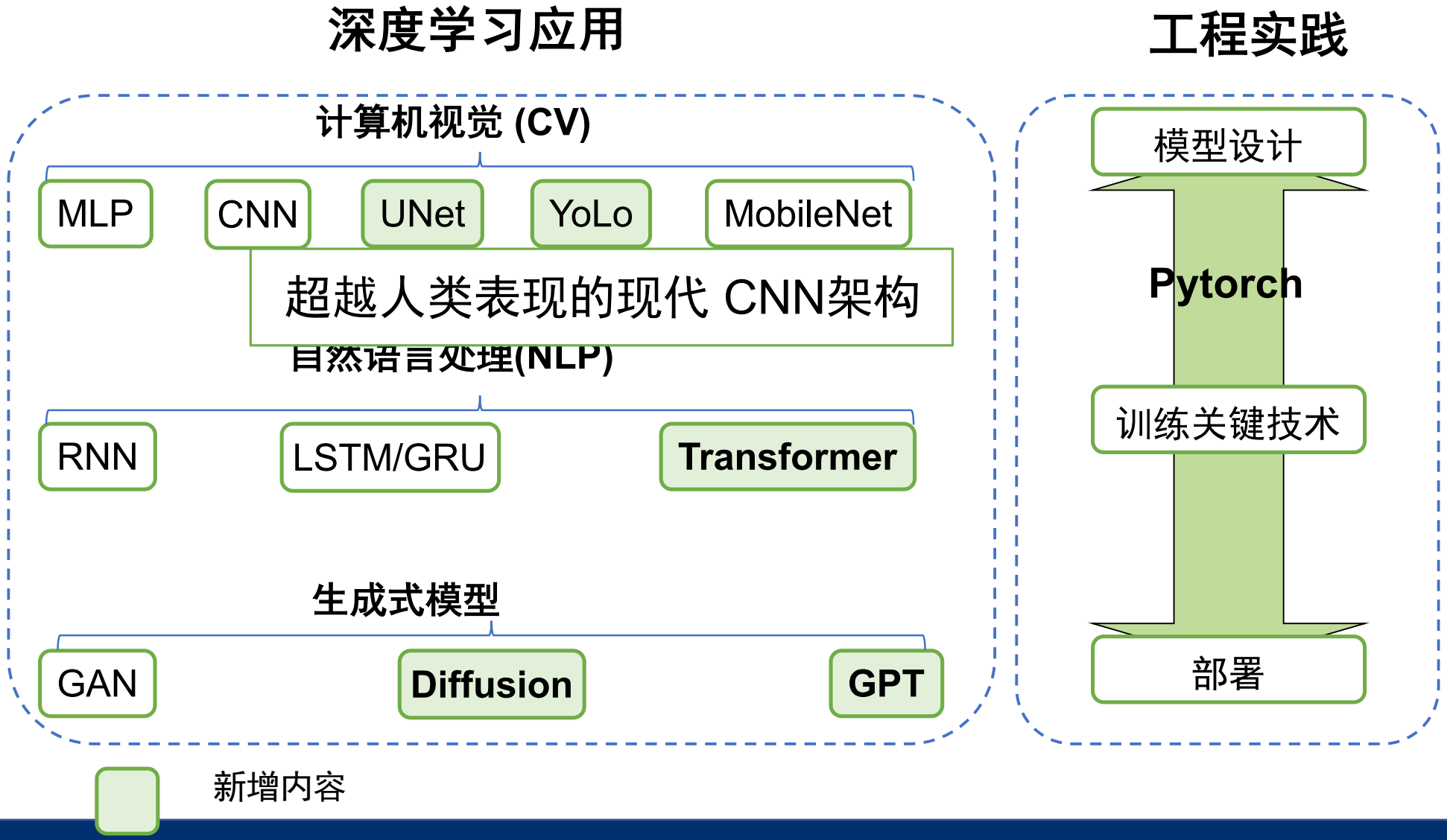
首都师范大学

為學為師求實求新

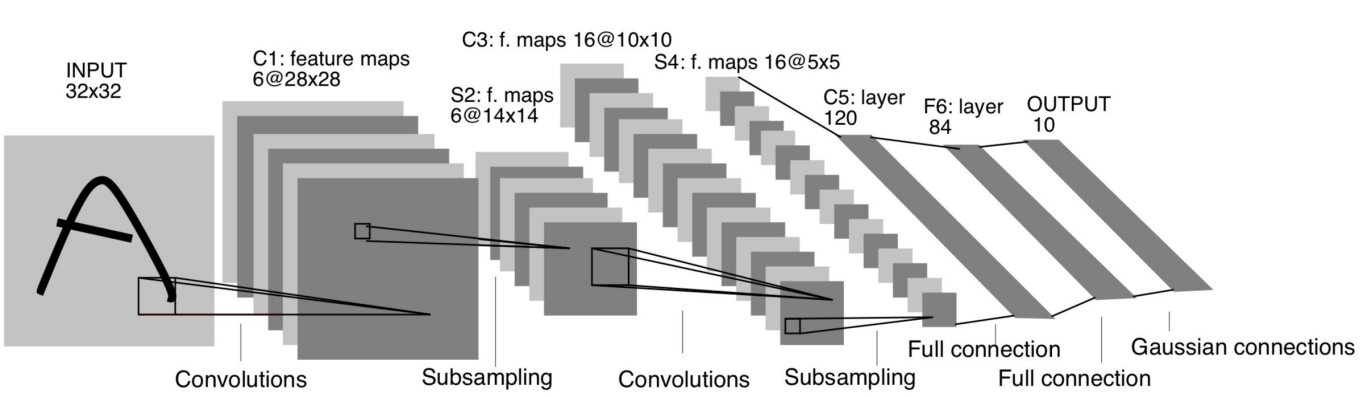
深度学习应用与工程实践
8. CNN 架构
8. CNN Architectures

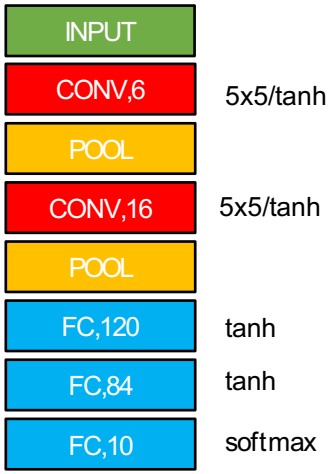
李冰
副研究员，硕士生导师
交叉科学研究院
首都师范大学

本门课的内容


深度学习应用
工程实践
计算机视觉 (CV)
MLP
CNN
UNet
YoLo
MobileNet
超越人类表现的现代 CNN架构
自然语言处理(NLP)
RNN
LSTM/GRU
Transformer
生成式模型
GAN
Diffusion
GPT
模型设计
Pytorch
训练关键技术
部署
新增内容

回顾: LeNet-5





特点:

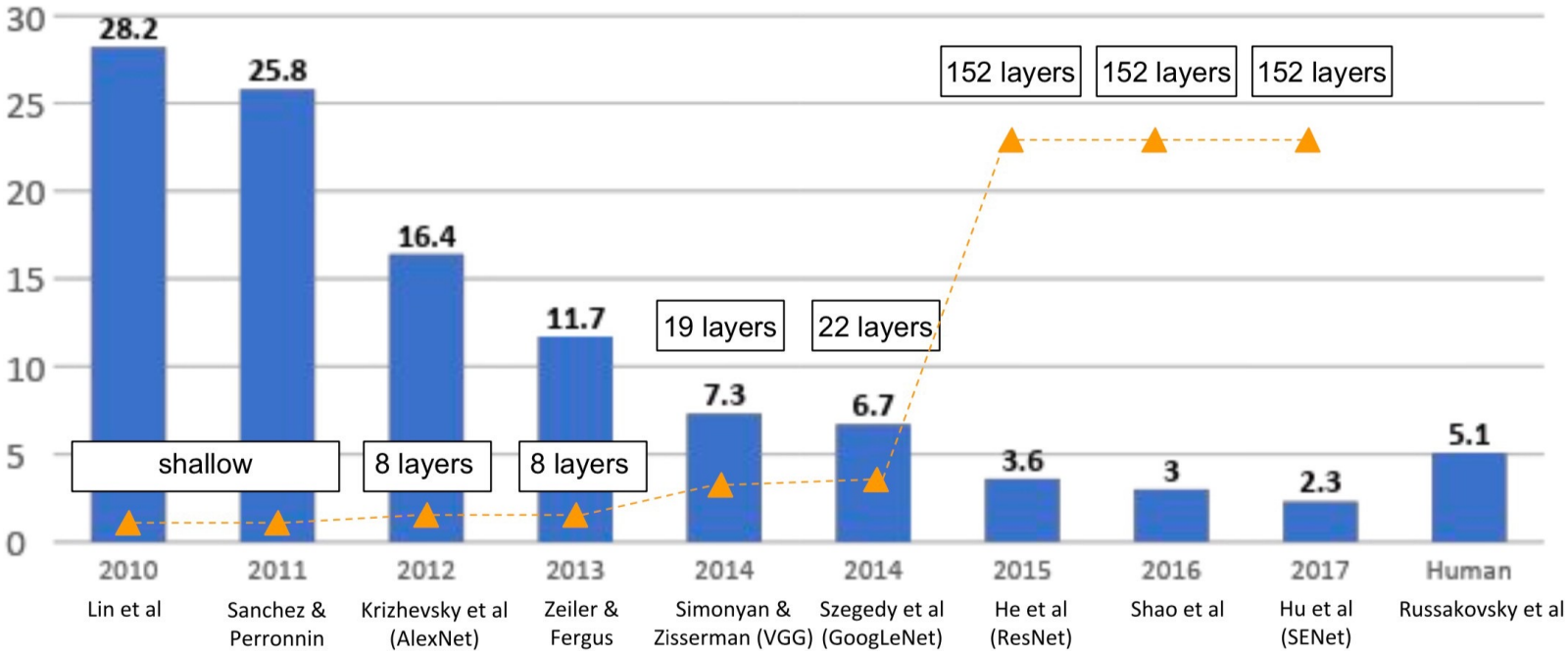
- LeNet-5 第一个有卷积-池化-卷积-池化架构的网络.
- LeNet-5 在MNIST数据集上取得了极佳的结果.
- LeNet-5 激活函数是tanh, 梯度消失问题严重.

CNN 架构

- AlexNet
- VGG
- GoogLeNet (Inception)
- ResNet
- DenseNet

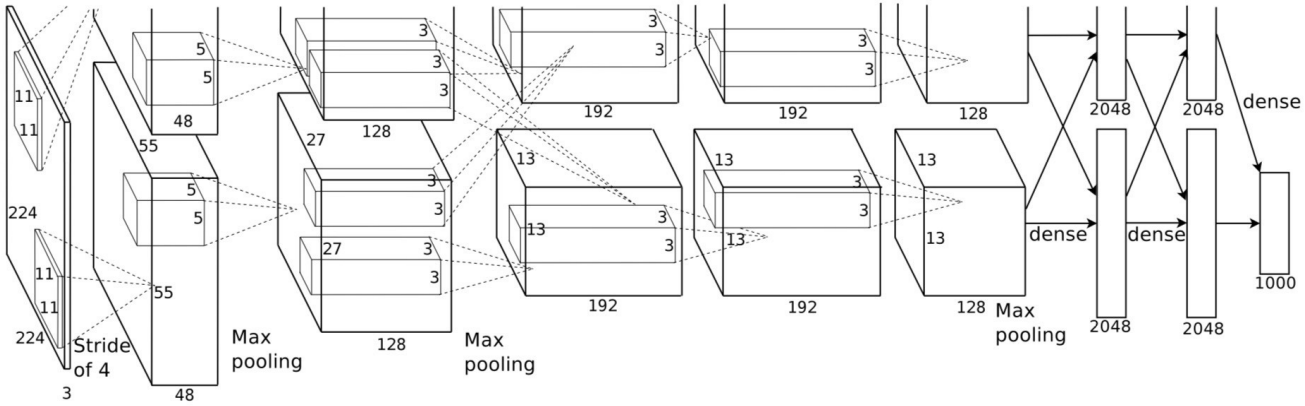
ILSVRC 冠军

- ImageNet Large Scale Visual Recognition Competition (ILSVRC)
- 全球著名的图像识别挑战赛.



AlexNet

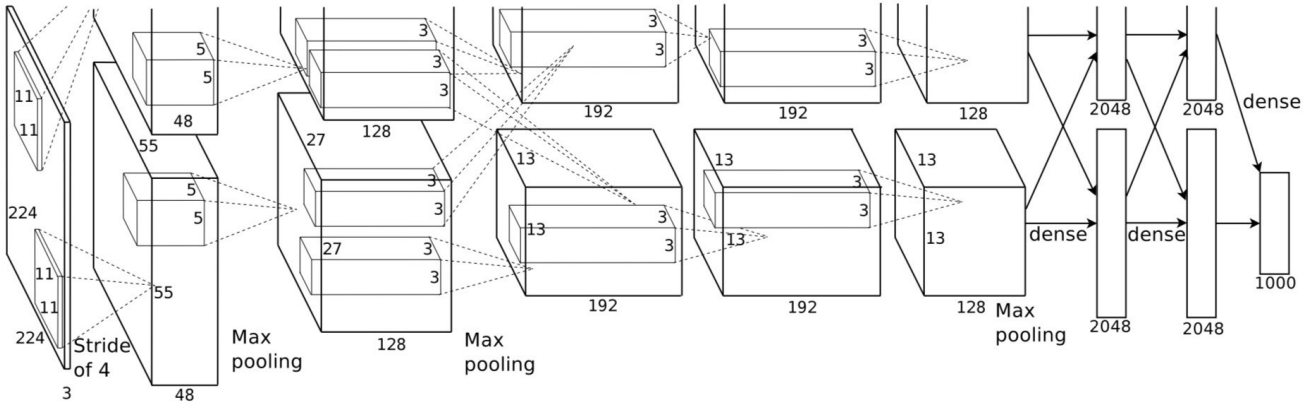
- 第一个在图像识别任务上取得巨大成功的CNN 模型.
- 横跨两个GPU,每个GPU上运行一半的通道。



- **特点**:
  - 卷积核尺寸大(11x11).
  - 第一次采用ReLU作为非线性激活函数.
  - 采用局部响应归一化(LRN)去加速训练.
  - ImageNet top-5 错误率: 16.4%

AlexNet

- 第一个在图像识别任务上取得巨大成功的CNN 模型.
- 横跨两个GPU,每个GPU上运行一半的通道。



- **特点**:
  - 卷积核尺寸大(11x11).
  - 第一次采用ReLU作为非线性激活函数.
  - 采用局部响应归一化(LRN)去加速训练.
  - ImageNet top-5 错误率: 16.4%

第一个卷积层的 MAC数是多少？

$11 \times 11 \times 3 \times 55 \times 55 \times 96 = 105.4M$

AlexNet

- 局部响应归一化(LRN)
  - AlexNet 中某些激活层之后还有LRN层

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

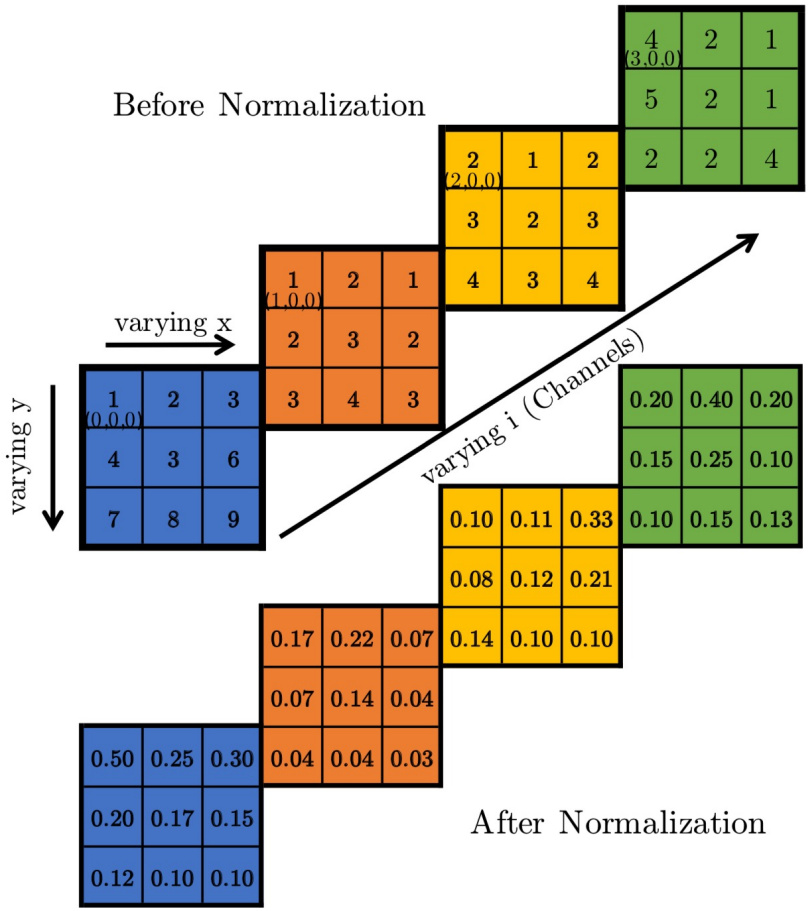
i表示第i个核在位置（x,y ）运用激活函数ReLU后的输出，N是通道总数，n 要归一化的窗口尺寸。
k, n, α, β是超参数

- LRN 归一化这些激活值，能够加速神经网络训练.
- 实践中, AlexNet在验证集上确定了k, n, α, β。采用k = 2, n = 5, α = 10^{-4}, 以及β = 0.75.

AlexNet

- 局部响应归一化(LRN)

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

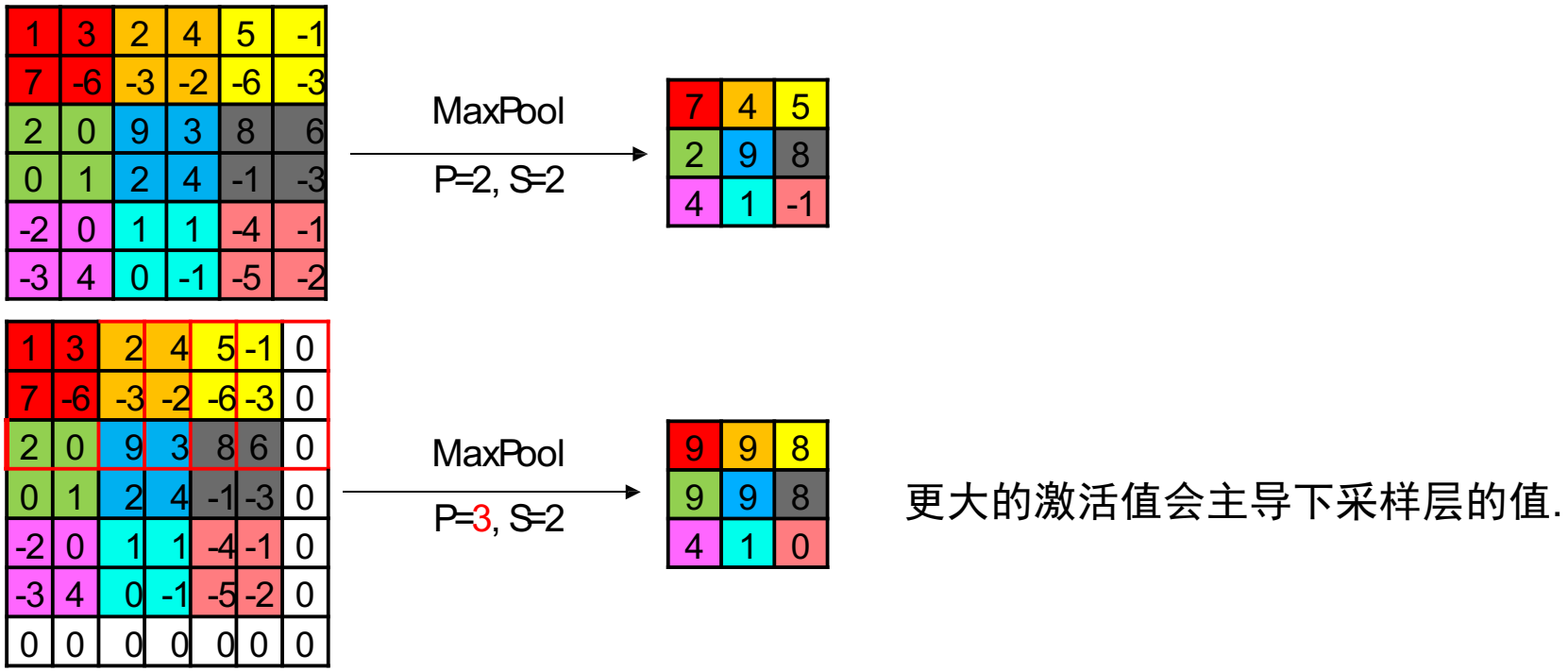


- N = 4, (k,α, β, n)=(0,1,1,2)情况下的例子
- 对于位置*(i,x,y)=（0,0,0）*的元素，*值为1*
- *（i-1, x, y）*不存在，*（i+1, x, y）*的值为1，
- *因此归一化的值=1/(1²+1²) = 0.5*

https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac

AlexNet

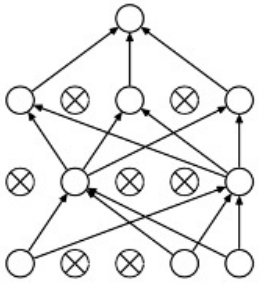
- 重叠池化操作
  - AlexNet 的池化大小比步长大，执行重叠的池化。
  - 考虑了这些池化域里各个值的重要程度，会不容易过拟合



更大的激活值会主导下采样层的值.

AlexNet

- **AlexNet的训练配置**
  - 批量大小128, 初始学习率0.01, 权重衰减5e-4
  - 带动量的SGD，其中动量因子是0.9
  - 用了归一化层LRN
  - FC 层的Dropout 率0.5 (最后一层 FC没有).
  - 数据扩充方法用了很多，移动,翻转等.
- 很多的训练方法都是采用了AlexNet的训练方法.



(b) After applying dropout.

$$v^{t+1} = \mu v_t - \alpha \nabla_{W^t} L(W^t)$$
$$W^{t+1} = W^t + v^{t+1}$$

α: 学习率
μ: 动量因子

VGG

- VGG第一个非常深的网络，(11-19 层)

特点:

- 卷积核大小3x3.
- ImageNet top-5 错误率: 7.3%
- VGG 模型被用在迁移学习中执行其他任务(比如,物体检测).
- 非常大的模型.
  - VGG 16/19
  - 参数量140/144 Million
  - MAC数16G/20G.

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014)

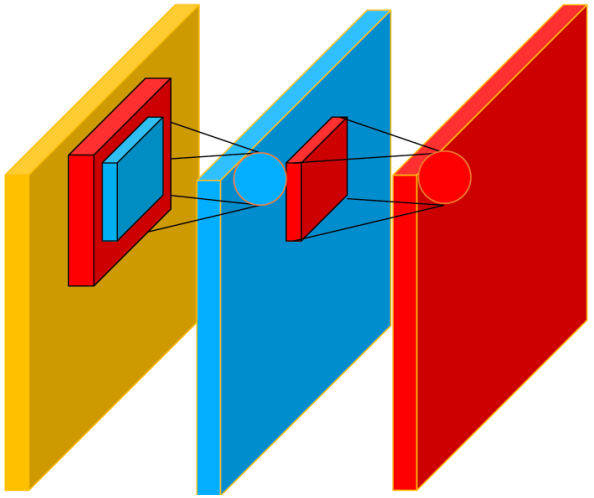
VGG

回顾: 感受野

- 红色神经元的输出受第一个橘色输入特征图中红色区域的影响.
- 这两个卷积核的感受野大小是3+3-1=5.

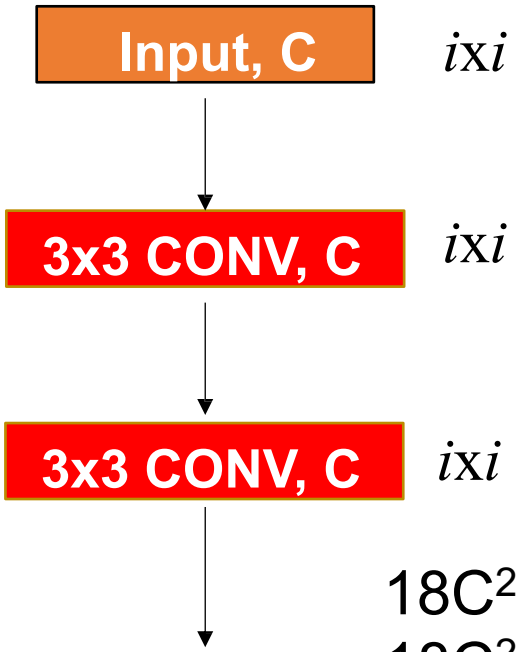
$$l_k = l_{k-1} + (f_k - 1) * \prod_i^{k-1} s_i$$

其中l_{k-1}为第$k-1$层对应的感受野大小;
f_k为第k层的卷积核大小
第k层一个元素的感受野，等价于k-1层fk*fk个感受野的叠加。
k层上前进1个元素相当于k-1层上前进s_k个元素



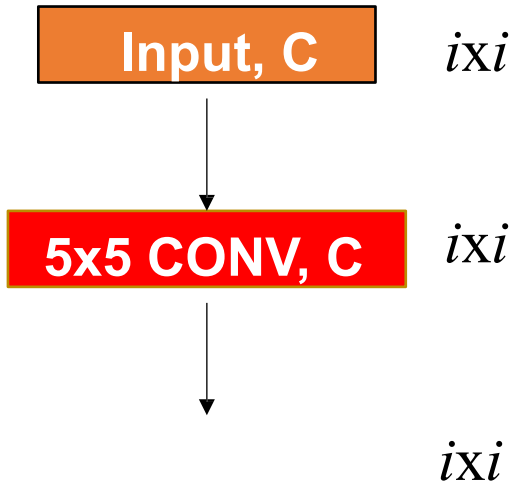
VGG

- VGG 的卷积层只用3x3卷积核.
  - 两个 3x3 卷积与一个5x5卷积有相同的感受野，但是参数和计算量更少.



$18C^2$ 参数量

$18C^2 \times i^2$ MACs

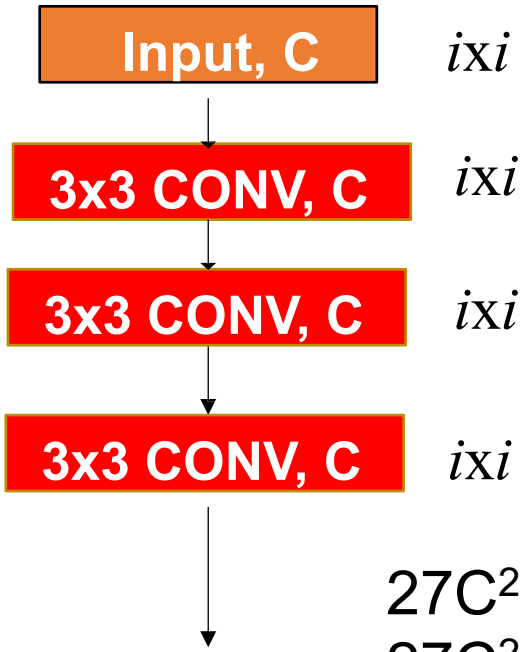


$25C^2$ 参数量

$25C^2 \times i^2$ MACs

VGG

- VGG 的卷积层只用3x3卷积核.
  - 三个 3x3 卷积与一个7x7卷积有相同的感受野，但是参数和计算量更少.



ixi

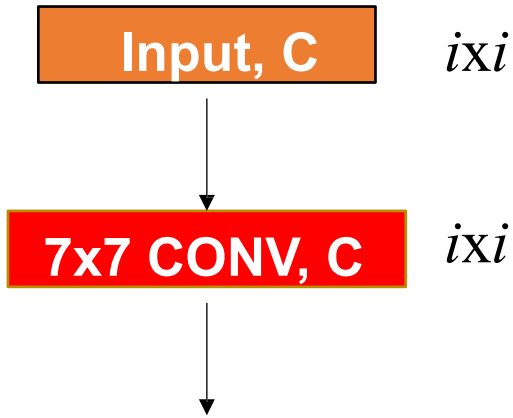
ixi

ixi

ixi

$27C^2$ 参数量

$27C^2$x i^2 MACs



ixi

ixi

ixi

$49C^2$ 参数量

$49C^2$x i^2 MACs

VGG

- **VGG的训练**
  - 批量大小是256, 初始学习率0.01, 权重衰减5e-4.
    - 如果验证准确率不变，学习率按0.1衰减.
  - 带动量的SGD，其中动量因子是0.9.
  - FC 层的Dropout 率0.5 (最后一层 FC没有).
  - 数据预处理的方法与AlexNet不同（详见论文）
    - 比如multi-crop evaluation, 就是把一张大图随机截取不同位置的 小图 然后送进模型预测
  - 模型集合的方法取得更好的结果.

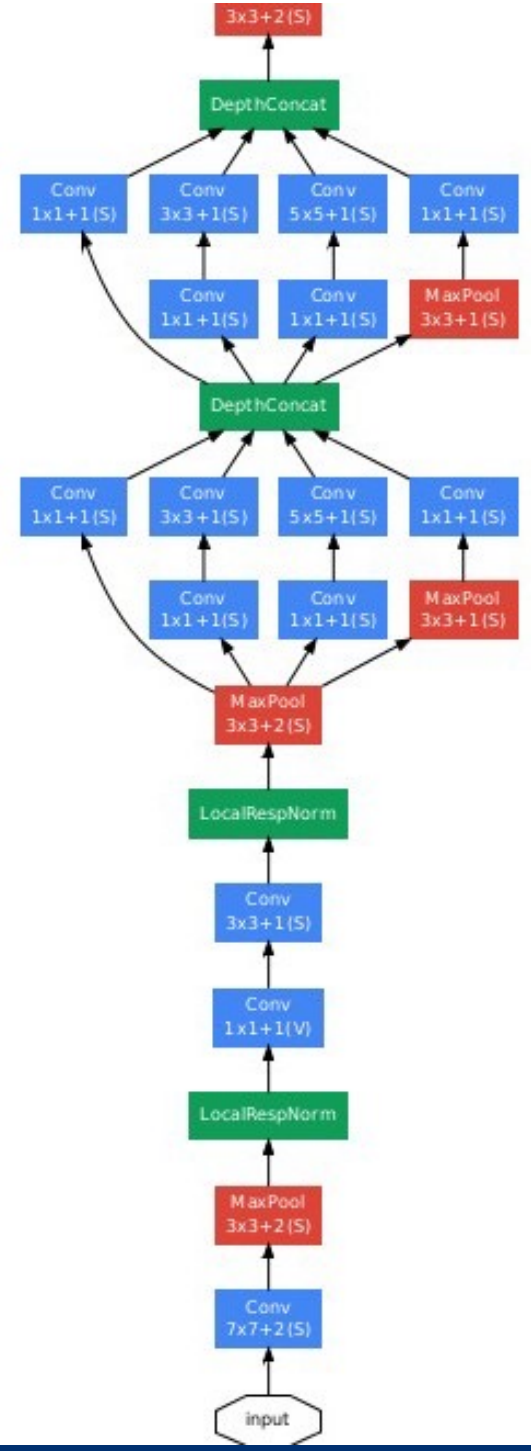
- torch.optim.lr_scheduler
- 提供了许多调节学习率的方法
- torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1, patience=10, threshold=0.0001, threshold_mode='rel', cooldown=0, min_lr=0, eps=1e-08, verbose=False)

GoogLeNet (Inception)

- GoogLeNet开始，网络不止更深(22层)，而且进化出多分支结构.

特征:

- 使用多种卷积核（3x3、5x5、1x1）构建更深的网络，擅长提取多维信息
- 用局部响应归一化（LRN）来加速训练
- ImageNet top-5 错误率: 6.7%
- 使用瓶颈结构(Bottleneck structure)来减少参数.
- 耗费内存.
- 训练时间长，花费一周才能收敛.

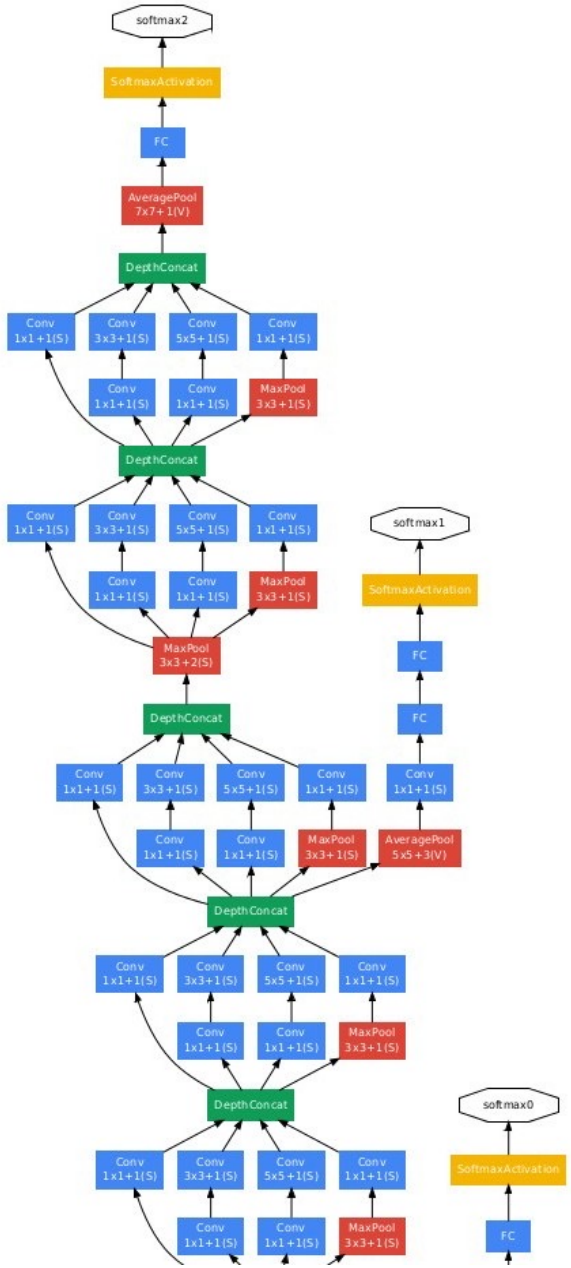


GoogLeNet (Inception)

- GoogLeNet开始，网络不止更深(22层)，而且进化出多分支结构.

特征:

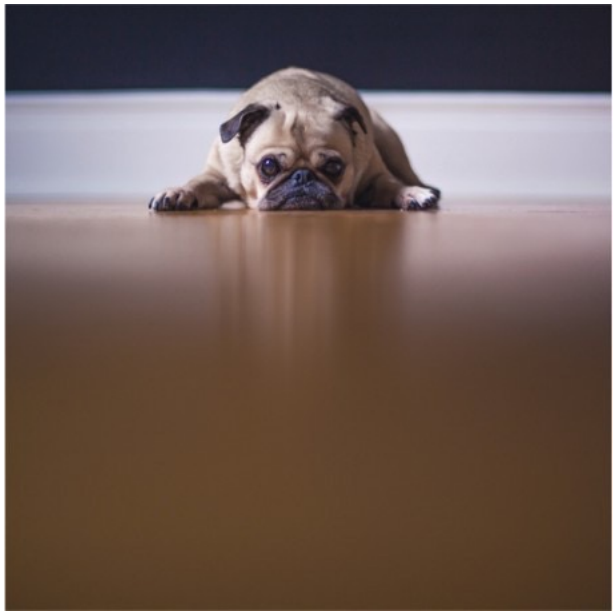
- 使用多种卷积核（3x3、5x5、1x1）构建更深的网络，擅长提取多维信息
- 用局部响应归一化（LRN）来加速训练
- ImageNet top-5 错误率: 6.7%
- 使用瓶颈结构(Bottleneck structure)来减少参数.
- 耗费内存.
- 训练时间长，花费一周才能收敛.



GoogLeNet

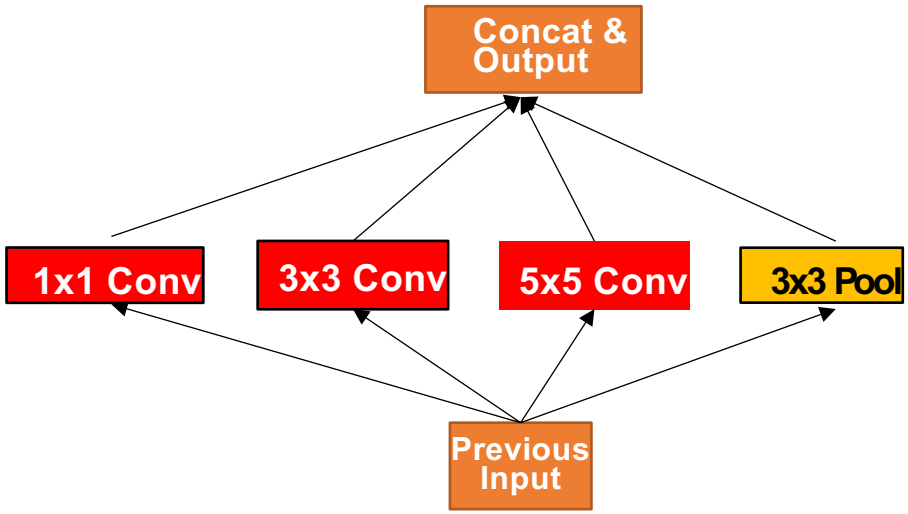
- 输入中物体位置不同，选择合适的卷积核大小是很困难的



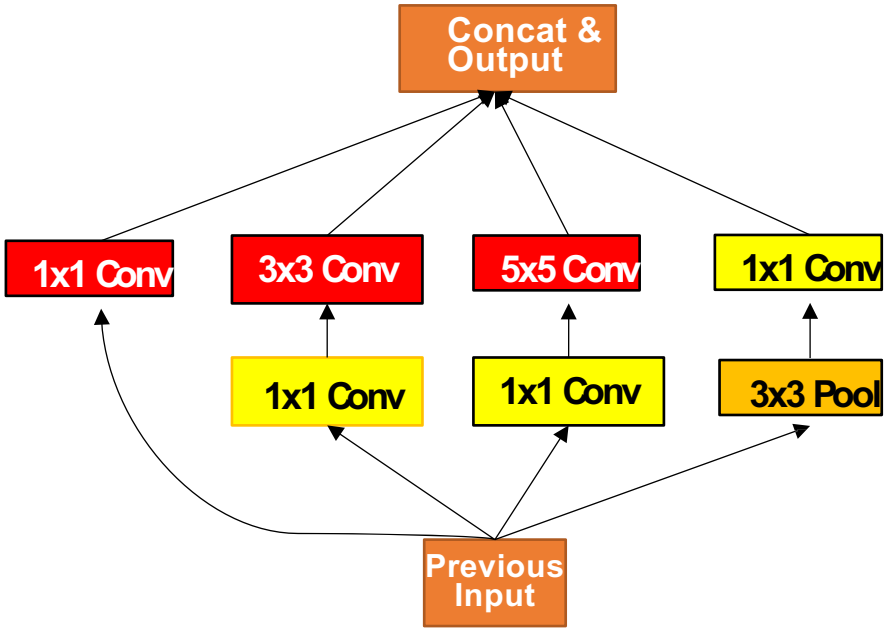


GoogLeNet

- 瓶颈结构
  - GoogLeNet 里的瓶颈结构



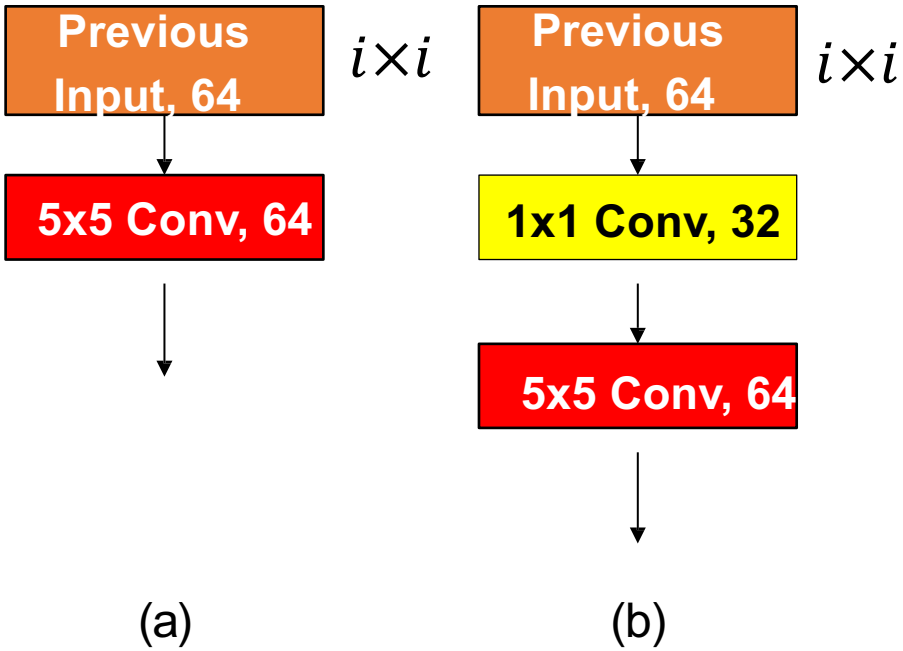
Naïve GoogLeNet



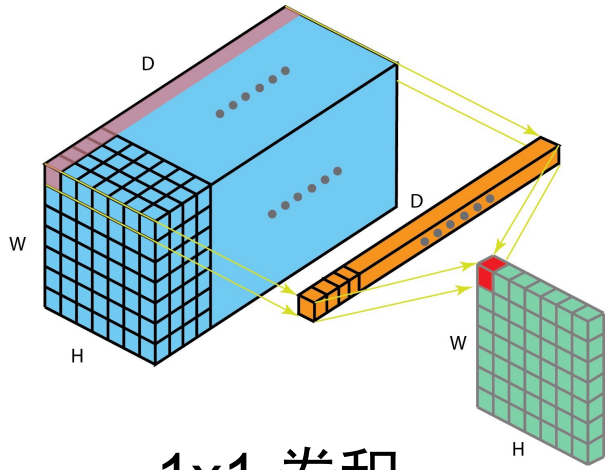
维度下降的GoogLeNet

GoogLeNet

- 瓶颈结构
  - 卷积核的数目减少后和更大的卷积核（比如5x5）进行卷积.



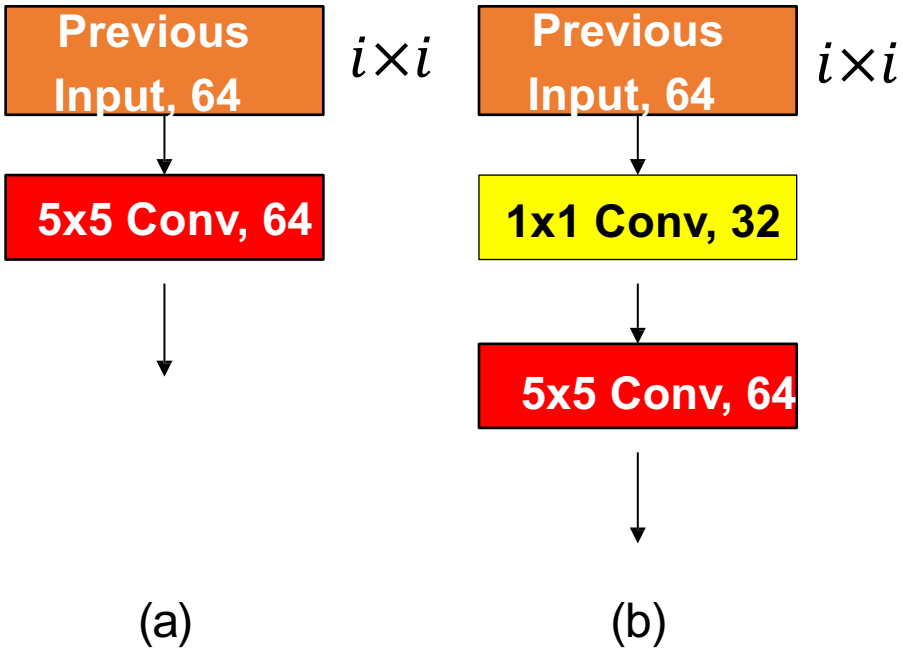
Q: (a)和(b)的结构，计算它们各自的权重数和乘法次数.



1x1 卷积

GoogLeNet

- **瓶颈结构**
  - 卷积核的数目减少后和更大的卷积核（比如5x5）进行卷积.



Q: (a)和(b)的结构，计算它们各自的权重数和乘法次数.

A:

(a)

$5 \times 5 \times 64 \times 64 + 64$

=102464权重数

$5 \times 5 \times 64 \times 64 \times i \times i$

=10240 i^2 MACs

(b)

$1 \times 1 \times 64 \times 32 + 32$

$+5 \times 5 \times 32 \times 64 + 64$

$=$ 53344权重数

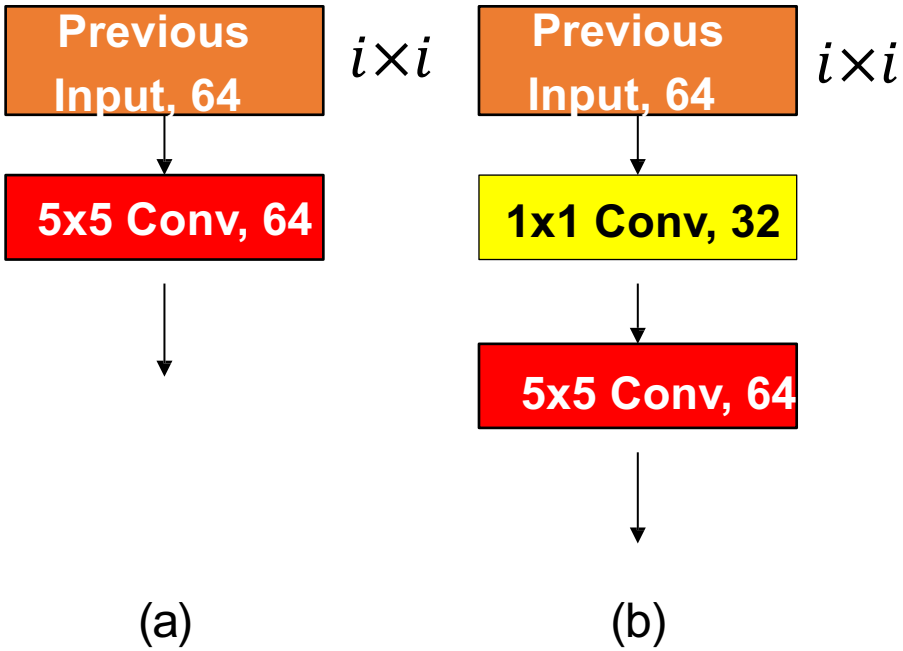
$(1 \times 1 \times 64 \times 32 + 5 \times 5 \times 32 \times 64) \times i \times i$

= 53248 i^2 MACs

Szegedy, Christian, et al. “Going deeper with convolutions.’ (2015)

GoogLeNet

- 瓶颈结构减少了特征图的数目，造成信息的丢失。
- 但不会导致性能的下降，为什么?



- 网络的信息容量并不总是和网络的宽度(通道数/特征图数目)成正比.
- 高维输入投射到低维空间是能够保留有效信息。
- 这种投射，是能够起到正则化的作用，防止过拟合，进而提高了GoogLeNet的性能。

Szegedy, Christian, et al. “Going deeper with convolutions.’ (2015)

GoogLeNet

- **GoogLeNet的训练**
  - 批量大小是256, 初始学习率0.01, 学习率每8个训练周期后衰减0.96
  - 40% dropout 率（最后一层FC没有）.
  - 带动量的SGD，动量因子 0.9.
  - 数据预处理与VGG不同， 详见（GoogLeNet-V1 (V2)论文
    - 增加了亮度对比度调整以及随机扭曲图片的比例

GoogLeNet-V2 (Inception-V2)

- 非线性激活前增加批标准化

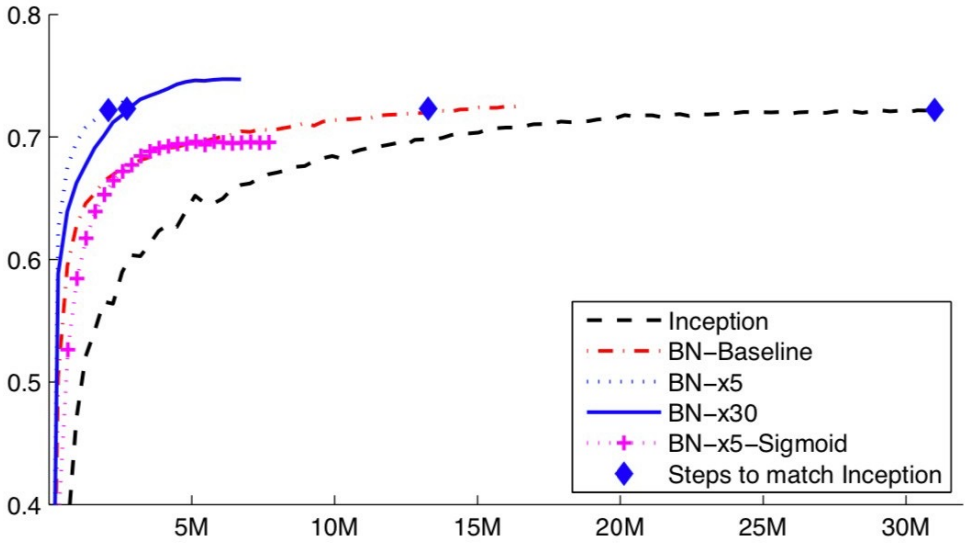
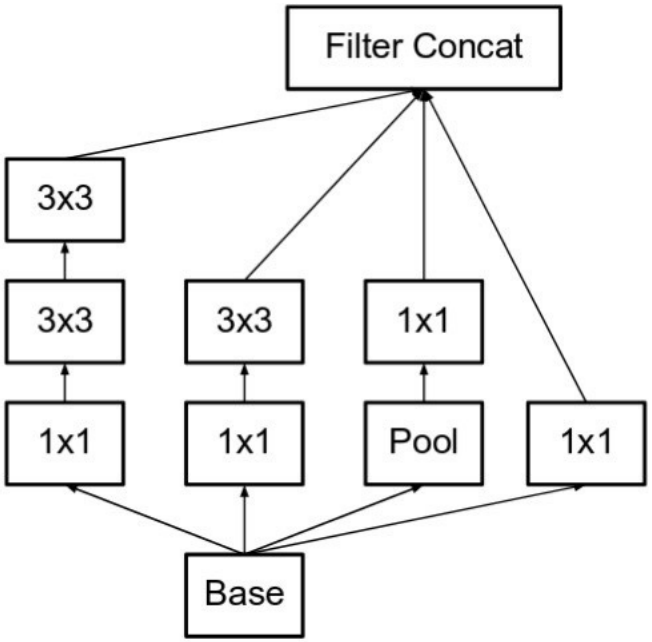


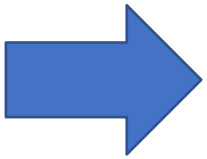
Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

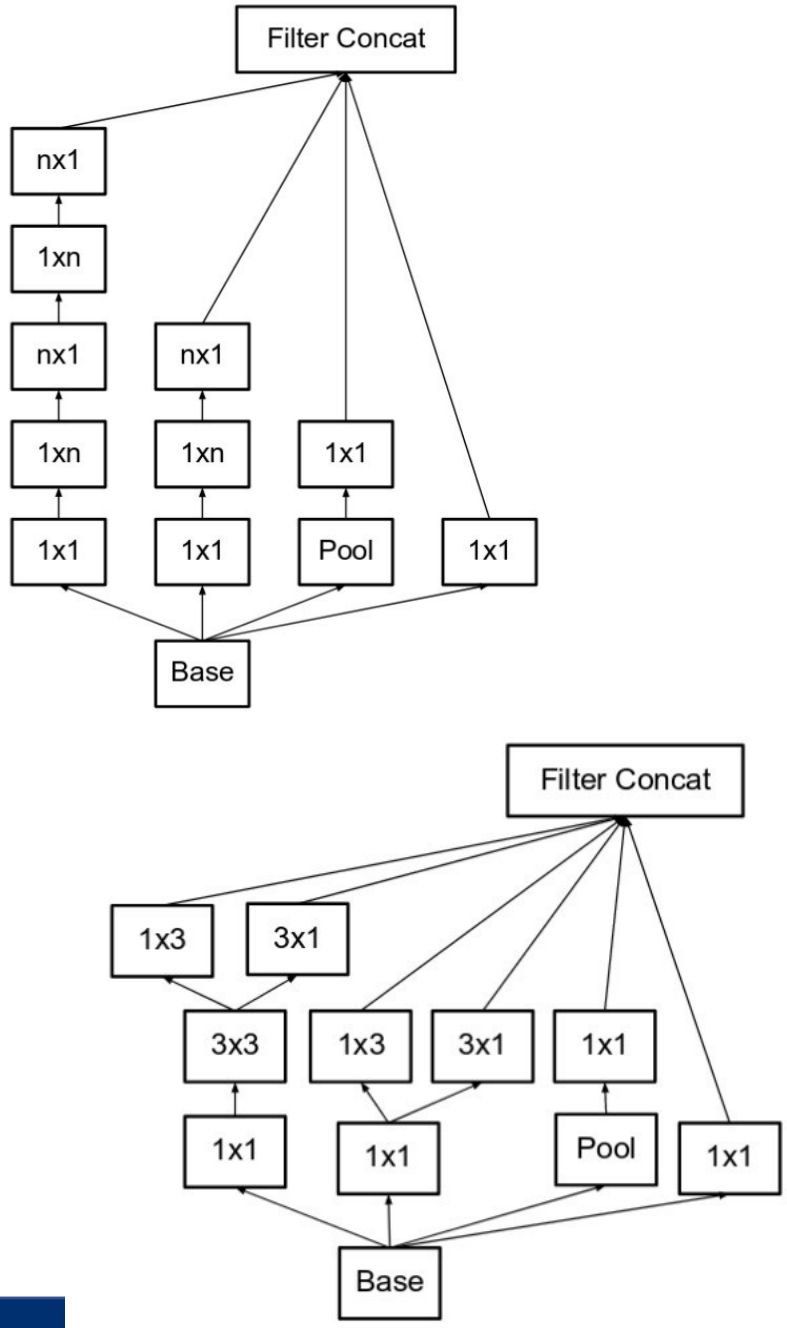
特点:

- 移除了局部响应归一化.
- GoogLeNet-V2 比V1训练更快.
- ImageNet top-5 错误率:4.9%

GoogLeNet-V2







GoogLeNet-V3

- 相比V2,新增
  - RMSProp优化方法.
  - 7x7卷积分解.
  - 批标准化.
  - Label Smoothing
- 相比于V2，top-5 错误率下降3%

$$G^{t+1} = \mu G^t + (1-\mu)\left(\nabla_W L(W^t)\right)^2$$

$$W^{t+1} = W^t - \frac{\alpha \nabla_W L(W^t)}{\sqrt{G^{t+1} + \epsilon}}$$

Input: values of x over a mini-batch.
Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i = \gamma \hat{x}_i + \beta$$

μ_B: 当前批次的平均值
σ_B^2: 当前批次的方差
γ: 可训练的scale参数
β: 可训练的偏差值.
ϵ: 很小的值避免零除. 小数据集设为1e-5 ，大数据集是1e-3

Label smoothing: 原始的标签

$$y_{label} = [0,1,0,0,0,\dots]$$

替换为

$$y_{label} = \left[\frac{\epsilon}{K-1}, 1-\epsilon, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1}, \frac{\epsilon}{K-1}, \dots\right]$$

K: 类别数
ϵ: 标签平滑因子.

CNN 架构

- AlexNet
- VGG
- GoogLeNet (Inception)
- ResNet
- DenseNet

ResNet

- 即使有批标准化, 非常深的神经网络.

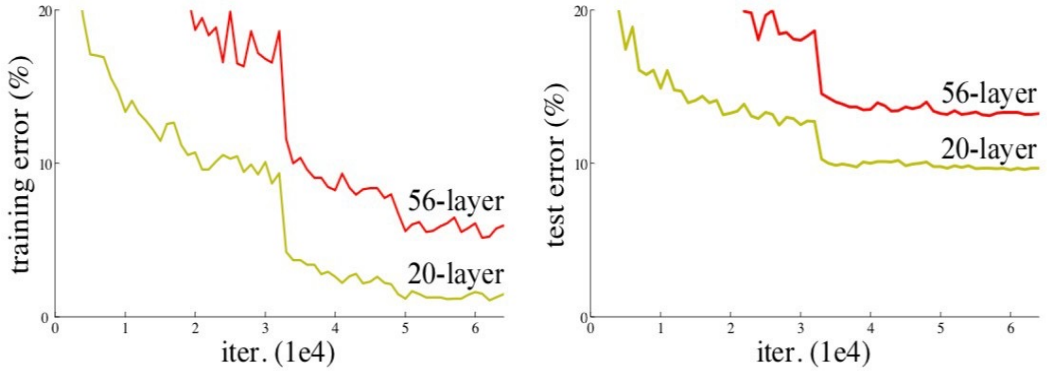
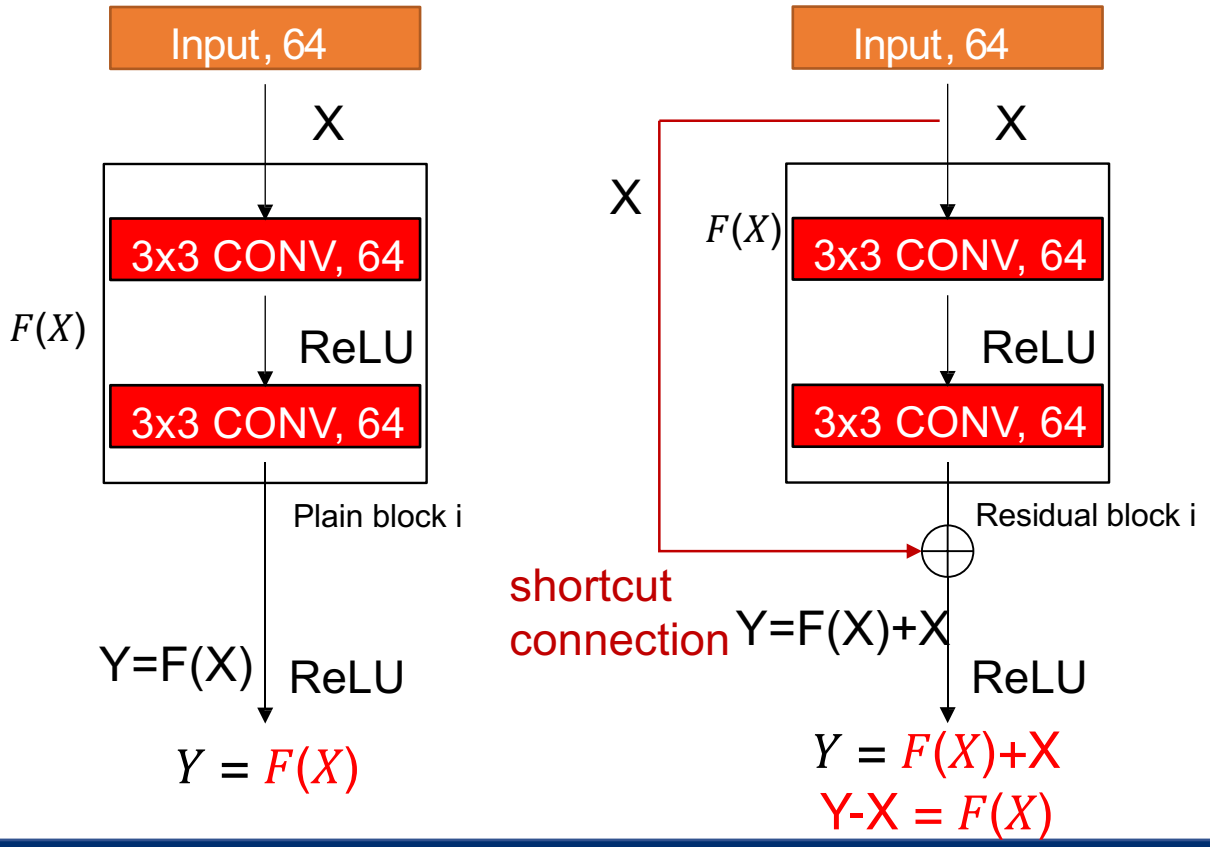


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- 随着网络层数和参数量增大， 网络优化变得更困难
- 梯度爆炸/消失更容易发生.

ResNet

- DNN 训练时, 我们是学习函数F(X)=Y.
- ResNet 增加了一个近路连接来拟合F(X)=Y-X.

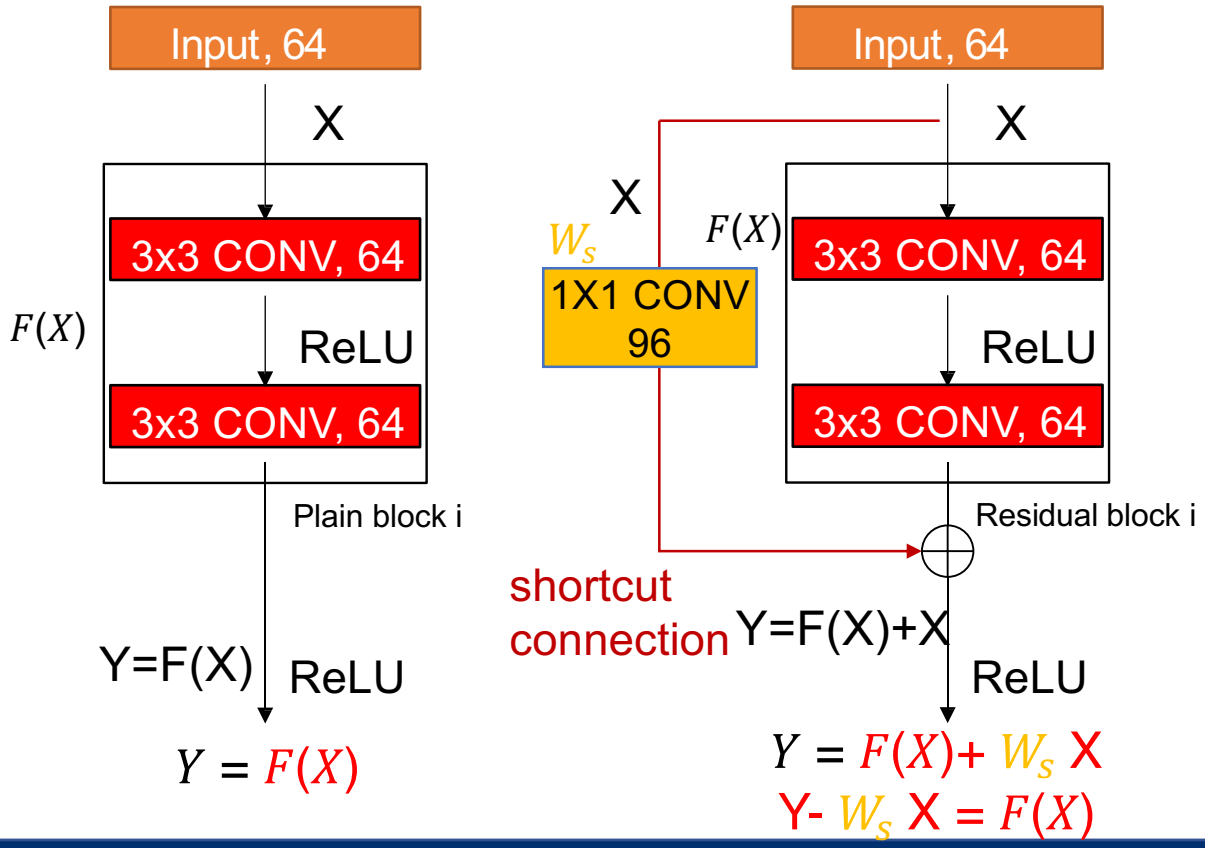


$$Y = F(x, \{W_i\}) + X$$

残差映射　恒等映射

ResNet

- 经过F(X)之后Y中的输出**卷积核数目**变化怎么办?
- Solution: 用**1x1 卷积**来保证尺寸匹配.



$$Y = F(x, \{W_i\}) + W_s\, X$$

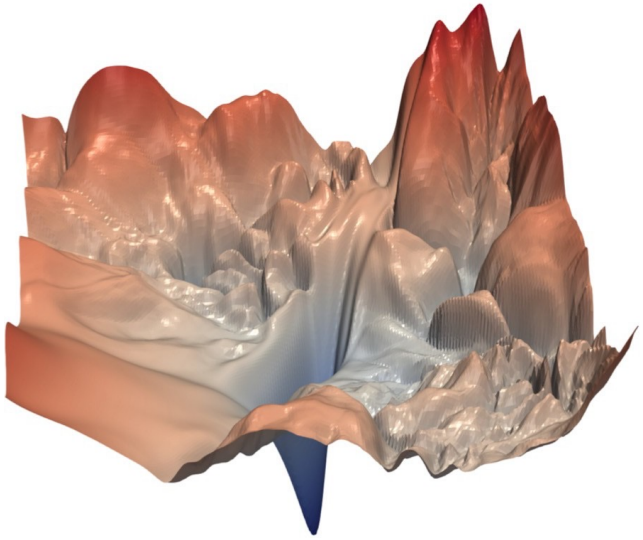
原始映射　残差映射

$$W_s \in R^{1\times1\times64\times96}$$

ResNet

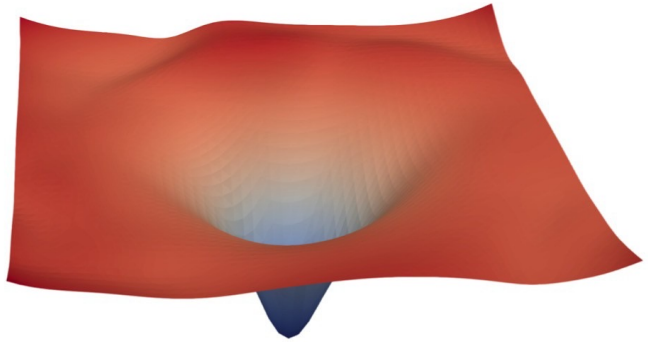
- 可视化
  - 残余学习如何简化训练的？

噪声，容易陷入本地最优点



没有残差连接的损失表面

平滑，容易优化

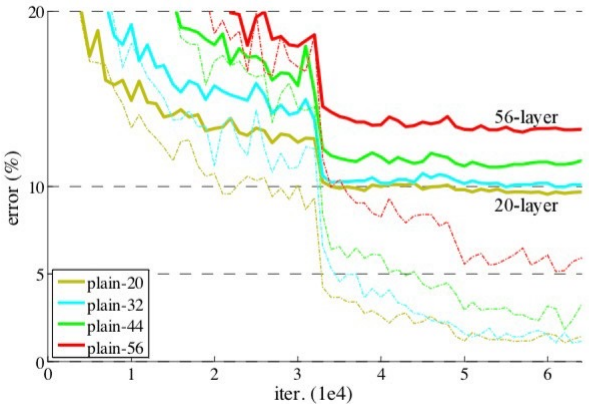


有残差连接的损失表面

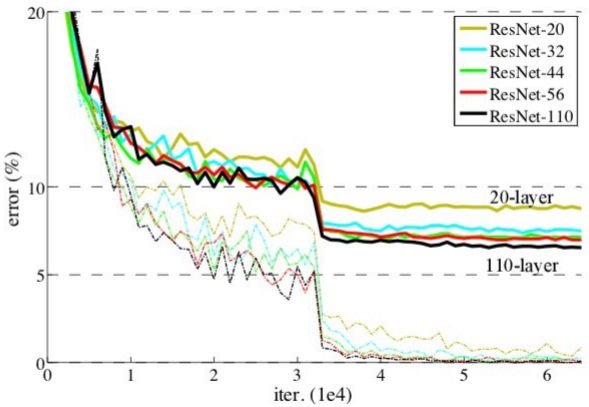
残差结构使损失平面更平滑，更易于优化

ResNet

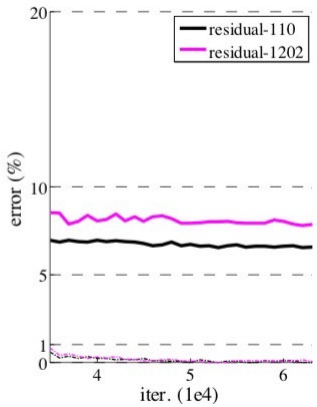
- 可视化
  - ResNet训练时loss如何变化的？



普通网络



ResNet

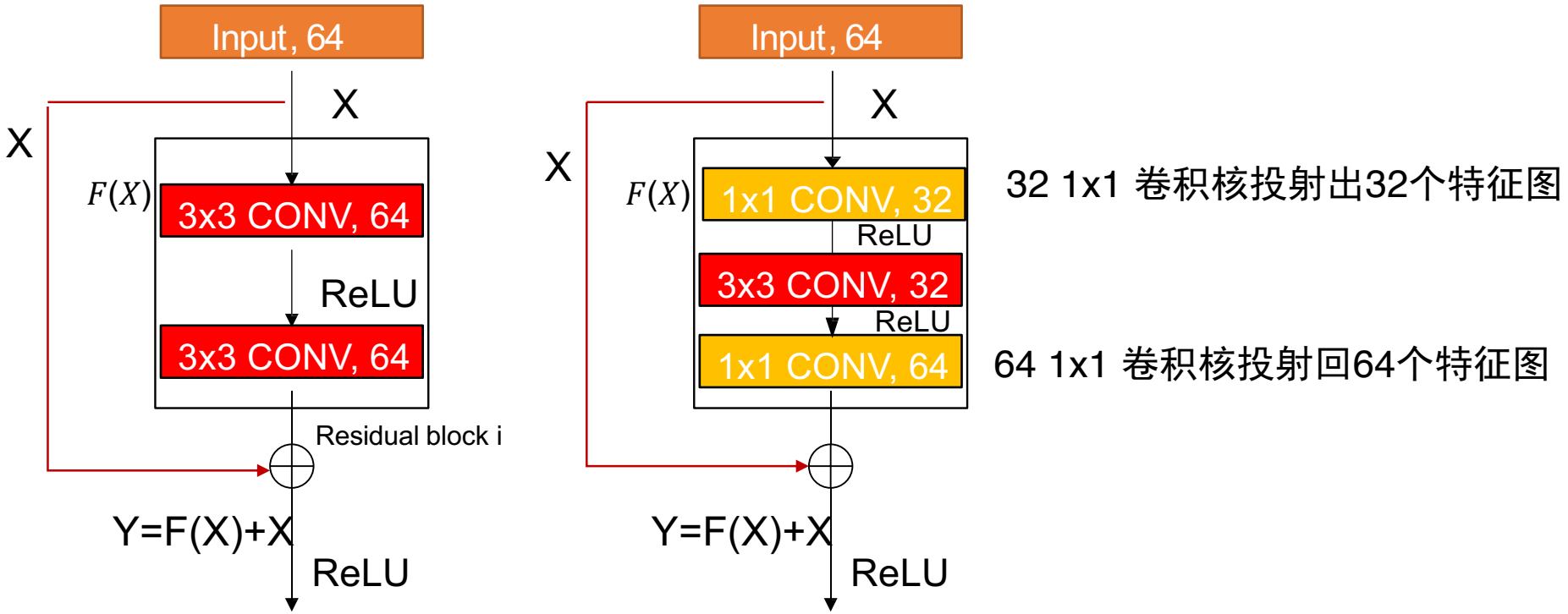


非常深的ResNet

我们不能无限地增加神经网络的深度。
超过1000层的ResNet仍然难以优化！

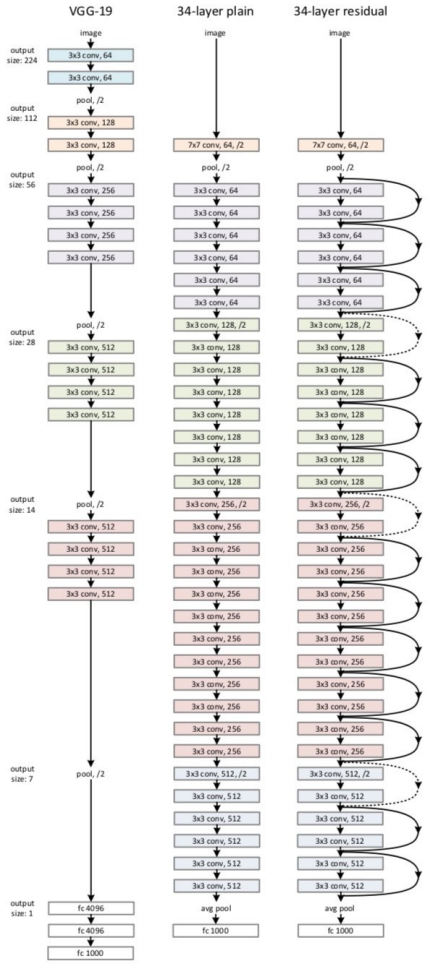
ResNet

- 更深的瓶颈结构
  - 类似GoogLeNet, ResNet 用瓶颈层提高训练效率



ResNet

- 总结
  - ResNet通过堆叠Residual Block残余块来
  - 构建神经网络。 每个残余块具有3×3卷 积。
  - ResNet可以设计非常深的卷积网络（从 ResNet-18到ResNet-152）。
  - 通过使用瓶颈结构，ResNet可以更高效，具有优化的参数与计算量。
  - ResNet在ImageNet数据集上实现3.57%top-5错误率。

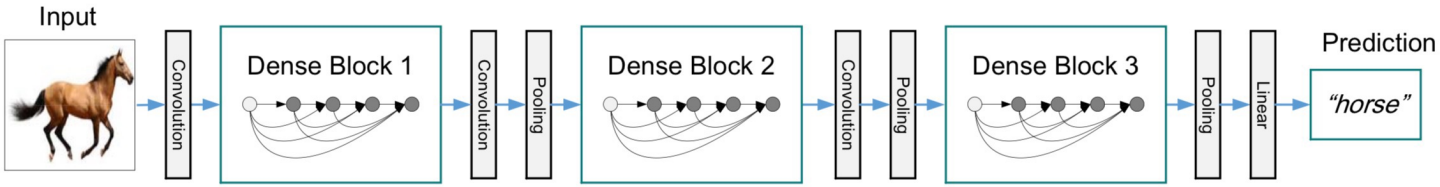


ResNet

- ResNet的训练
  - Batch size 256, 学习率0.01 来warm up 若干周期，然后学习率调整到 0.1.
  - 验证集准确率不变的时候学习率开始衰减0.1.
  - 带动量的SGD，动量因子0.9
  - 权重衰减因子 1e-5.
  - inception网络类似的数据预处理

DenseNet

- DenseNet 制定了层之间的密集连接. 信息可以在DenseNet的不同层 之间灵活地传递。

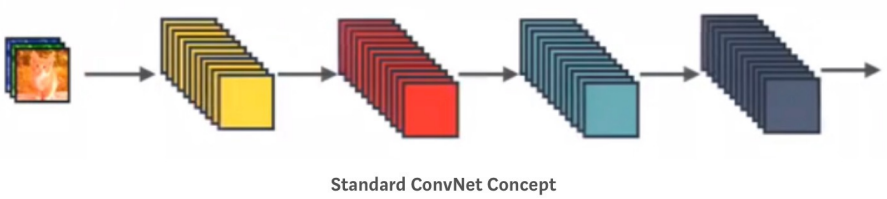


- **特征**
  - DenseNet鼓励特征复用和不同层之间的共享.
  - DenseNet定义了过渡层(transitional layer)做下采样.
  - 对于相似深度的特征图，DenseNet通过使用卷积核的连接来减少权重参数。但是，DenseNet可能导致高内存消耗（特征图的参数占用大）。

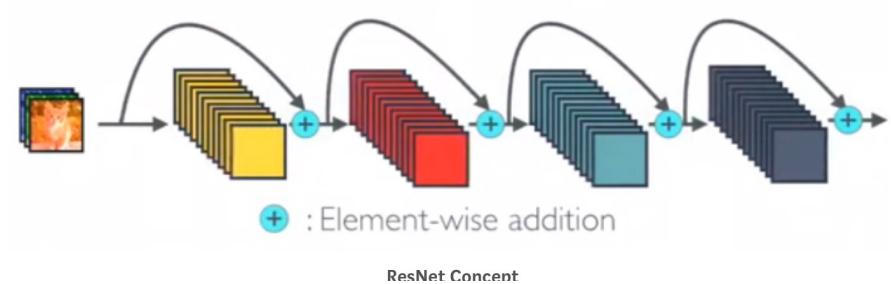
Huang, Gao, et al. “Densely connected convolutional networks.” (2017)

DenseNet

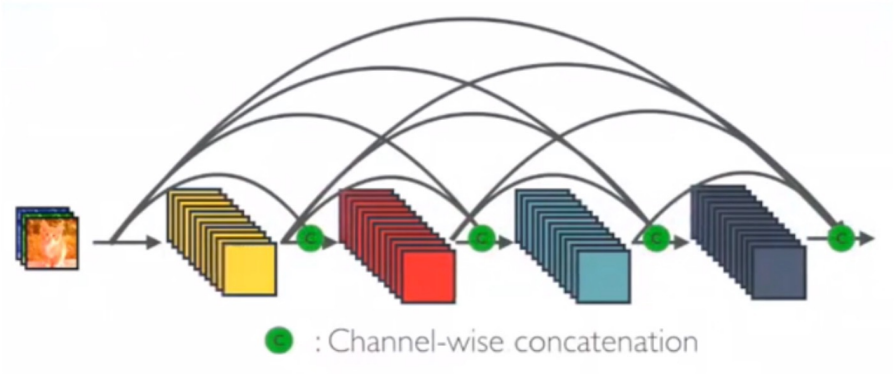
- **DenseNet卷积核连接**



Standard ConvNet Concept



ResNet Concept



One Dense Block in DenseNet

标准ConvNet:
当前层的输出直接输出下一层

ResNet:
当前层的输出先和当前残余块的相同层(Identity)相加，再输出到下一层.

DenseNet:
之前层的输入在dense模块先连接，再输出到下一层.

DenseNet

- **训练:**
  - 与ResNet类似.
- **ResNet的训练**
  - Batch size 256, 学习率0.01 来warm up 若干周期，然后学习 率调整到 0.1.
  - 验证集准确率不变的时候学习率开始衰减0.1.
  - 带动量的SGD，动量因子0.9
  - 权重衰减因子 1e-5.
  - inception网络类似的数据预处理

CNN 架构总结

网络	单一路径?	用了小（k<=3）卷积内核	深度?
AlexNet	是	否	浅 (8 层)
VGG	是	是	深 (16-19 层)
GoogLeNet	否	(V3)是, (V1，V2，V4)否	更深 (22 层)
ResNet	否	是	非常深(34-152 层)
DenseNet	否	是	非常深(121-264 层)

网络	输入大小	参数量	MACs
AlexNet	224x224	233 MB	727 M
VGG-16/19	224x224	528 MB / 548 MB	16 G /20 G
GoogLeNet	224x224	51 MB	2 G
ResNet-18/34	224x224	45 MB / 83 MB	2 G / 4 G
DenseNet-121	224x224	31 MB	3 G

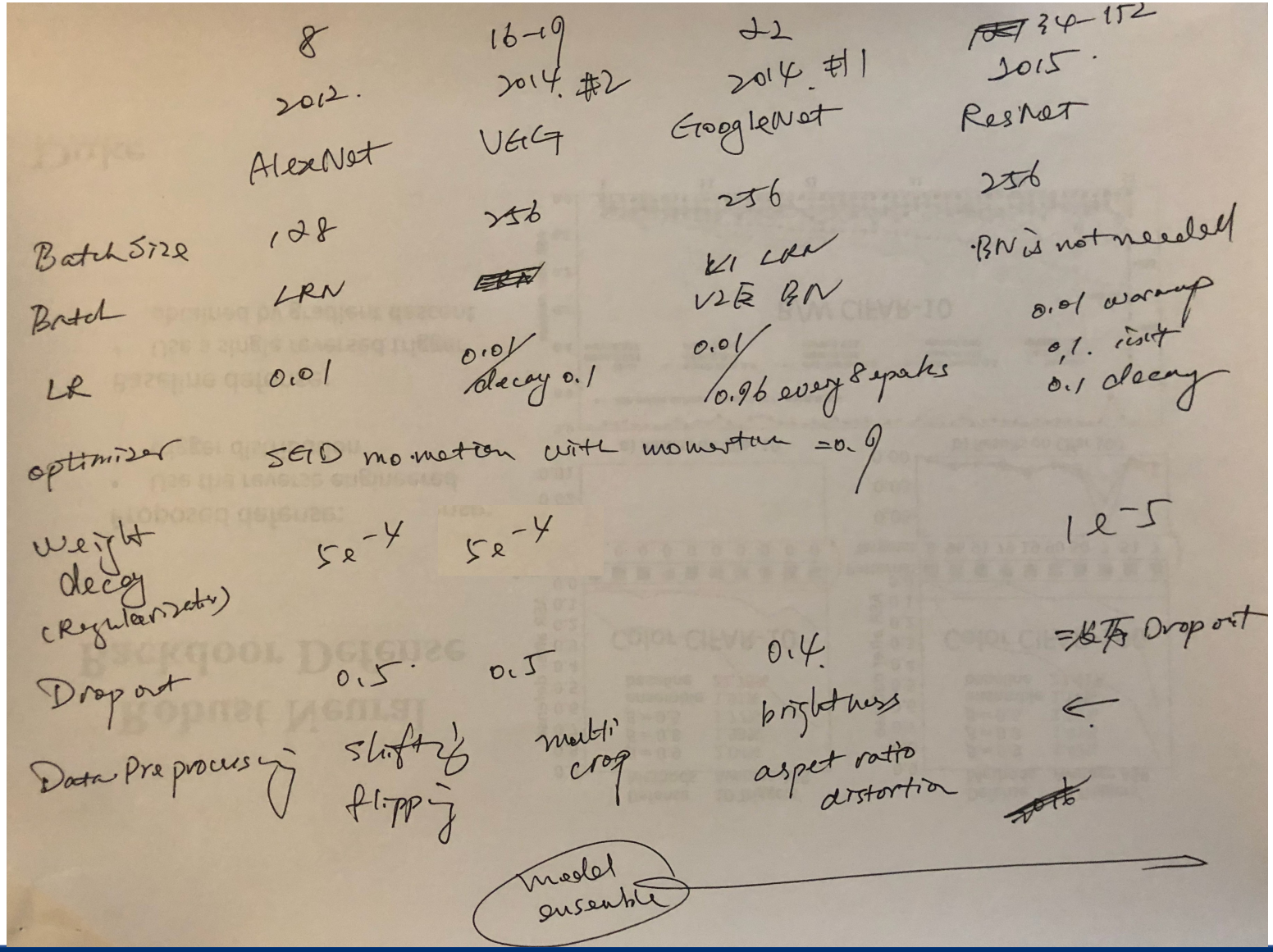
CNN 架构总结

Metrics	LeNet 5	AlexNet	Overfeat fast	VGG 16	GoogLeNet v1	ResNet 50
Top-5 error†	n/a	16.4	14.2	7.4	6.7	5.3
Top-5 error (single crop)†	n/a	19.8	17.0	8.8	10.7	7.0
Input Size	28×28	227×227	231×231	224×224	224×224	224×224
# of CONV Layers	2	5	5	13	57	53
Depth in # of CONV Layers	2	5	5	13	21	49
Filter Sizes	5	3,5,11	3,5,11	3	1,3,5,7	1,3,7
# of Channels	1, 20	3-256	3-1024	3-512	3-832	3-2048
# of Filters	20, 50	96-384	96-1024	64-512	16-384	64-2048
Stride	1	1,4	1,4	1	1,2	1,2
Weights	2.6k	2.3M	16M	14.7M	6.0M	23.5M
MACs	283k	666M	2.67G	15.3G	1.43G	3.86G
# of FC Layers	2	3	3	3	1	1
Filter Sizes	1,4	1,6	1,6,12	1,7	1	1
# of Channels	50, 500	256-4096	1024-4096	512-4096	1024	2048
# of Filters	10, 500	1000-4096	1000-4096	1000-4096	1000	1000
Weights	58k	58.6M	130M	124M	1M	2M
MACs	58k	58.6M	130M	124M	1M	2M
Total Weights	60k	61M	146M	138M	7M	25.5M
Total MACs	341k	724M	2.8G	15.5G	1.43G	3.9G
Pretrained Model Website	[56]‡	[57, 58]	n/a	[57–59]	[57–59]	[57–59]

TABLE II

https://arxiv.org/abs/1703.09039

CNN 架构总结


8
2012.
AlexNet
16-19
2014. #2
VGG
22
2014. #11
GoogleNet
34-152
2015.
ResNet
Batch Size
128
256
256
256
BN is not needed
Batch
LRN
LRN
VGG BN
LR
0.01
0.01
decay 0.1
0.01
0.96 every 8 epoks
0.01 warmup
0.1. rise
0.1 decay
optimizer
SGD momentum with momentum =0.9
Weight decay (Regularization)
5e-4
5e-4
1e-5
Drop out
0.5.
0.5
0.4.
=VGG Drop out
Data Preprocessing
shifting
flipping
multi crop
brightness
aspect ratio distortion
model ensemble

Pytorch示例-VGG

- 准备网络

```
import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg11', pretrained=True)
# or any of these variants
# model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg11_bn', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg13', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg13_bn', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16_bn', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg19', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg19_bn', pretrained=True)
model.eval()
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (11): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (12): ReLU(inplace=True)
    (13): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (14): ReLU(inplace=True)
    (15): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (16): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (17): ReLU(inplace=True)
    (18): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (19): ReLU(inplace=True)
    (20): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Pytorch示例-VGG

- 准备数据

```
# Download an example image from the pytorch website
import urllib
url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
try: urllib.URLopener().retrieve(url, filename)
except: urllib.request.urlretrieve(url, filename)
```

```
# sample execution (requires torchvision)
from PIL import Image
from torchvision import transforms
input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
```

Pytorch示例-VGG

- 推断，前向传播

```
# sample execution (requires torchvision)
from PIL import Image
from torchvision import transforms
input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by the model

# move the input and model to GPU for speed if available
if torch.cuda.is_available():
    input_batch = input_batch.to('cuda')
    model.to('cuda')

with torch.no_grad():
    output = model(input_batch)
# Tensor of shape 1000, with confidence scores over Imagenet's 1000 classes
print(output[0])
# The output has unnormalized scores. To get probabilities, you can run a softmax on it.
probabilities = torch.nn.functional.softmax(output[0], dim=0)
print(probabilities)
```

Pytorch示例-VGG

- 判断类别

```
# Download ImageNet labels
!wget https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt
```

```
# Read the categories
with open("imagenet_classes.txt", "r") as f:
    categories = [s.strip() for s in f.readlines()]
# Show top categories per image
top5_prob, top5_catid = torch.topk(probabilities, 5)
for i in range(top5_prob.size(0)):
    print(categories[top5_catid[i]], top5_prob[i].item())
```

```
Samoyed 0.6673735976219177
Pomeranian 0.16195248067378998
Eskimo dog 0.017759328708052635
collie 0.017686177045106888
keeshond 0.01706554740667343
```