

**EE 569**  
**Homework #4:**  
**Texture Analysis and Segmentation and Image  
Feature Extractors**

**Name: Zhiwei Deng**  
**Date: 3/19/2020**

# Contents

<b>1. PROBLEM 1.....</b>	<b>3</b>
1.1 TEXTURE CLASSIFICATION.....	3
1.1.1 Abstract and Motivation.....	3
1.1.2 Approach and Procedures.....	4
1.1.3 Experiment Result.....	10
1.1.4 Discussion.....	13
1.2 TEXTURE SEGMENTATION.....	17
1.2.1 Abstract and Motivation.....	17
1.2.2 Approach and Procedures.....	18
1.2.3 Experimental Result.....	19
1.2.4 Discussion.....	21
<b>2. PROBLEM 2.....</b>	<b>24</b>
2.1 SALIENT POINT DESCRIPTOR.....	24
2.1.1 Robustness to Geometrical Modifications.....	24
2.1.2 How to achieve the Robustness.....	24
2.1.3 Robustness to Illuminance Changes.....	24
2.1.4 DoG over LoG.....	25
2.1.5 Output Vector Size.....	25
2.2 IMAGE MATCHING.....	26
2.2.1 Abstract and Motivation.....	26
2.2.2 Approach and Procedures.....	27
2.2.3 Experimental Result.....	29
2.2.4 Discussion.....	32
2.3 BAG OF WORDS.....	34
2.3.1 Abstract and Motivation.....	34
2.3.2 Approach and Procedures.....	34
2.3.3 Experimental Result.....	35
2.3.4 Discussion.....	41
REFERENCES.....	44

# 1. Problem 1

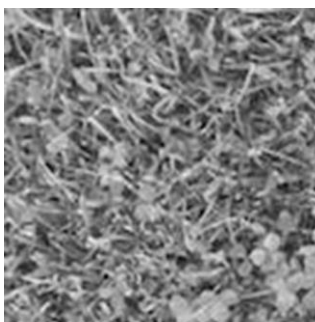
## 1.1 Texture Classification

### 1.1.1 Abstract and Motivation

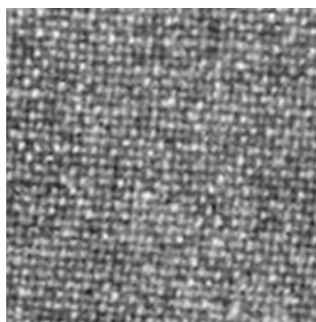
Texture is a basic property of many images. Texture classification is a basic technique that is used in many advanced image processing methods, like object recognition and image segmentation. This process can help machines to understand the world better as human views. In industries, this technique can be used in medical image analysis and industry check.

In this part, there are 2 steps to do the texture classification. First, I extracted the image features use Laws' filters and represented them with 25-dimensional vectors. Construct 15-dimensional vectors based on the pairwise filters. Then build the three-dimensional vectors using principle component analysis (PCA). Secondly, I implemented K-Means, Random Forest and Support Vector Machine algorithm to classify the 12 test images. The procedures of different methods are explained. The performance of K-means, RF and SVM are compared and the classification result is shown and discussed.

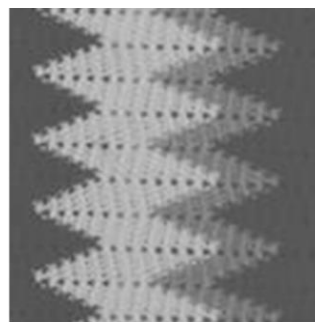
The 12 test texture images are shown as Figure 1.



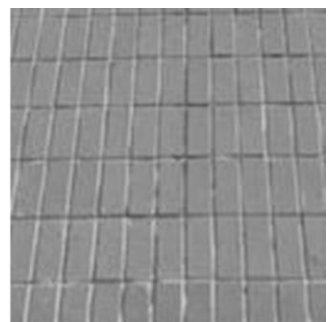
Texture 1



Texture 2



Texture 3



Texture 4

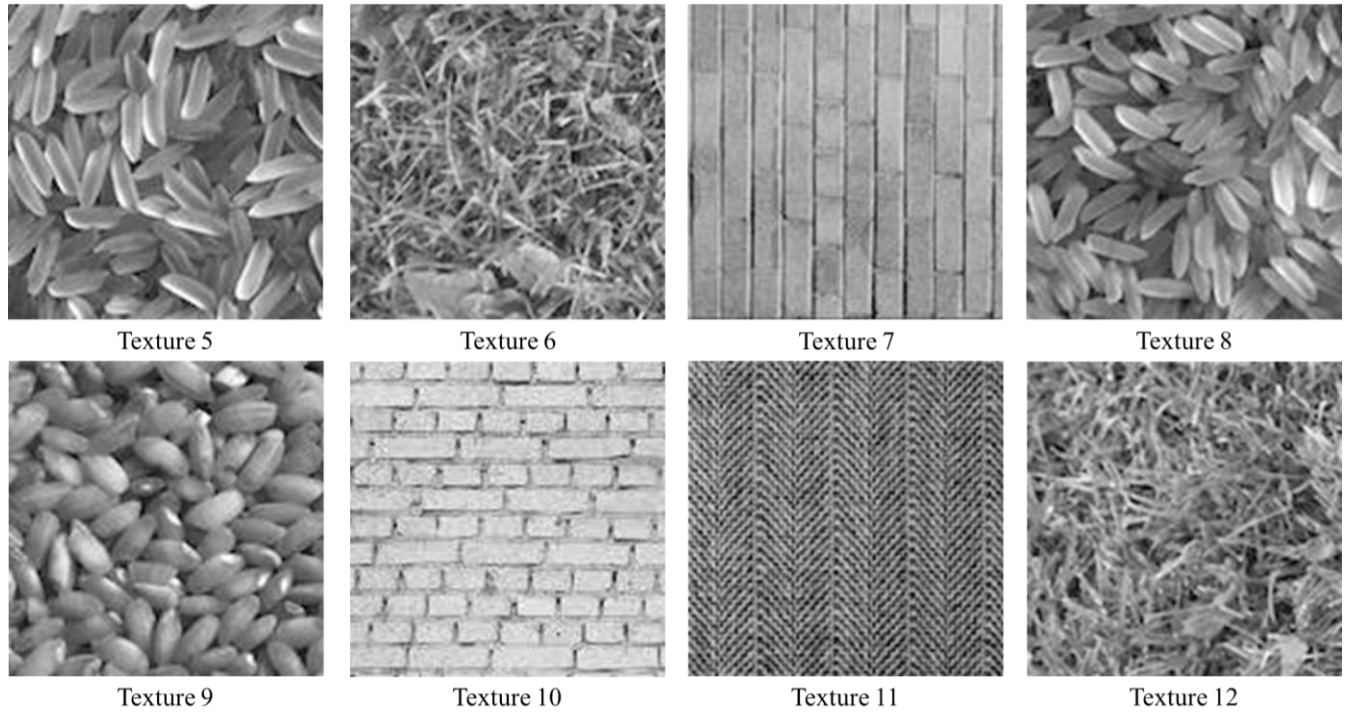


Figure 1. Test Textures

### 1.1.2 Approach and Procedures

**Laws Filters:** Laws filters are shown as 1D kernel in Table 1, and the 2D kernel of the Laws filters are shown as Table 2. The 2-dimensional filters are generated by the tensor product of two 1 dimensional kernel.

Table 1. 1 Dimensional Laws Filters

Filter Name	1-D 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. 2 Dimensional Laws Filters

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

The computation of E5L5 filter is shown in equation (1).

$$\begin{bmatrix} -1 \\ -2 \\ 0 \\ 2 \\ 1 \end{bmatrix} * [1 \quad 4 \quad 6 \quad 4 \quad 1] = \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -1 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix} \quad (1)$$

Each 1-D kernel detects one kind feature in horizontal direction, so the 2-dimensional filter detects two different features in horizontal direction and vertical direction. Like the matrix shown above, the matrix detects the E5 pattern in vertical direction and it detects the L5 pattern in horizontal direction. Apply these filters on each pixel to get the response of different features, these features can represent the whole image characteristics, which is very useful to classify the textures. Also, we can get 25 response images, where each pixel has a 25-dimensional vector to describe its features.

**Average Energy:** In this sub-problem, I use the values between 0~1 to represent the grayscale image pixel values. More specifically, the pixel values of the texture images,  $F(x)$  are computed by equation (2). Also, for every pixel, the global mean is subtracted to reduce the influences of the luminance

$$F(x) = \frac{G(x)}{255} \quad (2)$$

Since one-pixel features cannot represent the texture features, we need the whole picture of the image features to classify the textures clearly, it is important to calculate the average energies of certain image instead of the pixel energies. Also, we want to get the magnitudes of the responses rather than the signs of the responses. So, the average energies of the images are computed as following in equation (3). Of course, there are different ways to compute the average energies of images like squared sum, which also can get the magnitudes of each pixel values to avoid the positive and negative responses canceling each other.

$$\text{Average Energy} = \frac{1}{M*N} \sum \text{abs}(F(x)) \quad (3)$$

**Convert to 15D features:** There are some filters are related with each other like the E5L5 and L5E5, these two values are supposed to be same, but the two values have a slight difference in practice. To reduce the dimensions of the feature vectors, we can combine these pairs into one feature and shrink the dimension of the features from 25D to 15D. The features are L5L5, L5E5/E5L5, E5S5/S5E5, E5E5, L5S5/S5L5, E5W5/W5E5, S5S5, L5W5/W5L5, E5R5/R5E5, W5W5, L5R5/R5L5, S5W5/W5S5, R5R5, W5R5/R5W5 and S5R5/R5S5. To combine these pairs, we average the two different feature responses.

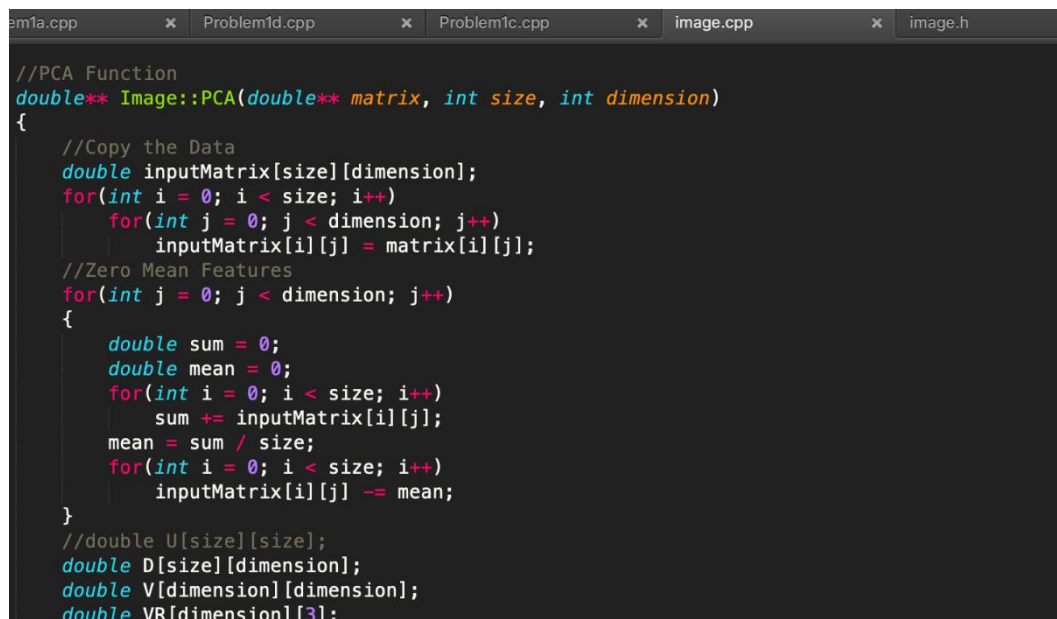
**Principle Component Analysis:** For big data processing, 25D or 15D vectors may be too large and slow to compute. Some features may not give a useful information for classifying certain textures, so we can apply the PCA procedure to find the most representative features of the data. PCA can

represent the higher dimension data using lower dimension data, with little data loss. In this part, I implemented the PCA algorithm by using the singular value decomposition (SVD). Based on the linear algebra theory, a matrix can be decomposed by 3 matrices as shown in equation (4).

$$X_{n \times m} = U_{n \times n} \times S_{n \times m} \times V_{m \times m} \quad (4)$$

The S is known as the singular value matrix, V is eigen-vector matrix. The values on the diagonal of S are the magnitudes of the corresponding eigenvectors. So, from V, we can get the eigen vectors of the data with 3 largest magnitudes, which represent the ‘main direction’ of the data. We can achieve the minimum data loss during the 15D to 3D transformation by using the top three eigenvectors. Then, we multiply X with V’ to get the dimension reduced data. To be clear, the X should be zero-mean in each feature.

**Bonus1:** The PCA algorithm is implemented by me and it works fine. Although, it works a little slower than the function in OpenCV. The code is shown as Figure 2.



```

//PCA Function
double** Image::PCA(double** matrix, int size, int dimension)
{
    //Copy the Data
    double inputMatrix[size][dimension];
    for(int i = 0; i < size; i++)
        for(int j = 0; j < dimension; j++)
            inputMatrix[i][j] = matrix[i][j];
    //Zero Mean Features
    for(int j = 0; j < dimension; j++)
    {
        double sum = 0;
        double mean = 0;
        for(int i = 0; i < size; i++)
            sum += inputMatrix[i][j];
        mean = sum / size;
        for(int i = 0; i < size; i++)
            inputMatrix[i][j] -= mean;
    }
    //double U[size][size];
    double D[size][dimension];
    double V[dimension][dimension];
    double VR[dimension][3];
}

```

```

double YR[size][3];
double** result = allocate_2D_Double(dimension, size);
CvMat featureMatrix = cvMat(size, dimension, CV_64FC1, inputMatrix);
//CvMat leftMatrix = cvMat(size, size, CV_64FC1, U);
CvMat singMatrix = cvMat(size, dimension, CV_64FC1, D);
CvMat rigtMatrix = cvMat(dimension, dimension, CV_64FC1, V);
CvMat ritMatrixR = cvMat(dimension, 3, CV_64FC1, VR);
CvMat rsltMatrix = cvMat(size, 3, CV_64FC1, YR);
cvSVD(&featureMatrix, &singMatrix, NULL, &rigtMatrix);

//Get the First K Vectors
for(int i = 0; i < dimension; i++)
    for(int j = 0; j < 3; j++)
        CV_MAT_ELEM(ritMatrixR, double, i, j) = CV_MAT_ELEM(rigtMatrix, double, i, j);

//Do the Multiplication
cvMatMul(&featureMatrix, &ritMatrixR, &rsltMatrix);

for(int i = 0; i < size; i++)
    for(int j = 0; j < dimension; j++)
        result[i][j] = CV_MAT_ELEM(featureMatrix, double, i, j);

return result;
}

```

Figure 2. PCA Bonus Codes

**K-Means:** K-Means is a simple unsupervised classification algorithm. It can divide any dimensional data into K clusters. This procedure is done by iterations and change the  $k$ th center's location by averaging the locations of all data that tagged as  $k$ . The centers' location will remain the same if the data converges into K clusters which means iteration is over. The performance of K-means algorithm is highly dependent on the initialization of the center locations. For the fairness of the randomness initial, I randomly pick K data points as the initial K cluster centers. Due to this randomness of the algorithm, the performances of K-means can vary a large range. To illustrate this more specifically, I ran this algorithm multiple times and rank the results based on the error rate low to high.

**Bonus2:** The K-Means algorithm is implemented by me and it works fine. Although, it works a little slower than the function in OpenCV. The code is shown as Figure 3.



```

//K-means
unsigned char* Image::Kmeans(double **features, int num, int size, int dimension)
{
    unsigned char* classes = allocate_ID(size, 1, 1);
    double center[num][dimension];
    //Initialize with first n positions
    int r[num];
    //srand(time(0));
    for (int i = 0; i < num; i++)
    {
        r[i] = rand() % size;
        for (int j = 0; j < 1; j++)
            if (r[i] == r[j])
                i--;
    }
    for (int i = 0; i < num; i++)
        for (int j = 0; j < dimension; j++)
            center[i][j] = features[r[i]][j];

    int iter = 0;
    bool finish = false;
    while(!finish)
    {
        iter++;
        //Create the Temp to Compare
        int temp[size];
        for (int i = 0; i < size; i++)
            temp[i] = classes[i];

        //Compute the Distance with Centers
        for (int i = 0; i < size; i++)
        {
            int index = 0;
            double mindist = 9999;
            for (int k = 0; k < num; k++)
            {
                double dist = 0;
                for (int j = 0; j < dimension; j++)
                    dist += (features[i][j] - center[k][j]) * (features[i][j] - center[k][j]);
                //dist = sqrt(dist);
                if (dist < mindist)
                {
                    mindist = dist;
                    index = k;
                }
            }
            classes[i] = (unsigned char)index;
        }

        //Update Center
        for (int i = 0; i < num; i++)
        {
            int count = 0;
            double sum[dimension];
            //Initialize
            for (int j = 0; j < dimension; j++)
                sum[j] = 0;
            //Check Tags
            for (int j = 0; j < size; j++)
            {
                if (classes[j] == i)
                {
                    for (int k = 0; k < dimension; k++)
                        sum[k] += features[j][k];
                    count++;
                }
            }
            for (int j = 0; j < dimension; j++)
                center[i][j] = sum[j] / count;
        }

        //Check the Finish Flag
        finish = true;
        for (int i = 0; i < size; i++)
            if (classes[i] != temp[i])
            {
                finish = false;
                break;
            }
    }
    if (size < 30)
    {
        for (int i = 0; i < size; i++)
            cout << (int)classes[i] << " ";
        cout << endl;
    }
    //cout << iter << endl;
    return classes;
}

```

Figure 3. K-Means Bonus Code

**Discriminant Power:** The total variance/standard deviation of each feature can represent the discriminant power of this feature. Cause the feature with bigger sigma means the data are spread in a bigger range in high dimensional space, which makes it harder to find clusters in space. While a smaller sigma means it changes a lot when the variable only changes little, which means a higher discriminant power. Also, the inter class and the intra class can more precisely define the discriminant. A higher discriminant feature should have a low intra-class variance and a high inter-class variance. Which means the same label data can cluster easily.

**Procedure:** The diagram of procedures for texture classification is shown below as Figure 4. The first half part is feature extraction of (a) sub problem and the second half is the classification of (b) sub problem.

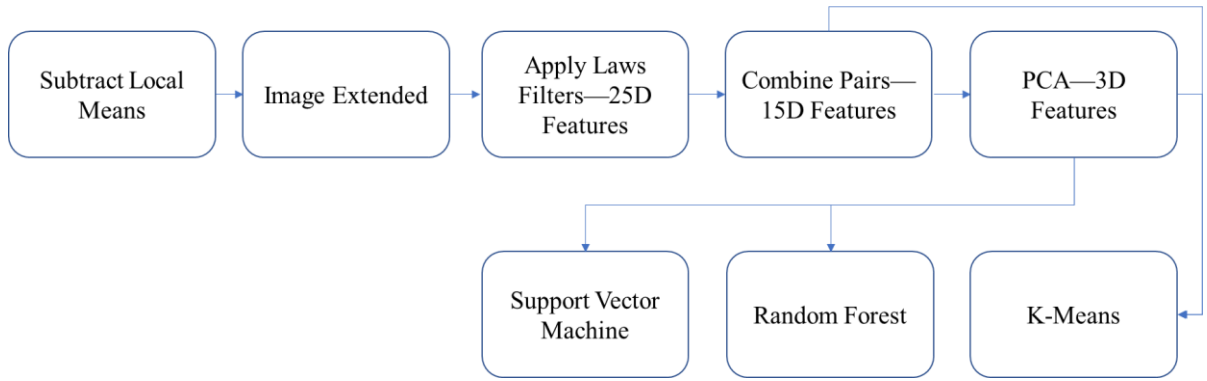


Figure 4. Procedure of Texture Classification

### 1.1.3 Experiment Result

The discriminant power of each feature is shown as below in Table 3. The strongest and weakest discriminant feature is shown in Table 4.

Table 3. Different Variance Values of 15D Features

Variance Kind	Values
Inter/Intra	[0.0262547, 0.0470041, 0.127729, <b>0.197358</b> , <b>0.0193565</b> , 0.0636389, 0.0493253, 0.0263874, 0.0430404, 0.0241554, 0.0287244, 0.0306196, 0.0213678, 0.0228823, 0.0269867]
Standard Deviation	[ <b>7.94829</b> , 1.24383, 0.25529, 0.394461, 0.860748, 0.299841, <b>0.244372</b> , 0.957487, 0.520447, 0.536042, 1.58771, 0.354777, 1.52787, 0.904839, 0.61554]
Inter/Intra(Norm)	[nan, <b>0.0180041</b> , 0.0515204, <b>0.0732671</b> , 0.0550879, 0.038594, 0.0352494, 0.0724834, 0.038246, 0.028557, 0.0617913, 0.0308909, 0.0282777, 0.0283958, 0.0306544]
Standard Deviation(Norm)	[0, 0.0751426, <b>0.0237033</b> , 0.0260711, 0.0698882, 0.0342304, 0.0296143, 0.0827721, 0.0616663, 0.0692964, 0.149322, 0.0453856, <b>0.196773</b> , 0.117379, 0.0791372]

Table 4. Strongest and Weakest Discriminant Feature

Variance Kind	Strongest	Weakest
Inter/Intra	E5E5	L5S5/S5L5
Standard Deviation	S5S5	L5L5
Inter/Intra(Norm)	E5E5	L5E5/L5E5
Standard Deviation(Norm)	E5S5/S5E5	R5R5

The projection of the 3D features after PCA of trainset is shown as below in Figure 5.

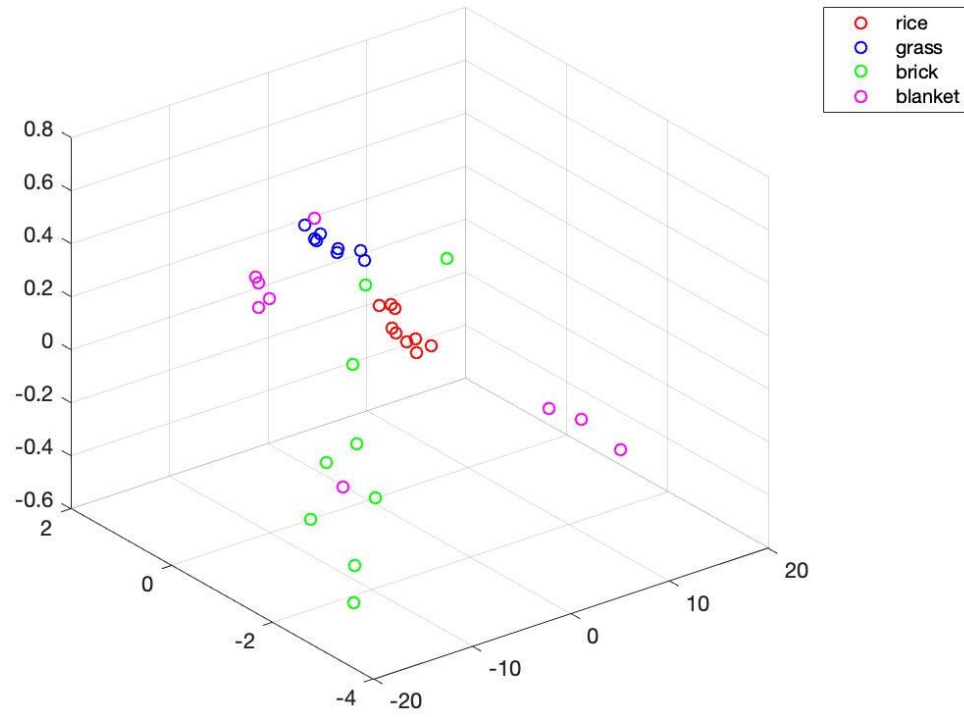


Figure 5. Trainset's 3D Feature in Feature Space

Some possible results of the K-Means process for 15D features and the correct rates are shown as Table 5.

Table 5. 15D Features K-Means Results

15D - Correct:[1, 4, 4, 2, 3, 1, 2, 3, 3, 2, 4, 1]	
Correct-Rate	Results
83.33%	[1, 2, 3, 2, 3, 1, 2, 3, 3, 2, 4, 1]
83.33%	[1, 2, 4, 2, 3, 1, 2, 3, 3, 2, 2, 1]
75.00%	[1, 4, 3, 2, 3, 1, 4, 3, 3, 4, 4, 1]
75.00%	[1, 1, 4, 2, 3, 1, 2, 3, 3, 1, 2, 1]
50.00%	[1, 2, 1, 3, 1, 1, 2, 1, 1, 2, 4, 1]
50.00%	[1, 2, 4, 3, 1, 1, 2, 1, 1, 2, 2, 1]

Some possible results of the K-Means process for 3D features and the correct rates are shown as Table 6.

Table 6. 3D Features K-Means Results

3D - Correct:[1, 4, 4, 2, 3, 1, 2, 3, 3, 2, 4, 1]	
Correct-Rate	Results
83.33%	[1, 2, 4, 2, 3, 1, 2, 3, 3, 2, 2, 1]
83.33%	[1, 2, 3, 2, 3, 1, 2, 3, 3, 2, 4, 1]
75.00%	[1, 1, 4, 2, 3, 1, 2, 3, 3, 1, 2, 1]
75.00%	[1, 2, 3, 4, 3, 1, 2, 3, 3, 2, 4, 1]
66.67%	[1, 2, 3, 4, 3, 1, 2, 3, 3, 2, 2, 1]
41.67%	[1, 2, 1, 4, 1, 1, 3, 1, 1, 2, 4, 1]

The results of supervised learning with Random Forest and Support Vector Machine is shown as below in table 7.

Table 7. 3D Features Supervise Learning

Supervised Learning - Correct:[1, 3, 3, 2, 0, 1, 2, 0, 0, 2, 3, 1]		
Methods	Correct-Rate	Results
RF	100%	[1, 3, 3, 2, 0, 1, 2, 0, 0, 2, 3, 1]
SVM	100%	[1, 3, 3, 2, 0, 1, 2, 0, 0, 2, 3, 1]

**Note:** The 3D features of test images and train images has different principle component. Theoretically, the component matrix should both be trainset's. But according the TA, this task is relatively easy and small for supervised learning, it will both work fine in these two scenarios. To extract features more effectvively I project principle component matrix of the datasets on themselves.

### 1.1.4 Discussion

**3D Features:** We can see from Figure 5, that the rice and grass images' features are relatively more clustered than bricks and blanket textures. These features can tell the difficulties of K-Means to cluster them together. For blanket image textures, it's more difficult to cluster all the data points together since they spread relatively in a wide range and into other classes. But it is more easy for the algorithm to identify the locations of rice images and grass images. This conclusion can be reached by Table 5 and Table 6, where blanket test images are often misclassified and grass and rice images have higher accuracy.

**Discriminant Power:** In this part, I calculated the different kind of the trainset image features to find the feature with the strongest discriminant power and weakest discriminant power. There are 4 kinds of variance which can represent the power. The normalized/unnormalized ratio of inter-class variance and intra-class variance, and the normalized/unnormalized total variance. The result is shown as Table 3 and 4. For the variances, a large variance means there are small change in features with a big variable shift, which will make it hard to tell one class from another. A small variance means the features change drastically even with small shift of image, which could be a good criteria to classify objects. But the variance of the feature does not give the information of the classes. The feature with smallest variance can only separate all the training images into isolated ones, but it cannot make sure the distances between the data points of same class are

small enough, which might cause a problem in classification.

To resolve this shortcoming, the inter-class variance and the intra-class variance are used. The ratio of these two values gives the distribution of each feature in same classes and in different classes. The higher inter-class variance is, the class have more clustered data points, the smaller intra-class variance is, the classes separate each other most. So, we can get the feature that can keep the data in same classes clustered and isolate each classes most by comparing the ratio of inter-class variance / intra-class variance.

**Classification Results:** Some possible results of K-Means algorithm of the test images are given in Table 5 and Table 6. From the table, we can get that the highest accuracy of the K-Means algorithm can reach more than 83%. Also, the algorithm can perform badly if the initialization is not appropriate.

The result concludes that the K-Means cannot get all images classified correctly, since some images features can distributed in a wide range and have intersection with others. That is the shortcoming of the unsupervised learning, which is hard to create a complex non-linear classification.

**Unexpected Results:** From Table 5 and Table 6, we can see that the 2nd, 3rd and the 11th texture are easily to get mis classified (tagged as red) compared to other textures. These three textures are all belong to blanket class, but the pattern of these three blankets are quite different with each other. The first one has the patterns like little squares, the second one has the patterns like arrows in the horizontal direction and the third one has the arrow patterns in vertical direction. These unlike features are more difficult for an

unsupervised learning algorithm to get the correct classification. On the other hand, the grass and rice images which have more apparent features tends to get the right classification in most cases.

**Comparison between 15D and 3D:** For the K-Means with features after PCA ( 3D features), the performance improves a little. From Table 6, we can see that the correct rate has a liitle improvement compared with the Table 5, but they still have the same highest correct rate. The 2nd and 3rd image are still easily to be misclassified. The PCA process disregards the weak and useless features that might cause intersections and overlaps, which can improve the accuracy a little. We can say that the PCA is an useful technique for K-Means cluster.

**Efficiency of PCA:** To figure out how much does PCA improve the speed of the program, I implemented a clock to see the time the algorithm uses in K-Means. Generally, for 15D feature K-Means, the program uses around 56 clocks per excution, but for 3D feature K-Means, the program uses around 42 clocks per excution. This running time is including the data processing and so on. Also, 3D feature K-Means uses fewer iterations than the 15D feature K-Means since its data has a lower dimension.

The aspect cannot be ignored of the efficiency of this PCA procedure is the size of the testset. In this problem, the testset only contains 12 images and 4 different classes of the image, which is relatively very small amount for a dataset. For a much bigger dataset, the PCA will show its efficiency more clearly, and that is why a PCA process is important in image processing tasks.

**Comparison between RF and SVM:** From Table 7, we can see that both RF and SVM can do a good work in this task. There is no error in the classification of the test data images. As we mentioned above, that this task is a relatively easy one, and the unsupervised learning can get a high accuracy. So, it is not surprising that RF and SVM make it all right. Although there is more to compare these two methods.

For RF, it is relatively faster than SVM and it can generate a more generalized model for prediction. I used other blanket image from the internet to test and it can still get the right class. But its obvious shortcoming is that it is very easy to get over fitting. For this task, I tried many parameters of the RF generator. The classification is precise when the maximum number of trees are around 10, but when the depth of trees or the maximum number of trees are too high, the precision can go down very fast. So, I think the robustness of RF is no better than SVM.

For SVM, it uses more time to train and make the prediction. But it shows a more robust property in this task. The parameters of the SVM are not selected precisely but the result can be very stable and accurate. However, it is unwise to use SVM if the data sets scales, the computation of SVM is based on the linear operations which involves the  $M \times N$  matrix computation, which requires more computation resources and time. In other words, it is less efficient than RF.



## 1.2 Texture Segmentation

### 1.2.1 Abstract and Motivation

Beyond the classification techniques of the whole texture image, texture segmentation is another useful skill in the industry and research field. The ideas of this two methods are similar, for the texture segmentation, it needs to classify the every pixel of the image rather than the whole texture features.

I used the Laws Filters to extract the features from the image to create 25 filtered images. Then the pairwise combination and L5L5 based normalization is applied to all features. At last, the pairwise filtered pixels are classified by the K-Means algorithm that shows above. For the post process, I used PCA to reduce the dimensions of the pixel features and see whether the performance is improved or not. After the experiments, the effects of different window sizes will be discussed and the performances are going to be compared. The test image is shown as Figure 6.

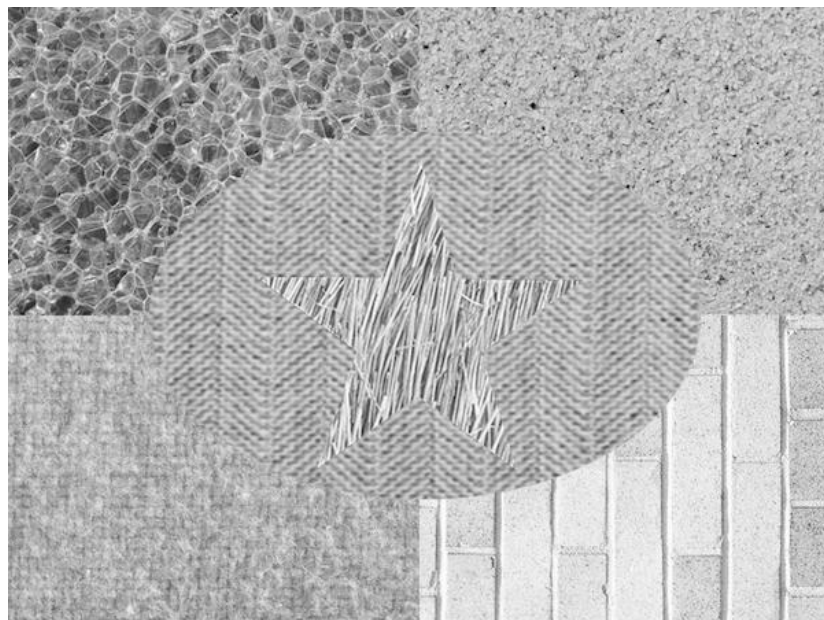


Figure 6. Comp.raw

### 1.2.2 Approach and Procedures

**Feature Normalization:** Different features of the same image cannot be compared effectively since the ranges of different features vary differently. So, in order to make the features more comparable, we need to do the normalization. Also, from the structure of the 2D Laws Filters, we can easily get that the L5L5 is the only feature does not have a zero mean, which makes it can represent the overall energies of the input image. So, to make all input image features has a same scale, we use L5L5 value to normalize all other feature values. The equation si shown as below in equation(5). Through this procedure, we can get 14 normalized feature values for each pixel in the range of 0~1.

$$NewFeature = Feature / L5L5 \quad (5)$$

**Energy Feature Computation:** In this task, we use a window approach to calculate the energy of each pixel. Since the textures is not defined by a single pixel but defined by the whole picture and neighbours of every pixel. So, we need to take a window into consideration to decide a signle pixel's class. In this problem, I used a uniform mean method to get the features of the input image, every pixel's features are obtained by averaging the N by N window centered by itself.

**Improving Technique:** For this sub-problem, I used PCA to extract the most representative k-D features and apply the K-Means to it to figure it out whether this technique has better performance than the 15D features. I reduced the feature dimensions into 3D. Another technique is to apply the

median filter to eliminate the hole in the backgrounds. This post-process can produce a more smooth and connected result.

**Procedure:** The diagram of the whole procedure of texture segmentation is shown as below in Figure 7.

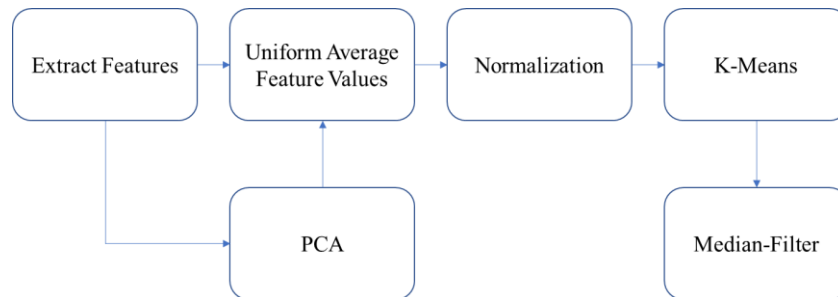
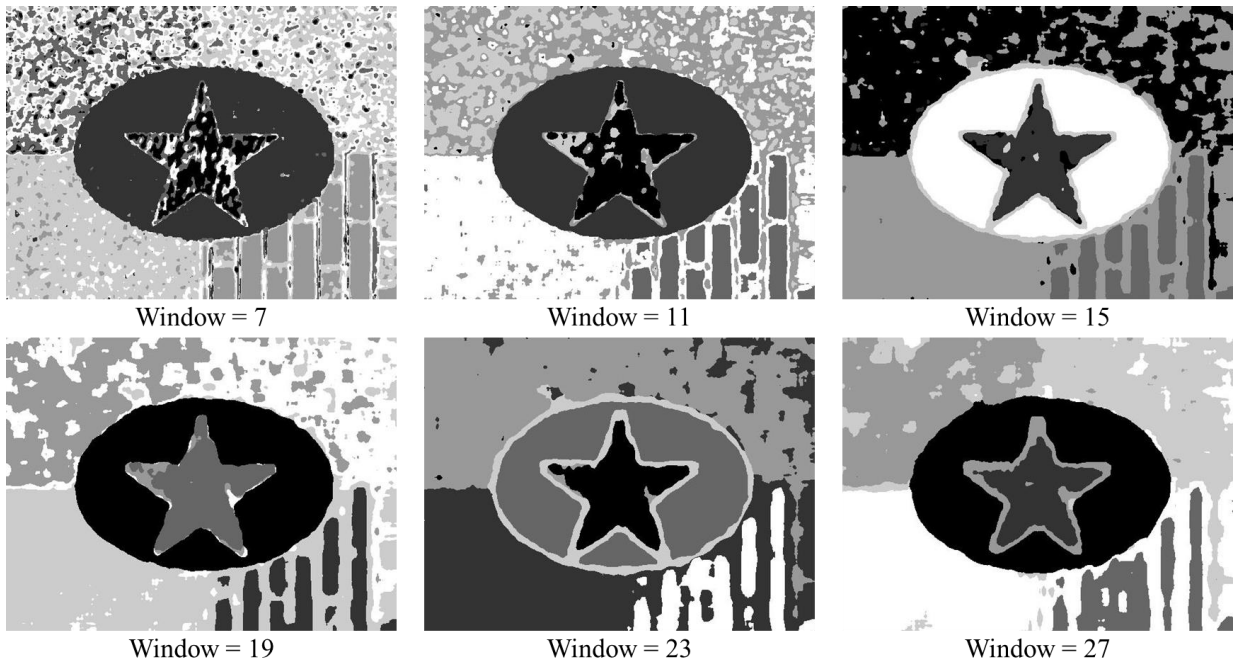


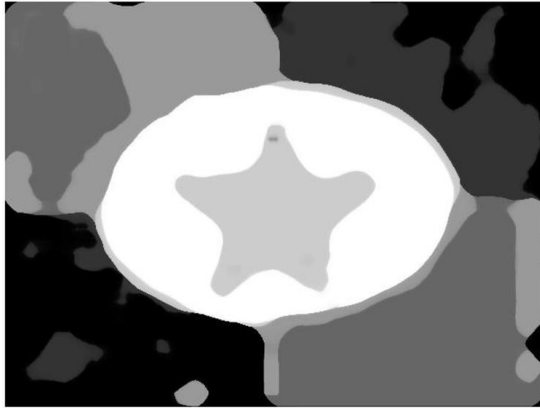
Figure 7. Diagram of Texture Segmentation

### 1.2.3 Experimental Result.

The result images that produced by the 3D image features and median filter with different window size is shown as below in Figure 8. And the result image that produced by the 14D image features in Figure 9.



Result Image by 3D Features



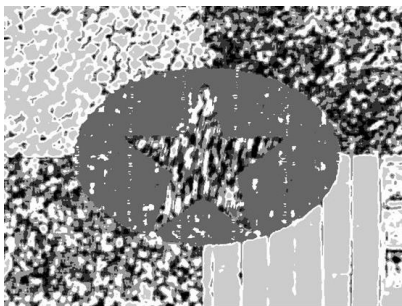
Window=31  
Filter Window = 20



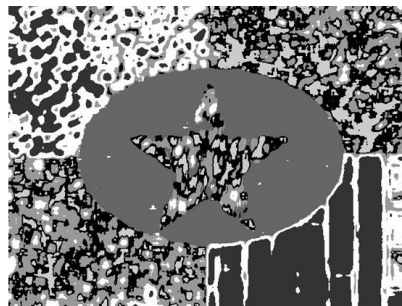
Window=35  
Filter Window = 15

### Result Images after Median Filter

Figure 8. Result Images of Improve Techniques



Window = 7



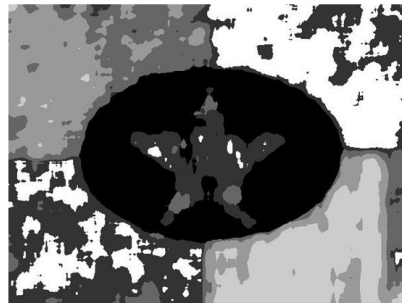
Window = 11



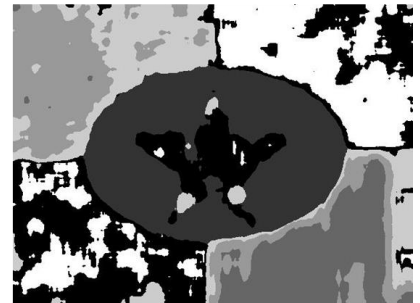
Window = 15



Window = 19



Window = 23



Window = 27



Window = 31



Window = 35



Window = 41

Figure 9. Result Image by 14D Features

### 1.2.4 Discussion

**Performace:** From Figure 8 and Figure 9, we can see that no result segments matchs the right boundaries and classes. It might mean that the Laws filter is not a perfect enough tool to classify the textures. Although the sketch of the textures is apparent to see. The results can tell a rough distibution of the textures but cannot precisely classify the pixels into correct classes. The reason for that might be the Laws filter cannot tell the whole properties of one pixel.

**Effects of Window Size:** From Figure 8 and Figure 9, we can see that the segmentation has many little circles in each area when the window size is small. Since the algorithm only averages a small window values to current pixel features. The differences will accumulate along with the distances of two pixels. While the window size grows, more pixels are averaged, which means larger area was regarded as one patch that can give more information about the texture. But along with the window grows, the speed of the algorithm is slower since it needs more convolution computation. Another thing is that the edges of each texture getting worse while the window is too large. This might because of the uniform filter can just erase some high frequency proerties. A larger window will filter more high frequencies in the image as discussed in image denoising.

In conclusion, the window size cannot be too small in order to prevernt too many circles. Also, it cannot be too big in order to prevent the feature loss

and the running time increasing.

**Improvement of PCA:** Generally speaking, the 3D features give slightly better results than 14D segmentation. We can tell that Figure 8 has less small circles than Figure 9. Also, we can see from the star that the inside get a little more clear. And furthermore, the shape of the star and the ellipse remains more like original than Figure 9. But it has other problems also, for example, we can see that the boundary and the edge of the star was enlarged in Figure 8. But the PCA can significantly speed up the algorithm, instead of computing K-Means on 15 features in a large scale (height \* width), this performance is a fair trade for efficiency. Besides that, PCA results seem to be more easily effected by the initialization of the K-Means, we can see from Figure 8 that it has two images that merge two textures into same class. This happens because the initialization might be located in an boundary region, which will separate the edges as a new texture, like Figure 8 (Window = 23).

Other methods could be applied to improve the performances, like a median filter can be applied to reduce the small holes in the segmentation. And the performance might get better if we increase the values of the K, which will help to identify the boundaries and separate the textures more clearly.

**Improvement of Median Filter:** From the figures above we can see that the median filter can significantly improve the performance of the algorithm. The window size of the filter should be bigger if the mean filter's window size is smaller. From Figure 8, we can see the median filter removes many small circles in each texture area. And the texture becomes smoother.

But of course, the four corners begins to have black pixels due to the padding method. And the edges are getting less sharp. In general, we can say that the median filter dose improve the performance of the segmentation.

## 2. Problem 2

### 2.1 Salient Point Descriptor

#### 2.1.1 Robustness to Geometrical Modifications

The SIFT is robust to the scaling, translation and rotation modification in spatial domain.

#### 2.1.2 How to achieve the Robustness

For scaling, SIFT uses Gaussian Filter to filter the image in different scale and construct the pyramid of the images. The feature points are the local maximum in all octaves of the pyramid is kept. So, the SIFT is robust through doing this filtering in many different scales

For translation, the keypoints are the local maximum or minimum of the DoG function. And they are described by a descriptor with the gradients of local direction. While a translation modification does not change this descriptor's information. And also, the keypoints should still be the local extremas of the DoG.

For rotation, same with the translation. The keypoints' properties are defined by the descriptor, which is generated by the local gradients of the pixel, and the local information will not change by rotations. And the keypoints should be the extremas of DoG, which will not be changed by rotation operation.

#### 2.1.3 Robustness to Illuminance Changes

For every pixel, the gradients can change a lot if the illuminance changes a lot. Which might create a large magnitude of gradient values. In SIFT, the



magnitude of the gradient in each direction is bounded by 0.1 times of the maximum possible gradient value. Also, the descriptor is constructed by the gradient values of 128 directions, which can represent the illuminance change in some level.

### 2.1.4 DoG over LoG

The DoG function is generated by doing the subtraction of the two images. While the LoG needs to calculate the second derivative of the image, which is a much harder work to do. For computation, the DoG' speed is much faster than LoG. And from Figure 10, we can see that the graph and properties are very similar between this two function. In this case, to accelerate the SIFT algorithm, it is wise to use DoG rather than LoG.

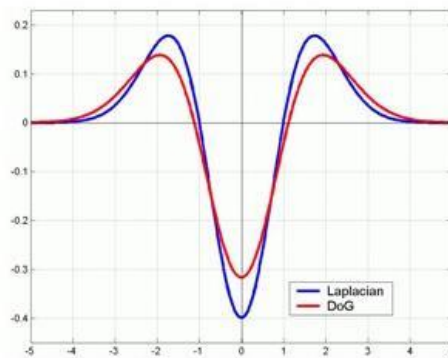


Figure 10. LoG and DoG Function Plot

### 2.1.5 Output Vector Size

The output vector should contains the pixels that in the circle with radius of 8 and centered by every keypoints. SIFT uses 8 divided orientations, which means each grid shoul possess 8 values of gradients. And in the orginal paper, it uses two grid types, 4 by 4 and 2 by 2. So the total size of the output vector is  $4*4*8+2*2*8 = 128+32 = 160$ .

## 2.2 Image Matching

### 2.2.1 Abstract and Motivation

Image matching is a very useful technique that used in many advanced image processing methods like object recognition and image segmentation. Transforms like SIFT can extract feature keypoints in the images and represent this keypoints and features with a lower dimensional vectore. The keypoints contain the local information that can make the features has the most discriminant power. These key point represent a local region information, and they are robust to many geometric transforms. This property make the method used widely both in industry and in research field.

In this part, I used OpenCV to implement several algorithms to extract the image keypoint and descriptors using SIFT, find the largest scale keypoints and match the image keypoints. After the experiment, the performance of the SIFT algorithm is going to be discussed and the explanation of the method will be given. Also, the calculation of the distances and location of the keypoints will be presented. The test images are shown as below in Figure 11.



Husky\_1.raw



Husky\_2.raw



Husky\_3.raw



Puppy\_1.raw

Figure 11. Test Images of Image Matching

### 2.2.2 Approach and Procedures

**OpenCV:** The OpenCV library contains of the SIFTdetector in the non-free headfiles. For the usage of the SIFT detector, we need to specify the parameters of the initialization of the detector. Some of the most important parameters is consisted by keypoints number, octave number and contrast threshold.

**Parameters:** For image feature extractor, there can be different number of keypoints for different images like Husky\_3 and Husky\_1. Also, the total number of the keypoints are decided by the octave number and the contrast threshold. Furthermore, in this task we want to focus our interests on the husky faces instead of the environment objects like grass, trees or snow ground. So, the keypoints number of the images should be fixed to filter out the background keypoints. Also, the contrast threshold and edge threshold should be selected precisely to prevent many keypoints are located in the enviroment objects. For this task. the parameters are (100, 3, 0.05, 30, 1.6).

**The Largest Scale Keypoint:** Every keypoint of the image has a descriptor to represent it. In OpenCV, the descriptor is a 128-dimension vector, whose element describes the gradient magnitude of the keypoint in certain direction. Also, every keypoint has a size, which shows the size of the area covered by this descriptor in the original image since the keypoints could be got in other scales. There are two ways to calculate the scales of the keypoint, one is using the keypoint's size or radius, the other is using the descriptor's values to calculate the magnitude of the 128-dimension vector's magnitude. In this problem, I used the later one, which uses the magnitude of the vector to represent its scale.

**Nearest Neighbour:** For the largest scale keypoint of Husky\_3, we need to find the nearest neighbour keypoint in the Husky\_1 image. There are two different ways to calculate it, too. Cause the direction of every gradient is very important, a method is to calculate the differences of the directions between the direction. Like we can compute the cosine value of two vectors to see if they are similar in the orientations. We can compute the dot product of the two vectors and divided the value by the multiplication of their magnitudes to get the cosine value between the two vector. The smaller value is, the closer the vectors are. Another method is to compute the Euclidean distances of two vectors. This will give the spatial distances of two data points in the high dimensional space. Both ways can be used to find the nearest keypoint in Husky\_1, in this problem, I used the second method (Euclidean distance) to find the nearest keypoint.

**Orientation of Keypoints:** The orientation of each keypoint is decided by the gradient direction histograms. The magnitudes of the gradients is computed as follow in equation (6). And the gradient orientation can be computed by the equation (7). For each keypoint, SIFT computes its gradient orientation with neighbours. Then SIFT computes the histogram of the orientations of the gradients. The x-axis of the histogram has 36 values which divide the 360 degree into 36 intervals with 10 degrees. The y-axis is the sum of the gradients in the same interval. Let the interval has the largest scale be the orientation of the keypoint. If there is another interval that reaches 80% the biggest scale can be marked as an assistant orientation.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (6)$$

$$\theta(x, y) = \tan^{-1} \left[ \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \right] \quad (7)$$

**Procedure:** The diagram of the whole procedure of image matching is shown as below in Figure 12.

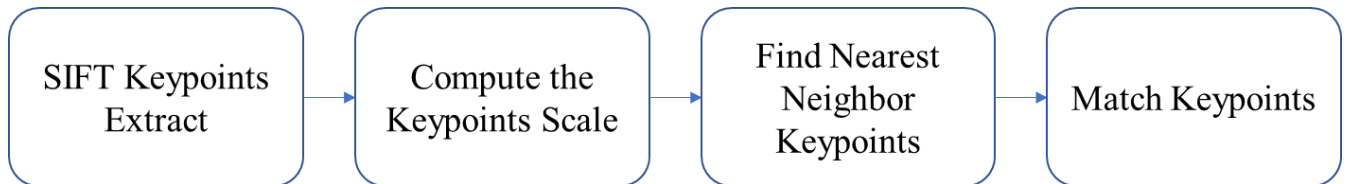


Figure 12. Image Matching Procedure

### 2.2.3 Experimental Result

The result of the largest keypoint of Husky\_3 and the nearest keypoint in Husky\_1 are shown in Figure 13. The circle's radius shows in the image shows the size of each keypoint.

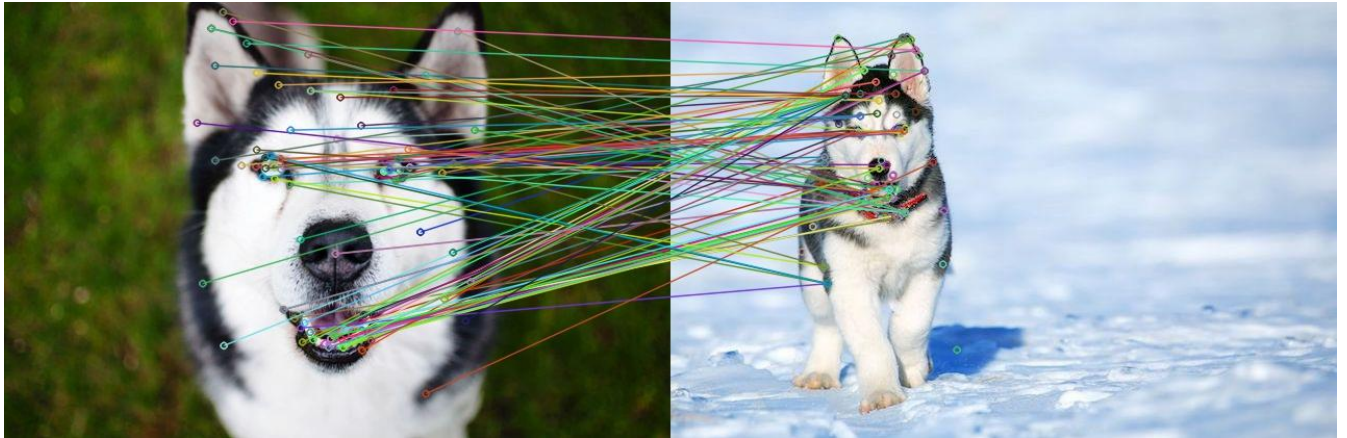




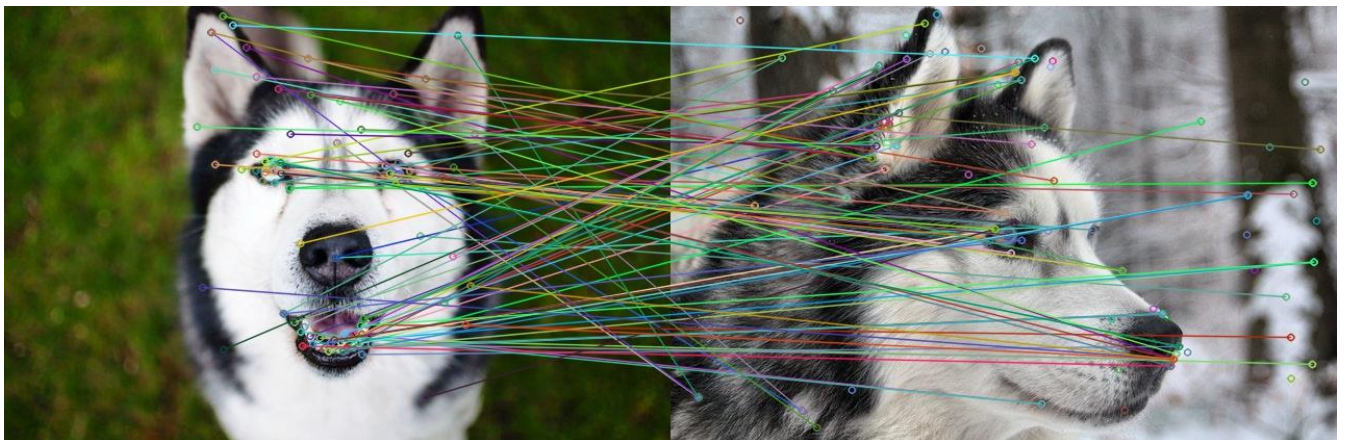
Figure 13. Largest Keypoint and Nearest Neighbor

The match results of image matching between Husky\_1 and Husky\_3,

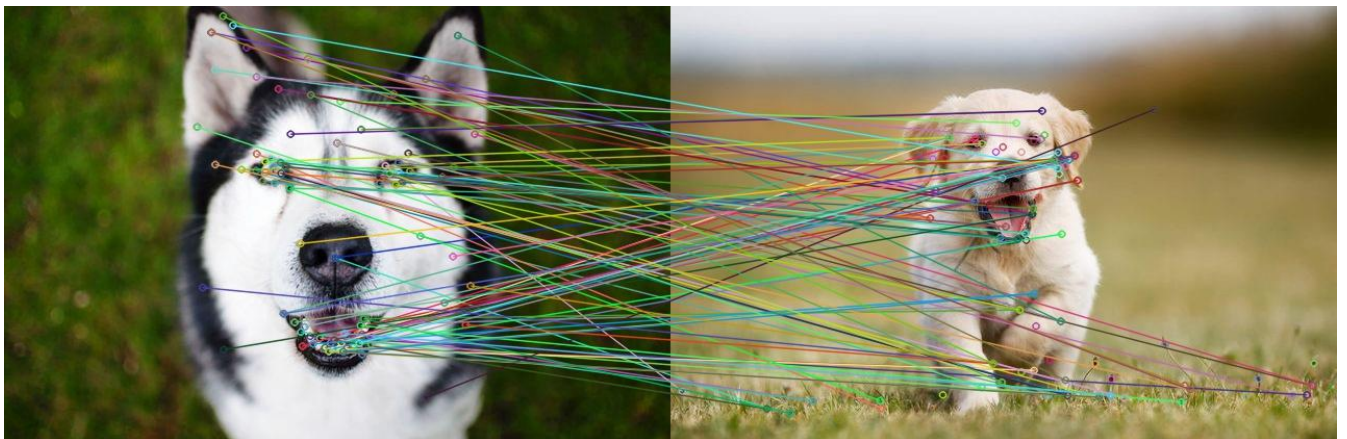
Husky\_3 and Husky\_2, Husky\_3 and Puppy\_1, Husky 1 and Puppy\_1 are shown as below in Figure 14.



Husky\_3 and Husky\_1 Matching



Husky\_3 and Husky\_2 Matching



Husky\_3 and Puppy\_1 Matching



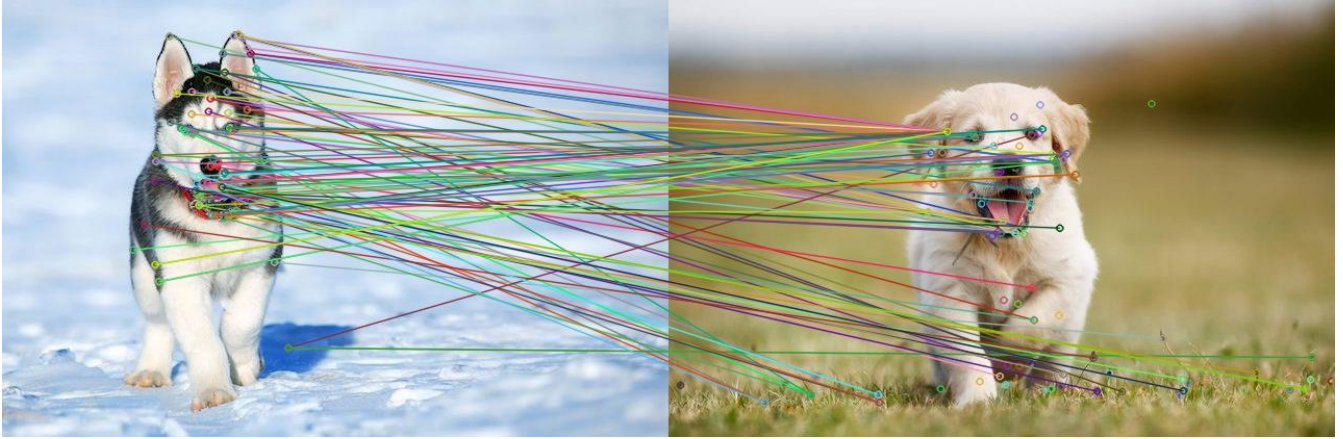


Figure 14. Image Matching Results

The property of the largest keypoint and matched keypoint is shown as Table 8.

Table 8. Property of Largest Keypoint and Matched Keypoint

Property	Largest Keypoint	Matched Keypoint
Size	69.394	11.171
Angle	110.243	58.5922
Response	0.0633106	0.0675486
Octave	13631748	15204865

### 2.2.4 Discussion

**Performances:** For the visual evaluation, we can see that the largest keypoint match is relatively acceptable since it maps the husky face to husky face even though the location are not exactly same. This error might be caused by the different perspective of the dog face and the image environmental backgrounds.

For image matching, we can clearly see that the match performance between Husky\_3 and Husky\_1 is the best. More keypoints are matched from dog face to dog face in that pair, while the other pairs have many keypoints



matched from the dog face to the environment objects. The reason for that is the perspectives of Husky\_3 and Husky\_1 are more similar compared with Husky\_2, and the face features of Huskys are more similar compared to Puppy image. More specifically, in Husky\_2 and Husky\_3 match, cause Husky\_2 does not have symmetric face features and the features of teeth and mouths. Many features of Husky\_3 matches to the environmental objects like trees. In Husky\_3 and Puppy\_1 match, cause Puppy\_1 does not have standing ears features. We can see that many keypoints of ears in Husky\_3 matched mistakenly in Puppy\_1. While in Husky\_1 and Husky\_3 pair, the matches are relatively more reasonable, only few keypoints are matched to the backgrounds. Also, since the details of teeth in Husky\_1 is small and ambiguous. So, some keypoints of Husky\_3 on the teeth area matches mistakenly to the ears of Husky\_1.

**Property of Keypoints:** The size of these two keypoints have different scale. This is expected since the husky face in Husky\_3 and Husky\_1 image has different scale factor. Also, we can see that the responses of these two keypoints are similar. For the angle of the keypoints, we can see a big gap between them, this might because of the not exactly match in the image.

**Orientation of Keypoints:** What needs to be specified is that the angle of the keypoints does not represent the gradient orientation. The orientation of the gradients of each keypoints can be computed by the 128-dimensional descriptor. While we can use the cosine distances to determine whether these two keypoints are similar in orientation.

## 2.3 Bag of Words

### 2.3.1 Abstract and Motivation

Bag of words model can regard the image features as visual words to form a codebook. The image can be represented by a histogram of the words in the codebook. This model can be used in the image comparison and classification.

In this part, I implemented two algorithms to compare the differences between Husky\_3 and other images. These two methods' results can verify each other. After the experiments, the performances will be discussed and a conclusion will be given.

### 2.3.2 Approach and Procedures

**Feature Extraction:** The features of each image can be extracted by using the SIFT. For the fairness of each image, the number of the keypoints should be the same. In this problem, I used 100 keypoints in each image to form the feature dataset. So, for the first method, the dataset is a 400 by 128 2-D array. For the second method, the dataset is 4 2-D arrays with height is 100 and the width is 128.

**Codebook:** The codebook contains all the codewords of the image. The codeword is a representation of some similar features vectors. The general procedure to generate the codeword is to use the K-Means cluster algorithm for all feature vectors and use the centers of cluster to represent the codewords. Also, all the codewords can form a codebook, which can be regarded as a dictionary. Every feature vector of each image can be assigned

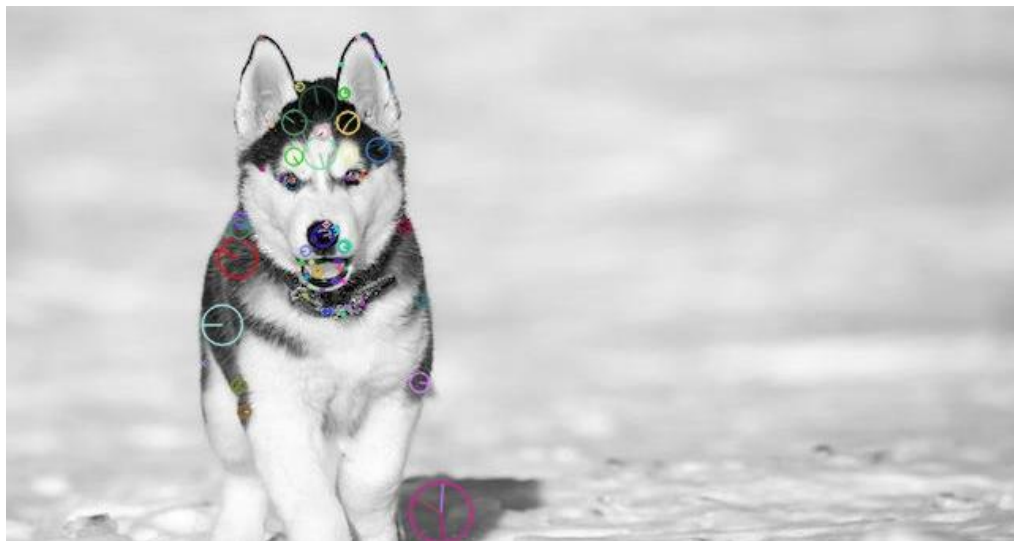
to one code word, which can construct the histogram of each image.

**Histogram Plot:** I used the Euclidean distance to assign the feature vectors to the certain code words and plot the frequency of each codeword. For the first method, the x-axis of all histograms is same, the 8 centroids of all the feature vectors. For the second method, the x-axis of all histograms are different, but they are sorted based on the similarity between the centroids of Husky\_3 image.

**Methods:** I used two different methods of BoW. Firstly, the 8 centroids of 4 images are generated by the all image descriptors. And the histogram is plotted based on these 8 centroids. Secondly, each image descriptors generate 8 centroids, and the other three images' centroids are sorted based on the distances between it and the Husky\_3's centroids. This two methods' results can verify each other.

### 2.3.3 Experimental Result

The keypoints of different images are shown as below in Figure 15.



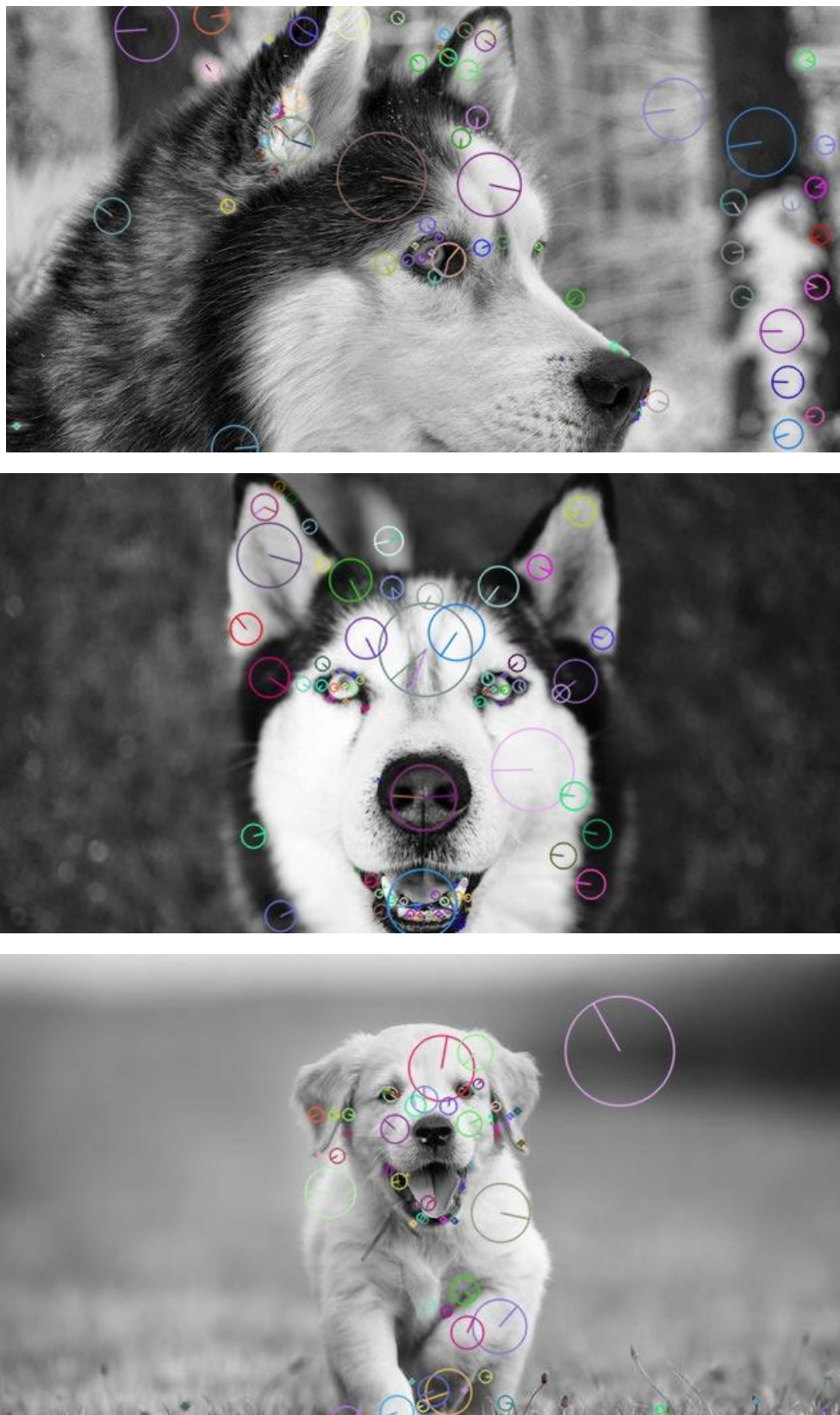
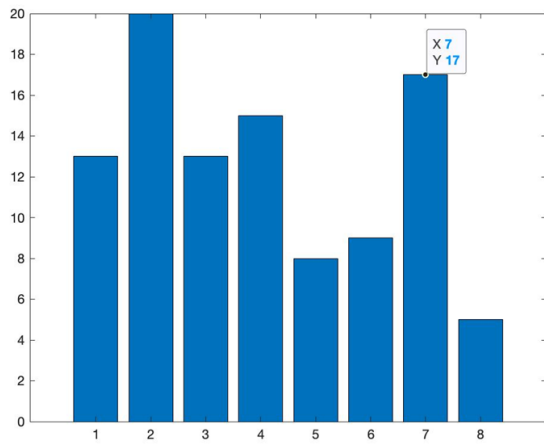


Figure 15. Keypoints of 4 Images (Num = 100)

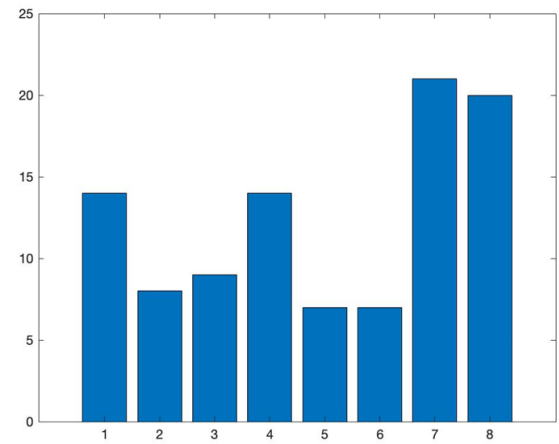
The intersection of Histograms of different Images are shown as Table 9. By using the different centroids.

Table 9. Histogram Intersection Rate

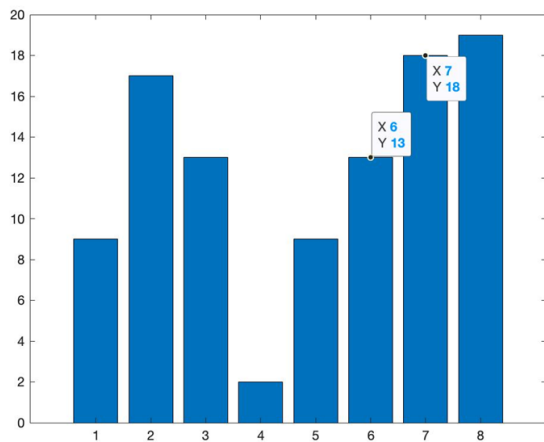
Intersection	Husky_1	Husky_2	Husky_3	Puppy_1
Figure 16	80%	79%	100%	64%
Figure 17	79%	76%	100%	68%
Figure 18	80%	76%	100%	66%



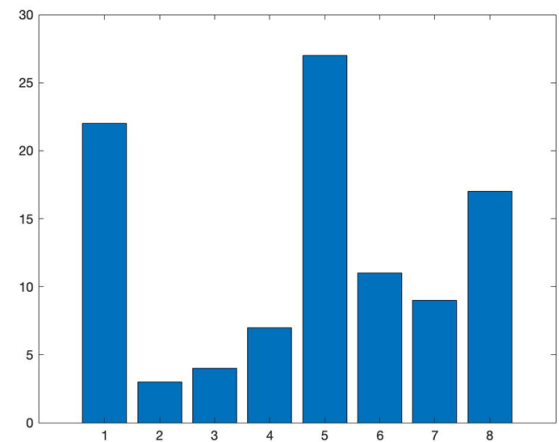
Husky\_1 Histogram



Husky\_2 Histogram

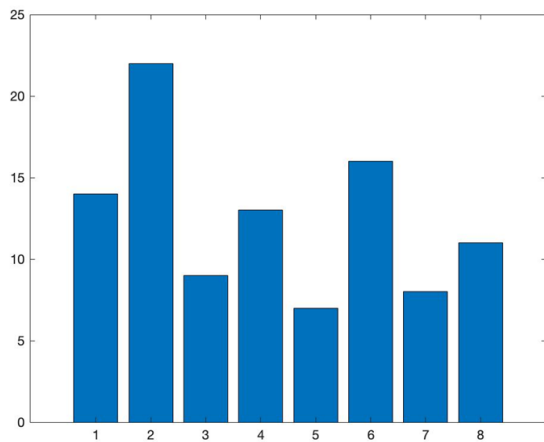


Husky\_3 Histogram

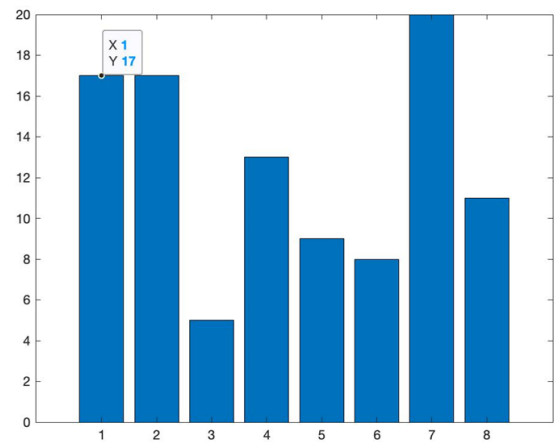


Puppy\_1 Histogram

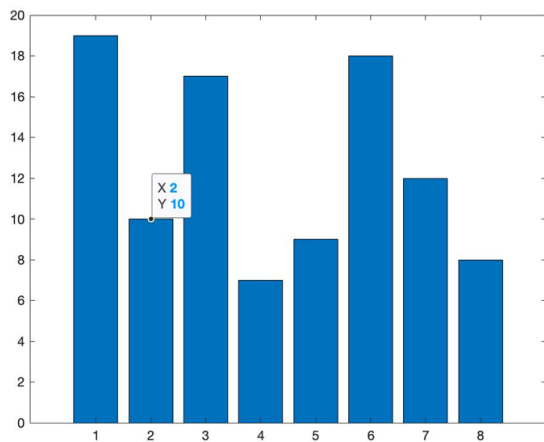
Figure 16. Trial 1 of Histogram Plot



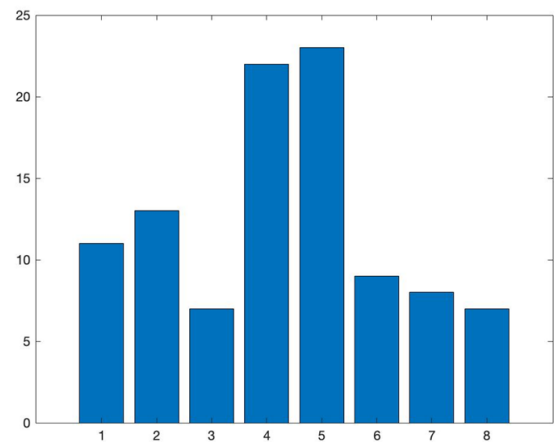
Husky\_1 Histogram



Husky\_2 Histogram

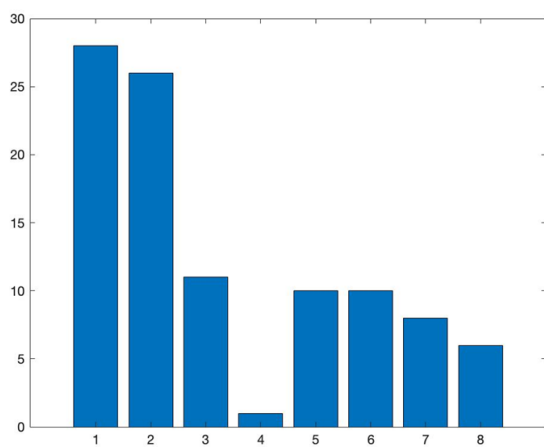


Husky\_3 Histogram

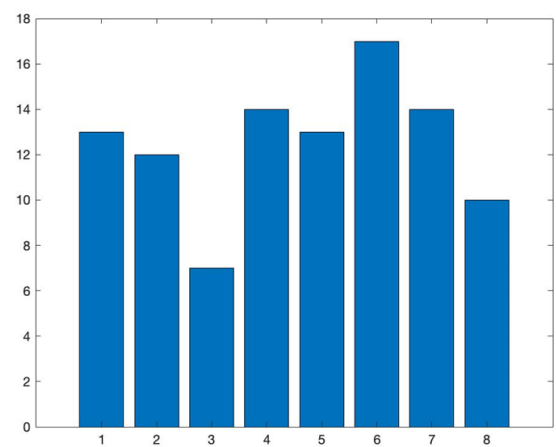


Puppy\_1 Histogram

Figure 17. Trial 2 of Histogram Plot



Husky\_1 Histogram



Husky\_2 Histogram

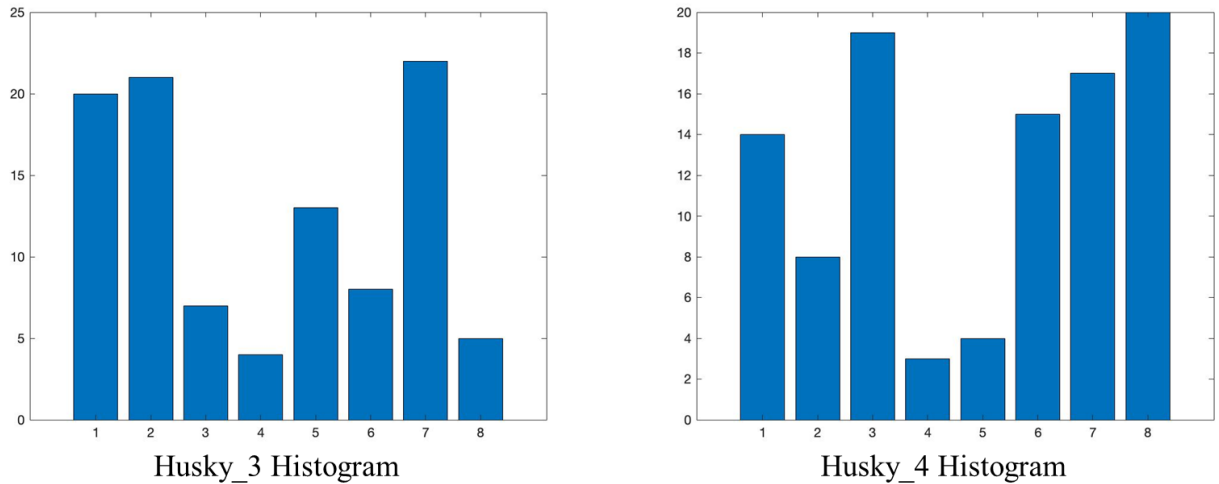
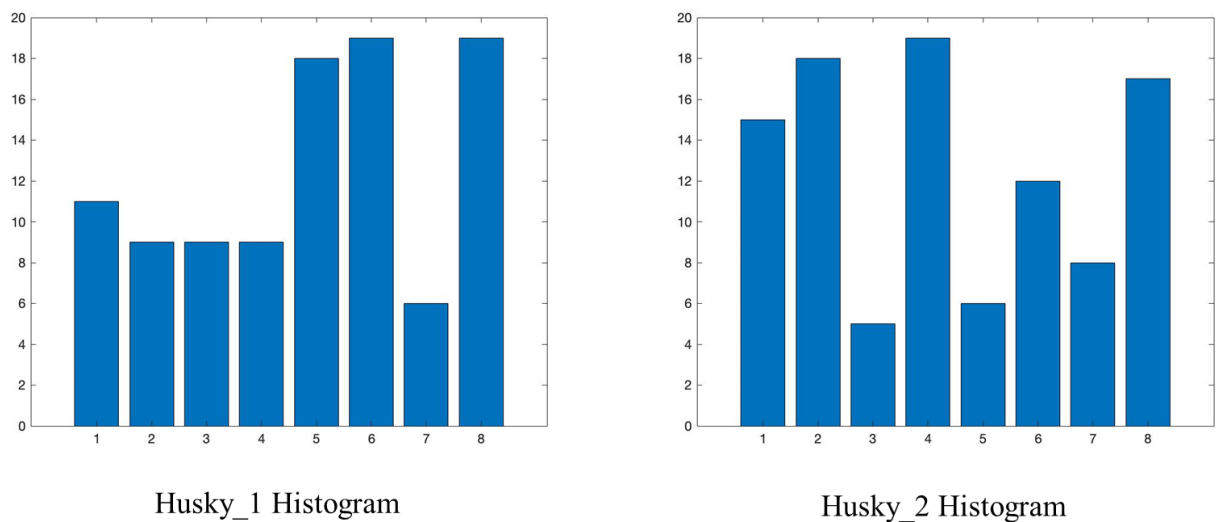


Figure 18. Trial 3 of Histogram Plot

The intersection of Histograms of different Images are shown as Table 10. By using the the same centroids.

Table 10. Histogram Intersection Rate Same Centroids

Intersection	Husky_1	Husky_2	Husky_3	Puppy_1
Figure 19	89%	82%	100%	68%
Figure 20	81%	87%	100%	73%
Figure 21	72%	84%	100%	63%



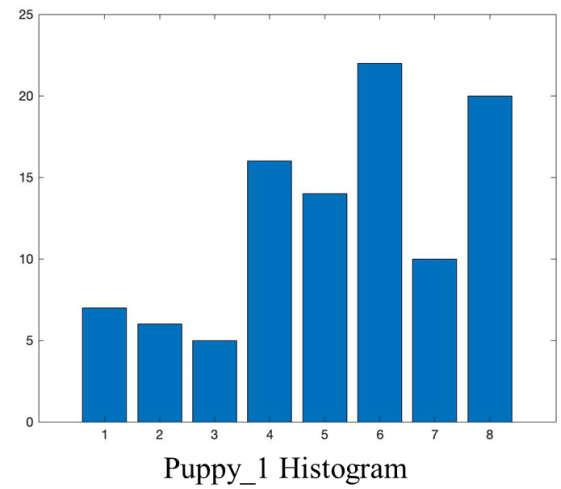
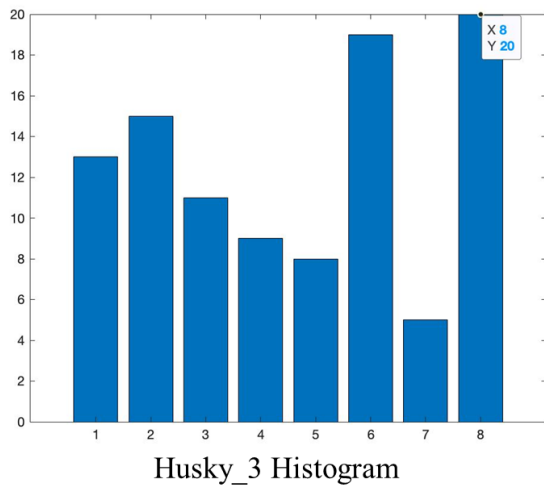


Figure 19. Trial 1 of Histogram Plot Same Centroids

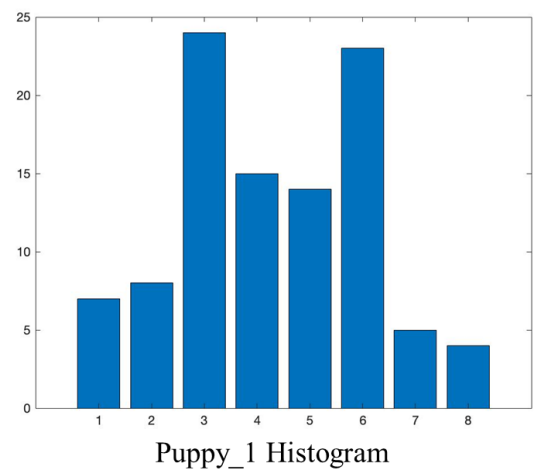
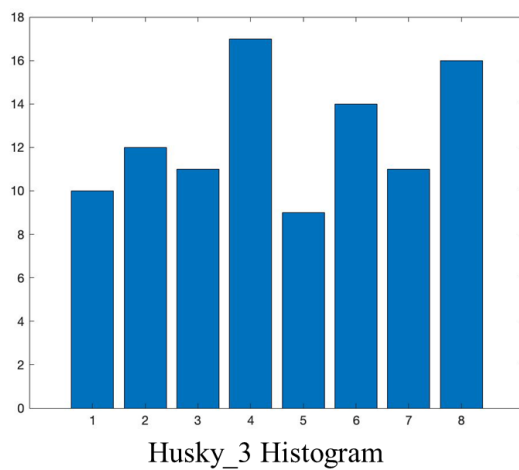
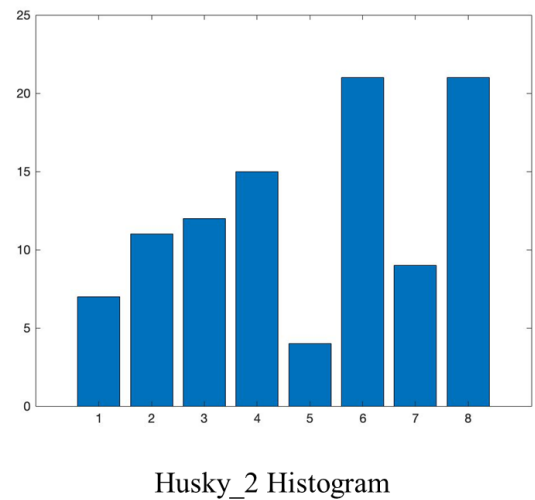
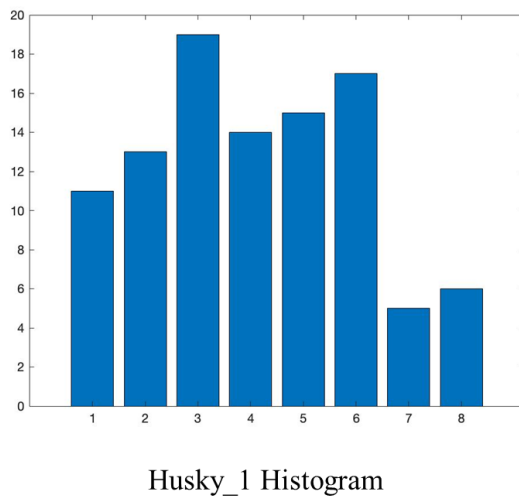


Figure 20. Trial 2 of Histogram Plot Same Centroids



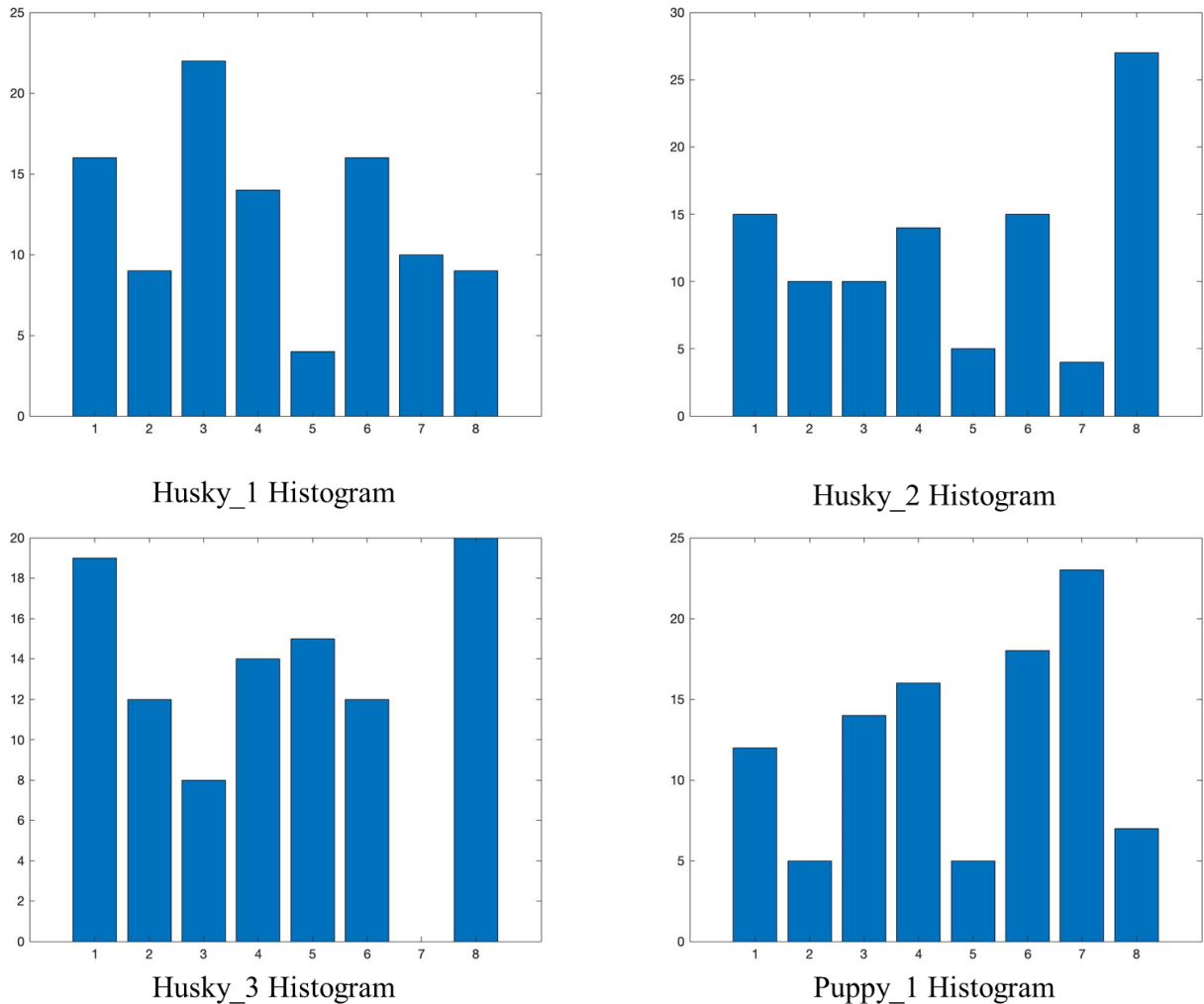


Figure 21. Trial 3 of Histogram Plot Same Centroids

### 2.3.4 Discussion

**Differences of Two Methods:** These two methods have some common features. They both use K-Means to cluster the feature vectors. Also, they use Euclidean distances to calculate the distances of different data points. While the method with 8 centroids of each image can possess more information since it does the K-Means on each image. Each centroid should possess some information about the image itself. On the other hand, the method with 8 same centroids should have less information about the individual images,

but have a more general information. Cause it dose the K-Means on the whole image set. It is a little bit harder to compare the histogram with Husky\_3 directly for the first method, the x-axis is different for each image, we need to sort the centroids in the order of the distance between each centroid and the centroids of Husky\_3. While for the second method, the histogram intersection can be computed directly.

**Results:** To evalutae the similarity more precisely, I ran the algorithm multiple times to see the intersections between Husky\_3 and other images. The results are shown as Table 9 and 10. The histograms are shown as Figure16-21. From Table 9 and Table 10, we can tell that the Husky\_3's histogram has more intersection with the histograms of Husky\_1 and Husky\_2, which indicates that the Husky\_3 is more similar to Husky\_1 and Husky\_2 image and less like to Puppy\_1 image. In Table 9, the Husky\_3 is more likely to Husky\_1, this can be explained intuitively. Since the dog face's perspective and outlibes are very similar, the only thing changes is the scale of the face. The perspective of Husky\_2 and Husky\_3 has a huge differnce, that might be why the Husky\_2's histogram is less likely than Husky\_1. While in Table 10, the lowest similarity still belongs to the Puppy\_1 image, but the Husky\_2's similarity is a slightly higher than Husky\_1. For this change, there might be two explanations. The first one is that do the K-Means on the whole image set may create a more unstable system since the data points have a much wider range than 1 image data points. The other is the same centroids method may be more sensitive to the

scale transformation than the rotation or affine transformation, which might still be generated by the large range of data.

From the histogram, we can see directly, the shape of the histogram of Husky\_1, Husky\_2 and Husky\_3 are more similar. While the histogram of Puppy\_1 differs from the others. More specifically, for Figure 17, the Husky\_1 and Husky\_3 both have low frequencies on centroid 4 and 5, while the Puppy\_1 image has extremely high frequencies of these two centroids.

More over, from Figure 20, we can even see that the histogram of Puppy\_1 is similar to the histogram of Husky\_1. The order from the largest frequency to lowest frequency of Puppy\_1 is [3, 6, 5, 4, 2, 1, 8, 7]. The same order for Husky\_3 is [3, 6, 4, 5, 2, 1, 7, 8]. They are relatively similar in some level. This observation might can help to prove the algorithm of method 2 is more sensitive to scaling transformation since the scale of the dog face is very similar in Husky\_1 and Puppy\_1.

**Conclusion:** The Husky\_3 images are more similar to Husky\_1 and Husky\_2 since the local features of husky dog is distinctive from the normal dog face. Two methods' results both prove this point, but the similarity to Husky\_1 and Husky\_3 is different. This might caused by the large range of data points and the second method might be more sensitive to the scaling transform.

## References

- [1] Lowe, David G. "Object recognition from local scale invariant features." iccv. Ieee, 1999.