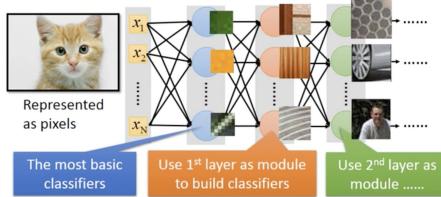


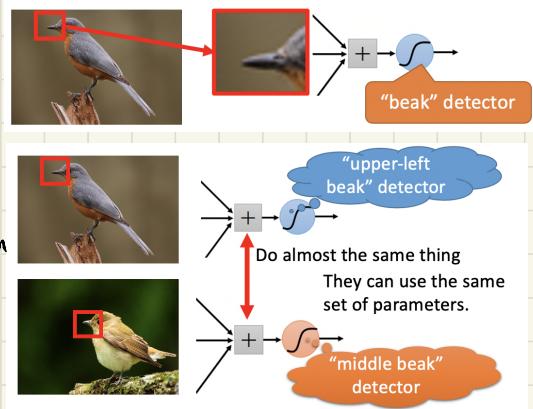
# Chapter 7 Convolutional Neural Network

## 1. CNN的起源

- 将深度神经网络应用在图像分类过程中，假设图像为 $100 \times 100$ 像素，那么 Input Layer 就是  $1 \times 30000$  的 image vector，假设使用全连接神经网络，Hidden Layer 的 Neuron 数量为 1000 个，则一层连接就有  $30000 \times 1000$  个神经元，整个神经网络的权重数量过于庞大。



- 对于图像而言，一些有利识别的 pattern 可能要远小于整个图片的大小。因此神经元不需要观察整个图片，只观察 pattern 所在的像素区（鸟嘴/爪子/尾巴）即可。



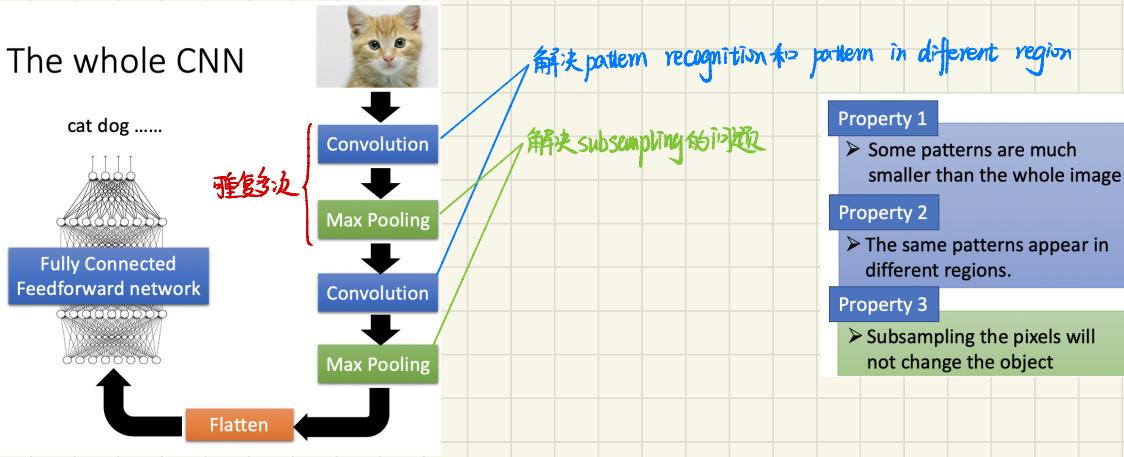
- 利用 Subsampling 抽取图像的奇数行奇数列对图像进行抽样，也可以有效的减少神经网络的参数量。



## 2. CNN的整体结构

CNN由 Input、Convolution + Max Pooling (可重复多次)、Flatten、Fully Connected Feedforward Network 构成

# The whole CNN



## ① Convolution

- 对于 $6 \times 6$ 的图像，有很多Filter(以Matrix形式存在，以 $3 \times 3$ 大小为例)。
- 每个Filter相当于Fully Connected Network中的一个Neuron，Filter中的数字是网络中的参数，由网络自动学习出
- $3 \times 3$ 大小的Filter可以检测 $3 \times 3$ 大小的pattern

Those are the network parameters to be learned.

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| 1 | 0 | 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 |   |

$6 \times 6$  image

|    |   |    |
|----|---|----|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

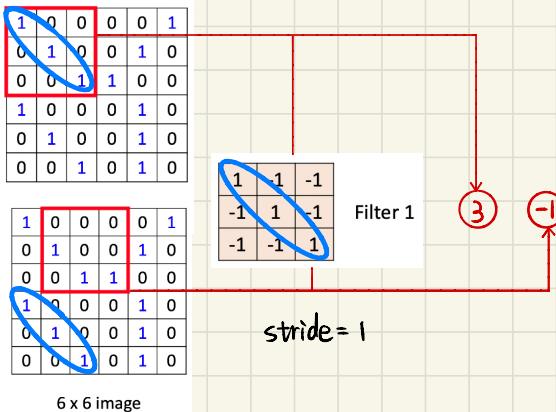
Filter 1  
Matrix

|    |   |    |
|----|---|----|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2  
Matrix

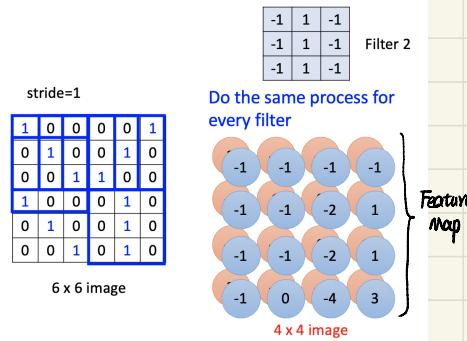
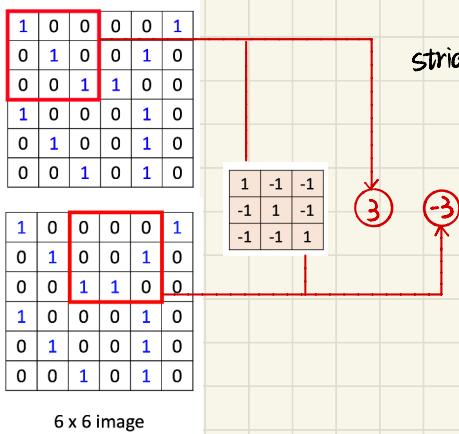
Property 1 Each filter detects a small pattern ( $3 \times 3$ ).

- 使用image matrix to filter matrix进行卷积操作，其中 stride 表示红框每次移动的步长



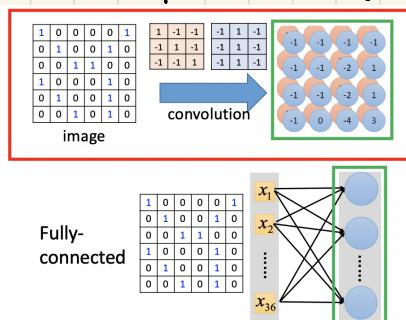
经过卷积将 $6 \times 6$ 的图像压缩至 $4 \times 4$ 的矩阵，其中  
Filter 1 的作用是检测是否存在左上右下为1的pattern

|    |    |    |    |
|----|----|----|----|
| 3  | -1 | -3 | -1 |
| -3 | 1  | 0  | -3 |
| -3 | -3 | 0  | 1  |
| 3  | -2 | -2 | -1 |

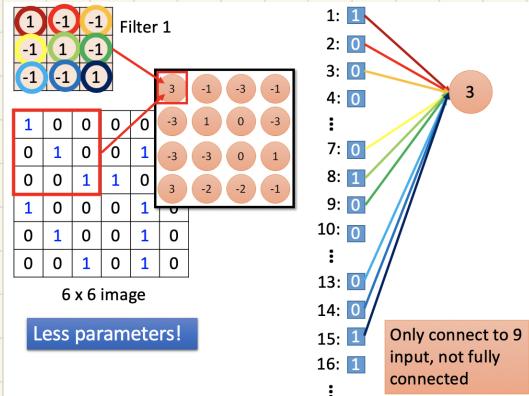


- 有很多 Filter 时，图像经过和每个 Filter 做卷积操作后会得到一个压缩后的 image，所有的压缩 image 合在一起称为 Feature Map。
- 当图像为彩色 (RGB, 3个 channel)，Image Matrix 和 Filter Matrix 是三层的 Matrix 叠加起来的

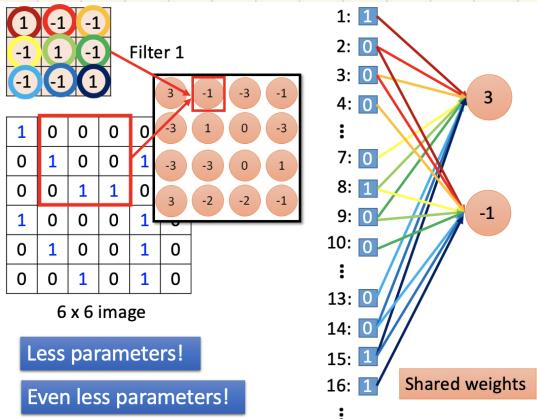
- Convolution 与 Fully Connected 的关系是：Convolution 相当于将 Fully Connected Network 的一些权重丢弃。一次卷积操作生成的 Feature Map 对于一层 Hidden Layer 的输出



卷积操作相当于将图片矩阵展开成向量后与一个 Neuron 相连，但只连接了第 1/2/3, 7/8/9, 13/14/15 共 9 个输入，其它输入不连接。这样与全连接相比可以使用更少的参数



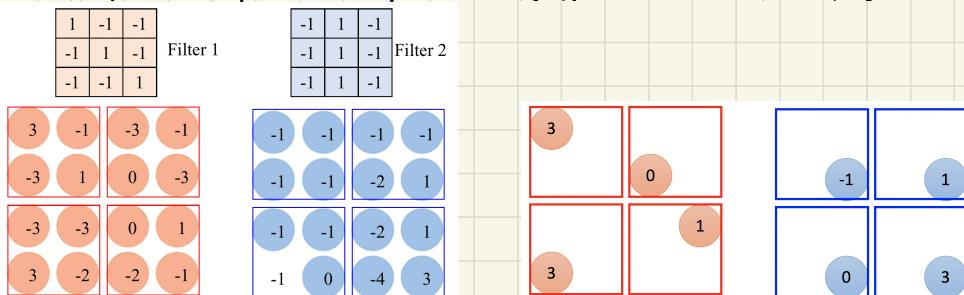
- 相比全连接网络，卷积可以降低参数量。除此之外因为③与 input 1 的连接权重等于①与 input 2 的连接权重



以此类推，巧妙的通过 shared weights 达到了更少的参数量

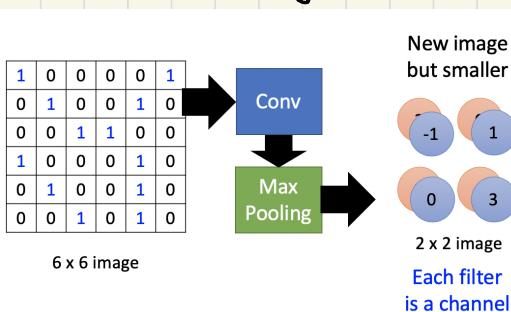
## ② Max Pooling

- Max Pooling 相当于将每个 Feature Map 划分成一定的区域，每个区域取域内取最大值代表区域，这样可以进一步压缩图像大小。其中取最大值不可微分的问题可以参照 Maxout 激活函数的解决方法。

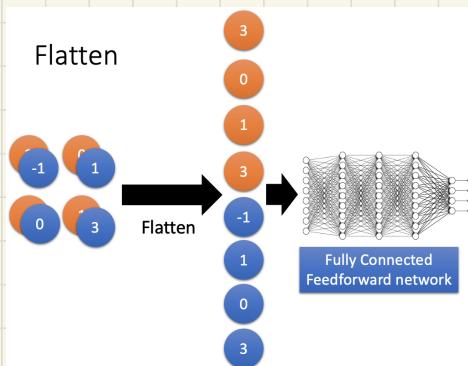


## ③ CNN 整体结构

- 经过 Convolution 和 Max Pooling 操作，可以最大限度的压缩图片。以上述为例，6x6 的 Image 被压缩为 2x2

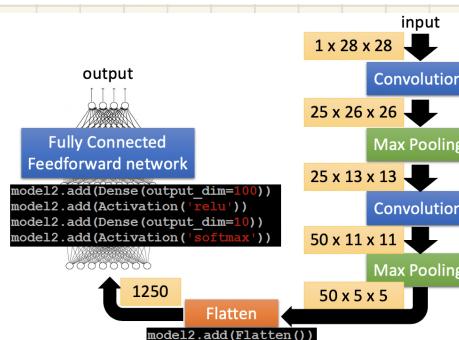
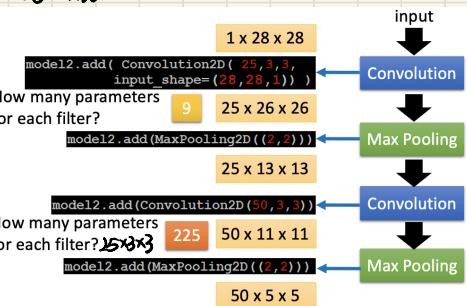
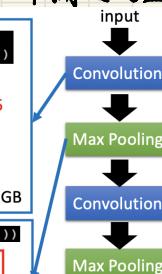
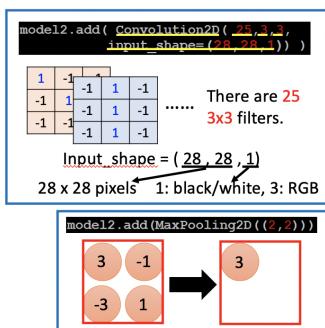


- Flatten指将最后得到的image还原成向量后作为输入送进全连接神经网络



#### ④ CNN in keras

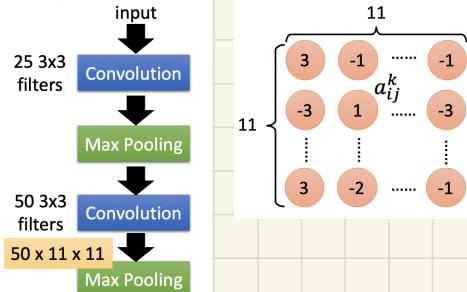
- 在DNN中输入一个向量，在CNN中输入要调整为一个3-D的tensor



### 3. CNN的特征学习过程

#### ①多卷积层中Filter的特征学习能力

- 经过Filter卷积后，网络学到的特征如下所示。第一个卷积层学到的特征可以轻松的从Filter Matrix中看出，第二个卷积层学到的东西变得更多更复杂，因为其输入是已经卷积池化过的图像。第二个卷积层第k个filter的输出为 $11 \times 11$ 的矩阵

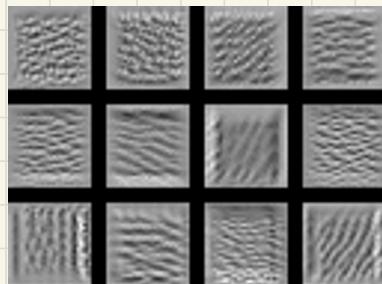


$a_{ij}^k$  表示第 k 个 filter 的输出的第 i 行第 j 列

定义第 k 个 filter 的 activation degree, 表示输入与其的匹配程度或第 k 个 filter 被激活的程度 :  $a^k = \sum_{i=1}^{11} \sum_{j=1}^{11} a_{ij}^k$

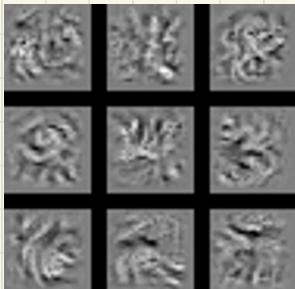
假设 input 为  $x$ , 寻找一个输入使得第 k 个 filter 被最大程度激活, 即  $x^* = \arg \max_x a^k$ ; 通过梯度下降即可求解  $x^*$

在以往的神经网络中, 都是输入固定, 通过梯度下降求解最优的权重; 而在 CNN 中, 通过梯度下降求解使  $k^{th}$  filter 激活程度最大的 input。假设第二个卷积层有 50 个 filter, 取出前 12 个 filter 对应的  $x^*$  如下所示。对应的输入均是各种各样的条纹状, 即该层的 filter 检测的是局部的条纹状 Pattern



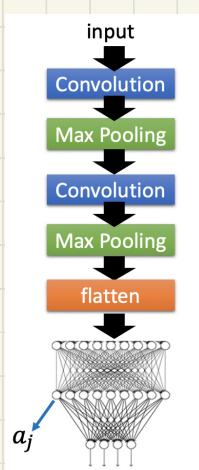
#### ②全连接神经网络的特征学习能力

类似于上一节, 求解使得神经元  $a_j$  的输出最大的 input, 即  $x^* = \arg \max_x a_j$

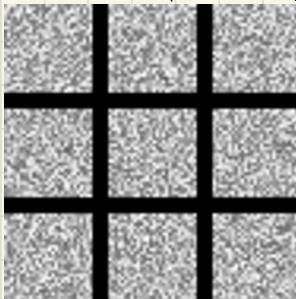


图中每一个图像对应于一个 Neuron 的  $x^*$ , 由图可知

其检测的是整个图片的特征



- 类似于上一节，求解使得输出层  $y_i$  的输出最大的 input，即  $x^* = \arg \max_x y_i$

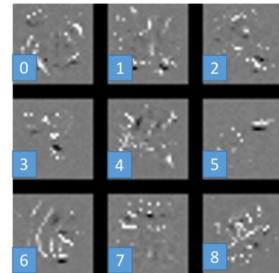


图中每一个图像对应于一个输出单元的  $x^*$ ，与实际的数字 0~9 不同



- 对于  $x^* = \arg \max_x y_i$  时，获得的是计算机认为的数字应该具有的样子，而不是人们可以分辨的样子，所以需要进行正则化。当  $x_{ij}$  为 1 时表示笔迹的黑色，对于手写数字来说大多数  $x_{ij}$  均为 0。引入正则化时： $x^* = \arg \max_x (y_i - \sum_{i,j} |x_{ij}|)$ 。这种正则化方法可以得到较为稀疏的图像，即要少一些。具体图像如下所示。图像略微出现了线条的样子，例如数字“6”。

$$x^* = \arg \max_x \left( y_i - \sum_{i,j} |x_{ij}| \right)$$



- Deep Dream：给定一张照片将CNN训练好以后，将网络中的某一层 hidden layer 或 filter 的数值进行放大，此时按照新的网络参数利用梯度下降找到使得该层输出最大的图像，两张图片的相差之处就是CNN所关注的 Pattern。即 “CNN exaggerates what it sees”

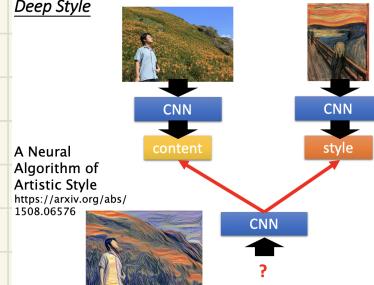


Deep Style：给定两张图片，将一张图片的风格转换为另一张 (A Neural Algorithm of Artistic Style)

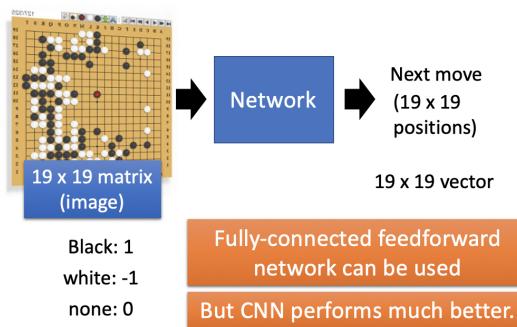


其实现原理为：训练两个CNN，一个关注filter的输出（即原图片的内容），另一个关注filter输出之间的相关度（即目标风格）  
然后同时使这两个均达到最大值的图像合成生成的图像

Deep Style



- 在围棋人机对战中，使用传统的DNN进行训练时需要将棋盘转换成一个列向量作为输入，而在使用CNN时，可以直接将棋盘表示为图像进行输入



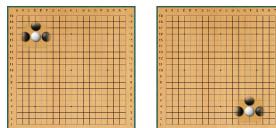
CNN可以取得较好效果的本质是问题具有与图像类似的特征，前两个Property是符合的（如左图），但subsampling会丢失一些棋盘和棋子，这种情况下效果不提升的原因是：AlphaGo没有用Max Pooling

- Some patterns are much smaller than the whole image

Alpha Go uses 5 x 5 for first layer



- The same patterns appear in different regions.

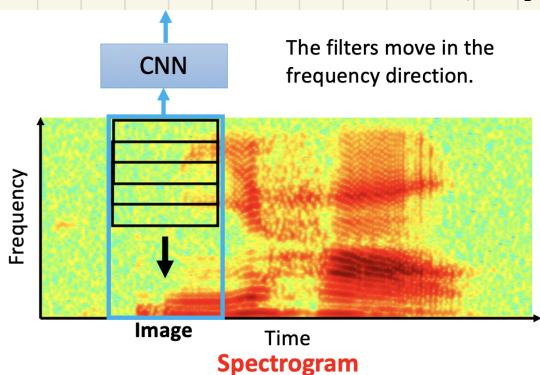


- Subsampling the pixels will not change the object

→ Max 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  $23 \times 23$  image, then convolves  $k$  filters of kernel size  $5 \times 5$  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  $21 \times 21$  image, then convolves  $k$  filters of kernel size  $3 \times 3$  with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size  $1 \times 1$  with stride 1 with a different bias for each position, and applies a softmax function. The Alpha Go does not use Max Pooling ..... Extended Data Table 3 additionally show the results of training with  $k = 128, 256$  and 384 filters.

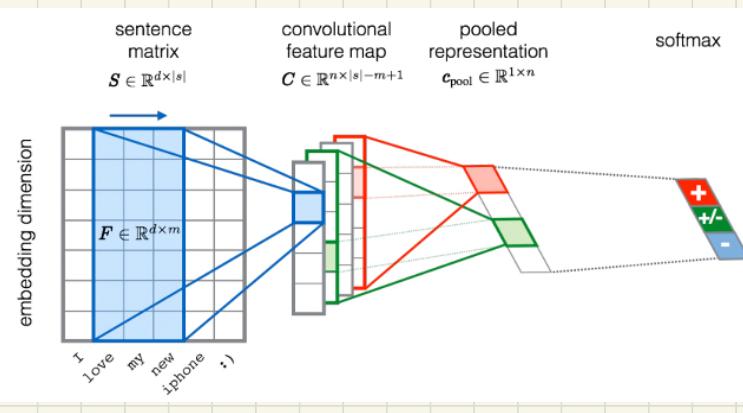
- CNN在语音领域的应用：将语音转化成频谱 (spectrogram) 送入 CNN 训练，最后网络可以通过频谱生成语音



- Filter是在频率的方向上移动的，例如男生和女性的“你好”表现的 pattern 相同，频率不同

- Filter 在时间轴上移动效果不会提升，因为时间的关系主要靠后续的 RNN, LSTM 完成

- CNN在文本领域的应用：利用 Word Embedding 将 Sequence 表示为 sentence matrix，送入 CNN。其中 Filter



embedding dimension 同样大，

其只在 time 方向上移动 (因为 word

embedding 后不同的 dimension 之

间独立，没有关系，不会出现

pattern in different region)

- Machine Drawing Image: PixelRNN, VAE, GAN