GoogLeNet

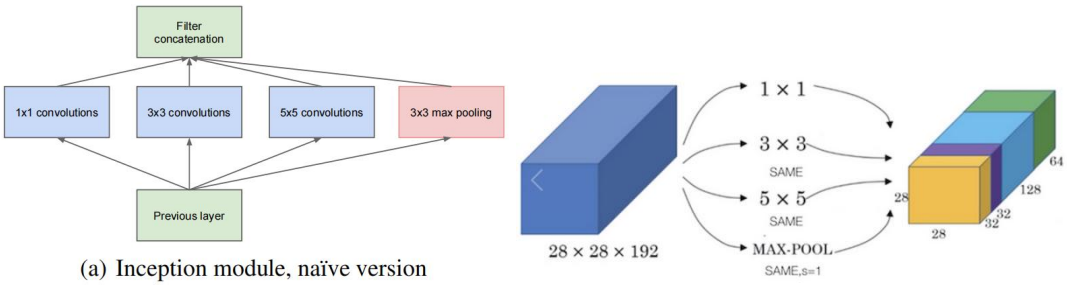
GoogLeNet 是 2014 年 Christian Szegedy 等人在 2014 年大规模视觉挑战赛(ILSVRC-2014)上使用的一种全新卷积神经网络结构，并以 6.65%的错误率力压 VGGNet 等模型取得了 ILSVRC-2014 在分类任务上的冠军，于 2015 年在 CVPR 发表了论文《Going Deeper with Convolutions》。在这之前的 AlexNet、VGG 等结构都是通过增大网络的深度（层数）来获得更好的训练效果，但层数的增加会带来很多负作用，比如 overfitting、梯度消失、梯度爆炸等，GoogLeNet 则做了更加大胆的网络结构尝试，Inception 的提出则从另一种角度来提升训练结果：能更高效的利用计算资源，在相同的计算量下能提取到更多的特征，从而提升训练结果，采用了 Inception 结构的 GoogLeNet 深度只有 22 层，其参数约为 AlexNet 的 1/12，是同时期 VGGNet 的 1/3。

GoogLeNet 是谷歌(Google)提出的深度网络结构，为什么不叫“GoogleNet”，而叫“GoogLeNet”，是为了向经典模型“LeNet”致敬

Inception 结构

Inception(盗梦空间结构)是经典模型 GoogLeNet 中最核心的子网络结构，GoogLeNet 是 Google 团队提出的一种神经网络模型，并在 2014 年 ImageNet 挑战赛(ILSVRC14)上获得了冠军。Google 团队在随后 2 年里不断改进，相继提出了 v1-v4 和 xcpetion 结构。

Inception 就是将多个卷积或池化操作放在一起组装成一个网络模块，设计神经网络时，以模块为单位去组装整个网络结构。Inception 结构设计了一个稀疏网络结构，但是能够产生稠密的数据，既能增加神经网络表现，又能保证计算资源的使用效率。



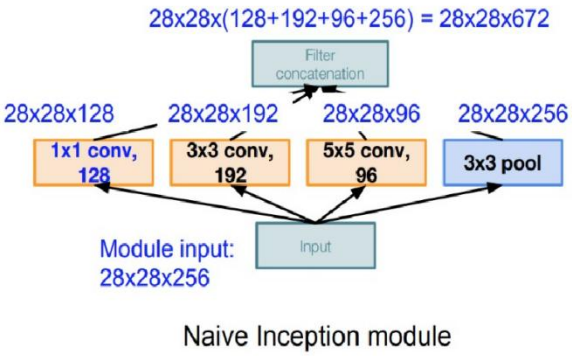
(a) Inception module, naïve version

如图所示，原始 inception 的结构就是对于同一个特征图输入，并行放置一个 1*1,3*3,5*5 和一个 3*3 池化层，让这些网络层同步对前面的特征图计算提取信息，然后将其在通道维度上合并（这也就意味着这些网络层计算输出的新特征图的长宽尺寸是一致的）。好处是可以使提取出来的特征具有多样化，并且特征

之间的 co-relationship 不会很大，最后用把 feature map 都 concatenate 起来使网络做得很宽，然后堆叠 Inception Module 将网络变深。**但仅仅简单这么做会使一层的计算量爆炸式增长**。

这里涉及到另一个问题，因为卷积层参数的数量是 size*size*in_channel*out_channel，这也就意味着 3*3 和 5*5 尺寸的卷积核在通道数多的时候参数是巨大的，

假设 input feature map 的 size 为 28 × 28 × 256 28\times28\times256 28×28×256，output feature map 的 size 为 28 × 28 × 480 28\times28\times480 28×28×480，则 native Inception Module 的计算量有 854M。计算过程如下



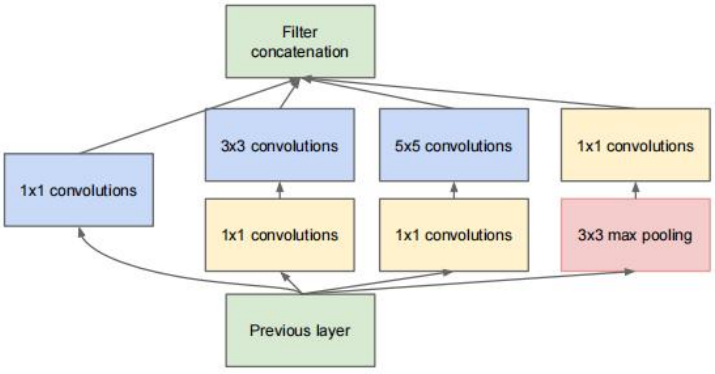
Naive Inception module

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

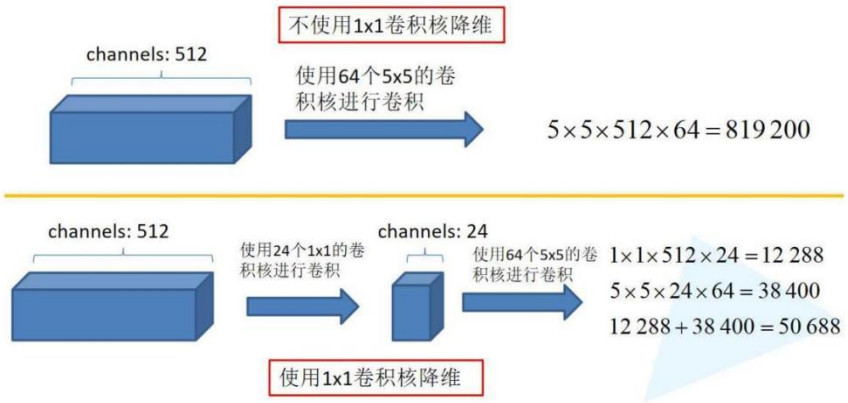
Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!

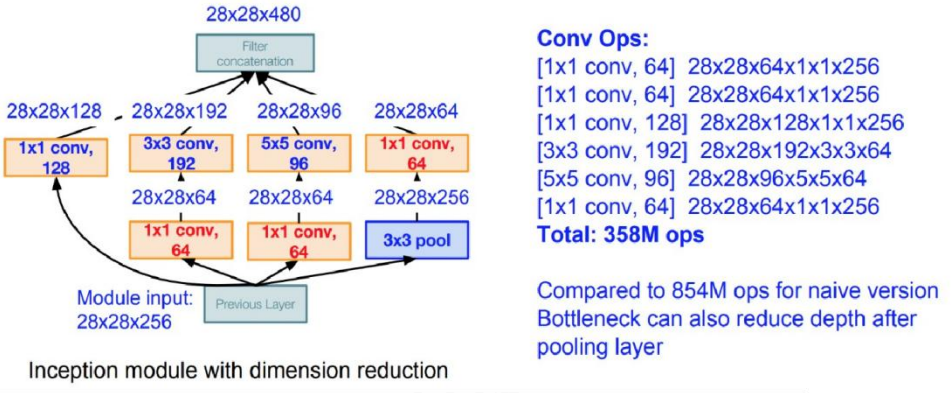
从上图可以看出，计算量主要来自高维卷积核的卷积操作，因而在每一个卷积前先使用 1×1 卷积核将输入图片的 feature map 维度先降低，进行信息压缩，在使用 3x3 卷积核进行特征提取运算，



(b) Inception module with dimension reductions



相同情况下，Inception v1 的计算量仅为 358M。



GoogLeNet 中使用了 9 个 Inception v1 module，分别被命名为 inception(3a)、inception(3b) 、 inception(4a) 、 inception(4b) 、 inception(4c) 、 inception(4d) 、inception(4e)、inception(5a)、inception(5b)。

全局平均池化

作用：若预测 K 个类别，在卷积特征抽取部分的最后一层卷积层，就会生成 K 个特征图，然后通过全局平均池化就可以得到 K 个 1×1 的特征图，将这些 1×1 的特征图输入到 softmax layer 之后，每一个输出结果代表着这 K 个类别的概率（或置信度 confidence），起到取代全连接层的效果。

优点：

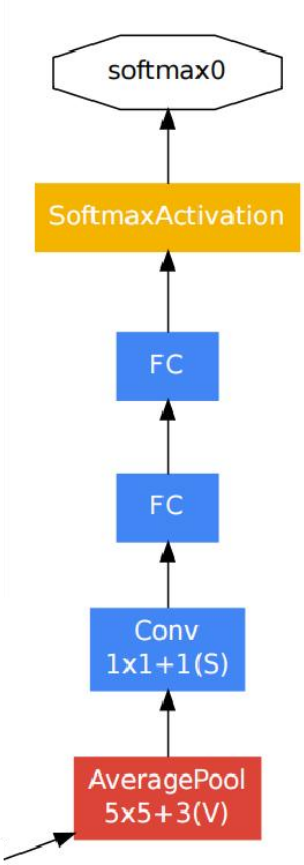
和全连接层相比，使用全局平均池化技术，对于建立特征图和类别之间的关系，是一种更朴素的卷积结构选择。

全局平均池化层不需要参数，避免在该层产生过拟合。

全局平均池化对空间信息进行求和，对输入的空间变化的鲁棒性更强。

辅助分类器

GoogLeNet 网络结构中有深层和浅层 2 个分类器，两个辅助分类器结构是一模一样的，其组成如下图所示，这两个辅助分类器的输入分别来自 Inception(4a)和 Inception(4d)。



辅助分类器的第一层是一个平均池化下采样层，池化核大小为 5x5，stride=3；第二层是卷积层，卷积核大小为 1x1，stride=1，卷积核个数是 128；第三层是全连接层，节点个数是 1024；第四层是全连接层，节点个数是 1000（对应分类的类别个数）。

在模型训练时的损失函数按照： $Loss=L_0+0.3*L_1+0.3*L_2$ Loss=L_0+0.3*L_1+0.3*L_2 Loss=L0 +0.3* L1 +0.3* L2 ， L 0 L_0 L0 是最后的分类损失。

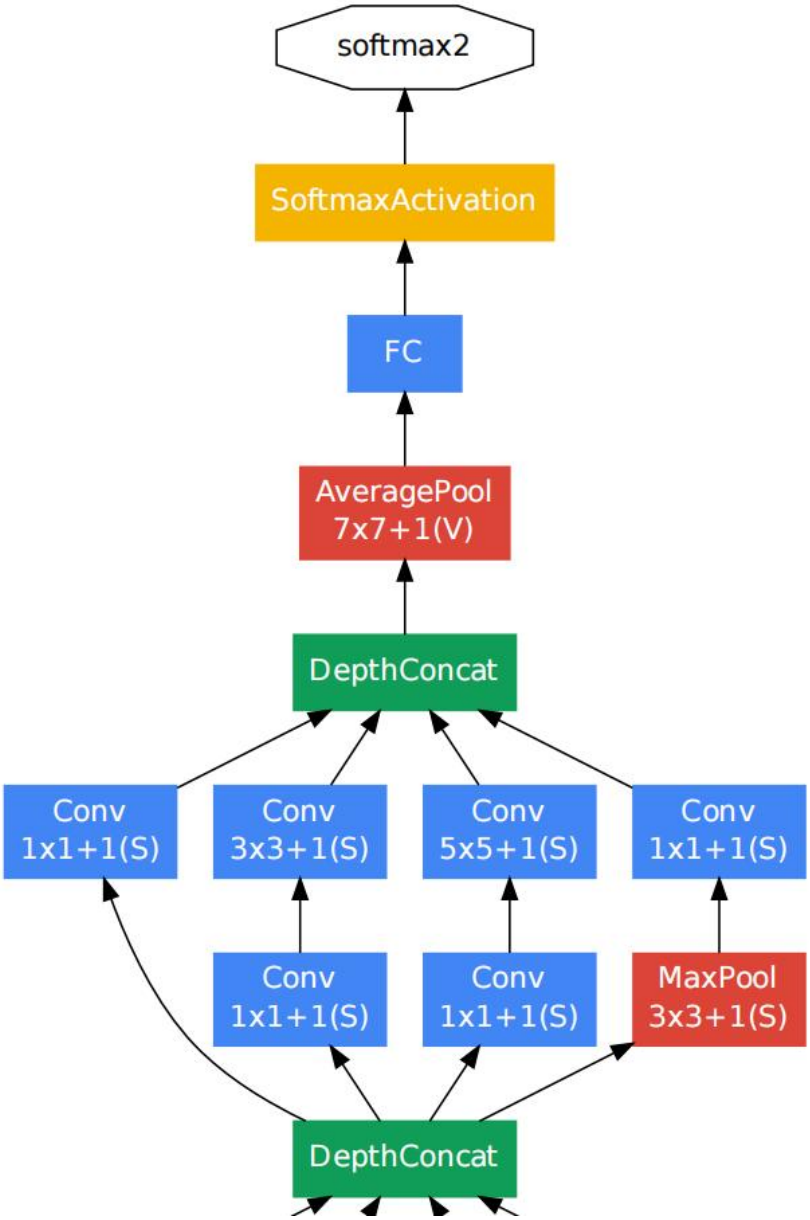
在训练模型时，将两个辅助分类器的损失乘以权重（论文中是 0.3）加到网络的整体损失上，再进行反向传播。在测试阶段则去掉辅助分类器，只记最终的分类损失。

辅助分类器的两个分支有什么用呢？

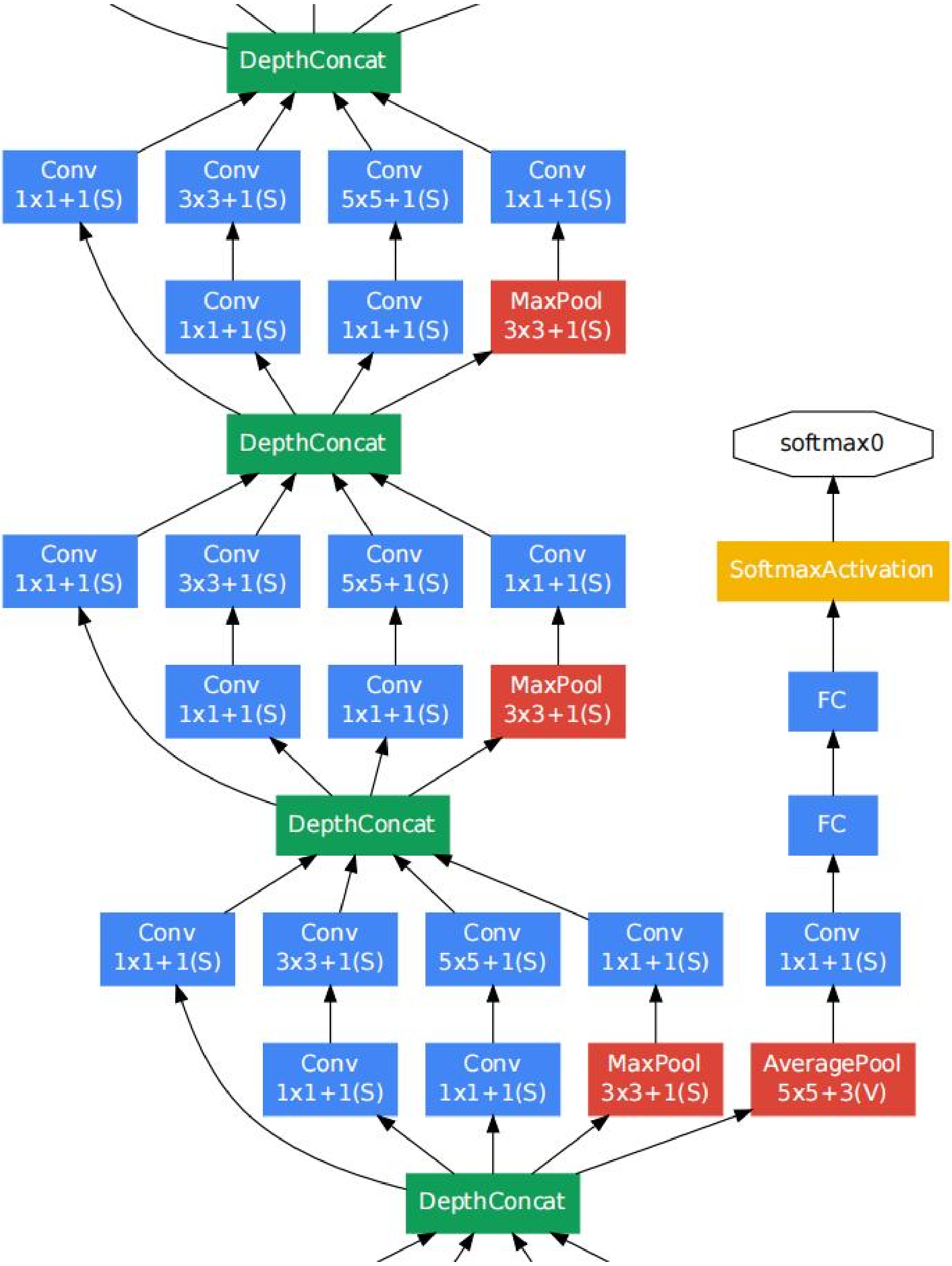
作用一：可以把它看做 inception 网络中的一个小细节，它确保了即便是隐藏单元和中间层也参与了特征计算，也能预测图片的类别，在 inception 网络中起到一种调整的效果，并且能防止网络发生过拟合。

作用二：给定深度相对较大的网络，有效传播梯度反向通过所有层的能力是一个问题。通过将辅助分类器添加到这些中间层，可以期望较低阶段分类器的判别力。在训练期间，它们的损失以折扣权重（辅助分类器损失的权重是 0.3）加到网络的整个损失上。

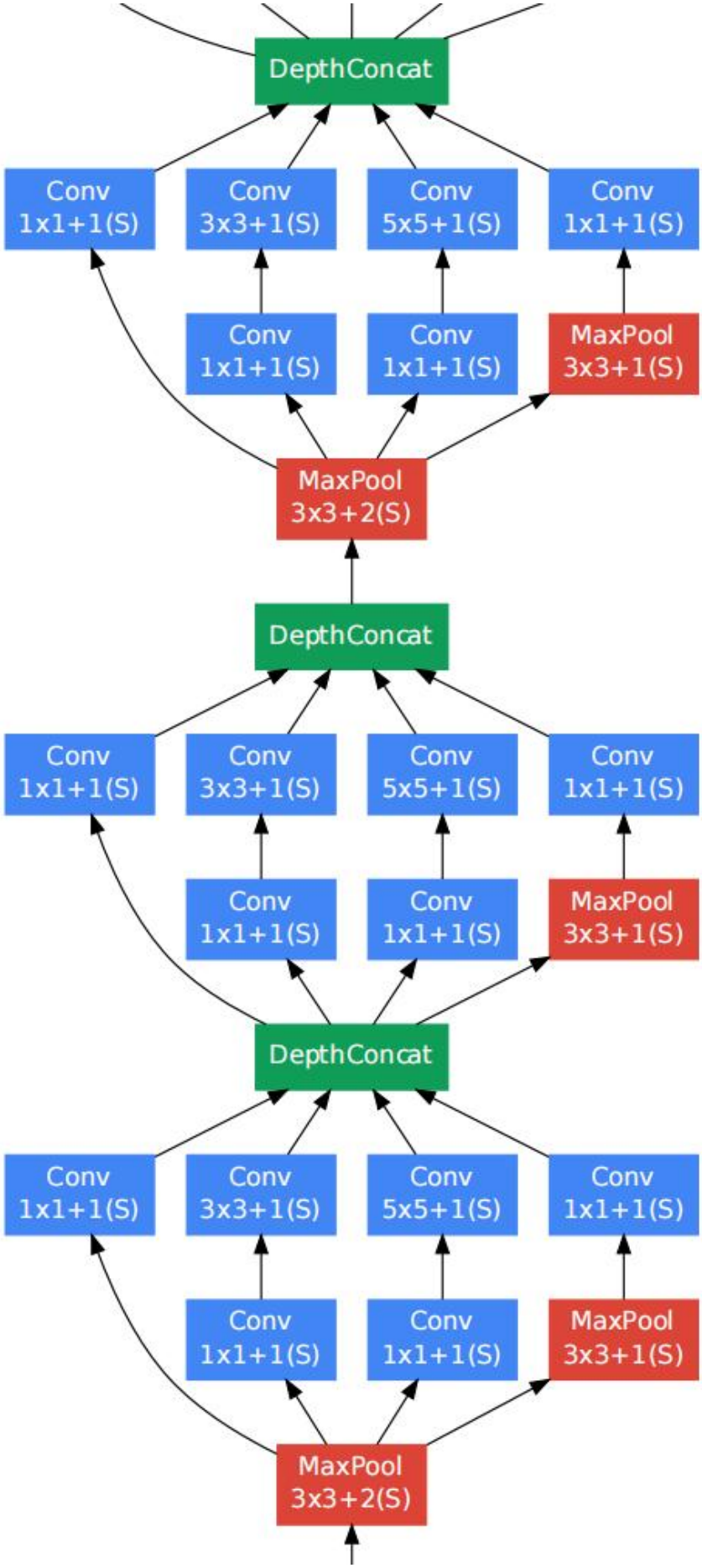
模型结构

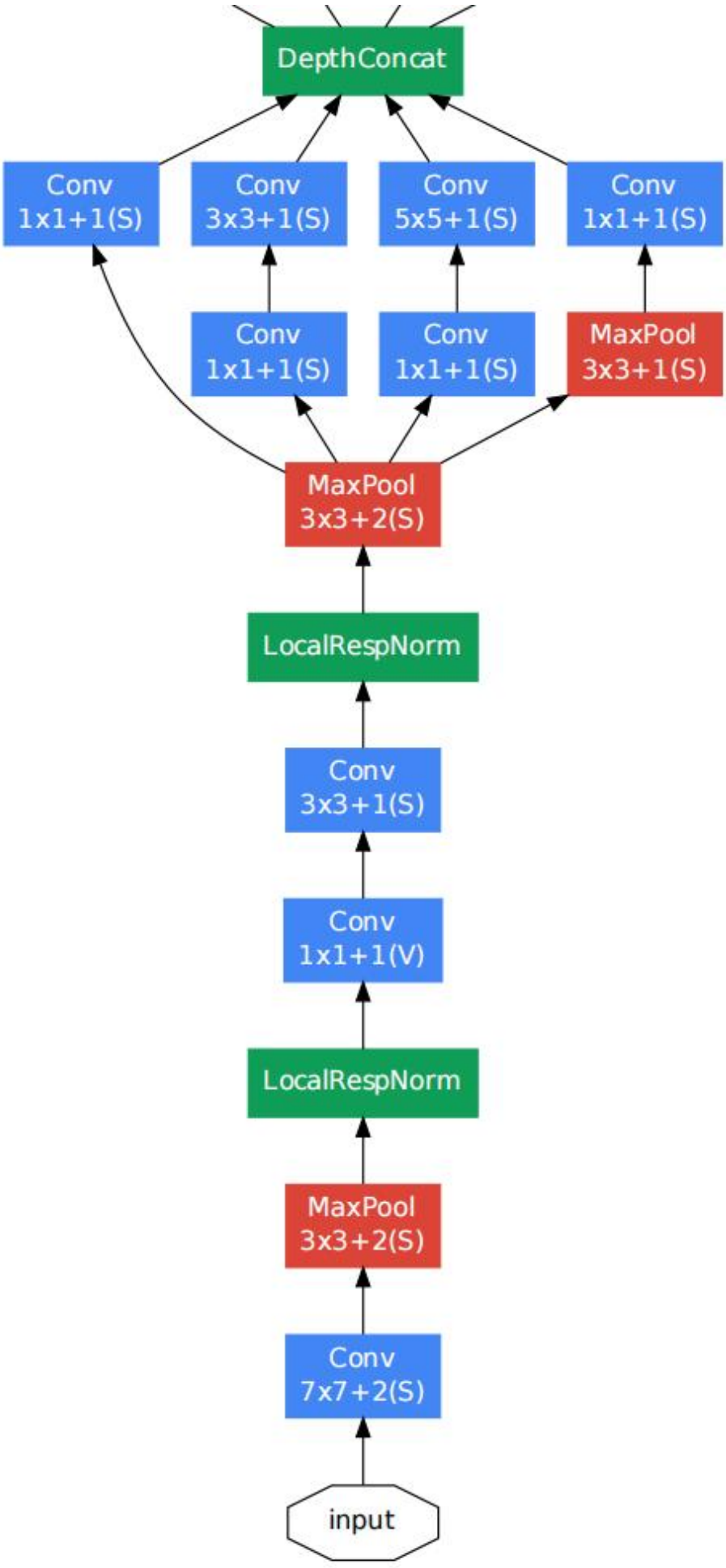



DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax1
SoftmaxActivation
FC
FC
Conv 1x1+1(S)
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


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)


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

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

- 9 个 inception 模块堆叠
- depth，共 22 层，算上池化层就 27 层
- 权重和计算量均匀分配给各层
- output size = #1×1 + #3×3 + #5×5 + pool proj
- 所有卷积都使用了修正线性激活(rectified linear activation，ReLU)
- avg pool：Global Average Pooling（全局平均池化），一个 channel 用一个平均值代表，取代 FC 层，减少参数量。优点：①便于迁移学习②top-1 提高了 0.6%的准确率
- 线性层：使网络能很容易地适应其它的标签集。

辅助分类器：

- 为了避免网络过深引起的浅层梯度消失问题

· 原理：浅层特征对于分类已经有足够的区分性
· 位于 Inception (4a)和 Inception (4b)模块的输出上
· 训练时：损失函数

- L = L 主 + 0.3 * L 辅 1 + 0.3 * L 辅 2 ； 预测时：不管这两个辅助分类器。
- 后面的控制实验表明辅助网络的影响相对较小（约 0.5），只需要其中一个就能取得同样的效果。

包括辅助分类器在内的附加网络的具体结构如下：

- 一个大小 5×5，步长 3 的平均池化层
- 具有 128 个 1×1 卷积，用于降维和修正线性激活
- 具有 1024 个单元和修正线性激活的全连接层

- 70% dropout 层
- 带有 softmax 损失的线性层作为分类器（作为主分类器预测同样的 1000 类，但在推断时移除）

模型亮点

首先说说该模型的亮点：

采用了模块化的设计（stem, stacked inception module, axuiliary function 和 classifier），方便层的添加与修改。

Stem 部分：论文指出 Inception module 要在网络中间使用的效果比较好，因此网络前半部分依旧使用传统的卷积层代替

辅助函数(Axuiliary Function)：从信息流动的角度看梯度消失，因为是梯度信息在 BP 过程中能量衰减，无法到达浅层区域，因此在中间开个口子，加个辅助损失函数直接为浅层

Classifier 部分：从 VGGNet 以及 NIN 的论文中可知，fc 层具有大量层数，因此用 average pooling 替代 fc,减少参数数量防止过拟合。在 softmax 前的 fc 之间加入 dropout，p=0.7,进一步防止过拟合。

使用 1x1 的卷积核进行降维以及映射处理 （虽然 VGG 网络中也有，但该论文介绍的更详细）。

引入了 Inception 结构（融合不同尺度的特征信息）。

丢弃全连接层，使用平均池化(average pooling)层，大大减少模型参数。

为了避免梯度消失，网络额外增加了 2 个辅助的 softmax 用于向前传导梯度（辅助分类器）。辅助分类器是将中间某一层的输出用作分类，并按一个较小的权重（0.3）加到最终分类结果中，这样相当于做了模型融合，同时给网络增加了反向传播的梯度信号，也提供了额外的正则化，对于整个网络的训练很有裨益。而在实际测试的时候，这两个额外的 softmax 会被去掉。

代码

model.py

```
import torch
import torch.nn as nn
import torch.optim
```

```
import random
import torch.nn.functional as F

#将卷积层和 relu 层封装到一起
class BasicConv2d(nn.Module):
    def __init__(self,in_channel,out_channel,**kwargs):
        super(BasicConv2d,self).__init__()

self.conv=nn.Conv2d(in_channels=in_channel,out_channels=out_channel,**kwargs)
        # ReLU(inplace=True)：将 tensor 直接修改，不找变量做中间的传递，节省运算内存，不用多存储额外的变量
        self.relu=nn.ReLU(inplace=True)
    def forward(self,x):
        x=self.conv(x)
        x=self.relu(x)
        return x

class Inception(nn.Module):
    def __init__(self,in_channels,ch1x1,ch3x3red,ch3x3, ch5x5red, ch5x5,
pool_proj):
        super(Inception,self).__init__()
        # 分支 1，单 1x1 卷积层
        self.branch1=BasicConv2d(in_channels,ch1x1,kernel_size=1)
        # 分支 2，1x1 卷积层后接 3x3 卷积层
        self.branch2=nn.Sequential(
            BasicConv2d(in_channels,ch3x3red,kernel_size=1),
            # 保证输出大小等于输入大小
            BasicConv2d(ch3x3red,ch3x3,kernel_size=3,padding=1)
        )
        # 分支 3，1x1 卷积层后接 5x5 卷积层
        self.branch3=nn.Sequential(
            BasicConv2d(in_channels,ch5x5red,kernel_size=1),
            # 保证输出大小等于输入大小
            BasicConv2d(ch5x5red,ch5x5,kernel_size=5,padding=2)
        )
        # 分支 4，3x3 最大池化层后接 1x1 卷积层
        self.branch4=nn.Sequential(
            nn.MaxPool2d(kernel_size=3,stride=1,padding=1),
            BasicConv2d(in_channels,pool_proj,kernel_size=1)
        )
    # forward()：定义前向传播过程,描述了各层之间的连接关系及数据的流动和维度变化
    def forward(self,x):
        branch1=self.branch1(x)
```

```
            branch2=self.branch2(x)
            branch3=self.branch3(x)
            branch4=self.branch4(x)
            # 在通道维上连结输出,四个维度分别是 batch_size,通道，高和宽
            outputs=[branch1,branch2,branch3,branch4]
            # cat()：在给定维度上对输入的张量序列进行连接操作，通道在第 1 维
            return    torch.cat(outputs,dim=1)
# 辅助分类器
class InceptionAux(nn.Module):
    def __init__(self,in_channels,num_classes):
            super(InceptionAux,self).__init__()
            self.averagePool=nn.AvgPool2d(kernel_size=5,stride=3)
            self.conv=BasicConv2d(in_channels,out_channel=128,kernel_size=1)
            # in_features,上一层 output[batch, 128, 4, 4]，128X4X4=2048
            self.fc1=nn.Linear(in_features=2048,out_features=1024)
            self.fc2=nn.Linear(in_features=1024,out_features=num_classes)
    def forward(self,x):
            # 输入：分类器 1：Nx512x14x14，分类器 2：Nx528x14x14
            x=self.averagePool(x)
            # 输入：分类器 1：Nx512x14x14，分类器 2：Nx528x14x14
            x=self.conv(x)
            # 输入：N x 128 x 4 x 4
            x=torch.flatten(x,1)
            # 设置.train()时为训练模式，self.training=True
            x=F.dropout(x,p=0.5,training=self.training)
            # 输入：N x 2048
            x=F.relu(self.fc1(x),inplace=True)
            x=F.dropout(x,p=0.5,training=self.training)
            # 输入：N x 1024
            x = self.fc2(x)
            # 返回值：N*num_classes
            return x

# 定义 GoogLeNet 网络模型
class GoogLeNet(nn.Module):
    # init()：进行初始化，申明模型中各层的定义
    # num_classes：需要分类的类别个数
    # aux_logits：训练过程是否使用辅助分类器，init_weights：是否对网络进行权重初始化
    def __init__(self, num_classes=1000, aux_logits=True, init_weights=False):
            super(GoogLeNet, self).__init__()
            self.aux_logits = aux_logits

            self.conv1 = BasicConv2d(3, 64, kernel_size=7, stride=2, padding=3)
```

```
        # ceil_mode=true 时，将不够池化的数据自动补足 NAN 至 kernel_size
大小
        self.maxpool1 = nn.MaxPool2d(3, stride=2, ceil_mode=True)

        self.conv2 = BasicConv2d(64, 64, kernel_size=1)
        self.conv3 = BasicConv2d(64, 192, kernel_size=3, padding=1)
        self.maxpool2 = nn.MaxPool2d(3, stride=2, ceil_mode=True)

        self.inception3a = Inception(192, 64, 96, 128, 16, 32, 32)
        self.inception3b = Inception(256, 128, 128, 192, 32, 96, 64)
        self.maxpool3 = nn.MaxPool2d(3, stride=2, ceil_mode=True)

        self.inception4a = Inception(480, 192, 96, 208, 16, 48, 64)
        self.inception4b = Inception(512, 160, 112, 224, 24, 64, 64)
        self.inception4c = Inception(512, 128, 128, 256, 24, 64, 64)
        self.inception4d = Inception(512, 112, 144, 288, 32, 64, 64)
        self.inception4e = Inception(528, 256, 160, 320, 32, 128, 128)
        self.maxpool4 = nn.MaxPool2d(3, stride=2, ceil_mode=True)

        self.inception5a = Inception(832, 256, 160, 320, 32, 128, 128)
        self.inception5b = Inception(832, 384, 192, 384, 48, 128, 128)

        # 如果为真，则使用分类器
        if self.aux_logits:
            self.aux1 = InceptionAux(512, num_classes)
            self.aux2 = InceptionAux(528, num_classes)
        # AdaptiveAvgPool2d：自适应平均池化，指定输出（H，W）
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.dropout = nn.Dropout(0.4)
        self.fc = nn.Linear(1024, num_classes)
        # 如果为真，则对网络参数进行初始化
        if init_weights:
            self._initialize_weights()

    # forward()：定义前向传播过程,描述了各层之间的连接关系
    def forward(self, x):
        # N x 3 x 224 x 224
        x = self.conv1(x)
        # N x 64 x 112 x 112
        x = self.maxpool1(x)
        # N x 64 x 56 x 56
        x = self.conv2(x)
        # N x 64 x 56 x 56
        x = self.conv3(x)
```

```
# N x 192 x 56 x 56
x = self.maxpool2(x)

# N x 192 x 28 x 28
x = self.inception3a(x)
# N x 256 x 28 x 28
x = self.inception3b(x)
# N x 480 x 28 x 28
x = self.maxpool3(x)
# N x 480 x 14 x 14
x = self.inception4a(x)
# N x 512 x 14 x 14
#  设置.train()时为训练模式，self.training=True
if self.training and self.aux_logits:
        aux1 = self.aux1(x)

x = self.inception4b(x)
# N x 512 x 14 x 14
x = self.inception4c(x)
# N x 512 x 14 x 14
x = self.inception4d(x)
# N x 528 x 14 x 14
if self.training and self.aux_logits:
        aux2 = self.aux2(x)

x = self.inception4e(x)
# N x 832 x 14 x 14
x = self.maxpool4(x)
# N x 832 x 7 x 7
x = self.inception5a(x)
# N x 832 x 7 x 7
x = self.inception5b(x)
# N x 1024 x 7 x 7

x = self.avgpool(x)
# N x 1024 x 1 x 1
x = torch.flatten(x, 1)
# N x 1024
x = self.dropout(x)
x = self.fc(x)
# N x 1000 (num_classes)
if self.training and self.aux_logits:
        return x, aux2, aux1
return x
```

```
    # 网络结构参数初始化
    def _initialize_weights(self):
        # 遍历网络中的每一层
        for m in self.modules():
            # isinstance(object, type)，如果指定的对象拥有指定的类型，则 isinstance()函数返回 True
            # 如果是卷积层
            if isinstance(m, nn.Conv2d):
                # Kaiming 正态分布方式的权重初始化
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity='relu')
                # 如果偏置不是 0，将偏置置成 0，对偏置进行初始化
                if m.bias is not None:
                    # torch.nn.init.constant_(tensor, val)，初始化整个矩阵为常数 val
                    nn.init.constant_(m.bias, 0)
            # 如果是全连接层
            elif isinstance(m, nn.Linear):
                # init.normal_(tensor, mean=0.0, std=1.0)，使用从正态分布中提取的值填充输入张量
                # 参数：tensor：一个 n 维 Tensor，mean：正态分布的平均值，std：正态分布的标准差
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.constant_(m.bias, 0)


if __name__=="__main__":
    x = torch.randn([1, 3, 224, 224])
    # [3, 4, 6, 3] 等则代表了 bolck 的重复堆叠次数
    # blocks_num=[3,4,6,3]
    # model=ResNet(BasicBlock,blocks_num,num_classes=7)
#   blocks_num = [3, 4, 6, 3]
    model = GoogLeNet(num_classes=7)
    y = model(x)
    #print(y.size())
    print(model)
```

参考资料

[1] https://zh.d2l.ai/index.html
[2] https://blog.csdn.net/weixin_44772440/article/details/122952961
[3] https://zhuanlan.zhihu.com/p/54289848
[4] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., & Anguelov, D. & Rabinovich, A.(2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).