For office use only	Team Control Number	For office use only
T1 _____	**0025**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**A**	F4 _____

2018
MCM/ICM
Summary Sheet

unsure

Summary

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Keywords: keyword1; keyword2

1 Introduction

1.1 Background

With the rapid development of artificial intelligence, there are more and more applications of images, such as face recognition and automatic recognition. There are two things that cannot be ignored before using images for related tasks.The first is image compression, in order to reduce the redundant information in image data and to store and transmit data in a more efficient format; the second is image restoration, which USES certain techniques to reproduce the missing information in the image. These two technologies have laid a foundation for the rapid development of relevant image application technologies. In this context, we want to be able to build a process model to implement these two technologies.

1.2 Restatement of Problem

We are required to build some mathematical model to process the data of images:

- A lossy image compression model
- An inpainting model to recover the image with 20% missing information.
- A model to compress a surveillance video in a shopping mall.

1.3 Overview of Our Work

First, we find a few key points in these questions :

- The image compression we need to do is lossy compression.
- How to make image compression model suitable for general situation?
- What is the relationship between the models we want to build?
- Does the missing data proportion have special meaning?
- What is the measure of restoration?
- All missing pixel values for a given image are non-zero.
- How to deal with the static background of video?

2 Assumptions

- We are dealing with bitmaps rather than vectors.
- Assume that the missing types of information in question 3 are the same as those in the case diagram.

- Assume that the image the image compression algorithm to process is not too extreme for the its size, that is, the image size is not too small or too large.
- Assume the image compression algorithm is aimed at the image of RGB color.
- Assume that the image compression requires not to lose too much information, not considering the high compression of the signal loss.

3 Notations

4 Our Model

4.1 Model I

For a digital image using RGB color mode, we first separate the three color channels of red, green and blue.The image data of each channel can be represented by a 2-dimensional matrix. Each element in the 2-dimensional matrix corresponds to one pixel, and its value is the pixel's gray value. After obtaining the three 2-dimensional matrices, We first convert it from RGB to YCbCr. Where Y is the brightness component, Cb is the blue color component, and Cr is the red color component.The human eye is more sensitive to the Y component, so it will be less noticeable to the naked eye when subsampling the chromaticity component to reduce the chromaticity component.YCbCr color space and RGB color space conversion formulas are as follows:

$$\begin{cases} Y = 0.299R + 0.587G + 0.114B \\ Cb = 0.564(B - Y) \\ Cr = 0.713(R - Y) \end{cases} \tag{1}$$

$$\begin{cases} R = Y + 1.402Cr \\ G = Y - 0.344Cb - 0.714Cr \\ B = Y + 1.772Cb \end{cases} \tag{2}$$

We decide use YCbCr 4:2:0 format,which is by far the most common color representation used in compressedimages and video.[1] We sampled it in a 4:2:0 format. 4: 0 will correspond to four brightness values with 1 color value. For the first chromaticity element, there are two sample values, and the second chromaticity element will not be sampled. This does not produce a complete color image. In fact, 4:2:0 means that each scan has two color samples, which are sampled every two lines. This sampling is to reach a certain lossy compression without affecting the visual experience.

We divide them into sub-blocks separately to improve the efficiency of subsequent operations. Our specific operation is to divide the original matrix into sub-blocks of 8×8 in order from left to right and from top to bottom. After this step, A sub-block is considered as the basic unit to operate. There are 2 things to note about blocking:

- Each value in the 2-dimensional matrix needs to be subtracted by 128 before blocking so that the range of each value in the 2-dimensional matrix is changed from 0 to 255 to -128 to 127.

- The length or width of the original image is not a multiple of 8, which needs to make up to be multiples of 8 so that the 2-dimensional matrix is completely divided into several sub-blocks.

There are 3 advantages to blocking:

- To facilitate the DCT.
- There are no restrictions on the length-width ratio of the image.
- Take into account the local similarity of the picture.

The discrete cosine transform (DCT) is a technique for converting a signal into elementary frequency components. It is widely used in image compression.[2] And our model uses 2-dimensional DCT.The most common DCT definition of a 1-D sequence of length N is

$$C(u) = \alpha(u) \sum_{x=0}^{N-1} f(x) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \tag{3}$$

for u = 0,1,2...,$N-1$. The inverse transformation is defined as

$$f(x) = \sum_{x=0}^{N-1} \alpha(u) C(u) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \tag{4}$$

for u = 0,1,2...,$N-1$. In both equations (3) and (4) $\alpha(u)$ is defined as

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}}, & \text{for } u = 0 \\ \sqrt{\frac{2}{N}}, & \text{for } u \neq 0 \end{cases} \tag{5}$$

The 2-D DCT is a direct extension of the 1-D case and is given by

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right] \tag{6}$$

for u = 0,1,2...,$N-1$ and $\alpha(u)$ and $\alpha(v)$ are defined in (5). The inverse transform is defined as

$$f(x,y) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \alpha(u)\alpha(v) C(u,v) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right] \tag{7}$$

for u = 0,1,2...,$N-1$. The 2-D basis functions can be generated by multiplying the horizontally oriented 1-D basis functions with vertically oriented set of the same functions.[3] DCT is the transform of signal from time domain to frequency domain, and its transformation result is not complex, all is real.Each 8×8 original pixels is changed into 8×8 digits, and each number is composed of original 64 data through the basis function.

The next step is quantization. Quantization here refers to the quantization of the frequency coefficient after FDCT. The aim is to reduce the magnitude of the non-zero coefficient and increase the number of zero coefficients.

[[[[éĞŔåŇŰèąĺ]]]]

The quantization that we do is to divide the pixel value by the quantized table.Because the value in the upper left corner of the quantization table is smaller, the value in the upper right corner is larger, the purpose of maintaining low frequency components and suppressing high frequency components is achieved.

After that, Zig-Zag scan is performed on the quantized change coefficient. The so-called Zig-Zag scan is to start from the upper left corner of the matrix and scan according to the shape of the letter 'Z'. The purpose of the scanning is as follows:

- To group low frequency coefficients in top of vector.
- Maps $m \times n$ to a $1 \times m * n$ vector.

[[[[ZigZag]]]]

After Zig-Zag scanning, the low-frequency components in this one-dimensional vector are in the front, and the high frequency components are later. It is worth noting that the high frequency component has a large number of continuous zero values, which is making for 0-RLE.

0-RLE is an encoded mode we create which referred to RLE and aimming at zero. RLE(Run-length encoding) is a very simple form of lossless data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs. For example, With a RLE data compression algorithm applied to the a character string 'AAAABBB', it can be rendered as follows:'4A3B'. 0-RLE only do special coding for 0, other values are reserved. The specific step is as follows:

Initialize a queue, and the encoder will scan the elements of the vector to the tail. If zero is encountered, the num of zeroes is counted until a non-zero is encountered. If the non-zero value is encountered, zero and its count are pressed into the queue. If the next number is non-zero, the current number is pressed into the queue. When the tail is read, if the value is zero, zero and its count are pressed into the queue; if the value is non-zero, it is pressed into the queue.For example, With a RLE data compression algorithm applied to the a digital string '0 0 1 2 3 0 1 0 0 0 0 0 0 0 1', it can be rendered as follows :'0 2 1 2 3 0 1 1 0 7 1'.

Final, the 1-dimensional vector obtained from last step is encoded with huffman coding.Huffman coding is the basic method of image data scan again and calculate the probability of all pixels appear, according to the size of the probability of different length of code word only, the greater the probability of the pixels shorter code. The result of huffman coding we do is a hoffman collation of the image. The encoded data is the code word corresponding to each pixel, and the corresponding relation between the code word and the actual pixel value is recorded in the code table.

For a set of symbols with a uniform probability distribution and a number of members which is a power of two, Huffman coding is equivalent to simple binary block encoding.

Input

Alphabet $A = \{a_1, a_2, \cdots, a_n\}$, which is the symbol alphabet of size n.

Tuple $W = (w_1, w_2, \cdots, w_n)$, which is the tuple of the (positive) symbol weights (usually proportional to probabilities), i.e. $w_i = \text{weight}\,(a_i)\,, 1 \leq i \leq n$.

Output

Code $C\,(W) = (c_1, c_2, \cdots, c_n)$, which is the tuple of (binary) codewords, wherec_i is the codeword for $a_i, 1 \leq i \leq n$.

Goal

Let $L\,(C\,(W)) = \sum_{i=1}^{n} w_i \times \text{length}\,(c_i) L\,(C\,(W)) = \sum_{i=1}^{n} w_i \times \text{length}\,(c_i)$be the weighted path length of code C C. Condition: $L\,(C\,(W)) \leq L\,(T\,(W)) L\,(C\,(W)) \leq L\,(T\,(W))$for any code $T\,(W)$.

[[[Huffman coding]]]

For the above steps, the steps before the Zig-Zag scan is a lossy image compression to eliminate data redundancy, which is unble to recover after inverse transform. the steps after the Zig-Zag scan is a lossless image compression to eliminate encoding redundancy, which is able to reduce storage space needed.

4.2 Model II

4.2.1 Model proposed

For problem 3, due to missing data itself is irreversible, we find after the analyses of current situation of the lack of case diagram, as shown in the image information loss and image noise are similar, so this article will be treated as missing data processing as a noise.

Spatial filtering is an enhanced method based on neighborhood processing. Spatial filtering processes directly in the two-dimensional space of the image. In other words, the grayscale value of each pixel is processed. It use a template to perform some mathematical operation on all pixels of each pixel and its surround neighborhood to get that new gray value of the pixel. The new grayscale value is not only related to the gray value of the pixel, but also to the gray value of the pixel in the neighborhood. In this paper, a variety of filters are used and compared. As shown in the figure below, with the same 3X3 template, the median filter is finally found to be suitable for the case diagram.

[åŻ¿ WienerFilterting]

[åŻ¿ MeanFiltering]

[åŻ¿ Median Filtering]

Median filter is a commonly used nonlinear filter. The basic process is to select the median of each pixel in a neighborhood of pixels to be processed instead of the pixel to be processed. The main principle is that the gray value of the pixel is close to the surrounding pixel, and the isolated noise point will not be the median, so the median filter can eliminate the salt and pepper noise very well. In addition, compared with the mean filter, the median filter can effectively protect the boundary information of the

image while eliminating the noise, and it will not cause a great blur to the image. The formula is as follows:

$$\hat{f}(x,y) = median_{(s,t)\in S_{xy}}\{g(s,t)\} \tag{8}$$

We found in the test, the effect of median filter are greatly influenced by the filter window size, in eliminating noise and protect the image detail there exists a contradiction: the filter window is lesser, can very good protection in the image details, but for the noise of the filtering effect is not very good; On the contrary, the filter with larger window size has better noise filtering effect, but it can cause certain blurring to the image. In addition, according to the principle of median filter, if the number of noise points in the filter window is greater than the number of pixels in the entire window, the median filter can not filter out the noise very well.

Therefore the median filter mentioned above, under the condition of noise density is not very big (according to the experience, the probability of the presence of the noise is less than 0.2), the effect is good.

[åŻ¿ 3X3Median Filtering]

[åŻ¿ 3X3Adaptive Median Filtering]

So we finally chose the adaptive median filter. According to predefined condition, in the process of filtering, dynamic change the window size of the filter, as well as figure out whether the current pixel noise according to certain conditions. At the same time, it will judge whether the current pixel is noisy according to certain conditions, and if it is,replace the current pixel with the median value; If not, the pixel value is not changed.

Adaptive median filter has three purposes:

- Filter the salt and pepper noise.
- Smooth other non-impulsive noise.
- Protect the details of the image as much as possible to avoid thinning or roughing the edges of the image.

4.2.2 Model interpretation

The adaptive filter can not only filter the salt and pepper noise with higher probability, but also can better protect the details of the image,which is not possible with conventional median filter. The adaptive median filter also requires a rectangular window S_{xy}, which is different from the conventional median filter that the size of the window will change during the filtering process. It is important to note that the output of the filter is a pixel value, which is used to replace the pixel values at the point (x,y), and the point (x,y) is the central location of the filter window.

The adaptive median filter has two processing processes: A and B:

A:

$A_1 = Z_{med} Z_{min}$; $A_2 = Z_{med} Z_{max}$. If $A_1 > 0$ and $A_2 < 0$, jump to B; Otherwise, increase the size of the window. If the size of the rear window is less than or equal to S_{max}, repeat the A process. Otherwise, output Z_{med}.

B:

$B_1 = Z_{xy}Z_{min}$; $B_2 = Z_{xy}Z_{max}$. If $B_1 > 0$and $B_2 < 0$, then the output is Z_{xy}. Otherwise output Z_{med}.

The purpose of process A is to determine whether the median Z_{med} is noise in the current window.If $Z_{min} < Z_{xy} < Z_{max}$, the median $Z_{med}$is not noise, then it turn to the process B testing to test whether the pixel Zxy of the center position of the current window is a noise point. If $Z_{min} < Z_{xy} < Z_{max}$, Z_{xy} is not a noise, then the filter output Z_{xy}; If the above conditions are not met, Z_{xy} is determined to be noise, and the output value is Z_{med} (Z_{med} is not noise in A).

If Z_{med} does not meet the condition $Z_{min} < Z_{med} < Z_{max}$, the median Z_{med} is a noise. In this case, the window size of the filter needs to be enlarged, and the median value of a non-noise point is searched in a larger scope until it is found, then jump to B; Or, the size of the window reaches the maximum value, then returns the median found and exits.

The total flow chart is as follows: [åŻ¿ èĞłéĂĆåžŤäÿ■åĂijæżd'æşćåŹĺæĂżæţĄçĺŃåŻ¿]

From the above analysis, it can be seen that the probability of noise is low, and the adaptive median filter can get the results quickly, without the need to increase the size of the window. On the other hand, the probability of noise is higher, and it needs to increase the window size of the filter, which is the characteristic of a median filter which requires a larger filter window size when meets more noise points.

5 title

References

[1] D. E. KNUTH The TEXbook the American Mathematical Society and Addison-Wesley Publishing Company , 1984-1986.

[2] Lamport, Leslie, LATEX: " A Document Preparation System ", Addison-Wesley Publishing Company, 1986.

[3] `http://www.latexstudio.net/`

[4] `http://www.chinatex.org/`

Appendices

Appendix A First appendix

Aliquam lectus. Vivamus leo. Quisque ornare tellus ullamcorper nulla. Mauris porttitor pharetra tortor. Sed fringilla justo sed mauris. Mauris tellus. Sed non leo. Nullam elementum, magna in cursus sodales, augue est scelerisque sapien, venenatis congue nulla arcu et pede. Ut suscipit enim vel sapien. Donec congue. Maecenas urna mi, suscipit in, placerat ut, vestibulum ut, massa. Fusce ultrices nulla et nisl.

Here are simulation programmes we used in our model as follow.

Input matlab source:

```
function [t,seat,aisle]=OI6Sim(n,target,seated)
pab=rand(1,n);
for i=1:n
    if pab(i)<0.4
       aisleTime(i)=0;
    else
        aisleTime(i)=trirnd(3.2,7.1,38.7);
    end
end
```

Appendix B Second appendix

some more text **Input C++ source:**

```
//============================================================================
// Name        : Sudoku.cpp
// Author      : wzlf11
// Version     : a.0
// Copyright   : Your copyright notice
// Description : Sudoku in C++.
//============================================================================

#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int table[9][9];

int main() {

    for(int i = 0; i < 9; i++){
        table[0][i] = i + 1;
    }

    srand((unsigned int)time(NULL));

    shuffle((int *)&table[0], 9);

    while(!put_line(1))
    {
        shuffle((int *)&table[0], 9);
    }

    for(int x = 0; x < 9; x++){
        for(int y = 0; y < 9; y++){
            cout << table[x][y] << " ";
        }

        cout << endl;
    }
```

```
    return 0;
}
```