浙江大学实验报告

课程名称：图象信息处理　　指导老师：宋明黎　　成绩：_____

实验名称：Assignment-2 binarization operation

专业：求是科学班（计算机）

姓名：蒋仕彪

学号：3170102587

日期：2018/11/11

一、实验目的和要求

1.将一张图片二值化。

2.将二值化图片进行腐蚀（erosion）操作。

3.将二值化图片进行膨胀（dilation）操作。

4.将二值化图片进行开（opening）操作。

5.将二值化图片进行闭（closing）操作。

二、实验内容和原理

1.图片的二值化。

对于一张灰度图片，确定一个阈值 Threshold，并重构图片为：

- Construct a binary image: thresholding the grayscale image by reset the pixel value, i.e.
- $$\begin{cases} I(x,y) = 0 & if\ I(x,y) < Threshold \\ I(x,y) = 255 & if\ I(x,y) \geq Threshold \end{cases}$$

显然，对于不同的 Threshold 值会得到不同的图片。但我们要尽可能地保留原图的信息。我们的目的，是想让黑色块（0）和白色块（255）内部点的灰度值尽量接近，但彼此灰度值尽量远离。如果我们用方差去刻画这个问题，我们就是想让 σ^2(between) 最大。

- $\sigma^2_{within}(T) = \frac{N_{Fgrd}(T)}{N}\sigma^2_{Fgrd}(T) + \frac{N_{Bgrd}(T)}{N}\sigma^2_{Bgrd}(T)$
- $\sigma^2_{between}(T) = \sigma^2 - \sigma^2_{within}(T)$
- $= \left(\frac{1}{N}\sum_{x,y}(f^2[x,y] - \mu^2)\right) - \frac{N_{Fgrd}}{N}\left(\frac{1}{N_{Fgrd}}\sum_{x,y\in Fgrd}(f^2[x,y] - \mu^2_{Fgrd})\right) - \frac{N_{Bgrd}}{N}\left(\frac{1}{N_{Bgrd}}\sum_{x,y\in Bgrd}(f^2[x,y] - \mu^2_{Bgrd})\right)$
- $= -\mu^2 + \frac{N_{Fgrd}}{N}\mu^2_{Fgrd} + \frac{N_{Bgrd}}{N}\mu^2_{Bgrd}$
- $= \frac{N_{Fgrd}}{N}(\mu_{Fgrd} - \mu)^2 + \frac{N_{Bgrd}}{N}(\mu_{Bgrd} - \mu)^2$
- $\rightarrow \frac{N_{Fgrd}(T)\cdot N_{Bgrd}(T)}{N^2}\left(\mu_{Fgrd}(T) - \mu_{Bgrd}(T)\right)^2$

具体地，我们枚举每一个可能的阈值，带入公式计算，找到一个差距最大的方差值作为 Threshold

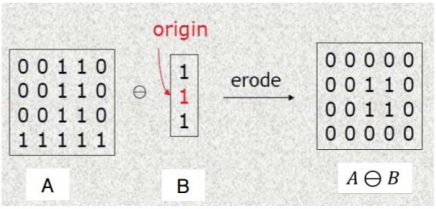
2. Erosion，Dilation，Opening，Closing 四个操作。

Erosion

A: Binary image
B: binary template, structure element

$$A \ominus B = \{(x, y) | (B)_{xy} \subseteq A\}$$

Physical meaning: remove boundary, remove unwanted small objects.



Dilation

A: Binary image
B: binary template, called structure element

Dilation: enlarging the foreground

A is dilated by B

$$A \oplus B = \{z | (B)_z \cap A \neq \emptyset\}$$

The intersection set between A and the translated B is not empty

Physical meaning

Dilation adopts the connected background pixels into the foreground, which extends its boundary and fill the holes in the foreground.

And whether "connected" is decided by the structure element.

Opening

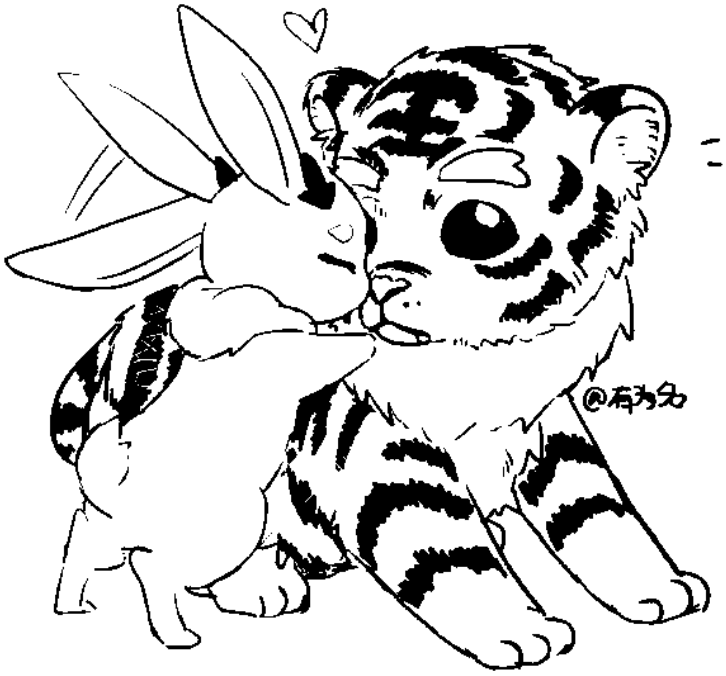
- Opening: erosion, then dilation
- $A \circ B = (A \ominus B) \oplus B$
- Erosion + Dilation = original binary image?

Closing

- Closing: Dilation, then erosion
- $A \bullet B = (A \oplus B) \ominus B$
- Dilation + Erosion = Erosion + Dilation?

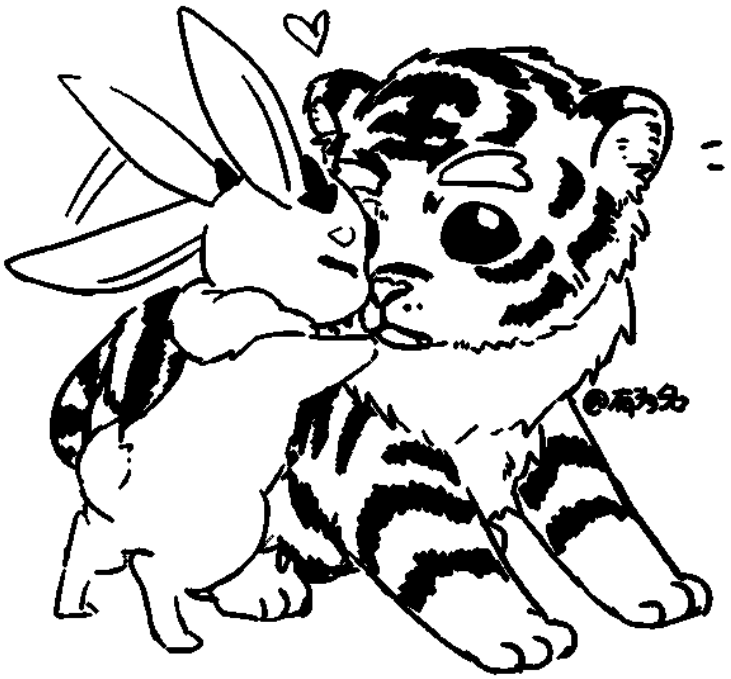
三、成果展示

原彩色图像



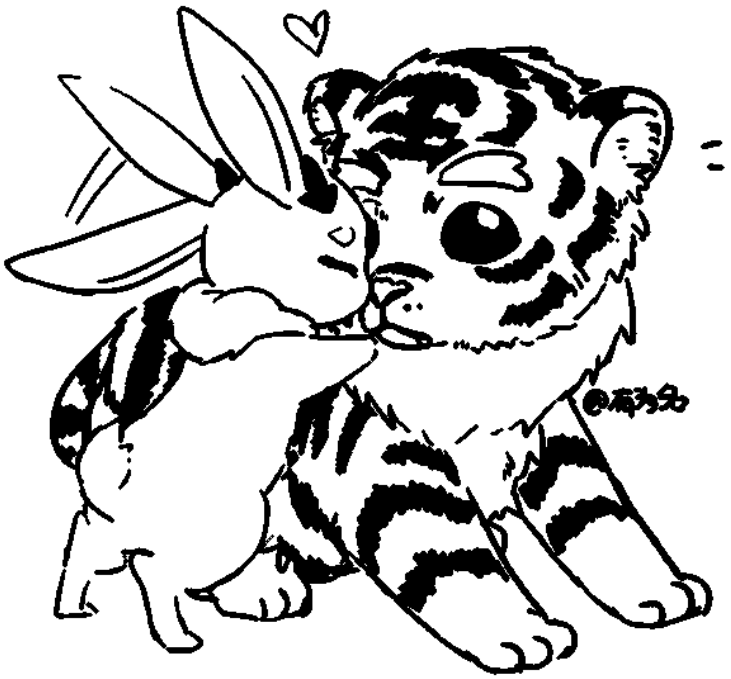
二值化之后的图像

Erosion 操作



Dilation 操作

Opening 操作



Closing 操作

四、源代码与分析

```
#include <stdio.h>
#include <assert.h>
#include <math.h>
#include <stdlib.h>

typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int DWORD;
typedef int LONG;


typedef struct  tagBITMAPFILEHEADER{
    WORD type;
    DWORD bfSize;
    WORD bfReserved1;
    WORD bfReserved2;
    DWORD bfOffBits;
}head1;
//定义第一个头

typedef struct tagBITMAPINFOHEADER{
    DWORD biSize;
    LONG biWidth;
    LONG biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD  biCompression;
    DWORD  biSizeImage;
    LONG  biXPelsPerMeter;
    LONG  biYPelsPerMeter;
    DWORD  biClrUsed;
    DWORD  biClrImportant;
}head2;
//定义第二个头

typedef struct _RGB{
    BYTE R;
    BYTE G;
    BYTE B;
}RGB;

typedef struct _YUV{
```

```
    short Y;
    short U;
    short V;
}YUV;
//YUV 格式可能会有负数，就直接用 short 存了

YUV RGB_To_YUV(RGB cur){
    YUV ret;
    ret.Y = round(0.299 * cur.R + 0.587 * cur.G + 0.114 * cur.B);
    ret.U = round(-0.147 * cur.R - 0.289 * cur.G + 0.435 * cur.B);
    ret.V = round(0.615 * cur.R - 0.515 * cur.G - 0.100 * cur.B);
    return ret;
}

BYTE In(short cur){
    if (cur > 255) cur = 255;
    if (cur < 0)   cur = 0;
    return (BYTE)cur;
}
//担心 YUV 转 RGB 时导致 RGB 范围出错，写一个框定范围的函数
RGB YUV_To_RGB(YUV cur){
    RGB ret;
    ret.R = In(round(cur.Y + 1.14 * cur.V));
    ret.G = In(round(cur.Y - 0.395 * cur.U - 0.581 * cur.V));
    ret.B = In(round(cur.Y + 2.033 * cur.U));
    return ret;
}

int line_byte, extra_byte, S;
head1 bmfh;
head2 bmih;

void IntoStream(RGB *cur, BYTE *p){
    for (int i = 0; i < S; i++){
        *p++ = cur->R;
        *p++ = cur->G;
        *p++ = cur->B;
        if ((i + 1) % bmih.biWidth == 0)
            for (int k = 0; k < extra_byte; k++)
                *p++ = 0;
        cur++;
    }
}
//将 RBG 数组导入到最后的输出流里
```

```
int Get_threshold(BYTE *cur, int S){
    int Min = cur[0], Max = cur[0];
    for (int i = 0; i < S; i++){
        if (cur[i] < Min) Min = cur[i];
        if (cur[i] > Max) Max = cur[i];
    }
    double best = -1;
    int ans = 0;
    for (int now = Min + 1; now <= Max; now++){
        double wf = 0, wb = 0;
        double avef = 0, aveb = 0;
        for (int i = 0; i < S; i++)
            if (cur[i] < now)
                avef += cur[i], wf++;
            else
                aveb += cur[i], wb++;
        avef /= wf; aveb /= wb;
        wf /= S; wb /= S;
        double ave = wf * avef + wb * aveb;
        double val = wf * wb * (avef - aveb) * (avef - aveb) ;
        if (val > best)
            best = val, ans = now;
    }
    return ans;
}
void Dilation(BYTE *cur, BYTE *New, int w, int h){
    const int matrix[3][3]={
        {0,1,0},
        {1,1,1},
        {0,1,0}
    };
    for (int i = 0; i < h; i++)
        for (int j = 0; j < w; j++){
            int ok = 0;
            for (int dx = -1; dx <= 1; dx++)
                for (int dy = -1; dy <= 1; dy++)
                    if (matrix[dx + 1][dy + 1]){
                        int x = i + dx, y = j + dy;
                        if (x >= 0 && x < h && y >= 0 && y < w && !cur[x * w + y])
                            ok = 1;
                    }
            New[i * w + j] = ok ? 0 : 255;
        }
}
```

```
void Erosion(BYTE *cur, BYTE *New, int w, int h){
    const int matrix[3][3]={
        {0,1,0},
        {1,1,1},
        {0,1,0}
    };
    for (int i = 0; i < h; i++)
        for (int j = 0; j < w; j++){
            int ok = 1;
            for (int dx = -1; dx <= 1; dx++)
                for (int dy = -1; dy <= 1; dy++)
                    if (matrix[dx + 1][dy + 1]){
                        int x = i + dx, y = j + dy;
                        if (x >= 0 && x < h && y >= 0 && y < w && cur[x * w + y])
                            ok = 0;
                    }
            New[i * w + j] = ok ? 0 : 255;
        }
}

void Opening(BYTE *cur, BYTE *New, int w, int h){
    BYTE *tmp = (BYTE *) malloc(w * h);
    Erosion(cur, tmp, w, h);
    Dilation(tmp, New, w, h);
}

void Closing(BYTE *cur, BYTE *New, int w, int h){
    BYTE *tmp = (BYTE *) malloc(w * h);
    Dilation(cur, tmp, w, h);
    Erosion(tmp, New, w, h);
}

int main(){
    FILE* fin = fopen("Origin.bmp", "rb"), * fout;   //读入文件

    fread(&bmfh, 14, 1, fin);
    fread(&bmih, sizeof(head2), 1, fin);

    line_byte = (bmih.biWidth * 3 + 3) / 4 * 4; //计算实际存储时每行的字节数
    extra_byte = line_byte - bmih.biWidth * 3;  //计算每行结尾空的字节数
    S = bmih.biWidth * bmih.biHeight;           //计算像素总个数
    int all =  line_byte * bmih.biHeight;       //计算像素矩阵总的字节数

    BYTE *Stream = (BYTE *) malloc(all);
```

```
fread(Stream, 1, all, fin);

RGB *Origin = (RGB*) malloc(S * sizeof(RGB));

BYTE *p = Stream;
RGB *cur = Origin;
for (int i = 0; i < S; i++){
    cur->R = *p++;
    cur->G = *p++;
    cur->B = *p++;
    if ((i + 1) % bmih.biWidth == 0)
        p = p + extra_byte;
    cur++;
}
//从读入流里获取实际的像素矩阵

YUV *New = (YUV *) malloc(S * 6);
BYTE *Gray = (BYTE *) malloc(S);
RGB *Print = (RGB *) malloc(S * 3);

for (int i = 0; i < S; i++){
    New[i] = RGB_To_YUV(Origin[i]);
    Gray[i] = New[i].Y;
}

int key = Get_threshold(Gray, S);

for (int i = 0; i < S; i++)
    Print[i].R = Print[i].G = Print[i].B = Gray[i] = Gray[i] < key ? 0 : 255;
IntoStream(Print, Stream);
fout = fopen("binaryzation.bmp", "wb");
fwrite(&bmfh, 14, 1, fout);
fwrite(&bmih, sizeof(head2), 1, fout);
fwrite(Stream, 1, all, fout);

BYTE *ret1 = (BYTE *) malloc(S);
Erosion(Gray, ret1, bmih.biWidth, bmih.biHeight);
for (int i = 0; i < S; i++)
    Print[i].R = Print[i].G = Print[i].B = ret1[i];
IntoStream(Print, Stream);
fout = fopen("binaryzation_erosion.bmp", "wb");
fwrite(&bmfh, 14, 1, fout);
fwrite(&bmih, sizeof(head2), 1, fout);
fwrite(Stream, 1, all, fout);
```

```
    BYTE *ret2 = (BYTE *) malloc(S);
    Dilation(Gray, ret2, bmih.biWidth, bmih.biHeight);
    for (int i = 0; i < S; i++)
        Print[i].R = Print[i].G = Print[i].B = ret2[i];
    IntoStream(Print, Stream);
    fout = fopen("binaryzation_dilation.bmp", "wb");
    fwrite(&bmfh, 14, 1, fout);
    fwrite(&bmih, sizeof(head2), 1, fout);
    fwrite(Stream, 1, all, fout);

    BYTE *ret3 = (BYTE *) malloc(S);
    Opening(Gray, ret3, bmih.biWidth, bmih.biHeight);
    for (int i = 0; i < S; i++)
        Print[i].R = Print[i].G = Print[i].B = ret3[i];
    IntoStream(Print, Stream);
    fout = fopen("binaryzation_opening.bmp", "wb");
    fwrite(&bmfh, 14, 1, fout);
    fwrite(&bmih, sizeof(head2), 1, fout);
    fwrite(Stream, 1, all, fout);

    BYTE *ret4 = (BYTE *) malloc(S);
    Closing(Gray, ret4, bmih.biWidth, bmih.biHeight);
    for (int i = 0; i < S; i++)
        Print[i].R = Print[i].G = Print[i].B = ret4[i];
    IntoStream(Print, Stream);
    fout = fopen("binaryzation_closing.bmp", "wb");
    fwrite(&bmfh, 14, 1, fout);
    fwrite(&bmih, sizeof(head2), 1, fout);
    fwrite(Stream, 1, all, fout);

    return 0;
}
```