EE569 Digital Image Processing

Spring 2018

Assignment 3

Name: Thiyagarajan Ramanathan

USC ID: 4973341255

Issue Date: 2/23/2018 Due Date: 3/28/2018

Problem 1: Texture Analysis and Segmentation (30 % + 10%)

In this problem, you will implement texture analysis and segmentation algorithms based on the 3x3 Laws filters discussed in the lecture or the 5x5 Laws filters constructed by the tensor product of the five 1D kernels in Table 1:

Table 1: 1D Kernel for 5x5 Laws 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]

1(a) Texture Classification (Basic: 10%)

In this part, please construct nine 5x5 Laws Filter using the following three filters:

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

Twelve texture images, from *texture1* to *texture12* (size 128x128) are provided for the texture classification task. Samples of these images are shown in Figure 1.

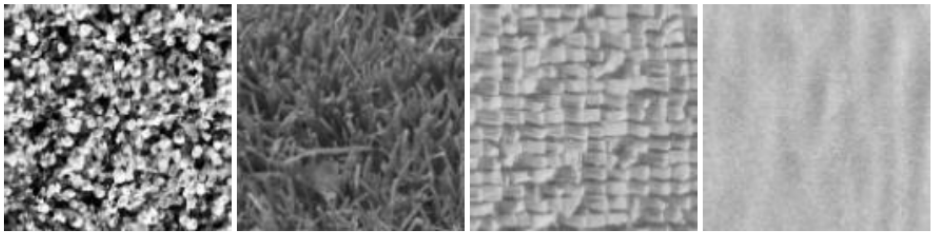


Figure 1: Four types of textures: rock, grass, weave, and sand

Please cluster them into four texture types with the following steps below to complete this problem.

1. Feature Extraction: Use the nine 5x5 Laws Filters to extract feature vectors from each pixel in the image (use appropriate boundary extensions).

2. Feature Averaging: Average the feature vectors of all image pixels, leading to a 9-D feature vector for each image. Which feature dimension has the strongest discriminant power? Which has the weakest? Please justify your answer.
3. Clustering: Use the K-means algorithm for image clustering based on the 9-D feature obtained in step 2.

Report your results and compare them with the reality (by checking the texture images by eyes). Discuss any discrepancy.

(b) Texture Segmentation (Basic: 20%)

In this part, apply the 3x3 Laws Filters to texture segmentation for the image shown in Figure 2 with the following steps.

1. **Laws feature extraction:** Apply all 3x3=9 Laws filters to the input image and get 9 gray–scale images.
2. **Energy computation:** Use a windowed approach to computer the energy measure for each pixel based on the results from Step 1. You may try a couple of different window sizes. After this step, you will obtain 9-D energy feature vector for each pixel.
3. **Normalization:** All kernels have a zero-mean except for $L3^T L3$. Note that $L3^T L3$ is typically not a useful feature, and can be used for the normalization of all other features.
4. **Segmentation:** Use the k–means algorithm to perform segmentation on the composite texture images given in Figure 2. If there are K textures in the image, your output image will be of K gray levels, with each level represents one type of texture. For example, there are six textures in Figure 2, you can use six gray levels (0, 51, 102, 153, 204, 255) to denote six segmented regions in the output image.

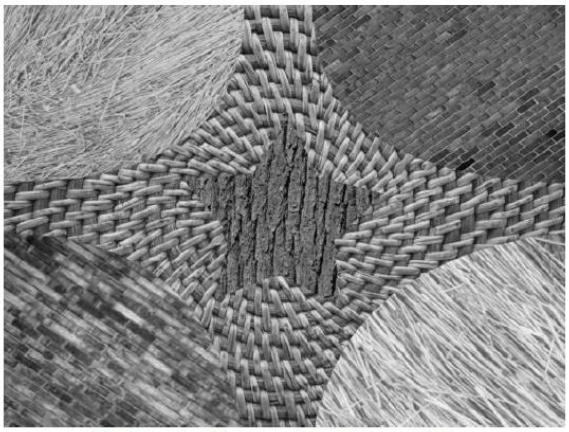


Figure 2: Composite texture image

(c) Advanced (Bonus: 10%)

You may not get good segmentation results for the complicated texture mosaic image in Figure 2. Please develop some techniques to improve your segmentation result. Several ideas are sketched below.

1. Post-processing technique to improve your segmentation results based on your observation.
2. Adopt the PCA for feature reduction. You may adopt all twenty-five 5x5 Laws Filters and use the Principal Component Analysis (PCA) method to reduce the dimensions of the feature vectors. Use dimension reduced features to do texture segmentation of Figure 2. You may use built-in C/Matlab functions of PCA.

(1.a) **Texture Classification**

Abstract and Motivation:

Image classification and object detection have become a very important tool in computer vision and machine learning. One of the major needs to identify the texture is to segment an image into different regions. This is called as Image Segmentation. Image Segmentation is one of the main applications of image processing and it is very much useful in the biomedical applications. So, the very first step is to identify the textures and their boundaries. Once this is done, the image can then be easily segmented. The objective of this question is to cluster the given 12 texture images with the use of law's filters and applying k-means

algorithm to feature vectors obtained by the application of the law's filters. This is one of the statistical methods for identifying a texture.

Approach and Procedure:

An image has an intensity value in each of its pixels. Different textures have different patterns in distribution of the intensity values among them. The law's filters help us extract the details about the spread of these textures in an image. There are 5 different 1D kernels for law's filters. Filter masks can be obtained by the tensor products of pairs of these five 1D kernels. The five 1D kernels are given below:

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]

As the name suggests, these kernels are used to maximize the effects of ripples, waves, spots, edges and levels in an image. These filters are very similar to the other low pass or high pass filters, except for one thing. Once these filters are applied to the images, the feature vector obtained for the image represent the influence of each pairs of the 1D kernels. If the image has a lot of variations in edges, then the level-edge mask will give a high response. For this problem, we are using the following three 1D kernels:

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

Since the given texture images completely consist of one texture, each image itself can be considered as a single value for each pair of filters.

So, the first step to cluster the texture images is to apply the all the filter masks to all the images and then generate one single value for an image per mask. This can be done by applying each of the mask to the texture and then calculating the average energy of the image by summing the squares of each of the pixel intensities and then dividing it by the number of pixels in the image. Hence, each image will have 9 values after applying the 9 filter masks. This is repeated for all the other texture images as well. In the end, we will have 12 data points with 9 dimensions each.

After obtaining this vector, the next step is to apply the k-means filter to cluster the data points. K-means clusters the data points to the clusters based on the distance between the data point and the cluster centroid. The data-point is allotted to the cluster for which the distance measure is minimum. This is an iterative process which happens with the updating of the mean, calculating the distance and allocating the data point to the cluster with the shortest distance. Once the data points are clustered, the texture images can be referenced from the data points. The texture images are now clustered. Various attributes of the image such as noise very much affect the clustering process.

The algorithm for k-means clustering is given below:

Step 1: Specify the number of clusters and allot random data points as cluster centroids

Step 2: Start the iteration process

Step 3: Calculate the distance between each data point and all the cluster centroids

Step 4: Allot the data point to the cluster with which the distance is less when compared to other centroids.

Step 5: Update the centroids for each cluster by calculating the mean of the data points that are present in that particular cluster

Step 6: Continue the above process until all the cluster centroids remain the same after one iteration or if the change is very small, under the tolerance level.

Step 7: Stop the iterative process

Experimental Results:

```
Texture 1 belongs to cluster 1
Texture 4 belongs to cluster 1
Texture 6 belongs to cluster 1
Texture 3 belongs to cluster 2
Texture 5 belongs to cluster 2
Texture 9 belongs to cluster 2
Texture 11 belongs to cluster 2
Texture 2 belongs to cluster 3
Texture 12 belongs to cluster 3
Texture 7 belongs to cluster 4
Texture 8 belongs to cluster 4
Texture 10 belongs to cluster 4
```

Figure 1

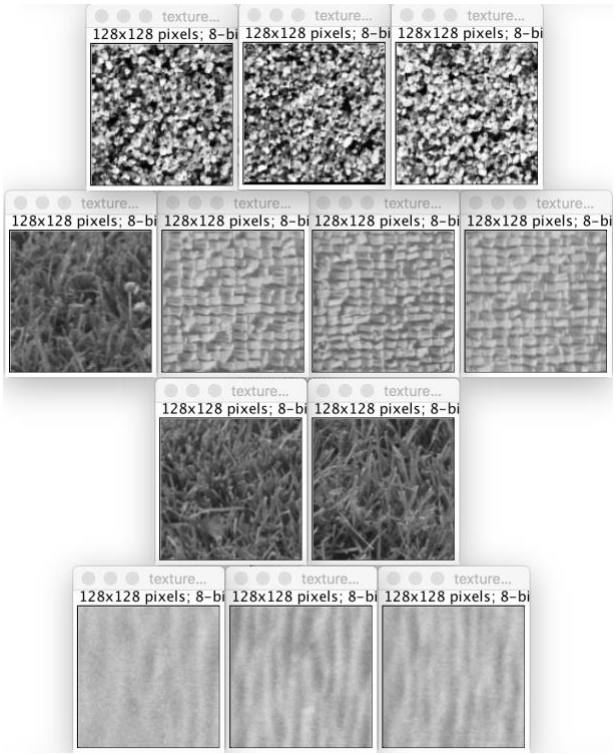


Figure 2

```
Texture 1 belongs to cluster 1
Texture 4 belongs to cluster 1
Texture 6 belongs to cluster 1
Texture 3 belongs to cluster 2
Texture 5 belongs to cluster 2
Texture 11 belongs to cluster 2
Texture 2 belongs to cluster 3
Texture 14 belongs to cluster 3
Texture 12 belongs to cluster 3
Texture 7 belongs to cluster 4
Texture 8 belongs to cluster 4
Texture 10 belongs to cluster 4
```

Figure 3

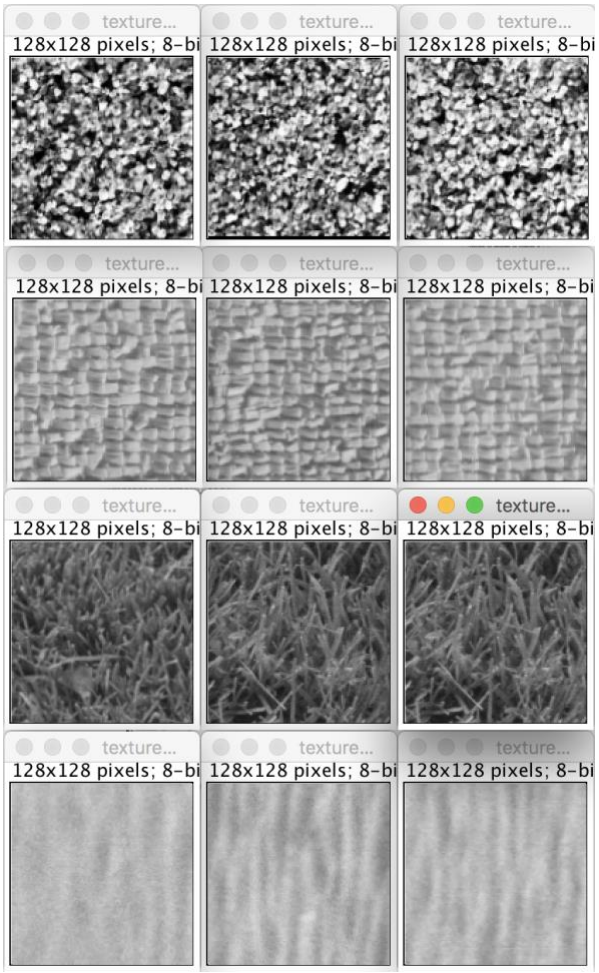


Figure 4

```
The variance of feature 1 is 7531.914440383417
The variance of feature 2 is 7216.805475183083
The variance of feature 3 is 627.7482510590667
The variance of feature 4 is 734.9720733955246
The variance of feature 5 is 335.69979801516666
The variance of feature 6 is 395.41215850270834
The variance of feature 7 is 22446.783109986438
The variance of feature 8 is 3193.624351351458
The variance of feature 9 is 48.174442272002075
```

Table: Variance values of different features

Discussion:

There are 12 textures in total. Each mask was applied over the 12 textures. And this gives us 9 images for one single image. The 9 images are reduced to a single numerical value as they contain the same texture throughout the entire image. So, we will have one value for each feature of the image. So, the number of data points will be 12 each having a dimension of 9. K-means algorithm was used to cluster these data points and once they were clustered, the images were classified. The classification results for considering the texture images from 1-12 are given in figure 1. If we see, there is one image that is being misclassified. The texture 9 belongs to cluster 3 but it is being classified as belonging to cluster 2. This is mainly due to the reason of noise. The textures according to their clusters are given in figure 2. If we see, the texture 9 (first element of the second column) is very noisy when compared to the other images. Due to this reason, the image is misclassified. This can be improved by denoising the image before it is being passed to extract the features.

Results obtained using texture14.raw which is the de-noised version of texture9.raw, are given in figure 3 and the images in accordance to their clusters are given in figure 4. As we can see, the results obtained are different from that of the previous versions. This is due to the use of de-noised version of texture9. All the textures are now correctly classified. Hence it is always a good method to de-noise the images before we use it for any image processing application. Since this is unsupervised clustering (we do not know the labels of each image), we have to specify the number of clusters initially. The accuracy of the k-means algorithm depends on the values of the starting values of the mean for each cluster. They have to be initialized with one of the data points. Only then the algorithm will give us good results.

The same algorithm can be used to identify textures present in a single image. In this case, we have images which contain the same texture throughout the image and this is the only reason why we can convert an entire image to a single value. In cases where the image contains different textures, we just need to specify the number of different textures present in the image and then calculate the energy of that pixel by calculating the average energy values in a certain window. Instead of having one value for an entire image, we have 9 features per pixel in the image. And hence we will have data points equal to the area of the image, for which we have to apply the k-means clustering algorithm to cluster different regions of the image. This is the process of clustering the textures with the help of law's filters and the k-means algorithm.

To identify the feature with the best discriminant power, the variance of each of the features must be measured with the help of the 12 data points. In Machine learning, the feature with low variance is considered noise as it does not help classification in any way. From the variance values given in table above, we see that the largest variance value is 22446 and the lowest variance value is 48.17. The feature having the variance 22446 is $E3E3'$ and the feature having the variance 48.17 is $W3W3'$. Thus, the feature with the strongest discriminant power is $E3E3'$ and the feature with the lowest discriminant power is $W3W3'$.

(1.b) **Texture Segmentation**

Abstract and Motivation:

Texture Segmentation is an important task in image processing. The idea is to segment the image based on the surrounding pixels of one type. Texture is a statistical parameter and hence there is no strict definition for it. There are many methods to perform this task of image segmentation based on the textures. Evolution of Convolutional Neural Networks has made this very much easier. Before the evolution, law's filters gave a good segmentation of the images based on the texture. Image segmentation is a classical problem in biomedical imaging. Image segmentation helps very much in thermal imaging and night vision as well. Texture segmentation is also a major component in computer vision applications. Filter based methods such as Gabor transforms, Fourier transforms, etc., give good results, but they are computationally intensive. Hence, law's filters have been trade-off between trade-off and computational effort.

Approach and Procedure:

The first step in this procedure is to extract the feature vector for each pixel in the image. To obtain the feature vectors, law's filter masks have to be applied to the image. As mentioned in the previous problem, there are five different 1D kernels, to detect edges, spots, levels, ripples and waves. The masks can be obtained by performing the tensor product between two 1D kernels. The three 1D law's 1D kernels are given below:

$$E3 = [-1\ 0\ 1]$$
$$L3 = [1\ 2\ 1]$$
$$S3 = [-1\ 2\ 1]$$

The filter masks are as follows:

$$E3E3^T,\ S3S3^T,\ L3L3^T,\ E3L3^T,\ E3S3^T,\ L3S3^T,\ L3E3^T,\ S3E3^T,\ S3L3^T$$

For this case, we consider the 3 dimensional 1D kernels for edge, level and spot. The 9 masks are created by taking the tensor products of these 1D kernels. Once the kernels are obtained, they are applied to the image that is to be segmented. This is just taking the inner product of each of the mask with one pixel and its surrounding 24 neighbors. This is repeated for all the pixels in the image. And then, the energy is to be obtained. Each pixel is allotted a value equal to the average of the sum of the squares of the energy in its surrounding window which can be of any size. To have a more global approach, different filtering windows can be used to find the final feature value for each pixel. So, each mask gives us one feature point for each of the pixel in the image. This can be repeated with the other pixels in the image. So, when all the 9 masks

are applied, 9 different images are obtained and hence we have 9 features for each data point. The number of data points is equal to the number of pixels in the image and each pixel has a feature dimension of 9.

After obtaining the data points, the next step is to implement the k-means clustering. From the image, we can observe that there are 6 different textures in the image. So, we can initialize the number of clusters to be 6. So, we need 6 initial centroids. These can be chosen randomly from the data points obtained from the images. After the k-means algorithm clusters the data points, they can be differentiated in the image by allocating different color values to the different textures. The summary of the procedure for texture segmentation is given below:

Step 1: Frame the filter masks by performing the tensor product between each pair of 1D kernels.

Step 2: Apply the filter to the image and get 9 different images

Step 3: Find the average energy by considering a window around the current pixel, by finding the average of the sum of squares of the intensities of the pixels in that particular window

Step 4: After extracting the 9 features for all the pixels in the image, apply k-means

Step 5: After the data points are clustered, differentiate them by allotting a specific color value to each of the cluster. For example, colors for the clusters can be: 0, 51, 102, 153, 204, 255

Sometimes, different results can be applied by normalizing the values in the image. The normalizing factor can be the value of $L3L3^T$ as this specific kernel has not got a zero mean unlike all the other masks. Another improvement can be to subtract each pixel with the local mean of the surrounding window to reduce the illumination effects.

Experimental Results:

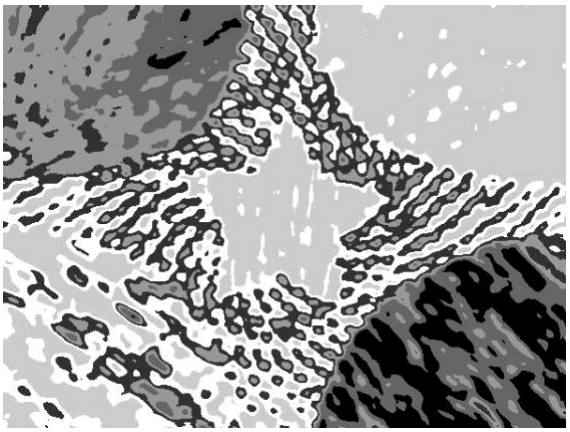


Figure 5: Energy window 15

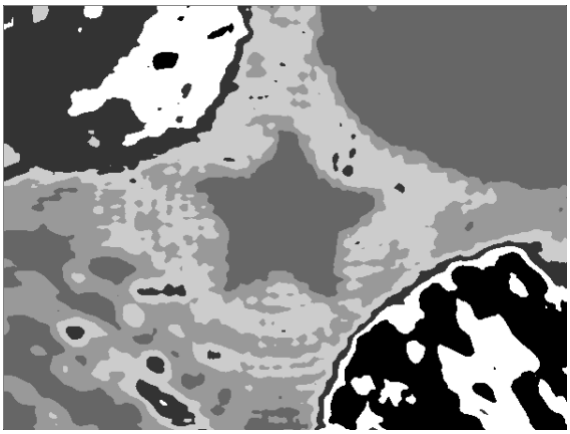


Figure 6: Energy window 19

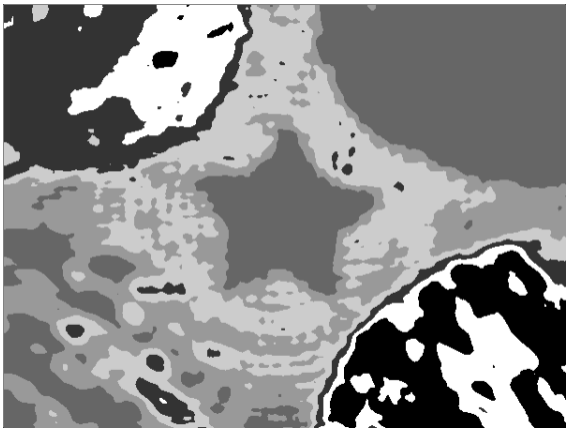


Figure 7: Energy window 23

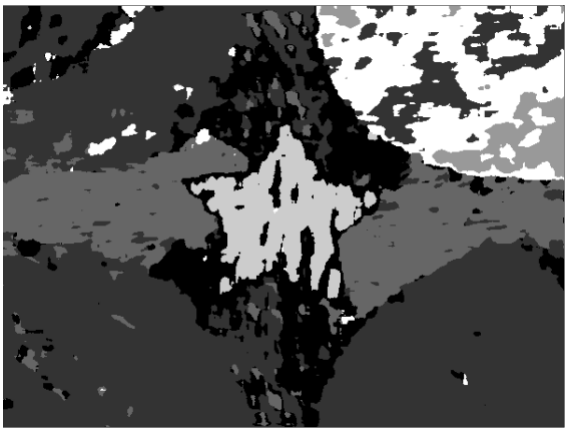


Figure 8: Normalizing energy with L3L3’

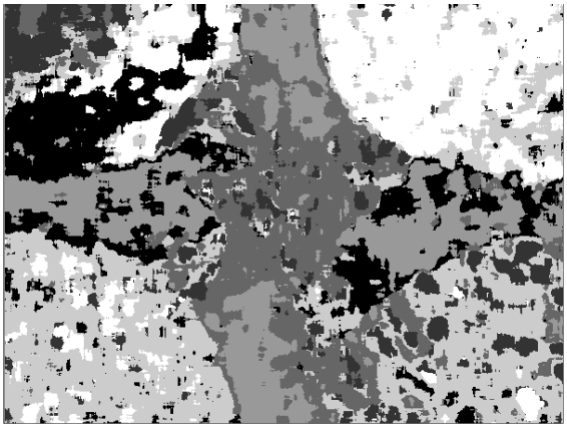


Figure 9: Subtracting mean (W:3x3)

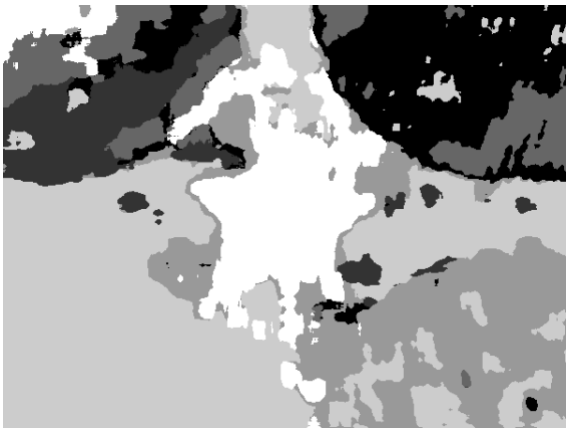


Figure 10: Subtracting mean (W:9x9)

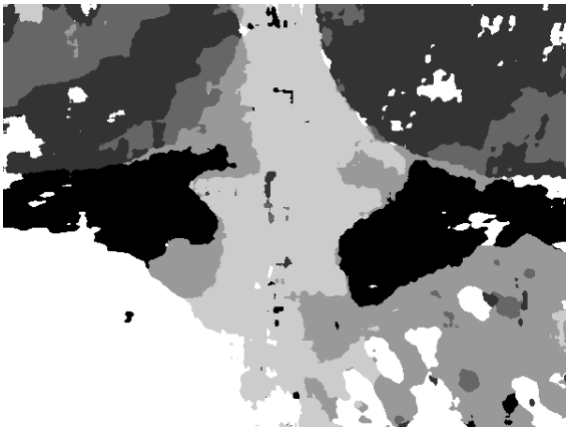


Figure 11: Subtracting Mean (W:15x15)

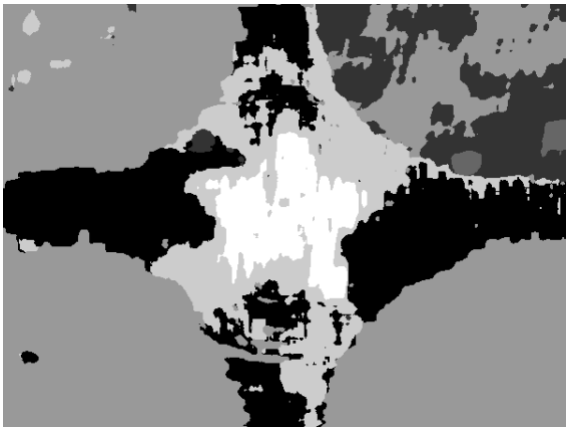


Figure 12: Normalizing Feature Vector using L3L3’

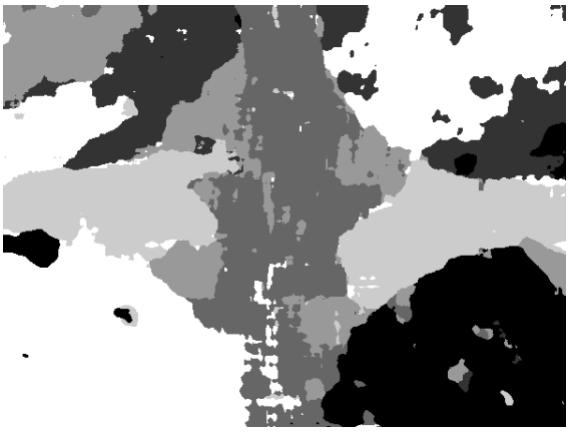


Figure 13: Subtracting Global Mean of the image

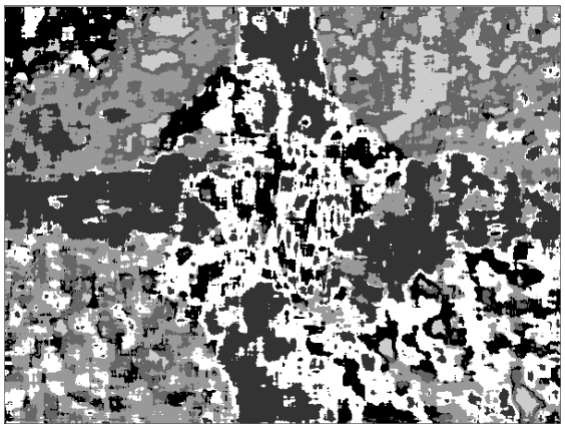


Figure 14: Normalized and subtracted mean (W:3x3)

Discussion:

The segmentation results obtained with different energy windows, by normalizing using the L3L3' feature and by subtracting the local as well as the global means. Figure 6 has got the best segmentation results. This image was obtained with using 9 features for all data points, and the for calculating the energy of each pixel the window used was 19x19. The energy of the window was calculated by averaging the sum of squares of the intensity values in the window of 19x19. This considers the surrounding pixels, depending on the other pixels as well, making textures a statistical aspect of the image making it depend on the neighboring pixels as well. This segmentation result is the best result for this particular texture segmentation task.

In figures 5 and 7, the energy windows were modified. The energy windows used in figure 5 was 15x15 and the energy windows used in figure 7 was 23x23. As we can visually see from the results, the image obtained with the window 23x23 has got a better segmentation than the image obtained with the window 15x15. This can be clearly differentiated in the top right region, top left region, bottom right region and the star in the middle of the image. In figure 5, these regions have presence of other clusters as well. Whereas, in figures 6 and 7 these regions clearly do not have any discrepancies. The only difficult part to segment is the bottom left region which is very badly segmented in all the different energy windows.

In figure 8, the energy window used is 15x15. But before the application of k-means clustering, the energy vectors were normalized by the values obtained with the L3L3' energy vector. This feature was particularly chosen to normalize the other vectors as this feature has not got zero mean unlike the other features. The segmentation results are not very bad as disturbance in each region is very less. Though the left half of the image is not segmented properly, the regions that are properly clustered do not have the influence of the other clusters in them. If this image was subjected to more post processing, the results may improve.

In figures 9, 10 and 11, before the feature vectors were applied to the image, each pixel in the image was subtracted by the local mean of a particular window. In figure 9, the mean window used was 3x3, in figure 10 the mean window used was 9x9 and in figure 11 the mean window used was 15x15. As we can see the results are very bad, especially for window 3x3. In figure 13, each pixel was reduced by the global mean

unlike the mean calculated with a window. This is just used to reduce the grains or other disturbances that may be present. The segmentation results are not that good when the global mean is subtracted as well. The idea of subtracting the local or global mean is suitable for image that contain a lot of noise. When a lot of noise is present, application of this idea can give us better segmentation results. This will also help us to reduce the luminance effect. But in this case, as the image is not noisy the mean filtered image does not give us good segmentation results.

In figure 12, the feature vector was normalized with the L3L3' feature before the calculation of the energy vector. After normalizing the feature, the energy was found and the k-means algorithm was applied. The results are definitely better than the normalization of the energy vector with the same features. All the four corners are properly segmented except for the star. As the star is very important, the clustering results are not that good. This method can also be improved by the use of post processing techniques.

In figure 14, the mean is subtracted and the energy vector is normalized by the L3L3' feature. This result is very bad when compared to the other results. This is due to the reason that, once the mean is subtracted, all the features tend to have a zero mean and hence normalizing using that feature does not give us better results. The best results are obtained with using all the 9 features and using energy windows of a larger size. The segmentation results in figures 5, 6 and 7 can be considered as good segmentation results when compared to other variations in the procedure. Also, better segmentation results can be obtained with the use of different k-means algorithm as well.

(1.c) **Texture Segmentation using PCA**

Abstract and Motivation:

Different segmentation results were discussed in the previous problem and only 9 features were used, with the masks being created with 1D kernels of dimension 3. The results can be improved by the use of 5 dimensional 1D kernels discussed in part a. So, there will be a total of 25 features. By intuition, we can say that the use of 25 features will definitely give us better results when compared to the use of just 9 features. The main problem with this approach is the computational intensity that is needed to perform this task. We do not know what the important features too. All the features may not be required for a proper clustering. Sometimes, just 10 features can give us good results. The main task is to reduce the number of features. This can be done by the use of Principle Component analysis. The algorithm will be explained below, but what this basically does is, it gives use a reduced dimension data set which can then be used for clustering with k-means algorithm. By doing these we will be reducing a 25-dimensional data set to any of the desired dimension by just taking the features that influence the dataset more. So, the objective is to use all the 25 features, use PCA to reduce the dimensions and then use k-means to cluster the textures.

Approach and Procedure:

The algorithm for extracting the features has been explained in the parts a and b. The main difference in this part is that, we have to use all the 5 law's 1D kernels to create the filtering masks. All the five 1D kernels have been given in part a. The filtering mask combinations are given below.

$$L5^T L5,\ L5^T E5,\ L5^T S5,\ L5^T W5,\ L5^T R5$$
$$E5^T L5,\ E5^T E5,\ E5^T S5,\ E5^T W5,\ E5^T R5$$
$$S5^T L5,\ S5^T E5,\ S5^T S5,\ S5^T W5,\ S5^T R5$$
$$W5^T L5,\ W5^T E5,\ W5^T S5,\ W5^T W5,\ W5^T R5$$
$$R5^T L5,\ R5^T E5,\ R5^T S5,\ R5^T W5,\ R5^T R5$$

After these masks are created, they are applied to the image to extract the 25 features of the image. Once this is done, the next step is to apply PCA to reduce the dimension of the data set.

The idea of Principle Component Analysis is based on Eigen values and Singular values. For any dimensional data, all the dimensions will not be needed to perform a classification or a regression algorithm. There will be some important features and some poor features. The idea of PCA is to project the dimensions with low importance onto the orthogonal space consisting of the most important features for the data set. Once this is done, the high dimensional data being projected onto a lower dimensional data, we are now in a lower dimensional space. After this, we can then apply the usual k-means algorithm to perform our classification task. The algorithm for PCA is given below.

Step 1: Calculate the covariance matrix between all the dimensions in the dataset
Step 2: Perform the singular value decomposition for the covariance matrix
Step 3: This gives us 2 matrices. One matrix contains the singular values of each feature in the decreasing order and the second matrix gives the corresponding orthonormal basis for the all the features.
Step 4: Decide the number of lower dimensions you want to project the data onto
Step 5: Project the data points onto the orthonormal basis based on the number of dimensions you want
Step 6: The data is now in a lower dimensional data and this can be used for other algorithms

PCA is a very important tool in Machine Learning. It is used to reduce the computational intensity of the entire machine learning application. For example, in most of the applications the number of data points are very less when compared to the dimensions. In text classifications, the dimensions are very high and the data is very sparse. In these cases, only after applying PCA can we get good results. PCA will also give us the most influential feature. For example, if the data has 100 dimensions, but most of the features are not

important then there is no purpose to include all the dimensions to get a higher accuracy. In these cases, PCA can give us good reduction of the dimensions at the same time, having the same accuracy as the original dimension data.

For this texture segmentation now each data point has 25 dimensions. After the application of PCA, we can reduce it to the desired number of dimensions. Once that is done, we need to implement k-means clustering algorithm to cluster the data points in the image. This is the procedure of using PCA to reduce the number of features and then using the reduced dimensions to segment the image based on the textures.

Experimental Results:

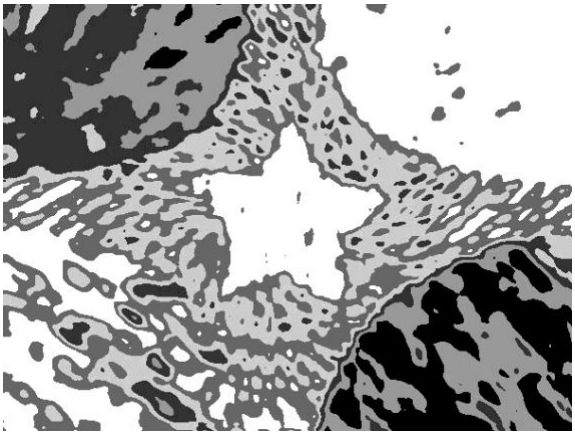


Figure 15: Using all features

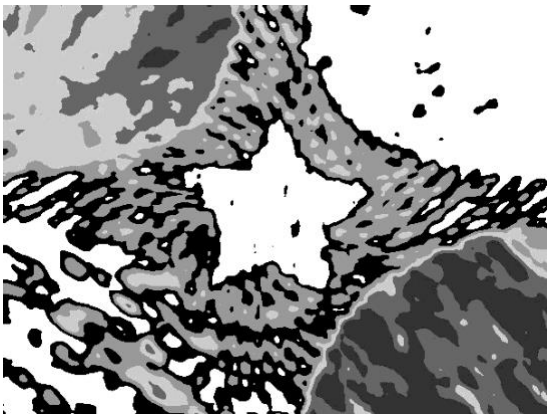


Figure 16: Using 20 best features

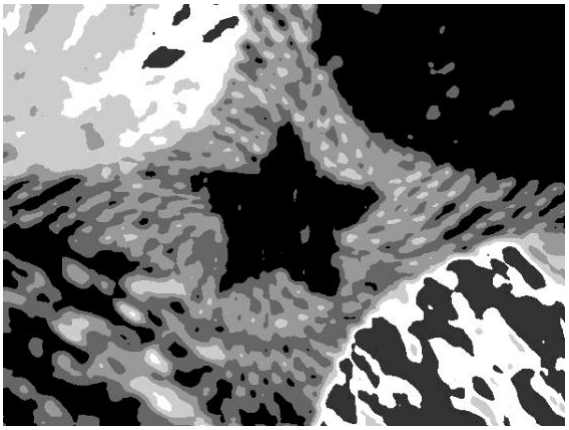


Figure 16: Using 10 best features

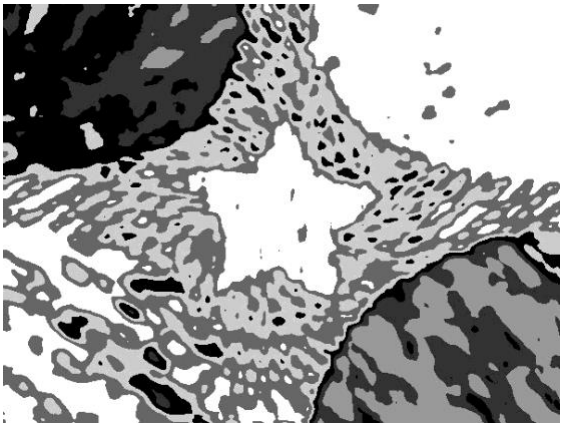


Figure 17: Using the best feature

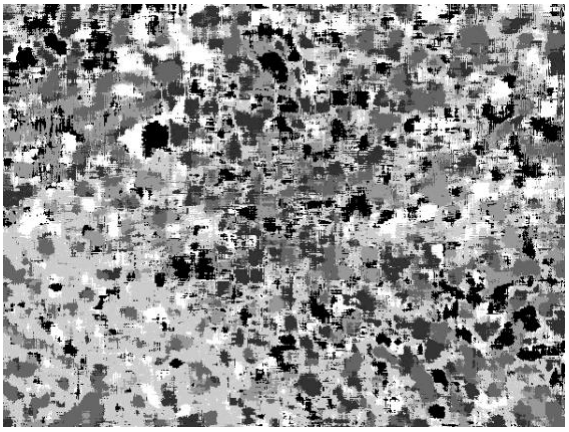


Figure 18: Worst 4 features

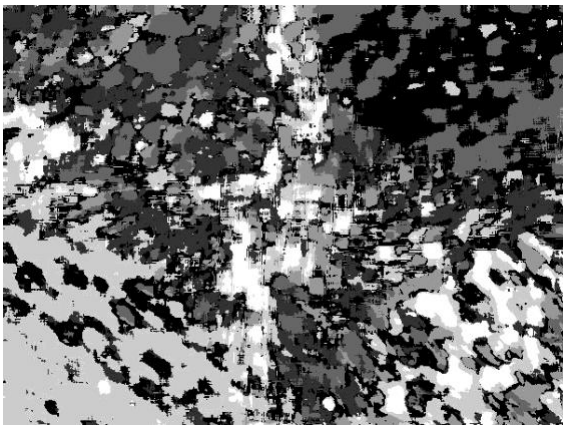


Figure 19: Worst 14 features

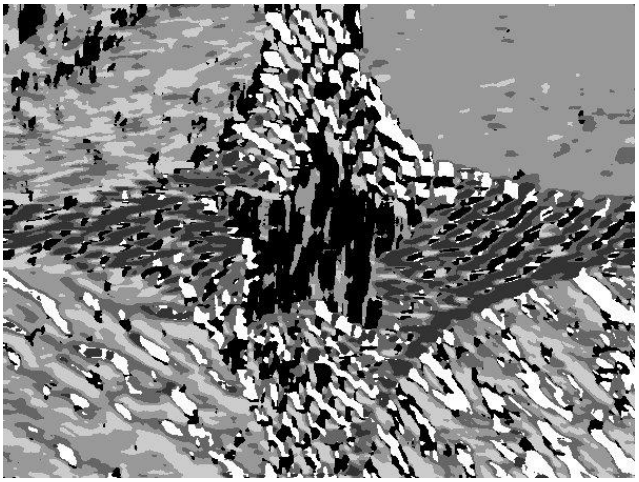


Figure 20: Using all features except the best

```
Variance ratio of feature 1 is : 0.9991413592972467
Variance ratio of feature 2 is : 0.000438058265626324
Variance ratio of feature 3 is : 0.00025758184145370245
Variance ratio of feature 4 is : 7.367373443366654e-05
Variance ratio of feature 5 is : 5.948683303220245e-05
Variance ratio of feature 6 is : 2.0994221796827262e-05
Variance ratio of feature 7 is : 3.3310330715682424e-06
Variance ratio of feature 8 is : 1.772105685277521e-06
Variance ratio of feature 9 is : 1.2848276488107017e-06
Variance ratio of feature 10 is : 7.576472927487907e-07
Variance ratio of feature 11 is : 6.324193649651 33e-07
Variance ratio of feature 12 is : 3.938182559130226e-07
Variance ratio of feature 13 is : 2.3224397569600558e-07
Variance ratio of feature 14 is : 1.3654563190572748e-07
Variance ratio of feature 15 is : 9.838383302169182e-08
Variance ratio of feature 16 is : 6.356025765214706e-08
Variance ratio of feature 17 is : 4.5504986818253856e-08
Variance ratio of feature 18 is : 2.611572162136793e-08
Variance ratio of feature 19 is : 2.2335112002234612e-08
Variance ratio of feature 20 is : 1.9617505032598346e-08
Variance ratio of feature 21 is : 1.005997289631374e-08
Variance ratio of feature 22 is : 8.24937382885098e-09
Variance ratio of feature 23 is : 4.505649340637327e-09
Variance ratio of feature 24 is : 4.245609304796347e-09
Variance ratio of feature 25 is : 2.587462121728916e-09
```

Figure 21: Importance values based on the variances

Discussion:

The segmentation results given in figures 15, 16, 17 and 18 have almost the same results. But the difference is that, the number of features used for segmentation in each of these images is different. The image in figure 15 has 25 features, image in figure 16 has the best 20 features, image in figure 17 has the best 10 features and the image in figure 18 has only the best feature. What this tells us is that, by projecting all the dimensions into the best one dimension, we can have the same clustering accuracy as the one similar to if we have all the dimensions. PCA will give us the basis of that one dimension onto which the all the other dimensions can be projected.

Once the data is projected onto the best dimension, the k-means can be performed. The only difference is that, instead of using all the 25 features to perform the clustering task which will increase the computational intensity very much, we can just use the best feature and still get the best result and this is possible only with the help of the Principle Component Analysis.

The dimension onto which we are projecting the data decides the accuracy of the classification greatly. If we check the images 18, 19 and 20 we can see that choosing the least important dimensions can give us very bad results. Even if take all the lower dimensions except for the best dimension we do not have the results similar to that of the image segmentation considering only the best dimension.

This can be explained with the help of the variance ratios given in figure 21. If we see, the influence of choosing the most important feature is 0.999 whereas all the other features sum up to the remaining 0.0001. This is exactly the reason why even if we use all the lower dimensions except for the best one, we get very low accuracy of texture segmentation. When we use the last 24 dimensions, we can differentiate the segments but not as good as the previous case. Hence, for this particular image one dimension is sufficient to cluster the segments in the image. It is very fast and the computational intensity is also very less. This is the advantage of using PCA to reduce the dimension of the dataset.

There are many other methods using which we can improve the results obtained with the previous cases. One such method can be check whether the neighbors belong to the same cluster or not on a polling basis. For example, if there is a region of misclassification in a texture region, for each pixel we can check if all the surrounding pixels have the same cluster allotted to them. If a majority of them are not allotted to the same cluster, we can say that the particular pixel is misclassified and hence update that particular pixel's cluster to the more prominent cluster among its neighbors. If this process is repeated to all the other pixels and the whole process is run for a number of times, then the quality of the segmentation based on textures will increase.

Problem 2: Edge Detection (40 %)

a) Basic Edge Detector (Basic: 10%)

Implement two edge detectors and apply them to clean and noisy *Boat* images as shown in Fig. 3 (a) and (b). Note that you need to convert RGB images to grey-level image first.

1) Sobel Detector
   - Normalize the x-gradient and the y-gradient values to 0-255 and show the results.
   - Tune the thresholds (in terms of percentage) to obtain your best edge map. An edge map is a binary image whose pixel values are either 0 (edge) or 255 (background)
2) Zero crossing Detector
   - Normalize the LoG response to 0-255 and show the results.
   - The LoG histogram is in form of a sharp symmetric spike. Develop an algorithm to determine the two knee locations of the LoG histogram. Then, you can use them as the threshold values to partition the LoG histogram into three values -1, 0 and 1. In order to show the corresponding ternary map, you can map -1, 0, 1 to gray-level 64, 128, 192.
   - Apply zero crossing to obtain your best edge map

Compare the results obtained in (1) and (2)

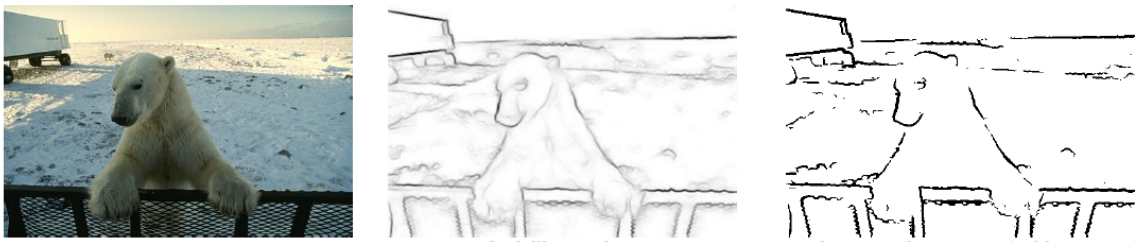
Fig. 3 (a) Original *Boat* image and (b) noisy *Boat* image

Figure 4: Animal and house images

b) Structured Edge (Advanced: 10%)

Apply the Structured Edge (SE) detector [4] to extract edge segments from a color image with online source codes. Exemplary edge maps generated by the SE method for the *Bear* image are shown in Figure 5. You can apply the SE detector to the RGB image directly without converting it into a grayscale image. Also, the SE detector will generate a probability edge map. To obtain a binary edge map, you need to binarize the probability edge map with a threshold.

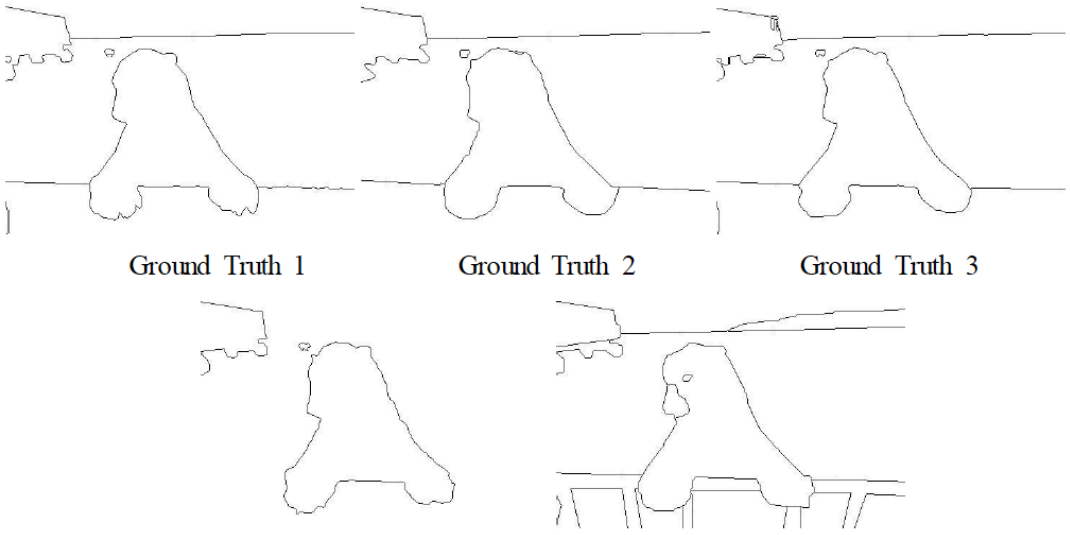
1. Please digest the SE detection algorithm. Summarize it with a flow chart and explain it in your own words (no more than 1 page, including both the flow chart and your explanation).
2. The Random Forest (RF) classifier is used in the SE detector. The RF classifier consists of multiple decision trees and integrate the results of these decision trees into one final probability function. Explain the process of decision tree construction and the principle of the RF classifier.
3. Apply the SE detector to *Animal* and *House* images. State the chosen parameters clearly and justify your selection. Compare and comment on the visual results of the Canny detector and the SE detector.



Bear | Probability edge map | Binary edge map (with p>0.2)

Figure 5: The *Bear* image and its probability and binary edge maps obtained by the SE detector

c) Performance Evaluation (Advanced: 20%)



Ground Truth 1 | Ground Truth 2 | Ground Truth 3

Ground Truth 4 | Ground Truth 5

Figure 6: Five ground truth edge maps for the *Bear* image

Perform quantitative comparison between different edge maps obtained by different edge detectors. The ultimate 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 6 shows 5 ground truth edge maps for the *Bear* image from the Berkeley Segmentation Dataset and Benchmarks 500 (BSDS 500) [5]. To evaluate the performance of an edge map, we need to identify the error. All pixels in an edge map belong to one of the following four classes:

(1) True positive: Edge pixels in the edge map coincide with edge pixels in the ground truth. These are edge pixels the algorithm successfully identifies.
(2) True negative: Non-edge pixels in the edge map coincide with non-edge pixels in the ground truth. These are non-edge pixels the algorithm successfully identifies.
(3) False positive: Edge pixels in the edge map correspond to the non-edge pixels in the ground truth. These are fake edge pixels the algorithm wrongly identifies.
(4) False negative: Non-edge pixels in the edge map correspond to the true edge pixels in the ground truth. These are edge pixels the algorithm misses.

Clearly, pixels in (1) and (2) are correct ones while those in (3) and (4) are error pixels of two different types to be evaluated. The performance of an edge detection algorithm can be measured using the F measure, which is a function of the precision and the recall.

$$\begin{aligned} \text{Precision}: P &= \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}} \\ \text{Recall}: R &= \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}} \\ F &= 2 \times \frac{P \times R}{P + R} \end{aligned} \tag{1}$$

One can make the precision higher by decreasing the threshold in deriving the binary edge map. However, this will result in a lower recall. Generally, we need to consider both precision and recall at the same time and a metric called the F measure is developed for this purpose. A higher F measure implies a better edge detector.

For the ground truth edge maps of *Animal* and *House* images (both 5 images), evaluate the quality of edge maps obtained in Parts (a)-(c) with the following:

1. Calculate the precision and recall for each ground truth (saved in .mat format) separately using the function provided by the SE software package and, then, compute the mean precision and the mean recall. Finally, calculate the F measure for each generated edge map based on the mean precision and the mean recall. Please use a table to show the precision and recall for each ground truth, their means and the final F measure. Comment on the performance of different edge detectors (i.e. their pros and cons.)
2. The F measure is image dependent. Which image is easier to a get high F measure - *Animal* or *House*? Please provide an intuitive explanation to your answer.
3. Discuss the rationale behind the F measure definition. Is it possible to get a high F measure if precision is significantly higher than recall, or vice versa? If the sum of precision and recall is a constant, show that the F measure reaches the maximum when precision is equal to recall.

(2.a) **Basic Edge Detector**

Abstract and Motivation:

Edge detection is the very first step in object detection. Object Detection is a very important application of computer vision. Edges give us the most important features from an image. Starting from object detection till image classification, edges give important information about the objects in the image. Edge detection has its applications in many fields. One such example, is character recognition. Edge detection is also needed in the field of image segmentation. Texture segmentation along with edge detection can help us specify the region of each object having the same texture. It is needed for almost all the applications of computer vision in self-driving cars, bio-medical imaging etc. With the presence of Convolutional Neural Vectors, the data points extracted have to contain all the information about an image. And for this purpose, edges in the image give us the most important information of the image. This is the application of edge detection. The motivation of this problem is to apply Sobel Edge detector and Laplacian of Gaussian Edge detectors to extract the edges from the given images.

Approach and Procedure:

There are many methods for detecting edges in images namely Canny edge detector, Sobel edge detector, Laplacian of Gaussian edge detector, etc. In this problem, we will be using the two edge detectors namely Sobel edge detector and Laplacian of Gaussian edge detector. First of all, edge is something where there is a large change in the intensity values between the neighboring pixels. So, we have to calculate the change in the intensity value and this is referred to as the gradient change. The Sobel edge detector helps us extract the gradient value of each pixel by application of filter masks.

There are 2 gradients in every image, the X-gradient and the Y-gradient. The X-gradient of each pixel helps us identify the change in the intensity values along the horizontal axis and the Y-gradient of each pixel helps us identify the change in the intensity values along the vertical axis of the image. The Sobel masks for these operations mentioned above are given below:

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

Gx

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

Gy

By applying these masks, we will get the X-gradient and the Y-gradient values of all the pixels. The filter response is not in the range of 0 to 255. So, we have to normalize the response values to 0-255. After normalizing, the x-gradients and the y-gradients can be plotted as different images. To get the edge map, we have to find the square root of the sum of squares of the x-gradient and the y-gradient values. After this, we will get values between 0-255, but an edge map can only have values 0 or 255. Hence, the next step is to choose the threshold values. This can be obtained CDF of the histogram obtained for the pre-final image. From the CDF, we can choose the threshold values for making the image binary. For example, different threshold values corresponding to the percentage of pixels such as 90%, 95%, etc., can be chosen and the images can be compared to identify the best edge map.

The summary of Sobel edge detector is given below:
Step 1: Apply the filter masks to get the x-gradient and the y-gradient responses
Step 2: Normalize the values of filter responses to 0 and 255
Step 3: After the normalized values are obtained, use them and allot the pixel value equal to the square root of the sum of the squares of the x-gradient and y-gradient values.
Step 4: Once the values are normalized, fix a threshold based on the CDF value to create an edge map.

The next detector to be used is the Laplacian of Gaussian(LoG) zero crossing detector. The laplacian gives us the measure of 2nd Spatial derivative of the image. The response of LoG is in such a way that there is a symmetric peak in it. The 2D LoG function centered on mean 0 and standard deviation σ is given below.

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

This equation can be framed in the form of a matrix for a specific σ value. The filter is designed in such a way that when there are no edges in a region, then the response of the filter is 0. Only when there is an edge present, then the value is non-zero. The sigma values can be changed to modify the region around a pixel in which we want to consider the presence of an edge. The laplacian kernels are given below.

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

These kernels give the second derivate discrete approximation of a pixel. The LoG kernel is obtained by convolving the above laplacian kernels with the Gaussian filter. Hence the name Laplacian of Gaussian. Different LoG filters can be obtained by changing the σ values. Depending on the image, different values of σ will give different edge maps. After getting the LoG response, we need to normalize the LoG response

between 0 and 255. The next step is to detect the knee points in the LoG response of the image. This will determine the threshold values needed for obtaining the edge maps. As an intermediate step, we can map the regions in the response based on the threshold values to -1, 0 and 1. The response can then be visualized by mapping the values -1, 0 and 1 to 64, 128 and 192 respectively.

The next step is to obtain the edge maps. This is where the zero-crossing detector is used. This can be done by checking the gradient changes in all directions. From the previous step, the pixels that have value 0 contain an edge. When there is a sign change in one of the surrounding pixels having a 0, then there is an edge in that particular direction. Using this, we can find the orientation of the edges as well. By repeating this process for other values -1 and 1 we can obtain an edge map. So, if one of the values surrounding a particular pixel is not the same as that center pixel, then we say there is an edge and that pixel can be allotted a value 0 (corresponding to an edge). This is how the edge map can be obtained.

The summary of the LoG filter is given below.

Step 1: Generate the LoG kernels by convolving the Laplacian kernels and the Gaussian kernels.
Step 2: Get the response of the image by applying the convolved LoG filter.
Step 3: Normalize the edge values to get a preliminary edge image.
Step 4: Obtain the histogram and find the knee points
Step 5: Create the mapping between the thresholds and -1, 0, 1 to obtain he ternary image
Step 6: Check for the surroundings of a pixel, and mark if an edge is present or not

The LoG filter and Sobel Filter are very similar in operation. The LoG kernel used for edge detection with $\sigma = 1.4$ is given below.

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

The above-mentioned procedure was followed for Sobel and LoG edge detectors and their results are discussed below.

Experimental Results:

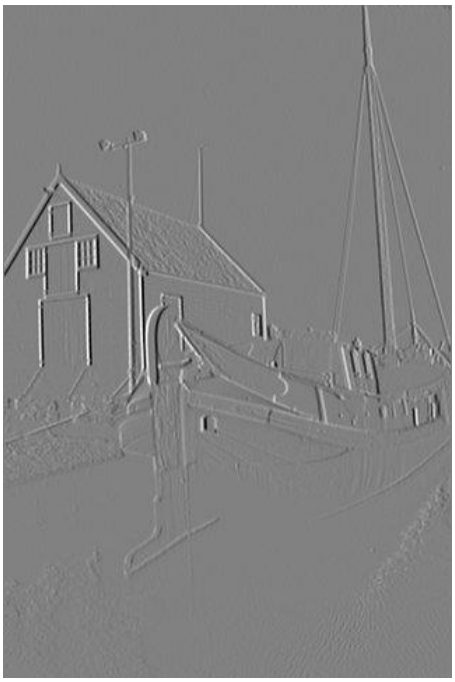


Figure 22: Boat X-Gradient

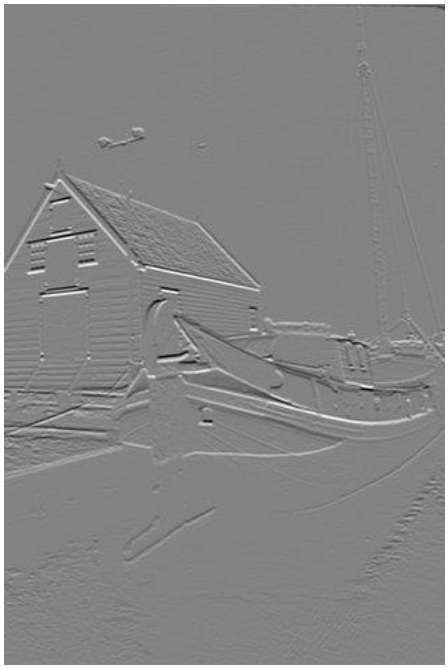


Figure 23: Boat Y-Gradient

Figure 24: Boat Noisy X-Gradient

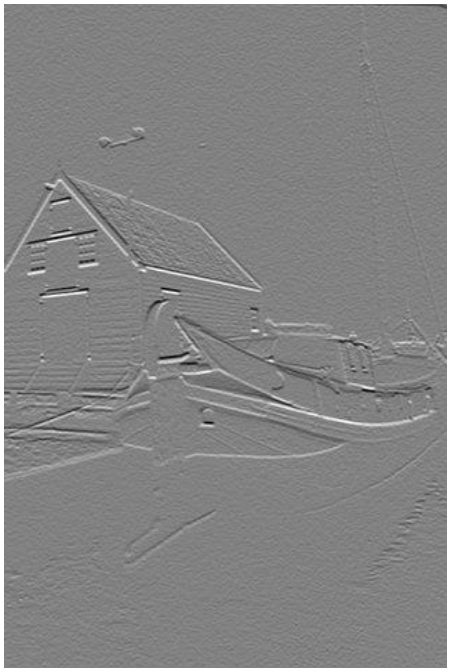


Figure 25: Boat Noisy Y-Gradient

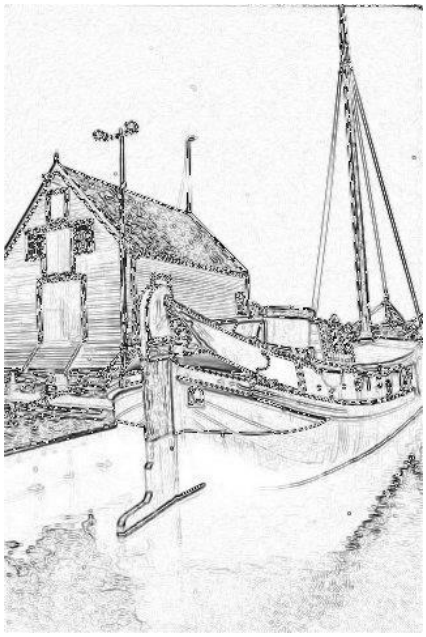


Figure 26: Boat Normalized Sobel

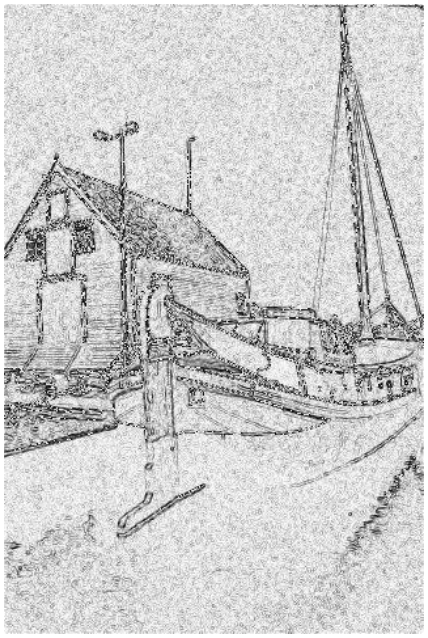
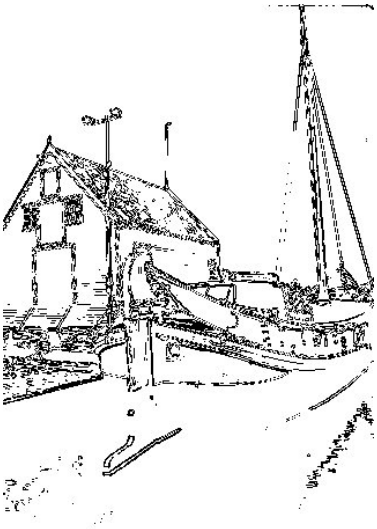
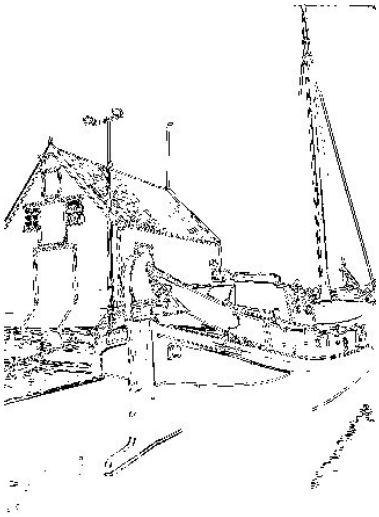


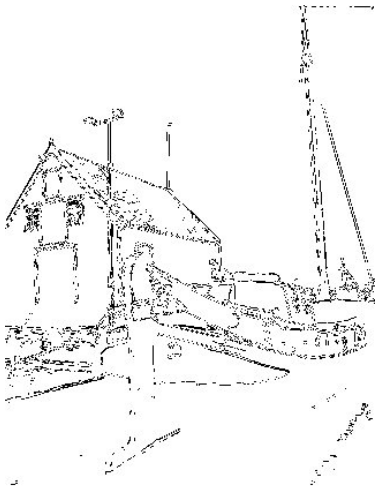
Figure 27: Boat Noisy Normalized Sobel



Threshold: 90%

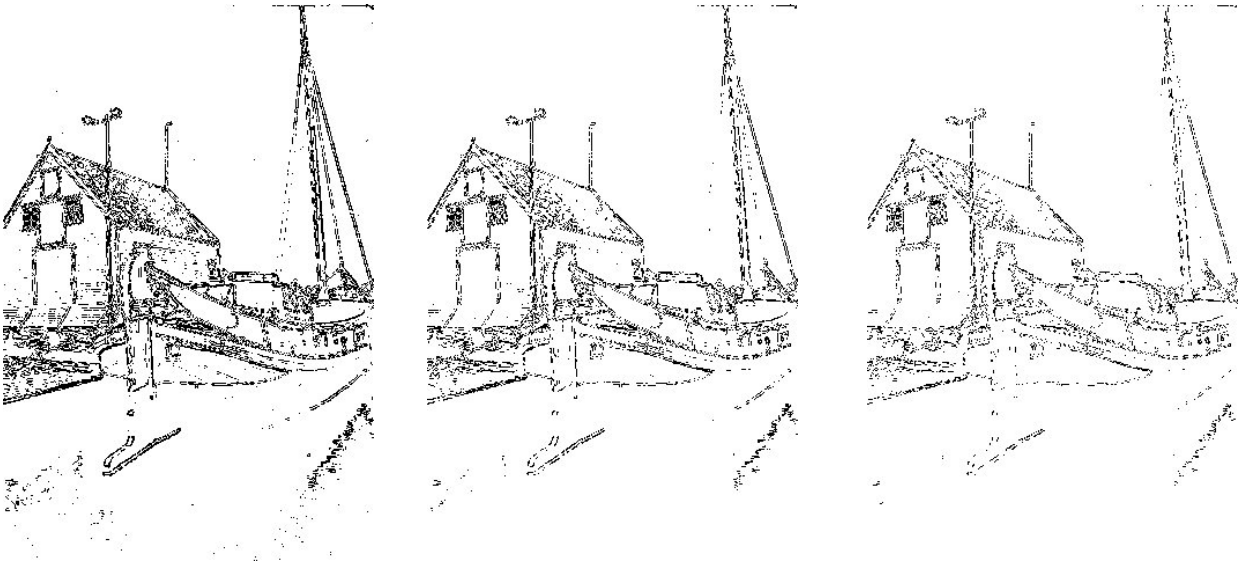


Threshold: 94%



Threshold: 96%

Figure 28: Boat Edge maps for different Thresholds (Sobel)



Threshold: 90% Threshold: 94% Threshold: 96%

Figure 29: Boat Noisy Edge maps for different thresholds (Sobel)

Figure 30: LoG response for Boat

Figure 31: LoG response for Boat Noisy

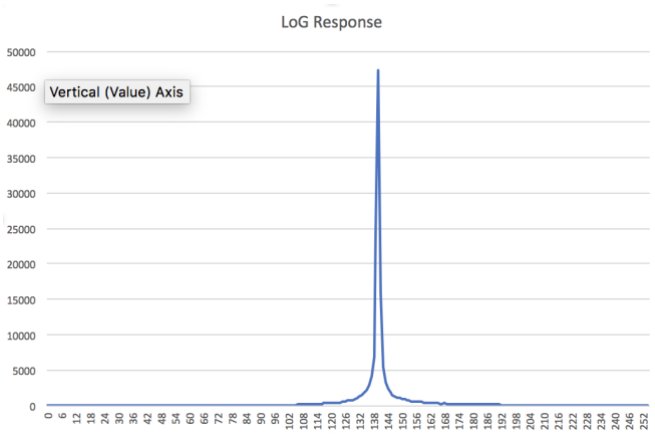


Figure 32: Response of LoG Detector for Boat

Figure 33: Boat Ternary Map (LoG)

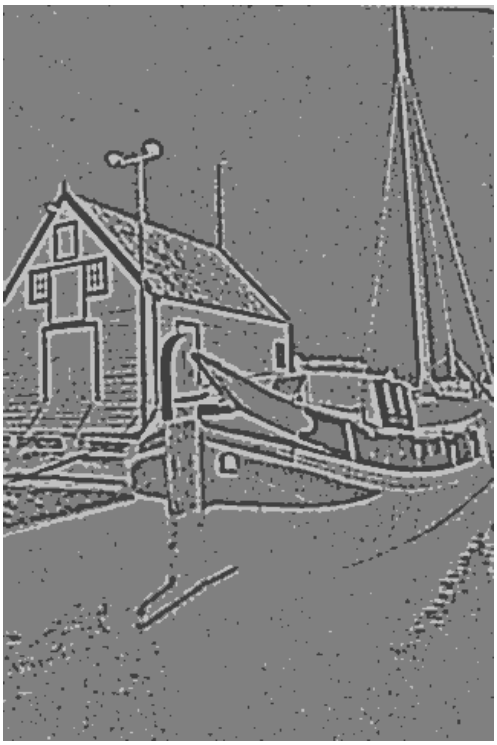
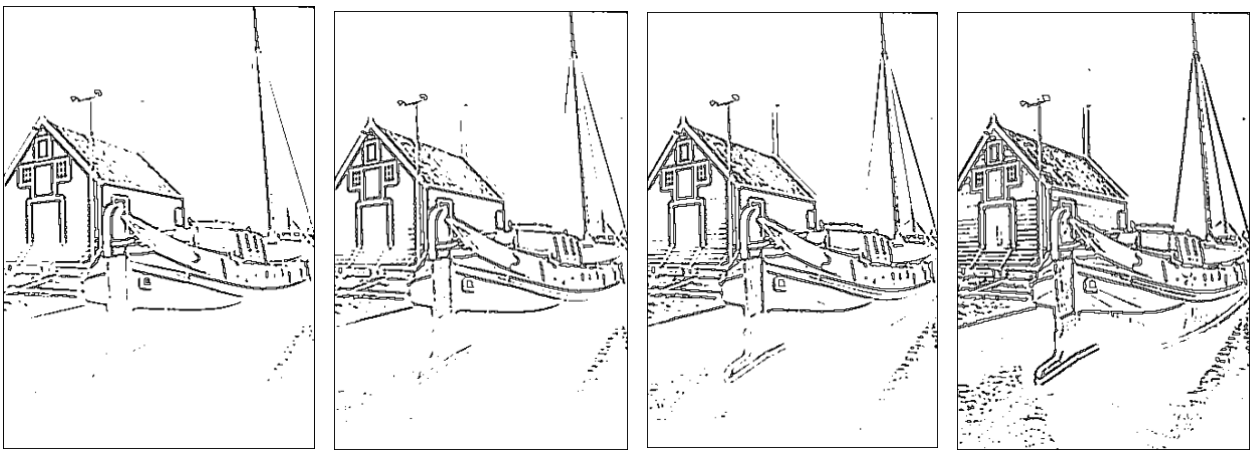
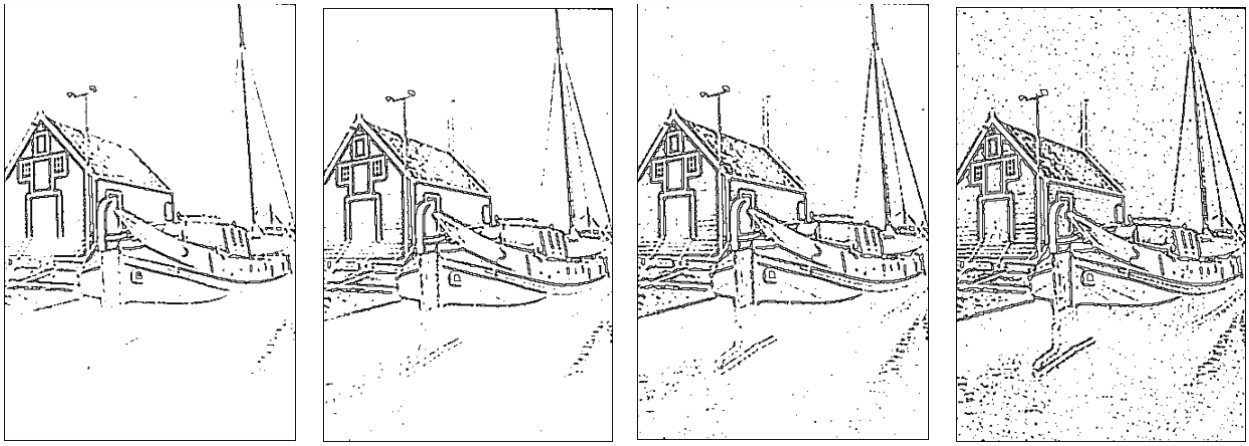


Figure 34: Boat Noisy Ternary Map (LoG)



Thresh: 6%, 94% Thresh: 8%, 92% Thresh: 10%, 90% Thresh: 15%, 85%

Figure 35: Boat LoG detect Edge maps for different thresholds



Thresh: 6%, 94% Thresh: 8%, 92% Thresh: 10%, 90% Thresh: 15%, 85%

Figure 36: Boat Noisy LoG detector Edge maps for different thresholds

Discussion:

The x-gradient and the y-gradient obtained using the Sobel Edge detector are given in figures 22 and 23 for boat image and in figures 24 and 25 for the noisy boat images. When we compare these with the original images, we can see that there is a sharp change in the x-axis when compared to the change in the y-axis. This is the reason why the edges in the y-gradient image are less prominent when compared to those in the x-gradient image. These images are obtained after applying the LoG filter to the image and the response is normalized to the range of 0-255.

The next step was to get the normalized image containing both the x and y gradients. The images shown in figures 26 and 27 represent the normalized image of boat and noisy boat respectively. This is somewhat similar to a probability edge map. A probability edge map is nothing but a matrix which stores values between 0 and 1. The value stored denotes the numerical value conveying the probability of an edge being present there. After this step, the image has to be binarized. This can be done by thresholding the image based on value obtained from the CDF of the image. The images shown in figures 28 and 29 contain the images obtained for different thresholds such as 90%, 94% and 96%. For example, if the threshold value is 90%, the normalized value CDF value at which 0.90 is crossed is taken as the threshold value. This is also done for other values such as 94% and 96%. Only after the threshold is applied, the binary edge map can be obtained, and hence this is a very important step.

The images given in figure 30 and 31 contain the normalized LoG responses obtained after applying the LoG kernel on the boat and the noisy boat images. When compared with the responses of the Sobel filter in images 22 and 23, we can see that the LoG response is detecting the edges more when compared to the Sobel filter responses. The next step after obtaining the LoG is response is to get a ternary map from which the final edge map can be obtained. For obtaining the ternary map, the response has to be normalized and threshold has to be applied. We can allot values -1 to pixels having a value lesser than the lower threshold and +1 to the pixels having a value greater than the upper threshold. All the pixels lying between the threshold values are allotted to be 0. The threshold values are chosen from the histogram. The histogram of the LoG response for the boat image is shown in figure 32. From this we can get the CDF. From the normalized CDF, we can get the lower and upper threshold for getting the ternary map. The ternary map obtained for the normal boat and the noisy boat images are shown in figures 33 and 34.

From these ternary images, we can obtain the final edge maps by applying the zero-crossing filter. The zero-crossing filter looks for changes between -1, 0 and 1. If a pixel has a value -1 and even if 1 of the pixels surrounding that pixel is not -1, there is a sign change and we assume that there is an edge at that location. This process is repeated for all the other values as well. The edge maps obtained are given in figure 35 and 36 for boat and noisy boat images. Different thresholds such as 6%-94%, 8%-92%, 10%-90% and 15%-85% are used to obtain the different edge maps shown in the figures 35 and 36.

The final edge map obtained with the Sobel and LoG edge detectors can be compared to point out the differences between these two methods of edge detection. From the visual understanding, we can say that the image obtained with the LoG filter has a better result. If we compare the noisy boat edge maps using these detectors, we can see that the LoG has a lot of grains in the edge map for the noisy image. The edge map obtained with the Sobel detector for noisy boat image is similar to that of the edge map of the non-noisy boat image. This is mainly due to the reason that Laplacian of Gaussian edge detector is very sensitive to noise. The advantage with Sobel filter is that it is very simple and hence requires less computational power unlike the LoG. The LoG filter fails to detect the corners where the range of the color intensity is very large. The main advantage with the LoG filter is that the Signal to noise ratio is very high when compared to other edge detection methods. The Sobel edge detector gives more priority to the diagonal edges when compared

to the horizontal and the vertical ones. Since Sobel is a first order edge detector, it can detect only the change in the intensity values. LoG is a second derivative edge detector, it is more sophisticated than the first derivative methods and it can detect gradient changes in the entire 2D space. We can see from the results that LoG has got a better edge map when compared to that of the Sobel edge detector. These are some of the main differences between the Sobel edge detector and the Laplacian of Gaussian edge detector.

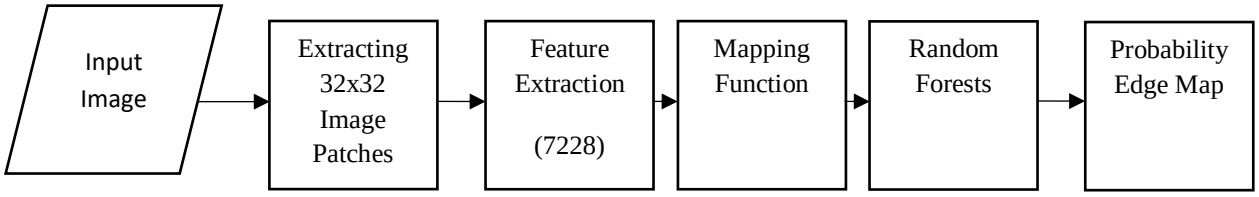
(2.b) **Structured Edge**

Abstract and Motivation:

We know that edge detection is a very important part of computer vision and biomedical imaging. We have seen that it in the previous part. The detectors such as Sobel and LoG are very simple and not suitable in this era where object detection is needed for autonomous cars which has a great impact on us. Hence, the need for better edge detection techniques is never ending. The evolution of probabilistic methods as well as high computational power has enabled the invention of more sophisticated algorithms. One such algorithm is the Structured Edge detection algorithm. This algorithm was developed in the recent years and was published by Microsoft Research. The purpose of this problem is to Structured edge detection algorithm and obtain the edge maps of the given two images.

Approach and Procedure:

The flow chart of the Structured Edge detection is given below:



Step 1:

The algorithm considers a 16x16 Segmentation masks from 32x32 Image patches. The first step is to extract these 32x32 image patches.

Step 2:

The next step is to extract the features from these image patches. The features are divided into two channels namely pixel lookups (the direct value) and pairwise differences (differences between different image features). 3 channels are computed in CIE-LUV color space with gradient at 2 resolutions namely original and half space. The gradient channel is also split into 4 channels based on the orientations. So, the total number of channels is 2+3+8=11 channels. The channels are then blurred with a radius 2 triangle filter, after which they are down-sampled by factor of 2 giving 3328 features. Similarly, considering the pairwise pixel

lookup, a large triangle blur is applied and down-sampled to a 5x5 patch giving us 300 features per channel getting to a total of 7228 features.

Step 3:

The image patches are 32x32 and the segmentation masks are 16x16 and hence there need to be a mapping function. This mapping is created by considering pairwise locations in the patch and checking if they belong to the segmentation mask or not. By doing this, the similar indices in the segmentation masks are represented by 1 index instead of 2 or more. This is the idea of the mapping function.

Step 4:

The next step is to train the Random forest model. Using the features extracted from the image, random data points and random features are chosen and the random forest model is trained. The random forest algorithm is explained below. The model is trained using the BSDS500 Segmentation dataset. Once the training is done, the presence of an edge is given by a probability map when a test image is passed to the random forest algorithm. With this probability map, the edge map can be obtained.

Algorithm of Random Forest:

One of the important concepts in structured edge Detection is Random forests. Random forest is an important algorithm in machine learning and has application in both classification and regression. The basis of a random forest is a decision tree. A decision tree is a structured decision-making system which either gives values or categories. For example, if we consider a simple decision-making system which predicts whether it will rain tomorrow. We consider the factors such as the season, the humidity etc. So, a decision tree considers these features and then asks questions to which the answers are very simple and decides the output based on the feature values passed. This is just a simple case of a decision tree. In real world, there may be millions of features like these. There are 3 types of nodes in a Decision tree namely Decision nodes, Chance nodes and end nodes. Usually in decision tree, the process starts with the root and ends up in a leaf, like a tree which is present upside down. At each node of the tree, a binary output is produced and when this propagates through all the nodes, we will have an answer in the end, i.e., the leaf.

One of the main problems with decision tree is that they tend to over-fit the data. When the model is over-fit it becomes less generic and will not have high accuracy for the unseen data. To prevent this, Random Forests can be used. The main principle of Random Forest classifier is, selecting random features and random data points for each decision tree and performing a majority polling to find out the output that is given by a large number of decision trees. In this method, a number of decision trees are created with different data points that are picked randomly for each tree and features also randomly picked for each tree. So, we will have many number of trees, each built on random features. So, to decide the final output of the random forest, majority polling is used. The value that is given by the majority number of trees is taken as the final output of the random forest. This is how a random forest classifier works.

Experimental Results:

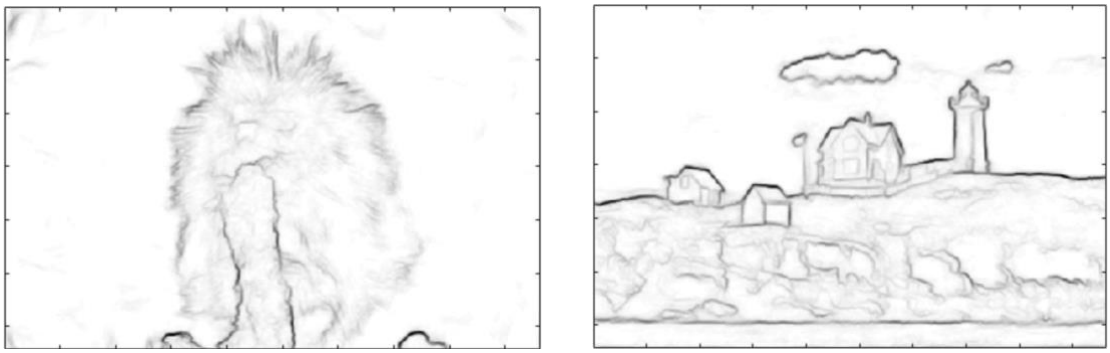


Figure 37: Probability maps for Animal and House Image

p>0.05 p>0.10 p>0.12

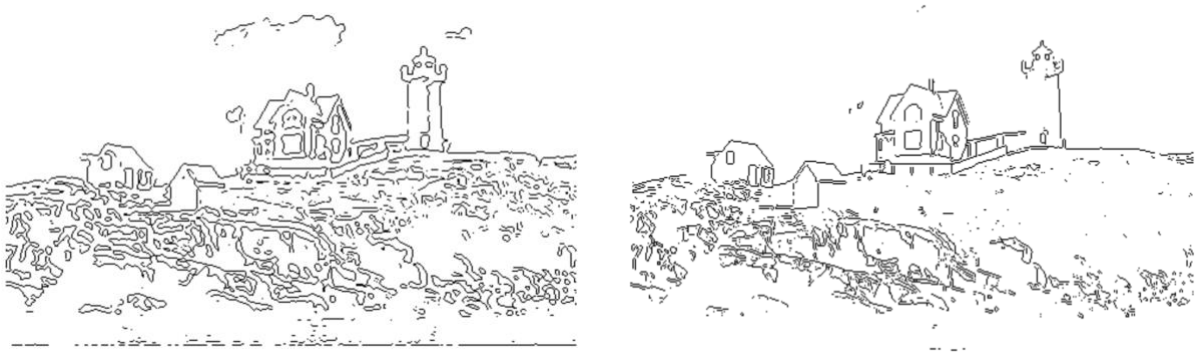
Figure 38: Edge Maps of Animal image obtained for different probabilities

p>0.15 p>0.2 p>0.23

Figure 39: Edge Maps of House image obtained for different probabilities

LoG Edge Detector Sobel edge Detector

Figure 40: Animal Edge images obtained using LoG filter and Sobel edge detector



LoG Edge Detector Sobel Edge Detector

Figure 41: House Edge maps obtained using LoG edge detector and Sobel Edge detector

Discussion:

The probability maps for the Animal and House image obtained with the Structured Edge Detector is given in figure 37. This image is the probability map obtained and this is represented as an image by multiplying the probability values by 255 to generate the color values for each pixel. After obtaining the probability map, the next step is to implement a thresholding to get the binary edge map from the image. This is done by choosing a particular value for the probability. If the probability value is greater than a certain value, we can say that there is an edge and vice versa.

The edge maps obtained for animal image for different threshold values are shown in figure 38. The probability threshold values chosen are 0.05, 0.10 and 0.12. The edge maps obtained for house image for different threshold values are shown in figure 39. The probability threshold values chosen are 0.15, 0.20 and 0.23. The edge maps obtained using the LoG and Sobel detector for the animal image are shown in figure 40. The corresponding edge maps for the house image are shown in figure 41.

There are certain parameters we need to set before the execution of the structured edge detection program. The multiscale parameter decides the resolution of the image to be used for the algorithm. If it is set True, three resolutions namely original, double the original resolution and half the original resolution are used in the algorithm. To get the final result from these images, the averages are taken after getting them back to their original resolution. The tree evaluation parameter decides the number of decision trees to be used in an alternating fashion. If this is set as 4, then 4 trees are evaluated at each adjacent location. These were the parameters that were chosen for the animal and house images are given below:

Sharpen= 2; Number of trees= 4; multiscale= 0 (since the time taken is very high if it is set as true)

From the visual analysis, we can say that the edge maps obtained for the images with the Structured edge detector are very good when compared to that of the LoG and Sobel edge detectors. This is mainly because of the sophistications of the Structured Edge Detection algorithm. The RF classifier and the number of features chosen from the image can help us identify the smallest structures in the image as well. The LoG and Sobel are very simple edge detection algorithms that depend only on the gradient values. But the Structured edge detection considers even the smallest edge, by testing it with the trained model. Hence, the results greatly depend on the trained model. In LoG and Sobel there is no prior training, and what we see as the edge map is just the filter response and there has been no processing on that image. In structured edge detection, there are many image processing techniques that are used before the image is passed to the model. Some of the techniques include morphological filtering techniques, noise removal, etc., These are some of the reasons why the Structured Edge Detection technique is very good when compared to the LoG and the Sobel Edge detectors. The images can be compared with the parameter called f-measure which is discussed in the next section.

(2.c) **Performance Evaluation of Structured Edge Detection**

Abstract and Motivation:

There are different edge detection techniques and each technique has its own advantage and disadvantage. To compare the efficiencies of the different edge detection techniques, a common parameter must be measured from the images obtained with each of the different edge detection techniques. Only with the help of this attribute can we compare the pros and cons of the edge detection techniques. Just by a visual analysis we cannot compare the different edge detection techniques. This parameter is called as the f-measure. The f-measure can be obtained only by comparison of the edge map obtained from the edge detection technique and the ground truth. Ground truth is defined as the true edge image that the human being understands from the image. Different people consider different edges and that's what makes this is f-measure an important parameter. The f-measure is calculated by comparing the edge maps with different ground truths

and taking the average of them. The objective of this problem is to obtain the f-measure values for different edge detection techniques such as Sobel, LoG and Structured Edge detection.

Approach and Procedure:

The performance of an edge detector can be obtained only with the help of the ground truth images. Different human beings have different ideas of edges in an image. To accommodate this diversity, a number of ground truths are taken to calculate the f-measure. We need to compare the edges obtained with the edge detection technique and the ground truth pixel by pixel. The following 4 measures needed to be calculated from an image for finding out the f-measure.

True Positive: The pixels that are considered as edges in both the ground truth and the edge map
False Positive: The pixels that are considered as edges in edge map but not in the ground truth
True Negative: The pixels that are not considered as edges in both the edge map and the ground truth
False Negative: The pixels that are considered as edges in the ground truth but not in the edge map

What we are interested in is the values of true positive and true negative as these values consider the presence and absence of edges in edge map and the ground truth. The f-measure is a function of two parameters called as precision and recall. These two values depend on the values of true negative, true positive, false negative and false positive. Precision and recall are given by the following formulas:

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

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

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

The Precision and Recall values are somewhat inversely proportional to each other. The precision value can be higher if a lower probability threshold value is used, but at the same time the recall value reduces. This is the type of relation that exists between the precision and recall values. We have 5 ground truth values given for the house and the animal image. We can also compare the Sobel edge detector, LoG edge detector and the Structured edge detector with the help of the f-measure values.

Experimental Results:

F Measure values: Using Structured Edge Detector

Animal Image

Ground Truth Number	Mean Precision	Mean Recall	F-Measure
1	0.747734	0.588935	0.658902
2	0.771903	0.520897	0.622033
3	0.447885	0.426619	0.436993
4	0.776738	0.619073	0.689001
5	0.693593	0.551801	0.614625

Table 1

House Image

Ground Truth Number	Mean Precision	Mean Recall	F-Measure
1	0.696624	0.815224	0.751272
2	0.795864	0.646533	0.713469
3	0.784508	0.645953	0.708520
4	0.557339	0.711120	0.624908
5	0.525885	0.700567	0.600786

Table 2

Final F Measure values

Image	Structured Edge	LoG	Sobel
Animal	0.683465 (0.12)	0.308491	0.301287
House	0.782232 (0.23)	0.3661	0.312934

Table 3

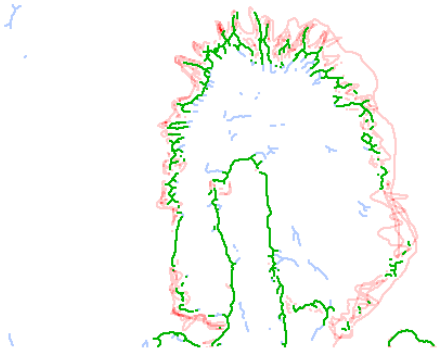


Figure: Edge map of the animal image for best threshold (0.12)

Figure: Edge map of the house image for the best threshold (0.23)

Discussion:

The f-measure values obtained for different ground truths for the animal image are given in table 1 and the f-measure values obtained for different ground truths for the house image are given in table 2. Along with the f-measure values, the values of the mean precision and mean recall for each ground truth are also given in the tables. As we can see from the table, different f-measure values are obtained for different ground truths. Some ground truth may be very extensive and some may be very lethargic. To get a more diverse f-measure value, more than 5 ground-truths have to be used.

The final f-measure values obtained for the house and animal images are given in table 3. The corresponding threshold values are also given for the Structured edge detection technique in brackets. From the results obtained for Structured edge maps in figures 38 and 39 obtained using the thresholds given in

table 3 we can see that the edge maps are very good for these threshold values. For the other threshold values, there are either more disturbances or less number of edges.

The edges evaluation function gives us a vector of Recall and Precision values from which we can obtain a vector of f-measure values. The threshold value corresponding to the highest f-measure values are used to obtain the edge maps in figures 38 and 39. The f-measure values obtained for LoG and Sobel are also included in the table 3. We can see that the f-measure values for Sobel and LoG are less than half of the f-measure values for the structured edge detection method. This can be understood from the quality of the images as well. Thus, structured edge detection gives the best result when compared to the Sobel and LoG edge detectors.

The edge maps obtained for the animal and the house image are given above. In this image, the red color represents the false negative values, the green color represents the true positive values and the blue color represents the false positive values. The threshold chosen for the animal image was 0.12 and for the house image it was 0.23 as these values gave the best f-measure values.

The f-measure value is image independent. The value of f-measure greatly depends on the ground truth edge maps very much. If we consider the animal image, it is very difficult for us to clearly visualize the edges. Even from the edge maps obtained, we can see that. But the ground truth images have only an outer lining of the animal in the image. But the edge maps obtained contain all the spikes and the inner contours of the animal as well. This is the main reason why f-measure value is low for the animal image. The edge detection algorithms will consider all the contours, whereas the ground truth contain only the animal's contour as the edges. If we consider the house image, the edges are clearly distinguished. The ground truths will also have the more prominent edges similar to that of the edge maps obtained. This is the main reason why the f-measure values are high for the house image when compared to that of the animal image. And this is mainly because of the ground truth images, as humans will not give importance to the spikes inside the animal.

f-measure can be written directly in terms of the true positive, true negative, false positive and false negative values as given below:

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

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

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

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

We can see that Precision and Recall values are inversely proportional to each other. If P increases, then R decreases. Hence it is not possible to get higher f-measure values just by increasing the value of P or R as the sum is constant. The maximum f-measure value can be obtained only when P=R. When P=R, the false negative and the false positive values become 0. When P=R,

$$\#FP = \#FR = 0$$

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

$$F = 1$$

Thus, the maximum value of f-measure is obtained when Precision and Recall values are equal.

As a conclusion, we can say that the Structured edge detector is more robust when compared to the other types of edge detectors and this can be understood from the images of the edge maps as well as from the values of the f-measure.

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

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

SIFT [1] and SURF [2] are effective tools to extract salient points in an image. Extract and show both SIFT and SURF features from the two vehicle images in Figure 7. Compare their results in terms of performance and efficiency according to their strength and weakness. You are allowed to use open source library (OpenCV or VLFeat) to extract features.

(a) Optimus prime truck

(b) Bumblebee car

Figure 7: Vehicle images

(b) Image Matching (Basic: 10%)

You can apply SIFT and SURF to object matching. Extract and show SIFT features from the two racing car images in Figure 8. Then, show the corresponding SIFT pairs between the two images. Repeat the same task for the SURF feature.

(a) Ferrari_1 (b) Ferrari_2

Figure 8: Ferrari racing car images

The matching may not work well between different objects and against the same object but with a large viewing angel difference. Perform the same job with the following two image pairs:1) *Ferrari _1* and *Optimus prime truck*, and 2) *Ferrari _1* and *Bumblebee car*. Show and comment on the matching results. Explain why it works or fails in some cases.

(c) Bag of Words (Advanced: 10%)

Apply the k-means clustering to extracted SIFT features from three images (*Optimus prime truck*, *Bumblebee car*, and *Ferrari_1*) to form a codebook. The codebook contains K=8 bins, where each bin is characterized by the centroid of the SIFT feature vector. In other words, each image can be represented as

a histogram of SIFT feature vectors. This representation is called the Bag of Words (BoW). Create codewords for all four images (*Optimus prime truck, Bumblebee car, Ferrari_1 and Ferrari_2*), and match Ferrari_2's codewords with other images. Show the results and discuss your observation.

(3.a) **Extraction and Description of Salient Points**

Abstract and Motivation:

Image classification is becoming an important task in understanding what the image is trying to convey. For any image classification algorithm, the first thing needed is the important features in the image. By important features, what we mean is the key-points which have the capability to represent the important features of the image. There are 2 types of features that are extensively used for understanding the image. They are Scale Invariant Feature Transform (SIFT) and Speeded-up Robust Features (SURF). The objective of this problem is to extract the SIFT and SURF key points from the given 2 images.

Approach and Procedure:

The Scale Invariant Feature Transform is an important algorithm for feature detection. As the name suggests, it is invariant to rotations, scaling and viewpoint. The SIFT algorithm is computationally very intensive. The algorithm of SIFT is explained below:

Step 1: Building a scale space

This is done by creating different octaves based on the different sizes. The SIFT algorithm has 5 blur images per octave and has a total of 4 octaves. The blur can be created by applying a Gaussian filter to the image. So, the objective is to create 5 blurred images for each octave using different values of sigma in the Gaussian kernel. The first octave is obtained by doubling the original image, the second is the original image itself, the third is half of the original image and the last octave is of 1/4th the original image. And Gaussian blur with different sigma values are applied to the image in each octave to get 5 blurred images per octave. So now we have constructed the scale space which has 4 octaves and 5 images per octave.

Step 2: LoG approximations

The next step is to identify the edges and corners from the image. For this, the laplacian of Gaussian edge detector can be used. But the problem with this is that it is computationally more intensive and it is very noise sensitive. Hence, we use an approximation called the 'Difference of Gaussians (DoG)'. This is done by subtracting 2 images in each octave. Though this is not equivalent to the LoG detector, DoG is a very good approximation of the LoG. So now we have 4 DoG images in each octave.

Step 3: Finding the key-points

The next important step is finding the key-points from the DoG images. This can be done by identify the maxima/minima points in the image. This is done by considering three consecutive images out of the five and omitting the top and the bottom images. These 3 images are scanned pixel by pixel, and comparing the neighbors for the same pixel in all the three images. Using this, we can find the maxima and the minima points in the images. After finding the maxima and the minima points we can find the subpixel maxima and minima using the second-degree Taylor series approximation for every maxima/minima pixel and its surrounding neighbors. On differentiating the Taylor series approximation, we can find the location of the subpixel maxima and minima points. These points are the key-points of the images.

Step 4: Reducing the key points

After step 3 we will have numerous number of keypoints. The next step is to remove the keypoints that are not important. This can be done by removing the keypoints that lie along an edge and the points having a very low contrast. The Hessian matrix can also be used to retain the keypoints that represent corners. Corners are very important features from an image.

Step 5: Keypoint Orientations

In this step, we have to collect information about the orientation of a keypoint. This can be done by finding the orientations of the edges in a particular region and then allotting the pixel with the most prominent orientations from that region.

Step 6: Generation of feature vector

The next step is to get a feature vector for each generated keypoints. This is called as the finger print of each keypoint. For every keypoint, a 16x16 window is taken and divided into 4 sub-blocks and the orientation of each of the blocks is divided into 8 bin values between 0 and 360. This is repeated for all the 4 blocks and we will have 4*4*8=128 numbers for each keypoint. These are then normalized to get the feature vector for each keypoint.

This is the algorithm for SIFT. In SIFT, each dimension has a 128-dimensional feature vector, hence it is a slow process. To remove the problem of computational intensity with SIFT, Speeded-up Robust Features was invented. The algorithm of SURF is similar to that of SIFT, except for some changes.

The SURF algorithm uses Box Filter instead of DoG filters. Instead of using Gaussian averaging filter, the box filter is used on the integral of the images and hence it is very fast when compared to SIFT as it can be applied on images with different scales in parallel. The BLOB detector is used to get the important points from the image. The Hessian matrix is the principle of the BLOB detector. Once the keypoints are detected, to find the feature vector SURF uses the wavelet responses. Similar to the SIFT algorithm's feature description, the SURF also gets the region around the keypoint and then gets the wavelet responses of each block surrounding the keypoint. To check the brightness of the keypoint which is needed for the feature description, the sign of laplacian detected in the previous step is used. The dimension of the feature vector is 64 unlike the SIFT which enable faster computation. This can be extended to 128 dimensions for more accurate results as well. The SIFT and SURF algorithms are almost similar except for the use the BLOB detector and wavelet response to get the feature descriptor.

Experimental Results:

Figure 42: Best 200 Keypoints obtained using SIFT for Optimus Prime

Figure 43: Best 200 Keypoints obtained using SIFT for Bumblebee

Figure 44: All Keypoints obtained using SIFT for Optimus Prime

Figure 45: All Keypoints obtained using SIFT for Bumblebee

Hessian Threshold: 7000

Figure 46: 200 Keypoints obtained using SURF for Optimus Prime

Hessian Threshold: 4000

Figure 47: 200 Keypoints obtained using SURF for Bumblebee

Figure 48: All Keypoints obtained using SURF for Optimus Prime

Figure 49: All Keypoints obtained using SURF for Bumblebee

Discussion:

The SIFT and SURF keypoints were extracted using the OPENCV library. The top 200 keypoints obtained using SIFT features for the Optimus Prime truck and the Bumblebee car are given in figures 42 and 43. All the default keypoints given by the SIFT features for the Optimus and Bumblebee car are given in figures 44 and 45.

The SURF keypoints using different Hessian thresholds for the Optimus Prime truck and the Bumblebee car are given in figures 46 and 47. All the SURF keypoints for the two images are shown in figures 48 and 49. By comparing the images in figure 44 and 48 which show the keypoints given by SIFT and SURF for Optimus prime truck, we see that SURF gives better keypoints when compared to SIFT. If we consider the 200 keypoint version obtained using SIFT and SURF for Optimus Prime truck in figures 42 and 46, we can see that SURF has given better keypoints. As we can see, the SURF has given keypoints from the truck itself, that are on the edges of the truck, that are on the corners of the truck and we can be sure that these keypoints give important information about the images. Whereas, if we consider the keypoints given by SIFT there are many points on the background and some keypoints on the design on the truck which are not important at all.

If we consider the keypoints for the bumblebee car, both SIFT and SURF have given more or less similar keypoints. This is because, the bumblebee car image has not got any background. Also, in the case of the bumblebee car it seems that both SIFT and SURF features are giving us useful information. The default SURF Hessian Matrix of 400 has given a lot of keypoints. To reduce the number of keypoints in SURF, the hessian matrix threshold has to be increased. The hessian matrix threshold of 4000 for bumblebee car and 7000 for the Optimus prime truck are used to get around 200 keypoints for SURF. The SURF can give us higher number keypoints in lesser time intervals. The power of the algorithms can be understood only when there is noise, or if the image is subject to rotations and homographic projections. SIFT and SURF both seem to give similar accuracies, but SURF is very much faster when compared to SIFT. Also, in presence of noise SIFT seems to perform better than SURF. And when there is a rotation in the image, SURF seems to perform better than SIFT. These are some of the differences between the SIFT and SURF keypoint detectors. They have to be carefully chosen based on the image.

(3.b) **Image Matching**

Abstract and Motivation:

Image matching is an important technique in computer vision. Image matching is needed for panoramic image stitching and in homographic projections. The main objective of image matching is to match two portions of one image and the second. This can be done only after the extraction of features from the images. The features that can be used for image matching can be extracted from SIFT, SURF, ORB etc., This is an important application as this helps the machine to stitch the images based on these keypoints and then understand the environment especially in real time machine imaging. The objective of this problem is

to implement image matching between the given Ferrari 1 and Ferrari 2 cars after extracting their SIFT and SURF features and also between (i) Ferrari 1 and Optimus prime (ii) Ferrari 1 and bumblebee car.

Approach and Procedure:

The procedures for extracting the SIFT and SURF features from an image are explained in part a. The next step is to use an image matching algorithm to match the keypoints extracted from the two images. There are many methods for doing this namely, Brute Force matcher and FLANN based matcher. Both these methods are somewhat similar and the Brute Force matcher has been used here for image matching.

The algorithm for the brute force matcher is very simple. Each keypoint is matched with a keypoint in the second image based on the distance measure. The keypoint may simply be matched with the keypoint that has the closest distance in the second image. Another alternative for this is to use the k-nearest neighbor method. In this method also, we map a keypoint from one image to another based on the nearest keypoint in the second image. Using the second method, we can specify the number of best matches to be returned on the matched image unlike the first one in which all the nearest matches are returned in the matched image.

We do not want all the matches to appear in the matched image. We want only the best matches to be on the matched image and the number has to be small. For this we can use the ratio test. Using the k-nearest neighbor based matching, we can specify k=2. Now, we will have 2 distance measures for each keypoint. We can find the ratio between the two values and match that particular keypoint to the keypoint in the second image having a higher ratio value. This is one of the methods for getting the best matches out of the image. The brute force matcher combined with the ratio test can give us very good results. The ratio value used is 0.75. If the ratio is greater than 0.75, only then it is considered as a good match and it is returned in the matched image.

Experimental Results:

Ferrari 1 Ferrari 2

Figure 50: Sift Features for Ferrari 1 and Ferrari 2 images

Ferrari 1 Ferrari 2

Figure 51: Surf Features for Ferrari 1 and Ferrari 2 images

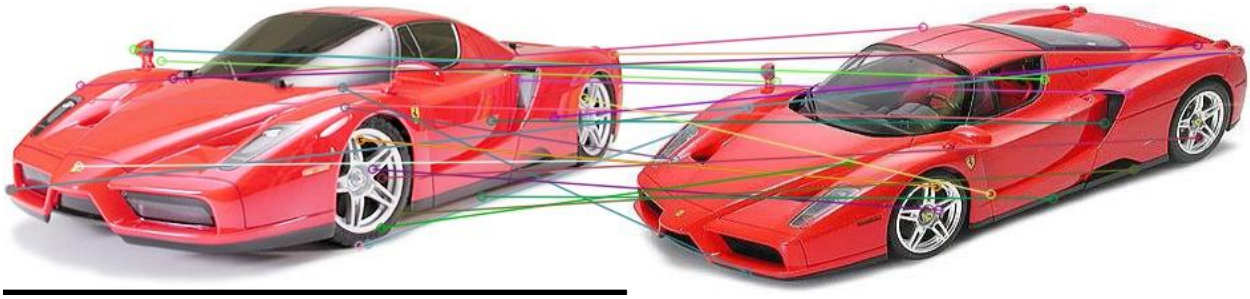


Figure 52: Images matched based on the SIFT features (Ratio: 0.75)

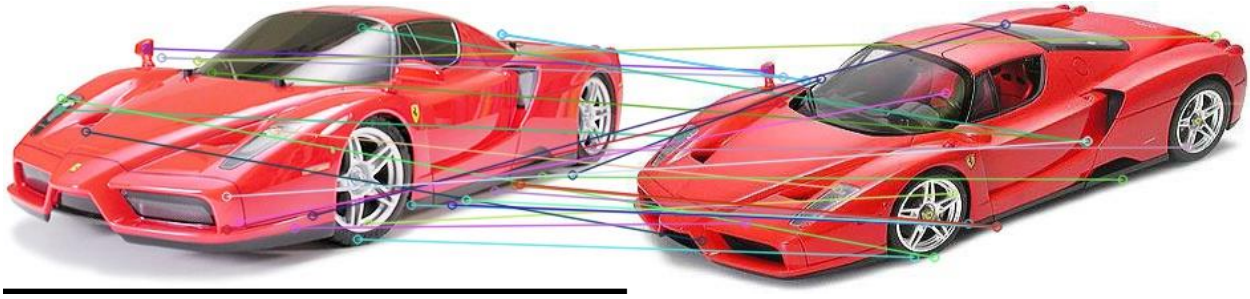


Figure 53: Images matched based on the SURF (Ratio: 0.75)

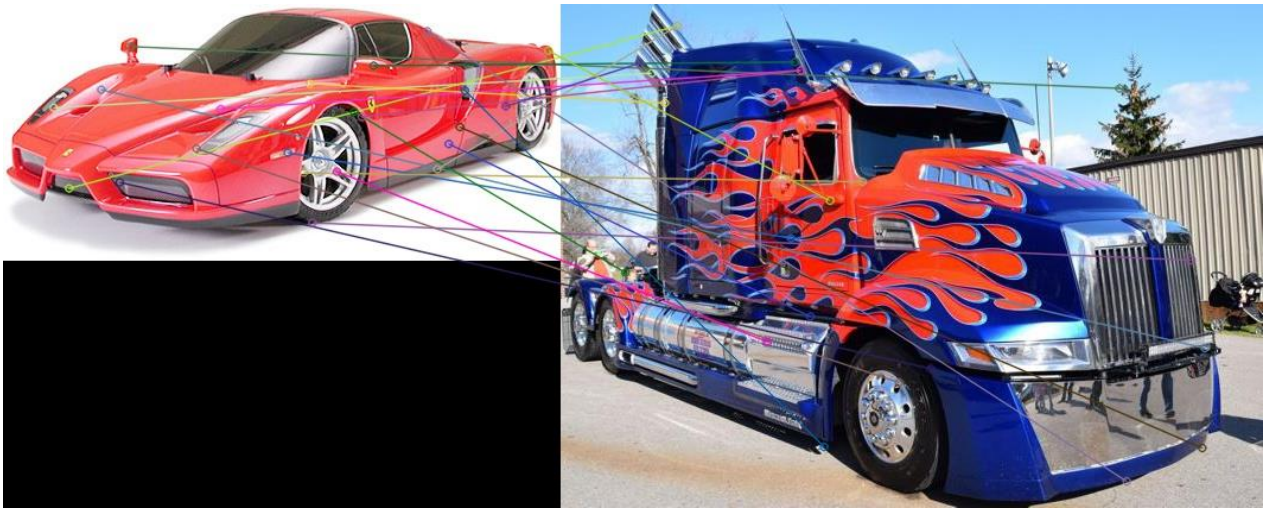


Figure 54: Image matching for Ferrari 1 and Optimus Prime using SIFT features (Ratio: 0.85)

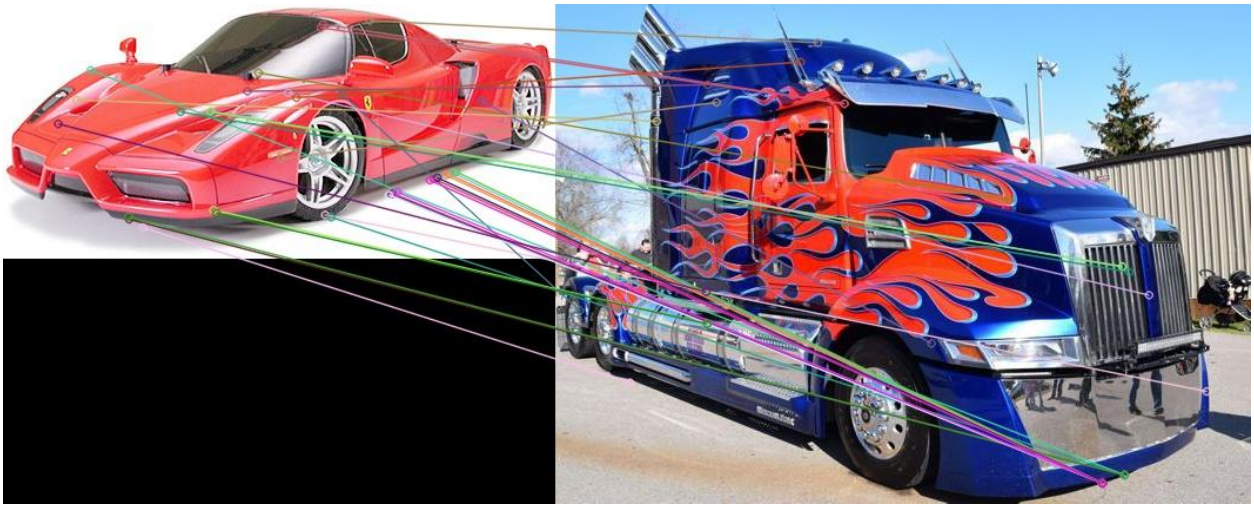


Figure 55: Image matching for Ferrari1 and Optimus Prime using SURF (Ratio: 0.8)

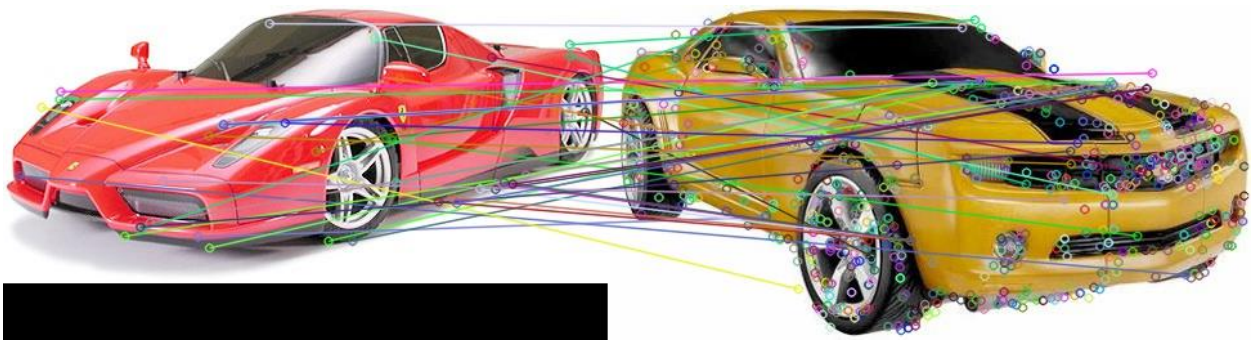


Figure 56: Image matching for Ferrari 1 and Optimus Prime using SIFT features (Ratio: 0.8)

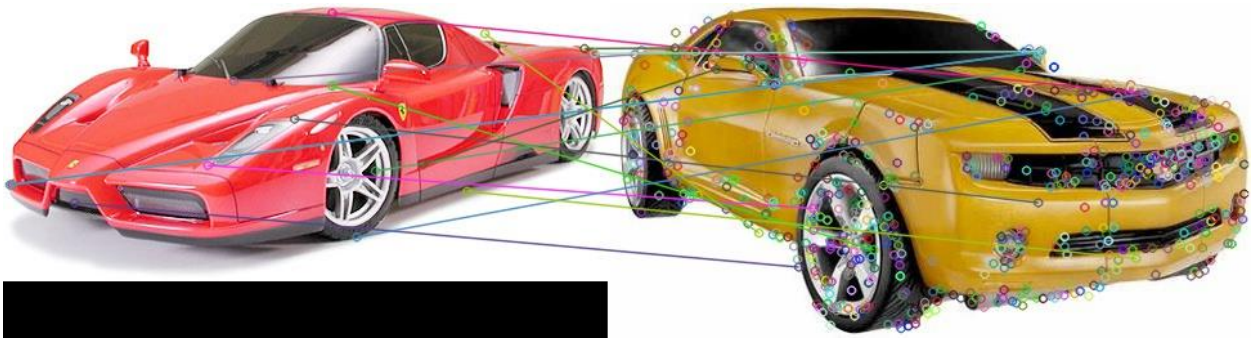


Figure 57: Image matching for Ferrari1 and Optimus Prime using SURF (Ratio: 0.8)

Discussion:

The SURF and SIFT features obtained for the Ferrari 1 and Ferrari 2 images are shown in figures 50 and 51 respectively. The images obtained by using the brute force matching method based on the SIFT and SURF features for ferrari1 and Ferrari 2 are given in figures 52 and 53 respectively. The threshold value for the ratio method was 0.75 in both the cases.

The matched image obtained by matching Ferrari 1 and Optimus truck based on the SIFT and SURF features are given in figures 54 and 55. The threshold ratio value used for SIFT was 0.85 and for SURF was 0.8. The matched image between Ferrari 1 and bumblebee car based on the SIFT and SURF features are given in figures 56 and 57. Both the methods have the same threshold, 0.8.

From the matched images obtained, we can see that the SURF features have better matching when compared to the SIFT features. We can also see that some of the features are not getting correctly mapped. This is due to the fact that the images are present in an angle. This is more evident when we consider the mapping between Ferrari and Bumblebee car. Both the cars are presented in a completely different angle. This makes it difficult to match the key points. Though this problem is present, more than half of the points

in the images are getting mapped correctly. This conveys the power of this image matching algorithm. When we consider the image matching between the Ferrari 1 and Optimus truck, most of keypoints are getting mapped correctly. This is due to the fact that they are automobiles.

In most of the cases, the keypoints get matched. As we can see from the results, the mapping between the Ferrari1 and Ferrari 2 are very good, and the matching between Ferrari 1 and Optimus truck is not very bad as we had expected. The matching between Ferrari 1 and bumblebee is not that good due to the angle difference between the two cars. This may also be due to the best keypoints chosen between the two images. The algorithm can be improved by choosing better key points as well. When we consider the mapping between Ferrari 1 and Optimus prime truck based on the SIFT and SURF features, we can see that the mapping based on the SURF features is very good when compared to the mapping obtained using the SIFT features. This can be evidenced from the other image matches as well. Thus, we can conclude that SURF features are better for image matching when compared to the SIFT features. To improve the image matching, we can use other matchers instead of Brute Force method. It may give better results.

(3.c) **Bag of Words**

Abstract and Motivation:

Bag of Words is an important concept for object detection techniques. It represents the use of clustering techniques to identify the specific patterns in an image. Bag of words for image processing contains specific patterns in the image as words. By applying the Bag of Words to a new image can give the similar words (image patterns) present in that specific image in similarity to the images using which the codebook was obtained. The objective of this problem is to apply bag of words technique and get the words to form a codebook based on the clusters of keypoints from Ferrari 1, bumblebee car and Optimus prime truck. We have to use the obtained codebook and find the similar words present in Ferrari 2 by obtaining the word histograms with the Ferrari 1, bumblebee car and Optimus prime truck and find the similarities using the histograms obtained.

Approach and Procedure:

The first step to obtain the bag of words codebook is to extract the SIFT features from the training images. In this case, the training images are Ferrari 1, Optimus Prime truck and Bumblebee car. We have to obtain the SIFT features from these images. The dimension of each of the keypoint is 128. So, we will have different number of keypoints from each of these images. For comparison purposes, we can pick a common number of keypoints from each of the images.

After the keypoints are chosen, we have to implement the k-means clustering algorithm to cluster each of the keypoint into one cluster. We have been asked to create a codebook with 8-bins, which basically means that we need to have 8 cluster centroids for each of the bins. We can specify the number of clusters to

be 8 for the k-means clustering algorithm. We have to train the k-means model and get the respective cluster centroids. After that, based on those cluster centroids, we can predict the cluster labels for each of the data point for the Ferrari 1, Optimus prime truck and the bumblebee car images. Once this is done, the training part is done.

For the testing part, where we want to compare the histograms of the Ferrari 2 image with the other images, same number of keypoints extracted from each of the training images has to be extracted from the Ferrari 2 image as well. Once the key points are extracted, they have to be clustered into 8 clusters with the cluster centroid obtained from the k-means model previously evaluated. Once the data points are clustered into the 8 clusters, we can obtain the histogram of the number of keypoints present in each of the clusters and using this, we can compare different images and understand whether they are related or not. This is the algorithm for bag of words.

Experimental Results:

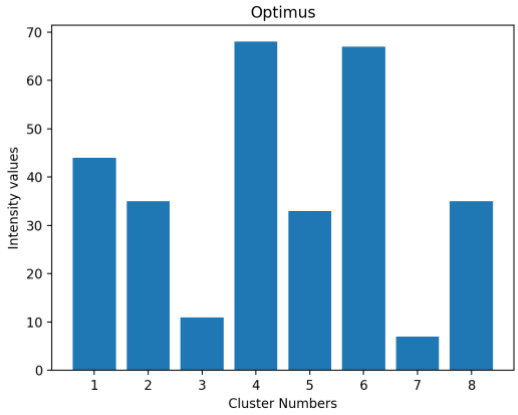


Figure 58: Bag of Words Histogram for Optimus Prime truck

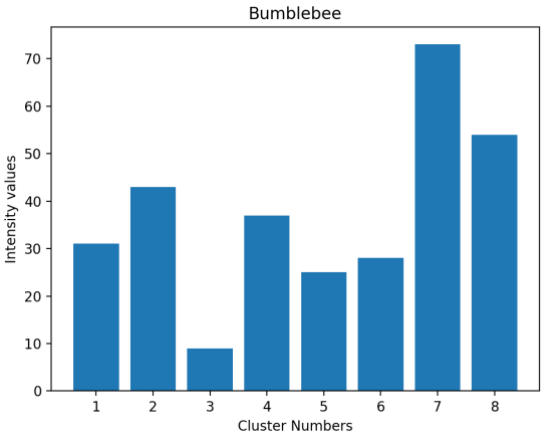


Figure 59: Bag of Words Histogram for Bumblebee

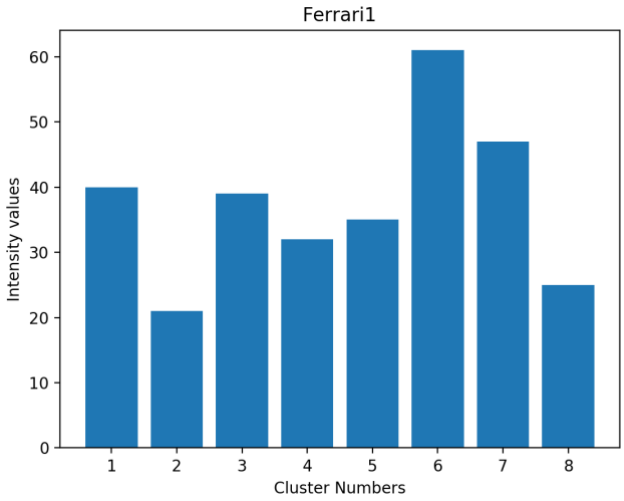


Figure 60: Bag of Words Histogram for Ferrari1

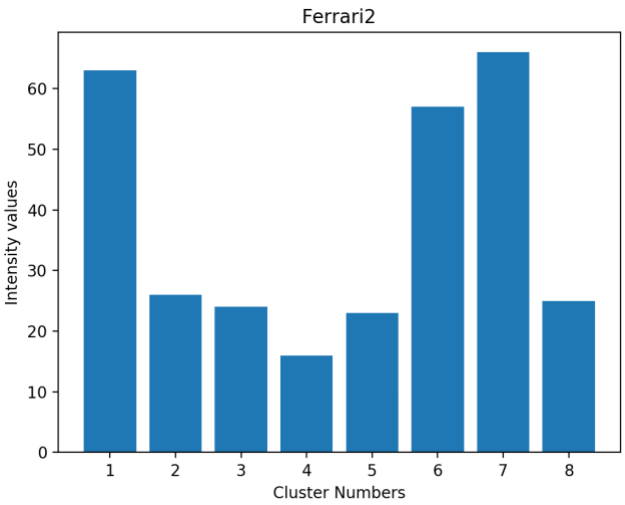


Figure 61: Bag of Words Histogram for Ferrari2

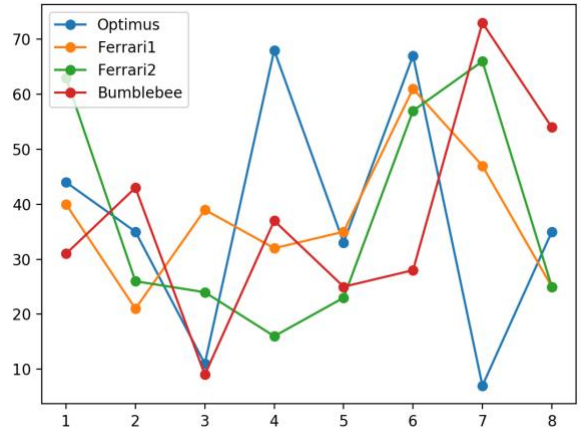


Figure 62: Bag of words plot for all four images

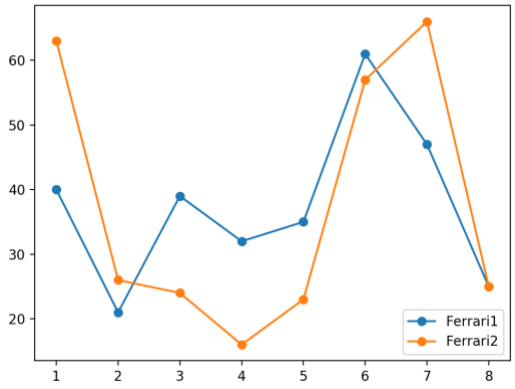


Figure 63: Bag of words plot for all ferrari1 vs ferrari2

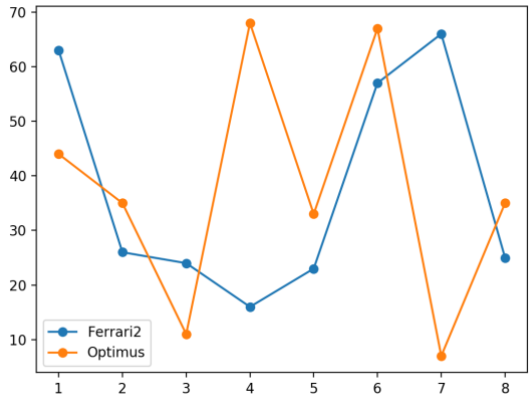


Figure 64: Bag of words plot for all Optimus vs ferrari2

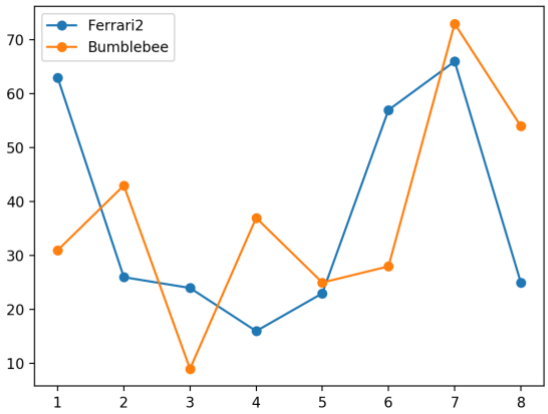


Figure 65: Bag of words plot for all Bumblebee vs ferrari2

Discussion:

The bag of words histogram for the Optimus prime, bumblebee, Ferrari 1 and Ferrari 2 are given in images 58, 59, 60 and 61 respectively. The centroids for these clusters were obtained by training the images Ferrari 1, Optimus prime and Bumblebee truck. To compare the bag of words of Ferrari 2 with the other images, the comparison plots are given in figures 62, 63, 64 and 65. The comparison plot given in figure 62 contains the histogram values of all the four images in single plot. The comparison plot given in figure 63 contains the plots of Ferrari 1 and Ferrari 2. In figure 64, the comparison plot is between Optimus truck and Ferrari 2. In figure 65, the comparison plot is between Bumblebee and Ferrari 2.

From these plots, we can find the similarities in the bag of words. For example, if we consider the bag of words histogram for Ferrari 1 and Ferrari 2, we can find that they are almost similar. We can say that the clusters are created having similar keypoints in them. This is the idea of Bag of Words. Just by comparing the histogram values, we can say if the images are represented by these histograms are similar or not. We can also find similarities between the bumblebee car, Ferrari 1 and Ferrari 2 as they come in the same category of cars. If we compare these histograms with the Optimus prime truck, we can see that they are completely different, conveying that they belong to 2 categories namely cars and trucks.

To find the difference between these histograms apart from a visual understanding, we can also compare the number of data points in each cluster and the difference in each of them can be obtained to find the similarity between the original images. In this case, the difference in clusters between Ferrari 2 and the other images are given below:

Ferrari 2 and Ferrari 1:	54
Ferrari 2 and Bumblebee:	64
Ferrari 2 and Optimus:	83

This is just a difference in the l1 norm values between the histogram bins in each image. From these plots and values, we can say that the images Ferrari 2 is very similar to Ferrari 1 and somewhat closer to bumblebee as these three images come under the category of cars. When compared with that of the Optimus prime truck, the plot of Ferrari 2 is very difference conveying the difference between the categories of cars and trucks.