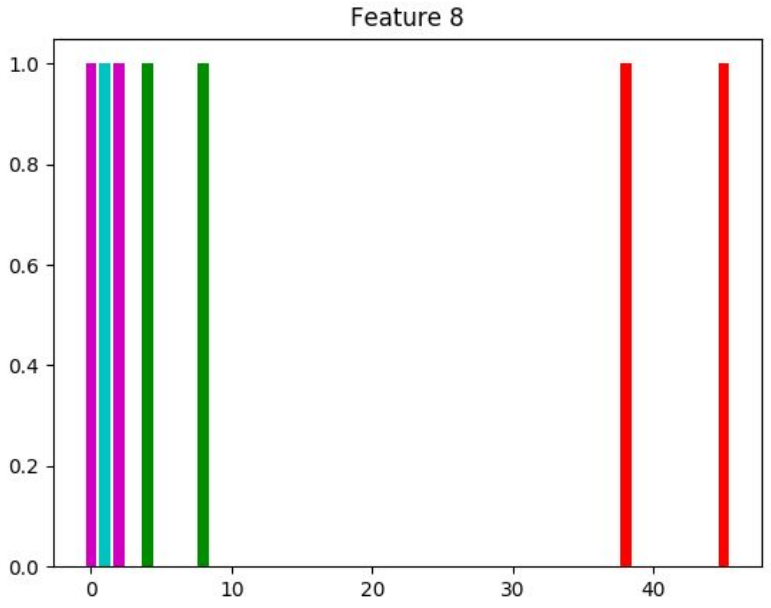
Similarly upon evaluation it was found that despite many of the Laws filters producing plots with overlapping lines and hence making it hard to differentiate and separate the images, the

Feature 8 filter - Weave and Sand performed the worst since it was producing plots such that it created a vector between two different class of images and hence it was not possible to classify the images properly. There we can say that Weave and Sand feature has the weakest discriminant power.

LAWS FILTER TYPE	PLOT OF EACH FEATURE FOR THE 12 TEXTURES
Edge ⊗ Edge (5x5 filters) - We observe that for this feature, the one of the image class is not distinctly visible	
Edge ⊗ Sand (5x5 filters) **STRONGEST DISCRIMINANT POWER** -We can see that all the 12 images can be represented as 12 lines and hence they can be clearly discriminated from one another using this feature	

Edge ⊗ Weave (5x5 filters) - We observe an overlap of images for this feature, hence the thicker lines	
Sand ⊗ Edge (5x5 filters) - We observe an overlap of images for this feature, hence the thicker lines, But majorly the there are fewer overlaps, there is a overlap of the grass and	

Sand ⊗ Weave (5x5 filters) - We observe an overlap of images for this feature, hence the thicker lines	
Sand ⊗ Weave (5x5 filters) - We observe an overlap of images for this feature, hence the thicker lines	

Weave ⊗ Edge (5x5 filters) - We observe an overlap of images for this feature, hence the thicker lines	
Weave ⊗ Sand (5x5 filters) -We observe an overlap of images for this feature, hence the thicker lines	

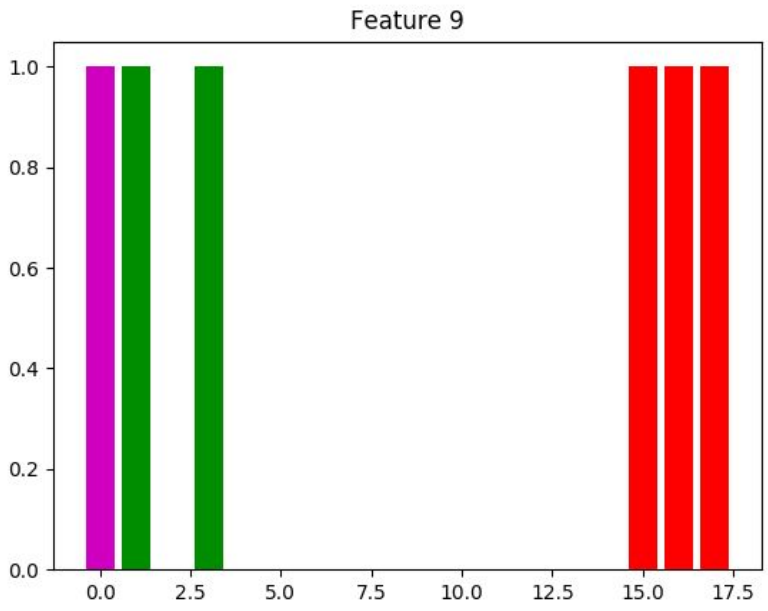
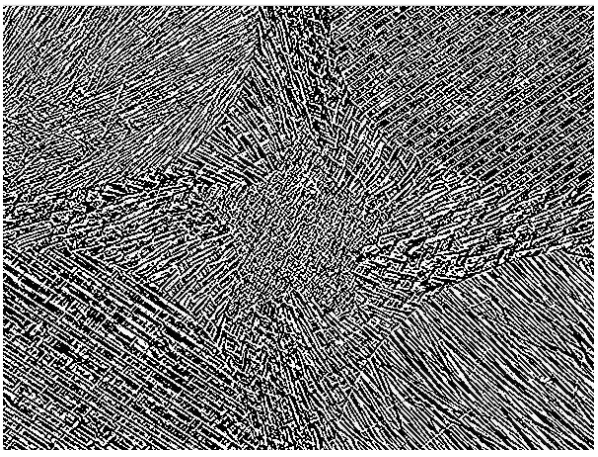
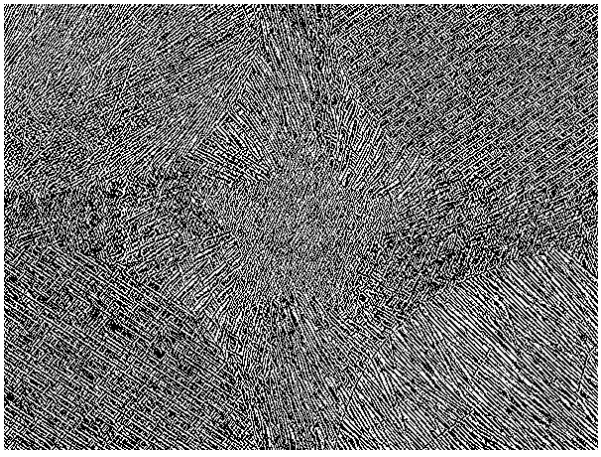
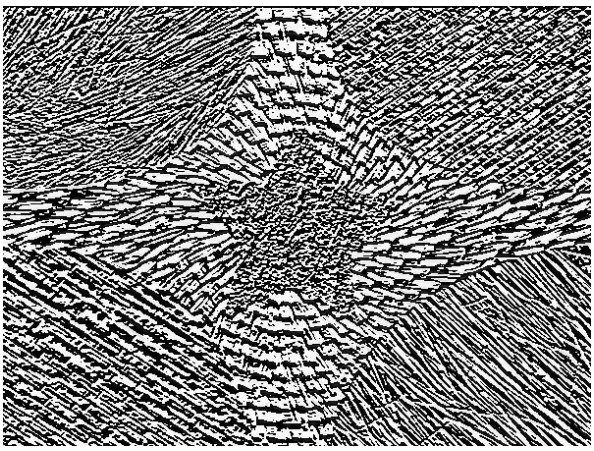
Weave ⊗ Weave (5x5 filters) - We observe an overlap of images for this feature, hence the thicker lines	

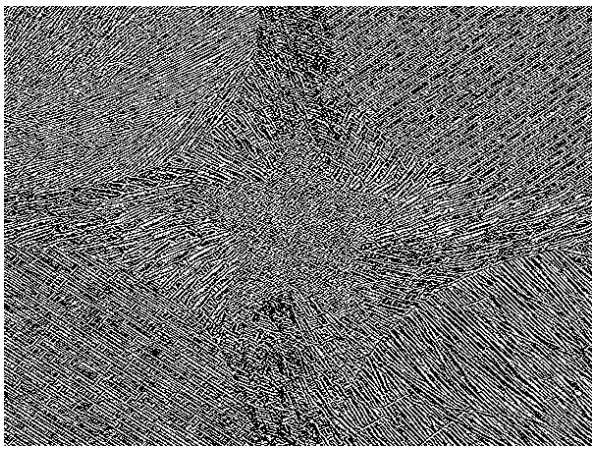
IMAGE SEGMENTATION:

EXPERIMENTAL RESULTS:

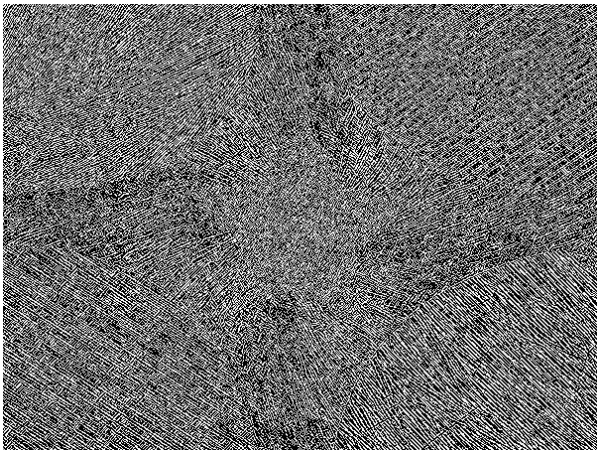
 Filter 1 - E3E3	 Filter 2 - E3S3



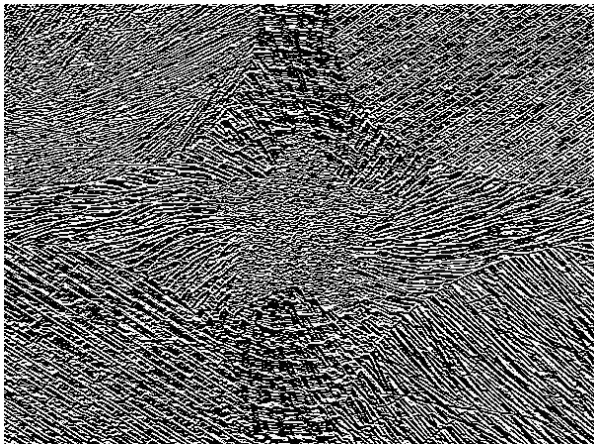
Filter 3 - E3L3



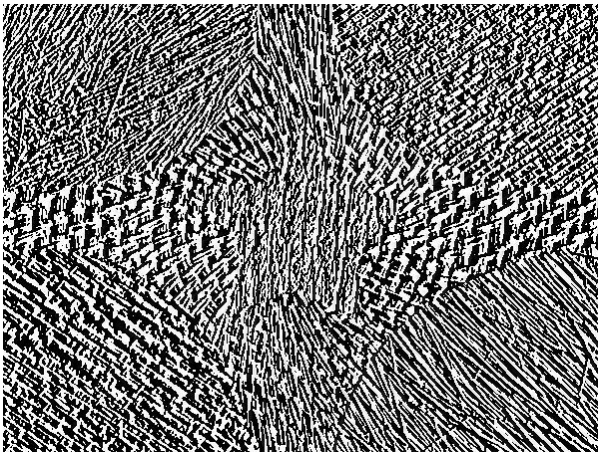
Filter 4 - S3E3



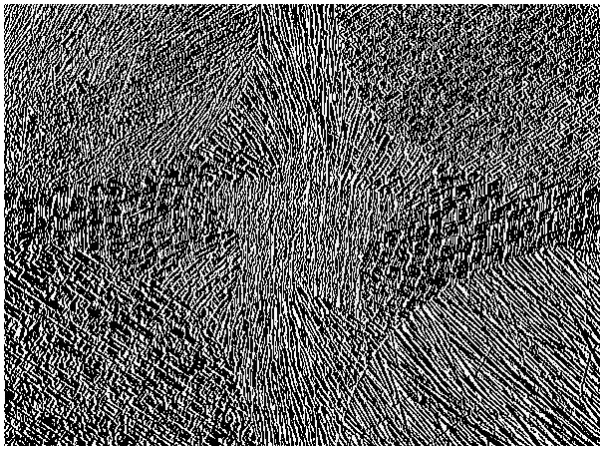
Filter 5 - S3S3



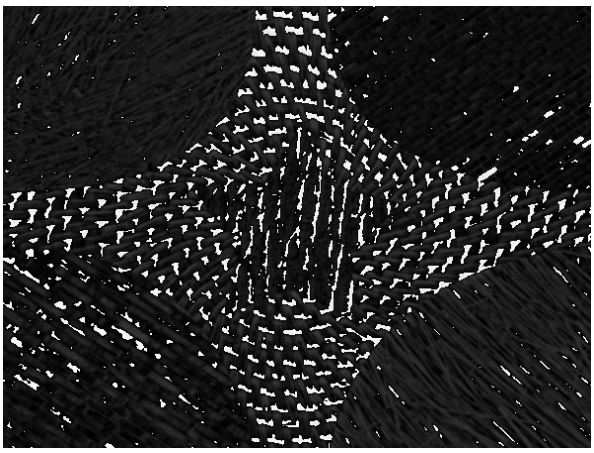
Filter 6 - S3L3



Filter 7 - L3E3



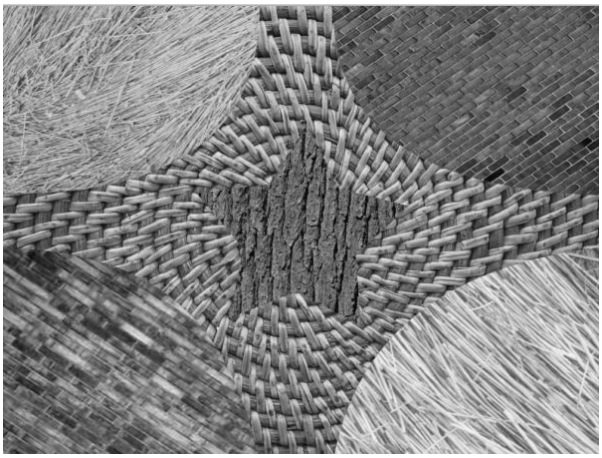
Filter 8 - L3S3



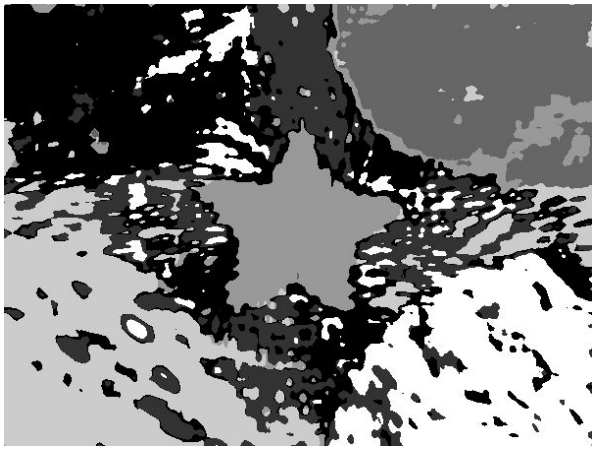
Filter 9 - L3L3

N = 19 output

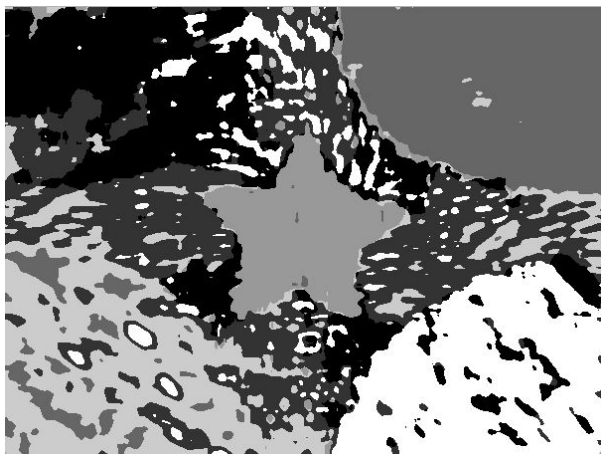
I have considered iterations from 0 - 20 and shown output after every 5th stage:



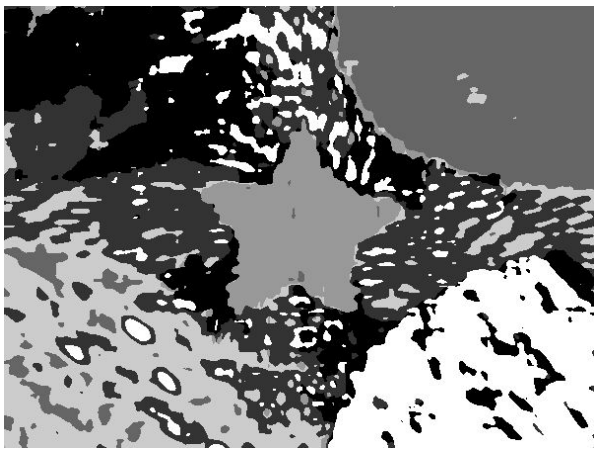
Original input = comb.raw



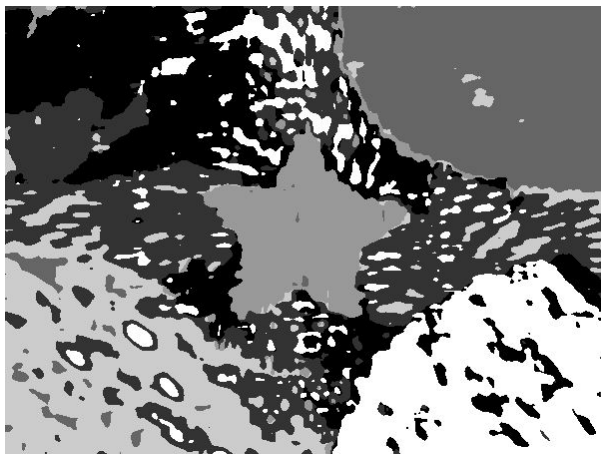
N = 19



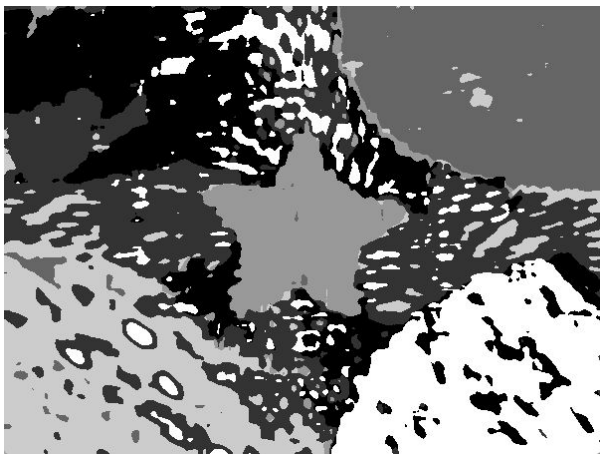
N = 19



N = 19

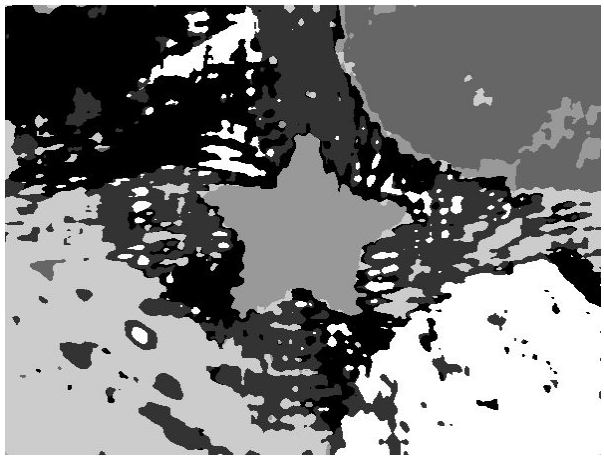


N = 19

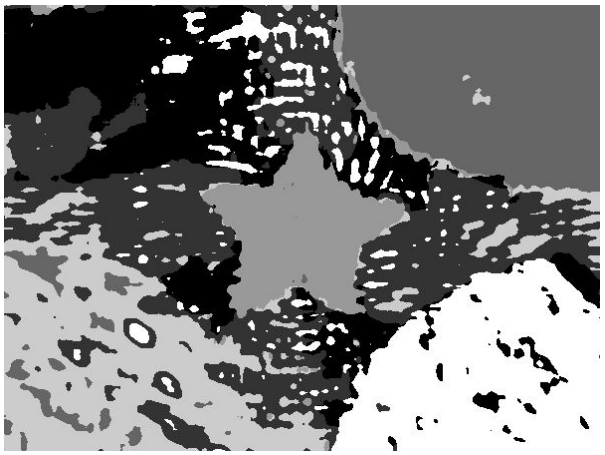


N = 19

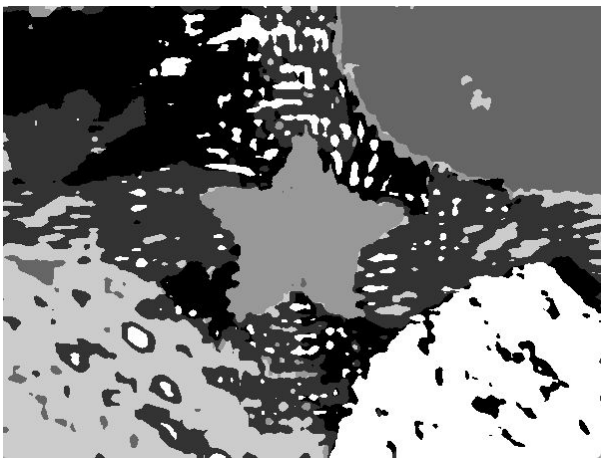
N = 21 - BEST OUTPUT OBTAINED FOR THIS WINDOW SIZE



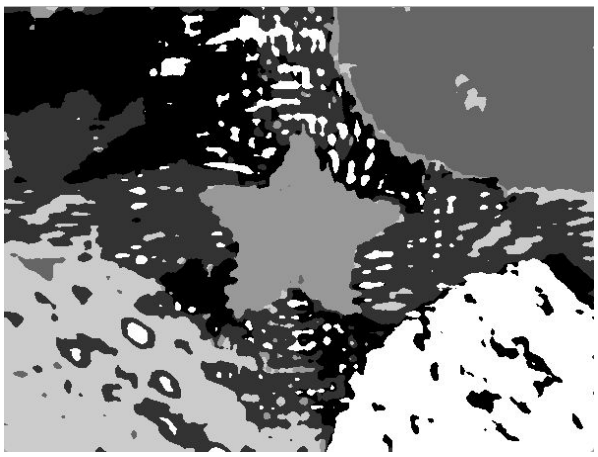
N = 21



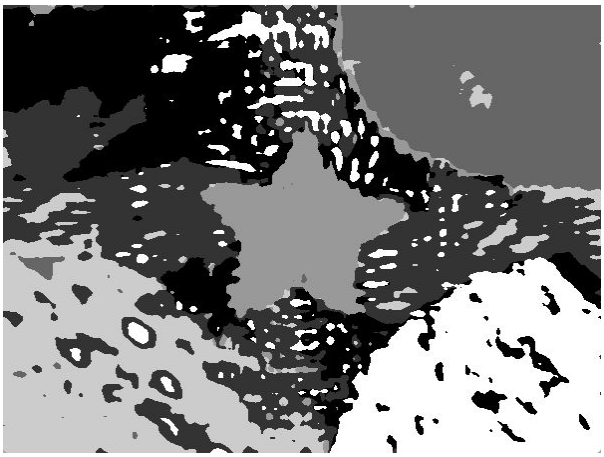
N = 21



N = 21

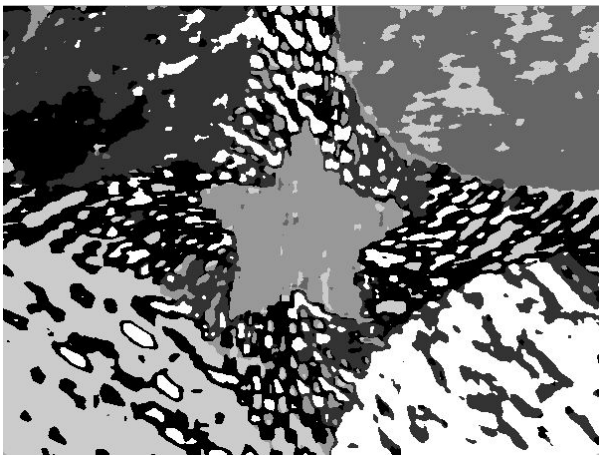


N = 21

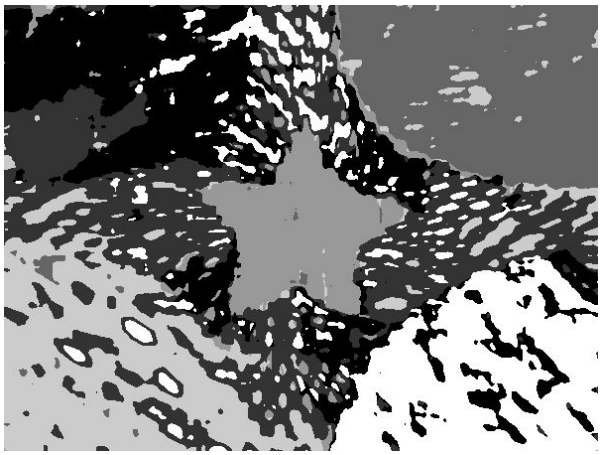


N = 21

OTHER WINDOW SIZES = 15, 17

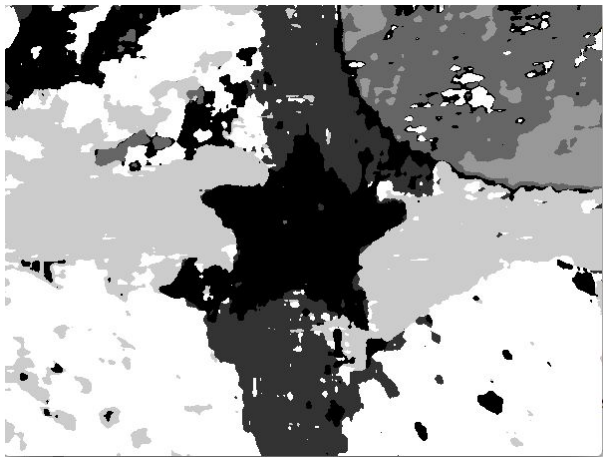


N = 15

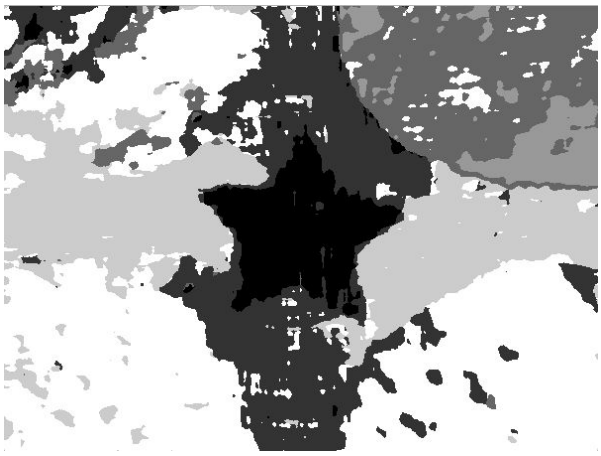


N = 17

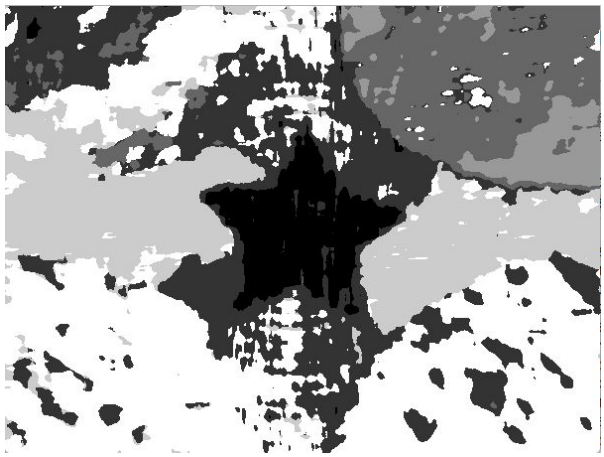
With $L3L3^T$ normalization :



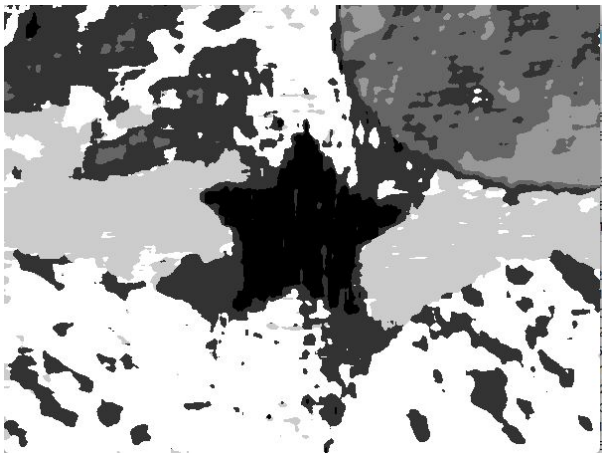
N = 19



N = 19

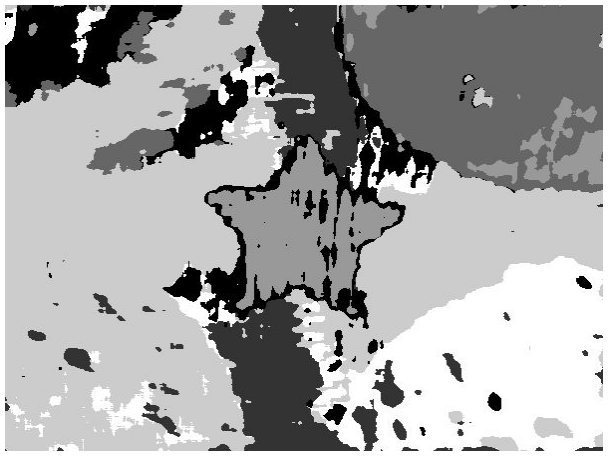


N = 19

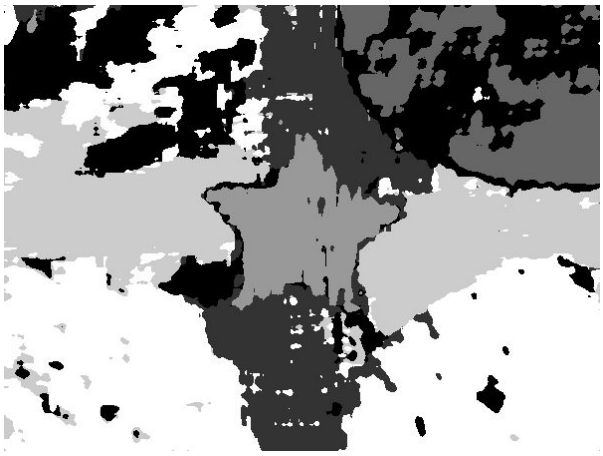


N = 19

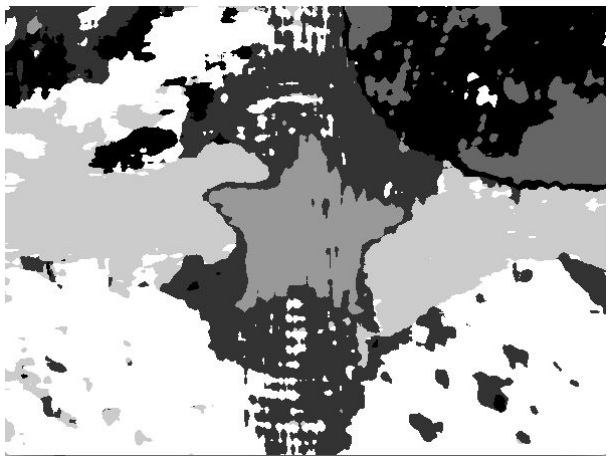
With $L3L3^T$-normalization : N = 21



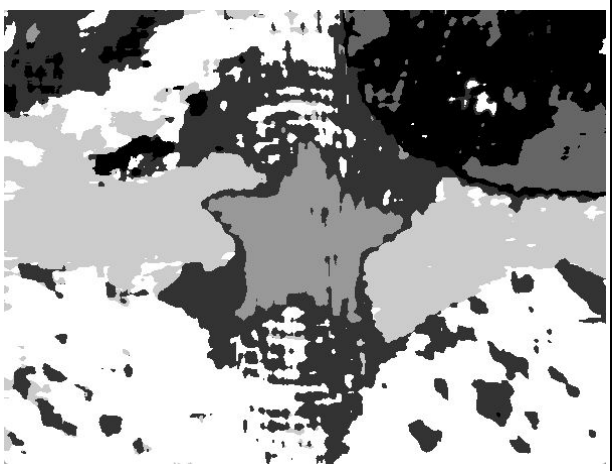
N = 21



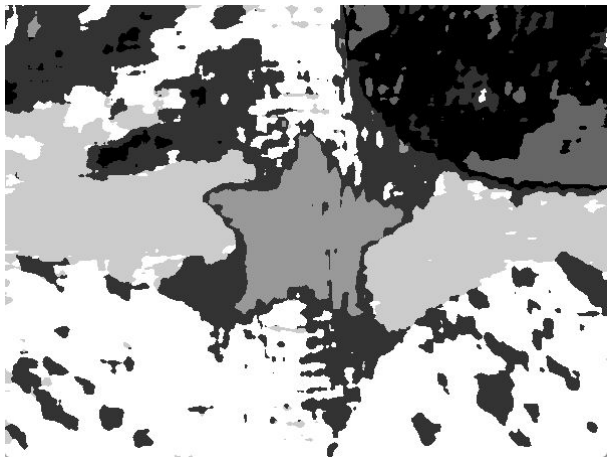
N = 21



N = 21

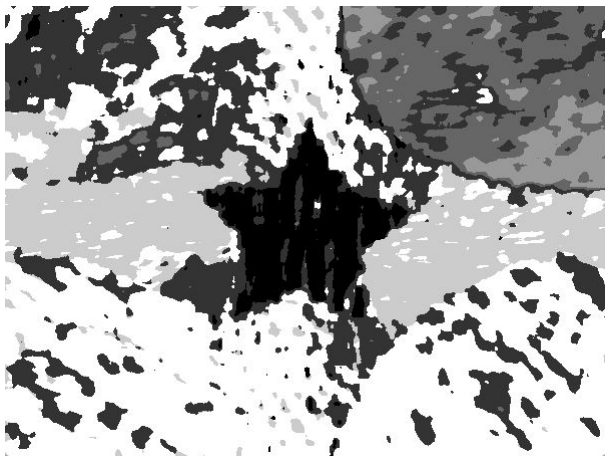


N = 21

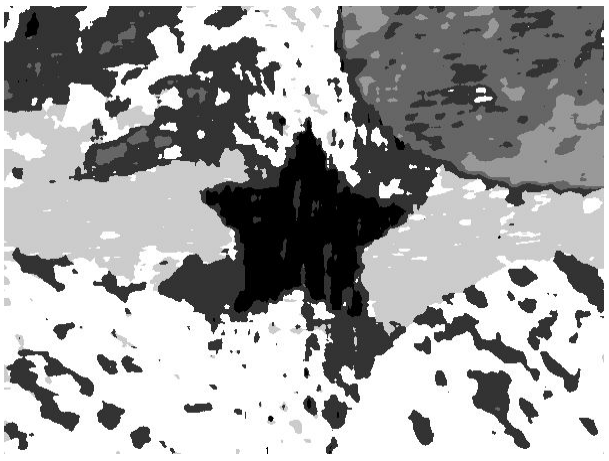


N = 21

L3L3^{T} Normalized output : For different window sizes : N= 15, N = 17



N = 15



N = 17

DISCUSSION

Texture Segmentation has been performed on the comb.raw image and 6 different segments have been obtained which are assigned 6 different gray levels to differentiate between them. The output using the nine 3*3 laws filter only is not very good as it does not separate the textures completely. There is some discrepancy seen in the output. It takes a lot of time to run the code as there are high number of dimensions and many of them are redundant.

Different window sizes (15*15), (17*17), (19*19) and (21*21) were used to compute mean of the image and the energy vectors. Mean of the image is calculated to reduce the illumination effects in the image. Window size (21*21) gives the best output. Increasing the window size beyond (21*21) does not make any difference in the output and hence should not be increased as it will add to the time complexity.

When L3L3 is not used for normalizing the values, the segmentation gives better output.

To improve the segmentation output, post processing methods like non-maximal suppression and PCA can be used to improve the results. PCA will do dimension reduction and the redundant features will be discarded. Thus the computation time (time complexity) will be reduced and better results will be obtained.

Image segmentation using PCA

ABSTRACT AND MOTIVATION

PCA is a dimensionality reduction technique which can be used in Image Processing to improve the time complexity of a model or algorithm. PCA stands for Principal Component Analysis and preserves only important features and discards the redundant features.

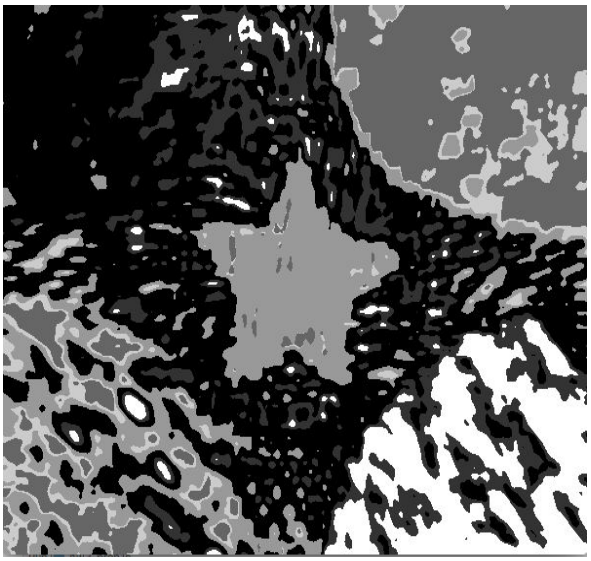
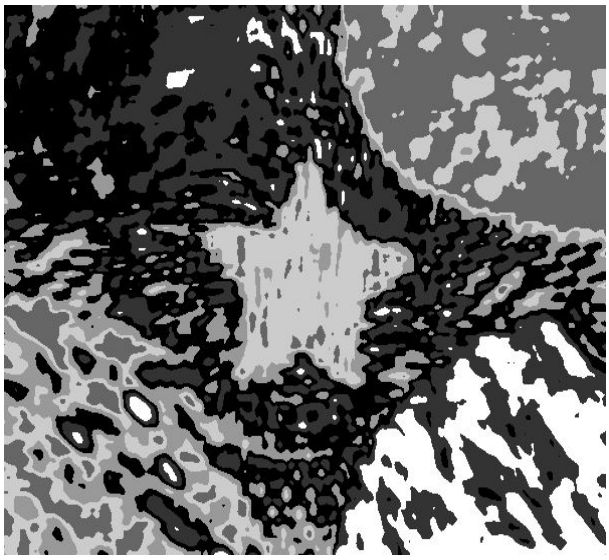
PROCEDURE

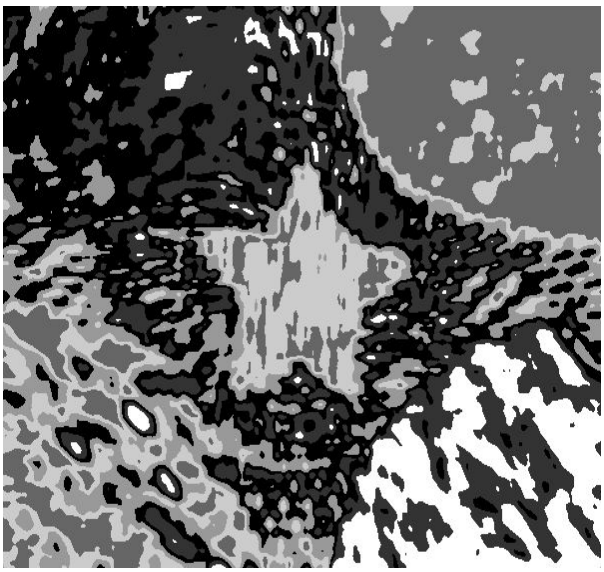
The algorithm for this remains the same except that here all the five 1D kernels are used to get 25 laws filters.
Once the energy vectors are generated, all the vectors are exported to MATLAB for Principal Component Analysis where an energy vector with five dimensions is generated and the rest dimensions are discarded as they are redundant. All the important information about the composite texture is preserved which is present in the first five dimensions.

IMAGE SEGMENTATION USING PCA :

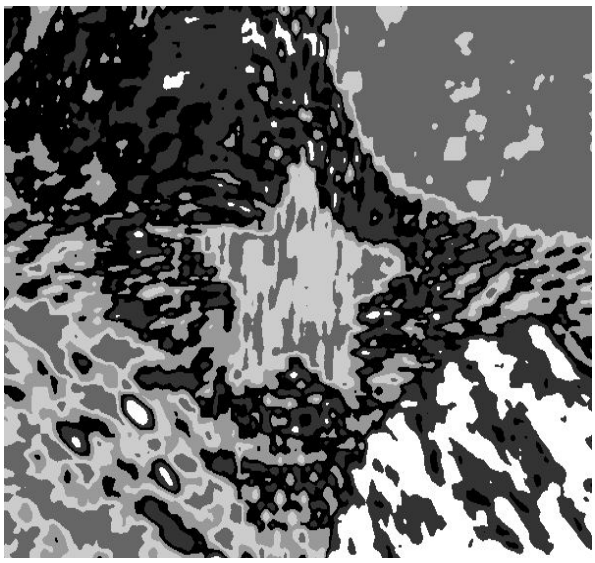
EXPERIMENTAL RESULTS:

The outputs were obtained after every 5 iterations

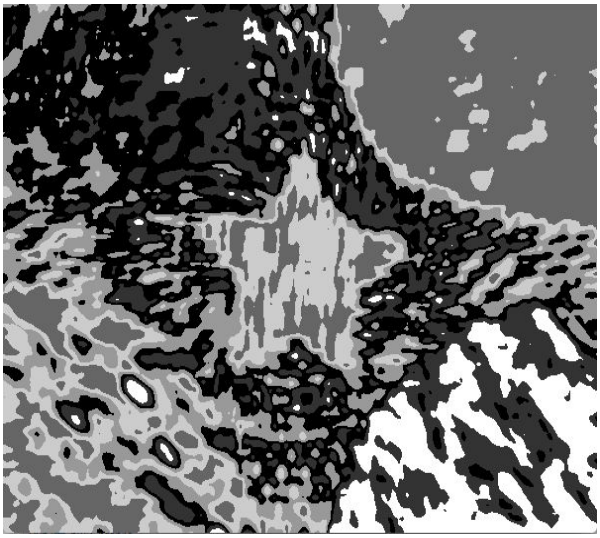
 N = 21	 N = 21



N = 21



N = 21



N = 21

DISCUSSION

Dimensionality reduction was done wherein the feature vectors were reduced to 5 dimensions. Again, I tried different window sizes wherein I discovered that as the window size is increased, the output has distinct regions. The segmentation result is proper for window size of 21. However, the image segmentation output without applying dimension reduction is better than the segmentation output obtained after doing PCA.
I think that doing some post-processing on the PCA output will prove to be beneficial. The post processing can be in the form of noise removal to smoothen the textures to give distinct regions.

PROBLEM 2: EDGE DETECTION

2 A. BASIC EDGE DETECTOR

ABSTRACT AND MOTIVATION

Edge detection is the process of identifying and locating the edges in an image. Edges can be defined as a sharp discontinuities in an image or the points in an image along which the intensity of pixel values varies rapidly with a large gradient. Edges generally characterize the boundaries of objects and are necessary to identify each object or make sense of the image.

Edge detection in images is necessary to understand the contents of the image and is helpful in applications like image segmentation and object detection which has applications in computer vision. Edge detection is generally done using 2D filters which are constructed such that they are sensitive to sharp variations in intensities. There are many edge detection techniques available, many of them designed to work on noisy images and images in which edges are characterised by gradual changes in the image and not sharp intensity changes, which are more common while performing edge detection operations in reality. Thus efficient edge detection algorithms take care of common problems like false edge detection, missing true edges, edge localization, high computational time and problems due to noise.

APPROACH AND PROCEDURES

In this problem I understood and implemented three edge detection techniques - Sobel edge detector, Laplacian of Gaussian and Structured Edge detector.

Sobel edge detector
The Sobel operator is an edge detection operator uses a 3x3 kernel which is convolved with the input image (taking 3x3 image pixels at a time) to generate the approximations of X-gradient and Y-gradient in the horizontal and vertical directions respectively. The kernels are applied on the image separately and outputs are combined to find the magnitude of the Gradient given by the formula:

$$G_{magnitude} = \sqrt{G_x^2 + G_y^2}$$

The direction of the gradient is given by the following equation:

$$G_{gradient} = tan(\frac{G_x}{G_y})$$

The Sobel operators are given by:

X Gradient:

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

Y Gradient:

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

Laplacian of Gaussian:

The Laplacian operator of Gaussian filter is equivalent to applying a Gaussian filter on an image and then applying the Laplacian filter for edge detection. The Laplacian filter is designed to detect extreme changes in intensity of the image and is a single kernel filter which calculated the second order derivative of the image to perform edge detection.

Since the 2D Laplacian filter is taking the second order derivative, it is very sensitive to noise and hence providing a Gaussian smoothed image is necessary. Applying the Gaussian filter removing the high frequency noisy components from the image. The LOG filter is obtained by convolving the Gaussian filter with the Laplacian filter and this resultant filter is used to perform the edge detection operations.

The 2D LOG filter function with a standard deviation of $\sigma = 1.4$ is given below:

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

The following 9D filter was used for the performing the edge detection operation:

$$\begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -24 & -40 & -24 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -24 & -12 & 3 & 5 & 2 \\ 1 & 4 & 5 & 3 & 0 & 3 & 5 & 4 & 1 \\ 1 & 2 & 4 & 5 & 5 & 5 & 4 & 2 & 1 \\ 0 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 0 \end{bmatrix}$$

The zero crossing detector is then used to detect the edges in the image which has been filtered by the Laplacian of Gaussian filter. This detector looks for points in the image where the pixel intensity crosses zero i.e the LOG output changes signs. It has been generally observed that edges occur at these points in the image. Therefore we can say that the zero crossing detector output is generally a binary image with the edges being represented by lines of single thickness, showing the positions of the zero crossing points.

Procedure for Sobel detector:

1) Read the input file into a 1D array and extract the R,G and B channels.
2) Use the luminosity method to convert the image from color to grayscale :
    Grayscale = (0.21 * R) + (0.72 * G) + (0.07 * B)
3) Save the X and Y gradient filters for the image as a 1D array of 9 elements
4) Generate the X and Y gradient images by
    a) multiplying every pixel of the input grayscale image and its 8 neighbours with the elements of the mask
    b) taking a sum of all the products and replacing the result into a new Image array
5) Calculate the magnitude of the gradient of the image by combining the X and Y gradient image values
6) The resultant magnitude values will be greater than the range 0 – 255, hence we need to normalize the outputs
7) Normalize the pixel values in the magnitude array by finding the minimum and maximum values and using the following formula:
    Normalized value = 255 * (Value – min) / (max – min)
8) Obtain the cumulative histogram of the values in the normalized Image.
9) Take the value at the 90% location of the histogram as the threshold
10) Compare each value of the normalized value with the threshold and set the values below the threshold to 255(black intensity) and every value greater than the threshold to 0(white intensity)

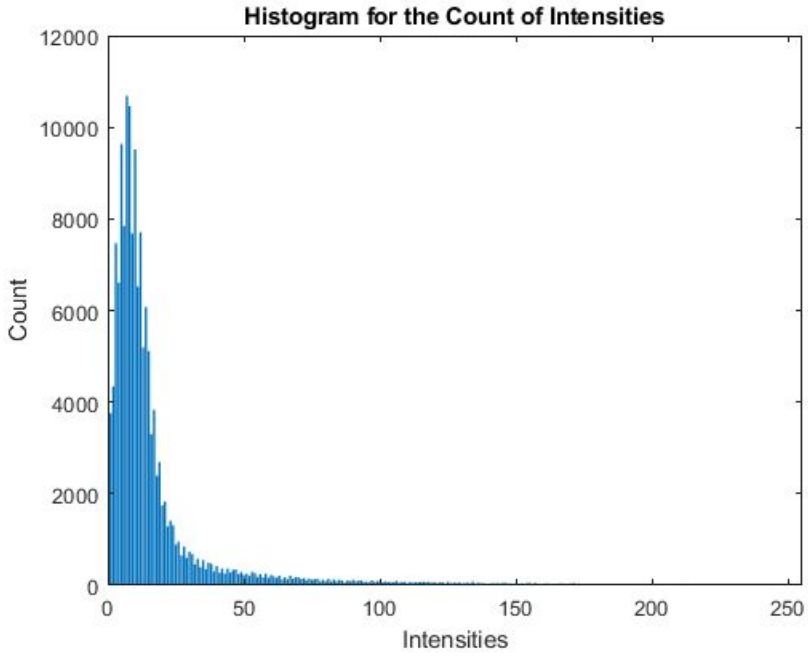
11) This will give us the final output image with only the edges.

Procedure for Laplacian of Gaussian detector

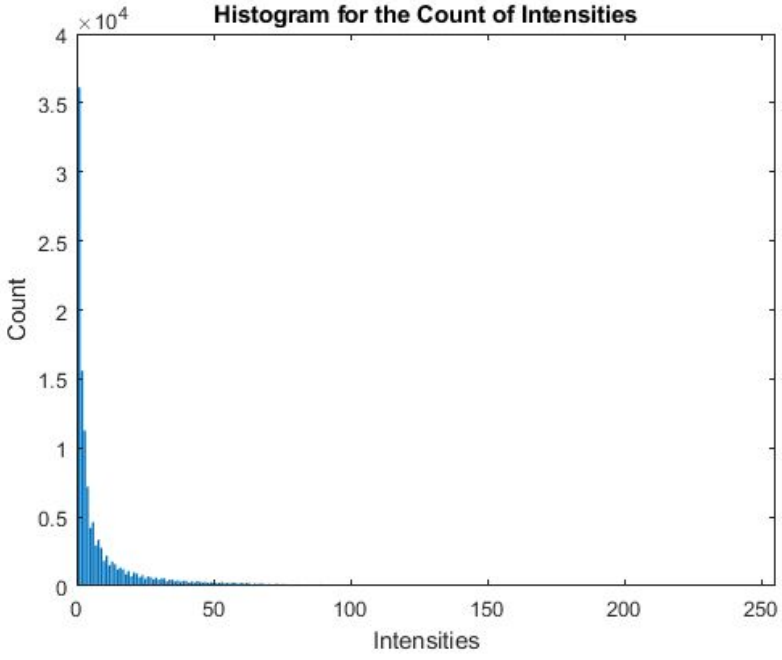
1) Read the input file into a 1D array and extract the R,G and B channels.
2) Use the luminosity method to convert the image from color to grayscale :
Grayscale = (0.21 * R) + (0.72 * G) + (0.07 * B)
3) Save the 9x9 LOG filter as a 1D array of 81 elements
4) Generate the gradient image by
    a) multiplying every pixel of the input grayscale image and its 80 neighbours with the elements of the LOG filter
    b) taking a sum of all the products and replacing the result into a new Image array
5) Convert the gradient image array into 1D to calculate min, max values for normalization
6) The resultant gradient values will be greater than the range 0 – 255, hence we need to normalize the outputs
7) Normalize the pixel values in the gradient array by finding the minimum and maximum values and using the following formula:
        Normalized value = 255 * (Value – min) / (max – min)
8) Obtain the histogram of the values in the gradient image
9) Plot the histogram in Matlab
10) From the plot get the two consecutive knee values around the spike
11) Use the two values and thresholds 1 and 2.
12) Set all the values below the lower threshold as -1 and values greater than the higher threshold as 1. All the values between the lower and higher threshold are set to 0. This will give the values for the zero crossing detector image used in the next stage.
13) Set all pixel values in locations corresponding to the -1, 0 and 1 to intensities 64, 128 and 192. This will give us the grayscale thresholded image.
14) Now consider a window of 3x3 which will traverse through the zero crossing detector image
15) Check if a pixel in the zero crossing detector image is zero:
    a) if yes, then check if any of its 8 neighbors in the 3x4 neighborhood are 1 or -1.
        i) If yes, then set that pixels intensity in the output image to 255.
        ii) If no, then set that pixels intensity in the output image to 0.
    b) If no, then set that pixel intensity in the output image to 0.
16) This gives the final output image with the edge pixels set to black and remaining pixels set to white

EXPERIMENTAL RESULTS:

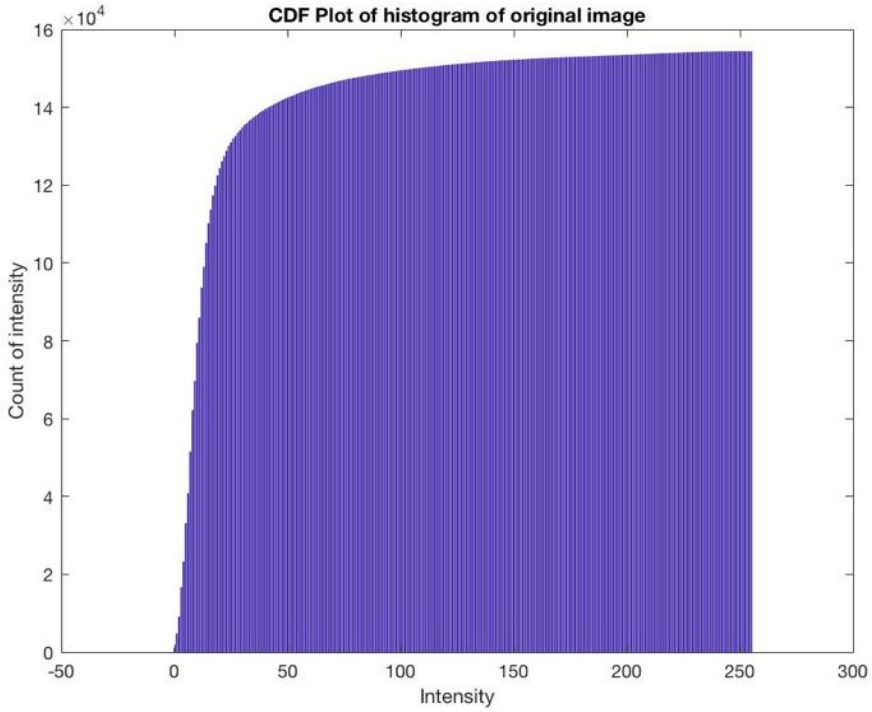
SOBEL EDGE DETECTOR:



Boat_noisy.raw image

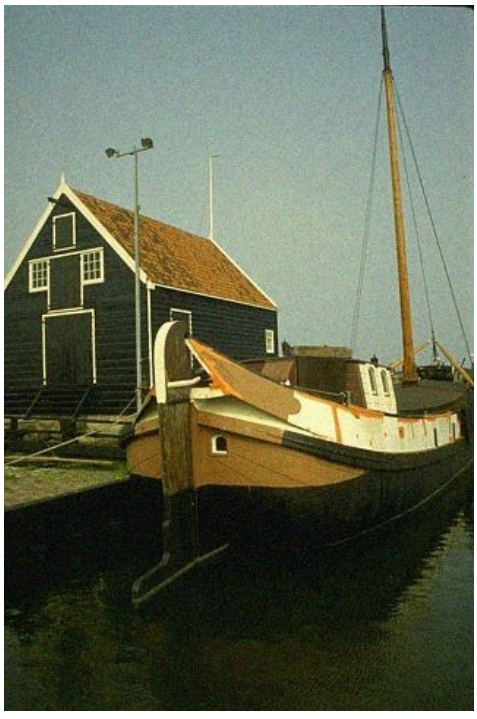


Boat.raw image



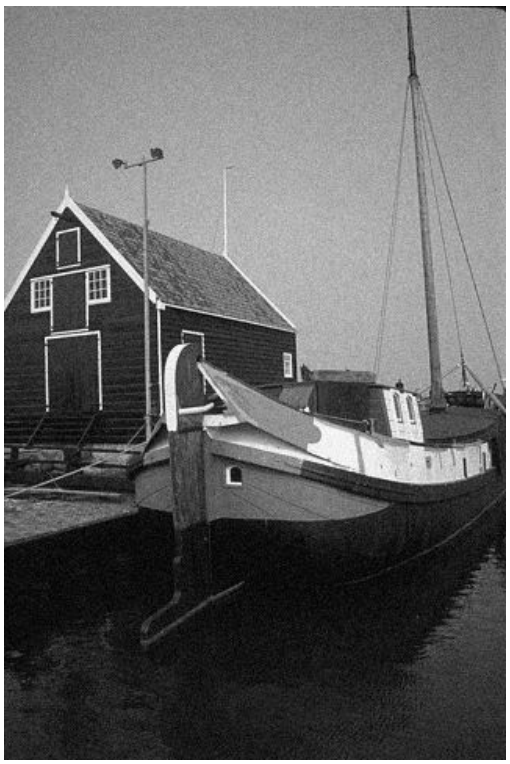
CDF Plot of the boat.raw image - which is used to choose the 90% threshold value for edge detection.

Original - boat.raw image

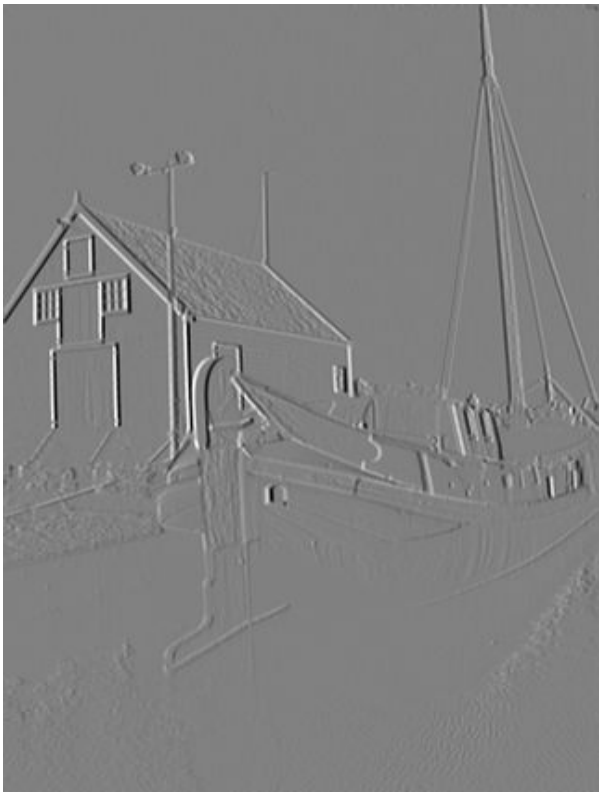


Original - boat_noisy.raw image

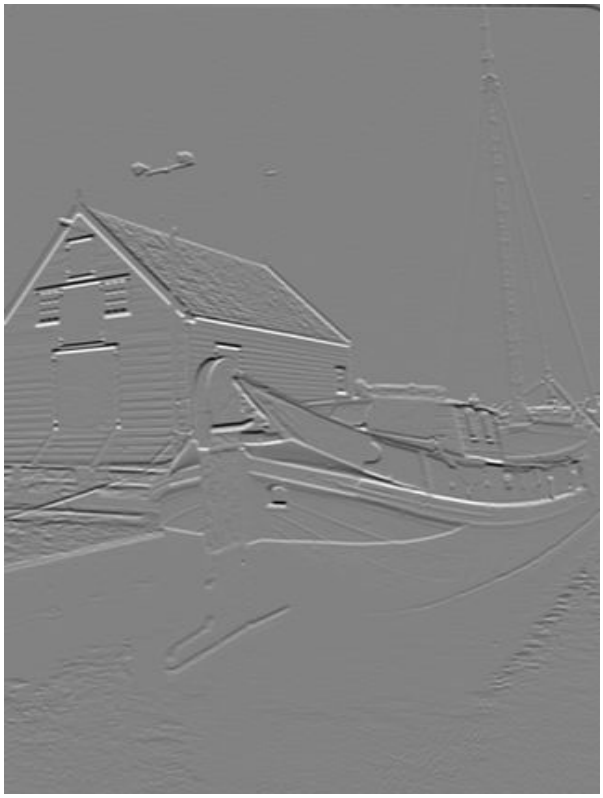
Grayscale image - boat.raw



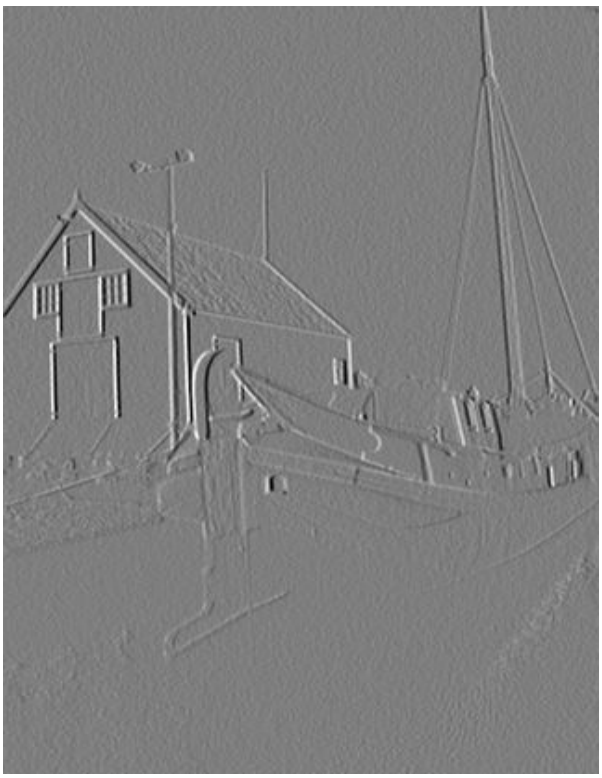
Grayscale image - boat_noisy.raw



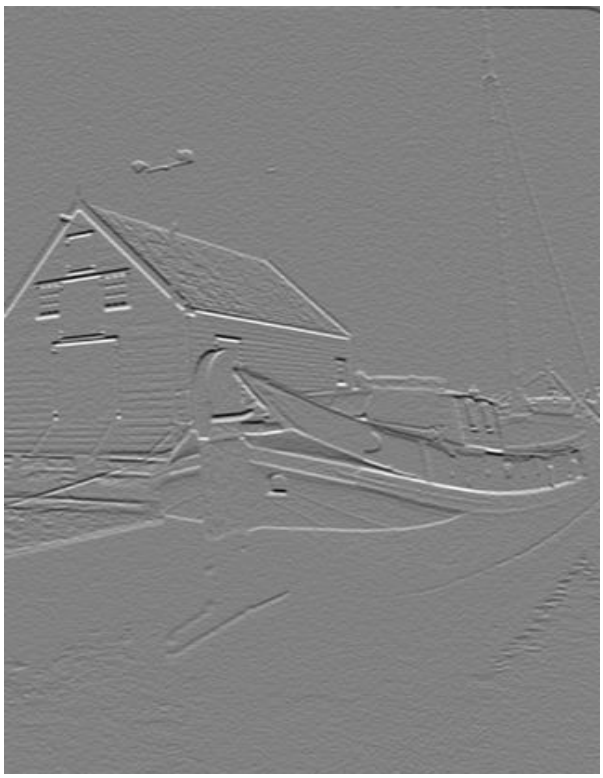
X Gradient output - boat.raw



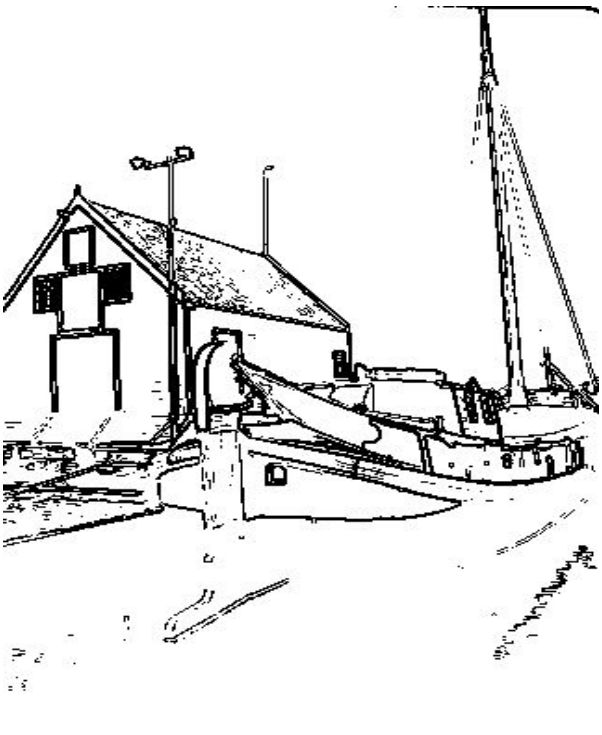
Y Gradient output - boat.raw



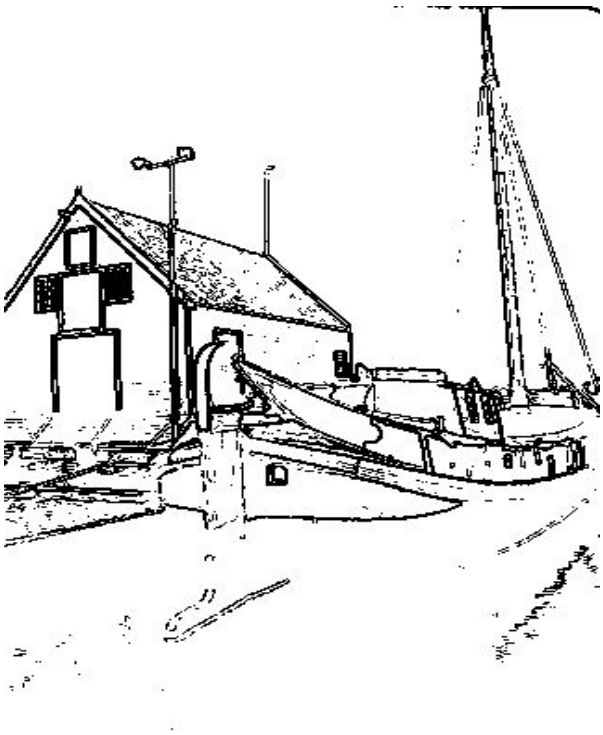
X Gradient output - boat_noisy.raw



Y Gradient output - boat_noisy.raw



Sobel edge output - boat.raw



Sobel edge output - boat_noisy.raw

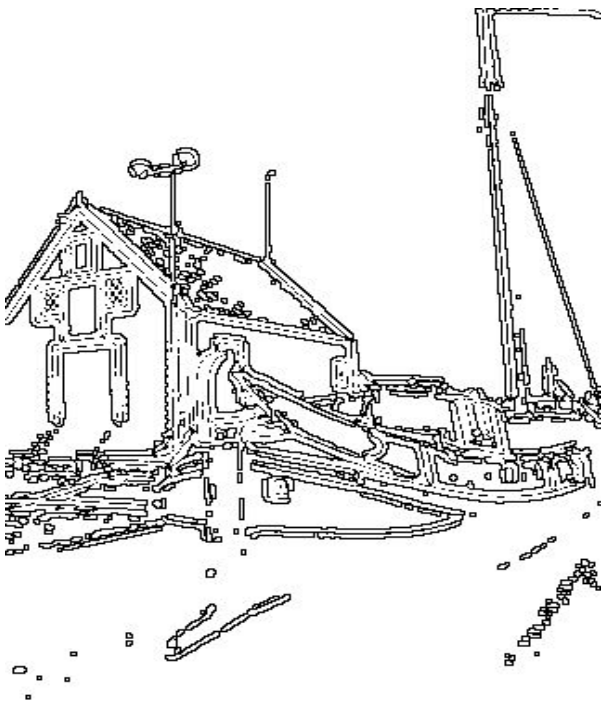
LAPLACIAN OF GAUSSIAN EDGE DETECTOR:

Original - boat.raw image

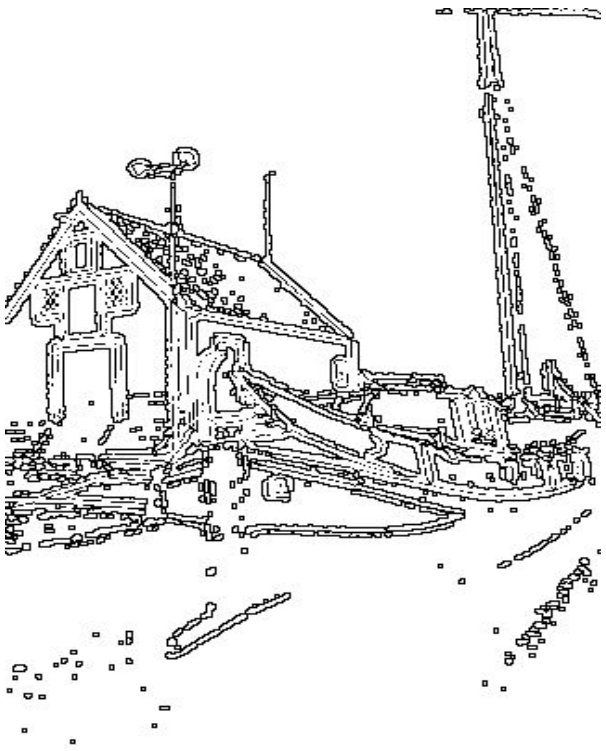
Original - boat_noisy.raw image

LOG Grayscale output - boat.raw

LOG Grayscale output - boat_noisy.raw



LOG Edge output - boat.raw



LOG Edge output - boat_noisy.raw

DISCUSSION:

Sobel mask:

The output for the Sobel edge detector was obtained by applying the Sobel masks and calculating the X and Y gradient values. The magnitude component of the gradient was calculated using the formula mentioned above. The X-gradient and Y-gradient outputs for the boat.raw and boat_noisy.raw images have been shown above. The X gradient output shows that the X-gradient mask detected mainly the horizontal edges and the Y-gradient mask output indicated that the vertical edges were detected by it.

The 90% threshold value is obtained from the CDF of the magnitude gradient image and using this the sobel edges are obtained. The Sobel mask does not perform very well for the noisy image as can be seen in the image above where the noisy image has more dots than the non-noisy image. Also the edges are thicker in the Sobel outputs, thereby we can say the Sobel mask is not very efficient.

Laplacian of Gaussian edge detector:

The LoG filter is a combination of the Laplacian and the Gaussian filter. The Laplacian filter being used for edge detection and the Gaussian filter being used for removing the noise components in the image. The outputs obtained for the LoG filter is very good compared to the

Sobel detector, mainly due to the removal of noisy components and the zero-crossing detector which ensures the edges are actually edges and not thicker patches of pixels.

The thresholds for the zero-crossing detector - 122 and 156 are the knee points which are obtained by plotting the histogram after applying the LoG filter and taking the points next to the highest spike, from which the values start increasing towards the spike(increasing for the left threshold and decreasing for the right threshold).

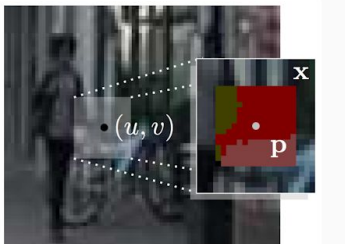
The output obtained for the LoG filtered noisy boat image is very much better than that of the Sobel filter since the noisy components are removed from the image and hence a good edge image is obtained.

STRUCTURED EDGE DETECTOR:

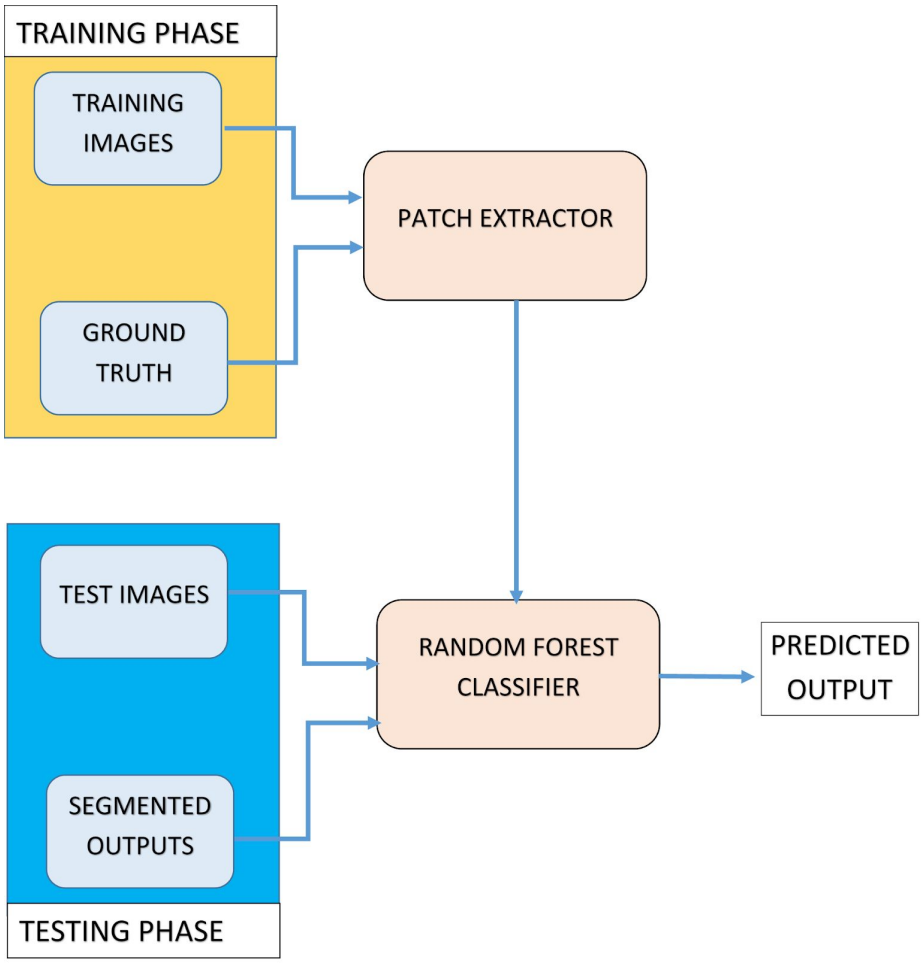
APPROACH AND PROCEDURE:

The structured edge detector is a supervised edge detector which uses the concept of random forests to perform edge detection and we can evaluate the accuracy of the prediction by comparing the predicted outputs of the image with the actual ground truth values obtained by using the image descriptions of how people perceived the images and the edges in them.

The structured edge detector considers patch around the target pixel and calculates what is the likelihood of the pixel being an edge. Patches which are edges are then classified as edge tokens to using the popular classifier like the random forest classifiers. These sketch tokens generally depict the important portions of the image like the parallel lines, T junctions etcs. When the input datasets like the training images are complex, a structured learning approach is used to get the predictions of the edges.



Patch extracted from the input image w.r.t the target pixel [5]

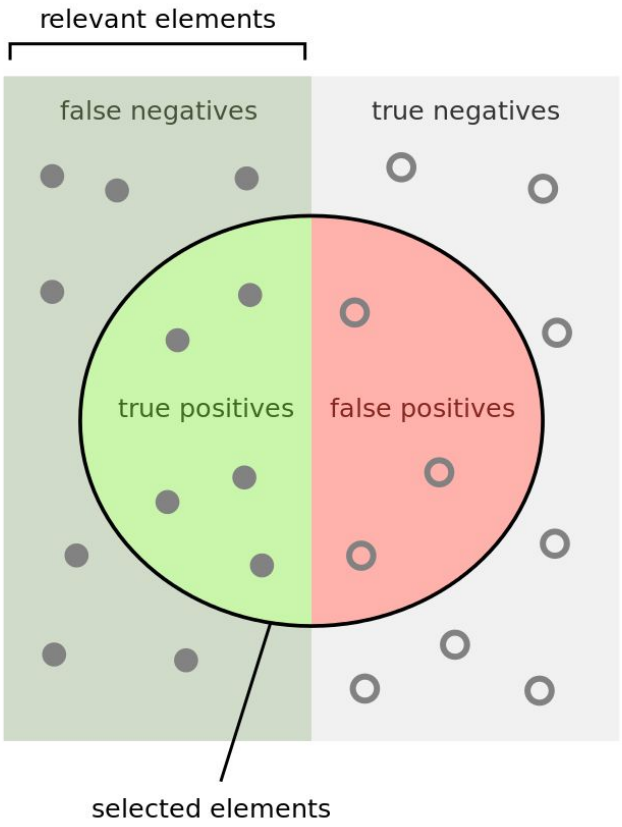


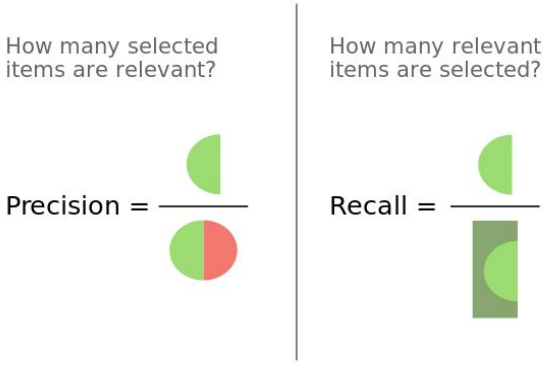
FLOWCHART OF THE STRUCTURED EDGE DETECTOR

The random forest is trained by taking a patch from the training images and another patch from one of the ground truths which are the actual edges in from the original image and performing computations to train the model as to whether a certain edge in the input image should be classified as edge or not. Once the training procedure is repeated for multiple images, the model can be used for predicted the actual set of images whose edges are to be determined. In the testing phase, the segmented outputs and the test images are passed as inputs to the random forest classifier.

The random forest classifier works on the principle of divide and conquer to attain the optimum speed and performance. The smallest and main unit of the random forest classifier is the decision tree and the decision tree is used to classify any input into categories or to extract the edges from the image by using the features of the images as the determining factors. These decision trees are slow learners and can be trusted completely and multiple trees are used to predict the outputs and hence the average of the entire forest is used to predict the final output.

Performance evaluation:





In order to evaluate the performance of the structured edge detector in comparison to the other edge detectors, we have used the Matlab source code available at [7] and various changes were made to compute the Precision and Recall values. The edge map of the input image was generated and the ground truths were loaded into the Matlab function. These values were then passed to the Matlab function. The binary edge map was generated by thresholding the images by choosing an appropriate threshold and other parameters like sharpness, nms, trees etc such that the maximum number of edges in the image could be obtained. The edge map was then

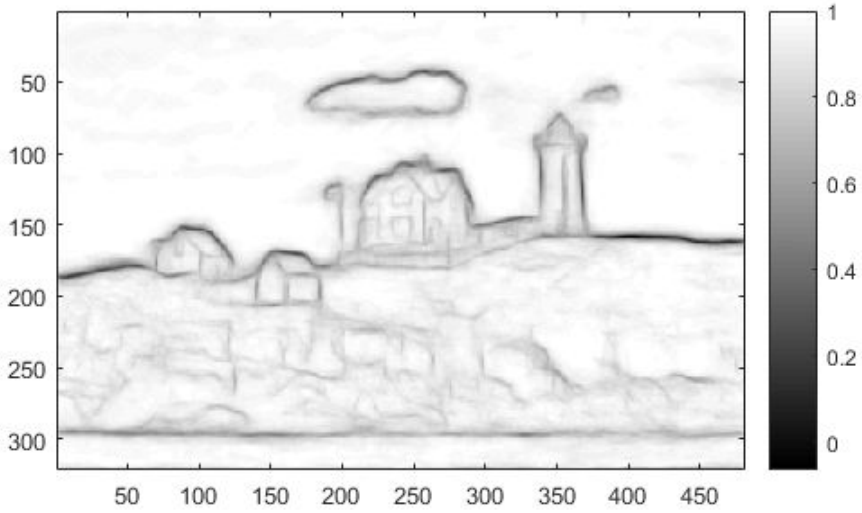
passed to the random forest classifier which generates the true positive, false positive, true negative, false negative values from which the Recall, Precision and F score values of the images are calculated by using the definitions given above.

The F-score is defined as the harmonic mean of the precision and recall and is given by the following formula :

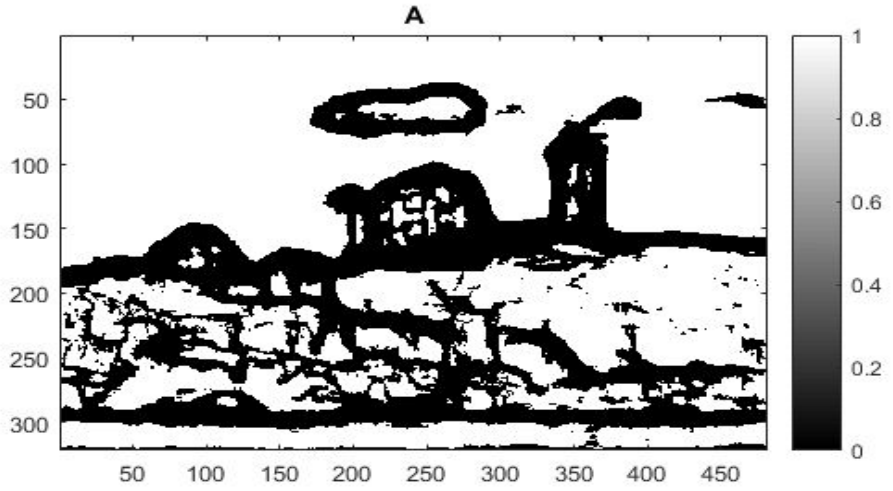
$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

A high precision value says that the prediction of an edge or contour in an image is actually correct. A high recall values conveys that most predictions made about the edges in the image is accurate. When you have a high F-score it says that the performance of the model is pretty good and hence the predictions are close to perfect accuracy.

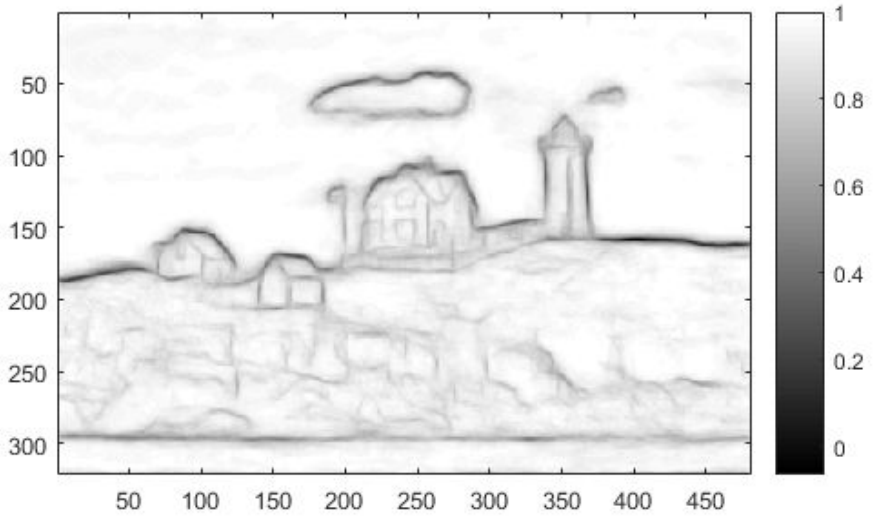
EXPERIMENTAL RESULTS (STRUCTURED EDGE AND PERFORMANCE EVALUATION):



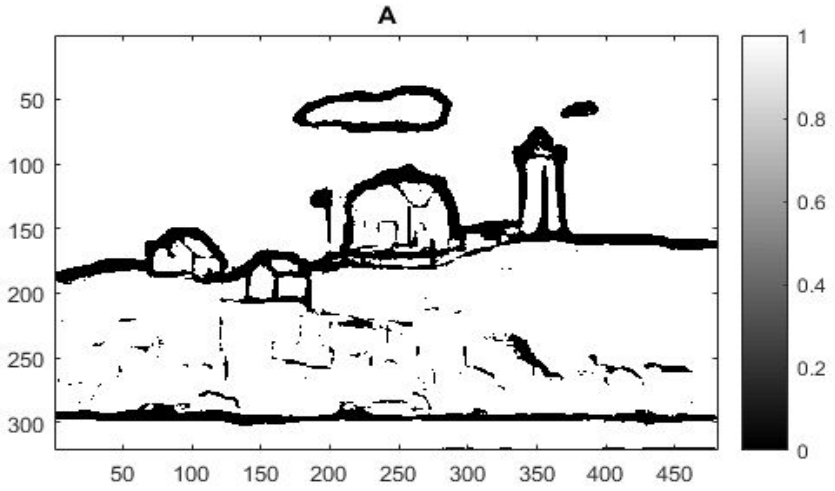
Probability Edge map SE Detector - House.jpg : threshold (0.07), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



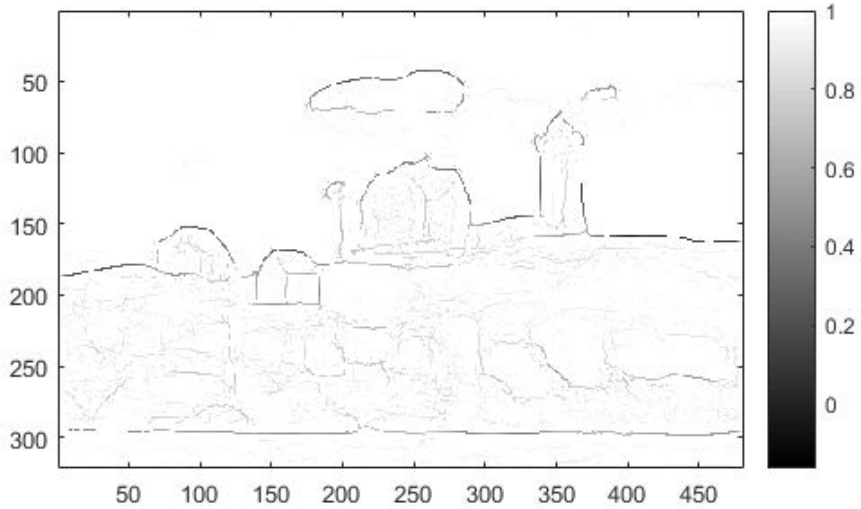
Binary Edge Map SE Detector: House.jpg: threshold (0.07), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



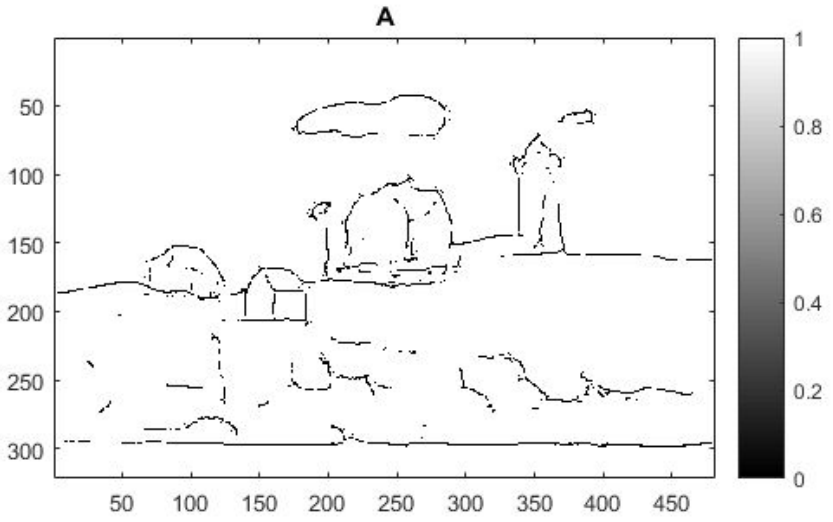
Probability Edge Map SE Detector: House.jpg: threshold (0.17), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



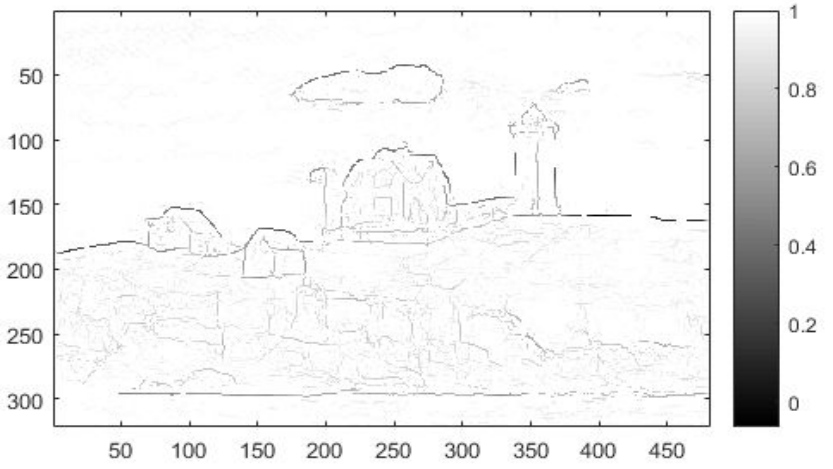
Binary Edge Map SE Detector: House.jpg: threshold (0.17), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



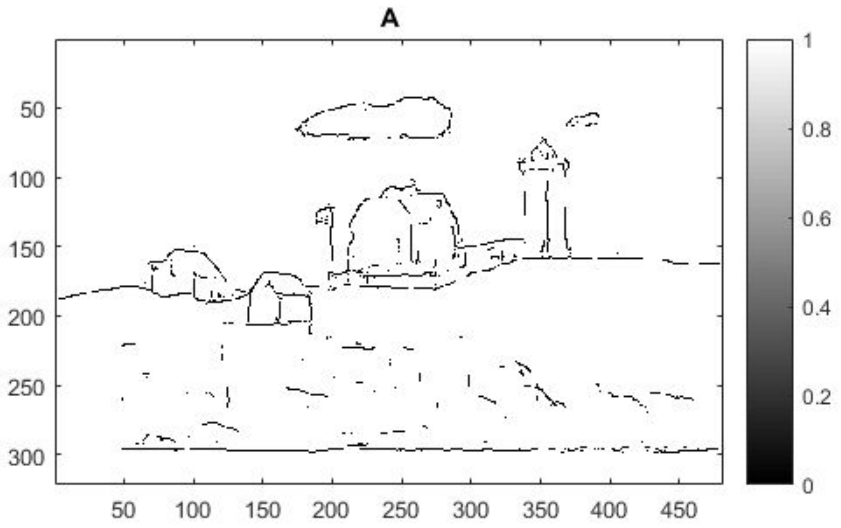
Probability Edge Map SE Detector: House.jpg: threshold (0.17), multiscale (0), sharpen (0), Trees (1), threads (4), nms (1)



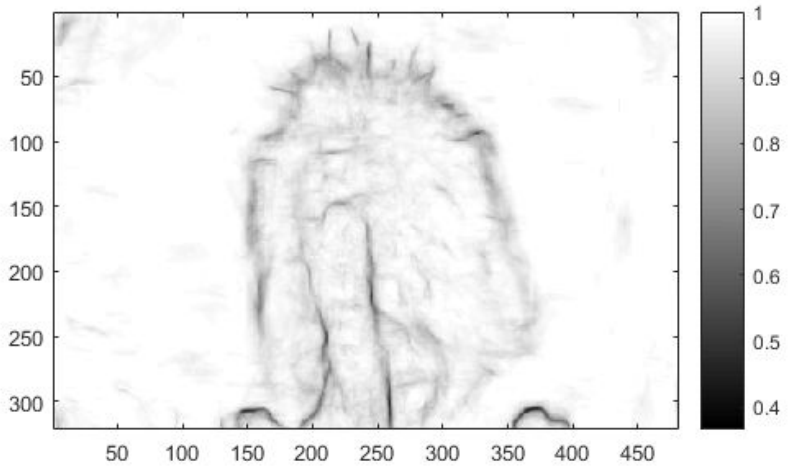
Binary Edge Map SE Detector: House.jpg: threshold (0.17), multiscale (0), sharpen (0), Trees (1), threads (4), nms (1)



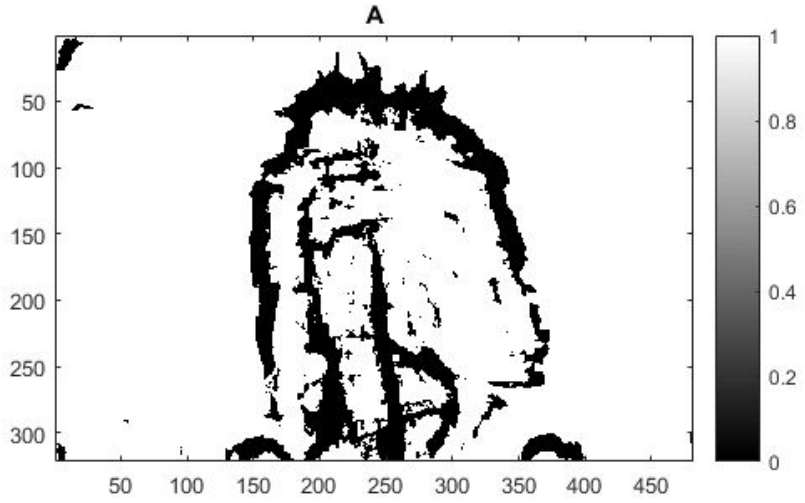
Probability Edge Map SE Detector: House.jpg: threshold (0.2), multiscale (1), sharpen (0), Trees (1), threads (4), nms (1)



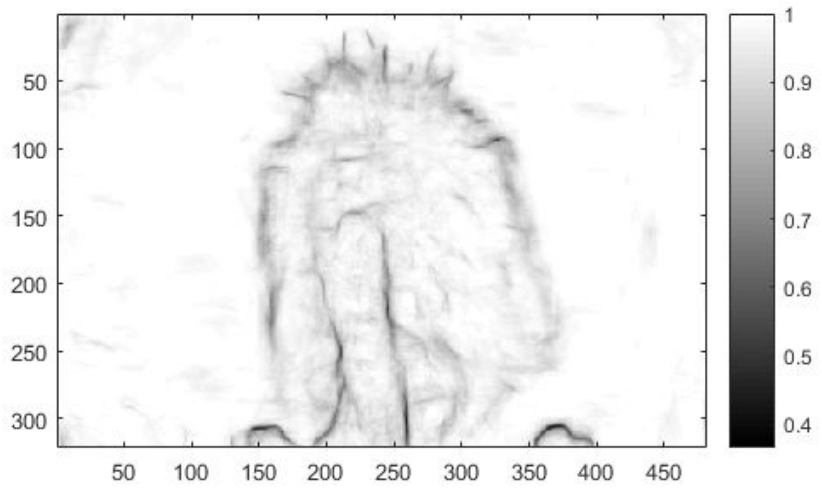
Binary Edge Map SE Detector: House.jpg: threshold (0.2), multiscale (1), sharpen (0), Trees (1), threads (4), nms (1)



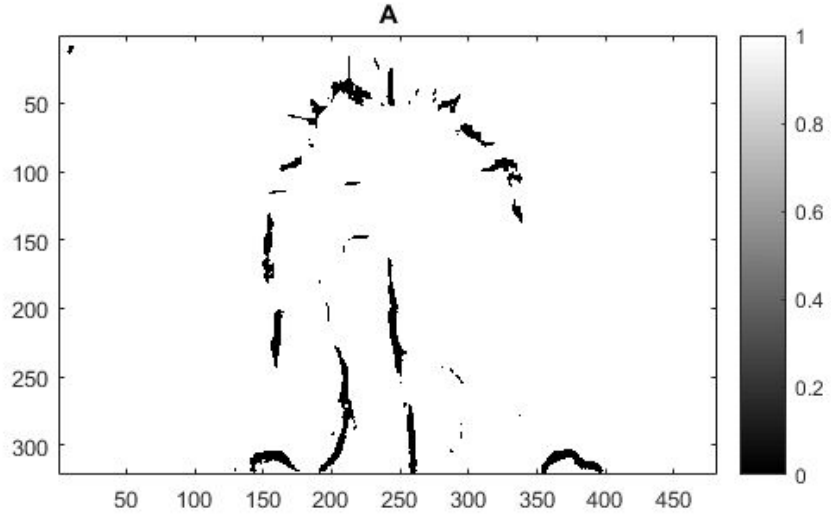
Probability Edge map SE Detector- Animal.jpg : threshold (0.07), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



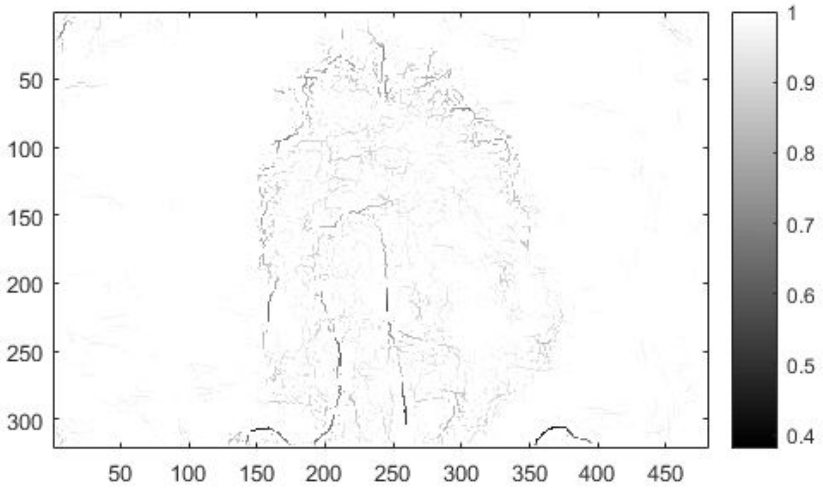
Binary Edge map SE Detector- Animal.jpg : threshold (0.07), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



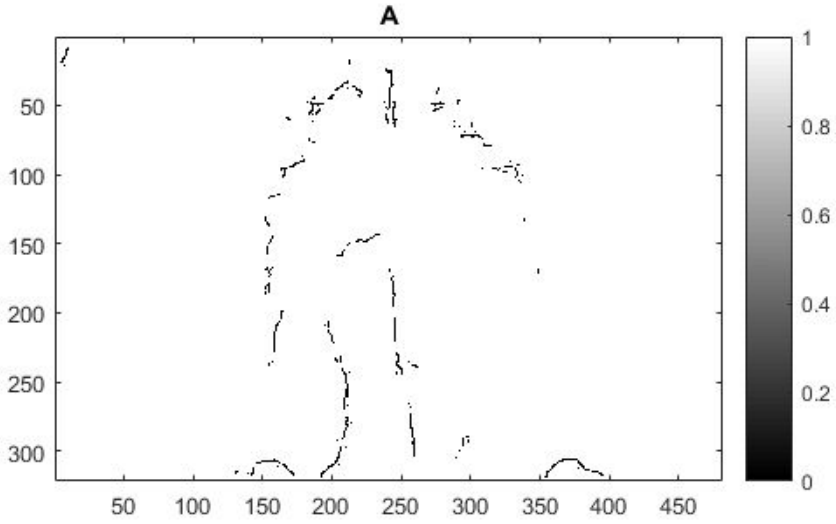
Probability Edge Map SE Detector: Animal.jpg: threshold (0.17), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



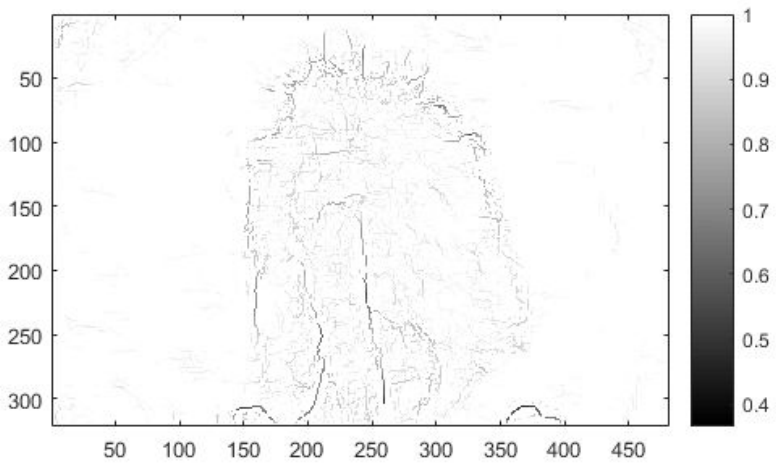
Binary Edge Map SE Detector: Animal.jpg: threshold (0.17), multiscale (1), sharpen (0), Trees (1), threads (4), nms (0)



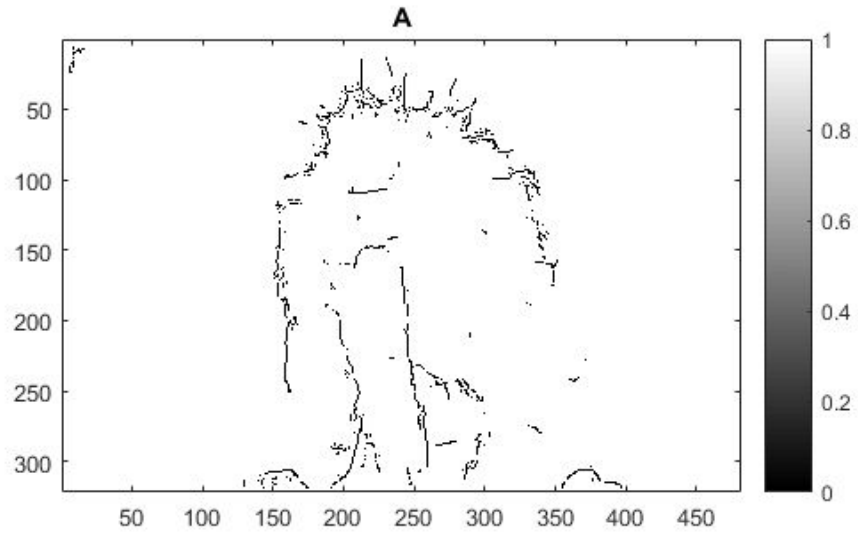
Probability Edge Map Detector: Animal.jpg: threshold (0.17), multiscale (0), sharpen (0), Trees (1), threads (4), nms (1)



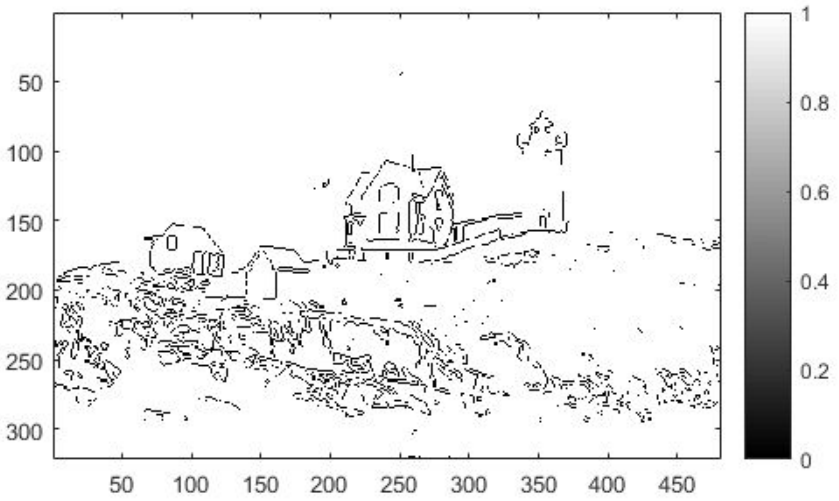
Binary Edge Map SE Detector: Animal.jpg: threshold (0.17), multiscale (0), sharpen (0), Trees (1), threads (4), nms (1)



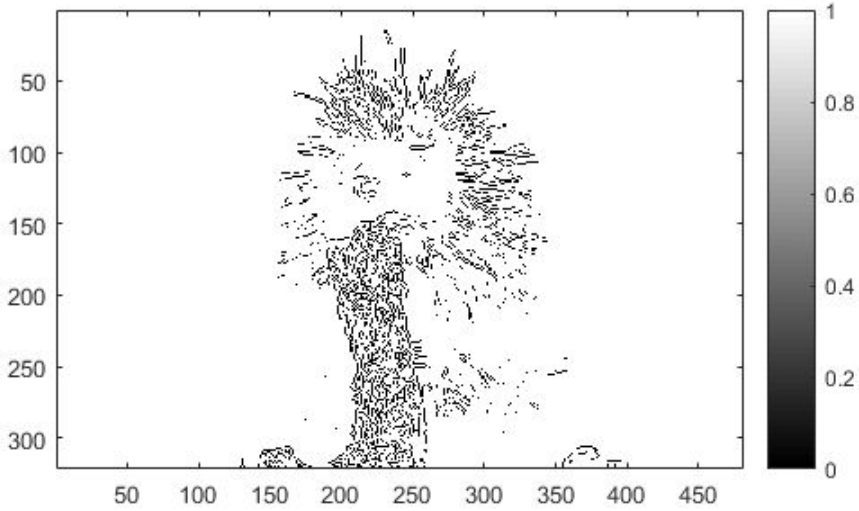
Probability Edge Map SE Detector: Animal.jpg: threshold (0.12), multiscale (1), sharpen (0), Trees (1), threads (4), nms (1)



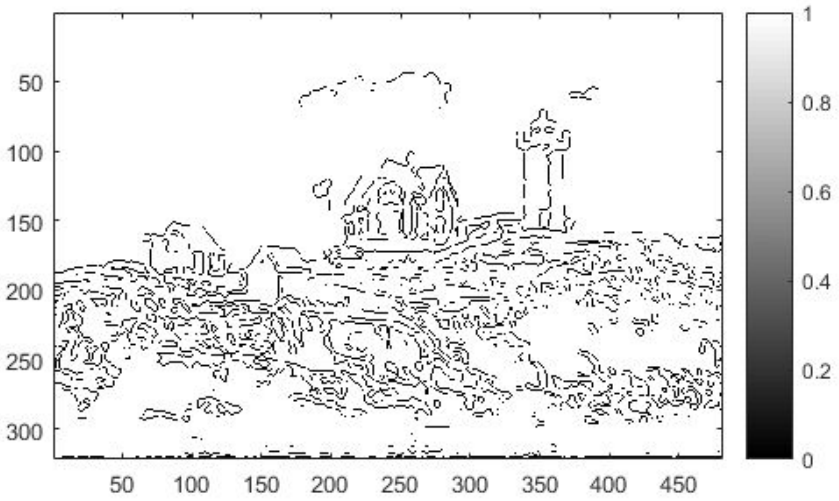
Binary Edge Map SE Detector: Animal.jpg: threshold (0.12), multiscale (1), sharpen (0), Trees (1), threads (4), nms (1)



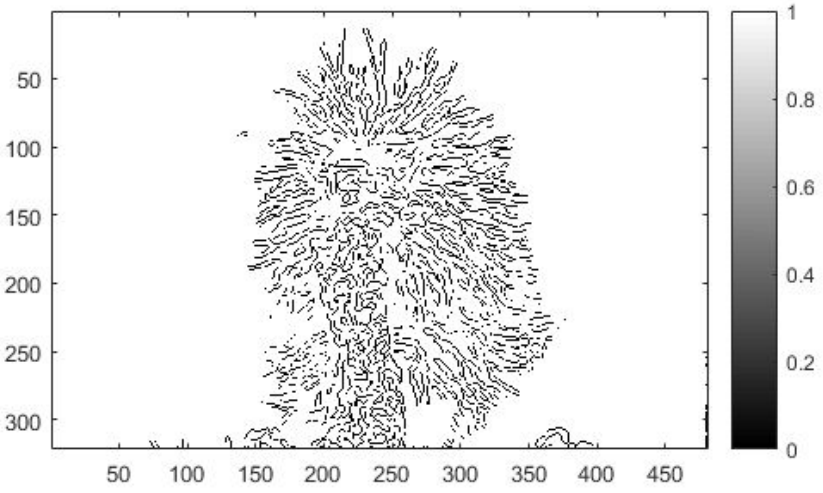
Edge map for Sobel: House.jpg



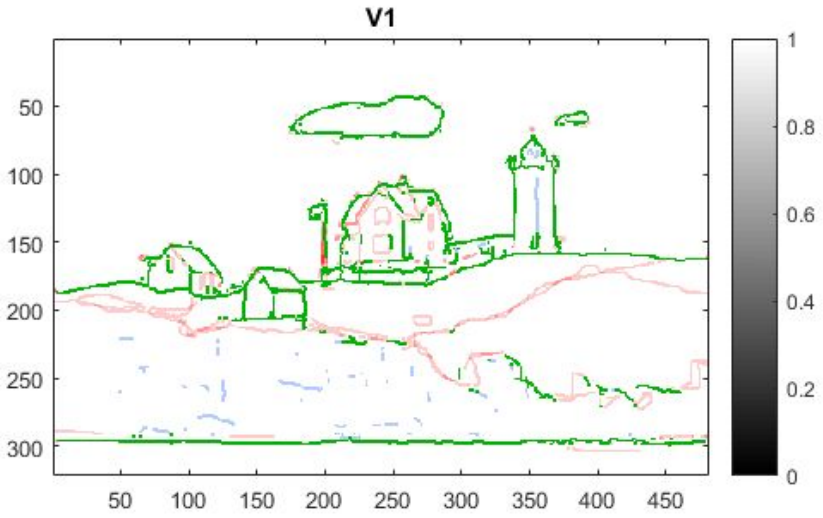
Edge map for Sobel: Animal.jpg



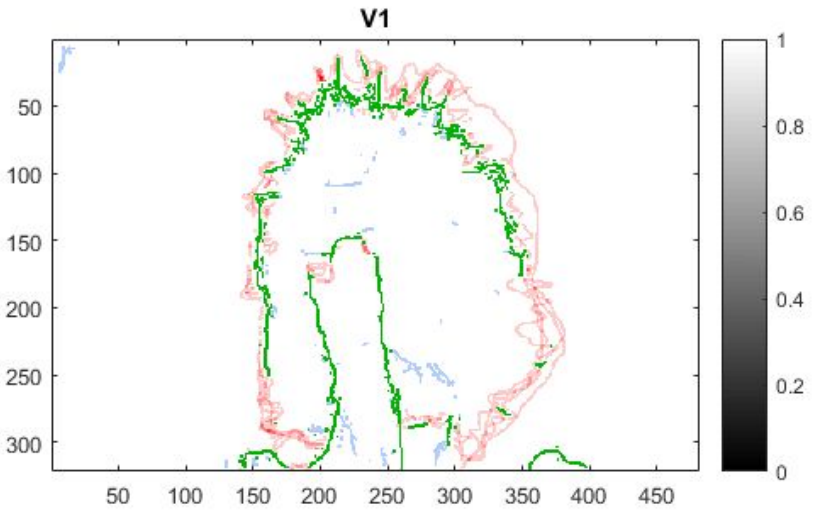
Edge map for Zero Crossing: House.jpg



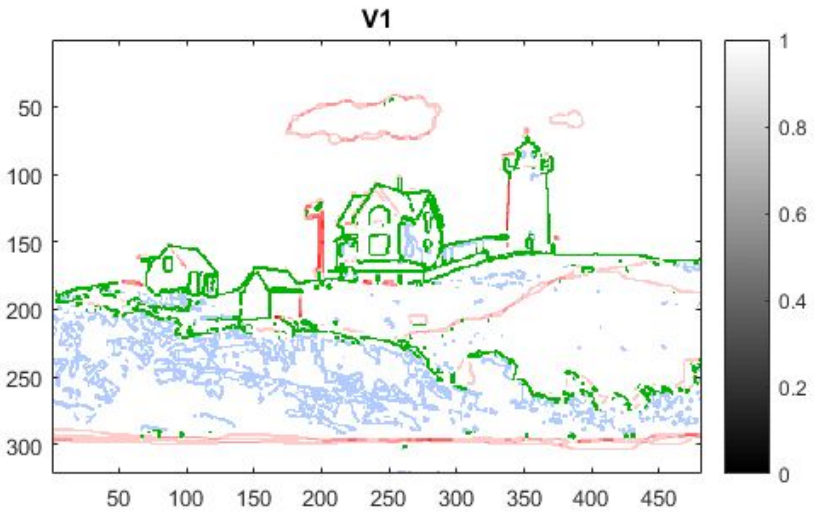
Edge map for Zero Crossing: Animal.jpg



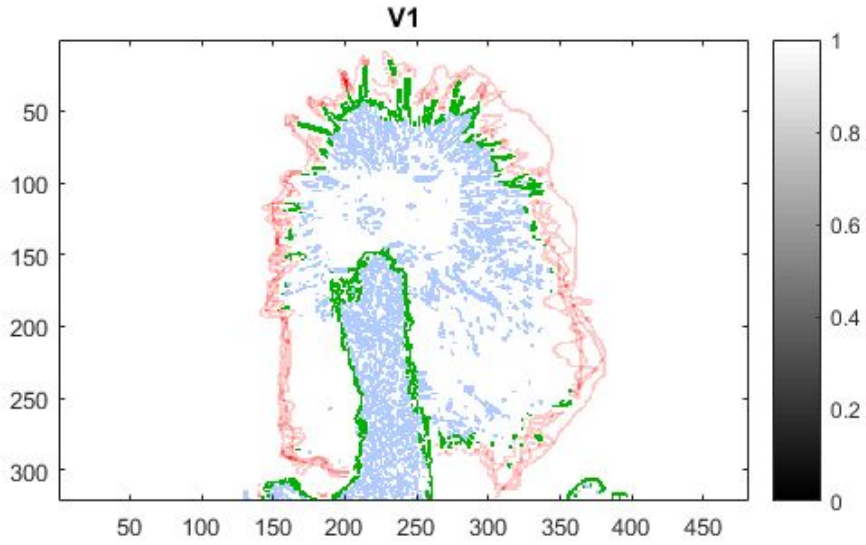
Visualization map for SE Detector: House.jpg



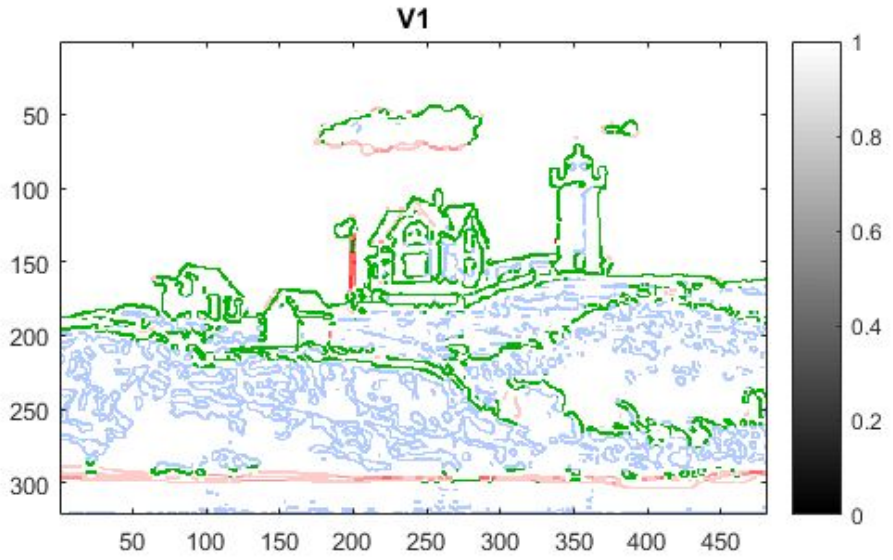
Visualization map for SE Detector: Animal.jpg



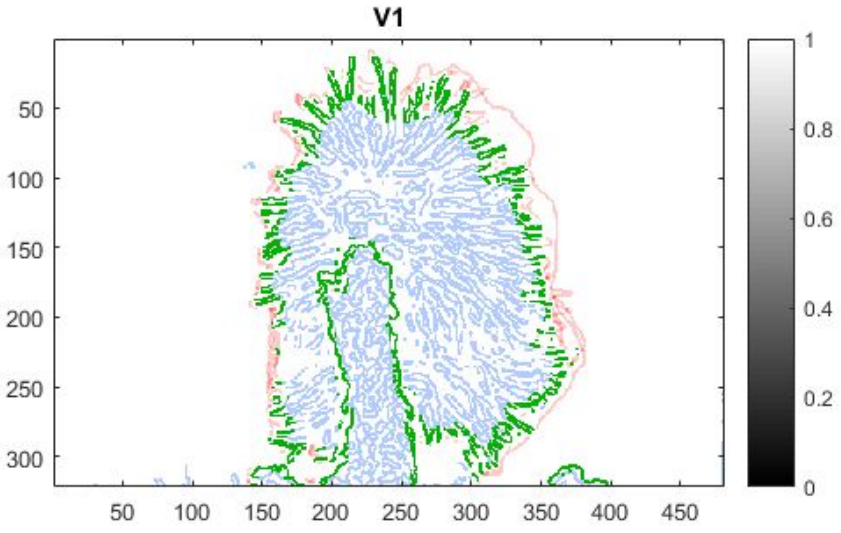
Visualization map for Sobel Detector: House.jpg



Visualization map for Sobel Detector: Animal.jpg



Visualization map for Zero Crossing Detector: House.jpg



Visualization map for Zero Crossing Detector: Animal.jpg

Red lines indicate false negatives, blue edges indicate false positives and true positives are represented by green colored edges in the visualization map of SE Detector, Sobel Detector and Zero Crossing Detector.

Tables for Precision, Recall, F-Score and Threshold:

(i)Structured Edge - House.jpg

Ground truth	Threshold	Mean precision	Recall	F-score
Ground truth 1	0.26	0.7369	0.8757	0.8003
Ground truth 2	0.23	0.7780	0.6758	0.7235
Ground truth 3	0.25	0.8045	0.7101	0.7544
Ground truth 4	0.22	0.6989	0.7454	0.7214
Ground truth 5	0.19	0.5683	0.7534	0.6479
Mean :	0.2	0.8878	0.7878	0.8348

(ii)Structured Edge - Animal.jpg

Ground truth	Threshold	Mean precision	Recall	F-score
Ground truth 1	0.13	0.6422	0.6276	0.6348
Ground truth 2	0.12	0.6337	0.6075	0.6203
Ground truth 3	0.16	0.5023	0.3971	0.4436
Ground truth 4	0.13	0.7395	0.6473	0.6902
Ground truth 5	0.13	0.6099	0.5551	0.5812
Mean :	0.12	0.7656	0.6069	0.6770

(iii)Sobel Edge detection - House.jpg

Ground truth	Threshold	Mean precision	Recall	F-score
Ground truth 1	0.01	0.1626	0.6153	0.2572

Ground truth 2	0.01	0.2356	0.5486	0.3297
Ground truth 3	0.01	0.2037	0.5425	0.2962
Ground truth 4	0.02	0.2529	0.6885	0.3699
Ground truth 5	0.01	0.3030	0.8617	0.4484
Mean :	0.68	0.8879	0.6475	0.4961

(iv)Sobel Edge detection - Animal.jpg

Ground truth	**Threshold**	**Mean precision**	**Recall**	**F-score**
Ground truth 1	0.01	0.1467	0.5449	0.2312
Ground truth 2	0.01	0.1690	0.5377	0.2571
Ground truth 3	0.01	0.09	0.4094	0.1491
Ground truth 4	0.01	0.1890	0.6287	0.2907
Ground truth 5	0.01	0.1512	0.5230	0.2346
Mean :	0.01	0.2428	0.5353	0.3340

(v)LoG Edge detection - House.jpg

Ground truth	**Threshold**	**Mean precision**	**Recall**	**F-score**
Ground truth 1	0.02	0.1880	0.5892	0.2851
Ground truth 2	0.01	0.2592	0.4997	0.3414
Ground truth 3	0.01	0.2436	0.5372	0.3351
Ground truth 4	0.02	0.2815	0.6350	0.3902
Ground truth 5	0.01	0.3417	0.8045	0.4797
Mean :	0.005	0.4400	0.6023	0.5107

(vi)LoG Edge detection - Animal.jpg

Ground truth	Threshold	Mean precision	Recall	F-score
Ground truth 1	0.01	0.1616	0.9007	0.2738
Ground truth 2	0.02	0.1782	0.8532	0.2952
Ground truth 3	0.01	0.0929	0.6209	0.1602
Ground truth 4	0.01	0.1910	0.9544	0.3182
Ground truth 5	0.01	0.1594	0.8283	0.2674
Mean :	0.55	0.2679	0.8419	0.4060

DISCUSSION

The chosen parameters for the Structured Edge Detection were:
Multiscale: 1 (true for top accuracy)
Sharpen: 0 (for top speed)
Trees:1 (for top speed)
Threads:4 (for top speed)
Nms: 1 (for proper edges)
'nms' stands for non-maximal suppression which is a post processing technique to improve the Edge Detector Output.

Sobel Detector outputs have lots of noisy points and also many false positives(blue colored edges) and false negatives(red colored edges) in the visualization map. It is susceptible to noise and does not give an ideal output for edge detection.

Zero Crossing Detector also is susceptible to noise but it works better than Sobel as it involves Laplacian of Gaussian filter for detecting the edges. The visualization map contains less number of false negatives and false positives compared to Sobel.

SE Detector has a much better output than Sobel and Zero Crossing because it is not sensitive to noise and works on Random Forest principle. The f-score of SE Detector is higher than the other two edge detectors. F-score is the measure of how close is the edge detector output to the ground truths. Hence, higher the f-score, better is the detector.

The precision, recall and f-scores of all the detectors for house.jpg and animal.jpg image have been tabulated above. From the tables, it can be observed that the F-measure is high for House.jpg image as it has edges which can be easily differentiated which is not the case with animal.jpg.

F-measure is used to combine all the evaluation parameters into one so that it is easy to compute how good a model performs. Higher f-score hence implies a better edge detector which in our case is Structured Edge Detector out of the three (Sobel, Zero Crossing and SE).

There should be a trade-off between precision and recall to get a maximum f-score as it is a harmonic mean of the previous parameters. Setting one parameter very high and the other two low will result in a low f-score. To prove that precision and recall should be the same:

- Consider C = P+R ----------(1)
- Where C is a Constant, P is Precision, R is Recall and F is F-score
- F = (2*P*R)/(P+R)
- Replace P with C-R,
- F = (2*(C-R)*R)/(C)
- Differentiate w.r.t., R to get max f-score,
- 4*R/C = 2
- R = C/2 form the first equation, R=P=C/2

Hence, for maximum f-score, precision should be equal to recall which will return a better edge detected output.

PROBLEM 3: SALIENT POINT DESCRIPTORS AND IMAGE MATCHING

MOTIVATION AND ABSTRACT

Salient point descriptors or interest point detectors are used to find interest points in images in order to use the points for future processing. Salient points are used in various applications of computer vision like image matching and object detection primarily based on the view. The criteria to keep in mind while performing salient point detection are that the points should be invariant to scale, the areas surrounding the point should be rich in local information and should be stable under local and global perturbations like variations in the characteristics of the image like brightness, contrast etc.

There are various applications in which image matching is required such as object recognition, fingerprint recognition, comparing faces to check for criminal records etc. There are methods using which image matching is performed. The popular methods of feature based image matching are SIFT, SURF and ORB. These algorithms have been designed such that they are invariant to different kinds of image transformations such as scaling, rotation, affine transforms, etc.

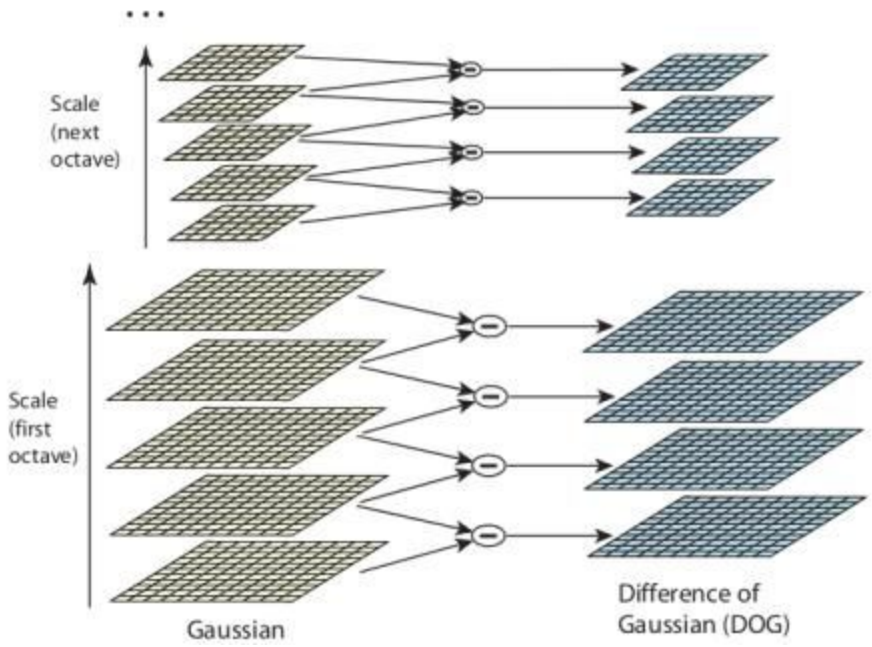
SIFT

SIFT or Scale Invariant Feature Transform is an algorithm presented by Lowe to detect and extract the features of an image which are invariant to scaling and rotation. The SIFT algorithm consists of 4 main steps: Scale Space Extrema Detection, Keypoint Localization, Orientation Assignment and Description Generation.

a) Scale space extrema detection:

The main intuition behind this method is that a constant sized window cannot be used to detected features and edges of different shapes and sizes. Hence space filtering is performed to extract features which are invariant to scale. This is performed using the Difference of Gaussians method where the Gaussian filters of standard deviation σ and $k\sigma$ are applied to an image and difference of the outputs is obtained. The DoG method is an approximation of the Laplacian of Gaussian method. This process is repeated for different levels of octaves of the image in the Gaussian pyramid as shown in the figure below.

[3]



Pyramid of Difference of Gaussian filters

For each octave, the size of the images will keep decreasing. Once the DoG is obtained, we search for extreme values in a local region over scale and space. Each pixel is compared with 8 other neighbours in the same image as well as images in other octaves, if it is found to be a minimum, then it is chosen as a potential keypoint.

b) Keypoint localization

Once the potential keypoints are obtained, they are fine-tuned to get more accurate results by using the Taylor series expansion of scale space. If the intensity value is less than the extrema value obtained from the above method then, then the intensity value is discarded. This extrema value is called the contrast threshold in OpenCV terminology.

The DoG filter has been found to have a higher response to edges too, hence they have to removed too. For this purpose a 2x2 Hessian matrix (H) is used to calculate the curvature and

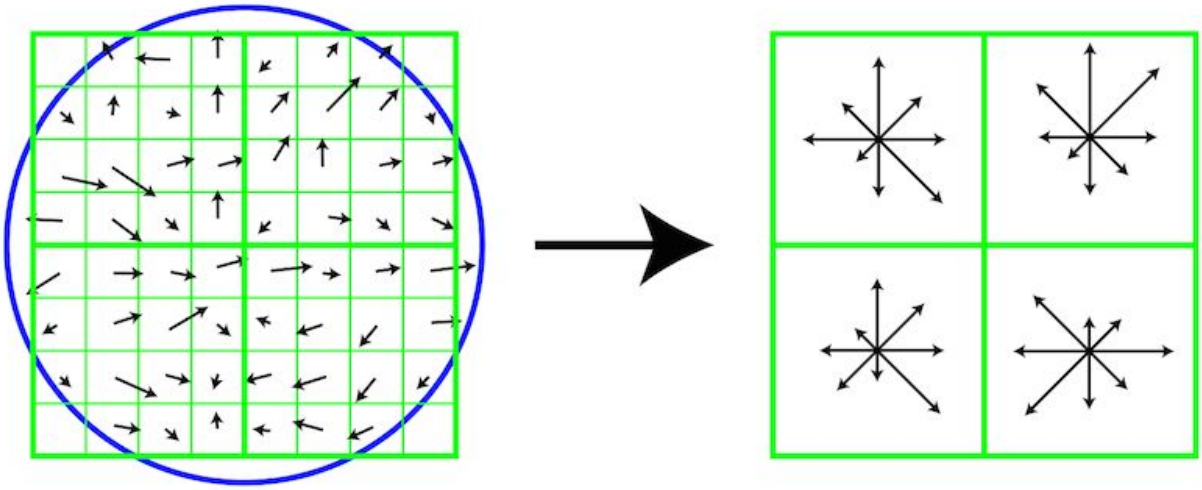
the ratio of eigenvalues is obtained. If the ratio is greater than the edge threshold, then the keypoint is discarded. This step finally eliminates all the key points which have low contrast and are edges to leave strong key points in the end.

c) Orientation Assignment

In this step an orientation is assigned to each keypoint in order to make the image invariant to rotation. A neighbourhood around the keypoint is taken and the gradient magnitude and direction are computed in that region. A orientation histogram with 36 bins covering 360 degrees is created. The highest peak in the histogram is chosen. Then the peaks which are greater than 80% of this peak are chosen to determine the orientation of the key points in the image.

d) Keypoint descriptor

In this step a keypoint descriptor has to be created for every point in the image. This is accomplished by taking a 16x16 neighbourhood around the keypoints. This neighbourhood is divided into 16 sub-blocks, each having a size of 4x4. This results in 128 bin values which are rearranged in the form of a vector to form a keypoint descriptor.



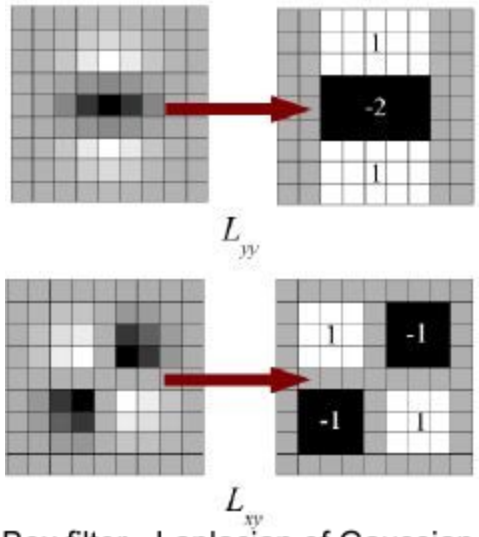
e) Keypoint Matching

Key Points in two images are matched by using the nearest neighbours algorithm. But in some cases the key points can have multiple neighbours very close to them. In that case the ratio of the distances is taken to eliminate such ambiguous cases.

SURF

SURF or Speeded-Up Robust Features, is a local feature detector and descriptor algorithm, which is used extensively in computer vision in order to perform tasks like image registration, 3D reconstruction etc. SURF follows the same principles as SIFT except for some changes in the substeps. SURF algorithm has 3 main steps - interest point detection, local neighborhood description and matching.

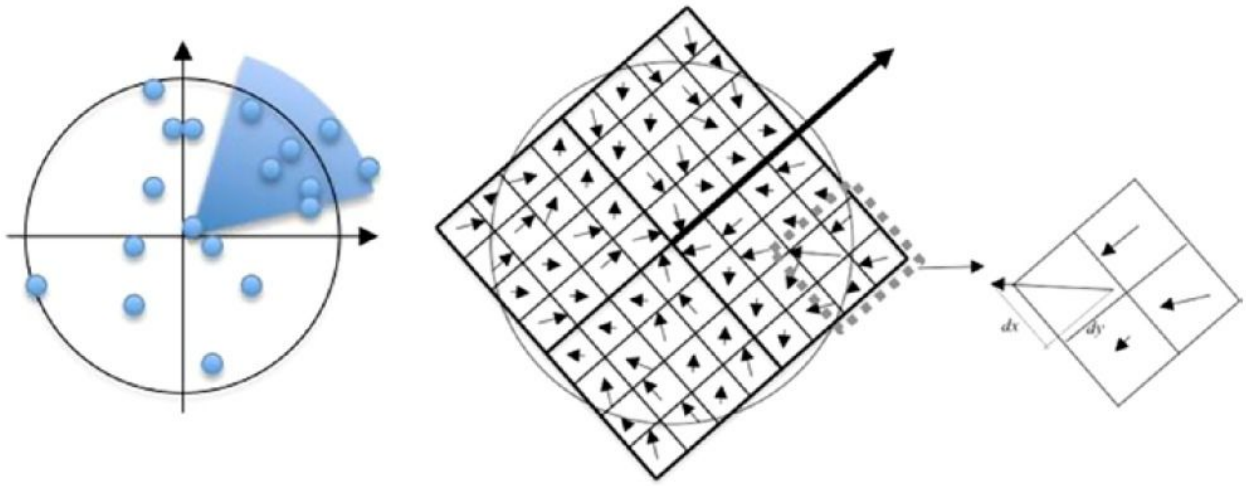
In the SURF algorithm the Laplacian of Gaussian filter is approximated using the Box filter which is then used to obtain the potential keypoints. The determinant of the Hessian matrix is obtained. The LoG filter is used as the derivative in the Hessian matrix. From the resultant integral image, we obtain the sum of the intensities of the squares present in the Hessian box filters. By varying the filter size, we are controlling the number of interest points to be detected.



Box filter - Laplacian of Gaussian filter

The second step of the SURF detector involves finding the major interest points in the image, using the non-maximal suppression filter in scales, above and below the current image scale. We will need to use interpolation to get the interest points in the correct scale, by comparing with the neighbouring scales.

The next step involves extraction of the feature direction. This is done by using the Haar Transform. We take a six range radius and obtain all the pixels in that range which is six times that of our scale space. The X and Y Haar transforms are calculated for all the pixels. The direction which has the maximum weight will be assigned as the direction of that feature.



SURF descriptor with the Haar transforms

From the Haar wavelet responses obtained in the X and Y directions - dx and dy respectively, we obtained in the previous step. This is obtained for the 16 sub-regions and then we have to calculate the sum of the dx, dy and their absolute values. These values are calculated in the direction of rotation to give the SURF descriptors.

There are many feature matching and image matching algorithms which have been defined in OpenCV and we use two such algorithms to perform feature matching between images.

Brute Force Matcher

While using the brute force matcher and comparing two images, each descriptor in the first set is compared with every other descriptor in the second set one after another. The descriptor with the smallest distance is identified as the closest match.

FLANN Matcher

We use the FLANN matcher in order to match the two images which are being compared. FLANN stands for Fast Library for Approximate Nearest Neighbors. Its is a library used for performing the searching the nearest neighbours of an interest point in high dimensional spaces. This library has a collection of algorithms from which chooses the algorithm yielding the most optimal performance and the best output based the dataset passed as input. The FLANN matcher produces better results when using large datasets.

Procedure for finding the salient points using SIFT and SURF:

1) Load the two images for which the salient points are to be calculated into 2 matrices.
2) Create an object of the inbuilt SIFT detector, by passing the minHessian value as the threshold using which the required number of key points are calculated.
3) Use the detect() function to compute the key points in the image
4) Plot the key points using the drawKeypoints function.
5) Repeat the above steps for the SURF detector.

Procedure for Image Matching:

1) Load the two images which are to be matched into matrices to perform further processing
2) Create an object of the SIFT DETECTOR
3) Detect the keypoints using SIFT Detector by setting the minimum Hessian threshold value
4) Call the detectAndCompute() function to compute the keypoints and descriptors for both the images
5) Create an instance of the Flann matcher
6) Use the matches() function to find all the key points which match in both the images
7) Perform a calculation of max and min distances between keypoints
8) Using the minimum and max distances filter out the best matching keypoints
9) Draw only "good" matches using the drawMatches function
10) Show detected matches using the imshow() function
11) Repeat the same procedure as above for the SURF detector too.

EXPERIMENTAL RESULTS:

EXTRACTION AND DESCRIPTION OF SALIENT POINTS:

Salient points SIFT - Hessian threshold : 400 - Truck.jpg

Salient points SIFT- Hessian threshold : 400 - Truck.jpg

Salient points SURF- Hessian threshold : 400 - Truck.jpg

Salient points SURF- Hessian threshold : 600 - Truck.jpg

Salient points SIFT- Hessian threshold : 400 - Bumblebee.jpg

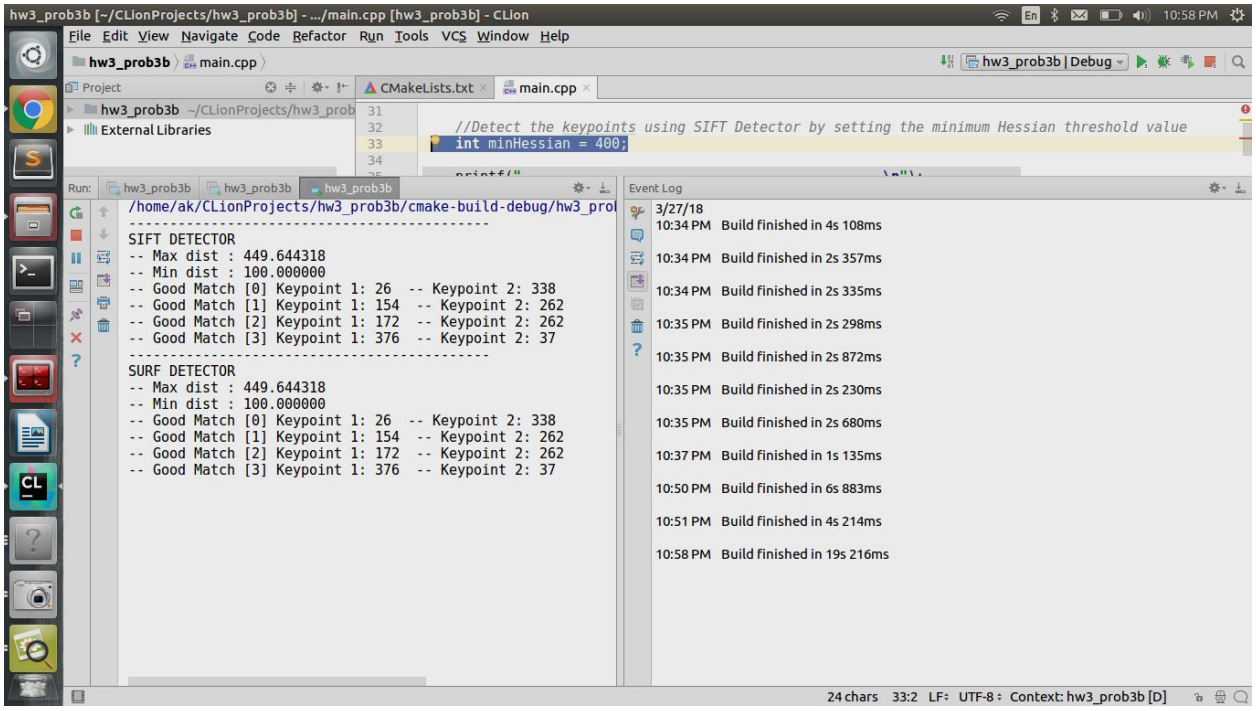
Salient points SIFT- Hessian threshold : 600 - Bumblebee.jpg

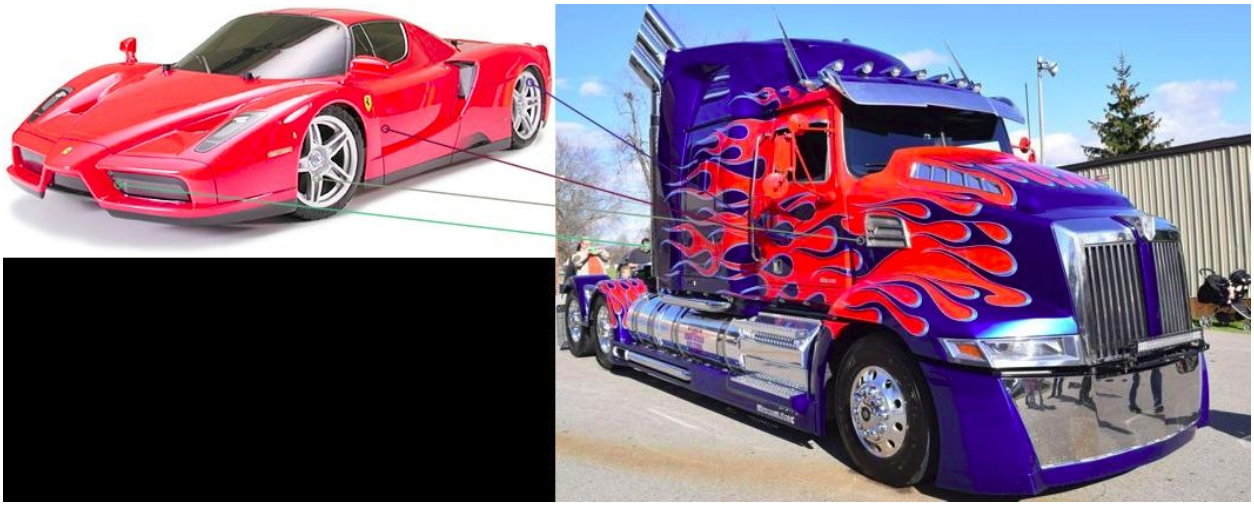
Salient points SURF- Hessian threshold : 400 - Bumblebee.jpg

Salient points SURF- Hessian threshold : 600 - Bumblebee.jpg

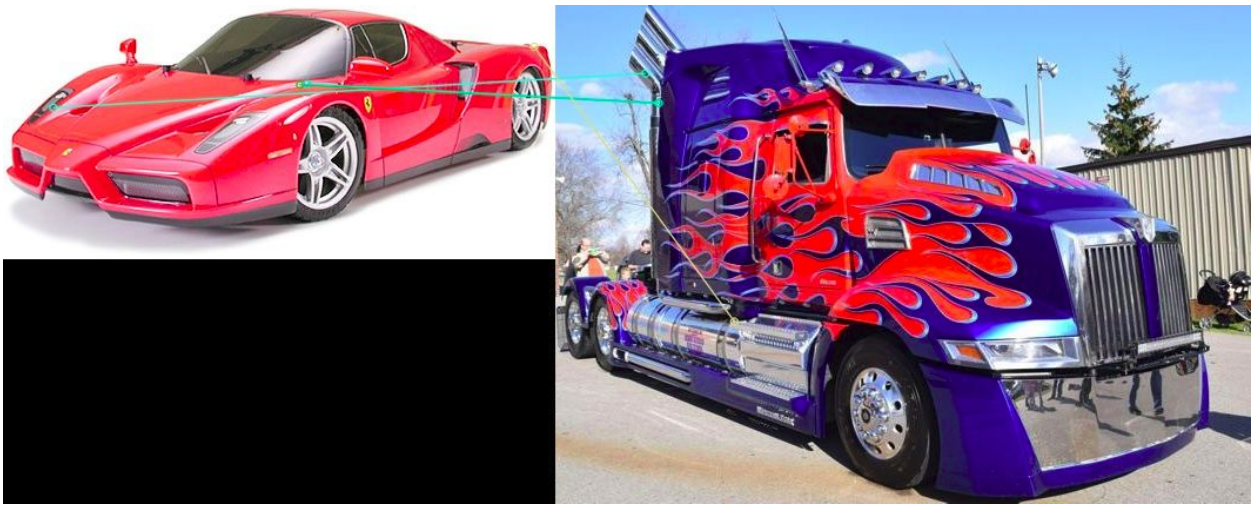
IMAGE MATCHING: FLANN MATCHER ALGORITHM

Images compared: Ferrari 1 and Optimus Prime truck
minHessian threshold: 400
Feature matcher: FLANN matcher



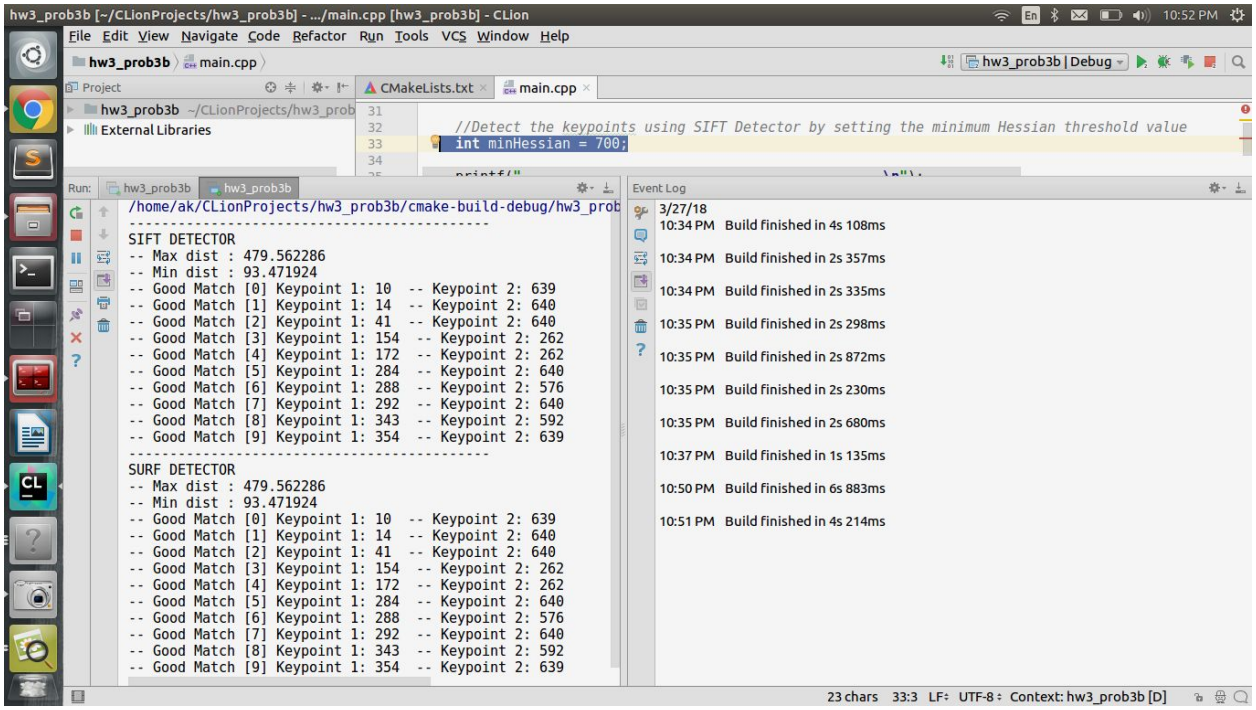


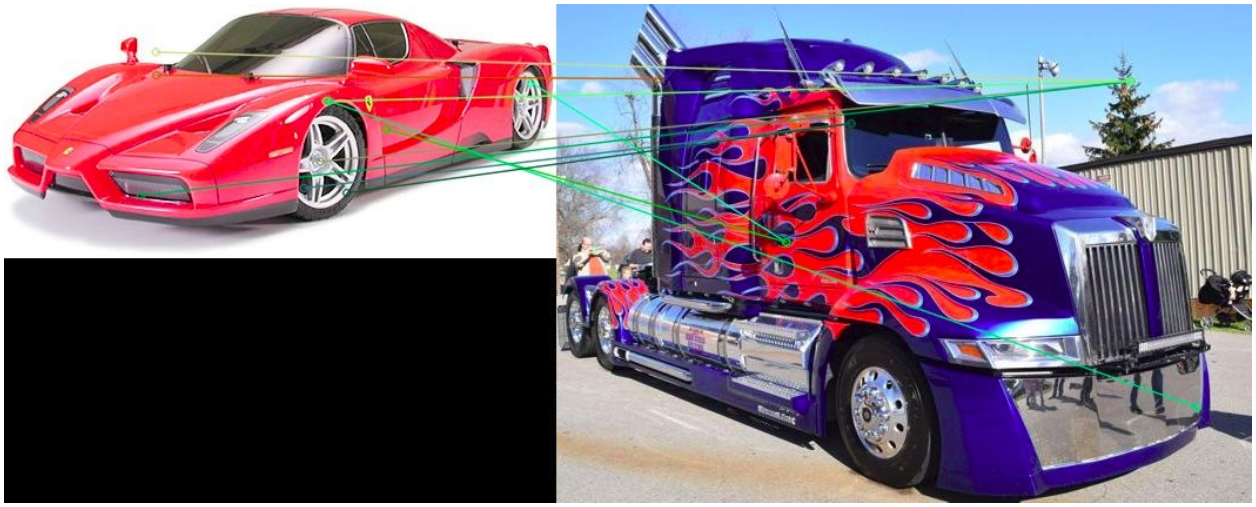
Ferrari_1 and Optimus Prime - SURF output



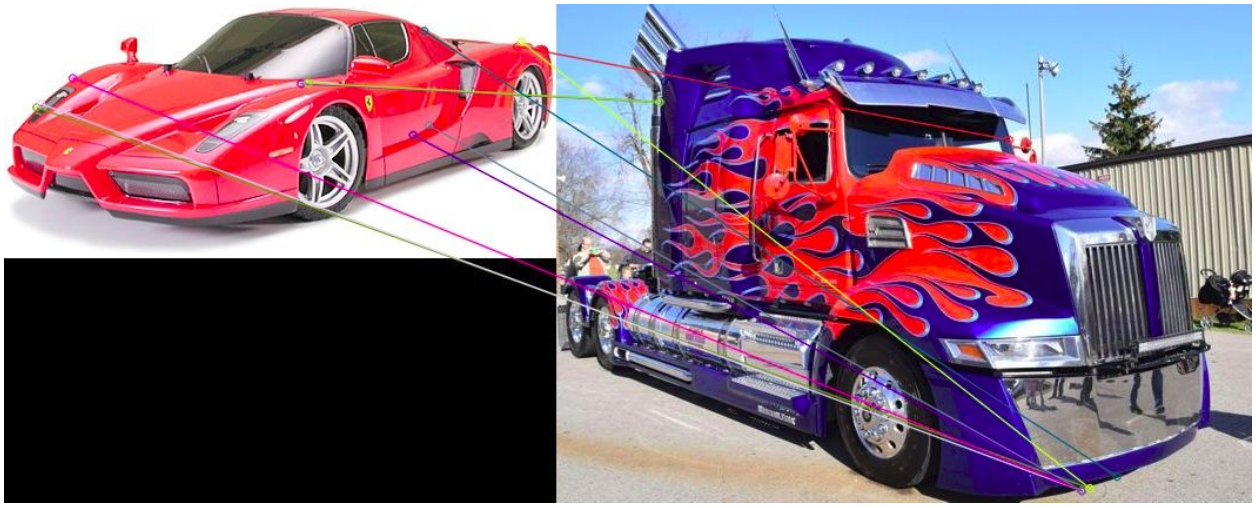
Ferrari_1 and Optimus Prime - SIFT output

Images compared: Ferrari 1 and Optimus Prime truck
minHessian threshold: 700
Feature matcher: FLANN matcher



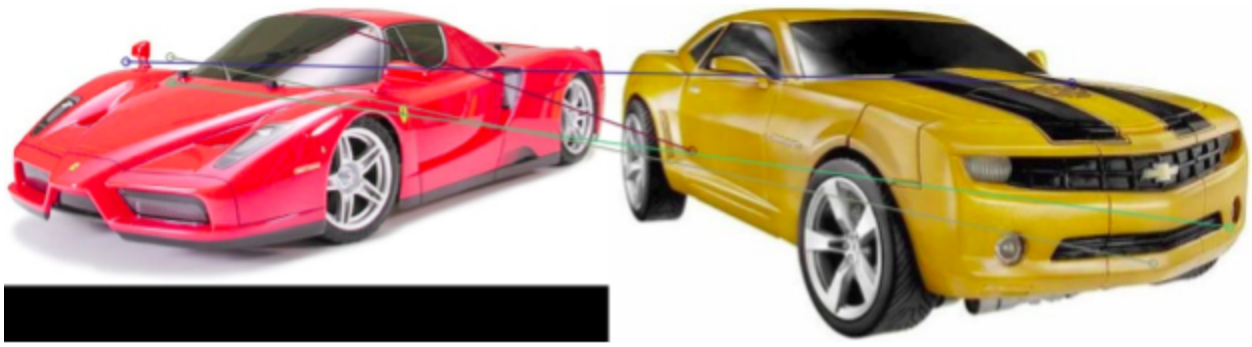


Ferrari_1 and Optimus Prime - SURF output

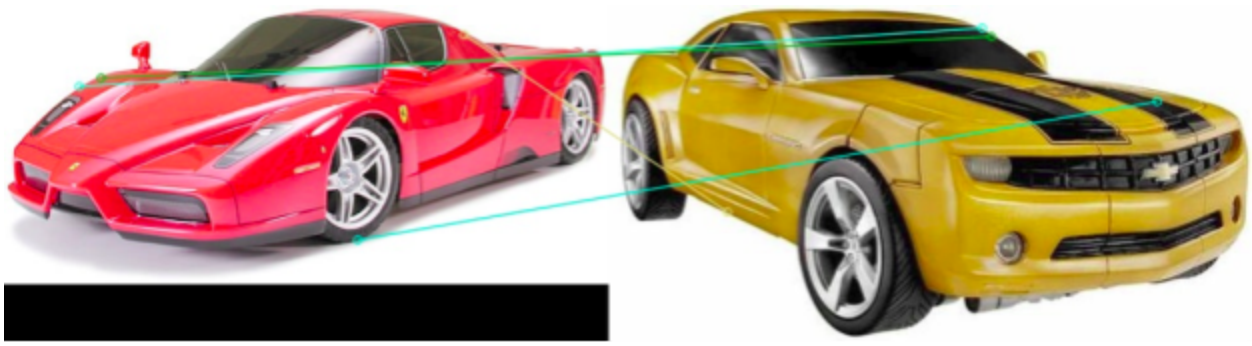


Ferrari_1 and Optimus Prime - SIFT output

Images compared : Ferrari_1 and Bumblebee
Hessian threshold: 400
Feature matcher: FLANN matcher

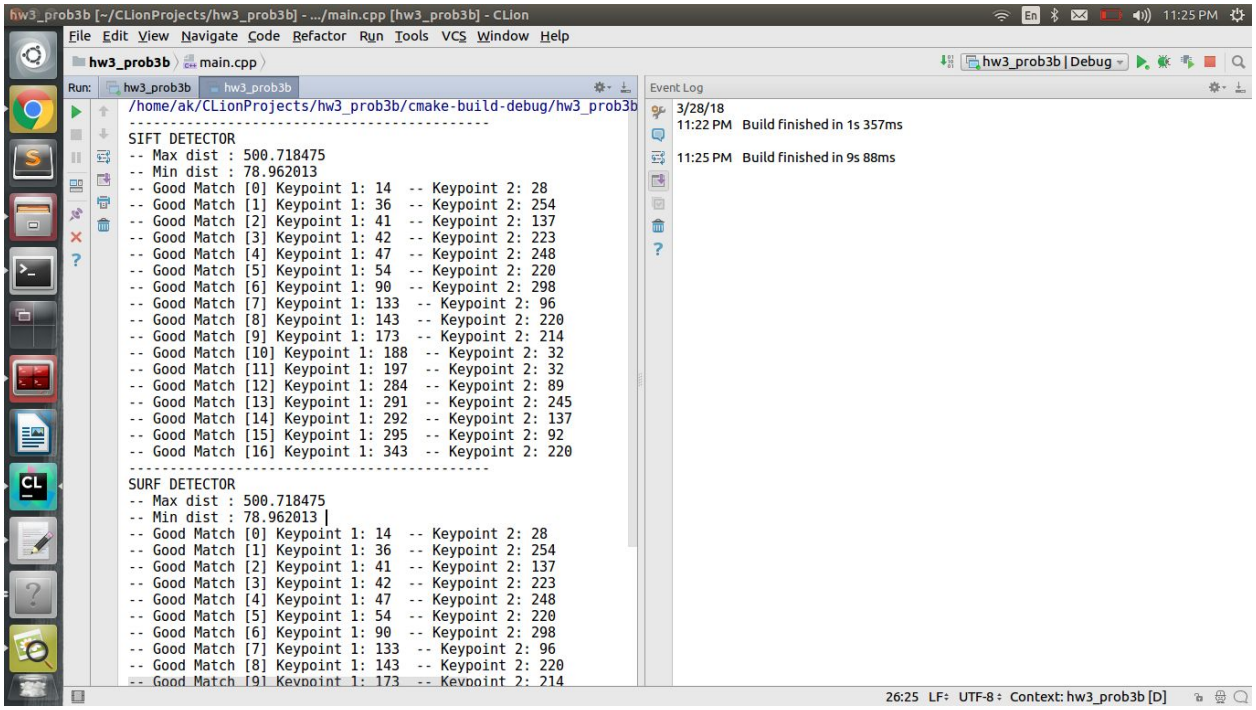


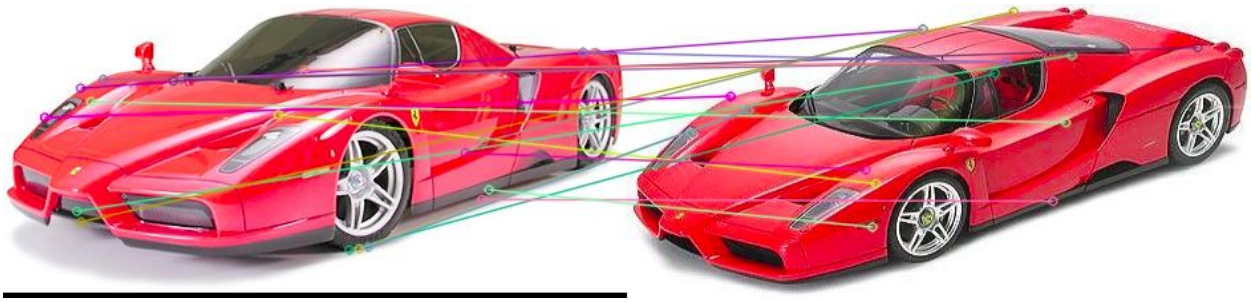
Ferrari_1 and Bumblebee - SURF output



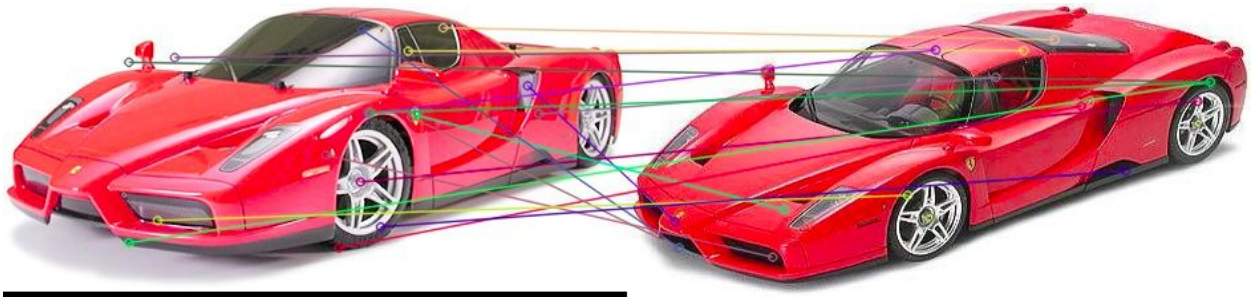
Ferrari_1 and Bumblebee - SIFT output

Images compared : Ferrari_1 and Bumblebee
Hessian threshold: 400
Feature matcher: FLANN matcher



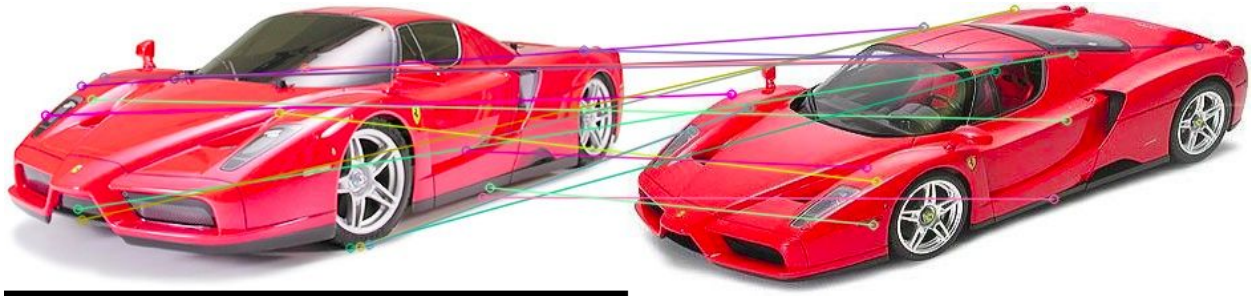


Ferrari 1 and Ferrari 2 - SIFT output

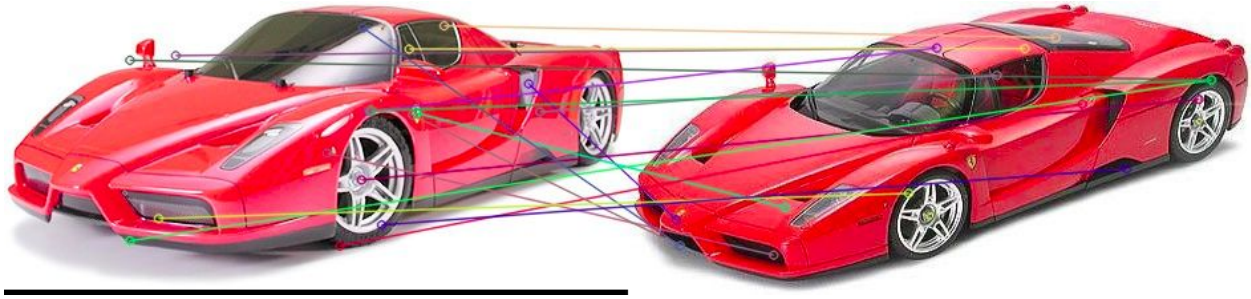


Ferrari 1 and Ferrari 2 - SURF output

Images compared : Ferrari_1 and Bumblebee
Hessian threshold: 700
Feature matcher: FLANN matcher

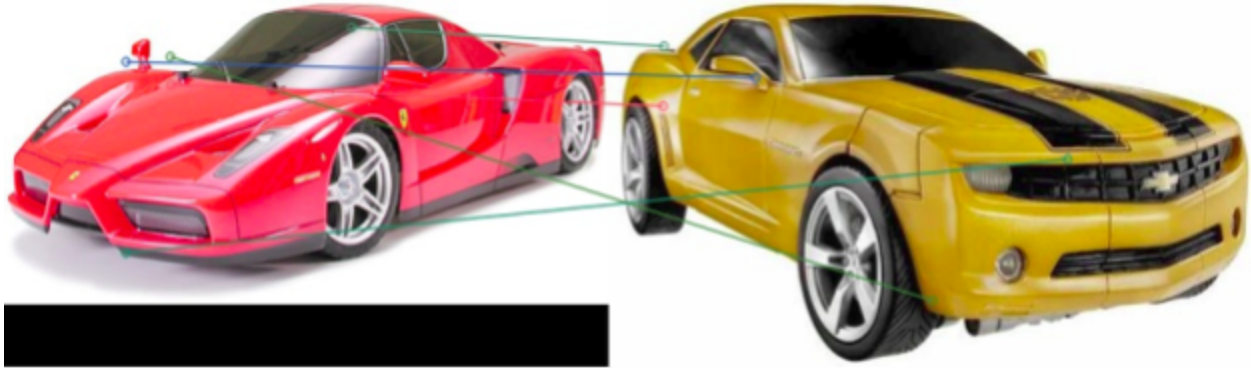


Ferrari 1 and Ferrari 2 - SIFT output

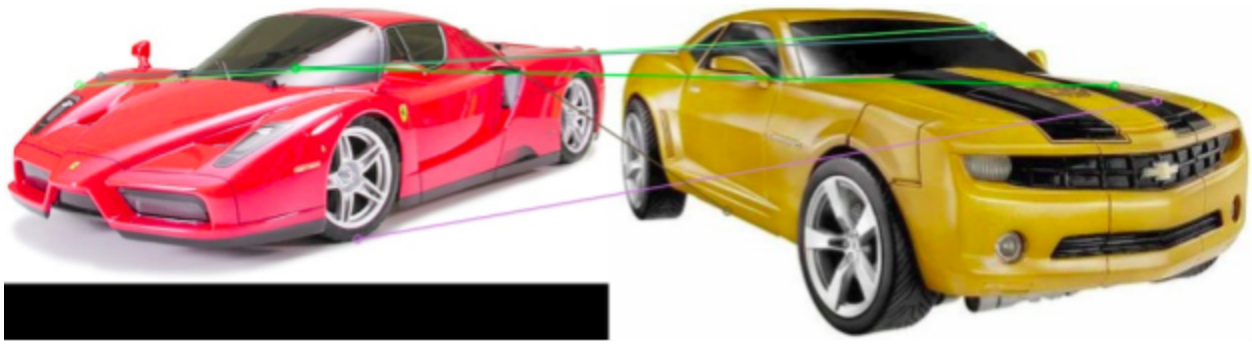


Ferrari 1 and Ferrari 2 - SURF output

Images compared : Ferrari_1 and Bumblebee
Hessian threshold: 700
Feature matcher: FLANN matcher

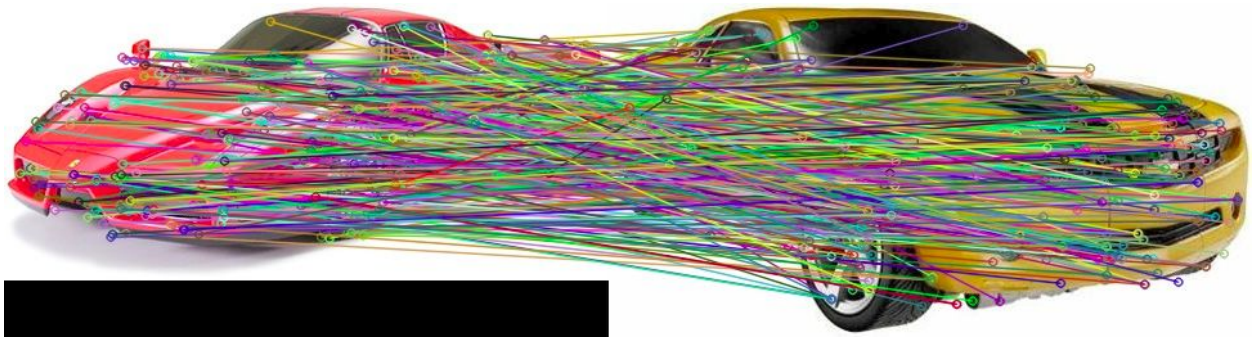


Ferrari_1 and Bumblebee - SURF output

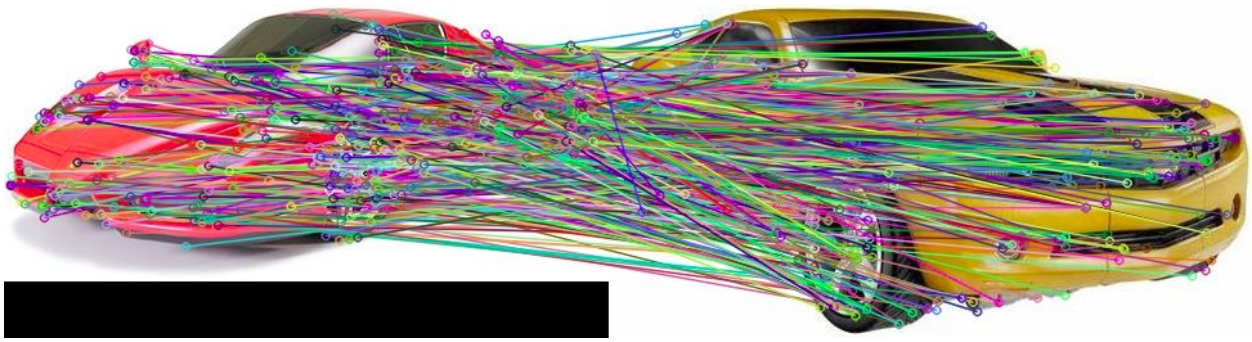


Ferrari1 and Bumblebee - SIFT output

IMAGE MATCHING ALGORITHM : BRUTE FORCE
Images compared : Ferrari_1 and Bumblebee
Hessian threshold: 400
Feature matcher: Brute Force matcher



Ferrari 1 and Bumblebee - SURF output



Ferrari_1 and Bumblebee - SIFT output

DISCUSSION:

Salient point extraction and keypoint detection

The SIFT and SURF methods of feature extraction are used to obtain the salient points. We see that as the Hessian threshold is increased the number of keypoints in the image decrease and this results in the key points which are good matches and all the unnecessary points are discarded.

Image Matching

The SIFT and SURF feature extraction algorithms are used to extract the key points from the images and the FLANN and Brute force matcher are used to perform image matching.

It can be observed Brute Matcher produces more results compared to FLANN, because brute force rigorously compares all combinations of points and then arrives at the matched keypoints whereas considers the nearest neighbour and then provides the best match points

3 C. BAG OF WORDS

APPROACH AND PROCEDURE

In this problem we use the algorithm of k-means clustering and the bag of words method to find the closest match of a target image to the remaining images provided as arguments to the program. This popular computer vision method is mainly used for image classification, which the image feature vectors are treated as codewords. The three main steps involved in the Bag of Words approach are - Feature detection, feature description and codebook generation.

a) Feature detection:

In this step, local neighbourhoods of pixels are chosen and the existence of features is detected by applying various algorithms and filters. The output obtained after applying these algorithms will be feature vectors which can represent isolated points in the image, continuous curves and connected regions in the image.

b) Feature representation:

After the feature detection phase, each image is processed as a collection of several patches. Feature representation algorithms like SIFT are applied, which generate feature descriptors. A good feature descriptor is a vector describing the properties in that patch and is invariant to scale, rotation, and other transformations. These feature descriptors are passed to the next stage for codeword generation

c) Codeword generation

The final step of the BoW approach is the codebook generation, where the feature descriptors are converted into codewords, a collection of which form the dictionary/ codebook. The codewords are generated by using the k-means clustering approach. At the end of k-means clustering approach, a codebook with k clusters would have been generated and the centers of these clusters are called centroids. The number of centroids, determine the codebook size. Each patch in the image will mapped to a certain codeword in the book and finally the entire image can be represented as a histogram of codewords.

Procedure for Bag of Words:

1) Read the sample images and the target images into matrices for future processing
2) Set the minimum Hessian threshold value to get an optimal number of points to detect
3) Create an object of the SIFT Detector by passing the Hessian threshold value
4) Calculate the keypoints and descriptors for all the 4 images
5) Creating a training set of 8 clusters by passing the descriptors of the images which are to be compared against the target image
6) Create a vocabulary of words from the descriptors of the sample images and using the cluster() function. This function generates the cluster centers or centroids for training
7) Calculate the Euclidean distance between every descriptor of an image and the centroid of each cluster of the vocabulary and assign labels to that feature descriptor
8) Get the frequency of how many feature descriptors or codewords belong to each cluster and write the values to a file
9) Now compare the number of codewords in every cluster in the target image with those of the remaining images and calculate the error distance between them
10) Find the pair of images which have the least distance between them by comparing codeword count in each cluster.
11) Confirm that the images with the minimum error are most similar compared to other pairs

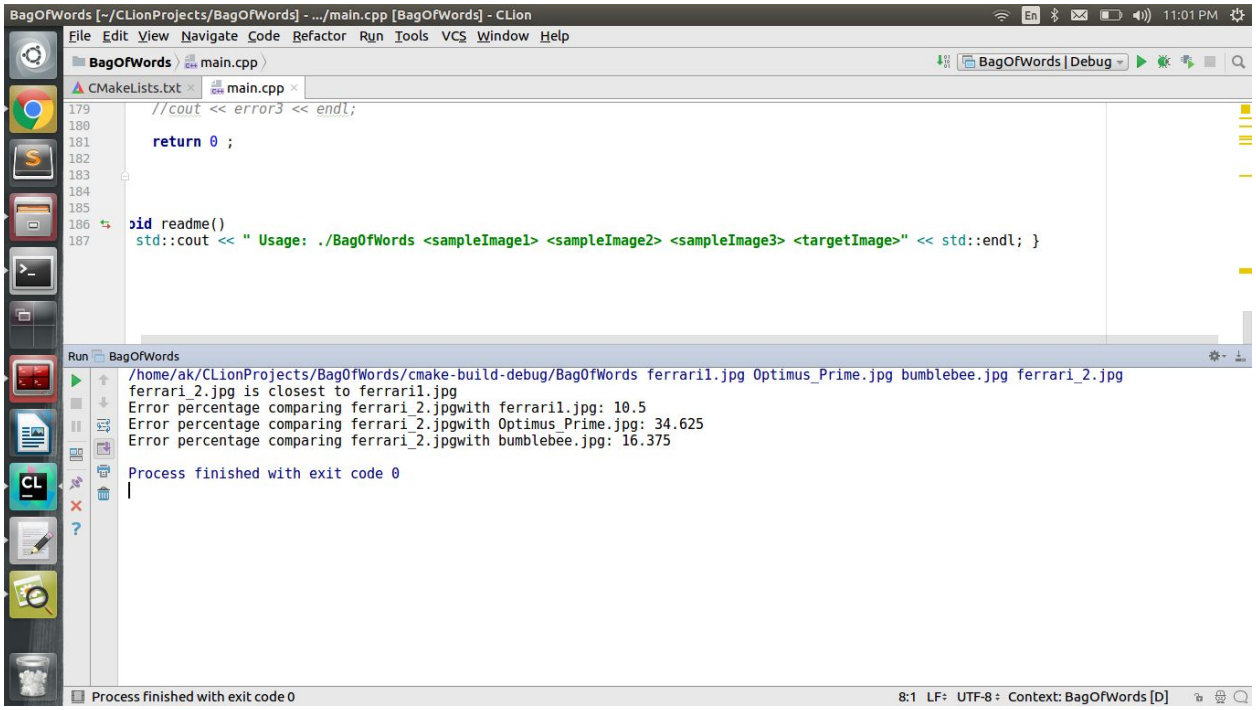
EXPERIMENTAL RESULTS:

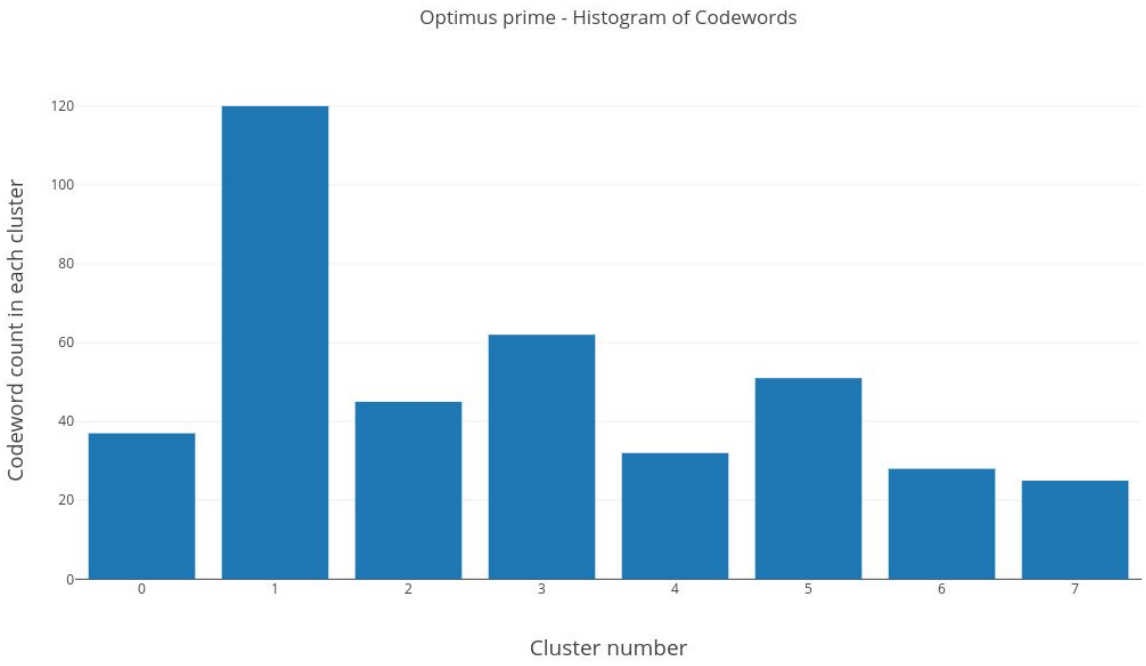
In this problem we used the bag-of-words(BoW) approach to represent an image as a collection of codewords and then used k-means clustering to find the closest match to a target image. In practical applications, the Bag of words approach is used for object categorization.

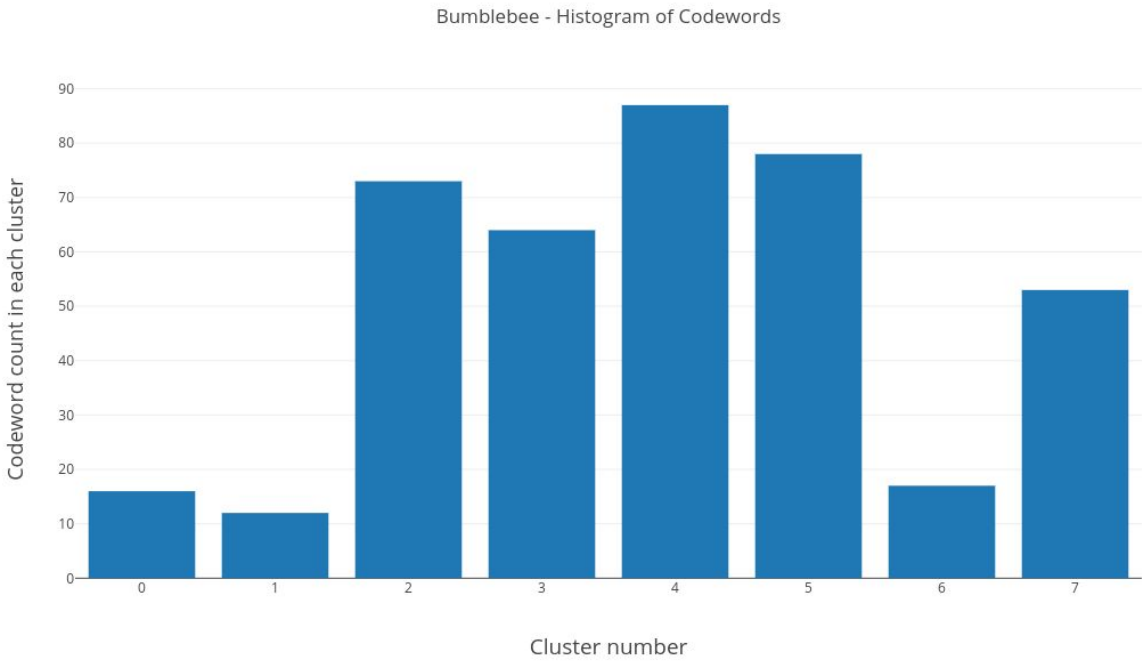
In the current problem, after the feature descriptors had been obtained from the Images using the SIFT detector, the training set is created. The number of clusters in the training set was chosen to be 8. With the number of clusters as 8, codewords are generated for every feature vector are compared with the centroids of the 8 clusters and they should be assigned labels accordingly. Then the histogram of the number of codewords in each cluster is obtained, which tells us how many codewords in an image are similar.

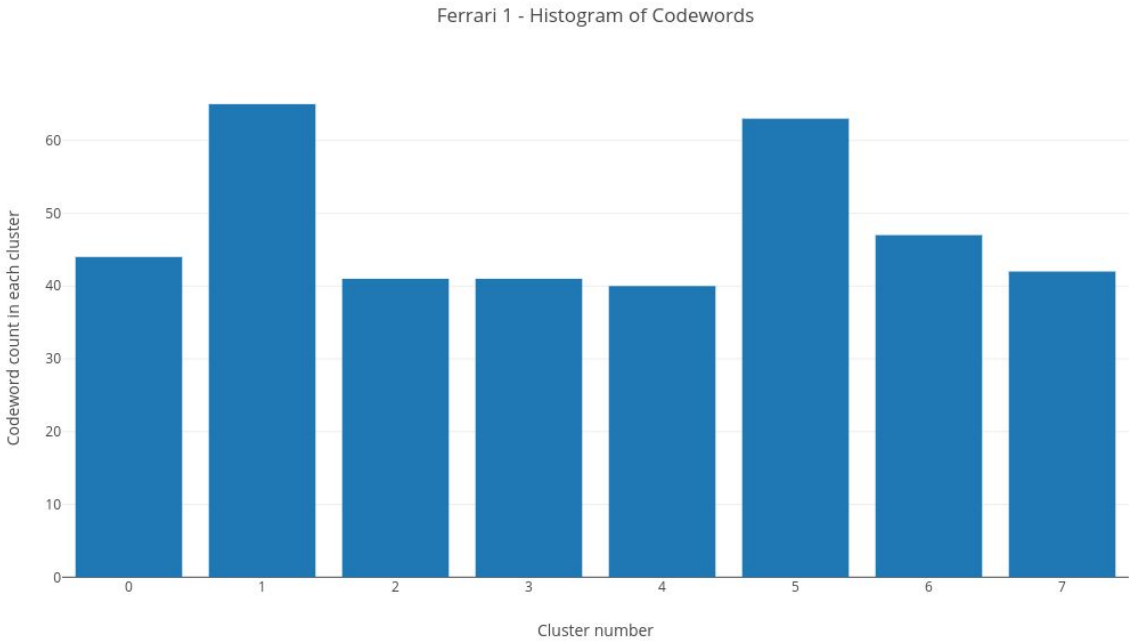
The information conveyed by each of the eight clusters is that there are 8 unique sets of features which can be used to represent the entire image. Hence for two images to be similar the number of codewords/features in each of the 8 clusters should be similar or approximately the same value thereby conveying that the image has similar features and hence the two images might be similar in nature.

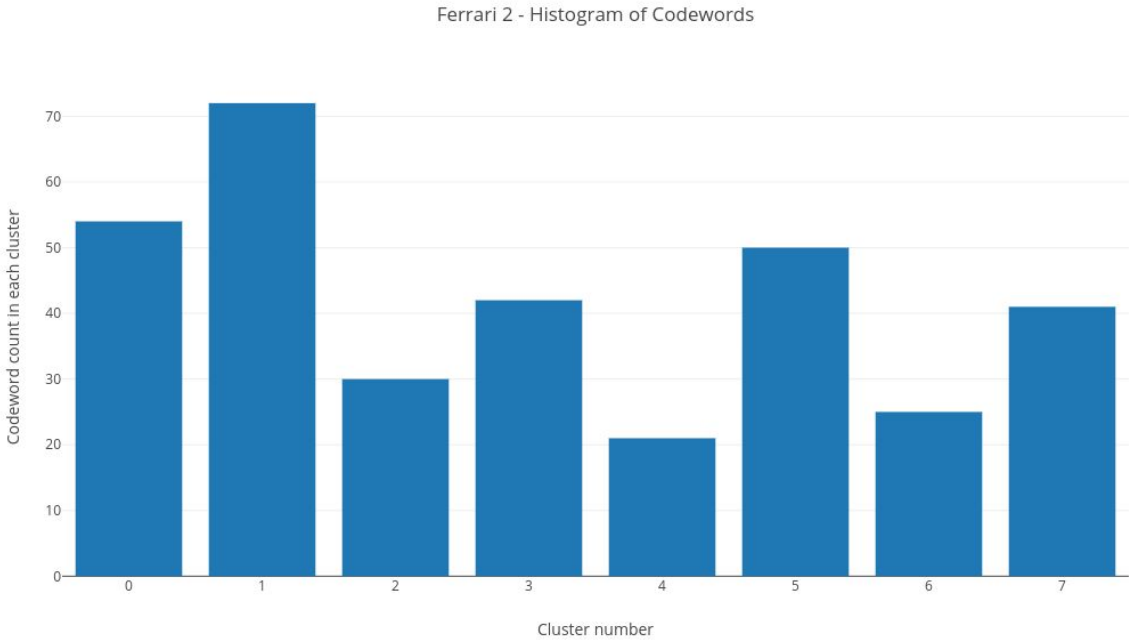
Hence when the Ferrari 1 and Ferrari 2 images were compared and I got an error value of around 10.5%, I could conclude that some of the features in the two images had to be same or very similar hence giving me such a small error. The error rates obtained when the Ferrari 2 image was compared with Bumblebee or Optimus prime are 34.625% and 16.375% respectively and they are high values since there very few parts of the pairs of images which could be similar.




Optimus prime - Histogram of Codewords
Codeword count in each cluster
120
100
80
60
40
20
0
0
1
2
3
4
5
6
7
Cluster number


Bumblebee - Histogram of Codewords
Codeword count in each cluster
90
80
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
7
Cluster number


Ferrari 1 - Histogram of Codewords
Codeword count in each cluster
0
10
20
30
40
50
60
0
1
2
3
4
5
6
7
Cluster number


Ferrari 2 - Histogram of Codewords
Codeword count in each cluster
0
10
20
30
40
50
60
70
0
1
2
3
4
5
6
7
Cluster number

References:

1) Study and Comparison of Various Image Edge Detection Techniques - http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.927&rep=rep1&type=pdf
2) Laplacian/Laplacian of Gaussian - https://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm
3) Open CV tutorials - http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
4) http://savvash.blogspot.com/2015/03/from-feature-descriptors-to-deep.html
5) http://graphics.cs.cmu.edu/courses/16-824/2016_spring/slides/seg_1.pdf
6) https://arxiv.org/pdf/1406.5549.pdf
7) https://github.com/pdollar/edges
8) https://cs.nyu.edu/~fergus/teaching/vision_2012/9_BoW.pdf
9) https://stackoverflow.com/questions/14710031/is-dense-sift-better-for-bag-of-words-than-sift