CS205 C/ C++ Program Design - Assignment 4

个人和项目信息

Project Author: 吕昊泽

Student ID ： 11912814

Project Compiling Environment

OS: window10 X86-64

CPU: Inter I7 8700H(12) 2.39GH

Total Memory: 15389MiB

Project Developing Environment:

IDE/Editor: Clion 2020.2.1 Visual Studio 2017

Compiler: Visual Studio 2017

GitHub :

https://github.com/Antoniano1963/vscode_cpp/tree/master/cpp_window/Week15/Final_project

项目要求

基本任务要求

根据给定的权重参数和网络结构，构建一个 CNN 正向计算的网络，用 Opencv 读入图片，通过计算得到正确的值

1（20 Points）实现卷积类，支持 3*3 过滤器，且可自选步长和填充

2（30 Points）程序可以根据给定的参数正确的输出结果

3(20 Pionts) 优化您的实现并在报告中引入它。一些比较、分析和结论是值得欢迎的。

4(5 Points) 程序可在 ARM X86 上运行

5（5 Points） 代码放在 Github 上

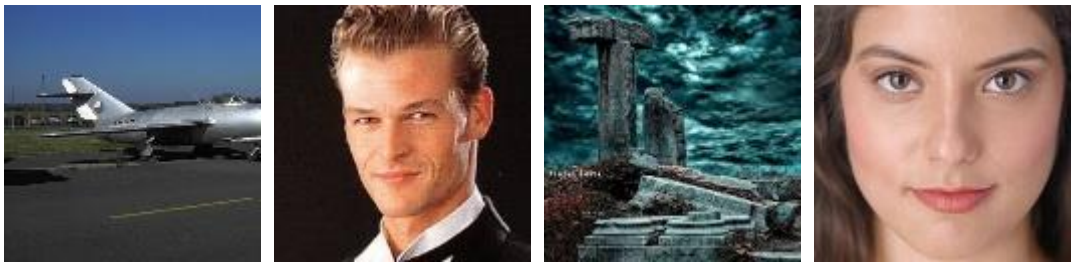
6（5 Points）report 不能超过 15 页，10 页最佳

实现和优化

项目结果展示

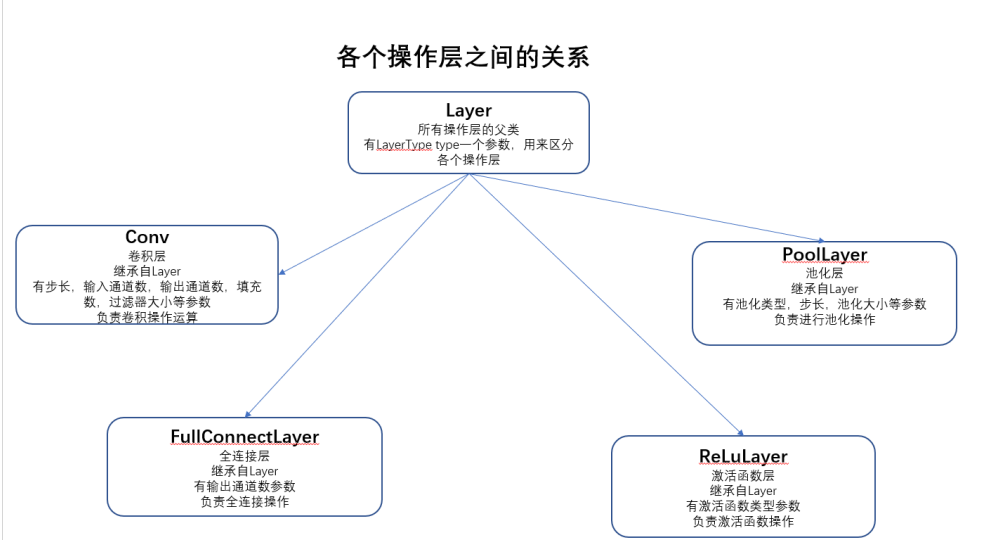
```
source from: C:\cpp\Clion\Final_project\bg.jpg
bg score: 1.000000, face score: 0.000000.
source from: C:\cpp\Clion\Final_project\face.jpg
bg score: 0.000000, face score: 1.000000.
source from: C:\cpp\Clion\Final_project\samples\bg02.jpg
bg score: 0.999882, face score: 0.000118.
source from: C:\cpp\Clion\Final_project\samples\face_woman00.jpg
bg score: 0.178820, face score: 0.821180.
```

输入所示，读取的四张图片依次是

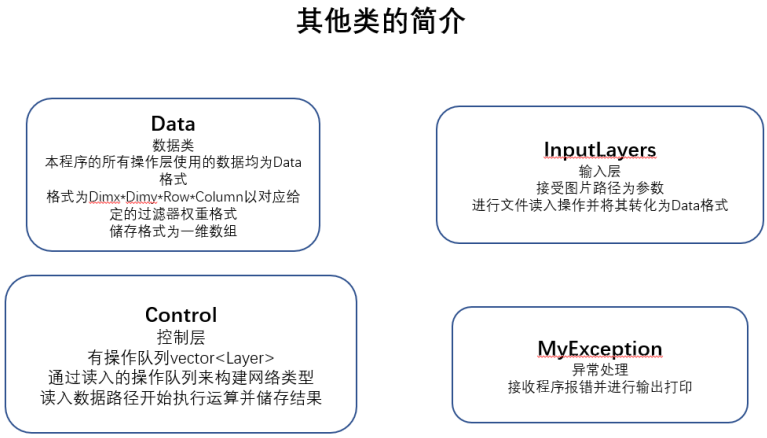


上行为图片路径，下行为 CNN 模型计算出的结果

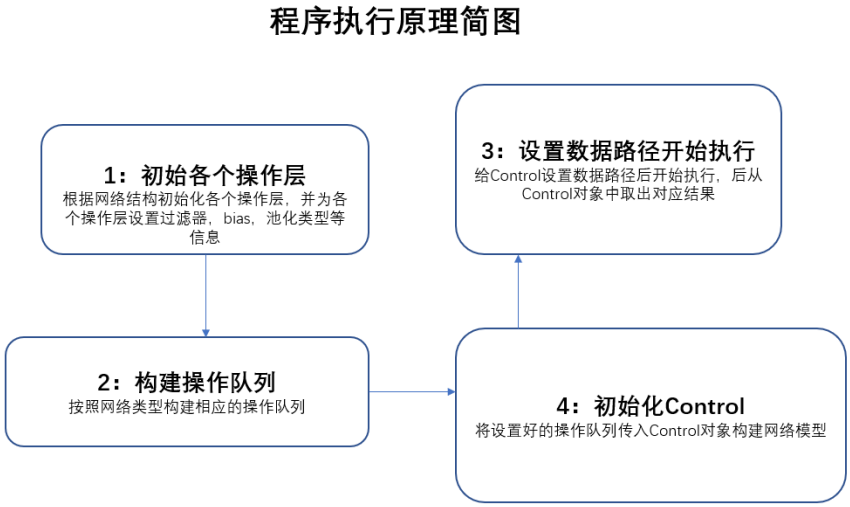
项目中各个 class 之间的层级关系



其他类介绍



程序执行原理图



各个类的具体介绍

Layer 类

```
6   #include "Data.h"
7   template <typename T>
8   class Layer{
9   public:
10      enum LayerType
11      {
12          POOL, CONV, RELU, FUCN, UNKNOW
13      };
14      LayerType type;
15      Layer(LayerType type);
16      LayerType getType(){return this->type;};
17      virtual void startPrograming1(){return;};
18      virtual Data<T> startPrograming(){
19          return Data<T>(1,1,1,1);
20      };
21      virtual void setInputData(Data<T> * inputData){return;};
22      virtual T * startPrograming2(){
23          return new T (0);
24      };
25
26  };
```

所有操作类的父类，确定了操作类型，并使得操作队列得以实现

Line 7:使用了模板

Line 10-13:使用了 enum 类型来定义各个层的关系，其中 UNKNOW 用来检验报错

Line 14:操作层类型

Line 15:构造方法，将传入的 tpye 作为对象的 type

Line 16:返回 type

Line 17:虚函数，用来让层开始工作，为没有返回返回值的层使用（ReLu）

Line 18:虚函数，让返回 Data 的层开始工作（Conv,Pool）

Line 21:虚函数，将数据传入操作层

Line 22:虚函数，让返回 T*的层开始工作(FullCN)

Conv 类

```
8   template <typename T>
9   class Conv:public Layer<T>{
10  private:
11      int in_channels;
12      int out_channels;
13      int kernel_size;
14      int stride;
15      int padding;
16      Data<T> *inputData;
17      Data<T> *Filter;
18      T *bais;
19  public:
20      Conv(int in_channels,int out_channels,int kernel_size,int stride, int padding);
21      void setInputData(Data<T> *inputData);
22      void setFilter(Data<T> *Filter);
23      void setBais(T *bais);
24      Data<T> startPrograming();
25      T getConvNum(int currentChannel, int currentRowNum, int currentColumnNum);
26      int getPadding();
```

卷积类，通过输入的参数进行卷积操作并将结果返回

Line 8-9:使用模板并继承于 Layer

Line 11-18:各个参数，依次为 输入通道数，输出通道数，过滤器大小，步长，填充数，输入数据，过滤器，偏移量

Line 20:构造方法，设置基本参数，并调用 Layer 构造方法，设置类型为 CONV

Line 21：虚函数具体实现，设置输入数据

Line 22-23:设置过滤器和偏移量

Line 24:虚函数具体实现，让层工作并返回结果

POOL 类

```
8   template <typename T>
9   class PoolLayer: public Layer<T>{
10  private:
11      int size;
12      int stride;
13      Data<T> *inputData;
14      T maxPool(int currentChannel, int currentRow, int currentColumn);
15      T avlPool(int currentChannel, int currentRow, int currentColumn);
16      T minPool(int currentChannel, int currentRow, int currentColumn);
17
18  public:
19      enum Type
20      {
21          MAX, MIN, AVE
22      };
23      Type type;
24      PoolLayer(Type type, int size, int stride);
25      void setInputData(Data<T> *inputData);
26      Data<T> startPrograming();
27  };
```

池化类，支持 3 种不同的池化类型

Line 8-9:使用模板并继承自 Layer

Line 10-13:参数，依次为 池化大小，步长，输入数据

Line 14-16:三种 POOL 函数

Line 19-23:enum 变量，确定池化方式，并储存在 type 中

Line 24:构造函数，并调用 Layer 的构造函数，将类型设置为 POOL

Line 25-26:虚函数具体实现，设置输入数据和让程序开始运行

ReLu 类

```
8   template <typename T>
9   class ReLuLayer: public Layer<T>{
10      Data<T> *inputData;
11      void funcDRELU();
12      void funcRELU();
13  public:
14      enum ReLuType{
15          RELU,DRELU
16      };
17      ReLuType reLuType;
18      void startPrograming1();
19      ReLuLayer(ReLuType reLuType);
20      void setInputData(Data<T> *inputData);
21  };
```

激活函数类，支持两种不同的激活函数

Line 8-9:使用模板，继承自 Layer

Line 10:参数 输入数据

Line 11-12:两个激活函数，RELU 和 DRELU

Line 14-17:enum 类，用来区分激活函数的类型

Line 18 20:虚函数的实现，用来设置输入数据和开始执行

Line 19: 构造方法，确定使用的激活函数的类型，并调用 Layer 类构造方法，设置类型为 RELU

FULLConnect 类

```
10  template <typename T>
11  class FullConnectLayer: public Layer<T>{
12  private:
13      Data<T> * inputData;
14      T * Weights;
15      T * Bais;
16      int channels;
17      int size;
18      T getCurrentChannelResult(int currentChannel);
19  public:
20      FullConnectLayer(int channels,int size);
21      void setInputData(Data<T> * inputData);
22      void setWeights(T * Weights);
23      void setBais(T *bias);
24      T * startPrograming2();
25  };
```

FULLConnect 类继承自 Layer，实现全连接操作，在设置好输入数据，权重和偏移量之后可开始执行返回结果指针。

Line 10-11:使用模板并继承

Line 13-17:参数，依次为：输入数据，全连接使用的权重[channels*size] 偏移量，输出通道数，全连接大小

Line 18:得到全连接计算结果，更新对应 result

Line 20:构造方法，设置输出通道数和全连接大小，同时调用 Layer 构造方法设置类型为 FUCN

Line 21 24:虚函数具体实现，实现数据输入和开始执行

Line 22-23:设置权重和偏移量

DATA 类

```
11  template <typename T>
12  class Data {
13  private:
14      mutex *mu;
15      int Dim_x,Dim_y;
16      int row_num, column_num;
17      long size;
18      T *matrixdata;
19      int *refcount;
20      int padding;
21  public:
22      Data(T *data,int Dim_x=1,int Dim_y=1,int row_num=1,int column_num=1);
23      Data(int Dim_x=1,int Dim_y=1,int row_num=1,int column_num=1);
24      Data(const Data &rhs);
25      ~Data();
26      Data &operator=(const Data &rhs);
27      void Display();
28      T getDataFromPosition(int Dim_x,int Dim_y,int row_num, int column_num);
29      int getRow_num();
30      int getColumn_num();
31      int getDim_x();
32      int getDim_y();
33      int getPadding();
34      void setPadding(int padding);
35      void setDataFromPosition(int Dim_x,int Dim_y,int row_num, int column_num,T value);
36      void addDataFromPosition(int Dim_x,int Dim_y,int row_num, int column_num,T value);
37      long getSize();
38  };
```

本次 Data 类构建方法参考了 as4 中 Matrix 的构建思路，使用了 refcount 记录同一数据使用次数，避免了空间的浪费，同时相同数据的对象使用同一互斥锁以维护 refcount 的改变，样例如下（引用拷贝和析构函数中互斥锁和 refcount 的使用）：这使得在调用多线程时程序不至于崩溃。具体原理和细节详见 Assignment4 报告（参考的 OpenCV 矩阵类的结构）https://github.com/Antoniano1963/vscode_cpp/blob/master/cpp_window/Week12/assignment4/Assignment4%2011912814%20%E5%90%95%E6%98%8A%E6%B3%BD.pdf

```
82  template <typename T>
83  Data<T>::Data(const Data<T> &rhs) {
```

```
92      (*mu).lock();
93      this->refcount = rhs.refcount;
94      *(this->refcount)=(*this->refcount)+1;
95      (*mu).unlock();
```

```
98  Data<T>::~Data(){
99      (*mu).lock();
100     if (*(this->refcount) < 2) {
101         delete[] this->matrixdata;
102         delete this->refcount;
103     }
104     else {
105         (*this->refcount)=(*this->refcount)-1;
106     }
107     (*mu).unlock();
```

Line 11-12: 使用模板

Line 14-20:参数列表 依次为： 互斥锁指针,x, y ,r ,c 维度数，数据总体大小，数据存放地址，引用次数存放地址，填充数

Line 22-23:构造方法，可以通过 T*data 指针构造一个 Data 对象，也可以构造一个对应大小，T 均为 0 的 Data 对象

Line 24 26: 引用复制和=复制 使用互斥锁，并对 T*data 指针直接传递实现空间复用

Line 25:析构方法，使用了互斥锁对 refcount 修改

Line 28:获得对应位置的数据

Line 29-33: 获得各种参数信息

Line 35 36: 修改对应位置的 Data 数据

Control 类

```
15  template <typename T>
16  class Control{
17  private:
18      vector<Layer<T>*> processingQueue;
19      Data<T> inputData;
20      int processNum;
21      T * result;
22  public:
23      Control(int processNum);
24      void setProcessingQueue(vector<Layer<T>*>);
25      void startProcessing(string dataPath);
26      T * getResult();
27  };
```

Control 类是程序模型构建成功后运行的总控类，通过设置执行队列构建不同的网络模型。

Line15-16:使用模板

Line 18-21:参数列表，依次为 操作队列，操作数，和最后的结果

Line 23:构造方法，定义操作数

Line 24:设置操作队列

Line 25:读入数据路径并让模型开始执行，具体实现详见项目优点

Line 26:获得结果

MyException 类

```
11  class MyException:public exception
12  {
13  public:
14      string wrong;
15      MyException(string wrong);
16  };
```

继承自标准 exception 库

Line 14:参数，保存错误信息

Line 15:构造方法，设置错误信息

inputLayer 类

```
12  template <typename T>
13  class InputLayer{
14  private:
15      cv::Mat img;
16      string path;
17  public:
18      InputLayer(string path);
19      Data<T> startPrograming();
20  };
```

输入类，通过读入的相对路径用 OpenCV 读入图片，并将其转化成对应的 Data 格式

Line 12-13:使用模板

Line 15-16:参数列表，依次为读入的图片信息（OpenCV Mat 格式） 数据相对路径

Line 18:构造方法，传入数据相对路径

Line 19:读入数据，并将其转化为 Data 类型

Main 函数结构简述

```
13  int main() {
14
15
16      Data<float> conv_wei0(conv0_weight,16,3,3,3);
17      Data<float> conv_wei1(conv1_weight,32,16,3,3);
18      Data<float> conv_wei2(conv2_weight,32,32,3,3);
19
20
21      Conv<float> conv1(3,16,3,2,1);
22      conv1.setFilter(&conv_wei0);
23      conv1.setBais(conv0_bias);
24      ReLuLayer<float> *reLuLayer0=new ReLuLayer<float>();
25      PoolLayer<float> poolLayer0(PoolLayer<float>::MAX,2,2);
26      Conv<float> conv2(16,32,3,1,0);
27      conv2.setFilter(&conv_wei1);
28      conv2.setBais(conv1_bias);
29      ReLuLayer<float> *reLuLayer1=new ReLuLayer<float>();
30      PoolLayer<float> poolLayer1(PoolLayer<float>::MAX,2,2);
31      Conv<float> conv3(32,32,3,2,1);
32      conv3.setFilter(&conv_wei2);
33      conv3.setBais(conv2_bias);
34      ReLuLayer<float> *reLuLayer2=new ReLuLayer<float>();
35      FullConnectLayer<float> fullConnectLayer0(2,2048);
36      fullConnectLayer0.setWeights(fc0_weight);
37      fullConnectLayer0.setBais(fc0_bias);
38
39
40      vector<Layer<float>*> processQueue={static_cast<Layer<float>*>(&conv1),static_cast<Layer<float>*>(reLuLayer0),static_cast<Layer<float>*>(&poolLayer0),
        static_cast<Layer<float>*>(&conv2),static_cast<Layer<float>*>(reLuLayer1),static_cast<Layer<float>*>(&poolLayer1),static_cast<Layer<float>*>(&conv3),
        static_cast<Layer<float>*>(reLuLayer2),static_cast<Layer<float>*>(&fullConnectLayer0)};
```

Line 16-18:利用给定的数组初始化三个对应的过滤器

Line 21-37:构建各种操作层

Line 40: 构建操作队列

模型构建准备工作结束后

```
47      float *result;
48      try{
49          model1.startProcessing("C:\\cpp\\Clion\\Final_project\\samples\\bg01.jpg");
50      } catch (MyException e) {
51          cout<<e.wrong<<endl;
52
53      }
```

Line 44：构建模型，操作数为 9

Line 45: 设置操作队列

Line 49：设置数据路径并让模型开始执行

执行结束后对结果进行获取并进行打印

```
54      result=model1.getResult();
55      cout << "source from: " << "C:\\cpp\\Clion\\Final_project\\samples\\bg01.jpg" << endl;
56      printf("bg score: %6f, face score: %6f.",exp(result[0])/(exp(result[0])+exp(result[1])),exp(result[1])/(exp(result[0])+exp(result[1])));
57      cout<<endl;
```

Line 54:获取计算结果

Line 55-57:对结果进行处理后打印

程序优化和特色

程序使用了模板类进行构建

```
Control<float>
Data<float>
Conv<float> c
ReLuLayer<float>
PoolLayer<float>
```

整个程序都使用了模板类进行构建，使得程序支持不同的数据类型，用户可以根据自身性能和图片性质需要使用 INT LONG DOUBLE 等数据类型对网络进行计算，而不局限于 float 一种数据类型

程序使用了异常处理类对异常进行管理

程序使用了专门的异常管理类对异常进行管理

```
class MyException:public exception
```

并对运行中可能出现的通道数不匹配等状况进行了异常管理

如 CONV 类中对各种通道不匹配进行了异常管理

```
62          throw MyException("The channel of Filter is not equals to out_channels");
63      }
64      if(Filter->getDim_y()!=inputData->getDim_y()){
65          throw MyException("The Y channel of Filter is not equals to Y channel of inputData");
66      }
67      if(Filter->getRow_num()!=this->kernel_size||Filter->getColumn_num()!=this->kernel_size){
68          throw MyException("The Filter size is not equal to the kernel_size");
69      }
70      if(inputData->getDim_y()!=in_channels){
71          throw MyException("The channel of inputData is not equals to the in_channel");
72      }
```

通过在 Mian 函数中的 try-catch 结构对异常进行管理和打印

```
try{
    model1.startProcessing("C:\\cpp\\Clion\\Final_project\\samples\\face.jpg");
} catch (MyException e) {
    cout<<e.wrong<<endl;

}
```

程序 POOL RELU 层支持多种类型操作

程序在 POOL 和 RELU 层实现中，参考网络上 CNN 网络的相关资料，添加了如平均池化，最小池化等不同的

池化操作，使得程序可以实现更多 CNN 网络的计算需求。

如在 POOL 初始化时指定类型为 MAX

```
PoolLayer<float> poolLayer0(PoolLayer<float>::MAX,2,2);
```

程序通过继承的方式实现了用操作队列的方式构建网络模型

程序参考 python 给予的 model.py 中的队列实现了类似的操作队列结构

```
44  class SimpleCLS(nn.Module):
45      def __init__(self, input_size=128, num_cls=2, phase='train'):
46          super(SimpleCLS, self).__init__()
47
48          self.input_size = input_size
49          self.phase = phase.lower()
50
51          self.backbone = nn.Sequential(
52              ConvBNReLU(3, 16, 3, 2, 1),   # 128 -> 64
53              nn.MaxPool2d(2, 2),            # 64 -> 32
54              ConvBNReLU(16, 32, 3, 1),      # 32 -> 30
55              nn.MaxPool2d(2, 2),            # 30 -> 15
56              ConvBNReLU(32, 32, 3, 2, 1)    # 15 -> 8
57          )
58
59          self.classifier = nn.Sequential(
60              nn.Linear(in_features=32*8*8,
61                        out_features=num_cls,
62                        bias=True)
63          )
64
```

```
vector<Layer<float>*> processQueue={static_cast<Layer<float>*>(&conv1),static_cast<Layer<float>*>(reLuLayer0),static_cast<Layer<float>*>(&poolLayer0),static_cast<Layer<float>
```

通过操作队列构建模型可以更好的更改模型层间结构，关键实现函数介绍如下

```
40  void Control<T>::startProcessing(string dataPath) {
41
42      Data<T> currentData;
43      InputLayer<float> input0(dataPath);
44      currentData=input0.startPrograming();
45      Layer<T> *currentProcessing;
46      for(int i=0;i<this->processNum;i++){
47          currentProcessing=this->processingQueue[i];
48          switch (currentProcessing->type) {
49              case (Layer<T>::LayerType::POOL): {
50                  PoolLayer<T> *currentPool = static_cast<PoolLayer<T> *>(currentProcessing);
51                  currentPool->setInputData(&currentData);
52                  currentData = currentPool->startPrograming();
53                  break;
54              }
55              case (Layer<T>::LayerType::CONV): {
56                  Conv<T> *currentConv = static_cast<Conv<T> *>(currentProcessing);
57                  currentConv->setInputData(&currentData);
58                  currentData = currentConv->startPrograming();
59                  break;
60              }
61              case Layer<T>::LayerType::RELU: {
62                  ReLuLayer<T> *currentRelu = static_cast<ReLuLayer<T> *>(currentProcessing);
63                  currentRelu->setInputData(&currentData);
64                  currentRelu->startPrograming1();
65                  break;
66              }
67              case Layer<T>::LayerType::FUCN: {
68                  FullConnectLayer<T> *currentFuC = static_cast<FullConnectLayer<T> *>(currentProcessing);
69                  currentProcessing->setInputData(&currentData);
70                  result = currentProcessing->startPrograming2();
71                  break;
72              }
73              default: {
74                  throw MyException("Unknown Type of Layer!");
75              }
76          }
77      }
78
```

Line 43-44:程序根据输入的数据路径用 Input 层对数据进行初始画操作，并将其转换成 Data 格式

Line 46-72: 从操作队列中依次读出各步操作，使用 static_cast 实现指针类型转换，然后调用函数完成操作层数据的传入和执行的开始

Line 73:异常管理

程序使用了虚函数

由之前介绍可知，Layer 类中定义了虚函数，在继承的 CONV POOL 类中得到了各自的实现

如 setInputData()

Layer 类中

```
21      virtual void setInputData(Data<T> * inputData){return;};
```

Conv 类中

```
21      void setInputData(Data<T> *inputData);
```

Pool 类中

```
25      void setInputData(Data<T> *inputData);
```

Cmake 和 ARM 开发板的运行

CmakeLists 文件介绍

程序使用了 Cmake 进行代码管理，来实现跨平台的程序使用，CmakLists.txt 文件结构如下图

```
1   cmake_minimum_required(VERSION 3.17)
2   project(Final_project)
3
4   set(CMAKE_CXX_STANDARD 17)
5   set(CMAKE_PREFIX_PATH   C:\\OpenCv_finalProject)
6   set(OpenCV_DIR C:\\OpenCv_finalProject\\x64\\vc15\\lib)
7   find_package(OpenCV REQUIRED)
8   include_directories(${OpenCV_INCLUDE_DIRS})
9
10
11  add_executable(Final_project main.cpp Data.h Conv.h PoolLayer.h ReLuLayer.h InputLayer.h FullConnectLayer.h Layer.h Control.h MyException.cpp MyExcep
12
13  target_link_libraries(Final_project ${OpenCV_LIBS})
```

如图所示：line1:指定了 cmake 最小需求版本

Line2:指定了项目并称

Line5:指定了 OpenCV 工程中 Cmake 文件的路径

Line6:指定了 OpenCV 工程链接库路径

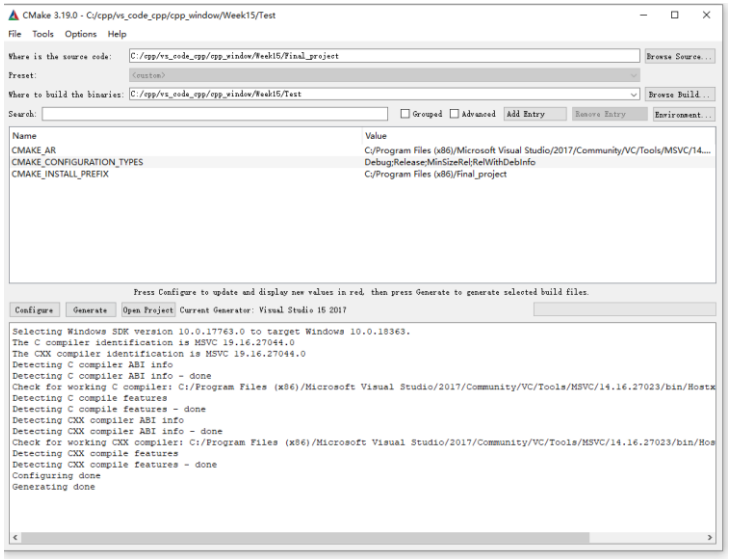
Line8:指定了 include 的路径

Line11:生成可执行程序

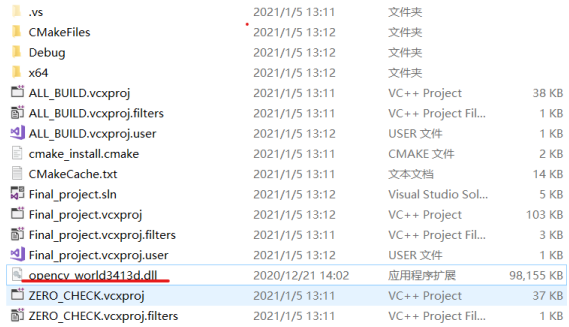
Line13:与指定链接库链接

Windows 具体操作如下

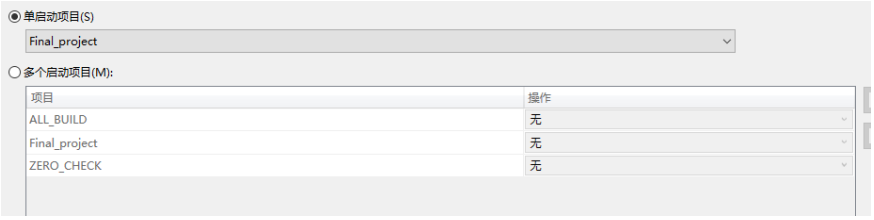
首先使用 Cmake 选择源文件目录和工程生成目录，并选择生成 vs2017 工程，然后进行工程的生成



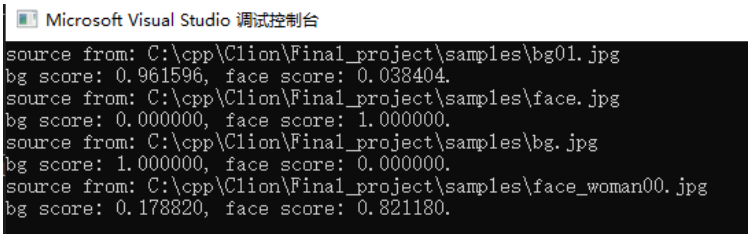
打开工程目录，发现 vs2017 工程已经成功生成，这时需要把 opencv_world3413d.dll 文件复制到工程目录下



随后打开工程，设置单启动项为 Final_Project

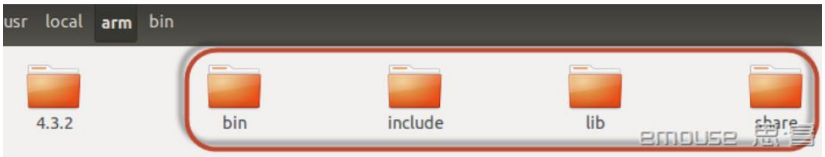


后将启动类型改成 X64 即可成功执行程序

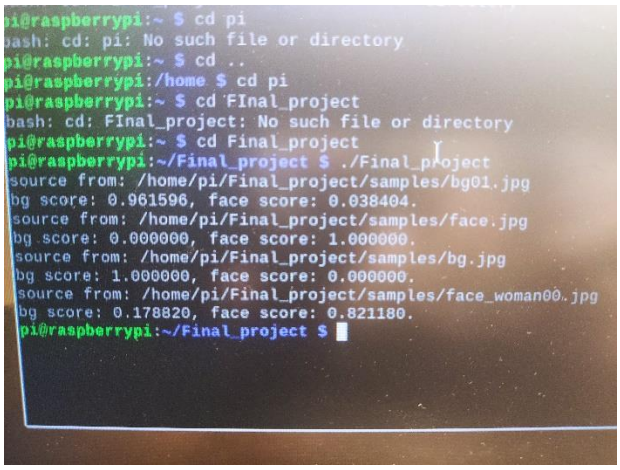


ARM 具体操作如下

在 ARM 开发板上开发和在 Windows 下开发类似，首先在 ARM 开发板上安装 OpenCV，我这里参考 https://blog.csdn.net/lg1259156776/article/details/52344708



安装后得到如下路径，后修改 CmakeLists 文件中的 include 路径，和链接库路径，即可在 ARM 上正常运行程序，结果如下



Part 4 总体概述和心路历程

本次期末 Project，老师选择了一个看似遥不可及的题目，卷积神经网络。最开始拿到这个题目的时候感觉这个题目是那么的遥不可及，但是在查询了相关资料和听了老师的讲解之后，逐渐感觉 CNN 网络的正向传播实际上是我们能够驾驭和执行的。在本次设计过程中，我尝试使用了老师上课讲到的模板类来让自己的程序支持更多的模型，并采用了继承和虚函数的方式实现了操作队列，使得修改模型变得更为简单。在实现运算和数据存储操作的过程中，我参考了 as4 的矩阵类结构，实现了数据的复用。在随后 Cmake 进行项目管理的时候，也由于之前项目的经验使得我很快就解决了生成 VS 和 ARM 开发板上面临的问题。

这次 project 是一个学期的总结，我也尝试着用了这一个学期学到的 C++的各种特性让自己的程序更加完善。