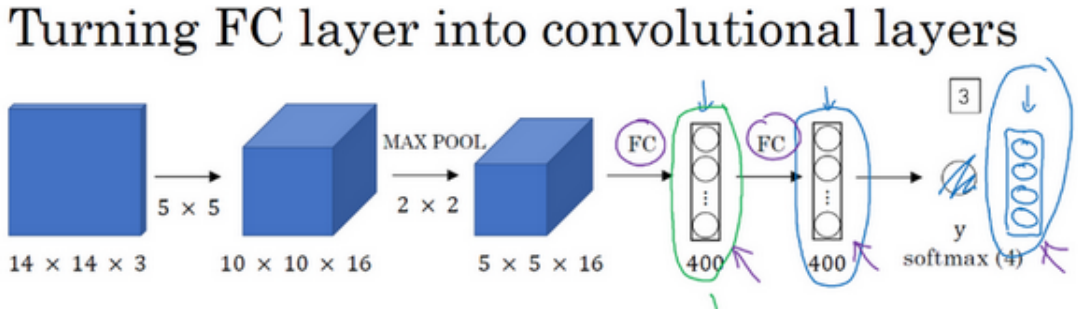
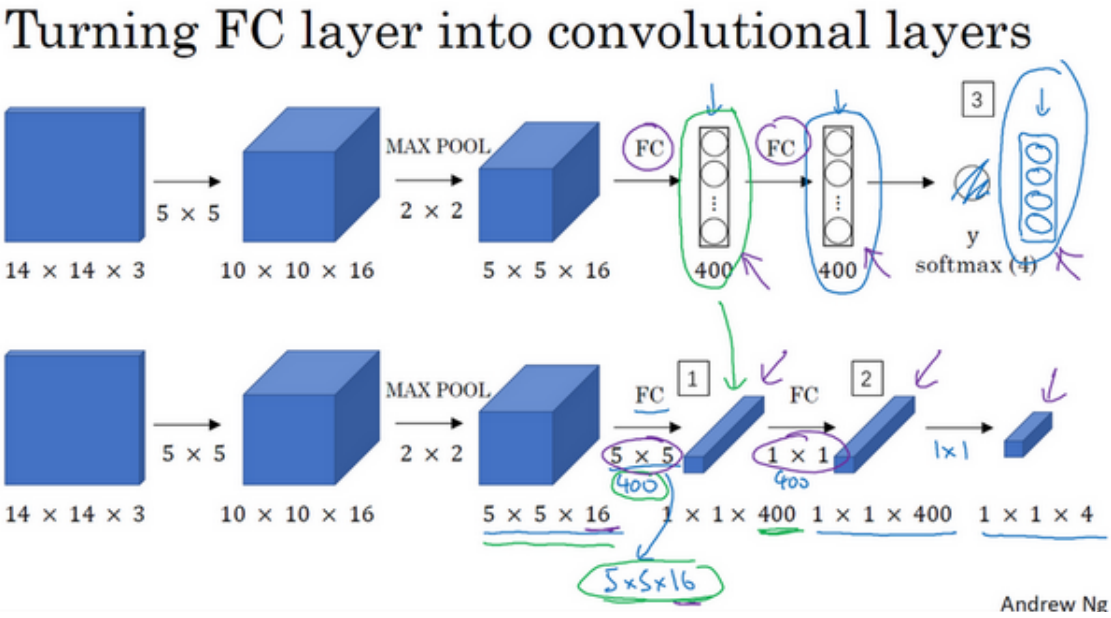
3.4 滑动窗口的卷积实现（Convolutional implementation of sliding windows）

上节课，我们学习了如何通过卷积网络实现滑动窗口对象检测算法，但效率很低。这节课我们讲讲如何在卷积层上应用这个算法。

为了构建滑动窗口的卷积应用，首先要知道如何把神经网络的全连接层转化成卷积层。我们先讲解这部分内容，下一张幻灯片，我们将按照这个思路来演示卷积的应用过程。



假设对象检测算法输入一个 14×14×3 的图像，图像很小，不过演示起来方便。在这里过滤器大小为 5×5，数量是 16，14×14×3 的图像在过滤器处理之后映射为 10×10×16。然后通过参数为 2×2 的最大池化操作，图像减小到 5×5×16。然后添加一个连接 400 个单元的全连接层，接着再添加一个全连接层，最后通过 **softmax** 单元输出y。为了跟下图区分开，我先做一点改动，用 4 个数字来表示y，它们分别对应 **softmax** 单元所输出的 4 个分类出现的概率。这 4 个分类可以是行人、汽车、摩托车和背景或其它对象。



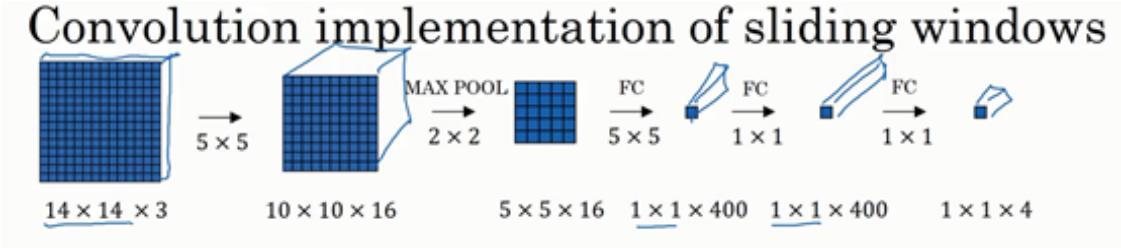
现在我要演示的就是如何把这些全连接层转化为卷积层，画一个这样的卷积网络，它的前几层和之前的一样，而对于下一层，也就是这个全连接层，我们可以用 5×5 的过滤器来实现，数量是 400 个（编号 1 所示），输入图像大小为 5×5×16，用 5×5 的过滤器对它进行卷积操作，过滤器实际上是 5×5×16，因为在卷积过程中，过滤器会遍历这 16 个通道，所以这两处的通道数量必须保持一致，输出结果为 1×1。假设应用 400 个这样的 5×5×16 过滤器，输出维度就是 1×1×400，我们不再把它看作一个含有 400 个节点的集合，而是一个 1×1×400 的输出层。从数学角度看，它和全连接层是一样的，因为这 400 个节点中每个节点都有一个 5×5×16 维度的过滤器，所以每个值都是上一层这些 5×5×16 激活值经过某个任意线性函数的输出结果。

我们再添加另外一个卷积层（编号 2 所示），这里用的是 1×1 卷积，假设有 400 个 1×1 的过滤器，在这 400 个过滤器的作用下，下一层的维度是 1×1×400，它其实就是上个网络中的这一全连接层。最后经由 1×1 过滤器的处理，得到一个 **softmax** 激活值，通过卷积网络，我们最终得到这个 1×1×4 的输出层，而不是这 4 个数字（编号 3 所示）。

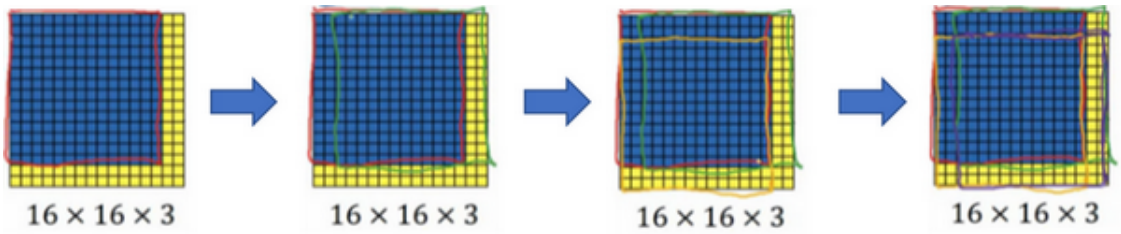
以上就是用卷积层代替全连接层的过程，结果这几个单元集变成了 1×1×400 和 1×1×4 的维度。

参考论文：**Sermanet, Pierre, et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks."** ***Eprint Arxiv*** **(2013).**

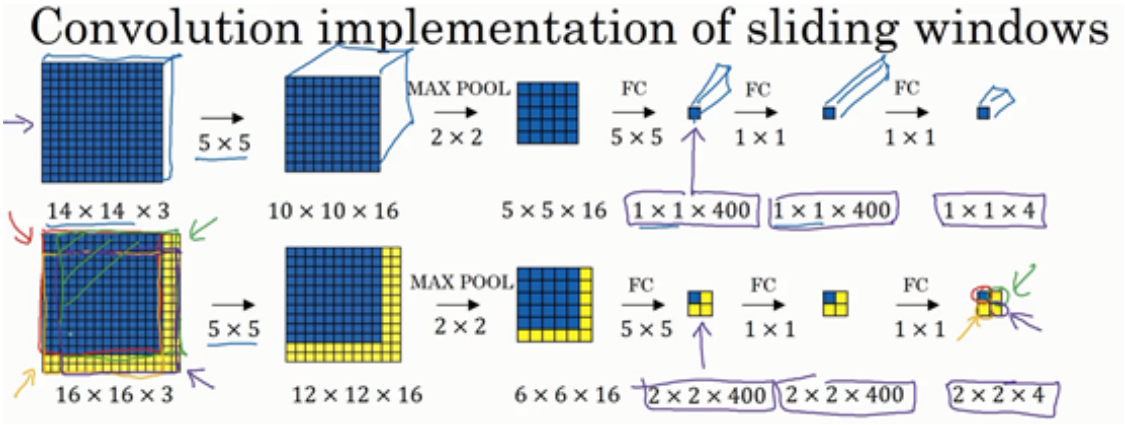
掌握了卷积知识，我们再看看如何通过卷积实现滑动窗口对象检测算法。讲义中的内容借鉴了屏幕下方这篇关于 **OverFeat** 的论文，它的作者包括 **Pierre Sermanet**，**David Eigen**，**张翔**，**Michael Mathieu**，**Rob Fergus**，**Yann LeCun**。



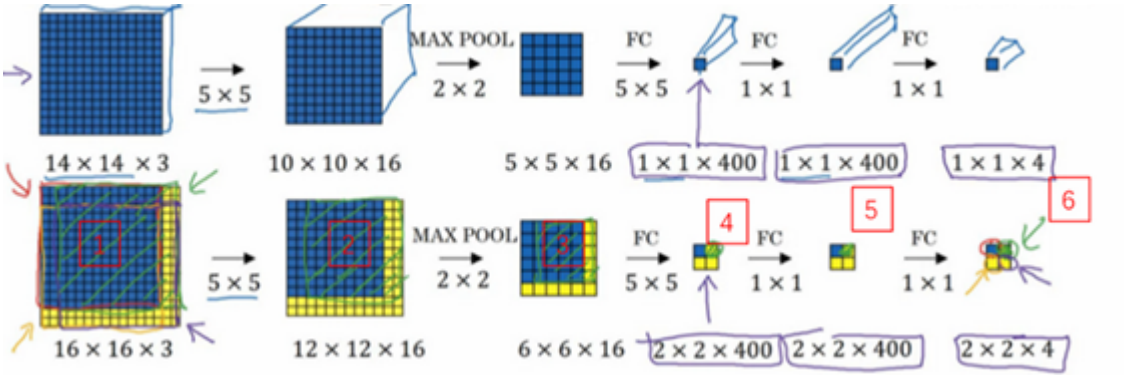
假设向滑动窗口卷积网络输入 14×14×3 的图片，为了简化演示和计算过程，这里我们依然用 14×14 的小图片。和前面一样，神经网络最后的输出层，即 **softmax** 单元的输出是 1×1×4，我画得比较简单，严格来说，14×14×3 应该是一个长方体，第二个 10×10×16 也是一个长方体，但为了方便，我只画了正面。所以，对于 1×1×400 的这个输出层，我也只画了它 1×1 的那一面，所以这里显示的都是平面图，而不是 3D 图像。



假设输入给卷积网络的图片大小是 14×14×3，测试集图片是 16×16×3，现在给这个输入图片加上黄色条块，在最初的滑动窗口算法中，你会把这片蓝色区域输入卷积网络（红色笔标记）生成 0 或 1 分类。接着滑动窗口，步幅为 2 个像素，向右滑动 2 个像素，将这个绿框区域输入给卷积网络，运行整个卷积网络，得到另外一个标签 0 或 1。继续将这个橘色区域输入给卷积网络，卷积后得到另一个标签，最后对右下方的紫色区域进行最后一次卷积操作。我们在这个 16×16×3 的小图像上滑动窗口，卷积网络运行了 4 次，于是输出了了 4 个标签。

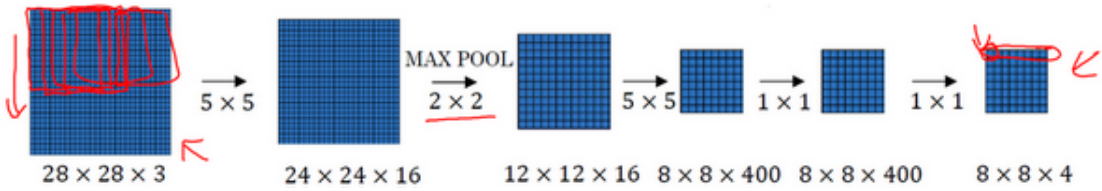


结果发现，这 4 次卷积操作中很多计算都是重复的。所以执行滑动窗口的卷积时使得卷积网络在这 4 次前向传播过程中共享很多计算，尤其是在这一步操作中（编号 1），卷积网络运行同样的参数，使得相同的 5×5×16 过滤器进行卷积操作，得到 12×12×16 的输出层。然后执行同样的最大池化（编号 2），输出结果 6×6×16。照旧应用 400 个 5×5 的过滤器（编号 3），得到一个 2×2×400 的输出层，现在输出层为 2×2×400，而不是 1×1×400。应用 1×1 过滤器（编号 4）得到另一个 2×2×400 的输出层。再做一次全连接的操作（编号 5），最终得到 2×2×4 的输出层，而不是 1×1×4。最终，在输出层这 4 个子方块中，蓝色的是图像左上部分 14×14 的输出（红色箭头标识），右上角方块是图像右上部分（绿色箭头标识）的对应输出，左下角方块是输入层左下角（橘色箭头标识），也就是这个 14×14 区域经过卷积网络处理后的结果，同样，右下角这个方块是卷积网络处理输入层右下角 14×14 区域(紫色箭头标识)的结果。

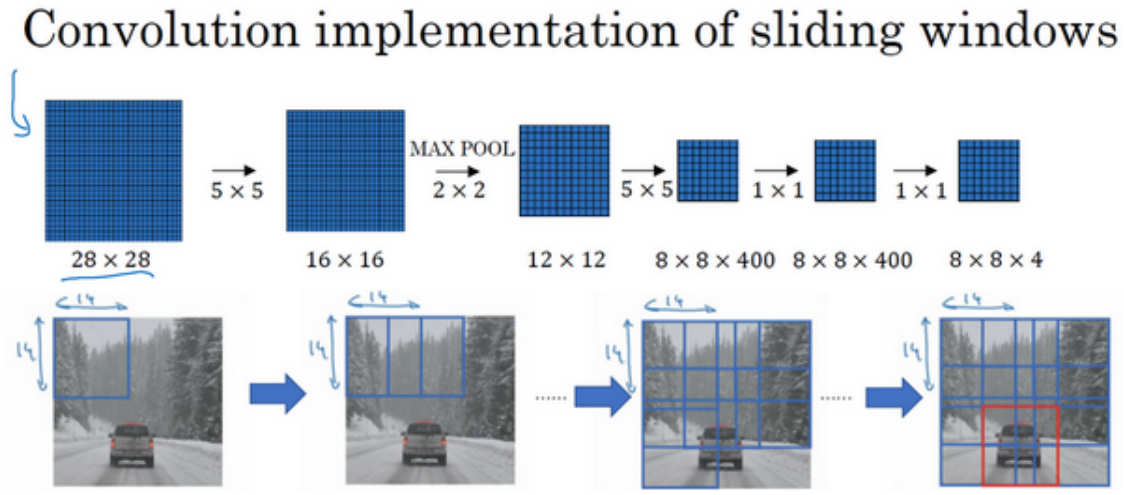


如果你想了解具体的计算步骤，以绿色方块为例，假设你剪切出这块区域（编号 1），传递给卷积网络，第一层的激活值就是这块区域（编号 2），最大池化后的下一层的激活值是这块区域（编号 3），这块区域对应着后面几层输出的右上角方块（编号 4，5，6）。

所以该卷积操作的原理是我们不需要把输入图像分割成四个子集，分别执行前向传播，而是把它们作为一张图片输入给卷积网络进行计算，其中的公共区域可以共享很多计算，就像这里我们看到的这个 4 个 14×14 的方块一样。

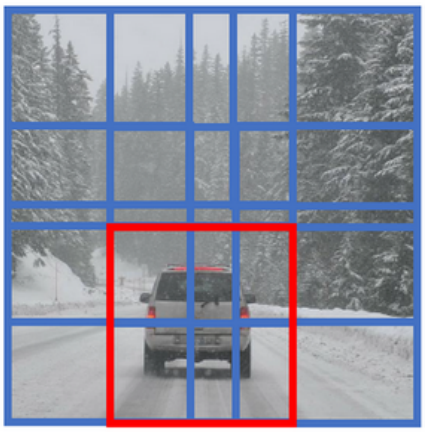


下面我们再看一个更大的图片样本，假如对一个 28×28×3 的图片应用滑动窗口操作，如果以同样的方式运行前向传播，最后得到 8×8×4 的结果。跟上一个范例一样，以 14×14 区域滑动窗口，首先在这个区域应用滑动窗口，其结果对应输出层的左上角部分。接着以大小为 2 的步幅不断地向右移动窗口，直到第 8 个单元格，得到输出层的第一行。然后向图片下方移动，最终输出这个 8×8×4 的结果。因为最大池化参数为 2，相当于以大小为 2 的步幅在原始图片上应用神经网络。



总结一下滑动窗口的实现过程，在图片上剪切出一块区域，假设它的大小是 14×14，把它输入到卷积网络。继续输入下一块区域，大小同样是 14×14，重复操作，直到某个区域识别到汽车。

但是正如在前一页所看到的，我们不能依靠连续的卷积操作来识别图片中的汽车，比如，我们可以对大小为 28×28 的整张图片进行卷积操作，一次得到所有预测值，如果足够幸运，神经网络便可以识别出汽车的位置。



以上就是在卷积层上应用滑动窗口算法的内容，它提高了整个算法的效率。不过这种算法仍然存在一个缺点，就是边界框的位置可能不够准确。下节课，我们将学习如何解决这个问题。