CS205 Project 3

1.思路分析

本次project要求用C语言实现深度学习中的卷积层，并且要求设计一个函数，可以读取输入的图片，根据卷积核和其他参数对原位图进行卷积计算，然后生成新的图片。

项目的关键在于程序的效率，是上一次project的延续，因此可以使用上一节project中学到的方法进行优化，并测量运行时间。此外，根据我的资料查询，在卷积核较大时，卷积的运算可以改用FFT（快速傅里叶变换）进行优化。FFT可以将卷积运算转化为点乘运算，从而大大降低卷积计算的复杂度，运算的时间复杂度将会从O(n^2)优化为O($N*log(N)$)。

2.基础代码及具体实现

基础代码实现分为五大板块。

首先，我定义了$float32$的结构体，结构体中包含高、宽、通道数和储存图像信息的三维矩阵，后续的输入、输出和卷积核都会用到这个结构体。

```
typedef struct
{
    int height;
    int width;
    int channel;
    float ***data;
} float32;
```

其次是图像的读取，我调用了$FreeImage$的库，以此将图片（任何类型）转换为位图，并将相关信息赋给$float32$*类型的变量。

```
// 读取图像
void read_image(char *filename, float32* input)
{
    FIBITMAP *bitmap = FreeImage_Load(FreeImage_GetFIFFromFilename(filename),
filename, 0);
    if (!bitmap)
    {
        fprintf(stderr, "文件读取失败！\n");
    }

    input->height = FreeImage_GetHeight(bitmap);
    input->width = FreeImage_GetWidth(bitmap);
    int bpp = FreeImage_GetBPP(bitmap); // 获取位深
    input->channel = bpp / 8;

    // 为input分配内存（内存在后续free）
    input->data = (float ***)malloc(input->height * sizeof(float **));
    for (int i = 0; i < input->height; i++)
```

```
    {
        input->data[i] = (float **)malloc(input->width * sizeof(float *));
        for (int j = 0; j < input->width; j++)
        {
            input->data[i][j] = (float *)malloc(input->channel * sizeof(float));
        }
    }

    // 将数据分配到data中，data为三维矩阵，从内到外分别为高、宽、通道,通道按R、G、B分开
    RGBQUAD pixel;
    for (int i = 0; i < input->height; i++)
    {
        for (int j = 0; j < input->width; j++)
        {
            FreeImage_GetPixelColor(bitmap, j, i, &pixel);
            input->data[i][j][0] = (float)pixel.rgbRed;
            input->data[i][j][1] = (float)pixel.rgbGreen;
            input->data[i][j][2] = (float)pixel.rgbBlue;
        }
    }
    FreeImage_Unload(bitmap);
}
```

然后是图像的写入，当图像生成的矩阵处理完毕后，则用该函数生成矩阵对应的图像，类型同样可以自定。

```
// 写入图像
void write_image(char *filename, const float32* output)
{
    // 分配内存以储存图像数据
    FIBITMAP *bitmap = FreeImage_Allocate(output->width, output->height, output-
>channel * 8, 0, 0, 0);

    RGBQUAD color;
    color.rgbReserved = 255;
    for (int y = 0; y < output->height; y++) {
        for (int x = 0; x < output->width; x++) {
            color.rgbRed = (BYTE)(output->data[y][x][0]);
            color.rgbGreen = (BYTE)(output->data[y][x][1]);
            color.rgbBlue = (BYTE)(output->data[y][x][2]);
            FreeImage_SetPixelColor(bitmap, x, y, &color);
        }
    }


    // 生成图像
    FreeImage_Save(FreeImage_GetFIFFromFilename(filename), bitmap, filename, 0);
    FreeImage_Unload(bitmap);
}
```

接下来是padding操作，因为当卷积核的大小是3及以上时，原始图像必定会有部分内容无法读取，因此为了确保生成图像与输入图像的大小一致，故需要padding操作。padding填充部分通常为0，但我也写了padding的值是复制相邻色块的情况。padding的大小取决于卷积核的大小，通常为**卷积核大小减1再除2。**

```
// padding操作
void padding(const float32* input, int paddingSize,float32* paddingImage)
{
    // 初始化
    paddingImage->width = input->width + 2*paddingSize;
    paddingImage->height = input->height + 2*paddingSize;
    paddingImage->channel = input->channel;

    // 分配内存
    paddingImage->data = (float ***)malloc(paddingImage->height * sizeof(float **));
    for (int i = 0; i < paddingImage->height; i++)
    {
        paddingImage->data[i] = (float **)malloc(paddingImage->width * sizeof(float
*));
        for (int j = 0; j < paddingImage->width; j++)
        {
            paddingImage->data[i][j] = (float *)malloc(paddingImage->channel *
sizeof(float));
        }
    }

    for(int i = 0; i < paddingImage->height; i++){
        for(int j = 0; j < paddingImage->width; j++){
            for(int k = 0; k < paddingImage->channel; k++){
                paddingImage->data[i][j][k] = 0;
            }
        }
    }
    // 赋值
    for (int i = 0; i < input->height; i++)
    {
        for (int j = 0; j < input->width; j++)
        {
            paddingImage->data[i + paddingSize][j + paddingSize][0] = input->data[i]
[j][0];
            paddingImage->data[i + paddingSize][j + paddingSize][1] = input->data[i]
[j][1];
            paddingImage->data[i + paddingSize][j + paddingSize][2] = input->data[i]
[j][2];
        }
    }

    // // 上边界复制像素值
    // for (int i = 0; i < paddingSize; i++)
    // {
    //     for (int j = 0; j < input->width; j++)
    //     {
```

```
    //         paddingImage->data[i][j + paddingSize][0] = input->data[0][j][0];
    //         paddingImage->data[i][j + paddingSize][1] = input->data[0][j][1];
    //         paddingImage->data[i][j + paddingSize][2] = input->data[0][j][2];
    //     }
    // }
    // // 下边界复制像素值
    // for (int i = input->height + paddingSize; i < input->height + 2 *
paddingSize; i++)
    // {
    //     for (int j = 0; j < input->width; j++)
    //     {
    //         paddingImage->data[i][j + paddingSize][0] = input->data[input->height
- 1][j][0];
    //         paddingImage->data[i][j + paddingSize][1] = input->data[input->height
- 1][j][1];
    //         paddingImage->data[i][j + paddingSize][2] = input->data[input->height
- 1][j][2];
    //     }
    // }
    // // 左边界复制像素值
    // for (int i = 0; i < input->height; i++)
    // {
    //     for (int j = 0; j < paddingSize; j++)
    //     {
    //         paddingImage->data[i + paddingSize][j][0] = input->data[i][0][0];
    //         paddingImage->data[i + paddingSize][j][1] = input->data[i][0][1];
    //         paddingImage->data[i + paddingSize][j][2] = input->data[i][0][2];
    //     }
    // }
    // // 右边界复制像素值
    // for (int i = 0; i < input->height; i++)
    // {
    //     for (int j = input->width + paddingSize; j < input->width + 2 *
paddingSize; j++)
    //     {
    //         paddingImage->data[i + paddingSize][j][0] = paddingImage->data[i]
[input->width - 1][0];
    //         paddingImage->data[i + paddingSize][j][1] = paddingImage->data[i]
[input->width - 1][1];
    //         paddingImage->data[i + paddingSize][j][2] = paddingImage->data[i]
[input->width - 1][2];
    //     }
    // }

    // // 复制四个角的像素值
    // // 左上角
    // for (int i = 0; i < paddingSize; i++) {
    //     for (int j = 0; j < paddingSize; j++) {
    //         paddingImage->data[i][j][0] = input->data[0][0][0];
    //         paddingImage->data[i][j][1] = input->data[0][0][1];
    //         paddingImage->data[i][j][2] = input->data[0][0][2];
    //     }
```

```
    // }
    // // 左下角
    // for (int i = input->height + paddingSize; i < input->height + 2 *
paddingSize; i++) {
    //     for (int j = 0; j < paddingSize; j++) {
    //         paddingImage->data[i][j][0] = input->data[input->height - 1][0][0];
    //         paddingImage->data[i][j][1] = input->data[input->height - 1][0][1];
    //         paddingImage->data[i][j][2] = input->data[input->height - 1][0][2];
    //     }
    // }
    // // 右下角
    // for (int i = input->height + paddingSize; i < input->height + 2 *
paddingSize; i++) {
    //     for (int j = input->width + paddingSize; j < input->width + 2 *
paddingSize; j++) {
    //         paddingImage->data[i][j][0] = input->data[input->height - 1][input-
>width - 1][0];
    //         paddingImage->data[i][j][1] = input->data[input->height - 1][input-
>width - 1][1];
    //         paddingImage->data[i][j][2] = input->data[input->height - 1][input-
>width - 1][2];
    //     }
    // }
    // // 右上角
    // for (int i = 0; i < paddingSize; i++) {
    //     for (int j = input->width + paddingSize; j < input->width + 2 *
paddingSize; j++) {
    //         paddingImage->data[i][j][0] = input->data[0][input->width - 1][0];
    //         paddingImage->data[i][j][1] = input->data[0][input->width - 1][1];
    //         paddingImage->data[i][j][2] = input->data[0][input->width - 1][2];
    //     }
    // }

}
```

然后是最重要的卷积操作，我在此处采用了定义的方法实现卷积运算。经过进一步的资料查询，先前分析所提到的FFT在卷积神经网络中应用较少。因为FFT主要关注全局信息，但卷积神经网络主要关注局部特征，因此FFT的过程可能会造成信息的缺失，因此最终我选择以定义的方法实现。

```
// 卷积操作
void convolution(const float32* input, float32* output, const float32* kernel, int
paddingSize) {
    int kernel_size = kernel->height;

    // padding操作
    float32* paddingImage = (float32*)malloc(sizeof(float32));
    padding(input,paddingSize,paddingImage);

    // 时间计算
    clock_t start, end;
    double cpu_time_used;
```

```
    // 进行卷积操作
    start = clock();
    float sum = 0.0f;
    for (int c = 0; c < output->channel; c++) {
        for (int i = paddingSize; i < input->height + paddingSize - kernel_size + 1;
i++) {
            for (int j = paddingSize; j < input->width + paddingSize - kernel_size +
1; j++) {
                sum = 0.0f;
                for (int k = 0; k < kernel_size; k++) {
                    for (int l = 0; l < kernel_size; l++) {
                        sum += kernel->data[k][l][c] * paddingImage->data[i + k][j +
l][c];
                    }
                }
                output->data[i - paddingSize][j - paddingSize][c] = sum;
            }
        }
    }
    end = clock();

    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("程序运行时间为 %f 秒\n", cpu_time_used);

    // 释放内存
    for (int i = 0; i < input->height + 2 * paddingSize; i++) {
        for (int j = 0; j < input->width + 2 * paddingSize; j++) {
            free(paddingImage->data[i][j]);
        }
        free(paddingImage->data[i]);
    }
    free(paddingImage->data);
}
```

最后是主函数，程序的运行需要在terminal输入输入的文件与输出的文件，然后进行一系列的操作，最后生成相应的图片，再释放所有需要释放的内存。

```
int main(int argc, char *argv[])
{
    if (argc < 3)
    {
        fprintf(stderr, "输入规范: %s 输入文件 输出文件\n", argv[0]);
        return 1;
    }

    float32* input = (float32*)malloc(sizeof(float32));
    // 读取图像并初始化input
    read_image(argv[1],input);

    // 初始化output
```

```
    float32* output = (float32*)malloc(sizeof(float32));
    output->height = input->height;
    output->width = input->width;
    output->channel = input->channel;
    // 分配内存
    output->data = (float ***)malloc(output->height * sizeof(float **));
    for (int i = 0; i < output->height; i++)
    {
        output->data[i] = (float **)malloc(output->width * sizeof(float *));
        for (int j = 0; j < output->width; j++)
        {
            output->data[i][j] = (float *)malloc(output->channel * sizeof(float));
        }
    }

    // 初始化kernel
    float32* kernel = (float32*)malloc(sizeof(float32));
    kernel->height = 5;
    kernel->width = 5;
    kernel->channel = 3;
    // 分配内存
    kernel->data = (float ***)malloc(kernel->height * sizeof(float **));
    for (int i = 0; i < kernel->height; i++)
    {
        kernel->data[i] = (float **)malloc(kernel->width * sizeof(float *));
        for (int j = 0; j < kernel->width; j++)
        {
            kernel->data[i][j] = (float *)malloc(kernel->channel * sizeof(float));
        }
    }
    for(int k = 0; k < kernel->channel; k++){
        // 定义具体kernel，此处略
    }

    // 定义paddingSize
    int paddingSize = (kernel->height - 1) / 2;
    // 卷积操作
    convolution(input,output,kernel,paddingSize);
    // 写入结果
    write_image(argv[2],output);

    // 释放内存
    for (int i = 0; i < input->height; i++) {
        for (int j = 0; j < input->width; j++) {
            free(input->data[i][j]);
            free(output->data[i][j]);
        }
        free(input->data[i]);
        free(output->data[i]);
    }
    free(input->data);
    free(output->data);
```

```
    for (int i = 0; i < kernel->height; i++) {
        for (int j = 0; j < kernel->width; j++) {
            free(kernel->data[i][j]);
        }
        free(kernel->data[i]);
    }
    free(kernel->data);
    free(input);
    free(output);
    free(kernel);

    return 0;
}
```

基础代码部分结束。

3.效率优化

3.1 卷积核大小不同时

当卷积核大小为1x1时，令卷积核RGB三通道的值均为1，得出来的运算时间约为0.035秒。

当卷积核大小为3x3时，同样令所有值为1，运算时间约为0.131秒，为1x1时的4.3倍。

当卷积核大小为5x5时，同样令所有值为1，运算时间约为0.337秒，为1x1时的9.6倍，3x3时的2.6倍。

3.2 其余优化方式

根据课上所学内容与上节课查询所得，还有O3指令优化、openmp指令优化、SIMD指令优化、多核运行等方式，但由于不知名的原因，以上指令均会报错（也许是和库有关），碍于时间限制，无法排查出原因并一一进行测试，故留遗憾于此。