Project 2 -CS205 C/C++ Program Design

Outline

Project 2 -CS205 C/C++ Program Design
- Outline
- 1. Personal and Project Information
  - Attentions:
- 2. Project Requirements
  - Basic Requirements:
  - Requirements:
  - Complete Condition:
- 3. Realization and Optimization
  - overview
  - Class Matrix
    - Function:
    - Structure:
    - Realization and Advantages:
  - Class Pool
    - Function:
    - Structure:
    - Realization and Advantages:
  - Class Conv
    - Structure:
    - Realization and Advantages:
  - Class Full
    - Function:
    - Structure:
    - Realization and Advantages:
  - Class Layer
    - Function:
    - Structure:
    - Realization and Advantages:
  - Class Processor
    - Function:
    - Structure:
    - Realization and Advantages:
  - Others:
- 4. Result
  - Test Result are as follows:
- 5.Conclusion

1. Personal and Project Information

- **Project Author: 刘旭坤**

- **Student ID: 11912823**
- **Project Development Environment:**
    - **Desktop**
        - OS: Windows 10 Pro for Workstation,20H2
        - CPU: AMD Ryzen 9 3950X 16-Core Processor
        - Memory: 80GB
    - **ARM Development Board**
        - OS: Ubuntu 20.10
        - Host: Raspberry Pi 4 Model B Rev 1.4
        - Kernel: 5.8.0-1006-raspi
        - Shell: bash 5.0.17
        - CPU: BCM2835 (4) @ 1.500GHz
        - Memory: 8GB
- **Code Storage Location:** **Github Link**

Attentions:

1. If you are reading this report in **GitHub repository**, because the markdown parsing of GitHub does not cover mathematical formulas and some extended markdown syntax, it is recommended that you download the PDF version or open this report using the Typora application for the best reading experience.
2. In the following report, if there is no special declaration, the compilation and execution environment of all code is the environment declared at the beginning of this report. The project involved in this project does **NOT Guarantee** normal compilation and operation in other environments.
3. To compile and run this project normally, you need to install **OpenCV**.
4. This project uses **CMake** to manage the dependency relationship between files, which needs to be modified before it can be used normally in other environments.
5. This project provides **Two versions**, using **generics** and **function pointers** respectively. The version of generics is used to correspond to the **.h** file, and the function pointer is used to correspond to the **.cpp** file. These two implementations **both** have examples in the main function.

2. Project Requirements

Basic Requirements:

Implement a **convolutional neural network (CNN)** using C/C++.

Requirements:

1. (20 points) The convolutional layer for 3x3 kernels is correctly implemented. It should support stride=1 and stride=2 as well as padding=1.
2. (30 points) The program can output the confidence scores correctly for images. You can take the sample images at *SimpleCNNbyCPP* web site to test.
3. (20 points) Optimize your implementation and introduce it in your report. Some comparisons, analysis and conclusions are welcome.
4. (5 points) The program is tested both on X86 and ARM. It can output the same results for the same inputs on the two platforms.
5. (5 points) Please host your source code at GitHub.com. you can just put a link in the report.

6. (20 points) The report is well organized, and **NOT** longer than 15 pages. It is fine to have ~10 pages. The font size should be ~10.

Complete Condition:

Requirement	Condition
1	Support any size of kernel, stride and pad.
2	The program can output the correct results according to the given parameters. High accuracy on data set from *SimpleCNNbyCPP*.
3	Use generics, template classes, function pointers, enumeration classes,initializer_list... to optimize programs.
4	The program can run on ARM x86.
5	Put code on GitHub.
6	Report cannot exceed 15 pages, 10 pages is best.

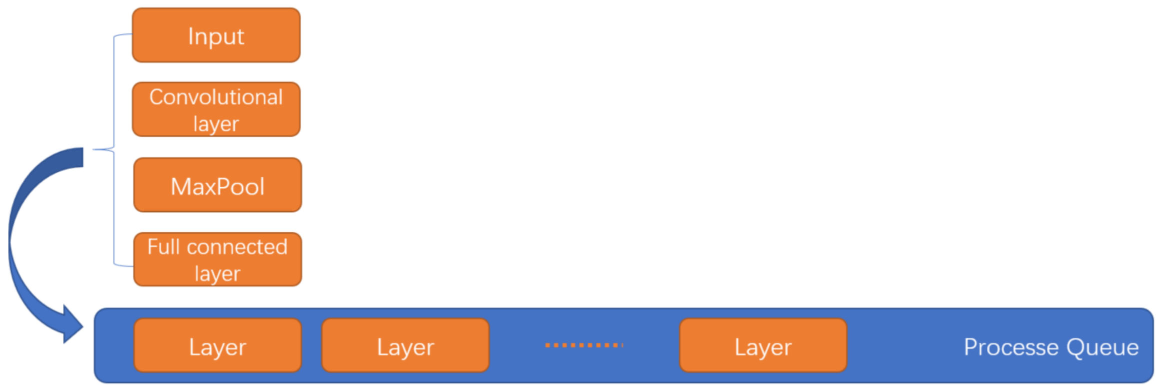
3. Realization and Optimization

overview

The **main structure** of the project is shown in the figure below.

There are four kinds of CNN layers in this project: **Input,Convolutional Layer,MaxPool,Full connected layer**. All of them inherit a common parent class :**Layer** and based on class **Matrix**.

In addition, the project also contains a **Process Queue** for automatic operations.



Next, I will start with the basic class and introduce the whole project in turn.

Class Matrix

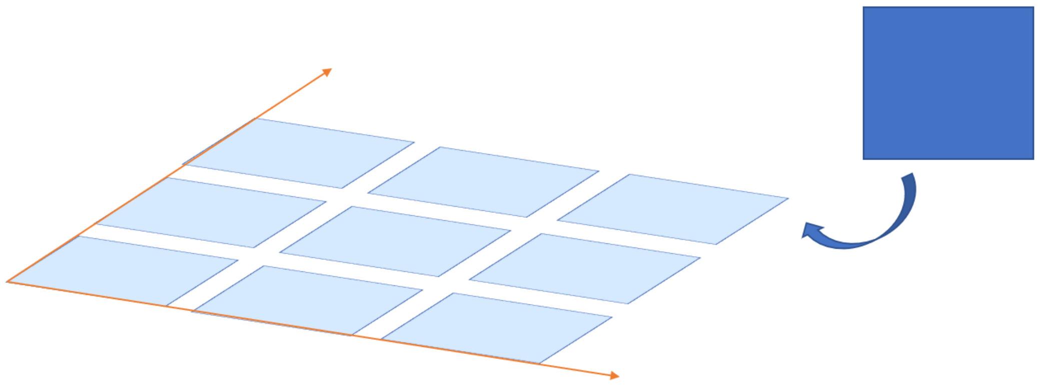
Function:

Encapsulate matrix class to support operation.

Structure:

```
shared_ptr<T[]> matrixs;
int dimX,dimY;
int x,y;
int pad;
```

Matrix class uses C + + **smart pointer**, encapsulates **reference count**, and can intelligently reclaim space. Reduce space consumption. Here I use one-dimensional arrays to describe matrices. Refer to CUDA's 3D grid, block and thread. I finally adopted the following method:



The matrix is two-dimensional distribution in space, with **X and Y** dimensions. This kind of encapsulation saves computing time and makes the data storage of the program more intuitive.

dimX and dimY in the above code describe the dimensions of the matrix.**x. y** describes the size of the matrix, **pad** describes the number of padding zeros of the matrix.

Realization and Advantages:

1. **getPos**

   This function can get the number on the (x, y) position of the matrix whose coordinates are (dimx, dimy). I use mathematical methods for coordinate transformation:

   $$(dimX * DimY + DimY) * (x * y) + y * (x - pad) + (y - pad)$$

2. **Padding**

   This matrix class can change the number of zeros at will under the time complexity of O(1). And it will not increase the additional space consumption. The reason why I go back to the mathematical coordinate transformation is that the following condition is satisfied:

   ```
   if(x<this->pad||y<this->pad||x>=this->x+this->pad||y>=this->y+this->pad)
   ```

3. **template**

   This matrix class uses **template class** and **template functions**. It can meet the different needs of users. It makes the application range wider.

4. **Matrix Matrix::operator=(const Matrix& other)**

   This matrix class **overloads the = operation** and uses **reference passing** instead of value passing. Greatly reduce the space consumption.

   The relevant codes are as follows

```
template <typename T>
Matrix<T> Matrix<T>::operator=(const Matrix& other)
{
    this->dimX=other.dimX;
    this->dimY=other.dimY;
    this->x=other.x;
    this->y=other.y;
    this->pad=other.pad;
    this->matrixs=other.matrixs;
    return Matrix(*this);
}
```

Class Pool

Function:

Realize the pooling operation of matrix. Support multiple pooling

Structure:

```
class Pool: public Layer<T>
{
private:
//    typedef void (*PoolFunc)(Pool<T>&,int,int,Matrix<T>&);
//    friend void funcMax(Pool<T>&,int,int,Matrix<T>&);
//    friend void funcMin(Pool<T>&,int,int,Matrix<T>&);
//    friend void funcAve(Pool<T>&,int,int,Matrix<T>&);
//    PoolFunc func;
    void funcMax(Pool<T>&,int,int,Matrix<T>&);
    void funcMin(Pool<T>&,int,int,Matrix<T>&);
    void funcAve(Pool<T>&,int,int,Matrix<T>&);

public:
    enum Type
    {
        MAX, MIN, AVE
    };
    int pad;
    int DimX,DimY;
    int inX,inY;
    int poolX,poolY;
    int outX,outY;
    Type type;
};
```

In the above code:

Type is an enumeration class, the values are: **MAX, MIN, AVE** They represent maximum pooling, minimum pooling and average pooling respectively.
Pad represents the number of levels of zeroing the input matrix.
inX and inY represent the size of the input matrix.
poolX and poolY represent the pooled size. It can take any value.
outX and outY represent the size of the output matrix.
dimX and dimY represent input and output matrix dimensions.

Realization and Advantages:

1. **Function pointer**

   In order to reduce the types of pooling layer and facilitate user selection and use. This pooling layer uses function pointer, which is similar to **OpenCV's** way of reading pictures. It can select the corresponding algorithm through the constructor, and realize multiple pooling in a class, which greatly reduces the amount of code.

2. **Enumeration class**

   Using enumeration class instead of numerical value to judge the condition makes the user more intuitive when using the function. It also makes the code more beautiful and more standard.

   And this is an Example for these two point:

   When we declare the pooling layer, we can use the following statement:

   ```
   Pool pool0(64,64,2,2,1,16,0,Pool::Type::MAX);
   ```

   That is to say, declare a maximum pooling layer with input matrix size of 64 * 64, pooling size of 2 * 2, input and output dimension of 1 * 16.

   Similarly, if you want to declare a minimum pooling layer, just change the type to **MIN**:

   ```
   Pool pool0(64,64,2,2,1,16,0,Pool::Type::MIN);
   ```

   This function is implemented in the following way. When an illegal value is input, the program will throw an exception.

   ```
       switch (type)
       {
           case MIN:
               this->func = funcMin;
               this->type = MIN;
               break;
           case MAX:
               this->func = funcMax;
               this->type = MAX;
               break;
           case AVE:
               this->func = funcAve;
               this->type = AVE;
               break;
           default:
               throw "Unknown Type of PoolFunction!";
               break;
   ```

```
    }
```

Class Conv

```
#### Function:
```

Conv+BN+Activation function,such as **ReLU**.

Structure:

```
class Conv: public Layer<T>
{
private:
    //typedef void (*ConvFunc)(Conv<T>&);
    //friend void funcReLU(Conv<T>&);
    //friend void funcdReLU(Conv<T>& conv);
    //ConvFunc func;
    void funcReLU(Conv<T>&);
    void funcdReLU(Conv<T>& conv);
public:
    enum Type
    {
        ReLU,dReLU
    };
    Matrix<T>* bias= nullptr;
    Matrix<T>* kernal= nullptr;
    int inX,inY;
    int convX,convY;
    int outX,outY;
    int stride,pad;
    int inDimX,inDimY;
    int outDimX,outDimY;
    Type type;
};
```

In the above code:

Type is an enumeration class, the values are: **ReLU, dReLU** They represent **ReLU and dReLU** activation function respectively.
Pad represents the number of levels of zeroing the input matrix.

stride represents the **step size of convolution kernel movement**.

inX and inY represent the size of the input matrix.
convX and convY represent the kernel size. It can take any value.
outX and outY represent the size of the output matrix.
inDimX and inDimY represent input matrix dimensions.

outDimX and outDimY represent output matrix dimensions.

bias represents the offset.

kernal represents kernel args.

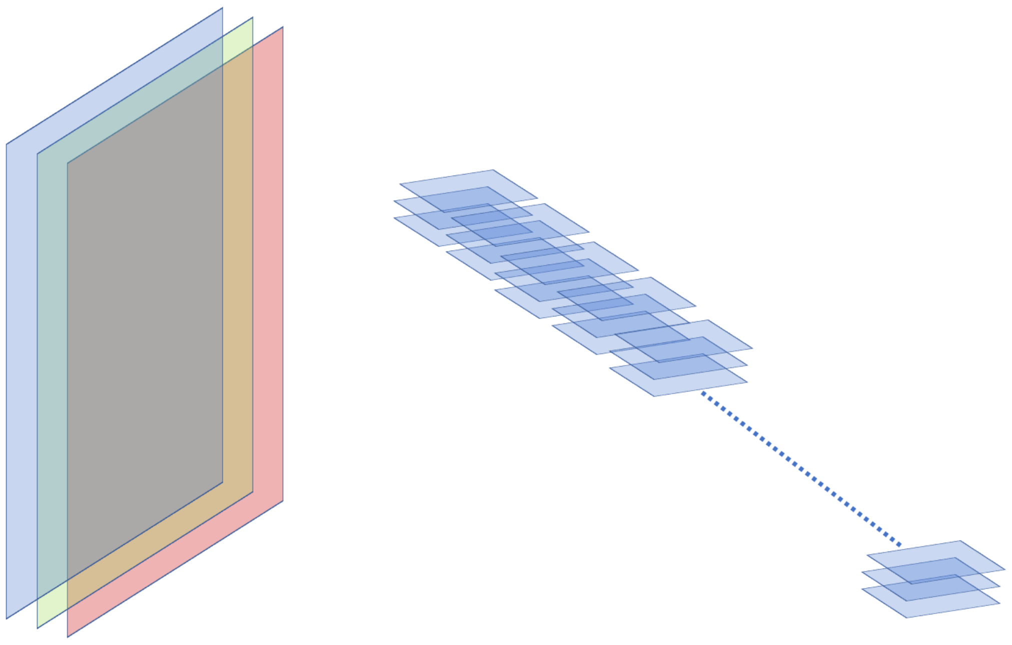
Realization and Advantages:

1. **Convolution**

   Convolution is the core of convolution layer. In my convolution layer, the calculation function of this part is mainly realized by **the following two functions**:

   ```
   void Conv<T>::work(Matrix<T>& input,Matrix<T>& kernal,Matrix<T>& bias)
   void Conv<T>::convolution(Conv<T>& conv,int I,int J,Matrix<T>&
   input,Matrix<T>& kernal,Matrix<T>& bias)
   ```

   The first function is responsible for the **allocation of dimensions**. In other words, this function ensures that the **correct convolution kernel is selected for each dimension of the input**.
   The second function is responsible for **part of the calculation task**, and obtains the convolution result of the corresponding output position through **coordinate transformation**.



In addition, in order to **reduce the consumption of time and space**, the convolution layer **DOES not save parameters and inputs, only the final results are saved.** Parameters and inputs only **record pointers**, which greatly improves the efficiency of the program.

Similarly, due to the **good encapsulation** of **Matrix class** in the bottom layer of the program and the efficient implementation of **Matrix class**, the program **only needs to change the pad value before operation and change it back after operation**. The time consumption is **O(1)**.

The specific implementation of this part is as follows:

```
input.setPad(this->pad);
for(int i=0;i<this->outDimY;i++)
{
    for(int j=0;j<this->inDimY;j++)
    {
        this->convolution(*this,i,j,input,kernal,bias);
    }
}
input.setPad(0);
//this->normalization();
```

```
//this->func(*this);
switch (this->type)
{
    case dReLU:
        this->funcdReLU(*this);
        break;
    case ReLU:
        this->funcReLU(*this);
        break;
    default:
        throw "Error";
        break;
}
```

Class Full

Function:

Fully connected layer.

Structure:

```
template <typename T>
class Full: public Layer<T>
{
private:
    Matrix<T>* kernal;
    Matrix<T>* bias;
    int inDimX,inDimY;
    int inX,inY;
    int outDimX,outDimY;
    int outX,outY;
    int pad;
};
```

In the above code:
Pad represents the number of levels of zeroing the input matrix.

inX and inY represent the size of the input matrix.
outX and outY represent the size of the output matrix.
inDimX and inDimY represent input matrix dimensions.

outDimX and outDimY represent output matrix dimensions.

bias represents the offset.

kernal represents kernel args.

Realization and Advantages:

NULL

Class Layer

Function:

Base of all other layers(**Other layers inherit it**),It provides convenience for the implementation of **task queue**.

Structure:

```
template <typename T>
class Layer
{
public:
    Matrix<T> result;
    enum Type
    {
        Input,Pool,Conv,Full
    };
    Layer();
    Layer(Type type);
    virtual Matrix<T>& getResult();
    virtual void work();
    virtual void work(Matrix<T>& input);
    virtual void setArgs(Matrix<T>* bias,Matrix<T>* kernal);
    virtual void display();
    virtual int getX();
    virtual int getY();
    virtual bool loadFile(string path,cv::ImreadModes type);
    virtual void structure();
private:
    Type type;
};
```

In the above code:

result save calculation results.

Type is an enumeration class, the values are: **Input,Pool,Conv,Full** They represent **Input Layer**,**Pooling Layer** ,**Convolutional Layer** and **Full Connected Layer** respectively.

Realization and Advantages:

1. **The usage of virtual function**

   Subclass implements these virtual functions, which makes subclass interface more **standardized and unified**, and calls more convenient.

   For Example:

   Different layers need different numbers of parameters for operation, so there are many types of **work()** functions of subclasses,such as :

   ```
   void Pool<T>::work(Matrix<T>& input)
   void Conv<T>::work(Matrix<T>& input,Matrix<T>& kernal,Matrix<T>& bias)
   ```

   The use of virtual functions in the parent class makes the interface more **uniform**, so we can rewrite the above functions into the following form:

```
void Conv<T>::work(Matrix<T> &input)
```

It makes function easily to use.

Class Processor

Function:

Combine and evaluate Layers.

Structure:

```
template <typename T>
class Processor
{
private:
    vector<Layer<T>*> Queue;
    Matrix<T> result;
public:
    Processor();
    Processor(std::initializer_list<Layer<T>*> layers);
    void work();
    void clear();
    void push_back(Layer<T>* layer);
    void display();
    void structure();
    Matrix<T>& getResult();
};
```

In the above code:

result save calculation results.

Queue is a **task queue** composed of different layers.

Realization and Advantages:

1. **The usage of initializer_list**

   In order to adapt the program to as many situations as possible, I use **initializer_ list** implements **variable length parameter transmission**. It greatly simplifies the way of building network and code length.

   For Example,You can **declare and run** a network in this simple way:

   ```
   Processor<float> pro={data,conv0,pool0,conv1,pool1,conv2,full};
   pro.work();
   ```

   In addition, compared with **cstdarg**, **initializer_ list is memory safe**, which increases the **stability** of the program.

2. **Storage of Queue**

   It is not difficult to find that the task queue in the program **does not save the layer, but chooses to save its corresponding pointer**. This greatly reduces the time and space consumption of the program.

3. **Task Queue**

All kinds of layers can work with the **same function**, thanks to the layer class's normative effect on its subclasses. Therefore, you only need to transfer the reference of the output of the previous layer to the next layer. The specific code implementation is as follows:

```
template <typename T>
void Processor<T>::work()
{
    int l=this->Queue.size();
    for(int i=0;i<l;i++)
    {
        if(i==0)
        {
            this->Queue[i]->work();
        }
        else
        {
            this->Queue[i]->getResult().clear();
            this->Queue[i]->work(this->Queue[i-1]->getResult());
        }
    }
    this->result=this->Queue[l-1]->getResult();
}
```

Others:

The above description only selects **the core part of each class**. I'll give you a brief introduction to other functions you may use.

1. **get...**

   Get values.

2. **set...**

   Set values.

3. **display**

   Show detail and Struct of a Layer.

4. **Structure**

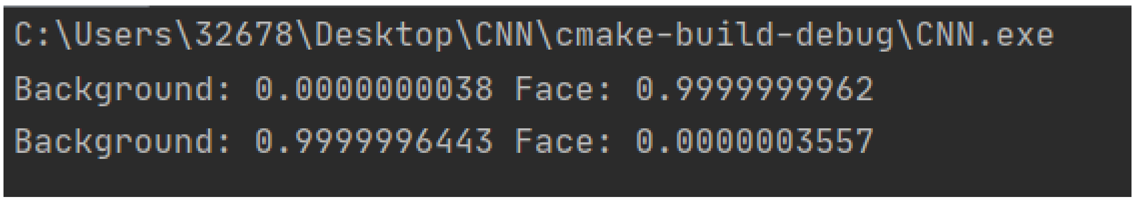
   Show Struct of a Layer.

In addition, the program has some **other advantages**, such as **exception handling**. Due to the **limited space**, I will not repeat it here. Please refer to the code for details.

4. Result

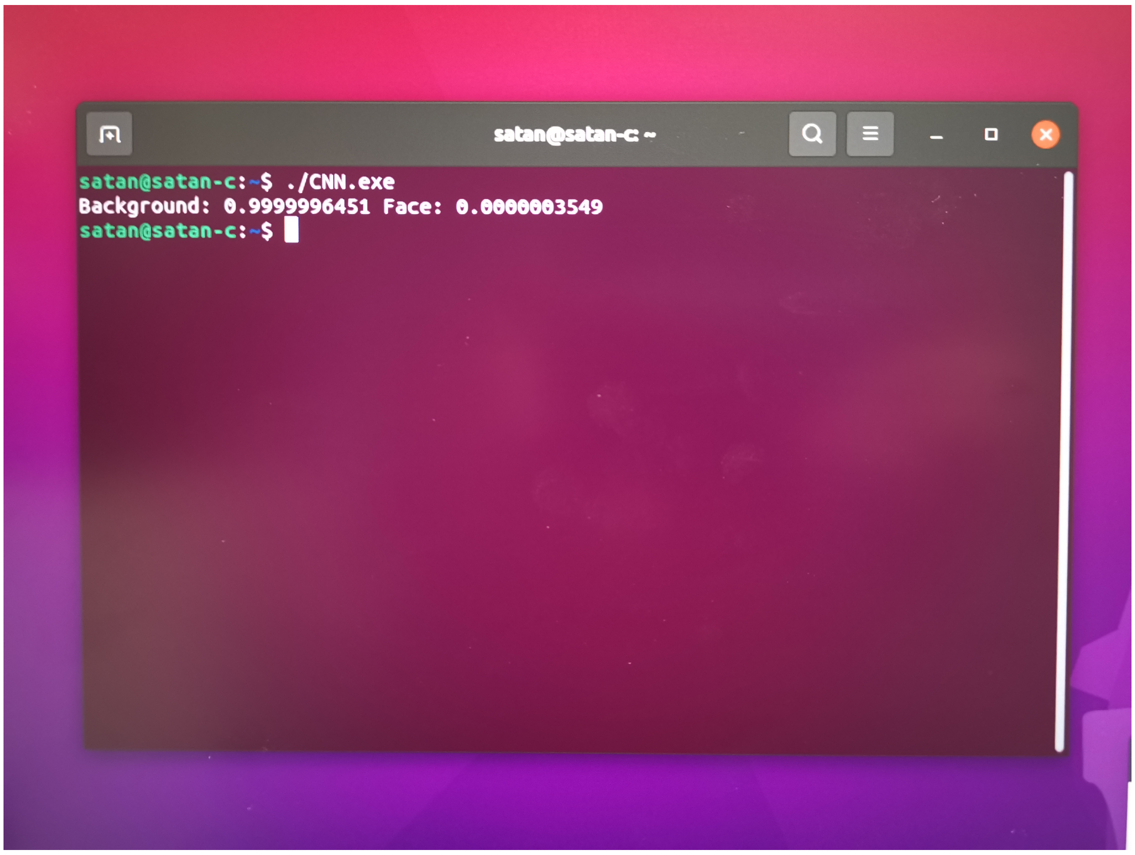
Test Result are as follows:

1. **On Desktop**



```
C:\Users\32678\Desktop\CNN\cmake-build-debug\CNN.exe
Background: 0.0000000038 Face: 0.9999999962
Background: 0.9999996443 Face: 0.0000003557
```

2. **On ARM Development Board**



```
satan@satan-c:~$ ./CNN.exe
Background: 0.9999996451 Face: 0.0000003549
satan@satan-c:~$ 
```

The above shows **only the sample provided by the teacher**, and **other test samples can guarantee the correctness in addition to one picture**:

Picture name	Background	Face
bg.jpg	0.9999996443	0.0000003557
*bg00.jpg	0.2788120093	0.7211879907
bg01.jpg	0.9615962148	0.0384037852
bg02.jpg	0.9998820900	0.0001179100
bg03.jpg	1.0000000000	0.0000000000
bg04.jpg	1.0000000000	0.0000000000
face.jpg	0.0000000038	0.9999999962
face_child_male00.jpg	0.0000000936	0.9999999064
face_man00.jpg	0.0000000000	1.0000000000
face_man01.jpg	0.0070250332	0.9929749668
face_man02.jpg	0.0155021285	0.9844978715
face_woman00.jpg	0.1788201566	0.8211798434
face_woman01.jpg	0.0000743654	0.9999256346
face_woman02.jpg	0.0135080548	0.9864919452

5.Conclusion

It is not difficult to see from the above results that the accuracy of this CNN project can be guaranteed. In addition, the results of this program running on desktop and arm development board are slightly different. It is speculated that it is caused by floating-point error, which does not affect the correctness of the results.