Notebook credit: Based on the original D2L notebook here.

Pooling(池化)

下面是CNN的几个关键组成部分:(可以重复, 且并不是严格顺序)

1. 卷积层 (Convolutional Layer): 这是CNN的核心层, 主要负责从输入图像中提取特征. 通过使用一系列的过滤器(或称为核), 每个卷积层可以捕捉输入数据的局部依赖性和空间层级结构. 每个过滤器负责从输入图像中提取不同的特征, 例如边缘、角点或更复杂的纹理等。
2. 激活函数 (Activation Function): 激活函数通常应用在卷积层的输出上, 用于引入非线性, 使得网络可以学习更复杂的模式。最常用的激活函数是ReLU.
3. 池化层 (Pooling Layer): 池化层主要用于减少数据的空间大小 (即降采样)，这有助于减少参数数量和计算量，同时还能使特征检测器具有一定程度的位置不变性。常见的池化操作包括最大池化(Max Pooling) 和平均池化 (Average Pooling), 其中最大池化是提取特定窗口内的最大值，平均池化则是计算窗口内的平均值。
4. 全连接层 (Fully Connected Layer): 在多个卷积层和池化层之后, 全连接层用于将网络学习到的特征转化为最终的输出, 如分类标签 (softmax等)。在传统的CNN架构中, 全连接层通常位于网络的末端. 通过这些层, 可以将前面层次提取到的特征整合起来, 进行最终的决策。
5. 归一化层 (Normalization Layer): 归一化层, 如批量归一化(Batch Normalization), 被用于网络中以稳定学习过程，通过规范化层的输入来加速训练并提高性能。
6. 丢弃层 (Dropout Layer): 在训练过程中，丢弃层随机地丢弃（即设置为零）一部分神经元的输出，这有助于防止模型过拟合。

通常, 在处理图像时, 我们希望逐渐降低 spatial resolution of our hidden representation, 将信息聚合在一起. 这样在网络中越往上层, 每个隐藏节点敏感的 receptive field 就越大.

通常情况下，我们的最终任务会提出一些关于图像的全局性问题，例如，图像中是否有一只猫？因此，我们最终层的单元通常应该对整个输入都很敏感。通过逐步聚合信息，生成越来越粗糙的地图，我们就能实现最终学习全局表示的目标，同时保持卷积层在中间处理层的所有优点.

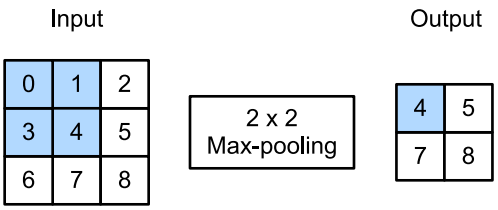
此外，在检测边缘等较低层次的特征时，我们往往希望我们的 representations 在某种程度上不受平移的影响. 例如，如果我们将黑白分明的图像 `X` 向右移动一个像素，即 `Z[i, j] = X[i, j + 1]`，那么新图像 `Z` 的输出可能会大不相同。边缘将偏移一个像素。在现实中，物体几乎不会出现在完全相同的位置。事实上，即使使用三脚架和一个静止的物体，快门移动导致的相机振动也可能会使所有物体移动一个像素左右（高端相机具有解决这一问题的特殊功能）。

本节将介绍池化层，池化层具有双重作用，一是减轻卷积层对位置的敏感性，二是对表示进行spatially downsampling(降采样) representations..

Maximum Pooling and Average Pooling

Like convolutional layers, *pooling* operators consist of a fixed-shape window that is slid over all regions in the input according to its stride, computing a single output for each location traversed by the fixed-shape window (sometimes known as the *pooling window*). However, unlike the cross-correlation computation of the inputs and kernels in the convolutional layer, the pooling layer contains no parameters (there is no *kernel*). Instead, pooling operators are deterministic, typically calculating either the maximum or the average value of the elements in the pooling window. These operations are called *maximum pooling* (*max pooling* for short) and *average pooling*, respectively.

In both cases, as with the cross-correlation operator, we can think of the pooling window as starting from the upper-left of the input tensor and sliding across the input tensor from left to right and top to bottom. At each location that the pooling window hits, it computes the maximum or average value of the input subtensor in the window, depending on whether max or average pooling is employed.



The output tensor has a height of 2 and a width of 2. The four elements are derived from the maximum value in each pooling window:

$$\max(0, 1, 3, 4) = 4,$$
$$\max(1, 2, 4, 5) = 5,$$
$$\max(3, 4, 6, 7) = 7,$$
$$\max(4, 5, 7, 8) = 8.$$

A pooling layer with a pooling window shape of $p \times q$ is called a $p \times q$ pooling layer. The pooling operation is called $p \times q$ pooling.

Let us return to the object edge detection example mentioned at the beginning of this section. Now we will use the output of the convolutional layer as the input for 2×2 maximum pooling. Set the convolutional layer input as `X` and the pooling layer output as `Y`. Whether or not the values of `X[i, j]` and `X[i, j + 1]` are different, or `X[i, j + 1]` and `X[i, j + 2]` are different, the pooling layer always outputs `Y[i, j] = 1`. That is to say, using the 2×2 maximum pooling layer, we can

still detect if the pattern recognized by the convolutional layer moves no more than one element in height or width.

In the code below, we (**implement the forward propagation of the pooling layer**) in the `pool2d` function. This function is similar to the `corr2d` function. However, here we have no kernel, computing the output as either the maximum or the average of each region in the input.

```
import tensorflow as tf


def pool2d(X, pool_size, mode='max'):
    p_h, p_w = pool_size
    Y = tf.Variable(tf.zeros((X.shape[0] - p_h + 1, X.shape[1] - p_w +1)))
    for i in range(Y.shape[0]):
        for j in range(Y.shape[1]):
            if mode == 'max':
                Y[i, j].assign(tf.reduce_max(X[i: i + p_h, j: j + p_w]))
            elif mode =='avg':
                Y[i, j].assign(tf.reduce_mean(X[i: i + p_h, j: j + p_w]))
    return Y
```

We can construct the input tensor `X` in the figure above to [**validate the output of the two-dimensional maximum pooling layer**].

```
X = tf.constant([[0.0, 1.0, 2.0], [3.0, 4.0, 5.0], [6.0, 7.0, 8.0]])
pool2d(X, (2, 2))
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32, numpy=
array([[4., 5.],
       [7., 8.]], dtype=float32)>
```

Also, we experiment with (**the average pooling layer**).

```
pool2d(X, (2, 2), 'avg')
```

```
<tf.Variable 'Variable:0' shape=(2, 2) dtype=float32, numpy=
array([[2., 3.],
       [5., 6.]], dtype=float32)>
```

[Padding and Stride]

As with convolutional layers, pooling layers can also change the output shape. And as before, we can alter the operation to achieve a desired output shape by padding the input and adjusting the stride. We can demonstrate the use of padding and strides in pooling layers via the built-in two-

dimensional maximum pooling layer from the deep learning framework. We first construct an input tensor `X` whose shape has four dimensions, where the number of examples (batch size) and number of channels are both 1.

It is important to note that tensorflow prefers and is optimized for *channels-last* input.

```
X = tf.reshape(tf.range(16, dtype=tf.float32), (1, 4, 4, 1))
X
```

```
<tf.Tensor: shape=(1, 4, 4, 1), dtype=float32, numpy=
array([[[[ 0.],
         [ 1.],
         [ 2.],
         [ 3.]],

        [[ 4.],
         [ 5.],
         [ 6.],
         [ 7.]],

        [[ 8.],
         [ 9.],
         [10.],
         [11.]],

        [[12.],
         [13.],
         [14.],
         [15.]]]], dtype=float32)>
```

By default, (**the stride and the pooling window in the instance from the framework's built-in class have the same shape.**) Below, we use a pooling window of shape `(3, 3)`, so we get a stride shape of `(3, 3)` by default.

```
pool2d = tf.keras.layers.MaxPool2D(pool_size=[3, 3])
pool2d(X)
```

```
<tf.Tensor: shape=(1, 1, 1, 1), dtype=float32, numpy=array([[[[10.]]]],
dtype=float32)>
```

[**The stride and padding can be manually specified.**]

```
paddings = tf.constant([[0, 0], [1,0], [1,0], [0,0]])
X_padded = tf.pad(X, paddings, "CONSTANT")
pool2d = tf.keras.layers.MaxPool2D(pool_size=[3, 3], padding='valid',
                                   strides=2)
pool2d(X_padded)
```

```
<tf.Tensor: shape=(1, 2, 2, 1), dtype=float32, numpy=
array([[[[ 5.],
         [ 7.]],

        [[13.],
         [15.]]]], dtype=float32)>
```

Of course, we can specify an arbitrary rectangular pooling window and specify the padding and stride for height and width, respectively.

```
paddings = tf.constant([[0, 0], [0, 0], [1, 1], [0, 0]])
X_padded = tf.pad(X, paddings, "CONSTANT")

pool2d = tf.keras.layers.MaxPool2D(pool_size=[2, 3], padding='valid',
                                   strides=(2, 3))
pool2d(X_padded)
```

```
<tf.Tensor: shape=(1, 2, 2, 1), dtype=float32, numpy=
array([[[[ 5.],
         [ 7.]],

        [[13.],
         [15.]]]], dtype=float32)>
```

Multiple Channels

在处理 multi-channel input data 时，pooling layer 会分开 pool 每个 input channel, 而不是像 convolutional layer 那样将各 channels 的 inputs 相加. 这意味着 pooling layer 的 output channels 数量与 input channels 的数量相同. 下面，我们将 concatenate tensors `X` and `X + 1` on the channel dimension to construct an input with 2 channels.

Note that this will require a concatenation along the last dimension for TensorFlow due to the channels-last syntax.

```
X = tf.concat([X, X + 1], 3)  # Concatenate along `dim=3` due to channels-last synta
```

As we can see, the number of output channels is still 2 after pooling.

```
paddings = tf.constant([[0, 0], [1,0], [1,0], [0,0]])
X_padded = tf.pad(X, paddings, "CONSTANT")
pool2d = tf.keras.layers.MaxPool2D(pool_size=[3, 3], padding='valid',
                                   strides=2)
pool2d(X_padded)
```

```
<tf.Tensor: shape=(1, 2, 2, 2), dtype=float32, numpy=
array([[[[ 5.,  6.],
         [ 7.,  8.]],

        [[13., 14.],
         [15., 16.]]]], dtype=float32)>
```

Summary

- 对于池化窗口中的输入元素, maximum pooling operation 将最大值作为 output, average pooling operation 将平均值作为 output.
- pooling layer 的主要好处之一是减轻 convolution layer 对位置的过度敏感性.
- 我们可以指定 pooling layer 的 padding 和 stride.
- Maximum pooling 结合 大于 1 的 stride 可用于减少 spatial dimensions (e.g., width and height).
- pooling layer 的 output channels 数量与 input channels 的数量相同.