

Modern CNN

More Accurate or More Efficient



ResNeXt

Aggregated Residual Transformations for Deep Neural Networks

Saining Xie¹

Ross Girshick²

Piotr Dollár²

Zhuowen Tu¹

Kaiming He²

¹UC San Diego

²Facebook AI Research

{s9xie, ztu}@ucsd.edu

{rbg, pdollar, kaiminghe}@fb.com

Abstract

We present a simple, highly modularized network architecture for image classification. Our network is constructed by repeating a building block that aggregates a set of transformations with the same topology. Our simple design results in a homogeneous, multi-branch architecture that has only a few hyper-parameters to set. This strategy exposes a new dimension, which we call “cardinality” (the size of the set of transformations), as an essential factor in addition to the dimensions of depth and width. On the ImageNet-1K dataset, we empirically show that even under the restricted condition of maintaining complexity, increasing cardinality is able to improve classification accuracy. Moreover, in

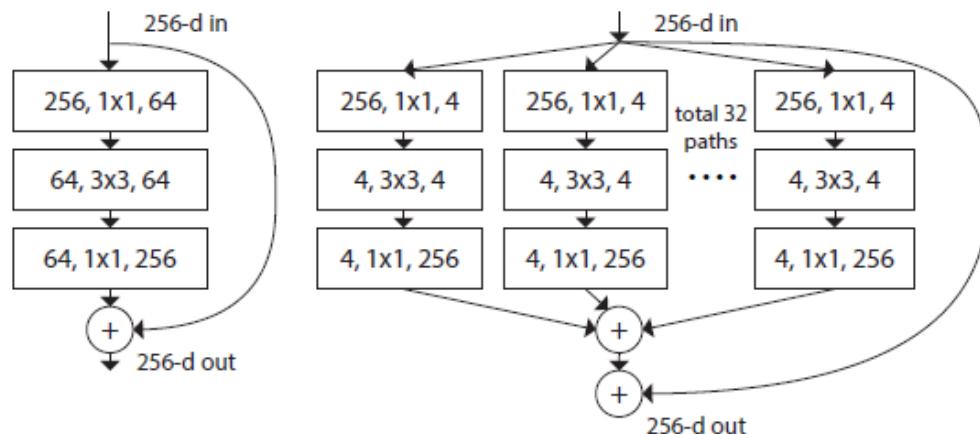


Figure 1. Left: A block of ResNet [14]. Right: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ImageNet 2016 Results

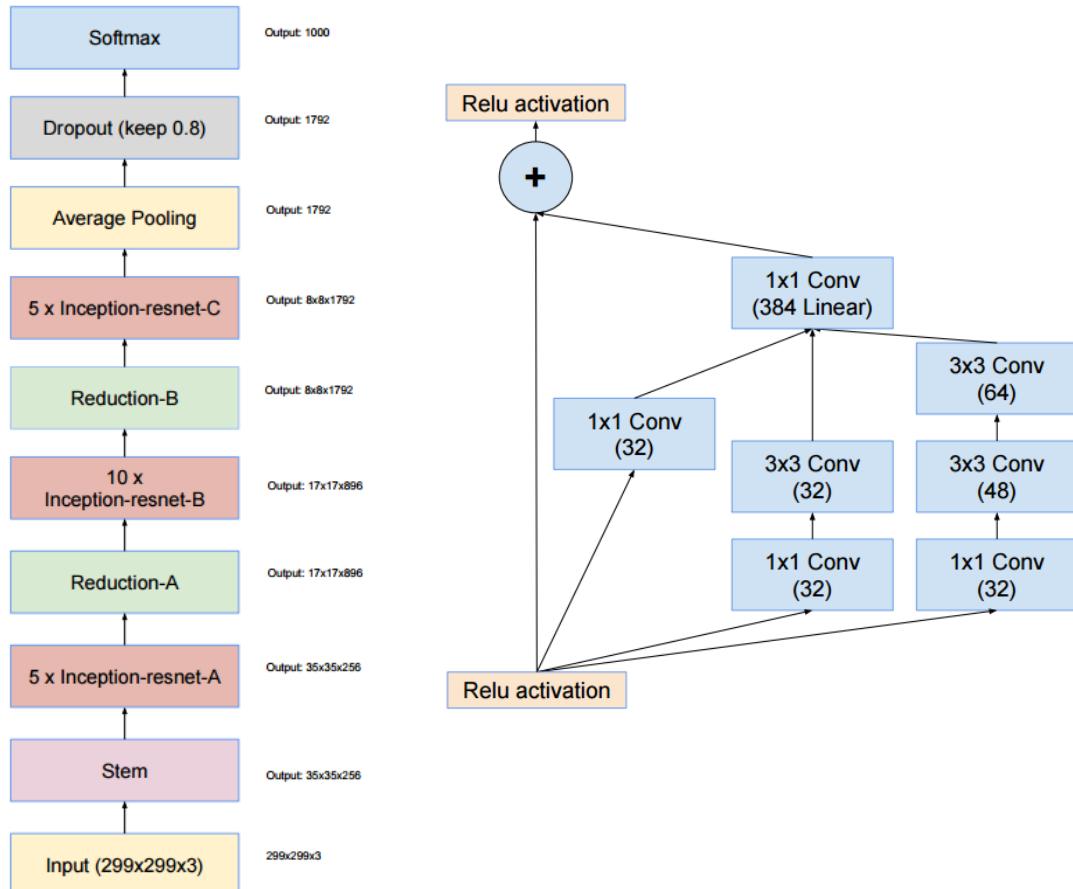
Team name	Entry description	Classification error	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink	GrandUnion + Fused-scale EnsembleNet	0.03042	0.098892
CU-DeepLink	GrandUnion + Multi-scale EnsembleNet	0.03046	0.099006
CU-DeepLink	GrandUnion + Basic Ensemble	0.03049	0.098954
ResNeXt	Ensemble B, weighted average, tuned on val. [No bounding box results]	0.03092	0.737484
CU-DeepLink	GrandUnion + Class-reweighted Ensemble	0.03096	0.099369
CU-DeepLink	GrandUnion + Class-reweighted Ensemble with Per-instance Normalization	0.03103	0.099349
ResNeXt	Ensemble C, weighted average. [No bounding box results]	0.03124	0.737526
Trimps-Soushen	Ensemble 1	0.03144	0.079068
ResNeXt	Ensemble A, simple average. [No bounding box results]	0.0315	0.737505
SamExynos	3 model only for classification	0.03171	0.236561
ResNeXt	Ensemble B, weighted average. [No bounding box results]	0.03203	0.737681
KAISTNIA_ETRI	Ensembles A	0.03256	0.102015
KAISTNIA_ETRI	Ensembles C	0.03256	0.102056
KAISTNIA_ETRI	Ensembles B	0.03256	0.100676

Growing Number of Hyper-parameters

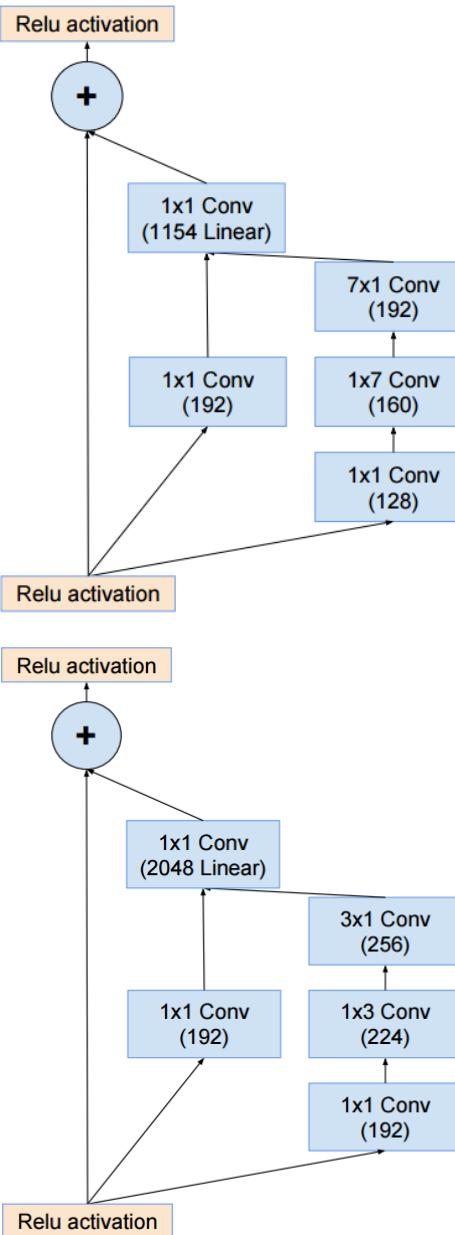
- VGGNet exhibit a simple yet effective strategy of constructing very deep network – **stacking building blocks of the same shape**
- ResNet inherit this strategy with **stacking modules of the same topology**
- Unlike VGGNet, the family of Inception models have demonstrated that carefully designed topologies are able to achieve compelling accuracy
 - Important common property is split-transform-merge strategy
 - Split – 1×1 conv, transform – 3×3 , 5×5 conv, merge - concatenation

Inception Learns ResNet

- Inception + ResNet

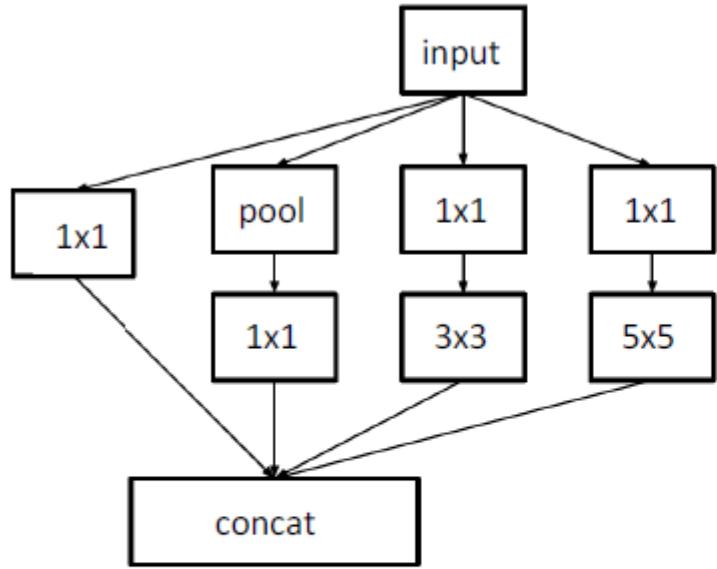


"Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning"



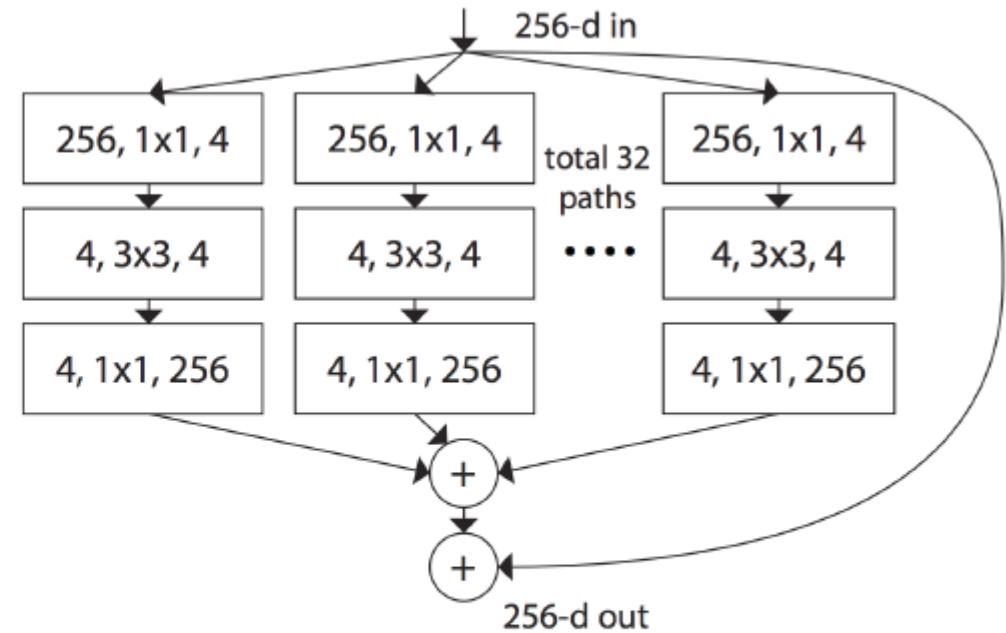
ResNet Learns Inception??

- Multi-branch + ResNet



Inception:

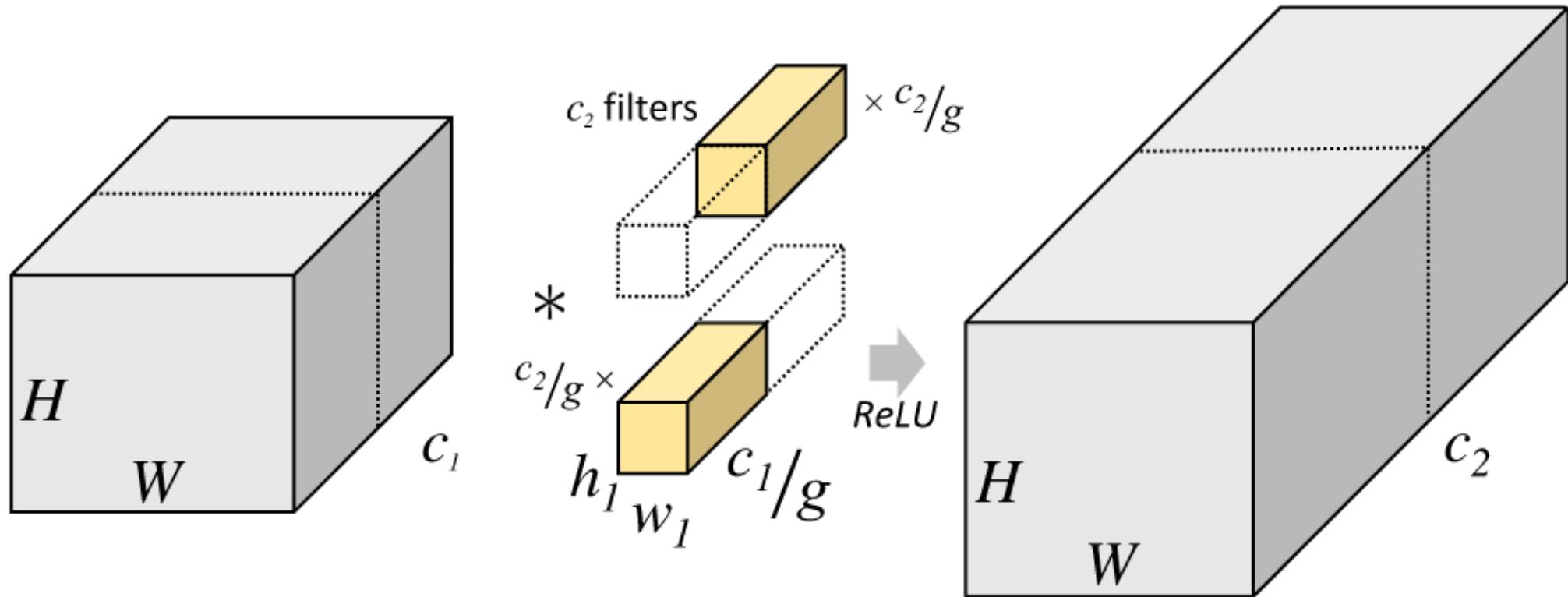
heterogeneous multi-branch



ResNeXt:

uniform multi-branch

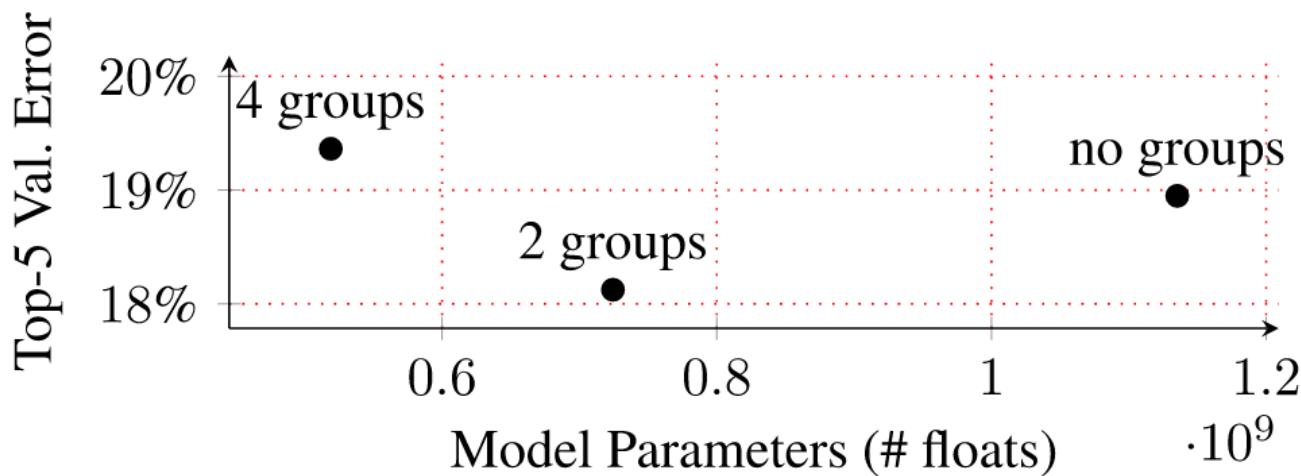
Group Convolution



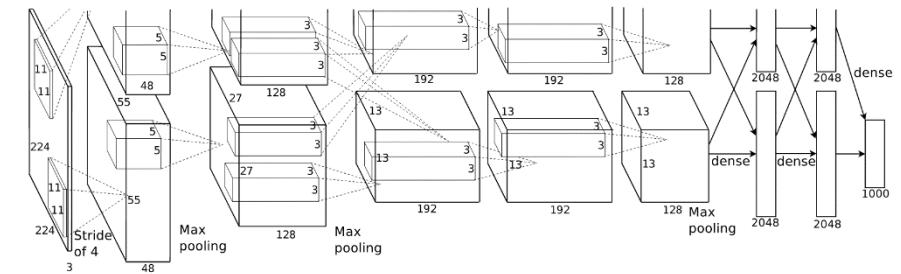
A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

Grouped Convolution of AlexNet

- AlexNet's primary motivation was to allow the training of the network over two Nvidia GTX580 GPUs with 1.5GB of memory each
- AlexNet without filter groups is not only less efficient(both in parameters and compute), but also slightly less accurate!



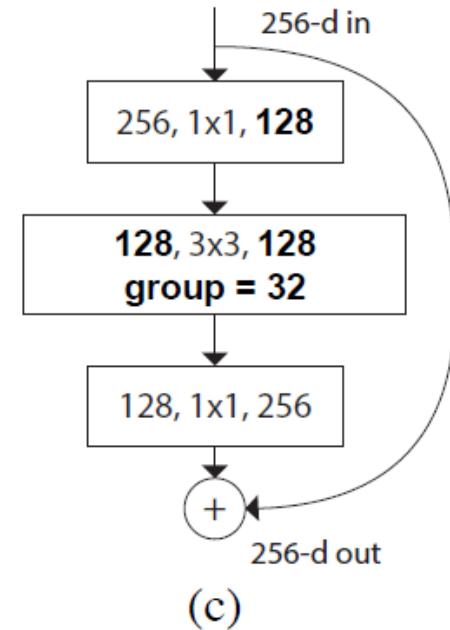
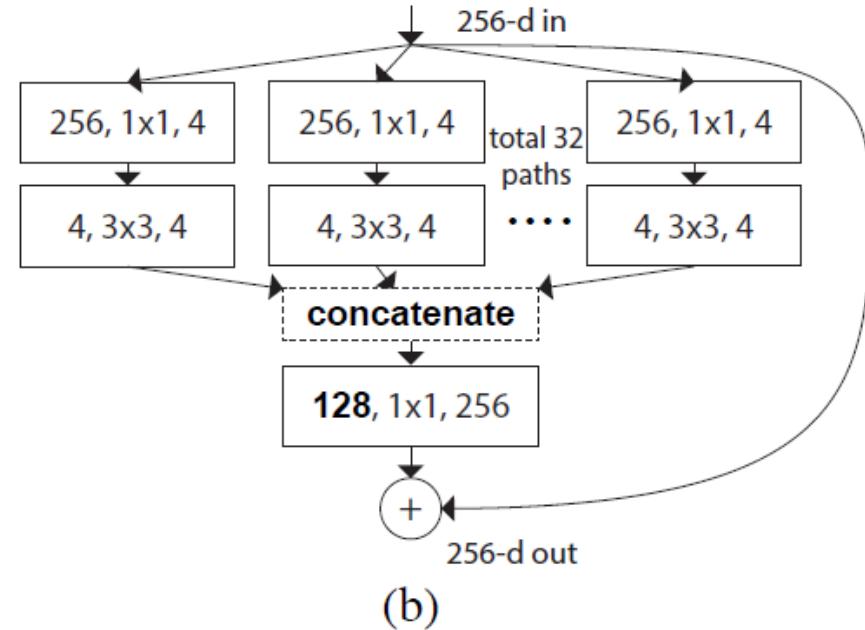
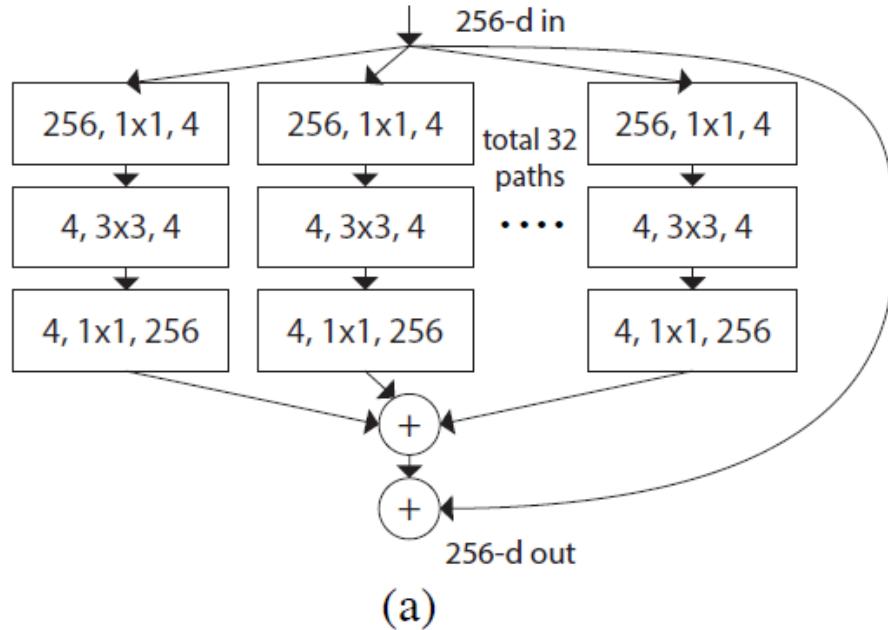
AlexNet trained with varying numbers of filter groups, from 1 (i.e. no filter groups), to 4. When trained with 2 filter groups, AlexNet is more efficient and yet achieves the same if not lower validation error.



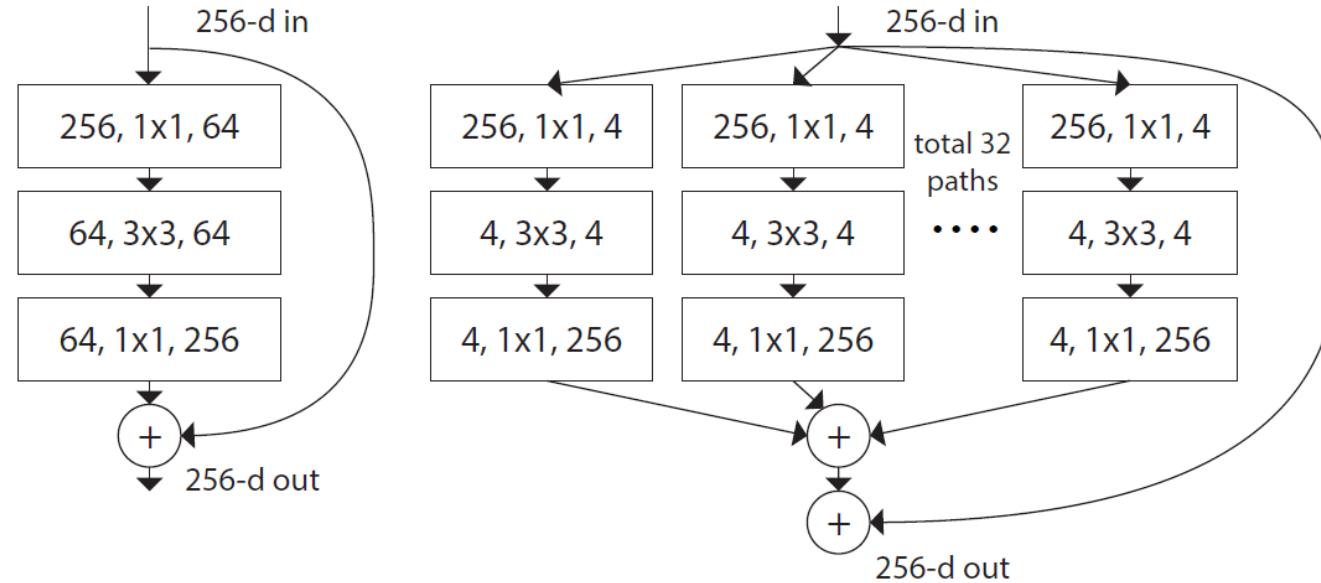
ResNeXt

- Concatenation and Addition are interchangeable
- Uniform multi-branching can be done by group-conv

equivalent

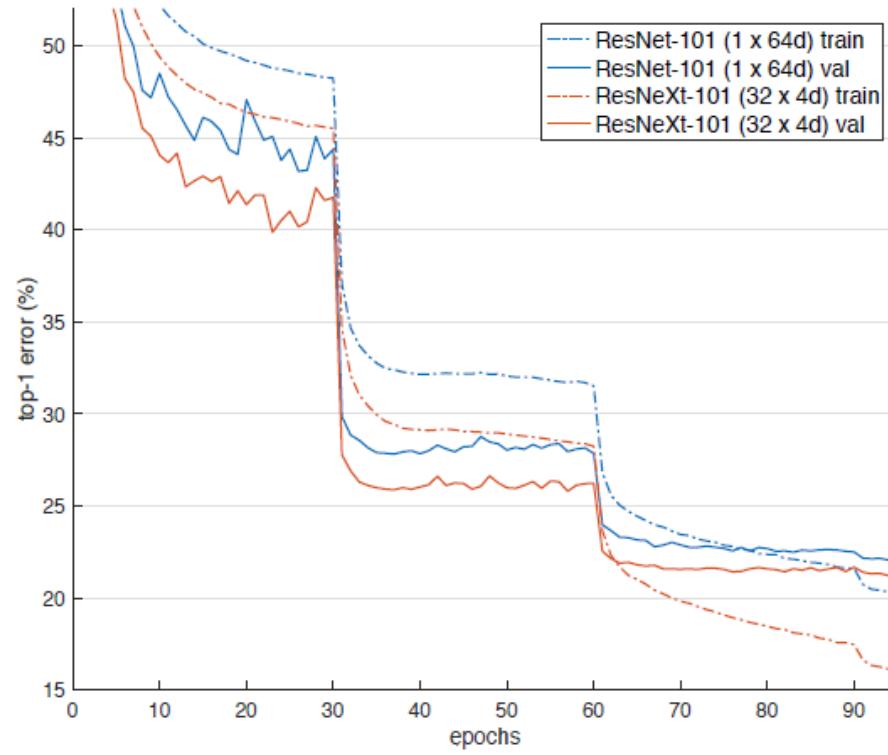
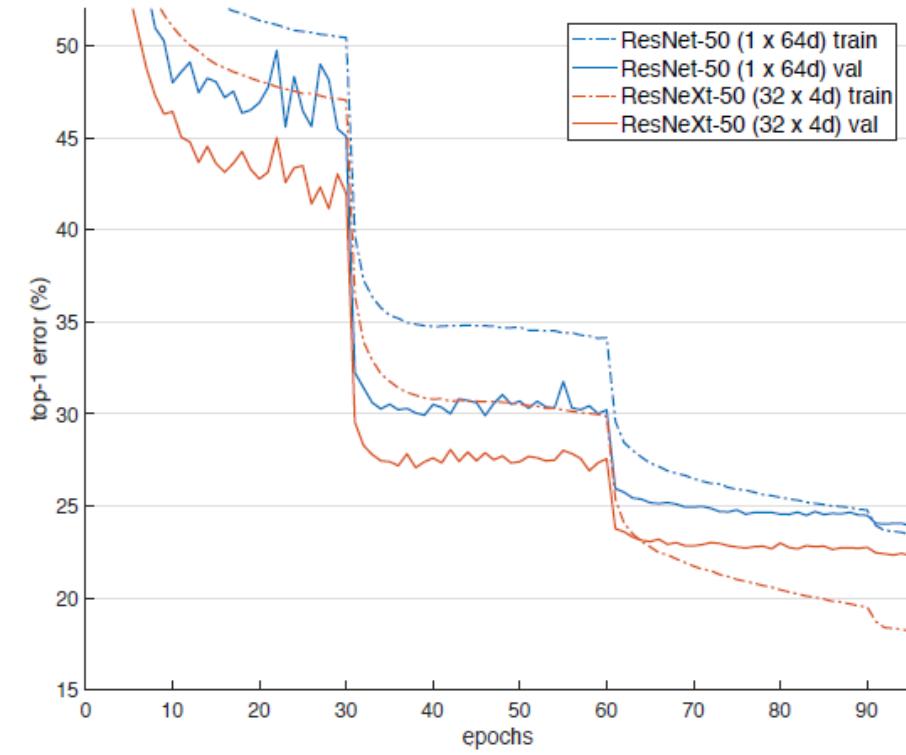


Model Capacity (# of parameters)



- Original ResNet(left) : $256 \times 64 + 3 \times 3 \times 64 \times 64 + 64 \times 256 = 70k$
- ResNeXt(right – with bottleneck width d and cardinality C) :
 $C \times (256 \times d + 3 \times 3 \times d \times d + d \times 256) = 70k$, when $C = 32$ and $d = 4$

Results – Cardinality vs Width



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

Results – Increasing Cardinality vs Deeper/Wider

	setting	top-1 err (%)	top-5 err (%)
<i>1 × complexity references:</i>			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
<i>2 × complexity models follow:</i>			
ResNet- 200 [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × 100d	21.3	5.7
ResNeXt-101	2 × 64d	20.7	5.5
ResNeXt-101	64 × 4d	20.4	5.3

Squeeze-and-Excitation Networks

Squeeze-and-Excitation Networks

Jie Hu^[0000–0002–5150–1003] Li Shen^[0000–0002–2283–4976] Samuel Albanie^[0000–0001–9736–5134]
Gang Sun^[0000–0001–6913–6799] Enhua Wu^[0000–0002–2174–1428]

Abstract—The central building block of convolutional neural networks (CNNs) is the convolution operator, which enables networks to construct informative features by fusing both spatial and channel-wise information within local receptive fields at each layer. A broad range of prior research has investigated the spatial component of this relationship, seeking to strengthen the representational power of a CNN by enhancing the quality of spatial encodings throughout its feature hierarchy. In this work, we focus instead on the channel relationship and propose a novel architectural unit, which we term the “Squeeze-and-Excitation” (SE) block, that adaptively recalibrates channel-wise feature responses by explicitly modelling interdependencies between channels. We show that these blocks can be stacked together to form SENet architectures that generalise extremely effectively across different datasets. We further demonstrate that SE blocks bring significant improvements in performance for existing state-of-the-art CNNs at minimal additional computational cost. Squeeze-and-Excitation Networks formed the foundation of our ILSVRC 2017 classification submission which won first place and reduced the top-5 error to 2.251%, surpassing the winning entry of 2016 by a relative improvement of ~25%. Models and code are available at <https://github.com/hujie-frank/SENet>.

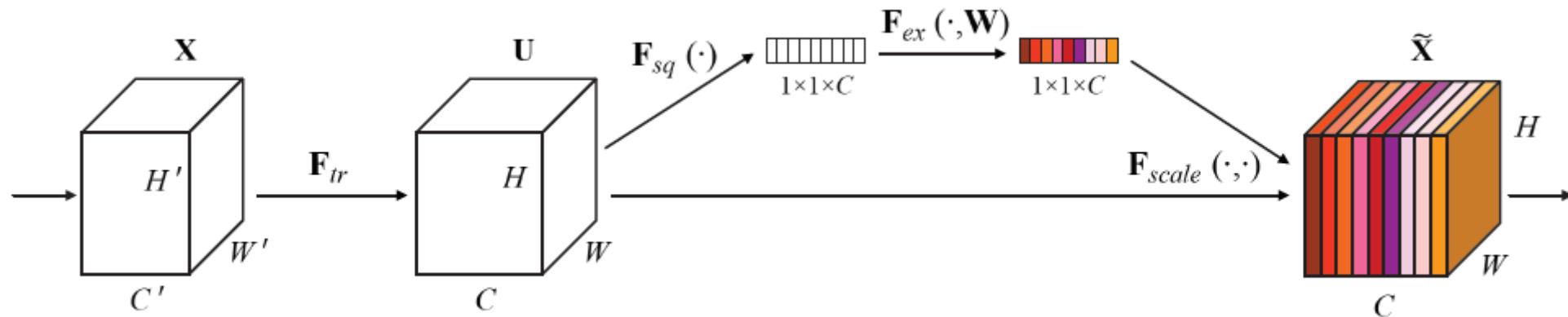
Index Terms—Squeeze-and-Excitation, Image classification, Convolutional Neural Network.

Squeeze-and-Excitation Networks

Team name	Entry description	Classification error	Localization error
WMW	Ensemble C [No bounding box results]	0.02251	0.590987
WMW	Ensemble E [No bounding box results]	0.02258	0.591018
WMW	Ensemble A [No bounding box results]	0.0227	0.591153
WMW	Ensemble D [No bounding box results]	0.0227	0.591039
WMW	Ensemble B [No bounding box results]	0.0227	0.59106
Trmps-Soushen	Result-1	0.02481	0.067698
Trmps-Soushen	Result-2	0.02481	0.06525
Trmps-Soushen	Result-3	0.02481	0.064991
Trmps-Soushen	Result-4	0.02481	0.065261
Trmps-Soushen	Result-5	0.02481	0.065302
NUS-Qihoo_DPNs (CLS-LOC)	[E2] CLS:: Dual Path Networks + Basic Ensemble	0.0274	0.088093
NUS-Qihoo_DPNs (CLS-LOC)	[E1] CLS:: Dual Path Networks + Basic Ensemble	0.02744	0.088269
BDAT	provide_class	0.02962	0.086942
BDAT	provide_box	0.03158	0.081392
MIL_UT	Ensemble of 9 models (classification-only)	0.03205	0.596164
SIIT_KAIST-SKT	ensemble 2	0.03226	0.128924
MIL_UT	Ensemble of 10 models (classification-only)	0.03228	0.596174

Squeeze-and-Excitation Blocks

- Improving the quality of representations produced by a network by **explicitly modelling the interdependencies between the channels of its convolutional features**

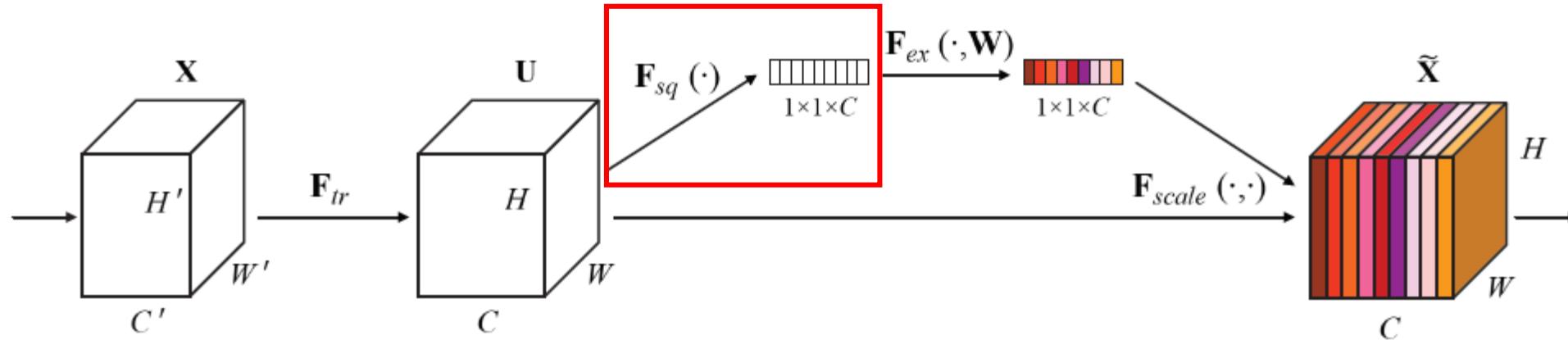


Standard Convolution

- In standard convolution case, the output is produced by a summation through all channels
- Channel dependencies are implicitly embedded in output feature maps
- But, they are entangled with the local spatial correlation captured by filters

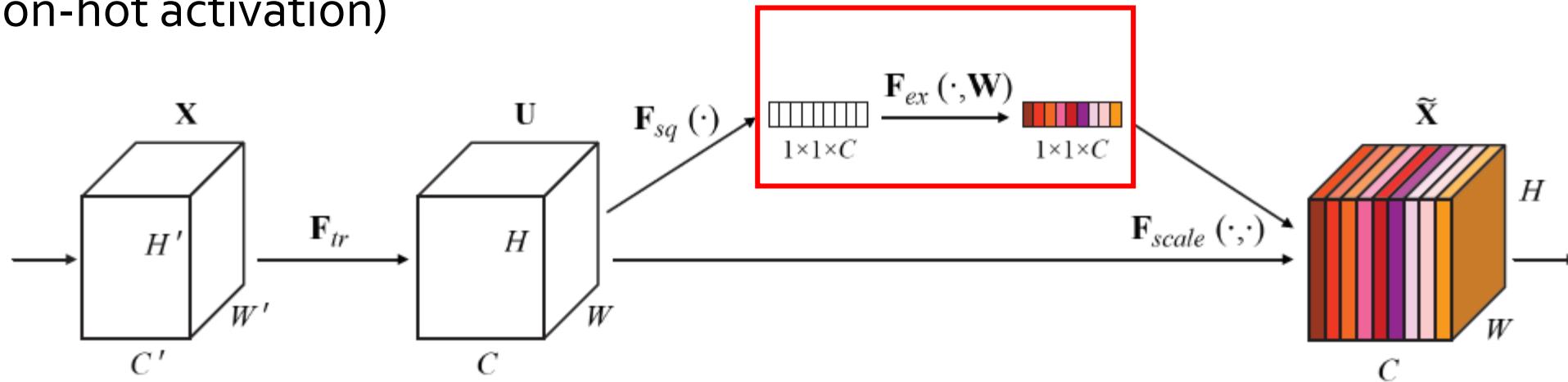
Squeeze : Global Information Embedding

- Authors propose to squeeze global spatial information into a channel descriptor.
- This is achieved by using **global average pooling** to generate channel-wise statistics.
- The output of the transformation(GAP) can be interpreted as a collection of the local descriptors whose statistics are expressive for whole image



Excitation : Adaptive Recalibration

- To make use of the information aggregated in the squeeze operation, authors follow it with a second operation which **aims to fully capture channel-wise dependencies**.
- The function must meet two criteria
 - It must be **flexible**(it must be capable of learning a **nonlinear interaction** btw channels)
 - It must learn a **non-mutually-exclusive relationship**(rather than enforcing a on-hot activation)

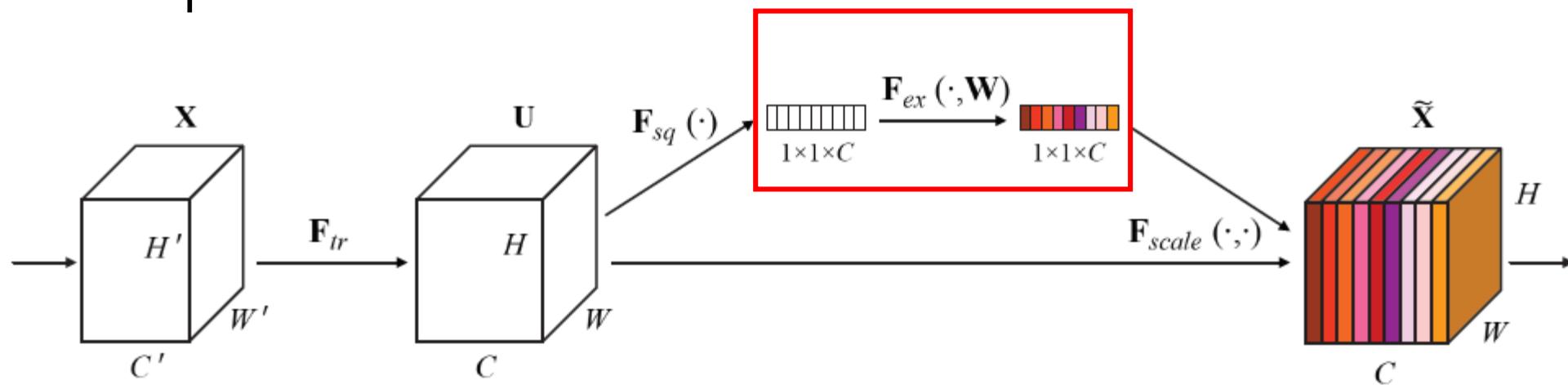


Excitation : Adaptive Recalibration

- To meet these criteria, authors opt to employ a simple gating mechanism with a sigmoid activation (σ : sigmoid, δ : ReLU)

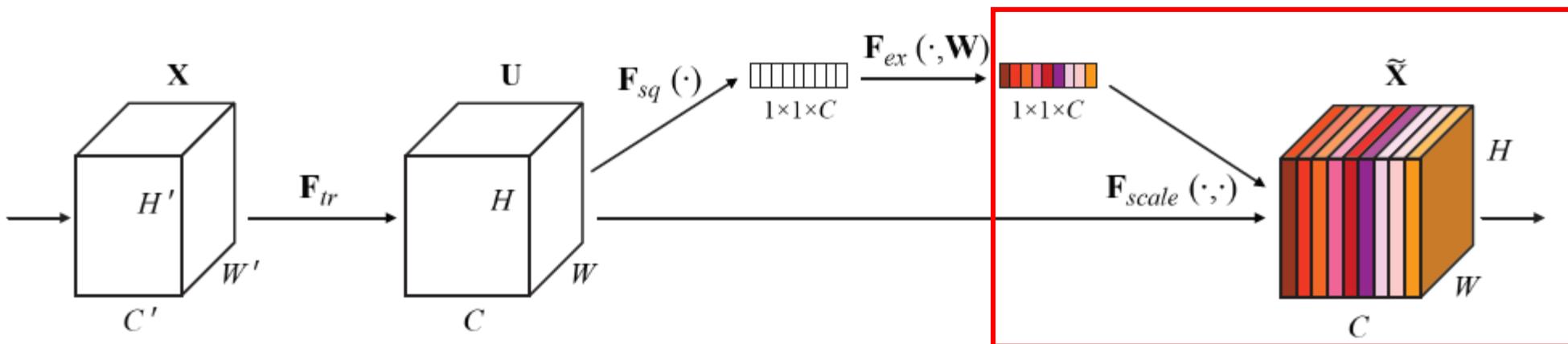
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})),$$

- Forming a bottleneck with **two fully connected layers** around the non-linearity. **A dimensionality-reduction layer** with parameters \mathbf{W}_1 and **reduction ratio r**, a **ReLU** and then a dimensionality-increasing layer with parameters \mathbf{W}_2



Excitation : Adaptive Recalibration

- The final output of the block is obtained by rescaling the transformation output with the activations – **channel-wise multiplication**



Instantiation

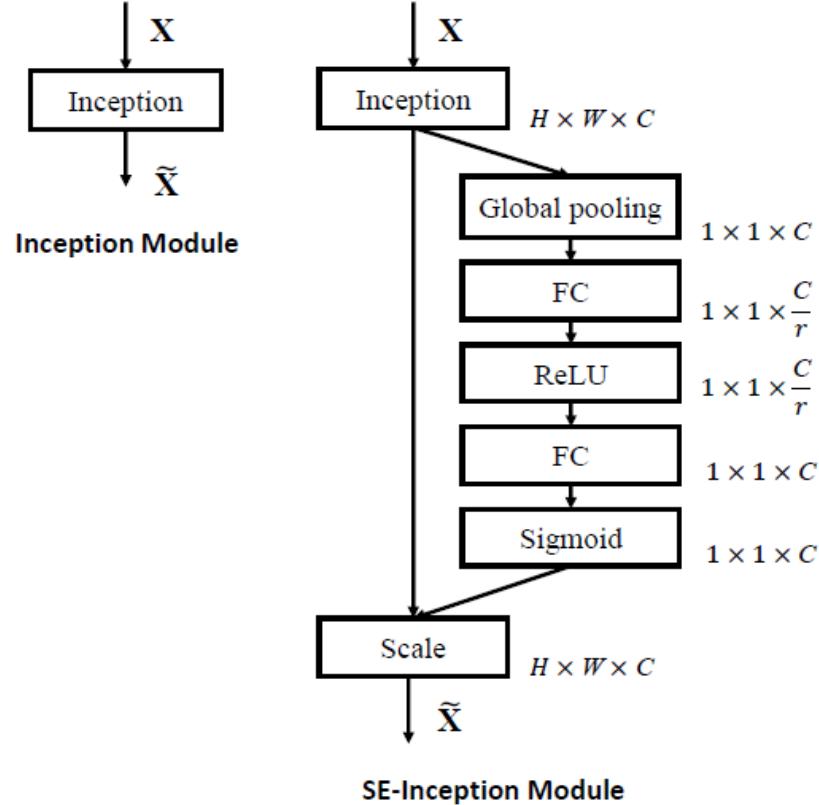


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

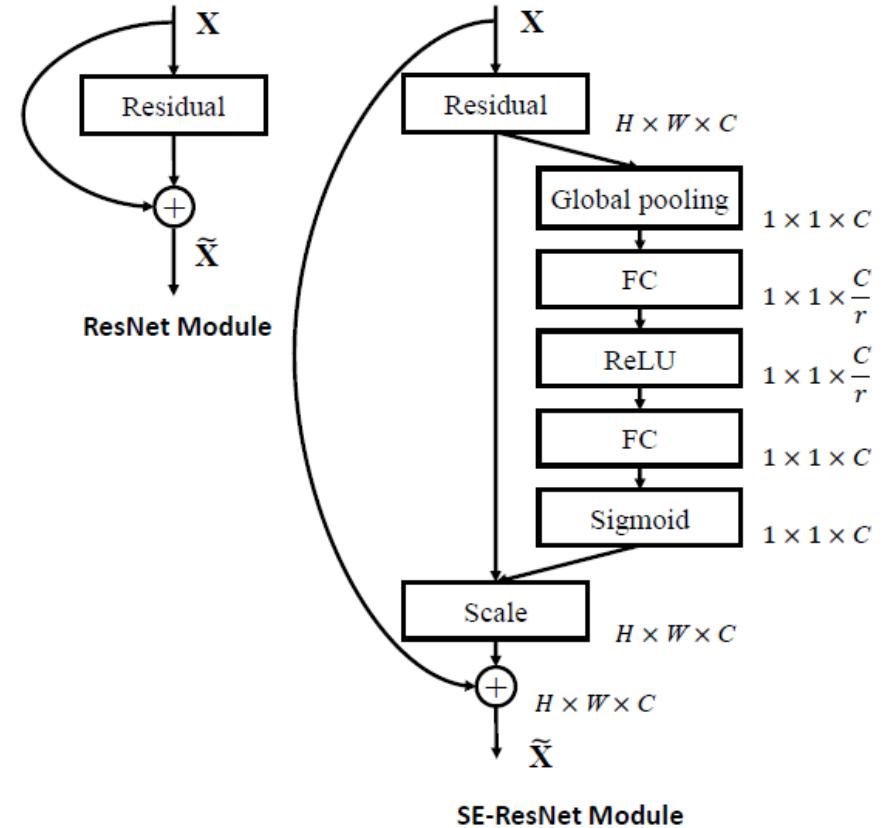


Fig. 3. The schema of the original Residual module (left) and the SE-ResNet module (right).

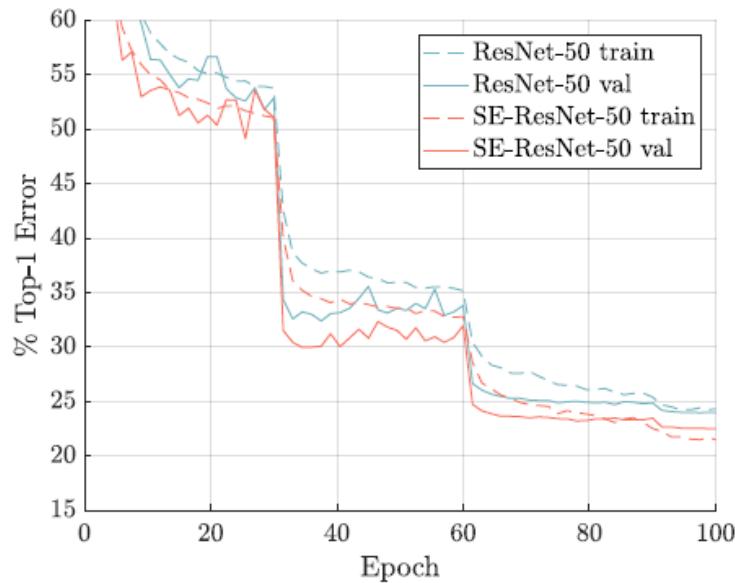
Example Architecture

TABLE 1

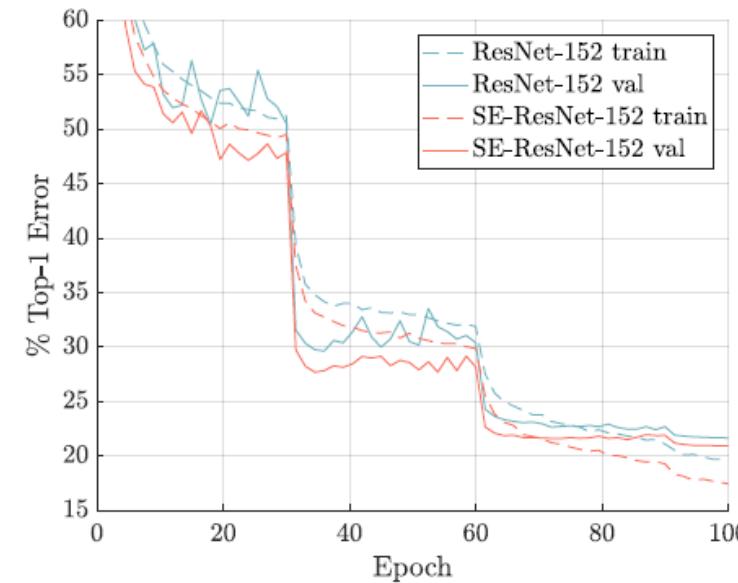
(Left) ResNet-50. (Middle) SE-ResNet-50. (Right) SE-ResNeXt-50 with a $32 \times 4d$ template. The shapes and operations with specific parameter settings of a residual building block are listed inside the brackets and the number of stacked blocks in a stage is presented outside. The inner brackets following by *fc* indicates the output dimension of the two fully connected layers in an SE module.

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 ($32 \times 4d$)
112×112		conv, 7×7 , 64, stride 2	
56×56	$\begin{bmatrix} \text{conv, } 1 \times 1, 64 \\ \text{conv, } 3 \times 3, 64 \\ \text{conv, } 1 \times 1, 256 \end{bmatrix} \times 3$	max pool, 3×3 , stride 2 $\begin{bmatrix} \text{conv, } 1 \times 1, 64 \\ \text{conv, } 3 \times 3, 64 \\ \text{conv, } 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 128 \\ \text{conv, } 3 \times 3, 128 \\ \text{conv, } 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28×28	$\begin{bmatrix} \text{conv, } 1 \times 1, 128 \\ \text{conv, } 3 \times 3, 128 \\ \text{conv, } 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv, } 1 \times 1, 128 \\ \text{conv, } 3 \times 3, 128 \\ \text{conv, } 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv, } 1 \times 1, 256 \\ \text{conv, } 3 \times 3, 256 \\ \text{conv, } 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14×14	$\begin{bmatrix} \text{conv, } 1 \times 1, 256 \\ \text{conv, } 3 \times 3, 256 \\ \text{conv, } 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv, } 1 \times 1, 256 \\ \text{conv, } 3 \times 3, 256 \\ \text{conv, } 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv, } 1 \times 1, 512 \\ \text{conv, } 3 \times 3, 512 \\ \text{conv, } 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv, } 1 \times 1, 512 \\ \text{conv, } 3 \times 3, 512 \\ \text{conv, } 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 512 \\ \text{conv, } 3 \times 3, 512 \\ \text{conv, } 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 1024 \\ \text{conv, } 3 \times 3, 1024 \\ \text{conv, } 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1×1	global average pool, 1000-d <i>fc</i> , softmax		

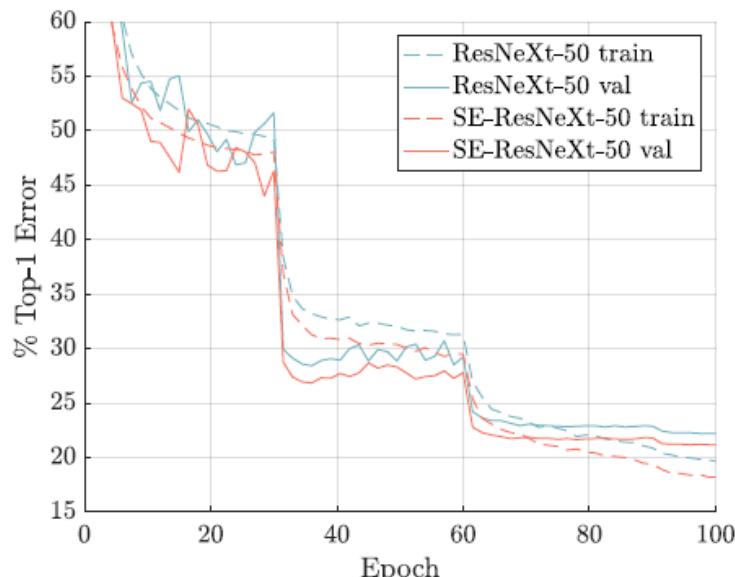
Results



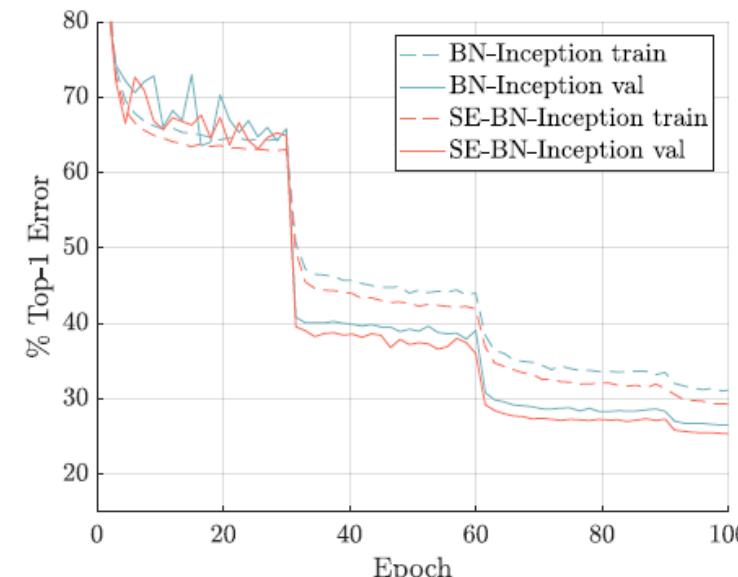
(a) ResNet-50 and SE-ResNet-50



(b) ResNet-152 and SE-ResNet-152



(c) ResNeXt-50 and SE-ResNeXt-50



(d) BN-Inception and SE-BN-Inception

Xception

.02357v3 [cs.CV] 4 Apr 2017

Xception: Deep Learning with Depthwise Separable Convolutions

François Chollet

Google, Inc.

fchollet@google.com

Abstract

We present an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution can be understood as an Inception module with a maximally large number of towers. This observation leads us to propose a novel deep convolutional neural network architecture inspired by Inception, where Inception modules have been replaced with depthwise separable convolutions. We show that this architecture, dubbed Xception, slightly outperforms Inception V3 on the ImageNet dataset (which Inception V3 was designed for), and significantly outperforms Inception V3 on a larger image classification dataset comprising 350 million images and 17,000 classes. Since the Xception architecture has the same number of parameters as Inception V3, the performance gains are not due to increased capacity but rather to a more efficient use of model parameters.

as GoogLeNet (Inception V1), later refined as Inception V2 [7], Inception V3 [21], and most recently Inception-ResNet [19]. Inception itself was inspired by the earlier Network-In-Network architecture [11]. Since its first introduction, Inception has been one of the best performing family of models on the ImageNet dataset [14], as well as internal datasets in use at Google, in particular JFT [5].

The fundamental building block of Inception-style models is the Inception module, of which several different versions exist. In figure 1 we show the canonical form of an Inception module, as found in the Inception V3 architecture. An Inception model can be understood as a stack of such modules. This is a departure from earlier VGG-style networks which were stacks of simple convolution layers.

While Inception modules are conceptually similar to convolutions (they are convolutional feature extractors), they empirically appear to be capable of learning richer representations with less parameters. How do they work, and how do they differ from regular convolutions? What design strategies come after Inception?

Xception

- Observation
 - Inception module try to explicitly factoring two tasks done by a single convolution kernel: mapping cross-channel correlation and spatial correlation
- Inception hypothesis
 - By inception module, these two correlations are sufficiently decoupled
→ Would it be reasonable to make a much stronger hypothesis than the Inception hypothesis?

Xception

Figure 1. A canonical Inception module (Inception V3).

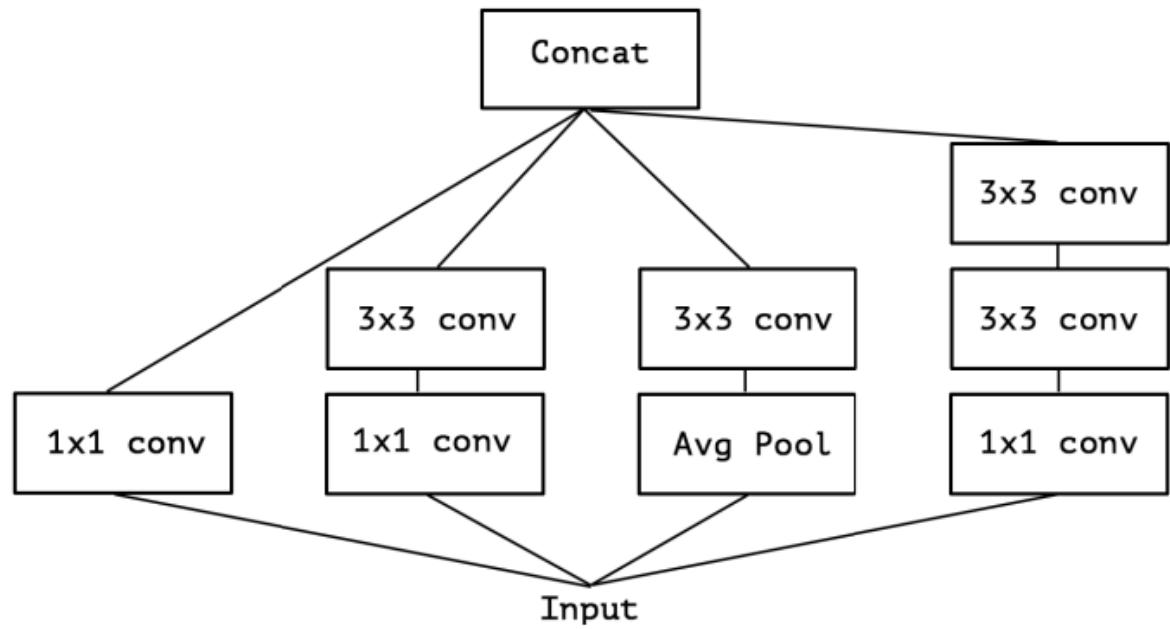
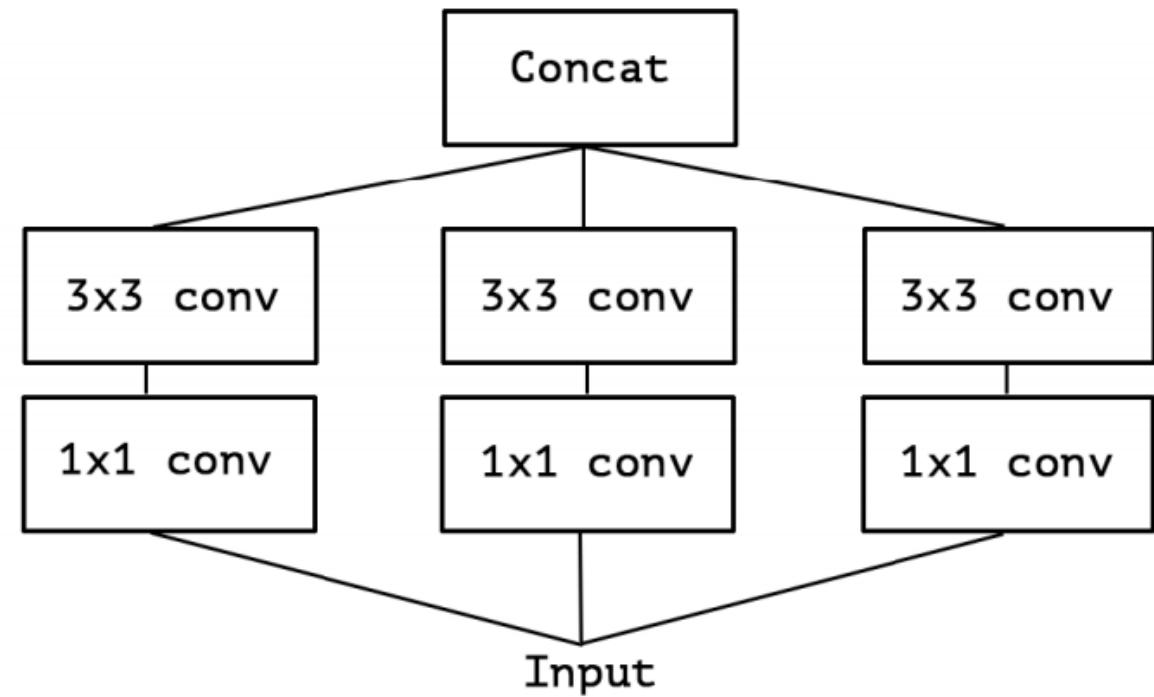


Figure 2. A simplified Inception module.



Equivalent Reformulation

Figure 2. A simplified Inception module.

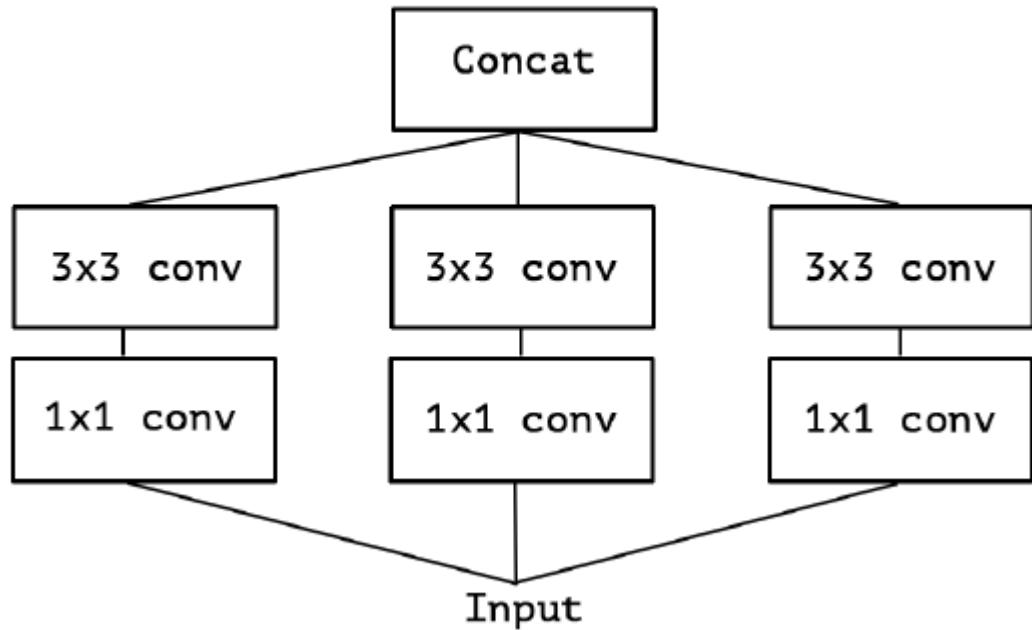
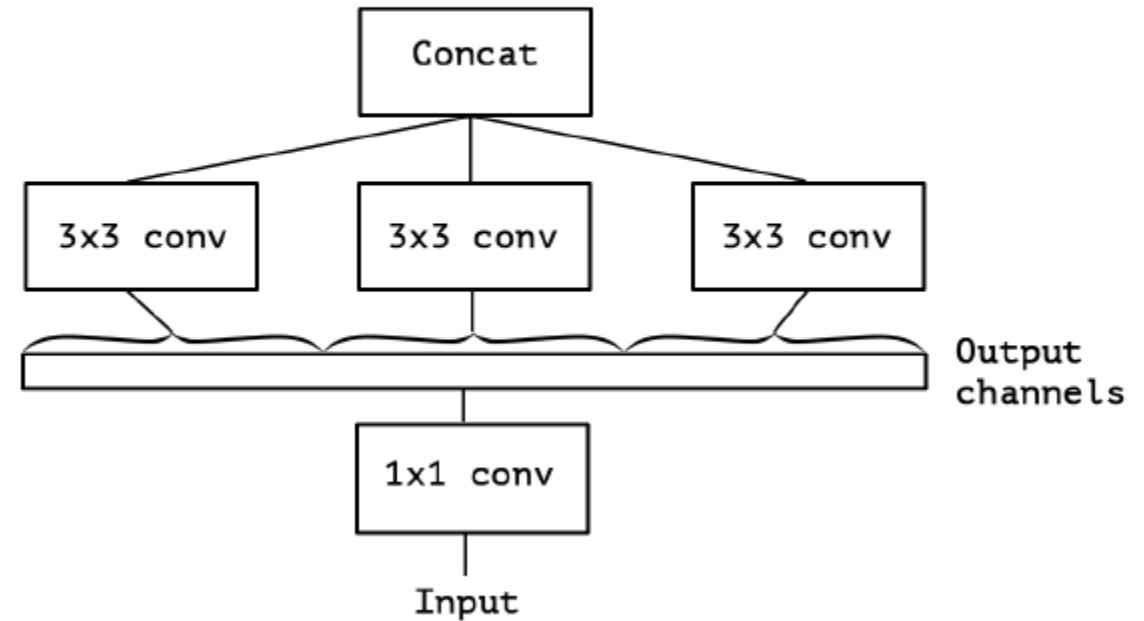
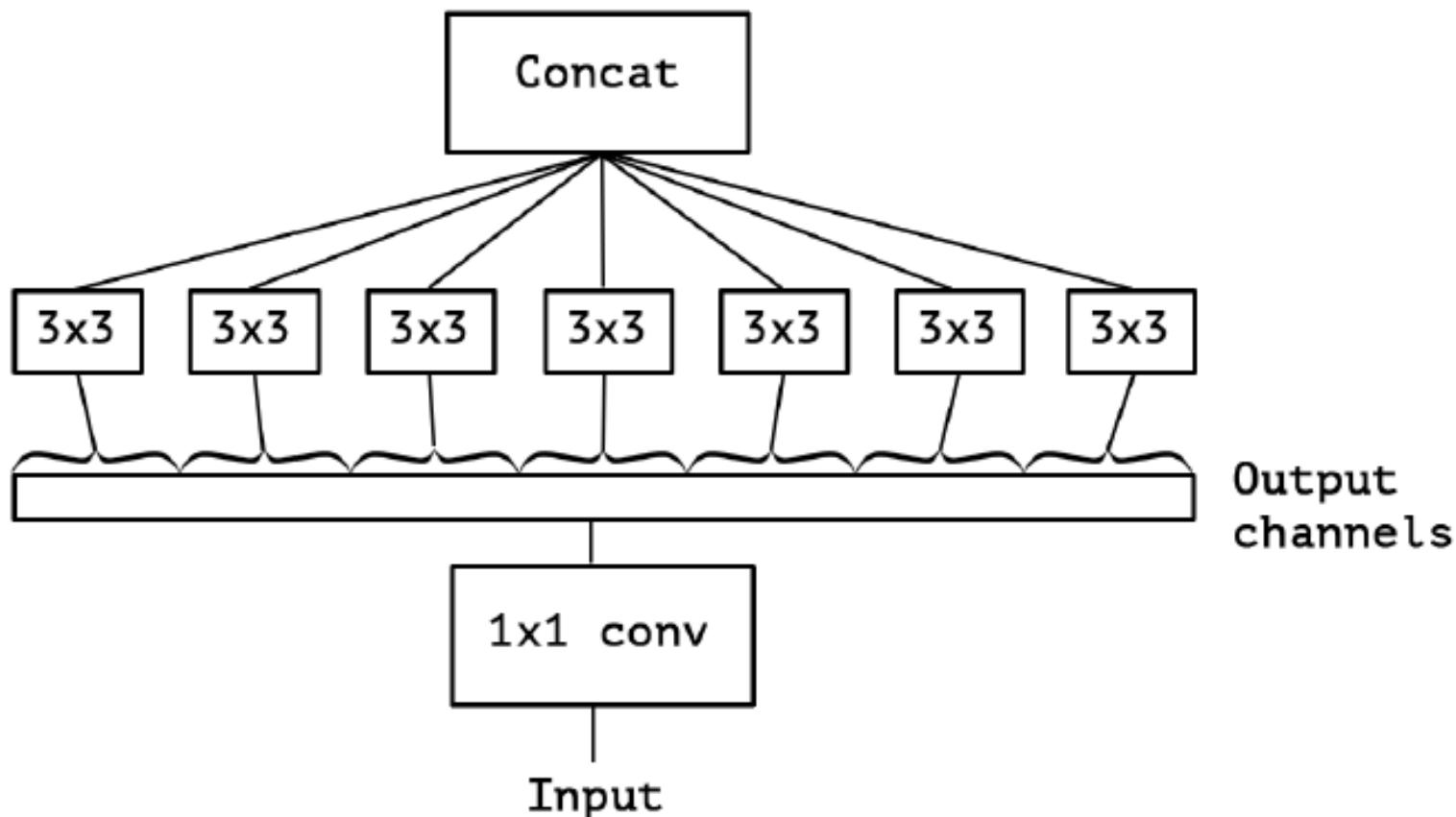


Figure 3. A strictly equivalent reformulation of the simplified Inception module.



Extreme Version of Inception Module

Figure 4. An “extreme” version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.



Xception vs Depthwise Separable Convolution

- The order of the operations
- The presence or absence of a non-linearity after the first operation

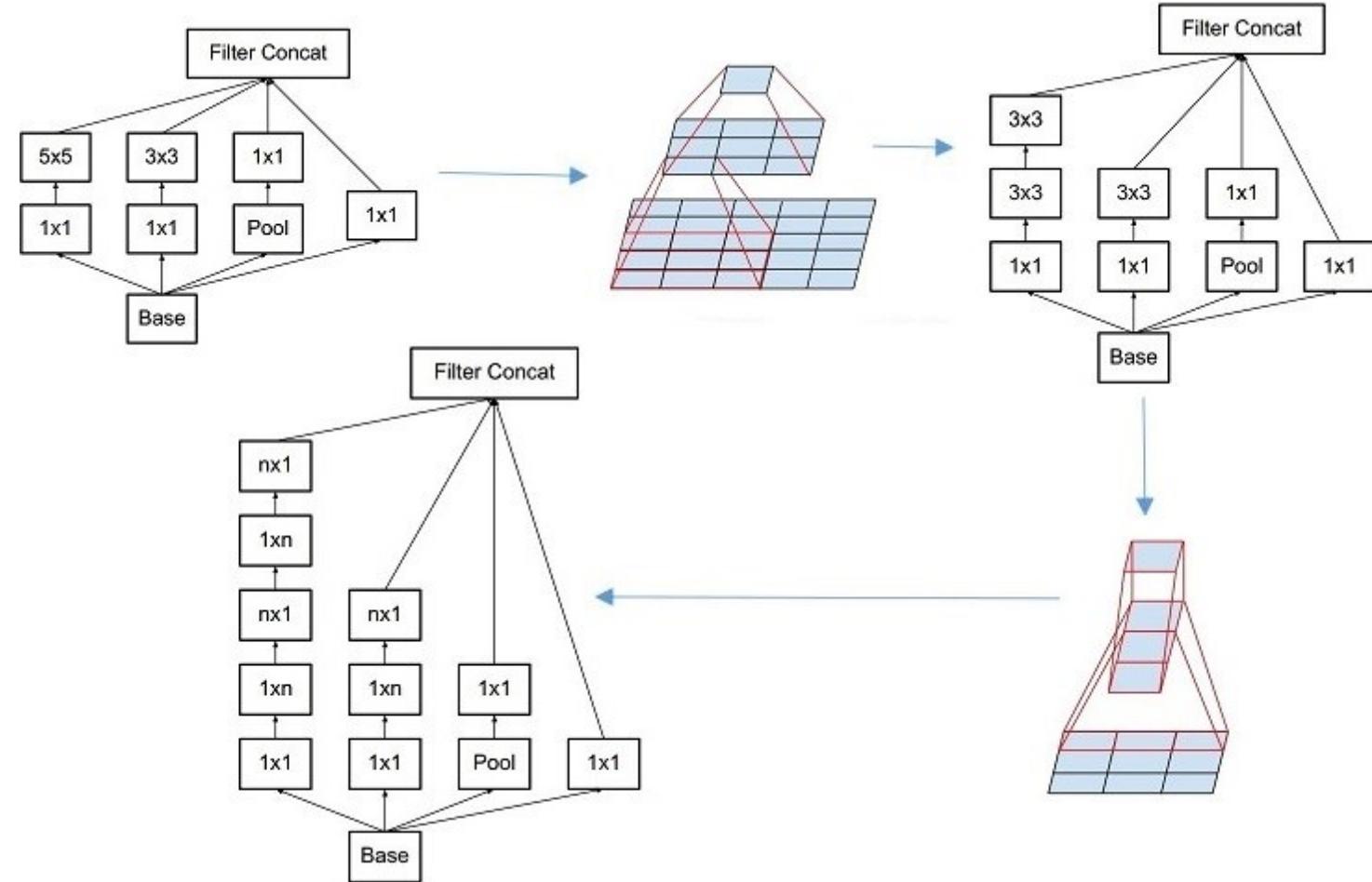
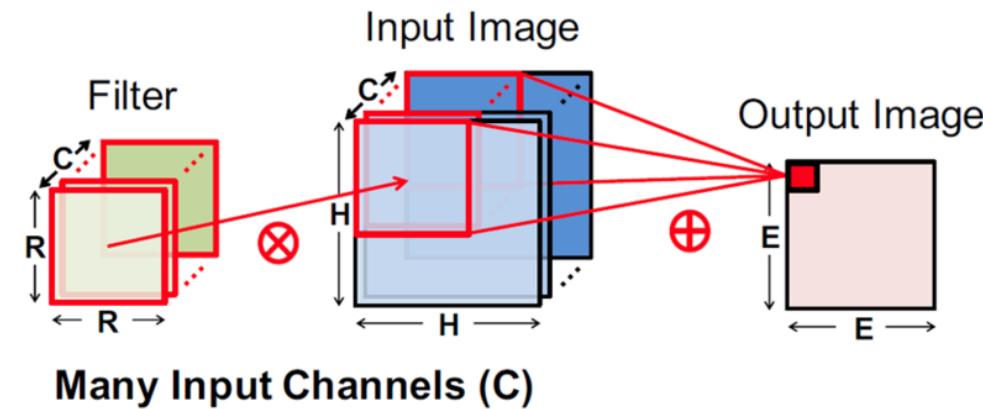


Inception modules lie in between!

Xception Hypothesis

: Make the mapping that *entirely* decouples the cross-channels correlations and spatial correlations

Filter Factorization of CNN



Filter Factorization

Depthwise Separable Convolution

Results

Table 1. Classification performance comparison on ImageNet (single crop, single model). VGG-16 and ResNet-152 numbers are only included as a reminder. The version of Inception V3 being benchmarked does not include the auxiliary tower.

	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	0.790	0.945

ImageNet

Table 2. Classification performance comparison on JFT (single crop, single model).

	FastEval14k MAP@100
Inception V3 - no FC layers	6.36
Xception - no FC layers	6.70
Inception V3 with FC layers	6.50
Xception with FC layers	6.78

JFT

MobileNets

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

Andrew G. Howard

Weijun Wang

Menglong Zhu

Tobias Weyand

Bo Chen

Marco Andreetto

Dmitry Kalenichenko

Hartwig Adam

Google Inc.

{howarda, menglong, bochen, dkalenichenko, weijunw, weyand, anm, hadam}@google.com

Abstract

We present a class of efficient models called MobileNets for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks. We introduce two simple global hyperparameters that efficiently trade off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem. We present extensive experiments on resource and accuracy tradeoffs and show

models. Section 3 describes the MobileNet architecture and two hyper-parameters width multiplier and resolution multiplier to define smaller and more efficient MobileNets. Section 4 describes experiments on ImageNet as well a variety of different applications and use cases. Section 5 closes with a summary and conclusion.

2. Prior Work

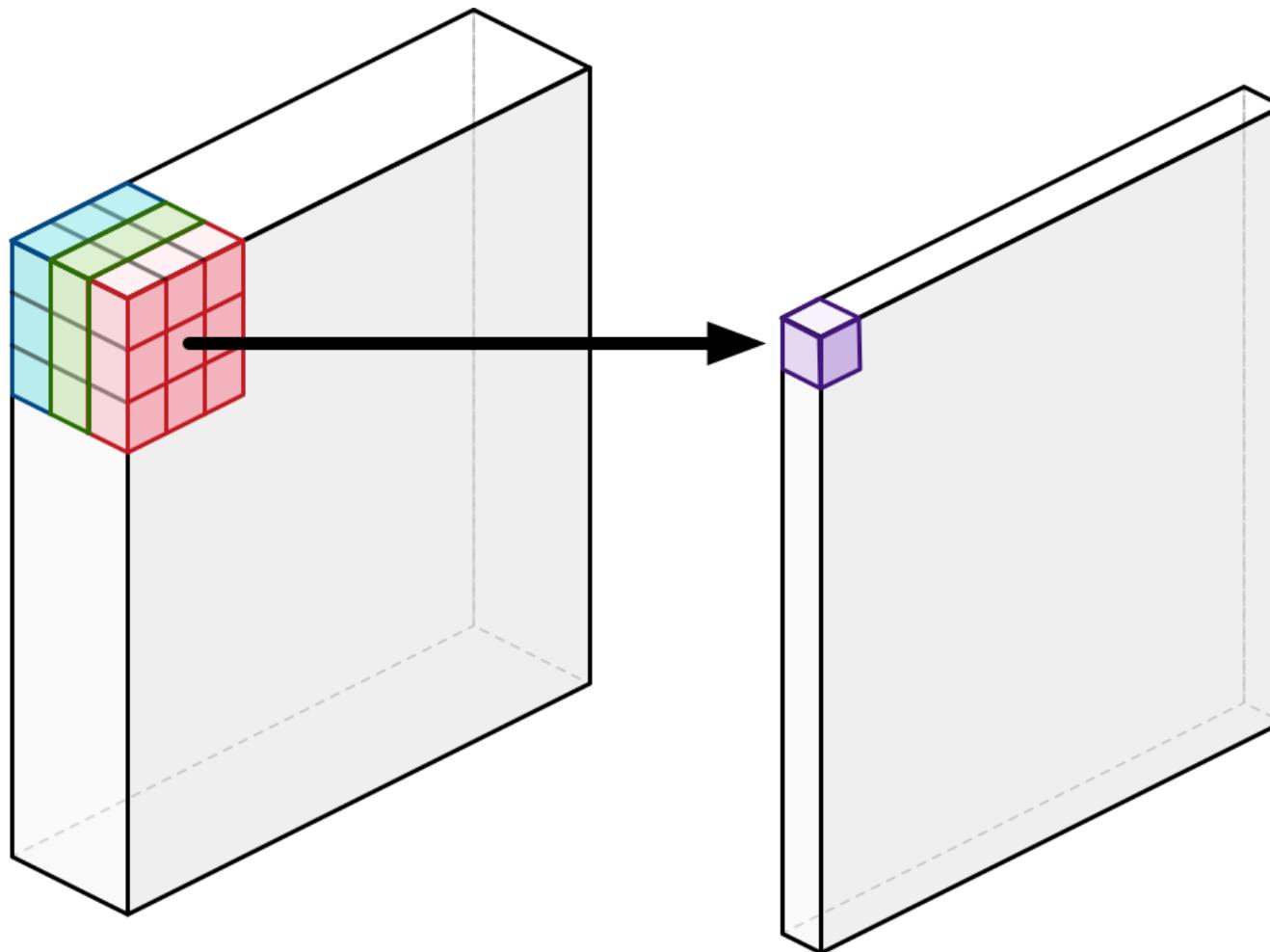
There has been rising interest in building small and efficient neural networks in the recent literature, e.g. [16, 34, 12, 36, 22]. Many different approaches can be generally classified into three main categories: 1) pruning

MobileNets



Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

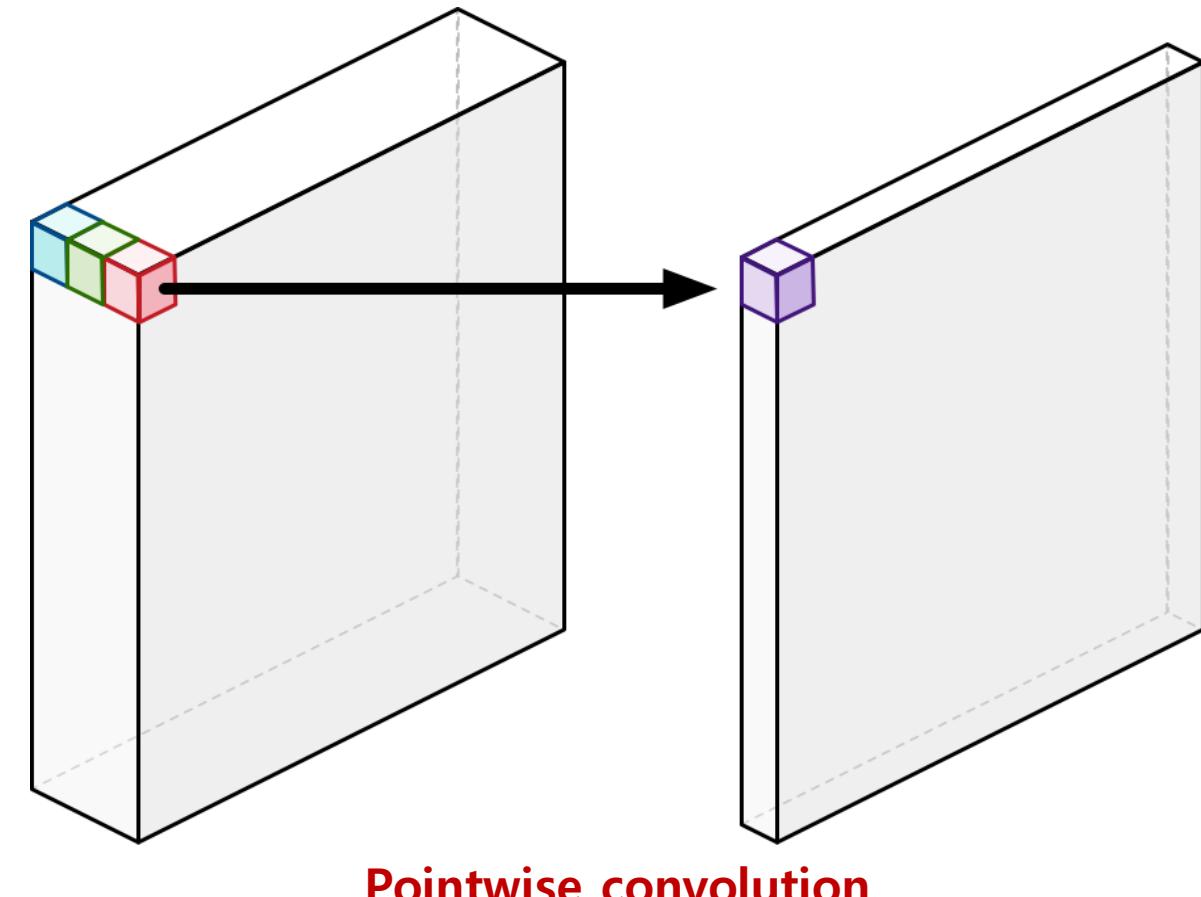
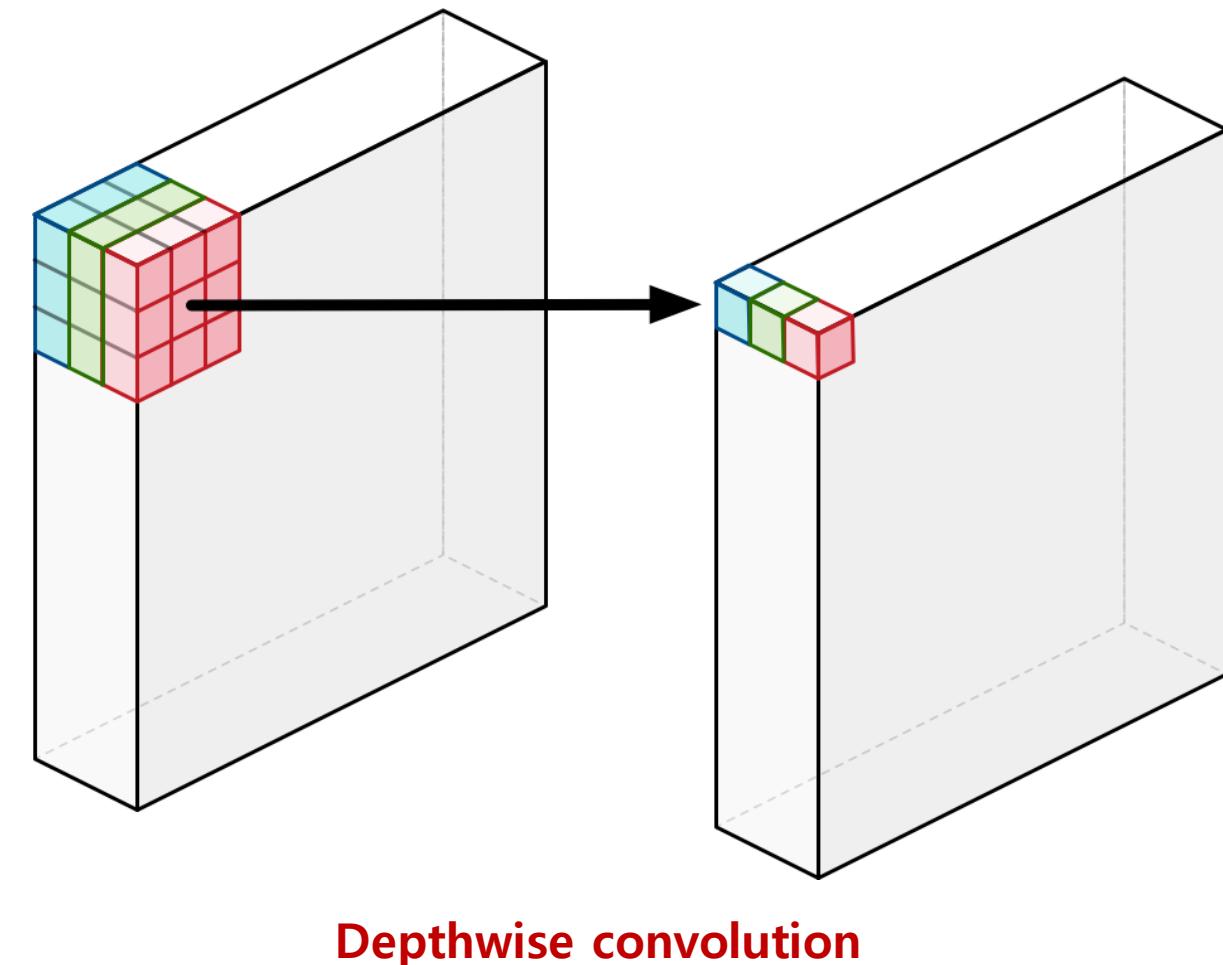
Standard Convolution



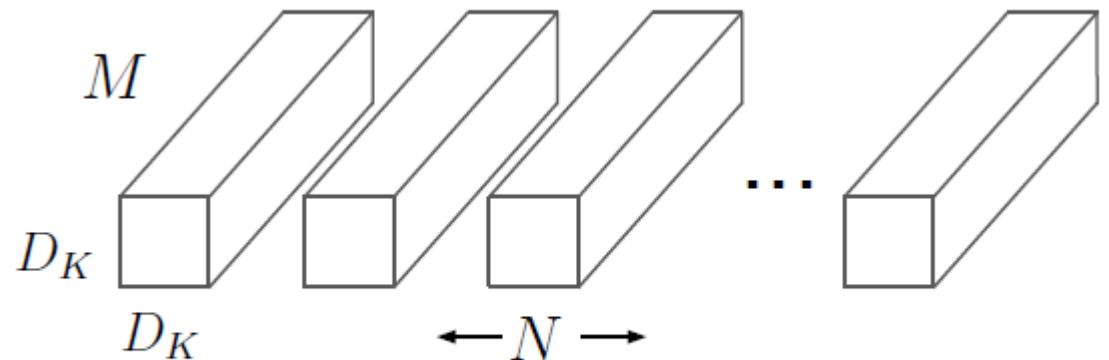
Standard convolution

Depthwise Separable Convolution

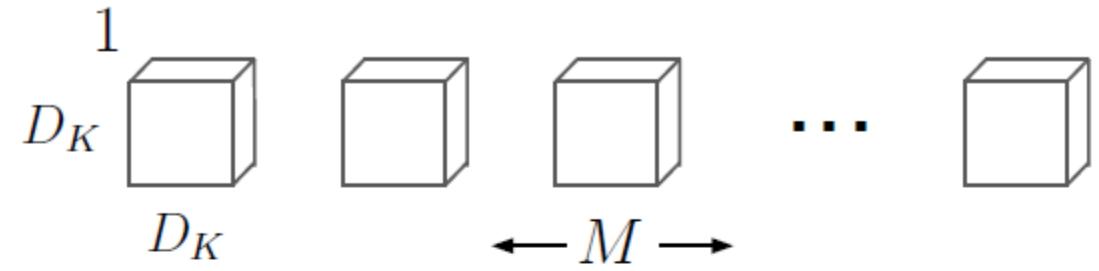
- Depthwise Convolution + Pointwise Convolution(1×1 convolution)



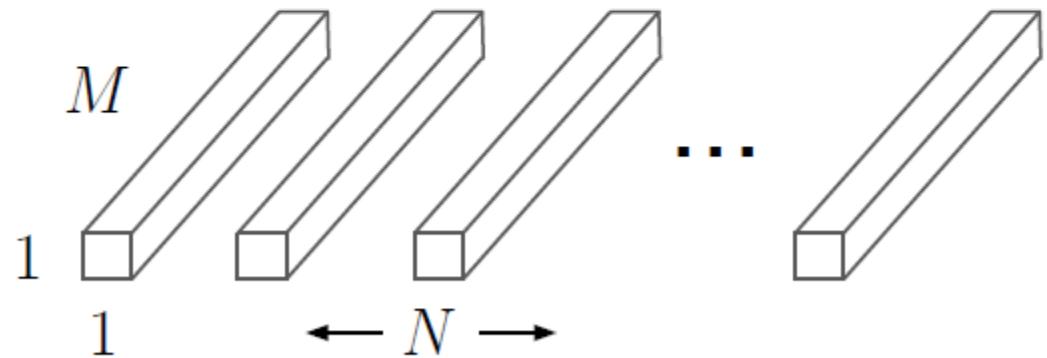
Standard Convolution vs Depthwise Separable Convolution



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Standard Convolution vs Depthwise Separable Convolution

- Standard convolutions have the computational cost of
 - $D_K \times D_K \times M \times N \times D_F \times D_F$
- Depthwise separable convolutions cost
 - $D_K \times D_K \times M \times D_F \times D_F + M \times N \times D_F \times D_F$
- Reduction in computations
 - $1/N + 1/D_K^2$
 - If we use 3×3 depthwise separable convolutions, we get between 8 to 9 times less computations

Depthwise Separable Convolutions

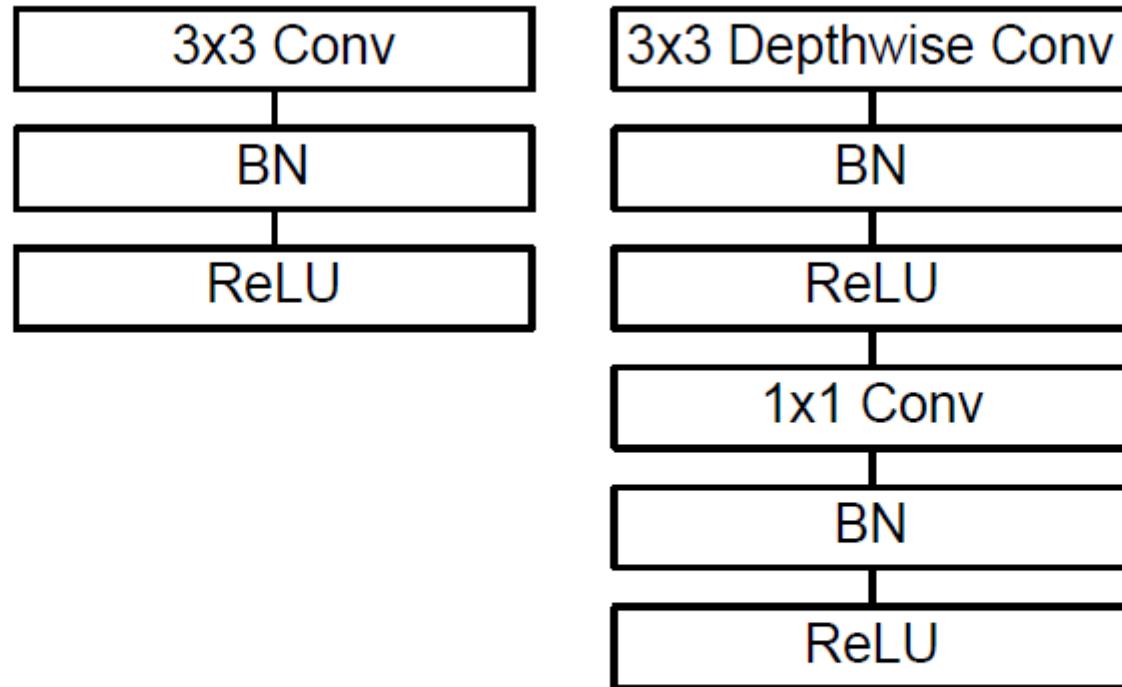


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Model Structure

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	$14 \times 14 \times 512$
	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
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	$1 \times 1 \times 1000$

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%

Width Multiplier & Resolution Multiplier

- For a given layer and width multiplier α , the number of input channels M becomes αM and the number of output channels N becomes αN – where α with typical settings of 1, 0.75, 0.5 and 0.25
- The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ
- Computational cost:
$$D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$$

Width Multiplier & Resolution Multiplier

Table 3. Resource usage for modifications to standard convolution. Note that each row is a cumulative effect adding on top of the previous row. This example is for an internal MobileNet layer with $D_K = 3$, $M = 512$, $N = 512$, $D_F = 14$.

Layer/Modification	Million Mult-Adds	Million Parameters
Convolution	462	2.36
Depthwise Separable Conv	52.3	0.27
$\alpha = 0.75$	29.6	0.15
$\rho = 0.714$	15.1	0.15

Experiments – Model Choices

Table 4. Depthwise Separable vs Full Convolution MobileNet

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

Table 5. Narrow vs Shallow MobileNet

Model	ImageNet	Million	Million
	Accuracy	Mult-Adds	Parameters
0.75 MobileNet	68.4%	325	2.6
Shallow MobileNet	65.3%	307	2.9

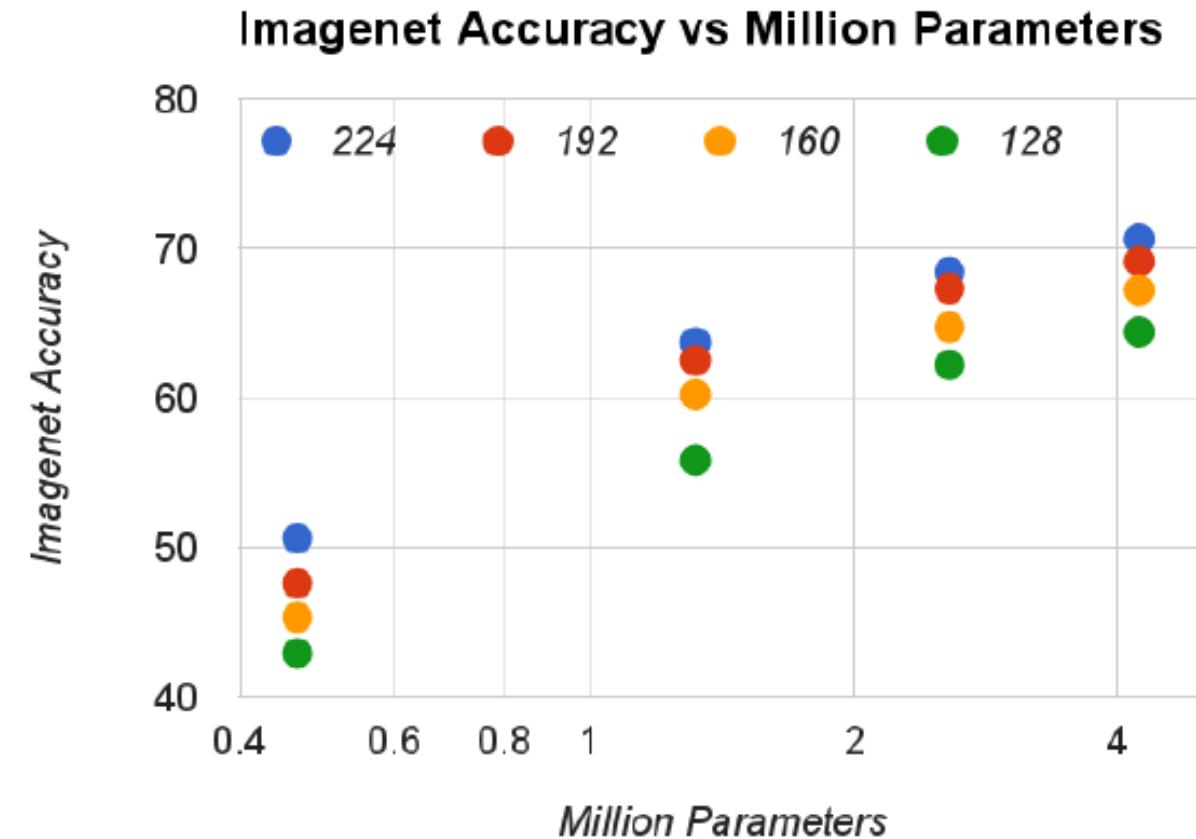
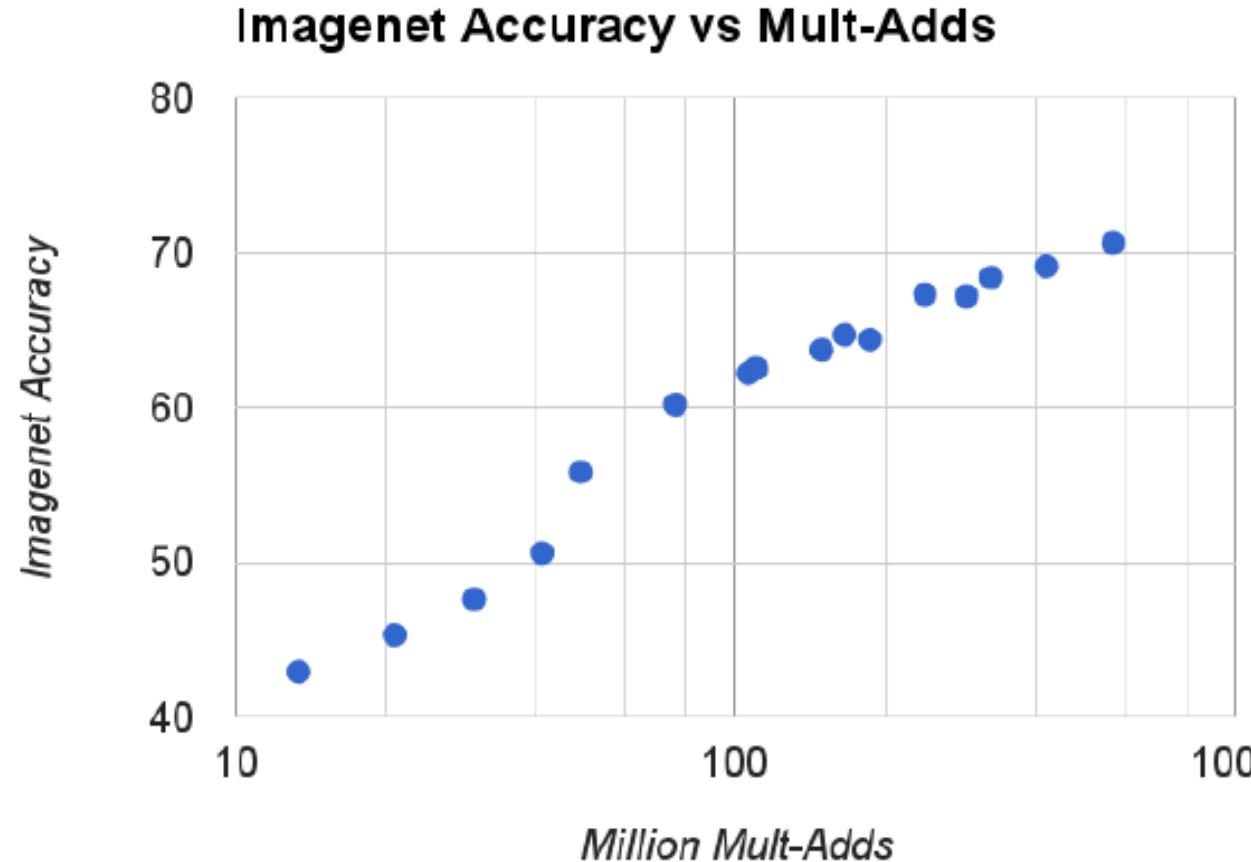
Table 6. MobileNet Width Multiplier

Width Multiplier	ImageNet	Million	Million
	Accuracy	Mult-Adds	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	Million	Million
	Accuracy	Mult-Adds	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

Model Shrinking Hyperparameters



ShuffleNet

ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices

Xiangyu Zhang*

Xinyu Zhou*

Mengxiao Lin

Jian Sun

Megvii Inc (Face++)

{zhangxiangyu, zxy, linmengxiao, sunjian}@megvii.com

Abstract

We introduce an extremely computation-efficient CNN architecture named *ShuffleNet*, which is designed specially for mobile devices with very limited computing power (e.g., 10-150 MFLOPs). The new architecture utilizes two new operations, pointwise group convolution and channel shuffle, to greatly reduce computation cost while maintaining accuracy. Experiments on ImageNet classification and MS COCO object detection demonstrate the superior performance of *ShuffleNet* over other structures, e.g. lower top-1 error (absolute 7.8%) than recent *MobileNet* [12] on ImageNet classification task, under the computation budget of 40 MFLOPs. On an ARM-based mobile device, *ShuffleNet* achieves $\sim 13\times$ actual speedup over *AlexNet* while maintaining comparable accuracy.

tions to reduce computation complexity of 1×1 convolutions. To overcome the side effects brought by group convolutions, we come up with a novel *channel shuffle* operation to help the information flowing across feature channels. Based on the two techniques, we build a highly efficient architecture called *ShuffleNet*. Compared with popular structures like [30, 9, 40], for a given computation complexity budget, our *ShuffleNet* allows more feature map channels, which helps to encode more information and is especially critical to the performance of very small networks.

We evaluate our models on the challenging ImageNet classification [4, 29] and MS COCO object detection [23] tasks. A series of controlled experiments shows the effectiveness of our design principles and the better performance over other structures. Compared with the state-of-the-art architecture *MobileNet* [12], *ShuffleNet* achieves superior performance by a significant margin, e.g. absolute 7.8%

Toward More Efficient Network Architecture

- The 1×1 convolution accounts for most of the computation

Table 2. Resource Per Layer Type

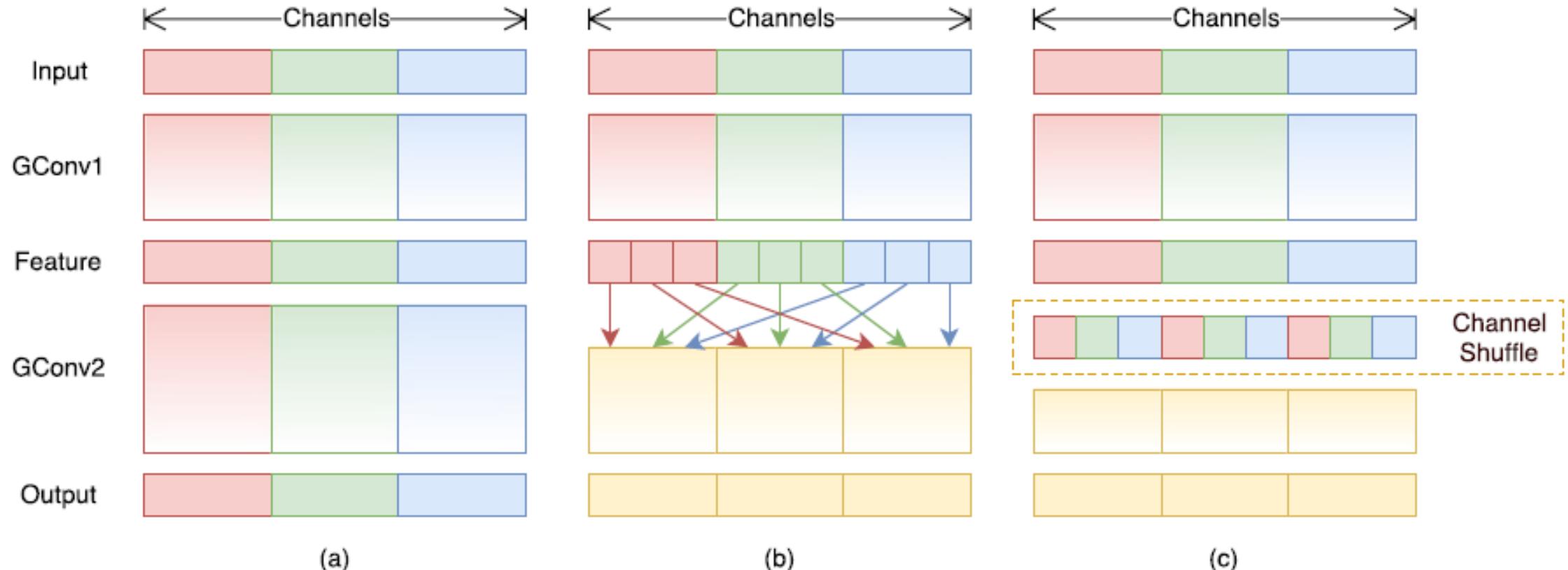
Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

- Can we reduce more?

Main Ideas of ShuffleNet

- (Use depthwise separable convolution)
- Grouped convolution on **1x1 convolution layers** – pointwise group convolution
- Channel shuffle operation after pointwise group convolution

1x1 Group Convolution with Channel Shuffling



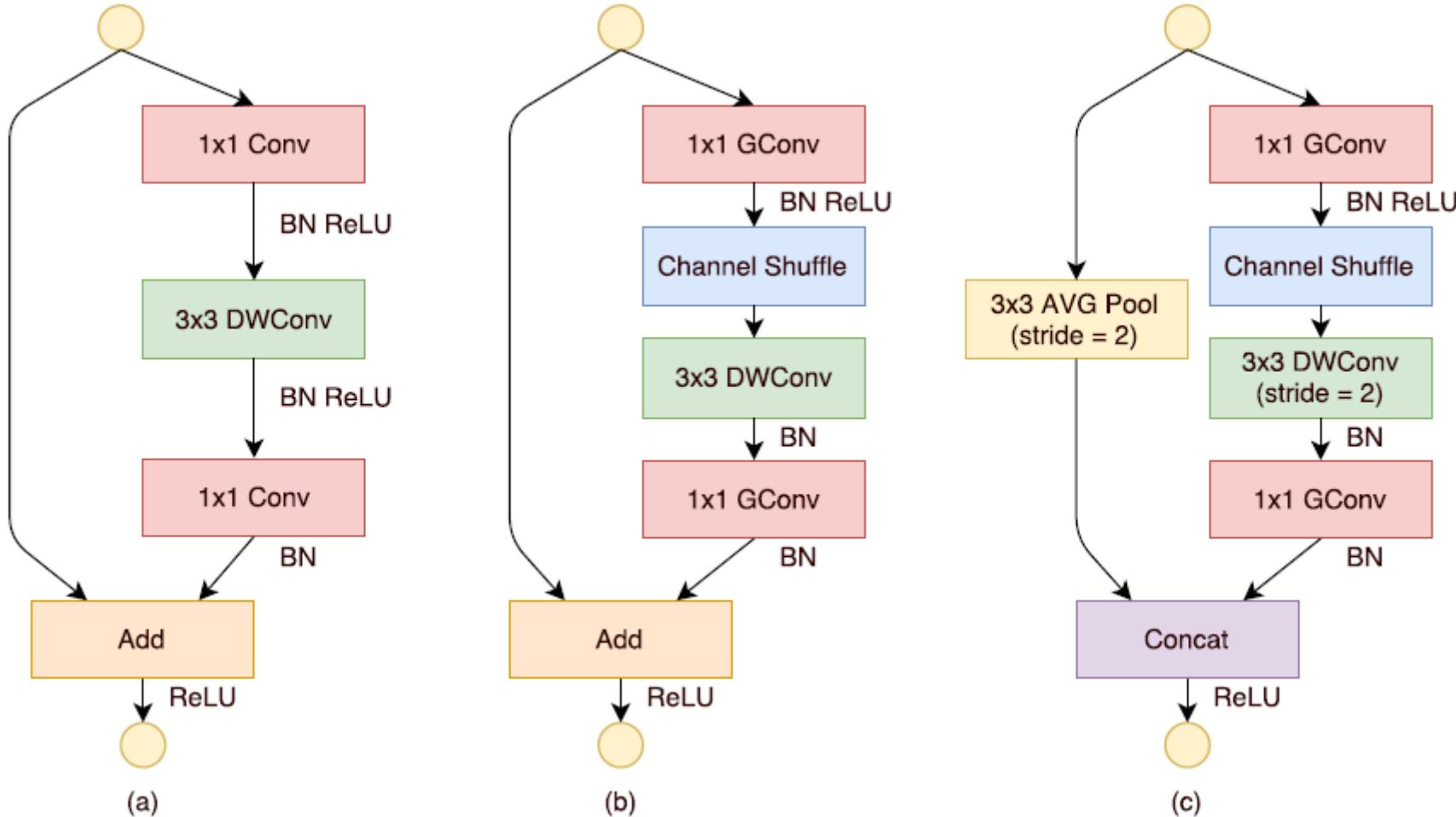
- If multiple group convolutions stack together, there is one side effect(a)
 - Outputs from a certain channel are only derived from a small fraction of input channels
- If we allow group convolution to obtain input data from different groups, the input and output channels will be fully related.

Channel Shuffle Operation

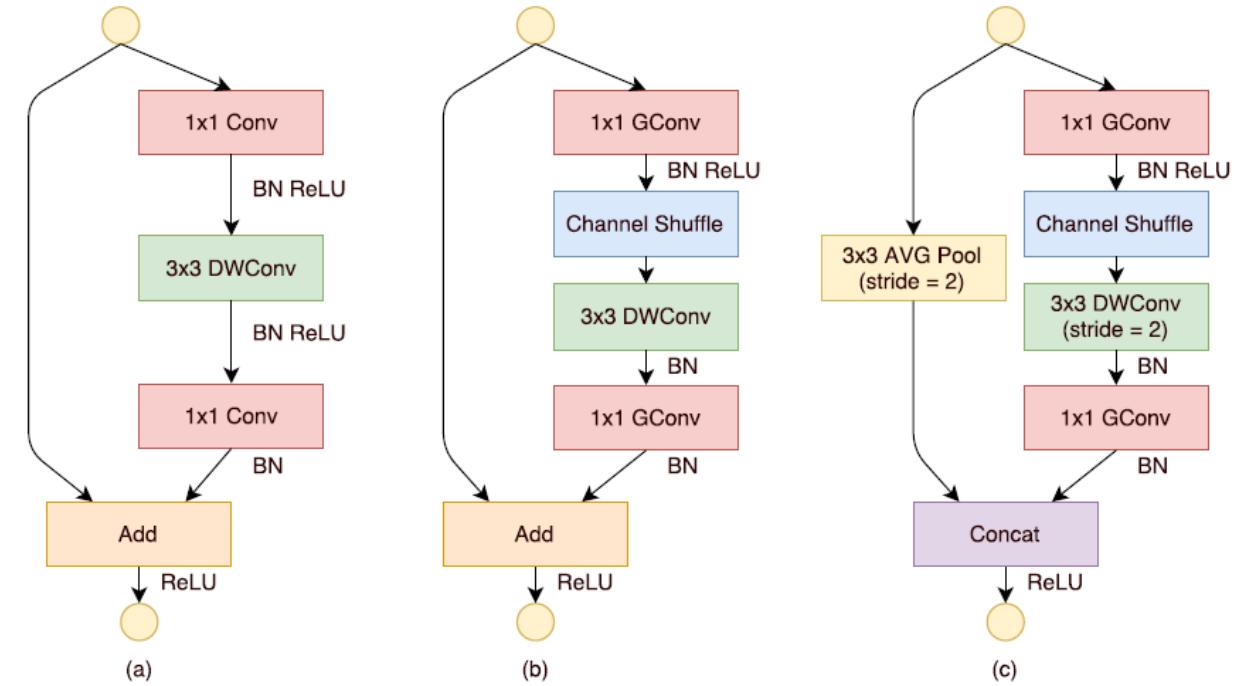
- Suppose a convolutional layer with g groups whose output has $g \times n$ channels; we first reshape the output channel dimension into (g, n) , transposing and then flattening it back as the input of next layer.
- Channel shuffle operation is also differentiable

```
def channel_shuffle(name, x, num_groups):  
    with tf.variable_scope(name) as scope:  
        n, h, w, c = x.shape.as_list()  
        x_reshaped = tf.reshape(x, [-1, h, w, num_groups, c // num_groups])  
        x_transposed = tf.transpose(x_reshaped, [0, 1, 2, 4, 3])  
        output = tf.reshape(x_transposed, [-1, h, w, c])  
    return output
```

ShuffleNet Units



ShuffleNet Units



- From (a), replace the first **1x1** layer with pointwise group convolution followed by a channel shuffle operation
- ReLU is not applied to **3x3 DWConv**
- As for the case where ShuffleNet is applied with stride, simply make to modifications
 - Add **3x3 average pooling** on the shortcut path
 - Replace element-wise addition with channel concatenation to enlarge channel dimension with little extra computation

Experimental Results

- It is clear that ShuffleNet models are superior to MobileNet for all the complexities though ShuffleNet network is specially designed for small models (< 150 MFLOPs)
- Results show that the shallower model is still significantly better than the corresponding MobileNet, which implies that the effectiveness of ShuffleNet mainly results from its efficient structure, not the depth.

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

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

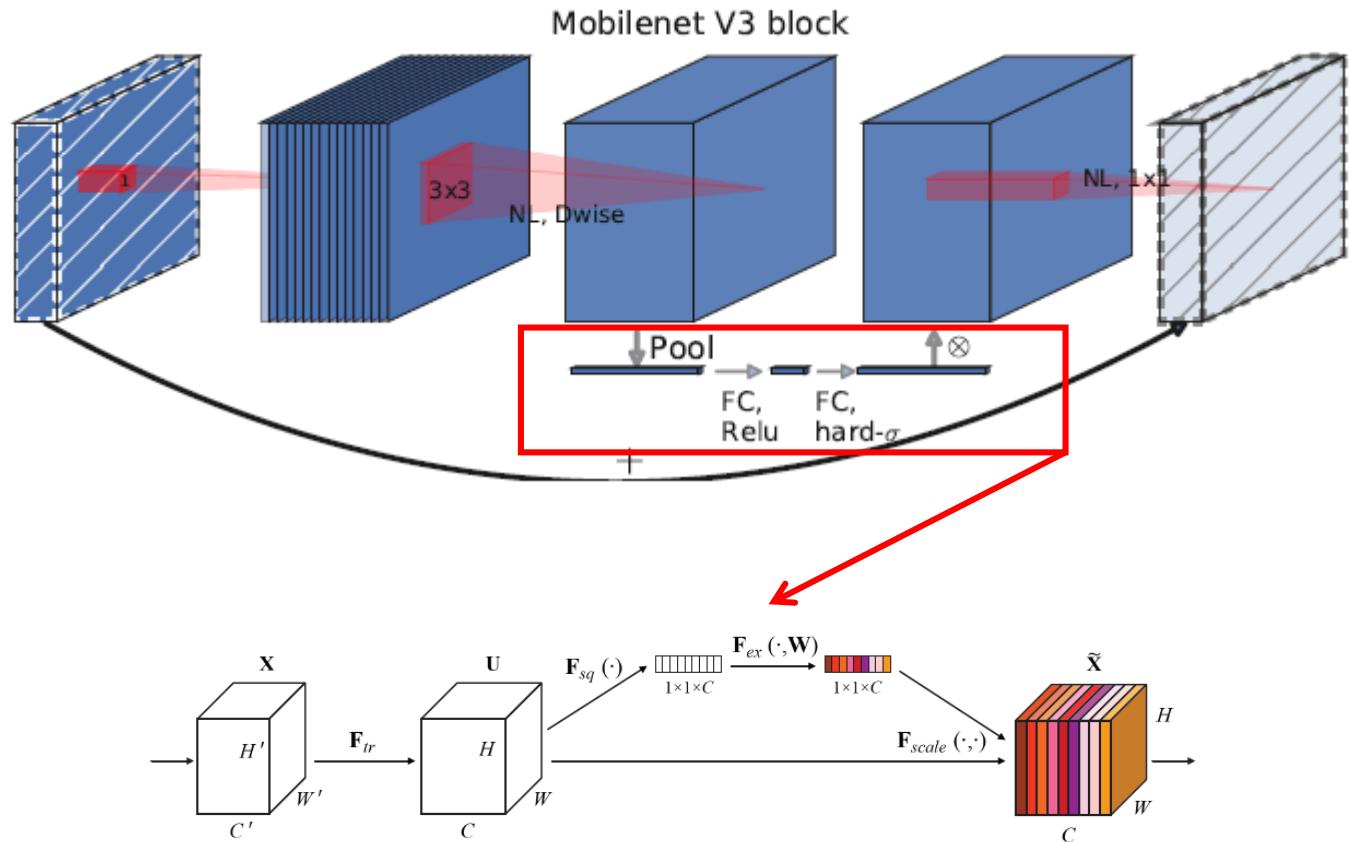
Experimental Results

- Results show that with similar accuracy ShuffleNet is much more efficient than others.

Model	Clss err. (%)	Complexity (MFLOPs)
VGG-16 [30]	28.5	15300
ShuffleNet $2\times$ ($g = 3$)	26.3	524
GoogleNet [33]*	31.3	1500
ShuffleNet $1\times$ ($g = 8$)	32.4	140
AlexNet [21]	42.8	720
SqueezeNet [14]	42.5	833
ShuffleNet $0.5\times$ ($g = 4$)	41.6	38

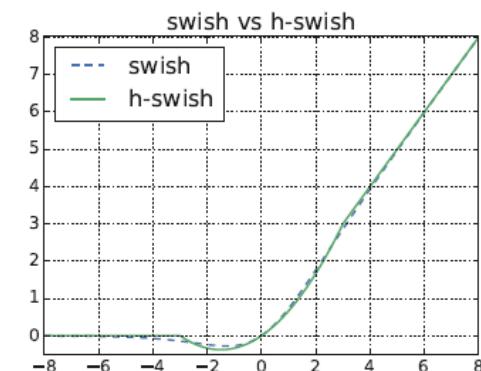
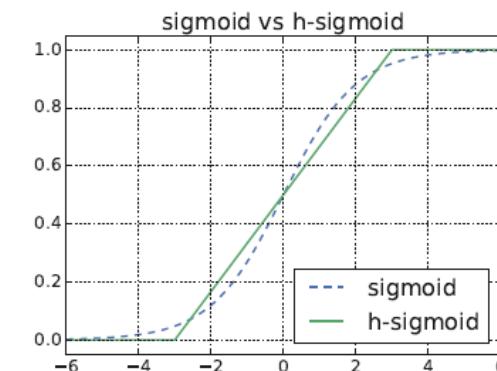
Table 6. Complexity comparison. *Implemented by BVLC (https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet)

MobileNetV2, V3

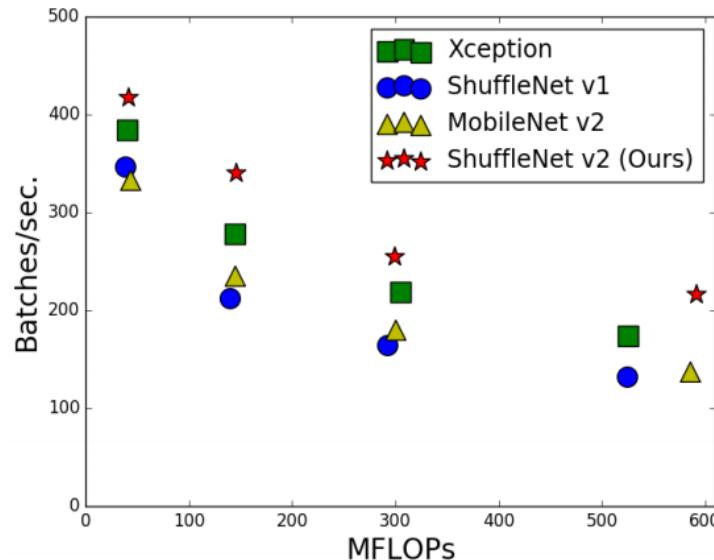


$$\text{swish } x = x \cdot \sigma(x)$$

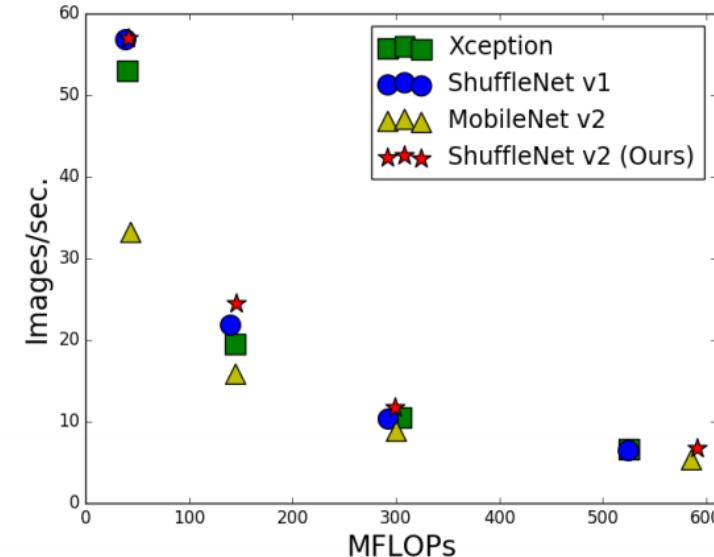
$$\text{h-swish}[x] = x \frac{\text{ReLU6}(x + 3)}{6}$$



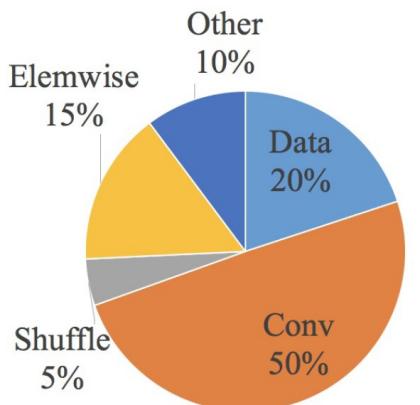
ShuffleNet V2



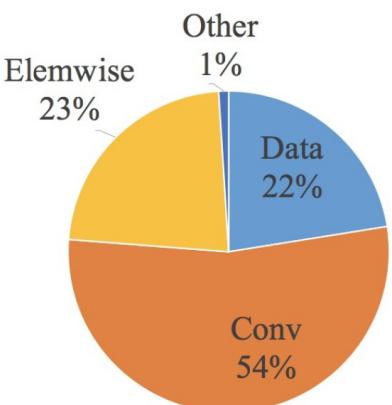
(c) GPU



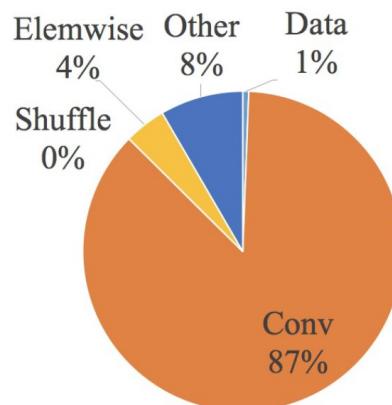
(d) ARM



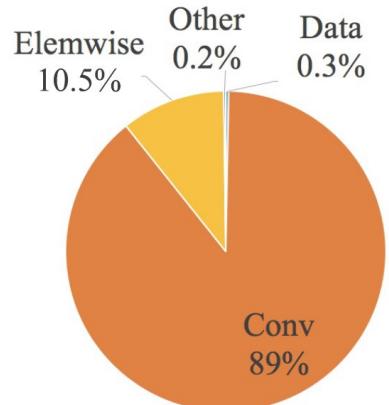
ShuffleNet V1 on GPU



MobileNet V2 on GPU



ShuffleNet V1 on ARM



MobileNet V2 on ARM

ShuffleNet V2 – Guide 1

- **Equal channel width minimizes memory access cost (MAC).**
- Let h and w be the spatial size of the feature map, the FLOPs of the 1×1 convolution is $B = hwc_1c_2$.
- The memory access cost (MAC), or the number of memory access operations, is $MAC = hw(c_1 + c_2) + c_1c_2$.
- MAC has a lower bound given by FLOPs. It reaches the lower bound when the numbers of input and output channels are equal.

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

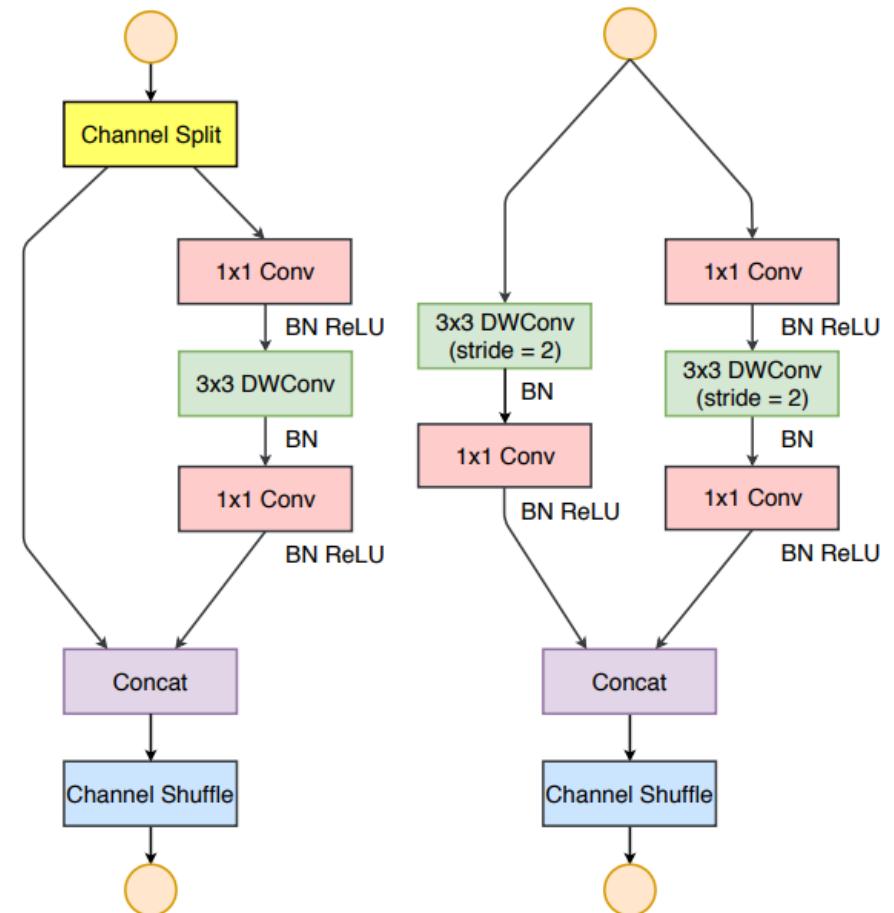
ShuffleNet V2 – Guide 1

c1:c2	(c1,c2) for $\times 1$	GPU (Batches/sec.)			ARM (Images/sec.)			
		$\times 1$	$\times 2$	$\times 4$	(c1,c2) for $\times 1$	$\times 1$	$\times 2$	$\times 4$
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 (c_1 and c_2) are tested, while the total FLOPs under the four ratios is fixed by varying the number of channels. Input image size is 56×56 .

ShuffleNet V2

1. Use "balanced" convolutions (equal channel width).
2. Be aware of the cost of using group convolution.
3. Reduce the degree of fragmentation.
4. Reduce element-wise operations.



NASNet

Learning Transferable Architectures for Scalable Image Recognition

Barret Zoph

Google Brain

barrettzoph@google.com

Vijay Vasudevan

Google Brain

vrv@google.com

Jonathon Shlens

Google Brain

shlens@google.com

Quoc V. Le

Google Brain

qvl@google.com

Abstract

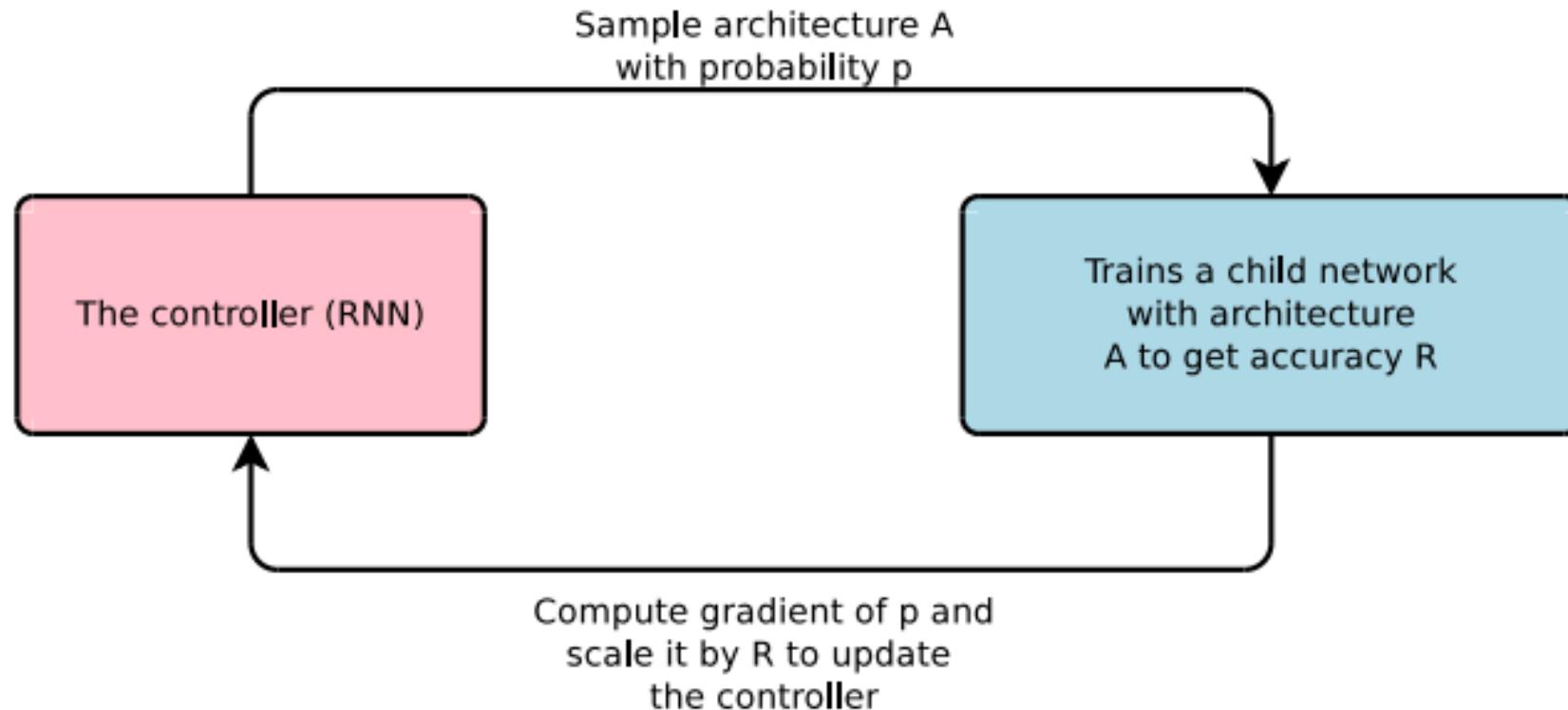
Developing neural network image classification models often requires significant architecture engineering. In this paper, we attempt to automate this engineering process by learning the model architectures directly on the dataset of interest. As this approach is expensive when the dataset is large, we propose to search for an architectural building block on a small dataset and then transfer the block to a larger dataset. Our key contribution is the design of a new search space which enables transferability. In our experiments, we search for the best convolutional layer (or “cell”) on the CIFAR-10 dataset and then apply this cell to the ImageNet dataset by stacking together more copies of this

cation represents one of the most important breakthroughs in deep learning. Successive advancements on this benchmark based on convolutional neural networks (CNNs) have achieved impressive results through significant architecture engineering [52, 58, 20, 59, 57, 67].

In this paper, we consider learning the convolutional architectures directly from data with application to ImageNet classification. In addition to being an difficult and important benchmark in computer vision, features derived from ImageNet classifiers are of great importance to many other computer vision tasks. For example, features from networks that perform well on ImageNet classification provide state-of-the-art performance when transferred to other computer vision tasks where labeled data is limited [13].

Neural Architecture Search

- B. Zoph and Q. V. Le., “Neural architecture search with reinforcement learning”, ICLR-2017



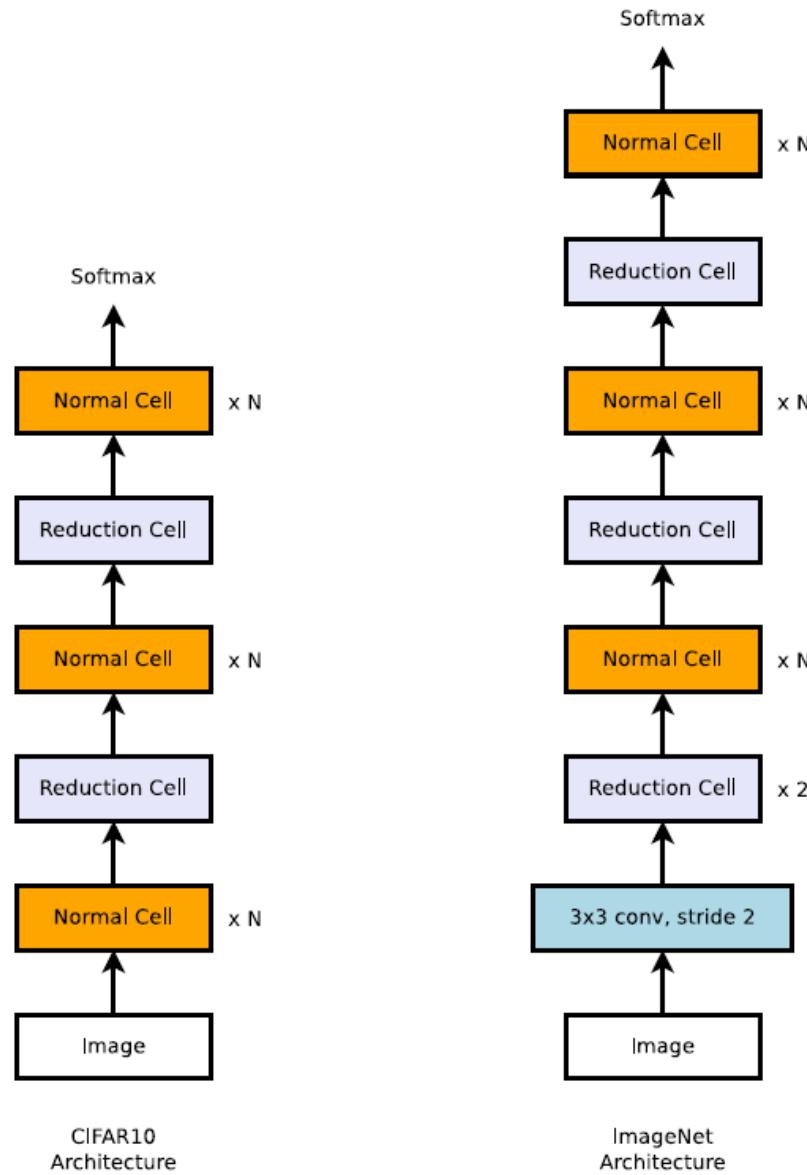
Motivation & Idea

- NAS used 800 GPUs for 28days resulting in **537,600 GPU-hours** → too long (ImageNet dataset)
- Many state-of-the-art models have **repeated modules**
- Searching for a good architecture on the far smaller CIFAR-10 dataset, and **automatically transfer the learned architecture** to ImageNet
- Achieving this transferability by designing a search space so that the complexity of **the architecture is independent of the depth of the network**
- All convolutional networks in search space are composed of convolutional layers(or “**cells**”) with identical structure but different weights

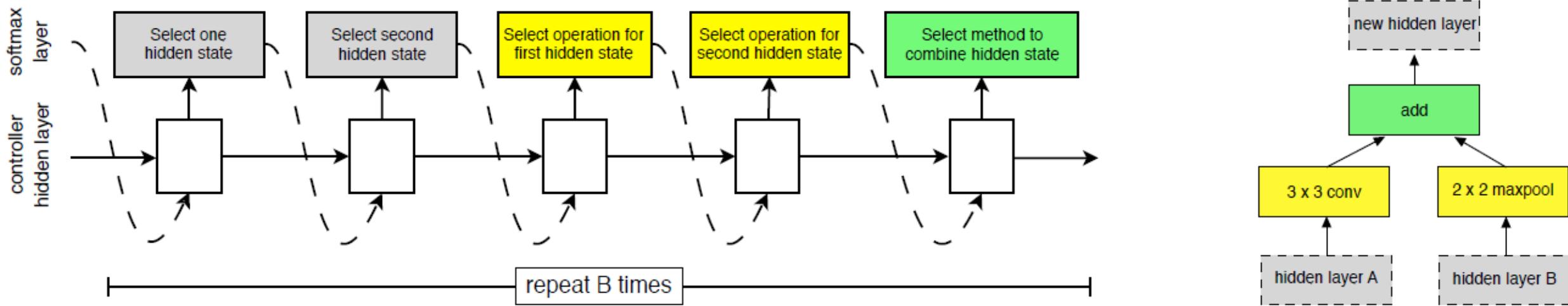
Method

- Overall architectures of the convolutional nets are manually predetermined
 - **Normal Cell** – convolutional cells that return a feature map of the same dimension
 - **Reduction Cell** – convolutional cells that return a feature map where the feature map height and width is reduced by a factor of two
- Using common heuristic to double the number of filters in the output whenever the spatial activation size is reduced

Scalable Architectures for Image Classification

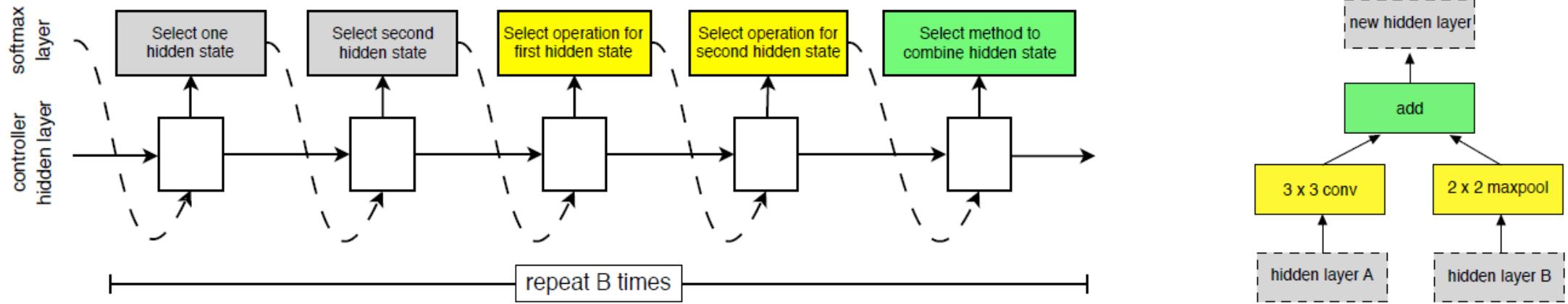


Models and Algorithms



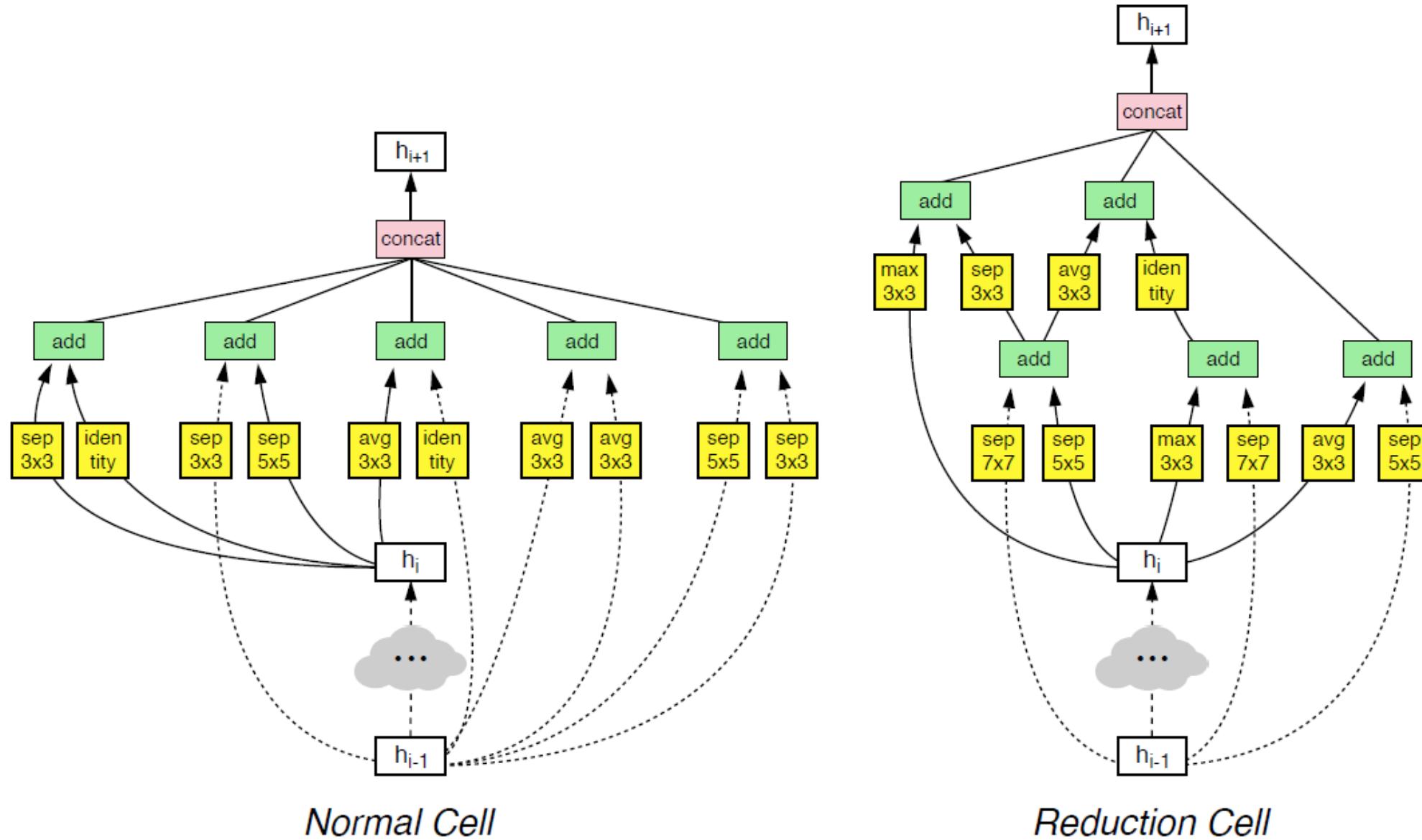
- Step 1.** Select a hidden state from h_i, h_{i-1} or from the set of hidden states created in previous blocks.
- Step 2.** Select a second hidden state from the same options as in Step 1.
- Step 3.** Select an operation to apply to the hidden state selected in Step 1.
- Step 4.** Select an operation to apply to the hidden state selected in Step 2.
- Step 5.** Select a method to combine the outputs of Step 3 and 4 to create a new hidden state.

Search Space in a Cell (Step 3 and 4)

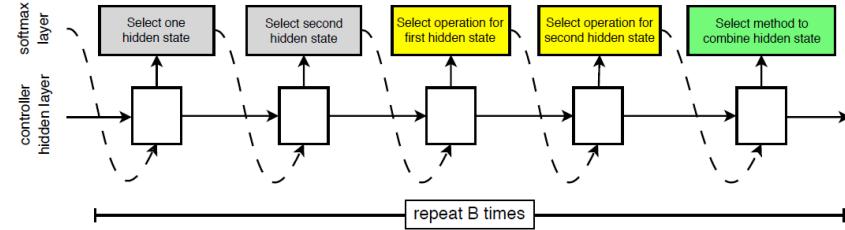


- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-separable conv

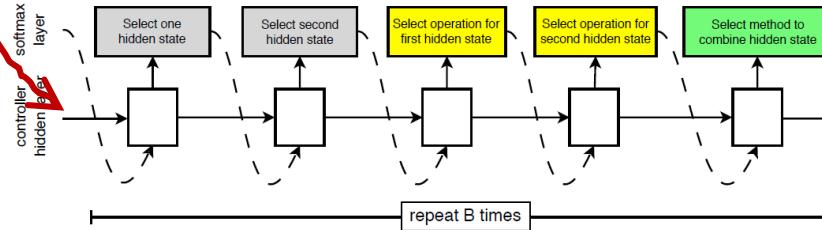
Best Architecture (NASNet-A)



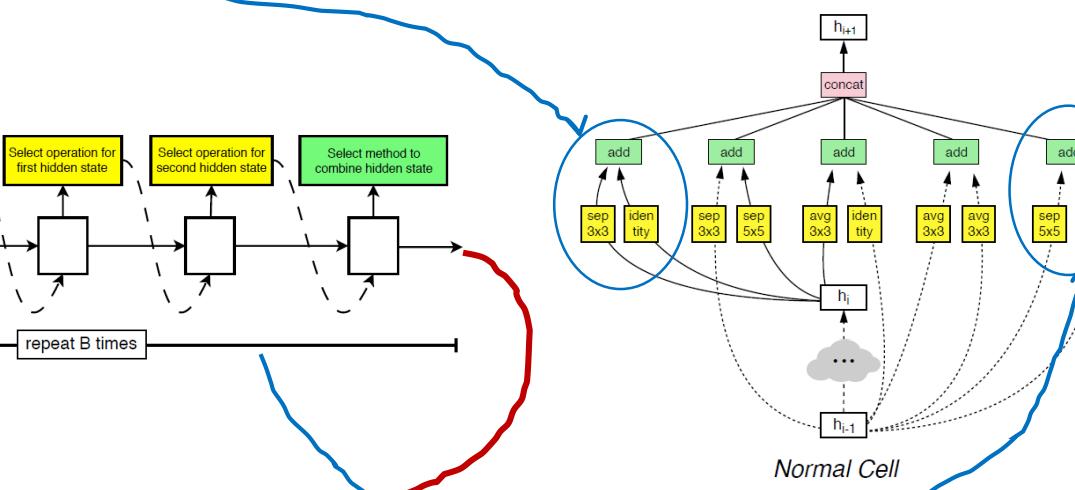
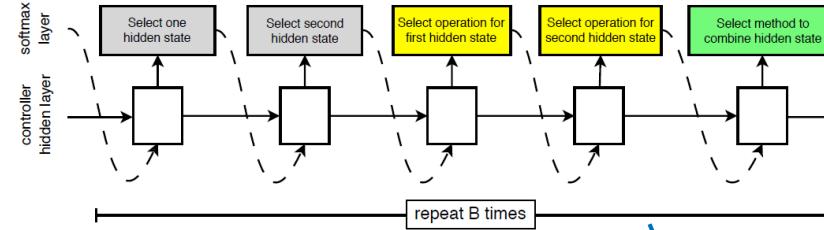
Best Architecture (NASNet-A)



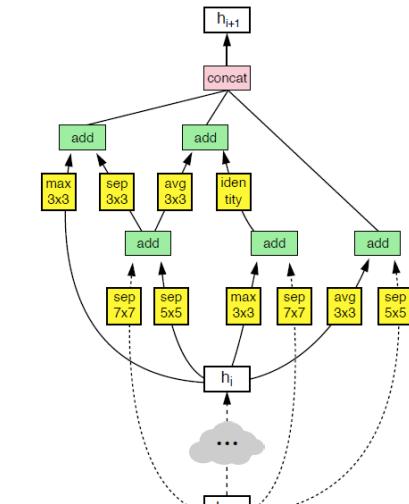
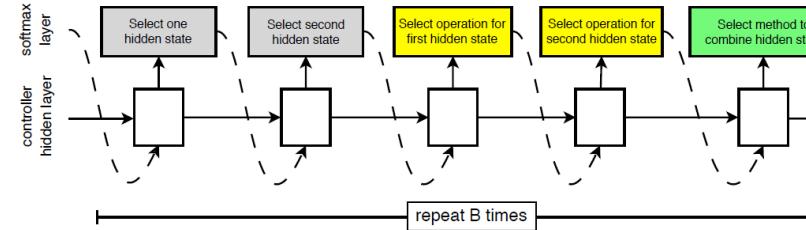
<Normal Cell>



<Reduction Cell>

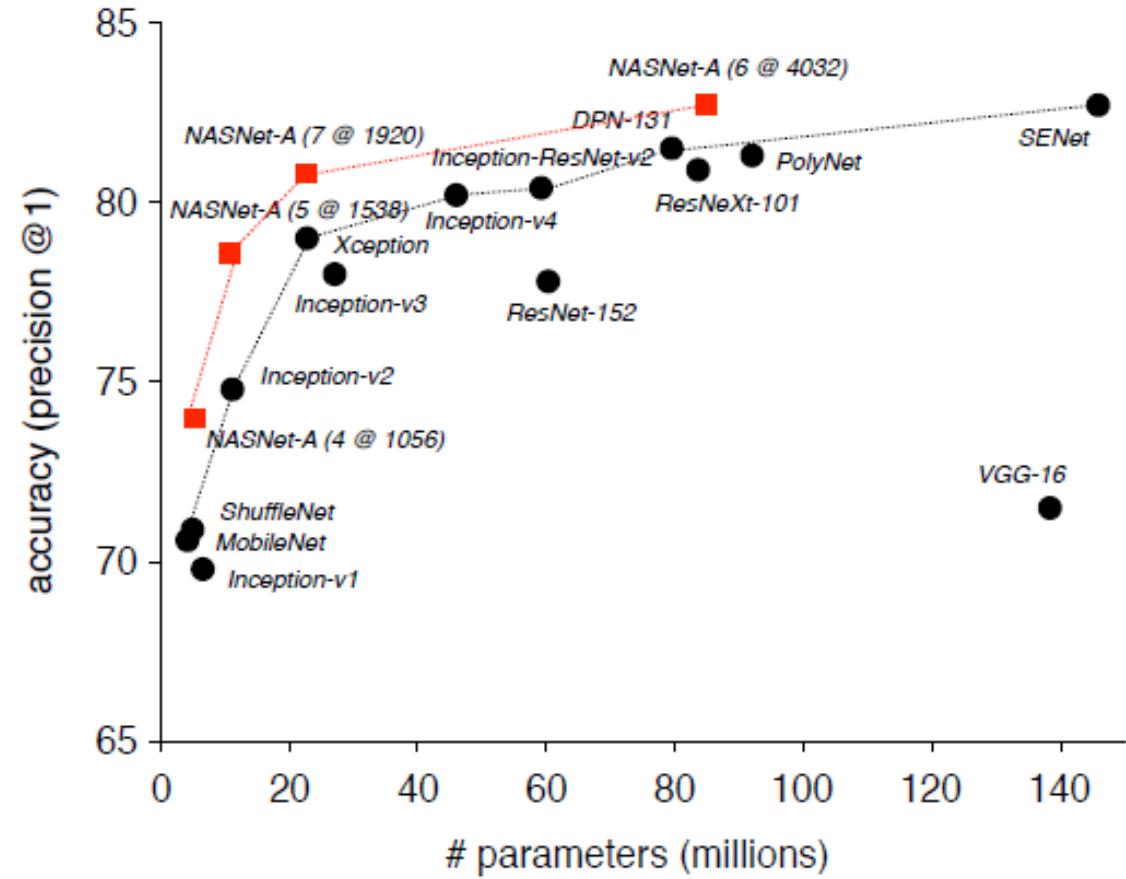
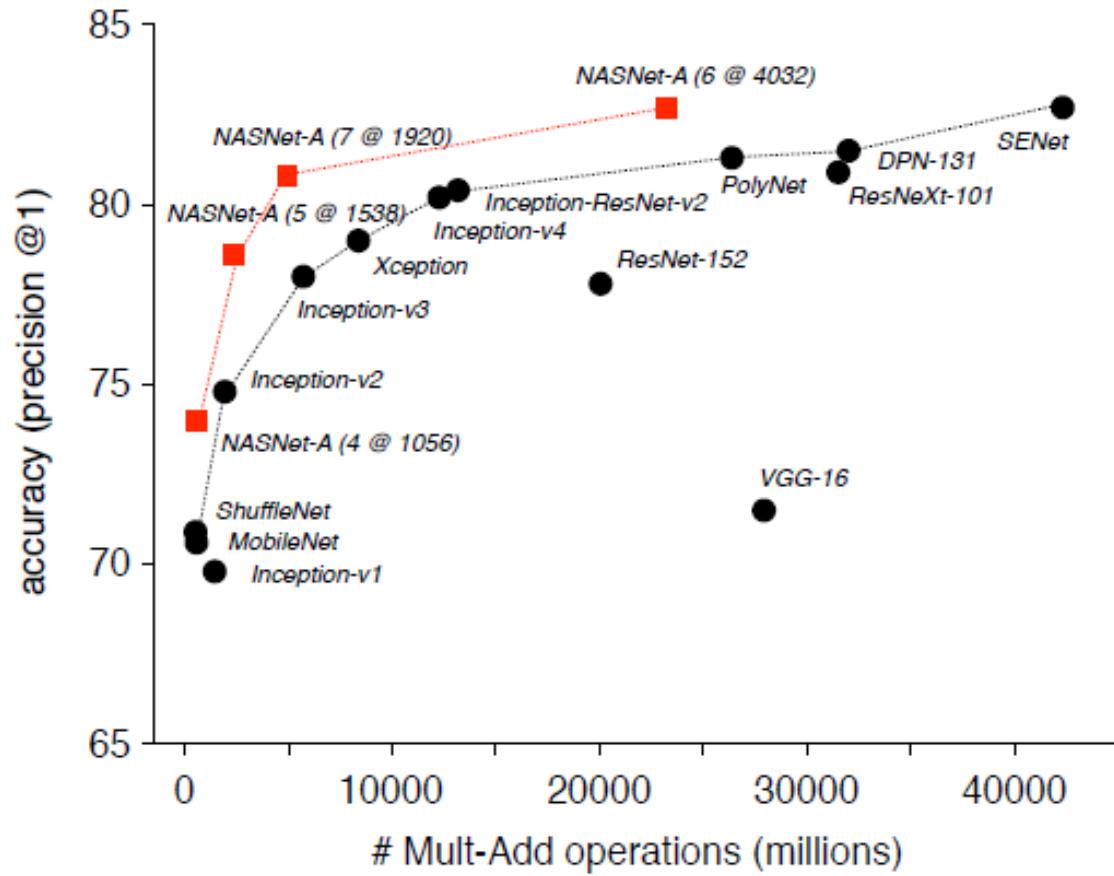


Normal Cell

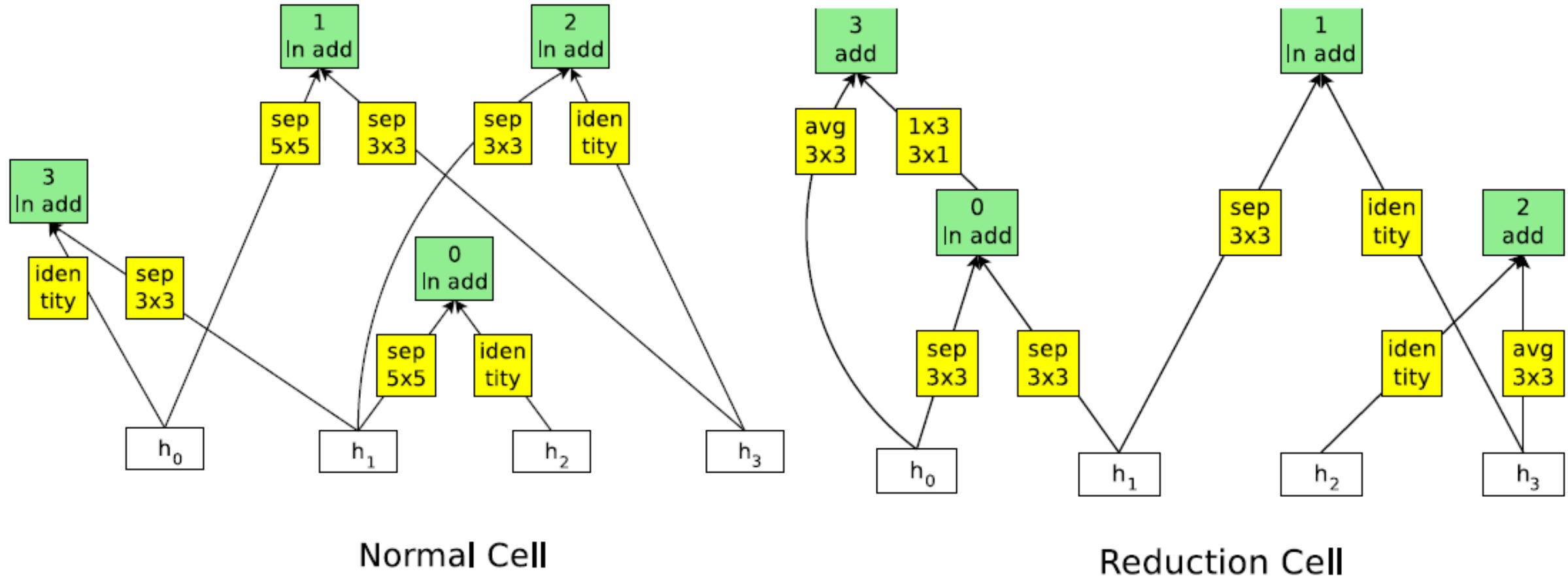


Reduction Cell

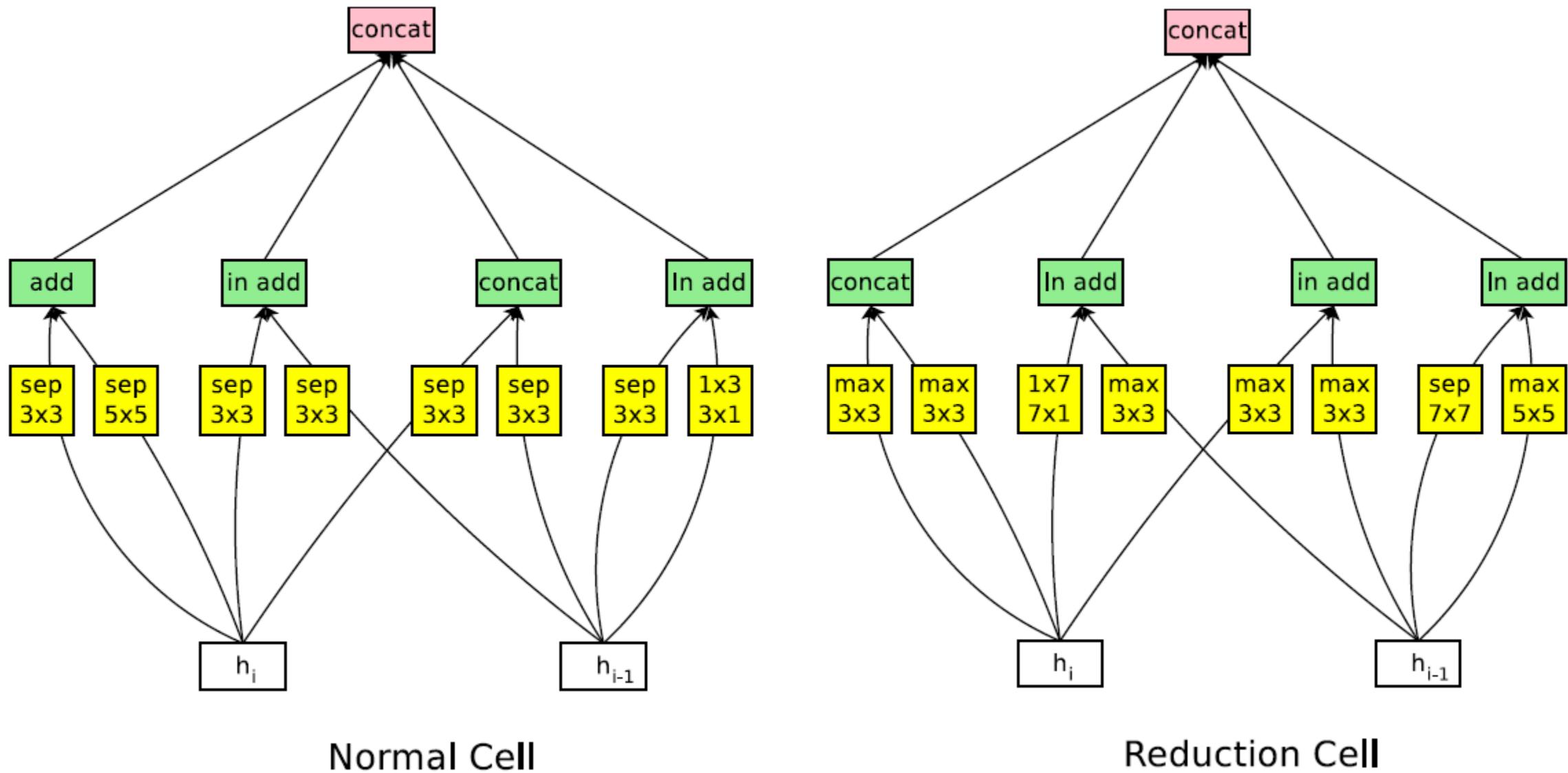
Results on ImageNet



NASNet-B



NASNet-C



Randomly Wired Neural Networks

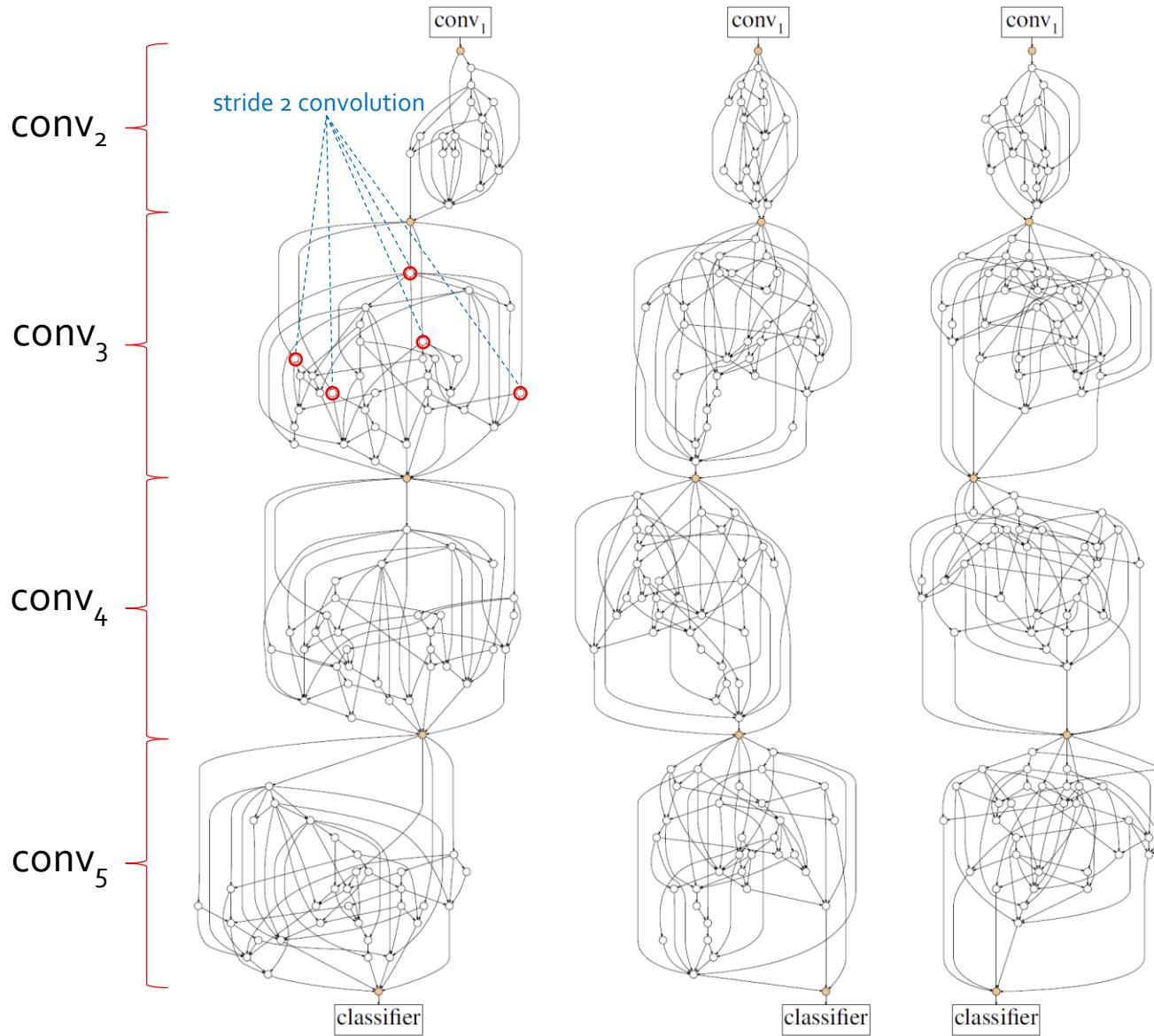


Figure 1. **Randomly wired neural networks** generated by the classical Watts-Strogatz (WS) [50] model: these three instances of random networks achieve (left-to-right) 79.1%, 79.1%, 79.0% classification accuracy on ImageNet under a similar computational budget to ResNet-50, which has 77.1% accuracy.