CS303B Assignment 1 Team Report

Yihan Sun, Xudong Zhang, Xuhan Kong
Department of Electronic and Electrical Engineering, SUSTech
1088 Xueyuan Avenue, Nanshan District, Shenzhen, Guangdong, China
12012128@mail.sustech.edu.cn, 12011923@mail.sustech.edu.cn,
12012112@mail.sustech.edu.cn

Abstract

In assignment 1 in 2023 fall CS303B, our objective is to develop, evaluate, and report on image processing software to automatically create composite images of various types, using images automatically selected as being of certain classes of scene, which is divided into three parts, Composite image generation, Binary image classification, and Deduplication and re-identification. This report investigates methods for computing image similarity, image classification, and duplicate image recognition in computer vision. In terms of system and algorithm design, the report proposes algorithms for generating composite images, binary image classification, and deduplication recognition. Experimental results showcase the software's performance across various tasks, and discussions on improvements and optimizations are provided. Overall, the report empirically demonstrates the effectiveness of the designed software in computer vision tasks.

1. Introduction

In recent years, the dynamic field of computer vision has become integral to numerous applications, ranging from autonomous vehicles and medical imaging to facial recognition and augmented reality. Computer vision involves the extraction of meaningful information from visual data, enabling machines to interpret and comprehend the visual world much like humans. One fundamental aspect of computer vision is image processing and analysis, which plays a pivotal role in tasks such as image classification, object detection, and similarity measurement. In this assignment report, we focus on methods to calculate image similarities, classify images, and identify duplicate images.

The rest of this report is organized as follows. Section 2 introduces some related work and methods for image similarities calculation, image classification, and image duplication detection, and Section 3 describes the design of algorithm in our software for each part task. In Section 4, we provide experimental results in each part to evaluate the performance of our software. Finally, Section 5 concludes the report.

2. Related Work and Methods

In this section, we will introduce some related methods which are commonly used in computer vision tasks which include image similarity comparison methods, image classification methods, clustering algorithms, and so on.

2.1. Compare the Similarity of Two Histograms

In the realm of data analysis and pattern recognition, comparing the similarity of histograms has emerged as a critical task with applications spanning various domains. Histograms provide a concise representation of the distribution of data, making them valuable tools for understanding and comparing datasets. Numerous methods have been proposed to quantify the similarity between two histograms, each with its strengths and limitations. We will show some common approaches include Correlation-based Approaches, Bhattacharyya Distance, Euclidean Distance, Chi-Square Test, and Histogram Intersection. Each method has its advantages and limitations, catering to different characteristics of datasets.

2.1.1 Correlation-based Approaches

One widely adopted approach involves measuring the correlation between two histograms. The correlation comparison formula is as follows:

$$D(h_1, h_2) = \frac{\sum_i (h_1(i) - \overline{h_1})(h_2(i) - \overline{h_2})}{\sqrt{\sum_i (h_1(i) - \overline{h_1})^2 \sum_i (h_2(i) - \overline{h_2})^2}} \qquad (1)$$

Where $\overline{h_k} = \frac{1}{N} \sum_j h_k(j)$.

In this case, if $h_1 = h_2$, i.e., the histograms of two images are identical, the numerator equals the denominator, resulting in a value of 1. Therefore, in a less strict scenario, when the value is 1, it can be considered that the two images are identical. However, it is also possible for two images to be different while their histograms are the same. This discrepancy arises because histogram computation reflects the distribution of pixel counts but does not indicate the spatial arrangement of pixels. Consequently, there may be instances where two images exhibit dissimilar overall appearances but share an identical distribution of pixel counts. In such cases, the histograms are still equivalent, although these situations are relatively uncommon [1].

2.1.2 Bhattacharyya Distance

The Bhattacharyya distance is a statistical measure that quantifies the similarity between two probability distributions. It is based on the Bhattacharyya coefficient, which is a measure of the similarity of two probability distributions [2, 3].

The Bhattacharyya coefficient is a measure of overlap between two statistical samples or populations. It quantifies the similarity between two probability distributions and is often used as a precursor to computing the Bhattacharyya distance, which can be calculated as follows:

$$\mathrm{BC} = \sum_{i=1}^{N} \sqrt{p(i) \cdot q(i)} \tag{2}$$

For continuous probability distributions, the Bhattacharyya coefficient can be expressed as an integral:

$$\mathrm{BC} = \int \sqrt{p(x) \cdot q(x)}\, dx \tag{3}$$

Where $p(x)$ and $q(x)$ represent the probability density functions of the two distributions.

The Bhattacharyya coefficient ranges from 0 to 1, where 1 indicates complete overlap and 0 indicates no overlap.

Mathematically, the Bhattacharyya distance between two continuous probability density functions is defined as follows:

$$D_B = -\ln(\mathrm{BC}) = -\ln \left(\sum_{i=1}^{N} \sqrt{p(i) \cdot q(i)}\right) \tag{4}$$

The Bhattacharyya distance ranges from 0 to infinity, where 0 indicates perfect similarity and higher values indicate increasing dissimilarity [3].

2.1.3 Euclidean Distance

In image processing, Euclidean Distance is commonly employed for calculating the similarity of image histograms. The Euclidean Distance is calculated using the formula as follows:

$$D(h, h') = \sqrt{\sum_{i=0}^{n} (h(i) - h'(i))^2} \tag{5}$$

Apply the Euclidean Distance formula to the two normalized histograms. A smaller Euclidean Distance indicates greater similarity between the two images. The distance value can be transformed into a similarity score, for example, by taking the reciprocal or applying an exponential function.

This method is straightforward, but it has limitations such as insensitivity to color distribution and lack of robustness to image transformations.

2.1.4 Chi-Square Distance

The Chi-Squared distance, also known as the Chi-Squared test, measures the difference between two histograms by comparing the observed frequencies to the expected frequencies. The formula for the Chi-Square Test is:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{6}$$

Where O_i is the observed frequency in bin i, E_i is the expected frequency in bin i.

The Chi-Squared distance is sensitive to large differences between the observed and expected frequencies, and is commonly used in hypothesis testing to determine if two histograms come from the same distribution [2].

2.1.5 Histogram Intersection

Histogram intersection is a method for comparing two histograms, often used in image processing and computer vision [4]. It provides a straightforward measure of similarity by calculating the area under the minimum values of corresponding bins in two histograms. This method is computationally efficient and particularly useful when capturing local similarities between distributions.

2.2. Feature Vector Extraction

Feature vector extraction of images is a critical step in image processing and machine learning. It involves capturing meaningful information from images and representing it as a numerical vector. Various algorithms and techniques are employed for feature vector extraction of images, each with its strengths and limitations. We will show some common methods used in this field, including feature vector extraction based on histogram, HOG (histogram of oriented gradient) and SIFT (Scale-Invariant Feature Transform).

2.2.1 HOG

Histogram of Oriented Gradients (HOG) is a feature descriptor widely used in computer vision and image processing. It is a method for capturing and representing the local intensity gradients in an image. The technique is based on counting occurrences of gradient orientations in localized portions of an image. The step of calculating HOG descriptor is as following:

1. Resize the image into an image of 128×64 pixels (128 pixels height and 64 width)
2. The gradient of the image is calculated. The gradient is obtained by combining magnitude and angle from the image. Considering a block of 3×3 pixels.

$$G_x(r,c) = I(r,c+1) - I(r,c-1) \qquad (7)$$

$$G_y(r,c) = I(r+1,c) - I(r-1,c) \qquad (8)$$

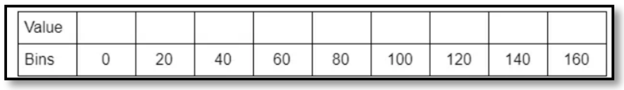
$$Magnitude(\mu) = \sqrt{G_x^2 + G_y^2} \qquad (9)$$

$$Angle(\theta) = |tan^{-1}(\frac{G_y}{G_x})| \qquad (10)$$

3. After obtaining the gradient of each pixel, the gradient matrices (magnitude and angle matrix) are divided into 8×8 cells to form a block. For each block, a 9-point histogram is calculated. A 9-point histogram develops a histogram with 9 bins and each bin has an angle range of 20 degrees

$$number\ of\ bins = 9\ (ranging\ from\ 0^\circ\ to\ 180^\circ) \qquad (11)$$

$$\Delta\theta = \frac{180^\circ}{number\ of\ bins} = 20^\circ \qquad (12)$$



Value									
Bins	0	20	40	60	80	100	120	140	160

Figure 1. Representation of a 9 bins histogram.

4. For each cell in a block, we will first calculate the jth bin and then the value that will be provided to the jth and $(j+1)th$ bin respectively. The value is given by the following formula:

$$j = \left\lfloor \frac{\theta}{\Delta\theta} - \frac{1}{2} \right\rfloor \qquad (13)$$

$$V_j = \mu \cdot \left\lfloor \frac{\theta}{\Delta\theta} - \frac{1}{2} \right\rfloor \qquad (14)$$

$$V_{j+1} = \mu \cdot \left\lfloor \frac{\theta - \Delta\theta(j+0.5)}{\Delta\theta} \right\rfloor \qquad (15)$$

5. An array is taken as a bin for a block and values of V_j and V_{j+1} is appended in the array at the index of jth and $(j+1)th$ bin calculated for each pixel.
6. The resultant matrix after the above calculations will have the shape of $16x8x9$
7. Once histogram computation is over for all blocks, 4 blocks from the 9-point histogram matrix are clubbed together to form a new block $(2x2)$. This clubbing is done in an overlapping manner with a stride of 8 pixels. For all 4 cells in a block, we concatenate all the 9-point histograms for each constituent cell to form a 36-feature vector.

$$f_{bi} = [b_1, b_2, b_3, \ldots, b_{36}] \qquad (16)$$

8. Values of f_{bi} for each block is normalized by the $L2$ norm:

$$f_{bi} = \frac{f_{bi}}{\sqrt{\|f_{bi}\|^2 + \epsilon}} \qquad (17)$$

9. To normalize, the value of k is first calculated by the following formulae:

$$k = \sqrt{b_1^2 + b_2^2 + b_3^2 + \cdots + b_{36}^2} \qquad (18)$$

$$f_{bi} = \left[\left(\frac{b_1}{k}\right), \left(\frac{b_2}{k}\right), \left(\frac{b_3}{k}\right), \ldots, \left(\frac{b_{36}}{k}\right)\right] \qquad (19)$$

10. This normalization is done to reduce the effect of changes in contrast between images of the same object. From each block. A 36-point feature vector is collected. In the horizontal direction there are 7 blocks and in the vertical direction there are 15 blocks. So, the total length of HOG features will be: $7\ x\ 15\ x\ 36\ =\ 3780$.

HOG is robust to variations in lighting and contrast, enhancing its performance in diverse environments. However, HOG may be sensitive to noise in images, impacting its performance in noisy environments [5].

2.2.2 SIFT

SIFT, or Scale-Invariant Feature Transform, is a computer vision algorithm developed by David G. Lowe for detecting and describing local features in images. The key characteristics of SIFT include its ability to identify features irrespective of scale, rotation, and illumination changes. The algorithm works by identifying key points and extracting distinctive descriptors to represent them. This makes SIFT valuable in various computer vision applications, such as object recognition, image stitching, and 3D reconstruction. There are mainly four steps involved in SIFT algorithm.

1. Scale-space Extrema Detection

SIFT algorithm uses Difference of Gaussians which is an approximation of LoG. Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and $k\sigma$. This process is done for different octaves of the image in Gaussian Pyramid. It is represented in below image:

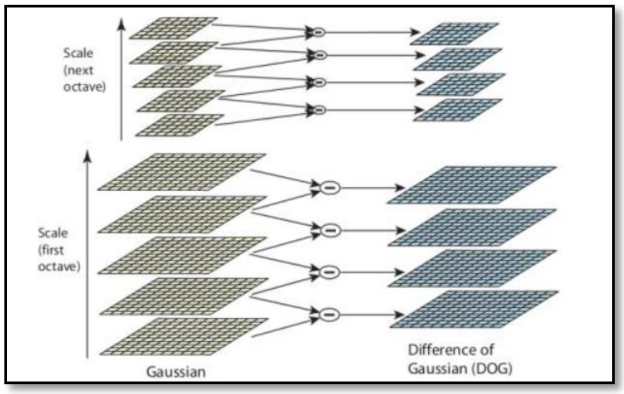


Figure 2. Difference of Gaussian.

Once this DoG are found, images are searched for local extrema over scale and space. For e.g., one pixel in an image is compared with its 8 neighbors as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extremum, it is a potential key point. It basically means that key point is best represented in that scale. It is shown in below image:

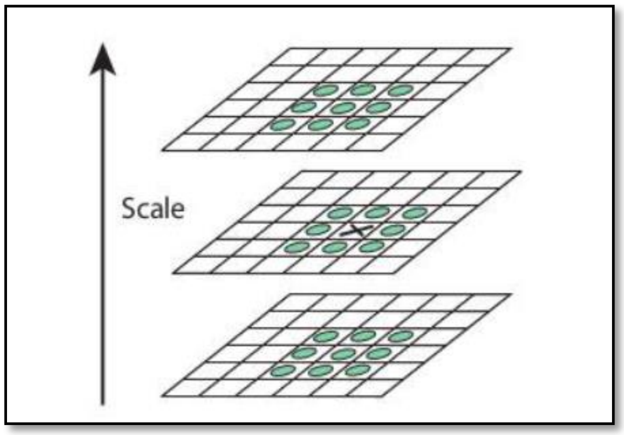


Figure 3. searched for local extrema.

Regarding different parameters, the paper gives some empirical data which can be summarized as, number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6$, $k = \sqrt{2}$ etc. as optimal values.

2. Key point Localization

Once potential key points locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extremum is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called contrast Threshold in OpenCV. DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a $2x2$ Hessian matrix (H) to compute the principal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function. If this ratio is greater than a threshold, called edge Threshold in OpenCV, that key point is discarded. It is given as 10 in paper. So, it eliminates any low-contrast key points and edge key points and what remains is strong interest points.

3. Orientation Assignment

Now an orientation is assigned to each key point to achieve invariance to image rotation. A neighborhood is taken around the key point location depending on the scale, and the gradient magnitude and direction is calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created (It is weighted by gradient magnitude and gaussian-weighted circular window with σ equal to 1.5 times the scale of key point). The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation. It creates key points with same location and scale, but different directions. It contributes to stability of matching.

4. Key point Descriptor

Now key point descriptor is created. A $16x16$ neighborhood around the key point is taken. It is divided into 16 sub-blocks of $4x4$ size. For each sub-block, 8 bin orientation histogram is created. So, a total of 128 bin values are available. It is represented as a vector to form keypoint descriptor. In addition to this, several measures are taken to achieve robustness against illumination changes, rotation etc. [6].

2.3. Classification Method

Image classification involves categorizing an image into predefined classes or labels. Various techniques are employed for image classification. We will show some common methods used in this field, including K-nearest neighbor classifier and SVM (support vector machine).

2.3.1 K-nearest Neighbor Classifier

The K-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm used for classification and regression. It works by finding the K nearest data points in the training set to a given data point and classifying or predicting based on the majority class or average value of its neighbors.

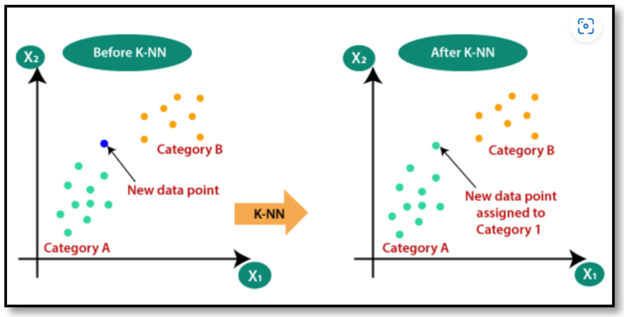


Figure 4. A simple example of KNN algorithm.

The step of K-nearest neighbor classifier is as following:

1. Select the number K of the neighbors
2. Calculate the Euclidean distance of K number of neighbors.
3. Take the K nearest neighbors as per the calculated Euclidean distance.
4. Among these k neighbors, count the number of the data points in each category.
5. Assign the new data points to that category for which the number of the neighbor is maximum.
6. Our model is ready [7].

2.3.2 SVM

Support Vector Machines (SVMs) are powerful machine learning algorithms used for classification and regression tasks. They operate by finding the optimal hyperplane that separates different classes in a high-dimensional space. SVM is a supervised learning algorithm that excels in both classification and regression tasks. It's particularly effective in high-dimensional spaces and is widely used in various domains, including image recognition, text classification, and bioinformatics. SVM models are fitted by identifying the hyperplane that maximizes the margin between classes.

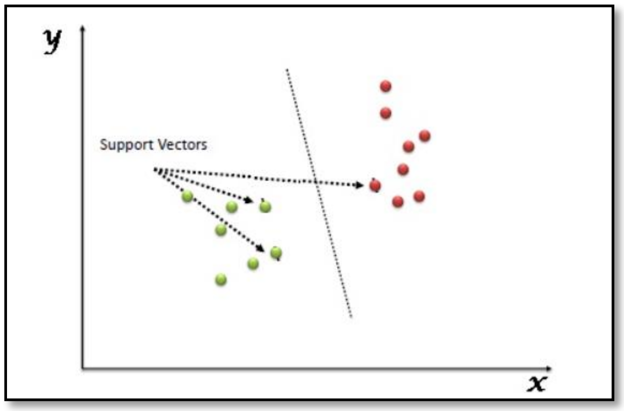


Figure 5. A simple example of SVM algorithm.

In the SVM algorithm, each data item is plotted as a point in n-dimensional space (where n is the number of features you have), with the value of each feature being the value of a particular coordinate. Then, classification is performed by finding the optimal hyper-plane that differentiates the two classes very well. Support Vectors are simply the coordinates of individual observation, and a hyper-plane is a form of SVM visualization. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/line).

3. System and Algorithm Design

In this assignment, we will use sets of images of several hundred outdoor scenes to develop, evaluate and report on image processing software to automatically create composite images of various types, using images automatically selected as being of certain classes of scene.

Each image in images sets has been labelled as either *natural* or *manmade*, where the *natural* images are of scenes such as mountains, hills, desert, sky, and so on, and the *manmade* images are of scenes such as commercial buildings, shops, markets, cities, towns, and so on. And the images sets are divided into disjoint subsets for training and testing, where the training set has 500 natural images and 500 manmade images, and the testing set has 250 natural images and 250 manmade images.

In the following of this part, we will introduce the assignment in each part, the experiment process, and our design of algorithm in each task in details.

3.1. PART A: Composite image generation

In part A, we will develop a software which is capable of creating composite images, which is called the *target image* in this assignment. And we use both of the training set of 1000 images and the 457 expanded data source images from students in CS303B to obtain *source images*, where the *source images* are the images used to create the tiles of the composite image.

The main goal of our system design in this part is to replace each tile of the target image with the most similar one in the source images. The code also uses thread pools in concurrent programming to speed up processing.

The design of our algorithm is as follows. First, read the target image and source images from the image path. Then, resize the source images, and calculate the histogram of the R, G, B channel in both source and target images, respectively. After that, iteratively compare the histogram between the resized source images and the tiles in the target image to calculate the similarities between them, then pick out the one with the largest similarities as the final tile and put it into the corresponding place in the target image.

Besides, to improve the coherence between each tile in the final composite image, we add the smoothing method in the code, which aims to remove noise in the image as well as make the result look smoother. In our software, we use smooth method in two places, the first one is before resizing the source images, using smoothing to remove the noise in them, and the second one is after finishing the iteration to product the composite image, using smoothing to make the result look smoother. In Section 4, we will show our results and compare the difference between them.

Algorithm 1

```
1:  for x in range(0, row, tile_height):
2:      for y in range(0, column, tile_weight):
3:          Extract a tile in the corresponding place of the
            target image
4:          Calculate the histogram of the extracted tile in
            R, G, B channel, respectively
5:          Calculate the similarity between the extracted
            tile and each resized source images according to
            their histograms
6:          Put the resized source image which has the
            largest similarity with the extracted tile into the
            corresponding place in the target image
7:      end loop
8:  end loop
```

From the algorithm above, it is not hard to find that the complexity of this algorithm is extremely high. Therefore, to speed up our software, we have modified the code structure, and try to use the parallel computation, which can make code runs more than ten times faster in our image composition task.

3.2. PART B: Binary image classification

In part B, we will develop and quantitatively evaluate software to classify images as belonging to one of two classes: natural or manmade. And we use the dataset of 500 natural images and 500 manmade images as training datasets. The rest of 250 natural images and 250 manmade images as test datasets.

The design of our algorithm is as follows.

Algorithm 2

```
1:    X_train=[],Y_train=[],X_test=[],Y_test=[]
2:    for image in manmade_training images:
3:        extract feature vector of image
4:        add the feature vector into X_train
5:        Add 0 into Y_train
6:    end loop
7:    for image in natural_training images:
8:        extract feature vector of image
9:        add the feature vector into X_train
0:        Add 1 into Y_train
1:    end loop
2:    for image in manmade_test images:
3:        extract feature vector of image
4:        add the feature vector into X_test
5:        Add 0 into Y_test
6:    end loop
7:    for image in natural_test images:
8:        extract feature vector of image
9:        add the feature vector into X_test
0:        Add 1 into Y_test
1:    end loop
2:    model=classifier.fit(X_train,Y_train)
3:    Y_pred=model.predict(X_test)
4:    Accuracy=accuracyCalculate(Y_pred,Y_test)
```

3.3. PART C: Deduplication and re-identification

In this section, considering the differences in color structure, angle, and similarity of the provided dataset images, different image similarity recognition methods may have different effects on different images. Therefore, we adopted multiple image similarity recognition methods, including but not limited to: MSE mean square error, PSNR peak signal-to-noise ratio, SSIM structural similarity, cosine similarity, HASH similarity (including PHash, AHash, and DHash), and clustering methods (VLAD). Below, I will introduce the pros and cons of our analysis one by one.

3.3.1 MSE mean square error

Prediction of MSE Calculation Model $\hat{Y}$ The closeness to the real label Y. The formula is expressed as:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y_i})^2 \qquad (20)$$

Where n represents the number of samples, The advantage of this method is that the MSE： function curve is smooth, continuous, and differentiable everywhere, making it easy to use the gradient descent algorithm. It is a commonly used loss function. Moreover, as the error decreases, the gradient also decreases, which is conducive to convergence. Even with a fixed learning rate, it can quickly converge to the minimum value. But at the same time, when the difference between the true value y and the predicted value f (x) is greater than 1, the error will be amplified; When the difference is less than 1, the error will be reduced, which is determined by the square operation. MSE will impose larger penalties for larger errors (>1), while smaller errors (<1) will receive smaller penalties. That is to say, it is more sensitive to outliers and is greatly affected by them.

3.3.2 PSNR peak signal-to-noise ratio

PSNR (Peak Signal to Noise Ratio) is an objective standard for evaluating images., There are many application scenarios. It has locality, and PSNR is the abbreviation for "Peak Signal to Noise Ratio". The Chinese meaning of 'peak' is' vertex '. And ratio means ratio or ratio. The entire meaning is to reach the peak signal of the noise ratio, and PSNR is generally used for an engineering project between the maximum signal and background noise. Usually, after image compression, the output image will differ to some extent from the original image. In order to measure the quality of processed images, the PSNR value is usually used to measure whether a certain processing program is satisfactory. It is the logarithmic value of the mean square error between the original image and the processed image relative to (2n-1) 2 (the square of the maximum signal value, where n is the number of bits per sampled value), and its unit is dB.

PSNR is obtained through MSE, and the formula is as follows:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \quad (21)$$

Among which, MAXI is the maximum value representing the color of image points. If each sampling point is represented by 8 bits, it is 255.

So the smaller the MSE, the larger the PSNR; So the larger the PSNR, the better the image quality.

PSNR is the most common and widely used objective evaluation indicator for images, but it is based on the error between corresponding pixel points, which is an error sensitive image quality evaluation. Due to the lack of consideration for the visual characteristics of the human eye (the human eye is more sensitive to contrast differences with lower spatial frequencies, the human eye is more sensitive to brightness contrast differences than chromaticity, and the perception of an area by the human eye is influenced by its surrounding neighboring areas, etc.), there are often situations where the evaluation results are inconsistent with the subjective perception of the human eye.

But based on our theoretical analysis of the raw data, we do not believe that this would be a suitable and efficient solution, so we did not use it.

3.3.3 SSIM structural similarity

SSIM (structural similarity) is an indicator that measures the similarity between two images [11]. The SSIM algorithm is mainly used to detect the similarity between two images of the same size, or to detect the degree of image distortion. The SSIM formula is based on three comparative measures between samples x and y: brightness, contrast, and structure.

$$S(x,y) = f(I(x,y), c(x,y), s(x,y)) \quad (22)$$

Brightness: Using the average grayscale of the image as the estimation value, the brightness contrast function I (x, y) is a function of μ_x， μ_y:

$$I(x,y) = \frac{2\mu_x\mu_y + C1}{{\mu_x}^2 + {\mu_y}^2 + C1}, \mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \quad (23)$$

Special, we choose $C_1 = (K_1L)^2$, where L is the grayscale level of the image.。

Contrast: The standard deviation of the image is used as the estimation value, and the contrast function is:

$$c(x,y) = \frac{2\sigma_x\sigma_y + C2}{{\sigma_x}^2 + {\sigma_y}^2 + C2} \quad (24)$$

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}} \quad (25)$$

Structure: The image is divided by its own standard deviation, and the structure comparison function is:

$$s(x,y) = \frac{\sigma_{xy} + C3}{\sigma_x\sigma_y + C3} \quad (26)$$

Among which:

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{n}(x_i - \mu_x)(y_i - \mu_y) \quad (27)$$

SSIM General Equation:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C1)(2\sigma_{xy} + C2)}{({\mu_x}^2 + {\mu_y}^2 + C1)({\sigma_x}^2 + {\sigma_y}^2 + C2)} \quad (28)$$

In practical applications, sliding windows can be used to block images. Considering the influence of window shape on the segmentation, Gaussian weighting can be used to calculate the mean, variance, and covariance of each window C, and then calculate the structural similarity SSIM of the corresponding block. Finally, the mean value is used as the structural similarity measure of the two images, namely the average structural similarity SSIM.

3.3.4 Cosine similarity

Represent an image as a vector and measure the similarity between two images by calculating the cosine distance between the vectors.

Cosine similarity algorithm: The cosine value between the angles between two vectors in a vector space is used to measure the difference between two individuals. When the cosine value approaches 1 and the angle approaches 0, it indicates that the more similar the two vectors are, the closer the cosine value approaches 0 and the angle approaches 90 degrees, indicating that the two vectors are less similar.

$$\cos(\theta) = \frac{a^2 + b^2 - c^2}{2\text{ab}} = \frac{x_1^2 + y_1^2 + x_2^2 + y_2^2 - (x_2 - x_1)^2 - (y_2 - y_1)^2}{2\sqrt{x_1^2 + y_1^2} * \sqrt{x_2^2 + y_2^2}} = \frac{x_1 * x_2 + y_1 * y_2}{\sqrt{x_1^2 + y_1^2} * \sqrt{x_2^2 + y_2^2}}\psi \quad (29)$$

3.3.5 HASH similarity

There are three hash algorithms for image similarity comparison: mean hash algorithm (AHash), difference hash algorithm (DHash), and perceptual hash algorithm (PHash).

AHash: Average hash. The speed is relatively fast, but often not very accurate.

PHash: Perceived hash. The accuracy is relatively high, but the speed is relatively poor.

DHash: Differential value hash. High accuracy and very fast speed.

Hashing is not calculated in a strict way, but rather in a more relative way, because similarity is a relative judgment. The value hash algorithm, difference hash algorithm, and perceptual hash algorithm all have values ranging from 0 to 64, indicating how different the 64 bit hash values are in the Hamming distance. The values of three histograms and single channel histograms are 0-1, and the larger the value, the higher the similarity.

A picture is a two-dimensional signal that contains components of different frequencies. The areas with small changes in brightness are low-frequency components that describe a wide range of information. The areas with significant changes in brightness (such as the edges of objects) are high-frequency components that describe specific details. Alternatively, high-frequency can provide detailed information about images, while low-frequency can provide a framework. A large and detailed image has a high frequency, while small images lack image details, so they are all low-frequency. So our usual downsampling, which is the process of reducing images, is actually a process of losing high-frequency information. The mean hash algorithm utilizes the low-frequency information of an image.

The perceptual hash algorithm is a more robust algorithm than the mean hash algorithm, which differs from the mean hash algorithm in that the perceptual hash algorithm obtains low-frequency information of images through DCT (Discrete Cosine Transform).

Discrete Cosine Transform (DCT) is an image compression algorithm that transforms an image from the pixel domain to the frequency domain. Then, general images have a lot of redundancy and correlation, so after converting to the frequency domain, only a small portion of the frequency components have coefficients that are not 0, and most of the coefficients are 0 (or close to 0). The frequency of the coefficient matrix after DCT transformation increases from the top left corner to the bottom right corner, so the energy of the image is mainly retained in the low-frequency coefficients in the top left corner.

3.3.6 VLAD algorithm

The VLAD algorithm is an improved version of the BOW algorithm.

Applying the BOW model to the image field, that is, treating the image as a set of local features independent of position. Local features are equivalent to words in text, called "visual words", and the set of visual words is called "visual dictionaries" [13]. The basic process of image retrieval is:

Feature extraction (SIFT algorithm), learning the "visual vocabulary" (k-means algorithm), quantifying the input feature set based on the visual dictionary, converting the input image into a frequency histogram of visual words, constructing an inverted table of features to the image, quickly indexing relevant images through the inverted table, and performing histogram matching based on the indexing results.

The difference from the BOW algorithm is that BOW calculates the number of SIFT feature subsets to which each cluster center belongs, and encodes them based on the number of subsets they belong to. The VLAD algorithm calculates and encodes the distance between the SIFT feature sub and the cluster center to which each cluster center belongs. The feature vectors encoded by the VLAD algorithm not only contain the quantity information of feature vectors, but also the distance information from feature vectors to the cluster center. Reflects more information than the BOW algorithm. A simple example is shown in the following figure.

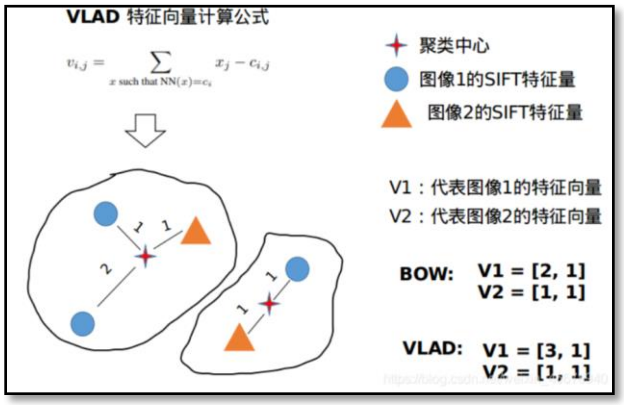


Figure 6. A simple example of VLAD algorithm.

The following uses OpenCV combined with KMeans clustering in sklearn to implement the VLAD algorithm. Clustering belongs to unsupervised learning [9], and K-means clustering is the most basic and commonly used clustering algorithm. Its basic idea is to iteratively find a partitioning scheme for K clusters, so as to minimize the loss function corresponding to the clustering results. Among them, the loss function can be defined as the sum of the squares of the errors between each sample and the center point of the cluster it belongs to:

$$J(c,\mu) = \sum_{i=1}^{M} ||x_i - \mu_{c_i}||^2 \qquad (30)$$

Where x_i represents the i-th sample, and c_i is the cluster to which x_i belongs, μ_{c_i}represents the center point corresponding to the cluster, and M is the total number of samples.

The core goal of KMeans is to divide a given dataset into K clusters (where K is a hyperparameter) and provide the corresponding center point for each sample data. Specific steps [12]:

1. Data preprocessing
2. Randomly select K centers, denoted as $\mu_1^{(0)}, \mu_2^{(0)}, \dots, \mu_k^{(0)}$
3. Define the loss function:

$$J(c,\mu) = min \sum_{i=1}^{M} ||x_i - \mu_{c_i}||^2 \qquad (31)$$

4. Let t=0,1,2 To determine the number of iteration steps, repeat the following process until J converges:

Assign each sample x_i to the nearest center:

$$c_i^t < -argmin_k ||x_i - \mu_k^t||^2 \qquad (32)$$

For each class center k, recalculate the center of that class:

$$\mu_k^{(t+1)} <\!\!- argmin_\mu \sum_{i:c_i^t=k}^{b} ||x_i - \mu||^2 \qquad (33)$$

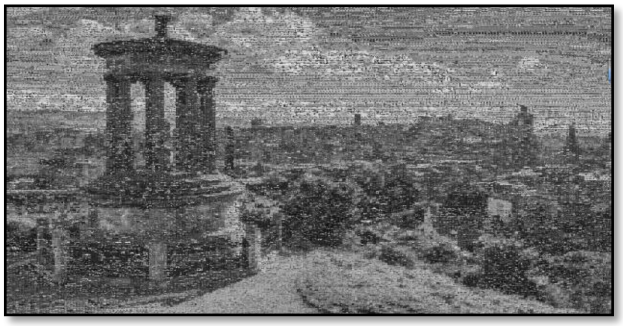
The core part of KMeans is to first fix the center point, adjust the category of each sample to reduce j, then fix the category of each sample, and adjust the center point to continue reducing j. The two processes alternate and cycle, with j monotonically decreasing until the minimum (minimum) value, and the center point and sample classification converge simultaneously.

4. Experimental Results

In this section, we will show our experimental results and the comparison between them in the following parts.

4.1. PART A: Composite image generation

In part A, to validate our software works well, we first use the grayscale images to composite the target image in different resolution in which we use 40,000 source images in size (6,9) and use 160,000 source images in size (8,12) to composite the target image 1 and target image 2, respectively. And the composite image 1 has 200 resized source images in each column as well as in each row, while the composite image 2 has 400 resized source images in each column as well as in each row. The results are shown in Figure 7.



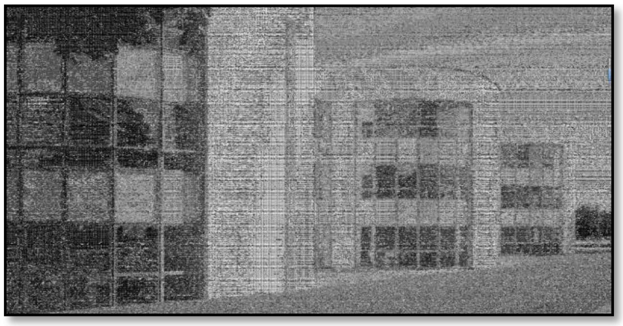


Figure 7. Experimental results of composite images in grayscale. The top figure is composite image 1, and the bottom figure is composite image 2.

From Figure 7, we can validate our design of software is feasible. And in the next step, we will try to process the RGB images according to this thought.

As the RGB color image has R, G, B three channels, we need to make some simple changes to our software in which we calculate the histogram of each channel of the images separately, and in the similar way to compare the similarity of each channel between each image. And to get better composite results, we use three different resolutions to composite each target image. For target image 1, which is in size (1200,1800), we use 40,000 source images in size (6,9), 240,000 source images in size (3,3), and 360,000 source images in size (2,3) to composite it, respectively. And for target image 2, which is in size (1600,2400), we use 40,000 source images in size (8,12), 160,000 source images in size (4,6), and 640,000 source images in size (2,3) to composite it, respectively. The results of the composite image 1 and composite image 2 are shown in Figure 8 and Figure 9, respectively.

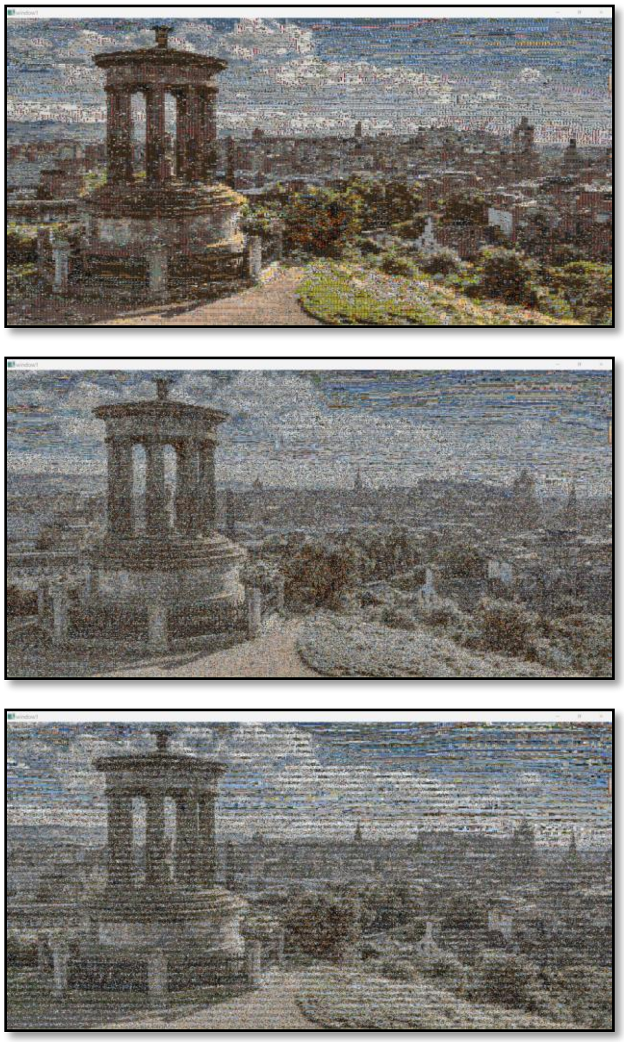


Figure 8. Experimental results of RGB composite image 1. The figures are using 40,000 source images in size (6,9), 240,000 source images in size (3,3), and 360,000 source images in size (2,3) to carry out composition, respectively.

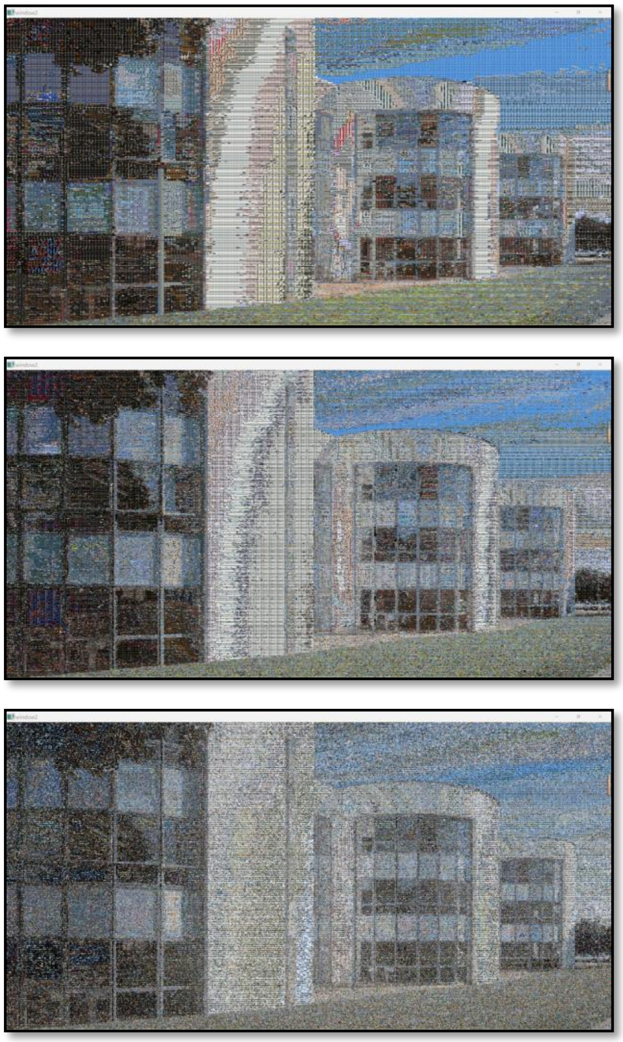


Figure 9. Experimental results of RGB composite image 2. The figures are using 40,000 source images in size (8,12), 160,000 source images in size (4,6), and 640,000 source images in size (2,3) to carry out composition, respectively.

From Figure 8 and Figure 9, we can prove that we get good RGB composite results by using our design of software. And it is clearly to find that as the resolution increases, the result of the composition of the feature in target image becomes better, such as the building in those images, however, the result of the composition of the color in target image becomes worse, it is more like a grayscale image. In addition, we also can find that the transitions between the tiles of the composite image are not very smooth, especially in low-resolution composite images.

Therefore, in the next improvement, we will first use the smoothing method to remove the noise in the source images before resize (downsample) them. In this case, we choose to use Gaussian smoothing method because of its good performance in balancing denoising as well as preserving image detail. Thus, in our software, we first carry out Gaussian smoothing to remove the noise in the source images and then downsample to resize them into tiles. Then, composite them as before, and finally carry out Gaussian smoothing to the composite image to make the result looks smoother. And we also use three different resolutions to composite each target image in this step which is the same as the last step. For target image 1, use 40,000 source images in size (6,9), 240,000 source images in size (3,3), and 360,000 source images in size (2,3) to composite it, respectively. And for target image 2, use 40,000 source images in size (8,12), 160,000 source images in size (4,6), and 640,000 source images in size (2,3) to composite it, respectively. The results of the composite image 1 and composite image 2 with using Gaussian smoothing are shown in Figure 10 and Figure 11, respectively.

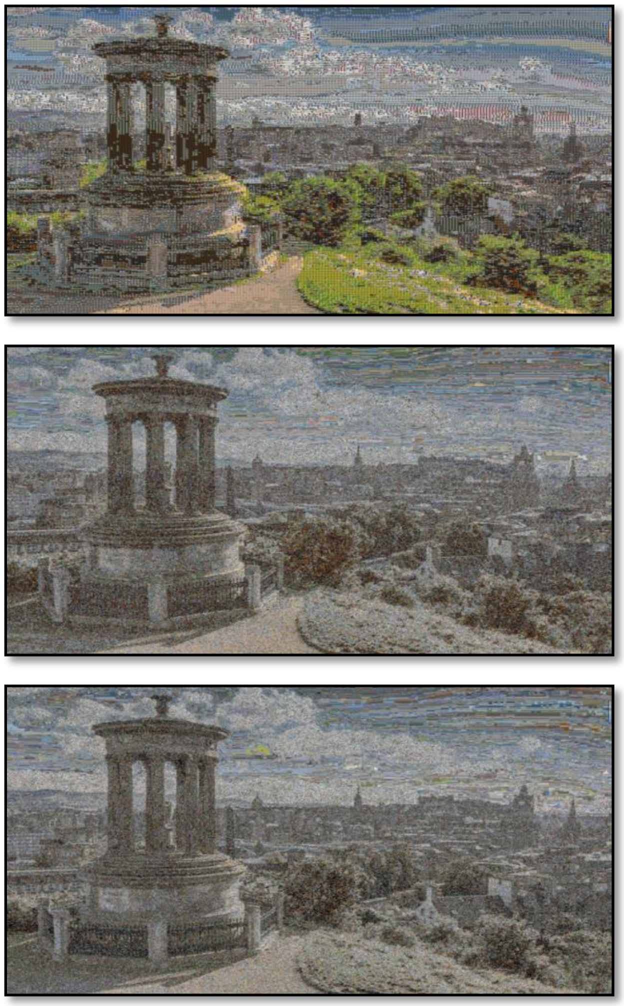


Figure 10. Experimental results of RGB composite image 1 with using Gaussian smoothing. The figures are using 40,000 source images in size (6,9), 240,000 source images in size (3,3), and 360,000 source images in size (2,3) to carry out composition, respectively.

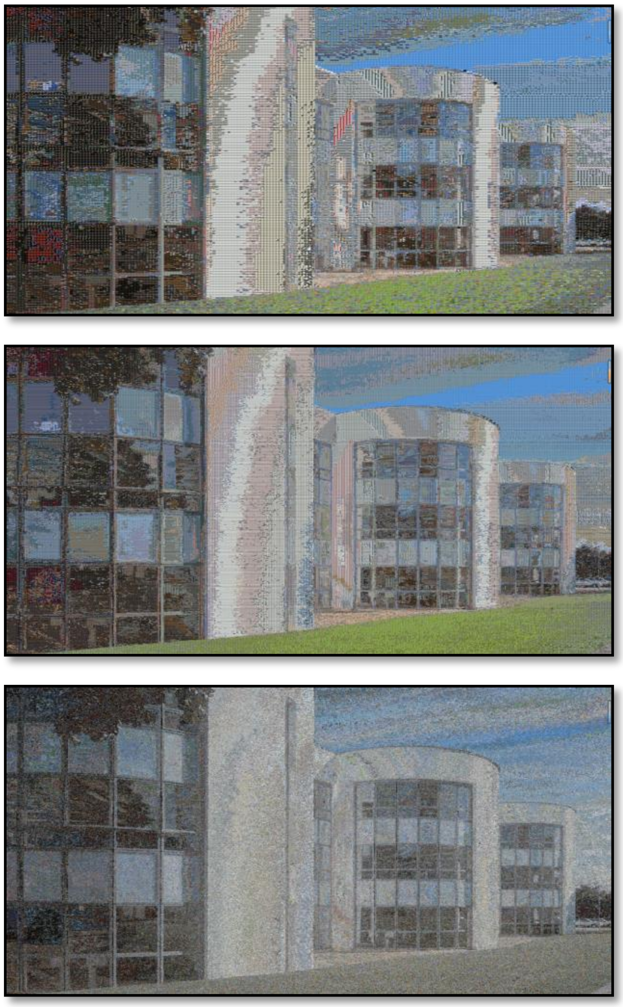


Figure 11. Experimental results of RGB composite image 2 with using Gaussian smoothing. The figures are using 40,000 source images in size (8,12), 160,000 source images in size (4,6), and 640,000 source images in size (2,3) to carry out composition, respectively.

From Figure 10 and Figure 11, we can find that the results are better than those without using Gaussian smoothing as the coherences between different tiles are smoother. And from the results in different resolutions, it also illustrates that as the resolution increases, the result of the composition of the feature in target image becomes better, but the color in composite image becomes worse. This may because when we reduce the size of each tile, it may result in color averaging, which means a decrease in the number of pixels contained in each tile. When calculating the histogram for each tile, the reduction in pixel count may weaken the peaks of the histogram, making the color distribution more uniform. As a result, the color of each tile becomes closer to the average color of the entire image, making it appear more similar to a grayscale image. Reducing the size of tiles may also lead to the loss of color information, meaning smaller tiles might not capture the color variations present in larger tiles, resulting in the final composite image losing some details and color depth.

From all of the results above, by considering the tradeoff between the performance of the degree of image details restoration and the degree of image color restoration, for targe image 1, the relatively best result is that uses 40,000 source images in size (6,9) to composite the image (the first picture in Figure 10), and for targe image 2, the relatively best one is that uses 40,000 source images in size (8,12) to composite the image (the first picture in Figure 11).

Additionally, to solve the color averaging and the loss of color information problems, we try to calculate the histograms of the original source images first. With this improvement, the results look better for high-resolution composite images, which are shown in Figure 12 and Figure 13. For the purpose of showing only the performance improvements, we will only show the results of using 240,000 source images in size (3,3) to composite target image 1, and using 160,000 source images in size (4,6) to composite target image 2.

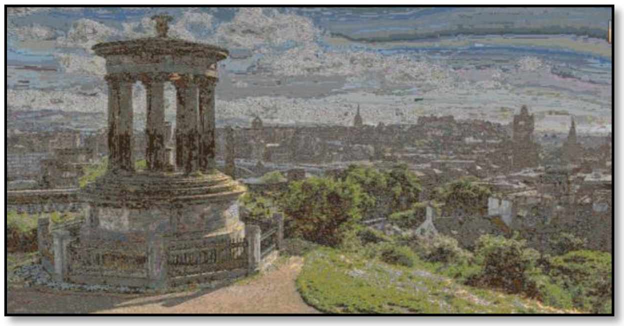


Figure 12. Experimental results of RGB composite image 1 with using Gaussian smoothing after improvement, which is composited from 240,000 source images in size (3,3).

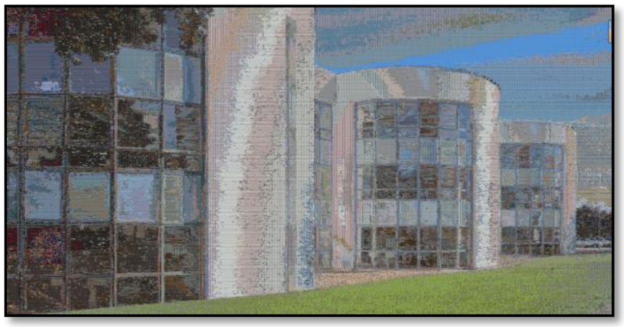


Figure 13. Experimental results of RGB composite image 2 with using Gaussian smoothing after improvement, which is composited from 160,000 source images in size (4,6).

4.2. PART B: Binary image classification

In part B, we compare the results of the various combinations in terms of computation time, misclassification rates for each class, and overall accuracy. In the processing of extracting feature vectors using SIFT, we have found that the dimension of feature vector is different from each other, which does not provide a consistent feature vector for the classifier. In order to solve this problem, PCA (Principal Component Analysis) is used to reduce the dimension of feature vector to 128×1 dimension.

Principal Component Analysis (PCA) is a widely used technique in machine learning and image processing for dimensionality reduction. PCA is employed to reduce the number of features or dimensions in a dataset while retaining its essential information. It achieves this by transforming the data into a new coordinate system, where the first few principal components capture the maximum variance in the data.

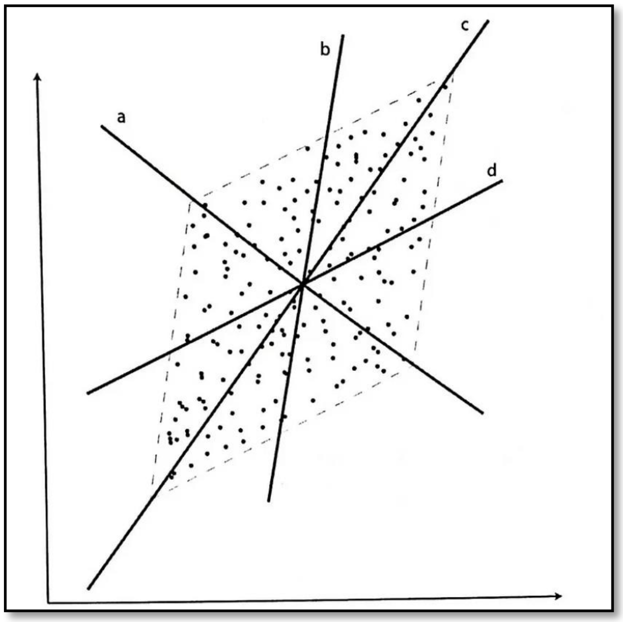


Figure 14. A simple example of PCA algorithm

First, the mentioned above three methods are used to extract feature vector of the following image.

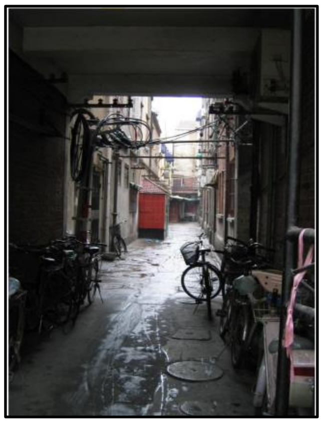
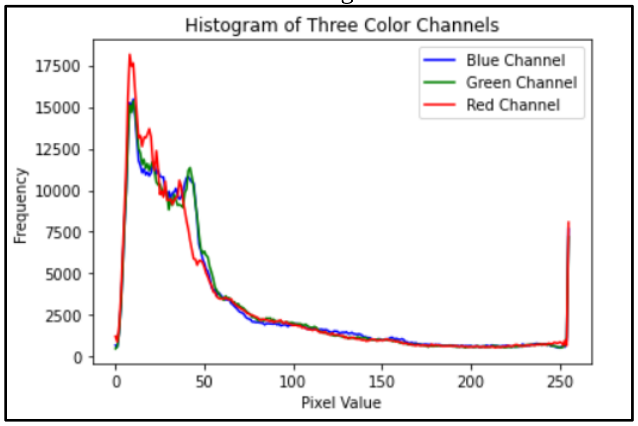
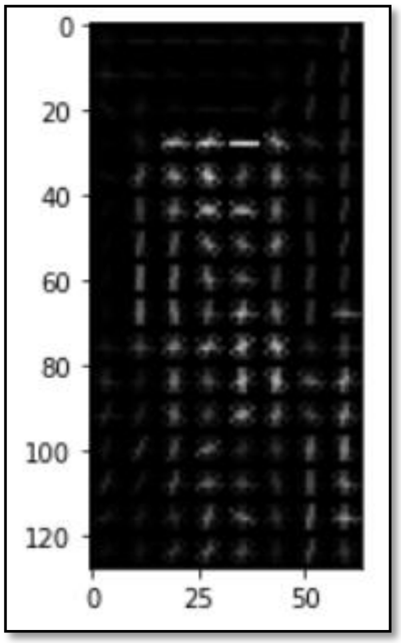


Figure 15. A simple image

The feature vector of the image is shown below:





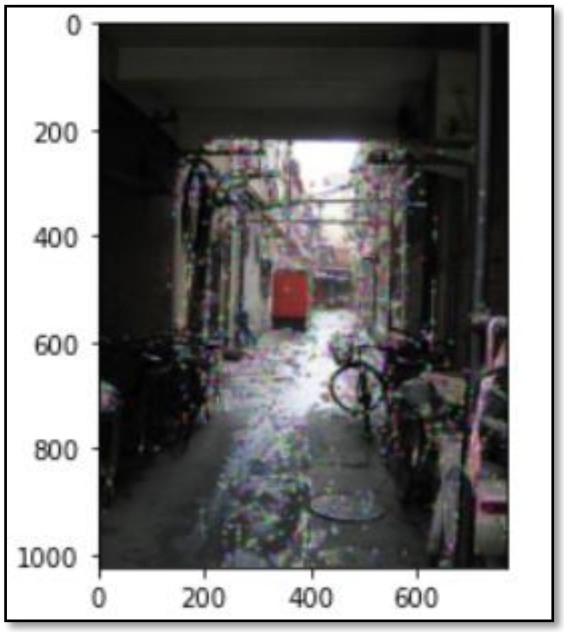


Figure 16. feature vector image extracted by three methods

It is obvious that nothing can be got from the feature vector image.

The following figure are the results of the various combinations in terms of computation time, misclassification rates for each class, and overall accuracy.

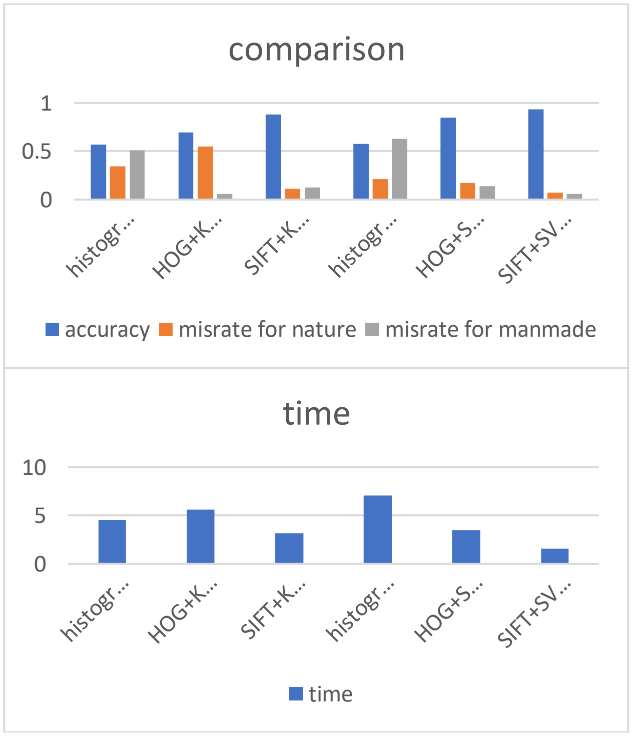


Figure 17. comparison

From the above figure, what can be known is that for the same method used to extract feature vector, the accuracy of SVM classifier is higher than that of KNN. The reason behind this is that SVM has better generalization ability than KNN. SVM can find the decision boundary and maximize the classification interval, which makes it have strong generalization ability to the unseen data. KNN relies on samples of local neighborhoods and may be more sensitive to new data. What's more, SVM is somewhat robust to noisy data because it focuses primarily on support vectors and is not affected by non-support vectors. KNN is sensitive to noise and may be affected by outliers.

Another finding is that for the sane classifier, the accuracy of SIFT is higher than those of HOG and histogram. The reason behind this is that feature vectors generated by SIFT have higher dimensions and can describe local features in the image more comprehensively, including position, scale, direction and other information.

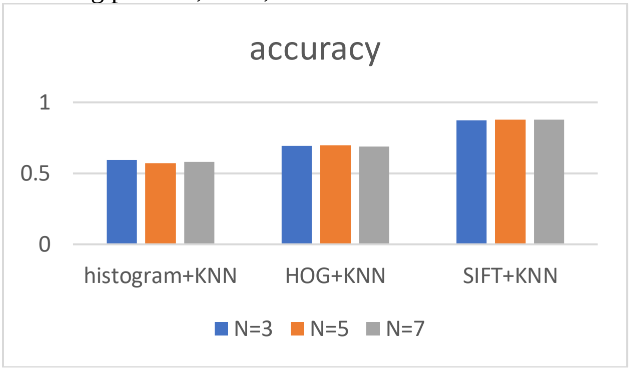


Figure 18. accuracy with different neighbors

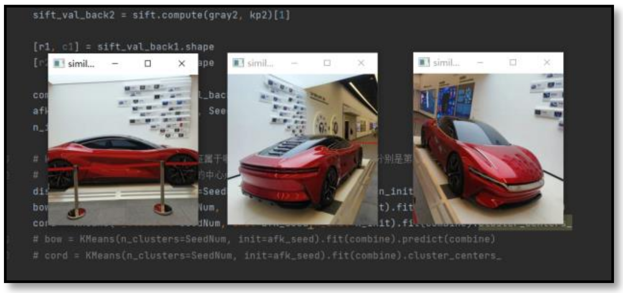
Among the different neighbors, the accuracy of HOG and SIFT is the highest when the number of neighbors is 5. However, the accuracy of histogram is the lowest when the number of neighbors is 5.

4.3. PART C: Deduplication and re-identification

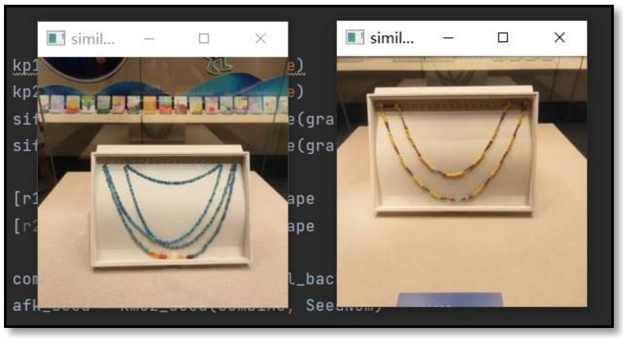
Our specific approach to identifying duplicate and similar images is to first place all images in the FIFO queue, create a two-dimensional list, place the first image in the first list, and then pop out the elements in the queue one by one to compare them with the elements in the sub list. The method for detecting similarity has been mentioned above. If it is determined to be similar, the image will be added to the sub list. If it is determined to be not similar, the image will be used as an element to create a new list. However, considering that the number of images in the sub list may not be 1, when this situation occurs, we will choose the average of the comparison results. When the judgment result is that the two images are exactly the same, we attempt to use the Unreferenced Image Spatial Quality Evaluator (BRISQUE) [10]. However, after taking our own images and testing, we found that the effect is not ideal. Therefore, in order to simplify the calculation and ensure processing speed, we simply remove two identical images simultaneously to achieve the effect of removing the same image.

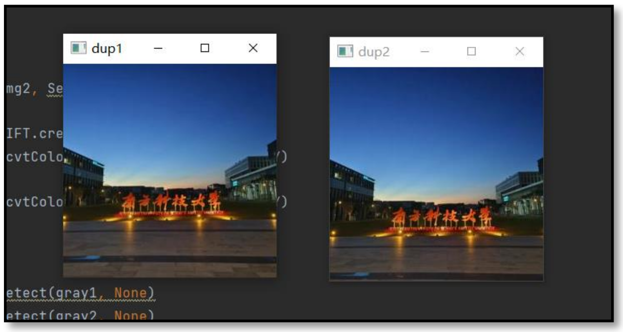
At the same time, we need to point out that 07.jpg and 04.jpg have high similarity in color distribution, image features, and other aspects. Although they were not captured in the same scene, after processing the similarity detection algorithm mentioned above, they both obtained similar results. Therefore, we believe that the two images are relatively similar.

The following is the code run result: we will indicate similarity and similarity on the framework.

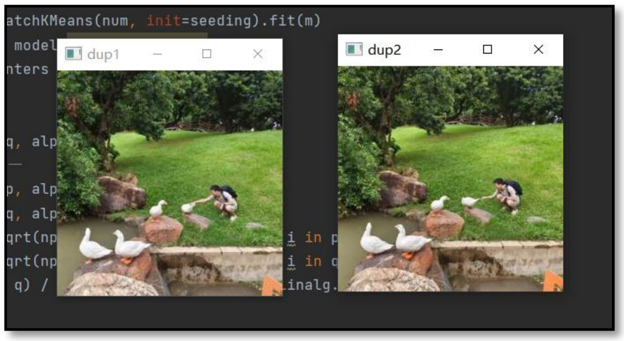


```
sift_val_back2 = sift.compute(gray2, kp2)[1]

[r1, c1] = sift_val_back1.shape

# bow = KMeans(n_clusters=SeedNum, init=afk_seed).fit(combine).predict(combine)
# cord = KMeans(n_clusters=SeedNum, init=afk_seed).fit(combine).cluster_centers_
```





```
etect(gray1, None)
```



```
atchKMeans(num, init=seeding).fit(m)
 model
nters


q, alp

p, alp
q, alp
qrt(np
qrt(np
 q) /
```

Figure 19. Part C code run results.

At the same time, we also encountered quite a few problems when processing the code, including the inability to accurately determine whether the two images are similar, whether the threshold setting is reasonable, whether the k value in the k-mean is appropriate, and when inputting the same image, due to the Gaussian distribution of the image in the coordinate system, the output similarity is also a random distribution, so we had to process multiple times to calculate the average value.

In this experiment, we conducted various attempts, and the following are the results of testing different similarity detection methods: we take the following two pairs of images as examples (01.jpg 03.jpg and 04.jpg 07.jpg):

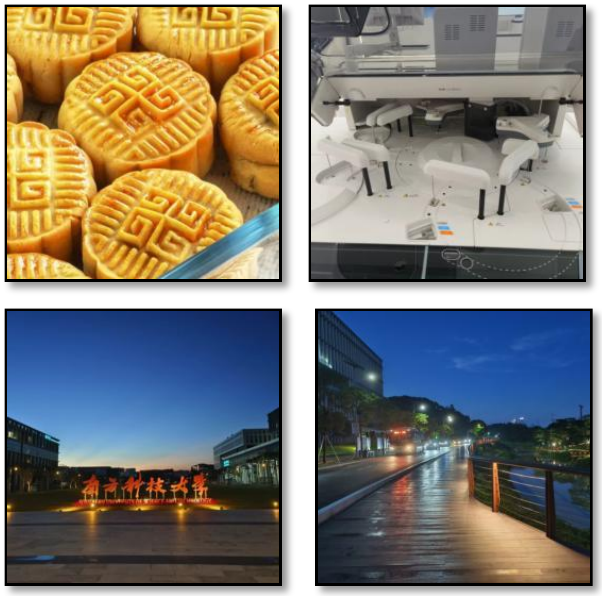
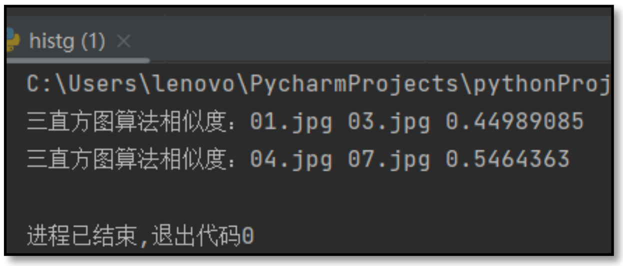


Figure 20. Results of testing different similarity detection methods (We take the following two pairs of images as examples(01.jpg 03.jpg and 04.jpg 07.jpg)).

Without specific explanation, the larger the output value, the more similar the two images will be:

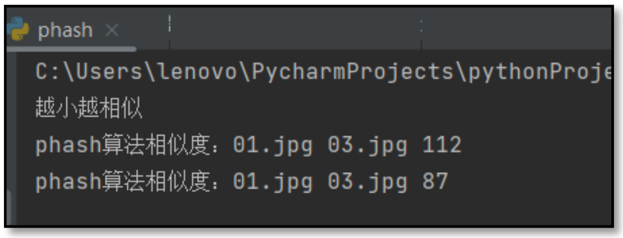
Three Histogram Algorithm:



```
C:\Users\lenovo\PycharmProjects\pythonProj
三直方图算法相似度：01.jpg 03.jpg 0.44989085
三直方图算法相似度：04.jpg 07.jpg 0.5464363

进程已结束,退出代码0
```

Figure 21. Results of Three Histogram Algorithm.

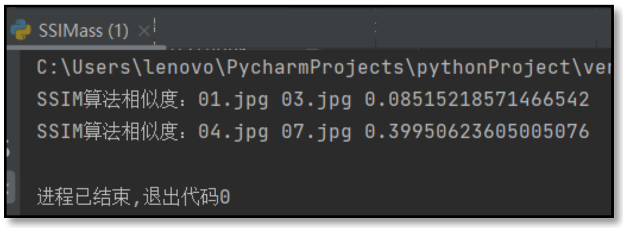
Phash Algorithm:



```
C:\Users\lenovo\PycharmProjects\pythonProje
越小越相似
phash算法相似度：01.jpg 03.jpg 112
phash算法相似度：01.jpg 03.jpg 87
```

Figure 22. Results of Phash Algorithm.

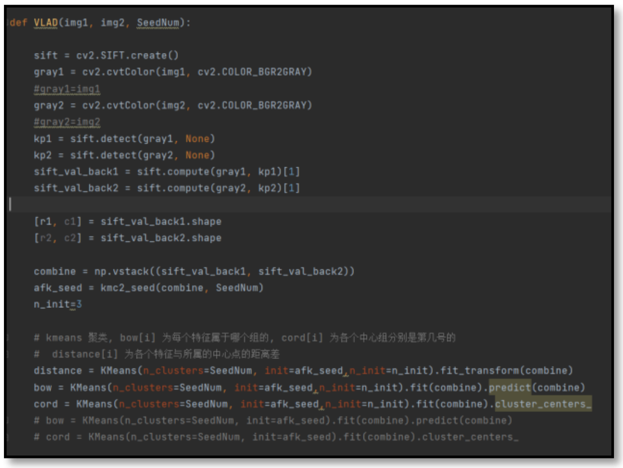
SSIM Algorithm:



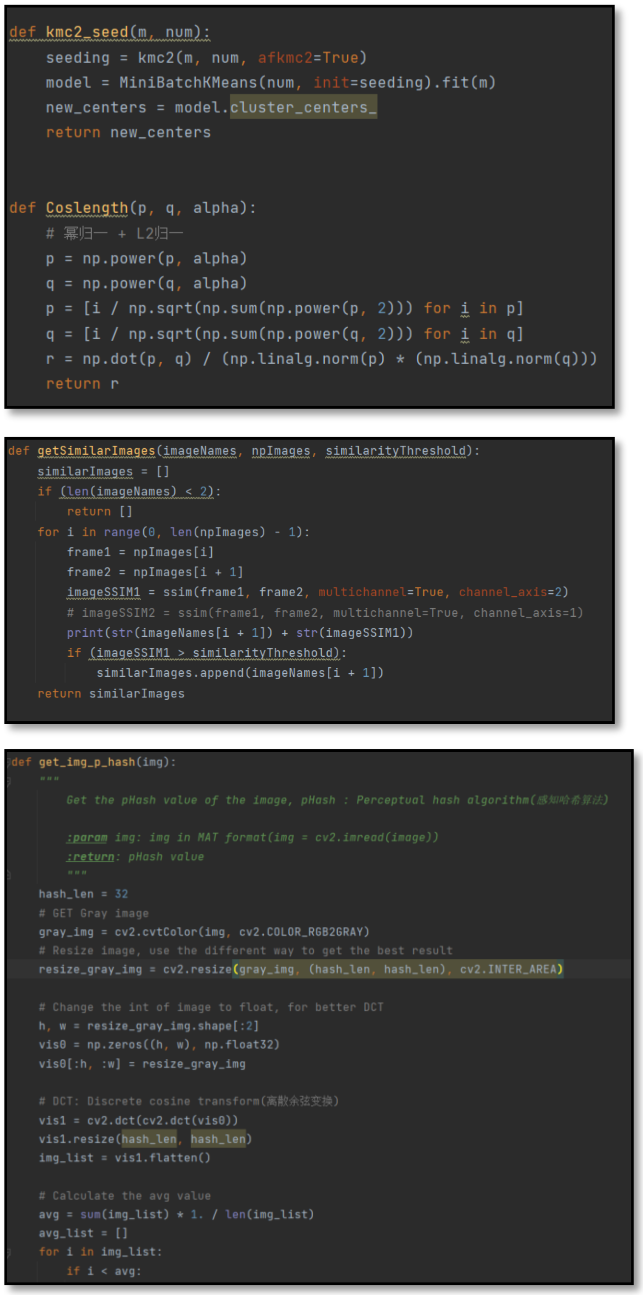
```
C:\Users\lenovo\PycharmProjects\pythonProject\ve
SSIM算法相似度：01.jpg 03.jpg 0.08515218571466542
SSIM算法相似度：04.jpg 07.jpg 0.39950623605005076

进程已结束,退出代码0
```

Figure 23. Results of SSIM Algorithm.

Here are some screenshots of the code:



```
def VLAD(img1, img2, SeedNum):

    sift = cv2.SIFT.create()
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    #gray1=img1
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    #gray2=img2
    kp1 = sift.detect(gray1, None)
    kp2 = sift.detect(gray2, None)
    sift_val_back1 = sift.compute(gray1, kp1)[1]
    sift_val_back2 = sift.compute(gray2, kp2)[1]

    [r1, c1] = sift_val_back1.shape
    [r2, c2] = sift_val_back2.shape

    combine = np.vstack((sift_val_back1, sift_val_back2))
    afk_seed = kmc2_seed(combine, SeedNum)
    n_init=3

    # kmeans 聚类, bow[i] 为每个特征属于哪个组的, cord[i] 为各个中心组分别是第几号的
    #  distance[i] 为各个特征与所属的中心点的距离差
    distance = KMeans(n_clusters=SeedNum, init=afk_seed,n_init=n_init).fit_transform(combine)
    bow = KMeans(n_clusters=SeedNum, init=afk_seed,n_init=n_init).fit(combine).predict(combine)
    cord = KMeans(n_clusters=SeedNum, init=afk_seed,n_init=n_init).fit(combine).cluster_centers_
    # bow = KMeans(n_clusters=SeedNum, init=afk_seed).fit(combine).predict(combine)
    # cord = KMeans(n_clusters=SeedNum, init=afk_seed).fit(combine).cluster_centers_
```



```
def kmc2_seed(m, num):
    seeding = kmc2(m, num, afkmc2=True)
    model = MiniBatchKMeans(num, init=seeding).fit(m)
    new_centers = model.cluster_centers_
    return new_centers


def Coslength(p, q, alpha):
    # 幂归一 + L2归一
    p = np.power(p, alpha)
    q = np.power(q, alpha)
    p = [i / np.sqrt(np.sum(np.power(p, 2))) for i in p]
    q = [i / np.sqrt(np.sum(np.power(q, 2))) for i in q]
    r = np.dot(p, q) / (np.linalg.norm(p) * (np.linalg.norm(q)))
    return r
```

```
def getSimilarImages(imageNames, npImages, similarityThreshold):
    similarImages = []
    if (len(imageNames) < 2):
        return []
    for i in range(0, len(npImages) - 1):
        frame1 = npImages[i]
        frame2 = npImages[i + 1]
        imageSSIM1 = ssim(frame1, frame2, multichannel=True, channel_axis=2)
        # imageSSIM2 = ssim(frame1, frame2, multichannel=True, channel_axis=1)
        print(str(imageNames[i + 1]) + str(imageSSIM1))
        if (imageSSIM1 > similarityThreshold):
            similarImages.append(imageNames[i + 1])
    return similarImages
```

```
def get_img_p_hash(img):
    """
        Get the pHash value of the image, pHash : Perceptual hash algorithm(感知哈希算法)

        :param img: img in MAT format(img = cv2.imread(image))
        :return: pHash value
        """
    hash_len = 32
    # GET Gray image
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    # Resize image, use the different way to get the best result
    resize_gray_img = cv2.resize(gray_img, (hash_len, hash_len), cv2.INTER_AREA)

    # Change the int of image to float, for better DCT
    h, w = resize_gray_img.shape[:2]
    vis0 = np.zeros((h, w), np.float32)
    vis0[:h, :w] = resize_gray_img

    # DCT: Discrete cosine transform(离散余弦变换)
    vis1 = cv2.dct(cv2.dct(vis0))
    vis1.resize(hash_len, hash_len)
    img_list = vis1.flatten()

    # Calculate the avg value
    avg = sum(img_list) * 1. / len(img_list)
    avg_list = []
    for i in img_list:
        if i < avg:
```

Figure 24. Some screenshots of the code in part C.

5. Conclusion

In this assignment, we implement the software which can be used to composite the target images of various types, execute binary image classification, and also carry out deduplicating nearly identical pictures and picking out similar pictures. From our experimental results, which can prove that we have successfully designed the software as well as implementing good performance for our tasks.

Acknowledgement

We would like to express our gratitude to Professor Jianguo Zhang for delivering the Computer Vision content in CS303B. Besides, we appreciate the companionship and efforts of the three teaching assistants during the first half of this semester. Moreover, we extend our thanks to the Department of Computer Science and Engineering for offering the CS303B course.

References

[1] Shuiyixin. 2018. Principle and usage of cv2.compareHist function in OpenCV. *https://blog.csdn.net/shuiyixin/article/details/80257822*.

[2] Krystian Safjan. 2020. Metrics Used to Compare Histograms. *https://safjan.com/metrics-to-compare-histograms/*.

[3] Krystian Safjan. 2023. Understanding Bhattacharyya Distance and Coefficient for Probability Distributions. *https://safjan.com/understanding-bhattacharyya-distance-and-coefficient-for-probability-distributions/*.

[4] Krystian Safjan. 2023. Histogram Intersection. *https://safjan.com/histogram-intersection/*.

[5] Mrinal Tyagi. HOG (Histogram of Oriented Gradients): An Overview. https://towardsdatascience.com/hog-histogram-of-oriented-gradients-67ecd887675f, 2023.

[6] OpenCV. Introduction to SIFT (Scale-Invariant Feature Transform). https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html,2023.

[7] JavaPoint. K-Nearest Neighbor (KNN) Algorithm for Machine Learning. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning, 2023.

[8] Sunil Ray. Learn How to Use Support Vector Machines (SVM) for Data Science. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/,2023.

[9] Obachem. 2017. kmc2. *GitHub - obachem/kmc2: Cython implementation of k-MC2 and AFK-MC2 seeding*.

[10] M_gn. 2020. Aggregating local descriptors into a compact image representation. *https://blog.csdn.net/weixin_45678940/article/details/106326239*.

[11] m0_61899108. 2023. 计算两幅图像的相似度 (PSNR、SSIM、MSE、余弦相似度、MD5、直方图、互信息、Hash) & 代码实现 与举例. *blog.csdn.net/m0_61899108/article/details/127715737*.

[12] Giant. 2020. KMeans聚类算法详解. *https://zhuanlan.zhihu.com/p/184686598*.

[13] Eating Lee. 2019. 基于BOW模型的图像检索. *https://blog.csdn.net/qq_40369926/article/details/90115483*.