


更深的网络



经典神经网络模型一

概览

1. 计算机视觉与卷积神经网络
2. AlexNet
3. VggNet



1. 计算机视觉与 卷积神经网络

神经网络的应用领域

- ▶ 计算机视觉（最为成功的应用领域）
- ▶ 自然语言处理（较为成功的应用领域）
- ▶ 智能博弈（较为成功的应用领域）
- ▶ 其它

计算机视觉简史

- ▶ 20世纪50年代归入模式识别——主要集中在二维图像分析与识别上，例如光学字符识别等。
- ▶ 20世纪60年代MIT的Roberts通过计算机程序从数字图像中提取出诸如立方体、棱柱体等多面体结构。其开创了三维计算机视觉。
- ▶ 20世纪80年代中期，计算机视觉获得了蓬勃发展，出现了很多新概念、新方法、新理论，例如主动视觉框架、视觉集成理论框架等。
- ▶ 2012年起，人工神经网络在计算机视觉应用中取得重大成果，开启计算机视觉新时代，到目前为止计算机视觉在多种任务中的精度等指标中已远远超过人类。

ILSVRC

- 超过1500万张图片
- 超过2200类别



ILSVRC classification

- 120万张图片用于训练1000分类
- 使用Top-5 error评价模型



EntleBucher



Appenzeller

ILSVRC2010

Codename	CLS	Institutions
Hminmax	54.4	Massachusetts Institute of Technology
IBM	70.1	IBM research [†] , Georgia Tech [‡]
ISIL	44.6	Intelligent Systems and Informatics Lab., The University of Tokyo
ITNLP	78.7	Harbin Institute of Technology
LIG	60.7	Laboratoire d'Informatique de Grenoble
NEC	28.2	NEC Labs America [†] , University of Illinois at Urbana-Champaign [‡] , Rutgers [⌘]
NII	74.2	National Institute of Informatics, Tokyo, Japan [†] , Hefei Normal Univ. Heifei, China [‡]
NTU	58.3	CeMNet, SCE, NTU, Singapore
Regularities	75.1	SRI International
UCI	46.6	University of California Irvine
XRCE	33.6	Xerox Research Centre Europe

ILSVRC2011

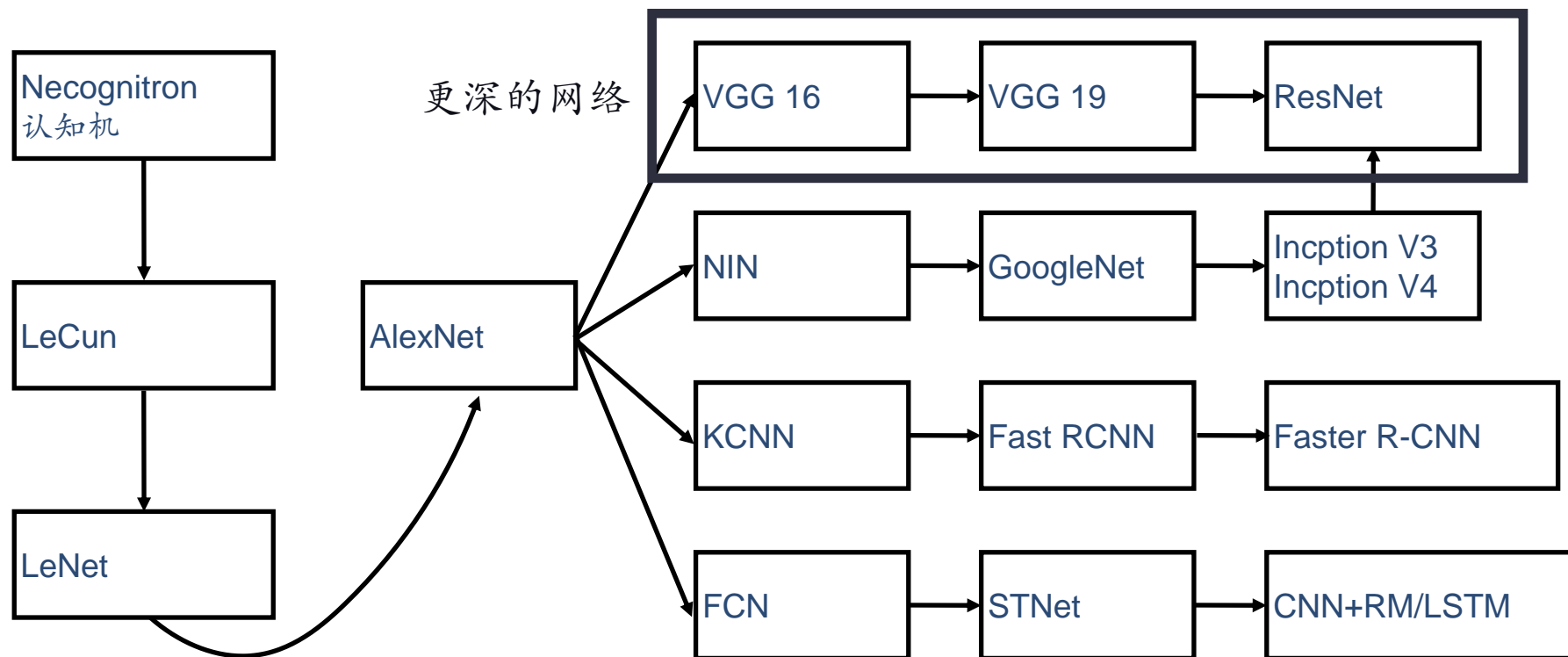
Codename	CLS	LOC	Institutions
ISI	36.0	-	Intelligent Systems and Informatics lab, University of Tokyo
NII	50.5	-	National Institute of Informatics, Japan
UvA	31.0	42.5	University of Amsterdam [†] , University of Trento [‡]
XRCE	25.8	56.5	Xerox Research Centre Europe [†] , CIII [‡]

ILSVRC2012

Codename	CLS	LOC	Institutions
ISI	26.2	53.6	University of Tokyo [†] , JST PRESTO [‡]
LEAR	34.5	-	LEAR INRIA Grenoble [†] , TVPA Xerox Research Centre Europe [‡]
VGG	27.0	50.0	University of Oxford
SuperVision	16.4	34.2	University of Toronto
UvA	29.6	-	University of Amsterdam
XRCE	27.1	-	Xerox Research Centre Europe [†] , LEAR INRIA [‡]

AlexNet大幅领先第二名26.2%的错误率

卷积神经网络结构演化史



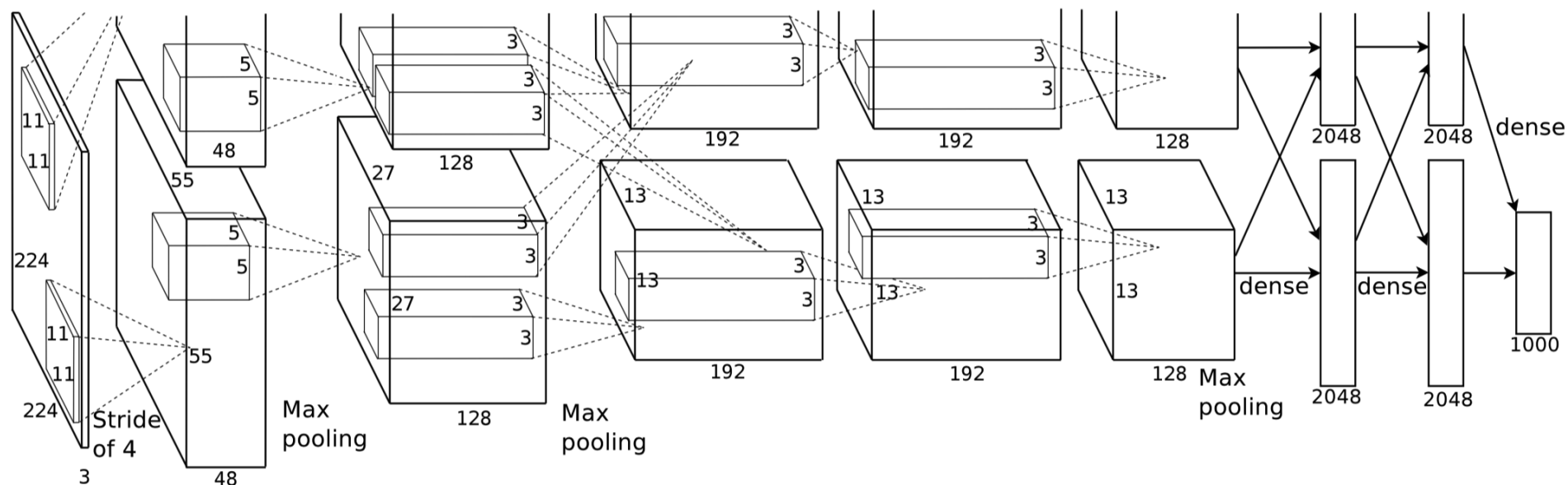


2. AlexNet



2.1 AlexNet 简介

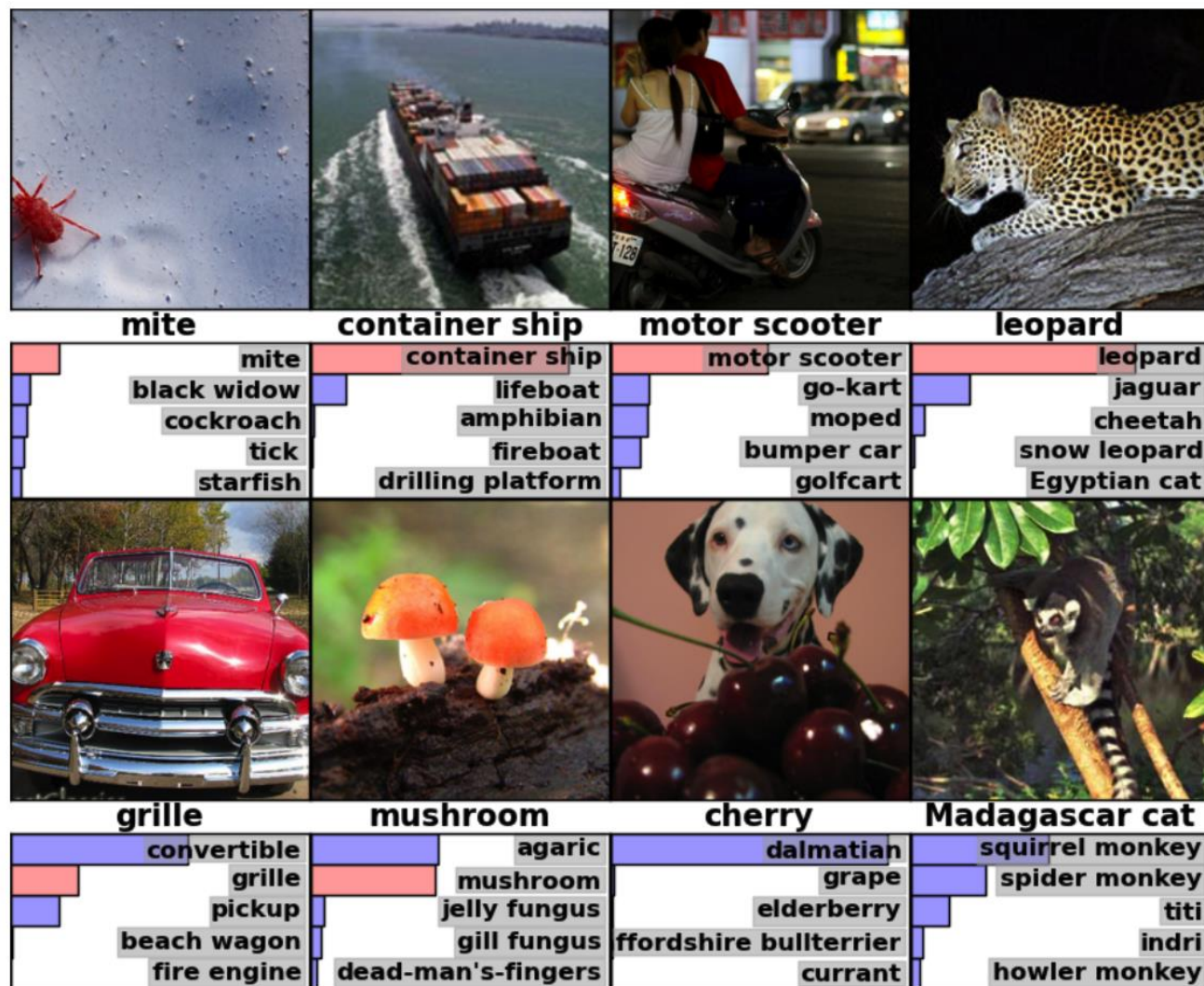
简介



AlexNet在ILSVRC-2012的1000分类竞赛上获得了第一名的成绩，AlexNet将

TOP-5错误率降低到了15.3%，大幅领先第二名的26.2%的错误率。

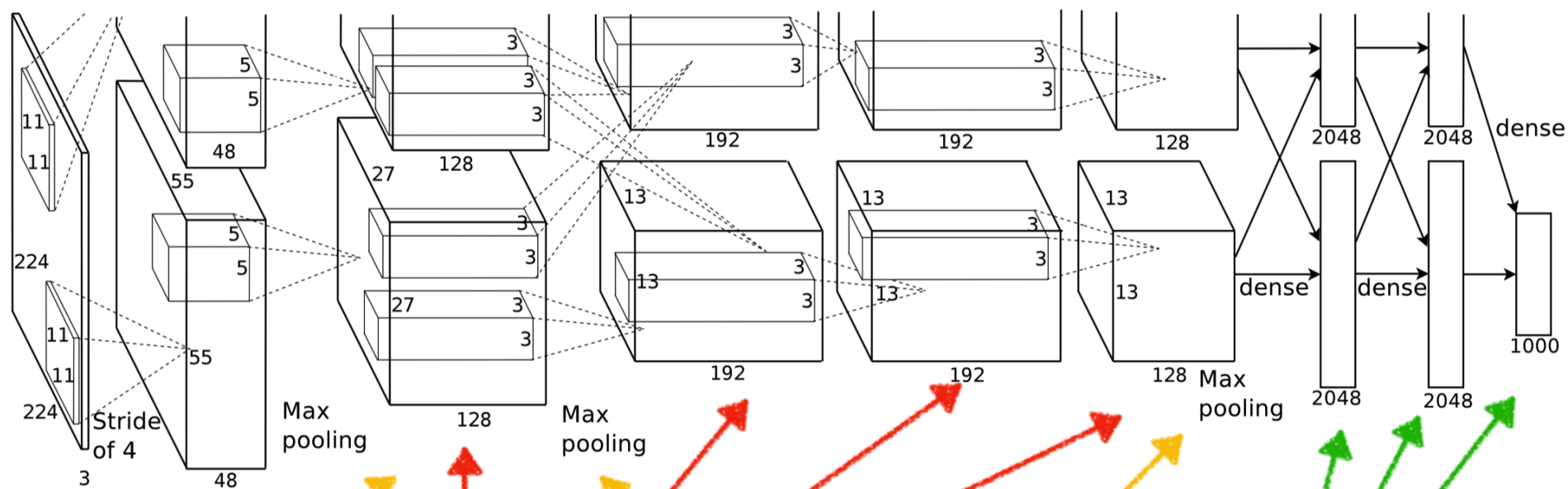
8张ILSVRC-2010测试图片结果





2.2 AlexNet 模型结构

AlexNet模型总体结构



共8个参数层

卷积层

池化层

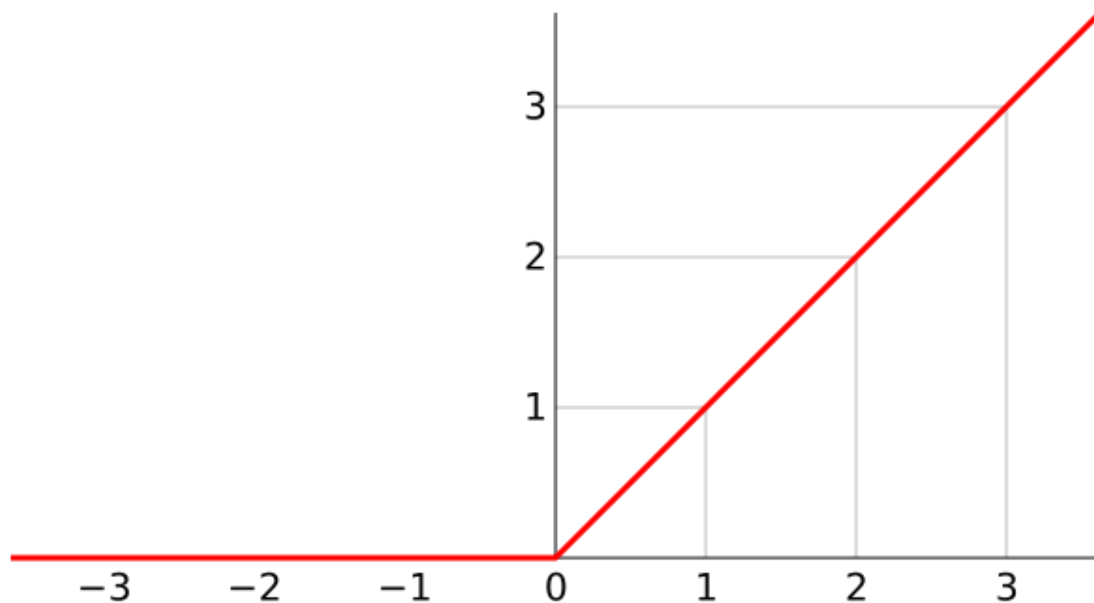
全连接层

1. 输入层：输入 $227*227*3$ 图片。
2. 卷积层：96个 $11*11$ 卷积核，步长为4，局部响应归一化。[输出尺寸为 $55*55*96$]
3. 池化层： $3*3$ 的滑动窗最大池化，步长为2。[输出尺寸为 $27*27*96$]
4. 卷积层：256个 $5*5$ 卷积核，步长为1，局部响应归一化。[输出尺寸为 $27*27*256$]
5. 池化层： $3*3$ 的滑动窗最大池化，步长为2。[输出尺寸为 $13*13*256$]
6. 卷积层：384个 $3*3$ 卷积核，步长为1。[输出尺寸为 $13*13*384$]
7. 卷积层：384个 $3*3$ 卷积核，步长为1。[输出尺寸为 $13*13*384$]
8. 卷积层：256个 $3*3$ 卷积核，步长为1。[输出尺寸为 $13*13*256$]
9. 池化层： $3*3$ 的滑动窗最大池化，步长为2。[输出尺寸为 $6*6*256$]
10. 全连接层：4096个神经元；接ReLU激活函数。
11. 全连接层：4096个神经元；接ReLU激活函数。
12. 全连接层：1000个神经元；接softmax激活函数。



2.3 AlexNet 主要贡献

ReLU激活函数

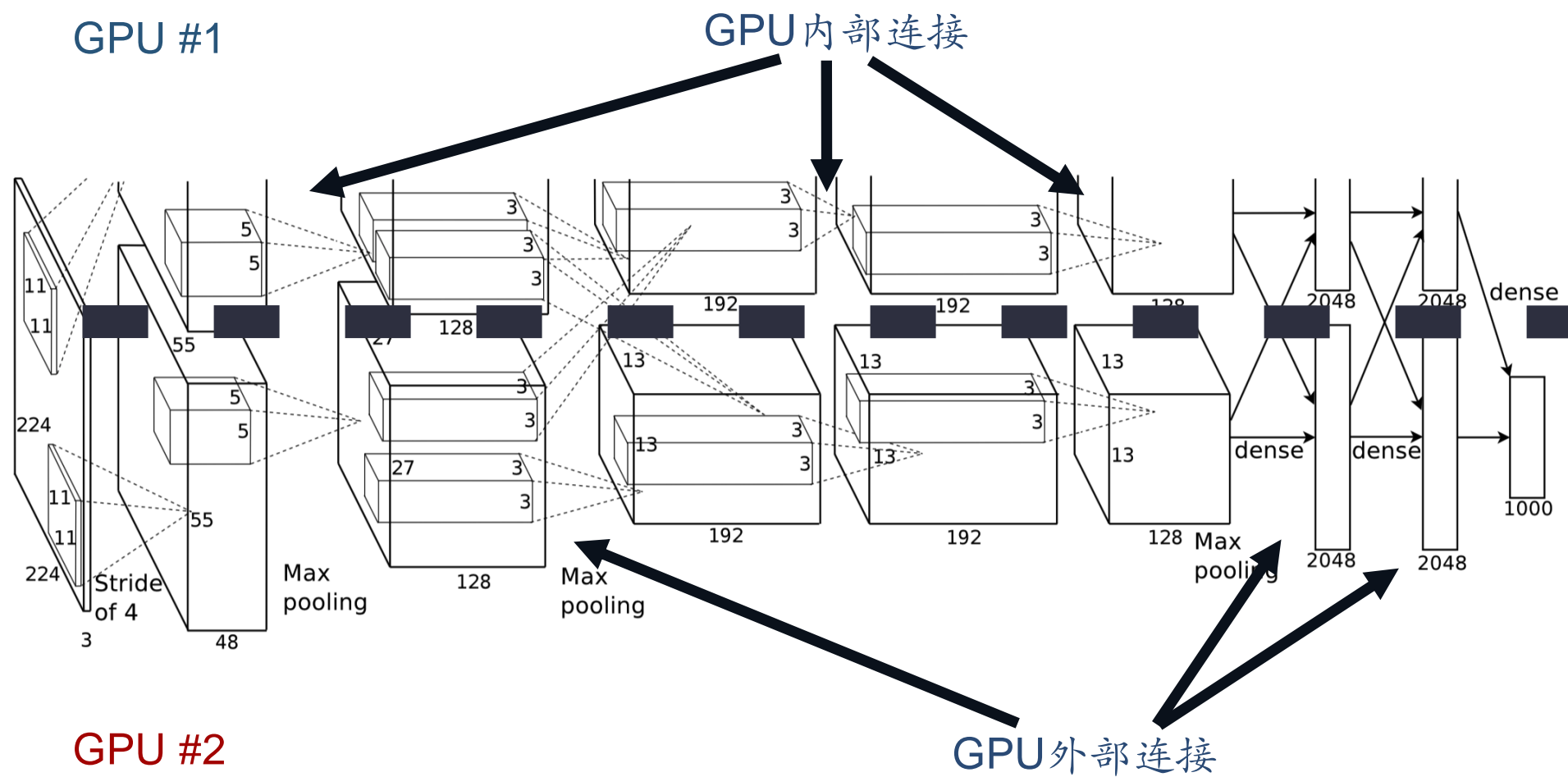


$$f(x) = \max(0, x)$$

优点:

1. 与logistic、tanh相比，计算量更小。
2. 导数值稳定，相对于其他激活函数对神经网络梯度不稳定问题更加鲁棒。

分布式GPU运算



分布式GPU运算



图为第一层2*48个卷积核可视化结果

- 上面48个在 GPU#1 上的卷积核只对边界感兴趣。
- 下面48个在 GPU#2 上的卷积核对颜色感兴趣。

分布式GPU运算的优点

- ▶ 使用GPU运算速度大大加快。
- ▶ 多GPU使得复杂模型得以训练。
- ▶ 文献中提到，与使用单GPU一半神经元相比，Top-1和Top5 error分别降低了1.7%与1.2%

LRN

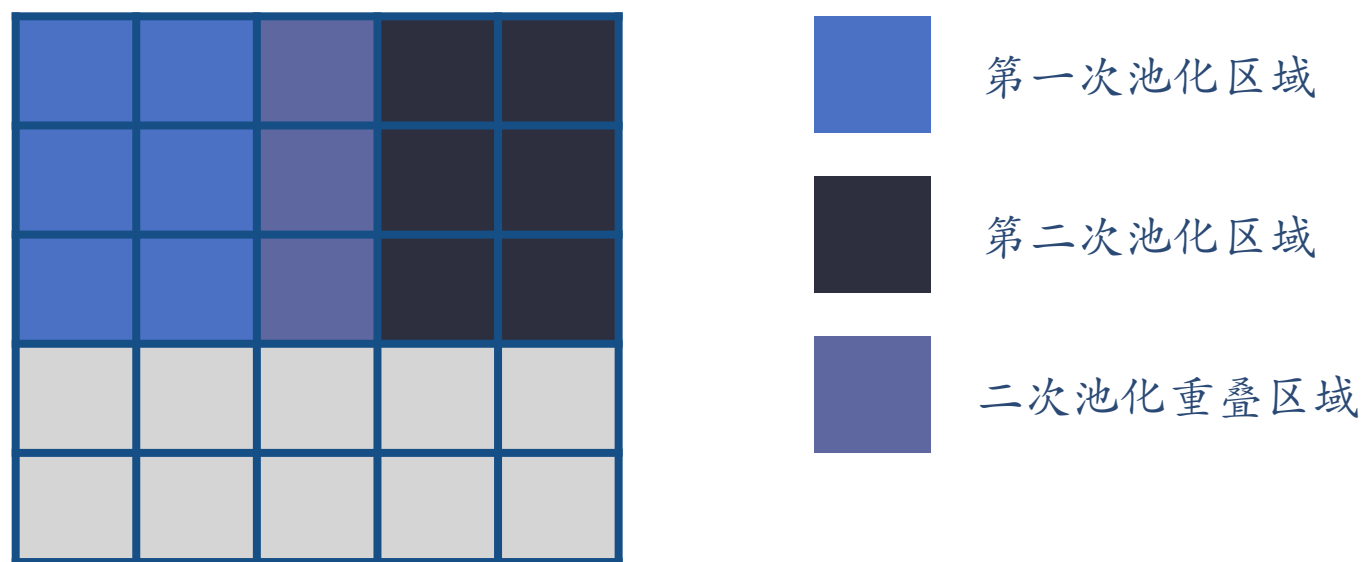
局部响应归一化（**Local Response Normalization, LRN**）是AlexNet中提出的配合ReLU激活函数使用提高泛化能力的方法。文献中指出LRN使得Top-1 error与Top-5 error 分别减少了1.4%与1.2%。部分实验证明LRN并没有作用。

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

其中 \mathbf{a} 表示经过激活函数的卷积特征图， N 表示卷积之后的特征图数量， n 表示近邻的特征图数量。 k 、 α 、 β 、 $n/2$ 均为超参数，通常 $k=2$ ， $\alpha=1e-4$ ， $\beta=0.75$ 、 $n=5$ 。

重叠池化

重叠池化（**Overlapping Pooling**）即使池化窗的步长小于池化层的大小，在池化时产生重叠。文献中提到使用重叠池化Top-1和Top-5 error分别减少了0.4%和0.3%。



重叠池化：窗口长3、步长2



2.4 AlexNet 正则化方法

数据集增强

- ▶ 样本缩放与裁剪；
- ▶ 随机改变图片亮度。

数据集增强一

- ▶ 首先，将图片等比缩放，使得最小边缩放为256。
- ▶ 其次，在缩放后的图片中部取一块[256, 256, 3]的图片。
- ▶ 再次，将取出的图片滑动裁剪出[224, 224, 3]的图片。
- ▶ 最后，随机对裁剪后的图片进行水平翻转。

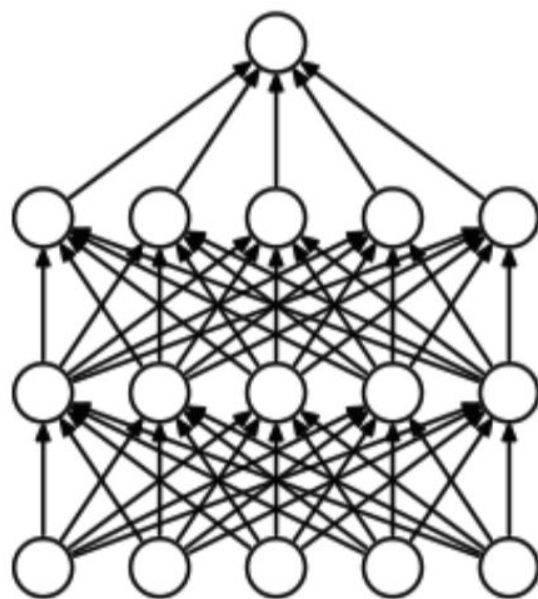
训练时： 使得数据集扩大了 $(256 - 224)^2 * 2 = 2048$ 倍

数据集增强一

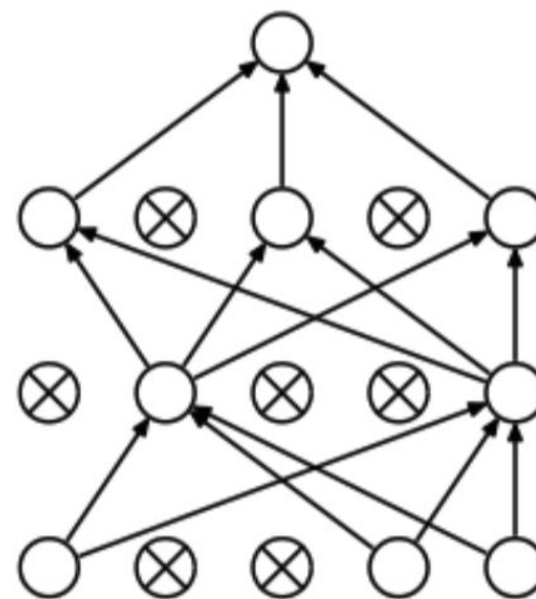
测试时：

将原图按照训练时的方法，得到 $[256, 256, 3]$ 图片，再分别取四个角与中部共5张大小为 $[224, 224, 3]$ 的图片，再进行翻转，共得到10张图片，送入模型进行预测，取10次结果的平均值作为预测结果。

Dropout



Standard Neural Net



After applying dropout.

- 在两个4096个神经元的全连接层中使用Dropout。
- 保留神经元的概率 (keep prob) 为：0.5



2.5 AlexNet 训练方法

Momentum

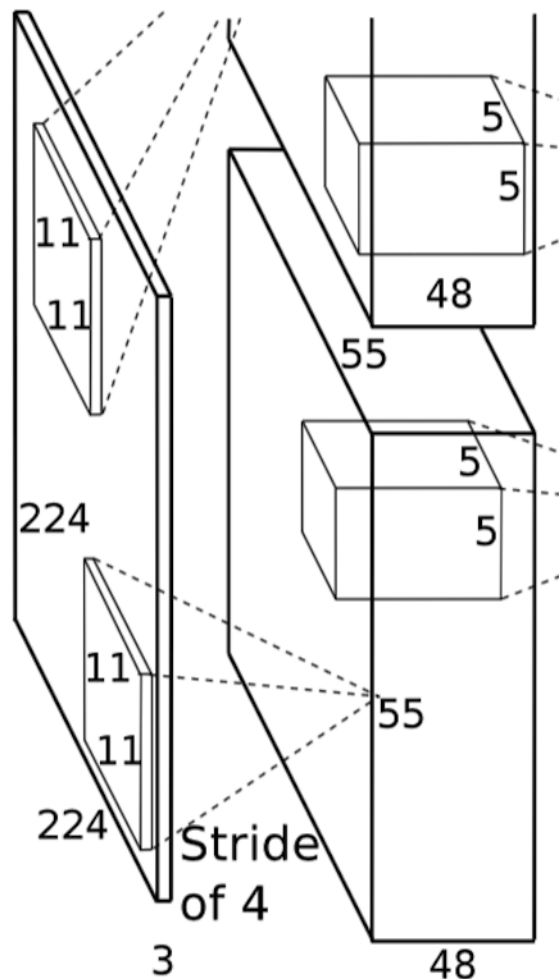
$$\begin{aligned} v_{i+1} &:= \overbrace{0.9}^{\text{动量 (阻尼系数)}} \cdot v_i - \overbrace{0.0005}^{\text{权重衰减系数}} \cdot \underbrace{\epsilon}_{\text{学习率}} \cdot w_i - \underbrace{\epsilon}_{\text{学习率}} \cdot \underbrace{\left\langle \frac{\partial L}{\partial w} \Big|_{w_i} \right\rangle_{D_i}}_{\text{一个批次数据 (128个) 的平均梯度}} \\ w_{i+1} &:= w_i + v_{i+1} \end{aligned}$$

在2块 NVIDIA GTX 580 3GB 上训练5-6天



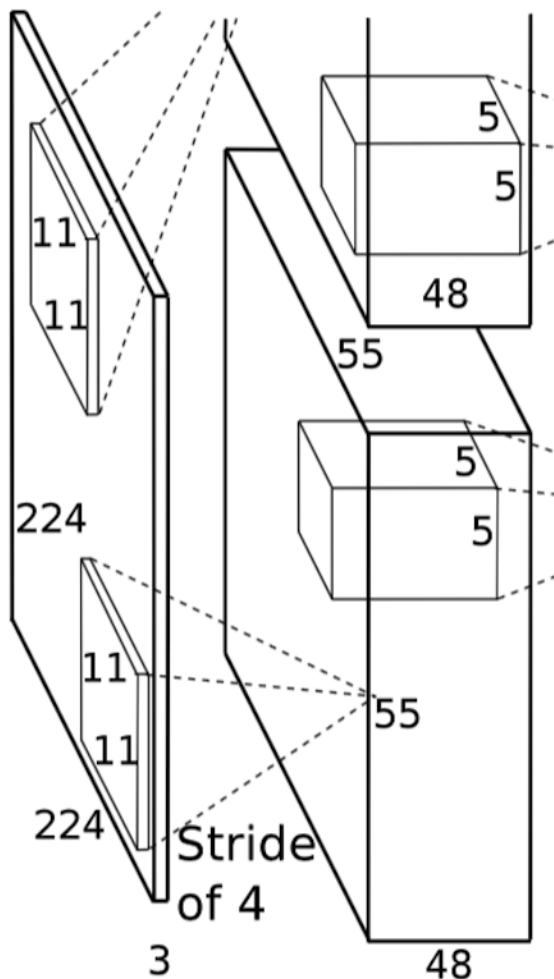
2.6 AlexNet 参数 数量与计算性能

Layer 1



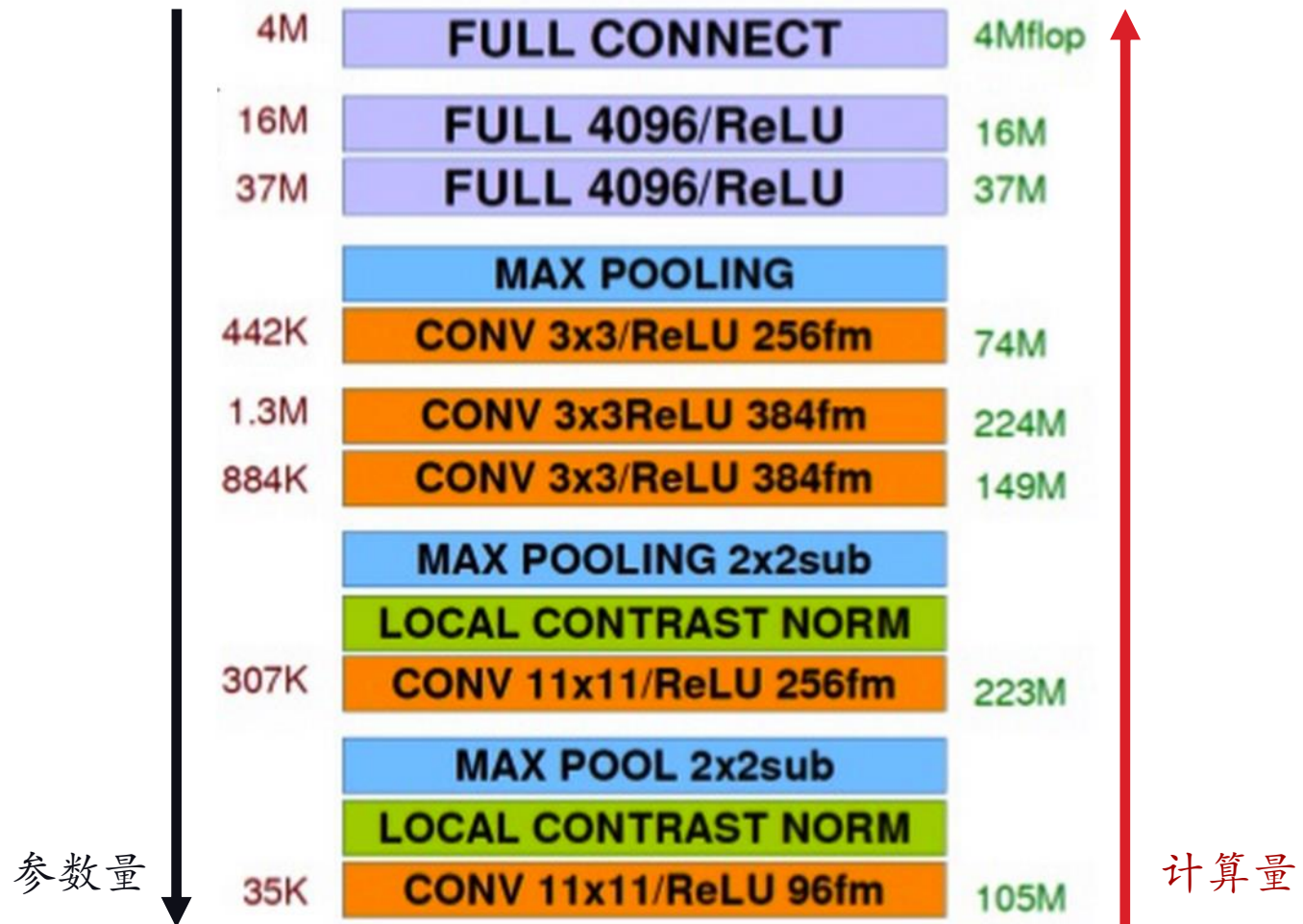
- 输入图片: $224 * 224 * 3$
- 边界填充: $3 * 3$
- 卷积核: $11 * 11$
- 步长: 4
- 激活函数: ReLU
- 输出: $55 * 55 * 96$

Layer 1



- 特征图: $55 * 55 * 96 = 290400$
- 参数量: $(11 * 11 * 3 + 1) * 96 = 34944$
- 每个卷积核计算量: $11 * 11 * 3 = 363$
- 总计算量: $290400 * 363 = 105415200$

模型结构





2.7 使用Pytorch 实现 AlexNet 模型

AlexNet模型实现要求

- ▶ 无需实现与加入LRN层（虽然TensorFlow有相关API）；
- ▶ 无需实现多GPU分布式计算，但需要实现AlexNet在多GPU上的模型结构；
- ▶ 实现其代价函数（但无需训练）；
- ▶ 输出模型参数数量。



TensorFlow实现

```
num_classes = 1000
```

```
inputs_img = tf.placeholder(shape=[None, 227, 227, 3], dtype=tf.float32)
labels = tf.placeholder(shape=[None, num_classes], dtype=tf.int32)
```

```
net_GPU_0 = slim.conv2d(inputs_img, 48, [11, 11], stride=4, padding='VALID')
net_GPU_1 = slim.conv2d(inputs_img, 48, [11, 11], stride=4, padding='VALID')
```

```
net_GPU_0 = slim.max_pool2d(net_GPU_0, [3, 3], stride=2, padding='VALID')
net_GPU_1 = slim.max_pool2d(net_GPU_1, [3, 3], stride=2, padding='VALID')
```

```
net_GPU_0 = slim.conv2d(net_GPU_0, 128, [5, 5], stride=1, padding='VALID')
net_GPU_1 = slim.conv2d(net_GPU_1, 128, [5, 5], stride=1, padding='VALID')
```

```
net_GPU_0 = slim.max_pool2d(net_GPU_0, [3, 3], stride=2, padding='VALID')
net_GPU_1 = slim.max_pool2d(net_GPU_1, [3, 3], stride=2, padding='VALID')
```

```
net = tf.concat([net_GPU_0, net_GPU_1], 3)
```

```
net_GPU_0 = slim.conv2d(net, 192, [3, 3], stride=1, padding='VALID')
net_GPU_1 = slim.conv2d(net, 192, [3, 3], stride=1, padding='VALID')
```

```
net_GPU_0 = slim.conv2d(net_GPU_0, 192, [3, 3], stride=1, padding='VALID')
net_GPU_1 = slim.conv2d(net_GPU_1, 192, [3, 3], stride=1, padding='VALID')
```

```
net_GPU_0 = slim.conv2d(net_GPU_0, 128, [3, 3], stride=1, padding='VALID')
net_GPU_1 = slim.conv2d(net_GPU_1, 128, [3, 3], stride=1, padding='VALID')
```

```
net_GPU_0 = slim.max_pool2d(net_GPU_0, [3, 3], stride=2, padding='VALID')
net_GPU_1 = slim.max_pool2d(net_GPU_1, [3, 3], stride=2, padding='VALID')
```

```
net = tf.concat([net_GPU_0, net_GPU_1], 3)
```

```
net = slim.flatten(net)
```

```
net = slim.fully_connected(net, 4096)
```

```
net = slim.fully_connected(net, 4096)
```

```
outputs = slim.fully_connected(net, num_classes, activation_fn=tf.nn.softmax)
```




Pytorch实现

```

class AlexNet(nn.Module):

    def __init__(self, num_classes=1000):
        super(AlexNet, self).__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(64, 192, kernel_size=5, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            nn.Conv2d(192, 384, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
        )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(),
            nn.Linear(256 * 6 * 6, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), 256 * 6 * 6)
        x = self.classifier(x)
        return x

```

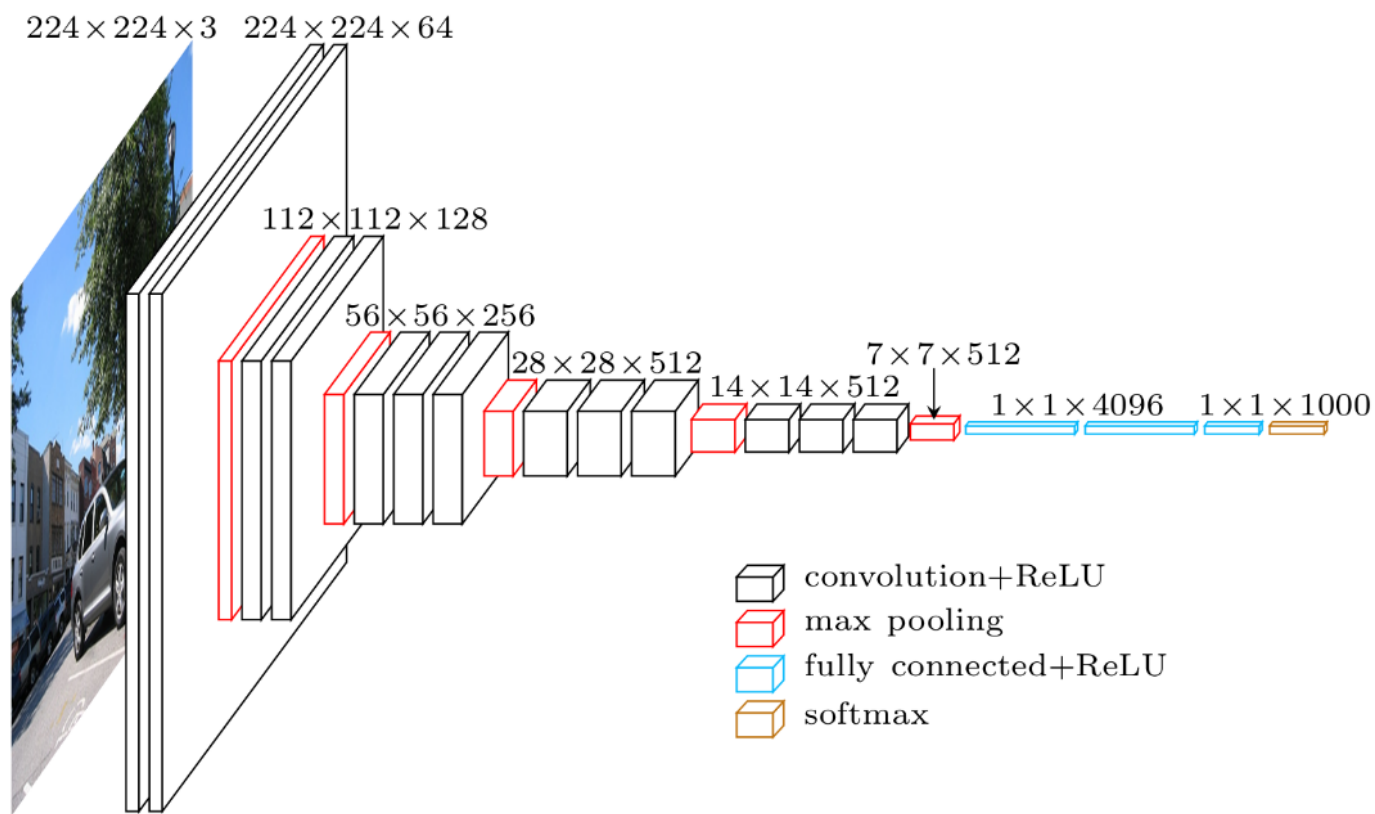


3. VggNet



3.1 VggNet 简介

简介



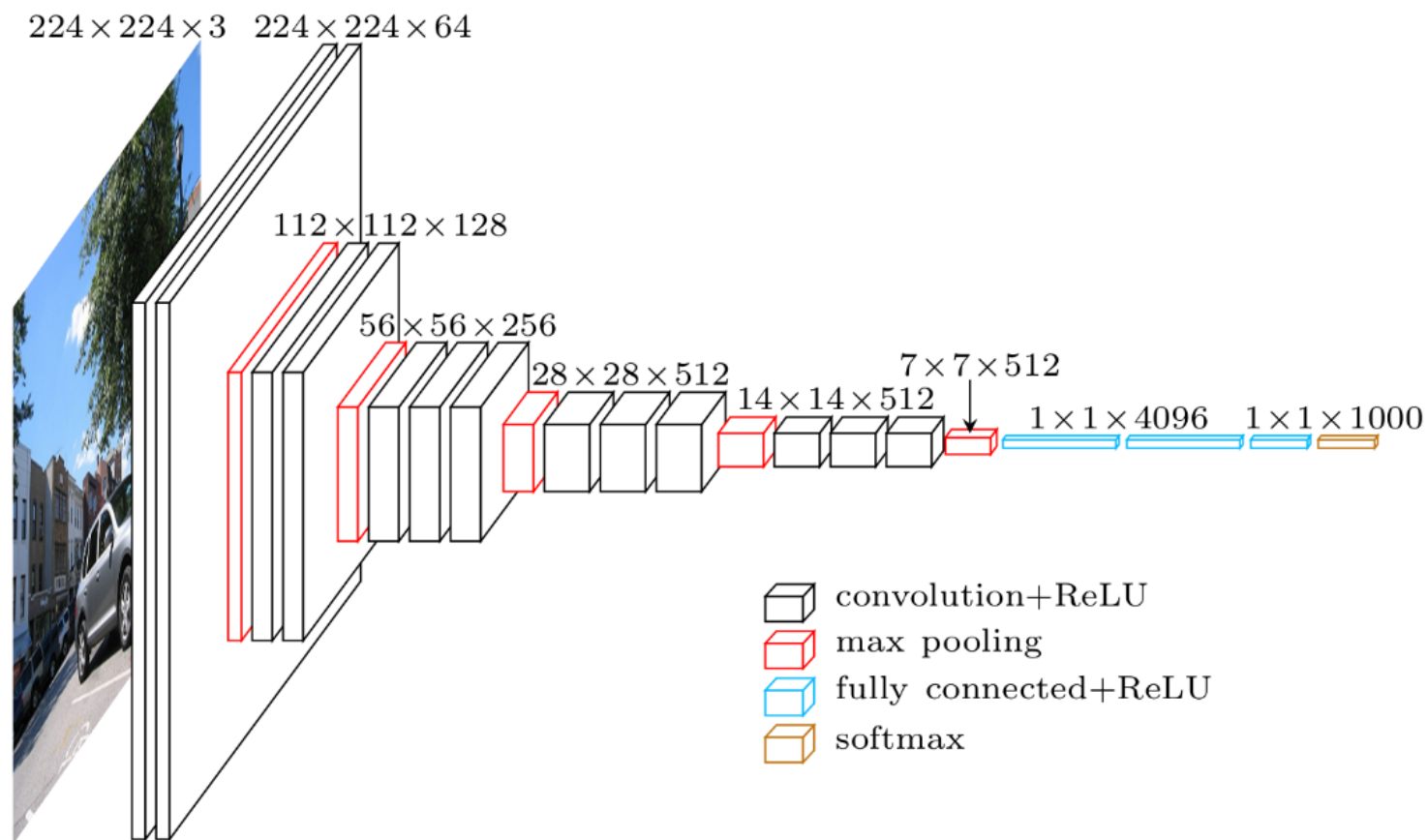
VggNet在ILSVRC-2014中获得了定位第一、分类第二的好成绩，top-5 error

降低至7.3%。VggNet的特点是足够深。参数层达到了16层与19层之多。



3.2 VggNet 模型结构

Vgg16



卷积层全部使用 3×3 小卷积核，共16个参数层，Vgg19又增加了3个卷积层。

Vgg-n

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					

Vgg-16

Vgg-19

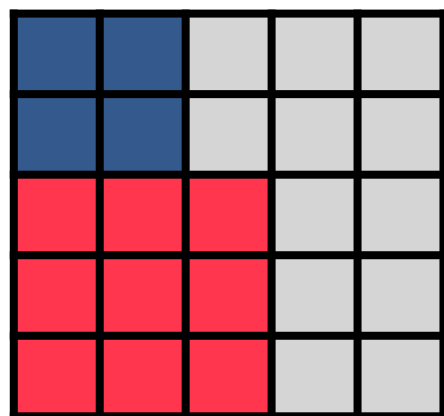
经对比测试Vgg-A与Vgg-A-LRN，发现LRN并没有对模型性能有贡献，反而增加了模型的计算量。



3.3 VggNet 主要贡献

使用更小的卷积核

使用更大的卷积核的意义是什么？

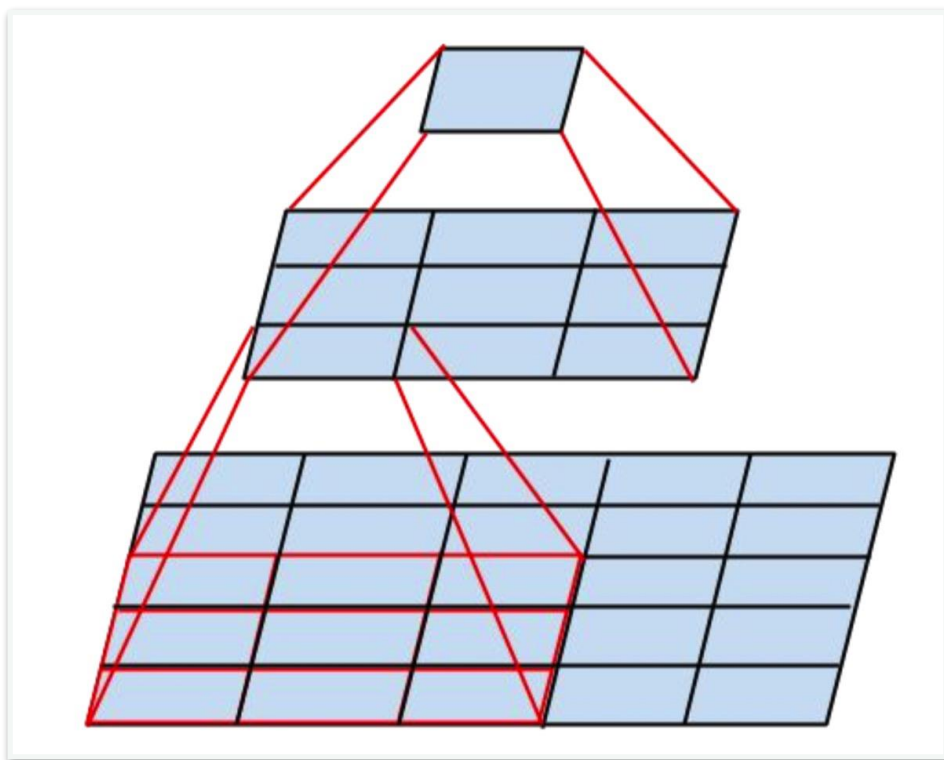


2 * 2卷积核“探测”面积小

3 * 3卷积核“探测”面积大

实验：使用CNN在MNIST识别任务中，观察不同大小卷积核对模型性能的影响。

使用更小的卷积核



1个 $5 * 5$ 卷积核的参数数量:

$$5 * 5 * C^2$$

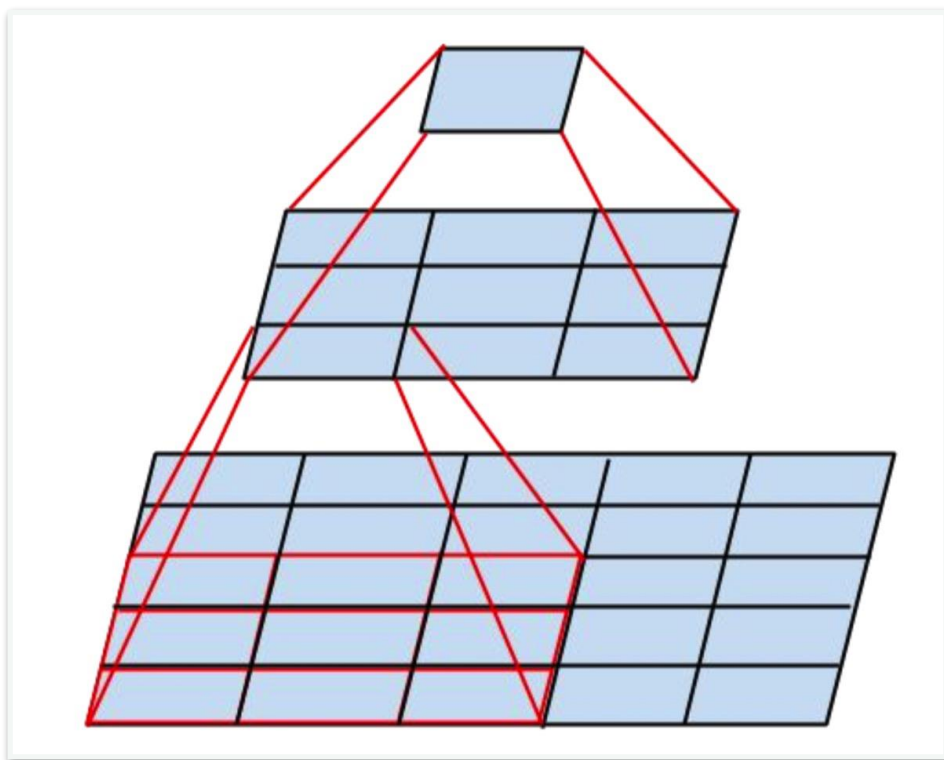
2个 $3 * 3$ 卷积核的参数数量:

$$2 * 3 * 3 * C^2$$

其中C表示输入输出通道数。

使用2个 $3 * 3$ 卷积核的“探测”面积等价于1个 $5 * 5$ 卷积核的“探测”面积。

使用更小的卷积核



1个 $7 * 7$ 卷积核的参数数量:

$$7 * 7 * C^2$$

3个 $3 * 3$ 卷积核的参数数量:

$$3 * 3 * 3 * C^2$$

使用3个 $3 * 3$ 卷积核的“探测”面积等价于1个 $7 * 7$ 卷积核的“探测”面积。

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					

→ 近似于5 * 5卷积

→ 近似于7 * 7卷积

→ 近似于9 * 9卷积

小的卷积核的优点

- ▶ 小卷积核堆叠可达到同大卷积核一样的“探测面积”；
- ▶ 相比大卷积核，小卷积核堆叠使用更少的参数；
- ▶ 相比大卷积核，小卷积核堆叠使用更少的计算量；
- ▶ 相比大卷积核，小卷积核堆叠的非线性表达能力更强。

更深的模型

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

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

VggNet通过实验证明：更深的模型性能更好。



3.4 VggNet 训练方法

Mutil-Scale 数据预处理

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	24.8	7.5
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	24.8	7.5

方法：随机等比缩放图片，使图片最小边在一个范围内，然后裁剪出224 * 224的训练图片。

Momentum

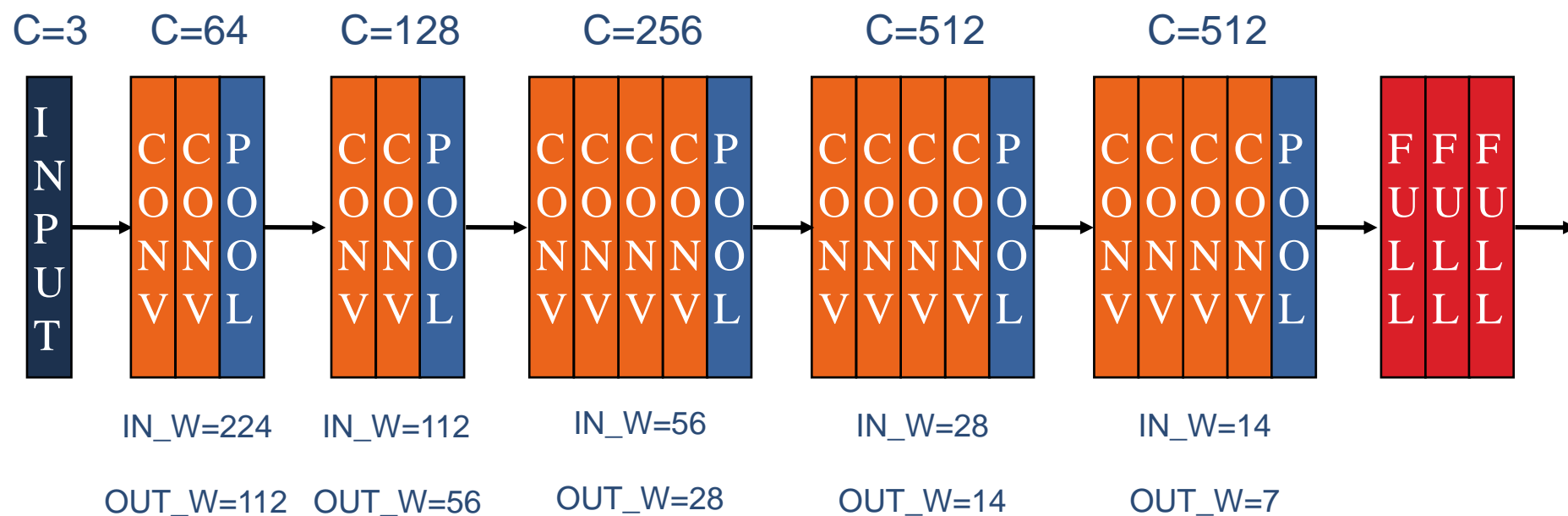
类似于AlexNet，使用动量梯度下降法训练模型。批次大小为256，动量大小为0.9

使用4个NVIDIA Titan Black GPU训练模型2-3周，每个GPU分别计算同一批次中的一部分数据的梯度，求平均值进行更新参数，速度提高了3.75倍。



3.5 VggNet 参数 数量与计算性能

Vgg19 参数数量



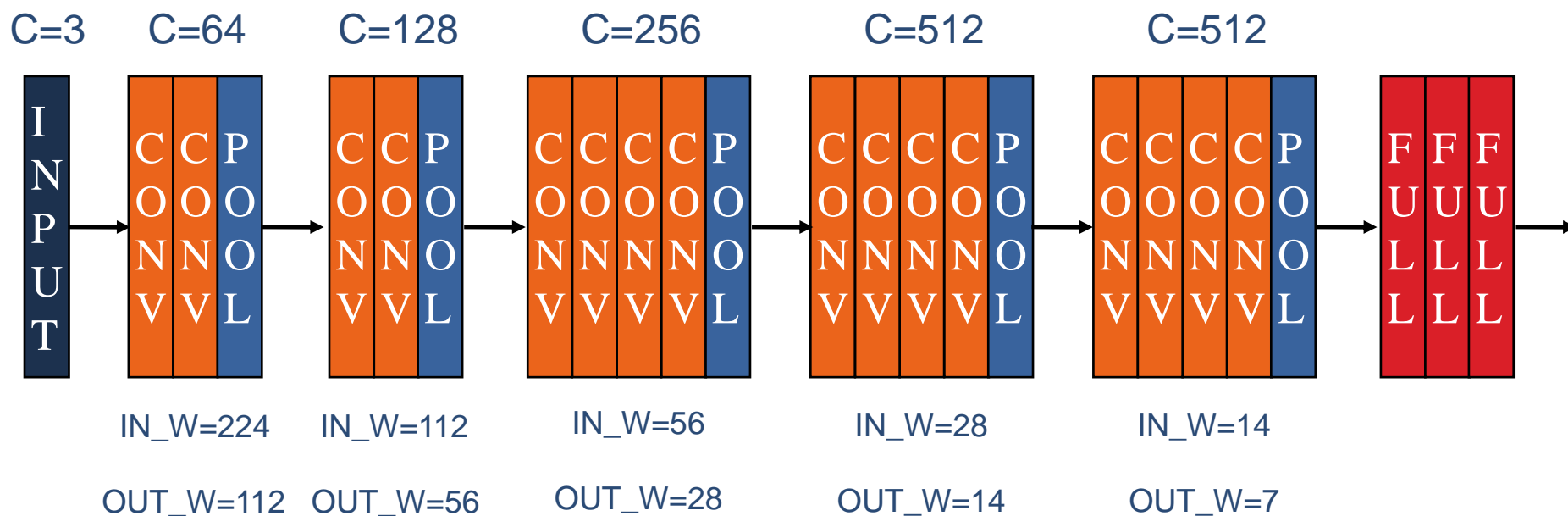
总共大约1.44亿个参数，其中全连接层大约有1.24亿个参数。

VggNet 参数数量

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

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

Vgg19 计算量



总共大约196亿Flops，计算量主要来自于各个卷积层。



3.6 使用Pytorch 实现 Vgg19 模型

小结

- ▶ 人工神经网络与计算机视觉中的关系。
- ▶ AlexNet主要贡献：ReLU激活函数、分布式GPU计算、LRN、重叠最大池化。
- ▶ VggNet主要贡献：使用较小卷积核堆叠近似较大卷积核，降低模型参数；验证了更深的层拥有更好的性能。



THANKS