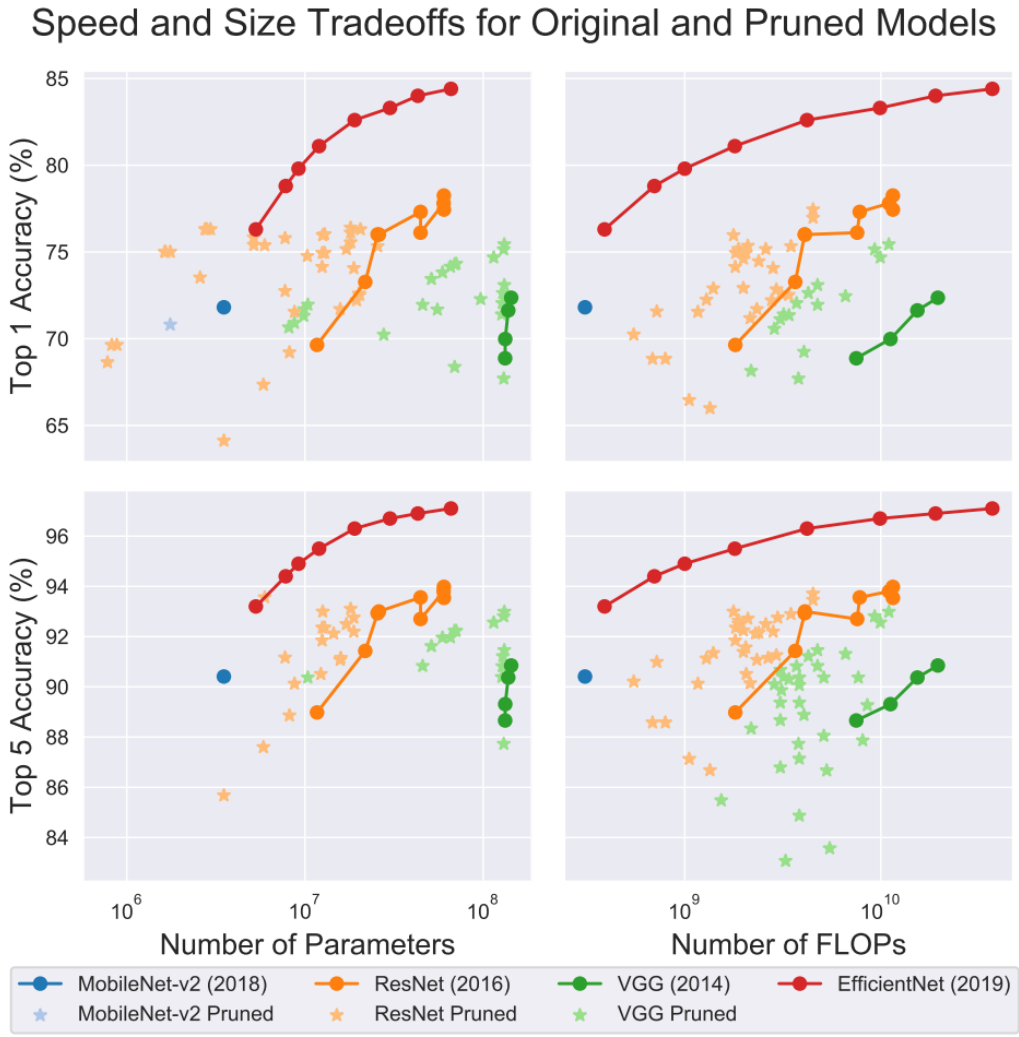
LOW COMPLEXITY MODEL OR COMPACT MODEL FROM SCRATCH

Outline

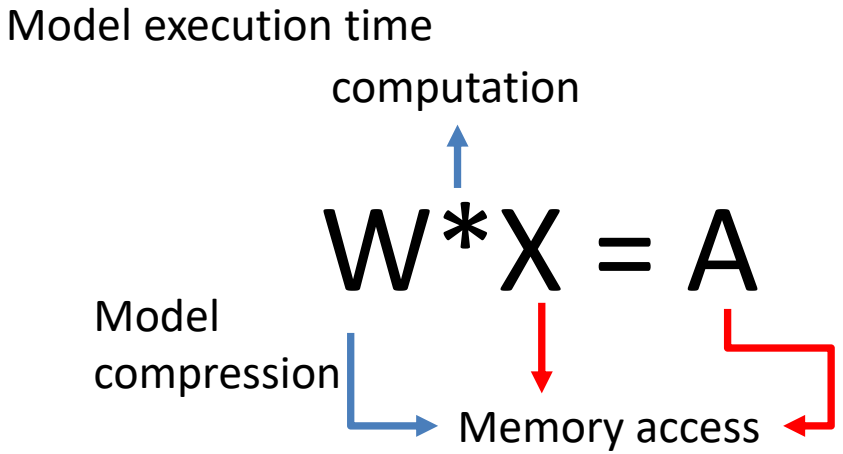
- Overview
- Channel partition
- Model decomposition and memory access optimization
- tinyML on MCU
- Knowledge distillation

Pruning versus advanced network design methods

- Efficient network design is very effective
  - Better than pruning
  - But small model is hard to train



From Model Compression to Acceleration



- 執行時間 = 計算時間 + 資料搬移時間
- 計算時間 = 輸入 * model 大小 / 硬體平台平行度
- 資料搬移時間 = (model 大小 + 輸出入) /硬體平台輸出入寬度
- Model 大小 = N_{input} * N_{output} * (K*K) * Bits
  - (Input channel number * output channel number * filter size * bit number)

Concept: Decomposition and Memory Access

- Execution time
- 執行時間 = 計算時間 + 資料搬移時間
- 計算時間 = convolution + other elementwise operation
- convolution計算時間
  - 輸入影像大小* filter size *(Input channel * output channel)
  - H*W*(K*K) * C_{input} * C_{output}

Decomposition

- Standard convolution: $H*W*(K*K) * C_{in} * C_{out}$
- Decomposition
  - KxK => Kx1, 1xK, asymmetric convolution
  - KxKxC => KxKx$[C/g]^2$ * g,
    - g < C_{out}, group convolutions
    - g == C_{out}, depthwise convolution
    - How to fuse groups to avoid accuracy loss
      - 1x1, shuffle=> mobileNet, shuffleNet
      - Inverted residual => mobileNet v2
- Reduce channels
  - Channel reduction: use 1x1 to reduce channel
  - Channel partition
    - Each partition uses different kernel size: SqueezeNet
    - Partial channel skip: CSPNet

$$H*W*(K*K) * C_{input} * C_{output}$$

這些方法可以套用在現有的 convolution layer

- Key: create or reuse gradient diversity as much as possible
  - For better performance with smaller parameters or low FLOPS
- Extremely small model with small channels
  - Standard convolution works best

- How to restore accuracy loss
  - Scale input size/depth/width
  - Mixed kernel size (larger kernel): 3x3, 5x5, 7x7
  - Attention
    - Squeeze-Excitation network
  - Knowledge distillation

Channel Reduction: SqueezeNet

- Compress AlexNet
- Strategy
  - Replace 3x3 filters with 1x1 filters: 9x fewer parameters
    - make the majority of these filters 1x1
  - Decrease the number of input channels to 3x3 filters
    - total quantity of parameters in this layer is (number of input channels) * (number of filters) * (3*3)
  - Downsample late in the network
    - large activation maps (due to delayed downsampling) can lead to higher classification accuracy

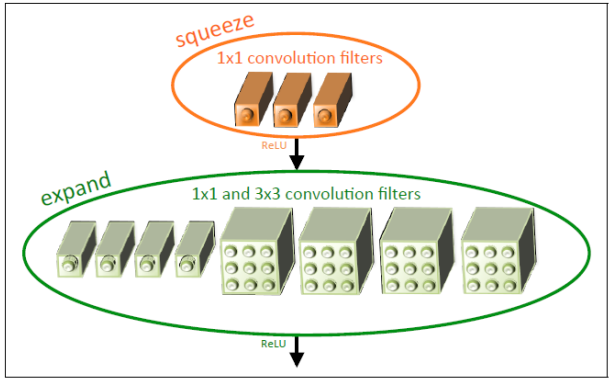
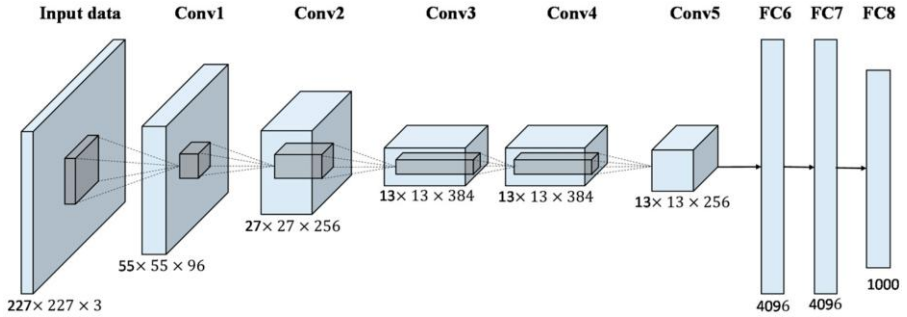
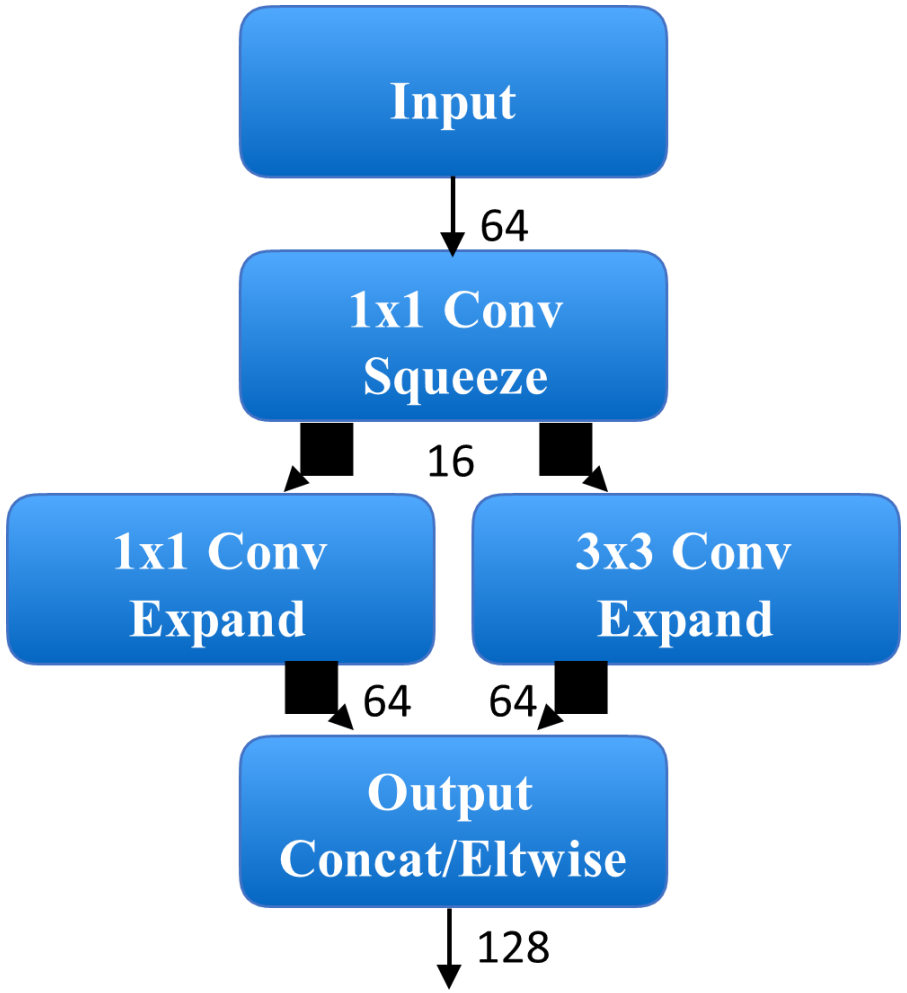
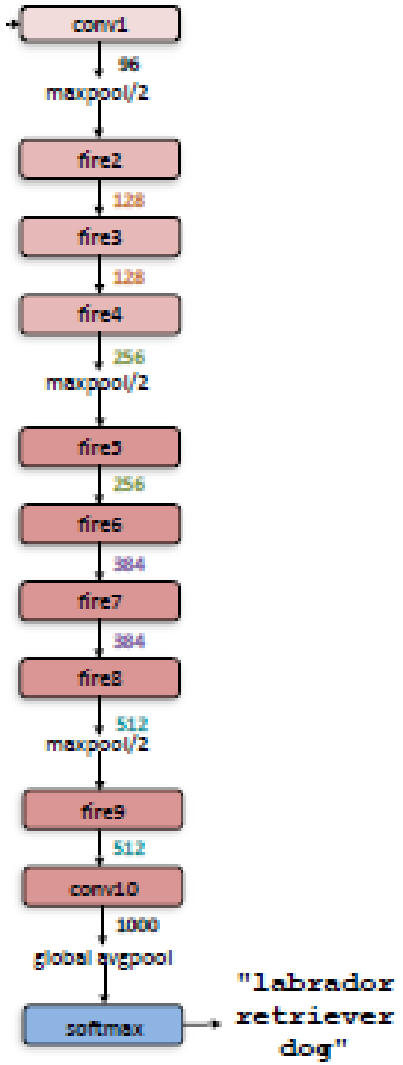


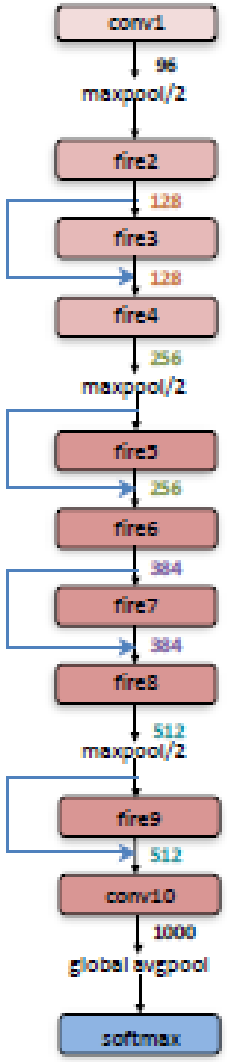
Figure 1. Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

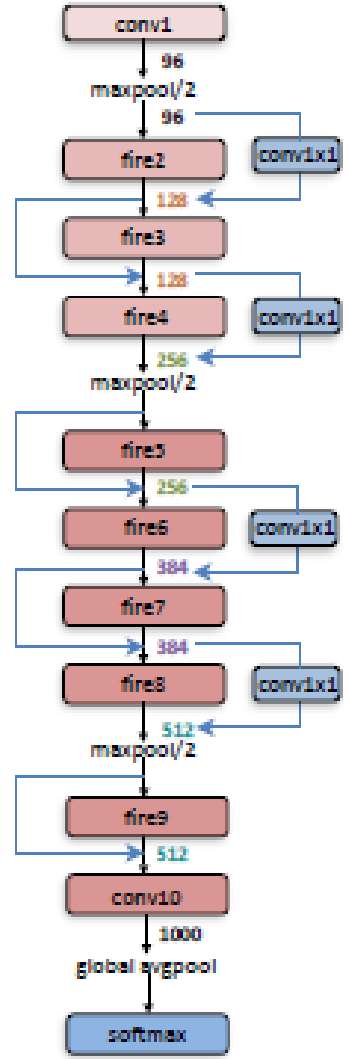


Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv 2016









Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv 2016

Sqeezenet

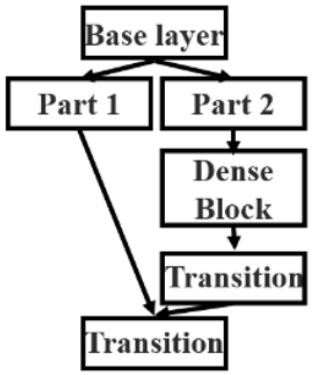
- Good compression, but still needs a lot of computation
  - MAC 861M ~= AlexNet

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%

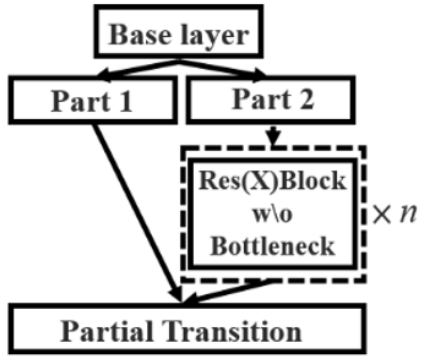
Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", arXiv 2016

Channel Partition: CSPNet in YOLOv4

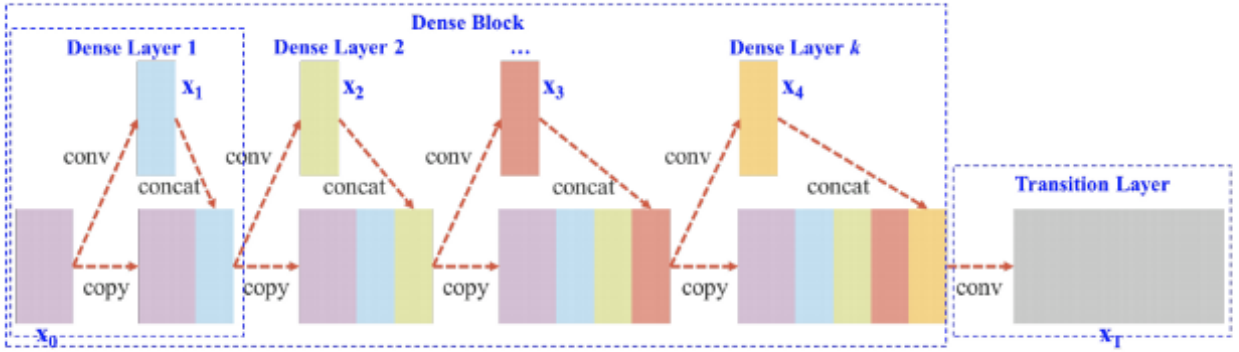
- Cross stage partial DenseNet
  - 增加梯度路徑
  - 每一層的平衡計算
  - 減少記憶體流量



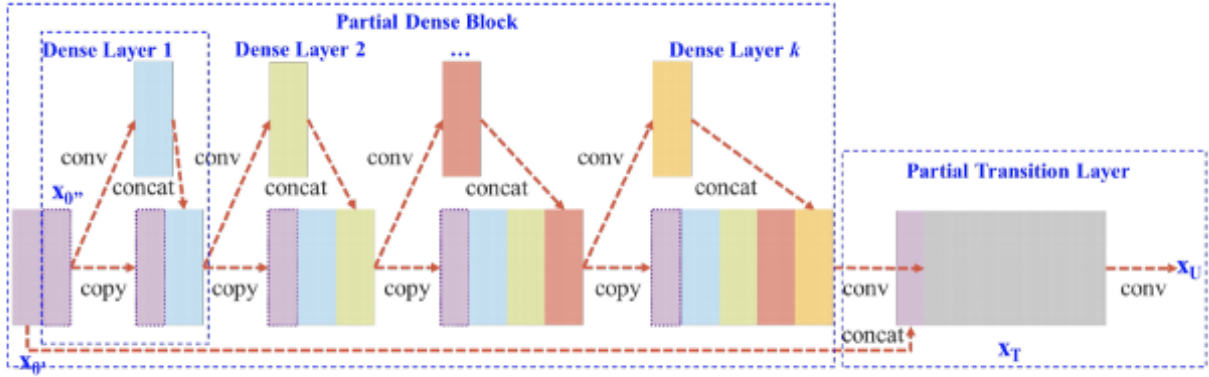
(b) CSPDenseNet



(b) CSPResNe(X)t



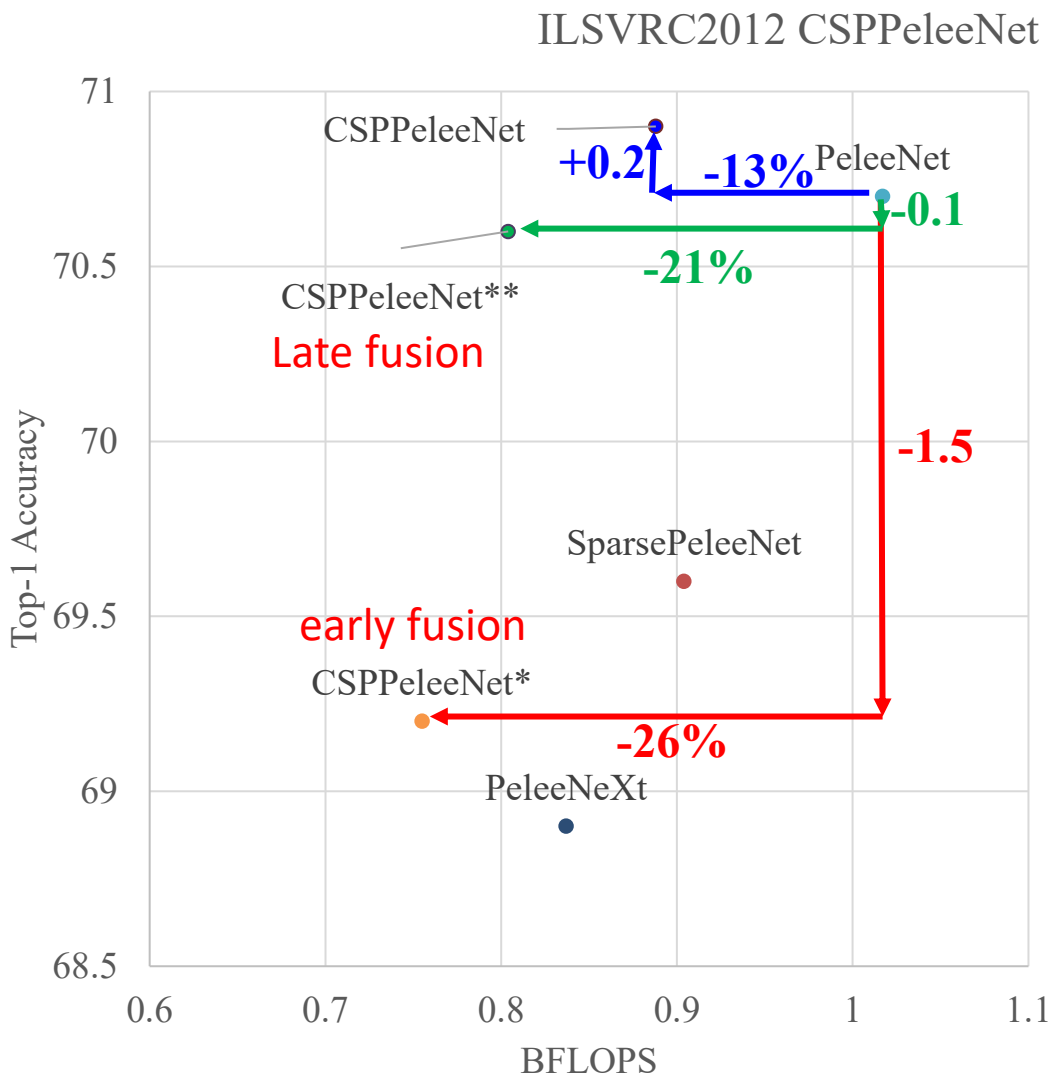
(a) DenseNet

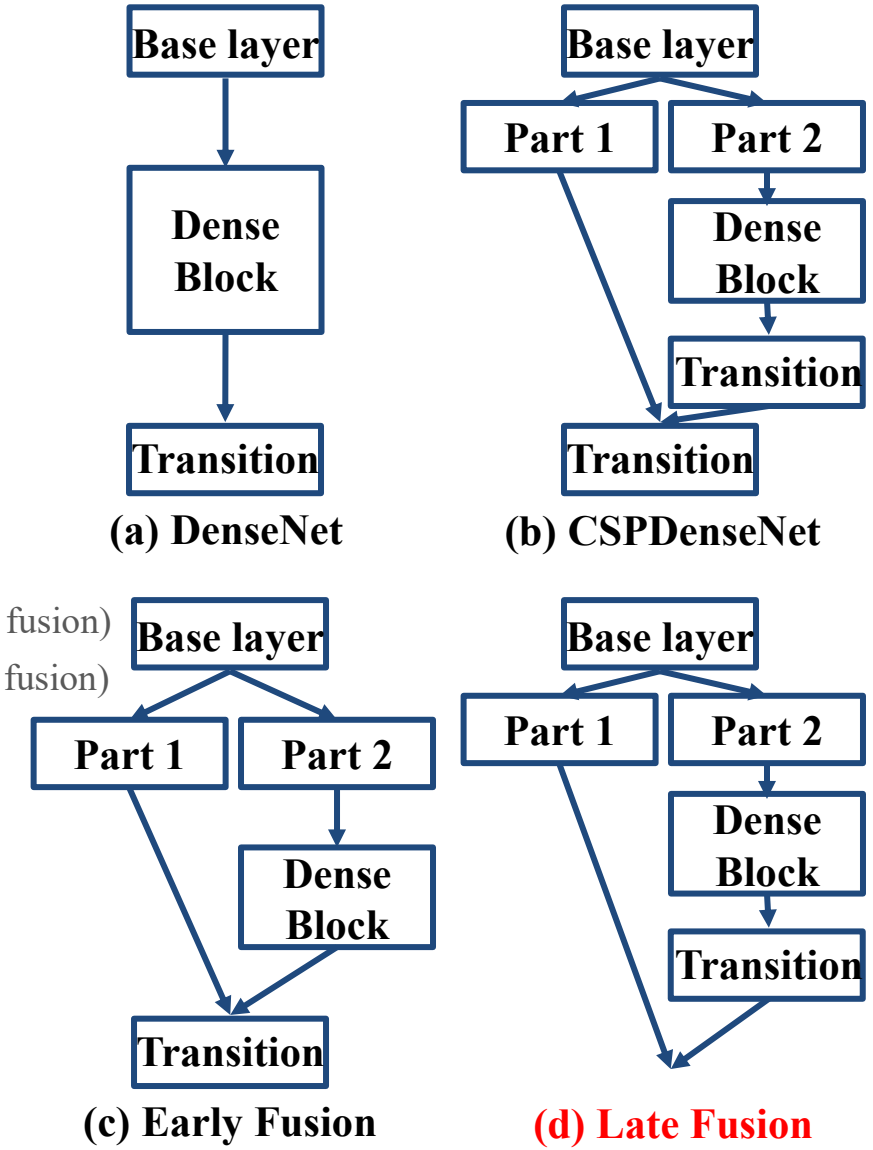


(b) Cross Stage Partial DenseNet

增加學習過程中每個layer的梯度組合

Ablation Study of CSPPeleeNet





Fusion First：會損失大量的梯度資訊。

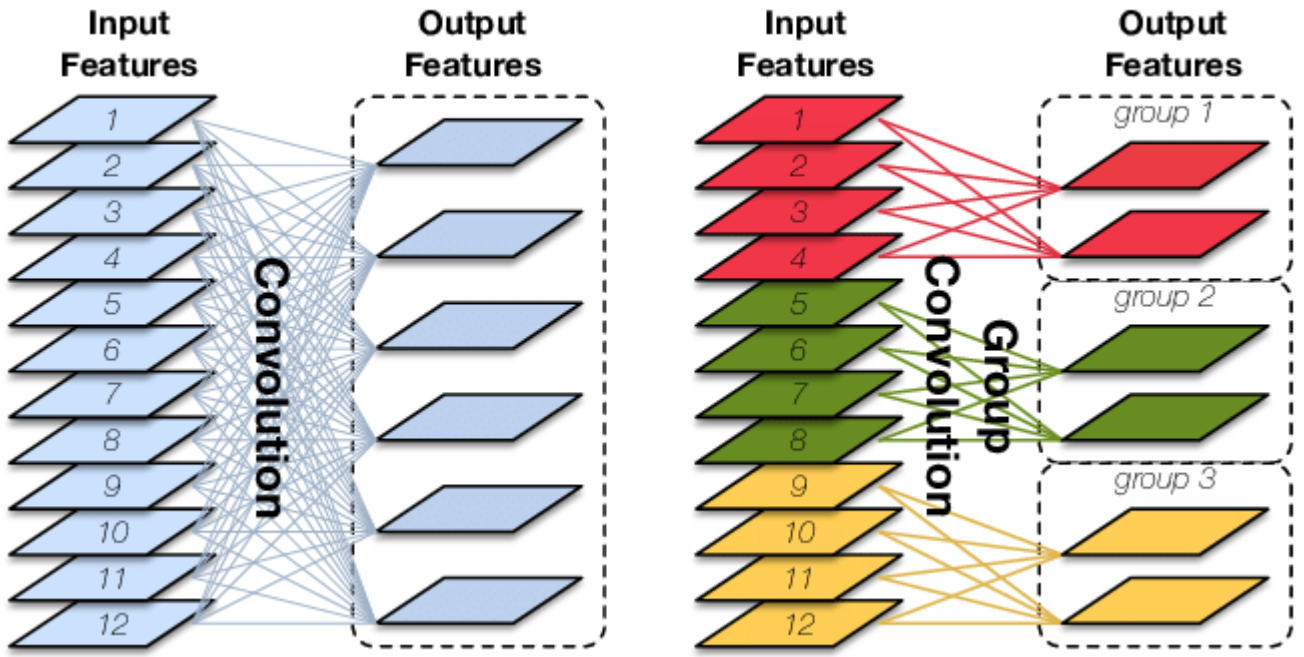
Fusion Last：梯度流被截斷，梯度資訊將不會被重複使用。

late fusion架構在減少21%運算量時，僅降低了0.1%正確率。

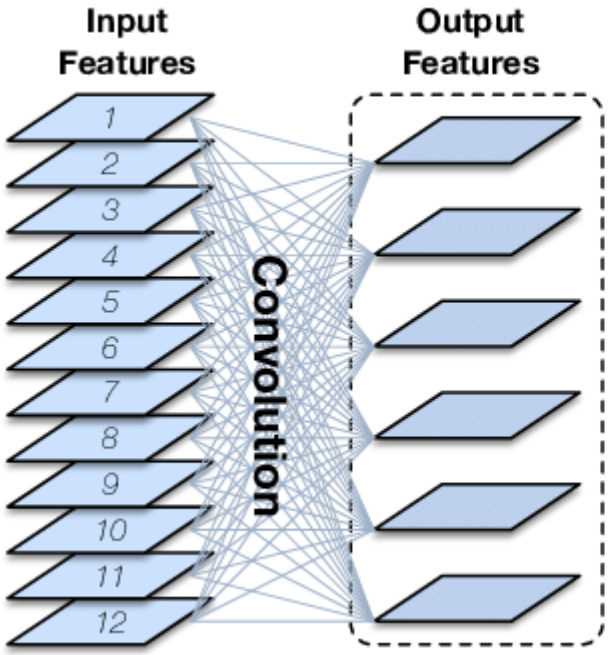
DECOMPOSITION

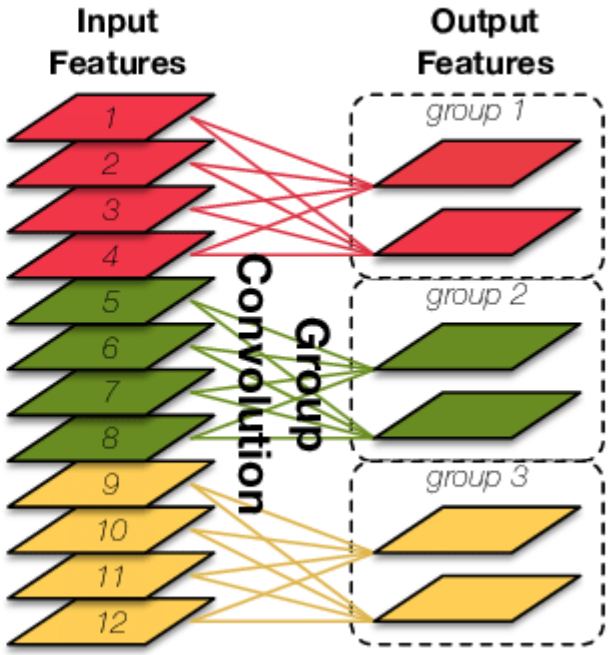
How Group Convolution Saves Parameters

- Standard convolution
  - Parameters number $C_{in}*C_{out}*k*k \sim= C^2*k^2$
- Divide channel into g group
  - Parameter number $(C/g)^2*g*k^2$



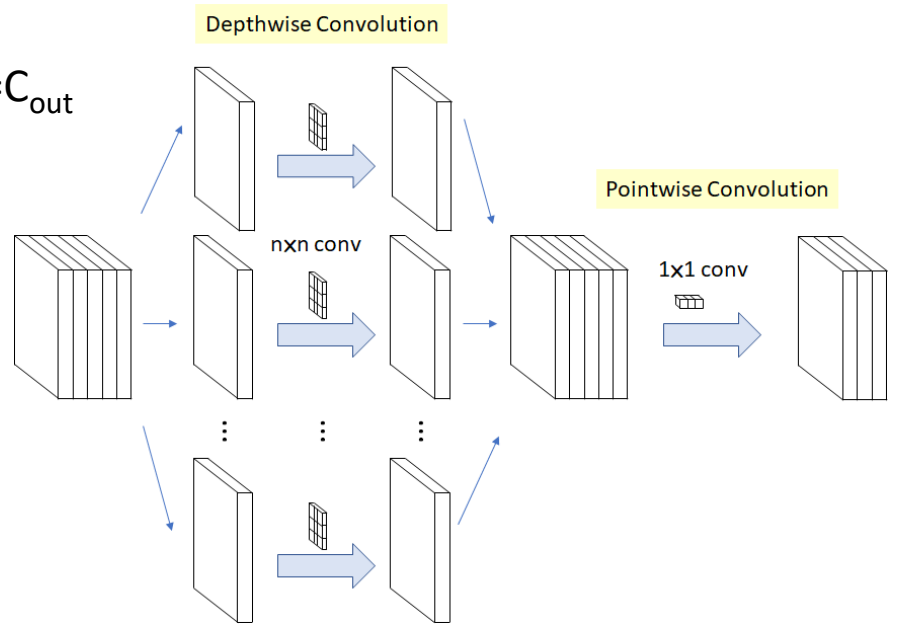
Group Convolution



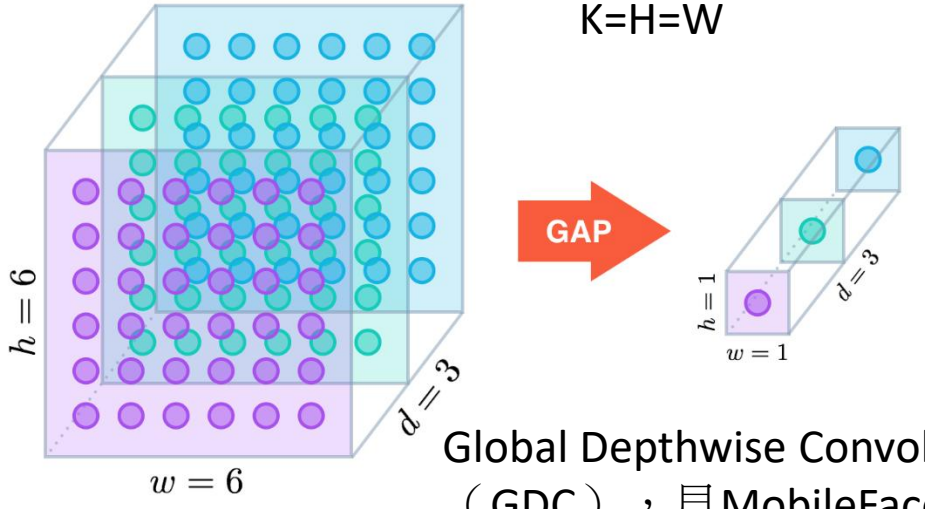


$$\frac{C}{G} * K * K$$

G=C_{out}



K=H=W



Global Depthwise Convolution（GDC），見MobileFaceNet

ResNeXt: ResNet (Group Conv. Version)

128 => 32 groups x 4 channels/group

Grouped convolutions

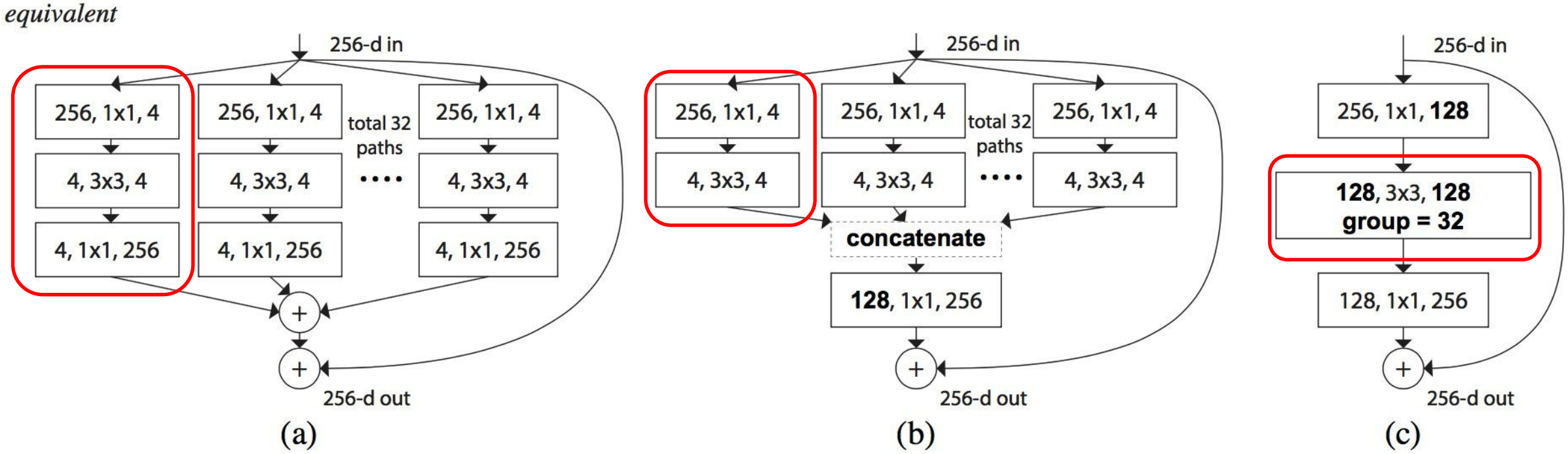


Figure 3. Equivalent building blocks of ResNeXt. **(a)**: Aggregated residual transformations, the same as Fig. 1 right. **(b)**: A block equivale to (a), implemented as early concatenation. **(c)**: A block equivalent to (a,b), implemented as grouped convolutions [23]. Notations in **bo** text highlight the reformulation changes. A layer is denoted as (# input channels, filter size, # output channels).

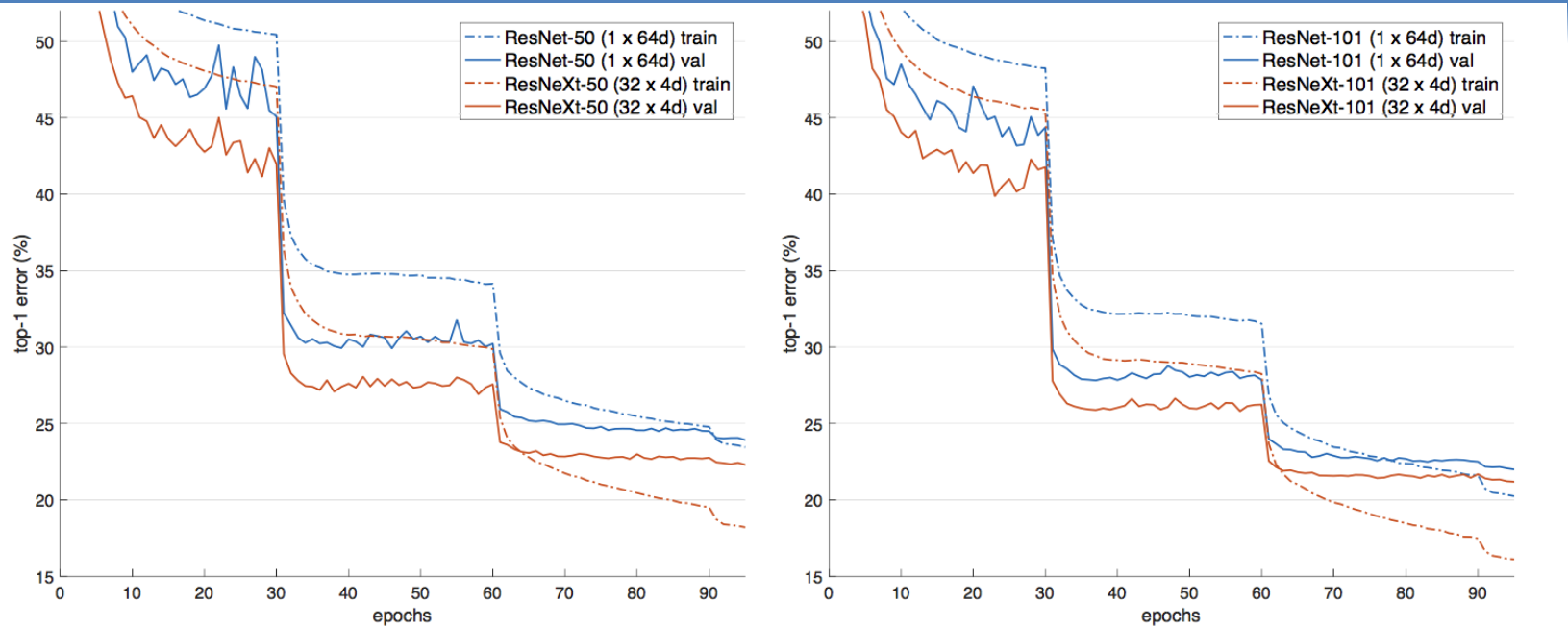


Figure 5. Training curves on ImageNet-1K. (**Left**): ResNet/ResNeXt-50 with preserved complexity (∼4.1 billion FLOPs, ∼25 million parameters); (**Right**): ResNet/ResNeXt-101 with preserved complexity (∼7.8 billion FLOPs, ∼44 million parameters).

	setting	top-1 error (%)
ResNet-50	1 × 64d	23.9
ResNeXt-50	2 × 40d	23.0
ResNeXt-50	4 × 24d	22.6
ResNeXt-50	8 × 14d	22.3
ResNeXt-50	32 × 4d	**22.2**
ResNet-101	1 × 64d	22.0
ResNeXt-101	2 × 40d	21.7
ResNeXt-101	4 × 24d	21.4
ResNeXt-101	8 × 14d	21.3
ResNeXt-101	32 × 4d	**21.2**

Table 3. Ablation experiments on ImageNet-1K. (**Top**): ResNet-50 with preserved complexity (∼4.1 billion FLOPs); (**Bottom**): ResNet-101 with preserved complexity (∼7.8 billion FLOPs). The error rate is evaluated on the single crop of 224×224 pixels.

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-**200** [14]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Table 4. Comparisons on ImageNet-1K when the number of FLOPs is increased to 2× of ResNet-101's. The error rate is evaluated on the single crop of 224×224 pixels. The highlighted factors are the factors that increase complexity.

Same model size
stronger representations

MobileNet

- Standard convolution
  - 3D: 2D + depth
- MobileNet:
  - 2D (spatial mixing) => all depth (channel mixing)
  - Depthwise separable convolution +
  - 1x1 convolution (pointwise convolution)

$$HWNK^2\,(depthwise) + HWNM\,(pointwise) = HWN(K^2 + M)$$

$$\frac{depthwise+pointwise}{conv} = \frac{(K^2 + M)HWN}{K^2MHWN} = \frac{1}{M} + \frac{1}{K^2}$$

For 3x3 convolution, complexity reduction ~1/9

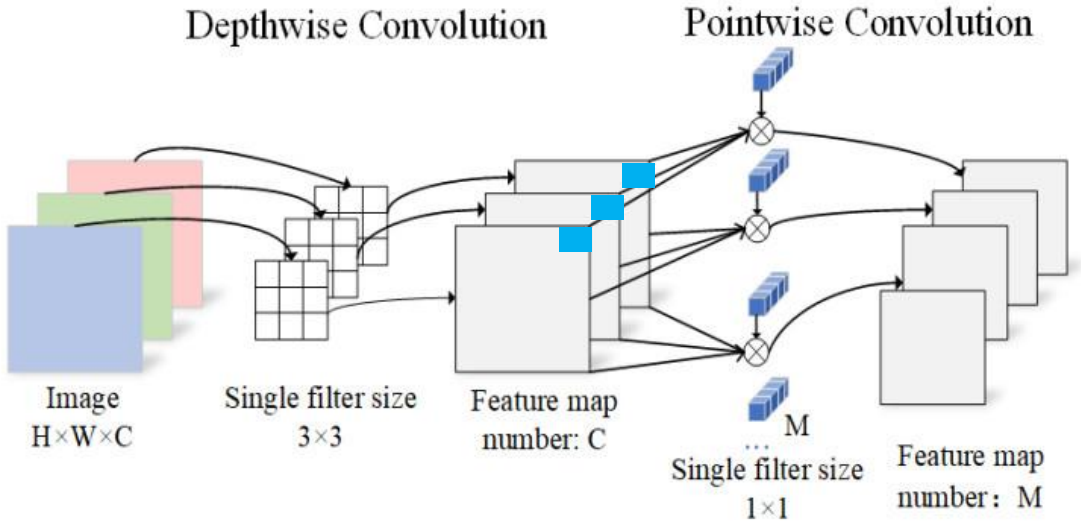


Table 4. Depthwise Separable vs Full Convolution MobileNet

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

~1/7 parameters, ~1/9 MACs

MobileNet

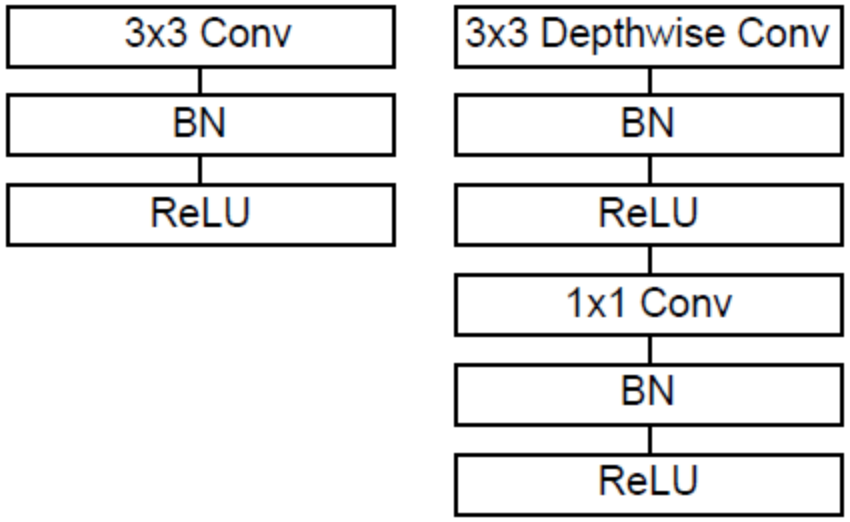


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 2. Resource Per Layer Type

Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

Table 1. MobileNet Body Architecture

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	$1 \times 1 \times 1000$

MobileNet: Results

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Table 13. COCO object detection results comparison using different frameworks and network architectures. mAP is reported with COCO primary challenge metric (AP at IoU=0.50:0.05:0.95)

Framework Resolution	Model	mAP	Billion Mult-Adds	Million Parameters
SSD 300	deeplab-VGG	21.1%	34.9	33.1
	Inception V2	22.0%	3.8	13.7
	MobileNet	19.3%	1.2	6.8
Faster-RCNN 300	VGG	22.9%	64.3	138.5
	Inception V2	15.4%	118.2	13.3
	MobileNet	16.4%	25.2	6.1
Faster-RCNN 600	VGG	25.7%	149.6	138.5
	Inception V2	21.9%	129.6	13.3
	Mobilenet	19.8%	30.5	6.1

Table 14. MobileNet Distilled from FaceNet

Model	1e-4 Accuracy	Million Mult-Adds	Million Parameters
FaceNet [25]	83%	1600	7.5
1.0 MobileNet-160	79.4%	286	4.9
1.0 MobileNet-128	78.3%	185	5.5
0.75 MobileNet-128	75.2%	166	3.4
0.75 MobileNet-128	72.5%	108	3.8

A Quantization-Friendly Separable Convolution for MobileNet

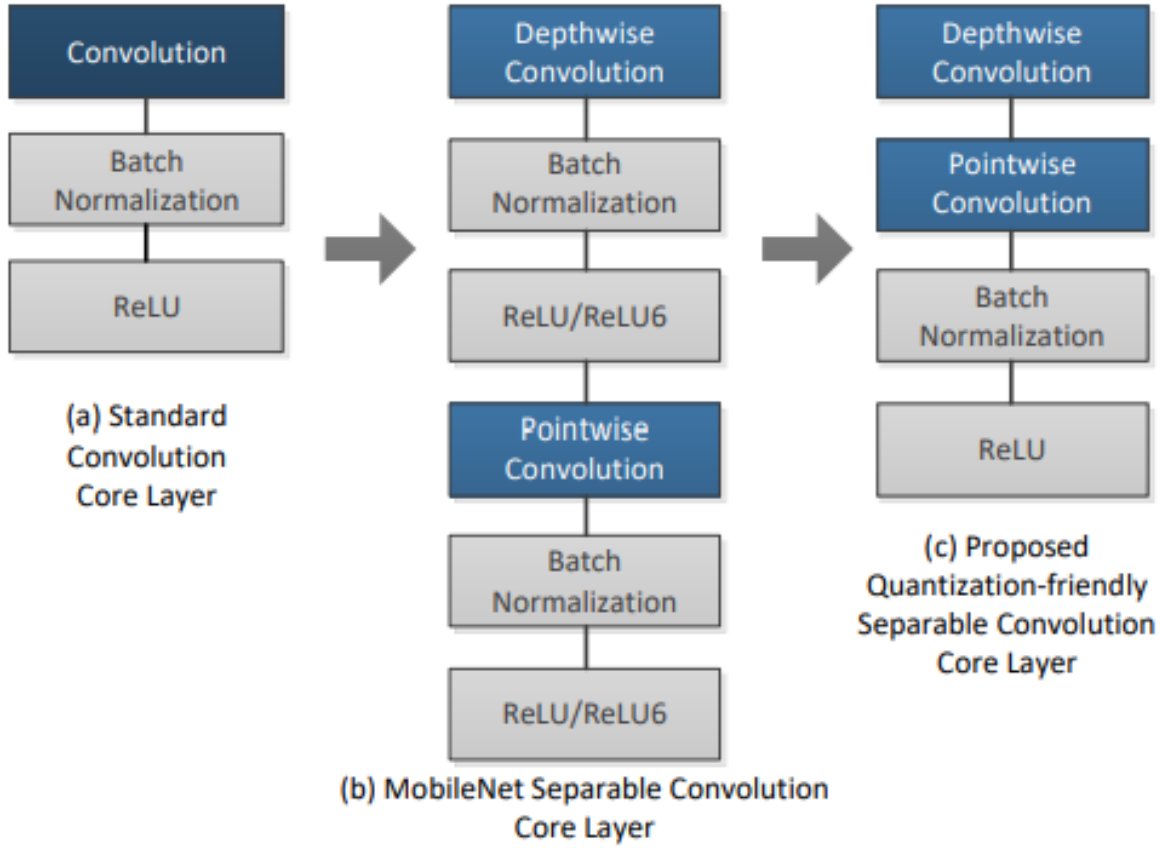


Figure 1. Our proposed quantization-friendly separable convolution core layer design vs. separable convolution in MobileNets and standard convolution

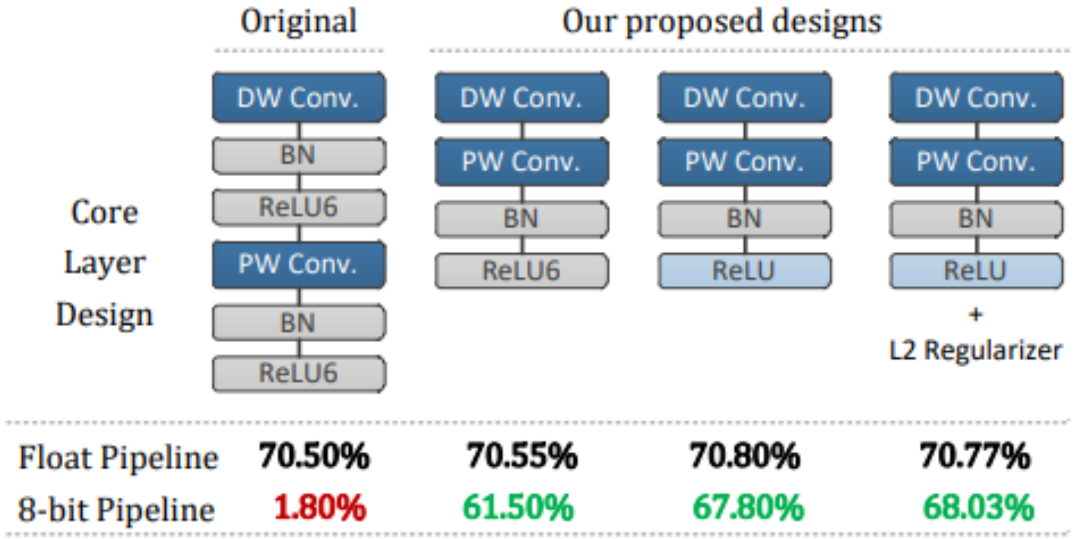


Figure 5. Top-1 accuracy with different core layer designs on ImageNet2012 validation dataset

ShuffleNet

- Problem of mobilenet: expensive pointwise (1x1) convolution

- Concept: channel sparsity
  - 1st Approach: pointwise group convolution
- Side effect: how to get features across group
  - 2nd Approach: channel shuffle

$$H*W*(K*K) * C_{input} * C_{output}$$

ShuffleNet

- divide the channels in each group into several subgroups, then
- feed each group in the next layer with different subgroups

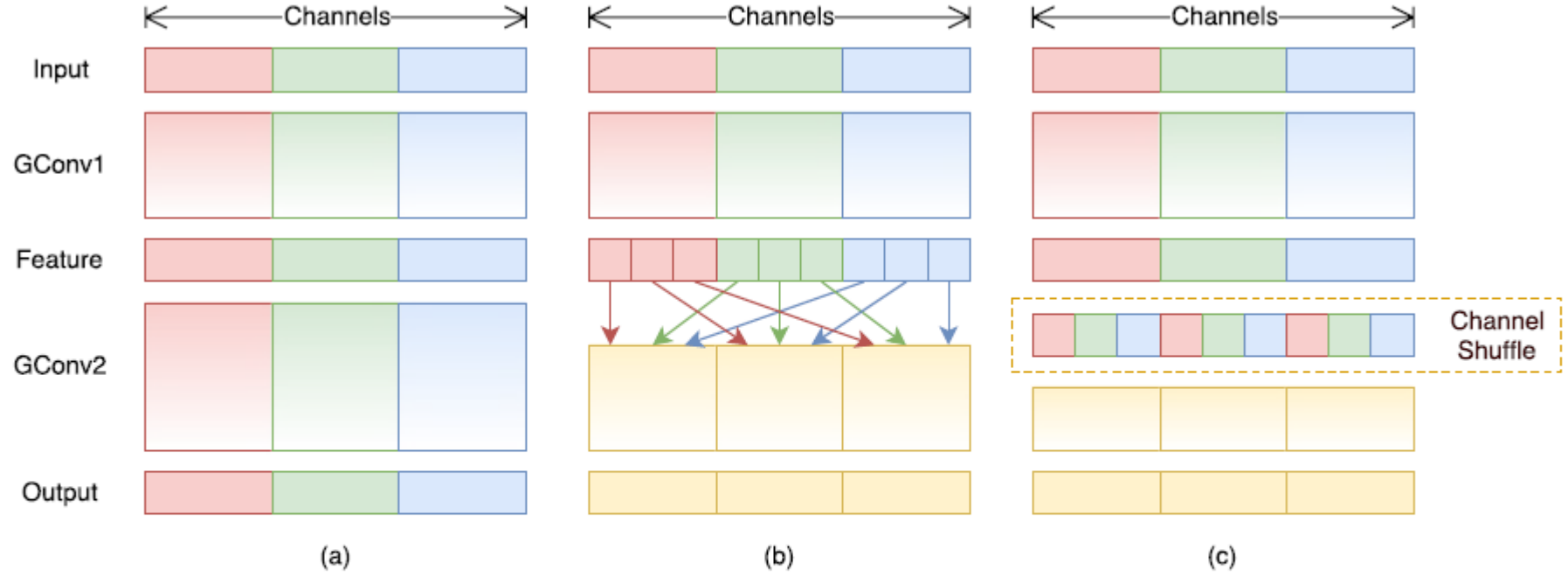
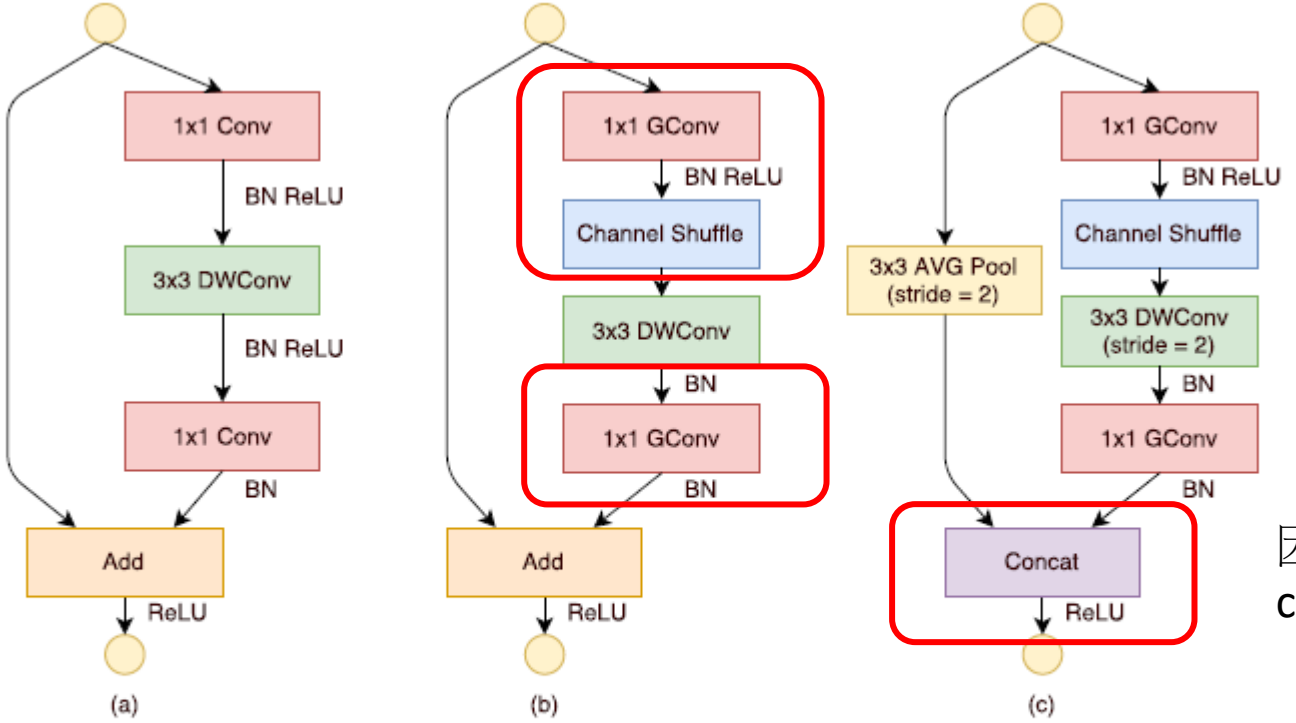


Figure 1: Channel shuffle with two stacked group convolutions. GConv stands for group convolution. a) two stacked convolution layers with the same number of groups. Each output channel only relates to the input channels within the group. No cross talk; b) input and output channels are fully related when GConv2 takes data from different groups after GConv1; c) an equivalent implementation to b) using channel shuffle.

ShuffleNet



因為解析度降低，用concat保留資訊

Figure 2: ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

Channel Shuffle 對現有CPU 不大友好，破壞了數據存儲的連續性

Table 4: Classification error vs. various structures (%, *smaller number represents better performance*)

Complexity (MFLOPs)	VGG-like[4]	ResNet	Xception-like	ResNeXt	ShuffleNet (ours)
140	56.0	38.7	35.1	34.3	**34.1** ($1\times$, $g = 3$)
38	-	48.9	46.1	46.3	**43.7** ($0.5\times$, $g = 4$)
13	-	61.6	56.7	59.2	**53.7** ($0.25\times$, $g = 8$)
40 (arch2)	-	48.5	45.7	47.2	**42.7** ($0.5\times$, $g = 8$)
13 (arch2)	-	61.3	56.5	61.0	**53.3** ($0.25\times$, $g = 8$)

Table 5: ShuffleNet vs. MobileNet [12] on ImageNet Classification

Model	Complexity (MFLOPs)	Cls err. (%)	Δ err. (%)
1.0 MobileNet-224	569	29.4	-
ShuffleNet $2\times$ ($g = 3$)	524	**29.1**	0.3
0.75 MobileNet-224	325	31.6	-
ShuffleNet $1.5\times$ ($g = 3$)	292	**31.0**	0.6
0.5 MobileNet-224	149	36.3	-
ShuffleNet $1\times$ ($g = 3$)	140	**34.1**	2.2
0.25 MobileNet-224	41	49.4	-
ShuffleNet $0.5\times$ (arch2, $g = 8$)	40	**42.7**	6.7
ShuffleNet $0.5\times$ (shallow, $g = 3$)	40	45.2	4.2

ShuffleNet

Table 6: Complexity comparison

Model	Cls err. (%)	Complexity (MFLOPs)
VGG-16 [27]	28.5	15300
ShuffleNet 2× ($g = 3$)	29.1	**524**
PVANET [18] (*our impl.*)	35.3	557
ShuffleNet 1× ($g = 3$)	34.1	**140**
AlexNet [19]	42.8	720
SqueezeNet [13]	42.5	833
ShuffleNet 0.5× (arch2, $g = 8$)	42.7	**40**

Table 7: Object detection results on MS COCO (*larger numbers represents better performance*)

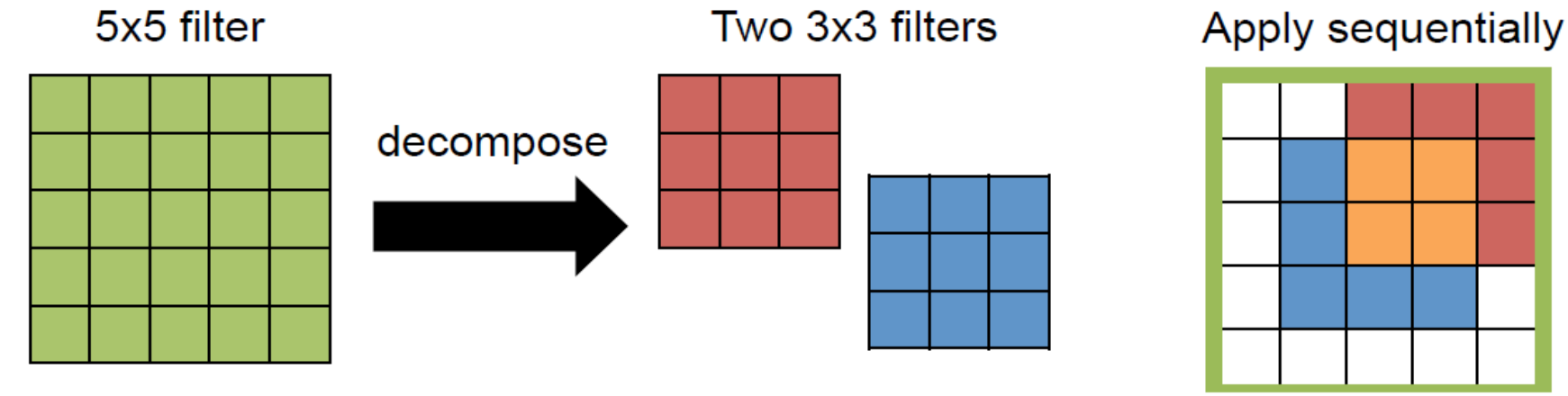
Model	mAP [.5, .95] (300× image)	mAP [.5, .95] (600× image)
ShuffleNet 2× ($g = 3$)	**18.0%**	**24.5%**
ShuffleNet 1× ($g = 3$)	14.3%	19.8%
1.0 MobileNet-224 [12]	16.4%	19.8%

Table 8: Actual inference time on mobile device (*smaller number represents better performance*)

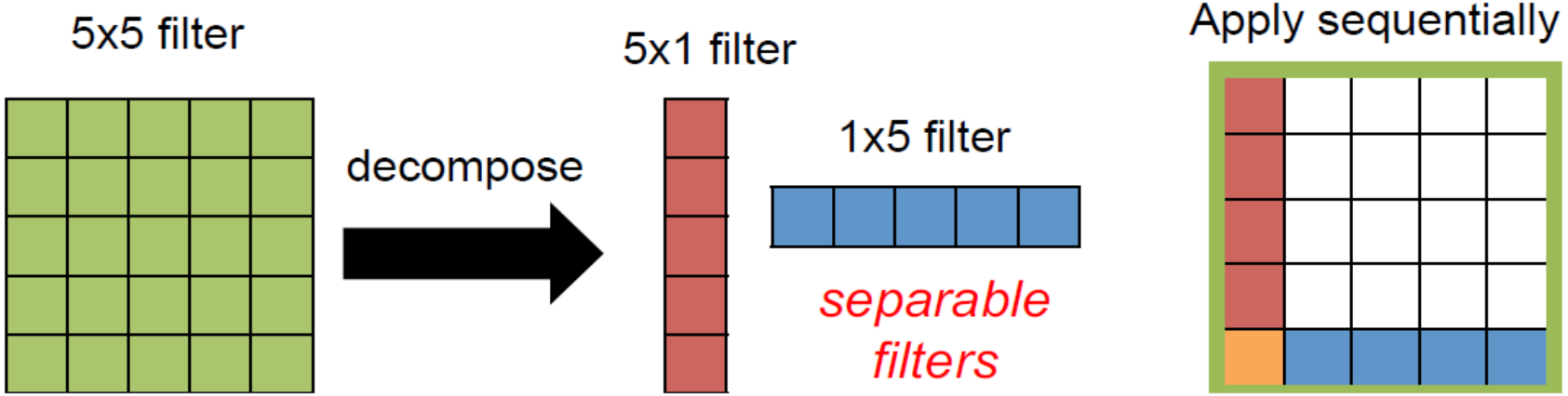
Model	Cls err. (%)	FLOPs	224 × 224	480 × 640	720 × 1280
ShuffleNet 0.5× (arch2, $g = 3$)	43.8	40M	15.2ms	87.4ms	260.1ms
ShuffleNet 1× ($g = 3$)	34.1	140M	37.8ms	222.2ms	684.5ms
AlexNet [19]	42.8	720M	184.0ms	1156.7ms	3633.9ms

Reducing filter sizes: filter decomposition

VGG-16



GoogleNet/Inception v3



asymmetric depthwise separable convolution

$$H*W*(K*K) * C_{input} * C_{output}$$

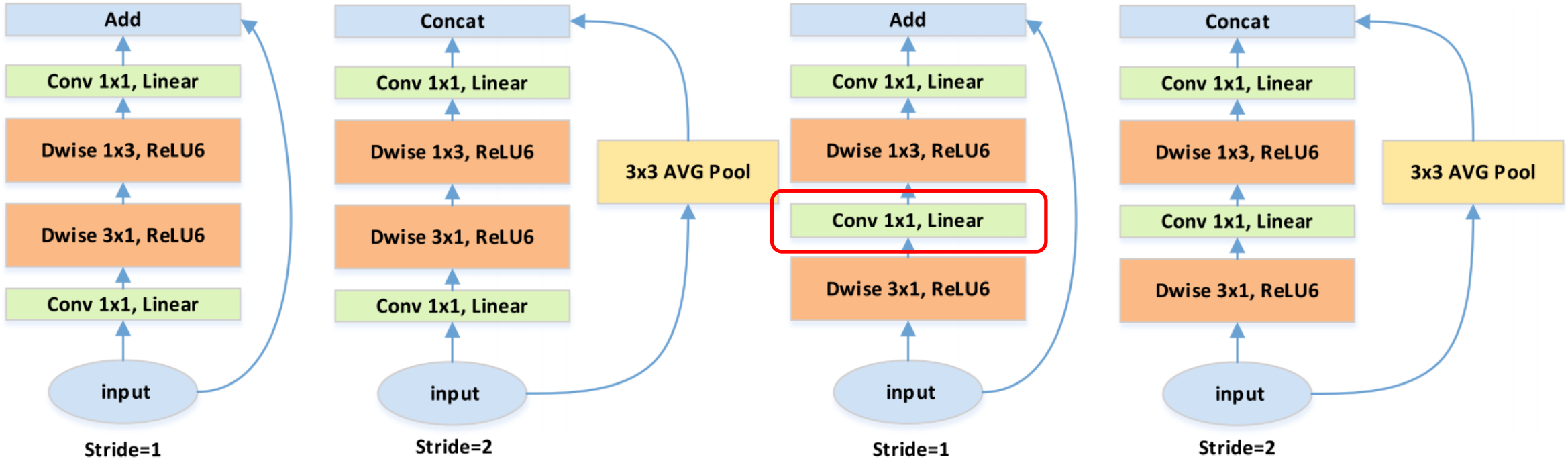


Fig. 5 Before and after reordering in ADSCNet units

Table 2 Comparison between different order of convolutional layers

Model	Complexity (GFLOPs)	mIou(Normal order)	mIou(Reorder)	Δ mIou
ADSCNet 1.0×	8.2	65.4	**67.5**	2.1
ADSCNet 0.75×	6.1	63.0	**65.0**	2.0
ADSCNet 0.5×	4.1	61.3	**62.1**	0.8

Table 3 With/without average pooling before ADSCNet unit in DDCC

Model	Complexity (GFLOPs)	mIou (no average pooling)	mIou (average pooling)	Δ mIou
ADSCNet 1.0×	8.2	64.0	**67.5**	3.5
ADSCNet 0.75×	6.1	63.2	**65.0**	1.8
ADSCNet 0.5×	4.1	60.3	**62.1**	1.8

Table 4 Comparison with different network units

Complexity (GFLOPs)	IGCV2	ShuffleNet	MobileNet	MobileNetV2	ADSCNet
8.2	55.7	57.6	62.3	64.2	**67.5**
6.1	55.3	56.2	61.9	63.0	**65.0**
4.1	54.9	55.1	58.0	60.3	**62.1**

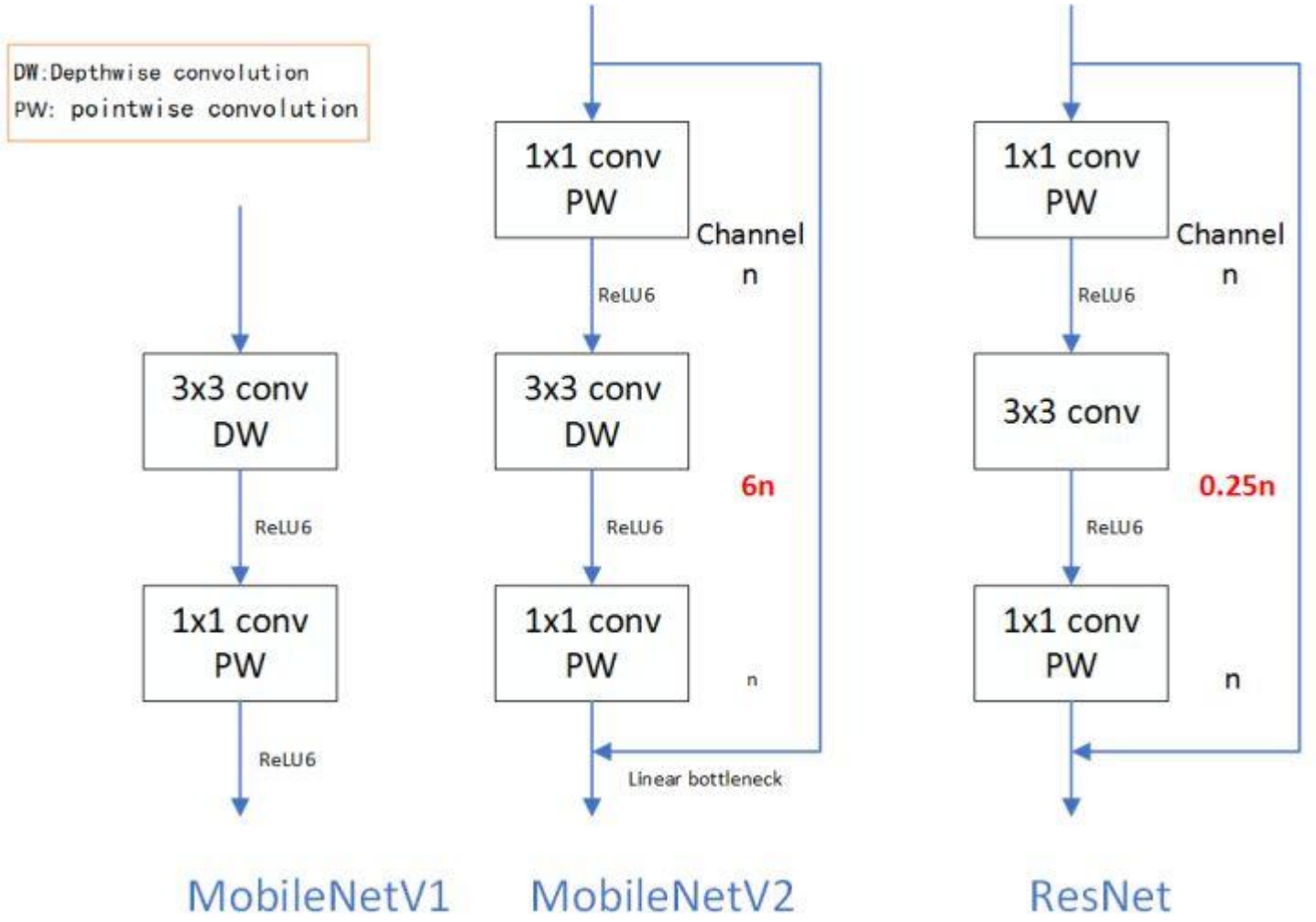
Problems of MobileNet v1 & Why MobileNet v2

MobileNet v1

- Structure
  - Structure is too simple like VGG, low c/p ratio (performance v.s. parameters)
  - Move to better one like ResNet, DenseNet
- Potential problems of depth wise convolution
  - Easy to be degenerated due to smaller kernel dimension and ReLU

MobileNet v2

- Inverted residual
  - Short connection + wide middle layer
- Linear bottleneck
  - Reduce effect of ReLU (ReLU is not good for depthwise convolution)



ResNet是：壓縮”→“卷積提特徵”→“擴張”，
MobileNetV2則是Inverted residuals,即：“擴張”→“卷積提特徵”→“壓縮”

- Depth-wise convolution之前多了一個1*1的“擴張”層，目的是為了提升通道數，獲得更多特徵；
- 最後不採用Relu，而是Linear，目的是防止Relu破壞特徵。

MobileNet v2

- Inverted residual

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	-	

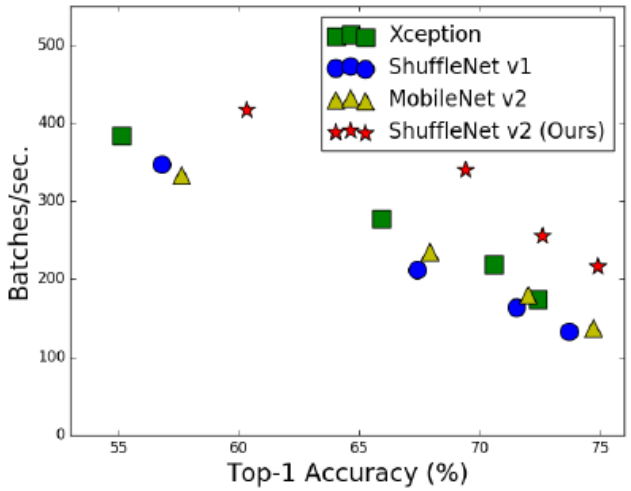
除了最後的avgpool，整個網路並沒有採用pooling進行下采樣，而是利用stride=2來下采樣

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**

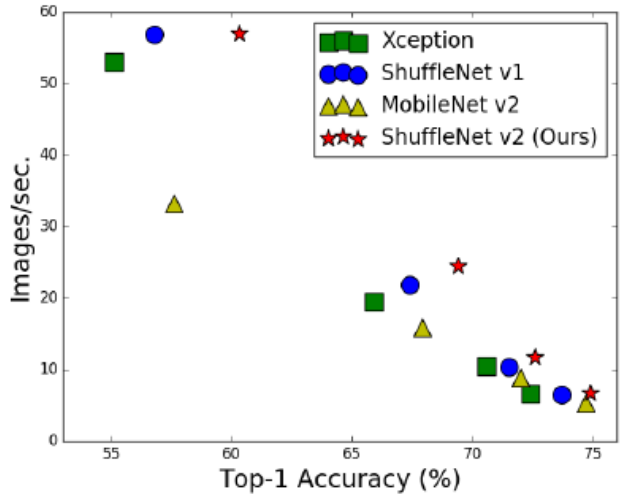
Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

ShuffleNet v2

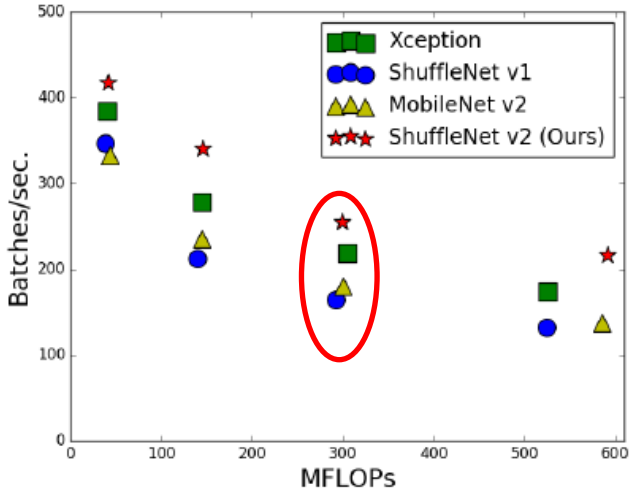
- accuracy, speed and FLOPs
- Metrics for complexity
  - Indirect: FLOPS
  - Direct: execution time/ speed/ latency
  - Same flops but different speed



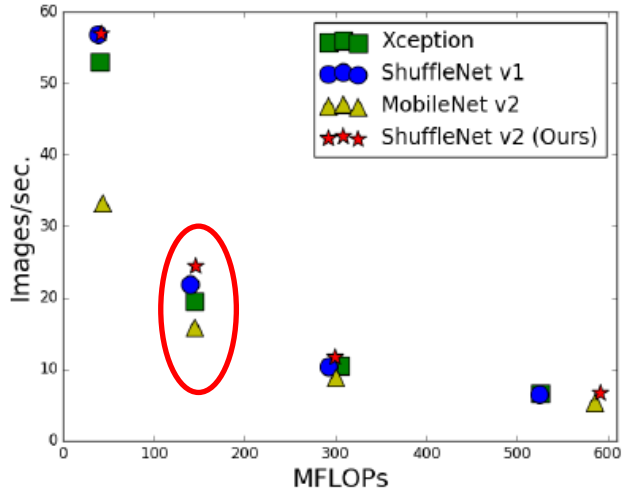
(a) GPU



(b) ARM



(c) GPU



(d) ARM

- Metrics for complexity
  - Indirect: FLOPS
  - Direct: execution time/ speed/ latency
  - Same flops different speed
    - CUDNN specially optimized for 3x3
    - We cannot certainly think that 3x3 conv is 9 times slower than 1x1 conv
- What missed
  - Memory access cost
    - Significant part in low complexity layer: Group convolution
    - Bottleneck for strong computing devices, eg. GPU
  - Degree of parallelism
    - Layers with high parallelism run faster than one with low parallelism

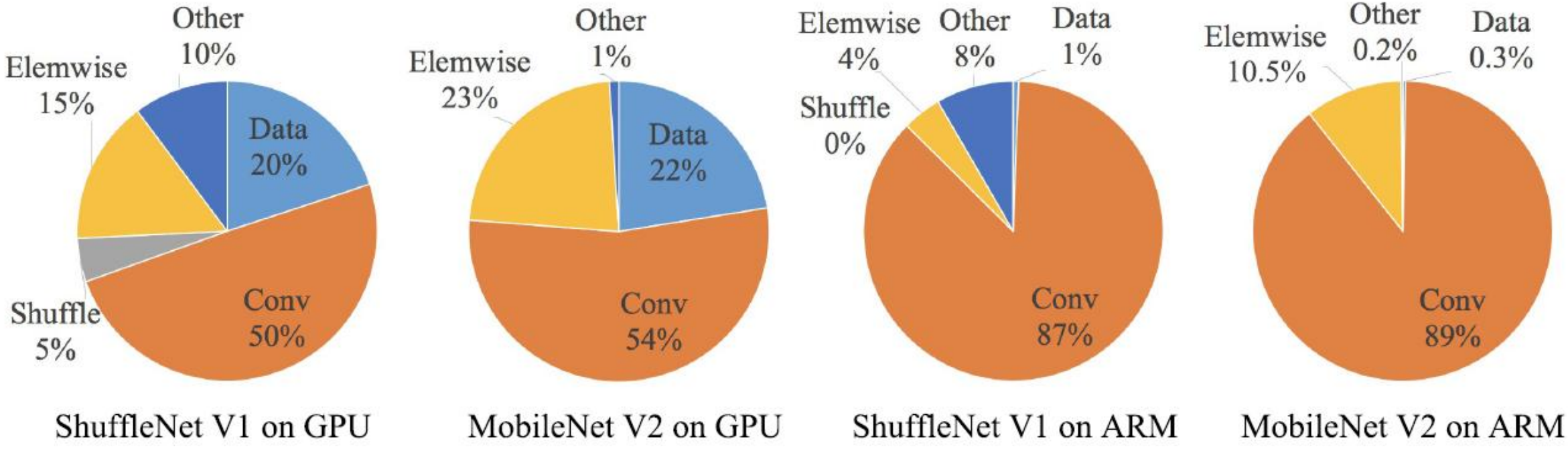


Fig. 2: Run time decomposition on two representative state-of-the-art network architectures, *ShuffeNet v1* [15] ($1\times$, $g = 3$) and *MobileNet v2* [14] ($1\times$).

Four Guideline for Network Designs

- G1: Equal channel width minimizes memory access cost
  - Assume depthwise conv + 1x1
  - Focus on channel no. of 1x1 convolution (account for most of the complexity)

		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

Table 1: Validation experiment for **Guideline 1**. Four different ratios of number of input/output channels ($c1$ and $c2$) are tested, while the total FLOPs under the four ratios is fixed by varying the number of channels. Input image size is 56×56.

1x1 輸入和輸出特徵通道數相等時MAC最小

- FLOPS
  - 1x1, B = hwc1c2
- MAC (memory access cost)
  - MAC = feature map + kernel = hw(c1+c2) + c1c2
- By mean value theorem

$$\mathrm{MAC} \geq 2\sqrt{hwB} + \frac{B}{hw}.$$

  - Substitue MAC and B to above equation, get (c1-c2) ^{2} >=0
    - C1 = C2 is the lower bound
  - C1: c2 = 1: 1 has the lowest execution time

- G2) Excessive group convolution increases MAC (memory access).

$$\begin{aligned}\text{MAC} &= hw(c_1 + c_2) + \frac{c_1 c_2}{g} \\ &= hwc_1 + \frac{Bg}{c_1} + \frac{B}{hw},\end{aligned}$$

- MAC increases with the growth of g if others fixed

- C = c1 + c2
- Same flops

		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

Table 2: Validation experiment for **Guideline 2**. Four values of group number g are tested, while the total FLOPs under the four values is fixed by varying the total channel number c. Input image size is 56×56.

過多的group操作會增大Memory access，從而使模型速度變慢

- G3) Network fragmentation reduces degree of parallelism.

	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

Table 3: Validation experiment for **Guideline 3**. *c* denotes the number of channels for *1-fragment*. The channel number in other fragmented structures is adjusted so that the FLOPs is the same as *1-fragment*. Input image size is 56×56.

模型支路越多（fragment程度越高）對於並行計算越不利

- G4) Element-wise operations are non-negligible (e.g. shortcut)
  - Depthwise conv is also regarded as element wise operation (low FLOPs but high MAC).

		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

Table 4: Validation experiment for **Guideline 4**. The ReLU and shortcut operations are removed from the "bottleneck" unit [4], separately. c is the number of channels in unit. The unit is stacked repeatedly for 10 times to benchmark the speed.

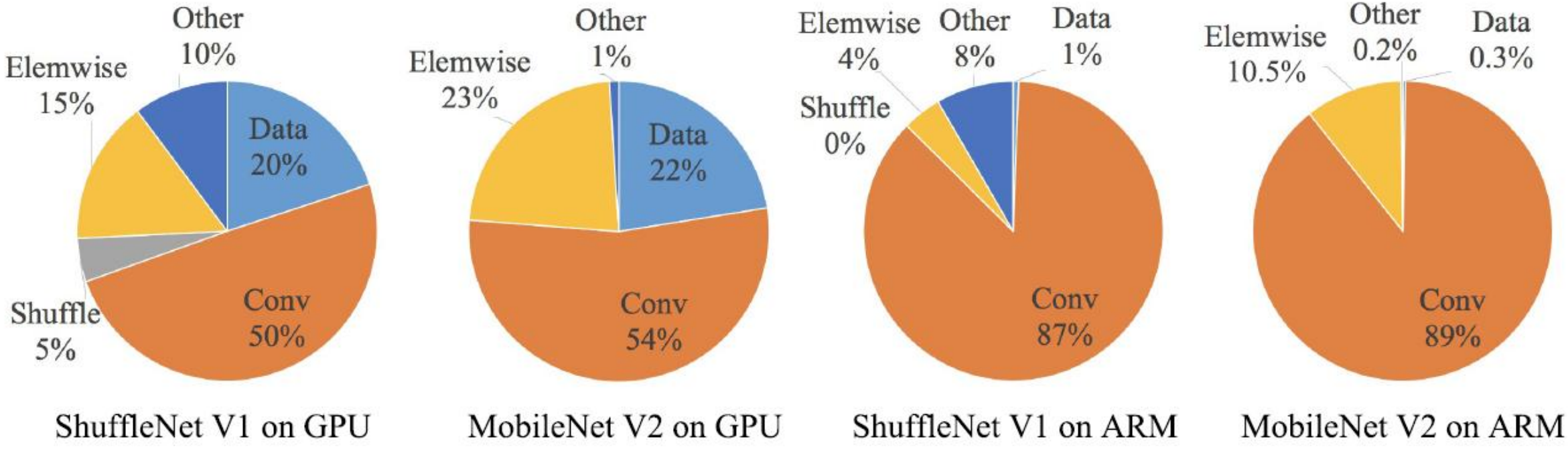
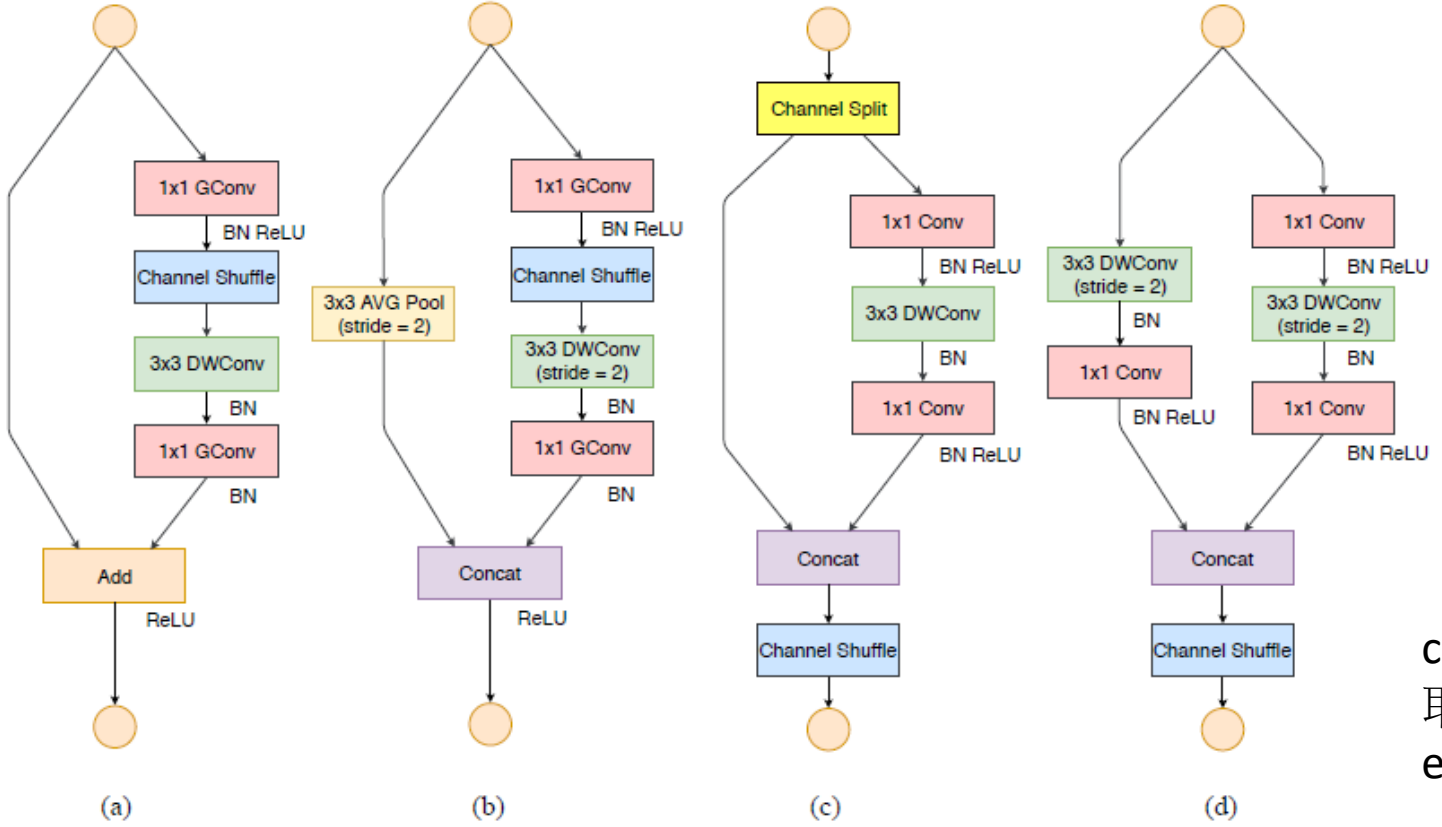


Fig. 2: Run time decomposition on two representative state-of-the-art network architectures, *ShuffeNet v1* [15] ($1\times$, $g = 3$) and *MobileNet v2* [14] ($1\times$).

Some discussions

- Shuffnet v1
  - Heavily depends on group convolutions (against G2)
  - And bottleneck like blocks (against G1)
- MobileNet v2
  - Inverted bottleneck (against G1)
  - Depthwise and ReLUs on thick feature maps (against G4)
- Nasnet
  - Fragmented structure (against G3)

ShuffleNet v2



channel split
取消了1*1卷積層中的group操作
element-wise add操作替換成concat

Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling (2×); (c) our basic unit; (d) our unit for spatial down sampling (2×). **DWConv**: depthwise convolution. **GConv**: group convolution.

MobileNetV3

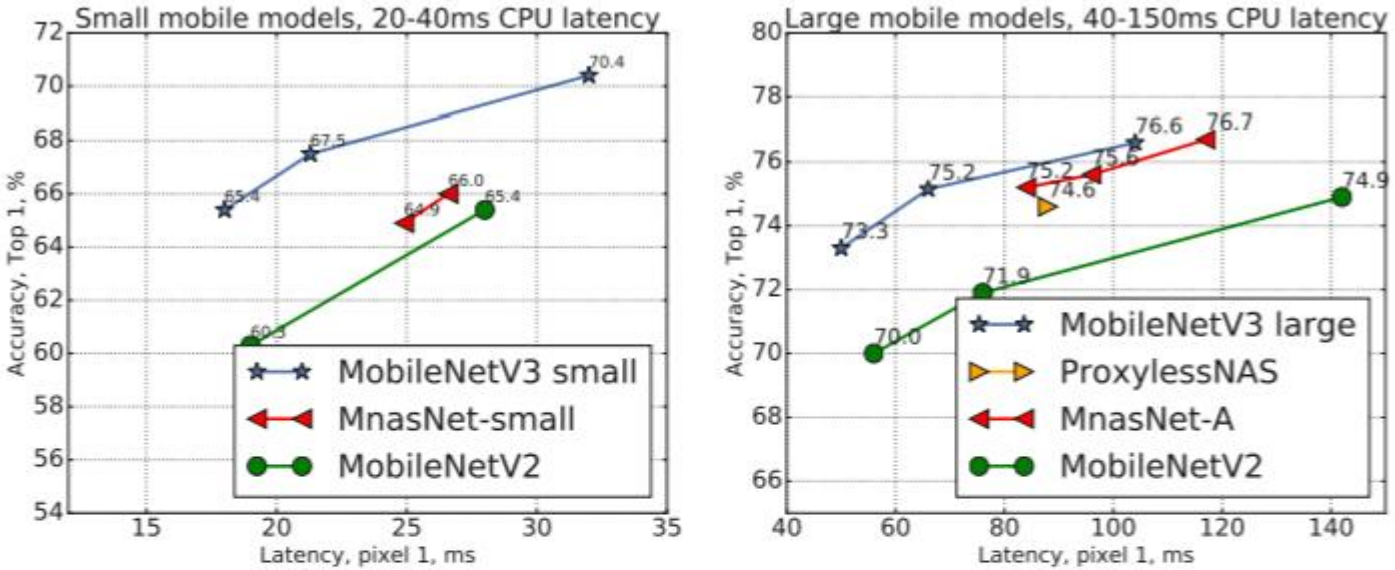


Figure 1. The trade-off between Pixel 1 latency and top-1 ImageNet accuracy. All models use the input resolution 224. V3 large and V3 small use multipliers 0.75, 1 and 1.25 to show optimal frontier. All latencies were measured on a single large core of the same device using TFLite[1]. MobileNetV3-Small and Large are our proposed next-generation mobile models.

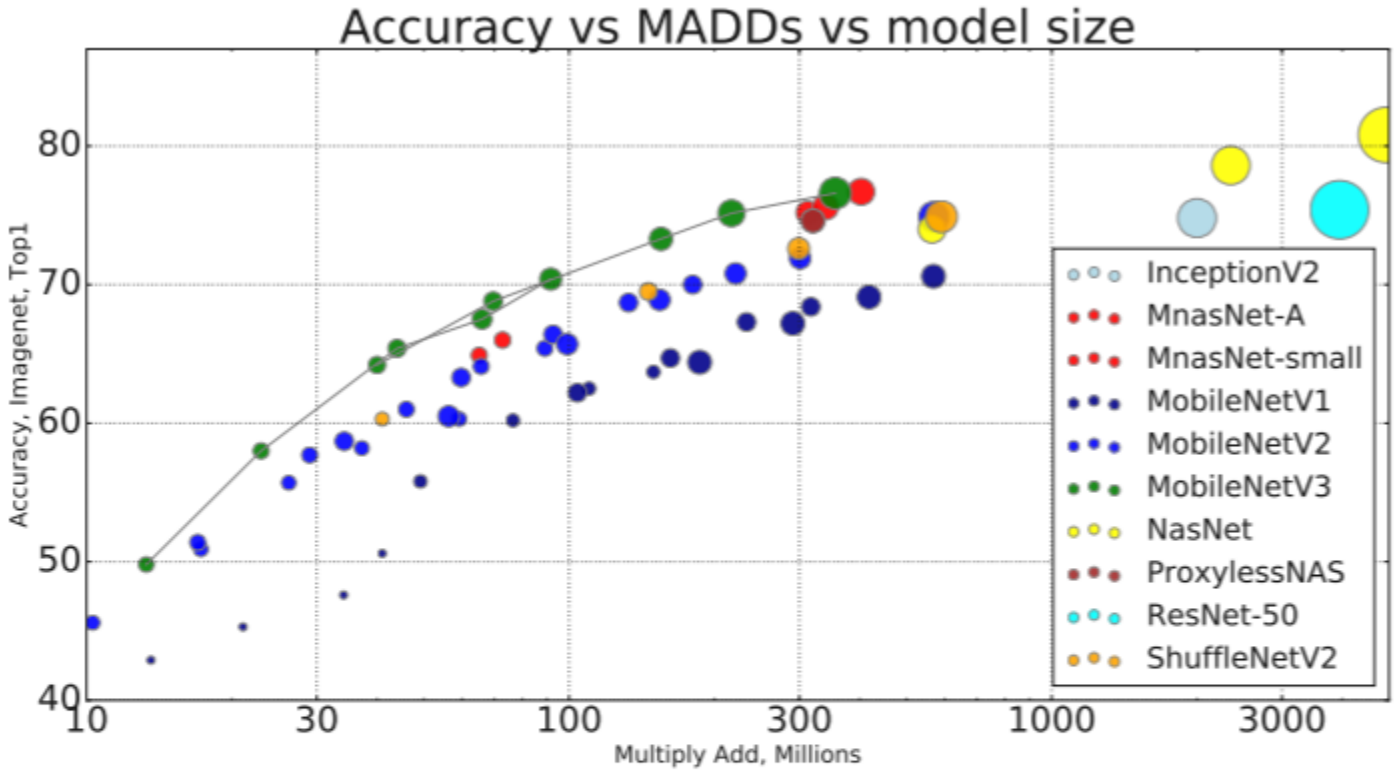
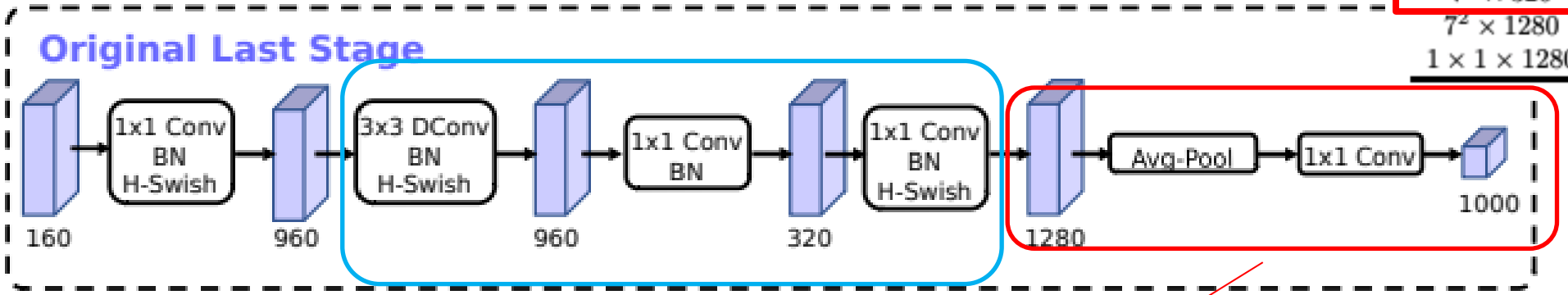


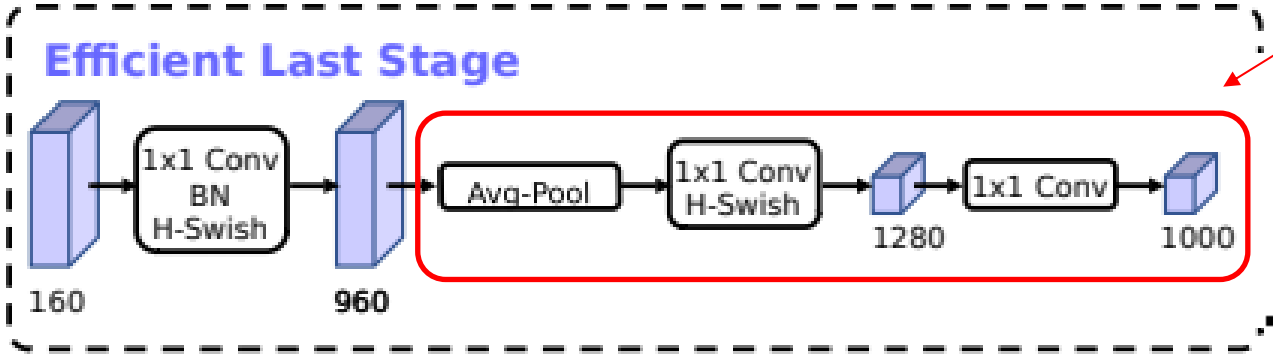
Figure 2. The trade-off between MAdds and top-1 accuracy. This allows to compare models that were targeted different hardware or software frameworks. All MobileNetV3 are for input resolution 224 and use multipliers 0.35, 0.5, 0.75, 1 and 1.25. See section 6 for other resolutions. Best viewed in color.

- MobileNet v1
  - Depthwise separable conv
- MobileNet v2
  - linear bottleneck
  - inverted residual structure
- MNasNet
  - lightweight attention modules based on squeeze and excitation into the bottleneck structure.
- MobileNet v3
  - Based on platform aware network architecture search for global architecture
  - Combinations of above (+ SE, + 5x5 depthwise)
  - Modified swish (hard swish)

Redesigning Expensive Layers

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	-	





$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

直接在1x1 spatial resolution 上作運算後分類

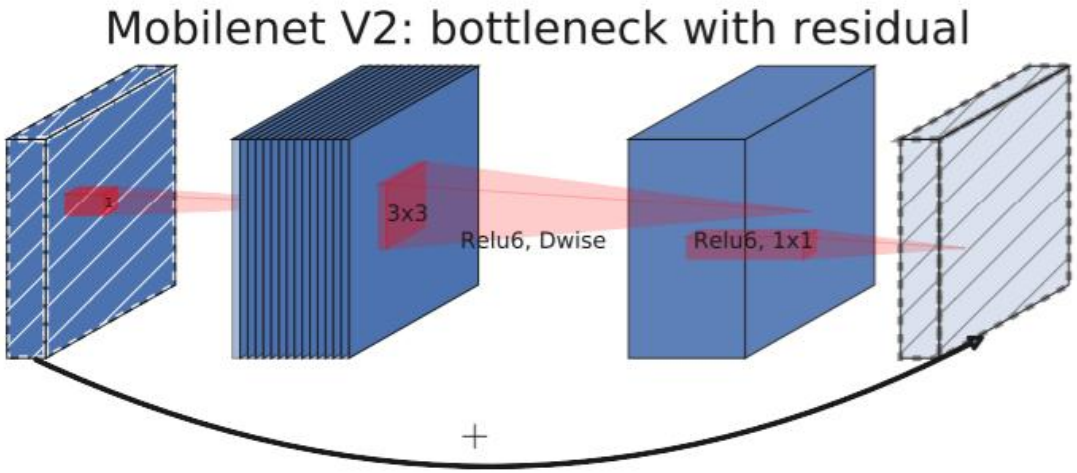


Figure 3. MobileNetV2 [39] layer (Inverted Residual and Linear Bottleneck). Each block consists of narrow input and output (bottleneck), which don't have nonlinearity, followed by expansion to a much higher-dimensional space and projection to the output. The residual connects bottleneck (rather than expansion).

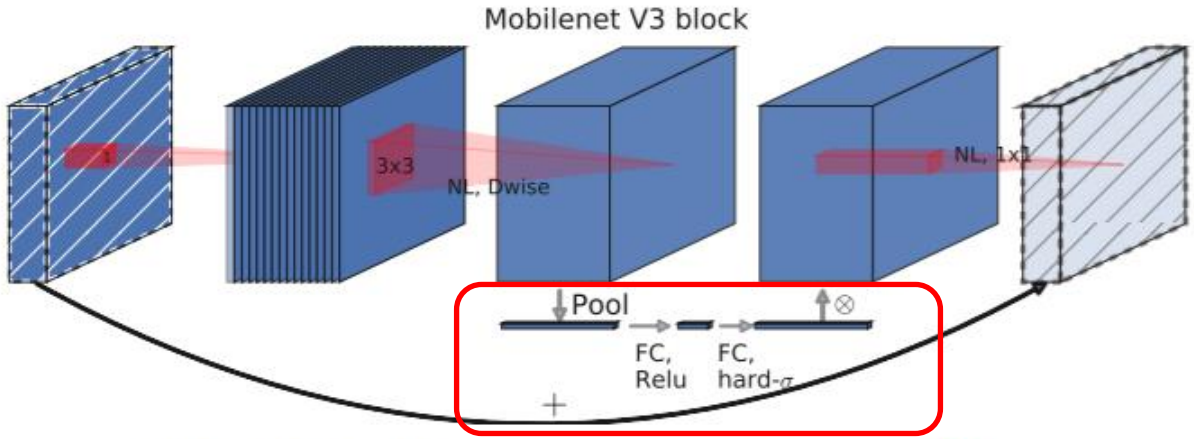
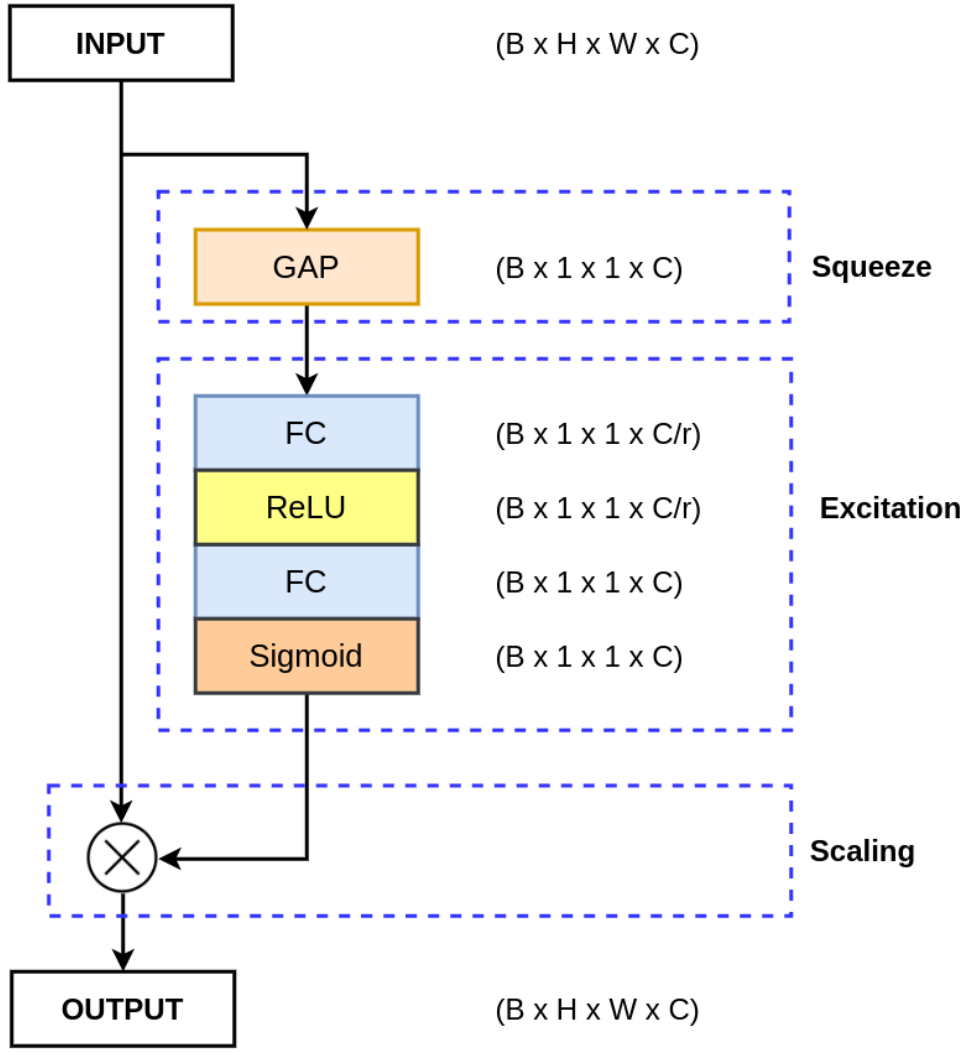


Figure 4. MobileNetV2 + Squeeze-and-Excite [20]. In contrast with [20] we apply the squeeze and excite in the residual layer. We use different nonlinearity depending on the layer, see section 5.2 for details.

該結構與傳統的residual block中維度先縮減後擴增相反，而是先將輸入的低維feature map擴增到高維，然後用depthwise convolution方式做卷積運算，然後再使用一個線性的捲積將其映射到低維空間中。

Squeeze-Excitation Network

- Squeeze-excitation module
  - Squeeze: Shrink feature map to get global information (global avg pooling)
  - Excitation: Learn a channel-wise weights to combine channel information
  - Channel attention



Input	Operator	exp size	#out	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

able 1. Specification for MobileNetV3-Large. SE denotes whether there is a Squeeze-And-Excite in that block. NL denotes he type of nonlinearity used. Here, HS denotes h-swish and RE enotes ReLU. NBN denotes no batch normalization. s denotes

作者發現swish激活函數能夠有效提高網絡的精度，然而，swish的計算量太大了。作者提出h-swish（hard version of swish）

$$\text{h-swish}[x] = x\frac{\text{ReLU6}(x+3)}{6}$$

MobileNet v4

- Universally efficient architecture for mobile GPU, DSP, and CPU
- UIB+Mobile MQA+ NAS

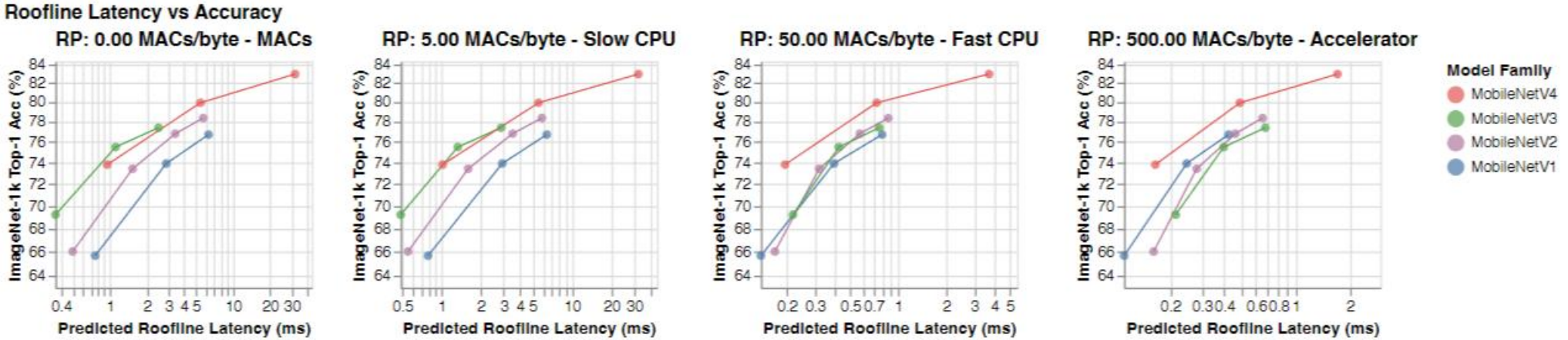


Fig. 2: Ridge Points and Latency/Accuracy Trade-Offs: The ridge point measures the relationship between memory bandwidth MACs relate in the roofline performance model. If memory bandwidth is constant, high-compute hardware (accelerators) have a higher ridge point than low-compute hardware (CPUs). MobileNetV4 is mostly Pareto-optimal from a ridge point of 0 to 500 MACs/byte. These analytically-derived (Eqs. 1) charts reflect the real hardware measurements in Figure 1.

$$\text{ModelTime} = \sum_i \max(\text{MACTime}_i, \text{MemTime}_i)$$
$$\text{MACTime}_i = \frac{\text{LayerMACs}_i}{\text{PeakMACs}}, \quad \text{MemTime}_i = \frac{\text{WeightBytes}_i + \text{ActivationBytes}_i}{\text{PeakMemBW}} \tag{1}$$

In the roofline model, hardware behavior is summarized by the *Ridge Point* (RP)—the ratio of a hardware's PeakMACs to PeakMemBW. I.e. the minimum operational intensity required to achieve maximum performance.[2] In order to

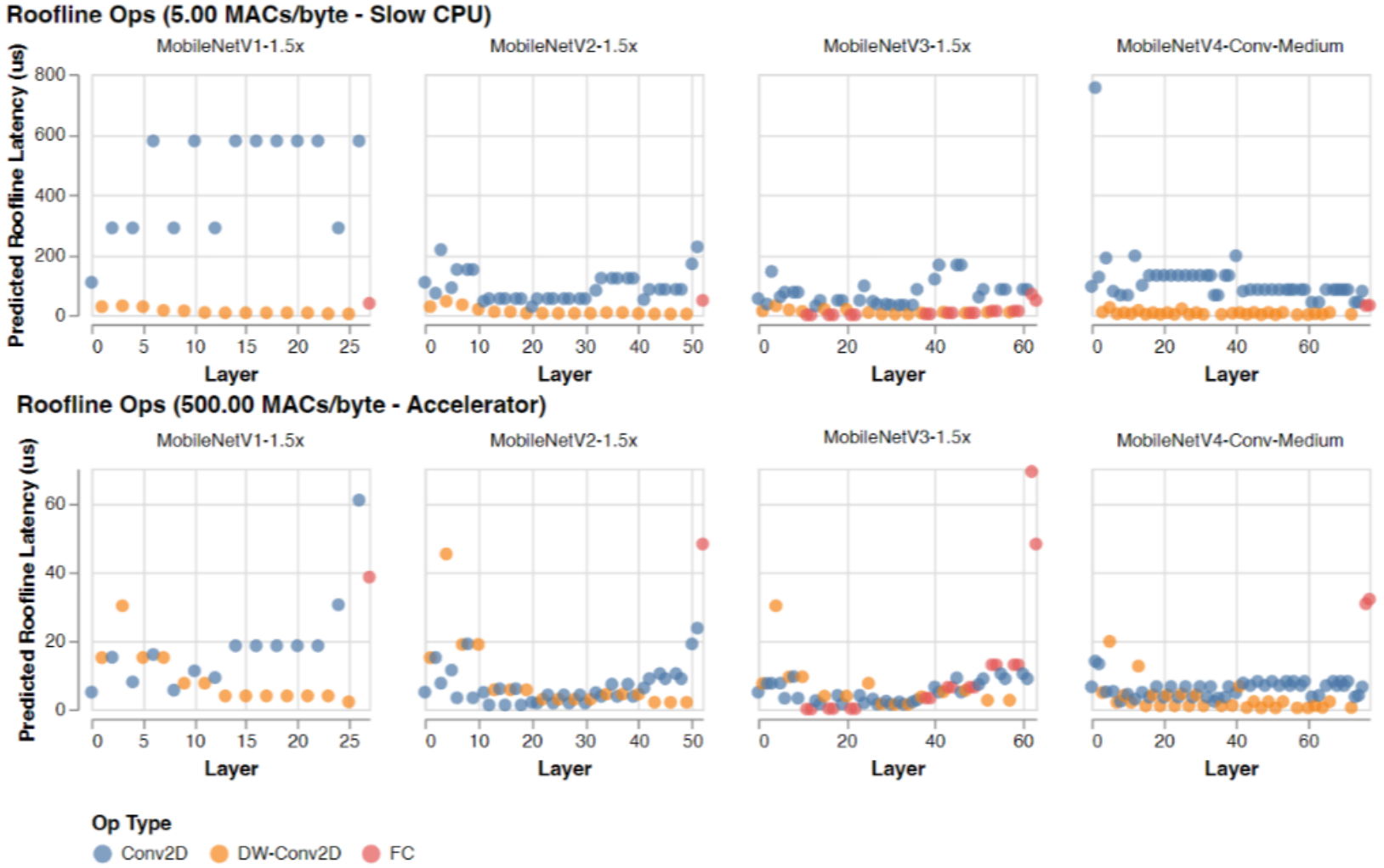


Fig. 3: Op Cost vs Ridge Point: Each sub-chart displays the roofline latency (Eqs. 1) of a network's ops. Networks start on the left. Large Conv2Ds are expensive on low ridge point (RP) hardware (top row), but add cheap model capacity on high-RP hardware (bottom row). FC layers and DW-Conv2Ds are cheap at low RPs and expensive at high RPs. MobileNetV4 balances MAC-intensive Conv2D layers and memory-intensive FC layers where they contribute most to the network—the beginning and end, respectively.

- Inverted Bottleneck (IB) + two optional DW
  - Different spatial and channel mixing, receptive field and HW utilization
  - Spatial mixing of features:
    - IB (expensive but greater capacity), ConvNext-like (cheaper with larger kernel size)
  - Extra DW
    - Inexpensively increase network depth and receptive field

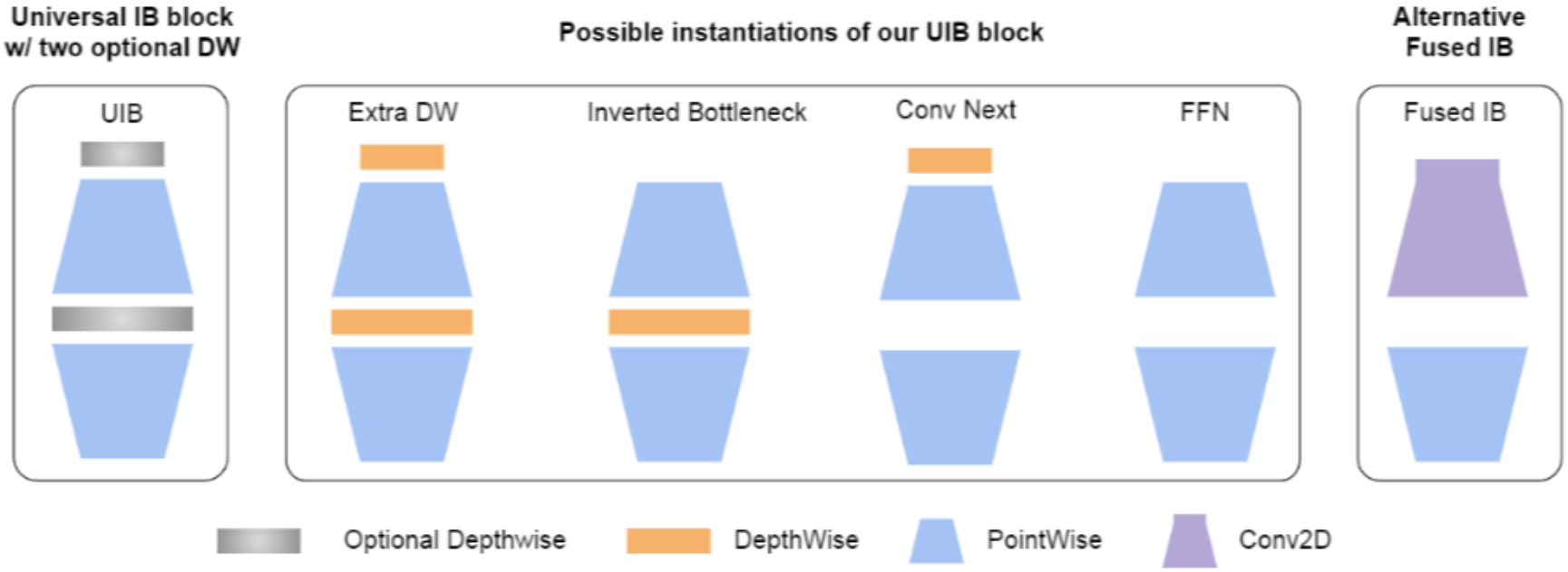


Fig. 4: Universal Inverted Bottleneck (UIB) blocks.

Mobile MQA

- Bottleneck at memory bandwidth
  - Consider Operational Intensity, the ratio of arithmetic operations to memory access
- MQA通過在所有head共用keys和values的方式大幅減少記憶體訪問開銷

Mobile MQA Here we present our Mobile MQA block:

$$\text{Mobile\_MQA}(\mathbf{X}) = \text{Concat}(\text{attention}_1, \ldots, \text{attention}_n)\mathbf{W}^O$$

$$\text{where attention}_j = \text{softmax}\left(\frac{(\mathbf{X}\mathbf{W}^{Q_j})(SR(\mathbf{X})\mathbf{W}^K)^T}{\sqrt{d_k}}\right)(SR(\mathbf{X})\mathbf{W}^V) \quad (2)$$

where SR denotes either spatial reduction, a DW with stride of 2 in our design, or the identity function in the case that spacial reduction isn't used. As shown in Table 2, incorporating asymmetric spatial down-sampling yields over 20% efficiency gain with minimal accuracy loss (-0.06%).

Model	Top-1	Params (M)	MACs (G)	Latency (ms) Pixel 6 CPU	Pixel 8 EdgeTPU	iPhone 13 CoreML	Pixel 4 Hexagon	Pixel 7 GPU	Samsung S23 CPU	Samsung S23 GPU
MobileNet-V2-0.5x [36]	66	2.0	0.1	2.4	0.7	0.5	2.9	8.3	1.8	1.9
MobileNet-V3L-0.5x [18]	69.2	2.7	0.1	2.4	0.8	0.45	3.5	9.9	2.0	2.1
MobileOne-S0 [43]	71.4	2.1	0.3	4.2	0.7	0.5	2.9	10.7	3.3	1.7
MobileNet-V2 [36]	73.4	3.5	0.3	5.0	0.7	0.7	3.9	13.6	4.1	2.5
MNv4-Conv-S	**73.8**	**3.8**	**0.2**	**2.4**	**0.7**	**0.6**	**2.4**	**8.4**	**1.8**	**2.0**
MobileNet-V1 [20]	74.0	4.2	0.6	6.1	0.8	0.7	3.2	13.0	4.6	2.1
FastViT-T8† [42]	75.6	3.6	0.7	49.3	1.3	0.7	*Failed*	40.7	43.6	24.7
MobileNet-V2-1.5x [36]	76.8	6.8	0.7	9.3	0.9	1.0	5.6	16.4	7.3	3.3
MultiHardware-MAX-1.5x [8]	77.9	8.9	0.8	9.8	1.0	-	5.7	23.2	-	4.1
MultiHardware-AVG-1.5x [8]	78.2	10.0	1.0	12.0	1.1	-	6.1	20.3	-	4.5
MobileNet-V2-2.0x [36]	78.4	11.2	1.1	13.9	1.1	1.5	6.9	19.1	10.6	4.2
MobileOne-S4 [43]	79.4	14.8	1.5	26.7	1.7	1.5	9.0	28.6	19.4	5.9
FastViT-S12† [42]	79.8	8.8	1.8	83.0	1.8	1.6	*Failed*	75.0	69.2	47.0
MIT-EfficientViT-B1-r224 [13]	79.4	9.1	0.5	-	-	2.4	-	-	18.1	5.0
MNv4-Conv-M	**79.9**	**9.2**	**1.0**	**11.4**	**1.1**	**1.1**	**7.3**	**18.1**	**8.6**	**4.1**
FastViT-SA12 [42]	80.6	10.9	1.9	86.5	2.0	1.6	*Failed*	79.6	69.5	52.1
MNv4-Hybrid-M	**80.7**	**10.5**	**1.2**	**14.3**	**1.5**	-	***Failed***	**17.9**	**10.8**	**5.9**
FastViT-SA24 [42]	82.6	20.6	3.8	171.6	3.2	2.4	*Failed*	131.9	136.3	107.5
MIT-EfficientViT-B2-r256 [13]	82.7	24.0	2.1	-	-	5.4	-	-	64.9	9.5
MNv4-Conv-L	**82.9**	**31**	**5.9**	**59.9**	**2.4**	**3.0**	**20.8**	**37.6**	**43.0**	**13.2**
ConvNext-S [32]	83.1	50	8.7	314.9	3.7	-	*Failed*	45.2	243.9	18.5
NextViT-B [26]	83.2	44.8	8.3	-	-	-	-	-	-	-
MIT-EfficientViT-B3-r224 [13]	83.5	49.0	4.0	-	-	12.2	-	-	125.9	18.4
MNv4-Hybrid-L	**83.4**	**35.9**	**7.2**	**87.6**	**3.8**	-	***Failed***	**61.3**	**61.8**	**18.1**
FastViT-SA36 [42]	83.6	30.4	5.6	241.6	4.3	-	*Failed*	186.5	206.3	138.1

Table 5: Classification results on ImageNet-1K [11], along with on-device benchmarks. Median latency is reported. – indicates that we did not benchmark a model due to missing corresponding model file for a platform. *Failed* indicates that the

Multi-path efficiency concerns: Group convolutions [52] and similar multi-path designs, despite lower FLOP counts, can be less efficient due to memory access complexity.

Hardware support matters: Advanced modules like Squeeze and Excite (SE) [21], GELU [16], LayerNorm [1] are not well supported on DSPs, with LayerNorm also lagging behind BatchNorm [23], and SE slow on accelerators.

The Power of Simplicity: Conventional components – depthwise and point-wise convolutions, ReLU [35], BatchNorm, and simple attention (e.g., MHSA) – demonstrate superior efficiency and hardware compatibility.

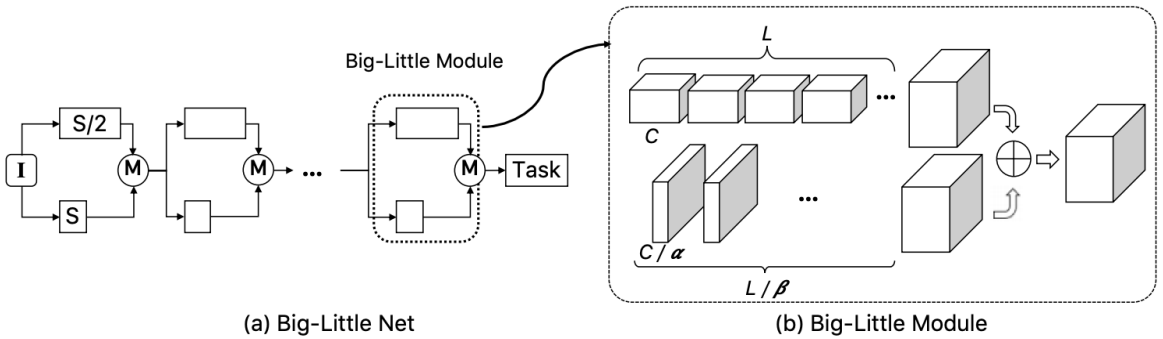
Based on these findings, we established a set of design principles:

- **Standard Components**: We prioritize widely supported elements for seamless deployment and hardware efficiency.
- **Flexible UIB Blocks**: Our novel searchable UIB building block allows for adaptable spatial and channel mixing, receptive field adjustments, and maximized computational utilization, facilitating a balanced compromise between efficiency and accuracy through Network Architecture Search (NAS).
- **Employ Straightforward Attention**: Our Mobile MQA mechanism prioritizes simplicity for optimal performance.

VARIATION OF DEPTHWISE/GROUP CONVOLUTION

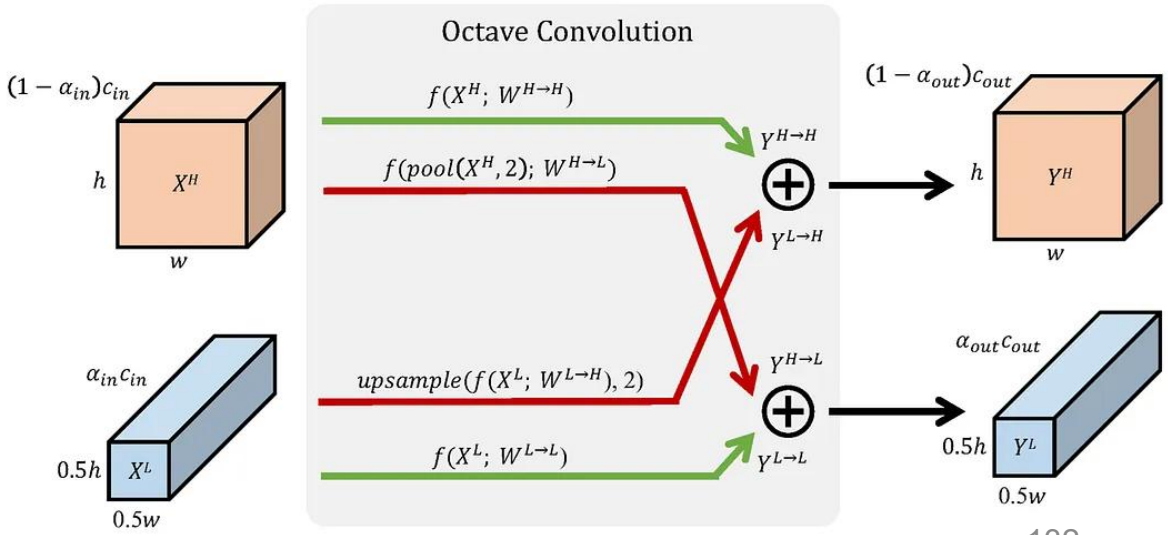
- Channel only decomposition: group convolution
  - + channel shuffle
  - Learn multi-scale features
  - Grouped by different filter size (big-little net)
  - Grouped by different input Octave convolution

more layers and channels at low resolutions

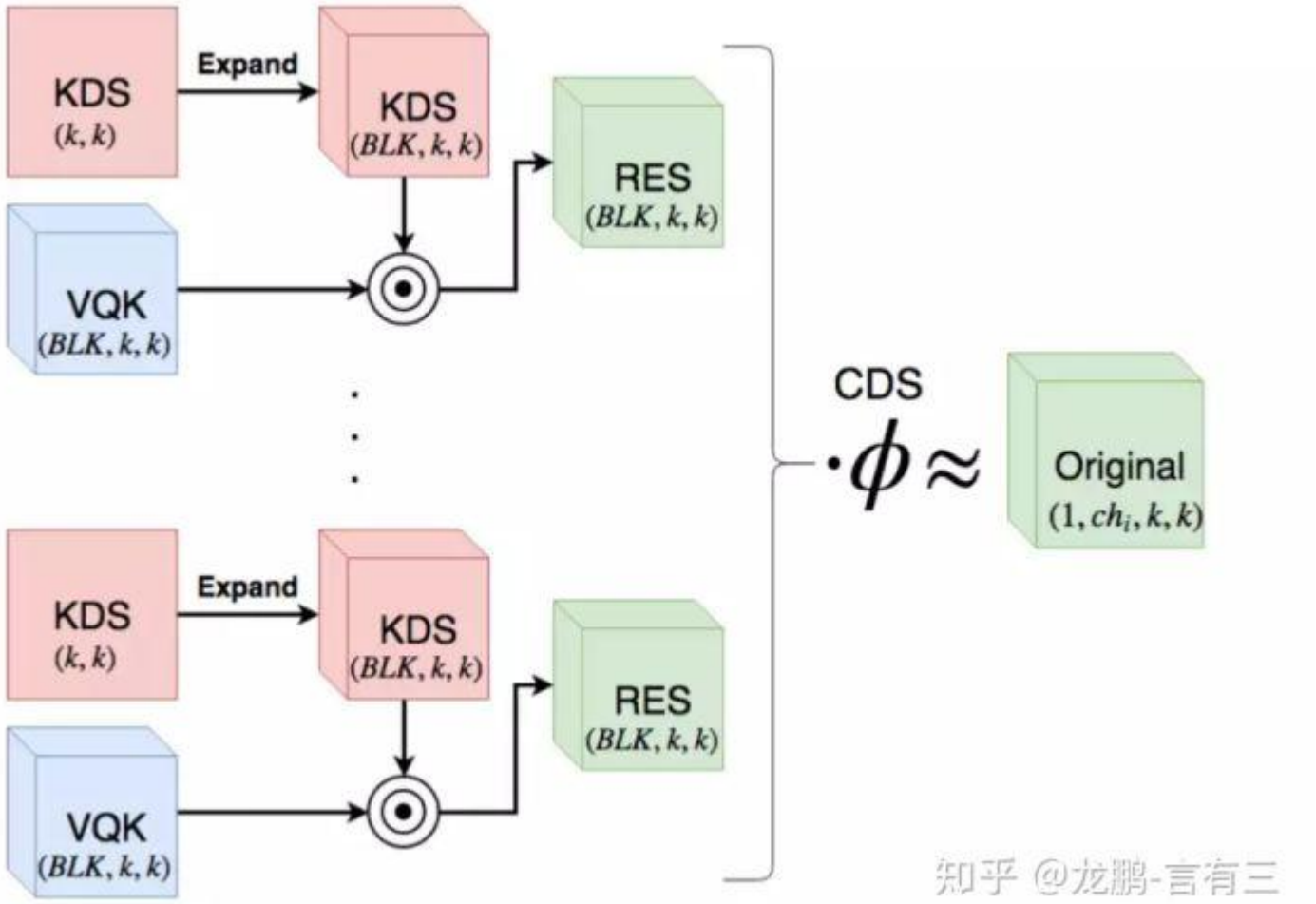


(a) Big-Little Net (b) Big-Little Module

fewer layers and channels at high resolution

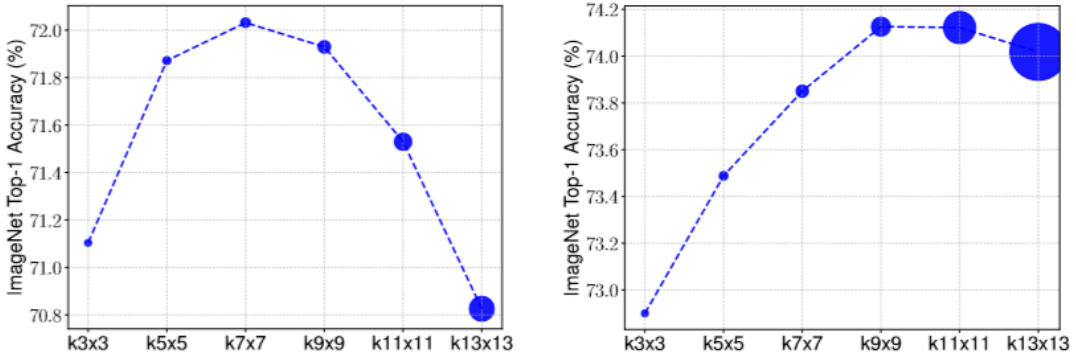


- Grouped by different precision



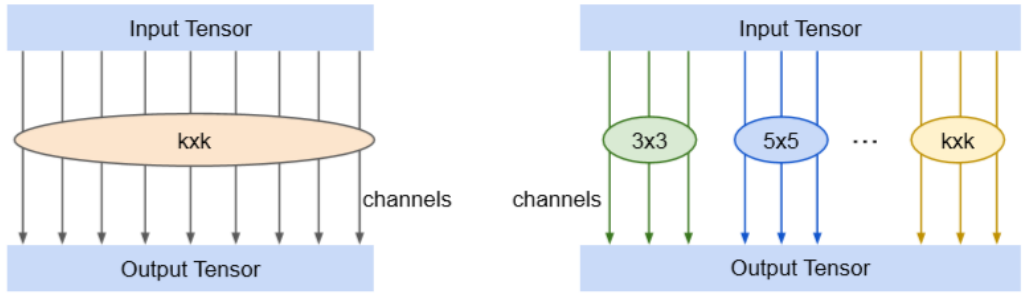
MIXNet: Mixed Depthwise Convolution

- do larger kernels always achieve higher accuracy?



(a) MobileNetV1 (b) MobileNetV2

Figure 1: **Accuracy vs kernel sizes** – Each point represents a model variant of MobileNet V1[7] and V2 [23], where model size is represented by point size. Larger kernels lead to more parameters, but the accuracy actually drops down when kernel size is larger than 9x9.

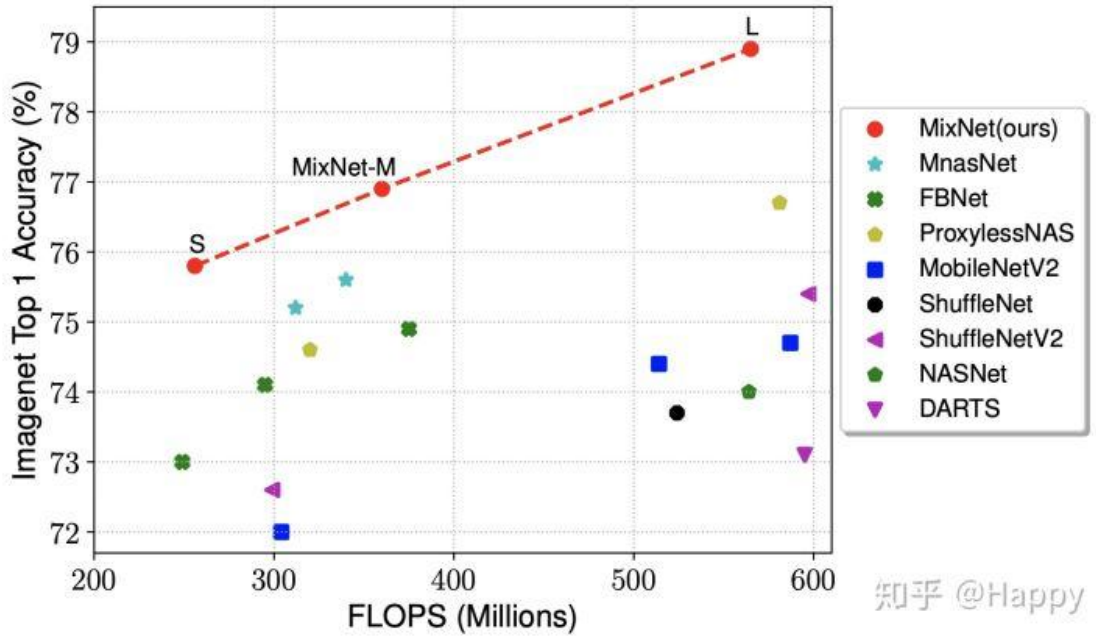


(a) Vanilla Depthwise Convolution (b) Our proposed MixConv

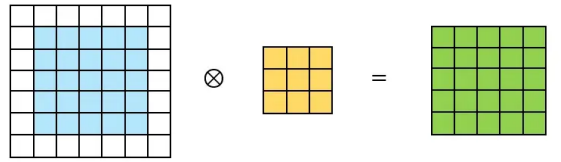
Figure 2: **Mixed depthwise convolution (MixConv)** – Unlike vanilla depthwise convolution that applies a single kernel to all channels, MixConv partitions channels into groups and apply different kernel size to each group.

當卷積核達到9x9時模型精度達到最大，繼續擴大卷積核會導致精度的下降。

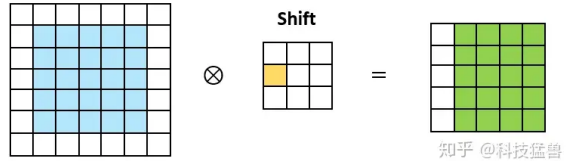
這同時也意味著單尺度核尺寸的局限性，模型需要大尺寸核捕獲高解析度模式，同時需要小尺寸核捕獲低解析度模式以及獲得更高的精度和效率。



Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions



- Replace depthwise with shift
  - This could be learnable (~10% needs to be shifted)



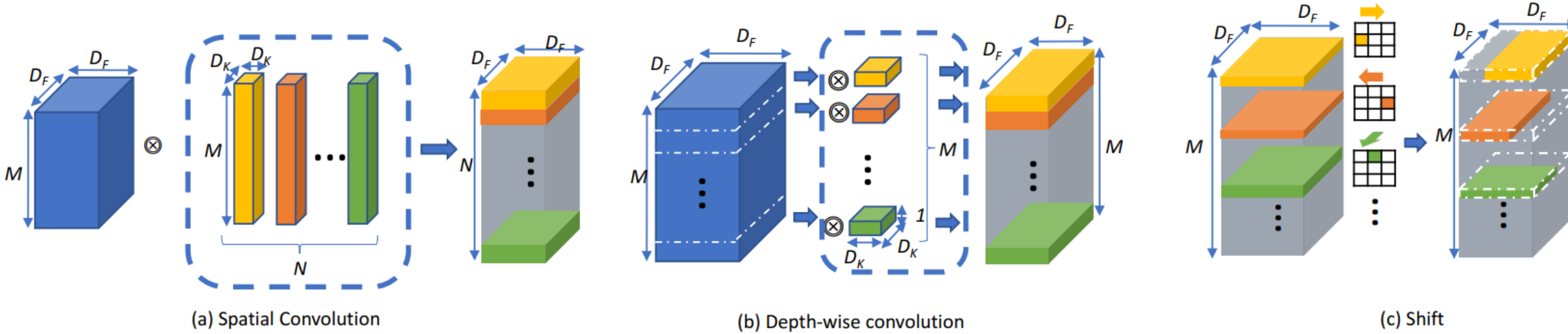


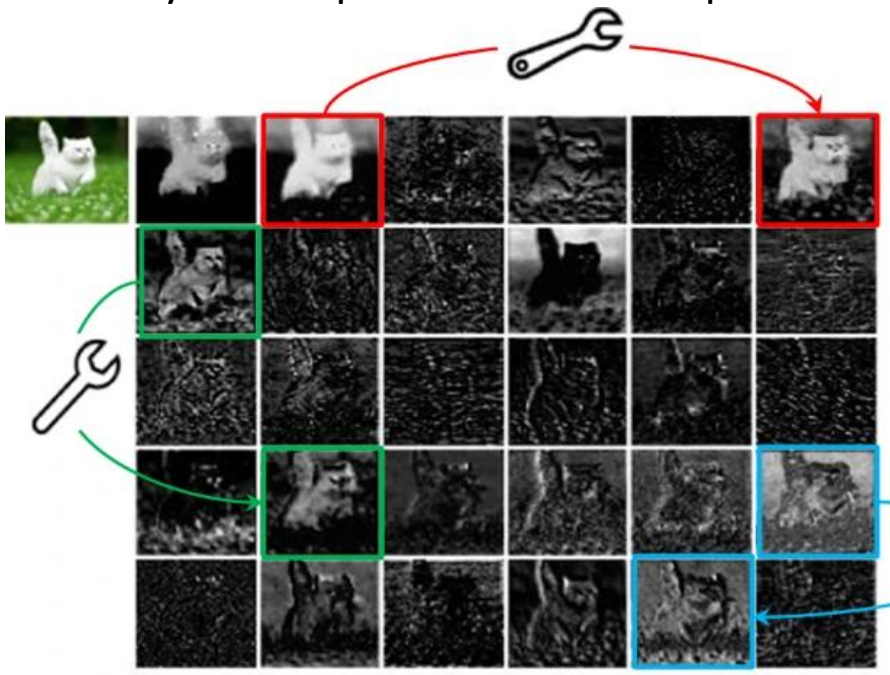
Figure 2: Illustration of (a) spatial convolutions, (b) depth-wise convolutions and (c) shift. In (c), the 3x3 grids denote a shi
matrix with a kernel size of 3. The lighted cell denotes a 1 at that position and white cells denote 0s.

Shift運算在表達能力上不如加法網絡和乘法，規模要顯著變大，Shift操作的特點是硬體效率高，表達能力弱

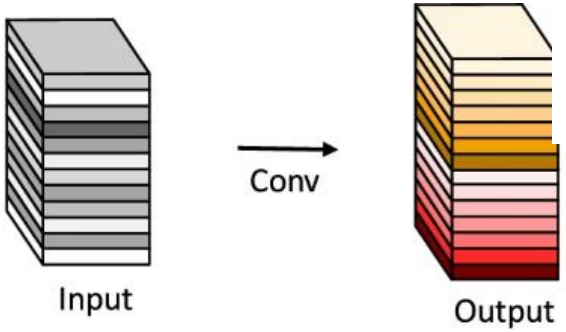
GhostNets on Heterogeneous Devices via Cheap Operations

- Add cheap operations for feature maps

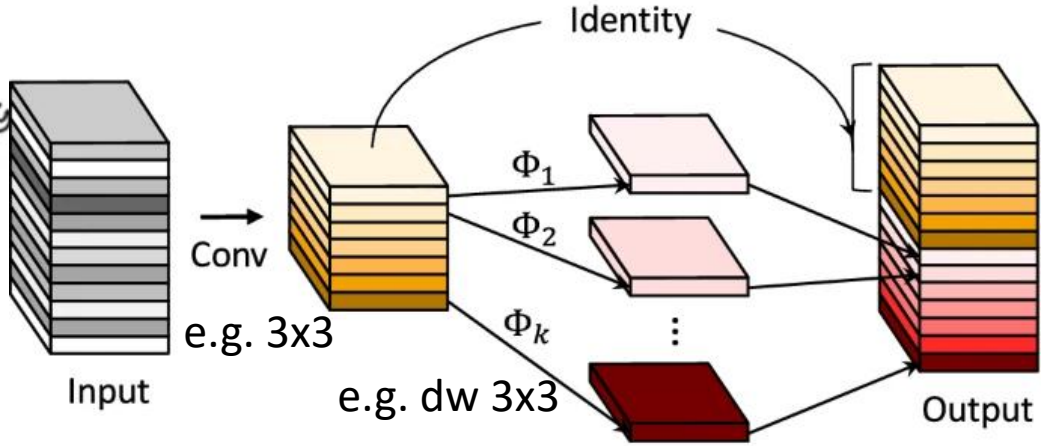
many similar pairs of feature maps



One feature map in the pair can be approximately obtained by transforming the other one through cheap operations



(a) The convolutional layer.



(b) The C-Ghost module.

Table 5. Comparison of state-of-the-art methods for compressing VGG-16 and ResNet-56 on CIFAR-10. - represents no reported results available.

Model	Weights	FLOPs	Acc. (%)
VGG-16	15M	313M	93.6
ℓ_1-VGG-16 [28, 34]	5.4M	206M	93.4
SBP-VGG-16 [14]	-	136M	92.5
Ghost-VGG-16 (s=2)	7.7M	158M	**93.7**
ResNet-56	0.85M	125M	93.0
CP-ResNet-56 [14]	-	63M	92.0
ℓ_1-ResNet-56 [28, 34]	0.73M	91M	92.5
AMC-ResNet-56 [13]	-	63M	91.9
Ghost-ResNet-56 (s=2)	0.43M	63M	**92.7**

ShiftAddNet

- Multiplication => L1 norm

$$Y(m,n,t) = \sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} S\big(X(m+i,n+j,k), F(i,j,k,t)\big),$$

$$Y(m,n,t) = -\sum_{i=0}^{d}\sum_{j=0}^{d}\sum_{k=0}^{c_{in}} |X(m+i,n+j,k) - F(i,j,k,t)|.$$

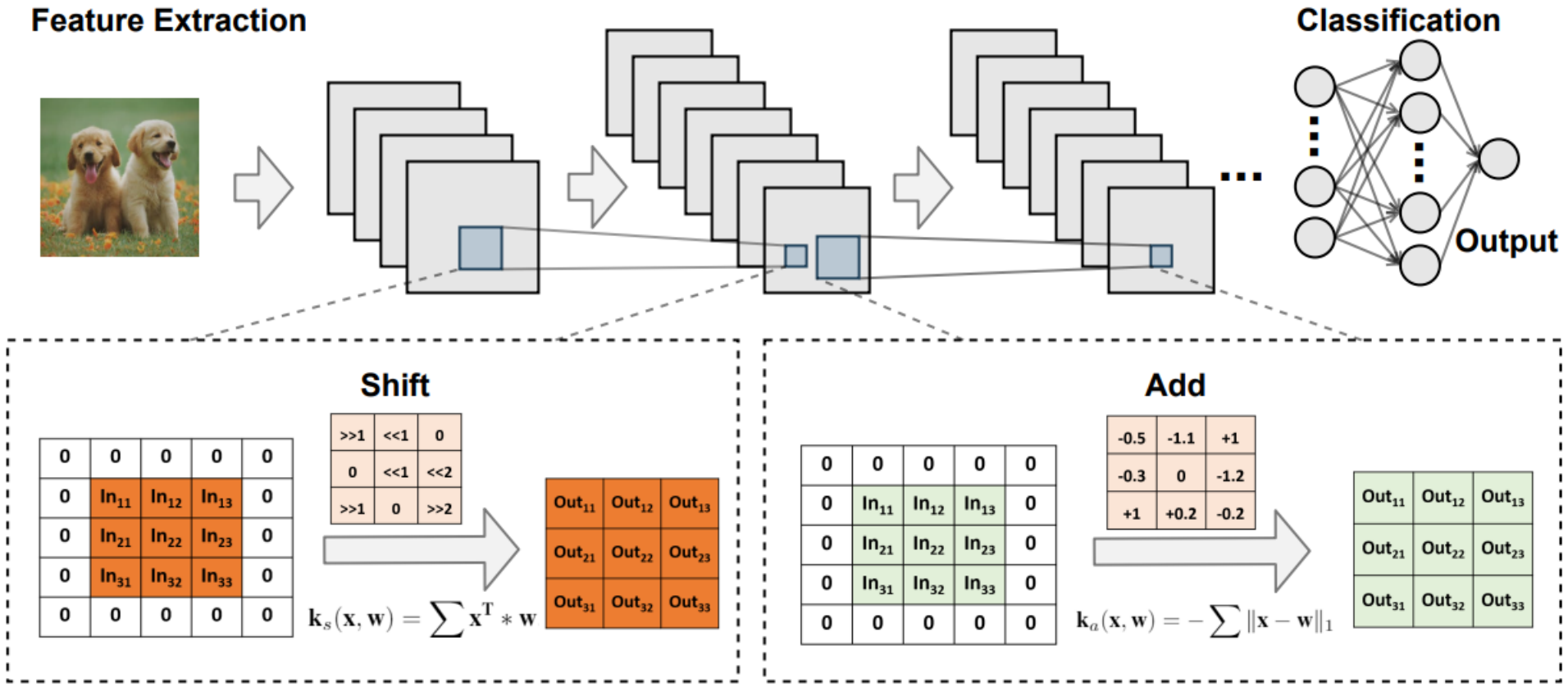
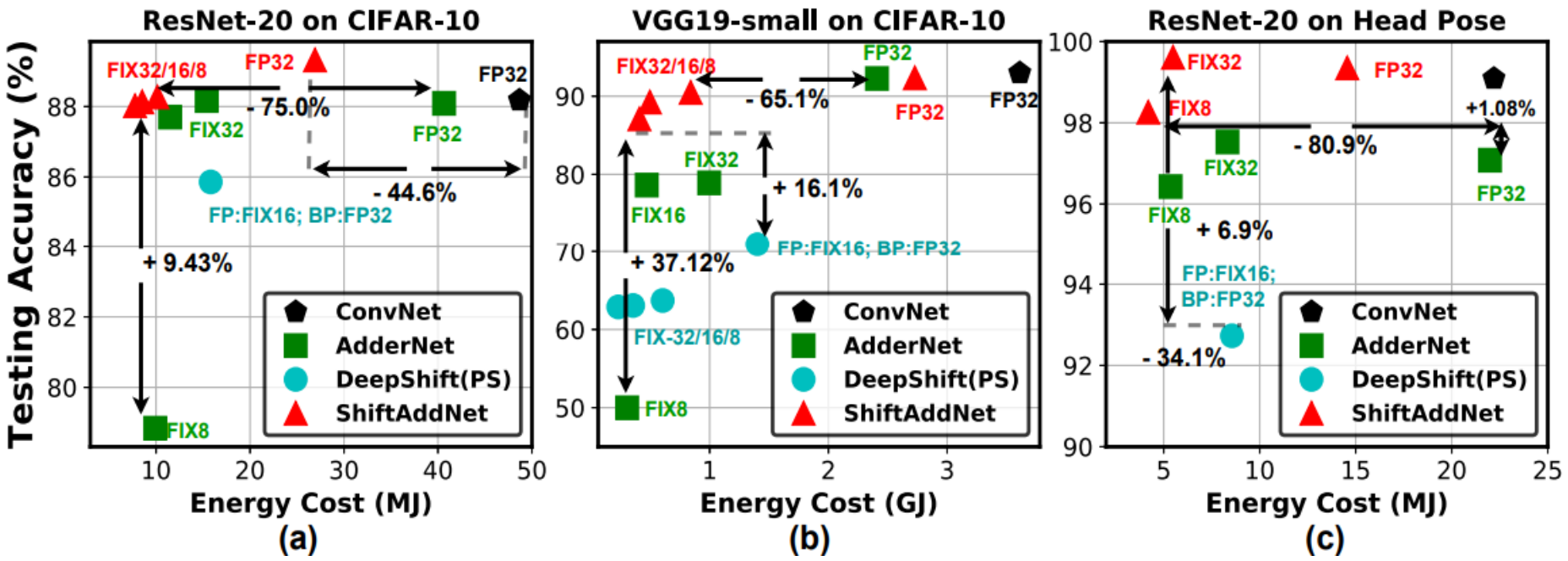
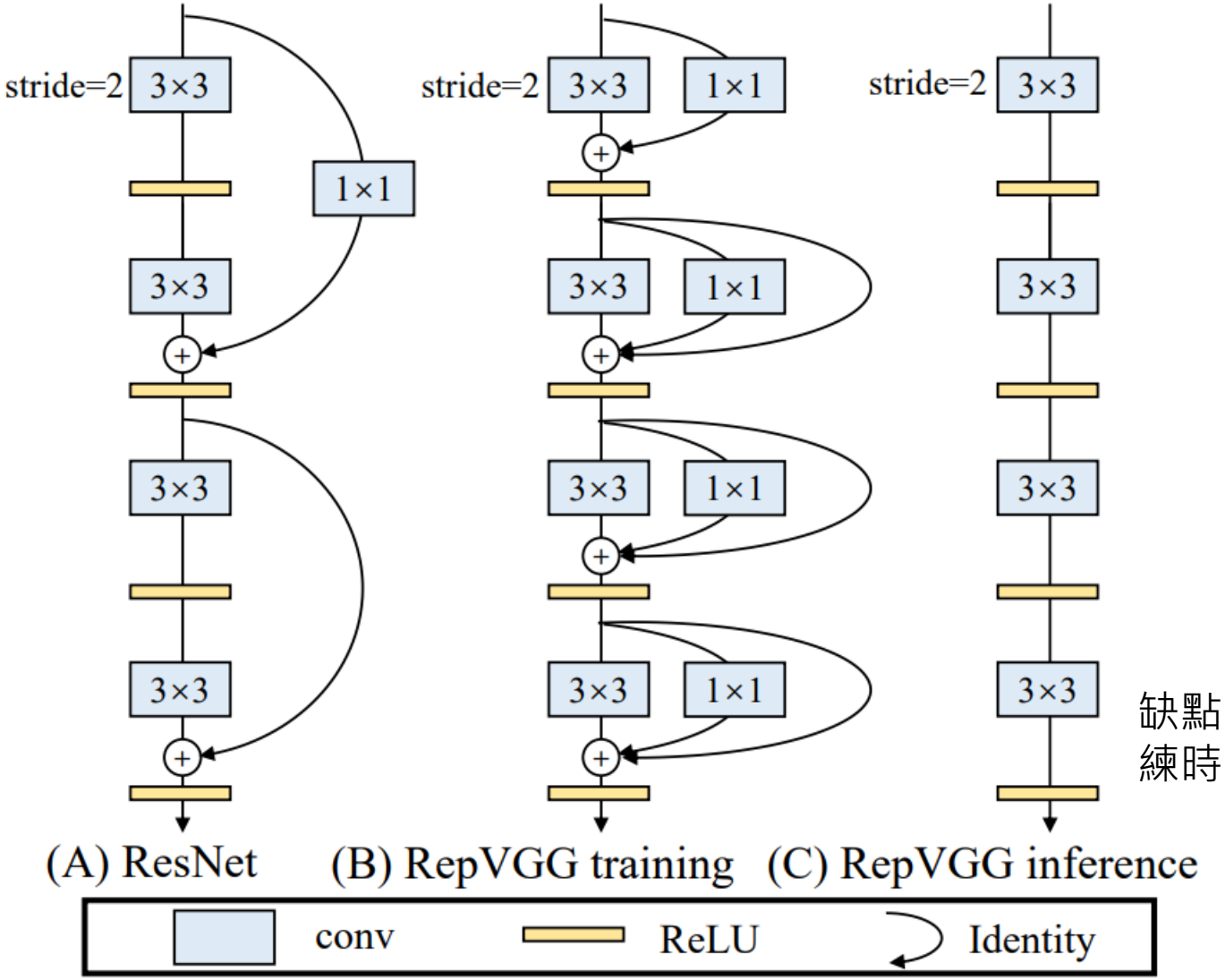


Figure 1: Illustrating the overview structure of ShiftAddNet.


ResNet-20 on CIFAR-10
Testing Accuracy (%)
FIX32/16/8
FP32
FP32
- 75.0%
FIX32
FP32
- 44.6%
FP:FIX16; BP:FP32
+ 9.43%
FIX8
ConvNet
AdderNet
DeepShift(PS)
ShiftAddNet
Energy Cost (MJ)
(a)
VGG19-small on CIFAR-10
FIX32/16/8
FP32
FP32
FP32
- 65.1%
FIX32
+ 16.1%
FIX16
FP:FIX16; BP:FP32
+ 37.12%
FIX-32/16/8
FIX8
ConvNet
AdderNet
DeepShift(PS)
ShiftAddNet
Energy Cost (GJ)
(b)
ResNet-20 on Head Pose
FIX32
FP32
FIX8
+1.08%
- 80.9%
FIX32
FP32
FIX8
+ 6.9%
FP:FIX16;
BP:FP32
- 34.1%
ConvNet
AdderNet
DeepShift(PS)
ShiftAddNet
Energy Cost (MJ)
(c)

Model Reparameterization: RepVGG



缺點: 訓練過程中有多個分支，需要更多顯存和訓練時間

Model Reparameterization: RepVGG2

- RepVGG加入了結構先驗（如1x1，identity分支），並使用常規優化器訓練。而RepOptVGG則是將這種先驗知識加入優化器實作中

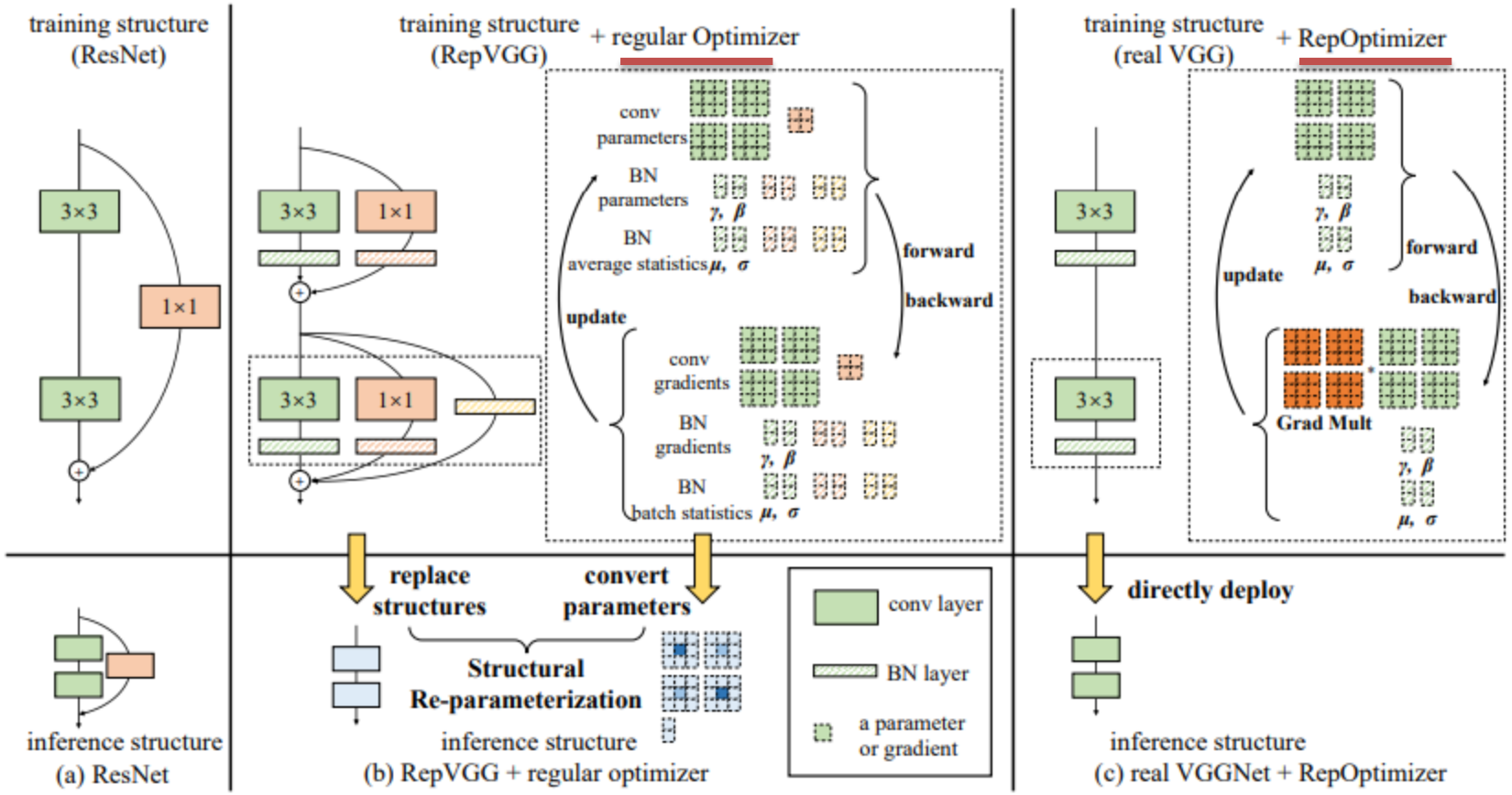
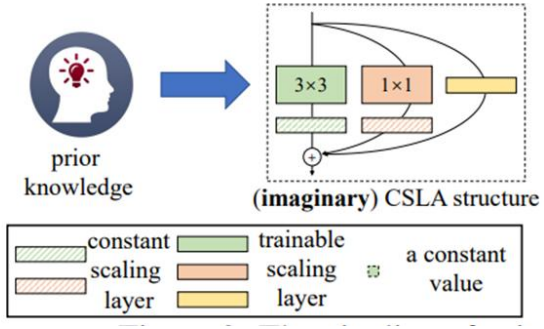


Figure 1: The differences among (a) a regular model, (b) a structurally reparameterized RepVGG



- Constant-Scale Linear Addition(CSLA)
  - 每個分支包含一個線性可訓練參數，加上一個常數縮放值，只要該縮放值設定合理，則模型效能依舊會很高。
  - 我們先從一個簡單的CSLA範例入手，考慮一個輸入，經過2個卷積分支+線性縮放，並加到一個輸出中：

$$\mathrm{Y}_{CSLA} = \alpha_A(\mathrm{X} * \mathrm{W}^{(A)}) + \alpha_B(\mathrm{X} * \mathrm{W}^{(B)}).$$

融合的權重為 $$\mathrm{W}'^{(0)} \leftarrow \alpha_A \mathrm{W}^{(A)(0)} + \alpha_B \mathrm{W}^{(B)(0)}$$

更新規則 $$\mathrm{W}'^{(i+1)} \leftarrow \mathrm{W}'^{(i)} - \lambda(\alpha_A^2 + \alpha_B^2)\frac{\partial L}{\partial \mathrm{W}'^{(i)}}$$

- 進一步拓展到多分支中，假設s，t分別是3x3卷積，1x1卷積的縮放係數，那麼對應的更新規則為 (Gradient multiplier)

$$\mathrm{M}_{c,d,p,q} = \begin{cases} 1 + \mathbf{s}_c^2 + \mathbf{t}_c^2 & \text{if } c = d, p = 2 \text{ and } q = 2\,, \\ \mathbf{s}_c^2 + \mathbf{t}_c^2 & \text{if } c \neq d, p = 2 \text{ and } q = 2\,, \\ \mathbf{s}_c^2 & \text{elsewise}\,. \end{cases}$$

  - 第一條公式對應輸入通道==輸出通道，此時共有3個分支，分別是identity，conv3x3， conv1x1
  - 第二個公式對應輸入頻道！=輸出通道，此時只有conv3x3, conv1x1兩個分支
  - 第三條公式對應其他情況

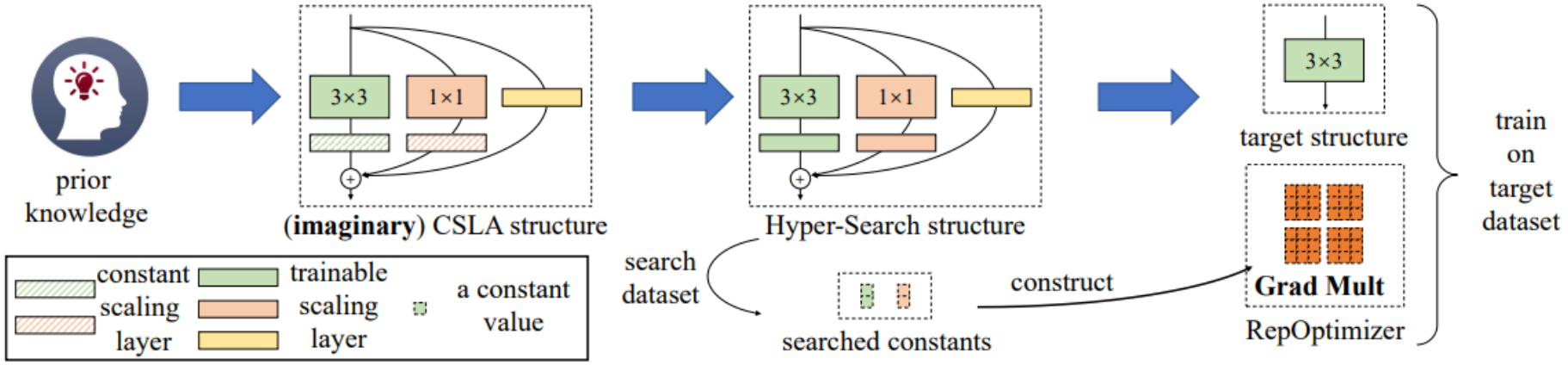


Figure 2: The pipeline of using RepOptimizers with RepOpt-VGG as an example.

Table 2: ImageNet accuracy and training speed (2080Ti, measured in samples/second/GPU).

Model	Train MaxBS	Top-1 accuracy @BS=32	Top-1 accuracy @MaxBS	Train speed @BS=32	Train speed @MaxBS	Params (M) train	Params (M) inference
RepVGG-B1	198	78.41±0.01	78.65±0.03	213	243	57.4	51.8
RepOpt-VGG-B1	**260**	78.47±0.03	78.62±0.03	**380**	**445**	**51.8**	51.8
RepVGG-B2	153	79.53±0.08	79.78±0.14	142	163	89.0	80.3
RepOpt-VGG-B2	**200**	79.36±0.09	79.66±0.12	**246**	**264**	**80.3**	80.3
RepVGG-L1	106	79.39±0.07	79.84±0.02	124	136	84.3	76.0
RepOpt-VGG-L1	**140**	79.46±0.02	79.83±0.03	**221**	**252**	**76.0**	76.0
RepVGG-L2	77	80.52±0.05	80.56±0.09	82	86	131.0	118.1
RepOpt-VGG-L2	**103**	80.29±0.02	80.53±0.09	**138**	**149**	**118.1**	118.1

Table 8: Quantized accuracy and standard deviation of parameters from RepOpt-VGG or RepVGG.

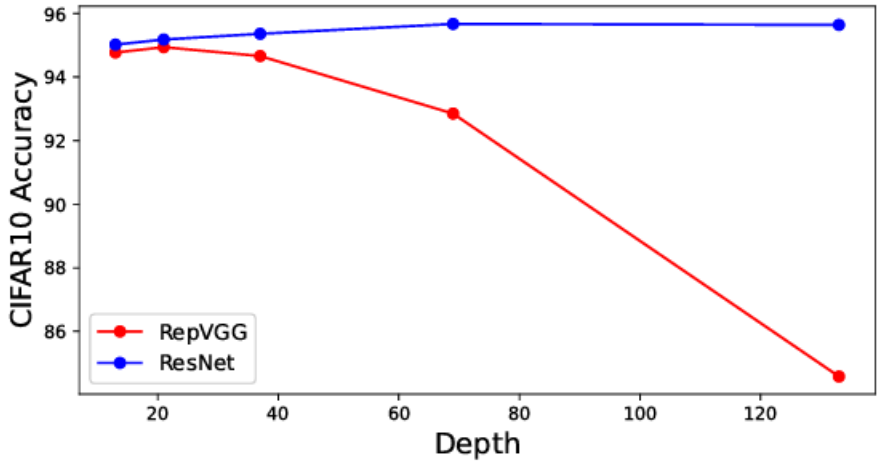
Model	Quantized accuracy	Standard deviation of parameters
RepOpt-VGG-B1	75.89	0.0066
RepVGG-B1	54.55	0.0234

訓練中沒有多分支，可訓練的batchsize也能增大，模型吞吐量也提升不少。對於量化訓練十分友善

在一個小資料集帶上bn訓練，得到一個適當的線性縮放值（就當作是起到BN的作用），然後將這個線性縮放值作為可學習參數的初始化值，來進行完整訓練（此時就沒有BN了，也就可以用repopt了）

RMNET: EQUIVALENTLY REMOVING RESIDUAL CONNECTION FROM NETWORKS

- RM: reserving and merging to convert residual network to plain net
- Drawback of RepVGG
  - Degrade a lot for deeper network
- ResNet
  - ResNet-50中的殘差連接約佔特性圖全部記憶體使用量的40%，這將減緩推理過程。此外，網路中的殘留連接對網路剪枝也不友善。相較之下，VGG-like模型(本文也稱plain模型)只有一條路徑，速度快、記憶體經濟、平行友善。



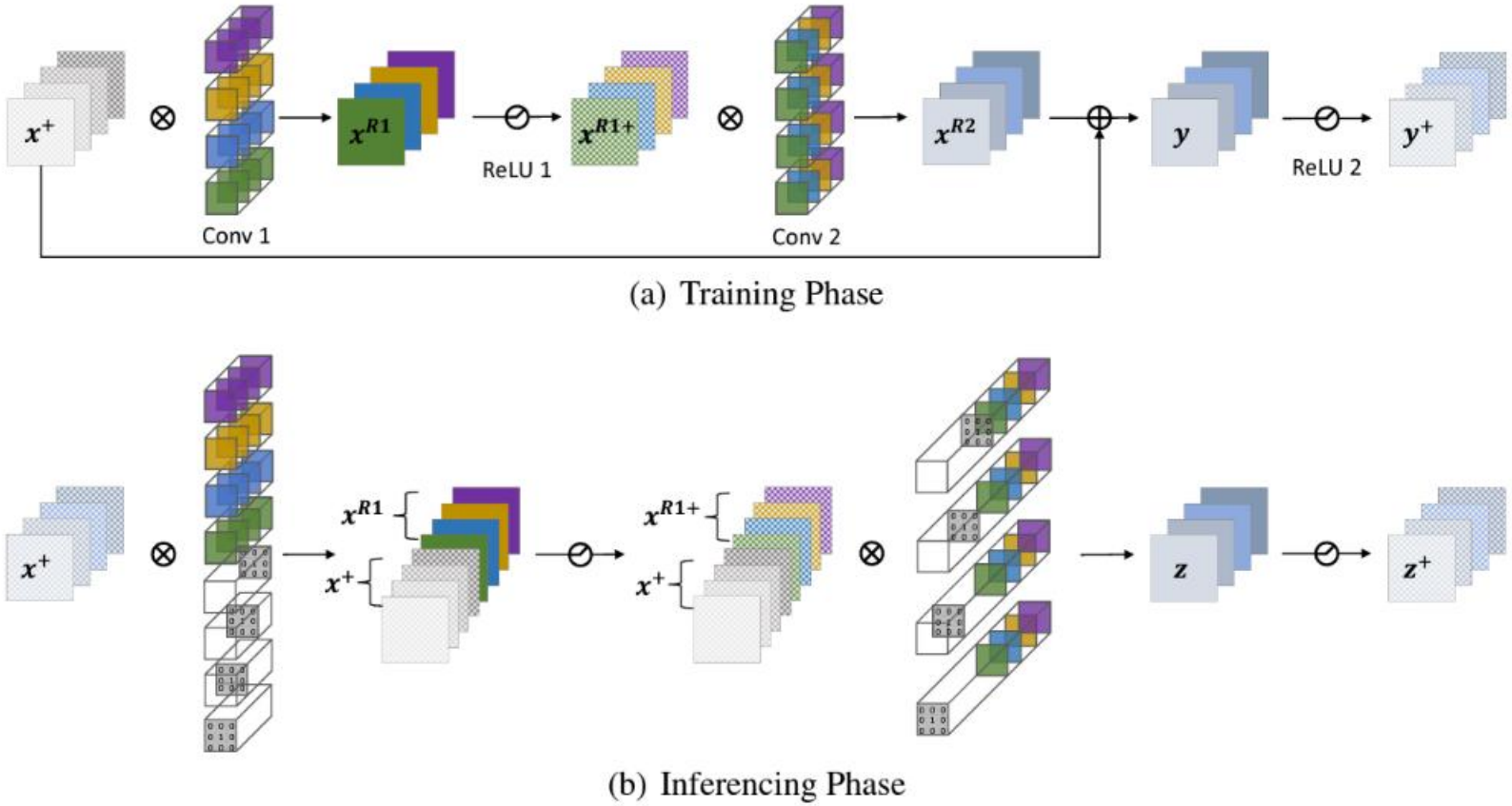


Figure 3: The upper figure shows a ResBlock during the training phase. The lower figure is the converted RMBlock for inference, which has no residual connections. Both blocks have equal output given the same input.

Reserving

1. Add Dirac initialized filters to conv1 to reserve channel information of input feature maps
2. Make BN to be identity function
3. For ReLU, set parameter to 1 in PReLU

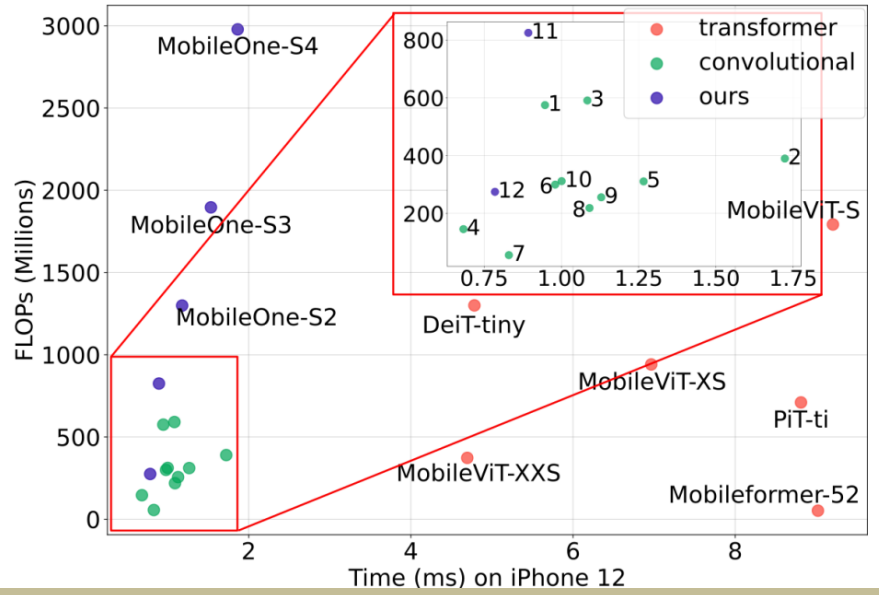
Merging

1. Extend input channels in Conv2 and dirac initialize these channels
2. 第i個通道的值z是原始的第i個濾波器輸出與原始ResBlock中第i個輸入特徵映射的和

Model Reparameterization: MobileOne

- MobileOne
  - Mobilenet + RepVGG+ training trick
  - Optimized for iphone12
  - 1ms inference for ImageNet

FLOPs或參數量的降低與inference 速度不直接相關



Hyperparaemeter k to control repara branch

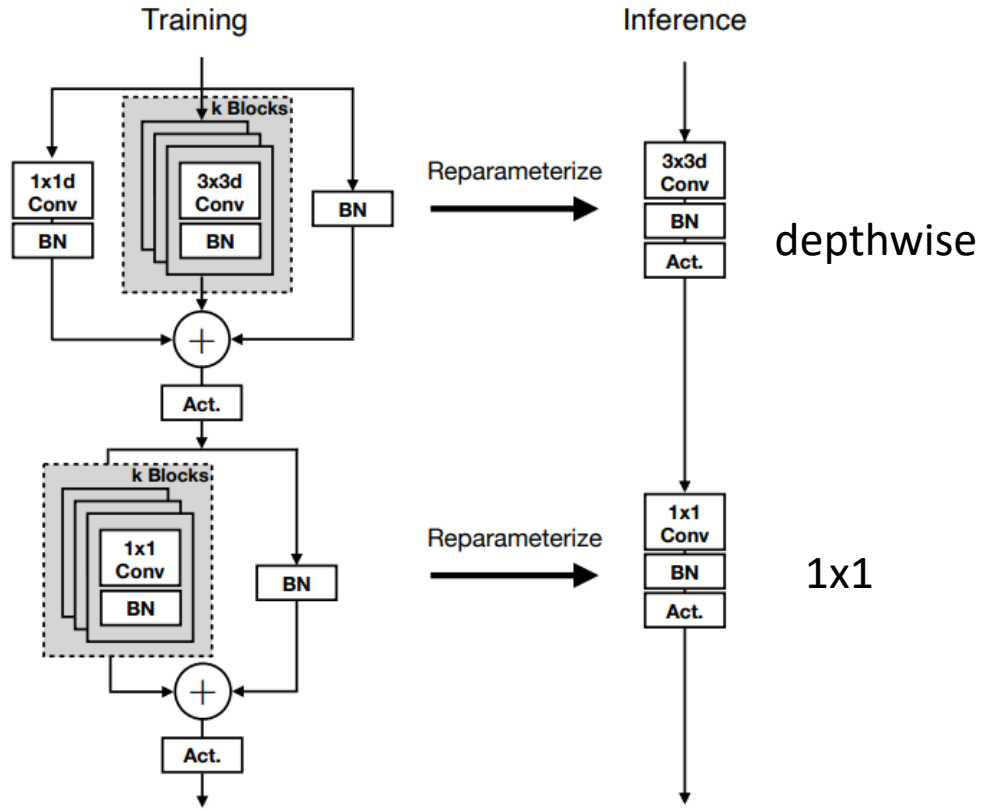
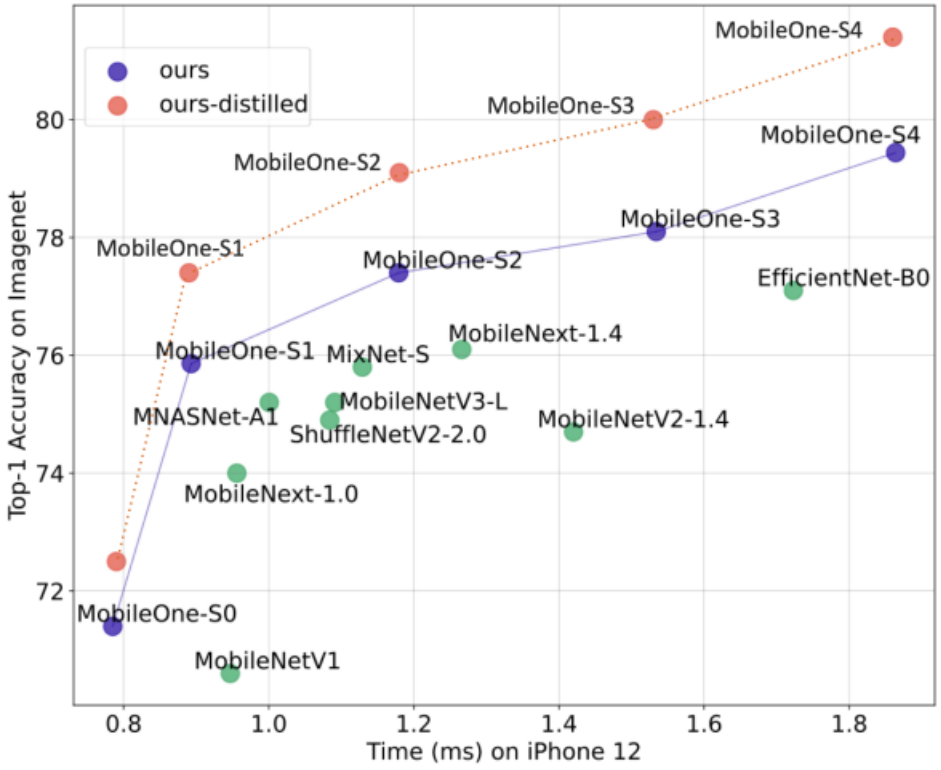
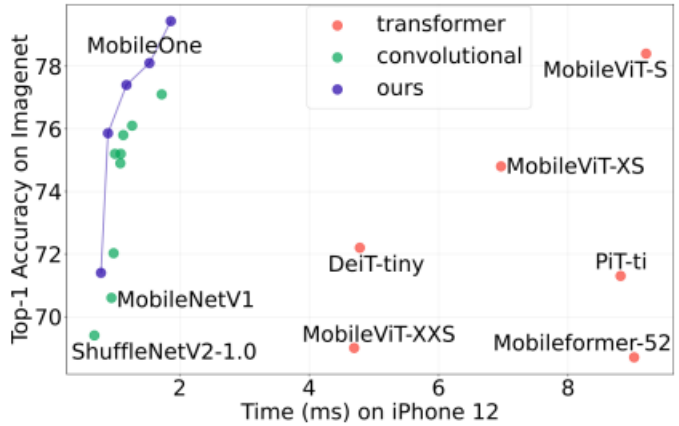


Figure 3. MobileOne block has two different structures at train time and test time. Left: Train time MobileOne block with reparameterizable branches. Right: MobileOne block at inference where the branches are reparameterized. Either ReLU or SE-ReLU is used as activation. The trivial over-parameterization fac-

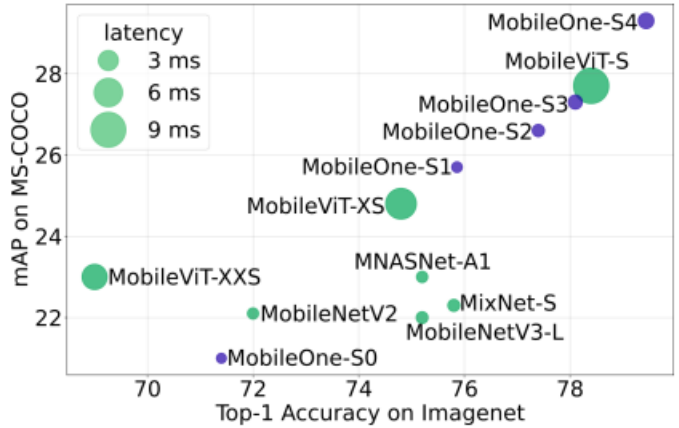
MobileOne: An Improved One millisecond Mobile Backbone



(a) Top 1 accuracy vs Latency on iPhone 12.



(b) Zoomed out (a)



(c) Top-1 accuracy vs mAP.

Figure 1. We show comparisons of Top-1 accuracy on image classification vs latency on an iPhone 12 (a), and zoomed out area (b) to include recent transformer architectures. We show mAP on object detection vs Top-1 accuracy on image classification in (c) with size of the marker indicating latency of the backbone on iPhone 12. Our models have significantly smaller latency compared to related works. Please refer to supp. mat. for higher resolution figures.

SCALING

EfficientNet: Compound Scaling CNN

- Systematic, principled scaling of depth, width and resolution

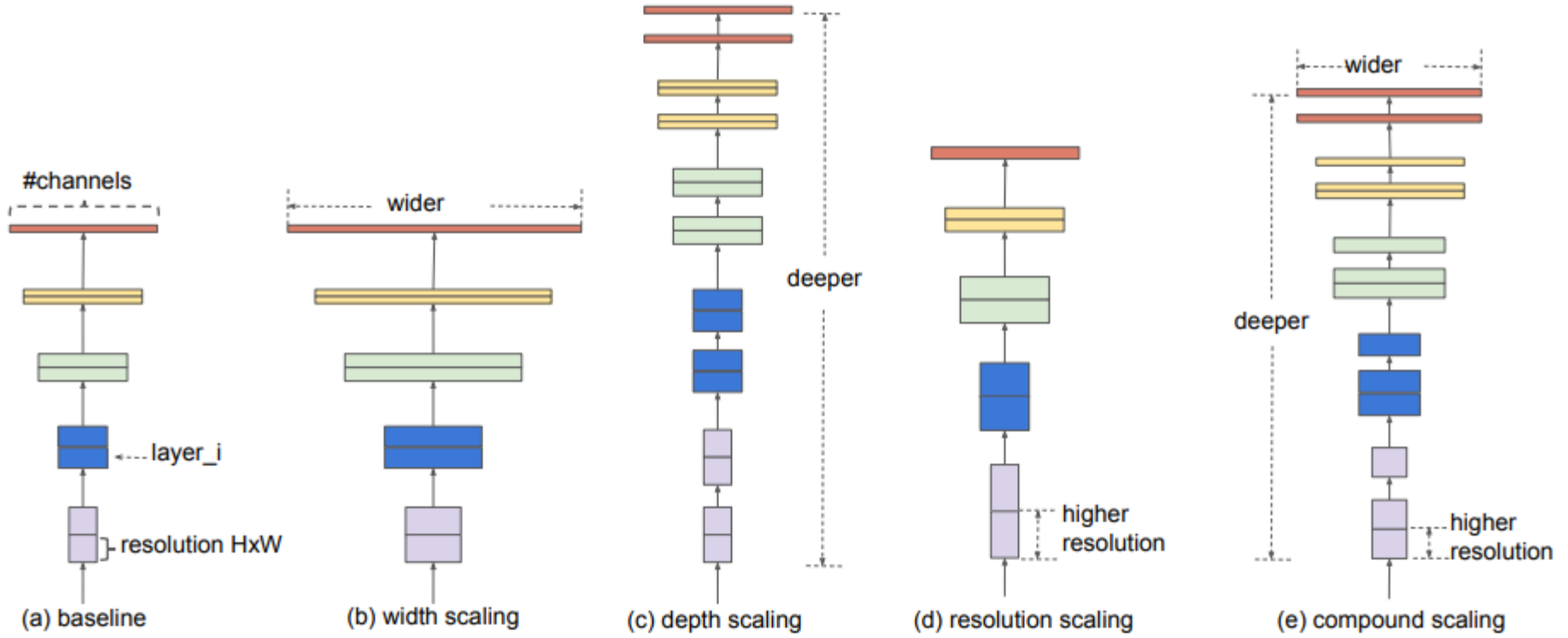


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

- Scaling up any dimension of network width, depth, or resolution improves accuracy, but the accuracy gain diminishes for bigger models

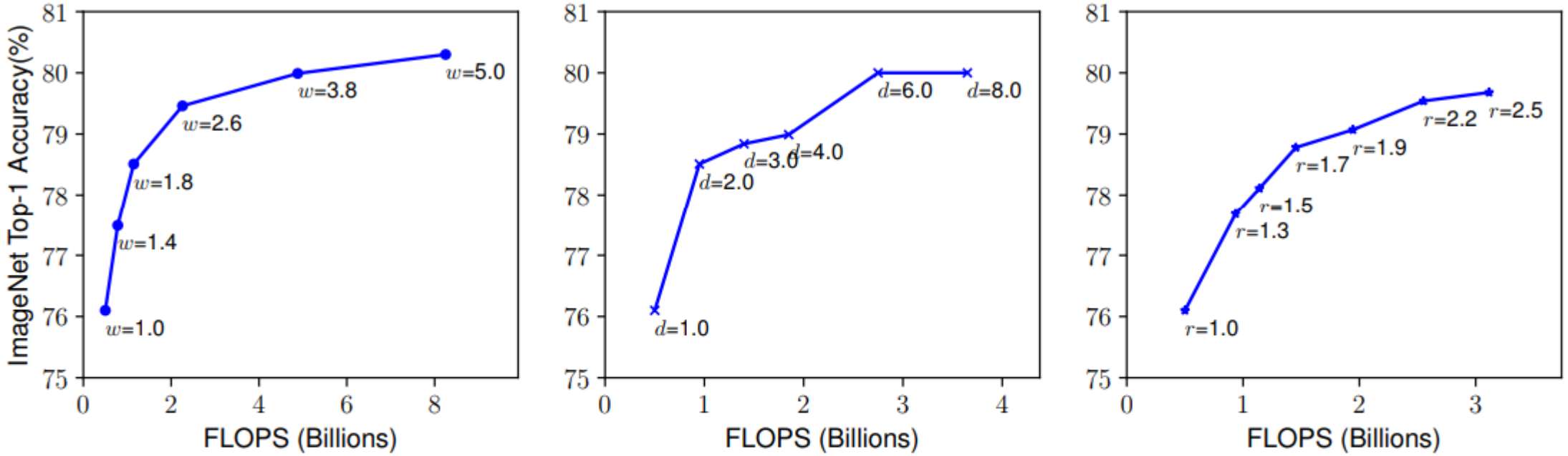


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

Proportional to d, w^2, r^2

- Double width or resolution will increase FLOPS by 4X
- Scaling a conv with this equation will increase the FLOPS by
  - Alpha * $Beta^2$ * $gamma^2$
  - Constrain this to be = 2 such that for any new phi, the total FLOPS will approximately increase by 2^{phi}
- Fix phi = 1, assume twice more resources available, and do a small grid search of alpha, beta, gamma
  - EfficientNet-B0: alpha = 1.2, Beta = 1.1, gamma =1.15,
  - Fix alpha, beta, gamma as constants and scale up baseline network with different phi to obtain EfficientNet-B1 to B7

Compound Model Scaling

- Balance dimensions of width/depth/resolution by scaling with a constant ratio
- Double width or resolution will increase FLOPS by 4X

$$\max_{d,w,r} \quad Accuracy(\mathcal{N}(d,w,r))$$

$$s.t. \quad \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\right)$$

$$\text{Memory}(\mathcal{N}) \le \text{target\_memory}$$

$$\text{FLOPS}(\mathcal{N}) \le \text{target\_flops} \qquad (2)$$

where w, d, r are coefficients for scaling network width, depth, and resolution; $\hat{\mathcal{F}}_i, \hat{L}_i, \hat{H}_i, \hat{W}_i, \hat{C}_i$ are predefined parameters in baseline network (see Table 1 as an example).

$$\text{depth: } d = \alpha^\phi$$

$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \ge 1, \beta \ge 1, \gamma \ge 1$$

Table 1. **EfficientNet-B0 baseline network** – Each row describes a stage i with $\hat{L}_i$ layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels $\hat{C}_i$. Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224 × 224	32	1
2	MBConv1, k3x3	112 × 112	16	1
3	MBConv6, k3x3	112 × 112	24	2
4	MBConv6, k5x5	56 × 56	40	2
5	MBConv6, k3x3	28 × 28	80	3
6	MBConv6, k5x5	28 × 28	112	3
7	MBConv6, k5x5	14 × 14	192	4
8	MBConv6, k3x3	7 × 7	320	1
9	Conv1x1 & Pooling & FC	7 × 7	1280	1

Table 3. **Scaling Up MobileNets and ResNet.**

Model	FLOPS	Top-1 Acc.
Baseline MobileNetV1 (Howard et al., 2017)	0.6B	70.6%
Scale MobileNetV1 by width (w=2)	2.2B	74.2%
Scale MobileNetV1 by resolution (r=2)	2.2B	72.7%
compound scale (d=1.4, w=1.2, r=1.3)	**2.3B**	**75.6%**
Baseline MobileNetV2 (Sandler et al., 2018)	0.3B	72.0%
Scale MobileNetV2 by depth (d=4)	1.2B	76.8%
Scale MobileNetV2 by width (w=2)	1.1B	76.4%
Scale MobileNetV2 by resolution (r=2)	1.2B	74.8%
MobileNetV2 compound scale	**1.3B**	**77.4%**
Baseline ResNet-50 (He et al., 2016)	4.1B	76.0%
Scale ResNet-50 by depth (d=4)	16.2B	78.1%
Scale ResNet-50 by width (w=2)	14.7B	77.7%
Scale ResNet-50 by resolution (r=2)	16.4B	77.5%
ResNet-50 compound scale	**16.7B**	**78.8%**

Table 4. **Inference Latency Comparison** – Latency is measured with batch size 1 on a single core of Intel Xeon CPU E5-2690.

	Acc. @ Latency		Acc. @ Latency
ResNet-152	77.8% @ 0.554s	GPipe	84.3% @ 19.0s
EfficientNet-B1	78.8% @ 0.098s	EfficientNet-B7	84.4% @ 3.1s
Speedup	**5.7x**	**Speedup**	**6.1x**

- EfficientNet-87
  - 84.4% top-1/97.1% top-5 ImageNet accuracy
  - 8.4x smaller and 6.1x faster on inference than the best existing cnn
- Transferring network
  - 91.% accuracy [Cifar-100]
    - State-of-the-art
  - 98.8% accuracy [Flowers dataset]

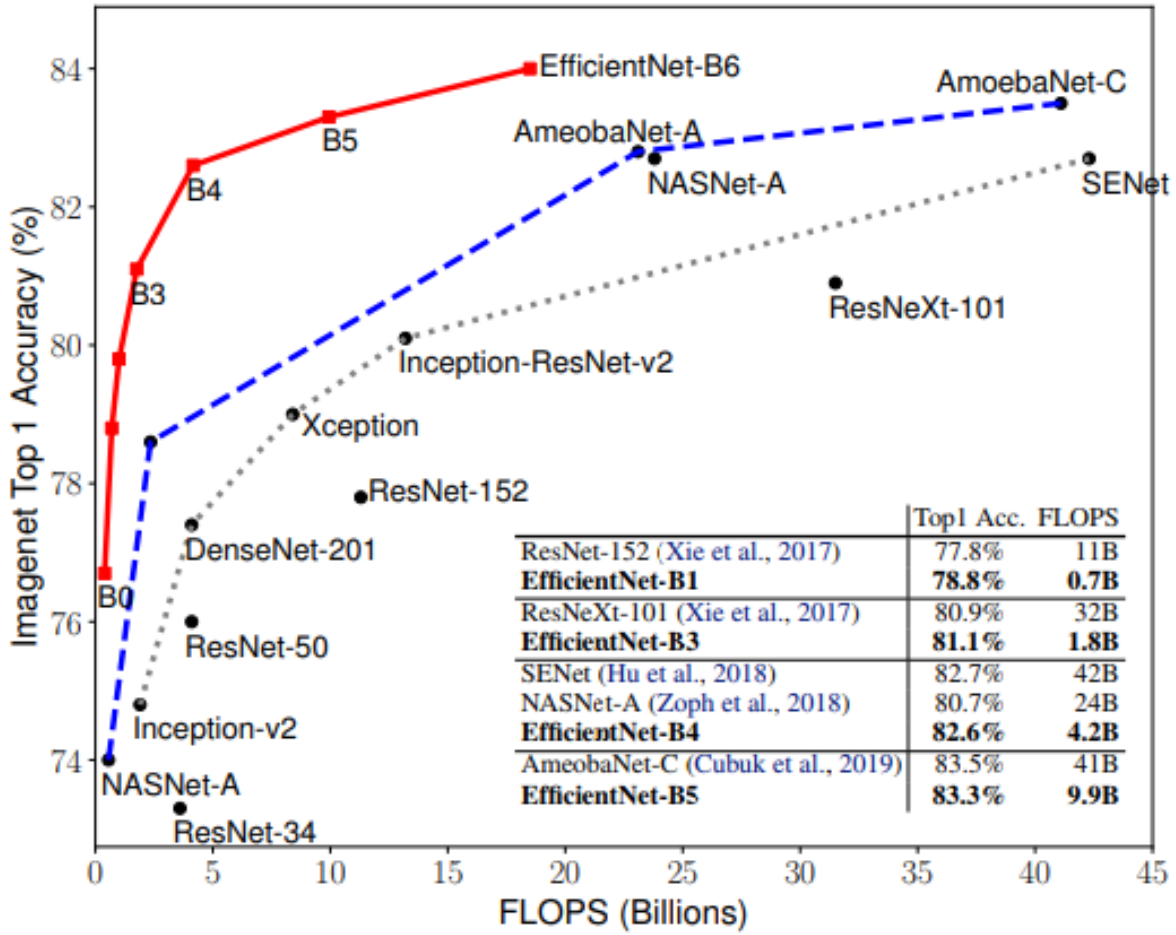
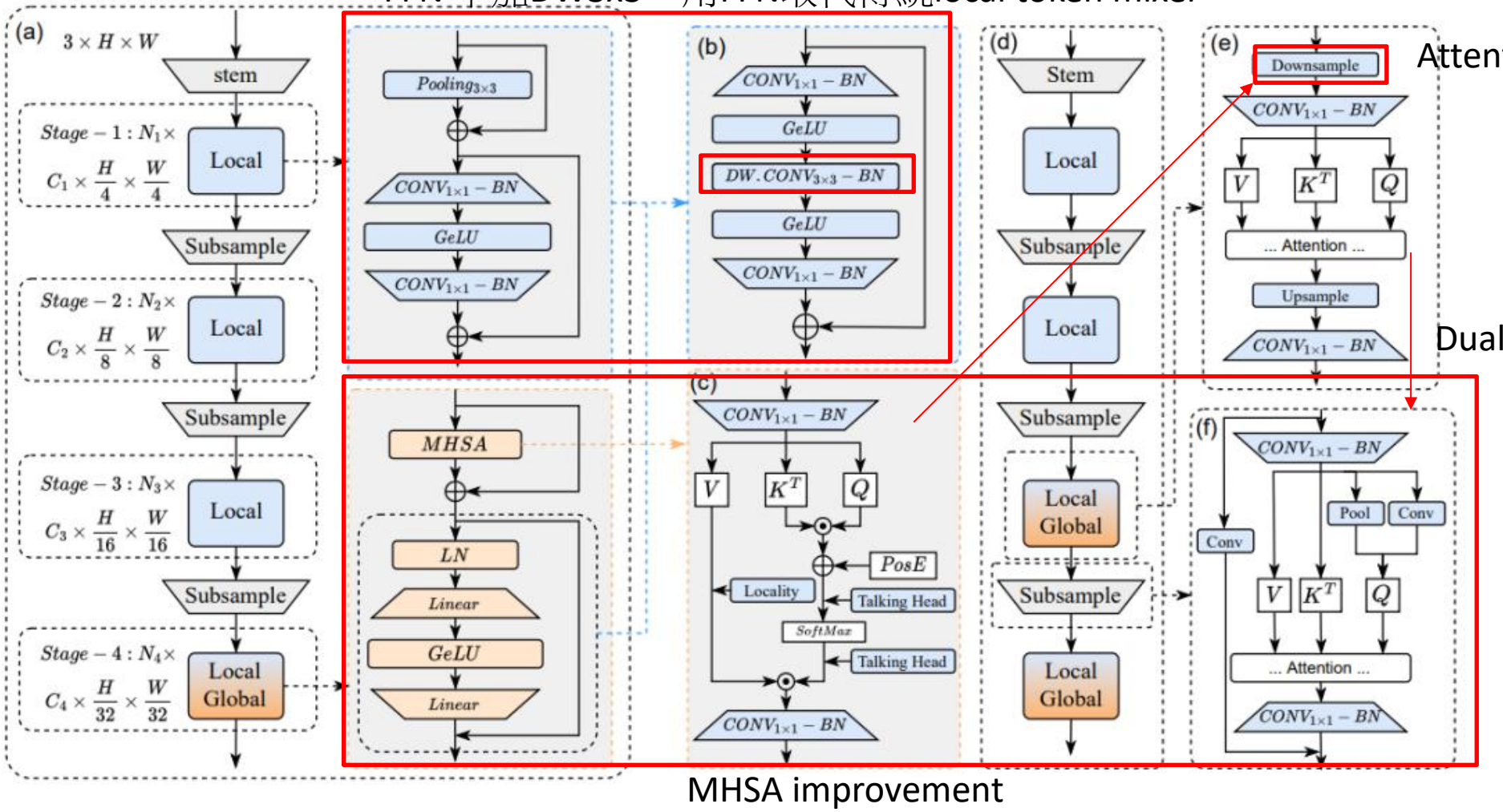


Figure 5. **FLOPS vs. ImageNet Accuracy** – Similar to Figure 1 except it compares FLOPS rather than model size.

EfficientFormerV2: Lightweight ViT

FFN 中加DW3x3，用FFN取代傳統local token mixer

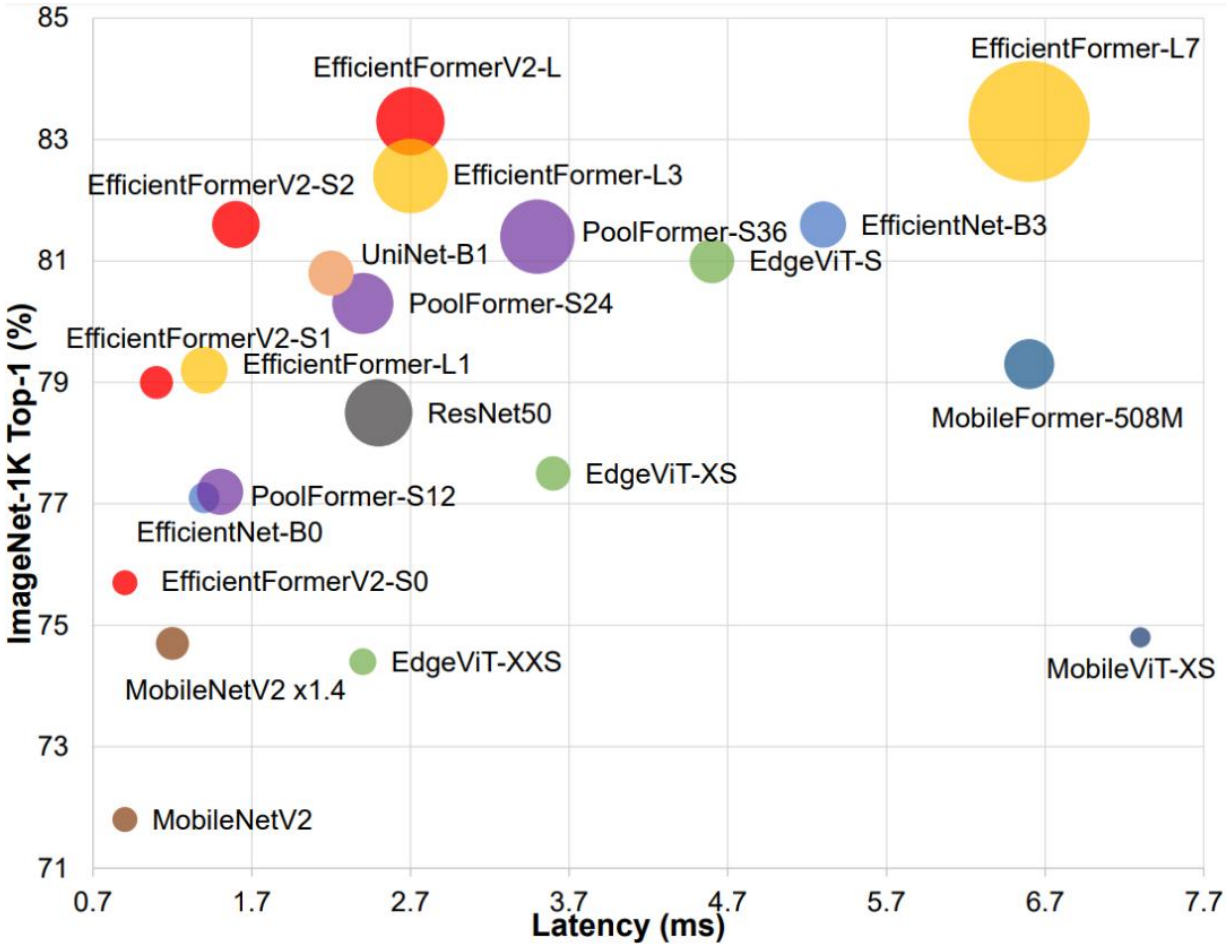


Attention at 1/32 scale

Dual path attention

MHSA improvement

Figure 2. **Network architectures.** We consider three metrics, *i.e.*, model performance, size, and inference speed, and study the models that improve any metric without hurting others. (a) Network of EfficientFormer [43] that serves as a baseline model. (b) Unified FFN (Sec. 3.1). (c) MHSA improvements (Sec. 3.3). (d)&(e) Attention on higher resolution (Sec. 3.4). (f) Attention downsampling (Sec. 3.5).


ImageNet-1K Top-1 (%)
85
83
81
79
77
75
73
71
Latency (ms)
0.7
1.7
2.7
3.7
4.7
5.7
6.7
7.7
EfficientFormerV2-L
EfficientFormer-L7
EfficientFormerV2-S2
EfficientFormer-L3
PoolFormer-S36
EfficientNet-B3
UniNet-B1
EdgeViT-S
PoolFormer-S24
EfficientFormerV2-S1
EfficientFormer-L1
MobileFormer-508M
ResNet50
EdgeViT-XS
PoolFormer-S12
EfficientNet-B0
EfficientFormerV2-S0
EdgeViT-XXS
MobileViT-XS
MobileNetV2 x1.4
MobileNetV2

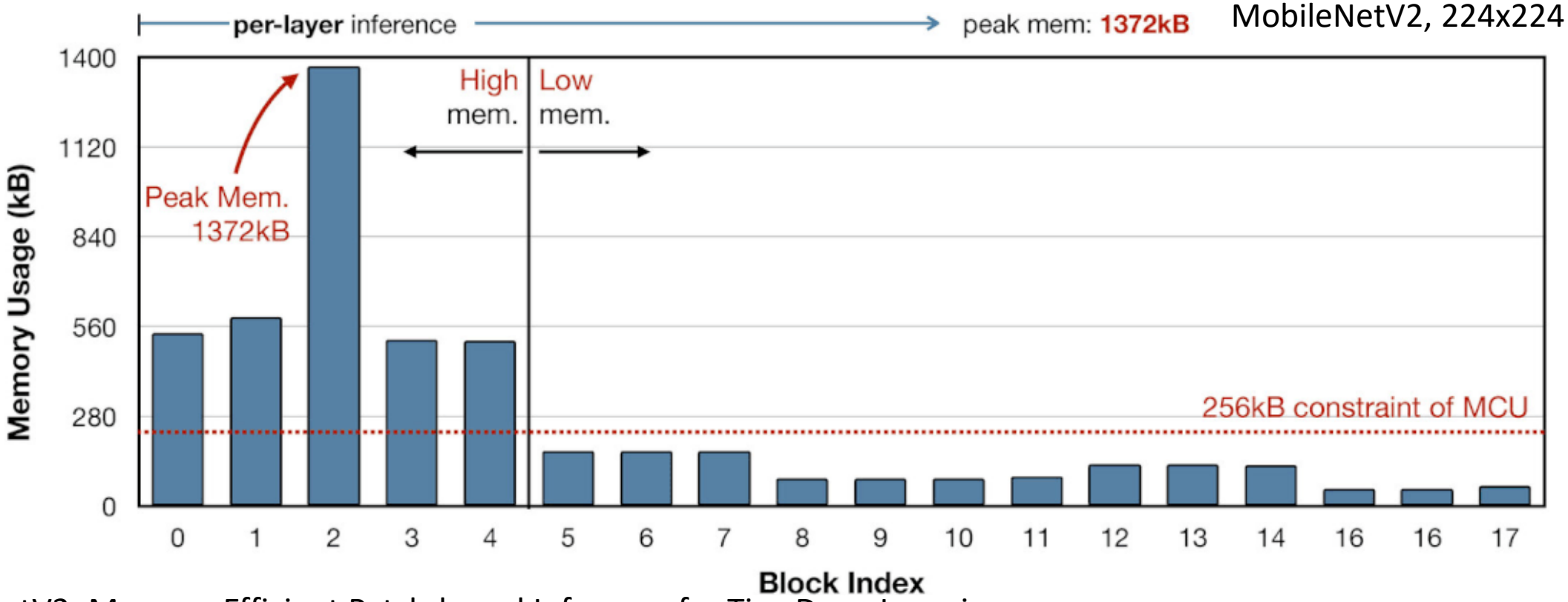
Section	Method	#Params (M)	MACs (G)	Latency (ms)	Top-1 (%)
(Baseline)	EfficientFormer-L1	12.25	1.30	1.4	79.2
Sec. 3.1	Pool Mixer → $\text{DWCONV}_{3\times3}$	12.27	1.30	1.4	79.8
	✓ Feed Forward Network	12.37	1.33	1.4	80.3
Sec. 3.2	✓ Vary Depth and Width	12.24	1.20	1.3	80.5
	5-Stage Network	12.63	1.08	1.5	80.3
Sec. 3.3	✓ Locality in V & Talking Head	12.25	1.21	1.3	80.8
Sec. 3.4	Attention at Higher Resolution	13.10	1.48	3.5	81.7
	✓ Stride Attention	13.10	1.31	1.5	81.5
Sec. 3.5	✓ Attention Downsampling	13.40	1.35	1.6	81.8

TINY ML

MCUNet v2

- TinyDL on MCU: Memory bottleneck
  - Goal: fit run time memory within MCU on-chip memory
  - Imbalanced memory distribution

MobileNetV2, 224x224 input

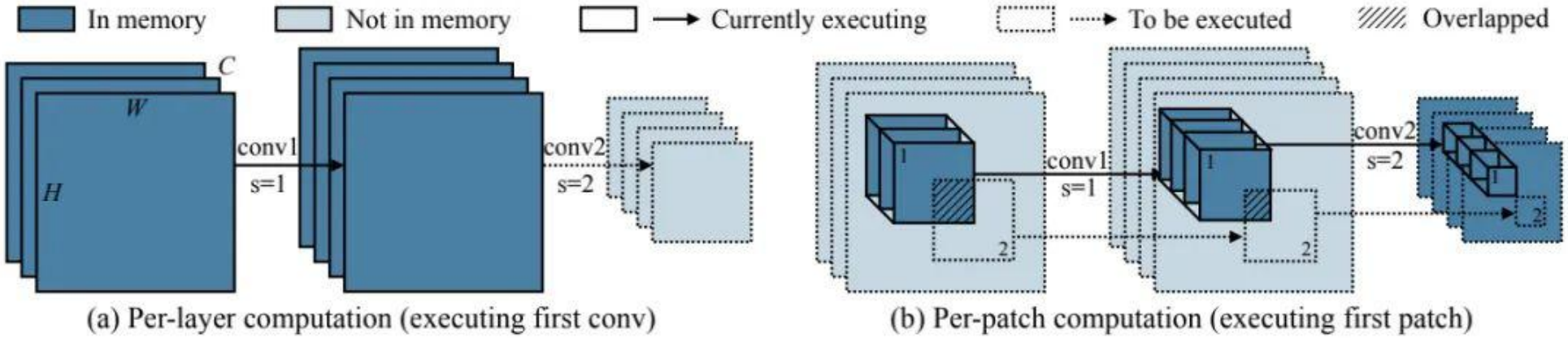


MCUNetV2: Memory-Efficient Patch-based Inference for Tiny Deep Learning

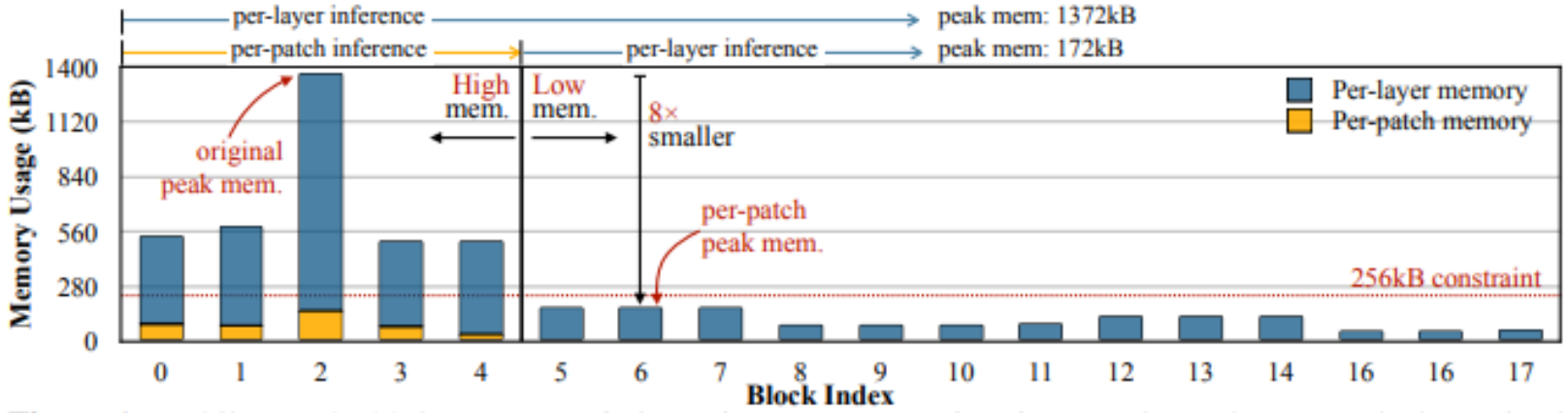
MCUNet v2

- Per layer => per patch inference

10~17% computation

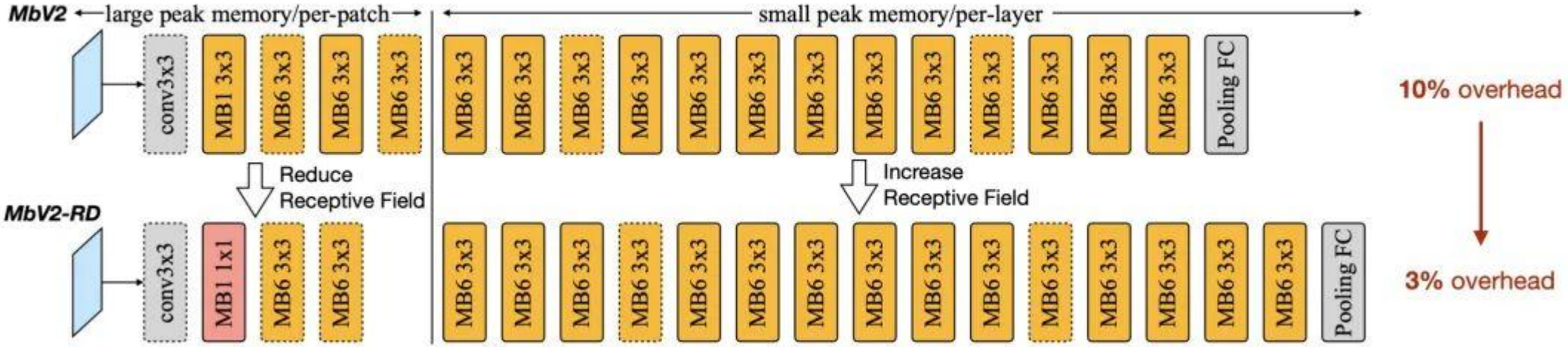


在memory密集階段以patch-by-patch　　其他採用常規的layer-by-layer



MCUNet v2

- Redistributing the Receptive Field



降低patch部分的輸入尺寸與重複計算

提升layer部分的感受野以補償性能損失

Table 1. Per-patch inference reduces the peak memory by 8× for MobileNetV2 [44] (1372kB to 172kB), but it increases the overall computation by 10% due to patch overlapping. We futher propose receptive field redistribution (MbV2-RD) which reduces the overall overhead to only 3% without hurting performance.

Model	Patch Size	Comp. overhead		MACs$_{(4\times4\text{ patches})}$		Peak SRAM		ImgNet Top-1	VOC mAP
		patch-stage	overall	patch-stage	overall	per-layer	per-patch		
MbV2 [44]	75^2	+42%	+10%	130M	330M	1372kB	172kB (**8**×↓)	72.2%	75.4%
MbV2-RD	63^2	+18%	+3%	73M	301M	1372kB	172kB (**8**×↓)	72.1%	75.7%

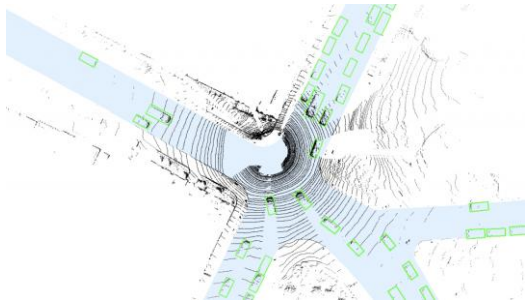
Table 3. MCUNetV2 significantly improves Pascal VOC [13] object detection on MCU by allowing a higher input resolution. Under STM32H743 MCU constraints, MCUNetV2-H7 improves the mAP by 16.9% compared to [30], achieving a record performance on MCU. It can also scale down to cheaper MCU STM32F412 with only 256kB SRAM while still improving mAP by 13.2% at 1.9× smaller peak SRAM and a similar computation.

MCU Model	Constraint	Model	#Param	MACs	peak SRAM	VOC mAP	Gain
H743 (~$7)	SRAM <512kB	MbV2+CMSIS [30]	0.87M	34M	~~519kB~~	31.6%	-
		MCUNet [30]	1.20M	168M	466kB	51.4%	0%
		MCUNetV2-H7	0.67M	343M	438kB	**68.3%**	+16.9%
F412 (~$4)	<256kB	MCUNetV2-M4	0.47M	172M	**247kB**	64.6%	+13.2%

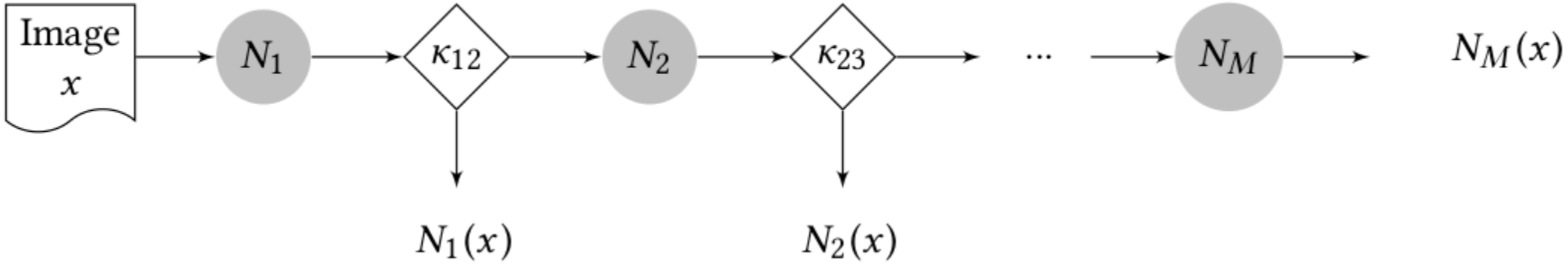
DYNAMIC EXECUTION BASED ON INPUT

Concept

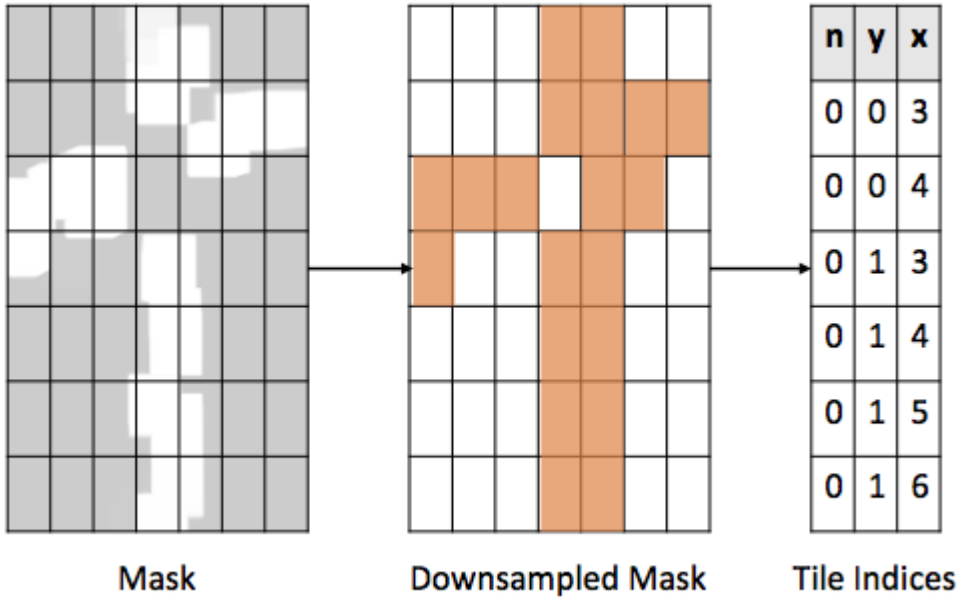
- Skip invalid input and its computations
  - LiDAR for traffic detection
  - Skip objects outside the road

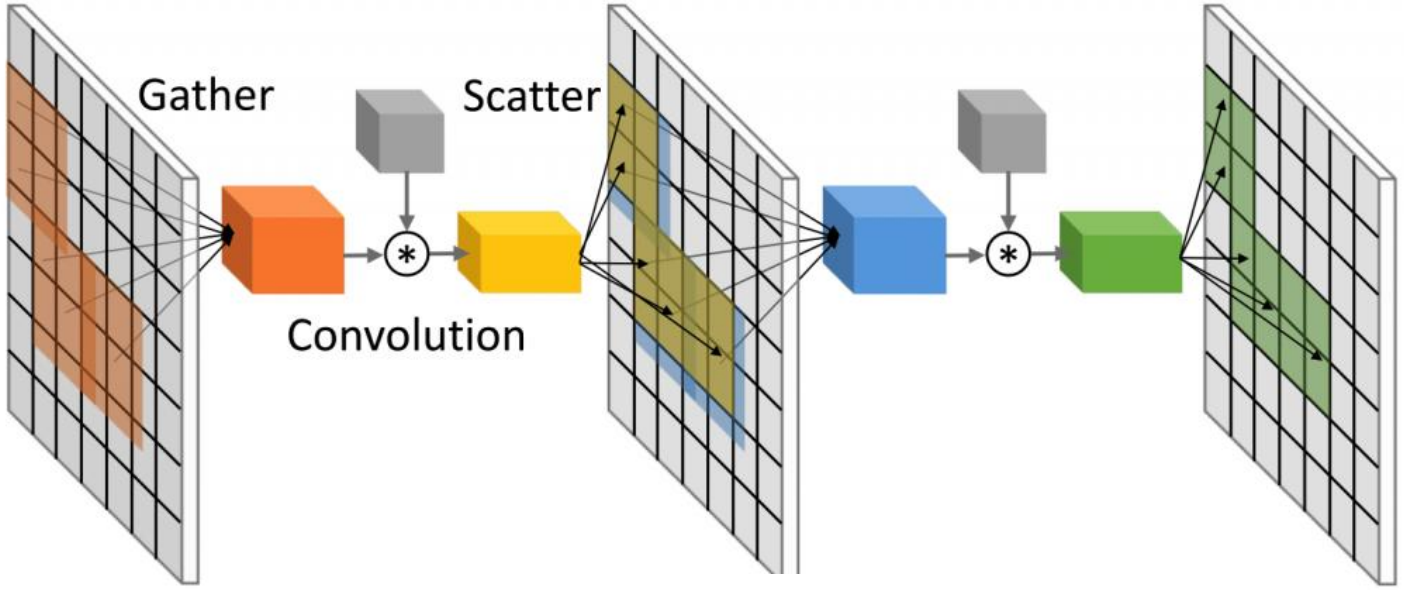


- Not all input are created equal
  - Easy input just needs simpler architecture



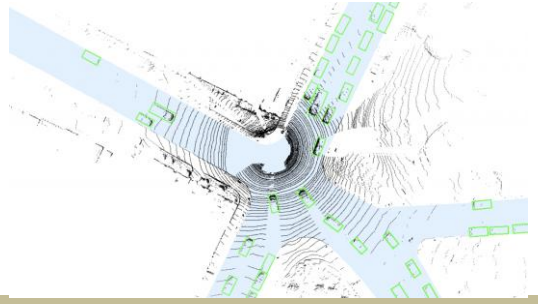
SBNet

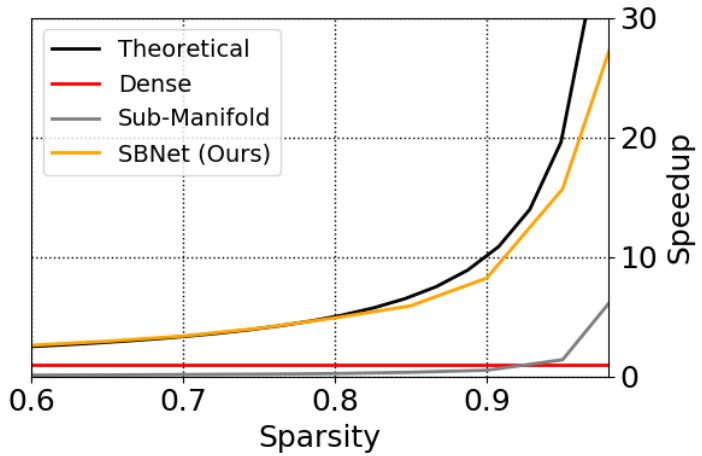




To exploit sparsity in the activations of CNNs, SBNet first converts a computation mask to a tile index list.

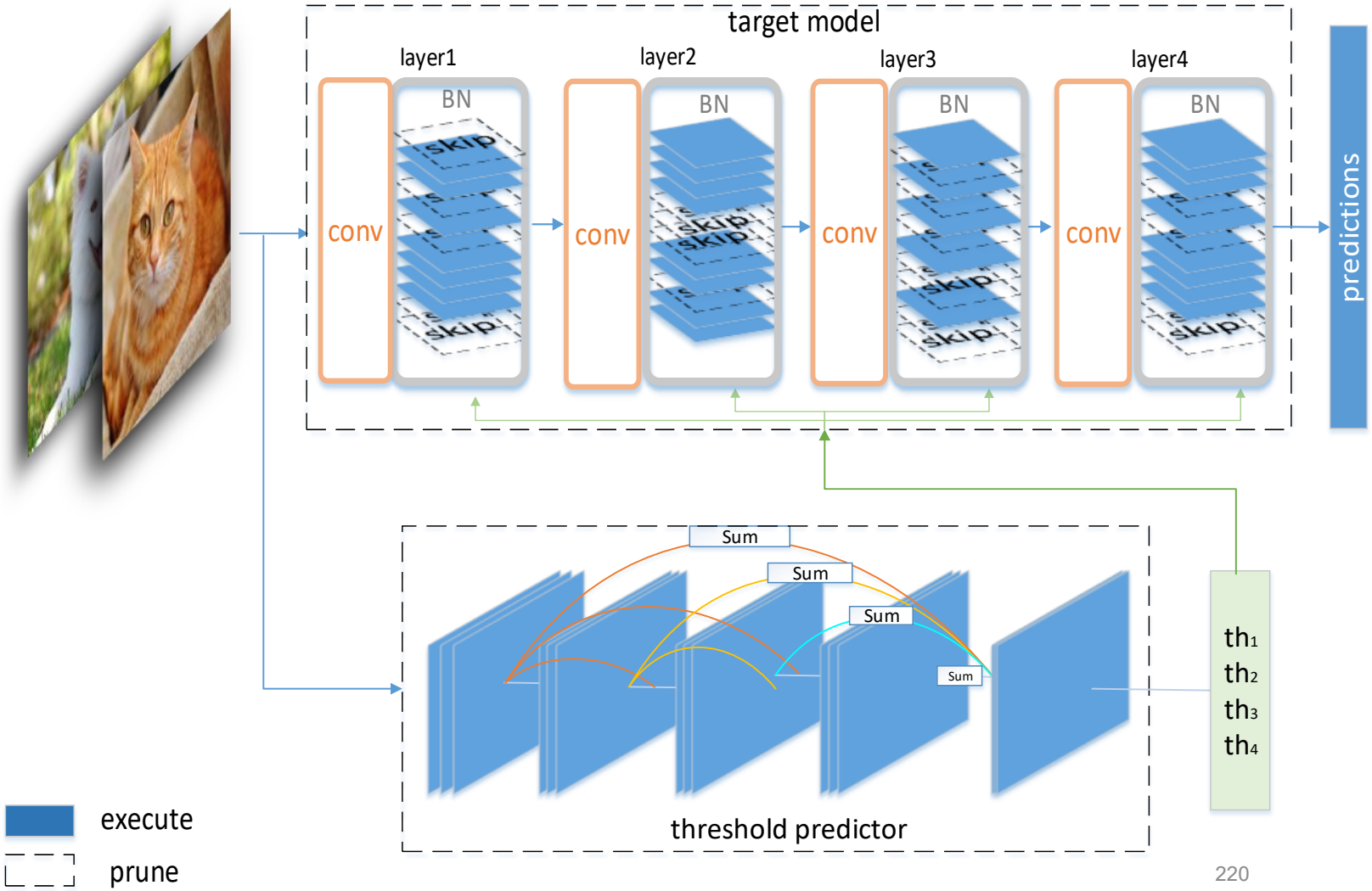
Gather along batch dim



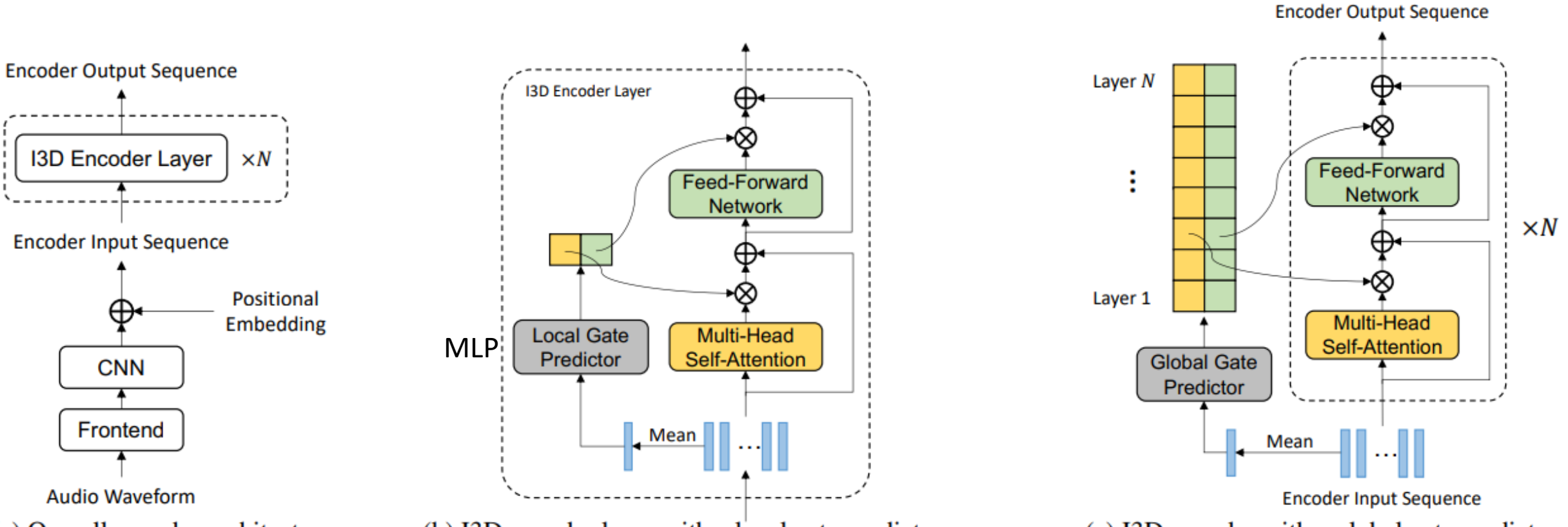


Dynamic Network Slimming

- Structure Overview
  - Predictor outputs a group of thresholds with input images
  - The target model is pruned by the group of thresholds



I3D: TRANSFORMER ARCHITECTURES WITH INPUT-DEPENDENT DYNAMIC DEPTH FOR SPEECH RECOGNITION



(a) Overall encoder architecture. (b) I3D encoder layer with a local gate predictor. (c) I3D encoder with a global gate predictor.

Fig. 1: Architectures of our proposed I3D encoders.

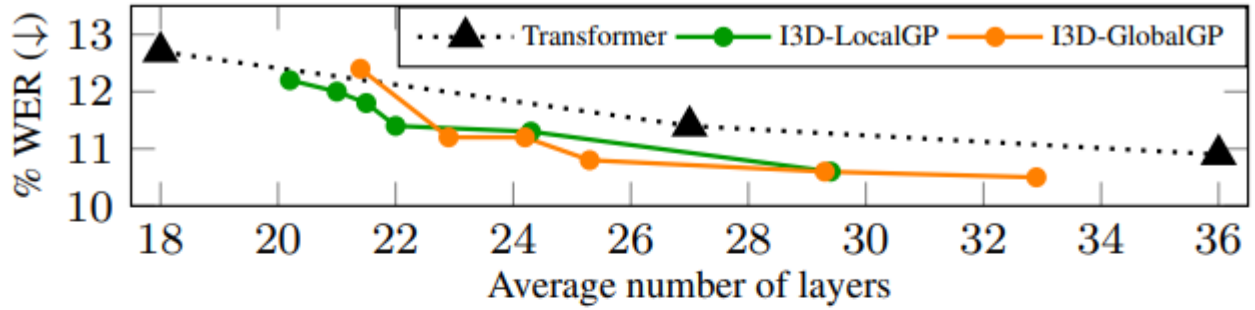


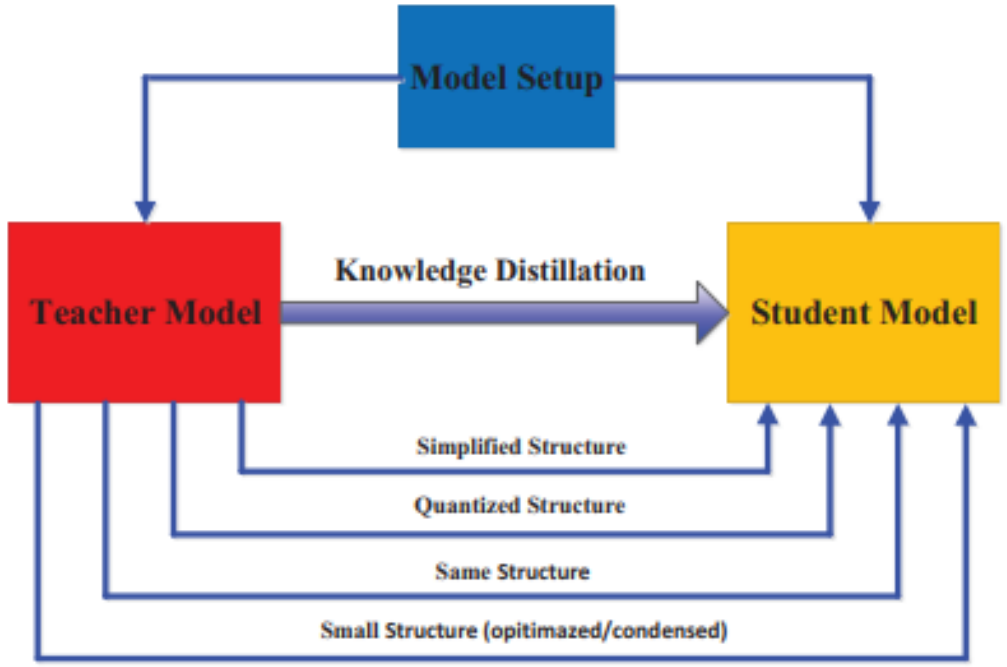
Fig. 4: Word error rates (%) of **CTC**-based models vs. average number of layers used for inference on the **Tedlium2** test set.

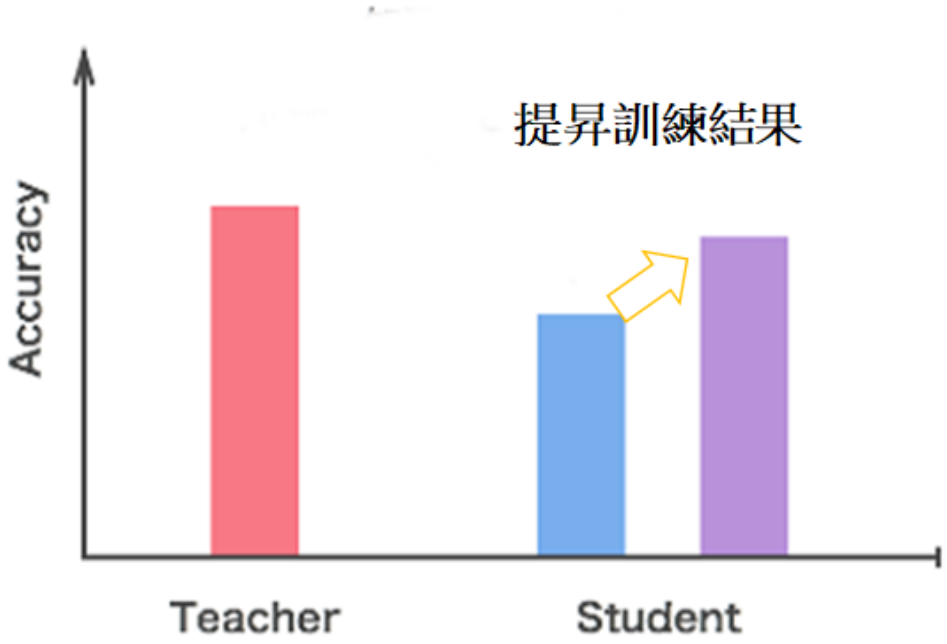
Table 1: Word error rates (%) and average number of inference layers of **AED**-based models on **LibriSpeech 100h**.

Model	dev clean		test clean	
	Ave #layers	WER (↓)	Ave #layers	WER (↓)
Transformer	36	7.8	36	8.0
	27	8.2	27	8.5
I3D-LGP-36	27.3	7.9	27.1	8.3
I3D-GGP-36	27.2	7.8	27.1	8.2

KNOWLEDGE DISTILLATION

Knowledge Distillation





Knowledge distillation: a survey. You J, Yu B, Maybank SJ, Tao D

https://chtseng.wordpress.com/2020/05/12/%E7%9F%A5%E8%AD%98%E8%92%B8%E9%A4%BE-knowledgedistillation/

Knowledge Distillation

Teacher Net (Large)

Student Net (Small)

Learning target

Hard target

cow	dog	cat	car
0	1	0	0

Cross-entropy minimization

?

Soft target

cow	dog	cat	car
10^{-6}	.9	.1	10^{-9}

用機率分佈，而非單純的對與不對，提供了更多的隱含資訊

Knowledge Distillation

https://arxiv.org/pdf/1503.02531.pdf

Knowledge Distillation

Ensemble

Average many models

N Networks

Student Net (Small)

Learning target

cow	dog	cat	car
10^{-6}	.9	.1	10^{-9}

Cross-entropy minimization

?

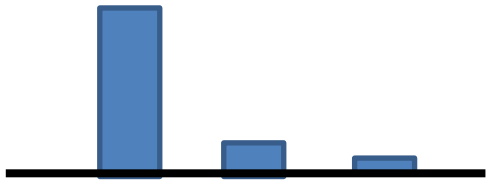
Knowledge Distillation

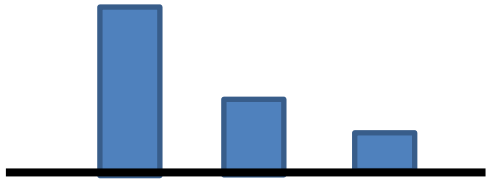
https://arxiv.org/pdf/1503.02531.pdf

Knowledge Distillation

- 分類的softmax output: 正確類別機率=>1, 其他 =>0
- Temperature for softmax: softenen probability (T=3, in general)

$$y_i' = \frac{exp(y_i)}{\sum_j exp(y_j)} \quad \xrightarrow{T = 100} \quad y_i' = \frac{exp(y_i/T)}{\sum_j exp(y_j/T)}$$





cow	dog	cat	car
10^{-6}	.9	.1	10^{-9}

cow	dog	cat	car
.05	.3	.2	.005

$y_1 = 100 \quad y_1' = 1$

$y_2 = 10 \quad y_2' \approx 0$

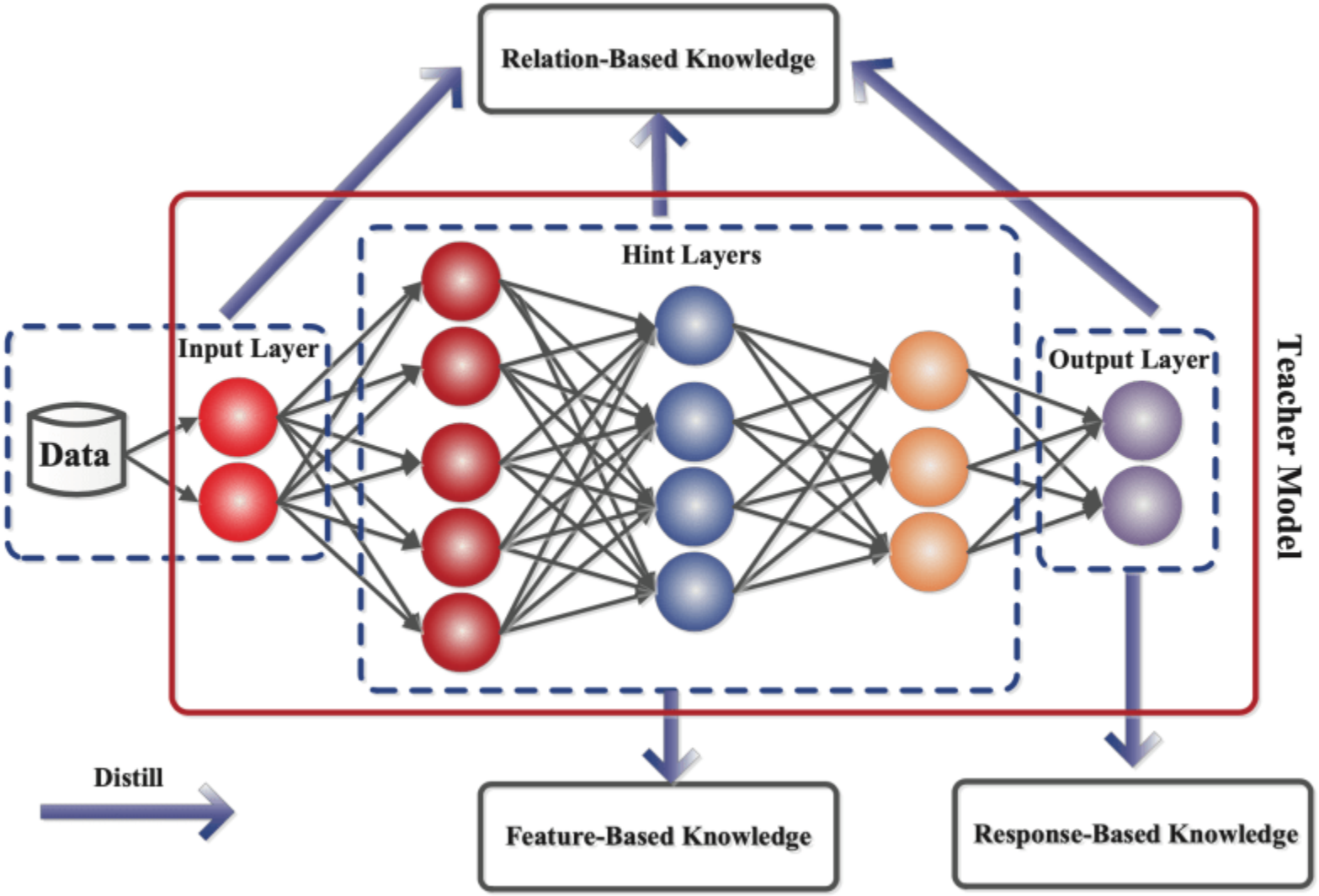
$y_3 = 1 \quad y_3' \approx 0$

$y_1/T = 1 \quad y_1' = 0.56$

$y_2/T = 0.1 \quad y_2' = 0.23$

$y_3/T = 0.01 \quad y_3' = 0.21$

The different kinds of knowledge in a teacher model



Knowledge distillation: a survey. You J, Yu B, Maybank SJ, Tao D

Conclusion

- Model size/FLOPS != Model execution time
  - Beware of hardware and software library support
- Trend
  - Resource constrained model pruning + low complexity model
  - Pruning as one type of architecture search (expected 3X~10X reduction)
  - Train large model and then compress
- Quantization is the first to try (32bit -> 8bit, 4X reduction)
- Dynamic execution is orthogonal to others
- Output class reduction => pruning for this
  - Need fewer features, and thus fewer parameters
- Fine-tuning is your best friend to restore accuracy