回到梦的起点：GoogLeNet 的复现和理解

陈宇航*，莫益军†

华中科技大学 计算机科学与技术学院

【摘　要】 在 2010 年代初，有多个深度学习神经网络被提出，他们都有各自极为突出的特点和优势。比如，Yann LeCun 等人的 LeNet-5[1]，是最早期的深度卷积神经网络之一，用于手写数字识别。它包含了卷积层（Convolutional Layer）、池化层（Pooling Layer）和全连接层（Fully Connected Layer），为后来的卷积神经网络架构提供了基础；Alex Krizhevsky 等人提出的 AlexNet[2]，它的创新之处在于使用了 ReLU 激活函数、数据增强、Dropout 技术和 GPU 加速训练；Simonyan, Zisserman 等人提出的 VGGNet[3]，通过使用更小的卷积核（3x3）和堆叠多个卷积层来增加网络的深度。然而 Szegedy 等人提出来的 GoogLeNet[4]，由于引入 Inception 板块，导致它具有一共 100 层的卷积层和约 22 层的的网络层，这个模块即提高了计算精度又提高了运算速度，这让他在首次被提出的时候震惊世界。GoogLeNet 的提出是深度学习领域的重要突破，尤其是在提升计算效率的同时保持高精度。同时，它的设计理念为后续的架构提供了重大灵感支持，也对我们理解深度学习卷积神经如何运行有很有利的理解。

【关键词】 GoogLeNet, **卷积神经网络**（CNN）, **池化层**（Pooling）

1　引 言

由于深度学习的进展，特别是卷积神经网络（Convolutional Networks），图像识别和目标检测的质量取得了飞速的提升。一个令人鼓舞的消息是，大多数进展不仅仅是更强大硬件、更大数据集和更大模型的结果，更主要是新的理念、算法和改进的网络架构的成果。例如，在 ILSVRC 2014 竞赛中，除了用于检测目的的分类数据集外，排名靠前的参赛者并未使用新的数据源。GoogLeNet 提交实际上比两年前 Krizhevsky 等人的获胜架构使用了 12 倍更少的参数，同时在准确性上有显著提升。目标检测领域的最大进展并非仅来自深度网络或更大模型的应用，而是深度架构与经典计算机视觉算法（如 Girshick 等人的 R-CNN 算法）的协同作用

另一个值得注意的因素是，随着移动和嵌入式计算的持续发展，算法的效率——尤其是其功耗和内存使用——变得越来越重要。值得一提的是，GoogLeNet 深度架构设计考虑了这一因素，而不是单纯地关注准确度数字。

GoogLeNet 的高效深度神经网络架构，代号 Inception：首先，它表示我们引入了一种新的组织结构形式——”Inception 模块”；其次，它指的是网络深度的增加。一般来说，可以将 Inception 模型视为 NIN 的逻辑延伸，同时受到 Arora 等人的理论工作的启发和指导。该架构的优势在 ILSVRC 2014 分类和检测挑战中得到了实验验证，并显著超越了当前的技术水平。

2　动 机

提高深度神经网络性能的最直接方法是增加其规模。这包括增加网络的深度——层数——以及宽度：每一层的单元数。这是一种简单且安全的训练更高质量模型的方法，尤其是在有大量标注训练数据的情况下。然而，这种简单的解决方案有两个主要的缺点。

首先，增大规模通常意味着更多的参数，这使得扩展后的网络更容易发生过拟合，尤其是在训练集中的标注样本有限的情况下。这可能成为一个主要的瓶颈，因为创建高质量的训练集可能是棘手且昂贵的，尤其是如果需要专家人工标注来区分像 ImageNet 中那些细粒度的视觉类别（即使是在 1000 类 ILSVRC 子集中的类别），如图1所示。

报告时间: 2024-12-10

† **指导教师**

***学号**：U202315752

图 1 分类挑战中的两个不同类别

另一个缺点是，在深度上增加网络规模会导致计算资源的急剧增加。例如，在深度视觉网络中，如果两个卷积层是串联的，任何均匀增加它们滤波器数量的操作都会导致计算量的平方增加。如果增加的计算能力没有得到有效利用（例如，如果大部分权重接近零），那么大量计算就会被浪费。由于在实际应用中计算预算总是有限的，因此计算资源的有效分配优于无差别地增加网络规模，即使主要目标是提高结果的质量

在过往的研究中，包括 LeNet,AlexNet 等模型[5]，尽管已经取得了及其优秀的成果，但是他们都有一个局限性。他们仅仅用来多种卷积层和池化层在纵向上进行逻辑上的拓展，这当然会导致更好的成果，但是也会导致更多的参数和超参数计算。这就导致如果我们继续在纵向上加深层数，参数会大到无法计算。所以，GoogLeNet 在宽度上进行拓展，而且宽度上的层也都参考了上述的四个模型

不过，尽管 GoogLeNet 架构在计算机视觉中取得了成功，但仍然疑惑的是构建 Inception 的原则，包括在论文中也没有详细提出 Inception 层具有的特定参数是如何得到的。我们只能假设，Inception 的特点参数也是作为超参数训练的结果。所以，我们能否设计一种自动化算法构建的自动化系统去自动计算类似 Inception 层的参数，以得到可以得到的更加优化的网络流拓扑 (如自动化获得 Inception 的 V3 和 V4 优化版本[6])。至少，Inception 架构的初步成功为未来朝这一方向进行激动人心的工作提供了可取的灵感。

3 代码模块分析

3.1 Inception 模块

Inception 模块使用不同窗口和大小的卷积层，四个窗口各有各自的功能。在一个 Inception 模块中，输入和输出在高宽上是等同的，不同的是通道数量。通过四个窗口在不同层面获取信息，然后在输出通道维合并。如图2所示，蓝色块是用来抽取信息，而由于抽取信息的卷积网络的计算复杂度和通道数量相关，所以白色块的作用是为后续卷积网络的处理降低运算复杂。对于比赛和论文中使用的输入（192x28x28），1x1 的卷积网络只在通道上降低维度以获得通道信息，通道数降低到 64，而不获取宽高信息；白色块的 1x1 卷积网络也不获取信息，只用于减少通道数目，为了平衡总体效率，一个以降低到 96 的通道维度传给 3x3，pad 为 1 的卷积层，最后得到 128 层通道；另一个以降低到 16 的通道维度传给 5x5，pad 为 1 的卷积层，最后得到 32 层通道。这里两个高维度卷积层是为了获取宽高信息，同时，由于预处理减少了通道数量，仅仅使用一个窗口处理宽高信息可能导致信息的丢失，并采取多样卷积层保证输入处理不好重复处理同一块内容；3x3 的最大池化层获取宽高信息，作用于鲁棒性，保证数据不会在微调过程中出现不兼容问题，这一窗口最后得到 32 通道的结果。最后，输出模块把四个窗口各自的结果链接实现输出。

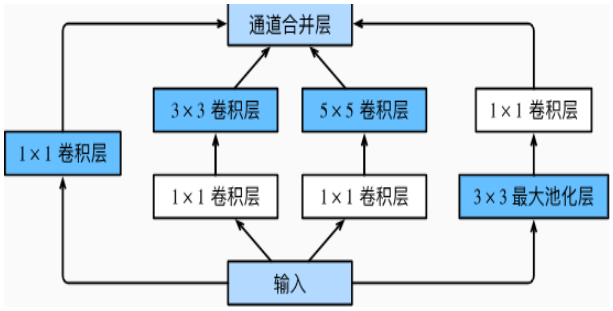

图 2 Inception 块的结构

3.2 GoogLeNet 结构

GoogLeNet 只有 22 层结构，但是却用了上百个卷积网络。其中大部分用在 Inception 层。GoogLeNet 除了 22 层结构外，为了加速输出和问题复杂性可选择性，实际上会有三个出口以应对不同的输入数据和需求，如图3

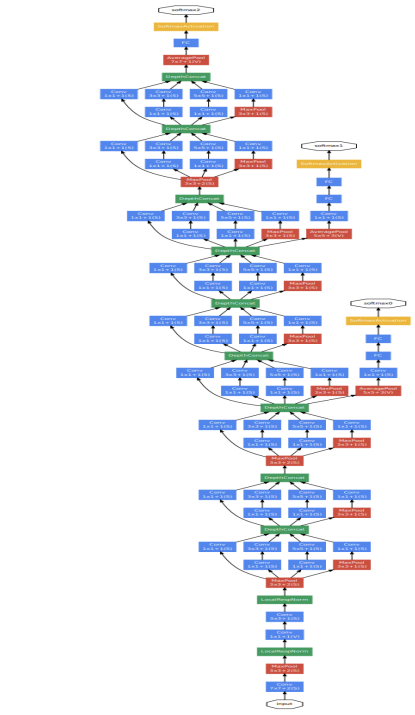

Figure 3: GoogLeNet network with all the bells and whistles

图 3 GoogLeNet 原论文配图

这里为了分析，我们只分析最长的层结构，如图4。在另外两个分支，只在最后一步的平均化池改为 5x5，因为数据通道数还没有到最终时那么大。最深层的平均化池为 7x7，因为数据通道量太大，必须这样处理得到合理的数据

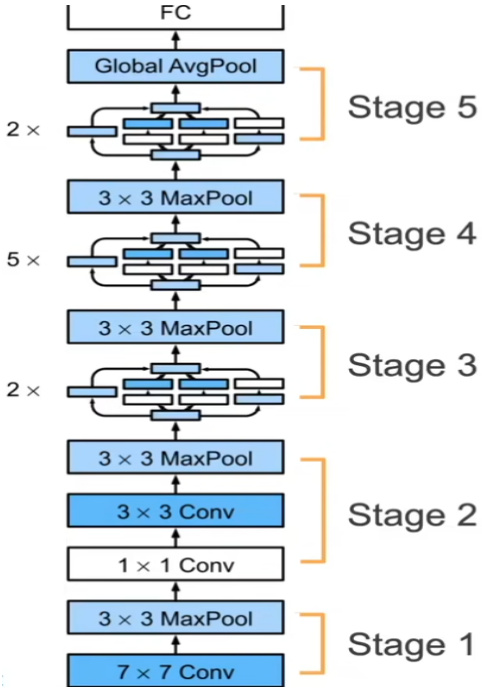

图 4 GoogLeNet 分析结构

GoogLeNet 的总体结构参考了 VGGNet 的结构，也是有五个阶段（stage），每一个阶段的划分是以高宽减半分类。第一个阶段和 VGGNet 相同，用于减少输入数据处理量，并且提高鲁棒性；第二阶段和 VGGNet 也大体类似，先用 1x1 的卷积层降低通道数，再配合 3x3 的最大化池（Max Pooling）和卷积层提高鲁棒性，降低高宽。后续使用了大量的 Inception 块进行通道减少处理，通过最大化池和平均池（Global AvgPool）进行高宽减少和链接平均。当做完平均池化和全连接层之后，获得了一个指明分类的向量。

3.2.1 阶段 1 与阶段 2

阶段 1 与阶段 2 的示意图如图5所示。相较于 AlexNet，GoogLeNet 的处理较为缓慢，目的是用更小的卷积层，保留更多的高宽保留信息，减少计算量，并且可以取得更深的网络。

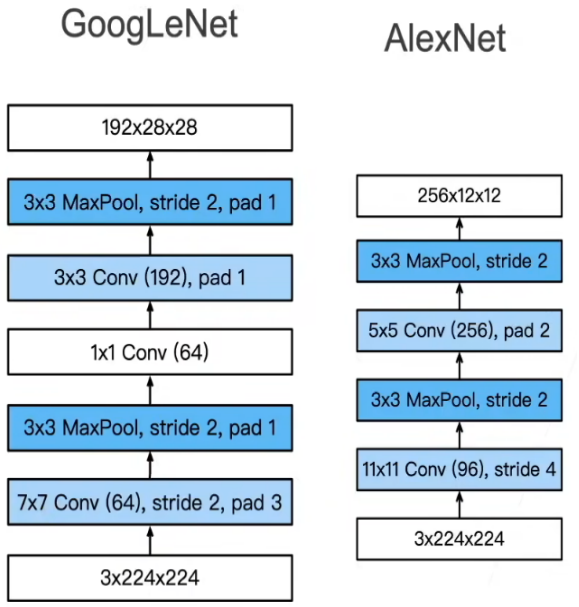

图 5 stage1 和 2

3.2.2 阶段 3，阶段 4 和阶段 5

后续的阶段实际上就是 Inception 块的组合，每一个阶段结束都是宽和高减半，每一个通道的分配不同，输出的通道增加。但是，每一个通道分配的数据并没有规律可言，我们认为他们是进行了大量的超参数搜索确定的数据分配。也因为这个原因，GoogLeNet 论文复现中对于超参数的获取极其难以实现。的如图6所示，是阶段 3 的超参数分配示意图

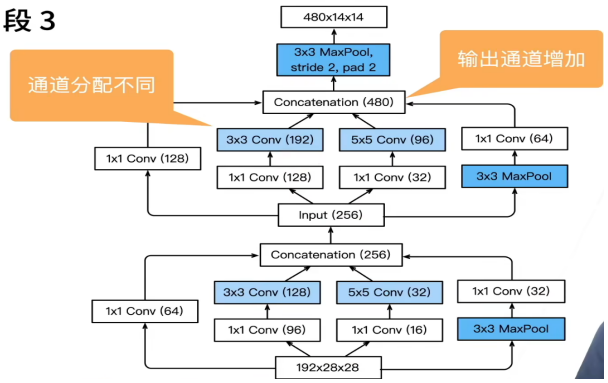

图 6 stage3

4 代码复现和实验

代码复现使用 Jupyter Notebook 平台，涉及到的包为 pytorch 2.0.1，numpy 1.23.4 和 torchinfo

4.1 卷积层实现

由于 Inception 层包含有多个卷积块，我们首先实现卷积层。如图7所示。在 GoogLnet 模型中，偏差 bias 是不存在的（一直为 False），我们设置学习率为 0.01 获得更多精度，ReLu 函数的 inplace 设置为 True

```
class BasicConv2d(nn.Module):

    def __init__(self, in_channels, out_channels, **kwargs):
        super(BasicConv2d, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, bias=False, **kwargs)
        self.bn = nn.BatchNorm2d(out_channels, eps=0.001)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)
        return F.relu(x, inplace=True)
```

```
BasicConv2d(3,5,kernel_size=3)# kernel_size是卷积层内核数 填入了一次kernel_size会代入**kwargs，后面调用就不用了必填了
```

```
BasicConv2d(
  (conv): Conv2d(3, 5, kernel_size=(3, 3), stride=(1, 1), bias=False)
  (bn): BatchNorm2d(5, eps=0.001, momentum=0.1, affine=True, track_running_stats=True)
)
```

图 7 卷积层实现

4.2 Inception 层实现

Inception 块的实现实际上就是卷积层的叠加。在前面的分析中，我们已经知道了 Inception 块的构建方法和构建逻辑 (图2)。在代码逻辑中，我们要做的仅仅只是把他们搭建起来即可。这里涉及到的函数除了我们上述实现的 BasicConv2d 函数，还要使用 torch 包中自带的 nn.Sequential(用于将多个神经网络合并) 和 nn.MaxPool2d（用于最大池化的实现），如图8所示。

```
def __init__(self, in_channels, ch1x1, ch3x3red, ch3x3, ch5x5red, ch5x5, pool_proj,
             conv_block=None):
    super(Inception, self).__init__()
    if conv_block is None:
        conv_block = BasicConv2d
    self.branch1 = conv_block(in_channels, ch1x1, kernel_size=1)

    self.branch2 = nn.Sequential(
        conv_block(in_channels, ch3x3red, kernel_size=1),
        conv_block(ch3x3red, ch3x3, kernel_size=3, padding=1)
    )

    self.branch3 = nn.Sequential(
        conv_block(in_channels, ch5x5red, kernel_size=1),
        # Here, kernel_size=3 instead of kernel_size=5 is a known bug.
        # Please see https://github.com/pytorch/vision/issues/906 for details.
        conv_block(ch5x5red, ch5x5, kernel_size=3, padding=1)
    )

    self.branch4 = nn.Sequential(
        nn.MaxPool2d(kernel_size=3, stride=1, padding=1, ceil_mode=True),
        conv_block(in_channels, pool_proj, kernel_size=1)
    )
```

图 8 Inception 块四个窗口实现

在定义整个 Inception 块的输出传递部分（forward 用法）的时候，需要我们用 torch.cat 将第一维的输出（output）进行连接，如图9所示

```
def _forward(self, x):
    branch1 = self.branch1(x)
    branch2 = self.branch2(x)
    branch3 = self.branch3(x)
    branch4 = self.branch4(x)

    outputs = [branch1, branch2, branch3, branch4]
    return outputs

def forward(self, x):
    outputs = self._forward(x)
    return torch.cat(outputs, 1)
```

图 9 forward 用法

4.3 Inception 层的测试

作为 GoogLeNet 网络中最为重要的模块，我们根据论文表格数据对它进行测试。GoogLeNet 中有一个参考的数据输入输出表格图10。参考这一表格，我们设计测试数据的大小, 选用表格中【Inception3a】的数据进行测试

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								

图 10 参考数据结构

通过11和图10的比较，我们确认代码复现成功，输出的 shape 和下一层网络可以对应

```
inception3a = Inception(192,64,96,128,16,32,32)
```

```
data = torch.ones(30,192,28,28)
```

```
inception3a(data).shape
```

```
torch.Size([30, 256, 28, 28])
```

图 11 测试数据和结果

4.4 辅助分类器实现

根据实验数据，发现神经网络的中间层也具有很强的识别能力，为了利用中间层抽象的特征，在某些中间层中添加含有多层的分类器，实现如图12。也就是上述的可以实现中间层即输出的部分。它的存在一是为了避免梯度消失，用于向前传导梯度。反向传播时如果有一层求导为 0，链式求导结果则为 0；二是将中间某一层输出用作分类，起到模型融合作用。实际测试时，这两个辅助 softmax 分支会被去掉。

```
class AuxClf(nn.Module):
    def __init__(self
                 ,in_channels
                 ,num_classes
                 ,**kwargs):
        super().__init__()
        self.feature_=nn.Sequential(nn.AvgPool2d(kernel_size=5,stride=3)
                                    ,BasicConv2d(in_channels,128,kernel_size=1))
        self.clf_=nn.Sequential(nn.Linear(4*4*128,1024)
                                ,nn.ReLU(inplace=True)
                                ,nn.Dropout(0.7)
                                ,nn.Linear(1024,num_classes))
        def forward(self,x):
            x=self.feature_(x)
            x=x.view(-1,4*4*128)
            x=self.clf_(x)
            return x
```

图 12 辅助分类器

通过图10中的参考输出，我们利用在【Inception4a】输出的数据结构进行测试，如图13所示

```
# 4a
```

```
AuxClf(512,1000)
```

```
AuxClf(
  (feature_): Sequential(
    (0): AvgPool2d(kernel_size=5, stride=3, padding=0)
    (1): BasicConv2d(
      (conv): Sequential(
        (0): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (2): ReLU(inplace=True)
      )
    )
  )
  (clf_): Sequential(
    (0): Linear(in_features=2048, out_features=1024, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.7, inplace=False)
    (3): Linear(in_features=1024, out_features=1000, bias=True)
  )
)
```

图 13 辅助分类器测试结果

。

4.5 GoogLet 搭建

完成了上述的几个主要模块的搭建后，GoogLeNet 的搭建将会是搭积木的工程。由于代码过长，我们将代码放在结尾的附录中，我们仅在此说明一些细节。(a)$Kernel\_size$ 和 $padding$ 有对应关系，为了控制输出的大小不变，他们的对应规律是 $2 * padding = Kernel\_size$ (b) 在计算过程中，遇到浮点数结果取他们的上界，使用 $ceil\_mode = True$ 实现 (c) 最后的平均适应池化只需要一个参数，表明只需输入特征尺寸图的大小 (d) 最后的输出是我们的分类数 $num\_classes$ (e) 辅助分类器的实现是在【Inception4a】和【Inception4b】传入的数据，对他们的输出结果做一个变化，保证我们可以充分利用算力 (f) 在全局平均池化后，尺寸变成了 1x1

复现结果如图14:

```
data = torch.ones(20,3,224,224)
```

```
net = GoogLeNet()
```

```
x,aux1,aux2 = net(data)
```

```
for i in [x,aux1,aux2]:
    print(i.shape)
```

```
torch.Size([20, 1000])
torch.Size([20, 1000])
torch.Size([20, 1000])
```

图 14 GoogLet 搭建结果

5 数据测试和比较

GoogLeNet 中的 Inception 块和总体架构的第一，第二部分均来自 VGG，我们将其和 VGG 进行比较。VGG 的代码复现将和 GoogLeNet 的代码一同上交。图片数据集来自 Kaggle 平台的猫狗图片识别[7]，训练过程曲线如下图15所示:

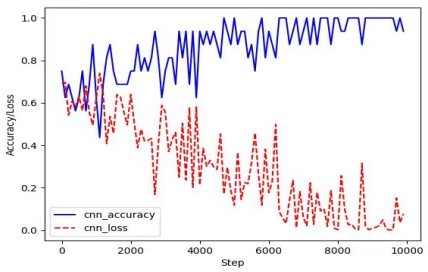

图 15 训练曲线

在进行一定微调的情况下设置 epoch 为 0.02，

对他们进行了分类判断。图片测试为每次测试 20 张照片，通道为 3，宽为 224，高为 224。我们得到，在数据精度方面，二者表现相当，为 93.41% 和 93.33% 但是在运行内存方面，在输入数据的内存大小为 12.04 MB 条件下，VGG 总参数量为 138,357,544，前向和反向传播时的复杂度为 309.68 GFLOPs，每次前向和反向传播的内存占用大小为 2169.07 MB，模型参数的内存占用为 553.43 MB16

```
----------------------------------------
Total params: 13,385,816
Trainable params: 13,385,816
Non-trainable params: 0
Total mult-adds (G): 31.82
========================================
Input size (MB): 12.04
Forward/backward pass size (MB): 1034.49
Params size (MB): 53.54
Estimated Total Size (MB): 1100.08
```

图 16 VGG 复杂度

GoogLeNet 的效果更加良好。输入数据的内存占用为 12.04 MB 的情况下，模型的总参数量为 13,385,816，每次前向和反向传播的运算复杂为 31.82 GFLOPs，每次前向和反向传播时所需的内存为 1034.49 MB，模型的参数占用内存为 53.54 MB，估算的模型总大小为 1100.08 MB。虽然用了更多的参数和参数空间，但是把速度提高了约 27 倍17

```
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
Total mult-adds (G): 309.68
========================================
Input size (MB): 12.04
Forward/backward pass size (MB): 2169.07
Params size (MB): 553.43
Estimated Total Size (MB): 2734.55
```

图 17 GoogLeNet 复杂度

6 不 足

GoogLeNet（Inception v1）尽管在图像识别领域取得了显著成绩，但也存在一些不足之处。首先，GoogLeNet 的结构较为复杂，尤其是其多个 Inception 模块的设计，使得模型的可解释性较差，且优化过程较为困难。其次，由于模型采用了较深的网络结构，训练过程中可能出现梯度消失或梯度爆炸的问题，尽管通过使用辅助分类器有所缓解。此外，GoogLeNet 的参数量相对较大，虽然相对于其他网络如 VGG 来说有所减少，但仍可能导致在资源有限的设备上训练和推理效率不高。最后，GoogLeNet 在处理较小的图像数据集时，可能无法充分发挥其优势，特别是在复杂度较低的任务中，训练过程中可能出现过拟合问题[8] [9] [10]

7 优 化

7.1 GoogLeNet V3 (Inception v3) 的改进

GoogLeNet V3 在原始的 GoogLeNet 基础上进行了多个优化，主要包括以下几个方面：

- **优化的 Inception 模块：**
  - *Factorization into smaller convolutions:* 在 V3 中，部分大卷积核被分解为更小的卷积核，尤其是 5x5 被分解为两个 3x3 的卷积，减少了计算量并提高了效率。[11]
  - *Asymmetric convolutions:* 采用不对称的卷积结构，比如将 7x7 卷积分解为 1x7 和 7x1 的卷积，从而进一步减少计算量。
  - *Depthwise separable convolutions:* 引入了深度可分卷积（depthwise separable convolutions），这种卷积方式将标准卷积操作分解为两步：首先对每个输入通道进行逐通道卷积，然后再进行逐点卷积，显著减少了参数量和计算复杂度。[12]
- **Auxiliary Classifier 的优化：** V3 对原始 GoogLeNet 中的辅助分类器（auxiliary classifier）进行了改进，通过将辅助分类器的输出与主网络的输出融合，提升了训练的稳定性。
- **Batch Normalization：** V3 引入了批归一化（Batch Normalization）来缓解梯度消失和加速训练。批归一化有助于加快收敛速度，并提高模型的泛化能力。
- **RMSProp Optimizer：** 在 V3 中，优化器从传统的 SGD 改进为 RMSProp，这种优化器对学习率进行动态调整，帮助解决了训练过程中学习率选择的问题，尤其适用于较深的网络。[13]
- **迁移学习和数据增强：** V3 强调了数据增强的使用，并且可以通过迁移学习技术，直接将预

训练的模型应用到不同的任务中，从而提升了模型的性能。

7.2 GoogLeNet V4 (Inception v4) 的改进

GoogLeNet V4 对 V3 做出了更进一步的改进，旨在进一步提高网络的性能和训练效率，主要改进包括：

- **更深的网络结构：** V4 增加了网络的深度，通过更多的 Inception 模块和更复杂的卷积层设计，进一步提高了网络的表示能力。V4 的网络结构变得更加复杂，但也使得模型能够捕捉到更细致的特征。
- **Inception-ResNet 模块：** 在 V4 中，提出了 *Inception-ResNet* 模块，结合了 Inception 网络和残差网络（ResNet）的优点。残差连接能够帮助网络更好地训练深度结构，缓解梯度消失问题，从而提升了模型的训练效率和性能。通过将残差连接引入 Inception 模块，V4 提高了模型的表现力，同时保持了计算效率。[14]
- **进一步的卷积优化：** 在卷积设计上，V4 进一步优化了各个层的卷积核选择，并将卷积核的尺寸控制在更适合处理大规模数据的范围内。通过改进卷积结构，V4 更好地平衡了计算复杂度和性能。
- **精细化的正则化技术：** V4 在模型训练过程中采用了更为精细的正则化技术，包括 Dropout 和 L2 正则化等，这些方法帮助减少了过拟合，提高了模型的泛化能力。[15]
- **更强的迁移学习能力：** V4 强化了在各种视觉任务上的迁移学习能力。通过更强的特征提取能力，V4 能够更好地应用于图像分类、目标检测、语义分割等多个任务。
- **提高计算效率：** 尽管 V4 增加了网络的深度，但也进一步优化了计算过程，特别是在减少不必要的计算开销上，使得 V4 在性能和效率之间取得了良好的平衡。

8 结 论

GoogLeNet，作为一种创新的深度卷积神经网络架构，在计算机视觉领域取得了显著的成果。通过引入 **Inception 模块**，GoogLeNet 实现了高效的特征提取，并通过并行的多尺度卷积层有效地捕捉了不同尺度的图像特征，减少了计算复杂度。与传统的卷积神经网络相比，GoogLeNet 更加注重计算资源的优化和参数的高效利用，从而能够在有限的硬件资源下训练深层网络。

GoogLeNet 的设计理念不仅体现在其结构创新上，还包括对模型性能的综合优化，例如使用**辅助分类器**来防止过拟合、引入 **批归一化（Batch Normalization）**来加速训练等技术。这些设计理念和技术使得 GoogLeNet 在 2014 年的 *ImageNet* 图像分类竞赛中表现出色，并推动了深度学习在计算机视觉领域的广泛应用。

随着 **GoogLeNet V3** 和 **GoogLeNet V4** 的推出，网络结构得到了进一步优化。在 V3 中，网络引入了深度可分卷积和改进的 Inception 模块，使得模型更加高效。而在 V4 中，融合了 *Inception-ResNet* 模块，通过引入残差学习的方式，进一步提升了网络的训练效率和精度。

尽管 GoogLeNet 在图像分类、目标检测等任务中取得了诸多成就，但随着网络架构的不断演进，诸如 **EfficientNet**、**DenseNet** 等新型架构的出现，逐步推动了更加高效和灵活的网络设计。未来的研究将继续探索如何在保留高精度的同时，进一步提升模型的计算效率和泛化能力。总的来说，GoogLeNet 不仅是深度学习领域的重要里程碑，也为后续的神经网络架构创新提供了宝贵的经验和启示。

参考文献

[1] LECUN Y, BOTTOU L, BENGIO Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998 (11): 2278-2324.

[2] KRIZHEVSKY I, A.and Sutskever, HINTON G E. Imagenet classification with deep convolutional neural networks[J]. Advances in Neural Information Processing Systems (NeurIPS), 2012(25): 1097-1105.

[3] SIMONYAN K, ZISSERMAN A. Very deep convolutional networks for large-scale image recognition[J]. International Conference on Machine Learning (ICML), 2014: 1-10.

[4] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[J]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015: 1-9.

[5] LIN M, CHEN Q, YAN S. Network in network[J]. 2014 Interna-

tional Conference on Learning Representations (ICLR), 2013.

[6] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the inception architecture for computer vision[J]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016: 2818-2826.

[7] CUKIERSKI W. Dogs vs. cats redux: Kernels edition[EB/OL]. 2016. https://kaggle.com/competitions/dogs-vs-cats-redux-kernels-edition.

[8] SZEGEDY C, LIU W, JIA Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2015: 1-9.

[9] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Rethinking the inception architecture for computer vision[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.: s.n.], 2016: 2818-2826.

[10] ZHANG X, ZHOU S. Googlenet: Performance and challenges[J]. Journal of Computer Science and Technology, 2017, 32(5): 1034-1045.

[11] SZEGEDY C, VANHOUCKE V, IOFFE S, et al. Inception-v4, inception-resnet and the impact of residual connections on learning[J]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017: 4278-4287.

[12] TAN M, LE Q V. Efficientnet: Rethinking model scaling for convolutional neural networks[J]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019: 6105-6114.

[13] HUANG G, LIU Z, VAN DER MAATEN L, et al. Densely connected convolutional networks[J]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017: 4700-4708.

[14] CHOLLET F. Xception: Deep learning with depthwise separable convolutions[J]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017: 1800-1807.

[15] HE K, ZHANG X, REN S, et al. Identity mappings in deep residual networks[J]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016: 630-638.

A GoogLeNet 代码实现

```
import torch
from torch import nn
from torchinfo import summary

class BasicConv2d(nn.Module):

    def __init__(self, in_channels
        , out_channels, **kwargs):
        super().__init__()
    self.conv=nn.Sequential(nn.
        Conv2d(in_channels,
        out_channels,bias=False,**
        kwargs),nn.BatchNorm2d(
        out_channel),nn.ReLU(
        inplace=True))

    def forward(self, x):
        x = self.conv(x)
        return x
    class Inception(nn.Module):

    def __init__(self,in_channels,
        ch1x1, ch3x3red, ch3x3,
        ch5x5red, ch5x5, pool_proj,
                ):
        super().__init__()
        self.branch1 = BasicConv2d
            (in_channels, ch1x1,
            kernel_size=1)

        self.branch2 = nn.
            Sequential(
             BasicConv2d(
                in_channels,
                ch3x3red,
                kernel_size=1),
             BasicConv2d(ch3x3red,
                ch3x3, kernel_size
                =3, padding=1)
        )
```

```
        self.branch3 = nn.
            Sequential(
             BasicConv2d(
                in_channels,
                ch5x5red,
                kernel_size=1),
             # Here, kernel_size=3
                instead of
                kernel_size=5 is a
                known bug.
             # Please see https://
                github.com/pytorch/
                vision/issues/906
                for details.
             BasicConv2d(ch5x5red,
                ch5x5, kernel_size
                =5, padding=2)
        )

        self.branch4 = nn.
            Sequential(
             nn.MaxPool2d(
                kernel_size=3,
                stride=1, padding
                =1),
             BasicConv2d(
                in_channels,
                pool_proj,
                kernel_size=1)
        )

    def forward(self, x):
        branch1 = self.branch1(x)
        branch2 = self.branch2(x)
        branch3 = self.branch3(x)
        branch4 = self.branch4(x)

        outputs = [branch1,
            branch2, branch3,
            branch4]
        return torch.cat(outputs,
            1)
```

```
    class AuxClf(nn.Module):
    def __init__(self
                 ,in_channels
                 ,num_classes
                 ,**Kwargs):
        super().__init__()
        self.feature_=nn.
            Sequential(nn.AvgPool2d
            (kernel_size=5,stride
            =3),BasicConv2d(
            in_channels,128,
            kernel_size=1))
        self.clf_=nn.Sequential(nn
            .Linear(4*4*128,1024),
            nn.ReLU(inplace=True),
            nn.Dropout(0.7),nn.
            Linear(1024,num_classes
            ))
    def forward(self,x):
            x=self.feature_(x)
            x=x.view(-1,4*4*128)
            x=self.clf_(x)
            return x
    class GoogLeNet(nn.Module):
     def __init__(self,num_classes
        =1000):
        super().__init__()
        self.conv1 = BasicConv2d
            (3, 64, kernel_size=7,
            stride=2, padding=3)
        self.maxpool1 = nn.
            MaxPool2d(kernel_size
            =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(kernel_size
            =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(kernel_size
            =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)



        self.avgpool = nn.
            AdaptiveAvgPool2d((1,
            1))
        self.dropout = nn.Dropout
            (0.4)
        self.fc = nn.Linear(1024,
            num_classes)

        self.aux1 = AuxClf(512,
            num_classes)
        self.aux2 = AuxClf(528,
            num_classes)
    def forward(self,x):
        # type: (Tensor) -> Tuple[
            Tensor, Optional[Tensor
            ], Optional[Tensor]]
        # N x 3 x 224 x 224
        # x = self.conv1(x)
        # N x 64 x 112 x 112
        x = self.maxpool1(self.
            conv1(x))

        # N x 64 x 56 x 56
        # x = self.conv2(x)
        # N x 64 x 56 x 56
        x = self.maxpool2(self.
            conv3(self.conv2(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(self.
            inception3b(x))


        # N x 480 x 14 x 14
        x = self.inception4a(x)
        # N x 512 x 14 x 14

        aux_1 = 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


        aux_2 = self.aux2(x)


        # N x 832 x 14 x 14
        x = self.maxpool4(self.
            inception4e(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)

        return x, aux_1, aux_2
```

B VGG 代码实现

```
import torch
from torch import nn
from torch.nn import functional as
    F
from torchinfo import summary
class VGG16(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1=nn.Conv2d
            (3,64,3,padding=1)
        self.conv2=nn.Conv2d
            (64,64,3,padding=1)
        self.pool1=nn.MaxPool2d(2)

        self.conv3=nn.Conv2d
            (64,128,3,padding=1)
        self.conv4=nn.Conv2d
            (128,128,3,padding=1)
        self.pool2=nn.MaxPool2d(2)

        self.conv5=nn.Conv2d
            (128,256,3,padding=1)
        self.conv6=nn.Conv2d
            (256,256,3,padding=1)
        self.conv7=nn.Conv2d
            (256,256,3,padding=1)
        self.pool3=nn.MaxPool2d(2)

        self.conv8=nn.Conv2d
            (256,512,3,padding=1)
        self.conv9=nn.Conv2d
            (512,512,3,padding=1)
        self.conv10=nn.Conv2d
            (512,512,3,padding=1)
        self.pool4=nn.MaxPool2d(2)

        self.conv11=nn.Conv2d
            (512,512,3,padding=1)
        self.conv12=nn.Conv2d
            (512,512,3,padding=1)
        self.conv13=nn.Conv2d
            (512,512,3,padding=1)
        self.pool5=nn.MaxPool2d(2)

        self.lr1=nn.Linear
            (7*7*512,4096)
        self.lr2=nn.Linear
            (4096,4096)
        self.lr3=nn.Linear
            (4096,1000)
```

```
    def forward(self,x):
        x=F.relu(self.conv1(x))
        x=self.pool1(F.relu(self.
            conv2(x)))
        x=F.relu(self.conv3(x))
        x=self.pool2(F.relu(self.
            conv4(x)))
        x=F.relu(self.conv5(x))
        x=F.relu(self.conv6(x))
        x=self.pool3(F.relu(self.
            conv7(x)))
        x=F.relu(self.conv8(x))
        x=F.relu(self.conv9(x))
        x=self.pool4(F.relu(self.
            conv10(x)))
        x=F.relu(self.conv11(x))
        x=F.relu(self.conv12(x))
        x=self.pool5(F.relu(self.
            conv13(x)))
        x=x.view(-1,7*7*512)
        x=F.relu(self.lr1(F.
            dropout(x,p=0.5)))
        x=F.relu(self.lr2(F.
            dropout(x,p=0.5)))
        output = F.softmax(self.
            lr3(x),dim=1)
```