深度学习用于计算机视觉
Deep Learning for Computer Vision

参考资料

- 吴恩达：
  deeplearning.ai
  网易云课堂
- 李飞飞：CS231n课程

第5章 深度学习用于计算机视觉
第7章 高级的深度学习最佳实践

教材代码： https://github.com/fchollet/deep-learning-with-python-notebooks

本章内容

- CNN（Convolutional Neural Network 卷积神经网络）
- 在小型数据集上从头开始训练一个CNN
- 使用预训练的CNN （迁移学习）
- CNN的可视化
- CNN Applications
  - Classification based on CNN
  - Object detection based on CNN

本章内容

- CNN（Convolutional Neural Network 卷积神经网络）
- 在小型数据集上从头开始训练一个CNN
- 使用预训练的CNN （迁移学习）
- CNN的可视化
- CNN Applications
  - Classification based on CNN
  - Object detection based on CNN

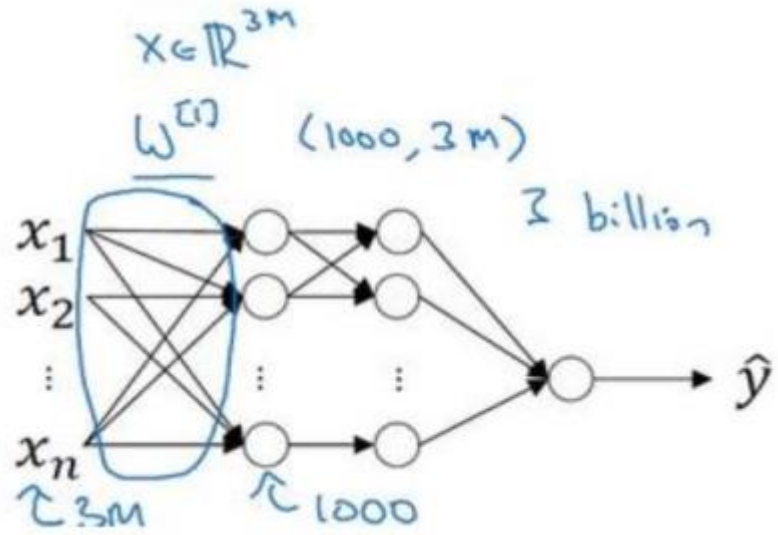
Why CNN for Image?

64x64 × 3

Cat? (0/1)

12288

1000 × 1000 × 3
= 3 million



- Huge Parameters !!!
- Can the network be simplified by considering the properties of images to reduce the amount of parameters?

Why CNN for Image?

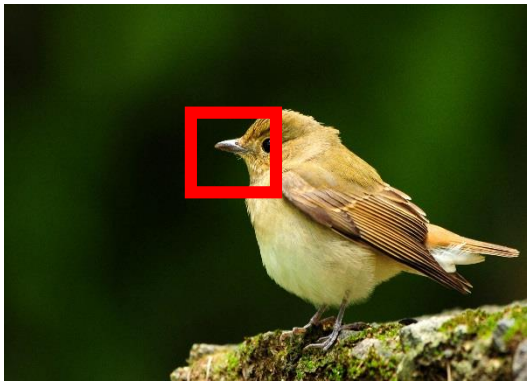
Property 1: Some patterns are much smaller than the whole image

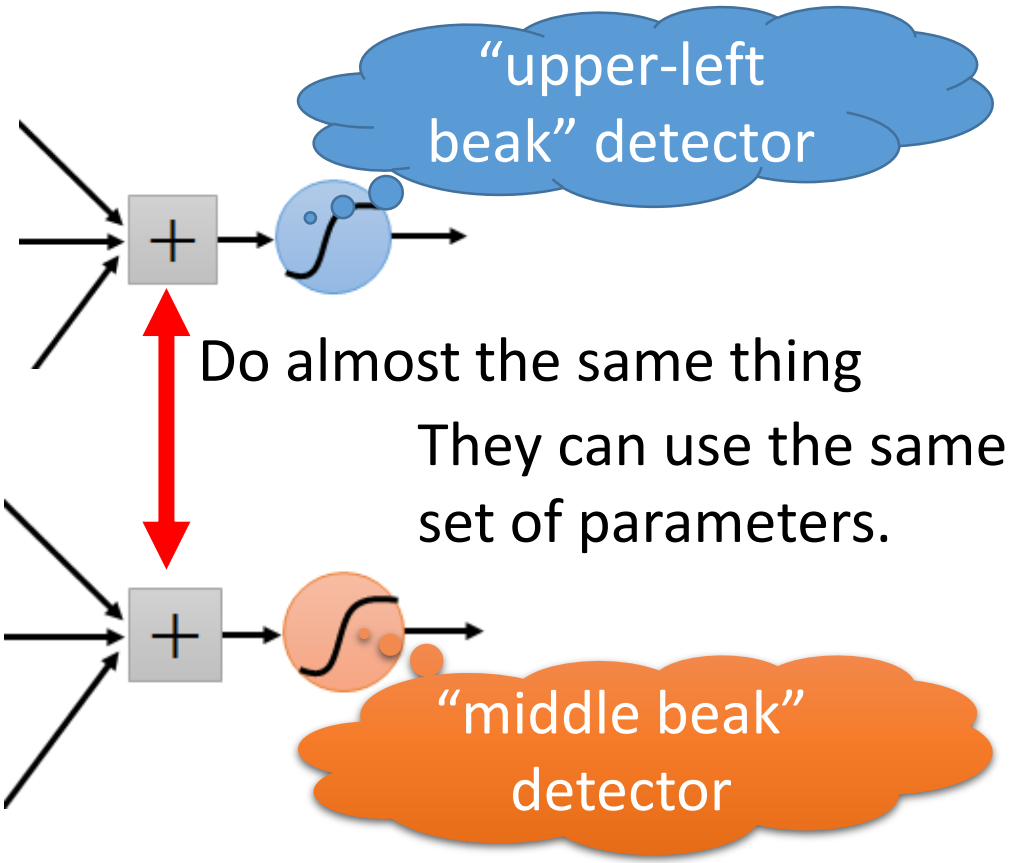
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters

Why CNN for Image

Property 2: The same patterns appear in different regions.





Why CNN for Image

Property 3: Subsampling the pixels will not change the object

bird

subsampling

bird

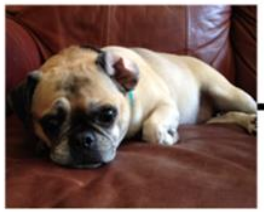
We can subsample the pixels to make image smaller

→ Less parameters for the network to process the image

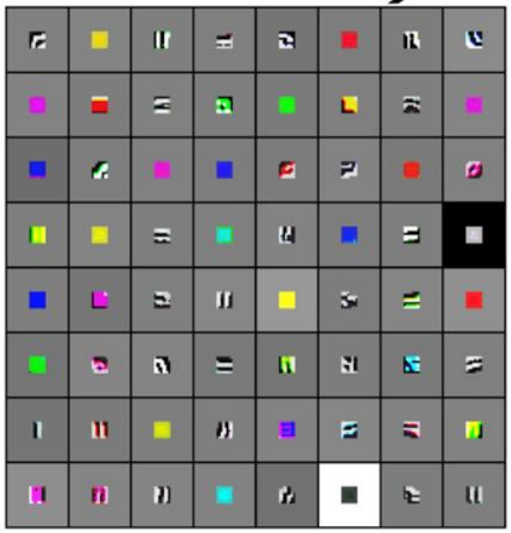
Preview

[Zeiler and Fergus 2013]

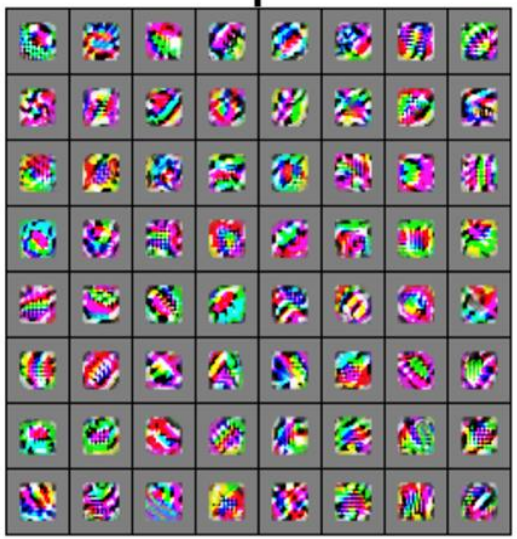
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



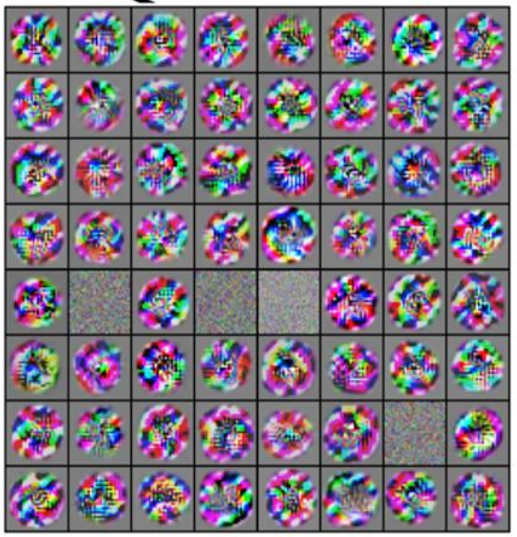
Low-level features → Mid-level features → High-level features → Linearly separable classifier



VGG-16 Conv1_1



VGG-16 Conv3_2



VGG-16 Conv5_3

一个简单的卷积神经网络示例

- 使用卷积神经网络对MNIST数字进行分类
  - 用密集连接网络，测试精度为97.8%（这个任务我们在第2章）。
- 这个简单卷积神经网络的测试精度达到了99.3%，将错误率降低了68%（相对比例）。

代码清单 5-1　实例化一个小型的卷积神经网络

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

为什么效果这么好？

代码清单 2-2　网络架构

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
network.add(layers.Dense(10, activation='softmax'))
```

- 对比参数量：

CNN网络：

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 13, 13, 32)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 5, 5, 64)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                36928
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

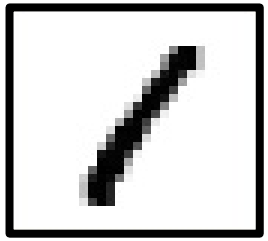
数据集：60 000张训练图像和10 000张测试图像，每张灰度图像为：28像素×28像素。

故，**全连接神经网络**的参数量：约40万个参数

$28 \times 28 \times 512+512+512 \times 10+10 = 407,050$

CNN的参数量少，且测试性能好，为什么？
观察CNN的结构：conv2d层和max-pooling2d层

The whole CNN



Convolution

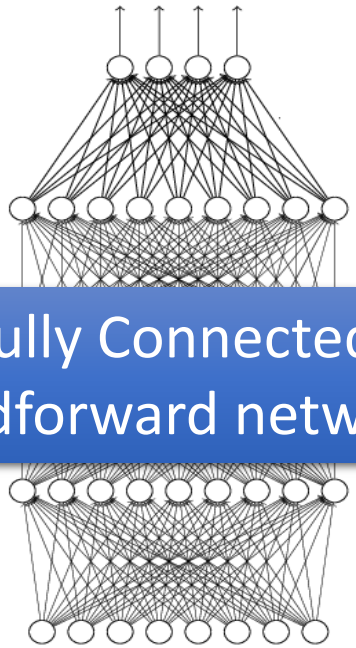
Max Pooling

Convolution

Max Pooling

Can repeat many times

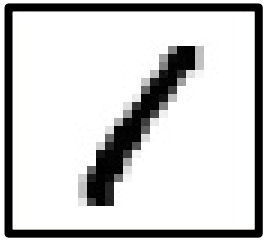
Flatten



Fully Connected Feedforward network

1 0 0 0 ……

The whole CNN



Property 1

- Some patterns are much smaller than the whole image

Property 2

- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object

Convolution

Max Pooling

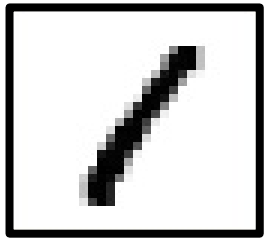
Convolution

Max Pooling

Can repeat many times

Flatten

The whole CNN



Convolution

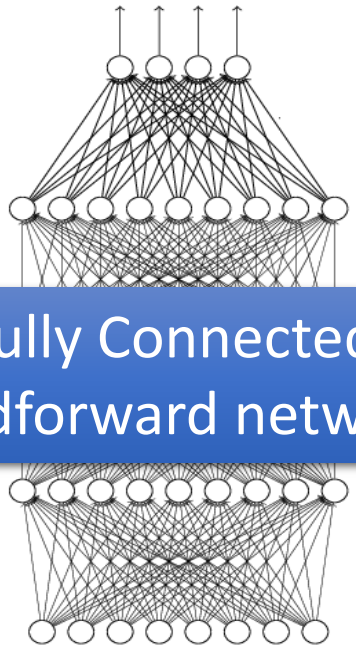
Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten



Fully Connected Feedforward network

1 0 0 0 ……

CNN – Convolution

Those are the network parameters to be learned. Now, assume the filter found the best parameters.

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

Matrix

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

Filter 2

Matrix

⋮

Property 1

Each filter detects a small pattern (3 x 3).

CNN – Convolution

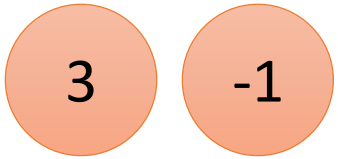
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=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

6 x 6 image



CNN – Convolution

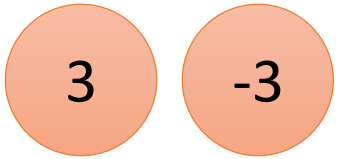
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

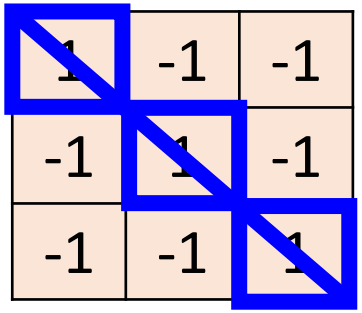
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



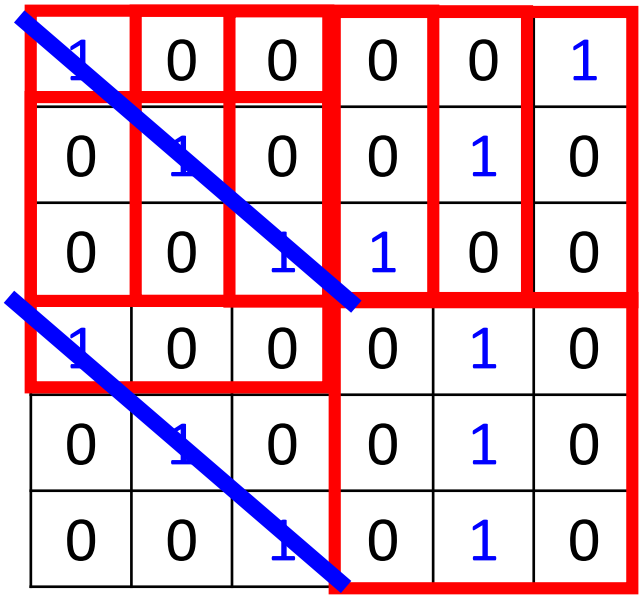
We set stride=1 below

CNN – Convolution

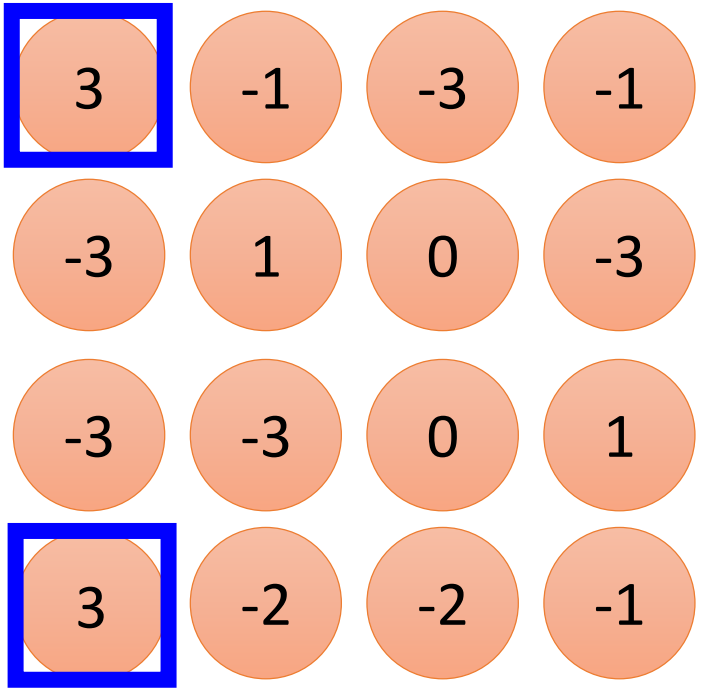


Filter 1

stride=1



6 x 6 image



Property 2

CNN – Convolution with more filters

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

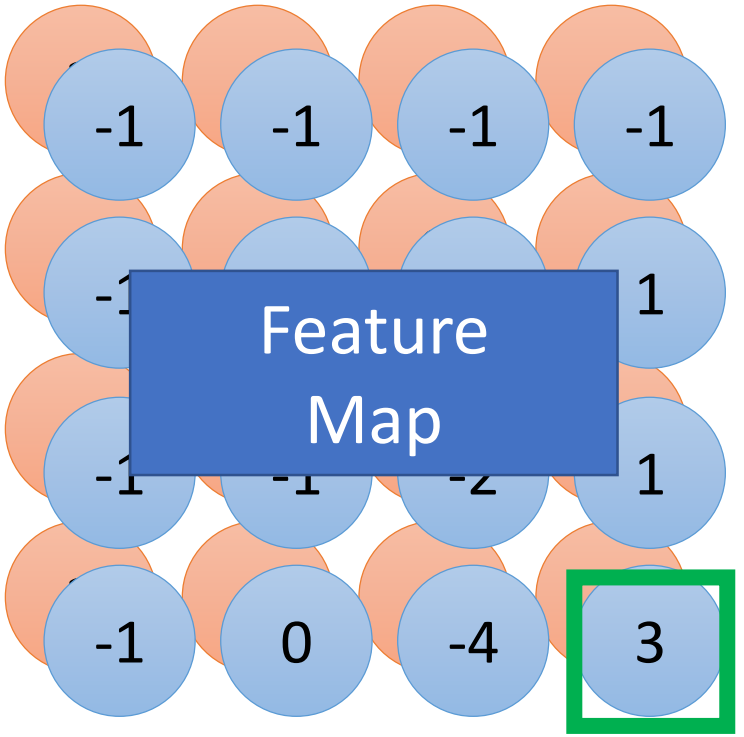
Filter 2

stride=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

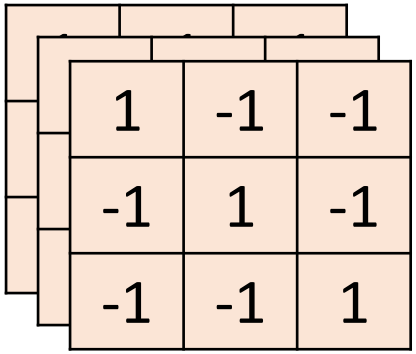
6 x 6 image

Do the same process for every filter

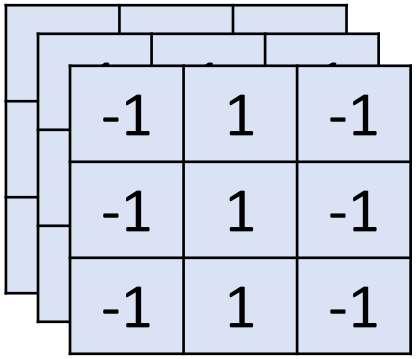


4 x 4 image

CNN – Colorful image

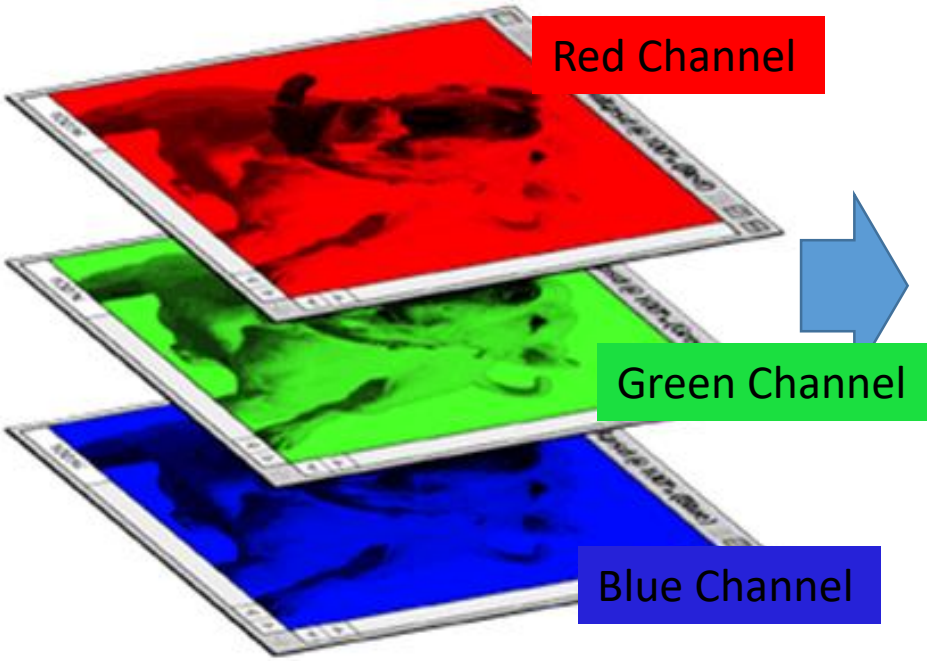


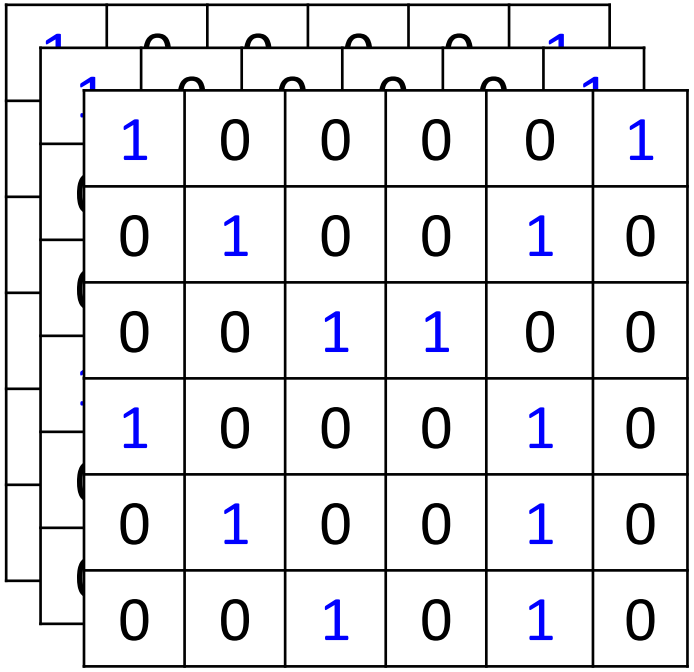
Filter 1



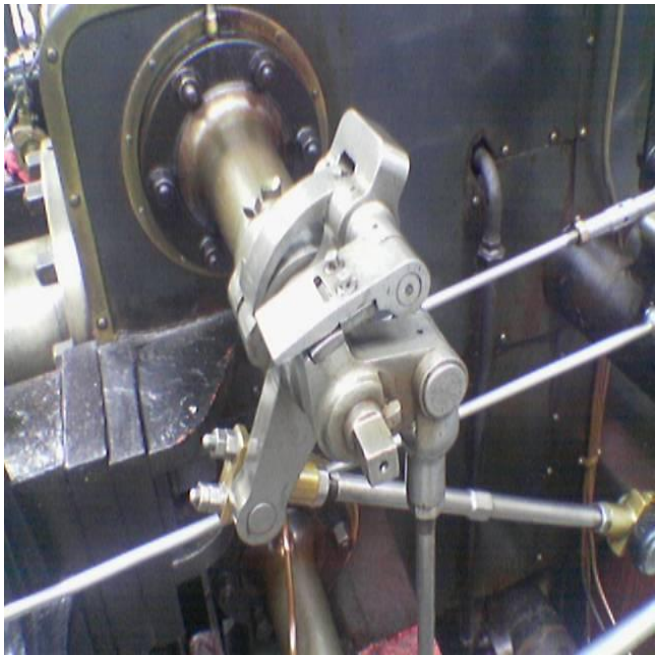
Filter 2

Colorful image





Convolving “filters” is not a new idea



Sobel operator:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

Convolution without training

input

Kernel for blurring

0.0625	0.125	0.0625
0.125	0.25	0.125
0.0625	0.125	0.0625

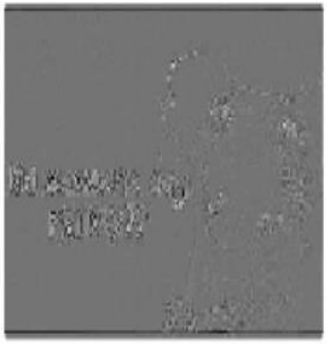
tf.nn.conv2d

output

input

sharpen

edge



top sobel

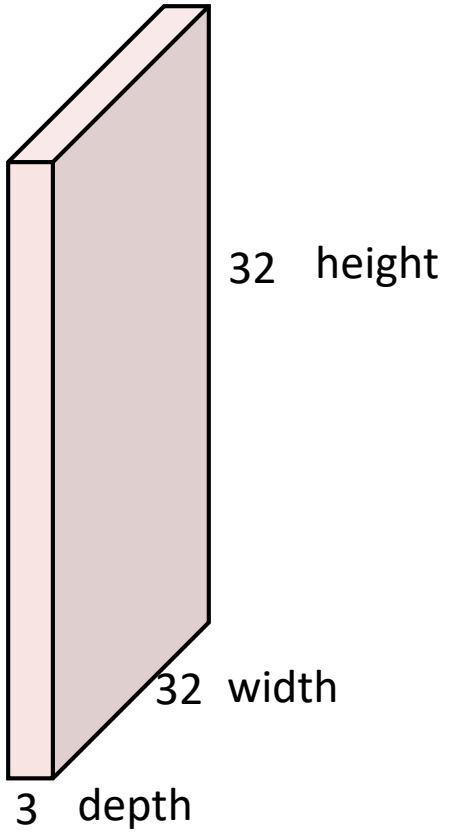
emboss

Don’t hard-code the values of your kernels.

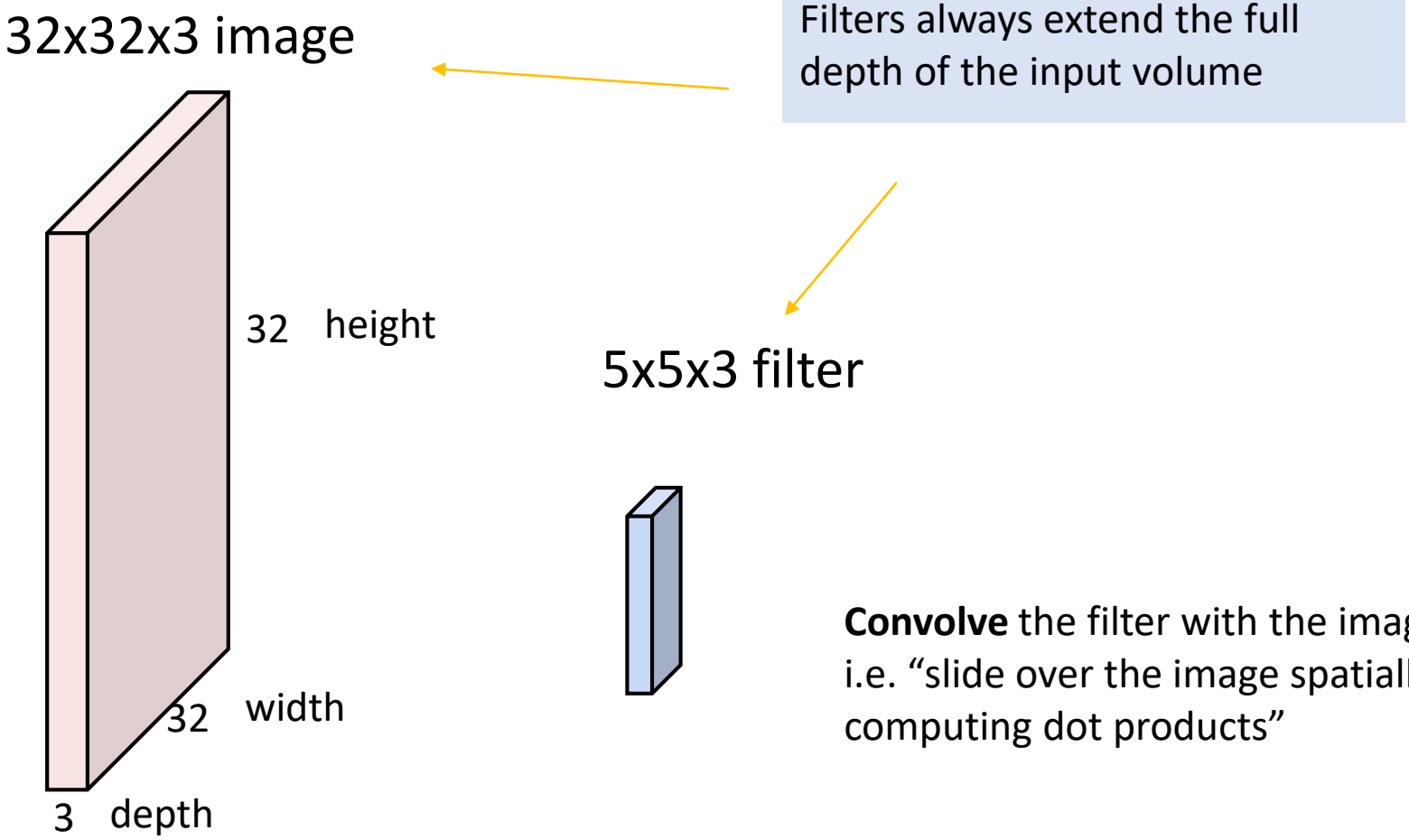
Learn the optimal kernels through training!

Convolution Layer

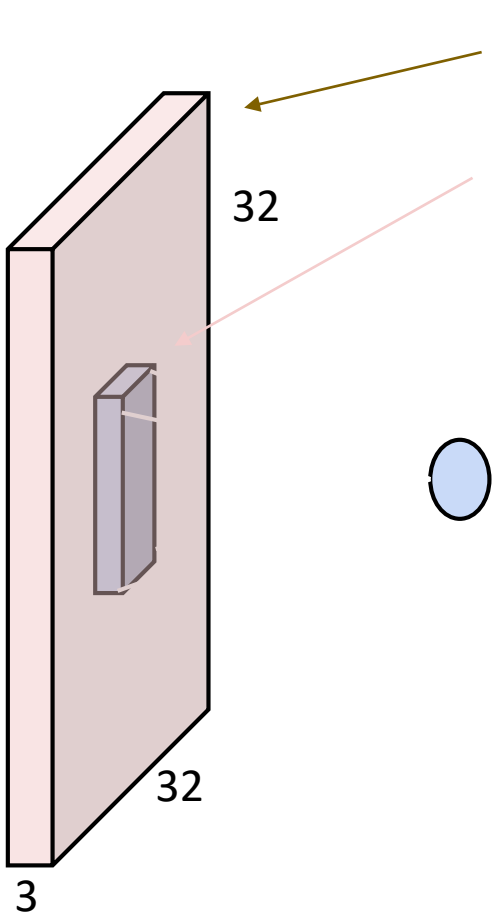
32x32x3 image



Convolution Layer



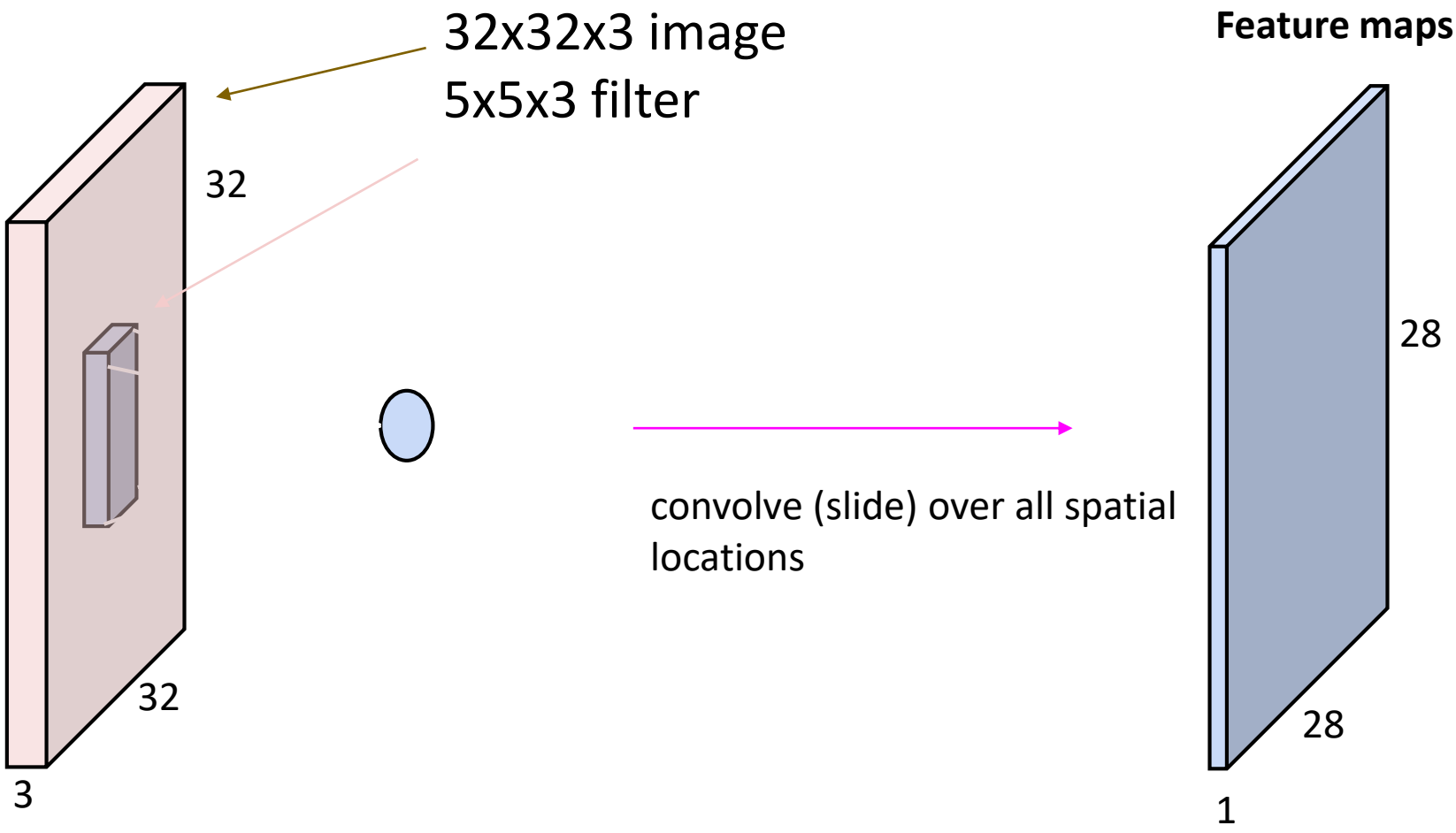
Convolution Layer



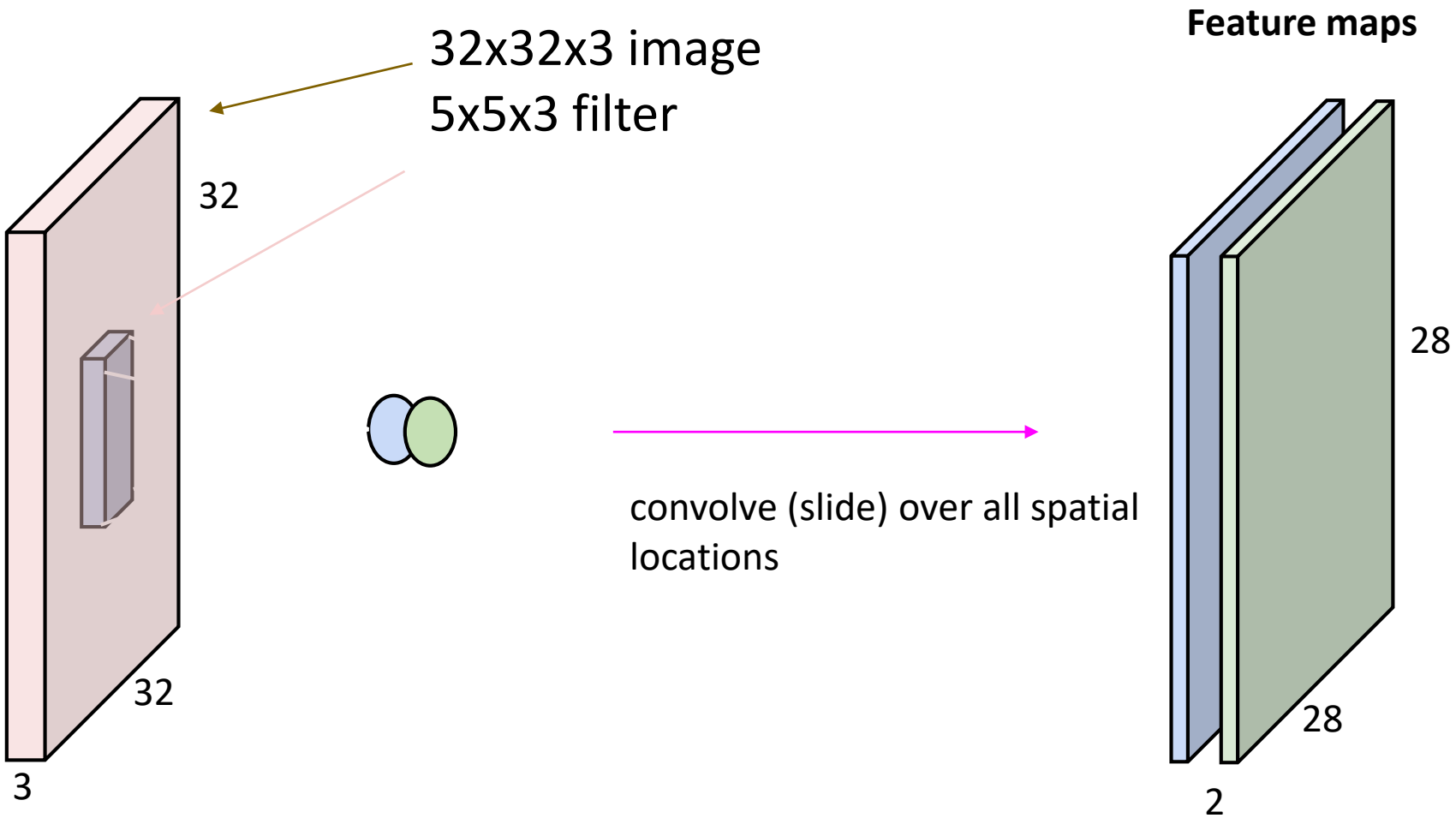
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer

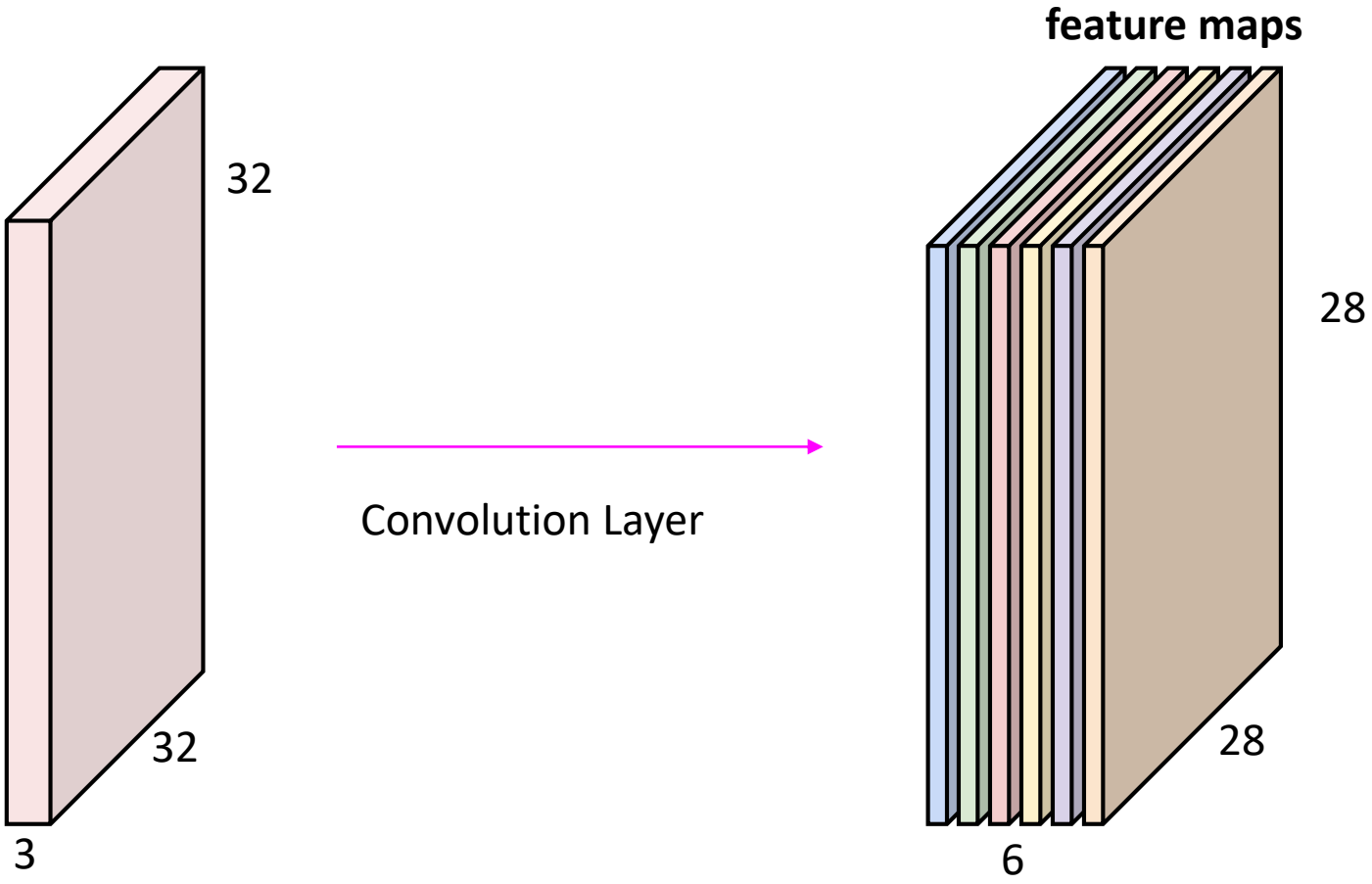


Convolution Layer



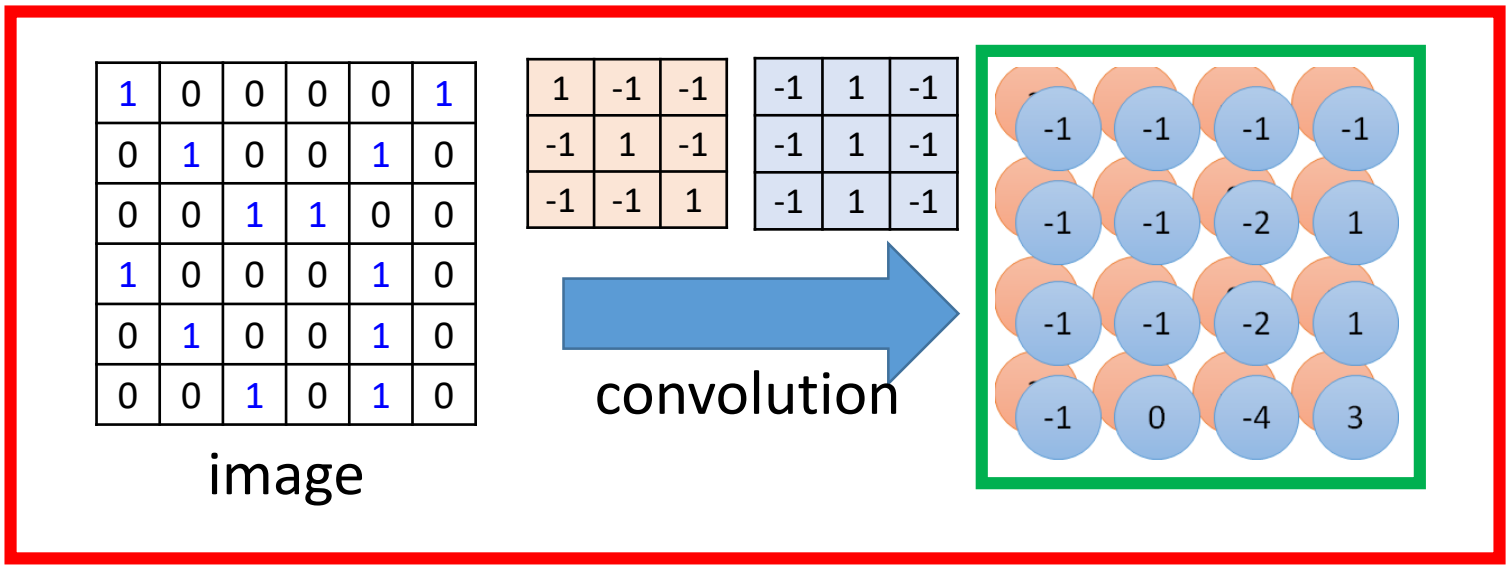
consider a second, green filter

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

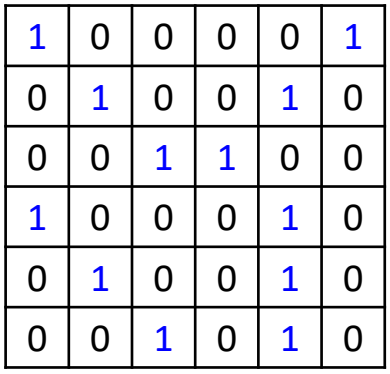


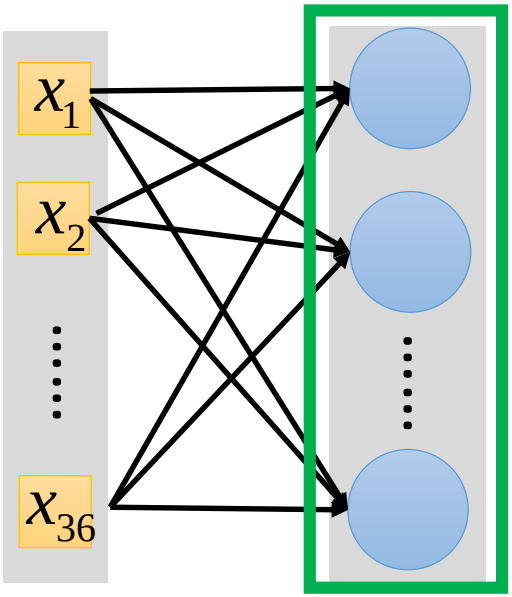
We stack these up to get a new "image" of size 28x28x6!

Convolution v.s. Fully Connected

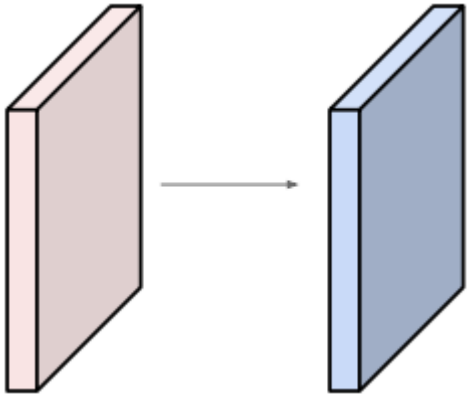


Fully-connected





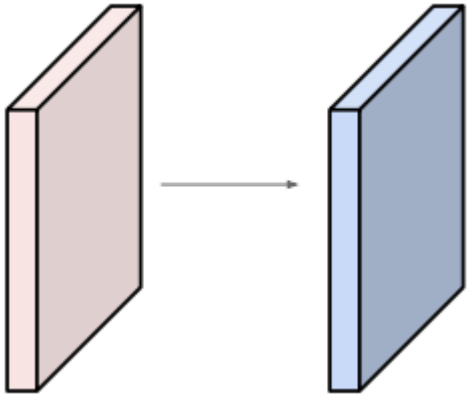
课堂练习



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

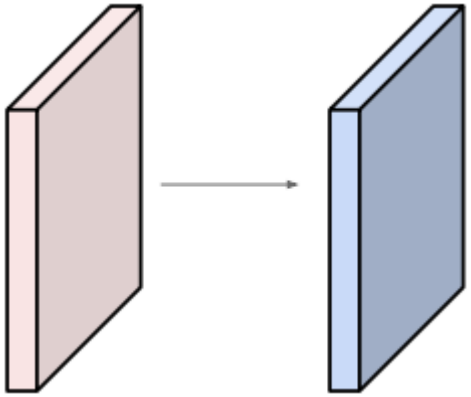
课堂练习



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

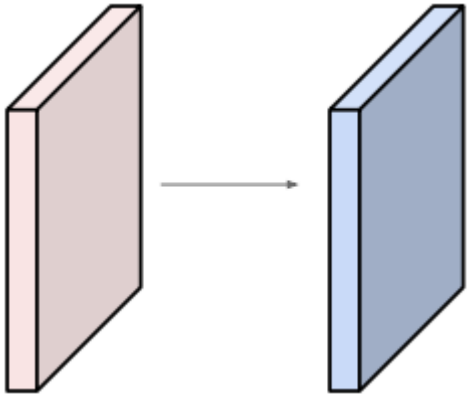
课堂练习



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

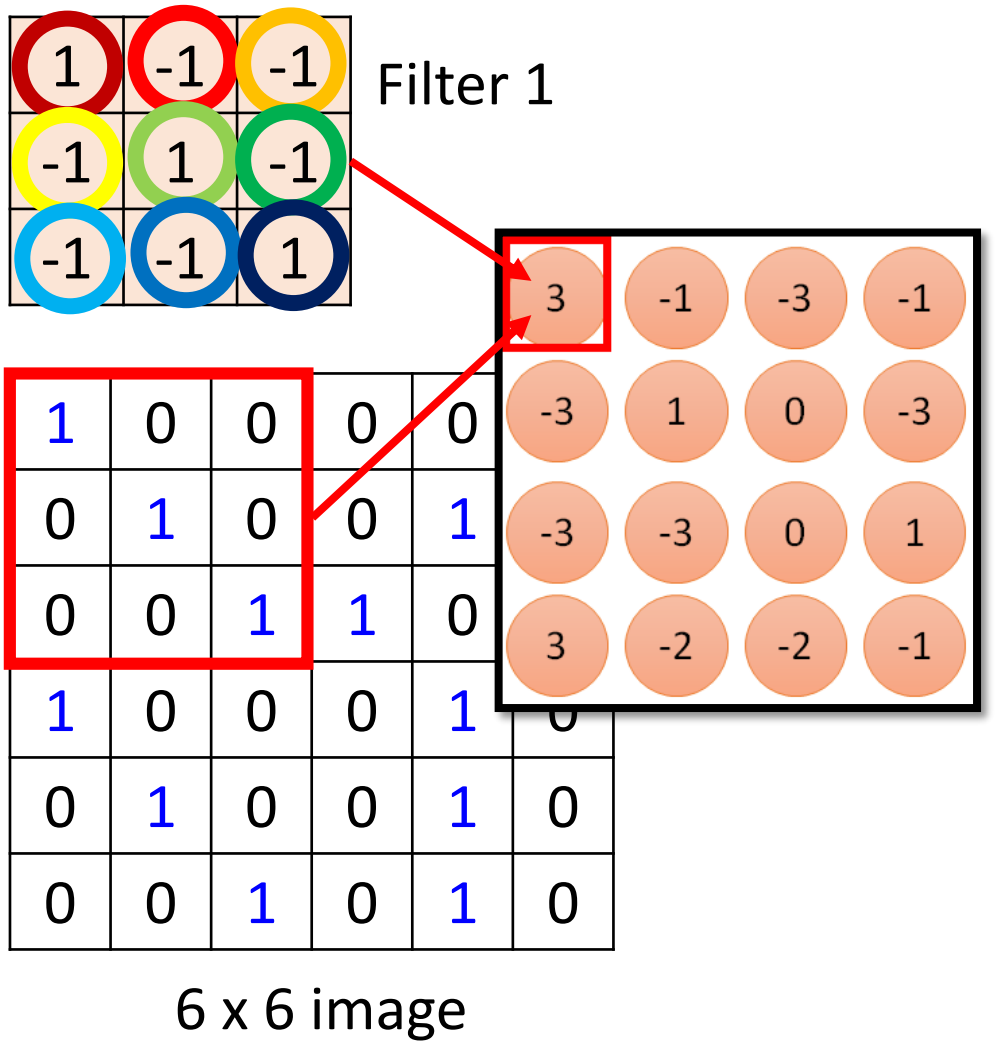
Number of parameters in this layer?

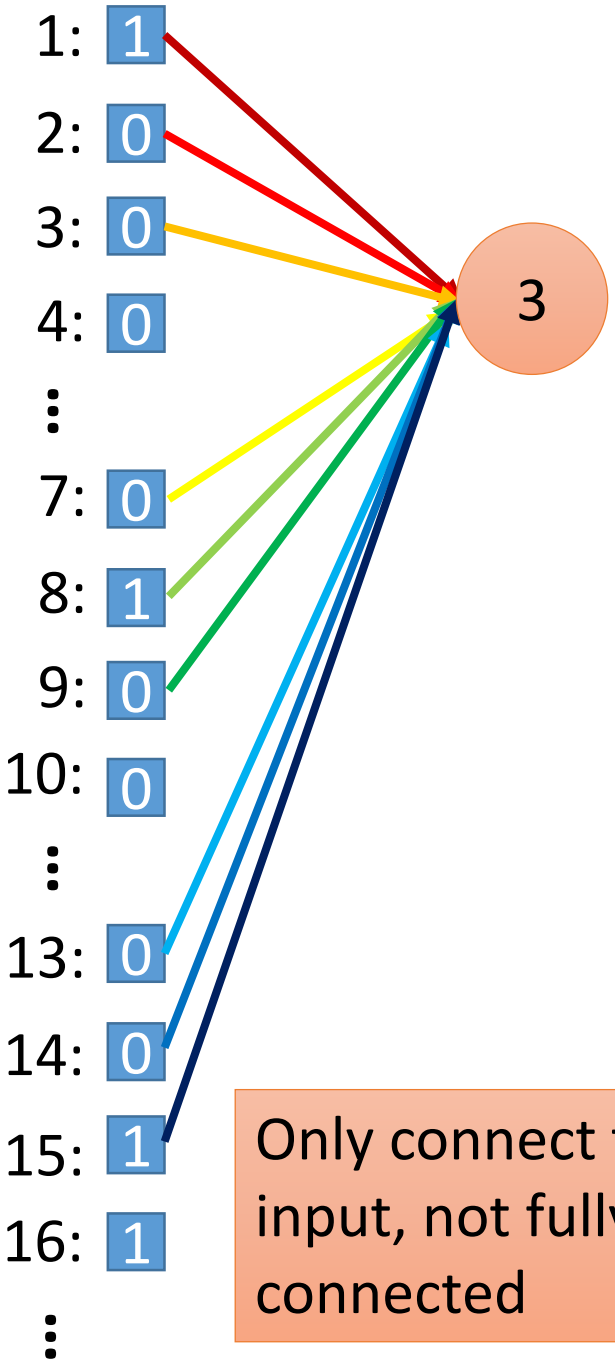
课堂练习



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

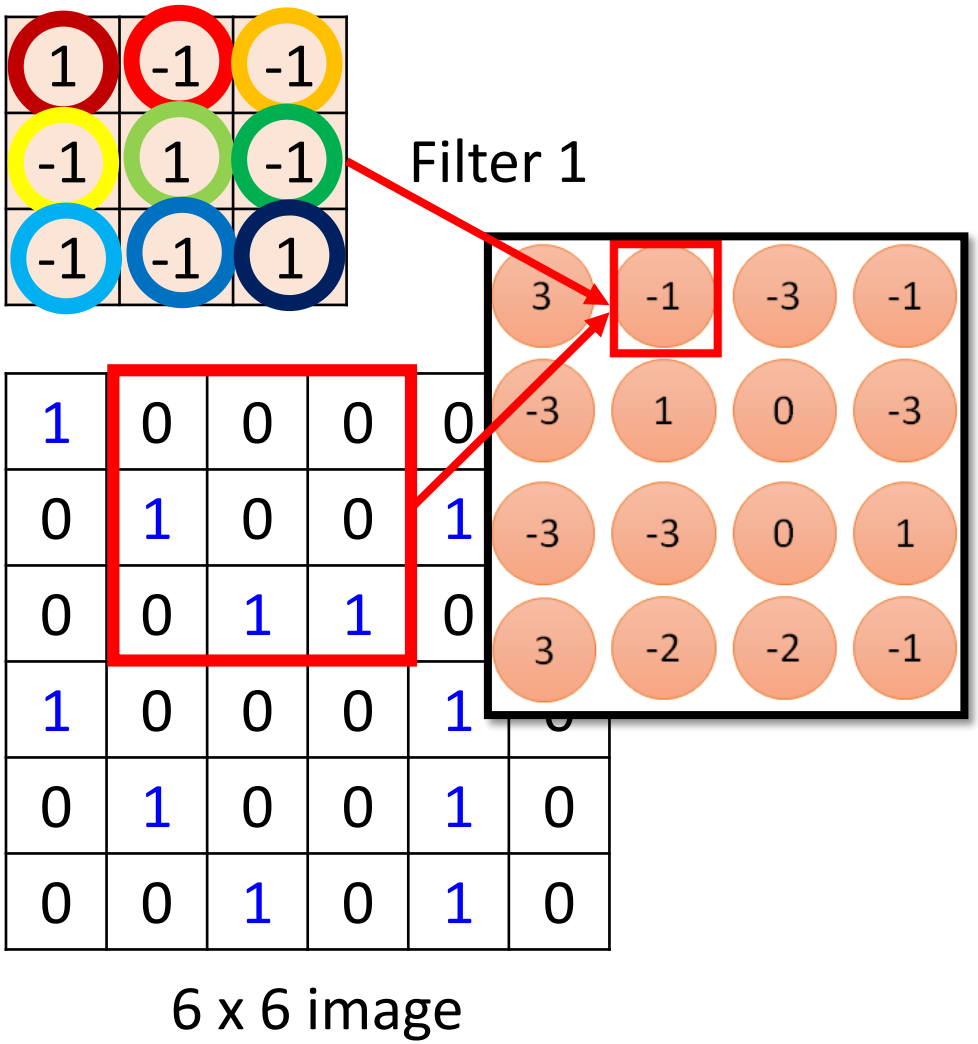
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

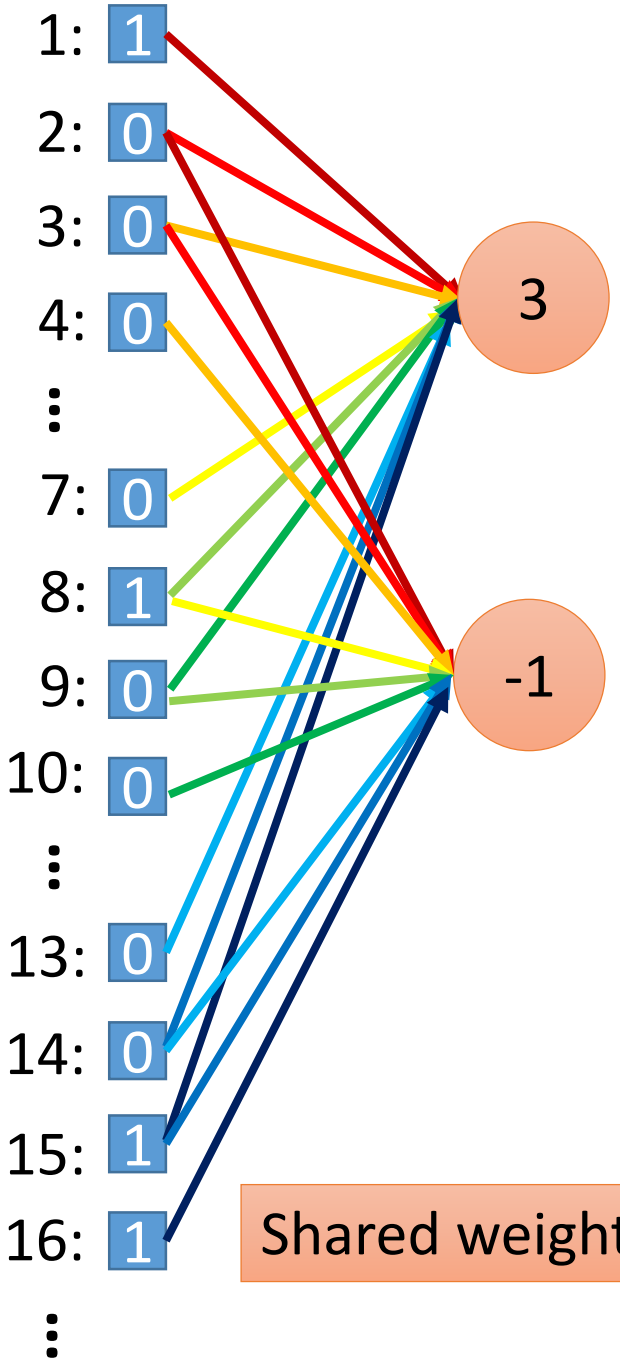




Sparsity of connections: Less parameters!

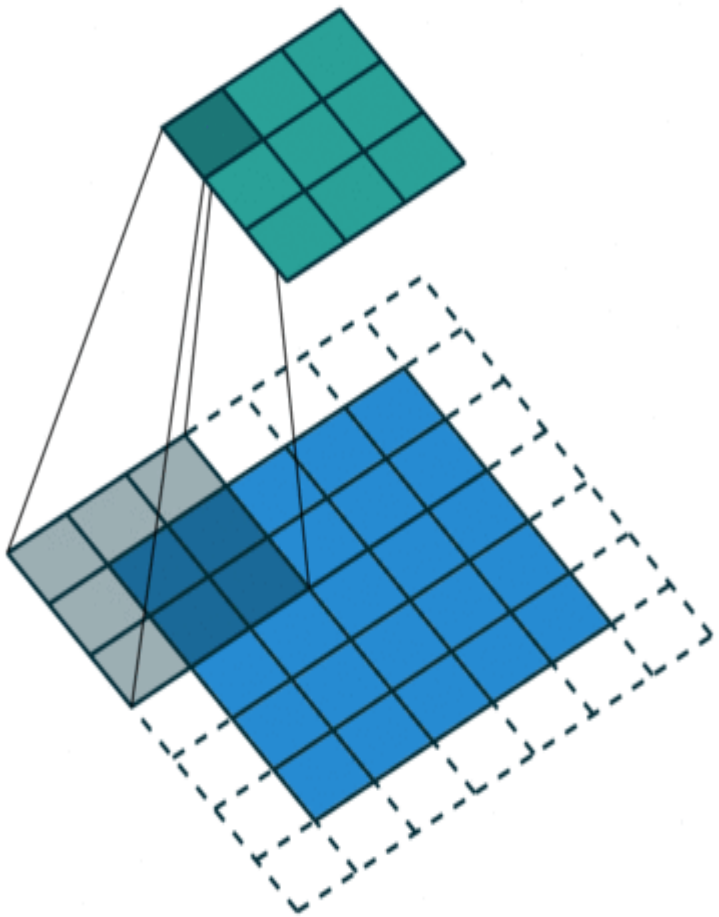
Only connect to 9 input, not fully connected





Sparsity of connections: Less parameters!

Parameter sharing: Even less parameters!



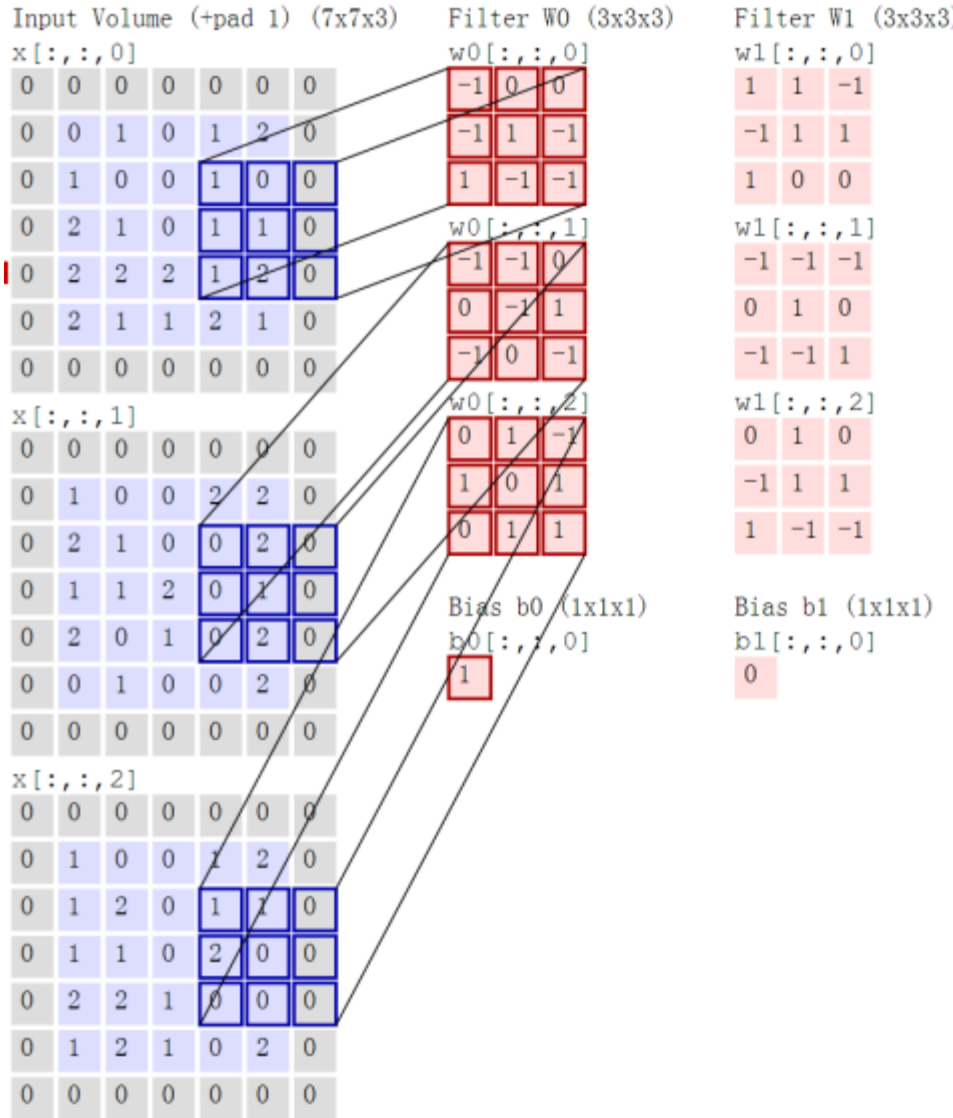
Output

Filter

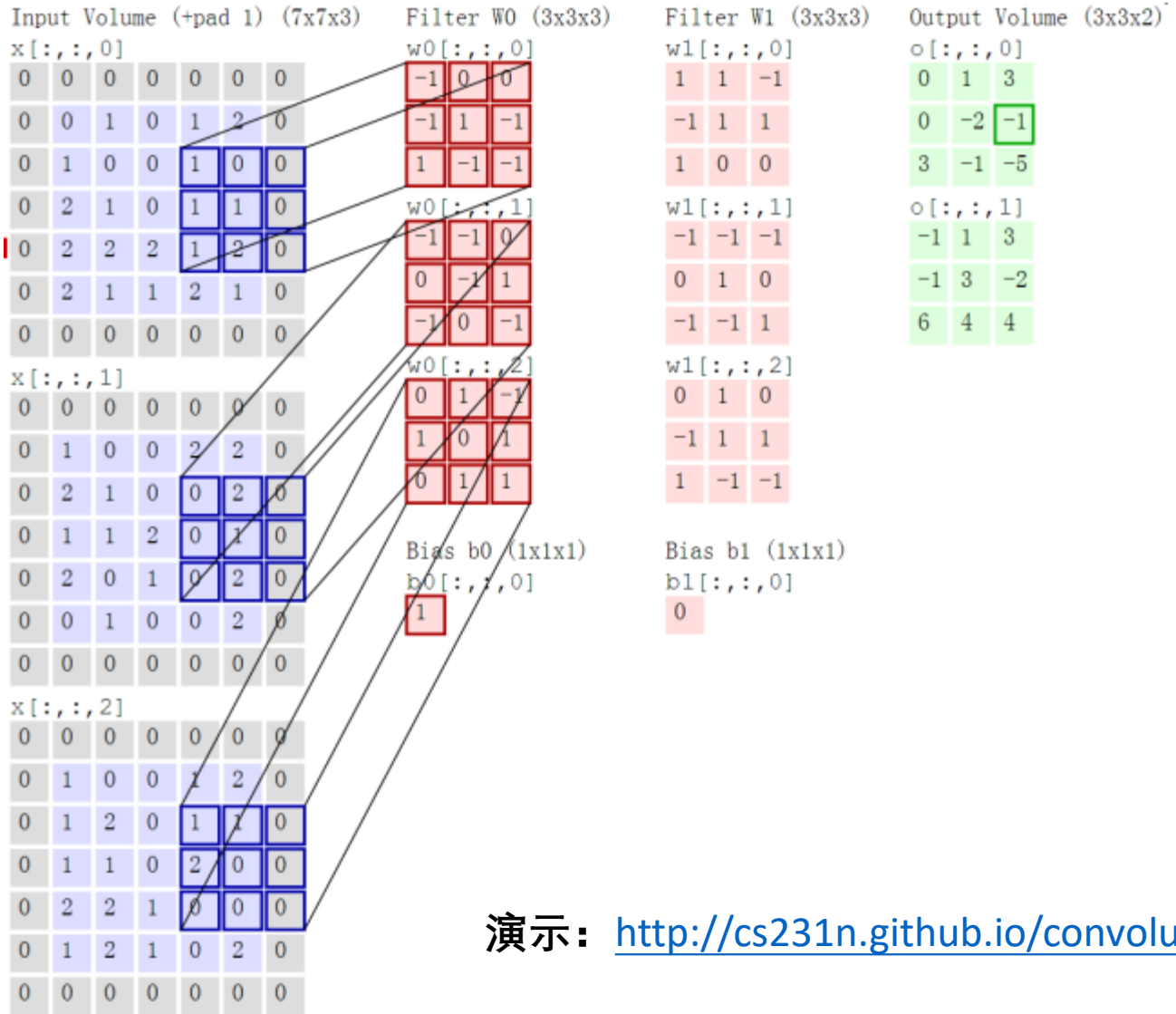
Input

Padding

课堂练习——卷积层的计算


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 1 0 1 2 0
0 1 0 0 1 0 0
0 2 1 0 1 1 0
0 2 2 2 1 2 0
0 2 1 1 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 2 2 0
0 2 1 0 0 2 0
0 1 1 2 0 1 0
0 2 0 1 0 2 0
0 0 1 0 0 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 0 0 1 2 0
0 1 2 0 1 1 0
0 1 1 0 2 0 0
0 2 2 1 0 0 0
0 1 2 1 0 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
-1 0 0
-1 1 -1
1 -1 -1
w0[:,:,1]
-1 -1 0
0 -1 1
-1 0 -1
w0[:,:,2]
0 1 -1
1 0 1
0 1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
1 1 -1
-1 1 1
1 0 0
w1[:,:,1]
-1 -1 -1
0 1 0
-1 -1 1
w1[:,:,2]
0 1 0
-1 1 1
1 -1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0

课堂练习——卷积层的计算

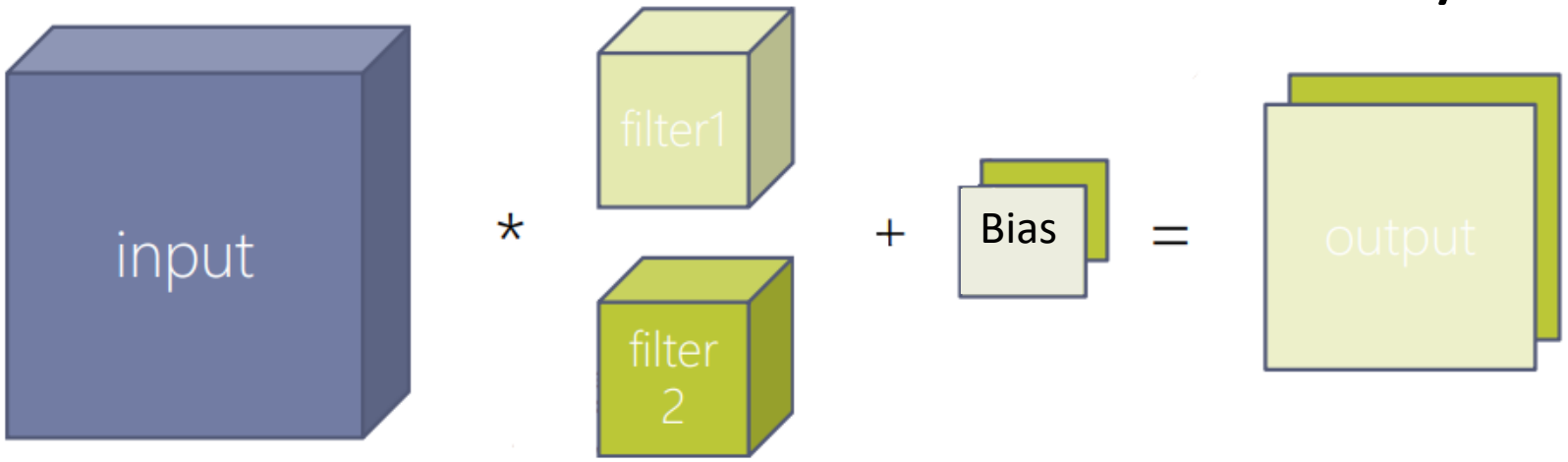


演示： http://cs231n.github.io/convolutional-networks/

More details in Convolution Layer

- Kernel/Filter
  - 2个关键超参数：Filter数量和Filter尺寸
- Stride
  - 滑动Filter时每次移动的像素点个数
  - 与padding共同确定输出图像尺寸
- Padding
  - 扩大输入图像/特征图的尺寸并填充像素；
  - 防止深度网络中图像被动持续减小；
  - 强化图像边缘信息；

More details in Convolution Layer



$N_H[L-1] \times N_W[L-1] \times N_c[L-1]$ | $N_f \times f[L] \times f[L] \times N_c[L-1]$ | $1 \times 1 \times N_f$ | $N_H[L] \times N_W[L] \times N_c[L]$

- Hyperparamters for the convolution layer L:
  - f[L]: Filter Size
    Each filter is: $f[L] \times f[L] \times N_c[L-1]$
  - N_f : number of filters
  - p[L]: Padding
  - s[L]: Stride

Input: $N_H[L-1] \times N_W[L-1] \times N_c[L-1]$
Output: $N_H[L] \times N_W[L] \times N_c[L]$

$$N_H[L] = \left\lfloor \frac{N_H[L-1] + 2*p[L] - f[L]}{s[L]} + 1 \right\rfloor$$

$$N_W[L] = \left\lfloor \frac{N_W[L-1] + 2*p[L] - f[L]}{s[L]} + 1 \right\rfloor$$

$$N_c[L] = N_f$$

总结：Convolution Layer

- Conv层：学到的是局部模式
  - 一个Filter就是一个局部模式的提取器
  - Dense层学到的是全局模式：输入的所有数据上的模式。
- Conv层的两个性质：
  - 平移不变性
  - 模式的空间层次结构：简单、具体 -> 复杂、抽象（语义）
- Conv层大大的缩减参数量：
  - 稀疏连接 & 权值共享

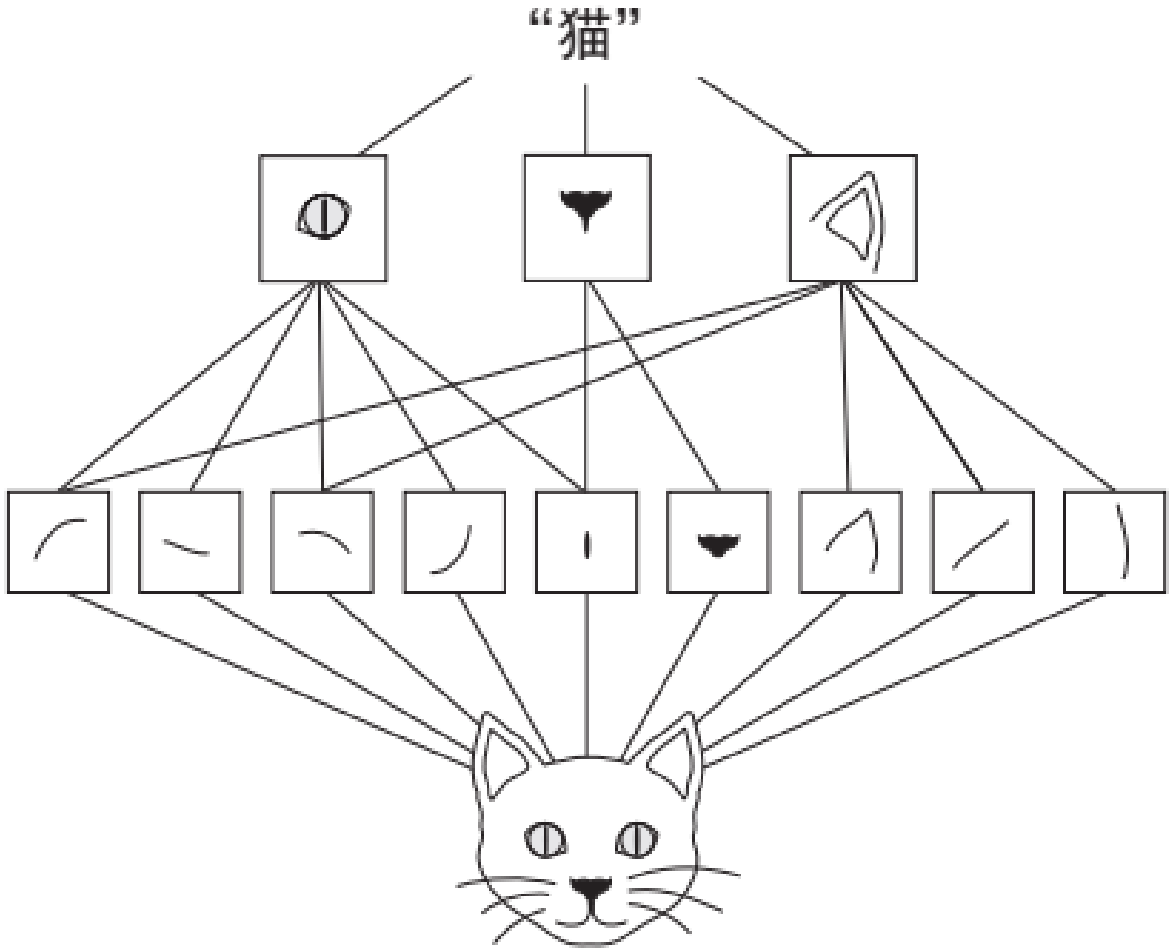


图 5-2　视觉世界形成了视觉模块的空间层次结构：超局部的边缘组合成局部的对象，比如眼睛或耳朵，这些局部对象又组合成高级概念，比如“猫”

Conv Layer in Keras

Conv2D [source]

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_for
```

2D convolution layer (e.g. spatial convolution over images).

https://keras.io/layers/convolutional/

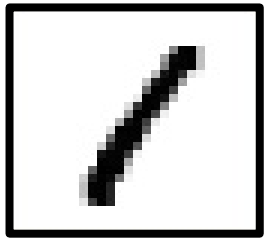
This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the batch axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

Arguments

- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of `"valid"` or `"same"` (case-insensitive). Note that `"same"` is slightly inconsistent across backends with `strides` != 1, as described here
- **data_format**: A string, one of `"channels_last"` or `"channels_first"`. The ordering of the

The whole CNN



Convolution

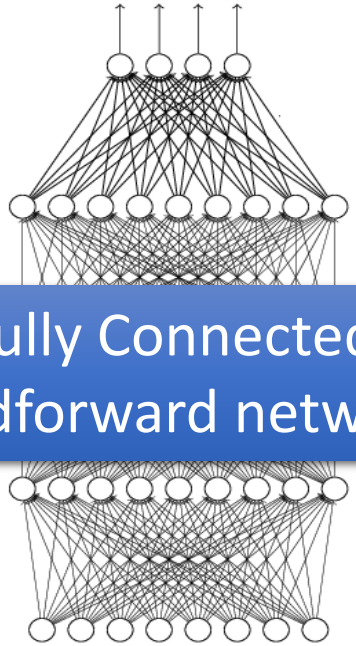
Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten



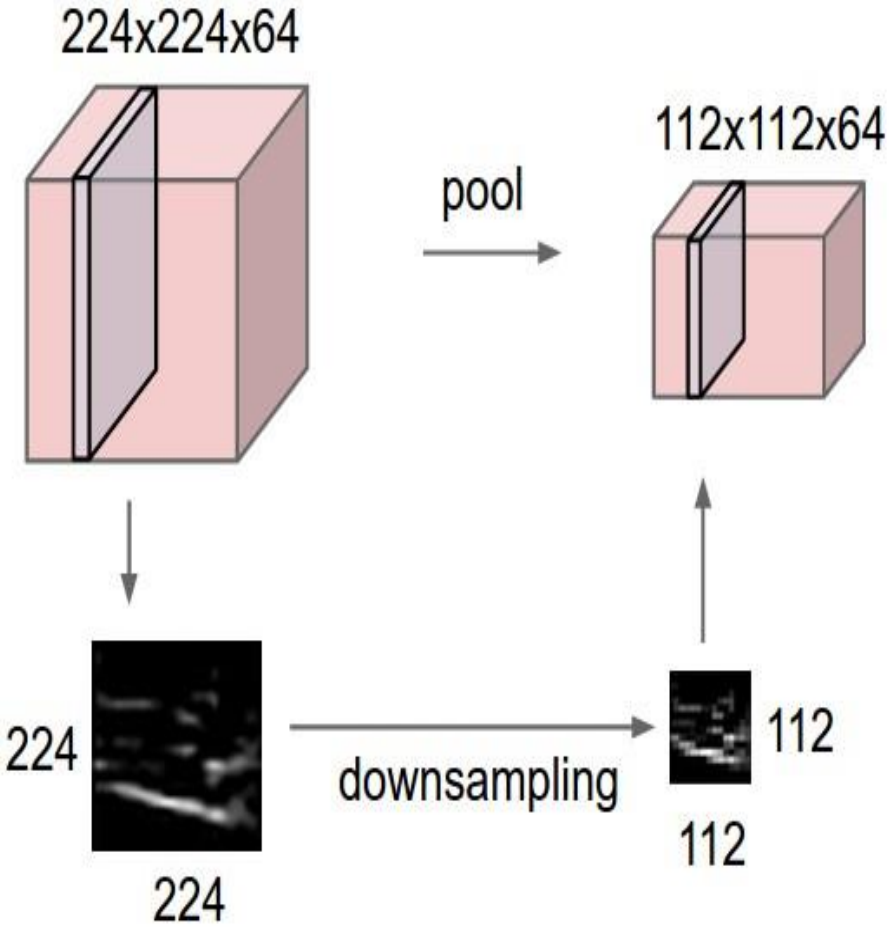
Fully Connected Feedforward network

1 0 0 0 ……

CNN – Pooling Layer

1. makes the representations smaller and more robust

2. operates over each activation map independently



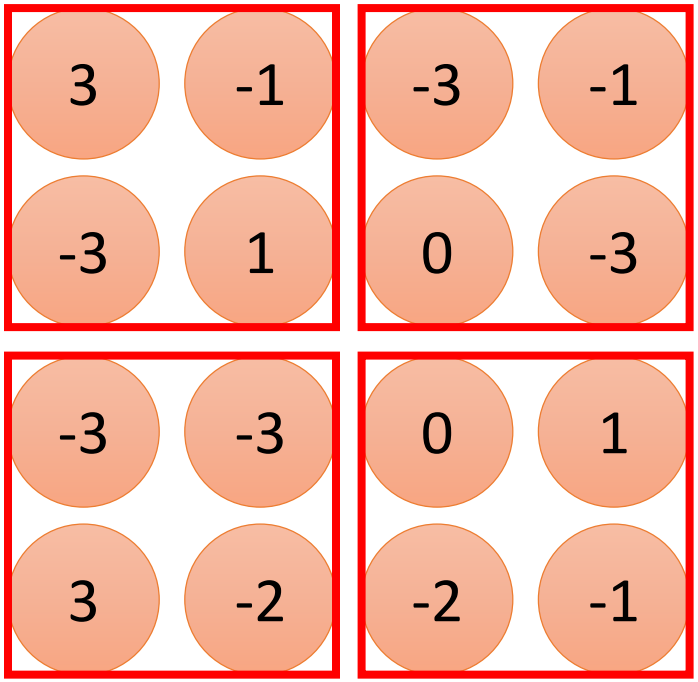
CNN – Max Pooling

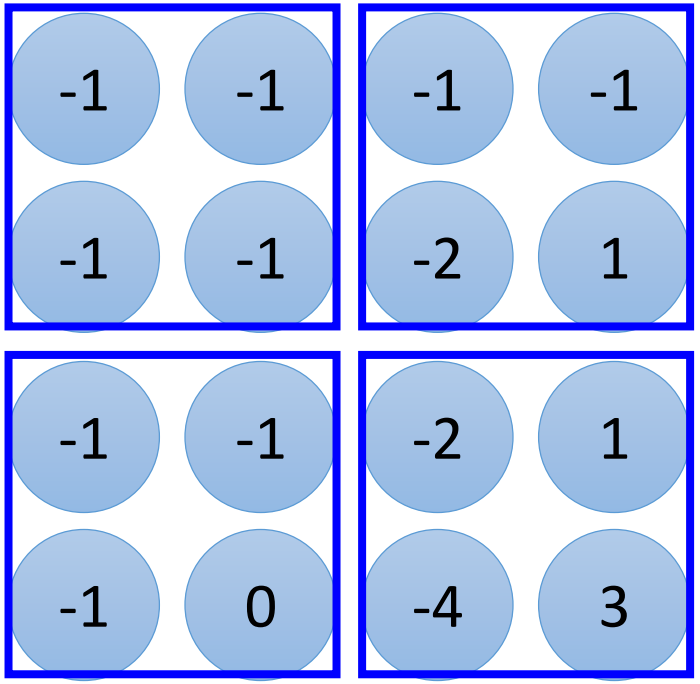
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

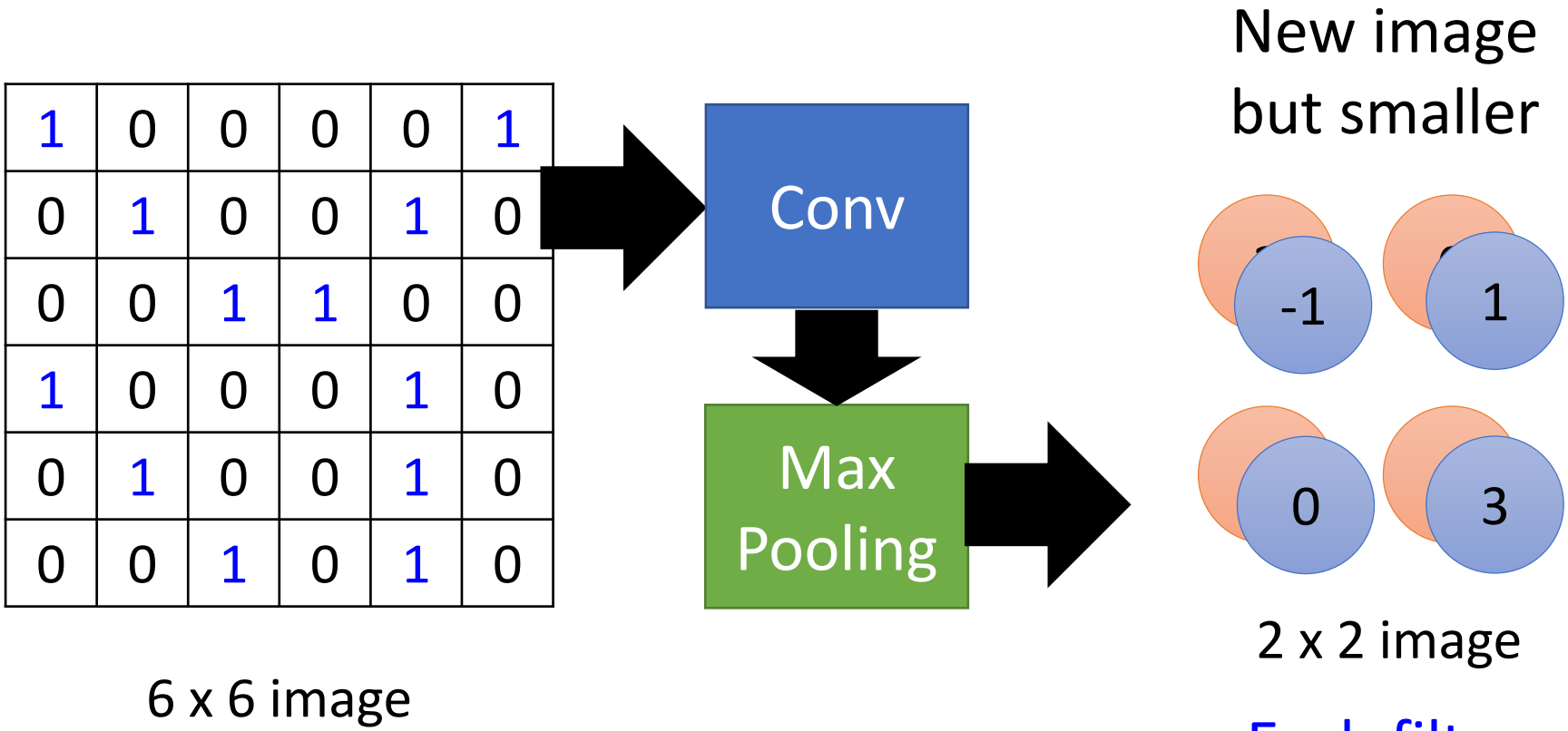
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

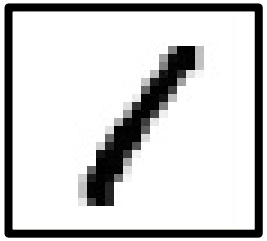


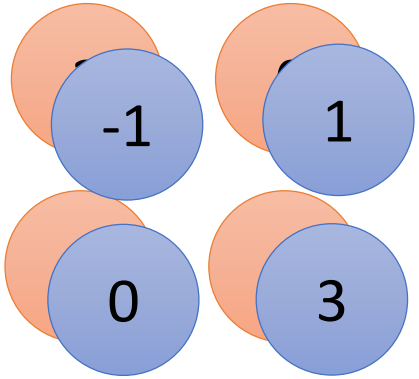


CNN – Max Pooling



The whole CNN





A new image

Convolution

Max Pooling

Convolution

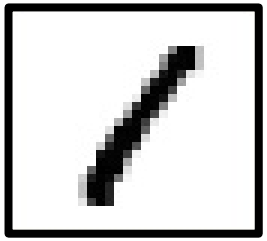
Max Pooling

Can repeat many times

Smaller than the original image

The number of the channel is the number of filters

The whole CNN



Convolution

Max Pooling

A new image

Convolution

Max Pooling

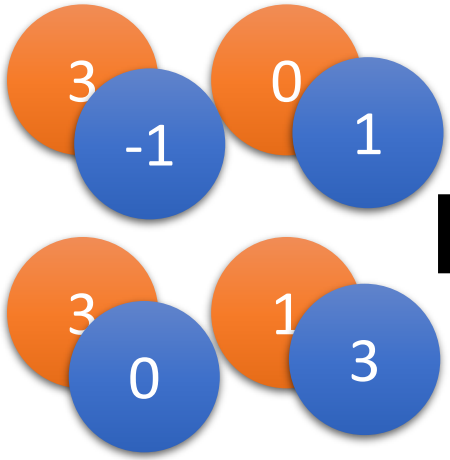
A new image

Flatten

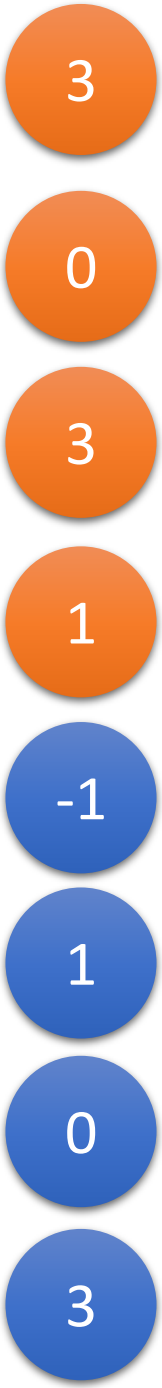
Fully Connected Feedforward network

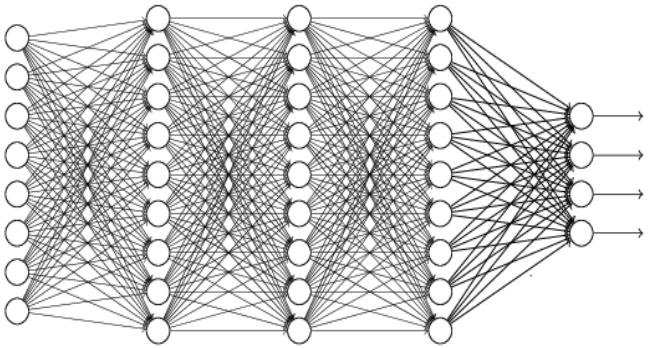
1 0 0 0 ……

Flatten



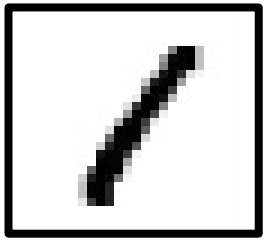
Flatten





Fully Connected
Feedforward network

总结：CNN结构



特征提取

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flatten

分类

1 0 0 0 ……

Fully Connected Feedforward network

总结：CNN结构

- 最常见的卷积神经网络结构如下：

INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC

- 其中*指的是重复次数，**POOL?** 指的是一个可选的汇聚层。其中**N >=0, M>=0， K>=0,** 通常**N<=3， K<3**。

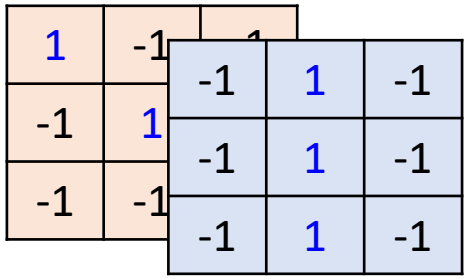
例如，下面是一些常见的网络结构规律：

- **INPUT -> FC,**实现一个线性分类器，此处**N = M = K = 0**。
- **INPUT -> CONV -> RELU -> FC**
- **INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC**。此处在两个池化层之间有一个卷积层。
- **INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC**。此处每个池化层前有两个卷积层，这个思路适用于更大更深的网络，因为在执行具有破坏性的池化操作前，多重的卷积层可以从输入数据中学习到更多的复杂特征。

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(28,28,1)) )
```



.....

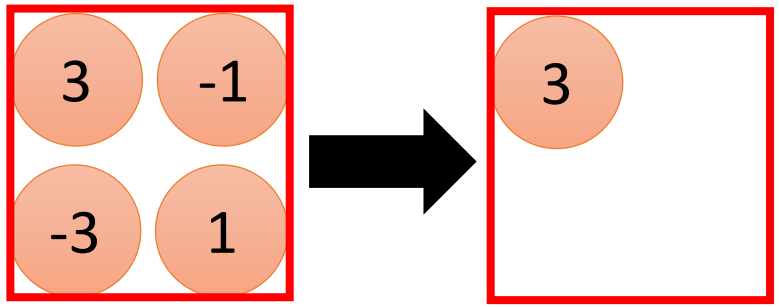
There are 25 3x3 filters.

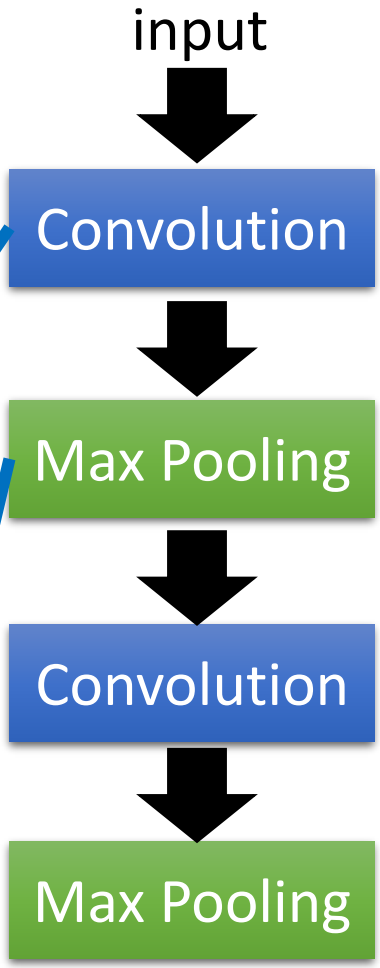
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

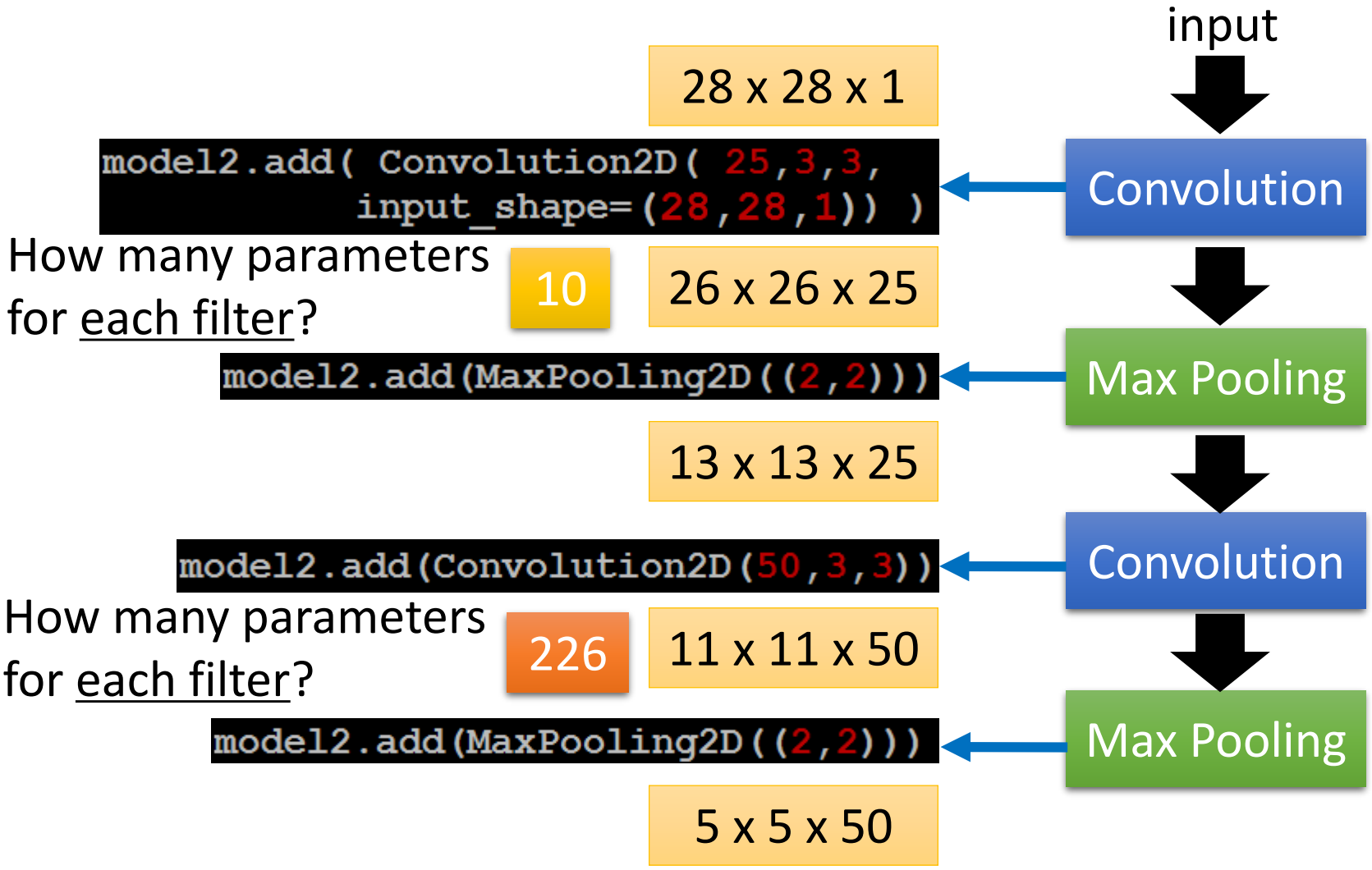
```
model2.add(MaxPooling2D((2,2)))
```





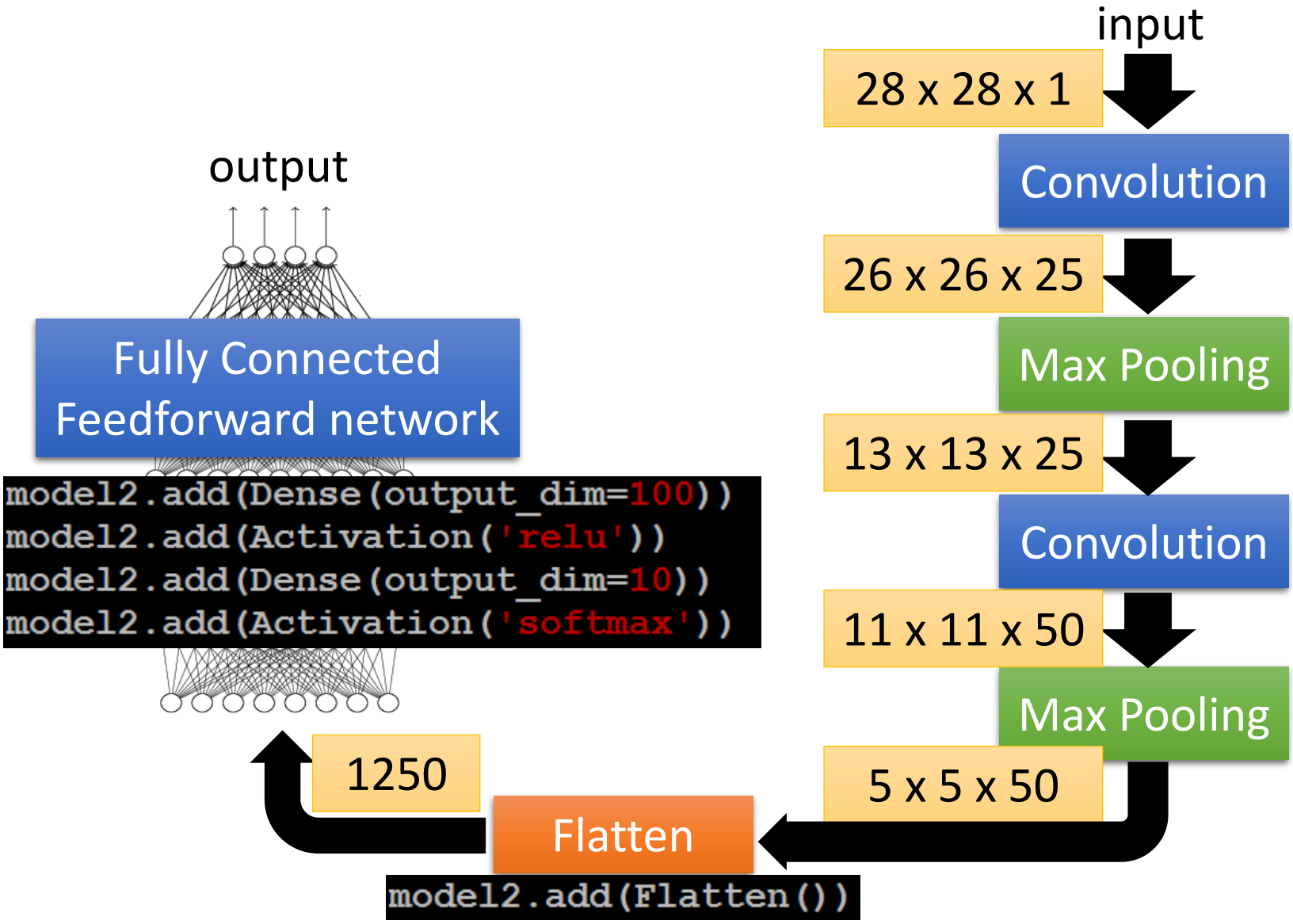
CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***



本章内容

- CNN（Convolutional Neural Network 卷积神经网络）
- 在小型数据集上从头开始训练一个CNN
- 使用预训练的CNN （迁移学习）
- CNN的可视化
- CNN Applications
  - Classification based on CNN
  - Object detection based on CNN

- 首先，在2000个训练样本上训练一个作为**基准模型**的简单的小型卷积神经网络，用于猫狗识别。
  - 要求不做任何正则化处理;
  - 这会得到71%的分类精度;
  - 此时主要的问题在于过拟合;
- 然后，我们会**引入数据增强**（data augmentation），将网络精度提高到82%。
  - 数据增强（data augmentation），在计算机视觉领域是一种非常强大的降低过拟合的技术。

训练基准网络模型

- 下载数据 & 确定成功指标
- 构建网络
- 数据预处理
- 训练模型

训练基准网络模型

- 下载数据 & 确定成功指标
  - 原始数据集包含25 000张猫狗图像（每个类别都有12 500张）；
  - 构造 小样本数据集：我们只使用4000张猫狗图像（每个类别都有2000张）作为数据集；将数据集分为三个子集：每个类别各1000个样本的训练集、每个类别各500个样本的验证集和每个类别各500个样本的测试集。
  - 这是一个平衡的二分类问题，分类精度可作为衡量成功的指标。
- 构建网络
- 数据预处理
- 训练模型

训练基准网络模型——构建网络

代码清单 5-5　将猫狗分类的小型卷积神经网络实例化

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

代码清单 5-6　配置模型用于训练

```
from keras import optimizers

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

课堂练习

代码清单 5-5　将猫狗分类的小型卷积神经网络实例化

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)
_________________________________________
max_pooling2d_1 (MaxPooling2D)
_________________________________________
conv2d_2 (Conv2D)
_________________________________________
max_pooling2d_2 (MaxPooling2D)
_________________________________________
conv2d_3 (Conv2D)
_________________________________________
max_pooling2d_3 (MaxPooling2D)
_________________________________________
conv2d_4 (Conv2D)
_________________________________________
max_pooling2d_4 (MaxPooling2D)
_________________________________________
flatten_1 (Flatten)
_________________________________________
dense_1 (Dense)
_________________________________________
dense_2 (Dense)
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

课堂练习

代码清单 5-5　将猫狗分类的小型卷积神经网络实例化

```
from keras import layers
from keras import models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 74, 74, 32)     0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 36, 36, 64)     0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 17, 17, 128)    0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_4 (MaxPooling2D)  (None, 7, 7, 128)      0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6272)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3211776
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0
```

训练基准网络模型——数据预处理

(1) 读取图像文件。

(2) 将JPEG文件解码为RGB像素网格。

(3) 将这些像素网格转换为浮点数张量。

(4) 将像素值（0~255范围内）缩放到[0, 1]区间（正如你所知，神经网络喜欢处理较小的输入值）。

keras.preprocessing.image中的ImageDataGenerator类，可以快速创建Python生成器，能够将硬盘上的图像文件自动转换为预处理好的张量批量。

代码清单 5-7　使用 ImageDataGenerator 从目录中读取图像

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        validation_dir,
        target_size=(150, 150),
        batch_size=20,
        class_mode='binary')
```

将所有图像乘以 1/255 缩放

目标目录

将所有图像的大小调整为 150×150

因为使用了 **binary_crossentropy** 损失，所以需要用二进制标签

训练基准网络模型——拟合数据

代码清单 5-8　利用批量生成器拟合模型

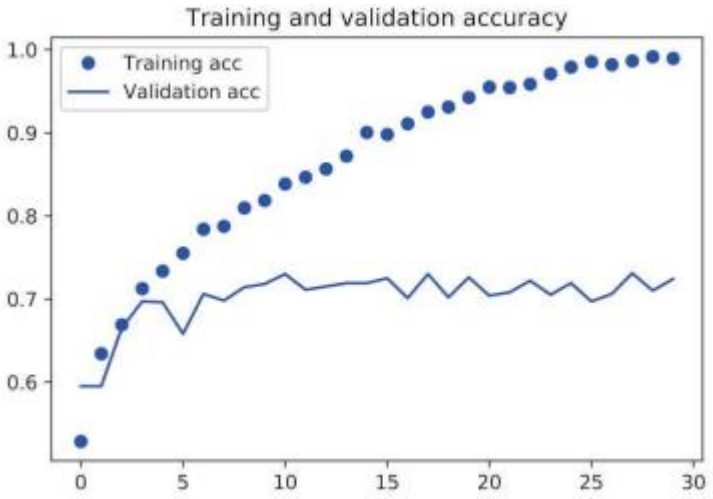
```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

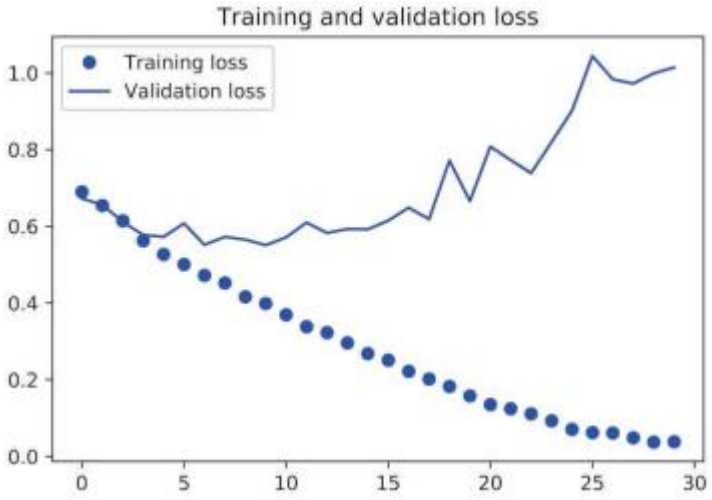
代码清单 5-9　保存模型

```
model.save('cats_and_dogs_small_1.h5')
```

- 训练精度随着时间线性增加，直到接近100%，而验证精度则停留在70%~72%。
- 验证损失仅在5轮后就达到最小值，然后保持不变，而训练损失则一直线性下降，直到接近于0。
- 过拟合!

观察训练集和验证集上的学习曲线:





- 利用数据增强（data augmentation）来解决过拟合!

代码清单 5-11　利用 ImageDataGenerator 来设置数据增强

```
datagen = ImageDataGenerator(
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')
```

注意，不能增强验证集!

Rotation_range是角度值（在0~180范围内），表示图像随机旋转的角度范围。
width_shift 和 **height_shift**是图像在水平或垂直方向上平移的范围（相对于总宽度或总高度的比例）。
shear_range是随机错切变换的角度。
zoom_range是图像随机缩放的范围。
horizontal_flip是随机将一半图像水平翻转。如果没有水平不对称的假设（比如真实世界的图像），这种做法是有意义的。
fill_mode是用于填充新创建像素的方法，这些新像素可能来自于旋转或宽度/高度平移。

图 5-11　通过随机数据增强生成的猫图像

- 数据增强，使得网络的输入变得高度相关，不足以完全消除过拟合。
- 需添加一个Dropout层，进一步降低过拟合

代码清单 5-13　定义一个包含 dropout 的新卷积神经网络

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

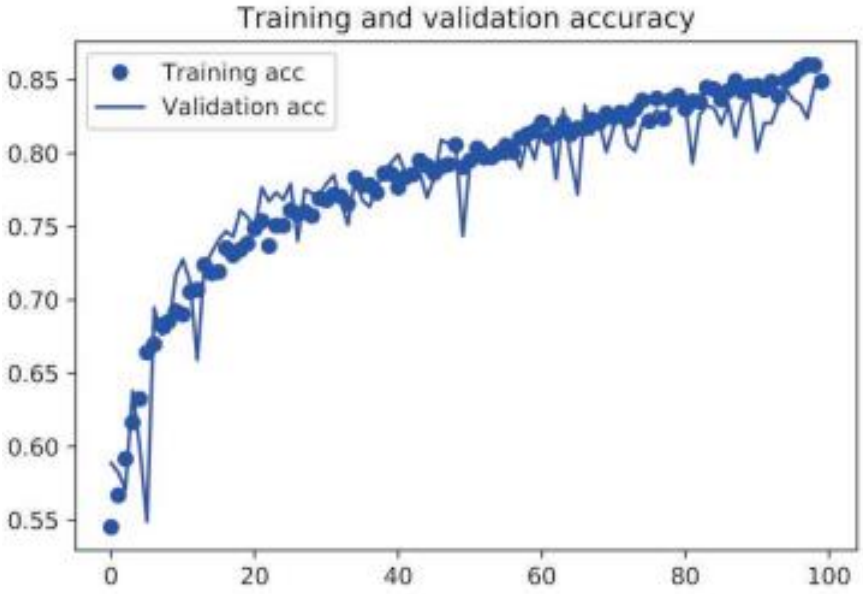


图 5-12　采用数据增强后的训练精度和验证精度

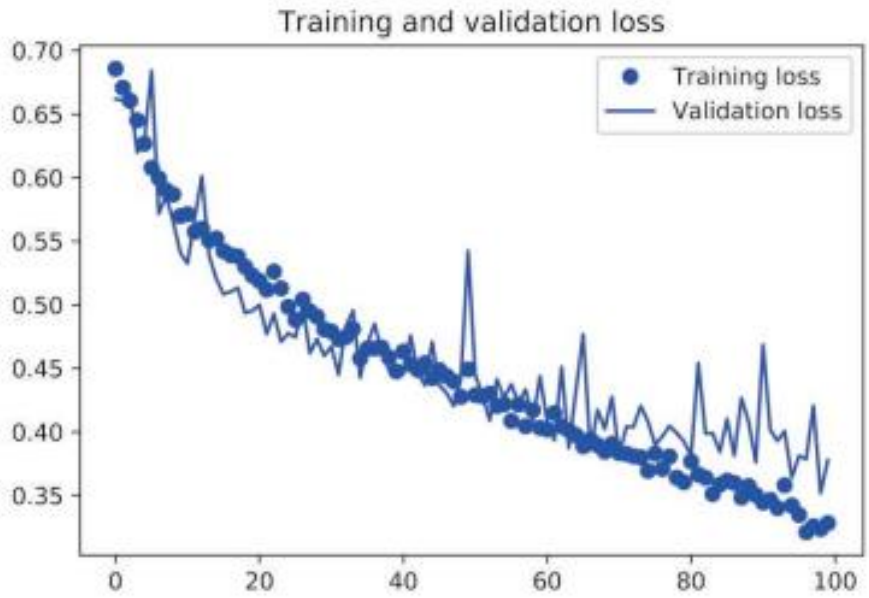


图 5-13　采用数据增强后的训练损失和验证损失

观察到：使用了数据增强和dropout之后，模型不再过拟合：训练曲线紧紧跟随着验证曲线。
现在的精度为82%，比未正则化的模型提高了15%（相对比例）。

本章内容

- CNN（Convolutional Neural Network 卷积神经网络）
- 在小型数据集上从头开始训练一个CNN
- 使用预训练的CNN （迁移学习）
- CNN的可视化
- CNN Applications
  - Classification based on CNN
  - Object detection based on CNN

- 使用预训练网络，可以将深度神经网络模型应用于小型图像数据集上。
- **预训练网络（pretrained network）**是一个保存好的网络，之前已在大型数据集（通常是大规模图像分类任务）上训练好。
- 如果这个原始数据集足够大且足够通用，那么预训练网络学到的特征的空间层次结构可以有效地作为视觉世界的通用模型，因此这些特征可用于各种不同的计算机视觉问题，即使这些新问题涉及的类别和原始任务完全不同。
  - 比如，你在ImageNet上训练了一个网络（其类别主要是动物和日常用品），然后将这个训练好的网络应用于某个不相干的任务，比如在图像中识别家具。这种学到的特征在不同问题之间的可移植性，是深度学习与许多早期浅层学习方法相比的重要优势，它使得深度学习对小数据问题非常有效。

- Keras中内置一些预训练网络：（可以从keras.applications模块中导入；它们都是在ImageNet数据集上预训练得到的）：
  - VGG16
  - Xception
  - Inception V3
  - ResNet50
  - VGG16
  - VGG19
  - MobileNet

- 如何使用预训练网络？（两种方法）
  - 特征提取（feature extraction）：取出预训练好的网络的卷积基，在上面运行新数据；然后基于卷积基的输出结果，训练一个新的分类器
    - 不使用数据增强的特征提取
    - 使用数据增强的特征提取：冻结卷积基
  - 微调模型（fine-tuning）；

特征提取（feature extraction）

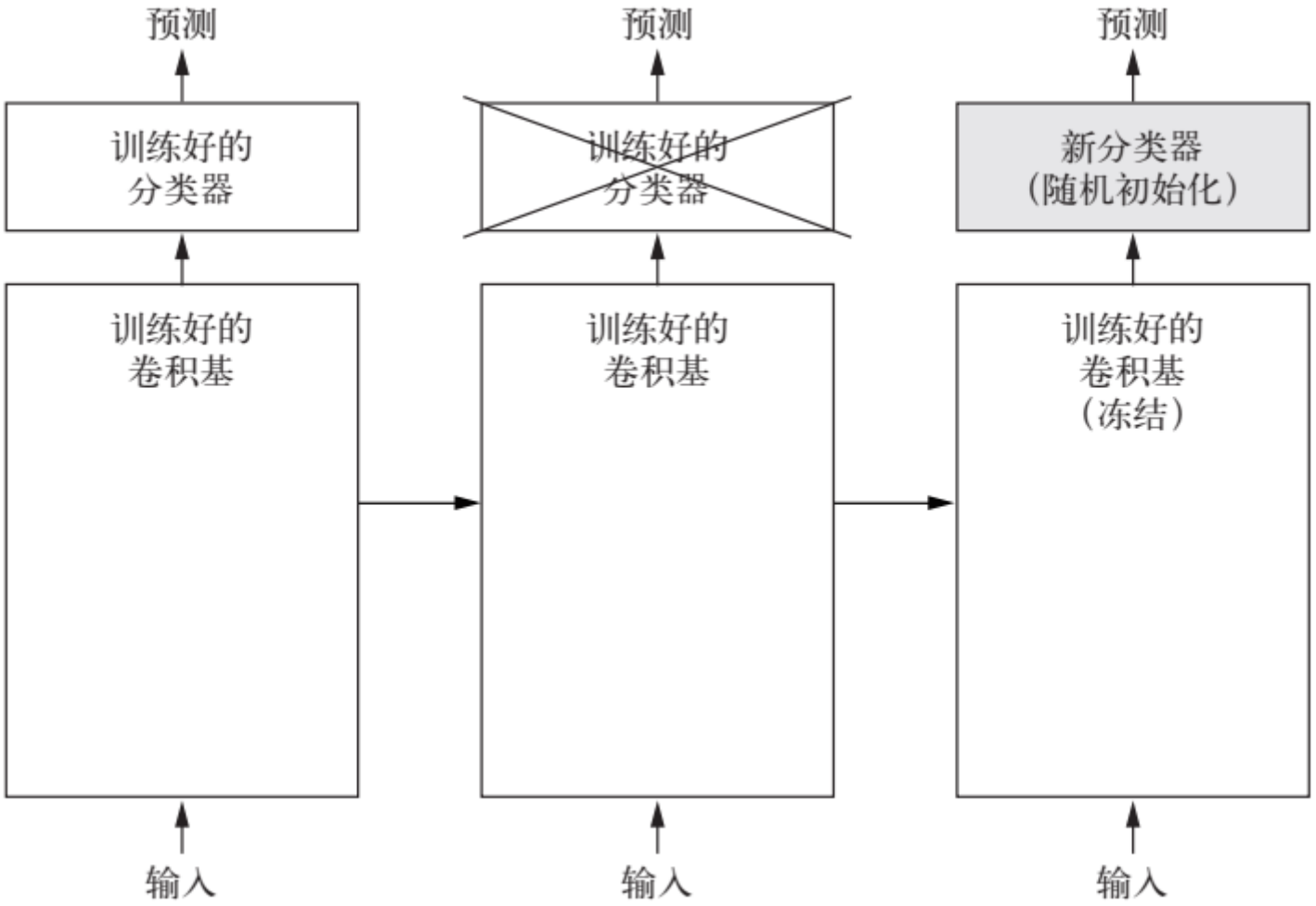


图 5-14　保持卷积基不变，改变分类器

不使用数据增强的特征提取

代码清单 5-16　将 VGG16 卷积基实例化

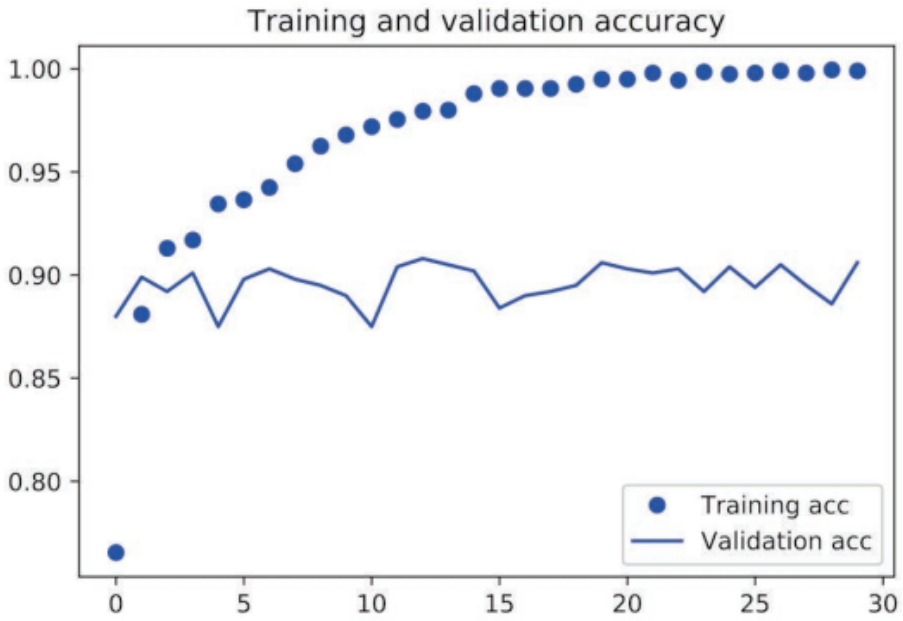
```
from keras.applications import VGG16

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))
```

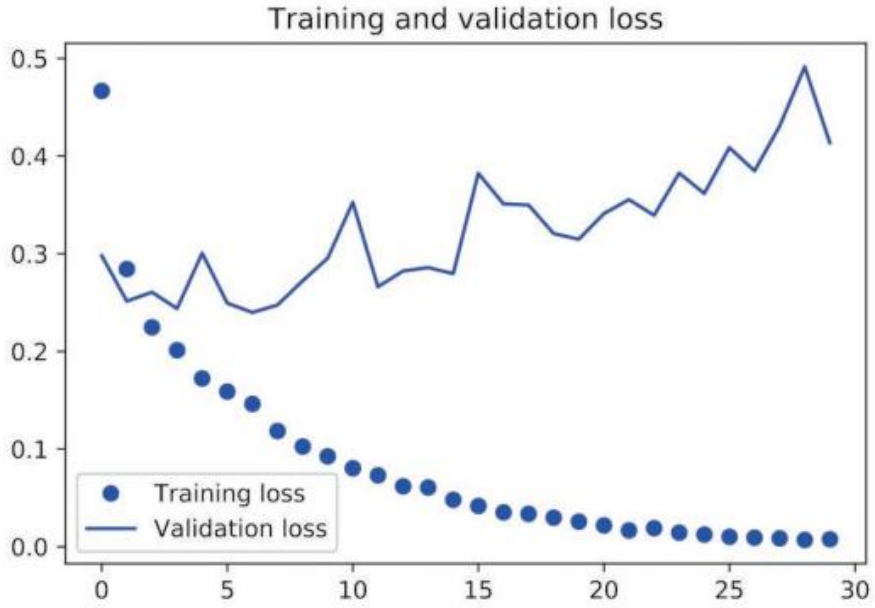
- weights指定模型初始化的权重检查点。
- include_top指定模型最后是否包含密集连接分类器。
- input_shape（可选）是输入到网络中的图像张量的形状。若未指定，那么网络能够处理任意形状的输入。

利用卷积基提取特征，并保存到张量变量**train_features**中

构建模型，训练模型

```
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer=optimizers.RMSprop(lr=2e-5),
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=30,
                    batch_size=20,
                    validation_data=(validation_features, validation_labels))
```





- 验证精度达到了约90%，比上一节从头开始训练的小型模型效果要好得多。
- 但从图中也可以看出，虽然dropout比率相当大，但模型几乎从一开始就过拟合。这是因为本方法没有使用数据增强，而数据增强对防止小型图像数据集的过拟合非常重要。

使用数据增强的特征提取

代码清单 5-20　在卷积基上添加一个密集连接分类器

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(conv_base)
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

定义模型

现在模型的架构如下所示。

```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               2097408
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 16,812,353
Trainable params: 16,812,353
Non-trainable params: 0
```

- 在编译和训练模型之前，一定要“冻结”卷积基。

```
conv_base.trainable = False
```

- 使用数据增强
- 编译模型

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['acc'])
```

- 训练模型

```
history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=30,
      validation_data=validation_generator,
      validation_steps=50)
```

验证精度约为96%!

微调模型(Tune model)

- 微调，是指将其卷机基的顶部（靠近输出层）的几层“解冻”，并将这解冻的几层和新增加的部分（本例中是全连接分类器）联合训练。
- 微调时，要求解冻层的权值更新不要变化太大，因而学习率设置得偏小。

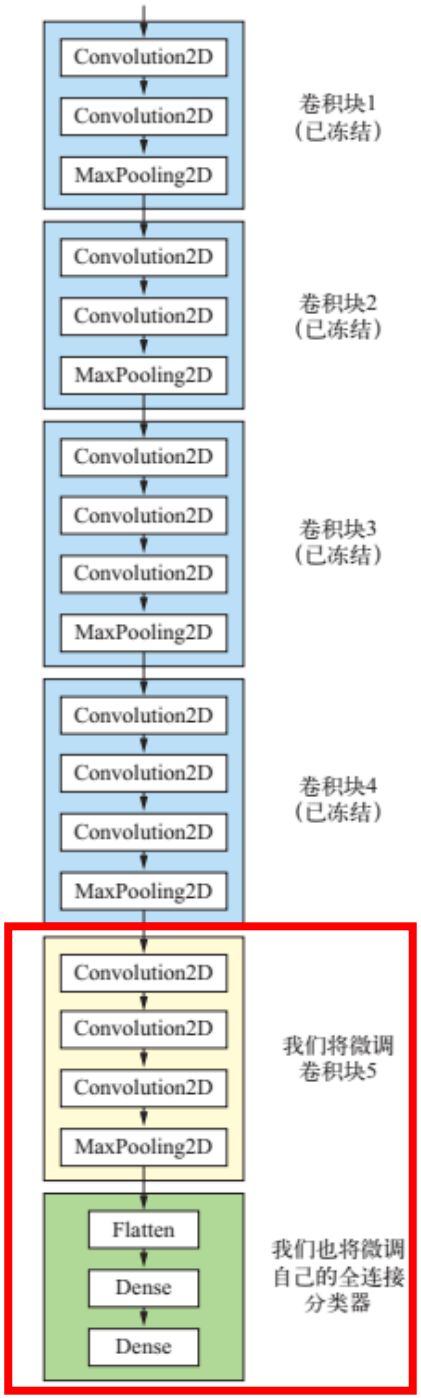
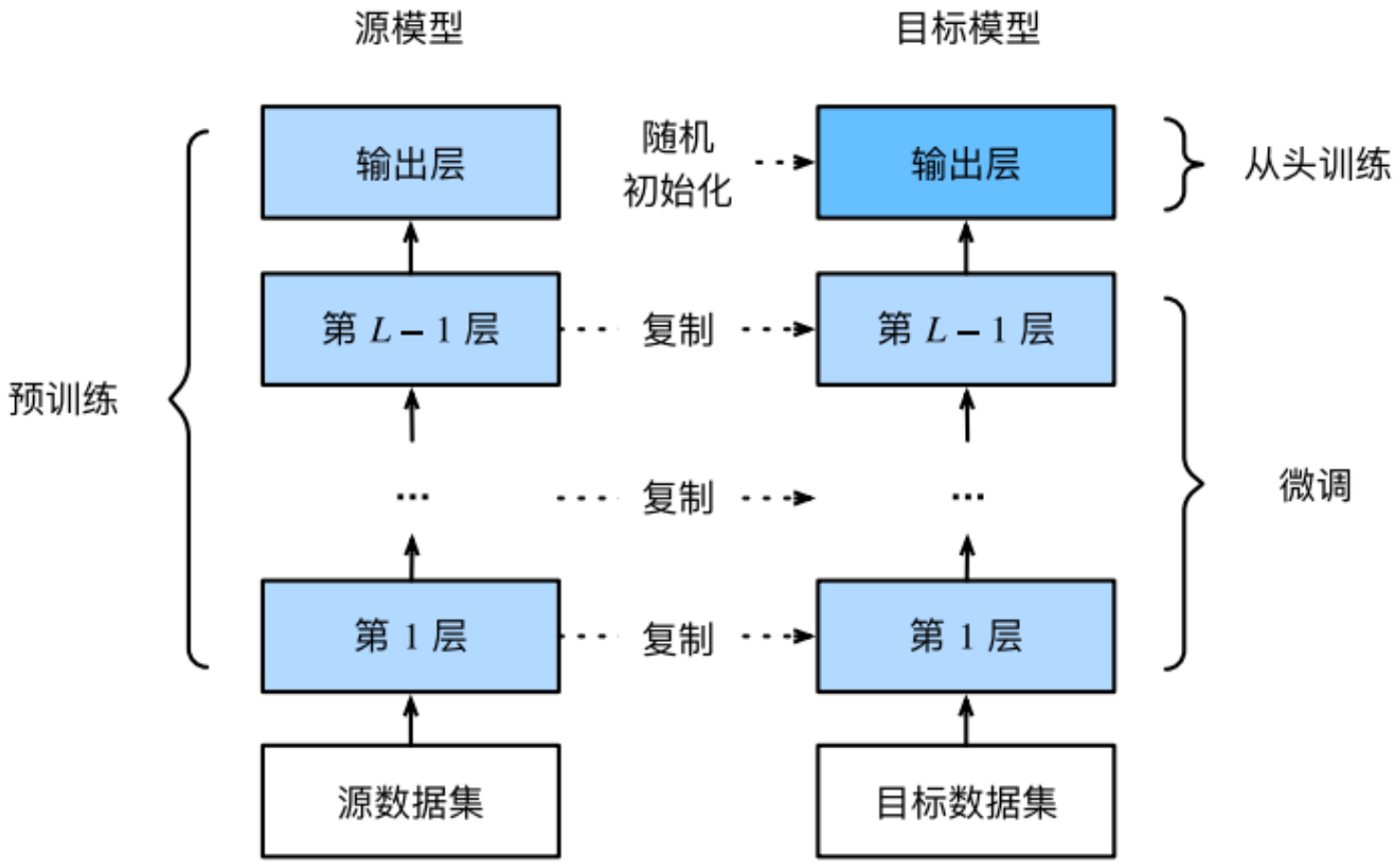


图 5-19　微调 VGG16 网络的最后一个卷积块


源模型
目标模型
输出层
随机
初始化
输出层
从头训练
第 $L-1$ 层
复制
第 $L-1$ 层
预训练
...
复制
...
微调
第 1 层
复制
第 1 层
源数据集
目标数据集

- 微调网络的步骤如下。

  (1) 在已经训练好的基网络（base network）上添加自定义网络。

  (2) 冻结基网络。

  (3) 训练所添加的部分。

  (4) 解冻基网络的一些层。

  (5) 联合训练解冻的这些层和添加的部分。

代码清单 5-22　冻结直到某一层的所有层

```
conv_base.trainable = True

set_trainable = False
for layer in conv_base.layers:
    if layer.name == 'block5_conv1':
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

直到block4_pool的所有层都应该被冻结，而block5_conv1、block5_conv2和block5_conv3三层应该是可训练的。

代码清单 5-23　微调模型

```
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-5),
              metrics=['acc'])

history = model.fit_generator(
      train_generator,
      steps_per_epoch=100,
      epochs=100,
      validation_data=validation_generator,
      validation_steps=50)
```

- **97%的测试精度！（97%是这个数据集的原始Kaggle竞赛中的最佳结果之一）**
  - **Kaggle竞赛中，使用约20,000个样本做训练集。**
- **但我们只用一小部分训练数据（约10%）就得到了这个结果。**
  - **训练20 000个样本与训练2000个样本是有很大差别的！**

Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

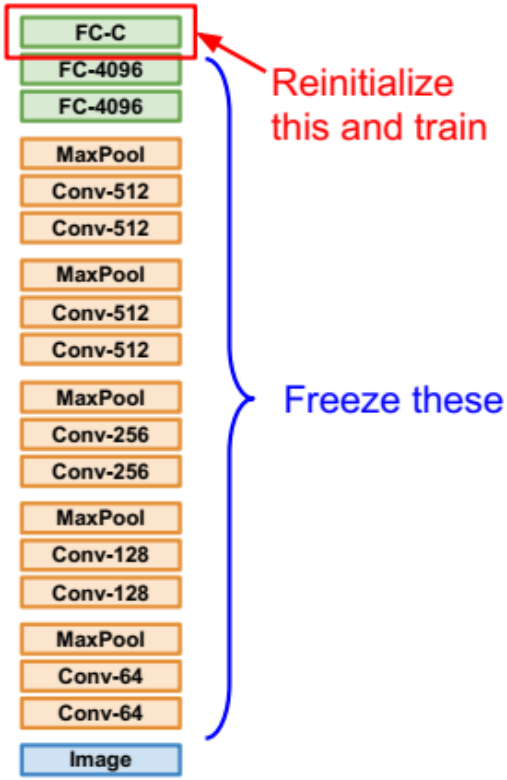
1. Train on Imagenet

Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet

2. Small Dataset (C classes)

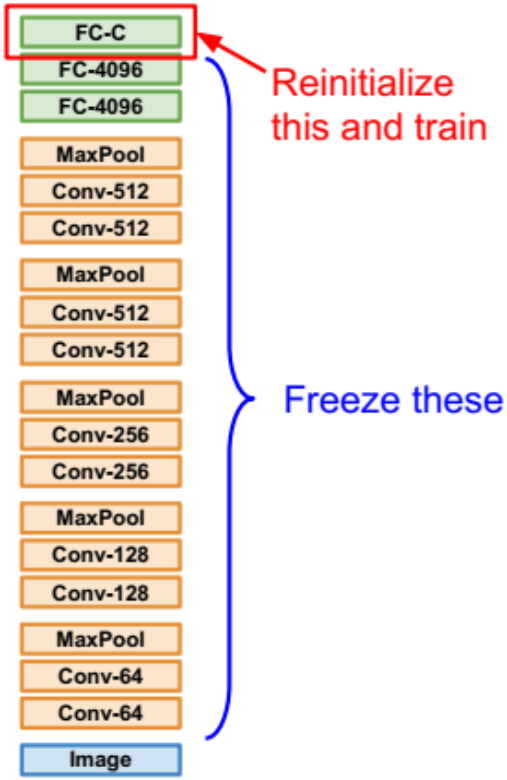


Transfer Learning with CNNs

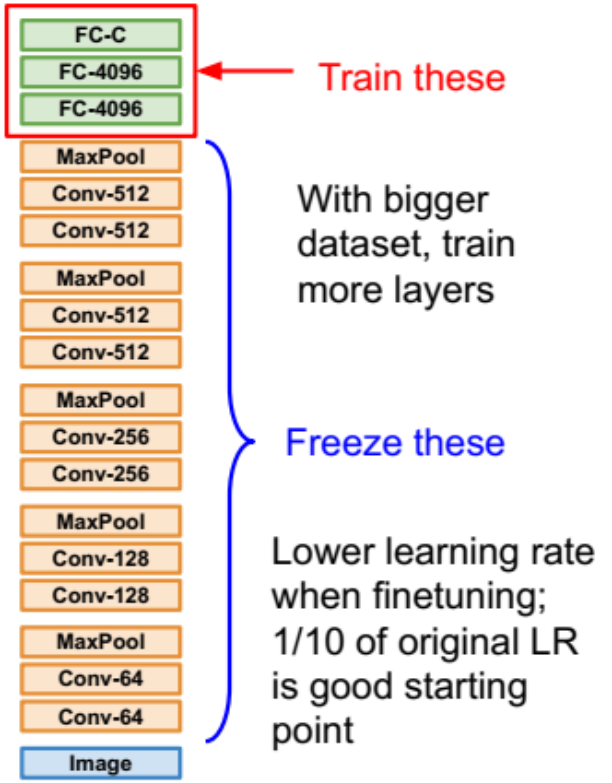
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet

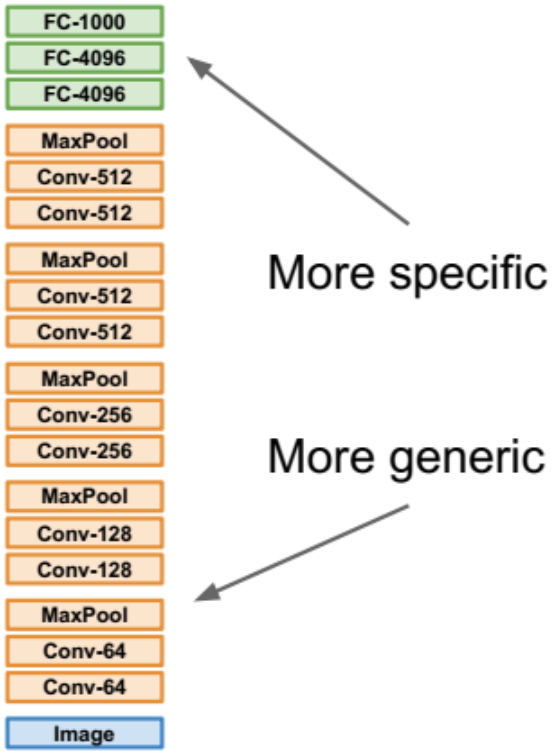
2. Small Dataset (C classes)



3. Bigger dataset

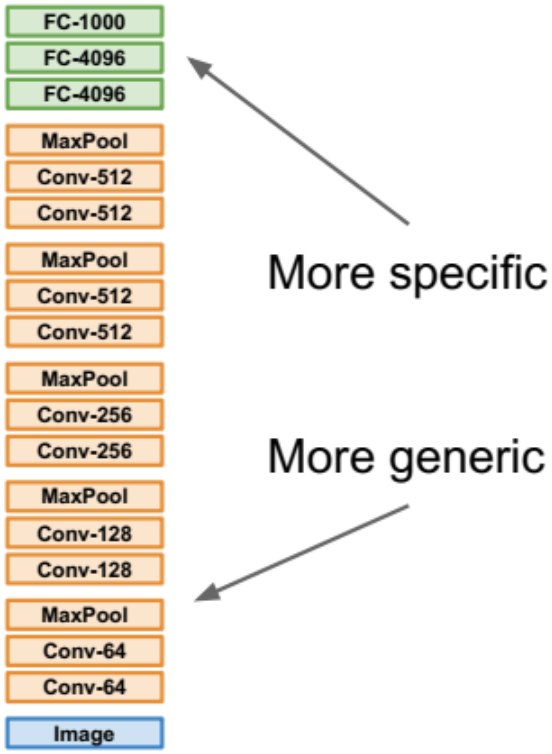


什么时候可以使用迁移学习?



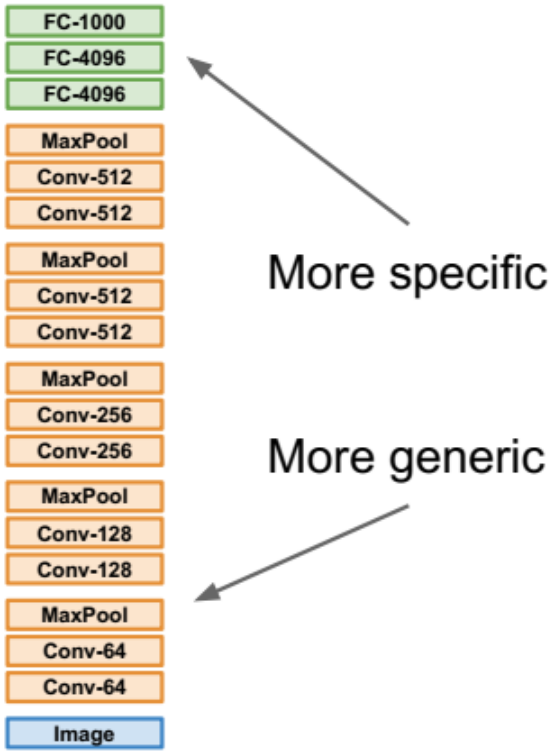
	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?

什么时候可以使用迁移学习?



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?

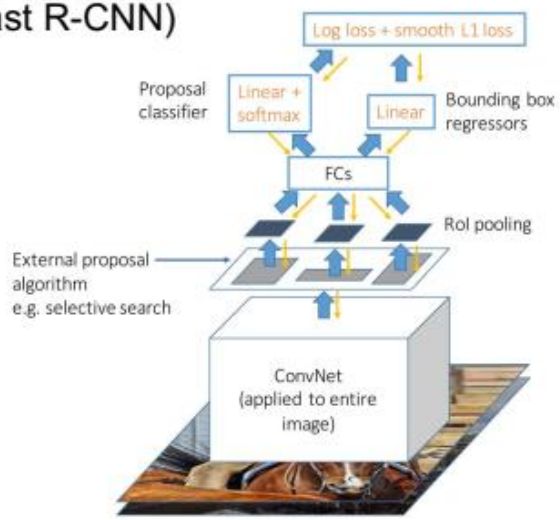
什么时候可以使用迁移学习?



	very similar dataset	very different dataset
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

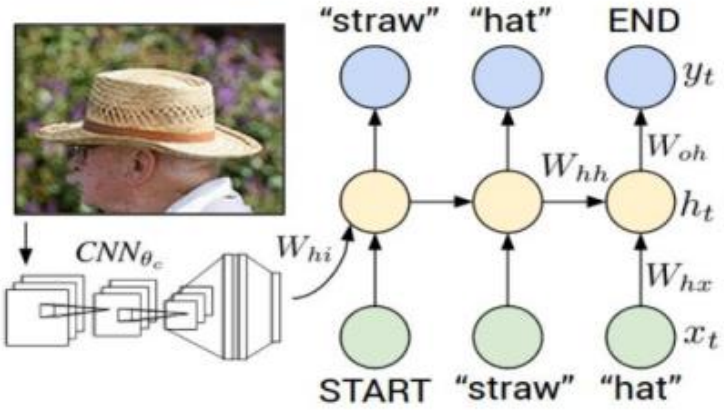
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

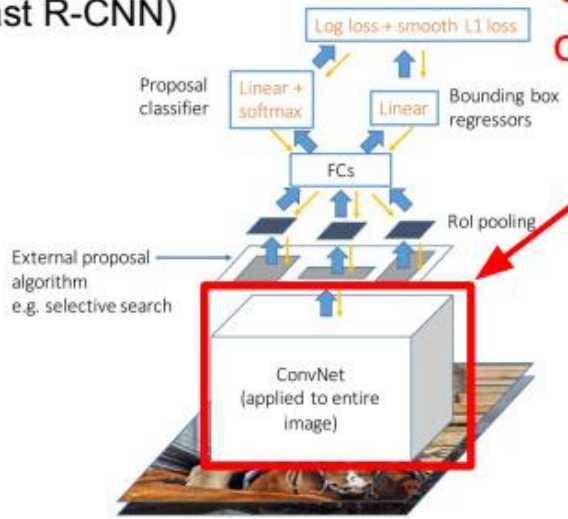
Image Captioning: CNN + RNN



Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

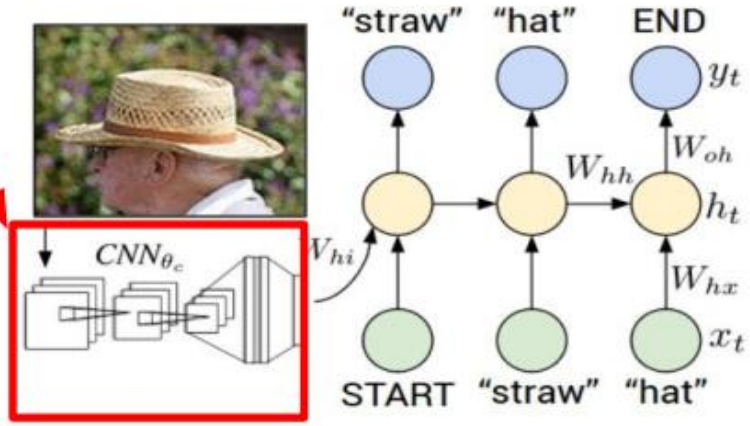
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN

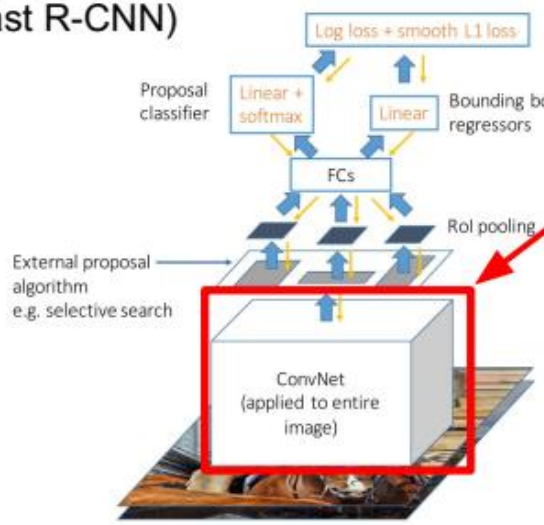


Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

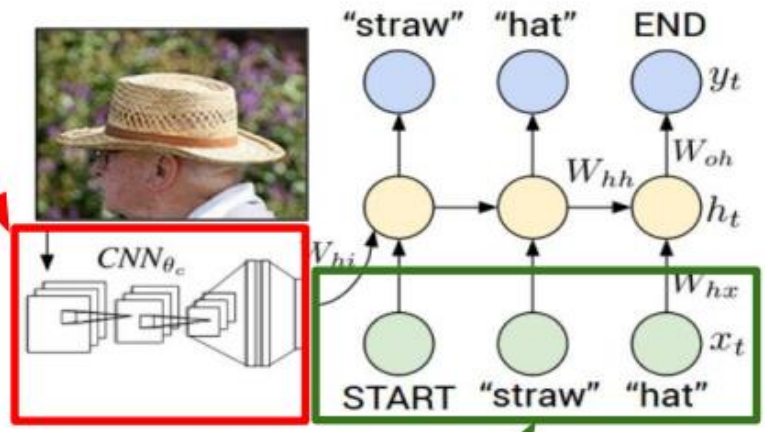
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

Object Detection
(Fast R-CNN)



CNN pretrained on ImageNet

Image Captioning: CNN + RNN



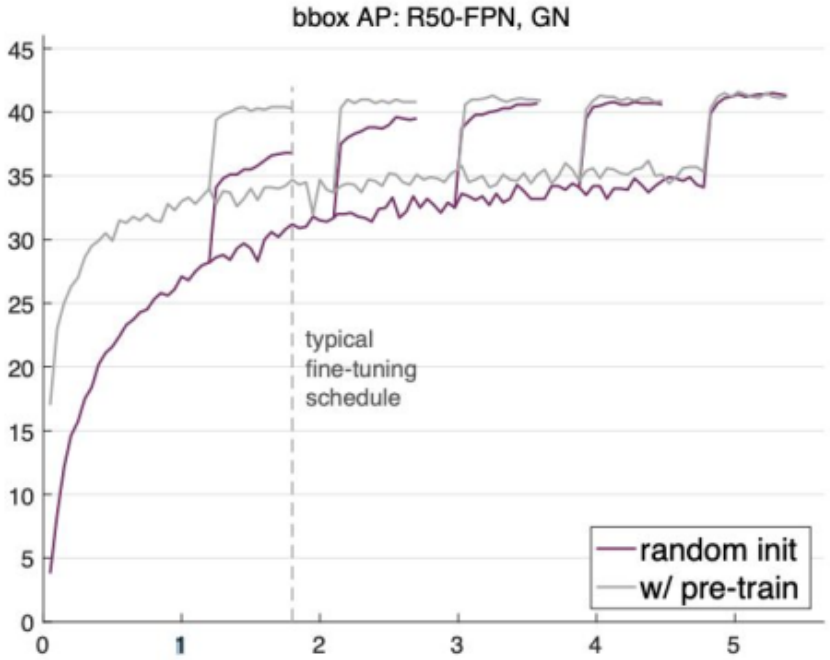
Word vectors pretrained with word2vec

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Figure copyright IEEE, 2015. Reproduced for educational purposes.

Transfer learning with CNNs is pervasive…

But recent results show it might not always be necessary!



He et al, “Rethinking ImageNet Pre-training”, arXiv 2018

Takeaway for your projects and beyond:

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a "Model Zoo" of pretrained models so you don't need to train your own

TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision

小结：小型数据集上的CNN

- 卷积神经网络是用于计算机视觉任务的最佳机器学习模型。即使在非常小的数据集上也可以从头开始训练一个卷积神经网络，而且得到的结果还不错。
- 在小型数据集上的主要问题是过拟合。在处理图像数据时，数据增强是一种降低过拟合的强大方法。
- 利用特征提取，可以很容易将现有的卷积神经网络复用于新的数据集。对于小型图像数据集，这是一种很有价值的方法。
- 作为特征提取的补充，你还可以使用微调，将现有模型之前学到的一些数据表示应用于新问题。这种方法可以进一步提高模型性能。

本章内容

- CNN（Convolutional Neural Network 卷积神经网络）
- 在小型数据集上从头开始训练一个CNN
- 使用预训练的CNN （迁移学习）
- CNN的可视化
- CNN Applications
  - Classification based on CNN
  - Object detection based on CNN

- 深度学习模型是“黑盒”，即模型学到的表示很难用人类可以理解的方式来提取和呈现。
- 卷积神经网络学到的表示非常适合可视化，很大程度上是因为它们是视觉概念的表示。自2013年以来，人们开发了多种技术来对这些表示进行可视化和解释。
- CNN可视化的三种典型方法：
  - 可视化卷积神经网络的中间输出（中间激活）：有助于理解卷积神经网络连续的层如何对输入进行变换，也有助于初步了解卷积神经网络每个过滤器的含义。
  - 可视化卷积神经网络的过滤器：有助于精确理解卷积神经网络中每个过滤器容易接受的视觉模式或视觉概念。
  - 可视化图像中类激活的热力图：有助于理解图像的哪个部分被识别为属于某个类别，从而可以定位图像中的物体。

可视化CNN的中间激活

- 可视化 5.2节中的从头开始训练的小型卷积神经网络
- **可视化中间激活**，是指对于给定输入，展示网络中各个卷积层和池化层输出的特征图（层的输出通常被称为该层的激活，即激活函数的输出）。
  - 在三个维度对特征图进行可视化：宽度、高度和深度（通道）。每个通道都对应相对独立的特征;
  - 将这些特征图可视化的正确方法是将每个通道的内容分别绘制成二维图像。
  - 可以看到输入如何被分解为网络学到的不同过滤器。

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_5 (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_6 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_6 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_7 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_7 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_8 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_8 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten_2 (Flatten)	(None, 6272)	0
dropout_1 (Dropout)	(None, 6272)	0
dense_3 (Dense)	(None, 512)	3211776
dense_4 (Dense)	(None, 1)	513

Total params: 3,453,121
Trainable params: 3,453,121
Non-trainable params: 0

32个148×148的特征图

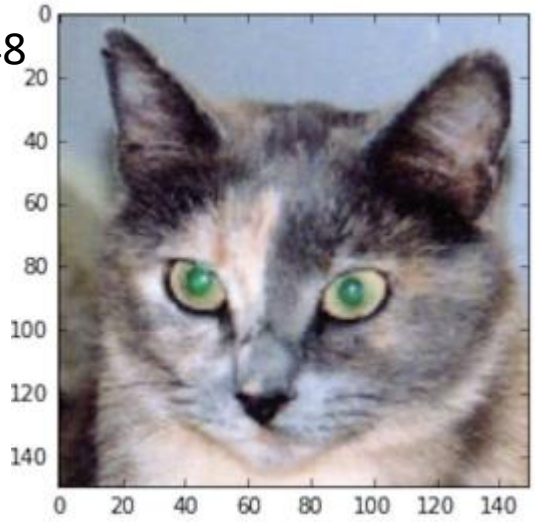


图 5-24　测试的猫图像

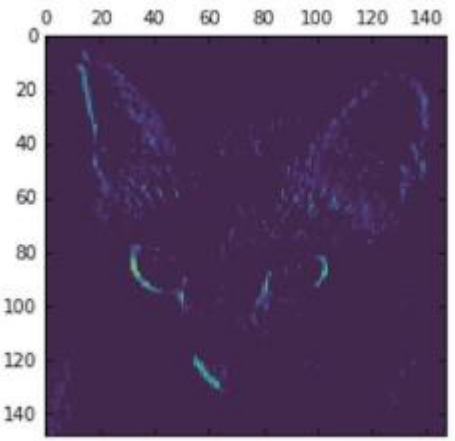


图 5-25　对于测试的猫图像，第一层激活的第 4 个通道

对角边缘检测器

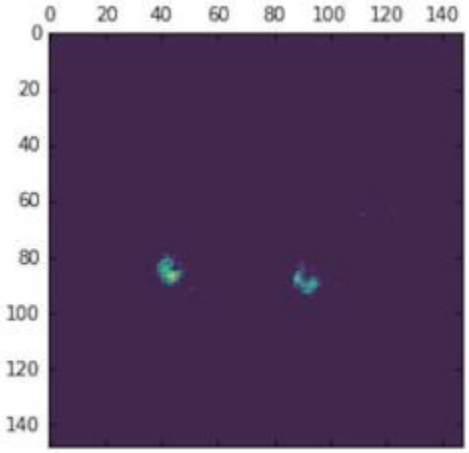


图 5-26　对于测试的猫图像，第一层激活的第 7 个通道

“鲜绿色圆点”检测器

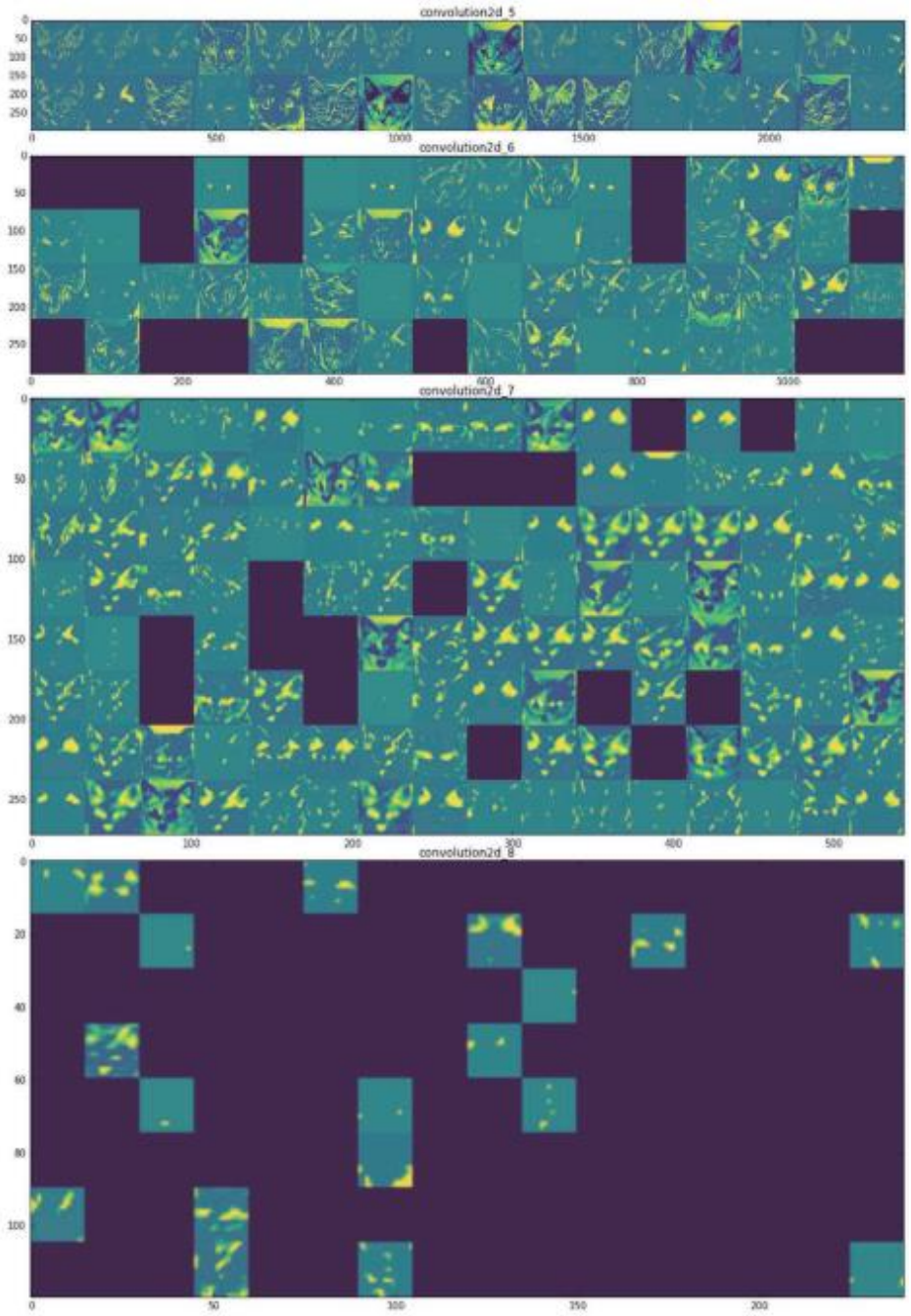


图 5-27　对于测试的猫图像，每个层激活的所有通道

- 第一层是各种边缘探测器的集合。在这一阶段，激活几乎保留了原始图像中的所有信息。
- 随着层数的加深，激活变得越来越抽象，并且越来越难以直观地理解。它们开始表示更高层次的概念，比如“猫耳朵”和“猫眼睛”。层数越深，其表示中关于图像视觉内容的信息就越少，而关于类别的信息就越多。
- 激活的稀疏度（sparsity）随着层数的加深而增大。在第一层里，所有过滤器都被输入图像激活，但在后面的层里，越来越多的过滤器是空白的。也就是说，输入图像中找不到这些过滤器所编码的模式。

深度神经网络 作为 信息蒸馏管道（information distillation pipeline），输入原始数据（本例中是RGB图像），反复对其进行变换，将无关信息过滤掉（比如图像的具体外观），并放大和细化有用的信息（比如图像的类别）。

图 5-28 （左图）试着凭记忆画一辆自行车；（右图）自行车示意图

深度神经网络 类似 人类和动物感知世界的方式

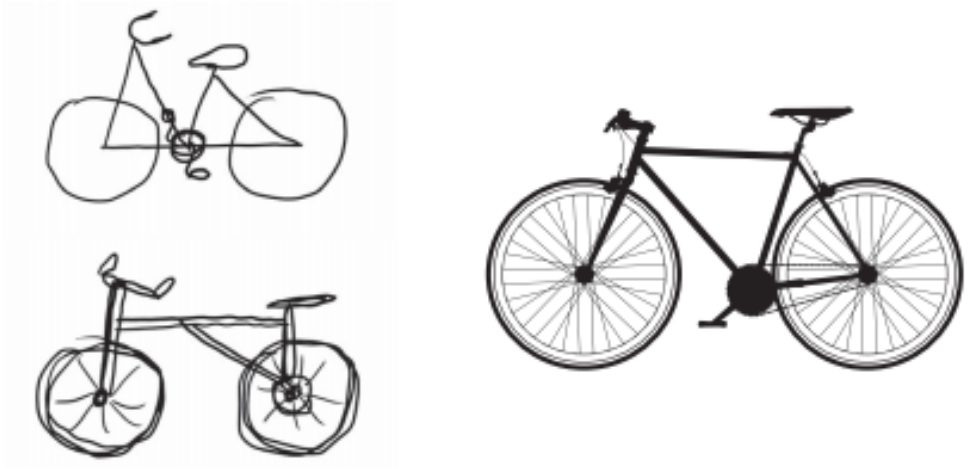


图 5-28 （左图）试着凭记忆画一辆自行车；（右图）自行车示意图

深度神经网络 类似 人类和动物感知世界的方式

可视化CNN的Filter

- VGG16模型
- 显示每个过滤器所响应的视觉模式;
- 如何找到?
  - 通过在输入空间中进行梯度上升来实现：从空白输入图像开始，将梯度下降应用于卷积神经网络输入图像的值，其目的是让某个过滤器的响应最大化。
  - 使得过滤器的响应最大化的输入图像，就是该过滤器具有最大响应的图像，代表了该过滤器所响应的视觉模式

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 150, 150, 3)	0
block1_conv1 (Conv2D)	(None, 150, 150, 64)	1792
block1_conv2 (Conv2D)	(None, 150, 150, 64)	36928
block1_pool (MaxPooling2D)	(None, 75, 75, 64)	0
block2_conv1 (Conv2D)	(None, 75, 75, 128)	73856
block2_conv2 (Conv2D)	(None, 75, 75, 128)	147584
block2_pool (MaxPooling2D)	(None, 37, 37, 128)	0
block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0

Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

256个3×3 Filter

VGG模型架构

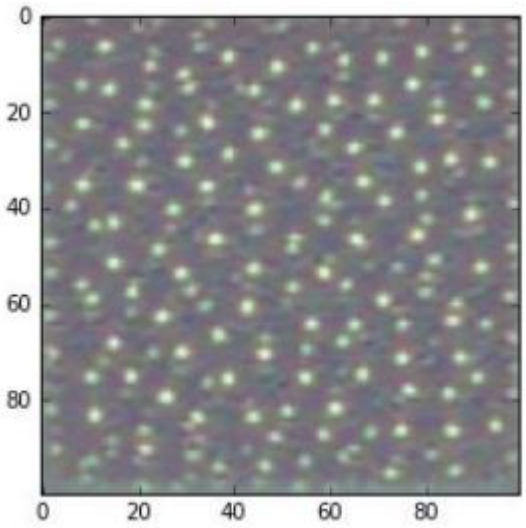


图 5-29　block3_conv1 层第 0 个通道具有最大响应的模式

block3_conv1层第0个过滤器响应的是波尔卡点（polka-dot）图案

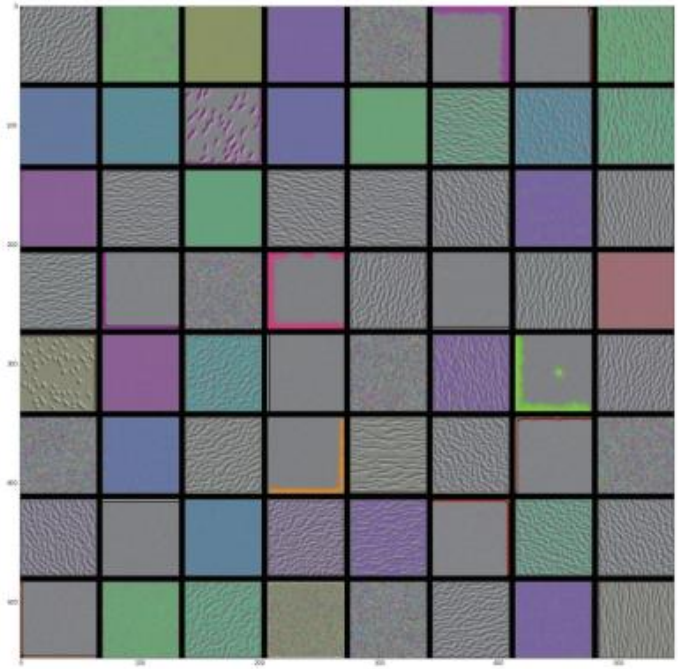


图 5-30　block1_conv1 层的过滤器模式

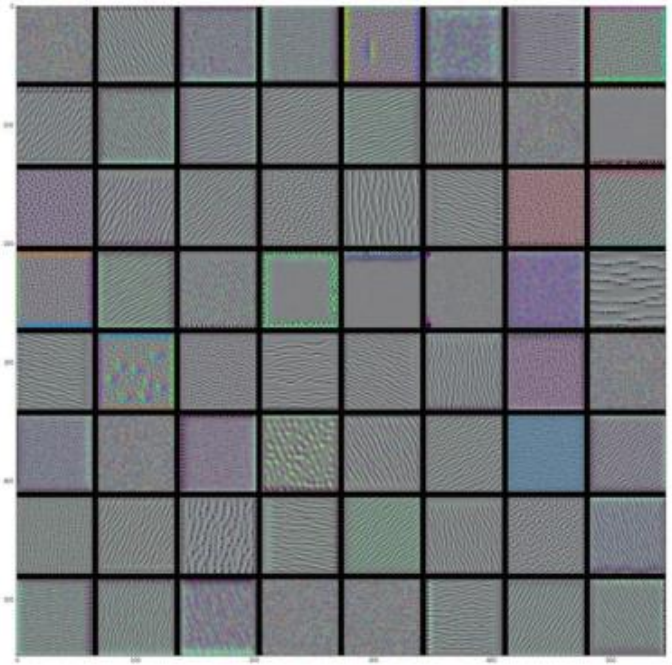


图 5-31　block2_conv1 层的过滤器模式

只查看每一层的前64个过滤器

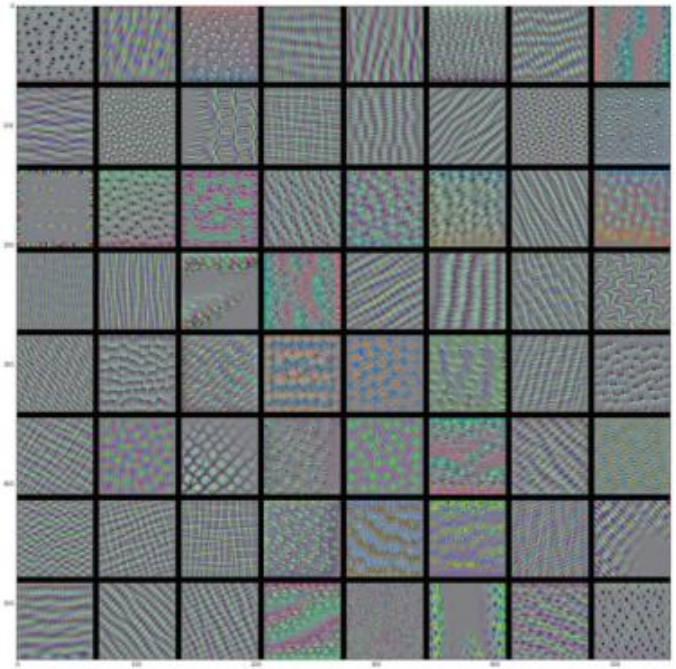


图 5-32　block3_conv1 层的过滤器模式

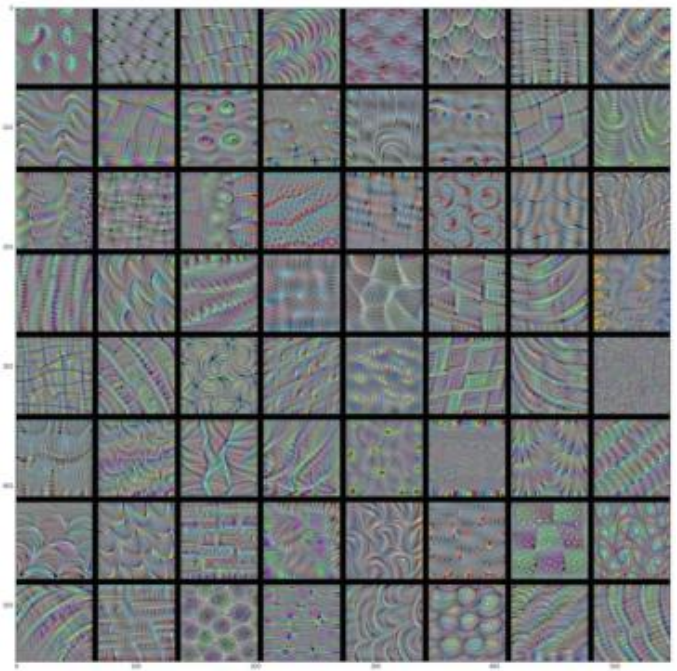


图 5-33　block4_conv1 层的过滤器模式

可视化图像中类激活的热力图

- VGG16模型
- 类激活图（CAM，class activation map）可视化，它是指对输入图像生成类激活的热力图。

图 5-34　非洲象的测试图像

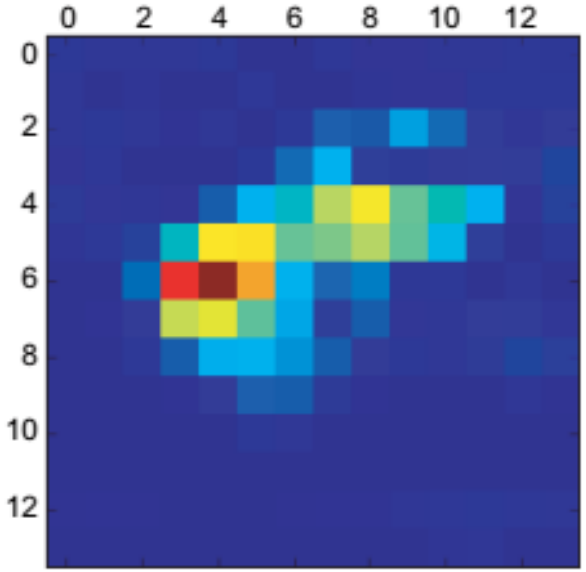


图 5-35　测试图像的“非洲象”类激活热力图

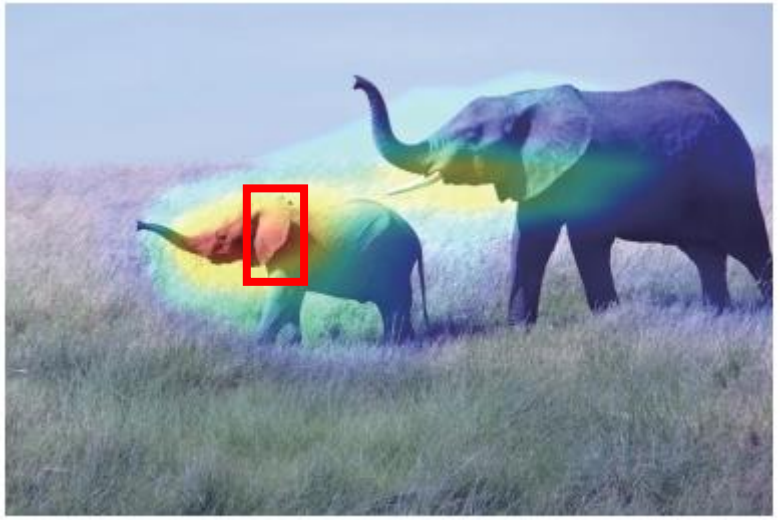


图 5-36　将类激活热力图叠加到原始图像上

- 这种可视化方法回答了两个重要问题：
  - 网络为什么会认为这张图像中包含一头非洲象？
  - 非洲象在图像中的什么位置？

[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

本章内容

- CNN（Convolutional Neural Network 卷积神经网络）
- 在小型数据集上从头开始训练一个CNN
- 使用预训练的CNN （迁移学习）
- CNN的可视化
- CNN Applications
  - Classification based on CNN
  - Object detection based on CNN

CNN Applications

Fast-forward to today: ConvNets are everywhere

Classification

Retrieval

Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

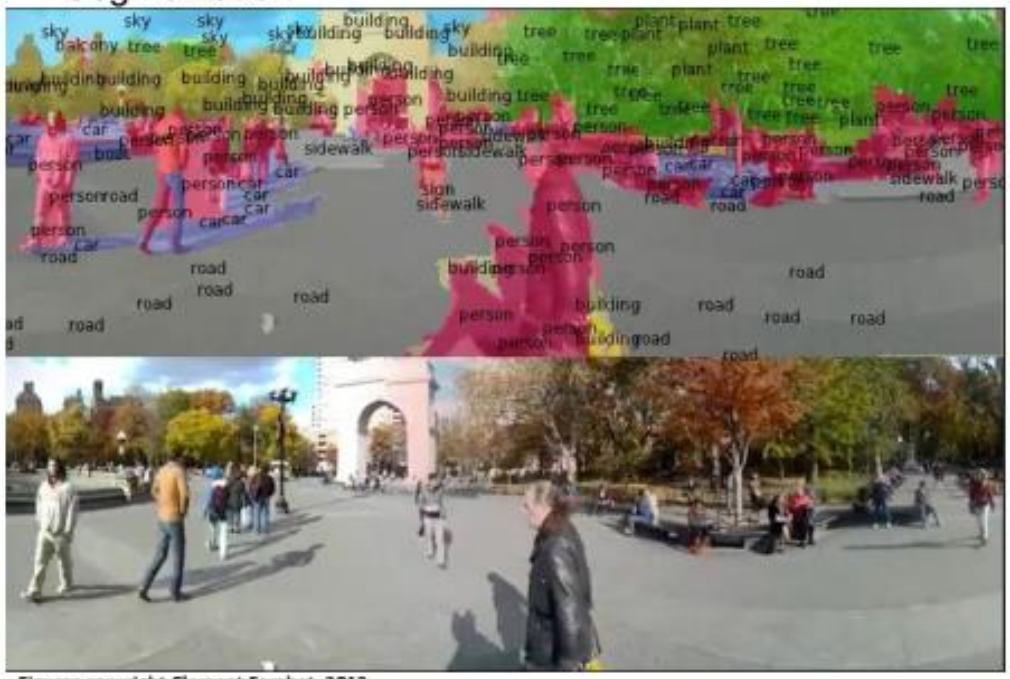
Fast-forward to today: ConvNets are everywhere

Detection

Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

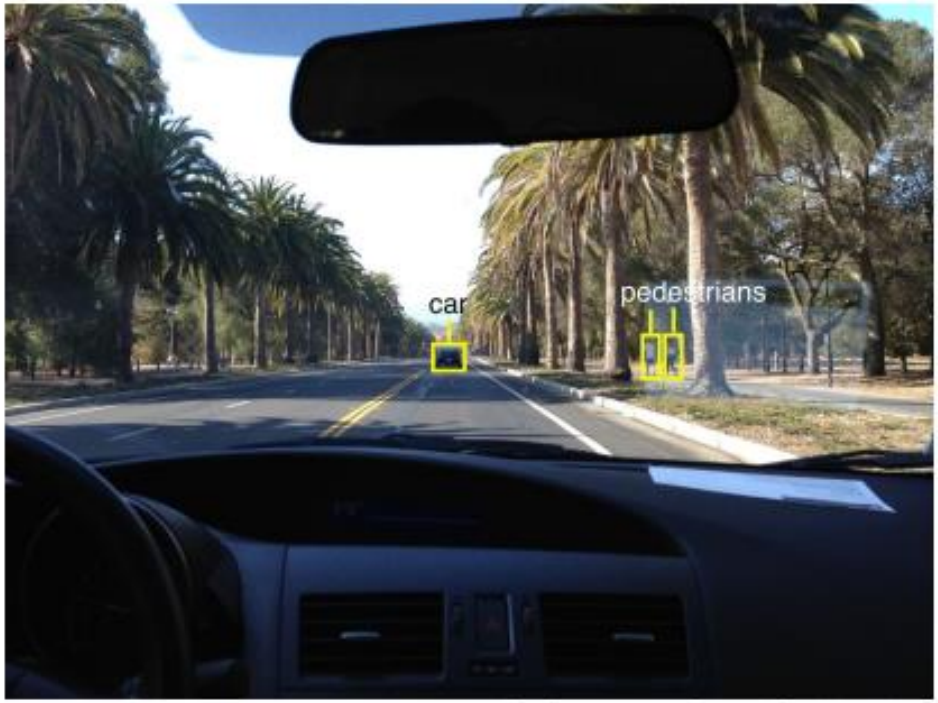


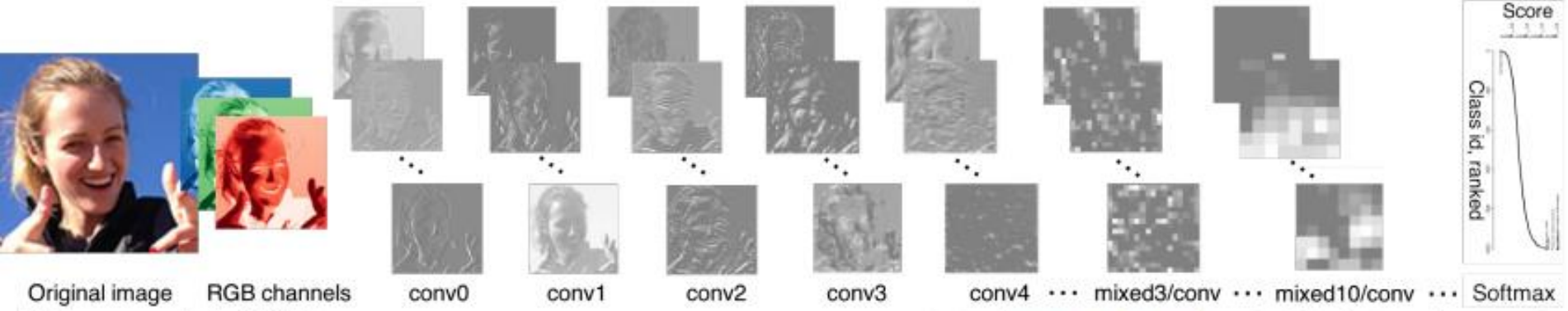
Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

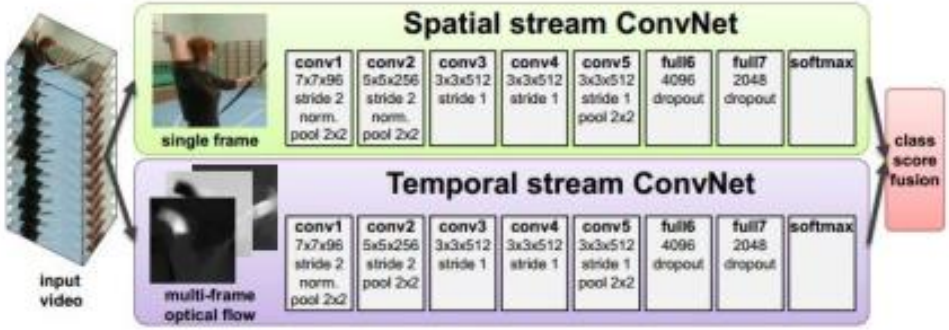
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

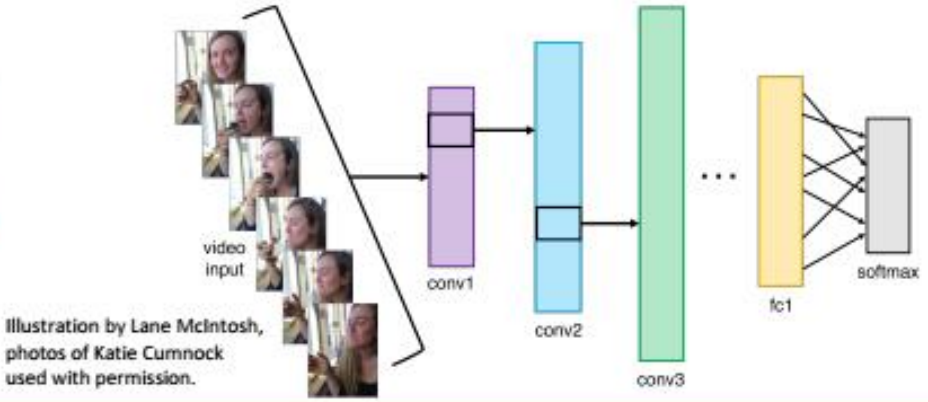
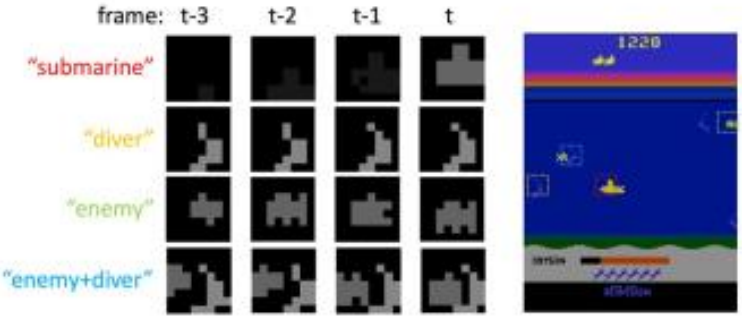


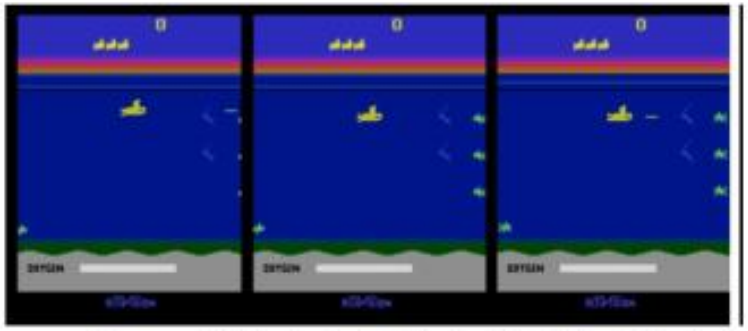
Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

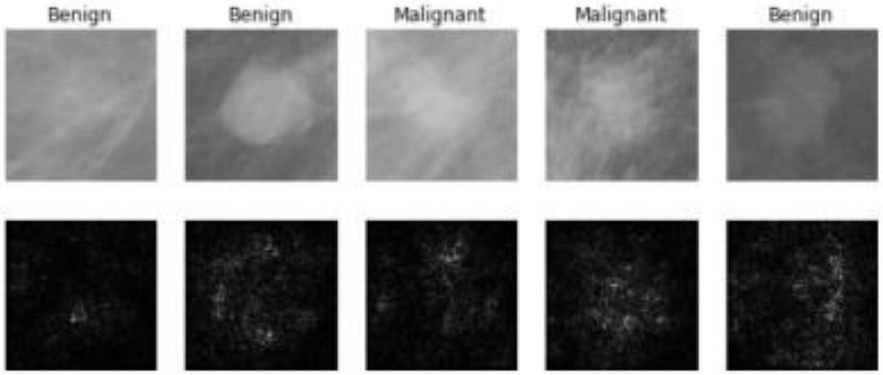




[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

[Dieleman et al. 2014]

[Sermanet et al. 2011]
[Ciresan et al.]

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

No errors

A white teddy bear sitting in the grass

A man riding a wave on top of a surfboard

Minor errors

A man in a baseball uniform throwing a ball

A cat sitting on a suitcase on the floor

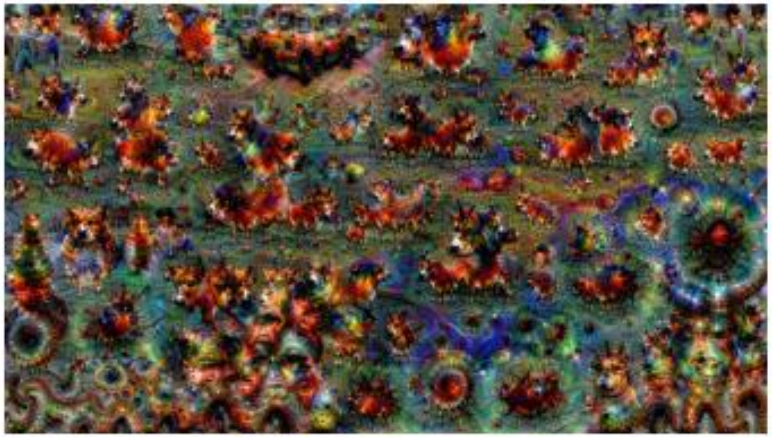
Somewhat related

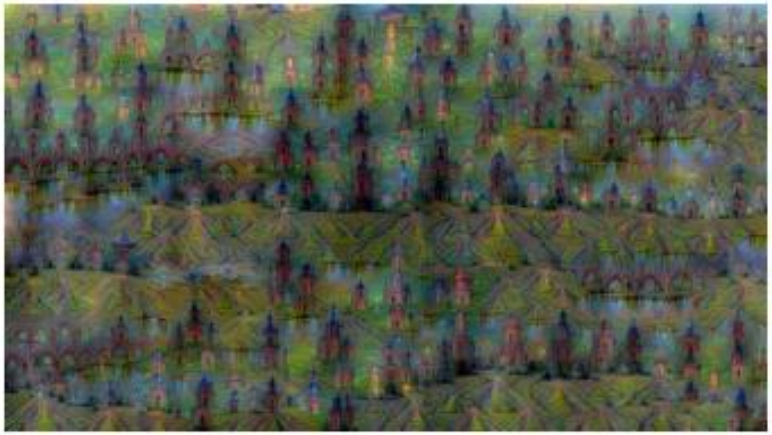
A woman is holding a cat in her hand

A woman standing on a beach holding a surfboard

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using NeuralTalk2





Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.

Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017; reproduced with permission

Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Focus

- Classification based on CNN
- Object detection based on CNN

IMAGENET Large Scale Visual Recognition Challenge

Steel drum

The Image Classification Challenge:

1,000 object classes

1,431,167 images

评价标准:Top-5 错误率

Output:		**Output:**	
Scale		Scale	
T-shirt		T-shirt	
Steel drum	✔	Giant panda	✗
Drumstick		Drumstick	
Mud turtle		Mud turtle	

Russakovsky et al. arXiv, 2014

Computer Vision Tasks

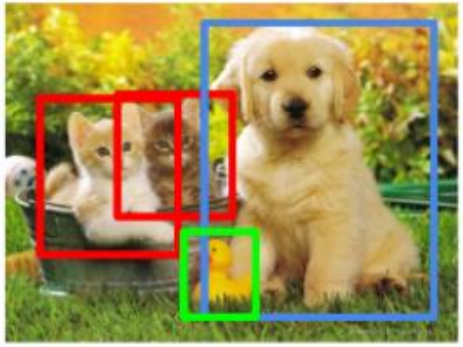
Classification

CAT

Classification + Localization

CAT

Object Detection



CAT, DOG, DUCK

Instance Segmentation

CAT, DOG, DUCK

Single object

Multiple objects

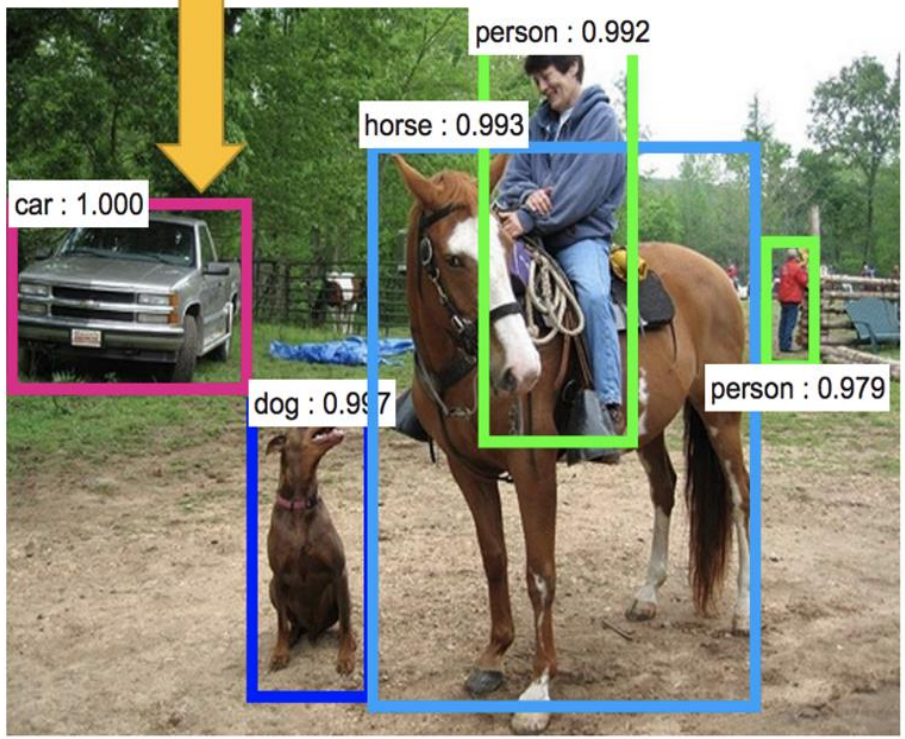
Object Detection = What, and Where

Localization

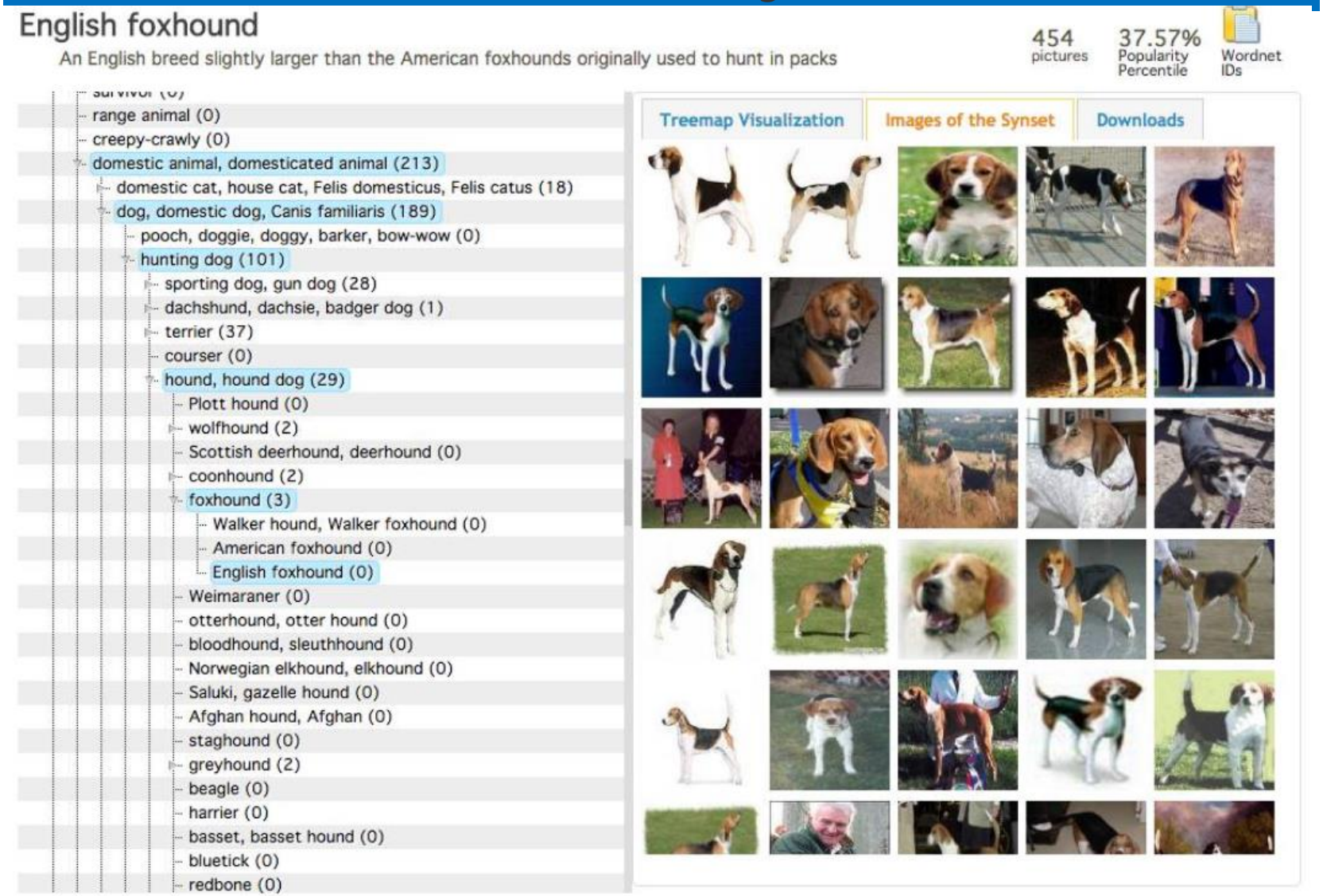
Where?

Recognition

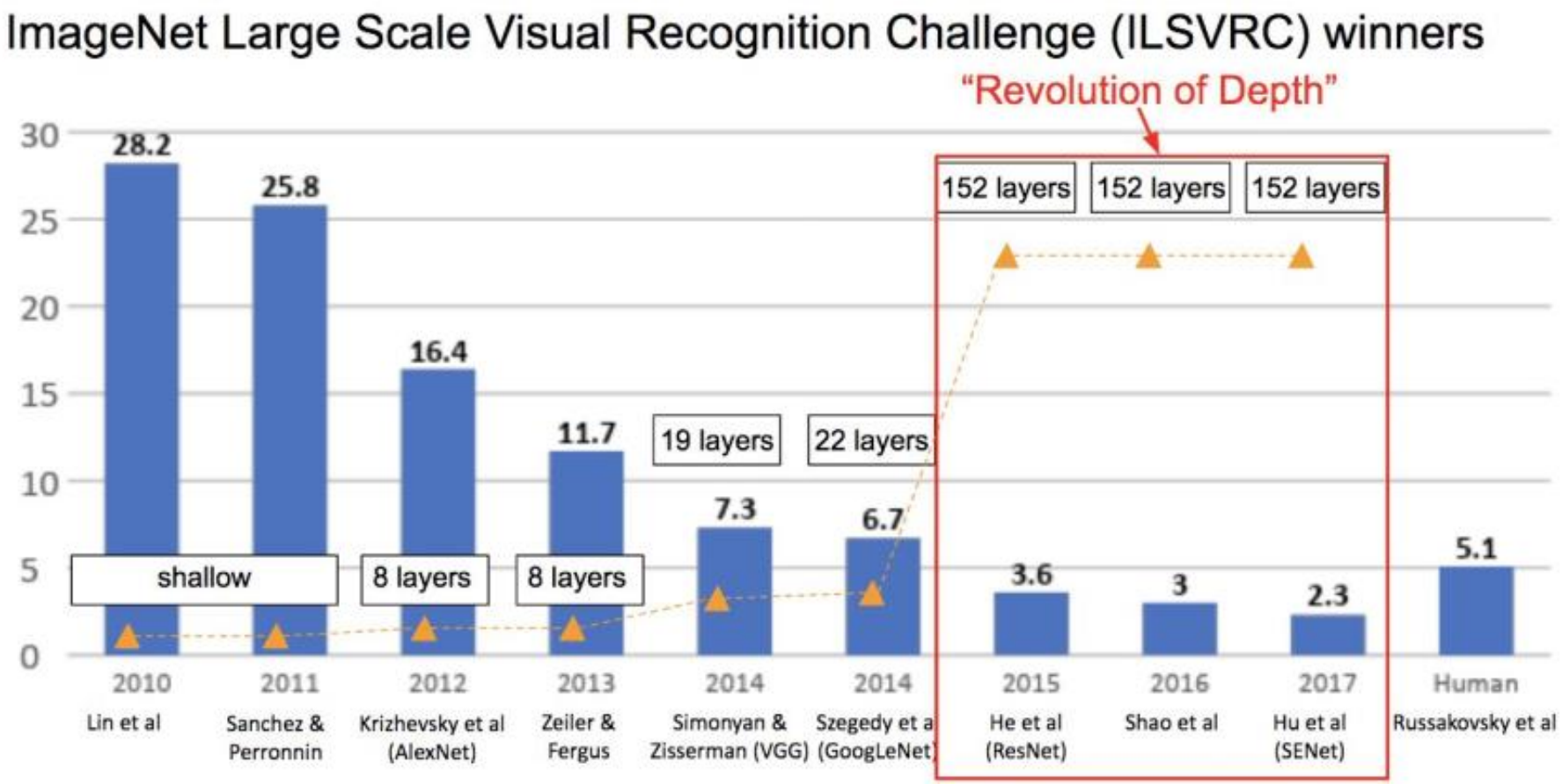
What?



截至 2016 年，ImageNet 中含有超过 1500 万由人手工注释的图片网址，也就是带标签的图片，标签说明了图片中的内容，超过 2.2 万个类别。其中，至少有 100 万张里面提供了边框（bounding box）。



ImageNet 数据集中“猎狐犬”的部分示例


ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners
"Revolution of Depth"
30
25
20
15
10
5
0
28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
152 layers
152 layers
2010
2011
2012
2013
2014
2014
2015
2016
2017
Human
Lin et al
Sanchez & Perronnin
Krizhevsky et al (AlexNet)
Zeiler & Fergus
Simonyan & Zisserman (VGG)
Szegedy et a (GoogLeNet)
He et al (ResNet)
Shao et al
Hu et al (SENet)
Russakovsky et al


IMAGENET + kaggle
ImageNet Object Localization Challenge
ImageNet Object Detection Challenge
ImageNet Object Detection from Video Challenge

WebVision竞赛：超越ILSVRC

- 侧重图像学习和理解
- WebVision 数据集：
  - 使用与 2012 年 ImageNet 竞赛相同的 1000 个类别，涵盖了直接从网络收集到的 240 万张现代图像（包括谷歌图像搜索中获得的 100 万张，以及来自 Flickr 的 140 万张图像）和元数据。
  - 通过苏黎世科技大学计算机视觉实验室的网络数据团队收集的。这一数据集的开发得到了谷歌研究院苏黎世分部的支持。

Focus

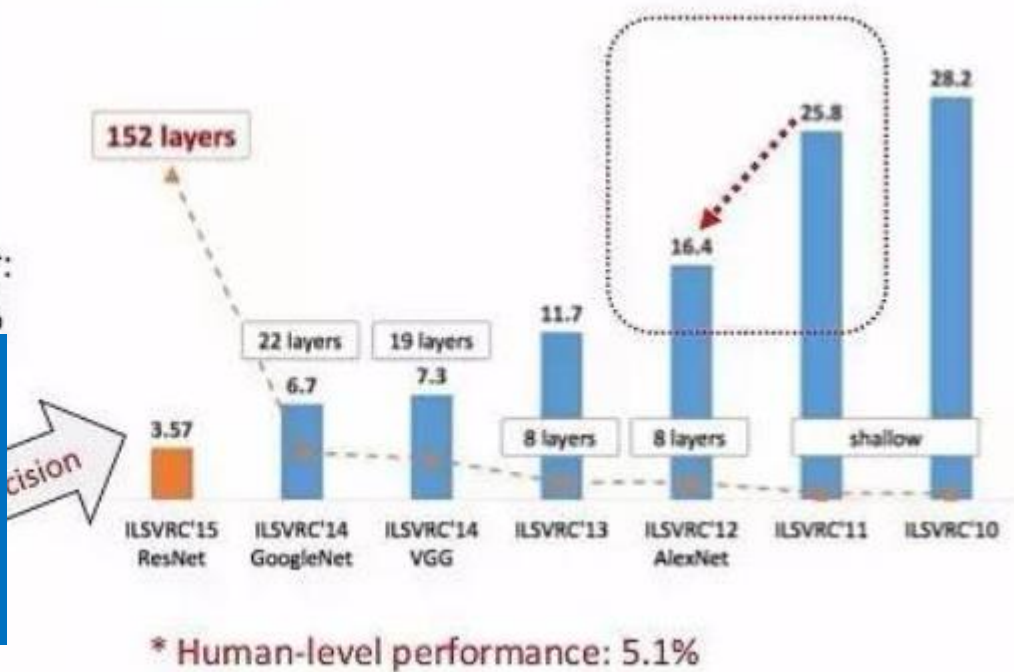
- Classification based on CNN

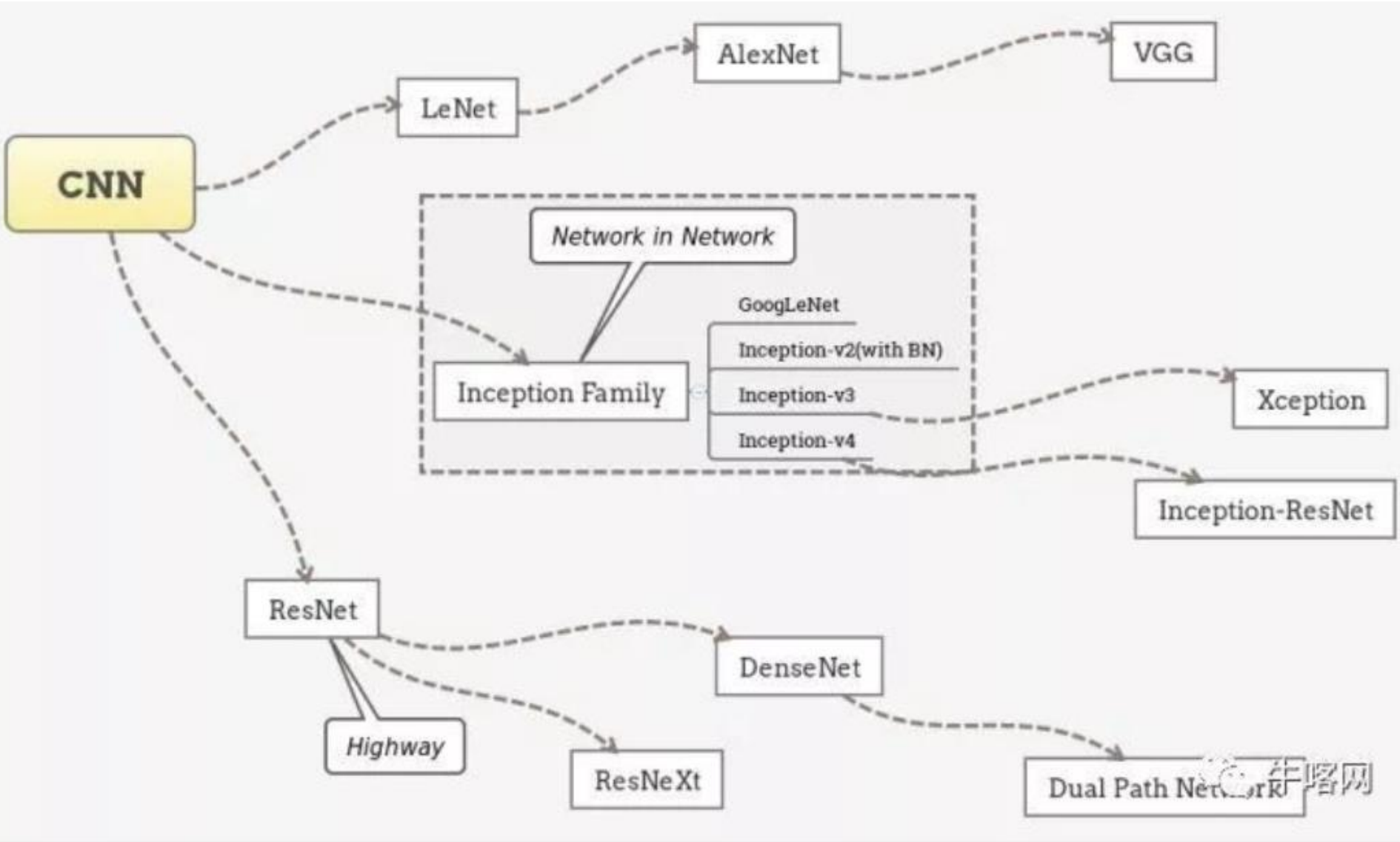
- Object detection based on CNN

Classification based on CNN

- LeNet-5 [LeCun et al., 1998]
- AlexNet [Krizhevsky et al., 2012]
  - 使用CNN解决图像分类问题，在ILSVRC'12中大大获胜并大大提高了State-of-art的准确率，促进了CNN在图像分类领域的快速发展
- VGG
- Inception系列
- ResNet

从以下三个方面把握：
1）网络整体结构是怎样的？
2）创新点是什么？（包含网络结构的创新和比较新颖的激活函数等方法）
2）创新点可以带来什么好效果？为什么？



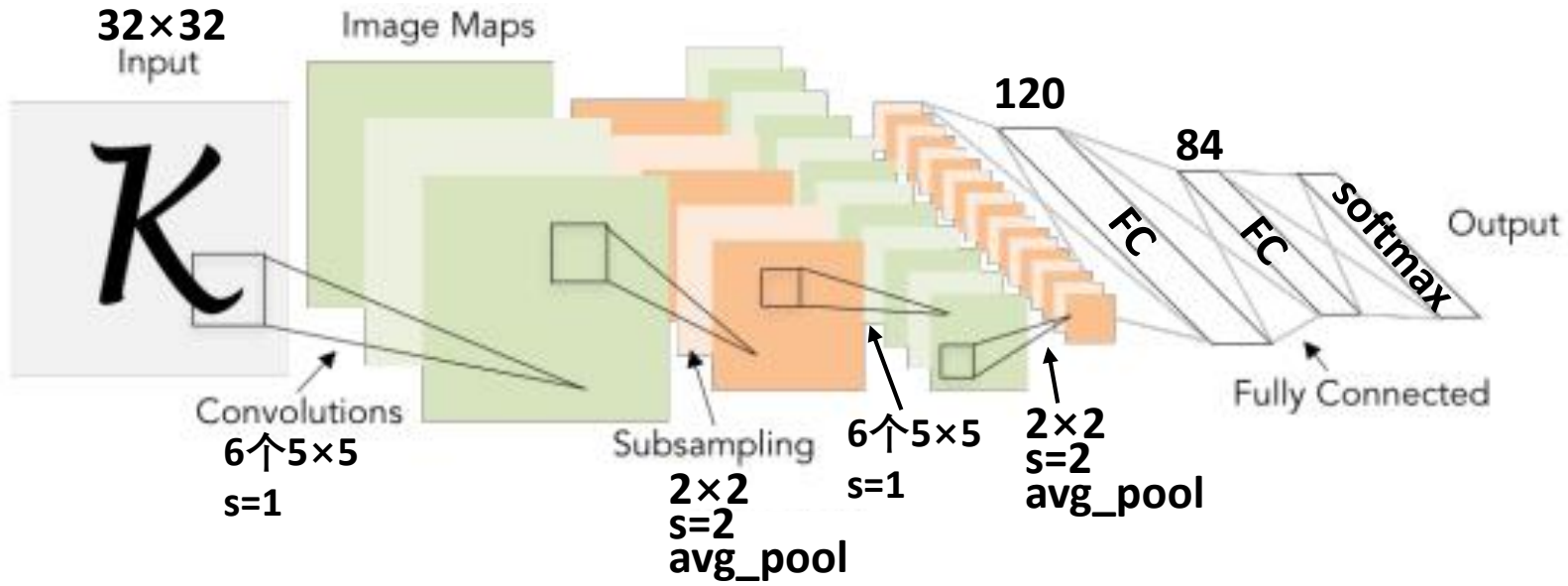

CNN
LeNet
AlexNet
VGG
Network in Network
GoogLeNet
Inception-v2(with BN)
Inception Family
Inception-v3
Inception-v4
Xception
Inception-ResNet
ResNet
DenseNet
Highway
ResNeXt
Dual Path Network

LeNet-5

Q1：包含哪5层？
Q2：每层的输出的shape是？
Q3：总参数量是多少？

Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]

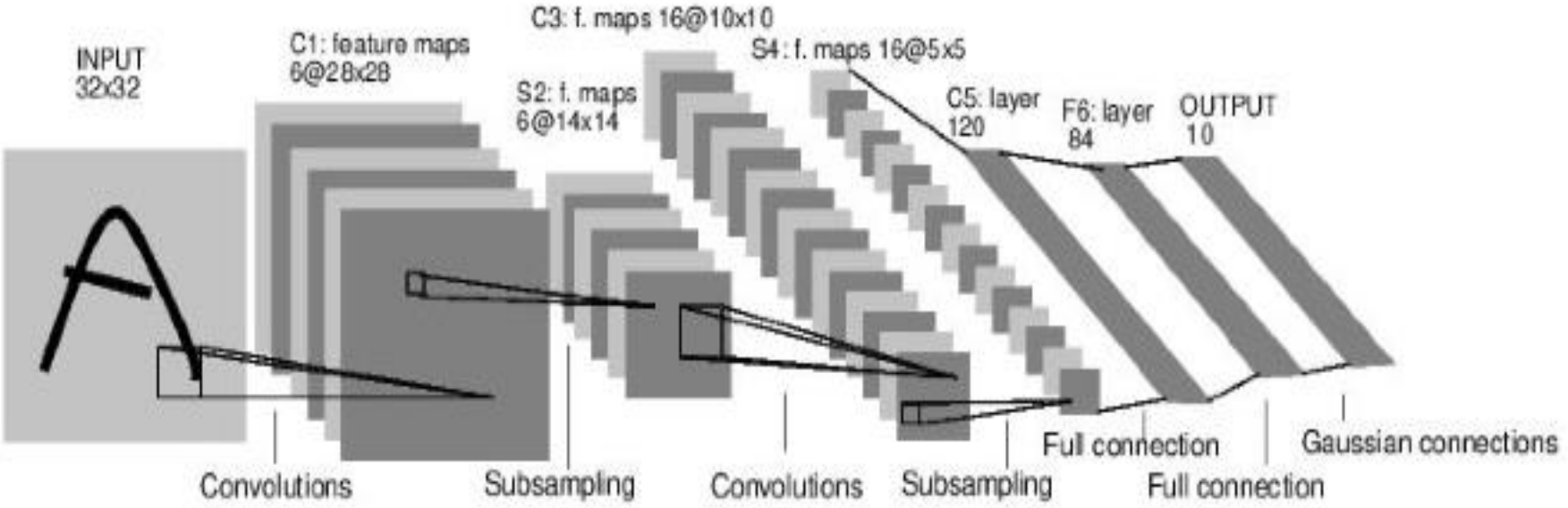
1. 池化层使用avg pooling
2. 激活函数使用sigmoid函数和tanh函数



1. 随着网络越来越深，图像的宽度和高度在缩小，而通道数一直在增加。
2. 网络架构为：[CONV-POOL-CONV-POOL-FC-FC]

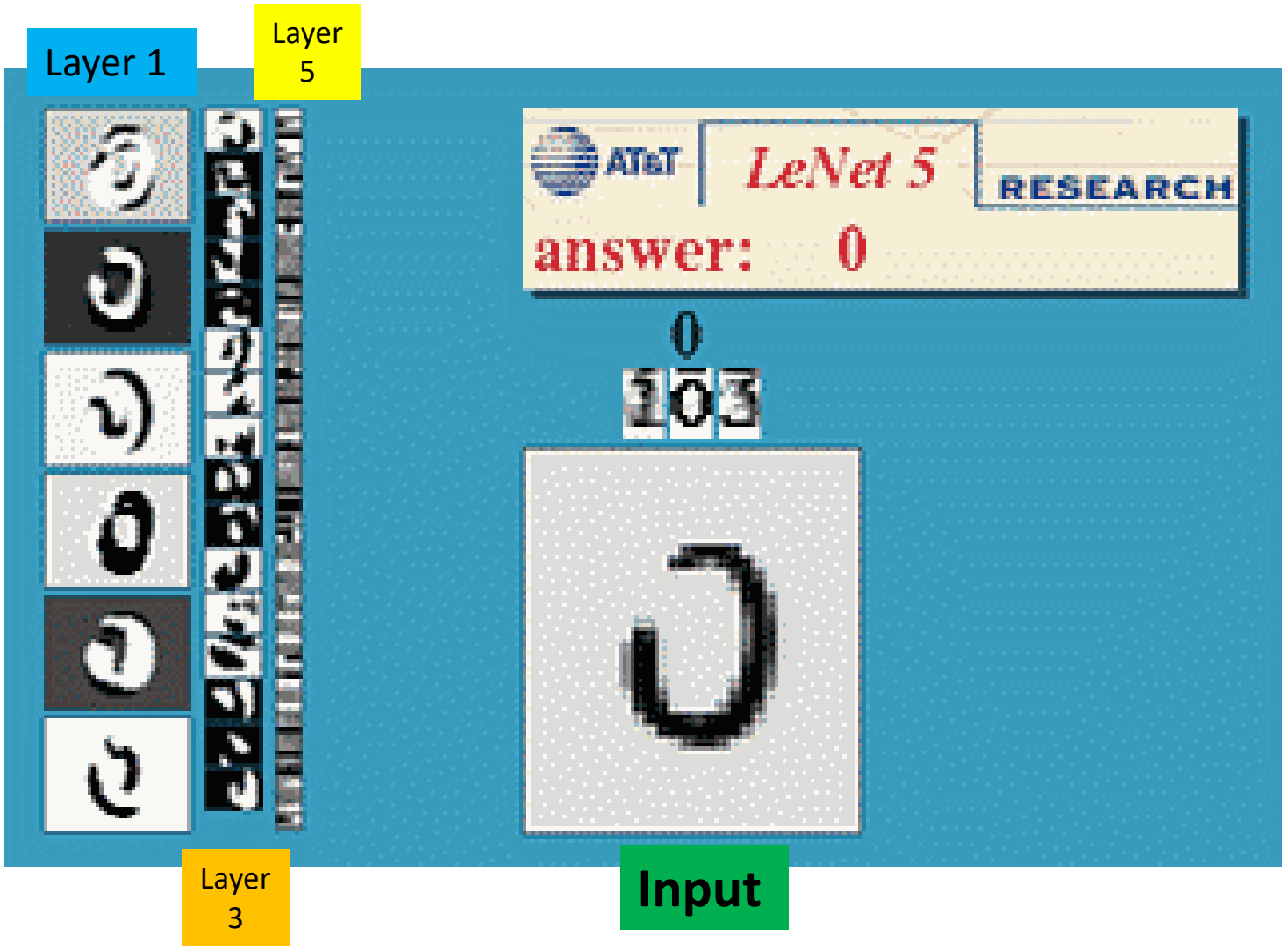
Case Study: LeNet-5

[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]

avg pooling

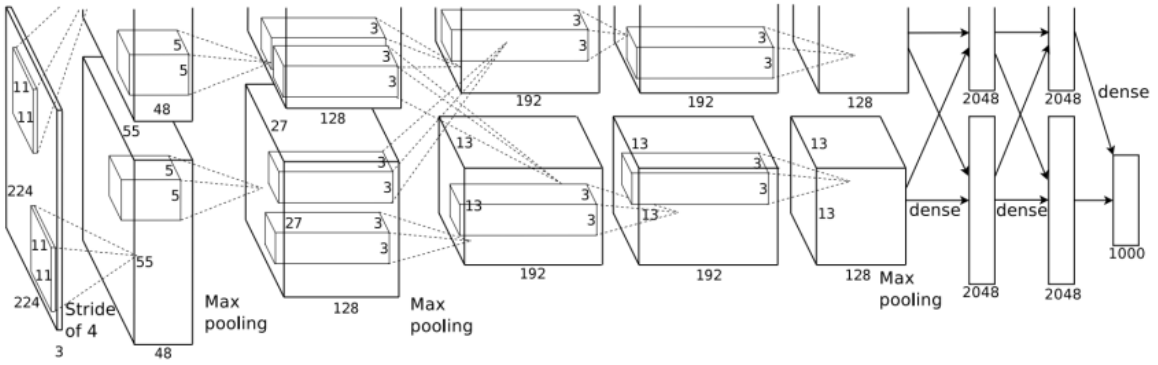


http://yann.lecun.com/exdb/lenet/

Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

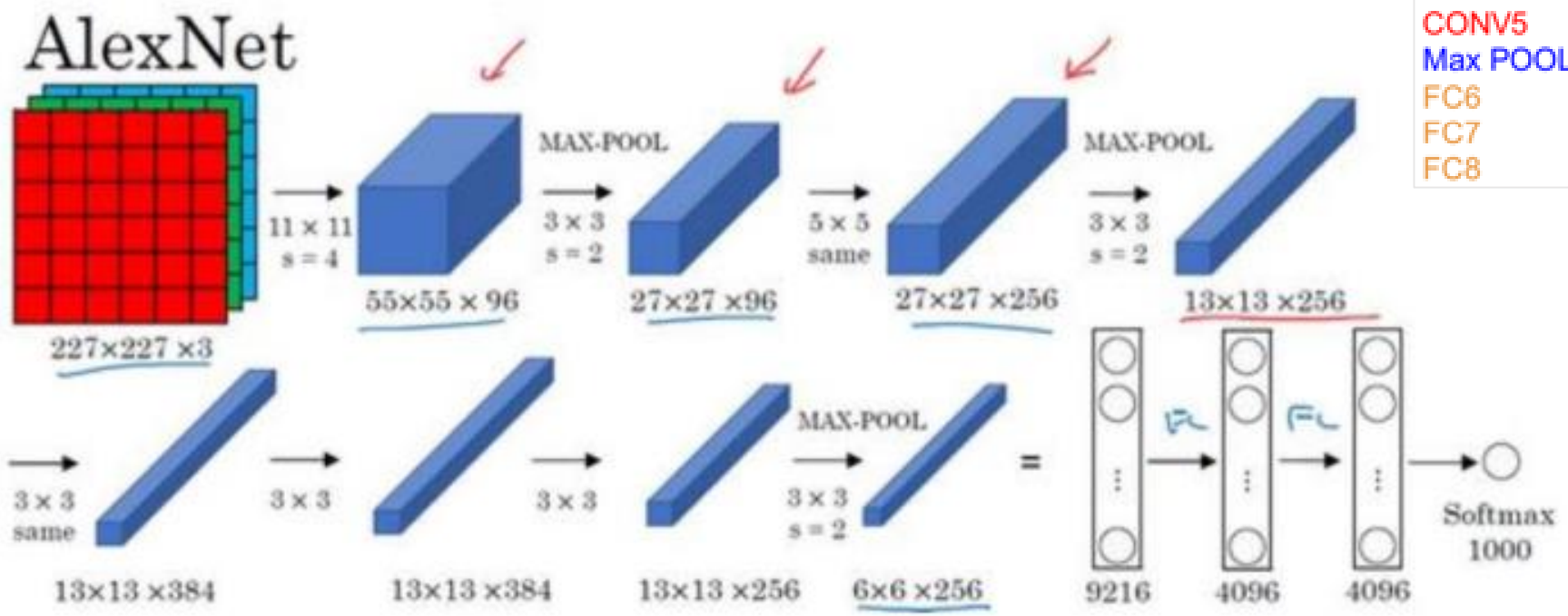


论文中给出的网络结构（用两台GPU训练）

Paper: ImageNet Classification with Deep Convolutional Neural Network (2012)
Author: Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton
Performance: error rate on ImageNet, top1: 37.5%, top 5: 17.0%

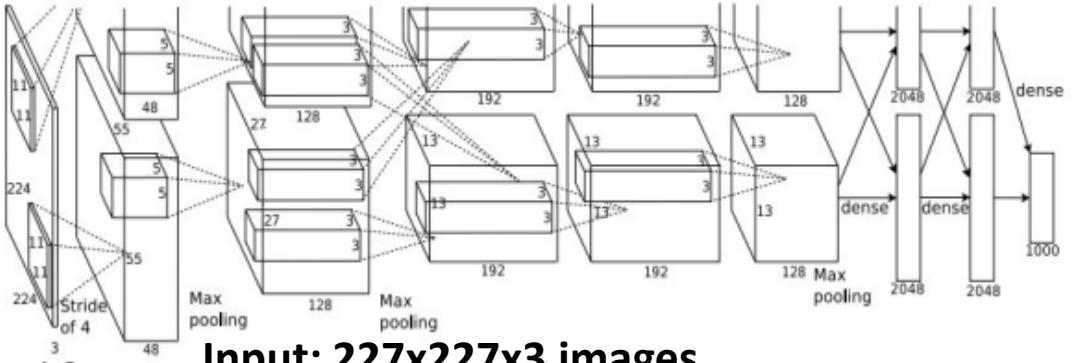
- Similar to LeNet, but much bigger
- Use ReLU
- Multiple GPU
- Local response normalization: Normalization cross channels
- Use max-pooling
- Use drop out

Architecture
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



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)

Input: 227x227x3 images

Q1: for CONV1 layer, what is the output volume size (not counting bias)? What is the total number of parameters in this layer?

A1: Output volume[55x55x96]

Parameters: (11*11*3)*96 =35K

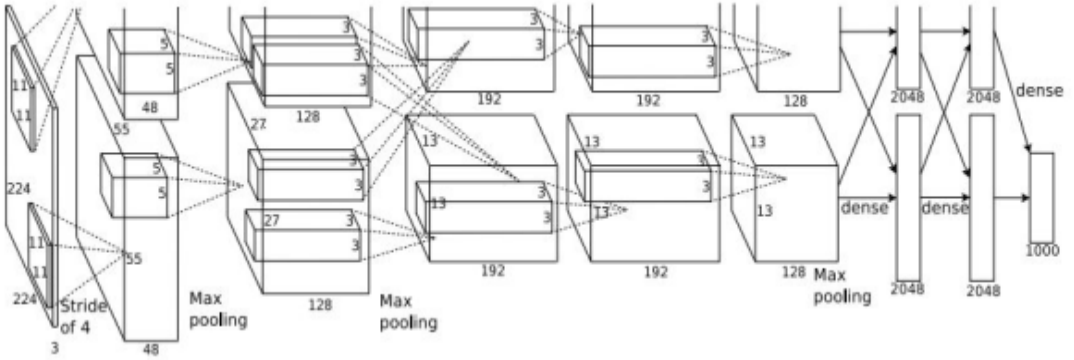
Q2: for Max Pool1 layer, what is the output volume size? what is the number of parameters in this layer?

A2: Output volume: 27x27x96

Parameters: 0!

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)

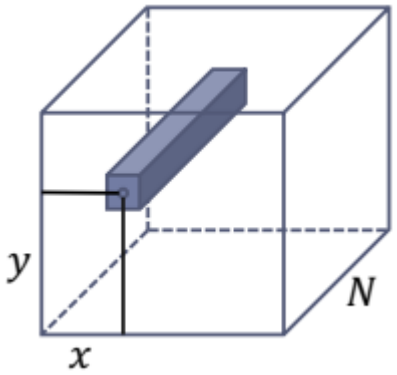
Details/Retrospectives:

- 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%

AlexNet

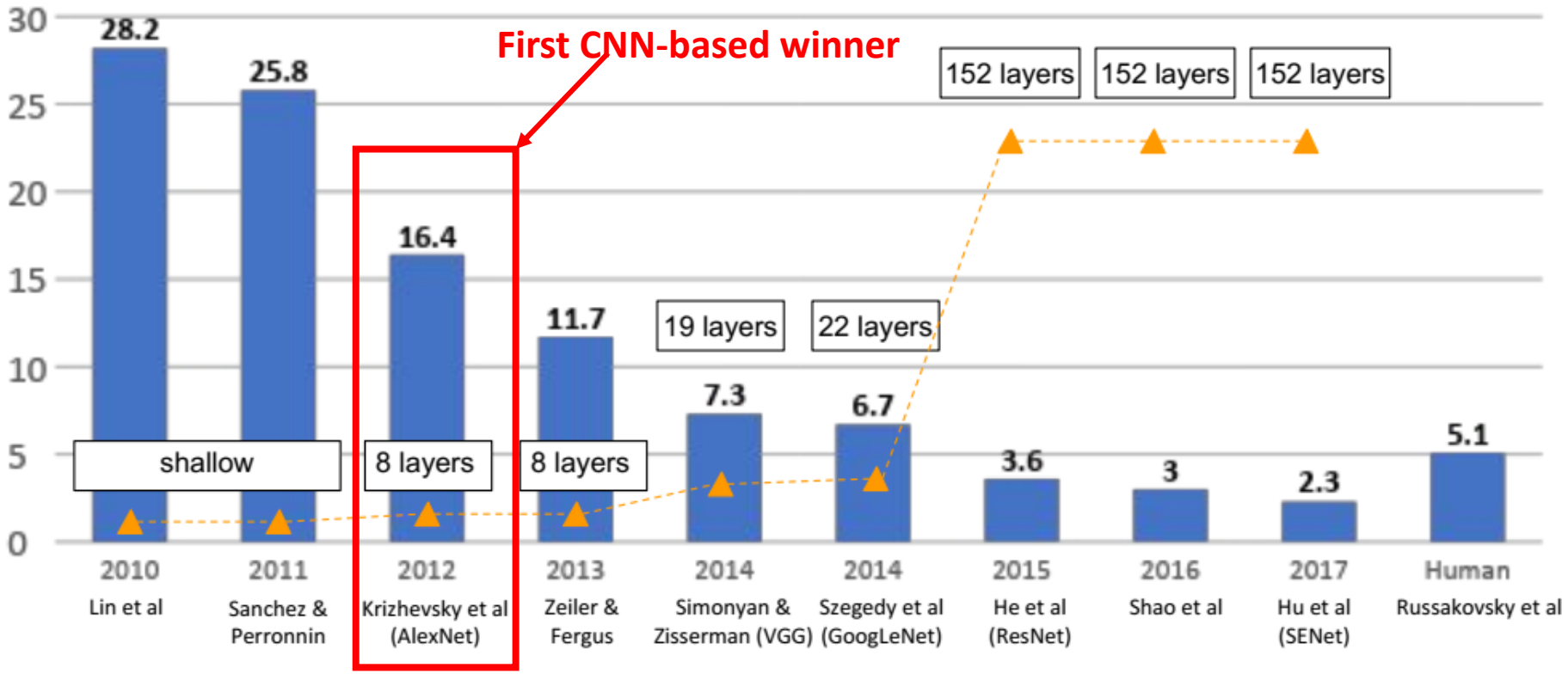
- Local Response Normalization (LRN)
- 局部响应归一化:

$$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^{2} \right)^{\beta}$$

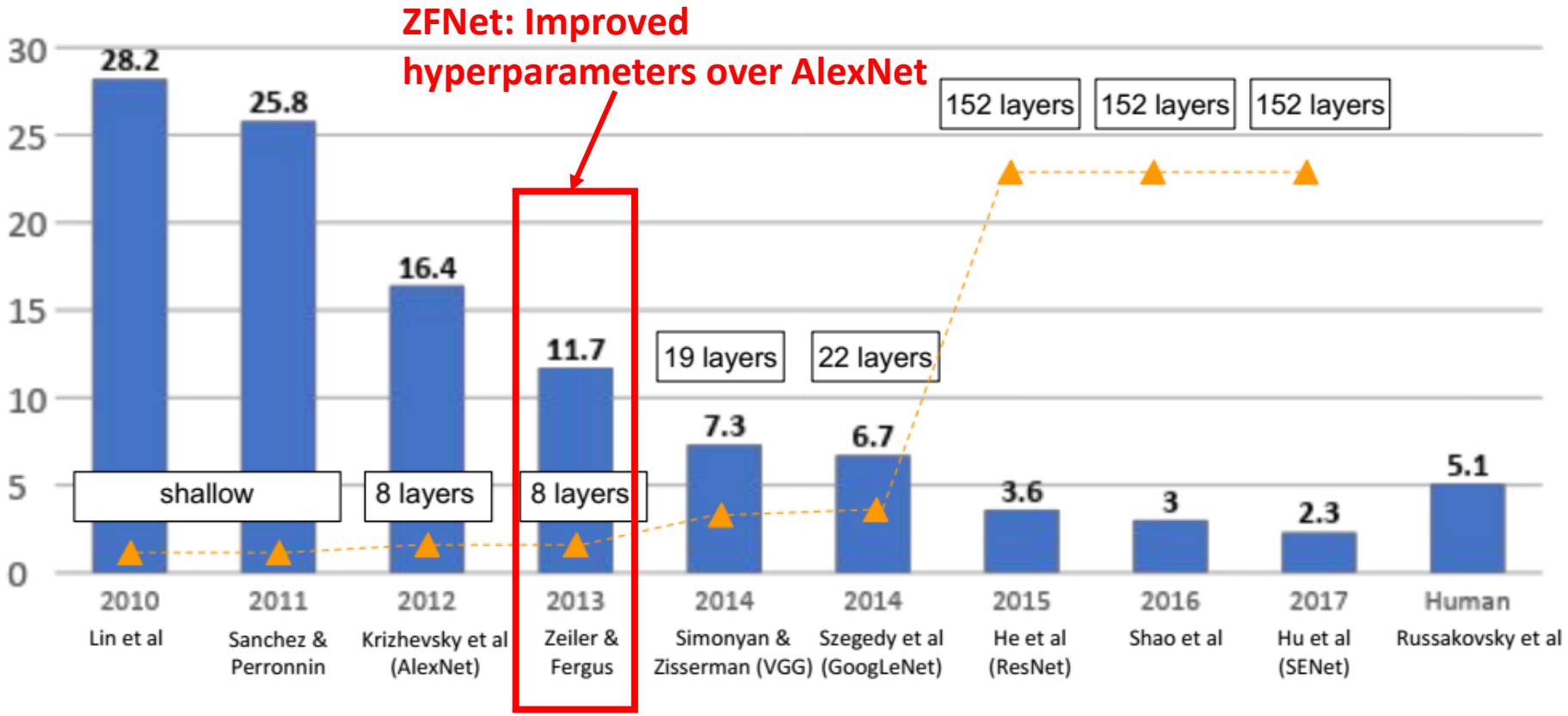


- 对图像的每个“位置”，提升高响应特征，抑制低响应特征;
- 减少高激活神经元数量，提高训练速度，抑制过拟合;
- 但后来被研究者发现无明显效果，故现在很少使用

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



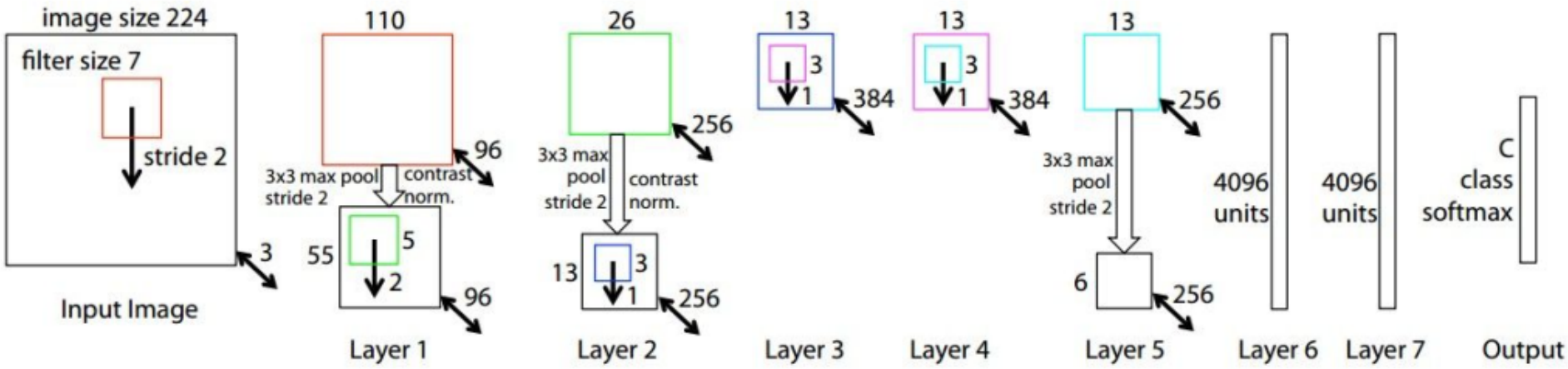
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



趋势：更小的卷积核，更大的通道数.....

ZFNet

[Zeiler and Fergus, 2013]

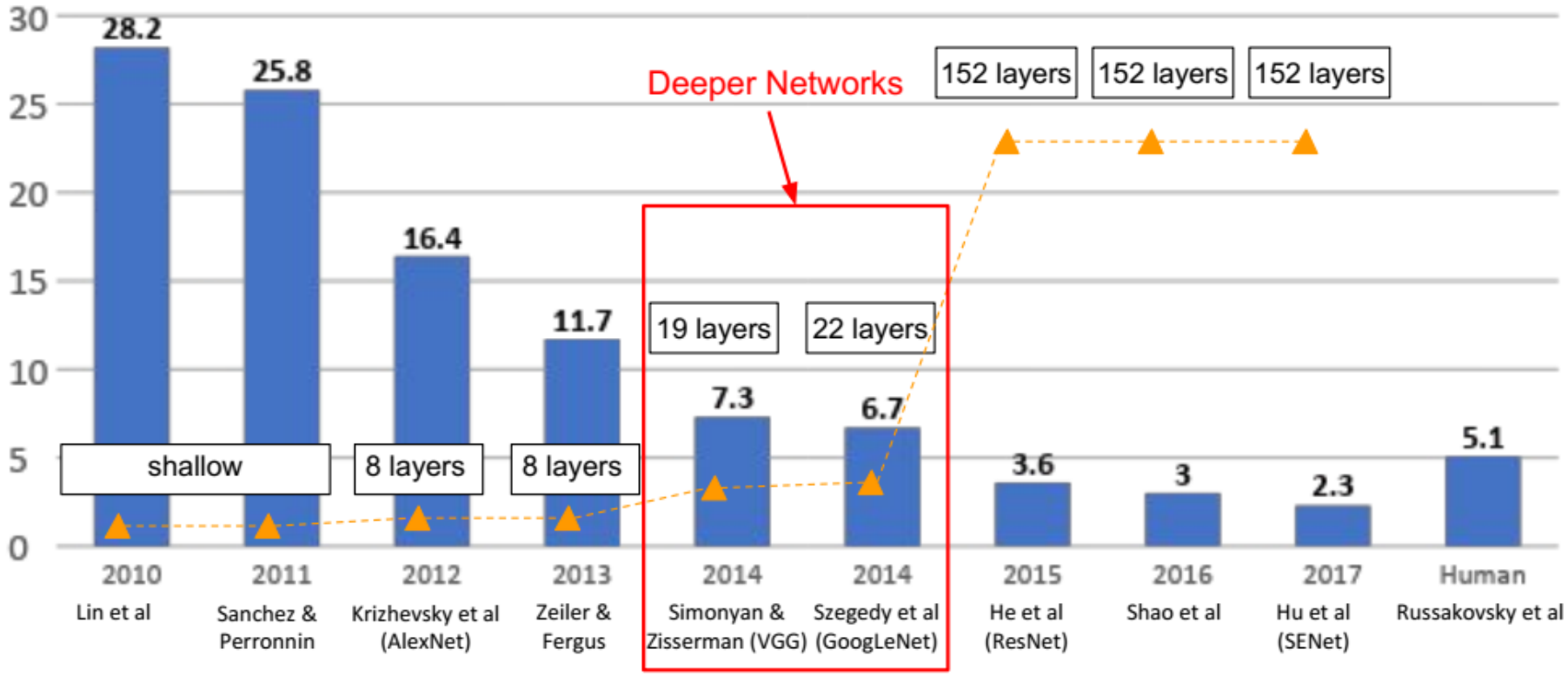


AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 16.4% -> 11.7%

网络能否更深?

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



VGG (from Lab: Visual Geometry Group)

- Paper: Very Deep Convolutional Networks for Large-Scale Image Recognition (2014)
- Author: K. Simonyan, A. Zisserman
- Test: error rate on ImageNet, top1: 24.7%, top5: 7.5%

Case Study: VGGNet

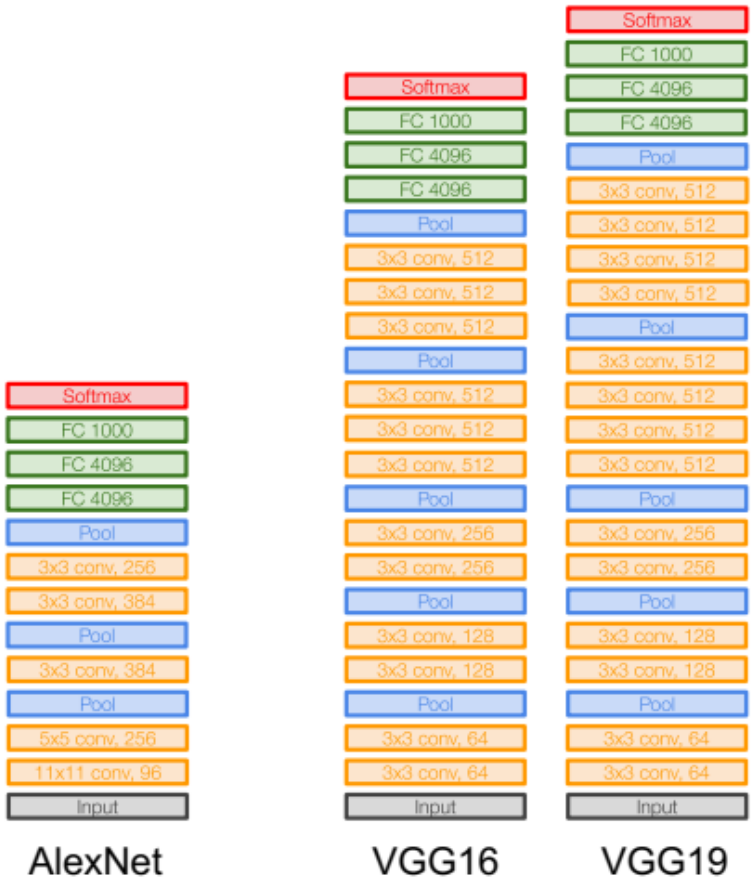
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

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

11.7% top 5 error in ILSVRC'13 (ZFNet)
-> 7.3% top 5 error in ILSVRC'14



规整的卷积-池化 结构

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				224×224×3	
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool				112×112×64	
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool				56×56×128	
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				28×28×256	
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				14×14×512	
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				7×7×512	

卷积层：负责数据体深度变换（控制特征图数量）

池化层：负责数据体长宽变换（控制特征图大小）

图像缩小的比例和通道数增加的比例呈现规律性：随着网络的加深，图像的高度和宽度都在以一定的规律不断缩小，每次池化后刚好缩小一半，而通道数量在不断增加，而且刚好也是在每组卷积操作后增加一倍。

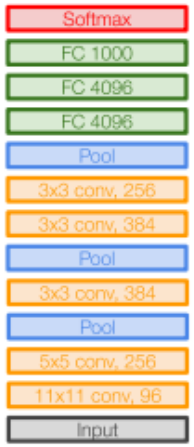
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

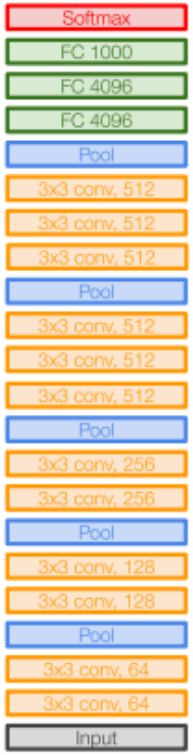
Small filters, Deeper networks

Q: Why use smaller filters? (3x3 conv)

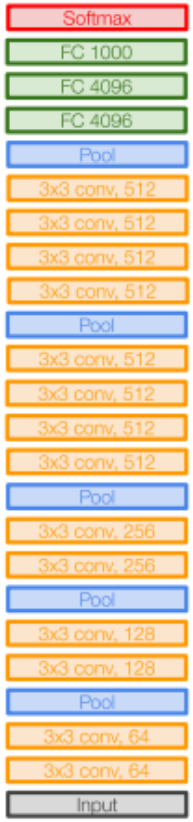
- Stack of three 3x3 conv (stride 1) layers has same effective receptive field as one 7x7 conv layer;
- But deeper, more non-linearities;
- And fewer parameters: $3 \times (3^2 \times C^2)$ vs. $7^2 \times C^2$ for C channels per layer.
  - Assuming that both the input and the output of a three-layer 3×3 convolution stack has C channels



AlexNet

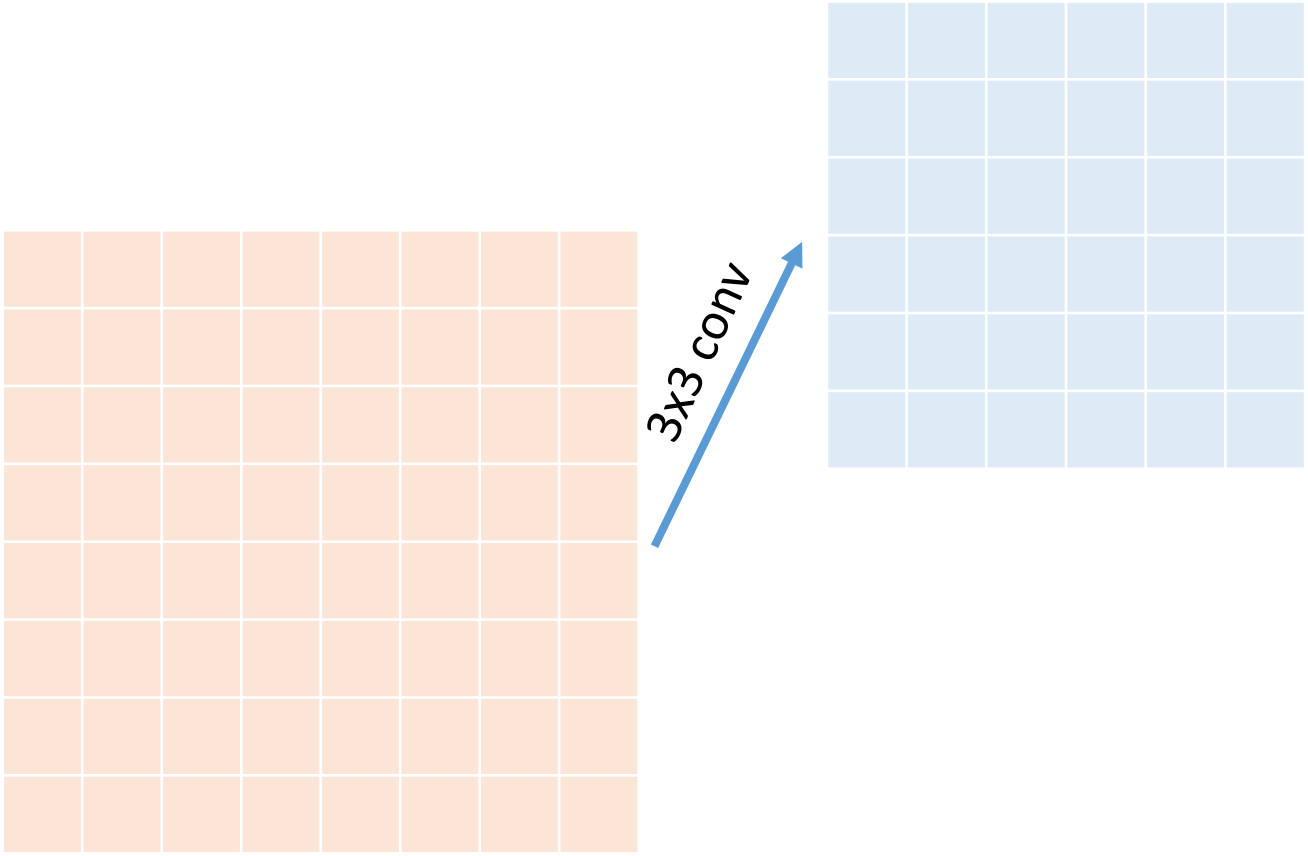


VGG16

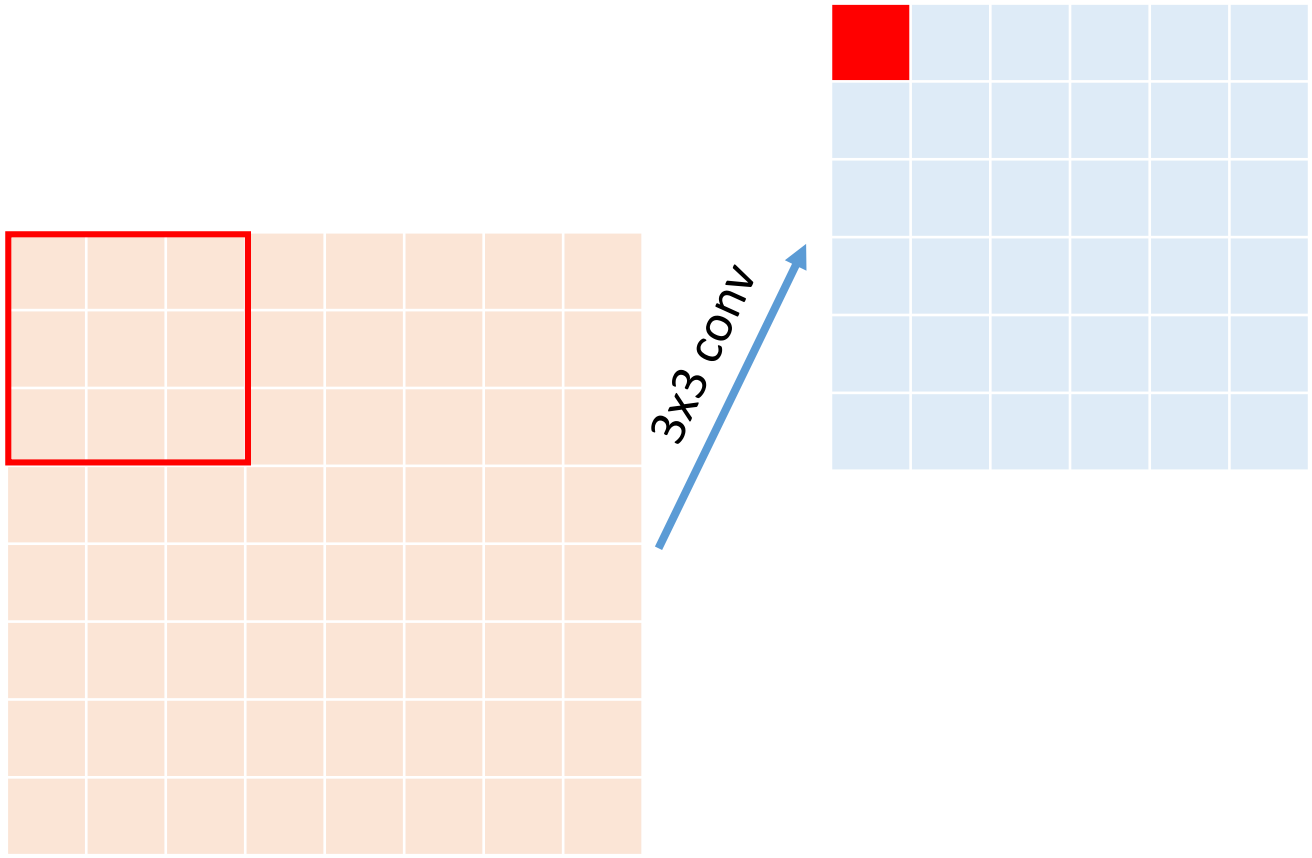


VGG19

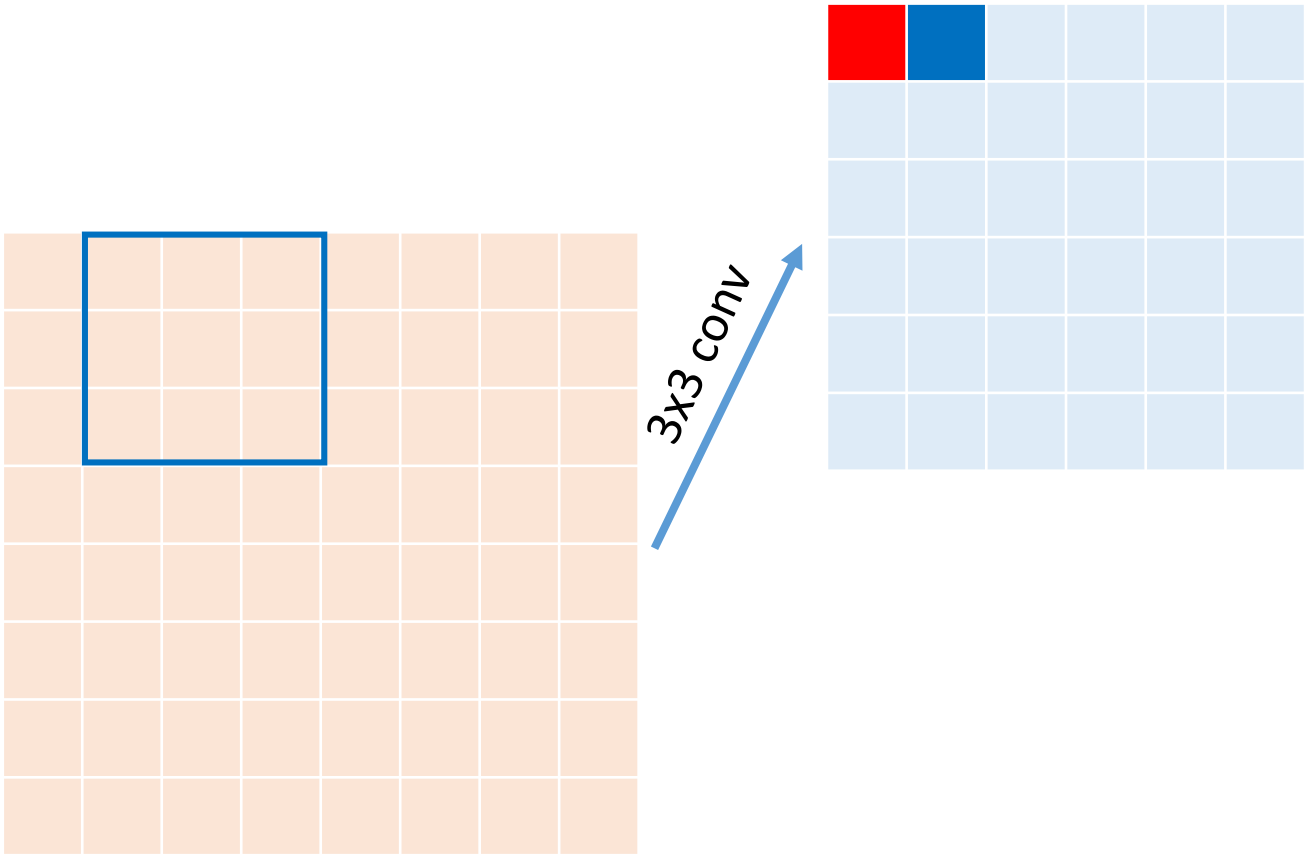
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



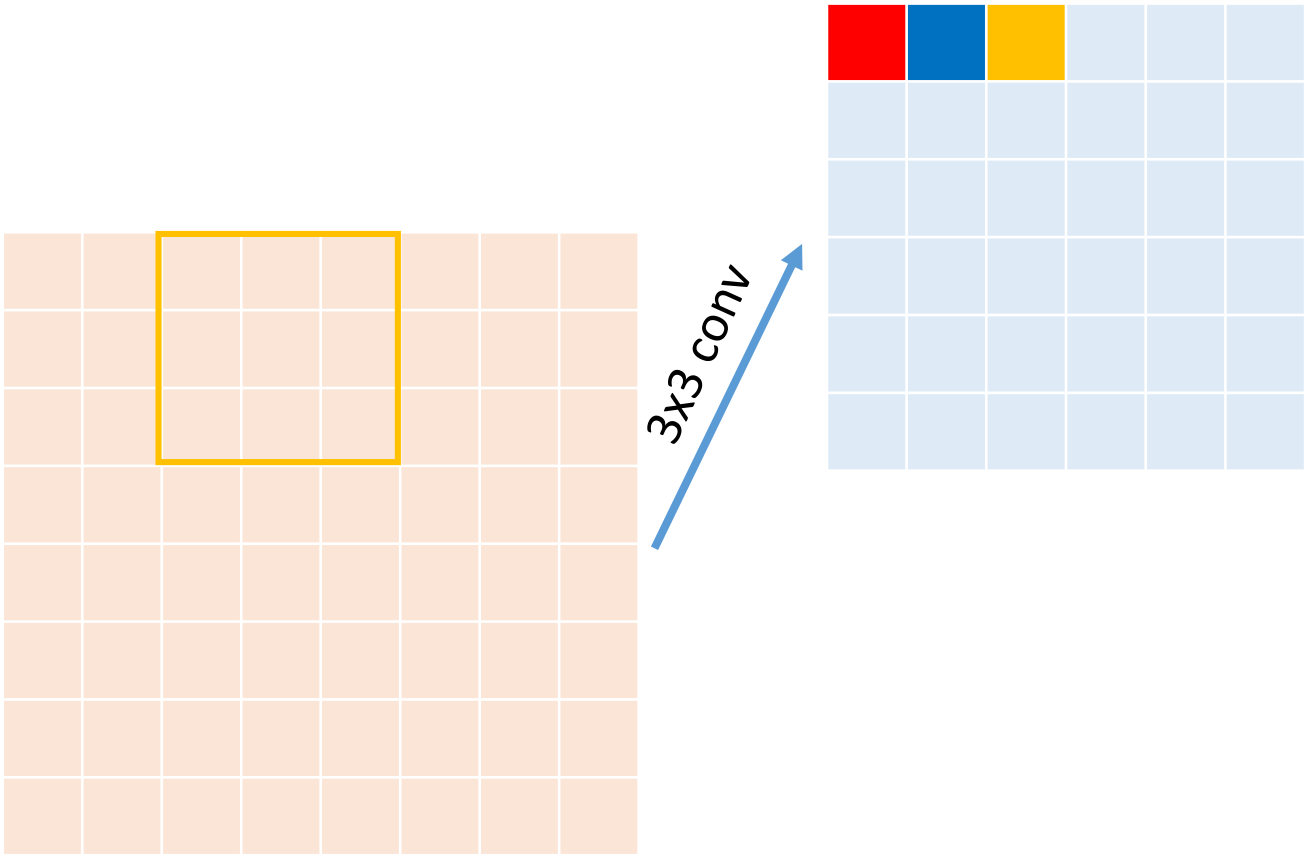
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



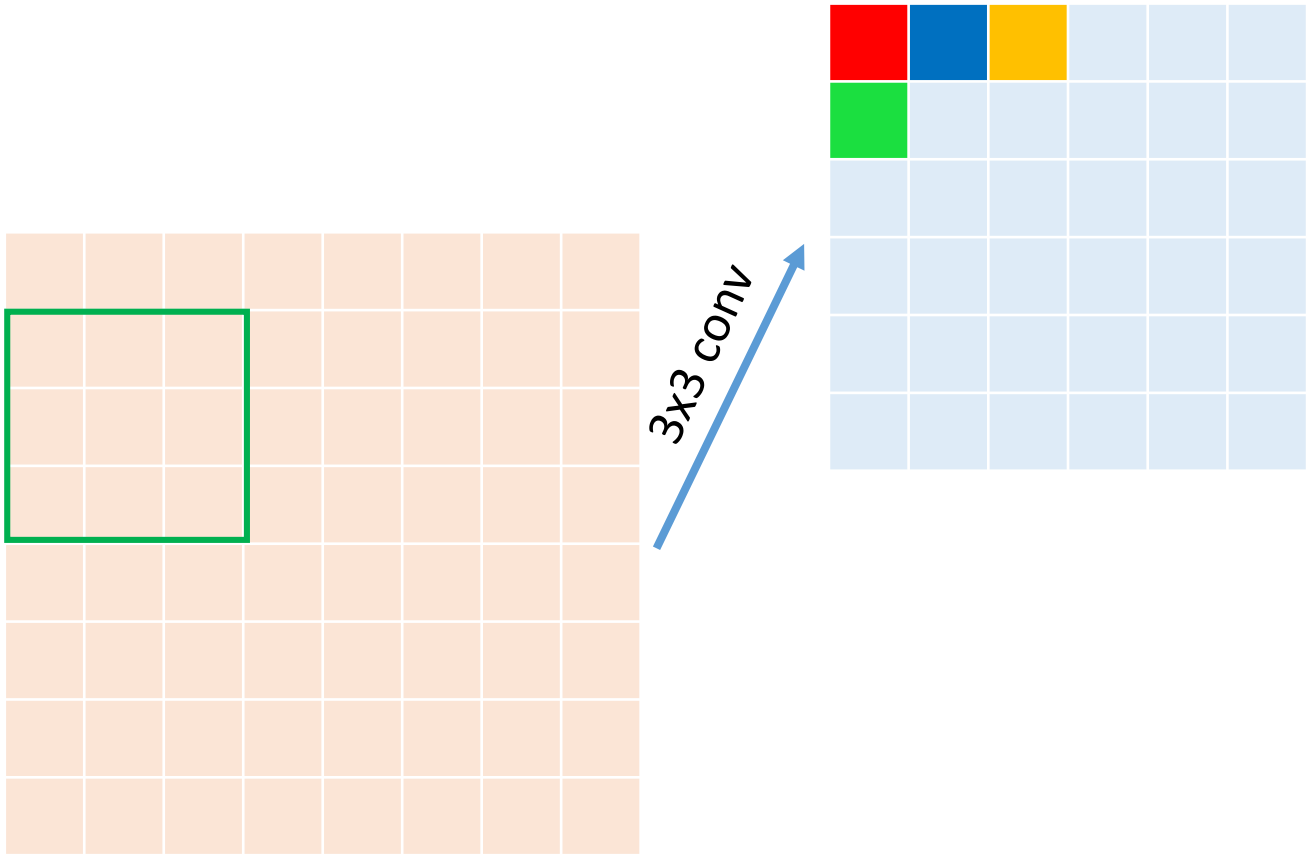
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



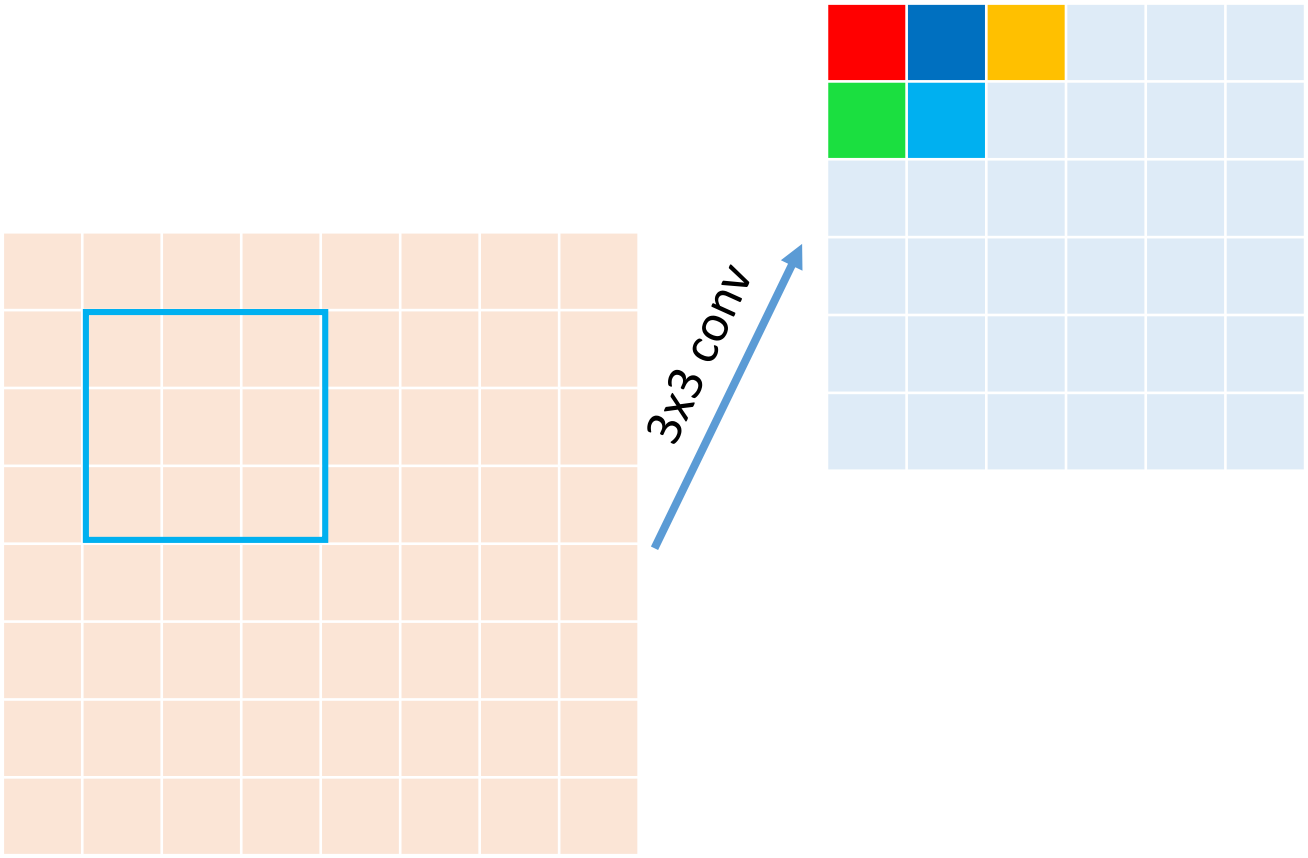
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



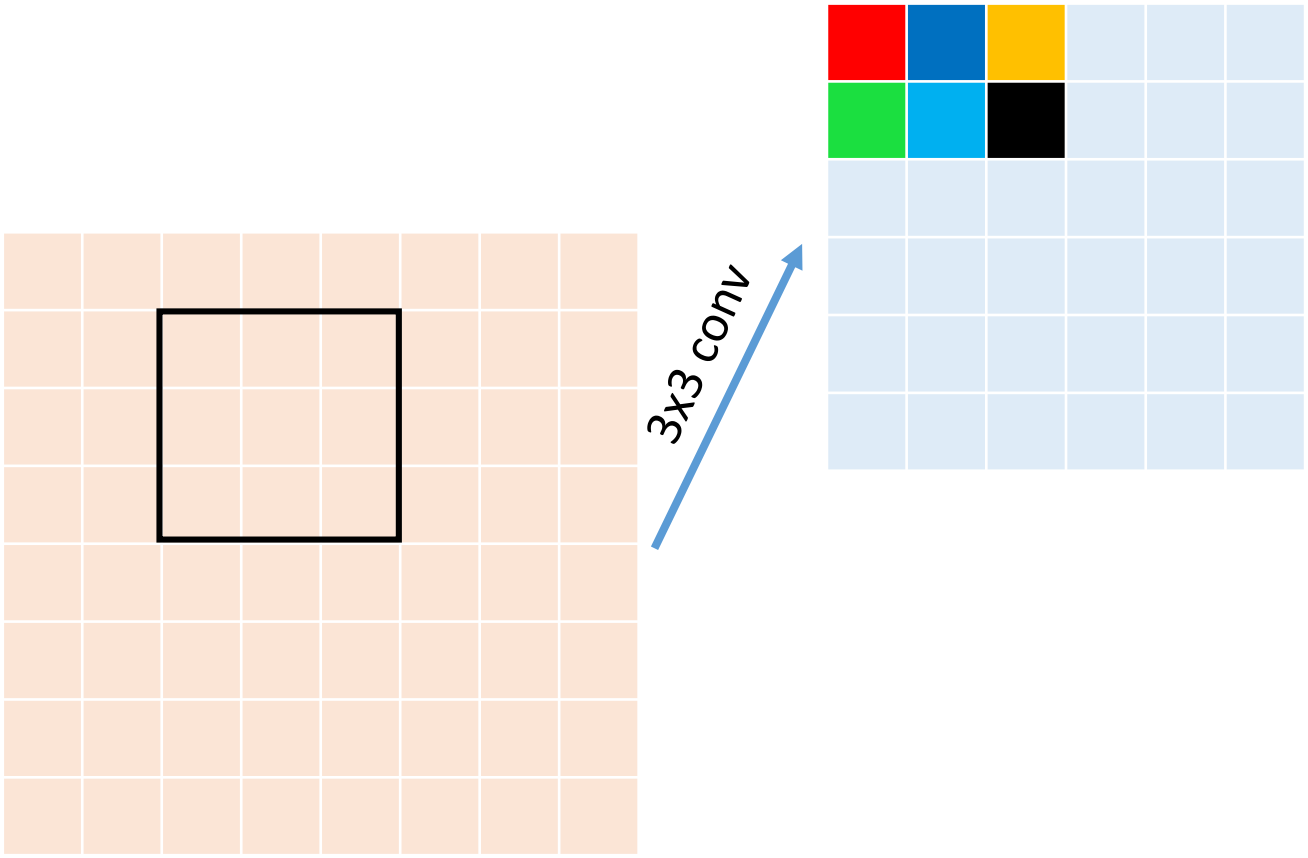
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



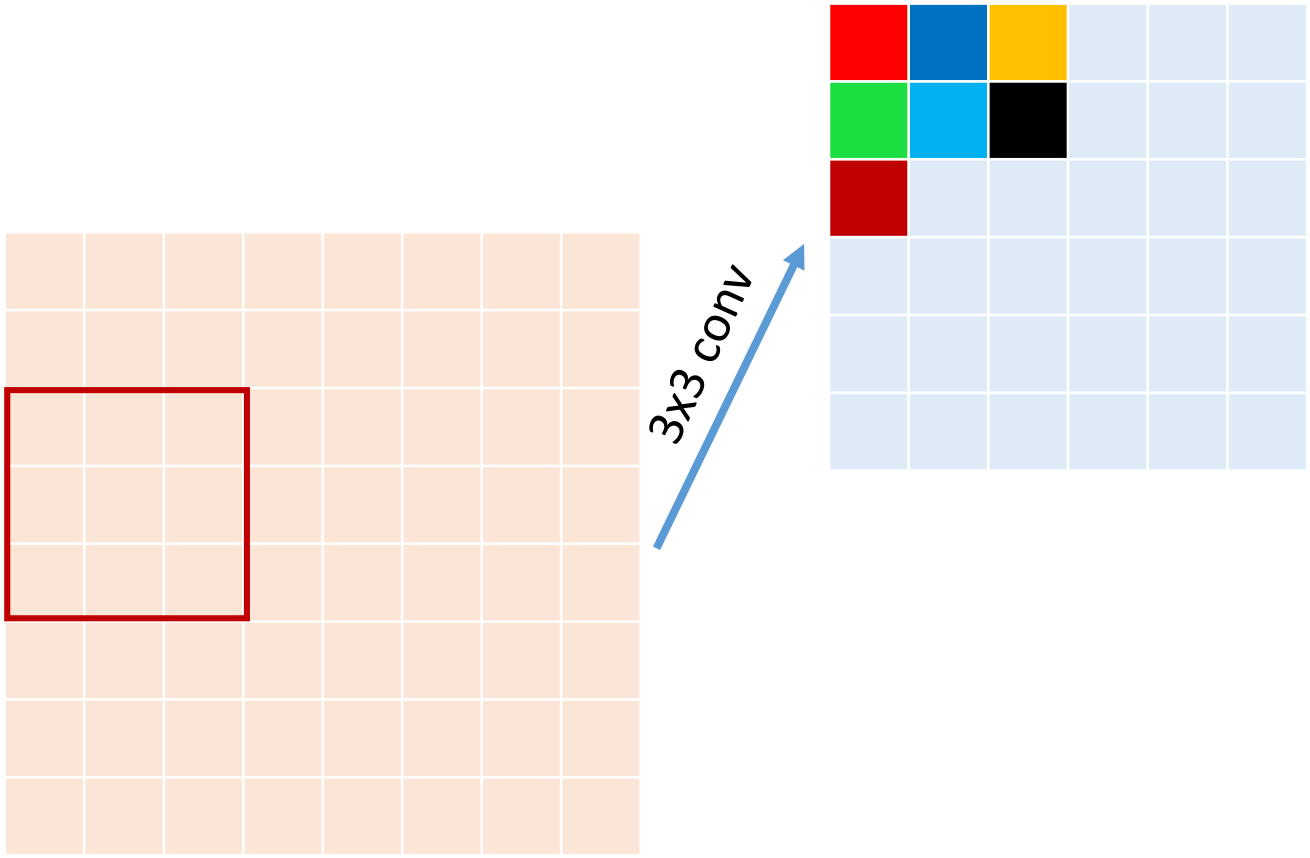
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



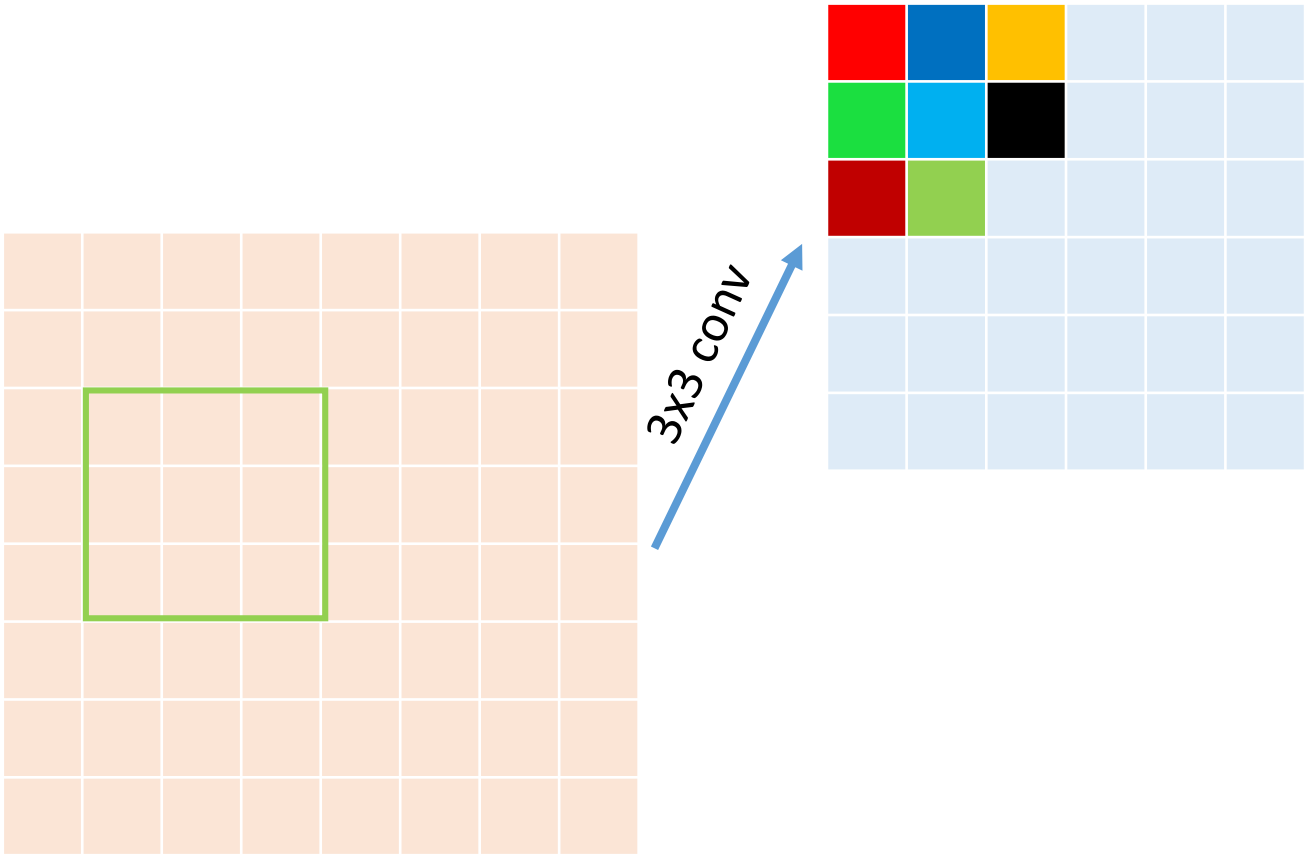
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



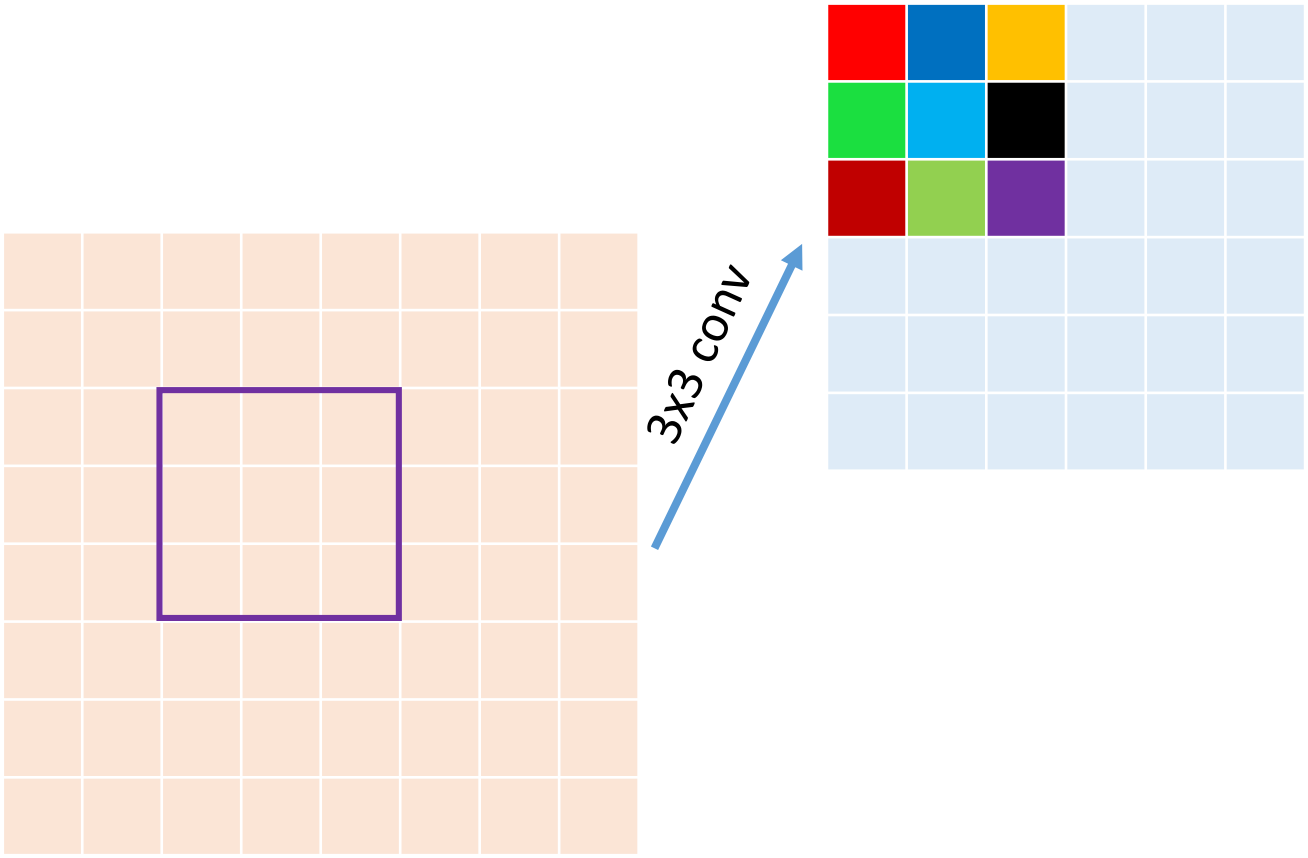
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



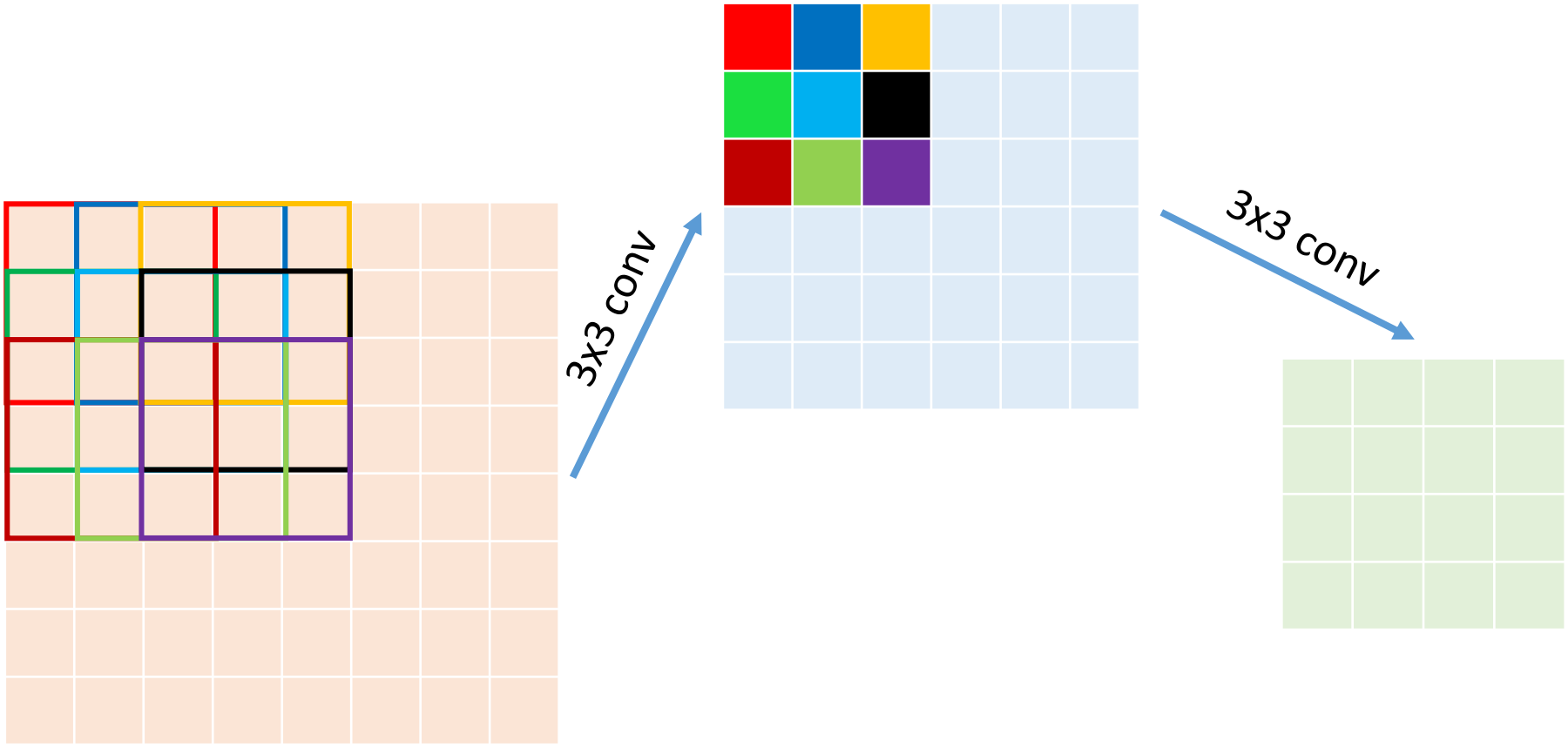
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



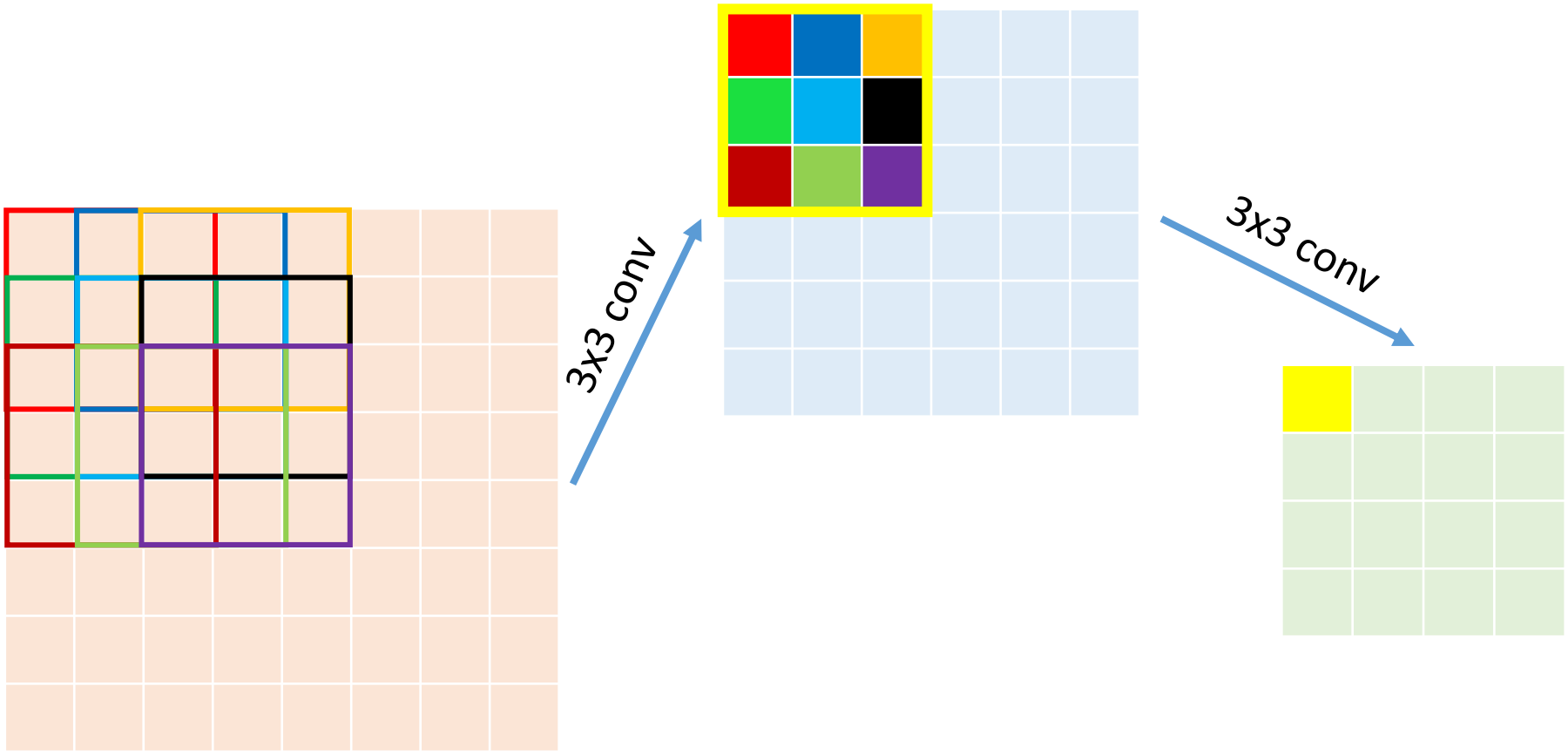
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



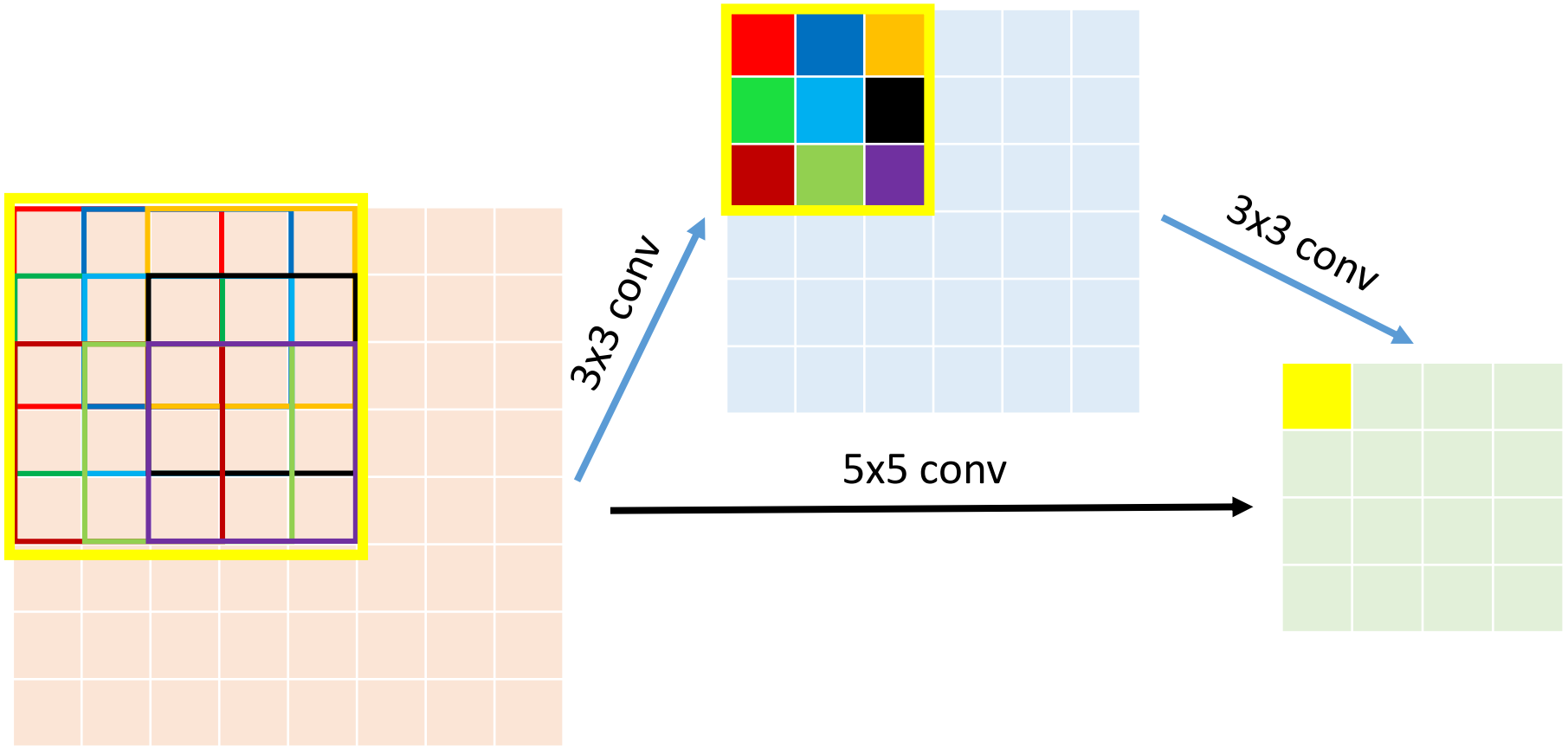
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



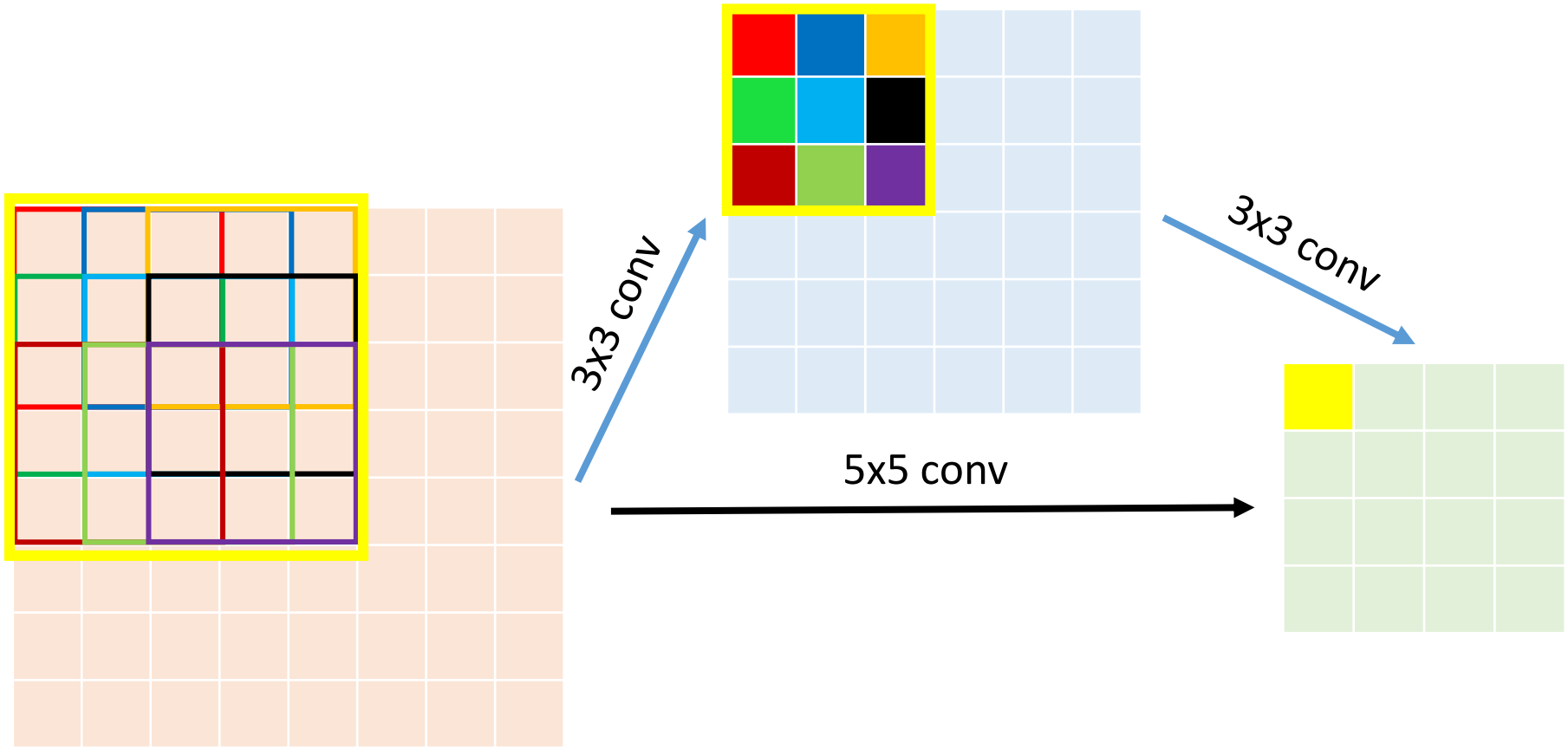
- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



Q: 三层的3x3 conv 的感受野 相当于 一层的FilterSize=?的conv的感受野?

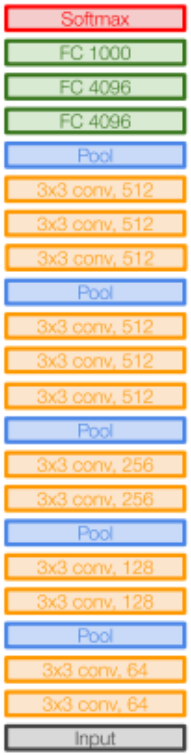
A: 三层的3x3 conv 的感受野 相当于 一层的7x7的conv的感受野

- 一层5x5 conv 的感受野 VS. 两层3x3 conv 的感受野



(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000



VGG16

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

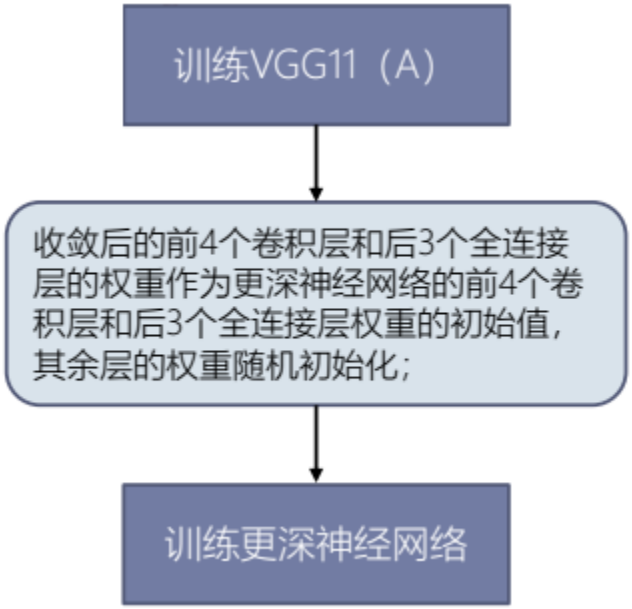
Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

- 由简单到复杂的网络架构

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					

如何训练更深网络结构?



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					

- 对比A和A-LRN：增加LRN，准确率无明显提升；
- 对比A, B, D, E：层数越多准确率越高；
- 对比C和D: conv3×3比conv1×1的准确率高；
- Multi-scale training: 可明显提高准确率.

Table 3: **ConvNet performance at a single test scale.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

VGG成功的原因

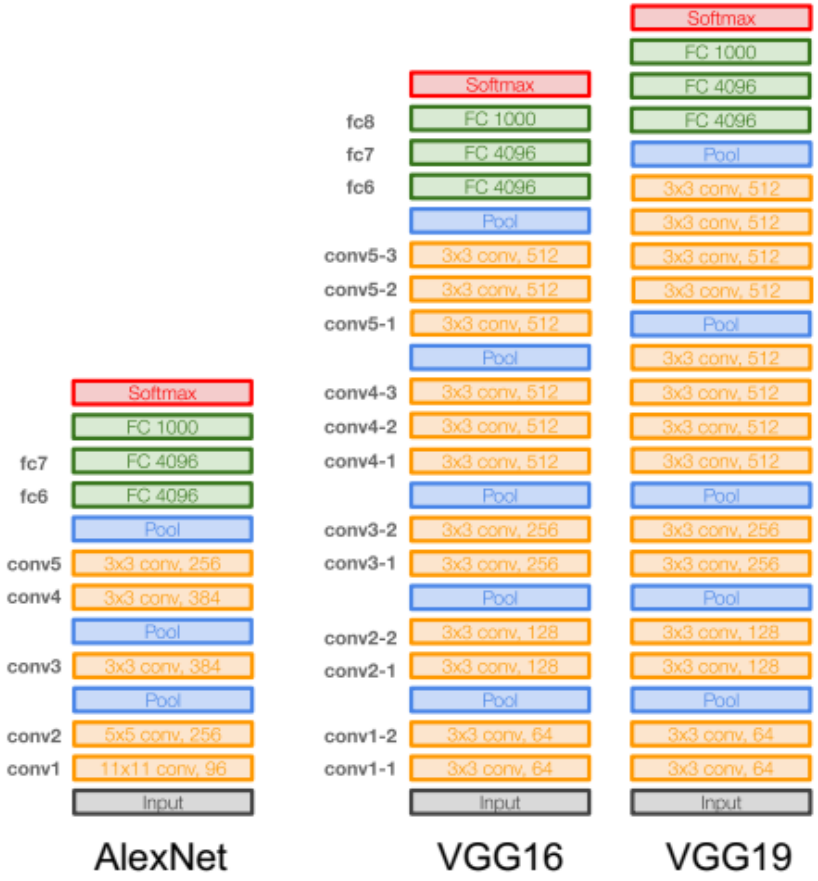
- 更深的卷积神经网络，更多的卷积层和非线性激活函数，提升分类准确率
- 使用规律的多层小卷积替代大卷积，减少参数数量，提高训练收敛速度
- 部分网络层参数的预初始化，提高训练收敛速度

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

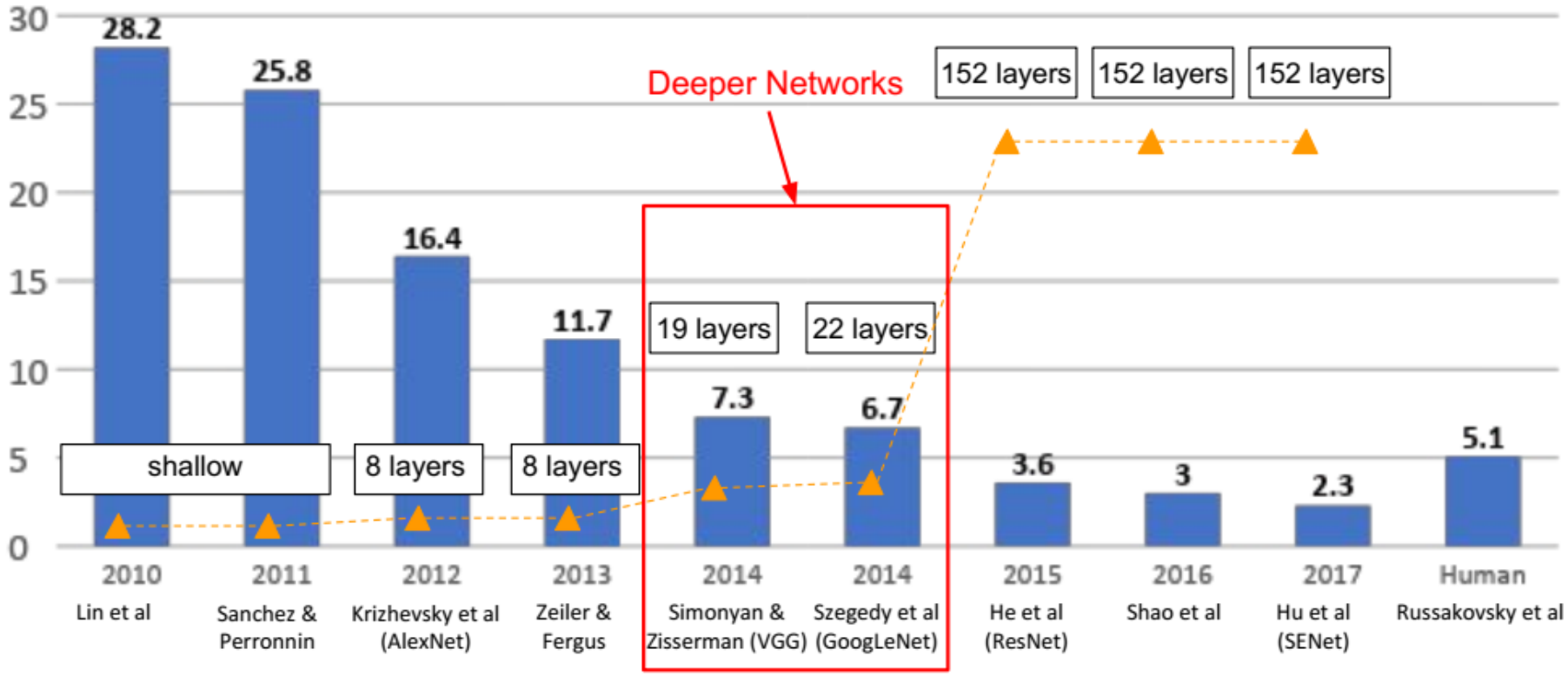
Details:

- ILSVRC'14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



卷积核能不能更小？网络能不能更深？

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

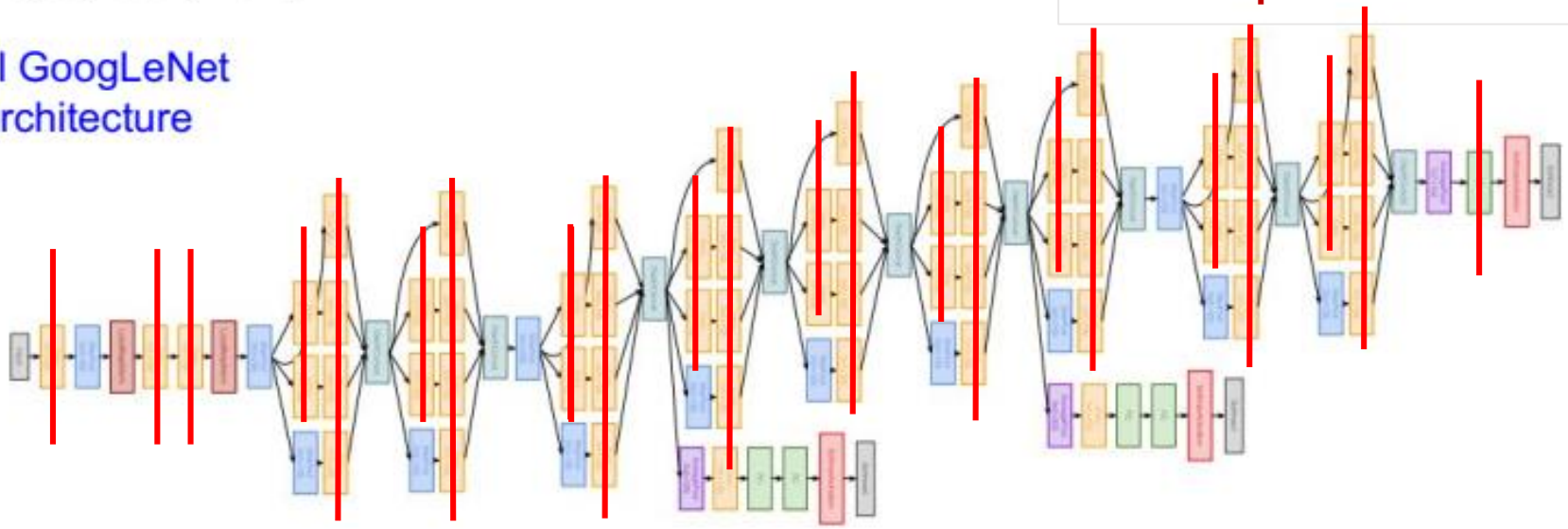


Case Study: GoogLeNet

[Szegedy et al., 2014]

Convolution
MaxPooling
Filter concatenation
Softmax
AveragePooling
FC
LocalResponseNormalization

Full GoogLeNet architecture



22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don't count auxiliary output layers)

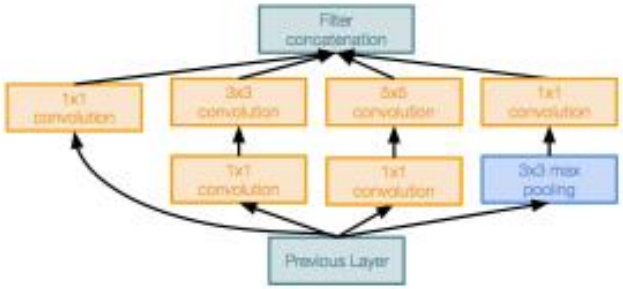
Case Study: GoogLeNet

[Szegedy et al., 2014]

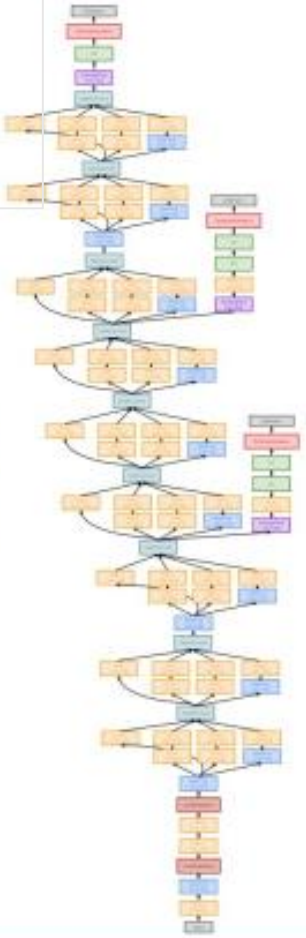
Deeper networks, with computational efficiency

- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC'14 classification winner
  (6.7% top 5 error)

Convolution
MaxPooling
Filter concatenation
Softmax
AveragePooling
FC
LocalResponseNormalization



Inception module

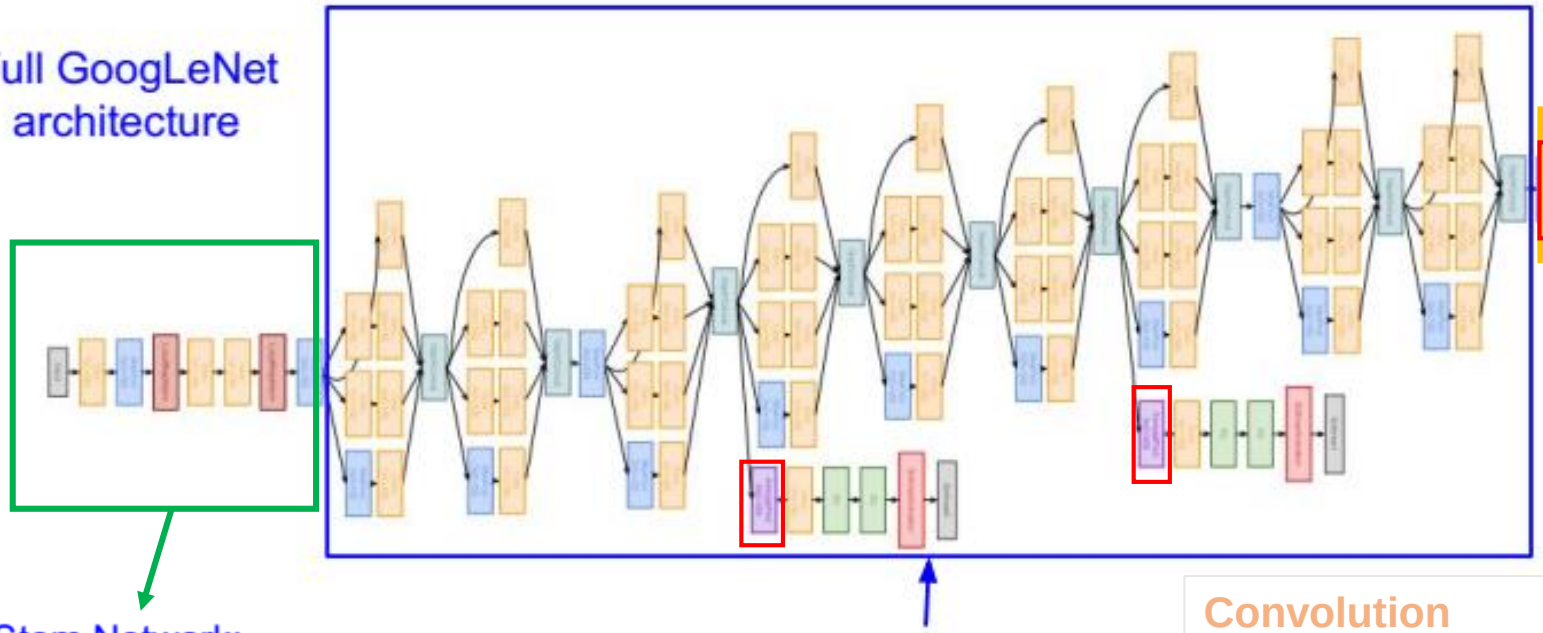


Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture



Classifier output

Stem Network: Conv-Pool- 2x Conv-Pool

Stacked Inception Modules

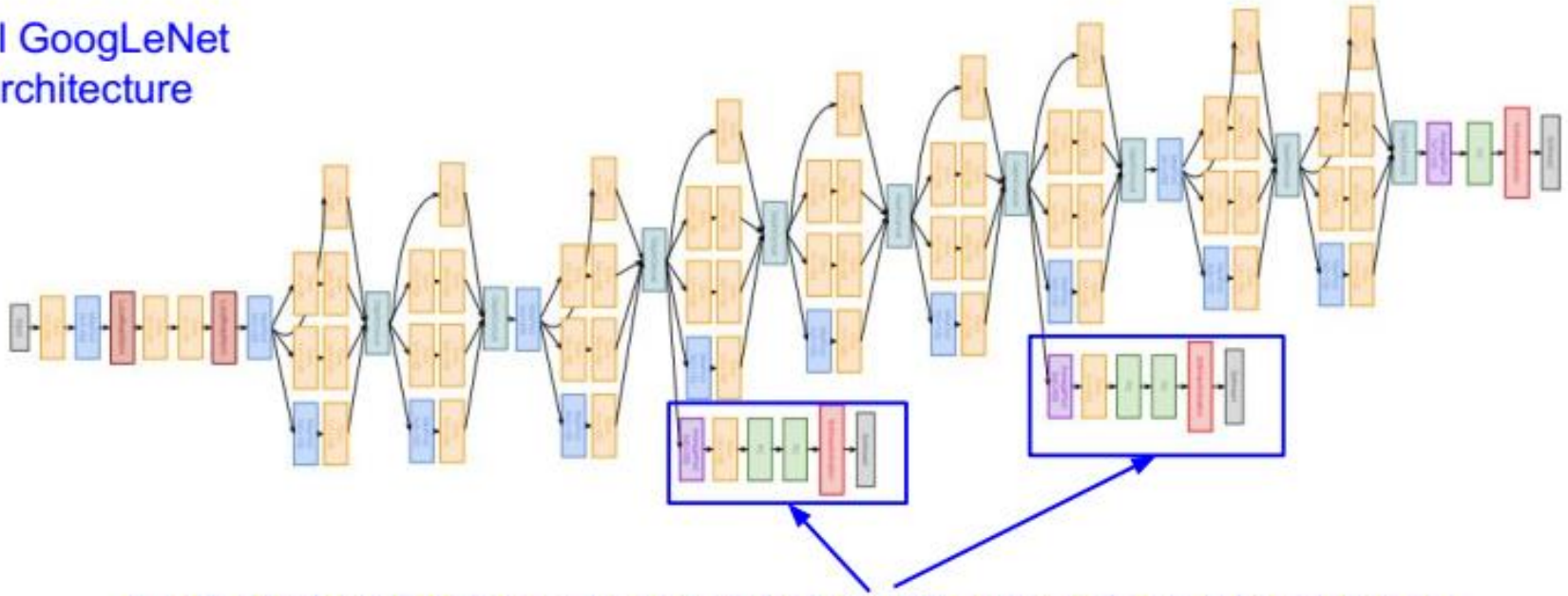
Convolution
MaxPooling
Filter concatenation
Softmax
AveragePooling
FC
LocalResponseNormalization

Case Study: GoogLeNet

[Szegedy et al., 2014]

Convolution
MaxPooling
Filter concatenation
Softmax
AveragePooling
FC
LocalResponseNormalization

Full GoogLeNet architecture



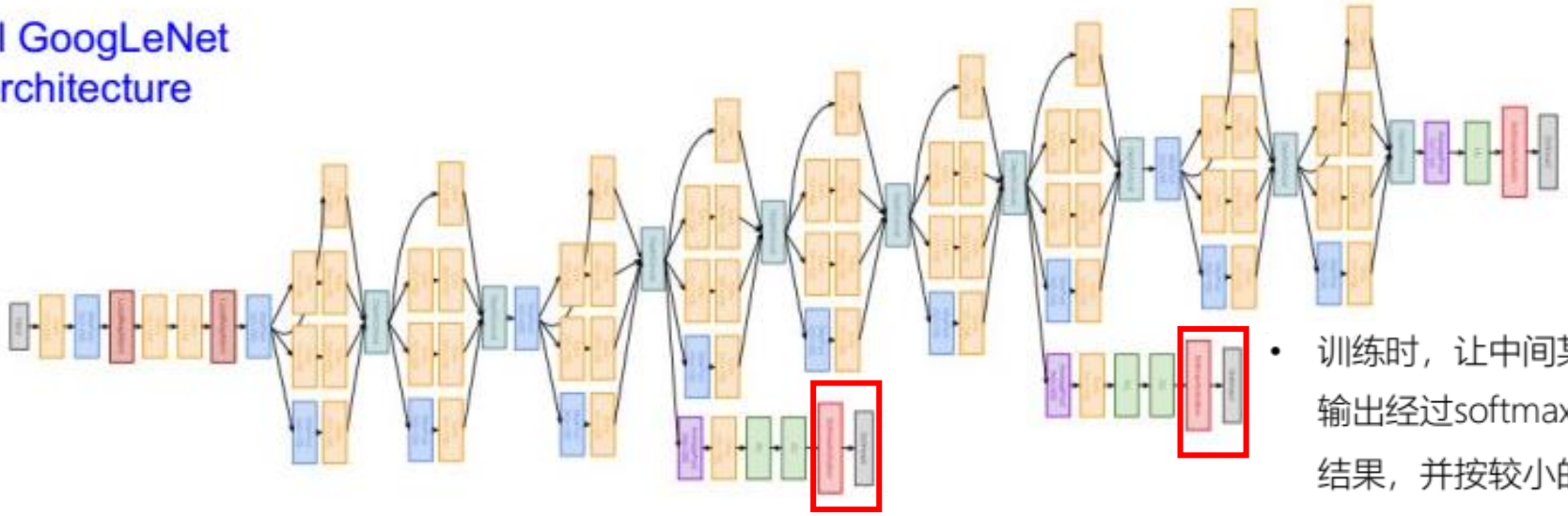
Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)

Case Study: GoogLeNet

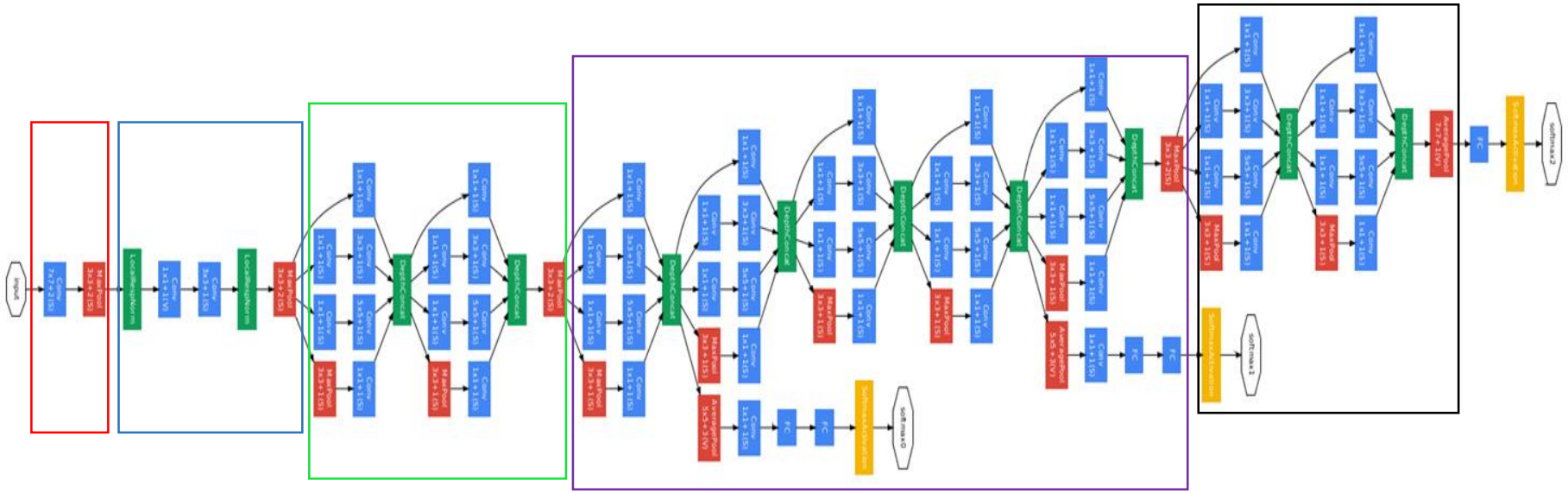
[Szegedy et al., 2014]

Convolution
MaxPooling
Filter concatenation
Softmax
AveragePooling
FC
LocalResponseNormalization

Full GoogLeNet architecture



- 训练时，让中间某一层的输出经过softmax得到分类结果，并按较小的权重加到最终分类结果中，相当于模型融合。防止多层神经网络训练过程中梯度消失。
- 推理时，softmax辅助分类网络会被去掉。



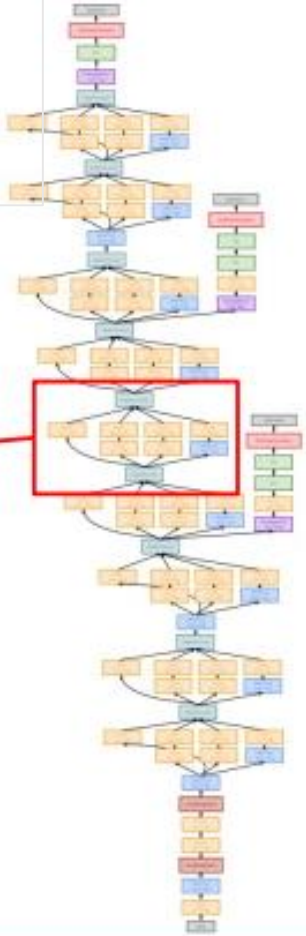
type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	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								

Table 1: GoogLeNet incarnation of the Inception architecture

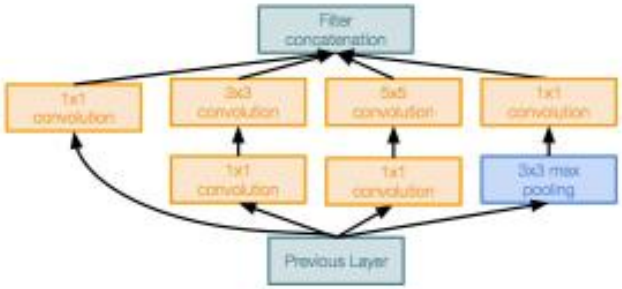
Case Study: GoogLeNet

[Szegedy et al., 2014]

Convolution
MaxPooling
Filter concatenation
Softmax
AveragePooling
FC
LocalResponseNormalization



"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other



Inception module

训练速度：

计算量

参数量

其他技巧：比如BN

内存空间开销：中间输出结果的数量

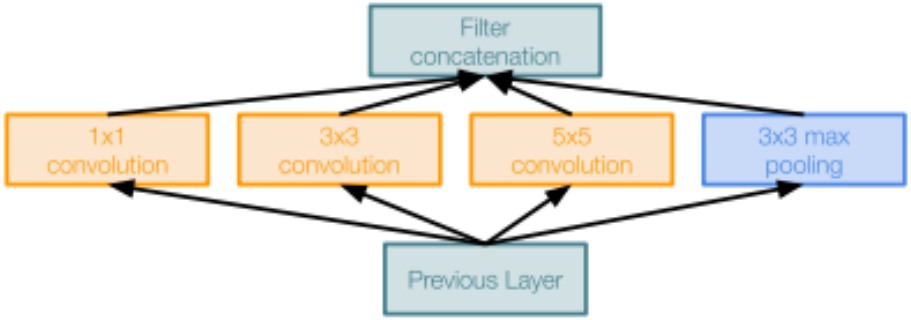
Inception

问题：三张狗的图片，狗所在的区域不同，大小不同，如何选择合适尺度的卷积核？

对策：使用Inception网络。 Inception网络不需要人为决定使用哪个过滤器或者是否需要池化，而是由网络自行确定这些参数，你可以给网络添加这些参数的所有可能值，然后把这些输出连接起来，让网络自己学习它需要什么样的参数，采用哪些过滤器组合。

Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

Apply parallel filter operations on the input from previous layer:

- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

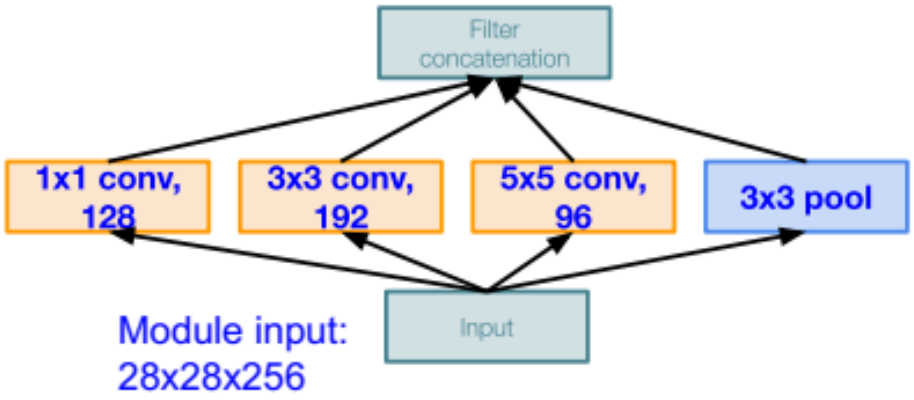
Concatenate all filter outputs together depth-wise

Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:



Naive Inception module

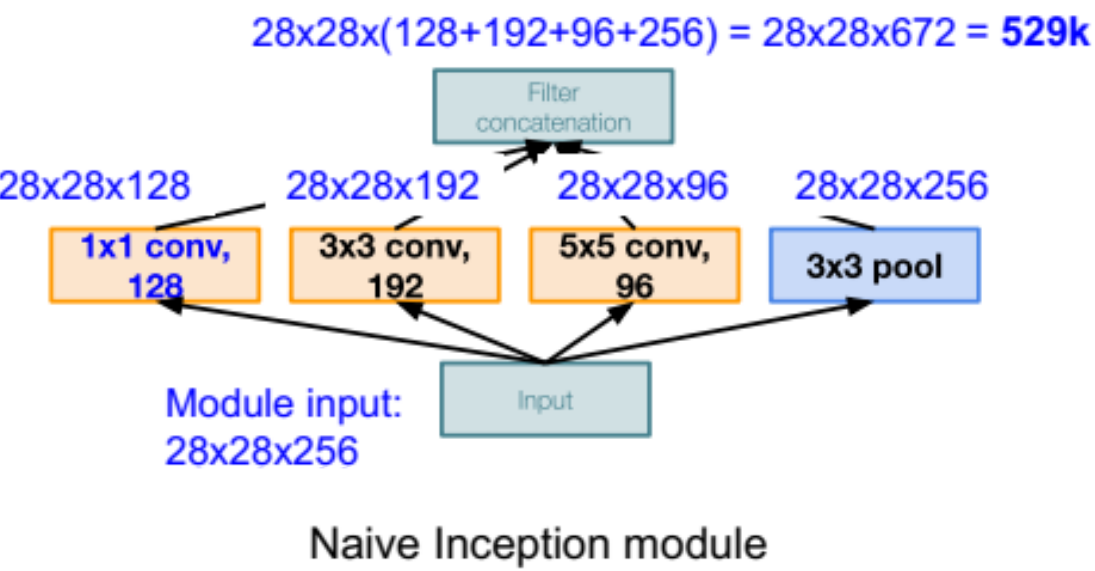
Q1: What is the output size of the1x1 conv, with 128 filters?
Q2: What are the output sizes of all different filter operations?
Q3:What is output size after filter concatenation?

Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:



Naive Inception module

Q1: What is the output size of the1x1 conv, with 128 filters?
Q2: What are the output sizes of all different filter operations?
Q3:What is output size after filter concatenation?

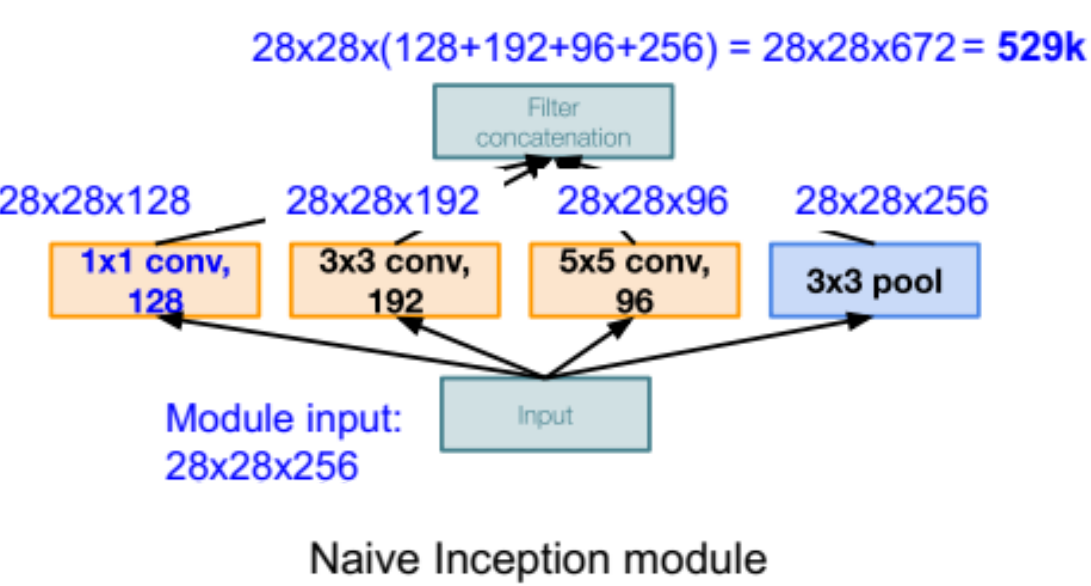
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Case Study: GoogLeNet

[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:



Naive Inception module

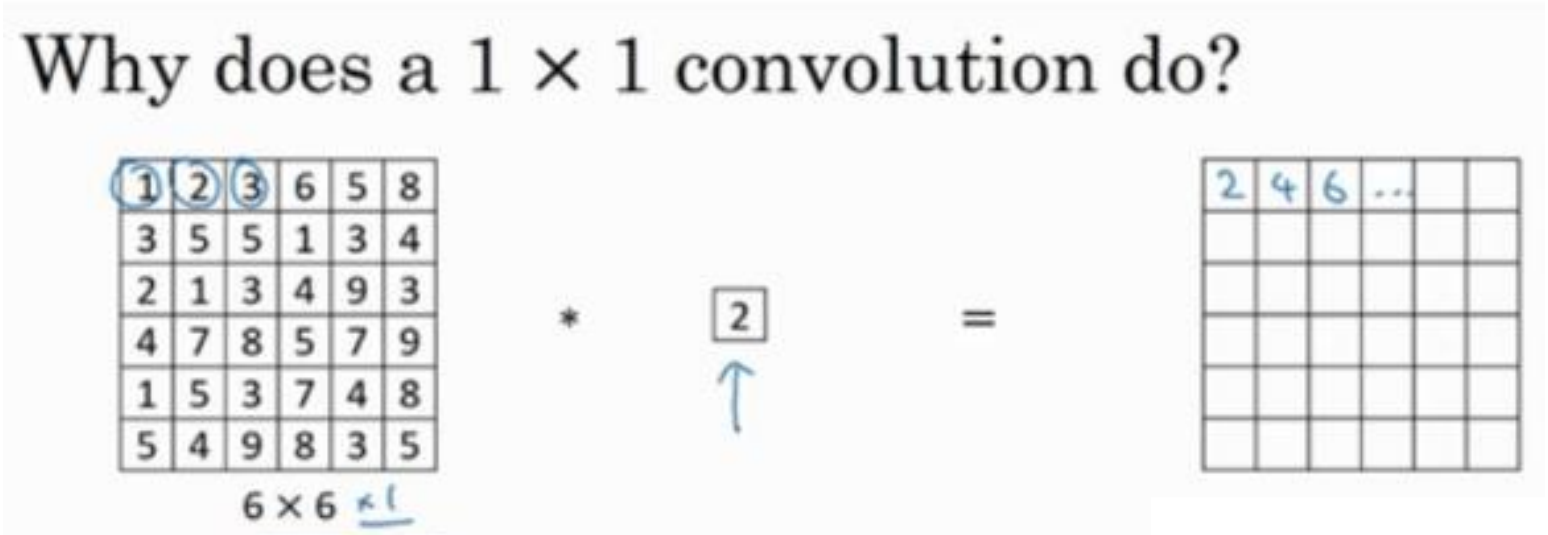
Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

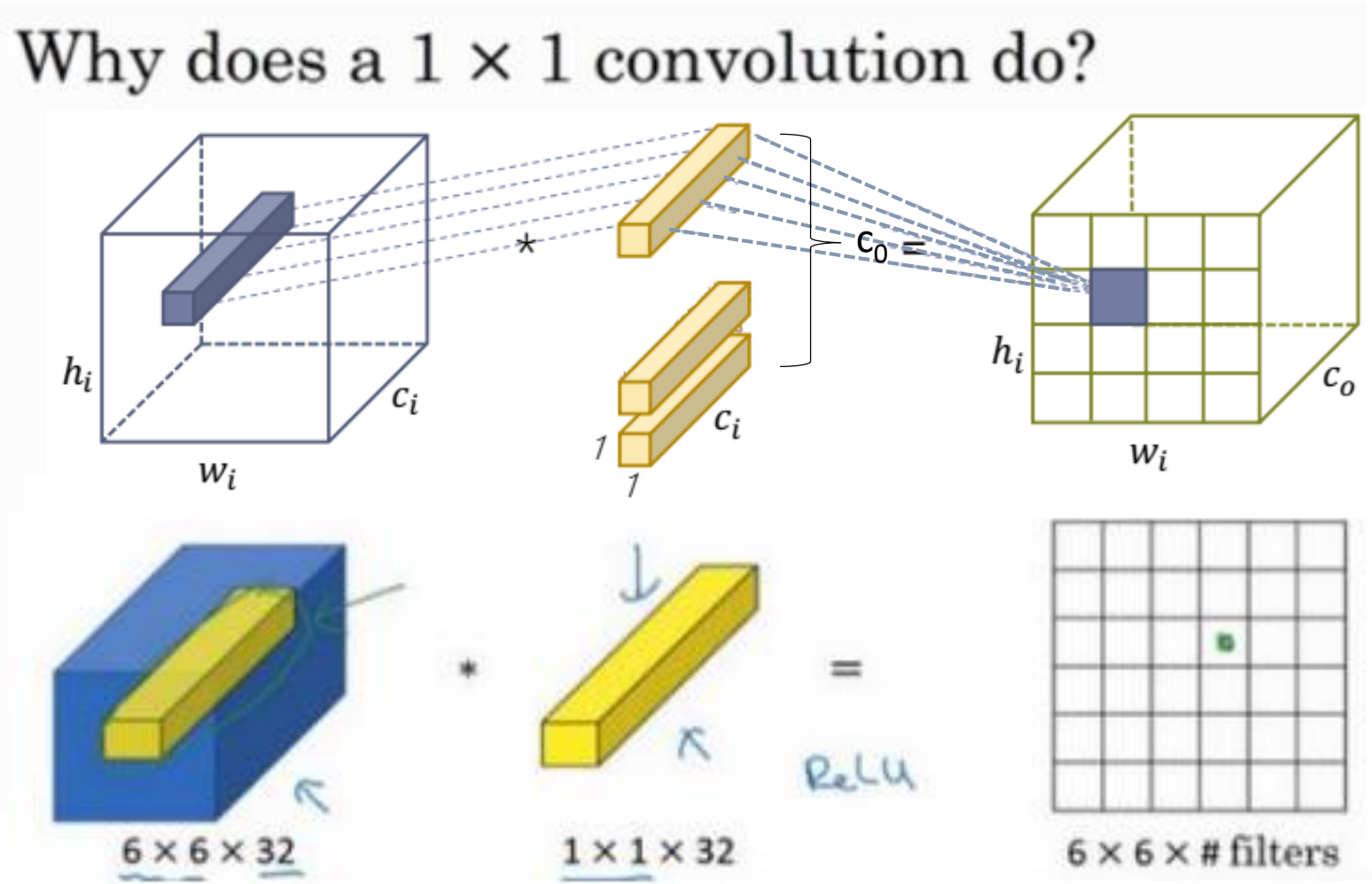
Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

对策：利用1×1卷积核！

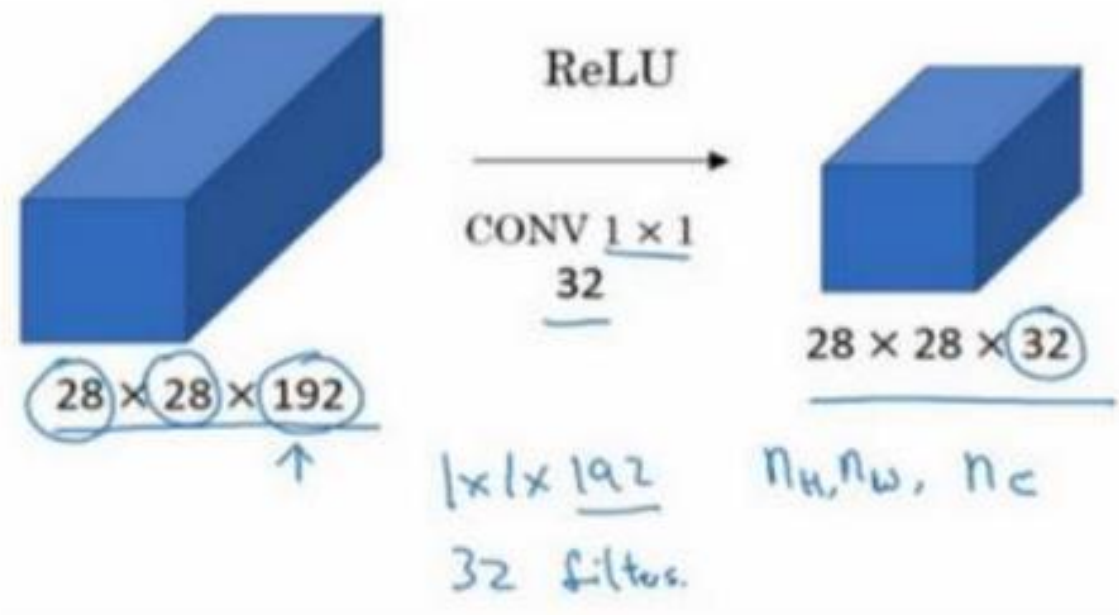
1×1卷积


Why does a 1 × 1 convolution do?
1 2 3 6 5 8
3 5 5 1 3 4
2 1 3 4 9 3
4 7 8 5 7 9
1 5 3 7 4 8
5 4 9 8 3 5
6 × 6 × 1
*
2
=
2 4 6 ...

1×1卷积


Why does a 1 × 1 convolution do?
h_i
w_i
c_i
*
1
1
c_i
c_0 =
h_i
w_i
c_o
6 × 6 × 32
*
1 × 1 × 32
=
ReLU
6 × 6 × # filters

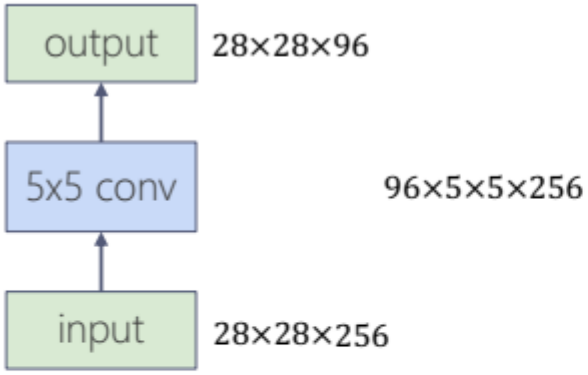
1×1卷积



1×1卷积层的作用：保持输入的宽和高不变，可自主控制输出的通道数。

- 给神经网络添加了一个非线性函数，从而减少或保持输入层中的通道数量不变；也可以增加通道数量。

- 使用1×1卷积，形成“瓶颈层”，可有效减少计算量和参数量

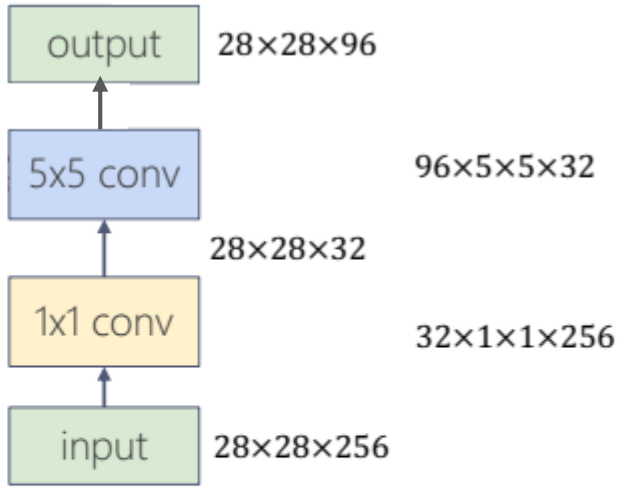


乘法次数：

28×28×96×5×5×256≈4.8×10^8

参数数量：

96×5×5×256 ≈6.1×10^5



乘法次数：

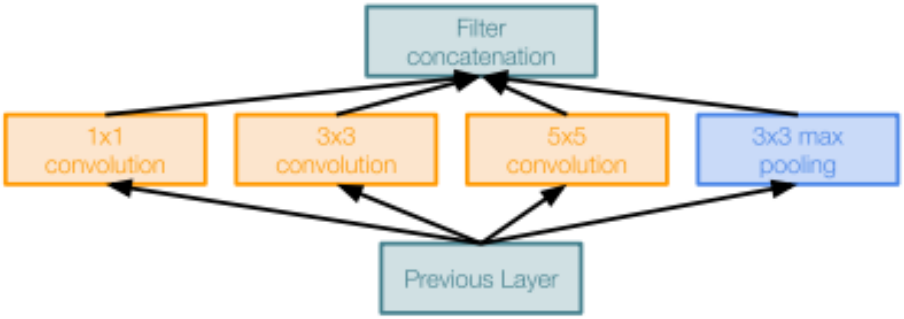
28×28×32×256+28×28×96×5×5×32≈6.7×10^7

参数数量：

32×256+96×5×5×32 ≈8.5×10^4

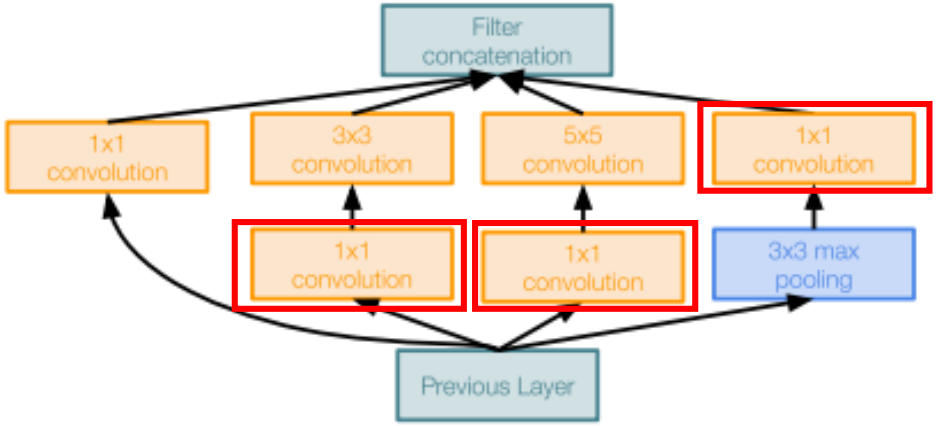
Case Study: GoogLeNet

[Szegedy et al., 2014]



Naive Inception module

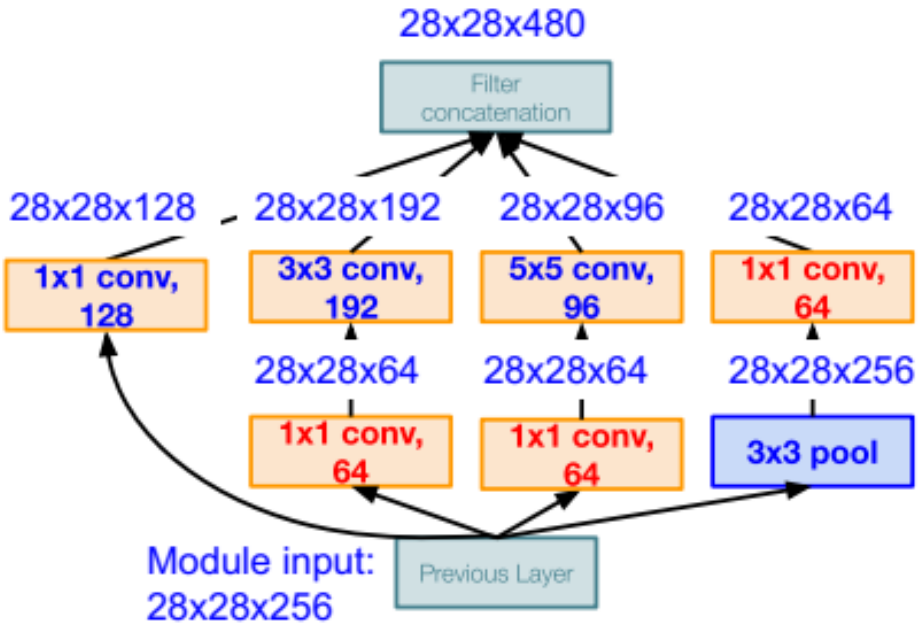
1x1 conv "bottleneck" layers



Inception module with dimension reduction

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module with dimension reduction

Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

GoogLeNet in Keras

- 图7-8给出了这种模块的一个示例，它来自于Inception V3

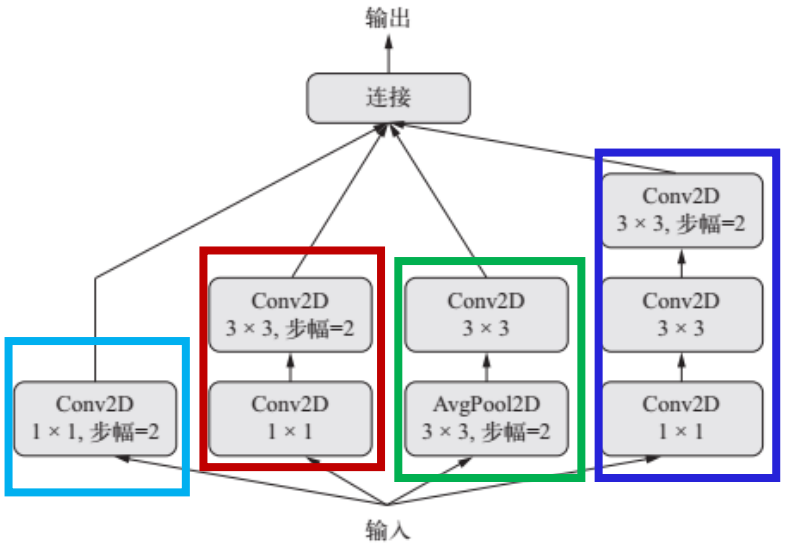


图 7-8　Inception 模块

每个分支都有相同的步幅值（2），这对于保持所有分支输出具有相同的尺寸是很有必要的，这样你才能将它们连接在一起

```
from keras import layers

branch_a = layers.Conv2D(128, 1,
                         activation='relu', strides=2)(x)
branch_b = layers.Conv2D(128, 1, activation='relu')(x)
branch_b = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_b)
branch_c = layers.AveragePooling2D(3, strides=2)(x)
branch_c = layers.Conv2D(128, 3, activation='relu')(branch_c)
branch_d = layers.Conv2D(128, 1, activation='relu')(x)
branch_d = layers.Conv2D(128, 3, activation='relu')(branch_d)
branch_d = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_d)

output = layers.concatenate(
    [branch_a, branch_b, branch_c, branch_d], axis=-1)
```

在这个分支中，空间卷积层用到了步幅

在这个分支中，平均池化层用到了步幅

将分支输出连接在一起，得到模块输出

总结：GoogLeNet

- 如果你在构建神经网络层的时候，不想决定池化层是使用1×1，3×3还是5×5的过滤器，那么Inception模块就是最好的选择

- 通过使用1×1卷积来构建瓶颈层，从而大大降低计算成本。

Inception家族

网络	主要创新	Top5错误率	网络层数
GoogLeNet	提出inception结构	6.67%	22
BN-Inception	提出 Batch Normalization, 用3x3代替5x5	4.82%	—
Inception-v3	将一个二维卷积拆成两个一维卷积，辅助分类器的全连接层做BN	3.5%	42
Inception-v4	inception模块化, 结合ResNet的跳转结构	3.08%	—

BN-Inception

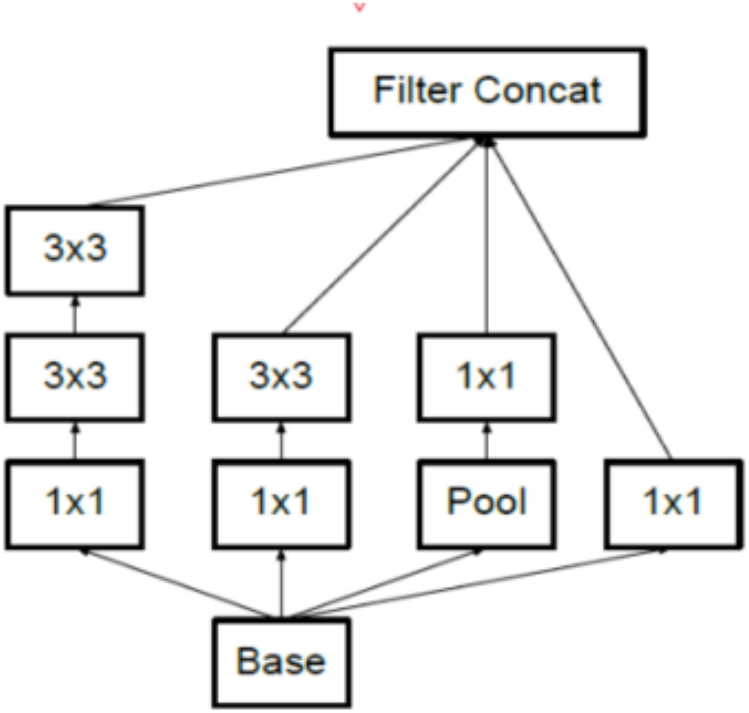
- 学习VGG，用2层3×3卷积替代一层5×5卷积
- 在每个卷积层后、激活函数之前，引入BN
  - BatchNorm 效果
    - BN 可替代 LRN / Dropout / L2 Normalization;
    - 可提高收敛速度、训练速度;
    - 可选择更高的学习率，方便调参;

Model	Steps to 72.2%	Max accuracy
Inception	$31.0 \cdot 10^6$	72.2%
BN-Baseline	$13.3 \cdot 10^6$	72.7%
BN-x5	$2.1 \cdot 10^6$	73.0%
BN-x30	$2.7 \cdot 10^6$	74.8%
BN-x5-Sigmoid		69.8%

Figure 3. For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.

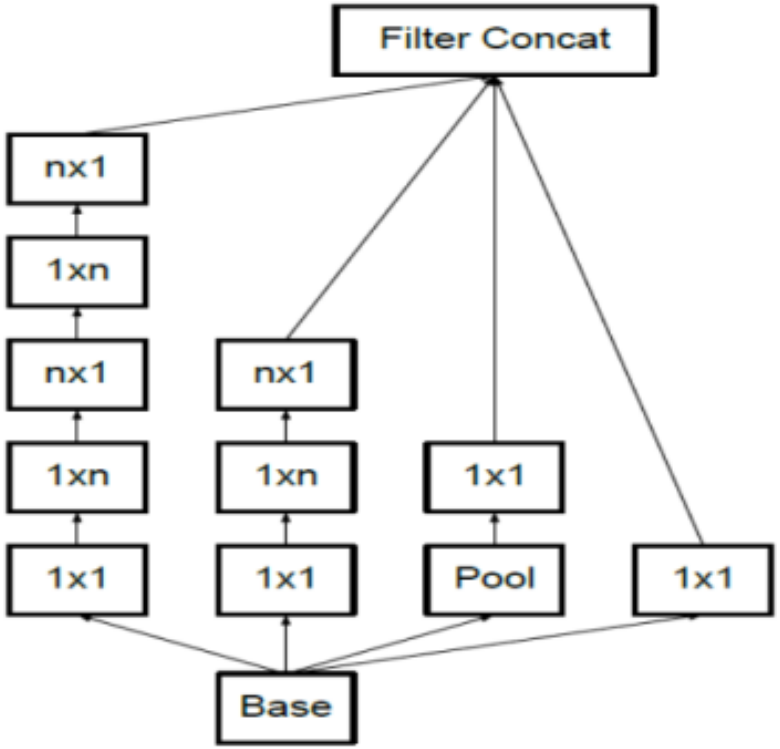
- -x5 表示学习率设为inception初始学习率的5倍。

Inception V3



用 3x3 卷积核代替 5x5 卷积核

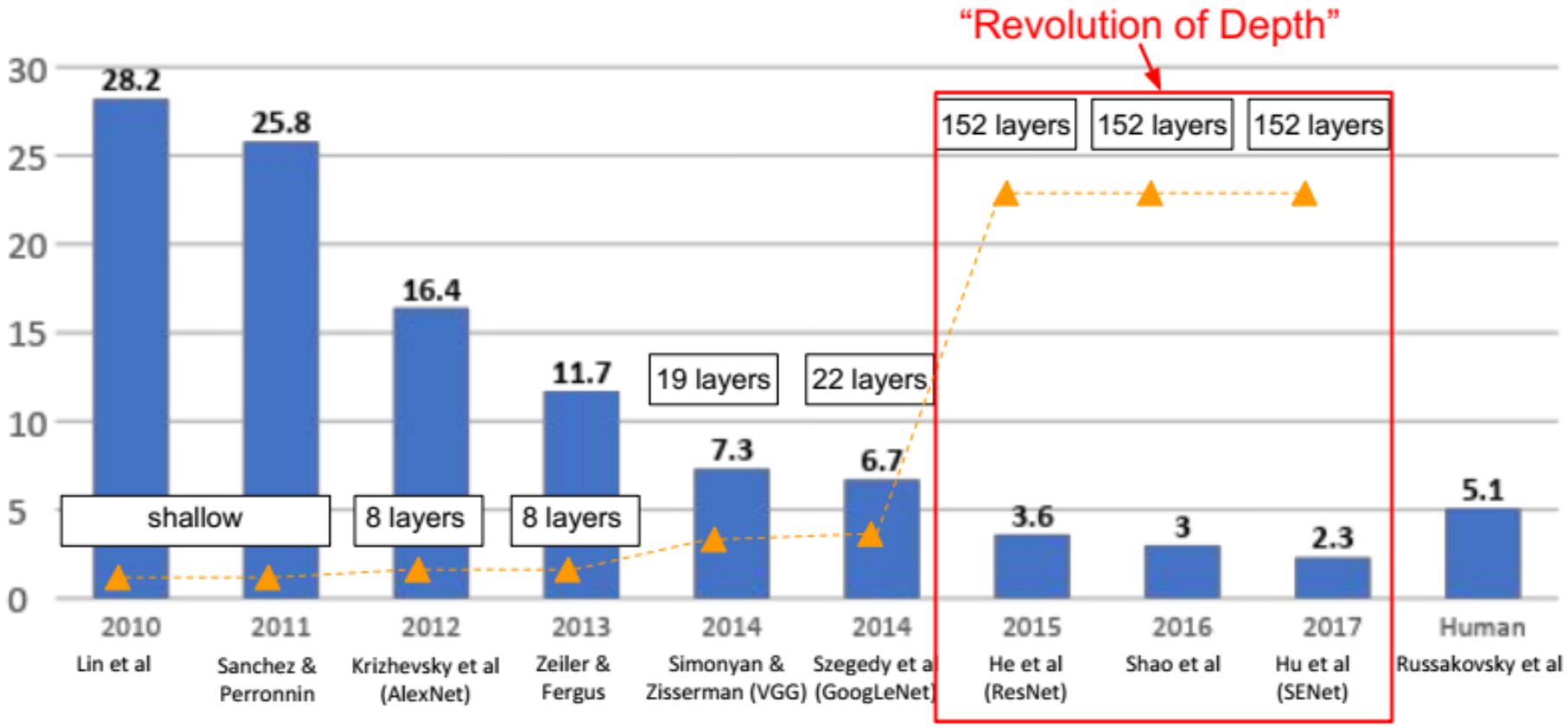
（小卷积核代替大卷积核）



用 1xn 和 nx1 卷积核代替 nxn 卷积核

Inception V2 结构

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

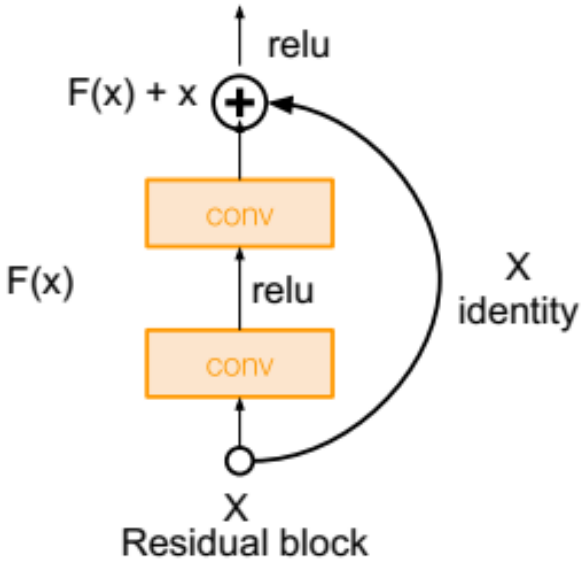


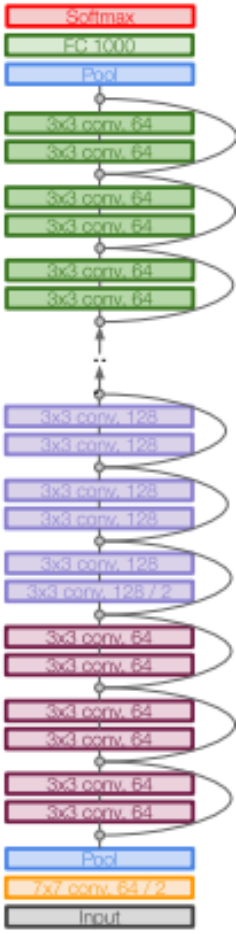
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



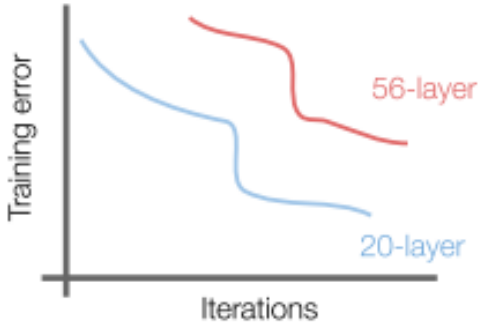


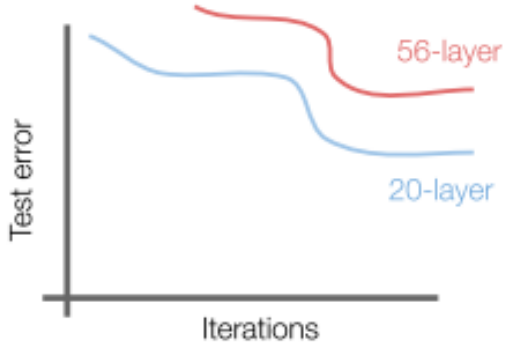
Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

神经网络退化：收敛到极值点而非最值，误差大。





56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Case Study: ResNet

[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

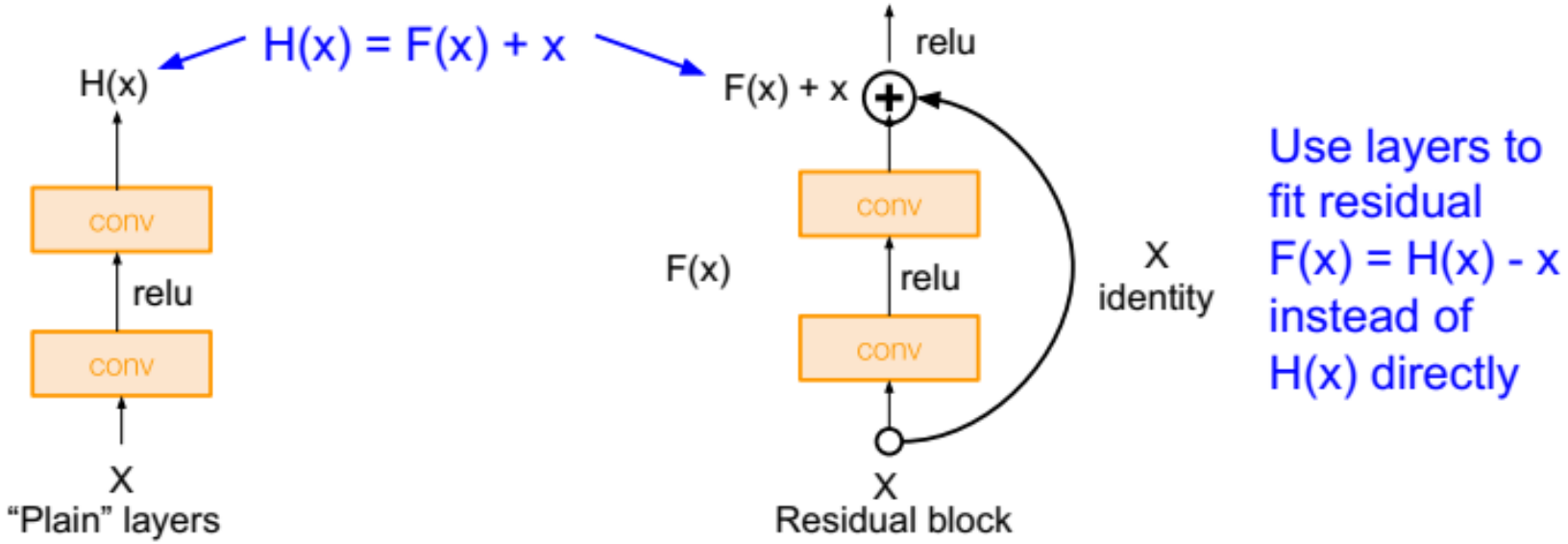
The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

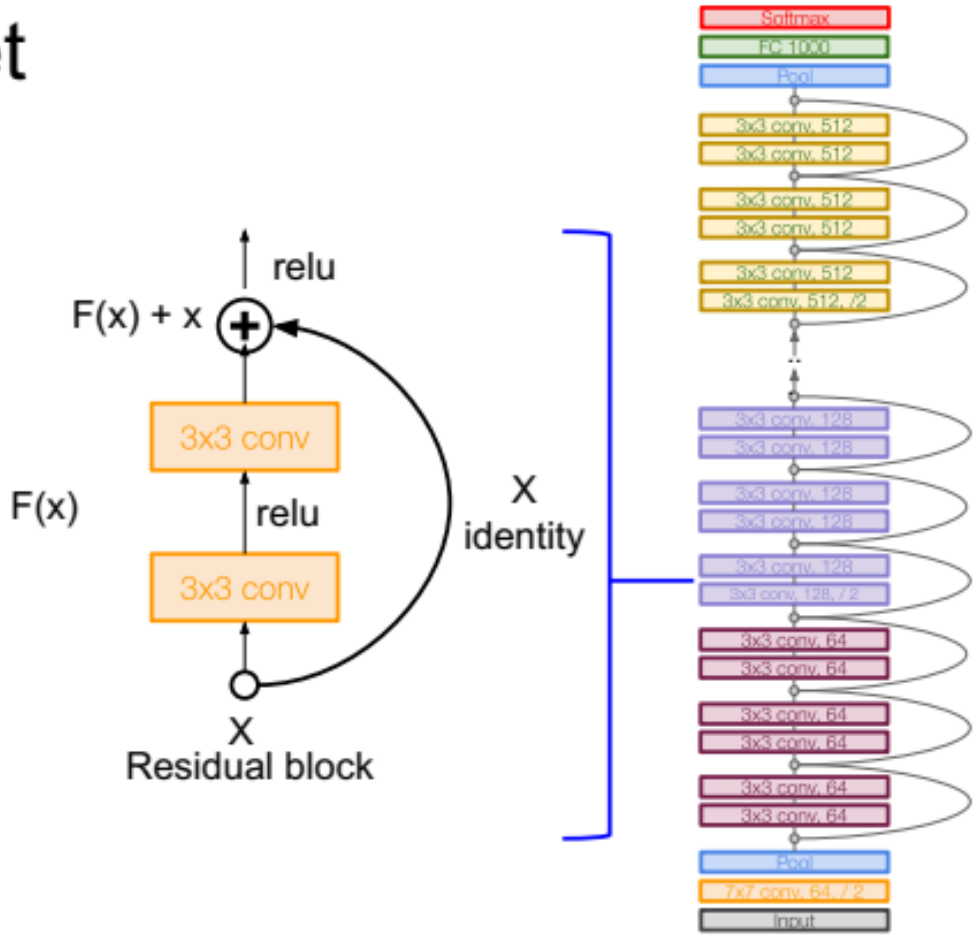


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers

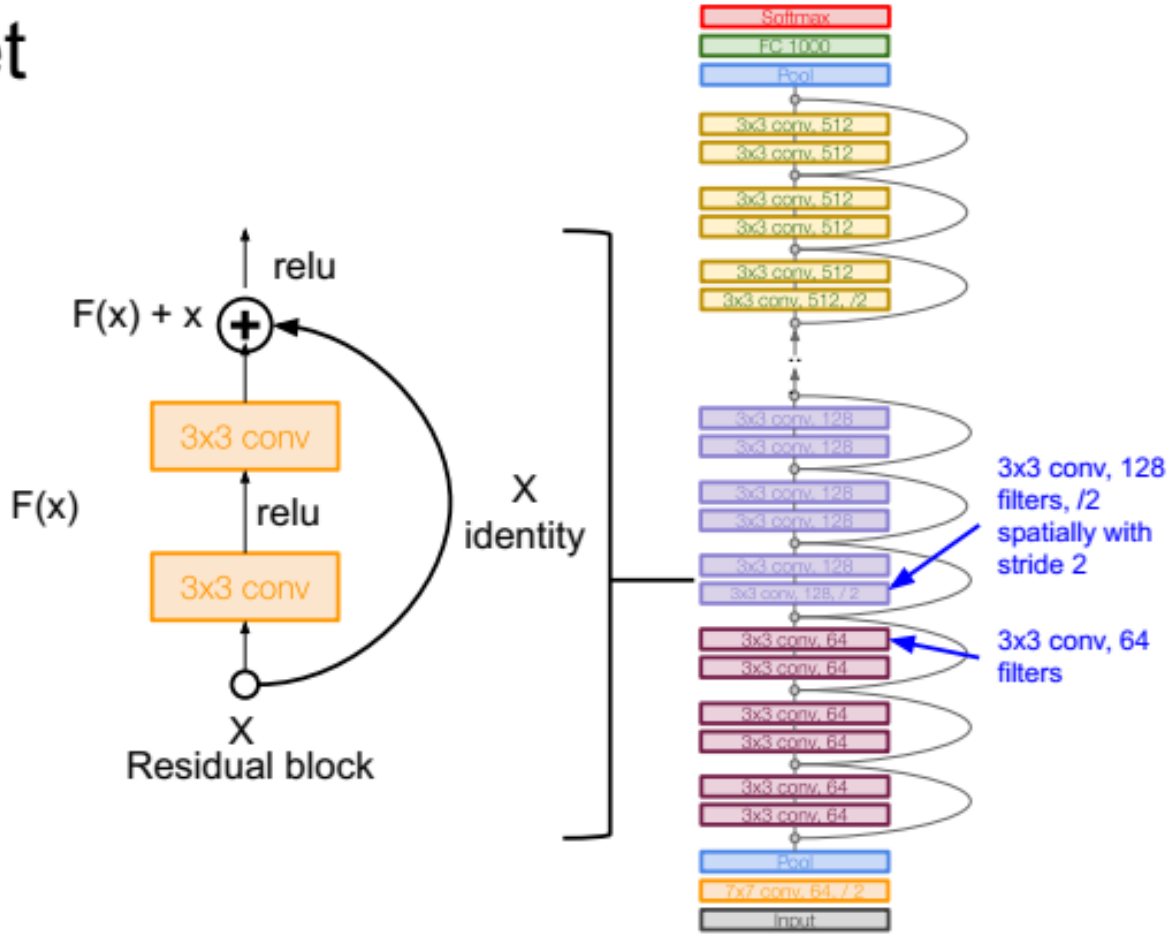


Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)

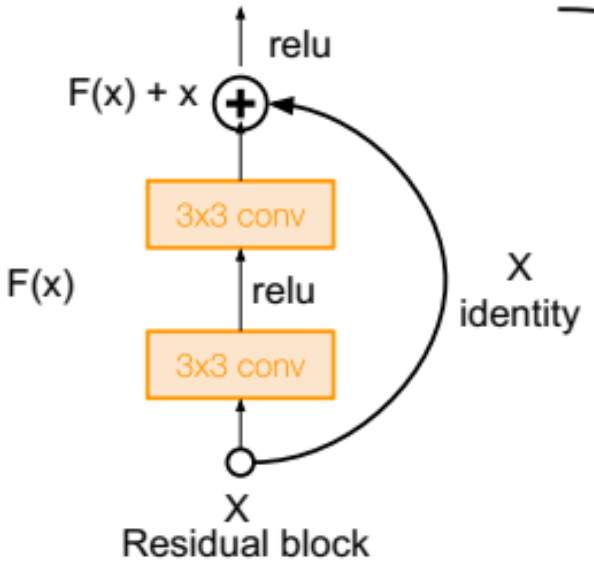


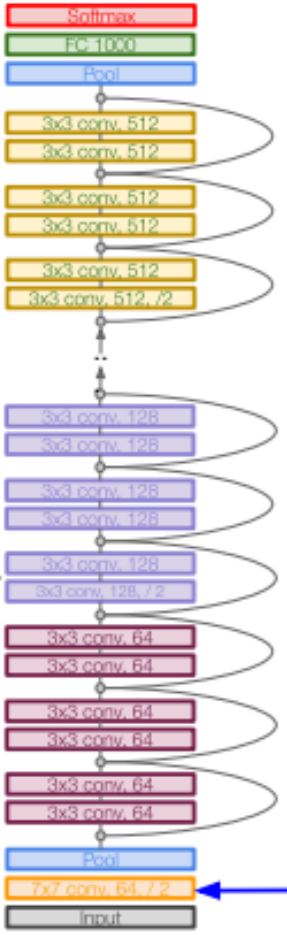
Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning



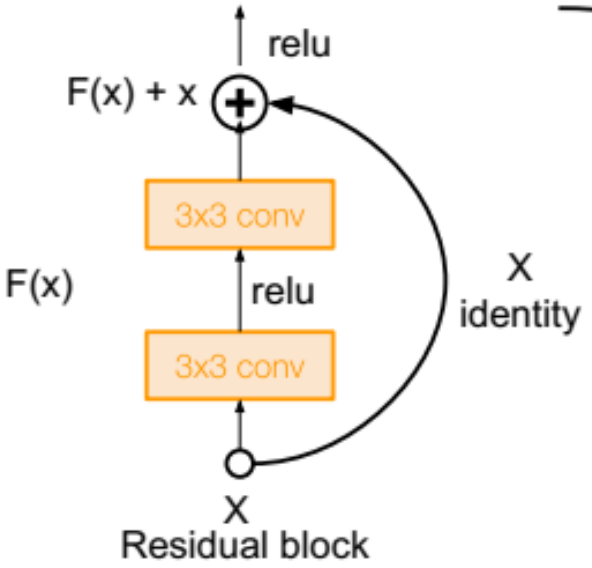


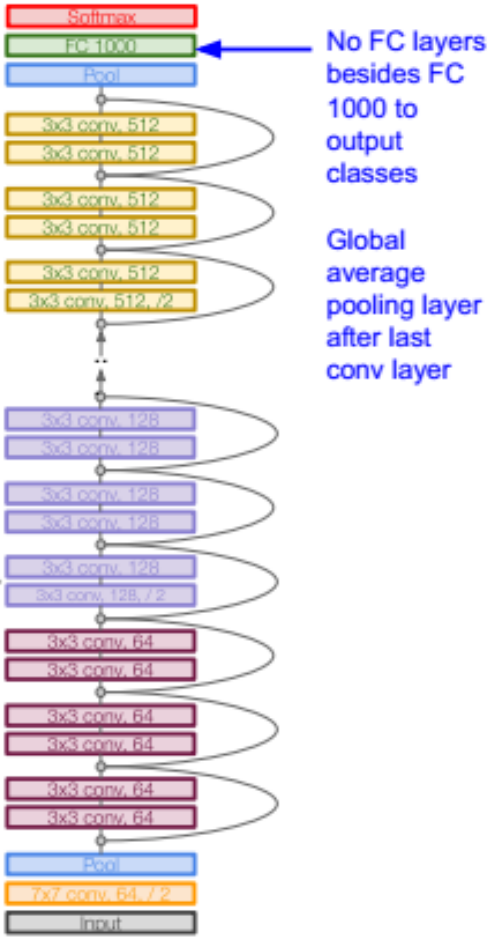
Case Study: ResNet

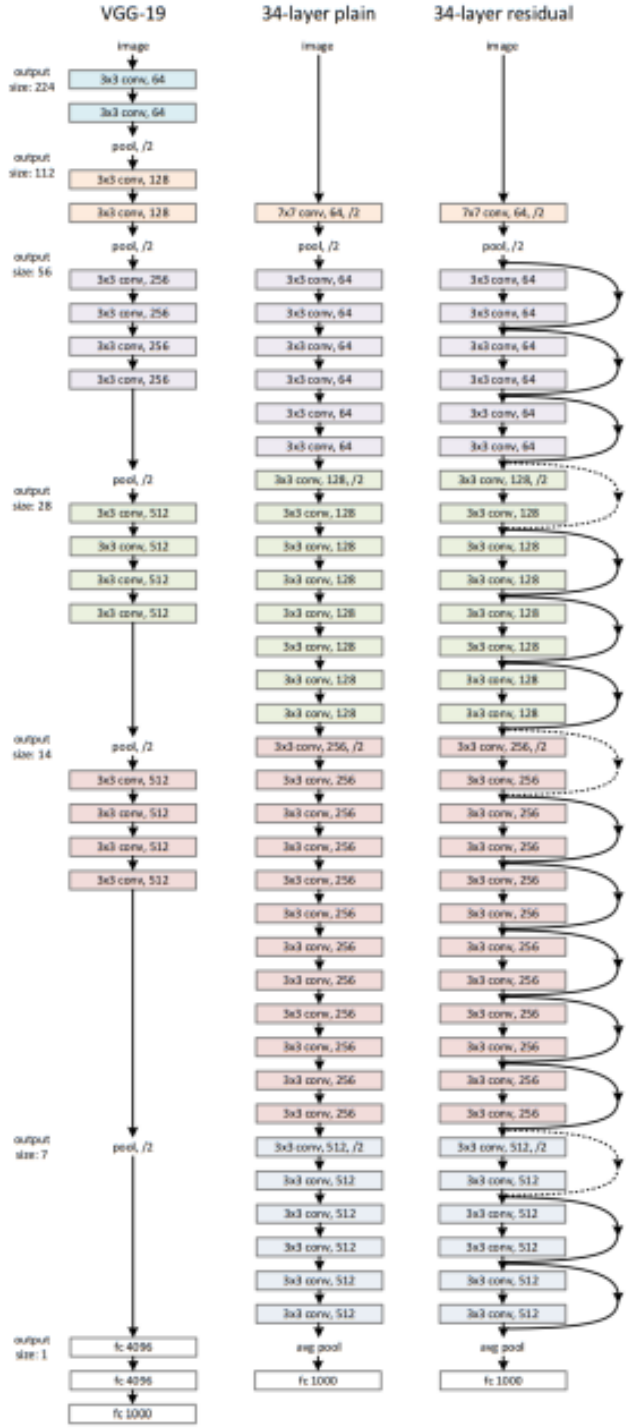
[He et al., 2015]

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)







将残差块应用到普通网络

改造VGG得到 plain-network

- plain-network：无跳转连接的普通网络；
- 基本全部由卷积层构成：filter=3*3, stride=1, pad=SAME；
- 特征图尺寸的减小由 stride=2 的卷积层完成；
- 若特征图的尺寸不变，则特征的数量也不变；若特征图的尺寸减半，则特征图的数量翻倍；

增加跳转连接得到 resnet

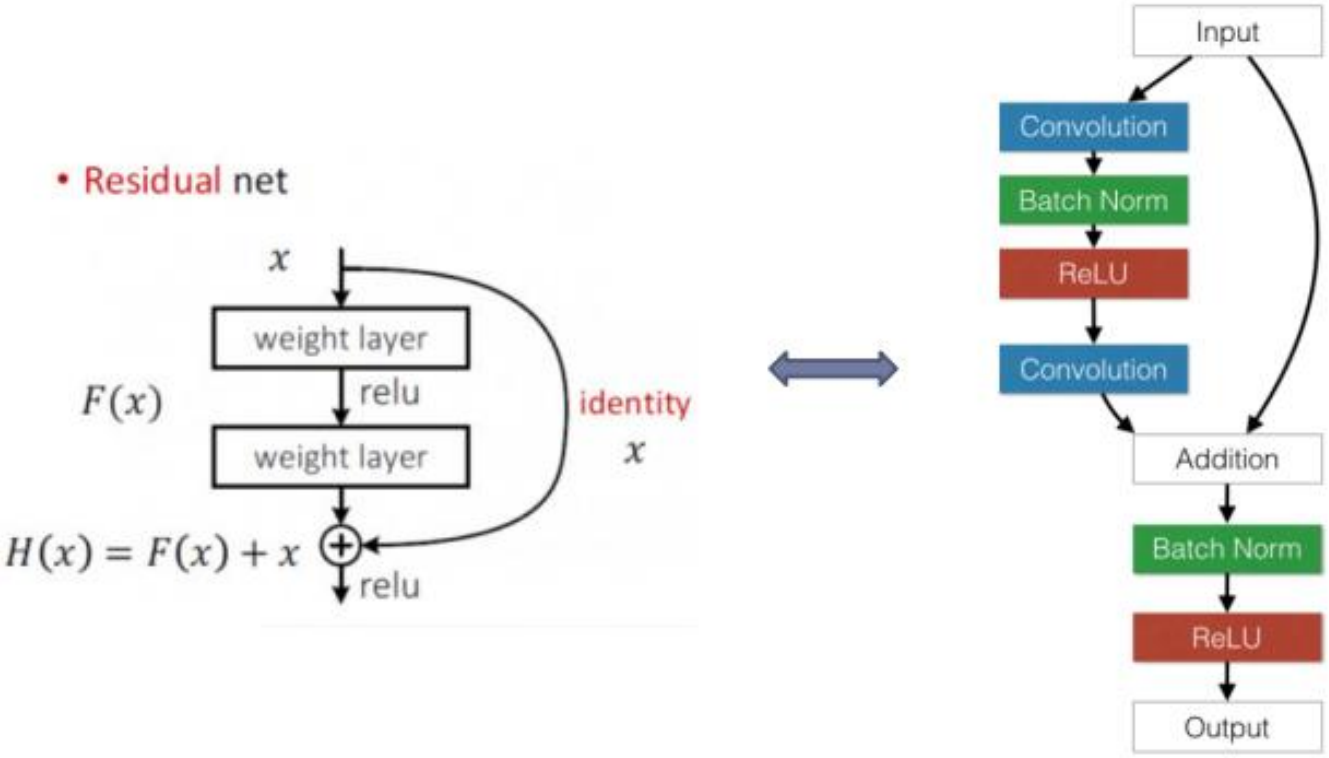
- 实线：特征图尺寸和特征数量不变，直接相连；
- 虚线：特征图尺寸减半，特征图数量翻倍；
- 两种方法：

a. 以stride=2直接取值，不够的特征补0(不引入额外参数)；

b. 用stride=2，特征数量翻倍的的1x1卷积做映射，卷积的权值经过学习得到，会引入额外参数；

ResNet

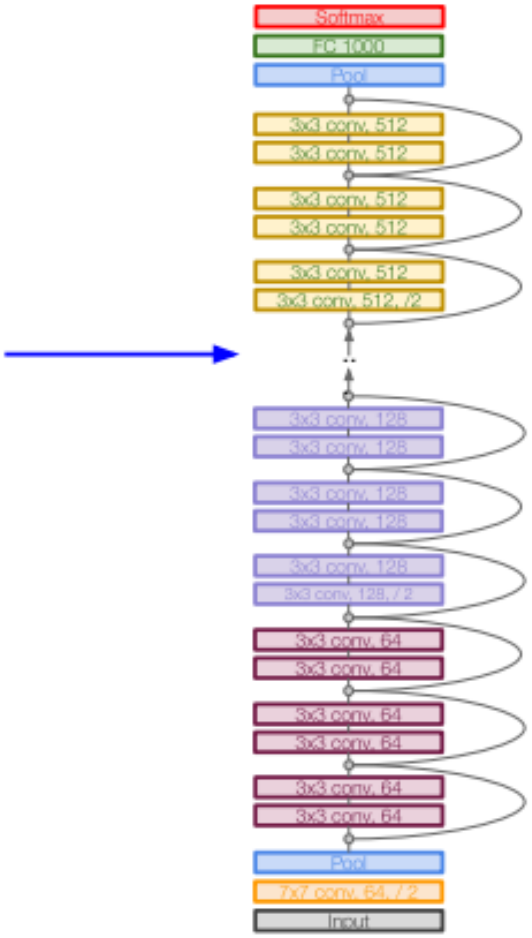
- BatchNorm
  - 为避免梯度消失，在每个卷积层之后、激活函数之前增加BN层



Case Study: ResNet

[He et al., 2015]

Total depths of 34, 50, 101, or 152 layers for ImageNet



Case Study: ResNet

[He et al., 2015]

Experimental Results

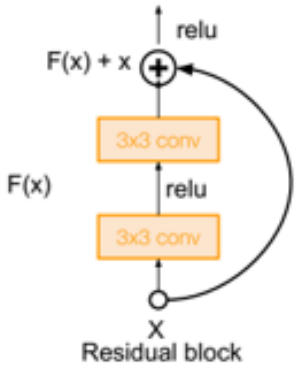
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than "human performance"! (Russakovsky 2014)

ResNet in Keras



假设我们有一个四维输入张量x。

如果输入输出特征图的尺寸相同

```
from keras import layers

x = ...
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)

y = layers.add([y, x])
```

对 x 进行变换

将原始 x 与输出特征相加

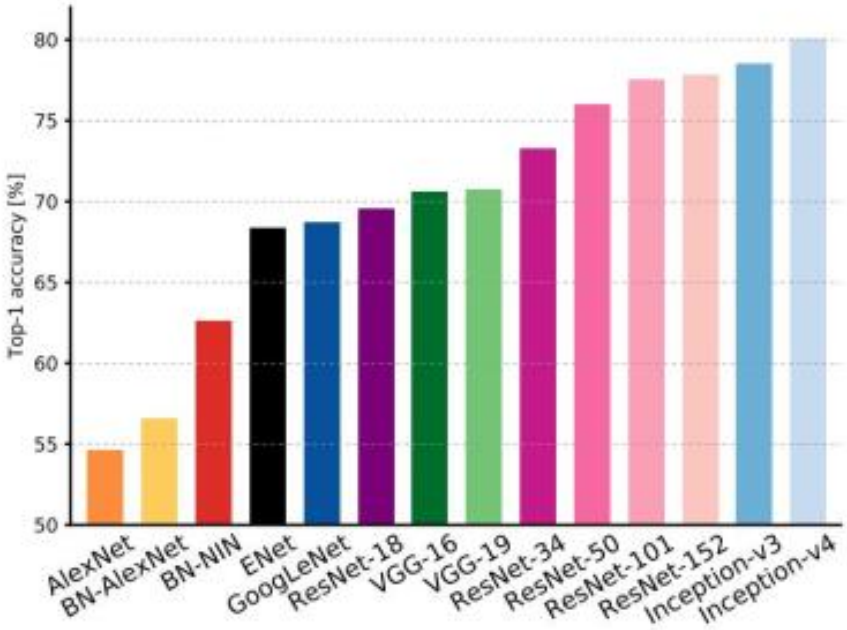
如果输入输出特征图的尺寸不相同

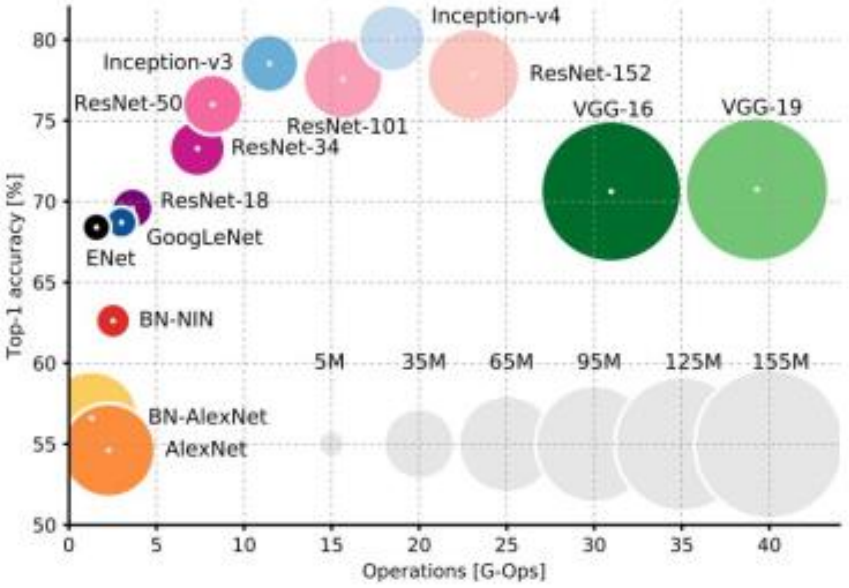
```
from keras import layers

x = ...
y = layers.Conv2D(128, 3, activation='relu', padding='same')(x)
y = layers.Conv2D(128, 3, activation='relu', padding='same')(y)
y = layers.MaxPooling2D(2, strides=2)(y)

residual = layers.Conv2D(128, 1, strides=2, padding='same')(x)

y = layers.add([y, residual])
```

将残差张量与输出特征相加

使用 1×1 卷积，将原始 x 张量线性下采样为与 y 具有相同的形状

Comparing complexity...

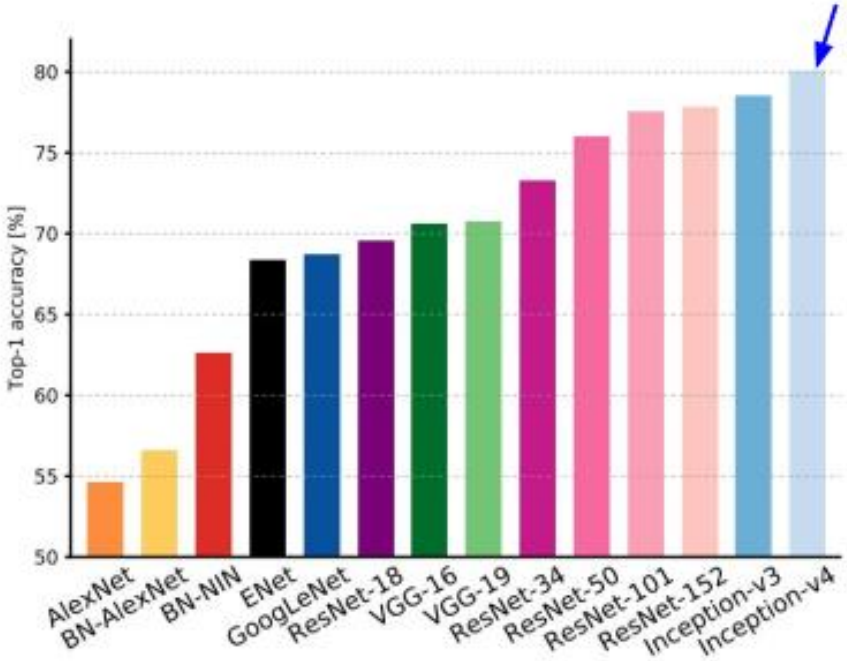


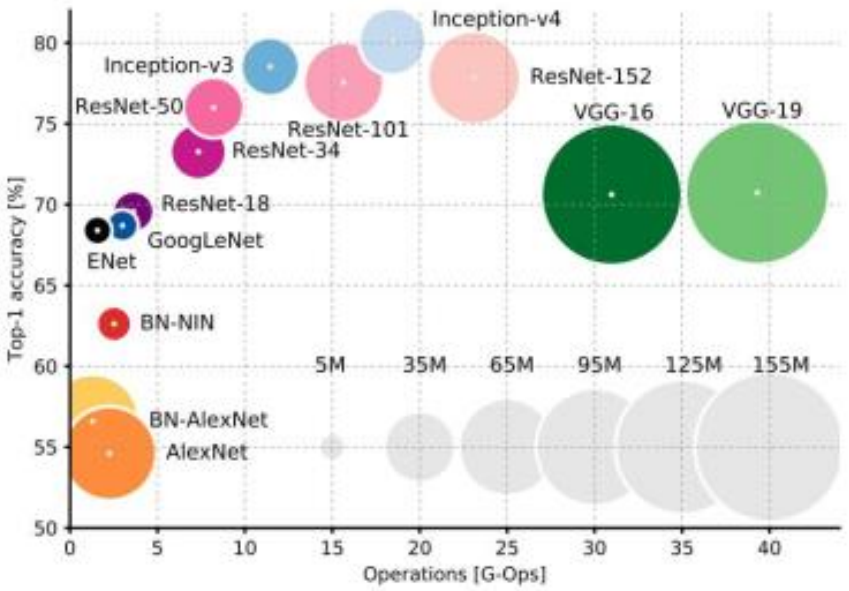


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

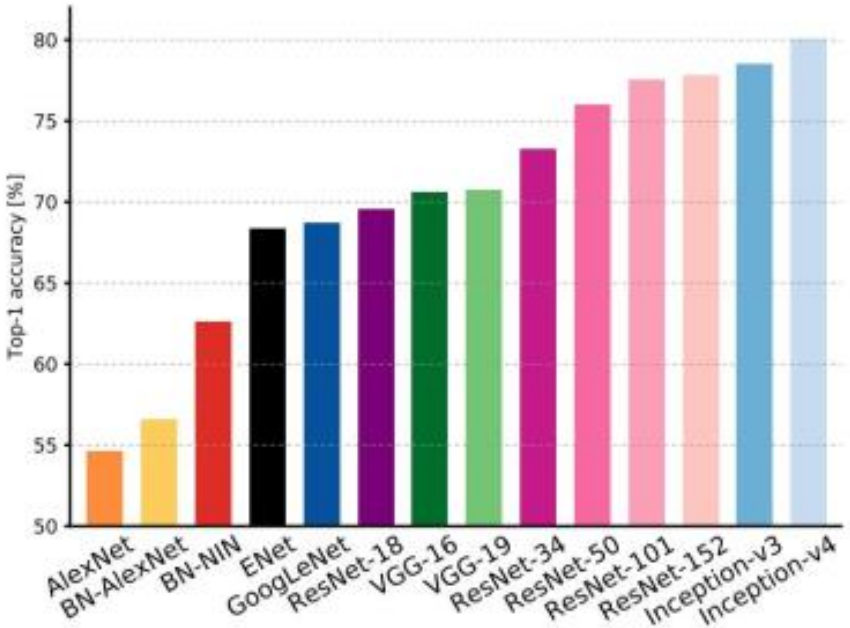
Inception-v4: Resnet + Inception!



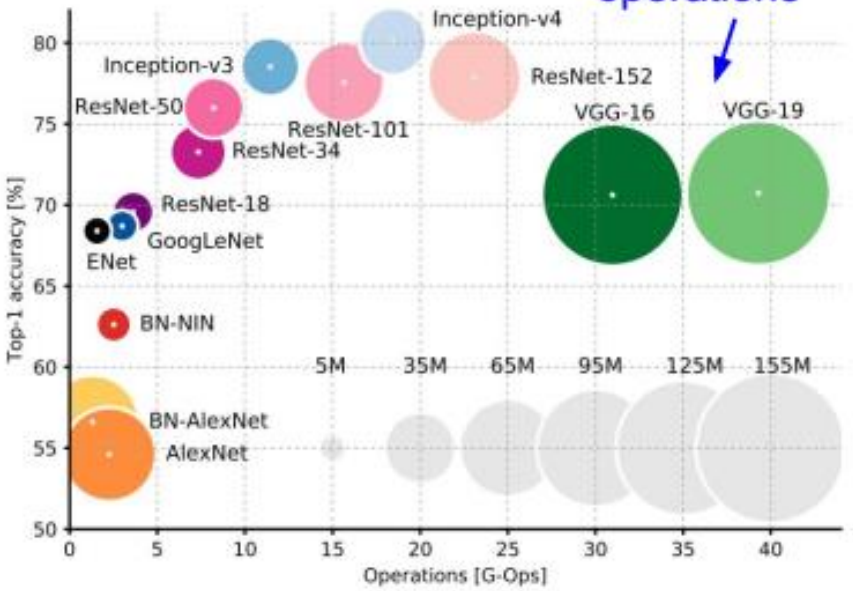


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

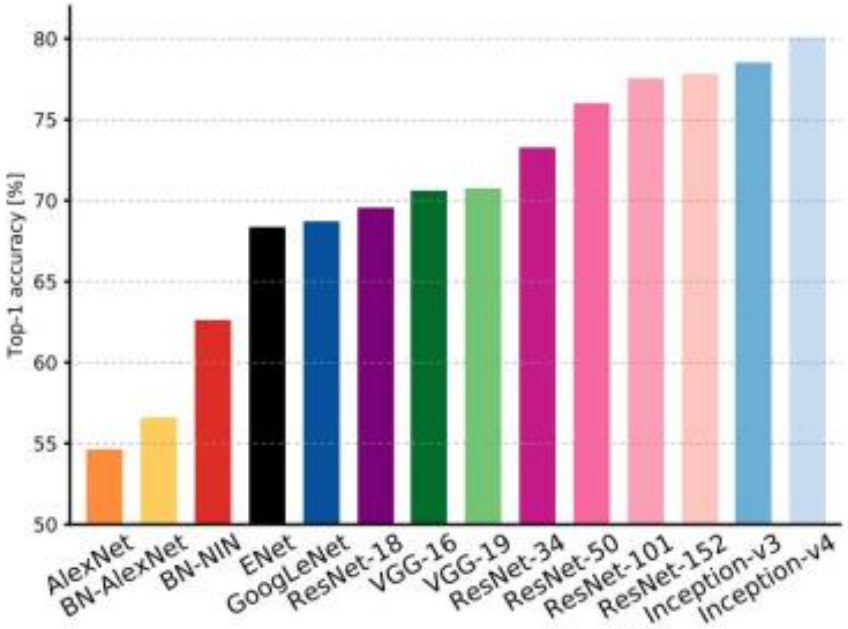


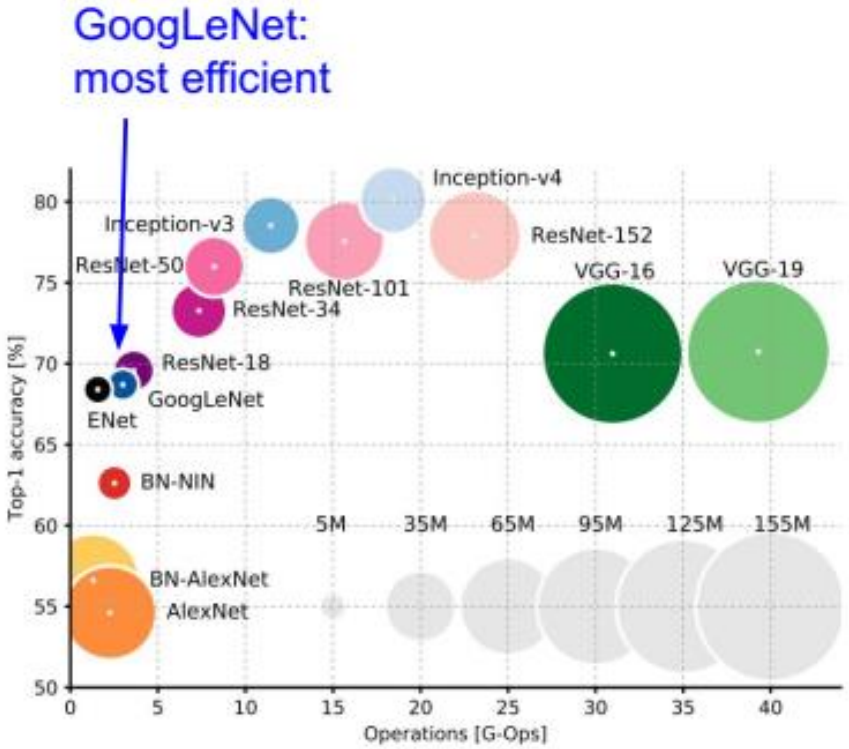
VGG: Highest memory, most operations



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

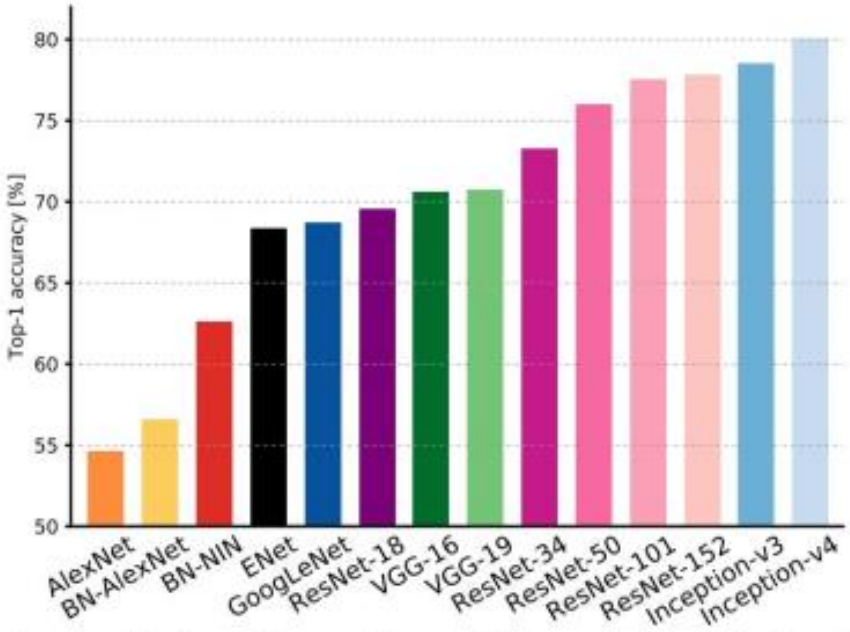
Comparing complexity...



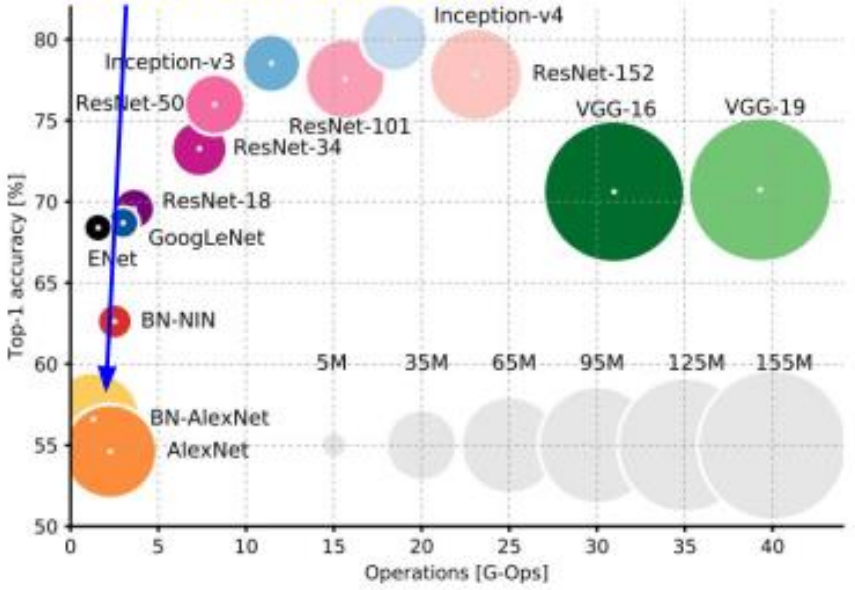


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

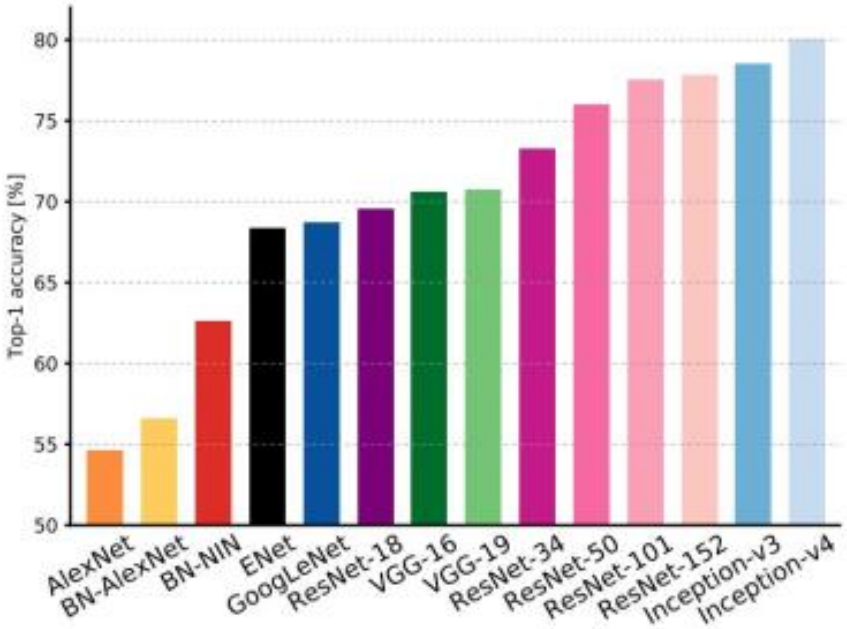


AlexNet:
Smaller compute, still memory heavy, lower accuracy

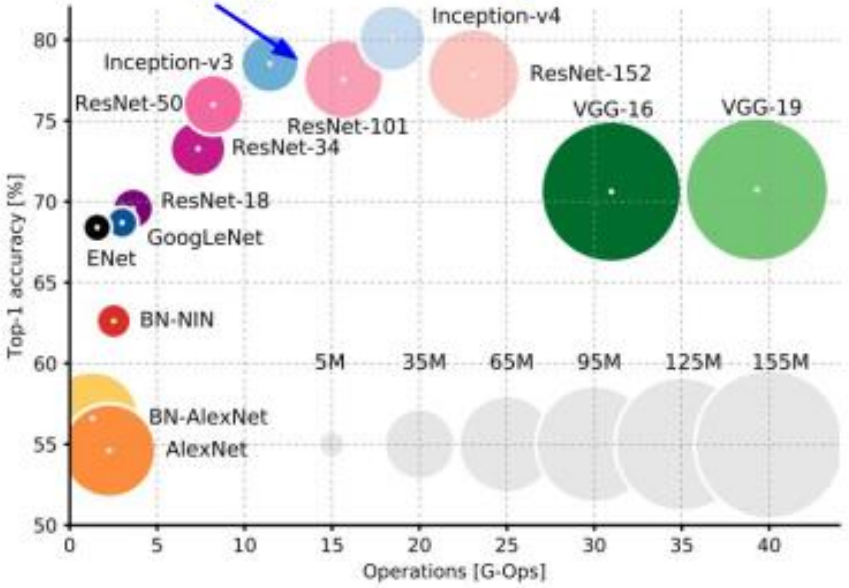


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Comparing complexity...

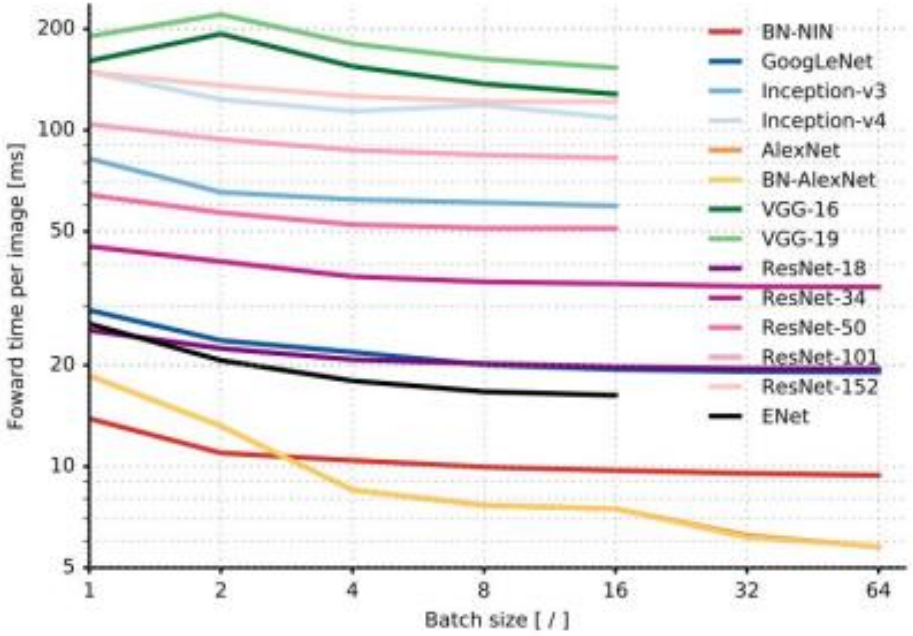


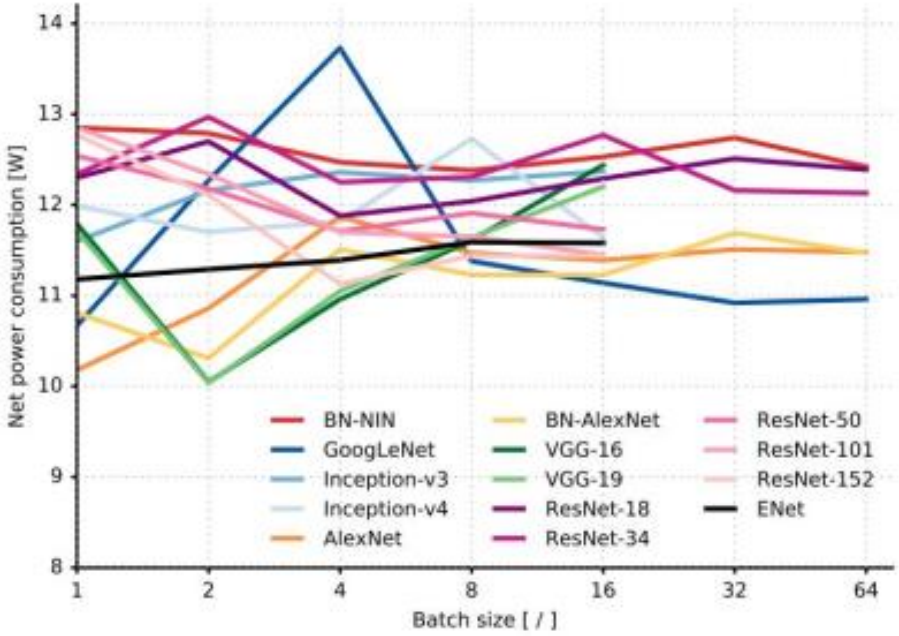
ResNet:
Moderate efficiency depending on model, highest accuracy



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Forward pass time and power consumption





An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Focus

- Classification based on CNN

- Object detection based on CNN

  R-CNN , Fast R-CNN， Faster R-CNN
  YOLO
  SSD

Object Detection

Classification

CAT

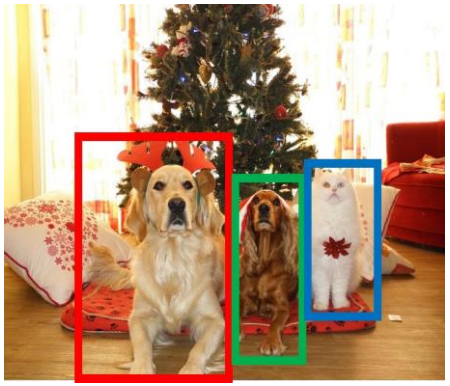
No spatial extent

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



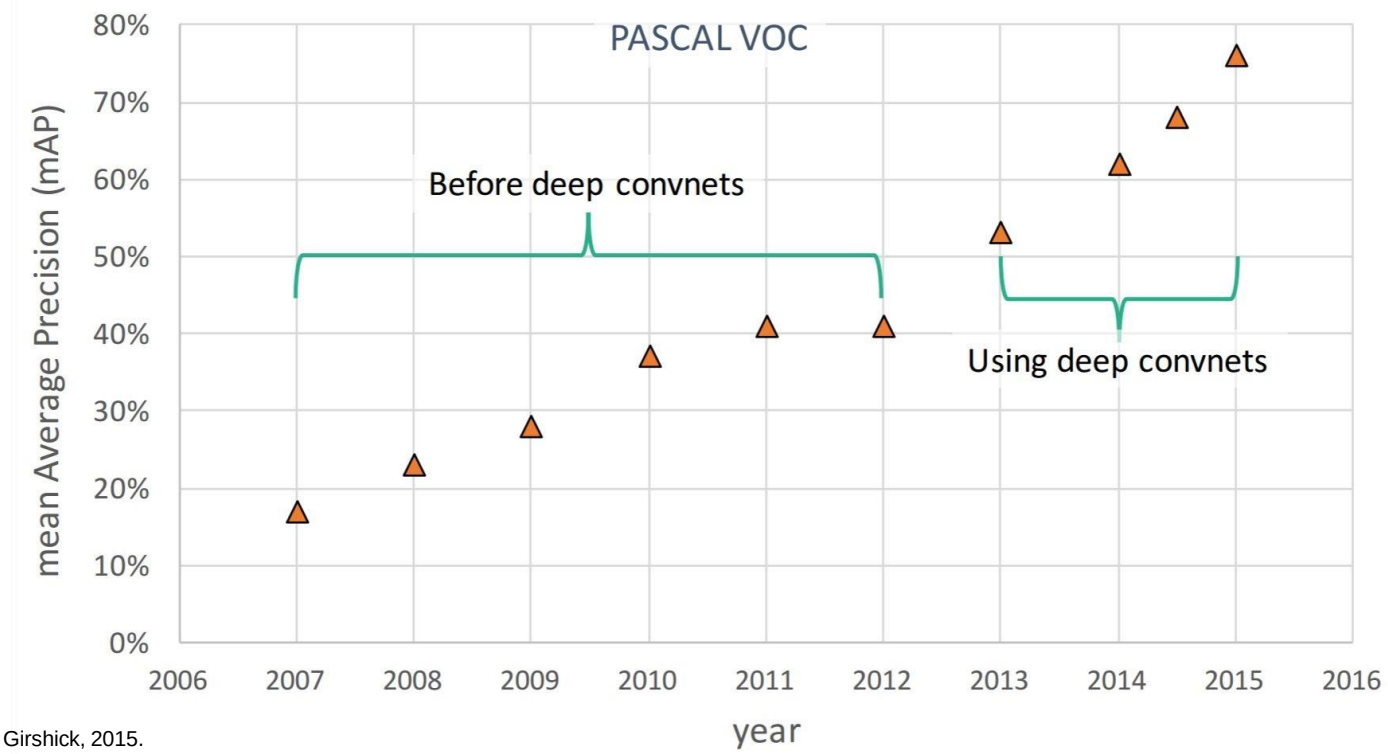
DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Multiple Object

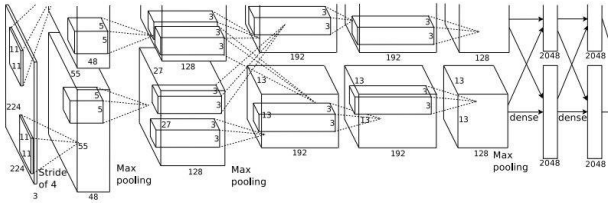
Object Detection: Impact of Deep Learning



Object Detection: Single Object

(Classification + Localization)

This image is CC0 public domain



Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Vector: 4096

Fully Connected: 4096 to 4

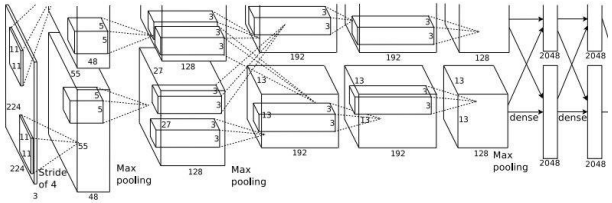
Box Coordinates
(x, y, w, h)

Treat localization as a regression problem!

Object Detection: Single Object

(Classification + Localization)

This image is CC0 public domain



Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

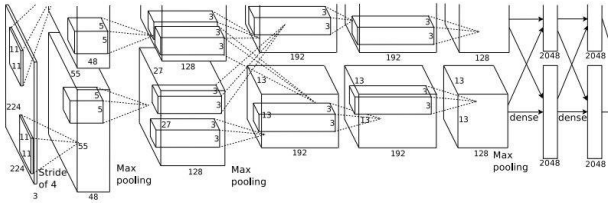
L2 Loss

Correct box: (x', y', w', h')

Treat localization as a regression problem!

Object Detection: Single Object

(Classification + Localization)



Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

+ → **Loss**

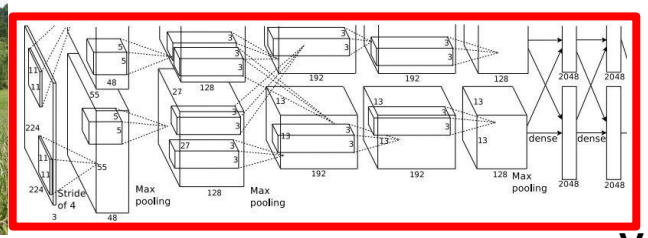
Multitask Loss

Treat localization as a regression problem!

Object Detection: Single Object

(Classification + Localization)

This image is CC0 public domain



Often pretrained on ImageNet
(Transfer learning)

Vector: 4096

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct label: Cat

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

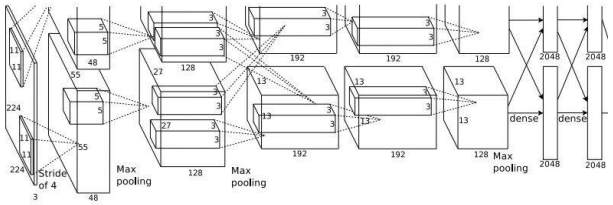
L2 Loss

Correct box: (x', y', w', h')

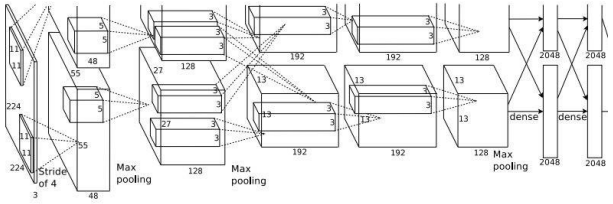
Loss

Treat localization as a regression problem!

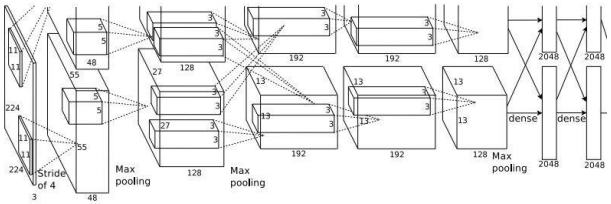
Object Detection: Multiple Objects



CAT: (x, y, w, h)



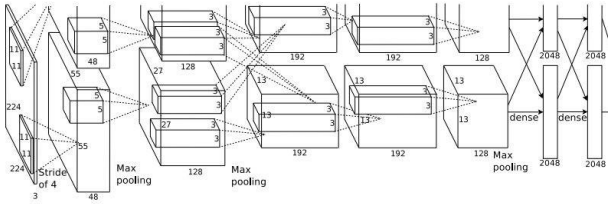
DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)



DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

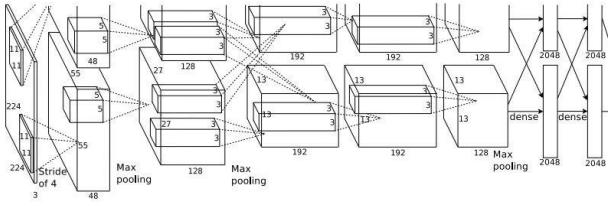
Object Detection: Multiple Objects

Each image needs a different number of outputs!



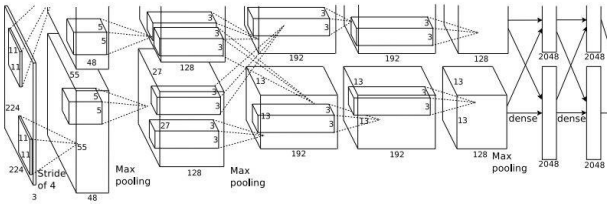
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

12 numbers

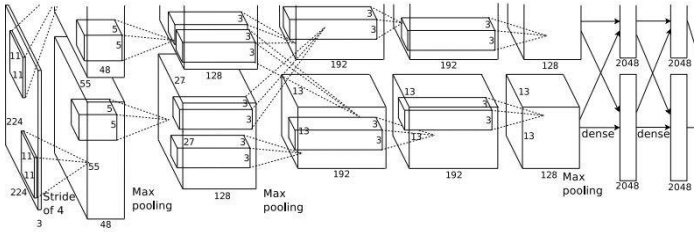


DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
….

Many numbers!

Object Detection: Multiple Objects

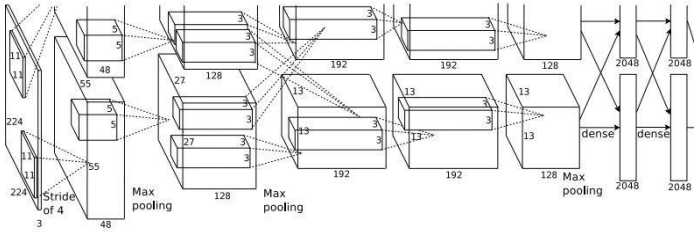
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? NO
Background? YES

Object Detection: Multiple Objects

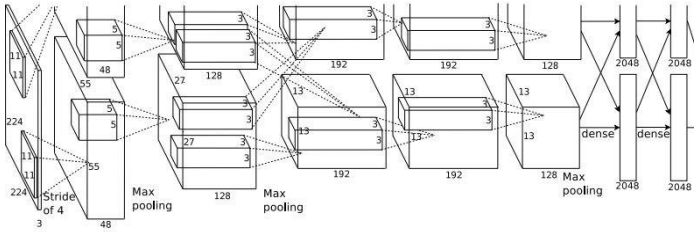
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Object Detection: Multiple Objects

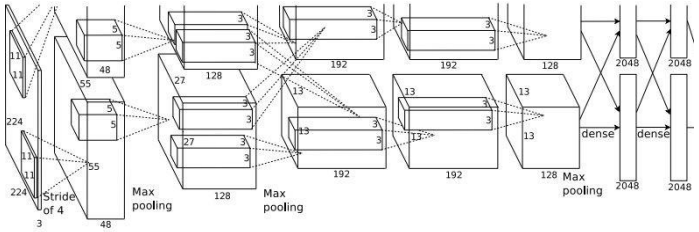
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Object Detection: Multiple Objects

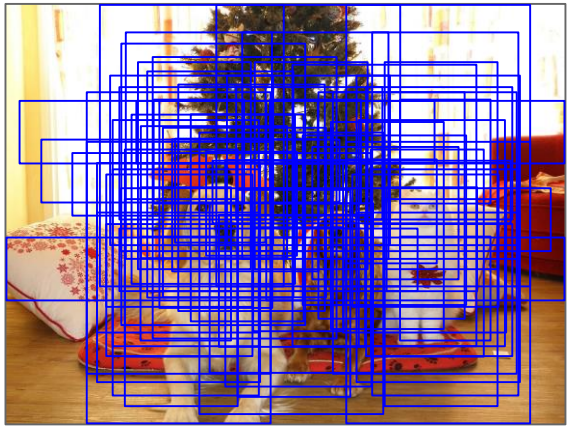
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

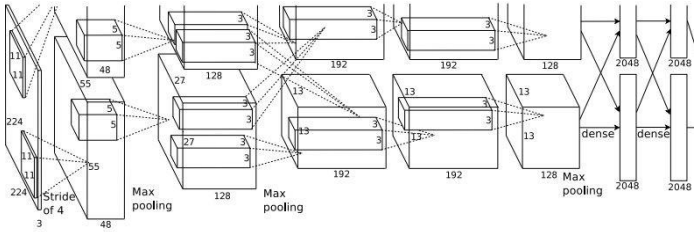


Dog? NO
Cat? YES
Background? NO

Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



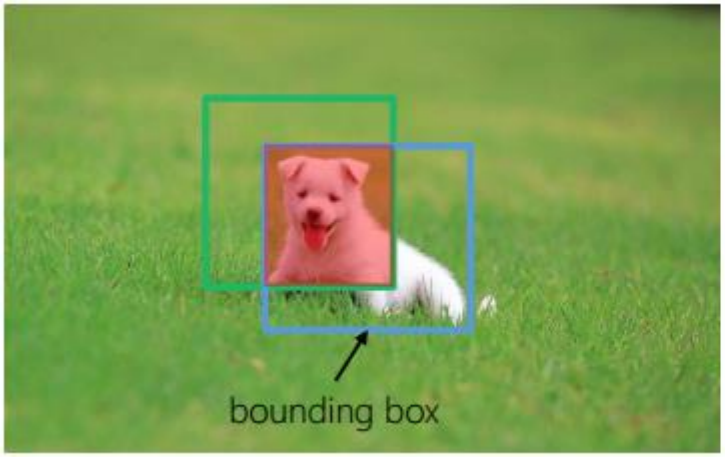


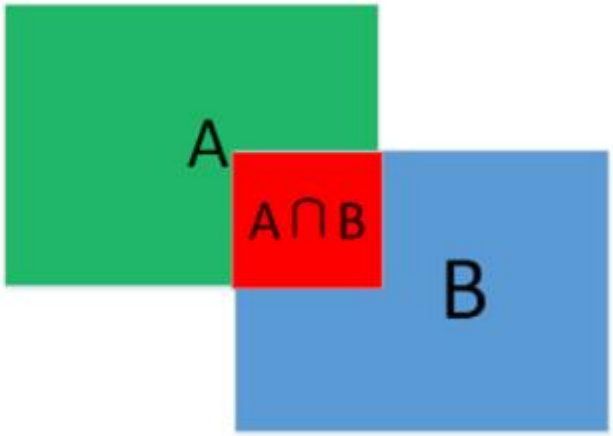
Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

目标检测的评估指标

- IoU（交并比）：衡量定位准确度 $IOU = \frac{A \cap B}{A \cup B}$
  - 一般，IoU≥0.5，则认为定位正确。





- mAP(mean Average Precision平均精度均值）：用于衡量分类模型在测试集上的优劣程度。
  - mAP值越高，则表示检测结果越好。
- 检测速度

主流的目标检测算法

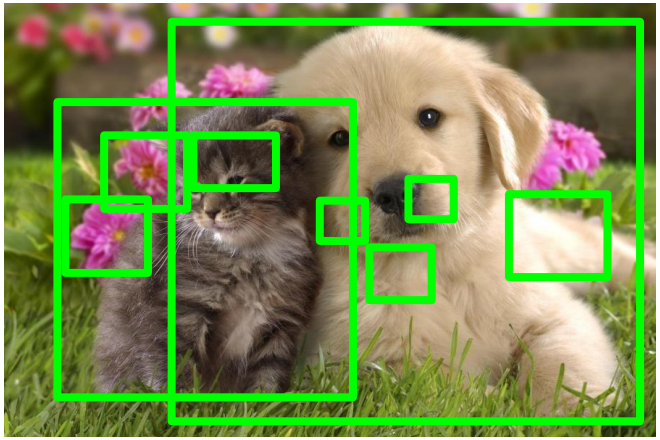
（1）One-Stage目标检测算法，这类检测算法不需要Region Proposal阶段，可以通过一个Stage直接产生物体的类别概率和位置坐标值，比较典型的算法有YOLO、SSD和CornerNet；

（2）Two-Stage目标检测算法，这类检测算法将检测问题划分为两个阶段：

- 第一个阶段首先产生候选区域（Region Proposals），包含目标大概的位置信息；
- 第二个阶段对候选区域进行分类和位置精修，这类算法的典型代表有R-CNN (Region-CNN)，Fast R-CNN，Faster R-CNN等。

Region Proposals: Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



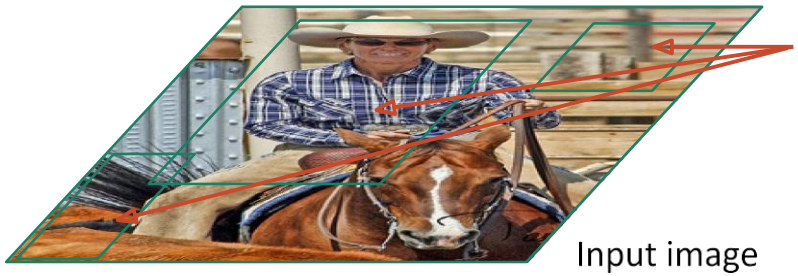
Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

R-CNN

Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

R-CNN

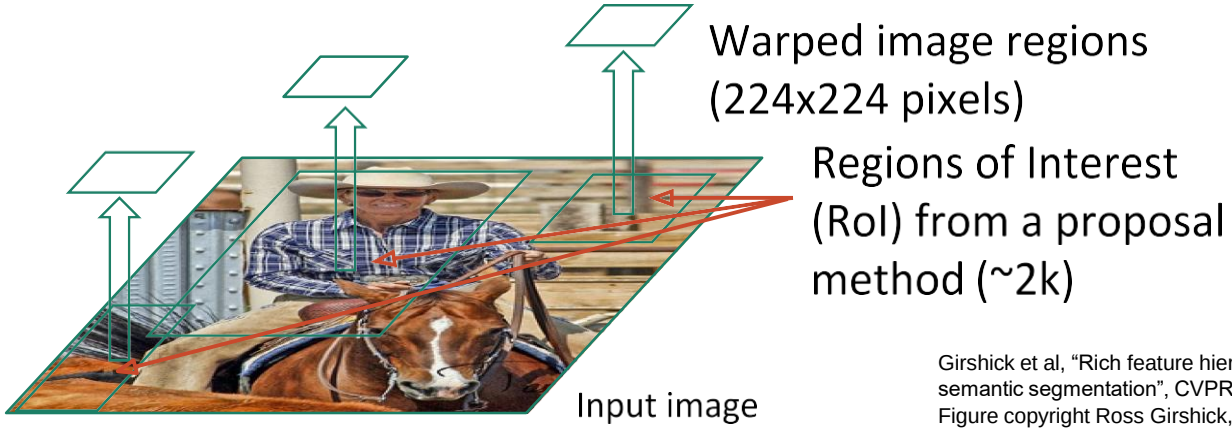


Input image

Regions of Interest (RoI) from a proposal method (~2k)

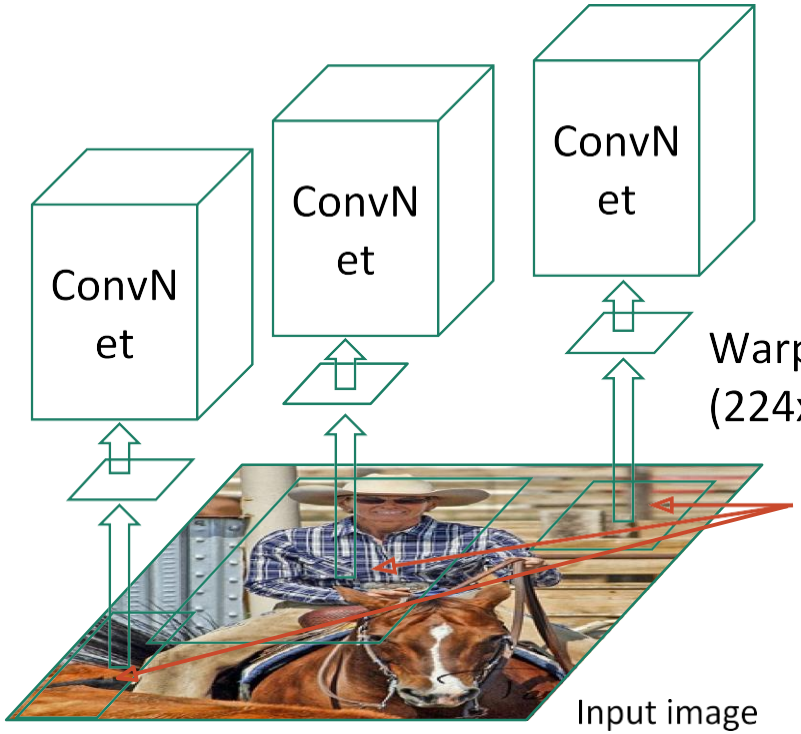
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



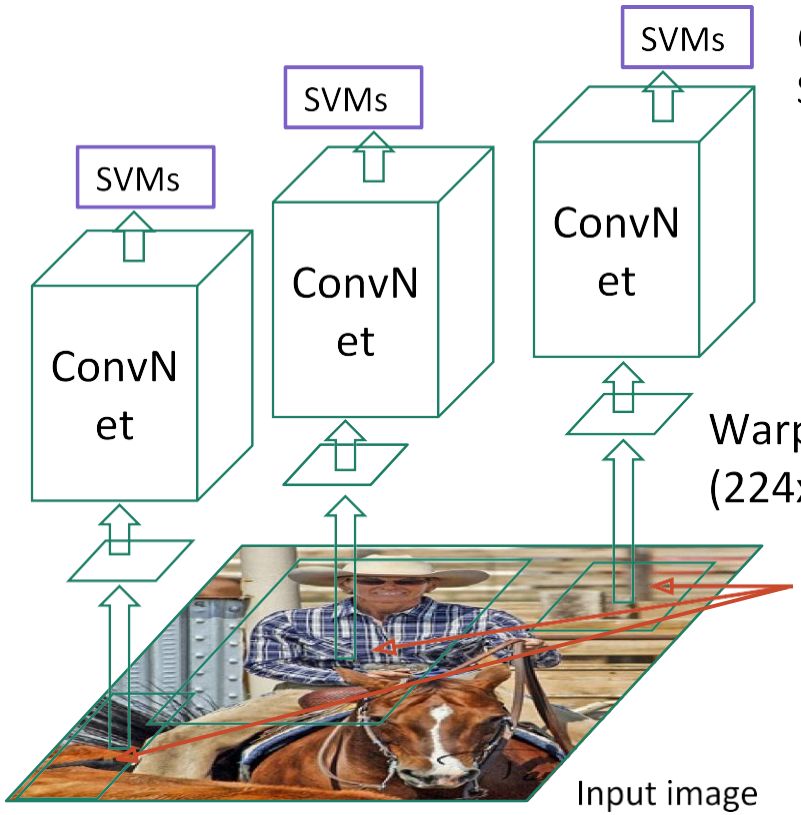
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

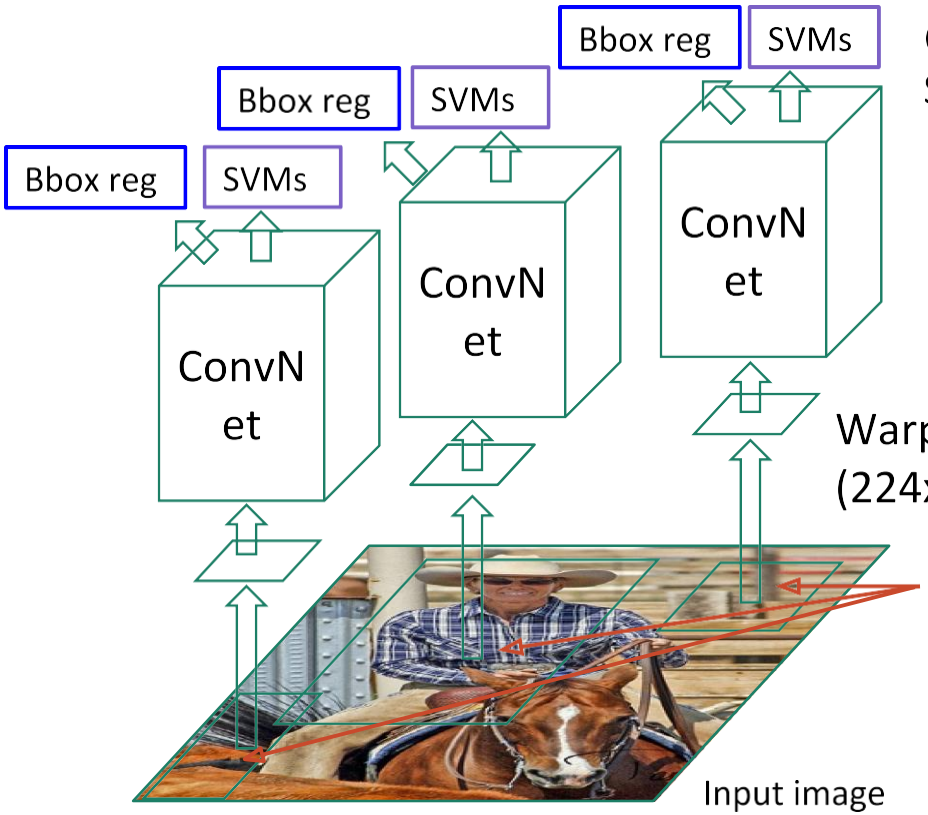
R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

1. SVM分类

多个**SVM**

每个**SVM**做二分类

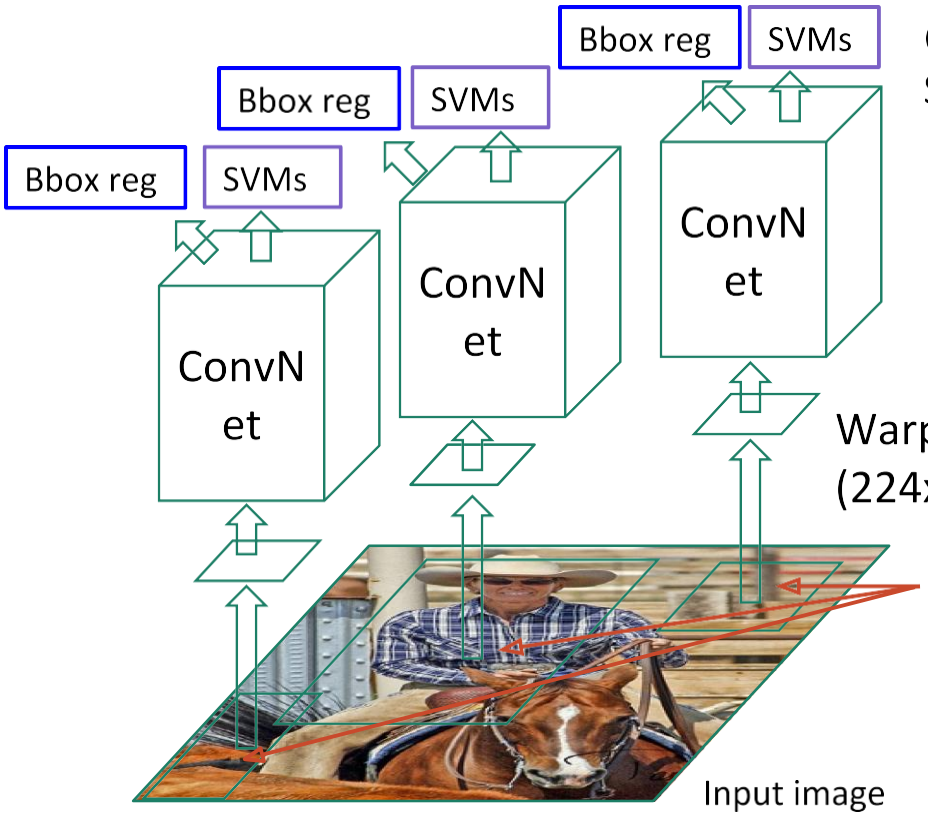
2. 非极大值抑制

3. BoundingBox回归

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

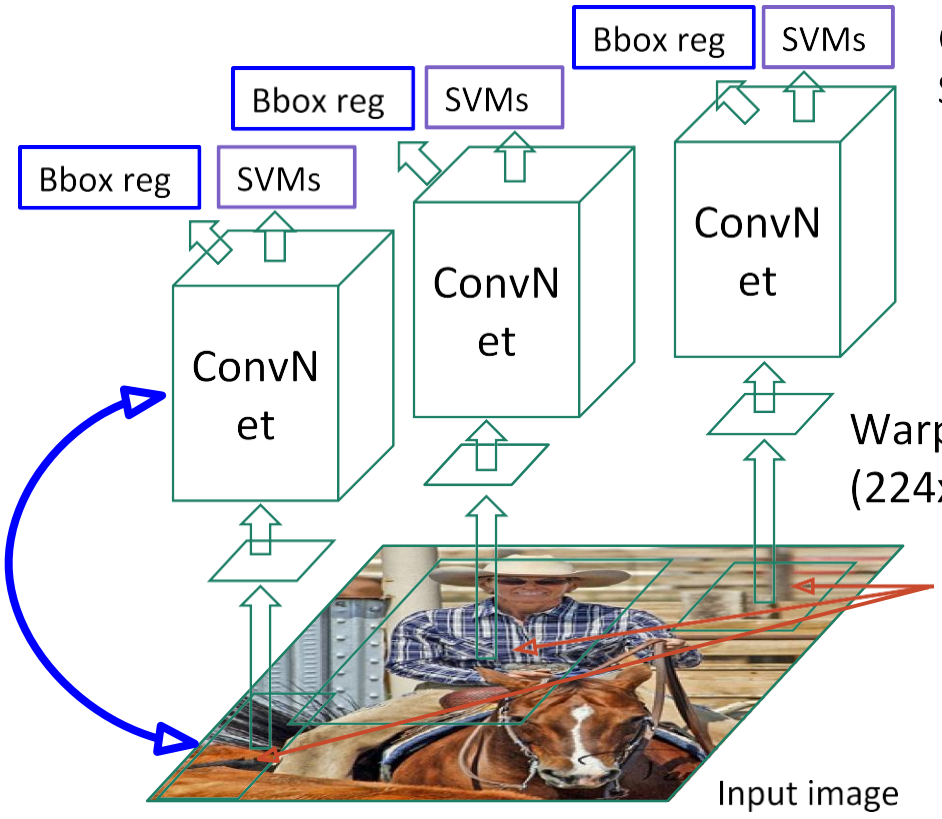
Regions of Interest (RoI) from a proposal method (~2k)

Problem: Very slow! Need to do ~2k independent forward passes for each image!

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

"Slow" R-CNN

Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)



Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

Regions of Interest (RoI) from a proposal method (~2k)

Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Process image before cropping! Swap convolution and cropping!

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

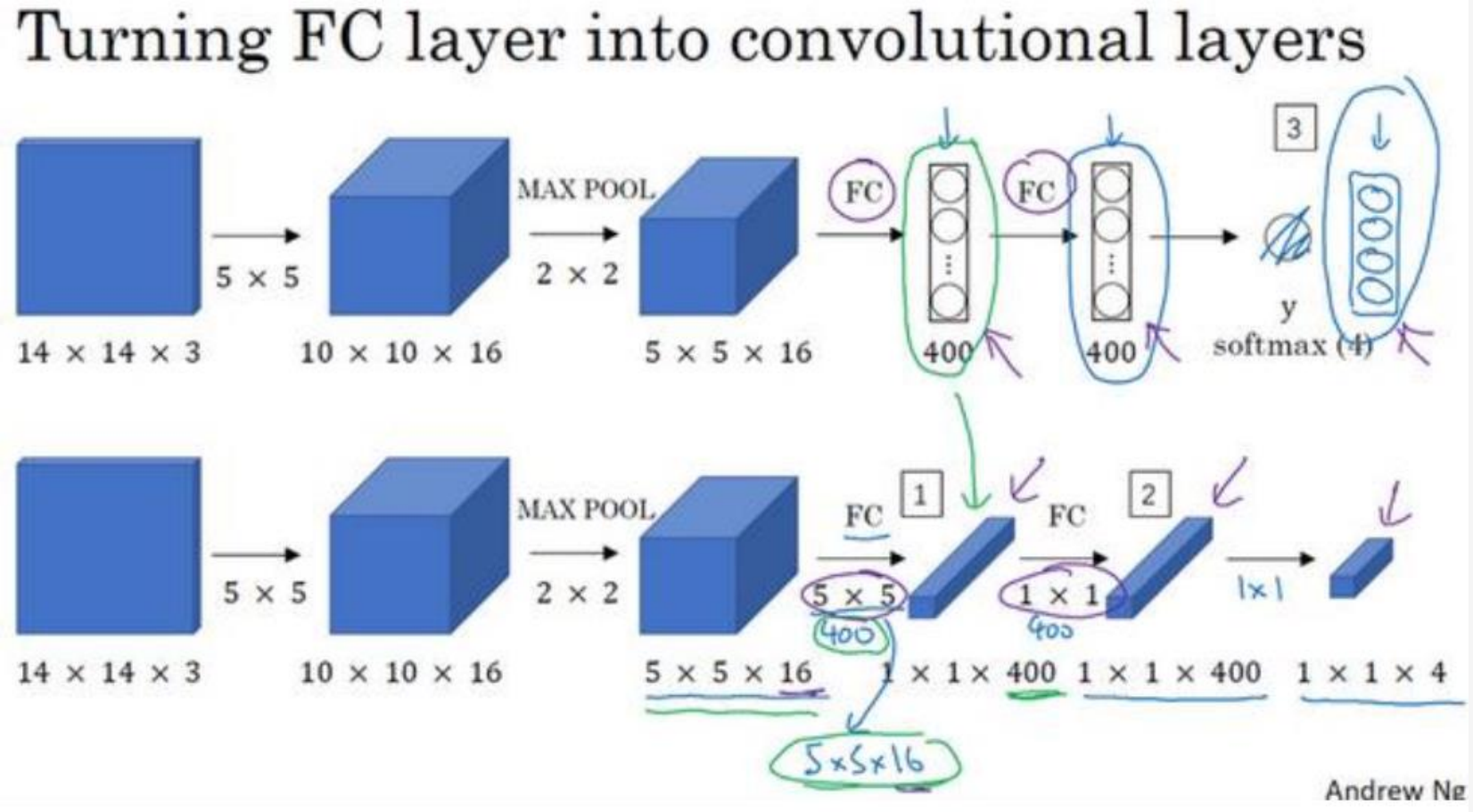
总结：R-CNN

虽然R-CNN在ILSVRC 2013数据集上的mAP由Overfeat的24.3%提升到了31.4%，第一次有了质的改变。

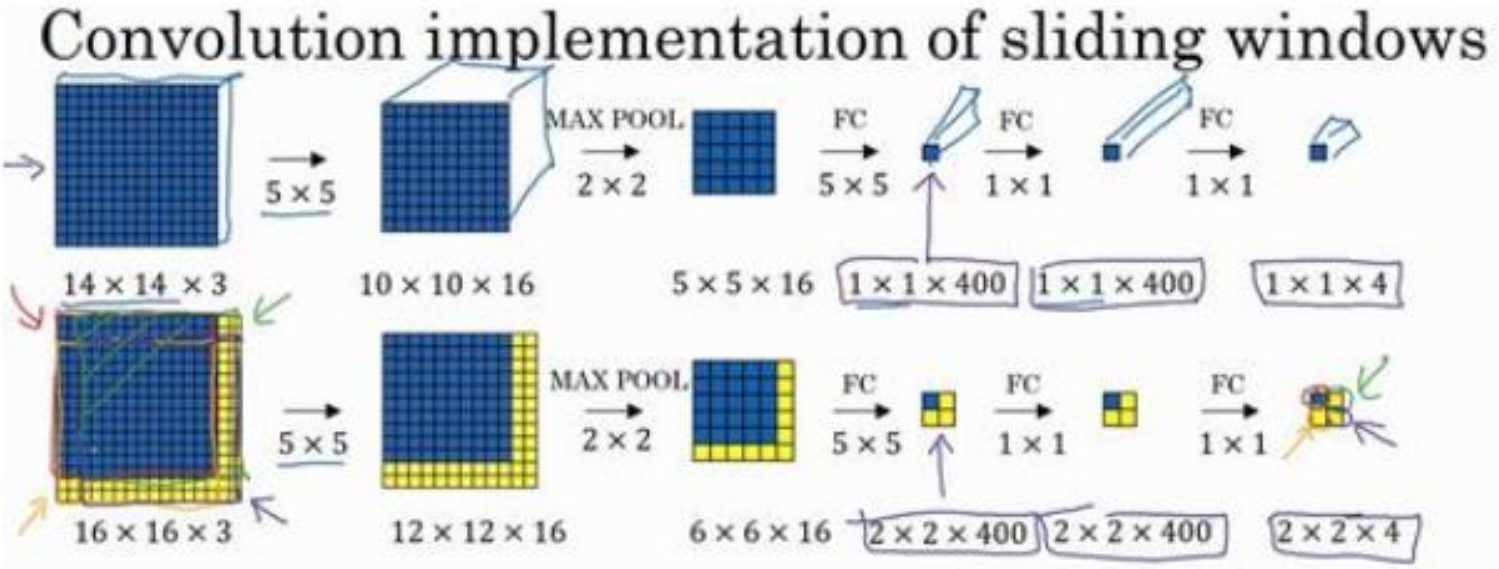
但R-CNN有很多缺点：

- 重复计算：R-CNN虽然不再是穷举，但依然有两千个左右的候选框，这些候选框都需要进行CNN操作，计算量依然很大，其中有不少其实是重复计算；
- SVM模型：而且还是线性模型，在标注数据不缺的时候显然不是最好的选择；
- 训练测试分为多步：区域提名、特征提取、分类、回归都是断开的训练的过程，中间数据还需要单独保存；训练的空间和时间代价很高
- GPU上处理一张图片需要13秒，CPU上则需要53秒。

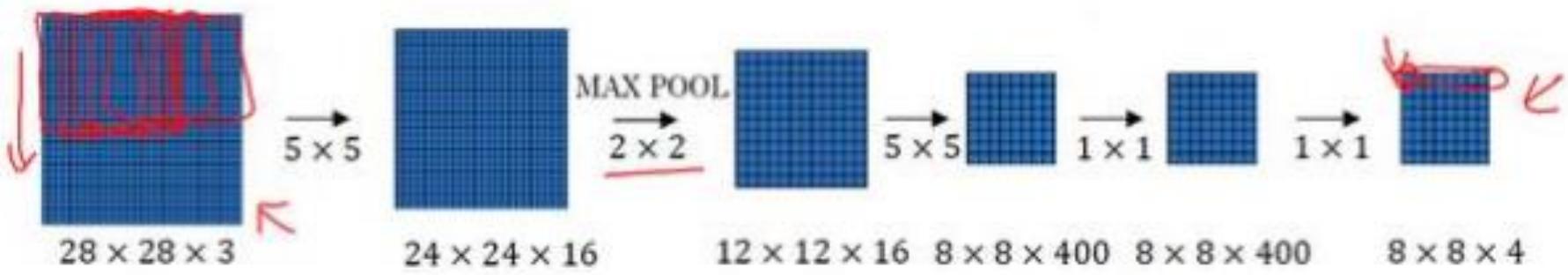
如何把神经网络的全连接层转化成卷积层？

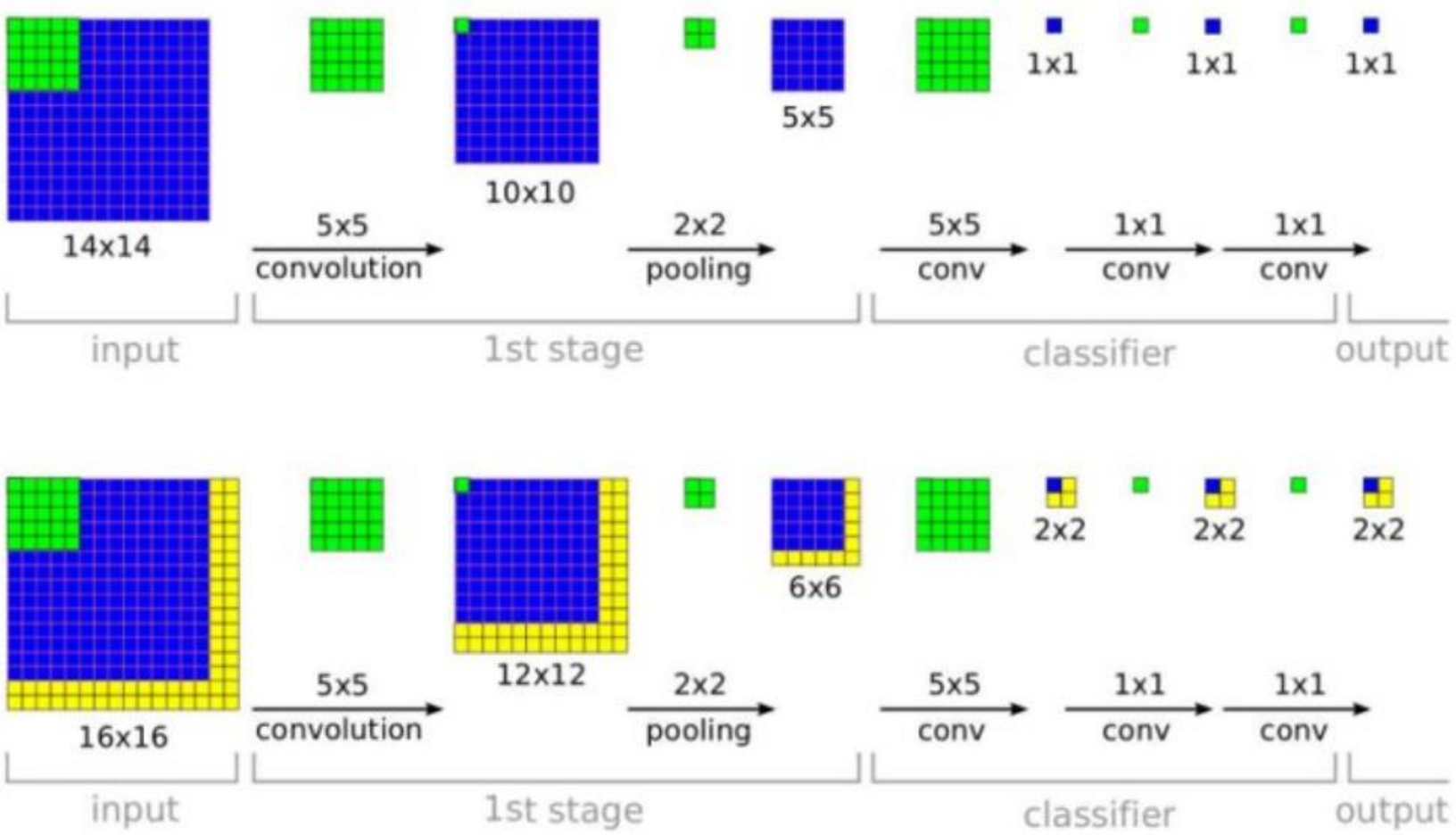


Sermanet, Pierre, et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks."Eprint Arxiv(2013).



不需要把输入图像分割成四个子集，分别执行前向传播，而是把它们作为一张图片输入给卷积网络进行计算

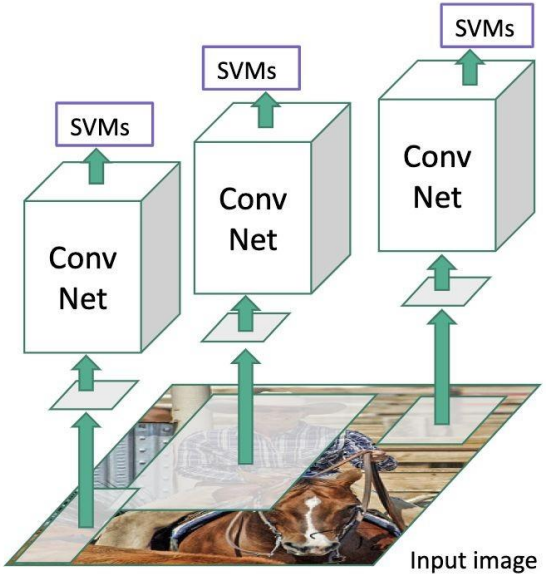




Sermanet, Pierre, et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks."Eprint Arxiv(2013).

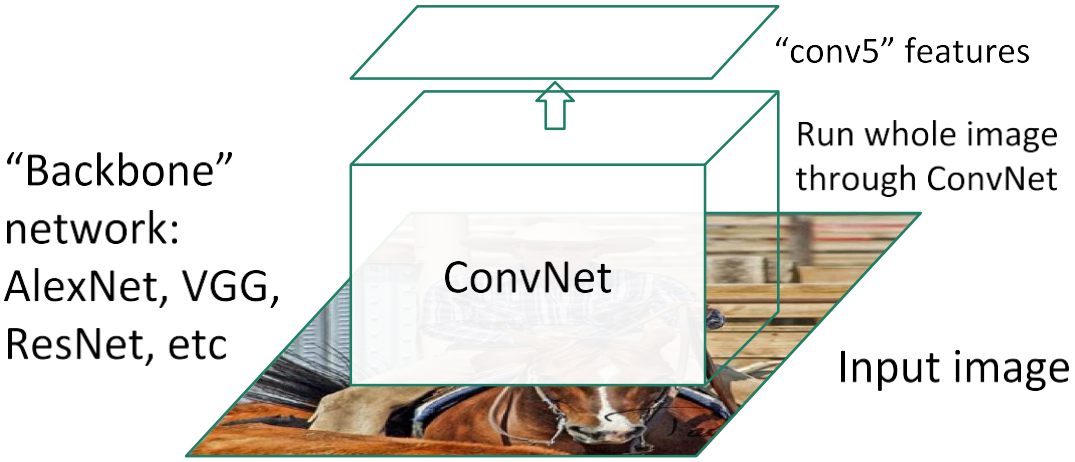
Fast R-CNN

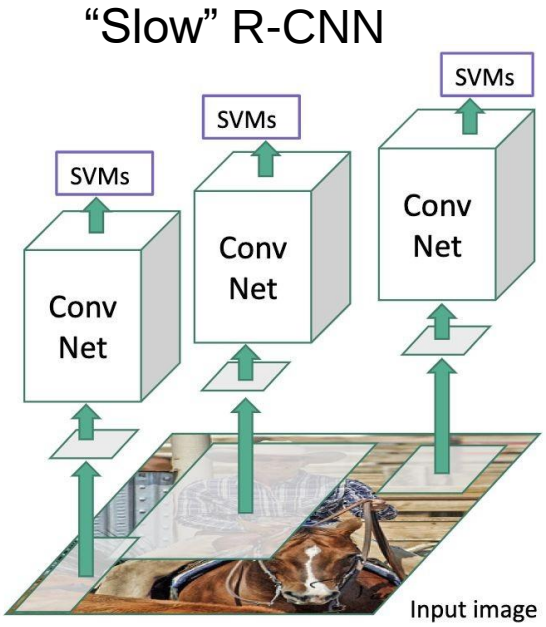
Input image



Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

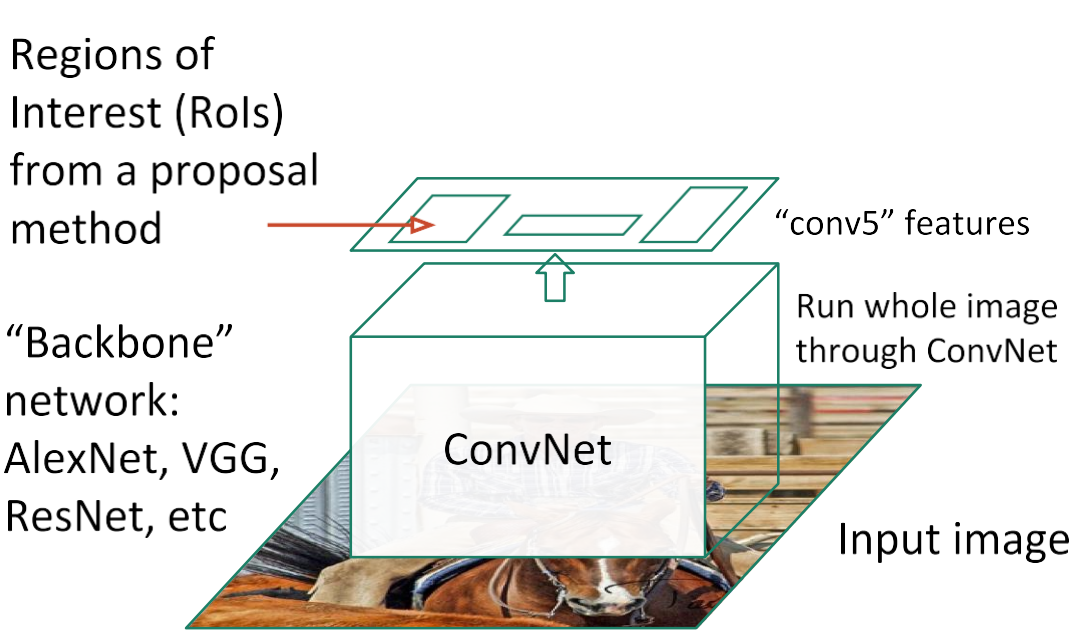
Fast R-CNN

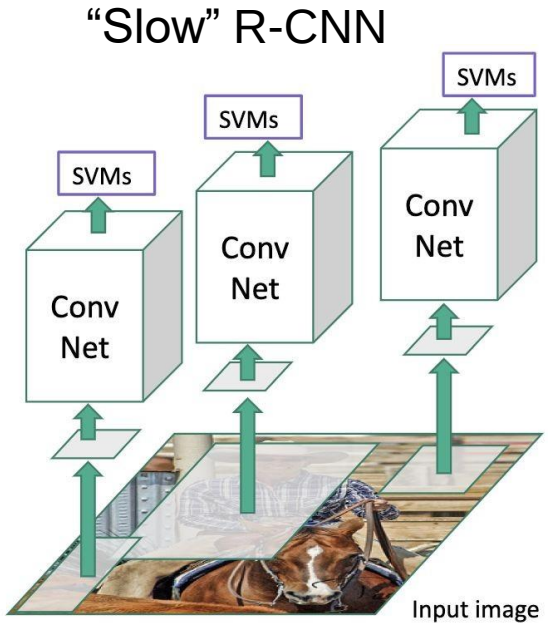




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

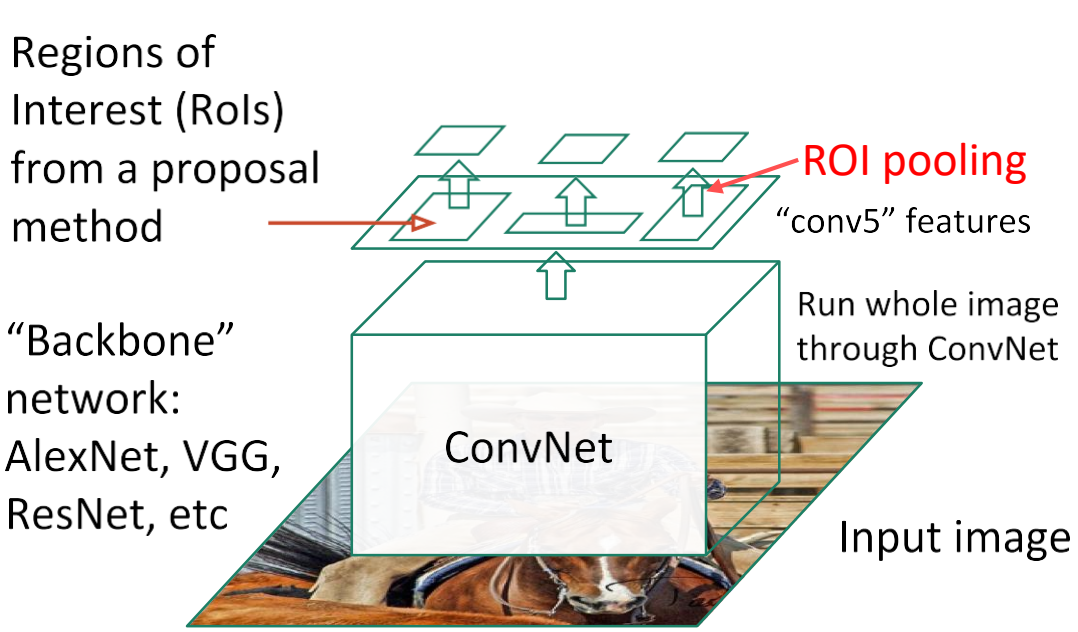
Fast R-CNN

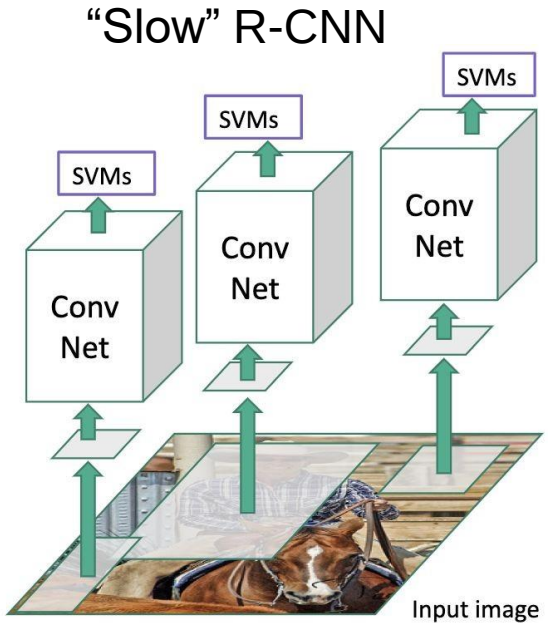




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

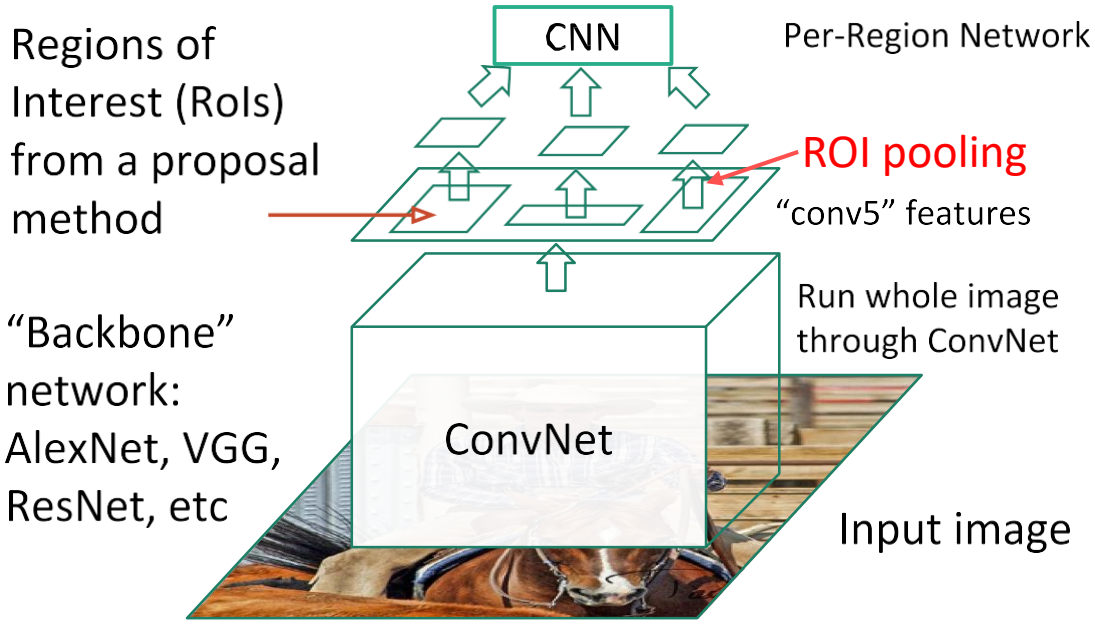
Fast R-CNN

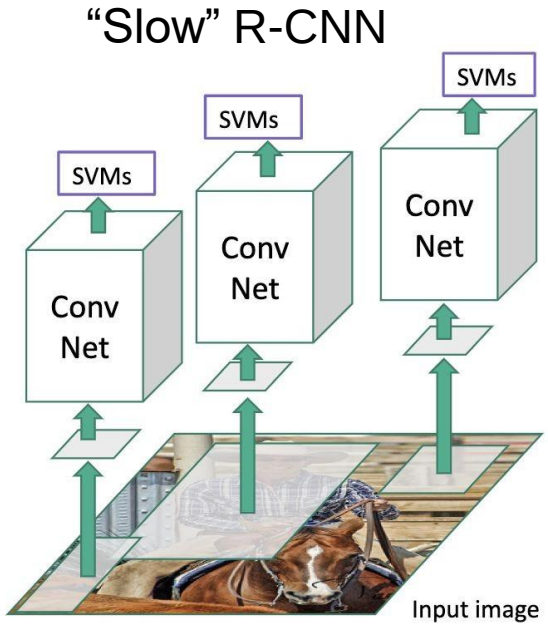




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

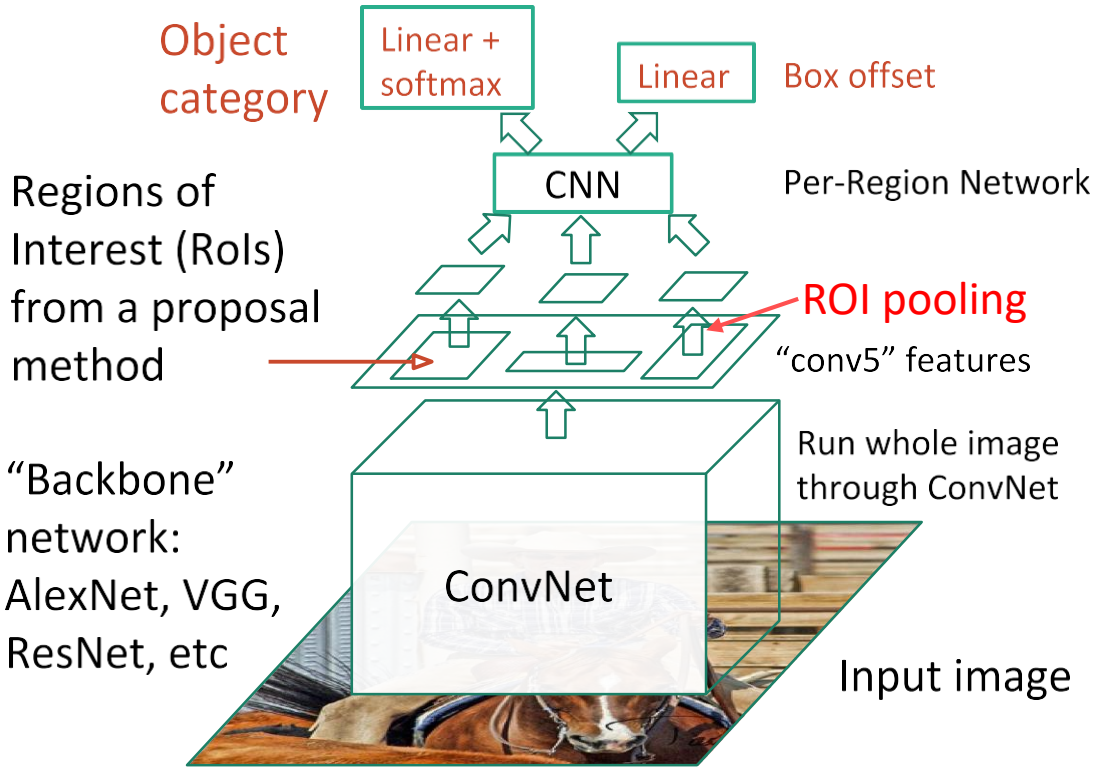
Fast R-CNN

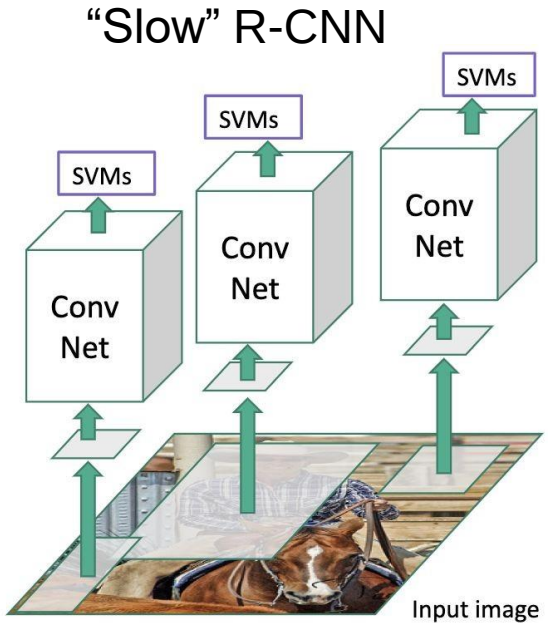




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

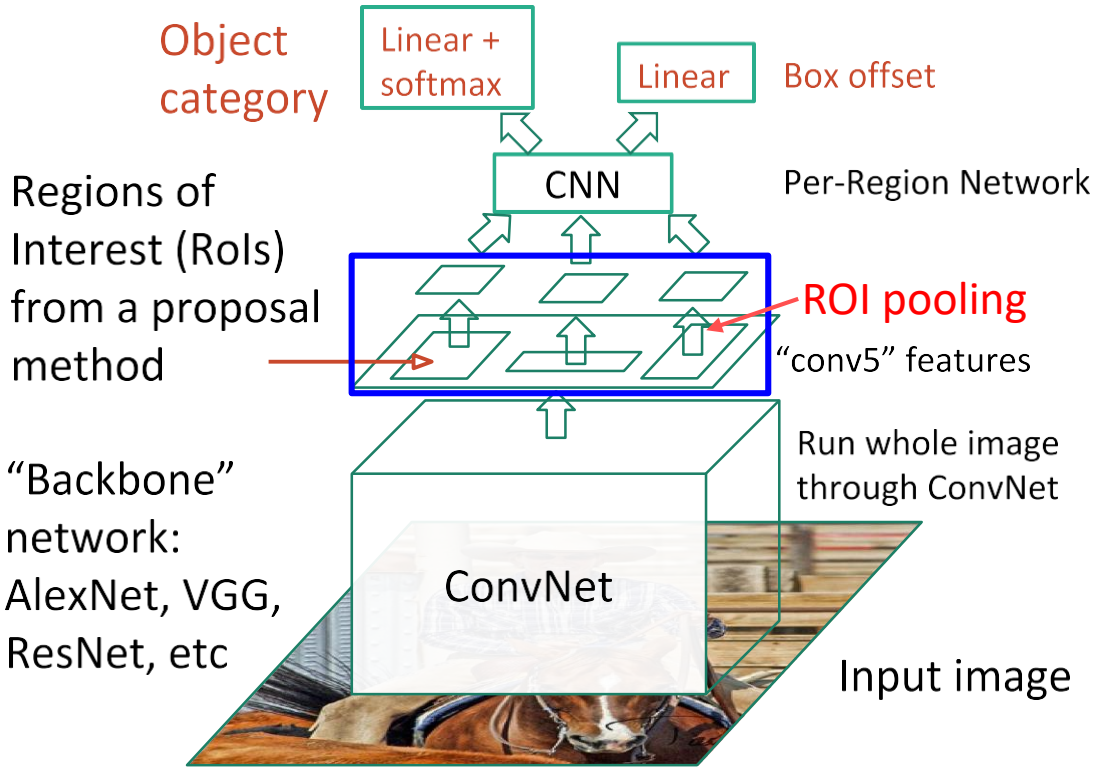
Fast R-CNN

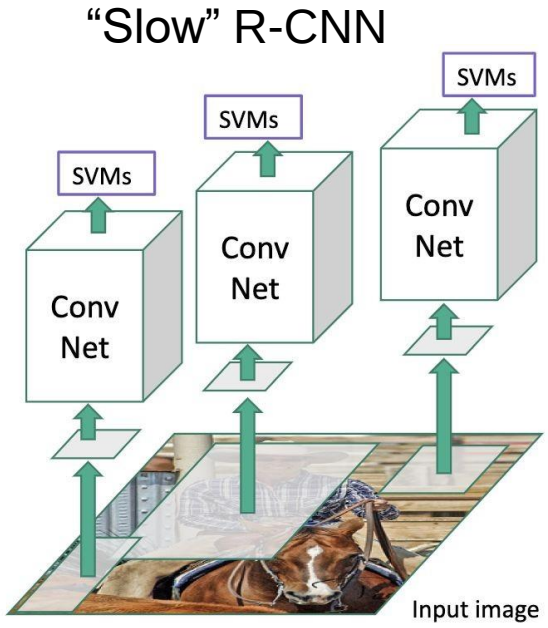




Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN





Girshick, "Fast R-CNN", ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

多任务损失函数：

$p = (p_0, \ldots, p_K)$ $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$

$$L(p, k^*, t, t^*) = L_{cls}(p, k^*) + \lambda[k^* \geq 1] L_{loc}(t, t^*)$$

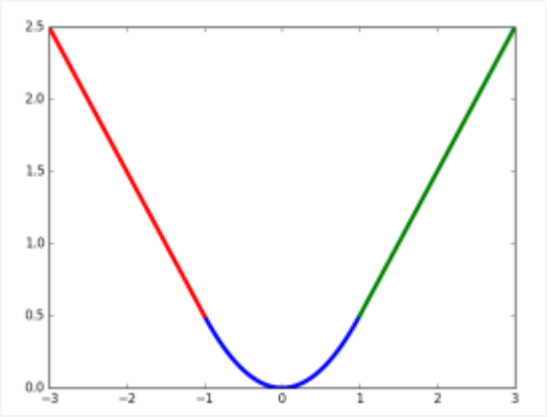
第一项是分类损失函数，第二项是定位损失函数，k是类别索引，R个ROI的损失值取平均，k*是实际类别

边框回归：

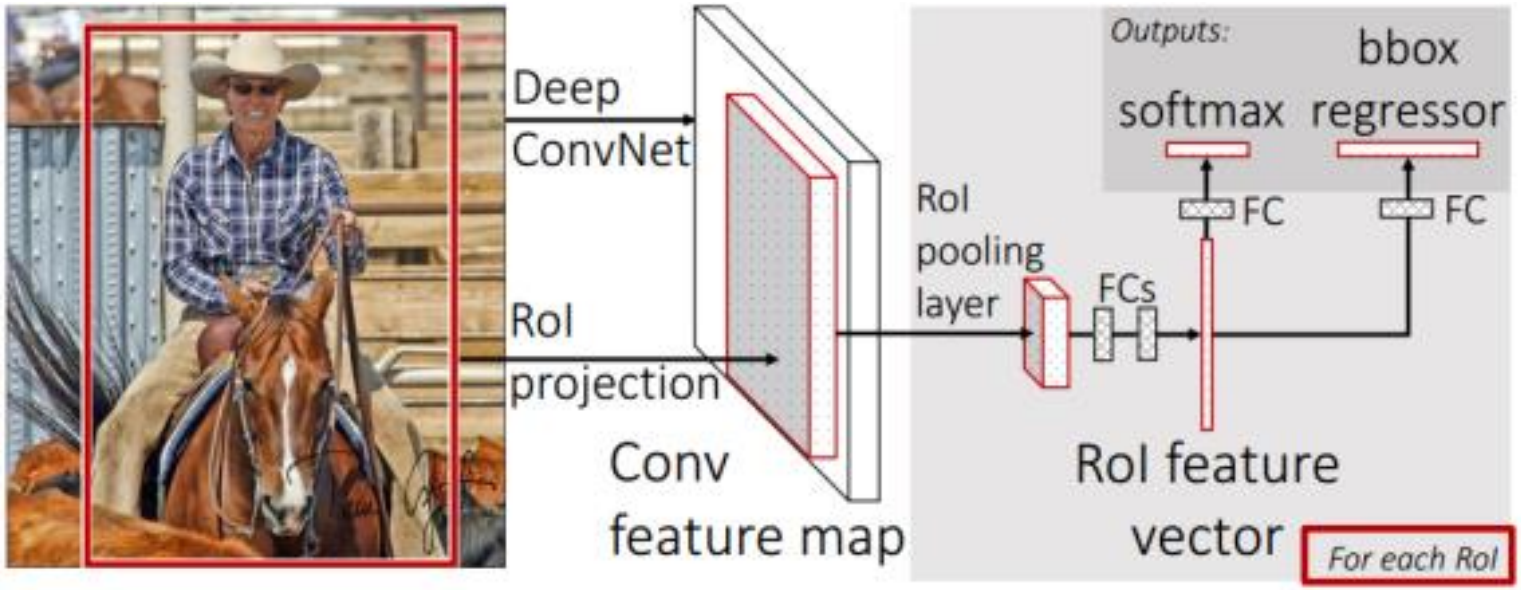
A smooth L1 loss which is less sensitive to outliers than L2 loss

$$L_{loc}(t, t^*) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t_i, t_i^*)$$

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

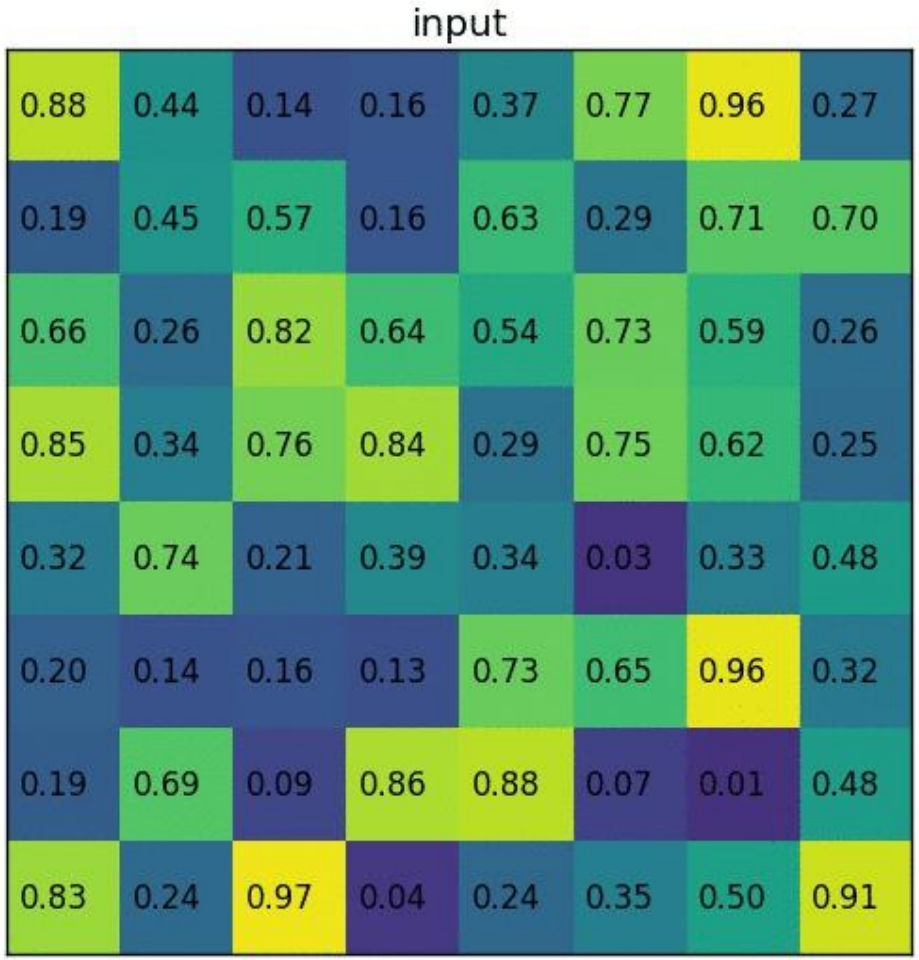


ROI pooling



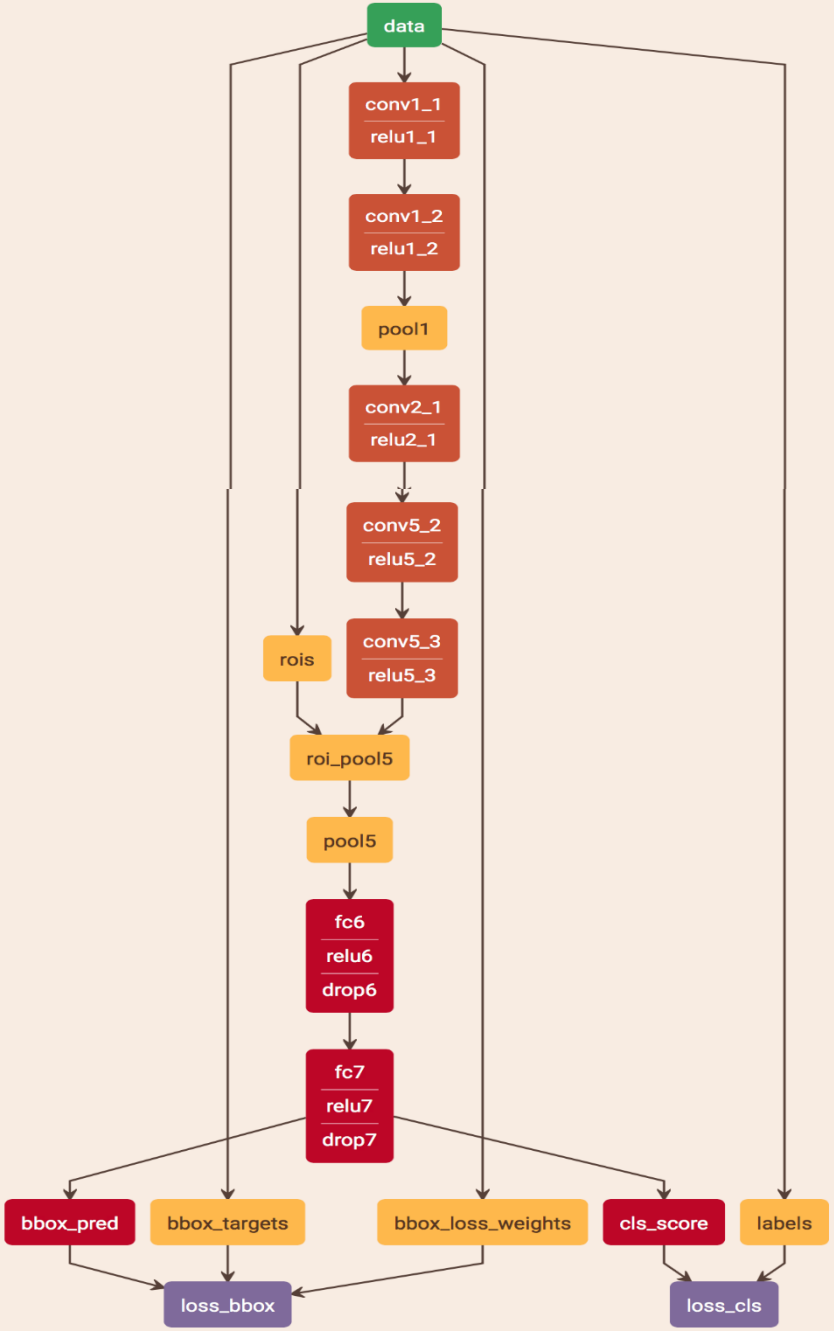
- 为什么需要ROI pooling?
  - FC的输入要求是固定尺寸
- ROI pooling是怎么计算的?

ROI pooling

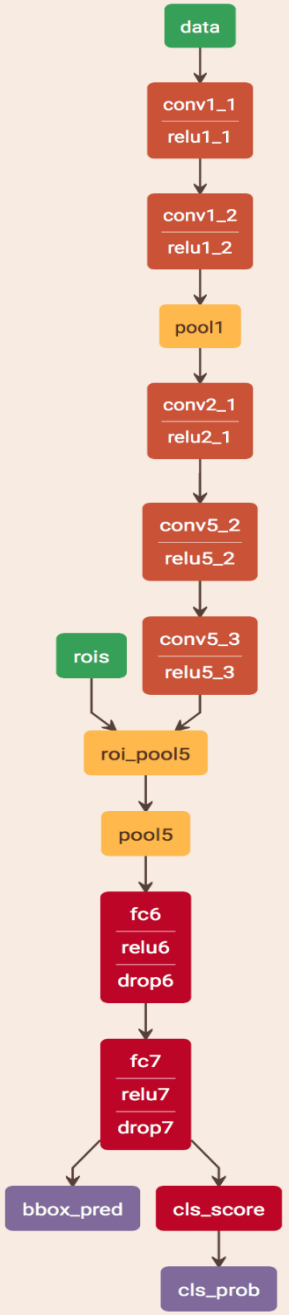


- **Input：任意尺寸的一个Region proposal；**
- **Output：固定尺寸2×2；**
- **如何实现？**
  1. **将输入的Region proposal划分为相等大小的部分（其数量与输出的维度相同）；**
  2. **找到每个部分的最大值；**
  3. **将这些最大值复制到输出(max pooling)；**

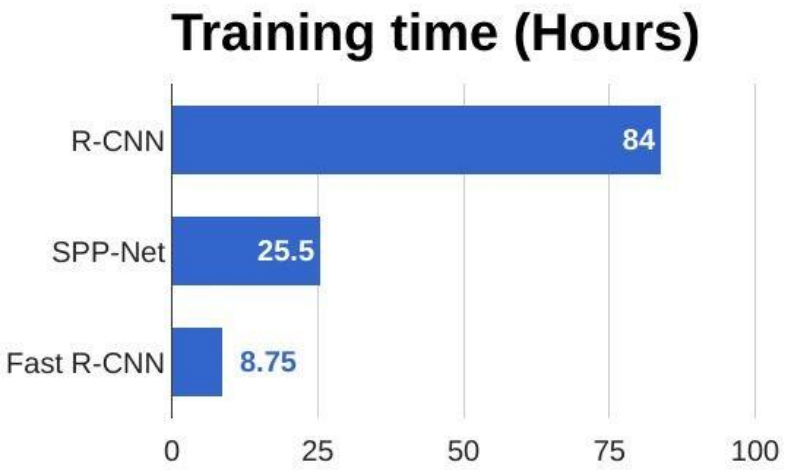
fast RCNN train

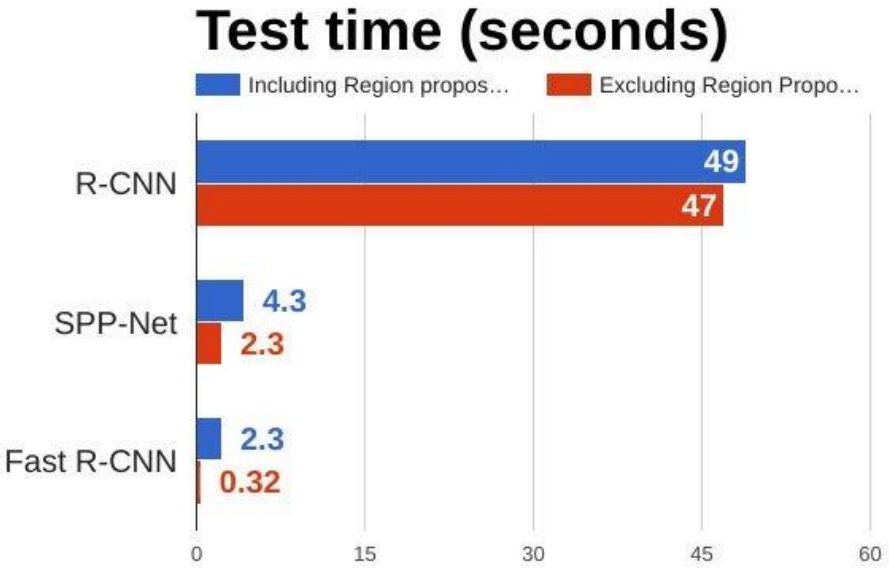


fast RCNN test



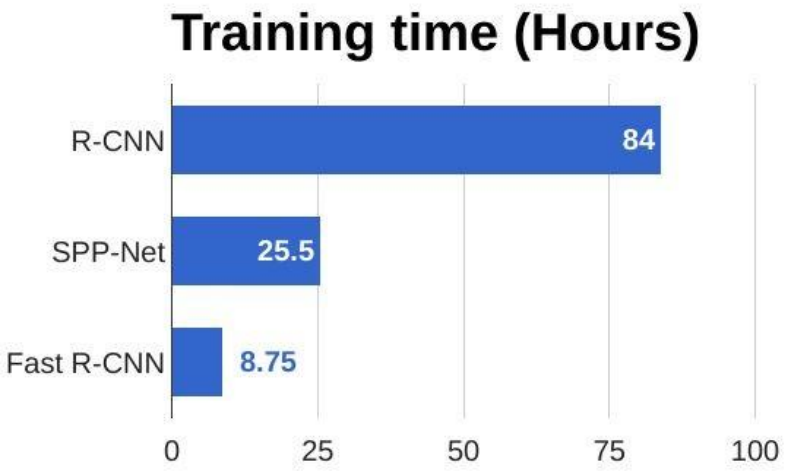
R-CNN vs Fast R-CNN

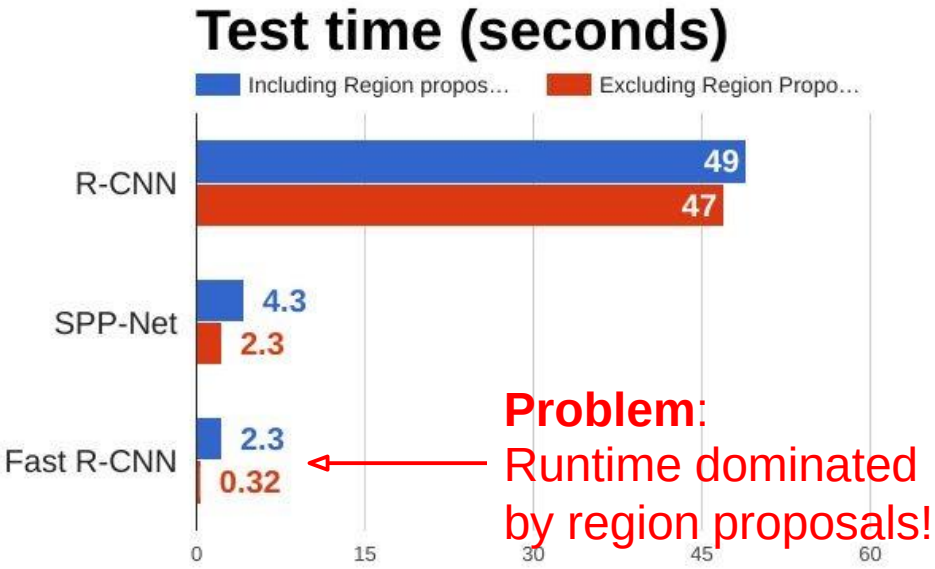




Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

R-CNN vs Fast R-CNN





Problem:
Runtime dominated by region proposals!

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014
Girshick, "Fast R-CNN", ICCV 2015

Fast R-CNN

- 改进之处：（与R-CNN相比）
  - 直接对输入的整张图像做卷积，不再对每个候选区域分别做卷积，从而减少大量重复计算；
  - 用ROI pooling对不同候选框的特征进行尺寸归一化；
  - 多任务损失函数：将边界框回归器放进网络一起训练，每个类别对应一个回归器；
  - 用softmax代替SVM分类器
- 缺点：
  - 候选区域提取仍使用selective search，目标检测时间大多消耗在selective search。

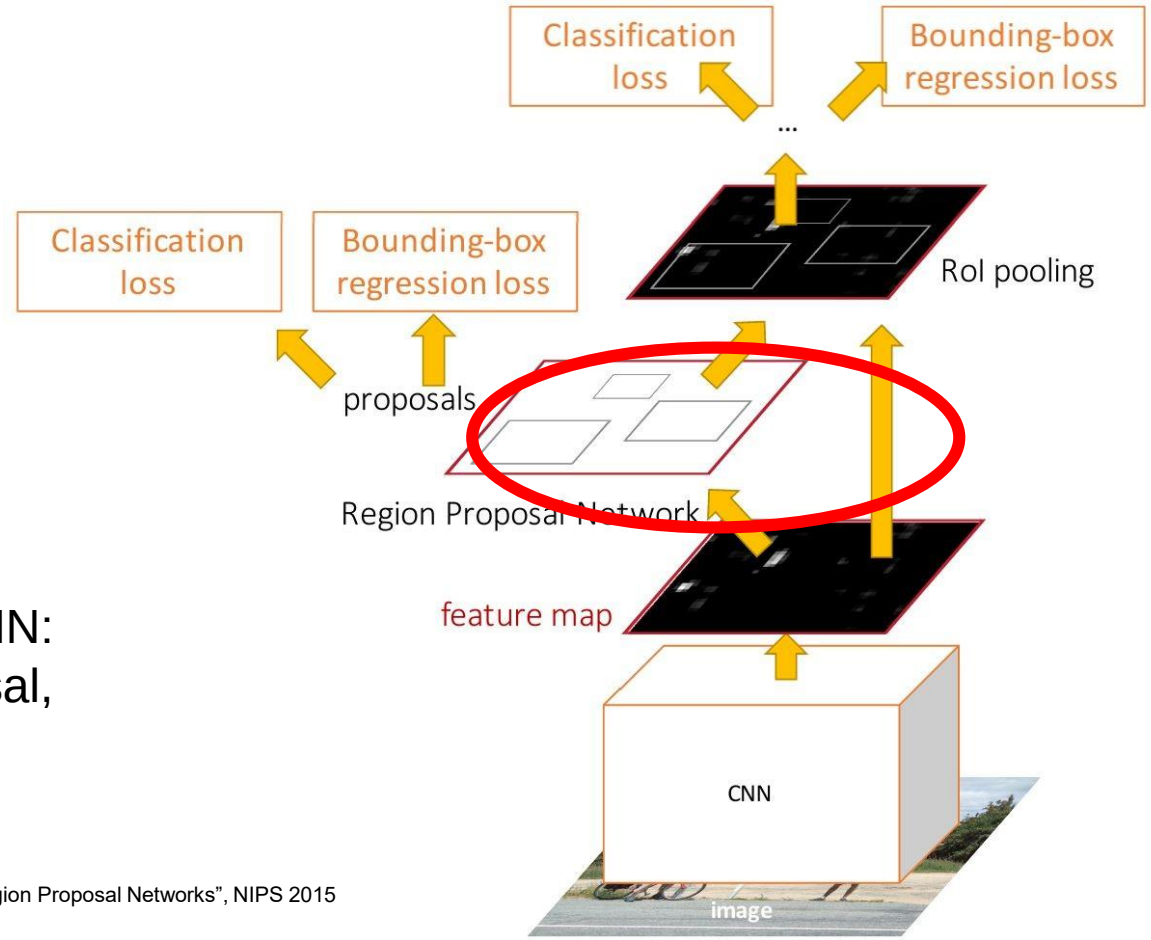
寻找更高效的候选区域生成方法……

Faster R-CNN:

Make CNN do proposals!

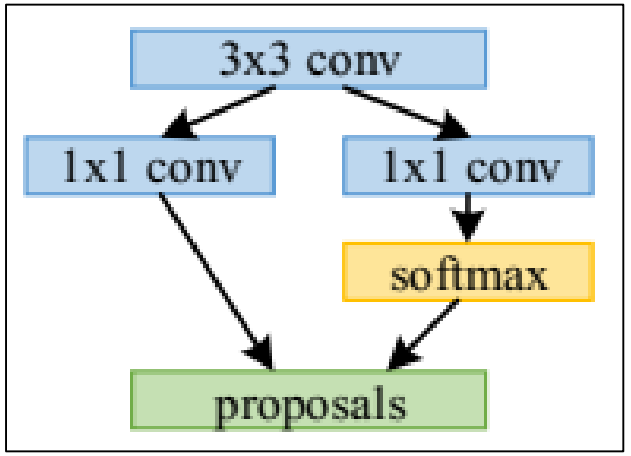
Insert **Region Proposal Network (RPN)** to predict proposals from features

Otherwise same as Fast R-CNN: Crop features for each proposal, classify each one



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

- RPN（Region Proposal Networks）：
- 生成不同尺度的候选区域anchor boxs，并用softmax判断候选框是否为前景；
- 选出前景候选框并利用bounding box regression调整候选框的位置；
- 用NMS（非最大值抑制）去除冗余候选框，最后输出候选区域。



Region Proposal Network

Input Image
(e.g. 3 x 640 x 480)

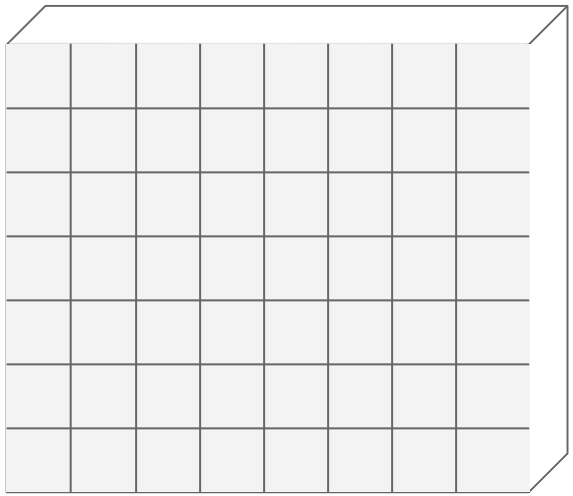


Image features
(e.g. 512 x 20 x 15)

Region Proposal Network

Imagine an **anchor box** of fixed size at each point in the feature map

Input Image
(e.g. 3 x 640 x 480)

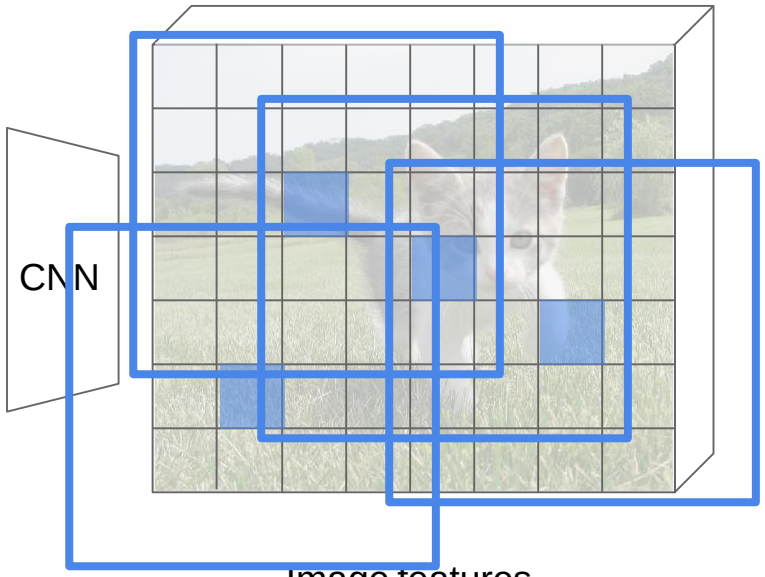


Image features
(e.g. 512 x 20 x 15)

Region Proposal Network

Imagine an **anchor box** of fixed size at each point in the feature map

Input Image
(e.g. 3 x 640 x 480)

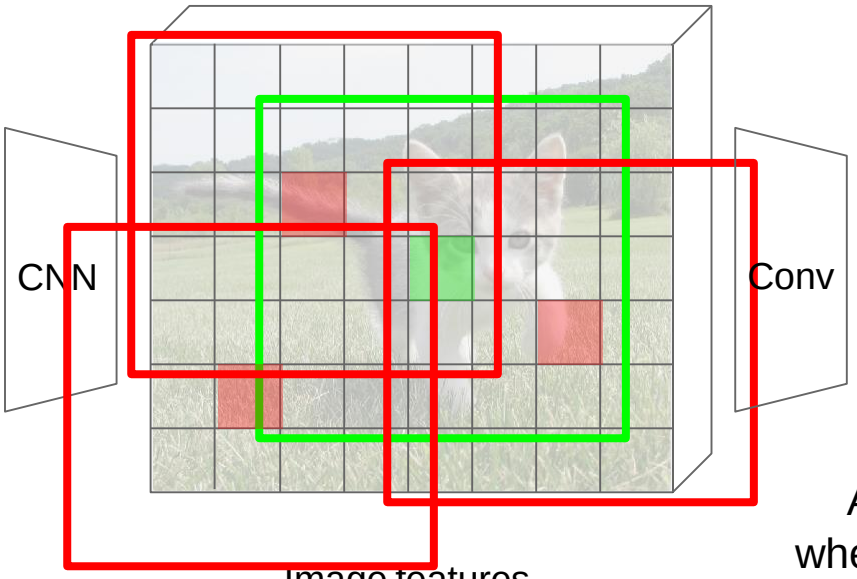


Image features
(e.g. 512 x 20 x 15)

At each point, predict whether the corresponding anchor contains an object (per-pixel logistic regression)

Region Proposal Network

Imagine an **anchor box** of fixed size at each point in the feature map

Input Image
(e.g. 3 x 640 x 480)

CNN

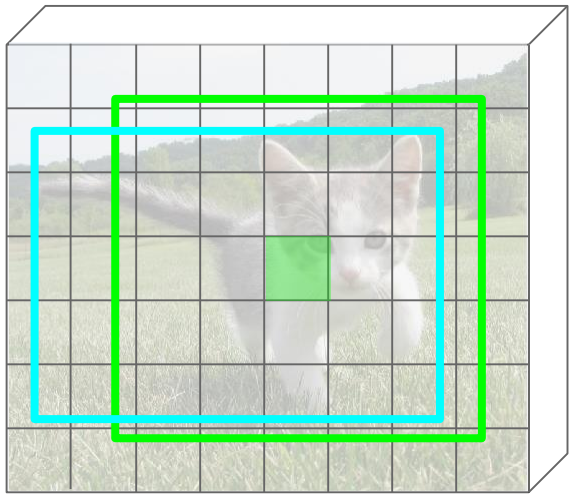


Image features
(e.g. 512 x 20 x 15)

Conv

Anchor is an object?
1 x 20 x 15

Box transforms
4 x 20 x 15

For positive boxes, also predict a transformation from the anchor to the ground-truth box (regress 4 numbers per pixel)

Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point

Input Image
(e.g. 3 x 640 x 480)

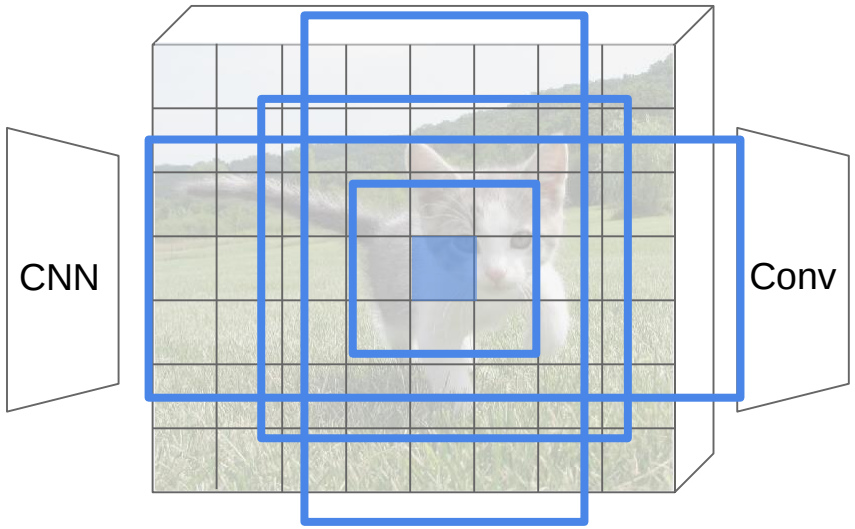


Image features
(e.g. 512 x 20 x 15)

Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point

Input Image
(e.g. 3 x 640 x 480)

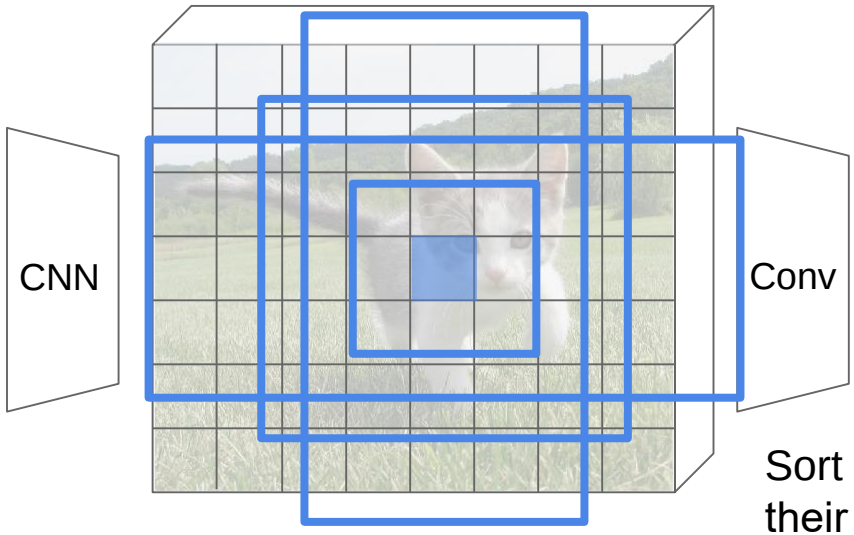
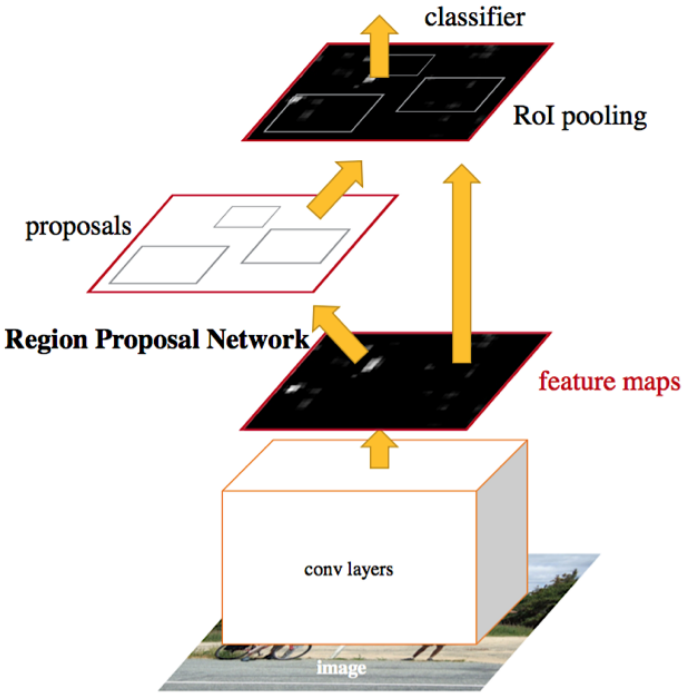


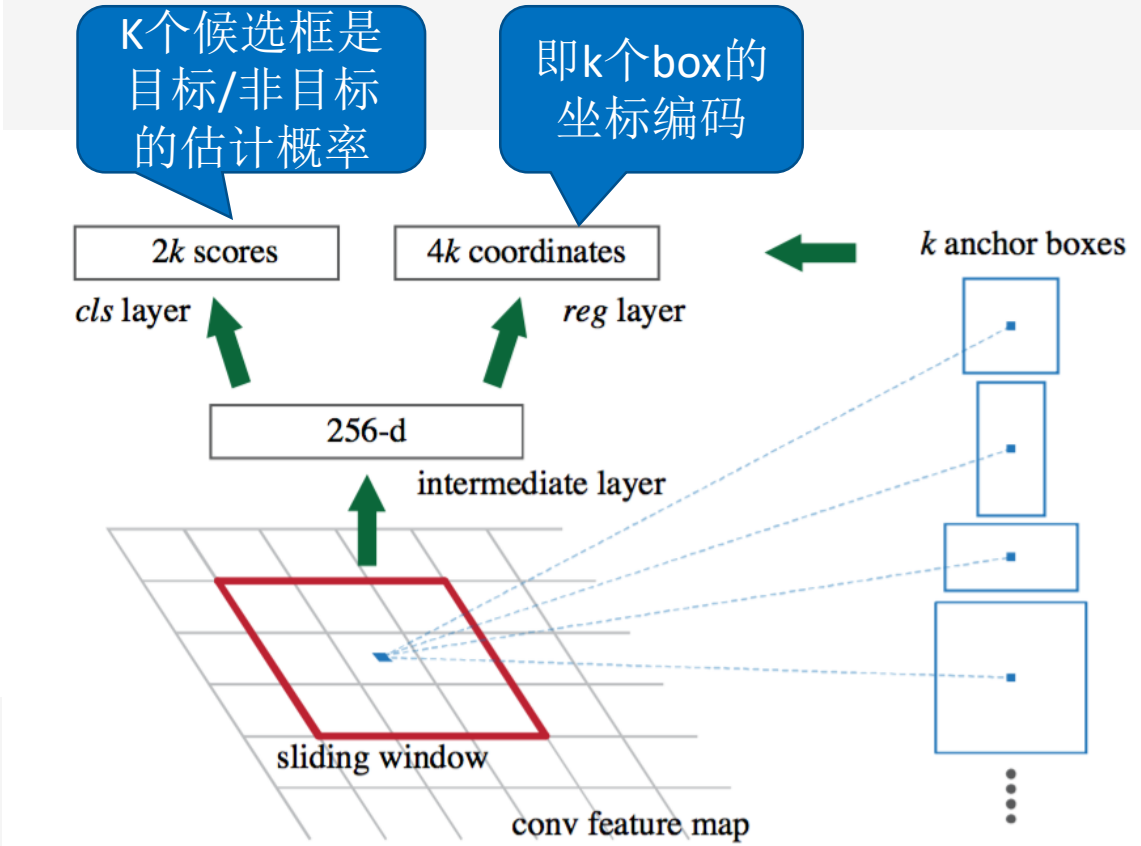
Image features
(e.g. 512 x 20 x 15)

Anchor is an object?
K x 20 x 15

Box transforms
4K x 20 x 15

Sort the K*20*15 boxes by their “object” score, take top ~300 as our proposals





anchor：每个anchor以当前滑动窗口中心为中心，并对应一种尺度和长宽比

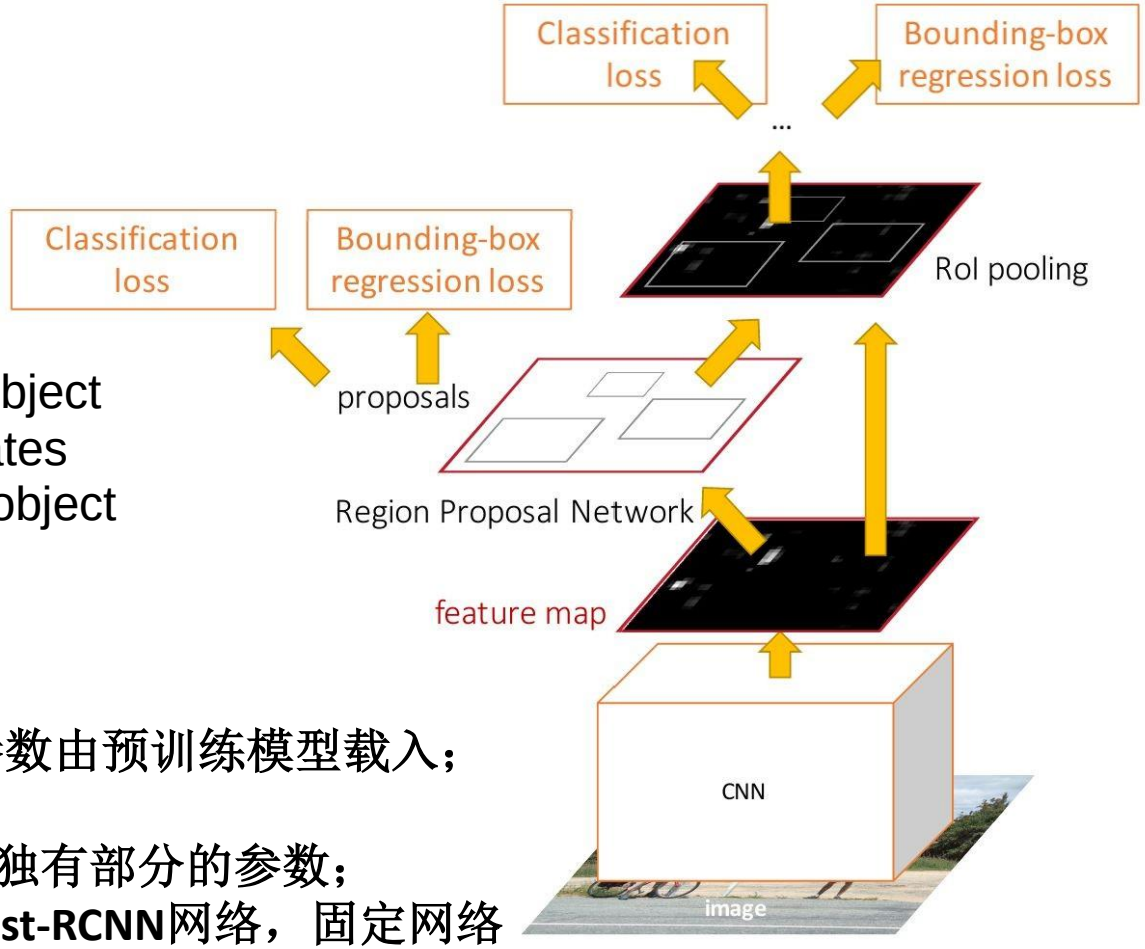
例如：3个尺度和三个长宽比对于每个位置就有k=9个anchor

128x128、256x256、512x512
1:2、1:1、2:1

平移不变性

Faster R-CNN:

Make CNN do proposals!



Jointly train with 4 losses:

1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

采用四步训练法：

1） 单独训练**RPN**网络，网络参数由预训练模型载入；

2） 单独训练**Fast-RCNN**网络；

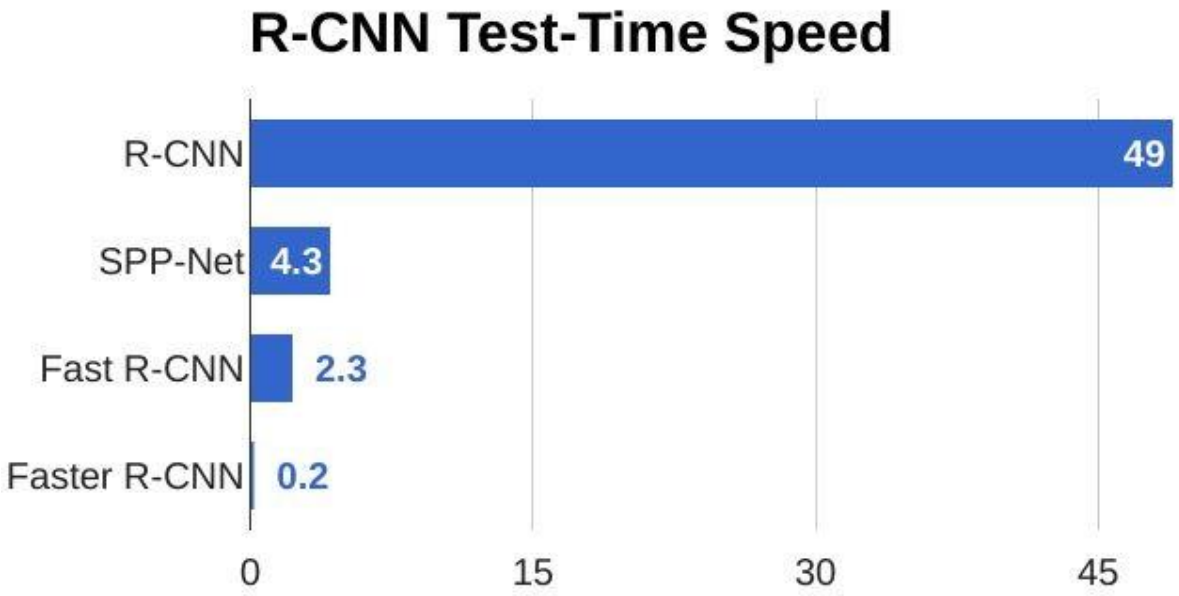
3） 再次训练**RPN**，只更新**RPN**独有部分的参数；

4） 利用**RPN**的结果再次微调**Fast-RCNN**网络，固定网络公共部分的参数，只更新**Fast-RCNN**独有部分的参数。

Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015

Faster R-CNN:

Make CNN do proposals!

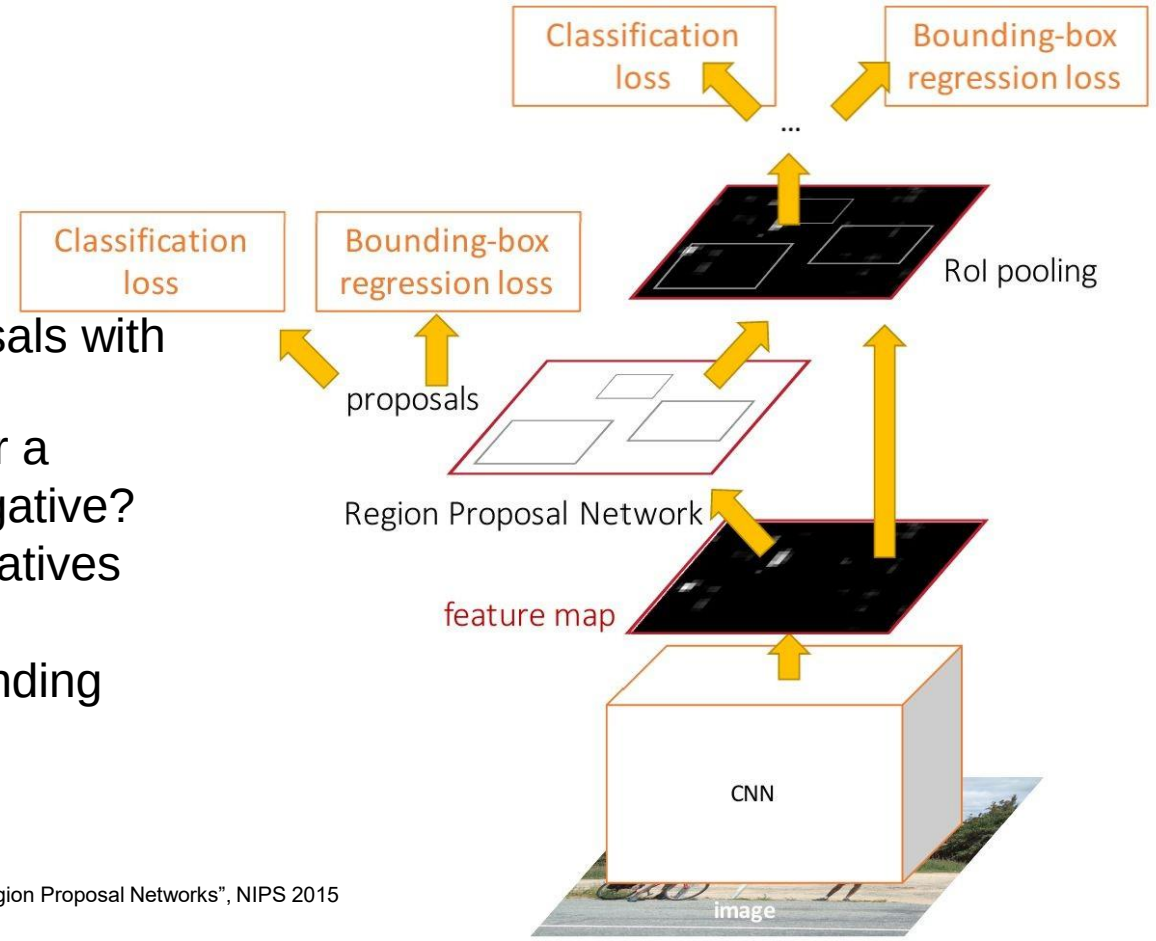


Faster R-CNN:

Make CNN do proposals!

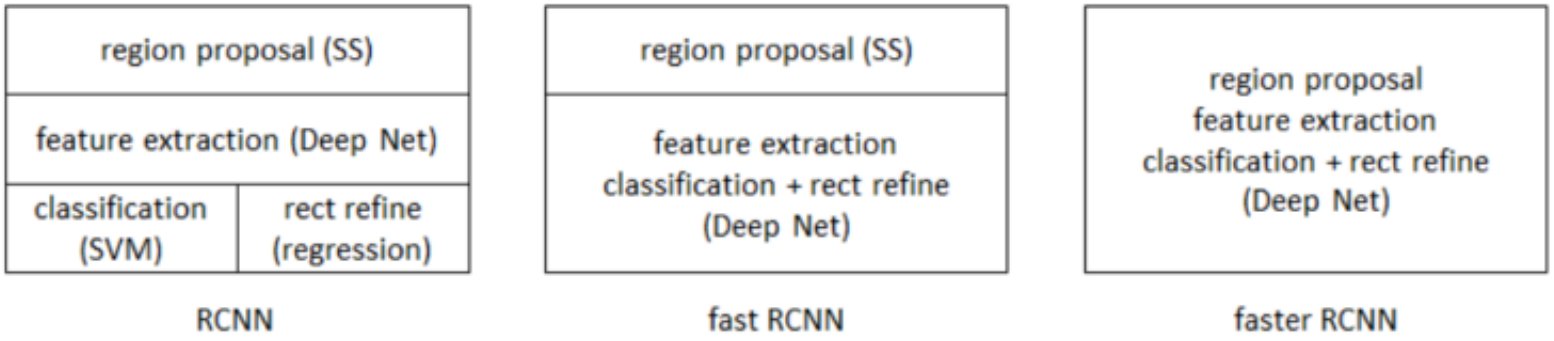
Glossing over many details:

- Ignore overlapping proposals with **non-max suppression**
- How to determine whether a proposal is positive or negative?
- How many positives / negatives to send to second stage?
- How to parameterize bounding box regression?



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

总结： R-CNN系列



网络	mAP (VOC 2012)	单帧检测时间
R-CNN	53.3%	50 s
Fast R-CNN	65.7%	2 s
Faster R-CNN	67.0%	0.2 s

从R-CNN到Fast R-CNN，再到Faster R-CNN，目标检测的四个基本步骤（候选区域生成，特征提取，分类，位置调整）终于被统一到一个深度网络框架之内，大大提高了运行速度。

拓展：R-CNN训练相关，Ross Girshick在ICCV15的演讲，Training R-CNNs of various velocities(Slow, fast, and faster)

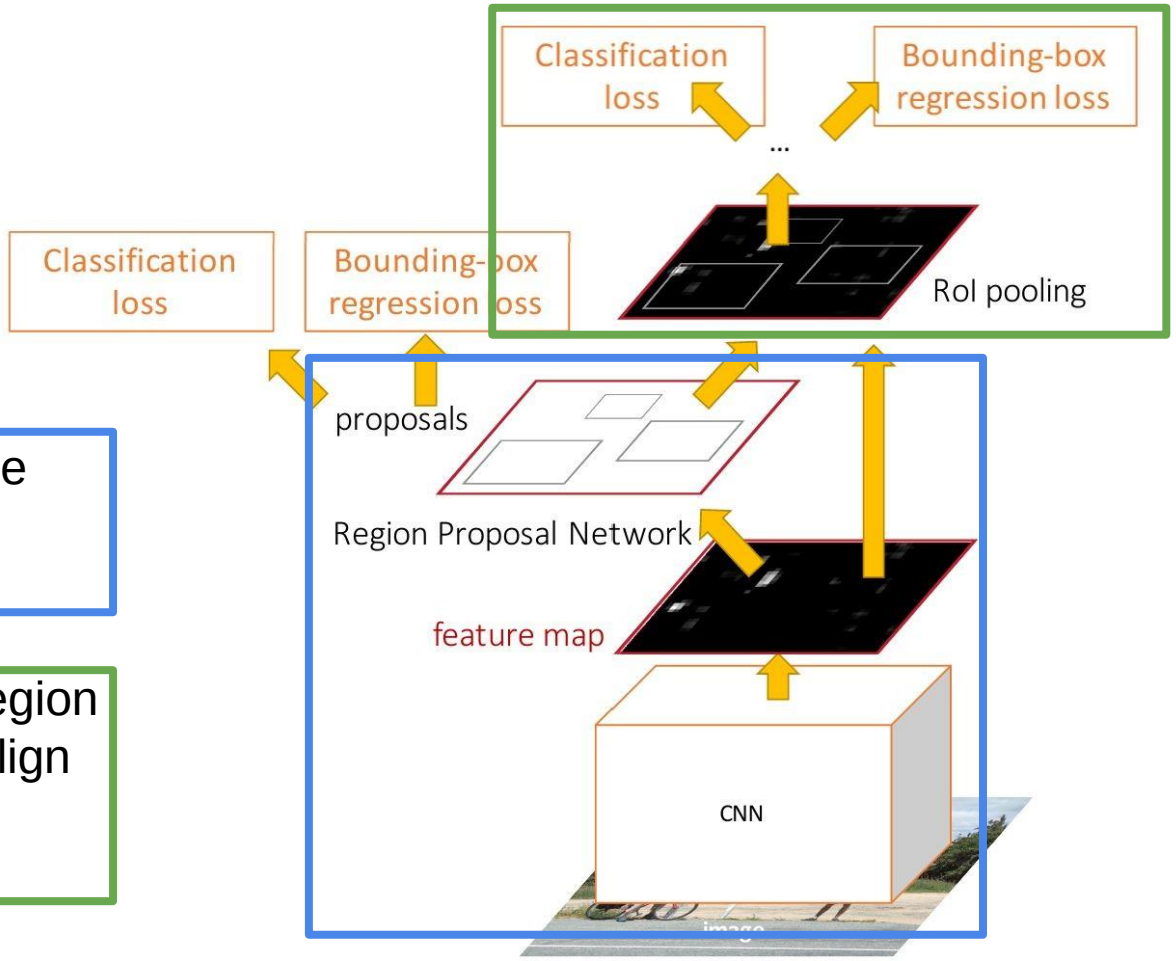
Faster R-CNN:

Make CNN do proposals!

Faster R-CNN is a
Two-stage object detector

First stage: Run once per image
- Backbone network
- Region proposal network

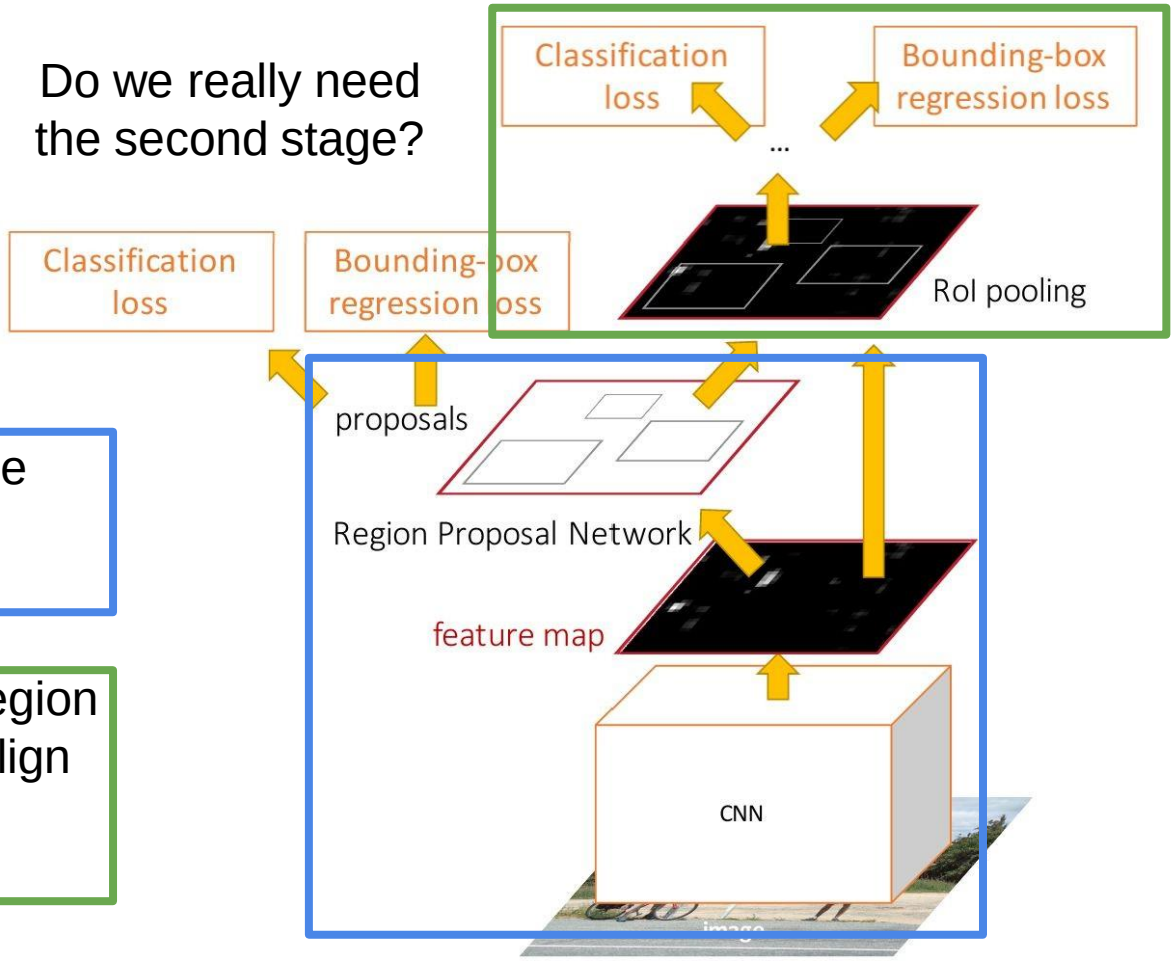
Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



Faster R-CNN:

Make CNN do proposals!

Do we really need the second stage?



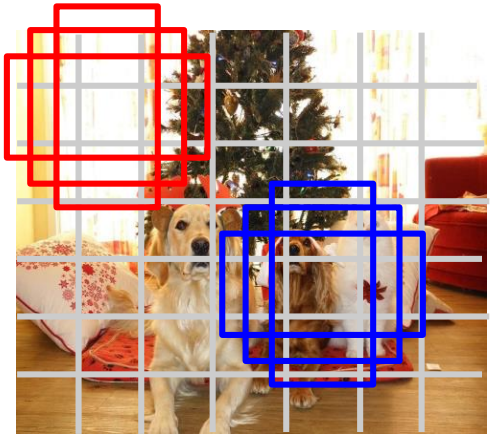
Faster R-CNN is a **Two-stage object detector**

First stage: Run once per image
- Backbone network
- Region proposal network

Second stage: Run once per region
- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset

Single-Stage Object Detectors: YOLO / SSD / RetinaNet

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

YOLO (You Only Look Once)

- 主要思想：将目标检测问题 转换为 直接从图像中提取bounding boxes和类别概率的回归问题
- YOLO算法开创了one-stage检测的先河，将目标分类和边界框定位合二为一，实现了端到端的目标检测。
- YOLO的运行速度非常快，达到45帧/秒，满足实时性需求。

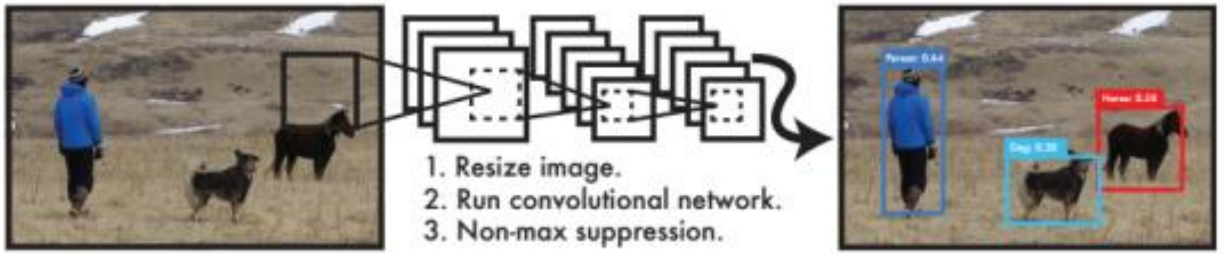
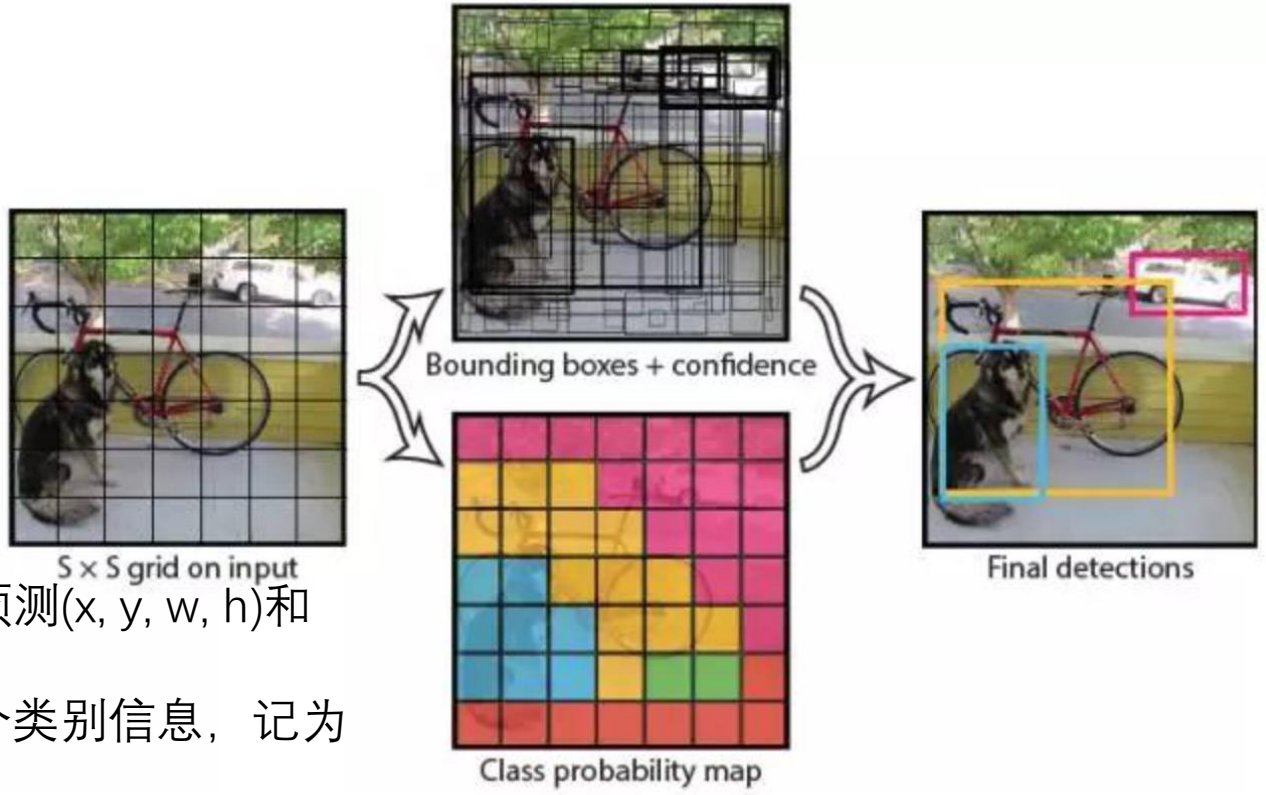


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

YOLO

YOLO的核心思想:

利用整张图作为网络的输入，直接在输出层回归bounding box的位置和bounding box所属的类别。



- 每个bounding box要预测(x, y, w, h)和confidence共5个值
- 每个网格还要预测一个类别信息，记为C类

(1) 给个一个输入图像，首先将图像划分成SxS的网格

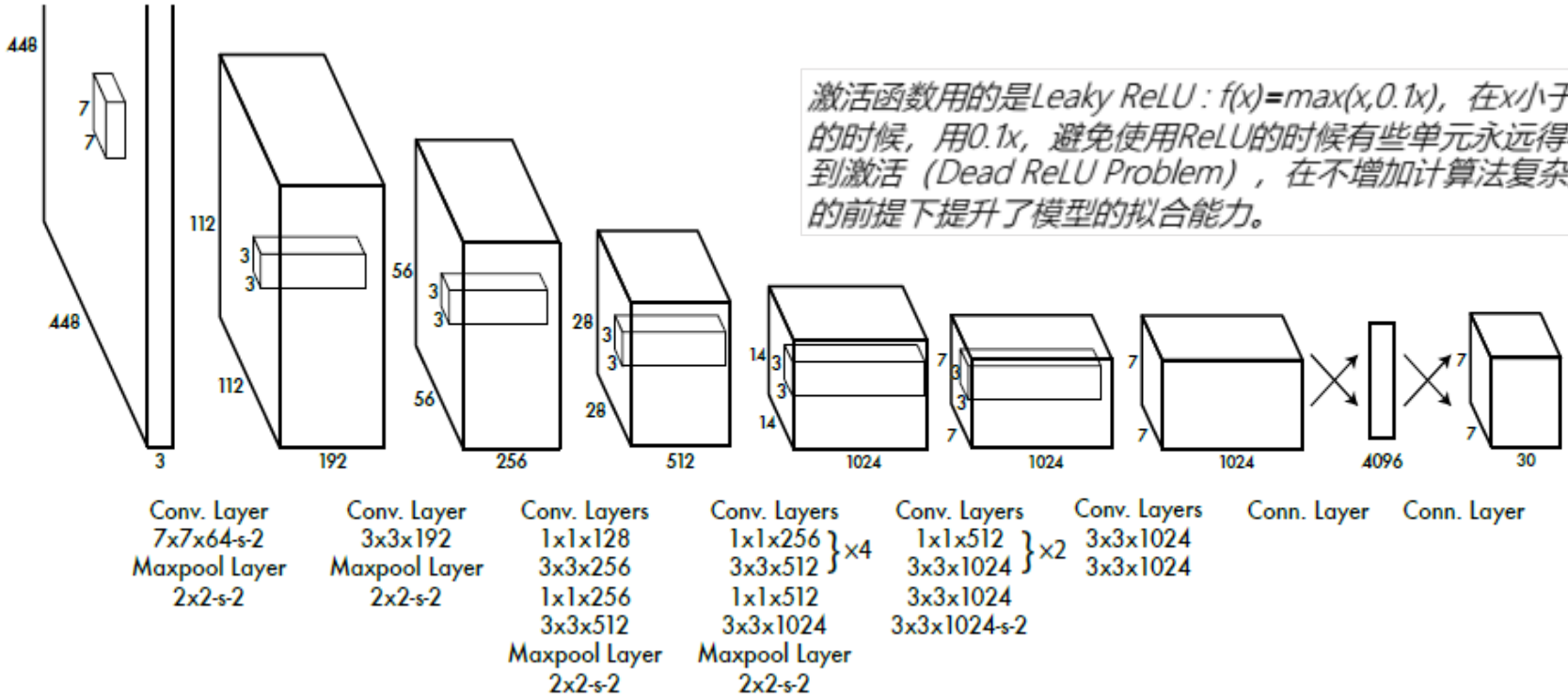
(2) 对于每个网格，我们都预测B个边框（每个边框的预测值包括：每个边框是目标的置信度以及每个边框区域在多个类别上的概率）

(3) 根据上一步可以预测出SxSxB个目标窗口，然后根据阈值去除可能性比较低的目标窗口，最后NMS去除冗余窗口即可。

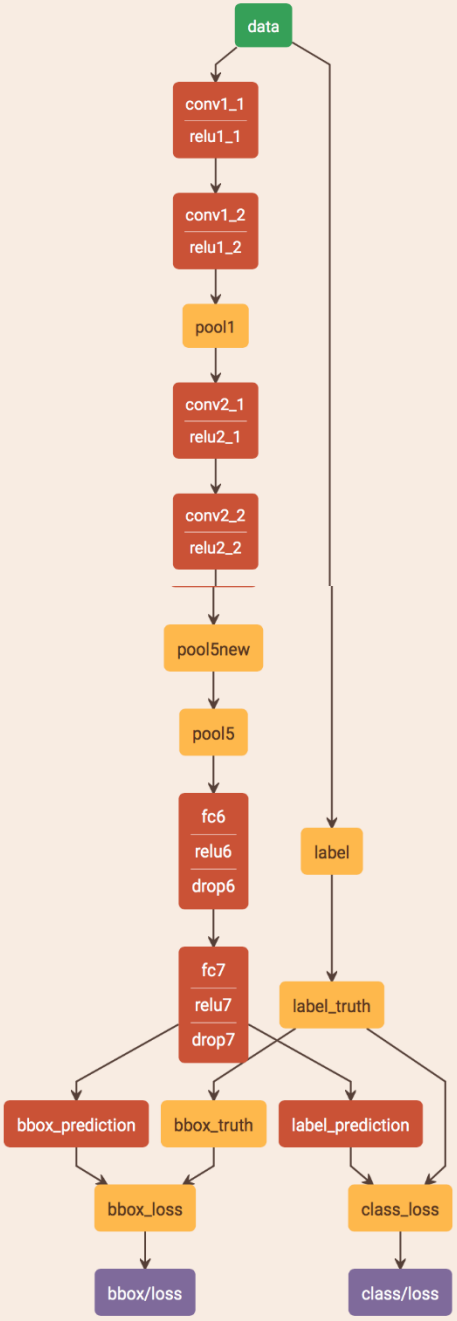
输出就是S x S x (5*B+C)的一个tensor

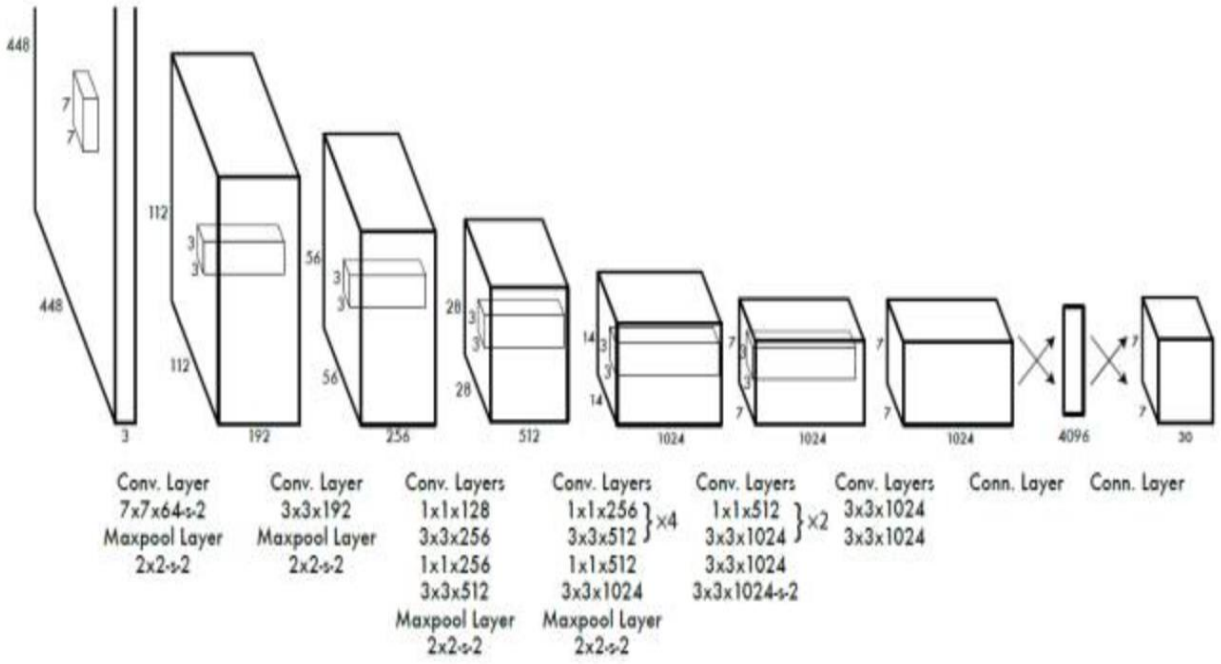
YOLO

- 在PASCAL VOC中，图像输入为448x448，取S=7，B=2，一共有20个类别(C=20)。则输出就是7x7x(5*2+20)=7x7x30的一个tensor
- 每个grid有30维，这30维中，8维是回归box的坐标，2维是box的confidence，还有20维是类别，其中坐标的x,y用对应网格的offset归一化到0-1之间，w,h用图像的width和height归一化到0-1之间。



VGG ILSVRC 16 layers YOLO





增强版本GPU中能跑45fps，简化版本155fps

YOLO可以每秒处理45张图像

每个网络预测目标窗口时使用的是全图信息

只使用7*7的网格回归会使得目标不能非常精准的定位，检测精度并不是很高

总结： YOLO

- YOLO对相互靠的很近的物体，还有很小的群体 检测效果不好，这是因为一个网格中只预测了两个框，并且只属于一类。
- 对测试图像中，同一类物体出现的新的不常见的长宽比和其他情况是。泛化能力偏弱。
- 由于损失函数的问题，定位误差是影响检测效果的主要原因。尤其是大小物体的处理上，还有待加强。

Object Detection: Lots of variables ...

Backbone Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception ResNet
MobileNet

"Meta-Architecture"
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

Image Size
 # Region Proposals
…

Takeaways
Faster R-CNN is slower but more accurate

SSD is much faster but not as accurate

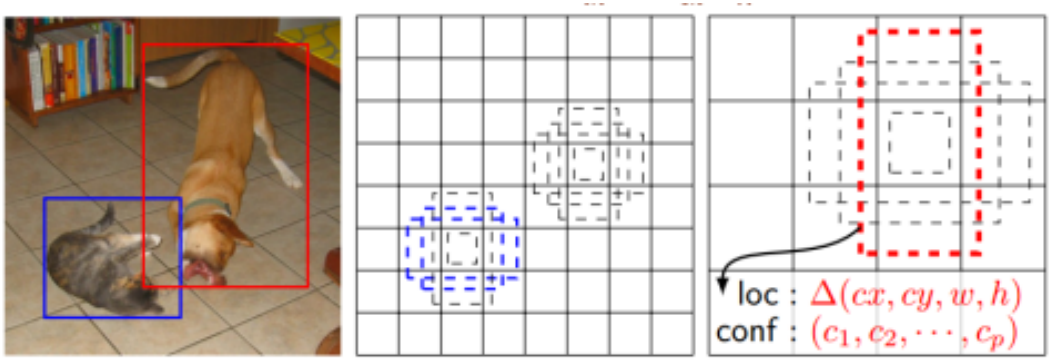
Bigger / Deeper backbones work better

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017

R-FCN: Dai et al, "R-FCN: Object Detection via Region-based Fully Convolutional Networks", NIPS 2016
Inception-V2: Ioffe and Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015
Inception V3: Szegedy et al, "Rethinking the Inception Architecture for Computer Vision", arXiv 2016
Inception ResNet: Szegedy et al, "Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning", arXiv 2016
MobileNet: Howard et al, "Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017

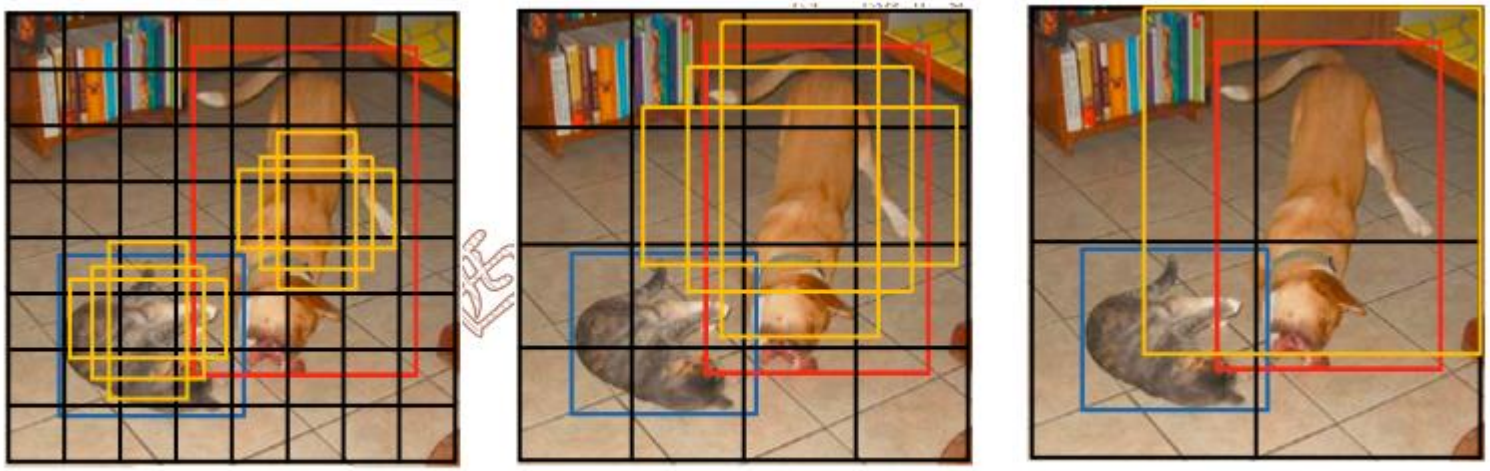
SSD（单发多框检测器）

- SSD，全称Single Shot MultiBox Detector，是Wei Liu在ECCV 2016上提出的一种多尺度的目标检测算法。
- 相比Faster-RCNN有明显的速度优势，相比YOLO又有明显的mAP优势。
- SSD具有如下主要特点：
  1. 从YOLO中继承了将detection转化为regression的思路，同时一次即可完成网络训练
  2. 基于Faster-RCNN中的anchor，提出了相似的prior box；
  3. 加入基于特征金字塔（Pyramidal Feature Hierarchy）的检测方式，相当于半个FPN思路



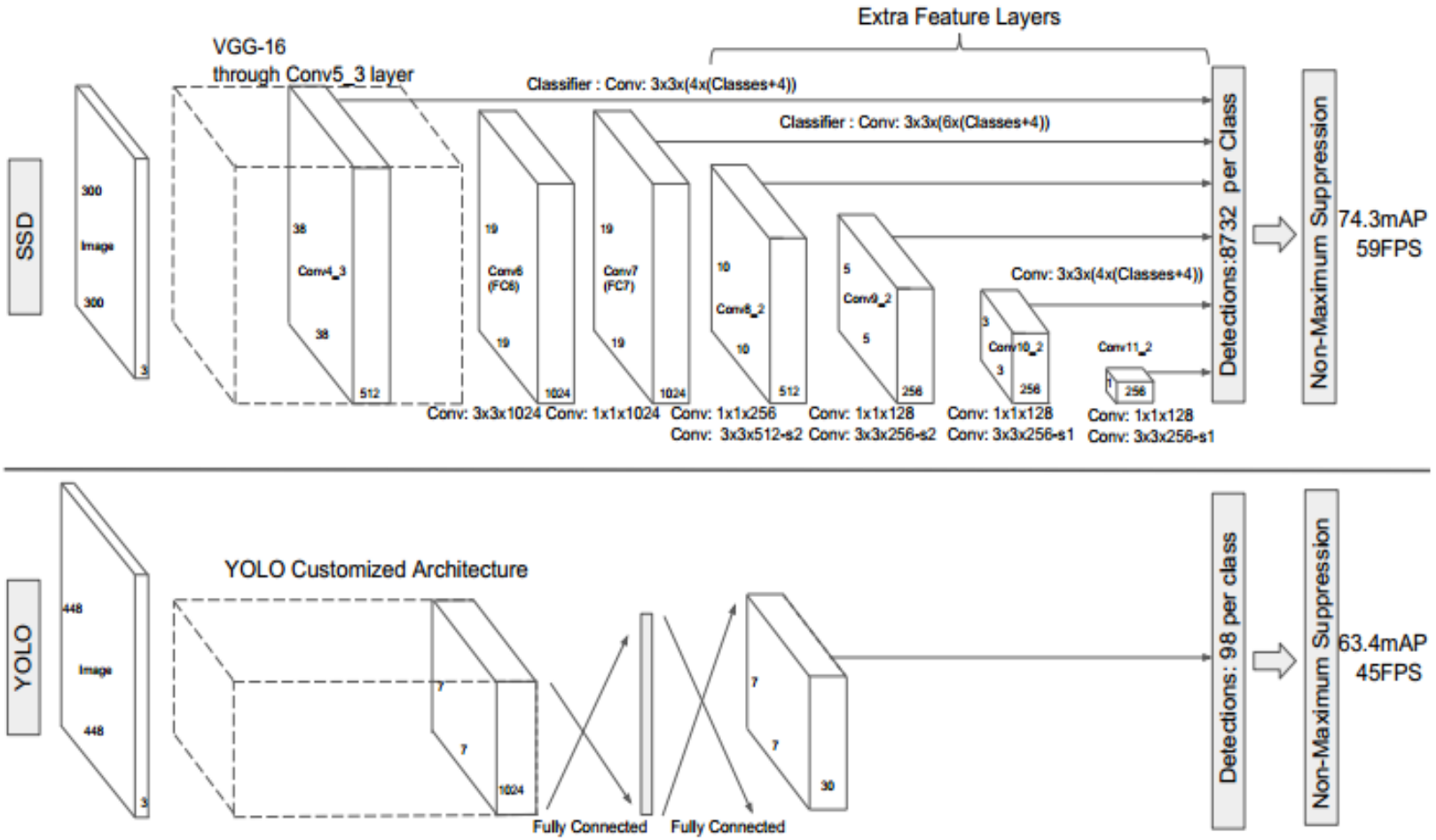
(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

- 比较大的特征图可以用来检测相对较小的目标，而小的特征图负责检测大目标，例如左图中8x8的特征图可以划分成更多单元，其每个单元的先验框尺度较小，适合用于检测较小的目标。



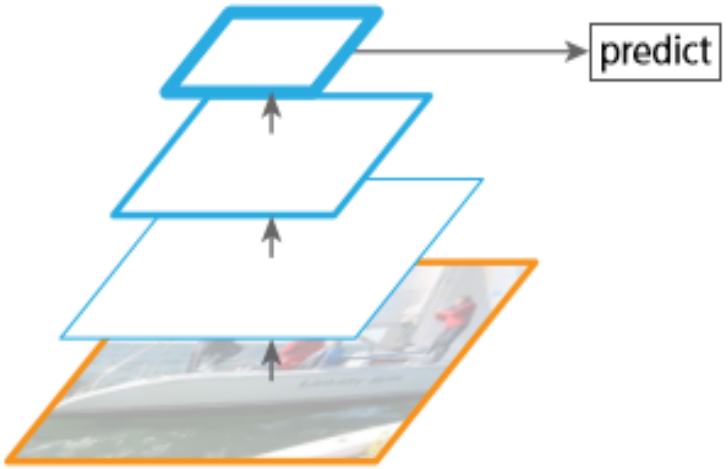
较低层级的特征图　　中间层级的特征图　　较高层级的特征图

SSD网络结构 V.S. YOLO网络结构

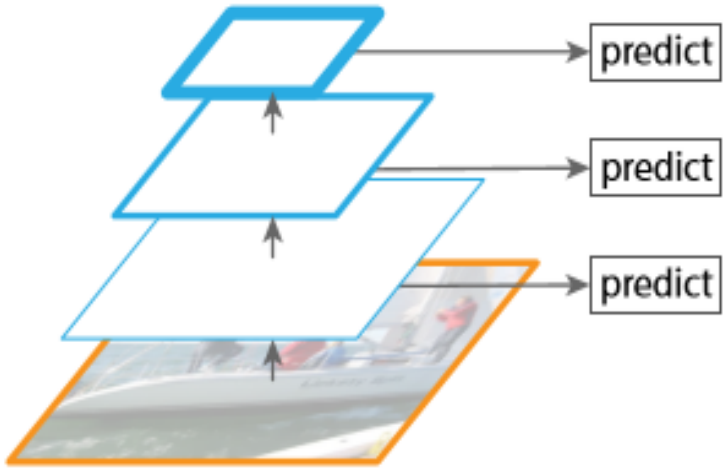


SSD

- YOLO在卷积层后接全连接层，即检测时只利用了最高层feature maps（包括Faster RCNN也是如此）；
- 而SSD采用了特征金字塔结构进行检测，即检测时利用了conv4-3，conv-7（FC7），conv6-2，conv7-2，conv8_2，conv9_2这些大小不同的feature maps，在多个feature maps上同时进行softmax分类和位置回归。



Single feature map



Pyramidal feature hierarchy

总结：SSD

- SSD算法的优点很明显：运行速度可以和YOLO媲美，检测精度可以和Faster RCNN媲美。
- 缺点：
  - 需要人工设置prior box的min_size，max_size和aspect_ratio值。网络中prior box的基础大小和形状不能直接通过学习获得，而是需要手工设置。而网络中每一层feature使用的prior box大小和形状恰好都不一样，导致调试过程非常依赖经验。
  - 虽然采用了pyramdial feature hierarchy的思路，但是对小目标的recall依然一般，并没有达到碾压Faster RCNN的级别。

Object Detection: Lots of variables ...

Backbone Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception ResNet
MobileNet

"Meta-Architecture"
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

Image Size
 # Region Proposals
…

Takeaways
Faster R-CNN is slower but more accurate

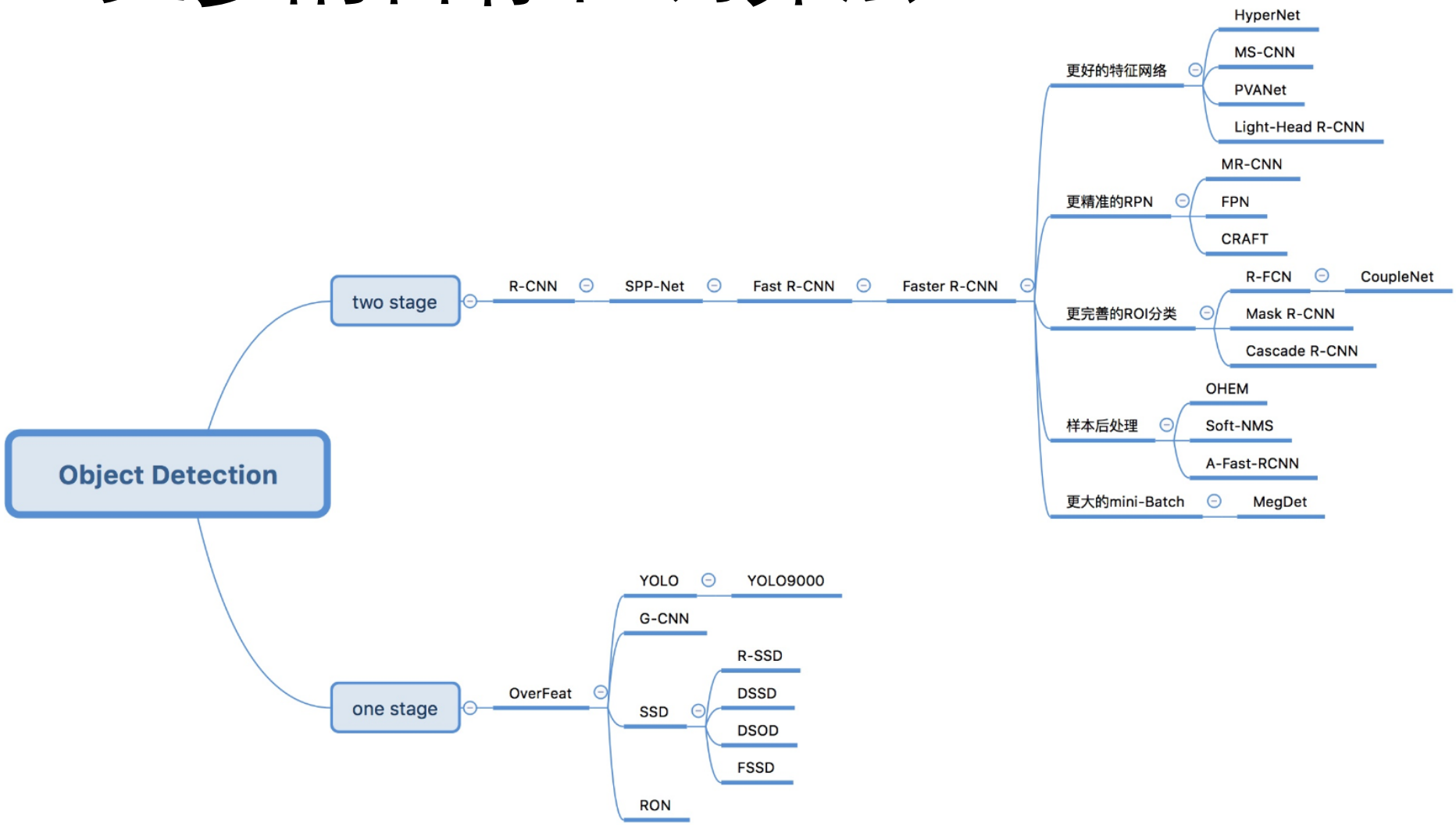
SSD is much faster but not as accurate

Bigger / Deeper backbones work better

Huang et al, "Speed/accuracy trade-offs for modern convolutional object detectors", CVPR 2017
Zou et al, "Object Detection in 20 Years: A Survey", arXiv 2019 (today!)

R-FCN: Dai et al, "R-FCN: Object Detection via Region-based Fully Convolutional Networks", NIPS 2016
Inception-V2: Ioffe and Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML 2015
Inception V3: Szegedy et al, "Rethinking the Inception Architecture for Computer Vision", arXiv 2016
Inception ResNet: Szegedy et al, "Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning", arXiv 2016
MobileNet: Howard et al, "Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017

更多的目标检测算法……



https://zhuanlan.zhihu.com/p/33277354

谢谢!