CS205 C/ C++ Programming – Project 2

Name: 姜润智（JIANG Runzhi）

SID: 11810112

Part 1 – Analysis

> 本次project内容为使用C/C++实现卷积神经网络（convolutional neural network (CNN)），包含3个卷积层和1个全连接层。实现预测输入图片是为人脸还是背景。
>
> **Requirement 1（20 points）：** 实现3x3大小的卷积核进行卷积的卷积层，支持步长为1或步长为2（stride=1 and stride=2），且padding=1。
>
> **Requirement 2（30 points）：** 程序能够正确输出对图片的confidence scores，即对于图片上是否存在人脸的确信程度（介于0和1之间）。
>
> **Requirement 3（20 points）：**优化程序的实现。
>
> **Requirement 4（5 points）：**在X86和ARM平台上都对程序进行测试，且能在两平台上输出相同结果。
>
> **Requirement 5（5 points）：**将源代码放在GitHub.com上。
>
> **Requirement 6（20 points）：**合理组织报告，且不超过15页，字号应为10号左右。

Part 2 – Fulfillment

实现步骤：

1. 输入一张128*128大小的彩色图片，使用OpenCV读取照片每个位置的数据（数据范围[0,255]）放入矩阵中。
2. 将数据全部除以255.0进行标准化，转换为数据范围[0,1]的float类型浮点数。
3. 经过卷积（与卷积核进行乘法运算），Batch Normalization(本次已包含在卷积操作内), ReLU(将小于0的数据统一设置为0)， MaxPool（最大值池化）等操作，共进行三次卷积，两次池化。
4. 最终进行一次全连接操作（Full connection）得到我们需要的分别代表"背景"和"人脸"的两个数值，再通过 softmax把这两个数值进行标准化，得到confidence scores。

```
//128 -> 64 channel: 3 -> 16
float* out_mat1 = convBNReLU(in_mat, 128, 128 ,conv_params[0]);

//64 -> 32 channel: 16 -> 16
float* out_mat2 = maxPool(out_mat1, 16, 64, 64);

//32 -> 30 channel: 16 -> 32
float* out_mat3 = convBNReLU(out_mat2, 32, 32, conv_params[1]);

//30 -> 15 channel: 32 -> 32
float* out_mat4 = maxPool(out_mat3, 32, 30, 30);

//15 -> 8 channel: 32 -> 32
float* out_mat5 = convBNReLU(out_mat4, 15, 15, conv_params[2]);
```

fig.2.1 人脸识别卷积神经网络运行主要步骤

Requirement 1: 实现3x3大小的卷积核进行卷积的卷积层，支持步长为1或步长为2（stride=1 and stride=2），且padding=1。本次人脸识别模型的卷积神经网络需要实现三层卷积，分别是：

第一次：stride = 2, padding = 1。通过16个卷积核，将3*128*128的图片卷积为16*64*64大小的多层矩阵。

因为此次卷积有1的padding，且步长为2，因此输出矩阵的高和宽会是输入矩阵的各1/2。因为padding的存在，理论上矩阵的四周应该加上一圈0才可以开始计算。而在实际实现过程中，真正加上一圈0是对时间和空间的双重浪费，实现也会比较繁琐。因此在逻辑上而非实际上加上这一圈0才是正确的做法。在这层卷积中，我通过分类讨论的方法来实现矩阵不同位置元素的卷积操作。

在步长为2时且有padding时，对于高和宽为偶数的输入矩阵，其卷积后的输出矩阵的尺寸恰好是其1/2。这时从左上角的元素开始卷积，会成为卷积中心的点使用“ * ”表示（如图fig.3所示）。

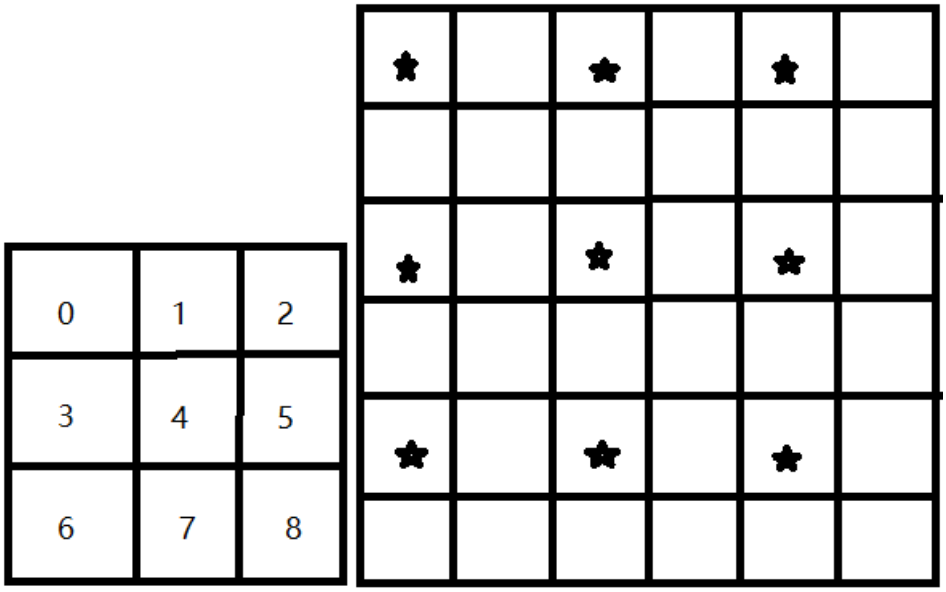


fig. 2.2 卷积核的一层 示意图　　*fig. 2.3 卷积前的矩阵（宽高为偶数） 示意图*

可以发现：

1. 卷积中心为矩阵左上角（第一行第一列）时，只有卷积核的（4，5，7，8）四个位置对应的元素参与运算，其它位置相当于和0相乘（如图fig.2所示）。

```
//left upper cornor
if (h == 0 && w == 0) {
    out_mat[out_c *(in_h / 2)*(in_w / 2) + (in_w / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * (in_h / 2) * (in_w / 2) + (in_w / 2) * h + w] +=
            in_mat[(in_h)*(in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2.4 卷积中心在矩阵左上角的卷积运算 代码实现

2. 当卷积中心为矩阵左侧的非左上角的点（第一列，非第一行）时，卷积核的（1，2，4，5，7，8）位置的元素参与运算。

```
//left but not upper cornor
else if (h != 0 && w == 0) {
    out_mat[out_c * (in_h / 2) * (in_w / 2) + (in_w / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * (in_h / 2) * (in_w / 2) + (in_w / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h)-1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 2 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2.5 卷积中心在矩阵左侧（非左上角）的卷积运算 代码实现

3. 卷积中心为矩阵上侧的非左上角的点（第一行，非第一列）时，卷积核的（3，4，5，6，

7，8）位置的元素参与运算。

```
//upper but not left cornor
else if (h == 0 && w != 0) {
    out_mat[out_c * (in_h / 2) * (in_w / 2) + (in_w / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * (in_h / 2) * (in_w / 2) + (in_w / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 6 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2.6 卷积中心在矩阵上侧（非左上角）的卷积运算 代码实现

以上三种情况是卷积操作的特殊情况，不存在卷积中心在其它位置的特殊情况。对于卷积中心在其它位置的卷积操作，卷积核的所有位置的元素都参与运算。

```
//common case
else {
    out_mat[out_c * (in_h / 2) * (in_w / 2) + (in_w / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * (in_h / 2) * (in_w / 2) + (in_w / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 0 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 2 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 6 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2.7 卷积中心在矩阵其它位置的卷积运算 代码实现

第二次：stride = 1, padding = 0。通过32个卷积核，将16*32*32的多层矩阵卷积为32*30*30大小的多层矩阵。

此次卷积padding为0，步长为1。因为此次不需要逻辑上在外侧加上一圈0，所以不存在卷积过程中的特殊位置，因此也不需要再进行分类讨论。

```
out_mat[out_c * (in_h - 2) * (in_w - 2) + (in_w - 2) * h + w] = conv_param.p_bias[out_c];
for (int i = 0; i < conv_param.in_channels; i++) {
    out_mat[out_c * (in_h - 2) * (in_w - 2) + (in_w - 2) * h + w] +=
        in_mat[(in_h) * (in_w)*i + (in_w)*h + w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 0 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w)*h + w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w)*h + w + 2] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 2 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w) * (h + 1) + w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w) * (h + 1) + w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w) * (h + 1) + w + 2] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w) * (h + 2) + w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 6 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w) * (h + 2) + w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
        + in_mat[(in_h) * (in_w)*i + (in_w) * (h + 2) + w + 2] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
}
```

fig. 2.8 第二次卷积 每个位置卷积运算的代码实现

第三次：stride = 2, padding = 1。通过32个卷积核，将32*15*15的多层矩阵卷积为

32*8*8大小的多层矩阵。

此次卷积与第一次卷积最大的不同点在于，输入的矩阵的宽和高为奇数。因为stride = 2，padding = 1，所以输出矩阵的宽和高应为输入矩阵的1/2。但是输入矩阵的宽和高不是偶数，因此输出矩阵的宽和高实际为输入矩阵（宽+1）和（高+1）的1/2。下图（fig.10）中会成为卷积中心的点使用“ * ”表示。

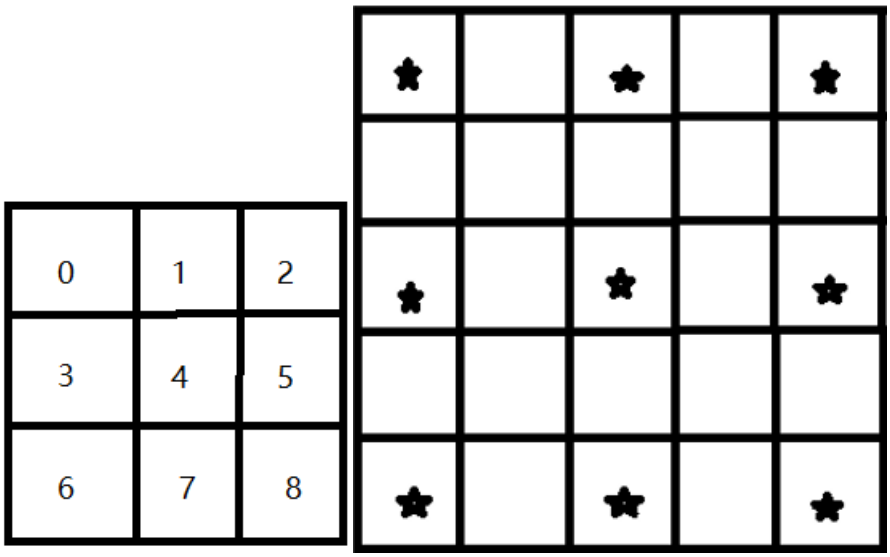


fig. 2.9 卷积核的一层 示意图　　*fig. 2.10 卷积前的矩阵（宽高为奇数） 示意图*

可以发现，和第一次的卷积不同，因为输入的矩阵的宽和高为奇数，所以矩阵右侧和下侧的点也会成为卷积中心，因此此次分情况讨论，且情况会比第一次卷积要多，共有9种。

1. 卷积中心为矩阵左上角（第一行第一列）时，只有卷积核的（4，5，7，8）四个位置对应的元素参与运算，其它位置相当于和0相乘。

```
//1. left upper cornor
if (h == 0 && w == 0) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_h + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2. 11 卷积中心在矩阵左上角的卷积运算 代码实现

2. 卷积中心为矩阵右上角（第一行最后一列）时，只有卷积核的（3，4，6，7）四个位置对应元素参与运算。

```
//2. right upper cornor
else if (h == 0 && w == (((in_w + 1) / 2) - 1)) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 6 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9];
    }
}
```

fig. 2.12 卷积中心在矩阵右上角的卷积运算 代码实现

3. 卷积中心为矩阵上侧非角位置（第一行非第一列非最后一列）时，卷积核的（3，4，5，6，7，8）位置对应元素参与运算。

```
//3. upper but not cornor
else if (h == 0 && w != 0 && w != (((in_w + 1) / 2)-1)) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 6 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2.13 卷积中心在矩阵上侧非角位置的卷积运算 代码实现

4. 卷积中心为矩阵左下角（最后一行第一列）时，卷积核的（1，2，4，5）四个位置参与运算。

```
//4. left lower cornor
else if (h == (((in_h + 1) / 2) - 1) && w == 0) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 2 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9];
    }
}
```

fig. 2.14 卷积中心在矩阵左下角的卷积运算 代码实现

5. 卷积中心在矩阵左侧非角位置（第一列非第一行非最后一行）时，卷积核的（1，2，4，5，7，8）位置对应元素参与运算。

```
//5. left but not cornor
else if (h != 0 && h != (((in_h + 1) / 2) - 1) && w == 0) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 2 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2.15 卷积中心在矩阵左侧非角位置的卷积运算 代码实现

6. 卷积中心在矩阵右下角位置（最后一列最后一行）时，卷积核的（0，1，3，4）位置对应元素参与运算。

```
//6. right lower cornor
else if (h == (((in_h + 1) / 2) - 1) && w == (((in_w + 1) / 2) - 1)) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 0 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9];
    }
}
```

fig. 2.16 卷积中心在矩阵右下角的卷积运算 代码实现

7. 卷积中心在矩阵右侧非角位置（最后一列非第一行非最后一行）时，卷积核的（0，1，3，4，6，7）位置对应元素参与运算。

```
//8. right but not cornor
else if (w == (((in_w + 1) / 2) - 1) && h != (((in_h + 1) / 2) - 1) && h != 0) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 0 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 6 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9];
    }
}
```

fig. 2.17 卷积中心在矩阵右侧非角位置的卷积运算 代码实现

8. 卷积中心在矩阵下侧非角位置（最后一行非第一列非最后一列）时，卷积核的（3，4，5，6，7，8）位置对应元素参与运算。

```
//7. lower but not cornor
else if (h == (((in_h + 1) / 2) - 1) && w != (((in_w + 1) / 2) - 1) && w != 0) {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 0 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 2 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9];
    }
}
```

fig. 2.18 卷积中心在矩阵下侧非角位置的卷积运算 代码实现

以上8种情况为此类卷积的特殊情况，当卷积中心不在边角位置时，卷积核所有位置的元素都参与运算。

```
//9. common case
else {
    out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] = conv_param.p_bias[out_c];
    for (int i = 0; i < conv_param.in_channels; i++) {
        out_mat[out_c * ((in_h + 1) / 2) * ((in_w + 1) / 2) + ((in_w + 1) / 2) * h + w] +=
            in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 0 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 1 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) - 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 2 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 3 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 4 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * (2 * h) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 5 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w - 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 6 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 7 + i * 9]
            + in_mat[(in_h) * (in_w)*i + (in_w) * ((2 * h) + 1) + 2 * w + 1] * conv_param.p_weight[out_c * conv_param.in_channels * 9 + 8 + i * 9];
    }
}
```

fig. 2.19 卷积中心在矩阵非边角位置的卷积运算 代码实现

Requirement 2: 程序能够正确输出对图片的confidence scores，即对于图片上是否存在人脸的确信程度（介于0和1之间）。

在三次卷积两次池化操作全部完成后，得到的是32*8*8大小的多层矩阵。若要得到图片的confidence scores，需要先将多重矩阵“拉平”，即按照顺序展开成为一维的2048个元素的向量。因为原本就使用的一维数组来表示的多重矩阵，所以此步并不需要进行操作。

2048个元素分别有对应的两套weight和bias。weight和它们相乘，将和加起来，再加上bias，重复两次的操作即可得到我们需要的两个值。

```
//FC
float con1 = fc_params[0].p_bias[0];
for (int i = 0; i < 2018; i++) {
    con1 += out_mat5[i] * fc_params[0].p_weight[i];
}
double con2 = fc_params[0].p_bias[1];
for (int i = 0; i < 2018; i++) {
    con2 += out_mat5[i] * fc_params[0].p_weight[2018 + i];
}
```

fig. 2.20 全连接操作 代码实现

此时的两个值已经分别代表着"背景"和"人脸"的可能程度，若要更加直观地得到每类的概率（confidence scores），还需要一步 softmax 操作，即通过公式将两个值进行标准化，成为介于0和1之间，和为1的两个值。

$$p_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

```
//soft max
double p1 = 1.0 / (1.0 + exp(con2-con1));
double p2 = 1.0 - p1;
cout <<"bg score: "<< p1 << endl;
cout <<"face score: "<< p2 << endl;
```

fig. 2.21 softmax操作 计算公式（左） 代码实现（右）

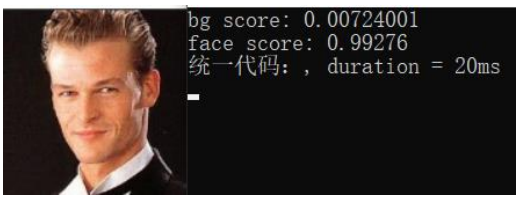
Requirement 3: 优化程序的实现。

在上述进行卷积操作时，第一次卷积和第三次卷积的步长和padding都是一样的，而因为第三次卷积输入的多重矩阵的高和宽为奇数，因此卷积后输出的多重矩阵的宽和高分别为（in_w+1）/2 和（in_h+1）/2，同时有更多的边角情况需要考虑。因此在写convBNReLU( )函数的时候，我将这两次卷积按照两种不同的代码来进行实现。

经过仔细思考，发现当输入矩阵宽和高都是偶数时，输出的多重矩阵的宽和高分别为in_w/2和in_h/2，实际上和（in_w+1）/2 和（in_h+1）/2是相等的。而考虑了更多边角情况的代码也能正确运行较少情况的第一次卷积，因此，完全可以抛弃第一次卷积使用的代码，而与第三次卷积共用一套代码，从而减少代码量。

记录从第一次卷积开始前到输出结果总共所用的时间：

对比第一次卷积和第三次卷积使用同一套代码与使用两套不同代码，测试同一张图运行时间：




```
bg score: 0.00724001
face score: 0.99276
统一代码: , duration = 20ms
```

```
bg score: 0.00724001
face score: 0.99276
统一代码: , duration = 19ms
```

使用同一套代码：三次测试耗时分别为20ms, 20ms, 19ms。


```
bg score: 0.000274585
face score: 0.999725
不同代码: , duration = 19ms
```

```
bg score: 0.000274585
face score: 0.999725
不同代码: , duration = 19ms
```

使用两套不同代码：三次测试耗时分别为18ms, 19ms, 19ms。

看上去使用同一套代码会略微增加耗时，为了使结果更加直观有效，我们使用3张不同的照片，记录每张照片运行10次的时间进行对比：

代码\照片名	face.jpg	bg.jpg	face_woman00.jpg
统一代码	212ms	200ms	216ms
两套代码	201ms	196ms	206ms

测试发现，使用两套代码确实比统一使用第三次卷积的代码要更加省时。原因在于第一次卷积并不需要进行如此多的判断，而第三次卷积边角情况较多，前面较多判断拖累了时间。

基于此，如果两次卷积使用不同的代码，考虑到基础情况是最多的，将基础情况放在第一个位置进行判断是不是会一定程度上节省时间呢?

将基础情况放在第一个位置进行判断，进行与上述相同的实验，结果如下：

代码\照片名	face.jpg	bg.jpg	face_woman00.jpg
基础情况上移	198ms	192ms	202ms

结果证实将基础情况优先判断是省时的。

此外，在编译时附加-O3选项可以使程序提速4~5倍。使用SIMD指令（single instruction, multiple data）将多个数据放入同一寄存器中进行计算，提高效率。因为每个float是4个字节，可以同时把8个浮点数放在一个寄存器里，理论上加速8倍，实际上也能达到1倍左右的加速。

另外一个加速策略是利用OpenMP去做并行计算。因为并行运算会使得for循环不按照顺序进行运算，而是共同进行运算，因此可能会出现计算结果错误的情况。而对于cnn的卷积运算，每次计算都是将结果赋值到对应的输出矩阵的位置，计算的先后顺序并不会影响结果的正确性。因此可以使用OpenMP #pragma omp parallel for来进行加速运算。但由于每次运算都是较为复杂的，因此，经测试，OpenMP并没有产生很好的加速效果，但确实带来了一定程度上的提升。

代码\照片名	face.jpg	bg.jpg	face_woman00.jpg
OpenMP加速	200ms	186ms	195ms

Part 3 – Code

代码已上传至https://github.com/JiangRunzhi/cpp_final_project_cnn。

Part 4 - Result & Verification

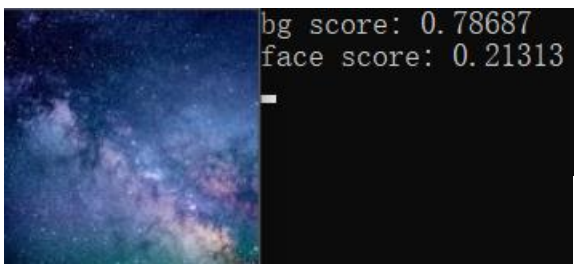
通过测试不同的128*128尺寸的彩色照片验证结果。

测试标准：
一致：C++实现的结果若与python实现的结果相差不超过0.01
答案一致：实现的结果相差超过0.01，但判断照片是人脸还是背景的答案一致
不一致：实现的结果相差超过0.01，但判断照片是人脸还是背景的答案不一致

测试结果如下：(左图为实现的c++得到的结果，右图为demo.py得到的结果。)

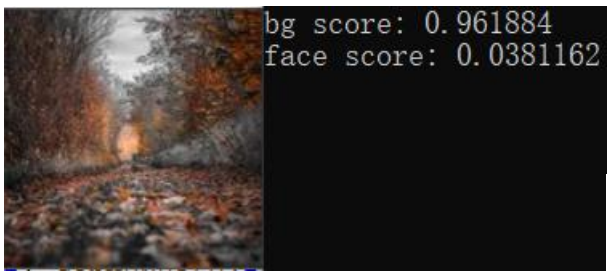
bg score: 0.999996
face score: 0.000004.

fig. 4.1 c++与python结果一致



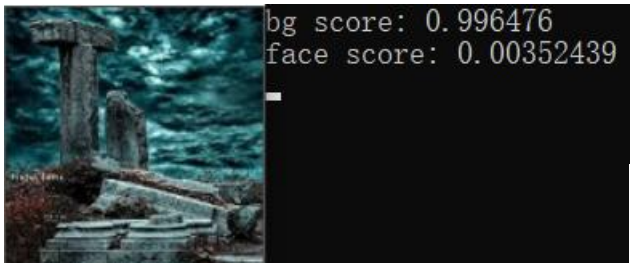
bg score: 0.997291
face score: 0.002709.

fig. 4.2 c++与python结果答案一致



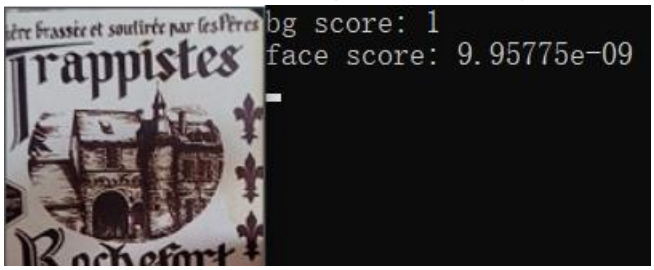
bg score: 0.342620
face score: 0.657380.

fig. 4.3 c++与python结果不一致，python判断错误



bg score: 1.000000
face score: 0.000000.

fig. 4.4 c++与python结果一致

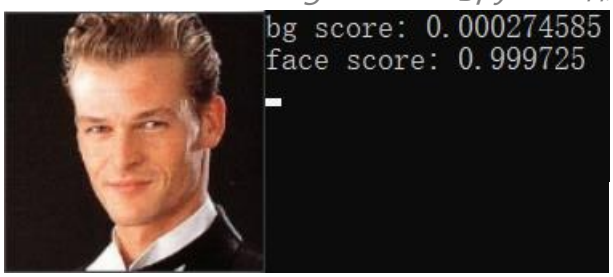


bg score: 1.000000
face score: 0.000000.

fig. 4.5 c++与python结果一致

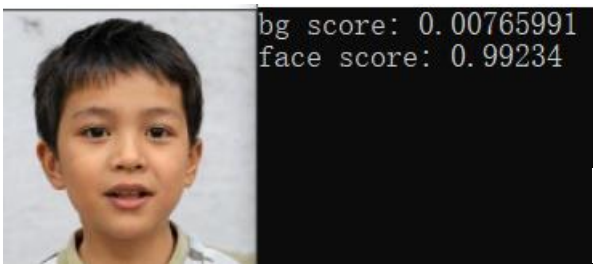
bg score: 1.000000
face score: 0.000000.

fig. 4.6 c++与python结果一致



bg score: 0.007086
face score: 0.992914.

fig. 4.7 c++与python结果一致



bg score: 0.050271
face score: 0.949729.

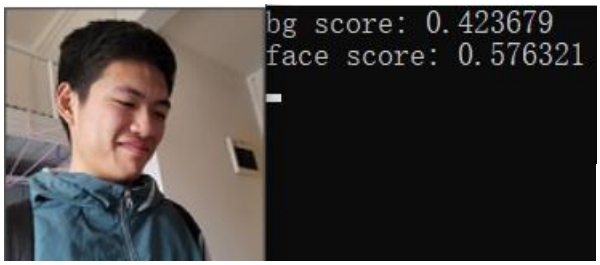
fig. 4.8 c++与python结果答案一致

bg score: 0.000031
face score: 0.999969.

fig. 4.9 c++与python结果一致

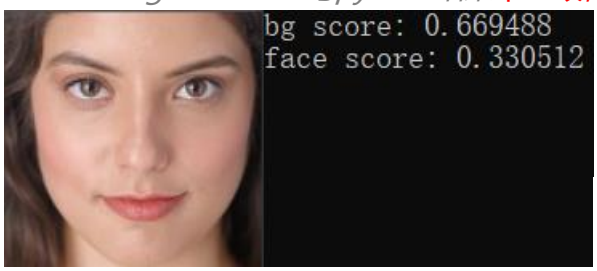
bg score: 0.411305
face score: 0.588695.

fig. 4.10 c++与python结果不一致，c++判断错误



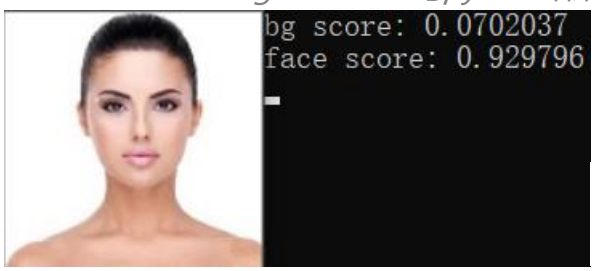
bg score: 0.999929
face score: 0.000071.

fig. 4.11 c++与python结果不一致，python判断错误



bg score: 0.935015
face score: 0.064985.

fig. 4.12 c++与python结果答案一致



bg score: 0.784782
face score: 0.215218.

fig. 4.13 c++与python结果不一致，python判断错误

bg score: 0.941948
face score: 0.058051.

fig. 4.14 c++与python结果不一致，python判断错误

上述14张测试图片，其中有9张c++和python运行的结果是一致或答案一致的，而5张不一致的图片中，有4张是python判断错误，而c++判断错误的只有一张。可以判断c++正确实现了人脸识别的卷积神经网络。

Python和c++所用的运算参数是完全一样的，之所以得到不同的结果，推测其调用的卷积神经网络的计算与此次project的计算有着细节上的不同。

Part 5 - Difficulties & Solutions

1. 此次project在一开始时好几天都不知如何去做，也不懂python代码，因此我自学了几天python来理解老师给的代码，后来发现实际没必要，但也学到了一些python，享受了收获知识的快乐。
2. 对于要做的东西不知如何下手，通过老师的视频慕课掌握了cnn工作的基本流程。
3. 不清楚如何将OpenCV读到的图像转成数据来输入到矩阵里面，通过网上搜索结合自身实际解决了这个问题。
4. 不知道老师给的参数如何去使用，经过反复观察，终于懂得了这些参数的用法。
5. 按照过程写出来程序之后，不能输出正确结果，无论输入什么图片，都会被自己写的程序判断成背景。后来经过对自己程序的反复观察，终于发现是循环里有个参数写的不对，改正之后程序终于正常运行。