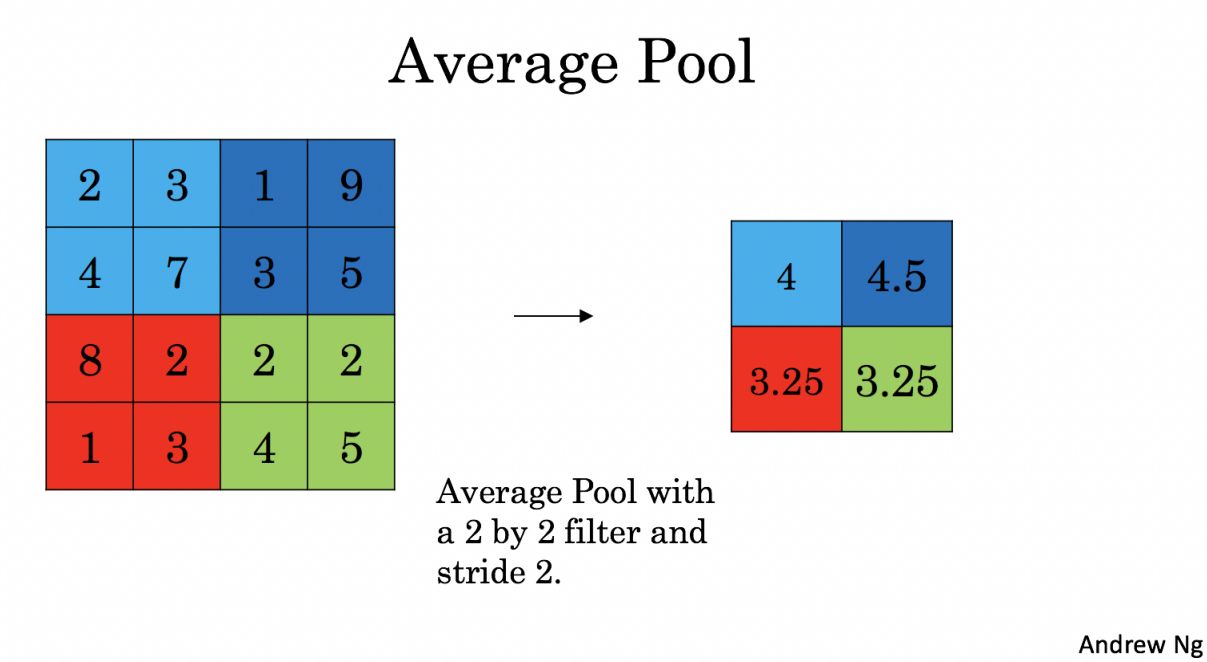
caffe中的池化实现

池化层在卷积神经网络中是经常用到的一个模块，那一般我们使用的池化是**最大池化**和**平均池化**这两种，这一节我们简单看一下**最大池化**和**平均池化**的前向传播和反向传播的直观理解和代码理解。

直观理解

平均池化的前向传播过程：

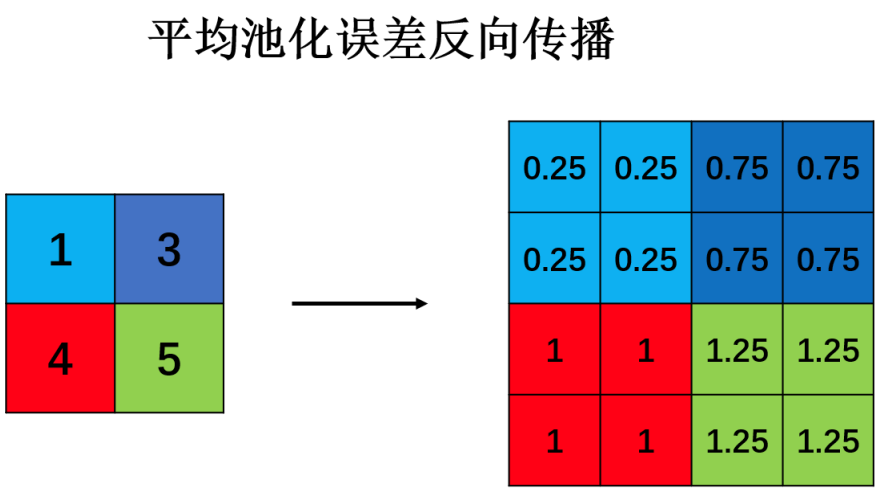
平均池化的前向传播过程非常简单，就是对所选区域进行求算术平均的操作，如下图。



Andrew Ng

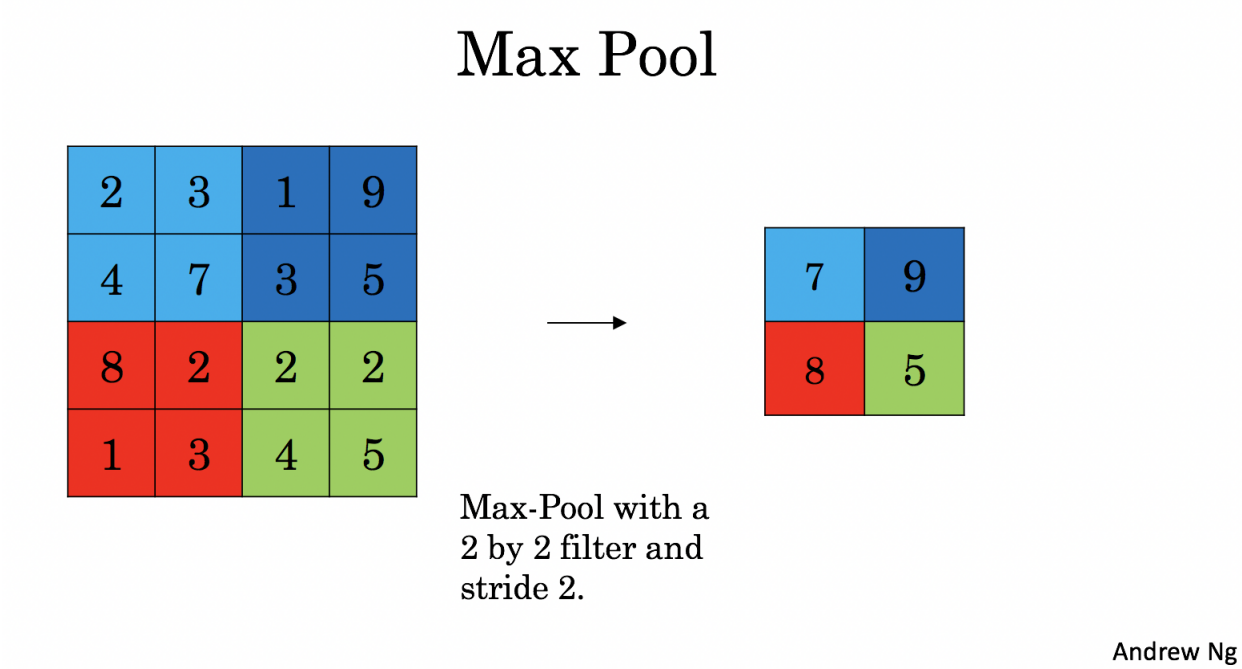
平均池化的反向传播过程：

平均池化这么一个动作我们该如何去定义其导数呢？很难，那我们就想一个比较合理的方法，就是将top层的梯度给上一层区域进行平分，因为前一层的每个值都是对后面的值都是有影响的，具体看下图：



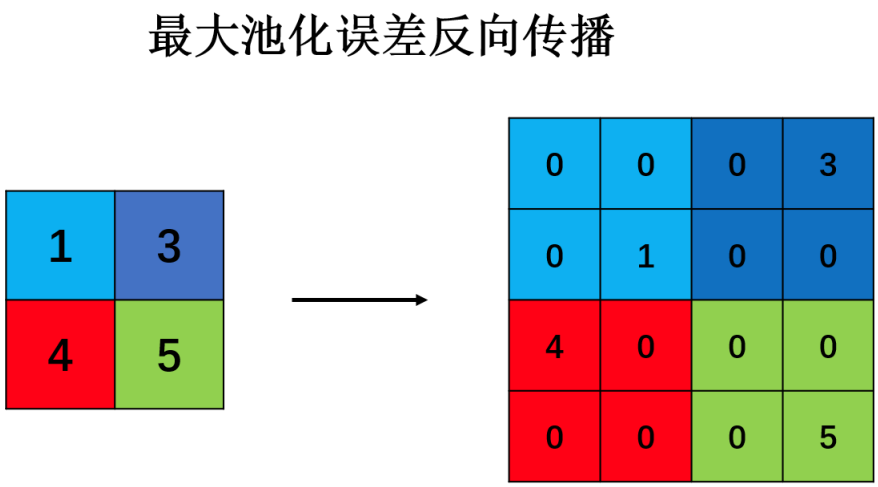
最大池化的前向传播过程：

最大池化的前向传播也是比较简单，就是将最大值获取出来，但是我们需要记住最大值在原来的feature map的具体位置，因为这个在后面反向传播的时候需要用到，前向传播的示意图如下：



最大池化的反向传播过程：

由于最大池化只有最大值对后面的误差会产生影响，所以反向传播的时候只传播最大值的反向传播，其他的位置的梯度都是0。具体见下图：



代码解析

头文件分析

首先我们看一下PoolingLayer类的变量，具体注释都已经写在代码中，这些成员变量都非常好理解，所以可以直接看代码就可以。

(include/caffe/layers/pooling_layer.hpp)

```
47   int kernel_h_, kernel_w_;             //kernel的大小
48   int stride_h_, stride_w_;             //stride的步伐
49   int pad_h_, pad_w_;                   //pad尺寸
50   int channels_;                        //输入feature map的通道数
51   int height_, width_;                  //输入feature map的宽和高
52   int pooled_height_, pooled_width_;        //池化之后的高和宽
53   bool global_pooling_;                 //是否采用全局池化
54   PoolingParameter_RoundMode round_mode_;   //除法模式（地板除法还是天花板除法）
55   Blob<Dtype> rand_idx_;                //记录一个kernel区域中选中随机数的位置
56   Blob<int> max_idx_;                   //记录一个kernel区域中最大元素的位置
```

关于PoolingParameter_RoundMode我们可以在proto文件中进行查看，其代码如下。

proto文件分析

(src/caffe/proto/caffe.proto)

```
919 message PoolingParameter {
920   enum PoolMethod {                 //选择池化的方法，有最大池化、平均池化和随机池化
921     MAX = 0;                        //三种，但是我们一般使用的都是平均池化和最大池化
922     AVE = 1;
923     STOCHASTIC = 2;
924   }
925   optional PoolMethod pool = 1 [default = MAX]; // 默认是最大池化
926   // Pad, kernel size, and stride are all given as a single value for equal
927   // dimensions in height and width or as Y, X pairs.
928   optional uint32 pad = 4 [default = 0]; // The padding size (equal in Y, X)
929   optional uint32 pad_h = 9 [default = 0]; // The padding height
930   optional uint32 pad_w = 10 [default = 0]; // The padding width
931   optional uint32 kernel_size = 2; // The kernel size (square)
932   optional uint32 kernel_h = 5; // The kernel height
933   optional uint32 kernel_w = 6; // The kernel width
934   optional uint32 stride = 3 [default = 1]; // The stride (equal in Y, X)
935   optional uint32 stride_h = 7; // The stride height
936   optional uint32 stride_w = 8; // The stride width
937   enum Engine {
938     DEFAULT = 0;
939     CAFFE = 1;
940     CUDNN = 2;
941   }
942   optional Engine engine = 11 [default = DEFAULT];
943   // If global_pooling then it will pool over the size of the bottom by doing
944   // kernel_h = bottom->height and kernel_w = bottom->width
945   optional bool global_pooling = 12 [default = false];  //相当于变成全连接层的感觉
946   // How to calculate the output size - using ceil (default) or floor rounding.
947   enum RoundMode {      //除法的方式，有天花板除法和地板除法
948     CEIL = 0;           //在代码中我们将会看到不同除法方式对程序所产生的影响
949     FLOOR = 1;
950   }
951   optional RoundMode round_mode = 13 [default = CEIL];  //默认的除法方式是天花板除法
952 }
```

```

```

cpp文件分析

LayerSetUp函数

LayerSetUp函数主要是对成员变量进行初始化的操作，这里的没啥需要特别注意的地方，所以可以直接看源代码。

Reshape函数

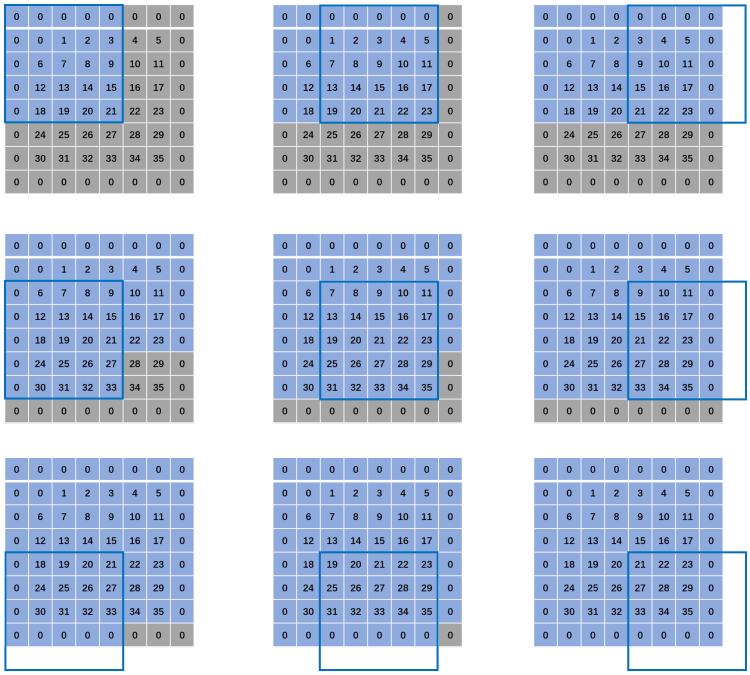
首先我们来看一下天花板除法和地板除法的区别。

(src/caffe/layers/pooling_layer.cpp)

```
switch (round_mode_) {
case PoolingParameter_RoundMode_CEIL:
  pooled_height_ = static_cast<int>(ceil(static_cast<float>(      //调用ceil
      height_ + 2 * pad_h_ - kernel_h_) / stride_h_)) + 1;
  pooled_width_ = static_cast<int>(ceil(static_cast<float>(
      width_ + 2 * pad_w_ - kernel_w_) / stride_w_)) + 1;
  break;
case PoolingParameter_RoundMode_FLOOR:
  pooled_height_ = static_cast<int>(floor(static_cast<float>(     //调用floor
      height_ + 2 * pad_h_ - kernel_h_) / stride_h_)) + 1;
  pooled_width_ = static_cast<int>(floor(static_cast<float>(
      width_ + 2 * pad_w_ - kernel_w_) / stride_w_)) + 1;
  break;
default:
  LOG(FATAL) << "Unknown rounding mode.";
}
```

如下图的地板除法可能会让边边角角的一些像素不能被利用起来，但是可以减少池化的步骤。

基于天花板除法的池化方式可以将每个像素都给利用起来，但是会增加一定的池化次数。



其实一般我们在设计网络的时候卷积或者池化计算的时候都是可以整除的，遇到天花板除法和地板除法的概率其实一般比较小，但上述两种情况是一般化的情况，需要掌握一下的。

但是也有特殊的情况，如果基于天花板除法的池化遇到下面这种情况的化，那在该方向的池化会自动变成地板除池化，就是如果某个方向上最后一次池化时kernel圈中的位置都是pad的像素，那该步池化就会被舍弃。

(src/caffe/layers/pooling_layer.cpp)

```
107   if (pad_h_ || pad_w_) {
108     // If we have padding, ensure that the last pooling starts strictly
109     // inside the image (instead of at the padding); otherwise clip the last.
110     if ((pooled_height_ - 1) * stride_h_ >= height_ + pad_h_) {
111       --pooled_height_;
112     }
113     if ((pooled_width_ - 1) * stride_w_ >= width_ + pad_w_) {
114       --pooled_width_;
115     }
116     CHECK_LT((pooled_height_ - 1) * stride_h_, height_ + pad_h_);
117     CHECK_LT((pooled_width_ - 1) * stride_w_, width_ + pad_w_);
118   }
```

最后我们需要对max_idx_和rand_idx_这两个blob的大小进行Reshape过，保证其大小和bottom的blob大小一样。

(src/caffe/layers/pooling_layer.cpp)

```
124   // If max pooling, we will initialize the vector index part.
125   if (this->layer_param_.pooling_param().pool() ==
126       PoolingParameter_PoolMethod_MAX && top.size() == 1) {
127     max_idx_.Reshape(bottom[0]->num(), channels_, pooled_height_,
128         pooled_width_);
129   }
130   // If stochastic pooling, we will initialize the random index part.
131   if (this->layer_param_.pooling_param().pool() ==
132       PoolingParameter_PoolMethod_STOCHASTIC) {
133     rand_idx_.Reshape(bottom[0]->num(), channels_, pooled_height_,
134       pooled_width_);
135   }
```

Forward_cpu函数

涉及到卷积核池化就会有一大堆的for循环代码，但是幸运的是池化的for循环代码非常容易理解。我们先来看一下最大池化的的for循环代码。

```
164     for (int n = 0; n < bottom[0]->num(); ++n) {
165       for (int c = 0; c < channels_; ++c) {
166         for (int ph = 0; ph < pooled_height_; ++ph) {
167           for (int pw = 0; pw < pooled_width_; ++pw) {
168             int hstart = ph * stride_h_ - pad_h_;   //计算在原feature map上行开始绝对位置
169             int wstart = pw * stride_w_ - pad_w_;   //计算在元feature map上列开始绝对位置
170             int hend = min(hstart + kernel_h_, height_);  //计算行结束位置
171             int wend = min(wstart + kernel_w_, width_);   //计算列结束位置
172             hstart = max(hstart, 0);    //因为pad对max pool没啥用，所以一般不考虑pad的区域
173             wstart = max(wstart, 0);    //前面求hend和wend也是有这方面的考虑了。
174             const int pool_index = ph * pooled_width_ + pw;  //这个是pool之后feature map
175             for (int h = hstart; h < hend; ++h) {           //像素点在数组上的位置
176               for (int w = wstart; w < wend; ++w) {
177                 const int index = h * width_ + w;
178                 if (bottom_data[index] > top_data[pool_index]) {
179                   top_data[pool_index] = bottom_data[index];
180                   if (use_top_mask) {    //如果我们使用top_mask的话用top mask记录最大值位置
181                     top_mask[pool_index] = static_cast<Dtype>(index);
182                   } else {
183                     mask[pool_index] = index;
184                   }
185                 }
186               }
187             }
188           }
189         }
190         // compute offset
191         bottom_data += bottom[0]->offset(0, 1); //bottom data往后移动一个通道的数据
192         top_data += top[0]->offset(0, 1);   //后面top data mask data都往后移动一个通道
193         if (use_top_mask) {
194           top_mask += top[0]->offset(0, 1);
195         } else {
196           mask += top[0]->offset(0, 1);
197         }
```

```
198         }
199       }
200       break;
```

我们先来看一下前4个for循环，它主要作用是通过池化之后feature map上像素点的位置来反推出kernel在输入feature map所圈住的地方，所以这段代码相当简单。

同理，我们在看平均池化的时候，这个代码就非常简单了，因为它不需要记录额外的信息，所以并没有相应的mask数组，其基本和最大池化差不多，所以就列出平均池化的代码了。

Backward_cpu函数

反向传播代码也非常简单，它就是利用前面正向传播的代码，然后反过来给原feature map的diff数据进行赋值。具体的代码结构形式和前向传播相差不大，所以也没有啥罗列的必要。