HOMEWORK #4

Issued: 03/04/2020 Due: 03/22/2020

Problem 1: Texture Analysis and Segmentation

I. **Abstract and Motivation:** As there is no formal mathematical definition for textures, we can define textures by examples. However, in a nutshell textures can be defined as the repetition of **textons** (basic structure). Texture analysis is important because they are widely used in **Remote Sensing** and **Medical Image Processing.** Most of the aerial images are textures which include ocean, forests, landscapes, ultrasound and CT scan images. Figure 1 shows the example of textures.

Figure 1: Grass and Leaf textures

Texture analysis includes Texture classification, Texture Segmentation and Texture Synthesis. We will discuss Texture classification and Texture Segmentation in this section.

Texture Classification using Law's Filters: Kenneth Law who was a USC student came up with a set of filters in 1980 which are popularly known as Law's filters. He came up with a set of 1D filters of different dimensions and then found tensor products or outer products to obtain 2D filters [5].

The set of three 1D filters are as shown below,

$$L3 = (1, 2, 1)$$
$$E3 = (-1, 0, 1)$$
$$S3 = (-1, 2, -1)$$

L3 is for **level detection**, E3 is for **Edge detection** and S3 is for **spot detection.** Law's convolved the outer product/ tensor product of all the combination of above filters to obtain a set of symmetric and skew symmetric filters. He tried with different dimensions and he got good results for 5 dimensional filters. Figure 2 shows the 5 dimensional filters of Kenneth Law.

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

Figure 2: 5 dimension kernel of Law's filter

The results of tensor products of these filters are convolved with the Texture Images to obtain the projection of the image with respect to each filter. Law termed it as "macro statistic". He calculated the energy for each filter response and obtained the feature vector of each image for all combinations of the filters.

Figure 3 shows few filters obtained by 1D kernels.

-1	-2	0	2	1
-4	-8	0	8	4
-6	-12	0	12	6
-4	-8	0	8	4
-1	-2	0	2	1

L5E5

-1	0	2	0	-1
-2	0	4	0	-2
0	0	0	0	0
2	0	-4	0	2
1	0	-2	0	1

E5S5

1	-4	6	-4	1
-4	16	-24	16	-4
6	-24	36	-24	6
-4	16	-24	16	-4
1	-4	6	-4	1

R5R5

-1	0	2	0	-1
-4	0	8	0	-4
-6	0	12	0	-6
-4	0	8	0	-4
-1	0	2	0	-1

L5S5

Figure 3: Law's filters after Tensor product.

Figure 4 shows the schematic representation of Law's filter responses.

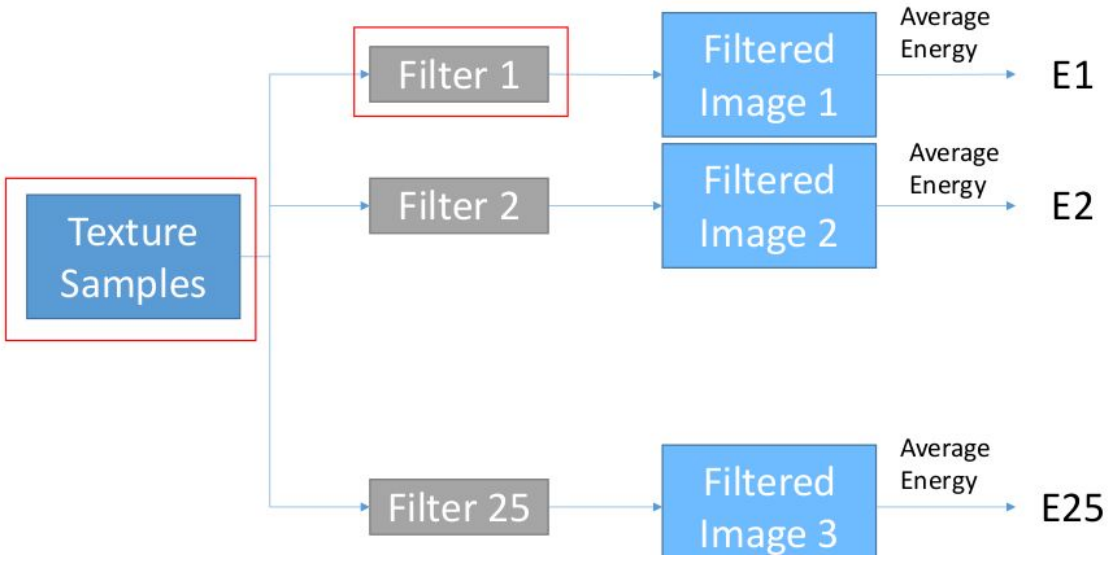


Figure 4: Schematic of feature extraction using Law's filters

Frequency response of Law's filters:

The magnitude response of each Law's filter and their properties are discussed in this section.

1. **Magnitude frequency response of L5 filter:** L5 is given by the filter coefficients (1, 4, 6, 4, 1). Filter magnitude response is given below.

$$\begin{aligned}\left|H_{L5}(\omega_1)\right| &= \left|e^{-j2\omega_1} + 4e^{-j\omega_1} + 6 + 4e^{j\omega_1} + e^{j2\omega_1}\right| \\ &= 4(1+\cos\omega_1)^2\end{aligned}$$

2. **Magnitude frequency response of E5 filter:** E5 is given by (-1, -2, 0,2,1). Filter magnitude response is given below.

$$\begin{aligned}\left|H_{E5}(\omega_1)\right| &= \left|H_{L3}(\omega_1).H_{E3}(\omega_1)\right| \\ &= 4\sin\omega_1\left(1+\cos\omega_1\right)\end{aligned}$$

3. **Magnitude frequency response of S5 filter:** S5 is given by (-1,0, 2,0,-1). Filter magnitude response is given below.

$$\begin{aligned}\left|H_{S5}(\omega_1)\right| &= \left|H_{E3}(\omega_1).H_{E3}(\omega_1)\right| \\ &= 4\sin^2\omega_1\end{aligned}$$

4. **Magnitude frequency response of R5 filter:** R5 is given by (1,-4,6,-4, 1). Filter magnitude response is given below.

$$\begin{aligned}\left|H_{R5}(\omega_1)\right| &= \left|H_{S3}(\omega_1).H_{S3}(\omega_1)\right| \\ &= 4\left(1-\cos\omega_1\right)^2\end{aligned}$$

L5 is a low pass filter, E5 is a Band pass filter and S5 is a High pass filter. So using the response of the filter bank, we divide the frequency spectrum into 25 regions.

Figure 5 shows the frequency response of each of the 1D Law's kernels.

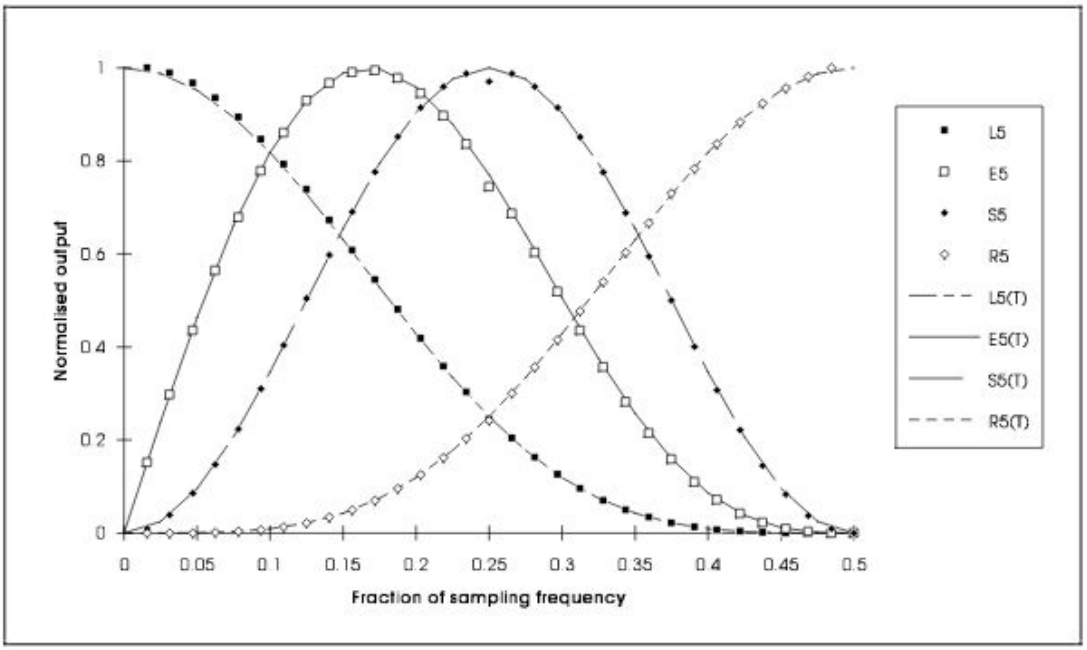


Figure 5: Frequency response of each of the 1D Law's kernels.

Similarly we find 2D frequency response of Law's filters. The 2D frequency response is shown below.

$$\begin{aligned}\left|H_{L5E5}(\omega_1,\omega_2)\right| &= \left|H_{L5}(\omega_2).H_{E5}(\omega_1)\right| \\ &= 4(1+\cos\omega_2)^2\,4\sin\omega_1\left(1+\cos\omega_1\right)\end{aligned}$$

$$\begin{aligned}\left|H_{E5S5}(\omega_1,\omega_2)\right| &= \left|H_{E5}(\omega_2).H_{S3}(\omega_1)\right| \\ &= 4\sin\omega_2\left(1+\cos\omega_2\right)4\sin^2\omega_1\end{aligned}$$

$$\begin{aligned}\left|H_{R5R5}(\omega_1,\omega_2)\right| &= \left|H_{R5}(\omega_2).H_{R5}(\omega_1)\right| \\ &= 4\left(1+\cos\omega_2\right)^2 4\left(1+\cos\omega_1\right)^2\end{aligned}$$

$$\begin{aligned}\left|H_{L5S5}(\omega_1,\omega_2)\right| &= \left|H_{L5}(\omega_2).H_{S5}(\omega_1)\right| \\ &= 4\left(1-\cos\omega_2\right)^2 4\sin^2\omega_1\end{aligned}$$

Figure 6-7 shows the frequency response plot of few 2D filter responses.

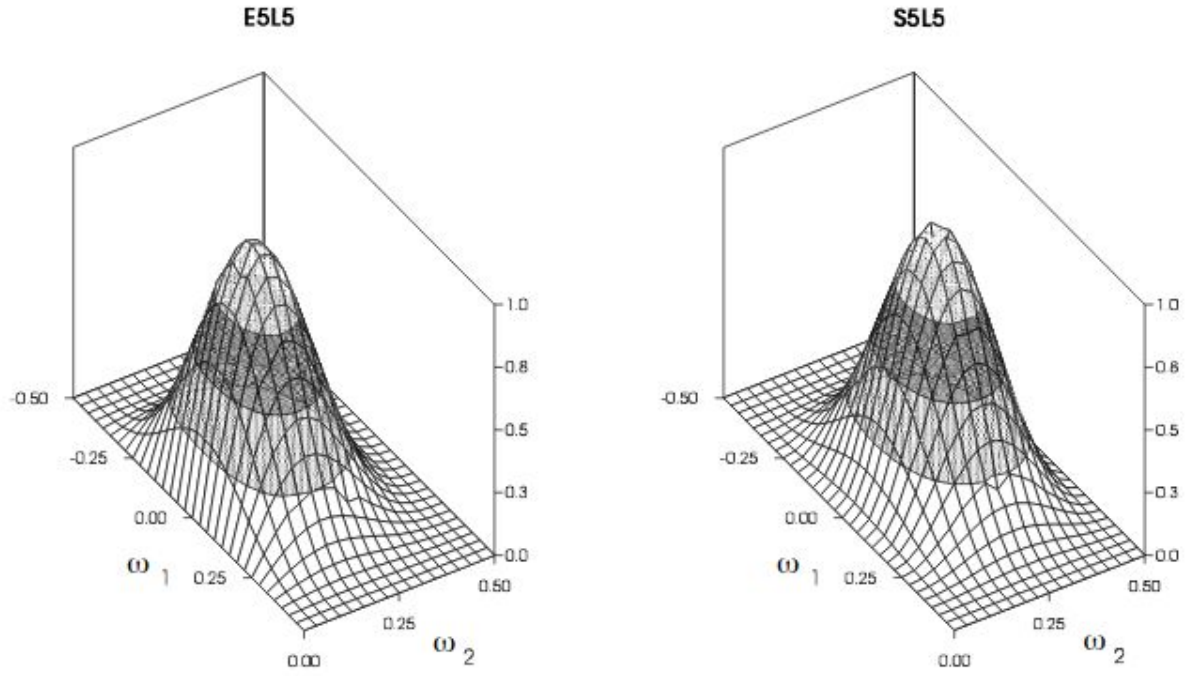


Figure 6: Frequency response of 2D E5L5 and S5L5

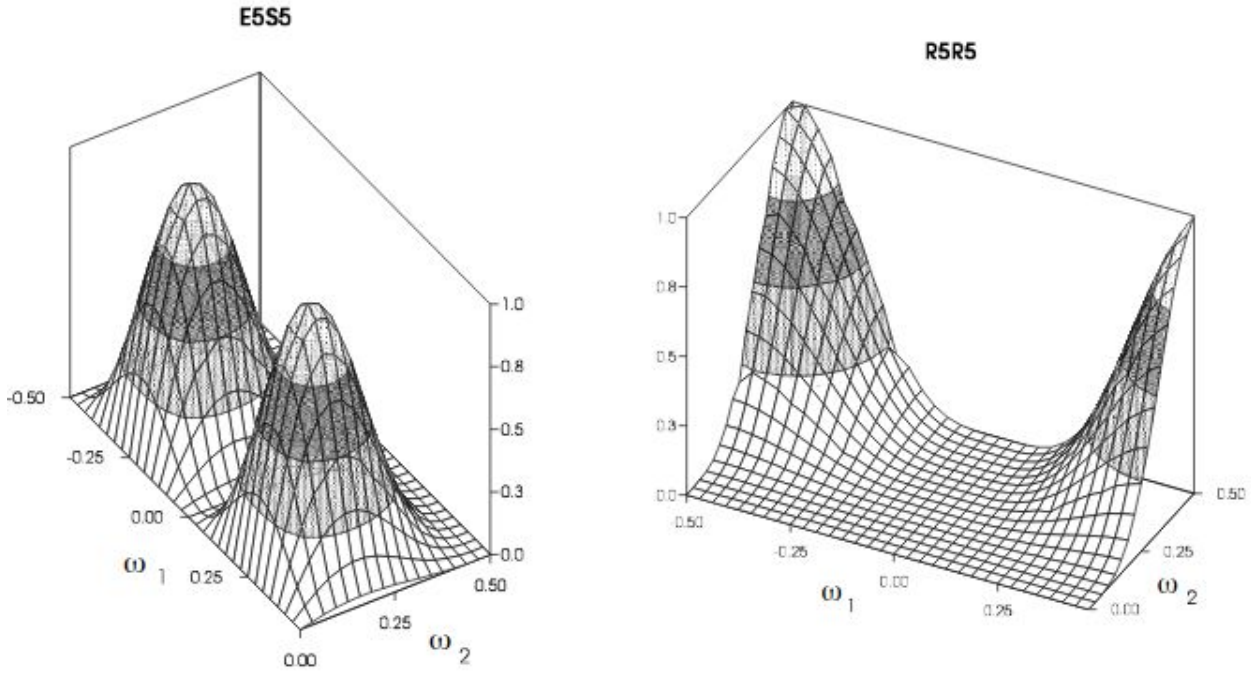


Figure 7: Frequency response of 2D E5S5 and R5R5

Figure 8 shows the 2D frequency response of all possible combinations.

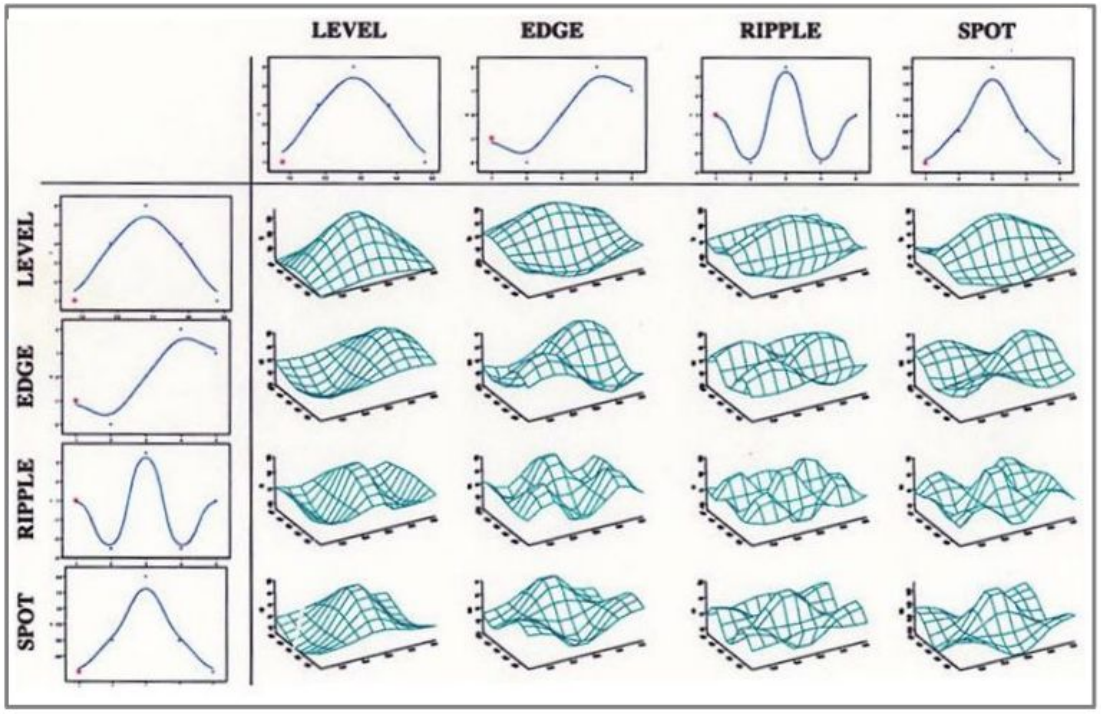


Figure 8: 2D Plot of Law's filters frequency responses

Texture Segmentation using Law's filter: Unlike texture classification, texture segmentation classifies each pixel into k classes of the textures. The procedure is the same as classification, but we use a window approach to compute the energy value of each pixel.

After finding the projection of each pixel to the frequency response of Law's filter bank. We use K-means clustering to classify each pixel to K clusters. The centroids of each cluster will be stored. For any new image we use the centroids and measure the Euclidean distance of each 25D feature vector with each centroid and assign to the respective class.

Inorder to improve the result, we use dimensionality reduction techniques like PCA.

Figure 9 shows the Example of Texture segmentation.

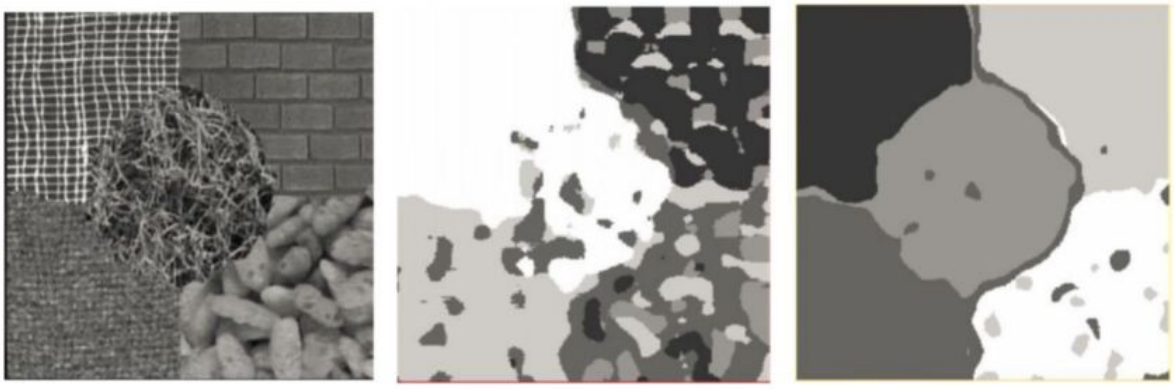


Figure 9: Texture segmentation

II. Approach and Implementation:

A. Texture Classification (Feature Extraction):

1. There were 48 texture images in total out of which 36 were training images and 12 were test images.
2. 36 training images are labelled and our first task is to extract the features from each image to construct a feature matrix of all the training images.
3. We read each image and store all the images as a 3D array in matlab. We now have 128*128*36 dimension array as training set.
4. We create 2D Law's filter using 1D Kernels. We get 25 Law's filters in the name of Filter Banks. The dimension of Filter Bank is 5*5*25.
5. Initially, we reduce the illumination of the image by subtracting the image average with all the pixel values.
6. After this step, we convolve each image with all the 25 filters. Now we get 25 Images for each filter response.
7. We average the absolute value of each filter response and construct a 25D feature vector for each image. The elements of the feature vector are called "Energy values".
8. Repeating this step for all the images results a 36*25 feature matrix.
9. But, L5E5 and E5L5 give the same projections. This is true for all other pairs as well. So, we need to reduce similar values and come up with a 15D matrix.
10. Implementing PCA will give low rank approximation of the feature matrix reducing it to 36*3 dimensions.

11. We repeat the same procedure for the test images and we get 12*15 and 12*3 feature matrix.
12. **Unsupervised Classification:** We use k-means clustering for classifying the test data from a 12*15 feature matrix. We note down the classification accuracy for 12*15 and 12*3 feature vectors.
13. **Supervised Classification:**SVM and Random Forest Classifiers are used to train the model on training images. We use the model to predict the classes of the test images. We compared the classification accuracy of all the classification methods.

B. Texture Segmentation:

1. The steps for texture segmentation are similar to Texture Classification.
2. Texture segmentation image is read and loaded to a 2 dimensional array in matlab. This gives 450*600 dimensional array.
3. Apply 25 Law's filter to each image to generate 25 responses for each image. We get 450*600*25 Dimensional array
4. Now, reduce 25 to 15 by averaging the similar valued filter responses.
5. At this stage, we follow windowing methods to find the energy values of each pixel. We define a window of size (typically 13*13) and average the absolute values in the window size and assign that value to the center pixel as energy value.
6. Since, L5'L5 has least discriminant power i.e., it doesn't contribute to the classification, we divide each dimension by the value of L5'L5 and remove 1st dimension. This gives us 450*600*14.
7. We convert this image to a 270000*14 feature matrix and assign labels to each pixel.
8. At the end we assign different gray level values to different labelled pixels.
9. **Improvisation:** To improve the result, we use PCA and hole filling method.
10. PCA and K-mean clustering algorithms are implemented from scratch according to the instructions given by the TA.

III. Results and Discussions:

A. Texture Classification (Feature Extraction):

a. **Results of Train Images:** Figure 10-13 shows the given labelled training images.

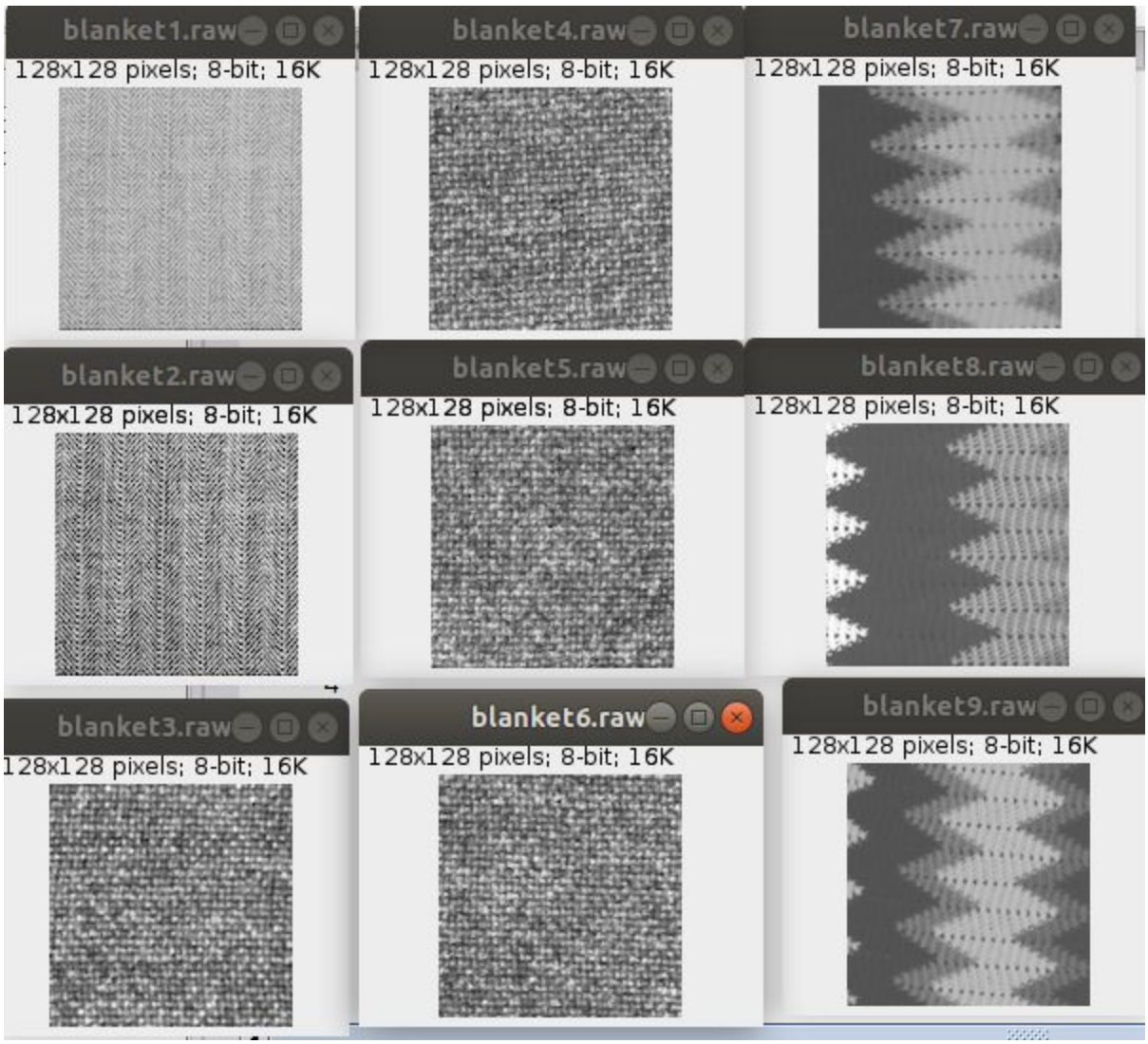


Figure 10: Blanket Train Image

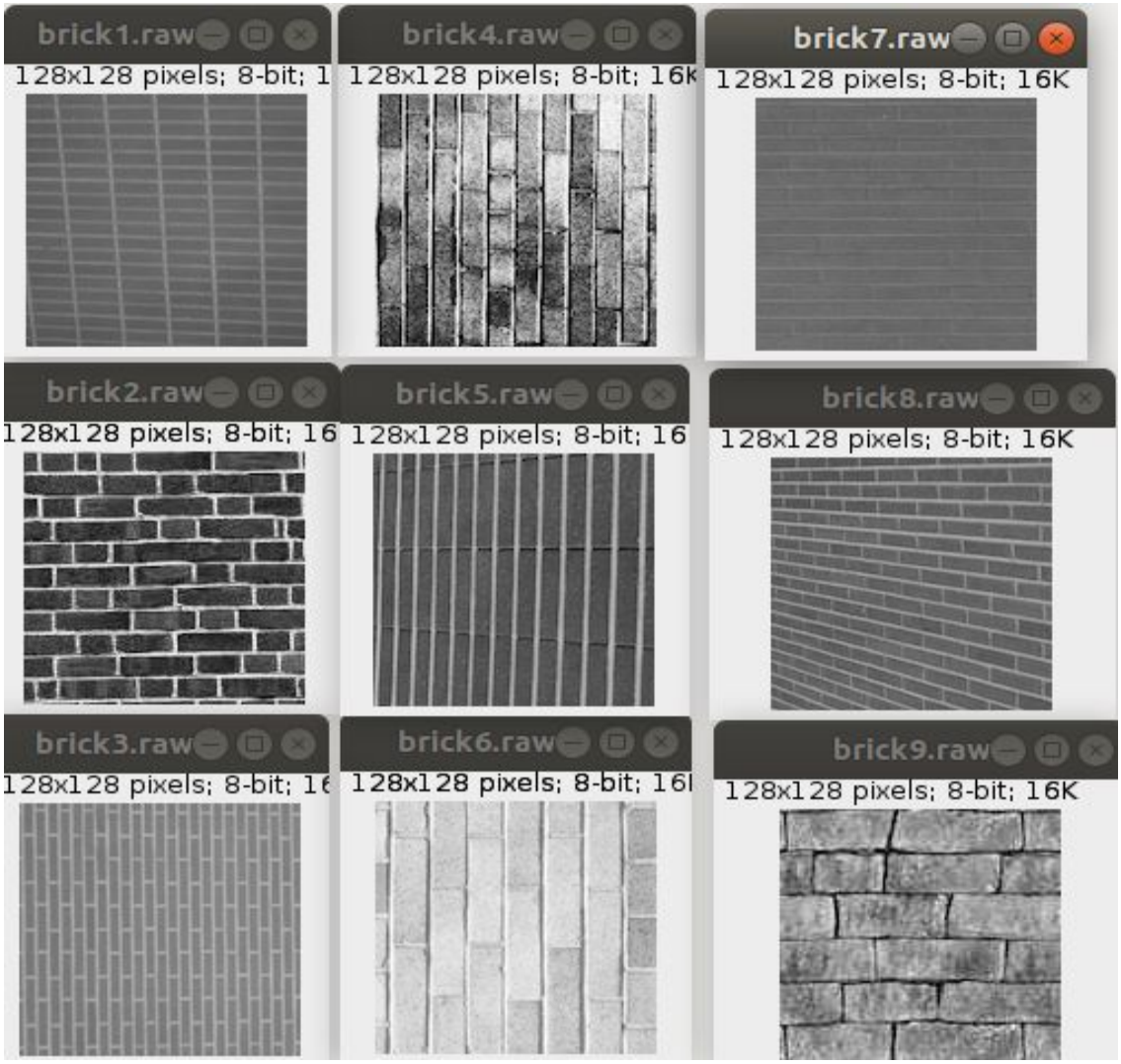


Figure 11: Brick Train Images

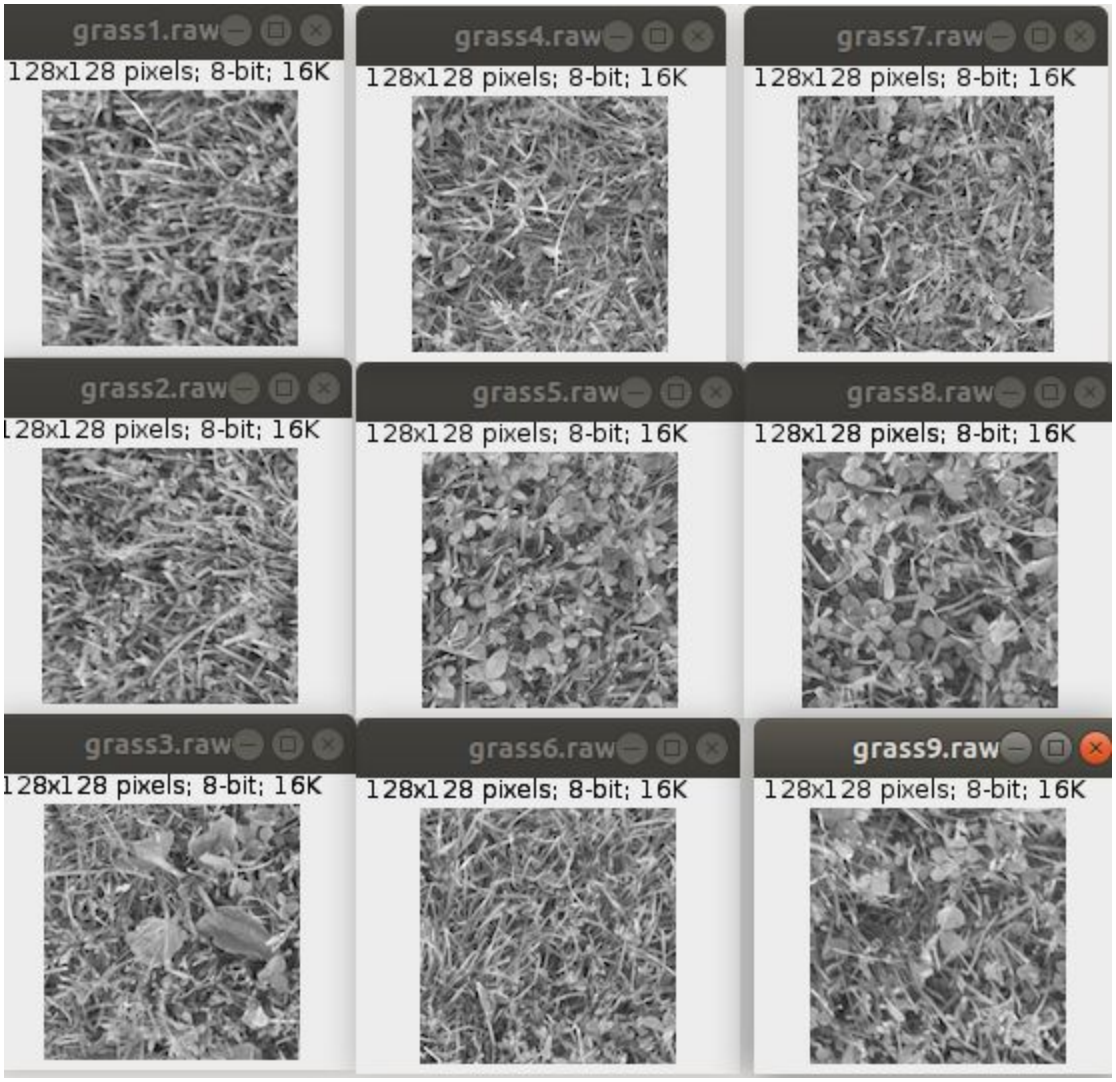


Figure 12: Grass Train Images

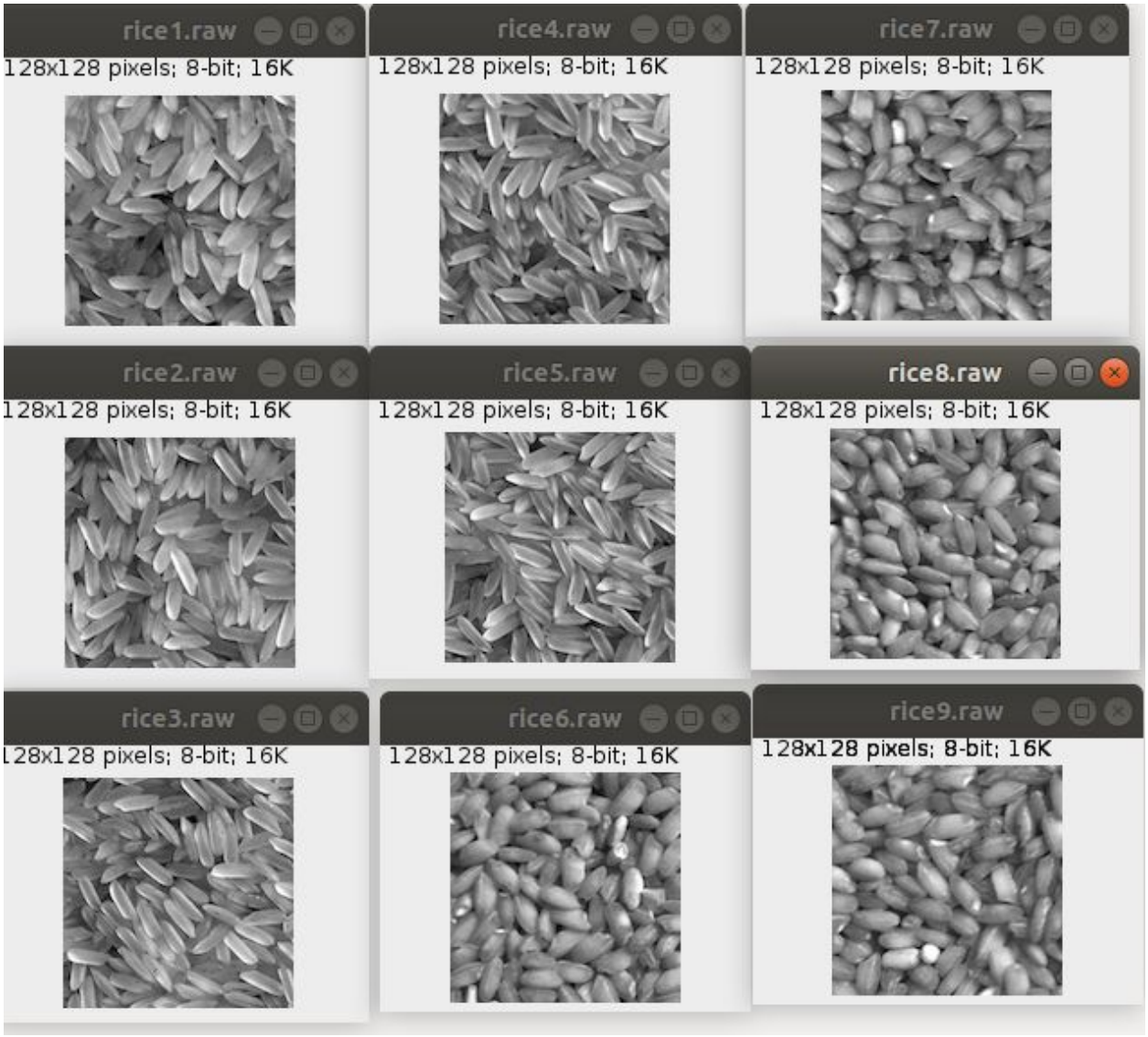


Figure 13: Rice Train Images

Discriminant Power: After extracting features from each image and building a feature 36*15 dimension matrix, we find the discriminant power of each of the features. We can find the discriminant power by calculating the variance of each dimension.

Table 1 shows the variance of each feature.

Features	Variance * 10^(6)
L5L5	4.5962
L5E5/E5L5	0.1153
E5S5/S5E5	0.0048
E5E5	0.0115
L5S5/S5L5	0.0552
E5W5/W5E5	0.0067
S5S5	0.0045
L5W5/W5L5	0.0684
E5R5/R5E5	0.0204
W5W5	0.0215
L5R5/R5L5	0.1953
S5W5/W5S5	0.0094
R5R5	0.1799
W5R5/R5W5	0.0622
S5R5/R5S5	0.0288

As we can see from the above image, the maximum variance is observed for L5L5 and maximum variance is observed for S5S5. **Therefore we can conclude that L5L5 has minimum discriminant power and S5S5 has Maximum Discriminant power.**

Reason: Variance measures the spread of the data. So, if variance is large enough, the density function converges to uniform function. On the other hand, if the variance is small then we get maximum change in the value of function. Therefore, small variance corresponds to strongest discriminant power and large variance corresponds to weakest discriminant power.

Also, this makes more sense if we analyze the frequency response of L5 and S5. L5 is the low pass filter and S5 is the high pass filter. Obviously, low pass filter response doesn't have much information whereas high pass filter responses have information regarding sharp boundaries and edges. **So, it is axiomatically justified that L5L5 has weakest discriminant power and S5S5 has strongest discriminant power.**

b. **Principal Component Analysis Results:** After getting a feature matrix of all the images (36*15), we use low rank approximation to project the higher dimension data to a 3 dimensional hyperplane. We use PCA to achieve this.

   PCA has been implemented by calculating the SVD of the covariance matrix. Code of PCA is attached in Figure 14.

```
function dimensionReducedFeatures = PCA_train(featureMatrix)
    data_std = zscore(featureMatrix);
    %Covariance matrix
    sigma = (1/12)*(data_std')*(data_std);
    %apply SVD to sigma
    [U, S, V] = svd(sigma);
    Ured = U(:,1:3);
    %finding k dim subspace (k = 3)
    subspace = Ured'*data_std';
    dimensionReducedFeatures = subspace';
    x = subspace(1,:)'; %First principal component
    y = subspace(2,:)'; %Second principal component
    z = subspace(3,:)'; %third principal component
    %Seggregating according to labels
    x1 = x(1:9,:); y1 = y(1:9,:);z1 = z(1:9,:);
    x2 = x(10:18,:);y2 = y(10:18,:); z2 = z(10:18,:);
    x3 = x(19:27,:); y3 = y(19:27,:); z3 = z(19:27,:);
    x4 = x(28:36,:);y4 = y(28:36,:); z4 = z(28:36,:);
    %Plotting
    scatter3(x1,y1,z1, 'r'); hold on;
    scatter3(x2, y2, z2, 'g');hold on;
    scatter3(x3, y3, z3, 'b'); hold on;
    scatter3(x4, y4, z4, 'c'); hold off
    xlabel('First Principal component');
    ylabel('Second Principal component');
    zlabel('Third Principal component');
    title('Train images');
end
```

Figure 14: Implementation of Principal Component Analysis

Figure 15 shows the result of PCA.

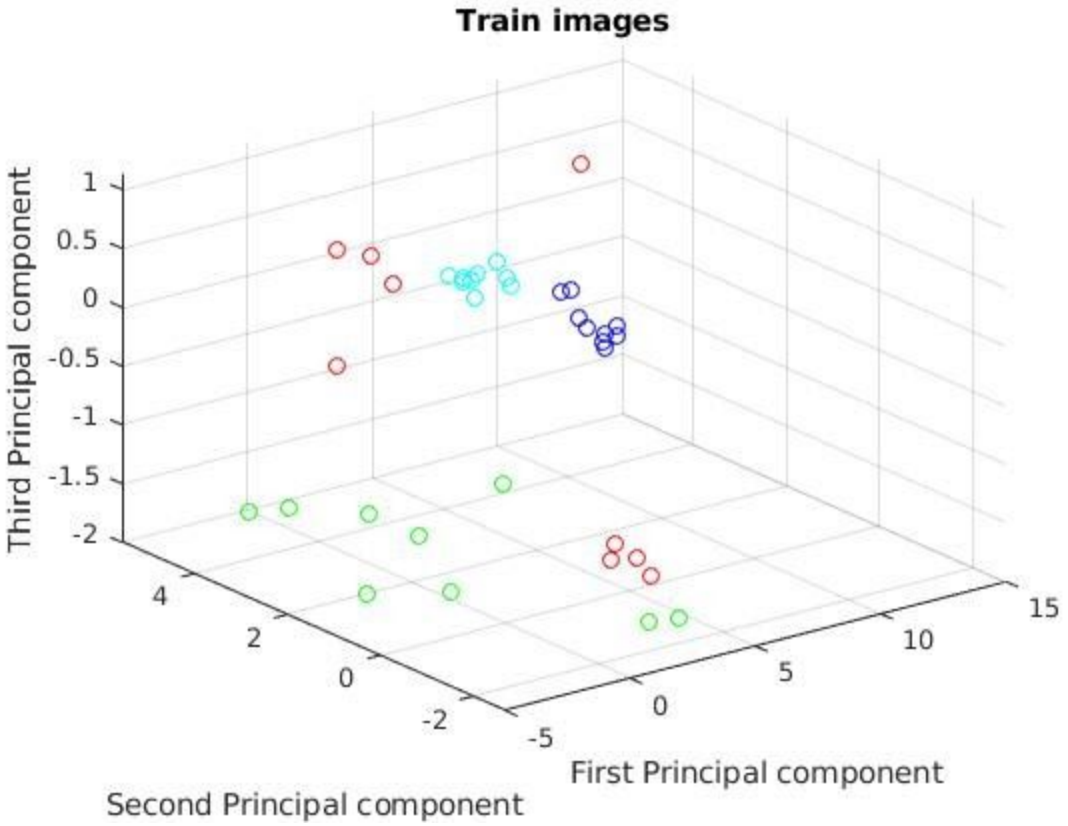


Figure 15: Plot of Principal Component Analysis.

Observations: As we can see from the plot in Figure 15 and Figures 10-13, the grass and rice images are almost similar which is represented by Cyan and Blue colors respectively. But there are 3 false images in the blanket images in the training set, so we have got a separate cluster of 3. The Bricks images represented in Green are spread widely. This is because the train images include a wide variety of brick images.

Observing the **eigenvalues of the covariance matrix** gives some important information regarding the feature matrix. Figure 16 shows the eigenvalues of the covariance matrix.

```
>> eig(cov)

ans =

    9.5426
    3.0220
    1.1536
    0.7089
    0.0955
    0.0426
    0.0102
    0.0053
    0.0014
    0.0008
    0.0003
    0.0001
    0.0000
    0.0000
    0.0000
```

Figure 16: Eigenvalues of the covariance matrix.

As we can see that the last three eigenvalues are 0. In a deeper sense, we have only the first three eigenvalues as dominant eigenvalues. This says a lot about the features. The features are highly correlated on each other because the covariance matrix is not a full rank matrix (non invertible).

c. **Results on Test Images:** We have 12 images as a test data set.
Figure 17 shows the test image data set. Similar to train images we will extract the feature matrix of the test images and apply PCA on that. We later use this reduced feature for supervised and unsupervised classification.

Figure 18 shows the plot of PCA on test data.

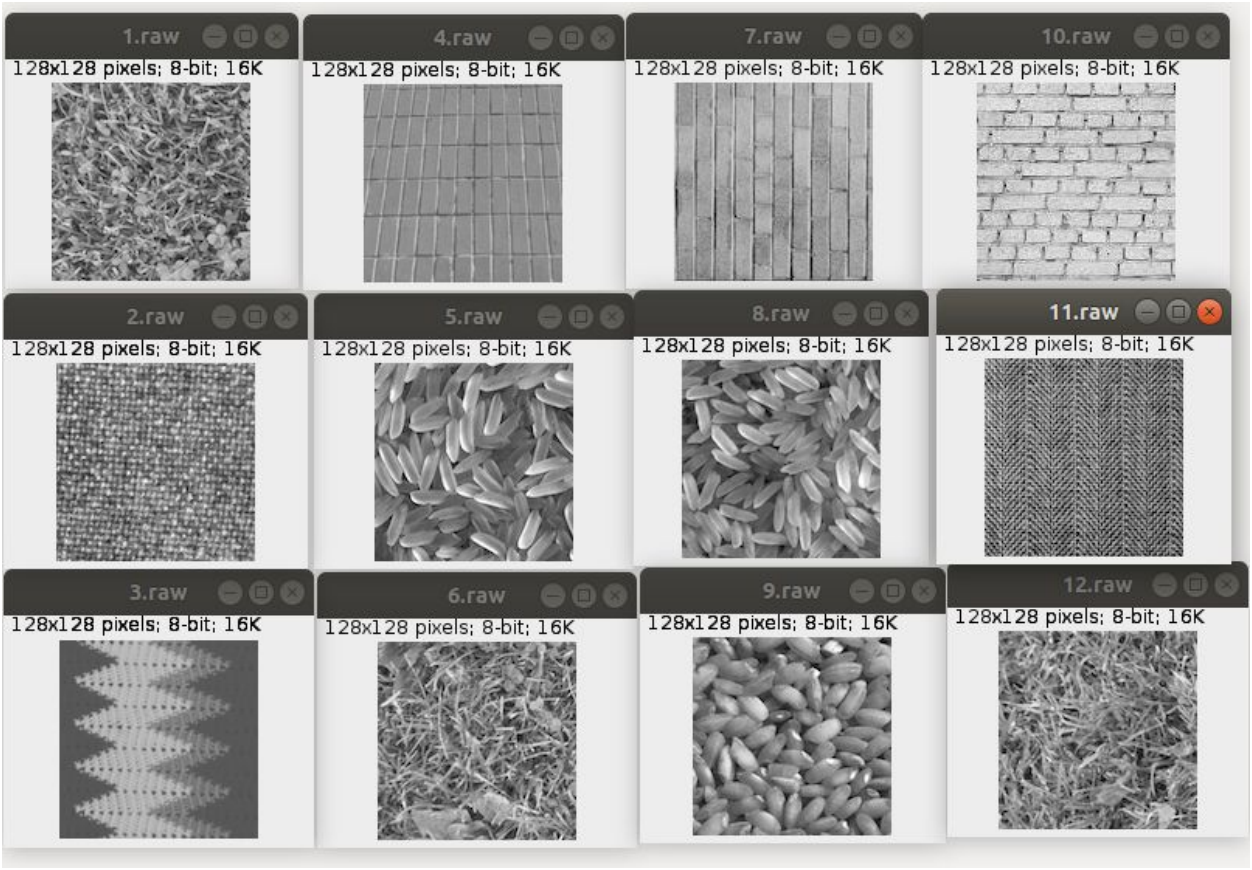


Figure 17: Test image data set

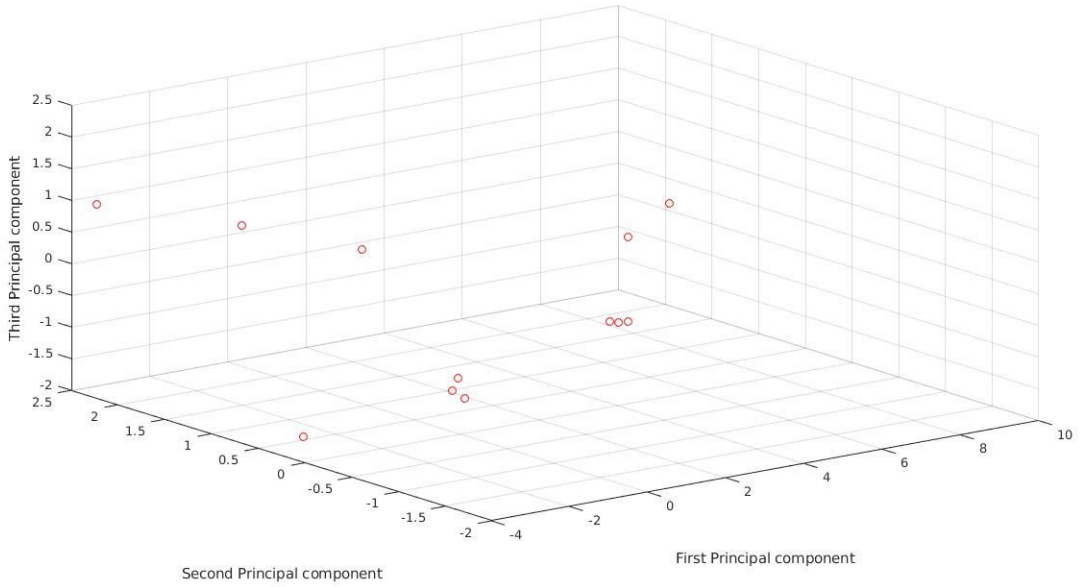


Figure 18: Plot of PCA on test data.

Similar to train data, even for the test data we get L5L5 as weakest discriminant power and S5S5 as strongest discriminant power.

B. Advanced Texture Classification:

a. Unsupervised Algorithm: We used K-mean clustering to classify the test data. K-mean algorithm is implemented from scratch and code is attached in Figure 19.

```
function [labels,clusterMatrix] = kmeans_code(featureMatrix, clusters) %#o
    %Randomly initialize k cluster centroids
    centroids = randn(clusters, 3);
    lables = zeros(270000, 1);
    while(cnt < 12000)
    %iterate through all the samples and assign each sample to one of the
    %clusters
    for i = 1:270000
       % find euclidean distance between
       diff = centroids(1,:)- featureMatrix(i,:);
       minimumDist = sqrt(diff*diff');
       idx = 1;
       for j = 1:clusters
           diff = centroids(j,:)- featureMatrix(i,:);
           dist = sqrt(diff*diff');
           if(dist< minimumDist)
                minimumDist = dist;
                idx = j;
           end
       end
       clusterMatrix(i,:,idx) =featureMatrix(i,:);  %#ok<AGROW>
    end
    %Find the average all the samles in each cluseter and update centroids
    for k = 1:clusters
       centroids(k,:) = sum(clusterMatrix(:,:,k))/sum(clusterMatrix);
    end
    end
```

```
    %Assigning labels
    |
    for k = 1:270000
        % find euclidean distance between
       diff = centroids(1,:)- featureMatrix(i,:);
       minimumDist = sqrt(diff*diff');
       idx = 1;
       for j = 1:clusters
           diff = centroids(j,:)- featureMatrix(i,:);
           dist = sqrt(diff*diff');
           if(dist< minimumDist)
                minimumDist = dist;
                idx = j;
           end
       end
       lables(i) = idx;
    end
end
```

Figure 19: K Mean Clustering Code

Result of K-Mean algorithm without dimensionality reduction (PCA): If we observe Figure 17, we can say that **(4, 7, 10)** belongs to bricks, **(1,6,12)** belongs to grass, **(5, 8, 9)** belongs to rice, **(2,11)** belongs to blanket and **3** is an unknown image.

Figure 20 shows the result of K-mean classification on an unreduced feature matrix of test data.

```
>> unreducedTest = ans;
>> K_means_test(unreducedTest)

ans =

     3
     3
     2
     1
     4
     3
     1
     4
     4
     3
     1
     3
```

Figure 20: Result of K-mean on unreduced test data

We can see that 10, 2 and 11 are misclassified. The accuracy rate is **75%.**

Result of K-Mean algorithm with dimensionality reduction (PCA):

Figure 21 shows the result of K-mean classification on dimension reduced data.

```
>> K_means(cluster)

ans =

     4
     2
     3
     1
     4
     4
     1
     4
     4
     1
     2
     4
```

Figure 21: K-mean classification on dimension reduced data.

In this result we can observe that (5, 8, 9) and (1, 6, 12) are being assigned to the same class. This is because we have an outlier in 3 and one entire class is being assigned to that. So, the misclassification is 2 in this case. Therefore the accuracy is **83.33%.**

Effectiveness of PCA on K-means algorithm: As we can see that the accuracy has been increased a bit after doing PCA. This is because PCA projects data in higher dimension to a lower dimensional subspace. This makes the data more categorical as we remove redundant features. The given test data is very small to observe a significant jump in the accuracy, but in real work problems PCA increases the accuracy significantly.

b. **Supervised Algorithms:** Unlike unsupervised algorithms, we train a learning model using the training data and predict the labels of test data. We use 2 supervised classifiers in this section namely Support Vector Machine and Random Forest algorithm.

Support Vector Machine: Support vector machine is a binary classifier which gives a decision boundary between two classes which has maximum margin between the outlier data points. We combine SVM with one vs one or one vs rest to make it a multiclass classification.

We train the SVM model using the “fitcecoc” function of matlab. We get the model with some parameters. Figure 22 shows the trained model using the given train images.

```
>> trainedSVMModel

trainedSVMModel =

  ClassificationECOC
             ResponseName: 'Y'
    CategoricalPredictors: []
               ClassNames: [0 1 2 3]
          ScoreTransform: 'none'
          BinaryLearners: {6×1 cell}
              CodingName: 'onevsone'


  Properties, Methods

>> trainedSVMModel.CodingMatrix

ans =

     1     1     1     0     0     0
    -1     0     0     1     1     0
     0    -1     0    -1     0     1
     0     0    -1     0    -1    -1
```

Figure 22: Tained model using the given train images.

Coding matrix gives the weight vector of a one vs one discriminant function. Initially in supervised learning, we find the train accuracy just to

make sure the model is working well on training data. Figure 23 shows the predicted labels of SVM train data.

```
label =

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

>>
```

Figure 23: Predicted labels of SVM train data.

We can see that there is only one misclassification leading to a train accuracy of **97.22%**

Figure 24 shows the predicted labels of SVM classifiers on test data.

```
>> [label] = predict(trainedSVMModel, testData)

label =

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

Figure 24: Predicted labels of SVM classifiers on test data.

As we can see, we are getting only 3 class labels in the test prediction. This is because we are using linear kernels while training SVM. (1, 6, 12) has been assigned to class 2, (4, 7, 10) has been assigned to class 0, (5, 8, 9) has been assigned to class 1, but class (2, 11) and (3) has been misclassified. This gives the **test accuracy of 75% with 3 misclassifications.**

Random Forest: Random forest is a statistical method of randomly sampling the features and building multiple decision trees. The final prediction will be made based on the maximum number of decisions made by all the decision trees. We train the Random forest model using train data and obtain the predicted labels for test data.

Challenging part is choosing the number of trees. Since we have less number of features (after feature reduction) we should choose less that 10 trees (convention). In this experiment we verified the classification accuracy using 5 decision trees.

Figure 25 shows the Random Forest Train model on given training images.

```
  TreeBagger
Ensemble with 5 bagged decision trees:
                    Training X:              [36x3]
                    Training Y:              [36x1]
                        Method:      classification
                NumPredictors:                   3
       NumPredictorsToSample:                    2
                   MinLeafSize:                   1
                InBagFraction:                    1
       SampleWithReplacement:                     1
        ComputeOOBPrediction:                     1
ComputeOOBPredictorImportance:                    0
                    Proximity:                   []
                   ClassNames:        '0'        '1'        '2'        '3'

  Properties, Methods
```

Figure 25: Random Forest Train model on given training images.

We analyzed the decision trees obtained by multiple decision trees of the random forest. Figure 26-27 shows the individual decision trees.

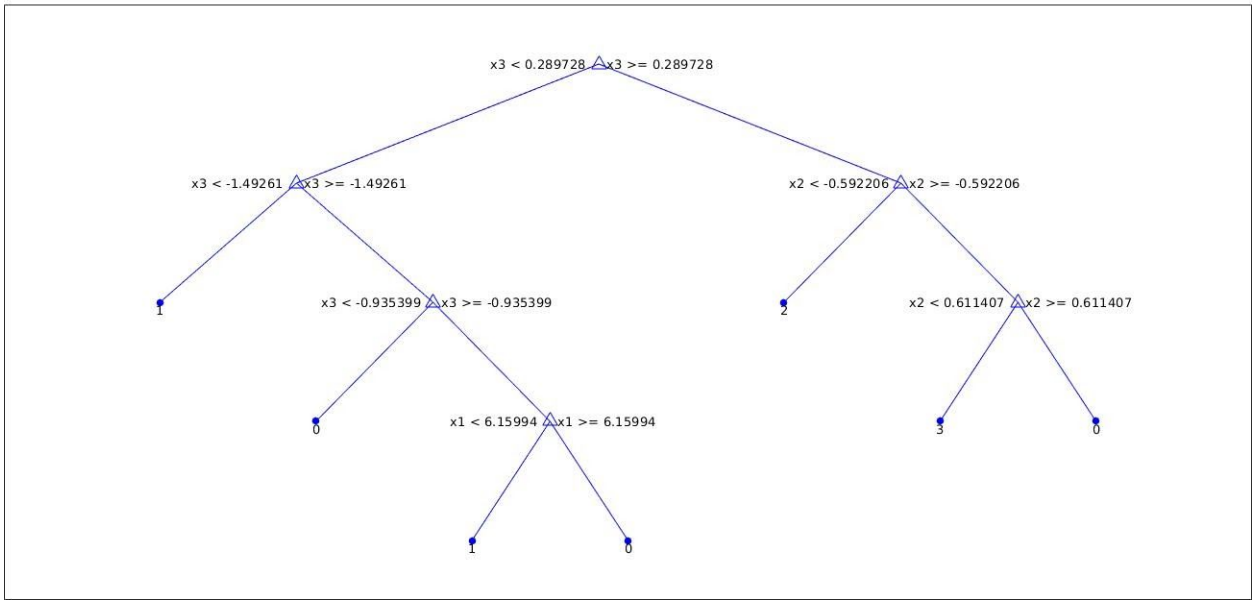


Figure 26: Decision tree 5

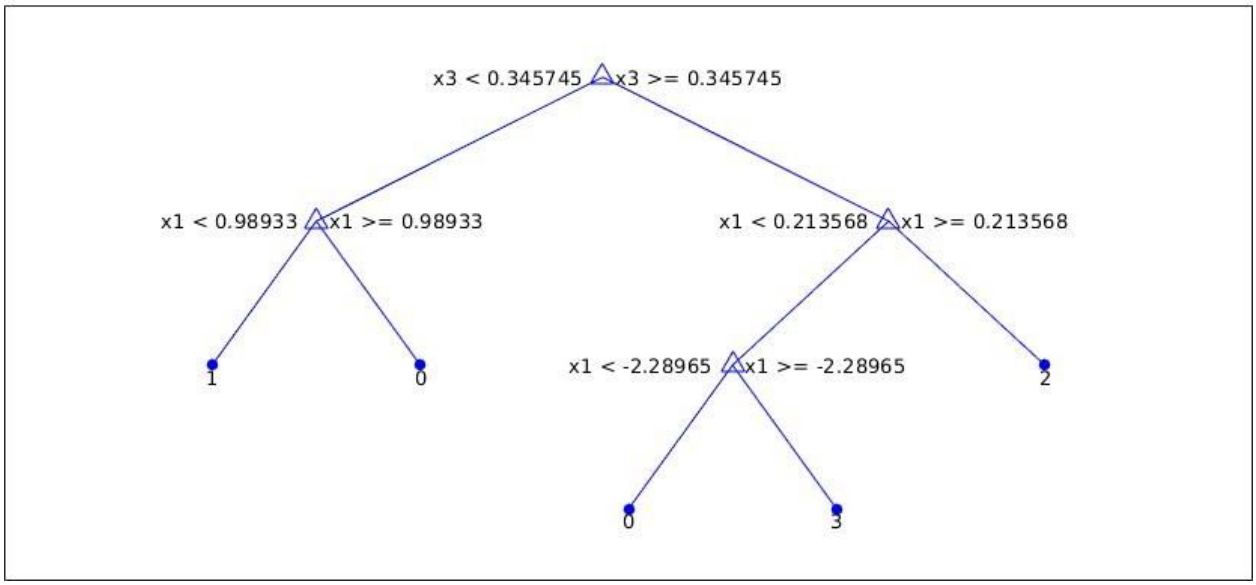


Figure 27: Decision tree 1

The train accuracy is the same as SVM. We got an accuracy of 97.22% on trai data for Random Forest also. Figure 28 shows the prediction labels on test data.

```
>> RandomForestClassifier(testdata, labels)

ans =

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

Figure 28: Prediction labels on test data.

As we can see there is only one misclassification. Therefore Random Forest gives an accuracy of **91.66%** on given test data.

Table 2 shows the accuracy of all three classifiers on test data.

Table 2: Accuracy Comparison

After PCA	K-Means	SVM	Random Forest
Accuracy	83.33 (75% - without PCA)	75%	91.66%

Comparison: Random Forest gives the highest accuracy among the three classifiers. K-mean clustering and SVM are nevertheless the same because there is a difference of only 1 misclassification. However, SVM with Gaussian Kernel gives more accurate results. Therefore, in general Supervised Learning Algorithms give better accuracy than un supervised learning algorithms.

K-means is a non-convex optimization and we won't get accurate classification and the algorithm is highly unstable for the initialization of centroids.

C. **Texture Segmentation:** Implemented the Segmentation algorithm on the given image. Figure 29 shows the image.

Results are tested for different window sizes.

Figure 30 shows the result of Segmentation for 13*13 Window and Figure 31 shows the result for 15*15 window.

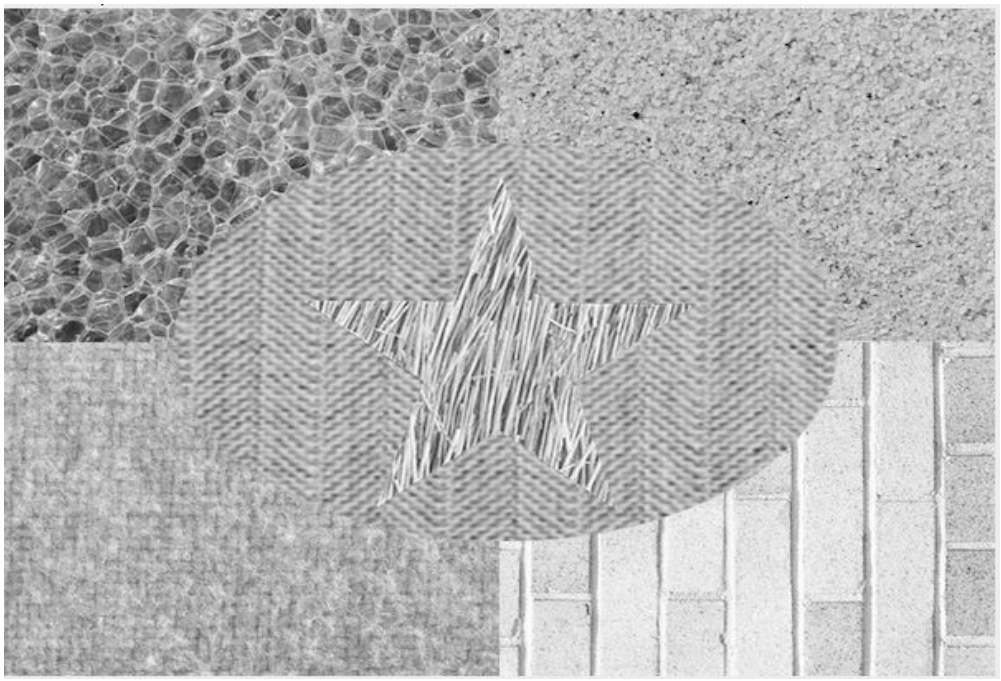


Figure 29: Input Image for Texture Segmentation

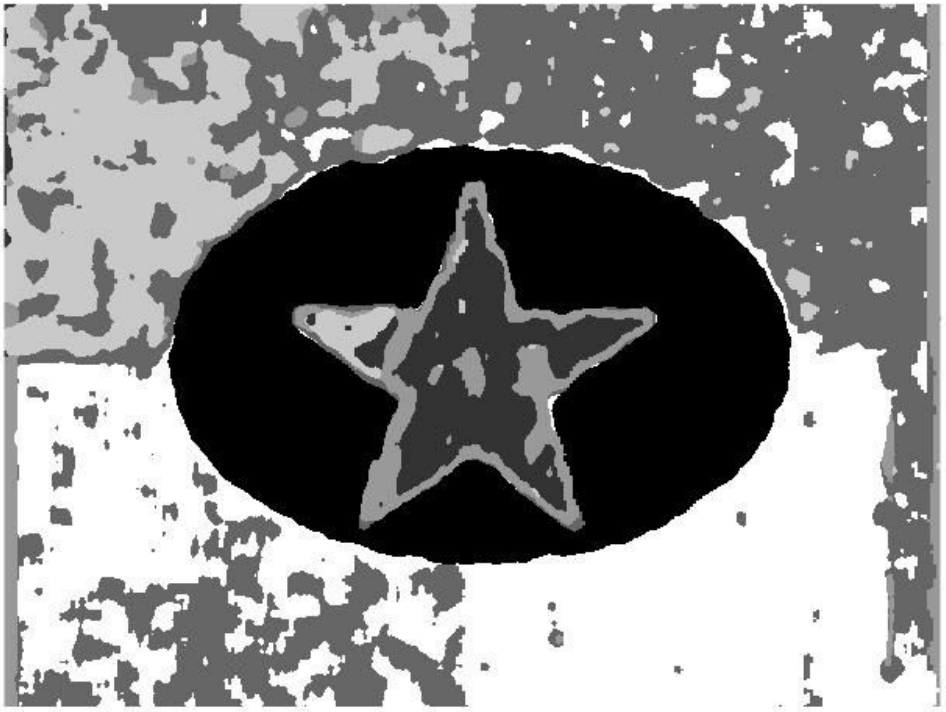


Figure 30: Result of Segmentation for 13*13 Window

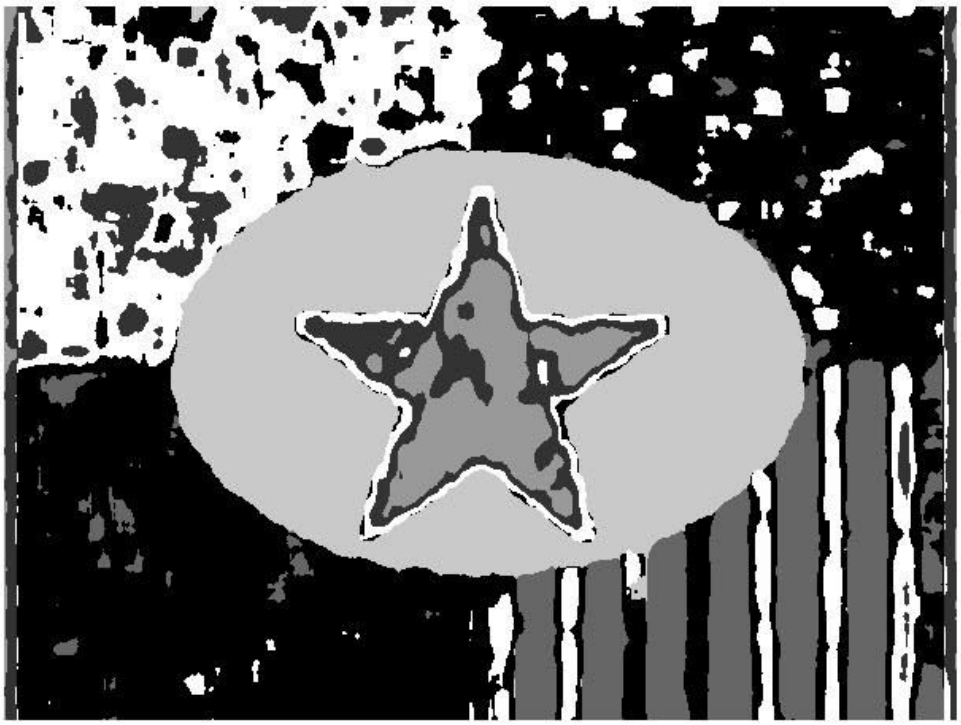


Figure 31: Result of Segmentation for 15*15 Window

Discussion: As we can see that the segmented image is bit more distorted with 13*13 windows compared to 15*15. Unlike texture classification, each pixel is a feature vector and we classify each pixel to one of the 6 classes. In this case we directly assumed the number of classes as 6 (Professor discussed in the class). However, ideally we take a different number of clusters and choose the best. We have used 270000*14 dimensional vector for K-mean clustering. By reducing the feature dimension using PCA we get better results. This is discussed in the next section.

D. **Improvisation of Texture Segmentation:** Instead of directly using the 270000*14 dimension feature matrix directly, we applied PCA and then used the feature reduced matrix for K-mean clustering. Inorder to remove the distortions, hole filling method is employed. This gave better results compared to the previous result.

Figure 32 shows the improvised segmented image.

Figure 33: Improvised segmented image.

Discussion: As we can see that the result has been considerably improved after PCA and hole filling. However further enhancements can be done by making boundaries more distinctive.

Problem 2: Image Feature Extractors

I. **Abstract and Motivation:** The advancement of Machine learning algorithms, specifically with respect to predictive modelling brought the new perspective towards computer vision especially in image classification [1]. Classification algorithms like Linear discriminant analysis, nearest neighborhood and SVM improved classification accuracy significantly. However, the feature extraction stage, especially from the images remained as one of the tough problems in the research community. Texture feature extraction was the rudimentary feature extraction technique where we can find many literatures. The algorithms improved gradually leading to the development of blob detectors, corner detectors and edge detectors. These feature extractors were efficient only on stoke-based images, but it terribly failed on real world images.

Scale Invariant Feature Transform: In 2004, David Lowe published his work which became one of the most cited research works in the Computer Vision community with the **total citations of 25688**. It is still considered as one of the influential research papers. The algorithm gained popularity because of its robustness and the features obtained are invariant of scaling, rotation, change in view angle, illumination and addition of noise.

Algorithm Overview:

1) Scale space extrema detection.
2) Key point localization.
3) Orientation assignment.
4) Keypoint descriptor.

- **Scale space extrema detection:** Research related to scale spaces to obtain different responses to a signal using different values of sigma was pioneered by Andrew Witkin [3]. His paper **"Scale-space Filtering - A new approach to Multiscale description"** explained in detail about the 1D scale spaces. This can be extended to 2D images to obtain images with different scales. Below steps briefly explains the method of building a scale space for an image from which we can extract feature points (key points).
    - First, we convolve the image with Gaussian kernels with different standard deviation values (variances).

- Doing this, we get images with different levels of smoothing.

$$L(x,\ y,\ \sigma_i)\ =\ G(x,\ y,\ \sigma_i) * I(x,y)$$

- Now we subtract resulting images with each other to get Laplacian of the image which is called **Scale Space.** But let us see what exactly is Laplacian of the image.

- **Approximation of Laplacian of Gaussian by Difference of Gaussian:**
  - The general heat equation is,

$$\frac{\partial G}{\partial \sigma} = \sigma \Delta^2 G$$

LHS is Difference of two Gaussiand and RHS is Laplacian.

$$\sigma \Delta^2 G = \frac{\partial G}{\partial \sigma} = \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \Delta^2 G$$

So, the Laplacian of Gaussian is obtained by difference of Gaussians with different values of sigma. Figure 1 schematically shows how to obtain Laplacian of Gaussian.

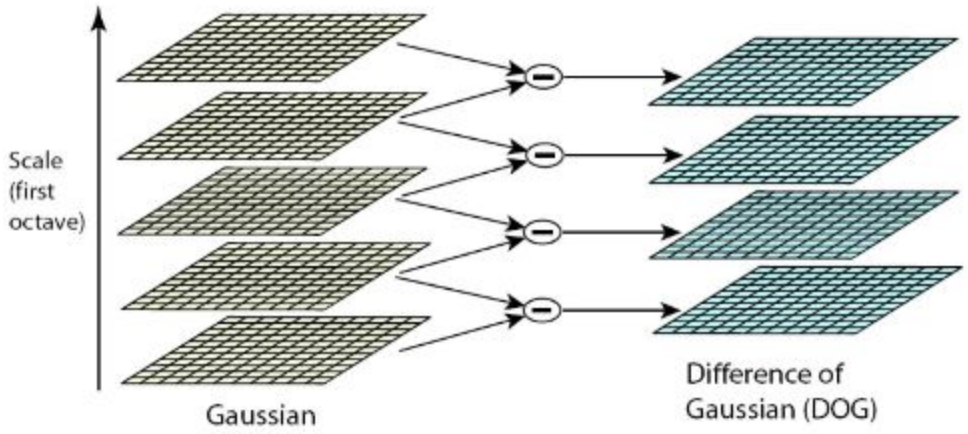


Figure 1: Laplacian of Gaussian using Difference of Gaussians

- Usually the typical values of k and sigma are **1.414 and 1.6** respectively.
- We obtain LoG for 5 scales and repeat the same procedure of 4 octaves. Figure 2 shows two different octaves.

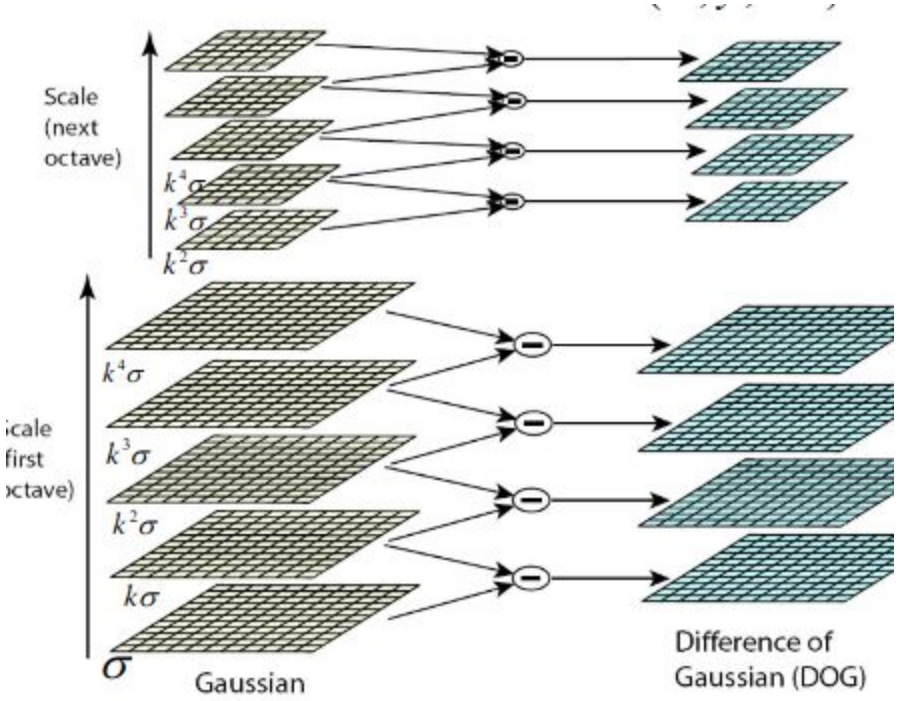


Figure 2: LoG for different octaves with 5 scales

- **Scale space peak detection:** This is the main step in keypoint selection. We compare each pixel in an octave with 26 pixels from above and below scales. Basically, we compare 3*3 patches from 3 scales above and below. We mark the pixel as a keypoint if it is only local minima or maxima. Figure 3 shows the keypoint selection.

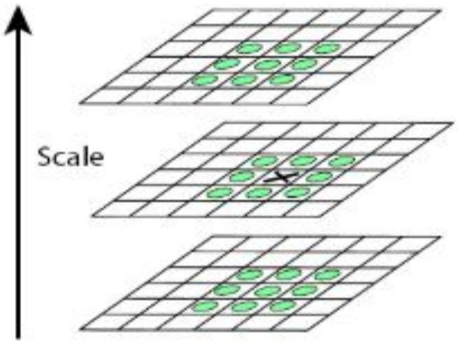


Figure 3: Scale space of LoG

This process will give many points which are often redundant and unnecessary. So we have to remove these points.

- **Key point localization:** In this step we eliminate outliers from the obtained key points in the previous stage**.** This will be done in 2-parse.
  - **Initial outlier elimination:** We eliminate low contrast pixels and poor pixels on the edges. To do this, we take Taylor series expansion of the Difference of Gaussians.

$$D(\mathbf{x}) = D + \frac{\partial D}{\partial \mathbf{x}}^T \mathbf{x} + \frac{1}{2}\mathbf{x}^\mathbf{T}\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \quad \Big| \quad \mathbf{x} = (x, y, \sigma)^T$$

Solving the above partial differential equation, we get maximum/minimum value at,

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}$$

  - **Further outlier elimination:** After removing the low contrast pixels, we need to eliminate edge maps. Difference of Gaussian gives a strong response to edges and noises (high frequency).
    - To achieve this, we assume LoG as a 2D surface.
    - So, along the edges the principal curvature gives low response and across the edges we get high response.

- We compute the Hessian Matrix of Difference of Gaussian function.

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \qquad Tr(H) = D_{xx} + D_{yy} = \lambda_1 + \lambda_2$$
$$Det(H) = D_{xx}D_{yy} - D_{xy}^2 = \lambda_1\lambda_2$$

- Now, we eliminate the outliers by evaluating a criterion function.

$$\frac{Tr(H)^2}{Det(H)} = \frac{(r+1)^2}{r} \qquad r = \frac{\lambda_1}{\lambda_2}$$
$$\frac{Tr(H)^2}{Det(H)} = \frac{(\lambda_1+\lambda_2)^2}{\lambda_1\lambda_2} = \frac{(r\lambda_2+\lambda_2)^2}{r\lambda_2^2} = \frac{(r+1)^2}{r}$$

- We remove the keypoint which has r>10. This step considerably eliminates many keypoints and keeps only the required and important ones.

- **Orientation assignment:** This step focuses mainly on making the features rotation invariant. Inorder to achieve this, we calculate central derivatives, direction of LoG and gradient magnitude at a particular scale of key point (x, y).

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

Now, we create a 36 bin histogram by scanning the weighted gradient direction at 10 degrees.
Figure 4 shows the Orientation assignment procedure.

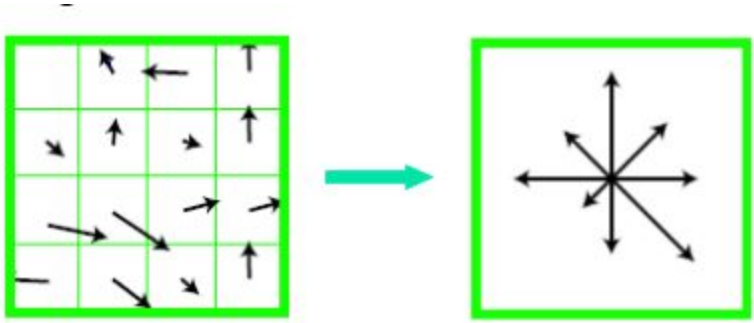


Figure 4: Orientation assignment procedure.

This gives orientation and direction of each keypoint.

- **Keypoint descriptor:** In this step, we describe each keypoint with a feature vector so that it represents the feature uniquely. Inorder to achieve this, we store sample of intensity values of the neighborhood pixels. This will provide invariance to illumination changes and 3D object transformation.
  We form weighted histogram each of 8 bin for a 4*4 regions to get 128 D feature vectors from each subblock. Figure 5 shows the Feature descriptor step.

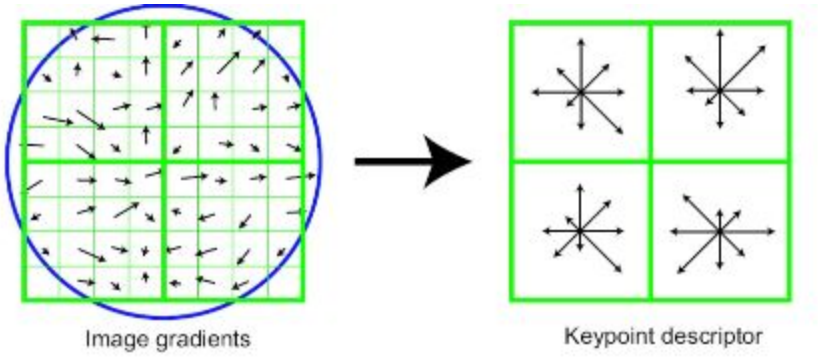


Figure 5: Feature descriptor Step.

- **Keypoint Matching:** When we want to match the features of 2 images, we obtain keypoints and their respective descriptors. We pick 1 keypoint from an image and find the L2 norm between the corresponding descriptor and all descriptors of the image. So, the nearest point is the match.

  We often get a close distance between 2 points in the target image. In that case, we can find the ratio between distance of 1st minimum and 2nd minimum. We usually take the ratio as 0.8.

Bag of Words (Visual Words): Image classification and object localization are the most challenging research domains in computer vision. Bag of Features a.k.a Bag of Visual words is the successful technique for Image classification and object localization before Convolution Neural Networks (CNN). BoW achieved an accuracy of 60% on different data sets.

Image classification is the problem of labelling images by assigning to one of the classes whereas object localization marks the location of the objects. Image classification has several problems which decrease the accuracy which includes camera positioning, variation in illumination and changes in internal parameters. Figure 6 shows the problem of orientation and variation of illumination.

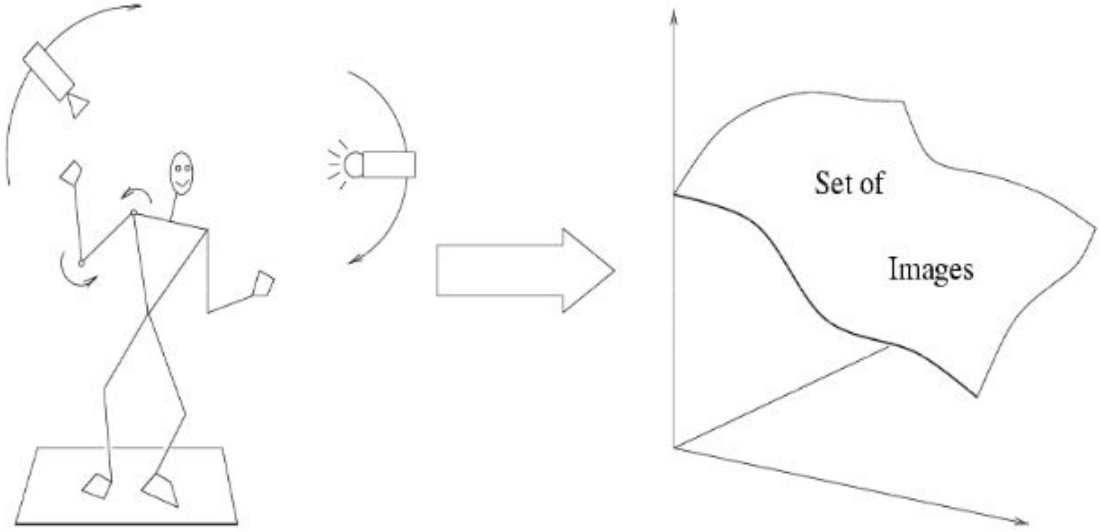
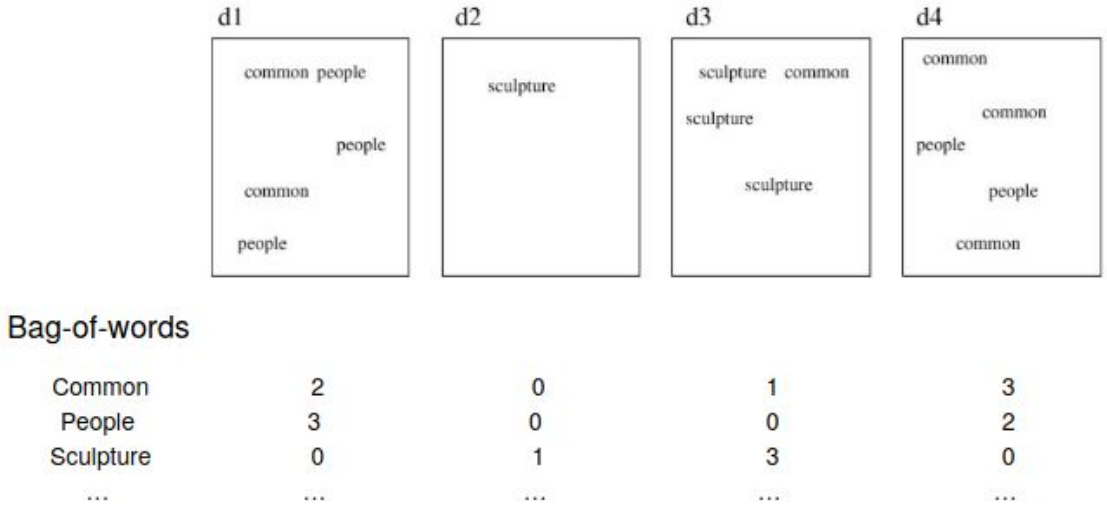


Figure 6: Problem of orientation and variation of illumination.

We can understand Bag of Visual Words as synonymous to Bag of words in documents. Suppose if we have n documents and if we find at least m keywords in each document, then we can find the frequency of each word in the document with respect to the keywords. This gives the histogram of the document which defines the document with respect to the given keywords. Figure 7 shows the graphical representation.



Bag-of-words

Common	2	0	1	3
People	3	0	0	2
Sculpture	0	1	3	0
…	…	…	…	…

Figure 7: Graphical Representation of Bag of Words (Text)

Similarly in Bag of words we match the images based on histogram intersection. Figure 8 shows the graphical representational flow of BoW with Image data.

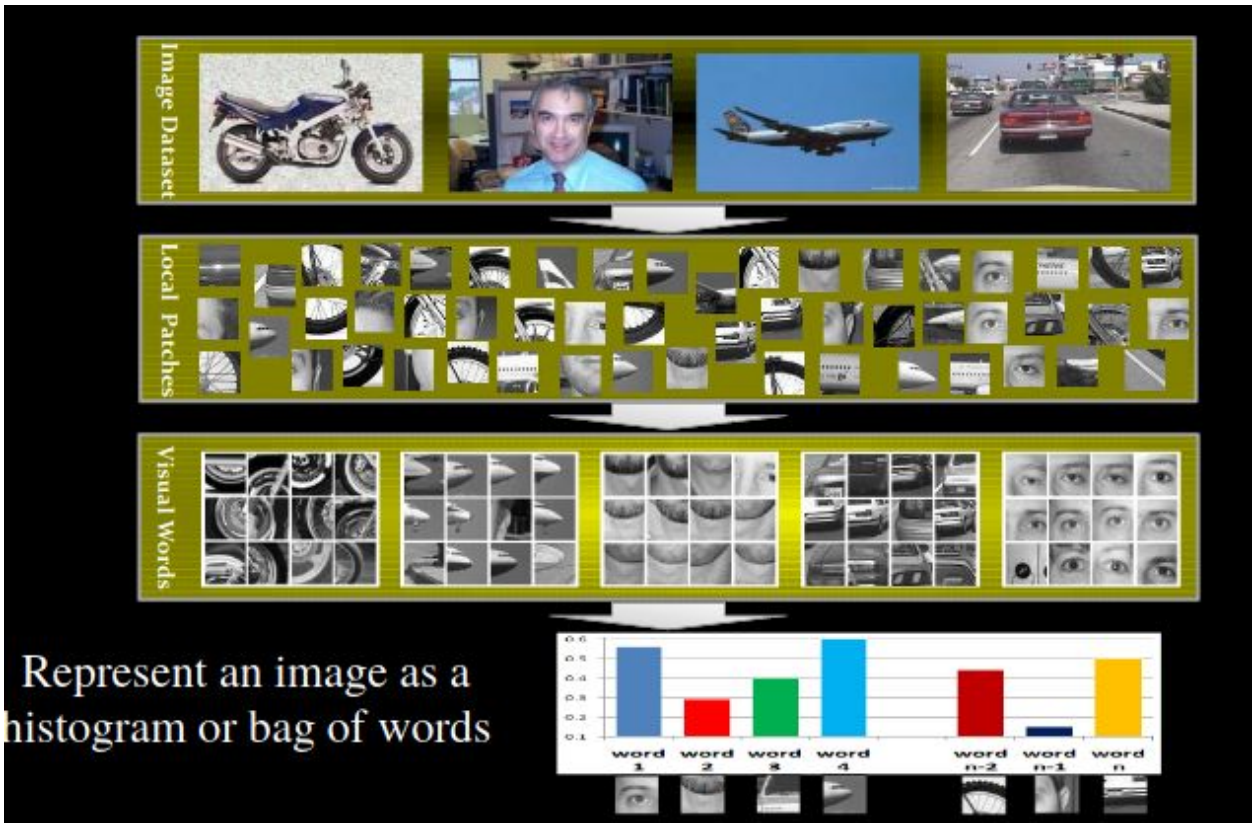


Figure 8: Graphical Representation of BoW for Image data

II. Approach and Implementation:

A. Image Matching:

1. The implementation is done using OpenCV - C++.
2. Extract key points, descriptors of each image.
3. Mark the keypoints on the image for visualization.
4. Sort the keypoints based on the size and get the maximum scale keypoint of the Husky_3 image.
5. Compare the distance between all the descriptors of Husky_1 and max-keypoint descriptor. We consider the descriptor with least distance as match.
6. We use the same approach to match all the keypoints pair of images.

B. Bag of words:

1. The implementation is done in OpenCV-C++ and Matlab.
2. We extracted the keypoints and descriptors using opencv.
3. Apply k-mean algorithm to find 8 centroids for the descriptors of all the images.
4. These centroids serve as visual words of the image.
5. We plot the histogram of each image by assigning each descriptor to one of the centroids by finding euclidean distance.
6. Find the histogram intersection and match other images against Husky_3 .

III. Results and Discussions:

a) Salient Point Descriptors: (Question and Answers)

i) **Answer:** SIFT is robust to a broad range of **Affine transformations** like Rotation, Scaling and Shear. It is also invariant to change in **3D orientation (view point), Variation in illumination and also addition of noise.** SIFT achieved this by transforming a given image to **"scale invariant coordinates".**

In addition to this SIFT also provides invariance to **Translation and Affine 3D Projection.**

ii) **Answer:** SIFT achieved this by transforming a given image to **"scale invariant coordinates"** Below points explains the technicalities briefly.

1) **Scale Invariance:** SIFT achieves scale invariance by constructing scale space. In each octave 5 responses of Image with Gaussian kernels with different k values are taken and LoG is constructed. Scaling down the image by 2, 4, 8, 16 gives us different octaves. Building scale space helps us to achieve scale invariance. This is shown in Figure 2.

2) **Rotation Invariance:** Rotational invariance is achieved in **Keypoint orientation assignment** step. After finding the key-points, we assign orientation to each key-point which represents the direction of each key-point. This is done by taking a 16*16 neighborhoods and finding the histogram of gradient magnitude and orientation of each pixel at every 10degrees. This gives 36 bins. Then we assign the orientation of the maximum gradient magnitude.

iii) **Answer:** The final 128-D **feature vector is normalized to unit length** in which we can avoid the change in pixel contrast where each pixel is multiplied by a constant. If we normalize the feature vector, then the features will become illumination invariance. For the illumination changes

in which a constant is added to each pixel will anyway not affect the gradient magnitude because we take differences while calculating gradient values.

Also, we store the pixel intensity values of the neighborhood to achieve lighting variations.

iv) **Answer:** Difference of Gaussian (DoG) gives a close approximation to Laplacian of Gaussian. Since LoG is computationally intensive, SIFT uses Difference of Gaussian as an approximation of LoG.

v) **Answer:** Keypoint descriptor feature vector size is 128-D in original paper. Figure 5 shows the construction of a 128D feature vector. The 16*16 neighborhood is divided into 4*4 subgroups and we compute gradient in each blocks, so total features are **4*4*8 = 128.**

b) Image Matching: Figure 9-12 shows the key points of Huky_1, Husky_2, Husky_3 and Puppy_1 images respectively.

Figure 9: Key Points of Husky_1

Figure 10: Key Points of Husky_2

Figure 11: Key Points of Husky_3

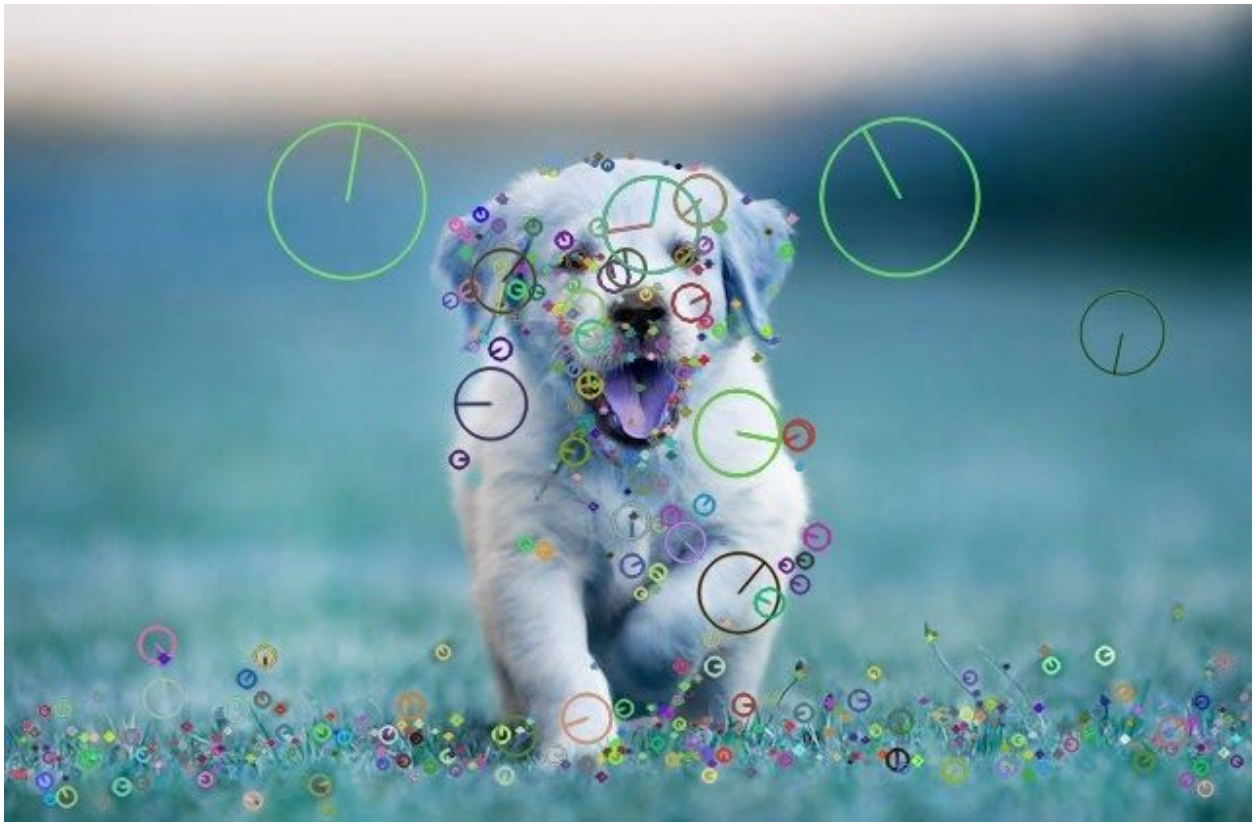


Figure 12: Key Points of Puppy_1

The keypoints are extracted using OpenCV C++. We have got **524, 884, 373 and 585** key points respectively for Husky_1, Husky_2, Husky_3 and Puppy_1. We can see the key points with different sizes and also there are orientations marked in the key points.

The OpenCV function returns the key points which have size, coordinates and orientation as properties. We extract the sizes of the keypoints using the key point property and extract the key point with the highest scale of Husky_3 image. Below are the details of the keypoint with the highest scale in the Husky_3 image.

Key point: (319.1381, 131.299)
Size of the Keypoint: 70.338066
Index: 198

Figure 13 shows the Key point with the highest scale of Husky_3 Image

Figure 13: Key point with the highest scale.

After obtaining the keypoint with the highest scale, we keep the descriptor of that key point as a template. Now we find the Euclidean distance between all the descriptors of Husky_1 image and the template descriptor. We match the descriptor which is nearest to the template descriptor of Husky_3 image.

Figure 14 shows the corresponding keypoint in Husky_1 Image. The keypoint is mapped in the ground (**a tiny purple mark on the ground**). The matching coordinate is (**320.7897, 380.9800**)

Figure 14: Corresponding keypoint in Husky_1 Image

Discussion: As we can observe that we have not got the match properly. This is because the minimum distance in most of the cases may not be the exact match. In the original paper, a ratio test is done in order to remove these false negative points. Generally, a ratio of 0.8 is fixed and we compare the minimum distance and 2nd minimum distance. If the distance is close enough we eliminate the match. By doing this we can avoid 95% of mismatches by losing around 5% of correct matches. Since, we used the nearest neighbor algorithm we are getting mismatches.

Comments on orientation of the key-points: SIFT does orientation assignment inorder to achieve rotation invariance. We do this by plotting the histogram of 16*16 neighborhood. We further subdivide this into 4*4 blocks. Then we calculate Gradient Magnitude and direction of each pixel and we create a 36 bin histogram for each key point by scanning each of 10 degrees. This is explained in detail in Abstract and Motivation. Figure 5 shows the representation of orientation assignment.

Image Matching between Images:

1. **Husky_1 and Husky_3:** Figure 15-16 shows the matching between Husky_1 and Husky_3 with 0.7 and 0.8 thresholds respectively.

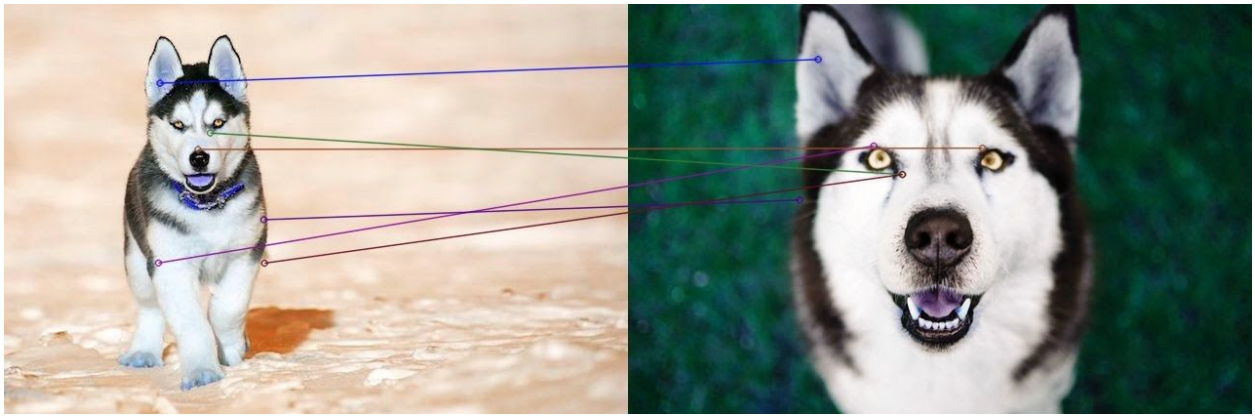


Figure 15: Husky1 and Husky3 (0.7 Threshold)

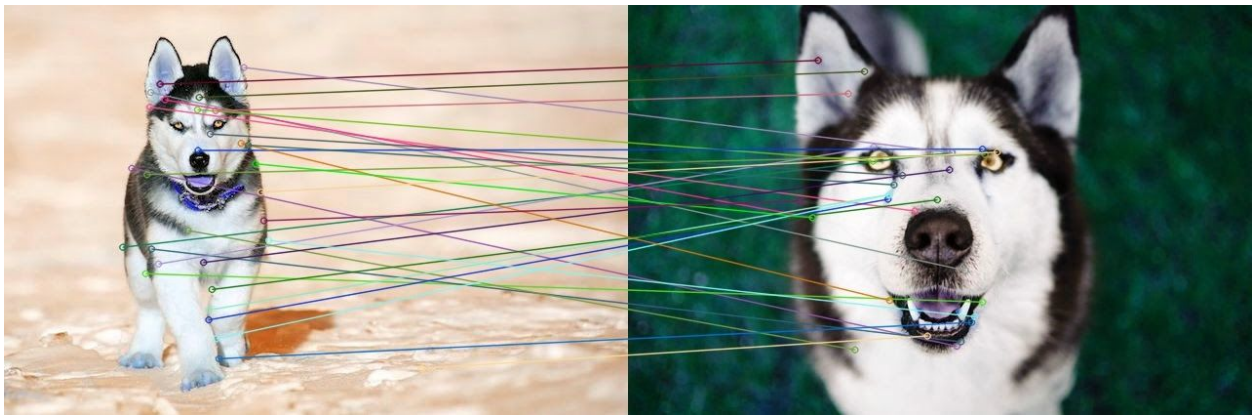


Figure 16: Husky1 and Husky3 (0.8 Threshold)

2. **Husky_3 and Husky_2:** Figure 17-18 shows the matching between Husky_3 and Husky_2 with 0.7 and 0.8 thresholds respectively.

Figure 17: Husky3 and Husky2 (0.7 Threshold)

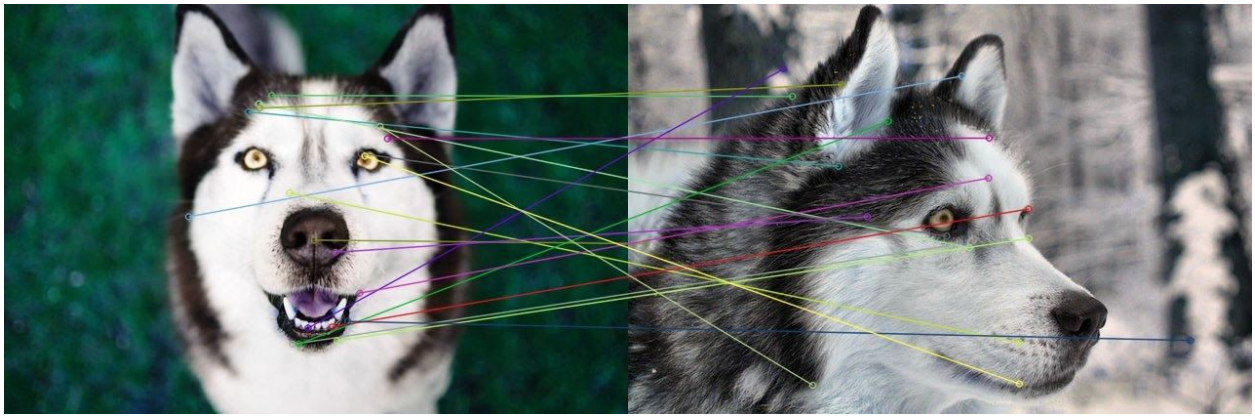


Figure 18: Husky3 and Husky2 (0.8 Threshold)

3. **Husky_3 and Puppy_1:** Figure 19-20 shows the matching between Husky_3 and Puppy_1 with 0.7 and 0.8 thresholds respectively.

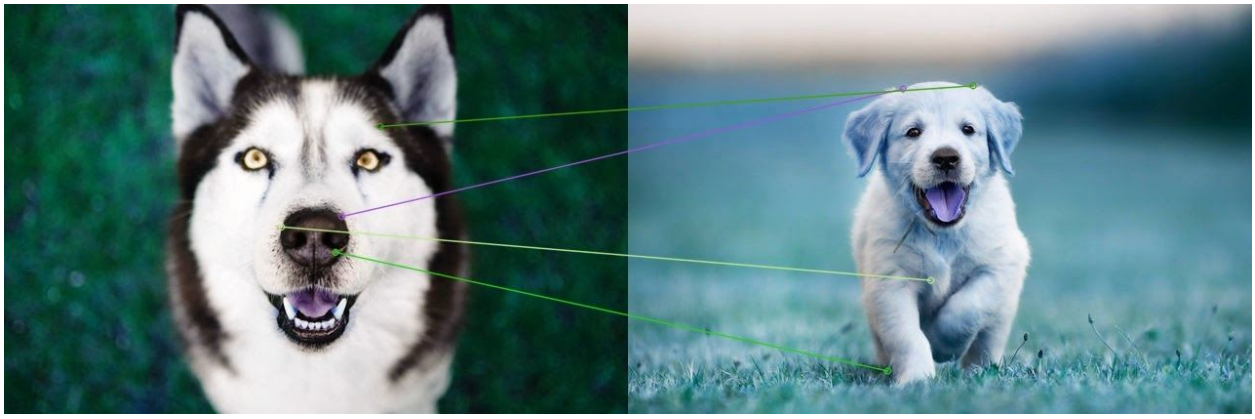


Figure 19: Husky3 and Puppy1 (0.7 Threshold)

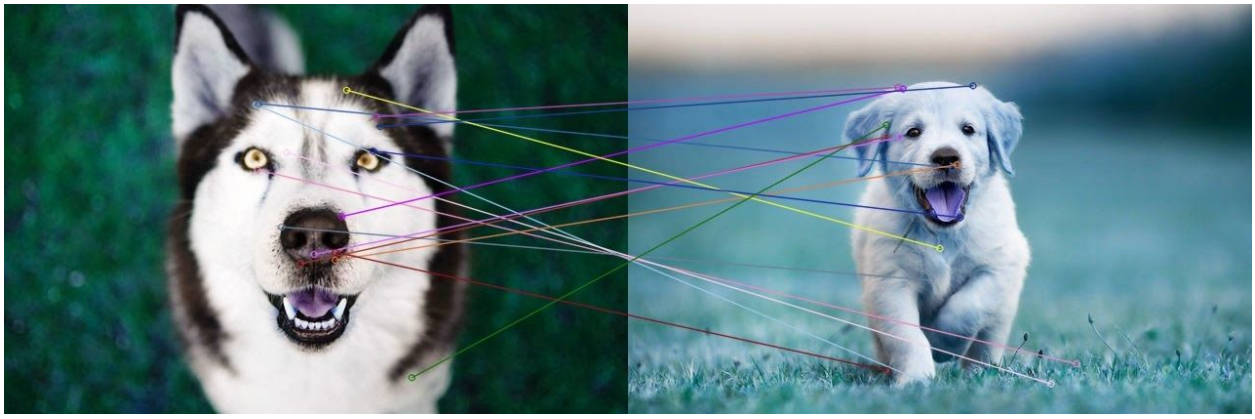


Figure 20: Husky3 and Puppy1 (0.8 Threshold)

4. **Husky1 and Puppy1:** Figure 21-22 shows the matching between Husky_1 and Puppy_1 with 0.7 and 0.8 thresholds respectively.

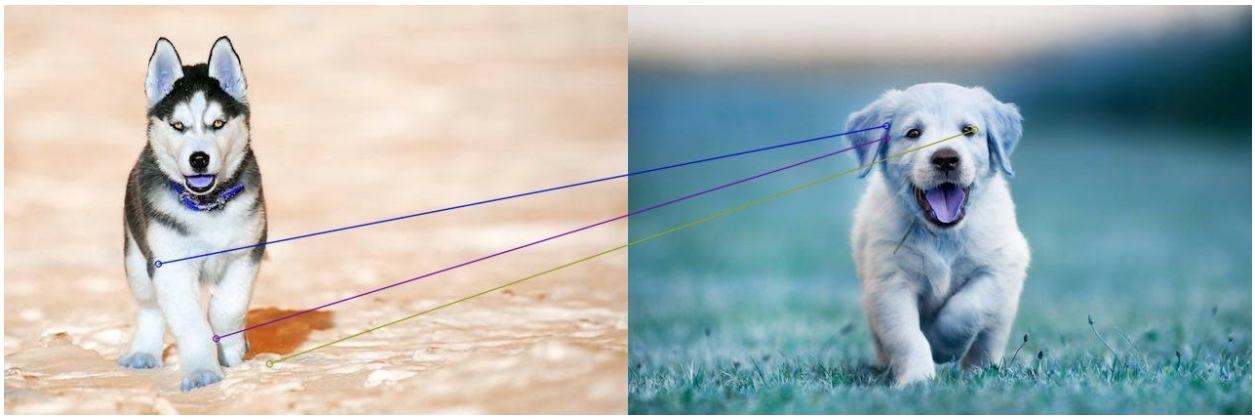


Figure 21: Husky1 and Puppy1 (0.7 Threshold)

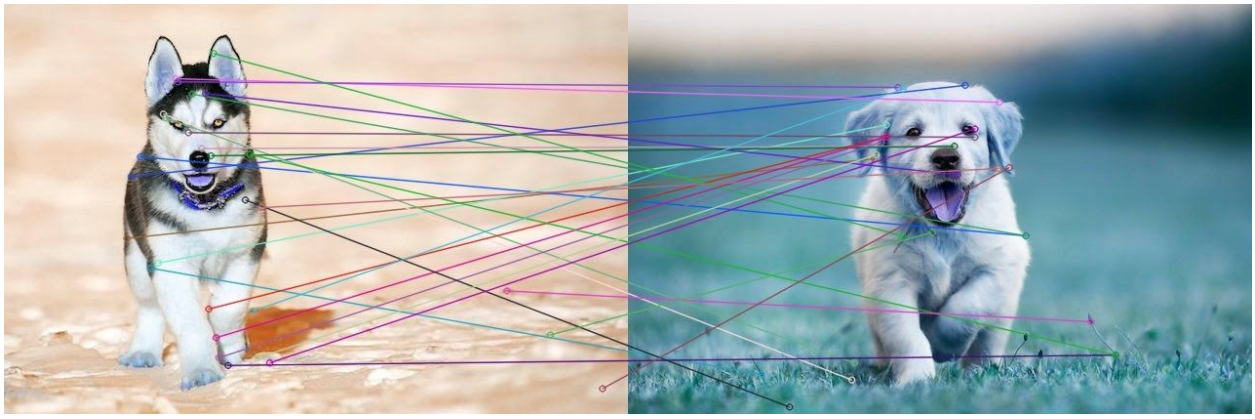


Figure 22: Husky1 and Puppy1 (0.7 Threshold)

Discussions: The above image matching is obtained for two thresholds. As we can see, we are getting more key points for thresholding 0.8 and less for 0.7.

Between Husky_1 and Husky_3 we are getting few correct matches and few mismatches. As we can see the matches are the same for key points near the eyes and nose. But the key points near the ear in Husku_3 are being matched to the body parts of Husky_1. Although SIFT is scale invariant and ideally it should get correct matches even if the images are rescaled. But we are getting few mismatches here because the texture of the husky_1 near its legs and texture of husky_3 near face borders are nearly the same. The dark to light transitions on the body is creating mismatches in this case. But if we carefully count the number of correct matches, we can say matches are more than mismatches.

Between Husky_3 and Husky_2, we actually got only one match for the 0.7 threshold, and even that is not matched properly. For the 0.8 threshold, we are getting few points but we can say most of the points are getting mismatched. The reason for this is Husky_2 is an affine transformed image where the camera viewpoint is being changed. Although in the original paper the author has been mentioned that SIFT is invariant to affine transformations, we still see many mismatches. This is because the change in camera view is considered to be a nonlinear affine transformation. SIFT works well for transformations like scaling, translation and rotation.

Between Husky3 and Puppy1, we can see a good number of matches for 0.8 threshold. Although the image itself is different, we got matches because the eyes, nose, ears are the common features among both the images. Similarly for Husky1 and Puppy1, we got a considerably good number of matches. The reason is obvious that we get good matches for the images which are different but we got not so good results for the images where the camera view point is different.

C. **Bag of Words:** Using the descriptors of all the 4 images we implemented k-means algorithm to get 8 cluster centroids for each image. We then plot the histogram of each image by comparing the euclidean distance between each descriptor and the centroids. But this only gives the histogram of each image with respect to its visual words (centroids). Since, we want to compare the images using the histogram we plotted the histogram of each image with respect to the centroid of Husky_3. By comparing the histograms we find the intersection and find the better matches.

Figure 23 shows the histogram of Husky_3 image.

Figure 24-26 shows the histogram of each image with respect to centroid of the Husky_3 image.

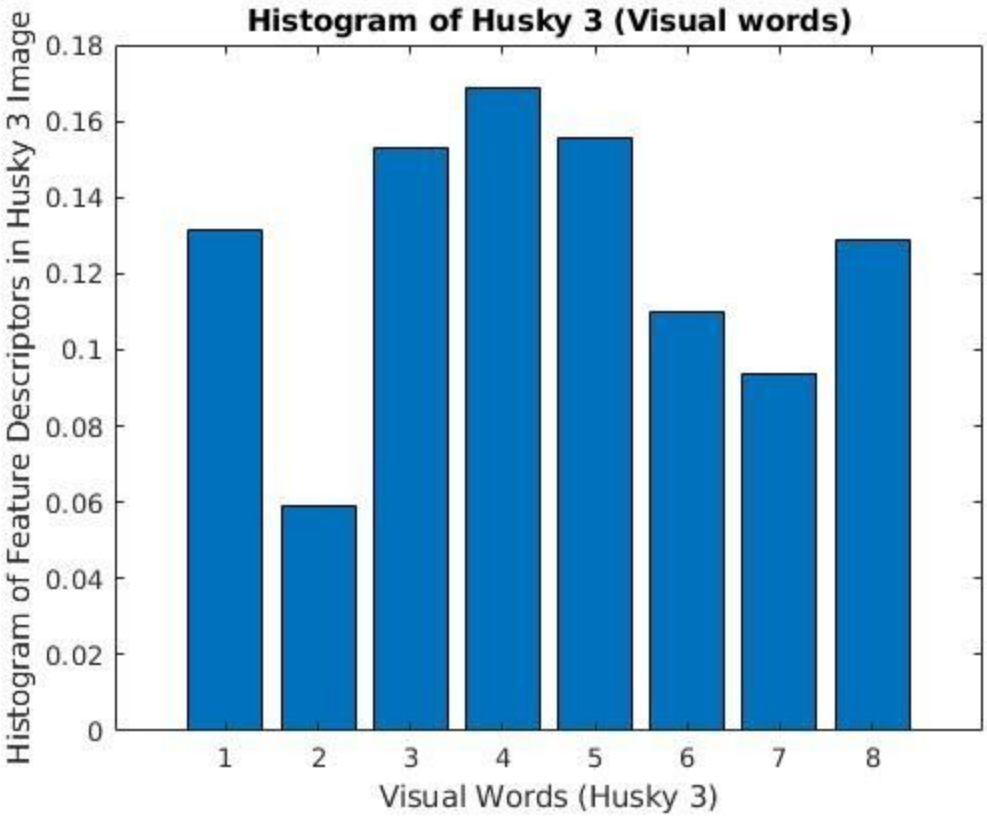


Figure 23: Histogram of Husky_3

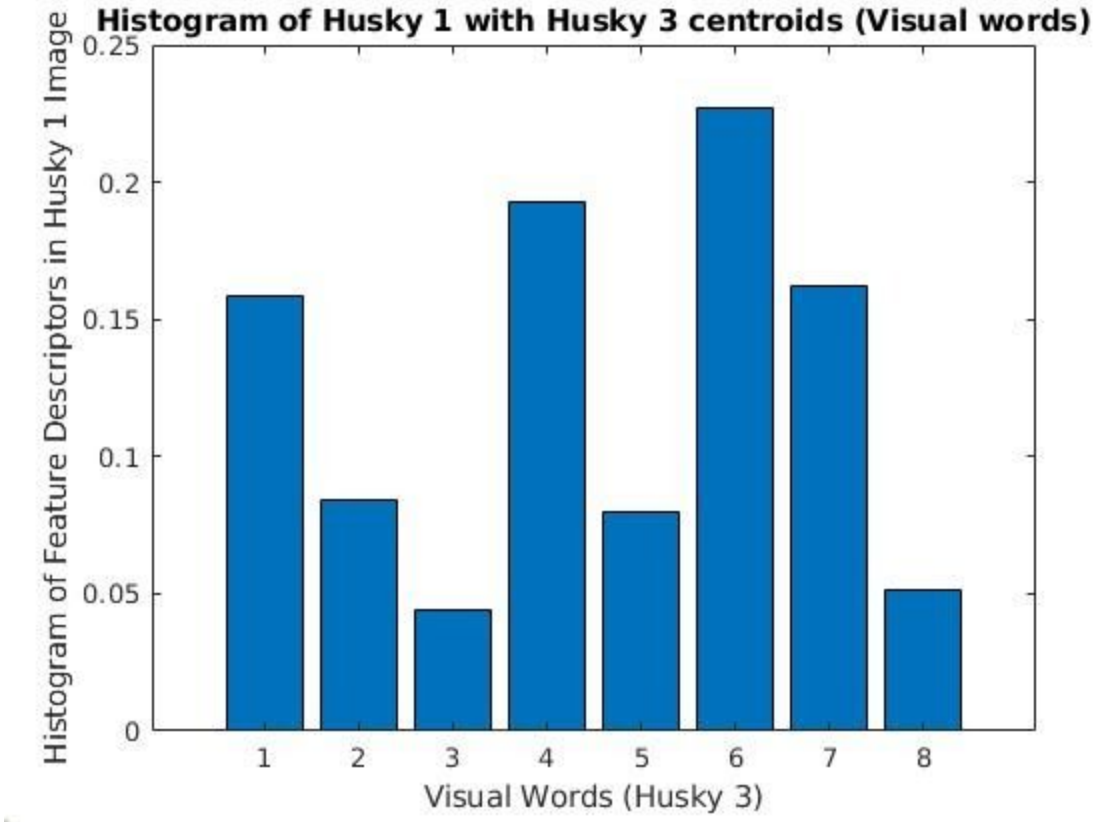


Figure 24: Intersection of Husky_1 with Husky_3

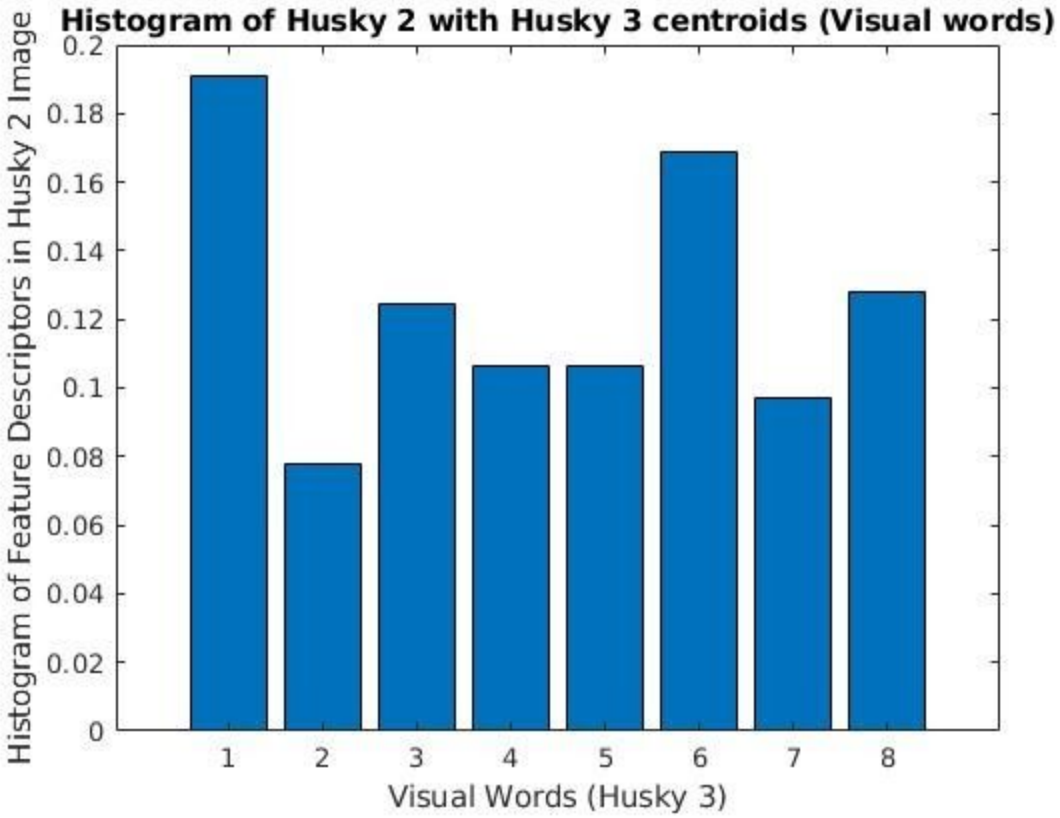


Figure 25: Intersection of Husky_2 with Husky_3

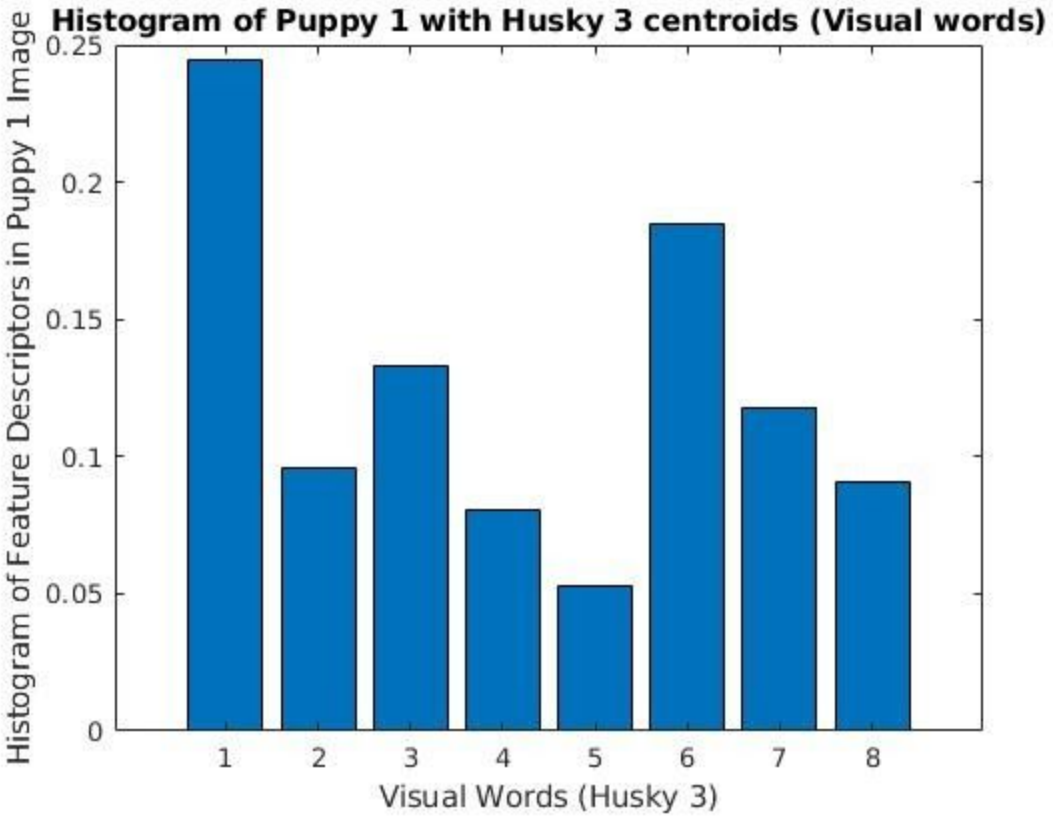


Figure 26: Intersection of Puppy_1 with Husky_3

Discussion: First of all, we are matching the image using histogram intersection method. But since we are using the k-means algorithm the labels will be different each time. So we sort the centroids with respect to the distance between the centroids of Husky_3. Then we plot the histogram of each image with respect to the code book (centroids) of the Husky_3 image. This seems fair comparison in contrary to text data,

We keep the histogram of Husky_3 as a template and compare the intersections of other images with Husky_3. As we can see from the histograms, we can conclude that the maximum intersection with Husky_3 is present with Husky_1 and the least intersection is present with Husky_2. Puppy_1 is in between these two.

The reason is quite similar to the previous one. Since Husky_2 image is taken from different viewpoints and hence it has least intersection with Husky_3. But Husky_1 is just a scaled image and we got the maximum intersection.Puppy_1

also got a similar intersection because it has the same physical features such as eyes, nose and ears.

So the matching order with Husky_3 is,

Husky_1 > Puppy_1> Husky_2

References:

1. http://clopinet.com/fextract-book/IntroFS.pdf
2. https://www.crcv.ucf.edu/wp-content/uploads/2019/03/Lecture-6-SIFT.pdf
3. https://pdfs.semanticscholar.org/f58b/22395f9585c3da65bbc948c67eed3377f701.pdf
4. https://www.crcv.ucf.edu/wp-content/uploads/2019/03/CAP5415_Fall2012_Lecture-17-BagOfWords.pdf
5. http://www.macs.hw.ac.uk/texturelab/files/publications/phds_mscs/MJC/ch5.pdf