EE569: Digital Image Processing

Homework #4

Yogesh Sanat Gajjar USC ID: 5476153636

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

1. TEXTURE ANALYSIS AND SEGMENTATION

Till the previous homework, we saw that geometric image modification and Morphological operation on images help us to transform the image by modifying the position of the pixel value in tan image and to give a final image a different perceptive keeping the pixel value as it is. In this homework we utilize the information of the image to generate essential features which helps in the classification and segmentation of the image.

In this homework, we deal with image texture and leverage its importance. Image textures are an essential part if the image which carries useful information about the image and help us in differentiating different objects in the image. Texture analysis in Digital Image Processing is often referred to as the segmentation of the regions of the image by their texture content. Texture analysis attempts to describe intuitive qualities of a textured image described by its roughness, smoothness, silkiness, or bumpiness as a function of the spatial variation in pixel intensities. In this sense, the roughness or bumpiness refers to variations in the gray levels.

Texture analysis has numerous applications in the field of biomedical and remote sensing. It is used in various applications, such as remote sensing, aerial photography, automated inspection, and medical image processing. The other application of texture analysis can be used to find the texture boundaries, called texture segmentation. Texture analysis can be advantageous in certain applications where objects in an image are characterized by their texture than by intensity, and traditional segmentation techniques cannot be used effectively.[1]

Textures are a quasi-periodic pattern where the texture is not smooth or contains random noise. Classification problem deals with finding the texture class given the texture whereas segmentation deals with given the texture and finding the boundary.

The three applications of Texture Analysis are :

1. Texture Classification – Classifies the images into different classes
2. Texture Segmentation – Segments an image into different sections
3. Texture Synthesis – Generate new samples from the given existing class

In this question, we perform classification and segmentation of the textured images and then we classify the images into different clusters using the algorithm called K-Means. Later, we use built-in algorithms such as Random Forest and Support Vector Machines for Supervised Learning.

1.1. TEXTURE CLASSIFICATION – FEATURE EXTRACTION

1.1.1. ABSTRACT AND MOTIVATION

Texture classification deals with the classifying of an image into a class. Image classification is a process in computer vision that deals with identifying the type of category of the image by looking at the features of it. For example, an image classification algorithm may be designed to identify if the image contains a dog or cat. This is a hard and difficult task and involves deep research. After the successful results of Deep Neural Networks, it has imparted a significant impact in the other visual problems such as segmentation and general object detection.

To perform texture classification, we use the law filters. The law filters designed the filter bank of size 5x5 to identify textures from it. Since the good amount of size is needed to determine the texture, 5x5 carries a lot of information.

Law filters can be represented as two different ways. These are subspace representation and frequency decomposition and frequency band representation. We use law filters as the representations. The law filters from 1 to 5 are represented into 25 1D filters after performing the tensor product. In the 1D filters each unit is called a law filter. In different types of images ranging from smooth to rough, the low frequency bands have more energy than the others. But for similar textures high frequency have more energy than low frequency. We then perform a dimension reduction procedure such as PCA to find the dimensions which generates the maximum energy.

The feature vectors or energy vectors are classified into textures using some classification algorithm like k-means clustering. Depending on the features, each image is classified based on the clusters in which it falls. We initialize the clusters by its centroid and the points classified around it belongs to that.

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

Figure 1. 1D Kernel for 5x5 Laws Filter

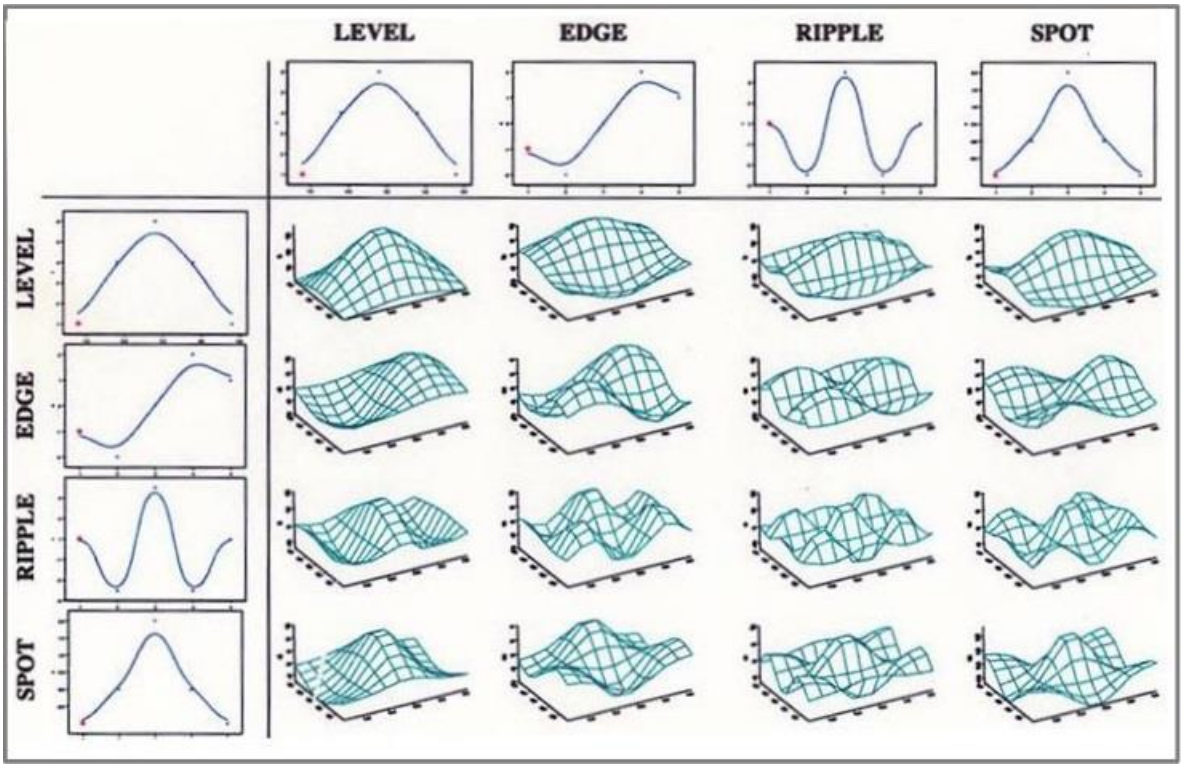


Figure 2. 5x5 Laws Filter representation

1.1.2. APPROACH AND PROCEDURE

The task of this question is to perform

- Feature Extraction
- Feature Averaging
- Feature Reduction

The texture classification problem to perform above three functions is implemented using the following procedure :

1. **<u>Reduce Illumination Effects</u>**
   This step essentially reduces the illumination from the images. Basically, we try to remove the DC components from the image by subtracting the mean. Here, the mean is calculated by averaging all the pixel values of the image, Here the window size is the entire image. The calculated mean is subtracted from each pixel of the image. Also, reducing the illumination prevents any high energy to dominate the energy vector calculation. Since we have 36 training images with us, we perform the illumination reduction for all of them.

The function can be summarized through equation as :

$$Image\ (Mean\ subtracted) = Input\ image - Pixel\ mean$$

2. **Filtering using the Laws Filters**

In the above section, we discussed the types of laws filters and the tensor product generates a 2d array containing 5x5 elements. These laws filters have a significance and they are used for performing a specific function.

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]

Table 1. Laws 1d Kernel

Here, L5 is used to average the values of the pixels. E5 detects edges in the image. S5 detects spots, W5 detects waves and R5 detects ripple.

Here, we use all the 5 Laws 1D kernels to make 25 5x5 Laws filters. The tensor product converts the 1D kernel to 5x5 filters, making the filter bank of 25 laws filter. To produce a filtering output in all the images, we convolve the train images with the 25 laws filters after extending the boundary in the input image. After multiplying the pixels with the law's filters, we obtain local energies from the images.

3. **Feature Vectors Averaging**

After the second step, we find the feature vectors by multiplying the laws filters with every pixel of the image. Its time to average the features leading to a 25D feature vector for each image. We calculate the averaged feature vectors; we square the all the pixel intensity of an image and divide it with the total number of pixels in the image. We perform this for all the 36 training images. To illustrate this better, a equation to calculate the averaged feature vectors is given as

$$Feature\ Averaging = \frac{1}{Total\ pixels} \sum_{x=1,y=1}^{row,cols} f(x,y)^2$$

where $f(x,y)$ = Pixel intensities

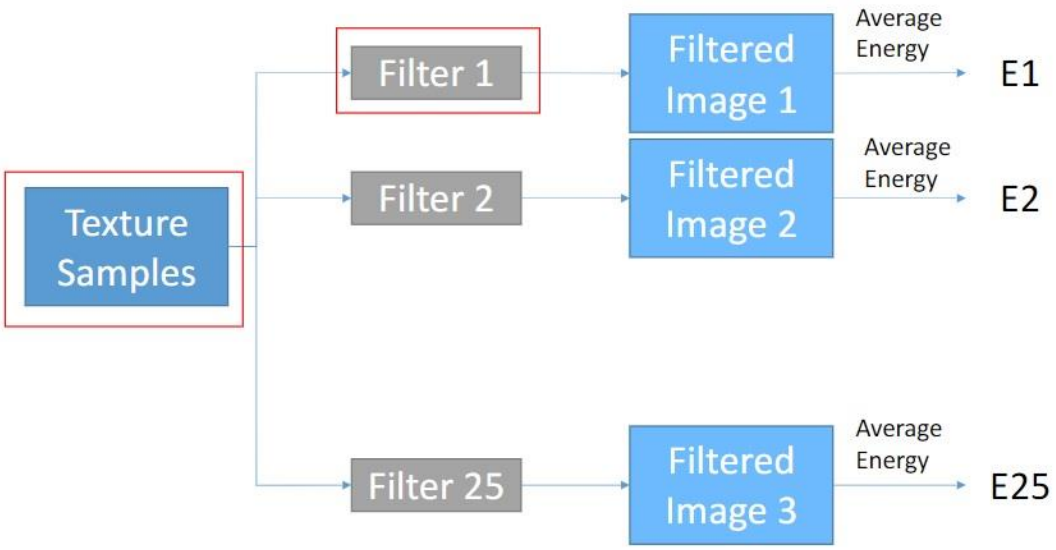


Figure 3. Pipeline for calculating the average energy

4. **<u>Convert the 25D to 15D feature vectors</u>**

   Here, after generating a 25D feature vector for all images, we convert the 25D feature vectors to 15D feature vectors. The idea behind converting is to reduce the number of features by avoiding redundancy. After generating a 25 5x5 laws filter, we see that few of them correspond to same values and removing the pair with direct replacement or averaging solves the redundancy problem.

   The 15D feature vector generated essentially contains the L5L5, L5E5, E5S5, E5E5, L5S5, E5W5, S5S5, L5W5, E5R5, W5W5, L5R5, S5W5, R5R5, W5R5, S5R5.

   Now the feature set consists of 36 samples x 15 features.

5. **<u>Evaluate the discriminant power of the features</u>**

   We evaluate the discriminant power by calculating the inter-class variance and the intra-class variance. Here, the inter-class variance is the variance among different categories or among different classes. Here we have 4 different classes. On the other hand, intra-class variance is the variance among different features for a certain class or category.

Then the total corrected sum of squares is

$$\sum_i \sum_j (y_{ij} - y_{\bullet\bullet})^2.$$

The intra-class sum of squares is

$$\sum_i \sum_j (y_{ij} - \bar{y}_{i\bullet})^2).$$

The inter-class sum of squares is

$$\sum_i \sum_j (\bar{y}_{i\bullet} - \bar{y}_{\bullet\bullet})^2 = \sum_i (n_i(\bar{y}_{i\bullet} - \bar{y}_{\bullet\bullet})^2)$$

Figure 4. Calculation of inter and intra class variance

Based on the inter-class variance and inter-class variance, we can prove the feature having good discriminant power if its intra-class variance decreases and its inter-class variance increases.

6. Feature reduction using PCA

Principal Component Analysis is a technique to reduce the dimensions of the feature set so that the data can be represented on a 2D or 3D graph without losing lot of essential information of the features. Here, the principal components are the eigen vectors of the covariance matrix which contains the essential information. The pixel with the maximum energy becomes the first principal component followed by the second largest energy. This method is to appropriate certain texture feature as the energy distribution in the 15D is not uniform.

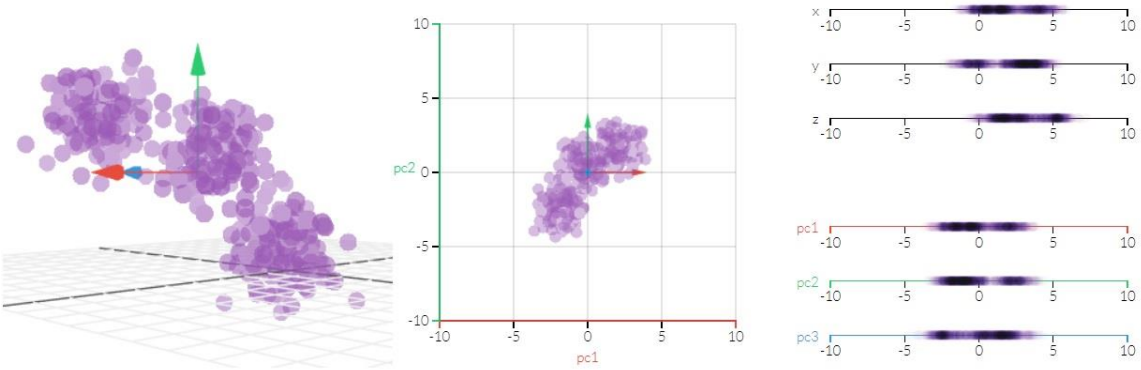


Figure 5. Principal Component Analysis representing 3D to 2D to 1D

I implemented my own PCA. The steps I followed to implement the PCA is as follows :

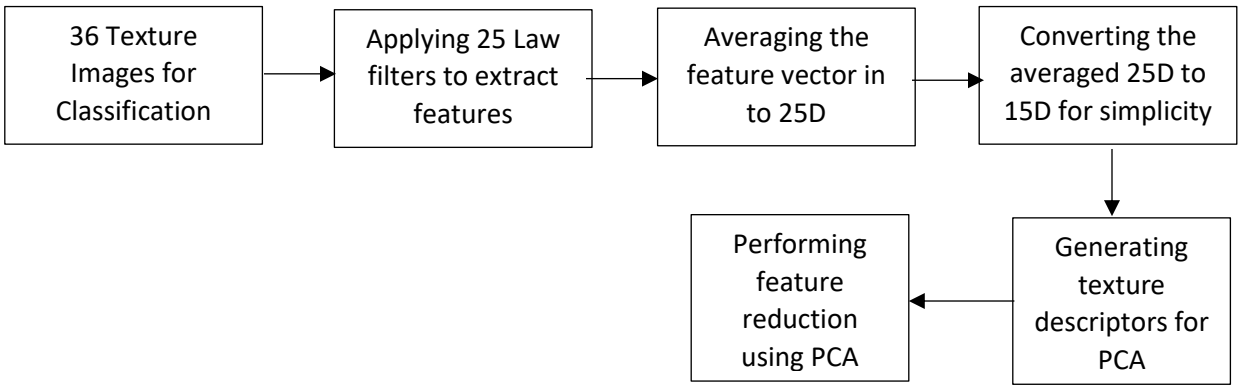
a. Considering the feature vector as X_{nxm} where n is the number of samples and m is the feature dimensions obtained from the above steps, we go ahead and calculate the mean of all the 15 dimensions.

b. The purpose of calculating the mean is to subtract the mean from the feature matrix and make it a zero-mean matrix.
The mean calculated can be shown using an equation

$$mean(m_X)_j = \sum_{i=1}^{n} \frac{X_{i,j}}{n}$$

c. I subtracted the mean for all the features to generate a zero mean data matrix Y_{nxm} where n is the number of images and m is the number of features. In out case we have 36 x 15.
d. We know that the principal components can be calculated using two ways, one using eigen value decomposition and other is using SVD. Since we have non-square matrix, I preferred using the SVD.
I computed the SVD of the Y matrix which generated three different matrix called U, S, and V. U and V are the left and right eigen vectors and S is the matrix containing singular values.
e. The reduced matrix Y_R is obtained by multiplying the original feature matrix X with the first three columns of the V matrix as those are the three principal directions.

The entire pipeline of the texture classification can be summarized using a flow diagram.



Algorithm for Texture Feature Extraction

1. Read the content of all the images in the directory using stringstream. Generate a name of the file and store it in a vector of strings. Also specify the height and width of the image.
2. For every image name present in the image string vector, remove the illumination by subtracting the mean from all the pixels of the image. Please refer the equation to reduce the illumination mentioned above.
3. Calculate the tensor product of the 5 laws filter and making it 25 different 5x5 laws filter
4. Extend the boundary by 2 pixels and perform convolution of the DC removed image with the 25 5x5 laws filters to obtain 25 filtered outputs for every image. Store it in a separate image 2D array.

5. Perform feature averaging by squaring all the pixels of the image and divide it with the total number of pixels. Refer the equation mentioned in the above section.
6. Reduce the 25D to 15D vector by averaging the filter pairs and replacing it to make it 15D vector. Also calculate the discriminant power of the feature using the technique mentioned above.
7. The final feature vector obtained will be of the dimensions 36 x 15 where 36 is the number of images and 15 is the dimensions.

Algorithm to perform PCA

1. The feature vector obtained from the feature extraction is used to calculate the mean of all the 15D feature vectors.
2. Convert the original feature vector and mean values of all the 15 columns into a text file to be read by MATLAB.
3. Read the file in MATLAB and convert the file into two different data matrices with dimensions 36x15 and 1x15.
4. Calculate the zero-mean matrix by subtracting the mean from all the columns of the feature vectors. Name the zero-mean matrix as Y.
5. Perform SVD of Y matrix using the function SVD in MATLAB with 'econ' feature. This generates three different matrices called U, S, V.
6. Extract the first three columns of the V matrix as the first three columns define the best three principal directions.
7. Finally, multiply the original feature vector with the first three columns of V making it a reduced feature vector.

1.1.3. EXPERIMENTAL RESULTS

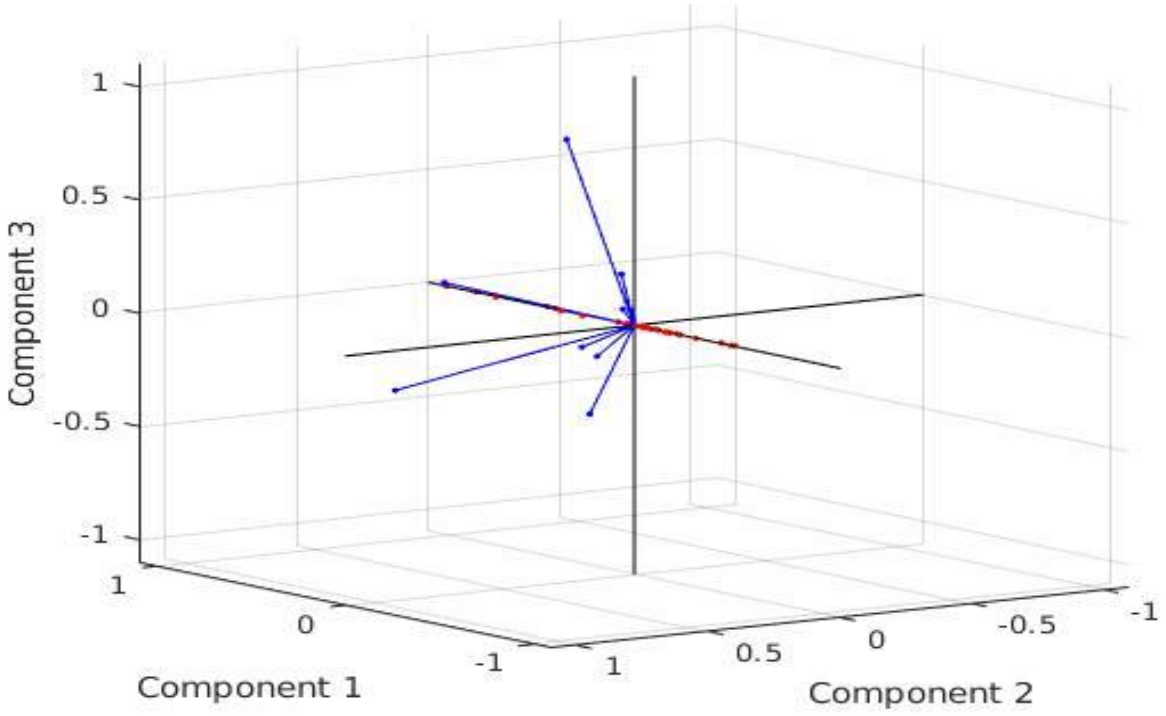


Figure 6 – Bi plot for the training data images for 3 principal components

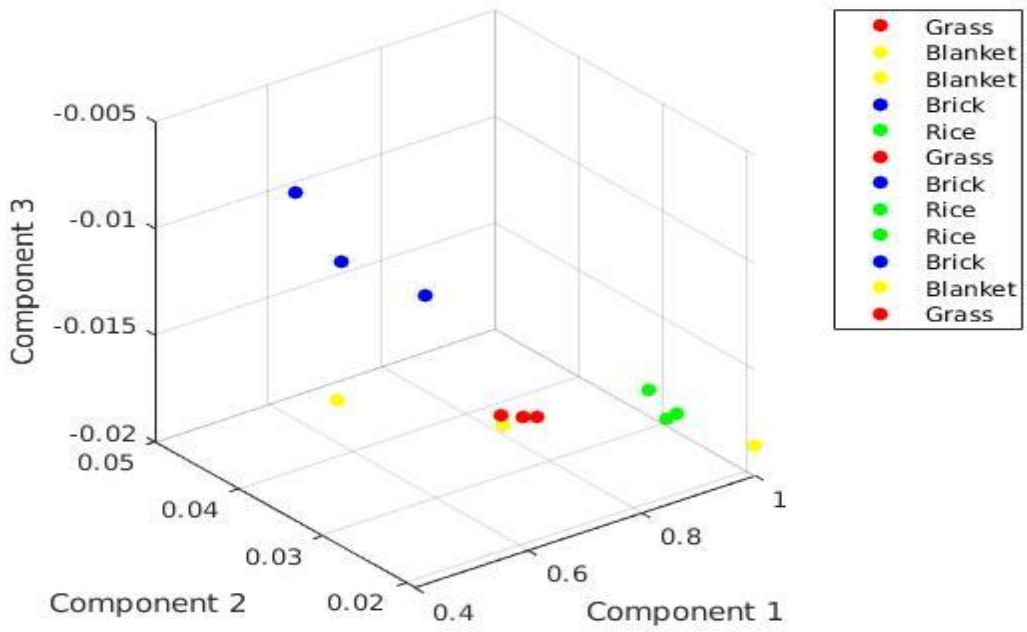


Figure 7 – Scatter plot for representing the test images forming different clusters after PCA

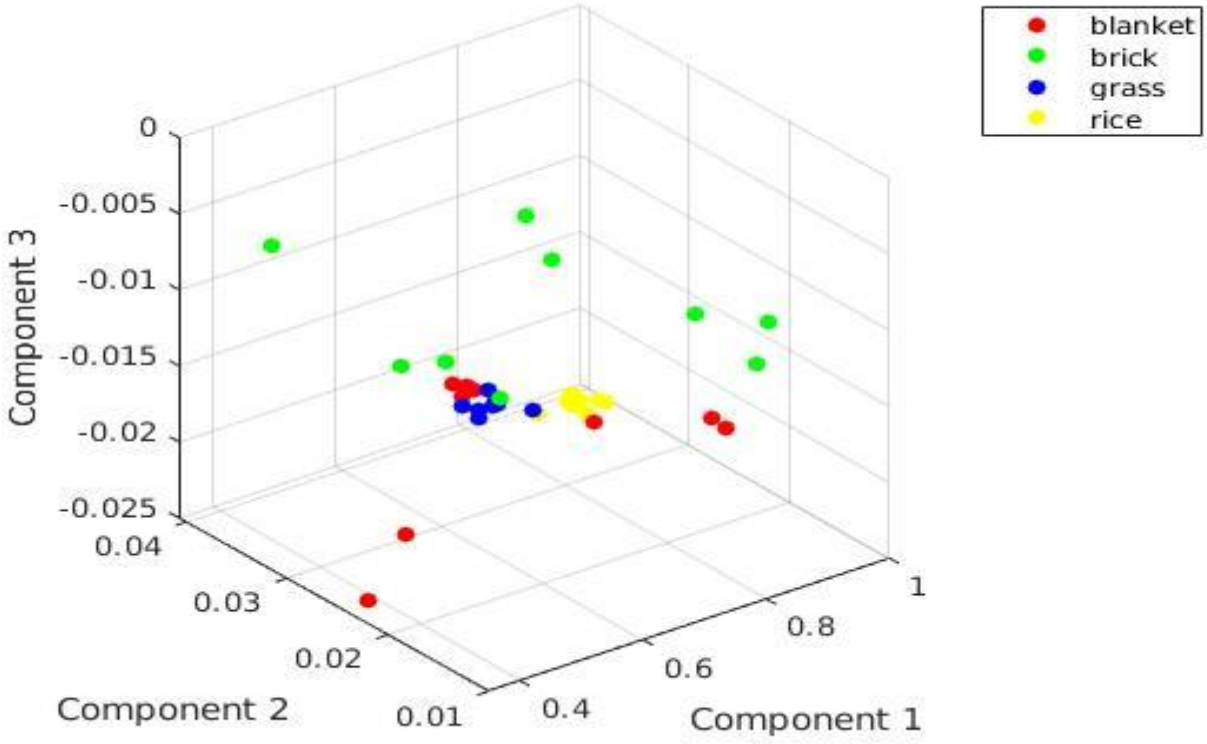


Figure 8 – Scatter plot for representing the train images forming different clusters after PCA

1.1.4. DISCUSSION

In this question, we perform the extraction of feature vector for all individual training images by convolving the image with laws filter and later calculating the averaged energy for all the 25-D feature vector.

Discriminant Power – Discriminant power here means the ability that can aid in discriminating all the image training samples. This essentially means the ability of the feature vector to help clustering the similar ones and bifurcating the different classes. A good feature is the one which effectively helps in clustering. By using the method of finding the inter-class variance and intra-class variance, I found the E5E5 as the one providing the best discriminant power and L5E5 giving the lowest one. I found the ration between the inter and intra class variance to conclude my answers.

Later we convert the 25-D feature vector to 15-D after detecting the discriminant power of each filter kernel. Later, we perform feature reduction using Principal Component Analysis. We reduce the feature vector dimensions from 15 to 3 and visualize the results. My observations after looking at the scatter plots in the result section is as follows,

1. For training images,
    a. We see from the all 36 points represent 36 images in the 3D space. The reason for dimension reduction using PCA is to visualize the features of the images in 2D or 3D space so that conclusions can be made.
    b. The features of rice and grass are clustered around each of the other points in of their own class. This implies that any classification techniques performed on this dataset containing features will correctly classify them. Moreover, being in a group also infers that unsupervised learning K- Means can also cluster them and classify them into correct classes. I know the fact that these are training images and we already know their ground truth labels so no classification is required. But the above comments are generic to the type of scatter plot generated.
    c. The features of blanket and brick are not clustered together and, they are scattered vary far. If we rotate the -D plot, we can see that the points of few points of blanket are present near rice and brick and few points of brick are present very far. They are outliers for this image feature set. If classified, these features of blanket and brick can lead to misclassification. Even when performed unsupervised K-Means clustering, the algorithm might mis – classify those points. I know the fact that these are training images and we already know their ground truth labels so no classification is required. But the above comments are generic to the type of scatter plot generated.

2. For test images,
    a. I calculated the reduced 3-D feature vector for test images by multiplying the feature matrix of size (12 x 15) with the principal directions generated by the train images. We see that all the 12 points represent 12 test images in the 3D space. The reason for dimension reduction using PCA is to visualize the features of the images in 2D or 3D space so that conclusions can be made.
    b. The features of rice and grass are clustered around each of the other points in of their own class. This implies that any classification techniques performed on this dataset

containing features will correctly classify them. Moreover, being in a group also infers that unsupervised learning K- Means can also cluster them and classify them into correct classes.

c. The features of blanket and brick are not clustered together and, they are scattered vary far. If we rotate the -D plot, we can see that the points of few points of blanket are present near rice and brick and few points of brick are present very far. They are outliers for this image feature set. If classified, these features of blanket and brick can lead to misclassification. Even when performed unsupervised K-Means clustering, the algorithm might mis – classify those points.

1.2. ADVANCE TEXTURE CLASSIFICATION – CLASSIFIER EXPLORE

1.2.1. ABSTRACT AND MOTIVATION

Texture classification deals with the classifying of an image into a class. Image classification is a process in computer vision that deals with identifying the type of category of the image by looking at the features of it. For example, an image classification algorithm may be designed to identify if the image contains a dog or cat. This is a hard and difficult task and involves deep research. After the successful results of Deep Neural Networks, it has imparted a significant impact in the other visual problems such as segmentation and general object detection.

In this section we will explore two different types of classification methods. One which doesn't require image labels and the other which requires image labels to predict the labels of the unseen data. The first one is called unsupervised learning where the image are classified based on the similarity in their features. K-Means classifier is an example of such learning which doesn't require an image label but rather groups based on the closeness of each feature with each other. The supervised learning utilizes the labels of the training images to predict the class of the test images.

1.2.2. APPROACH AND METHODOLOGY

For the unsupervised section of the classifier, I implemented my own K-Means algorithm for classifying the texture images into individual class. The algorithm is as follows and remains the same for 15D as well as the 3D feature vector :

1. **<u>Initialize the clusters centroid</u>**
   This is the basic first step while implementing the K-Mean algorithm. We fix an initial point in the space for a specific class. In our case we have 4 different classes and thus we initialize the centroid with 4 different points. As this step is iterative, the process repeats until any further change in centroid is obtained.

2. **Calculate the distance**

Calculating the distance helps to determine the similarity of two elements in the space. There are quite a few distance metrics that can be used. The known distance metrics are called as Euclidian distance, Manhattan distance, Mahalanobis distance and Chebyshev distance. The main reason behind calculating the distance is to determine how far the points are from the centroids. The near the point is, the point gets classified into the class the centroid is representing.

In this classification, I used Euclidian distance to calculate the difference between the pixel value of the feature vector with the centroid. The equation representing the Euclidian distance is as follows :

$$Dist_{xy} = \sqrt{\sum_{k=1}^{m} (x_{ik} - y_{ik})^2}$$

Figure 5. Distance Formula

3. **Find the minimum distance**

The next step is to calculate the minimum distance form the distance calculated in the above step. We focus on the distance which is the minimum because it confirms that the point which is closest to the centroid can be classified as a class like the one the centroid has. Every time we calculate the distance, we compare the minimum distance with the distance calculated in the above step and increment the class label if found matching.

$$L_i = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmin}} \; d(E_i, C_k)$$

Figure 6. Minimum distance formula

4. **Update the centroid**

Every time after the iteration, we update the centroid. To calculate the updated centroid, we add all the pixels intensities of the image classified to a class and divide it with the total number of classified points. This step ensures that the centroid is moving and not all points have been classified. Some points in the space are still unclassified and thus at every change of the pixels, we keep on updating the new centroid. The point at which the centroid is equal to the new centroid, we stop the algorithm.

The formula to calculate the new centroid is given as :

$$C_k = \frac{\sum_{i=1}^{n} l(L_i=k) E_i}{\sum_{i=1}^{n} l(L_i=k)}, where\ l(L_i = k) = \begin{cases} 1 & if\ L_i = k \\ 0 & if\ L_i \neq k \end{cases}$$

Algorithm for K-Means algorithm implementation

1. Initialize the centroid. Here, centroid is a 1D array which stores values of 4 class. I place the 36x15D or 36x3D feature vector in the feature space to initialize the centroids.
2. Calculate the Euclidean distance of each image with 4 centroids. The distance is the sum of squares of the difference between the centroid and the feature vector.
3. Find the minimum distance and compare the minimum distance with the distance obtained for a cluster. If found matching, update the label count of that specific class.
4. Update the centroid with a new value. The new centroid value is calculated by calculating the mean of the features classified for that class.
5. Repeat the iteration until 4 centroids do not change over the iterations and remain the same.

For supervised learning method, we use the results of two classifiers i.e. Random Forest and Support Vector Machines. These two algorithms or models utilizes the fact that we have class labels with us. It also requires training the models with correct ground truth, which makes the model predict the unseen data. Let's investigate the principle of working behind each model.

1. **Random Forest [2]**

   Random Forest classifier is an ensemble method which utilizes two or models outputs and accumulates it to give a prediction. It is made out large number of individual decision trees that operate as an ensemble. It is because a large number of relatively uncorrelated trees(models) operating together outperforms all of the individual models(trees). We know that decision trees generate high variance as they are sensitive to the data they are trained on and even a small change in the training set can result into significantly different results or predictions. Random Forest on the other end utilizes Bagging principle.

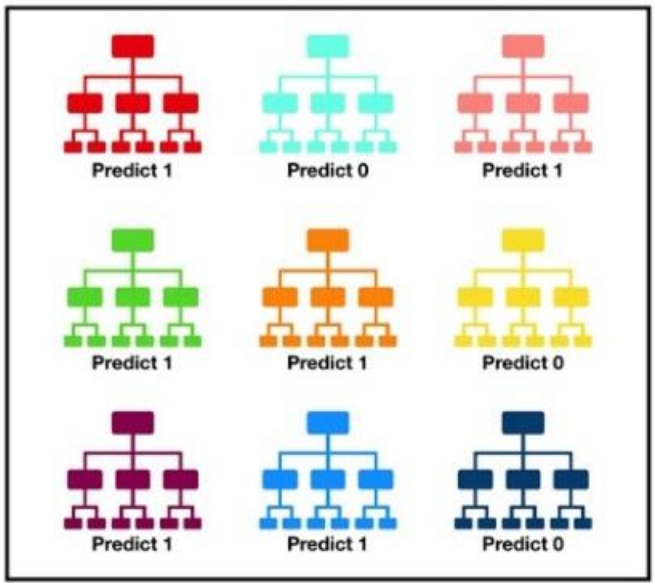


Figure 9. Decision Trees in Random Forest

Bagging (Bootstrap Aggregating) is a technique in which random forest allows every individual tree to randomly sample from the main data set with replacement, thus generating randomness in every subset taken for training the trees. We make sure the training set length remains the same and each tree is trained on a fixed length of data. Just that the data will be shuffled and picked with replacement form the main dataset.

Also, Random Forest uses another technique called Feature Randomness. When the tree splits a node, we usually tend to consider all possible feature and pick the one which gives the maximum separation between observations. This method gives us the best prediction on the unseen data. But, in feature randomness, each tree in the random forest picks form the random subset of the features. This forces more variation amongst the trees in the random forest model and generates lower correlation across trees.

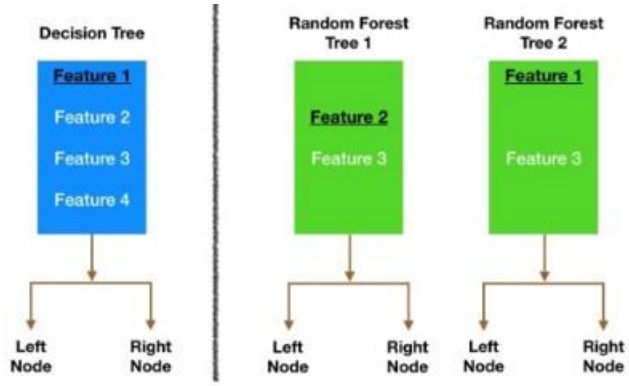


Figure 10. Node splitting based on random subset of feature for each tree.

2. Support Vector Machines [3]

Support Vector Machines of SVM is a supervised learning algorithm which is works on predicting a class or works as a regressor. Here, we will discuss the working of the SVM as a classifier in linear and non-linear space. In simple words, SVM is a model which works by finding the decision boundary to separate the different classes and maximize the margin. The classification is performed by finding the hyper-plane that differentiates the two classes perfectly. Let's go through some terminology for linear case.

Hyper-Plane is an (n minus 1)D subspace for an n-dimensional space. For example, in a 2D space, the hyperplane becomes 1D and we obtain a line. For 3-D space, the hyperplane is just a 2D space and obtain a plane.

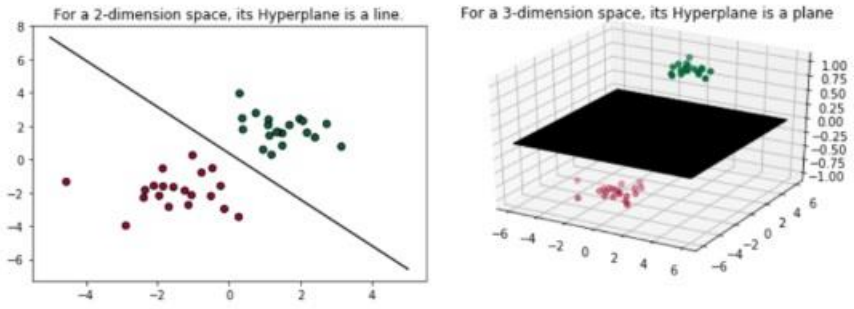


Figure 11. Hyperplane in SVM

Separating Hyperplane is a plane which divides the dataset into two different half or generating a separation between then in order to classify. In SVM, the separating hyperplanes also depends on the margin which ensures that the separating hyperplane is present at a distance which gives the maximum separability.

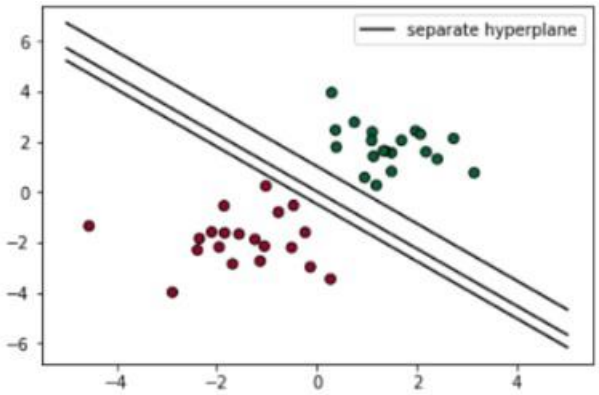


Figure 12. Separating Hyperplane

A Margin Is basically a distance between either side of the separating hyperplane. It ensures that the classification is maximized. The margin can be reduced for to allow no misclassification and can also be reduced to allow little misclassification to avoid overfitting the model.

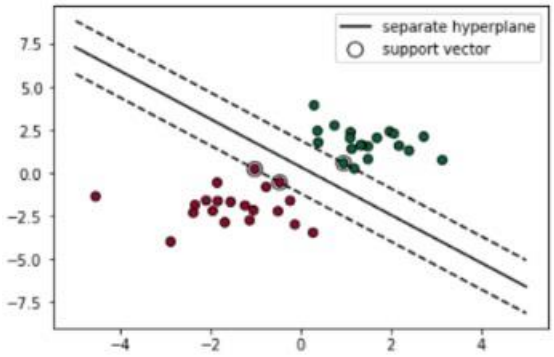


Figure 13. Margins and Support Vectors

In the linear separable cases, SVM tries to find the hyperplane that maximizes the margin such that we receive the minimum separability. But usually we don't find such cases in real life when the data is linearly separable. For non-linear case, SVM introduces two different concepts called soft margin and kernel trick. Soft Margin allows misclassification and tries to balance the trade-off between finding the line that maximizes the margin and reduces the misclassification. The kernel trick convert the existing features by applying some transformations to create a entirely new features. These new features help in classifying the non-linear datasets.

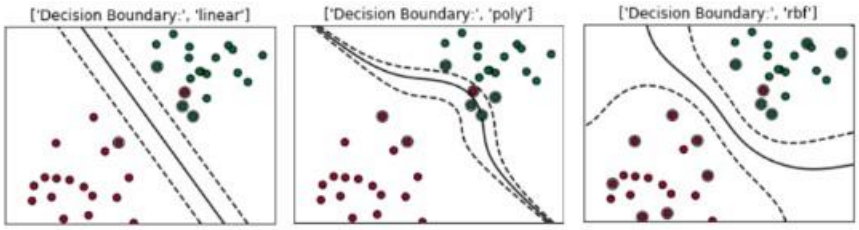


Figure 14. Kernel Trick for Non-Linear Data

1.2.3. EXPERIMENTAL RESULTS

The results for the K-Mean clustering are :

```
Test PCA mode:
----------------- Test Image with 15D features ----------------

Test Image              Class Label             Class Name              Ground Truth
1                       1                       Grass                   Grass
2                       2                       Blanket                 Blanket
3                       4                       Rice                    Blanket
4                       3                       Brick                   Brick
5                       4                       Rice                    Rice
6                       1                       Grass                   Grass
7                       3                       Brick                   Brick
8                       4                       Rice                    Rice
9                       4                       Rice                    Rice
10                      3                       Brick                   Brick
11                      3                       Brick                   Blanket
12                      1                       Grass                   Grass


Final Class Labels:
Images belong Grass: 1,6,12, and Count for each label is: 3
Images belong to Blanket: 2, and Count for each label is: 1
Images belong to Brick: 4,7,10,11, and Count for each label is: 4
Images belong to Rice: 3,5,8,9, and Count for each label is: 4

Misclassification error rate : 16.6667%


----------------- Test Image with PCA reduced 3D features ----------------

Test Image              Class Label             Class Name              Ground Truth
1                       1                       Grass                   Grass
2                       2                       Blanket                 Blanket
3                       4                       Rice                    Blanket
4                       3                       Brick                   Brick
5                       4                       Rice                    Rice
6                       1                       Grass                   Grass
7                       3                       Brick                   Brick
8                       4                       Rice                    Rice
9                       2                       Blanket                 Rice
10                      3                       Brick                   Brick
11                      3                       Brick                   Blanket
12                      1                       Grass                   Grass


Final Class Labels:
Images belong Grass: 1,6,12, and Count for each label is: 3
Images belong to Blanket: 2,9, and Count for each label is: 2
Images belong to Brick: 4,7,10,11, and Count for each label is: 4
Images belong to Rice: 3,5,8, and Count for each label is: 3

Misclassification error rate : 25%
```

1.2.4. DISCUSSION

The following are the observation and conclusions on the K-means clustering in the test images using both the 15-D and 3-D.

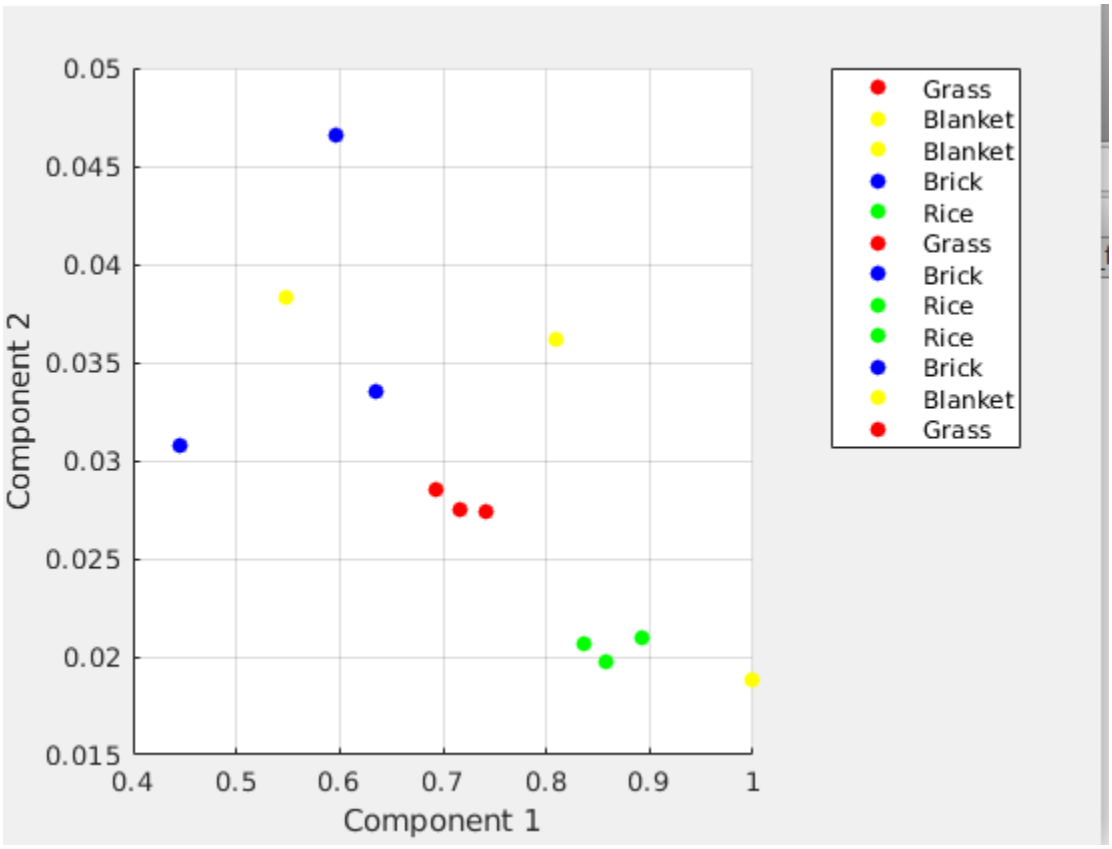


Figure 14. 2D representation of the test PCA features

1. The error rate which I got for misclassification of 12 images into their labels is 16.66%. This means that two points were misclassified. The two points are the "Blankets" which got misclassified into either "rice" or "brick". By looking at the above picture, we see that the point "blanket" is near to the cluster of "rice", which made the algorithm classify it as "rice". Secondly the other "blanket" point is near to cluster of "brick". This also made the algorithm cluster the "brick" including the "blanket" as well. The misclassification of the two "blanket" feature vectors was due to the feature vector falling the regions of the "rice" and "brick" feature points.
2. It was also intuitive by looking at the image of blanket i.e. image 3. It looked more like a different texture and not blanket. This proves that the feature generated from the blanket image was misleading and made it classify wrongly.
3. The classification method used in this example is K-Means clustering. We know that to make this algorithm work, we must initialize it with a centroid. So, the classification of all the pixels in the image depends on the centroid of the clusters. I assigned the clusters centroid randomly. Every time I change the centroid initialization, the output seems to

change a little but no significant change. The misclassification still remains the same and "blanket" image is misclassified.

Image number	Ground Truth (Visual)	K-Means Clustering(15D)	K-Means Clustering(3D)
1	Grass	Grass	Grass
2	Blanket	Blanket	Blanket
3	**Blanket**	**Rice**	**Rice**
4	Brick	Brick	Brick
5	Rice	Rice	Rice
6	Grass	Grass	Grass
7	Brick	Brick	Brick
8	Rice	Rice	Rice
9	Rice	Rice	Rice
10	Brick	Brick	Brick
11	**Blanket**	**Brick**	**Brick**
12	Grass	Grass	Grass

Table 3. Comparison of the Classification

Classification using Random Forest and SVM –

1. As we know we have 4 different classes. The modes Random Forest and Support Vector Machine models work for multi-class classification using two different strategies. They use one-vs-one classifier or the one-vs-rest classifier. In OpenCV, the one-vs-one classifier is used to train the model using the training data and use the same model to work on the testing data.
2. On comparing the performance, the misclassification in case of random Forest is 8.33% i.e. out of all the 12 testing images, the "blanket" image gets misclassified. The performance is better than the K-Means clustering in terms of the misclassification error rate. On the other hand, in SVM, the entire class gets mis-classified. This makes the classifier generate higher error rate in case of multi-class classification.
3. The principle of working in case of random forest and SVM is different. The linear SVM used in this case assumes that the boundary is linear, or the data is linearly separable. But we saw the points of "blanket" had been in the other regions near the "rice" and "brick". This made the support vectors misclassify the entire class and increased the misclassification.

1.3. TEXTURE SEGMENTATION

1.3.1. ABSTRACT AND MOTIVATION

The next step is to perform texture segmentation using Law's filter. We saw that these Law's filters work well to classify the textures into different classes using any of the supervised or unsupervised learning methods. But texture segmentation is difficult compared to texture classification as segmenting the texture into different color intensities requires classification of each pixel of the texture. On the other end, it also involves segmentation of the different boundaries of the texture which needs to be identified parallel to the segmentation of the textures. As said, the process includes classification of the individual pixels (different region pixels in the same input image) and not the entire image. Texture segmentation is mainly done to support such applications like object and image recognition, as it segments different objects in the image along the texture. Texture segmentation also utilizes the unsupervised learning to segment the textures. It uses the K-Means clustering which clusters the textures by generating feature vectors and then utilizing the properties for the features to predict different color intensities pertaining to different clusters.

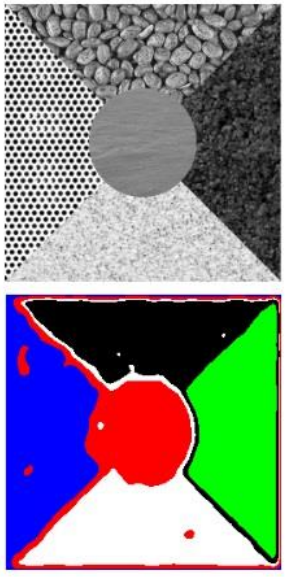


Figure 15. An example of texture segmentation

Dynamic texture segmentation is another different method used for segmentation. The problem of dynamic texture segmentation also lies in the same line of the problem of image texture segmentation. In case of image texture segmentation, we divide the image plane into regions of homogeneous spatial statistics. In case of dynamic image segmentation, we obtain similar partition, but make sure that we get homogeneous spatial and temporal statistics.

1.3.2. APPROACH AND MYTHOLOGY

The texture segmentation problem to perform above three functions is implemented using the following procedure :

1. **Reduce Illumination Effects**

   This step essentially reduces the illumination from the images. Basically, we try to remove the DC components from the image by subtracting the localized mean. Here, the localized mean is calculated by averaging all the pixel values of the image using windowing approach, Here the window size is variable as selected by the user. The calculated mean from the window is subtracted from each pixel of the image. Also, reducing the illumination prevents any high energy to dominate the energy vector calculation. The window size ranges from 0 to 30.

   The function can be summarized through equation as :

   $$Image\ (Mean\ subtracted) = Input\ image - Pixel\ mean\ in\ each\ window\ of\ the\ image$$

2. **Filtering using the Laws Filters**

   In the above section in texture classification, we discussed the types of laws filters and the tensor product generates a 2d array containing 5x5 elements. These laws filters have a significance and they are used for performing a specific function.

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]

Table 2. Laws 1d Kernel

Here, L5 is used to average the values of the pixels. E5 detects edges in the image. S5 detects spots, W5 detects waves and R5 detects ripple.

Here, we use all the 5 Laws 1D kernels to make 25 5x5 Laws filters. The tensor product converts the 1D kernel to 5x5 filters, making the filter bank of 25 laws filter. To produce a filtering output in all the images, we convolve the train images with the 15 laws filters and not with the entire 25 after extending the boundary in the input image. The idea behind converting is to reduce the number of features by avoiding redundancy. After generating

a 25 5x5 laws filter, we see that few of them correspond to same values and removing the pair with direct replacement or averaging solves the redundancy problem.

The 15D feature vector generated essentially contains the L5L5, L5E5, E5S5, E5E5, L5S5, E5W5, S5S5, L5W5, E5R5, W5W5, L5R5, S5W5, R5R5, W5R5, S5R5.

After multiplying the pixels with the law's filters, we obtain local energies from the images.

3. <u>Energy feature computation</u>

After the second step, we find the feature vectors or localized energy by multiplying the laws filters with every pixel of the image. It's time to average the localized energies to calculate the localized energy for all the pixels of the filtered image generated from the above step. We calculate the averaged feature energies; we square the all the pixel intensity of an image and divide it with the total pixels in the window. We perform this for all the 15 filtered outputs. To illustrate this better, an equation to calculate the averaged feature vectors is given as

$$Feature\ Averaging = \frac{1}{(window\ size)^2} \sum_{x=1,y=1}^{win,win} f(x,y)^2$$

where $f(x,y)$ = Pixel intensities

The output after computing the energy features gives an matrix of size 270000 x 15 where the rows are the individual pixels of the image and columns are the 15 law filtered energy vectors.

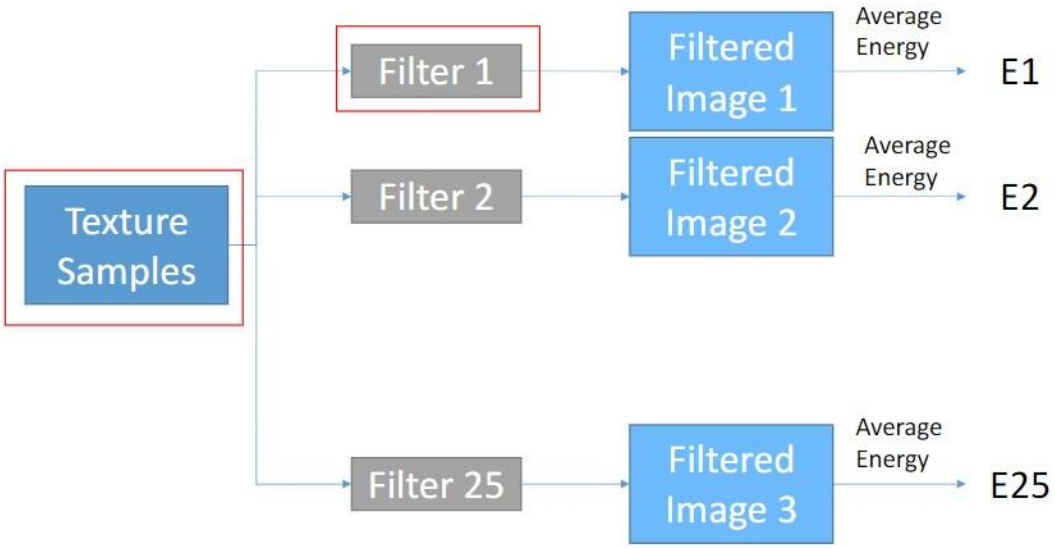


Figure 16. Pipeline for calculating the average energy

4. **Energy feature normalization**

   We see that all the kernels have a zero-mean except for $L5^T L5$. This 2D tensor product is a gaussian distribution as it is used to average the pixels of the image. Therefore, the feature extracted by the filter $L5^T L5$ provides no useful information and can be used to normalize other energy vectors. Thus, it's better to use its energy to normal all other features at each pixel.

5. **Converting 15D to 14D energy vector**

   To perform K-Means clustering to segment the different sections of the texture in an image, we drop the column generated by the tensor product of $L5^T L5$. After normalization, we see that the first column turns into all 1. This column also provides no useful information in the classification and thus we neglect the column and generate 14D energy vector for each pixels of the image.

Now its time to use the K-Means clustering to cluster each pixels of the image to then segment the image. The algorithm for K-Means is implemented on my own. It includes the following steps.

1. **Initialize the clusters centroid**

   This is the basic first step while implementing the K-Mean algorithm. We fix an initial point in the space for a specific class. In our case we have 4 different classes and thus we initialize the centroid with 4 different points. As this step is iterative, the process repeats until any further change in centroid is obtained.

2. **Calculate the distance**

   Calculating the distance helps to determine the similarity of two elements in the space. There are quite a few distance metrics that can be used. The known distance metrics are called as Euclidian distance, Manhattan distance, Mahalanobis distance and Chebyshev distance. The main reason behind calculating the distance is to determine how far the points are from the centroids. The near the point is, the point gets classified into the class the centroid is representing.

   In this classification, I used Euclidian distance to calculate the difference between the pixel value of the feature vector with the centroid. The equation representing the Euclidian distance is as follows :

$$Dist_{xy} = \sqrt{\sum_{k=1}^{m} (x_{ik} - y_{ik})^2}$$

Figure 5. Distance Formula

3. **Find the minimum distance**

   The next step is to calculate the minimum distance form the distance calculated in the above step. We focus on the distance which is the minimum because it confirms that the point which is closest to the centroid can be classified as a class like the one the centroid has. Every time we calculate the distance, we compare the minimum distance with the distance calculated in the above step and increment the class label if found matching.

$$L_i = \underset{k \in \{1,2,\ldots,K\}}{\operatorname{argmin}} \; d(E_i, C_k)$$

Figure 6. Minimum distance formula

4. **Update the centroid**

   Every time after the iteration, we update the centroid. To calculate the updated centroid, we add all the pixels intensities of the image classified to a class and divide it with the total number of classified points. This step ensures that the centroid is moving and not all points have been classified. Some points in the space are still unclassified and thus at every change of the pixels, we keep on updating the new centroid. The point at which the centroid is equal to the new centroid, we stop the algorithm.

   The formula to calculate the new centroid is given as :

$$C_k = \frac{\sum_{i=1}^{n} l(L_i=k) E_i}{\sum_{i=1}^{n} l(L_i=k)}, where\ l(L_i = k) = \begin{cases} 1 & if\ L_i = k \\ 0 & if\ L_i \neq k \end{cases}$$

Algorithm for Texture Segmentation

1. Read the content of the image, height and width using the terminal.
2. Now, to reduce the illumination of the image using windowing approach, declare the window size and generate the padding which his window/2. Based on the window padding, first extend the boundary using the mirroring approach. Now run the window over the image and subtract the mean of the values in the window with the image pixel.
3. Calculate the tensor product of the 5 laws filter and convert the 25D to 15-D by removing the redundant kernels.
4. Perform the convolution of the image with the laws filter to produce 15 filtered output which are basically 15 gray-scale images. Use boundary extension of 2 pixels before convolution.

5. Calculate the localized averaged energy features or vectors using the method described above. The neighborhood taken for calculating the energy vectors change as per the window size. Also perform feature normalization by dividing the entire feature vector matrix with the first element.
6. The feature matrix is ready for the K-Means clustering.

Algorithm for K-Means algorithm implementation

1. Initialize the centroid. Here, centroid is a 1D array which stores values of 6 class. I place the (270000x14) feature vector in the feature space to initialize the centroids.
2. Calculate the Euclidean distance of each image with 6 centroids. The distance is the sum of squares of the difference between the centroid and the feature vector.
3. Find the minimum distance and compare the minimum distance with the distance obtained for a cluster. If found matching, update the label count of that specific class.
4. Update the centroid with a new value. The new centroid value is calculated by calculating the mean of the features classified for that class.
5. Repeat the iteration until 6 centroids do not change over the iterations and remain the same.
6. Assign the 6 labels clustered by the algorithm by 6 different gray levels where each represent one texture. Use the vector to assign the gray levels {0,51,102,153,204,255} to denote six segmented regions in the output of six textures.

1.3.3. EXPERIMENTAL RESULTS

The following are the result of texture segmentation for various windows.

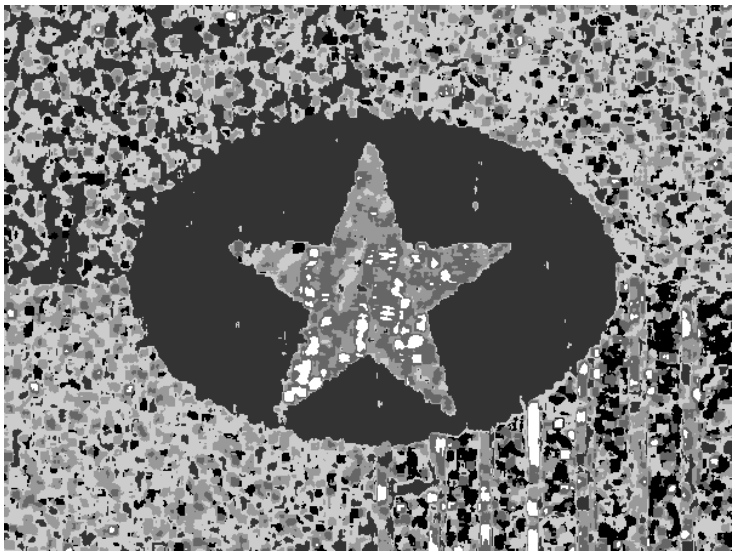


Figure 17. Segmentation for window size = 7

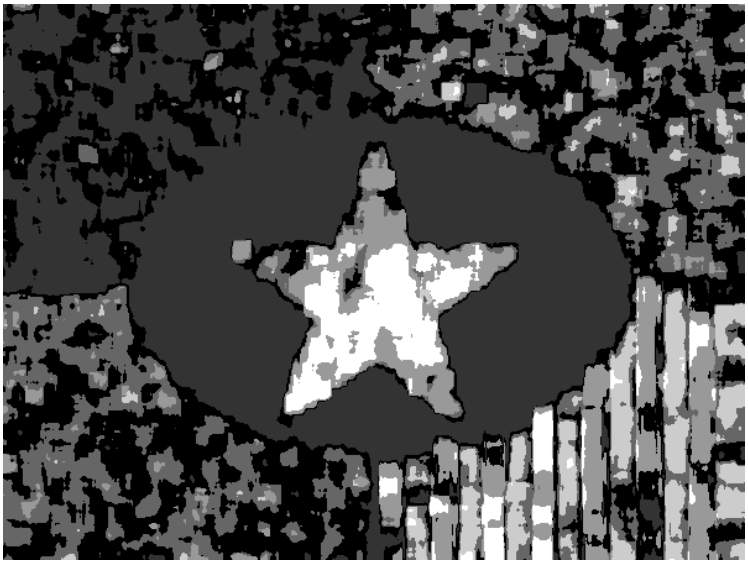


Figure 18. Segmentation for window size = 15

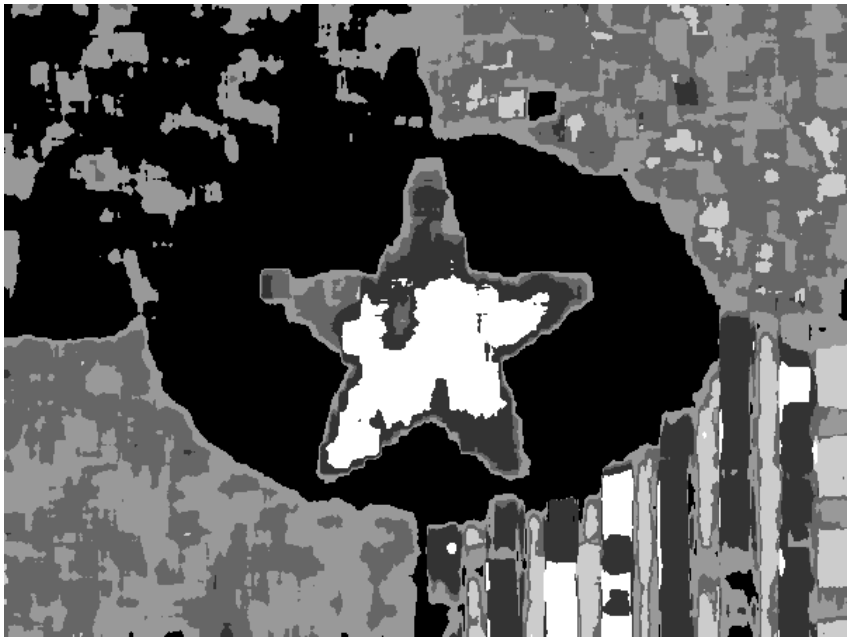


Figure 19. Segmentation for window size = 21

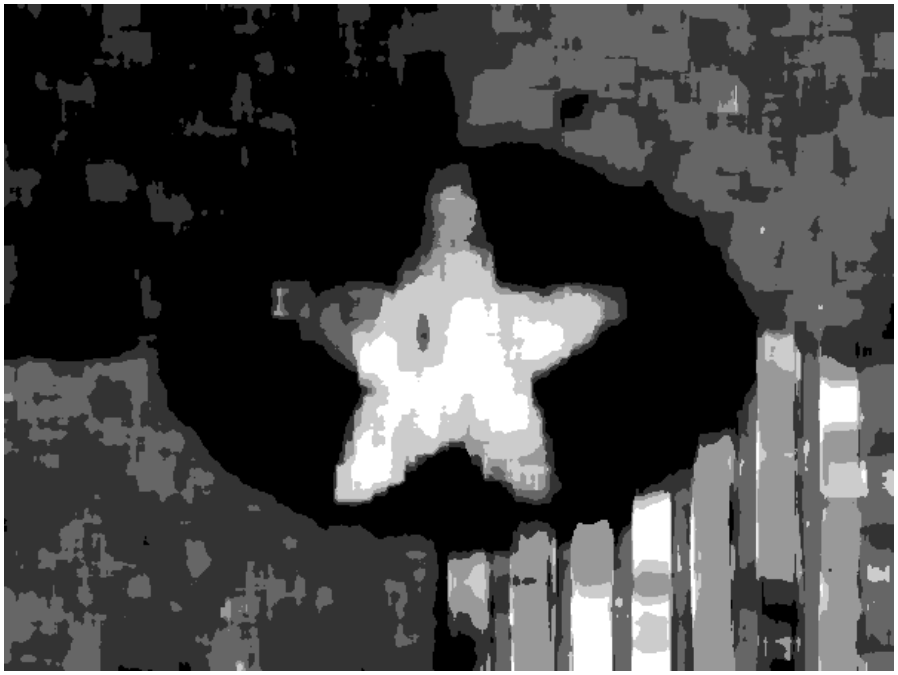


Figure 20. Segmentation for window size = 31

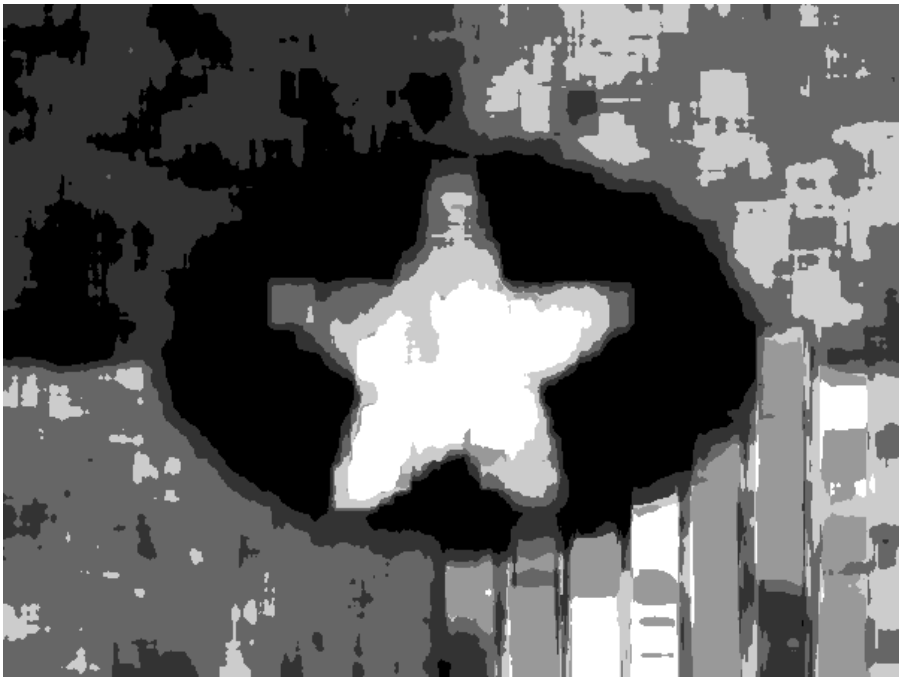


Figure 19. Segmentation for window size = 25

1.3.4. DISCUSSION

I followed the same procedure as mentioned in the above section. Starting with generating the laws filters, convolving the image with the generated laws filters, generating the energy vectors and then normalizing it with $L5^T L5$. I also followed the proper mirroring boundary extension technique work on the segmentation. Once the feature vectors are ready, implemented K-Means to classify each pixel and generate the output. The following are my observation after the final segmentation output for various windows.

1. We see from the image that the composite textured image consists of 6 different regions or textures. From the window size 7, we see that the output is not segmented properly. We can see different spots and regions which tells us that the entire image has just one cluster.
2. When we increase the window size, the output becomes clearer and we can see different segments in the image. We can see the boundary separating all the textures. Though the boundary is not clear, but the output is much better than the above window size of 7.
3. Similarly, as we increase the window size from 15 to 31, we see that the quality of segmentation improves and now we can see less of mixed clusters and more of uniform clusters pertaining to a specific cluster. Though the image seems little blur and unclear in certain regions.

4. This implies that as we increase the window size, the image's localized energy vectors are averaged with large pixels and as a result the final energy feature after normalization produces better segmentation results.
5. Normalization also affects the segmented output. The segmented output changed before normalization and after normalization gave different results. The output with performing the normalization gave good results than the normalized one which you can see in the above section. But overall the results were similar initially for less window size.
6. The classification method used in this example is K-Means clustering. We know that to make this algorithm work, we must initialize it with a centroid. So, the classification of all the pixels in the image depends on the centroid of the clusters and thus the segmentation. I assigned the clusters centroid randomly. Every time I change the centroid initialization, the output seems to change a little but no significant change.

Overall, the segmented output was best for the window size 15. After that the segmented image started to blur and produced output which were satisfactory but not that great.

1.4. ADVANCED TEXTURE SEGMENTATION

1.4.1. ABSTRACT AND MOTIVATION

From the above method for performing the texture segmentation we saw that there are places in the image where the result needs little crisper and uniform rather than small holes of different intensity getting classified. There are few techniques that can be performed which might improve the performance of the image segmentation. These techniques are :

1. Principal Component Analysis
   PCA is a technique to represent or convert the following set of observation into linearly uncorrelated principal components which carries the same information as the previous set of observations. The principal components are selected such that they generate the maximum variance between the values of the observation. The succeeding component in turn is orthogonal to the first component and carries the second largest variance among its values. In short, this method is a well know method for performing dimensionality reduction. Lao, it reduces the higher dimensional data into lower dimensional data.

   It follows two method to perform the dimensionality reduction. First is the eigen-value decomposition of the covariance matrix and the latter is SVD decomposition of data matrix after the normalization of the initial data. Here, in this question, I used the SVD decomposing to implement my own PCA to reduce the dimensionality of the data matrix.

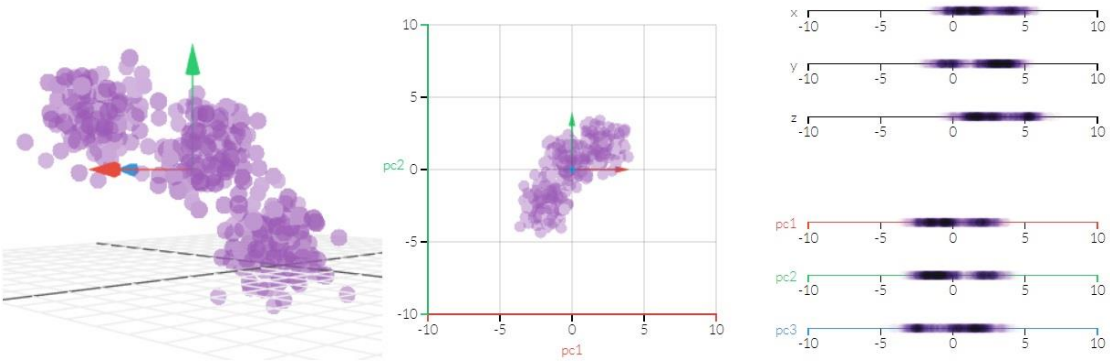


Figure 20 - Principal Component Analysis

2. Post – Processing technique to merge small holes
   We saw in the earlier results that the segmented image contained small holes of different intensities in every clusters. In order to reduce the small holes of varying intensities and to generate segmented output which consists of same gray scale intensity for intensity, we can do a post processing technique using the mode of the pixel intensities. This will ensure that the image remain segmented without any holes.

3. Enhance the boundary of the adjacent regions
   To enhance the boundary of the adjacent regions, we perform a technique called Region Mapping which uses Region Adjacency Graph to produce enhanced region merging for texture segmentation. The reference is from a paper "Enhancement of Region Merging Algorithm for Image Segmentation"[4].

1.4.2. APPROACH AND PROCEDURE

Dimensional Reduction using Principal Component Analysis

I implemented my own PCA. The steps I followed to implement the PCA is as follows :

1. Considering the feature vector as X_{nxm} where n is the number of samples and m is the feature dimensions obtained from the above steps, we go ahead and calculate the mean of all the 15 dimensions.
2. The purpose of calculating the mean is to subtract the mean from the feature matrix and make it a zero-mean matrix.
   The mean calculated can be shown using an equation

$$mean(m_X)_j = \sum_{i=1}^{n} \frac{X_{i,j}}{n}$$

3. I subtracted the mean for all the features to generate a zero mean data matrix Y_{nxm} where n is the number of images and m is the number of features. In out case we have 36 x 15.

4. We know that the principal components can be calculated using two ways, one using eigen value decomposition and other is using SVD. Since we have non-square matrix, I preferred using the SVD.
   I computed the SVD of the Y matrix which generated three different matrix called U, S, and V. U and V are the left and right eigen vectors and S is the matrix containing singular values.
5. The reduced matrix Y_R is obtained by multiplying the original feature matrix X with the first three columns of the V matrix as those are the three principal directions.

Algorithm for Texture Segmentation

1. Read the content of the image, height and width using the terminal.
2. Now, to reduce the illumination of the image using windowing approach, declare the window size and generate the padding which his window/2. Based on the window padding, first extend the boundary using the mirroring approach. Now run the window over the image and subtract the mean of the values in the window with the image pixel.
3. Calculate the tensor product of the 5 laws filter and convert the 25D to 15-D by removing the redundant kernels.
4. Perform the convolution of the image with the laws filter to produce 15 filtered output which are basically 15 gray-scale images. Use boundary extension of 2 pixels before convolution.
5. Calculate the localized averaged energy features or vectors using the method described above. The neighborhood taken for calculating the energy vectors change as per the window size. Also perform feature normalization by dividing the entire feature vector matrix with the first element.
6. The feature matrix is ready for PCA to reduce it to 5-D.
7. Performed PCA to convert the 14D to 5D.

Algorithm for K-Means algorithm implementation

1. Initialize the centroid. Here, centroid is a 1D array which stores values of 6 class. I place the (270000x5) feature vector in the feature space to initialize the centroids.
2. Calculate the Euclidean distance of each image with 6 centroids. The distance is the sum of squares of the difference between the centroid and the feature vector.
3. Find the minimum distance and compare the minimum distance with the distance obtained for a cluster. If found matching, update the label count of that specific class.
4. Update the centroid with a new value. The new centroid value is calculated by calculating the mean of the features classified for that class.
5. Repeat the iteration until 6 centroids do not change over the iterations and remain the same.

6. Assign the 6 labels clustered by the algorithm by 6 different gray levels where each represent one texture. Use the vector to assign the gray levels {0,51,102,153,204,255} to denote six segmented regions in the output of six textures.

1.4.3. EXPERIMENTAL RESULTS

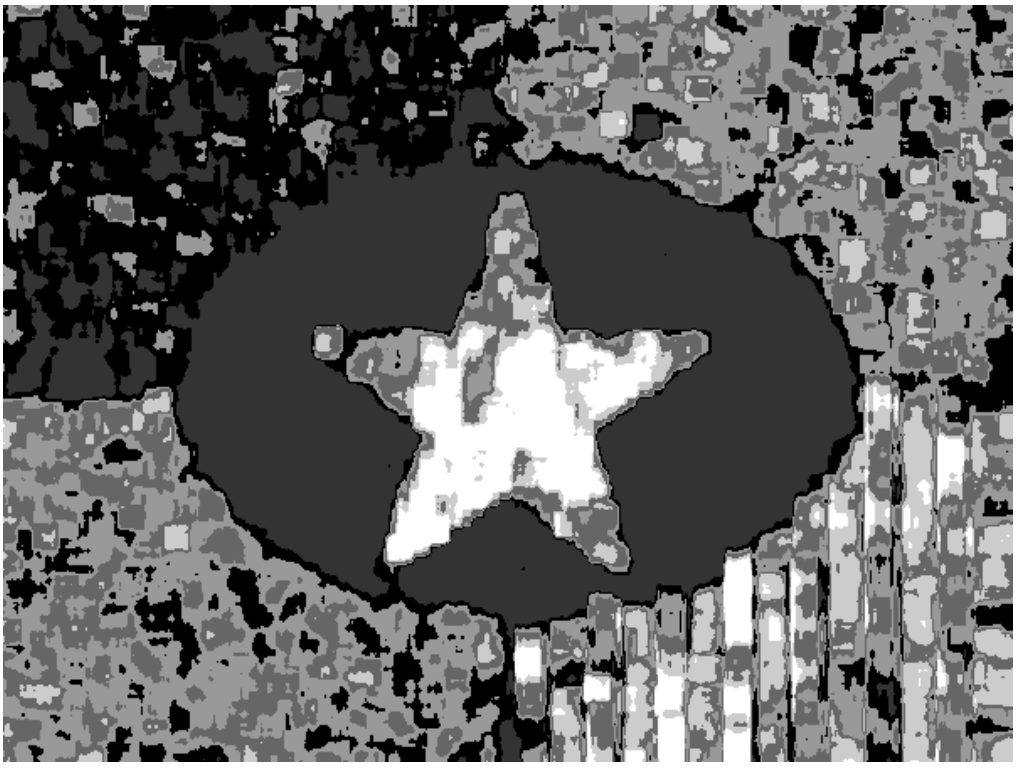


Figure 21. Segmentation for window size = 7

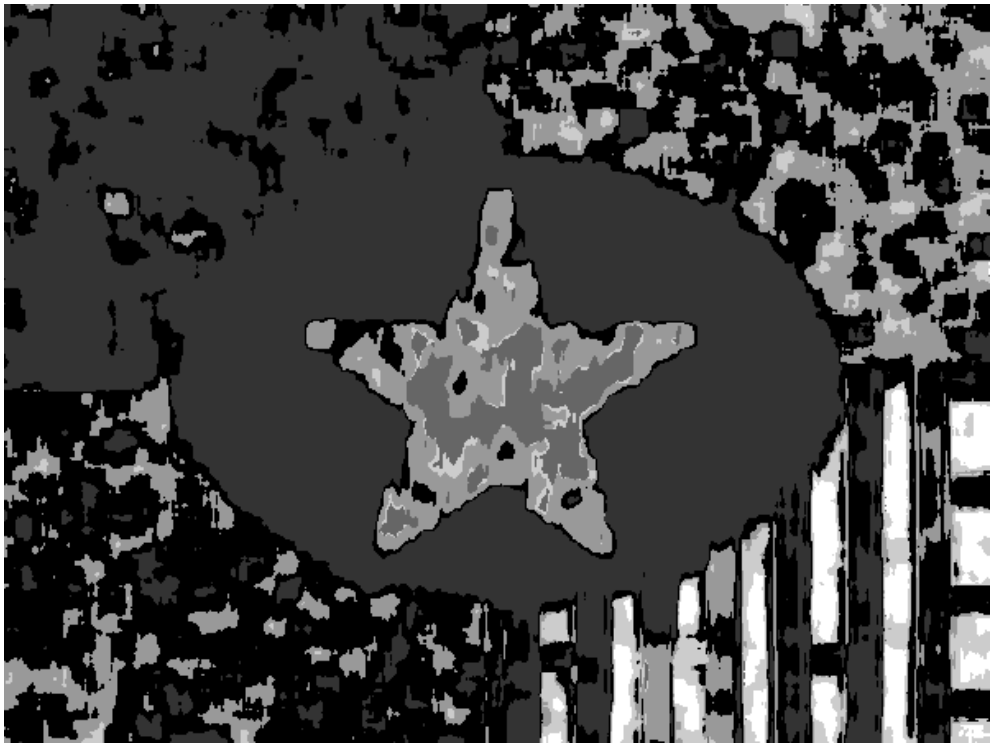


Figure 21. Segmentation for window size = 15

Figure 21. Segmentation for window size = 21

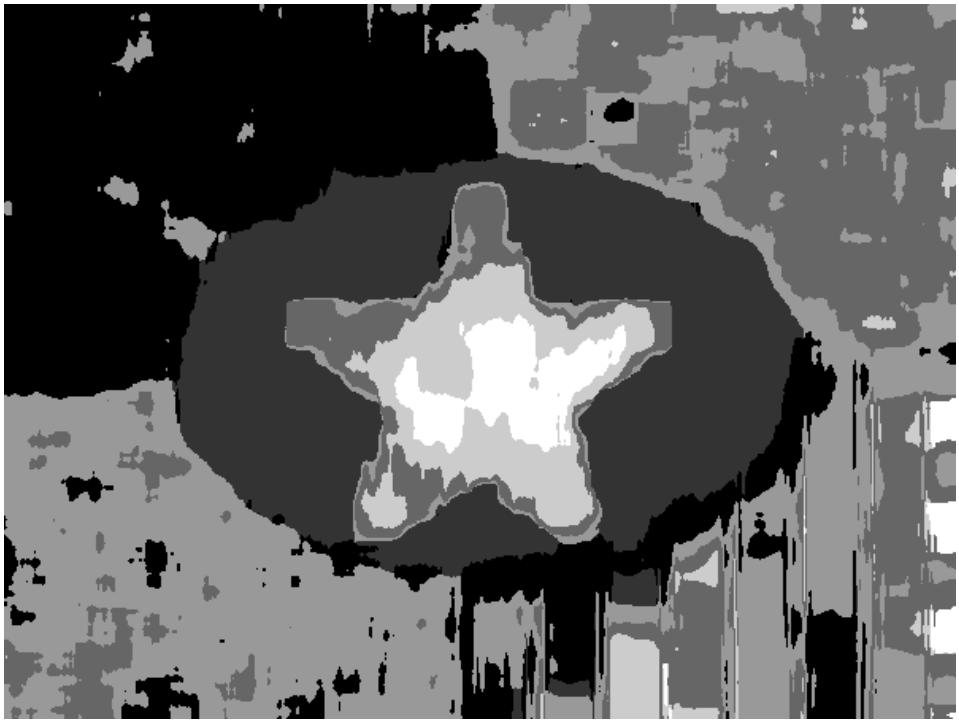


Figure 21. Segmentation for window size =31

1.4.4. DISCUSSION

I followed the same procedure as mentioned in the above section. Starting with generating the laws filters, convolving the image with the generated laws filters, generating the energy vectors and then normalizing it with $L5^T L5$. I also followed the proper mirroring boundary extension technique work on the segmentation. Once the feature vectors are ready, implemented K-Means to classify each pixel and generate the output. In addition, I used Principal Component Analysis to reduce the number of features. I reduced the features from 14-D to 5-D and used the K-Means clustering method to classify the points in the 5-D space. The following are my observation after the final segmentation output for various windows.

1. We see from the image that the composite textured image consists of 6 different regions or textures. From the window size 7, we see that the output is much better segmented than the one in the above example. Every individual cluster in the segmented output image is clear and its separated.
2. When we increase the window size, the output becomes clearer and we can see different segments in the image. We can see the boundary separating all the textures. Though the boundary is not clear, but the output is much better than the above window size of 7. This output also looks better than the one without PCA.
3. Similarly, as we increase the window size from 15 to 31, we see that the quality of segmentation improves and now we can see less of mixed clusters and more of uniform

clusters pertaining to a specific cluster. The outputs are not blur in this case and in fact clearer and crisper and bright.

4. This implies that as we increase the window size, the image's localized energy vectors are averaged with large pixels and as a result the final energy feature after normalization produces better segmentation results.
5. The results of PCA and before PCA when compared are different in terms of the quality of the image. Though the outputs after PCA for different window size are not that segmenting the image that great but its satisfactory. It gives visible change when compared to the one without PCA. This can be because PCA is generally used to reduce the number of dimensions to visualizer your data. Still when you reduce the feature dimensions, information is lost in that. The outputs might change if we perform feature selection methods like ridge or lasso.
6. The classification method used in this example is K-Means clustering. We know that to make this algorithm work, we must initialize it with a centroid. So, the classification of all the pixels in the image depends on the centroid of the clusters and thus the segmentation. I assigned the clusters centroid randomly. Every time I change the centroid initialization, the output seems to change a little but no significant change. The K-Means algorithm after PCA converged faster than the one before PCA, which is intuitive.

Moreover, I did recommend the post – processing technique that in order to reduce the small holes of varying intensities and to generate segmented output which consists of same gray scale intensity for segmentation, we can use the mode of the pixel intensities when a hole of different cluster is found. This will ensure that the image remain segmented without any holes. I have not implemented the above technique.

Also, to enhance the boundary of the adjacent regions, we perform a technique called Region Mapping which uses Region Adjacency Graph to produce enhanced region merging for texture segmentation. The reference is from a paper "Enhancement of Region Merging Algorithm for Image Segmentation"[4].

2. IMAGE FEATURE EXTRACTOR

Image feature is a simple image pattern, based on the which we describe what we see on the image[5][6]. In the field of machine learning and computer vision, feature extraction from the images is extremely important as it starts from initial set of measured data and builds features intended to be informative and non-redundant. The main aim behind forming or generating features from the image is to reduce the redundant information and convert it into essential information of limited size known as a feature vector. Feature vector an perform the desired task by this reduced representation instead of the complete initial data.

In image processing, there are many algorithms for feature detections and are categorized into :

1. Low Level
    a. Edge detection – generates features from the image which helps in detecting edges.
    b. Corner detection – extracts features to infer the content of the image. Used in motion detection, image and video tracking etc.
    c. Blob detection – Aims in detecting the regions in a digital image which expresses different properties such as brightness and color.
    d. Ridge Detection – extracts ridges from the images.
    e. Scale-invariant feature transform – Detects and describes local features in the image.

2. Shape Based
    a. Thresholding – A simplest method of segmenting the image used to create binary images.
    b. Blob Extraction – Aims in detecting the components in the image which share similar properties.
    c. Template matching – Aims in finding small part of an image which matches a similar template image.
    d. Hough transform – Aims to find the feature descriptor from the image based on gradients.

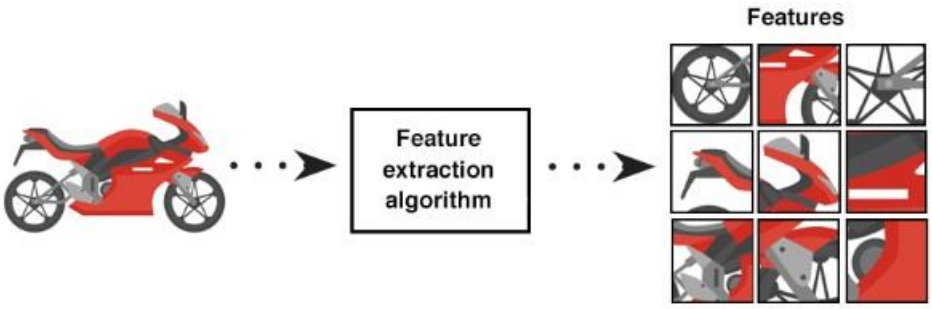


Figure 22. How a feature extraction algorithm work

2.1. SALIENT POINT DESCRIPTOR

2.1.1. DISCUSSION

Question 1. From the paper abstract, the SIFT is robust to what geometric modifications?

SIFT which stands for Scale-Invariant Feature Transform is robust to few geometric modifications. These are,

- Scale invariant
- Rotation Invariant
- Translation Invariant

In SIFT, all the features generated or extracted have a property that they possess invariance in image scaling, image rotation and image translation. In fact, they are also invariant to substantial range of affine distortion, addition of noise and change in illumination. This implies that if an image undergoes any kind of geometric modification in terms of scaling, rotation, translation, illumination or affine distortion changes, the SIFT algorithm remains robust to any modifications and provides the key points and descriptors.

Question 2. How does SIFT achieves its robustness to each of them?

It achieves robustness to the geometrical modifications using,

1. Scale – Invariance : Using multi-scale filter. The input image undergoes processing through the difference of Gaussian kernels and stack ups the pyramid to build multi-scale image. This essentially means that each level of pyramid helps the image to become scale invariant.
2. Rotational and Translational Invariance : Using difference of Gaussian functions. Essentially this function helps in providing invariance through identifying the key points at the maxima and minima of DoG in scale space.

Question 3. How does SIFT enhances its robustness to illumination change?

SIFT enhances its robustness to reduce the effect of illumination change by few steps :

- It converts the feature vector to unit length after normalizing. Normalizing the vector generally helps to cancel out any changes in the pixel intensities. It means that any pixel when multiplied with a constant will also multiply the gradients by the same constant and gets cancelled due to normalization.
- Brightness changes are handled in a way that gradient values remain prone to such changes because they are computed from pixel differences. Which makes it invariant to affine changes to illumination.
- SIFT also reduces the influence of large gradient magnitude by thresholding the values in the feature vector normalized to [0,1] to be limited to 0.2, and then renormalizing to unit length.

Question 4. What the advantages that SIFT uses Difference of Gaussians (DoG) instead of Laplacian of Gaussians (LoG)?

The advantage of using DoG over LoG in SIFT is :

- DoG or difference of Gaussians is a method which is fast than the Laplacian of Gaussians method.
- DoG efficiently computes L which is function that defines the scale space of the image.
- Also, we choose the key points at the maxima and minima of DoG in scale space to achieve rotational invariance.
- Laplacian of Gaussian is costly in terms of computation than DoG.

Question 5. What is the SIFT's output vector size in its original paper?

As per the original paper, SIFT's output size is 128. As per the paper, when a key point is selected, the algorithm creates a 16x16 neighborhood around it. Later, it is converted into 16 sub-blocks of size 4x4. The sub-block of 4x4 size is divided into 8 bins of orientation histogram. Which makes it total of 128 bin values and thus the size is 128. The experiments in his paper use a 4x4x8 = 128 element vector for each key point.

2.2. IMAGE MATCHING

2.2.1. ABSTRACT AND MOTIVATION

One of the feature extraction techniques in the image processing domain is called Scale-Invariant Feature Transform extracts the features form the image in form a key points and gradient orientations. The algorithm extracts feature which are scale invariant, rotation invariant and translation invariant. Also, they are robust to all kind of geometric transformations. The working of the SIFT feature extractor algorithm is utilizes 4 different steps such as,

1. Scale Space Extrema Detection
   This method ensures that the key points are chosen and are scale invariant. It apples Gaussian Kernel with different sigma values in octaves to the image and the octave are down sampled by 2 in every step. The variation in the sigma also results in the consideration of area in algorithm. Later, the difference of the Gaussians is taken into consideration by taking the difference in 2 consecutive LoG's in the octave.

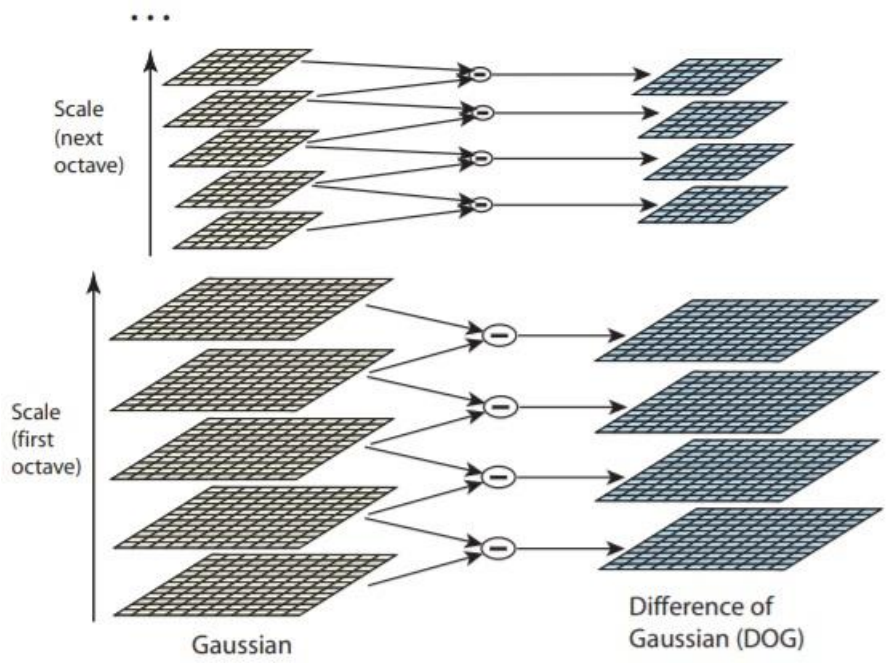


Figure 23 – Scale Space Extrema with difference of Gaussian

2. **<u>Key point Localization</u>**
   This step ensures the post processing of the key point found using the above method. This step removes the outliers and the low contrast points or edges from the images. Tayler series expansion is used to avoid errors and outliers. Moreover, the Hessian matrix is used to detect the principal curvatures. Also, the ratio of the highest eigen value to the lowest is taken to remove the low contrast points.

3. **Orientation Assignment**

   This step ensures rotational invariance by taking orientation value at each of the key point. This is done by taking the neighborhood of the point of interest. Orientation of each pixels in window is weighted by its magnitude and gaussian weighted circular window with 1.5 times scaling factor. Max orientation value is found out using histogram. To improve the robustness, 80% of the maximum value points are considered.

4. **Key point Descriptors**

   In this step a 16x16 neighborhood around the key point is taken. Followed by that we divided the neighborhood into 16 sub-blocks of 4x4 size for simplicity. For each sub-block generated, we create 8 bin orientation histograms. After doing this we get a total of 128 bin values which are available, and we can compare it in the dimensions in program. It is represented as a vector to form key point descriptor. [7]

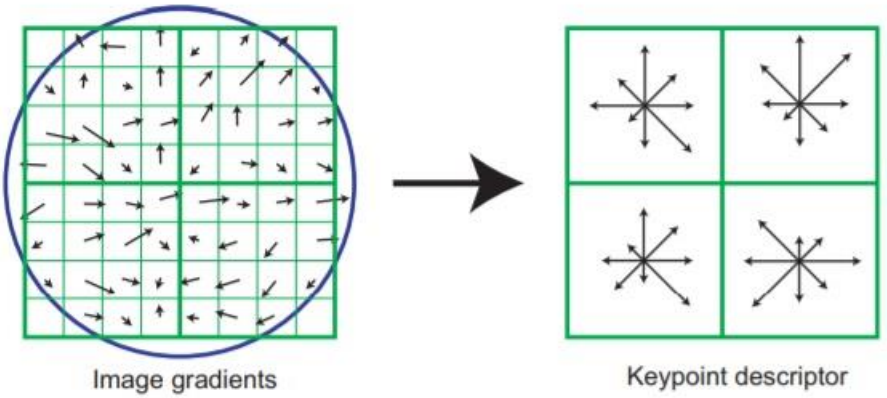


Figure 24. Key point descriptor generated from key points

5. **Key Point Matching**

   Key points between two images are matched by identifying their nearest neighbors. But in some cases, the second closest match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards only 5% correct matches, as per the paper.[7]

2.2.2. APPROACH AND PROCEDURE

By extracting the salient points using SIFT technique, key point descriptors are generated for different key points. The main reason to extract key points is to match them to another image to

find the matching points. This concept is mainly required in object recognition and computer vision applications.

To find the largest scale in the image, we check for the one having the largest L2 norm of descriptors. The L2 norm of the Euclidian distance is a good way to compare the feature descriptors.

For Feature Matching –

For filter matching, Lowe proposed to use a distance ratio test to try to eliminate false matches. The distance ratio between the two nearest matches of a considered key point is computed and it is a good match when this value is below a threshold. I used this method and compared the threshold value of 0.75. Indeed, this ratio test method helped in discriminating between ambiguous matches (distance ratio between the two nearest neighbors is close to one) and well discriminated matches. [8]

Nearest Neighbor Method -

There is also **cv2.drawMatchesKnn** which draws all the k best matches. If k=2, it will draw two match-lines for each key point.

FLANN stands for Fast Library for Approximate Nearest Neighbors. It contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features. It works more faster than BFMatcher for large datasets. [9]

Algorithm for finding the largest scale in the image and corresponding SIFT pairs

1. Read the input image Husky_1.jpg and Husky_3.jpg using imread().
2. Initialize the minHessain as 400.
3. Instantiate the SIFT detector using the function cv2::xfeature2d and give the minHessian in argument.
4. Compute the key point and descriptors for both the train and query image. In OpenCV this is done using the function detect() and compute().
5. Find the key point both the image using the function drawKeypoints() which generates key point on the image. Do it for both the train and query image.
6. Also find the orientation of each key point and save it into a file.
7. instantiate FLANN matching function and use the object to call knnMatch() function. This function takes in two descriptors and find the matches based on the nearest neighbor algorithm.
8. Consider the threshold value of 0.75 and for key point in the knn_match compare the L2 distance by multiplying the threshold. This will ensure that no two close points are captured due to noise.

9. Then perform cv2.drawMatch to generate a list of good matches. Take the first value of that list. This will generate the closest key point in the Husky_1.
10. Write this into a file.

The above method is for finding the best key point with the largest scale in Husky_3 and its closest neighboring key point in Husky_1. To find all corresponding SIFT pairs,

1. Repeat the same process as above.
2. Instantiate the FLANN matcher function.
3. Then perform the cv2.drawMatchesKnn to generate all good matches between the train and query images.
4. Repeat the same process for Husky_3.jpg and Husky_2.jpg, Husky_3.jpg and Puppy_1.jpg, Husky_1.jpg and Puppy_1.jpg.
5. Write the output in a file.

2.2.3. EXPERIMENTAL RESULTS

For part 1 where we need to find the key points, orientations of the key points and find the largest scale in Husky_3 and its nearest match in Husky_1.

Figure 25 – Key points in the image Husky_1

Figure 26 – Key point with orientation in Husky_1

Figure 27 - Key points in the image Husky_3

Figure 28 - Key point with orientation in Husky_3

The image for the largest scale in Husky_3 and its corresponding nearest match in Husky_1

Figure 29 – Key point with largest scale and it’s corresponding key point

For part 2 in the question, we show all the matching key point pairs in 4 different image combinations. The results are as follows :

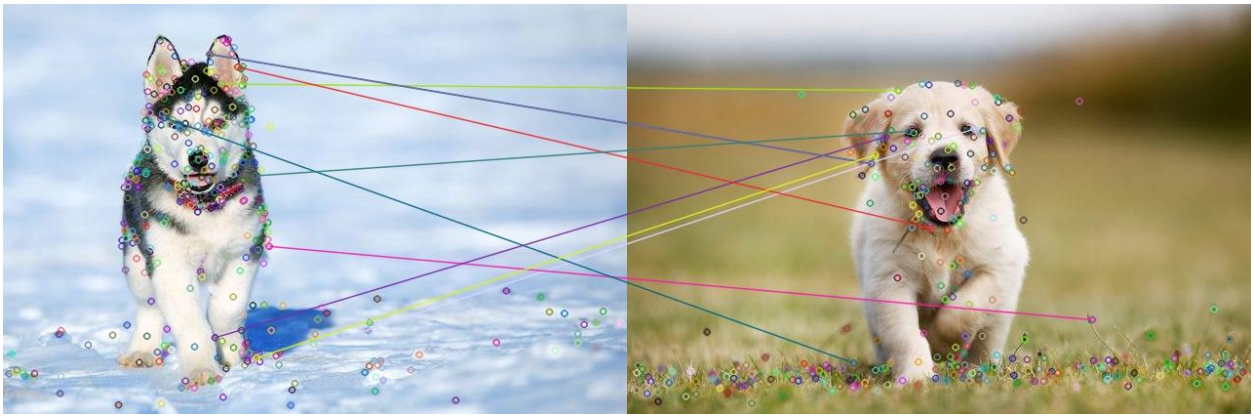


Figure 30 – Matching key point between images Husky_1 and Puppy_1

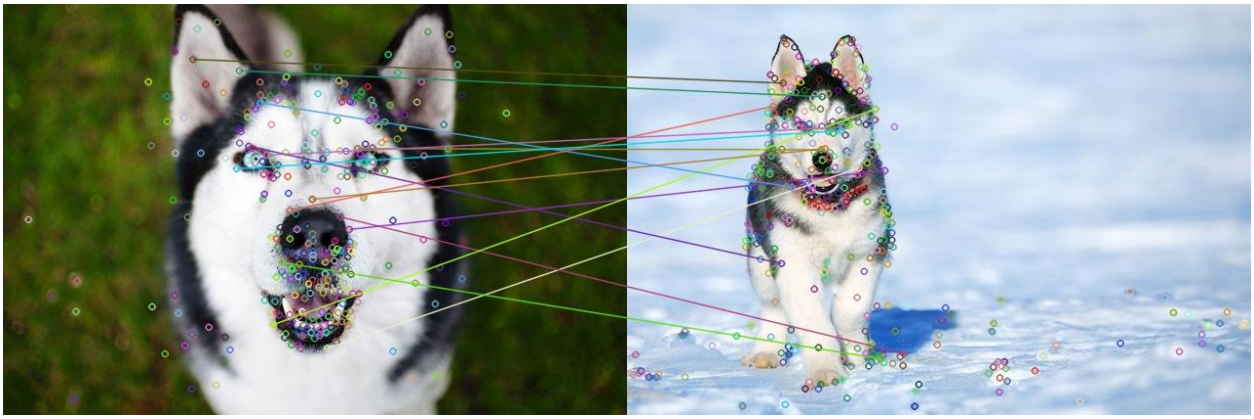


Figure 31 – Matching key point between images Husky_3 and Husky_1

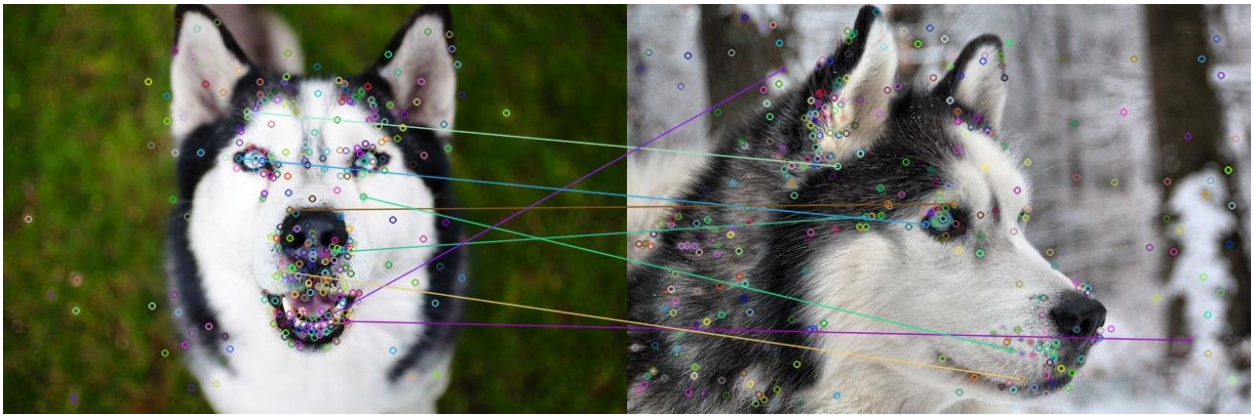


Figure 32 – Matching key point between images Husky_3 and Husky_2

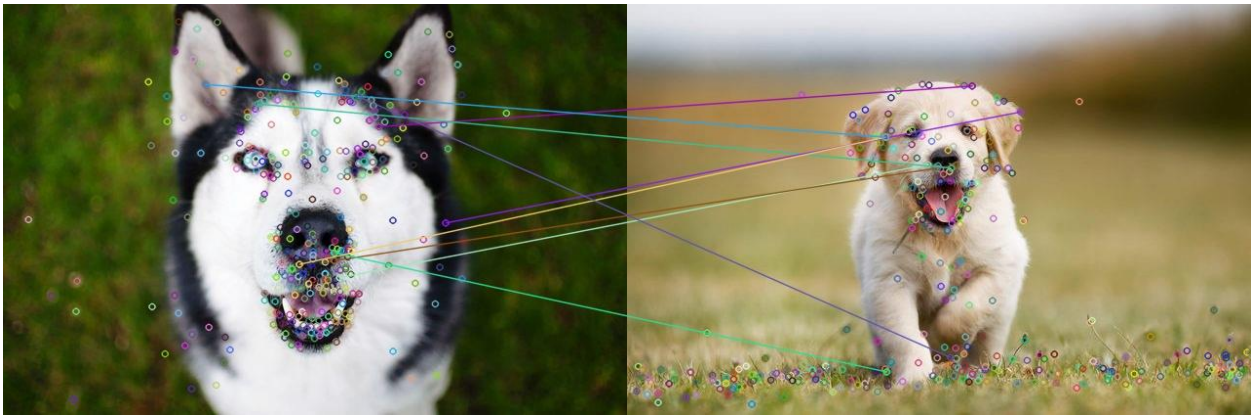


Figure 33 – Matching key point between images Husky_3 and Puppy_1

2.2.4. DISCUSSION

For the first part of the question,

1. SIFT key point generation gives the potential key points in both the images of husky. Its prominent from looking at the results that maximum key points are clustered around the face and body of the dog. Majority of the features collected by the algorithm is on the face of both the images. These key points when found using the SIFT are invariant to scale, rotation and translation as SIFT output doesn't change if the image undergoes any geometrical modification.
2. For feature matching, the strongest point in the husky_3 and its corresponding nearest point in the husky_1 was the key point near the ear of the both the images. By looking at the principle i.e. utilizing the Euclidian distance or L2 norm and matching it with the key points of other image, this result looks intuitive as both the key points in train and query image resembles to a common area. Also, when this same experiment is carried out using the FLANN feature matching, it yields the same result. This proves that the key point near the near of the husky_1 is the strongest point and its nearest neighboring key -point on husky_3 is also in the same area. It should be noted that we don't get a key point at a similar location because the train and query image are both different. But finding the nearest neighboring key-point is a satisfactory result.
3. The orientations of the key points in the SIFT algorithm is calculated by the weighted orientations of the neighboring pixels. The orientation which is calculated for a key point is invariant to the rotation. Also, the size of the orientation region around the key point depends on the scale i.e. if the scale is bigger ,the collection region is also assumed to increase.

   As per the paper by Lowe on SIFT, an orientation histogram with 36 bins covering the entire 360 degrees is constructed. The gradient magnitude and direction calculated in

that region is utilized for generating a circular window with the sigma equal to 1.5 times the scale of the key point.

For the part 2 of the question,

1. The key-point matching FLANN algorithm is used to calculate the matching key points in both the images. The results show the matching features where the key points correspond to the top features has the threshold ratio greater than 0.75. Which implies that the if the feature matching goes above the ratio threshold, it considers the match or else it discards. This is essential done to remove noisy matches around the same key point match.
2. By looking at the feature matching points of Husky_3 and Husky_1,
    a. We know that both the images are different, but they both have similar properties like both being the same husky breed and both having the front face in the image.
    b. The image has few matching similar points i.e. one near the ear, one point near the eye and few near the nose. We see that most of the key points in husky_3 are around the nose which lead to few matches in the husky_1 near the nose.
    c. Most of the matches are not matching in both images. Majority of the matched points by the algorithm are incorrect when looked through eyes. But one thing can be concluded that those points corresponds to the edges in the image. A key point falling on the edge of an object in the first image is matched with a similar edge in the other image. We know that edge also correspond to a good feature in any image, the detection and matching seems reasonable.
    d. The result of the FLANN based matcher is good as it doesn't match the noisy points or few points around the same key points. Also, for different value of Hessian, the matching points changes.
3. By looking at the feature matching points of Husky_3 and Puppy_1,
    a. We know that both the images are different, but they both have only two similar properties i.e. both having the front face in the image and both are dogs.
    b. The image has very few matching similar points i.e. only near the nose. We see that most of the key points in husky_3 are around the nose which lead to few matches in the Puppy_1 near the nose.
    c. Most of the matches are not matching in both images. Majority of the matched points by the algorithm are incorrect when looked through eyes. But one thing can be concluded that those points corresponds to the edges in the image. A key point falling on the edge of an object in the first image is matched with a similar edge in the other image. We know that edge also correspond to a good feature in any image, the detection and matching seems reasonable. Here the points are only matching because of the similar key points at edges.

    d. The result of the FLANN based matcher is good as it doesn't match the noisy points or few points around the same key points. Also, for different value of Hessian, the matching points changes.
4. By looking at the feature matching points of Husky_3 and Husky_2,
    a. We know that both the images are different, but they both one similar property like both being the same husky breed. They have different face orientations.
    b. The image has very few matching similar points i.e. one near the forehead, one point near the eye. We see that most of the key points in husky_3 are around the nose which lead to almost no matches in the husky_1 near the nose because they both have different orientations.
    c. Most of the matches are not matching in both images. Majority of the matched points by the algorithm are incorrect when looked through eyes. But one thing can be concluded that those points corresponds to the edges in the image. A key point falling on the edge of an object in the first image is matched with a similar edge in the other image. We know that edge also correspond to a good feature in any image, the detection and matching seems reasonable.
    d. The result of the FLANN based matcher is good as it doesn't match the noisy points or few points around the same key points. Also, for different value of Hessian, the matching points changes.
5. By looking at the feature matching points of Husky_1 and Puppy_1,
    a. We know that both the images are different, but they both have only two similar properties i.e. both having the front face in the image and both are dogs.
    b. The image has very few matching similar points i.e. one near the ear, one point near the eye. We see that most of the key points in husky_1 are around the nose which lead to almost no matches in the Puppy_1 near the nose.
    c. Most of the matches are not matching in both images. Majority of the matched points by the algorithm are incorrect when looked through eyes. But one thing can be concluded that those points corresponds to the edges in the image. A key point falling on the edge of an object in the first image is matched with a similar edge in the other image. We know that edge also correspond to a good feature in any image, the detection and matching seems reasonable.
    d. The result of the FLANN based matcher is good as it doesn't match the noisy points or few points around the same key points. Also, for different value of Hessian, the matching points changes.

2.3. BAG OF WORDS

2.3.1. ABSTRACT AND MOTIVATION

Bag of words is an algorithm developed initially to detect the frequency of occurrence of each word in a document which was grouped as a vocabulary. Bag of words is highly used in the field of natural language processing where machine learning algorithms detect the frequency of occurrence of a word in a document or generate an entire vocabulary of different words used in the document. The same idea was carried forward in the field of image processing where an image containing features were stored and represented as a histogram where the histogram denotes the frequency of occurrences. The general idea of bag of visual words (BOVW) is to represent an image as a set of features. Features here consist of key points and descriptors as seen the SIFT algorithm. In this algorithm, the frequency of occurrence of training images and test images are compared with the stored features in the form pf vocabulary of visual words and to determine which classes they belong to.

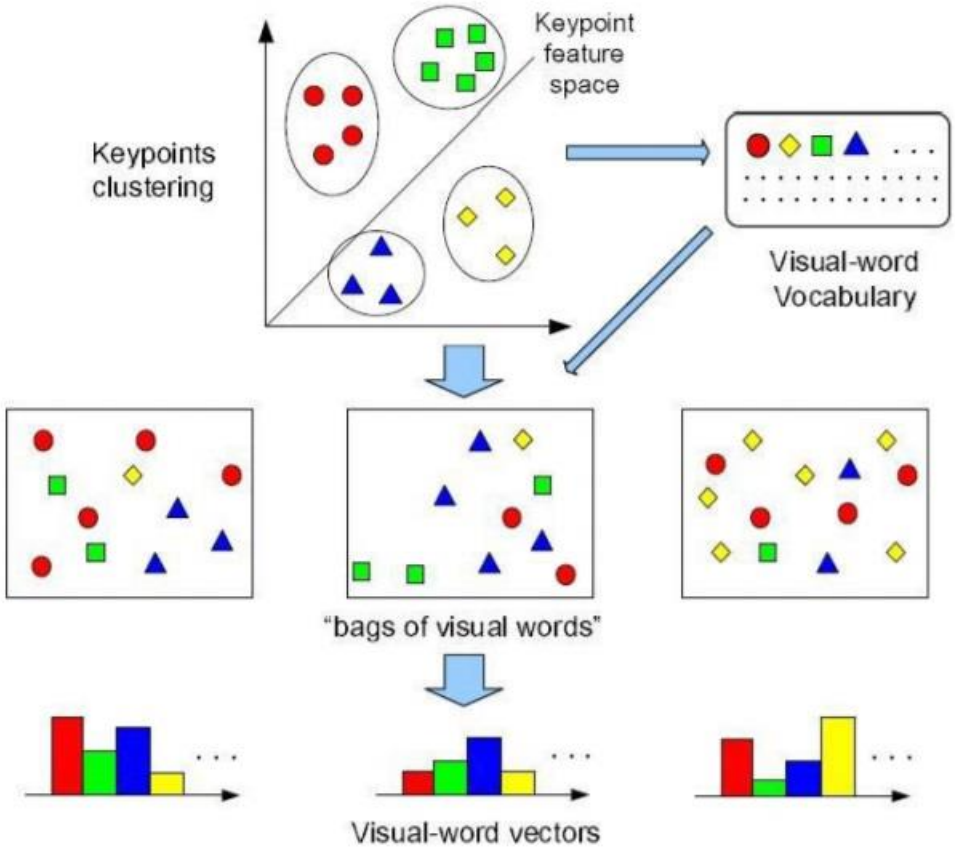


Figure 34 – Bag of Words working

2.3.2. APPROACH AND METHODOLOGY

Here, I implemented the bag of words using OpenCV. I used “BOW Trainer” class to train the images and cluster them based on the defined clusters. Here, we assume that we have 3 training images i.e. Hucky_1, Husky_2 and Puppy_1 and one test image which is Husky_3. I also used SIFT to extract the key points and descriptors of all the images. Using “BOWKMeansTrainer” class , I generated a codebook which contains the descriptors of the training images. By clustering we obtain the 8 centroids with each centroid of vector size 128 denoting the orientations. To find the closest distance between the training and testing image, I calculated the Euclidian distance and took index of the one giving the minimum distance from the centroid.

Algorithm for Bag of Words

1. Load all the images and read them using imread().
2. Initialize the detector using SIFT class and use compute() and detect() to identify the key points and the descriptors. Generate two vectors of descriptors. Once for the train images to be used for the cluster and one which contains all the descriptors of all images to be used in calculateDistance() function.
3. Add the descriptors of the training images to the KMeansTrainer and generate the cluster of it. The cluster length is defined as 8.
4. Once the codebook of training images is generated, compare it by calculating the Euclidian distance of each image with respect to the codebook. This will help us to visualize how different the feature vector is with the centroid of the codebook.
5. Find the minimum distance between the image descriptor and codebook to form a histogram by calculating the frequency of occurrence of descriptors in the image.
6. Compare the histogram and print the results.

2.3.3. EXPERIMENTAL RESULTS

The histogram’s and plots for the centroids of all images are given below.

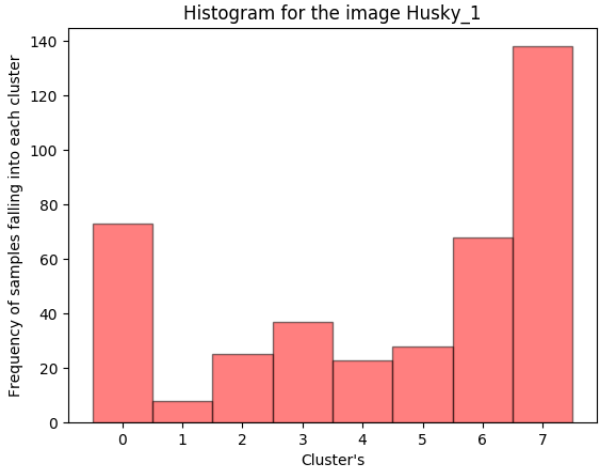


Figure 35 – Histogram of Husky_1

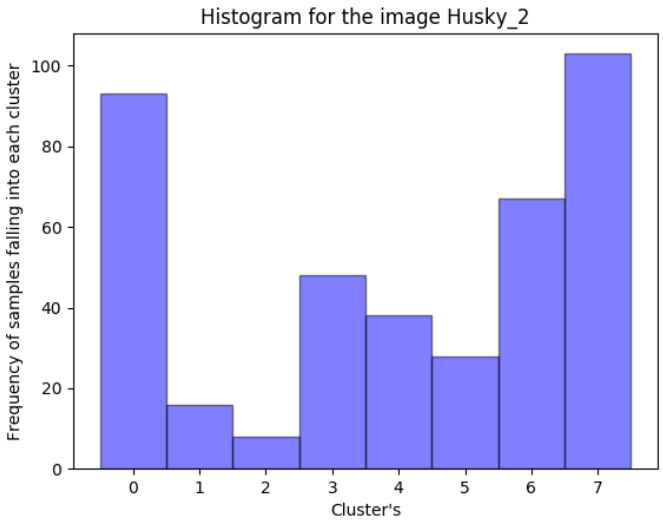


Figure 36 – Histogram of Husky_2

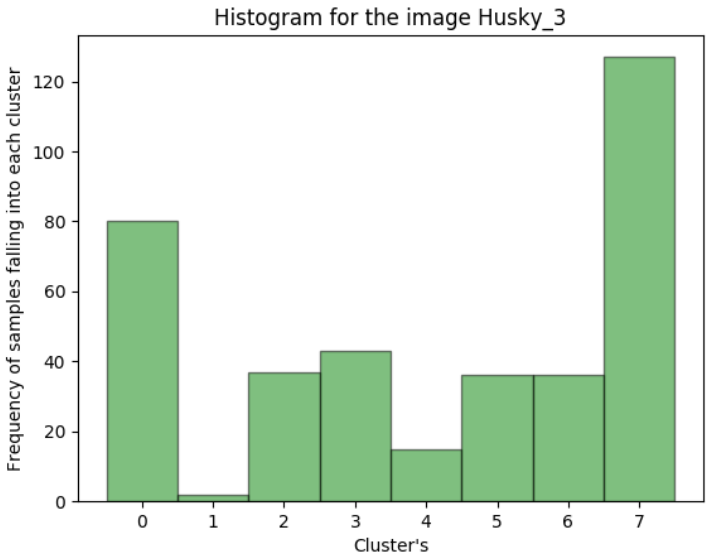


Figure 37 – Histogram of Husky_3

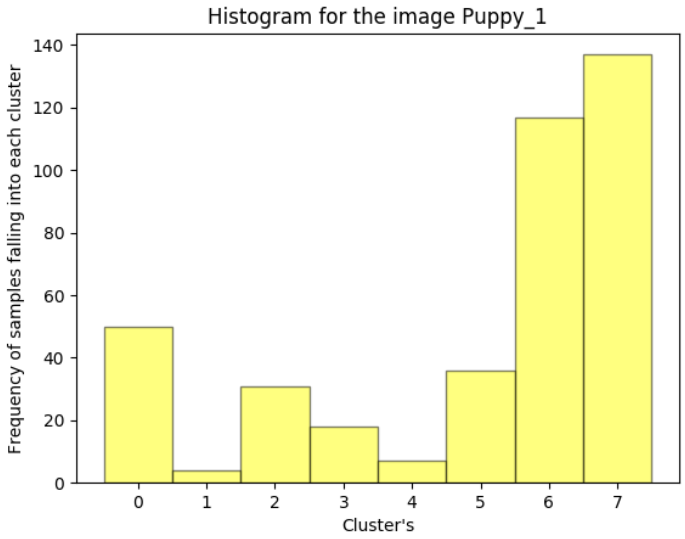


Figure 38 - Histogram of Puppy_1

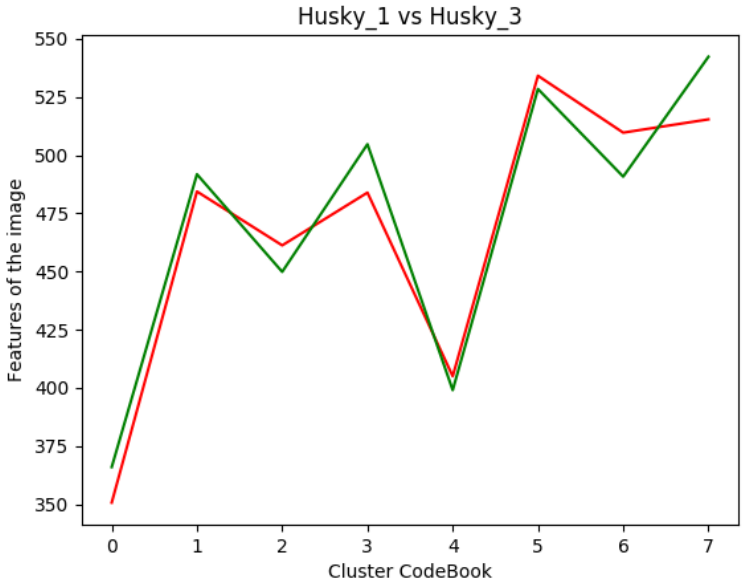


Figure 39 – Frequency of codewords matching for Husky_1 and Husky_3

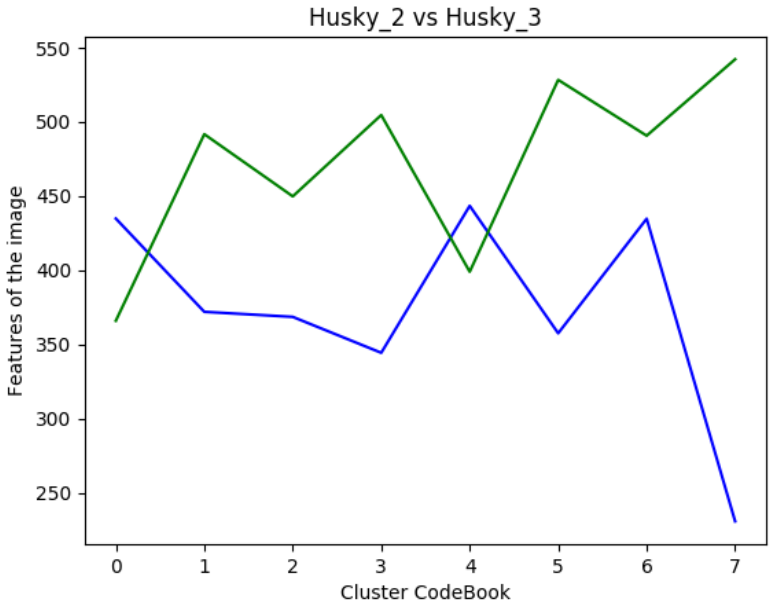


Figure 40 – Frequency of codewords matching for Husky_2 and Husky_3

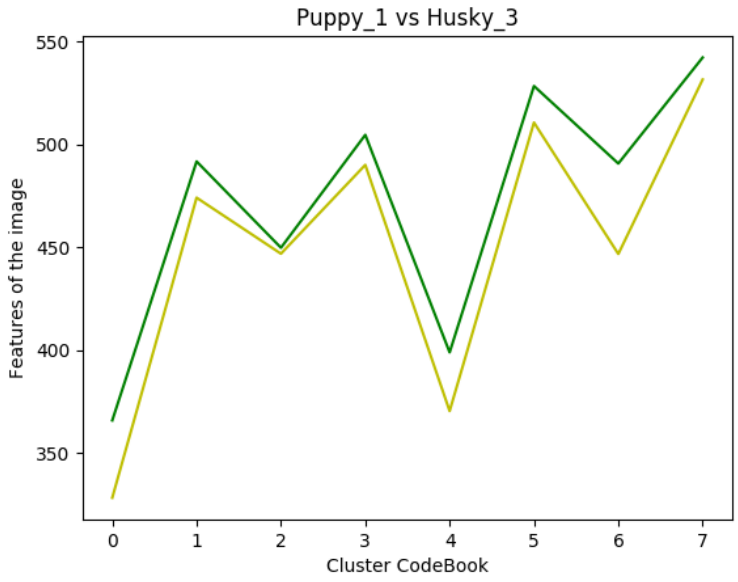


Figure 41 – Frequency of codewords matching for Puppy_1 and Husky_3

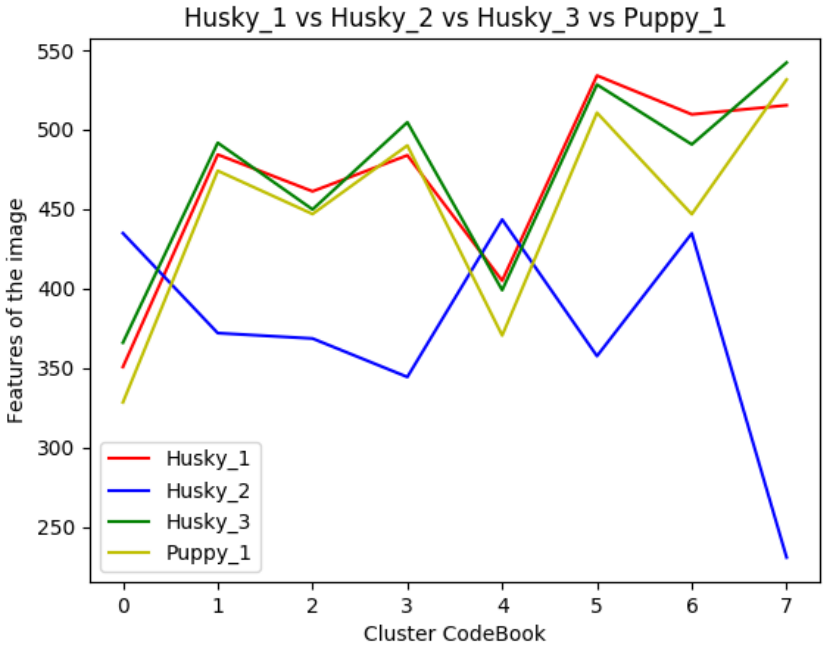


Figure 42 – Frequency of codewords matching of all images altogether

2.3.4. DISCUSSION

Bag of words is a way of representing your features into a bag and plotting the histogram provides us the information of the occurrence of a feature in the feature bag. In this example, we had four images which included 3 different images of Husky and 1 image of puppy. As from the image it is quite prominent that the images had less similarity. The following are my observations for the results I obtained,

1. Looking at the images, it is prominent that the three images of husky has similarities. In fact, the husky_1 and husky_3 are similar in the way that they are representing the front husky face. But in case of husky_2 we see the side face. Also, the puppy has a front face image.
2. From the histogram generated by plotting the frequency of features matching the codebook, we can say that the histogram of husky_1 and husky_3 are quite similar with some difference in the frequency of values in the cluster bins. This is because of the fact they both are husky, and the image has the front face. This made the features in the bog of words algorithm generate similarity in these two images.
3. By looking at the frequency of codewords matching plot of husky_1 and husky_3, we see that the assumption we had is true. Indeed, codewords of husky_3 is similar and matches husky_1.

4. Also, the plots and histograms bring a little similarity(less than husky_2) in when comparing the husky_3 and puppy_1. This also bring us back to our assumption that even puppy has a front face and that's why the algorithm found features relevant to the face of the dogs.

Also, on changing the values of Hessian, the algorithm yields the same results with husky_3 codewords like husky_1. Bag of words seems to work well for descriptors given they have highest match for every Hessian Value. I can also conclude that since the algorithm uses K-Means clustering and we know that the K-Means highly depends on the initialization of clusters.

3. REFERENCES

1. (https://www.mathworks.com/help/images/texture-analysis-1.html)
2. https://towardsdatascience.com/understanding-random-forest-58381e0602d2
3. https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496
4. http://iieng.org/images/proceedings_pdf/7795E0314085.pdf
5. https://en.wikipedia.org/wiki/Feature_extraction
6. https://medium.com/machine-learning-world/feature-extraction-and-similar-image-search-with-opencv-for-newbies-3c59796bf774
7. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
8. https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html
9. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html