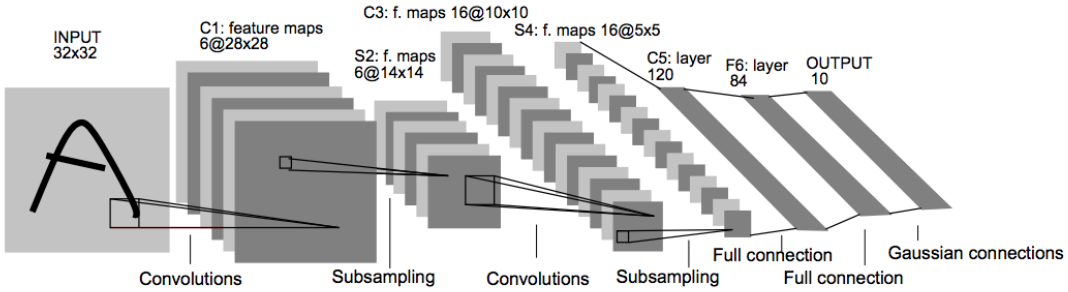
经典的神经网络介绍

本文主要针对这几年来比较流行的几个神经网络的框架和参数进行讨论分析。

LeNet

LeNet 神经网络简介：

LeNet 神经网络由深度学习三巨头之一的 Yan LeCun 提出，他同时也是卷积神经网络(CNN，Convolutional Neural Networks)之父。LeNet 主要用来进行手写字符的识别与分类，并在美国的银行中投入了使用。LeNet 的实现确立了 CNN 的结构，现在神经网络中的许多内容在 LeNet 的网络结构中都能看到，例如卷积层，Pooling 层，ReLU 层。虽然 LeNet 早在 20 世纪 90 年代就已经提出了，但由于当时缺乏大规模的训练数据，计算机硬件的性能也较低，因此 LeNet 神经网络在处理复杂问题时效果并不理想。虽然 LeNet 网络结构比较简单，但是刚好适合神经网络的入门学习，以下为 LeNet 的结构示意图。



LeNet 的网络结构

LeNet 各层的参数变化：

其具体的结构为[Conv1-Pool1-Conv2-Pool2-FC1-FC2]

Conv1：

输入大小：32*32

核大小：5*5

核数目：6

输出大小：28*28*6

训练参数数目：(5*5+1)*6=156

连接数：(5*5+1)*6*(32-2-2)*(32-2-2)=122304

Pool1:

输入大小：28*28*6

核大小：2*2

核数目：1

输出大小：14*14*6

训练参数数目：2*6=12，2=(w,b)

连接数：(2*2+1)*1*14*14*6=5880

Conv2:

输入大小：14*14*6

核大小：5*5

核数目：16

输出大小：10*10*16

训练参数数目：6*(3*5*5+1) + 6*(4*5*5+1) + 3*(4*5*5+1) + 1*(6*5*5+1)=1516

连接数：(6*(3*5*5+1) + 6*(4*5*5+1) + 3*(4*5*5+1) + 1*(6*5*5+1))*10*10=151600

Pool2：

输入大小：10*10*16

核大小：2*2

核数目：1

输出大小：5*5*16

训练参数数目：2*16=32

连接数：(2*2+1)*1*5*5*16=2000

Conv3：

输入大小：5*5*16

核大小：5*5

核数目：120

输出大小：120*1*1

训练参数数目：(5*5*16+1)*120*1*1=48120（因为是全连接）

连接数：(5*5*16+1)*120*1*1=48120

FC1：

输入大小：120

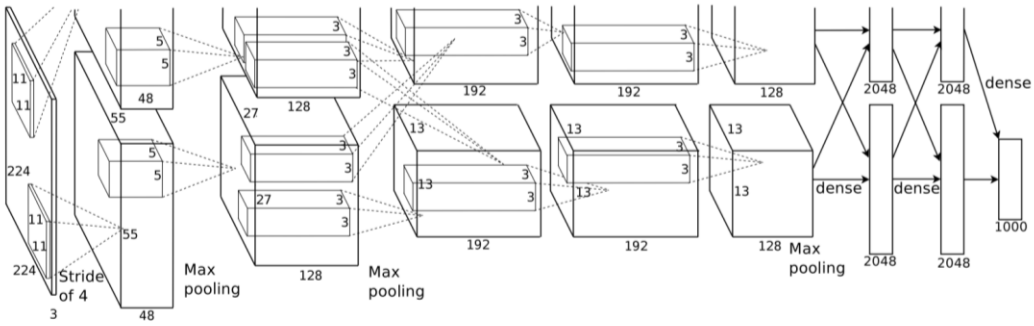
输出大小：84

训练参数数目：(120+1)*84=10164

连接数：(120+1)*84=10164

AlexNet

AlexNet 的具体结构图如下图所示：



AlexNet 的网络结构

AlexNet 和传统的 CNN（LeNet）相比具体改动的地方有数据增强、Dropout、ReLU 激活函数、Local Response Normalization、重叠的最大池化以及多 GPU 运行。

1、 数据增强（Data Augumentation）

一般数据增强的方法有水平翻转、随机剪裁、平移变换和颜色、光照变换。

2、 Dropout

Dropout 和数据增强方法一样是为了防止模型过拟合的。Dropout 应该是 AlexNet 中一个很大的创新。

3、 ReLU 激活函数

用 ReLU 激活函数代替了传统的 Tanh 或者 sigmod 函数，它的好处有：

a、 ReLU 本质上是分段线性函数，前向传播计算非常简单，无需进行指数操作之类。

b、 ReLU 的偏导非常简单，反向传播梯度，无需指数或者除法之类的操作。

c、 ReLU 不容易发生梯度发散的问题，Tanh 和 sigmoid 激活函数再两端的导数很容易趋向于零，多级连乘之后更加趋近于 0.

d、 ReLU 关闭了左边，从而使得很多的隐层的输出为 0，即网络变得稀疏，起到了类似 L1 正则化的作用，可以在一定程度上缓解了过拟合。

当然使用 ReLU 函数也是有缺点的，比如左边全部关闭之后会很容易导致某些隐藏的节点永无翻身之日，所以后来也出现了 leaky ReLU 等改进。同时 ReLU 会很容易的改变数据的分布，因此 ReLU 后面加 Batch Normalization 也是常用的改进方法。

4、 LRN(Local Response Normalization)

对局部神经元的活动创建竞争机制，使得其中响应比较大的值变得相对更大，并抑制其他反馈较小的神经元，增强了模型的泛化能力。但在后面的论文 Very Deep Convolutional Networks for Large-Scale Image Recognition.中提到这个 LRN 其实基本没什么用。

5、 重叠的最大池化

此前 CNN 中普遍使用平均池化，AlexNet 全部使用最大池化，避免平均池化的模糊化效果。并且 AlexNet 中提出让步长比池化核的尺寸小，这样池化层的输出之间会有重叠和覆盖，提升了特征的丰富性。

6、 多 GPU 运行

AlexNet 使用了两块 GTX 580 GPU 进行运算，当时也是由于 GPU 的显存比较小，一块无法把所有的参数全部装进去，使用了两块 GPU 之后也明显加快了速度。

具体的结构为:如下表所示：

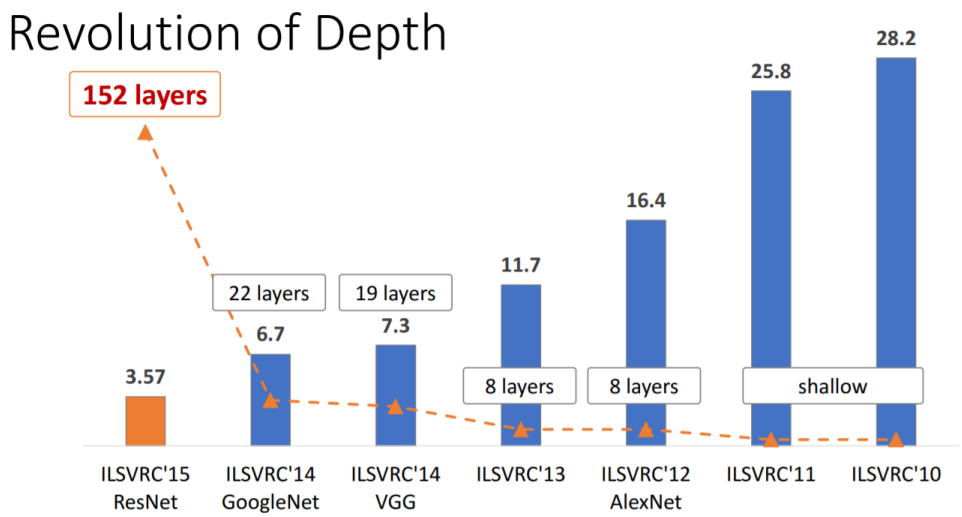
Alexnet 不同层的相应参数表

Layer	Input Shape	filters	stride	pad	Output shape	parameters
Input	227*227*3	None	None	None	227*227*3	0
Conv1	227*227*3	96 11*11	4	0	55*55*96	(11*11*3+1)*96
Max Pool1	55*55*96	3*3	2	None	27*27*96	0
Norm1	27*27*96	None	None	None	27*27*96	0
Conv2	27*27*96	256 5*5	1	2	27*27*256	(5*5*96+1)*256
Max Pool2	27*27*256	3*3	2	None	13*13*256	0
Norm2	13*13*256	None	None	None	13*13*256	0
Conv3	13*13*256	384 3*3	1	1	13*13*384	(3*3*256+1)*384
Conv4	13*13*384	384 3*3	1	1	13*13*384	(3*3*384+1)*384
Conv5	13*13*384	256 3*3	1	1	13*13*256	(3*3*384+1)*256
Max Pool3	13*13*256	3*3	2	None	6*6*256	0
FC0	6*6*256	None	None	None	9216 units	0

FC1	9216 units	None	None	None	4096 units	6*6*256*4096
FC2	4096 units	None	None	None	4096 units	4096*4096
logits	4096 units	None	None	None	1000 units	4096*4096

VGGNet

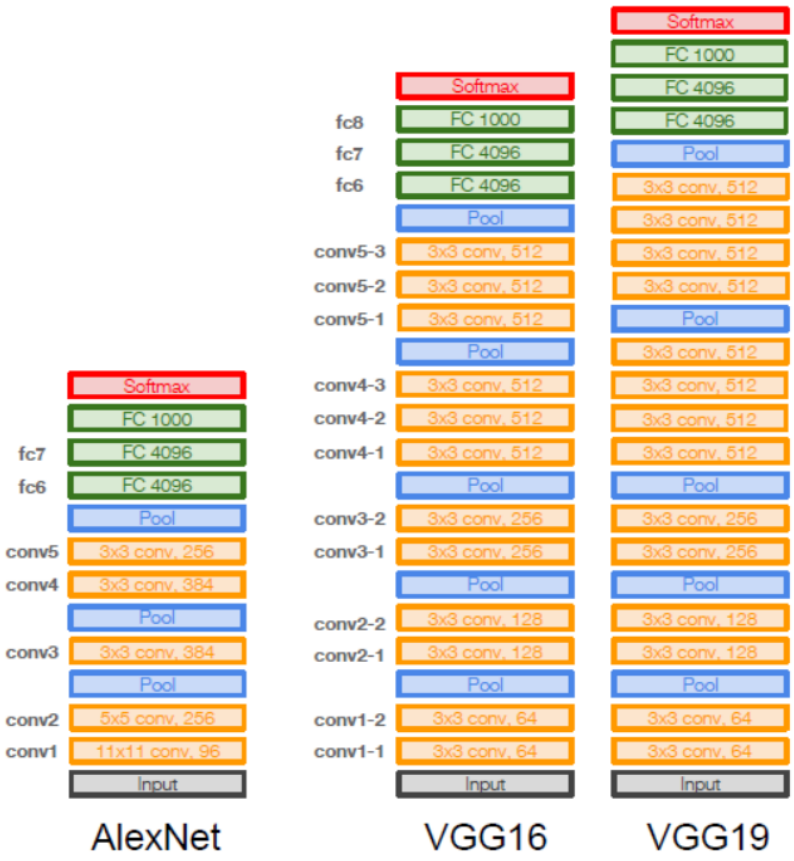
首先说明 VGGNet 的主要贡献是增加了网络结构的深度，同时使用了更小的 filter(3*3)，下图显示了从 VGG 开始网络的深度开始加深。



VGG 是在 Alex-net 的基础上发展而来的深层神经网络,如下图所示。主要修改为以下两个方面：

1、 在卷积层中使用更小的 filter 和 stride。

2、 在整个图片和 multi-scale 上训练和测试图片。



为什么几个小滤波卷积层的组合比要给大滤波器卷积层好：

假设你一层一层的重叠了 3 个 3*3 的卷积层（层与层之间有非线性激活函数）。在这个排列下，第一个卷积层中的每个神经元对输入数据体有一个 3*3 的视野。

第二个卷积层上的神经元对第一个卷积层有一个 3*3 的视野，也就是对原始的输入数据有一个 5*5 的视野，同样第三个卷积层上的神经元对第二个卷积层有 3*3 的视野，那么也就是对原始数据有一个 7*7 的视野。假设如果我们不采用这样一个 3*3 的视野，而是直接使用一个单独的 7*7 的感受野的卷积层，那么所有神经元的感受野都是 7*7，但是就有一些缺点。

首先多个卷积层与非线性的激活层交替的结构，比单一卷积层的结构更能提取出深层的更好的特征。其次，假设所有的数据有 C 个通道，那么单独的 7*7 卷积层将会包含 7*7*C=49*C 个参数，而 3 个 3*3 的卷积层组合仅有 27C 个参数。直观来说，最好选择带有小滤波器的卷积层组合，而不是用一个带有大的滤波器的卷积层。前者可以表达出输入数据中更多的强力特征，同时使用的参数也更少。唯一不足的是，在进行反向传播的时候，中间的卷积层可能会导致占用更多的内存。

下表是 VGG-16 的不同层相应参数表

Layer	Input Shape	filters	stride	pad	Output shape	parameters
Input	224*224*3	None	None	None	224*224*3	0
Conv1	224*224*3	64 3*3	1	1	224*224*64	(3*3*3+1)*64
Conv2	224*224*64	64 3*3	1	1	224*224*64	(3*3*64+1)*64
Max Pool1	224*224*64	2*2	2	None	112*112*64	0
Conv3	112*112*64	128 3*3	1	1	112*112*128	(3*3*64+1)*128
Conv4	112*112*128	128 3*3	1	1	112*112*128	(3*3*128+1)*128
Max Pool2	112*112*128	2*2	2	None	56*56*128	0
Conv5	56*56*128	256 3*3	1	1	56*56*256	(3*3*128+1)*256
Conv6	56*56*256	256 3*3	1	1	56*56*256	(3*3*256+1)*256
Conv7	56*56*256	256 3*3	1	1	56*56*256	(3*3*256+1)*256
Max Pool3	56*56*256	2*2	2	None	28*28*256	0
Conv8	28*28*256	512 3*3	1	1	28*28*512	(3*3*256+1)*512
Conv9	28*28*512	512 3*3	1	1	28*28*512	(3*3*512+1)*512
Conv10	28*28*512	512 3*3	1	1	28*28*512	(3*3*512+1)*512
Max Pool4	28*28*512	2*2	2	None	14*14*512	0
Conv11	14*14*512	512 3*3	1	1	14*14*512	(3*3*512+1)*512
Conv12	14*14*512	512 3*3	1	1	14*14*512	(3*3*512+1)*512
Conv13	14*14*512	512 3*3	1	1	14*14*512	(3*3*512+1)*512
Max Pool5	14*14*512	2*2	2	None	7*7*512	0
FC0	7*7*512	None	None	None	25088 units	0
FC1	25088 units	None	None	None	4096 units	(7*7*512+1)*4096
FC2	4096 units	None	None	None	4096 units	(4096+1)*4096
logits	4096 units	None	None	None	1000 units	(4096+1)*1000

从这张表中我们也可以看出在前面的卷积层的时候所占的内存最大，在后面全连接层的时候所要需要计算的参数最多。

总的内存：24M*4bytes ~=96MB/image 算上反向传播还要乘以 2。

总的参数：138M parameters

虽然 VGG 和 Alex-net 相比有更多的参数，更深的层次，但是 VGG 只需要很少的迭代次数就开始收敛了。主要原因为：

1、 深度和小的滤波器尺寸起到了正则化的作用

2、 一些层的 pre-initialization，pre-initialization 是网络的权值 W~(0,0.01)的高斯分布，bias 为 0；由于存在大量的 ReLu 函数，不好的权值初始值会对网络的训练影响较大。为了绕开这个问题，作者现在通过随机的方式训练最浅的网络 A；然后再训练其他网络的时候，把网络 A 的前 4 个卷积层和最后的全连层的全职当作其他网络的初始值，未赋值的中间层通过随机初始化。

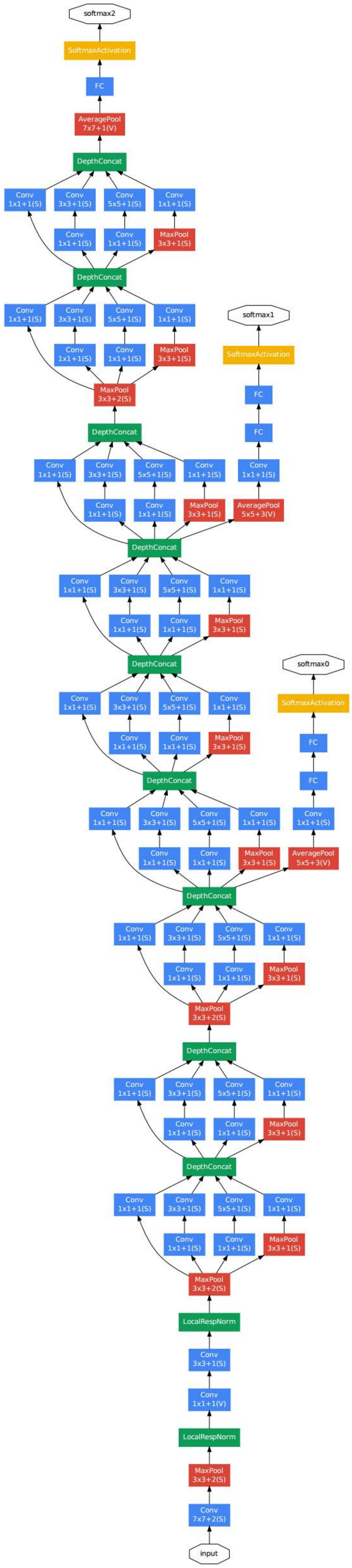
GoogLeNet

GoogLeNet 是一个 22 层的深层神经网络，该想法很简单，一方面我们人工的调整每层卷积窗口的尺寸，另一方面我们又想让神经网络的深度更深。如果我们只是想简单的加深神网络的深度和加宽神经网络的宽度会遇到两个主要的问题：

1、 随着训练模型参数的规模增大，如果我们的数据集不够大很有可能真个训练会陷入一种过拟合的状态从而很难调试

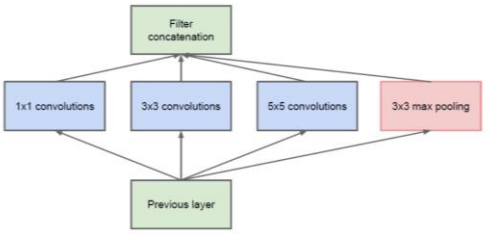
2、 大规模的神经网络会消耗大量的的计算资源。

在实际的搭建网络的过程中我们可能会比较对某一层具体是用什么类型的卷积核（1*1 or 3*3 or 5*5）或者还是添加池化层，GoogLeNet 就可以帮助我们很好的解决这一方面的问题，它在每一层都引入了池化层、1*1 卷积层、3*3 卷积层、5*5 卷积层，并将得到的结果在组合称一个新的层。同时使用了 1*1 的卷积核来减少计算量。首先我们可以看如下的一张 GoogLeNet 的结构图。

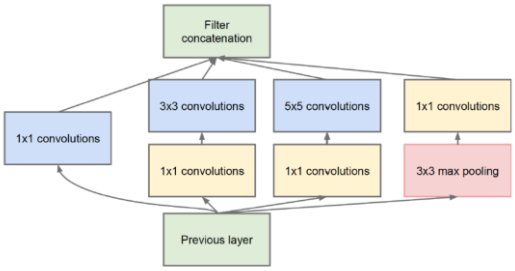

softmax2
SoftmaxActivation
FC
AveragePool 7x7+1(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
softmax1
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
SoftmaxActivation
MaxPool 3x3+2(S)
FC
DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
SoftmaxActivation
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
FC
DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 3x3+1(S)
Conv 1x1+1(V)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 7x7+2(S)
input

GoogLeNet 的 Inception 模块

很明显 Inception 模块是 GoogLeNet 的核心模块，Inception 模块设计了一种非常好的局部网络（Network within a network），然后把它们结合在一起。

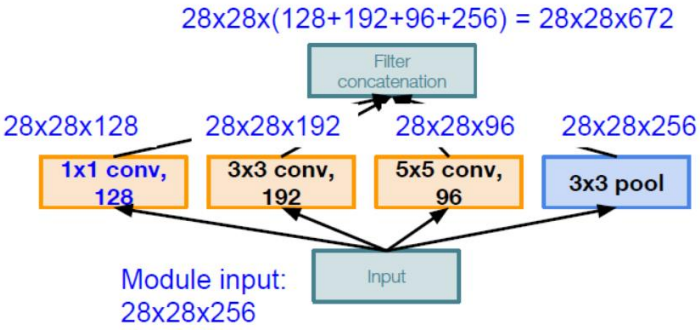


Inception 模块，原始版本



Inception 模块，维度缩减版

如上图所示，左边的图形是 Inception 模块的原始版本，原始版本和后边的维度缩减版本相比就是计算量太大，那么我们可以看一下二者的计算量的差距。



图中相应卷积操作的运算量

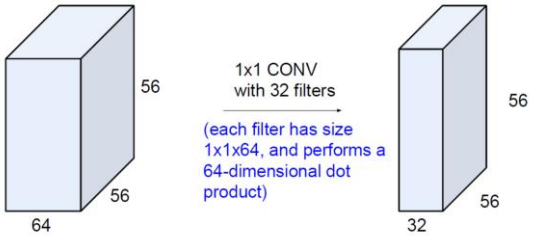
[1*1 conv, 128]　28*28*128*1*1*256

[3*3 conv, 192]　28*28*192*3*3*256

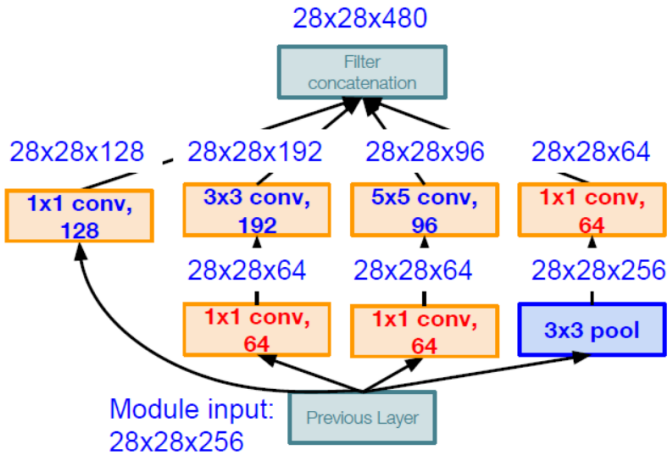
[5*5 conv, 96]　28*28*96*5*5*256

所以总共的运算量为 854M

相应的解决办法：加入一个使用 1*1 卷积的 bottleneck 层来降低特征的深度，如下图所示，我们可以将 56*56*64 的数据通过 32 个 1*1 的 filter 将其压缩到 56*56*32。



那么我们再来看应用 bottleneck 层之后的 Inception 模块的计算量



[1*1 conv, 64]　28*28*64*1*1*256

[1*1 conv, 64]　28*28*64*1*1*256

[1*1 conv, 128]　28*28*128*1*1*256

[3*3 conv, 192]　28*28*192*3*3*64

[5*5 conv, 96]　28*28*96*5*5*64

[1*1 conv, 64]　28*28*64*1*1*256

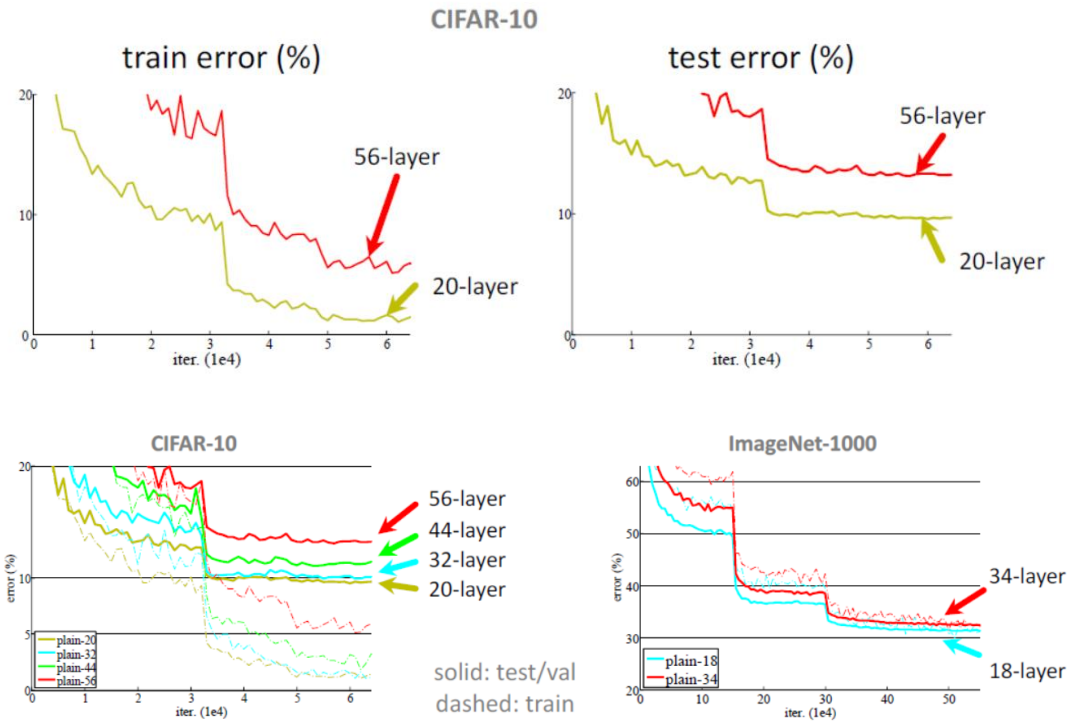
总共的运算量为 358M，比之前减少了一半多。

还有一点我们看 GoogLeNet，我们发现它在隐藏层那边还有输出两个 softmax 分类器，这样做的原因是为了防止梯度消失问题，同时也可以起到正则化的作用。在训练的时候两个旁边的分类器的损失我们定义的比重为 0.3。

ResNet

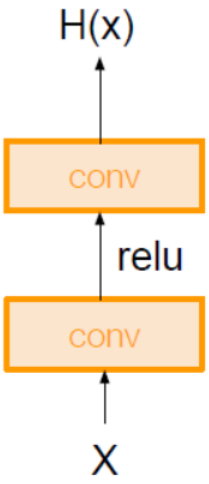
简单的增加神经层可以降低神经网络的训练误差嘛？答案是否定的，可以从以下四幅图中我们可以看出在不同的数据集中如果我们仅仅似乎简单的添加神经网络那么训练误差并没有降低反而还增加了。如果按照理论来分析，神经网络的层数越深，可覆盖的解空间越广，在理论上应该有更高的精度。但是随着网络的深度的增加，网络通常会变得越来越难训练，训练误差无法有效的传播到前面的神经网络当中，会出现比较严重的梯度消失问题。所以当网络的层数更深时，出现了“退化问题”。深层网络和浅层网络相比，训练误差

和测试误差都更大。虽然参数更多，但深层网络的问题显然不是 over-fitting，因此在训练集上表现的也很糟糕。

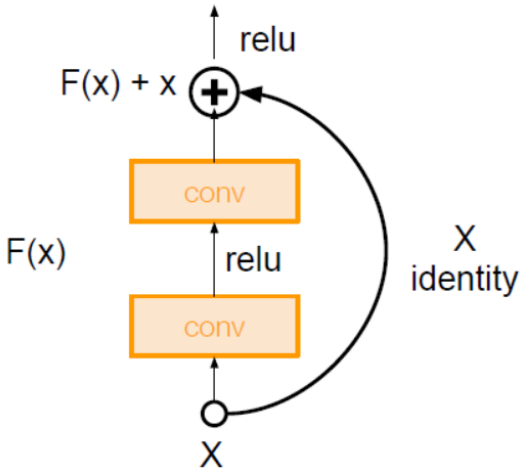


解决方案：

在传统的普通网络(Plain Network)中，一层的网络的数据来源只能时前一层网络，如图 1 所示，数据一层一层的向下流。对于卷积神经网络来说，每一层在通过卷积核后都会产生一种类似有损压缩的效果，可想而知在有损压缩到一定程度的时候，分不清原本清晰可辨的两张照片并不是什么以外的事情。这种有损压缩在实际工程中我们称之为降采样（Down sampling）——就是在向量通过网络的过程中经过一些滤波器（filter）的处理，产生的效果就是让输入向量通过降采样后具有更小的尺寸，在卷积网络中常见的就是卷积层和池化层，这两者都可以充当降采样的功能属性。主要目的时为了避免过拟合以及有一定的减少运算量的作用。在深度残差神经网络中，结构出现了比较明显的变化，如图 2 所示。



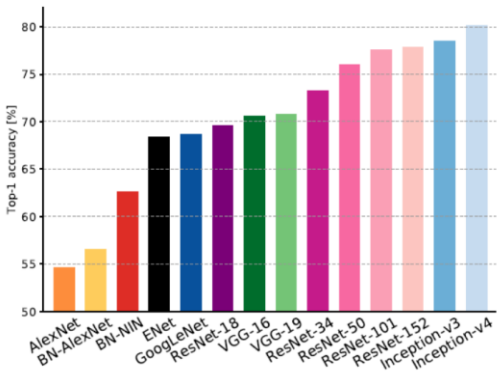
普通网络(plain network)

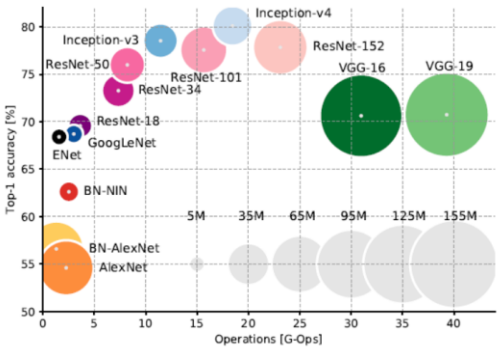


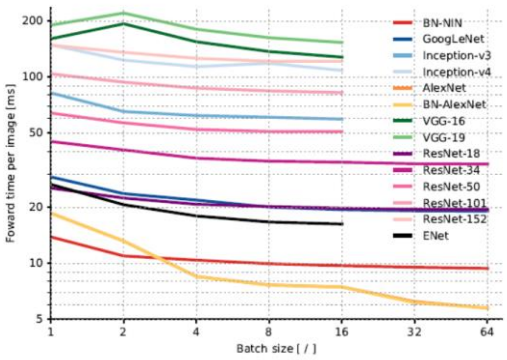
残差网络（residual network）

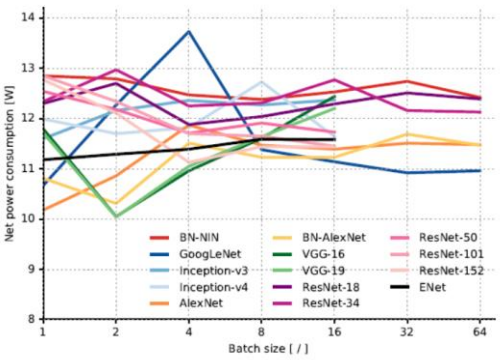
在这种情况下，会引入这种类似“短路”式的设计，将前面若干层的数据输入直接跳过多层而引入后面的数据层的输入部分，这会产生什么样的效果呢？简单的说就是前面层较为“清晰”的一些向量数据和后面进一步“有损压缩”过的数据共同作为后面的数据输入。而对比之前没有加过这个“短路”设计的普通网络来说，缺少这部分的数据参考，本身就是一种信息缺失丢失的现象本质。本来一个由 2 层网络组成的映射关系我们可以称之为 $F(x)$ 这样的一个函数进行拟合，而现在我们期望用 $H(x)=F(x)+x$ 来拟合，这本身就引入了更为丰富的参考信息或者更为丰富的维度。

几种不同模型的比较图片：






Foward time per image [ms]
Batch size [/]
BN-NIN
GoogLeNet
Inception-v3
Inception-v4
AlexNet
BN-AlexNet
VGG-16
VGG-19
ResNet-18
ResNet-34
ResNet-50
ResNet-101
ResNet-152
ENet


Net power consumption [W]
Batch size [/]
BN-NIN
GoogLeNet
Inception-v3
Inception-v4
AlexNet
BN-AlexNet
VGG-16
VGG-19
ResNet-18
ResNet-34
ResNet-50
ResNet-101
ResNet-152
ENet