EE569 Digital Image Processing

# HOMEWORK #4

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

Yao Fu
6786354176
yaof@usc.edu

EE 569 Homework #4
March 22, 2020

## Problem 1: Texture Analysis and Segmentation (60%)

(a) Texture Classification --- Feature Extraction (Basic: 15%)

(b) Advanced Texture Classification --- Classifier Explore (Basic: 15%)

(c) Texture Segmentation (Basic: 20%)

(d) Advanced Texture Segmentation (Advanced: 10%)

### 1.1 Abstract and Motivation

When digitally manipulating an image, it's nature for us to implement texture analysis and segmentation. An image texture is a set of metrics that are calculated during the course of image processing, making it possible to quantify the perceived texture of an image. Image texture, which is characterized by the spatial distribution of intensity levels in a neighborhood, is a feature used to partition images into regions of interest and to classify those regions. In other words, it can provide individuals with information related to the spatial arrangement of color or intensities in images or a selected region of images.

Texture analysis usually refers to the characterization of regions in a certain image by their texture content. What's more, figuring the image textures out is the only way that can help do image segmentation or classification.

In order to understand the basic mechanism of texture analysis and image segmentation, in this report, I am about to use MATLAB 2019b to implement algorithms and use some built-in functions to achieve them. Necessary resulting images will be presented later in my report.

### 1.2 Approach and Procedures

(a) Texture Classification --- Feature Extraction

In this part, I will follow the TA's guideline to extract features for all texture images, which include 9 "blanket", 9 "brick", 9 "grass", and 9 "rice" images. The approach to extracting features is stated as follows:

Step1: Subtract the image mean to reduce illumination effects.

Step2: Use the twenty-five 5 by 5 Laws Filters in Table 1 to extract feature vectors by doing convolution with those filters in the image (use the mirror reflection to do boundary extensions). The 2D filters are obtained by doing the tensor production with two filters as shown below:

$$\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} \times \begin{bmatrix}-1 & -2 & 0 & 2 & 1\end{bmatrix} = \begin{bmatrix}-1 & -2 & 0 & 2 & 1\\ -4 & -8 & 0 & 8 & 4\\ -6 & 12 & 0 & 12 & 6\\ -4 & -8 & 0 & 8 & 4\\ -1 & -2 & 0 & 2 & 1\end{bmatrix}$$

Table 1: 1D Kernel for 5 by 5 Laws Filters

| 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] |

Step3: Use the absolute values to average energy to form a 25-D feature vector. Then replacing each pair with average to define a 15-D feature vector as follows:

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

The feature set should be "n samples × 15 feature dimensions".

Moreover, I will evaluate the discriminant power of those features by using the formula given by the TA:

Let $y_{ij}$ be the $jth$ observation in the $ith$ class. The overall average is

$$\bar{y}_{..} = \frac{\sum_i \sum_j y_{ij}}{\sum_i \sum_j 1}.$$

The average within class $i$ is

$$\bar{y}_{i.} = \frac{\sum_j y_{ij}}{\sum_i 1},$$

where the number of values of $j$ may depend on $i$.

The intra-class sum of squares is

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

The inter-class sum of squares is

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

where $n_i$ is the number of the observations in the $ith$ class.

The discriminant power is

$$\frac{\text{The interclass sum of squares}}{\text{The intraclass sum of squares}} = \frac{\sum_i (n_i (\bar{y}_{i.} - \bar{y}_{..})^2)}{\sum_i \sum_j (y_{ij} - \bar{y}_{i.})^2}.$$

I will discuss the results of the discriminant power of those features later.

Step4: Reduce the feature dimension from 15 to 3 using the principle component analysis (PCA). I will write my own PCA function according to the TA's hint:

1. Consider the feature matrix $X_{m \times n}$, where $m$ is the number of samples and $n$ is the number of features.
2. Compute mean $(m_X)_j = \sum_{i=1}^{m} \frac{X_{i,j}}{m}, j = 1 \dots n$
3. Subtract mean to get zero mean data matrix $Y_{m \times n}$.
4. Compute SVD of Y: $Y = U \cdot S \cdot V^t, where\ U_{m \times m}, S_{m \times n}, V_{n \times n}$

5. Sort singular values in descending order of magnitude.
6. Retain first k singular values when considering reducing dimension from n to k

$$V_R = V[:, :k]$$

7. Form dimension reduced matrix $Y_R = X \times V_R$

(b) Advanced Texture Classification --- Classifier Explore

In this part, I will follow the requirements of homework to do both unsupervised and supervised learning.

Step1: **Unsupervised**: I will write my own K-means algorithm for test image clustering based on the 15-D and 3-D feature vectors. Later I will discuss the effectiveness and results. The procedure is shown as follows:

1. Initialize the cluster centroids by randomly selecting K samples from the data.
2. Calculate the distance between samples and centroids to check their similarity

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

3. Compare the distance of samples to $K$ centroids to see which centroid has the minimum distance to it, then label the sample to that cluster

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

4. Obtain the new cluster centers

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

5. Repeat step 2-5 until change in cluster centroids are effectively negligible.

Step2: **Supervised**: I will use the 3-D feature of training images to train Random Forest (RF) form [1] and Support Vector Machine (SVM) from [2] respectively with the help of built-in functions. The basic procedure is shown as follows:

1. Label train images as shown in the Table 2:

Table 2: The train images' labels

| Classes | Blanket | Brick | Grass | Rice |
|---|---|---|---|---|
| Labels | 1 | 2 | 3 | 4 |

2. Train the two classifiers with the help of train images and labels.
3. Predict labels for test images and calculate error rate. Later I will discuss my results.

(c) Texture Segmentation

In this part, I will segment the textures of the image "comp" by applying the 25 5x5 Laws Filters. The approach to doing the texture segmentation is stated as follows:

Step1: Subtract the image mean to reduce illumination effects.

Step2: Use the twenty-five 5 by 5 Laws Filters in Table 1 to extract feature vectors by doing convolution with those filters in the image (use the mirror reflection to do boundary extensions).

Step3: Use the absolute values to average energy to form a 25-D feature vector by choosing the

window size as 51. Then replacing each pair with average to define a 15-D feature vector as follows:

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

Then use $L5L5$ to do feature normalization. The feature set should be "n samples × 14 feature dimensions".

Step4: Use the K-means algorithm to perform segmentation on the "comp" image based on the 14-D energy feature vectors. Theoretically speaking, if there are K textures in the image, the output image will be of K gray levels, with each level represents one type of texture.

(d) Advanced Texture Segmentation

Unfortunately, I don't not get good segmentation results for the complicated texture mosaic image form the part (c). Then I will develop a post-processing algorithm to try to make holes merge and every single segmented region have its own grayscale, making it possible to let the final resulting image only have six regions.

Later I will discuss my results.

### 1.3 Experimental Results

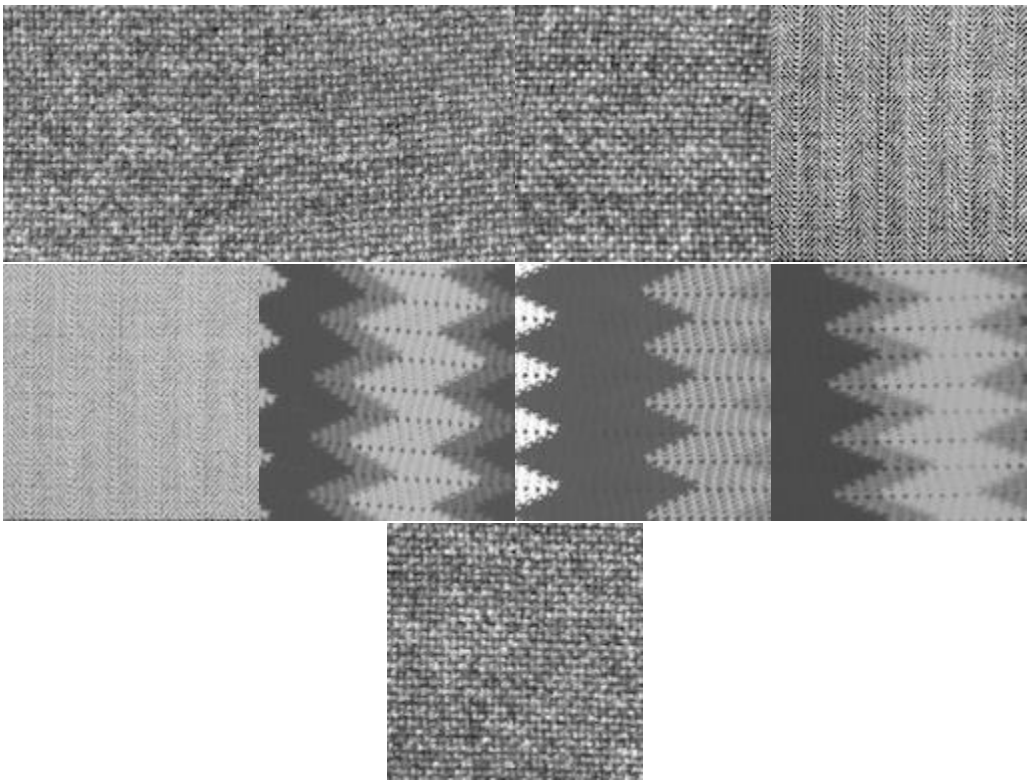

Fig. 1.1: The 9 training "blanket" images

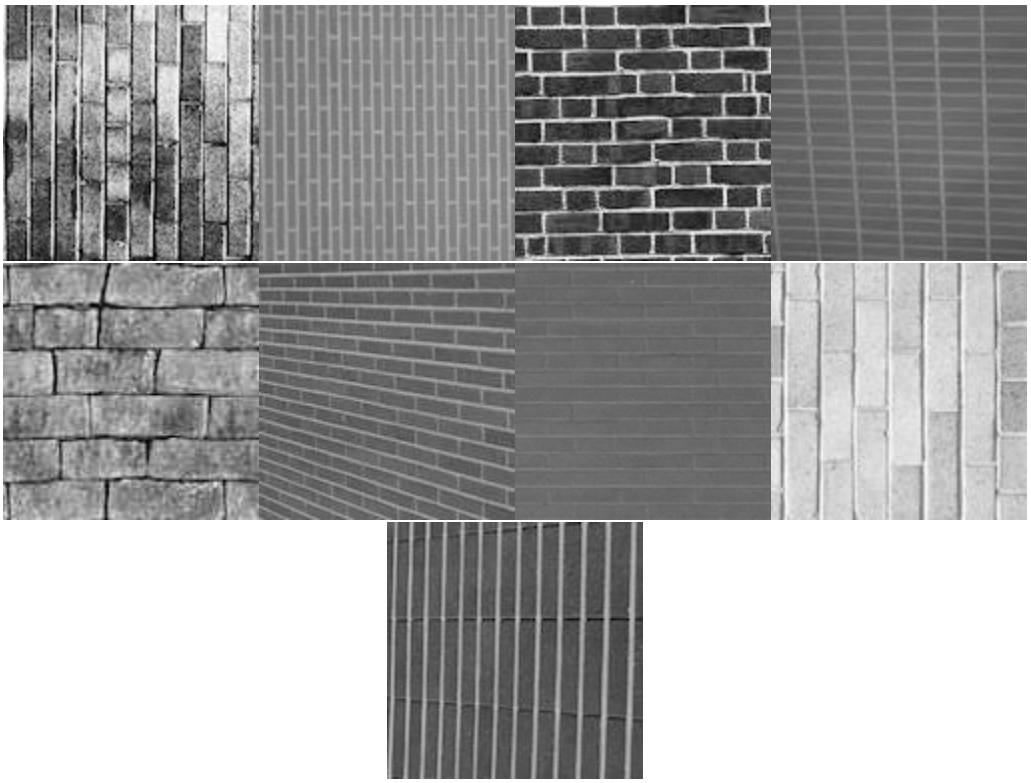

Fig. 1.2: The 9 training “brick” images

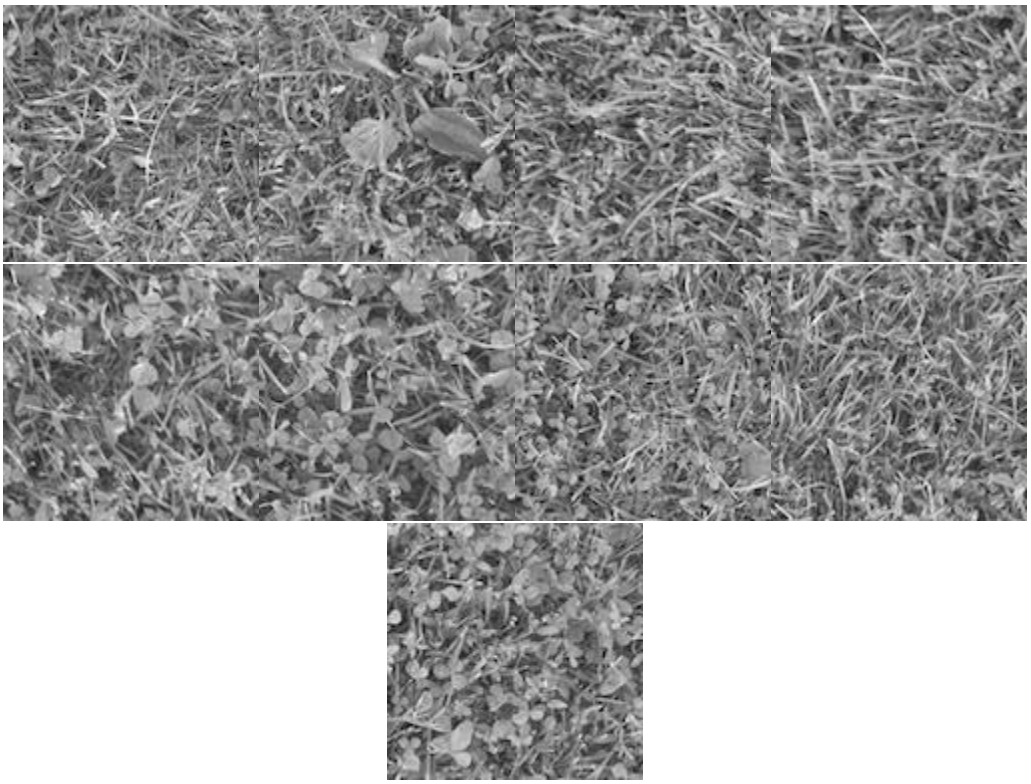

Fig. 1.3: The 9 training “grass” images

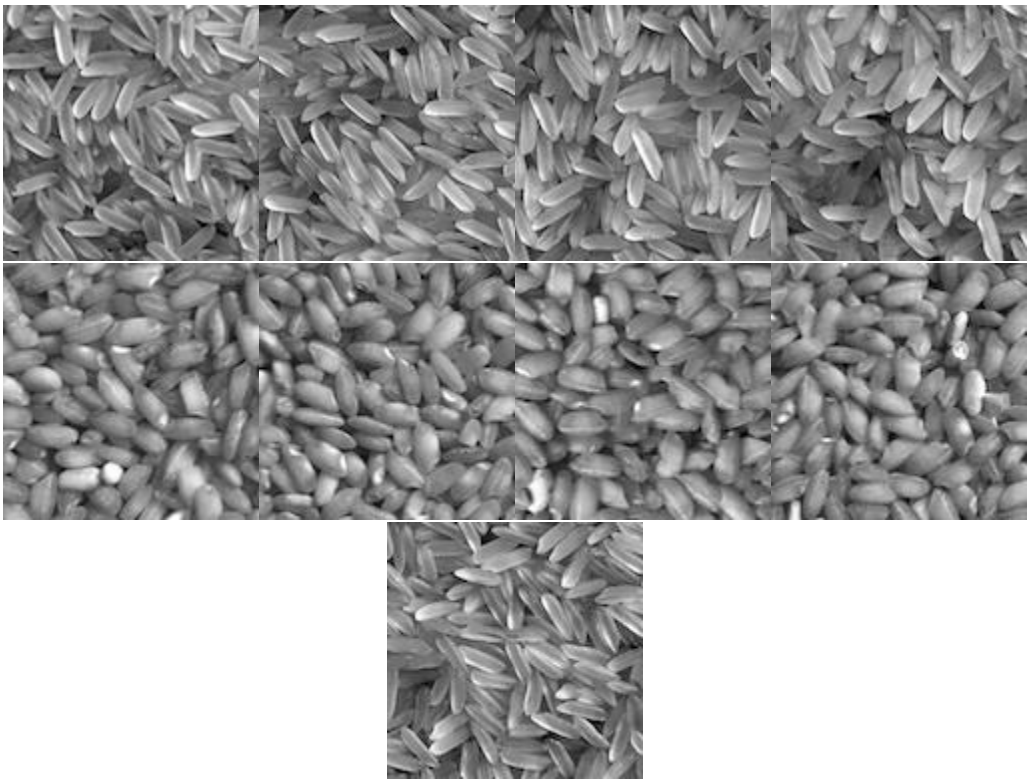

Fig. 1.4: The 9 training “rice” images

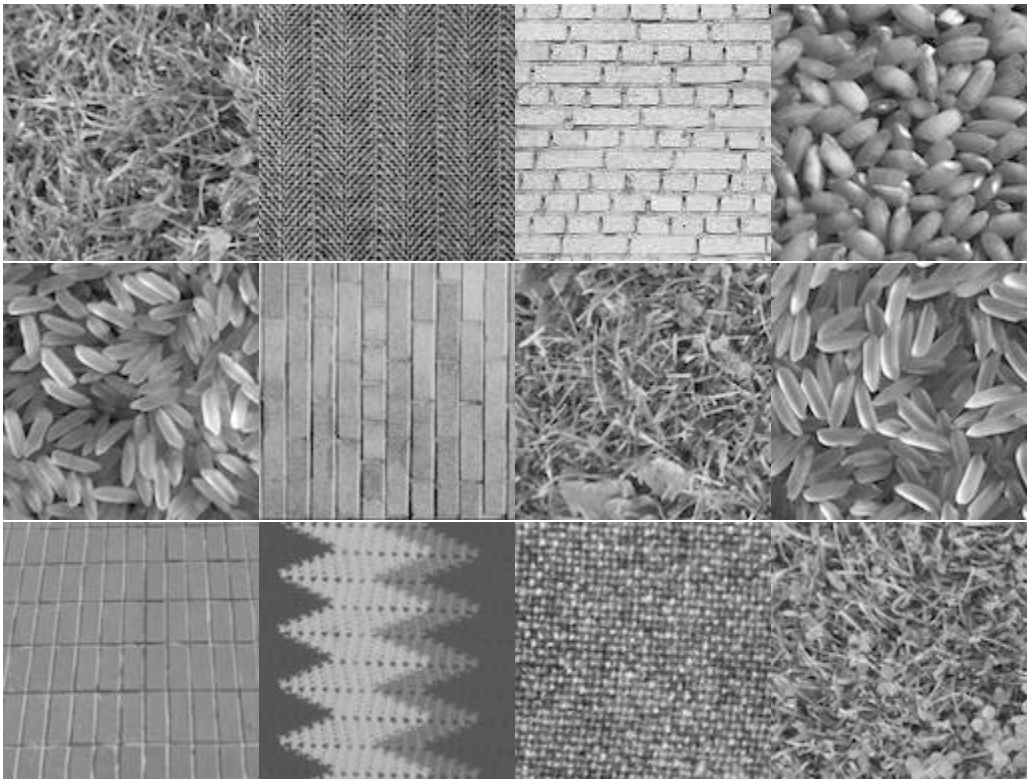

Fig. 1.5: The 12 testing images

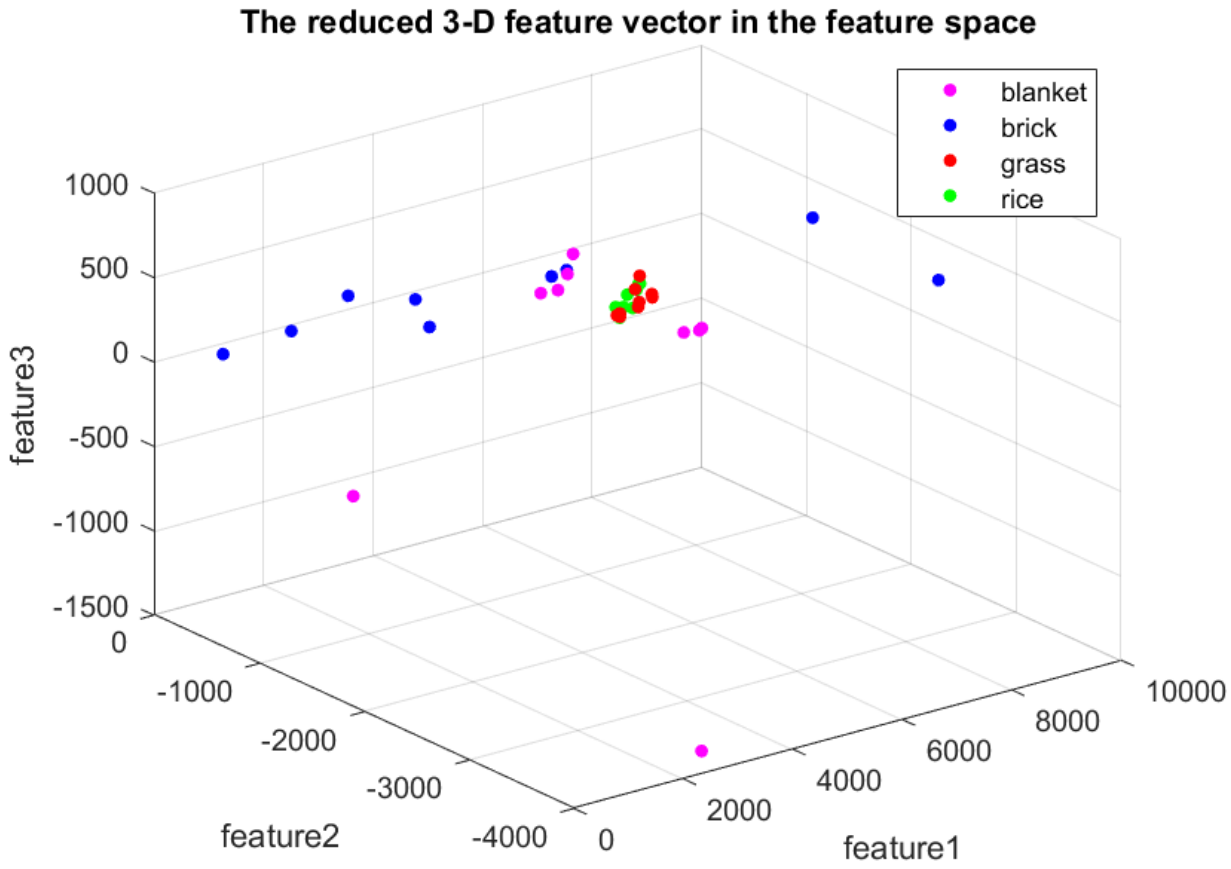


Fig. 1.6: The reduced 3-D feature vector in the feature space

Fig. 1.7: The texture segmentation result

Fig. 1.8: The texture segmentation result after post-processing

### 1.4 Discussion

As I have mentioned in the previous report, how to evaluate images' visual quality is always a huge challenge to people, even to image scholars themselves, because the process of that evaluation is so subjective that it's almost impossible for individuals to reach a consensus about whether certain images' visual quality is good enough. It's not unapparent that in order to evaluate an image's quality objectively, we have to ask a number of people's opinions.

However, in this report, I will merely use my own judgement to evaluate the visual quality of resulting images. My discussion of experiment images of Problem 1 is stated as below.

In part (a), I use MATLAB 2019b to implement the feature extraction. I have already discussed about the procedure of feature extraction, feature averaging, and feature reduction. After I get the 15-D feature vectors, I utilize the way, which is clearly presented in the step3, to figure out which feature dimension has the strongest discriminant power, and which has the weakest one. The results of discriminant power are shown in the Table 3, from which it's not hard for us to see that the L5S5/S5L5 dimension has the weakest discriminant power and the E5E5 has the strongest one, indicating that features from E5E5 are very strong. Moreover, in this part, after using the PCA method to reduce feature dimension, I get the figure of the reduced 3-D feature vector in the feature space as shown in Fig. 1.5.

Table 3: The values of discriminant power for each feature dimension

| **Filters** | L5L5 | L5E5/E5L5 | L5S5/S5L5 | L5W5/W5L5 | L5R5/R5L5 |
|---|---|---|---|---|---|
| **Values** | 0.2351 | 0.4124 | 0.1714 | 0.2360 | 0.2554 |
| E5E5 | E5S5/S5E5 | E5W5/W5E5 | E5R5/R5E5 | S5S5 | S5W5/W5S5 |
| 1.7737 | 1.1455 | 0.5683 | 0.3847 | 0.4414 | 0.2742 |
| S5R5/R5S5 | W5W5 | W5R5/R5W5 | R5R5 | | |
| 0.2420 | 0.2169 | 0.2056 | 0.1919 | | |

In part (b), I use MATLAB 2019b to realize unsupervised and supervised learning in order to explore classifiers. For the unsupervised learning, I will implement my own K-means algorithm for test image clustering based on the 3-D and 15-D feature. One thing I need to say is that I acknowledge the final classes of predictions based on the observation by eyes. For the 3-D, the results are shown in the Table 4 and the error rate is 0.25. For the 15-D, the results are shown in the Table 5 and the error rate is 0.25.From the Table 4 and the Table 5, it's interesting to see that two results are exactly the same! Perhaps the number of dimensions doesn't influence the classification results from my experiment. But actually, they are because my experiment fails to represent all. From my point of view, the error rate is largely due to how do I initialize four first centroids to let them begin iterating. According to the TA's hints, how to select initial cluster centers plays an important role in determining the final results ranging from error rate to which test is exactly predicted correctly. What's more, for the supervised learning, first of all, I use the Random Forest method. The results are shown in the Table 6 and the error rate is 0.08. Later, in

order to achieve Support Vector Machine, I use the built-in function *fitcsvm* four times (nine 1s with twenty-seven 0s, nine 2s with twenty-seven 0s, nine 3s with twenty-seven 0s, and nine 4s with twenty-seven 0s) to train four models. Then I use those four models to predict 4 labels separately. The final results are shown in the Table 7 and the error rate is 0.08.

Table 4: The results of K-means based on 3-D vectors

| **Test** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Labels** | 3 | 1 | 1 | 2 | 4 | 3 | 2 | 4 | 4 | 2 | 1 | 3 |
| **Predict** | 3 | 2 | 4 | 2 | 4 | 3 | 2 | 4 | 4 | 2 | 2 | 3 |

Table 5: The results of K-means based on 15-D vectors

| **Test** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Labels** | 3 | 1 | 1 | 2 | 4 | 3 | 2 | 4 | 4 | 2 | 1 | 3 |
| **Predict** | 3 | 2 | 4 | 2 | 4 | 3 | 2 | 4 | 4 | 2 | 2 | 3 |

Table 6: The results of RF based on 3-D vectors

| **Test** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Labels** | 3 | 1 | 1 | 2 | 4 | 3 | 2 | 4 | 4 | 2 | 1 | 3 |
| **Predict** | 3 | 1 | 1 | 2 | 4 | 3 | 2 | 4 | 4 | 3 | 1 | 3 |

Table 7: The results of SVM based on 3-D vectors

| **Test** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Labels** | 3 | 1 | 1 | 2 | 4 | 3 | 2 | 4 | 4 | 2 | 1 | 3 |
| **Predict** | 3 | 1 | 1 | 2 | 4 | 3 | 2 | 4 | 4 | 2 | 0 | 3 |

In part (c), I use MATLAB 2019b to texture segmentation. The procedure has already been stated clearly previously. The texture segmentation result is shown in the Fig. 1.7. It's not hard to see that there are many annoying holes which fail to denote six segmented regions clearly. So, in part (d), I develop a post-processing technique to merge all the holes as much as possible. In the meanwhile, I also try my best to make six segmented regions have the same grayscale. The texture segmentation result after post-processing is shown in Fig. 1.8.

## Problem 2: Image Feature Extractors (40%)

(a) Salient Point Descriptor (Basic: 10%)

(b) Image Matching (Basic: 20%)

(c) Bag of Words (Advanced: 10%)

### 2.1. Abstract and Motivation

In the field of digital image processing, feature extraction, which is related to dimensionality reduction, starts from an initial set of measured data and then builds derived values that ought to be informative and non-redundant, thus making it possible to facilitate the subsequent learning and generalization steps.

SIFT key points of objects are firstly extracted from several images of interest and stored in a database. An object is recognized or matched in a new image by individually comparing each feature from the new image to this database, during which the algorithms try to find candidate matching features usually based on Euclidean distance of their feature vectors. Among the full set of matches, subsets of key points agreeing on the object and its location, scale, and orientation in new images are considered to be desirable matches. Image matching is a common issue in Computer Vision, whose methods can be divided into three classes: signal-based matching, feature-based matching and structural matching.

The bag-of-words model is a simplifying representation widely used in natural language processing and information retrieval. In this model, a text (such as a sentence or a document) is represented as the bag of its words after disregarding grammar and word order but keeping multiplicity. However, the bag-of-words model can also be used for computer vision, where the occurrence of each image feature can be used for training a classifier.

In this report, firstly I will read the paper [3] to answer five questions and then use MATLAB 2019b to realize image matching and bag-of-word thinking.

### 2.2. Approach and Procedures

(a) Salient Point Descriptor

**Question (1):**

As for the geometric modifications, the SIFT features are invariant to image scaling and rotation.

**Question (2):**

First of all, for the invariance to image scaling, the paper firstly implements the keypoint detection by identifying locations and scales that can be repeatably assigned under different views of the same object with the help of Gaussian and Laplacian pyramids. Then if we want to detect locations invariant to scale change of the image, we can search for stable image features across all possible scales by using a continuous function of scale known as scale space. We can define the scale space of an image as $L(x, y, \sigma)$, which is produced from the convolution of a variable-scale Gaussian, $G(x, y, \sigma)$, with an input image, $I(x, y)$:

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

and

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

and the difference-of-Gaussian function convolved with the image, can be computed from the difference of two nearby scales separately by a constant multiplicative factor k:

$$\begin{aligned} D(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, \sigma)\big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma). \end{aligned}$$

Later, we find local extrema of the difference-of-Gaussian images by comparing a pixel to its 26 neighbors in 3 by 3 regions at the current and adjacent scales. Then we have to eliminate points that have low contrast or are poorly localized along an edge.

Secondly, for the invariance to image rotation, the paper askes people to assign a consistent orientation to each keypoint based on local image properties to create a keypoint descriptor that can be represented relative to this orientation and therefore achieve invariance to image rotation. The procedure is shown as follows:

Step1: For each image sample, $L(x, y)$, at this scale, the gradient magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, are precomputed using pixel differences:

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

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

Step2: A histogram is formed by quantizing the orientations of the 360-degree range into 36 bins. Peaks in the orientation histogram correspond to dominant directions of local gradients. The highest peak in the histogram is detected, and then any other local peak that is within 80% of the highest peak is also used to create a keypoint with that orientation.

Step3: Compute relative orientation and magnitude in a 16*16 neighborhood at each key pint.

**Question (3):**

We should compute a descriptor for the local image region that is highly distinctive as invariant as possible to remaining variations, such as change in illumination or 3D viewpoint.

Step1: The magnitude of each sample point is weighted by a Gaussian function with $\sigma$ equal to one half the width of the descriptor window.

Step2: Form the weighted orientation histograms (8 bin) for 4×4 regions by summing the gradient magnitudes near that direction within the region.

Step3: Distribute each sample to adjacent histogram bins by trilinear interpolation in order to avoid all boundary affects, where the descriptor abruptly changes as a sample shift smoothly from being within one histogram to another or from one orientation to another.

Step4: Concatenate 16 histograms in one long vector of 128 dimensions for each keypoint.

Step5: Normalize the feature vectors to unit length, aimed at reducing effect of illumination change because a change in image contrast, where each pixel value is multiplied by a constant, will multiply gradients by the same constant, so this contrast change will be canceled by vector normalization. A brightness change, where a constant is added to each image pixel, will not affect the gradient values, as they are computed from pixel differences.

Step6: Reduce the influence of large gradient magnitudes by thresholding the values in the unit feature vector to each no larger than 0.2, and then renormalizing to unit length in order to reduce non-linear illumination change.

**Question (4):**

First of all, the Difference of Gaussian (DoG) function is efficient to compute. It can't be denied that the smoothed images, $L$, need computing in any case for scale space feature description, and in that case, D can be computed by a simple image subtraction.

Secondly, the DoG function offers a close approximation to the scale-normalized Laplacian of Gaussian, $\sigma^2\nabla^2 G$, as studied by Lindeberg (1994), who showed that the normalization of the Laplacian with the factor $\sigma^2$ is required for true scale invariance. We can know

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

and therefore

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

which shows that when the DoG function has scales differing by a constant factor, it already incorporates the $\sigma^2$ scale normalization that is required for the scale-invariant Laplacian.

**Question (5):**

The paper uses a 128-element-feature-vector for each keypoint.

(b) Image Matching

In this part, I will use the online source from [4] to apply SIFT to finish some tasks of object matching.

First of all, use the built-in function *vl_sift* to find key-points of the two Husky images in the Fig. 2.1 and the Fig. 2.3. Then pick the key-point with the largest scale in the Fig. 2.3 and use the built-in function *vl_ubcmatch* to find its closest neighboring key-point in the Fig. 2.1.

What' more, use the built-in function *vl_sift* to show the corresponding SIFT pairs between the Fig. 2.1 and the Fig. 2.3. Then perform the same job with the following three image pairs: the Fig. 2.3 with the Fig. 2.2, the Fig. 2.3 with the Fig. 2.4, and the Fig. 2.1 with the Fig. 2.4.

(c) Bag of Words

Each image can be represented as a histogram of SIFT feature vectors. I will apply the K-means clustering to extracted SIFT features to form a codebook. The procedure is shown as follows:

Step1: Use the built-in function *vl_sift* to extract SIFT features from training images.

Step2: Apply k (k=8) clusters, which are cluster centroids, to create dictionary's visual words.

Step3: Calculate its nearest neighbor in terms of different centroids in the dictionary.

Step4: Build histograms of length k (new features).

Then create codewords for all four images and match the Fig. 2.3's codewords with other images by letting the 8 centroids of the Fig. 2.3 as the standard to build three histograms of length 8.

## 2.3. Experimental Results

Fig. 2.1: The original “Husky_1” image

Fig. 2.2: The original “Husky_2” image

Fig. 2.3: The original “Husky_3” image

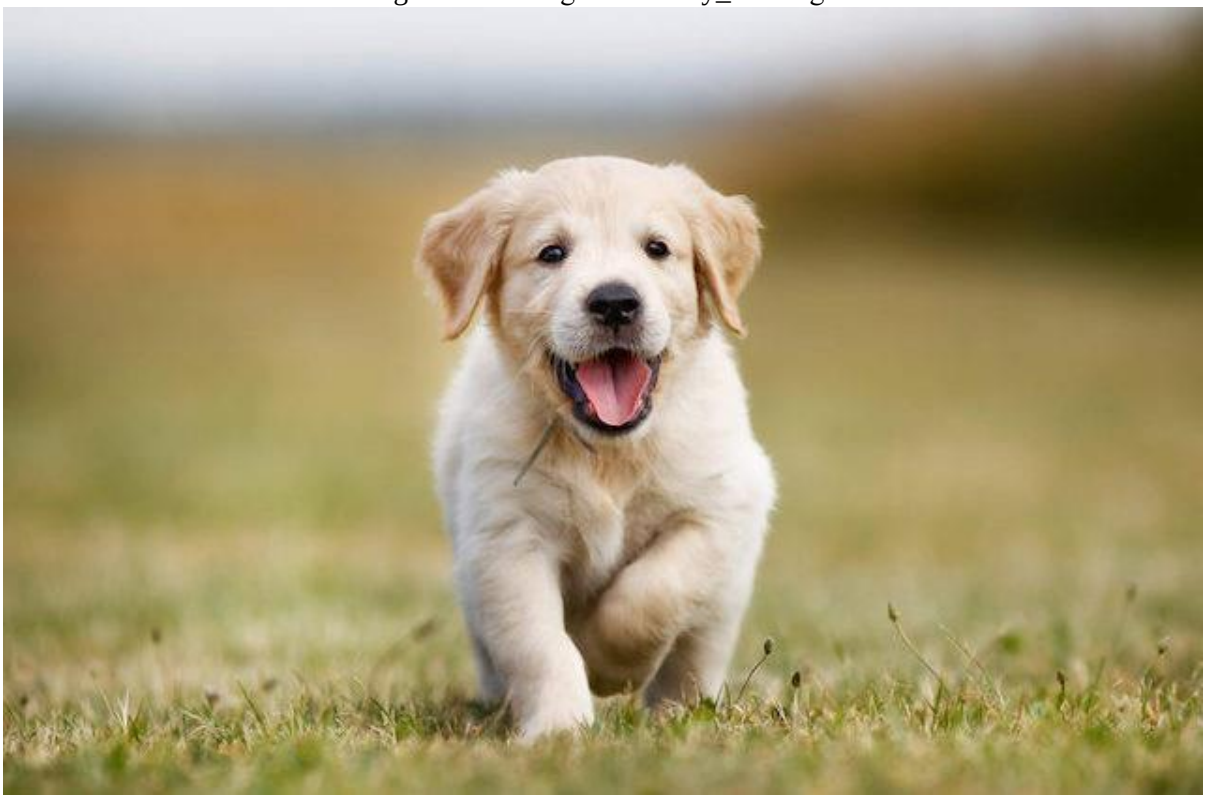

Fig. 2.4: The original “Puppy_1” image

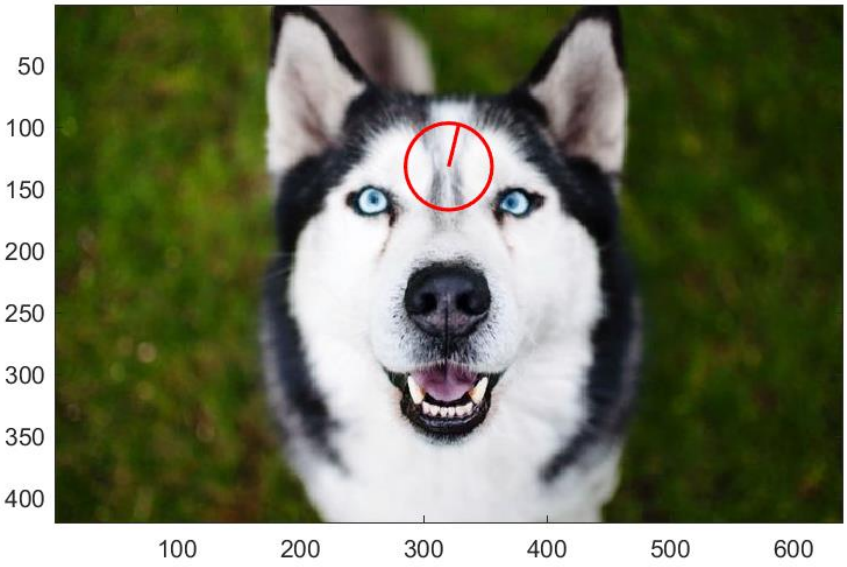


Fig. 2.5: The keypoint with the largest scale in the image “Husky_3”

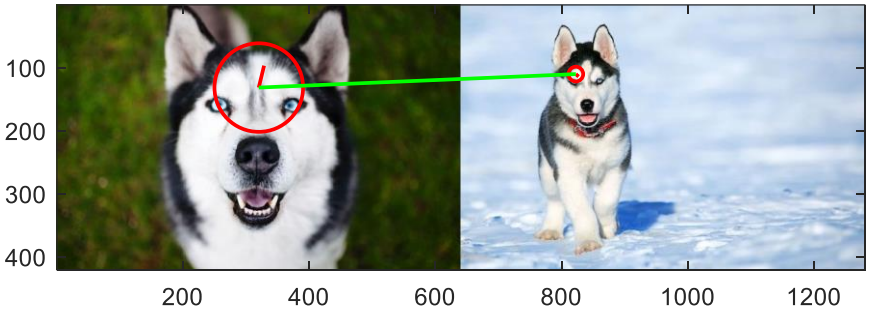


Fig. 2.6: The keypoint with the largest scale in the image “Husky_3” and its closest neighboring keypoint in the image “Husky_1”

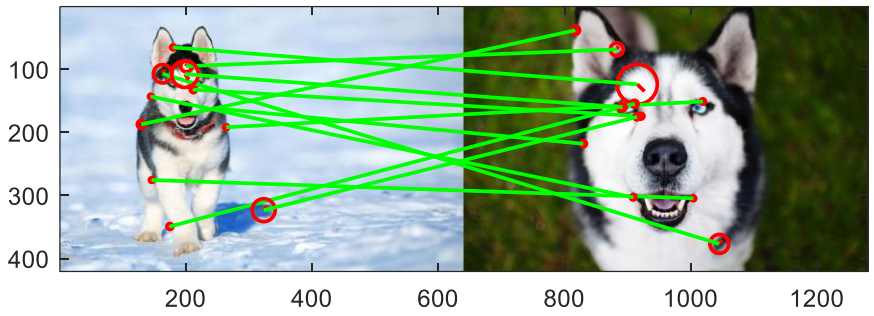


Fig. 2.7: The corresponding SIFT pairs between the image “Husky_1” and the image “Husky_3”

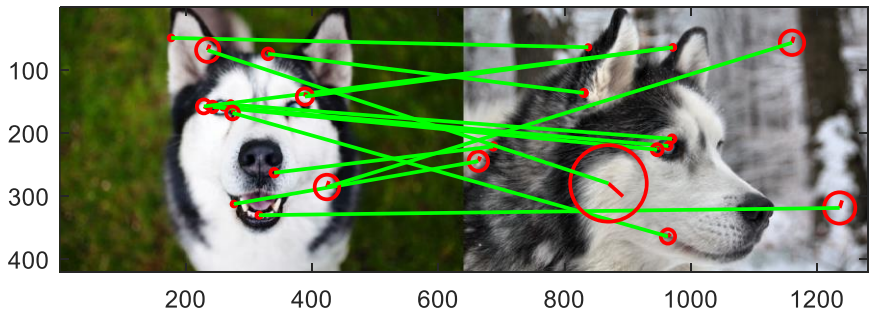


Fig. 2.8: The corresponding SIFT pairs between the image “Husky_3” and the image “Husky_2”

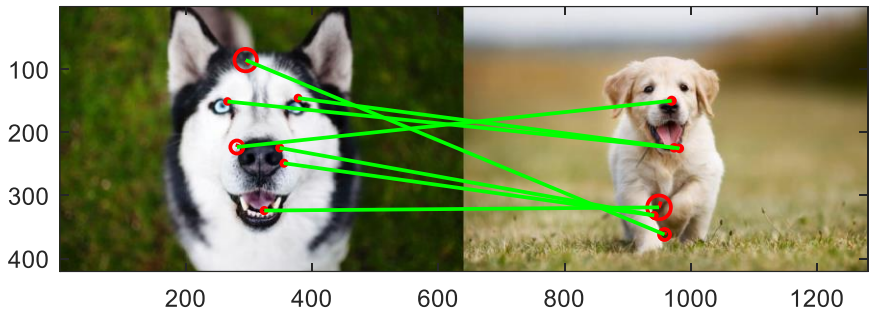


Fig. 2.9: The corresponding SIFT pairs between
the image “Husky_3” and the image “Puppy_1”

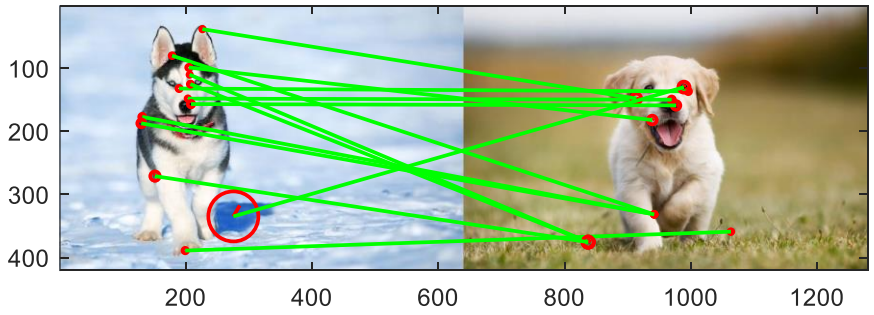


Fig. 2.10: The corresponding SIFT pairs between
the image “Husky_1” and the image “Puppy_1”

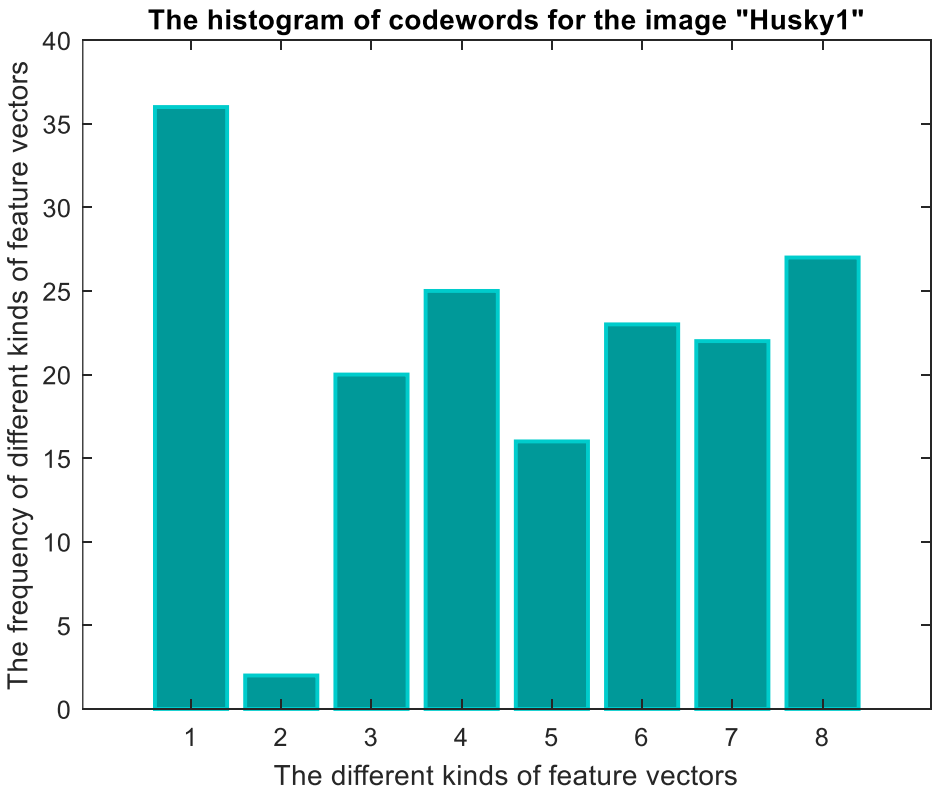


Fig. 2.11: The codewords for the image “Husky_1”

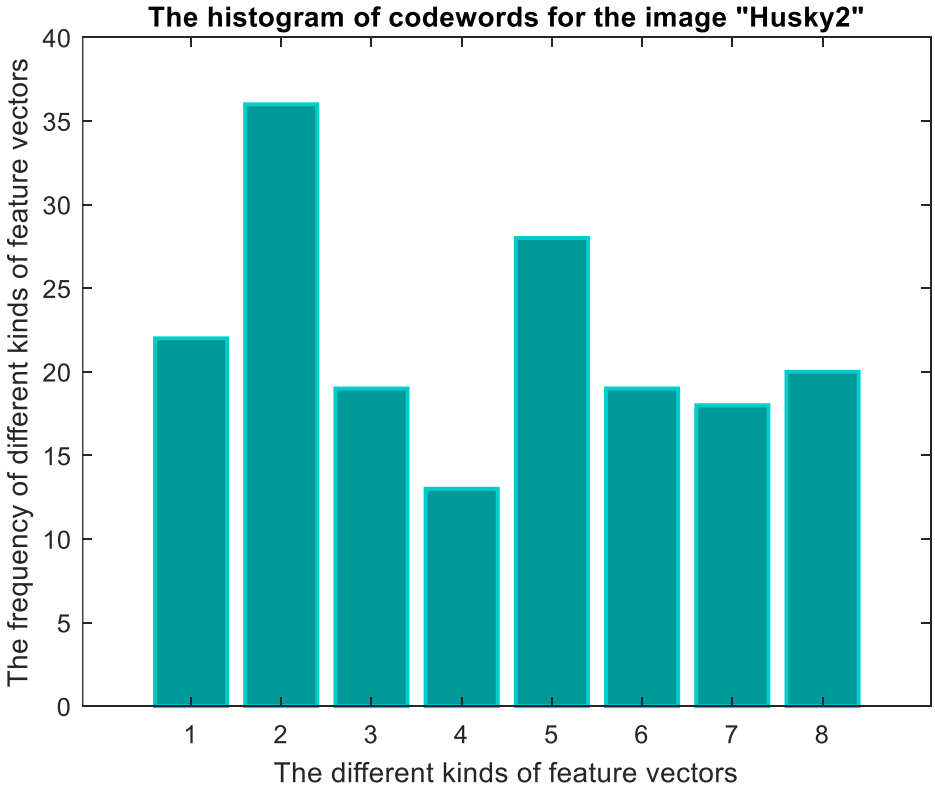


Fig. 2.12: The codewords for the image “Husky_2”

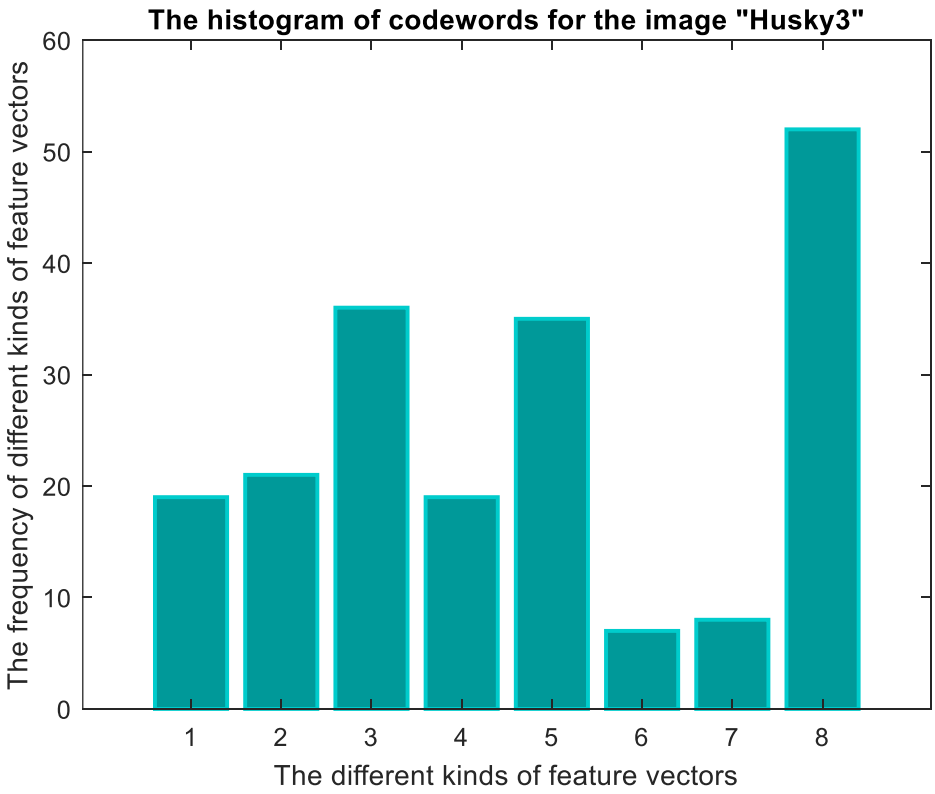


Fig. 2.13: The codewords for the image “Husky_3”

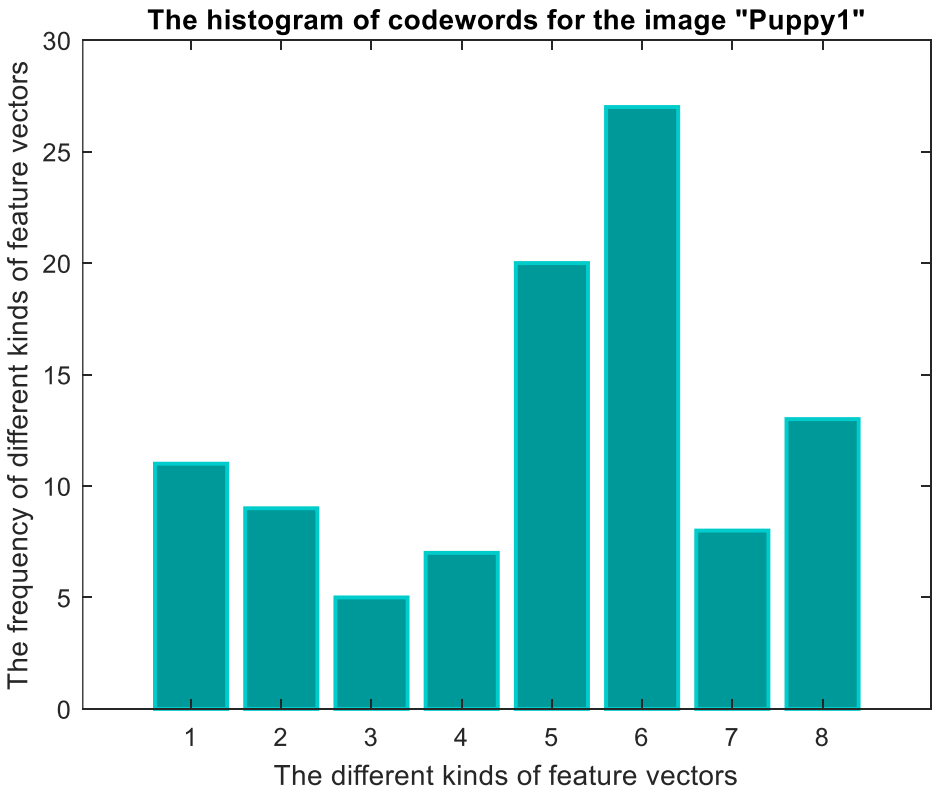


Fig. 2.14: The codewords for the image “Puppy _1”

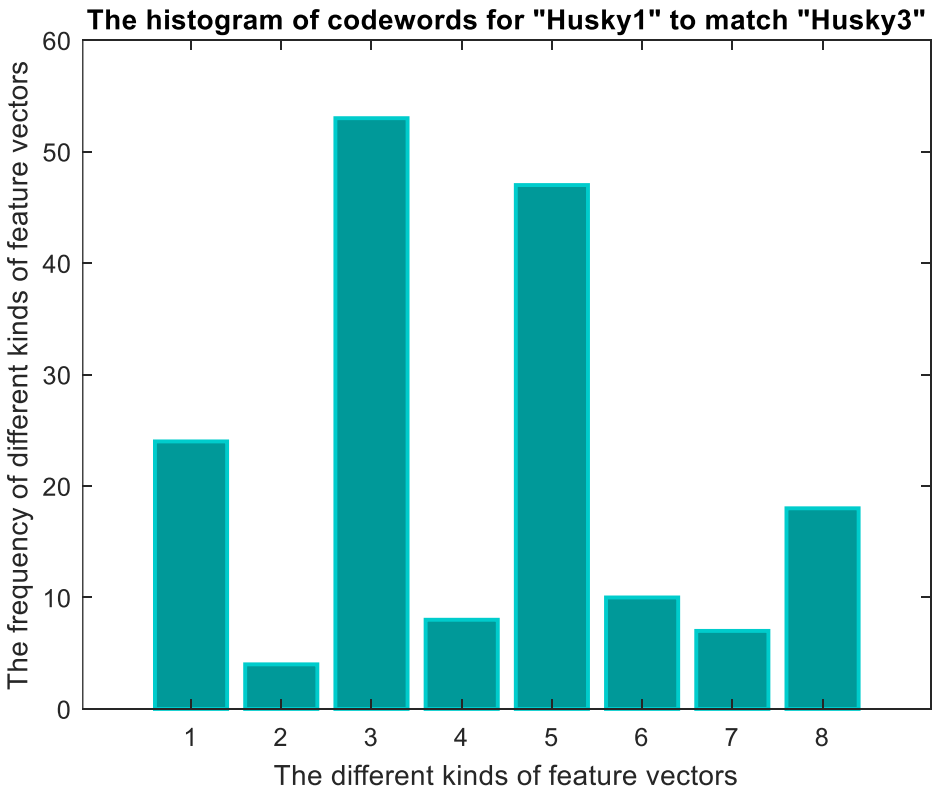


Fig. 2.15: The codewords for the image “Husky_1” to match the image “Husky_3”

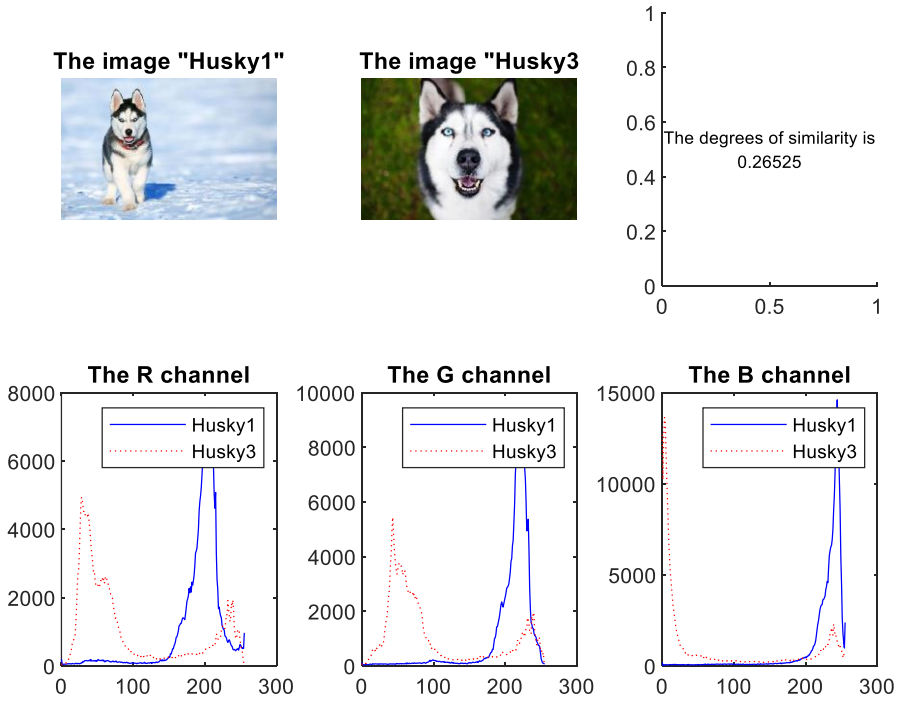


Fig. 2.16: The similarity check between the image “Husky_1” and the image “Husky_3”

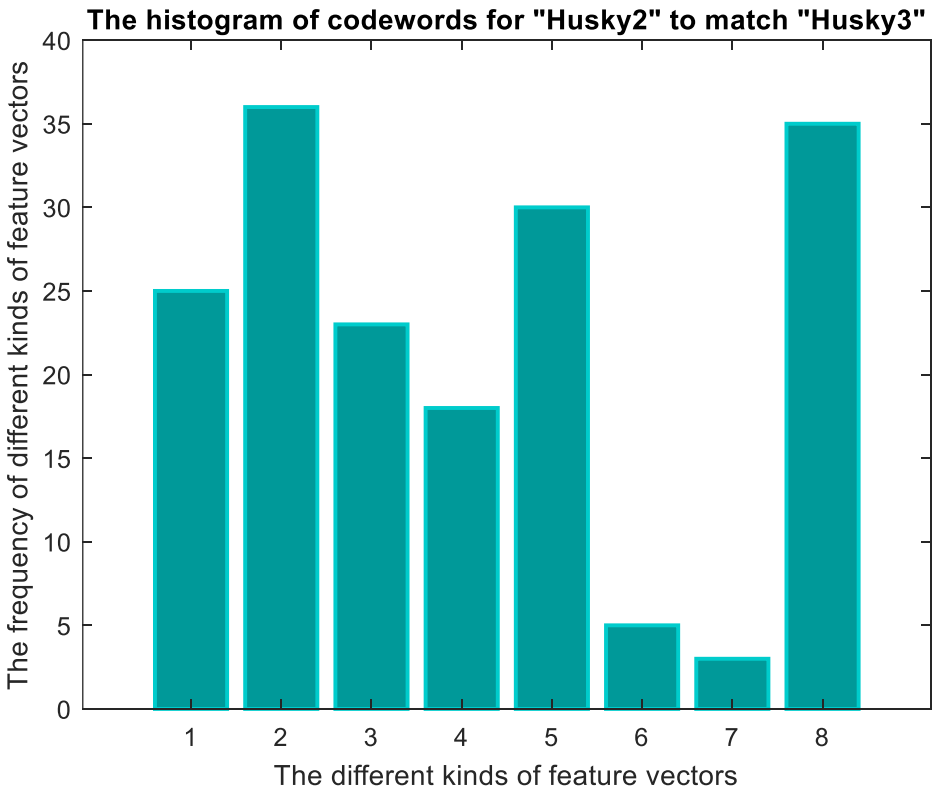


Fig. 2.17: The codewords for the image “Husky_2” to match the image “Husky_3”

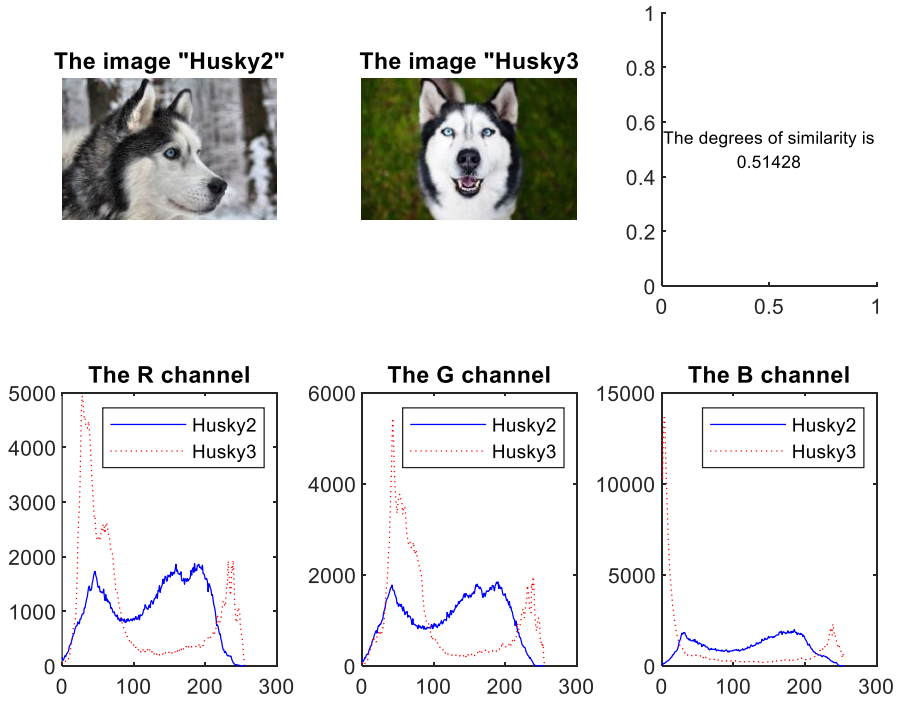


Fig. 2.18: The similarity check between the image “Husky_2” and the image “Husky_3”

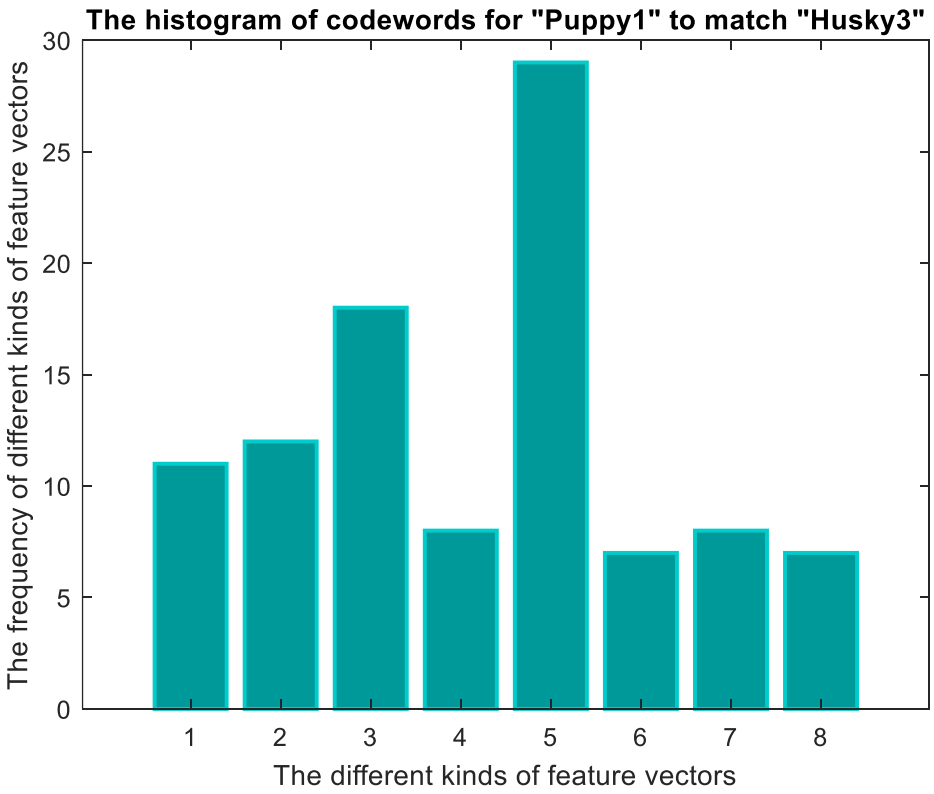


Fig. 2.19: The codewords for the image “Puppy _1” to match the image “Husky_3”

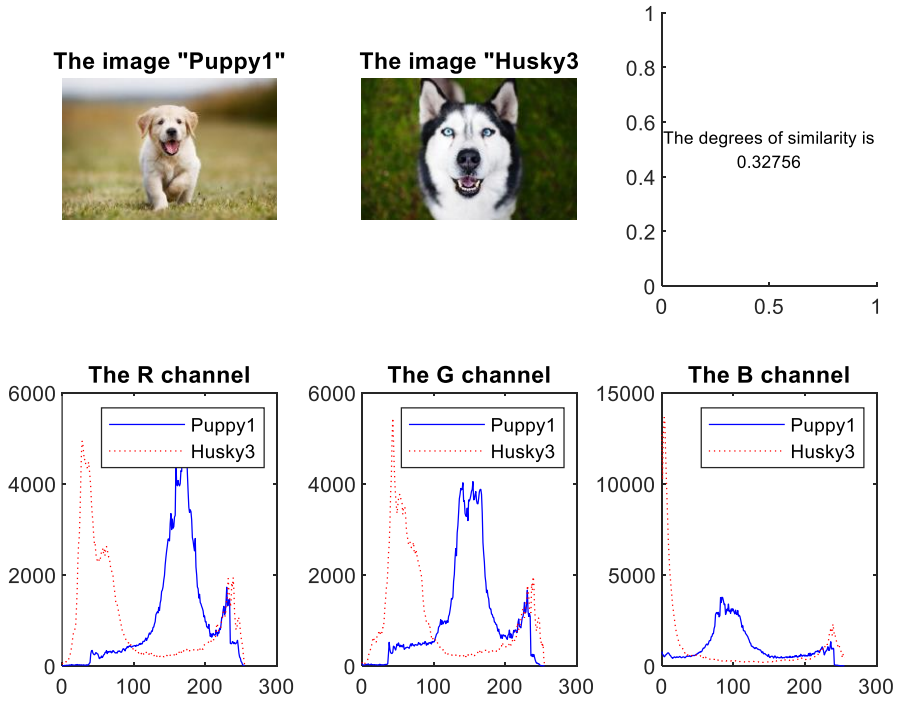


Fig. 2.20: The similarity check between the image “Puppy _1” and the image “Husky_3”

### 2.4. Discussion

As stated before, in this report, I will merely use my own judgement to evaluate the visual quality of resulting images. My discussion of experimental results of Problem 2 is stated as below.

In part (a), I have already answered five questions above.

In part (b), firstly, I have already picked the key-point with the largest scale in the image "Husky_3" as shown in the Fig. 2.5 with the help of the built-in function *vl_sift*. Then I utilize the built-in function *vl_ubcmatch* to find that key-point's closest one in the image "Husky_1" as shown in the Fig. 2.6. One thing that needs to be mentioned is that although my final result in the Fig. 2.6 isn't desirable enough, yet at first the key-point in the image "Husky_1" is located on the snow! At that moment, I checked the tutorial of the functions *vl_sift* and *vl_ubcmatch* and found out that I could modify the parameters of those two functions to make results look better. After annoyingly setting parameters manually, finally I let the function *vl_sift* 's 'Levels' equal to 5 and 'PeakThresh' equal to 4, and the function *vl_ubcmatch*'s threshold value equal to 1 to get the result shown in the Fig. 2.6. How to select the sizes and combinations of parameters is an empirical process, indicating that my result might not be the best. Also, we can see in the Fig. 2.4 that two orientations of two keypoints are not the same because the dog "Husky_3" is flinging back its head and staring at the sky and the dog "Husky_1" is looking straight ahead.

What's more, the corresponding SIFT pairs between the "Husky_1" and "Husky_3", "Husky_3" and "Husky_2", "Husky_3" and "Puppy_1", and "Husky_1" and "Puppy_1" are shown in the Fig. 2.7, the Fig. 2.8, the Fig. 2.9 and the Fig. 2.10 respectively. For the Fig. 2.7 and the Fig. 2.8, some keypoints locating on the eyes or ears, to some extent, can be matched well because those features are especial enough. However, some keypoints are not soundly matched. I think the reason for failure in some cases is that two dogs' texture of bodies is not exactly the same, making it possible that the features of the dog's face like the ones of the dog's leg. Also, how to select parameters can also contribute to the visual quality of final results. Moreover, for the Fig. 2.9 and the Fig. 2.10, because two dogs belong to two totally different breeds. Hence, we can't think about having good matching results. But we can see that the noses of two dogs and other similar features can be matched. In a word, the parameters of functions and similarity between two original images play a significant role in determining the final results of image matching.

In part (c), the histograms of SIFT feature vectors of "Husky_1", "Husky_2", "Husky_3" and "Puppy_1" are shown in the Fig. 2.11, Fig. 2.12, Fig. 2.13, and the Fig. 2.14 respectively. It's notable that I will acquire different histograms every time I run the code due to the randomness of K-means. Also, because initialization is crucial to the effect of the K-means algorithm, I just use the built-in K-means function. But in the Problem 1 I use my own function. Later, by regarding the "Husky_3" as the standard, I create codewords for Husky_1", "Husky_2", and "Puppy_1" to match it separately, as shown in the Fig. 2.15, the Fig. 2.17, and the Fig. 2.19. Then I found the source code online from [5] to check similarity between two images. As a result, from the Fig. 2.18, the "Husky_2" and the "Husky_3" is most similar. I believe this algorithm is not robust to scale or illuminance, thus leading to the least similarity between the "Husky_1" and the "Husky_3" . But after using the Euclidean-distance method by comparing the Fig. 2.15, the Fig. 2.17, the Fig. 2.19 with the Fig. 2.13, "Husky_1" and "Husky_2" are more similar to "Husky_3" than "Puppy_1".

**References**

[1] [Online] Available: https://www.mathworks.com/help/stats/treebagger.html

[2] [Online] Available: https://www.mathworks.com/help/stats/support-vector-machines-for-binary-classification.html#bsr5b6n

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

[4] [Online] Available: https://www.vlfeat.org/index.html

[5] [Online] Available: https://blog.csdn.net/weixin_42183170/ article/details/100546807?ops_request_misc=%257B%2522request%255Fid%2522%253A%252215849316781972484641229 5%2522%252C%2522scm%2522%253A%252220140713.130056874..%2522%257D&request_id=158493167819724846412295&biz_id=0&utm_source=distribute.pc_search_result.none-task