Name: Shreya Kate
USC ID: 2334973997
Email: shreyak@usc.edu

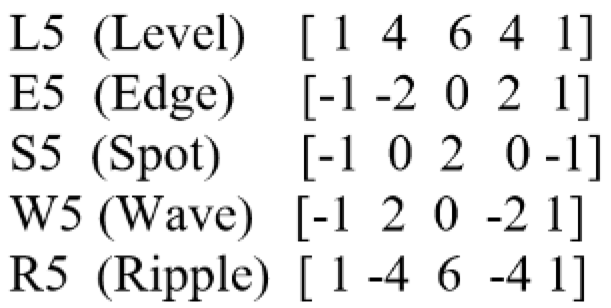
Homework 4

Problem 1:

(a)

The five 5x5 Laws' filters are used for extracting features from the 12 test images. I got 25 features, since the tensor products of the Laws' filters gave 25 filters and I applied these filters to the images.

Laws' filters are as follows:



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]

I extended the boundary of the images by 2 pixels on all sides of the image.

For extracting the features, first, image mean was calculated and was subtracted from every pixel of the image. This is done t reduce the illumination effects.

Next, the 25 Law's filters are applied to the images and I got 25 filtered images. Then I calculated the energy of each image and this formed the 25D feature vector, with each feature representing the energy of one filtered image. I reduced the dimension of this from 25D to 15D by averaging the values got with pairs like L*E' and E*L'.

Next, I calculated the discriminant power of each feature. I calculated the variance of each feature to see which is the strongest and weakest feature.

L*L which has a zero mean is used to do normalization. I divided all the other features with L*L feature and got a 14D feature vector.

I then used PCA to reduce the dimensions to 3D and plotted the 3D feature vector in the feature space.

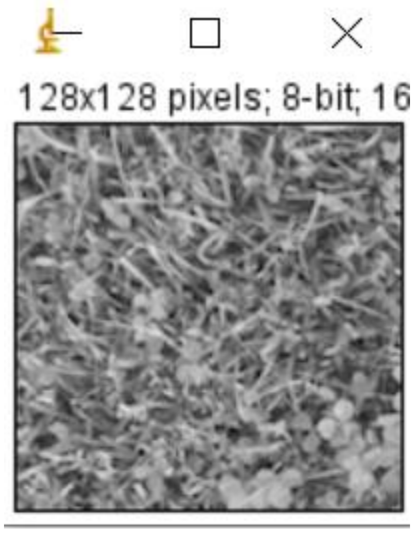
Results:

To find Discriminant Power:

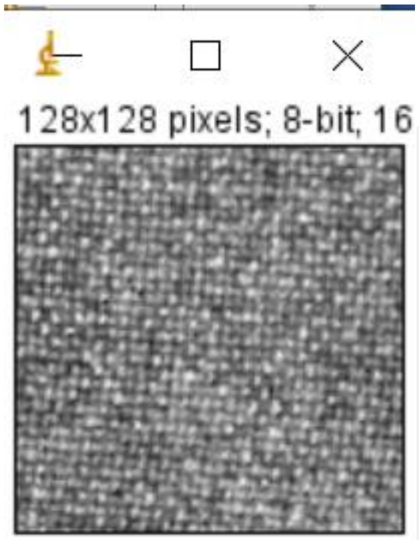
Feature and corresponding filter used	**Variance**
Feature1 : L*L	8.22667013901142e+15
Feature2 : L*E	74200.5933317295
Feature3 : L*S	34898.7592793979
Feature4 : L*W	51416.2493417173
Feature5 : L*R	346715.328669236
Feature6 : E*L	137435.956468245
Feature7 : E*E	19454.5905710664
Feature8 : E*S	13957.7299934751
Feature9 : E*W	13647.6342232932
Feature10 : E*R	21580.3915980113
Feature11 : S*L	67326.7756974729
Feature12 : S*E	12717.5919611039
Feature13 : S*S	10151.6568289526
Feature14 : S*W	9907.58315861316
Feature15 : S*R	15071.2532378221
Feature16 : W*L	64762.2884702913
Feature17 : W*E	9518.92687092171
Feature18 : W*S	7310.79108751325
Feature19 : W*W	7208.63828821488
Feature20 : W*R	12949.8465873205
Feature21 : R*L	467625.531679611
Feature22 : R*E	13863.6919997493
Feature23 : R*S	8549.06961986265
Feature24 : R*W	10287.5123998377
Feature25 : R*R	56908.1721156503

Feature 1 with filter L*L has the highest variance and hence it has the maximum intra-variability. So, it has the weakest discriminant power.
Feature19 with filter W*W has the least variance so it has the least intra-variability and therefore, its discriminant power is the strongest.

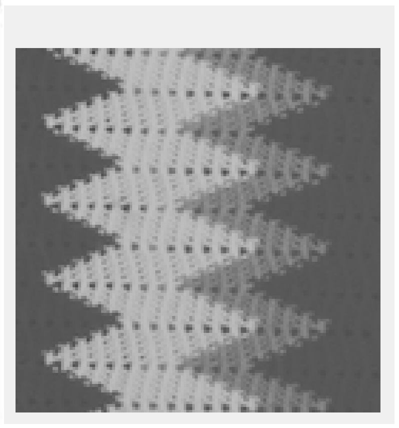
The feature extraction is done on the 12 test images given. The 12 images given are shown below:



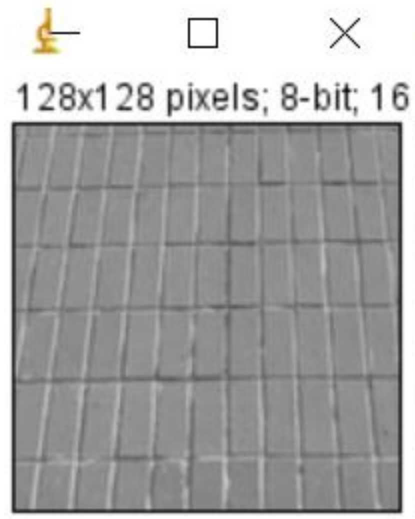
1.raw



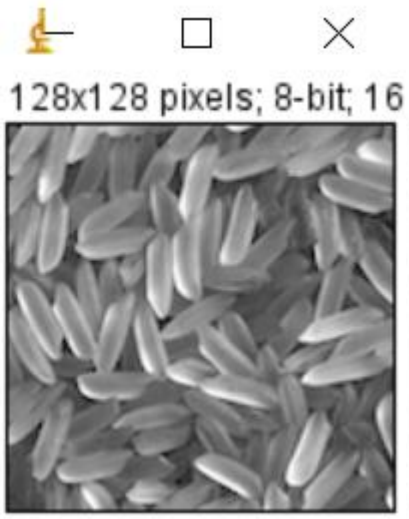
2.raw



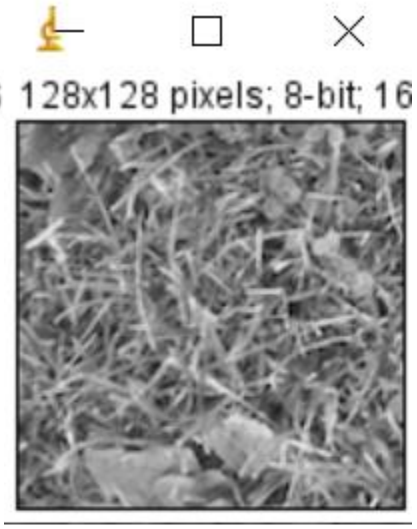
3.raw



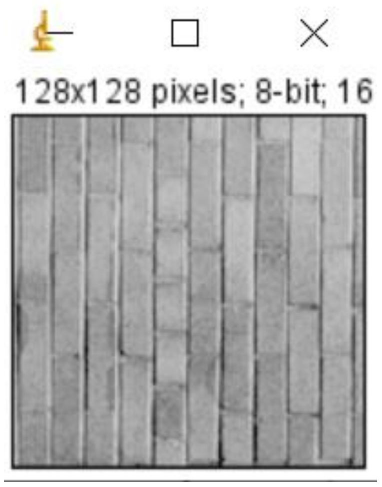
4.raw



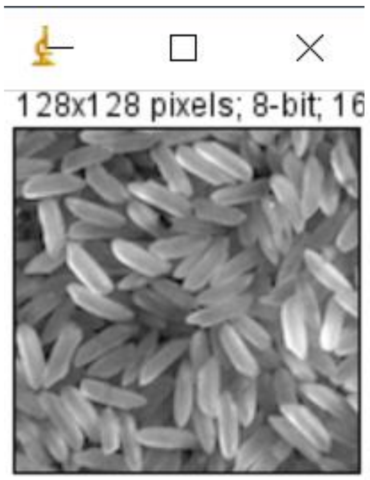
5.raw



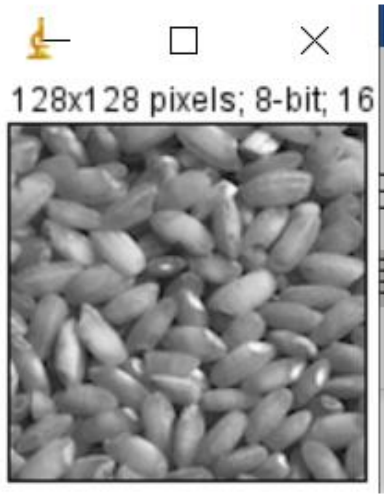
6.raw



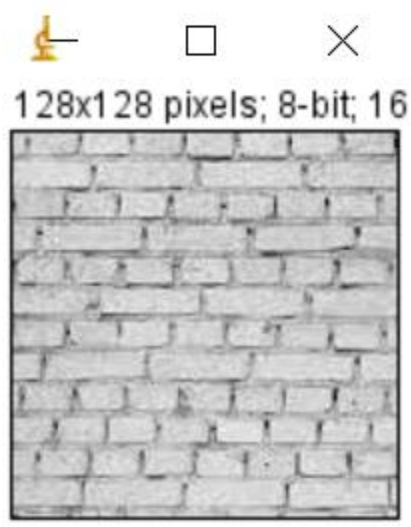
7.raw



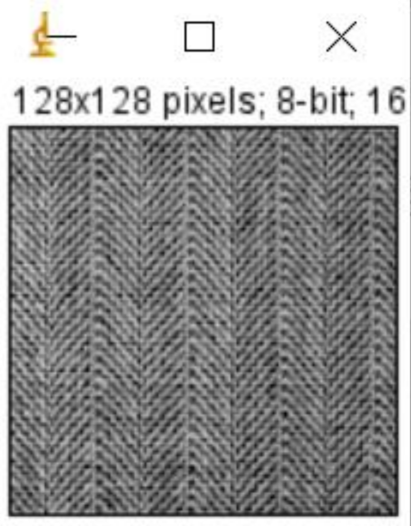
8.raw



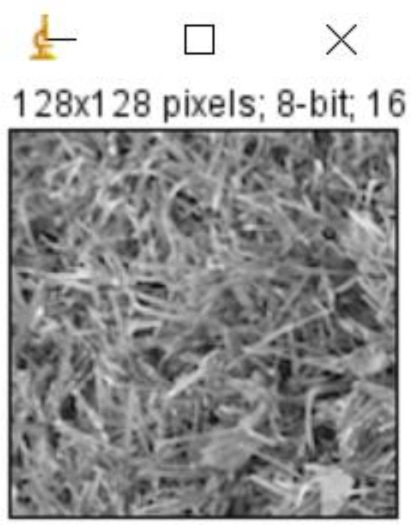
9.raw



10.raw



11.raw

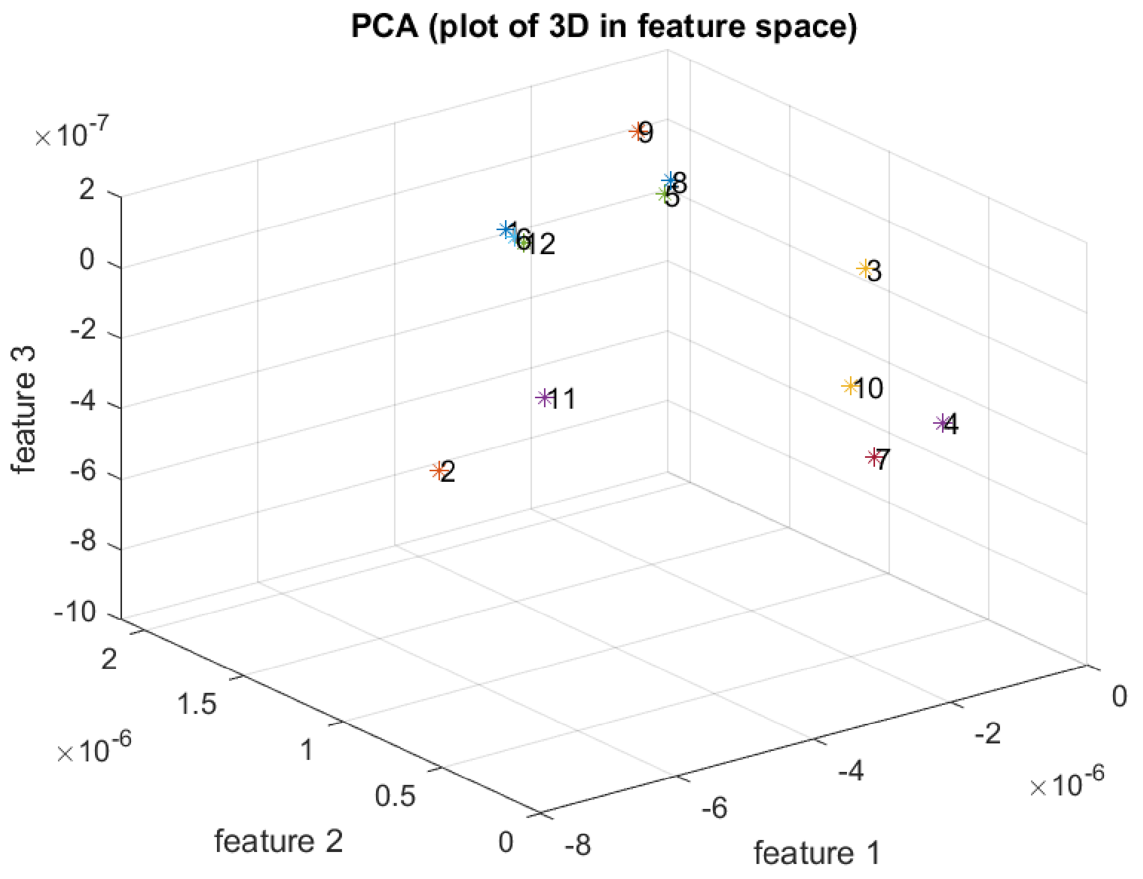


12.raw

As can be seen from the images above (visual inspection) , the correct labels for 1.raw – 12.raw are as follows:

Class	**images**
1 (grass)	1.raw , 6.raw, 12.raw
2 (bricks)	4.raw, 7.raw 10.raw
3 (blanket)	2.raw, 3.raw, 11.raw
4 (rice)	5.raw, 8.raw, 9.raw

The plot of PCA is shown below:



As can be seen from the figure, the clusters of 1,6,12 are close by (which is grass). 5,8,9 are also close to each other. They are the rice class.
4,7,10 are the brick class which are close to each other.
11,2,3 are the blanket class. 11 and 2 are close by but 3 is closer to brick class so it is an error of PCA. This could be because of the kind of design on the blanket in 3.raw.

I have implemented PCA on my own and have not used the built-in function.

Problem 1:
(b)
In this**, I implemented the kmeans algorithm for unsupervised learning on my own. I did not use the built-in function.**

There are 4 classes in our problem so I initialized 4 centroids by randomly selecting from the data points. Then I calculated the Euclidean distance between samples and the centroids.

$$d(E_i, C_k) = \sqrt[2]{\sum_{j=1}^{m} \left(E_{i,j} - C_{k,j}\right)^2}\, , where\ i = 1, \cdots, n\ and\ k = 1, \cdots, K$$

This gives how close to the centroids which samples are.
Next I checked which centroid was closest to the samples and assigned the label to the sample depending on which centroid it was closest to.
Thus, the samples were classified. Then I calculated the new centroids of these classes by averaging the points in each class. After I got the new centroids I repeated the process again till the difference between centroids in the previous run and the centroids in the present run were almost the same.

I got the following classification for 15D and 3D. The first column represents the image number (1.raw,2.raw…) The second column represents the classification.

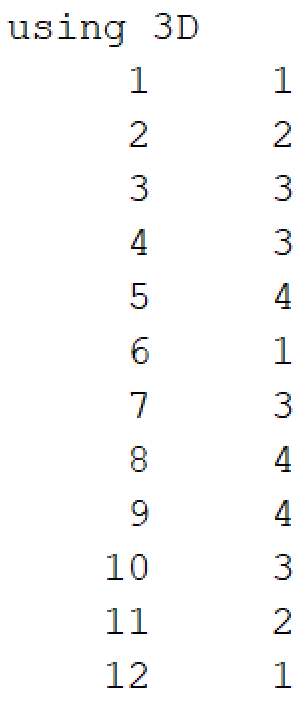
```
using 15D
     1     1
     2     1
     3     4
     4     4
     5     2
     6     1
     7     4
     8     2
     9     2
    10     4
    11     3
    12     1
```

The classification by kmeans for 15D is as follows:

Class	Images
1 (grass)	**1, 2, 6, 12**

2 (rice)	**5,8,9**
3 (blanket)	**11**
4 (bricks)	**3,4,7,10**

The error rate is 16.67% since 2 out of the 12 are misclassified.



```
using 3D
     1     1
     2     2
     3     3
     4     3
     5     4
     6     1
     7     3
     8     4
     9     4
    10     3
    11     2
    12     1
```

The classification by kmeans for 15D is as follows:

Class	**Images**
1 (grass)	**1, 6, 12**
2 (blanket)	**2,11**
3 (bricks)	**3,4,7,10**
4 (rice)	**5,8,9**

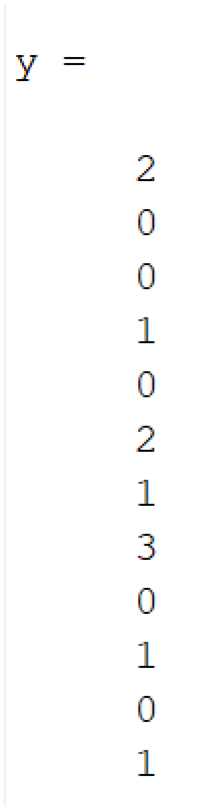
The error rate is 8.33% since only 1 out of the 12 are misclassified.

Advantage of feature dimension reduction:

The kmeans classifier works better with lesser number of features. Therefore, the error rate for 3D is less than the error rate for 15D. This is because when the dimension is high, all the data points are quite far away from one another. So, the probability of them being misclassified increases(because the centroids are also far away). In dimension reduced feature space, the data points are closer to one another and it gets easier to classify them as the proximity increases. The computational cost in fewer dimensions is also less as the calculations of L2 norm (Euclidean distance) reduces.

For supervised learning RF and SVM were implemented by using the built-in functions.
Since we have 4 classes of textures in our problem, I had to use fitcecoc instead of fitcsvm since fitcsvm only gives results for 2 class system. In RF, the function gave multi-class results.

Results:
RF

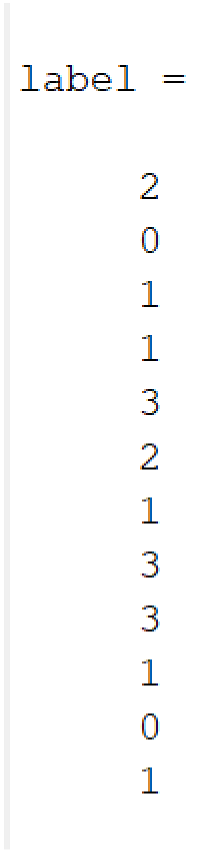


```
y =

     2
     0
     0
     1
     0
     2
     1
     3
     0
     1
     0
     1
```

The error rate for RF came out to be 16.67 percent since 2 points were misclassified.

SVM



```
label =

     2
     0
     1
     1
     3
     2
     1
     3
     3
     1
     0
     1
```

The error rate for SVM came out to be 8.33 percent since 1 point was misclassified.

I was getting varied results by changing some parameters, and these two were the best results I got out of everything I tried.

By looking at the results I got, SVM is better/ more accurate than RF.

RF goes through several decision trees (I chose 500 trees as a parameter) and then classifies the textures based on the result of the trees.

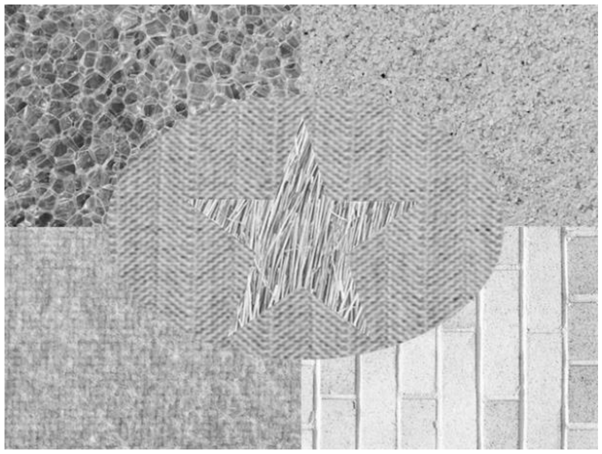
SVM has support vectors. These support vectors maximize the margins between different classes and thus improve accuracy by separating classes more efficiently and effectively. The errors are reduced due to this margin.

Thus, by the results I got, SVM gave a better performance than RF.

Problem 1:
(c)
For texture segmentation, I followed the following steps.
1. I applied the Laws' filters to get 15 grayscale images. The 25 Laws' filters give 25 images. I reduced them to 15 by taking the average of the images formed by using filters like E*L' and L*E'. This is because the images obtained by these 2 filters are almost the same. So I could reduce the 25 images to 15, without losing useful information.
2. I computed the energy by using the window approach. I got energy for each pixel value and hence got 15D energy feature vector for each pixel.
3. I normalized the feature vectors by using the first feature which was obtained by the filter L*L'. So, I got a 14D energy feature vector for each pixel.
4. Finally, I used the kmeans algorithm to do the segmentation. I used the 6 classes for 6 textures and gave them the following intensity values. So, based on which class/texture the pixel was classified as. It got the corresponding intensity value and got shaded in that color.

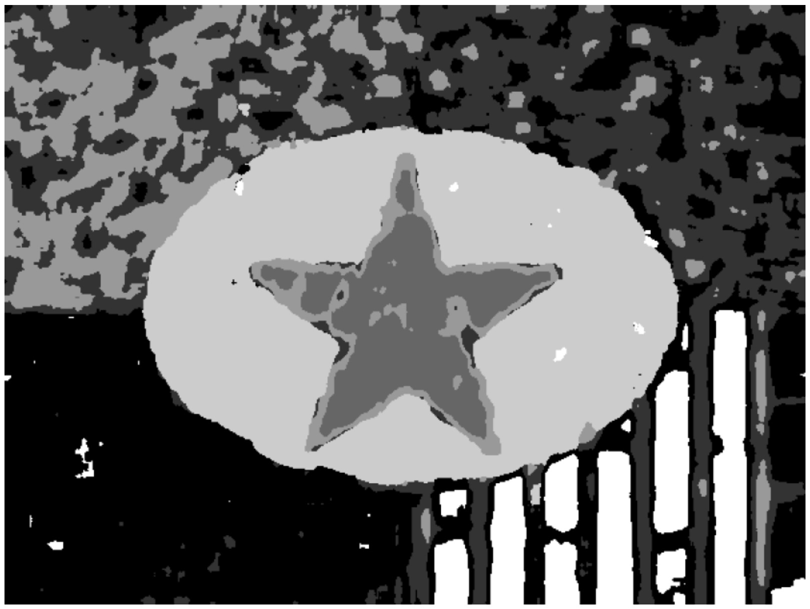
I changed the window size while computing the energy feature vector and tried out multiple window sizes to get results.

Results:
The original image is shown here:

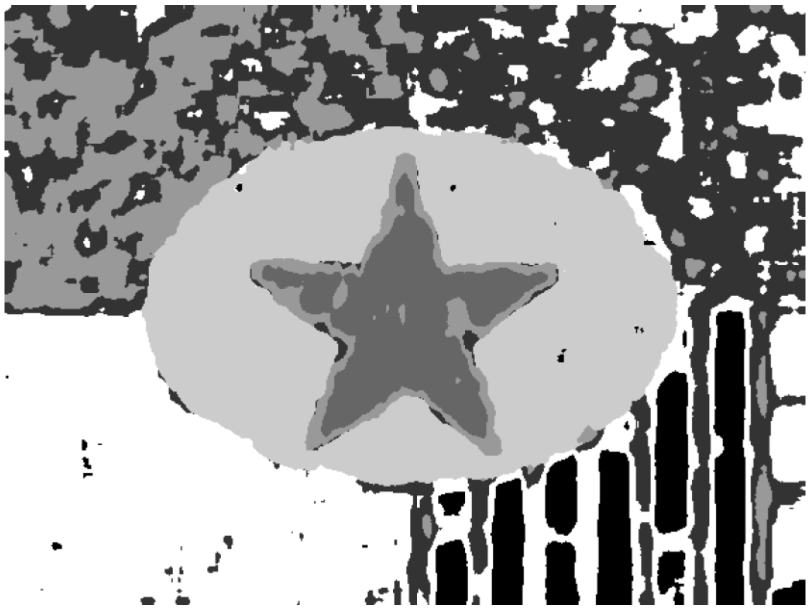


K means

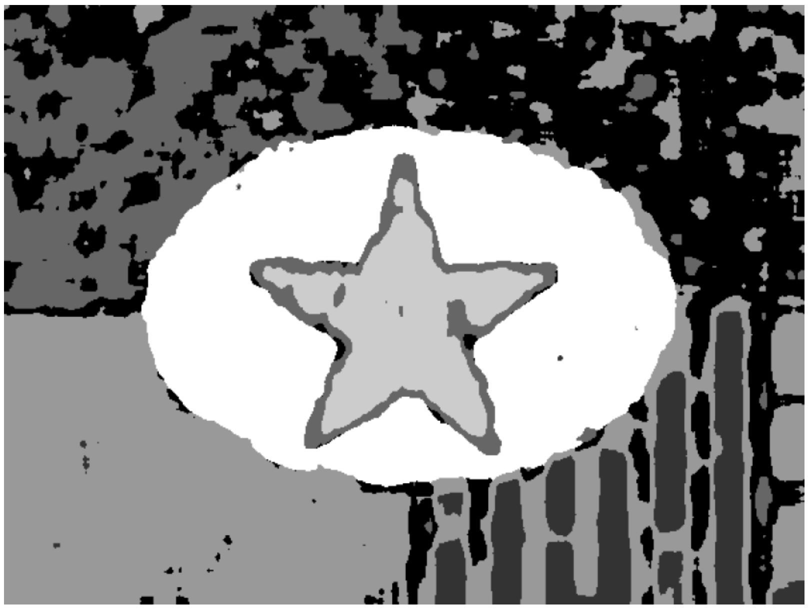
Window size = 15



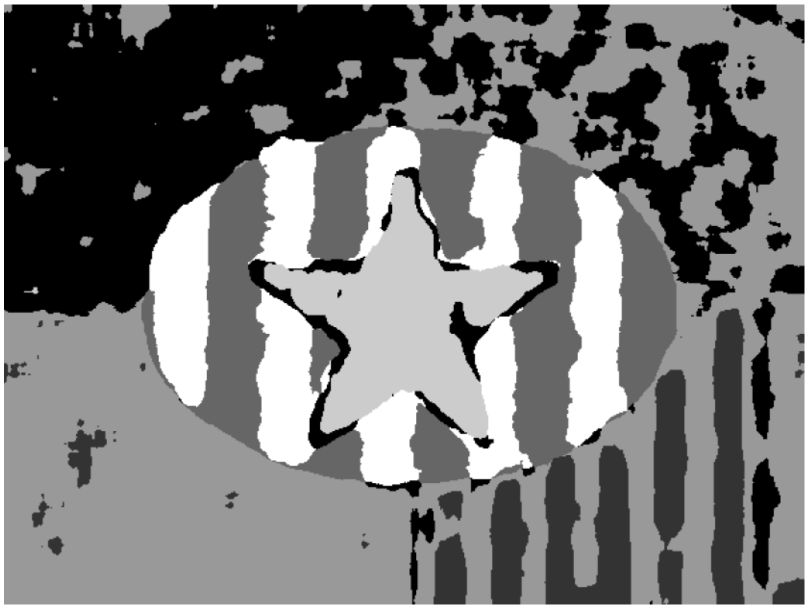
Window size = 17



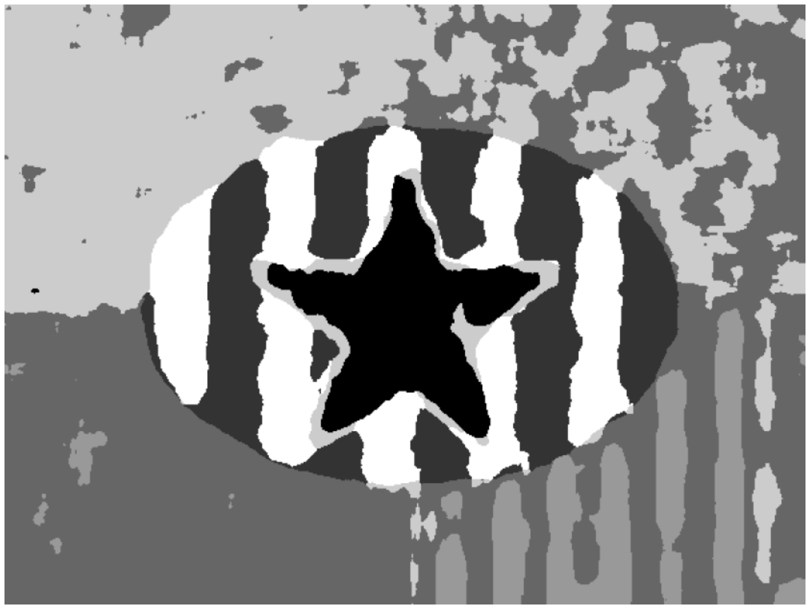
Window size = 19



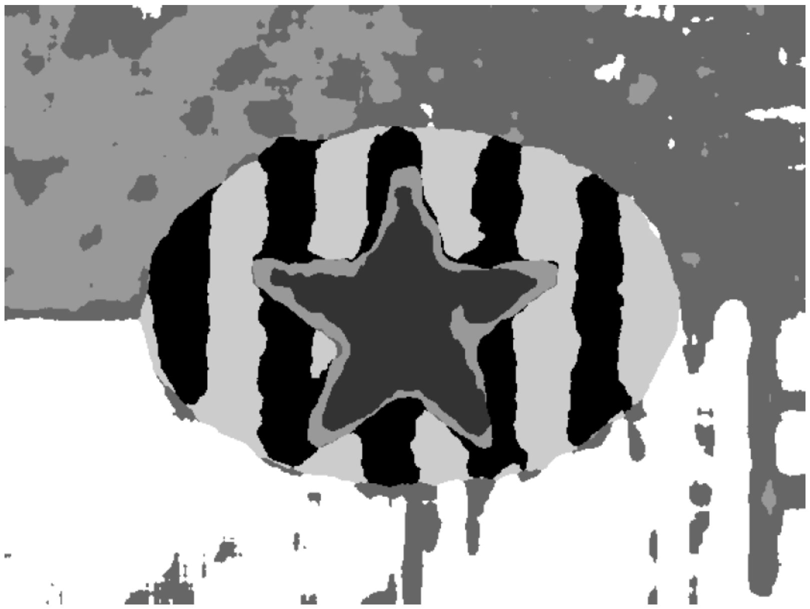
Window size = 21



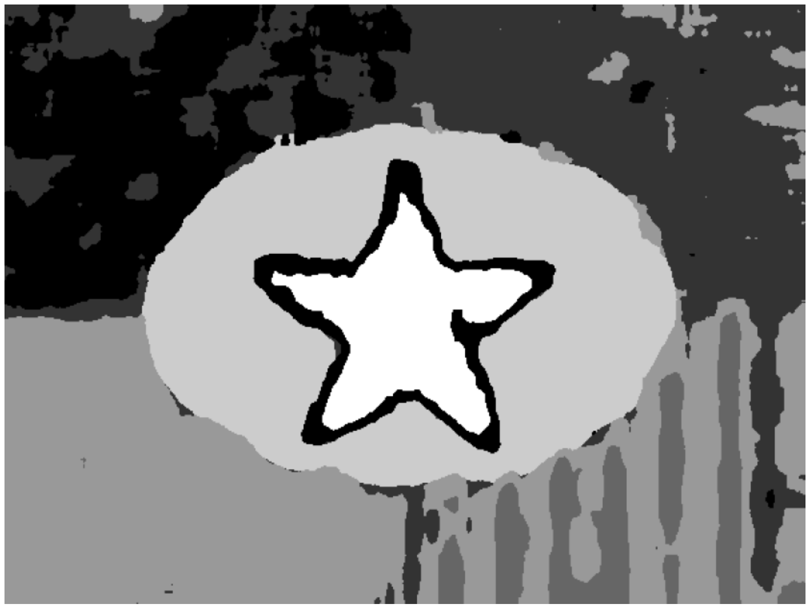
Window size = 23



Window size = 25



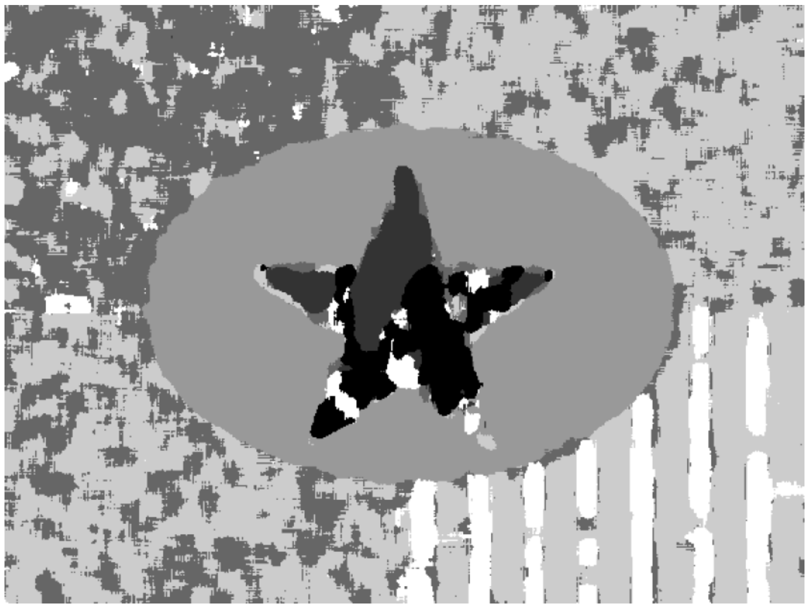
Window size = 27



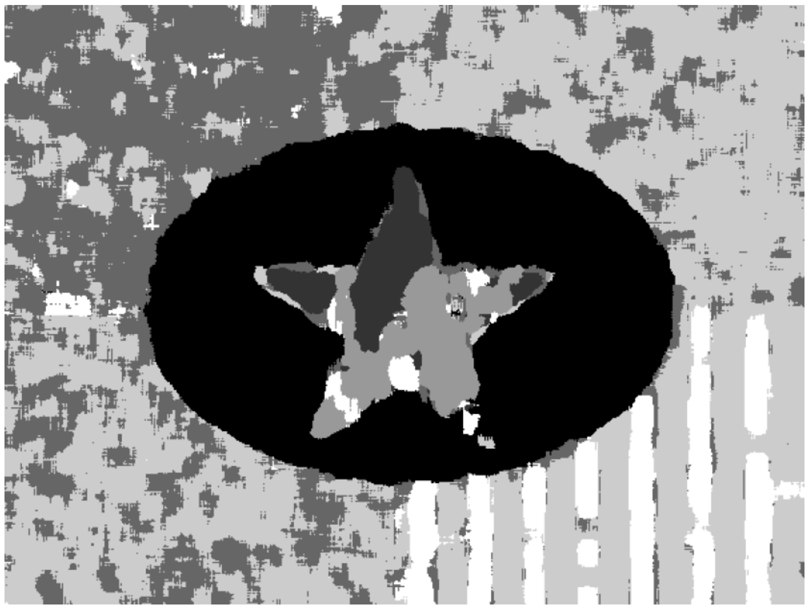
With increase in window size the boundaries of the textures can be seen more distinctly. There are a few holes and discrepancies in the segmentation, but looking at the whole picture, it gives a good idea of where different textures are located.
The bricks in some of the output are identified as different textures and I can see stripes, this is because alternate strips of the brick texture are different from one another. Similarly even for the centre oval cloth texture, the alternate patterns are slightly different which leads to them being identified as different textures in some cases.

Problem 1:
(d)
I did the same procedure for feature extraction as in Problem 1: I. But, after that, instead of kmeans I am using PCA to get the feature reduction. I reduced the 14 features to 7 features. I chose the first 7 features and got the following results.

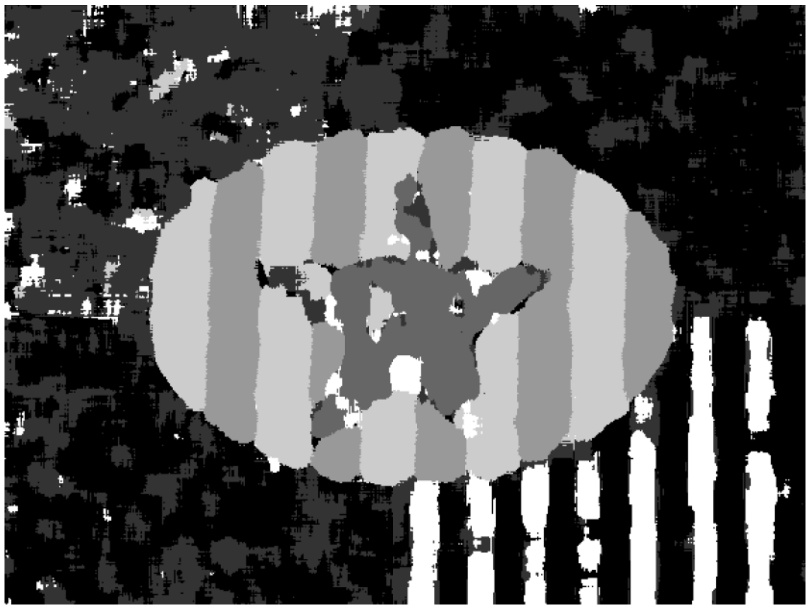
Window size = 15



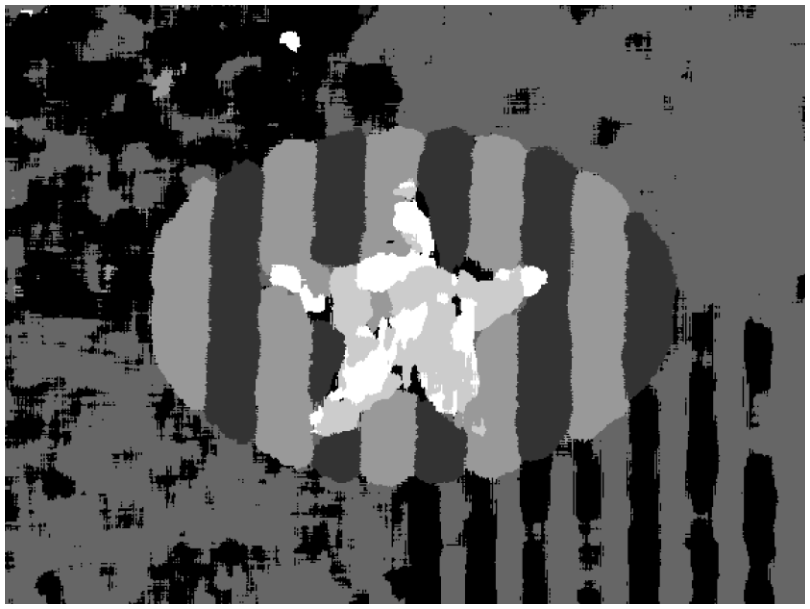
Window size = 17



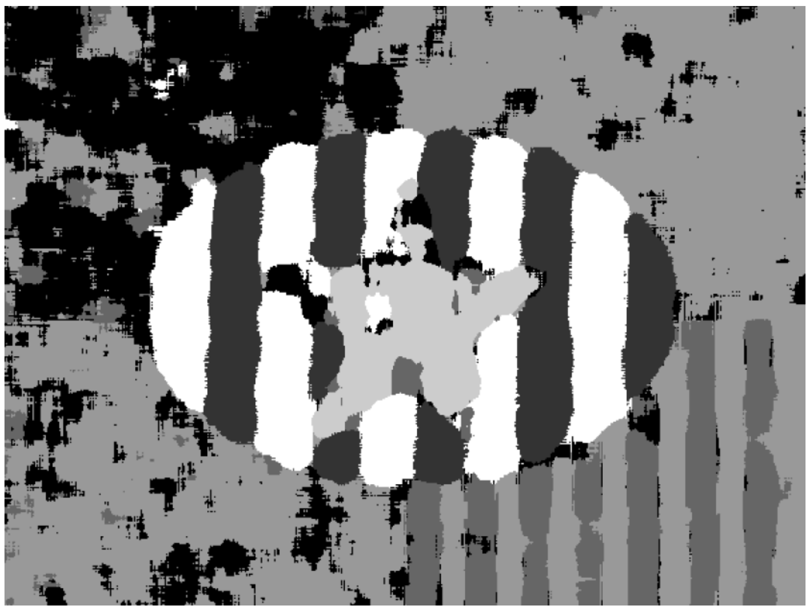
Window size = 19



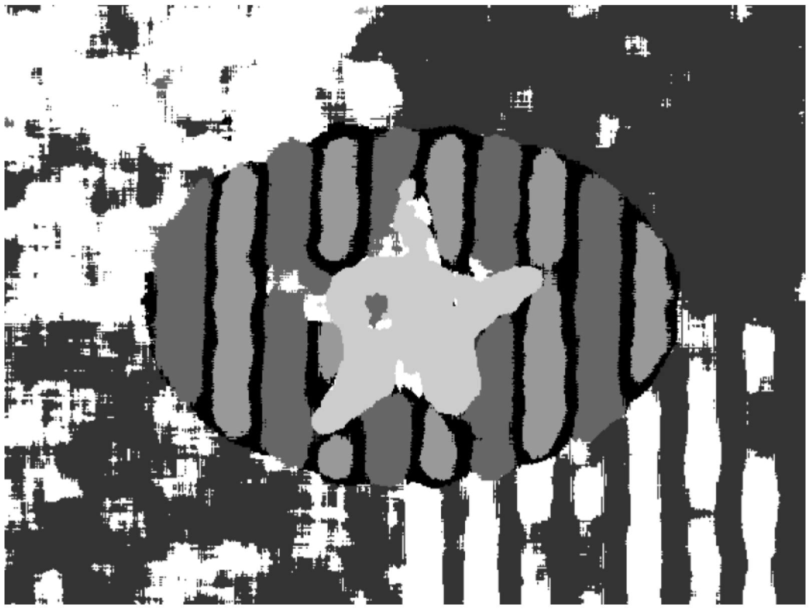
Window size = 21



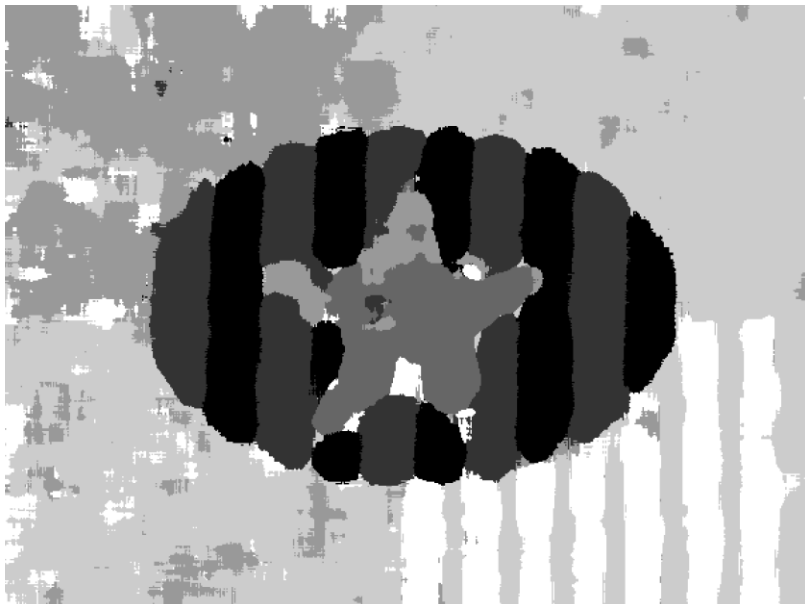
Window size = 23



Window size = 25



Window size = 27



PCA improves the result. It is computationally less expensive. It has only 7 features to do the segmentation, so the results are better. The features that you choose are important as different feature selection may give better results than others.

Problem 2:
(a)
We use SIFT (Scale Invariant Feature Transform) to extract discriminant functions/features from the images. It is a mathematical tool/model.

1. SIFT is robust to the following geometric modifications:
Rotation, scaling and translation. It is also partially invariant to affine 3D transformation and some illumination effects.

2. Rotation
Robustness to rotation is achieved by Orientation Assignment. The central derivatives and gradient magnitude and direction of L are calculated. This means

that we smooth the image at the scale of the key points. We do this by using the following formulas of magnitude and orientation.

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = tan^{-1}((L(x+1,y) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

The orientations calculated are quantized into 36 bins and thus a histogram is formed. From the histogram we get the orientation of each image patch. The orientation of image patch is the peak of the histogram.
We choose 16x16 neighborhoods around the key points and calculate the relative magnitude and orientation. Thus we have all the orientations match to a particular patch and rotation does not matter.

Scaling
Scale-space extrema detection takes care of scaling effects not being an issue. The goal of scale-space extrema detection is to find potential locations for feature extraction. Local extrema in scale-space of an image are called are called interest points and the sigma value used in the gaussian function is called the scale. We use Laplacian of Gaussian (second derivative filter) in this. It acts as the blob detector.
In this, we first smooth the image, and find zero-crossings that is the Laplacian filter. Thus, we get the Laplacian of gaussian filtered image.

– $O(x,y) = \nabla^2(I(x,y) * G(x,y))$

Just another linear filter.

$$\underbrace{\nabla^2(f(x,y) \otimes G(x,y))}_{\boxed{\text{Laplacian of Gaussian-filtered image}}} = \underbrace{\nabla^2 G(x,y) \otimes f(x,y)}_{\boxed{\text{Laplacian of Gaussian (LoG) -filtered image}}}$$

We repeat the procedure for several octaves. 1 octave corresponds to taking one scale value, i.e., 1 sigma value and then finding the extrema. The next octave will mean taking a higher sigma value and so on. Thus, many scales are covered in the process. And the algorithm becomes scale invariant.

We use the difference of gaussian instead of Laplacian of gaussian to improve the speed. The rest of the procedure remains the same.

$$\text{Difference-of-Gaussian: } D(x,y,\sigma) = L(x,y,k\sigma) - L(x,y,\sigma)$$

Translation
The translation effects become invariant because the SIFT scans through the entire second image to see if it finds a matching feature in the second image as the first image. So, the feature could be translated anywhere in the second image and does not need to be at the same location as the first image for it to be matched.

Affine and 3D projections, illumination
SIFT blurs image gradient locations, so it gets invariant to Affine and 3D projections. The Laplacian of gaussian is used to detect the key points and the gaussian is used to smooth the image.

3. Illumination change
Illumination change will have a large effect on gradient magnitude since it is related to the pixel values, but it has a small effect n gradient direction. The illumination effect is controlled by using thresholding where the gradient magnitudes are thresholded to be 0.1 times the maximum gradient value. This reduces the illumination effects.

4. SIFT uses DoG instead of LoG to improve the speed of the algorithm.

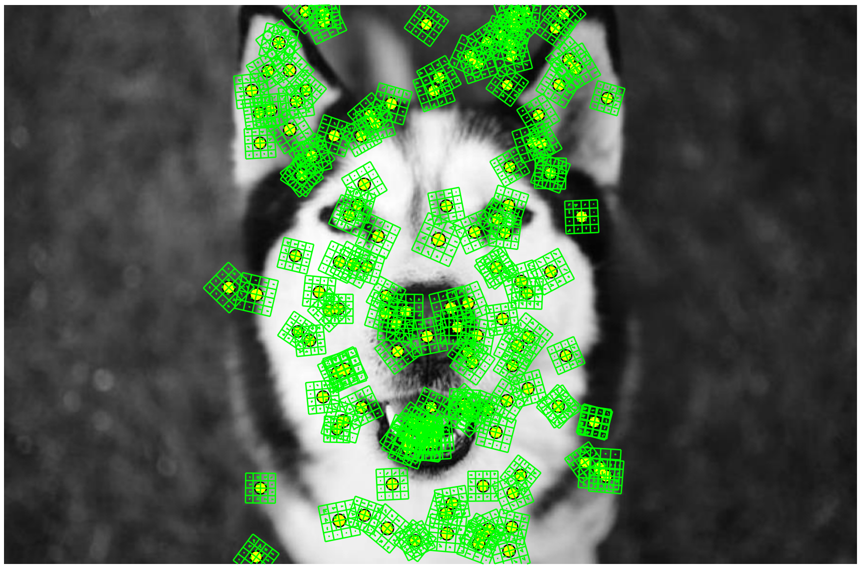
5. The output vector size is 160 in SIFT’s original paper.
The keypoint descriptor has 16x16 neighborhoods around each key point. This means it has 16 4x4 sublocks.

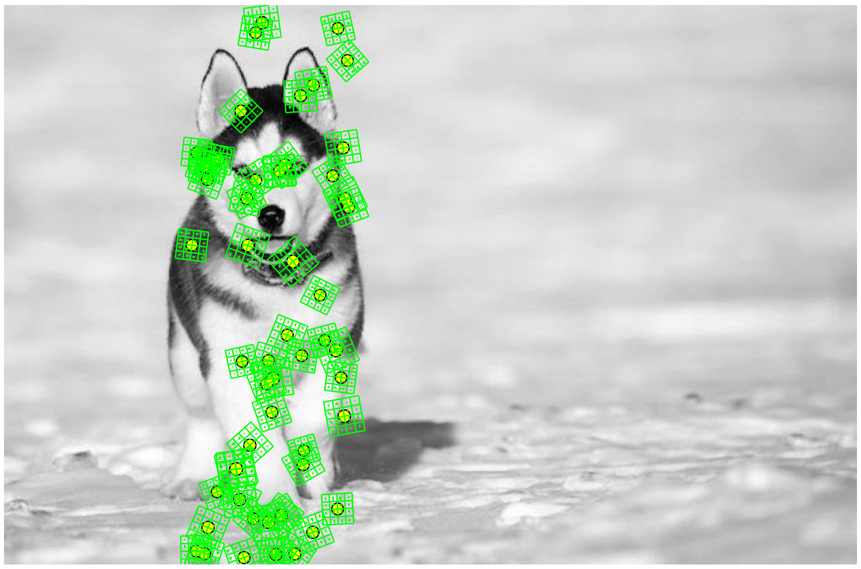
Every subblock will have 8 bins (histogram of gradient orientation). Each bin has 16 sample. So we have 128 samples. To sample at a latger scale, another octave is added which has 2x2 region. So 8*2*2= 32 is added to 128 which makes 160 samples in the output vector.

Problem 2:

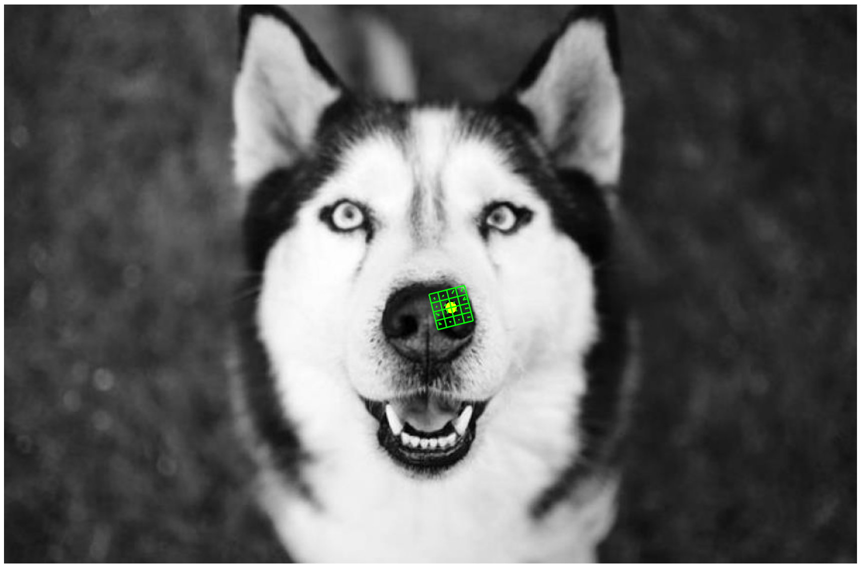
(b)

1. The key points of the Husky_3 and Husky_1 can be seen in the 2 images below. I got the key points of the 2 dogs on the faces of the dogs so that I will get better matching results.

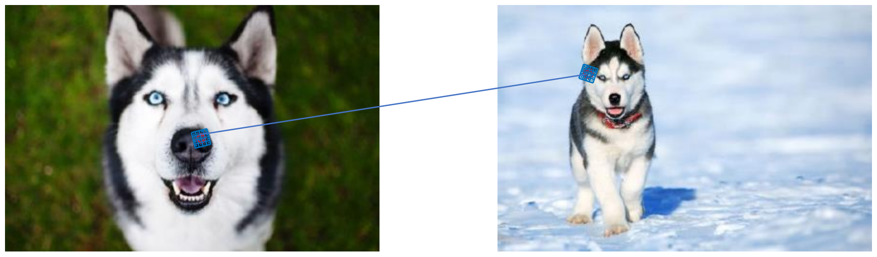




The largest scale key point of Husky_3 can be seen in the figure below. I extracted the largest scale feature and displayed it.



Next, I had to find the feature in Husky_1 that matched with the largest scale feature in Husky_3. The matching can be seen below:

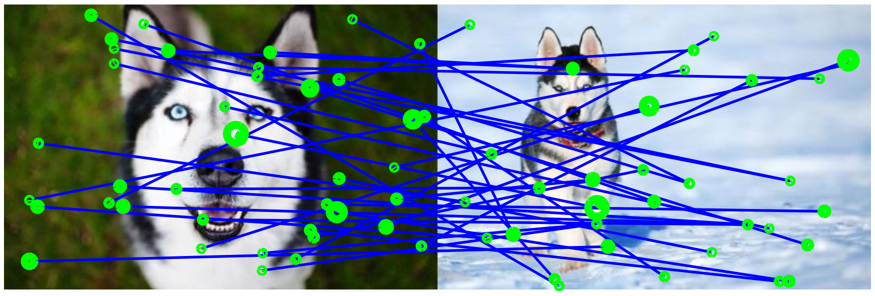


The feature matching is not accurate, but these two features are matched because the surrounding pixels of these features are black, and the algorithm finds that these two are similar. These keypoints are matched, which shows that the SIFT algorithm is invariant to scaling, rotation and translation.

I have used nearest neighbor search to find highest probability match in the images.

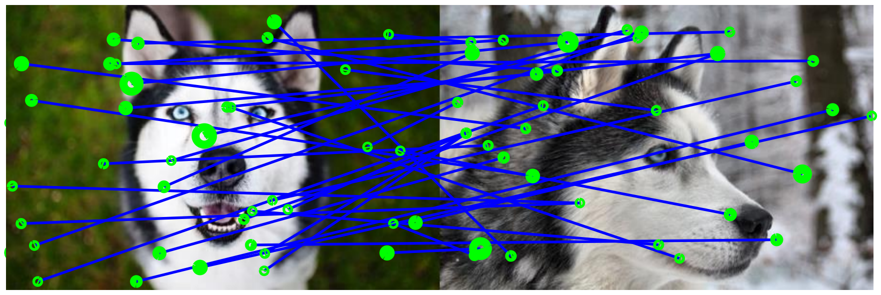
The orientations are assigned by using a histogram and the regions around it. This way we get keypoints that are distinctive and we use these points for matching.

2.
Husky3 and Husky 1 SIFT pairs
The matched features of these two images can be seen below. The feature matching has some positives but quite a few mistakes. This is because the viewing angle of the huskies is almost the same. But, the Husky 1 image has a lot of background and Husky_3 only has the face of the dog. Therefore there are quite a few mistakes.



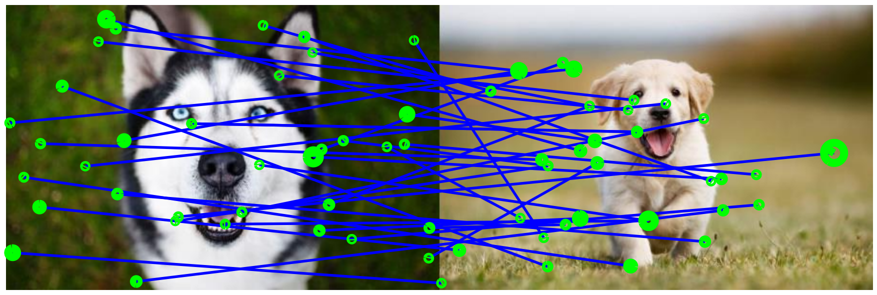
Husky3 and husky2

In this, the features are matched and some of the matching works well because both the images are zoomed in to the dogs' faces. But, the viewing angle of the dogs is different so many of the matchings are not good at all.



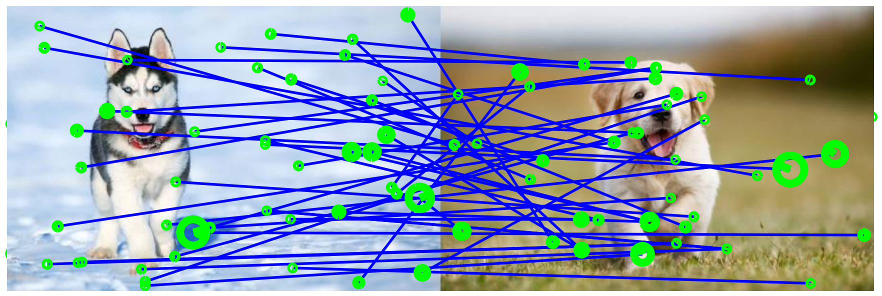
Husky3 and Puppy1

The matching does not work well at all with these two images. The reason is that the two dogs are of different breeds and the Husky_3 is a zoomed in image of the face of the dog and the Puppy_1 has a dog walking and the image has a lot of the background. So, these two images don’t have good matchings.



Husky1 and Puppy1

The matching between Husky_1 and Puppy_1 works decently okay, but it is not that great. The reason for it to work would be that both the images have a similar size of the 2 dogs, but the dogs' breeds are different, so their features are very different from one another. So, the matching would have worked better if the Puppy_1 image has a husky too in the same pose.

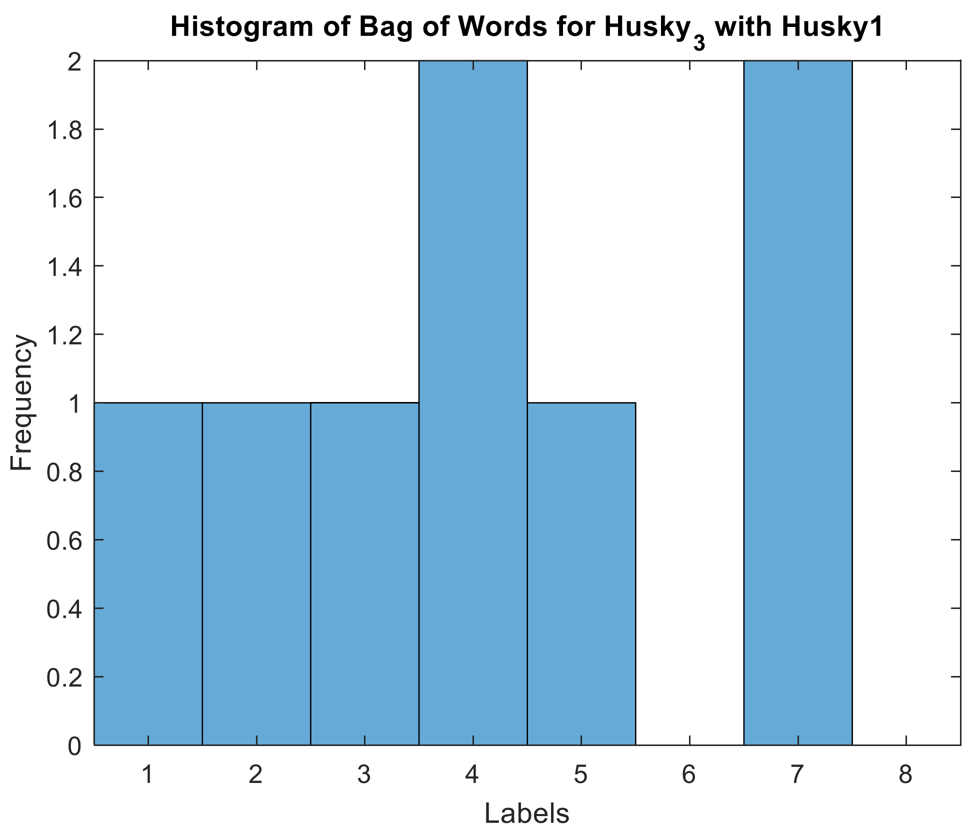


Problem 2:

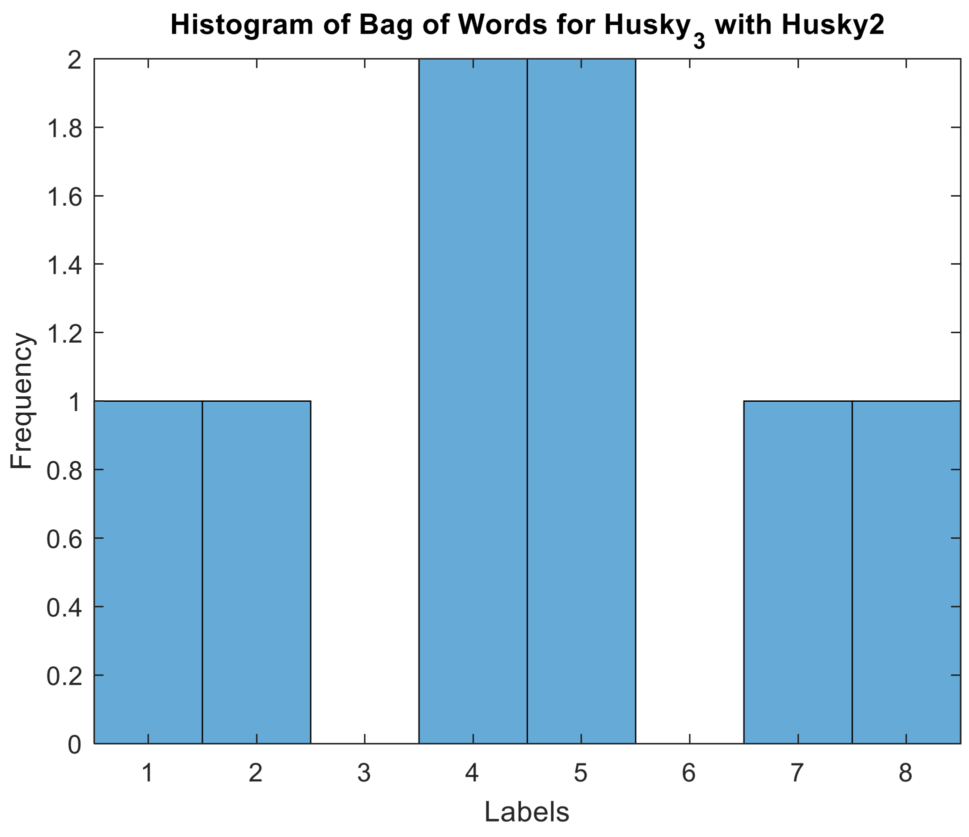
(c)

I applied the k means clustering to form codebooks of the three images (Husky_1, Husky_2 and Puppy_1). I formed codebooks of these 3 images because the features of these 3 were to be compared with the Husky_3 image to check their similarity.

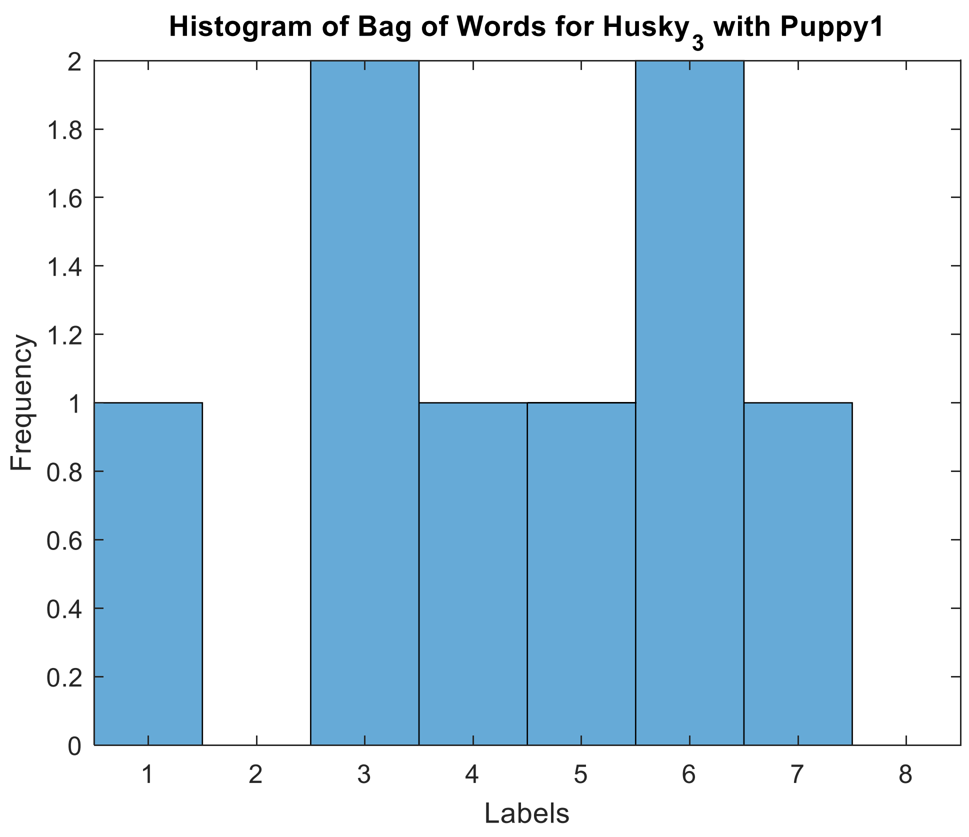
The histogram for Husky_1 with Husky_3, Husky_2 with Husky_3 and Puppy_1 with Husky_3 can be seen below:
These histograms represent the 8 features (codebook) on the x axis and the y axis measures how similar these features are to the Husky_3 image.
The y axis represents the distance from their centroids. More the distance, farther away are the centroids and less similar are the features to one another.



Histogram of Bag of Words for $Husky_3$ with Husky1
Frequency
Labels

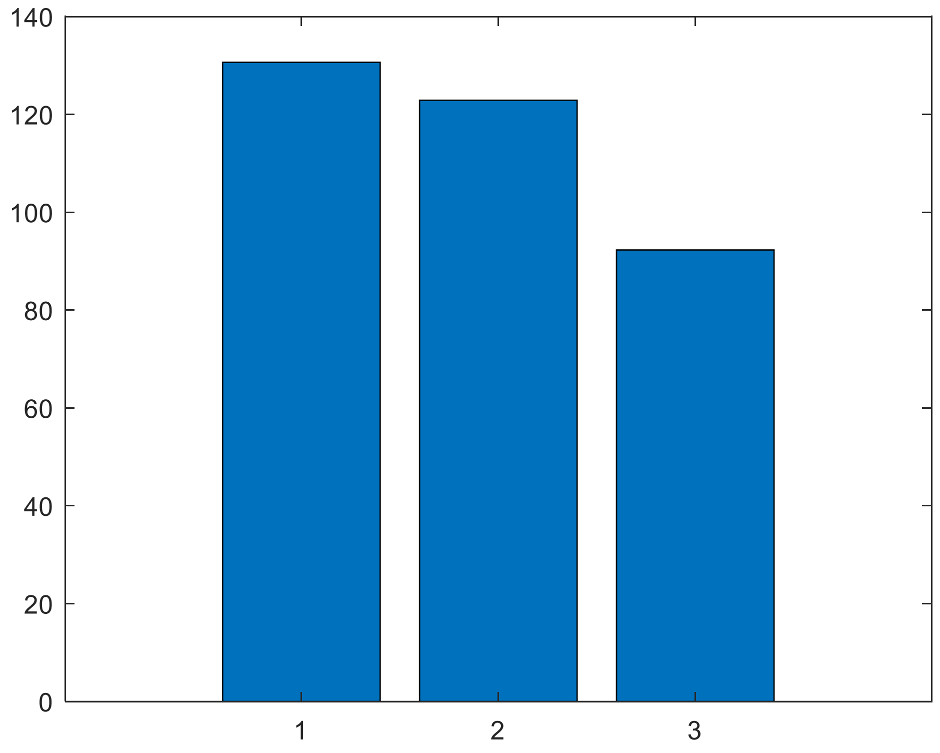


Histogram of Bag of Words for $Husky_3$ with Husky2
Frequency
Labels



The figure below represents the 3 images as one feature on the x axis and their similarity with Husky_3 is represented by the value of the y axis (i.e. the length of each bar)

The higher the bar, the lesser the Husky_3 is similar to that particular image.



From this we can see that Huksky_3 is most like Husky_1 (0), then Husky_2 (1) and then Puppy_1 (2)