中山大学数据科学与计算机院本科生实验报告

（2020 学年秋季学期）

课程名称：**高性能计算程序设计**　　任课教师：**黄聃**　　**批改人：**

年级+班级	2018 级五班	专业（方向）	计算机科学与技术
学号	18340126	姓名	罗仁良
Email	luorliang@mail2.sysu.edu.cn	完成日期	2021 年 1 月 6 日

1. 实验目的

1） 使用 CUDA 实现卷积运算。

2） 实现 im2col 转换，结合 GEMM 实现卷积运算。

3） 使用 cuDNN 的库实现卷积运算，与自己 CUDA 实现的卷积运算做性能比较。

2. 实验过程和核心代码

2.1 CUDA 实现卷积运算

使用 CUDA 实现卷积运算的思路比较简单，采用二维的线程块，每个线程计算卷积结果矩阵的一个元素，即一个线程进行一次卷积运算，结果为一个数。其中需要解决的问题是，在原矩阵有填充的情况下，给定一个结果矩阵的元素坐标，需要在原矩阵中找到对应位置，使其与卷积核做卷积运算。

2.1.1 坐标映射

经过一定的数学推导，可以得到如下结果：

$$\mathrm{Output(i,j)} = \sum_{0\le k\le h, 0\le l\le w} Input(i\cdot s-p+k, j*s-p+l)\cdot Kernel(k,l)$$

其中$\mathrm{Output(i,j)}$表示卷积结果矩阵的第i行第j列的元素，s表示步长，p 表示填充数。上述公式没有考虑边界情况，在计算的时候，没有实际把矩阵填 0 扩充，所以要排除使得$Input(i\cdot s-p+k, j*s-p+l)$越界的k 和 l。

2.1.2 核函数

```
__global__
void conv2(float *A, float *kernel,
           int inputSize, int depth, int kernelSize,
           int stride, int pad,
           float *B, int outputSize) {

    // 计算元素 output(i,j)的值 一次卷积运算
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    int j = threadIdx.y + blockDim.y * blockIdx.y;

    if( !(i < outputSize) || !(j < outputSize) ) return;

    int Ai = i*stride;
    int Aj = j*stride;
```

```
        // 除去填充的0
        int startk = (pad-Ai) < 0? 0 : pad-Ai;
        int endk = kernelSize < (inputSize + pad - Ai) ? kernelSize : (inputSize +
pad - Ai);
        int startl = (pad-Aj) < 0? 0 : pad-Aj;
        int endl = kernelSize < (inputSize + pad - Aj) ? kernelSize : (inputSize +
pad - Aj);
        float sum = 0;

        for(int d = 0; d < depth; d++) {
            for( int k = startk ; k < endk; k++) {
                for( int l = startl; l < endl; l++) {
                    sum += A[d*inputSize*inputSize + (Ai+k-pad)*inputSize +
Aj+l-pad]*kernel[d*kernelSize*kernelSize + k*kernelSize+l];
                }
            }
            B[d*outputSize*outputSize + i*outputSize + j] = sum;
        }
        B[i*outputSize + j] = sum;
}
```

2.2 使用 im2col 和 GEMM 实现卷积操作

使用了 CUDA 去实现 im2col 的转换过程，其中每一个线程负责把一次卷积运算的元素转换为一个行向量。得到 col 结果后，把 kernel 扩展为一个列向量，使用 GEMM 实现卷积操作。得到的卷积结果也是一个向量。

2.2.1 核函数 im2col

```
    __global__
    void im2col(float *A, int inputSize, int depth, int kernelSize, int stride, int
pad, float *col, int outputSize) {

        // 一个线程完成一次卷积操作中的转换  也就是说 一个线程转换生成 col 中的一个行向量
        int i = threadIdx.x + blockDim.x * blockIdx.x;
        int j = threadIdx.y + blockDim.y * blockIdx.y;

        if( !(i < outputSize) || !(j < outputSize) ) return;

        int Ai = i * stride;
        int Aj = j * stride;

        for( int d = 0; d < depth; d++ ) {
            for(int k = 0; k < kernelSize; k++ ) {
                for( int l = 0; l < kernelSize; l++) {
                    if( Ai + k - pad < 0 || !(Ai + k - pad < inputSize) ||
                        Aj + l - pad < 0 || !( Aj + l - pad < inputSize)) {
                        col[ (i*outputSize + j)*(kernelSize*kernelSize*depth)+
                            d*kernelSize*kernelSize + k*kernelSize + l] = 0;
                    }
                    else  col[ (i*outputSize + j)*(kernelSize*kernelSize*depth)+
                                d*kernelSize*kernelSize + k*kernelSize + l]
                           = A[d*inputSize*inputSize + (Ai + k - pad)*inputSize +
                                Aj + l - pad ];
                }
            }
        }

    }
```

2.2.2 核函数 gemm

```
// 计算 C = A*v A size m*n v size n*1
__global__
void gemm(float *A, float *B, float *C, int m, int n) {

    int i = threadIdx.x + blockDim.x * blockIdx.x;

    if( !( i < m ) ) return;

    float sum = 0;
    for( int l = 0; l < n; l++ ) {
        sum += A[i*n + l] * B[l];
    }

    C[i] = sum;
}
```

2.3 cuDNN 实现卷积操作

调用 cuDNN 实现卷积操作难度不高，主要是理解库中函数和对应参数的意义，网上有很多的教程，结合英伟达官方提供的手册即可完成实验。

cudnnConvolutionForward(…)是进行卷积运算的核心函数。在调用此函数前，还需要进行一系列的初始化准备，如卷积描述符、卷积核描述符等等，正确的初始化之后，调用此函数即可得到卷积结果，最后再把数据传会主机，输出结果。

2.3.1 核心代码

```
    float alpha = 1.0;
    float beta = 0.0;

    for(int i = 0; i < 3; i++ ) {
        checkCUDNN(cudnnConvolutionForward(
                    cudnn[i],
                    &alpha,
                    input_desc,d_A,
                    filter_desc,d_kernel[i],
                    conv_desc[i],algo[i],ws_data[i],ws_size[i],
                    &beta,
                    output_desc[i],d_B[i]));
    }
```

3. 实验结果

3.1 运行结果和以及性能分析

Time(ms) Input size / Method	31	63	127	255	511
CUDA	0.071	0.305	0.401	0.538	1.157
im2col	0.077	0.230	0.385	0.503	1.140
cuDNN	0.238	0.377	0.402	0.941	2.970

注：im2col 的时间只有 GEMM 运算的时间，不包含 im2col 的转化过程耗时

从表中可以看到，im2col 对矩阵的存储进行了一定的优化，相比于 CUDA 的实现有一定的提升。同时可以观察到一个奇怪的现象，cuDNN 的效率没有 CUDA 和 im2col 实现的效率高。个人猜测原因是，cuDNN 库在进行卷积运算时有一定的额外优化，导致了一定的额外开销，在实验比较测试中，输入的矩阵规模比较小，所以导致 cuDNN 的效率较

低。根据此猜测，扩大输入矩阵的规模测试，结果是使用 cuDNN 库的效率是 CUDA 实现的 6 倍左右。由此可见，cuDNN 库在处理大规模的矩阵卷积运算时，效率更高。

3.2 改进方向

从数据存储的空间分布出发，窗口滑动的过程中，相邻的几次卷积操作访问的数据有很多重复，可以使用共享内存进行缓存，减少存储访问的时间开销。同时，由于 GPU 上的线程数量有限，随着矩阵规模的上升，可能出现线程数量不足的情况，此时需要重新设计算法，合理分配每个线程的任务。

4. 实验感想

做完本次实验，初步了解了卷积的知识，结合使用 CUDA 实现，加深了对卷积的认识。同时，进一步熟悉了 CUDA 编程的方法。在任务 3 中，学习了如何调用 cuDNN 的库进行卷积操作，在查阅相关函数的参数过程中，进一步理解了为实现卷积需要的必要参数，指导了我的编程思路。