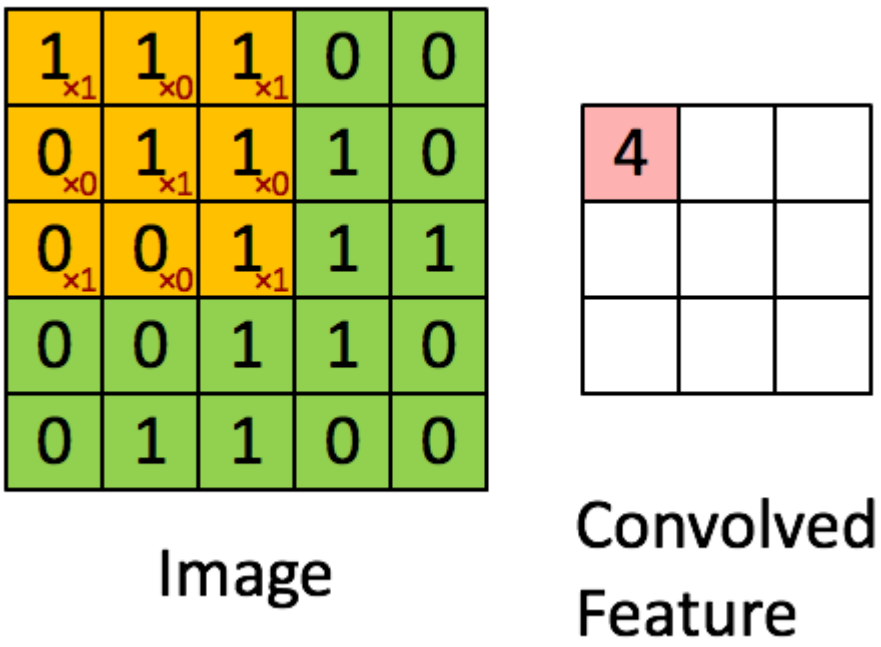
caffe中的im2col函数分析

首先要说一点，caffe中的im2col函数是进行卷积计算的核心函数之一，另外一个核心函数就是矩阵乘法函数。那为什么im2col函数是如此的重要呢，本节就简单的分析一下吧。

使用im2col的初衷



我们可以从上面的动图可以知道卷积的操作就是**卷积核**和卷积核所选取的区域进行对应元素进行一一相乘并且求和的动作，等等这个操作和我们学习的向量内积的操作是一样的，所以我们能不能将image转换成某种matrix，然后改matrix和kernel组成的matrix进行矩阵乘法从而得到我们的feature map，这就是引出im2col函数的初衷。

但是说实话通过im2col转换然后进行矩阵乘法的方法并不是最快的实现卷积的方法，还有一种方法可以更加快速的实现卷积过程——Winograd方法，在这里暂时不做分析，有兴趣的可以自行了解。

im2col函数的理解

关于im2col函数的解析在网络可以找到非常多好的资源，这里我就将im2col函数的理解分为**直观理解**和**结合代码理解**（很多地方也是整合网络上的一些优秀博客），如果仅仅想了解im2col函数的作用就看**直观理解**的部分就够了，但是想在进一步了解im2col函数的实现细节的话还看代码的部分。

直观理解

本段参考博客https://blog.csdn.net/Mrhiuser/article/details/52672824

单通道数据转换过程：

设定image和kernel的基本参数：

channel = 1;

height = 4;

width = 4;

kernel_h = 3;

kernel_w = 3;

stride_h = stride_w = 1;

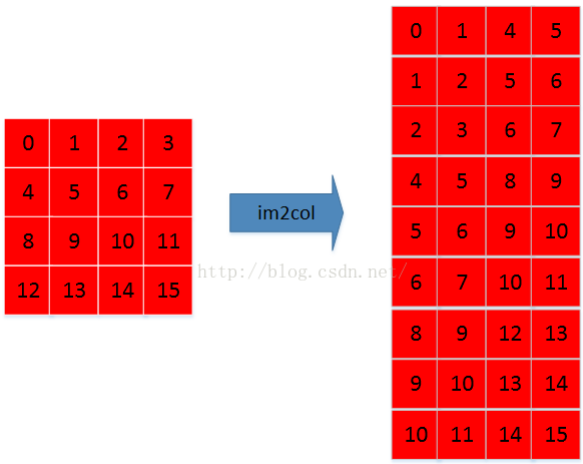
pad_h = pad_w = 0;

dilation_h = dilation_w = 1; （空洞卷积会在后面代码中分析，搞清楚简单的卷积之后理解空洞卷积也是比较容易的）

经过卷积后其输出结果为2×2的feature map。

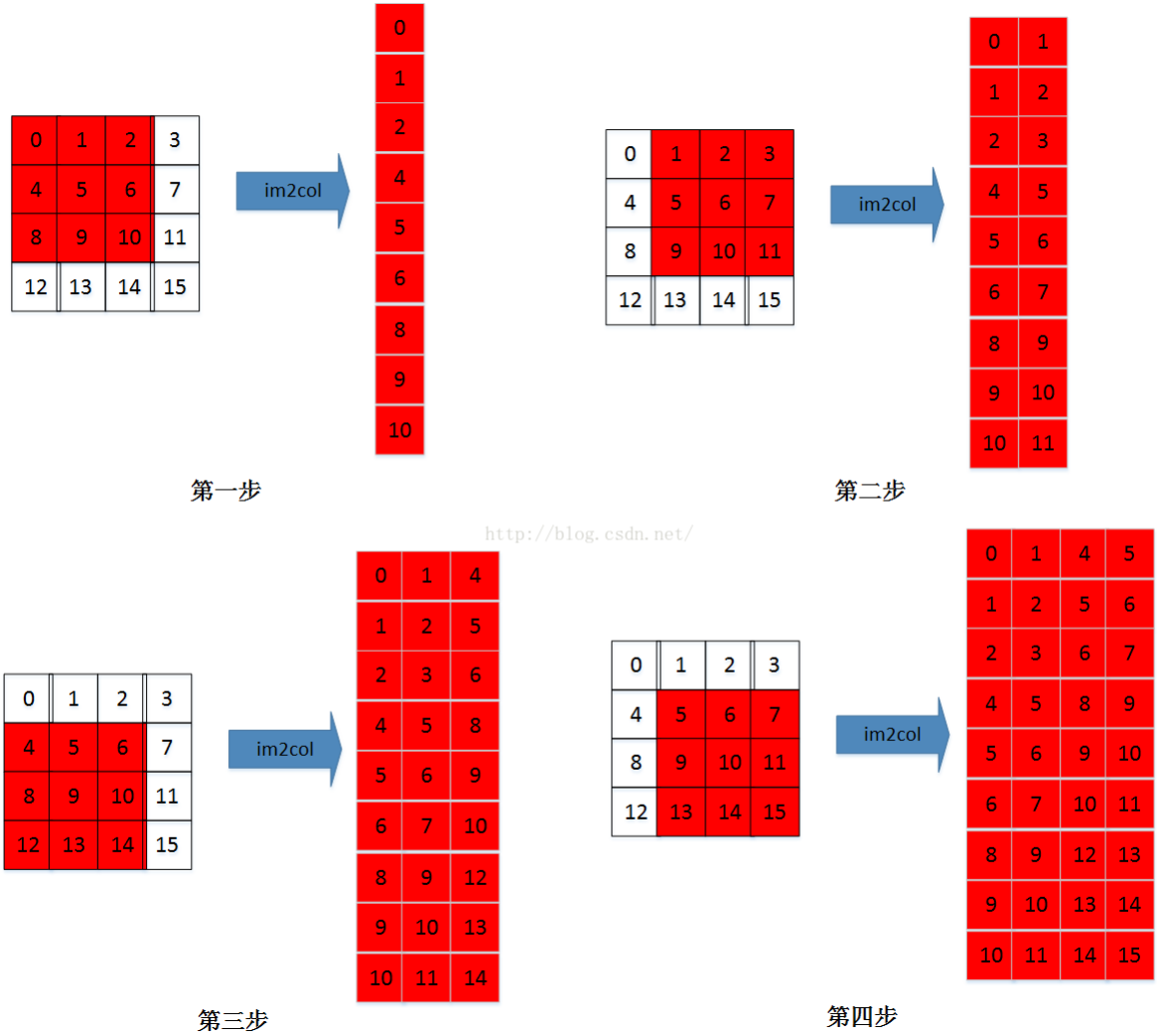
```
output_h = (height + 2 * pad_h - (dilation_h * (kernel_h - 1) + 1)) / stride_h + 1 = 2
output_w = (width + 2 * pad_w -(dilation_w * (kernel_w - 1) + 1)) / stride_w + 1 = 2
```

下面这张图就是im2col函数的作用，将图片转换为一个matrix。



(a) (b)

那将上述的动作拆分成四步来看一下具体是如何将一个图片拉成一列一列的。



第一步　　第二步

第三步　　第四步

上述过程就是一维图片的im2col过程，但是在这里需要注意一个问题，就是caffe中的数据存储都是row优先存储的，并且caffe是使用一维数组来存储相应的图片以及im2col之后的数据，所以我们去看数据的排列的时候image的数据是这样的：

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

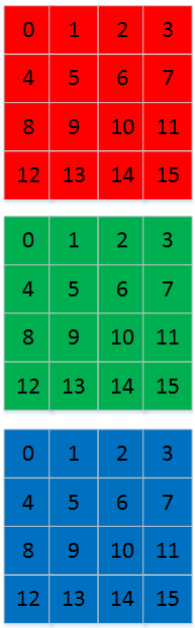
而经过im2col之后的数据是这样的：

```
[0, 1, 4, 5, 1, 2, 5, 6, 2, 3, 6, 7, 4, 5, 8, 9, 5, 6, 9, 10, 6, 7, 10, 11, 8, 9, 12, 13, 9, 10,
13, 14, 10, 11, 14, 15]
```

清楚caffe中的数据排列方式对后续的代码阅读很有帮助。

多通道的im2col

假设三个通道(R, G, B)的图像通道，图像在内存中存储的顺序是首先连续存储第一通道的数据，然后再存储第二通道的数据，最后存储第三通道的数据，注意每个通道的数据存储都是row优先的。



三通道图像

多通道的im2col过程，首先im2col第一通道，然后再im2col第二通道，最后是im2col第三通道，各通道的im2col的数据再内存中也是连续存储的，如下图：

0	1	4	5
1	2	5	6
2	3	6	7
4	5	8	9
5	6	9	10
6	7	10	11
8	9	12	13
9	10	13	14
10	11	14	15

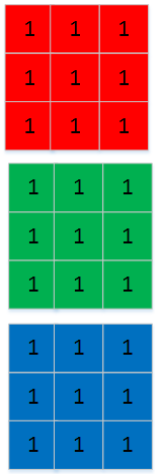
0	1	4	5
1	2	5	6
2	3	6	7
4	5	8	9
5	6	9	10
6	7	10	11
8	9	12	13
9	10	13	14
10	11	14	15

0	1	4	5
1	2	5	6
2	3	6	7
4	5	8	9
5	6	9	10
6	7	10	11
8	9	12	13
9	10	13	14
10	11	14	15

三通道图像im2col

卷积核kernel

图像的每个通道对应一个kernel通道，如下图：



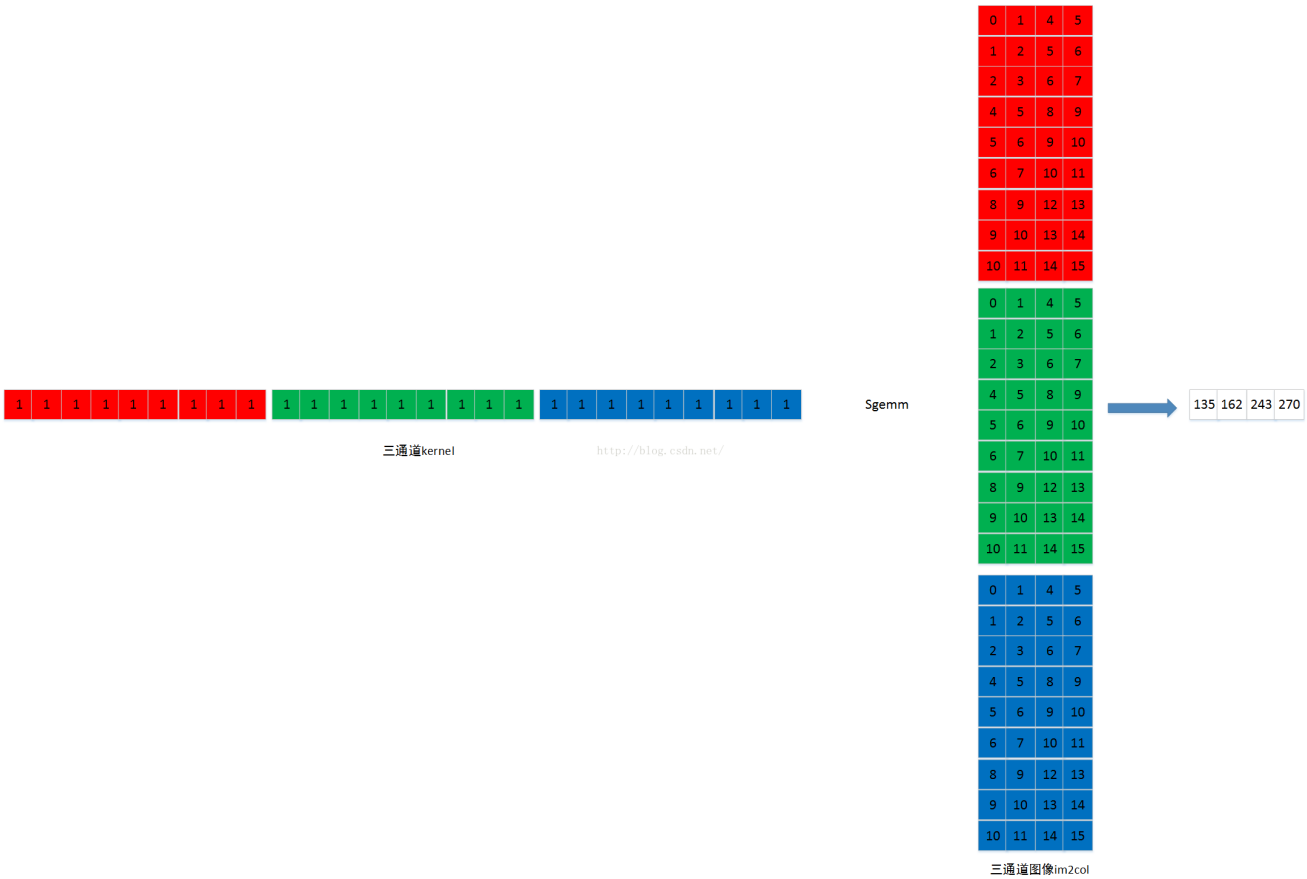
三通道kernel

kernel在内存中的数据是连续存储的，所以它本身就是一种行向量的形式，所以不需要任何其他的操作，那他在内存中的形式如下：

三通道kernel

矩阵gemm

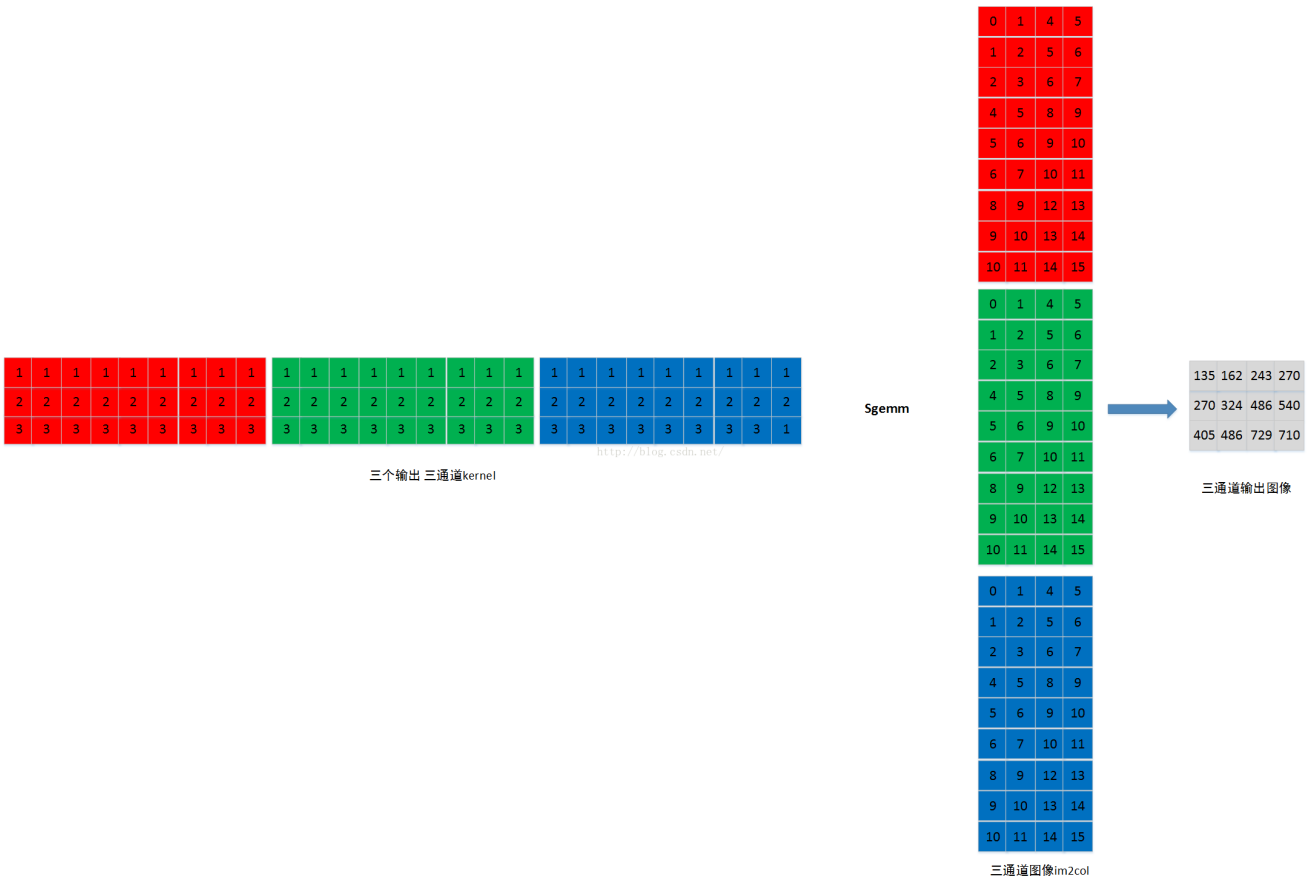
我们将kernel的数据和图像的im2col数据进行矩阵相乘，则得到如下结果：



那我们从上述获得的结果在内存中也是连续存储的，我们将其拆成2×2的形式，这就和我们大脑中理解的卷积是一样的了，其结果为：

多kernel矩阵相乘

在深度学习中我们遇到的kernel和image做卷积的时候都是好多个kernel，每个kernel生成一张feature map，然后多个kernel生成最终版的feature map，这个kernel的个数就是我们输出的feature map的channel数，其过程可见下图：

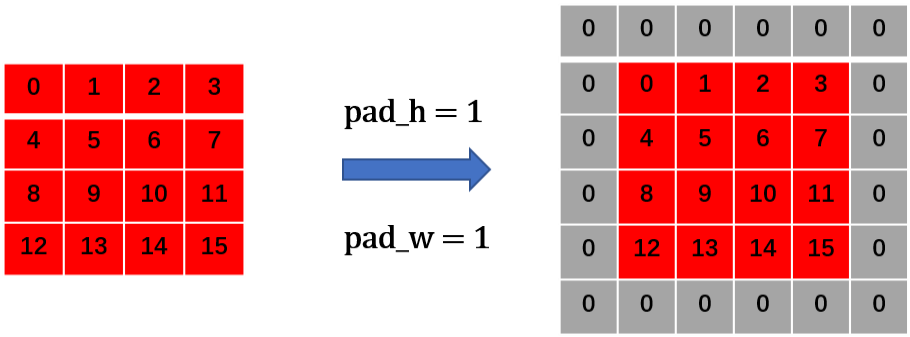


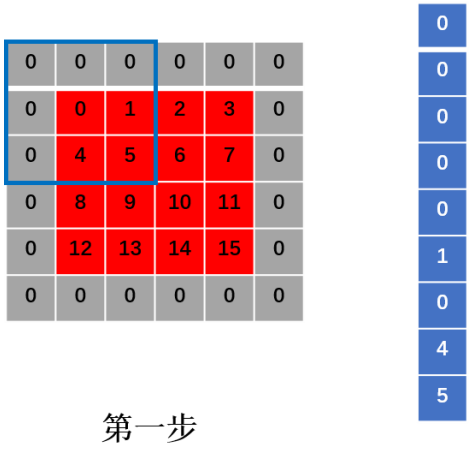
输出的feature map结果在内存中也是连续存储的，我们将其变换一下形式：

通过上述的过程就可以比较清楚的理解im2col函数的意图了。

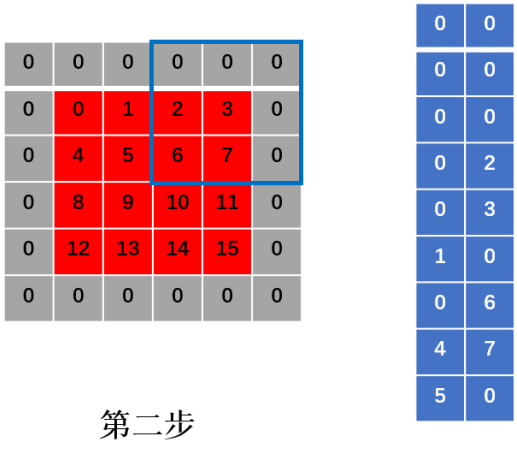
带有pad的示意图

下面给出一个一维图片带有pad的im2col过程（因为三维图片以及矩阵乘法和上述的差不多，所以就不再给出演示图）。

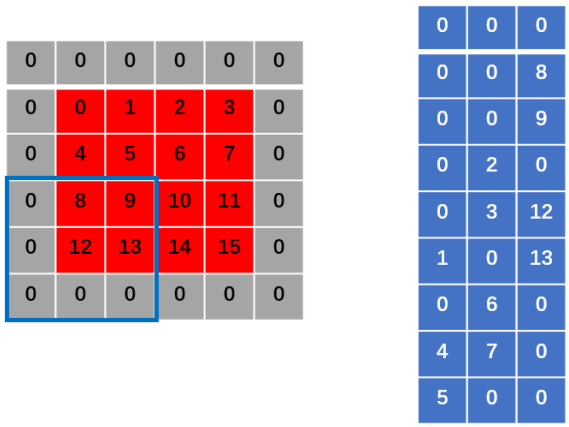




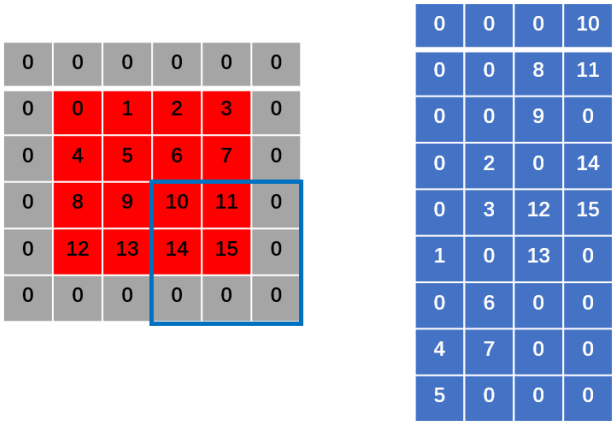
第一步



第二步



第三步



第四步

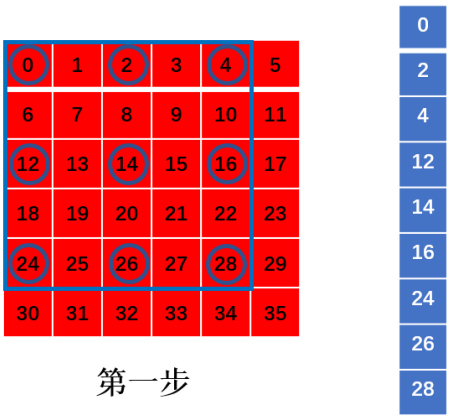
空洞卷积

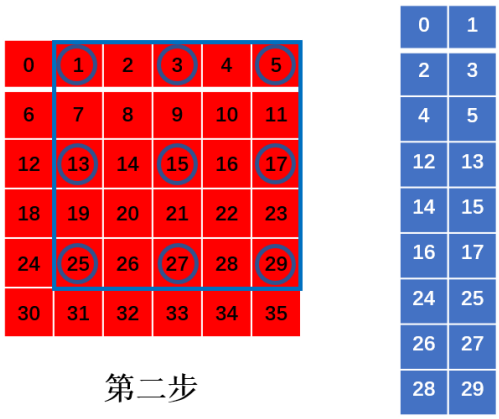
关于空洞卷积，其主要有两个作用，一个是可以扩大感受野，另外是可以捕捉上下文信息。这里对于空洞卷积的具体作用不分析，有兴趣的同学可以查阅空洞卷积的论文或者看这篇博客的总结，这里主要是看空洞卷积是具体如何计算的。

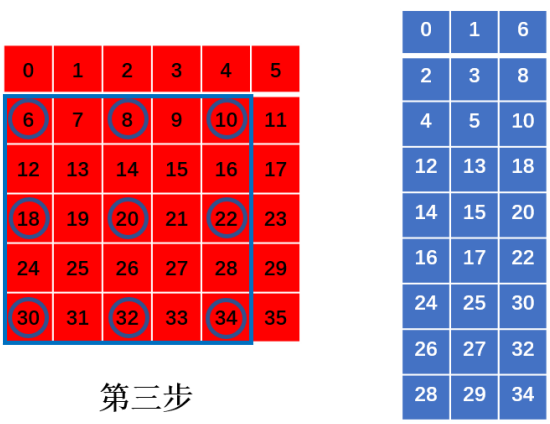
下面是一个空洞卷积im2col过程，原来的kernel的大小为3×3，dilation_h和dilation_w的值都为2。那经过空洞化的卷积核的大小为：

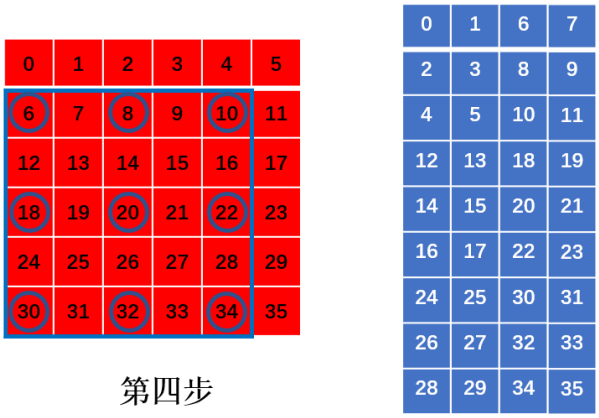
$$
\begin{aligned}
kernel\_H_{dilation} &= dilation\_h * (kernel\_H - 1) + 1 = 5 \\
kernel\_W_{dilation} &= dilation\_w * (kernel\_W - 1) + 1 = 5
\end{aligned} \tag{1}
$$

我们可以看到尽管卷积核的大小变大了，但是它实际的计算量是并没有变化的，处于空洞部分的值是不参与计算的。









结合代码理解

首先放代码，想象大家看到如此多的for循环绝对非常绝望，没办法即使绝望还是得硬着头皮看。当然我还是尽量以图片的形式给大家展现作者写im2col函数的思路，理解思路了，5层for循环又算什么呢?

```
template <typename Dtype>
void im2col_cpu(const Dtype* data_im, const int channels,
    const int height, const int width, const int kernel_h, const int kernel_w,
    const int pad_h, const int pad_w,
    const int stride_h, const int stride_w,
    const int dilation_h, const int dilation_w,
    Dtype* data_col) {
    const int output_h = (height + 2 * pad_h -
        (dilation_h * (kernel_h - 1) + 1)) / stride_h + 1;
    const int output_w = (width + 2 * pad_w -
        (dilation_w * (kernel_w - 1) + 1)) / stride_w + 1;
    const int channel_size = height * width;
    for (int channel = channels; channel--; data_im += channel_size) {
    for (int kernel_row = 0; kernel_row < kernel_h; kernel_row++) {
        for (int kernel_col = 0; kernel_col < kernel_w; kernel_col++) {
        int input_row = -pad_h + kernel_row * dilation_h;
        for (int output_rows = output_h; output_rows; output_rows--) {
            if (!is_a_ge_zero_and_a_lt_b(input_row, height)) {
            for (int output_cols = output_w; output_cols; output_cols--) {
                *(data_col++) = 0;      //1
```

```
            }
            } else {
            int input_col = -pad_w + kernel_col * dilation_w;
            for (int output_col = output_w; output_col; output_col--) {
                if (is_a_ge_zero_and_a_lt_b(input_col, width)) {
                *(data_col++) = data_im[input_row * width + input_col]; //1
                } else {
                *(data_col++) = 0;      //1
                }
                input_col += stride_w;
            }
            }
            input_row += stride_h;
        }
        }
    }
    }
}
```

首先我们看上述20，26和28行代码，这里的data_col这个数组是经过im2col的输出数组，作者应该是以data_col的依次遍历填入为主要思路来构建该for循环的。我们来看下面这张图，这边我是举了一个一般化的例子，由于要填充的内容很多，这边主要是画了前8步的过程。那我们通过下图就可以比较清楚的理解上述程序的for循环在做什么。

第一层for循环：该for循环是不同channel上面的im2col转换。

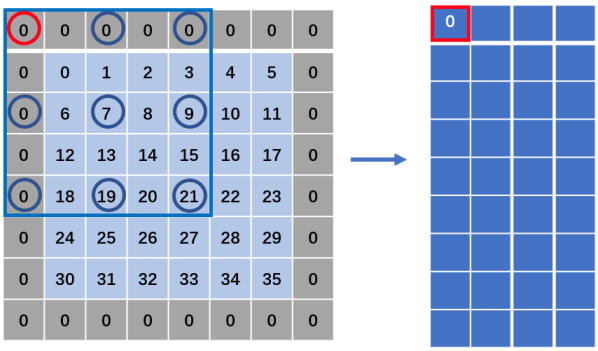
第二层for循环：我们的kernel的行遍历（注意这个是原生kernel的行遍历，而不是dilation之后的放大kernel的行遍历）。

第三层for循环：kernel的列遍历。

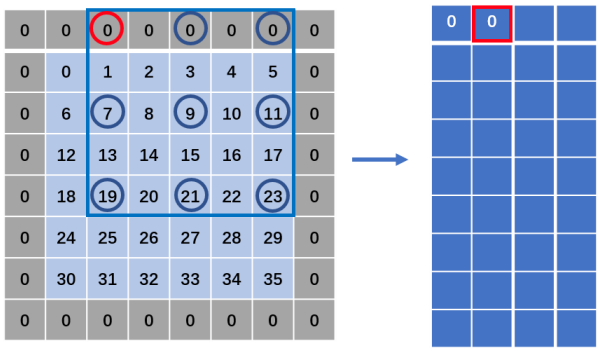
第四层for循环：整个kernel在图片上列方向跳跃遍历（跳跃的距离为stride_H）。

第五层for循环：整个kernel在图片上行方向跳跃遍历（跳跃的距离为stride_W）。

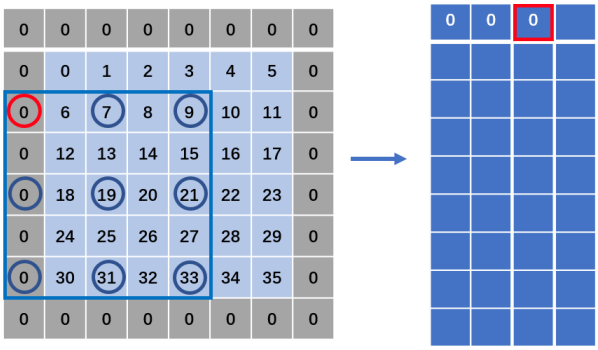
那我们下面的过程就是for循环从最里面往外面一层一层的剥出来。其实可以每个for都和图片对应起来，但是图片好多，偷个懒就不画了，大家脑补一下应该还是比较好理解的。



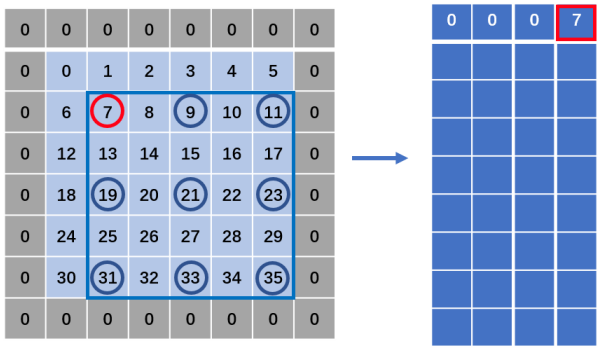
第1步



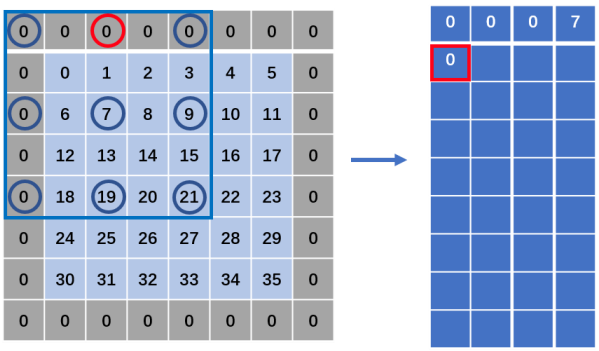
第2步



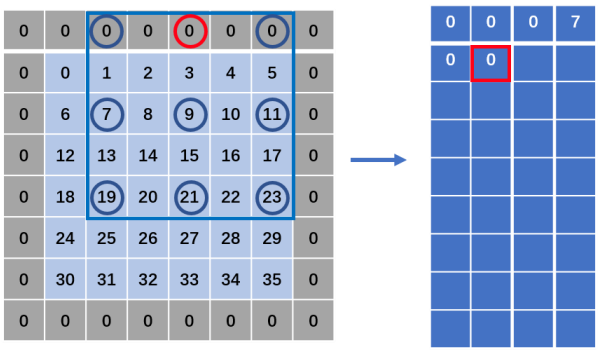
第3步



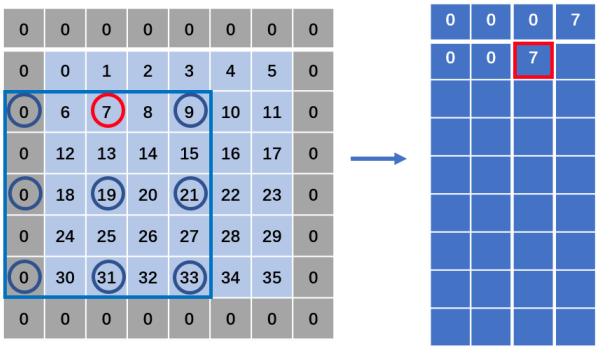
第4步



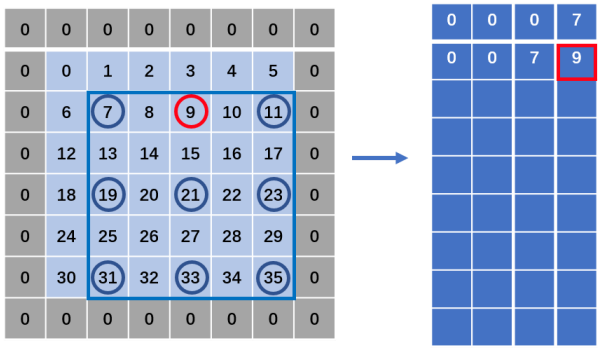
第5步



第6步

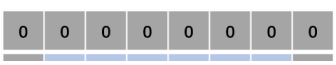


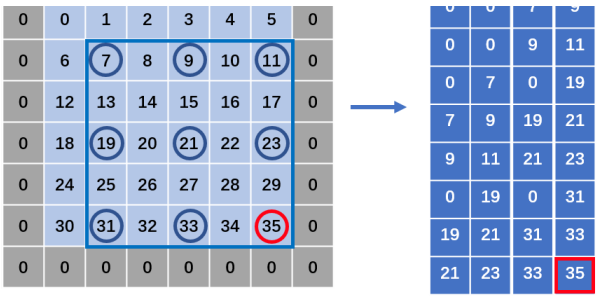
第7步



第8步

经过一系列的操作





最终

关于上述卷积过程的参数如下：

channel = 1;

height = 6;

width = 6;

kernel_h = 3;

kernel_w = 3;

stride_h = stride_w = 2;

pad_h = pad_w = 1;

dilation_h = dilation_w = 2;

如果不用im2col该怎么办

这个链接是关于卷积的性能比较，说实话原生caffe在卷积的实现上无论在速度还是内存效率上都是比较糟糕的。幸好caffe框架还使用了英伟达的cudnn卷积加速。当然卷积还有一种比较快速的方法Winograd，后面有时间可以具体了解一下它是如何实现的。