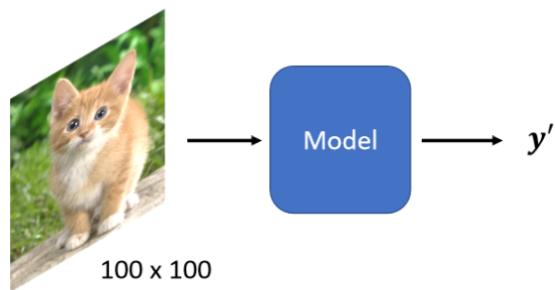


一、李宏毅2021春机器学习课程第三节：卷积神经网络(CNN)

1 图像分类问题

假设我们现在要做图像的分类，也就是给机器一张图片，它要去判断这张图片里的东西属于什么类别，那我们应该怎么做呢？



(All the images to be classified have the same size.)

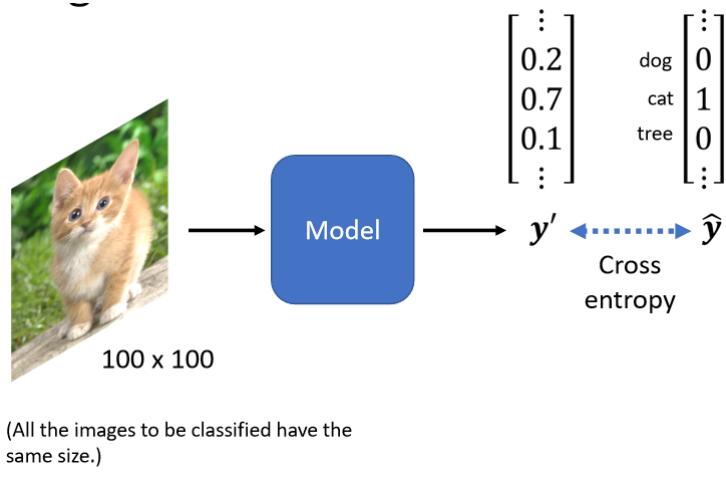
假设我们的模型输入的图片大小是固定的，就比如是 100×100 ，当然图片可能有大有小，我们需要提前把所有图片都 **Rescale** 成大小一样，再输入我们的系统。

由于模型的目标是分类，我们可以把每一个类别，表示成一个 **One-Hot** 的向量，目标向量就叫做 \hat{y} ，这里假设我们的目标类别是猫，那它的 One-Hot 向量表示如下所示：

$$\begin{matrix} & \begin{bmatrix} \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \\ \text{dog} & \\ \text{cat} & \\ \text{tree} & \end{matrix}$$

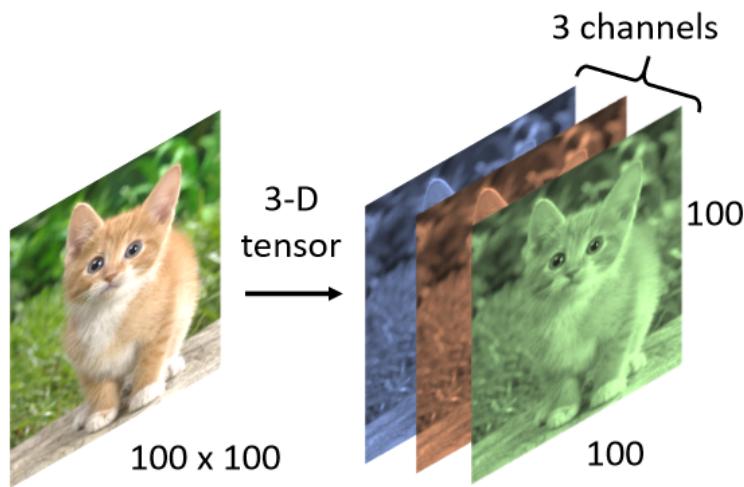
\hat{y}

之前在2.5节中已经提到了分类问题的处理方式，这里对于图像的分类也可以应用类似的方式，我们将模型的输出通过 Softmax 层，输出是我们的预测类别向量 y' ，然后我们希望 y' 和 \hat{y} 的 Cross Entropy 越小越好。

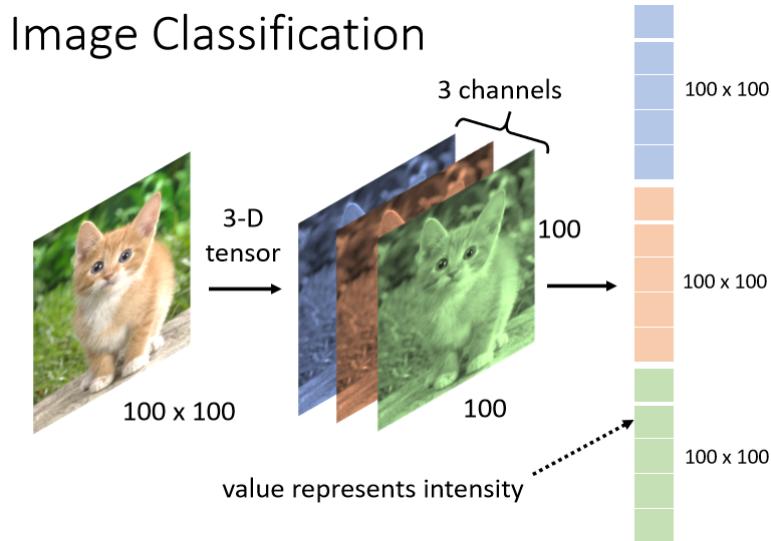


接下来的问题是，我们要怎么把一张图像作为我们模型的输入呢？

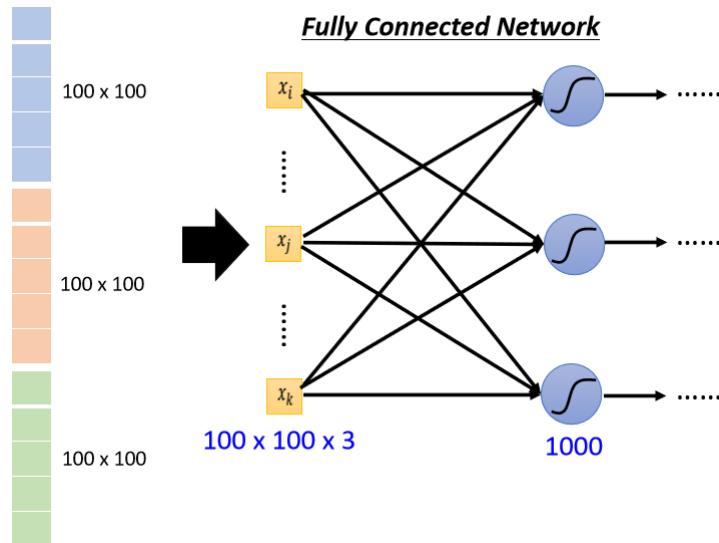
到目前为止我们所讲的 Network，它的输入其实都是一个向量，所以我们只要能够把一张图片变成一个向量，我们就可以把它当做是 Network 的输入。而一张图片就是一个三维的 Tensor，如果不太清楚 Tensor 是什么的话可以看看这篇文章 [笔记 | 什么是张量 \(tensor\) & 深度学习 - 知乎\(zhihu.com\)](#)，简单来说，Tensor 实际上就是个多维数组 (multidimensional array)。其中一维代表图片的宽，另外一维代表图片的高，还有一维代表图片的 Channel 的数目 (彩色图片的RGB三个通道)。



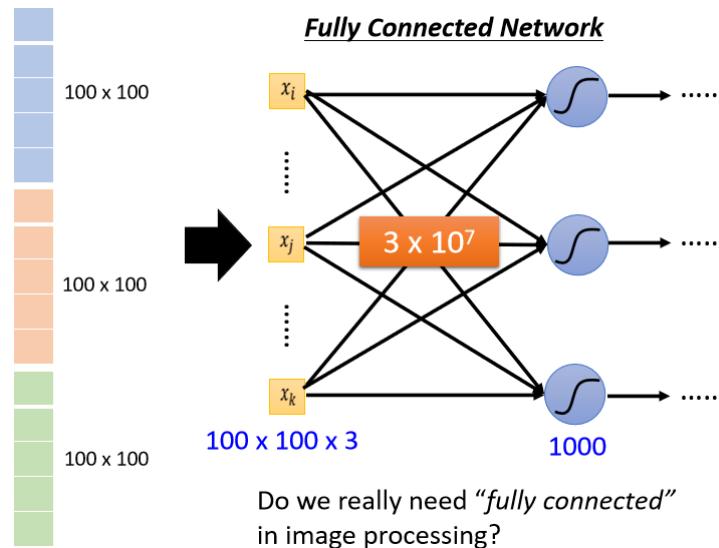
一个简单的想法：把这个三维的 Tensor 拉直，就可以作为 Network 的输入了。



在这个例子里面一张图片总共有 $100 \times 100 \times 3$ 个数字，把这些数字通通拿出来排成一排，就是一个巨大的向量，这个向量可以就作为 Network 的输入。目前我们还只讲过了 Fully Connected Network，那如果我们把这个向量当做 Network 的输入，Input 这边 Feature Vector 的长度就是 $100 \times 100 \times 3$ 。



那现在假设我们第一层的 Neuron 的数目有 1000 个，由于每一个 Neuron 跟输入向量的每一个数值，都会有一个 Weight，所以如果输入的向量长度是 $100 \times 100 \times 3$ ，有 1000 个 Neuron，那我们现在第一层的 Weight，就有 $1000 \times 100 \times 100 \times 3$,也就是 **3×10^7 的 7 次方**。



这个参数数量非常巨大，而且这还是只考虑了一层的情况，这么大的参数量会导致怎样的问题呢？

虽然随着参数的增加，我们可以增加模型的弹性，模型拟合数据的能力增强了，但是我们也**增加了 Overfitting 的风险**，并且这么大的参数量也会**带来很大的计算量**。所有接下来我们需要思考的一个问题是，怎么减少参数的个数呢？

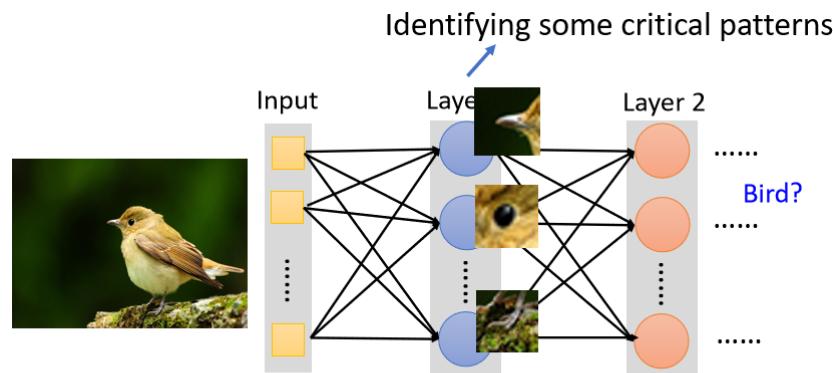
考虑到我们的输入是图像信息，我们可以从图像本身的特性入手，**其实我们并不一定需要 Fully Connected** 这件事，我们其实不需要每一个 Neuron，跟图像的每一个像素点之间都有一个 Weight。

接下来就是图像信息本身特性的一些观察。

2 观察一：通过某些模式来识别

假如现在让你辨认一个画面中是否有鸟，你会怎么做呢？

你会去看现在的画面中是不是存在鸟的形体，**是不是有鸟嘴，是不是有鸟爪，有翅膀和眼睛吗等等，而绿色树叶这些背景信息其实是不重要的。**所有这就启发我们，我们的 Neuron 能不能用来通过检测当前的图片中是否存在一些与鸟的特点相关的重要的 Pattern，从而判断图片中有没有鸟呢？

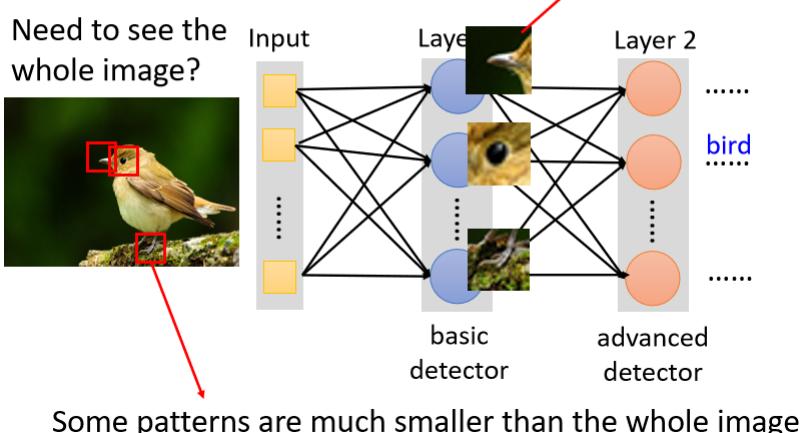


Perhaps human also identify birds in a similar way ... 😊

在这个想法下，假设我们现在用 Neuron 做的事情，其实就是判断说现在有没有某种 Pattern 出现，那**也许我们并不需要每一个 Neuron 都去看一张完整的图片**，因为那些比较重要的 Pattern，比如说鸟嘴、眼睛，并不需要看到整张完整的图片，才能够得到这些资讯。

Observation 1

A neuron does not have to see the whole image.



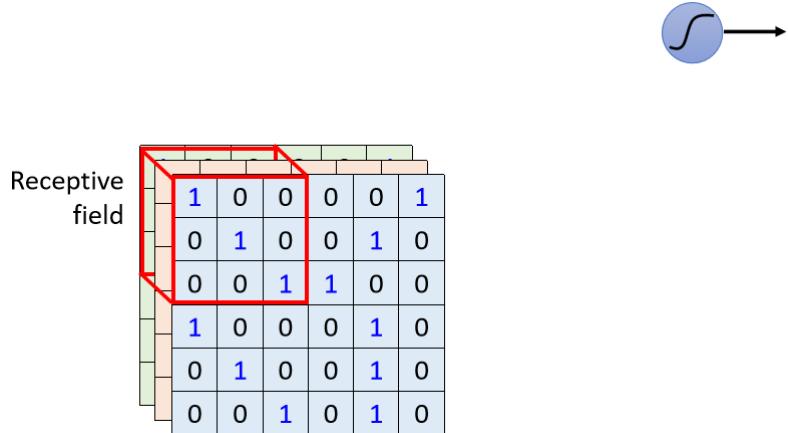
Some patterns are much smaller than the whole image.

所以这些 Neuron 也许根本就不需要把整张图片当作输入，它们只需要把图片的一小部分当作输入，就足以让它们侦测某些特别关键的 Pattern 有没有出现了。根据这个观察，我们就可以做接下来的第一个简化。

3 简化一：各扫门前雪

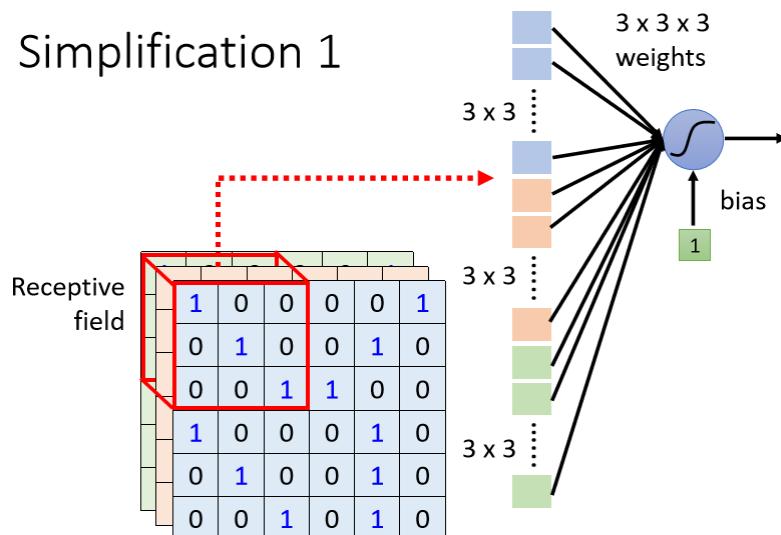
在 CNN 里面有一个这样的做法，我们会设定一个区域叫做 **Receptive Field**（感受野），每一个 **Neuron** 都只关心自己的 **Receptive Field** 里面发生的事情就好了，也就是“各扫门前雪”就行。

举例来说，我们定义一个**蓝色的 Neuron**，它只关心左上角的 $3 \times 3 \times 3$ 个数值，它并不在意自己的 **Receptive Field** 之外有什么东西。

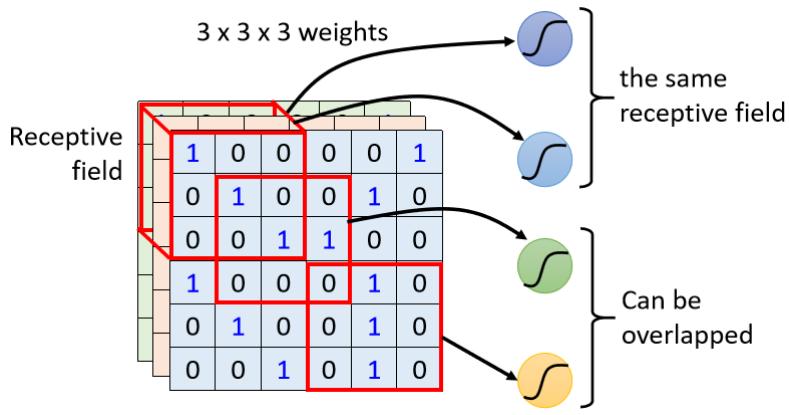


它要做的事情就是：

- 把这 $3 \times 3 \times 3$ 的数值拉直，变成一个长度是 $3 \times 3 \times 3$ 也就是 27 维的向量，再把这 27 维的向量作为这个 Neuron 的输入，
- 这个 Neuron 会给 27 维的向量的每一个 Dimension 一个 Weight，所以这个 Neuron 有 27 个 Weight，
- 再加上 Bias 得到的输出，这个输出再送给下一层的 Neuron 当作输入。



所以每一个 Neuron 都只考虑自己的 Receptive Field，而这个 Receptive Field 完全是由你自己决定的。



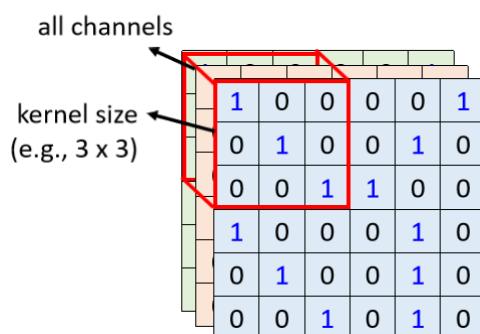
- **Receptive Field 彼此之间可以是重叠的**, 比如上图中绿色的 Neuron 跟蓝色的 Neuron 还有黄色的 Neuron 都有一些重叠的空间。
- 也可以**两个不同的 Neuron 有相同的Receptive Field**, 也许一个范围只使用一个 Neuron 没办法侦测所有的 Pattern, 所以同个范围可以有多个不同的 Neuron 来侦测。
- **Receptive Field 可以有大有小**, 毕竟 Pattern 也是有大有小的, 有的 Pattern 也许在 3×3 的范围内就可以被侦测出来, 有的 Pattern 也许要 11×11 的范围才能被侦测出来。但[一文读懂VGG网络 - 知乎\(zhihu.com\)](#)中提到了, VGG16 相比 AlexNet 的一个改进是采用连续的几个 3×3 的卷积核代替 AlexNet 中的较大卷积核 (11×11 , 7×7 , 5×5)。对于给定的感受野 (与输出有关的输入图片的局部大小), 采用堆积的小卷积核是优于采用大的卷积核, 因为多层非线性层可以增加网络深度来保证学习更复杂的模式, 而且代价还比较小 (参数更少)。
- **Receptive Field 可以只考虑某些 Channel**, 就比如也许有些 Pattern 只在红色的 Channel 会出现, 所所有是有这样的做法的。
- **Receptive Field 并不一定都是正方形的**, 这完全都是你自己设计的, 你是根据你对这个问题的理解来决定 Receptive Field 应该要长什么样子。

总而言之, Receptive Field 是根据你对当前需要解决问题的理解来设计的, 是很灵活的。但虽然你可以任意设计, 还有一些比较经典的 Receptive Field 的设计方式。

3.1 经典的 Receptive Field 设计方式

3.1.1 看所有的 Channel

Each receptive field has a set of neurons (e.g., 64 neurons).



在图像识别的相关问题中，大部分情况下不会说是某个 Pattern 只出现某一个 Channel 里面，所以一般情况下我们会看全部的 Channel，那我们在描述一个 Receptive Field 的时候，**只要描述它的高跟宽就好了**，而这个高跟宽合起来就叫做 Kernel Size。

3.1.2 使用 3×3 的 Kernel

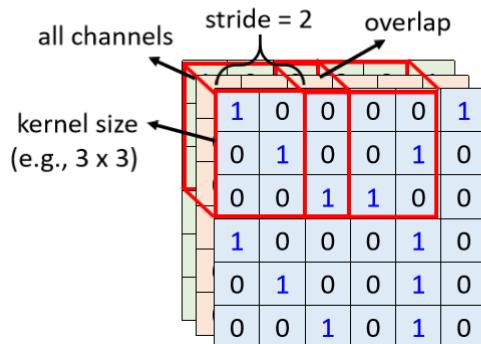
举例来说在这个例子里面，我们的 Kernel Size 就是 3×3 ，**一般我们在图像识别的问题中 3×3 的 Kernel Size 就足够了**，更大的卷积核其实也可以用连续的几个 3×3 的卷积核来替代。

一般同一个 Receptive Field，不会只有一个 Neuron 去关照它，往往会有组去侦测它，比如常用的会是 64 个或者是 128 个 Neuron 去侦测一个 Receptive Field 的范围。

3.1.3 使用较小的步长（一般为1或2）

到目前为止我们讲的都是一个 Receptive Field，那各个不同 Receptive Field 之间的关系是怎样的呢，我们可以把最左上角的这个 Receptive Field 往右移一点，这样就制造出了另外一个 Receptive Field，这个移动的量就叫做 **Stride**。

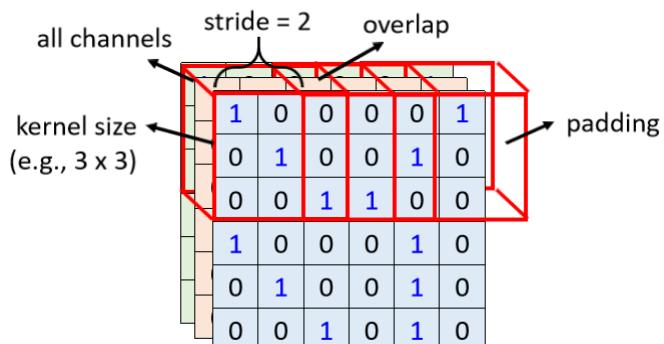
Each receptive field has a set of neurons (e.g., 64 neurons).



在这个例子中 Stride 就等于 2，**Stride 也是一个你自己决定的 Hyperparameter**，但这个 Stride 往往不会设太大，一般设 1 或 2 就可以了。这是因为我们**希望这些 Receptive Field 之间是有重叠的**，因为假设 Receptive Field 之间完全没有重叠，并且有一个 Pattern 恰好出现在两个 Receptive Field 的交界上面，那就会变成没有任何 Neuron 去侦测它，这样就很可能错过对这个 Pattern 的侦测，所以我们希望 Receptive Field 之间是有重叠的。

当然你可能会注意到另一个问题，在之前的例子中我们的 Stride 就等于 2，假设最左上角的 Receptive Field 向左走了一步到中间的 Receptive Field，那中间的这个要是再往右走一步，我们会发现它**超出了图像的范围**，这个时候就需要我们做 **Padding**。

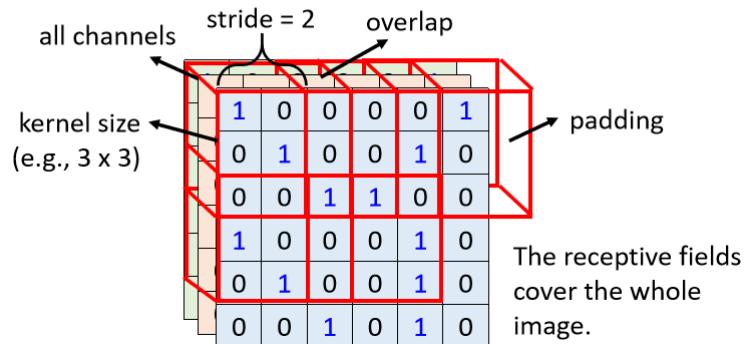
Each receptive field has a set of neurons (e.g., 64 neurons).



Padding 就是补值的意思，超出范围的值全部当做0来处理就是其中一种补值的方式，当然也有别的补值的方法，比如说补整张图片里面所有 Value 的平均，或者用图像边界上的值来填充，这些都可以，还是需要你根据对问题的理解自己决定。

当然啦，除了水平方向移动，也会有垂直方向的移动，这样才能保证我们的 Receptive Field 能够照顾到整张图片。

Each receptive field has a set of neurons (e.g., 64 neurons).



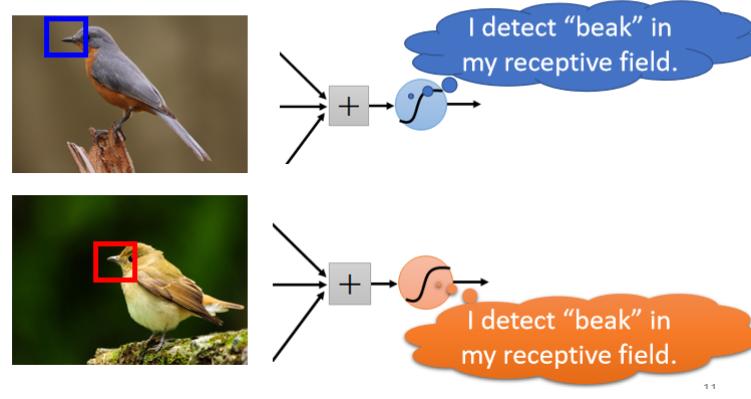
4 观察二：同样的 Pattern 可能会出现在图片的不同区域

- The same patterns appear in different regions.



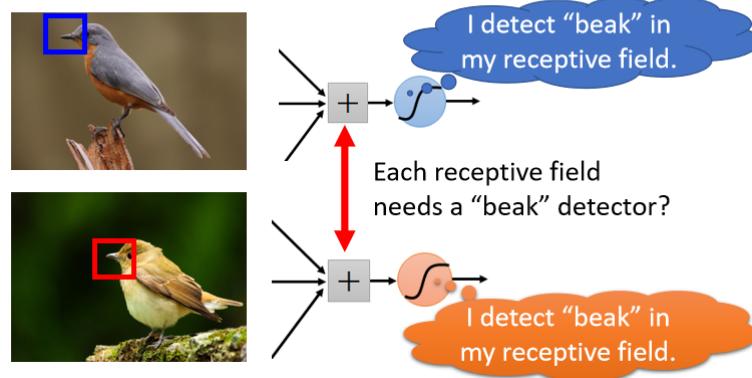
按照我们刚才的讨论，同样的 Pattern 出现在图片的不同的位置似乎也不是太大的问题，因为我们的 Receptive Field 在移动完之后是会照顾到整个图片的，所以不同位置的 Receptive Field 中会有不同的 Neuron 来负责侦测当前位置的 Pattern。

- The same patterns appear in different regions.



但这里的问题是，这些侦测鸟嘴的 Neuron 做的事情其实是一样的，只是它们侦测的范围不一样，**我们真的需要每一个 Receptive Field，都去放一个侦测鸟嘴的 Neuron 吗？这样参数的量会不会太多了呢？**

- The same patterns appear in different regions.

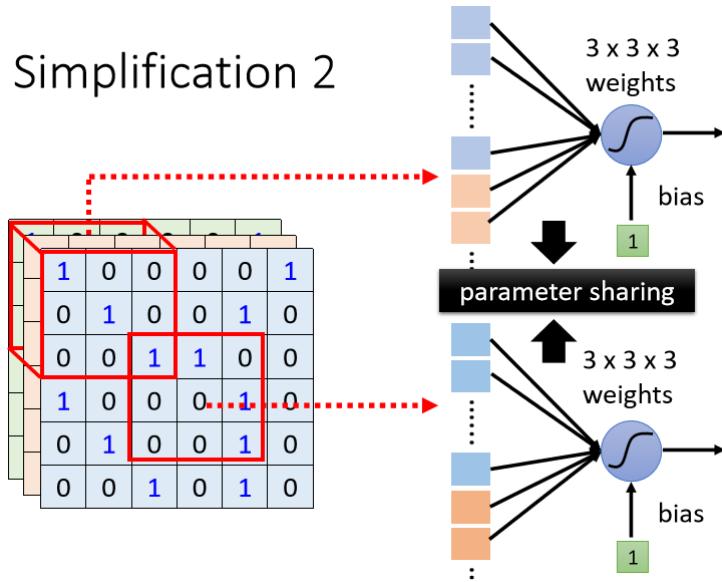


基于对这个问题的考虑，我们提出接下来的简化方法。

5 简化二：权值共享

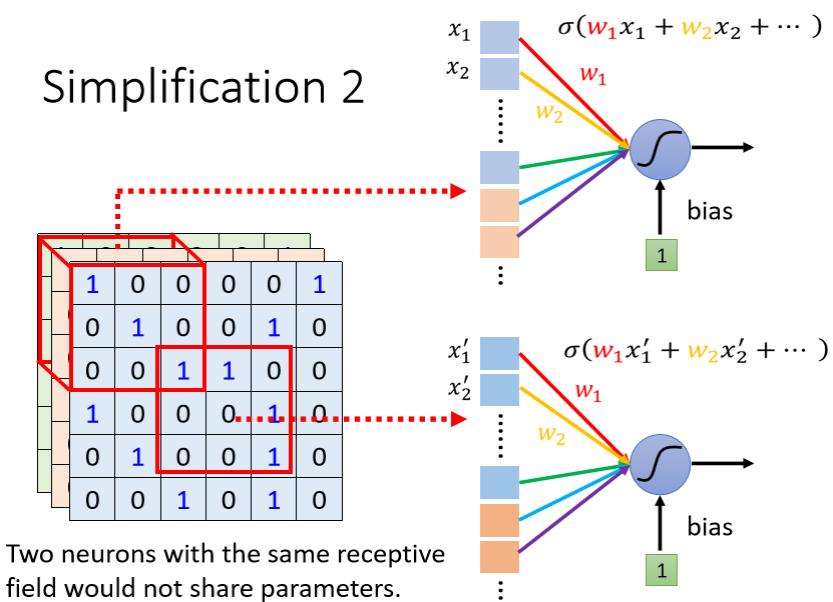
我们能不能让不同 Receptive Field 的 Neuron 共享参数，来减少模型整体的参数数量呢？

Simplification 2



所谓共享参数，就是让两个 Neuron 的 weights 完全是一样的，图中特别用相同的颜色来标识相同的权值。

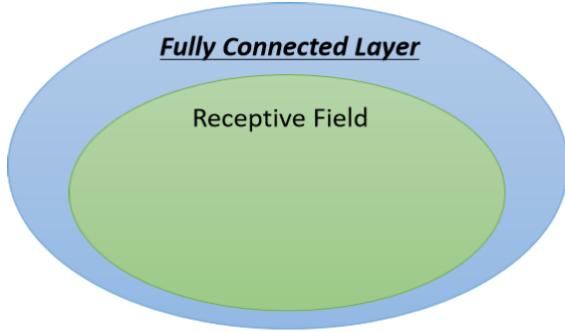
Simplification 2



上面这个 Neuron 跟下面这个 Neuron，它们的 Receptive Field 不一样，但是它们的参数是一模一样的。由于这两个 Neuron 的输入不一样，所有尽管它们有着完全相同的参数，输出也会是不一样的。通过这种方式，我们就进一步简化了我们的模型，降低了模型总体的参数量。

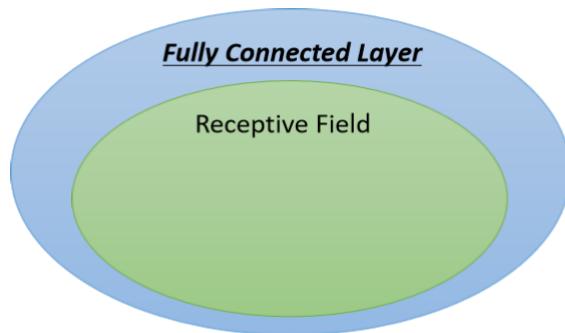
6 使用卷积层的好处

目前已经讲了两个简化的方法，现在我们来整理一下我们学到了什么，下图中最外层是 Fully Connected 的 Network：



- Some patterns are much smaller than the whole image.

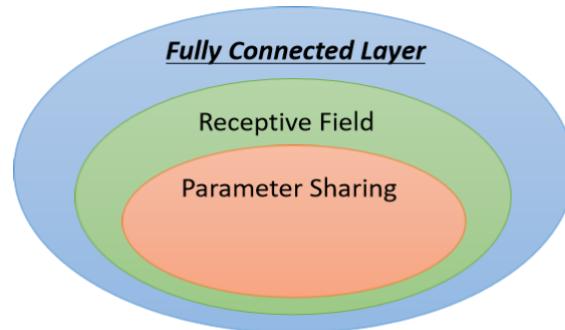
它是**弹性最大的**，但我们有时候不需要看整张图片，也许**只要看图片的一小部分就可以侦测出重要的 Pattern**，于是我们有了 **Receptive Field** 的概念：



- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

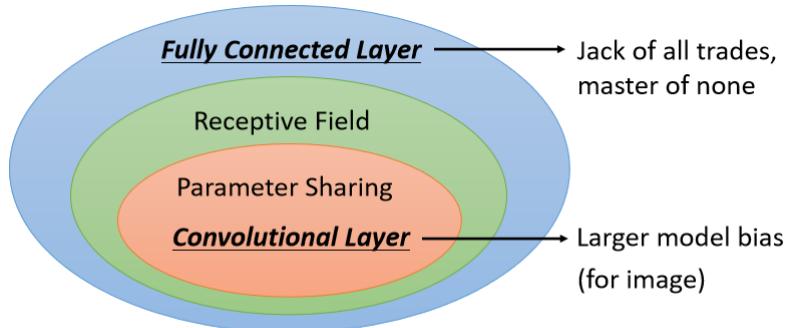
当我们强制一个 Neuron 只能看一张图片里面的一个范围的时候，它的**弹性是变小的**，因为如果是 Fully Connected 的 Network，它可以决定是看整张图片还是只看一个范围，如果它只想看一个范围，就把很多 Weight 设成 0 就好，所以加入 Receptive Field 以后你的 Network 的弹性是变小的。

接下来我们还有权值共享，权值共享又更进一步限制了 Network 的弹性：



- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

而 **Receptive Field 加上 Parameter Sharing**，就是 **Convolutional Layer**：

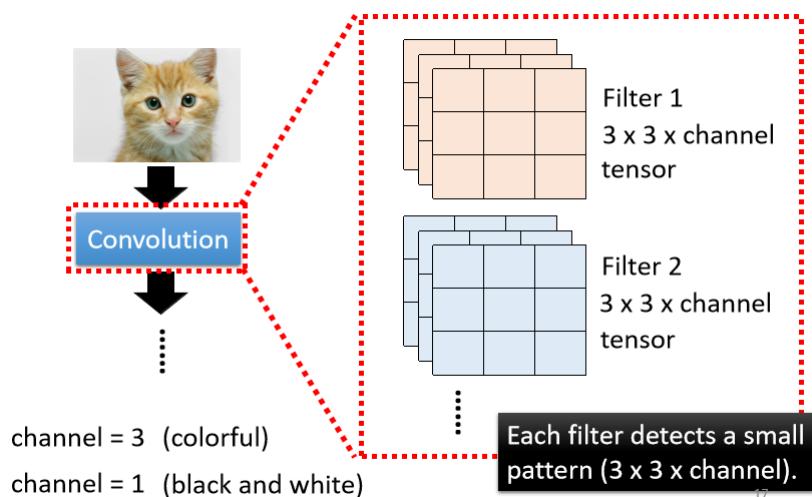


- Some patterns are much smaller than the whole image.
- The same patterns appear in different regions.

有用到 Convolutional Layer 的 Network，就叫 Convolutional Neural Network，就是 **CNN**，所以我们可以看出 **CNN 的弹性比较小，但这其实正是 CNN 在图像处理问题上能做的比较好的原因**。因为我们的 CNN 是专为图像处理设计的，**Fully Connected Network 可以做各式各样的事情，但他的弹性比较大，比较容易 Overfitting**，所以在图像处理这种特定的问题上，往往没有 CNN 做的好。

7 CNN 的另一种介绍方式

假设介绍 CNN 的第一个版本听不太明白的话，我们来听第二个版本，第二个版本是这样的，**Convolutional Layer 其实就是里面有很多的 Filter 来筛选特定的模式**。



假设我们这些 Filter 的大小都是 $3 \times 3 \times \text{Channel}$ ，一个 **Convolutional Layer** 里面就是有一组不同的 **Filter**，每一个 Filter 都是一个 $3 \times 3 \times \text{Channel}$ 大小的 Tensor。

每一个 Filter 的作用就是要去图片里面识别某一个 Pattern，那这些 Filter 具体是怎么去图片里面识别 Pattern 的呢，我们现在举一个实际的例子：

Convolutional Layer

Consider channel = 1
(black and white image)

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮

(The values in the filters
are unknown parameters.)

这个例子里面我们假设 Channel 是 1，也就是说我们图片是**黑白的图片**。我们假设这些 Filter 的参数是已知的，当然实际上这些 Filter 里面的数值其实也是 Model 里面的 Parameter，它们**其实是未知的，是要通过 gradient decent 来训练出来的**。

那我们现在假设这些 Filter 里面的数值已经找出来了，我们来看看这些 Filter 是怎么去图片里面识别 Pattern 的。

Convolutional Layer

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

3

6 x 6 image

假设我们要识别的图片是 6×6 的大小，那我们先把 Filter 盖在图片的左上角，然后把 Filter 里面所有的值，跟图片左上角这个范围内对应的值做相乘，再把乘出来的 9 个值相加：

老师用的 filter 为 $\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$ 与图片的第一个 3×3 矩阵 $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 做卷积运算的过程为

$$(1 \times 1) + (-1 \times 0) + (-1 \times 0) + (-1 \times 0) + (1 \times 1) + (-1 \times 0) + (-1 \times 0) + (-1 \times 0) + (1 \times 1) = 3$$

在下图中 filter 为 $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ 大家可以自行验证

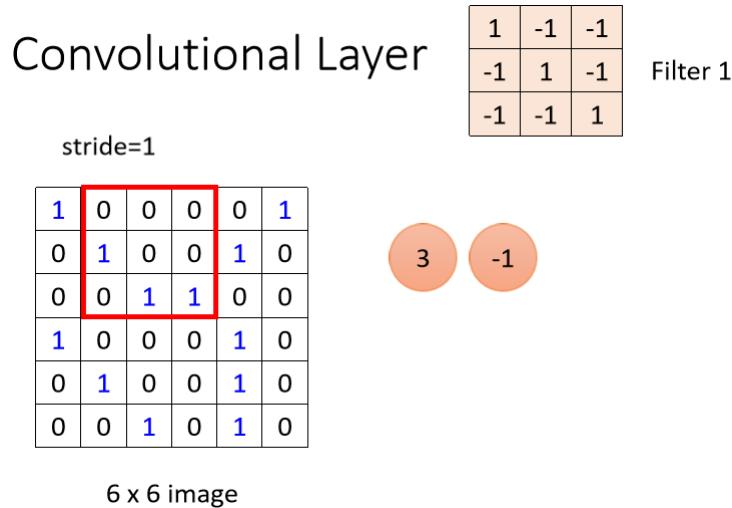
1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

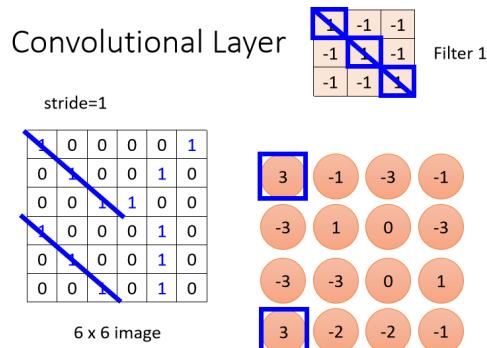
4		

Convolved Feature

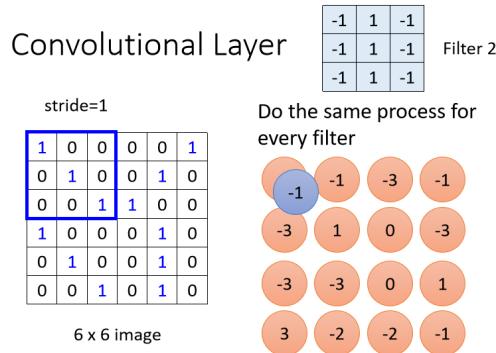
最开始的 Filter 放在左上角，算完一个区域后，接下来就往右移一点，这个移动的距离叫做 **Stride**



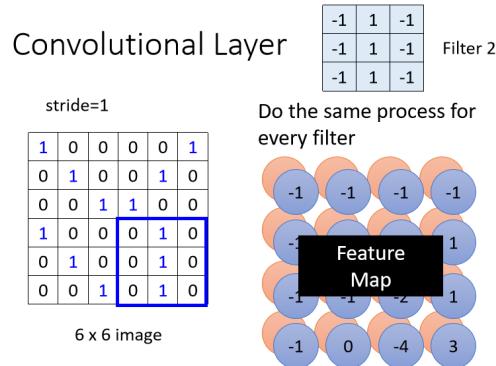
那为什么说这个 Filter 在侦测图片中的 Pattern 呢？可以看到我们现在的 Filter 1，它**对角线的地方都是1**，而其他地方的值都是-1，所以只有我们的图片在对角线也全为1的时候，Filter 与这个区域算出来的乘积最大。



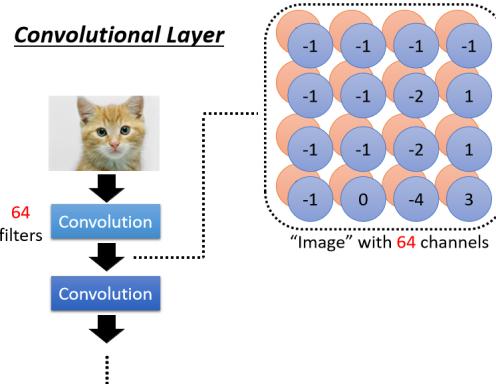
所以你会发现左上角和左下角算出来的值都是3，这就说明原来图片的左上角和左下角都包含了一个对角线的图案。当然除了侦测对角线的，我们还有 Filter 2是侦测竖直线的：



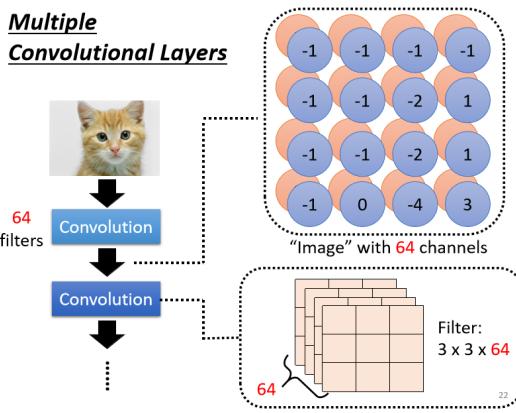
我们就把第二个 Filter 从左上角开始扫起，得到一个数值，往右移一点，再得到一个数值，重复同样的操作直到把整张图片都扫完，我们就得到另外一组数值。所以每一个 Filter 扫完图片都会给我们产生一个新的矩阵，如果我们有 64 个 Filter，我们就得到 64 个矩阵，这 64 个矩阵合在一起，叫做 **Feature Map**。



所以当我们把一张图片通过一个 Convolutional Layer，里面有一堆 Filter 的时候，我们产生了一个 Feature Map，这个 **Feature Map** 其实可以看做一张新的图片，它提取并浓缩了原图像的某些特征。



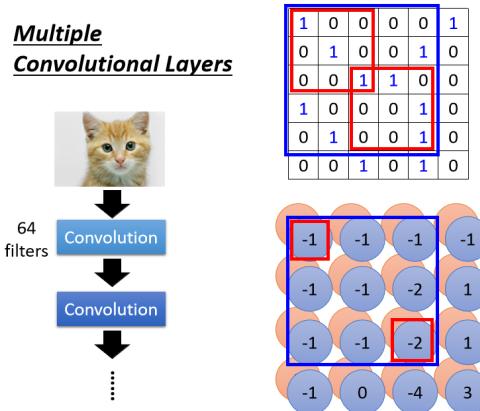
这个新图片的每一个 Channel 就对应到一个原来的 Filter，由于我们有 64 个 Filter，所以新的图像就有 64 个 Channel。当然这个 **Convolutional Layer** 是可以叠很多层的，假设我们现在想要加上第二层。



第二层里肯定也有一堆的 Filter，第二层里每一个 Filter 的大小，我们可以仍然用 3×3 ，但它的高度必须是 64，因为 Filter 的这个高度就是它要处理的图像的 Channel 数，因为之前我们也提到了在经典的设计方法中，我们的 Filter 是看图像所有的 Channel 的。

这里你可能会有一个疑问，如果我们的 Filter 的大小一直设 3×3 ，会不会让我们的 Network 没有办法看比较大范围的 Pattern 呢？

实际上这是不会的，这里就拿我们的第二层 Convolutional Layer 为例，我们的 Filter 的大小一样设 3×3 ，当我们从原始图片产生的新的 Feature Map 的左上角 3×3 的范围时，我们在原来的影像上其实是侦测了一个 5×5 的范围。所以只要你的 Network 够深，你就不用怕侦测不到比较大的 Pattern。



现在来比较一下我们解释 CNN 的这两个版本的故事，**实际上他们的本质是一模一样的**。这里简单的展示一下他们之间的一些对应关系：

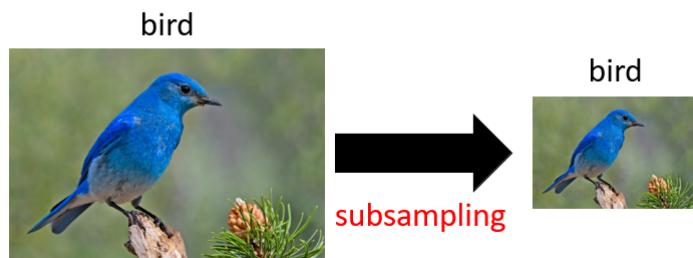
Convolutional Layer

<i>Neuron Version Story</i>	<i>Filter Version Story</i>
Each neuron only considers a receptive field.	There are a set of filters detecting small patterns.
The neurons with different receptive fields share the parameters.	Each filter convolves over the input image.

They are the same story.

8 观察三：通过池化操作压缩图像大小

- Subsampling the pixels will not change the object

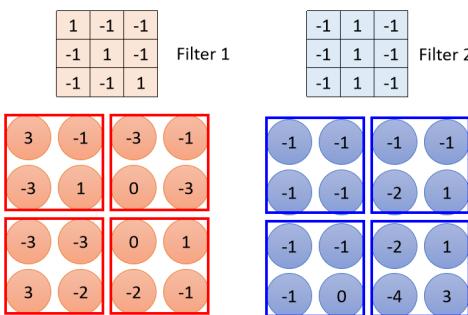


Pooling, 也就是所谓的池化究竟是什么呢?

这其实来源于图像信息的另一个特殊性，图像损失某些细节信息可能并不影响我们对图像内容的辨认，这种抛弃某些细节信息来压缩图像的方式我们叫做下采样 (Subsampling)，也就是刚刚提到的池化操作。举例来说，你把上面这张鸟的图片中偶数的 Column 都拿掉，奇数的 Row 都拿掉，图片大小会变为原来的1/4，但是这基本不会影响我们对图像内容的辨认，可以看到缩小后的图片里面还是一只鸟。

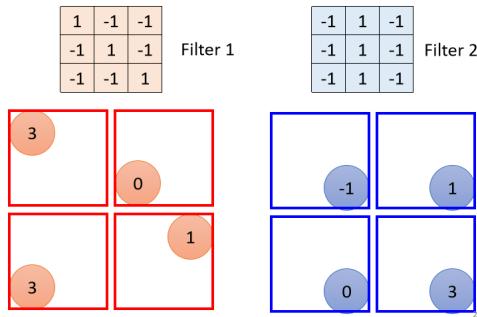
而 Pooling 本身也有很多不同的版本，这边以 Max Pooling 为例。

Pooling – Max Pooling



刚刚提到每一个 Filter 扫描完原始图片后都会产生一个新的矩阵，如果这个时候我们进行 Max Pooling 操作，在这个例子中就是把这个新的矩阵中的数值分为 2×2 个一组，每一组里面选一个代表，在 Max Pooling 的情况下，我们选的代表就是最大的那个，可以想见如果使用 Min Pooling 的话我们选的代表就是最小的那个。当然也可以用平均值来作为代表，这个是由你自己决定的。

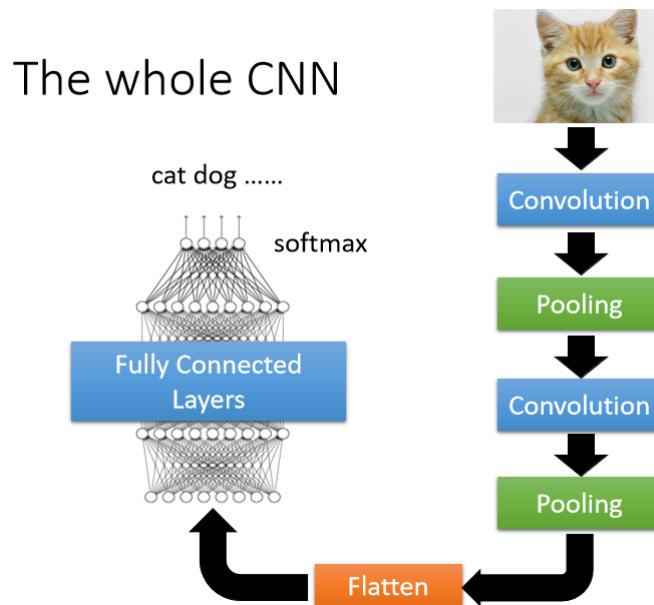
Pooling – Max Pooling



Pooling 做的事情就是把图片变小，不过可以想到 Pooling 操作对于你的图像识别可能会带来一些负面影响，特别是如果图像的某些细节信息对图像识别的影响很大时，过多的 Pooling 可能会使得你的识别率大大下降。

起初使用 Pooling 最主要的理由是为了减少运算量，而由于近年来运算能力越来越强，很多 CNN 的架构设计中会直接抛弃掉 Pooling，期望保留图像的细节信息。

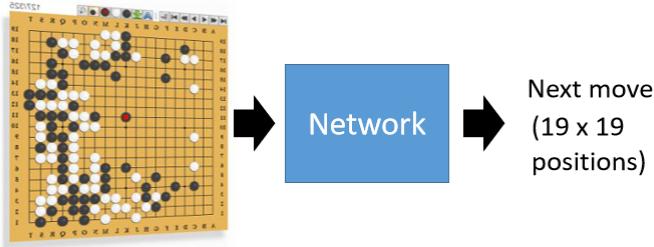
9 总结：CNN 的一般架构



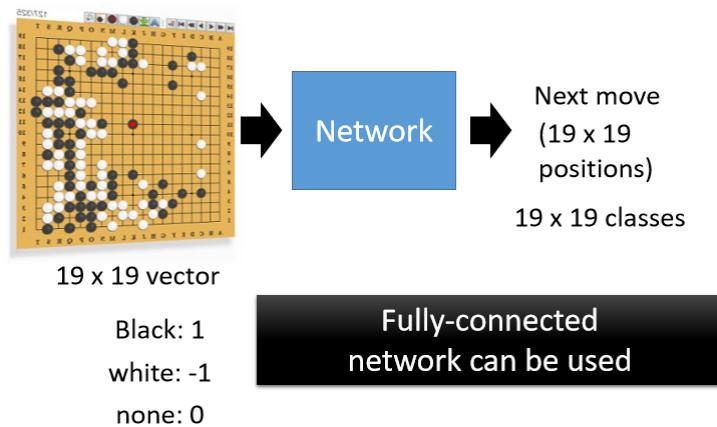
一般 CNN 的架构就是 Convolution 加 Pooling，可以做很多层，当然刚刚也说过现在 Pooling 也是可以丢掉的了，最后把 Pooling 的 Output 进行 Flatten 操作。Flatten 的意思就是把矩阵拉直成一个一维向量，再把这个向量作为 Fully Connected Layers 的输入，如果是图像分类问题，这就回到了我们最初讲的 Fully Connected Network 处理分类问题的过程，Fully Connected Layers 的输出通过一个 Softmax 层，最后就得到图像分类的结果。

10 应用： AlphaGo

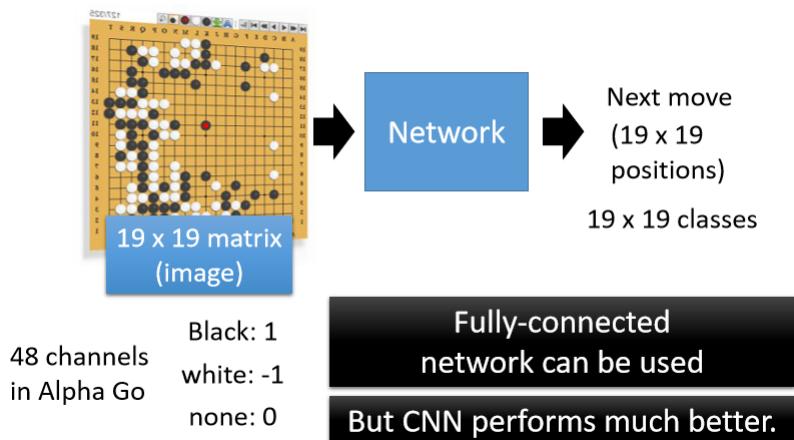
那除了处理图像分类问题之外，你可能听过 CNN 另外一个最耳熟能详的应用，就是用来下围棋，那怎么用这个 CNN 来下围棋呢？



那下围棋其实就是一个分类的问题，你的 Network 的输入，是棋盘上黑子跟白子的位置，你的输出就是下一步应该要落子的位置。棋盘本身能就可以看成是一张 19×19 的图片，如果棋盘某一个位置有一个黑子，那个位置我们就填 1，白子就填 -1，没有子我们就填 0。



这样我们就可以用一个 19×19 维的向量来描述一个棋盘，把这个向量输到一个 Network 里面，然后你就可以把下围棋当作一个有 **19 × 19 个类别的分类问题**，这个 Network 的输入是当前的棋局，输出就是**下一步应该落子的位置**。



一般图片的 Channel 就是 RGB，也就是三个 Channel，那表示棋盘的图片上每一个 Pixel 的 Channel 应该是什么呢？在 **AlphaGo 的原始论文里面**，**每一个棋盘上的 Pixel 是用 48 个 Channel 来描述**，也就是说棋盘上的每一个位置，它都用 48 个数字，来描述那个位置当前的状态。

那至于为什么是 48 个，这个显然是围棋高手设计出来的，所以现在的棋盘它就是一张 19×19 的图片，它的 Channel 为 48。

但是为什么 CNN 可以用在下围棋上呢，我们之前就提到过 **CNN 是专为图像处理设计的**，所以如果一个问题跟影像没有什么共通的特性的话，你其实不应该用 CNN。那这里既然使用了 CNN 来下围棋，我们就来考虑一下它们之间有哪些共通的特性。

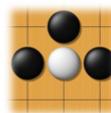
10.1 下围棋和图像的共通特性

我们之前提到在图像上的第一个观察是，**很多重要的 Pattern 我们只需要看小范围就知道**，下围棋其实也是一样的。

Why CNN for Go playing?

- Some patterns are much smaller than the whole image

Alpha Go uses 5×5 for first layer

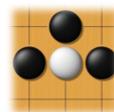


举例来说上图的 Pattern，你就算**不用看整个棋盘的盘势，你都知道这边发生了什么事**，很明显这个模式表明这个白子快被黑子围住了。在 AlphaGo 里面，它的第一个卷积层的 Filter 的大小就是 5×5 ，所以显然是设计这个 Network 的人觉得，**棋盘上很多重要的 Pattern 也许看 5×5 的范围就可以知道了**。

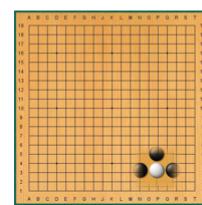
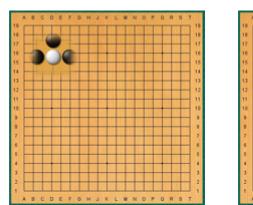
我们再来看之前提到在图像上的第二个观察是，**同样的 Pattern 可能会出现在不同的位置**，在下围棋里面也是一样的。

- Some patterns are much smaller than the whole image

Alpha Go uses 5×5 for first layer



- The same patterns appear in different regions.



但是当我们看之前提到在图像上的第三个观察时，这里就出现了一点小问题。第三个观察是说图像的 Pooling 操作可以在不影响对图像内容识别的情况下减小图像大小，但是很明显在棋盘中我们不能这样做，如果把棋盘上的奇数行跟偶数列拿掉，早就不是同一个棋局了。换句话说，**因为棋盘上的每一个像素点代表了一个棋子，所有每一个像素点都很重要，其实是不能够做 Pooling 操作的**。从 AlphaGo 的论文原文中对其使用的 Network 架构描述中，我们也可以看到它根本没有使用 Pooling。

- Subsampling the pixels will not change the object



Pooling

How to explain this???

Neural network architecture. The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 2c show that Alpha Go does not use Pooling | 256 and 384 filters

33

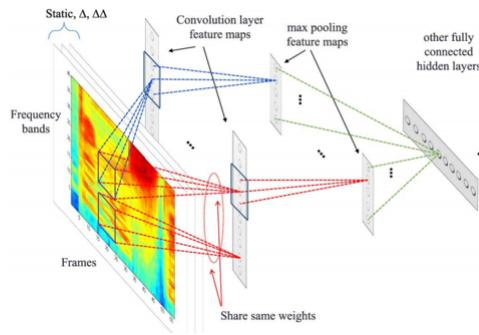
10.2 CNN 的更多应用领域

而那 CNN 除了被用来处理图像和下围棋之外，近年来也用在语音处理和文字处理上。

More Applications

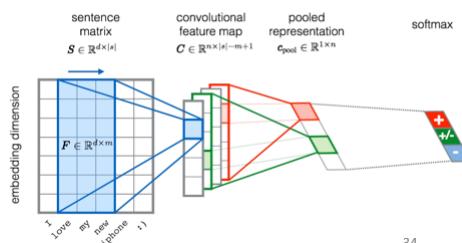
Speech

<https://dl.acm.org/doi/10.1109/TASLP.2014.2339736>



Natural Language Processing

<https://www.aclweb.org/anthology/S15-2079/>



34

但不要以为用在图像处理上的 CNN 架构直接套到语音上就 OK 了，如果直接生搬硬套很可能是不 Work 的，还是刚刚在讨论为什么 CNN 可以用来下围棋时提到的，你要想清楚语音处理、文字处理与图像有什么共通的特性，对于那些不共通的性质，就需要你进行一些特殊的设计来处理。

11 CNN 的局限性

CNN 在图像处理上也不是完全无敌的，它其实没有办法处理影像放大缩小或者是旋转的问题。举例来说，假设今天你给 CNN 看的狗都是上面那张图的大小，训练之后它可以辨认说这是一条狗，但当你把这个图片放大后输入给他之后，它可能就认不出来这是一条狗了。

- CNN is not invariant to scaling and rotation (we need data augmentation 😊).



Spatial Transformer Layer



<https://youtu.be/S0CywZ1hZak>
(in Mandarin)

因为对它来说这两张图片虽然狗的形状是一模一样的，但是 CNN 只能看到数值，如果你把它拉长成向量的话，它里面的数值就有了很大的差异，所以虽然你人眼一看觉得它们形状很像，但对 CNN 的 Network 来说这两张图片是非常不一样的。

当然其中的一个解决方法就是做 Data Augmentation，所谓的 Data Augmentation 就是修改现有的训练资料来进一步扩充你的训练资料，就比如你把训练资料中每张图片都一小块出来放大，或者把图片旋转，使用扩充后的资料来训练 CNN，这样就可以得到比较好的结果。

当然这里有一个架构叫 **Special Transformer Layer** 是可以处理 Scaling 和 Rotation 的问题的，这里放上一个相关视频的二维码，感兴趣的话可以进一步学习。