CS205 C/C++ Program Design – Project 2.

Name : 谢岳臻 SID : 11913008

> 我本次project 2的选题是做一个CNN的前向网络
>
> https://github.com/Unnamed-1408/Cpp_Project.git

Content

1. Analysis & Code

一. Test模块

顶层函数为Test，有main函数在内，负责调动一切的其他模块。

其功能有读入照片，使用opencv读取图片，读取成功则进行余下步骤，失败则程序停止。以及将对应的图片的RGB的[0-255]的值转化为[0-1]的值并存储入对应矩阵，将代表三原色的矩阵传入Start_CNN模块（在另外一个Fast_CNN/CNN.cpp文件）。

注：矩阵的height以及weight均加了2, 即表示在矩阵四周加了一圈padding，使得卷积以及maxpooling的时候更加方便

```
#include <iostream>
#include <opencv4/opencv2/opencv.hpp>
#include <sys/time.h>
#include "Fast_CNN.cpp"

using namespace std;
using namespace cv;

int main()
{
    Mat src;
    src = imread("T.jpg");    //图像加载
    if (src.empty()) {
        cout << "could not load image..." << endl;
        return -1;
    }
    else cout << "load succsessful." << endl;
    imshow("input", src);
    Mat rst;
    float xradio = (float)128 / (float)src.rows;
    float yradio = (float)128 / (float)src.cols;
    resize(src, rst, Size(), xradio, yradio);
    int height = rst.rows;
    int width = rst.cols;
    float* blue = new float[(height+2) * (width+2)]();
    float* green = new float[(height+2) * (width+2)]();
    float* red = new float[(height+2) * (width+2)]();

    for (int row = 0; row < height; row++) {
        for (int col = 0; col < width; col++) {
            float   b = src.at<Vec3b>(row, col)[0];  //读取通道值
            float   g = src.at<Vec3b>(row, col)[1];
```

```
            float   r = src.at<Vec3b>(row, col)[2];
            blue[(row+1) * (width+2) + col + 1] = b / 256.0f;
            green[(row+1) * (width+2) + col + 1] = g / 256.0f;
            red[(row+1) * (width+2) + col + 1] = r / 256.0f;
        }
    }
    auto start = std::chrono::steady_clock::now();
//start
    Start_CNN(blue, green, red, 128);
//end
    auto end = std::chrono::steady_clock::now();
    cout << "cost time : " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() <<
"ms" << endl;
    delete[]blue;
    delete[]green;
    delete[]red;
    waitKey(0);
    return 0;
}
```

二.CNN模块

此cpp文件中包含卷积，RELU，Maxpooling，Flatten全部CNN操作

- 结构体T

```
struct T{
    int size;
    int WNL;
    float **List;
};
```

结构体T代表一个layer所代表的的所有channel，其中的成员变量size代表这一层有几个 卷积核，WNL代表这一层的输出矩阵的行列数，List为二维指针，指向对应卷积核卷积出 的矩阵。将其设置为一个结构体主要是数据较为紧凑，用起来的时候较为直观。

- ConvBNReLU函数

  此函数将Conv以及ReLU合在一起,此函数为最核心的卷积函数，函数较长不展示。可能注意到了有多个重复的过程qaq，分别对应着第三层卷积的函数（为什么单独分出以后会解释），前两层的strike = 2的情况，以及strike = 1的情况，分开的原因仅是因为循环条件有个为小于等于一个是小于，因此造成了有三个情况分别对应着三层卷积。因为情况较为复杂，且没有时间将其分离，就只将其勉强跑起来。

  参数解释：layer 层数　fliter_num 那一层的第几个卷积核　In_channel 传入的通道

  在函数的最后顺便将bias以及ReLU一起做了可以提高效率。

  此函数卷积的时候采用的最直接的方法，直接相乘。

- MaxPooling函数

```
 float *Storage = new float[(In_channel->WNL/2 + 2) * (In_channel->WNL/2 +
2)]();
        int x_ptr = 1;
        int y_ptr = 1;
        for(y_ptr = 1;y_ptr <= In_channel->WNL; y_ptr += 2){
            for(x_ptr = 1; x_ptr <= In_channel->WNL; x_ptr += 2){
                Storage[1 + x_ptr/2 + (1 + y_ptr/2) * (In_channel->WNL/2 +
2)] = max(In_channel->List[num][x_ptr + y_ptr * (In_channel->WNL +
2)],In_channel->List[num][x_ptr + 1 + y_ptr * (In_channel->WNL + 2)]);
                Storage[1 + x_ptr/2 + (1 + y_ptr/2) * (In_channel->WNL/2 +
2)] = max(Storage[1 + x_ptr/2 + (1 + y_ptr/2) * (In_channel->WNL/2 +
2)],In_channel->List[num][x_ptr + 1 + (y_ptr + 1) * (In_channel->WNL + 2)]);
                Storage[1 + x_ptr/2 + (1 + y_ptr/2) * (In_channel->WNL/2 +
2)] = max(Storage[1 + x_ptr/2 + (1 + y_ptr/2) * (In_channel->WNL/2 +
2)],In_channel->List[num][x_ptr + (y_ptr + 1) * (In_channel->WNL + 2)]);
            }
        }
        return Storage;
```

池化函数，考虑到我额外加了一个pad，因此x和y从(1,1)开始进行MaxPooling

- Flatten函数

```
float *Flatten(T *In_channel){
    float *Large = new float[In_channel->size * In_channel->WNL * In_channel-
>WNL]();
    int Large_ptr = 0;
    for(int m = 0; m < In_channel->size; m++){
        for(int i = 1; i <= In_channel->WNL; i++){
            int tmp = i * (In_channel->WNL + 2);
            for(int j = 1; j <= In_channel->WNL; j++){
                Large[Large_ptr] = In_channel->List[m][tmp + j];
                Large_ptr++;
            }
        }
    }
    return Large;
}
```

Flatten函数不能直接将二维指针转化成一维指针将其平铺，但是因为有额外的pad因此需要将其分离开，所以要额外写一个函数实现。

- dot_product

最简单的点乘，尝试过openmp优化但是没加锁导致结果错乱，加了锁导致速度变慢，在极低数据量下甚至比单线程还慢，因此直接单线程。

```
![sendpix1](/home/bill/桌面/sendpix1.jpg)float *out = new float[2]();
    for(int i = 0; i < 2; i++){
        int tmp = i * 2048;
        for(int j = 0; j < 2048; j++){
            out[i] += Large[j] * fc_params->p_weight[tmp + j];
        }
        out[i] += fc_params->p_bias[i];
    }
    return out;
```

- Start_CNN主调用函数

```
void Start_CNN(float *B, float* G, float *R, int size){
    T *start = new T;
    start->size = 3;
    start->List = new float*[3];
    start->List[0] = B;
    start->List[1] = G;
    start->List[2] = R;
    start->WNL = size;

    T *first = new T;
    T *second = new T;
    T *third = new T;
    first->size = 16;
    first->List = new float*[16];
    first->WNL = 64;
    for(int i = 0; i < 16; i++){
        first->List[i] = ConvBNReLU(0, i, start);
    }


    for(int i = 0; i < 16; i++){
        float *tmp = MaxPooling(first, i, 1);
        delete first->List[i];
        first->List[i] = tmp;
    }
    first->WNL = 32;

    second->WNL = 32;
    second->size = 32;
    second->List = new float*[32]();
    for(int i = 0; i < 32; i++){
        second->List[i] = ConvBNReLU(1, i, first);
    }

    for(int i = 0; i < 32; i++){
        float *tmp = MaxPooling(second, i, 2);
        delete second->List[i];
        second->List[i] = tmp;
    }
    second->WNL = 16;

    third->size = 32;
    third->List = new float*[32]();
    third->WNL = 16;
    for(int i = 0; i < 32; i++){
        third->List[i] = ConvBNReLU(2, i, second);
    }
    third->WNL = 8;

    float *end = Flatten(third);

    float *to_out = dot_product(end);
    cout << "background : " <<(exp(to_out[0])/(exp(to_out[0]) +
exp(to_out[1]))) << endl;
```

```
    cout << "face : " << (exp(to_out[1])/(exp(to_out[0]) + exp(to_out[1])))
<< endl;
}
```

传入参数BGR代表Blue,Green,Red矩阵。start,first,second,third为channel中各个参数。可以结合流程图，此函数表达过程与流程图一致，在最后的时候将其flatten，并进行dotproduct，在dot_product的时候顺便加上了bias。最后调用 $$p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$ SoftMax函数得出最后答案。

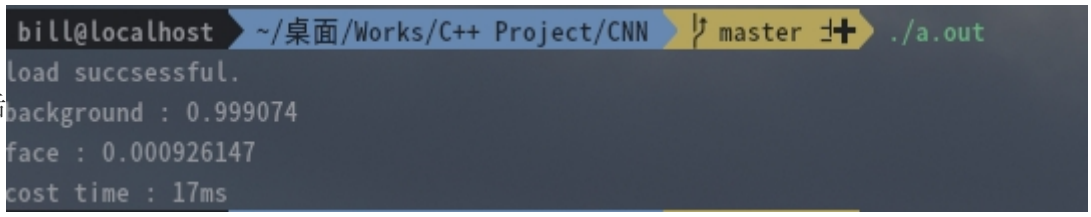
三.优化尝试

尝试使用过添加过预编译指令 unroll 以及 openmp，发现提升不明显，甚至会在莫名其妙的地方会有bug的出现。如unroll的时候处理边界条件，可能会造成越界错误。openmp在最大4000左右的数据量的时候无法发挥加快作用。

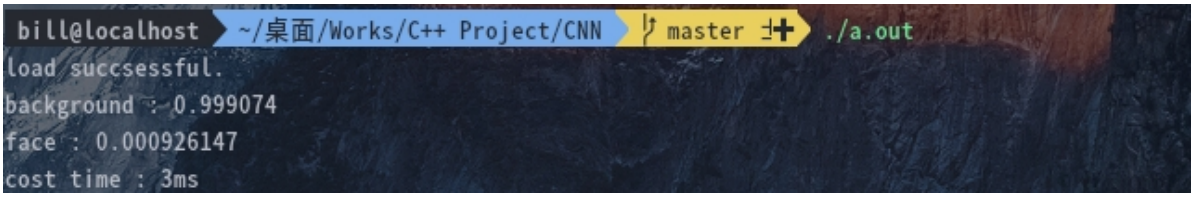
因此最后仅采用了减少乘法的计算以及开启O3的方式加快速度

未优化前

优化后



开启O3



提升效果不明显，因为计算量过小。

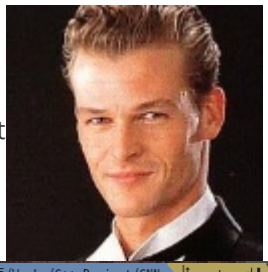
减少计算量

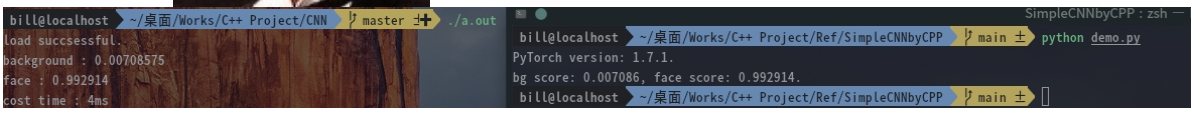
```
fliter = conv_params[layer].p_weight + fliter_num * In_channel->size * 9;
float tmp = 0;
int tmp_a = (y_ptr-1) * (In_channel->WNL+2) + x_ptr;
int tmp_b = (y_ptr) * (In_channel->WNL+2) + x_ptr;
int tmp_c = (y_ptr+1) * (In_channel->WNL+2) + x_ptr;
for(int channel_ptr = 0; channel_ptr < In_channel->size; channel_ptr++){
    tmp += fliter[0] * In_channel->List[channel_ptr][tmp_a - 1];
    tmp += fliter[1] * In_channel->List[channel_ptr][tmp_a];
    tmp += fliter[2] * In_channel->List[channel_ptr][tmp_a + 1];
    tmp += fliter[3] * In_channel->List[channel_ptr][tmp_b - 1];
    tmp += fliter[4] * In_channel->List[channel_ptr][tmp_b];
    tmp += fliter[5] * In_channel->List[channel_ptr][tmp_b + 1];
    tmp += fliter[6] * In_channel->List[channel_ptr][tmp_c - 1];
    tmp += fliter[7] * In_channel->List[channel_ptr][tmp_c];
    tmp += fliter[8] * In_channel->List[channel_ptr][tmp_c + 1];
    fliter = fliter + 9;
}
```

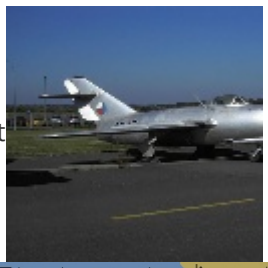
仅会提高5ms左右

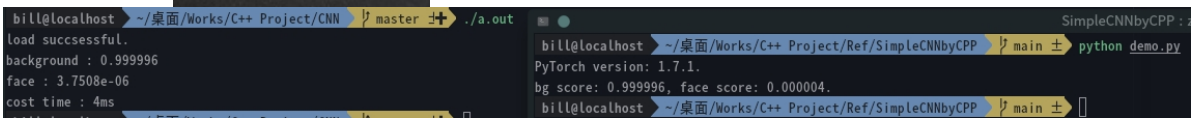
四. CNN Test

左边为图片，右边一为C++跑出来的，右二为SampleCNN中python跑出的结果，两个形成对比，可以看C++写的相对精确度，其数值与sample的python跑出来高度重合。

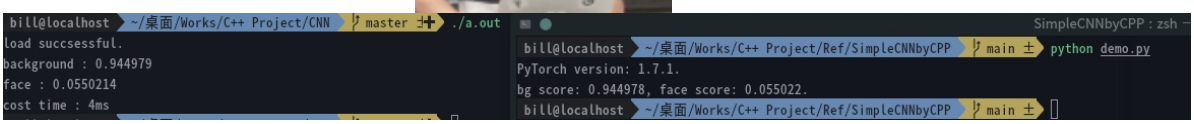
1.Sample Test 



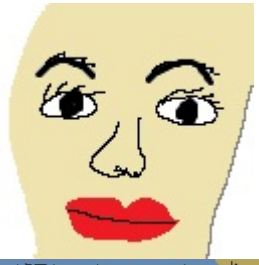
2.Sample Test 

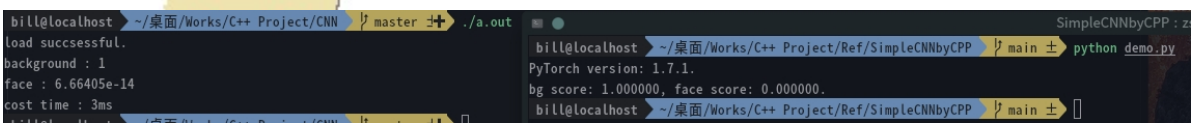


3.很难解释（发现检测不出人脸来） 

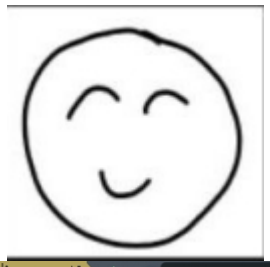


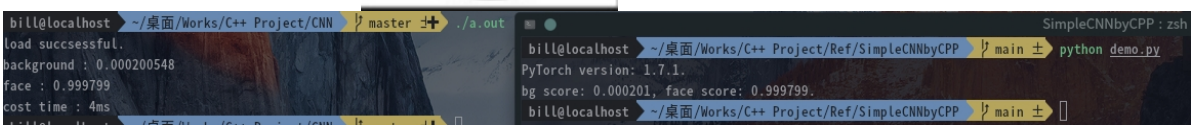
4.背景Test



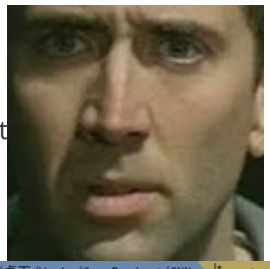


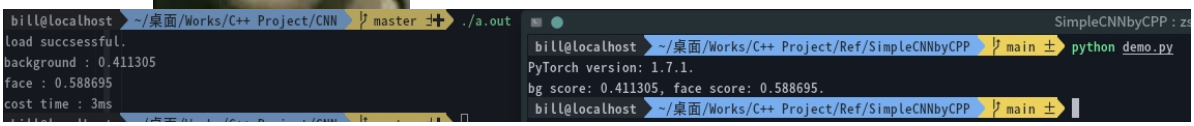
5.很难解释（face的分数过高）



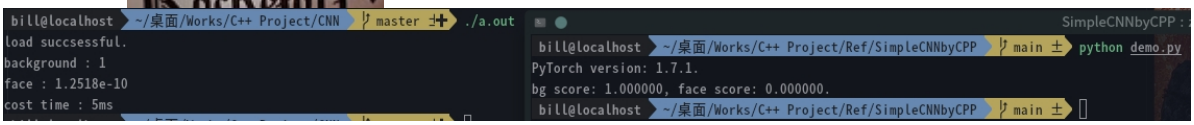


6.半人脸Test

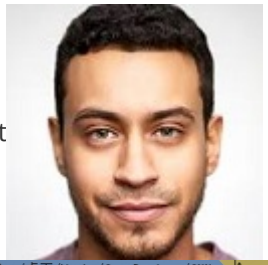


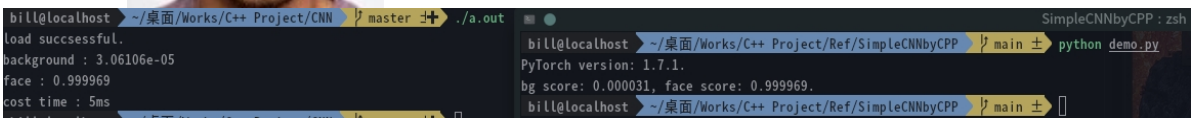


7.背景Test



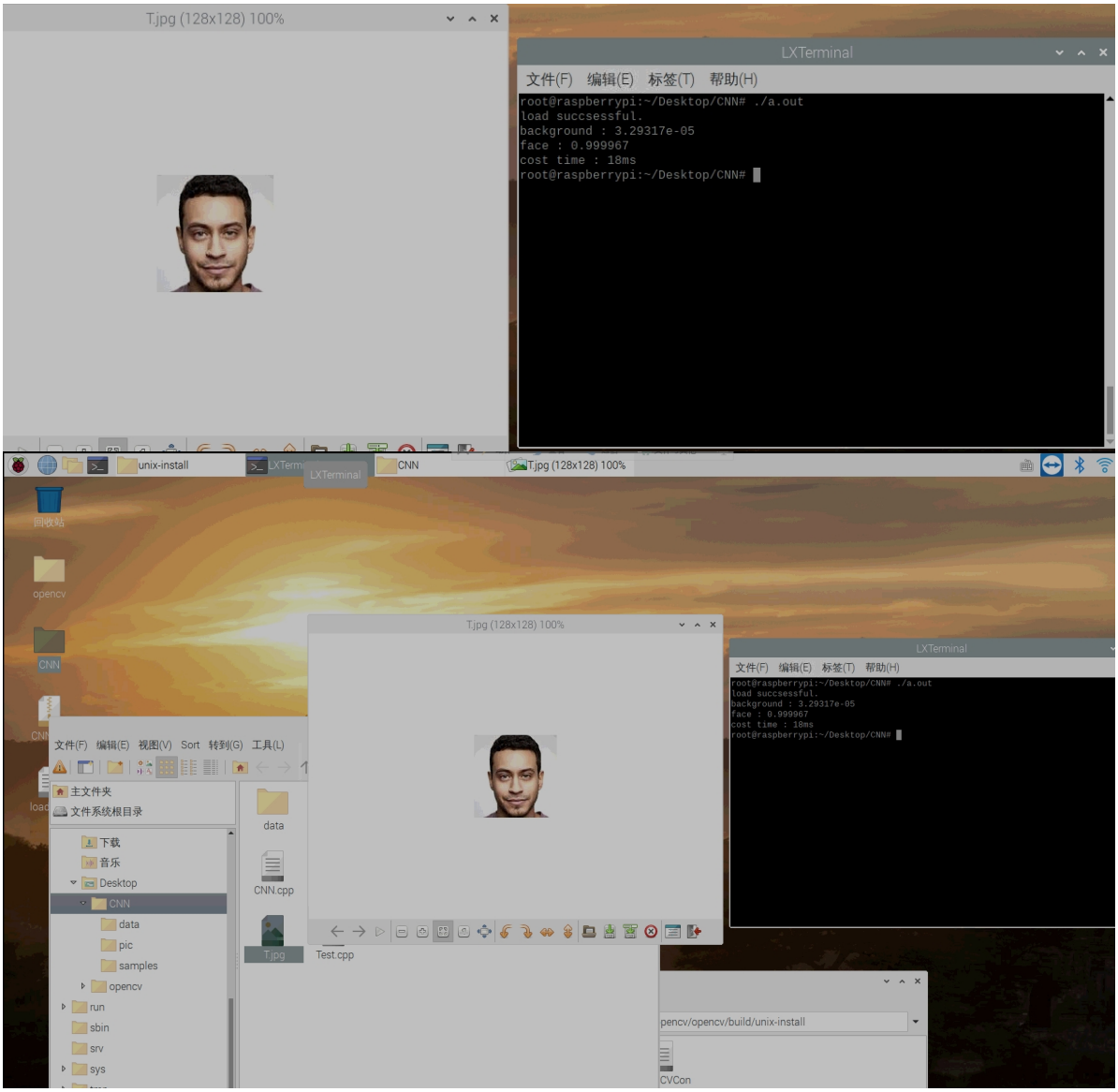
8.人脸Test





五. ARM Test

同上侧最后一个例子一样



因为在cmake的时候忘记将GTK的选项加进去，导致无法弹出显示图片的框，因此只能将其注释掉直接将图片打开展示。

（使用远程连接树梅派，310手动编译速度过慢，故换用树梅派）

六. Difficulties & Solutions, or others

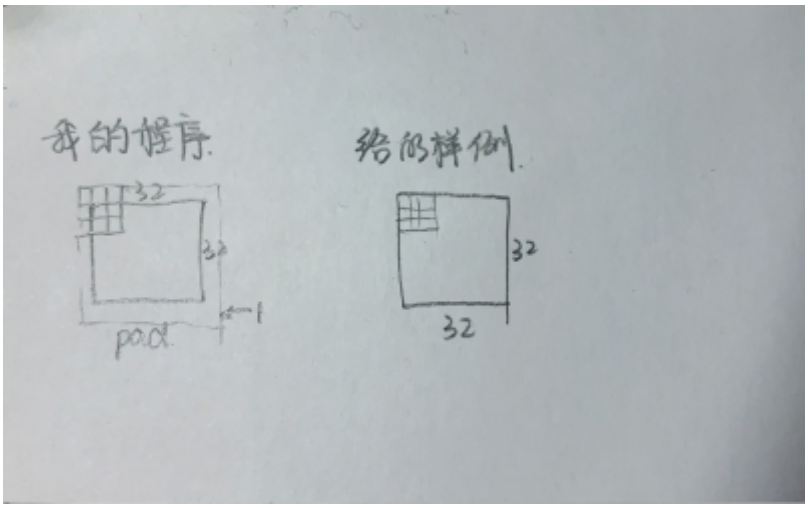
1. linux系统下，从仓库直接安装的时候发现头文件缺失并且无法链接到动态库，因此选择手动编译，并且用pkg进行管理动态链接。但是在include头文件的时候发现，虽然可以include进但是无法使用其中任何一个成员函数。在include的时候用#include <opencv4/opencv2/opencv.hpp>，最后进入头文件的时候发现里面是有个#include "opencv2/*..."并且我的系统path并未include入opencv4这个文件夹。

   ==>解决办法是将opencv2移出opencv4，在include的时候用#include <opencv2/opencv.hpp>解决问题，但是仍然不明白为什么要多一个opencv4的文件夹

2. 跑出来的结果不正确，并且差距较大

   ==>解决办法是对sample的python程序进行修改，使其print出一些中间过程的矩阵，一步一步对照

3. （Conv函数写了三个近乎相同的代码块原因）在对照中间矩阵的时候，发现在第二层卷积的时候本该是卷积出来的32 -> 32，变成了32 ->30，并且对照了我算出来的数值，发现我的数值均有一格的偏移，最后推测是卷积起始位置不同，如下图



发现问题后想要在尽可能小的改动上将其修复

==>解决方法就是将这个特殊情况拎出来分类讨论，因此导致了程序较为冗长。并且这个造成了一些后遗症，比如说多出的pad导致在maxpooling的时候有了一些偏移，必须为这个特例设置特别的参数才可以正常运行

4. 忘记了SoftMax，debug了半个小时未发现与标准有什么出入

   ==>在最后添加最后一步

5. 在ARM上在自己主机上的问题又遇到了一次，定义环境变量的时候PKG_CONFIG_PATH=...发现无法作用

   ==>必须定义全局环境变量用export才可以使用pkg