CS205 C/ C++ Program Design

Final Project

Name: 巫晓, **SID**: 11912803

Part 1. Source Code

https://github.com/XiaoLing12138/cppwork/tree/master/P2

Part 2. Result & Verification

Running Environment

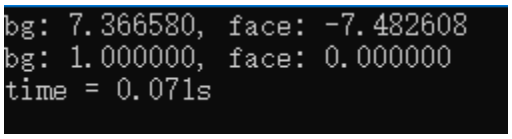
1. Computer: Lenovo LEGION Y700P 2019
2. Windows Version: Windows 10 64bits
3. CPU: Intel(R) Core(TM) i7-9750H @2.60GHz 2.59GHz
4. RAM: 16.0 GB (15.9 GB)
5. GPU: Intel(R) HUD Graphics 630 / NVIDIA GeForce GTX 1660 Ti

The weight and bias are using the trained model provided by my teacher.
The pictures used for testing are copied from the Internet, you are not allowed to use them for profit.

All the pictures are 128*128 and the model contains 3 convLayer, 2 maxPoolingLayer and a FCLayer.

Test case #1:

Run result:



Correct!

Test case #2:

$$\sin\theta=\frac{2\tan\frac{\theta}{2}}{1+\tan^2\frac{\theta}{2}}$$

万能惜逼公式

Run result:

```
bg: 10.057899, face: -10.334530
bg: 1.000000, face: 0.000000
time = 0.08s
```

Correct!

Test case #3

Run result:

```
bg: 0.877054, face: -0.940910
bg: 0.860322, face: 0.139678
time = 0.084s
```

Correct!

Test case #4

Run result:

```
bg: 5.718719, face: -6.029419
bg: 0.999992, face: 0.000008
time = 0.132s
```

Correct!

Test case #5

Run result:

```
bg: 4.675645, face: -4.470786
bg: 0.999893, face: 0.000107
time = 0.084s
```

Correct!

Test case #6

Run result:

```
bg: 1.131992, face: -1.043813
bg: 0.898056, face: 0.101944
time = 0.092s
```

Correct!

Test case #7

Run result:

```
bg: -9.542998, face: 9.854562
bg: 0.000000, face: 1.000000
time = 0.084s
```

Correct!

Test case #8

Run result:

```
bg: -5.143440, face: 5.391820
bg: 0.000027, face: 0.999973
time = 0.088s
```

Correct!

Test case #9

Run result:

```
bg: -11.001136, face: 11.189904
bg: 0.000000, face: 1.000000
time = 0.081s
```

Correct!

Test case #10

Run result:

```
bg: -9.325219, face: 9.502336
bg: 0.000000, face: 1.000000
time = 0.084s
```

Correct!

Test case #11

Run result:

```
bg: -7.062789, face: 7.175097
bg: 0.000001, face: 0.999999
time = 0.085s
```

Correct!

Test case #13

Run result:

```
bg: -6.427634, face: 6.413329
bg: 0.000003, face: 0.999997
time = 0.08s
```

Correct!

Test case #14

Run result:

```
bg: -5.292659, face: 5.521773
bg: 0.000020, face: 0.999980
time = 0.091s
```

Correct!

Test case #15

Run result:

```
bg: -7.625669, face: 7.604208
bg: 0.000000, face: 1.000000
time = 0.099s
```

Correct!

Test case #16

Run result:

```
bg: -1.856470, face: 2.109583
bg: 0.018596, face: 0.981404
time = 0.082s
```

Correct!

Test case #17

Run result:

```
bg: -3.648119, face: 3.831529
bg: 0.000564, face: 0.999436
time = 0.089s
```

Correct!

Test case #18

Run result:

```
bg: -5.136839, face: 5.428952
bg: 0.000026, face: 0.999974
time = 0.091s
```

Correct!

Test case #19

Run result:

```
bg: -2.355864, face: 2.388072
bg: 0.008629, face: 0.991371
time = 0.08s
```

Correct!

Test case #20

Run result:

```
bg: -8.343249, face: 8.087683
bg: 0.000000, face: 1.000000
time = 0.087s
```

Correct!

Test case #21

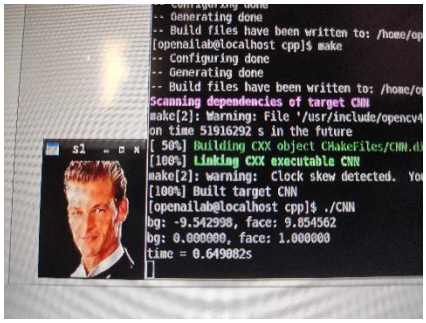
Run result:

```
bg: -2.469889, face: 2.272947
bg: 0.008639, face: 0.991361
time = 0.128s
```

Correct!

Test case #22 in ARM

Run result:



Correct!

Actually, much more pictures are used to test the model and the results are all correct. The CNN model seems to extract a special face feature, which can help it to find a face no matter colorful pictures or grey pictures. The expression or the side face won't make the model goes wrong as well. But according to the

test, I get a glimpse of how it works. From the test case #3 and #6, the face may have a curve and big yellow area. For test case #21, yellow face picture can be considered as a face. So maybe the answer is curve and yellow.

Part 3. Difficulties & Solutions, or others

I have implemented the CNN model in two ways: array version and mat version. So, I will compare two versions in the following report.

By the way, the image is 3*128*128 first. It will decrease to 16*64*64 after the first convLayer, and decrease to 16*32*32 after the first maxPool. 32*30*30 after the second convLayer. 32*15*15 after the second maxPool. 32*8*8 after the third convLayer. In the FCLayer, it will become a 2048-element vector and give the judged results.

First difficult: How to normalize the Mat data which read by OpenCV?

There is a convert function named "convertTo" which can normalization the data from unsigned char to float.

```
image.convertTo(image, CV_32FC3, 1 / 255.0f);
```

The "image" is the read mat and CV_32FC3 means 32 bits floating point number with three channels (BRG), and 1/255.0f is the coefficient.

Second difficult: How to change BGR to RGB?

Because the Mat read by OpenCV is BGR format, it should be changed to RGB because the trained weight and bias are corresponding to RGB.

Array Version:

I simply get the channels data by circulation:

```
for (int i = 0; i < 3; i++)
{
    for (int j = 0; j < 128; j++)
    {
        for (int k = 0; k < 128; k++)
        {
            a[i][j + 1][k + 1] = image.at<Vec3f>(j, k)[2 - i];
        }
    }
}
```

Here I use a function named "at" which can get the elements in the Mat.

For example, image.at<Vec3f>(i, j) will return a 3 elements-vector (for 3 channels, here is BGR). And image.at<float>(i, j) will return the element in row i, col j (if there is only one channel).

Mat Version:

Here I use two new functions "split" and "merge":

```
inline void BGRtoRGB(Mat& image)
{
    vector<Mat> temp;
    split(image, temp);
    swap(temp[0], temp[2]);
    merge(temp, image);
}
```

Split can divide the multiple-channel Mat to single-channel Mat. And merge can combine all single0 channel Mats to a multiple-channel Mat. Because the image I read is three channels Mat (BGR), I can just split it and swap the first Mat and the third Mat. Then I simply merge them.

Third difficult: How to expand channels?

Mat's format is fixed, so I need to find a way to expand channels. For example, from 3 channels to 16 channels.

Array Version:

Just simply use fixed arrays. Simple but useful.

```
float a[3][130][130], b[16][66][66], c[16][34][34],
    d[32][32][32], e[32][17][17], f[32][10][10], g[2050], h[3];
```

Mat Version:

I create a new Mat and split it to store the result. In the last, I merge them and I get an expanded Mat.

```
split(Mat(toRows, toCols, CV_32FC(toChannels)), to);
```

```
merge(to, m);
```

CV_32FC(x) means a x-channel Mat with 32 bits floating point data.

Fourth difficult: How to implement the convLayers with different step or padding?

Different step and padding will cause different index problems.

Array Version:

With a fixed array, I need to write three independent function to implement convLayers. It seems very

complex, but in this way, I can do the work very directly and won't be bother by the index problems.

```
template<typename T>
inline void convLayer1(T nowChannels, T toChannels, T kernelSize, T steps = 1);

template<typename T>
inline void convLayer2(T nowChannels, T toChannels, T kernelSize, T steps = 1);

template<typename T>
inline void convLayer3(T nowChannels, T toChannels, T kernelSize, T steps = 1);
```

Mat Version:

In the mat version, I use some branch statements to control it.

```
template<typename T>
inline void convLayer(Mat& m, T nowChannels, T toChannels
    , T kernelSize, T steps = 1, T padding = 0);
```

```
if (padding == 0)
{
    toRows -= 2;
    toCols -= 2;
}
if (toRows % 2 == 1)
{
    toRows++;
    toCols++;
}
```

```
if (padding == 0)
{
    to[k].at<float>(i - 1, j - 1) = sum;
}
else
{
    to[k].at<float>(i / steps, j / steps) = sum;
}
```

The left one will help me to solve the condition of 16*32*32→32*30*30 and 32*15*15→32*8*8. The right branch statements will help me decide where the result stores.

Fifth difficult: How to add a padding?

Padding is an annoying thing, but the CNN model need padding.

Array Version:

Fixed arrays have a nature padding if you start from index 1.

```
float a[3][130][130], b[16][66][66], c[16][34][34],
    d[32][32][32], e[32][17][17], f[32][10][10], g[2050], h[3];
```

Mat Version:

Mat's one-dimension array need an extra branch to judge whether it needs padding or not.

```
for (T i = 0; i < nowRows; i += steps)
{
    if ((padding == 0) && ((i == 0) || (i == nowRows - 1)))
    {
        continue;
    }
    for (T j = 0; j < nowCols; j += steps)
    {
        if ((padding == 0) && ((j == 0) || (j == nowCols - 1)))
        {
            continue;
        }
```

```
if ((i == 0) && (j == 0)) { ... }
else if ((i == nowRows - 1) && (j == 0)) { ... }
else if (j == 0) { ... }
else if ((i == 0) && (j == nowCols - 1)) { ... }
else if ((i == nowRows - 1) && (j == nowCols - 1)) { ... }
else if (j == nowCols - 1) { ... }
else if (i == 0) { ... }
else if (i == nowRows - 1) { ... }
else { ... }
```

Sixth difficult: How to show the image we test?

In OpenCV, there is a function named "imshow" and "waitKey", which can show the testing pictures.

```
Mat image = imread("./sample/face.jpg");
imshow("s1", image);
```

```
waitKey(0);
return 0;
```

The result is like follow:

Seventh difficult: How to compare the memory cost?

It is easy for us to monitor the memory use by Visual Studio.

 (For face.jpg)

Array Version:

Because it is a static array, so the memory cost is static, 6.1MB.

Mat Version:

Mat is a dynamic data type, and the highest memory cost is 5.6MB.

It is obvious that dynamic memory Mat is better in memory cost.

Eighth difficult: How to compare the time cost?

I use a clock lib to measure time.

```
clock_t start, end;
start = clock();
```

```
end = clock();
cout << "time = " << double(end - start) / CLOCKS_PER_SEC << "s" << endl;
```

Array Version:

```
bg: -9.542998, face: 9.854562
bg: 0.000000, face: 1.000000
time = 0.085s
```

Mat Version:

```
bg: -9.542998, face: 9.854562
bg: 0.000000, face: 1.000000
time = 0.599s
```

It seems that array version will faster. That's because I use memory to exchange time. Dynamic memory control needs much more time than fixed arrays.

Ninth difficult: How to use the given trained data?

This question bothers me for a long time because it is the first time for me to implement this model, and the information provided isn't clear. Thus, I spend hundreds of time on it.

1. For Mat's data with n channels. The first n elements are for 0 row, 0 col and channel-1 to channel-n. The next n elements are for 0 row, 1 col and channel-1 to channel-n. And so on.
2. For the conv0_weight, its size is 16*3*3*3, and it is a one-dimensional array. That means the first 3 elements are the row-0, channel-0, kernel-0. Then, row-1, channel-0, kernel-0 and so on.
3. The conv0_bias has 16 elements for each kernel result.
4. For fc0_weight, the first 2048 elements are for the first result, and the last 2048 elements are for the second result.

Tenth difficult: How to increase the flexibility?

My program is flexible and you can change the dimension to fix a new input picture. Because I have only the fixed weight and bias, I do not write more layers. But as long as I have the weight and bias, I can add more layers and different kernel size.

Array Version:

Fixed arrays need more work to add a new layer.

```
template<typename T>
inline void convLayer1(T nowChannels, T toChannels, T kernelSize, T steps = 1);

template<typename T>
inline void convLayer2(T nowChannels, T toChannels, T kernelSize, T steps = 1);

template<typename T>
inline void convLayer3(T nowChannels, T toChannels, T kernelSize, T steps = 1);

template<typename T>
inline void maxPool1(T row, T col, T nowChannels);

template<typename T>
inline void maxPool2(T row, T col, T nowChannels);

template<typename T>
inline void FCLayer(T nowChannels);
```

Mat Version:

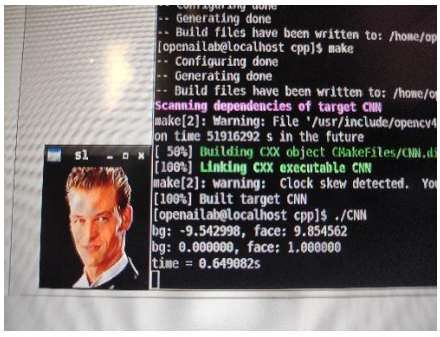
For the Mat one, you just need change the parameters to add a new layer. More flexible.

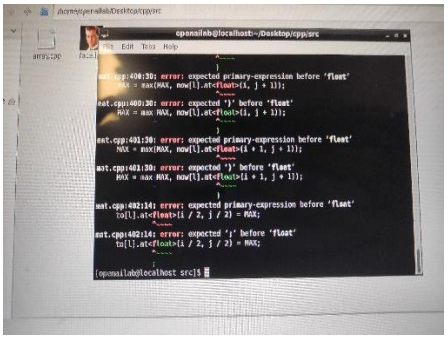
```
template<typename T>
inline void convLayer(Mat& m, T nowChannels, T toChannels
    , T kernelSize, T steps = 1, T padding = 0);

template<typename T>
inline void maxPool(Mat& m, T row, T col, T nowChannels);

inline void FCLayer(Mat& m);
```

Tenth difficult: How to run the program in ARM-Board?

ARM-Board: EAIDK-310

I use a CMakeList and OpenCV (4.2.0). First, it needs to install the OpenCV. This is a hard work but not the key points. Second, there is a detail that the image file's path is relative to CMakeList.txt. The Array Version can run correctly, but the Mat Version doesn't. Maybe it is the compiler's fault!





Conclusion:

Array version is faster but the Mat version is more flexible. If I want to accelerate, I may use OpenBlas or CUDA.