中国科学院大学计算机组成原理实验课

实 验 报 告

学号：2020K8009907032　姓名：唐嘉良　专业：计算机科学与技术

实验序号：05　　实验名称：深度学习算法与硬件加速器

注 1：撰写此 Word 格式实验报告后以 PDF 格式保存在~/COD-Lab/reports 目录下。文件命名规则：prjN.pdf，其中“prj”和后缀名“pdf”为小写，“N”为 1 至 4 的阿拉伯数字。例如：prj1.pdf。PDF 文件大小应控制在 5MB 以内。此外，实验项目 5 包含多个选做内容，每个选做实验应提交各自的实验报告文件，文件命名规则：prj5-projectname.pdf，其中“-”为英文标点符号的短横线。文件命名举例：prj5-dma.pdf。具体要求详见实验项目 5 讲义。
注 2：使用 git add 及 git commit 命令将实验报告 PDF 文件添加到本地仓库 master 分支，并通过 git push 推送到 GitLab 远程仓库 master 分支（具体命令详见实验报告）。
注 3：实验报告模板下列条目仅供参考，可包含但不限定如下内容。实验报告中无需重复描述讲义中的实验流程。

一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构图、相应信号的仿真波形和信号变化的说明等）

本实验主要操作就是对 3D 图像进行卷积、池化和硬件加速。先说卷积！

关键代码段如下：

```
//TODO: Please add your implementation here
int i=0;
int in_1 = input_fm_w*input_fm_h;
//int out_1 = conv_out_h*conv_out_w;
int weight_1 = rd_size.d1*(1+weight_size.d2*weight_size.d3);
int weight_2 = 1 + weight_size.d2*weight_size.d3;
for(int no=0;no<conv_size.d1;no++){
        int weight_x = no*weight_1;
        //for(int ni=0;ni<rd_size.d1;ni++){
                for(int y=0;y<conv_size.d2;y++){
                        int delta_y = 0;
                        for(int p=0;p<stride;p++){
                                delta_y += y;
                        }
                        //int delta_y = y*stride;
                        for(int x=0;x<conv_size.d3;x++){
                                int temp = 0;
                                int delta_x = 0;
                            for(int q=0;q<stride;q++){
                                        delta_x += x;
                                }
                                //int delta_x = x*stride;
                                //if(ni==0)
                                out[i] = weight[no*weight_1];
                                        //out[no*out_1 + y*conv_out_w + x] = weight[no*weight_1];
                                        //out[no][y][x] = weight[no][0][0];
                                        //initialize output: using bias
```

```
                out[i] = weight[no*weight_1];
                        //out[no*out_1 + y*conv_out_w + x] = weight[no*weight_1];
                        //out[no][y][x] = weight[no][0][0];
                        //initialize output: using bias
                for(int ni=0;ni<rd_size.d1;ni++){
                        int in_x = ni*in_1;
                        int weight_y = ni*weight_2;
                    for(int ky=0;ky<weight_size.d2;ky++){
                                int ih = ky + delta_y - pad;
                                int in_y = ih*input_fm_w;
                            for(int kx=0;kx<weight_size.d3;kx++){
                                    int iw = kx + delta_x - pad;
                                    //int ih = ky + delta_y - pad;
                                    //trough calculating, add "-pad" considering padding
                                    if(iw < 0 || ih < 0 || iw >= input_fm_w || ih >= input_fm_h){
                                        ;
                                            // hit the padding part, adding 0
                                            //that's to say, directly go to the next loop and do nothing to "temp"
                                    }
                                    else{
                                                temp += mul(in[in_x + in_y + iw] , weight[weight_x + weight_y + (1+ky*weight_size.d3+kx)]);
                                                //temp += mul(in[ni*in_1 + ih*input_fm_w + iw] , weight[no*weight_1 + ni*weight_2 + (1+ky*wei
//temp += (int)(in[ni][ih][iw]) * (int)(weight[no][ni][1+ky*weight_size.d3+kx]);
                                        //use int varable "temp" to store mid result
                                    }
                                }
                            }
                        }
                        out[i++] += (short)(((temp >> FRAC_BIT) & 0x7fff) | ((temp >> FIX_POINT_BIT) & 0x8000));
                    }
                }
            //}
        }
    }
```

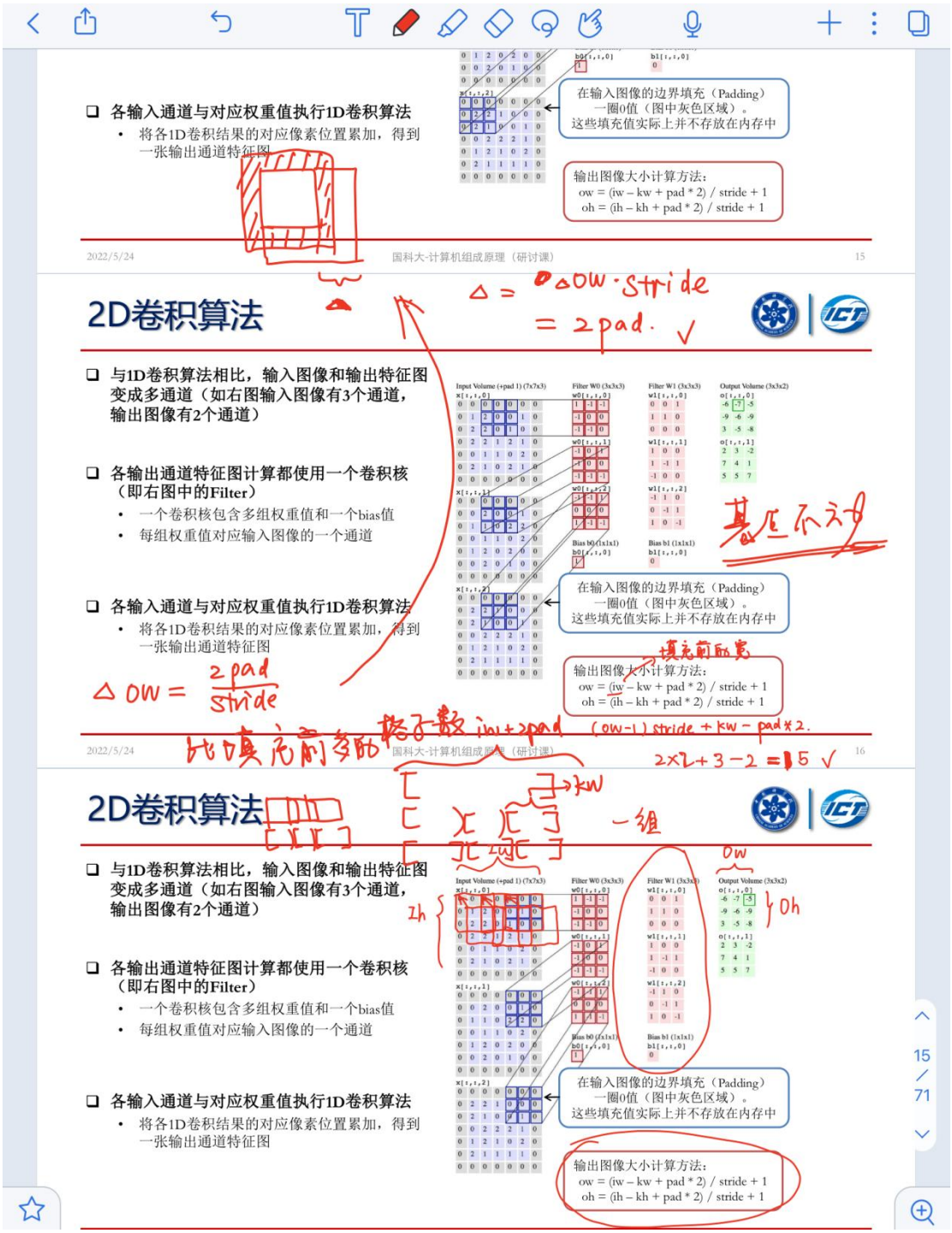
局部注释如下：

```
dering padding
w || ih >= input_fm_h){

g 0
o the next loop and do nothing to "temp"


 + iw] , weight[weight_x + weight_y + (1+ky*weight_size.d3+kx)]);
 ih*input_fm_w + iw] , weight[no*weight_1 + ni*weight_2 + (1+ky*weight_size.d3+kx)]);
ight_size.d3+kx]);
mid result
```

本代码的卷积思路和 PPT 上有所不同，PPT 上是将输出图像轮巡式地填写（观察循环顺序即可知道），而此处我选择了仅仅一次遍历，每次都将输出图像的一格完全填好。这样可以让最后 out 指针寻址偏移量仅需加 1，而非每次还要单独计算一遍偏移量。

```
out[i++] += (short)(((temp >> FRAC_BIT) & 0x7fff) | ((temp >> FIX_POINT_BIT) & 0x8000));
```

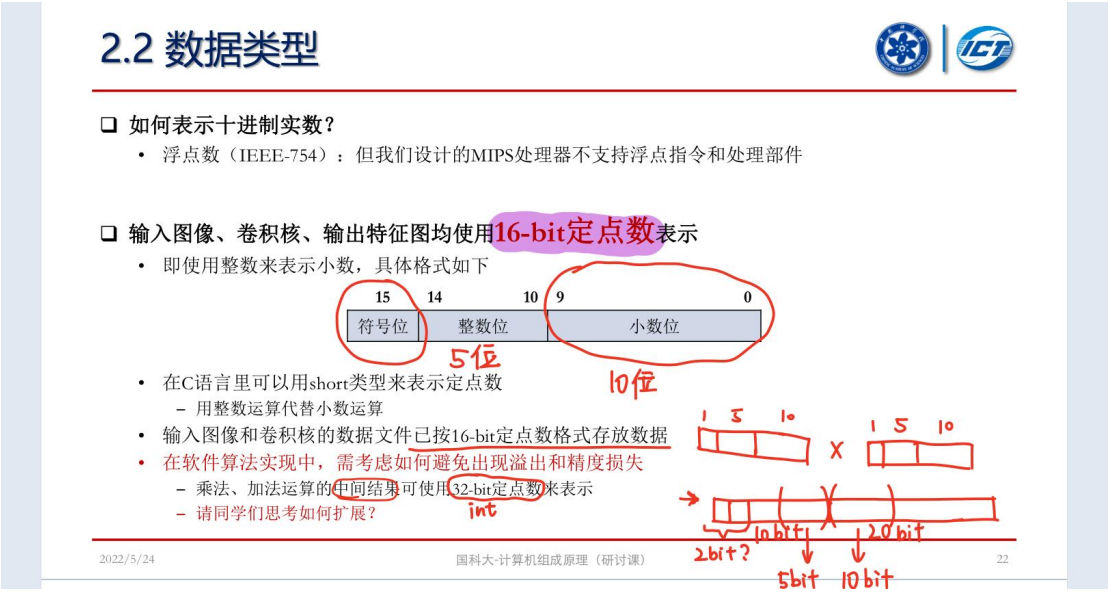
关于 Padding 的计算，画出示意图如上，可以发现（0，0）这一基点事实上没有变动，而填充面临着滑动前后格子覆盖的问题，于是分为两类：滑动时恰好不覆盖和滑动时覆盖，无论哪种，经过推导发现公式均是一致的。下面的截图笔记展示了我的思路。



但是本实验测试用例没有设置 padding，于是关于正确性的验证我也有考虑：我推导出的公式与 PPT 中所给的“输出图像大小计算公式”是自洽的。由此可见这种计算方法具备正确性。于是仅需判定是否落在 0 或者最大边长以外，若是则加 0（也就是不执行任何算数操作）即可。如下。

```
if(iw < 0 || ih < 0 || iw >= input_fm_w || ih >= input_fm_h){
    ;
        // hit the padding part, adding 0
        //that's to say, directly go to the next loop and do nothing to "temp"
}
else{
```

关于数值计算，由于本次实验采取 16bit 定点小数，计算方法较为特殊：



显然地，两个 16bit 定点小数（short）乘积得到 32 位数，我们需要将小数点附近 15 位取出（保持整数与小数位数不变），并将符号位拼接到最高位。这一“截取”的过程可见上面的示意图。

至于池化，则与卷积算法极度相似，只不过将 out 同时作为输入和输出，为此需要设置两个指针。事实上池化 kernel 同卷积 kernel 之理。所以仅需对卷积算法稍加修改即可，甚至还更简单。关键代码段如下：

```
    //TODO: Please add your implementation here
//int i=0;
    int in_1 = input_fm_h * input_fm_w;
    int pool_1 = pool_out_h * pool_out_w;
    for(int no=0;no<conv_size.d1;no++){
            int out_x = no*in_1;
            int out_x_pool = no*pool_1;
            for(int y=0;y<pool_out_h;y++){
                    int delta_y = 0;
                    for(int p=0;p<stride;p++){
                            delta_y += y;
                    }
                    //int delta_y = y*stride;
                    int out_y_pool = y*pool_out_w;
                    for(int x=0;x<pool_out_w;x++){
                            int delta_x = 0;
                        for(int q=0;q<stride;q++){
                                delta_x += x;
                        }
                        //int delta_x = x*stride;
                        short max = 0x8000;  //max is initially the min number of short-varable(signed)
                        for(int ky=0;ky<KERN_ATTR_POOL_KERN_SIZE;ky++){
                                int ih = ky + delta_y - pad;
                                int out_y = ih*input_fm_w;
                                for(int kx=0;kx<KERN_ATTR_POOL_KERN_SIZE;kx++){
                                        int iw = kx + delta_x - pad;
                                        //int ih = ky + delta_y - pad; //in the pooling function, input is the output of convolution
                                        short value;
                                        //trough calculating, add "-pad" considering padding
                                        if(iw < 0 || ih < 0 || iw >= input_fm_w || ih >= input_fm_h){
                                            value = 0;
                                                // hit the padding part, the value is 0, record it
                                        }
                                        else{
                                                value = out[out_x + out_y + iw];
                                                //value = out[no][ih][iw];
                                            // record the value in this lattice
                                        }
                                        if(value > max)
                                            max = value;
                                }
                                out[out_x_pool + out_y_pool + x] = max;
                                //out[no][y][x] = max;
                        }
                    }
            }
    }
```

与卷积最大的区别在于每次要找到每个 kernel 映像的最大值，为此，先设置 short 最小值变量 max = 0x8000，按朴素的找最大值的遍历算法寻找最大值（每个格子的 value 与 max 比较，大则替换）。

本实验我最主要的优化之处在于乘法外提和乘法变循环加和。

乘法外提的效果是比较明显的，毕竟乘法指令相对于加法指令需要更多的 cycle 来完成，所以将循环里要用的乘法语句提出来事先计算好，避免多次的重复计算。乘法外提前后效果对比如下：

```
total cycle is 1735193563
Passed!
benchmark finished
time 103278.95ms
```

（优化前 sw）

```
total cycle is 1488755405
Passed!
benchmark finished
time 100815.99ms
```

（优化后 sw）

```
total cycle is 680597
Passed!
benchmark finished
time 46.77ms
```

（优化前 hw）

```
total cycle is 666736
Passed!
benchmark finished
time 45.50ms
```

（优化后 hw）

```
total cycle is 238592140
Passed!
benchmark finished
time 2413.10ms
```

（优化前 sw_mul）

```
total cycle is 213507907
Passed!
benchmark finished
time 2162.28ms
```

（优化后 sw_mul）

这里存在一个问题：是否该 cycle 性能计数是有意义的？答案是对于 sw 和 hw 来说没有，因为 sw 和 hw 时钟周期数太多，有溢出风险。这也是运行时间与周期数不匹配的原因之一。

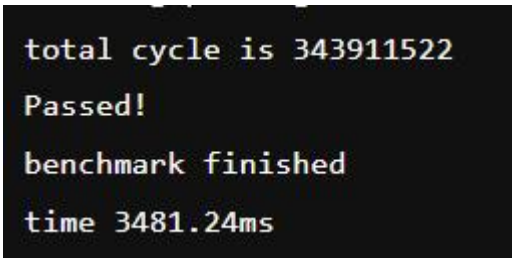
后来我又尝试了乘法改循环加和，结果居然更糟糕了！

```
total cycle is 1049345
Passed!
benchmark finished
time 73.82ms
```

（增加乘法变循环加和优化后 hw）

```
total cycle is -695548389
Passed!
benchmark finished
time 164886.22ms
```

（增加乘法变循环加和优化后 sw，可以看到 cycle 数溢出了）



（增加乘法变循环加和优化后 sw_mul）

根据我的分析，这应当是因为本实验事实上已经将 CPU 的时钟节拍设置成了 100ns/cycle，这导致有时乘法并不能显著快于加法，又考虑到增加循环这一层因素，所需时间反而更多。所以，**尽管这是一个基本且经典的优化手段，本实验参数设置的特殊性也决定了该方法没法完成预期中的优化效果。**

硬件加速算法如下：

```
#ifdef USE_HW_ACCEL
void launch_hw_accel()
{
        volatile int* gpio_start = (void*)(GPIO_START_ADDR);
        volatile int* gpio_done = (void*)(GPIO_DONE_ADDR);

        //TODO: Please add your implementation here
        *gpio_start = (*gpio_start) | 0x1;
        while(1){
                if((*gpio_done & 0x1) == 0x1){
                        break;
                }
        }
}
#endif
```

此处非常简单，一开始将 start 末位设置成 1，一直等到 done 末位为 1 时结束退出即可

二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，逻辑仿真和 FPGA 调试过程中的难点等）

没有遇到什么很值得讨论的问题。本实验应当是选做实验中最简单的实验。

三、 对讲义中思考题（如有）的理解和回答

无思考题。

四、在课后，你花费了大约___10___小时完成此次实验。

五、对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

本次实验的 PPT 完整度很高，但是 dma 实验缺少很多重要信息，如写自动机并没有给出，并且较难通过给出的读自动机类推，同时 PPT 上还存在一些错误，极具误导性。之所以在 dnn 实验报告中提出，是因为我正是因为这些重要信息的缺失而未能完成 dma 实验。

另外，希望助教验收时多进行本实验相关的提问，如卷积实现思路、padding 计算方法等，这些才是实验核心，更能体现思考。