EE 569

HOMEWORK 4

By

Morayo Ogunsina

USC ID : 7371213793

ogunsina@usc.edu

Issued 3/7/2021

Due 2/28/2021

Table of Contents

Problem 1 :

Texture Analysis

I. Abstract and Motivation

Texture analysis is useful in image processing to for many tasks particularly useful in computer vision and image analysis. Some of these areas are in medical image processing such as ultrasound and CT scans, remote sensing of forests, landscapes, water bodies and even in defense applications. One thing about the need for texture analysis and classification is that there is no formulaic way of representing different texture properties, hence the need for statistical methods employed in machine learning. For this problem, I made use of K-means clustering.

One thing to note is that Texture Analysis is a broad topic and includes texture segmentation, texture synthesis and texture classification, which is the focus of problem 1. For textures, the general idea is that they come in repeated patterns with some randomness here and there to make it unique from different texture patterns.

Human beings can differentiate and identify different texture properties of objects they see but this ability is much more difficult for a machine to accomplish. And this is the main motivation of Texture Analysis : to develop an effective way to encode different texture properties for image processing, basically can we ask a machine what kind of texture an object is/has?

Fig 1: Examples of Textures [2]

II. Approach and Procedures

Laws Filters

Laws Filters, designed by Kenneth Law, a USC grad student are a set of ID filters to analyze textures. These filters of different dimensions are generated by computing the tensor products to obtain a 2D filter, i.e., each filter is obtained by multiplying two of all the 1D kernels. Each kernel (there are five) is unique and can detect a pattern a pattern in a direction of an image.

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]

Fig 2: 1D Kernel for 5x5 Laws Filters [2]

From these kernels, 25 total filters can be generated by the matrix multiplication $\boldsymbol{F = K_1^T K_2}$, K_1 and K_2 can be any of the 5 kernels.

The 25 filters are given below:

L5L5	L5E5	L5S5	L5W5	L5R5
E5L5	E5E5	E5S5	E5W5	E5R5
S5L5	S5E5	S5S5	S5W5	S5R5
W5L5	W5E5	W5S5	W5W5	W5R5
R5L5	R5E5	R5S5	R5W5	R5R5

Fig 3: Table Showing the 25 Laws Filters after Kernel Matrix Multiplications.

For Laws Filters, they are needed for texture classification as they are used to extract useful features needed for the classification, in this case, KMeans. Just like the Gx and Gy filters detect edges horizontally and vertically respectively (from Homework 2), the filters in Fig 3 detects a pattern vertically while the second detects pattens horizontally. When the filters are applied, we get 25 response feature vectors for each image.

An important operation to do before applying the Laws Filters is to reduce the illumination effect on the image which is the same as subtracting the image mean for each image. Then Laws Filters are applied to the new image pixels.

Fig 4 : Illumination reduced for Images in Fig 1

Energy Features

The next step is to calculate the average energy for each feature from the 25D feature vector response for each image. Energy feature averaging is simply averaging all the response values of pixels within a small window which denotes the energy/strength of each of those pixels. For this problem I used a default window size of 15x15. One thing to note in doing this computation is that a response value can be negative too, so we need to take the absolute value of these responses to prevent computation error in averaging. We can also make use of a squared magnitude as well. So, there will be 25 energy magnitudes for all the 25 response vectors, and these become the 25D dimensional feature vector response of the texture.

I decided to combine symmetric pairs from the 25D feature vector response to get a reduction in dimension to 15D feature vector response. This means that the 2D patterns in the 25D features that have similar structures can be combined and the resulting value for the 15D feature vector is an averaging of the original values. So, for this case, the 15D vectors from the filter response are listed as **L5E5, E5E5, S5S5,W5W5, R5R5, L5E5/E5L5, L5S5/S5L5, L5W5/W5L5, L5R5/R5L5, W5R5/R5W5, E5S5/S5E5, E5W5/W5E5, E5R5/R5E5, S5W5/W5S5, and S5R5/R5S5**. I should note that this is an optional step and using the full 25D vector might give better results for the texture representation.

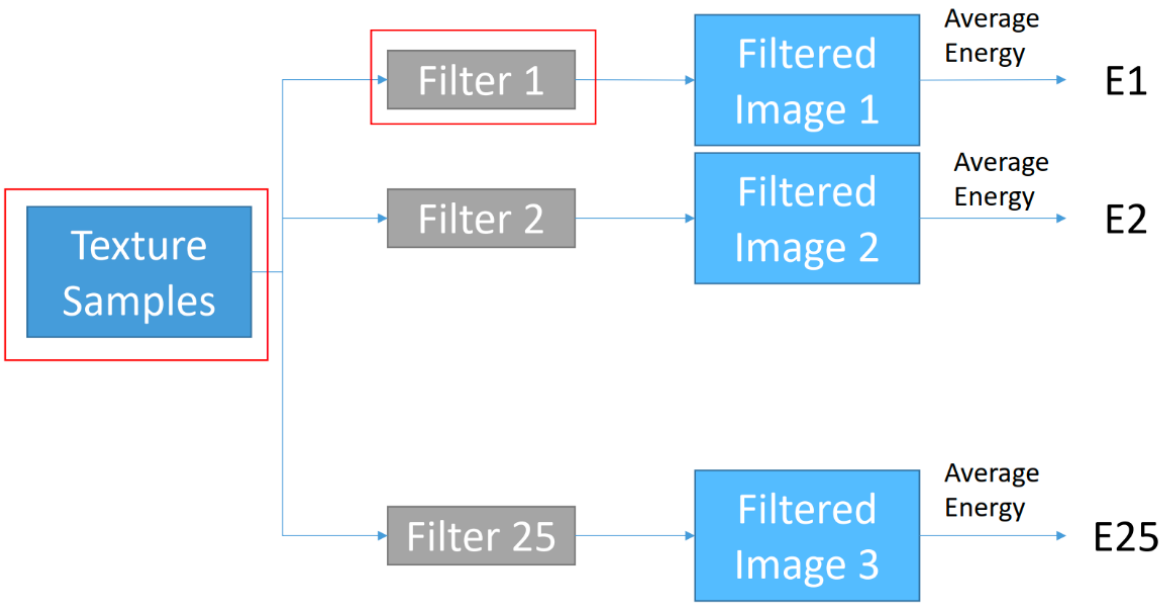


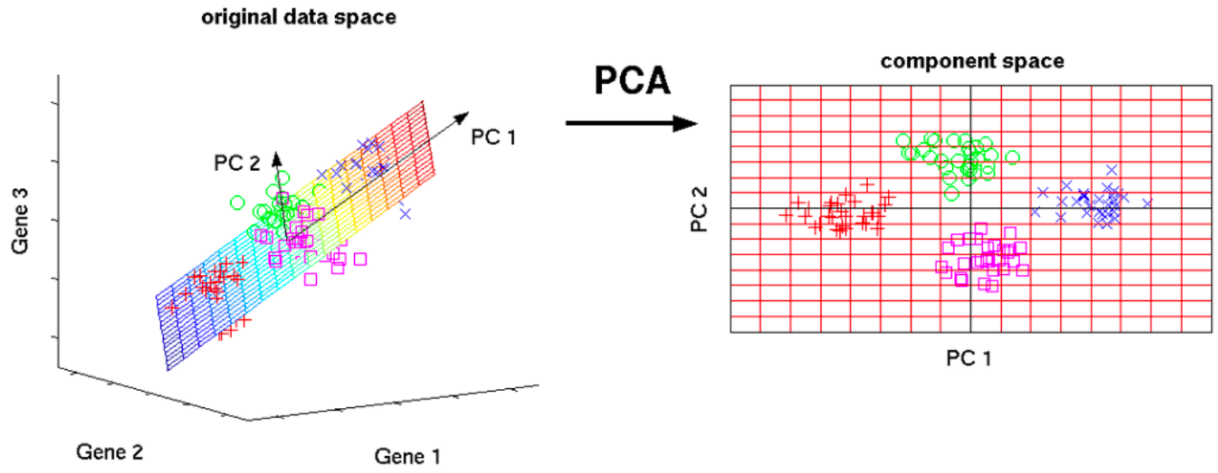
Fig 5: How the Energies are extracted from each 25D feature . [2]

Strongest Discriminant Power

I made use of MATLAB's var function to calculate the discriminant power of the 15D feature matrix. I will provide more information about this later.

Principal Component Analysis

The idea behind using Principal Component Analysis is to use fewer dimensions to represent high dimensional data with little loss as possible. We try to project a higher dimension vector to a subspace of lower dimension with little error loss as possible, i.e mean square error or L2 error (explained from lecture video). It is the most used method for dimensionality reduction. For this problem I made use of the MATLAB's svd function. Here the idea is to find the covariance matrix and then the eigen values. The eigen values that is the largest, is the first principal component and the next is the second and so on. Here, the singular, insignificant values in the 15D matrix are discarded and the final dimension of the feature vector becomes 3D. An important metric for this analysis is the eigenvector transform which I will be discussing later.



3D →2D

Fig 6: Example Result of PCA of 3D to 2D.

Machine Learning Algorithms

Here I outline the learning algorithms used for the texture classifications.

a. K Nearest Neighbour

This is an algorithm that classifies based on similarity measure. The "K" denotes the number of nearest neighbors that must be considered in the majority voting process. One good way of choosing K is to denote is as sqrt(Number of samples in training dataset). But for this problem, I decided to go with 3, having 36 training samples. We use a distance metric to define the similarity between two data points. For this problem, I used the Mahalanobis distance characterized as

$$D_M(\vec{x}) = \sqrt{(\vec{x} - \vec{\mu})^T S^{-1} (\vec{x} - \vec{\mu})}.$$

Where S^{-1} is the covariance matrix and Mu is the mean.

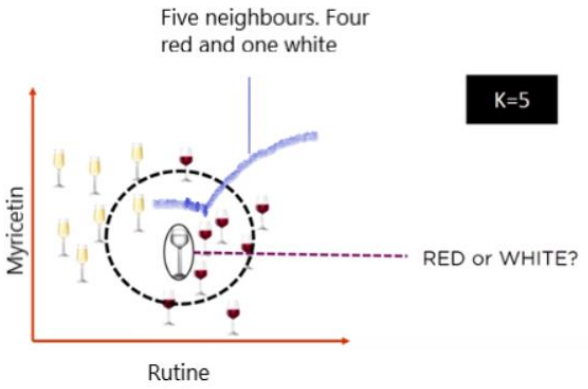


Fig 7 : Example of a KNN classification. [8]

```
%% STEPTRAIN 5 : Apply the K nearest Neighbour
% source : https://www.mathworks.com/help/stats/classification-nearest-neighbors.html?s
% NumNeighbours = Number of Nearest Neighbours to find = 3
% NSMethod : Nearest Neighbour search method use exhaustive
% Distance = use the Mahalanobis distance metric
KnnModel = fitcknn(PCAfeatureTrainMatrix,labels,'NumNeighbors',3,...
    'NSMethod','exhaustive','Distance','mahalanobis',...
    'Standardize',1);

% % use default params,
% had a problem here with pdist.
% followed the resolution at
predKnnTrain3D = predict(KnnModel,PCAfeatureTrainMatrix); % on training samples
```

Fig 8: MATLAB usage for KNN

b. K Means

This is an unsupervised machine learning algorithm in which the datasets are classified into k clusters. It is an iterative algorithm, and each iteration adjusts the centroid of each cluster through averaging. One thing to note is that this algorithm returns different results for different runs due to the randomness in the initialization procedure. In this case, multiple runs are better for to get a satisfiable result which is what I did. I only applied the Kmeans to the test images.

```
%% K-Means clustering on Test Images
% source : https://www.mathworks.com/help/stats/kmeans.html
kmeansPredPCA = kmeans(PCAFeatureTestMatrix, 4); % with PCA 3D
kmeansPred = kmeans(testfeatureMatrix, 4);  % without PCA 25D
```

Fig 9: MATLAB usage for Kmeans

- K-means: a type of unsupervised data analysis algorithm

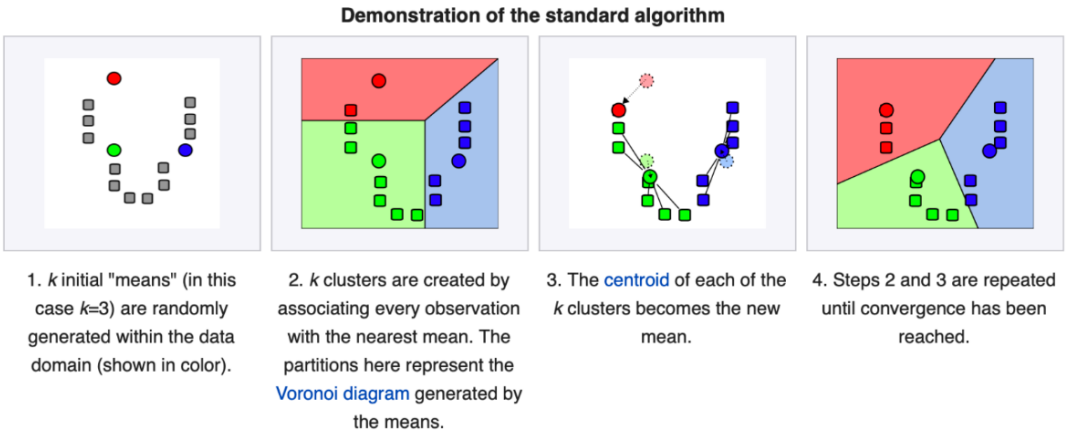


Fig 10 : Demonstration of the Kmeans Algorithm [2]

c. Support Vector Machine

The SVM is a type of supervised training algorithm which it finds an optimal solution to maximize the distance between decision boundaries for linear classifications. These hyperplanes are simply to distinctly classify data points.

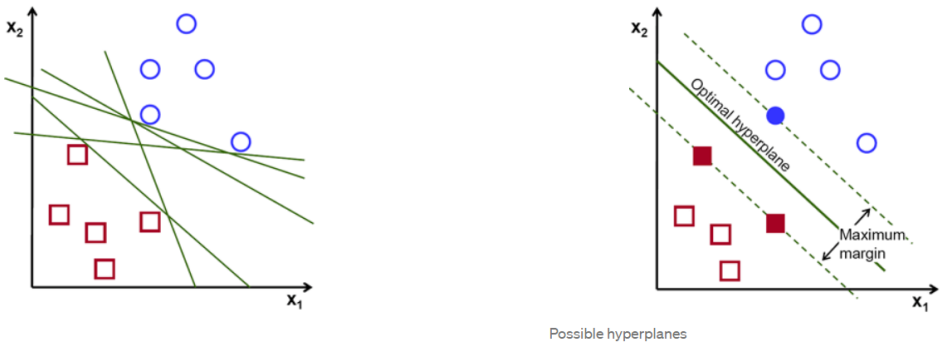


Fig 11 : Demonstration of an SVM classifier [9]

The objective is to choose a suitable hyperplane that has the maximum distance between data points of different classes. So, what are the support vectors then? They are simply the points we need to build the SVM ; they are closer to the hyperplane and can determine the position and orientation of the decision boundary (hyperplane).

```
%% SVM classifier on trainning images
% source https://www.mathworks.com/help/stats/fitcecoc.html
% % use default params,
% with PCA, 3D
SVMModel = fitcecoc(PCAfeatureTrainMatrix, labels);
[predTrain3D, scoreTrain3D] = predict(SVMModel,PCAfeatureTrainMatrix); % on training samples

% not needed
% without PCA, 25D
%SVMM25Dodel = fitcecoc(featureTrainMatrix, labels); % use default params,
%[predTrain25D, scoreTrain25D] = predict(SVMModel,featureTrainMatrix); % on training samples

% Now let's test it on the test samples
[predTest3D, scoreTest3D] = predict(SVMModel,PCAFeatureTestMatrix);

%disp(SVMModel.ClassNames);
%CVMdl = crossval(SVMModel,'Options'); % Cross-validate Mdl using 10-fold cross-validation.
%genError = kfoldLoss(CVMdl); % Estimate the generalized classification error.
```

Fig 12 : MATLAB usage of SVM.

d. Random Forests

The RF is also a type of supervised training algorithm which solves classification and regression problems. It is majorly based off on decision trees and uses multiple of these during training to give a result using a majority voting method for classification or mean/average for regression-type problems. I made use of RF for HW 2 for the edge detection and see how that might be relevant also for this homework. For my implementation, I made use of the ***TreeBagger*** function from MATLAB which denotes the general usage of aggregating or bagging tree learners, to put it simply.

```
%% Random Forest classifier : Create A Bag of Decisin Trees
% source : https://www.mathworks.com/help/stats/treebagger.html
% % use default params,
% with PCA, 3D
RFModel = TreeBagger(5,PCAfeatureTrainMatrix,labels, 'OOBPrediction', 'On', 'Method', 'classif
view(RFModel.Trees{1},'Mode','graph')

%predict of Train samples
predRFTrain3D = predict(RFModel,PCAfeatureTrainMatrix); % on training samples

% Now let's predict it on the test samples
predRFTest3D = predict(RFModel,PCAFeatureTestMatrix);
```

Fig 13 : MATLAB usage of Random Forest.

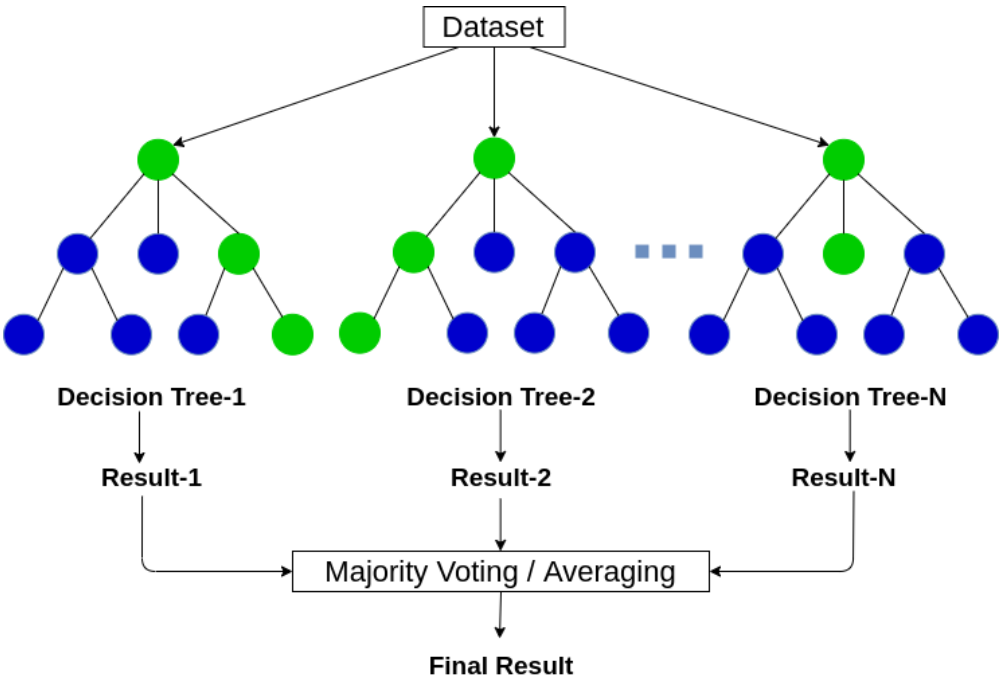


Fig 14 : Demonstration of a Random Forests classifier [9]

The steps for **Texture Classification, Feature Extraction** are summarized to :

- STEP 1 : Run program (check README for instructions).
- STEP 2: Read all 36 training images, 9 each for blanket, Brick, Grass and Rice. Each image has a ***128x128*** dimension. So, 4 Image matrices of ***128x128x9*** size.
- STEP 3: Reduce Illumination Effect by subtracting mean, each class matrix is still ***128x128x9*** in size.
- STEP 4: Apply laws filters, i.e convolve each image with the 25 filters and then calculate the average energy values to get feature vector response for each class matrix to as a ***9x25*** double-type sized matrix. These become our energy values.
- STEP 5: Combine all the 4-class feature into a single training feature matrix to give ***36x25*** double-type sized matrix.
- STEP 6 : Reduce the feature dimension from 25 to 15 to get ***36 x15*** double-type matrix.
- STEP 7 : Find the feature with the strongest discriminant power as a ***15x1*** double-type matrix.
- STEP 8: Apply PCA to reduce to from ***36x15*** to ***36x3*** dimensional matrix.
- STEP 9: Apply the KNN to these final feature matrices to obtain predicted classifications of ***36x1*** matrix of labels.
- STEP 8: Apply the KMeans to these final feature matrices to obtain predicted classifications of ***36x1*** matrix of labels.
- STEP 9: Apply the SVM to these final features for PCA-reduced matrices to obtain predicted classifications of ***36x1*** matrix of labels.

- STEP 10: Apply the RF to these final features for PCA-reduced matrices to obtain predicted classifications of ***36x1*** matrix of labels.

The steps for **Advanced Texture Classification: Classifier Explore** are summarized to :

- STEP 1: Read all 12 test images. Each image has a ***128x128*** dimension. So, matrices of ***128x128x12*** size.
- STEP 2: Reduce Illumination Effect by subtracting mean, each class matrix is still ***128x128x12*** in size.
- STEP 3: Apply laws filters, i.e convolve each image with the 25 filters and then calculate the average energy values to get feature vector response for each class matrix to as a ***12x25*** double-type sized matrix. These become our energy values.
- STEP 4 : Combine symmetric pairs or reduce the feature dimension from 25 to 15 to get ***12x15*** double-type matrix
- STEP 5 : Find the feature with the strongest discriminant power as a ***15x1*** double-type matrix.
- STEP 6: Apply PCA to reduce to from ***12x15*** to ***12x3*** dimensional matrix.
- STEP 7: Apply the KNN to these final feature matrices to obtain predicted classifications of ***12x1*** matrix of labels.
- STEP 8: Apply the KMeans to these final feature (both for PCA reduced and non-PCA reduced) matrices to obtain predicted classifications of ***12x1*** matrix of labels.
- STEP 9: Apply the SVM to these final features for PCA-reduced matrices to obtain predicted classifications of ***12x1*** matrix of labels.
- STEP 10: Apply the RF to these final features for PCA-reduced matrices to obtain predicted classifications of ***12x1*** matrix of labels.

I. ***Experimental Results***

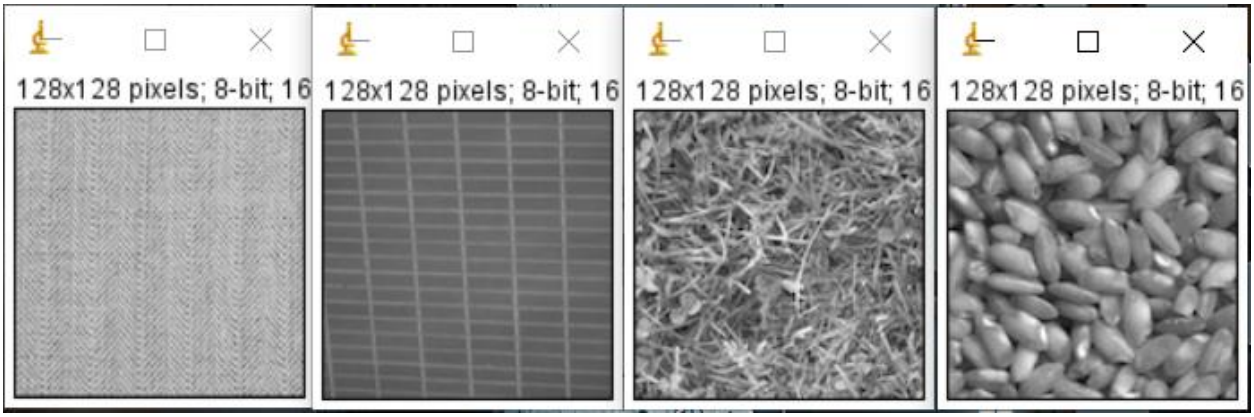


Fig 15: From left, blanket, brick, grass, and rice image samples for training.

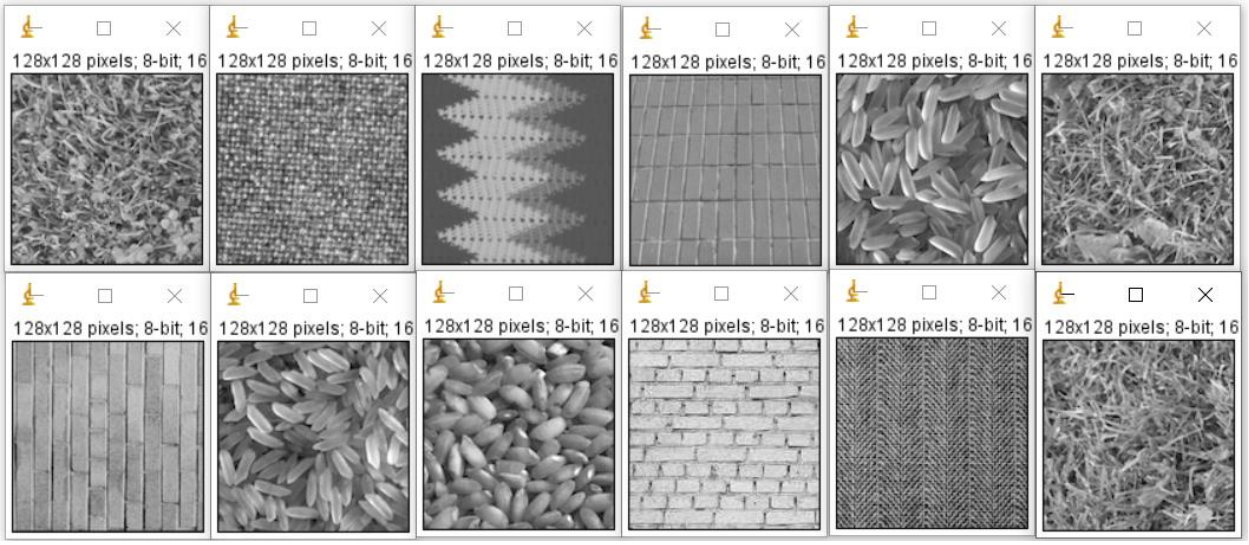


Fig 16: Image samples for testing. From top left to bottom right,

denoted as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

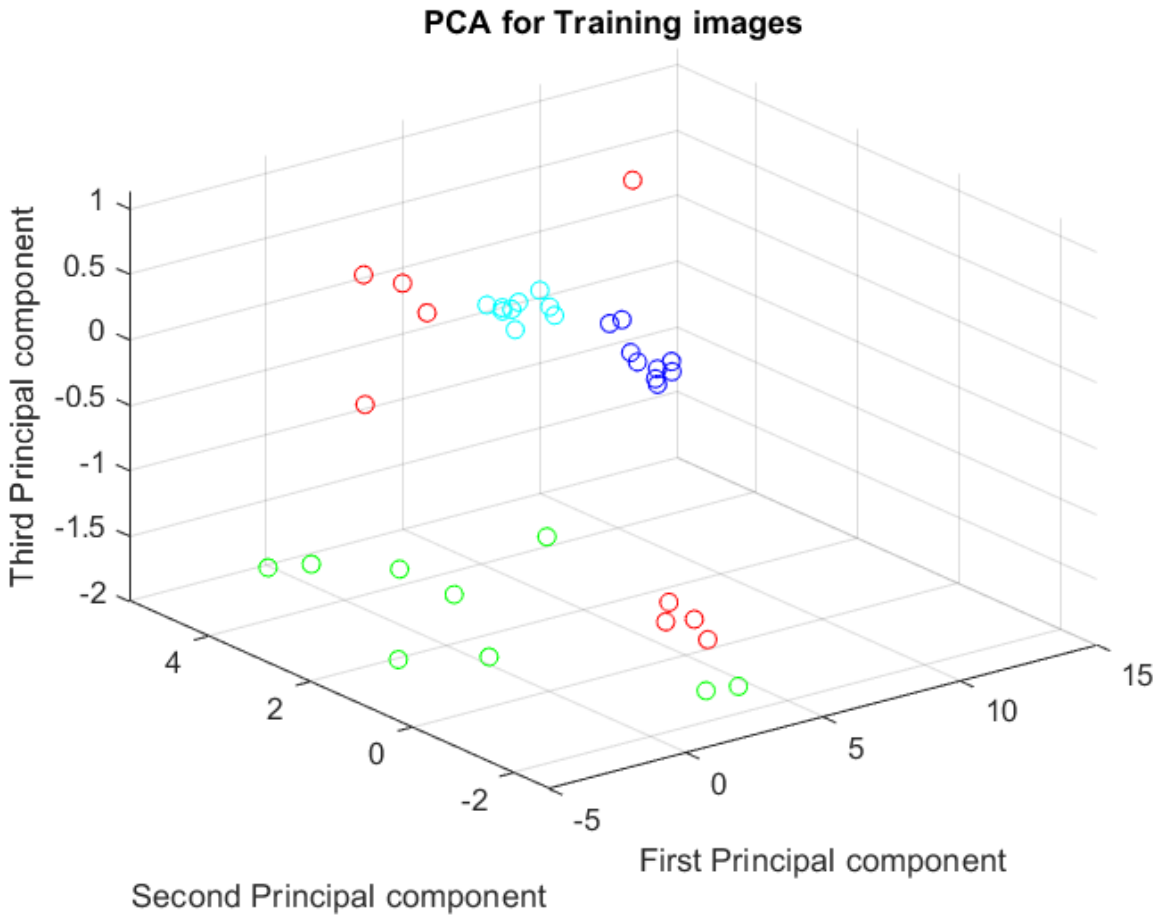


Fig 17: PCA feature space obtained for training images

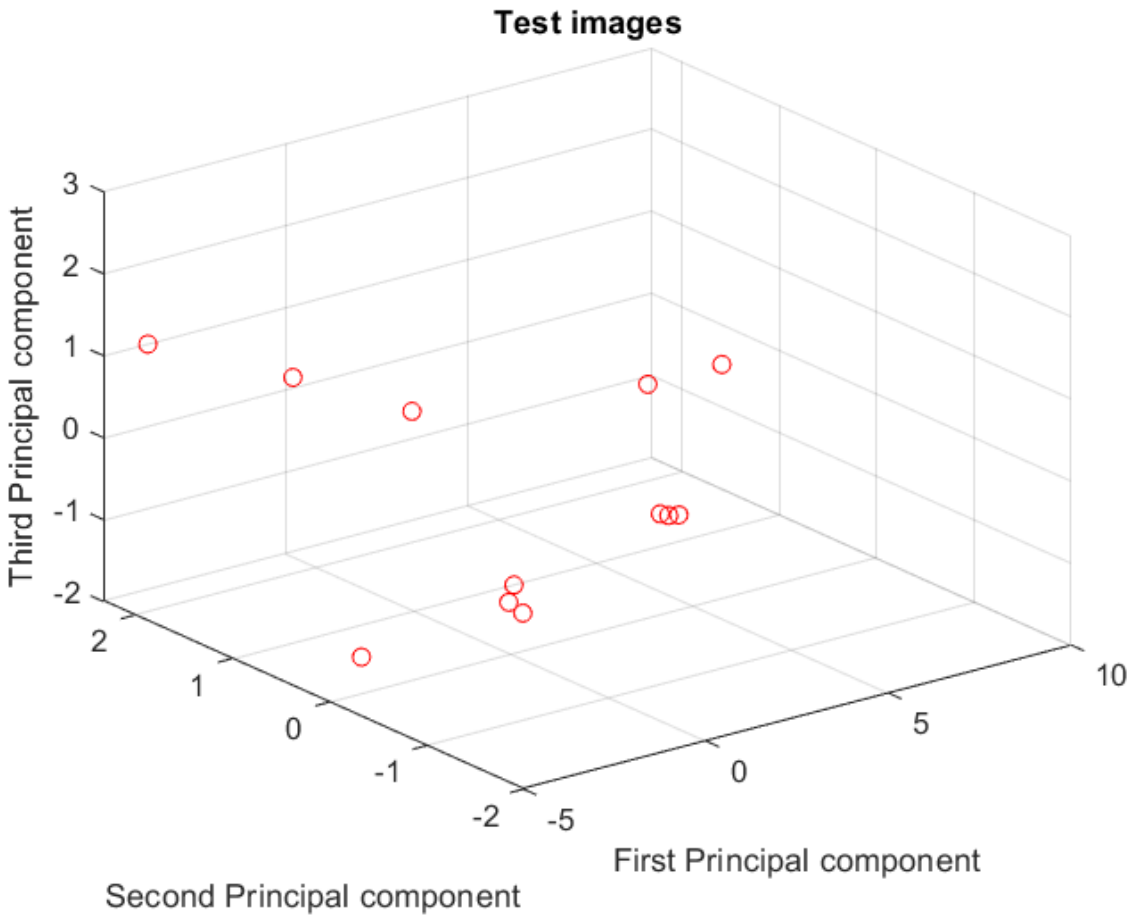


Fig 18: PCA feature space obtained for test images

```
Discriminant Power of Training Images at 15x1 dimension

4596151.78864419  largest = smallest Discriminant power
115296.233579597
4818.14631091614
11464.1276973614
55229.4367542832
6662.96981221648
4454.72594287281  smallest = largest discriminant power
68398.2969391776
20437.3379378597
21543.6225831010
195331.990390247
9421.35559467312
179912.287535774
62223.5270677784
28751.4856397918
```

Discriminant Power of Test Images at 15x1 dimension

4301928.88552279 largest = smallest Discriminant power
102004.662047193
14285.4891412847
20782.6346908159
46177.3919713746
12217.6720042773
11004.9029027849
49308.6547570079
18825.2684335700
7828.01716901589 smallest = largest discriminant power
384436.342007262
9276.61299854877
64504.1415609768
12780.3391651624
12711.6092784249

eigen values of covariance matrix from PCA of Training images

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

eigen values of covariance matrix from PCA of Test image

3.6321 + 0.0000i
0.5156 + 0.0000i
0.3575 + 0.0000i
0.0499 + 0.0000i

```
  0.0192 + 0.0000i
  0.0074 + 0.0000i
  0.0012 + 0.0000i
  0.0003 + 0.0000i
  0.0000 + 0.0000i
  0.0000 + 0.0000i
  0.0000 + 0.0000i
  0.0000 + 0.0000i
 -0.0000 + 0.0000i
  0.0000 + 0.0000i
  0.0000 - 0.0000i
```

KNN on PCA Training	SVM on PCA Training	RF on PCA Training
1	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
1	1	1
1	1	1
1	1	1
0	0	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
2	2	2
2	2	2
2	2	2
2	2	2
2	2	2
2	2	2
2	2	2
2	2	2
2	2	2
3	3	3
3	3	3
3	3	3
3	3	3

3	3	3
3	3	3
3	3	3
3	3	3
3	3	3

Fig a : Prediction results on training datasets for different learning algorithms

KNN on PCA Test	Kmeans without PCA on Test	Kmeans with PCA on Test	SVM ON PCA Test	RF on PCA Test
2 Grass	1 Grass	1 Grass	2 Grass	2 Grass
2 Grass x	3 Grass x	1 Grass x	2 Grass x	2 Grass x
1 Rice x	4 Blanket	4 Rice x	1 Rice x	1 Rice x
0 Bricks	3 Bricks	3 Bricks	0 Bricks	0 Bricks
1 Rice	2 Rice	4 Rice	1 Rice	1 Rice
2 Grass	1 Grass	1 Grass	2 Grass	2 Grass
0 Bricks	3 Bricks	3 Bricks	0 Bricks	0 Bricks
1 Rice	2 Rice	4 Rice	1 Rice	1 Rice
1 Rice	2 Rice	4 Rice	1 Rice	1 Rice
3 Blanket x	3 Bricks	4 Rice x	0 Bricks	0 Bricks
0 Bricks x	3 Bricks x	2 Blanket	0 Bricks x	0 Bricks x
2 Grass	1 Grass	1 Grass	2 Grass	2 Grass

Fig b : Prediction results on test dataset for different learning algorithms

Error Rate for Training Images

```
The train accuracy for KNN On PCA is 94.4%
The train accuracy for SVM on PCA is 97.2%
The train accuracy for RF on PCA is 100%
```

Error Rate for Training Images

```
From Fig 16,
Blanket = [2, 11] 3 is unknown due to the PCA separate cluster.
Grass = [1,6,12]
Brick = [4, 7, 10]
Rice = [5, 8, 9]
The test accuracy for KNN On PCA is 66.7%
The test accuracy for Kmeans on PCA is 75%
The test accuracy for Kmeans without PCA is 83.3%
The test accuracy for SVM on PCA is 75%
The test accuracy for RF on PCA is 75%
```

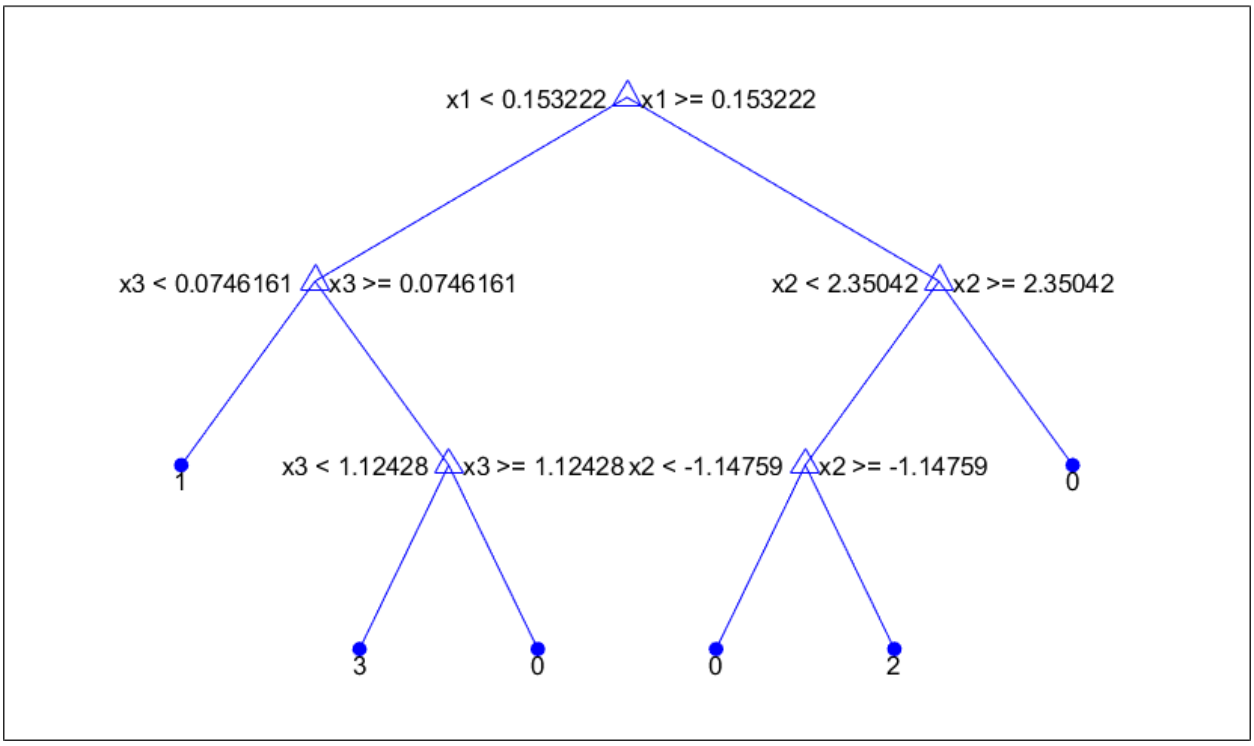


Fig 19: Decision Tree for RF Classifier obtained for test images

*II. **Discussion***

For the training images, there are 36 total image, 9 each belonging to the 4 classes of blanket, brick grass and rice.

I took the variance of the feature vectors to get the discriminant power. The largest variance value is 4596151.78864419 which is for the L5L5 filter and is thus the one with the smallest discriminant power. On the other hand, 4454.72594287281 is the smallest variance value and is for the S5S5 filter and is the one with the largest discriminant power. There is an inverse relationship with the variance magnitude and discriminant power. Small variance maps to a large discriminant power and a large variance maps to a small discriminant power.

The same applies for the test images as well. The largest value is 4301928.88552279 which is for the L5L5 filter. The smallest value is 7828.01716901589 which is for the W5W5 filter, and thus has the largest discriminant power.

Here, using the variance of the data allows us to understand how it is spread. A small variance implies that there is a large change in the value of a function while a large variance implies a more uniform convergence. Here we see that L5 is the one with smallest/weakest discriminant power and this makes sense because being a low-pass type of filter, the L5 will give little information about boundaries and edges. S5 in this case is a high-pass filter, so it has more discriminant power and gives more information on the edges and boundaries.

In figure 17, cyan represents Grass, blue represents Rice, green represents Bricks and Red represents Blankets. The cyan-colored points and the blue-colored points are very closely clustered, and this implies that they are similar in features. This colors map to the rice and grass images and when we look at it from a human perspective, it is an intuitive comparison. The Green points are more spread out and when we check the training images for the bricks, we see that this is because there are more varied types of brick images. For the red dots, there are 3 separate clusters around the feature map, and this makes sense because in our training data for blanket, blanket7,8 and 9 are all different from the rest of the images.

Another observation is the eigenvalues of the covariance matrix from the PCA. The bottom 3 are all 0 and we choose the first 3 values as our top choice as they are not very correlated and are not less or equal to 0. The same applies to the test image covariance matrix.

For the accuracy of the test and training data, Fig a and b outline the predictions from the algorithms. For testing, Kmeans without PCA seem to give the highest accuracy of 83.3% while SVM and RF and Kmeans with PCA are next best with an accuracy of 75%. This is rather interesting as RF and SVM perform well on the training images giving an accuracy of 100% and 97.2%. I think this is due to the larger dataset for training and the smaller dataset for testing. To improve on the accuracy for testing, more test dataset is needed.

I also noticed that with each run of the Kmeans, the labels are randomized, i.e it gives different classification results, some with good accuracy and some not, and this contributes to its inadequacy. In general, I think Supervised algorithms provide better results.

From Fig b, texture2, texture3, texture10 and texture11 seem to be the most difficult to classify in general with each method. Also, assuming more dataset for testing is not a problem, it seems dimension reduced features give worse results, so dimensionality reduction is not necessarily useful here.

Problem 2 : Texture Segmentation

I. Abstract and Motivation

Texture Segmentation is part of Texture analysis and it is used in computer vision tasks such as object detection. The main idea is to separate distinct texture segments or object outline in an image.

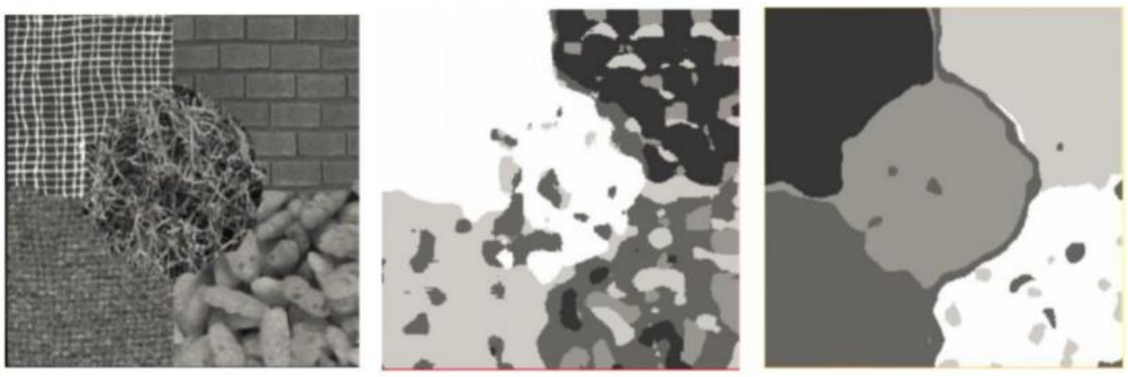


Fig 20 : Example of Texture segmentation of a composite image.[2]

Texture analysis is useful in image processing to for many tasks particularly useful in computer vision and image analysis. Some of these areas are in medical image processing such as ultrasound and CT scans, remote sensing of forests, landscapes, water bodies and even in defense applications. One thing about the need for texture analysis and classification is that there is no formulaic way of representing different texture properties, hence the need for statistical methods employed in machine learning. For this problem, I made use of K-means clustering.

One of these areas of texture analysis is texture segmentation. For textures, the general idea is that they come in repeated patterns with some randomness here and there to make it unique from different texture patterns. The main idea is to provide some algorithm to distinguish different textures combined in a pattern in a single image. An example of a suitable result is given in Fig 20.

II. Approach and Procedures

1 was employed the same approach I used in problem 1 with the use of Laws Filters, PCA, Feature reduction. The only difference was that I did not subtract the mean before applying the Laws Filters and for the obtaining the energy matrix of my feature response vectors, I calculated the average energy within a specified window sizes of 15, 23 and 39 – my results are in section 3.

The steps for **Texture Segmentation** are summarized to :

- STEP 1: Read the composite image. The image has a ***360x575*** dimension.
- STEP 2: Apply laws filters, i.e convolve each image with the 25 filters to get a ***360x575x25*** feature response vector.
- STEP 3: Obtain the Energy feature matrix by calculating the average energy of each pixel in the image within the specified window size. The default is 15. Here, the dimension of the result is a ***360x575x25*** double-type feature vector.
- STEP 4 : Reduce the feature dimension from 15 to 14 by normalizing each response with the L5'E5 filter and then discarding it from the result. Here, the dimension of the result is a 2D matrix having a dimension size of ***207000x24*** double-type feature vector.
- STEP 5: Apply the KMeans to these final feature (without PCA) matrices to obtain predicted classifications of ***12x1*** matrix of labels. Let k = 5 which denotes the number of gray levels in the composite image which havens to be 5 upon visual inspection. Result is a 207000x1 classification labels from 1 to 5.
- STEP 6: Use this result to assign the original pixels to the corresponding segmentation pixel values.
- STEP 5: Apply PCA to the result from STEP 4. The new result is a ***207000x3*** dimensional matrix.
- DO STEPS 5 and 6 for this result.
- STEP 8: Apply post-processing to fill up holes. Use MATLAB's ***imfill*** function on the resulting segmented images.
- Display the images PCA and non-PCA. I chose to use a colored segmented image for better visual inspection. The new dimension of the image is ***450x600.***

III. Experimental Results

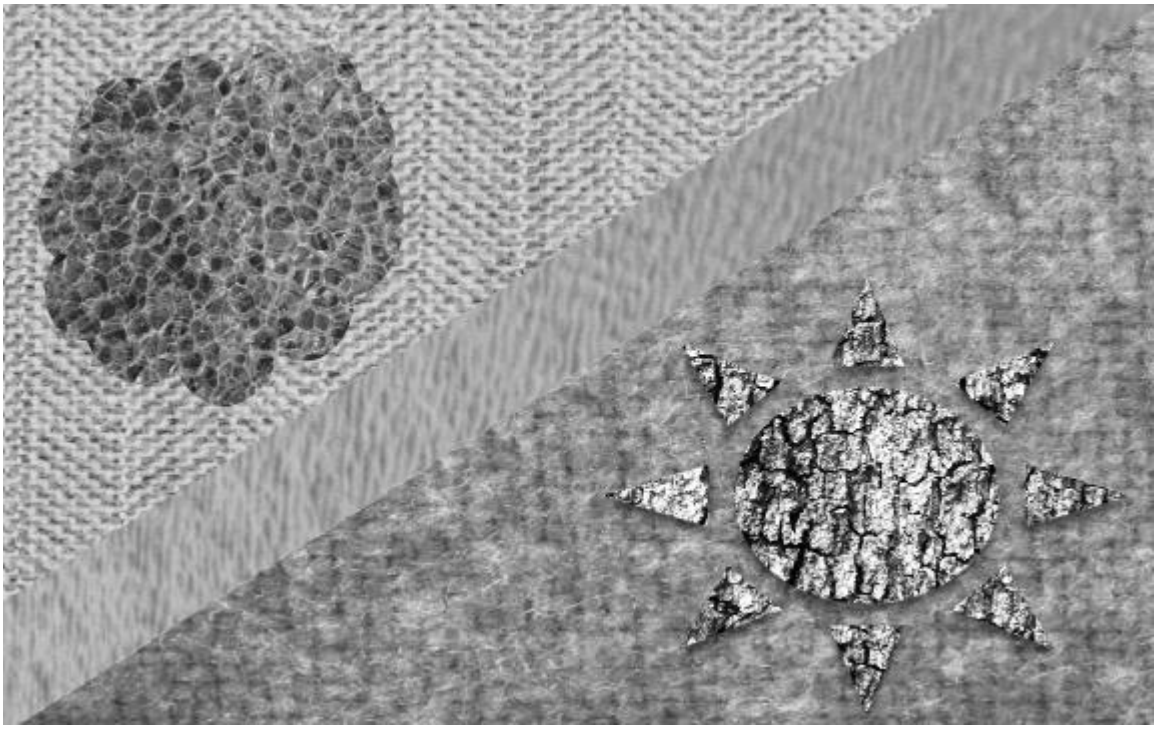


Fig 21 : composite.png, input Image for segmentation @ 360x575

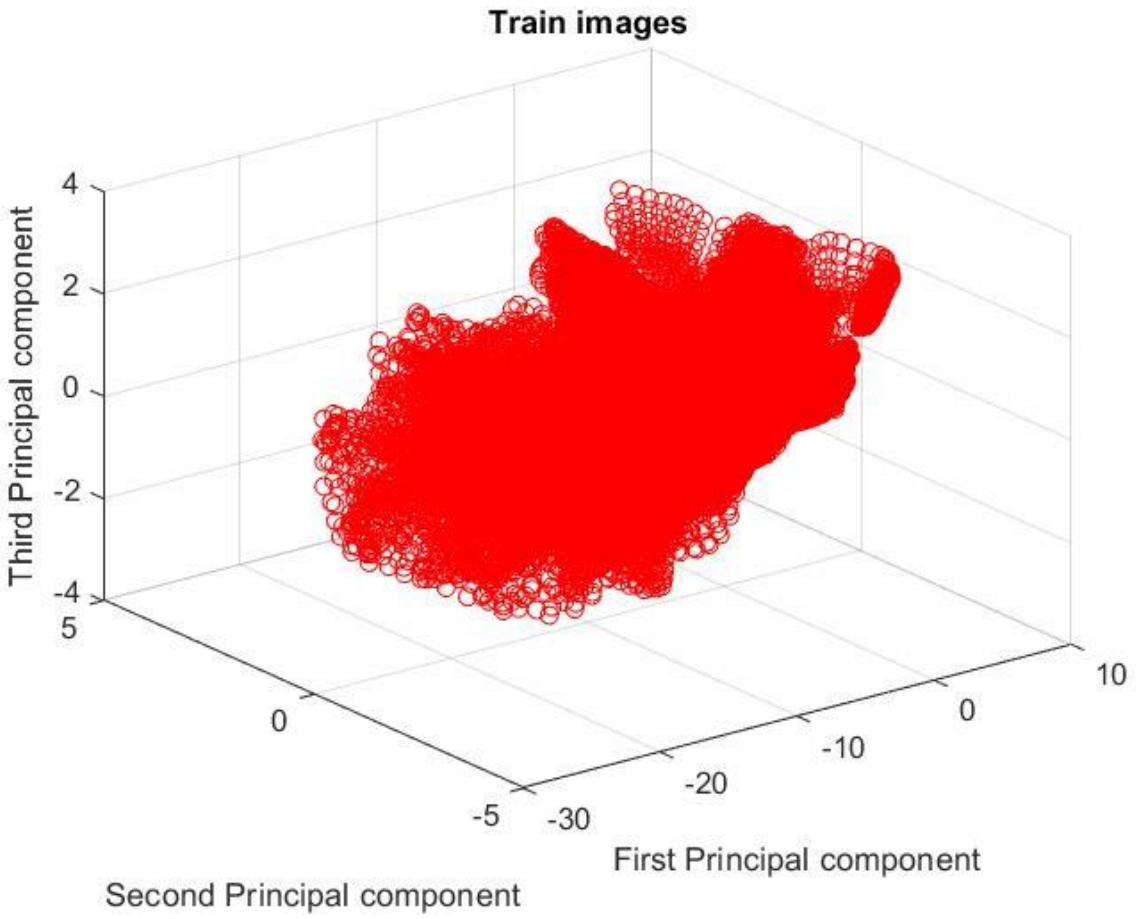


Fig 22 : PCA feature space obtained for composite image feature vector.

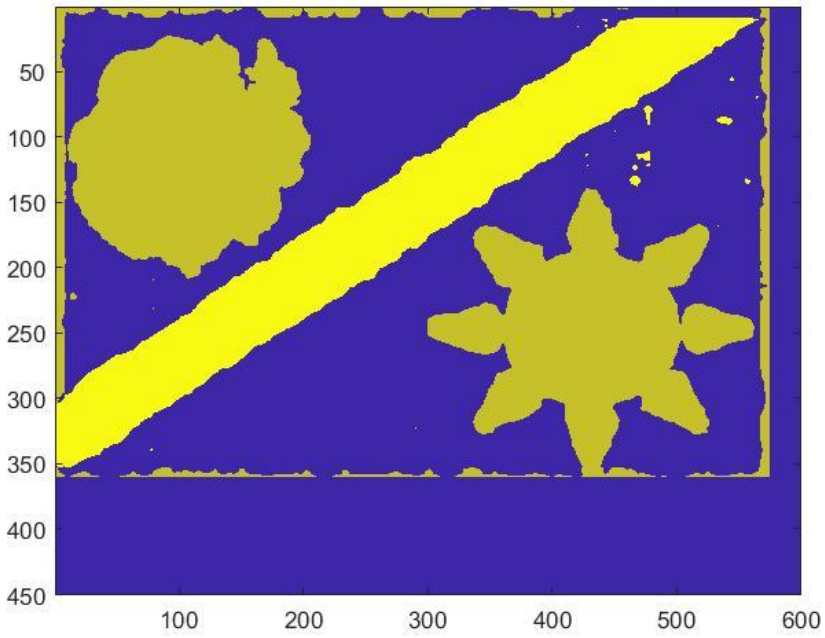


Fig 23 : Segmented Result for window size 15 on reduced feature vector of composite image

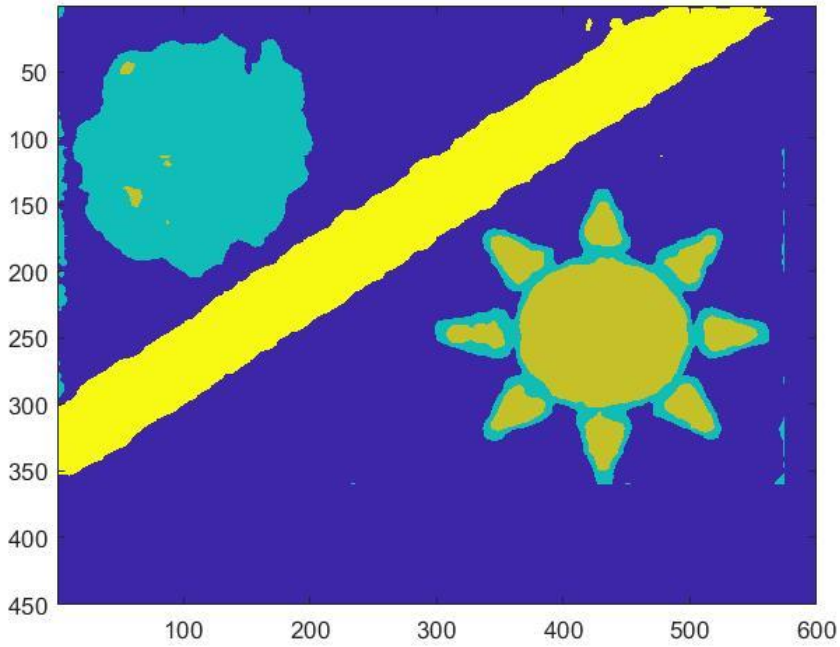


Fig 24 : Segmented Result for window size 15, with PCA on reduced feature vector of composite image

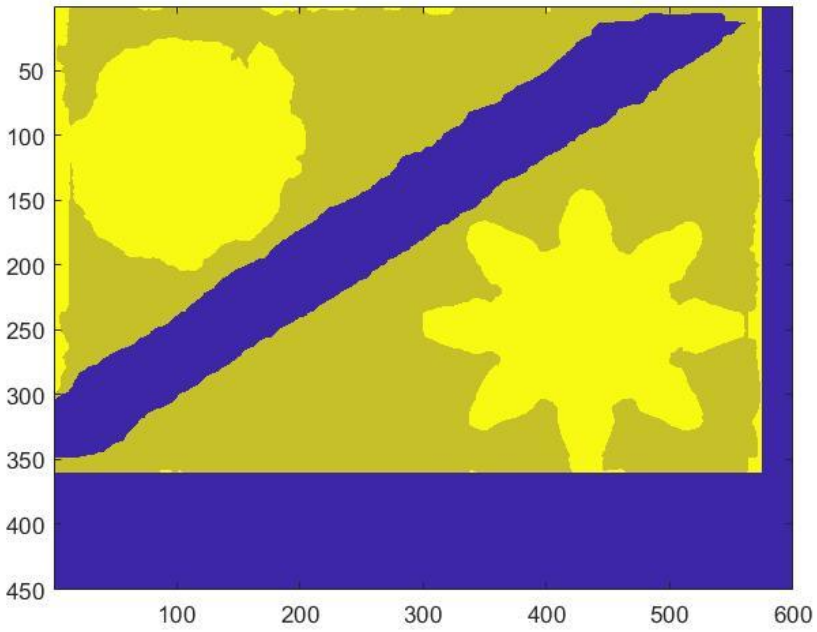


Fig 25 : Segmented Result for window size 23 on reduced feature vector of composite image

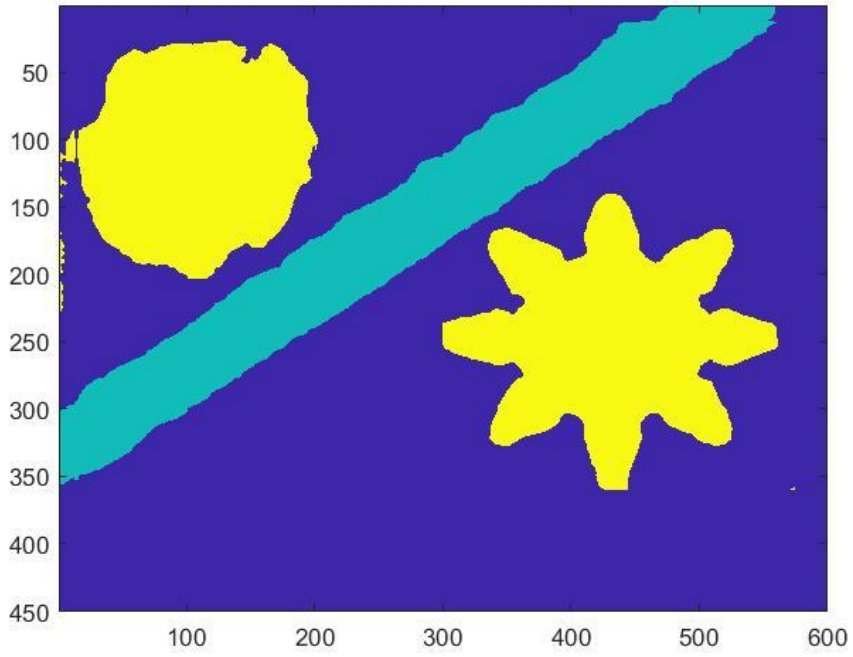


Fig 26 : Segmented Result for window size 23, with PCA on reduced feature vector of composite image

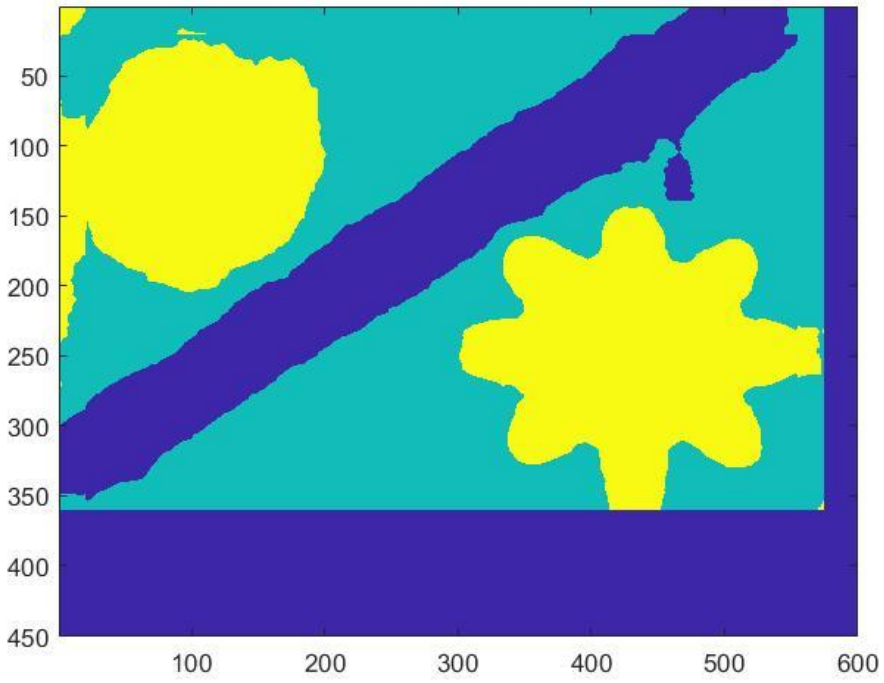


Fig 27 : Segmented Result for window size 39 on reduced feature vector of composite image

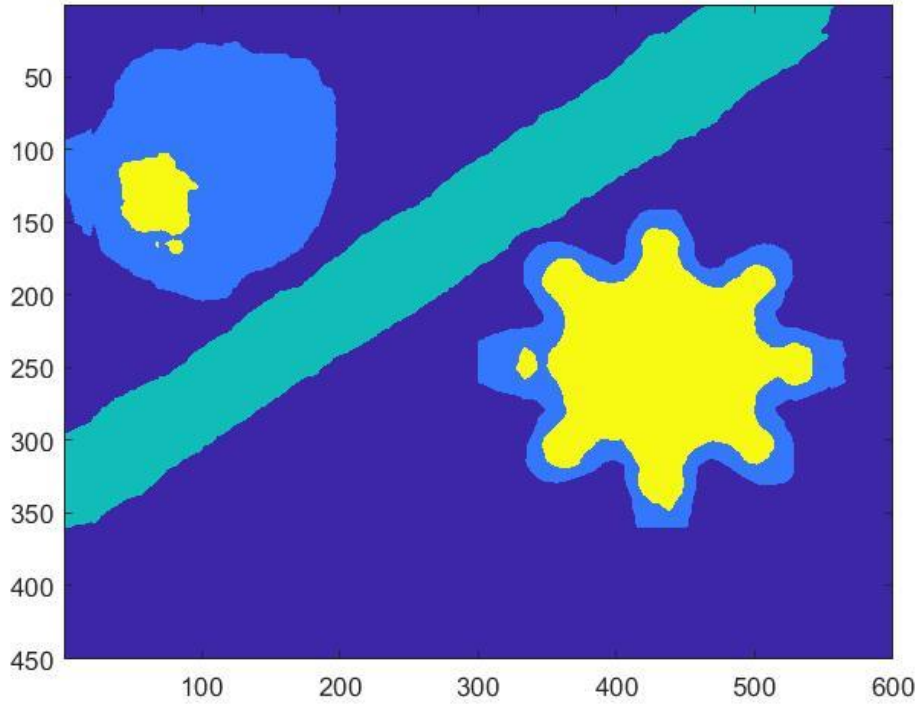


Fig 28 : Segmented Result for window size 39, with PCA on reduced feature vector of composite image

IV. Discussions

I noticed that different window sizes gave different results in terms of the visualization of boundaries. For example, for the star-like texture image at the bottom right, with larger window sizes, the sharp edges/points get more rounded. I think this makes sense as from my results with the filtering and image denoising from homework 1 where I used different filter sizes, with larger sizes causing more image blur. I think this applies to the segmented result I got. The energy feature computation using different window size imposed this change into the result. Larger window sizes make the segmentation much cleaner and less noisy. But this is not always ideal as larger window size might mean that computation time is longer, and the window size may make it less easy to segment an image because it will cover many regions of textures.

Another observation is in the use of the PCA and non-PCA features. I do not see any major observable differences in both, although I will say that the PCA result gives a less noisy segmentation, and it had a faster runtime.

Problem 3 : SIFT and Image Matching

I. Abstract and Motivation

Texture feature extraction is an important task needed by classification algorithms like KNN,SVM, RF and Kmeans. In computer vision, predictive modelling is useful in determining texture classes, and texture segmentations as shown in problem 1 and 2 respectively. For these sorts of modelling, feature extraction is the most important part and properly extracting useful features through statistical means is a challenge in the field. Although these methods gave comparable results, they were still lacking when it comes to real-world applications.

With this deficiency, David Low proposed the Scale Invariant Feature Transform (SIFT) [1] which became a major milestone for the advancement of computer vision tasks. In his work, he came up with a method that was robust enough to be flexible to scaling, rotation, illumination and even noise[1]. These, I think were some of the shortcomings of the existing feature extraction methods then, and I think this was also present in the methods I applied to problems 1 and 2.

To answer the Questions in 3A,

1. According to [1], SIFT is a robust to affine transformations such as rotation, shear and scaling. It is invariant to the change in orientation, illumination variation and noise addition. SIFT simply transforms a given image to become scale invariant.
2. SIFT achieves this by transforming an image to scale invariant coordinates. This means that it creates a scale space and from Fig 30, for each of the 5 octave responses, the Laplacian of Gaussian is created. It scales the image down by a factor of 2, 4, 8 and 16 which gives these different octaves. For rotation, it does the keypoint rotation assignment discussed in detail on the next pages. In this step, it assigns an orientation to each keypoints for an image which represents its direction. For this, the 16x16 window is convoluted to get the gradient magnitude and orientation of each pixel at every 10 degrees. As a result, 36 bins ae created and this becomes the histogram of the magnitude and orientation. From this, we get the orientation of the maximums.
3. For illumination changes, we simply normalize to enhance robustness. Here, the final N-D feature vector is normalized to a unit length. This makes the feature

invariant to illumination. Any new addition in form of a constant to a pixel will not give any significant changes due to the differences obtained from the gradient values.

4. One advantage of using Difference of Gaussian over Laplacian of Gaussian is that the former gives a much closer approximation to the later. To make it faster and efficient on computation resources, SIFT uses DoG instead of LOG as LoG is computationally intensive.
5. The SIFT vector output size in the original paper is 128-Dimension, gotten from the division of the 16x16 window into 4x4 groups, in which the gradient magnitude for each is computed to give a 4x4x8 feature vector of 128.

To understand the SIFT method, I will touch on the following listed concepts : Keypoint localization, Keypoint descriptor, Scale Space Extrema Detection, and Orientation.

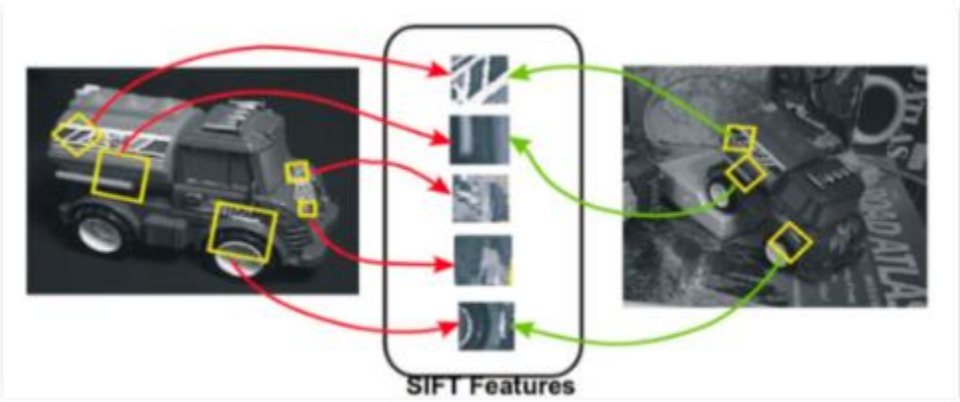


Fig 29: SIFT Features

Scale Space Extrema Detection

The idea here is to build a sort of scale space for the image we need to extract the features from, which are called the Keypoints. This is a method to obtain varied responses using different of sigma values. Convolution of image with Gaussian kernels of different variance values, resulting in different smoothened images. We must find the difference between these images and one another, or the Laplacian of Gaussian of the image.

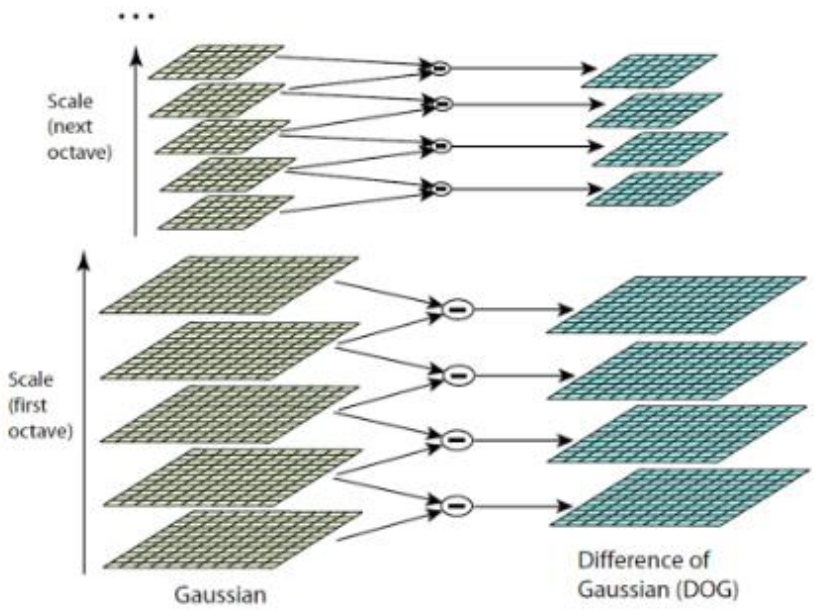


Fig 30: The Laplacian of Gaussian derived from the Difference of Gaussians

Smoothed image: $\mathrm{L}(x, y, \sigma) = G(x, y, \sigma) * I(x, y, \sigma)$

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

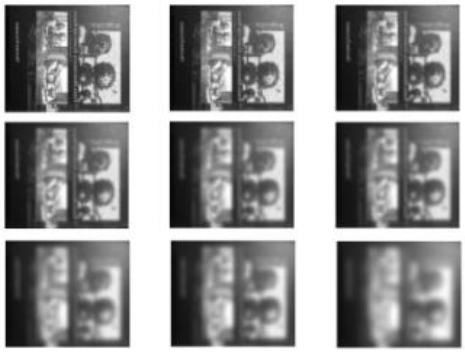


Fig 31: The different smoothened images from applying different sigma values

Using 5 as our scale number, we obtain the Laplacian . Next, we select the Keypoints by comparing each of the pixel in one octave (from Fig 30) with the 26 pixels from top and bottom scales. A pixel is a Keypoint if it is a local minima or maxima, as seen in Fig 33. This is called finding the extrema.

An approximation: $\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x,y,k\sigma)-G(x,y,\sigma)}{k\sigma-\sigma}$

$\rightarrow G(x,y,k\sigma) - G(x,y,\sigma) \approx (\mathrm{k}-1)\sigma^2\nabla^2 G$

→ extrema location depends on $\nabla^2 G = 0$, k is constant, σ^2 is scale normalization (Typical values: $\sigma = 1.6; k = \sqrt{2}$)

→ scale-invariant

Sample point is selected only if it is a minimum or a maximum of these points

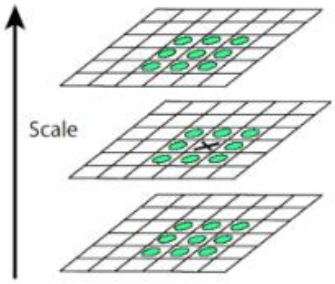


Fig 33 : Finding Keypoints/ extrema

Keypoint Localization

Here, the outlier keypoints are removed by initial outlier elimination and further outlier elimination. The first comes in form of a partial differential equation (PDE) and deals with removing the low contrast pixels. The second allows the removal of edge maps as DoG has strong response along the edges. This requires that we compute the Hessian Matrix for the DoG. This step gives a much cleaner keypoint frequency. We only eliminate the keypoints that have a ratio greater than r = 10.

Orientation Assignment

An important step for the SIFT method is to do orientation assignment and it involves making the features invariant to rotation. For this, we compute central derivatives, gradient magnitude and direction of LoG at a given keypoint like so:

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

.

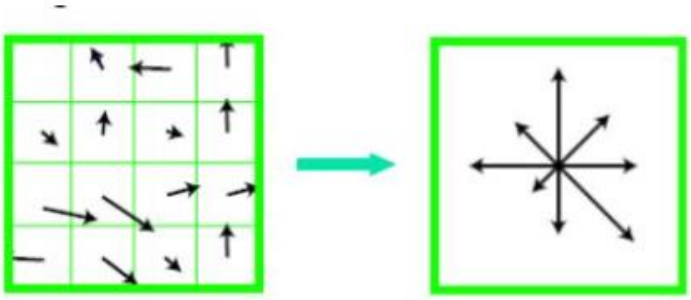


Fig 34 : Orientation Assignment

Keypoint Descriptor

We need this step to match for similar features of two images; we obtain the corresponding keypoints of each including the descriptors, then choose a keypoint from an image and find the mean square error or L2 between related descriptor and all descriptors of the image which means that the point that is nearest will be a match. For this problem, I found out that using the ratio of 0.8 works well for all the images.

Bag of Words

Simply put, we wish to represent an image as a histogram, or a bag of words as shown in the figure on the next page. To explain further, we wish to find the histogram intersection of some number of images with an image (Dog_3 in this case) by using the histogram intersection method as shown in Fig 36

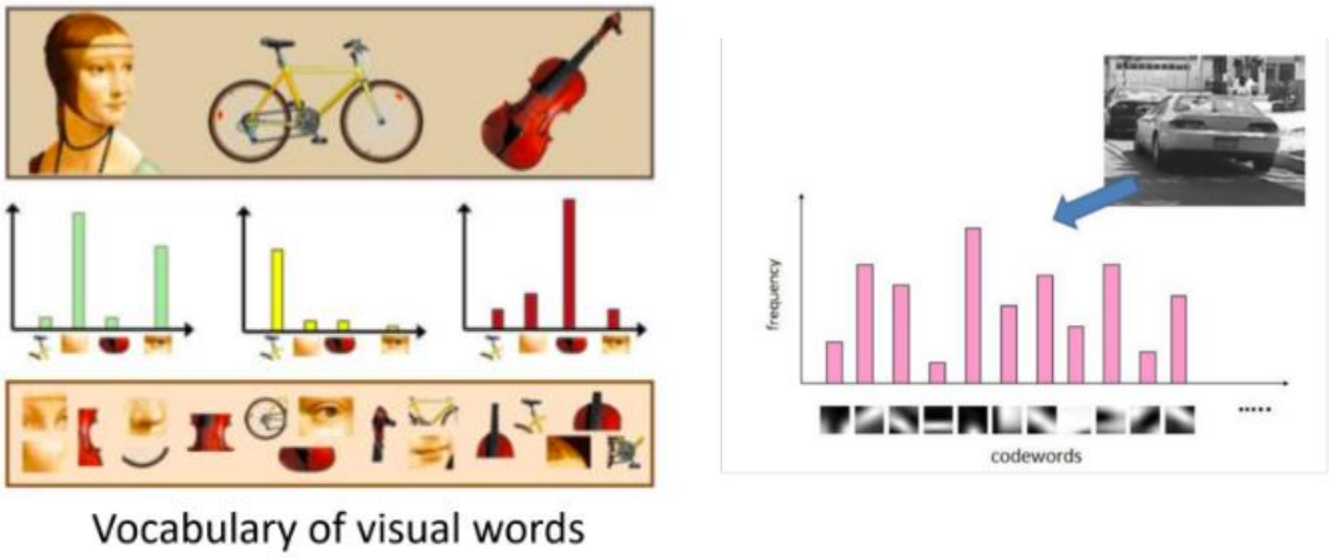


Fig 35 : Bag of Words

Fig

$$Similarity\ Index = \frac{\sum_{j=1}^{k} \min(I_j, M_j)}{\sum_{j=1}^{k} max(I_j, M_j)}$$

Where I_j is one of the three image's codewords histograms, M_j is the Dog_3 codeword histogram, and k is 8.

Fig 36: Histogram Intersection method

II. Approach and Procedures

The steps for **Image Matching using SIFT** are summarized to :

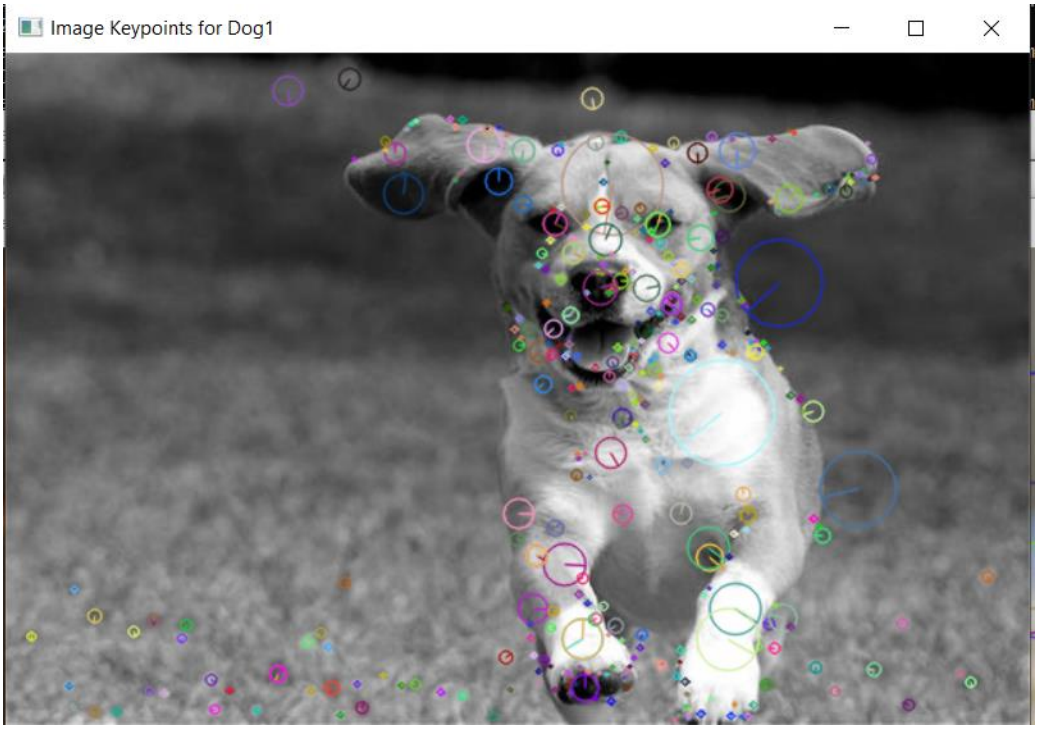
- STEP 1: Install OpenCV for Visual Studio : very delicate process, refer to documentation on official site.
- STEP 2: Read all the 4 images i.e Dog_1 Dog_2, Dog_3 and Cat. Convert them to Grayscale for SIFT procedure.
- STEP 3: Extract the keypoints and descriptors using the OpenCV SIFT functions and use the Kmeans to get the good matches (reduce the number of keypoints to just relevant ones). Basically, we compare the distance between the descriptors for our chosen image matches and choose those with the smallest distance only.
- STEP 4: Draw the matches between the images. I did those for Dog1-Dog2, Dog1-Dog3, Dog2-Dog3, Dog1-Cat, Dog3-Cat.
- For reporting purposes, I also displayed the keypoints and descriptors on each image separately.

The steps for **Bag of Words** are summarized to :

- STEP 1: Install OpenCV for Visual Studio : very delicate process, refer to documentation on official site.
- STEP 2: Read all the 4 images i.e Dog_1 Dog_2, Dog_3 and Cat. Convert them to Grayscale for SIFT procedure.
- STEP 3: Extract the keypoints and descriptors using the OpenCV SIFT functions
- STEP 4 : Use Kmeans or specifically, use the BOWKMeansTrainer from OpenCV to get he 8 centroids of the images for Dog1, Dog2 and Cat. These become our codebook.
- STEP 5 : Get the Histogram count for each image using the method described in Fig 36. Here ,we find the histogram intersection with Dog3 image.
- Plot the histogram. Shown in experimental results.

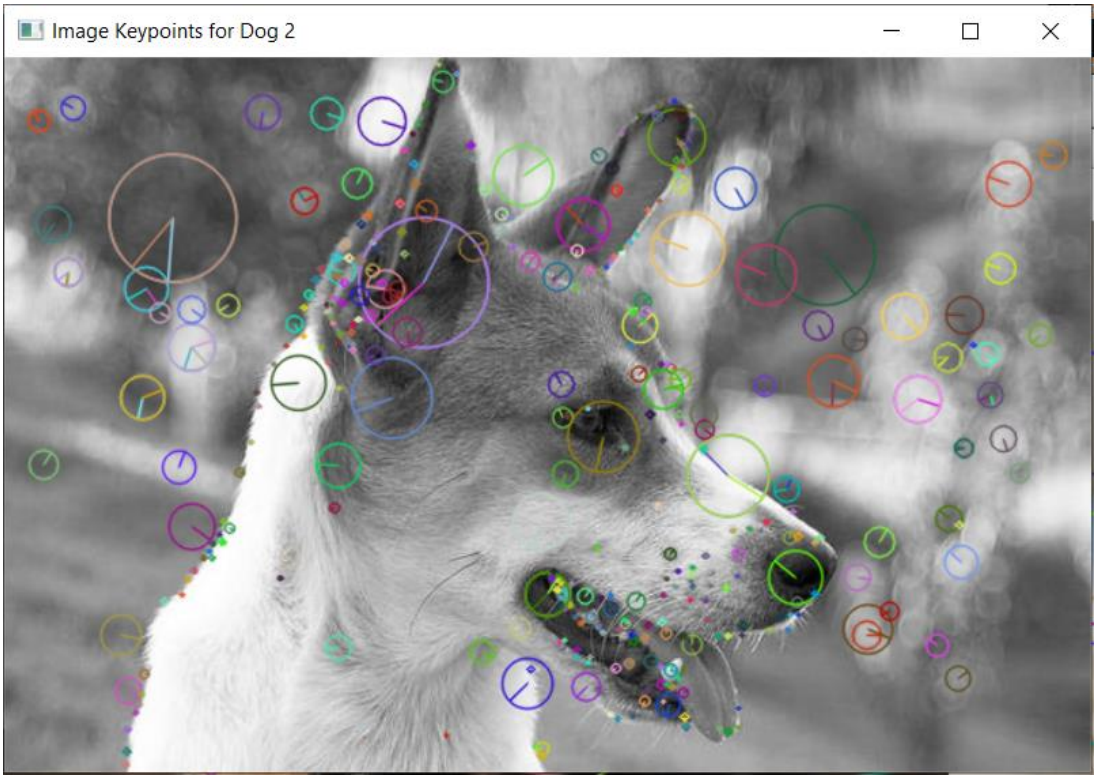
III. Experimental Results

Dog_1 Image



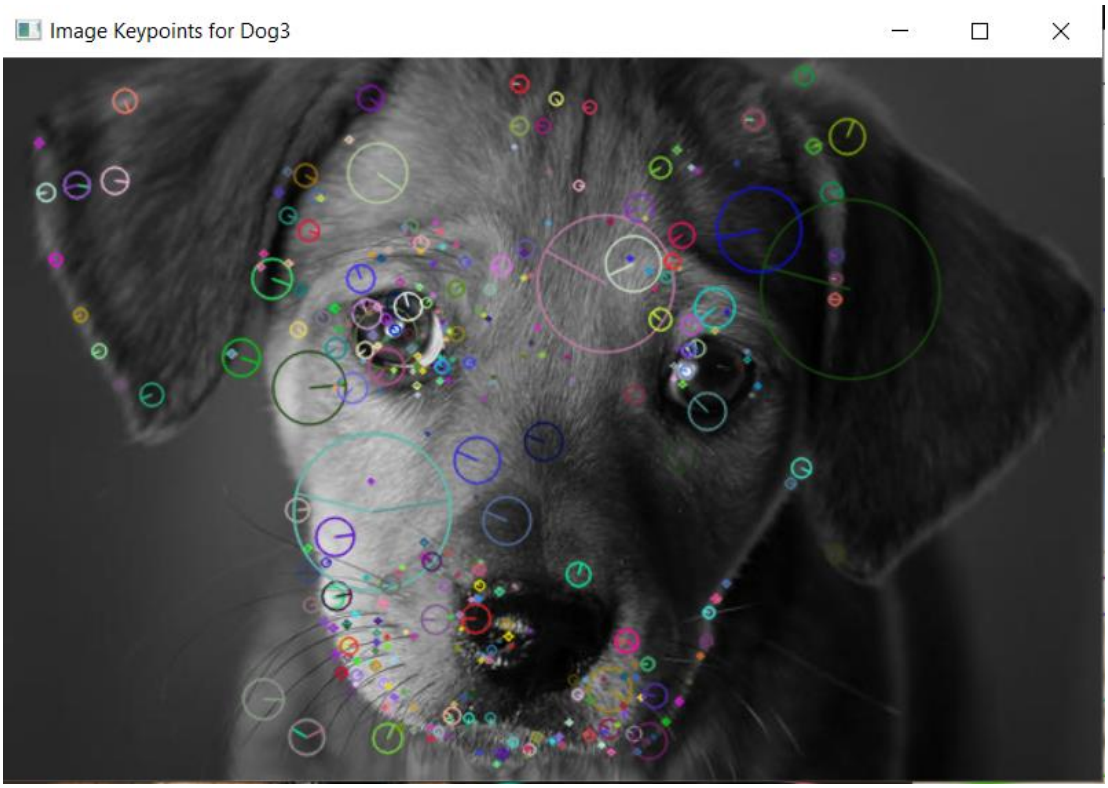
Keypoints on Dog_1 Image, grayscale

Dog_1 Image



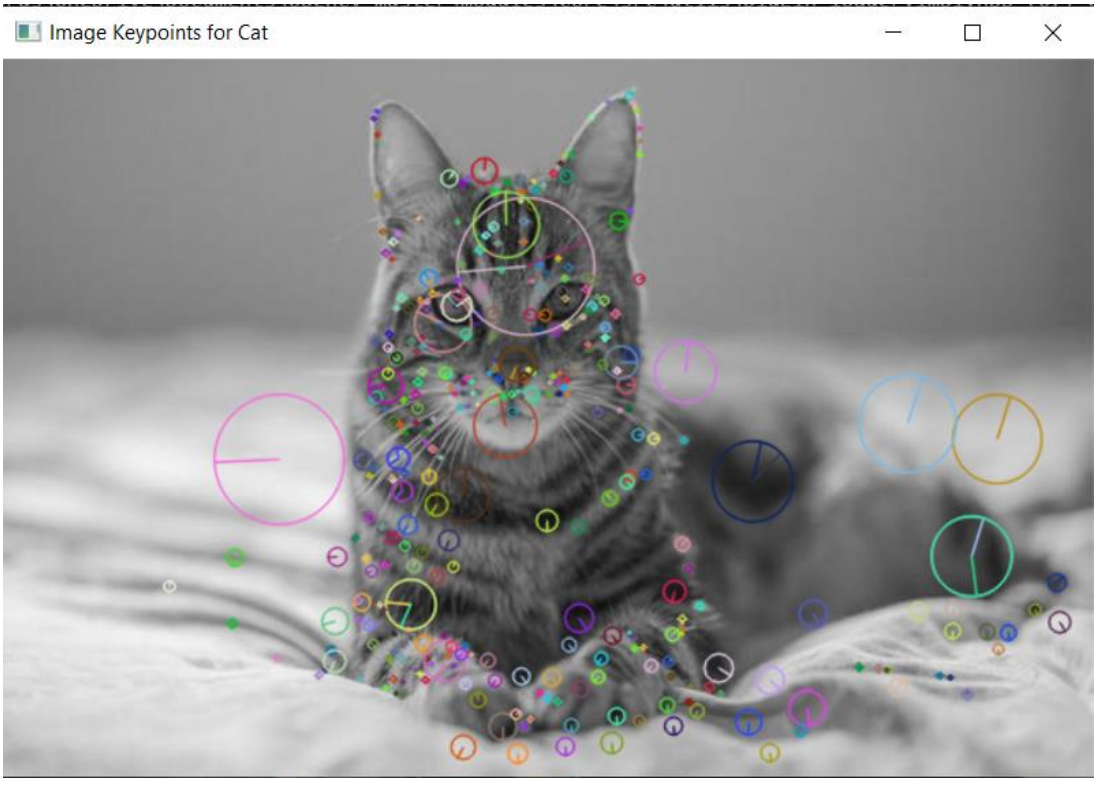
Keypoints on Dog_2 Image, grayscale

Dog_3 Image



Keypoints on Dog_3 Image, grayscale

Cat Image



Keypoints on Cat Image, grayscale

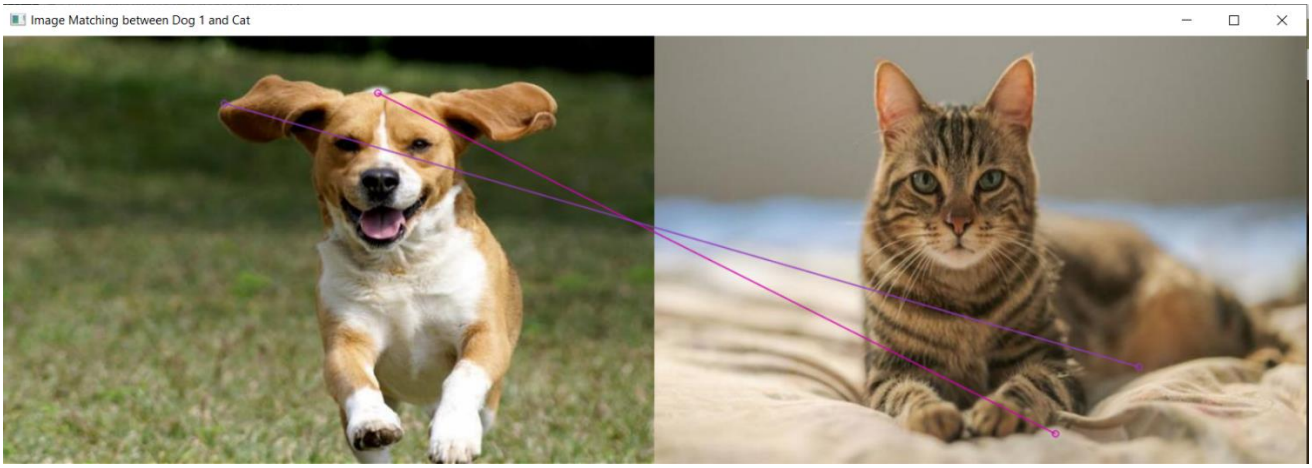


Image Matching between Dog_1 and Cat @ 0.7 ratio

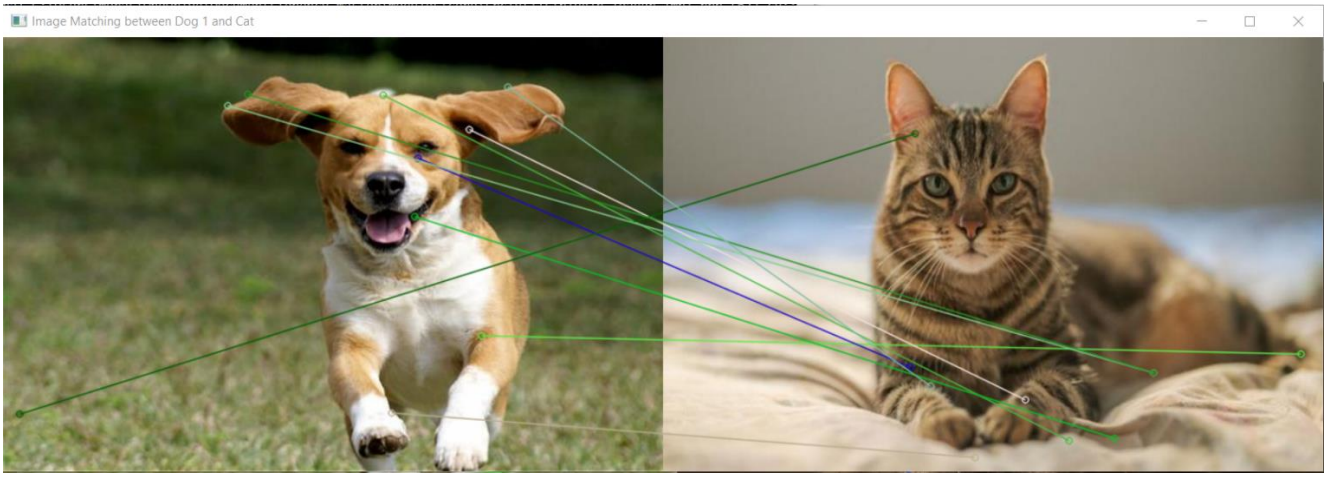


Image Matching between Dog_1 and Cat @ 0.8 ratio

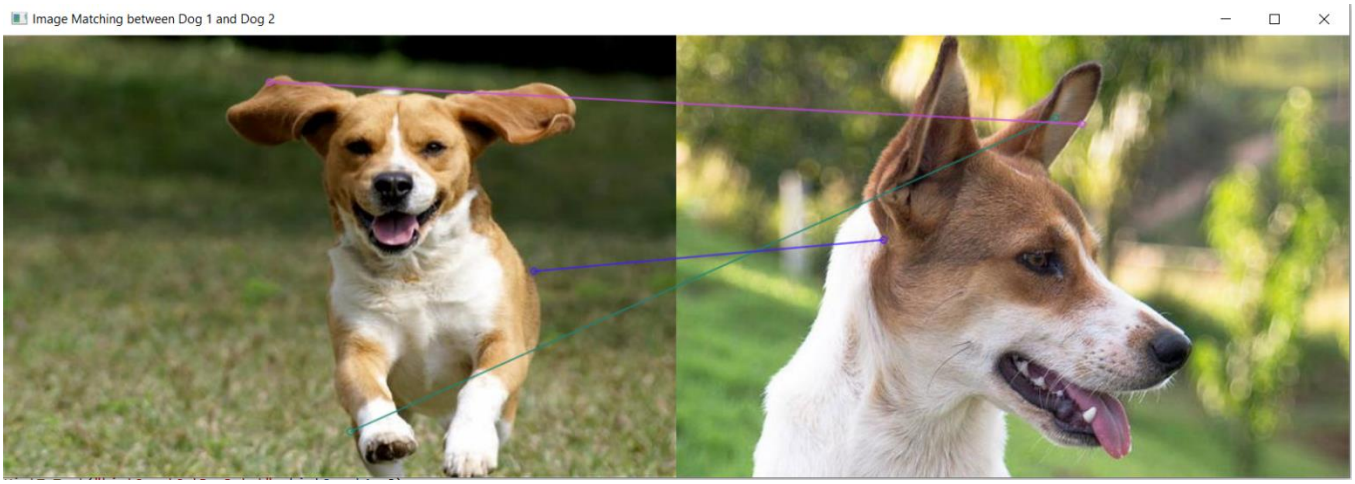


Image Matching between Dog_1 and Dog_2 @ 0.7 ratio

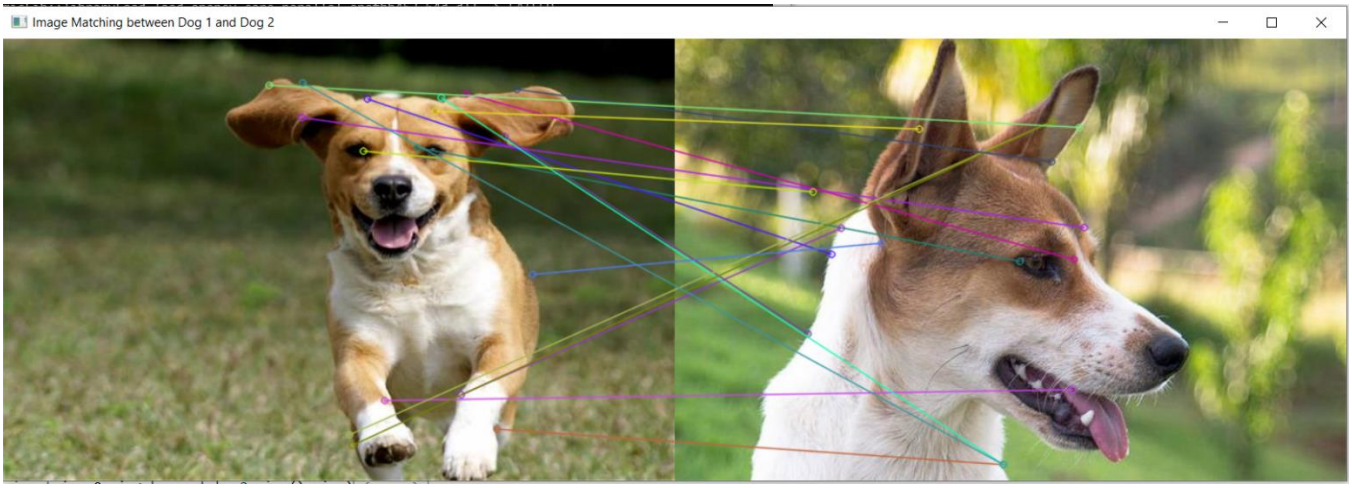


Image Matching between Dog_1 and Dog_2 @ 0.8 ratio

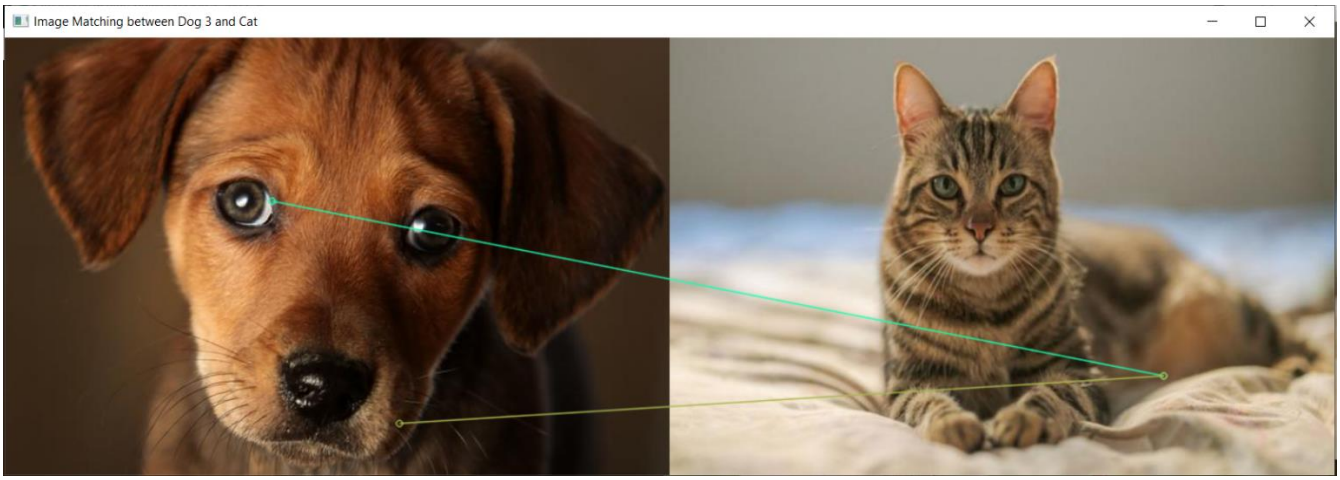


Image Matching between Dog_3 and Cat @ 0.7 ratio

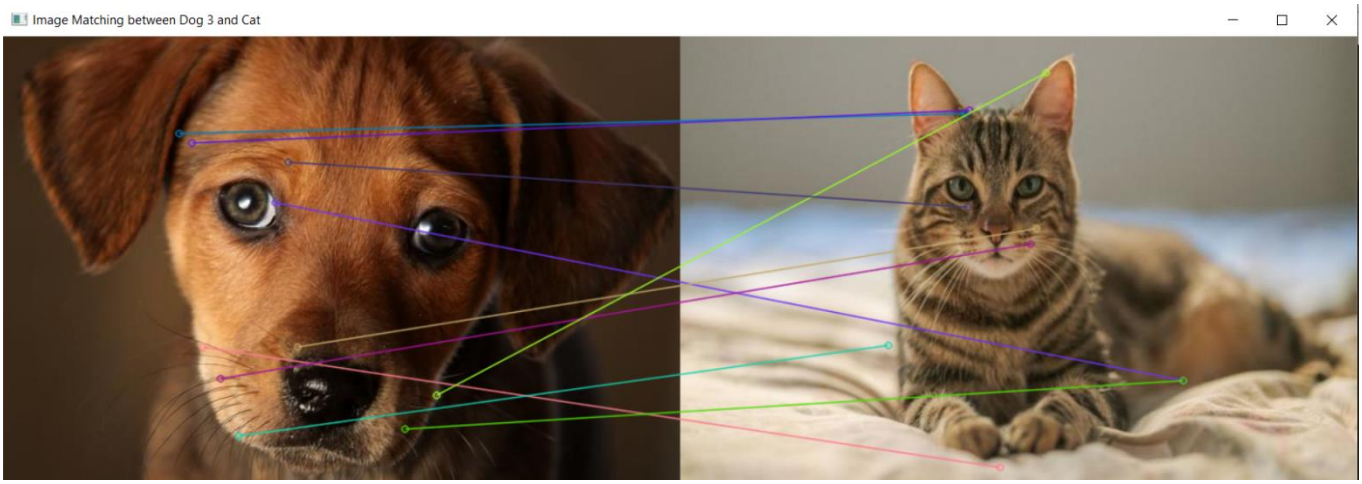


Image Matching between Dog_3 and Cat @ 0.8 ratio

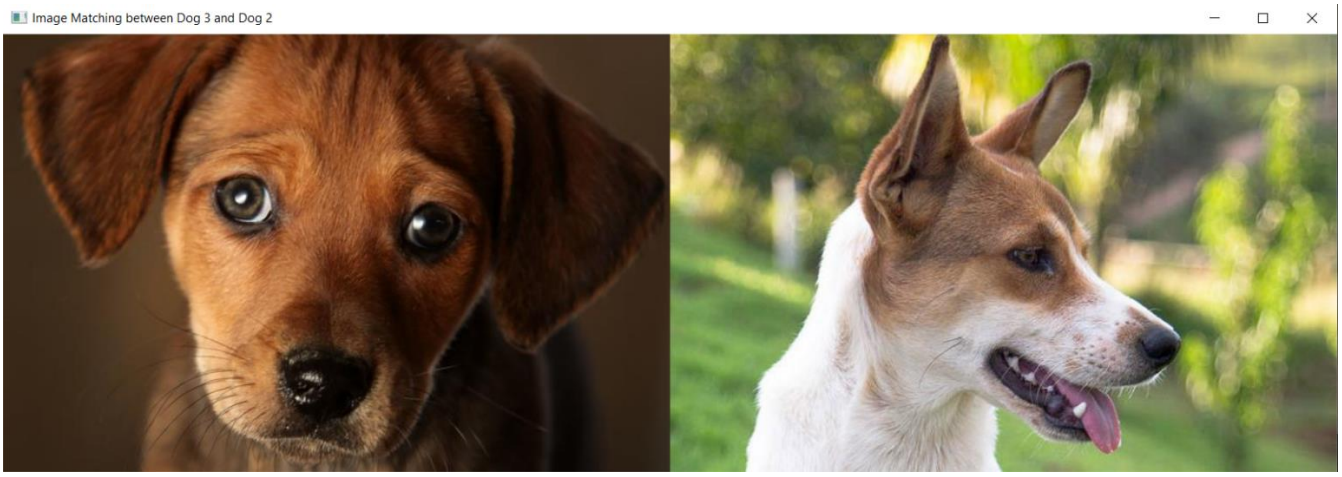


Image Matching between Dog_3 and Dog_2 @ 0.7 ratio

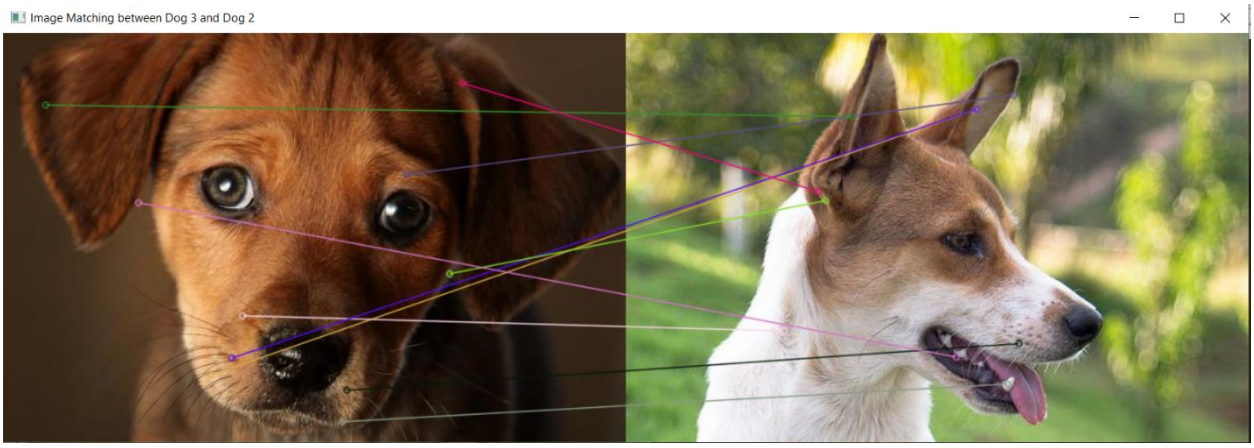
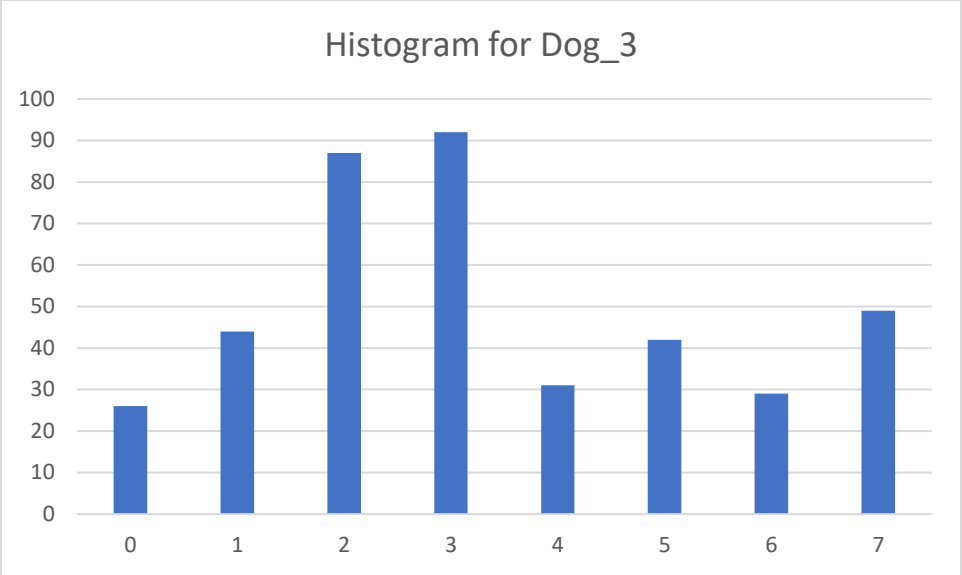
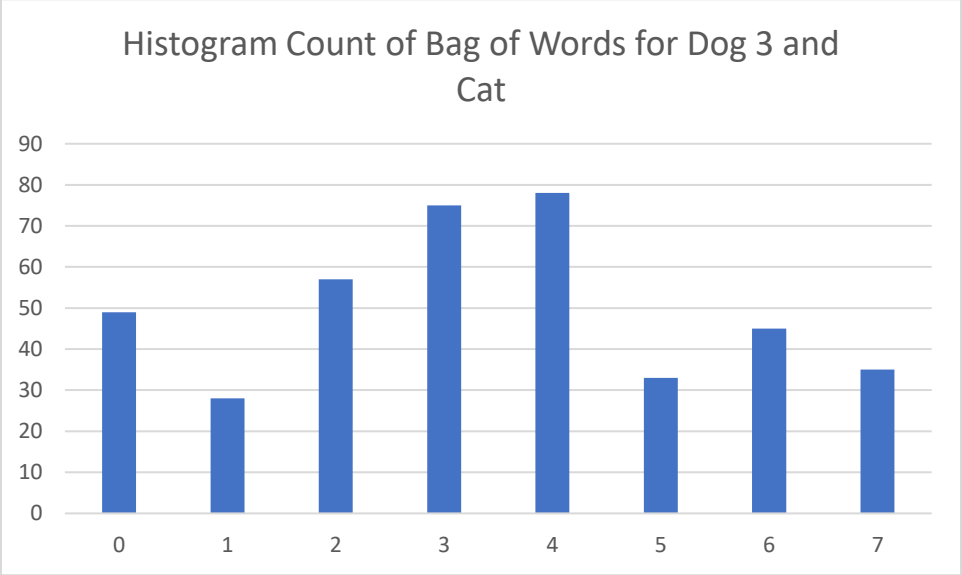
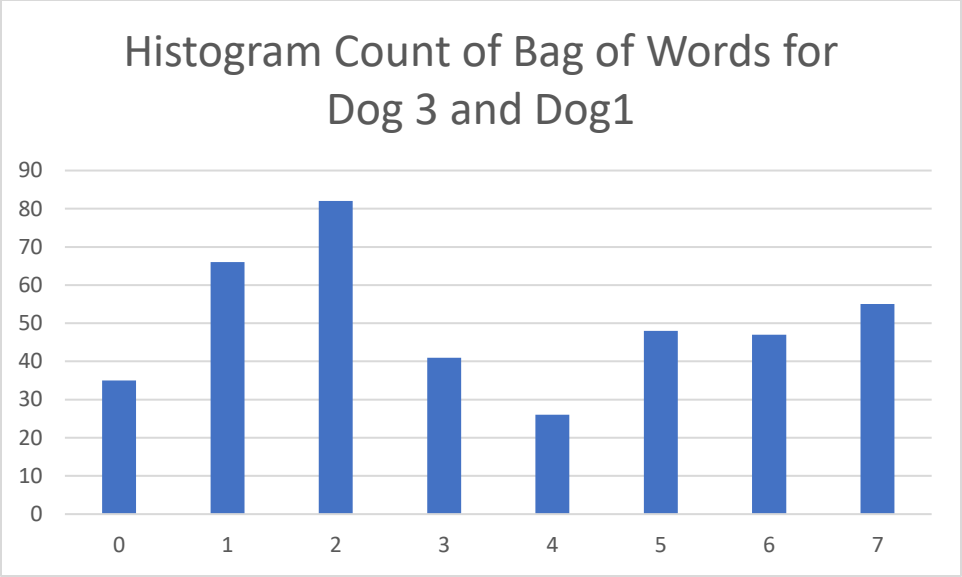
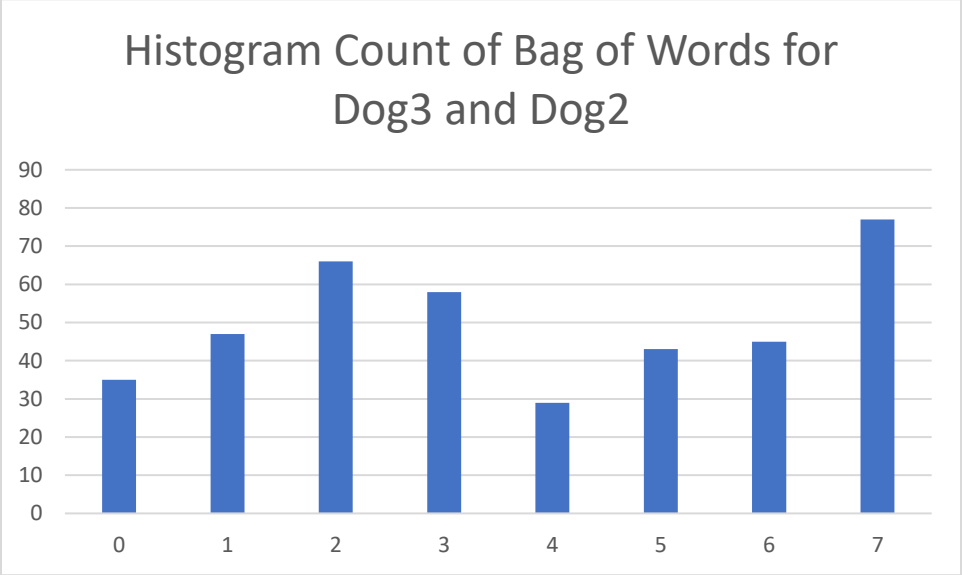


Image Matching between Dog_3 and Dog2 @ 0.8 ratio


Histogram for Dog_3
100
90
80
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
7


Histogram Count of Bag of Words for Dog 3 and Cat
90
80
70
60
50
40
30
20
10
0
0
1
2
3
4
5
6
7





IV. Discussions

One of the issues with kmeans, which I mentioned in my previous discussion is that is has randomized initialization and gives different label results for the classes for each run. So to

overcome this for the BoW, we simply sort the center of the clusters with respect to the distance between the centroids of Dog_3 and we obtain a histogram with respect to its codebook.

I noticed that for threshold of 0.7, there were less keypoints matching for the two images, with no matching for matching at all for Dog_3 and Dog_2. Switching to a threshold of 0.8, I got more keypoints matching with the images.

For Dog_3 and Dog_2, at 0.8 threshold, there were some good matches and some wrong matches, for example, the ears and whiskers match as a good, correct match but for a bad match, there are some keypoint matches where the eyes of Dog_3 match to the ears of Dog_2. As for getting no match at a 0.7 threshold, this is because of the mismatch in texture and the dark and light transitions for the Dog_3 having majorly brown pixels and Dog_2 having sharp transitions from white to green. Upon visual inspection, the matches are more than the mismatches for that at 0.8 threshold.

For Dog_3 and Cat, at 0.7 threshold, there are a total of 2 mismatches and no matches as the eyes and whiskers of Dog_3, are matched to a blur area on the cat's body. For a 0.8 threshold, we get more matches for the ears, eyes and whisker areas on both images and some mismatch on the dog to the background in the Cat image. I believe this is due to the change in camera view – there are some up-close and blur points.

For Dog_1 and Cat, the same mismatch in Dog_3 and Cat is observed with the mismatch in the ears on Dog_1 to the background and blur area on the Cat image, all at 0.7 threshold and there are only 2 keypoint matches even if they are mismatches. For a 0.8 threshold, we get more keypoint matches although there are more mismatches than matches, with only one correct match to the cat's hind legs and an obvious mismatch on the cat's ear toa background (grass) in the Dog_1 image. When the camera viewpoint is different, we seem to get more mismatches than correct matches. I think the blur area is contributing to the mismatch. One explanation for this is that in camera viewpoint is a non-linear affine transform while scaling, translation and rotation are linear-affine transforms. And SIFT works well on those.

For the Bag of words, Cat has 3 intersections with Dog_3 image at bins **[6, 4 , 0]**. Dog_1 has 5 intersections with Dog_3 at **[0, 1, 5, 6 ,7]**. Dog_2 has 3 intersections with Dog_3 at **[0, 1, 5, 6]**. So, the order can be denoted as

Dog_1 > Dog_2 > Cat.

Cat has the most change in camera view and least number of correct keypoint matches. So, this order makes sense. Dog_1 and Dog_2 differ in their number of intersections by 1 and this is explained by the number of correct keypoint matches we see on close visual inspection.

REFERENCES

[1] David G. Lowe, “Distinctive image features from scale-invariant keypoints,” International Journal of Computer Vision, 60(2), 91-110, 2004

[2] Lecture Notes, Discussion Notes, Piazza posts and Homework handout.

[3] MATLAB “kmeans” Function https://www.mathworks.com/help/stats/kmeans.html

[4] MATLAB “fitcknn” https://www.mathworks.com/help/stats/classification-nearest-neighbors.html?s_tid=CRUX_topnav

[5] MATLAB “fitceoc” Function https://www.mathworks.com/help/stats/fitcecoc.html

[6] OPENCV SURF function
https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html

[7] Wikipedia

[8] KNN A Simple Introduction to K-Nearest Neighbors Algorithm | by Dhilip Subramanian | Towards Data Science

[9] SVM Support Vector Machine — Introduction to Machine Learning Algorithms | by Rohith Gandhi | Towards Data Science