深度学习平台与应用

第六讲：卷积神经网络架构

范琦

fanqi@nju.edu.cn

2025年10月11日

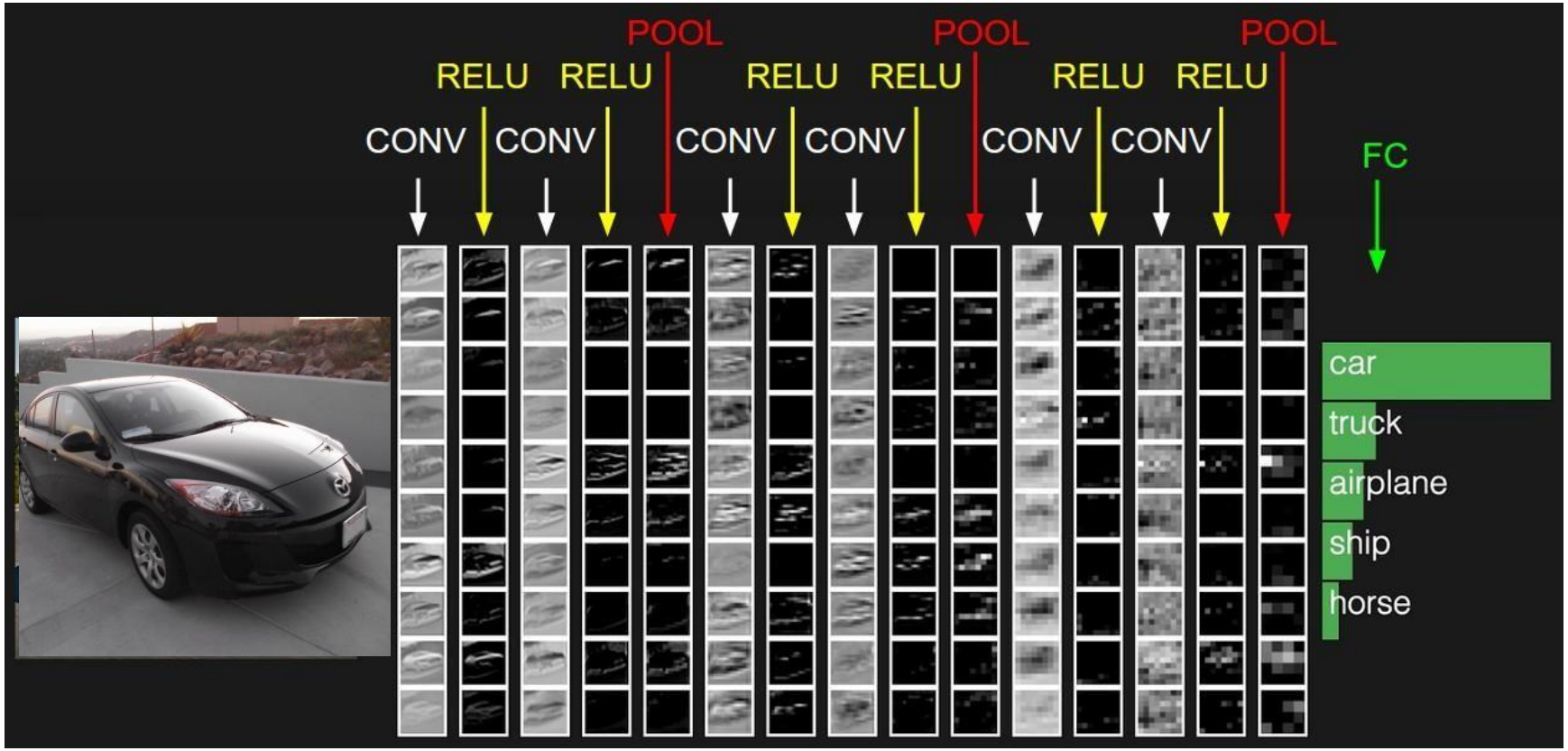
大　　纲

- 归一化层
- 经典卷积网络结构

卷积神经网络架构

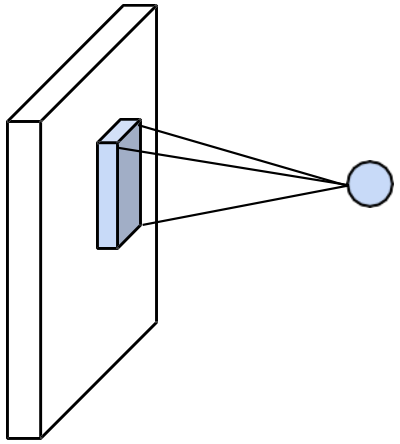
南京大学 NANJING UNIVERSITY

卷积神经网络结构

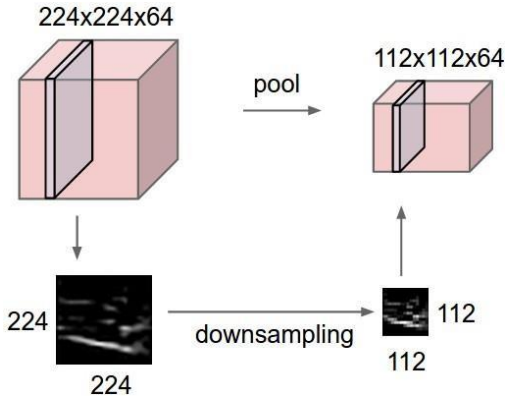


卷积神经网络组成部分

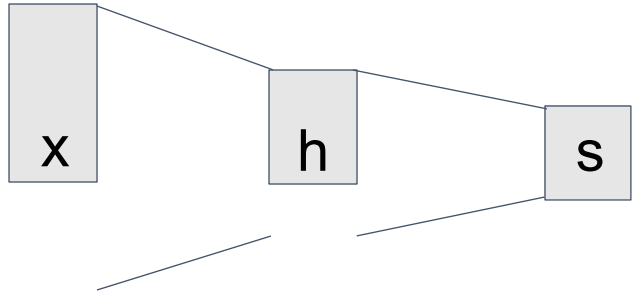
卷积层



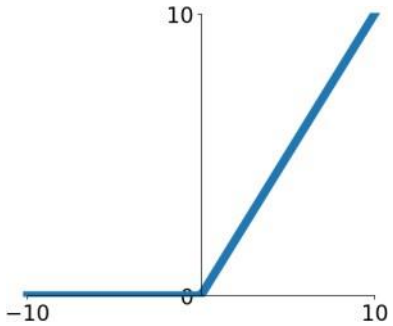
池化层



全连接层

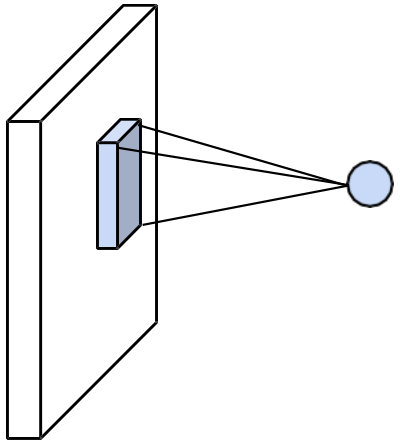


激活函数

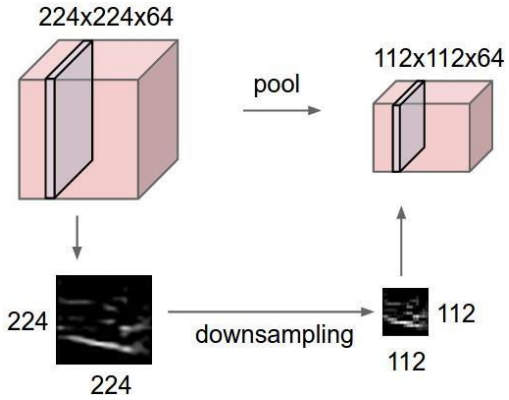


卷积神经网络组成部分

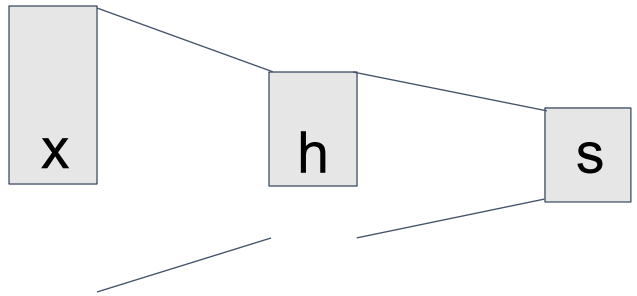
卷积层



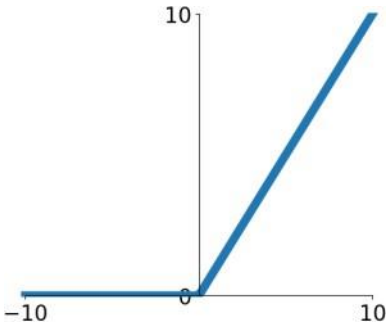
池化层



全连接层



激活函数



批归一化层

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

Batch Normalization（批归一化层）

南京大学 NANJING UNIVERSITY

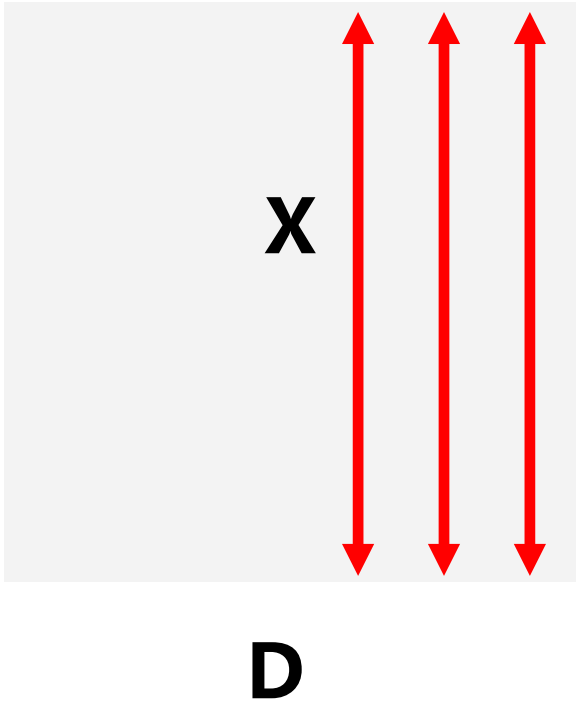
- 以下输入会导致网络难以优化：
  - 输入不以零为中心（有较大的偏置）
  - 输入的每个元素具有不同的缩放比例

Batch Normalization（批归一化层）

- 以下输入会导致网络难以优化：
  - 输入不以零为中心（有较大的偏置）
  - 输入的每个元素具有不同的缩放比例
- 解决方案：
  - 对输入进行缩放（归一化）

Batch Normalization（批归一化层）

输入：$x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

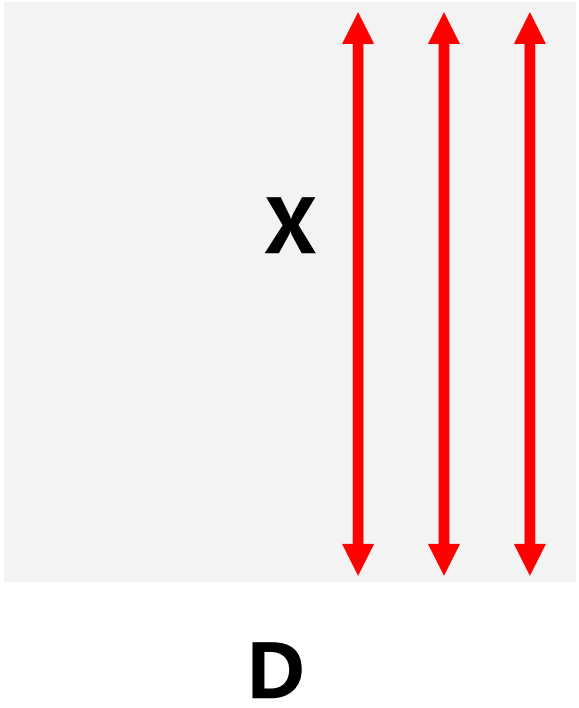
逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

Batch Normalization（批归一化层）

输入：$x : N \times D$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

问题：归一化后的输入丢失了大量数据信息（均值、方差）

Batch Normalization（批归一化层）

南京大学 NANJING UNIVERSITY

输入： $x : N \times D$

可学习的缩放和偏移参数：

$$\gamma, \beta : D$$

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

缩放后的输出，大小为N x D

Batch Normalization（批归一化层）

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

缩放后的输出，大小为N x D

训练：从minibatch中估计

Batch Normalization（批归一化层）

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

缩放后的输出，大小为N x D

训练：从minibatch中估计

测试：使用从训练集中估计的参数（移动平均）

Batch Normalization（批归一化层）

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

缩放后的输出，大小为N x D

训练：从minibatch中估计

测试：使用从训练集中估计的参数（移动平均）

训练：从数据集中学习

Batch Normalization（批归一化层）

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

缩放后的输出，大小为N x D

训练：从minibatch中估计

测试：使用从训练集中估计的参数（移动平均）

训练：从数据集中学习

测试：使用训练得到的参数

Batch Normalization（批归一化层）

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

逐通道均值，大小为D

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(x_{i,j} - \mu_j)^2$$

逐通道方差，大小为D

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

归一化后的输入，大小为N x D

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

缩放后的输出，大小为N x D

训练：从minibatch中估计

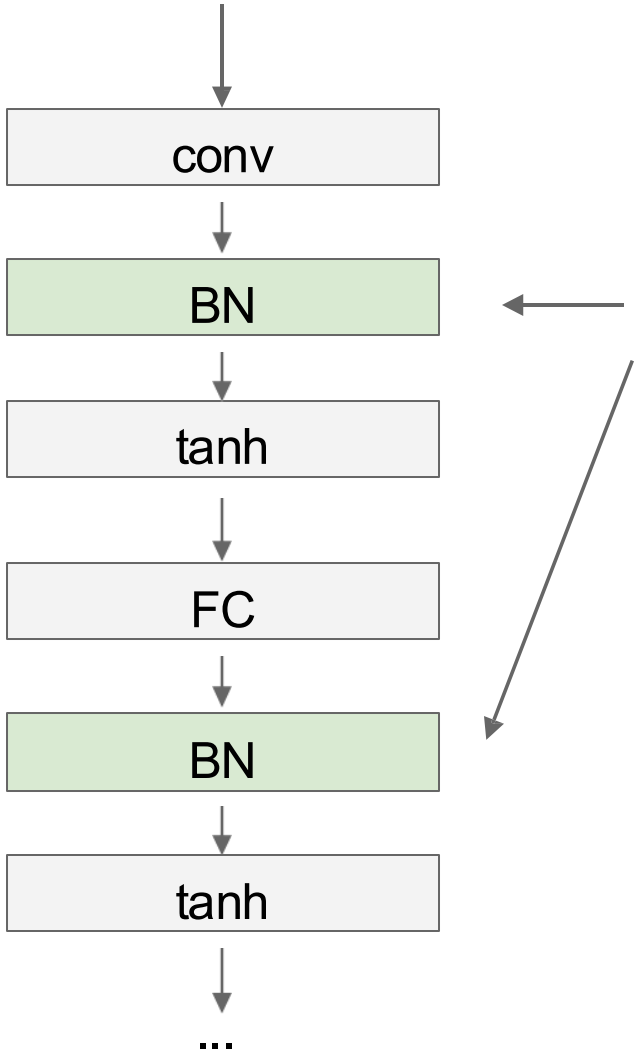
测试：使用从训练集中估计的参数（移动平均）

在测试时，batchnorm 变成了一个线性运算符，可以与之前的FC或conv层融合

训练：从数据集中学习

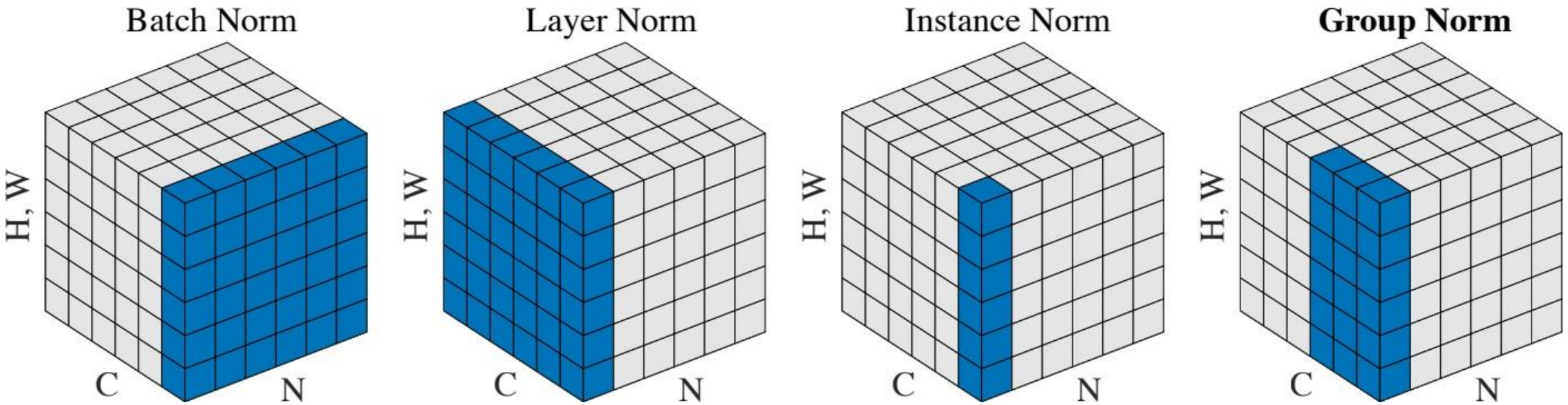
测试：使用训练得到的参数

Batch Normalization（批归一化层）



- 使深度网络更容易训练！
- 改善梯度流
- 允许更高的学习率，更快的收敛
- 网络对初始化变得更加稳健
- 在训练过程中起到正则化的作用
- 注意：训练和测试是不同的！

不同的归一化层


Batch Norm
H, W
C
N
Layer Norm
H, W
C
N
Instance Norm
H, W
C
N
Group Norm
H, W
C
N

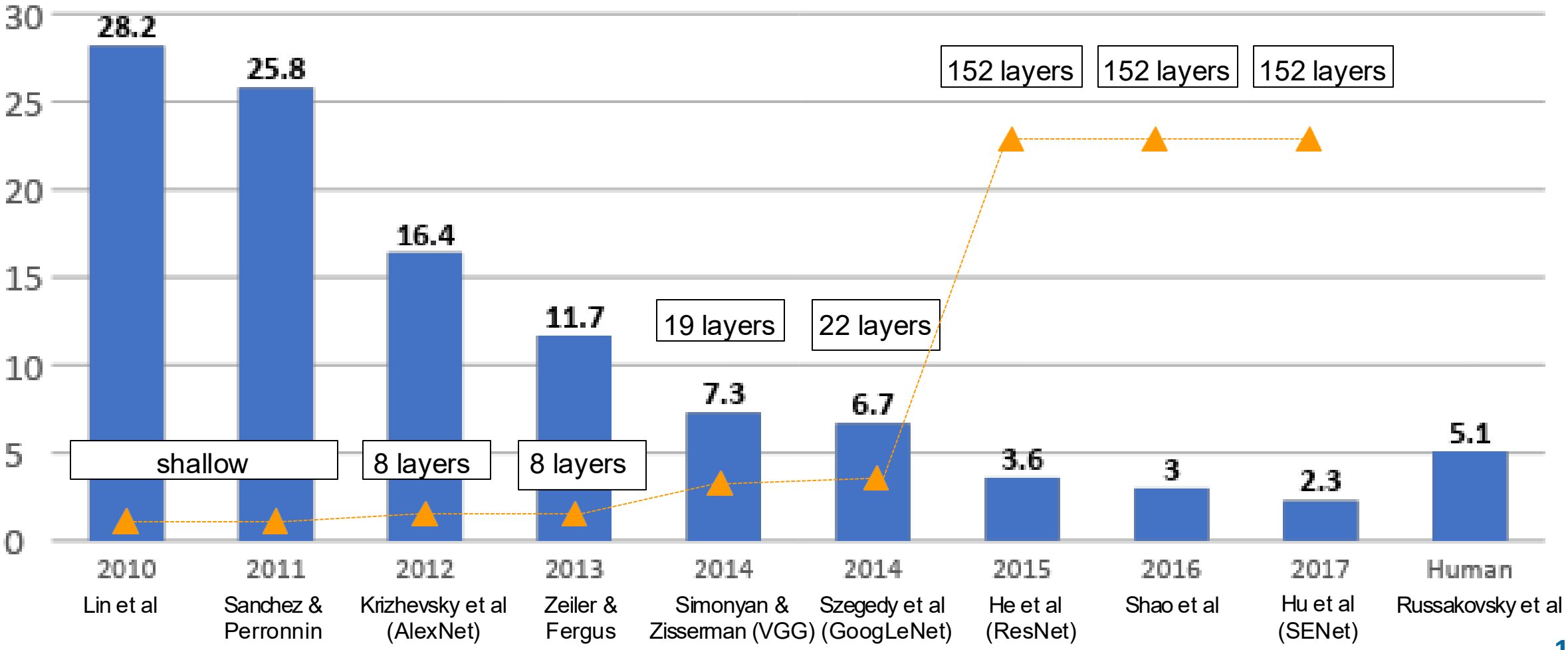
大　　纲

- 归一化层
- 经典卷积网络结构

历年 ILSVRC 比赛冠军

南京大学 NANJING UNIVERSITY

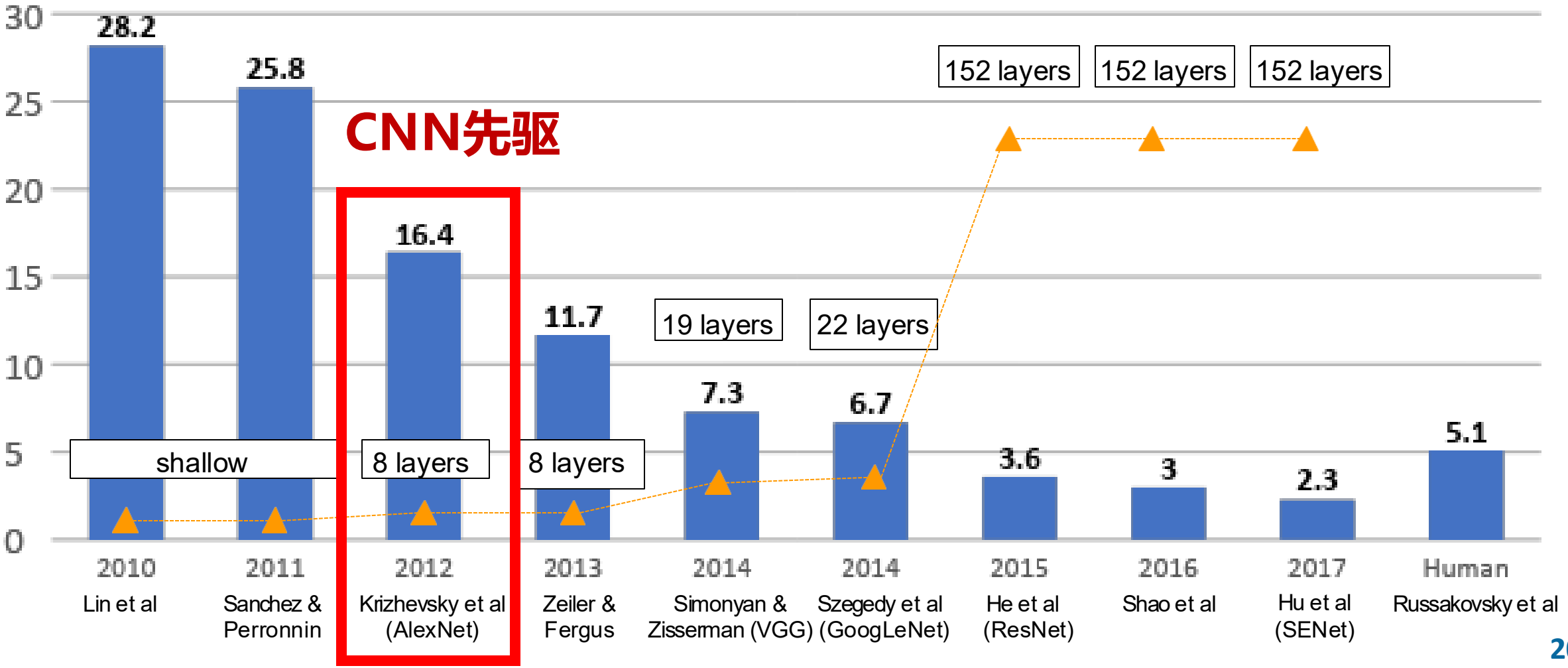
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



历年 ILSVRC 比赛冠军

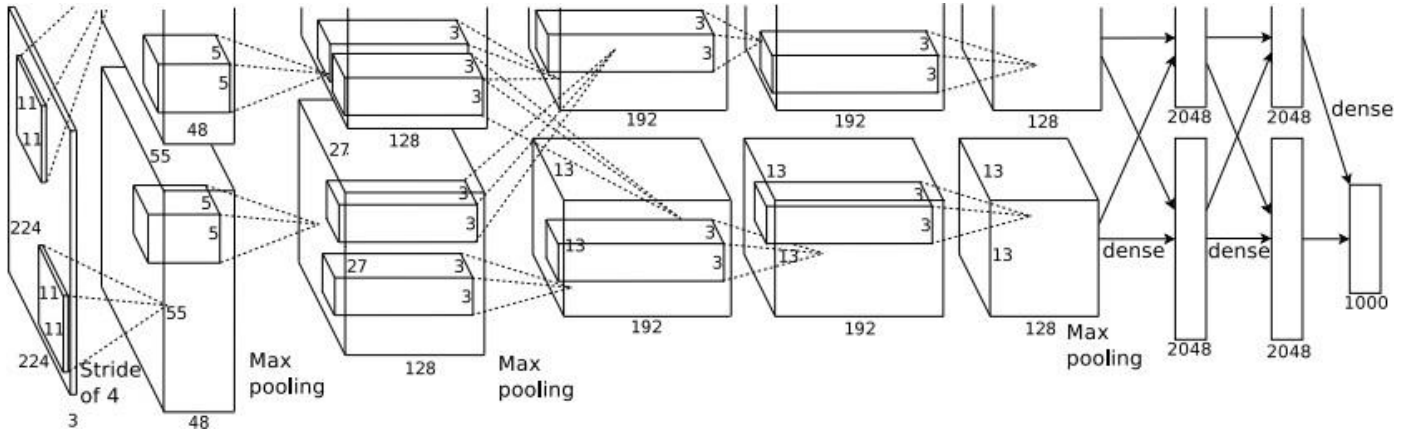
南京大学 NANJING UNIVERSITY

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



AlexNet

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

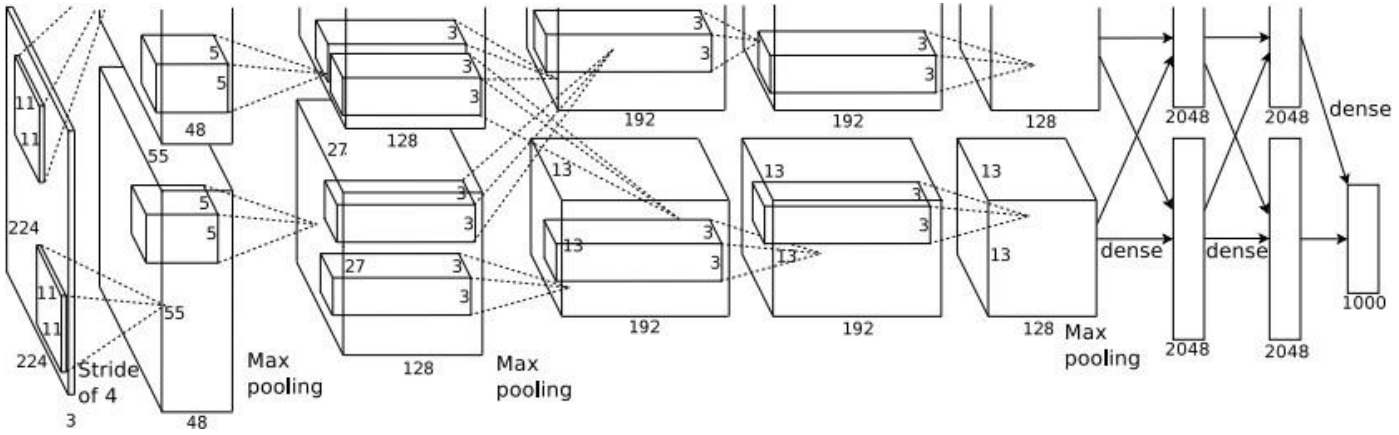


NIPS papers
https://papers.nips.cc › paper › 4824-imagenet-classificat...

ImageNet Classification with Deep Convolutional Neural ...

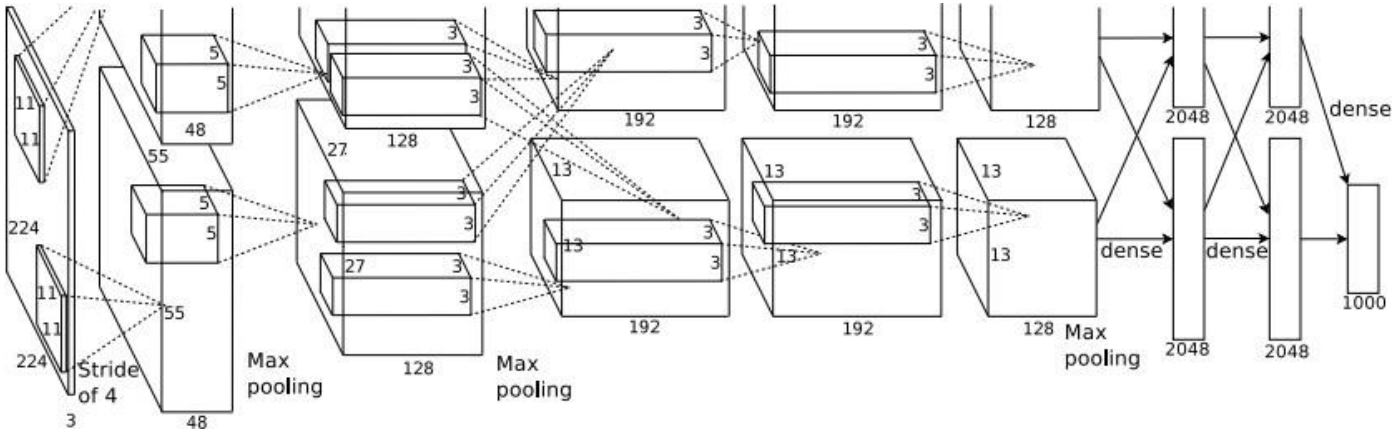
by A Krizhevsky · 2012 · Cited by 134307 — The neural network, which has 60 million parameters and 500,000 neurons, consists of five convolutional layers, some of which are…

AlexNet



- 输入：227x227x3
- 第一层：CONV1，96个11x11滤波器，步长为4

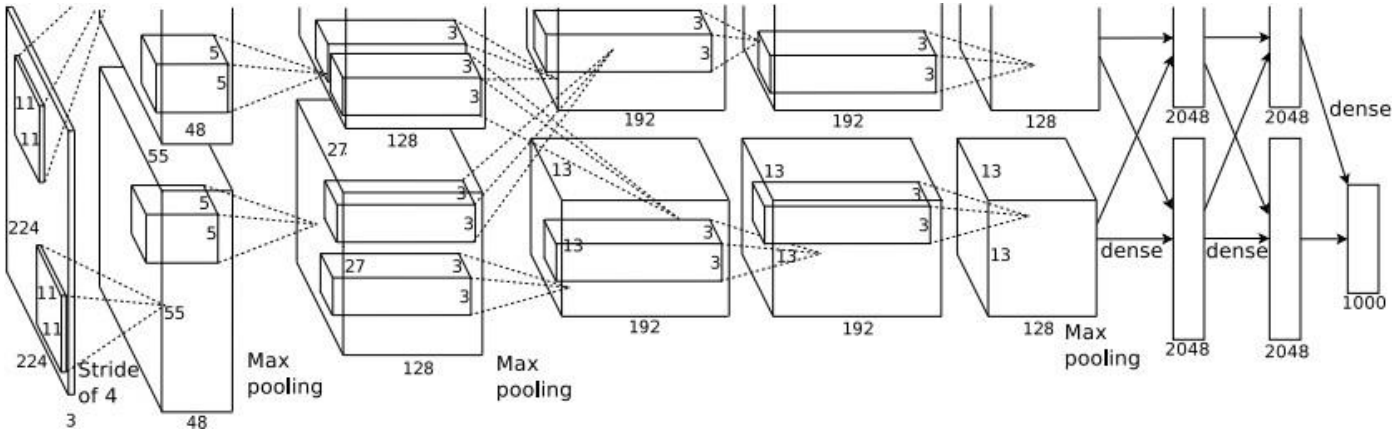
AlexNet



- 输入：227x227x3
- 第一层：CONV1，96个11x11滤波器，步长为4
- 输出：55x55x96

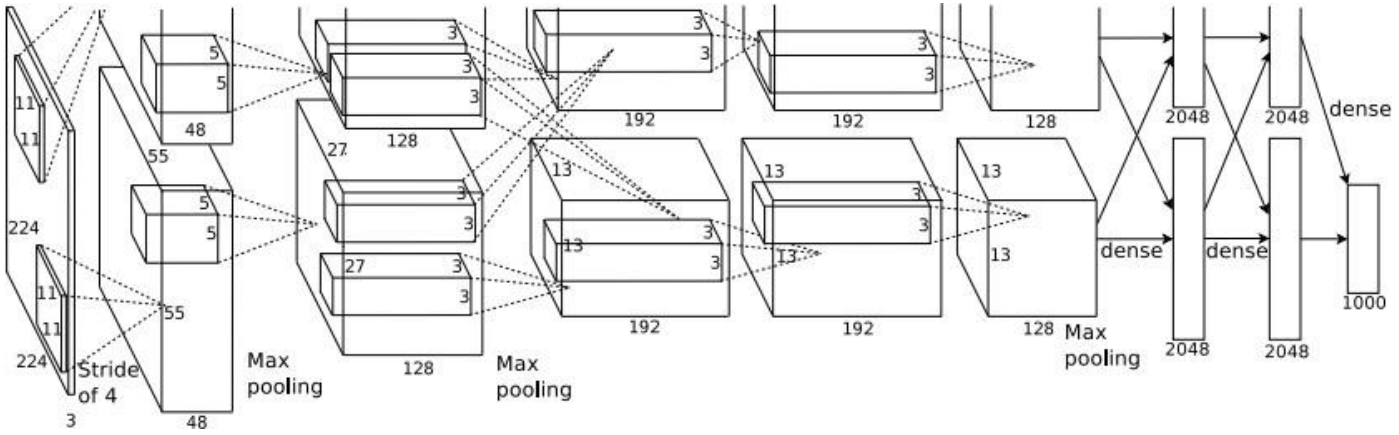
$$W' = (W - F + 2P) / S + 1$$

AlexNet



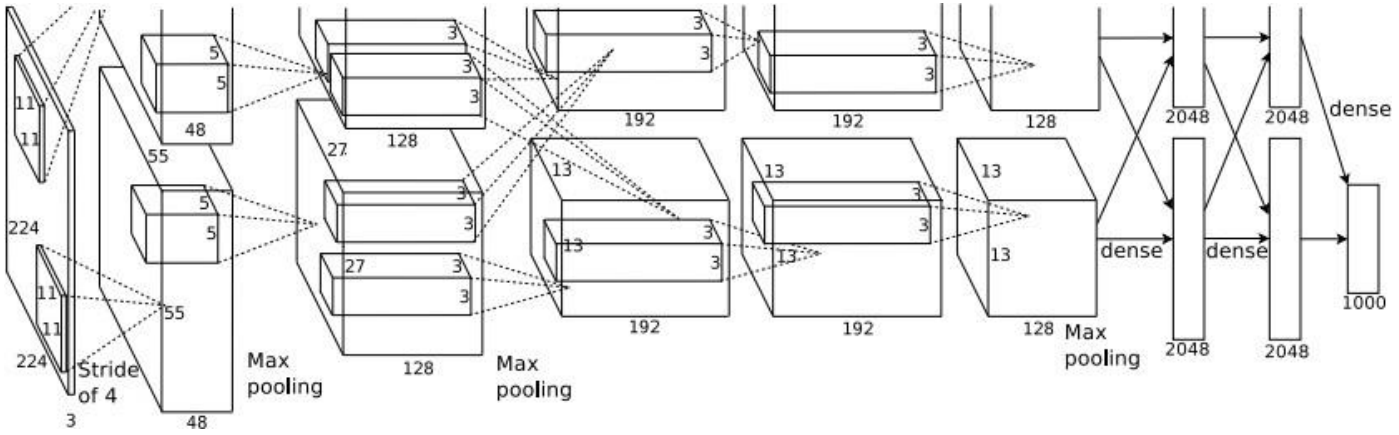
- 输入：227x227x3
- 第一层：CONV1，96个11x11滤波器，步长为4
- 输出：55x55x96
- 参数量： (11*11*3 + 1)*96 = 35K

AlexNet



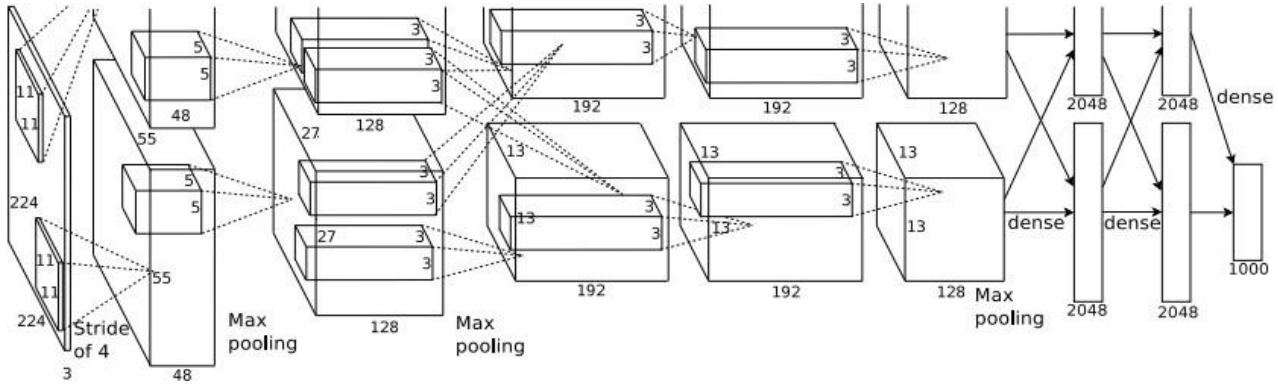
- 输入：227x227x3
- 第二层：POOL1，3x3大小，步长为2

AlexNet



- 输入：227x227x3
- 第二层：POOL1，3x3大小，步长为2
- 输出：27x27x96
- 参数量：0

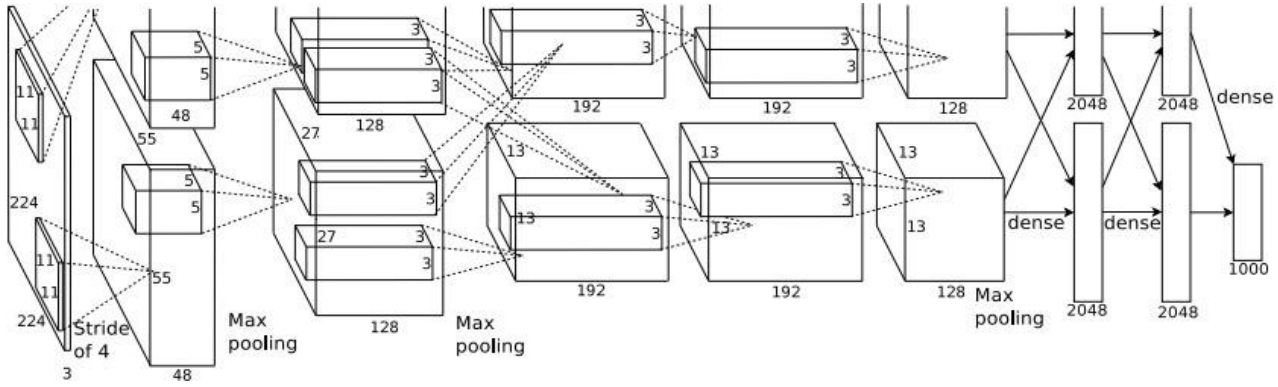
AlexNet



AlexNet 结构:
[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)

AlexNet

AlexNet 结构:
[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)



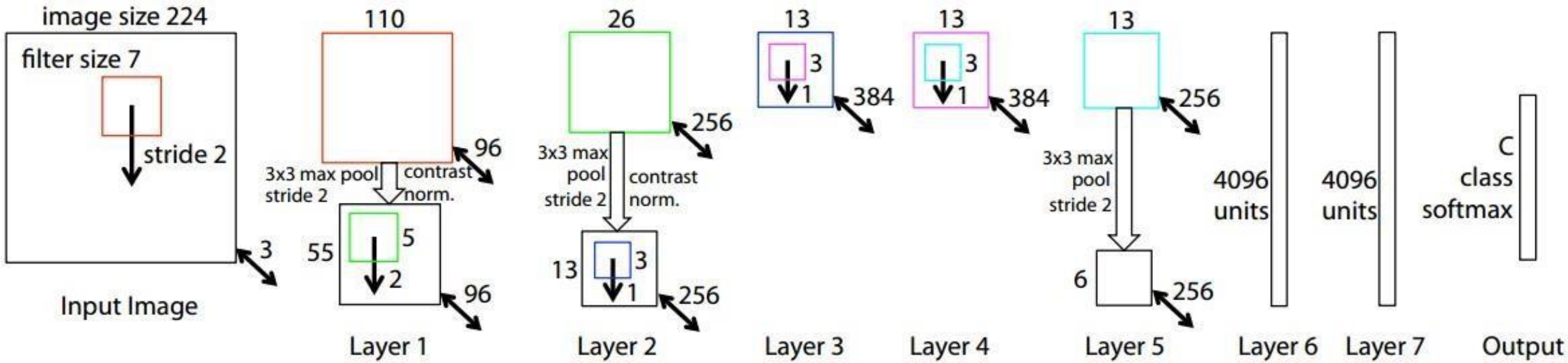
- 首次使用 ReLU
- 使用 LRN 层 (现在不常用)
- 大量使用数据增强
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- 学习率 1e-2, 在验证集准确率达到高原时学习率乘以0.1
- L2 weight decay 5e-4
- 7个模型融合: 18.2% -> 15.4%

ZFNet

arXiv
https://arxiv.org › cs

Visualizing and Understanding Convolutional Networks

by MD Zeiler · 2013 · Cited by 23189 — We introduce a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of the classifier.

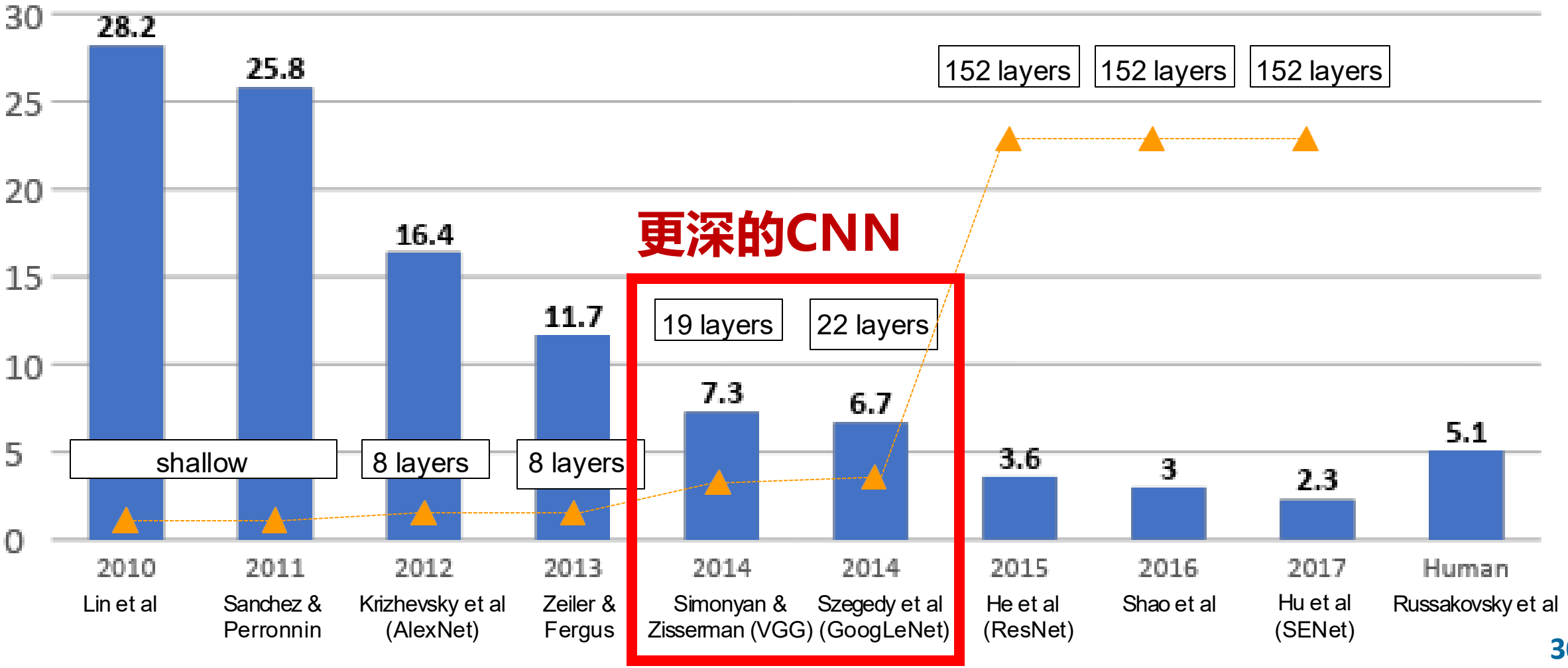


在AlexNet的基础上进行超参数优化：
CONV1: 由 (11x11 stride 4) 改为 (7x7 stride 2)
CONV3,4,5: 通道数由 384, 384, 256 改为 512, 1024, 512
ImageNet top5 error: 16.4% → 11.7%

更深的卷积神经网络

南京大学 NANJING UNIVERSITY

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



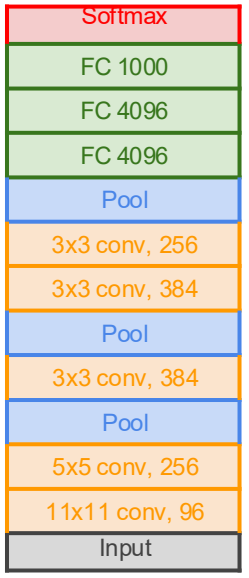
VGGNet

- 更小的滤波器，更深的网络
- 只使用
  - 3x3 Conv，步长1
  - 2x2 Max Pooling，步长2
- 深度：8→16-19
- 精度：11.7%→7.3%

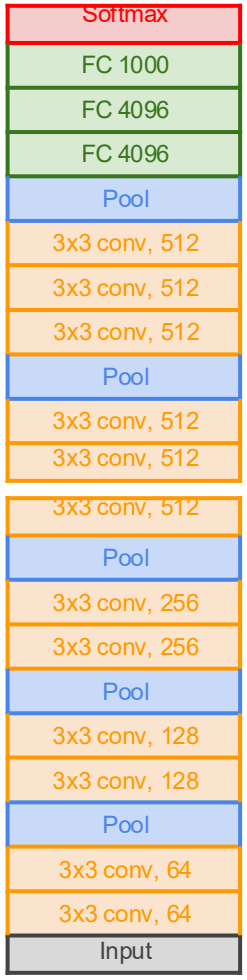
arXiv
https://arxiv.org › cs

Very Deep Convolutional Networks for Large-Scale Image ...

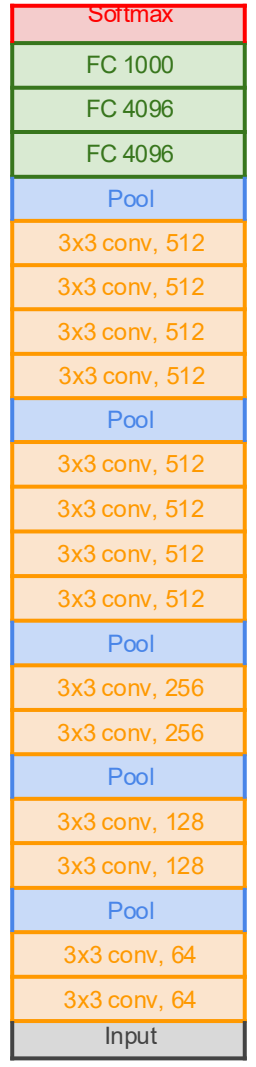
by K Simonyan · 2014 · Cited by 131122 — In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting.



AlexNet



VGG16



VGG19

VGGNet

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

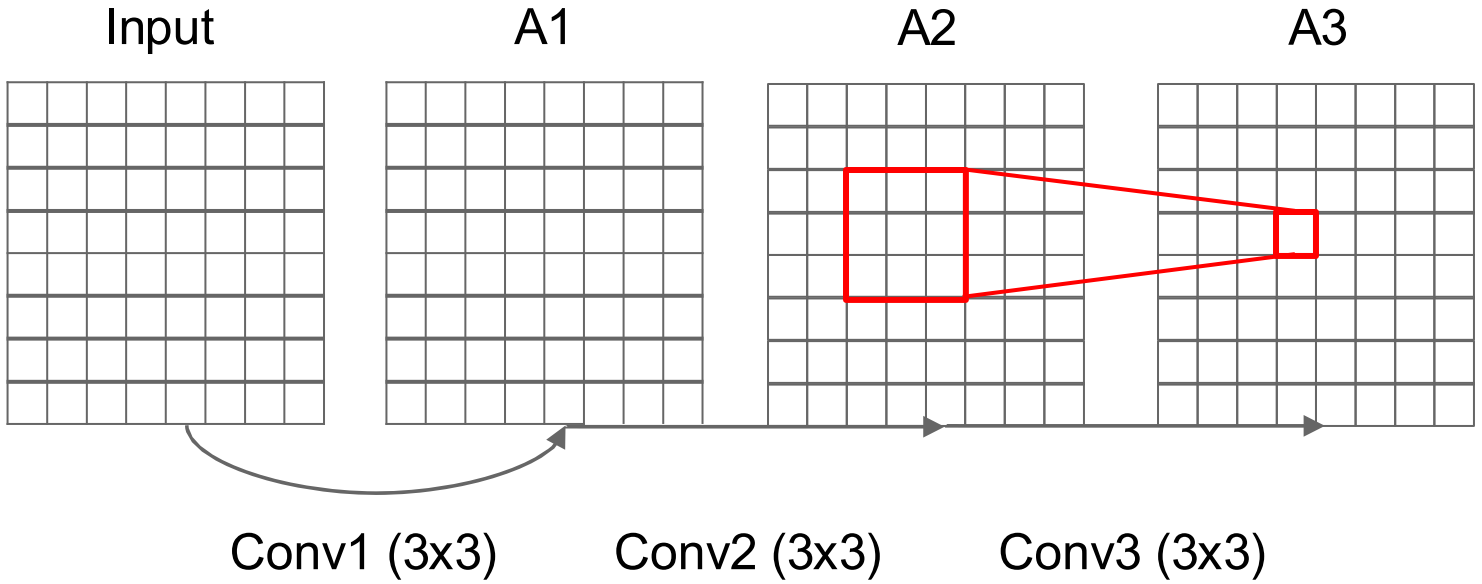
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

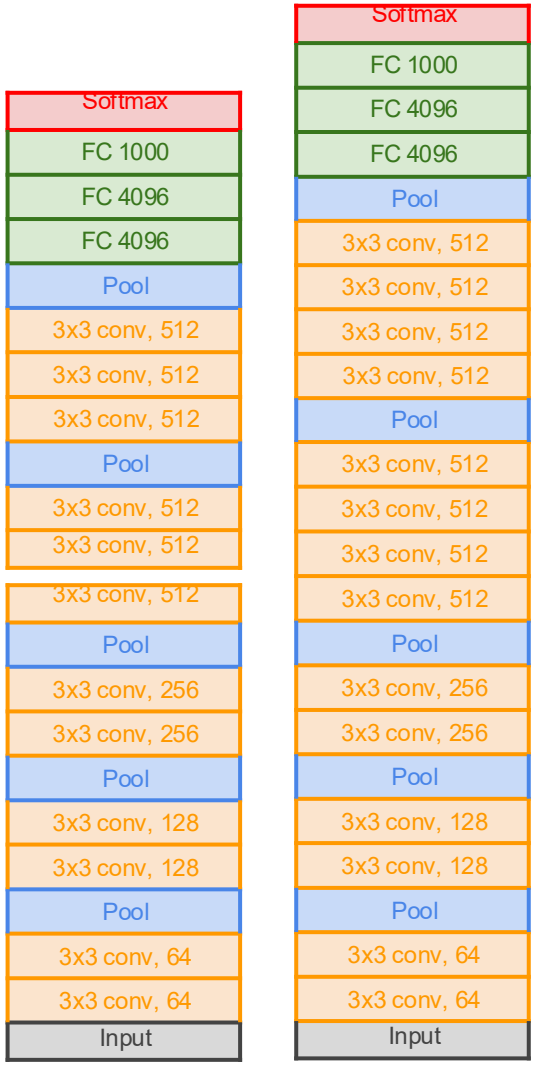
Table 2: **Number of parameters** (in millions).

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

VGGNet

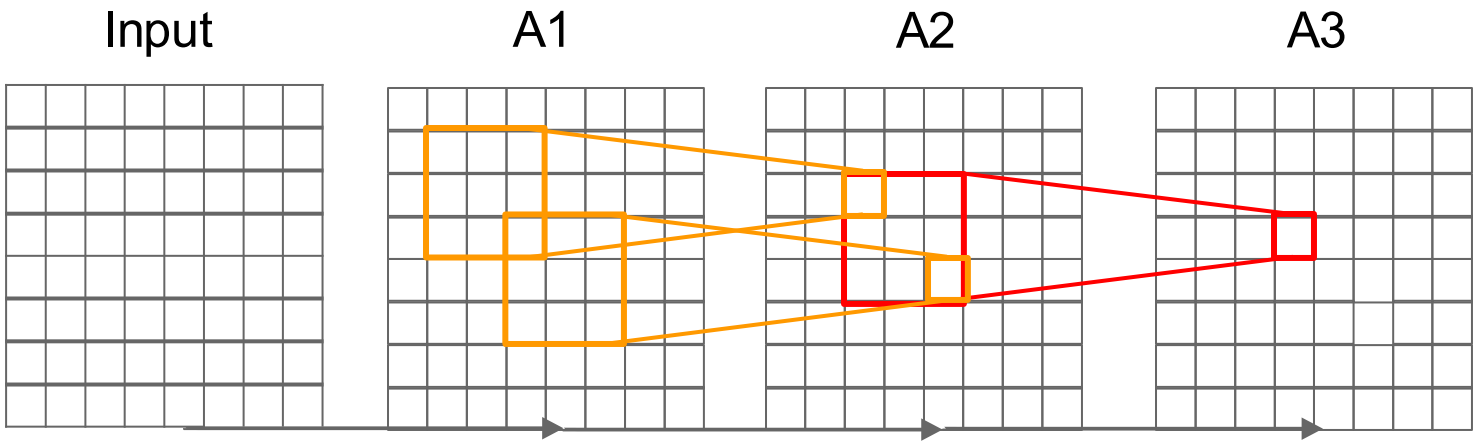
- 为什么只使用3x3 Conv?
- 多个 3x3 Conv 相对一个 7x7 Conv：
  - 相同的感受野
  - 更少的参数量

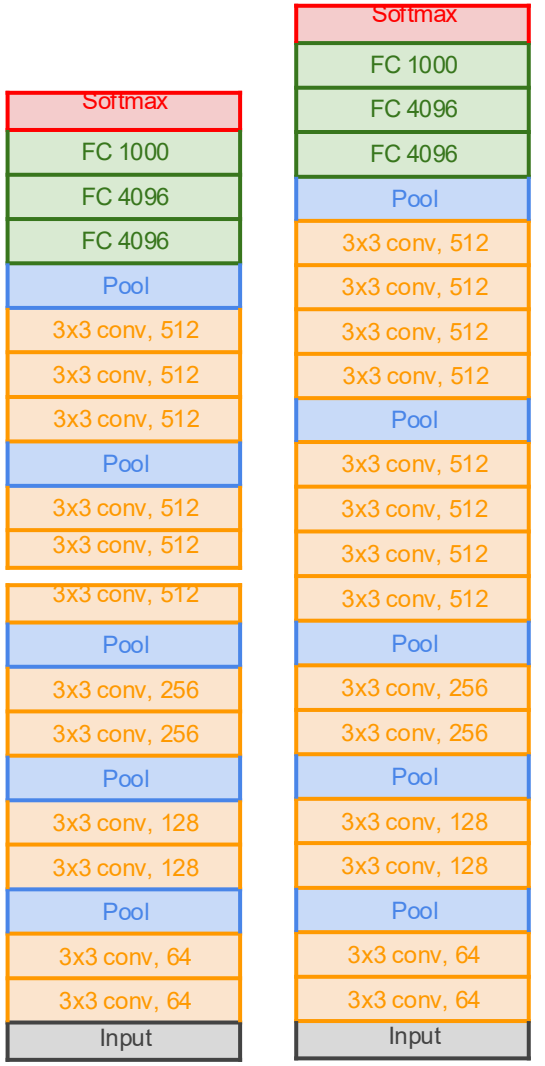




VGGNet

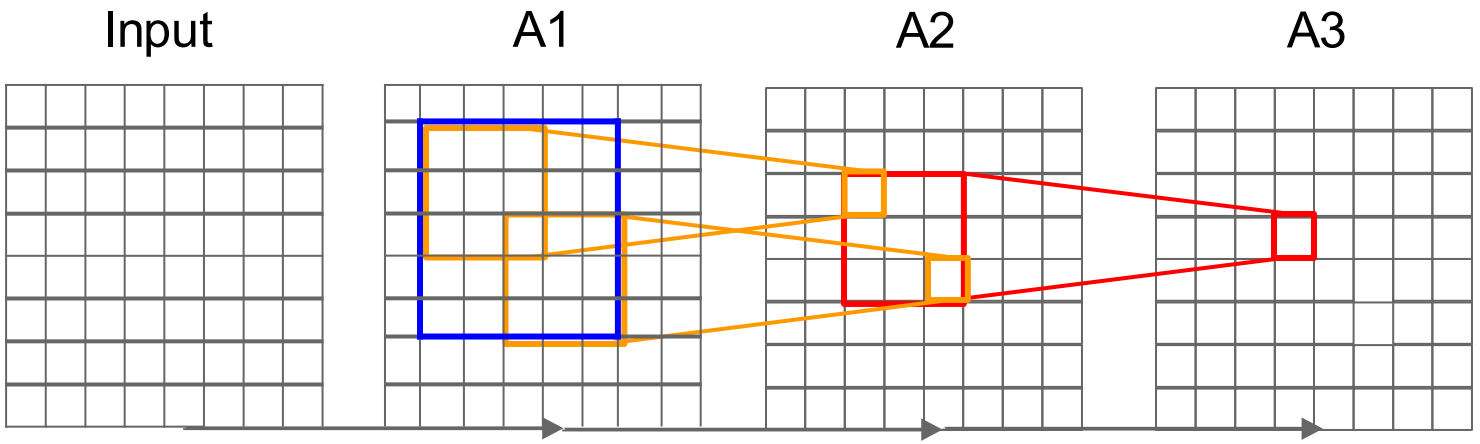
- 为什么只使用3x3 Conv?
- 多个 3x3 Conv 相对一个 7x7 Conv：
  - 相同的感受野
  - 更少的参数量

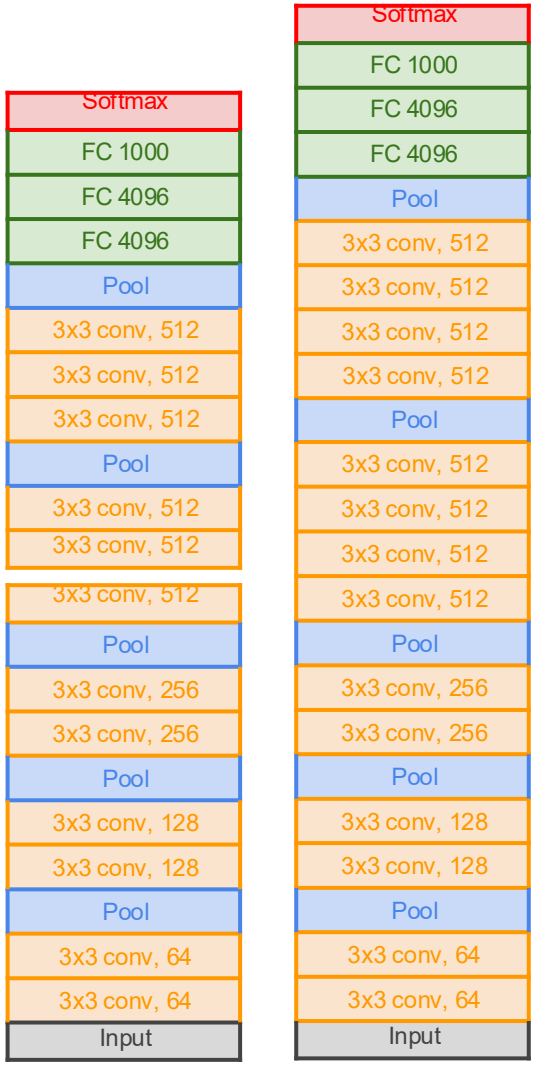




VGGNet

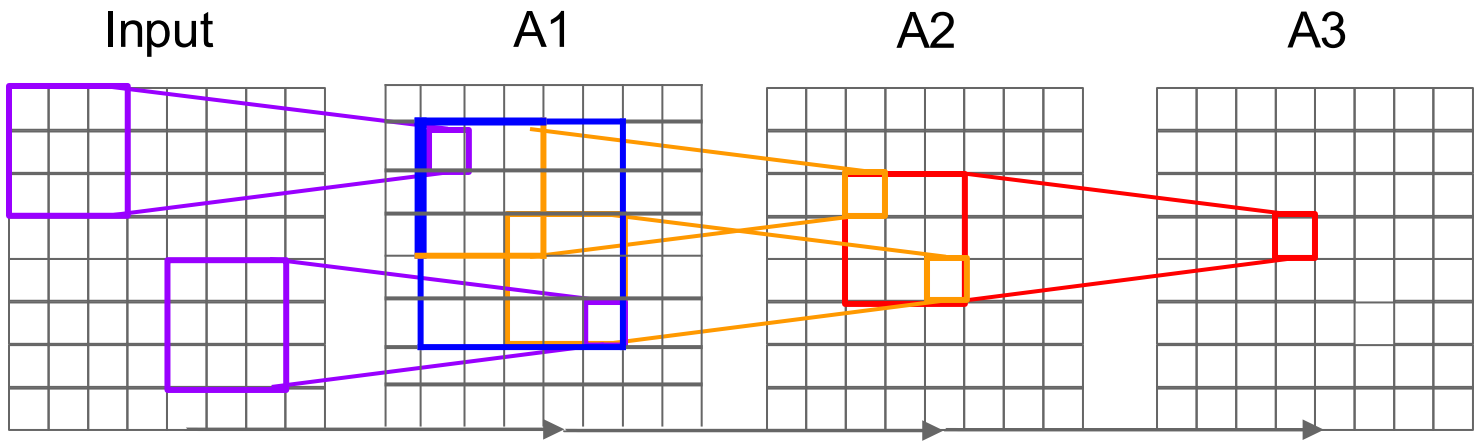
- 为什么只使用3x3 Conv?
- 多个 3x3 Conv 相对一个 7x7 Conv：
  - 相同的感受野
  - 更少的参数量

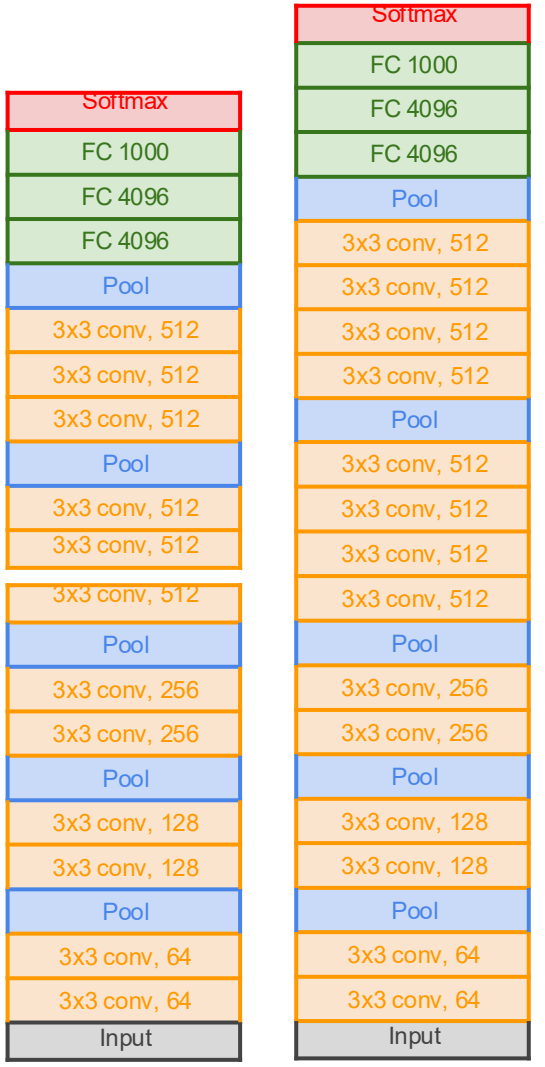




VGGNet

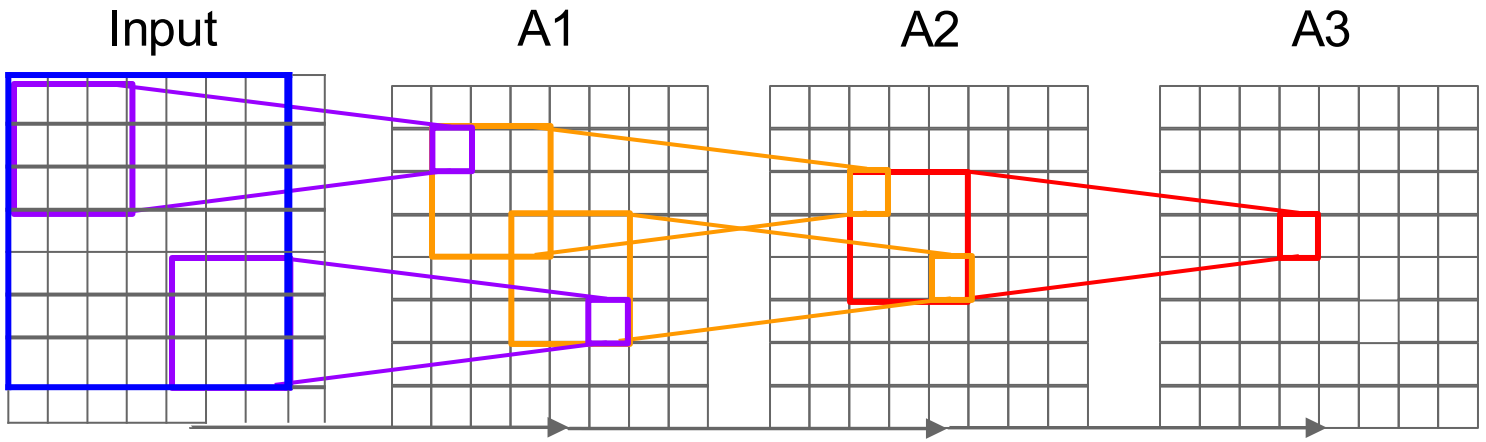
- 为什么只使用3x3 Conv?
- 多个 3x3 Conv 相对一个 7x7 Conv：
  - 相同的感受野
  - 更少的参数量

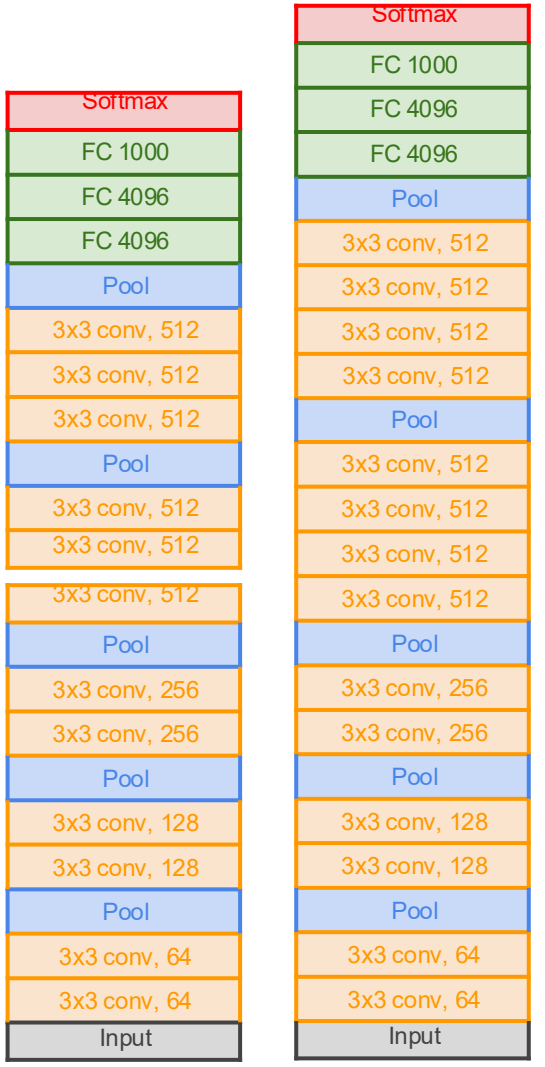




VGGNet

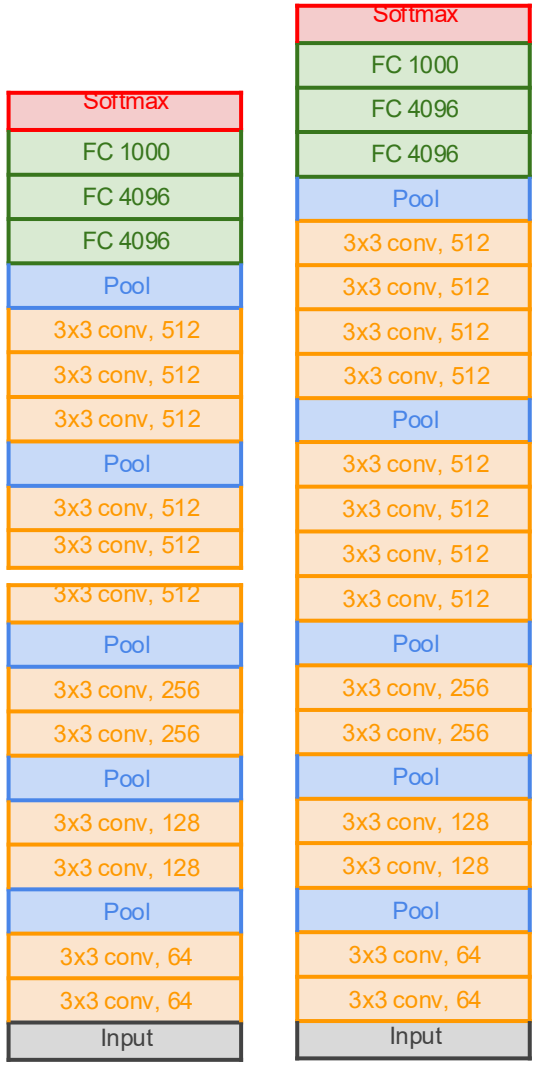
- 为什么只使用3x3 Conv?
- 多个 3x3 Conv 相对一个 7x7 Conv：
  - 相同的感受野
  - 更少的参数量





VGGNet

- 为什么只使用3x3 Conv?
- 多个 3x3 Conv 相对一个 7x7 Conv：
  - 相同的感受野
  - 更少的参数量
  - 更深，更多的非线性变换



VGG16　　VGG19

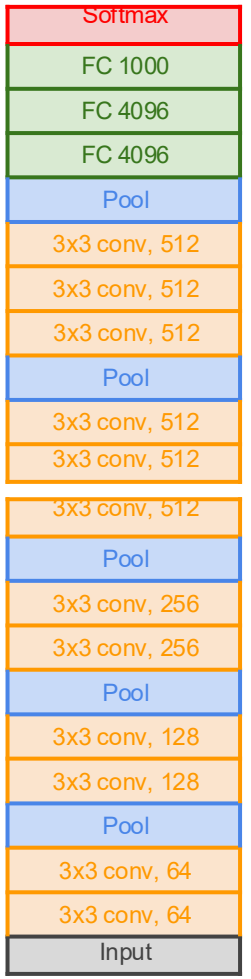
VGGNet

不计算bias

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

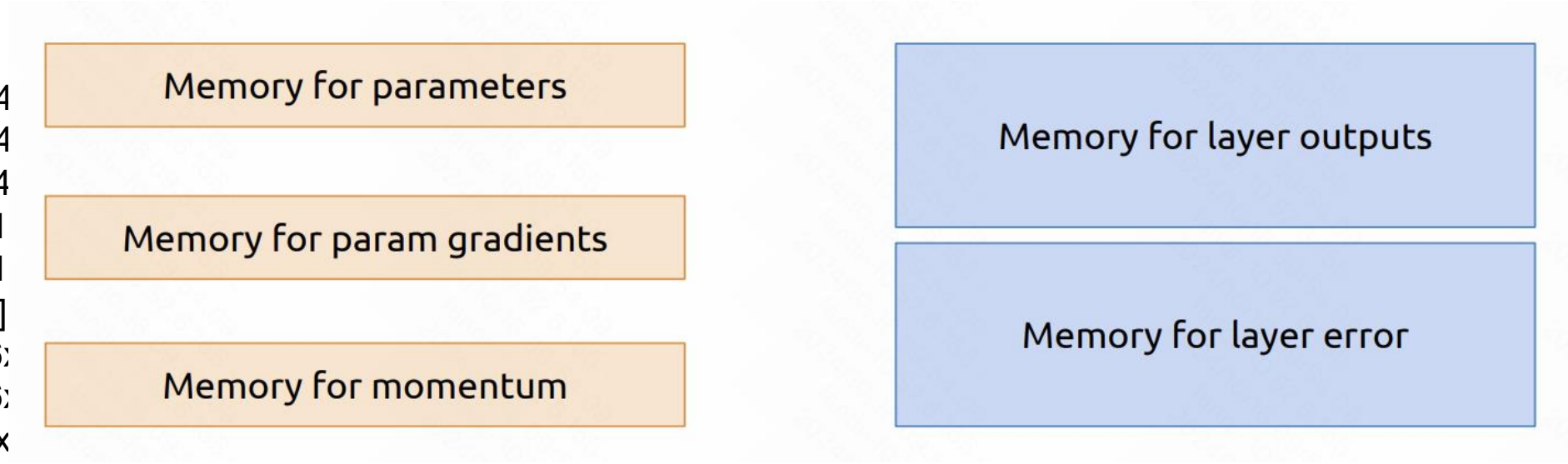
总显存占用: 24M * 4 bytes
~= 96MB / image

总参数量: 138M parameters



VGG16

VGGNet



INPUT: [224x224x3]
CONV3-64: [224x224
CONV3-64: [224x224
POOL2: [112x112x64
CONV3-128: [112x11
CONV3-128: [112x11
POOL2: [56x56x128]
CONV3-256: [56x56
CONV3-256: [56x56
CONV3-256: [56x56x
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

总显存占用: 24M * 4 bytes
~= 96MB / image

总参数量: 138M parameters

VGGNet

不计算bias

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

显存占用瓶颈
浅层Conv

参数量瓶颈
FC层

总显存占用: 24M * 4 bytes
~= 96MB / image

总参数量: 138M parameters

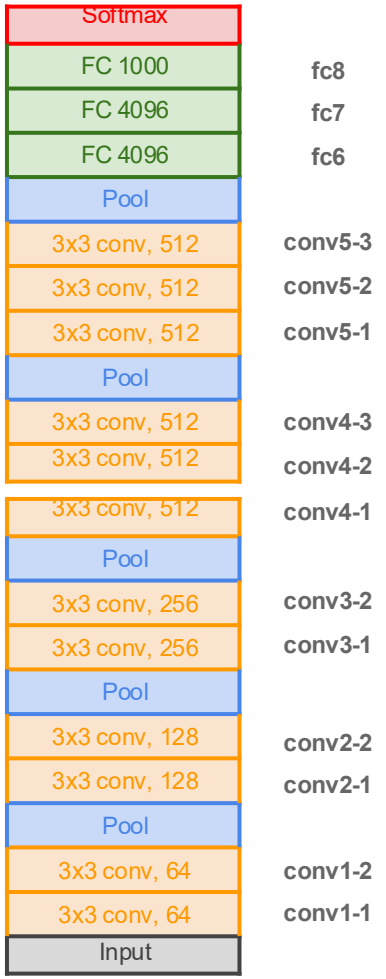
VGGNet

不计算bias

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

总显存占用: 24M * 4 bytes
~= 96MB / image

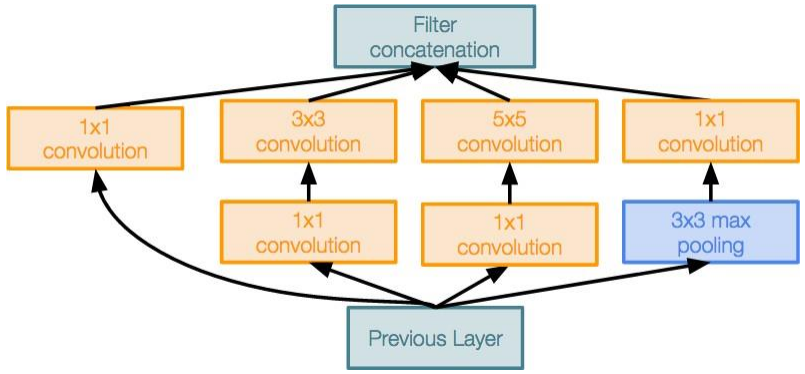
总参数量: 138M parameters



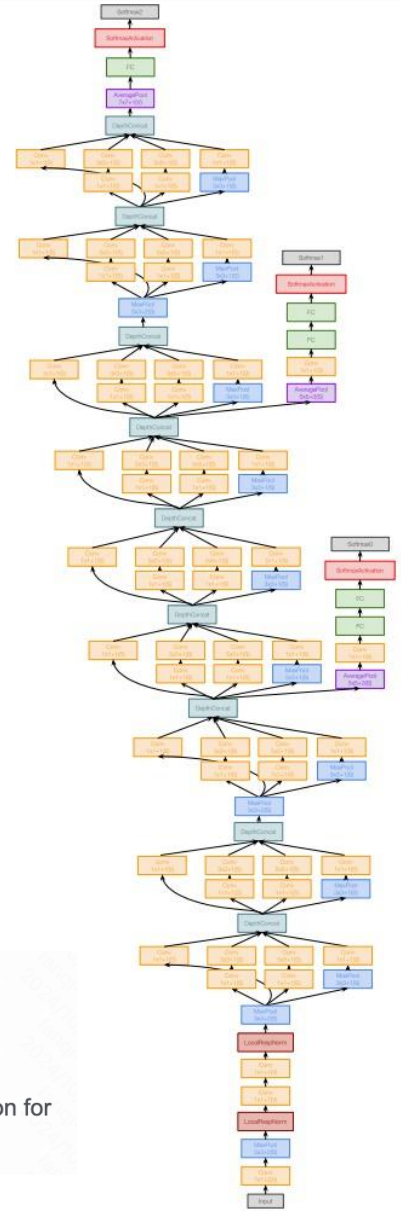
VGG16

GoogLeNet

- 更深的网络，更高的计算效率
- ILSVRC’14 分类任务冠军 (6.7% top 5 error)
- 22层
- 只有 5 million 参数
  - 12x less than AlexNet
  - 27x less than VGG-16
- 高效的 Inception 模块
- 没有 FC 隐藏层



Inception module



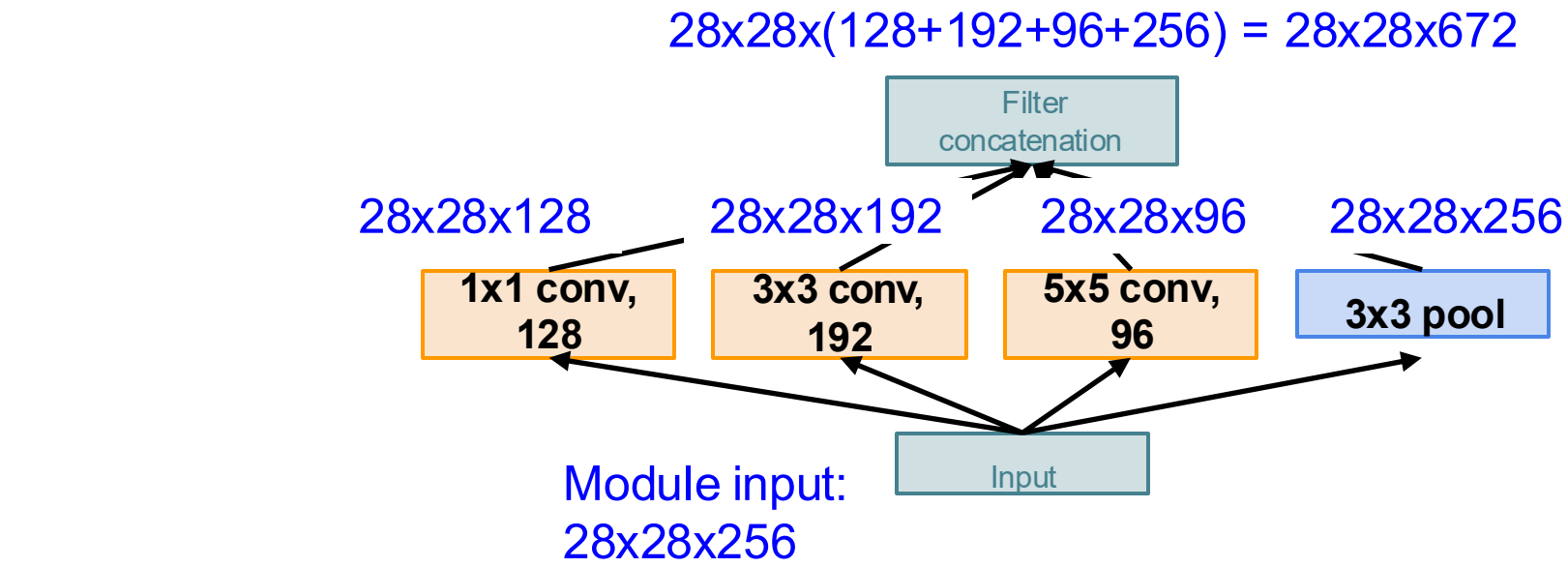
arXiv
https://arxiv.org › cs

[1409.4842] Going Deeper with Convolutions

by C Szegedy · 2014 · Cited by 62440 — One particular incarnation used in our submission for ILSVRC 2014 is called **GoogLeNet**, a 22 layers deep network, the quality of which is ...

Inception 模块

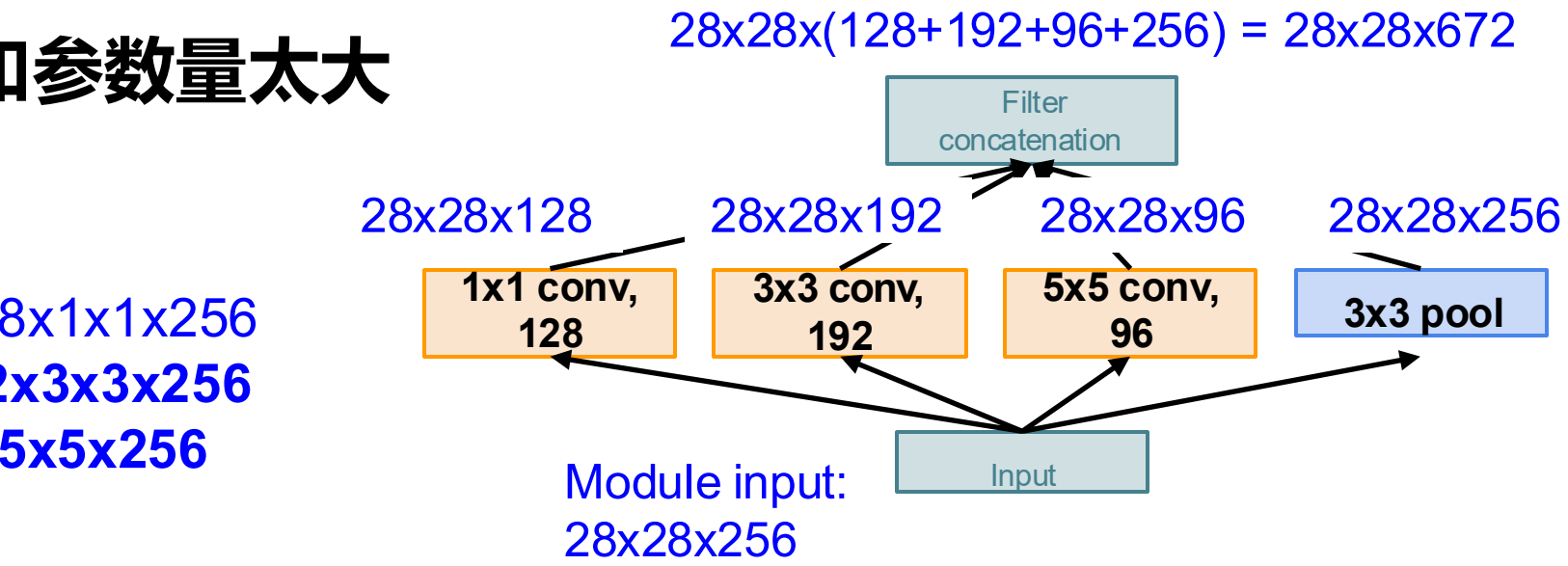
- 简单的 Inception：并联多个不同的算子



Inception 模块

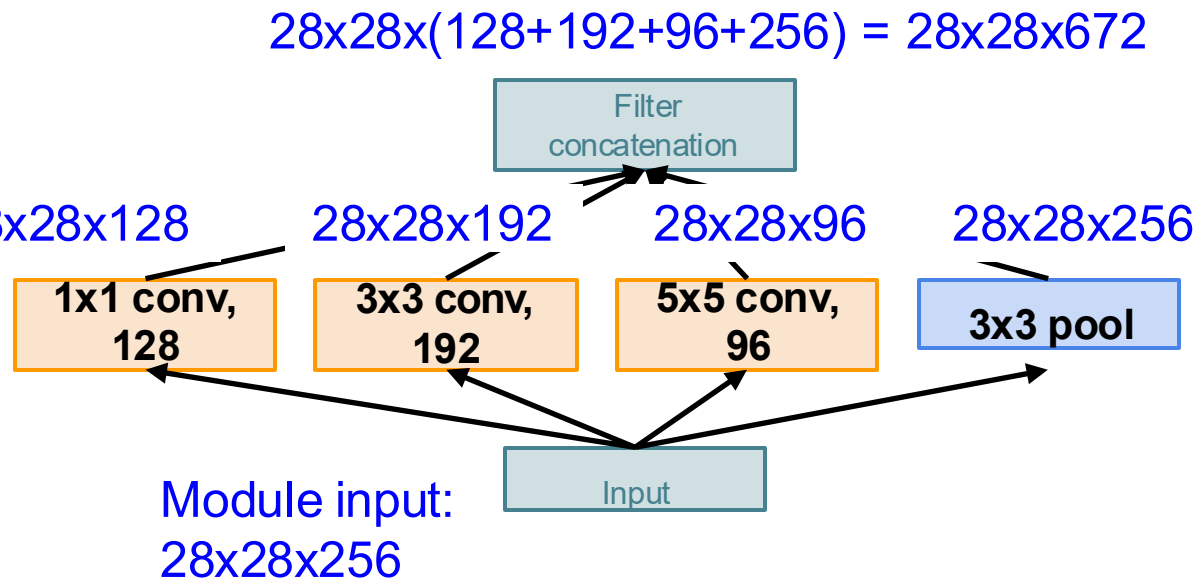
- 简单的 Inception：并联多个不同的算子
- 问题一：计算量和参数量太大

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x**192x3x3x256**
[5x5 conv, 96] 28x28x**96x5x5x256**
Total: 854M ops



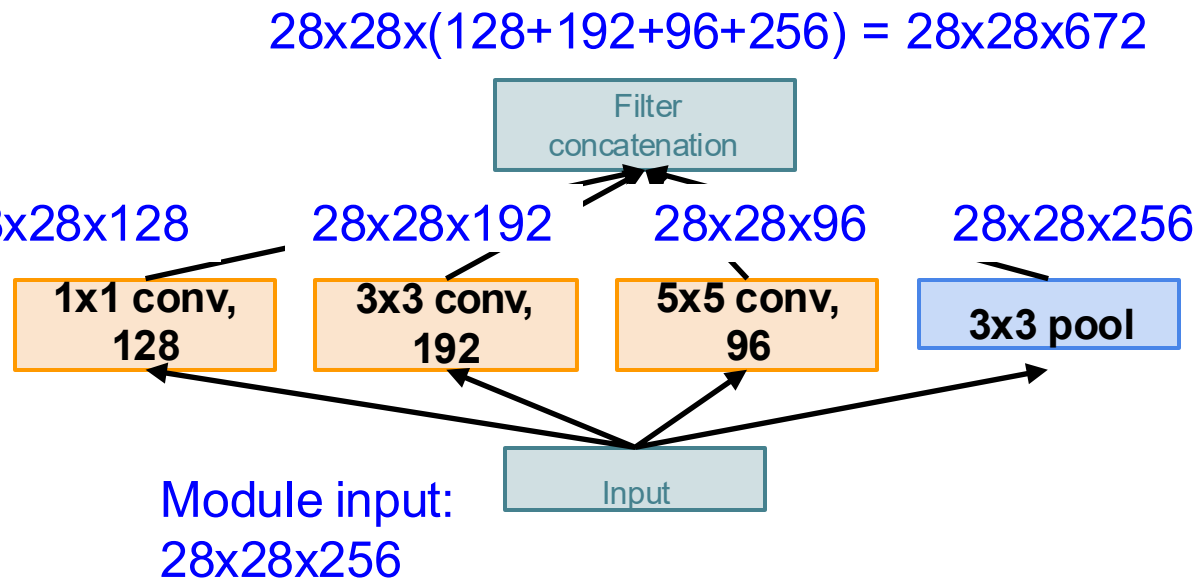
Inception 模块

- 简单的 Inception：并联多个不同的算子
- 问题一：计算量和参数量太大
- 问题二：通道数持续变大



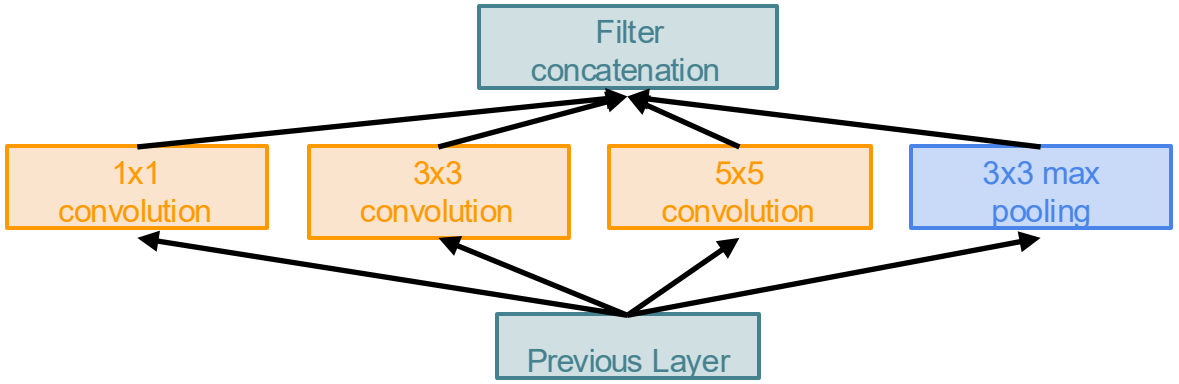
Inception 模块

- 简单的 Inception：并联多个不同的算子
- 问题一：计算量和参数量太大
- 问题二：通道数持续变大



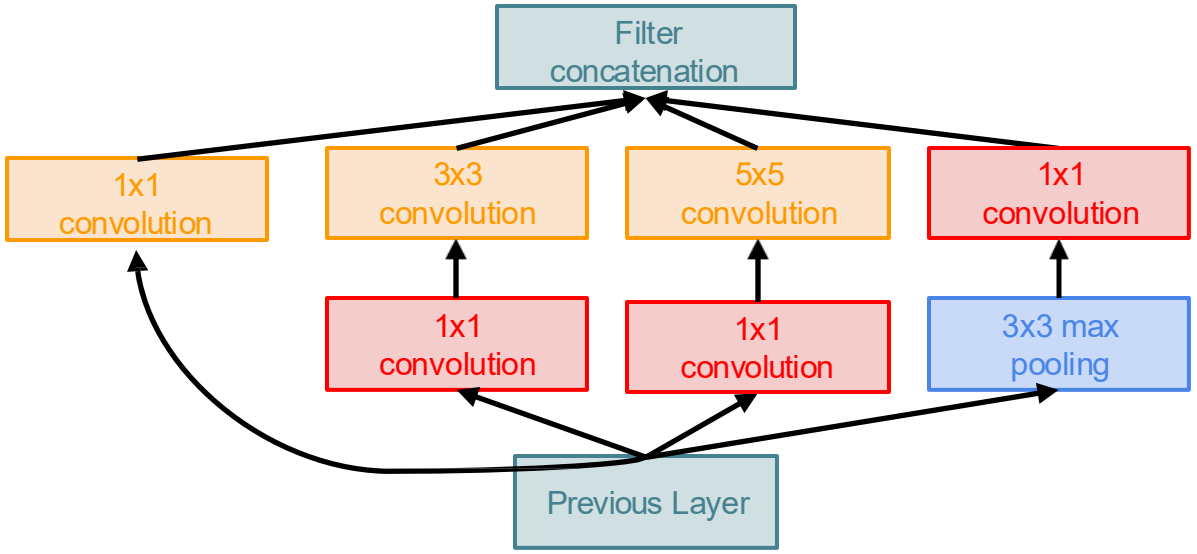
- Solution：使用 1x1 Conv 降低计算量、参数量、通道数

Inception 模块



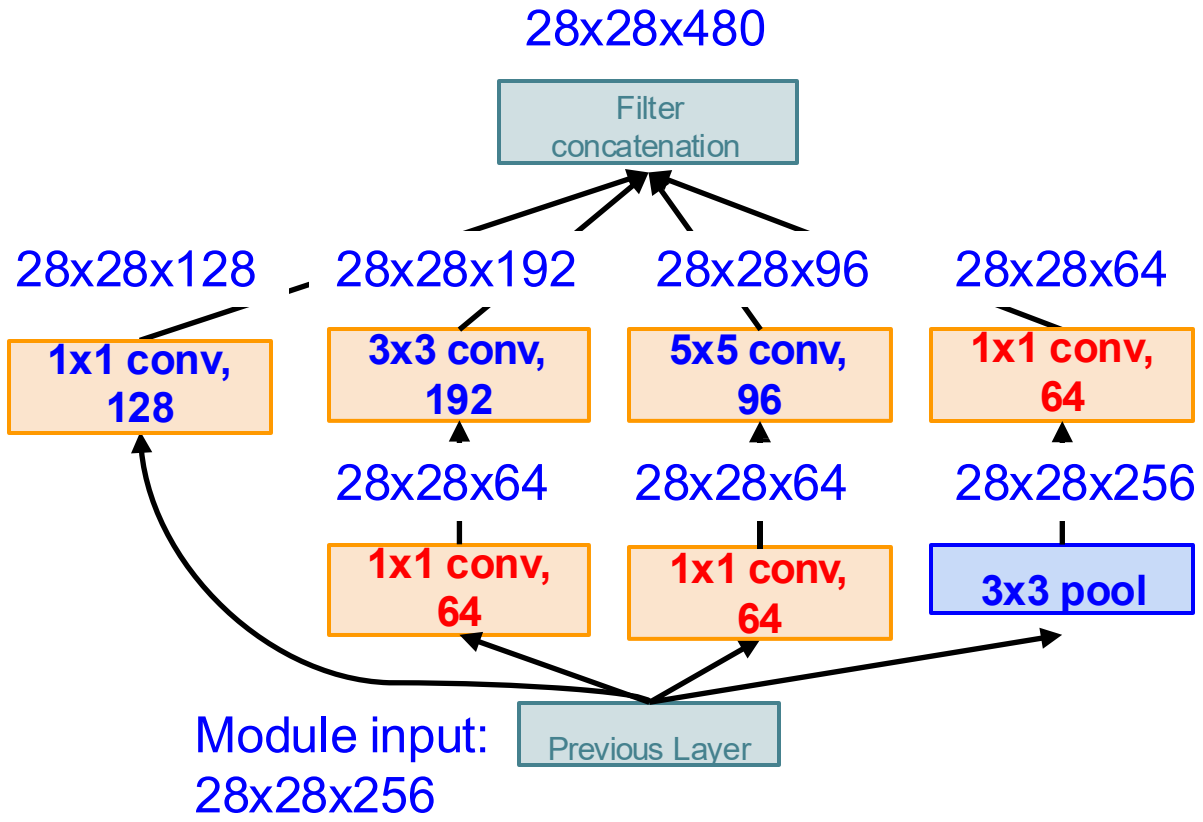
原始的 Inception 模块

1x1 conv 瓶颈层



GoogLeNet 中的 Inception 模块

Inception 模块

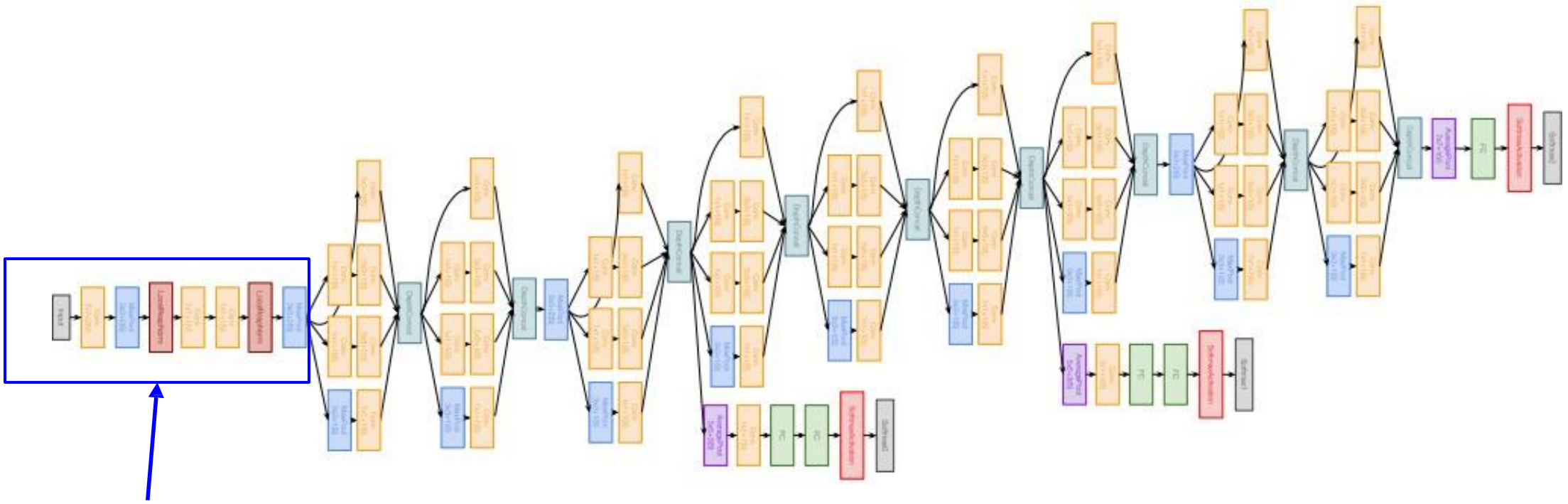


Conv Ops:

[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256

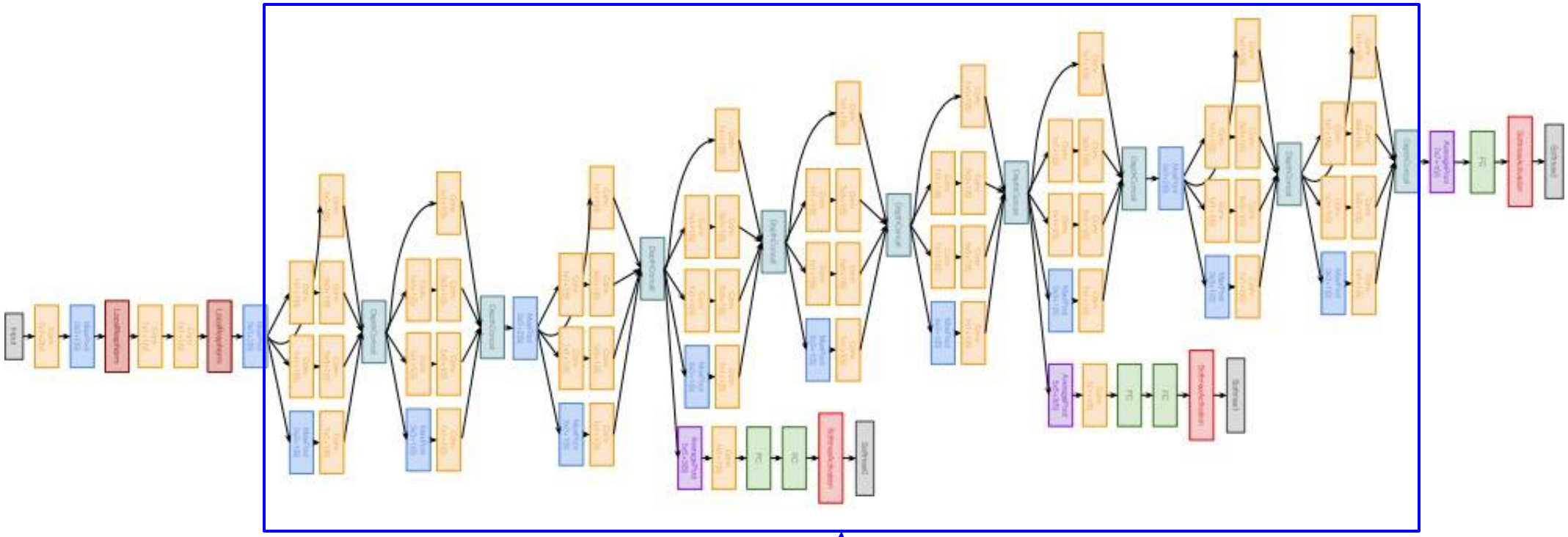
Total: 358M ops

GoogLeNet 网络架构



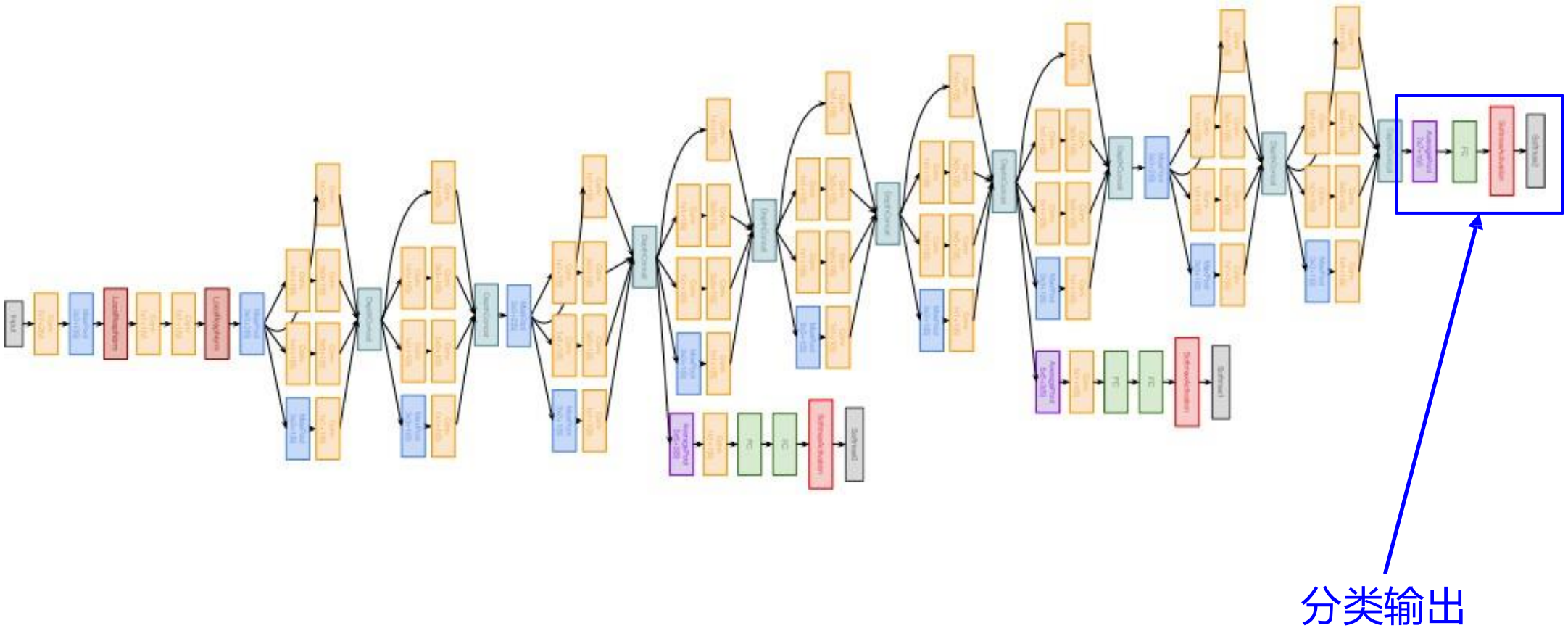
Stem Network:
Conv-Pool-
2x Conv-Pool

GoogLeNet 网络架构



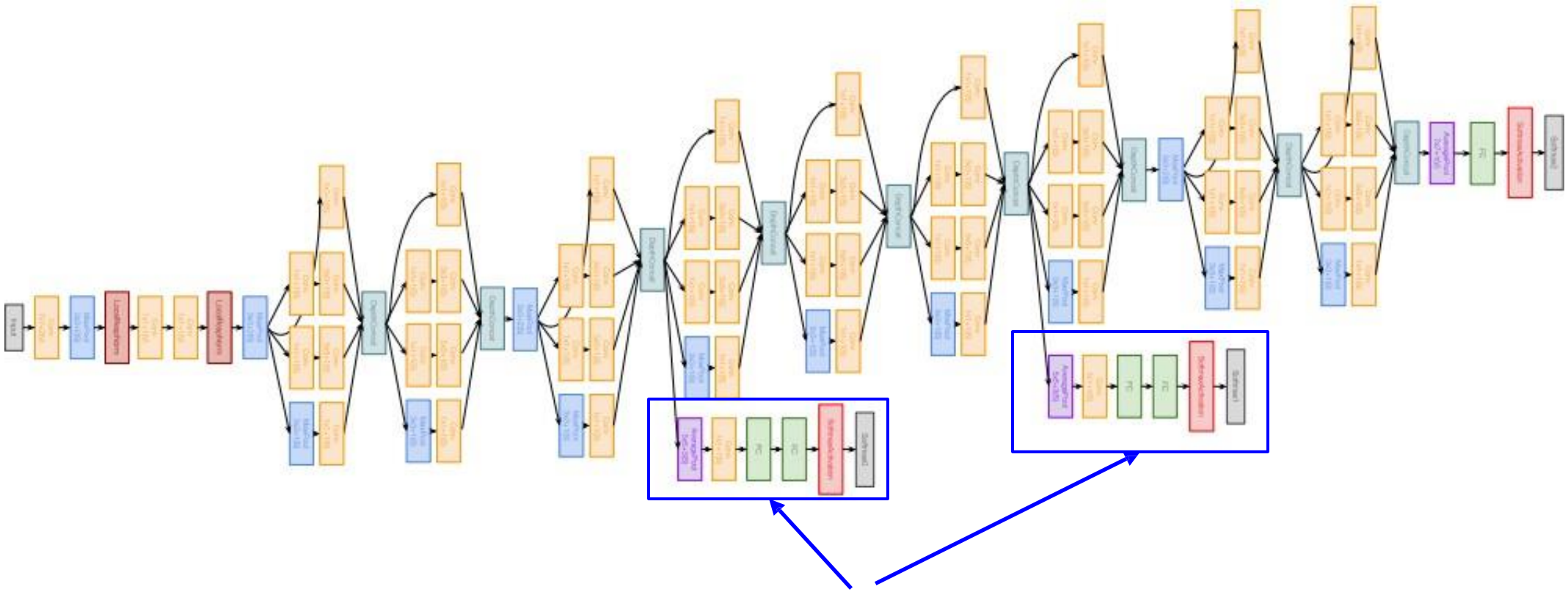
堆叠的 Inception 模块

GoogLeNet 网络架构

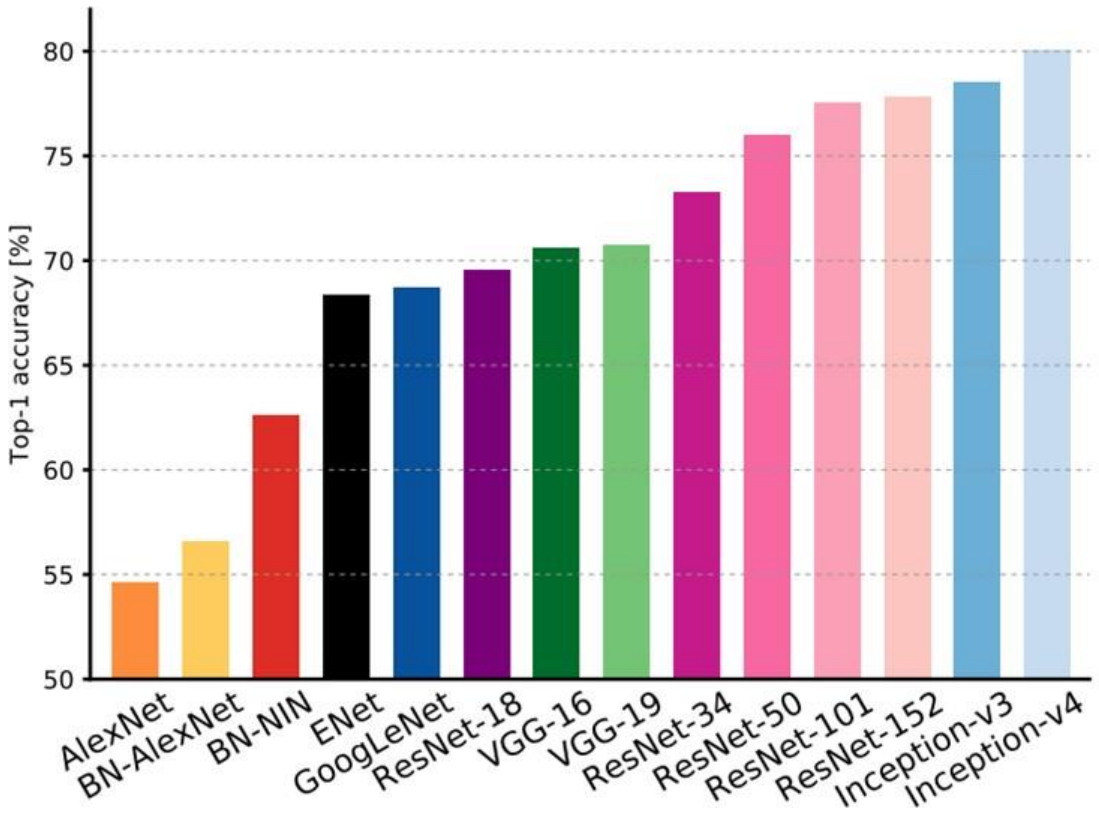


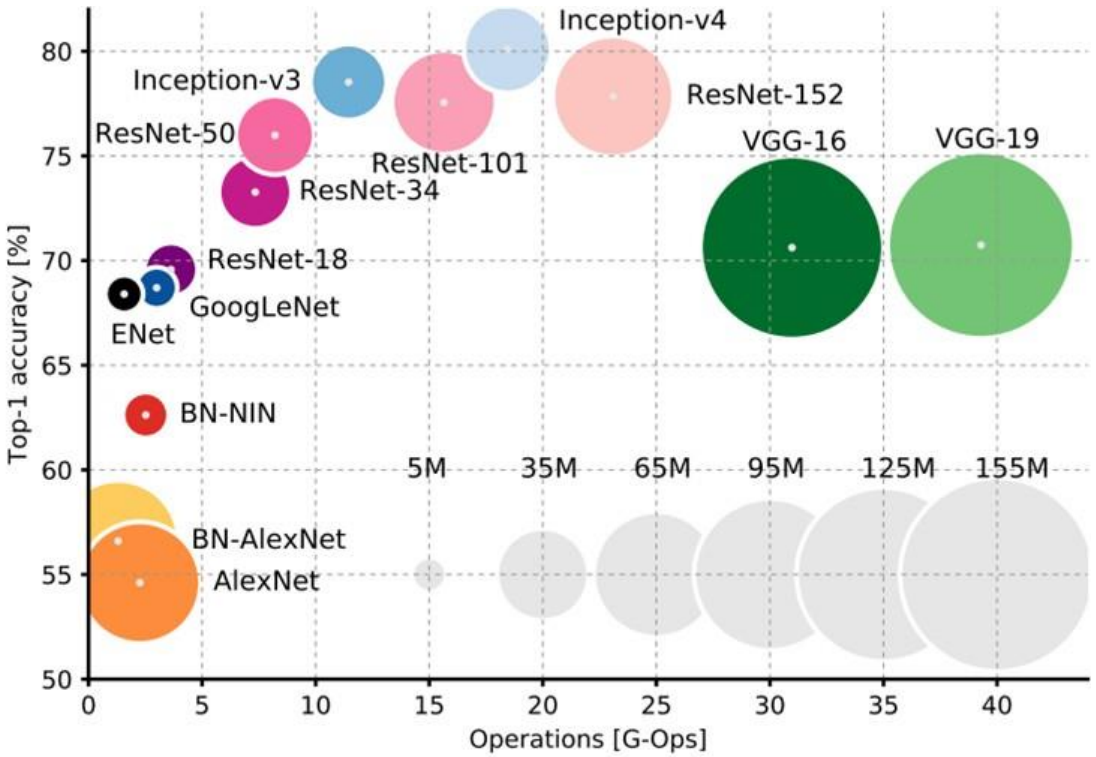
GoogLeNet 只在分类器中使用的 FC 层

GoogLeNet 网络架构



GoogLeNet 网络架构



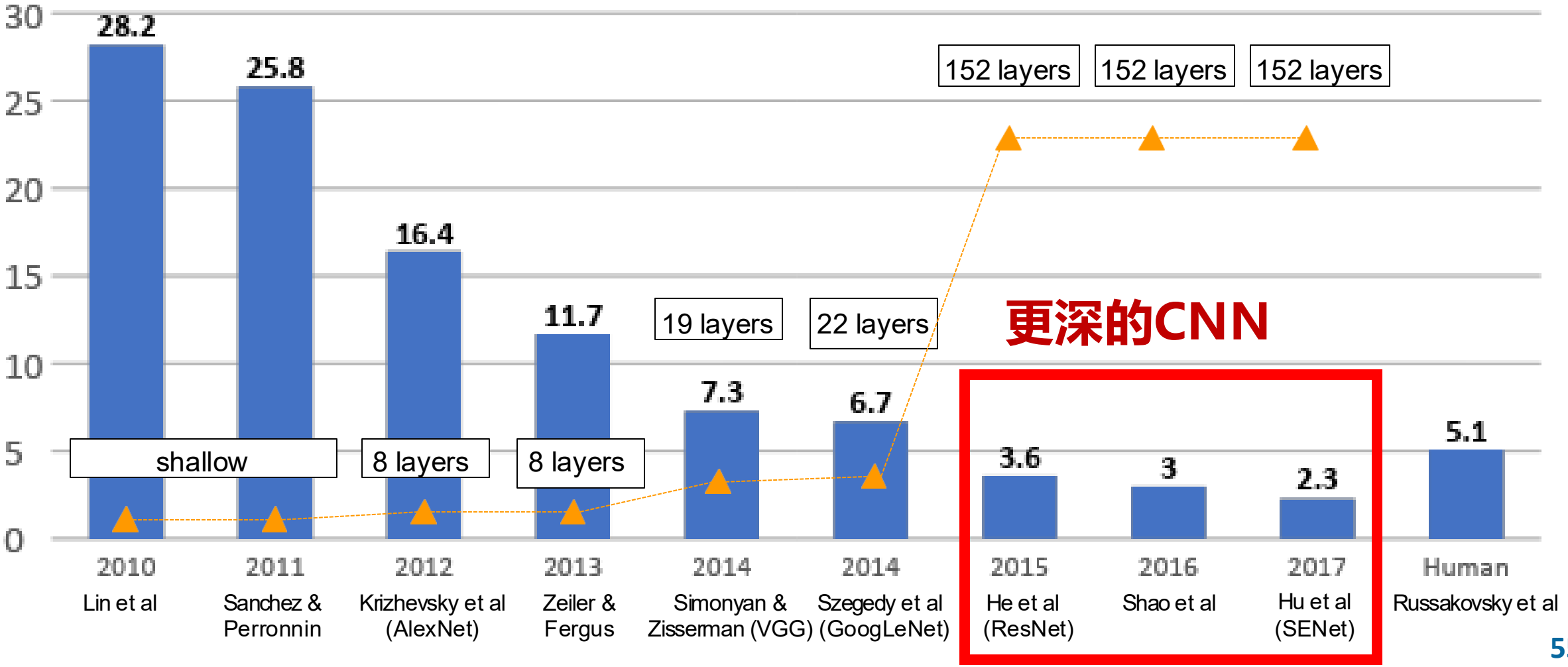


An Analysis of Deep Neural Network Models for Practical Applications, 2017.

残差卷积神经网络

南京大学 NANJING UNIVERSITY

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

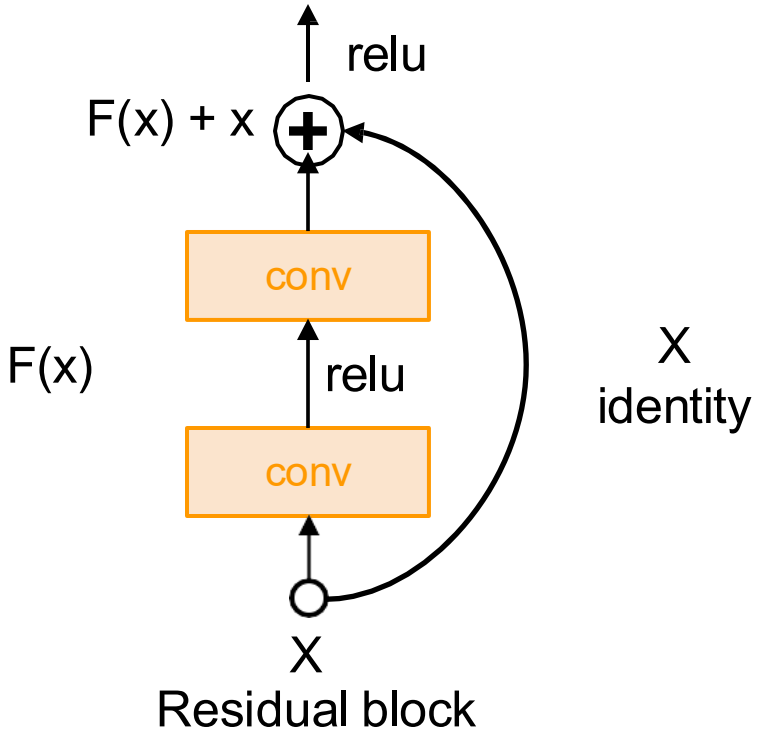


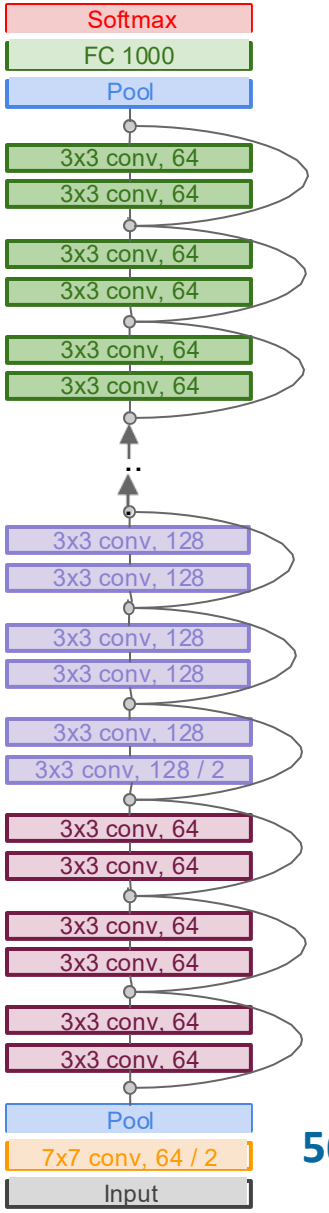
ResNet

- 残差神经网络
- ILSVRC’15 冠军
- 残差连接
- 解决 CNN 训练问题
- 152层（后续：1001层）

MSRA @ ILSVRC & COCO 2015 Competitions

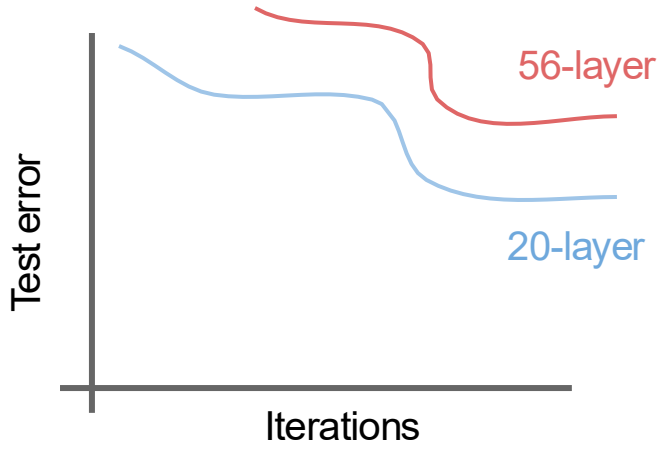
- **1st places in all five main tracks**
  - ImageNet Classification: *“Ultra-deep”* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

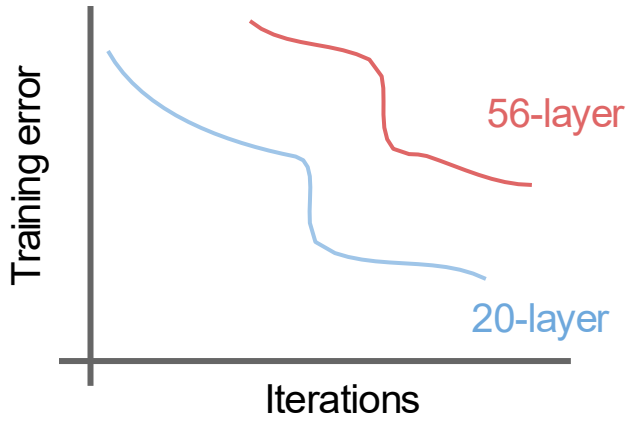




ResNet

- 训练深层卷积网络存在问题
  - 深层网络的性能比浅层的更差
  - 但**不是由过拟合造成的**





ResNet

- 训练深层卷积网络存在问题
  - 深层网络的性能比浅层的更差
  - 但不是由过拟合造成的
- 深层网络比浅层网络具有更强的表达能力（更多参数）

ResNet

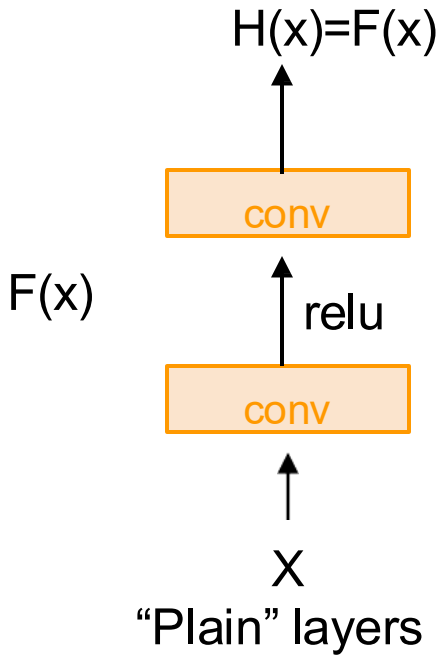
- 训练深层卷积网络存在问题
  - 深层网络的性能比浅层的更差
  - 但不是由过拟合造成的
- 深层网络比浅层网络具有更强的表达能力（更多参数）
- 猜想1：优化问题，深度网络更难优化

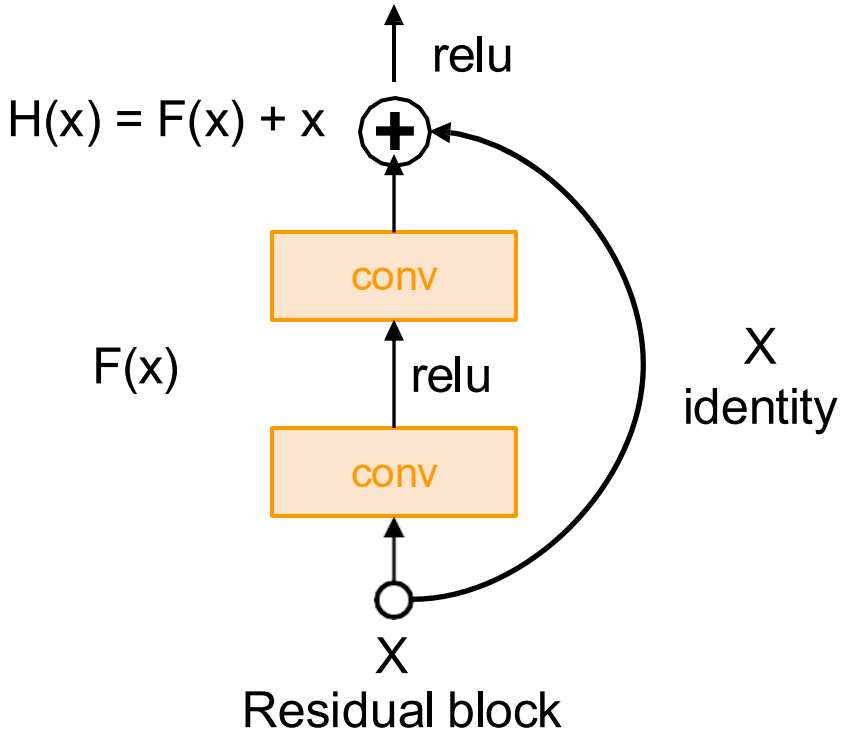
ResNet

- 训练深层卷积网络存在问题
  - 深层网络的性能比浅层的更差
  - 但不是由过拟合造成的
- 深层网络比浅层网络具有更强的表达能力（更多参数）
- 猜想1：优化问题，深度网络更难优化
- 关键问题：让深层网络的能力至少与浅层网络一样好（恒等映射）

ResNet

- **关键问题：让深层网络的能力至少与浅层网络一样好（恒等映射）**

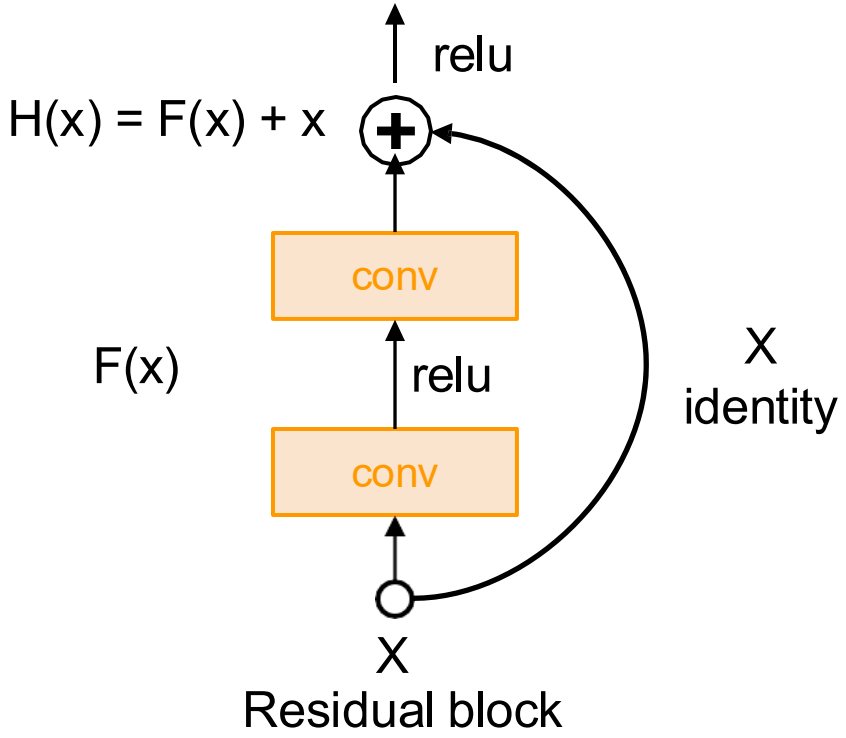


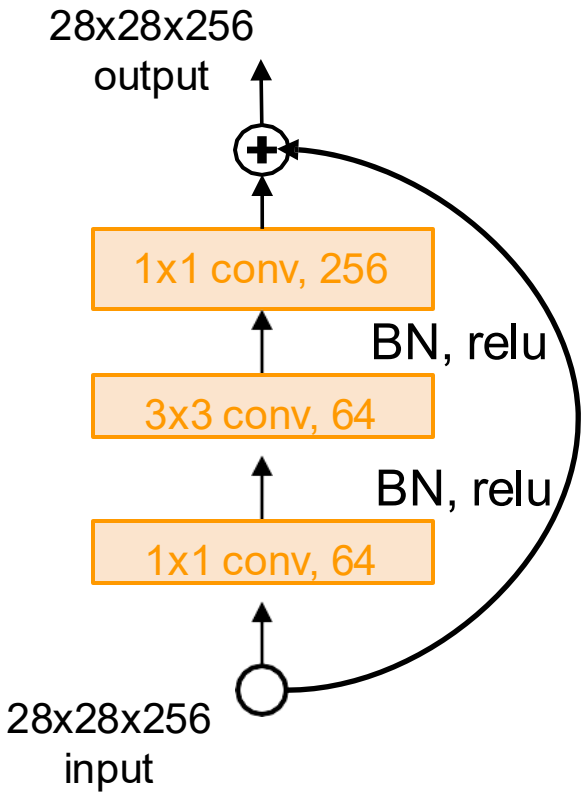


让网络拟合残差 F(x)=H(x)-x，
而不是直接拟合H(x)

ResNet

- 使用 Bottleneck 提升计算效率





ResNet 结构

arXiv
https://arxiv.org › cs

[1512.03385] Deep Residual Learning for Image Recognition

by K He · 2015 · Cited by 239504 — We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

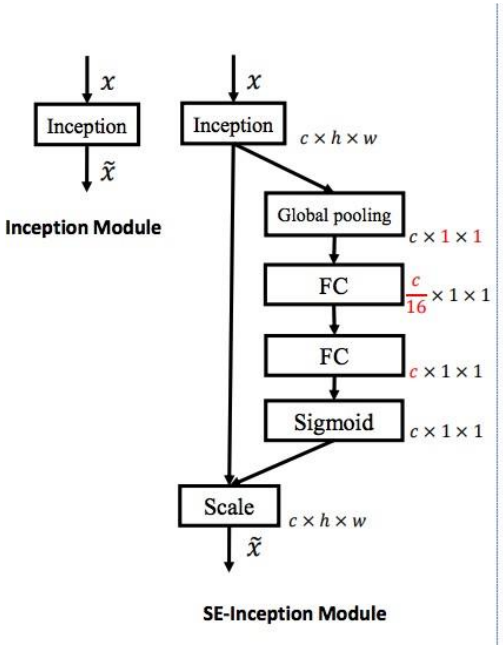
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix}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

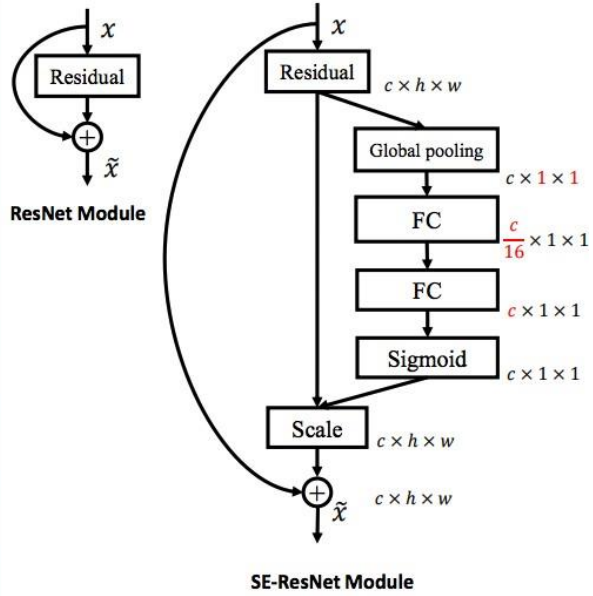
经典卷积网络总结

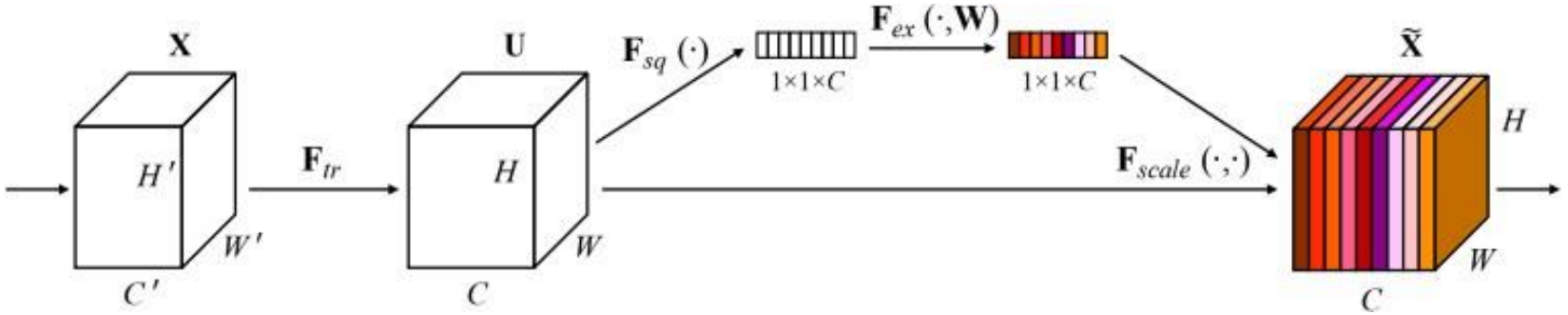
- AlexNet：CNN可以用于计算机视觉任务
- VGG：网络深度的重要性
- GoogLeNet：网络宽度的重要性
- ResNet：训练极深卷积网络，CNN 的表现优于人类！

基于 ResNet 的改进

- Squeeze-and-Excitation Networks (SENet, 2017)
- 为 ResNet 加上注意力
- ILSVRC’17 分类任务冠军
- 引领后续各种注意力
- Transformer 也是基于注意力

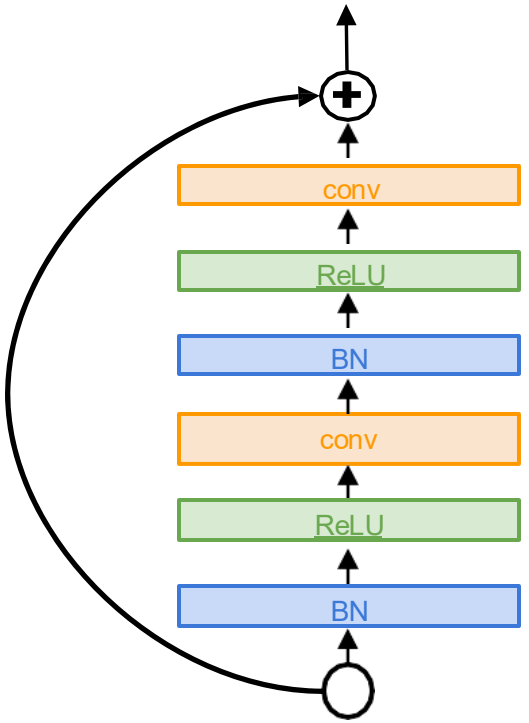






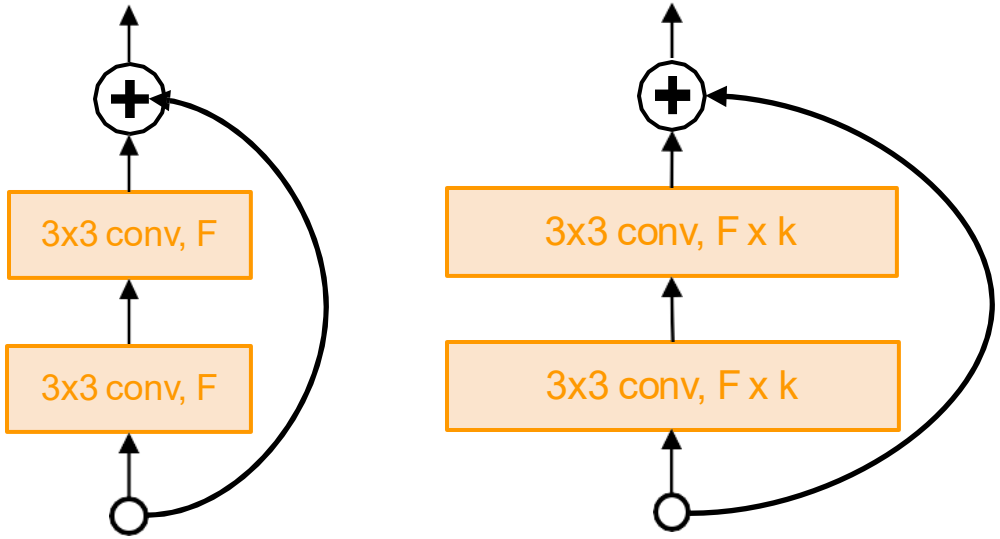
基于 ResNet 的改进

- Identity Mappings in Deep Residual Networks (2016)
- 改进 ResNet 模块设计
- 让网络中的信息更容易传播
- 深度达到1001层



基于 ResNet 的改进

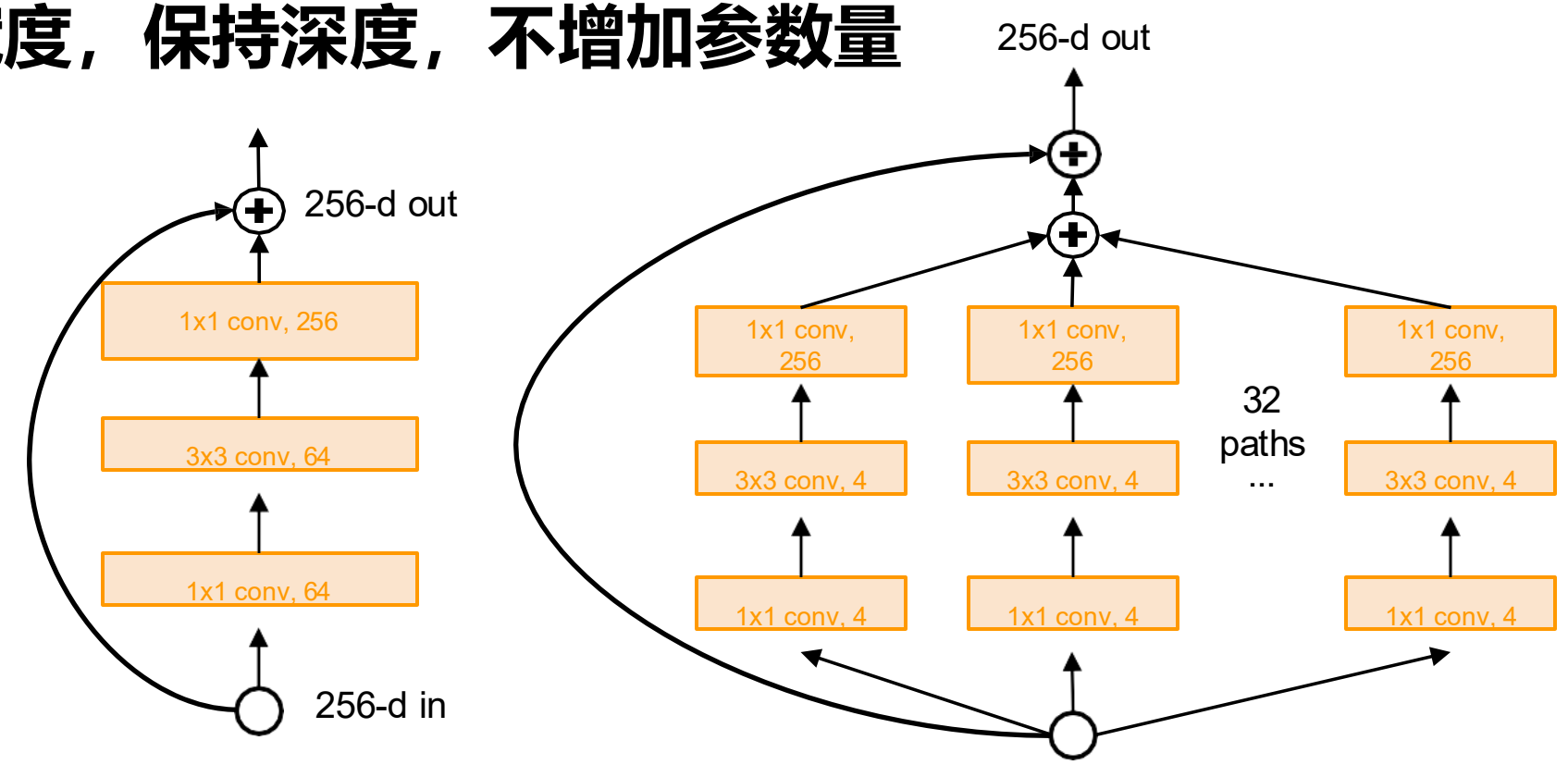
- Wide Residual Networks (2016)
- 增加宽度，减少深度，计算效率更高（可并行化）
- Wide-ResNet-50 性能优于 ResNet-152



Basic residual block　　Wide residual block

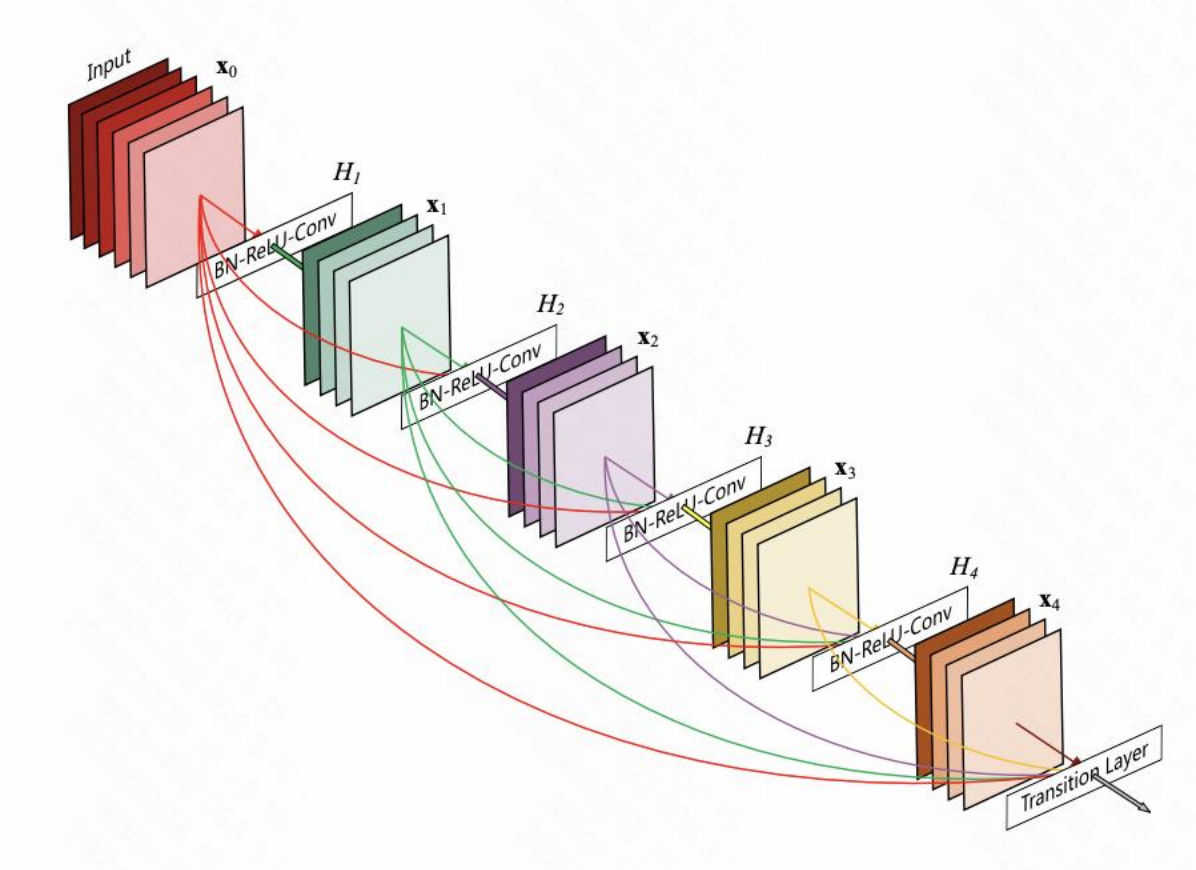
基于 ResNet 的改进

- Aggregated Residual Transformations for Deep Neural Networks (ResNeXt, 2016)
- 增加宽度，保持深度，不增加参数量



基于 ResNet 的改进

- Densely Connected Convolutional Networks (DenseNet, CVPR 2017 Best Paper)
- 密集使用残差模块
- 每一层都与前面所有层连接



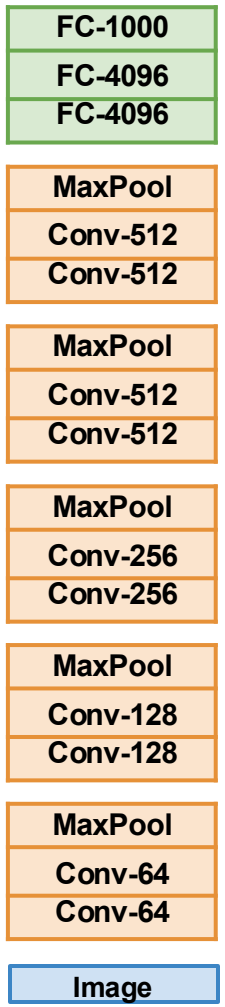
迁移学习

- 将在分类任务上训练好的网络迁移到其他网络
  - 分类任务数据量大，标注难度低，标注成本低
  - 其他任务数据量小，标注难度高，标注成本高
  - 直接在小数据上训练网络，性能很差

迁移学习

南京大学 NANJING UNIVERSITY

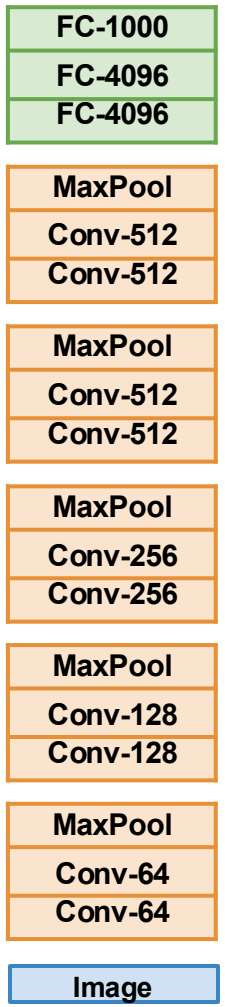
1.在分类任务上训练



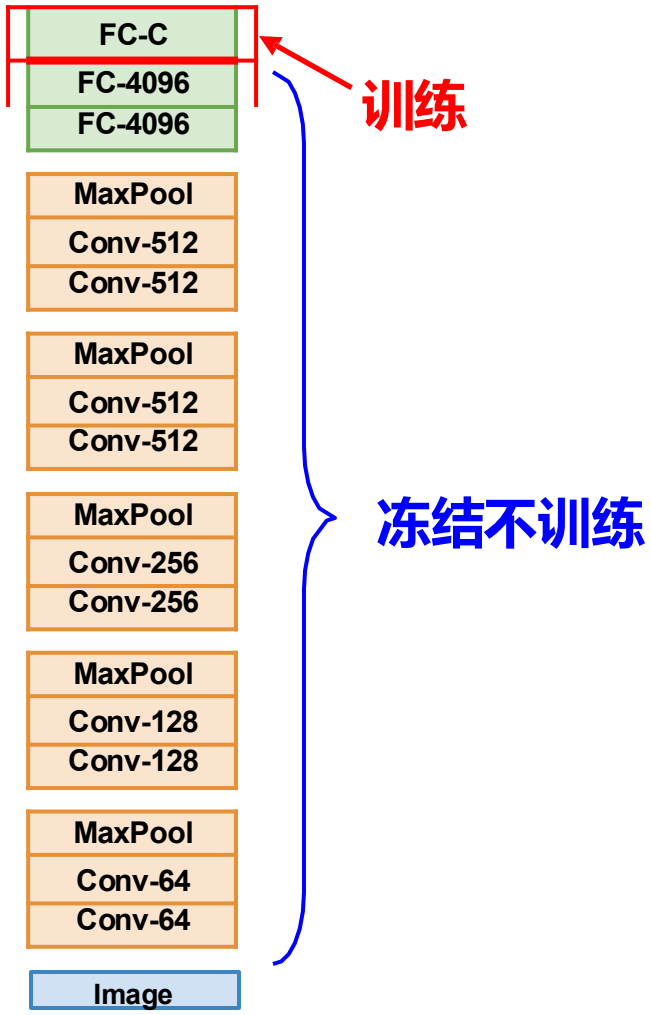
迁移学习

南京大学 NANJING UNIVERSITY

1.在分类任务上训练

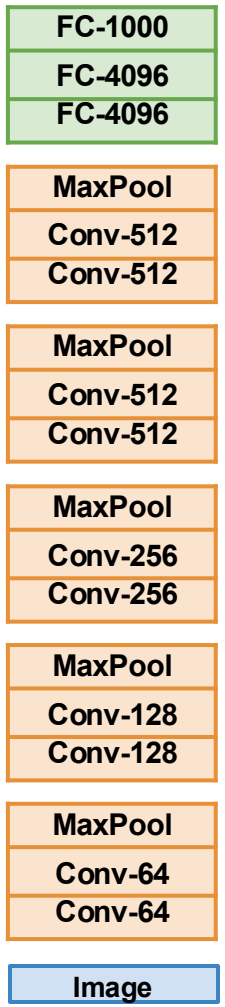


2.小数据集迁移

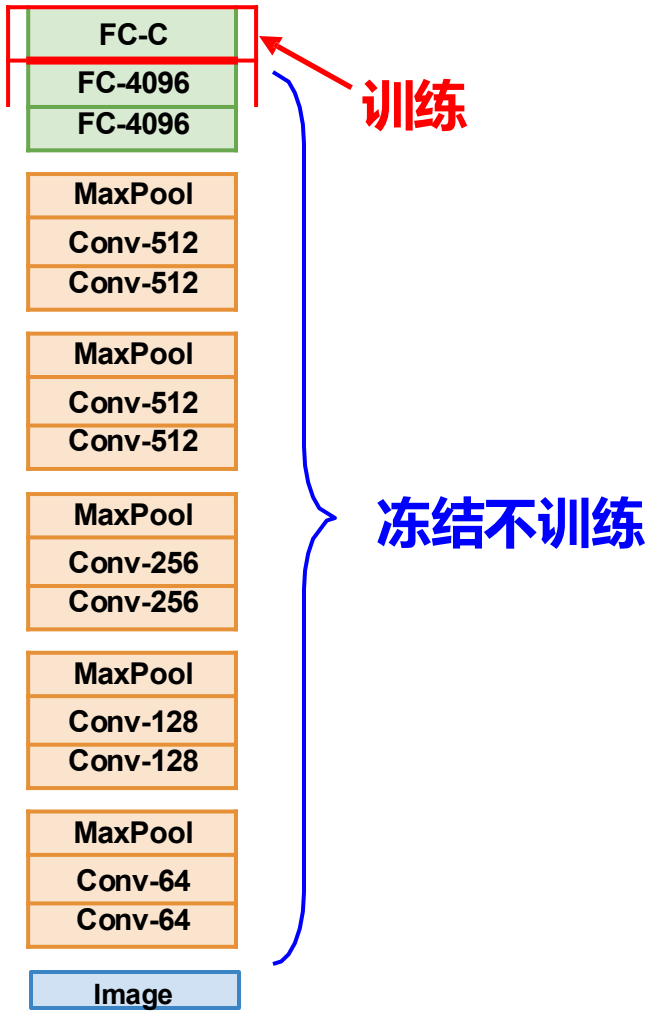


迁移学习

1.在分类任务上训练



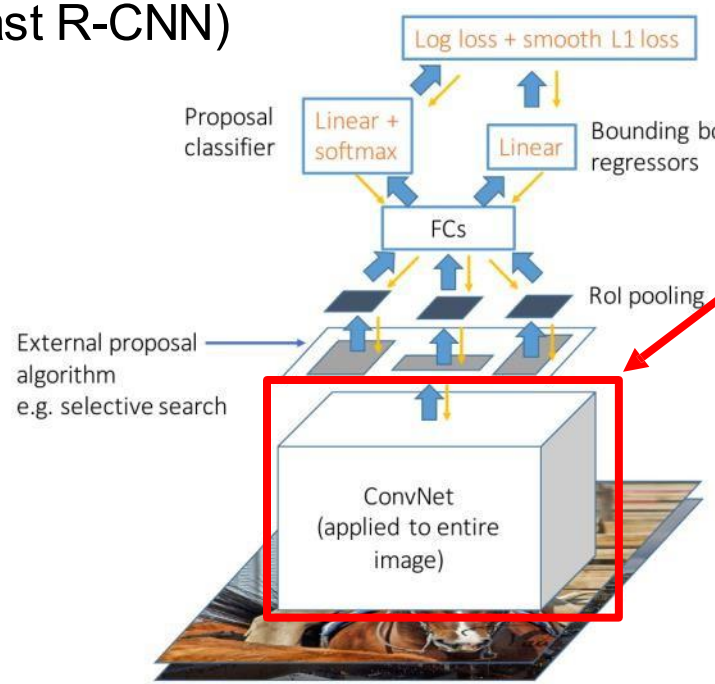
2.小数据集迁移



3.更大数据集迁移

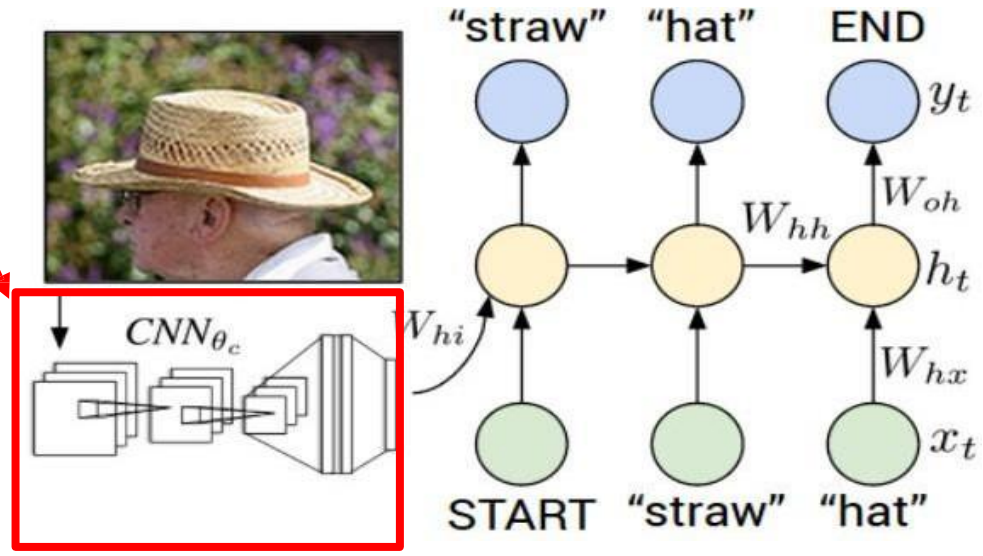
迁移学习

目标检测
(Fast R-CNN)

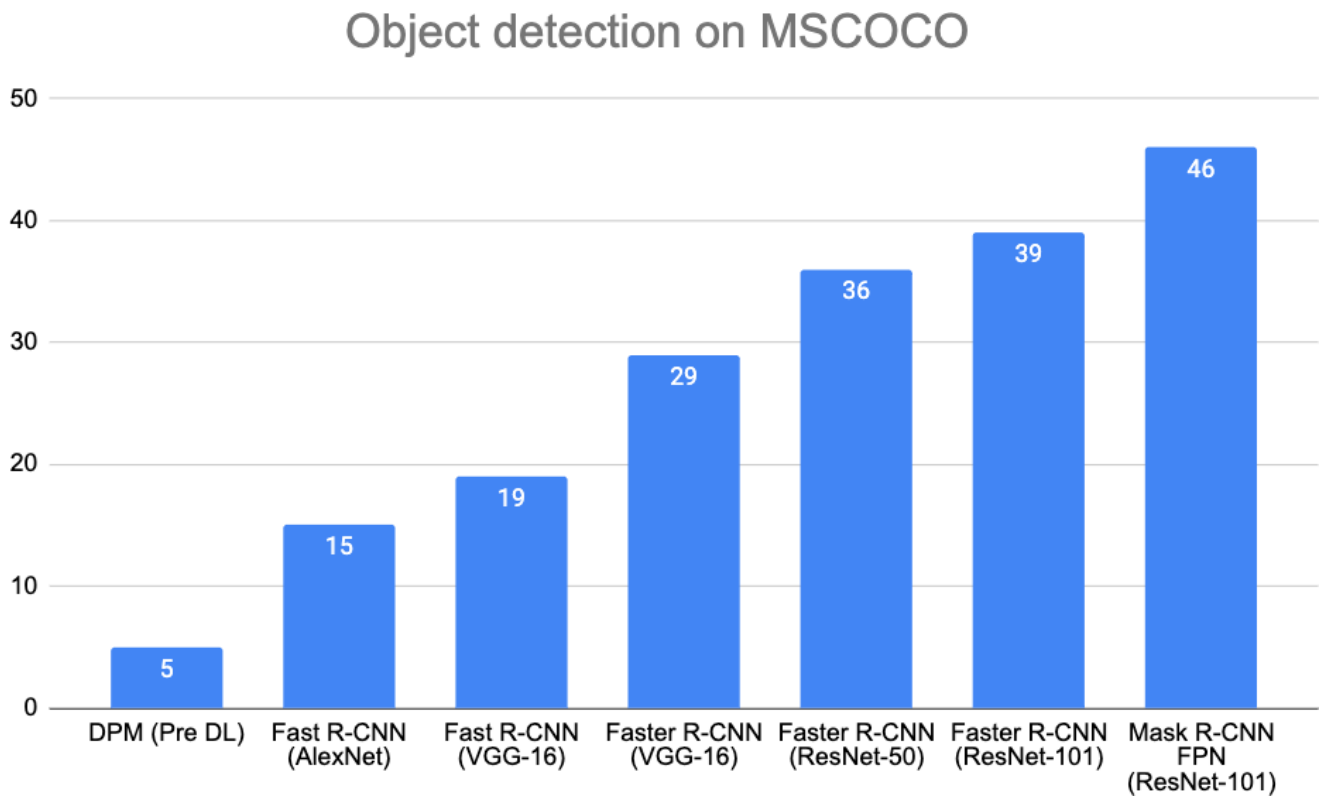


在ImageNet上
预训练的CNN

Image Captioning: CNN + RNN

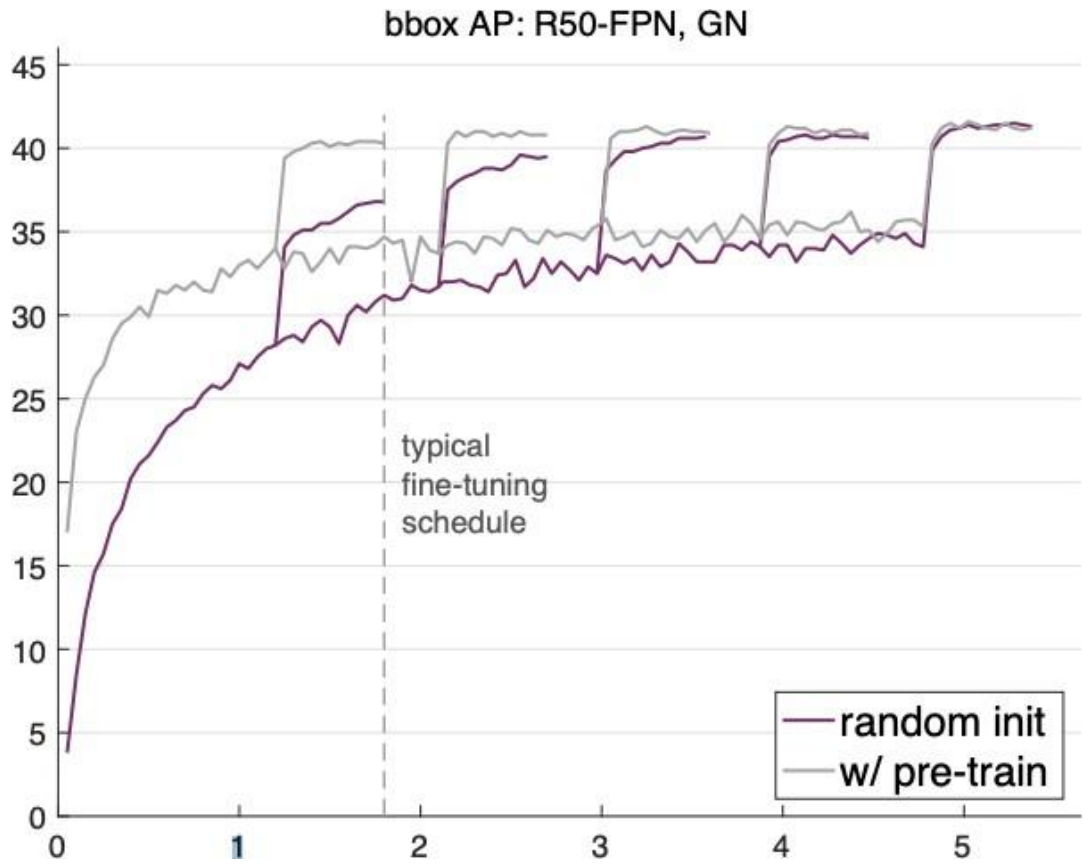


迁移学习-设计更好的模型



迁移学习

- 后续的研究发现预训练并不是必须的
- 更长的训练时长+tricks可以达到相同的性能
- 但是，使用**合适的预训练**模型一定不会得到更差的结果
- **合适的预训练！**



下节课

- 训练卷积神经网络