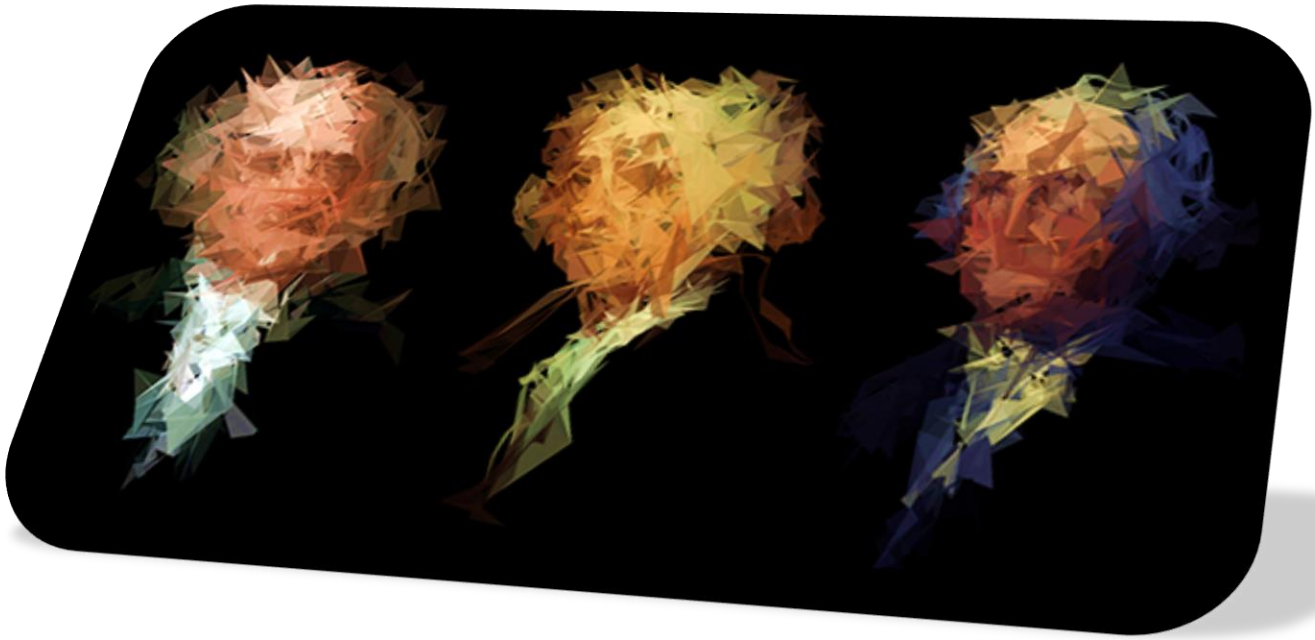
EE 569
HOMEWORK #3



SHIVA SHANMUGAPPA PATRE

spatre@usc.edu

1. Problem 1

1.1 Motivation

Another important application of Image Processing is Texture Classification. A texture of an image is something which obeys statistical properties. Image textures have similar structures that repeat on a regular basis in the image. The textures do have some certain amount of randomness. The textures provide important information about the spatial arrangement of pixel intensity values. It is very important to study different textures present in an image as they provide details which are essential for image segmentation.

It is important to differentiate textures based on their properties for segmentation purposes. We usually differentiate structures using K means clustering. This would need us to extract features from different images and represent them as feature vectors. We use different laws filters in this problem of size 5x5. These filters have certain specific properties. We basically need to obtain the energy values from each of the 9 filter responses for each texture type. K means clustering algorithm is applied and its performance is evaluated. We also study the feature reduction using PCA(Principal Component Analysis).

The main motivation to study texture classification is to improve image segmentation. In this problem we study this and the image segmentation as well. More of this is discussed in the approach and discussion.

1.2 Approach

1.2.1 Texture Classification

This is a classic case of a machine learning problem. We basically need to extract god features and utilize the K means clustering over the feature vectors. In this problem we are given 12 texture images which belong to four cluster types. We use the concept of centroids to cluster these images into the different textures.

To go about it, we first generate a 5x5 laws filter from the given three 1-D filters. Each of the laws filter has a specific frequency response that filters certain frequencies. Each 1D filter is combined with other 1D filter by taking a tensor product to produce a 5x5 filter. These nine 2D filters are then stored in a filter bank. We need to use the following filters in this problem.

$$E5 = \frac{1}{6}[-1 \ -2 \ 0 \ 2 \ 1], \qquad S5 = \frac{1}{4}[-1 \ 0 \ 2 \ 0 \ -1], \qquad W5 = \frac{1}{6}[-1 \ 2 \ 0 \ -2 \ 1],$$

The 5 filters are as follows.

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]

We then convolve the 9 filters in the filter bank over the images separately to obtain 9 filter responses per image. We then calculate the energy of each filter response by applying the following formula.

$$\textbf{Energy} = \frac{1}{N*M} * \sum_{i=0}^{N} \sum_{j=0}^{M} (I(i,j))^2$$

This is applied to every image's filter responses. Finally we get a 9D feature vector for each of the image. So we get 12 Feature vectors in total. This is then used for K means clustering.

In order to classify an unknown texture based on a set of known texture types such as Rock, Grass, Weave, Sand, we first cluster the feature vectors of these textures into 4 points, known as the Centroid. Now from this we can say that the feature vectors should be classified into the 4 clusters. This is the K means algorithm, which is an iterative one. We calculate the Euclidean distance of each of the centroid with each of the feature vectors and see which is the least and we reinitialize the centroid with new feature vectors in an iterative manner. This goes on until, the new centroid and the old centroid are the same. Once this is done, the loop breaks out and we have our classified output ready.

1.2.2 Texture Segmentation

This refers to dividing or splitting up an image into multiple segments according to the textures present in the image. The procedure is similar to the above method, but here we use 3 laws filter of size 3xx to obtain 9 masks. We then store them in a filter bank after taking the tensor products with one another. The filters used for this problem are as follows.

E3 = [-1 0 1]

S3 = [-1 2 -1]

L3 = [1 2 1]

We have to extend the image by copy padding the pixels on the rows and columns as described above. We then convolute the image using these 9 filters and obtain the output responses. We next pad the image based on different window sizes like 13x13 etc and calculate the energy, just like in the previous problem, in that window and assign it to the center pixel.

We then have to normalize the image by dividing each pixel intensity in the image by the corresponding L3'L3 image response. This gives us a 9D feature vector corresponding to each pixel in the image and we need to implement K Means clustering on this.

We now have to initialize 6 centroids having 9D for each cluster. This is done by using ImageJ to find the coordinate points of the different features visually and calculating the Feature vectors using that. Now it's the same story all over. We need to calculate the Euclidean distance of each feature vector wit the different centroids and update the centroid based on the labels obtained. We then have to stop the clustering once the old centroid and the new centroid match.

1.2.3 Advanced

Everything is similar to the texture classification steps. We need to obtain the twenty five 5x5 laws filters after the tensor products. We convolute over the entire image and we get the image response. Now select a window and compute the energy for the pixels in that window and assign it to the center pixel. Now we take the maximum value in each pixel and divide all the other layers with that. We apply PCA at each and every pixel value and reduce the dimensions. We then apply K means algorithm to this dimension reduced vectors and the clustering is done.

1.3 Results

Texture 1	Texture 2	Texture 3	Texture 4	Texture 5	Texture 6	Texture 7	Texture 8	Texture 9	Texture 10	Texture 11	Texture 12
27744.4	26519.7	23082.7	28853.5	23917.5	28491.1	15665	14537.2	26380.5	14609.4	21301.3	26380.5
26993.2	25244	23099	28895.9	24124.1	28270.3	16319.9	17591	28047	16024.5	22711.9	28047
28092.5	23144.8	18810.3	27679.2	20678.8	28107.7	10632.9	12626.1	23655.2	7038.19	17590.7	23655.2
27433.2	25057.3	23920.8	28925.2	24414.7	27135.8	16989.2	17684.8	26575.3	15262.6	22592.8	26575.3
27091.1	28310	25190.6	27408.2	25948.5	28253.1	19867.7	17983.8	27316.1	17489.3	22954.5	27316.1
29012.5	25892	20034	27166.8	21399.4	28434	13251.6	12962.2	24361	9325.27	18880	24361
28199.6	22266.9	19025.8	27410.2	21204.4	27420	10647.2	12561.5	23403.9	8646.25	16884.1	23403.9
29108	24885.6	20995.9	27778.7	23005.6	27946.1	13099.5	13576.3	23551.8	10387.3	17096.8	23551.8
26716.3	20372.6	15211.7	27576.2	15958.1	27427.3	7620.16	7167.75	21473.1	3847.92	11890.5	21473.1

The feature vectors obtained

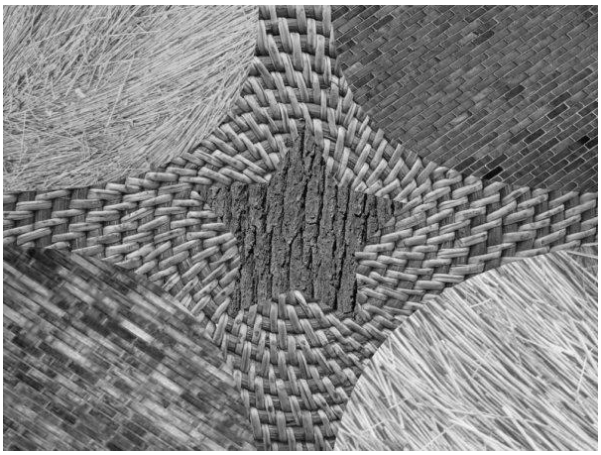
Textures	Visual Inspection	Output Labels
Texture_1	ROCK	1
Texture_2	GRASS	2
Texture_3	WEAVE	3
Texture_4	ROCK	1
Texture_5	WEAVE	3
Texture_6	ROCK	1
Texture_7	SAND	4
Texture_8	SAND	4
Texture_9	GRASS	2
Texture_10	SAND	4
Texture_11	WEAVE	3
Texture_12	GRASS	2

Output Values for Texture classification

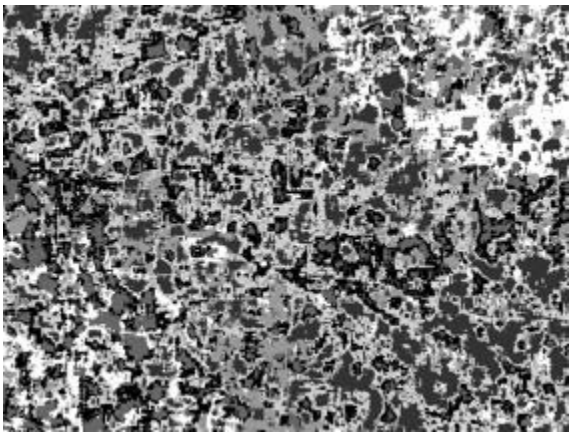
Textures	Labels
ROCK	1
GRASS	2
WEAVE	3
SAND	4

Class Descriptors

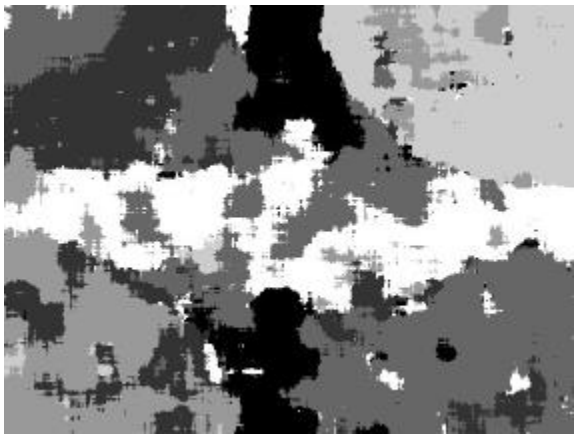
2b Outputs



comb.raw



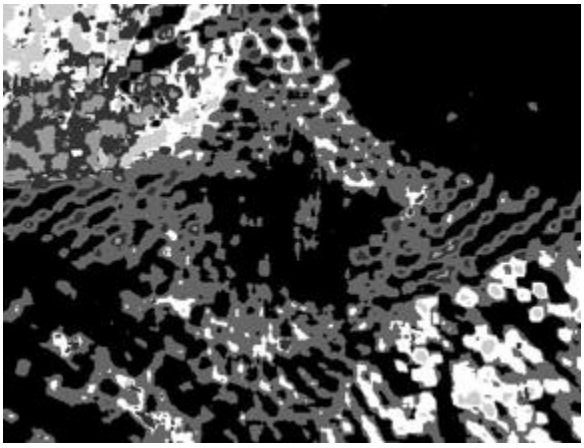
Window=13



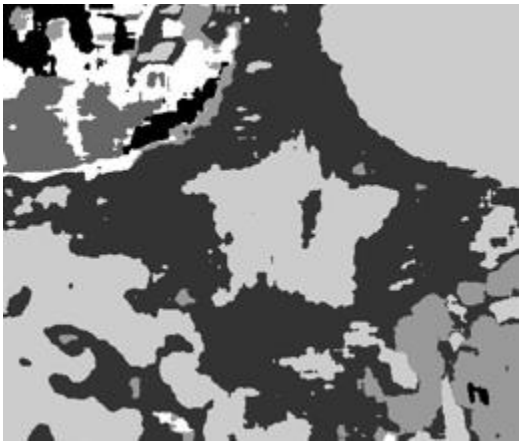
Window:33

Window:47

PCA Outputs



Window:13, Components:2



Window:21, Components:7

Window:47, Components:4

1.4 Discussion

The texture classification outputs are as shown above in the results. It is verified to be the same by visual inspection. It takes a few iterations to come up to the correct output. Hence, texture classification is successful.

The texture segmentation outputs for the comb.raw image is as shown above. We have varied the window sizes and the outputs have been obtained accordingly as seen above. As the window size increases, we can see a better segmentation results.

In PCA, the segmentation is visible better when compared to the above method. As we increase the window size and components, the image is getting smoother.

2. Problem 2

2.1 Motivation

One of the most important applications of Image Processing is Edge Detection. An edge can be defined as the points in the image where there is a sudden change in grayscale. Edge detection is a critical problem in the field of Computer Vision. It is used in the aspects of Object detection, Scene detection, Object tracking, etc. This is a challenging problem as there is a whole new level of interaction between pixels like the brightness variations, colors, continuity, symmetry and textures and a whole lot of mathematics to be considered to call a pixel an edge point.

In this problem we will apply Sobel edge detector method, Laplacian of Gaussian filter method and Structured edge detector for faster edge detections with a view point of segmenting a particular object from the image.

Sobel edge detection is used in Image processing and Computer Vision to create an image emphasizing edges. Sobel operator is basically convolving the image with a 3x3 integer mask or filter over the image in the horizontal and vertical directions. This is inexpensive in terms of computation.

Another famous edge detection method is the use of a LoG filter or the Laplacian of Gaussian. This is similar to Sobel, but uses a 5x5 pre calculated mask as the filter and is convolved over the image once and the edges are seen better.

Structured Edge detection method for edge detection takes the advantage of the structure present in the image patch to train a random forest classifier. This was mainly done with the motivation to apply effective edge detection in real time. We shall explore the working and quality of result in this method.

Let the fun begin!

2.2 Approach

2.2.1 Basic Edge Detector

The given image is RGB. We need to convert it to grayscale to be able to apply these edge detection techniques. So I used the Luminosity method to convert the RGB image into grayscale.

Luminosity Method : 0.21*R+0.72*G+0.07*B

Then I defined an X gradient filter and Y gradient filter of the Sobel operator as given in the Discussion, as follows.

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

Gx

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

Gy

Since this is a 3x3 mask, we need to extend the image by padding it with one row and column on each side. Instead of zero padding, I copy the penultimate row and column for each of the padded row and column correspondingly.

The next task is to convolve this Mask over the entire padded image and replacing the obtained sum at the center pixel. We do this for Gx and Gy. Once we get it , we can find the minimum and maximum values of this matrix. We use that to normalize the matrix to be able to display as the output, as follows.

255*((G(i,j)-min)/(max-min))

We then calculate G, using **G=sqrt(Gx^2+Gy^2).**

This is then normalized as above and we get the output as shown in the result.

To tune the threshold to get the best edge map, we first store the entire 2D array in a 1D array. This is then bubble sorted and I take the 90th percentile value of the index of the entire array. This value is used to threshold the best edge map. I found that the 85th and 90th percentile values give better result.

Next is the Zero Crossing detector. This is done using LoG method or the Laplacian of Gaussian. We use a 5x5 convolution kernel for the mask as follows.

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

In this, I calculated the PDF of the histogram and then the CDF from it. The CDF is used to find the threshold. I chose two thresholds at 50% and 95% and got the corresponding pixel intensity. I then classified the pixels below this first threshold to be 64. In between the

thresholds to be 128 and greater than or equal to the third threshold as 192. The output is as shown in the result.

The last part of this problem is we need to apply a zero crossing to obtain the best edge map. This is implemented by taking the output from the last stage and extending the boundary like before, and this time we define a 3x3 window and if the center pixel I equal to 128,we check the neighborhood of this pixel in the 3x3 window to see if it any of the pixel has 64 or 192 as the intensity value. If yes, then we assign the center pixel as 255. If not, we assign it a 0. This gives us the best edge map output as shown in the result. More is explained in Discussion.

2.2.2 Structured Edge

SE method of Edge detection is a new method which came out in the recent years. SE method is a state of the art algorithm that is used to compute the edges on a given image. The operation of SE method can be explained as something which considers a patch around the pixel and determines the likelihood of the pixel being an edge point or not. These edge patches are classified into sketch tokens using Random forest classifiers. Usually, a set of sketch tokens represent a variety of local edge structures such as parallel lines, straight lines, T junctions, Y junctions, curves etc. When the input and output spaces are complex, structured learning is utilized to address problems associated with training a mapping function.

The Random forest is trained by taking a patch from the input and ground truth images and forming a structured output of whether a certain combination should be classified as an edge or not. After the Random forest is trained by a sufficient number of samples, it can be tested. In the testing phase, only the test image is sent for segmentation, and because of the speed of random forests, the segmented output is computed. A Simple flowchart regarding the same is as shown in the figure below.

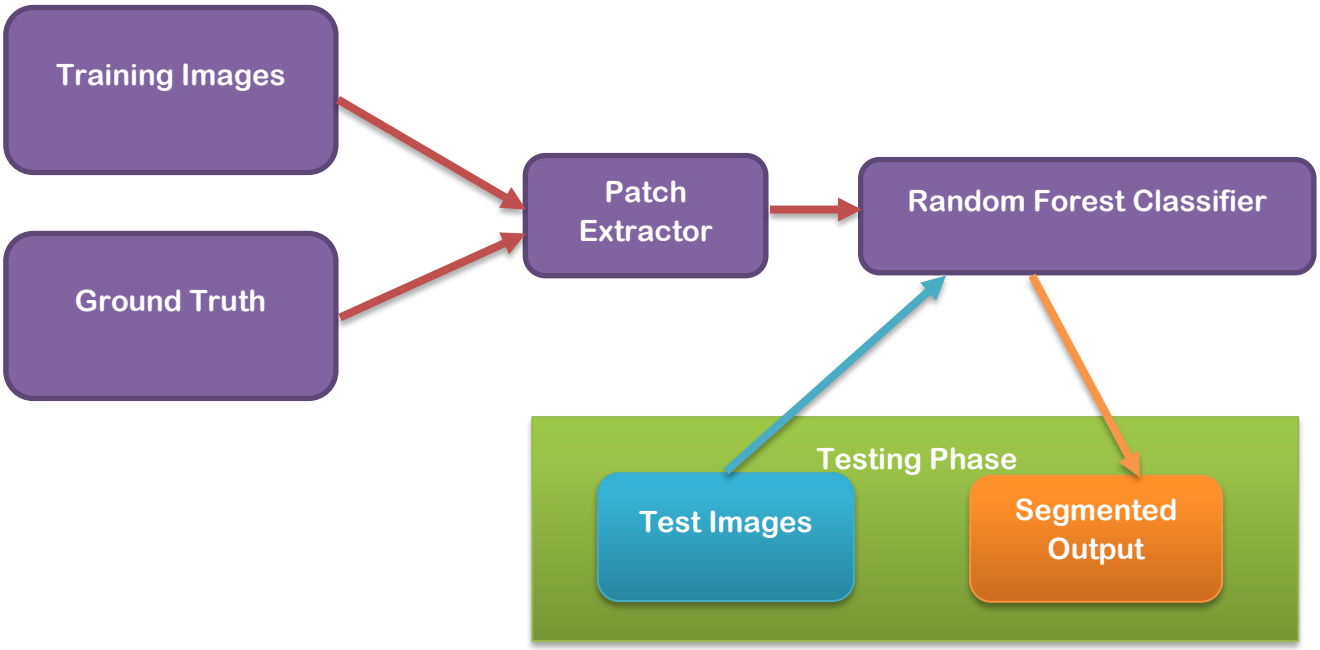


Figure: SE detection algorithm

A Random forest runs on the divide and conquer paradigm that helps to improve its performance. The main idea is to put a set of weak learners to form a strong learner. The basic element of a Random forest is a decision tree. The decision trees are weak learners that are put together to form a Random forest.

The decision tree basically classifies an input belonging to space 'A' as an output that belongs to Space 'B'. This input or output can be very complex in nature. Let's take for example, we have a histogram as the output. Input is entered at the root node in a decision tree. The input then traverses down the decision tree in a recursive manner till it reaches the leaf node. In the decision tree, each node has a binary split function with some parameters. This function decides whether the test sample should progress towards the right child node or the left child node. Usually this function is very complex.

To result in a good split of data, a set of such trees are trained independently to find the parameters in the split function. Splitting parameters are chosen to maximize the information gain.The online source code was provided and have used that to implement the above. The parameters – Multiscale, sharpen, nTrees and nms were varied to get the edge maps with thinner edges and less noise. The edge map is binarized with changeable threshold values to get the best zero crossing edge detected image.

2.2.3 Performance Evaluation

Since there are a lot of different edge detectors, I have used the online source codes to evaluate their performances. For this part of the problem, we take the output of Sobel, Log and SE. We need to convert them to binary and take the evaluation test.

Once the binary image is obtained, we pass this image to an evaluator function with different ground truths. This function returns the Precision and Recall of the edge detected image. The formula for Precision and Recall are as follows, as mentioned in the Homework.

$$\text{Precision}: P = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}$$

$$\text{Recall}: R = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}$$

We then calculate the F measure from the obtained Precision and Recall using the formula,

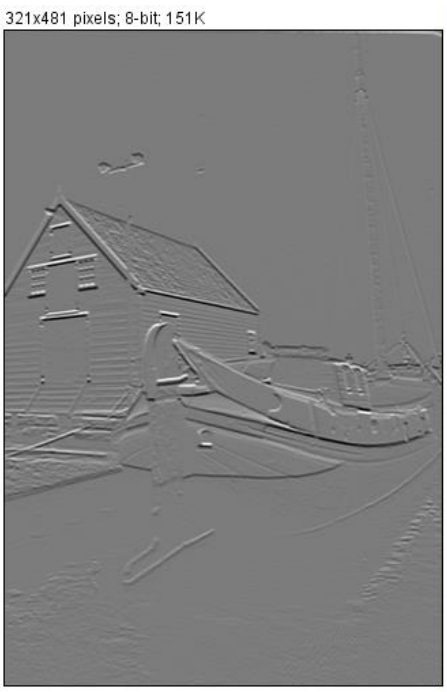
F=2*Precision*Recall/(Precision+Recall)

The online source code provided is used. The ground truths Animal.mat and House.mat have been compared with the edge maps obtained by the Sobel, LoG and the SE detector to obtain the Precision, Recall and F-Measure. The outputs are as shown below and more is discussed in the Discussion below.

2.3 Results(Sobel Outputs)

Boat.raw

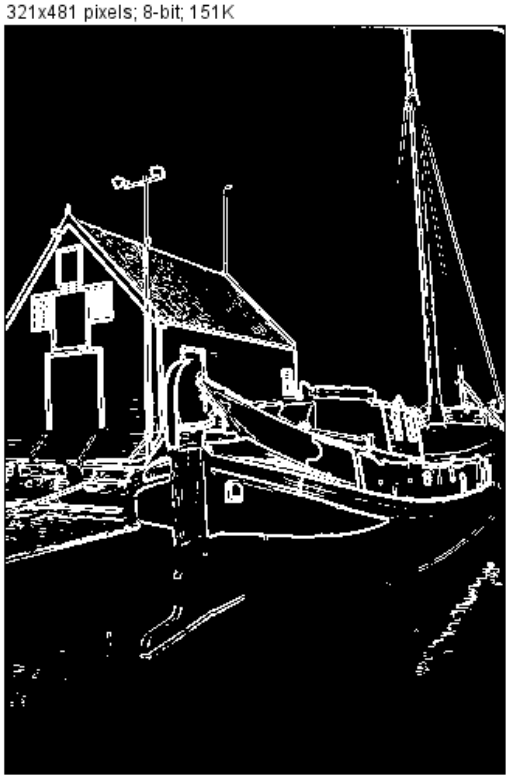
Gx Output



Gy Output

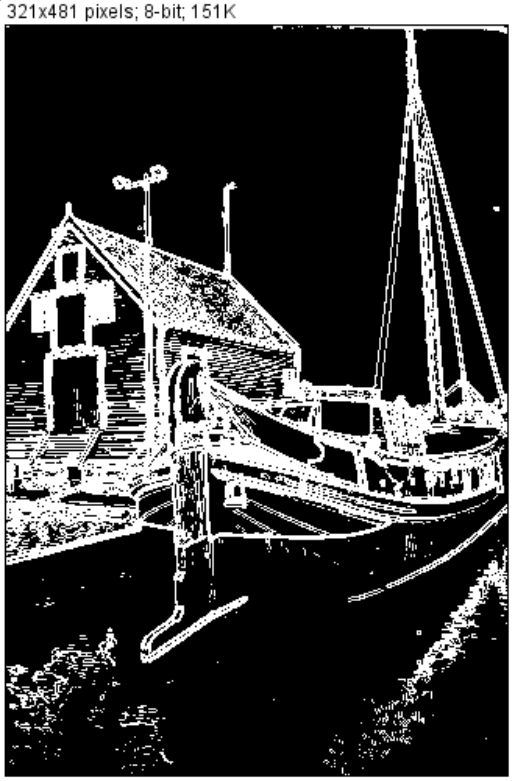
G Output

321x481 pixels; 8-bit; 151K



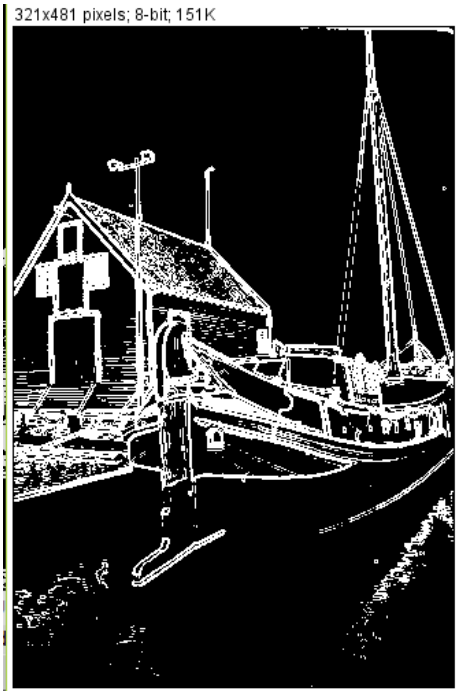
Threshold_90

321x481 pixels; 8-bit; 151K



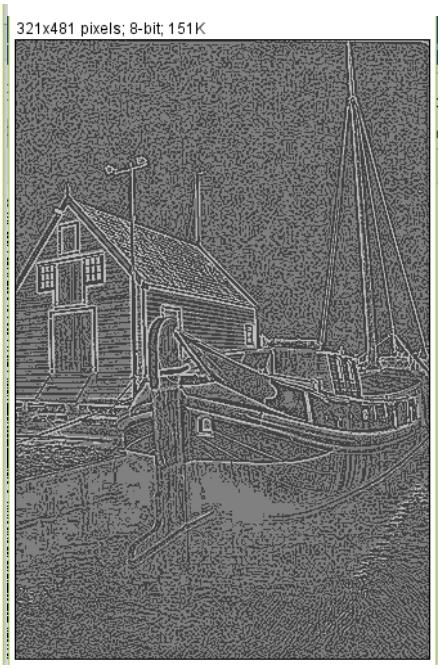
Threshold_80

321x481 pixels; 8-bit; 151K

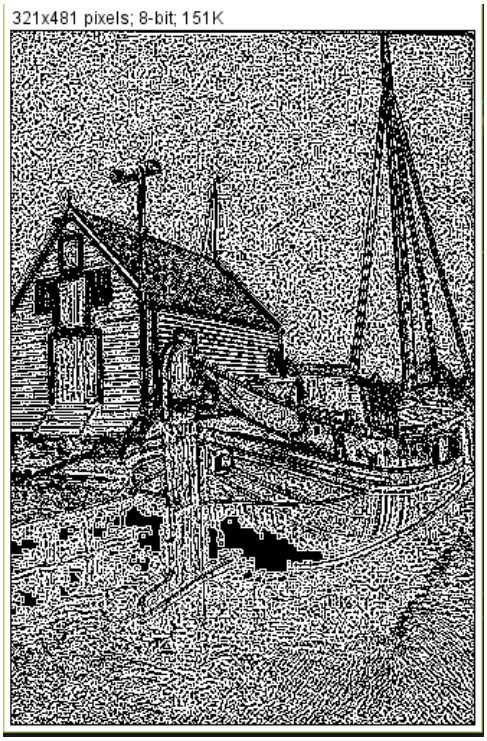


Threshold_85

Log_Normalize

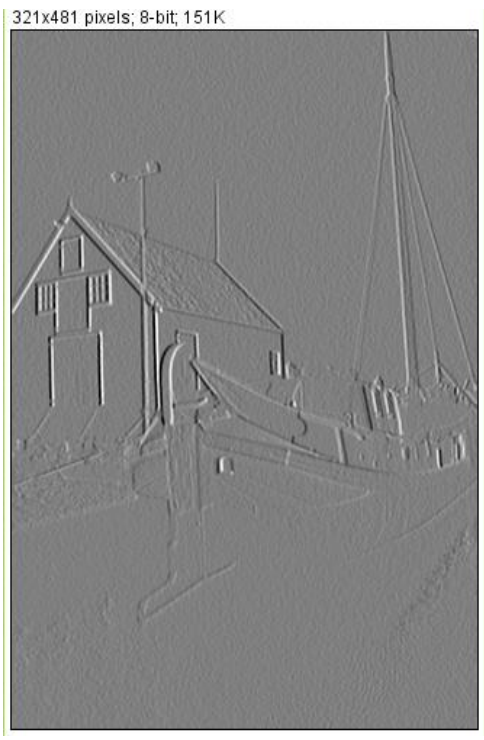


Log_Output

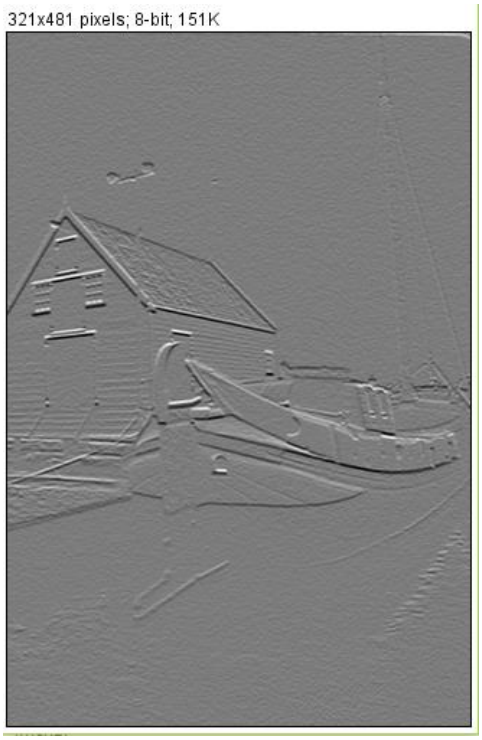


Best Edgemap for Boat.raw

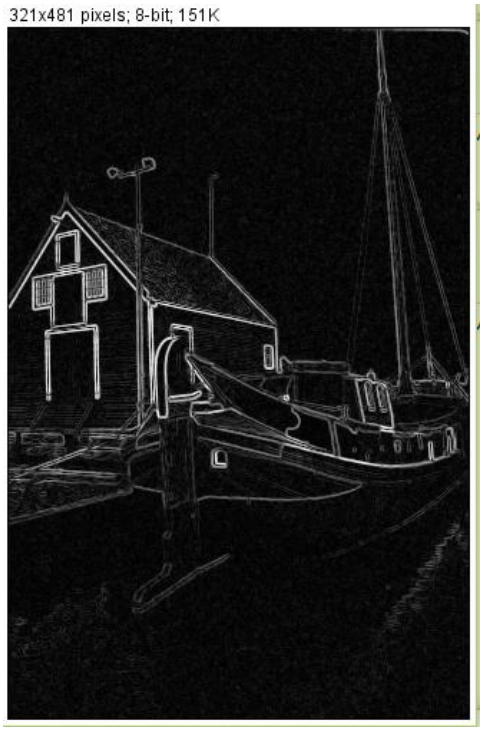
For Boat_noisy.raw(Sobel Outputs)



Gx_noisy

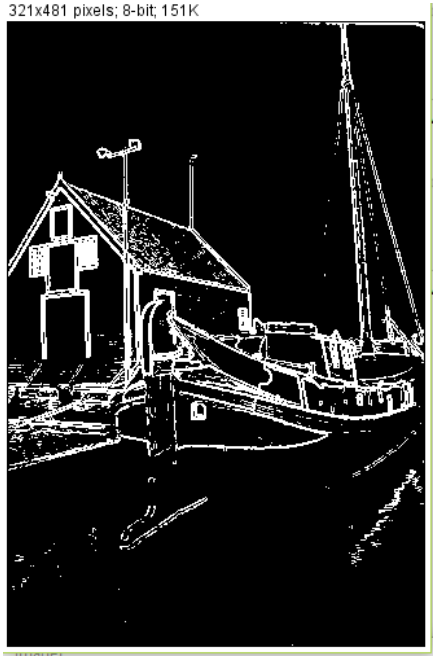


Gy_noisy



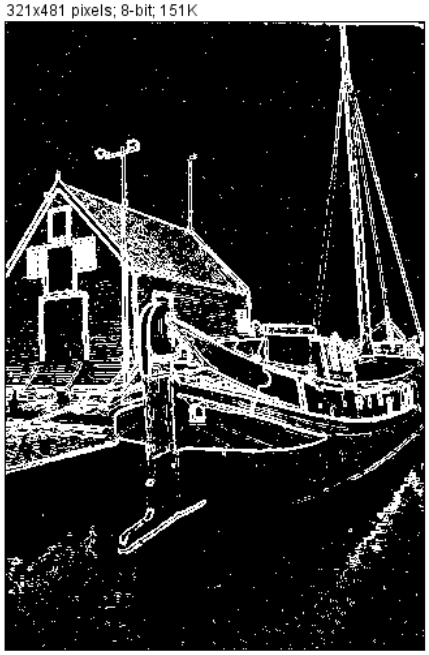
G_noisy

321x481 pixels; 8-bit; 151K



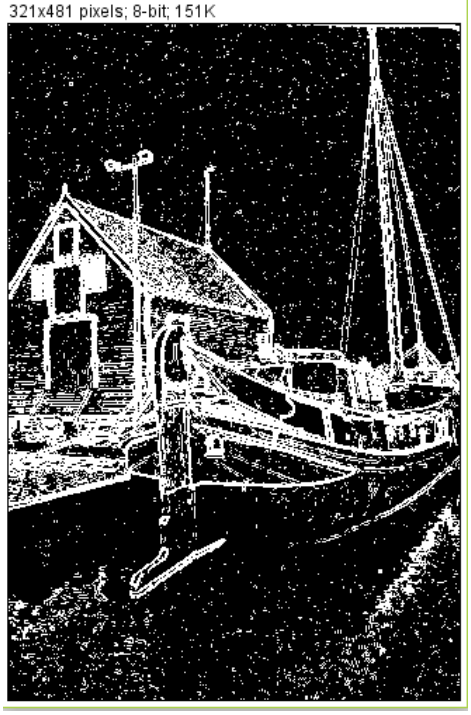
Threshold_90

321x481 pixels; 8-bit; 151K



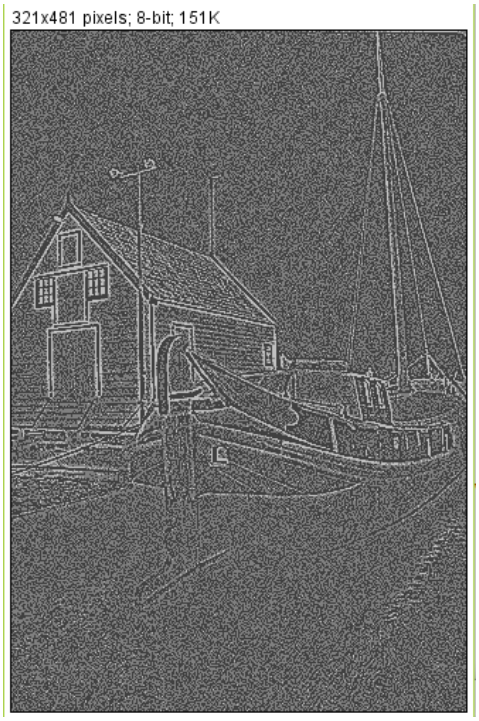
Threshold_85

321x481 pixels; 8-bit; 151K

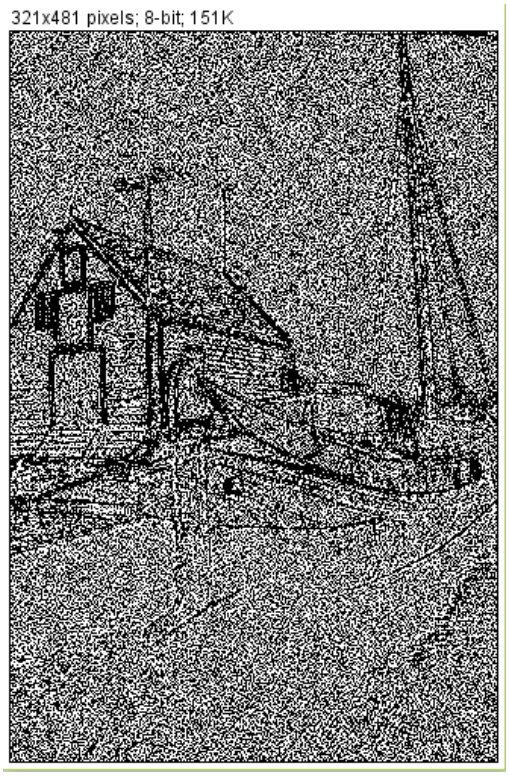


Threshold_80

Log_Normalize_noisy

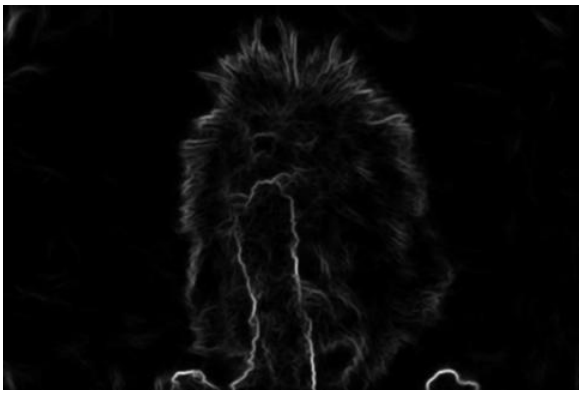


Log_output_noisy

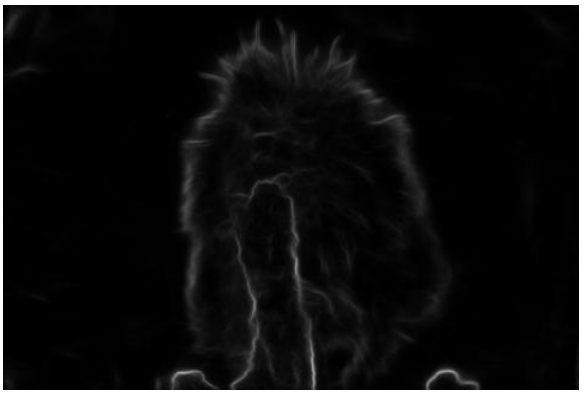


Bestedge_noisy

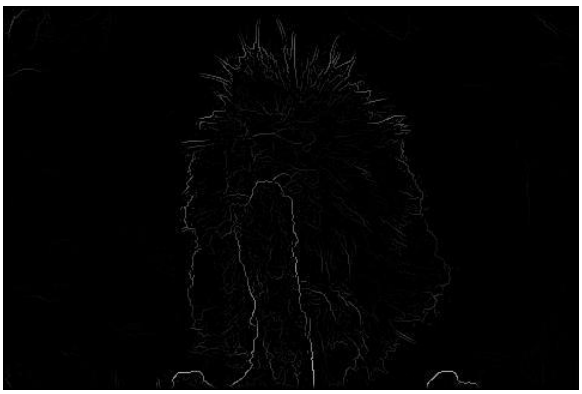
Animal image



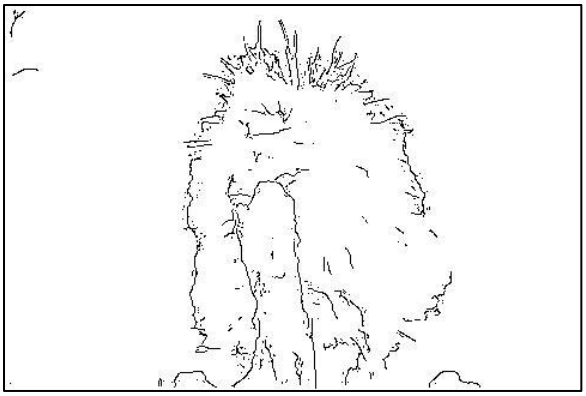
Multiscale = 0; sharpen = 2, nTrees = 4, nms = 0



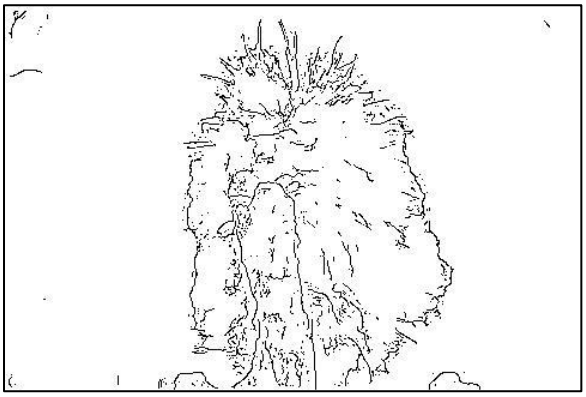
Multiscale = 0; sharpen = 1, nTrees = 4, nms = 0



Multiscale = 1; sharpen = 1, nTrees = 2, nms = 1



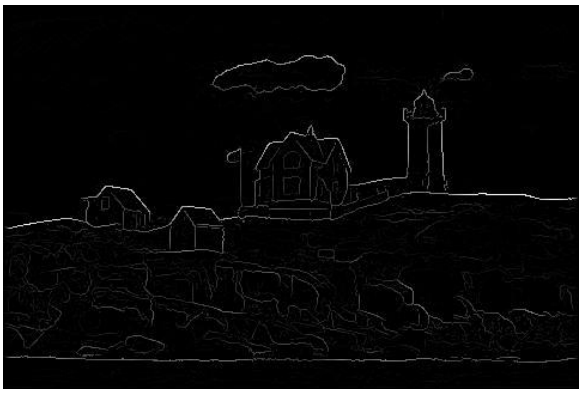
T = 0.070



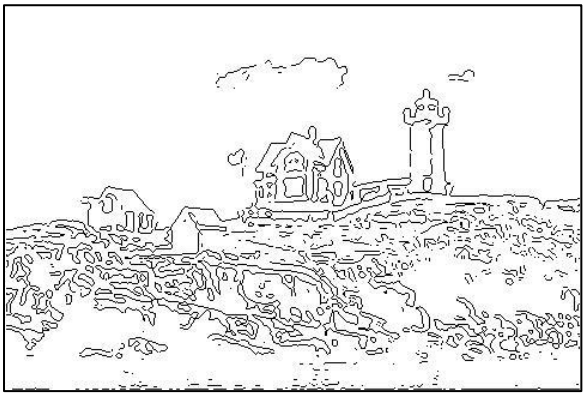
T = 0.045

House Image

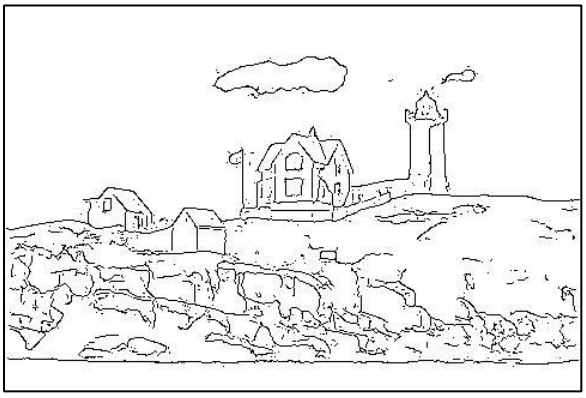
Multiscale = 0; sharpen = 2, nTrees = 4, nms = 0



Multiscale = 0; sharpen = 1, nTrees = 3, nms = 1



T = 0.015



T = 0.075

Image	Recall	Precision	F Score
House_gt1	0.6634	0.6589	0.7475
House_gt2	0.6215	0.7606	0.7248
House_gt3	0.6752	0.7214	0.7178
House_gt4	0.7354	0.6124	0.6573
House_gt5	0.7526	0.5589	0.6347
Mean	0.6896	0.6624	0.6964

Mean Precision and Recall for House image using Structured Edge

Image	Recall	Precision	F Score
Animal_gt1	0.6154	0.6986	0.6574
Animal_gt2	0.6566	0.6345	0.6423
Animal_gt3	0.4124	0.4978	0.4424
Animal_gt4	0.7275	0.6741	0.7105
Animal_gt5	0.6698	0.5545	0.6128
Mean	0.6163	0.6119	0.6131

Mean Precision and Recall for Animal image using Structured Edge

2.4 Discussion

As can be seen Sobel and Log detector successfully detects the edges. Sobel detects more edges and LoG loses some edges. When we do the same for Boat_noisy.raw, Sobel doesn't do anything with regard to noise , but Log has it reduced and the noise isn't that seen as shown in the above results.

As can be seen in the House and Animal Outputs above, SE edge detector detects the edges more successfully. Since it is a learning algorithm, it disregards the edges which are not relevant. I have varied the various parameters such as multiscale, sharpen, number of trees, and NMS. If we vary the NMS values we get thinner edges and less noise in the image and the output doesn't vary much with the other parameters varied. The Optimum threshold for Zero crossing is when it is 0.075 according to me.

The mean precision, recall and Fscore for Animal and House images for various ground truths are as above. This is lower for Animal when compared to House as the animal appears to be camouflaged into the background and makes it difficult to segment.

F measure is a measure of test accuracy and is directly influenced by precision and recall. If one of this is low, it directly results in a poor Fscore. But the precision and recall have an inverse relationship. It is because of this, if we want a good Fscore, we need to keep both precision and recall equal.

3. Problem 1

3.1 Motivation

One interesting and really useful application of Image processing is feature extraction. In many applications such as the object recognition, video tracking, gesture recognition, navigation etc, we need a means to extract meaningful features that can describe the image. For robustness, these extracted features must be invariant of the scale and rotation. SIFT and SURF are the two infamous algorithms to extract the important local features in an image that are invariant of scale and rotation.

SIFT and SURF algorithms are implemented in this problem to compare their performance and efficiency. We also utilize the obtained features to perform image matching using Brute Force technique. We will also discuss about its shortcomings and how we could overcome it.

One problem solving approach is known as the Bag of Words(BOW) approach. Sometimes there may arise a need to perform image categorization and localization to enhance the understanding of an image. The biggest challenge that arises when we need to do this task is the problems due to different camera positions, lighting differences, internal errors and various other variations in the image. This is solved by BOW model. The basic idea is to create a visual vocabulary from the training set and classify the unknown image based on this created vocabulary. If this is coupled with SIFT and SURF, we get amazing results. We shall discuss more about this below.

3.2 Approach

3.2.1 Extraction and Description of Salient Points

The only motto for extracting as good features as possible is that it contributes heavily towards the accuracy of classification. This problem gives us the opportunity to explore good features for the purpose of image classification.

SIFT stands for Scale invariant feature transform and was proposed by David G Lowe. Open CV contains SIFT And SURF features which we have used in this problem. According to the author David, the Sift feature steps are as follows.

To make the keypoints invariant of scale, we have to consider the image in different scales. This is done by Gaussian smoothing with different sigma values. After this is done, the difference of each gaussian is taken. This is done to approximate the LoG function, as the Log function calculation takes a lot of time. This is done by comparing a given point at different scales.

Now since the points of interest are found, we must avoid the once with noise and which are weak points due to low contrast or which are part of an edge. We have to use Taylor series of DoG and remove the points below the chosen threshold. Hessian matrix is computed and its Eigen values are compared.

For rotational invariance, we assign an orientation value to each key point. We first take a neighborhood around the point of interest. Then a histogram of 36 bins are created to represent the 360 degrees. The orientation of each pixel in the window is weighted by its magnitude and a Gaussian window. The weighted orientation is plotted on the histogram and the max value is chosen. Only the values which are within 80 percent of the max are considered for robustness.

We have used Open CV with inbuilt SIFT and SURF funtions which does all the above. SURF is nothing but Speeded Up Robust Features Thus it provides us a faster way to obtain feature points.

3.2.2 Image Matching

In the above extraction and description of salient points, we could see the importance of good features using SIFT and SURF. The reason to extract these features from the image is to find relevant matches among images. In this problem we shall implement the image matching. We consider only the top 75 match points and that is portrayed in the results below. Once SIFT And SURF is used to extract features using Open CV, we used Brute force matcher and FLANN based matchers to match the images.

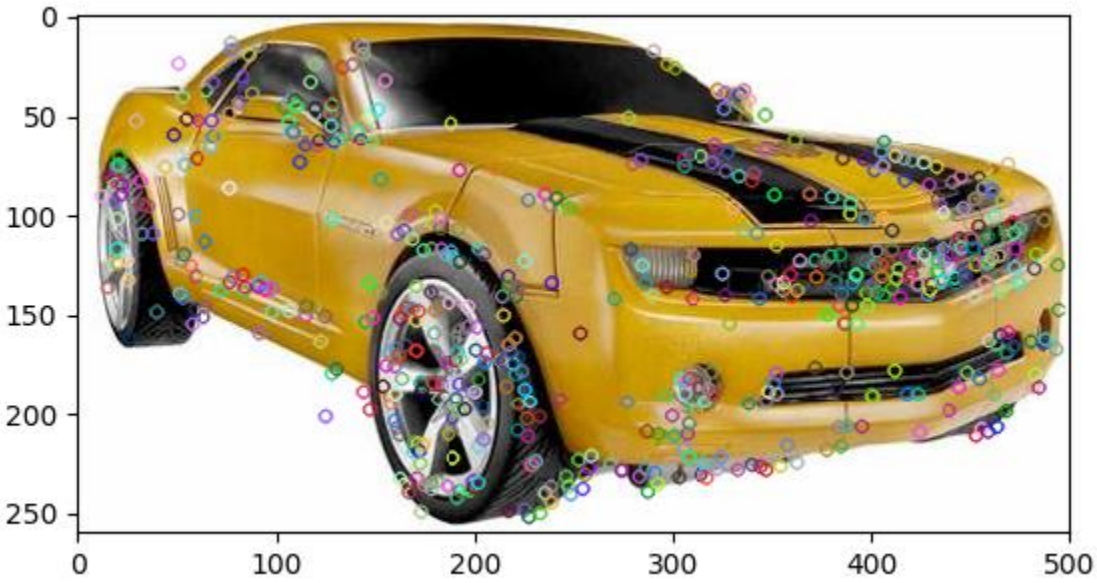
3.2.3 Bag of Words

In this problem, we classify the unknown test image which in this case is the Ferrari_2, based on the previously trained images. This is implemented using Open CV.

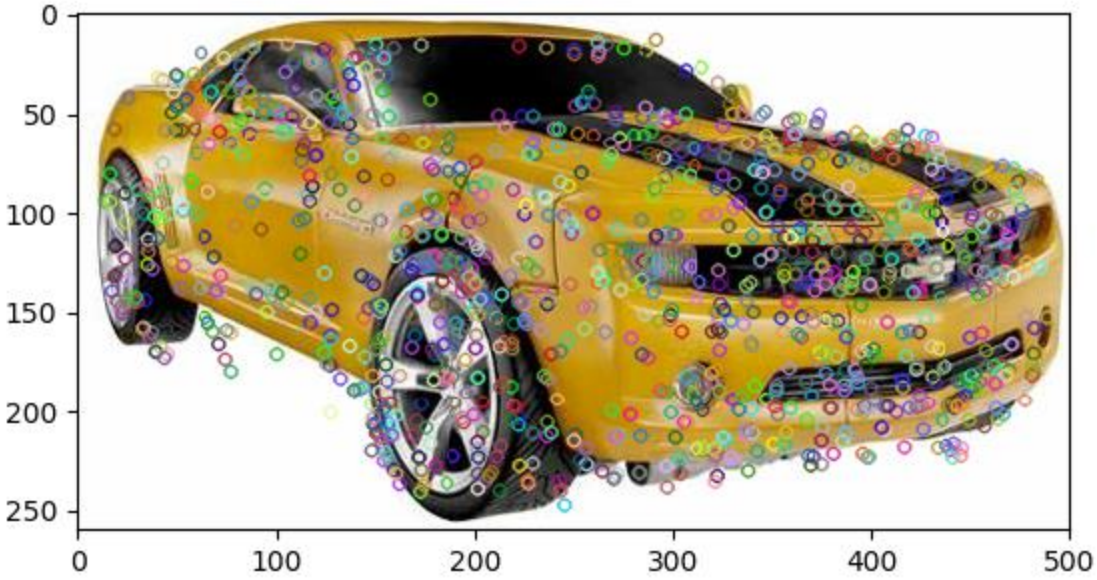
The training and test images were read and a BOW trainer was initialized with 8 clusters. From each training image we have extracted SIFT keypoints and descriptors and added it to BOW trainer.

This is then clustered with the help of K means. This builds the vocabulary of visual words that can be used to describe the image. Once the vocabulary is done, we get the histogram for each image which is of length 8. We then take the test image(Ferrari_2) and repeat the above steps. All we have to do now is to compare the histogram of the test image with that of the training image. The histogram with the closest values will be the one that the test image is similar to. To compare this histograms we have to compute chi square distance, which is calculated using chisquare function. We get pretty accurate results with this method.

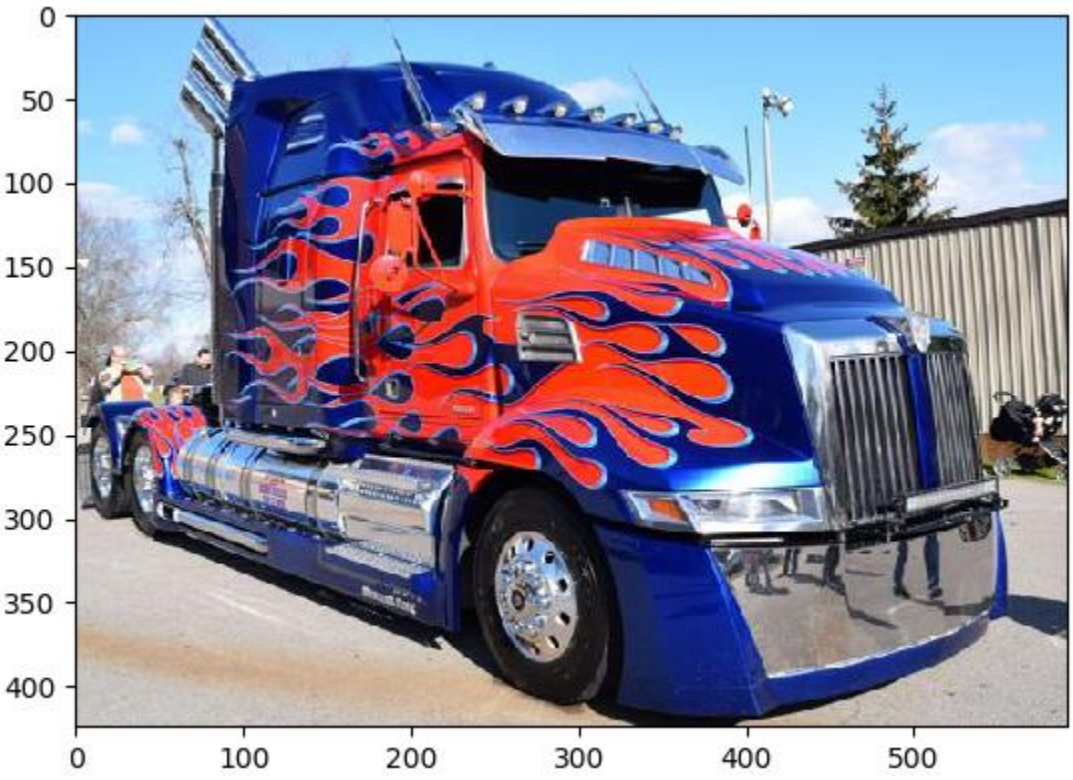
3.3 Results



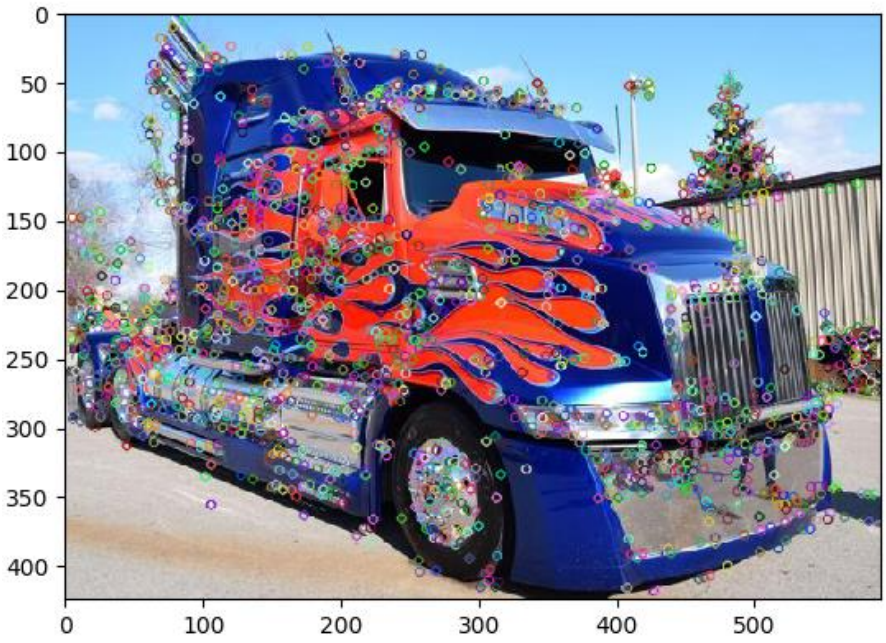
Bumblebee_SIFT



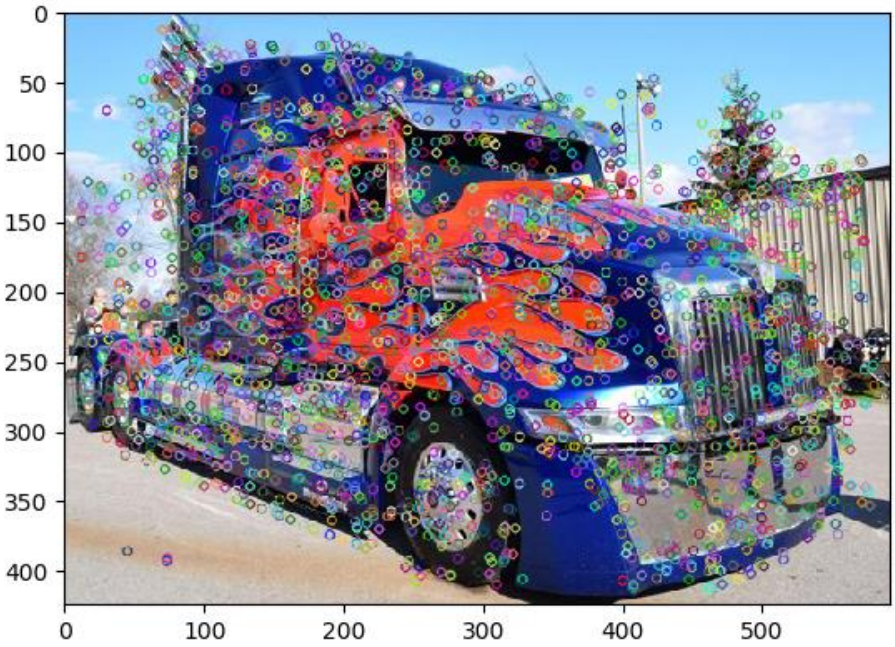
Bumblebee_SURF



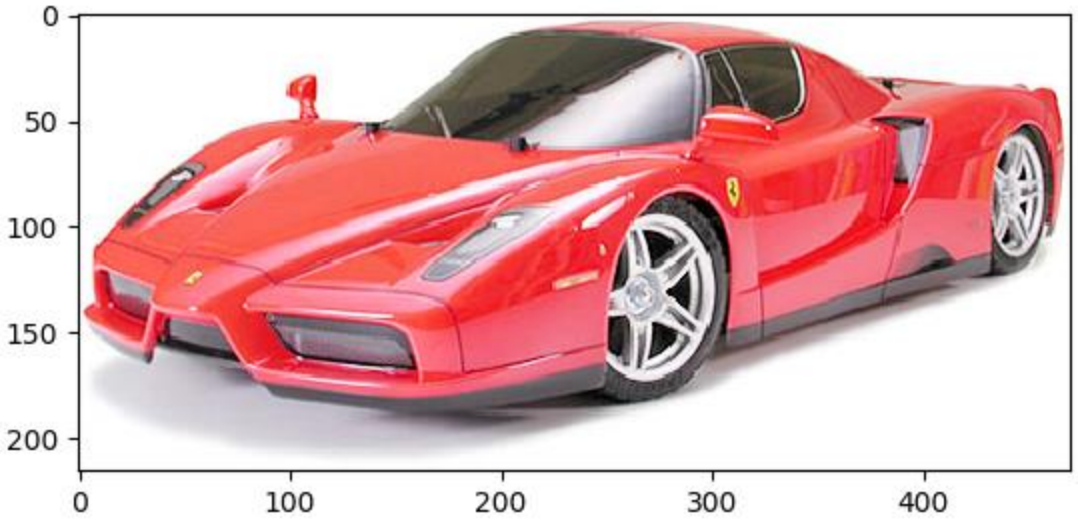
OptimusPrimeTruck



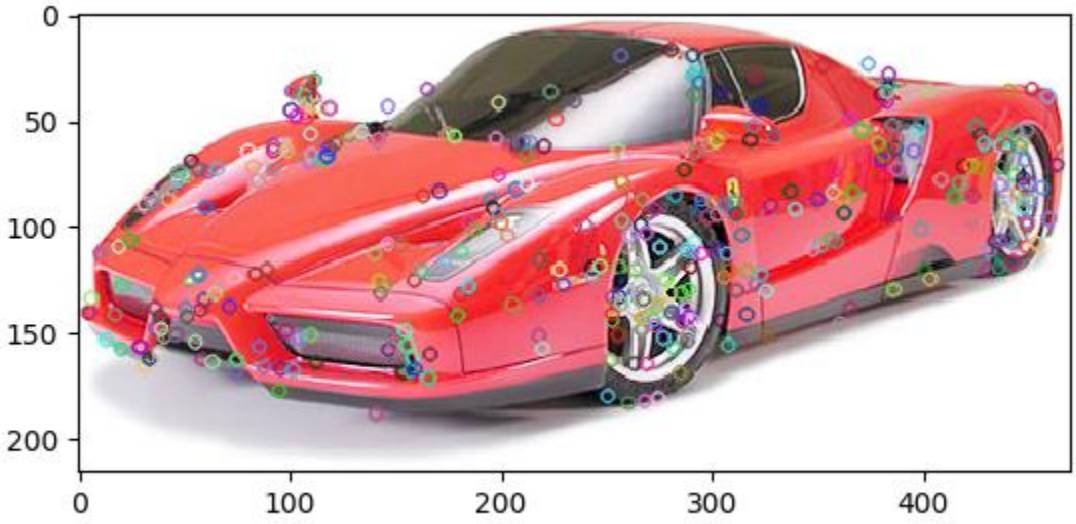
OptimusPrimeTruck_SIFT



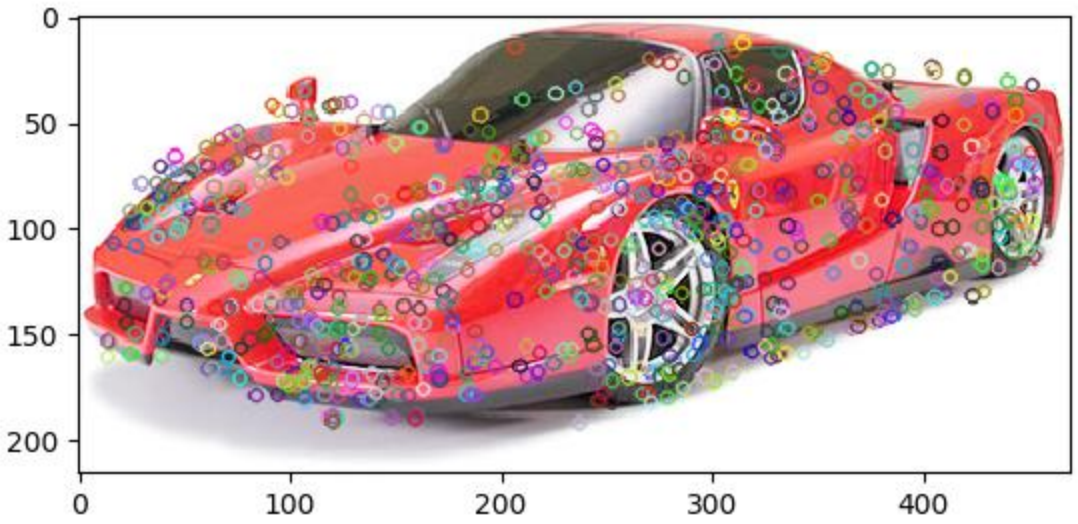
OptimusPrimeTruck_SIFT



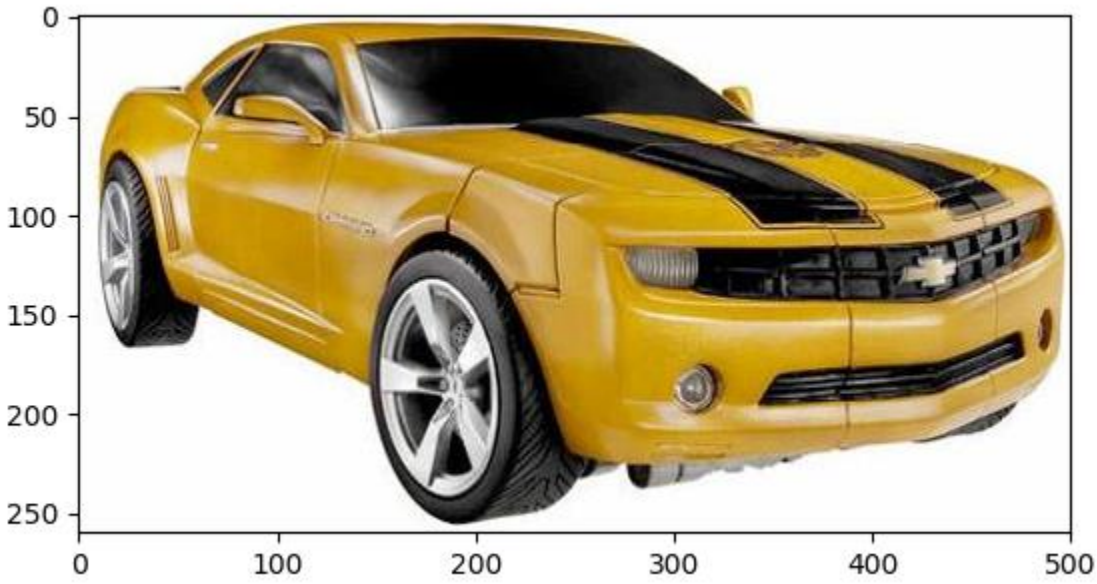
Ferrari_1



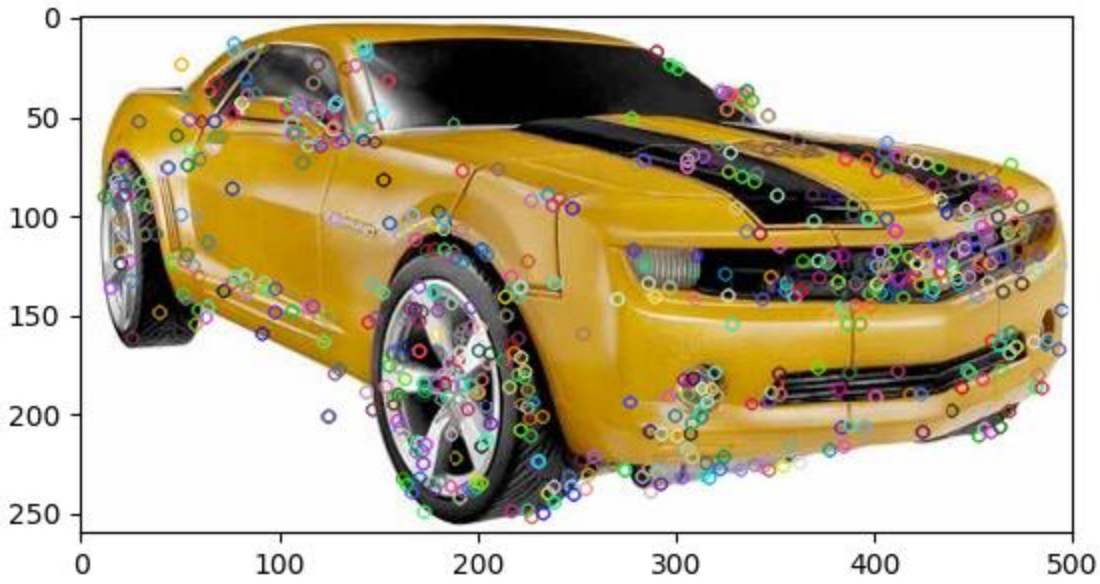
Sift_feature_ferrari1



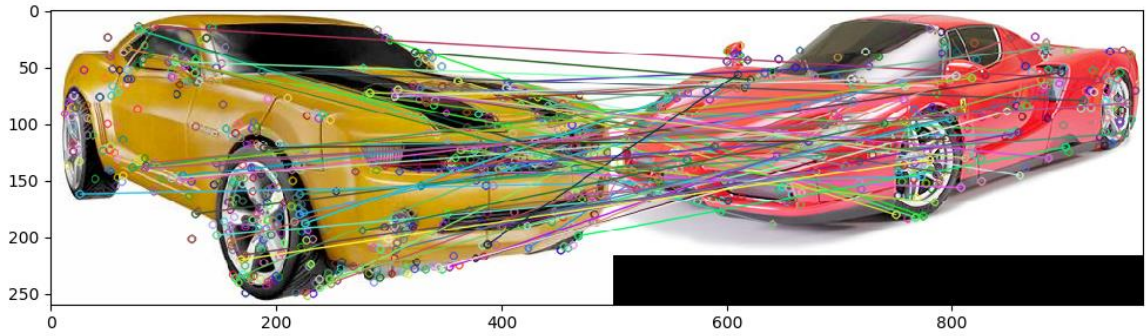
Surf_feature_ferrari1



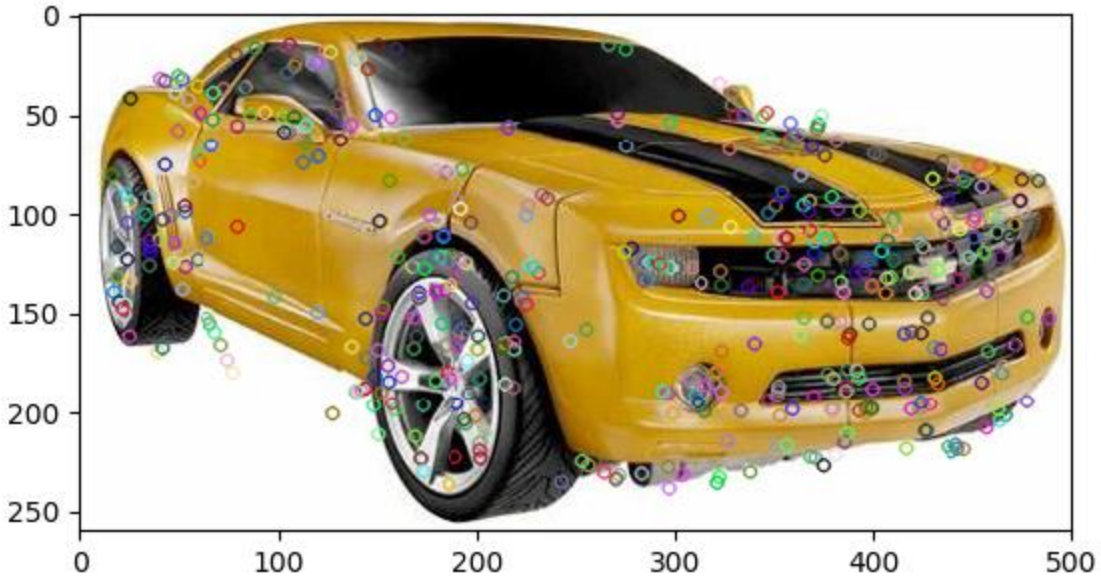
Bumblebee



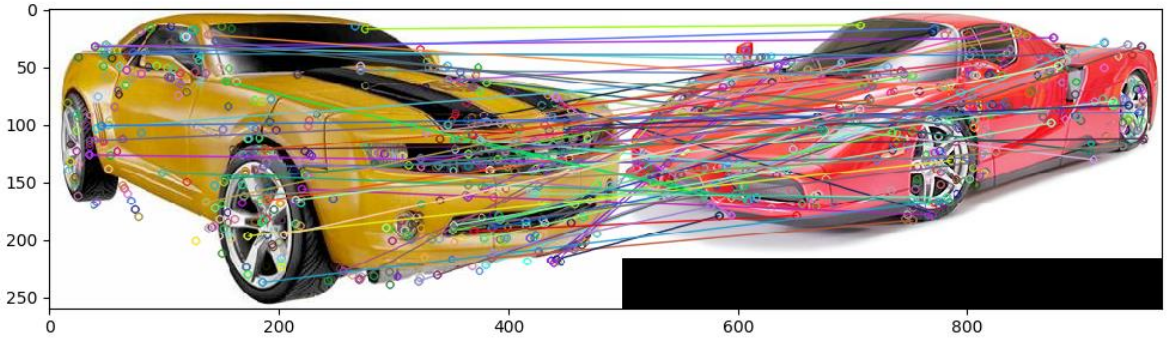
Bumblebee_sift_features



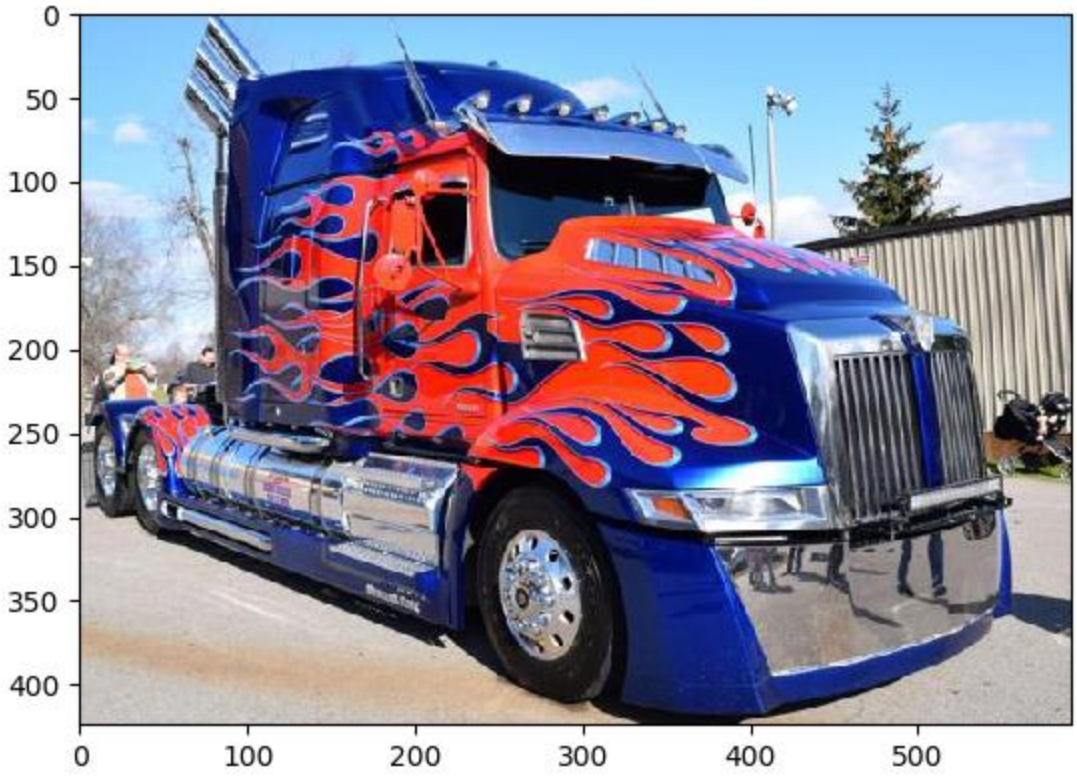
Bumblebee_ferrari_sift_matching_75toppoints



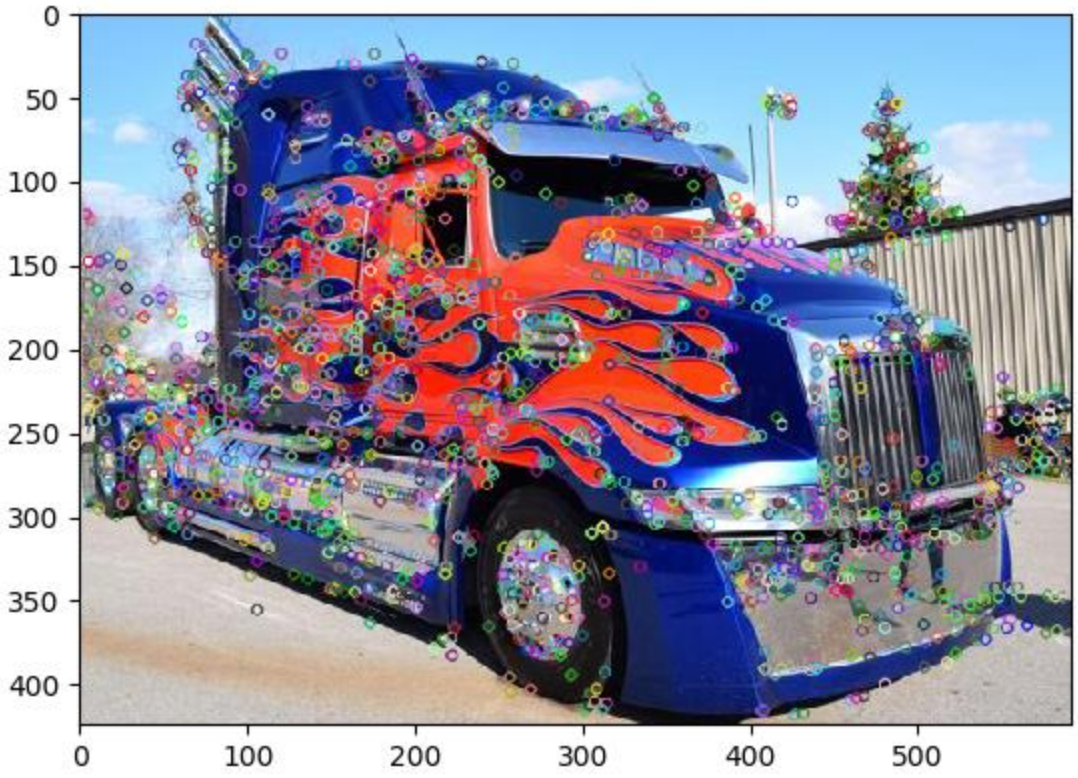
Bumblebee_surf_features



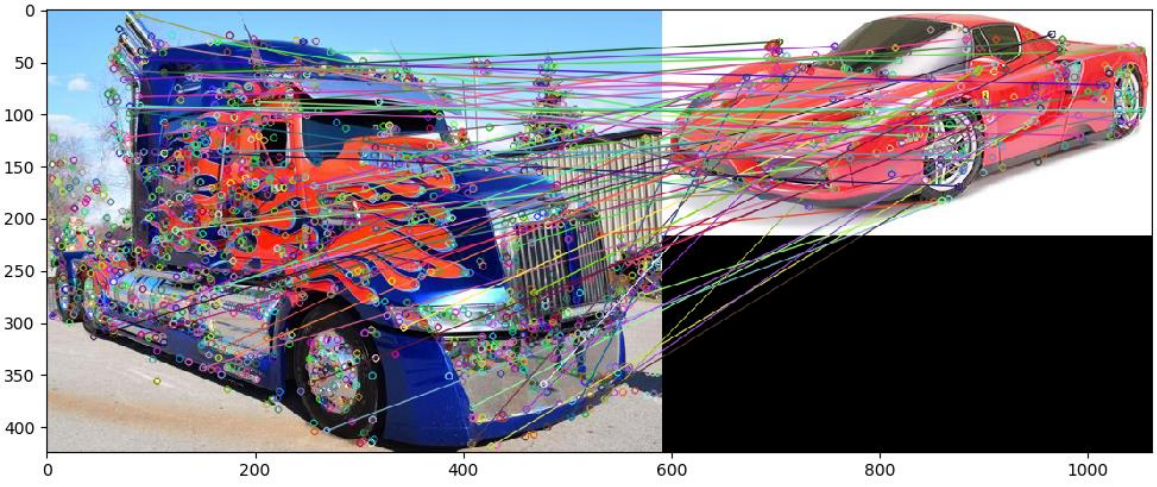
Bumblebee_ferrari_surf_matching_75toppoints



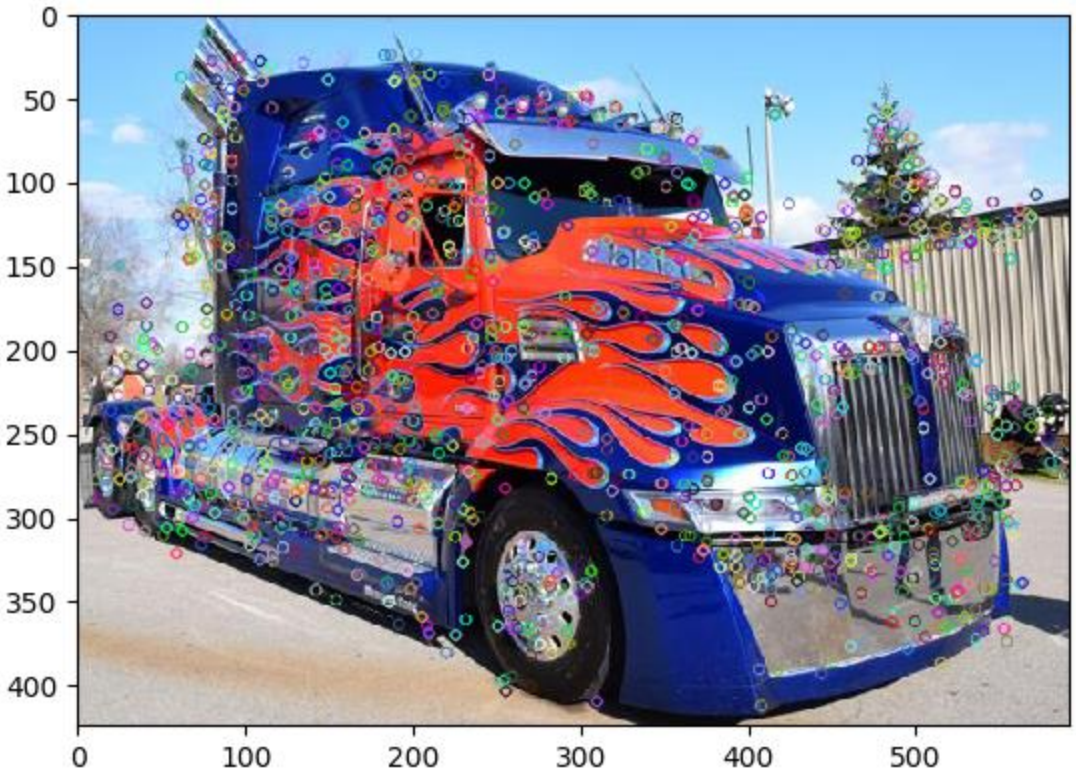
Optimus Prime Truck



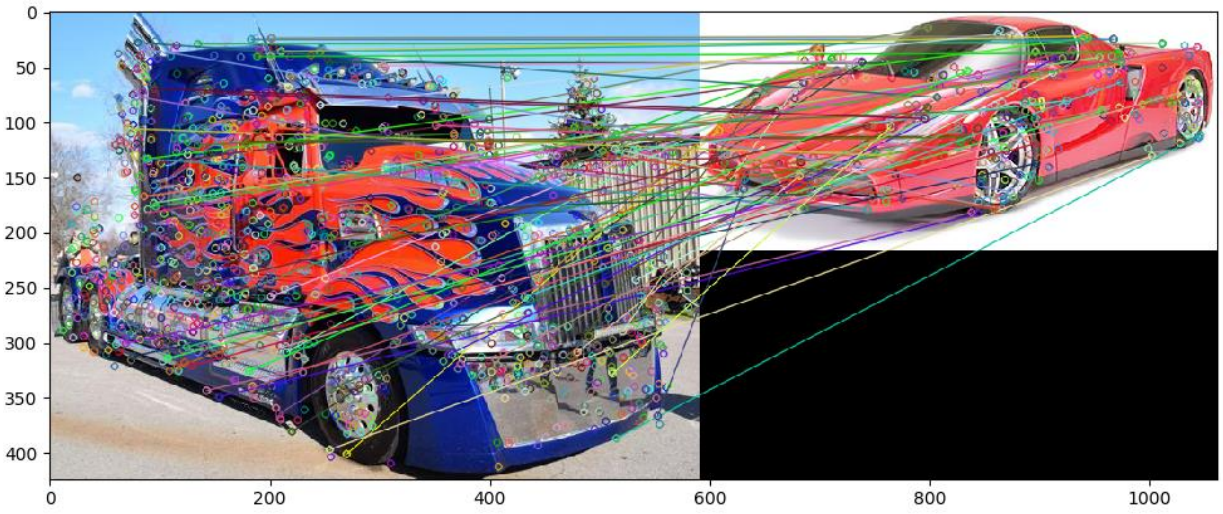
OptimusPrimeTruck_sift_features



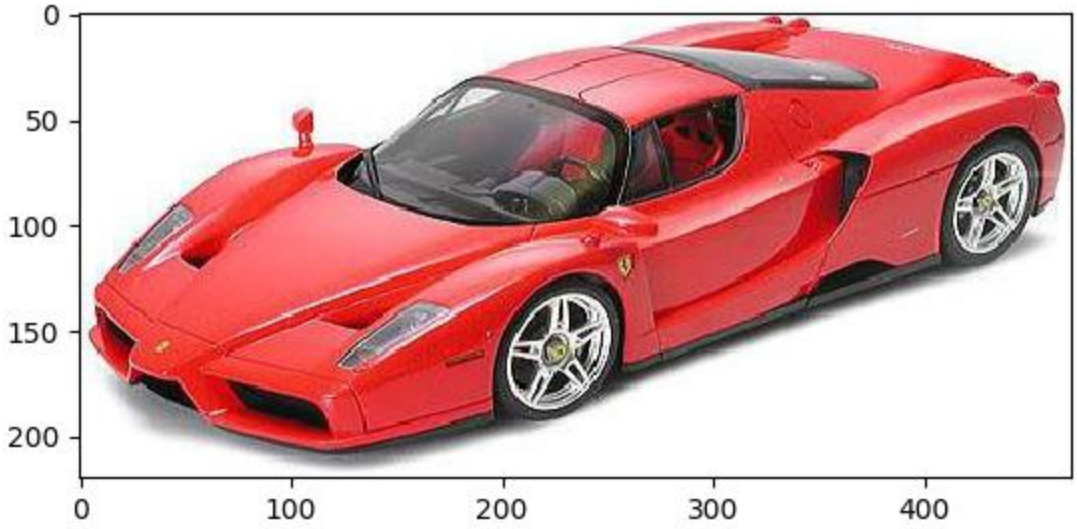
Optimus_ferrari1_sift_matching_75toppoints



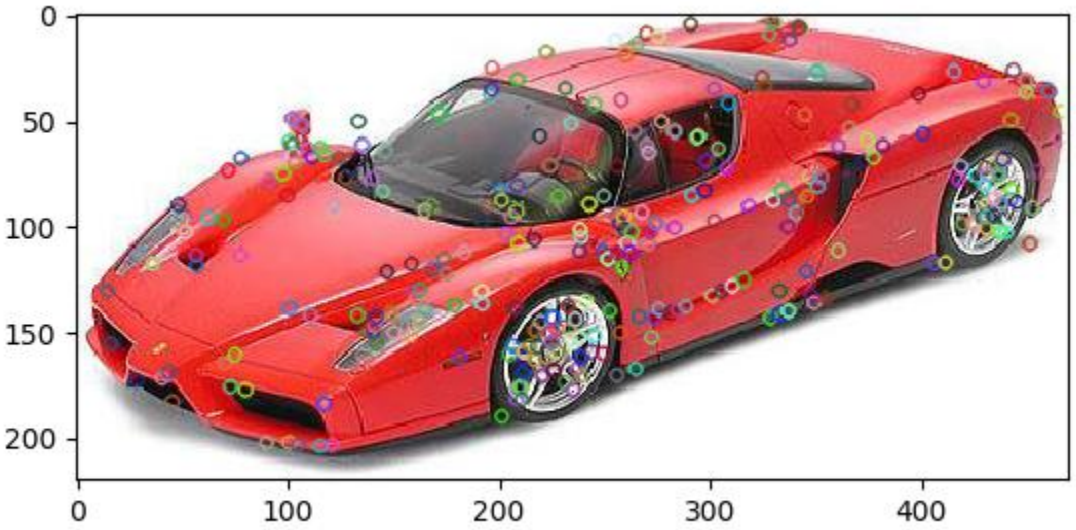
Optimus_surf_features



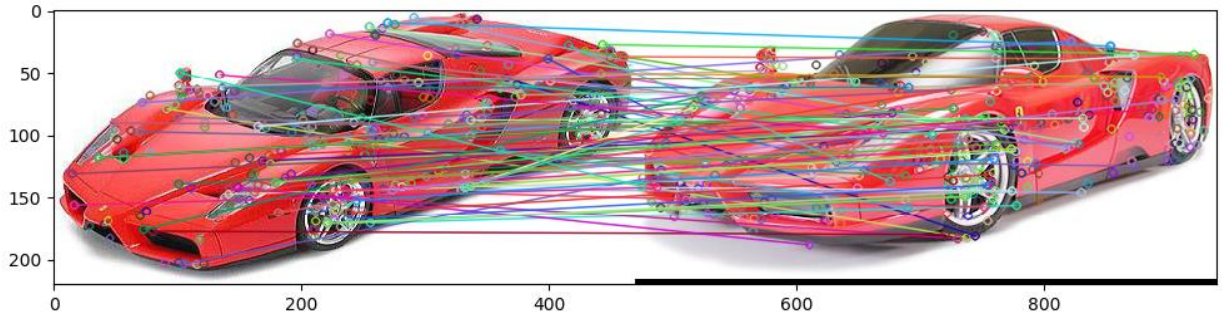
Optimus_ferrari1_surf_matching_75toppoints



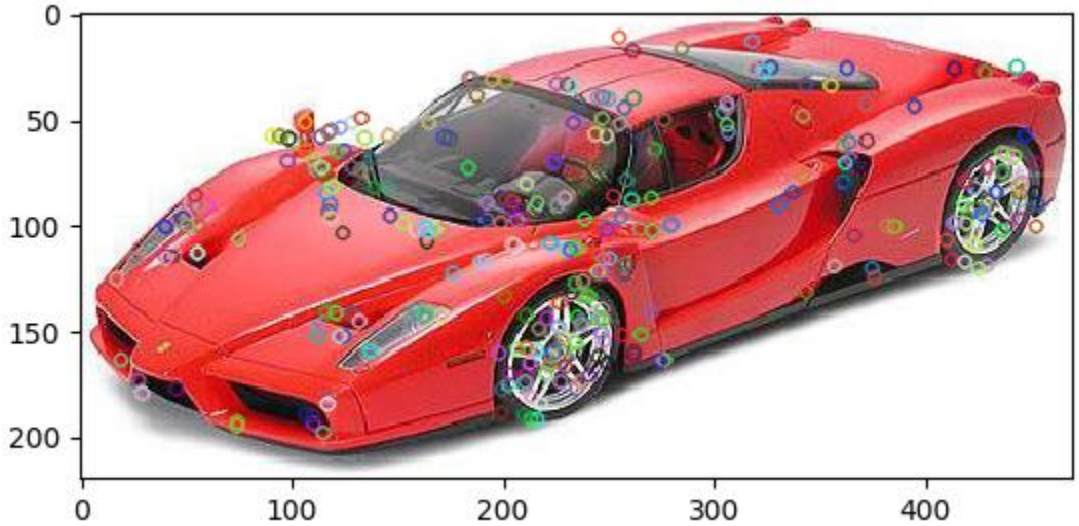
Ferrari2



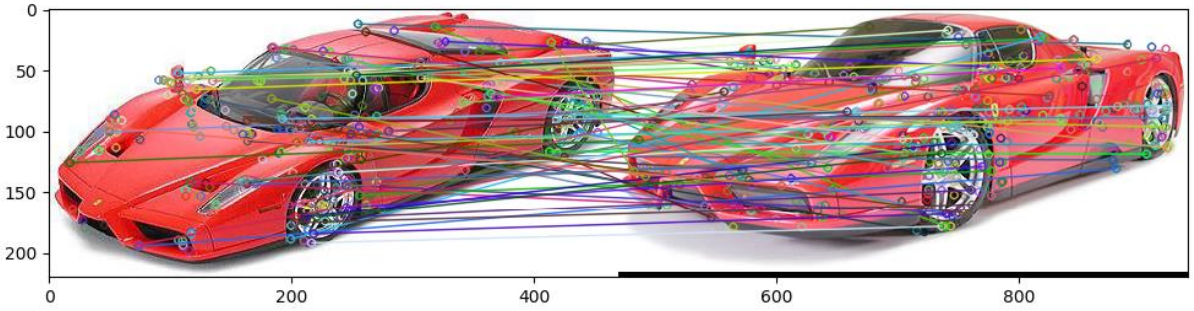
Ferrari2_sift_features



Ferrari1_ferrari2_sift_matching_75toppoints



Ferrari2_surf_features



Ferrari1_Ferrari2_surf_matching_75toppoints

Test Image	Bumblebee	Optimus Prime	Ferrari 1
Ferrari_2	0.17916636	0.15209861	0.04279352

ChiSquare result of the test image with the training images

3.4 Discussion

The SIFT and SURF features were extracted from the input images and the corresponding results are displayed above in the results. As can be seen from the results above, the SIFT has less points or in other words, the SURF has more points in interest. To better understand this we need to know the differences between them.

The scale space for SIFT is got by DoG convolved with different Gaussian filters, but in the case of SURF, various sized box filters are convolved with the image to form the scale space.

In the keypoint detection, SIFT utilizes non maximum suppression and gets rid of the outliers with the use of Hessian matrix. Whereas in SURF, it uses this Hessian matrix to determine the points which will be of interest.

In the case of SIFT, the overall orientation is calculated by taking the weighted orientation of the neighboring pixels. But SURF uses a sliding window orientation of size pi/3 to detect the dominant orientation of the Gaussian weighted Haar wavelet responses at every point of interest.

The keypoint descriptors for SIFT is achieve by taking the histogram of the orientation in a neighborhood of size 16x16,but in SURF, a grid with 4x4 square regions are considered and the wavelet responses are computed in relation to the orientation of the grid.

We can say that SURF computes interesting features with as much as one third of the time it takes by SIFT feature extractor. It provides better results than SIFT. This is due to the optimization used in SURF. SURF is way too fast compared to SIFT because of the way its built.

In Image matching, we have used Brute force approach, but this tries to match irrelevant points. This is solved by using a FLANN based matcher as it uses nearest neighbor technique.

Also the chisquare output as above in the table shows that Ferrari 1 and Ferrari 2 are similar as the distance is very small between them. The lower the value of chi square distance, the better it matches with the training image.

References

1. Ali Ghodsi, "Dimensionality Reduction A short tutorial", Department of Stats, University of Waterloo, Canada,2006.
2. David G. Lowe. "Distinctive Image features from Scale Invariant Key points", University of British Columbia,Vancouver, Canada.
3. P M Panchal, S R Panchal, S K Shah, "A comparison of SIFT and SURF", IJIRCCE, Vol 1, Issue 2, April 2013.
4. Mei Fang, GuangXue Yue, QingCang Yu, "The study of an application of Otsu's method in Canny Operator", ISIP'09, China.
5. Google.com/images
6. Wikipedia.org