A Simple CNN Model

NAME： 林洁芳

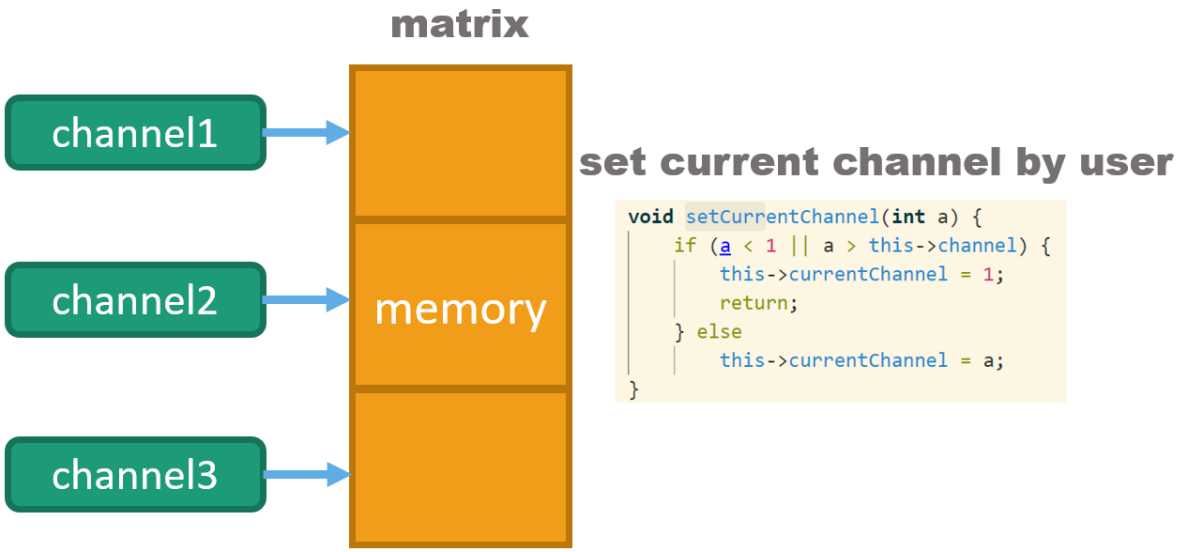
ID : 12011543

Note:

The test was conducted on VScode and Clion at the same time. Since the output of VScode was too small and arranged too closely, the display effect was not good, so most of the output of the display was from Clion.

1. You can connect to each specific section through the directory.

2. about channel




```
void setCurrentChannel(int a) {
    if (a < 1 || a > this->channel) {
        this->currentChannel = 1;
        return;
    } else
        this->currentChannel = a;
}
```

3. In project3, the implementation of SIMD is not portable because of the uncertainty of data type and matrix size, and the architecture of ARM is different from x86. ARM also has SIMD-like methods to match its architecture. Therefore, this project adopts matrix multiplication optimized by hardware.

1 Thought & Analysis

1.1 A simple CNN model is realized, in which the matrix convolution part can be converted to matrix multiplication for calculation.

Consider using the way called "Imcol + MEC" to make good use of ROI, to realize the transformation.

1.2 Consider implementing logical zeroing in the padding procedure to avoid memory waste.

1.3 Consider changing the order of pixel elements, that is, converting BRG to RGB, and compare and observe the calculation results.

In addition, the following **libraries** are used for this project :

```
#include <opencv2/opencv.hpp>
#include <iostream>
#include <fstream>
#include <chrono>
```

2 Code & Description

Firstly, show all the **declaration of functions**. ***(The complete code is attached as a file.)***

At the beginning, the padding process is first performed with hard copies to compute the correct result;

After that, logical zero complementation is considered to directly compute the matrix that can be computed by matrix multiplication.

Finally, generalization is considered, still using logical zero complementation, and it can be applied to multivariate padding, stride and kernel sizes.

matclass.hpp

```
#pragma once

#include <iostream>
#include <fstream>
#include <opencv2/opencv.hpp>

using namespace std;
//#pragma GCC optimize(3)

class Matrix {
private:
    size_t currentChannel;
    size_t column;
    size_t row;
    size_t channel;
    size_t size;
    size_t step;
    size_t *pointTime;
```

```
    float *matrix;
public:
    size_t getRow() const;

    size_t getColumn() const;

    size_t getCurrentChannel() const;

    void setCurrentChannel(size_t a);

    float *getMatrix();

    size_t *getPointTime();

    size_t getStep() const;

    size_t getChannel() const;

    float getValue(size_t i, size_t j) const;

    void setMatPoint(size_t i, size_t j, float num);

    ~Matrix();

    bool release();

    bool create(size_t , size_t , size_t , size_t);

    Matrix();

    Matrix(size_t r, size_t c, size_t ch);

    Matrix(size_t r, size_t c);

    explicit Matrix(float a);

    Matrix(size_t r, size_t c, float *a);

    Matrix(size_t r, size_t c, size_t ch, float *a);

    Matrix(size_t r, size_t c, size_t ch, cv::Mat &);

    Matrix(const Matrix &);

    friend std::ostream &operator<<(std::ostream &os, const Matrix &a);

    friend std::istream &operator>>(std::istream &is, Matrix &a);

    void outputToFile(std::ofstream &);

    void readFromFile(const string &);

    Matrix &operator=(const Matrix &);

    Matrix getSubMat(size_t i, size_t j, size_t r, size_t c,size_t ch);

    bool pad(Matrix &, size_t );
```

```
    void changeSize(size_t i, size_t j);

    //first realization
    bool convolutionImcolMecTotal(Matrix & re, Matrix &, size_t,size_t);

    bool maxPooling(Matrix &);

    bool unFold(Matrix &);

    bool fullConnected(Matrix & , const float *, Matrix &);
    //second realization
    bool changeIntoMulMat(Matrix & re, size_t padding, size_t kernelSize, size_t
stride);

    bool conLogicPad(Matrix & con, Matrix &re , size_t stride, size_t preSize);
    //second realization (more general)
    bool changeIntoMulMatGeneral(Matrix &re, size_t padding, size_t kernelSize,
size_t stride);

private:

    bool convolutionImcolMec( Matrix &, Matrix & , size_t);
};
```

2.1 matrix class

Matrix class implemented using Project 4 and with improved memory management.

Project 4, when implementing memory management, forgot to consider that the pointer also requests 4 bytes of memory, and when making a shallow copy, this part of memory also needs to be freed, otherwise it will also cause a memory leak.

In this project, the vulnerability is fixed and the part of the memory of the pointer is also managed. The following is an example of an overriden copy constructor.

```
bool Matrix::create(size_t r = 0, size_t c = 0, size_t s = 0, size_t ch = 1) {
    release();
    this->pointTime = new size_t{1};
    this->column = c;
    this->row = r;
    this->step = s;
    this->channel = ch;
    this->currentChannel = 1;
    this->size = this->column * this->row;
    if (this->row != 0 && this->column != 0) {
        this->matrix = new float[r * c * ch]{};
    } else this->matrix = nullptr;
    return true;
}

bool Matrix::release() {
    this->column = 0;
    this->row = 0;
    this->step = 0;
    this->size = 0;
    this->channel = 0;
    this->currentChannel = 0;
```

```
    if (this->matrix != nullptr) {
        delete[] this->matrix;
        this->matrix = nullptr;
    }
    if (this->pointTime != nullptr) {
        delete this->pointTime;
        this->pointTime = nullptr;
    }
    return true;
}

Matrix::Matrix(const Matrix &a) {
    if (a.row == 0 || a.column == 0 || a.matrix == nullptr)
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : a.matrix is null" << endl;
    this->matrix = nullptr;
    this->pointTime = nullptr;
    create(a.row, a.column, a.step, a.channel);
    delete[] this->matrix;
    this->matrix = a.matrix;
    delete this->pointTime;// relase the memory
    this->pointTime = a.pointTime;
    this->pointTime[0]++;
}
```

2.2 for "robustness"

Use the error output stream for error output to reduce functions reporting errors due to illegal input. Take ROI as an example as follow.

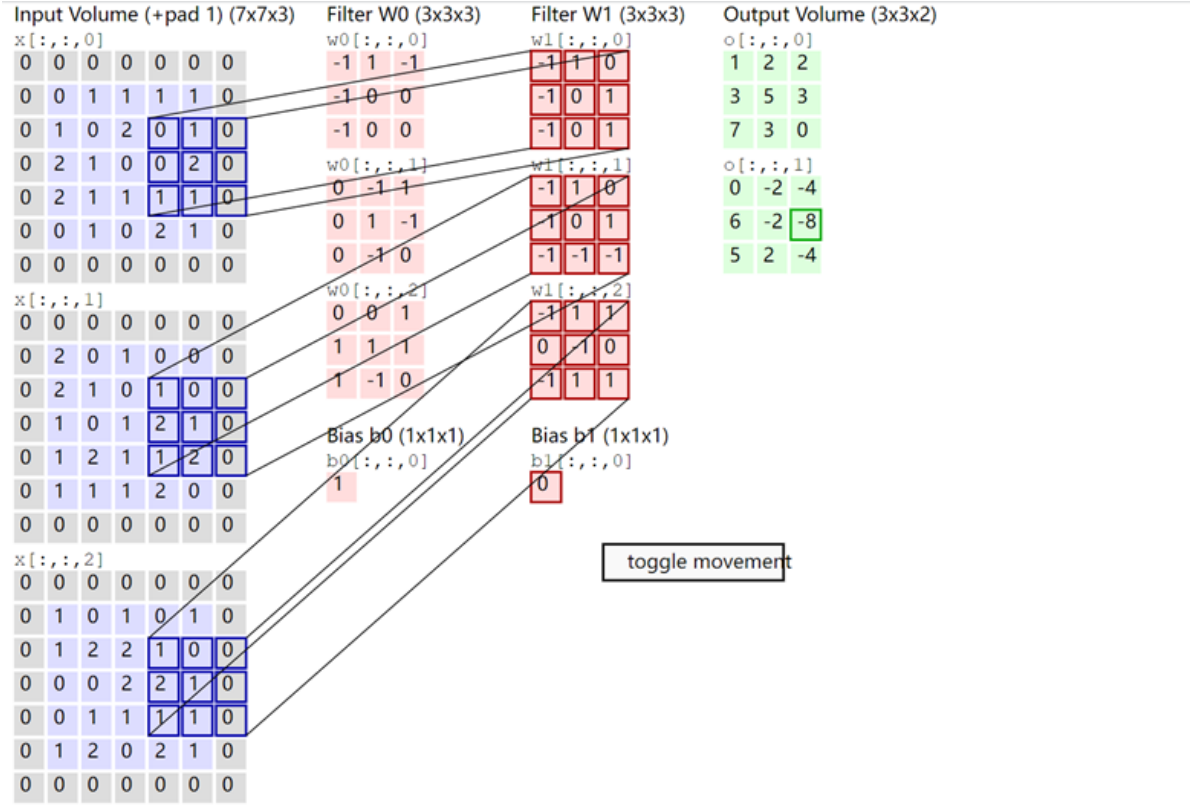
If an illegal parameter is passed in, an error message is output via the error output stream, including the file name, line number, function name, and error point.

```
Matrix Matrix::getSubMat(size_t i, size_t j, size_t r, size_t c, size_t ch) {
    // judge whether this matrix is null
    if (this->matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this matrix is null" << endl;
        return *this;
    }
    // judge whether i or j is out of range
    if (i > this->row - 1 || j > this->column - 1) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : point(i or j) is out of range" << endl;
        return *this;
    }
    // judge whether the size is out of range
    if (r > this->row || c > this->column || r + i > this->row || j + c > this-
>column) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : size is out of range" << endl;
        return *this;
    }
```

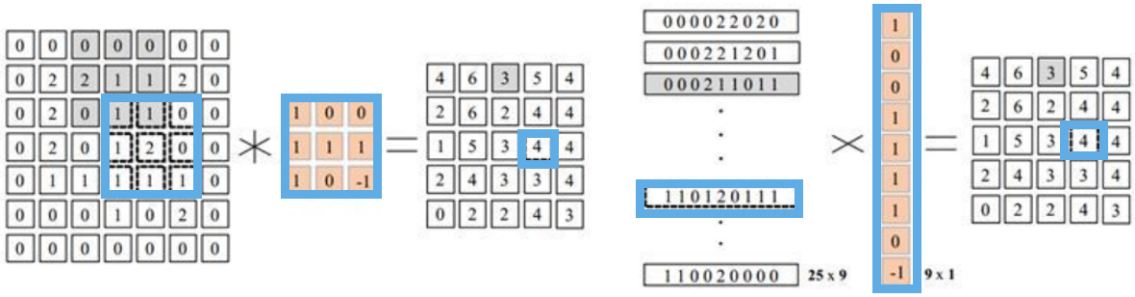
```
    // judge whether the channel is out of range
    if (ch == 0 || ch > this->channel) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : channel is out of range" << endl;
        return *this;
    }
    float *t = this->matrix + this->size * (ch - 1) + i * this->step + j;
    Matrix res = Matrix(r, c, t);
    res.pointTime[0] = this->pointTime[0] + 1;
    res.step = this->column;
    return res;
}
```

2.3 convolution to multiplication

To speed up the convolution calculation, the matrix convolution calculation can be converted into a matrix multiplication calculation. The matrix convolution is calculated schematically as follows.

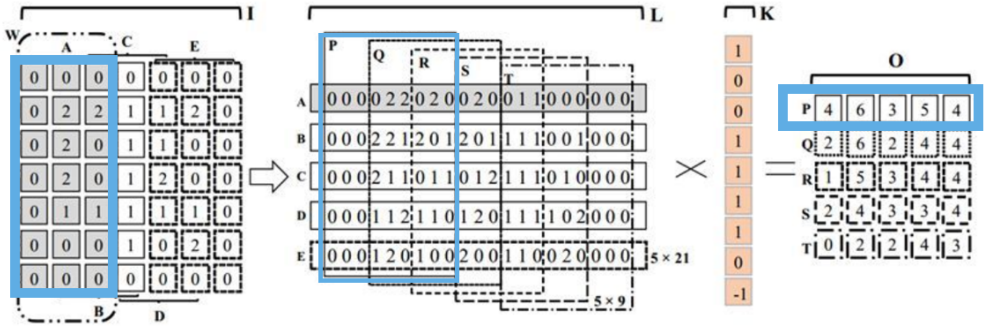


Firstly, one can consider a direct brute force approach to expand the matrix to be convolutionally computed into a one-dimensional vector. The matrix convolution into matrix multiplication is calculated schematically as follows (already padding) . **(called Imcol + GEMM)**



Converting in this way causes memory redundancy because there are multiple convolutional calculations in which the same data are used and these reused data are ignored.

Therefore, we consider saving memory and use another way of implementing convolution into matrix multiplication, showing the great advantage of ROI in the process. The schematic diagram is as follows (already padding) . **(called Imcol + MEC)**



The three convolutional computation methods memory usage analysis graph is as follows.

Take CNN model as an example, the matrix has n rows and columns, and the convolution kernel has m rows and columns.

calculation	memory cost
convolution (traditional)	(m + n - 1)*(m + n - 1)
Imcol+GEMM (violence)	m * m * n * n
Imcol+MEC	m * (n * (n + m – 1))

I implemented the code for **"Imcol + MEC"** to reduce the memory redundancy and use the experience from previous projects to optimize the hardware for the matrix multiplication process, thus reducing the program execution time and speeding up the program in the convolutional computation step. The code is shown below.

```
//based on already padding matrix
bool Matrix::convolutionImcolMec(Matrix &a, Matrix &re, size_t stride) {
    size_t n = 3 * stride;
    Matrix t = Matrix(re.row, re.row * n + (9 - n), 1);
    float *ch = t.matrix;
    const float *p = this->matrix + (this->currentChannel - 1) * this->size;

//This part is mainly about converting matrices into matrices that can be matrix
multiplied with the convolution kernel. And this process must use hard copy,
otherwise the memory is not continuous and the matrix multiplication acceleration
will be greatly reduced.

    for (int i = 0; i < t.row; ++i) {
        size_t cnt = 0;
        while (cnt < t.column) {
            for (int k = 0; k < this->row; ++k) {
                size_t cnt_k = 0;
                size_t si = k * this->column;
                while (cnt_k < 3) {
```

```
                    ch[cnt + i * t.column] = p[i * stride + cnt_k + si];
                    cnt_k++;
                    cnt++;
                }
            }
        }
    }
    size_t head_k = a.size * (a.currentChannel - 1);
    size_t head_re = re.size * (re.currentChannel - 1);

//Perform matrix multiplication calculations, using hardware optimization (which
works best) and relying on compilation optimization, multiple parallelism.

    for (int z = 0; z < re.row; ++z) {
        Matrix tem = t.getSubMat(0, z * stride * a.row, t.row, a.size, 1);
        float c;
#pragma omp parallel for
        for (int i = 0; i < t.row; i++) {
            for (int k = 0; k < a.size; k++) {
                c = tem.matrix[i * tem.step + k];
                for (int j = 0; j < 1; j++) {
                    re.matrix[head_re + z * re.column + i] += c *
a.matrix[head_k + k];
                }
            }
        }
    }
    return true;
}
```

2.4 first version realization

Initially, in order to verify the correctness of the whole calculation process first, the padding process was first implemented in a bad way, i.e., a direct hard copy of the complementary zeros, and the scope of application of the implemented method was very limited and only applicable to the calculation of this project. The code is shown below.

2.4.1 The functions:

```
//The code shows only the computational part related to convolution, the rest is
in the code file.

//using Imcol + MEC, Implement the convolution calculation of the already padding
matrix.
//ker means the convolution kernel, re means the result after convolution
//just a internal function!!!
bool Matrix::convolutionImcolMec(Matrix &ker, Matrix &re, size_t stride) {
    size_t n = 3 * stride;
    Matrix t = Matrix(re.row, re.row * n + (9 - n), 1);
    float *ch = t.matrix;
    const float *p = this->matrix + (this->currentChannel - 1) * this->size;
    for (int i = 0; i < t.row; ++i) {
        size_t cnt = 0;
        while (cnt < t.column) {
            for (int k = 0; k < this->row; ++k) {
                size_t cnt_k = 0;
                size_t si = k * this->column;
```

```
                while (cnt_k < 3) {
                    ch[cnt + i * t.column] = p[i * stride + cnt_k + si];
                    cnt_k++;
                    cnt++;
                }
            }
        }
    }
    size_t head_k = ker.size * (ker.currentChannel - 1);
    size_t head_re = re.size * (re.currentChannel - 1);
    for (int z = 0; z < re.row; ++z) {
        Matrix tem = t.getSubMat(0, z * stride * ker.row, t.row, ker.size, 1);
        float c;
#pragma omp parallel for
        for (int i = 0; i < t.row; i++) {
            for (int k = 0; k < ker.size; k++) {
                c = tem.matrix[i * tem.step + k];
                for (int j = 0; j < 1; j++) {
                    re.matrix[head_re + z * re.column + i] += c *
ker.matrix[head_k + k];
                }
            }
        }
    }
    return true;
}

//This method implements hard padding, i.e., by means of a hard copy.
bool Matrix::pad(Matrix &t, size_t padding) {
    if (this->column == 0 || this->row == 0 || this->matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this matrix is null" << endl;
        return false;
    }
    if (t.column == 0 || t.row == 0 || t.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : t.matrix is null" << endl;
        return false;
    }
    if (padding == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : padding is zero" << endl;
        return false;
    }
    for (int k = 0; k < t.channel; ++k) {
        size_t head_th = k * this->size;
        size_t head_t = k * t.size;
        size_t cnt = 0;
        if (padding == 1) {
            size_t edgeRow = this->row + 1;
            size_t edgeCol = this->column + 1;
            for (int i = 0; i < t.row; ++i) {
                if (i == 0 || i == edgeRow)
                    continue;
                else {
```

```
                    for (int j = 0; j < t.column; ++j) {
                        if (j == 0 || j == edgeCol)
                            continue;
                        else {
                            t.matrix[head_t + i * t.column + j] = this-
>matrix[head_th + cnt];
                            cnt++;
                        }
                    }
                }
            }
        } else if (padding == 2) {
            //调整
            size_t edgeRow = this->row + 2;
            size_t edgeCol = this->column + 2;
            size_t edgeRow1 = this->row + 1;
            size_t edgeCol1 = this->column + 1;
            for (int i = 0; i < t.row; ++i) {
                if (i == 0 || i == 1 || i == edgeRow || i == edgeRow1)
                    continue;
                else {
                    for (int j = 0; j < t.column; ++j) {
                        if (j == 0 || j == 1 || j == edgeCol || j == edgeCol1)
                            continue;
                        else {
                            t.matrix[head_t + i * t.column + j] = this-
>matrix[head_th + cnt++];
                        }
                    }
                }
            }
        }
    }
    return true;
}

//This function implements the full computation of all channels of the
convolution.
bool Matrix::convolutionImcolMecTotal(Matrix &re, Matrix &a, size_t stride,
size_t preSize) {
    if (this->column == 0 || this->row == 0 || this->matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this matrix is null" << endl;
        return false;
    }
    if (a.column == 0 || a.row == 0 || a.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : a's matrix is null" << endl;
        return false;
    }
    if (re.column == 0 || re.row == 0 || re.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : re's matrix is null" << endl;
        return false;
    }
```

```
    if (stride == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : stride is zero" << endl;
        return false;
    }
    if (this->column < stride || this->row < stride) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this->column < stride || this->row < stride cannot
calculate" << endl;
        return false;
    }
    if (preSize == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : perSize(inChannel) is zero" << endl;
        return false;
    }
    for (int i = 1; i <= re.channel; ++i) {
        re.setCurrentChannel(i);
        for (int j = 1; j <= preSize; ++j) {
            size_t curCh = (i - 1) * preSize + j;
            a.setCurrentChannel(curCh);
            this->setCurrentChannel(j);
            this->convolutionImcolMec(a, re, stride);
        }
    }
    return true;
}
```

2.4.2 The main:

The code is long, it will be shown in the file.

2.5 second version realization

After ensuring the accuracy of calculation, the method of logic padding in the padding process is realized, but it can only be applied to a limited padding value, and the size of the core can only be 3 * 3. The code is shown below.

2.5.1 The function:

```
//just a try, the code is not simply and elegant.
//only padding = 0/1/2,  kernel = 3; include all the channel.
//the transformation to the matrix required for matrix multiplication has been
implemented.

bool Matrix::changeIntoMulMat(Matrix &re, size_t padding, size_t kernelSize,
size_t stride) {
    if (this->column == 0 || this->row == 0 || this->matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this matrix is null" << endl;
        return false;
    }
    if (re.column == 0 || re.row == 0 || re.matrix == nullptr) {
```

```
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : re's matrix is null" << endl;
        return false;
    }
    if (kernelSize == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : kernel's size is zero" << endl;
        return false;
    }
    if (stride == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : stride is zero" << endl;
        return false;
    }
    if (kernelSize > this->row + 2 * padding){
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : kernel's size is illegal" << endl;
        return false;
    }
    size_t last = re.column - 1;
    size_t last1 = re.column - 2;
    size_t last2 = re.column - 3;
    size_t last3 = re.column - 4;
    size_t last4 = re.column - 5;
    size_t last5 = re.column - 6;
    for (int k = 1; k <= this->channel; ++k) {
        re.setCurrentChannel(k);
        this->setCurrentChannel(k);
        float *begin = re.matrix + (re.currentChannel - 1) * re.size;
        float *beginThis = this->matrix + (this->currentChannel - 1) * this-
>size;
        if (padding == 1) {
            for (size_t i = 0; i < re.row; ++i) {
                float *rePresent = begin + i * re.column;
                size_t i1 = 0;
                size_t i2;
                size_t firC = i * stride;
                size_t lastC = firC + kernelSize - 1;
                if (i == 0)
                    i2 = 0;
                else {
                    i2 = firC - 1;
                }
                for (int j = 0; j < re.column; ++j) {
                    if (j == 0 || j == 1 || j == 2)
                        continue;
                    if (j == last || j == last1 || j == last2)
                        continue;
                    if (i == 0) {
                        if (j % kernelSize == 0)
                            continue;
                        size_t availCol = kernelSize - 1;
                        if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
```

```
                            rePresent[j] = beginThis[i1 * this->column + i2];
                            i2++;
                        }
                        if (i2 == availCol || i2 == this->column) {
                            i1++;
                            i2 = 0;
                        }
                    } else if (i > 0 && i < re.row - 1) {
                        size_t availCol = firC - 1 + kernelSize;
                        if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                            rePresent[j] = beginThis[i1 * this->column + i2];
                            i2++;
                        }
                        if (i2 == availCol || i2 == this->column) {
                            i1++;
                            i2 = firC - 1;
                        }
                    } else {
                        if (((this->row + 2) - (kernelSize - stride)) % stride
== 0) {
                            if ((j + 1) % kernelSize == 0)
                                continue;
                        }
                        size_t availCol = firC - 1 + kernelSize;
                        if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                            rePresent[j] = beginThis[i1 * this->column + i2];
                            i2++;
                        }
                        if (i2 == availCol || i2 == this->column) {
                            i1++;
                            i2 = firC - 1;
                        }
                    }
                }
            }
        } else if (padding == 2) {
            for (size_t i = 0; i < re.row; ++i) {
                float *rePresent = begin + i * re.column;
                size_t i1 = 0;
                size_t i2;
                size_t firC = i * stride;
                size_t lastC = firC + kernelSize - 1;
                if (firC <= 2)
                    i2 = 0;
                else {
                    i2 = firC - 2;
                }
                for (int j = 0; j < re.column; ++j) {
                    if (j == 0 || j == 1 || j == 2 || j == 3 || j == 4 || j ==
5)
                        continue;
                    if (j == last || j == last1 || j == last2 || j == last3 || j
== last4 || j == last5)
                        continue;
                    if (i == 0) {
                        if (j % kernelSize == 0 || j % kernelSize == 1)
```

```
                            continue;
                        size_t availCol = kernelSize - 2;
                        if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                            rePresent[j] = beginThis[i1 * this->column + i2];
                            i2++;
                        }
                        if ((i2 == availCol || i2 == this->column)) {
                            i1++;
                            i2 = 0;
                        }
                    } else if (i == 1) {
                        if (stride >= 2) {
                            size_t availCol = firC - 2 + kernelSize;
                            if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                                rePresent[j] = beginThis[i1 * this->column +
i2];
                                i2++;
                            }
                            if ((i2 == availCol || i2 == this->column)) {
                                i1++;
                                i2 = 0;
                            }
                        } else {
                            if (j % kernelSize == 0)
                                continue;
                            size_t availCol = firC - 2 + kernelSize;
                            if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                                rePresent[j] = beginThis[i1 * this->column +
i2];
                                i2++;
                            }
                            if ((i2 == availCol || i2 == this->column)) {
                                i1++;
                                i2 = 0;
                            }
                        }
                    } else if (i > 1 && i < re.row - 2) {
                        size_t availCol = firC - 2 + kernelSize;
                        if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                            rePresent[j] = beginThis[i1 * this->column + i2];
                            i2++;
                        }
                        if (i2 == availCol || i2 == this->column) {
                            i1++;
                            i2 = firC - 2;
                        }
                    } else if (i == re.row - 2) {
                        if (((this->row + 4) - (kernelSize - stride)) % stride
!= 0) {
                            if ((j + 1) % kernelSize == 0)
                                continue;
                        }
                        size_t availCol = firC - 2 + kernelSize;
```

```
                        if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                            rePresent[j] = beginThis[i1 * this->column + i2];
                            i2++;
                        }
                        if (i2 == availCol || i2 == this->column) {
                            i1++;
                            i2 = firC - 2;
                        }
                    } else {
                        if (((this->row + 2) - (kernelSize - stride)) % stride
== 0) {
                            if (j % kernelSize == 1 || j % kernelSize == 2)
                                continue;
                        } else {
                            if ((j + 1) % kernelSize == 0)
                                continue;
                        }
                        size_t availCol = firC - 2 + kernelSize;
                        if (i2 < availCol && i1 < this->row && i2 < this-
>column) {
                            rePresent[j] = beginThis[i1 * this->column + i2];
                            i2++;
                        }
                        if (i2 == availCol || i2 == this->column) {
                            i1++;
                            i2 = firC - 2;
                        }
                    }
                }
            }
        } else if (padding == 0) {
            for (size_t i = 0; i < re.row; ++i) {
                float *rePresent = begin + i * re.column;
                size_t i1 = 0;
                size_t i2;
                size_t firC = i * stride;
                i2 = firC;
                for (int j = 0; j < re.column; ++j) {
                    size_t availCol = firC + kernelSize;
                    if (i2 < availCol && i1 < this->row && i2 < this->column) {
                        rePresent[j] = beginThis[i1 * this->column + i2];
                        i2++;
                    }
                    if (i2 == availCol || i2 == this->column) {
                        i1++;
                        i2 = firC;
                    }
                }
            }
        }
    }
    return true;
}

//realize matrix multiplication and complete the specific process of
convolution;
```

```
bool Matrix::conLogicPad(Matrix &con, Matrix &re, size_t stride, size_t preSize)
{
    if (this->column == 0 || this->row == 0 || this->matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this matrix is null" << endl;
        return false;
    }
    if (re.column == 0 || re.row == 0 || re.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : re's matrix is null" << endl;
        return false;
    }
    if (con.column == 0 || con.row == 0 || con.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : con's matrix is null" << endl;
        return false;
    }
    if (preSize == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : the pre size is zero" << endl;
        return false;
    }
    if (stride == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : stride is zero" << endl;
        return false;
    }
    for (int i = 1; i <= re.channel; ++i) {
        float *head_re = re.matrix + re.size * (i - 1);
        for (int j = 1; j <= preSize; ++j) {
            size_t curCh = (i - 1) * preSize + j;
            this->setCurrentChannel(j);
            float *head_k = con.matrix + con.size * (curCh - 1);
            for (int z = 0; z < re.row; ++z) {
                Matrix tem = this->getSubMat(0, z * stride * con.row, this->row,
con.size, j);
                float c;
                size_t p2 = z * re.column;
#pragma omp parallel for
                for (int i1 = 0; i1 < this->row; i1++) {
                    size_t p3 = p2 + i1;
                    size_t p1 = i1 * tem.step;
                    for (int k = 0; k < con.size; k++) {
                        c = tem.matrix[p1 + k];
                        for (int j1 = 0; j1 < 1; j1++) {
                            head_re[p3] += c * head_k[k];
                        }
                    }
                }
            }
        }
    }
    return true;
```

```
}
```

2.5.2 The main:

The code is long, it will be shown in the file.

2.6 last version realization

Finally, logical zeroing is generally realized, that is, the padding, stride and convolution kernel size are liberalized, and the code is simplified.

Since square matrices are commonly used in convolutional neural networks, this project only realizes the freedom of value when the convolution kernel is square matrix. The code is shown below.

2.6.1 The functions:

```
//realize matrix multiplication and complete the specific process of
convolution;

bool Matrix::conLogicPad(Matrix &con, Matrix &re, size_t stride, size_t preSize)
{
    if (this->column == 0 || this->row == 0 || this->matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this matrix is null" << endl;
        return false;
    }
    if (re.column == 0 || re.row == 0 || re.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : re's matrix is null" << endl;
        return false;
    }
    if (con.column == 0 || con.row == 0 || con.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : con's matrix is null" << endl;
        return false;
    }
    if (preSize == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : the pre size is zero" << endl;
        return false;
    }
    if (stride == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : stride is zero" << endl;
        return false;
    }
    for (int i = 1; i <= re.channel; ++i) {
        float *head_re = re.matrix + re.size * (i - 1);
        for (int j = 1; j <= preSize; ++j) {
            size_t curCh = (i - 1) * preSize + j;
            this->setCurrentChannel(j);
```

```
            float *head_k = con.matrix + con.size * (curCh - 1);
            for (int z = 0; z < re.row; ++z) {
                Matrix tem = this->getSubMat(0, z * stride * con.row, this->row,
con.size, j);
                float c;
                size_t p2 = z * re.column;
#pragma omp parallel for
                for (int i1 = 0; i1 < this->row; i1++) {
                    size_t p3 = p2 + i1;
                    size_t p1 = i1 * tem.step;
                    for (int k = 0; k < con.size; k++) {
                        c = tem.matrix[p1 + k];
                        for (int j1 = 0; j1 < 1; j1++) {
                            head_re[p3] += c * head_k[k];
                        }
                    }
                }
            }
        }
    }
    return true;
}

//logical zeroing is generally realized, that is, the padding, stride and
convolution kernel size are liberalized

bool Matrix::changeIntoMulMatGeneral(Matrix &re, size_t padding, size_t
kernelSize, size_t stride) {
    if (this->column == 0 || this->row == 0 || this->matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : this matrix is null" << endl;
        return false;
    }
    if (re.column == 0 || re.row == 0 || re.matrix == nullptr) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : re's matrix is null" << endl;
        return false;
    }
    if (kernelSize == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : the kernel size is zero" << endl;
        return false;
    }
    if (stride == 0) {
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : stride is zero" << endl;
        return false;
    }
    if (kernelSize > this->row + 2 * padding){
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : kernel's size is illegal" << endl;
        return false;
    }
```

```
    for (int k = 0; k < this->channel; ++k) {
        float *begin = re.matrix + k * re.size;
        float *beginThis = this->matrix + k * this->size;
        size_t mod = (this->row + 2 * padding - (kernelSize - padding)) %
stride;
        size_t rBound = this->row + padding * 2 - padding;
        for (size_t i = 0; i < re.row; ++i) {
            float *rePresent = begin + i * re.column;
            size_t i1 = 0;
            size_t i2;
            size_t firC = i * stride;
            size_t i3 = padding - firC;
            size_t lastC = firC + kernelSize - 1;
            size_t i4 = kernelSize - (lastC - rBound + 1);
            size_t upBound = padding * kernelSize;
            size_t downBound = re.column - upBound;
            if (firC <= padding)
                i2 = 0;
            else {
                i2 = firC - padding;
            }
            for (int j = 0; j < re.column; ++j) {
                if (j < padding * kernelSize)
                    continue;
                if (j >= downBound)
                    continue;
                if (firC < padding) {
                    if (j % kernelSize < i3)
                        continue;
                }
                if (lastC >= rBound) {
                    if (j % kernelSize >= i4)
                        continue;
                }
                size_t availCol = firC - padding + kernelSize;
                if (i2 < availCol && i1 < this->row && i2 < this->column) {
                    rePresent[j] = beginThis[i1 * this->column + i2];
                    i2++;
                }
                if (i2 == availCol || i2 == this->column) {
                    i1++;
                    if (firC <= padding)
                        i2 = 0;
                    else {
                        i2 = firC - padding;
                    }
                }

            }
        }
    }
    return true;
}
```

2.6.2 The main:

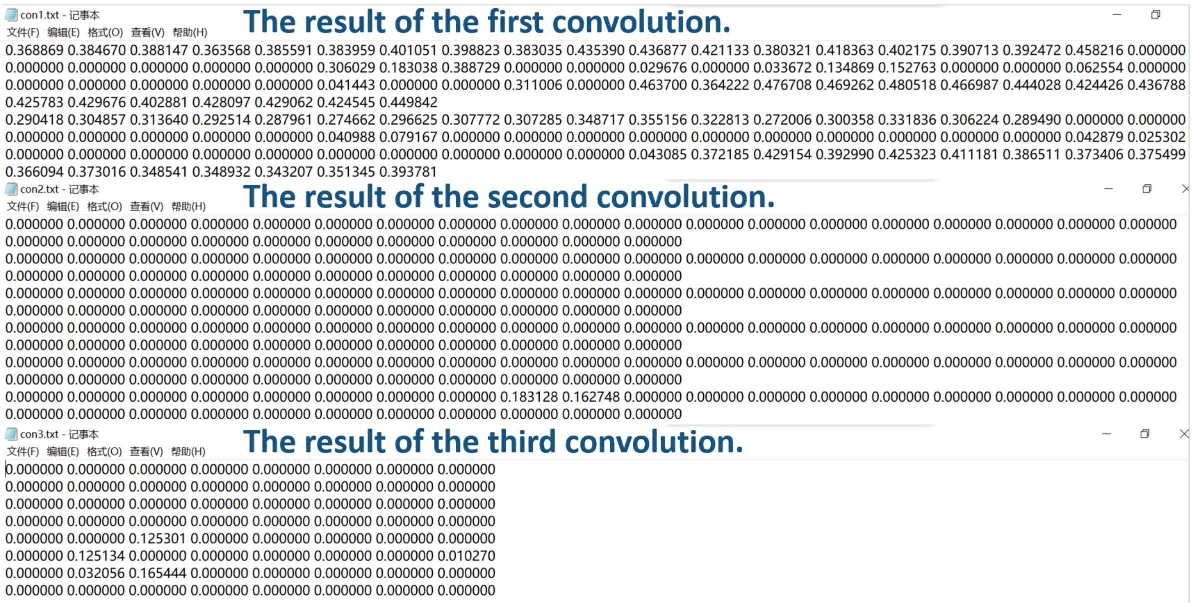
The code is long, it will be shown in the file.

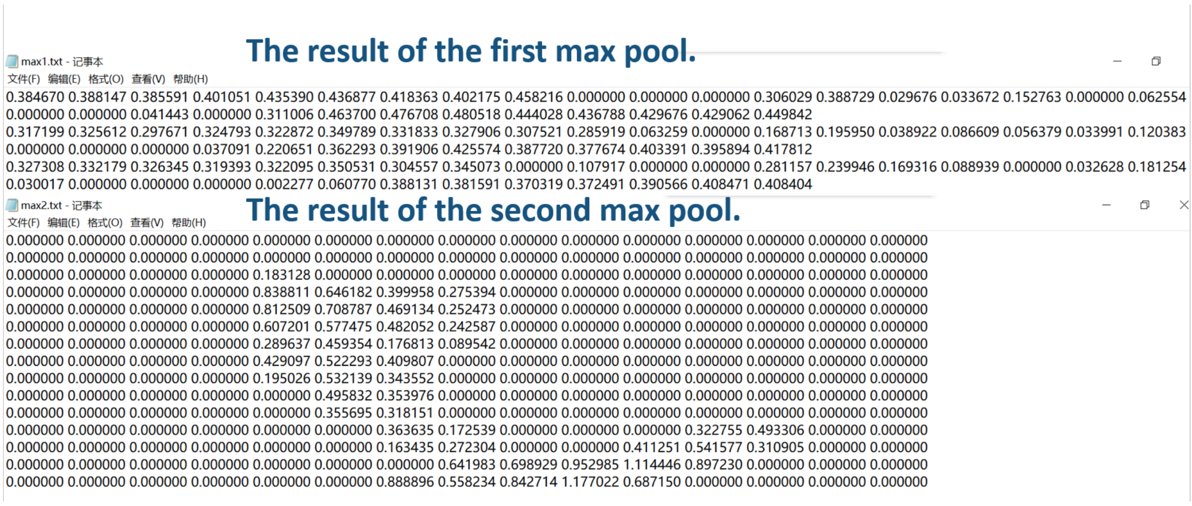
3 Result

All attempts at convolution implementations yield consistent results. (***warning: without changing BGR into RGB, the calculated result is consistent with the reference data provided by the teacher. Later, "Difficulty & Solution" will provide the analysis after changing the order, and the result is more accurate.***)

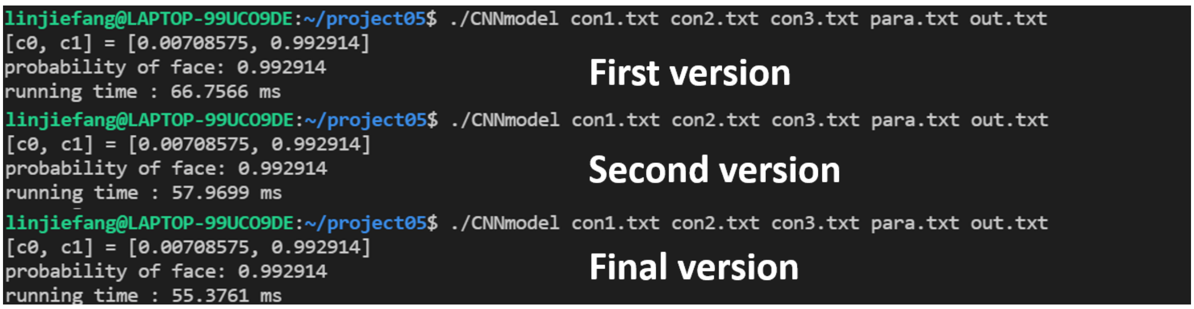
The **"running time"**, that is, the total program execution time, includes reading the convolution kernel from the file.

The result of convolution is **face.jpg** as an example.

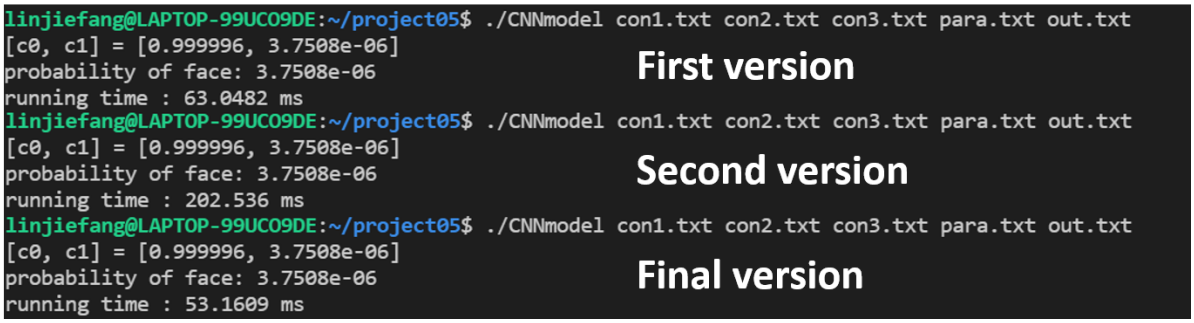




Here are the results of **face.jpg** and the total running time of the program.



Here are the results of **bg.jpg** and the total running time of the program.



The above answer is the same as the sample output on GitHub, but it doesn't change the read order. (without changing BGR into RGB)

> ### Example Output
>
> We provide a demo to output scores as an example in demo.py using PyTorch (>= 1.6.0) and two sample images in samples. You can run the demo and get the confidence scores as follows:
>
> ```
> $ python demo.py --img ./samples/face.jpg
> bg score: 0.007086, face score: 0.992914.
>
> $ python demo.py --img ./samples/bg.jpg
> bg score: 0.999996, face score: 0.000004.
> ```

4 Difficulty & Solution

4.1 BGR into RGB

After converting the BGR order to RGB order, the calculation results deviate from the sample output on GitHub, but the result is more correct. As can be seen from the following picture, the result of RGB image convolution is more accurate than that of BGR. Therefore, the reading order should be changed.

The change code for the read order is captured below.

BGR

```
uchar * cvImg = img.data;
for (int i = 0; i < this->row; ++i) {
    for (int j = 0; j < this->column; ++j) {
        ch1[cnt] = (float)cvImg[0] / 255.0f;
        ch2[cnt] = (float)cvImg[1] / 255.0f;
        ch3[cnt] = (float)cvImg[2] / 255.0f;
        cnt++;
        cvImg += 3;
    }
}
```

RGB

```
uchar * cvImg = img.data;
for (int i = 0; i < this->row; ++i) {
    for (int j = 0; j < this->column; ++j) {
        ch1[cnt] = (float)cvImg[2] / 255.0f;
        ch2[cnt] = (float)cvImg[1] / 255.0f;
        ch3[cnt] = (float)cvImg[0] / 255.0f;
        cnt++;
        cvImg += 3;
    }
}
```

The following lists test examples in different order.

```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt para.txt out.txt
[c0, c1] = [3.76485e-09, 1]
probability of face: 1
running time : 58.641 ms
```

face.jpg for RGB

```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt pa
[c0, c1] = [0.00708575, 0.992914]
probability of face: 0.992914
running time : 66.7566 ms
```

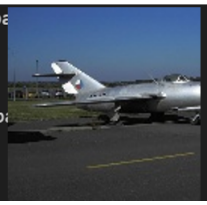
face.jpg for BGR

```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt p
[c0, c1] = [0.283473, 0.716527]
probability of face: 0.716527
running time : 68.4895 ms
```

face.jpg for BRG

```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt pa
[c0, c1] = [0.999996, 3.7508e-06]
probability of face: 3.7508e-06
running time : 63.0482 ms
```

bg.jpg for BGR

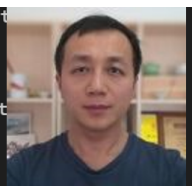


```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt pa
[c0, c1] = [1, 3.55695e-07]
probability of face: 3.55695e-07
running time : 65.6498 ms
```

bg.jpg for RGB

```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt para.t
[c0, c1] = [0.318777, 0.681223]
probability of face: 0.681223
running time : 66.5062 ms
```

yls.jpg for BGR



```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt para.t
[c0, c1] = [1.05165e-09, 1]
probability of face: 1
running time : 59.2471 ms
```

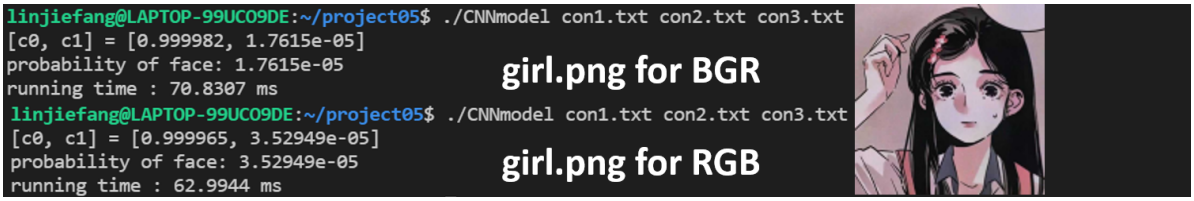
yls.jpg for RGB

```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt para.t
[c0, c1] = [0.112795, 0.887205]
probability of face: 0.887205
running time : 55.7393 ms
```

face-1.png for BGR

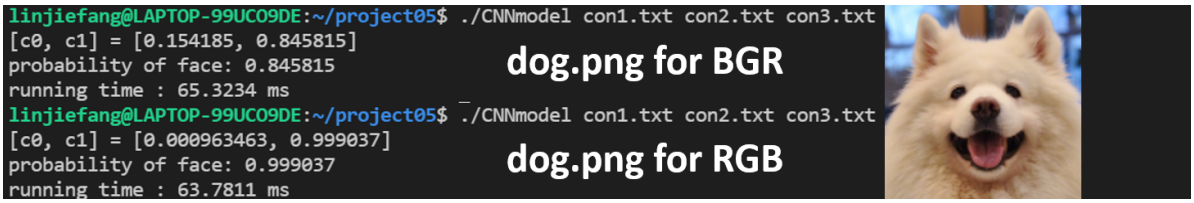
```
linjiefang@LAPTOP-99UCO9DE:~/project05$ ./CNNmodel con1.txt con2.txt con3.txt para.t
[c0, c1] = [1.89327e-06, 0.999998]
probability of face: 0.999998
running time : 61.4421 ms
```

face-1.png for RGB



Courtesy of my friends.

Sometimes the answers are strange. the following two images.

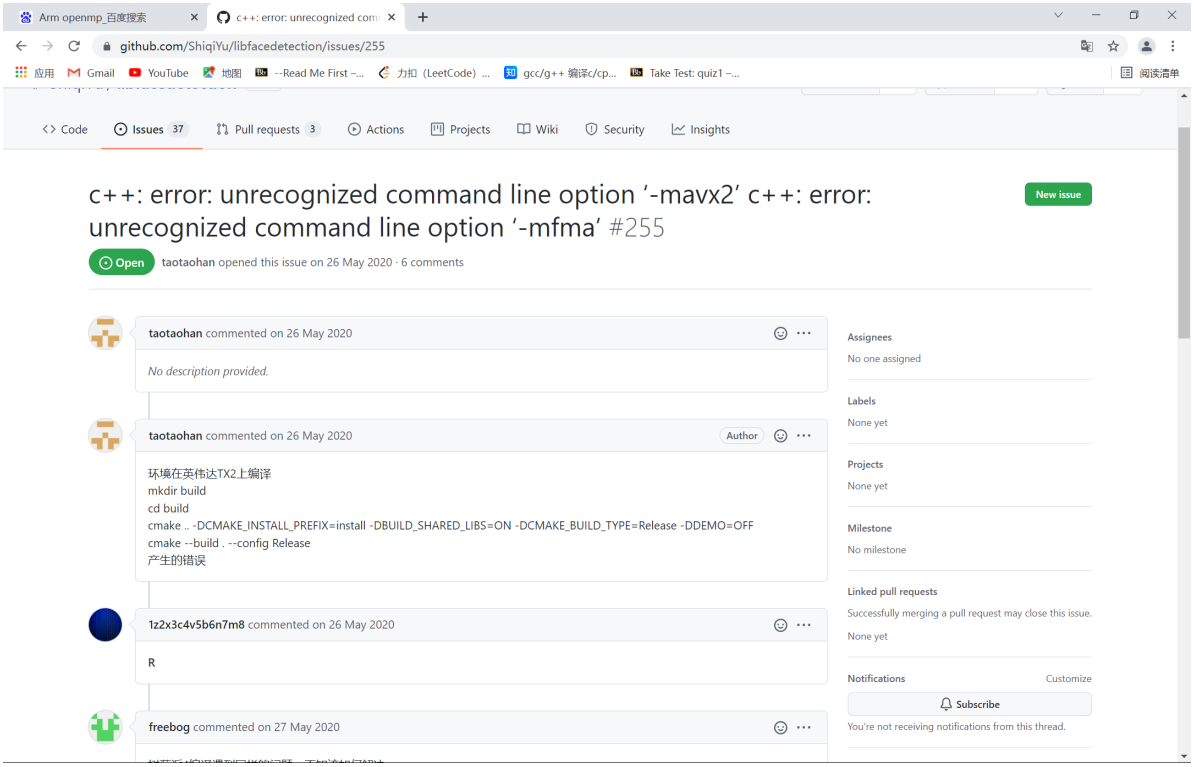


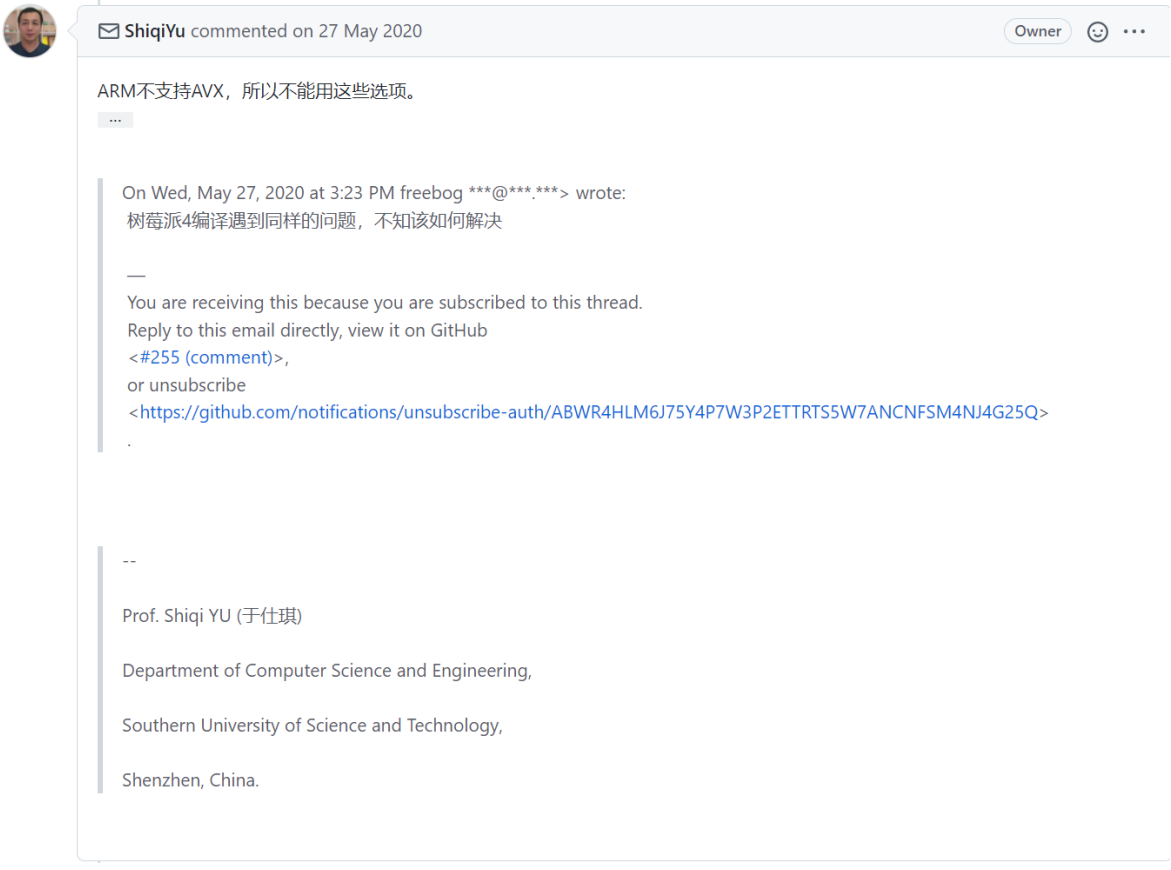
4.2 ARM & x86

4.2.1 data race

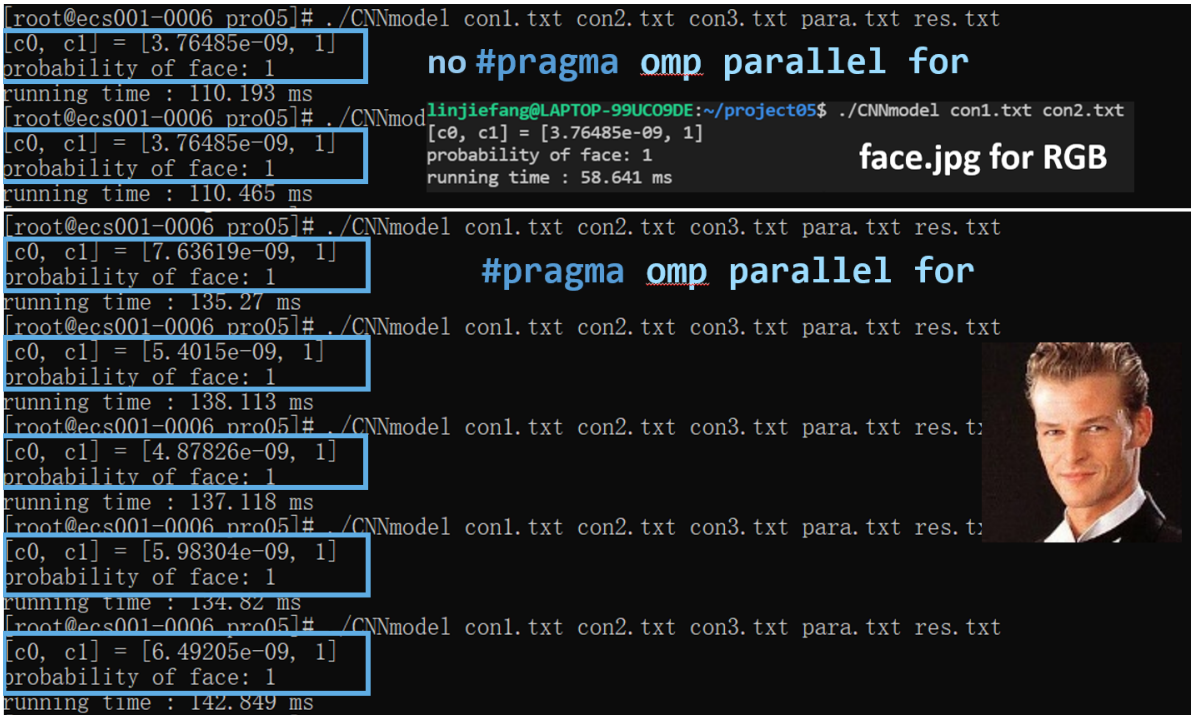
When tested on x86, compiler optimization handles concurrency errors automatically because it can be turned on at the same time as multiple parallelism, so the results are not random when the program is run locally.

However, ARM does not have compiler optimization to turn on, only multi-parallel computation can be carried out, because data competition leads to parallel errors, so the results will be random. The supporting evidence as follow :



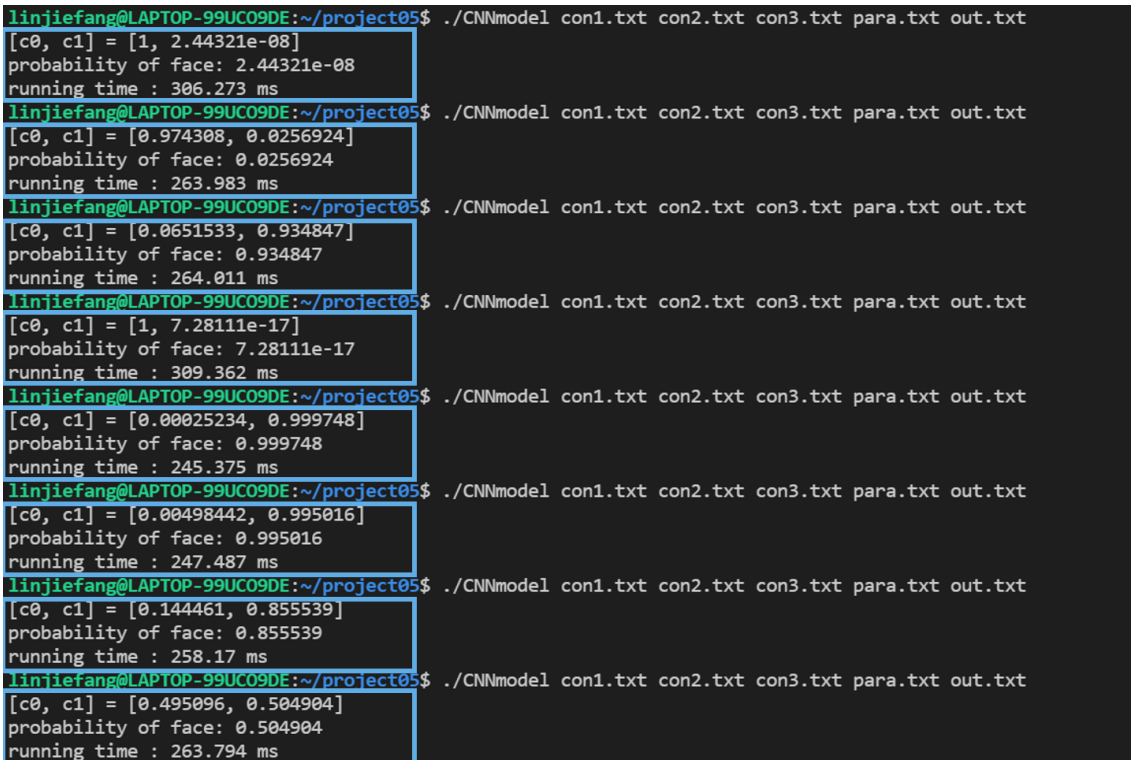


The following figure shows the difference between turning multiparallelism off and on. When multiparallelism is turned off, the results are consistent with x86 without randomness, whereas when multiparallelism is turned on, the results are slightly different.



conclusion

1. difference : ARM has no compiler optimization option, so concurrency errors cannot be fixed, while x86 can have both compiler optimization and multi-parallelism enabled, so concurrency errors can be fixed by O3.
2. same : If compilation optimizations are turned off locally, the same randomness as ARM can occur. The figure below shows the result of x86 with multi-parallelism turned off, and the randomness is very strong.



4.2.2 unsigned vs signed char

Since ARM's OpenCV stores data as signed char, I use the **uchar *** pointer to read pixels, which can be converted to unsigned char by type. The constructor code for reading pixels is as follows:

```
Matrix::Matrix(size_t r, size_t c, size_t ch, cv::Mat &img) {
```

```
    if (r == 0 || c == 0 || ch == 0)
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : row == 0 || column == 0 || channel == 0" << endl;
    if (img.empty())
        cerr << "file : " << __FILE__ << ", line : " << __LINE__ << ", function
: " << __FUNCTION__
             << ", error : image is empty" << endl;
    this->matrix = nullptr;
    this->pointTime = nullptr;
    create(r, c, c, ch);
    float *ch1 = this->matrix;
    float *ch2 = this->matrix + this->size;
    float *ch3 = ch2 + this->size;
    size_t cnt = 0;

    //can be converted to unsigned char by type

    uchar * cvImg = img.data;
    for (int i = 0; i < this->row; ++i) {
        for (int j = 0; j < this->column; ++j) {
            ch1[cnt] = (float)cvImg[2] / 255.0f;
            ch2[cnt] = (float)cvImg[1] / 255.0f;
            ch3[cnt] = (float)cvImg[0] / 255.0f;
            cnt++;
            cvImg += 3;
        }
    }
}
```

4.2.3 mentioned in project 4

1. **difference in running time**: ARM running time is slower than x86, on the basis of no compilation optimization.
2. **difference in storage address**: The upper part of the figure below is an example of x86 storage memory, and the following is an example of ARM storage memory.

Here is the memory storage example from project 4, “pointer” means the pointer of Matrix, “matrix pointer” means the pointer of the data;

```
pointer = 0x7fffc63997c0, matrix pointer = 0x7fffbebec4d0
pointer = 0x7fffc6399790, matrix pointer = 0x7fffbebec524
pointer = 0x7fffc6399760, matrix pointer = 0x7fffbebec4d0
pointer = 0x7fffc6399730, matrix pointer = 0x7fffbebec4d0
pointer = 0x7fffc6399700, matrix pointer = 0x7fffbebec4d0
pointer = 0xffffd92f99b8, matrix pointer = 0x36274530
pointer = 0xffffd92f99e8, matrix pointer = 0x36274584
pointer = 0xffffd92f9a18, matrix pointer = 0x36274530
pointer = 0xffffd92f9a48, matrix pointer = 0x36274530
pointer = 0xffffd92f9a78, matrix pointer = 0x36274530
```

5 Reference

https://github.com/ShiqiYu/libfacedetection/issues/255

https://cloud.tencent.com/developer/article/1512757

https://www.cnblogs.com/pinard/p/6483207.html

https://cs231n.github.io/assets/conv-demo/index.html

And a lot of blogs on CSDN.