

CS205 C/C++ Programming - Lab5 Assignment

Name: 宋明洋 (SONG Mingyang)

SID: 11811414

Part 1 - Analysis

这次试验目标以实现一个简单的CNN人脸识别神经网络为背景，综合了图像卷积，矩阵计算，opencv库的安装与使用等方面的内容，是一个综合性非常强的项目。由于涉及到了多次卷积的计算和结果的传递，这对我c++的类与对象的应用，c++指针的运用以及内存的管理都提出了更高的要求 and 锻炼。

Part 2 - Code

cnn.hpp

```
#pragma once
#include <iostream>
#include <arm_neon.h>
#include <opencv2/opencv.hpp>
#include "face_binary_cls.hpp"
#include "MyMatrix.hpp"

using namespace std;

class CNN
{
public:
    string FileName;
    size_t ImageRows;
    size_t ImageCols;
    size_t ImageChannels;
    MyMatrix * matrix = NULL;

    CNN(string file, size_t r, size_t c, size_t ch)
    {
        FileName = file;
        ImageRows = r;
        ImageCols = c;
        ImageChannels = ch;
        matrix = (MyMatrix*)malloc(sizeof(MyMatrix));
        matrix->num = (float*)malloc(sizeof(float)*ImageRows*ImageCols*ImageChannels);
        matrix->row = r;
        matrix->col = c;
        matrix->channel = ch;
    }
    ~CNN()
    {
    }
```

```

        delete []matrix->num;
    }

MyMatrix * setInput(MyMatrix * mt);
float * conv_relu(const float * in, conv_param & param, size_t row, size_t col);
float * maxPooling(float * in, size_t row, size_t col, size_t channel);
void execute();
};

```

cnn.cpp

```

#include "cnn.hpp"
#include <iostream>
#include "time.h"

using namespace std;

clock_t start, finish;

MyMatrix * CNN::setInput(MyMatrix * mt)
{
    if(mt == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The mt parameter is NULL.\n",
__FILE__, __LINE__, __FUNCTION__);
        return NULL;
    }
    cv::Mat InputImage = cv::imread(FileName);
    if(InputImage.data==nullptr)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The image does not exist!\n",
__FILE__, __LINE__, __FUNCTION__);
        return NULL;
    }else
    {
        float base = 255.0f;
        size_t step = mt->col*mt->row;
        float * mat_p = mt->num;

        if((InputImage.rows==ImageRows)&&(InputImage.cols==ImageCols))
        {
            for(size_t i=0; i<ImageRows; i++)
            {
                uchar * p = InputImage.ptr<uchar>(i);
                size_t var1 = i*ImageCols;
                #pragma omp parallel for
                for(size_t j=0; j<ImageCols; j++)
                {
                    mat_p[var1+j] = (float)p[3*j+1]/base;

```

```

        mat_p[step+var1+j] = (float)p[3*j+2]/base;
        mat_p[2*step+var1+j] = (float)p[3*j]/base;
    }
}
}else
{
    cv::resize(InputImage, InputImage, cv::Size(ImageRows, ImageCols));
    for(size_t i=0;i<ImageRows;i++)
    {
        uchar * p = InputImage.ptr<uchar>(i);
        size_t var1 = i*ImageCols;
        #pragma omp parallel for
        for(size_t j=0;j<ImageCols;j++)
        {
            mat_p[var1+j] = (float)p[3*j+1]/base;
            mat_p[step+var1+j] = (float)p[3*j+2]/base;
            mat_p[2*step+var1+j] = (float)p[3*j]/base;
        }
    }
}
}
return mt;
}

float * CNN::conv_relu(const float * in,conv_param & param,size_t row,size_t col)
{
    if(in == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The in parameter is NULL.\n",
__FILE__, __LINE__, __FUNCTION__);
        return NULL;
    }
    const size_t pad = param.pad;
    const size_t stride = param.stride;
    const size_t kernel_size = param.kernel_size;
    const size_t in_channels = param.in_channels;
    const size_t out_channels = param.out_channels;
    float * p_weight = param.p_weight;
    float * p_bias = param.p_bias;
    size_t length1 = kernel_size*kernel_size;
    size_t length2 = ((row+2*pad-kernel_size)/stride+1)*((col+2*pad-
kernel_size)/stride+1);
    MyMatrix * mtdata = NULL;
    MyMatrix * mtweights = NULL;
    float * out = new float[out_channels*length2]();
    float * temp_p = new float[in_channels*length2]();
    float * tempp = new float[length2]();
    mtweights = new MyMatrix(1,length1,1);
    mtdata = new MyMatrix(length1,length2,1);

```

```

size_t check = 0;
if(in_channels==16)
    check = 1;

if(pad == 0)
{
    for(size_t oc=0;oc<out_channels;oc++)
    {
        for(size_t ic=0;ic<in_channels;ic++)
        {
            // temp_p = out+oc*in_channels*length2+ic*length2;
            temp_p = temp_p+ic*length2;
            mtweights->num = p_weight+ic*length1+oc*length1*in_channels;
            //set the matrix of weights
            size_t k = 0;
            for(size_t i=1;i<=row-2;i+=stride)
            {
                for(size_t j=1;j<=col-2;j+=stride)
                {
                    size_t var1 = ic*row*col+i*col+j;
                    mtdata->num[k+length2*0] = in[var1-col-1];
                    mtdata->num[k+length2*1] = in[var1-col];
                    mtdata->num[k+length2*2] = in[var1-col+1];
                    mtdata->num[k+length2*3] = in[var1-1];
                    mtdata->num[k+length2*4] = in[var1];
                    mtdata->num[k+length2*5] = in[var1+1];
                    mtdata->num[k+length2*6] = in[var1+col-1];
                    mtdata->num[k+length2*7] = in[var1+col];
                    mtdata->num[k+length2*8] = in[var1+col+1];
                    k++;
                }
            }
            //set the matrix of data

            mul(mtweights,mtdata,temp_p,check);
        }

        for(size_t i=0;i<length2;i++)
        {
            for(size_t j=0;j<in_channels;j++)
            {
                out[oc*length2+i] += temp_p[j*length2+i];
            }
            out[oc*length2+i] += p_bias[oc];
        }
    }
    delete []temp_p;
    delete []mtdata->num;
}
else

```

```

{
    for(size_t oc=0;oc<out_channels;oc++)
    {
        for(size_t ic=0;ic<in_channels;ic++)
        {
            // temp_p = out+oc*in_channels*length2+ic*length2;
            temp_p = temp_p+ic*length2;
            // cout<<temp_p[ic*length2]<<endl;
            mtweights->num = p_weight+ic*length1+oc*length1*in_channels;
            size_t k = 0;
            for(size_t i=0;i<=row-1;i+=stride)
            {
                for(size_t j=0;j<=col-1;j+=stride)
                {
                    size_t var1 = ic*row*col+i*col+j;
                    mtdata->num[k+length2*0] = in[var1-col-1];
                    mtdata->num[k+length2*1] = in[var1-col];
                    mtdata->num[k+length2*2] = in[var1-col+1];
                    mtdata->num[k+length2*3] = in[var1-1];
                    mtdata->num[k+length2*4] = in[var1];
                    mtdata->num[k+length2*5] = in[var1+1];
                    mtdata->num[k+length2*6] = in[var1+col-1];
                    mtdata->num[k+length2*7] = in[var1+col];
                    mtdata->num[k+length2*8] = in[var1+col+1];
                    if(i==0)
                    {
                        mtdata->num[k+length2*0] = 0.0f;
                        mtdata->num[k+length2*1] = 0.0f;
                        mtdata->num[k+length2*2] = 0.0f;
                    }
                    if(i==row-1)
                    {
                        mtdata->num[k+length2*6] = 0.0f;
                        mtdata->num[k+length2*7] = 0.0f;
                        mtdata->num[k+length2*8] = 0.0f;
                    }
                    if(j==0)
                    {
                        mtdata->num[k+length2*0] = 0;
                        mtdata->num[k+length2*3] = 0;
                        mtdata->num[k+length2*6] = 0;
                    }
                    if(j==col-1)
                    {
                        mtdata->num[k+length2*2] = 0;
                        mtdata->num[k+length2*5] = 0;
                        mtdata->num[k+length2*8] = 0;
                    }
                }
            }
        }
    }
}

```

```

        k++;
    }
}
mul(mtweights, mtdata, tempp, check);
}

for(size_t i=0; i<length2; i++)
{
    for(size_t j=0; j<in_channels; j++)
    {
        out[oc*length2+i] += temp_p[j*length2+i];
    }
    out[oc*length2+i] += p_bias[oc];
}
}
delete []temp_p;
delete []mtdata->num;
}
float * po = out;
for(size_t i=0; i<out_channels*length2; i++)
{
    if(*po<0)
        *po = 0;
    po++;
}
return out;
}

float * CNN::maxPooling(float * in, size_t row, size_t col, size_t channel)
{
    if(in == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The in parameter is NULL.\n",
__FILE__, __LINE__, __FUNCTION__);
        return NULL;
    }
    float * output = new float[row/2*col/2*channel]();
    for(size_t c=0; c<channel; c++)
    {
        size_t var2 = c*row*col;
        for(size_t i=0; i<row; i+=2)
        {
            size_t var1 = i/2*col/2+c*row/2*col/2;
            for(size_t j=0; j<col; j+=2)
            {
                float var3=in[i*col+j+var2];
                if(var3<in[i*col+j+var2+1])
                    var3=in[i*col+j+var2+1];
                if(var3<in[i*col+j+var2+col])

```

```

        var3=in[i*col+j+var2+col];
        if(var3<in[i*col+j+var2+col+1])
            var3=in[i*col+j+var2+col+1];
        output[var1+j/2] = var3;
    }
}
}
return output;
}

void CNN::execute()
{
    MyMatrix * input = NULL;
    input = new MyMatrix(128,128,3);
    input = setInput(input);
    float * c1 = NULL;
    c1 = conv_relu(input->num,conv_params[0],128,128);
    float * m1 = NULL;
    m1 = maxPooling(c1,64,64,16);
    float * c2 = NULL;
    c2 = conv_relu(m1,conv_params[1],32,32);
    float * m2 = NULL;
    m2 = maxPooling(c2,30,30,32);
    float * c3 = NULL;
    c3 = conv_relu(m2,conv_params[2],16,16);

    float * result = NULL;
    result = new float[2]();
    for (size_t i=0;i<2048;i++)
    {
        result[0] += (fc_params[0].p_weight[i]*c3[i]);
        result[1] += (fc_params[0].p_weight[i + 2048]*c3[i]);
    }
    result[0] += fc_params[0].p_bias[0];
    result[1] += fc_params[0].p_bias[1];
    float background = 0;
    background = exp(result[0]) / (exp(result[0]) + exp(result[1]));
    float face = 0;
    face = exp(result[1]) / (exp(result[0]) + exp(result[1]));
    cout<<"Judging "<<FileName<<endl;
    cout << "background probability: " << background << endl;
    cout << "face probability: " << face << endl;
}

int main()
{
    double duration = 0;
    start = clock();
    CNN p5("../10.jpg",128,128,3);

```

```

    p5.execute();
    finish = clock();
    duration = (double)(finish - start) / CLOCKS_PER_SEC * 1000;
    cout<<"time consumption: "<<duration<<"ms"<<endl;
}

```

MyMatrix.hpp

```

#pragma once
#include <iostream>

using namespace std;

class MyMatrix
{
public:
    size_t row;
    size_t col;
    size_t * ref_count;
    size_t channel;
    float * num;

    MyMatrix()
    {
        // cout<<"a smy matrix is created"<<endl;
        ref_count = (size_t*)malloc(sizeof(size_t));
        *ref_count = 0;
    }

    MyMatrix(size_t r, size_t c, size_t ch)
    {
        // cout<<"a smy matrix is created"<<endl;
        ref_count = (size_t*)malloc(sizeof(size_t));
        *ref_count = 0;
        row = r;
        col = c;
        channel = ch;
        num = new float[row*col*channel]();
    }

    ~MyMatrix()
    {
        if(*ref_count==0)
        {
            delete []num;
            cout<<"a matrix is freed"<<endl;
        }
        *ref_count = *ref_count - 1;
    }

    // MyMatrix* operator=(const MyMatrix * mat);

```



```
};

bool add(MyMatrix * mat1, MyMatrix * mat2, MyMatrix * outcome);
bool mul(MyMatrix * mat1, MyMatrix * mat2, float * outcome, size_t check);
```

MyMatrix.cpp

```
#include "MyMatrix.hpp"
#include <iostream>

using namespace std;

// MyMatrix* MyMatrix::operator=(const MyMatrix * mat)
// {
//     cout<<"MyMatrix& MyMatrix::operator=(const MyMatrix & mat)"<<endl;
//     if(this == mat)
//         return this;
//     this->row = mat->row;
//     this->col = mat->col;
//     this->channel = mat->channel;
//     this->num = mat->num;
//     this->ref_count = mat->ref_count;
//     *mat->ref_count = *mat->ref_count + 1;
//     return this;
// }

bool add(MyMatrix * mat1, MyMatrix * mat2, MyMatrix * outcome)
{
    if(mat1 == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The mat1 parameter is
NULL.\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if(mat2 == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The mat2 parameter is
NULL.\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if(outcome == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The outcome parameter is
NULL.\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if((mat1->channel!=mat2->channel) || (mat2->channel!=outcome->channel))
    {
```

```

        fprintf(stderr, "File %s, Line %d, Function %s(): The channel of inputs does
not match.\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if((mat1->row!=mat2->row) || (mat1->col!=mat2->col))
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The dimension of mat1 and
mat2 does not match!\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if((outcome->row!=mat1->row) || (outcome->col!=mat1->col))
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The dimension of mat1 and
outcome does not match!\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    size_t length = mat1->row*mat1->col*mat1->channel;
    float * p1 = mat1->num;
    float * p2 = mat2->num;
    float * po = outcome->num;
    for(size_t i=0;i<length;i++)
    {
        *(po++) = *(p1++) + *(p2++);
    }
    return true;
}

bool mul(MyMatrix * mat1, MyMatrix * mat2, float * outcome,size_t check)
{
    if(mat1 == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The mat1 parameter is
NULL.\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if(mat2 == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The mat2 parameter is
NULL.\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if(outcome == NULL)
    {
        fprintf(stderr, "File %s, Line %d, Function %s(): The outcome parameter is
NULL.\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    if((mat1->col!=mat2->row))
    {

```

```











        cout<<mat1->col<<" "<<mat2->row<<endl;
        fprintf(stderr, "File %s, Line %d, Function %s(): The dimension of mat1 and
mat2 does not match!\n", __FILE__, __LINE__, __FUNCTION__);
        return false;
    }
    float * p1 = mat1->num;
    float * p2 = mat2->num;
    float * po = outcome;
    size_t rows = mat1->row;
    size_t cols = mat2->col;
    size_t channels = mat1->channel;
    float var = 0;
    for(size_t c=0;c<channels;c++)
    {
        for(size_t i=0;i<rows;i++)
        {
            for(size_t j=0;j<cols;j++)
            {
                for(size_t k=0;k<mat1->col;k++)
                {
                    po[i*cols+j]+=(p1[i*cols+k]*p2[k*cols+j]);
                    // if(check == 1)
                    // {
                    //     cout<<p1[i*cols+k]<<" "<<p2[k*cols+j]<<endl;
                    // }
                }
            }
        }
    }
    return true;
}

```

Part 3 - Result & Verification

测试分别基于arm的Apple M1 Pro和基于x86的Intel i7-8750H平台。识别的结果以及用时如下表展示：

测试数据及验证

文件名	图片	尺寸	face probability	Arm平台用时	测试结果	x86平台用时	测试结果
1.jpg		128×128	1	6.467ms	Apple M1 Pro Judging ../1.jpg background probability: 1.23781e-09 face probability: 1 time consumption: 6.467ms	11.983ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../1.jpg background probability: 1.23781e-09 face probability: 1 time consumption: 11.983ms
2.jpg		128×128	0.999972	6.144ms	Apple M1 Pro Judging ../2.jpg background probability: 2.80978e-05 face probability: 0.999972 time consumption: 6.144ms	14.624ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../2.jpg background probability: 2.80978e-05 face probability: 0.999972 time consumption: 11.865ms
3.jpg		128×128	0.94101	6.281ms	Apple M1 Pro Judging ../3.jpg background probability: 0.0589899 face probability: 0.94101 time consumption: 6.281ms	10.852ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../3.jpg background probability: 0.0589899 face probability: 0.94101 time consumption: 10.852ms
4.jpg		128×128	0	6.176ms	Apple M1 Pro Judging ../4.jpg background probability: 1 face probability: 4.72238e-42 time consumption: 6.176ms	12.808ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../4.jpg background probability: 1 face probability: 4.72238e-42 time consumption: 12.808ms
5.jpg		116×110	0.160129	6.136ms	Apple M1 Pro Judging ../5.jpg background probability: 0.839871 face probability: 0.160129 time consumption: 6.136ms	15.23ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../5.jpg background probability: 0.839871 face probability: 0.160129 time consumption: 15.23ms
6.jpg		274×276	0.0015981	6.83ms	Apple M1 Pro Judging ../6.jpg background probability: 0.998402 face probability: 0.0015981 time consumption: 6.83ms	16.371ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../6.jpg background probability: 0.998402 face probability: 0.0015981 time consumption: 16.371ms
7.jpg		138×134	0.989591	6.874ms	Apple M1 Pro Judging ../7.jpg background probability: 0.0104095 face probability: 0.989591 time consumption: 6.874ms	15.609ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../7.jpg background probability: 0.0104095 face probability: 0.989591 time consumption: 15.609ms
8.jpg		116×110	0.00106156	6.329ms	Apple M1 Pro Judging ../8.jpg background probability: 0.998938 face probability: 0.00106156 time consumption: 6.329ms	14.058ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../8.jpg background probability: 0.998938 face probability: 0.00106156 time consumption: 14.058ms
9.jpg		146×150	0.930276	6.737ms	Apple M1 Pro Judging ../9.jpg background probability: 0.0697236 face probability: 0.930276 time consumption: 6.737ms	15.735ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../9.jpg background probability: 0.0697236 face probability: 0.930276 time consumption: 15.735ms
10.jpg		134×136	0.99857	6.554ms	Apple M1 Pro Judging ../10.jpg background probability: 0.00143004 face probability: 0.99857 time consumption: 6.554ms	14.563ms	Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz Judging ../10.jpg background probability: 0.00143004 face probability: 0.99857 time consumption: 14.563ms

可以看出，在10组测试对图片中，arm平台的处理速度均显著快于x86平台，这主要得益于Apple M1 Pro更为先进的制程和架构。

数据集中的人脸图片，全部得到了正确的判断。并且其中4，6，8为背景图片，它们的face score都趋近于0。但是在一些拟人的图像，例如5，10中。该神经网络的face score有一定的可能性误判。

Part 4 - Difficulties & Solutions

- 在利用opencv进行图片的读取时，我增加了图片大小调整的功能，当输入图片的大小不是128*128时，可以将图片转换为我们需要的大小来进行检测。在读取图片是，我通过强制的类型转换，将cv读取的uchar类型的数据转换成为了float类型，方便之后矩阵运算的进行。
- 为了提升图片读取的速度，我使用了openmp多线程任务来加速opencv的图像读取：

```
#pragma omp parallel for
for(size_t j=0;j<ImageCols;j++)
{
    mat_p[var1+j] = (float)p[3*j+1]/base;
    mat_p[step+var1+j] = (float)p[3*j+2]/base;
    mat_p[2*step+var1+j] = (float)p[3*j]/base;
}
```

- 为了方便debug，我在程序中加入了参数检查的判断，并且利用stderr输出错误信息，并告知我错误的位置和方法名称。这一定程度上方便了我之后的debug工作。

```

if(mat1 == NULL)
{
    fprintf(stderr, "File %s, Line %d, Function %s(): The mat1 parameter is
NULL.\n", __FILE__, __LINE__, __FUNCTION__);
    return false;
}

```

4. 在本次实验中，我将卷积运算转换成了相应的矩阵乘法计算，这使得我可以将之前项目的MyMatrix类移植到本实验中。项目中我将3x3的卷积核转化成了1x9对矩阵mat1，并将输入需要进行卷积的数据集转化为9x（单个channel内需要进行的卷积运算的个数）的mat2，即事先将需要进行卷积的数据储存在矩阵内，通过mat1与mat2相乘，即实现了卷积到矩阵乘法的转换。该过程在程序中对具体实现如下：

```

for(size_t ic=0;ic<in_channels;ic++)
{
    // temp_p = out+oc*in_channels*length2+ic*length2;
    temp_p = out+oc*in_channels*length2+ic*length2;
    temp_p = temp_p+ic*length2;
    mtweights->num = p_weight+ic*length1+oc*length1*in_channels;
    //set the matrix of weights
    size_t k = 0;
    for(size_t i=1;i<=row-2;i+=stride)
    {
        for(size_t j=1;j<=col-2;j+=stride)
        {
            size_t var1 = ic*row*col+i*col+j;
            mtdata->num[k+length2*0] = in[var1-col-1];
            mtdata->num[k+length2*1] = in[var1-col];
            mtdata->num[k+length2*2] = in[var1-col+1];
            mtdata->num[k+length2*3] = in[var1-1];
            mtdata->num[k+length2*4] = in[var1];
            mtdata->num[k+length2*5] = in[var1+1];
            mtdata->num[k+length2*6] = in[var1+col-1];
            mtdata->num[k+length2*7] = in[var1+col];
            mtdata->num[k+length2*8] = in[var1+col+1];
            k++;
        }
    }
    //set the matrix of data

    mul(mtweights,mtdata,temp_p,check);
}

```

5. 在进行数据传递时，我使用了float类型的指针进行数据的传递，这一定程度上防止了内存的硬拷贝，并且我通过及时对不再需要对内存进行释放，提升了内存的利用效率，减少了内存的浪费。
6. 程序的编译优化

```
michael@songmingyangdeMacBook-Pro build % cmake -DCMAKE_BUILD_TYPE=Release ..  
  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /Users/michael/OneDrive/codes/c&cpp/project/p5/build  
michael@songmingyangdeMacBook-Pro build % make  
Consolidate compiler generated dependencies of target p5  
[ 33%] Building CXX object CMakeFiles/p5.dir/cnn.cpp.o  
[ 66%] Building CXX object CMakeFiles/p5.dir/MyMatrix.cpp.o  
[100%] Linking CXX executable p5  
[100%] Built target p5  
michael@songmingyangdeMacBook-Pro build % sysctl -n machdep.cpu.brand_string && ./p5  
Apple M1 Pro  
Judging ../1.jpg  
background probability: 1.23781e-09  
face probability: 1  
time consumption: 5.936ms
```

我利用cmake管理程序的源码，在编译时，我将cmake build type设置成了Release模式，这可以大大提升代码的执行速度。