CS205 C/ C++ Programming - Project 5

Name: 王宇琛

SID: 12011001

Part 1 - Analysis

1.1 Goal

To implement a simple CNN using C++ and optimize it.

1.2 Some methods to optimize

- SSE
- NEON(by using [DLTcollab/sse2neon: A translator from Intel SSE intrinsics to Arm/Aarch64 NEON implementation (github.com)]())
- O3

1.3 Some discoveries(analyzed in Part 3)

Here I consider Time consumption is the time used in Conv_0 layer, because Conv_0 and Conv_1 consume most of the time and I just choose Conv_0.

- Time consumption of x86 is less than arm in average
- Time consumption of arm is more steady than x86
- Time consumption of SSE and NEON optimization are longer than Normal version(maybe only on my PC)
- O3 optimization is super useful to accelerate the program

Part 2 - Code

2.1 Main.cpp

```
#include"CNN.hpp"

int main(){

    std::string img_path = "./samples/face.jpg";
    //freopen("CNN_out.txt","w",stdout);

    cv::Mat img_in = cv::imread(img_path,cv::IMREAD_UNCHANGED);

    //Check if the image has been read and continuous(in memory)
    //-----------------------------------------------------------------
    if( img_in.empty() ){

        std::fprintf( stderr, "No image input!\n" );
        return 0;
```

```
    }

    else{

        std::printf("Image has been read!\n");

        if( img_in.isContinuous() ){

            printf("Mat is continuous!\n");

        }

    }
    //-----------------------------------------------------------------


    clock_t start,end;


    Matrix in(img_in);


    //Conv_0
    start = clock();
    in.conv(0);
    end = clock();
    printf("Time used in conv_0:
%.6fms\n",1000*difftime(end,start)/CLOCKS_PER_SEC);

    //Maxpool_0
    start = clock();
    in.maxpool();
    end = clock();
    printf("Time used in max_pool_0:
%.6fms\n",1000*difftime(end,start)/CLOCKS_PER_SEC);

    //Conv_1
    start = clock();
    in.conv(1);
    end = clock();
    printf("Time used in conv_1:
%.6fms\n",1000*difftime(end,start)/CLOCKS_PER_SEC);

    //Maxpool_1
    start = clock();
    in.maxpool();
    end = clock();
    printf("Time used in max_pool_1:
%.6fms\n",1000*difftime(end,start)/CLOCKS_PER_SEC);

    //Conv_2
    start = clock();
    in.conv(2);
    end = clock();
    printf("Time used in conv_2:
%.6fms\n",1000*difftime(end,start)/CLOCKS_PER_SEC);
```

```
    //Flatten
    start = clock();
    in.flatten();
    end = clock();
    printf("Time used in flatten:
%.6fms\n",1000*difftime(end,start)/CLOCKS_PER_SEC);

    //Full Connect
    start = clock();
    in.fc();
    end = clock();
    printf("Time used in full_connect:
%.6fms\n",1000*difftime(end,start)/CLOCKS_PER_SEC);

    //Softmax
    in.softmax();


    printf("%dx%dx%d\n\n",in.channel,in.row,in.col);



    for(int c=0;c<in.channel;c++){
        for(int i=0;i<in.row;i++){
            for(int j=0;j<in.col;j++){

                printf("%.6f\n",in.data[i*in.col+j+c*in.row*in.col]);

            }
            // std::cout<<std::endl;
        }
        std::cout<<std::endl;
    }


    //fclose(stdout);


    return 0;
}
```

2.2 CNN.hpp

*The annotated codes in **void conv(int id)** are using SSE*

Here I just optimize Conv layers because they consume most of the time.

```
#pragma once

#include<opencv2/opencv.hpp>
#include<iostream>
#include<cmath>
#include<ctime>
//#include "sse2neon.h" // When this header is called, the next two header should
be annotated ; This header is used to change SSE to NEON on arm
```

```
#include<xmmintrin.h>
#include<emmintrin.h>
#include"face_binary_cls.cpp"
class Matrix{

    public:
        int row,col;
        int channel;
        float *data;

    public:
        Matrix(cv::Mat &in){

            if( in.empty() ){

                fprintf(stderr,"The input Figure is empty!\n");

            }

            else{

                row = in.rows;
                col = in.cols;
                channel = in.channels();
                data = new float[row*col*channel];

                int cnt = 0;
                for(int c=0; c<channel; c++){
                    for(int i=0; i<row; i++){
                        for(int j=0; j<col; j++){
                            //dt = in.at<cv::Vec3b>(i,j);

                            data[cnt] = (float)
(in.data[c+i*col*channel+j*channel]) / 255.0f ;

                            if(data[cnt]<0.0f) data[cnt] = 0.0f;

                            cnt++;
                            //std::cout<<row*col*c+i*col+j<<std::endl;

                        }
                    }
                }

            }


        }
        ~Matrix(){

            if(data != NULL){
                delete []data;
                data = NULL;
            }

        }
```

```
        void conv(int id){


            if(id<0 || id>2){

                fprintf(stderr,"Convulution id is out of range[0,2] !\n");
                return;

            }


            int in_chan,out_chan,pad,stride,k_size;

            pad       =     conv_params[id].pad;
            stride    =     conv_params[id].stride;
            in_chan   =     conv_params[id].in_channels;
            out_chan  =     conv_params[id].out_channels;
            k_size    =     conv_params[id].kernel_size;

            float *bias = conv_params[id].p_bias;
            float *weight = conv_params[id].p_weight;

            int out_row,out_col;
            out_row = (row+2*pad-k_size)/stride +1 ;
            out_col = (col+2*pad-k_size)/stride +1 ;


            float *dt = new float[out_row*out_col*out_chan];

            for(int i=0;i<out_chan*out_col*out_row;i++)dt[i] = 0.0f;

            int cnt = 0;

            for(int out=0; out<out_chan; ++out){

                int begin = out*out_col*out_row;

                for(int in=0; in<in_chan;++in){

                    cnt = begin;

                    // weights
                    // first row
                    float k_oi_00 = weight[out*(in_chan*3*3) + in*(3*3) + 0];
                    float k_oi_01 = weight[out*(in_chan*3*3) + in*(3*3) + 1];
                    float k_oi_02 = weight[out*(in_chan*3*3) + in*(3*3) + 2];
                    // second row
                    float k_oi_10 = weight[out*(in_chan*3*3) + in*(3*3) + 3];
                    float k_oi_11 = weight[out*(in_chan*3*3) + in*(3*3) + 4];
                    float k_oi_12 = weight[out*(in_chan*3*3) + in*(3*3) + 5];
                    // third row
                    float k_oi_20 = weight[out*(in_chan*3*3) + in*(3*3) + 6];
                    float k_oi_21 = weight[out*(in_chan*3*3) + in*(3*3) + 7];
                    float k_oi_22 = weight[out*(in_chan*3*3) + in*(3*3) + 8];


                    // __m128 k_00_to_10 =
_mm_set_ps(k_oi_00,k_oi_01,k_oi_02,k_oi_10);
```

```
                    // __m128 k_11_to_21 =
_mm_set_ps(k_oi_11,k_oi_12,k_oi_20,k_oi_21);
                    // __m128 k_22 = _mm_set_ps(k_oi_22,0.0f,0.0f,0.0f);


                    for(int i=0-pad; i<row+pad-2; i+=stride){
                        for(int j=0-pad; j<col+pad-2; j+=stride){


                            float dt00 = 0.0f, dt01 = 0.0f, dt02 = 0.0f;
                            float dt10 = 0.0f, dt11 = 0.0f, dt12 = 0.0f;
                            float dt20 = 0.0f, dt21 = 0.0f, dt22 = 0.0f;


                            if(i>=0 && j>=0 && i<row && j<col)    dt00 =
data[i*col+j+in*col*row];
                            if(i>=0 && j>=-1 && i<row && j<col-1) dt01 =
data[i*col+j+1+in*col*row];
                            if(i>=0 && j>=-2 && i<row && j<col-2) dt02 =
data[i*col+j+2+in*col*row];

                            if(i+1>=0 && j>=0 && i<row-1 && j<col)    dt10 =
data[(i+1)*col+j+in*col*row];
                            if(i+1>=0 && j>=-1 && i<row-1 && j<col-1) dt11 =
data[(i+1)*col+j+1+in*col*row];
                            if(i+1>=0 && j>=-2 && i<row-1 && j<col-2) dt12 =
data[(i+1)*col+j+2+in*col*row];

                            if(i+2>=0 && j>=0 && i<row-2 && j<col)    dt20 =
data[(i+2)*col+j+in*col*row];
                            if(i+2>=0 && j>=-1 && i<row-2 && j<col-1) dt21 =
data[(i+2)*col+j+1+in*col*row];
                            if(i+2>=0 && j>=-2 && i<row-2 && j<col-2) dt22 =
data[(i+2)*col+j+2+in*col*row];

                            // __m128 dt_00_to_10 =
_mm_set_ps(dt00,dt01,dt02,dt10);
                            // __m128 dt_11_to_21 =
_mm_set_ps(dt11,dt12,dt20,dt21);
                            // __m128 dt_22 = _mm_set_ps(dt22,0.0f,0.0f,0.0f);


                            // __m128 result_00_to_10 =
_mm_mul_ps(k_00_to_10,dt_00_to_10);
                            // __m128 result_11_to_21 =
_mm_mul_ps(k_11_to_21,dt_11_to_21);
                            // __m128 result_22 = _mm_mul_ps(k_22,dt_22);

                            // float
buffer_00_to_10[4],buffer_11_to_21[4],buffer_22[4];

                            // _mm_store_ps(buffer_00_to_10,result_00_to_10);
                            // _mm_store_ps(buffer_11_to_21,result_11_to_21);
                            // _mm_store_ps(buffer_22,result_22);

                            dt[cnt] += k_oi_00*dt00 + k_oi_01*dt01 +
k_oi_02*dt02;
```

```
                            dt[cnt] += k_oi_10*dt10 + k_oi_11*dt11 +
k_oi_12*dt12;
                            dt[cnt++] += k_oi_20*dt20 + k_oi_21*dt21 +
k_oi_22*dt22;

                            // dt[cnt] +=
buffer_00_to_10[0]+buffer_00_to_10[1]+buffer_00_to_10[2]+buffer_00_to_10[3];
                            // dt[cnt] +=
buffer_11_to_21[0]+buffer_11_to_21[1]+buffer_11_to_21[2]+buffer_11_to_21[3];
                            // dt[cnt++] += buffer_22[3];


                        }
                    }


                }

                float bias_oi = bias[out];

                for(int i=begin; i<cnt; i++)dt[i] += bias_oi;



            }

            // The next is to relu
            for(int i=0;i<out_chan*out_col*out_row;i++)dt[i] =
std::max(0.0f,dt[i]);

            // To change the data pointer
            delete []data;
            data = NULL;
            data = dt;
            row = out_row;
            col = out_col;
            channel = out_chan;

        }


        void maxpool(){

            if(!row || !col){

                fprintf(stderr,"The row or col is 0\n");
                return;

            }

            int out_row = row/2;
            int out_col = col/2;

            int stride = 2;

            float *dt = new float[channel*out_row*out_col];

            int cnt = 0;
```

```
            for(int c=0;c<channel;c++){
                for(int i=0;i<row;i+=stride){
                    for(int j=0;j<col;j+=stride){
                        float dt00,dt01,dt10,dt11;

                        dt00 = data[i*col+j+c*col*row];
                        dt01 = data[i*col+j+1+c*col*row];
                        dt10 = data[(i+1)*col+j+c*col*row];
                        dt11 = data[(i+1)*col+j+1+c*col*row];

                        // printf("dt00:%.9f,dt01:%.9f\n",dt00,dt01);
                        // printf("dt10:%.9f,dt11:%.9f\n",dt10,dt11);

                        dt[cnt++] = std::max( std::max(dt00,dt01),
std::max(dt10,dt11) );
                    }
                }
            }


            delete []data;
            data = NULL;
            data = dt;
            row = out_row;
            col = out_col;

        }



        void flatten(){

            int len = channel*col*row;
            channel = 1;
            col = 1;
            row = len;

        }


        void fc(){

            int out_features,in_features;

            out_features = fc_params[0].out_features;
            in_features = fc_params[0].in_features;

            float *weight = fc_params[0].p_weight;
            float *bias = fc_params[0].p_bias;

            float *dt = new float[out_features];

            for(int i=0;i<out_features;i++) dt[i] = 0.0f;

            int cnt = 0;

            for(int out=0; out<out_features; ++out){
```

```
                for(int in=0; in<in_features; ++in){

                    float w_oi = weight[out*in_features + in];
                    dt[cnt] += w_oi * data[in];

                }
                dt[cnt++] += bias[out];
            }


            delete []data;
            data = NULL;
            data = dt;
            row = out_features;
            col = 1;

        }


        void softmax(){

            float sum = 0.0f;

            for(int i=0;i<row*col;i++){

                sum += exp(data[i]);

            }

            for(int i=0;i<row*col;i++){

                data[i] = exp(data[i])/sum;

            }

        }


};
```

Part 3 - Result & Detail Analysis

3.1 Result

3.1.1 Result of CNN face check

- **sample 1:**

background: 0.963793

face: 0.036207

- **sample 2:**

background: 0.007086

face: 0.992914

- **sample 3:**

background: 0.999996

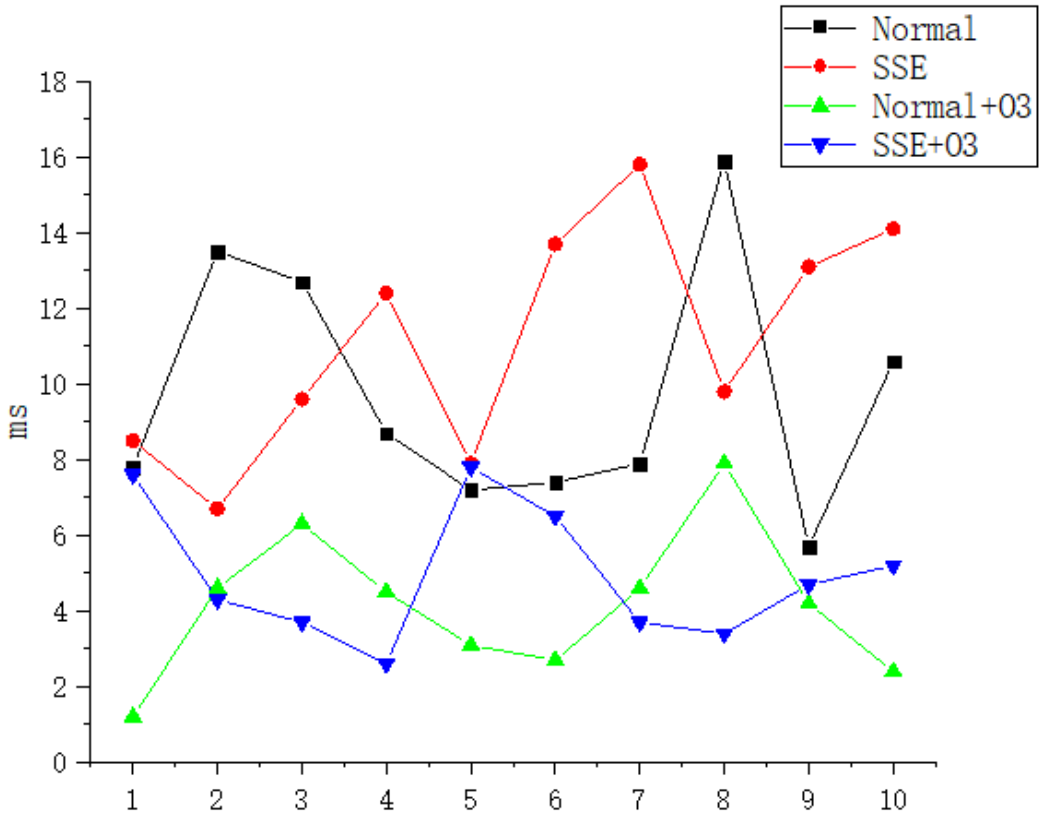
face: 0.000004

3.1.2 Result of Time Consumption(Only Conv_0 is considered)

- x86:

描述统计

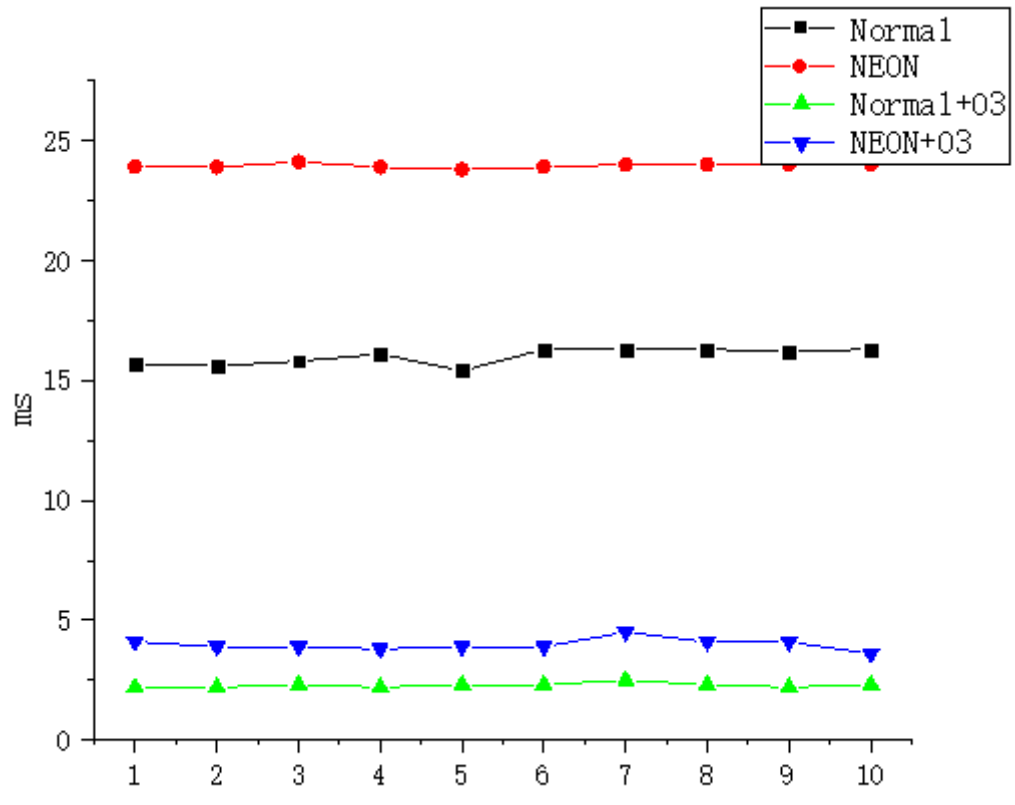
	总数N	均值	标准差	方差	总和	最小值	中位数	最大值
Normal	10	9.74	3.30091	10.896	97.4	5.7	8.3	15.9
SSE	10	11.22	3.09939	9.60622	112.2	6.7	11.1	15.8
Normal+O3	10	4.15	1.94893	3.79833	41.5	1.2	4.35	7.9
SSE+O3	10	4.95	1.79954	3.23833	49.5	2.6	4.5	7.8



- arm:

描述统计

	总数N	均值	标准差	方差	总和	最小值	中位数	最大值
Normal	10	16	0.34319	0.11778	160	15.4	16.15	16.3
NEON	10	23.95	0.08498	0.00722	239.5	23.8	23.95	24.1
Normal+03	10	2.28	0.09189	0.00844	22.8	2.2	2.3	2.5
NEON+03	10	3.98	0.23944	0.05733	39.8	3.6	3.9	4.5



3.2 Detail Analysis

3.2.1 CNN Accuracy

- This model maybe cannot detect a cartoon face, it will get a result that it is background.
- In fact, a model maybe cannot at the same time detect real face and virtual face.

3.2.2 Optimization Analysis

- Time consumption of x86 is less than arm in average

  x86 CPU is much stronger than Arm, therefore, the program speed on x86 is much faster than that on Arm.

- Time consumption of arm is more steady than x86

  After checking the data graph, I find that although the average speed of Arm is slower than x86, it has much more stability than x86.

  Due to the powerful CPU, x86 has more power dissipation. Therefore, the computation ability of x86 is unstable.

- Time consumption of SSE and NEON optimization are longer than Normal version(maybe only on my PC)

  It is strange that using SSE and NEON is slower than Normal computation.

  Maybe that is because my false usage of SSE and NEON, or I at the same time get data from memory and load them into the __m128.

- O3 optimization is super useful to accelerate the program

  It is obvious that the speed using O3 is much faster.

  After I checking some articles, I find that O3 will do better data operation and force to load data into register. It is similar to using SSE, but it is better version or it use SSE better than me.

Part 4 - Summary

- In this project, I learn how to implement a simple CNN.
- CNN is a magic tool that detect face, background. It is used in a number of fields. Trying to implement a simple one is really interesting.