CS303B Assignment-1

Hanqi Su 11912625   Tongshu Pang 11910431   Yuyang Zhang 11810123

Abstract

The purpose of this project is to develop, evaluate and report image processing software to automatically create various types of composite images, using automatically selected images as specific categories of scenes. We mainly implemented the two basic parts of synthetic image generation and binary image classification (natural scenery or artificial scenery) using machine learning methods, and combined the two parts to automatically determine which picture belongs to after selecting a specific picture category, and generate a composite image. In addition, we also use python to construct a neural network to achieve binary classification of pictures, and the effect is also very satisfactory.

1 Introduction

Image processing has already been used in the daily life not only for entertainment field but also for some academic research. The classical images processing is using filter to change the size of the picture. As for a human, when see a photo, we are tend to recognized the scene firstly through the color, than the shape of the objects and observe their arranging rules. Thus, the pictures' features using in computer vision, based on the human sensor system, are mainly for three parts: Color feature, Geometry features and Vein Features. According to the images' features will can finish the tasks of matching or classification. Our project focus on the matching based on the color features and classification using machine learning, KNN. In addition, we also explore the neural networks applying on images classification.

1.1 Related Works

Vein Matching When doing an image quilting job, we not only have to focus the color matching but also the veins. The vein matching can give the result picture a better performance, for it has a more natural connection between two tiles.[1]

Histogram The most basic features we can extra from the image is the color features, which first came up by Michael J. Swain and Dana H. Ballard [2]. And histogram is a direct way to present the percentage one color take in the whole picture. It using the statistic method to have the color information of the image.

Line Detection In 1962 Paul Hough firstly propose the method Hough Transform and Richard Duda and Peter Hart extended the idea in to the geometry feature detection [3].

HOG For the rotation of an object may mislead the detection of the objects, the method called Histogram of Oriented Gradien came up in 2005 [4]. Based on the gradient of the images we can describe the image outline information by HOG.

Knn & Classifier Combination A classifying algorithm based on the label of the nearest neighbor of the input data. Speaking about nearest neighbor, the nearest ones need to be determined based on the distance with the given data. Therefore, a set of features[1] needs to be extracted as the feature vector for further calculation of distance. And the method of calculating the distance[2] should also be determined. After known the distance from the given data to every other data, the number of nearest neighbor[3] that can influence the final result should be set. The number of sets of features extracted from a single image might be more than one set. Therefore, different result might be given when considering each individual feature set. Each can be treated as a weak classifier. with these classifiers combined, a stronger classifier[4] with higher accuracy can be built using the weighted voting algorithm.

After extracting features, the Knn classifier is used to classify different type of images. In our method of classification, each feature is used in one Knn classifier, which means multiple results will be produced simultaneously by multiple Knn classifiers fed with multiple type of features. We combined these results by giving each result a weight of voting and finally vote for the most supported result, in which the weight of each classified result is determined by the performance of the corresponding classifier. We used the accuracy of the classifier as the weight of voting of the result they produce. Eventually, a stronger classifier with higher accuracy than each individual classifier is produced by combining these weaker classifiers with lower accuracy. A demonstration graph is shown below to illustrate the combining process.

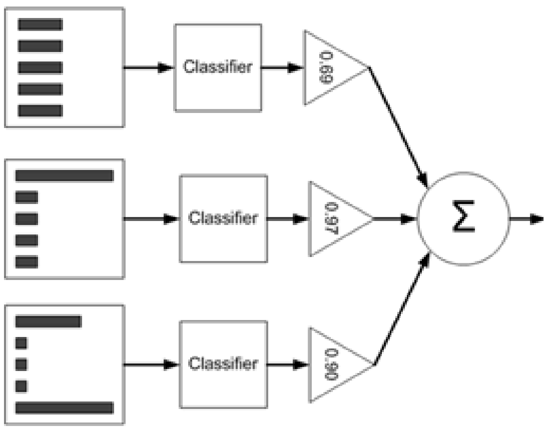


Figure 1 Combining Weak Classifiers into a Stronger One (Retrieved from: https://www.cnblogs.com/lpworkstudyspace1992/p/6668990.html)

HSV Color Space Unequally Divided Histogram Feature

By observing the images from two sets, we concluded that in the natural images, the images often contain the color of the sky, the trees or the rocks. Being sufficiently distinguished from that of the manmade scenery, the color of the images can be treated as another feature to classify images. Here we use HSV Color Space Unequally Divided Histogram. It basically converts the whole HSV color space image into a gray image by scaling each channel and conduct a summation. The method of scaling each channel is as follow.

$$G = \frac{H \times 16 + S \times 4 + V}{256} \quad (Round\ to\ integer)$$

This gray image can be further calculated into a histogram. Which classify color hue into 16 different types, saturation into 4 types and value or brightness into 4 types, which makes hue information more important and accurate in the histogram compared to the other two.

Color Coherence Vectors

This kind of method divide the 0~255 into bins (e.g., 25 bins: 1: 0~25, 2: 26~50…). Then replace the color with its bin's index, then statistics the connected colors.

$$\begin{matrix} 22 & 10 & 21 & 22 & 15 & 16 \\ 24 & 21 & 13 & 20 & 14 & 17 \\ 23 & 17 & 38 & 23 & 17 & 16 \\ 25 & 25 & 22 & 14 & 15 & 21 \\ 27 & 22 & 12 & 11 & 21 & 20 \\ 24 & 21 & 10 & 12 & 22 & 23 \end{matrix} \longrightarrow \begin{matrix} 2 & 1 & 2 & 2 & 1 & 1 \\ 2 & 2 & 1 & 2 & 1 & 1 \\ 2 & 1 & 3 & 2 & 1 & 1 \\ 2 & 2 & 2 & 1 & 1 & 2 \\ 2 & 2 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 2 & 2 \end{matrix}$$

Using characters to label connection regions

Label	*A*	*B*	*C*	*D*	*E*
Color	1	2	1	3	1
Size	12	15	3	1	5

Then find out the percentage of the connected colors and separates colors

Color	1	2	3
α	17	15	0
β	3	0	1

HOUGH This would produce the result of the line-segments in the image, including the angle of the line-segments, the length of the line-segments and the likelihood of existence of the line in the image. In the manmade scenery images, the straight lines tend to be more abundant than that of the natural images, especially longer lines. Also, the angles of these straight lines in the manmade images seems to appear more on vertical and horizontal directions than that of the natural images. Therefore, the first feature is extracted by selecting the most distinguished few line-segments and build a histogram of angles, which contains the total line length of lines lying on each angle intervals producing a 1-D histogram. The second feature can also be extracted by summing up the length of lines with enough strength, which will produce a single number.

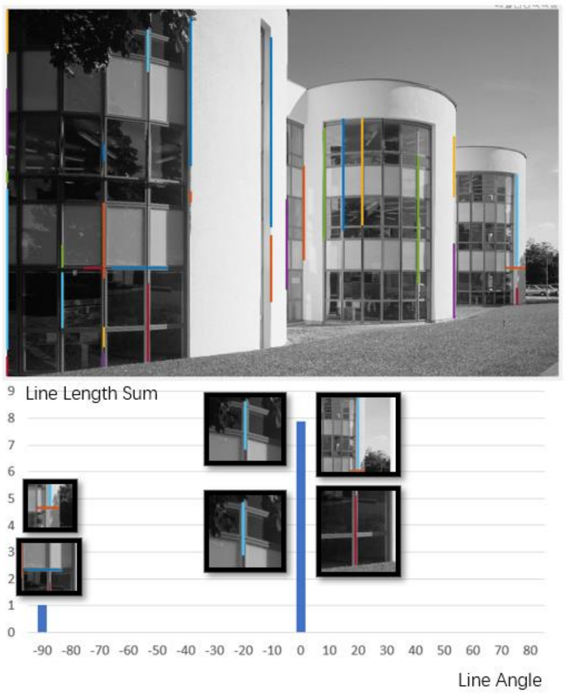


Figure 2 Hough Line Angle Histogram Demonstration

1.1.1.1. Color Moments

Color Moments is a mathematical based method to represent the color distribution. The Color moments only evaluate the mean, variance and skewness. We can see through the limitation that color moments loss the color position information, so this kind of method is normally used to narrow down the choices. The formula is shown below:

$$E_i = \frac{1}{N}\sum_{j=1}^{N} p_{ij} \ , \quad \sigma_i = (\frac{1}{N}\sum_{j=1}^{N}(p_{ij} - E_i)^2)^{\frac{1}{2}} \ \text{ and } \ s_i = (\frac{1}{N}\sum_{j=1}^{N}(p_{ij} - E_i)^3)^{\frac{1}{3}} \ .$$

Because we have 3 channels for each image, we will have a vector with 3*3=9 elements.

1.2 Innovation

The Part A of the project is to realize the Montage Picture. It is quote inefficient to compare each piece of the whole materials, for each image is in different kind of color tendency. Therefore, we are likely to divide the images in different sets (e.g., H of HSV features), than during the matching, we only need to consider smaller set of materials.

Also, for there are too much images we can get from one images, we tend to preprocessing the data by storing all relative information in struct of MATLAB, giving us a benefit to doing KNN job.

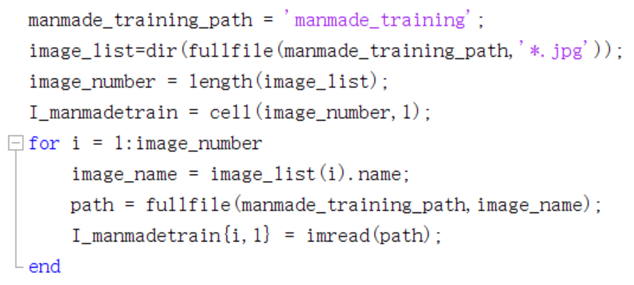
2 Method

2.1 Data Implementation

We have two methods to implement data in different tasks.

For image generation, in order to use these pictures more quickly and

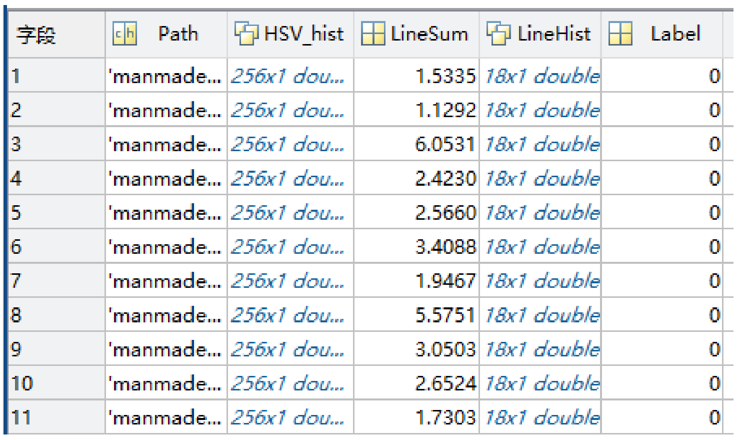
conveniently, we use MATLAB to read these pictures in advance and store them in the ‘*Datasave.mat*’ file. The code structure logic of the stored file is shown in Figure 3. (e.g., On manmade set, the idea is: we use ‘*image_list*’ to store the absolute path of each of our pictures, and then build a cell array ‘*I_manmadetrain*’ to store the pictures in it). Cell function is a data type that contains indexed data containers called cells, where each cell can contain any type of data. We can use curly braces {} for indexing to access the contents of the cell. Storing pictures in a cell array will help us call out the pictures we need more quickly, thereby improving efficiency. Finally, we store our image data into it through 4 cell arrays, called
'*I_manmadetest*','*I_manmadetrain*','*I_naturaltest*','*I_naturaltrain*'.



```
manmade_training_path = 'manmade_training';
image_list=dir(fullfile(manmade_training_path,'*.jpg'));
image_number = length(image_list);
I_manmadetrain = cell(image_number,1);
for i = 1:image_number
    image_name = image_list(i).name;
    path = fullfile(manmade_training_path,image_name);
    I_manmadetrain{i,1} = imread(path);
end
```

Figure 3　code structure logic of the stored file

For classification, using MATLAB, we can conveniently write code that can conduct matrix operation, mathematical operation and image processing operation. A structure data is built to store all the features temporarily in memory, a copy of extracted features of training set is stored in the hard drive for quick retrieve. The script which will firstly build a struct data containing the path of the images, the feature vectors and the classification label of this image. The struct will then be filled with the returned value of each feature extraction functions.



字段	Path	HSV_hist	LineSum	LineHist	Label
1	'manmade...	256x1 dou...	1.5335	18x1 double	0
2	'manmade...	256x1 dou...	1.1292	18x1 double	0
3	'manmade...	256x1 dou...	6.0531	18x1 double	0
4	'manmade...	256x1 dou...	2.4230	18x1 double	0
5	'manmade...	256x1 dou...	2.5660	18x1 double	0
6	'manmade...	256x1 dou...	3.4088	18x1 double	0
7	'manmade...	256x1 dou...	1.9467	18x1 double	0
8	'manmade...	256x1 dou...	5.5751	18x1 double	0
9	'manmade...	256x1 dou...	3.0503	18x1 double	0
10	'manmade...	256x1 dou...	2.6524	18x1 double	0
11	'manmade...	256x1 dou...	1.7303	18x1 double	0

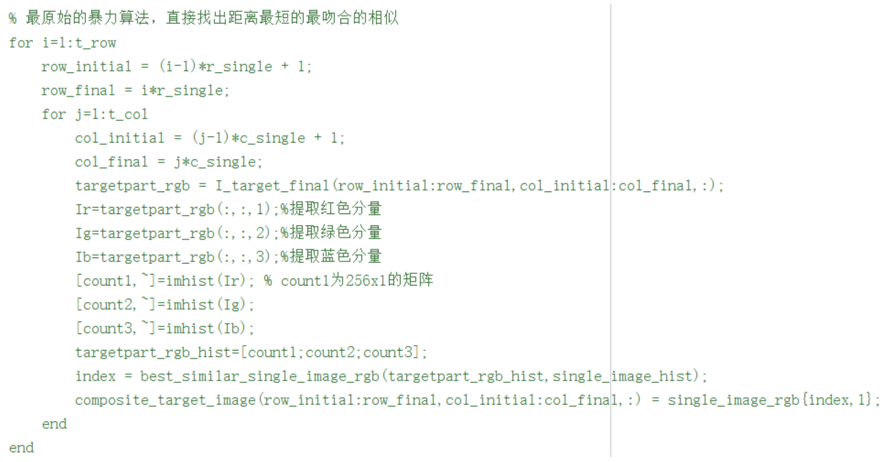
Figure 4 struct

2.2 Similarity Comparation

We wrote a total of three algorithms. According to the content learned in class, it is a natural and basic idea to compare the histogram of two pictures. Therefore, we used MATLAB to reproduce the operation.

2.2.1 RGB histogram

We first consider the RGB histogram as our thinking direction. The original idea was to directly use the brute force algorithm, a two-layer for loop, to subtract and square the histogram of each small target picture from the same bins of the histogram of the source picture after scaling (the default value of bins is 256), and this value also called Euclidean distance, and calculate the sum. This is what we mainly do in the *best_similar_single_image_rgb()* function. The smaller the sum, the smaller the difference between the two histograms, which further indicates that the two pictures are more similar, so we replace the source image with the smallest difference with the previous target sub-picture at this position. With this reciprocating operation, we can get a composite image.



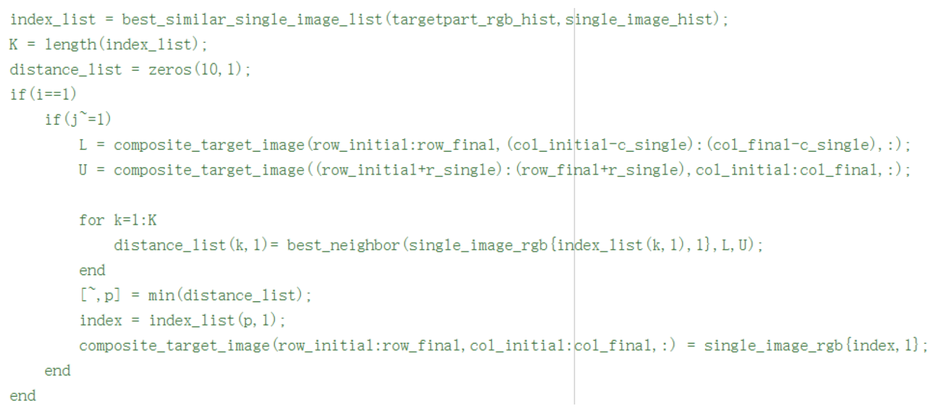
```
% 最原始的暴力算法，直接找出距离最短的最吻合的相似
for i=1:t_row
    row_initial = (i-1)*r_single + 1;
    row_final = i*r_single;
    for j=1:t_col
        col_initial = (j-1)*c_single + 1;
        col_final = j*c_single;
        targetpart_rgb = I_target_final(row_initial:row_final,col_initial:col_final,:);
        Ir=targetpart_rgb(:,:,1);%提取红色分量
        Ig=targetpart_rgb(:,:,2);%提取绿色分量
        Ib=targetpart_rgb(:,:,3);%提取蓝色分量
        [count1,~]=imhist(Ir); % count1为256x1的矩阵
        [count2,~]=imhist(Ig);
        [count3,~]=imhist(Ib);
        targetpart_rgb_hist=[count1;count2;count3];
        index = best_similar_single_image_rgb(targetpart_rgb_hist,single_image_hist);
        composite_target_image(row_initial:row_final,col_initial:col_final,:) = single_image_rgb{index,1};
    end
end
```

Figure 5 best_similar_single_image_rgb()

2.2.2 RGB histogram + edge process

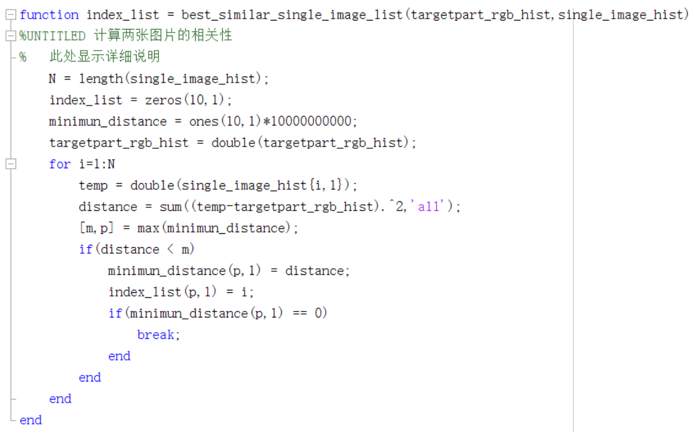
Since we are performing the for loop-operation, we search for the best source image for each small block by operating from left to right and from top to bottom. However, for two adjacent target sub-pictures (let's set the image on the left as A and the image on the right as B), they have indeed found their respective best source pictures to match. But for the right edge of A and the left edge of B, they may not fit well. Then this will also directly affect the synthesis effect of the final composite image. Then the same problem exists for the target sub-images that are adjacent to each other up and down.

Therefore, when we consider the calculation of the histogram, rather than considering the source image with the shortest distance. We can set a threshold size of a, that is, we can consider the relative distance. For a small source picture, the distance between them is very small. From this picture, find a picture with a good connection effect on the edge of the adjacent image, and use this image to replace the target sub-image, so this can improve the effect of the composite image to a certain extent. *best_similar_single_image_list()* function and *best_neighbor()* function achieves this as the Figure 7 and Figure 8 follows. For more details about edge process, we will discuss in the following page.



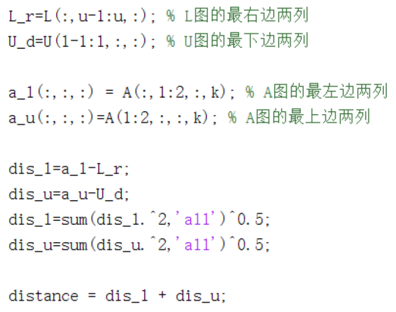
```
index_list = best_similar_single_image_list(targetpart_rgb_hist,single_image_hist);
K = length(index_list);
distance_list = zeros(10,1);
if(i==1)
    if(j~=1)
        L = composite_target_image(row_initial:row_final,(col_initial-c_single):(col_final-c_single),:);
        U = composite_target_image((row_initial+r_single):(row_final+r_single),col_initial:col_final,:);

        for k=1:K
            distance_list(k,1)= best_neighbor(single_image_rgb{index_list(k,1),1},L,U);
        end
        [~,p] = min(distance_list);
        index = index_list(p,1);
        composite_target_image(row_initial:row_final,col_initial:col_final,:) = single_image_rgb{index,1};
    end
end
```

Figure 6 One of the segments of the smooth connection algorithm for edge processing



```
function index_list = best_similar_single_image_list(targetpart_rgb_hist,single_image_hist)
%UNTITLED 计算两张图片的相关性
%   此处显示详细说明
    N = length(single_image_hist);
    index_list = zeros(10,1);
    minimun_distance = ones(10,1)*10000000000;
    targetpart_rgb_hist = double(targetpart_rgb_hist);
    for i=1:N
        temp = double(single_image_hist{i,1});
        distance = sum((temp-targetpart_rgb_hist).^2,'all');
        [m,p] = max(minimun_distance);
        if(distance < m)
            minimun_distance(p,1) = distance;
            index_list(p,1) = i;
            if(minimun_distance(p,1) == 0)
                break;
            end
        end
    end
end
```

Figure 7 best_similar_single_image_list() function



```
L_r=L(:,u-1:u,:); % L图的最右边两列
U_d=U(1-1:1,:,:); % U图的最下边两列

a_l(:,:,:) = A(:,1:2,:,k); % A图的最左边两列
a_u(:,:,:)=A(1:2,:,:,k); % A图的最上边两列

dis_l=a_l-L_r;
dis_u=a_u-U_d;
dis_l=sum(dis_l.^2,'all')^0.5;
dis_u=sum(dis_u.^2,'all')^0.5;

distance = dis_l + dis_u;
```

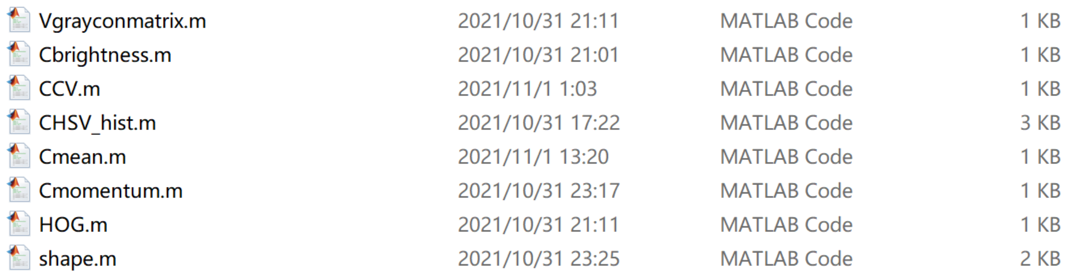
Figure 8 best_neighbor() function

2.2.3 HSV

In addition, we also wrote about using the HSV color model to compare similar images. HSV (Hue, Saturation, Value) is based on the intuitive characteristics of color: hue (H), saturation (S), lightness (V), hue H is measured by angle, and the value range is 0° ~360°. Calculated counterclockwise, *red* is 0°, *green* is 120°, and *blue* is 240°. Their complementary colors are: 60° for *yellow*, 180° for *cyan*, and 300° for *purple*. We classify the pre-processed pictures according to H, and add a corresponding label according to the H value, so that in the subsequent comparison calculation, we can filter the source images first according to whether the labels are the same, and find the target sub-images and the source images with the same label so that the source image can be similarly compared. This can greatly speed up efficiency. When looking for the best similar images, we mainly use the *best_similar_single_image_hsv()* function, which first calculates the H value of the target sub-image, and then finds the source image with the same label for similar comparison, using two images each value of the H matrix corresponds to subtracting and squaring to calculate 10 image with the minimum sum, and then for this 10 picture, calculate the similarity of the two pictures of the S matrix and the V matrix, so as to find the best one for the source picture, and replace the target sub image.

2.3 Feature Extracting

In the feature extraction procedure, we built different functions to compute the feature matrix of images, each will accept the input image and return the feature vector as $n \times 1$ double matrix. We build up many methods as demo to compare to get the bet perform in classification task shown in Figure 9.



Vgrayconmatrix.m	2021/10/31 21:11	MATLAB Code	1 KB
Cbrightness.m	2021/10/31 21:01	MATLAB Code	1 KB
CCV.m	2021/11/1 1:03	MATLAB Code	1 KB
CHSV_hist.m	2021/10/31 17:22	MATLAB Code	3 KB
Cmean.m	2021/11/1 13:20	MATLAB Code	1 KB
Cmomentum.m	2021/10/31 23:17	MATLAB Code	1 KB
HOG.m	2021/10/31 21:11	MATLAB Code	1 KB
shape.m	2021/10/31 23:25	MATLAB Code	2 KB

Figure 9 Feature extracting methods

2.4 KNN

After the feature extraction procedure, Knn can be put to use on each of the feature. Even though MATLAB provides prepared Knn model, however the users can only adjust the num of k in fitcknn(), which ignore the other features that may influence the accuracy of the KNN. Therefore, we developed our own Knn in which the user can decide the value of the k and also the way to compute the distance L using another function.

The Knn function will produce predicted result in numbers in which 0 represents manmade and 1 represents natural images. With results classified with multiple features, combining the result from different Knn classifier is implemented using the weighted voting algorithm. By summing up the product of the correctness and classified result of every classifier, followed by a division by the sum of the correctness of all classifier, we can get a decimal number. After rounding to integer, we can get the result label in 0 or 1 as the combined Knn classifier result.

2.5 Distance Calculation

There are a few methods of computing the distance of two set of data. One way is to directly subtract the number of each corresponding dimension and then sum up to a single number.

$$\text{Linear Distance: } L = \sum_{i=1}^{l} |F_1(i) - F_2(i)|$$

$$\text{Euclidean Distance: } L = \sum_{i=1}^{l} (F_1(i) - F_2(i))^2$$

Where l is the length of the feature vector.

Others including calculating the Euclidean distance, select the maximum distance of all dimensions. In the following classification process, we will be using simple direct distance since better classification accuracy is obtained. The experiment on selecting the method of distance is given in the later part of the report.

2.6 Neural Network

Inspired by the CNN using in classification, which is widely used in images classification nowadays. Therefore, refer to the tutorial of the pytorch we adjust the codes using in CIFAR10, that it can be more efficient in 2-classes classification. This part is just an extend exploration, we are interested using more efficient networks to get a better result.

3 Application

3.1 Image Generation

For this project, we were provided with sets of images of 1500 outdoor scenes. Each image has been labelled as either natural or manmade. The natural images are of scenes dominated by mountains, hills, desert, or sky, for example. The manmade images are of scenes dominated by commercial buildings, shops, markets, cities, and towns. We use these image sets for each Part of this project. The images sets are divided into disjoint subsets for training and testing as follows. We have two sets include training set (*500 natural images & 500 manmade images*) and test set (*250 natural images & 250 manmade images*)

3.1.1 Steps

Our tasks in Part A can be subdivided into the following six steps:

1) Extract image data, target image data;
2) Determine the size of the source image, the number of source images, and the ranks of the target image;
3) Resize the target image;
4) Individual source image processing: Obtain the size of the source image and the related three-color channel histogram required by the user;
5) Similar picture replacement: Find the best matching picture for each position in the target picture from the gallery;
6) Picture reconstruction;

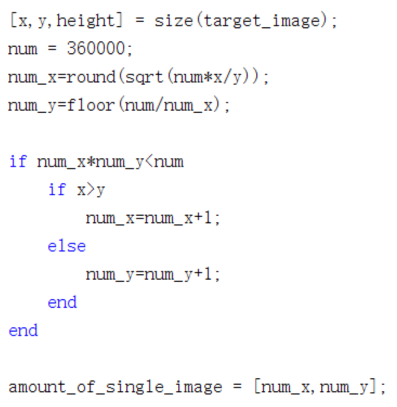
3.1.2 Basic process

According to our above task division, we use MATLAB to complete the code implementation of this part.

First, we need to import the pre-processed data into our program, that is, load the image data used to synthesize the target image: Load('*DataSave.mat*'). Then, load the target image and assign it to '*target_image*'.

Next, we need to get the number of pixels and the number of tiles from the user. Here, the user directly enters the row size and column size when inputting pixels, which is convenient for us to scale the source image appropriately according to the size of the pixels input by the user.

After the user enters the number of tiles, we can realize that the aspect ratio of the composite picture is similar to the aspect ratio of the target picture according to the aspect ratio of the target picture. The code for the approximate aspect ratio is shown in Figure 10:



```
[x,y,height] = size(target_image);
num = 360000;
num_x=round(sqrt(num*x/y));
num_y=floor(num/num_x);

if num_x*num_y<num
    if x>y
        num_x=num_x+1;
    else
        num_y=num_y+1;
    end
end

amount_of_single_image = [num_x,num_y];
```

Figure 10 : Determine the size of the composite picture and keep the same aspect ratio as the target picture

Finally, we need to call the composite_image_process() function to get our final composite image. We compare the original target picture and the synthesized target picture to complete the PART A part of the work.

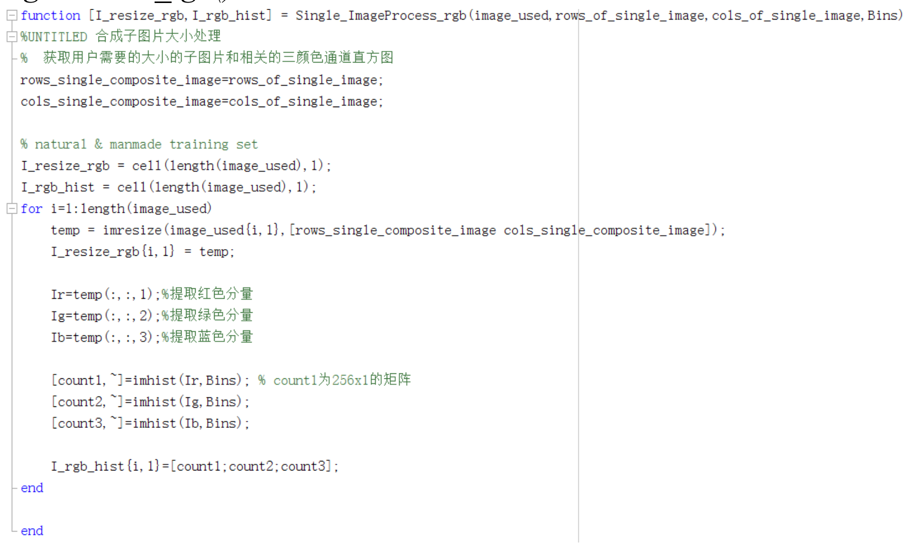
3.1.3 *composite_image_process()* function

We will mainly explain the specific implementation principle of the *composite_image_process()* function.

This function requires us to enter 4 parameters, namely *size_of_single_image* (the size requirement of the source picture), *amount_of_single_image* (the number of source pictures is how many pictures are used to composite image), *target_image* (contain target image data), image_used (which source images to use composite target image)

Entering the function body, we must first determine the zoom size of the source iamge, the number of source images used, and the division of the target image. As a result, we can scale the target image, where we can use the function *imresize()* that comes with MATLAB to achieve it. Then we need to use the function *Single_ImageProcess_rgb()* to get the sub-image of the size required by the user and the related three-color channel histogram. The purpose of this step is that we can process the size of the source image and extract the three-color channel histogram of each zoomed source image. This is beneficial to directly compare the corresponding histograms when replacing similar pictures later, which improves the computational efficiency and simplifies the time complexity. The figure 11 about

Single_ImageProcess_rgb() is as follows:



```
function [I_resize_rgb, I_rgb_hist] = Single_ImageProcess_rgb(image_used, rows_of_single_image, cols_of_single_image, Bins)
%UNTITLED 合成子图片大小处理
%  获取用户需要的大小的子图片和相关的三颜色通道直方图
rows_single_composite_image=rows_of_single_image;
cols_single_composite_image=cols_of_single_image;

% natural & manmade training set
I_resize_rgb = cell(length(image_used), 1);
I_rgb_hist = cell(length(image_used), 1);
for i=1:length(image_used)
    temp = imresize(image_used{i, 1}, [rows_single_composite_image cols_single_composite_image]);
    I_resize_rgb{i, 1} = temp;

    Ir=temp(:, :, 1);%提取红色分量
    Ig=temp(:, :, 2);%提取绿色分量
    Ib=temp(:, :, 3);%提取蓝色分量

    [count1, ~]=imhist(Ir, Bins); % count1为256x1的矩阵
    [count2, ~]=imhist(Ig, Bins);
    [count3, ~]=imhist(Ib, Bins);

    I_rgb_hist{i, 1}=[count1;count2;count3];
end

end
```

Figure 11　Single_ImageProcess_rgb()

The optimal synthetic image and the original image similarity can reach about 90%

3.1.4　Result

After we introduce these algorithms, we need test their effect. Figure 13 and figure 14 can show the effect of not considering the smooth connection of the edge and considering the smooth connection of the edge. Assume $bins = 256$, $pixel = 8x8$, $tile = 160000$. Figure 15 shows the effect by using HSV to composite target image. Assume $pixel = 8 \times 8, tile = 160000$.

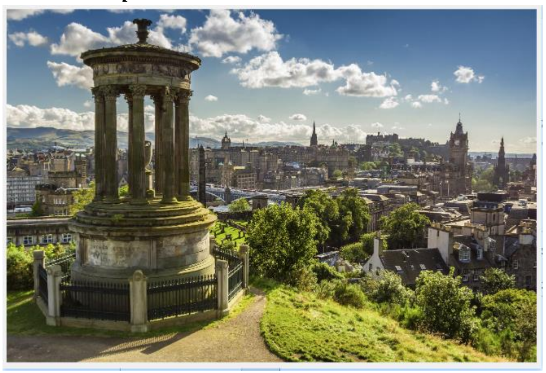


Figure 12 origin target image

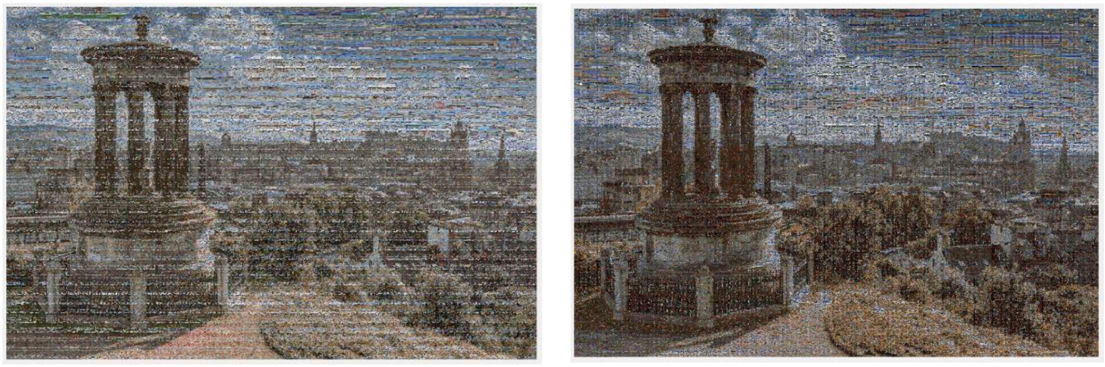


Figure 13 using RGB histogram （ Algorithm 1 ）

Figure 14 using RGB histogram + edge process to composite image (Algorithm 2)

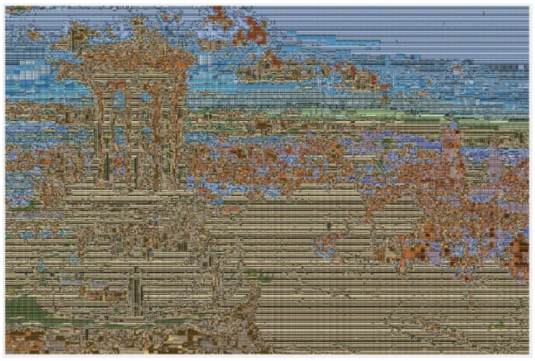


Figure 15 using HSV to composite image (Algorithm 3)

Algorithm 1 is rougher than algorithm 2's renderings, and we can obviously find that many adjacent edges are very unsmooth. Compared with algorithm one/two, algorithm three is closer to the target image in color, but the whole image is blurry, and many details of the target image cannot be restored well. So compare these three algorithms, we can find that algorithm2 is the best one.

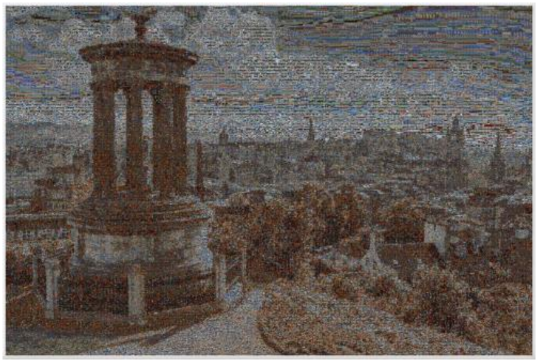
3.1.5 Fine tune

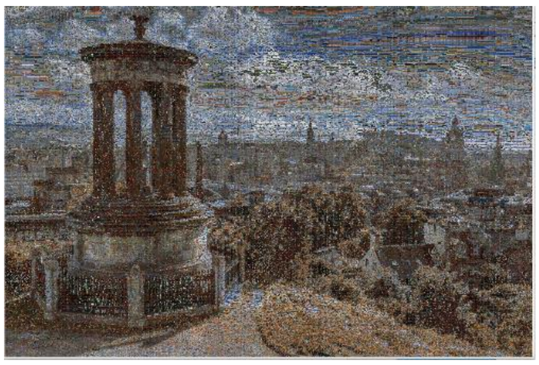
The adjustment of the following parameters adopts the principle of single variable, that is, changing one of the variables without changing the other variables.

As we know from the foregoing, Algorithm 2 has the best effect, because we perform parameter tuning work on the basis of Algorithm 2.

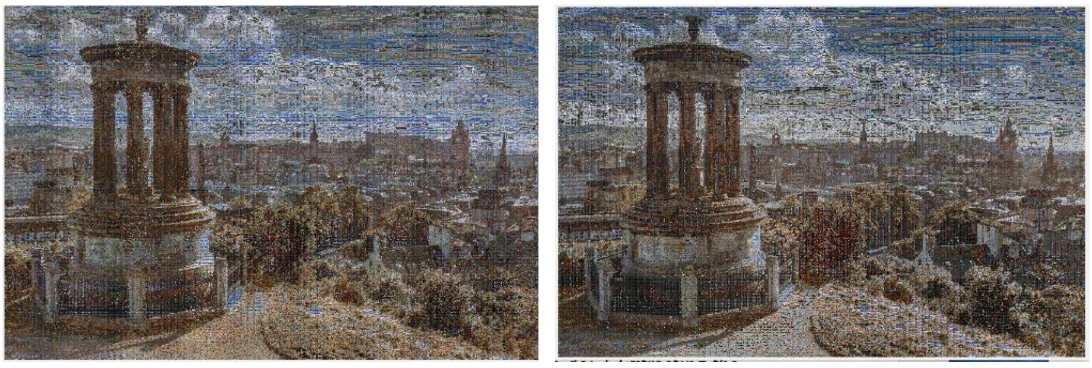
1) Pixel

We all know that pixels are composed of small squares of an image. These small squares have a clear position and assigned color value. The color and position of the small squares determine the appearance of the image. When we resampling the source image, we scale the source image to the pixel size specified by the user, where the size of the pixel is mxn. Of course, if the pixel is 1x1, the restored renderings will undoubtedly be better. But the problem is that this is to turn each source picture into a color block, and then compare and replace them one by one, so the realized picture does not have great theoretical significance. What we hope is to replace the target sub-picture with a more similar source picture. Therefore, we tried the results under different pixel cases, assuming that bins=256, the number of tiles is 160,000, and the following figure 16 is the effect diagram when the pixel is 3x3, 5x5, 8x8 and 10x10.





Pixel 1x1 pixel 5x5



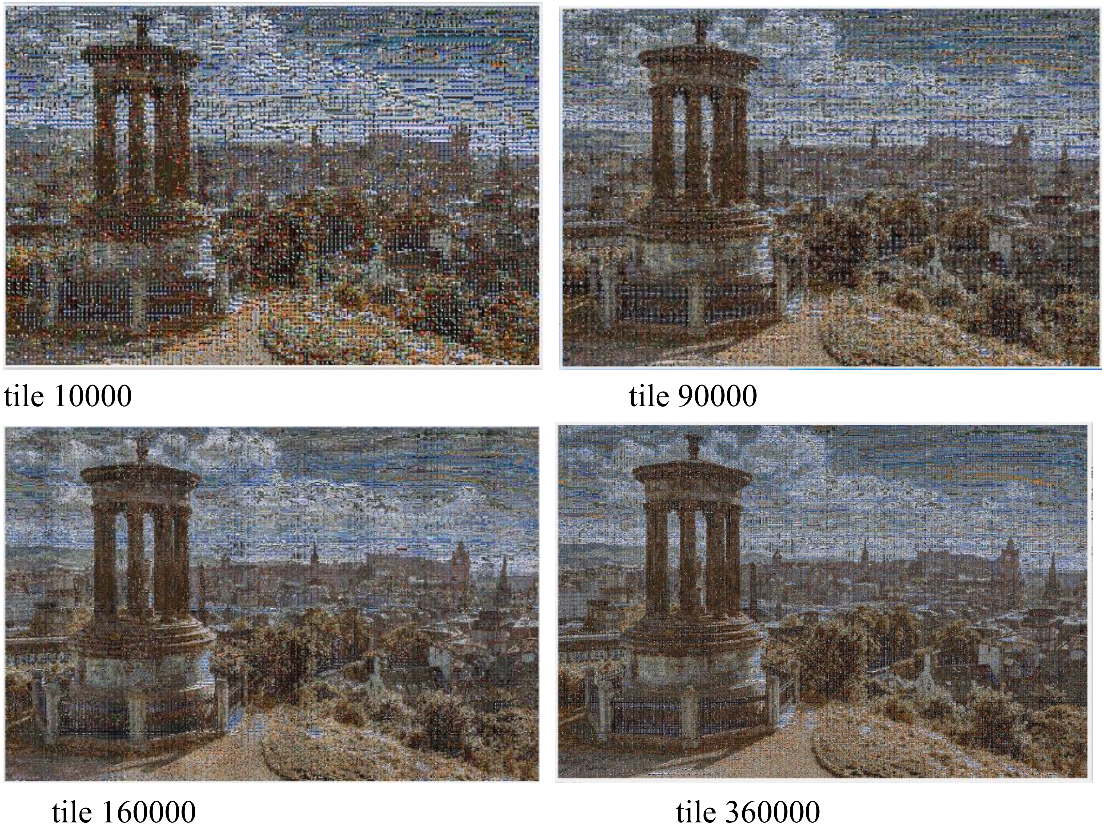
Pixel 8x8 pixel 10x10

Figure 16 different pixels influence the effect

As we can see, pixel 8x8 image maybe better than other since it is more clear and brighter than other images.

2) tile

We all know that how many images are used to composite the target image will have different effects on the final result. If the number of images is too small, then each image will account for a large proportion of the whole image, so it is difficult to restore our target image. Of course, the larger the number of images, the better the effect of composite images. However, due to the increase in the number of images, the calculation time of MATLAB will also increase, because we need to find a good balance point so that it takes less time to synthesize a target image with good results. Assuming bins = 256 and pixel = 8x8, in the following figure 17 is the effect diagram under the condition of 10,000, 9,0000, 16,0000, and 36,0000 respectively.



tile 10000 tile 90000

tile 160000 tile 360000

Figure 17 different tiles influence the effect

As we can see, with the number of pictures growing, composite pictures become clearer and smoother.

3) Bins (when using histogram)

We all know that using the *imhist()* function, we can find the histogram of the color image and get the intensity histograms corresponding to the three colors. The default value of bins in MATLAB is 256, but considering that our pixel is not large, there may be many values in each bin are all 0, because adjacent bins actually reflect the same color. Then if we consider the target sub-image and the histogram of the source image after scaling, if there are too many bins It will cause the sum also called Euclidean distance calculated by subtraction and squaring to be larger, but in fact the colors of adjacent bins are similar, which may cause us to be unable to choose a better source image. Therefore, controlling the number of bins is beneficial to improve the matching of similar images. Assuming that $tile = 16000, pixel = 8 \times 8$, the following figure 18 is the effect diagram under $bins = 32, 64, 128, 256,$ respectively

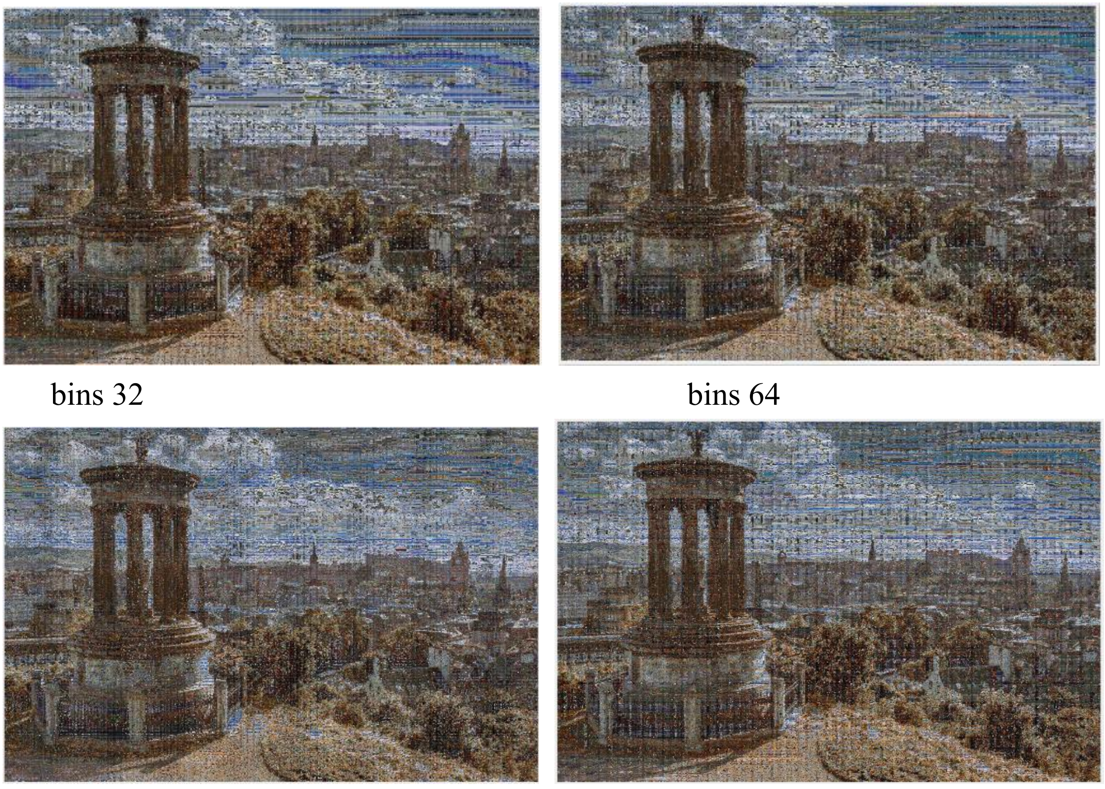


Figure 18 different bins influence the effect

Considering the clarity and smoothness at the same time, undoubtedly the effect is better when $bins = 128$.

In summary, we found that in the case of $pixel = 8 \times 8, tip = 160{,}000, and\ bins = 128$, it only takes 4 minutes to get the best result that our group thinks.

We tested it with different pictures and the results met our expectations.

The following Figure 19 shows the effect of the composite picture under the above conditions.

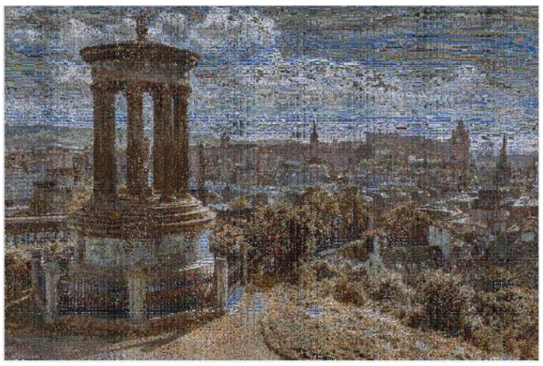


Figure 19 $pixel = 8x8, tip = 160{,}000, bins = 128$

3.1.6 Improvement

1) Consider edge processing for adjacent images.

For edge comparison, we refer to the classical methods image quilting[1]. Simply, we consider the connection edges of two neighbor images, evaluate their similarity (Euclid distance), then pick up the images can have smooth edge trans.

To concentrate some details, for we replace the tiles from top to bottom and from left to right, therefore for each inner picture we only need to compare the left and the above.

2) Compare the target sub-image and the source image directly, instead of comparing the three-color channel histograms of the two.

When comparing Histogram, we have actually overlooked a problem. That's when we went to get the Histogram of the picture, we actually ignored its location information. Since our pixel is not 1x1, there will be corresponding color information for each 1x1 cell on each pixel. Therefore, we are thinking about whether to directly subtract and square the values of the three corresponding matrices of the target sub-image and the original image to find the smallest one in the sum, which may further improve the effect of the composite image. So we conducted a test, assuming that bins = 128, pixel = 8x8, and tile = 160000, Figure x shows the effect of composite images after we directly compare the target sub-image with the source image. The following figure 20 shows the effect of the composite picture under the idea 2. Compared with RGB histogram, it is really better.

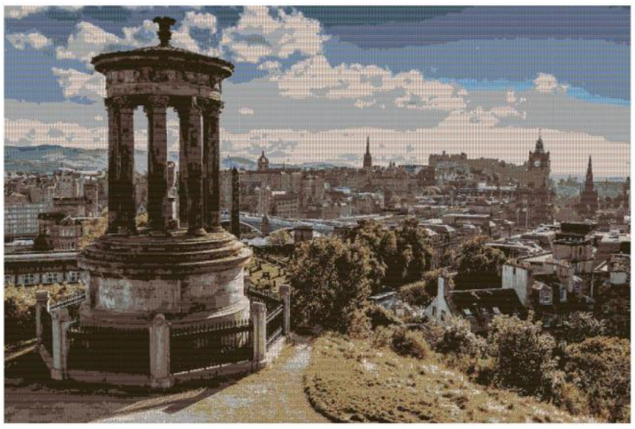


Figure 20 $pixel = 8x8, tip = 250{,}000, bins = 128$

3.2 Binary image classification

In this part of the, the binary classifier is to classify two kinds of image, which are either natural scenery or manmade scenery. We used K nearest neighbor (Knn) algorithm as the classifying method. Using the provided dataset of 1000 images with different dimensions, the classifier is able to identify whether the images from a different tests et are natural scenery or manmade scenery with up to 86.8% of accuracy.

3.3 Class-specific composite images

In this part of the task, we combined what we have implemented in the previous parts. We packed the previous parts' code as functions and can be easily invoked. Using Knn with features extracted, the type of image can be determined in second, followed by the composite image generation.

Algorithm Design

1.1.2. Feature Extraction

In the feature extraction process, we selected 3 features as the classifying features: Hough Line Angle Histogram, Hough Line Length Summation, HSV Color Space Unequally Divided Histogram. We also extracted other features, including Color Coherence Vector, HoG, Color Momentum, Color Brightness and Gray Level Co-occurrence Matrix. But we did not adopt these features in the final version of the classifier, since these features did not give satisfying result in the later classification process. For a given image, to extract its features, the dimension of the image should better be converted to a fixed dimension. Therefore, we firstly conducted scaling on every image to obtain images with the same dimension.

1.1.2.1. Hough Features

Then, we performed Hough transformation to extract the straight-line features of the image.

1.2. Implementation

3.4 CNN

We apply some adjustment on the exist networks. After all the adjustment, the accuracy became 85% during the tests in the limit of the facility.

1.3.1 Adjust related parameters

1. PART B: Binary Image Classification

1.3. Testing and Tuning

There are a few parameters to be determined in the whole process. First, the images need to be resized into a fixed dimension, which should be decided. We experimented on different dimensions and tested the accuracy performance of the later Knn process and found that for these given 1000 training set images and 500 test images, resizing to 768*768 image is better for Hough Histogram and 600*600 is better for Hough Line Length Summation.

CHSV_hist.m
LineHist.m
LineSum.m

Figure 21 The Functions for Feature Extraction

For Hough Transformation in the feature extraction part, the strength of the line appearing on the image can be set to filter the strongest lines in the images. After experiments, we take the 15 strongest lines in the image with threshold of above 0.15*the maximum value in the Hough transformation. After detecting the line, the line segments need to be decided using built in function houghlines(). we adjusted the FillGap and MinimumLength of the segment extraction process to $\frac{1}{150}$ of image size and $\frac{1}{30}$ of image size respectively. As for the number of bins of the histogram of angles, after experiments, we decided to divide the angle output of Hough lines which is -90° to 89° into 18 parts, i.e.: (-90:10:89). This will combine the line with the similar angle into the same bin and sum for the length of the lines.

In the HSV Color Space Unequally Divided Histogram feature extraction, the resolution of each of HSV channels is decided as 16, 4, 4, Which will give hue information more weight in the histogram.

In the Knn part, we need to decide different number of nearest neighbors for each classifier. After tuning for each Knn classifier, we concluded a set of K value being:

Hough Line Histogram:	K=30
Hough Line Length Summation:	K=200
HSV Color Space Unequally Divided Histogram:	K=5

1.4. Evaluation

1.4.1. Accuracy

After fine tuning the parameters, we are able to get relatively satisfying result of overall correctness of 86.8% in the test set traversal. The corresponding correctness of each classifier is shown below.

Feature	K (Number of Nearest Neighbor)	Correctness
Hough Line Histogram	30	82.6%
Hough Line Length Summation	200	82.8%
HSV Color Space Unequally Divided Histogram	5	71.4%
Combined Knn Classifier	/	86.8%

1.4.2. Speed

The program sequence contains the feature extraction and Knn algorithm. Time consumption of each is displayed below.\

Sequence	Time Consumption (s)
Training Set Feature Extraction	78.99
Test Set Feature Extraction	37.78
Knn Classifiers	1.46

*Data Measured Using Intel Core i5 10500es @4.0GHz

2. PART C Class-specific Composite Images

In this part of the task, we combined what we have implemented in the previous parts. We packed the previous parts' code as functions and can be easily invoked. Using Knn with features extracted, the type of image can be determined in second, followed by the composite image generation.

The generated images from the same class images are shown below.

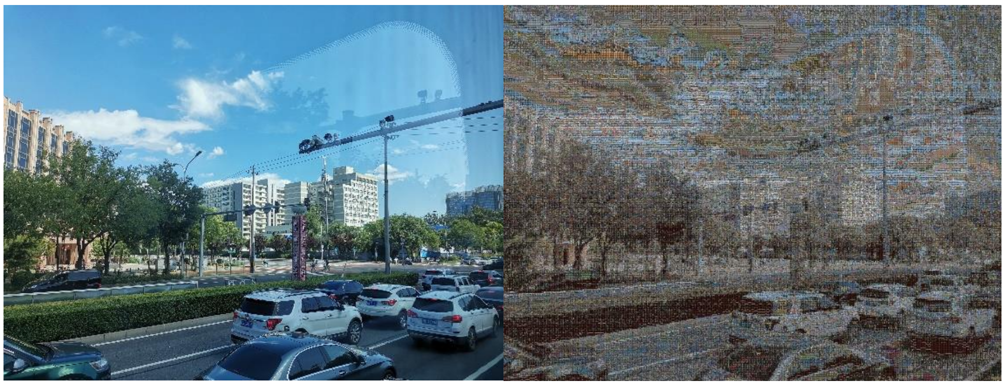


Figure 22 Manmade Image 1, Correctly Classified

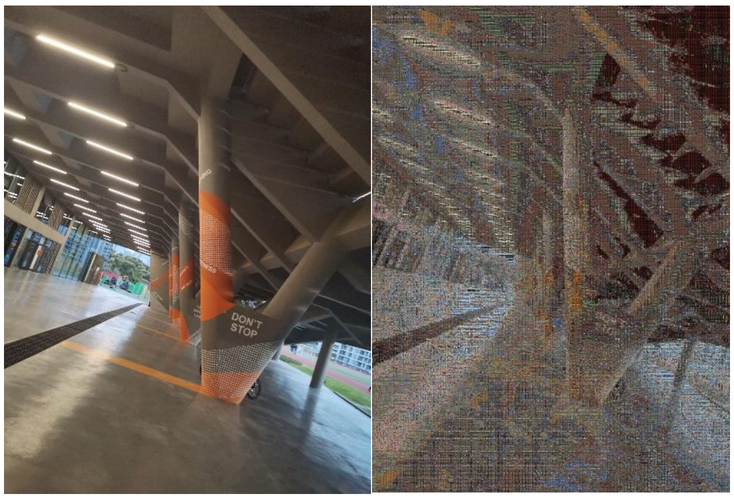


Figure 23 Manmade Image 2, Correctly Classified

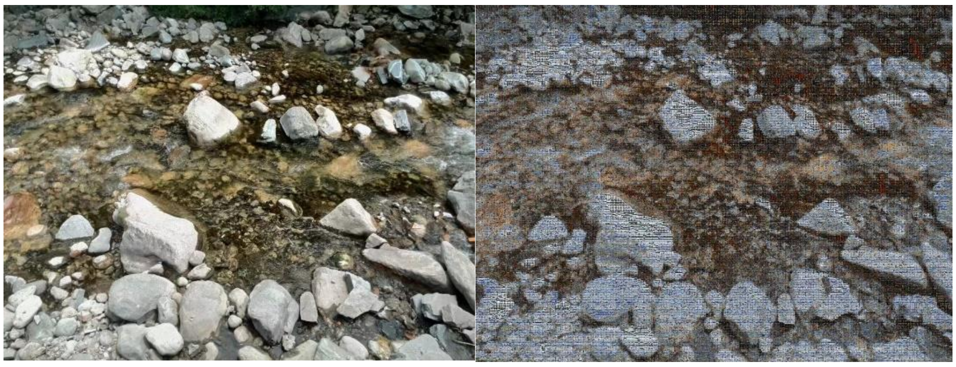


Figure 24 Natural Image 1, Correctly Classified

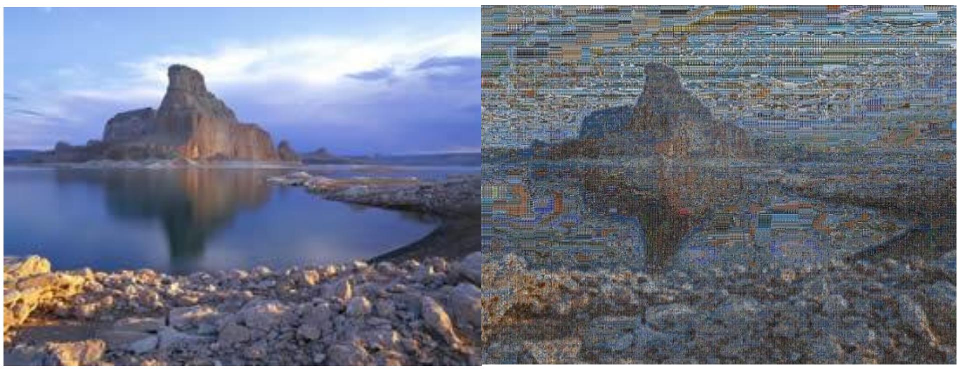


Figure 25 Natural Image 2, Correctly Classified

Part D: Expansion and future work
Inspired by the CNN using in classification, which is widely used in images classification nowadays. Therefore, refer to the tutorial of the pytorch we adjust the codes using in CIFAR10, that it can be more efficient in 2-classes classification.

We first change the value of some layers. The first element to be changed is the last layer. Linear-layer, for now we only have 2 categories.

```python
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, 7)
        self.conv2 = nn.Conv2d(16, 16, 7)
        self.pool = nn.MaxPool2d(2)

        self.fc1 = nn.Linear(16 * 51 * 51, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 2)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # print(x.shape)
        x = torch.flatten(x, 1)  # flatten all dimensions except batch
        # print(x.shape)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Then we adjust some hyperparameters. For batch size, consider about the task is to complete 2-classes classification, increase batch-size is obviously a good direction to start. Moreover, for the given data is quite limited (500 manmade, 500 natural), we have to increase the epoch to get fully usage of every data.

```python
batch_size = 8
# 1\2 都是78%
# 1、2>5>4
# 8 的loss似乎更小

classes = ('manmade', 'natural')
# imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join('%5s' % classes[labels[j]] for j in range(batch_size)))

net = Net().cuda()

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
# optimizer=optim.Adam(net.parameters(),lr=0.001)

for epoch in range(6):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(train_dataloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs=torch.autograd.Variable(inputs).cuda()
        labels=torch.autograd.Variable(labels).cuda()

        # forward + backward + optimize
        # print(inputs.shape)
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

After adjust of the CNN, we also have to provide processing on the image data. Mostly, we reshape the data in the same shape(224*224) (this also draw out some problems for some photos are “long”, while the others are “fat”)

```
transformed_trainset = MyDataset(root_dir='./data1/train',
                                  names_file='./data1/train/train.txt',
                                  transform=transforms.Compose(
                                      [Resize((224,224)),
                                       ToTensor()]
                                  ))

trainset_dataloader = DataLoader(dataset=transformed_trainset,
                                    batch_size=4,
                                    shuffle=True,
                                    num_workers=4)
```

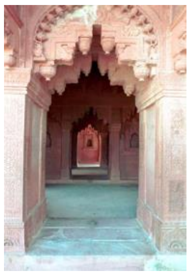


Figure 26 long

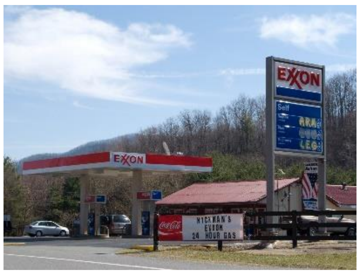
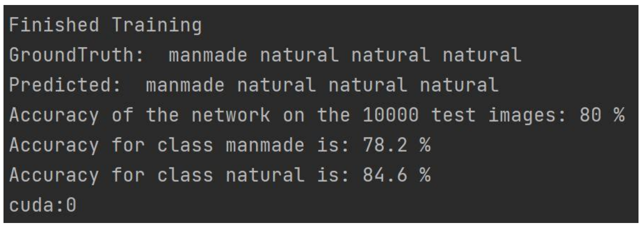


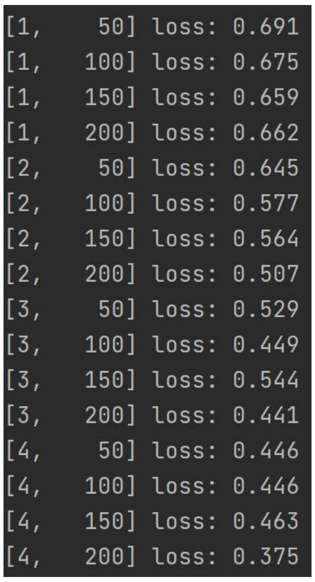
Figure 27 fat



```
Finished Training
GroundTruth:  manmade natural natural natural
Predicted:  manmade natural natural natural
Accuracy of the network on the 10000 test images: 80 %
Accuracy for class manmade is: 78.2 %
Accuracy for class natural is: 84.6 %
cuda:0
```

Figure 28 result

After all the adjustment, the accuracy became 85% during the tests in the limit of the facility. As it shown in the picture, the loss of the network was becoming less. Thus, for a reasonable guessing, if increase the epoch, the accuracy may increase a bit as well. And also when the CNN apply on the dataset of images having same shape, it will perform well.


```
[1,    50] loss: 0.691
[1,   100] loss: 0.675
[1,   150] loss: 0.659
[1,   200] loss: 0.662
[2,    50] loss: 0.645
[2,   100] loss: 0.577
[2,   150] loss: 0.564
[2,   200] loss: 0.507
[3,    50] loss: 0.529
[3,   100] loss: 0.449
[3,   150] loss: 0.544
[3,   200] loss: 0.441
[4,    50] loss: 0.446
[4,   100] loss: 0.446
[4,   150] loss: 0.463
[4,   200] loss: 0.375
```

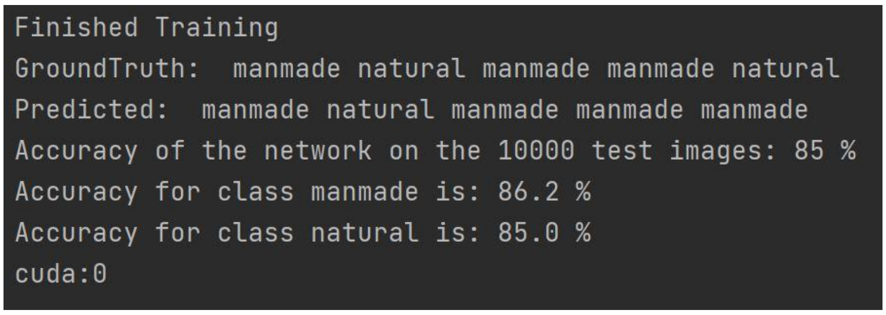

```
Finished Training
GroundTruth:  manmade natural manmade manmade natural
Predicted:  manmade natural manmade manmade manmade
Accuracy of the network on the 10000 test images: 85 %
Accuracy for class manmade is: 86.2 %
Accuracy for class natural is: 85.0 %
cuda:0
```

Figure 29 final result

Summary

We mainly implemented the two basic parts of synthetic image generation and binary image classification (natural scenery or artificial scenery) using machine learning methods, and combined the two parts to automatically determine which picture belongs to after selecting a specific picture category, and generate a composite image. In addition, we also use python to construct a neural network to achieve binary classification of pictures, and the effect is also very satisfactory.

Reference

[1] N. Silberman and S. Guadarrama. Tensorflowslim image classification model lib rary, 2016.

[2] MI Abdul-Rahim and Zheng
Sheng Yu. An in depth review paper on numerous image mosaicing approaches and te chniques. International Journal of Engineering and Management
Research (IJEMR), 8(2):19–35, 2018

[3] Bill Green. Canny edge detection tutorial. Retrieved: March, 6:2005, 2002.

[4] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragom ir Anguelov,Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going dee

per with convolutions. In Proceedings of the IEEE Conference on Computer Vision an
d Pattern Recognition (CVPR), pages 1–12, June 2015.