

# CS205: C/C++ Program Design Project2

Student Name: YeeTone Wang(王奕童)

Student ID: 11910104

Course Instructor: Shiqi Yu(于仕琪)

**Topic 1:** Implement CNN using C++. If you do not know how to do it, you can convert the open source project <https://github.com/ShiqiYu/libfacedetection> into a C++ one from C. If you can improve the project and your patches are merged into the project, you will get extra scores for project 2.

## Catalogue

There are 8 parts in this project, 10 pages in the report in total.

**Part 0. How to use it and test?**

**Part 1. Description**

**Part 2. Result& Verification**

**Part 3. Implementation of the Optimization**

**Part 4. Memory management**

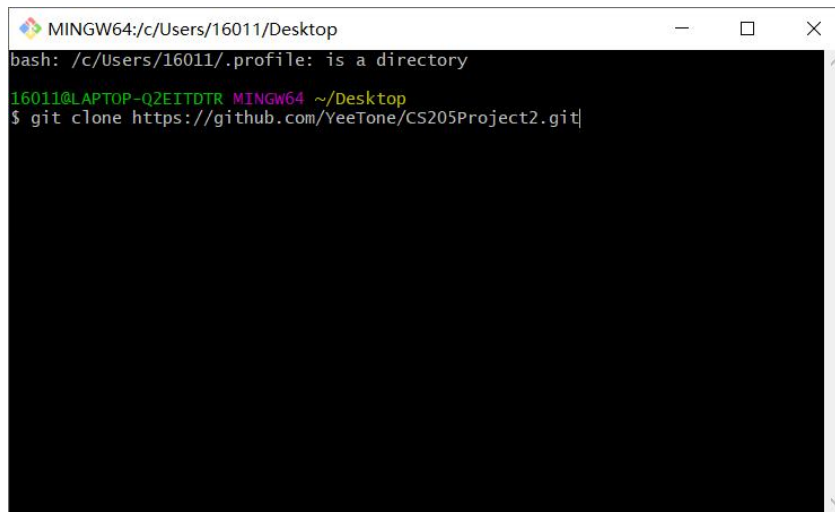
**Part 5. Test on Different Platform**

**Part 6.To Dos in the future**

**Part 7. Summary**

## Part 0. How to use it and test on VS2017

First open the git-Bash and input like this to clone the repository.



```
MINGW64:/c/Users/16011/Desktop
bash: /c/Users/16011/.profile: is a directory
16011@LAPTOP-Q2EITDTR MINGW64 ~/Desktop
$ git clone https://github.com/YeeTone/CS205Project2.git
```

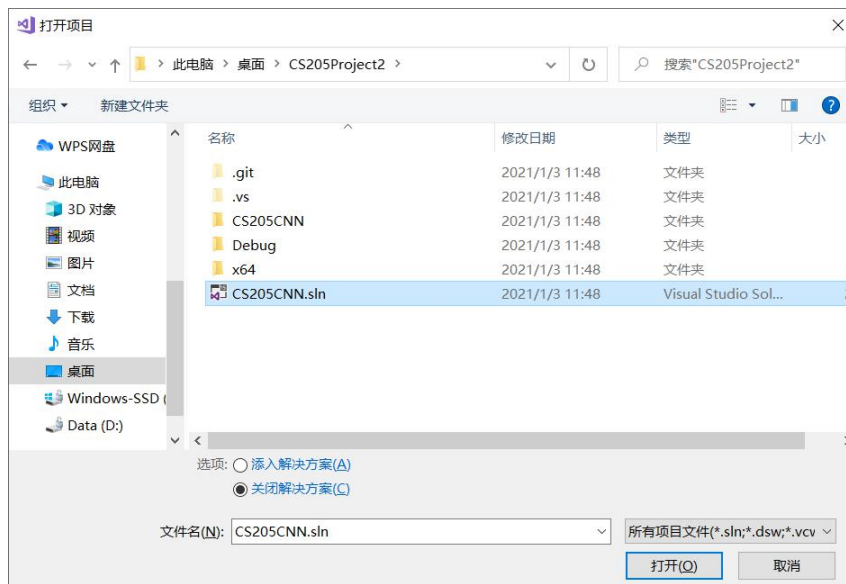
If you don't install VS2017 on your computer, please follow the instructions on LAB1 of CS205 2020 Fall.

※LAB1 --> [LAB1](#)

If you don't install OpenCV on your computer, please follow the instructions on LAB14 of CS205 2020 Fall.

※LAB14 --> [LAB14](#)


Then Open the VS2017(tr try to install if you don't have VS2017!) and choose the corresponding sln file.



Then you can just change the name of file in the main.cpp and start test!

## Part 1.Description

本次 Project 一共上交 5 个代码文件：

名称	修改日期	类型	大小
 Tools.cpp	2021/1/3 0:00	CPP 文件	268 KB
 Tools.h	2021/1/2 23:34	H 文件	1 KB
 main.cpp	2021/1/2 23:11	CPP 文件	1 KB
 Materials.h	2021/1/2 23:05	H 文件	1 KB
 Materials.cpp	2021/1/2 23:04	CPP 文件	1 KB

1. main.cpp: 程序执行入口，代码量极简；
2. Material.h: 主要用于卷积核和全连接层的结构体定义，代码量简练；
3. Material.cpp: 无内容，由于卷积核和全连接层无方法定义；
4. Tools.h: 主要用于封装一个禁止实例化的工具类——CNN\_Tools，其中所有的操作都由这个类加以实现；
5. Tools.cpp: 主要是封装训练好的数据以及对 Tools.h 中声明的方法加以实现。

### 【Requirement and Implementation】

1. 正确实现 3x3 的卷积核；该卷积核支持 stride=1 和 stride=2 情形，也支持 padding=1 的情形；
2. 对于输入的 128x128 的大小的图片，能够正确输出 Face 和 Background 的 confidence scores；对于非 128x128 的大小的图片，可以通过拉伸变换后正确输出 Face 和 Background 的 confidence scores；
3. 对于计算速度有编译选项、循环结构、内存访问等多种优化策略，并且报告中有相应的比较与分析；
4. 本程序支持跨平台，在 X86 和 ARM 平台上都能较好的支持；
5. Github 仓库：<https://github.com/YeeTone/CS205Project2.git>
6. project 报告长度适中，效果美观，易于阅读；



## Part 2. Result& Verification

测试环境：Windows 10

编译器：mingw-gcc/g++

开发 IDE：Visual Studio 2017

以下共有 8 个测试用例图片（4 个 face 和 4 个 background），会随着报告一同上传。

测试用例	说明	Face	Background	附图
0.jpg	标准测试用例 face	0.999948	5.20338e-05	
1.jpg	标准测试用例 bg	7.28705e-07	0.999999	

2.jpg	YeeTone 本人蓝底照片	1	1.19774e-09	
3.jpg	初音未来人脸照片	0.981678	0.0183221	
4.jpg	于仕琪老师主页人脸照片	1	9.095e-10	
5.jpg	南科大落日风景照	0.000740829	0.999259	
6.jpg	XP 系统大草原壁纸	5.2255e-05	0.999948	
7.jpg	Mac 系统银河壁纸	0.0285044	0.971496	

从以上 8 个测试用例的结果来看，大体符合我们的预期。因此可以验证此 CNN 算法的实现的正确性。

## Part 3. Implementation of the Optimization

### (1) CNN 实现的核心类——CNN\_Tools 的简要说明

核心方法及其作用：

[方法定义]`static void read_start(string s);`

[参数列表](`string s`)

[方法作用]提供良好的外部接入入口，只需传入一个字符串即可完成 CNN 的图片读取和卷积计算

[方法定义]`static Mat reshape(Mat& image);`

[参数列表](`Mat& image`)

[方法作用]如果传入的 `image` 是 128x128 的，则无需处理；否则通过 `cv::resize` 函数实现拉伸从而重新变为 128x128 的大小，从而增强程序的通用性。

[方法定义]`static float* bgr2rgb(Mat& image);`

[参数列表](`Mat& image`)

[方法作用]注意到 `cv::Mat` 一开始的数据类型是 BGR 顺序排布的 `unsigned char` 类型，不符合我们后续卷积层和全连接层的定义。这个方法是用于将 `uchar` 数据类型转换为 `float` 类型，并将 BGR 转化为 RGB 格式。

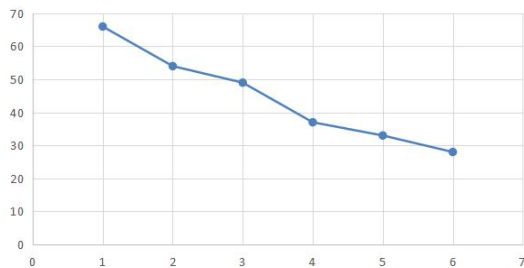
[方法定义]`static float* conv_RELU`  
(`int channels, int rows, int cols, float* input, conv_param& conv_p`);  
[参数列表]  
(`int channels, int rows, int cols, float* input, conv_param& conv_p`)  
[方法作用]这个方法集成卷积运算和 RELU 运算二合一，减小了函数调用的开销和内存开辟的空间花销。

[方法定义]`static float* max_pooling`  
(`int channels, int rows, int cols, float* input`);  
[参数列表](`int channels, int rows, int cols, float* input`)  
[方法作用]这个方法主要用于最大池化操作。

[方法定义]`static void matrix_Production`  
(`float* out, float* mat1, int r, int col, int c, float* mat2`);  
[参数列表](`float* out, float* mat1, int r, int col, int c, float* mat2`)  
[方法作用]这个方法主要是矩阵乘法，用于最后一步全连接层的操作。

## (2) 优化过程的运算时间折线图

以下折线图是针对 128x128 的图片的运算时间：



横轴单位：优化调试次数

竖轴单位：运行时间，单位 ms

前后效率提升：235%

## (3) BGR2RGB 优化过程介绍

[优化策略]增加循环步长从而减小循环次数

[踩过的坑]循环边界条件的判定失误，导致经常性的数组访问越界

[优化方案]经过于老师在 Lecture 中的介绍，考虑到输入图像经过 reshape 操作后必定为 128x128，我们可以尝试将关于 col 的循环步长从 1 变化为 4，从而减少运行过程中关于循环条件中 Comparison 的操作次数，并且提高访存命中率，从而节约运算时间。

简化版代码如下所示：

```
for (int i = 0; i < row; i++){
    for (int j = 0; j < col - 1; j += 4){
        //calculation
        ...
    }
}
```

```

    }
}

```

[优化策略]使用#pragma omp 语句实现并行运算。

[踩过的坑]在关于外层 row 循环遍历中使用语句有可能会运行时间变长，导致负优化；VS 里面如果不点选支持 OpenMP，则 Openmp 的并行不能生效。

[使用要求]需要在 VS2017 里面左上角的项目->属性->C/C++->语言->OpenMP 支持里面点选“是(/openmp)”选项，才能调用起 omp 的并行优化方案

[优化方案]在双层嵌套的 for 循环内外层使用 3 种 omp 优化并行语句：

最外层:#pragma omp parallel

循环层:#pragma omp single

计算层:#pragma omp task

简化版代码如下所示：

```

#pragma omp parallel
{
    #pragma omp single
    for (int i = 0; i < row; i++){
        ...
        for (int j = 0; j < col - 1; j += 4){
            #pragma omp task
            {
                //calculation
                ...
            }
        }
    }
}

```

## (4) Conv\_RELU 优化过程介绍

[优化策略]增加局部变量记录数值，减少乘法时间开销。

[踩过的坑]对于单次乘法来说没有必要记录数值，因为访问局部变量的时间有可能比乘法所需的代价还要高，从而导致负优化；应当记录多次乘法的时间，从而减少多次乘法的代价消耗。

[优化方案]使用局部变量记录多次乘法/除法的计算结果，如下所示：

```

int for_num = i_channel * kernel_size * kernel_size * i;
int fnumber = for_num + (j - 1) * kernel_size * kernel_size;
int fnumber2 = (j - 1) * (rows) * (cols) + k * (cols) + 1;

```

[优化策略]使用#pragma omp 语句实现并行运算

[踩过的坑]同之前，仍然是 omp 语句的循环次序的考虑问题；

[优化方案]反复调试，最终确定最优的 pragma 循环策略是：

```

for (int i = 0; i < o_cha; i++){
    int f_num = i * ks * ks * i_cha;
    for (int j = 1; j <= channels; j++){
        int fn = f_num + (j - 1) * ks * ks;
#pragma omp parallel for

```

```

    for (int k = 1; k <= rows - 2; k += conv_p.stride){
        for (int l = 1; l <= cols - 2; l += conv_p.stride){
            ...
        }
    }
}
}
}

```

在本机电脑上尝试在内层或者外层继续增加 `omp` 语句会导致运行时间变长，原因是本机电脑 CPU 核数有限，继续增加会导致 CPU 线程线程过多，如下所示：

```

线程 0x1a94 已退出，返回值为 0 (0x0)。
线程 0x1694 已退出，返回值为 0 (0x0)。
线程 0x45ac 已退出，返回值为 0 (0x0)。
线程 0x5f4 已退出，返回值为 0 (0x0)。
线程 0x8a8 已退出，返回值为 0 (0x0)。
线程 0x11d8 已退出，返回值为 0 (0x0)。
线程 0x1678 已退出，返回值为 0 (0x0)。
线程 0x994 已退出，返回值为 0 (0x0)。
线程 0x1890 已退出，返回值为 0 (0x0)。
线程 0x480c 已退出，返回值为 0 (0x0)。
线程 0x3bc8 已退出，返回值为 0 (0x0)。
线程 0x491c 已退出，返回值为 0 (0x0)。
线程 0x3068 已退出，返回值为 0 (0x0)。
线程 0x3368 已退出，返回值为 0 (0x0)。
线程 0x2258 已退出，返回值为 0 (0x0)。

```

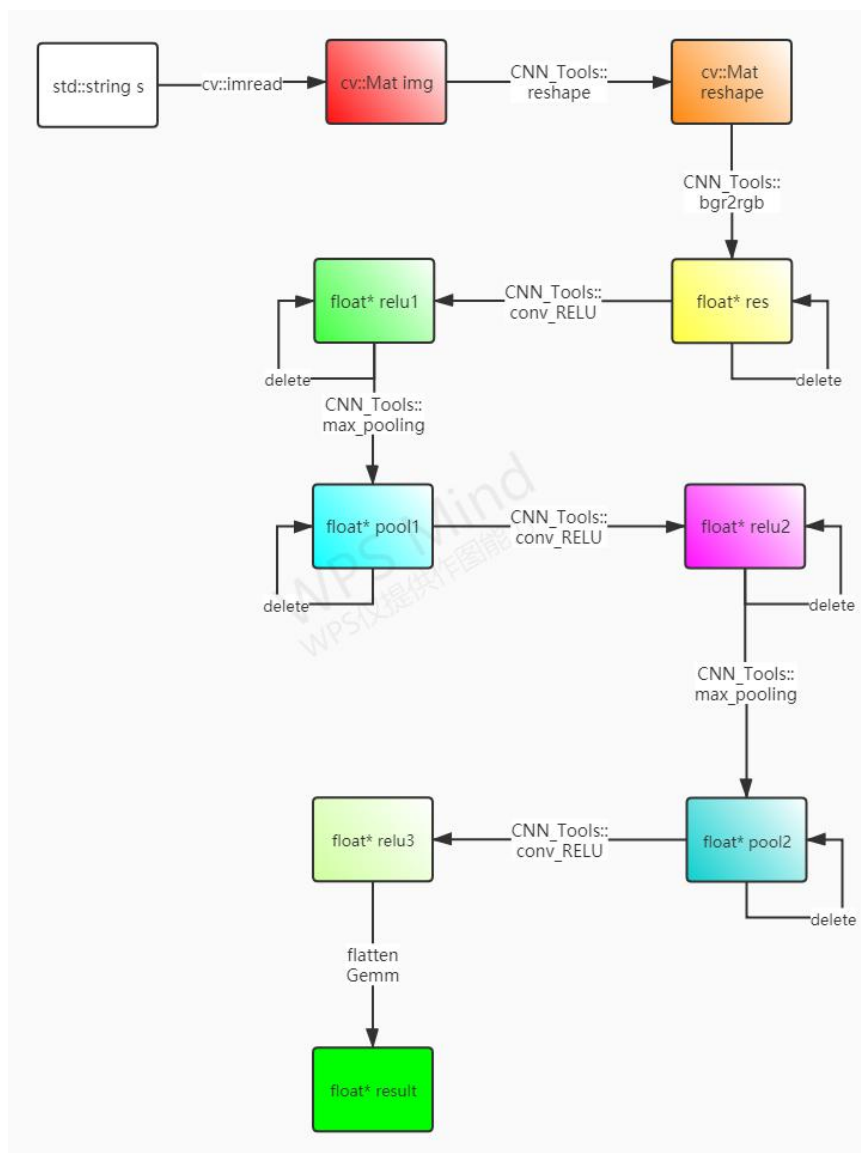
线程之间的通讯亦需要额外时间，导致运算速度变慢。

## Part 4.Memory management

在针对 CNN 算法的实现过程中，其中有一个很令人头大的事情就是内存管理与释放。虽然在 Assignment4 中已经动手实现了一个矩阵类用于自动化的内存管理，但是在这个 Project 当中，内存管理仍然是一个难点。以下类似页面在本次 project 的制作当中多次出现，而且多在 `system("PAUSE")`处或者 `return 0` 的时候出现：



为此，特意设计了以下的卷积运算过程中的数据传递与内存管理流程图从而方便理解：



内存管理与数据流向的流程图

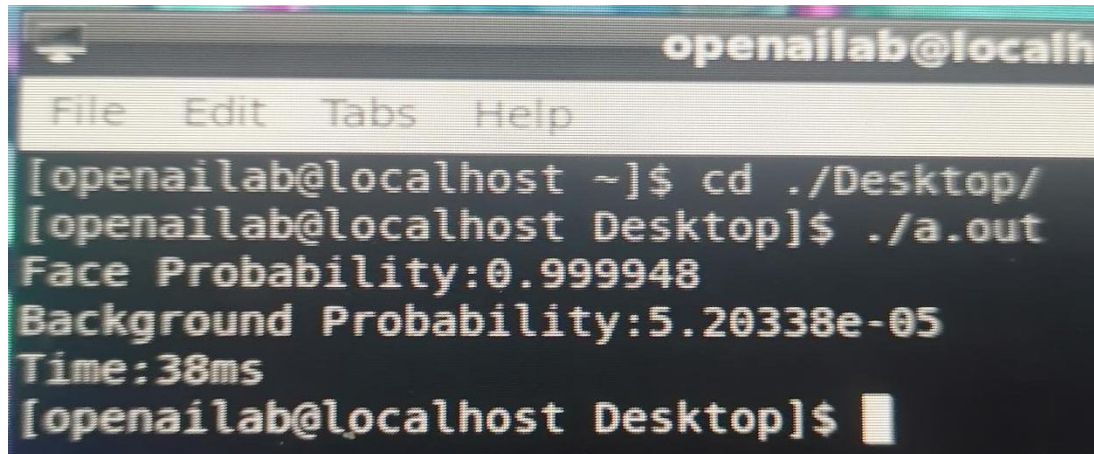
简而言之，就是数据传入在用完后立即进行回收，避免后续内存管理的问题，实现高效的数据管理。

## Part 5.Test on Different Platform

【Linux 系统】ARM 开发板上 2 个标准测试用例的运行结果：

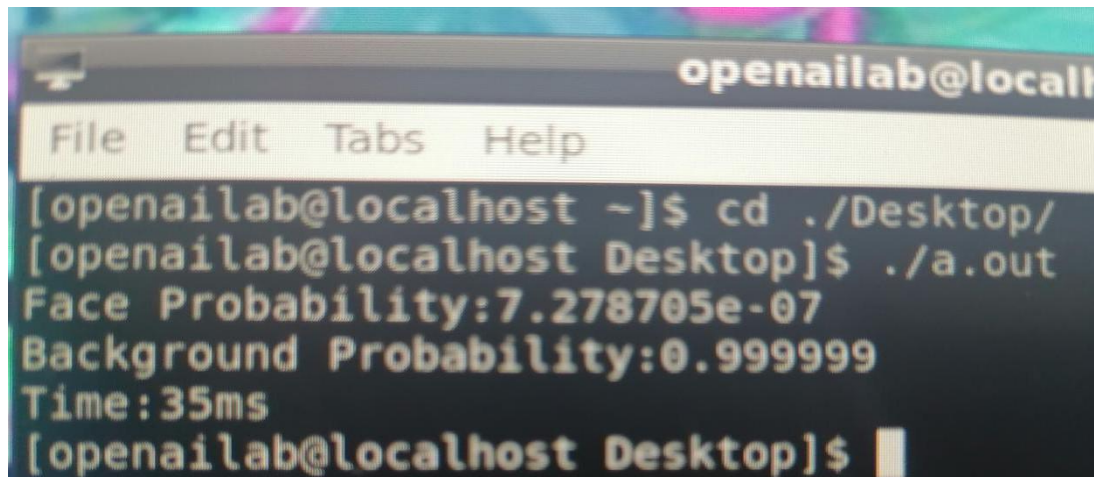
测试文件：0.jpg（face.jpg）





```
openailab@localh
File Edit Tabs Help
[openailab@localhost ~]$ cd ./Desktop/
[openailab@localhost Desktop]$ ./a.out
Face Probability:0.999948
Background Probability:5.20338e-05
Time:38ms
[openailab@localhost Desktop]$
```

测试文件：1.jpg（bg.jpg）



```
openailab@localh
File Edit Tabs Help
[openailab@localhost ~]$ cd ./Desktop/
[openailab@localhost Desktop]$ ./a.out
Face Probability:7.278705e-07
Background Probability:0.999999
Time:35ms
[openailab@localhost Desktop]$
```

从以上 SimpleCNNbyCPP 中提供的标准测试图像来看，

- ①两个测试文件的运行结果与 Windows10 平台相同，验证了本程序的跨平台运算的正确性；
- ②经过多次的时间测量，发现测试文件在 ARM 开发板的运行时间要长于 Windows 系统。这一部分是源于 ARM 开发板的内存没有本机 Windows10 系统大，导致运算性能受到了限制。

## Part 6.To Dos in the future

本次 project 由于临近期末，压力较大时间较紧，有很多头脑中的构想没有来得及在本次 CNN 实现的 project 展现出来，现将未来的 TO DO list 列出如下所示：

1. 增加更加完善的自定义的内存引用回收机制，实现内存空间的合理申请与释放；（该构想在 Assignment4 中已有实现，但是还未能来得及 merge 起来）
2. 增加对于不同平台下指令集（如 AVX，SSE 等）的使用，在保持平台兼容性的同时提高运算的效率；
3. 增加本程序的 JNI 接口实现，从而可以利用结合 Java 语言的跨平台优势和 C/C++语言的高效运算效率的优点，未来有可能移植至 Android 平台；
4. 增加对于 CUDA 接口的使用从而调用 GPU 进行更加高效地运算数据；

## Part 7.Summary

本次 CS205:C/C++程序设计课程的期末 project 是 Simple CNN 的 C++实现。本次期末 project 发布时感觉难度较大，后续经过于仕琪老师的理论课介绍与 B 站视频讲解，我发现 CNN 实现并没有想象得那么困难。正如于老师所说：之前作业和期中 project 都是为了这个 CNN 的实现做的铺垫；这个 project 没有那么难，但是很好地体现了 C++语言的特性。在做期末 project 的过程中，我认为我学到了很多以往课程里没有的内容：

1. C++语言的指针指向内存的问题的理解；
2. C++语言的局部变量的意义的认识；
3. C++语言的内存的使用与回收；
4. C++语言的运算的高效性；
5. C++语言在不同平台下的运算效率差异；
6. C++语言的不同平台下的指令集；
7. C++的类与对象的理解与使用；
8. opencv 库的安装、编译与使用；
9. CNN 的基本算法原理与实现；
10. VS 软件的各种报错的排错能力；
11. 面对自己写出的 bug 的耐心以及调试能力；

学期总结：本学期于仕琪老师对于 C/C++课程的教学别具一格，是 CS205 课程的一大创新。这个课程很好地培养了同学们的动手实践能力，大大扩展了同学们的知识面，涉及多方面的知识如计算机操作系统，计算机组成原理等等。我相信这一学期的 C++课程的学习对于我未来在计算机系的学习，将会大有裨益。