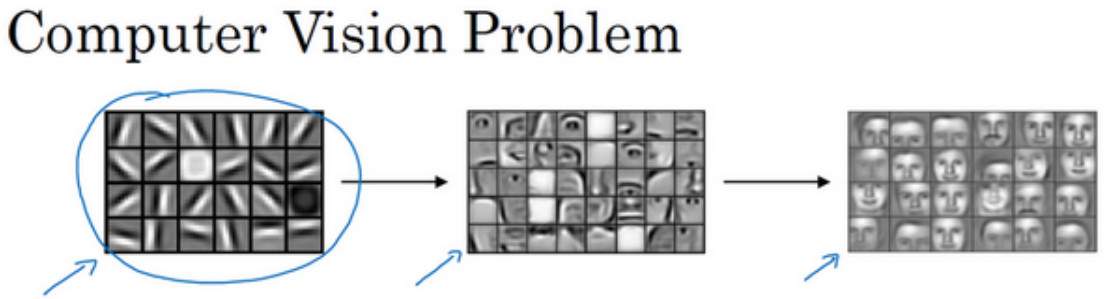
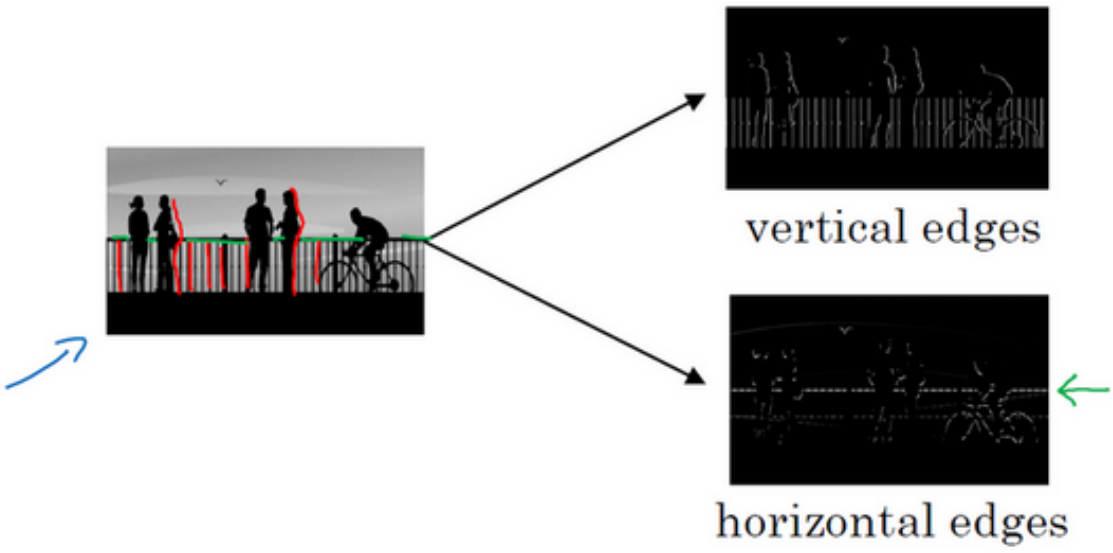
1.2 边缘检测示例（Edge detection example）

卷积运算是卷积神经网络最基本的组成部分，使用边缘检测作为入门样例。在这个视频中，你会看到卷积是如何进行运算的。



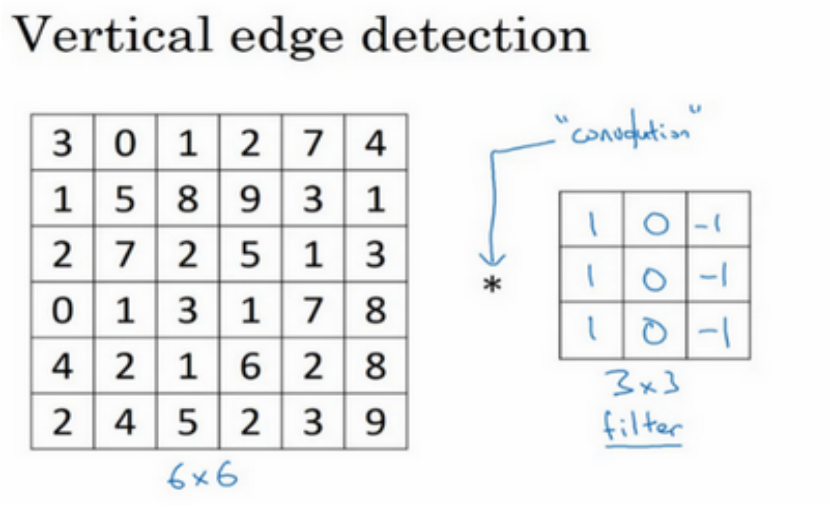
在之前的视频中，我说过神经网络的前几层是如何检测边缘的，然后，后面的层有可能检测到物体的部分区域，更靠后的一些层可能检测到完整的物体，这个例子中就是人脸。在这个视频中，你会看到如何在一张图片中进行边缘检测。



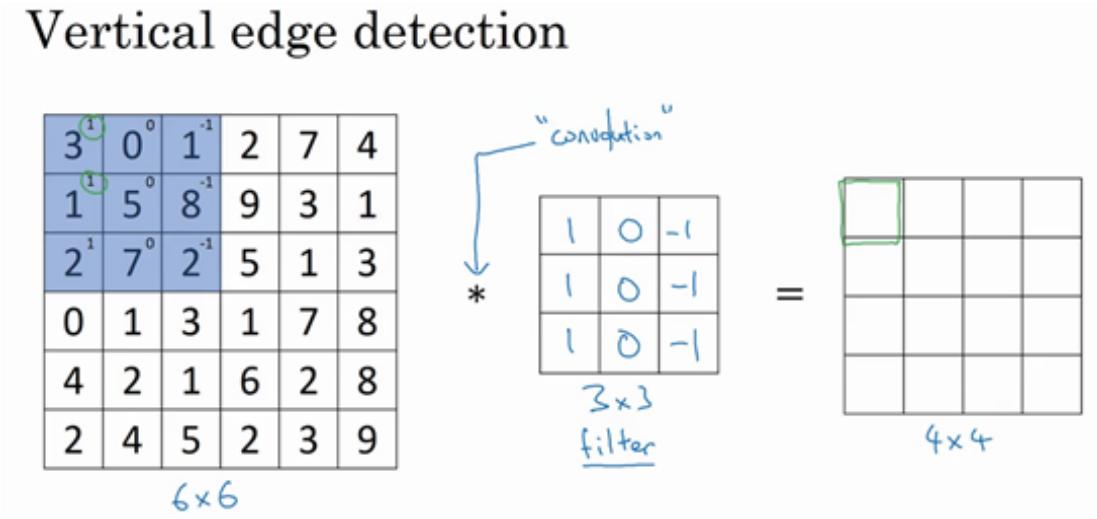
让我们举个例子，给了这样一张图片，让电脑去搞清楚这张照片里有什么物体，你可能做的第一件事是检测图片中的垂直边缘。比如说，在这张图片中的栏杆就对应垂直线，与此同时，这些行人的轮廓线某种程度上也是垂线，这些线是垂直边缘检测器的输出。同样，你可能也想检测水平边缘，比如说这些栏杆就是很明显的水平线，它们也能被检测到，结果在这。所以如何在图像中检测这些边缘？

看一个例子，这是一个 6×6 的灰度图像。因为是灰度图像，所以它是 6×6×1 的矩阵，而不是 6×6×3 的，因为没有 **RGB** 三通道。为了检测图像中的垂直边缘，你可以构造一个 3×3

矩阵。在共用习惯中，在卷积神经网络的术语中，它被称为过滤器。我要构造一个 3×3 的过滤器，像这样$\begin{bmatrix}1 & 0 & -1\\1 & 0 & -1\\1 & 0 & -1\end{bmatrix}$。在论文它有时候会被称为核，而不是过滤器，但在这个视频中，我将使用过滤器这个术语。对这个 6×6 的图像进行卷积运算，卷积运算用“*”来表示，用 3×3 的过滤器对其进行卷积。

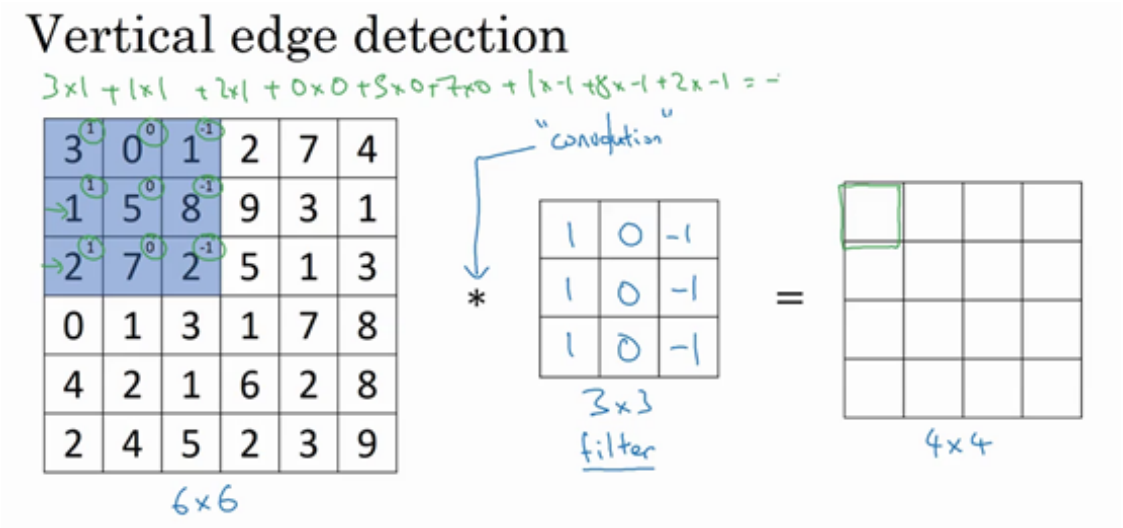


关于符号表示，有一些问题，在数学中“*”就是卷积的标准标志，但是在 **Python** 中，这个标识常常被用来表示乘法或者元素乘法。所以这个“*”有多层含义，它是一个重载符号，在这个视频中，当“*”表示卷积的时候我会特别说明。



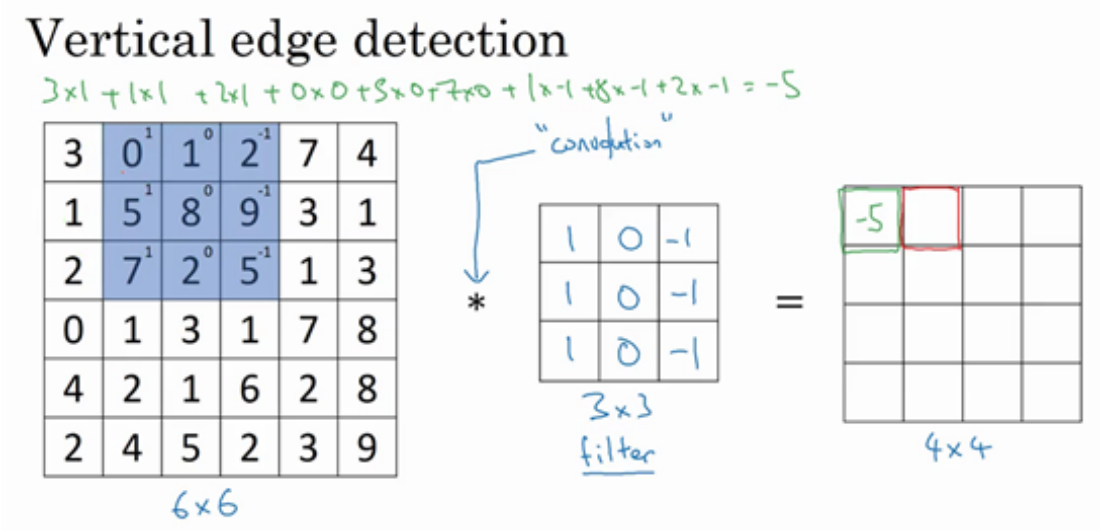
这个卷积运算的输出将会是一个 4×4 的矩阵，你可以将它看成一个 4×4 的图像。下面来说明是如何计算得到这个 4×4 矩阵的。为了计算第一个元素，在 4×4 左上角的那个元素，使用 3×3 的过滤器，将其覆盖在输入图像，如下图所示。然后进行元素乘法（**element-wise**

products）运算，所以$\begin{bmatrix} 3\times1 & 0\times0 & 1\times(1) \\ 1\times1 & 5\times0 & 8\times(-1) \\ 2\times1 & 7\times0 & 2\times(-1) \end{bmatrix}=\begin{bmatrix} 3 & 0 & -1 \\ 1 & 0 & -8 \\ 2 & 0 & -2 \end{bmatrix}$，然后将该矩阵每个元素相加得到最左上角的元素，即$3+1+2+0+0+0+(-1)+(-8)+(-2)=-5$。

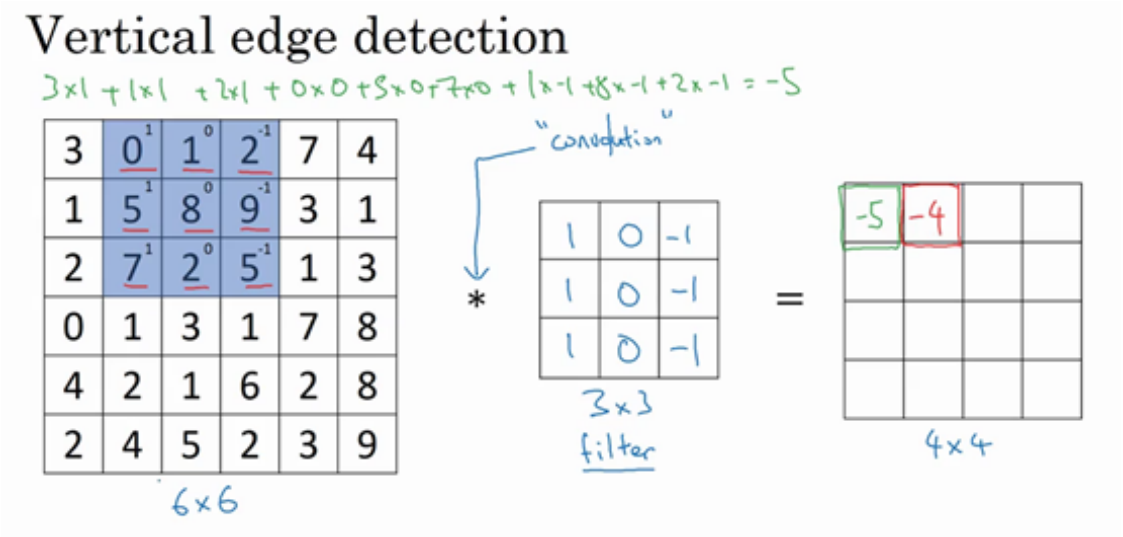


把这 9 个数加起来得到-5，当然，你可以把这 9 个数按任何顺序相加，我只是先写了第一列，然后第二列，第三列。

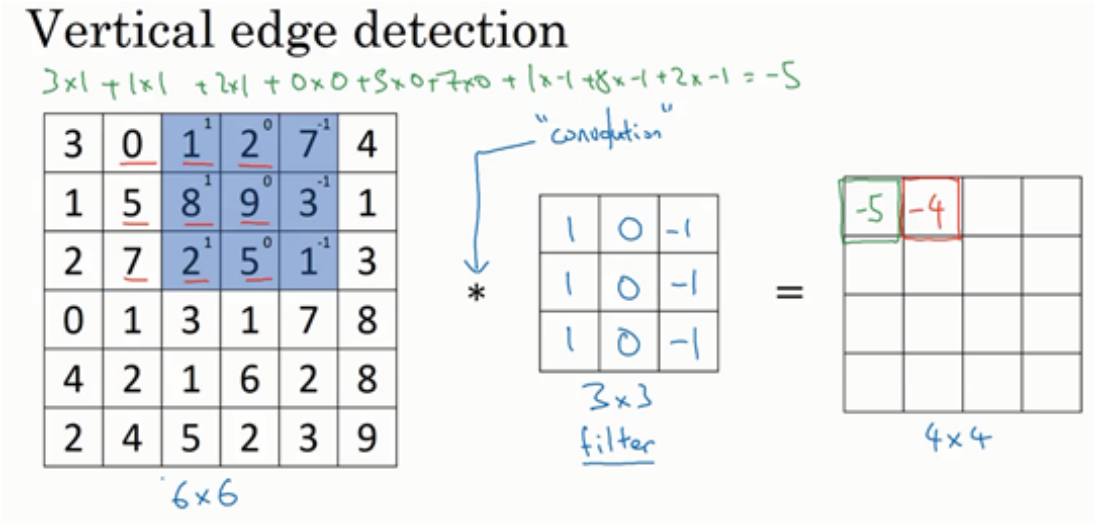
接下来，为了弄明白第二个元素是什么，你要把蓝色的方块，向右移动一步，像这样，把这些绿色的标记去掉：



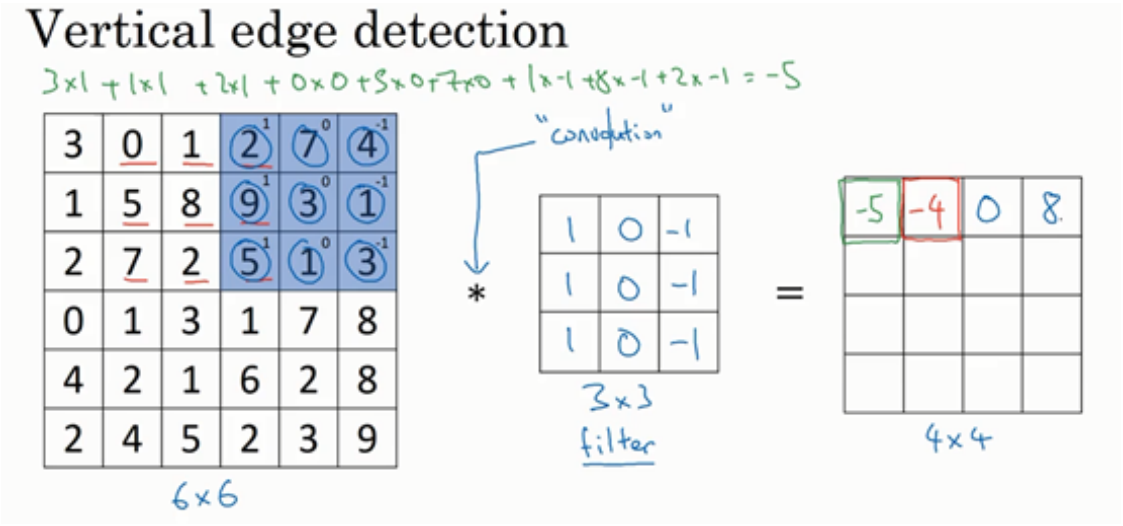
继续做同样的元素乘法，然后加起来，所以是 $0\times1+5\times1+7\times1+1\times0+8\times0+2\times0+2\times(-1)+9\times(-1)+5\times(-1)=-4$。



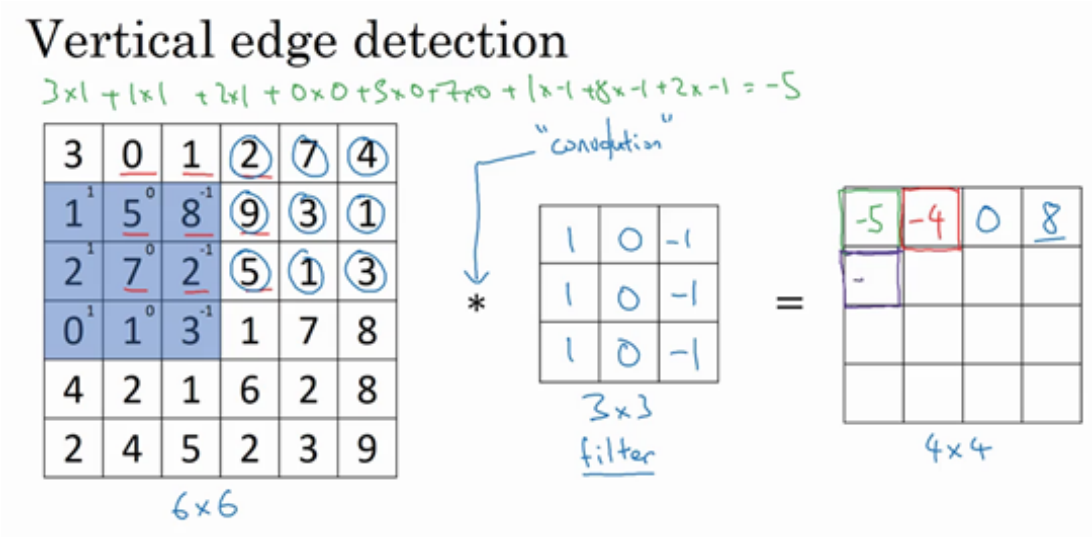
接下来也是一样，继续右移一步，把 9 个数的点积加起来得到 0。



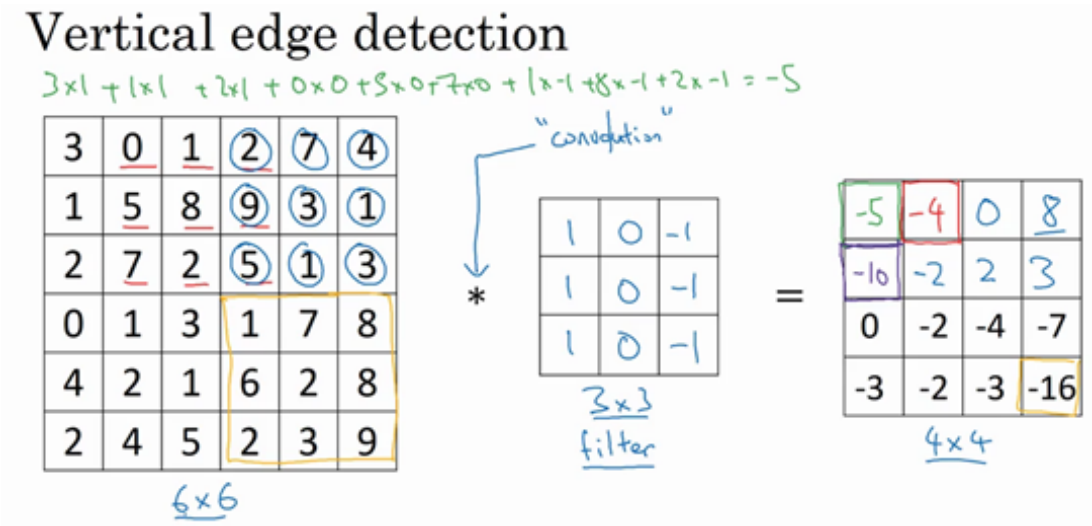
继续移得到 8，验证一下：$2\times1+9\times1+5\times1+7\times0+3\times0+1\times0+4\times(-1)+1\times(-1)+3\times(-1)=8$。



接下来为了得到下一行的元素，现在把蓝色块下移，现在蓝色块在这个位置：



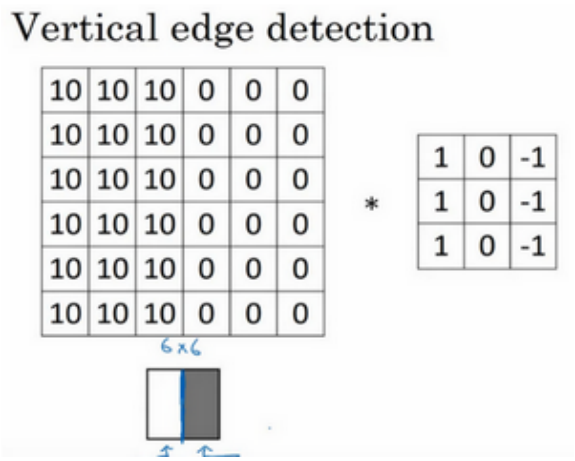
重复进行元素乘法，然后加起来。通过这样做得到-10。再将其右移得到-2，接着是 2，3。以此类推，这样计算完矩阵中的其他元素。



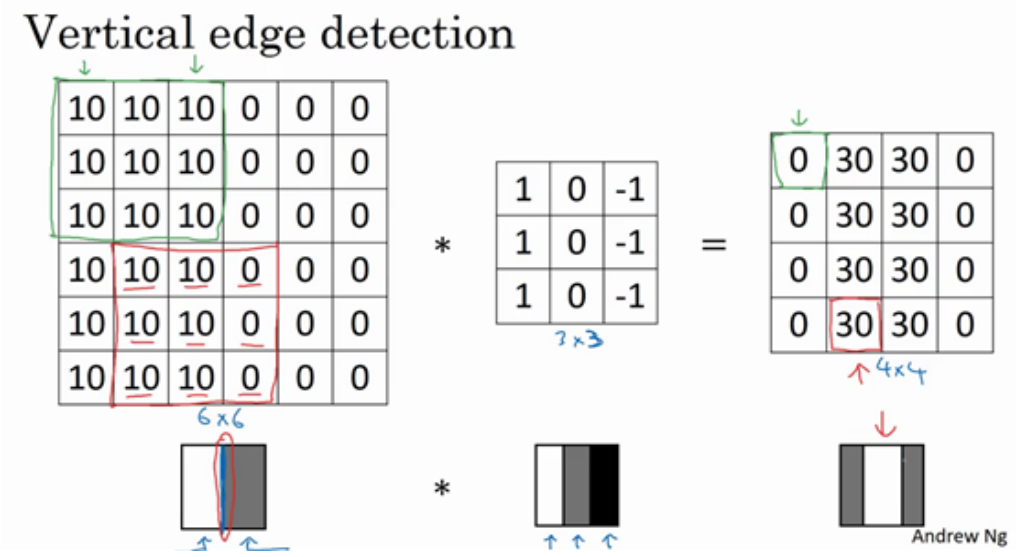
为了说得更清楚一点，这个-16 是通过底部右下角的 3×3 区域得到的。

因此 6×6 矩阵和 3×3 矩阵进行卷积运算得到 4×4 矩阵。这些图片和过滤器是不同维度的矩阵，但左边矩阵容易被理解为一张图片，中间的这个被理解为过滤器，右边的图片我们可以理解为另一张图片。这个就是垂直边缘检测器，下一页中你就会明白。

在往下讲之前，多说一句，如果你要使用编程语言实现这个运算，不同的编程语言有不同的函数，而不是用“*”来表示卷积。所以在编程练习中，你会使用一个叫 **conv_forward** 的函数。如果在 **tensorflow** 下，这个函数叫 **tf.conv2d**。在其他深度学习框架中，在后面的课程中，你将会看到 **Keras** 这个框架，在这个框架下用 **Conv2D** 实现卷积运算。所有的编程框架都有一些函数来实现卷积运算。



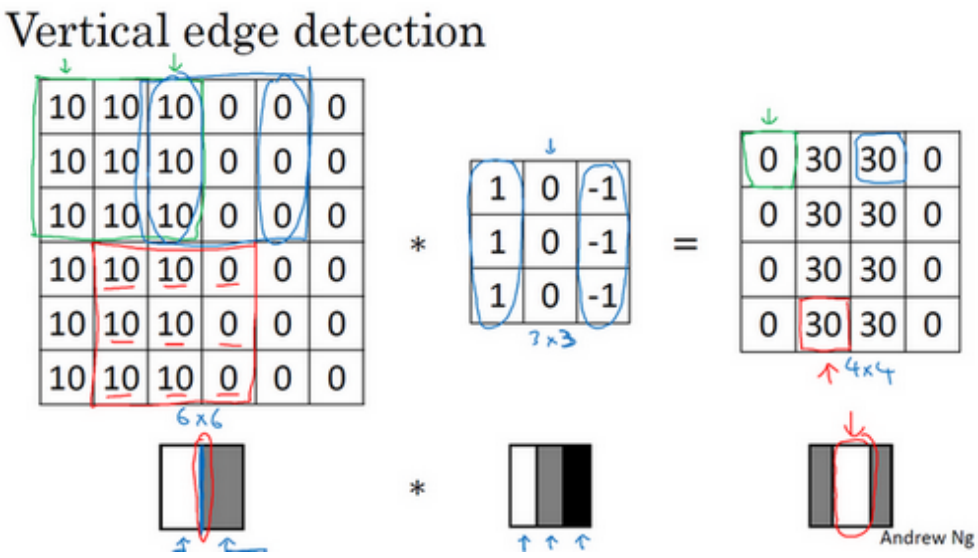
为什么这个可以做垂直边缘检测呢？让我们来看另外一个例子。为了讲清楚，我会用一个简单的例子。这是一个简单的 6×6 图像，左边的一半是 10，右边一般是 0。如果你把它当成一个图片，左边那部分看起来是白色的，像素值 10 是比较亮的像素值，右边像素值比较暗，我使用灰色来表示 0，尽管它也可以被画成黑的。图片里，有一个特别明显的垂直边缘在图像中间，这条垂直线是从黑到白的过渡线，或者从白色到深色。



所以，当你用一个 3×3 过滤器进行卷积运算的时候，这个 3×3 的过滤器可视化为下面这个样子，在左边有明亮的像素，然后有一个过渡，0 在中间，然后右边是深色的。卷积运算后，你得到的是右边的矩阵。如果你愿意，可以通过数学运算去验证。举例来说，最左上角的元素 0，就是由这个 3×3 块（绿色方框标记）经过元素乘积运算再求和得到的，$10 \times 1 + 10 \times 1 + 10 \times 1 + 10 \times 0 + 10 \times 0 + 10 \times 0 + 10 \times (-1) + 10 \times (-1) + 10 \times (-1) = 0$

。相反这个 30 是由这个（红色方框标记）得到的，

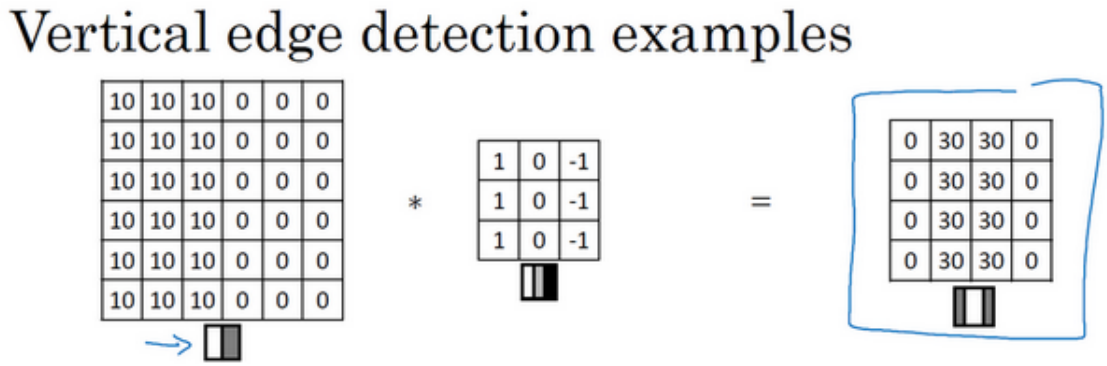
$10 \times 1 + 10 \times 1 + 10 \times 1 + 10 \times 0 + 10 \times 0 + 10 \times 0 + 0 \times (-1) + 0 \times (-1) + 0 \times (-1) = 30$。



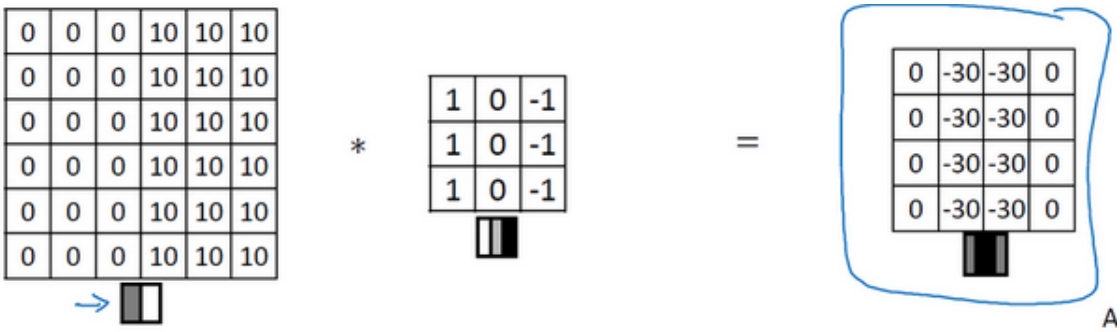
如果把最右边的矩阵当成图像，它是这个样子。在中间有段亮一点的区域，对应检查到这个 6×6 图像中间的垂直边缘。这里的维数似乎有点不正确，检测到的边缘太粗了。因为在这个例子中，图片太小了。如果你用一个 1000×1000 的图像，而不是 6×6 的图片，你会发现其会很好地检测出图像中的垂直边缘。在这个例子中，在输出图像中间的亮处，表示在图像中间有一个特别明显的垂直边缘。从垂直边缘检测中可以得到的启发是，因为我们使用 3×3 的矩阵（过滤器），所以垂直边缘是一个 3×3 的区域，左边是明亮的像素，中间的并不需要考虑，右边是深色像素。在这个 6×6 图像的中间部分，明亮的像素在左边，深色的像素在右边，就被视为一个垂直边缘，卷积运算提供了一个方便的方法来发现图像中的垂直边缘。

1.3 更多边缘检测内容（More edge detection）

你已经见识到用卷积运算实现垂直边缘检测，在本视频中，你将学习如何区分正边和负边，这实际就是由亮到暗与由暗到亮的区别，也就是边缘的过渡。你还能了解到其他类型的边缘检测以及如何去实现这些算法，而不要总想着去自己编写一个边缘检测程序，让我们开始吧。



还是上一个视频中的例子，这张 6×6 的图片，左边较亮，而右边较暗，将它与垂直边缘检测滤波器进行卷积，检测结果就显示在了右边这幅图的中间部分。

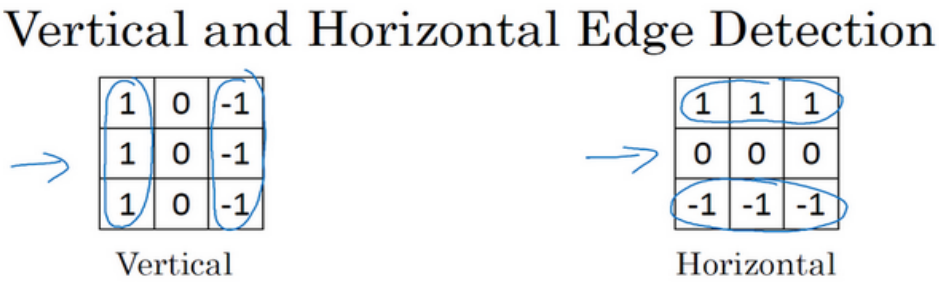


现在这幅图有什么变化呢？它的颜色被翻转了，变成了左边比较暗，而右边比较亮。现在亮度为 10 的点跑到了右边，为 0 的点则跑到了左边。如果你用它与相同的过滤器进行卷积，最后得到的图中间会是-30，而不是 30。如果你将矩阵转换为图片，就会是该矩阵下面图片的样子。现在中间的过渡部分被翻转了，**之前的 30 翻转成了-30，表明是由暗向亮过渡，而不是由亮向暗过渡。**

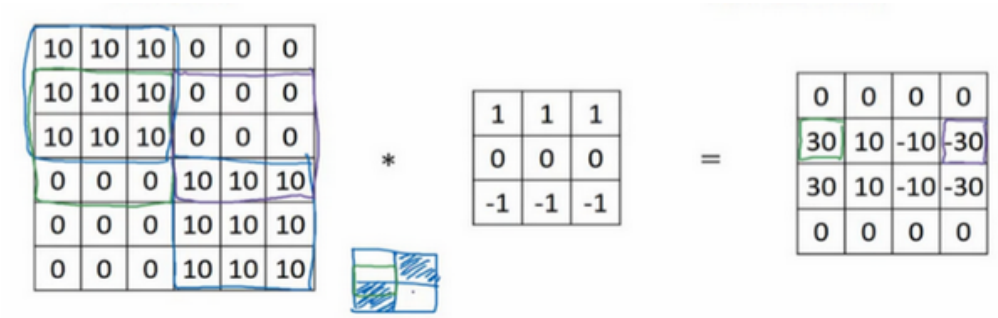
如果你不在乎这两者的区别，你可以取出矩阵的绝对值。但这个特定的过滤器确实可以为我们区分这两种明暗变化的区别。

再来看看更多的边缘检测的例子，我们已经见过这个 3×3 的过滤器，它可以检测出垂直的边缘。所以，看到右边这个过滤器，我想你应该猜出来了，它能让你检测出水平的边缘。提醒一下，一个垂直边缘过滤器是一个 3×3 的区域，它的左边相对较亮，而右边相对较暗。

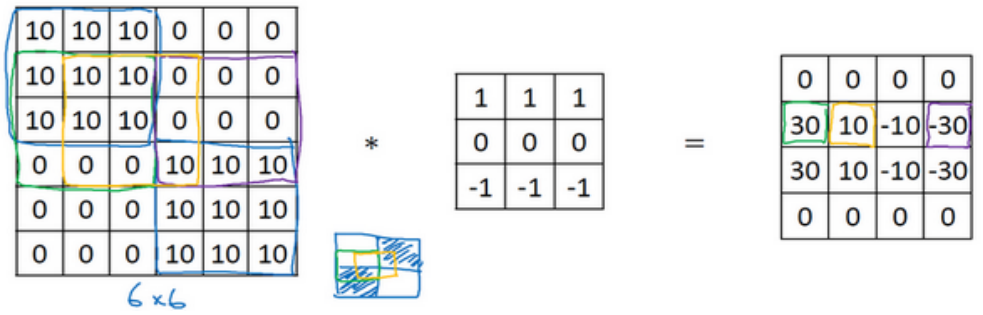
相似的，右边这个水平边缘过滤器也是一个 3×3 的区域，它的上边相对较亮，而下方相对较暗。



这里还有个更复杂的例子，左上方和右下方都是亮度为 10 的点。如果你将它绘成图片，右上角是比较暗的地方，这边都是亮度为 0 的点，我把这些比较暗的区域都加上阴影。而左上方和右下方都会相对较亮。如果你用这幅图与水平边缘过滤器卷积，就会得到右边这个矩阵。



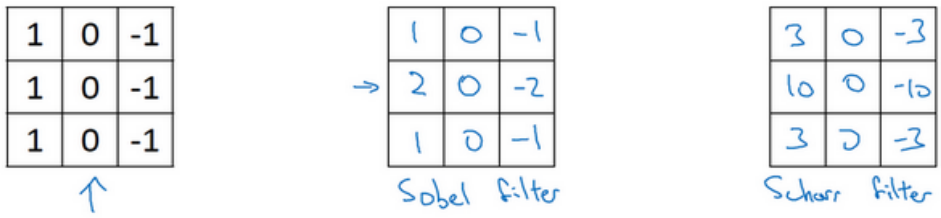
再举个例子，这里的 30（右边矩阵中绿色方框标记元素）代表了左边这块 3×3 的区域（左边矩阵绿色方框标记部分），这块区域确实是上边比较亮，而下边比较暗的，所以它在这里发现了一条正边缘。而这里的-30（右边矩阵中紫色方框标记元素）又代表了左边另一块区域（左边矩阵紫色方框标记部分），这块区域确实是底部比较亮，而上边则比较暗，所以在这里它是一条负边。



再次强调，我们现在所使用的都是相对很小的图片，仅有 6×6。但这些中间的数值，比如说这个 10（右边矩阵中黄色方框标记元素）代表的是左边这块区域（左边 6×6 矩阵中黄色方框标记的部分）。这块区域左边两列是正边，右边一列是负边，正边和负边的值加在一

起得到了一个中间值。但假如这个一个非常大的 1000×1000 的类似这样棋盘风格的大图，就不会出现这些亮度为 10 的过渡带了，因为图片尺寸很大，这些中间值就会变得非常小。

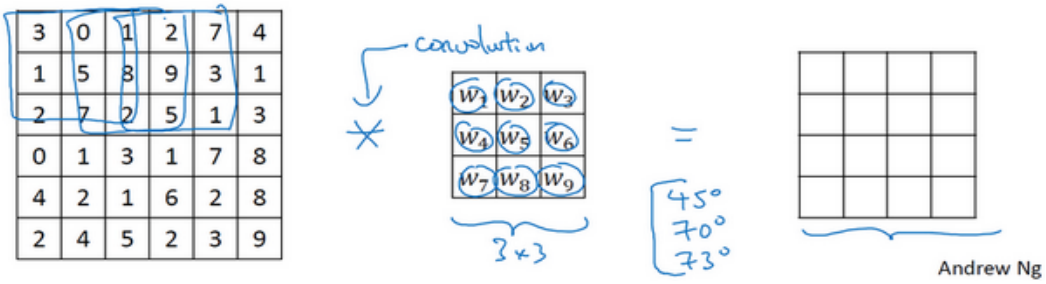
总而言之，通过使用不同的过滤器，你可以找出垂直的或是水平的边缘。但事实上，对于这个 3×3 的过滤器来说，我们使用了其中的一种数字组合。



但在历史上，在计算机视觉的文献中，曾公平地争论过怎样的数字组合才是最好的，所以你还可以使用这种：$\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}$，叫做 **Sobel 的过滤器**，它的优点在于增加了中间一行元素的权重，这使得结果的鲁棒性会更高一些。

但计算机视觉的研究者们也会经常使用其他的数字组合，比如这种：$\begin{bmatrix}3 & 0 & -3\\ 10 & 0 & -10\\ 3 & 0 & -3\end{bmatrix}$，这叫做 **Scharr 过滤器**，它有着和之前完全不同的特性，实际上也是一种垂直边缘检测，如果你将其翻转 90 度，你就能得到对应水平边缘检测。

随着深度学习的发展，我们学习的其中一件事就是当你真正想去检测出复杂图像的边缘，你不一定要去使用那些研究者们所选择的这九个数字，但你可以从中获益匪浅。**把这矩阵中的 9 个数字当成 9 个参数，并且在之后你可以学习使用反向传播算法，其目标就是去理解这 9 个参数。**



当你得到左边这个 6×6 的图片，将其与这个 3×3 的过滤器进行卷积，将会得到一个出色的边缘检测。这就是你在下节视频中将会看到的，把这 9 个数字当成参数的过滤器，通过反向传播，你可以学习这种$\begin{bmatrix}1 & 0 & -1\\ 1 & 0 & -1\\ 1 & 0 & -1\end{bmatrix}$的过滤器，或者 **Sobel** 过滤器和 **Scharr** 过滤器。还有另一种过滤器，这种过滤器对于数据的捕捉能力甚至可以胜过任何之前这些手写的过滤器。

相比这种单纯的垂直边缘和水平边缘，它可以检测出 45°或 70°或 73°，甚至是任何角度的边缘。所以将矩阵的所有数字都设置为参数，通过数据反馈，让神经网络自动去学习它们，我们会发现神经网络可以学习一些低级的特征，例如这些边缘的特征。尽管比起那些研究者们，我们要更费劲一些，但确实可以动手写出这些东西。不过构成这些计算的基础依然是卷积运算，使得反向传播算法能够让神经网络学习任何它所需要的 3×3 的过滤器，并在整幅图片上去应用它。这里，这里，还有这里（左边矩阵蓝色方框标记部分），去输出这些，任何它所检测到的特征，不管是垂直的边缘，水平的边缘，还有其他奇怪角度的边缘，甚至是其它的连名字都没有的过滤器。

所以这种将这 9 个数字当成参数的思想，已经成为计算机视觉中最为有效的思想之一。在接下来的课程中，也就是下个星期，我们将详细去探讨如何使用反向传播去让神经网络学习这 9 个数字。但在此之前，我们需要先讨论一些其它细节，比如一些基础的卷积运算的变量。在下面两节视频中，我将与你们讨论如何去使用 **padding**，以及卷积各种不同的发展，这两节内容将会是卷积神经网络中卷积模块的重要组成部分，所以我们下节视频再见。