### [Very Deep Convolutional Networks for Large-Scale Image Recognition](http://cn.arxiv.org/abs/1409.1556" \t "_blank)

Abstract

本文主要是研究卷积网络深度在大规模图像识别设置中的准确性的影响，主要贡献是使用非常小（3×3）卷积filter，此网络深度较大，为16-19，并表现出显著的性能提升。此网络在一些比赛中表现较好，并且可以很好的推广到其他数据集

Introduction

# ConvNet configurations

# Classification framework

# Classification experiments

# Conclusion

# Introduction

# 卷积网络最近在大规模图像和视频识别方面取得了巨大成功（Krizhevsky等，2012; Zeiler＆Fergus，2013; Sermanet等，2014; Simonyan和Zisserman，2014），这已成为可能 由于大型公共图像存储库，如ImageNet（Deng et al。，2009），以及高性能计算系统，如GPU或大规模分布式集群（Dean et al。，2012）。 特别是，ImageNet大规模视觉识别挑战赛（ILSVRC）（Russakovsky等，2014）在深度视觉识别架构的发展中发挥了重要作用，该挑战已经成为几代大规模的试验台。 图像分类系统，从高维浅层特征编码（Perronnin等，2010）（ILSVRC-2011的获胜者）到深度ConvNets（Krizhevsky等，2012）（ILSVRC-2012的获胜者）。

# 卷积网络在大规模图像和视频识别方面表现优异，这得益于大型数据库如Imagenet和高性能计算系统的发展，ImageNet大规模视觉识别挑战赛也发挥了重要作用

# 随着ConvNets成为计算机视觉领域的一种商品，已经进行了许多尝试来改进Krizhevsky的原始架构以实现更高的准确性。 例如，对ILSVRC2013的最佳表现（Zeiler＆Fergus，2013; Sermanet等，2014）利用较小的接收窗口尺寸和较小的第一卷积层步幅。 另一项改进涉及在整个图像和多个尺度上密集地训练和测试网络（Sermanet等，2014; Howard，2014）。 在本文中，我们讨论了ConvNet架构设计的另一个重要方面 - 它的深度。 为此，我们确定了架构的其他参数，并通过添加更多的卷积层来稳定地增加网络的深度，这是可行的，因为在所有层中使用非常小的（3×3）卷积滤波器。

# 随着卷积网络的使用日益广泛，有许多对其的改进。例如利用较小的接收窗口尺寸和较小的第一卷积层步长，另一项是在整个图像和多个尺度上密集地训练和测试网络，本文主要讨论网络的深度。

# 2.ConvNet configurations卷积网络配置

# 2.1 Aechitecture

# 在训练期间，我们的ConvNets输入是一个固定大小的224×224 RGB图像。我们唯一的预处理是从每个像素中减去在训练集上计算的平均RGB值。图像通过一堆卷积（转换）层，我们使用具有非常小的接收场的滤波器：3×3（这是捕获左/右，上/下，中心概念的最小尺寸） 。在其中一个配置中，我们还使用1×1卷积滤波器，可以看作是输入通道的线性变换（后面是非线性）。卷积步幅固定为1个像素;转换的空间填充。层输入使得在卷积之后保留空间分辨率，即，对于3×3转换，填充是1个像素。层。空间池是由五个最大池层执行的，这些层跟随一些转换。图层（并非所有转换图层都跟随最大池）。最大池化在2×2像素窗口上执行，步幅为2。

# 卷积网络的输入为224×224 的RGB图像，预处理是从每个像素中减去在训练集上计算的平均RGB值。使用的filter是3×3，其中一层用的是1×1，卷积步长为1，padding是same，使得卷积后的空间分辨率大小不变。池化层为最大池化，规模为2×2，步长为2

# 一堆卷积层（在不同的架构中具有不同的深度）之后是三个全连接（FC）层：前两个每层有4096个通道，第三个执行1000路ILSVRC分类，因此包含1000个通道（每个类别一个））。 最后一层是soft-max层。 全连接层的配置在所有网络中都是相同的。

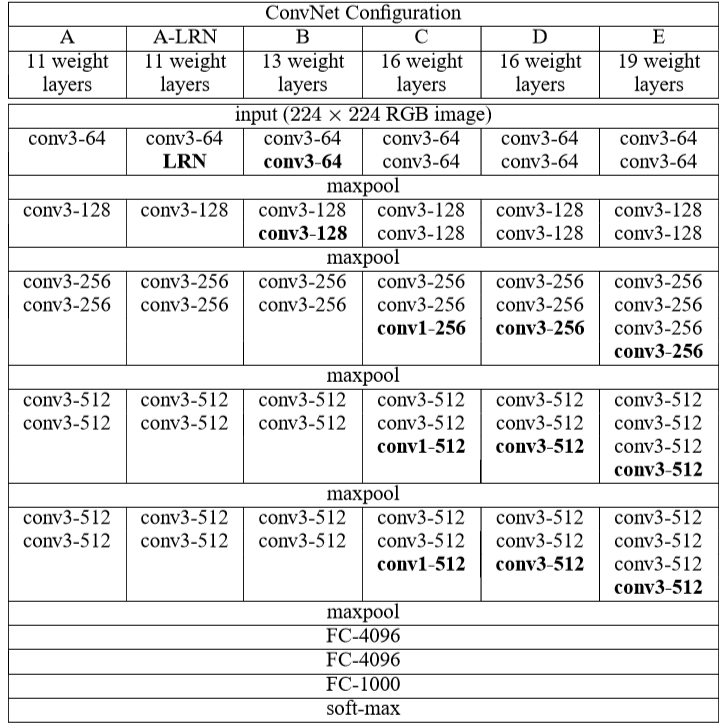
所有隐藏层都配备了Reci fi cation（ReLU（Krizhevsky等，2012））非线性。 我们注意到，我们的网络（除了一个）都没有包含本地响应规范化（LRN）规范化（Krizhevsky等，2012）：如将在Sect中所示。 如图4所示，这种标准化不会改善ILSVRC数据集的性能，但会导致内存消耗和计算时间增加。 在适用的情况下，LRN层的参数是（Krizhevsky等，2012）的参数。

卷积层后有三个全连接层， 前两个每层有4096个通道，第三个包含1000个通道。 最后一层是soft-max层。 全连接层的配置在所有网络中都是相同的。每个隐藏层后都有ReLU非线性函数。本文的网络没有局部响应归一化（LRN）,因为其无法改善性能。

2.2 Configuration

本文评估的ConvNet配置在表1中列出，每列一个。 在下文中，我们将通过名称（A-E）来引用网络。 所有配置均遵循Sect中提出的通用设计。 2.1，并且仅在深度上有所不同：从网络A中的11个权重层（8个转换层和3个FC层）到网络E中的19个权重层（16个转换层和3个FC层）。 转的宽度。 层（通道数）相当小，从第一层中的64开始，然后在每个最大池层之后增加2倍，直到达到512。

在表2中，我们报告了每个配置的参数数量。 尽管深度很大，但我们网中的权重数量不大于更大转数的更浅网中的权重数。 层宽和接收场（144M权重（Sermanet等，2014））。

本文的各种卷积网络配置均相同，只有深度不同，从网络A中的11个权重层（8个卷积层和3个全连接层）到网络E中的19个权重层（16个卷积层和3个全连接层）。每层的通道数也很小，从第一层中的64开始，然后在每个最大池层之后增加2倍，直到达到512。

2.3 discussion

两个3×3转。层（其间没有空间池）具有5×5的有效接收场;三个这样的层具有7×7有效接收场。那么我们通过使用例如三个3×3转换的堆栈获得了什么。层而不是单个7×7层？首先，我们合并了三个非线性整形层而不是一个非线性整形层，这使得决策函数更具有辨别力。其次，我们减少参数的数量：假设三层3×3卷积堆栈的输入和输出都有C通道，堆栈由332C2参数化？ = 27C2重量;同时，单个7×7转。层将需要72C2 = 49C2参数，即多81％。这可以看作是对7×7转换进行正则化。滤波器，迫使它们通过3×3滤波器进行分解（两者之间注入非线性）

2个3×3卷积层连在一起相当于一个5×5的卷积层，三个连在一起相当于单个7×7层。优点是合并了三个非线性层而不是一个非线性形层，这使得决策函数更具有判断性。其次可以减少参数数量。假设输入和输出的通道数都是C，那么3个3×3的filter的参数数量是3\*(3\*3\*c)\*c=27c\*c个参数，而使用一个7×7的filter的参数为(7\*7\*c)\*c=49c\*c个参数。

加入1×1转换。 层（配置C，表1）是一种增加决策函数非线性而不影响conv的接收域的方法。层。 即使在我们的情况下，1×1卷积基本上是在相同维度的空间上的线性投影（输入和输出通道的数量相同），但是通过整流功能引入了额外的非线性。 应该注意1×1转。 最近在Lin等人的“Network in Network”架构中使用了这些层。（2014）。

使用1×1大小的卷积，可以通过修正函数增加额外的非线性，本文的网络在1×1卷积后的输入输出维度没有改变

3. Classification framework

3.1 Training

卷积网络通过使用具有动量的小批量梯度下降优化多项逻辑回归目标来执行训练。批量大小设置为256，动量为0.9。通过权重衰减（L2惩罚乘数设置为5·10-4）和前两个完全连接层的dropout正则化（dropout率设置为0.5）对训练进行正则化。学习率初始设定为10-2，然后当验证设定精度停止改善时，学习率降低10倍。总共，学习率降低了3倍，并且在370K迭代（74个时期）之后学习被停止。我们推测，尽管与（Krizhevsky等人，2012）相比，参数数量更多，网络的深度更大，但网络需要更少的时期来收敛（a）由更大的深度和更小的转换所强加的隐式正则化。过滤器尺寸; （b）某些层的预初始化。

网络权重的初始化很重要，因为由于深网中梯度的不稳定性，不良的初始化会使学习失速。为了避免这个问题，我们开始训练配置A（表1），其足够浅以便随机初始化进行训练。然后，当训练更深层的体系结构时，我们初始化了前四个卷积层和最后三个完全连接的层与网A的层（中间层随机初始化）。我们没有降低预初始化层的学习率，允许它们在学习期间改变。对于随机初始化（如果适用），我们从具有零均值和10-2方差的正态分布中对权重进行采样。偏差初始化为零。值得注意的是，在提交论文后，我们发现通过使用Glorot＆Bengio（2010）的随机初始化程序，可以在没有预训练的情况下初始化权重。

为了获得固定大小的224×224ConvNet inputimages，他们从重新调整的训练图像中随机分割（每个SGD迭代每个图像一个作物）。为了进一步增加训练集，作物经历了随机水平移动和随机RGB色移（Krizhevsky等，2012）。下面解释训练图像重新缩放。

# 