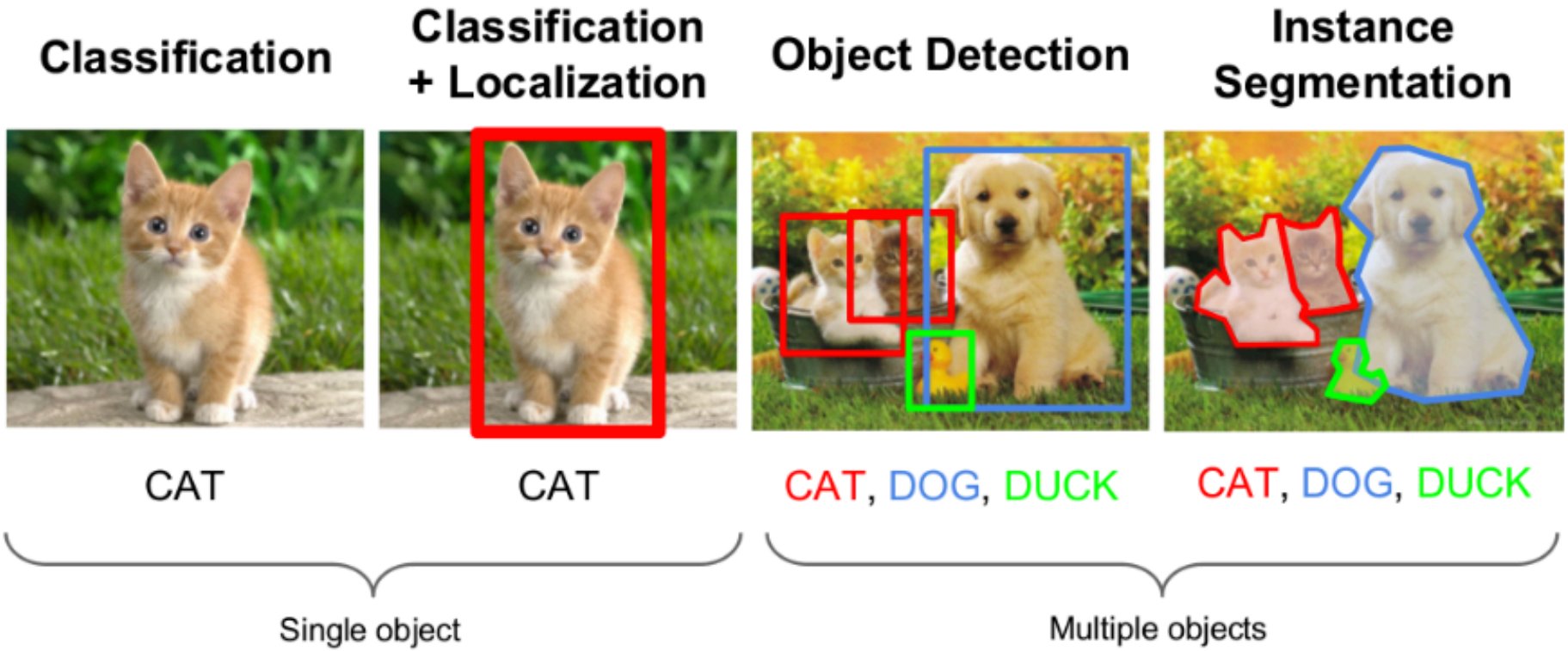
五、卷积神经网络

CNN: Convolutional Neural Networks

计算机视觉

CS231n: Convolutional Neural Networks for Visual Recognition
Standford
Fei-Fei Li



计算机视觉

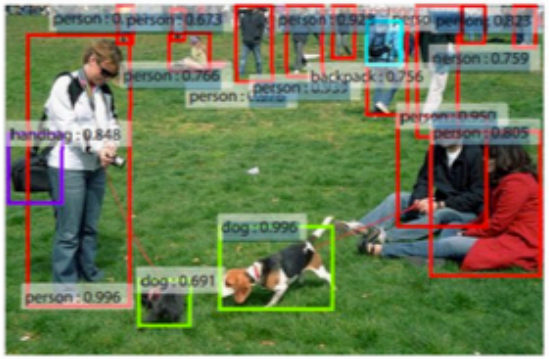
From image to text.

- Captions generated by a recurrent neural network (RNN) taking, as extra input, the representation extracted by a deep convolution neural network (CNN) from a test image, with the RNN trained to 'translate' high-level representations of images into captions.
- When the RNN is given the ability to focus its attention on a different location in the input image (the lighter patches were given more attention) as it generates each word (bold) .

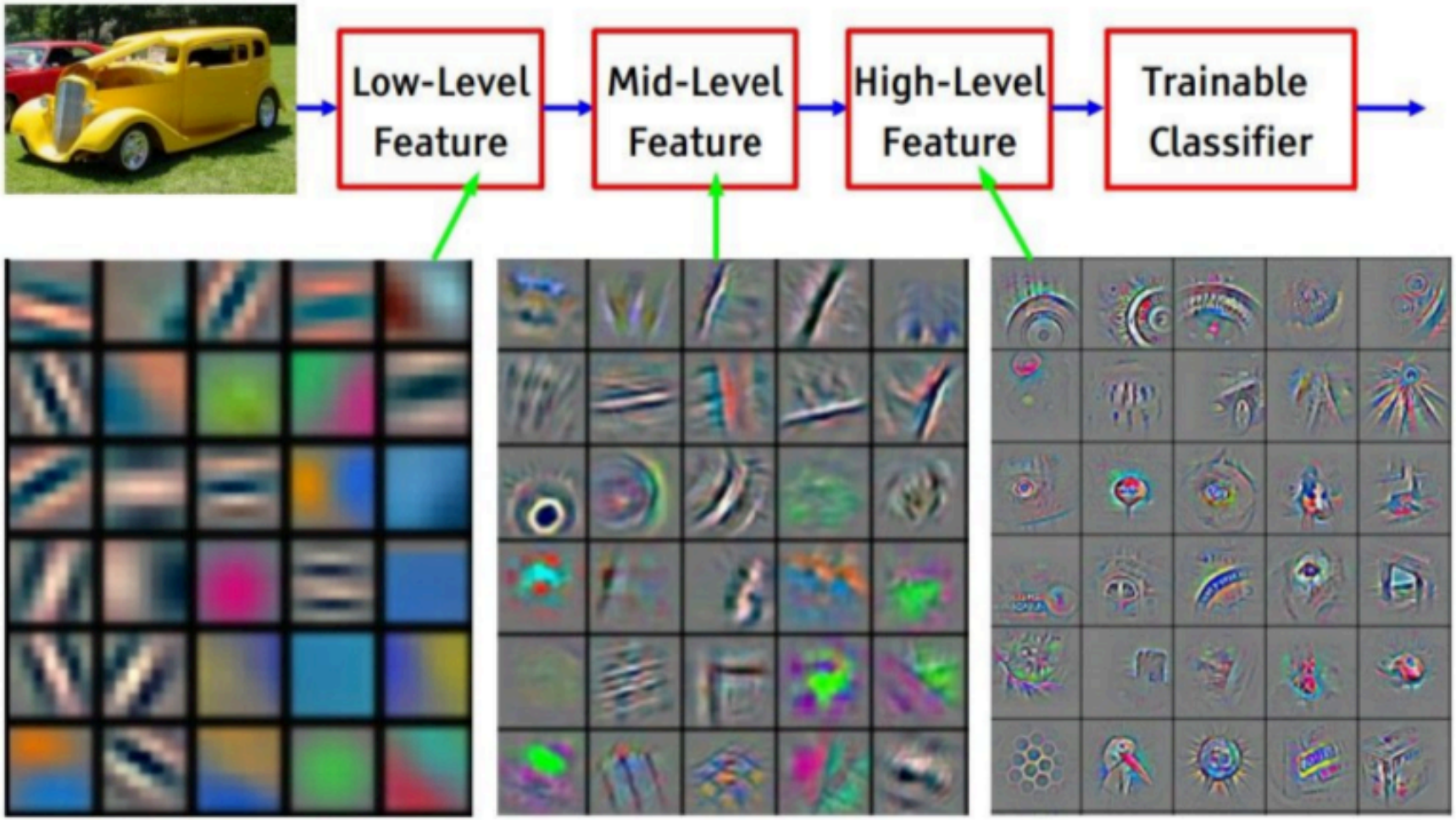
A woman is throwing a **frisbee** in a park.

A little **girl** sitting on a bed with a teddy bear.

Localization and Detection



计算机视觉



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

计算机视觉

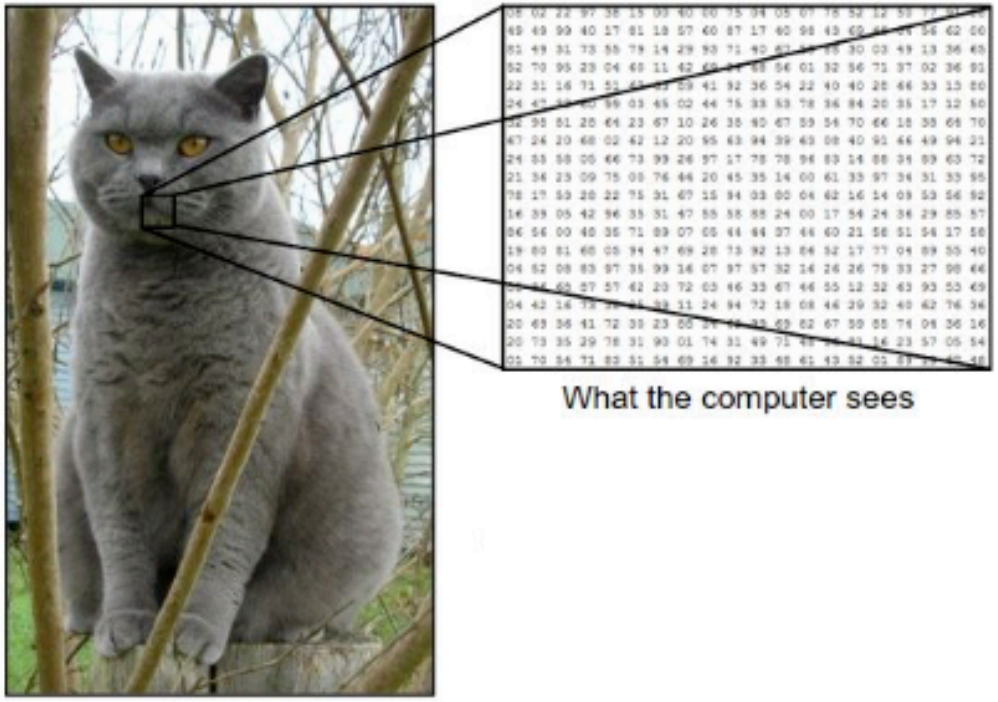
Image Classification: a core task in Computer Vision

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

cat

计算机视觉

The problem:
semantic gap



What the computer sees

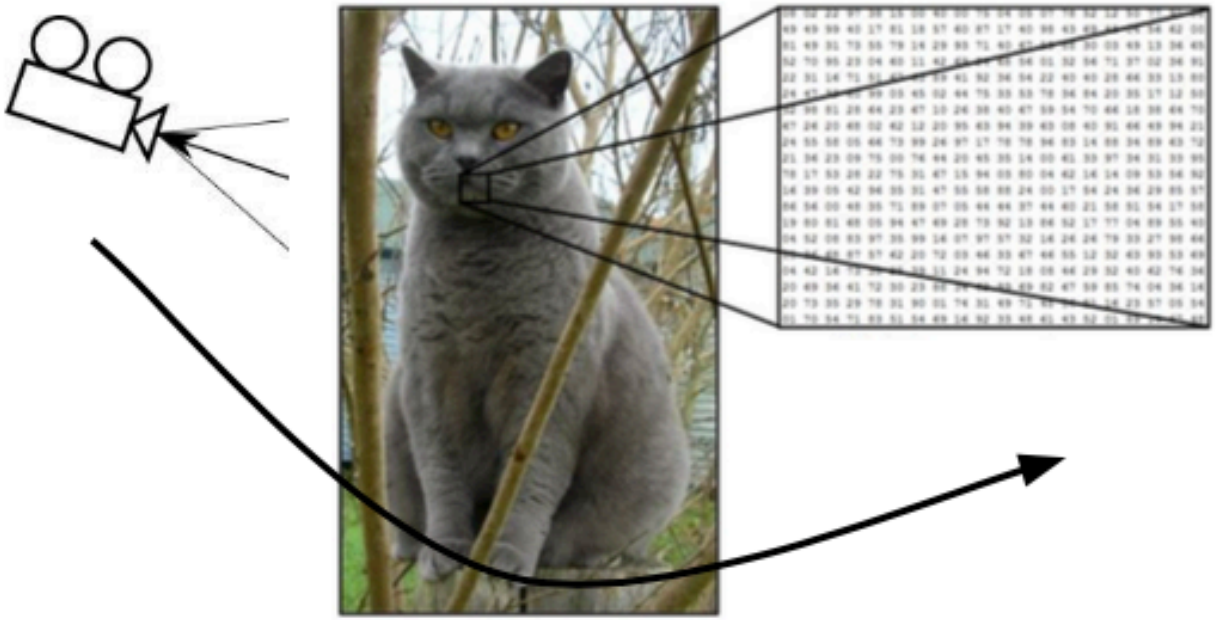
Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)

计算机视觉

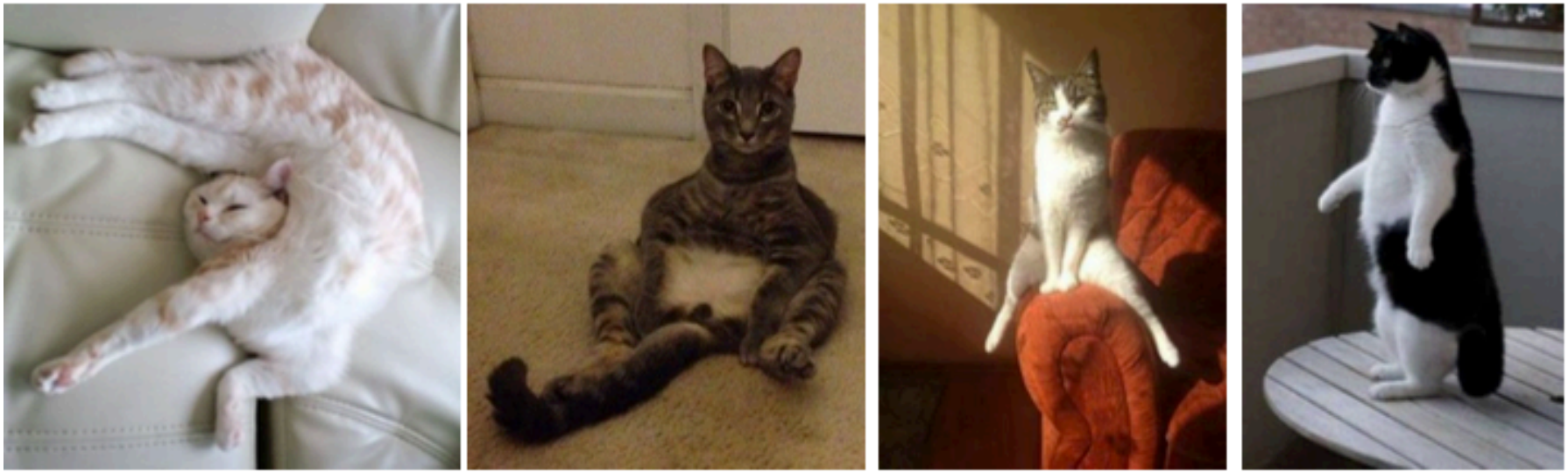
Challenges: Viewpoint Variation



Challenges: Illumination

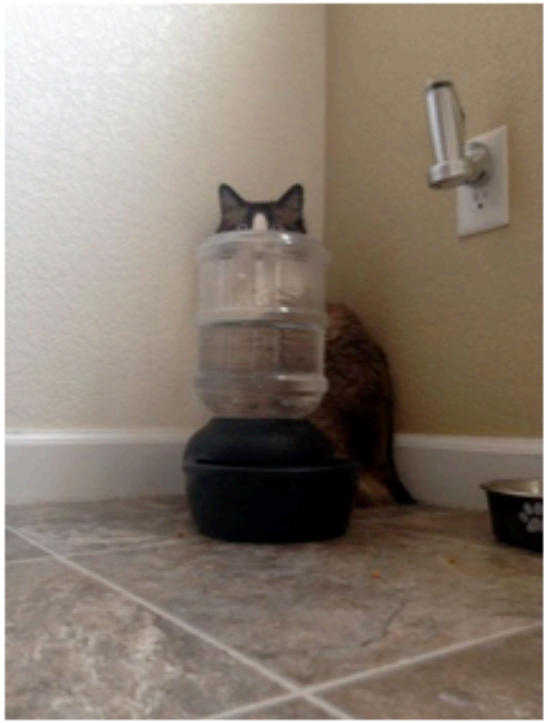
计算机视觉

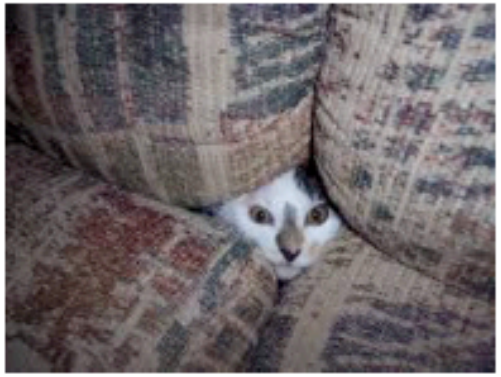
Challenges: Deformation



计算机视觉

Challenges: Occlusion





Challenges: Background clutter

计算机视觉

Challenges: Intraclass variation

ILSVRC

ILSVRC（ImageNet Large Scale Visual Recognition Challenge）

- 计算机视觉比赛ILSVRC使用的数据都来自ImageNet。ImageNet项目于2007年由斯坦福大学华人教授李飞飞创办，目标是收集大量带有标注信息的图片数据供计算机视觉模型训练。
- ImageNet拥有1500万张标注过的高清图片，总共拥有22000类，其中约有100万张标注了图片中主要物体的定位边框。
- ImageNet项目最早的灵感来自于人类通过视觉学习世界的方式，如果假定儿童的眼睛是生物照相机，他们平均每200ms就拍照一次（眼球转动一次的平均时间），那么3岁大时孩子就已经看过了上亿张真实世界的照片，可以算得上是一个非常大的数据集。
- ImageNet项目下载了互联网上近10亿张图片，使用亚马逊的土耳其机器人平台实现众包的标注过程，有来自世界上167个国家的近5万名工作者帮忙一起筛选、标注。

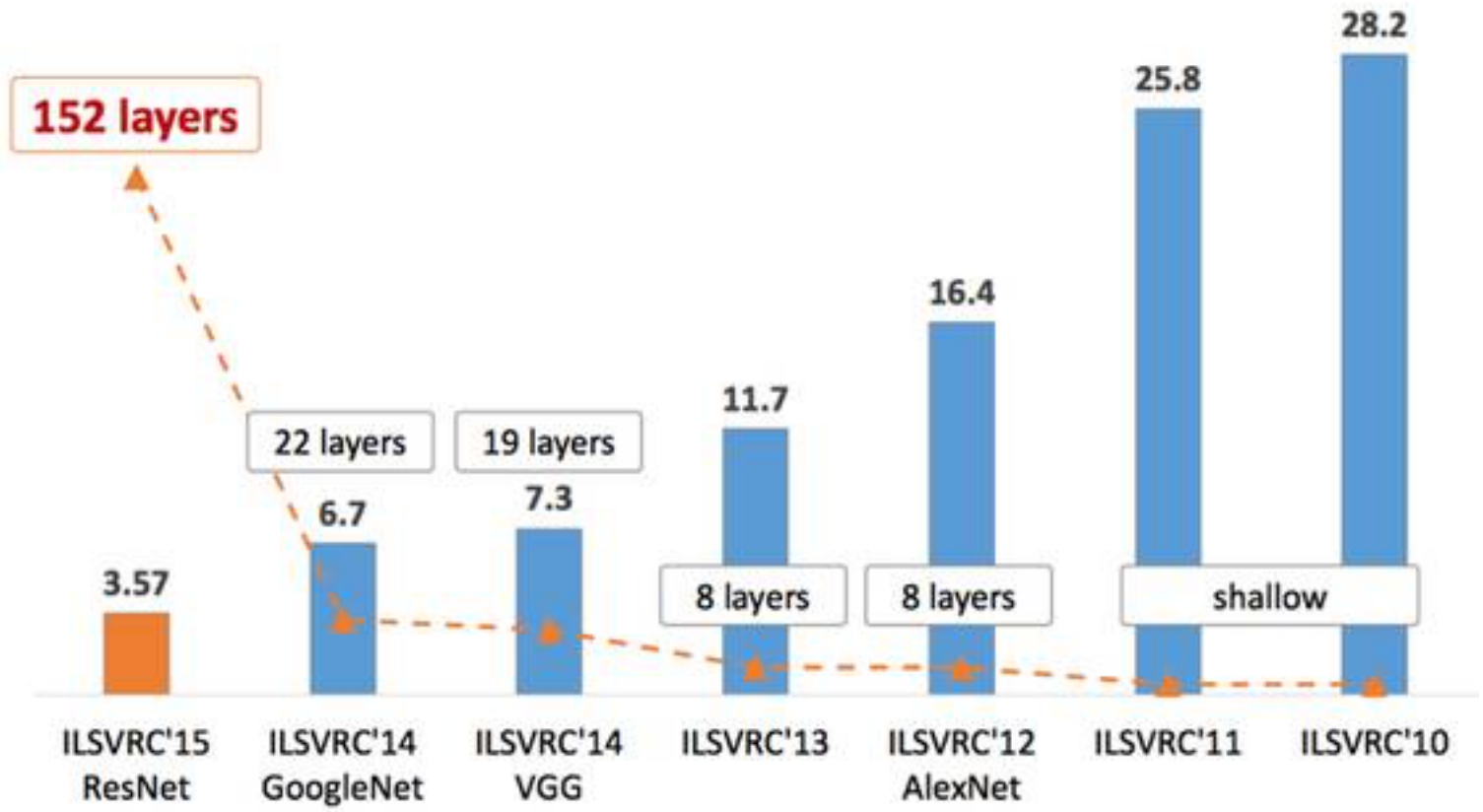
ILSVRC

ILSVRC（ImageNet Large Scale Visual Recognition Challenge）

- 每年度的ILSVRC比赛数据集中大概拥有120万张图片，以及1000类的标注，是ImageNet全部数据的一个子集。比赛一般采用top-5和top-1分类错误率作为模型性能的评测指标
- AlexNet识别ILSVRC数据集中图片的情况，每张图片下面是分类预测得分最高的5个分类及其分值。

ILSVRC

ILSVRC（ImageNet Large Scale Visual Recognition Challenge）



ILSVRC

ILSVRC（ImageNet Large Scale Visual Recognition Challenge）

Ordered by classification error

Team name	Entry description	**Classification error**	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink	GrandUnion + Fused-scale EnsembleNet	0.03042	0.098892
CU-DeepLink	GrandUnion + Multi-scale EnsembleNet	0.03046	0.099006
CU-DeepLink	GrandUnion + Basic Ensemble	0.03049	0.098954
ResNeXt	Ensemble B, weighted average, tuned on val. [No bounding box results]	0.03092	0.737484
CU-DeepLink	GrandUnion + Class-reweighted Ensemble	0.03096	0.099369
CU-DeepLink	GrandUnion + Class-reweighted Ensemble with Per-instance Normalization	0.03103	0.099349
ResNeXt	Ensemble C, weighted average. [No bounding box results]	0.03124	0.737526
Trimps-Soushen	Ensemble 1	0.03144	0.079068
ResNeXt	Ensemble A, simple average. [No bounding box results]	0.0315	0.737505
SamExynos	3 model only for classification	0.03171	0.236561
ResNeXt	Ensemble B, weighted average. [No bounding box results]	0.03203	0.737681
KAISTNIA_ETRI	Ensembles A	0.03256	0.102015
KAISTNIA_ETRI	Ensembles C	0.03256	0.102056
KAISTNIA_ETRI	Ensembles B	0.03256	0.100676

ILSVRC

ILSVRC（ImageNet Large Scale Visual Recognition Challenge）

ILSVRC2016落下帷幕，中国学术界和工业界团队包揽了多项冠军。具体成绩如下：

CUImage（商汤科技和港中文）：标检测第一；
Trimps-Soushen（公安部三所）：目标定位第一；
CUvideo（商汤和港中文）：视频中物体检测子项目第一；
NUIST（南京信息工程大学）：视频中的物体探测两个子项目第一；
HikVision（海康威视）：场景分类第一；
SenseCUSceneParsing（商汤和港中文）：场景分析第一。

最常用的深度学习模型：卷积神经网络

CNN: Convolutional NN [LeCun and Bengio, 1995; LeCun et al. , 1998]

卷积神经网络可以成功识别人脸、物体和交通信号，从而为机器人和自动驾驶汽车提供视力。

每个卷积层包含多个特征映射，每个特征映射是一个由多个神经元构成的“平面”，通过一种卷积滤波器提取输入的一种特征

采样层亦称“汇合(pooling)层”，其作用是基于局部相关性原理进行亚采样，从而在减少数据量的同时保留有用信息

连接层就是传统神经网络对隐层与输出层的全连接

四个主要操作：
1.卷积
2.非线性处理（ReLU）
3.池化或者亚采样
4.分类（全连接层）

LeNet

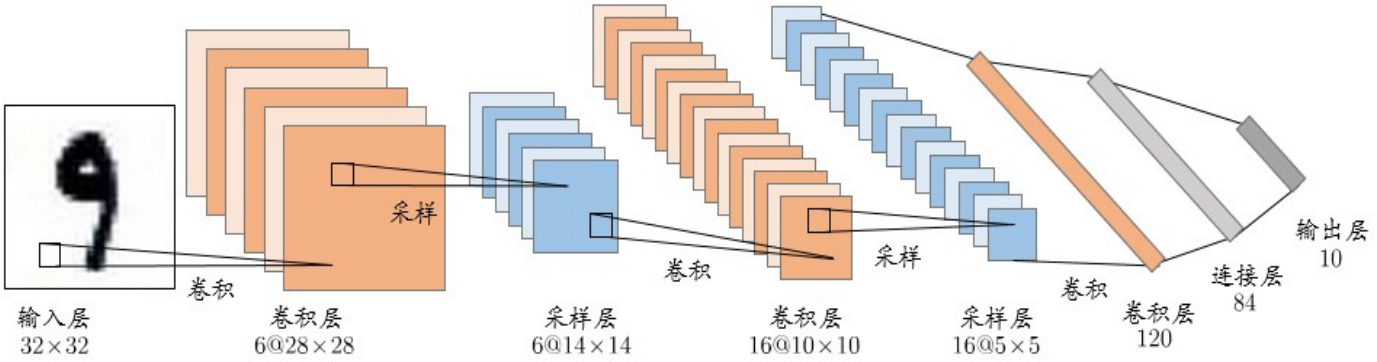
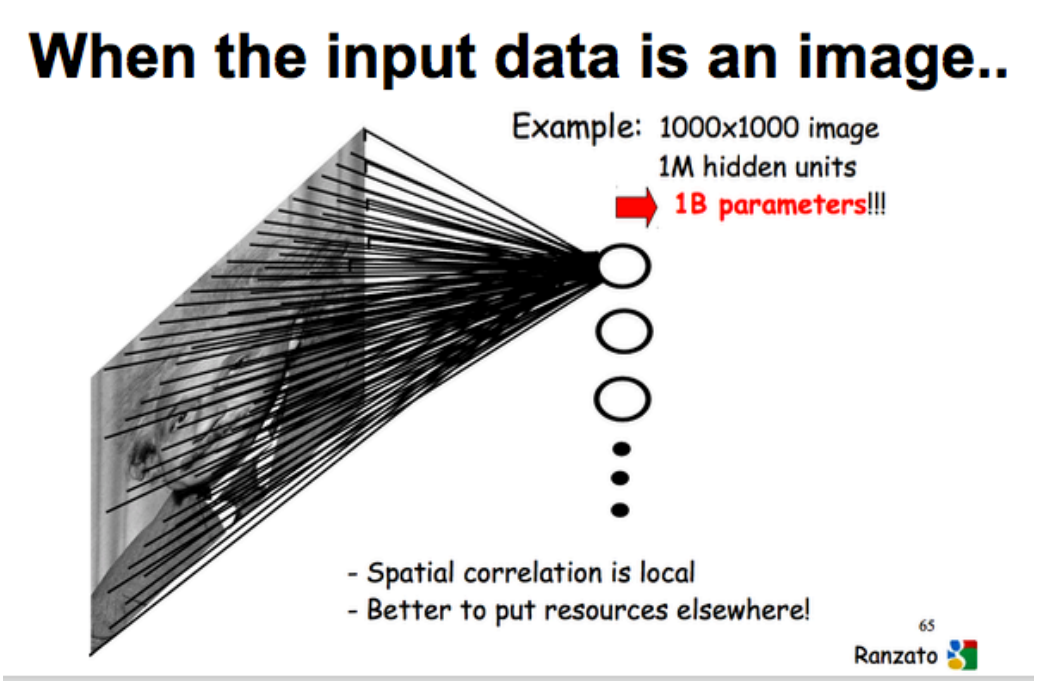


图 5.15 卷积神经网络用于手写数字识别 [LeCun et al., 1998]

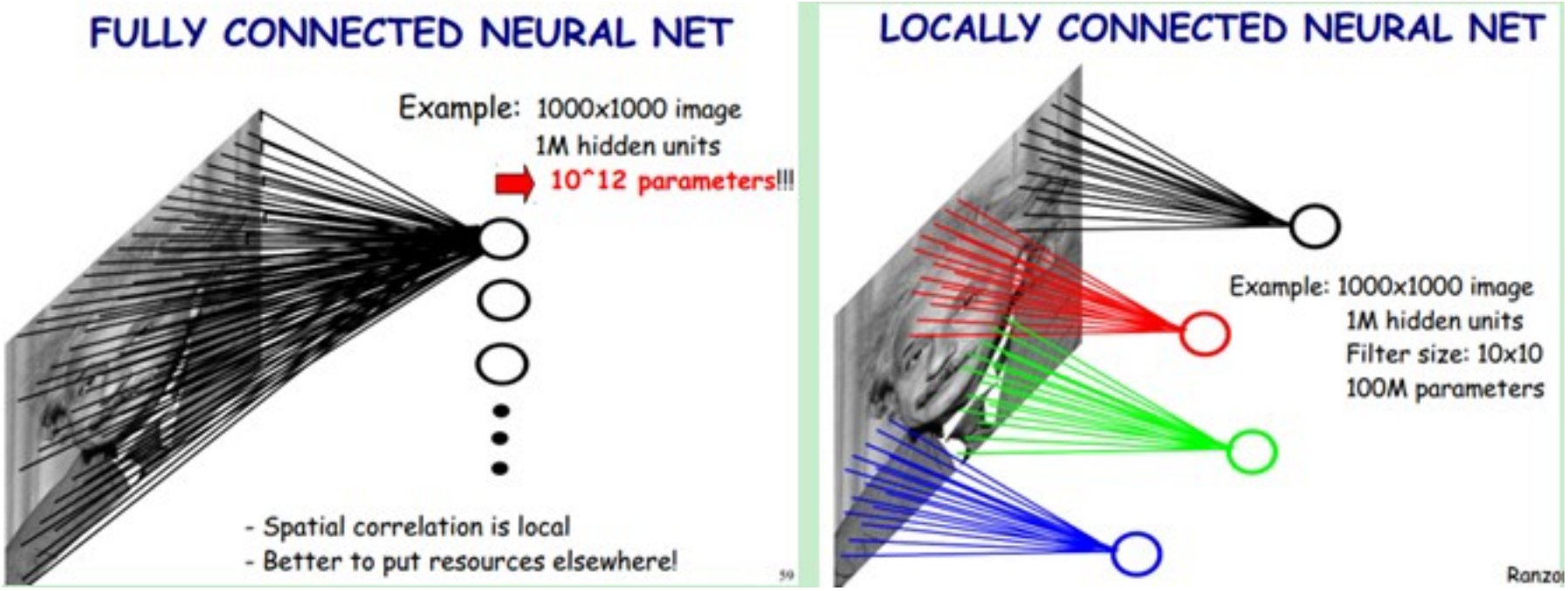
卷积神经网络



- 仔细想一想的话，使用全连接的网络来识别图像有一些奇怪。因为这样的网络结构没有考虑图像的空间结构。比如，它对于空间上很近或者很远的像素一样的对待。这些空间的概念必须靠网络从训练数据中推测出来，但是如果训练数据不够而且图像没有做居中等归一化的话，那么网络很可能学不到这样的特征。

在图像处理中，往往把图像表示为像素的向量，比如一个1000×1000的图像，可以表示为一个10^6的向量。如果**隐层数目与输入层一样**，即也是10^6时，那么输入层到隐含层的参数数据为$10^6 \times 10^6 = 10^{12}$，这样就太多了，基本没法训练。

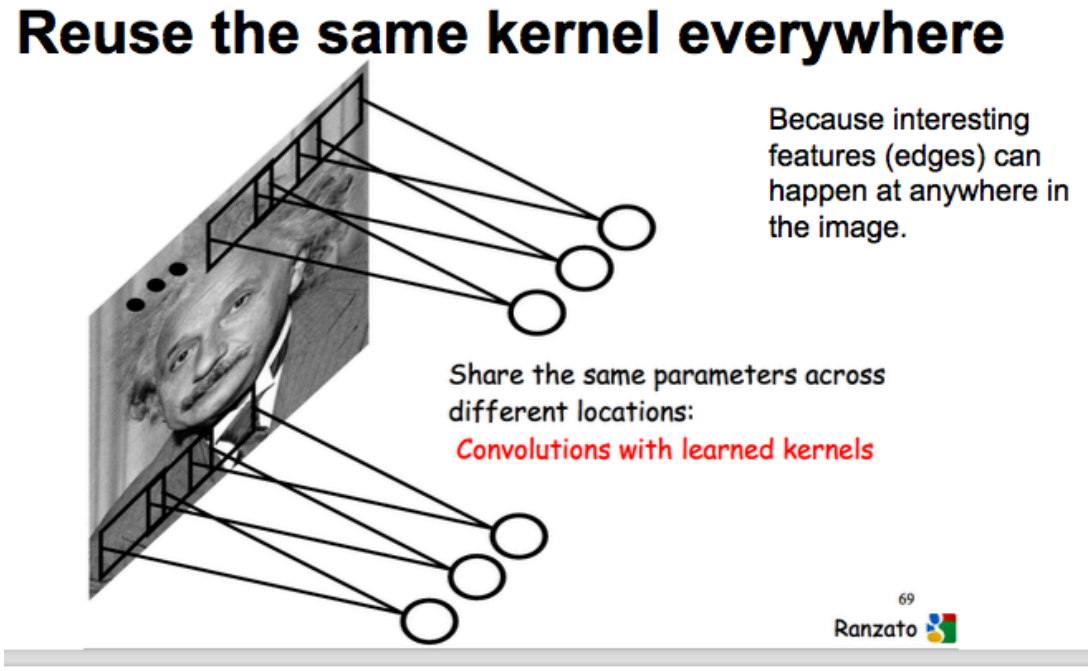
卷积神经网络 – 局部感知



模仿人的眼睛，想想看，我们在看东西的时候，目光是聚焦在一个相对很小的局部的。

假如每个神经元只和10×10个像素值相连，那么权值数据为10^6×100个参数，减少为原来的万分之一。-- 依然太多！

卷积神经网络 – 权值共享

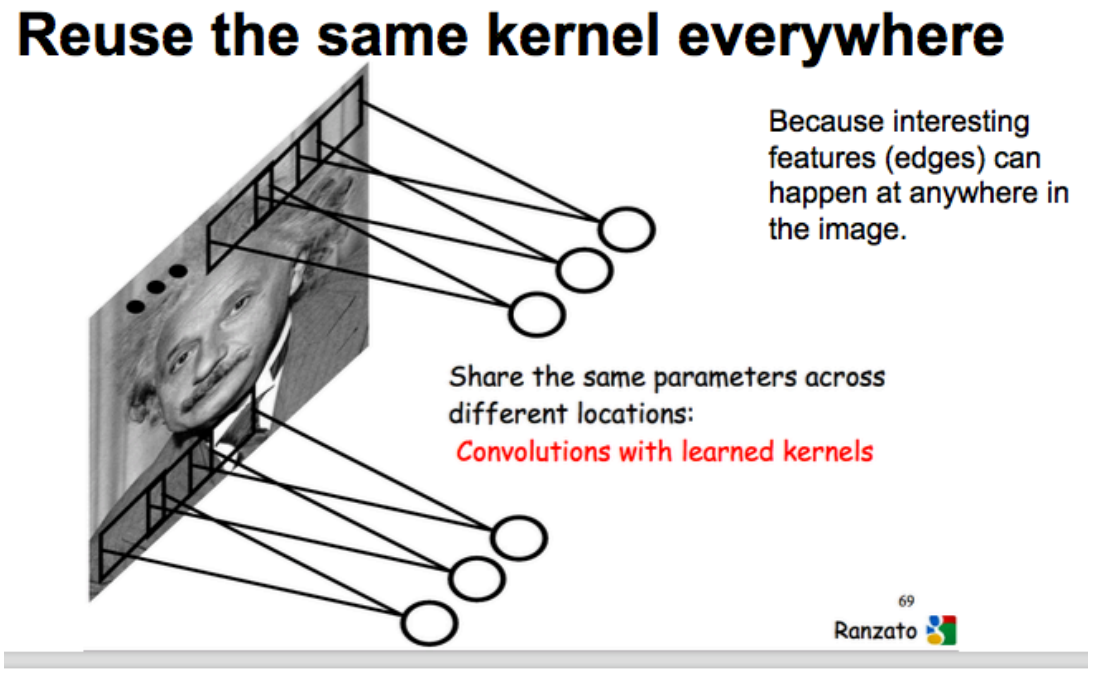


形象地说，就如同某个神经中枢中的神经细胞，它们的结构、功能是相同的，甚至是可以互相替代的。也就是，在卷积神经网中，同一个卷积核内，**所有的神经元的权值是相同的**，从而大大减少需要训练的参数。

假如每个神经元只和10×10个像素值相连（有100个参数），那么权值数据为1000000×100个参数，减少为原来的万分之一。-- 依然太多！

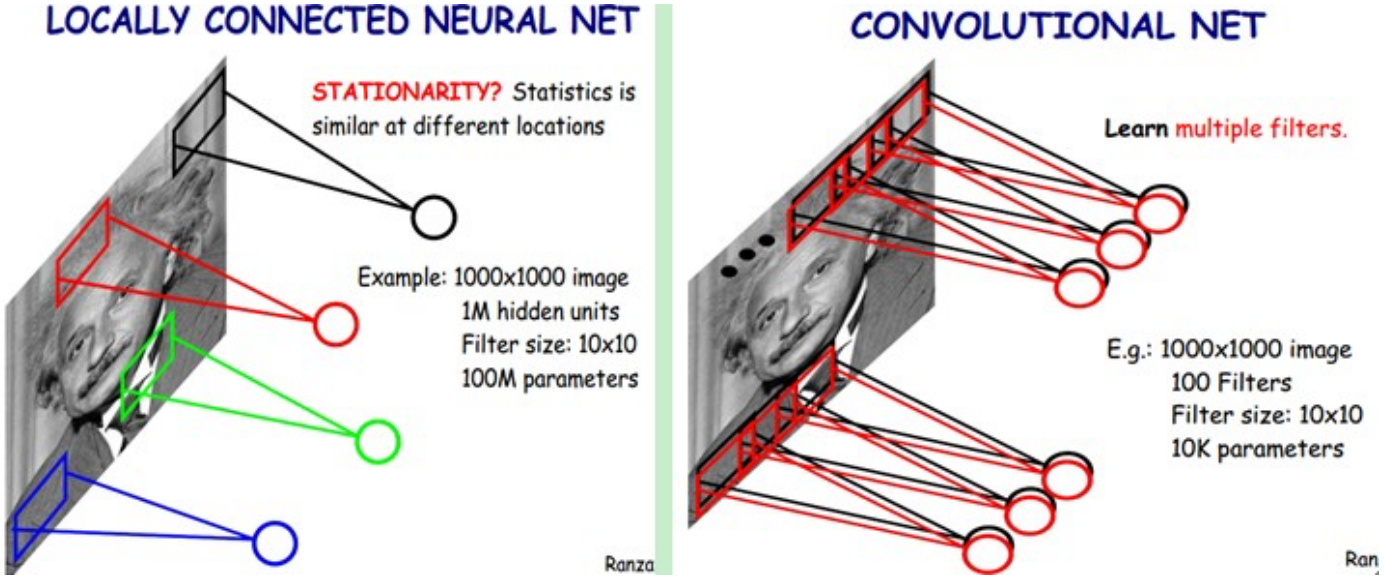
如果我们每个神经元这100个参数是相同的呢？也就是说所有神经元用的是同一个卷积核去卷积图像。不管隐层的神经元个数有多少，两层间的连接只有100个参数！-- 权值共享。

卷积神经网络 - 权值共享



- 这意味着这一个隐藏层的所有神经元都是检测同一个特征，只不过它们位于图片的不同位置而已。比如是某个局部感知域学到的用来识别一个图片中的猫的轮廓。
- 那么预测的时候不管个轮廓是在哪个位置，它都会被某个对应的局部感知域检测到。
- 更抽象一点，卷积网络能很好的适应图片的位置变化：把图片中的猫稍微移动一下位置，它仍然知道这是一只猫。

卷积神经网络 - 特征映射(feature map)

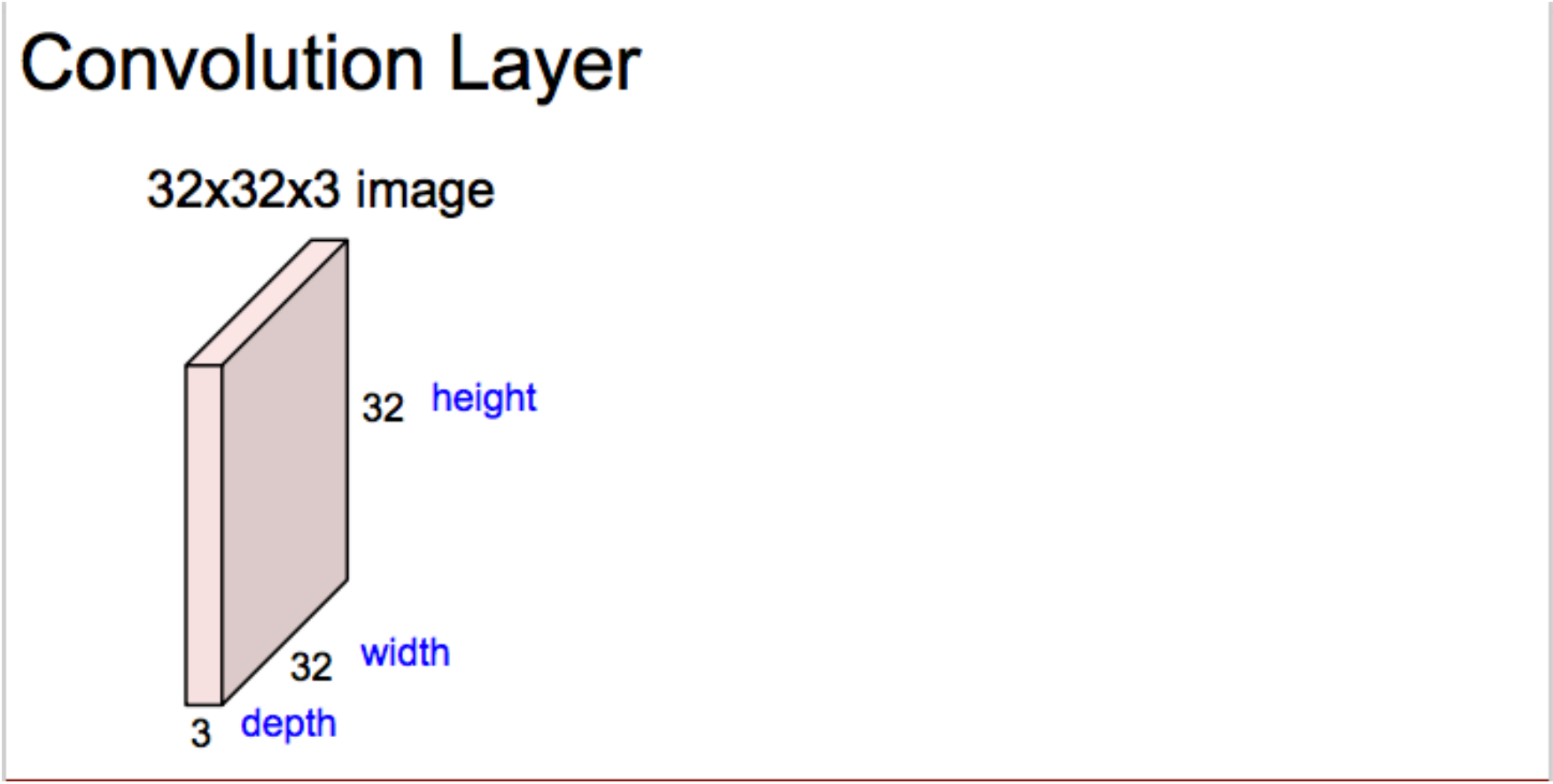


需要提取多种特征！

- 使用多个滤波器（filter）！每个filter提取不同的特征。轮廓，鼻子，眼睛….
- 假设加到100种滤波器，每种滤波器的参数不一样，表示提取出输入图像的不同特征。
- 100种卷积核 x 每种卷积核共享100个参数=100x100=10K，也就是1万个参数。
- 每种滤波器去卷积图像就得到对图像的不同特征的反映，称之为Feature Map。
- 100种卷积核就有100个Feature Map，这100个Feature Map就组成了一层神经元。

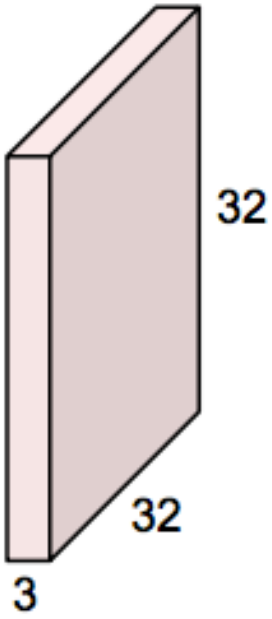
卷积神经网络 - 卷积层

卷积的主要目的是为了从输入图像中提取特征。卷积可以通过从输入的一小块数据中学到图像的特征，并可以保留像素间的空间关系。

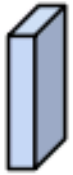


Convolution Layer

32x32x3 image



5x5x3 filter



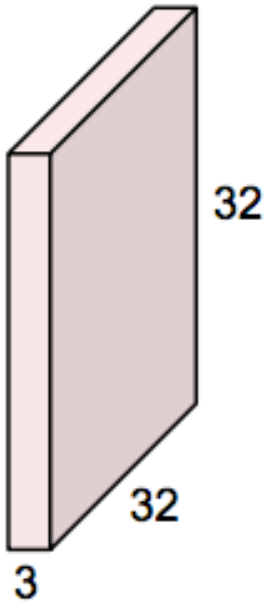
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

卷积神经网络 - 卷积层

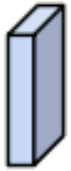
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

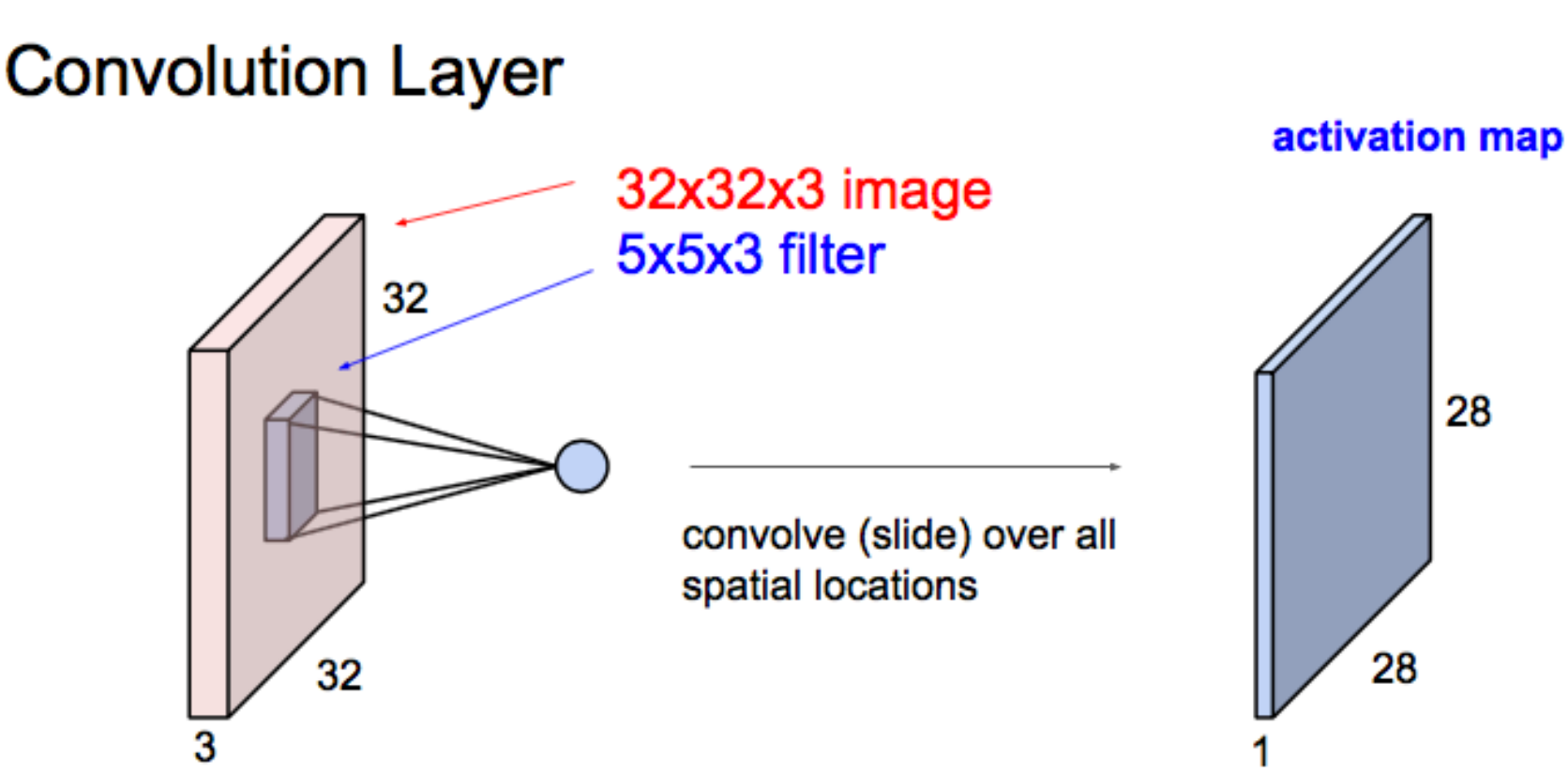


5x5x3 filter

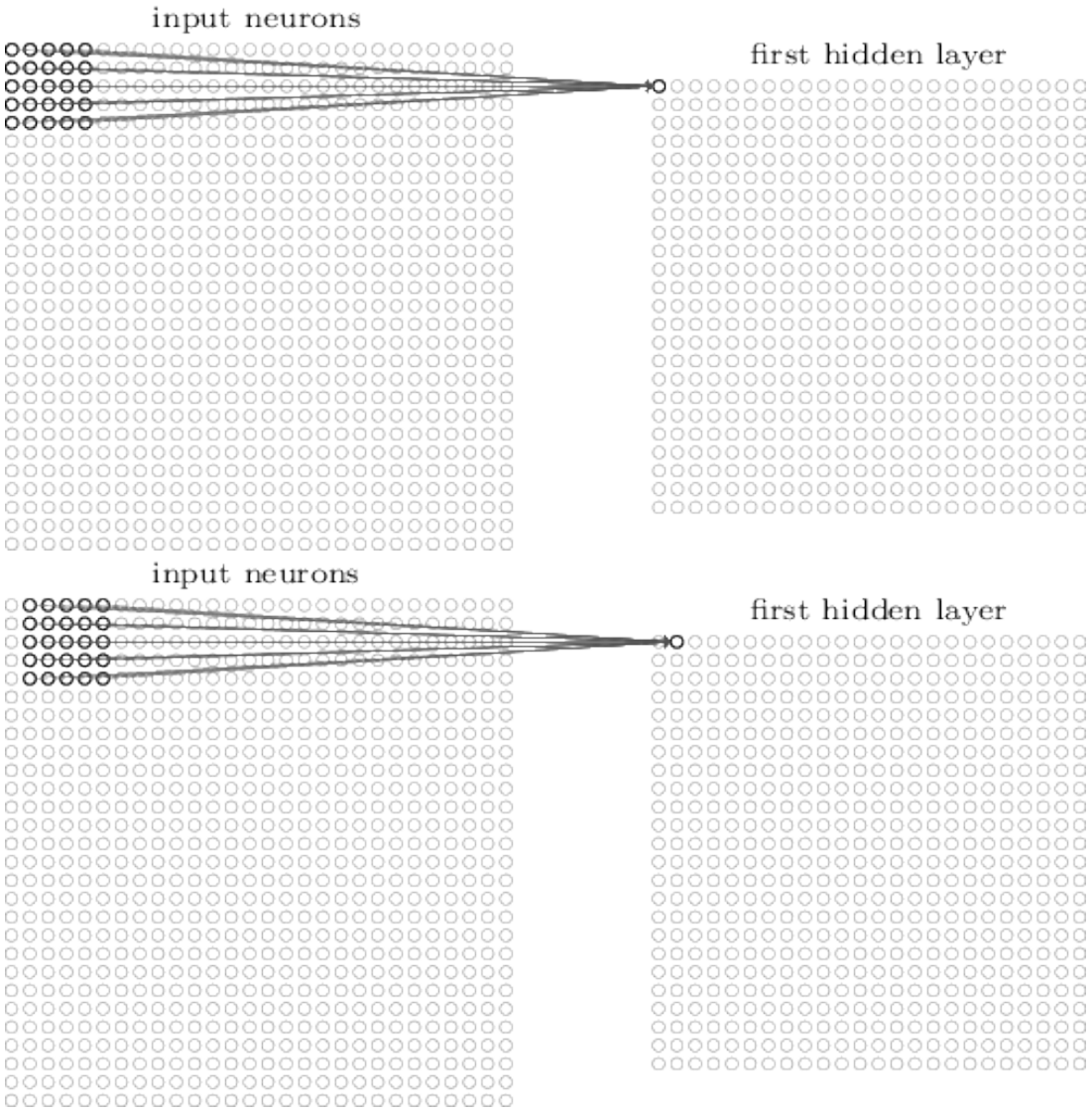


Convolve the filter with the image
i.e. "slide over the image spatially, computing dot products"

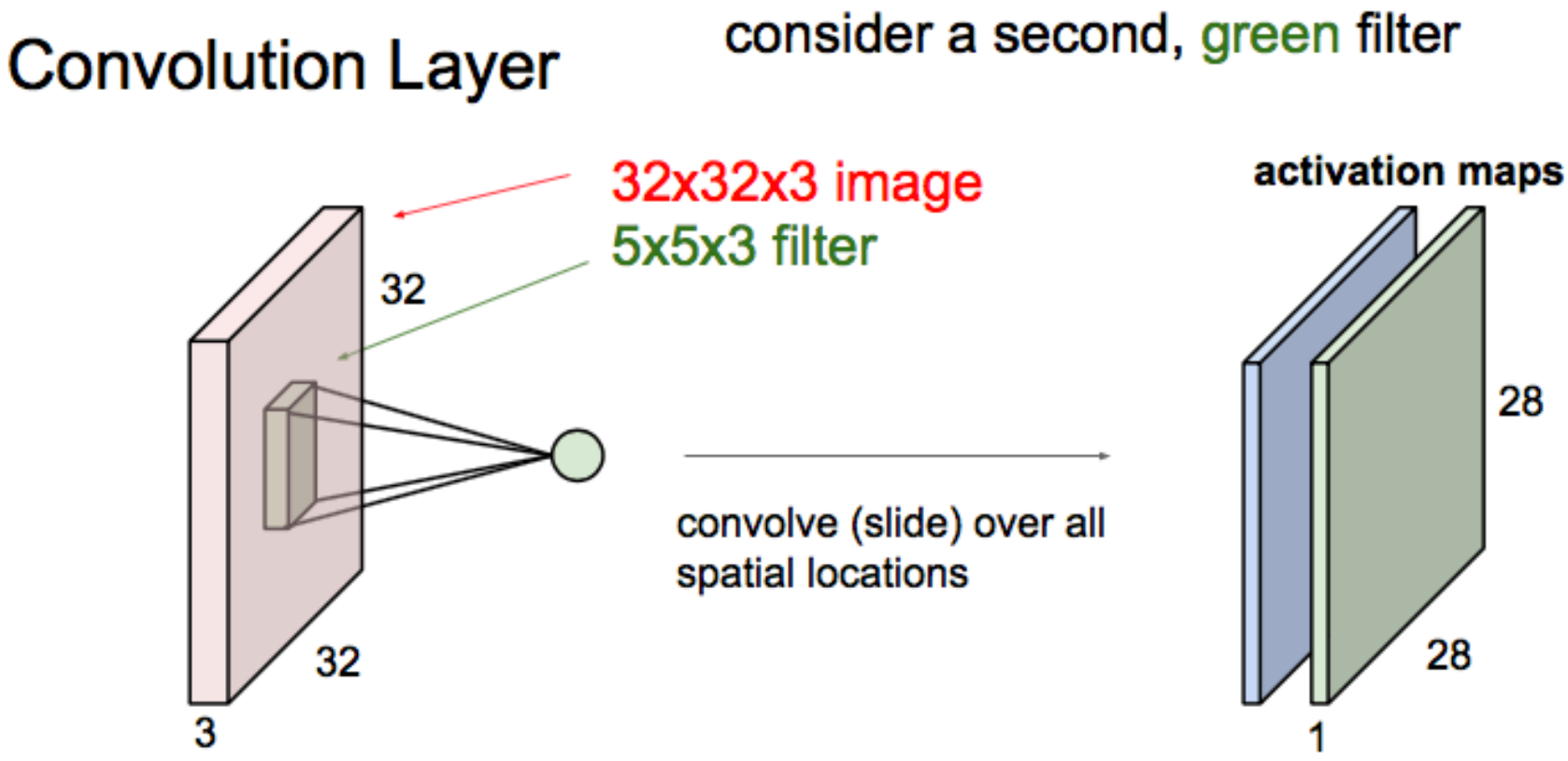
卷积神经网络 - 卷积层


Convolution Layer
activation map
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
28
28
1

卷积神经网络 - 卷积层

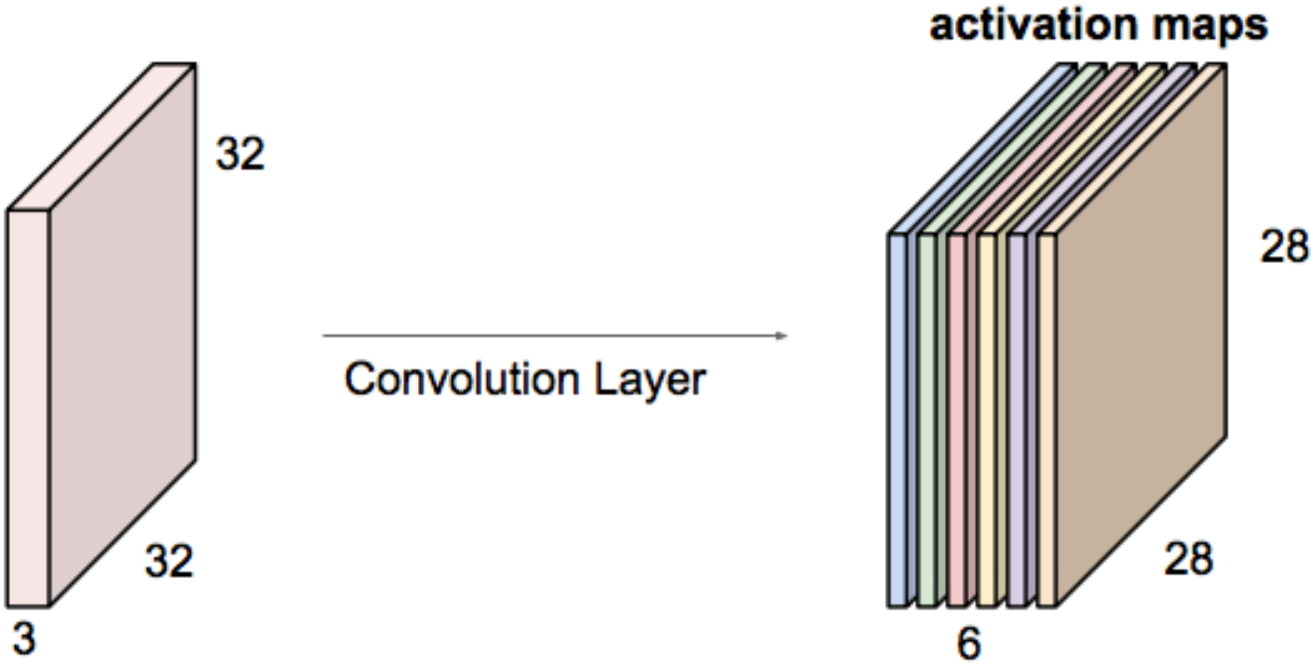

input neurons
first hidden layer
input neurons
first hidden layer

卷积神经网络 - 卷积层


Convolution Layer
consider a second, green filter
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation maps
28
28
1

卷积神经网络 – 卷积层

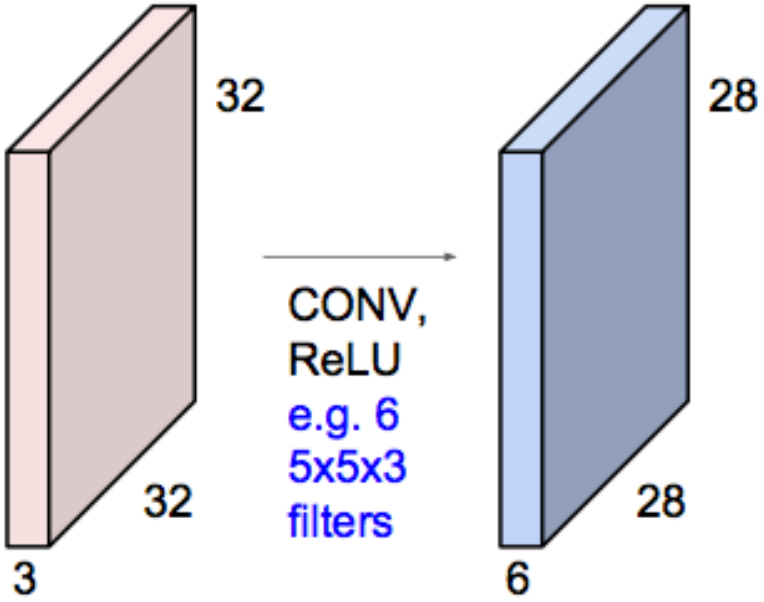
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



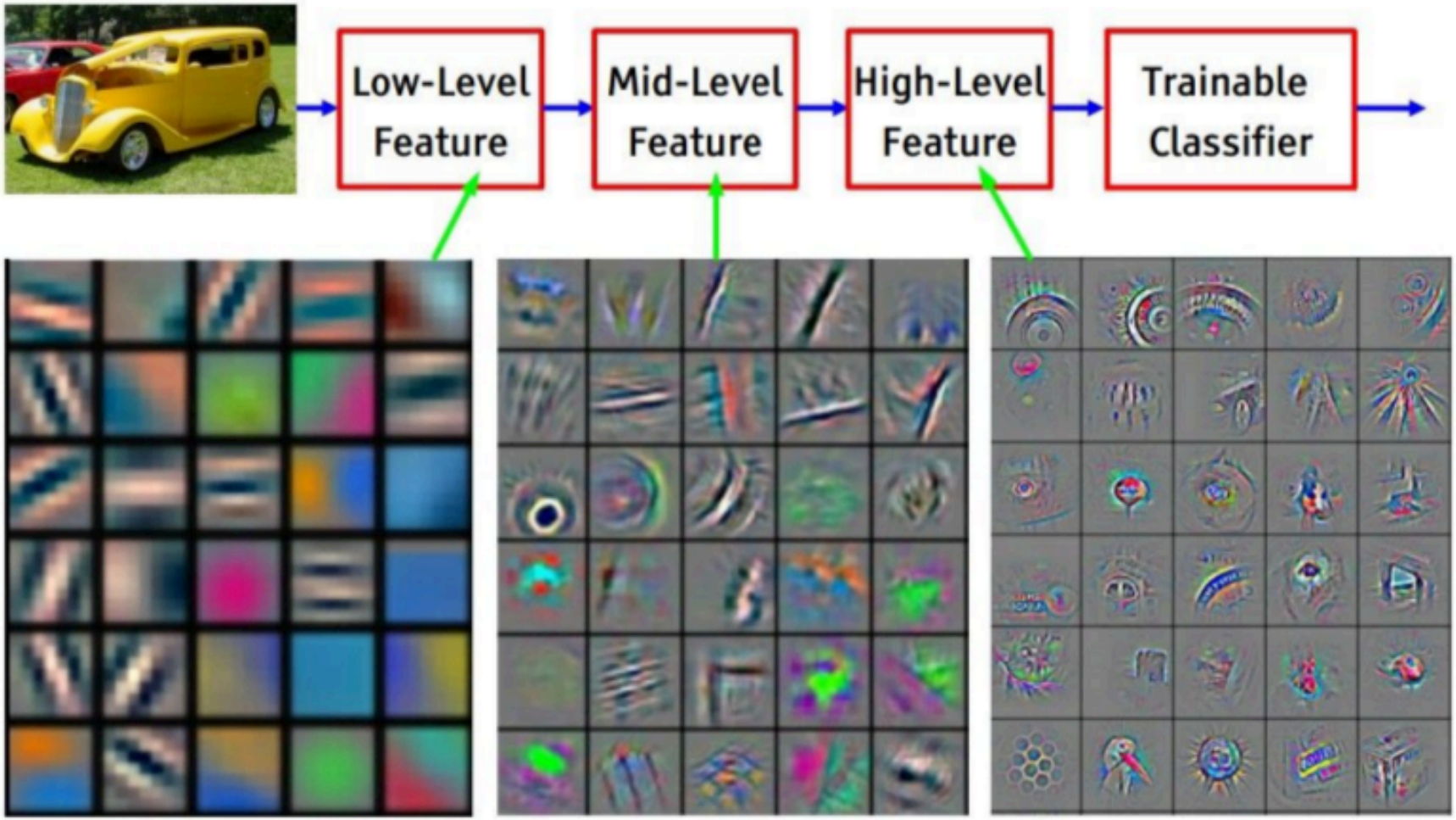
We stack these up to get a "new image" of size 28x28x6!

卷积神经网络 - 卷积层

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

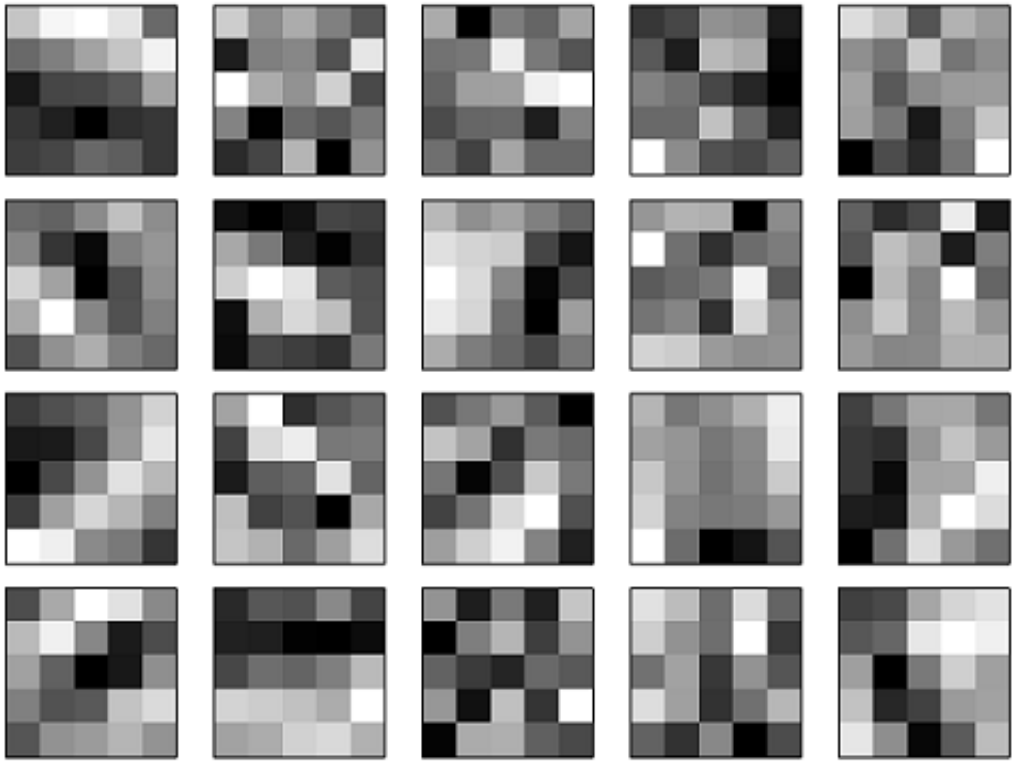


卷积层：学到的特征 Feature



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

卷积神经网络 – 卷积层



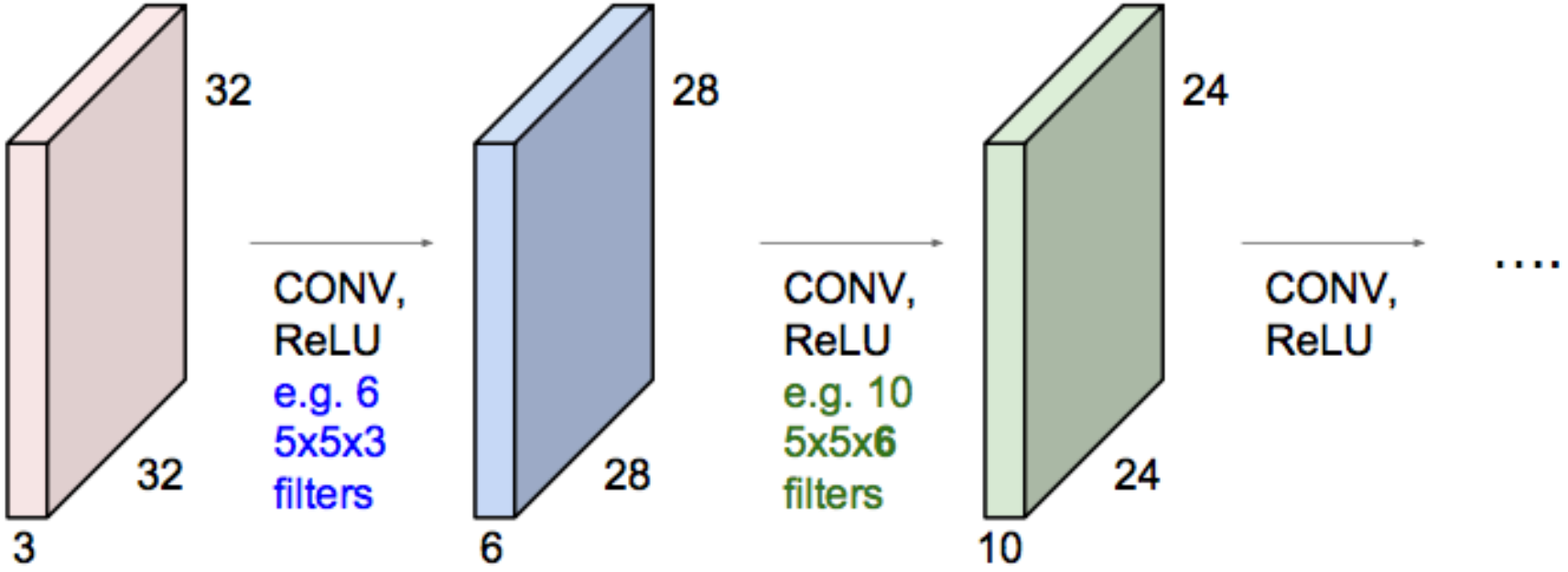
- 这20个图片对应了20个不同的特征映射。
- 每个映射是一个5×5的图像，对应于局部感知域的5×5个权重。
- 颜色越白(浅)说明权值越小(一般都是负的)，因此对应像素对于识别这个特征越不重要。颜色越深(黑)说明权值越大，对应的像素越重要。

那么我们可以从这些特征映射里得出什么结论呢？很显然这里包含了非随机的空间结构。这说明我们的网络学到了一些空间结构。**但是，也很难说它具体学到了哪些特征!!!**

有很多研究工作尝试理解机器到底学到了什么样的特征。如：Visualizing and Understanding Convolutional Networks

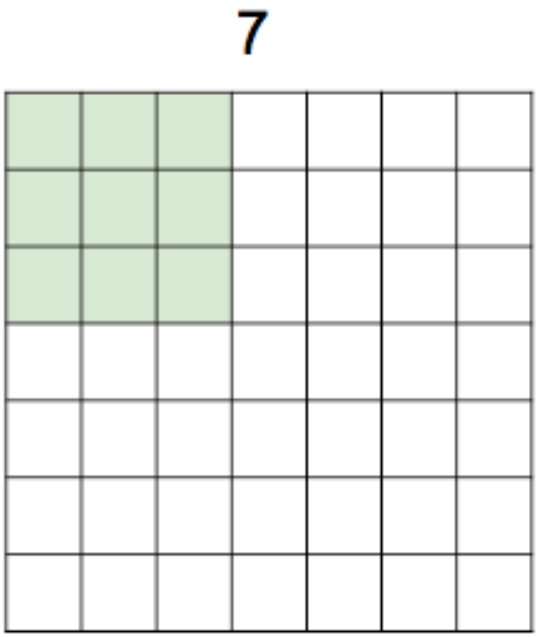
卷积神经网络 - 卷积层

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



卷积神经网络 - Spatial arrangement

A closer look at spatial dimensions:

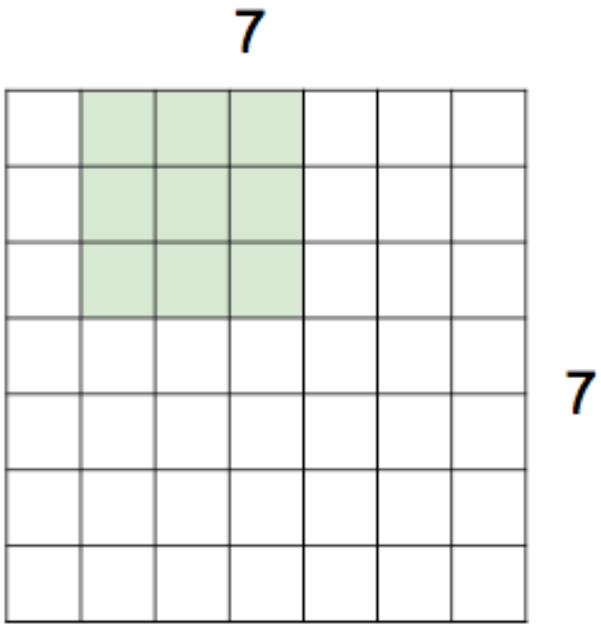


7x7 input (spatially)
assume 3x3 filter

Stride = 1

卷积神经网络 - Spatial arrangement

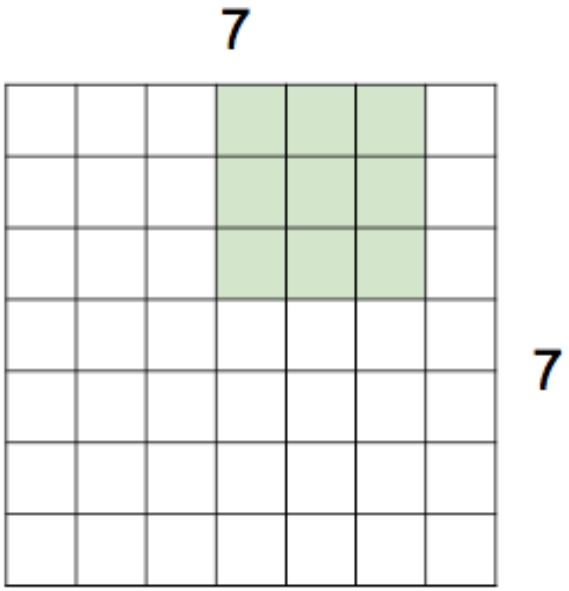
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

卷积神经网络 - Spatial arrangement

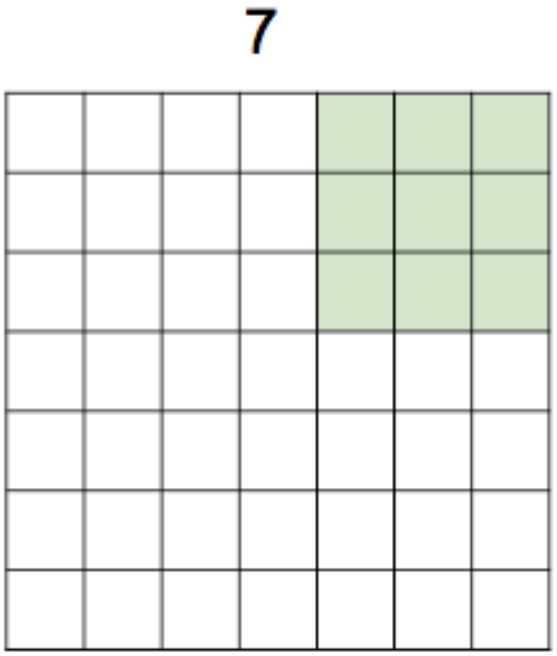
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

卷积神经网络 - Spatial arrangement

A closer look at spatial dimensions:

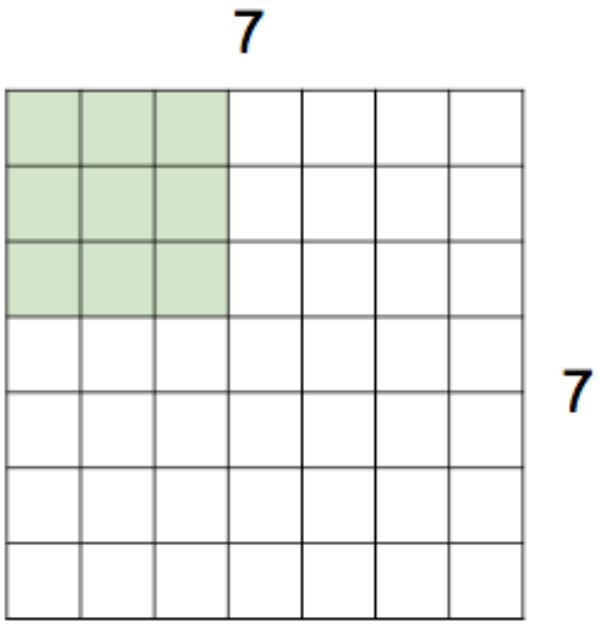


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

卷积神经网络 - Spatial arrangement

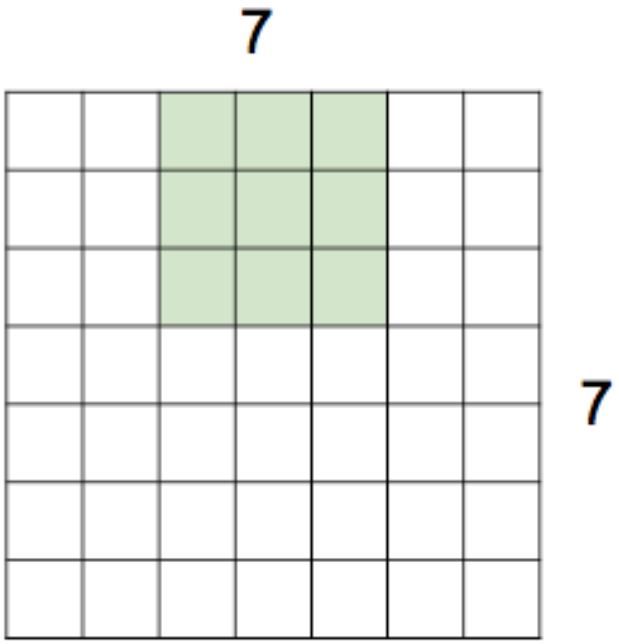
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

卷积神经网络 - Spatial arrangement

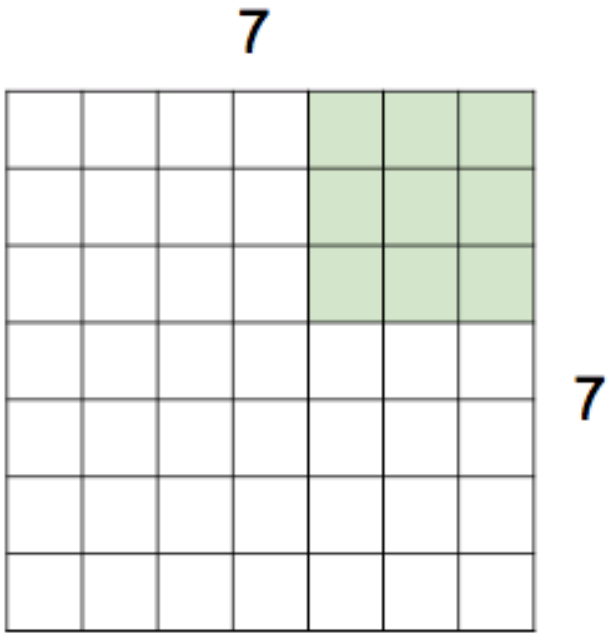
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

卷积神经网络 - Spatial arrangement

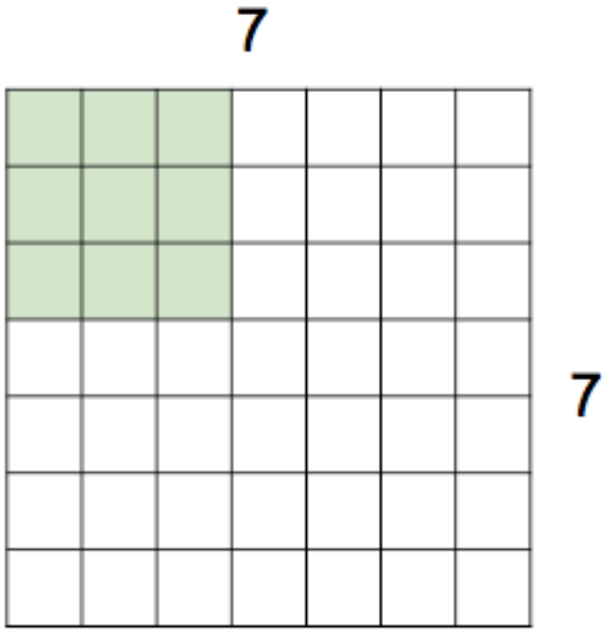
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

卷积神经网络 - Spatial arrangement

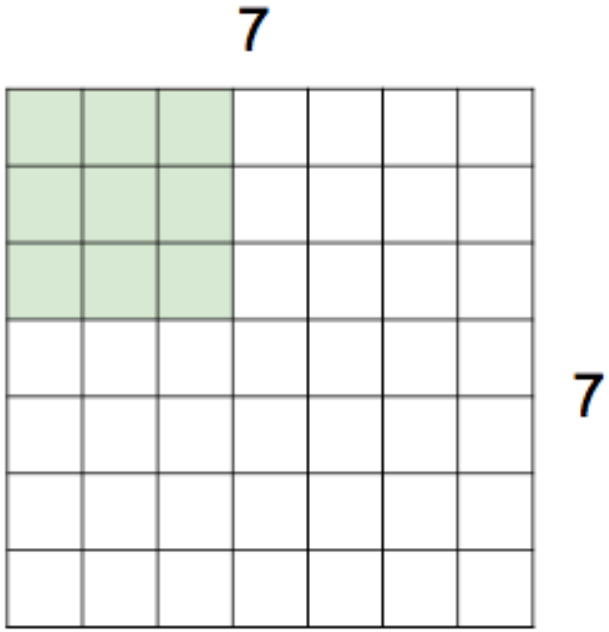
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

卷积神经网络 - Spatial arrangement

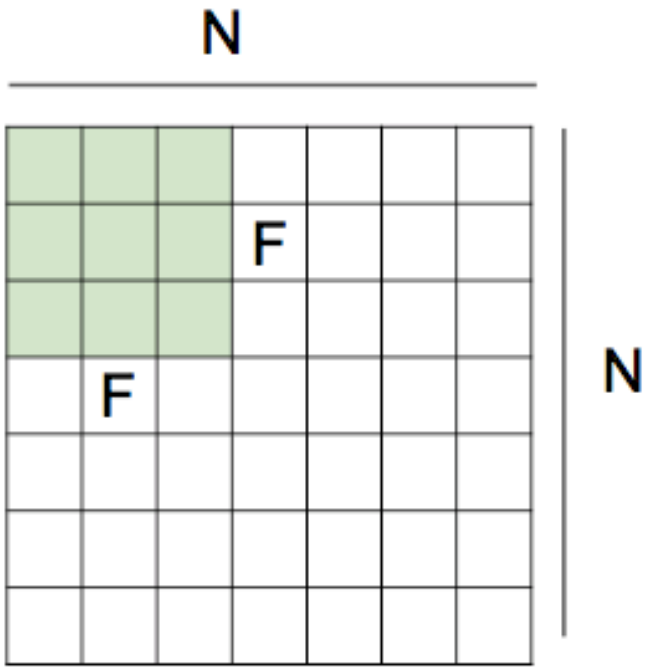
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

卷积神经网络 - Spatial arrangement

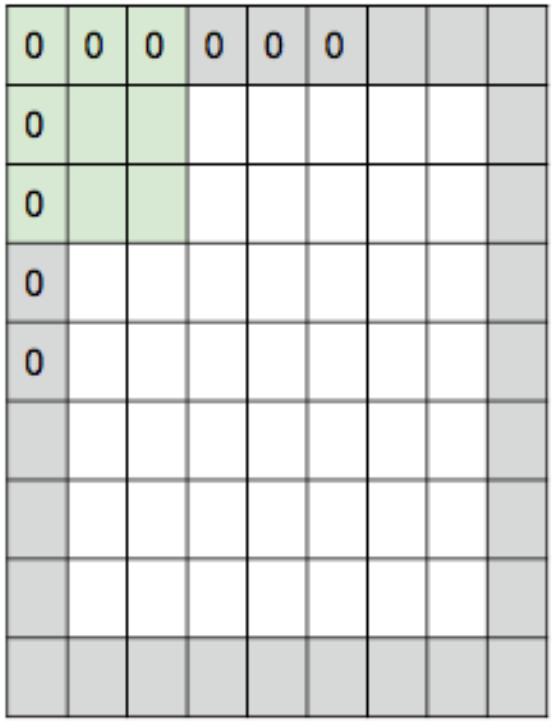


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

卷积神经网络 - Zero Padding

In practice: Common to zero pad the border

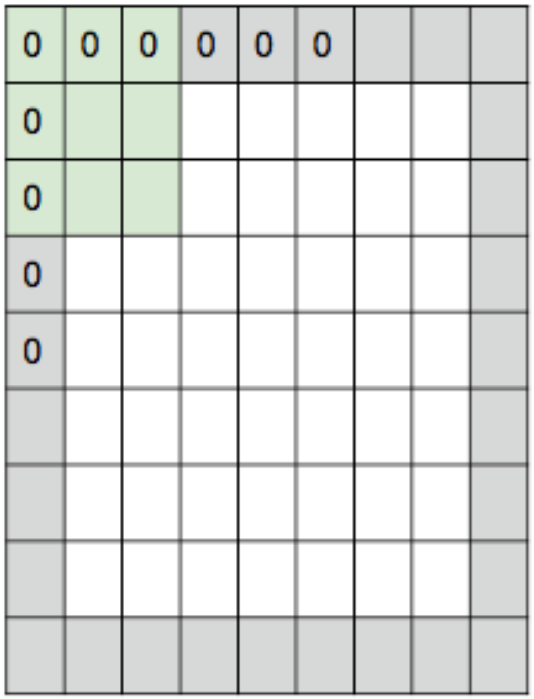


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

卷积神经网络 - Zero Padding

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Output size:
(N – F + 2P) / stride + 1

Why use padding? In addition to the aforementioned beneQt of keeping the spatial sizes constant after CONV, doing this actually improves performance. If the CONV layers were to not zero-pad the inputs and only perform valid convolutions, then the size of the volumes would reduce by a small amount after each CONV, and the information at the borders would be "washed away" too quickly.

卷积神经网络 - 卷积过程

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.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

卷积神经网络 - 卷积过程

Example: CONV layer in Caffe

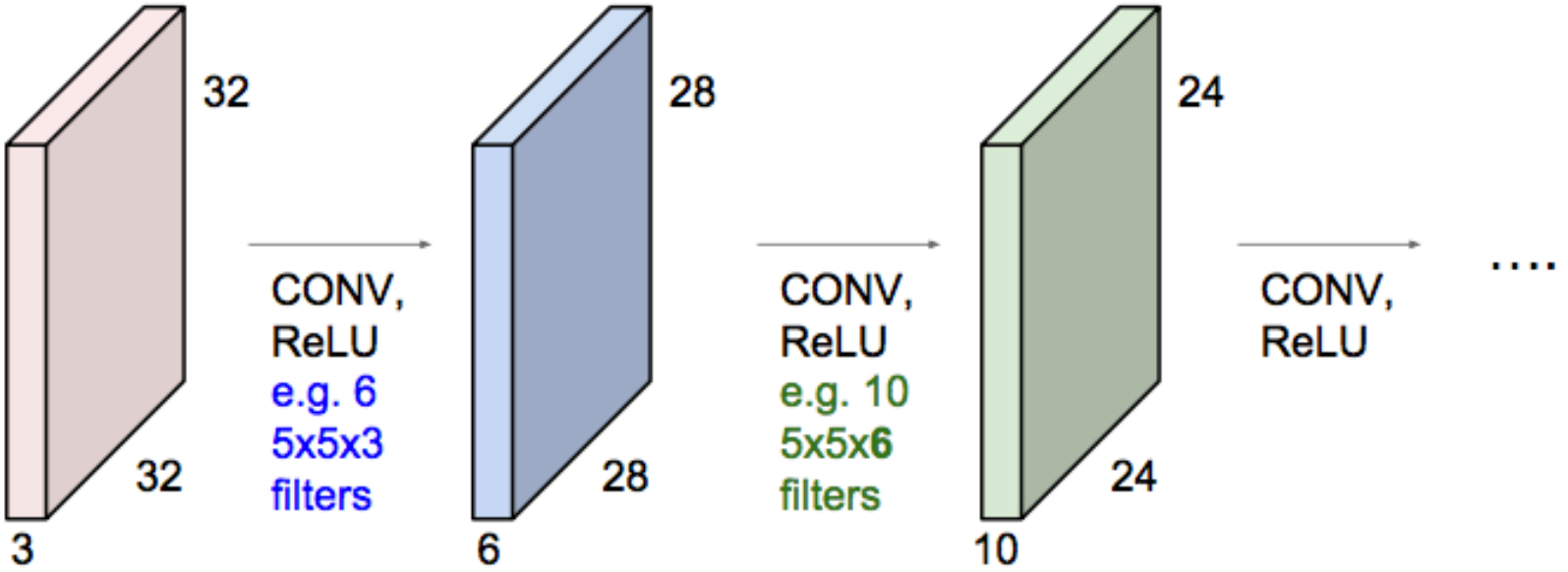
```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

Summary. To summarize, the Conv Layer:

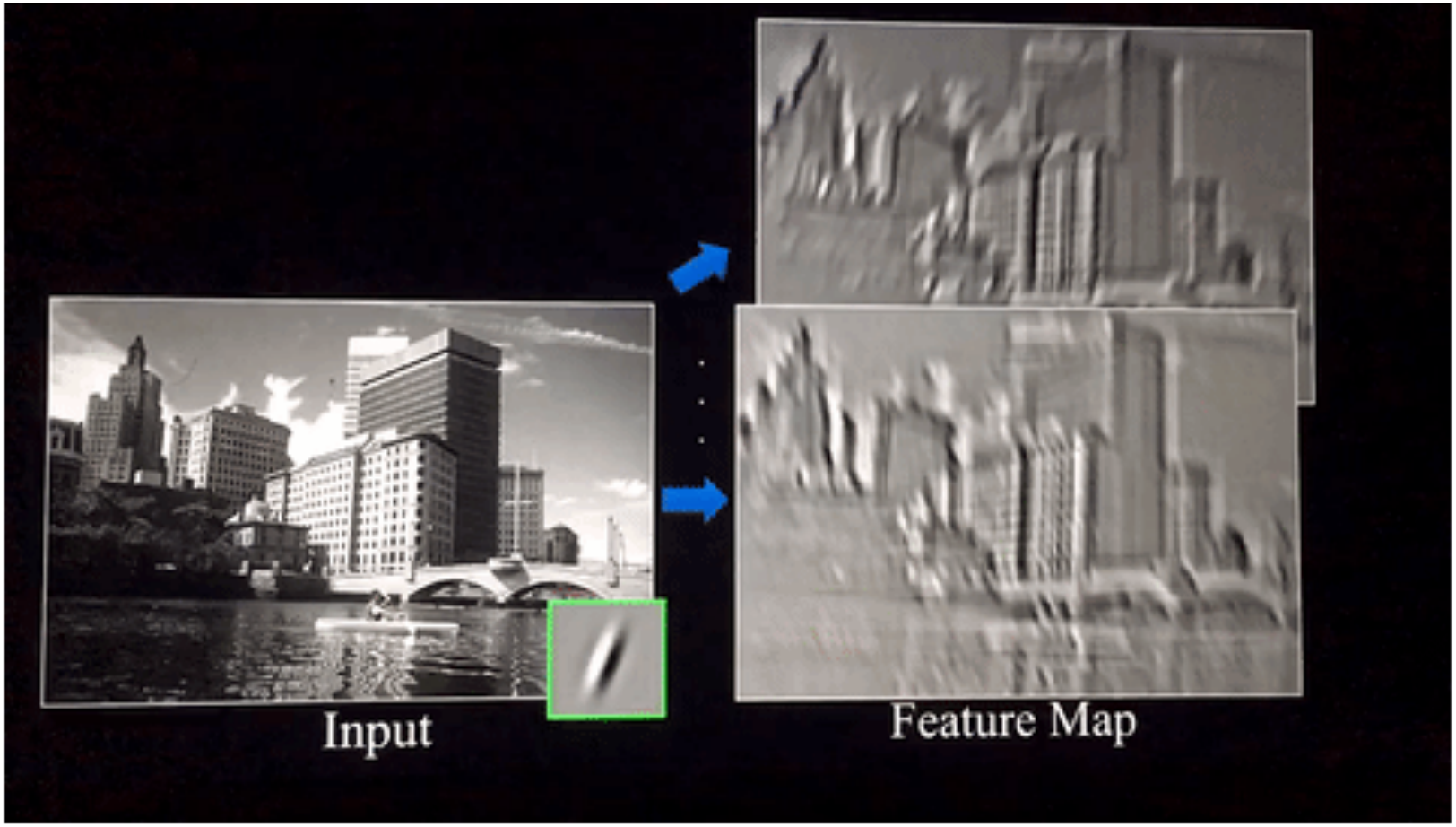
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

卷积神经网络 – 卷积过程

Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



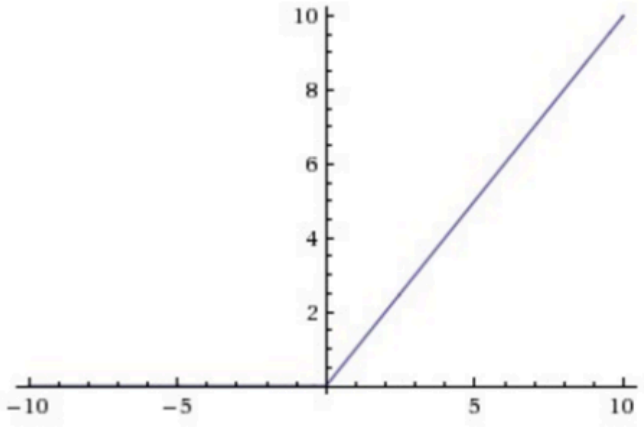
卷积神经网络 – 卷积例子



卷积神经网络 - ReLU (Rectified Linear Unit)

在每次的卷积操作后都使用了一个叫做 ReLU 的操作。ReLU 表示修正线性单元（Rectified Linear Unit），是一个非线性操作。它的输入如下所示：

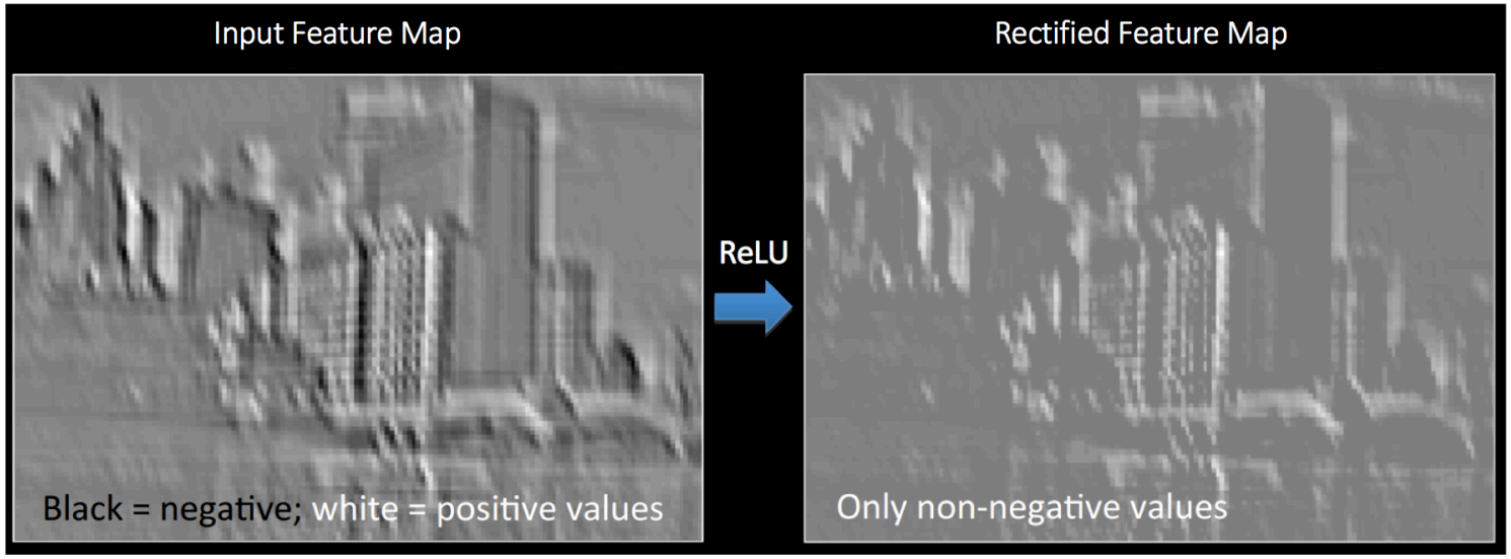
Output = Max(zero, Input)



ReLU 是一个元素级别的操作（应用到各个像素），并将特征图中的所有小于 0 的像素值设置为零。ReLU 的目的是在 ConvNet 中引入非线性，因为在大部分的时候我们希望 ConvNet 学习的实际数据是非线性的（卷积是一个线性操作——元素级别的矩阵相乘和相加，所以我们需要通过使用非线性函数 ReLU 来引入非线性。

卷积神经网络 - ReLU (Rectified Linear Unit)

其他激活函数，比如 tanh 或者 sigmoid 也可以用来替代 ReLU，但 ReLU 在大部分情况下表现是更好的。



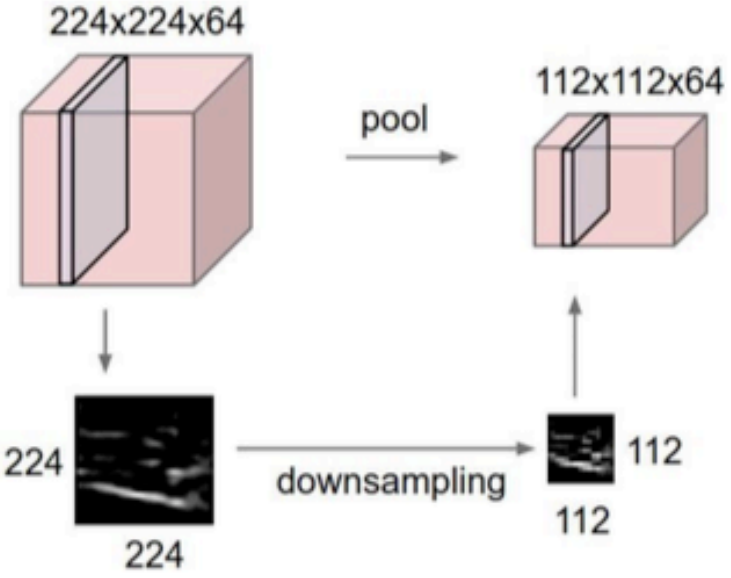
卷积神经网络 - 池化(Pooling)

在卷积神经网络中，经常会碰到池化操作，而池化层往往在卷积层后面，通过池化来降低卷积层输出的特征向量，同时改善结果（不易出现过拟合）。
也叫下采样（downsampling）.

目的是简化从卷积层输出的信息。

Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

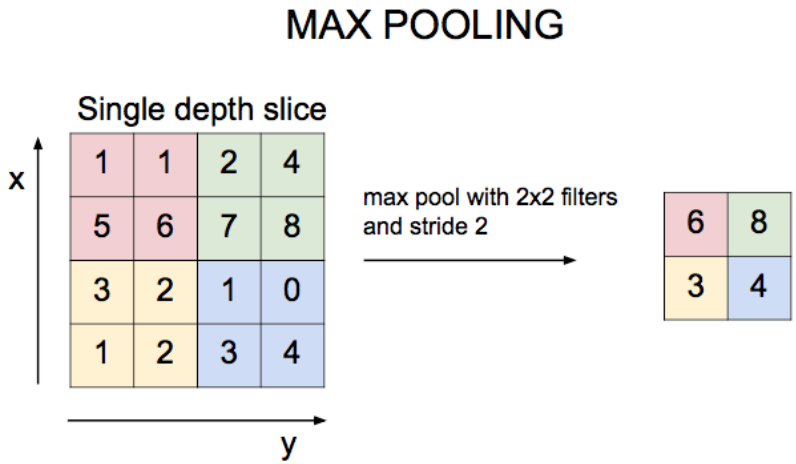


卷积神经网络 - 池化(Pooling)

为了描述大的图像，一个很自然的想法就是对不同位置的特征进行聚合统计，例如，人们可以计算图像一个区域上的某个特定特征的平均值 (或最大值)来代表这个区域的特征。

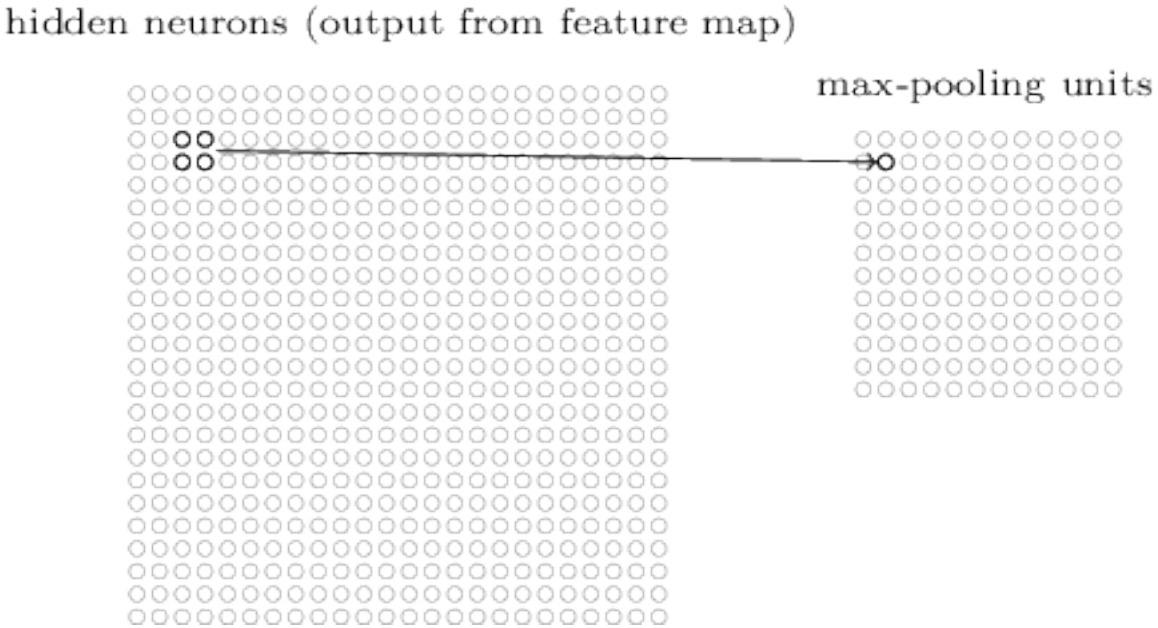
池化作用于图像中不重合的区域（这与卷积操作不同）。最大，平均....

- 使输入表示（特征维度）变得更小，并且网络中的参数和计算的数量更加可控的减小，因此，可以控制过拟合
- 使网络对于输入图像中更小的变化、冗余和变换具有不变性（输入的微小冗余将不会改变池化的输出——因为我们在局部邻域中使用了最大化/平均值的操作。



卷积神经网络 - 池化(Pooling)

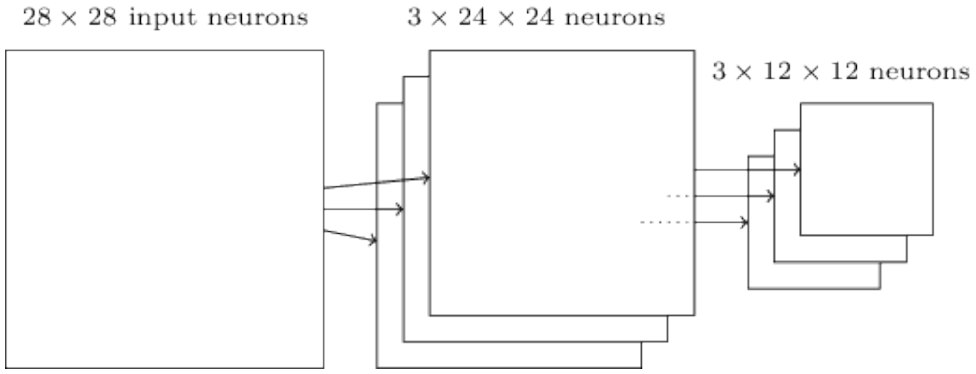
- 更具体一点，一个池化层把卷积层的输出作为其输入并且输出一个更紧凑(condensed)的特征映射。
- 在Max-Pooling中，这个神经元选择2×2区域里激活值最大的值。



- 注意卷积层的输出是24×24的，而池化后是12×12的。

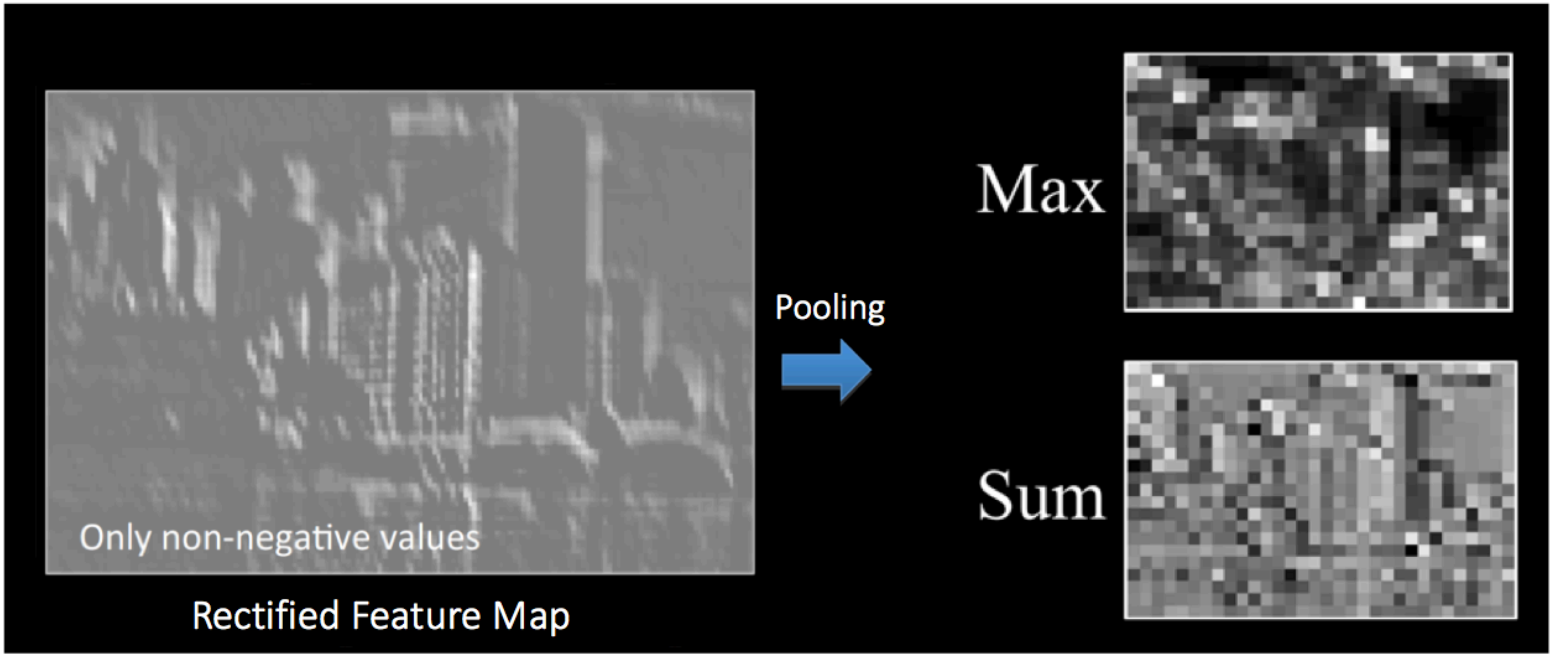
卷积神经网络 - 池化(Pooling)

- 卷积层通常会有多个特征映射。我们会对每一个特征映射进行pooling操作。因此，如果一个卷积层有3个特征映射，那么卷积加pooling后就如下图所示：



- 可以把池化看成神经网络关心某个特征在这个区域里是否出现。它忽略了这个特征出现的具体位置。
- 直觉上看，如果某个特征出现了，那么这个特征相对于其它特征的精确位置是不重要的【精确位置不重要，但是大致的位置是重要的，比如识别一个猫，两只眼睛和鼻子有一个大致的相对位置关系，但是在一个2×2的小区域里稍微移动一下眼睛，应该不太影响我们识别一只猫，而且它还能解决图像拍摄角度变化，扭曲等问题】。
- 而且一个很大的好处是池化可以减少特征的个数，因此减少了之后层的参数个数。

卷积神经网络 - 池化(Pooling)

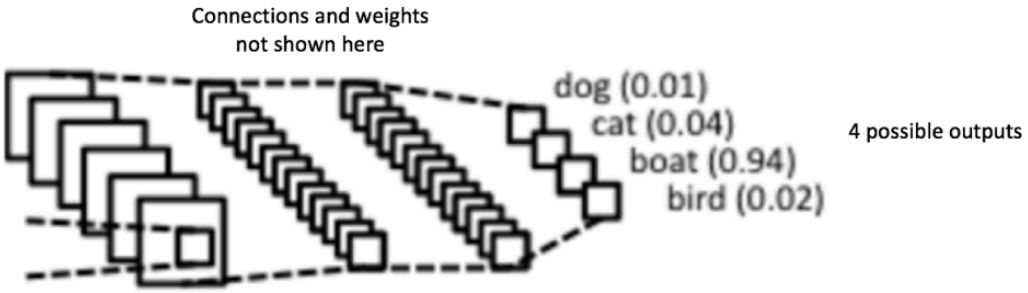


卷积神经网络 - 全连接(Fully Connected)

- 光栅化：图像经过池化-下采样后，得到的是一系列的特征图，而多层感知器接受的输入是一个向量。因此需要将这些特征图中的像素依次取出，排列成一个向量。
- 全连接层是传统的多层感知器，在输出层使用的是 softmax 激活函数（也可以使用其他像 SVM 的分类器）。
- “全连接（Fully Connected）”这个词表明前面层的所有神经元都与下一层的所有神经元连接。全连接层是一个分类器。

softmax $\sigma(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_{k=1}^{K} e^{\mathbf{z}_k}}$

- **softmax:如果某一个z_j大过其他z ,那这个映射的分量就逼近于1,其他就逼近于0;**
- **主要应用就是多分类，sigmoid函数只能分两类，而softmax能分多类.**



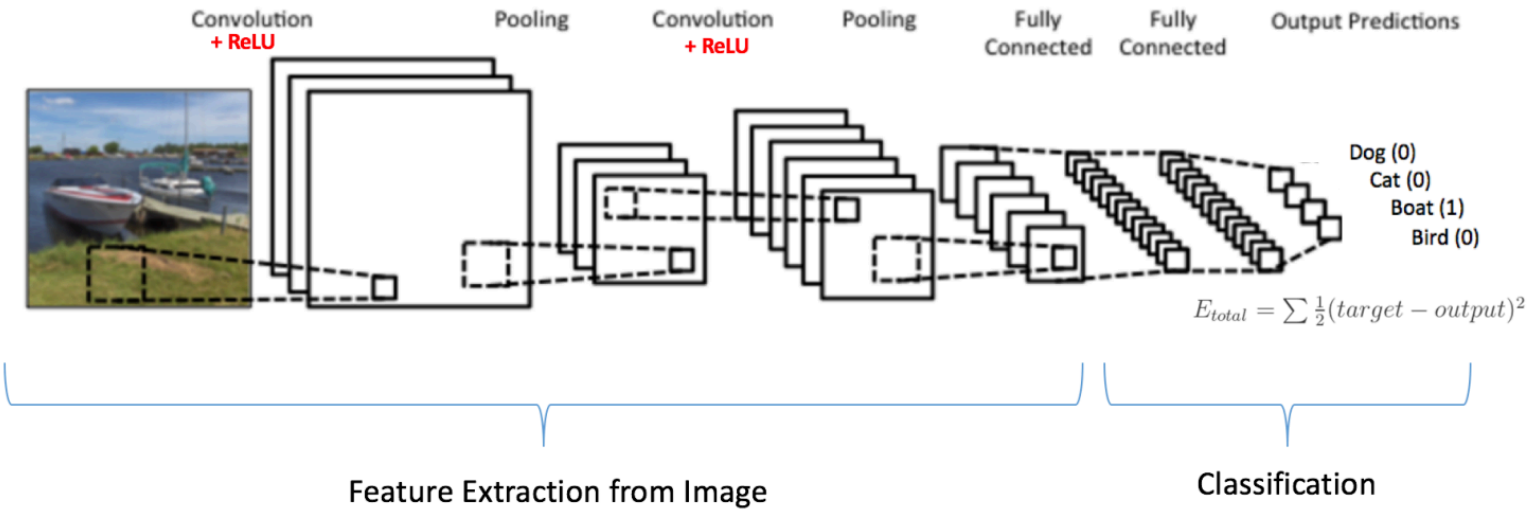
卷积神经网络 – 组合起来

卷积 + 池化层的作用是从输入图像中提取特征，而全连接层的作用是分类器。

注意在下面的图中，因为输入的图像是船，对于船这一类的目标概率是 1，而其他三类的目标概率是 0，即

- 输入图像 = 船
- 目标矢量 = [0, 0, 1, 0]

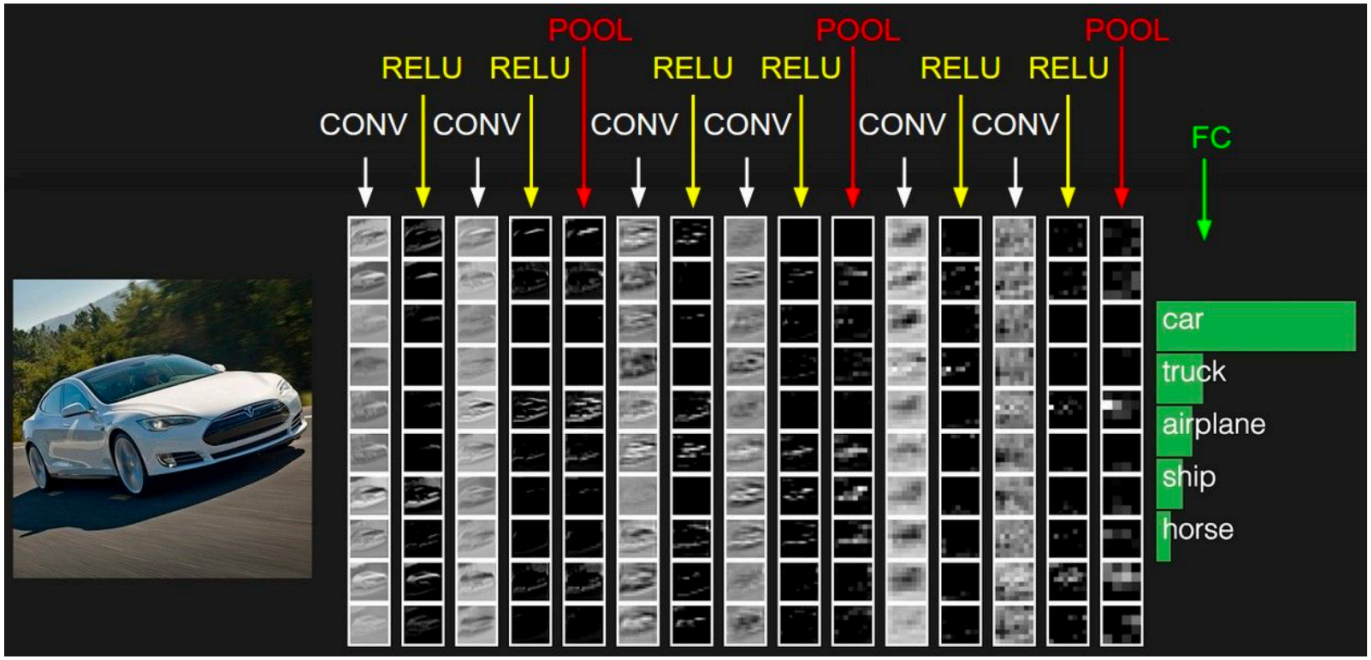
训练算法：反向误差传播（BP）



卷积神经网络 – 组合起来

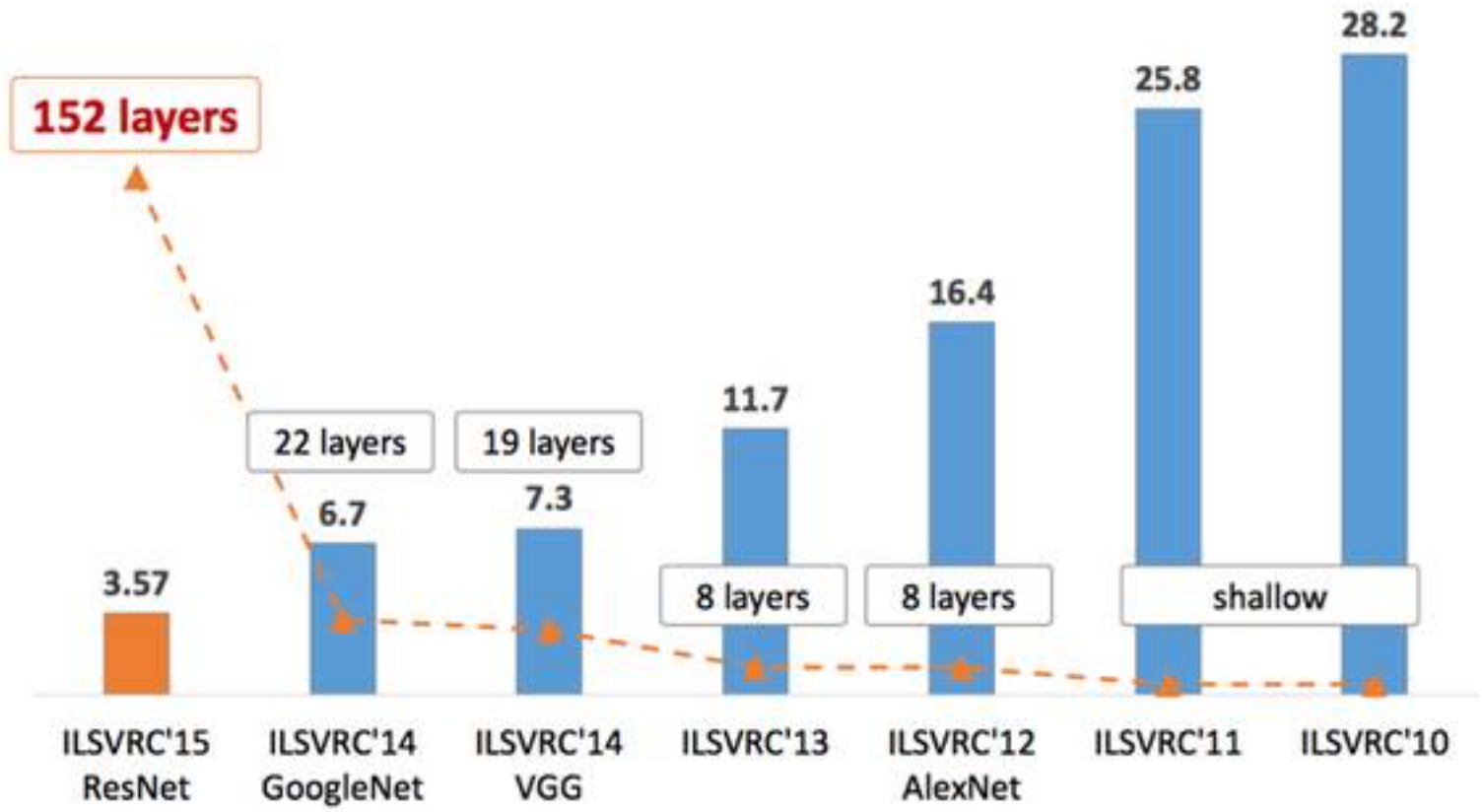
Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



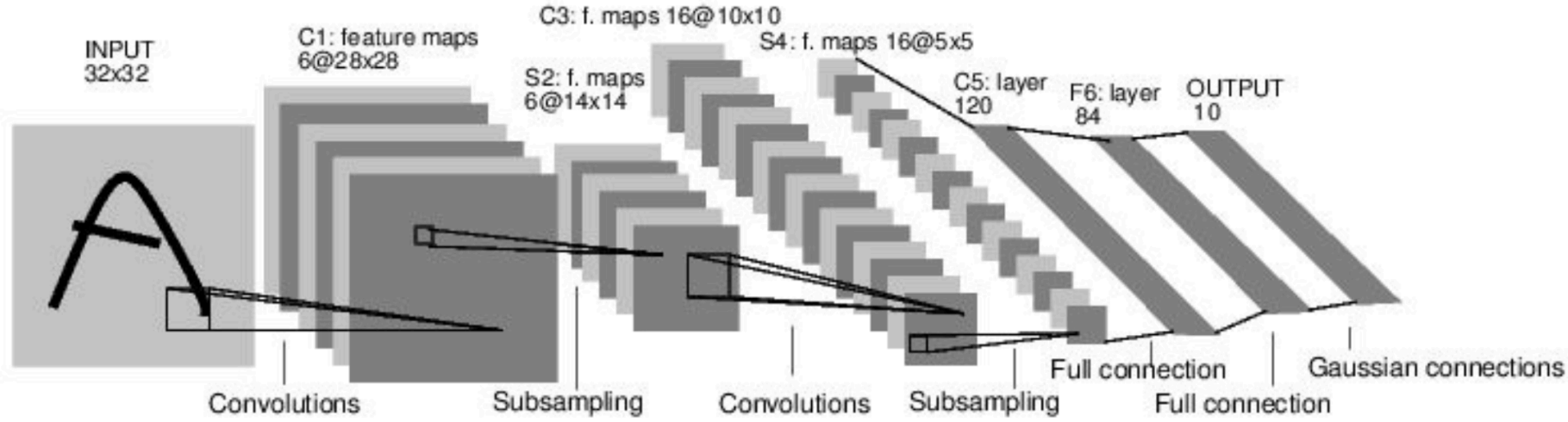
ILSVRC

ILSVRC（ImageNet Large Scale Visual Recognition Challenge）



Case Study: LeNet-5

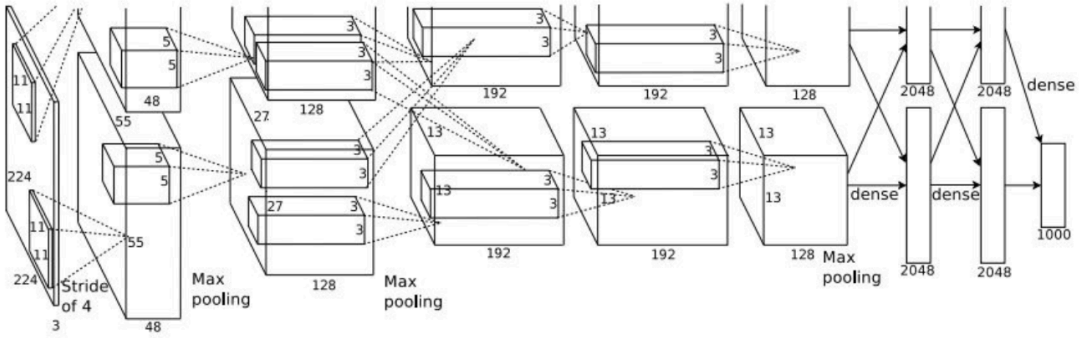
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]

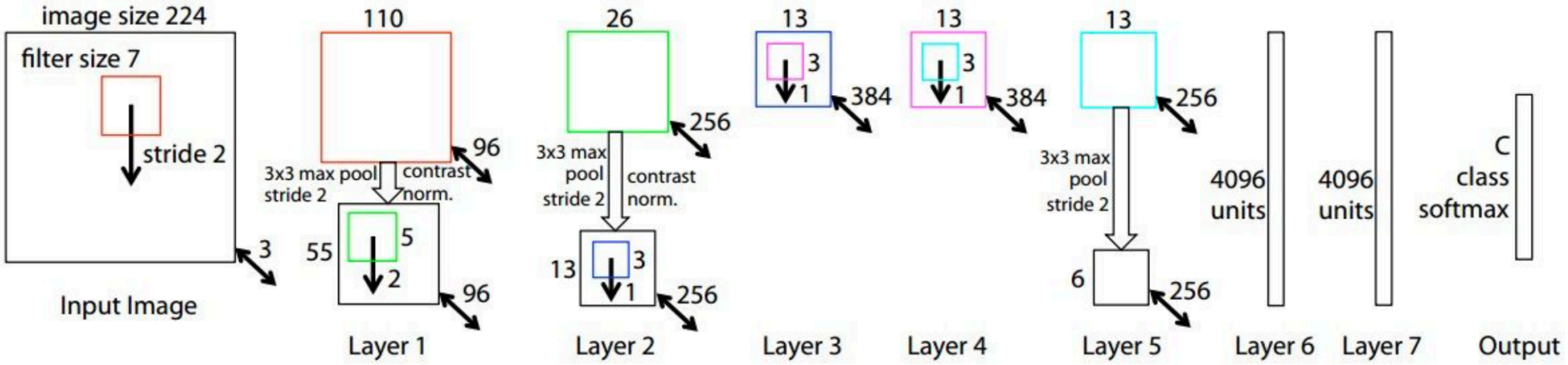


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Case Study: ZFNet

[Zeiler and Fergus, 2013]



AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

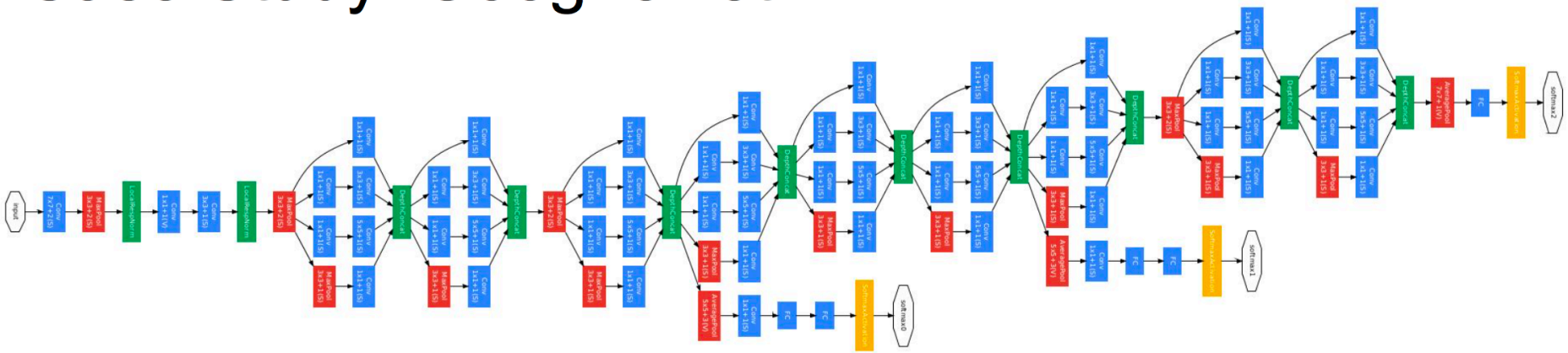
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

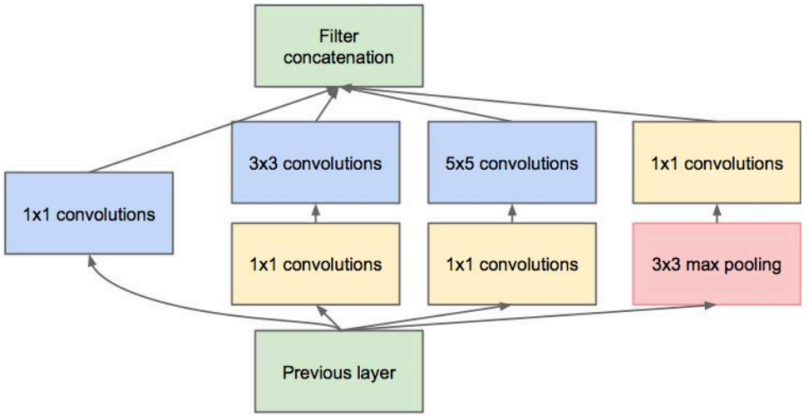
Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Case Study: GoogLeNet

[Szegedy et al., 2014]





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Additionally, this paper uses Average Pooling instead of Fully Connected layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much.

Case Study: GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

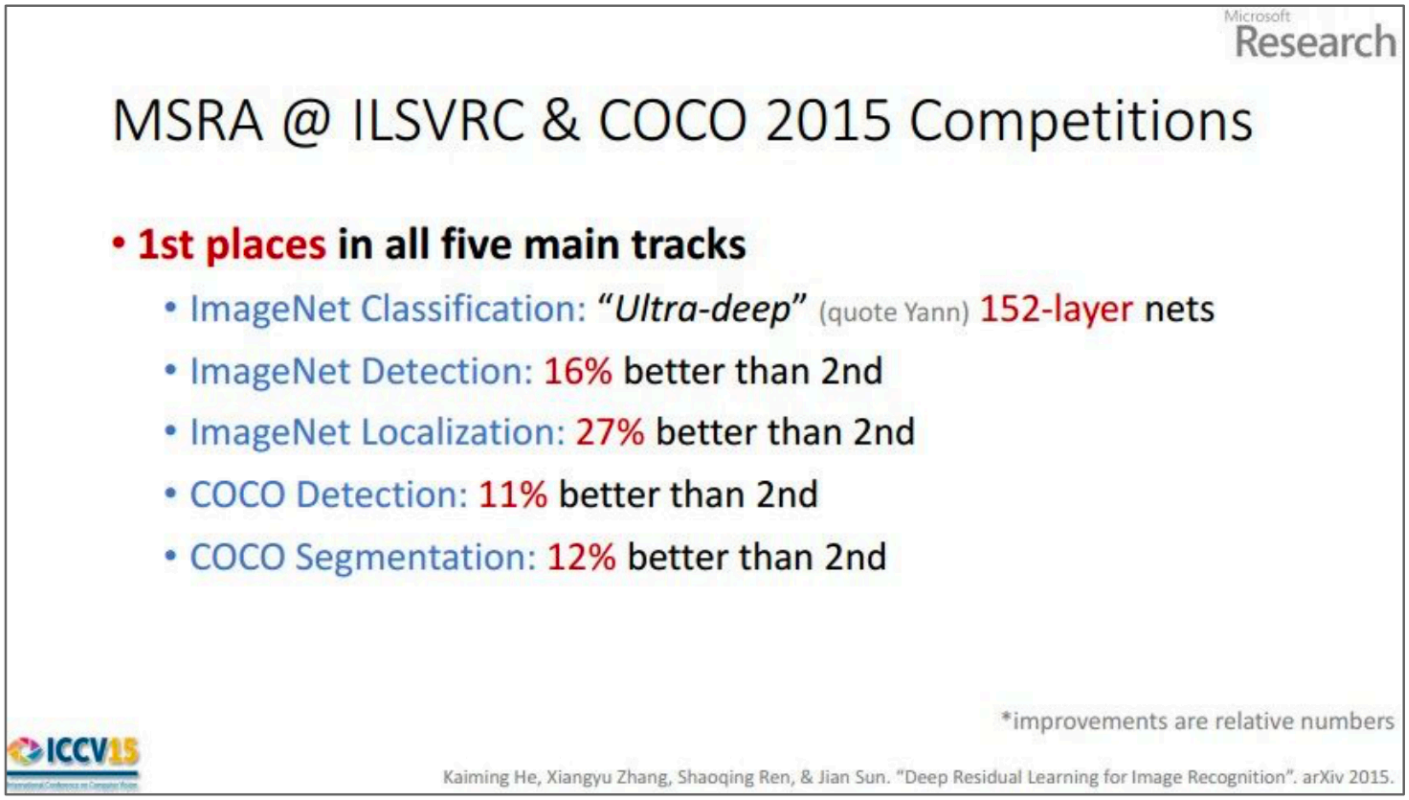
Fun features:

- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

Case Study: ResNet *[He et al., 2015]*

ILSVRC 2015 winner (3.6% top 5 error)



It features special skip connections and a heavy use of batch normalization.
The architecture is also missing fully connected layers at the end of the network.
ResNets are currently by far state of the art CNN models and are the default choice for using ConvNets in practice.

Case Study: ResNet *[He et al., 2015]*

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

深度学习最重要的特征：表示学习、联合优化

传统做法：

Feature Engineering

人工设计特征

学习分类

深度学习：

Representation learning

学习特征

学习分类

International Conference on Learning Representations (ICLR)

大数据、
高性能计算设备

深度学习 - Alpha Go

“We pass in the board position as a 19 × 19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network. ”

Alpha Go 由两套卷积神经网络和一棵蒙特卡洛树组成，其中一套被称作策略网络的卷积神经网络用于决定下一步落子可能的位置，另一套被称作值网络的卷积神经网络用于评估当前棋局获胜的概率。

- 围棋棋盘有19x19个格子，共是361个交叉点。每个交叉点有三种状态，可以用1表示黑子，-1表示白字，0表示无子，我们可以简化成361*3维的向量来表示一个棋盘的状态S。
- 棋手在状态S下走棋的结果记为b，不考虑无法落子的情况，b是一个361维度的向量，只有一个为1其余都是0。
- 分数socre=f(<S,b>),表示在S状态下，落子为b的胜率。这个socre函数算的有多准确，就说明AI机器的能力越强。

深度学习 - Alpha Go

最好的方法，就是在当前的棋局状态上遍历完整个棋局走势，把对手每一步所有可能走的棋都算一遍，遍历完后在当前状态哪一步能赢的机会多，这一步的分数就高。如今的计算机根本没有能力计算这么大的东西。这也是围棋为什么迟迟没有被AI搞定的原因。

棋面的数量，大约 250^{150}

我们不得以不简化计算socre的方法，这里就可以引出卷积神经网络了.

如果我们有很多人类选手的围棋对弈的棋局。这些棋局中每一个状态S，都会对应一个人类做出的落子b（361维度），这就组成了数量庞大的特征向量和目标向量。我们用<S, b>来表示。而且S是一个天然的二维向量，其实就可以看做一张图片（19x19）。每张图片对应一个目标向量，这不就是卷积神经分类的专长么。

如果我们的样本足够大，训练出的预测函数 P（S）不就很好的模拟出人下棋的选择了吗，预测出来的向量正好是人类在这个状态下走这361个点各自的概率。

深度学习 - Alpha Go

The input to the policy network is a 19 × 19 × 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 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

Output size:
(N – F + 2P) / stride + 1

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

深度学习 - Alpha Go

虽然不用全局遍历了，但是根据卷积神经网络的泛化能力，这种方法只能训练出和人类平均水平差不多的AI来，远远不能够和顶尖棋手对弈。

事实上，AlphaGo 比赛中运用了1920个cpu，280个GPU，其中的score函数策略非常复杂，不仅包含了P（S），里面仍然有通过其他方式得到的分数权重，这就涉及到蒙特卡洛搜索树（全局遍历简化版）和强化学习的一些评价函数。

Alpha Go 现在的改进：提高计算效率

没必要严格查找每一种棋面下的确切赢面，不妨把相似棋面的赢面，取个均值。

不妨把棋面看成一张图片，用卷积神经网络（CNN）提取层层抽象棋面特征。CNN的输出，是棋面特征的编码。

棋面特征的编码是一个数字向量，数字本身没有明确意义，但是相似棋局的数字向量很邻近。

新的Alpha Go中，特征编码的数量是 $5 * 10^5$，远远小于 250^{150}。换而言之， 250^{150} 种棋面，被 CNN 聚类成 5 * 10^5 种相似棋面。

深度学习

典型的深度学习模型就是很深层的神经网络

（例如微软研究院2015年在ImageNet竞赛获胜使用152层网络）

提升模型复杂度 → 提升学习能力

- 增加隐层神经元数目 (模型宽度)
- 增加隐层数目 (模型深度)

增加隐层数目比增加隐层神经元数目更有效

不仅增加了拥有激活函数的神经元数，还增加了激活函数嵌套的层数

提升模型复杂度 → 增加过拟合风险；
增加训练难度

- 过拟合风险：使用大量训练数据
- 训练困难：使用若干启发式诀窍

误差梯度在多隐层内传播时，往往会发散而不能收敛到稳定状态，因此，难以直接用经典BP算法训练

深度学习常用软件包

- Tensorflow
- CAFFE (Berkeley)
- Torch (Lua), PyTorch
- Theano (Python)
- MXNET (Amazon)

……

Reading List

1. Deep learning, Nature, 2015.
2. Human-level control through deep reinforcement learning (Playing games), Nature, 2015
3. Mastering the game of Go with deep neural networks and tree search (Alpha Go), Nature, 2016
4. On the Origin of Deep Learning, 2017 (history)

CNN:

1. ImageNet Classification with Deep Convolutional Neural Networks, 2012, NIPS
2. Deep residual learning for image recognition (152 Layers), CVPR, 2015

RNN:

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, 2014 (第一篇 seq2seq)
Sequence to sequence learning with neural networks. NIPS. 2014

热点研究方向：

图像识别，自然语言处理，语音识别，对话机器人，生成对抗网络，安全…