EE-569 DIGITAL IMAGE PROCESSING HOMEWORK #3

Name: Amit Abhimanyu Pandey
Email: amitabhp@usc.edu

Issued Date: 26th February 2017
Due Date: 26th March 2017

Amit Abhimanyu Pandey
amitabhp@usc.edu

Problem 1: Texture Analysis and Segmentation (30 %)

1.1) Motivation

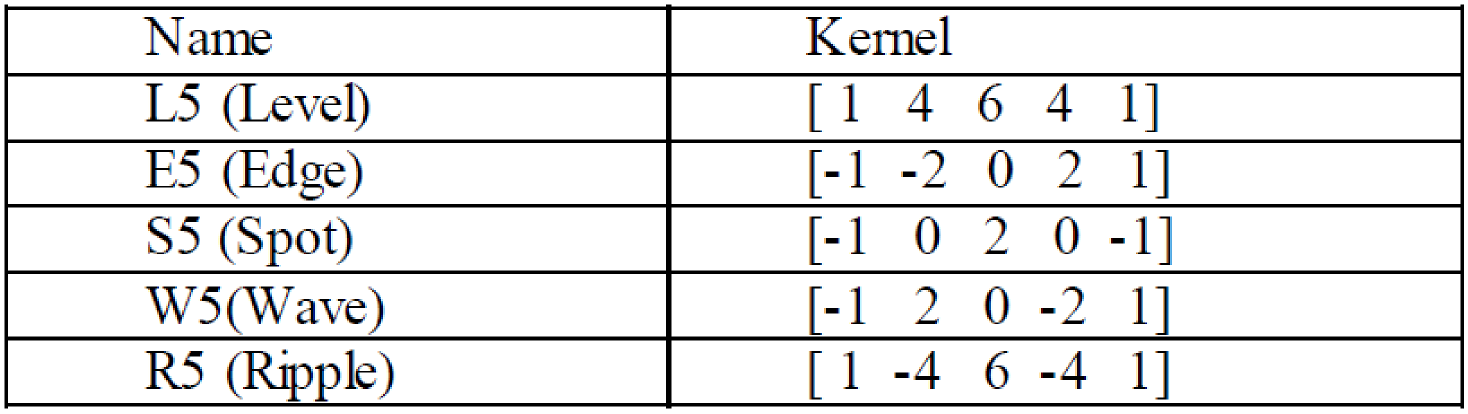
Texture Analysis and Segmentation is very important in many of the Computer Vision Applications. It was majorly used in the past for remote sensing applications from airplanes for defense applications. Texture in general means a lot of repeated patterns, but it not necessarily be repeated patterns. It is like a fundamental structure that obeys statistical properties. It has certain degree of randomness in it. It is important to study different textures as they are pertinent for image segmentation ahead. Various objects in an image can be segmented based on their unique textures. K-means clustering plays a very important role in texture classification and Image Segmentation. It is one the most primitive way which helps in analysis the textures based on the clustering of the features in the feature domain via K-means clustering.

So, main motivation behind doing texture analysis is to finally understand how can we classify different textures via K-means clustering after feature extraction via LAWS filters and apply it towards finally achieving Image Segmentation. So, the above question is divide in to two parts first part is based on classification of the textures and then testing the developed codebook after K-means with few similar textures. Then we need to extend the concept and use it to do image segmentation on the kitten image.

1.2) Approach

1.2.a) Texture Classification

There are many steps involved in texture classification which I will discuss in detail below. The main requirement to successfully implement texture classification is to extract good features. The requirement for the good features are that they shouldn't be scale, rotation or transform variant. Thus, we use LAWS filters banks which are generated from given below LAWS 1D kernel for 5*5 filters.



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]

*Figure 1: 1D kernel for 5*5 Laws Filter*

We are given twelve texture images of four different cluster types. These twelve images are going to be used for training and developing centroids of four different clusters. Where those four centroids will aid in testing other extra given six similar textures and classification of them via Euclidean distance.

To implement above firstly we need to generate 5*5 laws filters from the five 1D filter kernels, eventually various combinations between the filter kernels we will 25 such filters. Each of the filter matrix has specific frequency response that filters out specific frequencies. The frequency response of these filters is discussed later. Each 1D filter kernel is combined with another by taking the tensor product to produce a 5*5 filter.

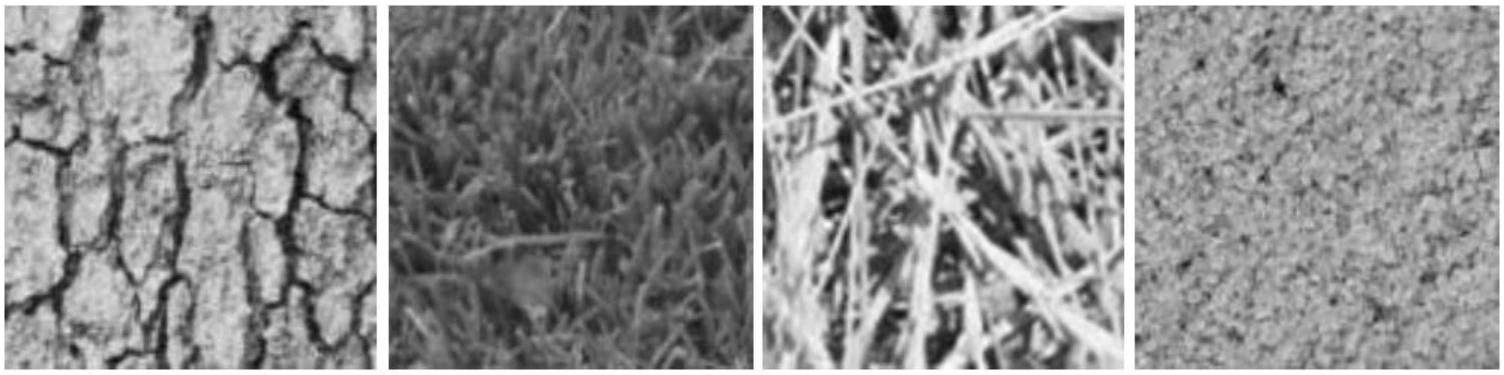


Figure 2: Four Different Class of Textures

Algorithm for Texture Classification:

- After the generation of the 2D filters, we need to subtract the global mean from each pixel of the all the twelve images read. This ensures that we don't have unnecessary high feature values, thus eliminating the low frequency bias. This is pre-processing step before feature extraction.
- After the above step lets now consider that we have just first texture with us of the tree/bark. We now need to convolve the bark texture with each of the 25 filters. Which will then give me 25 new 2D matrices consisting the features generated after convolution. Likewise, each training texture will give us 25, 2D matrices which will contain the convolution results in terms of features. So now we need to calculate energy of each filter response of for let's say the first texture by the following given formula.

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

- So now each filter response will give one energy value, so eventually I will have 25 energy values for one training texture. These energy values need to be clubbed together to form a one 25D feature vector per training texture.
- Thus, eventually we will have a matrix of size 12*25 where each row corresponds to the one 25D feature vector of each training textures. Now this information can be used for clustering and classification.
- Currently we have four different class of textures, this can be said based on visual inspection. That from the set of 12 images a group of 3 belongs to one class of textures. But the program doesn't know which texture belongs to which class. Thus, via K-means we can classify the 12 textures in to four different class.
- Before K-means we also normalized the 12*25 feature vector, because the L5L5 filter combination as not having zero mean dominated in terms of energy, thus we normalize 12*25.
- We were asked to do K-means twice, first considering just the 12*25 feature vectors and then via PCA reducing the dimensionality to achieve 12*3 feature vector matrix and compare both the results. Concept of PCA is explained later.
- So, I used index numbers as means of assigning labels to four different classes. If a texture belongs to bark class it will get an index number of 0, if it belongs to say grass it will an index number of 1 and so on for rest classes.
- The K-means clustering runs until the four centroids don't change much and are within certain tolerance range as K-means is an iterative algorithm. For K-Means clustering as we know there are only four classes or four different classes of textures, we give K=4. Thus, as now K=4, we need initial four centroids. There are two ways of assigning the or taking the initial centroids. Firstly, we can take four points randomly in the 25D space, or we

can take the feature vector of each class by visual inspection and take them as centroids to improve the convergence.

- After discussing the convergence issue with the TA, they recommended us to consider the later. Thus, after K-means clustering we will eventually get 4 best centroids representing 4 different classes.
- The accuracy of the clustering can be tested in two ways. Firstly, by just checking whether 3 textures belonging to each class are getting properly classified or not. And after that we have feature vectors from the six testing images. By visual inspection we know how it should get classified, thus via program we need to find the Euclidean distance between each feature vectors and each centroid. And from which ever centroid the distance is minimum we can classify it belonging to that class.

Note: Final testing was done by two methods Unsupervised and Supervised.

- In supervised learning the final codebook or 4 best centroids were selected by taking the average of the feature vectors of the three textures belonging to same class by visual inspection.

1.2.a) Texture Segmentation

So, in this part of the question we were given a kitten image shown below and we had to identify different textures and show them separately in one image by colors representing different textures.

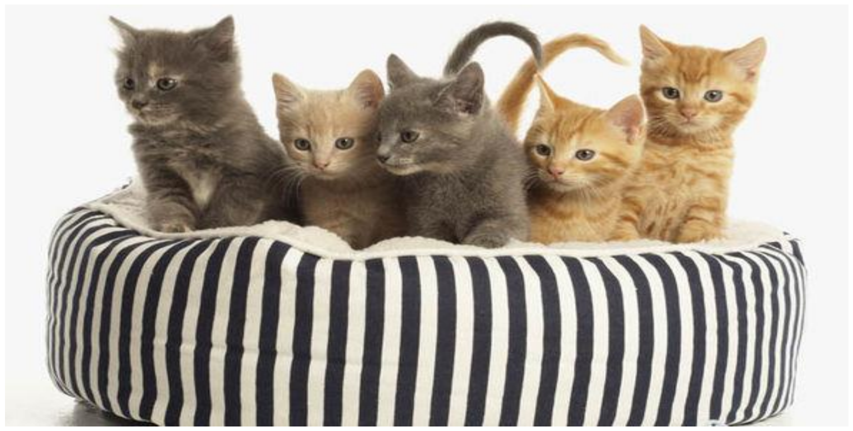
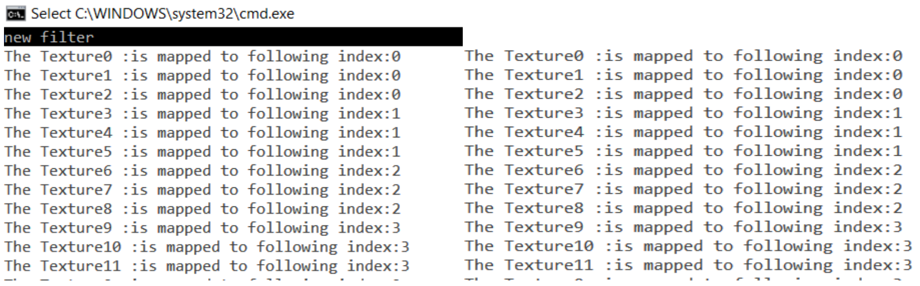


Figure 3: Kitten Image for Texture Segmentation

- Process pretty remains the same. The only thing that changes is the idea behind the feature vector generation. Previously, our feature vectors were calculated by calculating the energy of the entire image after convolution. But here we will deal in terms of pixels.
- Such that each pixel will have its own feature vector. Thus, we will firstly convert the RGB image to gray and then repeat the entire process and generate a stack of 25 convolved images one above the other.
- Now we will consider a window of 13*13 or 15*15 and move the patch over the expanded convolved images and replace the center pixels with the average local energy within that window.
- Thus, again we will get a stack of 25 images which are representing the feature vectors of each pixels.
- After which we need to apply K-means clustering and develop a 2D map representing different pixels getting mapped to different indexes i.e. classes.
- And later with the help of the map array developing a color image representing different textures with different colors in the image.

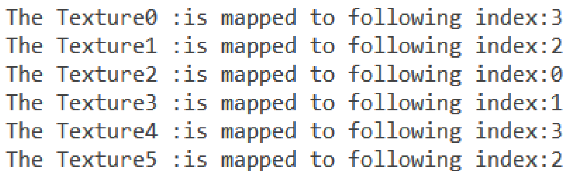
1.3) Results

1.3.a) Texture Classification



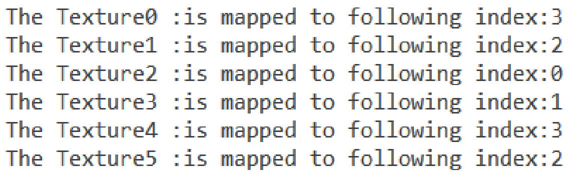
```
Select C:\WINDOWS\system32\cmd.exe
new filter
The Texture0 :is mapped to following index:0      The Texture0 :is mapped to following index:0
The Texture1 :is mapped to following index:0      The Texture1 :is mapped to following index:0
The Texture2 :is mapped to following index:0      The Texture2 :is mapped to following index:0
The Texture3 :is mapped to following index:1      The Texture3 :is mapped to following index:1
The Texture4 :is mapped to following index:1      The Texture4 :is mapped to following index:1
The Texture5 :is mapped to following index:1      The Texture5 :is mapped to following index:1
The Texture6 :is mapped to following index:2      The Texture6 :is mapped to following index:2
The Texture7 :is mapped to following index:2      The Texture7 :is mapped to following index:2
The Texture8 :is mapped to following index:2      The Texture8 :is mapped to following index:2
The Texture9 :is mapped to following index:3      The Texture9 :is mapped to following index:3
The Texture10 :is mapped to following index:3     The Texture10 :is mapped to following index:3
The Texture11 :is mapped to following index:3     The Texture11 :is mapped to following index:3
```

Figure 4: The classification of 12 Training Textures after 2 iterations (without PCA/unsupervised).



```
The Texture0 :is mapped to following index:3
The Texture1 :is mapped to following index:2
The Texture2 :is mapped to following index:0
The Texture3 :is mapped to following index:1
The Texture4 :is mapped to following index:3
The Texture5 :is mapped to following index:2
```

Figure 5: Testing Textures Classification (without PCA/unsupervised).



```
The Texture0 :is mapped to following index:3
The Texture1 :is mapped to following index:2
The Texture2 :is mapped to following index:0
The Texture3 :is mapped to following index:1
The Texture4 :is mapped to following index:3
The Texture5 :is mapped to following index:2
```

Figure 6: Testing Textures Classification (without PCA/ Supervised).

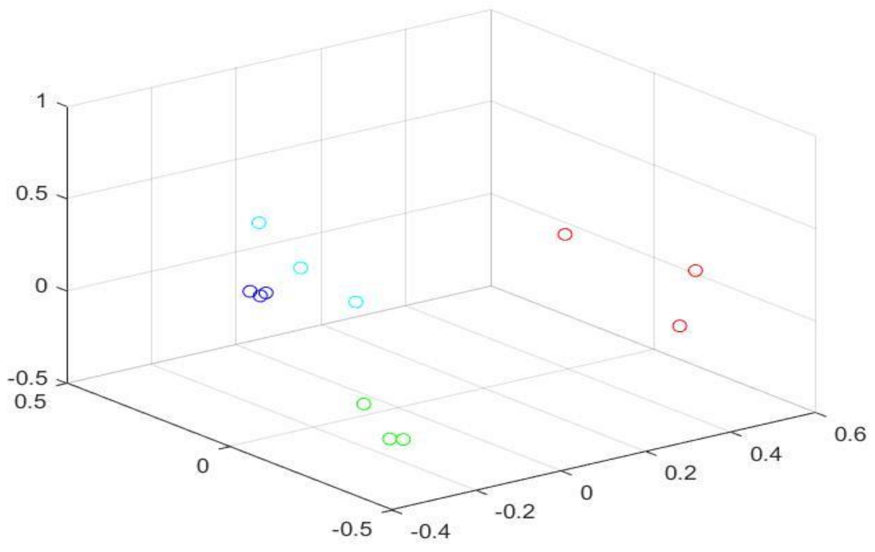


Figure 7: Scatter Plot after K-means with the PCA data in Matlab.

Note: Same results for the centroids after dimensionality reduction via PCA.

P.S. - 0: Bark Texture, 1: Sand Texture, 2: Grass Texture, 3: Straw Texture.

1.3.b) Texture Segmentation

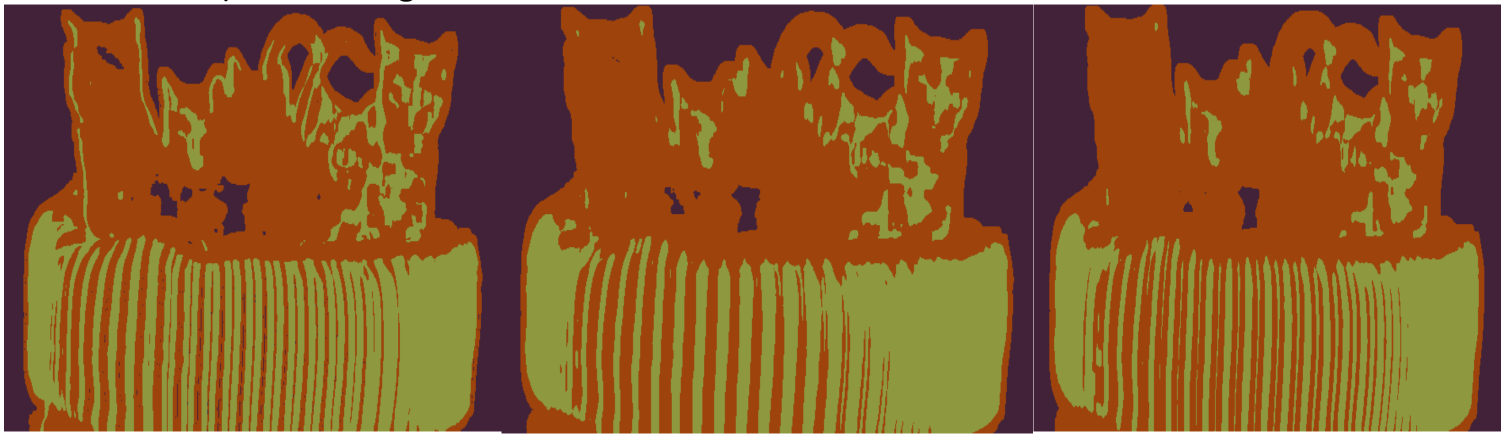


*Figure 8: K=3 and window Size 9*9,13*13,15*15 respectively*

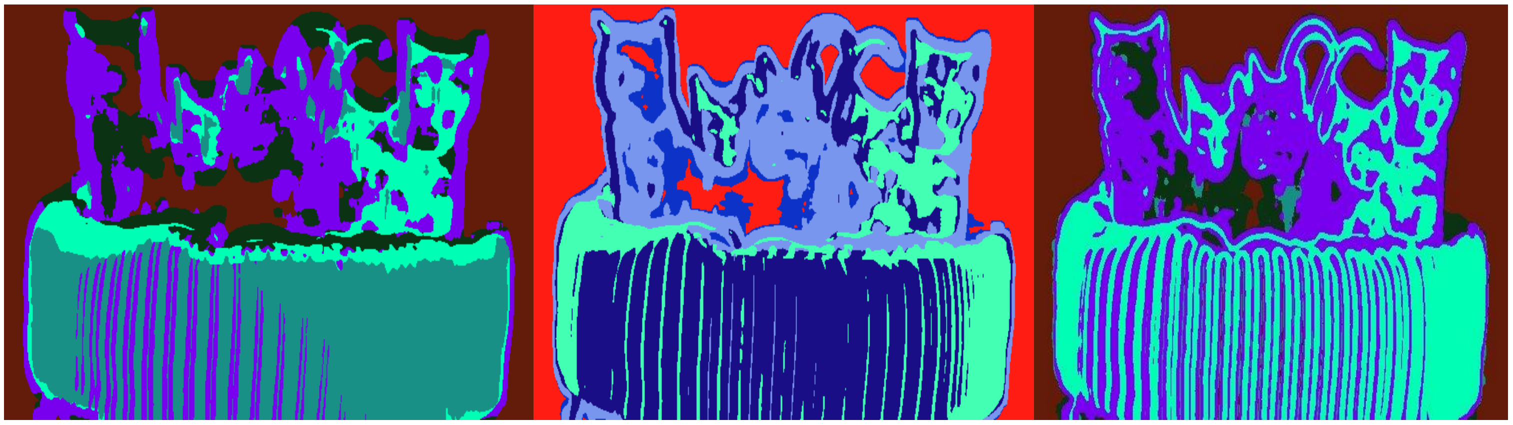


*Figure 9: K=5 and window Size 9*9,13*13,15*15 respectively*

Note: for color coding I am using srand() function in c++.

1.4) Discussion

1.4.a) Texture Classification

So, in this we need to discuss the effectiveness of the PCA for the K means. When I ran the code with the PCA feature code vectors the compilation time was reduce. What PCA does is it returns the important features which are discriminant. And thus, we reduce feature vector from 12*25 to 12*3. K-means clustering time and iteration improves when using PCA, there is not much difference in the output. So, we can comment that the PCA doesn't improve the classification but it improves the time required for K-means clustering.

- For Texture 2 and Texture 4

L5L5	0.416621	-1.04506	1.461681
L5E5	0.564991	-0.99444	1.559431
L5S5	0.846029	-0.90078	1.746805
L5W5	1.71059	-0.68546	2.396054
L5R5	2.29334	0.08957	2.20377
E5L5	0.293355	-0.70857	1.00192
E5E5	0.192543	-0.59577	0.788314
E5S5	0.536292	-0.46998	1.006267
E5W5	1.53992	-0.08793	1.627852
E5R5	2.13676	0.51388	1.62288
S5L5	0.691978	-0.28856	0.980536

S5E5	0.658685	-0.16852	0.827209
S5S5	1.06523	0.020981	1.044249
S5W5	1.83497	0.430853	1.404117
S5R5	2.06894	0.674587	1.394353
W5L5	1.39192	0.348331	1.043589
W5E5	1.47897	0.508804	0.970166
W5S5	1.75821	0.623572	1.134638
W5W5	2.02805	0.739728	1.288322
W5R5	2.05057	0.768462	1.282108
R5L5	1.77618	0.719958	1.056222
R5E5	1.84931	0.841854	1.007456
R5S5	1.90559	0.888814	1.016776
R5W5	1.99942	0.848167	1.151253
R5R5	2.04851	0.780154	1.268356

L5W5: gives the strongest discriminant power

E5E5: gives the weakest discriminant power.

This changes texture to texture and for every new set we can get different combination of the filters for classification.

1.4.b) Texture Segmentation

So, it can be seen from the segmented image results that, the segmentation improves with the increment in the window size as well as with up to certain extent with the K value. I tried with the K=7 as well but for that I was getting blobby images. The increment in the window size helps in identifying the textures better and thus eventually helps in better segmentation.

c) Further Improvement (Advanced: 10%)

For further improvement, we were given four options, from which I tried the first option. Took K=3 and further applied texture segmentation for K=5, but the output that I got was very blobby and I couldn’t make out the final kittens in the image.

So, I went ahead and did the feature reduction via PCA and given below is the output of the PCA and K-means clustering for K=5 and window size 15*15. The output doesn’t change much with the PCA feature reduction but yes the computation time improved for K-means clustering

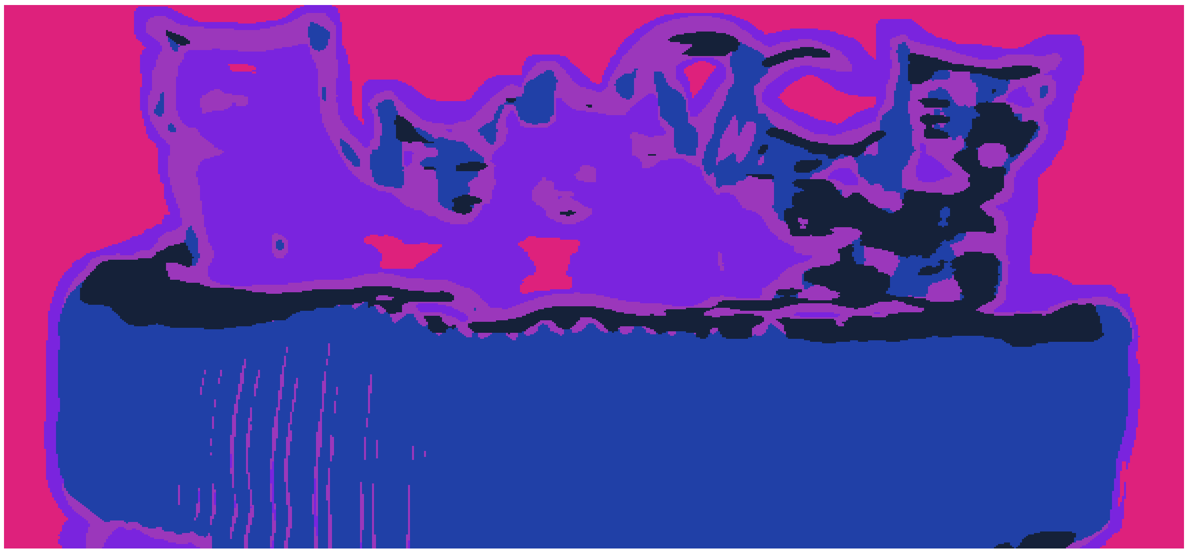


*Figure 10: Segmentation K=5 for 15*15*

Problem 2: Edge and Contour Detection (40 %)

2.1) Motivation

Edges can be generally defined as the pixels where there is sudden change in the grayscale values. Edge detection is important in many computer vision applications. It can be treated as a preprocessing stage, in applications such as object tracking, object recognition and so on. It has been a challenging problem to solve because by an efficient edge detector there are lots of things needed to consider such as colors variations, texture, symmetry etc. before commenting whether a pixel is an edge pixel/point or not.

Here in this question we asked to use Canny Edge Detector and Structured Edge Detector on two images given below and compare their strength/weaknesses. Canny edge detection is an extension of the Sobel edge detection. It uses the strengths of sobel edge detector and adds an extra layer of thresholding i.e. double hysteresis thresholding to remove weak unconnected edges in the image. It is one of the fast edge detector and can be used in the applications where we need parallel computing. Whereas Structured Edge detection makes use of the structure present in the local image patch to train the random forest classifier. It was done to promote efficient edge detection in real time.

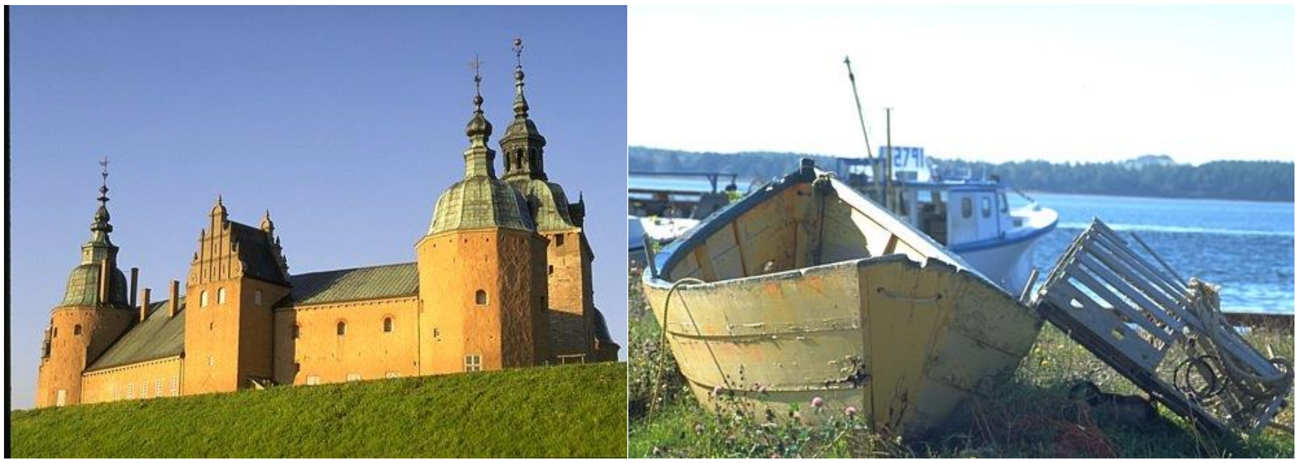


Figure 11: Input Images for Edge Detection

2.2) Approach

2.2.a) Canny Edge Detector

It is a multi-stage edge detector; it was developed John Canny and he developed this to solve two main motivation in edge detection. His first aim was to detect all edges in the image with very low error rate and Secondly, localize the edge points to minimize the distance between edges and center of the true edge.

So, the algorithm for implementing Canny Edge detection is quite straight forward. It takes the work ahead from sobel edge detector after blurring the images via Gaussian filter. The Gaussian filter helps in reducing the noise which is pertinent for avoiding false edges. We implemented the Canny Edge Detection in Open CV.

- After smoothing the image via convolution with Gaussian Filter of the size 5*5 and then we gave it to the Open CV Canny Operator. Threshold was given based on brute force method. There's also Otsu's thresholding formula, but I didn't employ that.
- Sigma value I choose 0.33 as I found that it said to be the most optimal value of the sigma.
- In canny Edge Detection algorithm after Sobel Filtering, Non-Maximum Suppression is done to remove the outlier's. For the generation of the final edge map we use two threshold values.

- Higher and Lower Threshold values are used in the following ways, if the point is above the higher threshold we classify it as edge point. If it is lesser than the lower threshold, then we classify it as non-edge point. If at all it lies in between the both and is a connected edge point we say it is an edge point otherwise not.

2.2.b) Structured Edge Detector(SE)

It is a very highly sophisticated algorithm that is used for edge detection. It was developed by Piotr and Zitnik. It is based on random forests classification and involves the decision trees in the training phase. It is very fast and it is quite efficient as it takes in to consideration the vote of confidence from the various trees in the forest before giving a decision.

It considers a patch around the pixel and determines the likelihood of the pixel being an edge or not. It finally generates a probability edge map indicating whether a pixel is edge or not. The patches are classified into sketch tokens and those set of sketch tokens represent a variety of local edge structures. The classification is done with the help of the voting provided by the trees in the random forest after training. The training is done by considering a patch from the input and ground truth images along with forming the structured output of whether a combination should be classified as an edge or not. Training is very pertinent for the whole structured edge algorithm to give proper results.

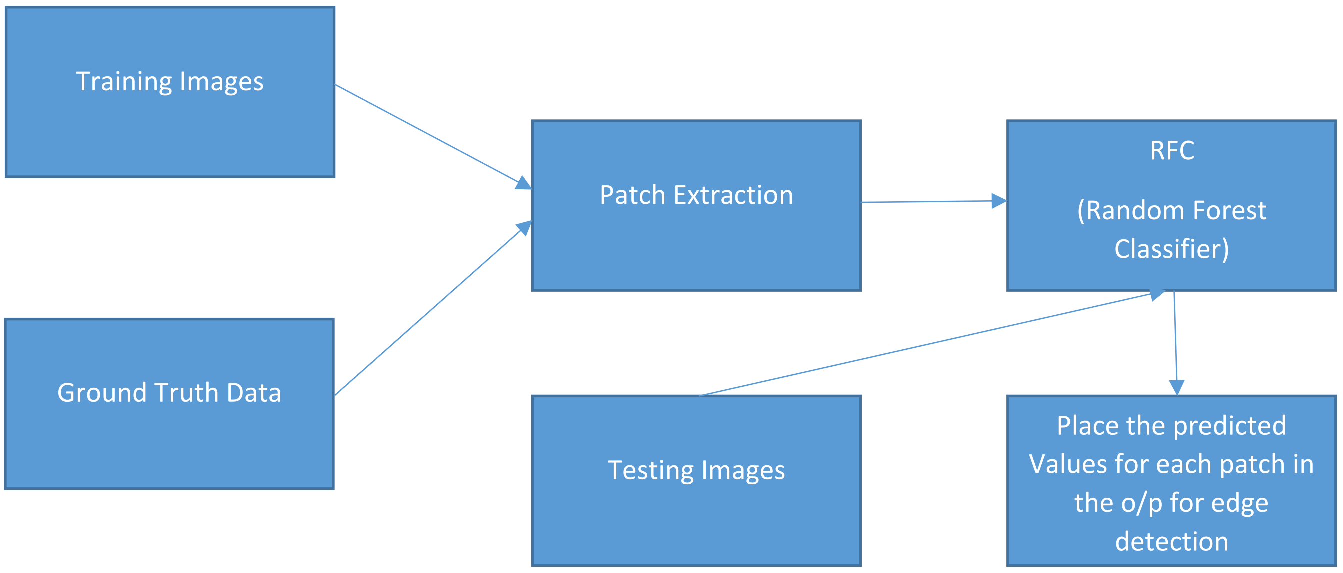
So, the main crux behind the learning of the trees in the random forest is that they run on the divide and conquer technique. In the decision tree the input is entered from the root node. Then the input traverses down the tree in an iterative manner till it reaches leaf node. At each node, it checks decision parameters if the classification is pure then it has reached the optimal decision i.e. the leaf node and then it should stop, if not then move forward. Basically, the decision parameter is called the split function.

$$h(x, \theta_j) \in \{0, 1\}$$

If $h(x, \theta_j) = 0$, then node j sends x left, otherwise right. The output of the tree on the input x is the prediction stored at the leaf reached by x, which may be a target label y belonging Y.

A set of decision trees are trained independently to find the parameters in the split function that result in a good split of data. Entropy drops at each node, to improve upon the information gain splitting parameters should be chosen wisely. For the training of the decision tree we use bagging technique, if we have a training set and few class labels, we can randomly sample the training set and use that training set to train the decision trees by iteratively sampling the training set. The leaf nodes contain the class labels after training and eventually we can give the testing images to check the classification.

Flow Chart:


Training Images
Ground Truth Data
Patch Extraction
RFC
(Random Forest Classifier)
Testing Images
Place the predicted Values for each patch in the o/p for edge detection

Also for implementation, we used online available codes, models and database.

2.2.c) Performance Evaluation

Perform quantitative comparison between different edge maps obtained by different edge detectors. The goal of edge detection is to enable the machine to generate contours of priority to human being. For this reason, we need the edge map provided by human (called the ground truth) to evaluate the quality of a machine-generated edge map. However, different people may have different opinions about important edge in an image. To handle the opinion diversity, it is typical to take the mean of a certain performance measure with respect to each ground truth, e.g. the mean precision, the mean recall, etc.

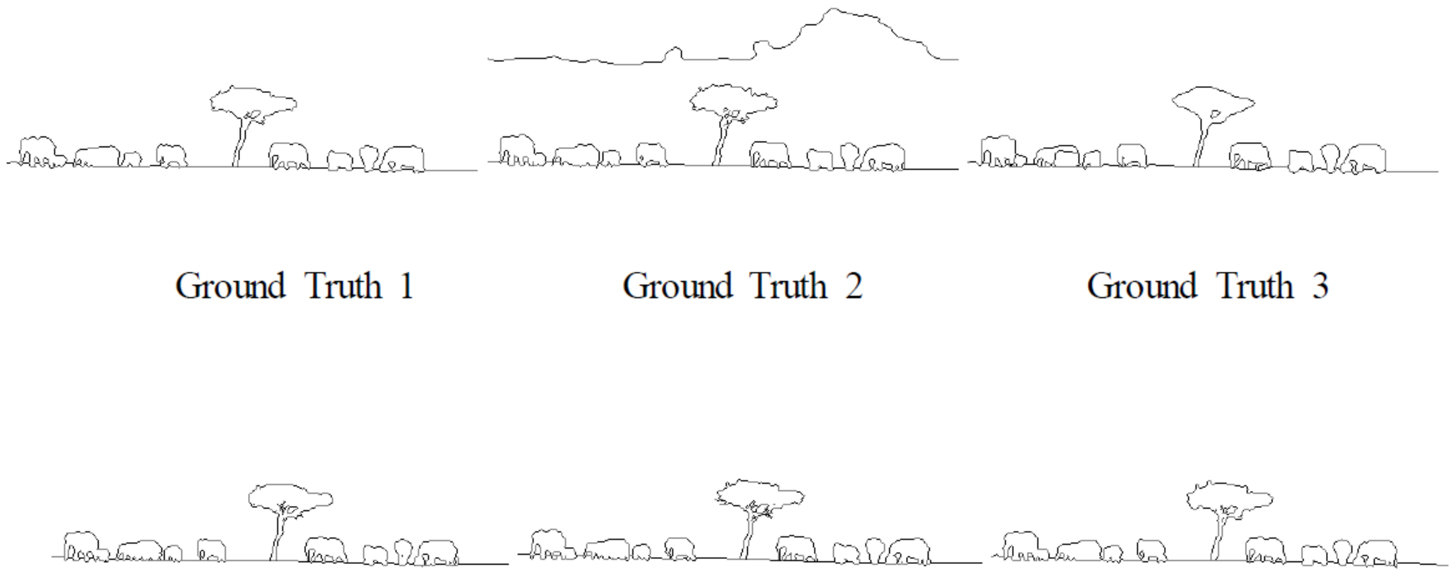


Figure 12: Example of Ground Truth Data

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

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

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

A higher F measure implies a better edge detector.

2.3) Results

2.3.a) Canny Edge Detector Results:

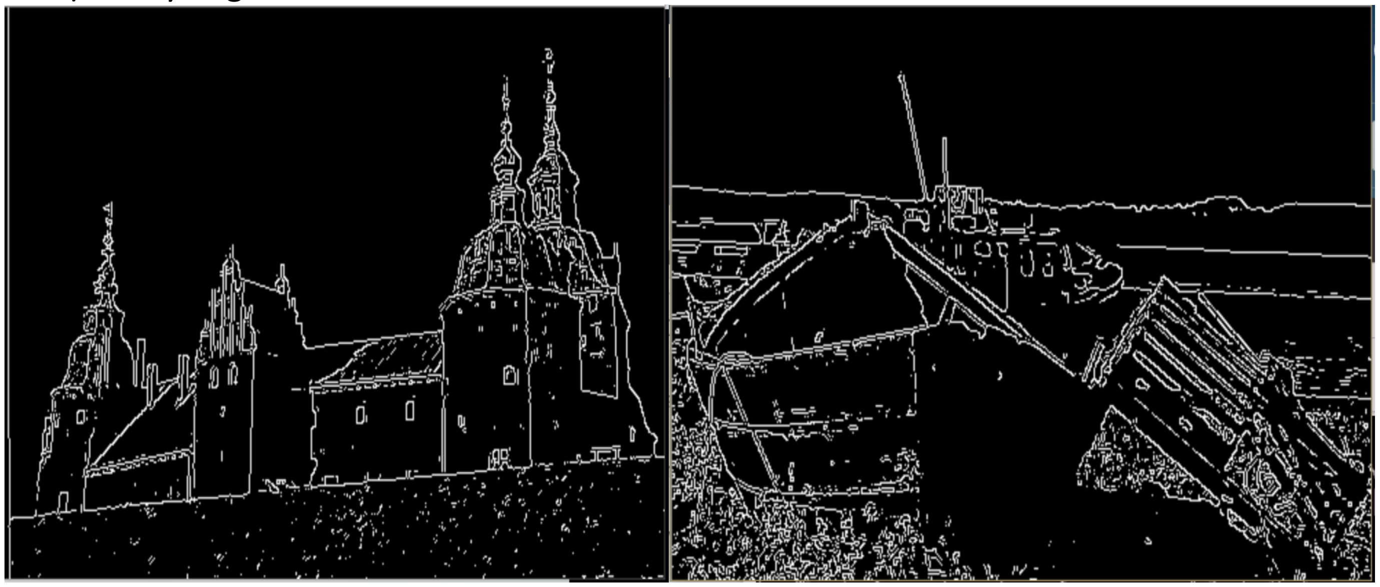


Figure 13: Castle and Boat Edge Map:(230,240)

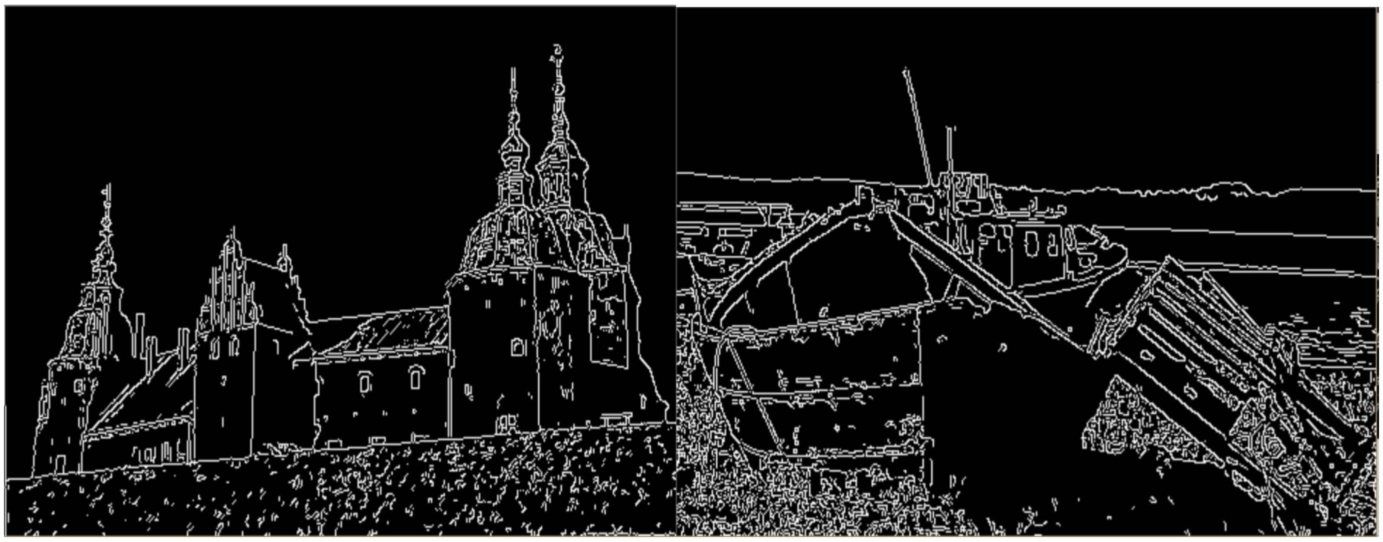


Figure 14 Castle and Boat Edge Map:(150,210)

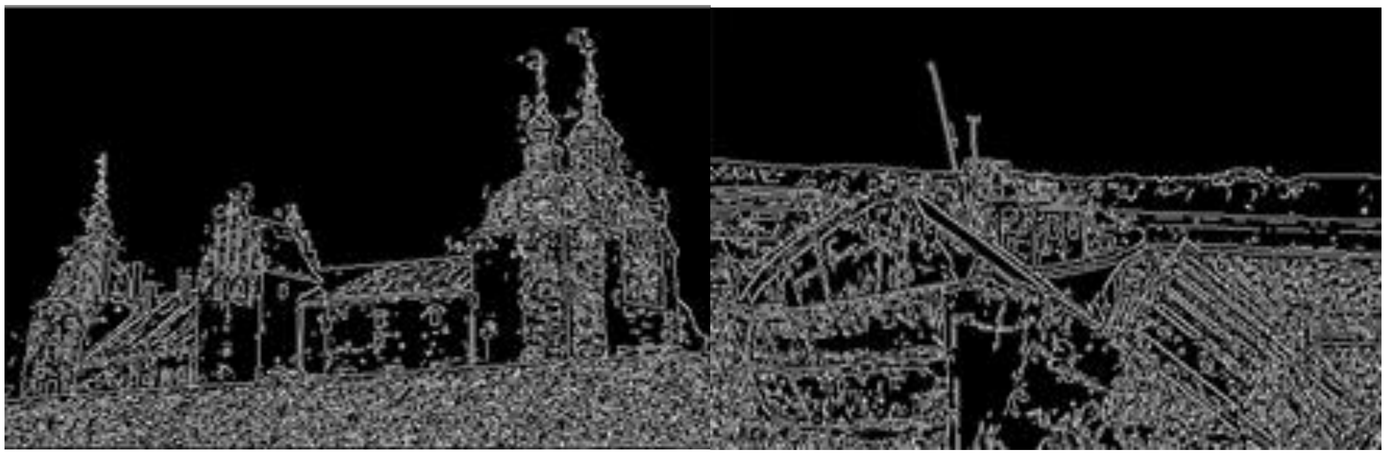


Figure 15:Castle and Boat Edge Map:(50,70)

2.3.b) Structured Edge Detector Results:

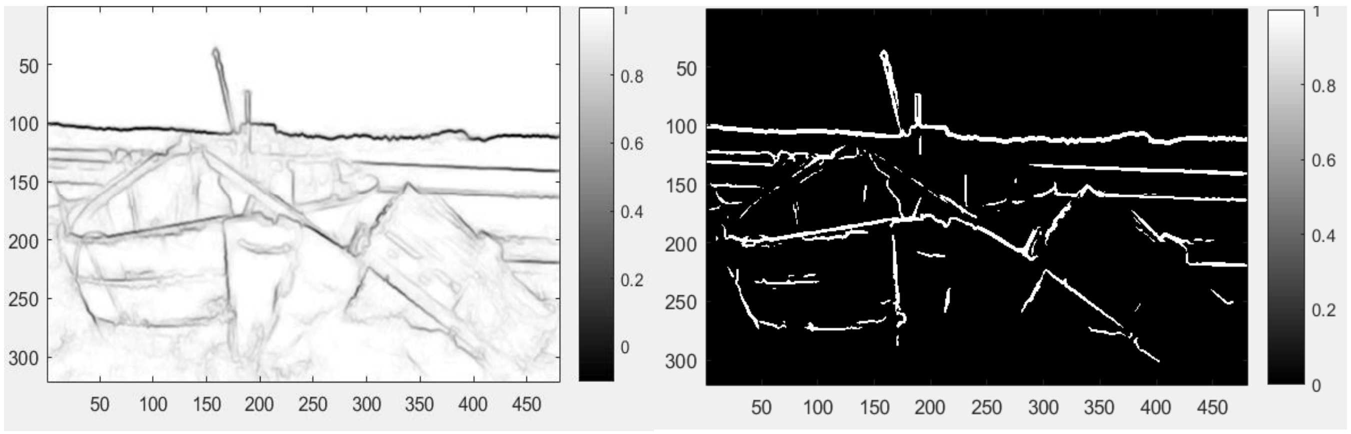


Figure 16: Boat Probability and Binary Edge Map (>0.25)

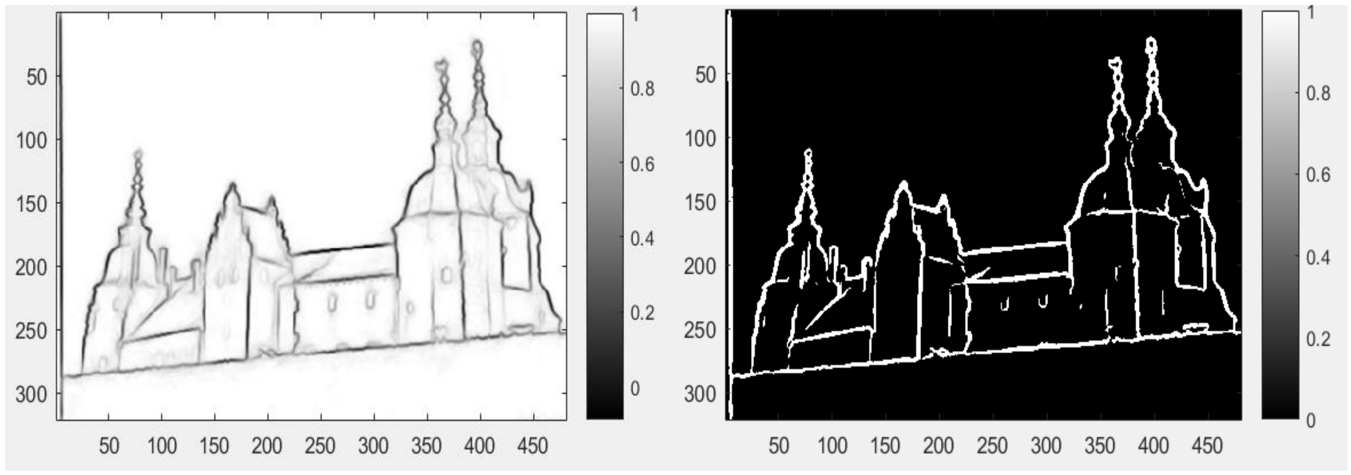


Figure 17 Castle Probability and Binary Edge Map (>0.25)

2.3.c) Performance Evaluation

Canny Detector Results Tabulated:

Image	Precision	Recall	F_Score
Boat_gt1	0.08992	0.31107	0.13951
Boat_gt2	0.03759	0.065095	0.06509
Boat_gt3	0.08288	0.28004	0.12791
Boat_gt4	0.05552	0.28222	0.09279
Boat_gt5	0.02585	0.18551	0.04538
Boat_gt6	0.03586	0.17938	0.05977
Mean	0.0546	0.2468	0.0884

Table: Mean, Precision, Recall and F_Score (threshold: 40,60)

Image	Precision	Recall	F_Score
Boat_gt1	0.03103	0.01628	0.02136
Boat_gt2	0.03131	0.02837	0.02977
Boat_gt3	0.05766	0.02832	0.03799
Boat_gt4	0.04846	0.03407	0.04001
Boat_gt5	0.04523	0.04138	0.04322
Boat_gt6	0.02343	0.01690	0.01963
Mean	0.0395	0.0276	0.0320

Table: Mean, Precision, Recall and F_Score (threshold: 30,250)

Image	Precision	Recall	F_Score
Castle_gt1	0.0820	0.1827	0.1132
Castle_gt2	0.0799	0.1484	0.1038
Castle_gt3	0.0895	0.0952	0.0923
Castle_gt4	0.0879	0.0797	0.0836
Castle_gt5	0.1002	0.1187	0.1087
Castle_gt6	0.1628	0.1482	0.1552
Mean	0.1004	0.1289	0.1095

Table: Mean, Precision, Recall and F_Score, (threshold: 30,212)

Structured Edge Detector Results Tabulated: **(threshold :0.27)**

Image	Precision	Recall	F_Score
Boat_gt1	0.6216	0.4569	0.3612
Boat_gt2	0.592	0.579	0.5665
Boat_gt3	0.6332	0.4574	0.35
Boat_gt4	0.6665	0.5942	0.536
Boat_gt5	0.4953	0.5151	0.5365
Boat_gt6	0.6553	0.5783	0.5175
Mean	0.61065	0.53015	0.479283

Image	Precision	Recall	F_Score
Castle_gt1	0.5218	0.6665	0.9223
Castle_gt2	0.5218	0.6694	0.9336
Castle_gt3	0.7125	0.7711	0.8401
Castle_gt4	0.8126	0.8024	0.789
Castle_gt5	0.7044	0.773	0.8563
Castle_gt6	0.7741	0.788	0.8024
Mean	0.674533	0.745067	0.857283

Table: Mean, Precision, Recall and F_Score, (threshold:0.27)

2.4) Discussion

2.4.a) Canny Edge Detector

There are few things worth discussing pertinent to canny edge detection as follows:

1.) When we have a look at above given canny edge detection results for different threshold, we comment few things.
   - The gradient magnitude with values above the upper threshold are considered strong edges and these edges are nicely detected by canny. Edges below the lower threshold are eliminated.
   - So, for few threshold values like 50,70, the lower threshold and upper threshold both are on the lower range thus we get so many edges which are not required and they occlude the object edges in it. This

is because the lower threshold value is towards the lower range and the difference between both threshold is quite less, thus canny following its algorithm will consider lot many edges.

- Whereas when we see the resulting image having a threshold 230,240, it has better edge detection because we are now in the higher range and difference is less. But as the difference in the higher range decreases chances of missing strong edges increases.
- Thus, there is a tradeoff between the range and difference in the canny edge detection as it is not intelligent.
- Castle Image performs better as compared to boat image with canny edge detection and it can be seen from the above results. The Castle Image has less distraction in the image whereas the Boat Image has more objects and distraction.
- Yes, we can tell the objects location but it is quite difficult because by brute force method understanding and selection of the threshold values are difficult.

2.) We can tune the parameters for edges on the perfect image, as in only the object in the image with no distractions or shadow etc. But it still quite difficult as in reality, images are in quite proximity with so many objects and backgrounds. Thus, making it difficult exactly fine tune the parameters for edges only on the object. As it can be seen in the boat image that there is shadow effect and so many other thing in the boat image.

3.) We can see that we got various outputs by using different thresholds values. The problem with higher threshold value is that, if the higher threshold value is lower, more pixels will be selected as edge point, which leads to inclusion of the more unnecessary edges. When they both are close on the higher range we get better localized edges thus leading to smooth results. Lower the low threshold noisier will be the edge map. More the textures in the images more will be unnecessary edges which can be avoided by higher range thresholding. And in the boat image we see that some portion of the Boat which has shadow is the weak boundary region as it very Rapidly loses the precision with changes in the threshold values.

2.4.b) Structured Edge Detector

- We can see from the above results that structured edge visually provides better results as compared to canny edge detector. Canny heavily relies on the threshold values. There are few parts of the boat image that gets occluded with extra edges but that’s not the case in structured edge.

2.4.c) Performance Evaluation

- F measure intuitively it is easier to higher value for Castle Image, as F measure being image dependent, because it is simpler to segment the Castle Image. Since the Boat image is camouflaged with so many other textures surrounding it. So thus, it becomes difficult to get high F measure for the Boat Image.
- Also if we look at the tabulated results of the Castle and Boat for Structured Edge and Canny, F measure is always on towards the higher range for Structured Edge and very low for Canny.
- The F measure is a measure of accuracy in terms of commenting statistically. Both precision and recall affect the F measure. If any one of them is the F measure will result in a bad or lower value. Also, there exists inverse relation between the P and R. Thus, if we want to maximize the F measure then we should try to make them equal or very close to each other.
- Precision (P) + Recall (R) = Constant (C), then P = C -R
  F = 2. (P.R) / (P + R) à F = 2. (C- R) (R) / C.
  Take derivative, and F’ = 0.
  Then, 2C – 4R = 0. R = C/2.
  So, when R = P, F- Measure is maximum.

Problem 3: Salient Point Descriptors and Image Matching (30 %)

3.1) Motivation

The main motivation here is how can we describe an image. The images can be better described based on the meaningful extraction of the features. In applications, such as object recognition, video tracking, panaroma etc., we need a means to extract features from the images. And as computer vision algorithm requires the computer to understand the world without losing the generality of the algorithm. So, for that we need to extract the features which are indeed general i.e. more robust, in other words rotation and scale invariant. SIFT and SURF are the two algorithms who help us to identify such important local features in an image.

Let's take an example of highway patrolling of car via cameras, in which we need to identify car is of which model and to which another image it matches to etc. So, in such applications it is very important to identify the key features point that can describe the image captured and help us to identify the car. SIFT and SURF come handy in such applications.

SIFT (Scale Invariant Feature Transform) was developed by David Lowe and SURF (Speeded UP Robust Features) was developed by Herbert Bay.

3.2) Approach

SIFT Feature Extraction Algorithm

To implement the above algorithm, I used Open CV inbuilt functions. It is carried out in 4 stages:

- Scale Space Extrema Detection: To achieve the key points to be scale invariant, we consider the input image on different scales. So, the author recommended that this can be done via Gaussian Smoothening Kernel with different values of sigma.

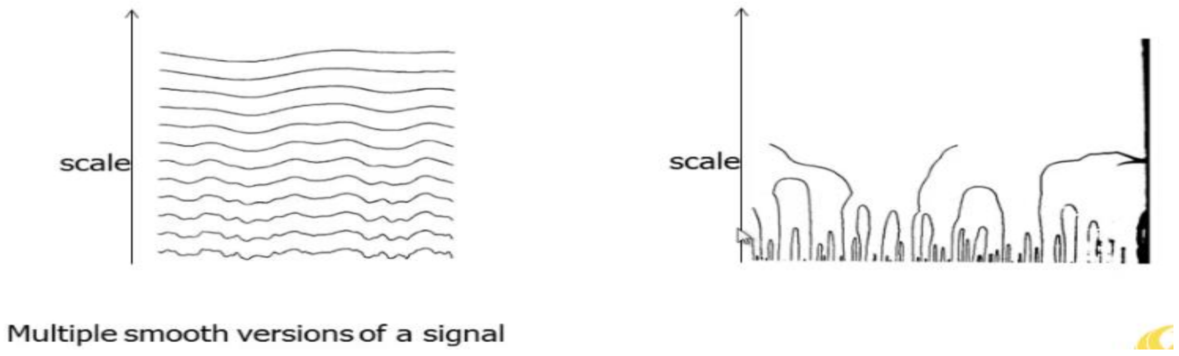


Figure 18: Scale Space

As the scale increases, along with that the variation decreases. Thus, at higher scales we get very less variations. These points generally are said to strong edge/key points because they survived the Gaussian Blurring effect at such higher scale. The scale space consists of n octaves. Each octave is down sampled by a factor of 2 following the first octave. The author empirically found the k value to sqrt (2), where multiples of k will be the sigma for different octaves. Also, the value of the sigma was found to be 1.6. After this step, we compute DoG which helps in approximate LoG (Laplacian of Gaussian). Figure below shows the blurred images which are then used to take differences to get LoG on the right-hand side. DoG gives us an edge map, to detect the point of interest from the edge maps, we iterate through all possible points and select point of interest. Our point of interest lies in the maxima and minima at the given scale.

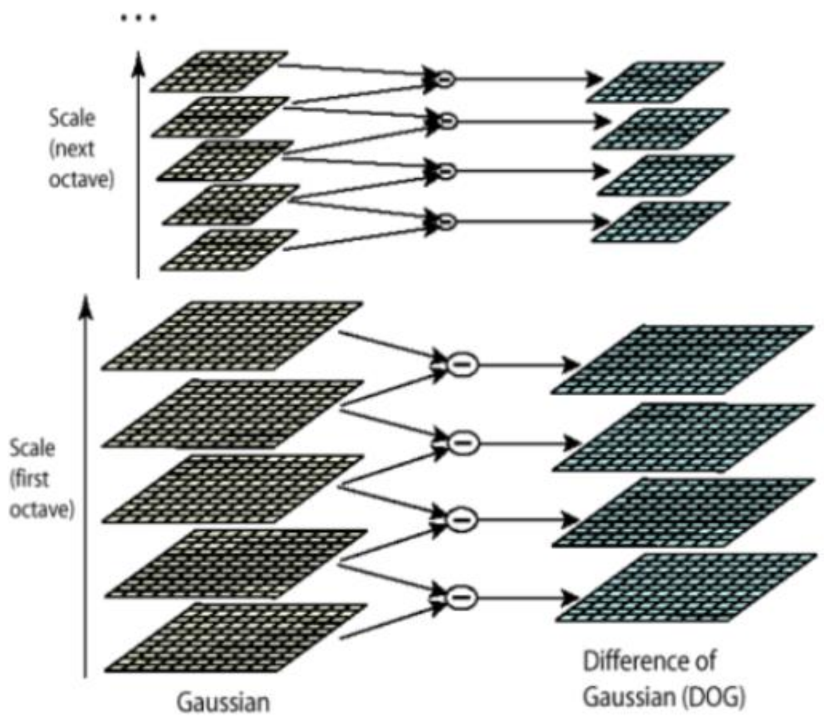


Figure 19: Scale Space and DOG (for approx. LOG)

- **Key point Localization**: Now to remove outlier's we perform Taylor series expansion of DoG and remove the points that lie below a threshold of 0.3. Our main aim is not to detect the edge points instead detect the corner points which are nothing but our points of interest. Corner points do not suffer from the Aperture problem. We also compute the hessian matrix and compare its Eigen values, if the ratio of the first and second Eigen values are greater than 10 that means that the point is edge point, so we discard it.
- **Key Point Descriptors:** For features vectors of the point of interest the author suggested to take the histogram of the neighborhood orientation as features as they provide variations against various issues. So, for this we will consider a patch of 16*16 around the key point this can be SIFT Descriptor which will helps us in developing the SIFT feature vector. Now divide the 16*16 patch in to sixteen 4*4 block and find the histogram of the orientation of the pixels. The resolution of the histogram was taken to 8 bins per 4*4 block. Now the 8 bins per block can be concatenated to produce a sift feature vector of 1*128 dimension. We also need to normalize the vector.

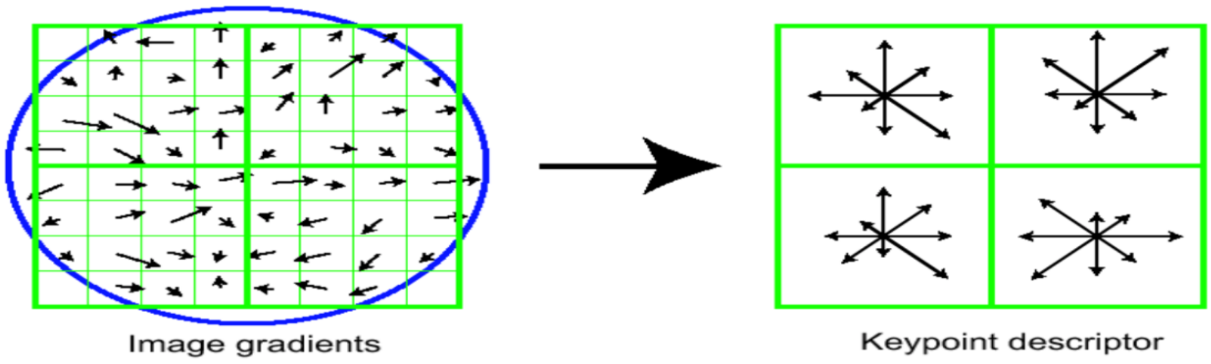
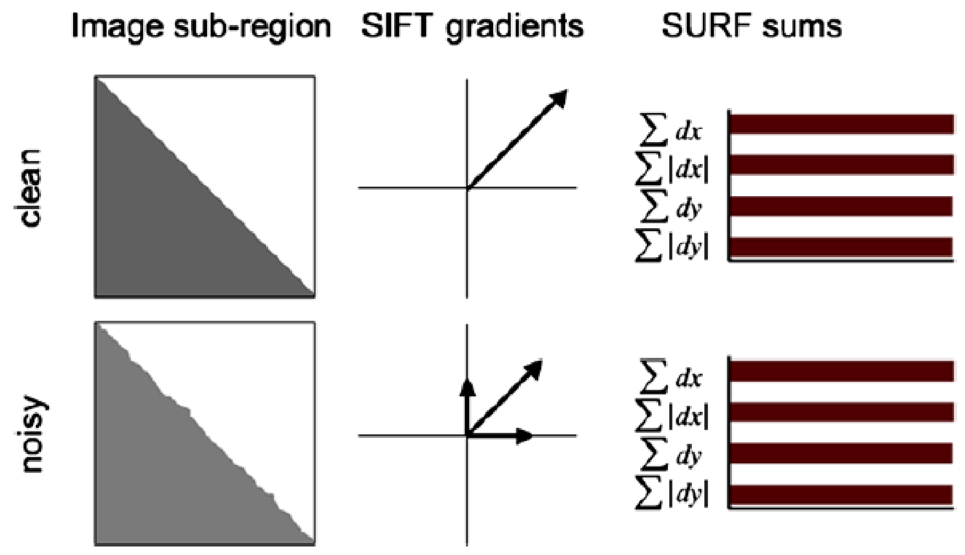


Figure 20:Histogram of Orientation

SURF Feature Extraction Algorithm:

- SURF came in to picture because the SIFT algorithm was computationally costly. SURF is fast when it comes to the computational time. It works and produces pretty much the same output as that by the SIFT algorithm and produces lots of features even for a small object in the image. It is robust against transformations.
- The main step which is involved in this algorithm is that it takes the integral of the image for speeding up the filtering. The integral filter is faster irrespective of the size of the filter matrix thus we can achieve better results in lesser time.
- Once the integral image is created we need to find the interest points and for doing that we need to determinant of the Hessian Matrix.
- LoG filter is used in Hessian Matrix. Using the integral image, we obtain the sum of intensities or square present in Hessian box filters, which is multiplied by the weight factor and then the addition of the net resultant sum.
- Next step is to find the major interest points. This can be obtained by using 3*3 Non- Maximal Suppression below and above the scale space of each octave.
- Now is the time for the feature direction detection and it is advised to use Haar transform for that purpose.
- Final step is generating the 64-dimensional feature vector. For 16 sub regions, which are created from the square description window dx and dy are Haar wavelet responses in the Horizontal & Vertical Directions.



Bag of Words Algorithm:

Bag of words algorithm was firstly used for documents classification. Then it was suggested to use in the image processing. The final motivation behind doing the bag words is that we can represent any similar image used for training in terms of histogram of the best possible sift descriptors.

- So, it is kind of extension of the SIFT algorithm, here we suppose multiple images from different class. Say a bike image, face image, plane image etc and then we find the key points in conjunction to it we will find the corresponding sift descriptors.
- After that we will perform the K-means Clustering on the given K bins, in our case it was 8 bins. After K-means clustering we will get best sift feature vectors which are the centroids as well, representing K clusters.
- Now the training part is completed, now we can use similar images and find Euclidean distance between its descriptors from the generated best centroids and then map them to that centroid. In other increase the count

of the bin of that centroid, eventually forming the histogram representation of that image in terms of the best centroids.

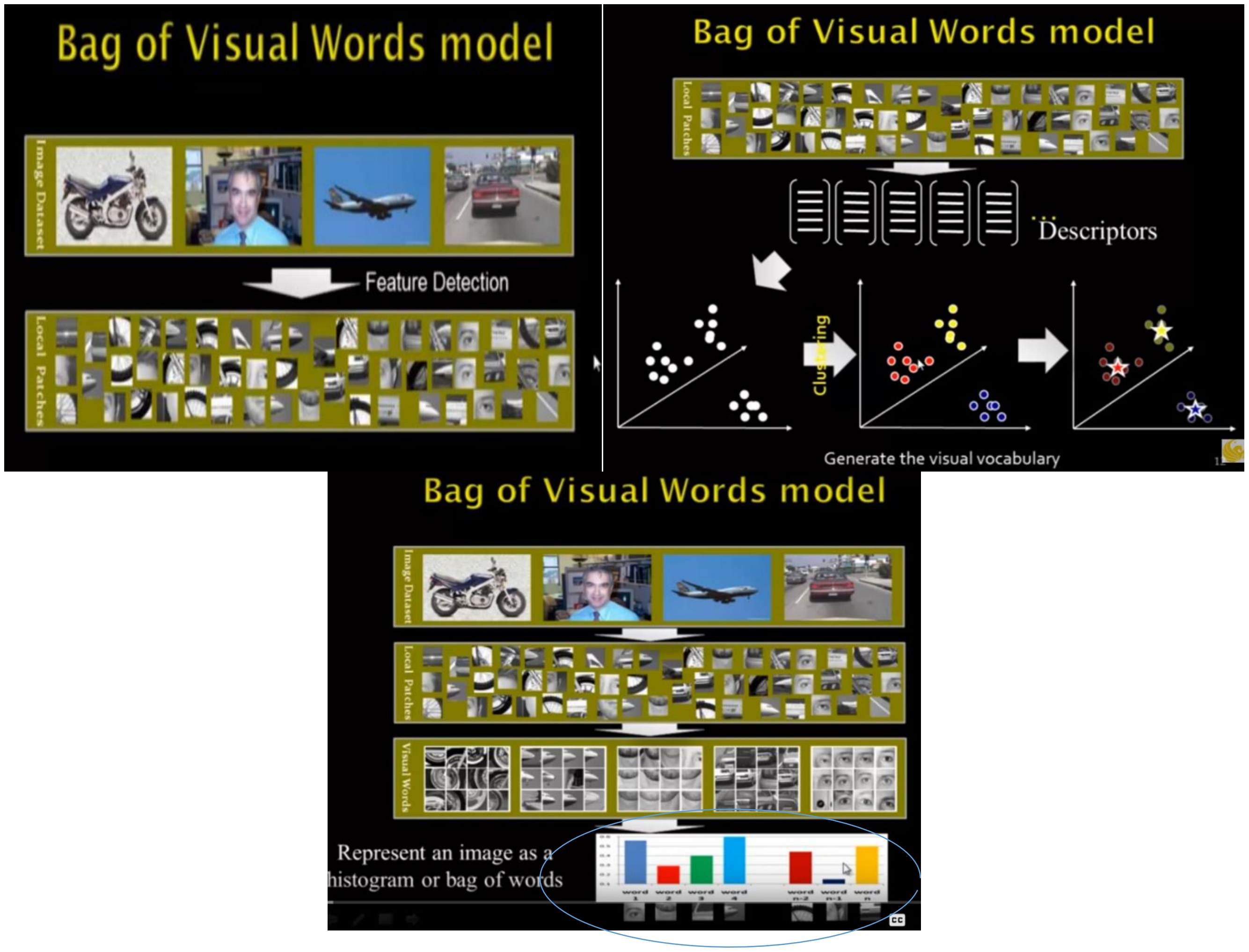

Bag of Visual Words model
Image Dataset
Feature Detection
Local Patches
Bag of Visual Words model
Local Patches
Descriptors
Clustering
Generate the visual vocabulary
Bag of Visual Words model
Image Dataset
Local Patches
Visual Words
Represent an image as a histogram or bag of words

Figure 21: Pictorial Representation of the BAG of Words Procedure.

Now the generated histogram can be further send for classification purpose to the classifier.

3.3) Results

3.3.a) Extraction and Description of Salient Points (Basic: 10%)

So here we were given following two images of SUV and Truck and we were asked to extract salient key points via SURF as well as SIFT. Following are the results:

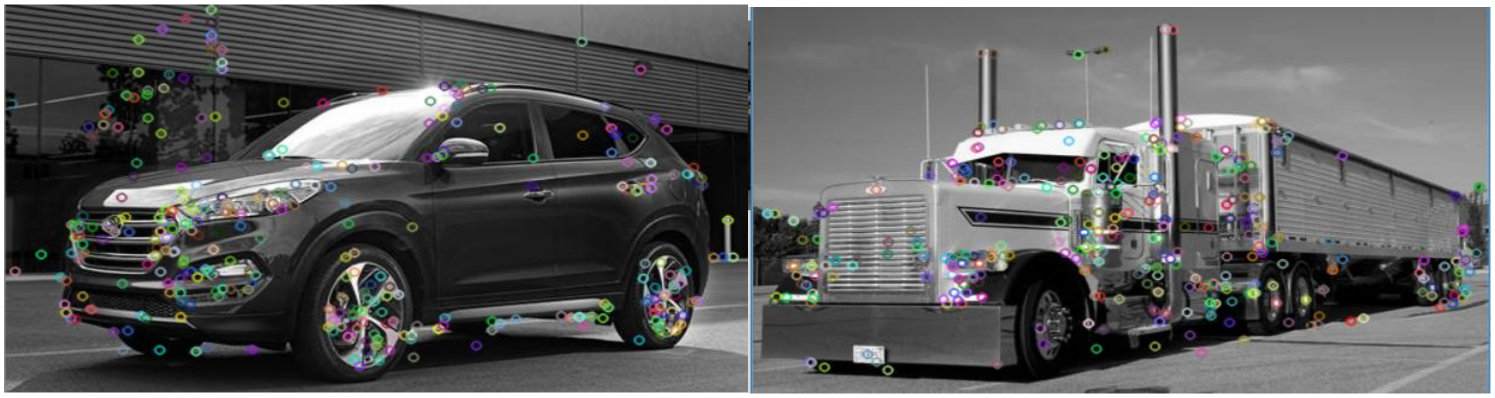


Figure 22: Key Points (SIFT)

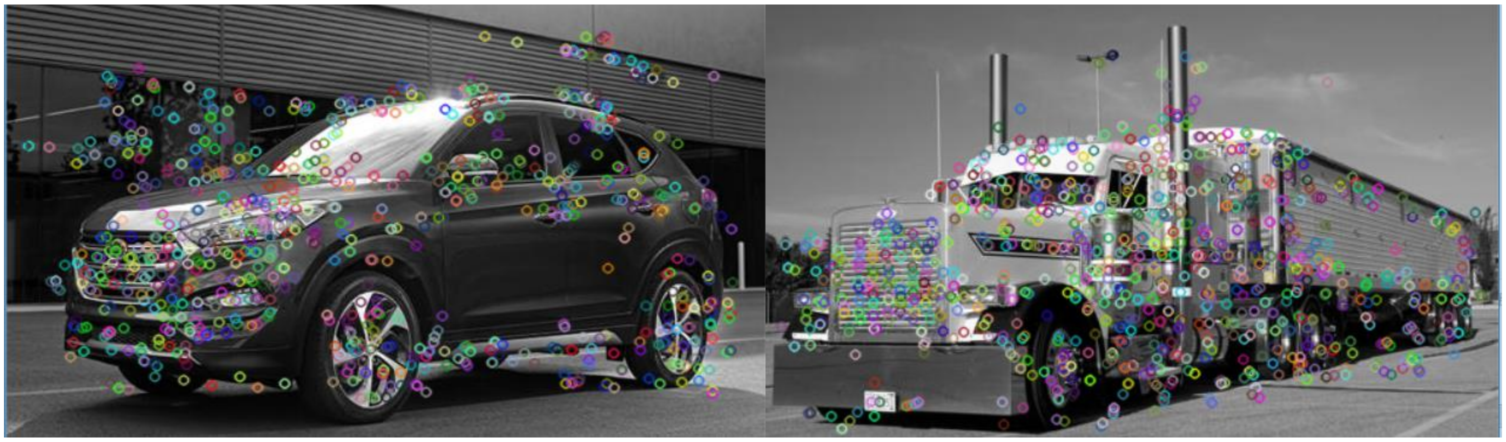


Figure 23: Key Points (SURF)

3.3.b) Image Matching (Basic: 10%)

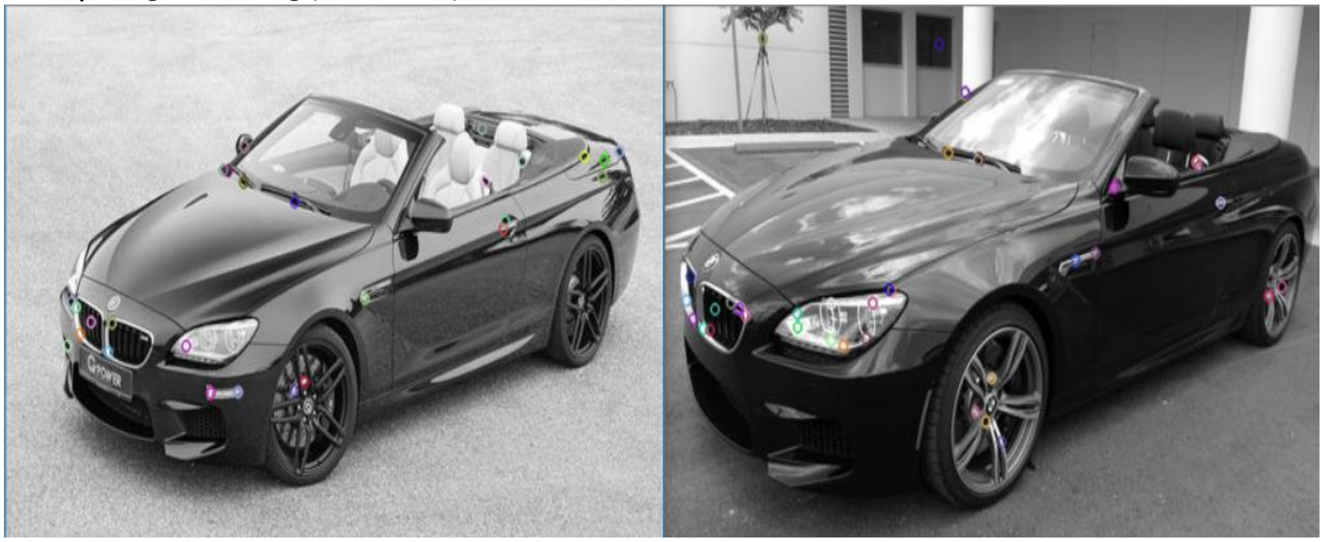


Figure 24: Key Points for (SIFT minHessian =40)

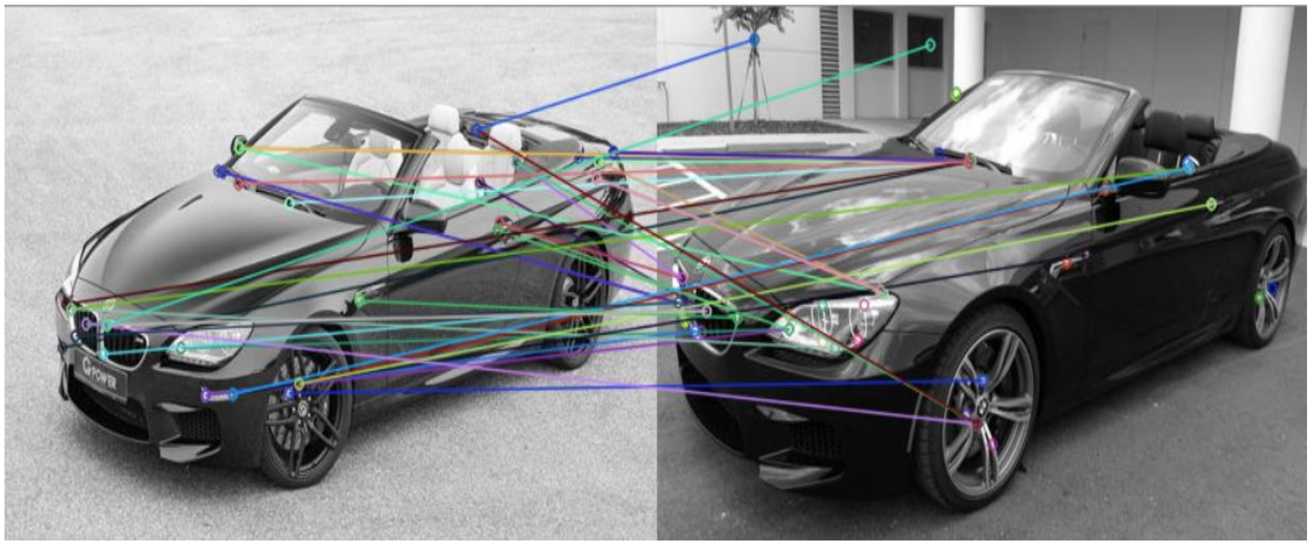


Figure 25: SIFT Key Point Matching

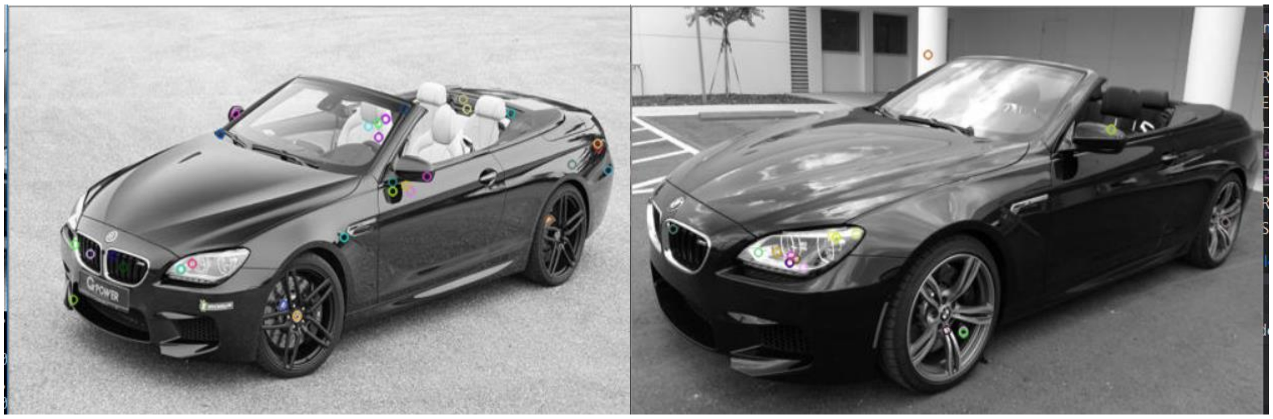


Figure 26: Key Points for (SURF minHessian =9000)

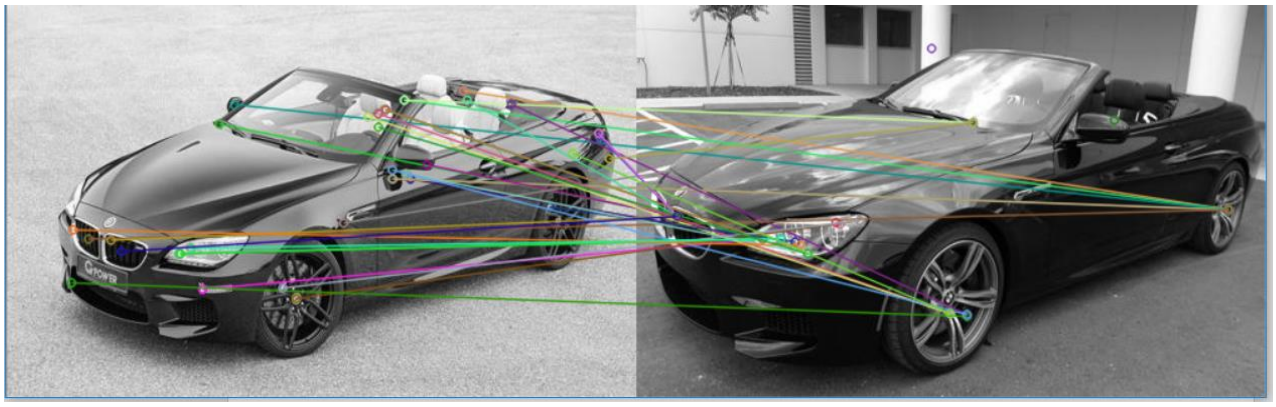


Figure 27: SURF Key Point Matching

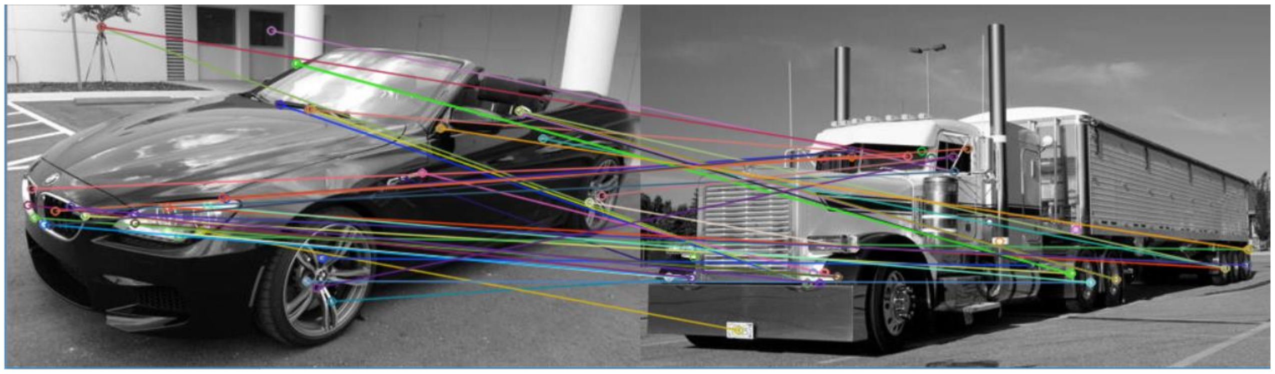


Figure 28: SIFT Key Point Matching (CONV_2 and Truck : minHessian =40)

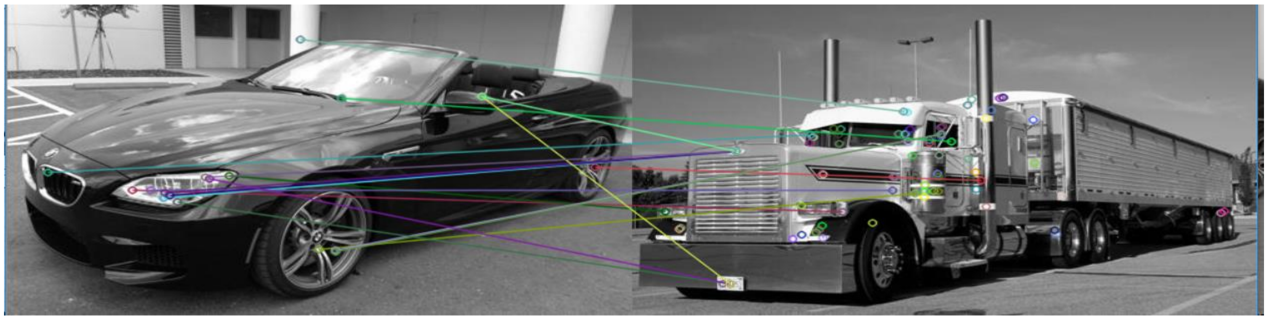


Figure 29: SURF Key Point Matching (CONV_2 and Truck; minHessian =9000)

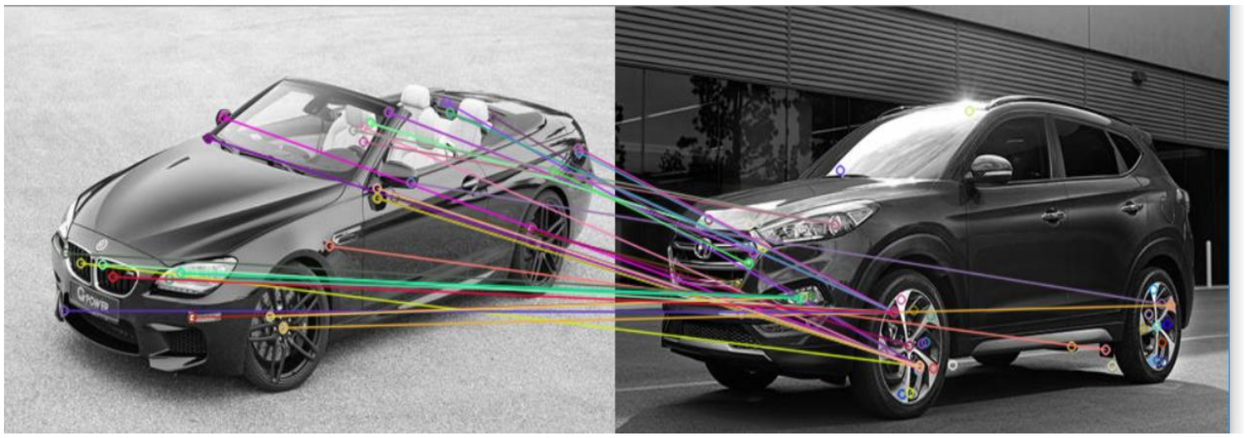


Figure 30: SIFT Key Point Matching (CONV_1 and SUV: minHessian =40)

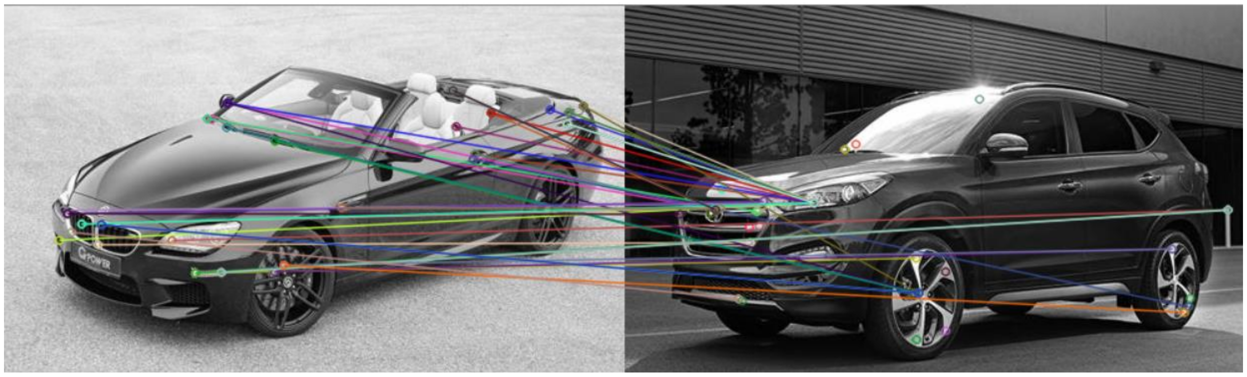


Figure 31: SURF Key Point Matching (CONV_1 and SUV; minHessian =9000)

3.3.c) Bag of Words

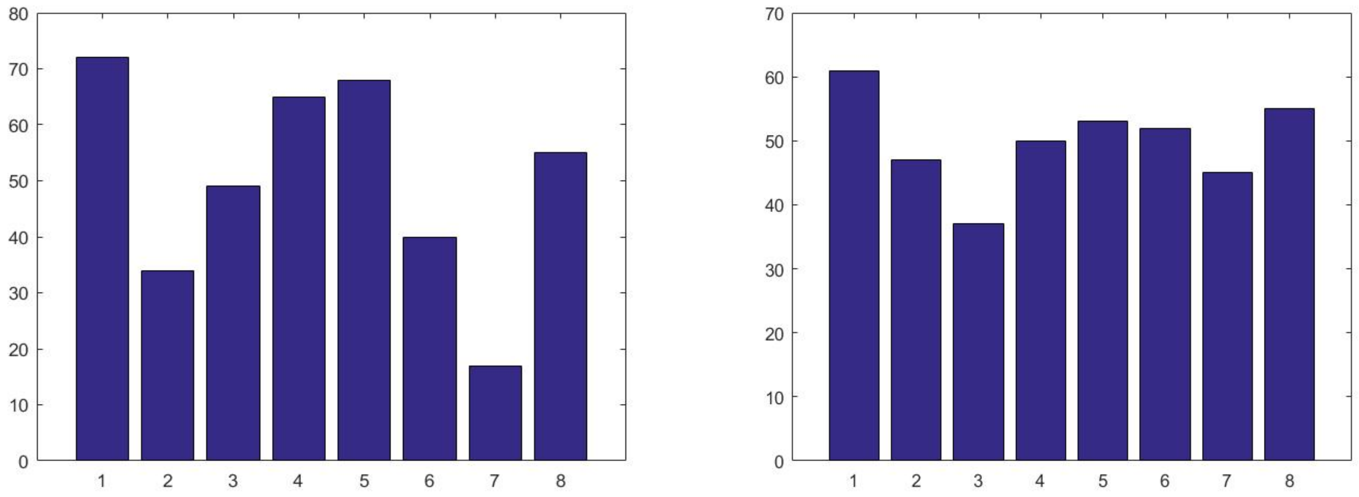
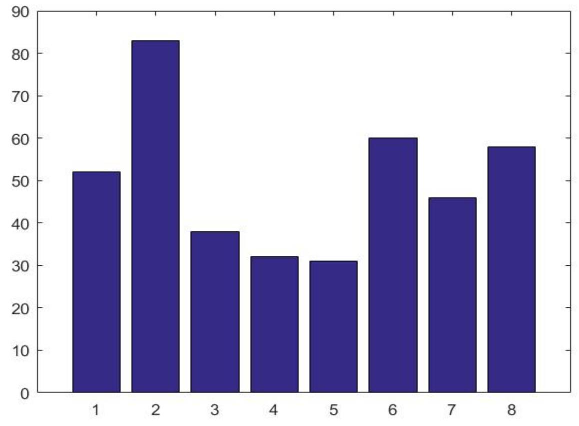


Figure 32: Histogram of the Conv_1 and Conv_2 respectively



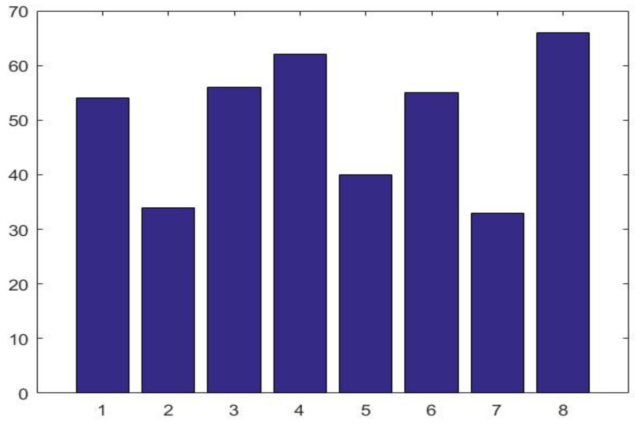


Figure 33: Histogram of the SUV and Truck respectively

3.4) Discussion

- SURF algorithm gives more key points as compared to the SIFT, and there are many reasons to support that. SURF method computes meaningful features in one third of the time taken by the SIFT. This happens because of the optimization setting in SURF. SURF uses integration with images, which takes very less time for convolution irrespective of the size of the filter.
- So the places it won't work is when the detected key point is actually not a corner and due to which the it can get matched to any similar point in the other image.

For the Matching Part of Conv_2 with other images via Histogram Matching:

So, for this part of the question the Conv_2 was said to be matched with the image with whom it has less error factor:

$$error\ factor = \frac{1}{8} * \sum abs(x_i - y_i)$$

So, for different values of minHessian I got different matching:

For minHessian value: 550 Conv_2 matches with Conv_1;

For minHessian value:300 Conv_2 matches with Truck;

For minHessian value:150 Conv_2 matches with SUV;

Min Hessian	Conv_1	SUV	Truck
550	13.25	16.5	17.625
300	8.5	9.75	7
150	7.625	3.75	6

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]www.wikipedia.com

[5] http://electronicimaging.spiedigitallibrary.org/data/journals/electim/23507/013018_1_1.png

[6] www.youtube.com

[7]www.google.com