第5章

轻量级DCNN模型

CHAPTER 5

本章学习目标

- MobileNet 系列
- ShuffleNet 系列
- 轻量级 DCNN 模型对比

视频讲解

第 4 章介绍了常用的 DCNN 模型,这些模型在推理过程中都有较大的计算量,很难应用在移动端或嵌入式平台等资源受限的硬件设备上。越来越多的 DCNN 模型设计开始关注于资源受限场景中的效率问题,提出了多种轻量级的 DCNN 模型。其主要的设计目标是在保证一定识别精度的前提下,尽可能地减少网络规模(参数量、计算量)。本章将详细介绍轻量级 DCNN 模型的设计思路,并介绍两种常用的轻量级 DCNN 模型。本章内容的框图如图 5-1 所示。

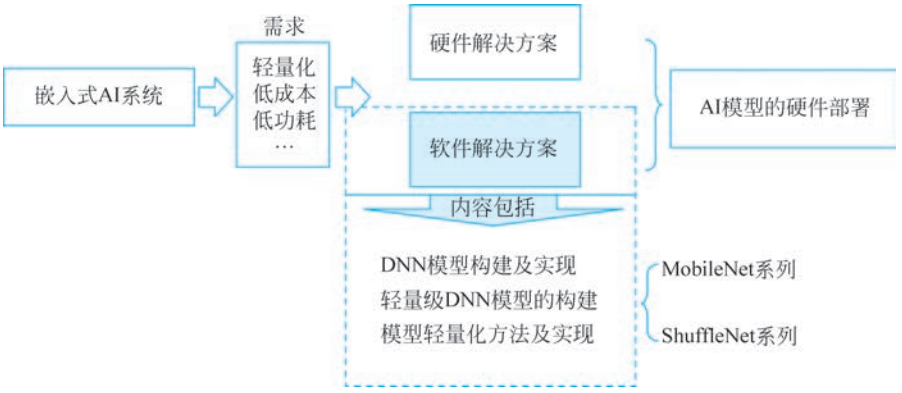


图 5-1　本章内容框图

目前比较成熟的轻量级网络包括 Google 公司的 MobileNet 系列,旷视公司的 ShuffleNet 系列等,这些轻量级的网络模型更适合 CPU 或是移动端硬件。如图 5-2 所示,2016 年直至现在,业内提出了 SqueezeNet、ShuffleNet、NSANet、MnasNet 以及 MobileNet(V1、V2 和 V3)等轻量级网络模型。这些模型使在移动终端、嵌入式设备中运行 DCNN 模型成为可能。本章将介绍两种常用的轻量级 DCNN 模型。

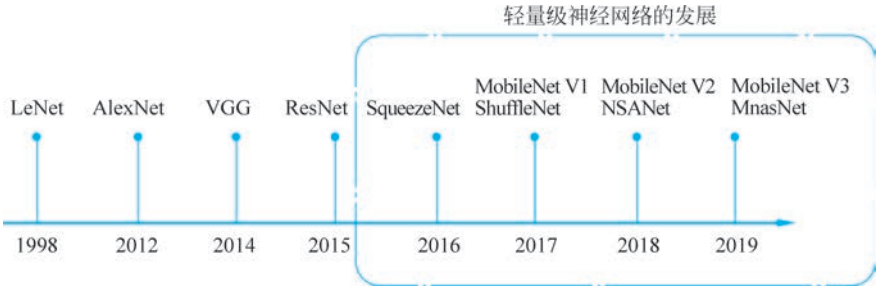


图 5-2　轻量级网络模型的发展示意图

5.1　MobileNet 系列

5.1.1　MobileNet V1

表 5-1 梳理了 MobileNet V1 的主要创新点,本节将分别介绍这些创新点。

表 5-1　MobileNet V1 主要创新点梳理

特　　性	作　　用
深度可分离卷积	替代常规卷积操作，减少参数量及计算量
宽度因子	调整神经网络中间产生特征大小的超参数
分辨率因子	通过调整输入数据的尺寸，调整网络的计算量
批规范化	加速训练收敛速度，提升准确度

1）深度可分离卷积(Depthwise Separable Convolution)

卷积神经网络中，最费时间的就是其中的卷积运算，MobileNet 系列提出通过 Depthwise 卷积与 Pointwise 卷积替代常规的卷积，从而降低参数量及运算成本。

常规的卷积如图 5-3 所示，对于一张 5×5 像素 3 通道彩色输入图片(尺寸 5×5×3)。经过 3×3 卷积核的卷积层(假设输出通道数为 4，则卷积核的尺度为 3×3×3×4)，最终输出 4 个特征图。深度可分离卷积是将一个完整的卷积运算分解为两步进行，即 Depthwise Convolution(DW)与 Pointwise Convolution(PW)。不同于常规卷积操作，Depthwise Convolution 的一个卷积核负责图像的一个通道，一个通道只被一个卷积核卷积。而常规卷积中每个卷积核是同时操作输入图片的每个通道。同样是对于一张 5×5 像素 3 通道彩色输入图片(尺寸为 5×5×3)。如图 5-4 所示，Depthwise Convolution 完成后的特征图数量与输入层的通道数相同，无法扩展特征图。而且这种运算对输入层的每个通道独立进行卷积运算，没有有效地利用不同通道在相同空间位置上的特征信息。因此需要 Pointwise Convolution 来将这些特征图进行组合生成新的特征图，如图 5-5 所示。Pointwise Convolution 的运算与常规卷积运算非常相似，它的卷积核的尺寸为 1×1×M，M 为预生成的通道数。所以这里的卷积运算会将上一步的输出在深度方向上进行加权组合，生成新的特征图。有几个 Pointwise Convolution 卷积核就有几个输出特征图，深度可分离卷积(Depthwise Separable)操作的示意图如图 5-6 所示。

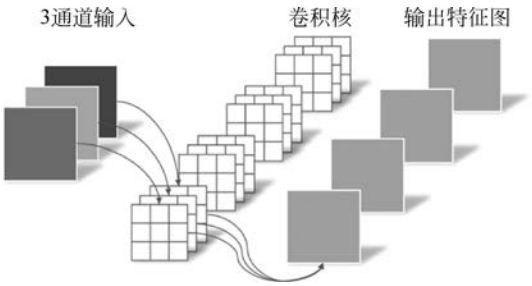


图 5-3　常规的卷积操作示意图

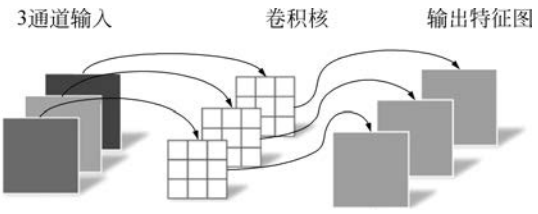


图 5-4　Depthwise Convolution 操作示意图

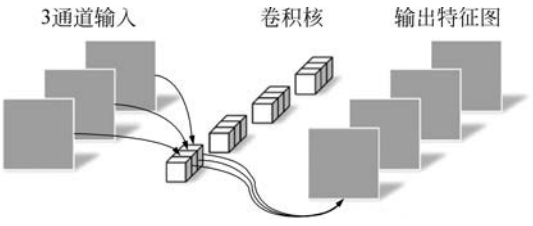


图 5-5　Pointwise Convolution 操作示意图

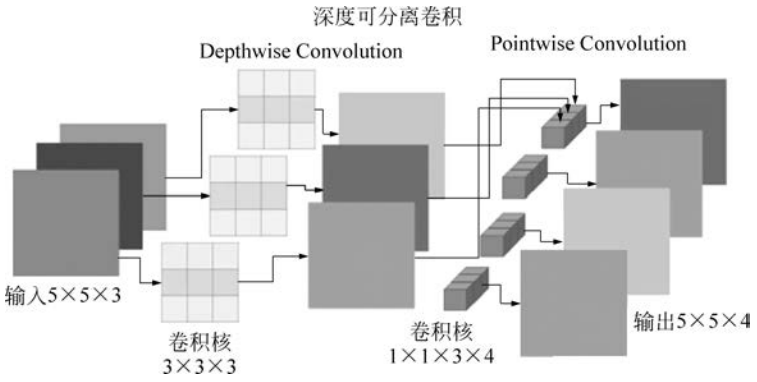


图 5-6 深度可分离卷积操作的示意图

2017 年 Google 提出了 MobileNet 模型,上述的深度可分离卷积为最大的创新点。以下比较不同卷积方式的参数量(params)和计算量(MultiAdd),假设输入输出大小是一样的。

输入尺寸:$h_{in} \times w_{in} \times c_{in}$。

输出尺寸:$h_{out} \times w_{out} \times c_{out}$。

卷积核大小:k。

标准卷积

$$\text{params} = k^2 c_{in} c_{out}$$

$$\text{MultiAdd} = k^2 c_{in} c_{out} \times h_{out} w_{out}$$

Depthwise 卷积

$$\text{params} = k^2 c_{in}$$

$$\text{MultiAdd} = k^2 c_{in} \times h_{out} w_{out}$$

深度可分离卷积

$$\text{params} = k^2 c_{in} + c_{in} c_{out}$$

$$\text{MultiAdd} = k^2 c_{in} \times h_{out} w_{out} + c_{in} c_{out} \times h_{out} w_{out}$$

相对于标准卷积,深度可分离卷积理论上的加速比可达:

$$\frac{1}{c_{out}} + \frac{1}{k}$$

其中,c_{in} 为输入通道数,c_{out} 为输出通道数,h_{out} 和 w_{out} 为输出特征图的尺寸。

2) 宽度因子

MobileNet 本身的网络结构已经比较小而且执行延迟较低,但为了适配一些特定的场景,MobileNet 提供了称为宽度因子(Width Multiplier)的超参数。宽度因子可以调整神经网络中间产生特征大小。由于调整的是特征图通道数量,从而可以调整运算量。宽度因子简单来说就是新网络中每一个模块要使用的卷积核数量相较于标准的 MobileNet 比例。对于深度卷积结合 1×1 方式的卷积核,计算量为:

$$\text{MultiAdd} = k^2 \alpha c_{in} \times h_{out} w_{out} + \alpha c_{in} \times \alpha c_{out} \times h_{out} w_{out}$$

其中,α 为宽度因子,α 常用的配置为 1、0.75、0.5、0.25;当 α 等于 1 时就是标准的 MobileNet。通过参数 α 可以非常有效地将计算量和参数数量约减到接近 α 的平方。表 5-2

为 MobileNet V1 使用不同宽度因子进行网络参数的调整时，在 ImageNet 上的准确率、计算量、参数数量之间的关系（每一个项中最前面的数字表示 α 的取值）。可以看到当输入分辨率固定为 224×224 时，随着宽度因子的减少，模型的计算量和参数越来越小。宽度因子为 0.25 时，MobileNet 的准确率比标准版 MobileNet 低 20%左右，但计算量和参数量几乎只有标准版 MobileNet 计算量和参数量的 10%。对于计算资源和存储资源都十分紧张的移动端平台，通过宽度因子调节网络的参数量的方法非常实用，可以按需调整宽度因子，达到准确率与性能的平衡。宽度因子一般写在 MobileNet 的前面，例如 α MobileNet，α 为宽度因子。

表 5-2　MobileNet V1 使用不同宽度因子的准确率、计算量、参数数量

宽 度 因 子	准确率（Top1）	计算量（Million）	参数量（Million）
1.0MobileNet-224	70.6%	569	4.3
0.75MobileNet-224	68.4%	325	2.6
0.5MobileNet-224	63.7%	149	1.3
0.25MobileNet-224	50.6%	41	0.5

3）分辨率因子

MobileNet 提供了另一个超参数——分辨率因子（Resolution Multiplier），供用户自定义网络结构。分辨率因子 β 的取值范围在[0,1]之间，是作用于每一个模块输入尺寸的约减因子，简单来说就是将输入数据以及由此在每一个模块产生的特征图都变小了，结合宽度因子 α，深度卷积结合 1×1 方式的卷积核计算量为：

$$\text{MultiAdd}=k^2\alpha c_{\text{in}}\times\beta h_{\text{out}}\times\beta w_{\text{out}}+\alpha c_{\text{in}}\times\alpha c_{\text{out}}\times\beta h_{\text{out}}\times\beta w_{\text{out}}$$

下表为 MobileNet V1 使用不同的 β 系数作用于标准 MobileNet 时，在 ImageNet 上对精度和计算量的影响（α 固定为 1.0）。可以表达为 MobileNet—β * S，S 为原尺寸 224。

表 5-3　MobileNet V1 使用不同分辨率因子的准确率、计算量、参数数量

分辨率因子	准确率（Top1）	计算量（Million）	参数量（Million）
1.0MobileNet-224	70.6%	569	4.3
1.0MobileNet-192	69.1%	418	4.2
1.0MobileNet-160	67.2%	290	4.2
1.0MobileNet-128	64.6%	186	4.2

图像分辨率的变化不会引起参数量的变化，只会改变模型的计算量。如表 5-3 所示，224 分辨率模型测试 ImageNet 数据集准确率为 70.6%，192 分辨率的模型准确率为 69.1%，但是计算量减少了 151M，在移动平台计算资源紧张的情况下，同样可以通过分辨率因子 β 调节网络输入特征图的分辨率，做模型精度与计算量的取舍。MobileNet V1 的网络结构如表 5-4 所示。

表 5-4　MobileNet V1 的网络结构

种类/步长	滤波器尺寸	输 入 尺 寸
Conv/s2	3×3×3×32	224×224×3
Conv dw/s1	3×3×32	112×112×32
Conv/s1	1×1×32×64	112×112×32

续表

种类/步长		滤波器尺寸	输入尺寸
Conv dw/s2		3×3×64	112×112×64
Conv/s1		1×1×64×128	56×56×64
Conv dw/s1		3×3×128	56×56×128
Conv/s1		1×1×128×128	56×56×128
Conv dw/s2		3×3×128	56×56×128
Conv/s1		1×1×128×256	28×28×128
Conv dw/s1		3×3×256	28×28×256
Conv/s1		1×1×256×256	28×28×256
Conv dw/s2		3×3×256	28×28×256
Conv/s1		1×1×256×512	14×14×256
5×	Conv dw/s1	3×3×512	14×14×512
	Conv/s1	1×1×512×512	14×14×512
Conv dw/s2		3×3×512	14×14×512
Conv/s1		1×1×512×1024	7×7×512
Conv dw/s2		3×3×1024	7×7×1024
Conv/s1		1×1×1024×1024	7×7×1024
Avg Pool/s1		Pool 7×7	7×7×1024
FC/s1		1024×1000	1×1×1024
Softmax/s1		Classfier	1×1×1000

5.1.2 MobileNet V2

2018 年,Google 推出了 MobileNet V2 版本,引入了 Inverted Residuals 和 Linear Bottlenecks 两个模块。MobileNet V2 主要借鉴了 ResNet 的残差网络的思想,在残差单元上进行了修改,成为 Inverted Residuals 模块。Linear Bottleneck 将最后一层的激活函数从 ReLU 替换成线性激活函数,而其他层的激活函数依然是 ReLU。

1) Inverted Residuals

ResNet 中提出的残差结构解决训练中随着网络深度的增加而出现的梯度消失问题,使反向传播过程中网络的浅层部分也能有效地得到梯度更新,从而提高特征的表达能力。在 ResNet 的残差结构的启发下,MobileNet 的残差结构如图 5-7 所示。Inverted Residual,顾名思义,颠倒的残差,其与 ResNet 中经典的残差块的结构相反。

2) Linear Bottleneck

Bottleneck 结构首次被提出是在 ResNet 网络中。该结构第一层使用 PW Convolution,第二层使用 3×3 大小卷积核进行 DW Convolution,第三层使用逐点卷积。MobileNet 中的 Bottleneck 结构最后一层 PW Convolution 使用的激活函数是 Linear,所以称其为线性瓶颈结构(Linear Bottleneck)。图 5-8 展示了 MobileNet V1 和 MobileNet V2 在深度可分离卷积模块结构上的差异。MobileNet V2 的网络结构如表 5-5 所示,创新点如表 5-6 所示。

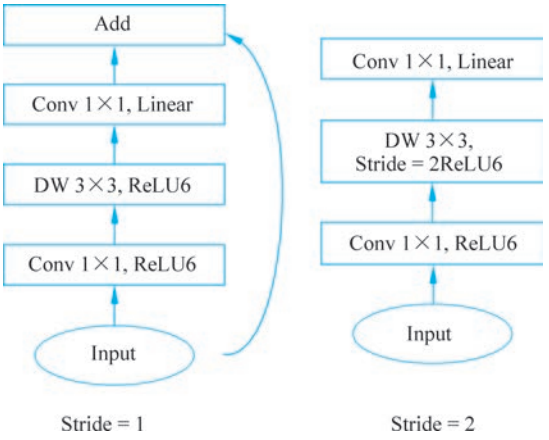


图 5-7　MobileNet 残差结构示意图

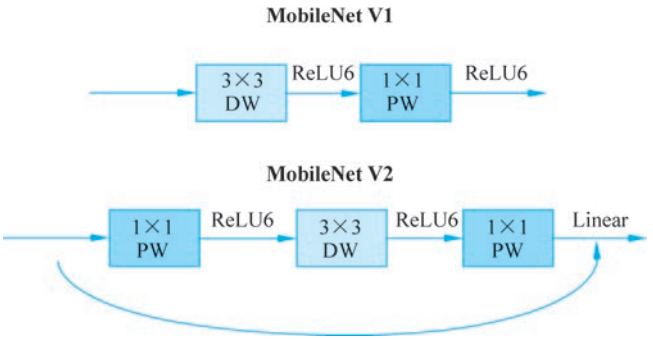


图 5-8　MobileNet V1 和 MobileNet V2 在深度可分离卷积模块结构上的差异

表 5-5　MobileNet V2 的网络结构

输　入	种　类	膨胀因子(t)	输出特征图数量(c)	重复次数(n)	步长(s)
224×224×3	Conv2d	—	32	1	2
112×112×32	Bottleneck	1	16	1	1
112×112×16	Bottleneck	6	24	2	2
56×56×24	Bottleneck	6	32	3	2
28×28×32	Bottleneck	6	64	4	2
14×14×64	Bottleneck	6	96	3	1
14×14×96	Bottleneck	6	160	3	2
7×7×160	Bottleneck	6	320	1	1
7×7×320	Conv2d 1×1	—	1280	1	1
7×7×1280	Avg Pool 7×7	—	—	1	—
1×1×1280	Conv2d 1×1	—	k	—	

表 5-6　MobileNet V2 主要创新点

特　性	作　用
线性瓶颈结构 (Linear Bottleneck)	减少参数量及计算量
Inverted Residuals	解决梯度消失问题，提高特征的表达能力
平均池化与逐点卷积	代替全连接层，减少参数量

5.1.3 MobileNet V3

2019 年 Google 公司提出了 MobileNet V3。首先,MobileNet V3 中利用了 5×5 大小的深度卷积代替部分 3×3 的深度卷积,在神经结构搜索(NAS)技术在搜索 MobileNet V3 网络结构的过程中,发现使用 5×5 大小的卷积核比使用 3×3 大小的卷积核效果更好,且准确率更高;其次,引入 Squeeze-and-excitation(SE)模块和 H-Swish(HS)激活函数以提高模型精度;最后,结尾两层 Pointwise Convolution 不使用批规范化(Batch Norm)(MobileNet V3 结构图中使用 NBN 标识部分)。MobileNet V3 分为 Large 和 Small 两个版本,Large 版本适用于计算和存储性能较高的平台,而 Small 版本适用于硬件性能较低的平台。

在嵌入式设备中计算 Sigmoid 函数是会耗费相当多的计算资源的,因此提出了 H-Swish(Hard Version of Swish)作为激活函数。而且随着网络的加深,非线性激活函数的成本也会随之减少。所以,只有在较深的层使用 H-Swish 才能获得更大的优势,激活函数 H-Swish 的表达式如下:

$$\text{H\_Swish}[x]=x\,\frac{\text{ReLU6}(x+3)}{6}$$

可以看出,其非线性结构在保持精度的情况下具有优势,ReLU6[①] 在量化时会避免数值精度的损失,而且具有运行快的特点。

MobileNet V3 还引入 SE 模块,SE 模块是一种轻量级的通道注意力模块。如图 5-9 所示,一个 SE 模块主要包含压缩(Squeeze)和激励(Excitation)两部分,W、H 表示特征图宽与高,C 表示通道数,则输入特征图大小为 $W\times H\times C$。

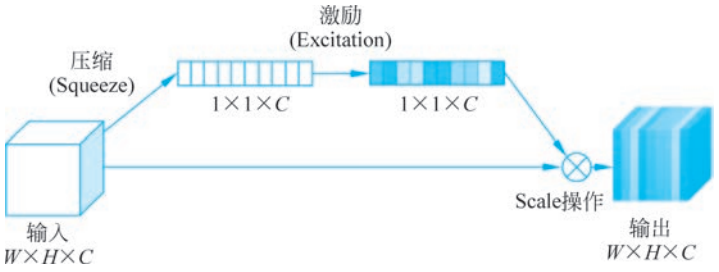


图 5-9 Squeeze-and-excitation 模块示意图

表 5-7 和表 5-8 介绍了 MobileNet V3 Large 与 MobileNet V3 Small 的网络结构。同时,我们梳理了 MobileNet V3 主要创新点,如表 5-9 所示。

表 5-7 MobileNet V3 Large 结构示意图

输　入	种　类	输出通道数量(c)	SE	激活函数	步长(s)
224×224×3	Conv2d	16	——	HS	2
112×112×16	Bottleneck 3×3	16	——	ReLU	1
112×112×16	Bottleneck 3×3	24	——	ReLU	2

① ReLU 6 是普通的 ReLU 函数,但限制其最大输出为 6。

续表

输　　入	种　　类	输出通道数量(c)	SE	激活函数	步长(s)
56×56×24	Bottleneck 3×3	24	——	ReLU	1
56×56×24	Bottleneck 5×5	40	√	ReLU	2
28×28×40	Bottleneck 5×5	40	√	ReLU	1
28×28×40	Bottleneck 5×5	40	√	ReLU	1
28×28×40	Bottleneck 3×3	80	——	HS	2
14×14×80	Bottleneck 3×3	80	——	HS	1
14×14×80	Bottleneck 3×3	80	——	HS	1
14×14×80	Bottleneck 3×3	80	——	HS	1
14×14×80	Bottleneck 3×3	112	√	HS	1
14×14×112	Bottleneck 3×3	112	√	HS	1
14×14×112	Bottleneck 5×5	160	√	HS	1
14×14×112	Bottleneck 5×5	160	√	HS	2
7×7×160	Bottleneck 5×5	160	√	HS	1
7×7×160	Conv2d 1×1	960	——	HS	1
7×7×960	Pool 7×7	—	—	HS	—
1×1×960	Conv2d 1×1 NBN	1280	—	HS	1
1×1×1280	Conv2d 1×1 NBN	—	k		—

表 5-8　MobileNet V3 Small 的网络结构

输　　入	种　　类	输出通道数量(c)	SE	激活函数	步长(s)
224×224×3	Conv2d 3×3	16	——	HS	2
112×112×24	Bottleneck 3×3	16	√	ReLU	2
56×56×24	Bottleneck 3×3	24	——	ReLU	2
28×28×24	Bottleneck 3×3	24	——	ReLU	1
28×28×40	Bottleneck 5×5	40	√	HS	1

续表

输　入	种　类	输出通道数量(c)	SE	激活函数	步长(s)
14×14×40	Bottleneck 5×5	40	√	HS	1
14×14×40	Bottleneck 5×5	48	√	HS	1
14×14×40	Bottleneck 5×5	48	√	HS	1
14×14×48	Bottleneck 5×5	96	√	HS	1
14×14×96	Bottleneck 5×5	96	√	HS	2
7×7×96	Bottleneck 5×5	96	√	HS	1
7×7×96	Bottleneck 5×5	576	√	HS	1
7×7×96	Conv2d 1×1	—	√	HS	1
7×7×576	Pool 7×7	1280	—	HS	7
1×1×576	Conv2d 1×1	—	—	HS	1
1×1×1280	Conv2d 1×1	k	—	HS	—

表 5-9　MobileNet V3 主要创新点

特　性	作　用
5×5 卷积核	代替 3×3 卷积
SE 模块	使得有效权重大,提高准确率
H-Swish 激活函数	代替部分 ReLU 激活函数

表 5-10 给出了 MobileNet 系列与常用的 DCNN 模型在准确度、计算量、参数量的对比结果,可以参考实验结果,根据实际需求选择更适合的网络模型。

表 5-10　MobileNet 系列与常用 DCNN 模型在准确度、计算量、参数量的对比结果

模　型	准确率(Top1)	计算量(Million)	参数量(Million)
VGG-16	71.5%	15300	138
GoogleNet	69.8%	1550	6.8
MobileNet V1	70.6%	569	4.2
MobileNet V2	72.0%	326	3.4
MobileNet V3-Large	75.2%	219	5.4
MobileNet V3-Small	67.4%	66	2.9

5.2　ShuffleNet 系列

5.2.1　ShuffleNet V1

ShuffleNet 是旷视科技公司提出的一种轻量化 CNN 模型,与 MobileNet 一样,主要是

希望将其应用于移动端或嵌入式设备端。ShuffleNet 的核心是采用了两种操作，Pointwise Group Convolution 和 Channel Shuffle，保证在保持精度的同时大幅地降低了模型的计算量。

Channel Shuffle 的示意图如图 5-10 所示。如图 5-10(a)所示，例如，输入的特征图的数量为 N，该卷积层的滤波器数量为 M，那么 M 个卷积核中的每一个滤波器都要和 N 个特征图做卷积，而后相加作为一个卷积的结果。如果引入分组操作的话，设小组数量为 g，那么 N 个输入特征图就被分成 g 个小组，而 M 个卷积核也被分成 g 个小组，然后在做卷积操作的时候，第一个小组中的 M/g 个滤波器中的和第一个小组的 N/g 个输入特征图做卷积得到结果，第二个小组到最后一个小组同理。很显然，这种操作可以大幅地减少计算量，因为每个卷积核不再是与全部特征图做卷积，而是和一个组的特征图做卷积。但是如果多个分组操作叠加在一起的话，会产生边界效应，也就是某个输出层仅来自输入层的一小部分，导致训练的特征非常局限。ShuffleNet 提出了通过 Channel Shuffle 来解决这个问题，如图 5-10(b)所示在 GConv2 之前，先对其输入的特征图做分配，将每个小组分成多个更小的组(Sub-group)，然后将不同小组的 Sub-group 作为 GConv2 的一个小组的输入，使得 GConv2 的每一个小组都能卷积输入到的所有小组的特征图，这和 5-10(c)的思想是一样的。

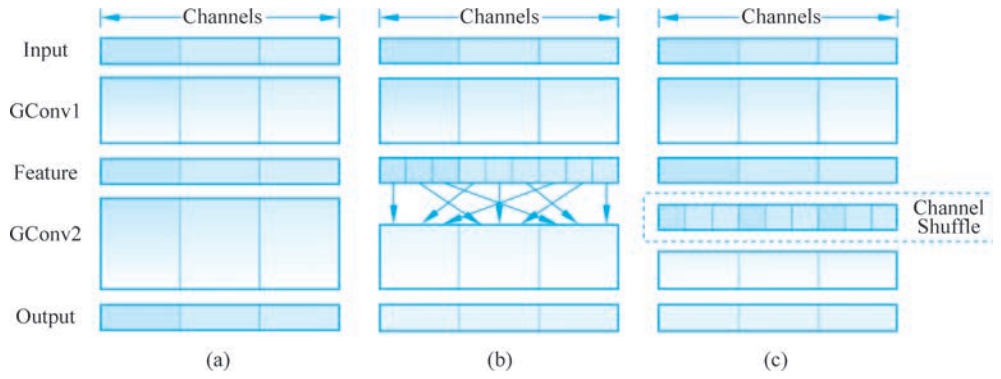
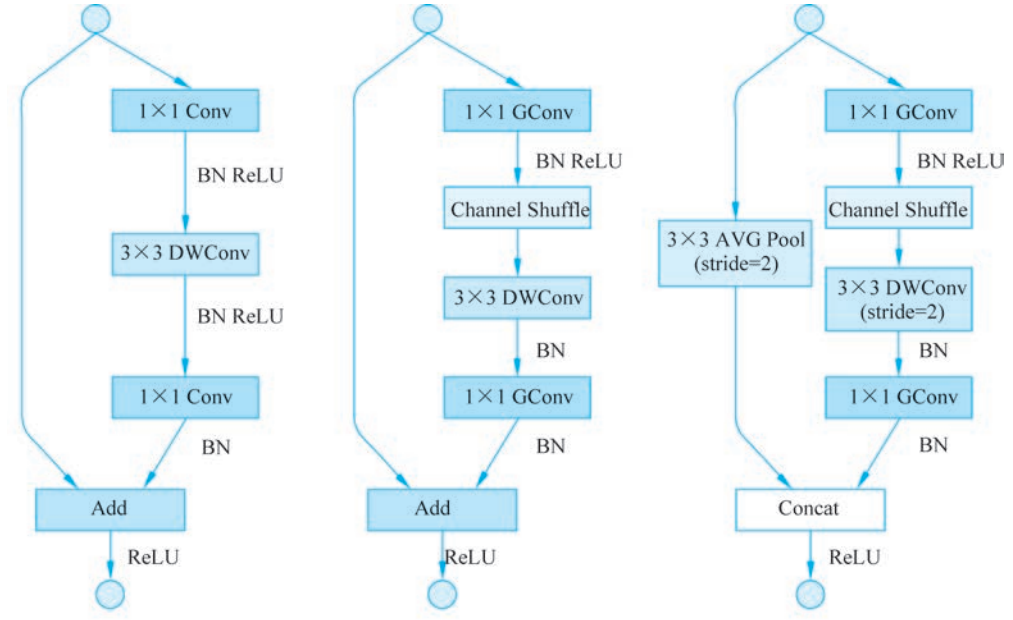


图 5-10　Channel Shuffle 与 Group Convolution 示意图

基于上面的设计理念，首先构造如图 5-11 所示的 ShuffleNet 的基本单元。ShuffleNet 的基本单元是在残差单元的基础上改进而成的。5-11(a)为 MobileNet 残差单元的结构。如图 5-11(b)所示，ShuffleNet 进行如下改进。首先将 1×1 的 Depth Convolution 替换成 1×1 的 Group Convolution；之后在第一个 1×1 卷积之后增加了一个 Channel Shuffle 操作。对于残差单元，如果步长(Stride)为 1 时，由于输入与输出尺度一致便可以直接相加，而当步长为 2 时，通道数增加，同时特征图的尺度减小，导致输入与输出不匹配。ShuffleNet 提出的策略如图 5-11(c)所示，对原输入采用步长为 2 的 3×3 平均池化，得到和输出一样大小的特征图，然后将得到的特征图与输出进行连接，而不是相加，其目的主要是降低计算量与参数大小。

基于以上改进的 ShuffleNet 基本单元，设计的 ShuffleNet V1 模型结构如表 5-11 所示。



(a) MobileNet残差单元的结构　(b) ShuffleNet的基本单元　(c) ShuffleNet提出的策略

图 5-11　ShuffleNet 的基本单元

表 5-11　ShuffleNet V1 模型结构

层	输出尺寸	K 尺寸	步长	重复次数	输出通道(g groups)				
					g=1	g=2	g=3	g=4	g=8
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	25
MaxPool	56×56	3×3	2						
Stage2	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity					143M	140M	137M	133M	137M

5.2.2　ShuffleNet V2

旷视科技又提出了 ShuffleNet V2,主要的想法是在设计网络结构时,需要同时考虑计算量 FLOPs、内存的访问代价(Memory Access Cost,MAC)、并行化对应的时间,以及不同的部署环境 ARM 或者 GPU。在考虑上述因素的前提下,提出网络设计方面的原则如下。

(G1) 卷积输入的通道数与输出的通道数应该尽量相同,这时 MAC 最小;

(G2) 过多的 Group Convolution 会增加内存访问时间;

(G3) 网络分支数量多会降低并行度；

(G4) Element wise(逐个元素运算)的运算增加内存访问时间。在计算 FLOPs 时往往只考虑卷积中的乘法操作，Element wise(ReLU 激活、偏置、单位加等)往往被忽略，这些操作看似数量很少，它对模型速度的影响却很大。

从上面的原则可以看出，ShuffleNet V1 与 MobileNet 系列都存在可以改进的空间。ShuffleNet V2 在 ShuffleNet V1 的基础上做的改进方案如下。

(1) 在输入层提出了 Channel Split 模型，代替原来 Group Convolution 的作用；

(2) 在输出结果时采用连接(Concat)操作，替代以前的加(Add)操作；

(3) 移除部分的 Channel Shuffle 操作。

图 5-12 对比了 ShuffleNet V1 和 V2 的基本单元，图 5-12(a)和图 5-12(b)都是 ShuffleNet V1 里面的基本单元，图 5-12(c)和图 5-12(d)则为 ShuffleNet V2 的基本单元。

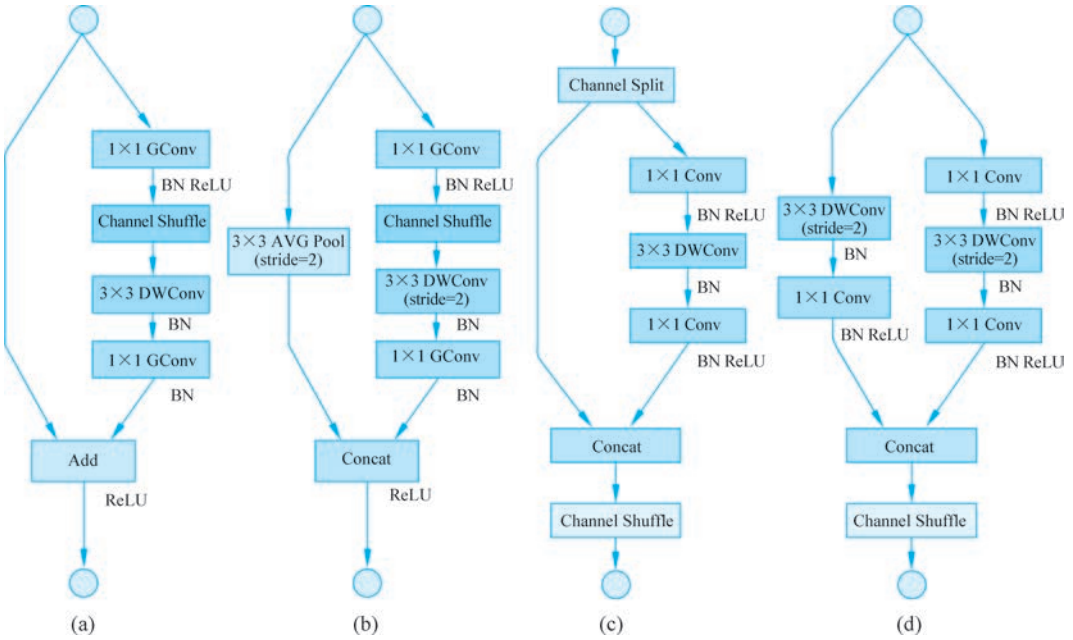


图 5-12　ShuffleNet V2 的基本单元

在图 5-12(c)中可以看到使用了 Channel Split 操作，这个操作将输入特征的通道 c 分成 c-c′和 c′，其目的是尽量控制分支数，以满足 G3 原则。

相比于 ShuffleNet V1，ShuffleNet V2 基本单元中的两个 1×1 卷积不再是分组卷积，满足 G2 原则，另外两个分支都采用连接操作进行合并，这样使得每个基本单元的输入通道数和输出通道数一样，这个操作可以满足 G1 原则。

ShuffleNet V2 的基本单元中已经没有了 ShuffleNet V1 中的 Add 和 ReLU 操作了，同时 Depthwise 卷积只在一个分支里面。Channel shuffle 以及 Channel split 这三个 Element wise 的相关操作已经合并成为一个单独的 Element wise 操作，这个操作可以满足 G4 原则。

基于以上改进的 ShuffleNet 基本单元，设计的 ShuffleNet V2 模型结构如表 5-12 所示。

表 5-12　ShuffleNet V2 模型结构

层	输出尺寸	K 尺寸	步长	重复次数	输出通道			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24
MaxPool	56×56	3×3	2					
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
Weights					1.4M	2.3M	3.5M	7.4M

5.3　轻量级 DCNN 模型对比

在 ShuffleNet V2 论文中对多种轻量级 DCNN 模型进行了对比实验,在 COCO 目标检测数据集中的检查实验结果如表 5-13 所示,介绍 FLOPs 为 500M 的情况。可以看出 ShuffleNet V2 在 GPU 平台上是最快的,精度也是最高的。

表 5-13　轻量级 DCNN 模型对比

模　　型	mmAP/(%)	GPU 平台速度(Images/s)
ShuffleNet V1	32.9	60
MobileNet V2	30.6	72
ShuffleNet V2	33.3	87

5.4　项目实战

构建 MobileNet V3 和 ShuffleNet V2 网络模型,并进行测试。

代码清单

5.4.1　MobileNet V3 模型构建

```
1. #导入库
2. import torch
3. import torch.nn as nn
4. import torch.nn.functional as F
```

```
__all__ = ['MobileNet V3', 'mobilenetv3']

def conv_bn(inp, oup, stride, conv_layer = nn.Conv2d, norm_layer = nn.BatchNorm2d, nlin_layer = nn.ReLU):
    return nn.Sequential(
        conv_layer(inp, oup, 3, stride, 1, bias = False),
        norm_layer(oup),
        nlin_layer(inplace = True)
    )


def conv_1x1_bn(inp, oup, conv_layer = nn.Conv2d, norm_layer = nn.BatchNorm2d, nlin_layer = nn.ReLU):
    return nn.Sequential(
        conv_layer(inp, oup, 1, 1, 0, bias = False),
        norm_layer(oup),
        nlin_layer(inplace = True)
    )

#定义 H-Swish(HS)激活函数
class Hswish(nn.Module):
    def __init__(self, inplace = True):
        super(Hswish, self).__init__()
        self.inplace = inplace

    def forward(self, x):
        return x * F.relu6(x + 3., inplace = self.inplace) / 6.

class Hsigmoid(nn.Module):
    def __init__(self, inplace = True):
        super(Hsigmoid, self).__init__()
        self.inplace = inplace

    def forward(self, x):
        return F.relu6(x + 3., inplace = self.inplace) / 6.

#定义 SE 模块
class SEModule(nn.Module):
    def __init__(self, channel, reduction = 4):
        super(SEModule, self).__init__()
        self.avg_pool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Sequential(
            nn.Linear(channel, channel // reduction, bias = False),
            nn.ReLU(inplace = True),
            nn.Linear(channel // reduction, channel, bias = False),
            Hsigmoid()
            # nn.Sigmoid()
        )

    def forward(self, x):
```

```
        b, c, _, _ = x.size()
        y = self.avg_pool(x).view(b, c)
        y = self.fc(y).view(b, c, 1, 1)
        return x * y.expand_as(x)


class Identity(nn.Module):
    def __init__(self, channel):
        super(Identity, self).__init__()

    def forward(self, x):
        return x


def make_divisible(x, divisible_by=8):
    import numpy as np
    return int(np.ceil(x * 1. / divisible_by) * divisible_by)

#定义 Bottleneck
class MobileBottleneck(nn.Module):
    def __init__(self, inp, oup, kernel, stride, exp, se=False, nl='RE'):
        super(MobileBottleneck, self).__init__()
        assert stride in [1, 2]
        assert kernel in [3, 5]
        padding = (kernel - 1) // 2
        self.use_res_connect = stride == 1 and inp == oup
        conv_layer = nn.Conv2d
        norm_layer = nn.BatchNorm2d
        if nl == 'RE':
            nlin_layer = nn.ReLU # or ReLU6
        elif nl == 'HS':
            nlin_layer = Hswish
        else:
            raise NotImplementedError
        if se:
            SELayer = SEModule
        else:
            SELayer = Identity

        self.conv = nn.Sequential(
            # pw
            conv_layer(inp, exp, 1, 1, 0, bias=False),
            norm_layer(exp),
            nlin_layer(inplace=True),
            # dw
            conv_layer(exp, exp, kernel, stride, padding, groups=exp, bias=False),
            norm_layer(exp),
             SELayer(exp),
             nlin_layer(inplace=True),
             # pw-linear
```

```
103.             conv_layer(exp, oup, 1, 1, 0, bias = False),
104.             norm_layer(oup),
105.         )
106.
107.     def forward(self, x):
108.         if self.use_res_connect:
109.             return x + self.conv(x)
110.         else:
111.             return self.conv(x)
112.
113. # 定义 MobileNet V2 网络类
114. class MobileNet V3(nn.Module):
115.     def __init__(self, n_class = 1000, input_size = 224, dropout = 0.8, mode = 'small',
         width_mult = 1.0):
116.         super(MobileNet V3, self).__init__()
117.         input_channel = 16
118.         last_channel = 1280
119.         if mode == 'large':
120.             # 对应表 5 - 7 的网络结构
121.             mobile_setting = [
122.                 # k, exp, c, se,     nl, s,
123.                 [3, 16, 16, False, 'RE', 1],
124.                 [3, 64, 24, False, 'RE', 2],
125.                 [3, 72, 24, False, 'RE', 1],
126.                 [5, 72, 40, True, 'RE', 2],
127.                 [5, 120, 40, True, 'RE', 1],
128.                 [5, 120, 40, True, 'RE', 1],
129.                 [3, 240, 80, False, 'HS', 2],
130.                 [3, 200, 80, False, 'HS', 1],
131.                 [3, 184, 80, False, 'HS', 1],
132.                 [3, 184, 80, False, 'HS', 1],
133.                 [3, 480, 112, True, 'HS', 1],
134.                 [3, 672, 112, True, 'HS', 1],
135.                 [5, 672, 160, True, 'HS', 2],
136.                 [5, 960, 160, True, 'HS', 1],
137.                 [5, 960, 160, True, 'HS', 1],
138.             ]
139.         elif mode == 'small':
140.             # 对应表 5 - 8 的网络结构
141.             mobile_setting = [
142.                 # k, exp, c, se,     nl, s,
143.                 [3, 16, 16, True, 'RE', 2],
144.                 [3, 72, 24, False, 'RE', 2],
145.                 [3, 88, 24, False, 'RE', 1],
146.                 [5, 96, 40, True, 'HS', 2],
147.                 [5, 240, 40, True, 'HS', 1],
148.                 [5, 240, 40, True, 'HS', 1],
149.                 [5, 120, 48, True, 'HS', 1],
150.                 [5, 144, 48, True, 'HS', 1],
151.                 [5, 288, 96, True, 'HS', 2],
```

```
                [5, 576, 96, True, 'HS', 1],
                [5, 576, 96, True, 'HS', 1],
            ]
        else:
            raise NotImplementedError

        # 构建第一层
        assert input_size % 32 == 0
        last_channel = make_divisible(last_channel * width_mult) if width_mult > 1.0
        else last_channel
        self.features = [conv_bn(3, input_channel, 2, nlin_layer = Hswish)]
        self.classifier = []

        # 构建 mobile blocks
        for k, exp, c, se, nl, s in mobile_setting:
            output_channel = make_divisible(c * width_mult)
            exp_channel = make_divisible(exp * width_mult)
            self.features.append(MobileBottleneck(input_channel, output_channel, k, s,
            exp_channel, se, nl))
            input_channel = output_channel

        # 构建最后几层
        if mode == 'large':
            last_conv = make_divisible(960 * width_mult)
            self.features.append(conv_1x1_bn(input_channel, last_conv, nlin_layer =
            Hswish))
            self.features.append(nn.AdaptiveAvgPool2d(1))
            self.features.append(nn.Conv2d(last_conv, last_channel, 1, 1, 0))
            self.features.append(Hswish(inplace = True))
        elif mode == 'small':
            last_conv = make_divisible(576 * width_mult)
            self.features.append(conv_1x1_bn(input_channel, last_conv, nlin_layer =
            Hswish))
            # self.features.append(SEModule(last_conv)) # refer to paper Table2, but I
            think this is a mistake
            self.features.append(nn.AdaptiveAvgPool2d(1))
            self.features.append(nn.Conv2d(last_conv, last_channel, 1, 1, 0))
            self.features.append(Hswish(inplace = True))
        else:
            raise NotImplementedError

        # 构建 nn.Sequential
        self.features = nn.Sequential( * self.features)

        # 构建分类器层
        self.classifier = nn.Sequential(
            nn.Dropout(p = dropout),     # refer to paper section 6
            nn.Linear(last_channel, n_class),
        )

```

```
197.         self._initialize_weights()
198.
199.     def forward(self, x):
200.         x = self.features(x)
201.         x = x.mean(3).mean(2)
202.         x = self.classifier(x)
203.         return x
204.         #权重初始化
205.     def _initialize_weights(self):
206.         for m in self.modules():
207.             if isinstance(m, nn.Conv2d):
208.                 nn.init.kaiming_normal_(m.weight, mode='fan_out')
209.                 if m.bias is not None:
210.                     nn.init.zeros_(m.bias)
211.             elif isinstance(m, nn.BatchNorm2d):
212.                 nn.init.ones_(m.weight)
213.                 nn.init.zeros_(m.bias)
214.             elif isinstance(m, nn.Linear):
215.                 nn.init.normal_(m.weight, 0, 0.01)
216.                 if m.bias is not None:
217.                     nn.init.zeros_(m.bias)
218.
219.
220. def mobilenetv3(pretrained=False, **kwargs):
221.     model = MobileNetV3(**kwargs)
222.     if pretrained:
223.         state_dict = torch.load('mobilenetv3_small_67.4.pth.tar')
224.         model.load_state_dict(state_dict, strict=True)
225.
226.     return model
227.
228.
229. if __name__ == '__main__':
230.     net = mobilenetv3(n_class=2)
231.     print('mobilenetv3:\n', net)
232.     print('Total params: %.2fM' % (sum(p.numel() for p in net.parameters())/1000000.0))
233.     input_size=(1, 3, 224, 224)
234.     x = torch.randn(input_size)
235. out = net(x)
236. #构建 SE 模块
237. class SEModule(nn.Module):
238.     def __init__(self, channel, reduction=4):
239.         super(SEModule, self).__init__()
240.         self.avg_pool = nn.AdaptiveAvgPool2d(1)
241.         self.fc = nn.Sequential(
242.             nn.Linear(channel, channel // reduction, bias=False),
243.             nn.ReLU(inplace=True),
244.             nn.Linear(channel // reduction, channel, bias=False),
245.             Hsigmoid()
246.             # nn.Sigmoid()
```

```
247.          )
248.
249.      def forward(self, x):
250.          b, c, _, _ = x.size()
251.          y = self.avg_pool(x).view(b, c)
252.          y = self.fc(y).view(b, c, 1, 1)
253.          return x * y.expand_as(x)
254.
255. # Swish 和 H-Swish 激活函数
256. import torch
257. from torch import nn
258. import torch.nn.functional as F
259. import numpy as np
260. import matplotlib.pyplot as plt
261. from torch.autograd import Variable
262.
263. class Hswish(nn.Module):
264.     def __init__(self, inplace=True):
265.         super(Hswish, self).__init__()
266.         self.inplace = inplace
267.
268.     def forward(self, x):
269.         return x * F.relu6(x + 3., inplace=self.inplace) / 6.
270.
271. class Hsigmoid(nn.Module):
272.     def __init__(self, inplace=True):
273.         super(Hsigmoid, self).__init__()
274.         self.inplace = inplace
275.
276.     def forward(self, x):
277.         return F.relu6(x + 3., inplace=self.inplace) / 6
278.
279. def _group_conv(x, filters, kernel, stride, groups):
280.     """
281.     Group convolution
282.     # Arguments
283.         x: Tensor, input tensor of with `channels_last` or 'channels_first' data format
284.         filters: Integer, number of output channels
285.         kernel: An integer or tuple/list of 2 integers, specifying the
286.             width and height of the 2D convolution window.
287.         strides: An integer or tuple/list of 2 integers,
288.             specifying the strides of the convolution along the width and height.
289.             Can be a single integer to specify the same value for
290.             all spatial dimensions.
291.         groups: Integer, number of groups per channel
292.
293.     # Returns
294.         Output tensor
295.     """
296.
```

```
    channel_axis = 1 if K.image_data_format() == 'channels_first' else -1
    in_channels = K.int_shape(x)[channel_axis]

    #每组输入通道数量
    nb_ig = in_channels // groups
    #每组输出的通道数量
    nb_og = filters // groups

    gc_list = []
    #确定过滤器的数量是否可被组的数量整除
    assert filters % groups == 0

    for i in range(groups):
        if channel_axis == -1:
            x_group = Lambda(lambda z: z[:, :, :, i * nb_ig: (i + 1) * nb_ig])(x)
        else:
            x_group = Lambda(lambda z: z[:, i * nb_ig: (i + 1) * nb_ig, :, :])(x)
        gc_list.append(Conv2D(filters = nb_og, kernel_size = kernel, strides = stride,
        padding = 'same', use_bias = False)(x_group))

    return Concatenate(axis = channel_axis)(gc_list)
```

5.4.2　ShuffleNet V2 模型构建

```
def split(x, groups):
    out = x.chunk(groups, dim = 1)

    return out


def shuffle( x, groups):
    N, C, H, W = x.size()
     out = x.view(N, groups, C // groups, H, W).permute(0, 2, 1, 3, 4).contiguous().view
     (N, C, H, W)
     return out
#构建 Shuffle 类
class ShuffleUnit(nn.Module):
    def __init__(self, in_channels, out_channels, stride):
        super().__init__()
        mid_channels = out_channels // 2
        if stride > 1:
            self.branch1 = nn.Sequential(
                nn.Conv2d( in_channels, in_channels, 3, stride = stride, padding = 1,
                groups = in_channels, bias = False),
                nn.BatchNorm2d(in_channels),
                nn.Conv2d(in_channels, mid_channels, 1, bias = False),
                nn.BatchNorm2d(mid_channels),
                nn.ReLU( inplace = True)
```

```
            )
            self.branch2 = nn.Sequential(
                nn.Conv2d(in_channels, mid_channels, 1, bias = False),
                nn.BatchNorm2d(mid_channels),
                nn.ReLU(inplace = True),
                nn.Conv2d(mid_channels, mid_channels, 3, stride = stride, padding = 1,
                groups = mid_channels, bias = False),
                nn.BatchNorm2d(mid_channels),
                nn.Conv2d(mid_channels, mid_channels, 1, bias = False),
                nn.BatchNorm2d(mid_channels),
                nn.ReLU(inplace = True)
            )
        else:
            self.branch1 = nn.Sequential()
            self.branch2 = nn.Sequential(
                nn.Conv2d(mid_channels, mid_channels, 1, bias = False),
                nn.BatchNorm2d(mid_channels),
                nn.ReLU(inplace = True),
                nn.Conv2d(mid_channels, mid_channels, 3, stride = stride, padding = 1,
                groups = mid_channels, bias = False),
                nn.BatchNorm2d(mid_channels),
                nn.Conv2d(mid_channels, mid_channels, 1, bias = False),
                nn.BatchNorm2d(mid_channels),
                nn.ReLU(inplace = True)
            )
        self.stride = stride
    def forward(self, x):
        if self.stride == 1:
            x1, x2 = split(x, 2)
            out = torch.cat((self.branch1(x1), self.branch2(x2)), dim = 1)
        else:
            out = torch.cat((self.branch1(x), self.branch2(x)), dim = 1)
        out = shuffle(out, 2)
        return out
# 构建 ShuffleNet V2 网络结构
class ShuffleNet V2(nn.Module):
    def __init__(self, channel_num, class_num = settings.CLASSES_NUM):
        super().__init__()
        self.conv1 = nn.Sequential(
                nn.Conv2d(3, 24, 3, stride = 2, padding = 1, bias = False),
                nn.BatchNorm2d(24),
                nn.ReLU(inplace = True)
        )
        self.maxpool = nn.MaxPool2d(kernel_size = 3, stride = 2, padding = 1)
        self.stage2 = self.make_layers(24, channel_num[0], 4, 2)
        self.stage3 = self.make_layers(channel_num[0], channel_num[1], 8, 2)
        self.stage4 = self.make_layers(channel_num[1], channel_num[2], 4, 2)
        self.conv5 = nn.Sequential(
                nn.Conv2d(channel_num[2], 1024, 1, bias = False),
                nn.BatchNorm2d(1024),
```

```
                nn.ReLU(inplace = True)
        )
        self.avgpool = nn.AdaptiveAvgPool2d(1)
        self.fc = nn.Linear(1024, class_num)
    def make_layers(self, in_channels, out_channels, layers_num, stride):
        layers = []
        layers.append(ShuffleUnit(in_channels, out_channels, stride))
        in_channels = out_channels
        for i in range(layers_num - 1):
            ShuffleUnit(in_channels, out_channels, 1)
        return nn.Sequential( * layers)
    def forward(self, x):
        out = self.conv1(x)
        out = self.maxpool(out)
        out = self.stage2(out)
        out = self.stage3(out)
        out = self.stage4(out)
        out = self.conv5(out)
        out = self.avgpool(out)
        out = out.flatten(1)
        out = self.fc(out)
        return out
```

5.5　本章小结

本章介绍了两类近年来最受关注的轻量级 DCNN 模型，MobileNet 系列和 ShuffleNet 系列，分别由 Google 公司与旷视科技提出。本章详细阐述了两种系列算法的设计思路、改进的原理与对比实验结果，并给出网络模型结构图、对比实验结果及代码。

5.6　习题

1. 常用的轻量化 DCNN 模型有哪些？
2. 轻量化模型构建的基本思路有哪些？
3. 简述 MobileNet 系列算法主要的设计思路。
4. 简述 ShuffleNet 系列算法主要的设计思路。