卷积层的输出：

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

如果 W2 或者 H2 计算得到不为整数：
not an integer, indicating that the neurons don't "fit" neatly and symmetrically across the input. Therefore, this setting of the hyperparameters is considered to be invalid, and a ConvNet library could throw an exception or zero pad the rest to make it fit, or crop the input to make it fit, or something. >>

使用 *tf.nn.conv2d* 或 *tf.layers.conv2d* 作为卷积函数，当不为整时根据 padding 取值（默认为"valid"）由下计算：

`padding`： `"valid"` 或者 `"same"`（不区分大小写）。`"valid"` 表示不够卷积核大小的块就丢弃，`"same"` 表示不够卷积核大小的块就补0。

`"valid"` 的输出形状为

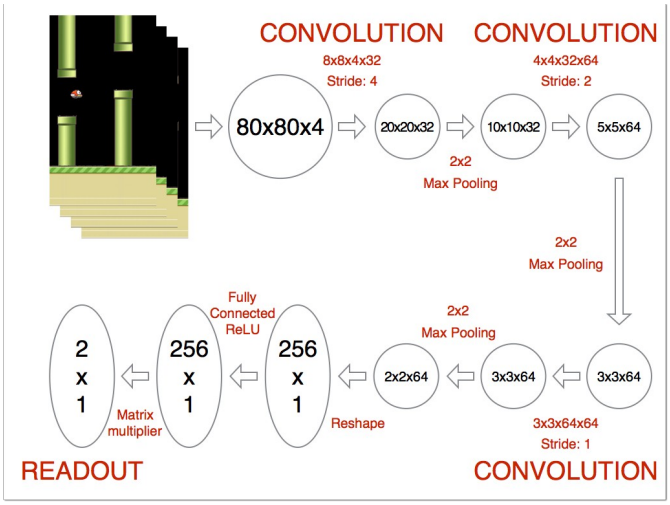
$$L_{new} = ceil(\frac{L - F + 1}{S})$$

`"same"` 的输出形状为

$$L_{new} = ceil(\frac{L}{S})$$

其中，L 为输入的 size（高或宽），F 为 filter 的 size，S 为 strides 的大小，$ceil()$ 为向上取整。

比如 DQN 解决 `Flappy bird` 中的 DQN 网络，其中卷积和池化用到的函数 `padding="same"`，可依次得到每后一层的 `size`。 >>



再比较下 `tf.nn.conv2d` 或 `tf.layers.conv2d` 两个函数，拿上图第一层卷积为例，先定义输入

```
input = tf.placeholder(tf.float32, shape=[None, 80, 80, 4])
```

若使用 `tf.nn.conv2d`:

```
kernel = tf.truncated_normal([8, 8, 4, 32], stddev = 0.01)
h1 = tf.nn.conv2d(input, kernel, strides = [1, 4, 4, 1], padding = "SAME")
h_conv1 = tf.nn.relu(h1)
```

若使用 `tf.layers.conv2d`:

```
h_conv1 = tf.layers.conv2d(input, 32, 8, strides=4, padding = "SAME", activation=tf.nn.relu)
```

或补全：

```
h_conv1 = tf.layers.conv2d(input, 32, (8, 8), strides=(4, 4), padding = "SAME", activation=tf.nn.relu)
```

反卷积的输出：
反卷积的过程和卷积操作相反，因此由卷积层的输入输出关系可以得到反卷积层的输入输出关系：

If you assume the following notation, `output = o`, `input = i`, `kernel = k`, `stride = s`, `padding = p`, the shape of the output will be:

$$o = s(i - 1) + k - 2p.$$

但是也存在如上正算卷积所得非整的情况，因此可以直接根据 padding 的不同取值对应的公式来求解输出的 size
举一例，先定义输入：

```
a = np.array([[1,1],[2,2]], dtype=np.float32)
a = np.reshape(a, [1,2,2,1])   # input 须为 a 4-D tensor: [batch, in_height, in_width, in_channel]
x = tf.constant(a, dtype=tf.float32)
```

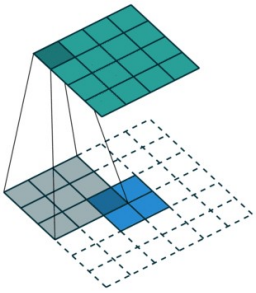
若使用 tf.nn.conv2d_transpose 函数，可以指定输出 size

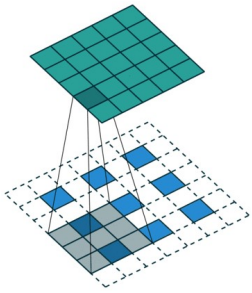
```
kernel = tf.constant(1.0, shape=[3,3,1,1])
upsample_x = tf.nn.conv2d_transpose(x, kernel, output_shape=(1,4,4,1), strides=(1,2,2,1), padding='SAME')
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    print(sess.run(upsample_x))
```

或指定 `output_shape=(1,3,3,1)`，但一定要科学，在此例中输出的 size 只能为 3 或 4（padding=”SAME”，反卷积的输出或卷积的输入为 2，因此反卷积的输出或卷积的输入只能为 3 或 4），如果令 output_shape=(1,5,5,1)则就会报错
若使用 tf.layers.conv2d_transpose(inputs,filters,kernel_size,strides=(1, 1))，不能指定输出 size，应该是按照最一般情况运行的，这里和指定(1,4,4,1)的结果相同

```
upsample_x = tf.layers.conv2d_transpose(x, 1, 3, strides=2, padding='SAME',
kernel_initializer=tf.ones_initializer())
```

再说说 strides，当其值为 1 时外围填充，如下左图所示；当其大于 1 时为间隙填充，如下右图所示：





通过上边的例子可以看到 tf.layer 相比与 tf.nn 库的函数更为简洁方便，一个可以集成一个层的全部定义，而 tf.nn 无法初始化 kernel，也无法在函数里直接给定激活函数。