首都师范大学

为学为师 求新求实

深度学习应用与工程实践
9. 轻量化神经网络架构设计
9. Compact Neural Architecture Design

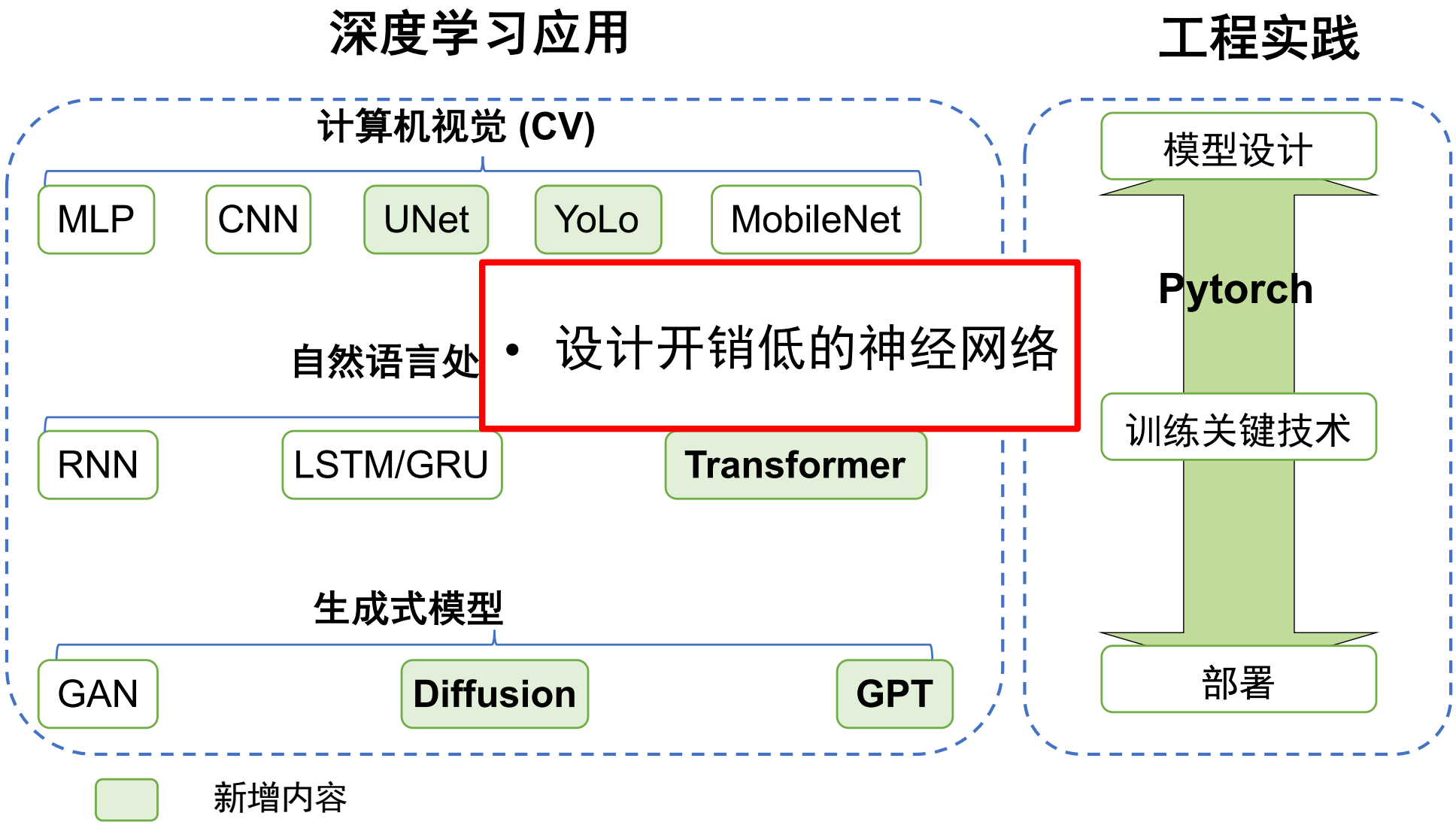
李冰

副研究员

交叉科学研究院

首都师范大学

本门课的内容


深度学习应用
工程实践
计算机视觉 (CV)
MLP
CNN
UNet
YoLo
MobileNet
自然语言处
RNN
LSTM/GRU
Transformer
生成式模型
GAN
Diffusion
GPT
• 设计开销低的神经网络
模型设计
Pytorch
训练关键技术
部署
新增内容

卷积神经网络的资源开销

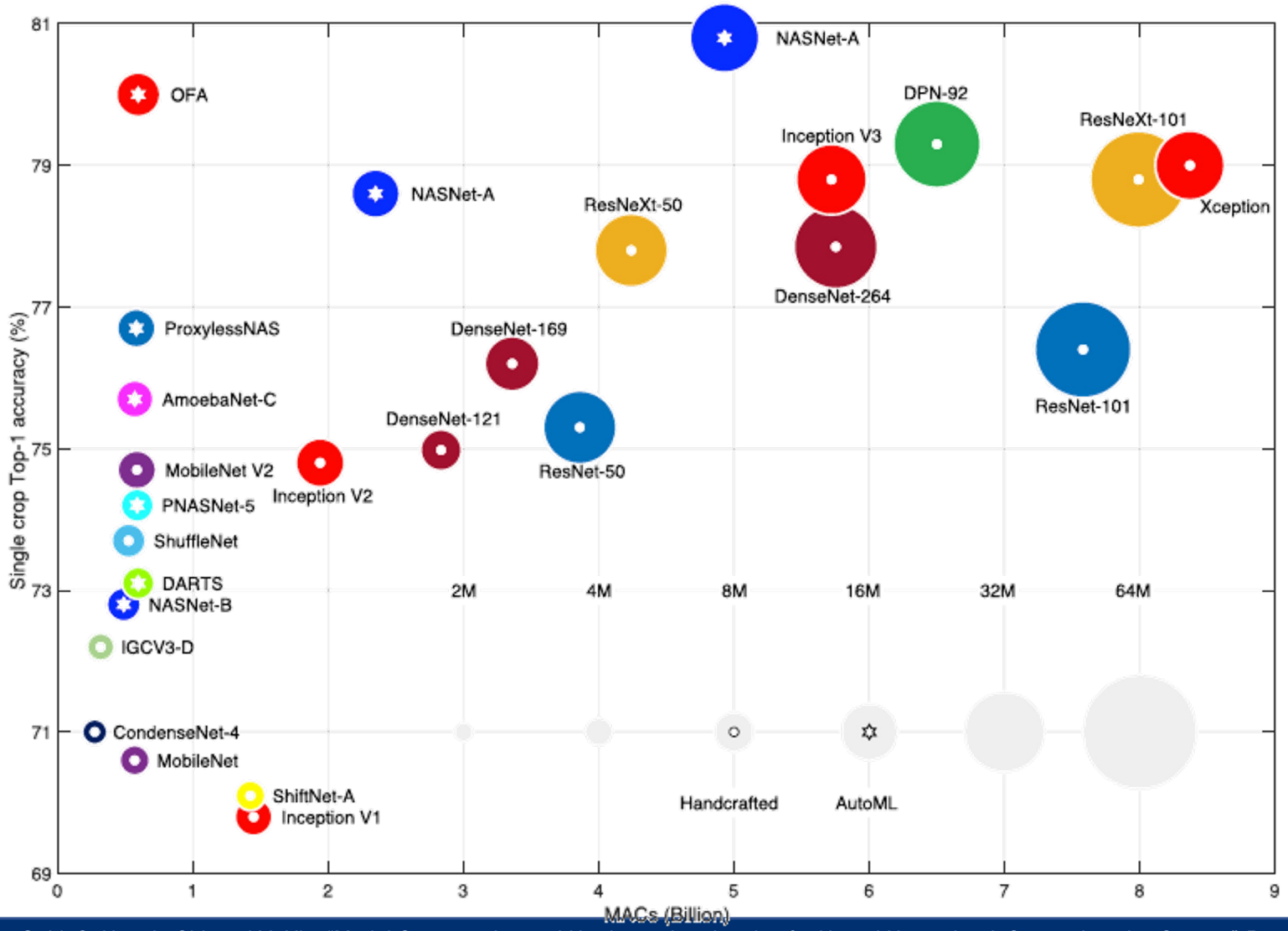
- 卷积神经网络的资源开销

网络	输入大小	参数量	MACs
AlexNet	224x224	233 MB	727 M
VGG-16/19	224x224	528 MB / 548 MB	16 G /20 G
GoogLeNet	224x224	51 MB	2 G
ResNet-18/34	224x224	45 MB / 83 MB	2 G/ 4 G
DenseNet-121	224x224	31 MB	3 G

- 移动端卷积神经网络的资源开销

网络	输入大小	参数量	MACs
SqueezeNet	224x224	**4.8 MB**	727 M
MobileNet	224x224	**16.8 MB**	575 M
ShuffleNet	224x224	**13.6 MB**	292 M
MobileNet-V2	224x224	**13.6 MB**	300 M

卷积神经网络的资源开销

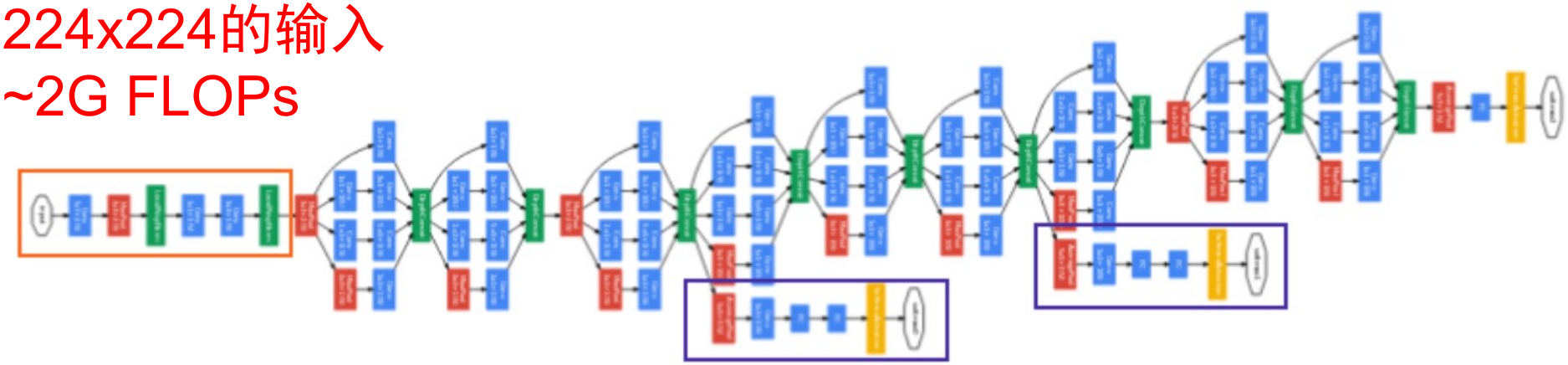


L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020, doi: 10.1109/JPROC.2020.2976475.

回顾: GoogLeNet

- GoogLeNet 的设计是为了取得更高的精度.
- 早期的设计都是这种设计思路.
- 并不会考虑模型的执行效率. 像GoogLeNet 这种大型模型是很难在移动设备上部署的。

224x224的输入
~2G FLOPs



Szegedy, Christian, et al. "Going deeper with convolutions." (2015)

回顾: GoogLeNet

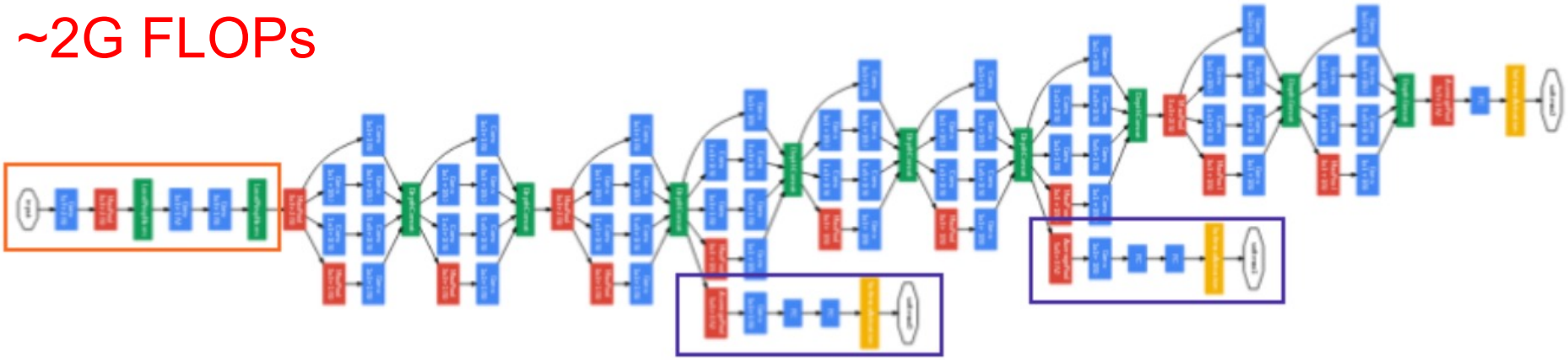
- 前向和后向的计算开销太高，使得训练非常慢

在Tesla K40上的训练

```
2016-03-07 12:26:28.905231: step 20, loss = 14.81 (9.5 examples/sec; 3.380 sec/batch)
2016-03-07 12:27:02.699719: step 30, loss = 14.45 (9.5 examples/sec; 3.378 sec/batch)
2016-03-07 12:27:36.515699: step 40, loss = 13.98 (9.5 examples/sec; 3.376 sec/batch)
2016-03-07 12:28:10.220956: step 50, loss = 13.92 (9.6 examples/sec; 3.327 sec/batch)
2016-03-07 12:28:43.658223: step 60, loss = 13.28 (9.6 examples/sec; 3.350 sec/batch)
```

Source: https://github.com/tensorflow/models/tree/master/research/inception

224x224输入
~2G FLOPs



高效的网络架构

- SqueezeNet
- MobileNet
- ShuffleNet
- MobileNet-V2

SqueezeNet

SqueezeNet旨在减少CNN模型的**内存开销**。 它使用3种策略来设计高效的网络。

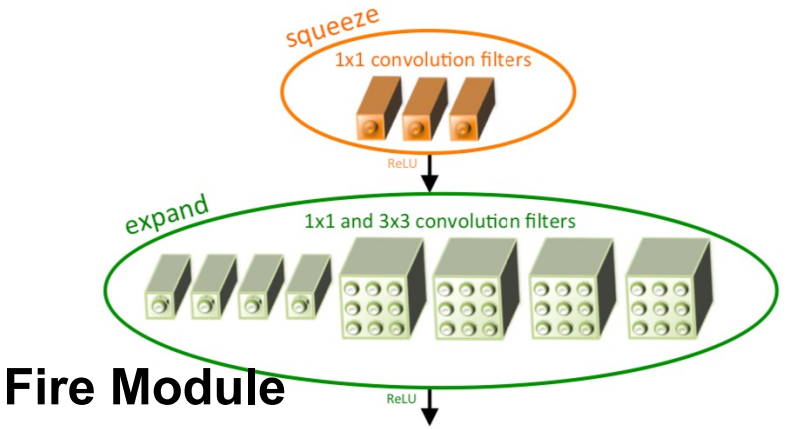
特点：

- SqueezeNet用1x1卷积核替换了某些3x3卷积核。
- SqueezeNet使用压缩层将到3x3过滤器的输入通道的数量减少。
- SqueezeNet在网络后期进行下采样以启用大型激活图。**延迟下采样**会带来更高的精度。

SqueezeNet – “Fire” 模块

“Fire” 模块

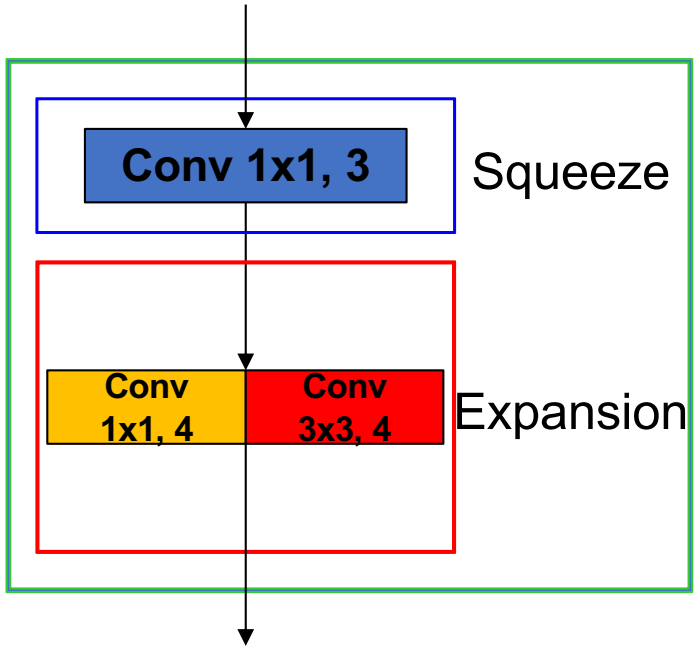
每个 fire 有压缩（squeeze）和扩张（expand）层.



Fire Module

压缩层: 像瓶颈结构， 压缩层把特征图映射到更低维度，能提高计算效能。

扩张层: 1×1 和3×3 卷积核的混合，来提取特征.



SqueezeNet – 延迟下采样

- 延迟下采样, SqueezeNet 的特征图尺寸很大，最后是一个平均池化层。

layer name/type	output size	filter size / stride (if not a fire layer)	depth	s_{1x1} (#1x1 squeeze)	e_{1x1} (#1x1 expand)	e_{3x3} (#3x3 expand)	s_{1x1} sparsity	e_{1x1} sparsity	e_{3x3} sparsity	# bits	#parameter before pruning	#parameter after pruning
input image	224x224x3										-	-
conv1	111x111x96	7x7/2 (x96)	1				100% (7x7)			6bit	14,208	14,208
maxpool1	55x55x96	3x3/2	0									
fire2	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	11,920	5,746
fire3	55x55x128		2	16	64	64	100%	100%	**33%**	6bit	12,432	6,258
fire4	55x55x256		2	32	128	128	100%	100%	**33%**	6bit	45,344	20,646
maxpool4	27x27x256	3x3/2	0									
fire5	27x27x256		2	32	128	128	100%	100%	**33%**	6bit	49,440	24,742
fire6	27x27x384		2	48	192	192	100%	**50%**	**33%**	6bit	104,880	44,700
fire7	27x27x384		2	48	192	192	**50%**	100%	**33%**	6bit	111,024	46,236
fire8	27x27x512		2	64	256	256	100%	**50%**	**33%**	6bit	188,992	77,581
maxpool8	13x12x512	3x3/2	0									
fire9	13x13x512		2	64	256	256	**50%**	100%	**30%**	6bit	197,184	77,581
conv10	13x13x1000	1x1/1 (x1000)	1				**20%** (3x3)			6bit	513,000	103,400
avgpool10	1x1x1000	13x13/1	0									
	activations			parameters			compression info				1,248,424 (total)	**421,098** (total)

Iandola, Forrest N., et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size." (2016)

SqueezeNet – 结果

- 相比于AlexNet, SqueezeNet 50倍的参数量下降, 但是相似的精确度.

Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network Pruning (Han et al., 2015b)	32 bit	240MB → 27MB	9x	57.2%	80.3%
AlexNet	Deep Compression (Han et al., 2015a)	5-8 bit	240MB → 6.9MB	35x	57.2%	80.3%
SqueezeNet (ours)	None	32 bit	4.8MB	**50x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	8 bit	4.8MB → 0.66MB	**363x**	57.5%	80.3%
SqueezeNet (ours)	Deep Compression	6 bit	4.8MB → 0.47MB	**510x**	57.5%	80.3%

MobileNet

MobileNet 的设计目标:

- 为轻量级神经体系结构构建精简，可扩展的体系结构。
- 在图像分类任务上达到最先进的资源和效率的权衡。
- 结构可以很容易迁移，用来执行其他视觉任务（例如物体检测）。

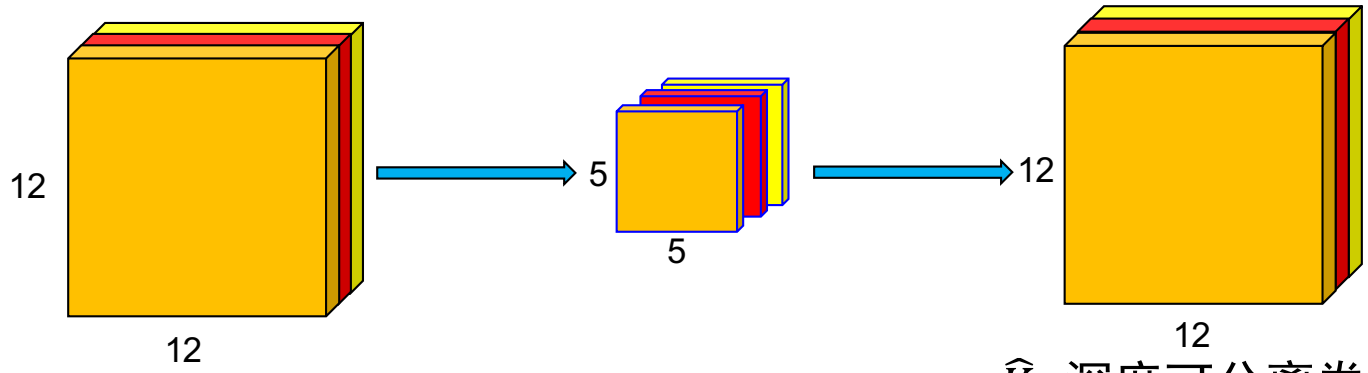
- **特点：**
- 引入**深度可分离**（ depthwise separable ）**卷积**以减少运算中的计算开销。
- 介绍各种方法来构建可伸缩的神经网络模型。

Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

MobileNet

深度卷积

- 每个输入通道只有一个连接到输出通道。
- 在这里，我们举一个5×5深度卷积的例子。 这里我们使用padding来保持特征图的大小不变.



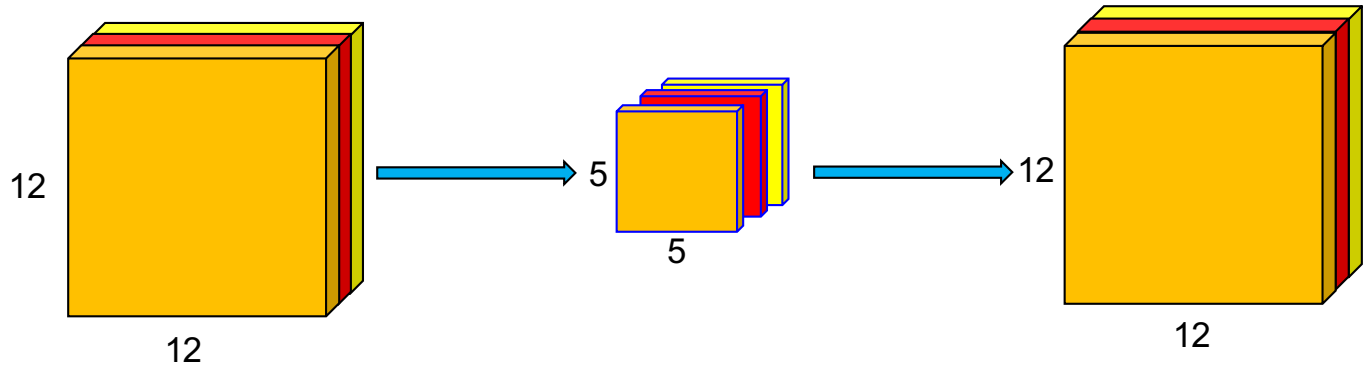
$$G_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,k} F_{k+i-1,l+j-1,m}$$

$\hat{K}$: 深度可分离卷积
G: 输出特征图
F: 特征图
i, j, m: 索引

MobileNet

深度卷积

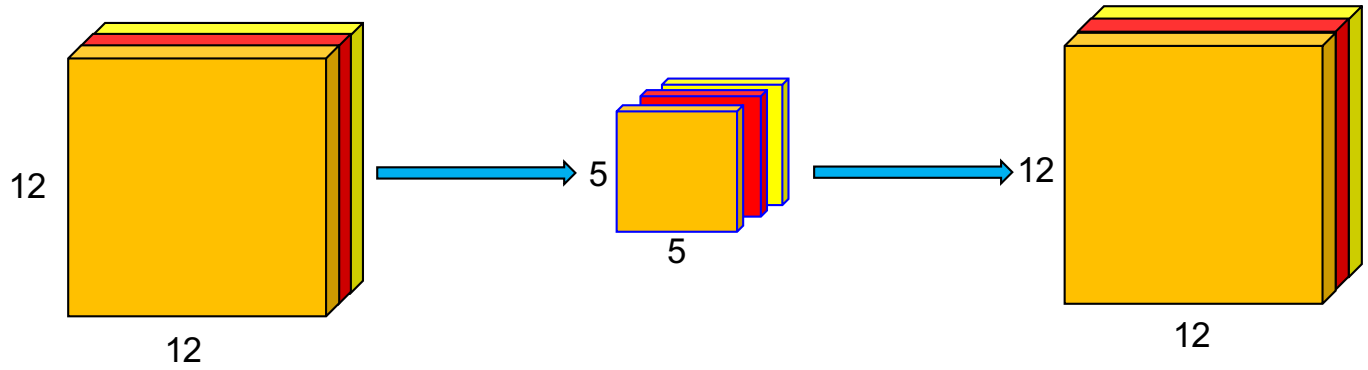
Q: 这里的网络参数和MACs分别是多少?



MobileNet

- 深度卷积

Q: 这里的网络参数和MACs分别是多少?



A:

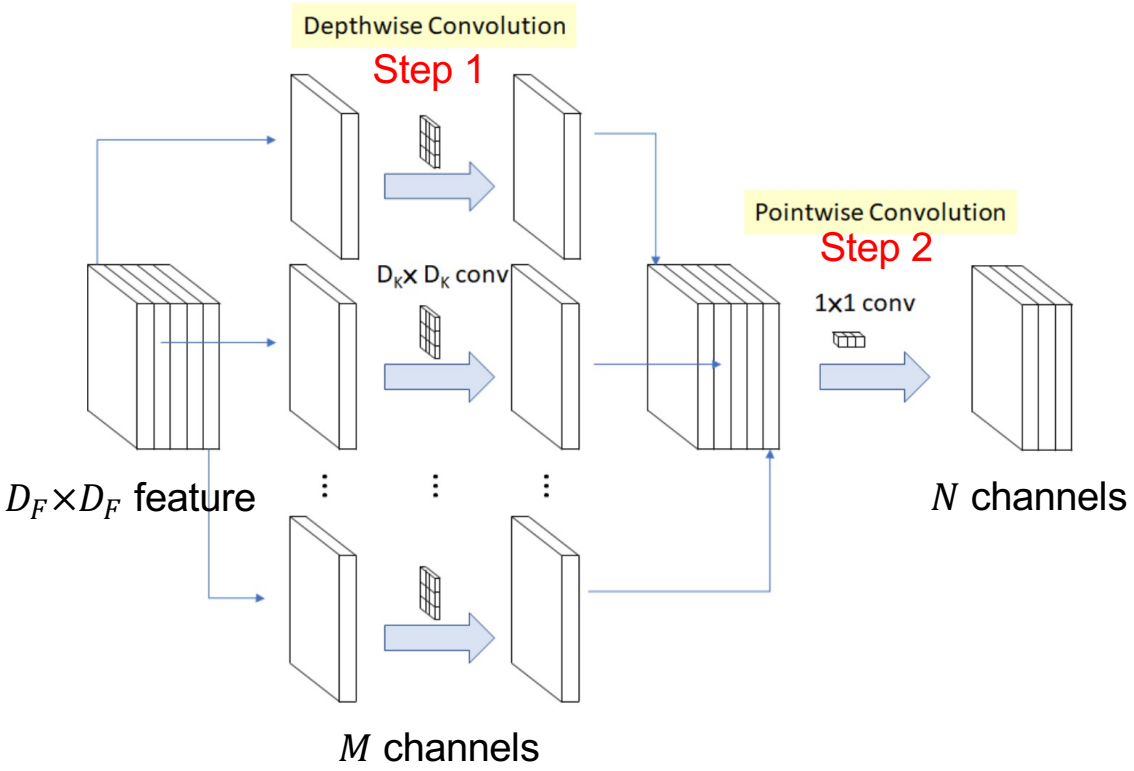
参数: $3\times5\times5+3=78$

MACs: $3\times5\times5\times12\times12=10800$

MobileNet

•深度可分卷积

普通的$\mathrm{D}_k \times D_k$ 卷积分解成 $\mathrm{D}_k \times D_k$ 深度卷积和 1×1 卷积，能够降低参数量和计算开销. 1×1 卷积是**点式卷积**.

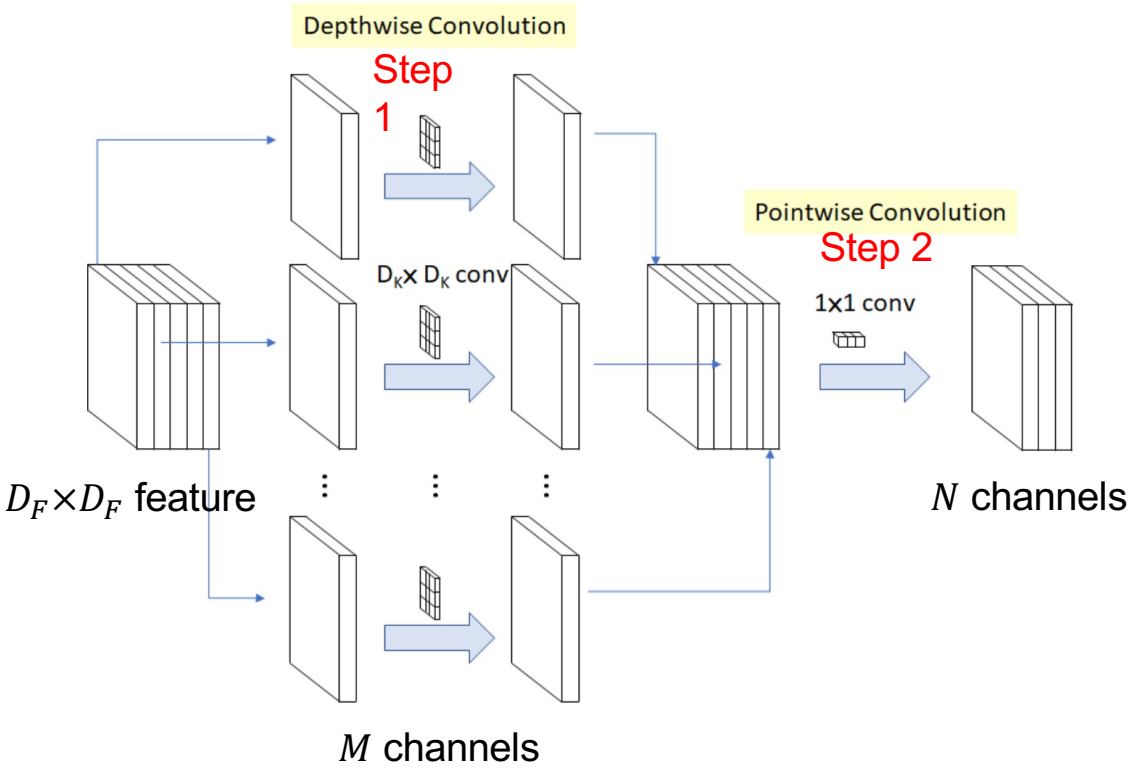


Step 1: 深度卷积 *提取每个通道的空间信息*

Step 2: 点式卷积 把深度卷积每个通道的结果投射到新的通道空间.

MobileNet

- **深度可分卷积**
  - 深度可分卷积的步长为 1，执行填充，这样输出特征图的大小与输入特征图大小相同。每一步的参数量和MACs 分布式多少？



深度卷积:

$D_K \times D_K \times M + M$ 参数量

$D_K \times D_K \times M \times D_F \times D_F$ MACs

点式卷积:

$M \times N + N$ 参数量

$D_F \times D_F \times M \times N$ MACs

Source: https://towardsdatascience.com/review-mobilenetv1-depthwise-separable-convolution-light-weight-model-a382df364b69

MobileNet

•深度可分卷积

3×3 卷积核最常见，我们来分析一下深度可分卷积的潜能。

3×3 卷积有 $3\times3\times M\times N\times D_F\times D_F$ MACs 和 $3\times3\times M\times N+$ N参数.

深度可分卷积有 $3\times3\times M\times D_F\times D_F+D_F\times D_F\times M\times N$ MACs 和 $3\times3\times M+M\times N+M+N$参数.

当 N, M的值非常大, 理论的加速比是:

$$\frac{3\times3\times M\times N\times D_F\times D_F}{3\times3\times M\times D_F\times D_F+\ D_F\times D_F\times M\times N}\approx 9$$

MobileNet

- 深度可分卷积

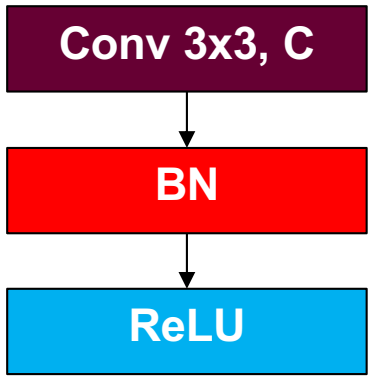
Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

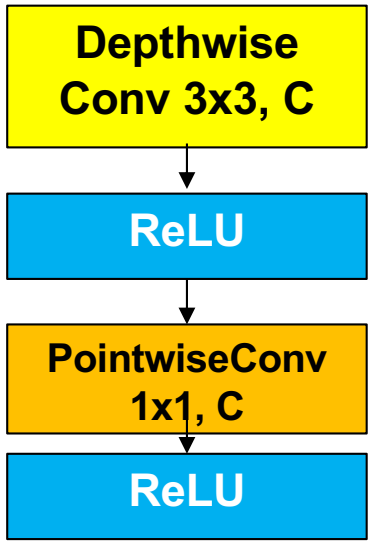
- 相比普通卷积，深度可分卷积有9倍的MAC减少.
- 在 ImageNet-1K 分类任务上只有 1% top-1 精度下降.

MobileNet

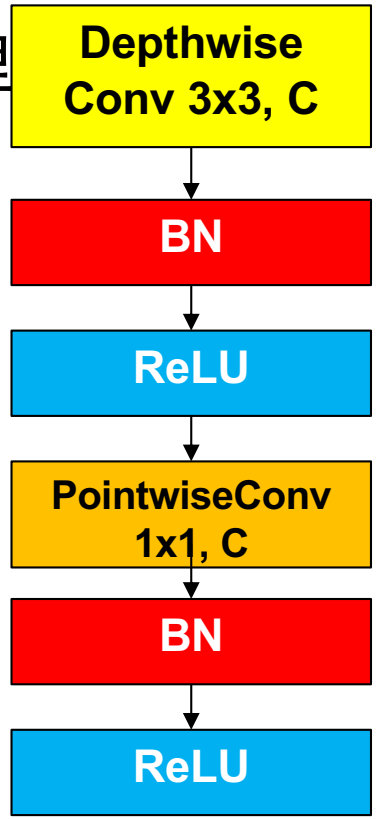
- 深度可分卷积
  - 激活层和batch normalization的位置在哪里



带BN的普通卷积



不带BN的深度可分卷积



带BN的深度可分卷积

MobileNet

- **MobileNet架构**

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

dw: 深度卷积
Conv: 普通 (点式卷积)
s: 步长 “s2” 指步长为2.

1x1 卷积占了 94.86% 的计算量，和74.59% 参数量，是影响效率的瓶颈.

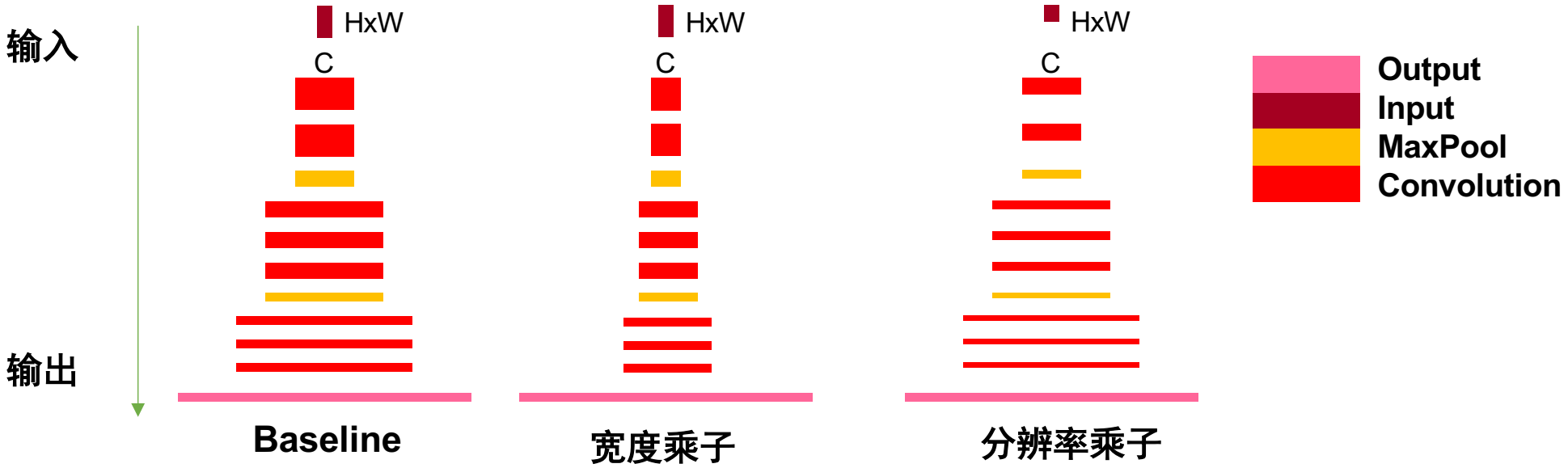
MobileNet

稳定的网络结构构建

- 与其设计更小的网络架构， MobileNet 应用宽度乘子和分辨率乘子来降低神经架构开销.

宽度乘子：缩小每层的通道数.

分辨率乘子: 缩小输入图片的大小，每层特征图的大小随之缩小



MobileNet

- MobileNet采用宽度和分辨率乘法之后取得的精度和资源权衡

Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

Resolution	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
1.0 MobileNet-192	69.1%	418	4.2
1.0 MobileNet-160	67.2%	290	4.2
1.0 MobileNet-128	64.4%	186	4.2

MobileNetV2

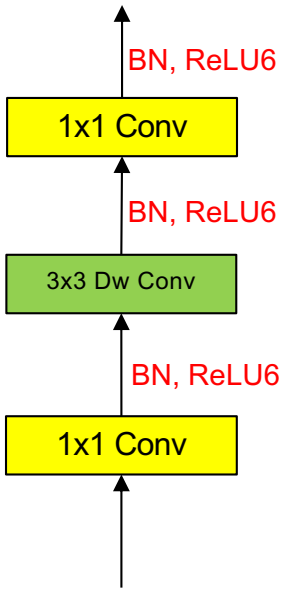
- MobileNetV2 进一步提高了模型性能同时保持高性能.

- **特点:**
  - MobileNetV2 用 **inverted bottleneck 提升表征能力**.
  - 类似MobileNet, 宽度乘子和分辨率乘子在 MobileNetV2模型中被深度采用.

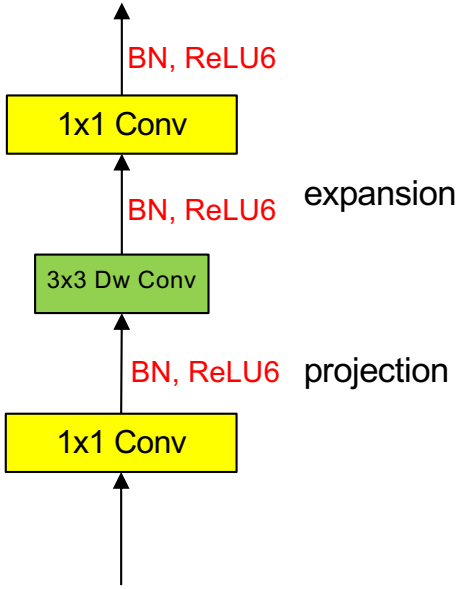
MobileNetV2

Inverted bottleneck 翻转瓶颈

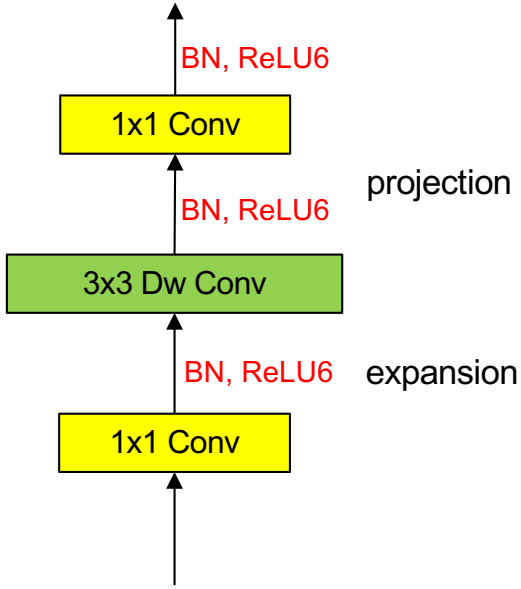
中间是‘宽’特征图，两端是‘窄’特征度.



每层卷积宽度相同



‘窄’特征图在中间，‘宽’特征图在两段.



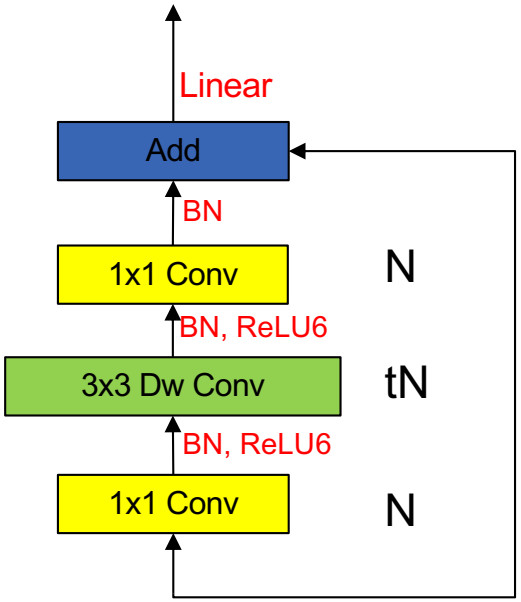
‘宽’特征图在中间，‘窄’特征图在两段.

MobileNetV2

MobileNetV2 模块

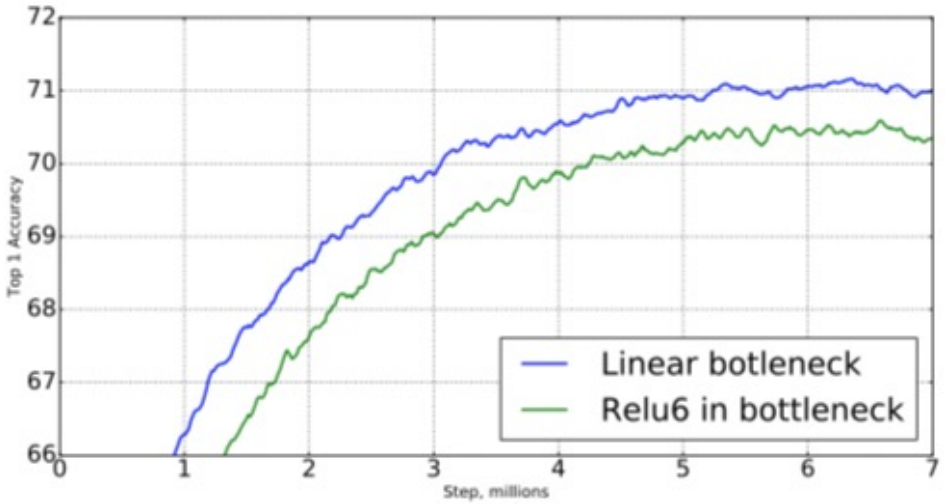
MobileNetV2 瓶颈包括残余连接和翻转瓶颈，实现收敛加速。

Fig: MobileNetV2 模块



MobileNetV2 每个瓶颈的最后没有激活函数，这样提高了模型的表征能力.

Linear vs. ReLU



Usually, t=6 in MobileNetV2.

MobileNetV2

MobileNet v2有多个MobileNet 瓶颈. 相比MobileNet, MobileNetV2翻转瓶颈产生更窄的特征图。

Table: MobileNetV2

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table: MobileNet

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

MobileNetV2

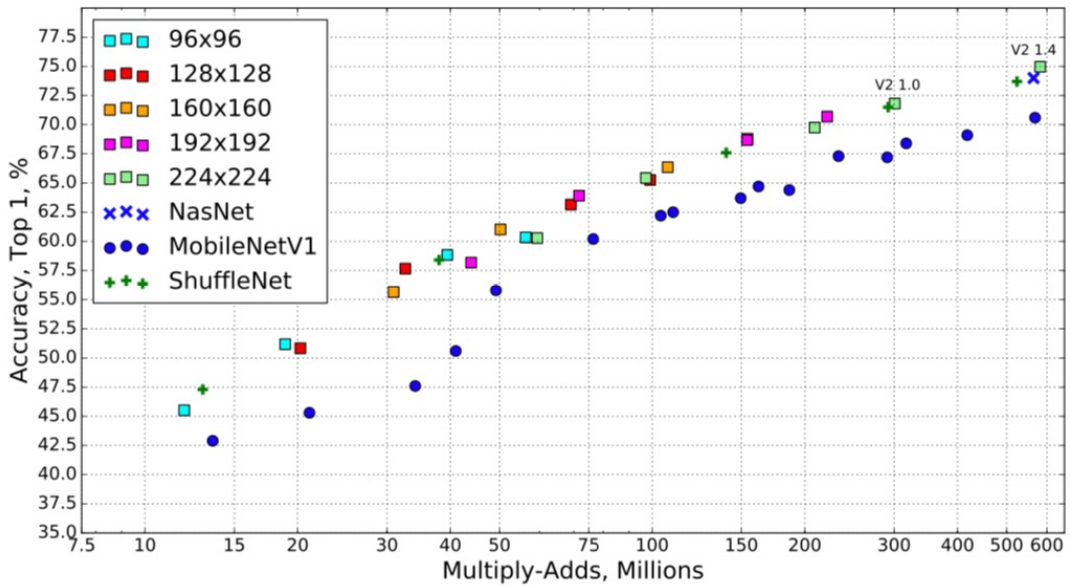
MobileNetV2结果

与V1版本比，用了翻转瓶颈后MobileNetV2性能提升显著. 与其他结构比，取得了更加的精度-计算权衡.

Table: 在ImageNet数据集上的Top1精度

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	**3.4M**	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	**72.0**	**3.4M**	**300M**	**75ms**
MobileNetV2 (1.4)	**74.7**	6.9M	585M	**143ms**

Table: 不同网络中精度与计算的权衡结果



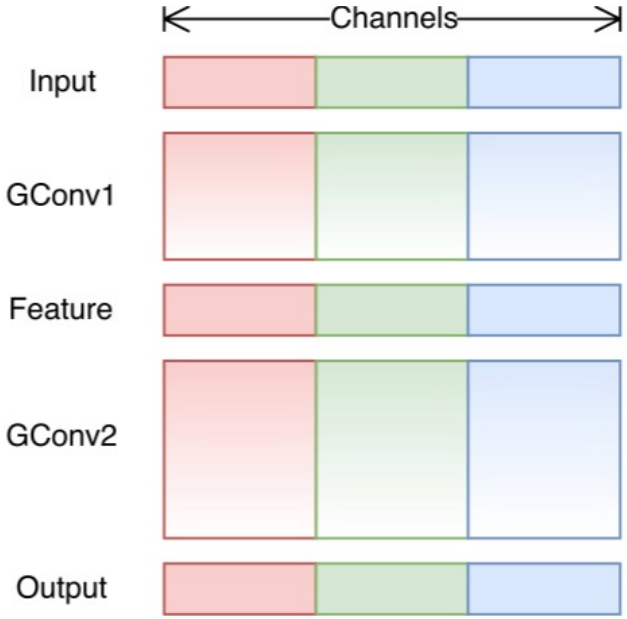
ShuffleNet

- ShuffleNet专为计算能力非常有限（100-150 MFLOPs）的移动设备而设计。
- ShuffleNet解决了代价高昂的1×1卷积问题，以进一步提高CNN架构的效率。

- **特征**：
- 使用1×1 **分组卷积**（group convolution）可减少1×1卷积的计算开销
- 引入通道混洗(channel shuffle)以在**分组卷积**中启用不同通道之间的串扰。

ShuffleNet

- **分组卷积**
- 分组卷积对输入通道的各个部分进行分组，并按通道分配权重以构造输出通道。
- 深度卷积是分组卷积的一种特殊情况，因为每个输入通道都是一个单独的组。



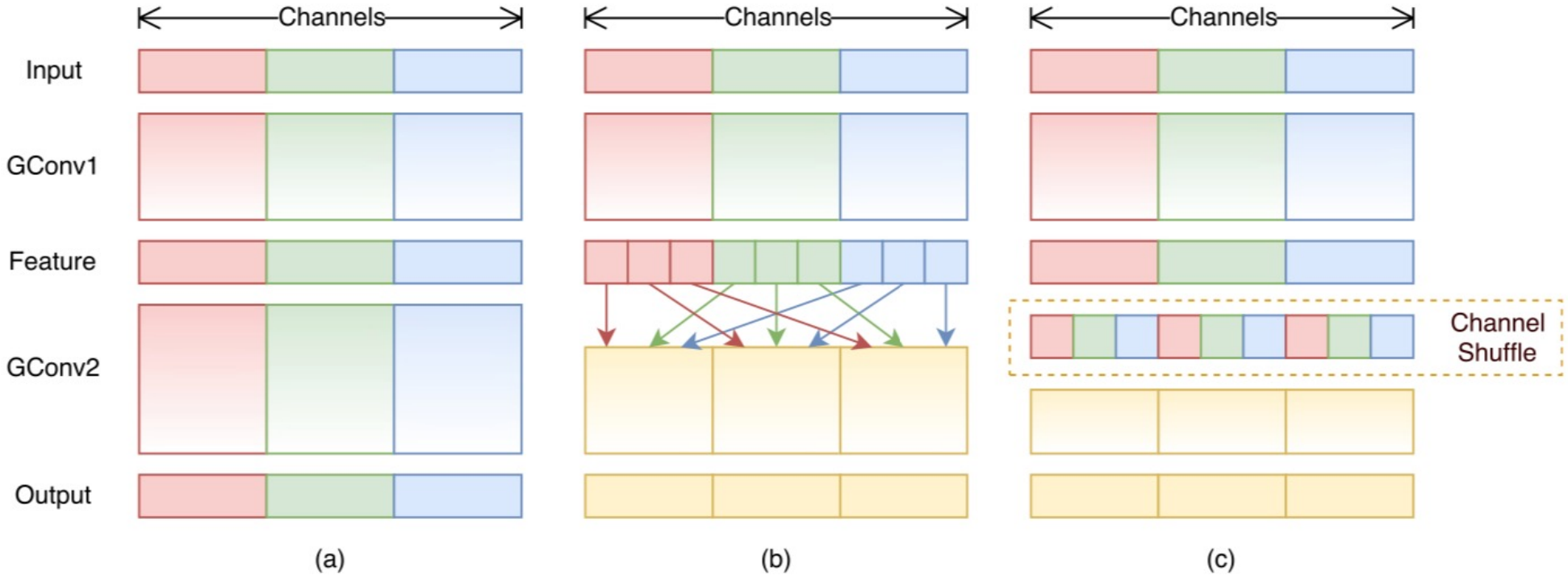
(a)

颜色相同的输入通道分在一组，执行相同的卷积。

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices."(2018)

ShuffleNet

- **通道混洗 Channel Shuffle**
- ShuffleNet 会混合不同分组卷积的输出，进一步增强了分组卷积的能力。



ShuffleNet

- **通道混洗 Channel shuffle**
- 通道混洗取得了显著的性能提升。

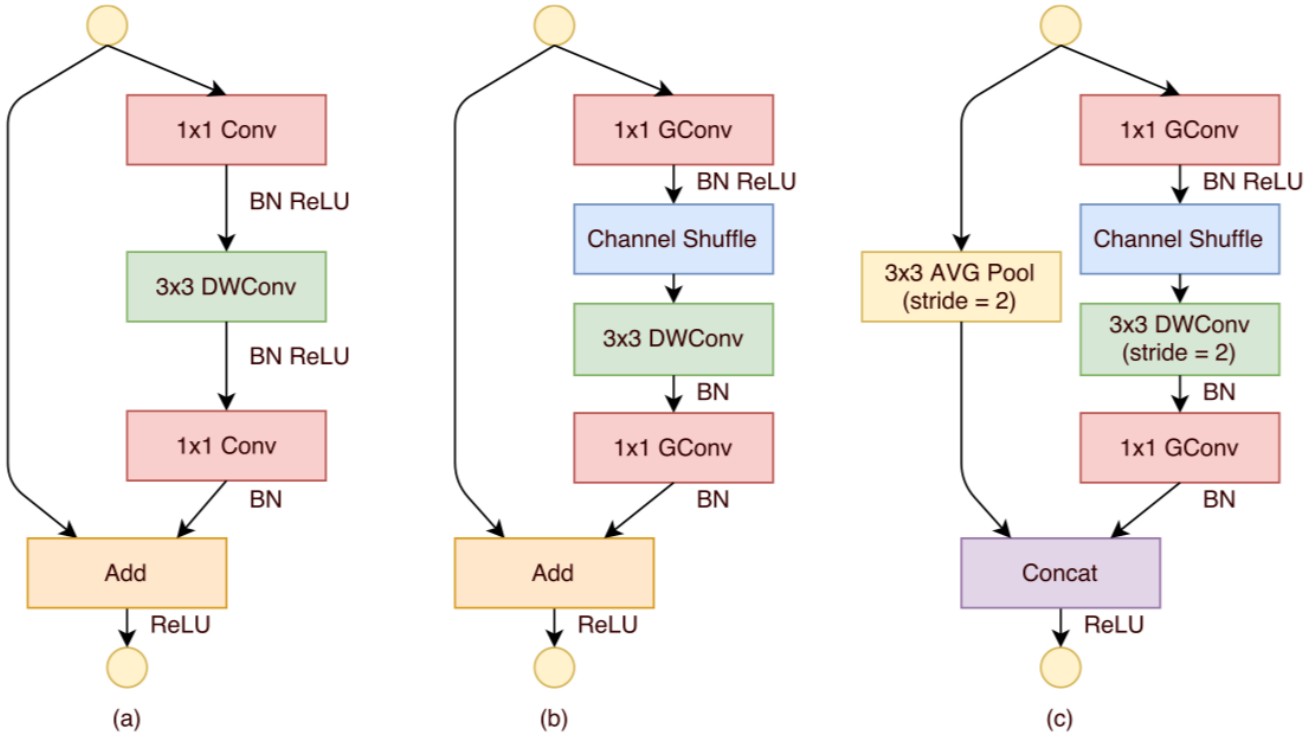
Table: 有/无通道混洗的ShuffleNet的性能对比

Model	Cls err. (%, no shuffle)	Cls err. (%, shuffle)	Δ err. (%)
ShuffleNet 1x ($g = 3$)	34.5	**32.6**	1.9
ShuffleNet 1x ($g = 8$)	37.6	**32.4**	5.2
ShuffleNet 0.5x ($g = 3$)	45.7	**43.2**	2.5
ShuffleNet 0.5x ($g = 8$)	48.1	**42.3**	5.8
ShuffleNet 0.25x ($g = 3$)	56.3	**55.0**	1.3
ShuffleNet 0.25x ($g = 8$)	56.5	**52.7**	3.8

Note:数值越小性能越好.

ShuffleNet

- **ShuffleNet 单元**
- ShuffleNet 单元是ShuffleNet模型里的构建模块. 3x3 深度卷积之后的操作替换为1x1 分组卷积，也可以称作分组点式卷积



(a) 带深度卷积的瓶颈结构
(b) 带点式分组卷积和通道混洗的ShuffleNet 单元
(c) 与（b）相同的ShuffleNet单元，但步长为2.

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices."(2018)

ShuffleNet

ShuffleNet 架构

- ShuffleNet 的计算开销在133M 到143M.

Layer	Output size	KSize	Stride	Repeat	Output channels (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	24
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

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices."(2018)

ShuffleNet

ShuffleNet 结果

- 与MobileNet相比，ShuffleNet通过使用点式组卷积（pointwise group convolution）减少了资源消耗和计算开销。
- ShuffleNet还使用通道混洗（channel shuffling）机制来进一步提高性能。

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet 2× ($g = 3$)	524	**26.3**	3.1
ShuffleNet 2× (with *SE*[13], $g = 3$)	527	**24.7**	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet 1.5× ($g = 3$)	292	**28.5**	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet 1× ($g = 8$)	140	**32.4**	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet 0.5× ($g = 4$)	38	**41.6**	7.8
ShuffleNet 0.5× (shallow, $g = 3$)	40	42.8	6.6

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices."(2018)

ShuffleNet

ShuffleNet 结果: 延迟

ShuffleNet 比AlexNet 快 12.10倍，而精确度相当！

Table: 在高通骁龙820 上的推断延迟

Model	Cls err. (%)	FLOPs	224 × 224	480 × 640	720 × 1280
ShuffleNet 0.5× ($g = 3$)	43.2	38M	15.2ms	87.4ms	260.1ms
ShuffleNet 1× ($g = 3$)	32.6	140M	37.8ms	222.2ms	684.5ms
ShuffleNet 2× ($g = 3$)	26.3	524M	108.8ms	617.0ms	1857.6ms
AlexNet [22]	42.8	720M	184.0ms	1156.7ms	3633.9ms
1.0 MobileNet-224 [12]	29.4	569M	110.0ms	612.0ms	1879.2ms

ShuffleNet V2

- ShuffleNet V2 通过分析模型的执行效率，指数了高效网络设计的四条指南。
  - 保持通道数恒定以最小化内存访问成本

- 输入通道数c1 与输出通道数c2 决定了网络形状。特征图的高度与宽度为 h 与 w，则1×1卷积的FLOPs为 B=hwc1c2。
- 假设缓存足够大能够储存下完整的特征图及参数。内存访问成本可以记为 MAC=hw(c1+c2)+c1c2。根据均值不等式可得

$$MAC >= 2\sqrt{hwB} + \frac{B}{hw}$$

MAC的下界由FLOPs决定。当输入与输出通道相等时值最小。

ShuffleNet V2

- ShuffleNet V2 通过分析模型的执行效率，指出高效网络设计的四条指南。
  - 保持通道数恒定以最小化内存访问成本

		GPU (Batches/sec.)				ARM (Images/sec.)		
c1:c2	(c1,c2) for ×1	×1	×2	×4	(c1,c2) for ×1	×1	×2	×4
1:1	(128,128)	1480	723	232	(32,32)	76.2	21.7	5.3
1:2	(90,180)	1296	586	206	(22,44)	72.9	20.5	5.1
1:6	(52,312)	876	489	189	(13,78)	69.1	17.9	4.6
1:12	(36,432)	748	392	163	(9,108)	57.6	15.1	4.4

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices."(2018)

ShuffleNet V2

- ShuffleNet V2 通过分析模型的执行效率，指出高效网络设计的四条指南。
  - 保持通道数恒定以最小化内存访问成本
  - 避免过度的分组卷积

- 1×1卷积的MAC公式可以记为：

$$MAC = hw(c_1 + c_2) + \frac{c_1 c_2}{g} = hwc_1 + \frac{Bg}{c_1} + \frac{B}{hw}$$

g为分组的数量，B是FILOPS， g增大时，MAC增大

ShuffleNet V2

- ShuffleNet V2 通过分析模型的执行效率，指出高效网络设计的四条指南。
  - 保持通道数恒定以最小化内存访问成本
  - 避免过度的分组卷积

		GPU (Batches/sec.)				CPU (Images/sec.)		
g	c for ×1	×1	×2	×4	c for ×1	×1	×2	×4
1	128	2451	1289	437	64	40.0	10.2	2.3
2	180	1725	873	341	90	35.0	9.5	2.2
4	256	1026	644	338	128	32.9	8.7	2.1
8	360	634	445	230	180	27.8	7.5	1.8

ShuffleNet V2

- ShuffleNet V2 通过分析模型的执行效率，指出高效网络设计的四条指南。
  - 保持通道数恒定以最小化内存访问成本
  - 避免过度的分组卷积
  - 谨慎使用碎片化的网络
  - 不可忽视逐元素操作的负面影响

	GPU (Batches/sec.)			CPU (Images/sec.)		
	c=128	c=256	c=512	c=64	c=128	c=256
1-fragment	2446	1274	434	40.2	10.1	2.3
2-fragment-series	1790	909	336	38.6	10.1	2.2
4-fragment-series	752	745	349	38.4	10.1	2.3
2-fragment-parallel	1537	803	320	33.4	9.1	2.2
4-fragment-parallel	691	572	292	35.0	8.4	2.1

ShuffleNet V2

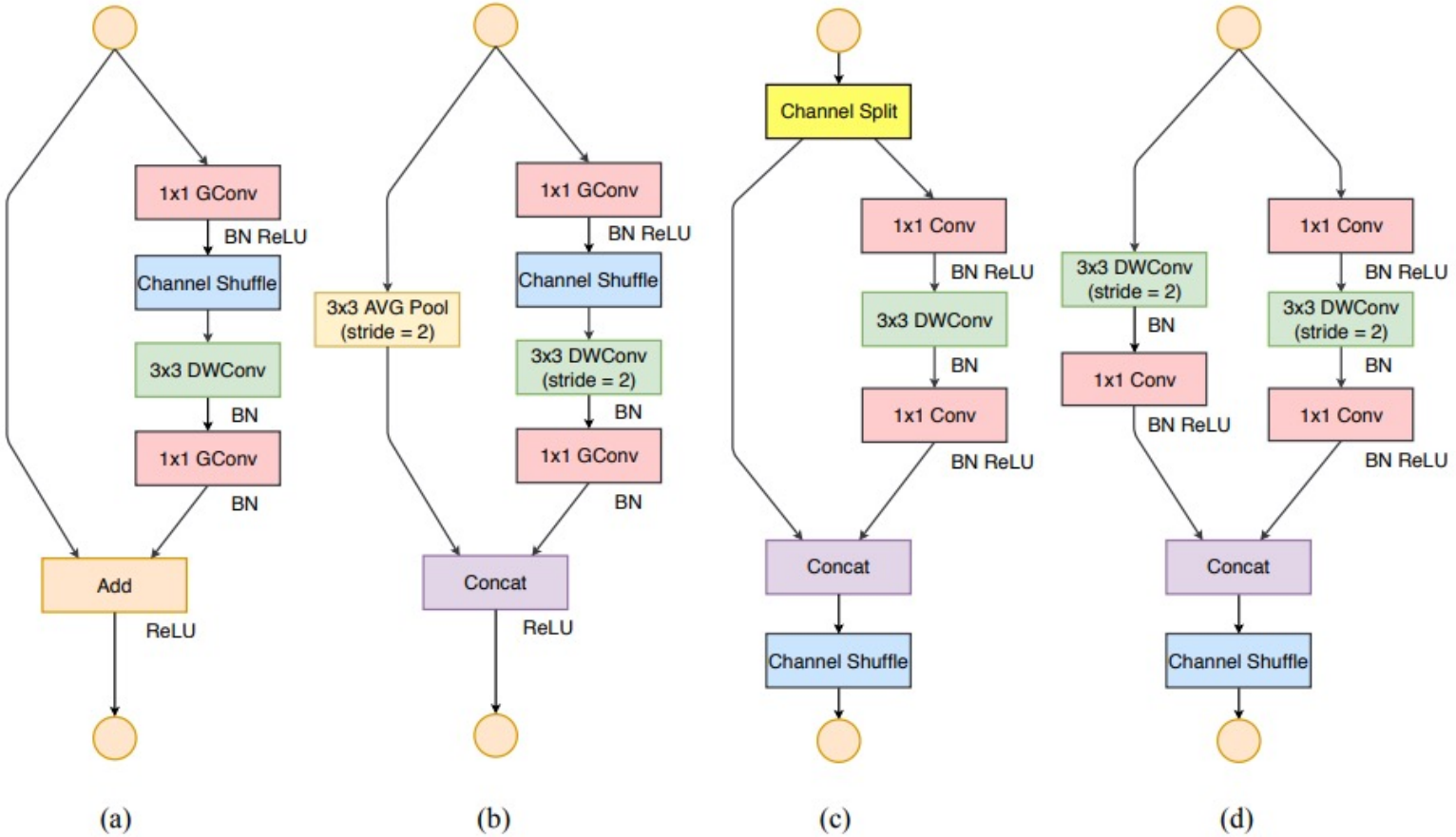
- ShuffleNet V2 通过分析模型的执行效率，指出高效网络设计的四条指南。
  - 保持通道数恒定以最小化内存访问成本
  - 避免过度的分组卷积
  - 谨慎使用碎片化的网络
  - 不可忽视逐元素操作的负面影响

		GPU (Batches/sec.)			CPU (Images/sec.)		
ReLU	short-cut	c=32	c=64	c=128	c=32	c=64	c=128
yes	yes	2427	2066	1436	56.7	16.9	5.0
yes	no	2647	2256	1735	61.9	18.8	5.2
no	yes	2672	2121	1458	57.3	18.2	5.1
no	no	2842	2376	1782	66.3	20.2	5.4

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices."(2018)

ShuffleNet V2

- ShuffleNet V2 提出了通道切分（channel split）。



ShuffleNet V2

- ShuffleNet V2 通过分析模型的执行效率，指出高效网络设计的四条指南。

Layer	Output size	KSize	Stride	Repeat	Output channels			
					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
# of Weights					1.4M	2.3M	3.5M	7.4M

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices."(2018)

ShuffleNet V2

- ShuffleNet V2 相同的FLOPS下，模型准确率更高。

Model	FLOPs	Top-1 err. (%)
ShuffleNet v2-50 (ours)	**2.3G**	**22.8**
ShuffleNet v1-50 [15] (our impl.)	**2.3G**	25.2
ResNet-50 [4]	3.8G	24.0
SE-ShuffleNet v2-164 (ours, with residual)	**12.7G**	**18.56**
SENet [8]	20.7G	18.68

Pytorch 中ShuffleNet V2

```
#ShuffleNet V2
import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'shufflenet_v2_x1_0', pretrained=True)
model.eval()
```

```
# Download an example image from the pytorch website
import urllib
url, filename = ("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
try: urllib.URLopener().retrieve(url, filename)
except: urllib.request.urlretrieve(url, filename)
```

```
# sample execution (requires torchvision)
from PIL import Image
from torchvision import transforms
input_image = Image.open(filename)
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as expected by the model
```

Pytorch 中ShuffleNet V2

```
with torch.no_grad():
    output = model(input_batch)
# Tensor of shape 1000, with confidence scores over Imagenet's 1000 classes
print(output[0])
# The output has unnormalized scores. To get probabilities, you can run a softmax on it.
probabilities = torch.nn.functional.softmax(output[0], dim=0)
print(probabilities)
```

```
# Download ImageNet labels
!wget https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.txt
```

```
# Read the categories
with open("imagenet_classes.txt", "r") as f:
    categories = [s.strip() for s in f.readlines()]
# Show top categories per image
top5_prob, top5_catid = torch.topk(probabilities, 5)
for i in range(top5_prob.size(0)):
    print(categories[top5_catid[i]], top5_prob[i].item())
```

```
Samoyed 0.9923681616783142
Pomeranian 0.0017292629927396774
Eskimo dog 0.0010772038949653506
Siberian husky 0.0008454254711978137
Great Pyrenees 0.0008299825130961835
```

这节课，我们学习了

- 我们为什么需要轻量级网络架构
  - 边缘端设备有限的计算资源
  - 更快的计算速度
- 轻量级架构
  - SqueezeNet (参数量大幅降低)
  - ShuffleNet (分组卷积+通道混洗)
  - MobileNetV1 (深度可分卷积)
  - MobileNetV2 (翻转瓶颈Inverted bottleneck)