

Modern CNN

More Accurate or More Efficient



Fast Campus
Start Deep Learning with TensorFlow

DenseNet

.06993v4 [cs.CV] 27 Aug 2017

Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage fea-

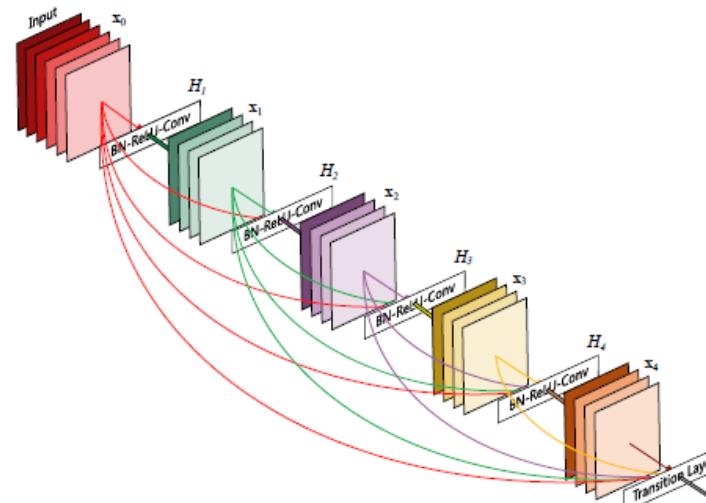
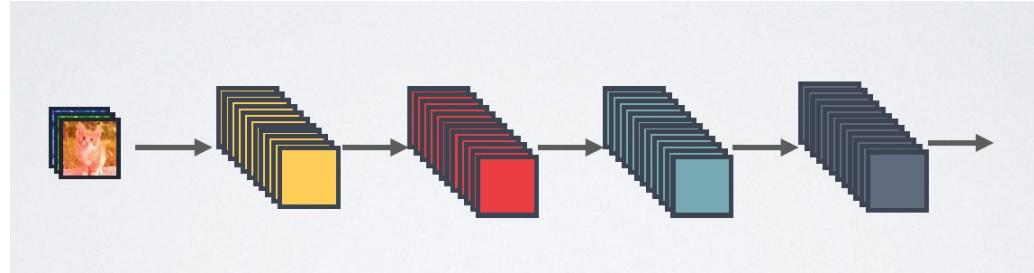


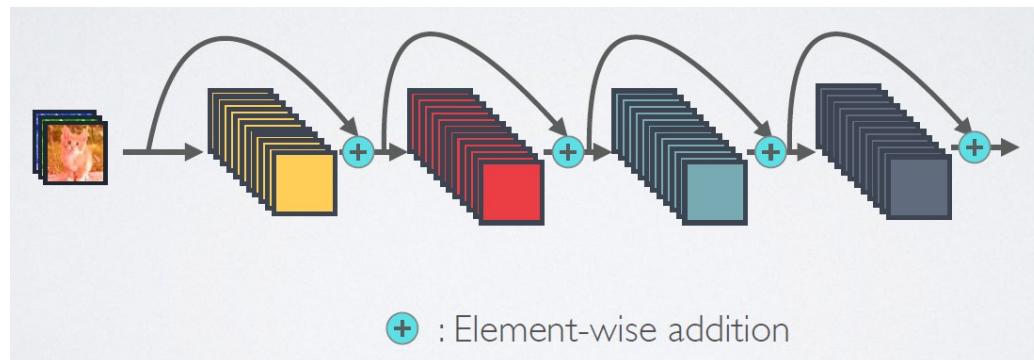
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Dense Connectivity

- Standard Connectivity

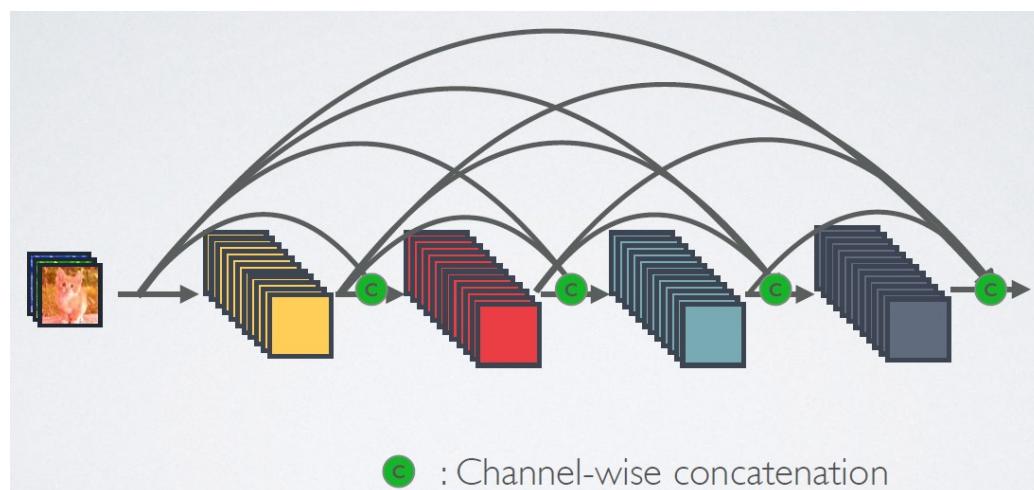


- ResNet Connectivity



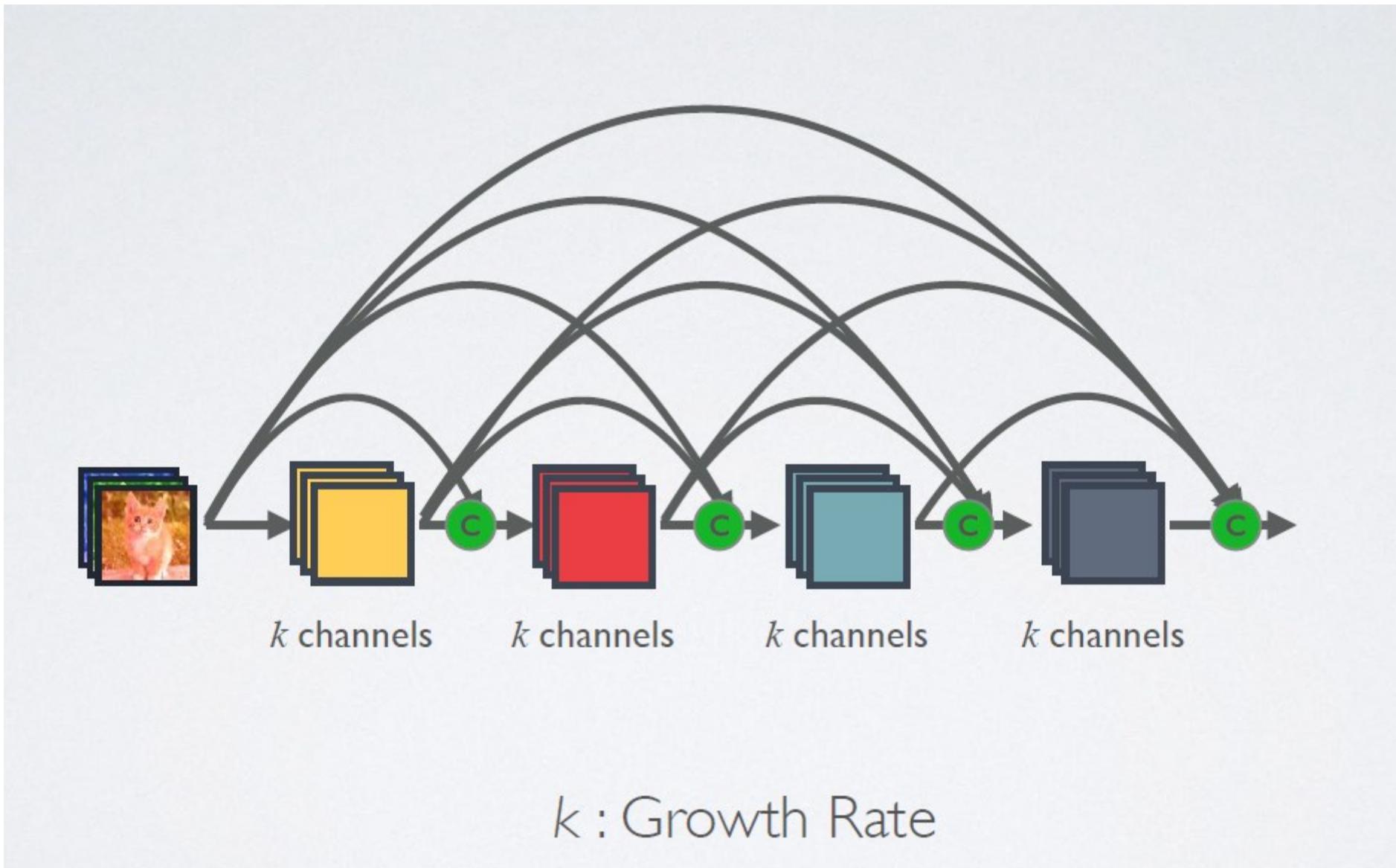
+ : Element-wise addition

- Dense Connectivity

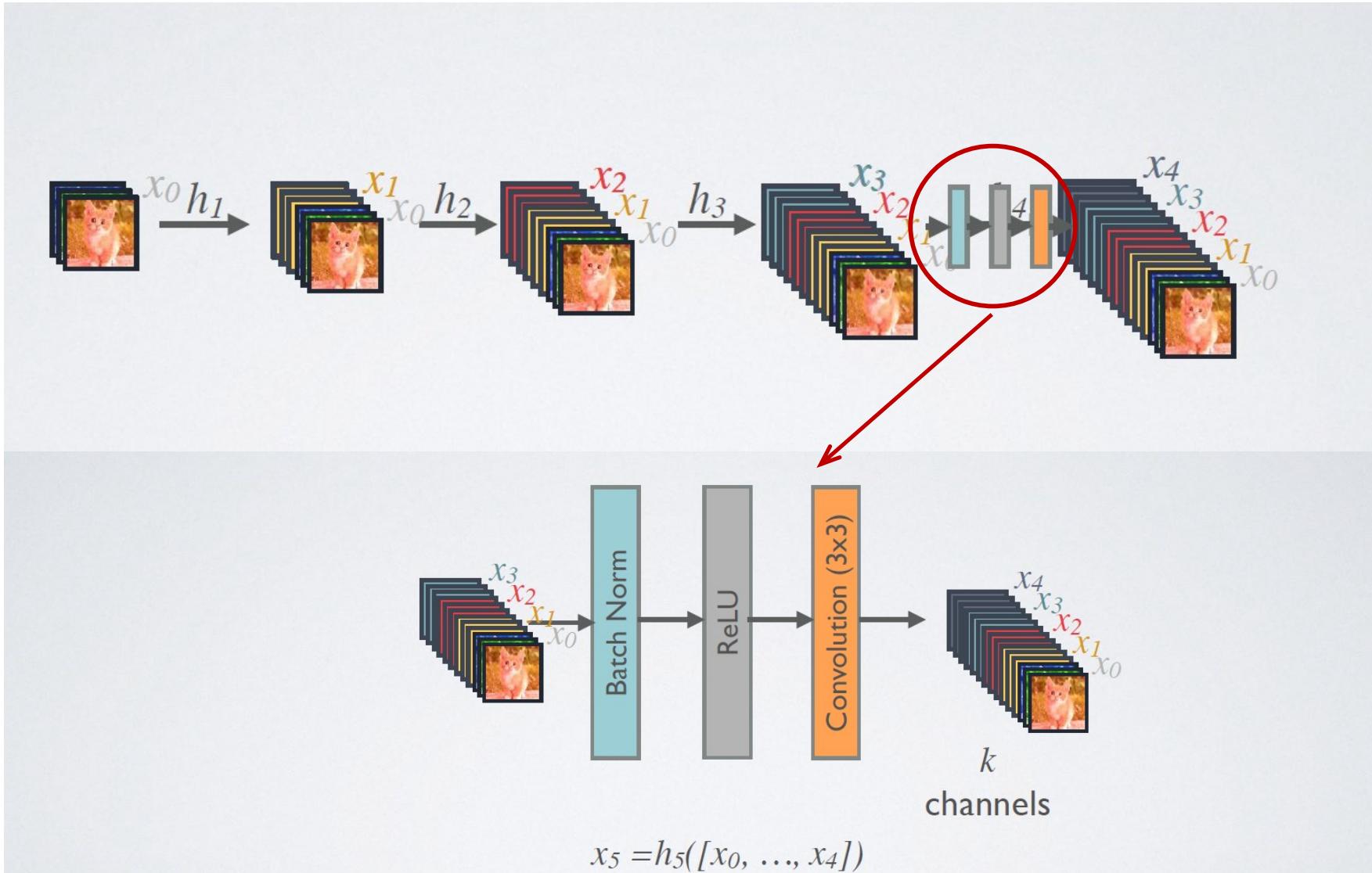


c : Channel-wise concatenation

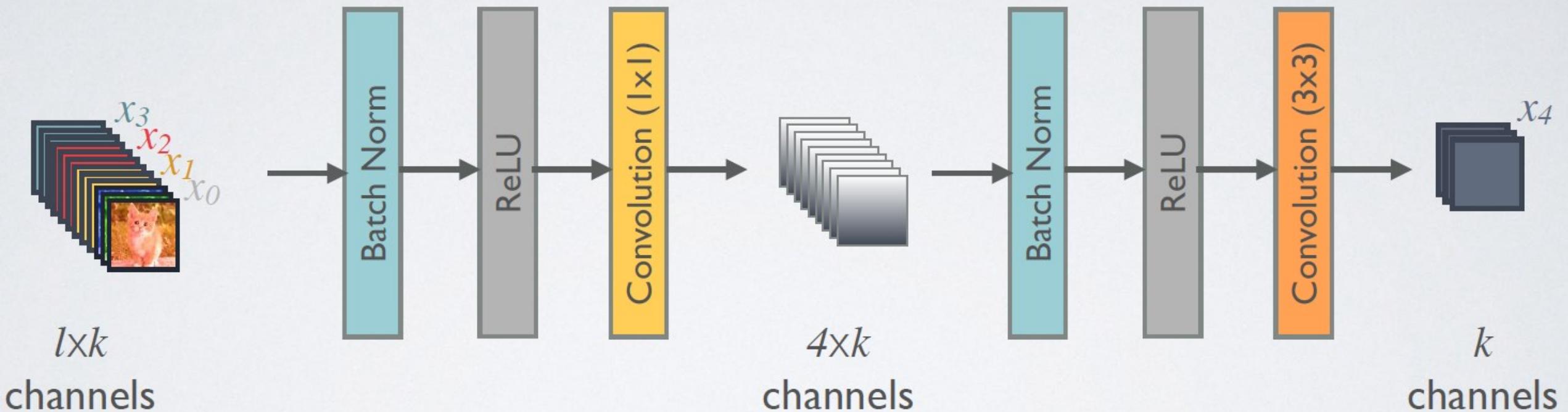
Dense and Slim



Forward Propagation

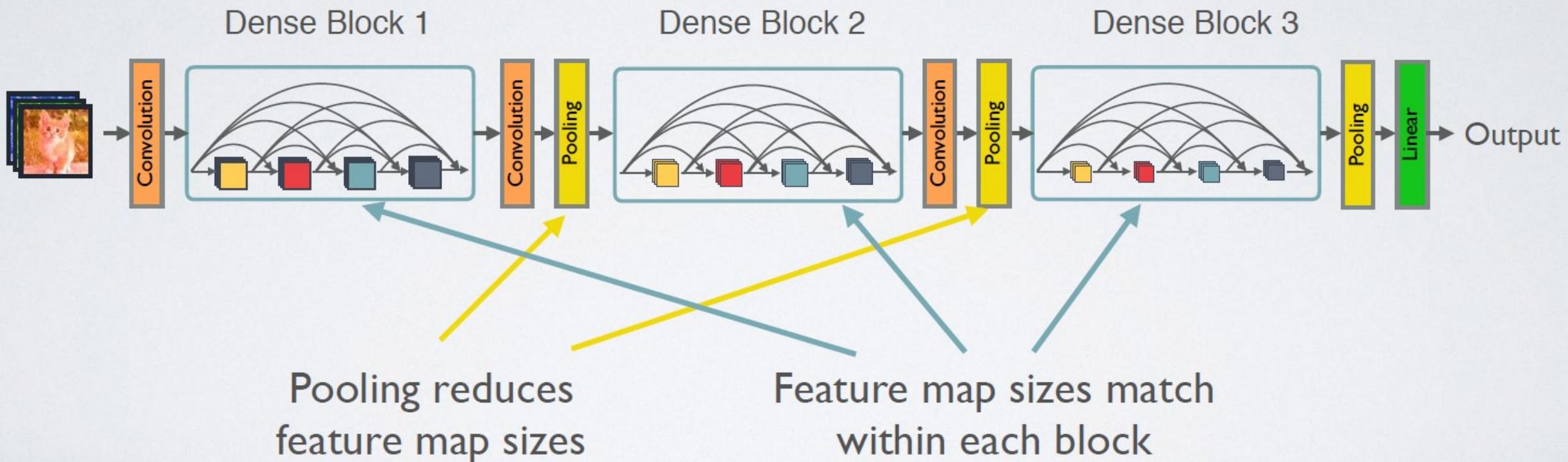


Composite Layer in DenseNet with Bottleneck Layer

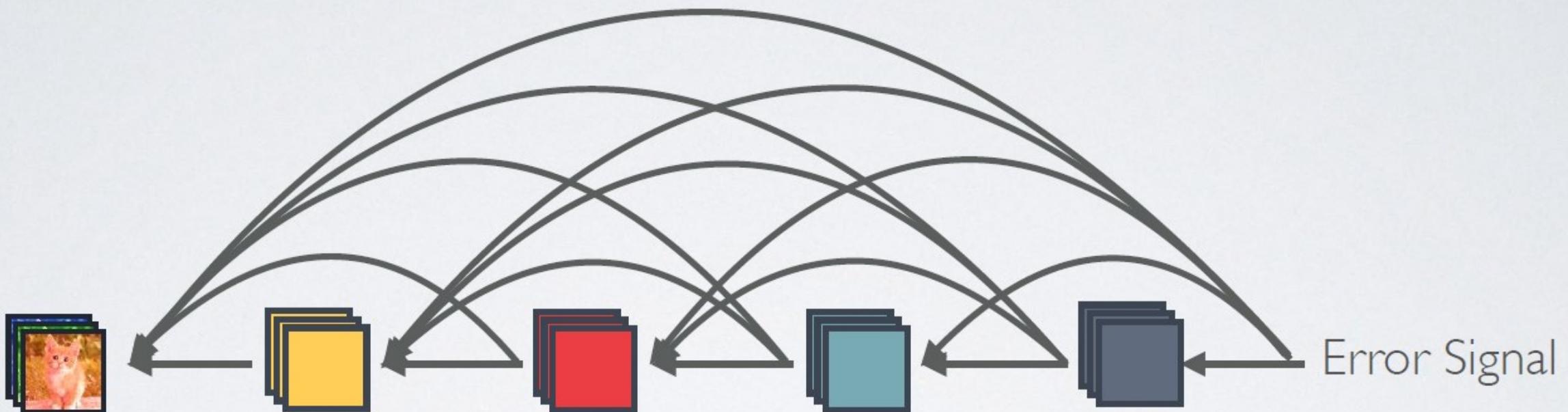


Higher parameter and computational efficiency

DenseNet



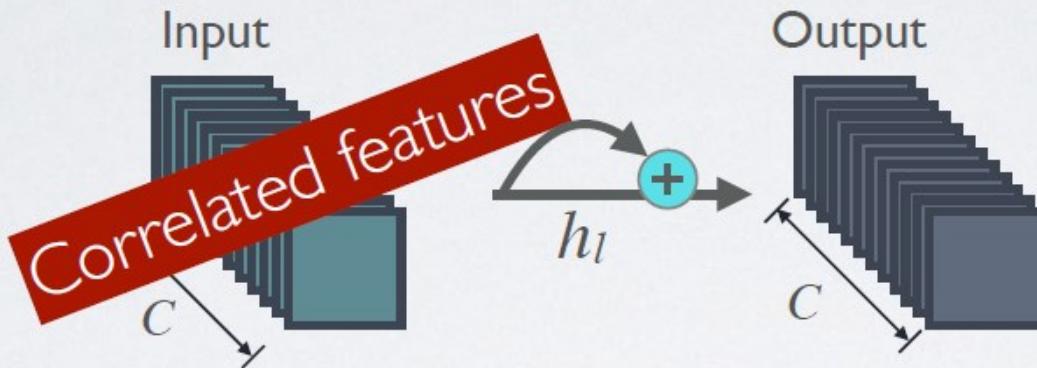
Advantage 1 : Strong Gradient Flow



Implicit “deep supervision”

Advantage 2 : Parameter & Computational Efficiency

ResNet connectivity:

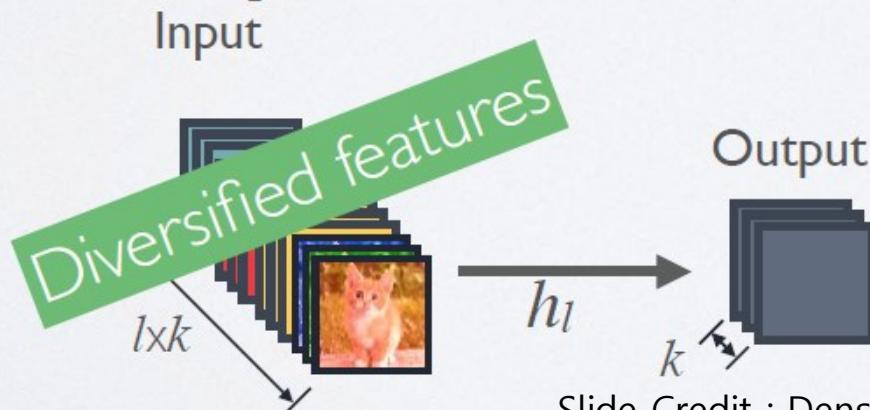


#parameters:

$$O(C \times C)$$

$k \ll C$

DenseNet connectivity:



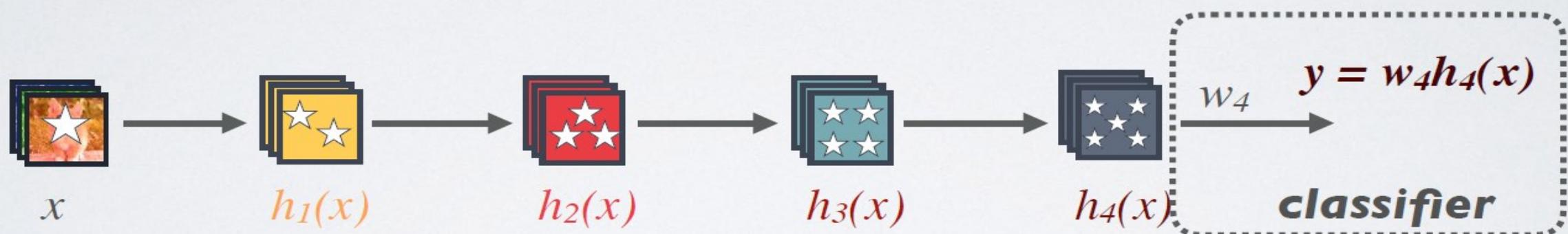
$$O(l \times k \times k)$$

k : Growth rate

Advantage 3 : Maintains Low Complexity Features

Standard Connectivity:

Classifier uses most complex (high level) features

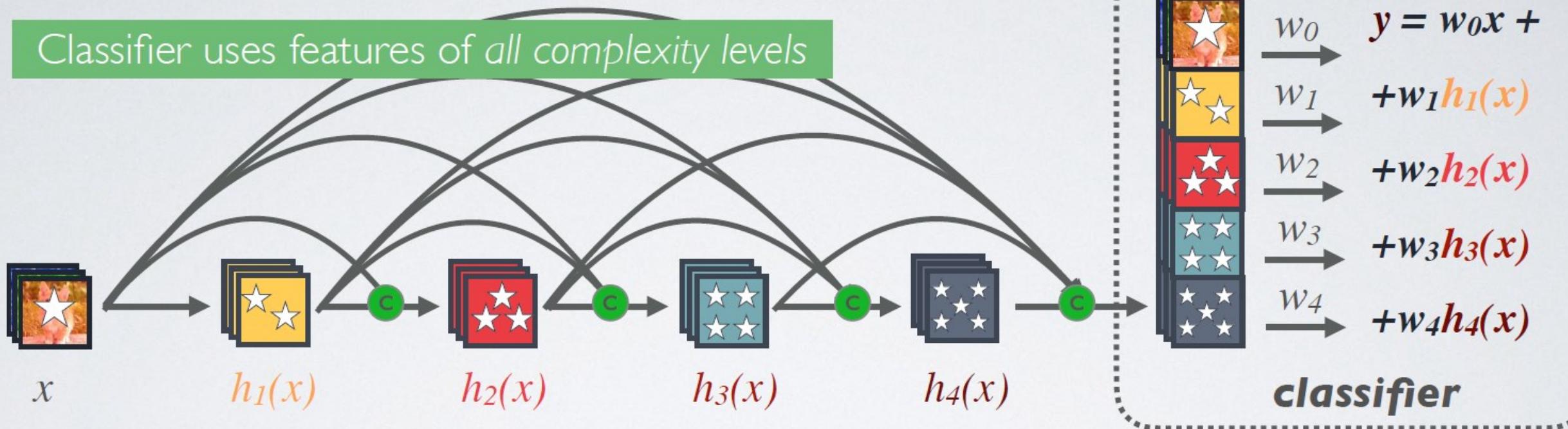


Increasingly complex features



Advantage 3 : Maintains Low Complexity Features

Dense Connectivity:

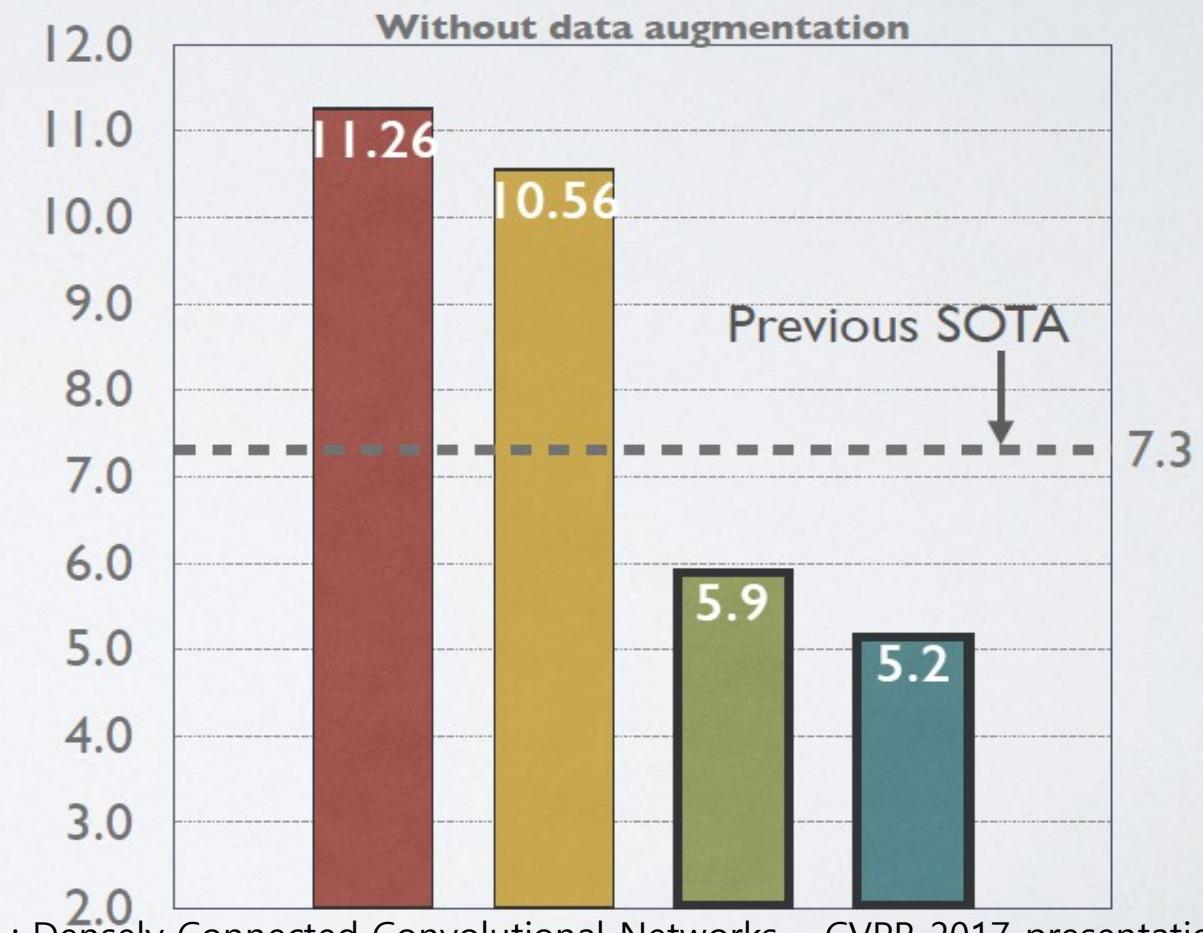
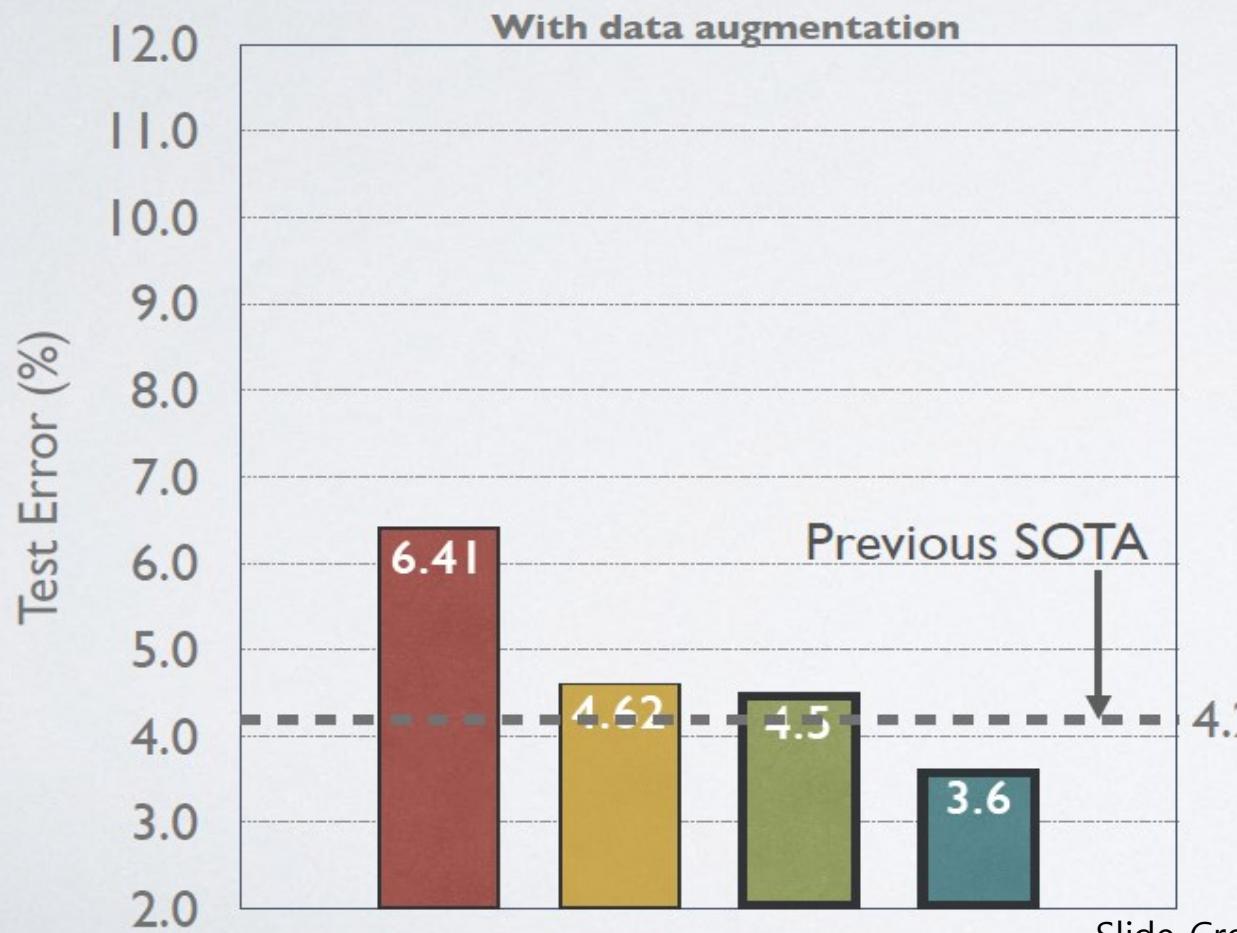


Increasingly complex features



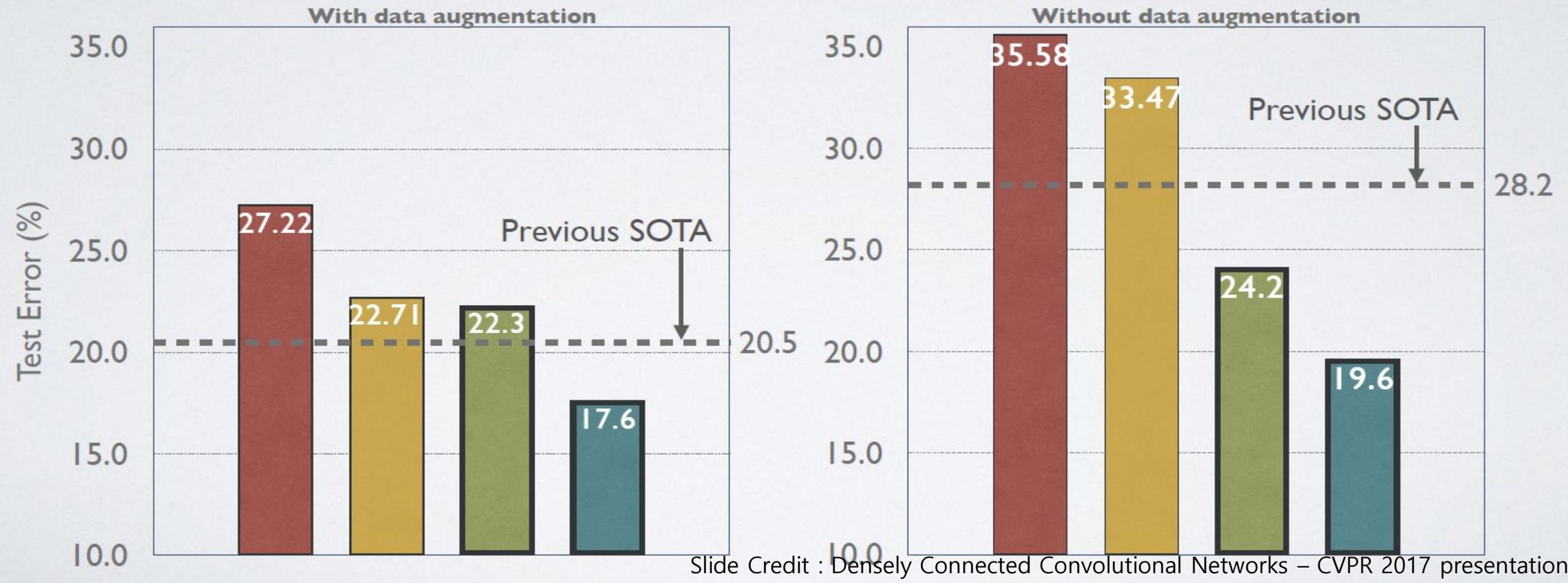
Results on CIFAR-10

ResNet (110 Layers, 1.7 M) ResNet (1001 Layers, 10.2 M)
DenseNet (100 Layers, 0.8 M) DenseNet (250 Layers, 15.3 M)

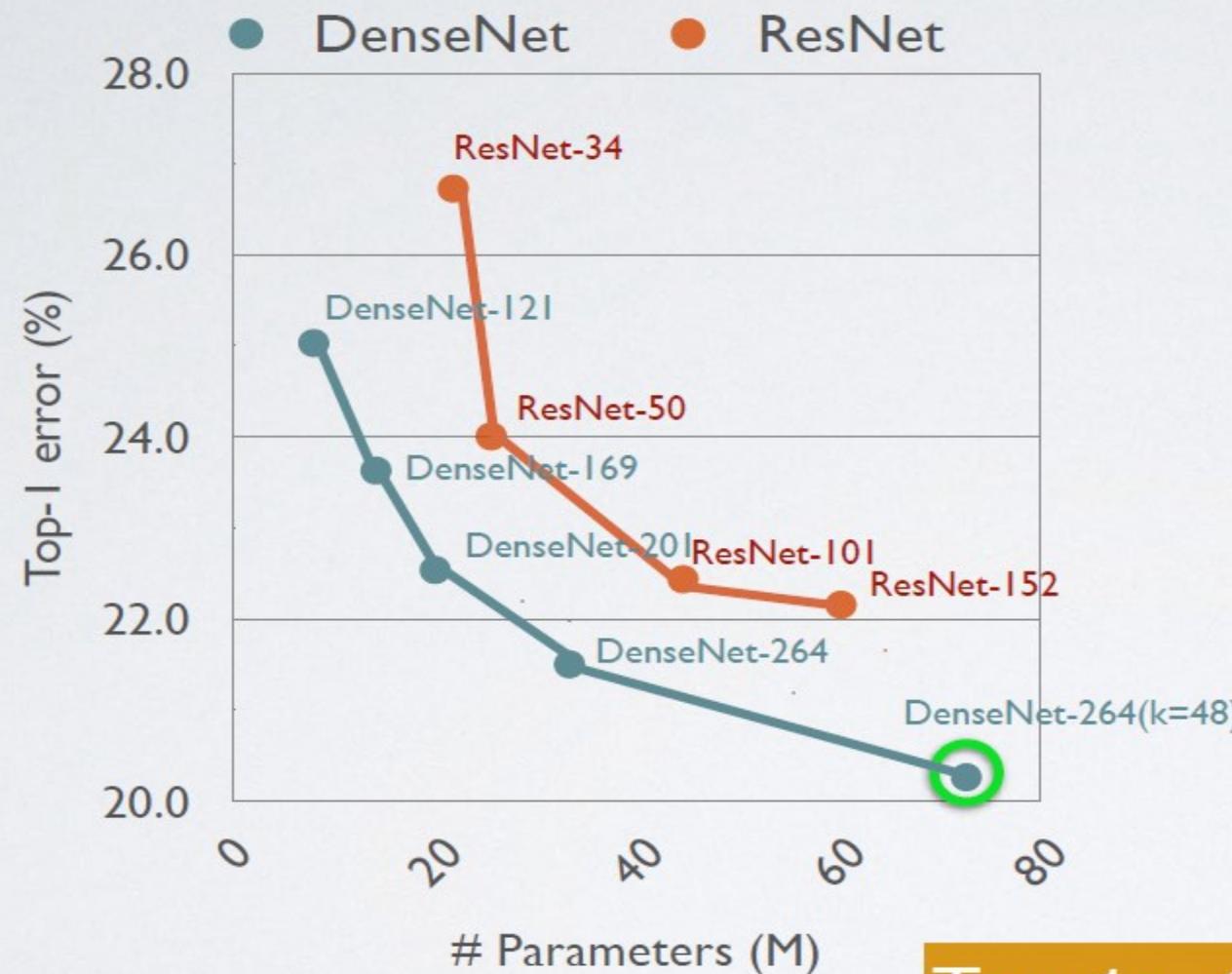


Results on CIFAR-100

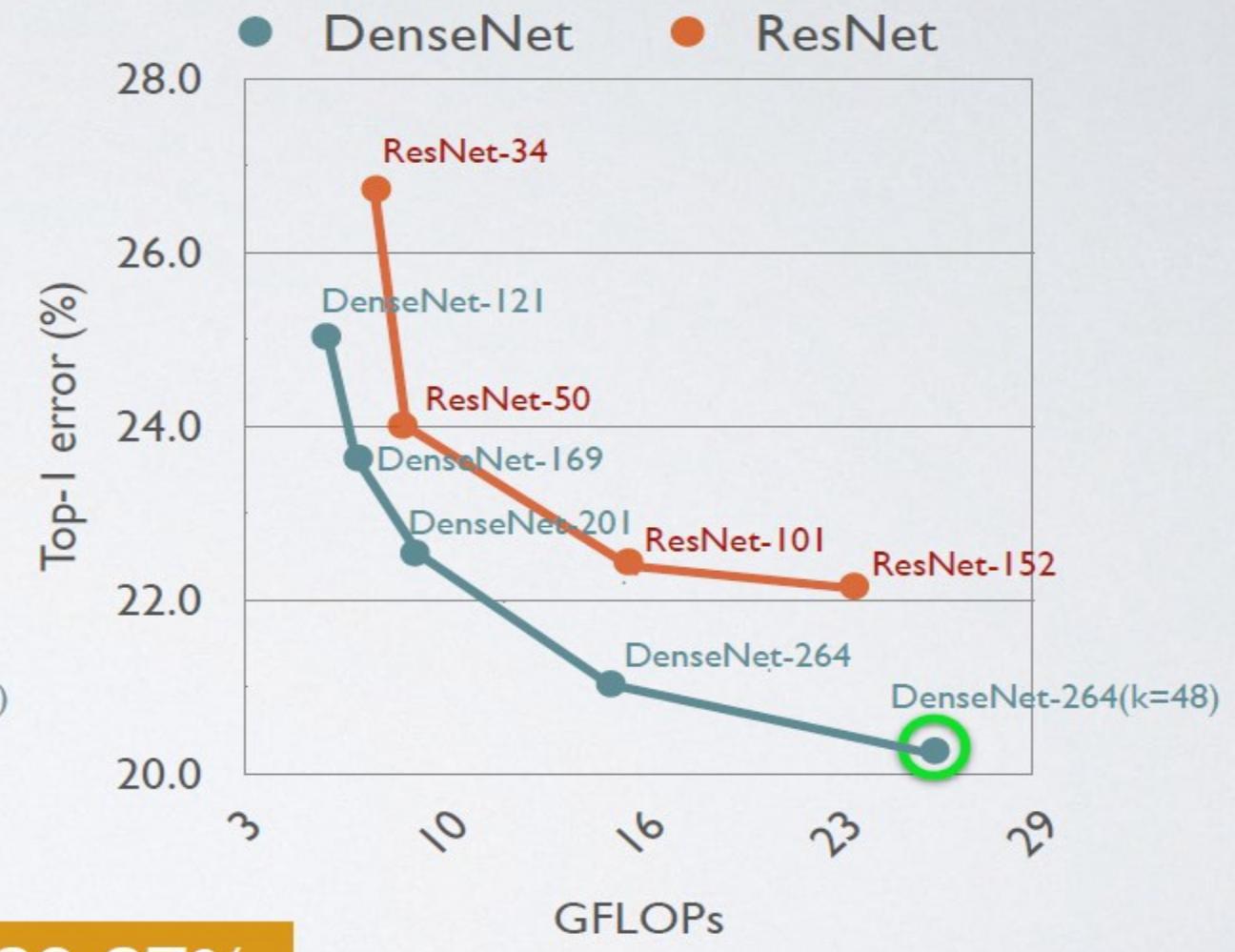
ResNet (110 Layers, 1.7 M) ResNet (1001 Layers, 10.2 M)
DenseNet (100 Layers, 0.8 M) DenseNet (250 Layers, 15.3 M)



Results on ImageNet



Top-1: 20.27%
Top-5: 5.17%



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

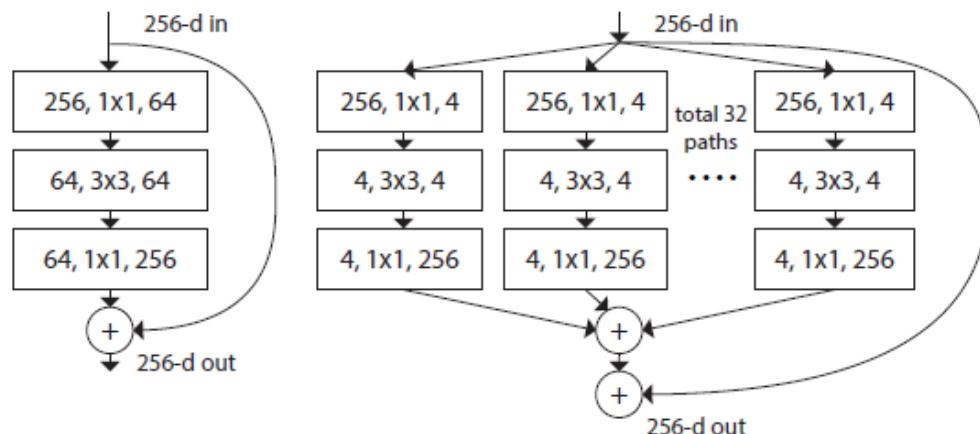


Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ImageNet 2016 Results

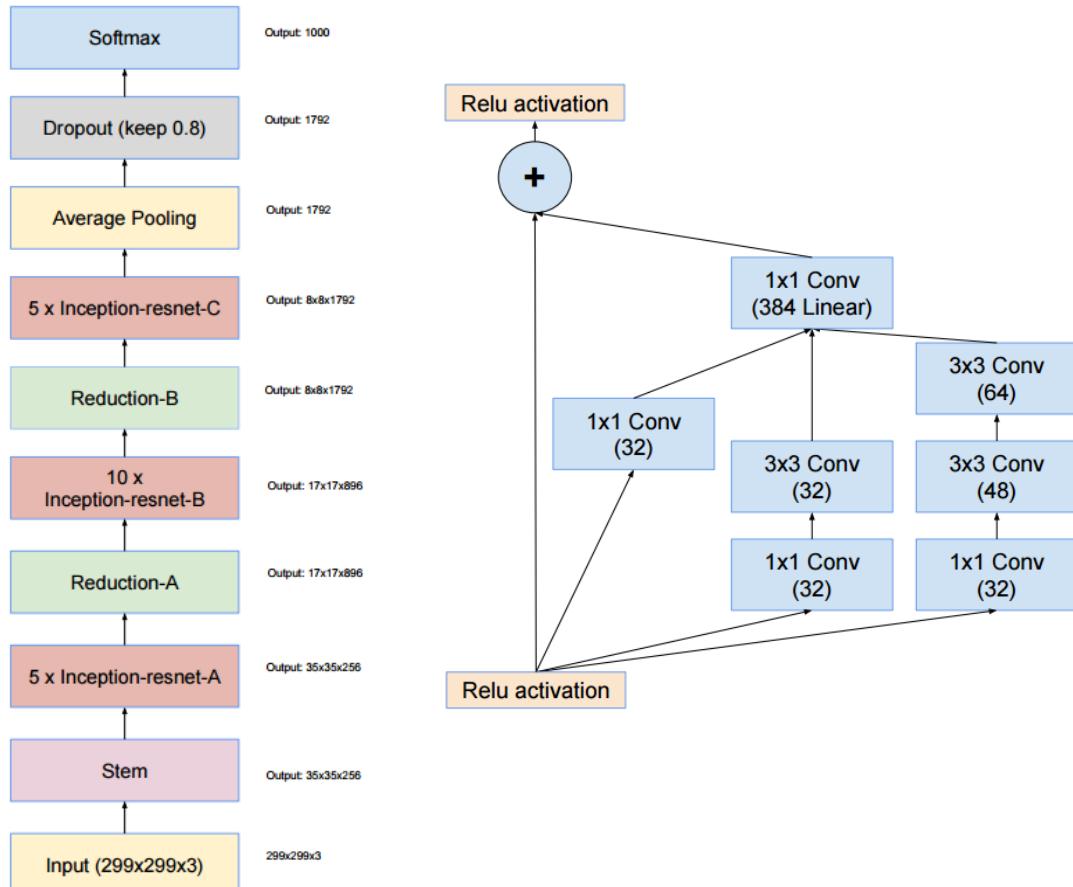
Team name	Entry description	Classification error	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink	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

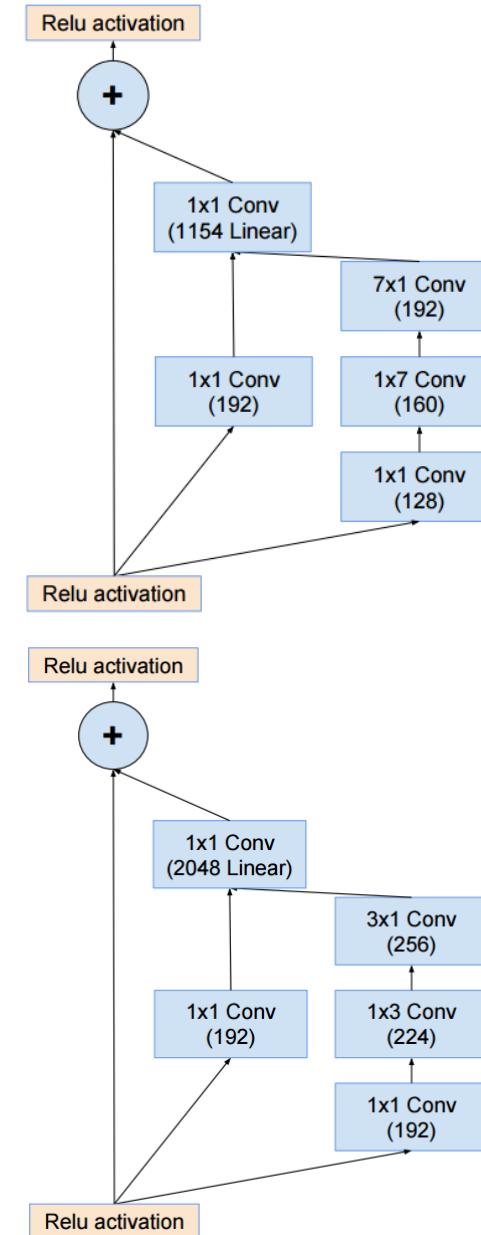
- VGGNet exhibit a simple yet effective strategy of constructing very deep network – **stacking building blocks of the same shape**
- ResNet inherit this strategy with **stacking modules of the same topology**
- Unlike VGGNet, the family of Inception models have demonstrated that carefully designed topologies 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

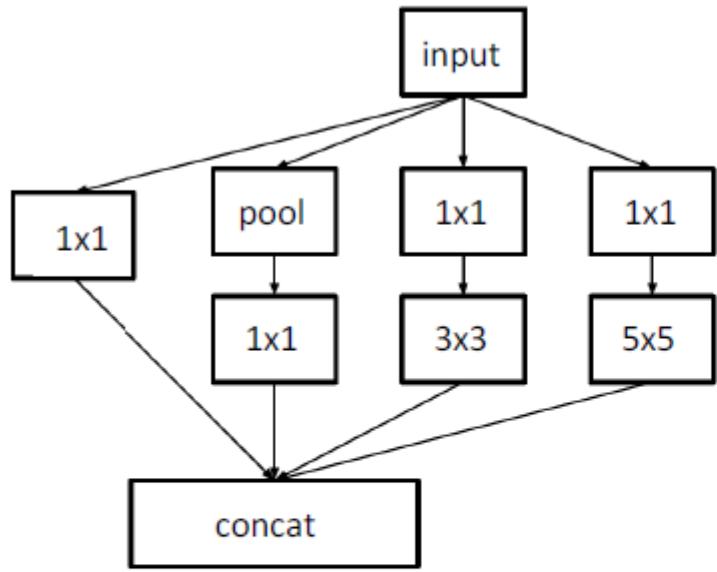


"Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning"



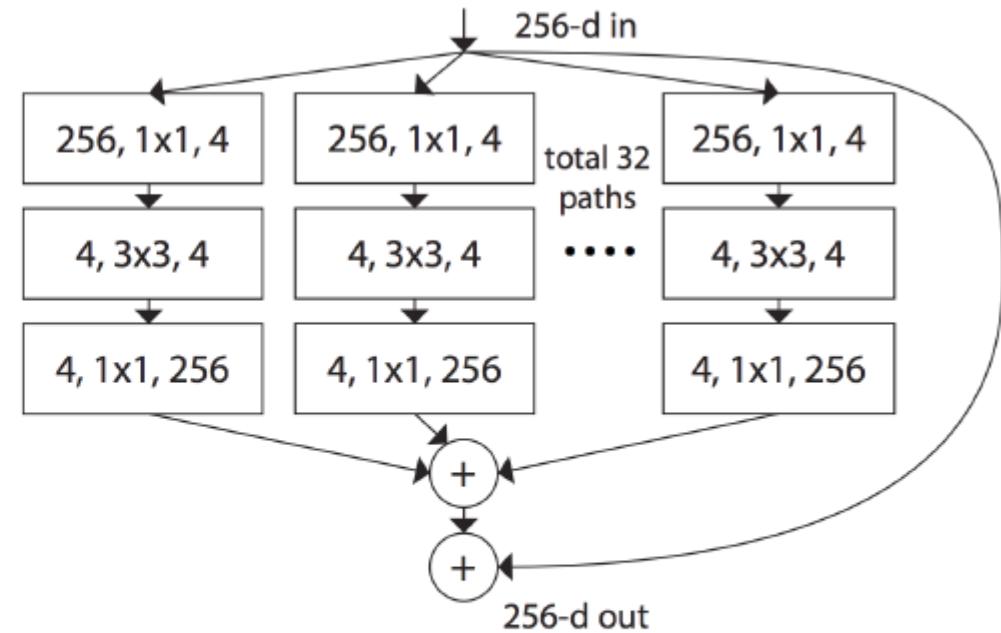
ResNet Learns Inception??

- Multi-branch + ResNet



Inception:

heterogeneous multi-branch

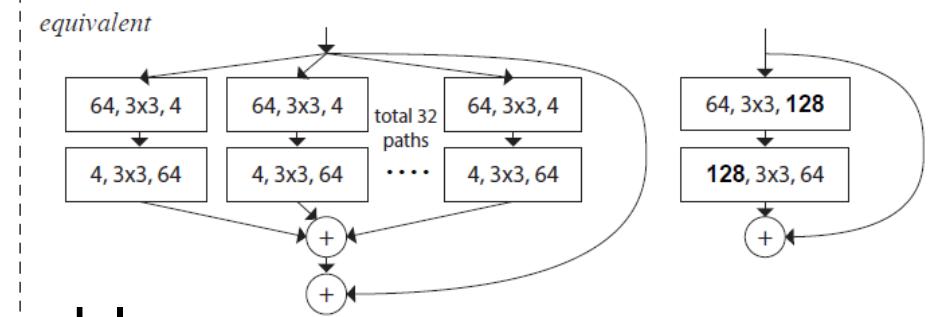


ResNeXt:

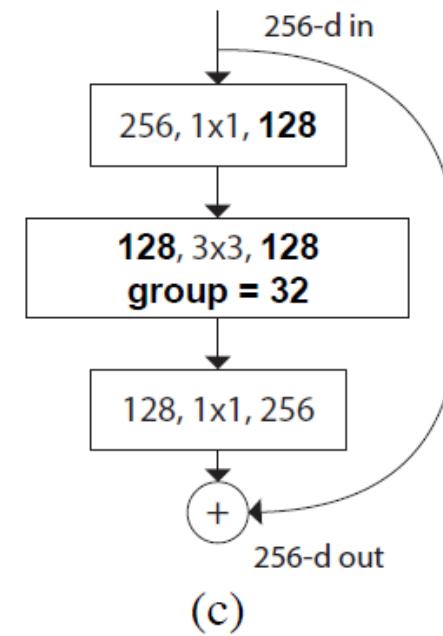
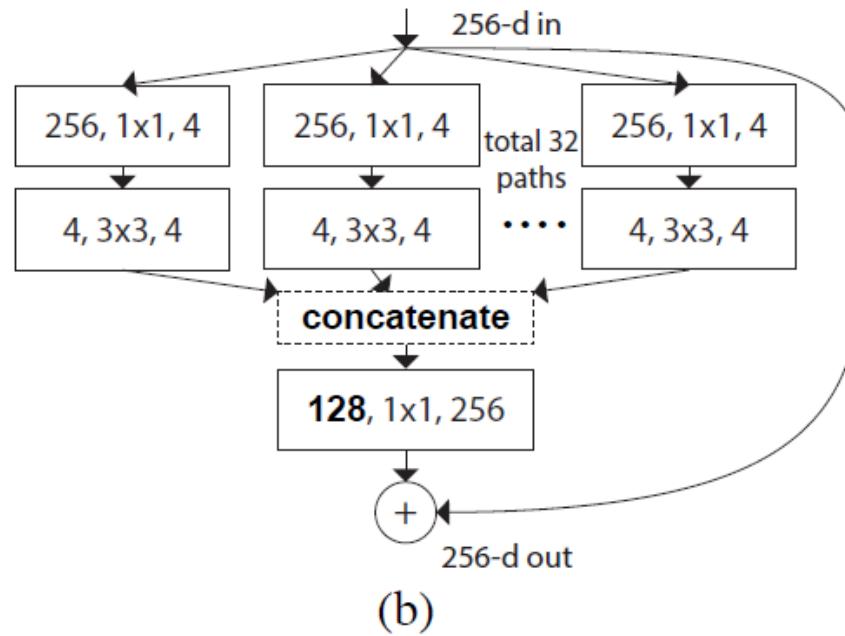
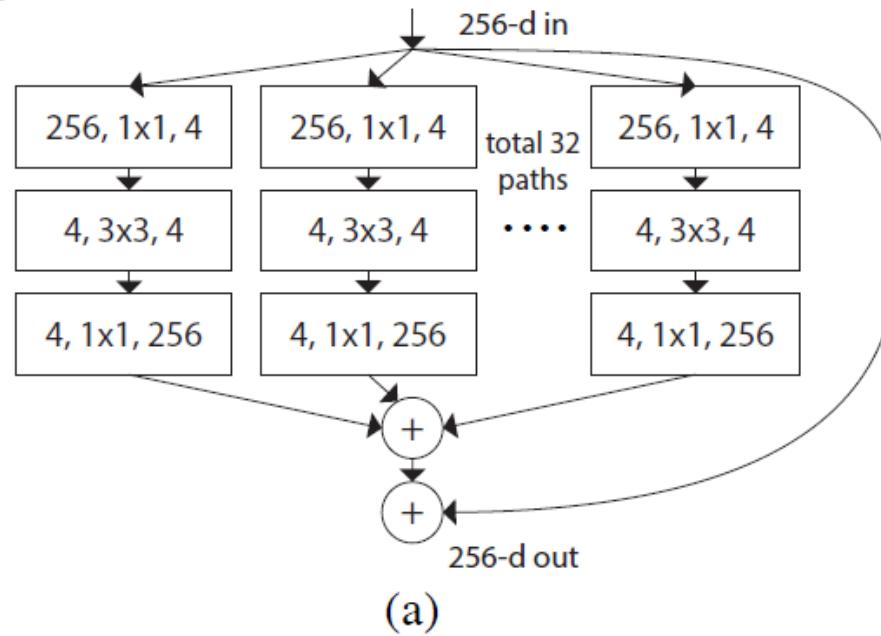
uniform multi-branch

ResNeXt

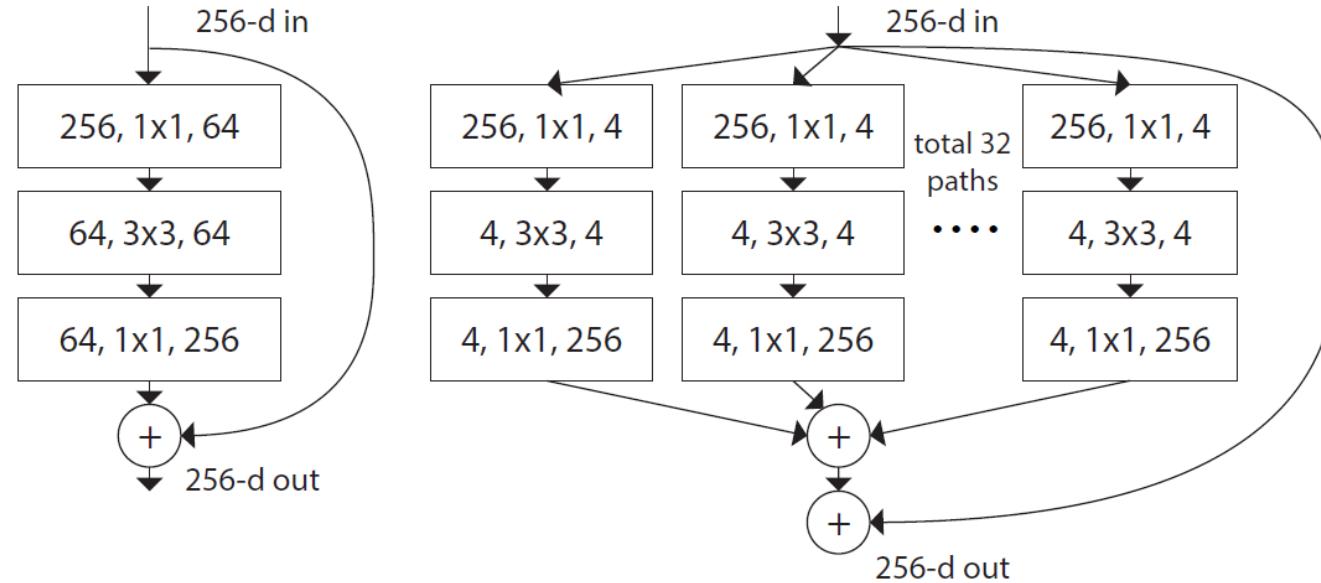
- Concatenation and Addition are interchangeable
- Uniform multi-branching can be done by group-conv



equivalent

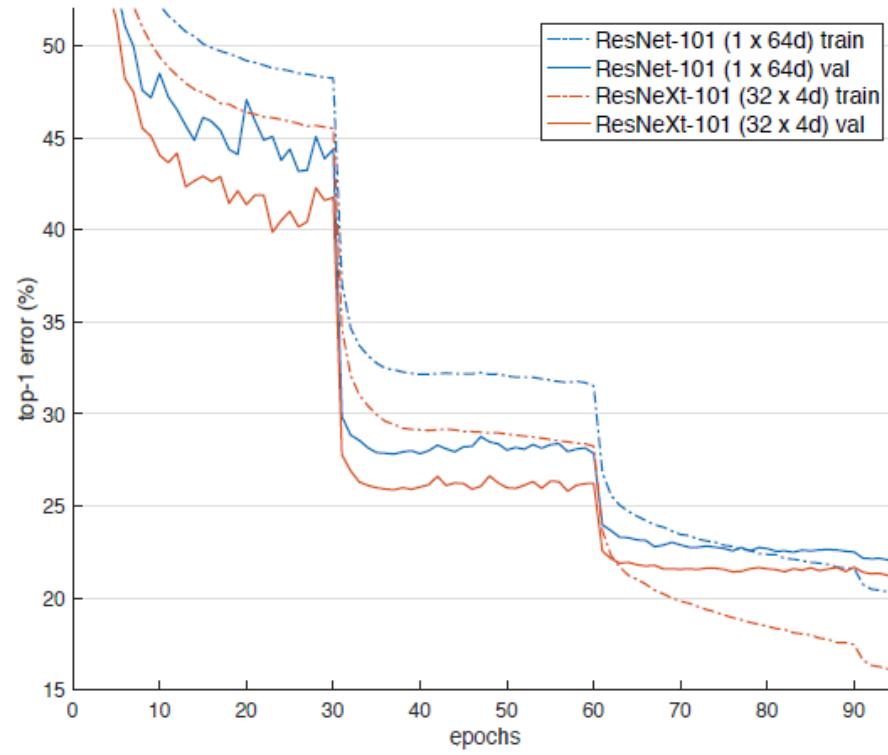
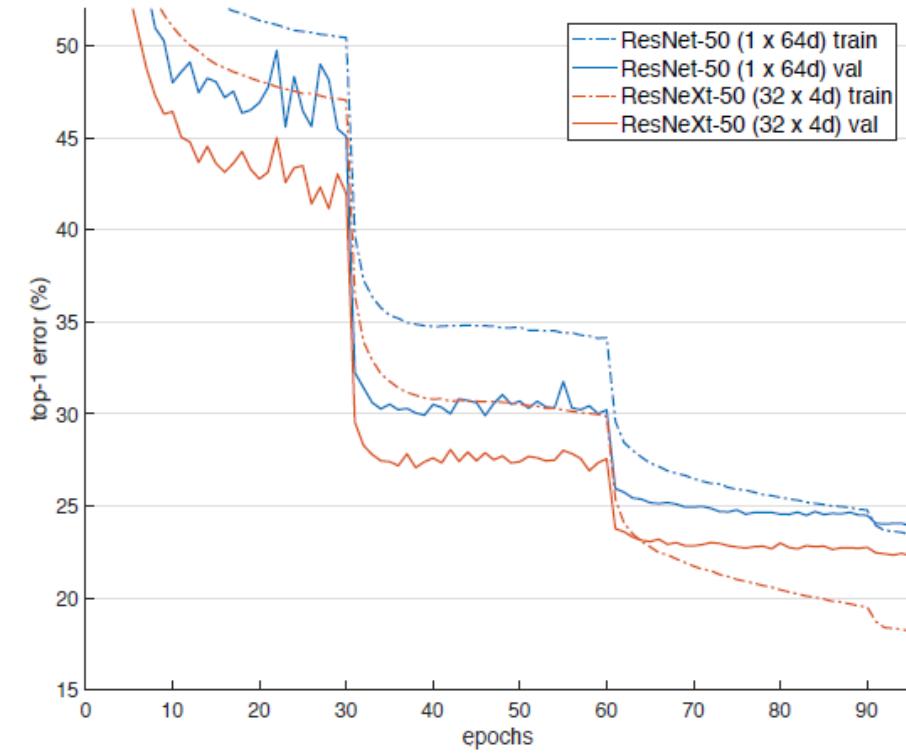


Model Capacity (# of parameters)



- 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

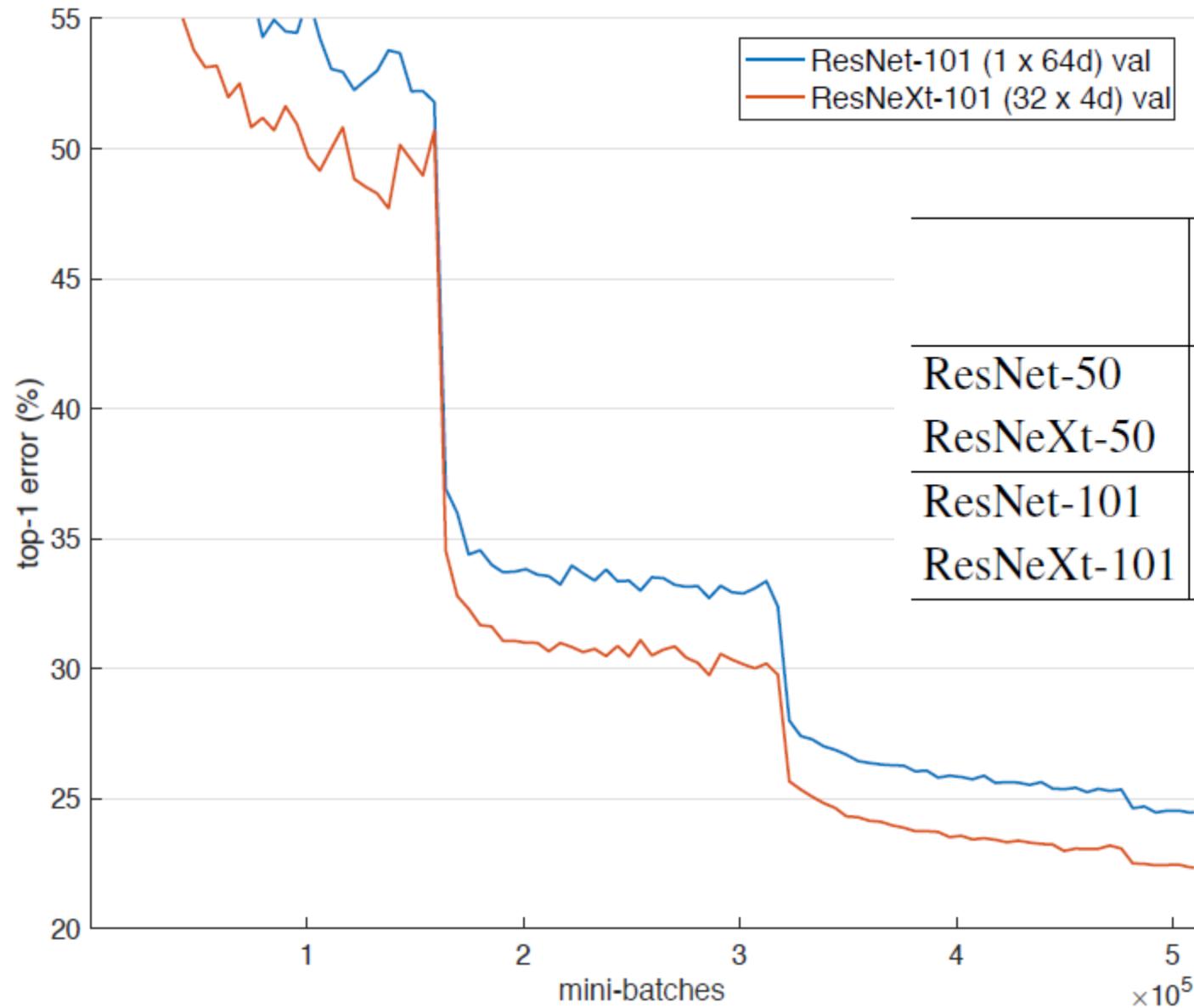
Results – Increasing Cardinality vs Deeper/Wider

	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

Results – vs SOTA Networks

	224×224		320×320 / 299×299	
	top-1 err	top-5 err	top-1 err	top-5 err
ResNet-101 [14]	22.0	6.0	-	-
ResNet-200 [15]	21.7	5.8	20.1	4.8
Inception-v3 [39]	-	-	21.2	5.6
Inception-v4 [37]	-	-	20.0	5.0
Inception-ResNet-v2 [37]	-	-	19.9	4.9
ResNeXt-101 (64 × 4d)	20.4	5.3	19.1	4.4

Results – ImageNet-5K



	setting	5K-way classification		1K-way classification	
		top-1	top-5	top-1	top-5
ResNet-50	1 × 64d	45.5	19.4	27.1	8.2
ResNeXt-50	32 × 4d	42.3	16.8	24.4	6.6
ResNet-101	1 × 64d	42.4	16.9	24.2	6.8
ResNeXt-101	32 × 4d	40.1	15.1	22.2	5.7

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>.

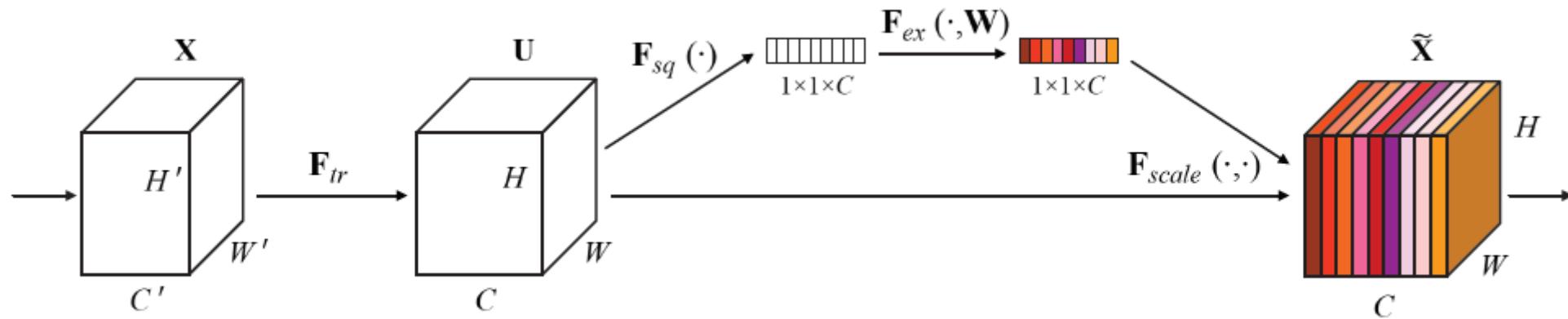
Index Terms—Squeeze-and-Excitation, Image classification, Convolutional Neural Network.

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**

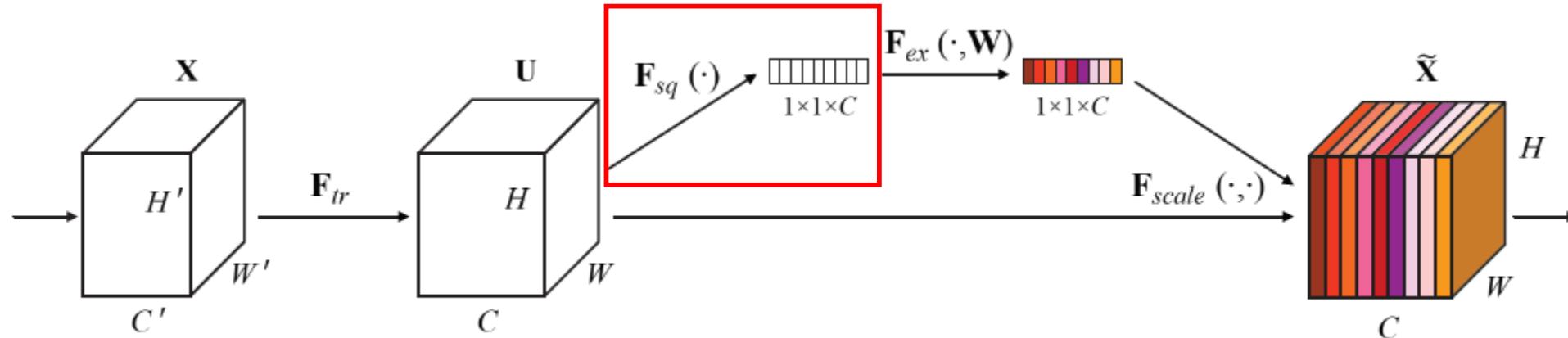


Standard Convolution

- 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

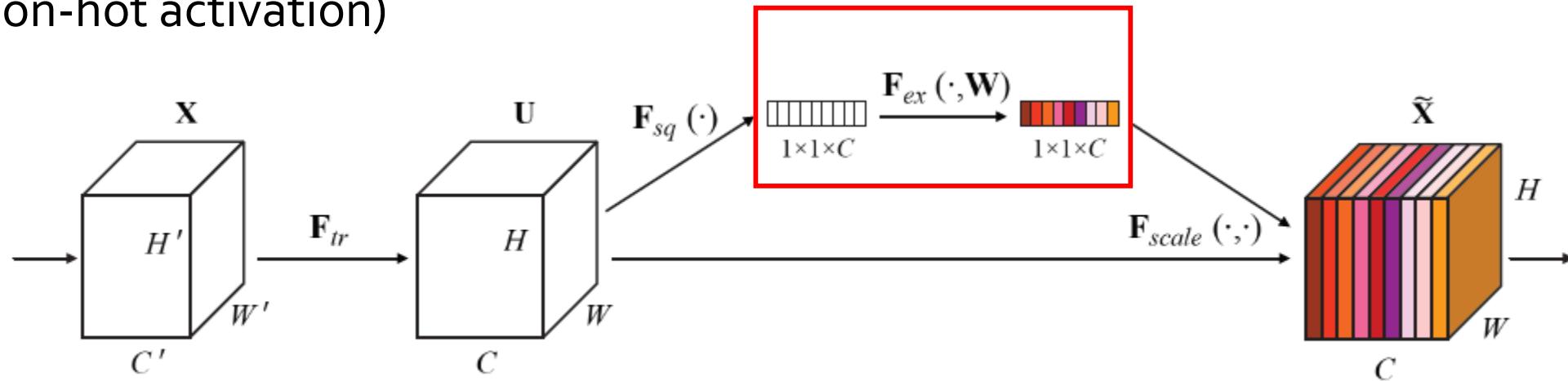
Squeeze : Global Information Embedding

- 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)

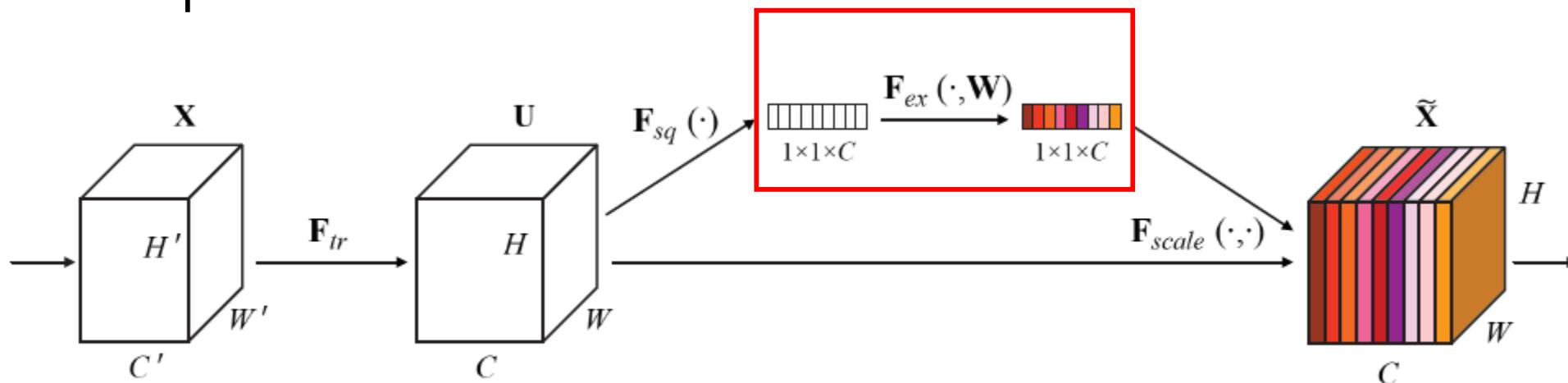


Excitation : Adaptive Recalibration

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

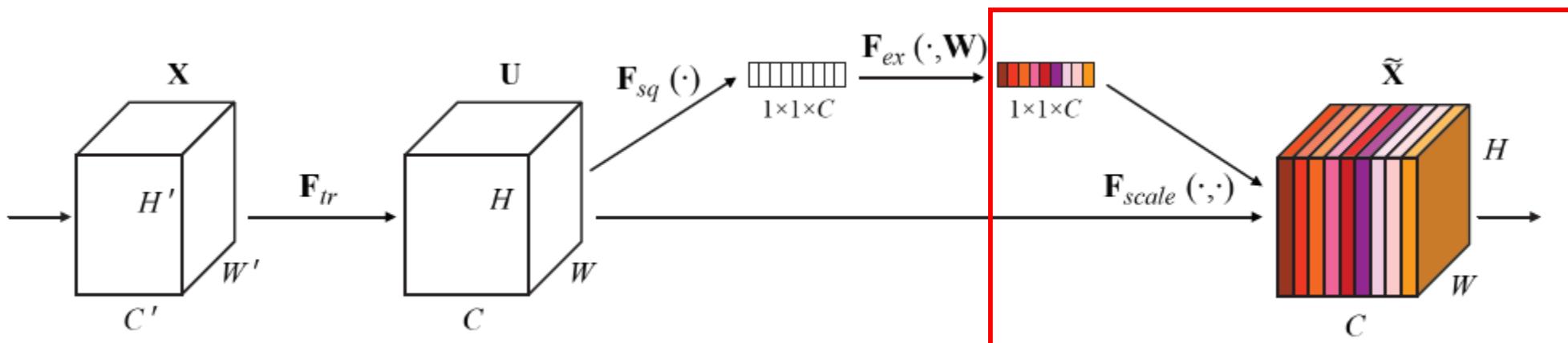
$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})),$$

- 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

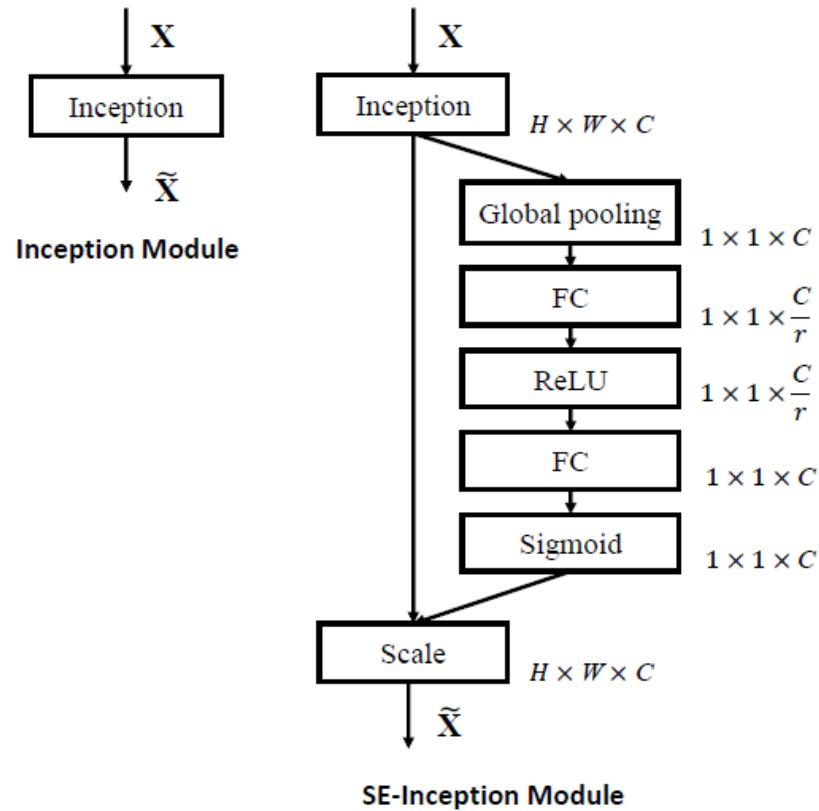


Fig. 2. The schema of the original Inception module (left) and the SE-Inception module (right).

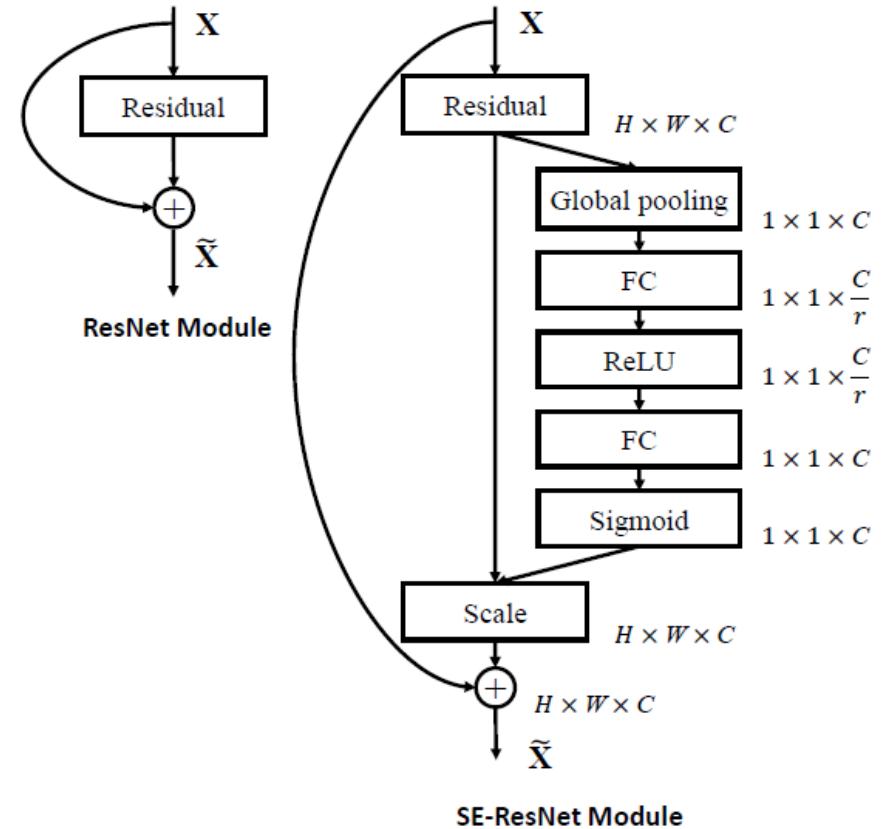


Fig. 3. The schema of the original ResNet module (left) and the SE-ResNet module (right).

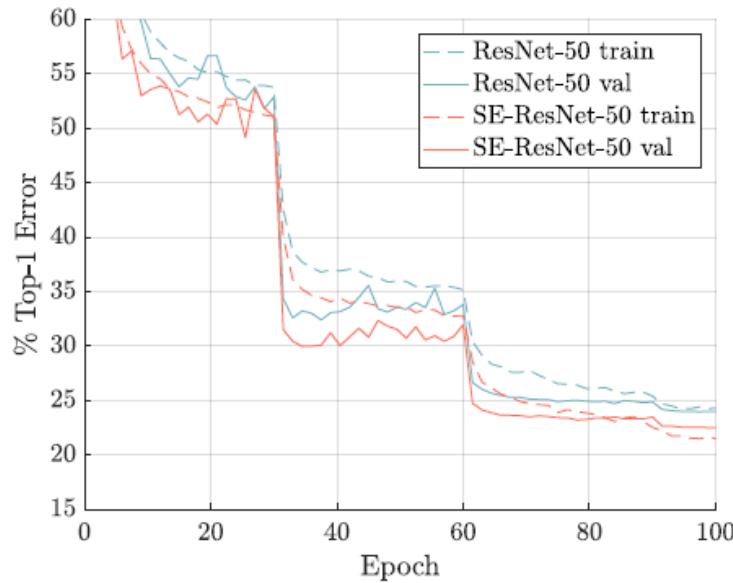
Example Architecture

TABLE 1

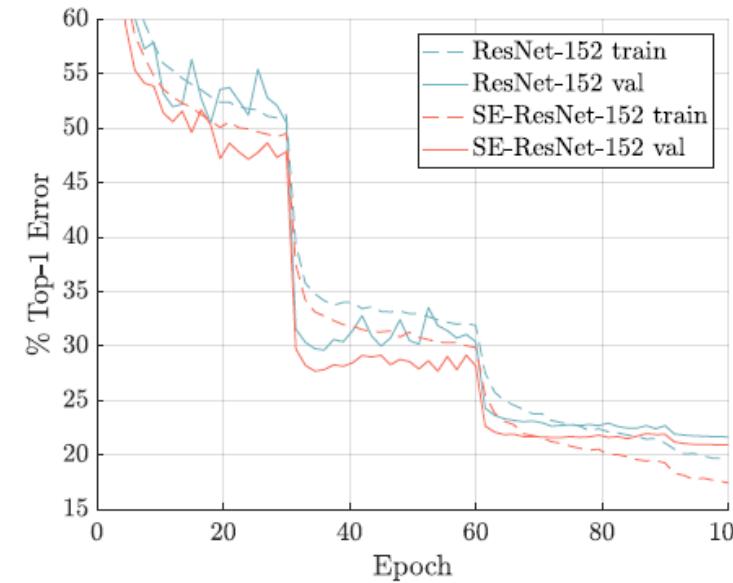
(Left) ResNet-50. (Middle) SE-ResNet-50. (Right) SE-ResNeXt-50 with a $32 \times 4d$ template. The shapes and operations with specific parameter settings of a residual building block are listed inside the brackets and the number of stacked blocks in a stage is presented outside. The inner brackets following 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		max pool, 3×3 , stride 2	
	$\begin{bmatrix} \text{conv, } 1 \times 1, 64 \\ \text{conv, } 3 \times 3, 64 \\ \text{conv, } 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 64 \\ \text{conv, } 3 \times 3, 64 \\ \text{conv, } 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv, } 1 \times 1, 128 \\ \text{conv, } 3 \times 3, 128 \\ \text{conv, } 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
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		

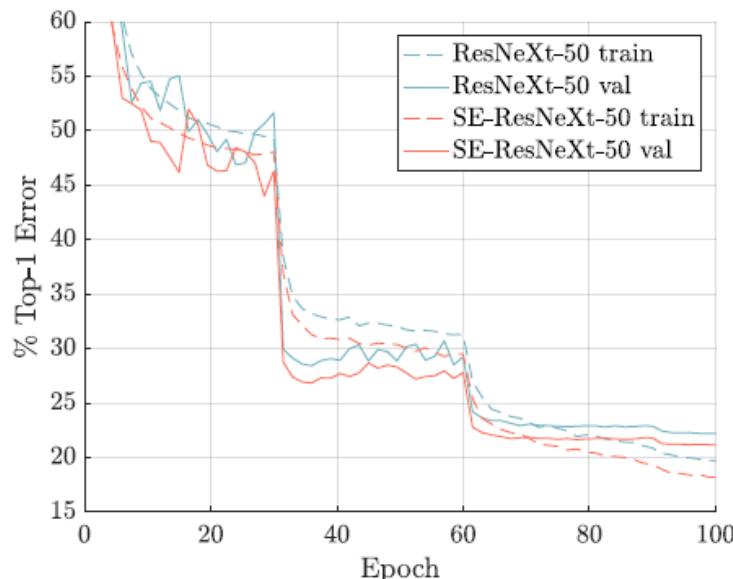
Results



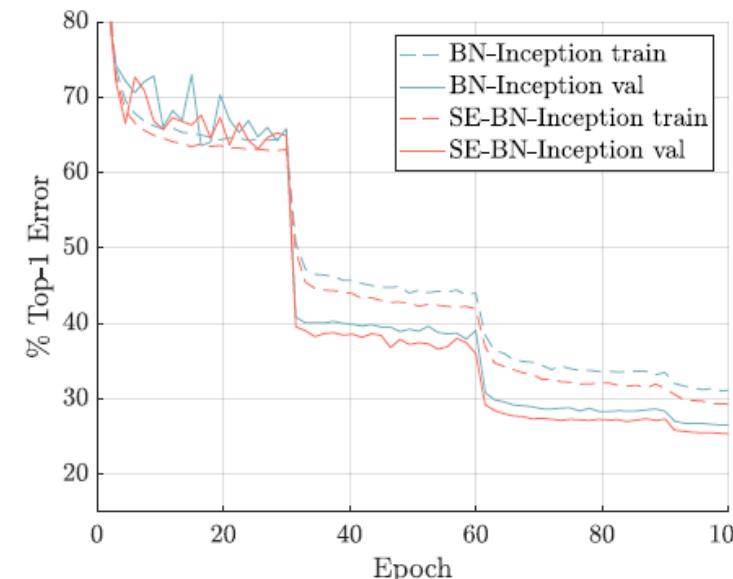
(a) ResNet-50 and SE-ResNet-50



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



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



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

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].

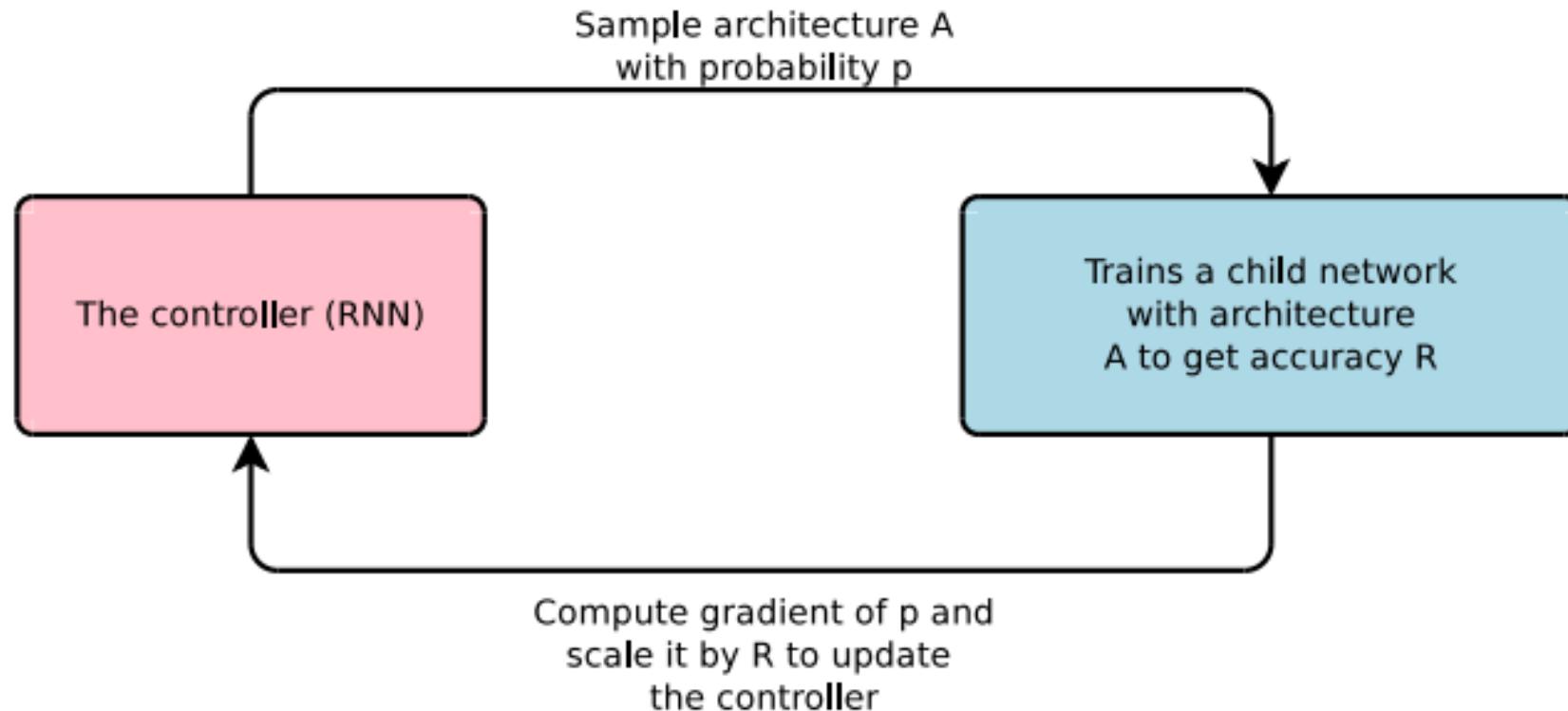
Tesorflow Pre-trained Models

- <https://github.com/tensorflow/models/tree/master/research/slim>

Model	TF-Slim File	Checkpoint	Top-1 Accuracy	Top-5 Accuracy
Inception V1	Code	inception_v1_2016_08_28.tar.gz	69.8	89.6
Inception V2	Code	inception_v2_2016_08_28.tar.gz	73.9	91.8
Inception V3	Code	inception_v3_2016_08_28.tar.gz	78.0	93.9
Inception V4	Code	inception_v4_2016_09_09.tar.gz	80.2	95.2
Inception-ResNet-v2	Code	inception_resnet_v2_2016_08_30.tar.gz	80.4	95.3
ResNet V1 50	Code	resnet_v1_50_2016_08_28.tar.gz	75.2	92.2
ResNet V1 101	Code	resnet_v1_101_2016_08_28.tar.gz	76.4	92.9
ResNet V1 152	Code	resnet_v1_152_2016_08_28.tar.gz	76.8	93.2
ResNet V2 50^	Code	resnet_v2_50_2017_04_14.tar.gz	75.6	92.8
ResNet V2 101^	Code	resnet_v2_101_2017_04_14.tar.gz	77.0	93.7
ResNet V2 152^	Code	resnet_v2_152_2017_04_14.tar.gz	77.8	94.1
ResNet V2 200	Code	TBA	79.9*	95.2*
VGG 16	Code	vgg_16_2016_08_28.tar.gz	71.5	89.8
VGG 19	Code	vgg_19_2016_08_28.tar.gz	71.1	89.8
MobileNet_v1_1.0_224	Code	mobilenet_v1_1.0_224_2017_06_14.tar.gz	70.7	89.5
MobileNet_v1_0.50_160	Code	mobilenet_v1_0.50_160_2017_06_14.tar.gz	59.9	82.5
MobileNet_v1_0.25_128	Code	mobilenet_v1_0.25_128_2017_06_14.tar.gz	41.3	66.2
NASNet-A_Mobile_224#	Code	nasnet-a_mobile_04_10_2017.tar.gz	74.0	91.6
NASNet-A_Large_331#	Code	nasnet-a_large_04_10_2017.tar.gz	82.7	96.2

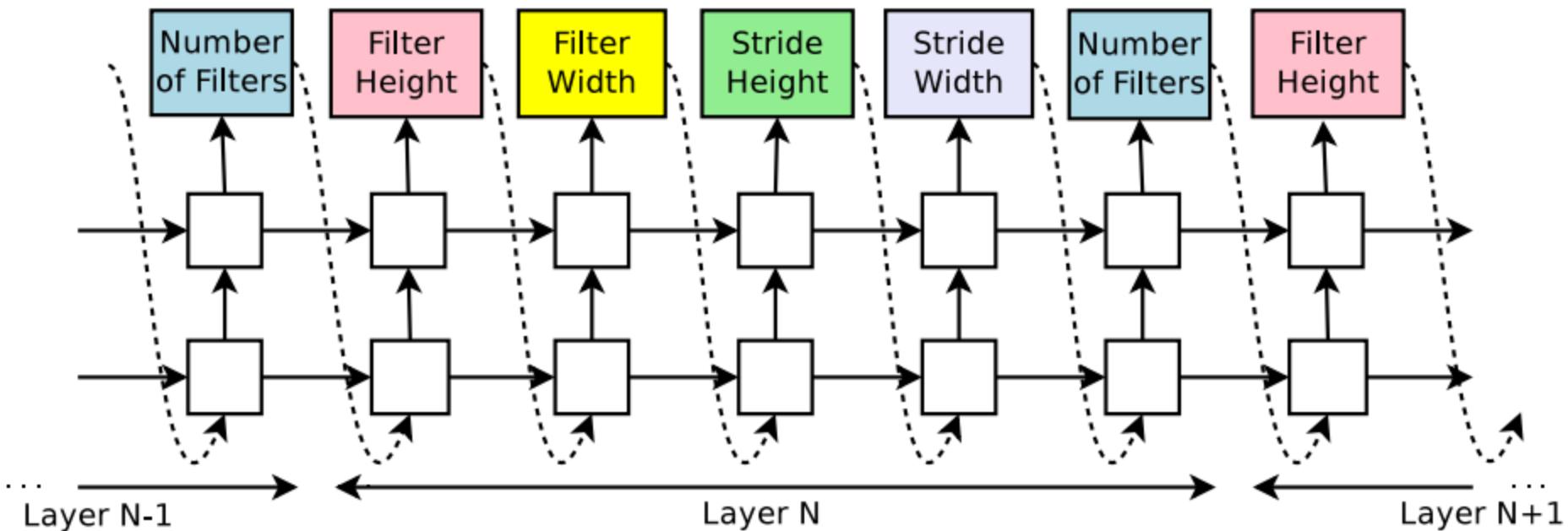
Neural Architecture Search

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



Neural Architecture Search

- How controller RNN samples a simple convolutional network



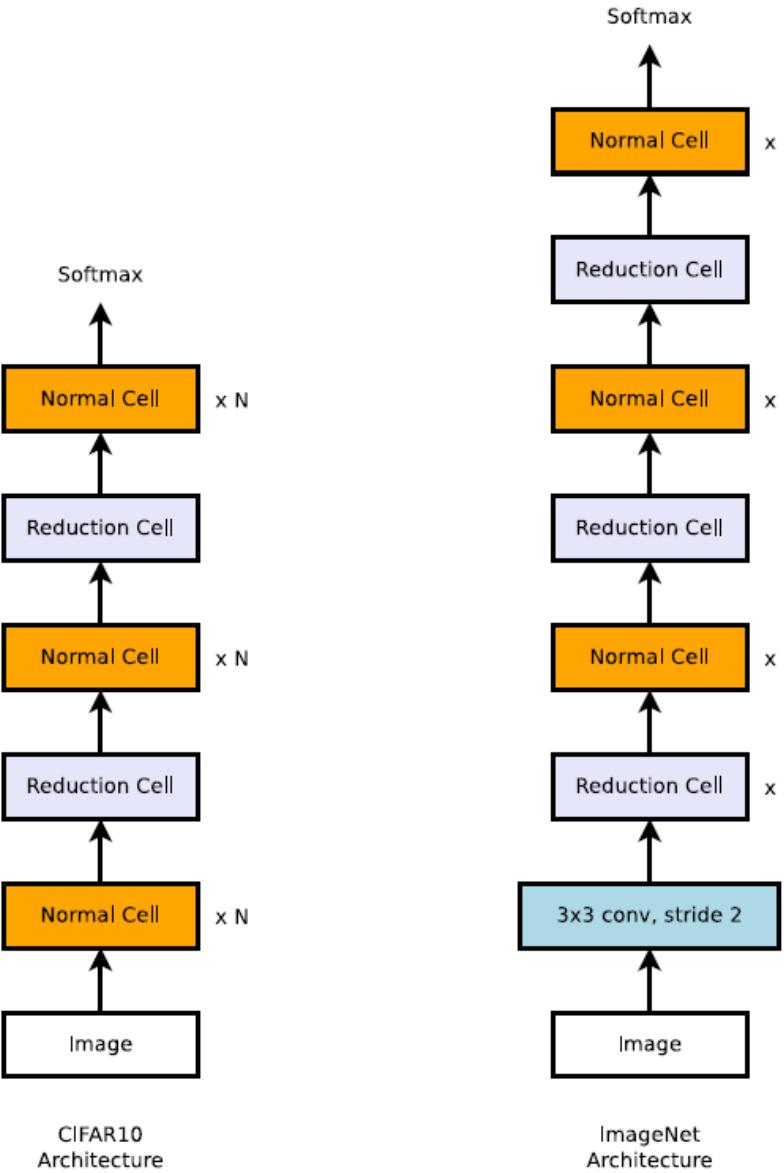
Motivation & Idea

- NAS used 800 GPUs for 28days resulting in **22,400 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

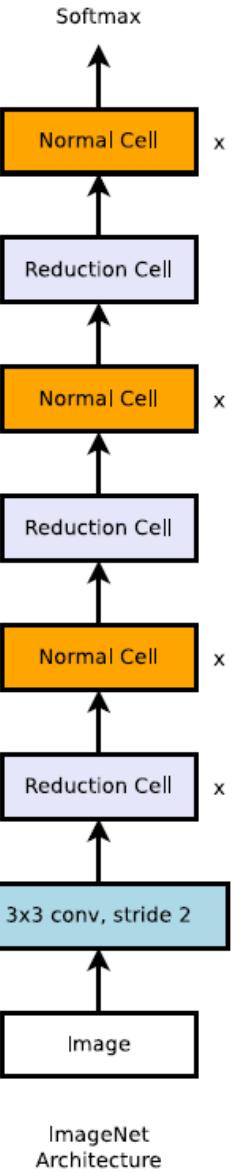
Method

- 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

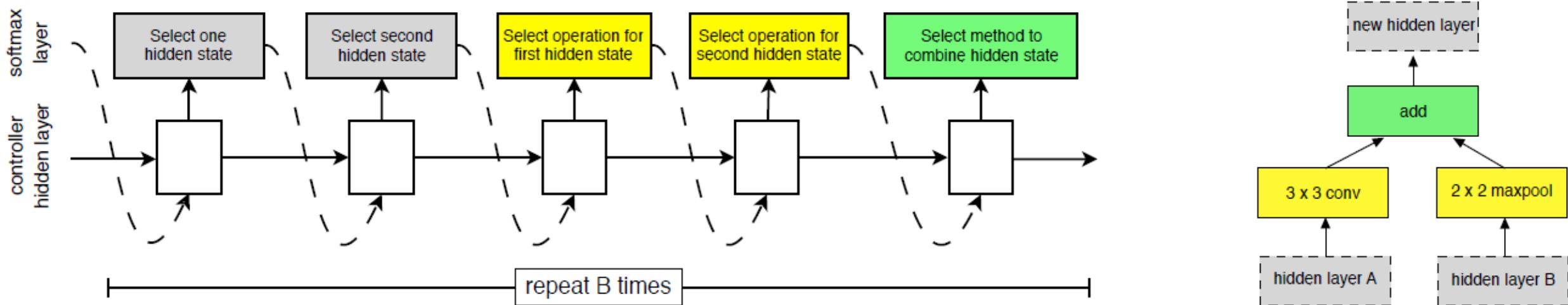


CIFAR10
Architecture



ImageNet
Architecture

Models and Algorithms



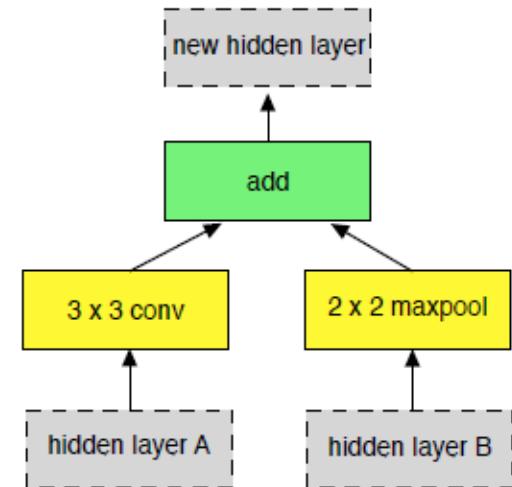
Step 1. Select a hidden state from h_i, h_{i-1} or from the set of hidden states created in previous blocks.

Step 2. Select a second hidden state from the same options as in Step 1.

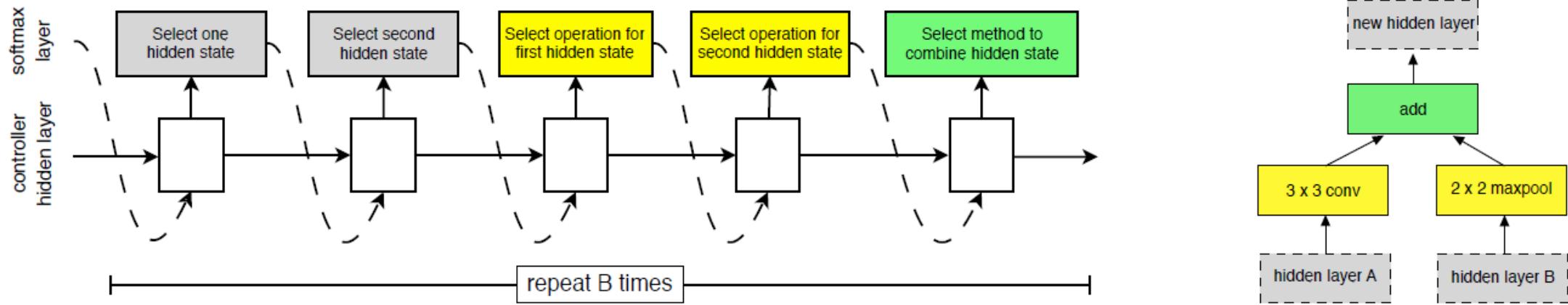
Step 3. Select an operation to apply to the hidden state selected in Step 1.

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.

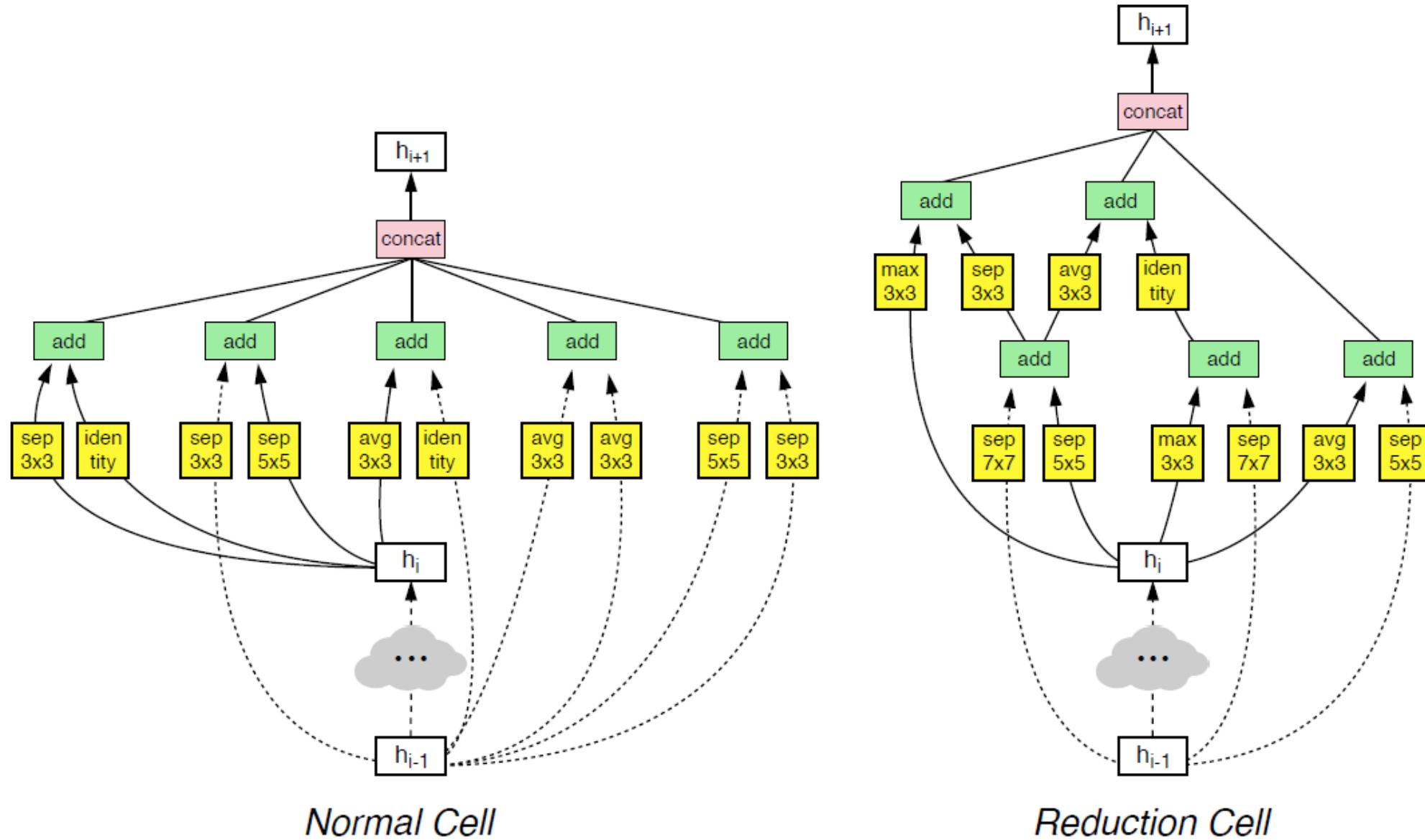


Search Space in a Cell (Step 3 and 4)

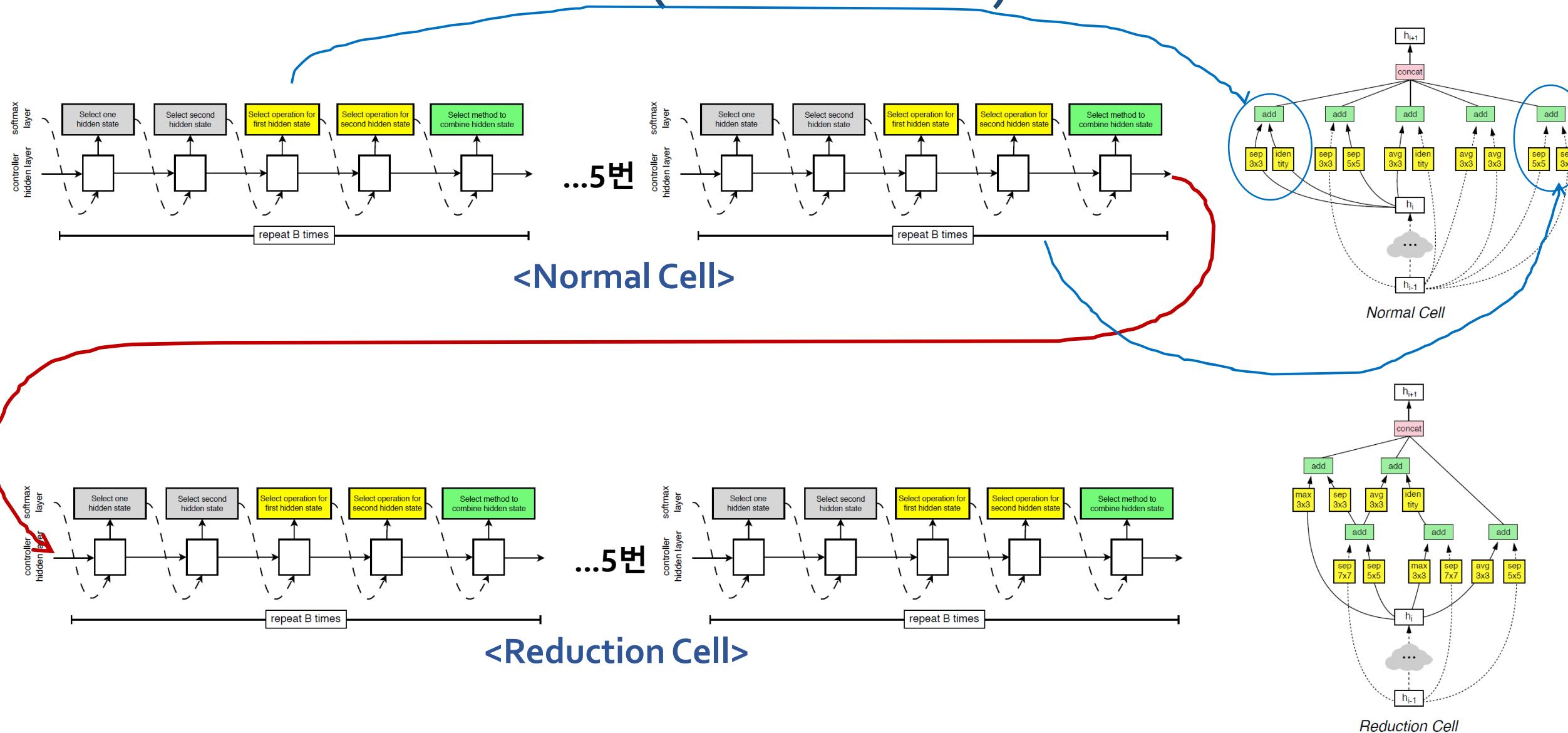


- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-separable conv

Best Architecture (NASNet-A)



Best Architecture (NASNet-A)



Results on CIFAR-10

model	depth	# params	error rate (%)
DenseNet ($L = 40, k = 12$) [26]	40	1.0M	5.24
DenseNet ($L = 100, k = 12$) [26]	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) [26]	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) [26]	190	25.6M	3.46
Shake-Shake 26 2x32d [18]	26	2.9M	3.55
Shake-Shake 26 2x96d [18]	26	26.2M	2.86
Shake-Shake 26 2x96d + cutout [12]	26	26.2M	2.56
NAS v3 [70]	39	7.1M	4.47
NAS v3 [70]	39	37.4M	3.65
NASNet-A (6 @ 768)	-	3.3M	3.41
NASNet-A (6 @ 768) + cutout	-	3.3M	2.65
NASNet-A (7 @ 2304)	-	27.6M	2.97
NASNet-A (7 @ 2304) + cutout	-	27.6M	2.40
NASNet-B (4 @ 1152)	-	2.6M	3.73
NASNet-C (4 @ 640)	-	3.1M	3.59

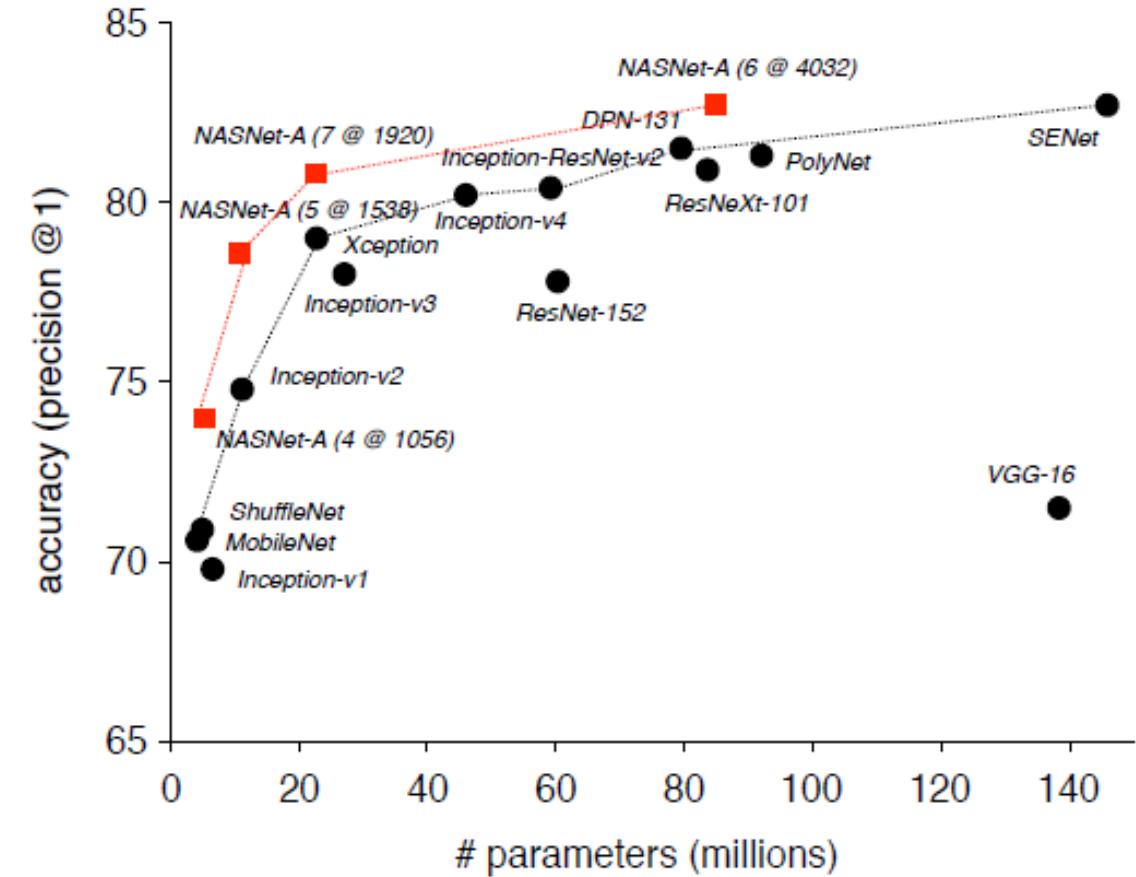
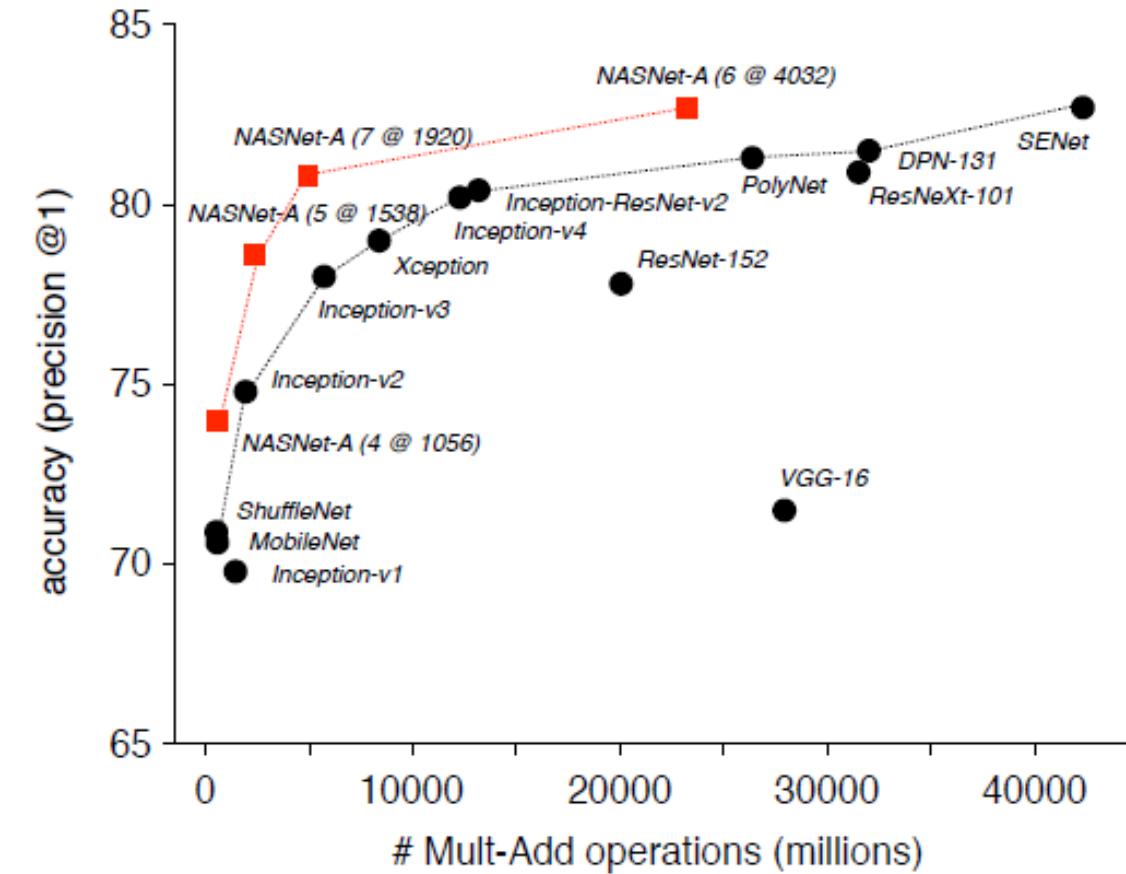
Results on ImageNet (vs SOTA)

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	299×299	10.9 M	2.35 B	78.6	94.2
Inception V3 [59]	299×299	23.8 M	5.72 B	78.0	93.9
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [57]	299×299	55.8 M	13.2 B	80.4	95.3
NASNet-A (7 @ 1920)	299×299	22.6 M	4.93 B	80.8	95.3
ResNeXt-101 (64 x 4d) [67]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [68]	331×331	92 M	34.7 B	81.3	95.8
DPN-131 [8]	320×320	79.5 M	32.0 B	81.5	95.8
SENet [25]	320×320	145.8 M	42.3 B	82.7	96.2
NASNet-A (6 @ 4032)	331×331	88.9 M	23.8 B	82.7	96.2

Results on ImageNet in a Resource-Constrained Setting

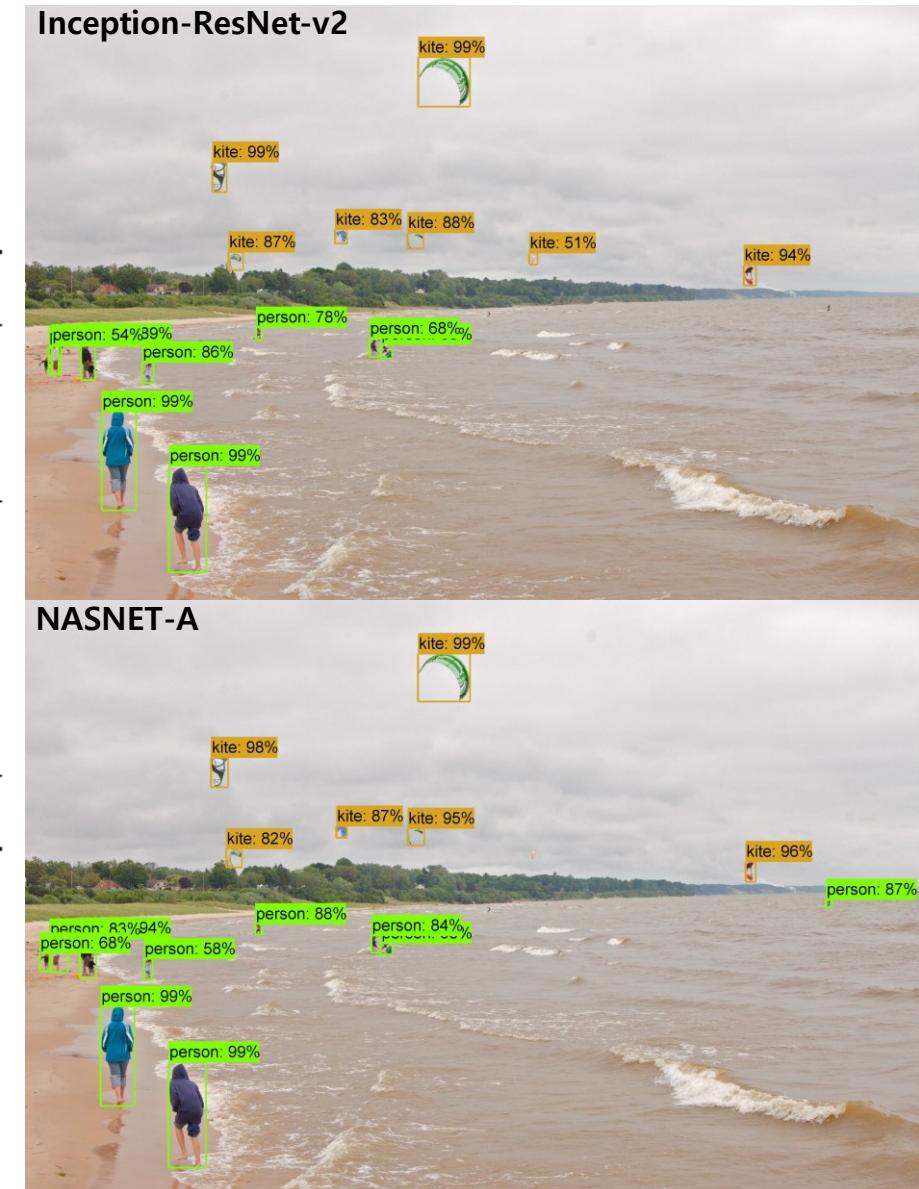
Model	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V1 [58]	6.6M	1,448 M	69.8	89.9
MobileNet-224 [24]	4.2 M	569 M	70.6	89.5
ShuffleNet (2x) [69]	~ 5M	524 M	70.9	89.8
NASNet-A (4 @ 1056)	5.3 M	564 M	74.0	91.6
NASNet-B (4 @ 1536)	5.3M	488 M	72.8	91.3
NASNet-C (3 @ 960)	4.9M	558 M	72.5	91.0

Results on ImageNet

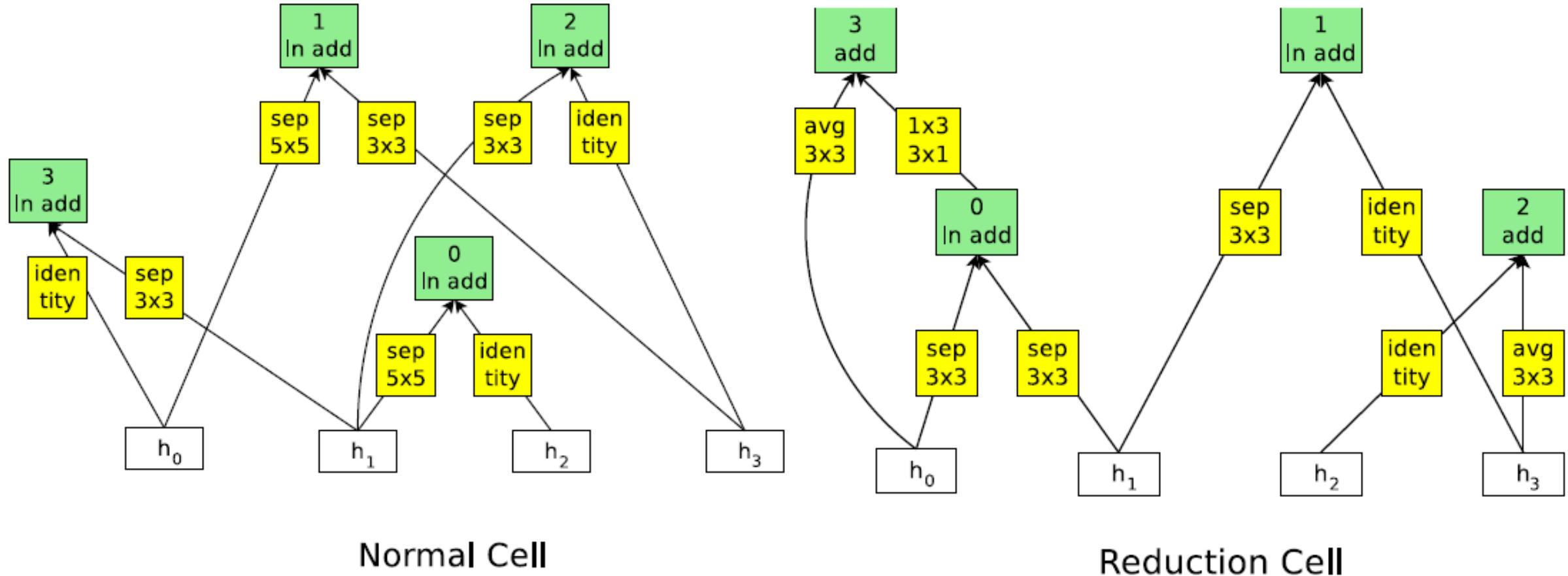


Object Detection Performance on COCO

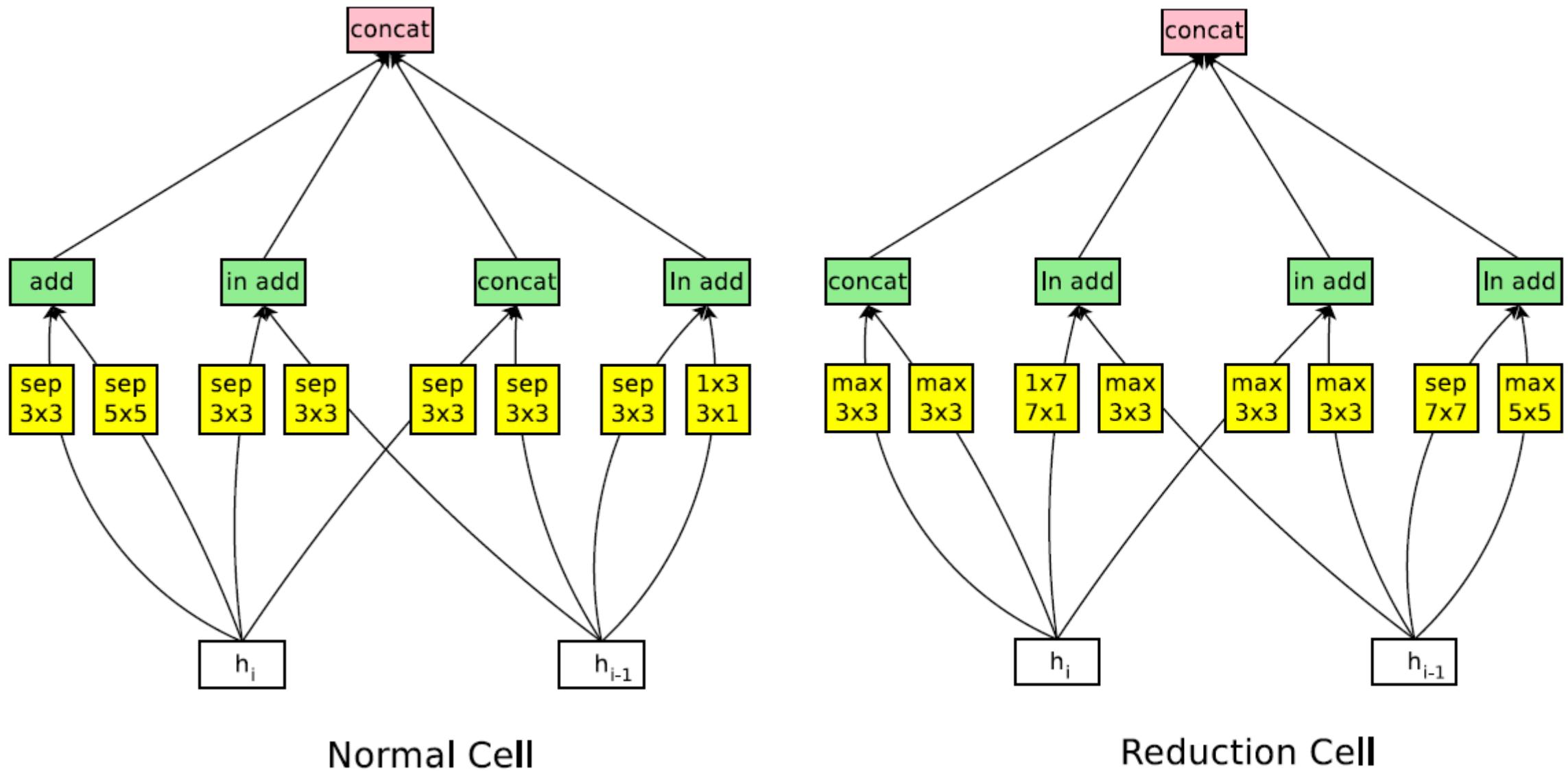
Model	resolution	mAP (mini-val)	mAP (test-dev)
MobileNet-224 [24]	600×600	19.8%	-
ShuffleNet (2x) [69]	600×600	24.5% [†]	-
NASNet-A (4 @ 1056)	600×600	29.6%	-
ResNet-101-FPN [35]	800 (short side)	-	36.2%
Inception-ResNet-v2 (G-RMI) [28]	600×600	35.7%	35.6%
Inception-ResNet-v2 (TDM) [51]	600×1000	37.3%	36.8%
NASNet-A (6 @ 4032)	800×800	41.3%	40.7%
NASNet-A (6 @ 4032)	1200×1200	43.2%	43.1%
ResNet-101-FPN (RetinaNet) [36]	800 (short side)	-	39.1%



NASNet-B

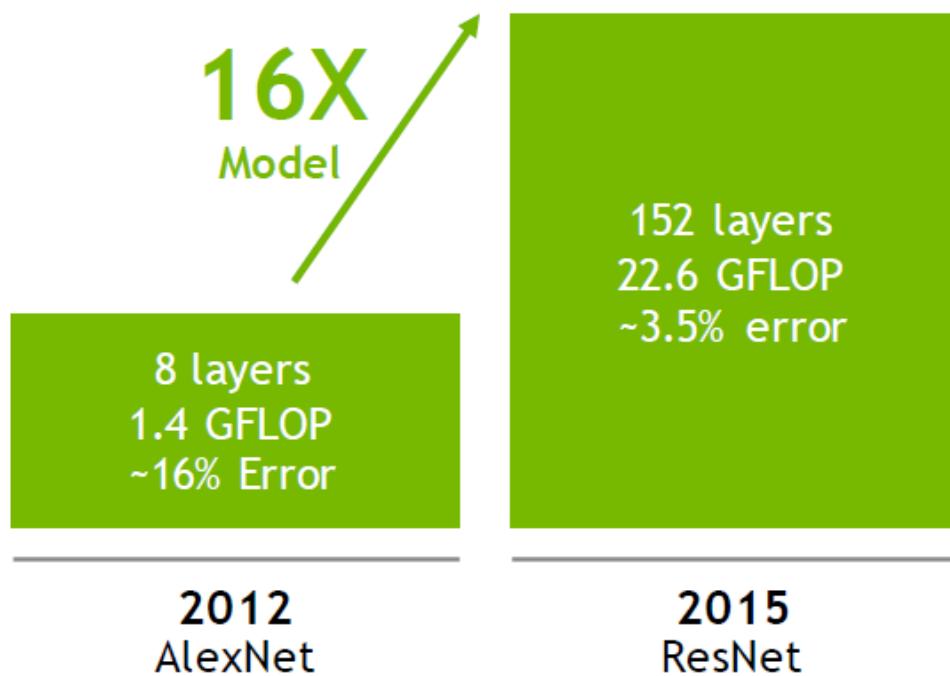


NASNet-C



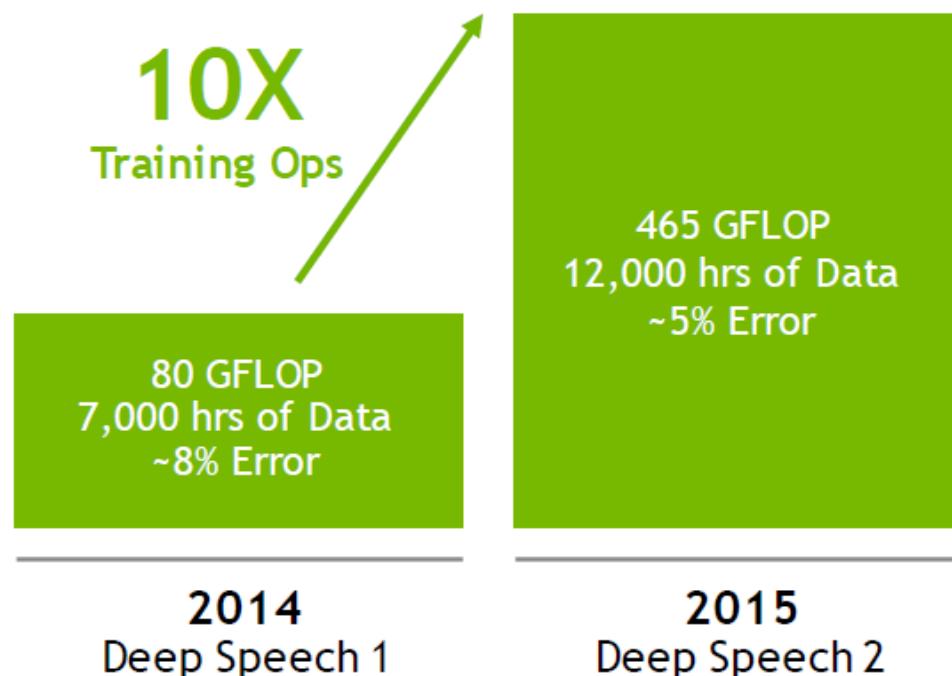
Deep Learning Models are Getting Larger

IMAGE RECOGNITION



Microsoft

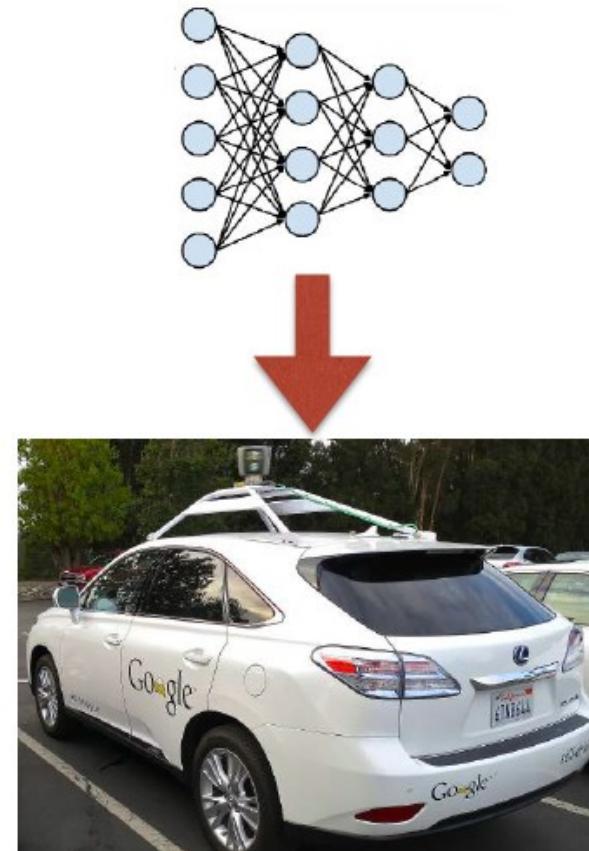
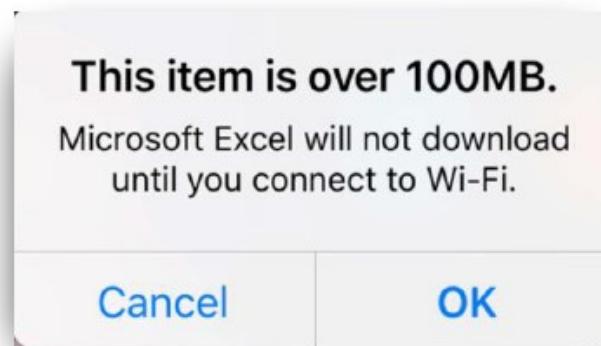
SPEECH RECOGNITION



Baidu

The First Challenge: Model Size

- Hard to distribute large models through over-the-air update



The Second Challenge: Speed

	Error rate	Training time
ResNet18:	10.76%	2.5 days
ResNet50:	7.02%	5 days
ResNet101:	6.21%	1 week
ResNet152:	6.16%	1.5 weeks

Such long training time limits ML researcher's productivity

Training time benchmarked with `fb.resnet.torch` using four M40 GPUs

The Third Challenge: Energy Efficiency



[This image is in the public domain](#)

AlphaGo: 1920 CPUs and 280 GPUs,
\$3000 electric bill per game



[This image is in the public domain](#)



[Phone image is licensed under CC-BY 2.0](#)

on mobile: **drains battery**
on data-center: **increases TCO**



[This image is licensed under CC-BY 2.0](#)

List of Papers

- “**SqueezeNet**: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size”
- “**Xception**: Deep Learning with Depthwise Separable Convolutions”
- “**MobileNets**: Efficient Convolutional Neural Networks for Mobile Vision Applications”
- “**ShuffleNet**: An Extremely Efficient Convolutional Neural Network for Mobile Devices”
- “**MobileNetV2**: Inverted Residuals and Linear Bottlenecks”
- “**SqueezeNext**: Hardware-Aware Neural Network Design”
- “**ShuffleNet V2**: Practical Guidelines for Efficient CNN Architecture Design”

All these papers are reviewed by PR12 members

SqueezeNet

SQUEEZE NET: ALEXNET-LEVEL ACCURACY WITH 50X FEWER PARAMETERS AND <0.5MB MODEL SIZE

Forrest N. Iandola¹, Song Han², Matthew W. Moskewicz¹, Khalid Ashraf¹,
William J. Dally², Kurt Keutzer¹

¹DeepScale* & UC Berkeley ²Stanford University

{forresti, moskewcz, kashraf, keutzer}@eecs.berkeley.edu
{songhan, dally}@stanford.edu

ABSTRACT

Recent research on deep convolutional neural networks (CNNs) has focused primarily on improving accuracy. For a given accuracy level, it is typically possible to identify multiple CNN architectures that achieve that accuracy level. With equivalent accuracy, smaller CNN architectures offer at least three advantages: (1) Smaller CNNs require less communication across servers during distributed training. (2) Smaller CNNs require less bandwidth to export a new model from the cloud to an autonomous car. (3) Smaller CNNs are more feasible to deploy on FPGAs and other hardware with limited memory. To provide all of these advantages, we propose a small CNN architecture called SqueezeNet. SqueezeNet achieves AlexNet-level accuracy on ImageNet with 50x fewer parameters. Additionally, with model compression techniques, we are able to compress SqueezeNet to less than 0.5MB (510 \times smaller than AlexNet).

The SqueezeNet architecture is available for download here:
<https://github.com/DeepScale/SqueezeNet>

SqueezeNet

- Architectural Design Strategies
 - Replace 3×3 filters with 1×1 filters
 - Decrease the number of input channels to 3×3 filters
 - Total quantity of parameters in 3×3 conv layer is (number of input channels) \times (number of filters) \times (3×3)
 - Downsample late in the network so that convolution layers have large activation maps
 - large activation maps (due to delayed downsampling) can lead to higher classification accuracy

The Fire Module

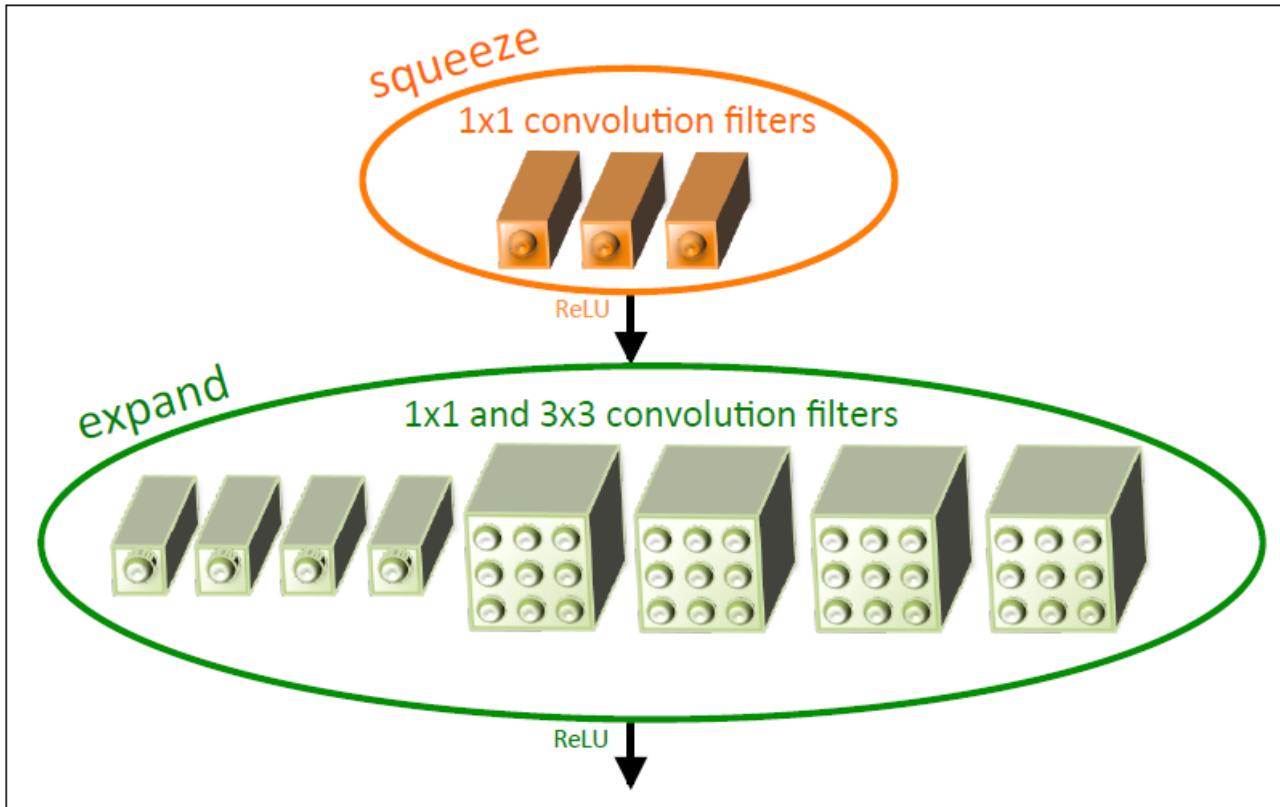
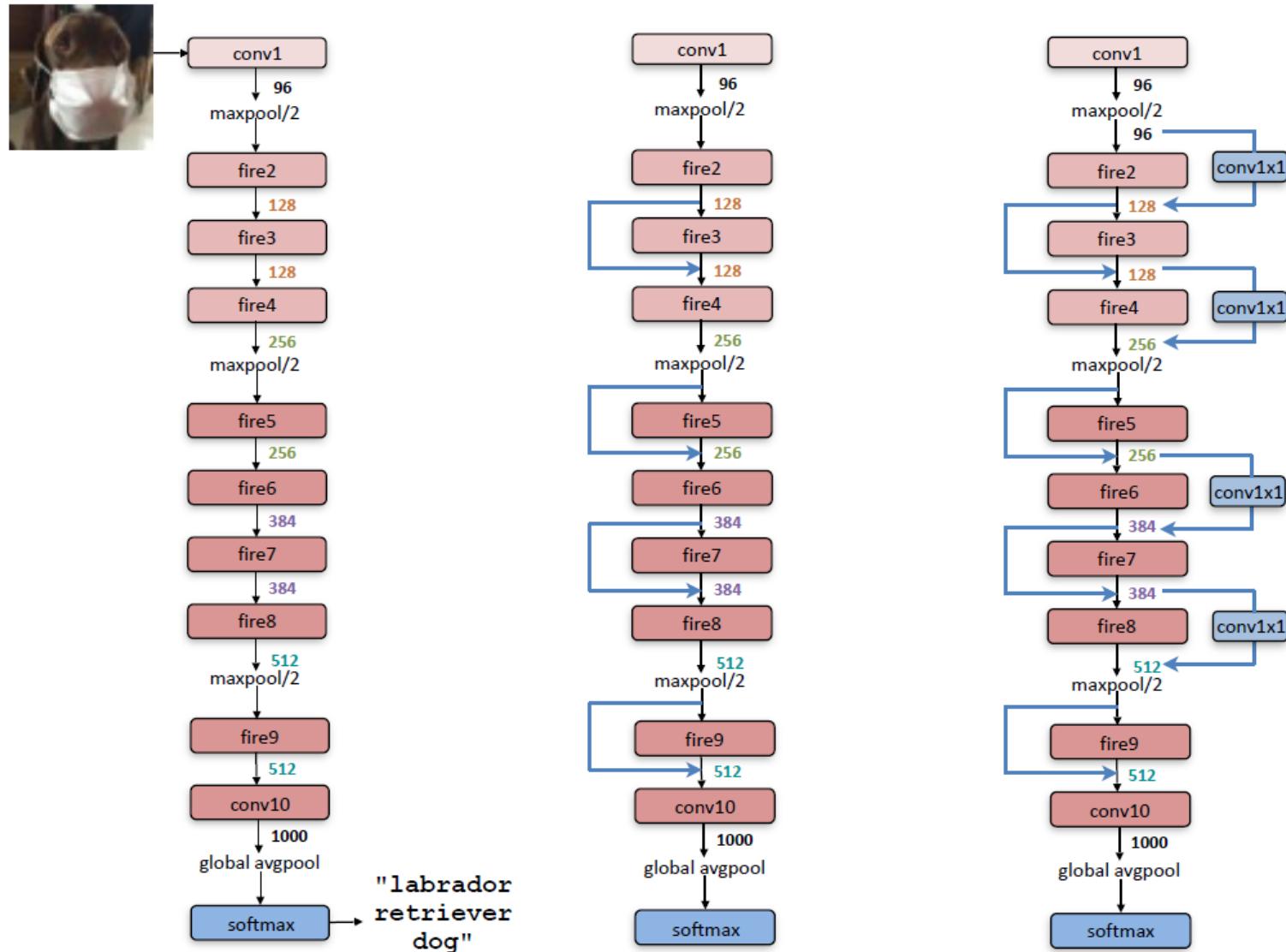


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

Macroarchitectural View



SqueezeNet Architecture

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

Results

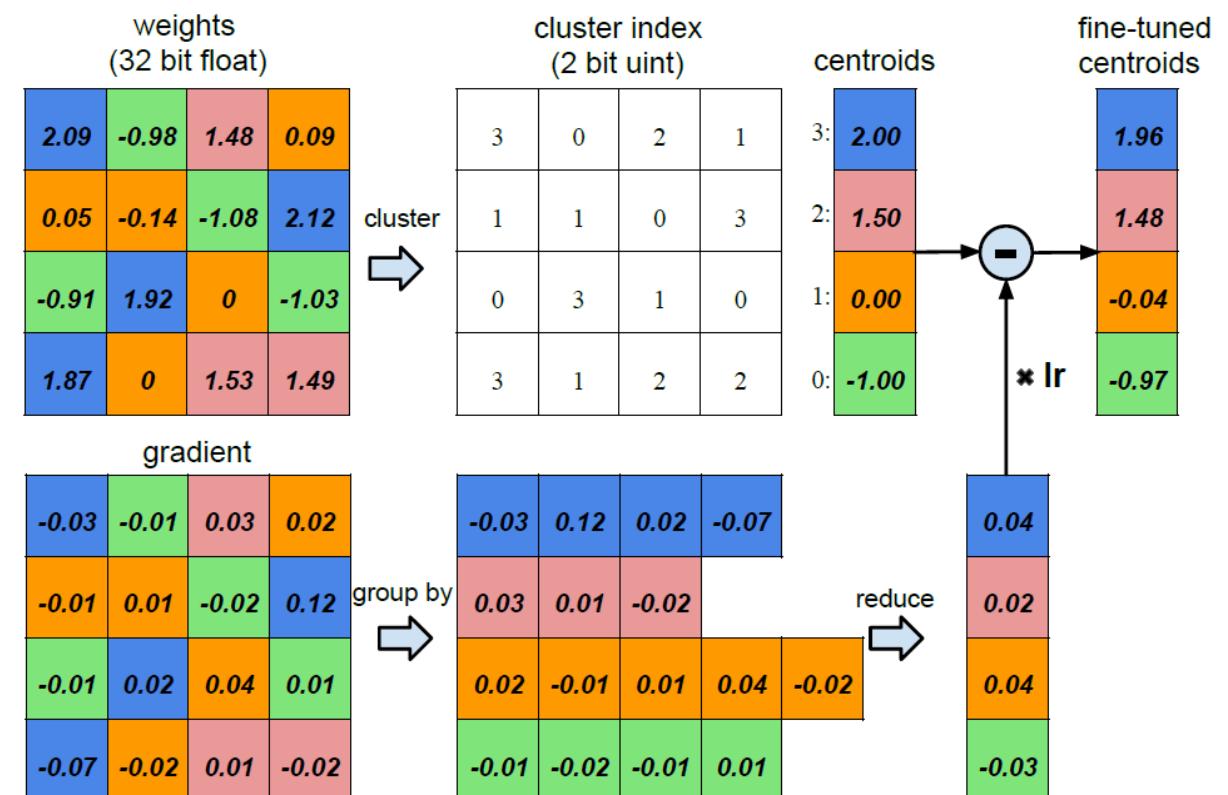
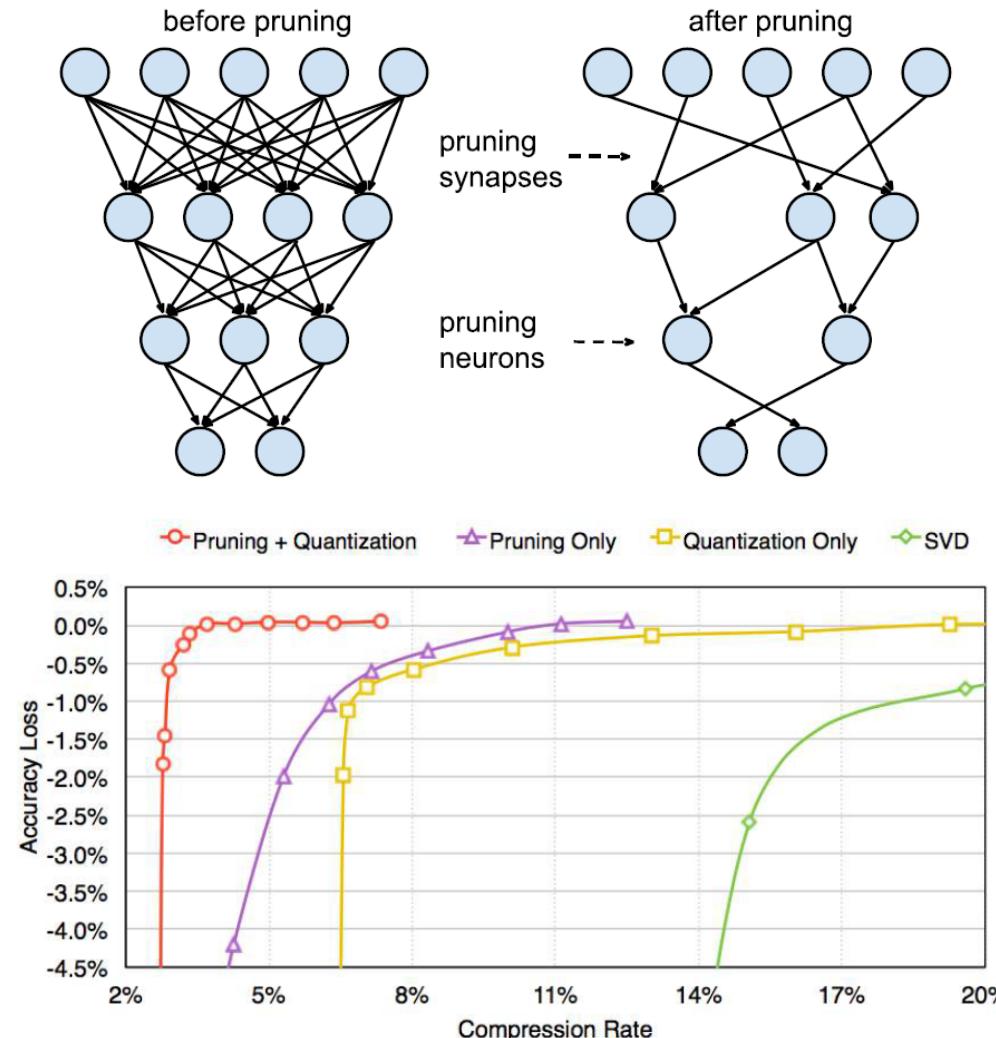
Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

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

Table 3: SqueezeNet accuracy and model size using different macroarchitecture configurations

Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size
Vanilla SqueezeNet	57.5%	80.3%	4.8MB
SqueezeNet + Simple Bypass	60.4%	82.5%	4.8MB
SqueezeNet + Complex Bypass	58.8%	82.0%	7.7MB

Network Pruning & Deep Compression



Xception

.02357v3 [cs.CV] 4 Apr 2017

Xception: Deep Learning with Depthwise Separable Convolutions

François Chollet

Google, Inc.

fchollet@google.com

Abstract

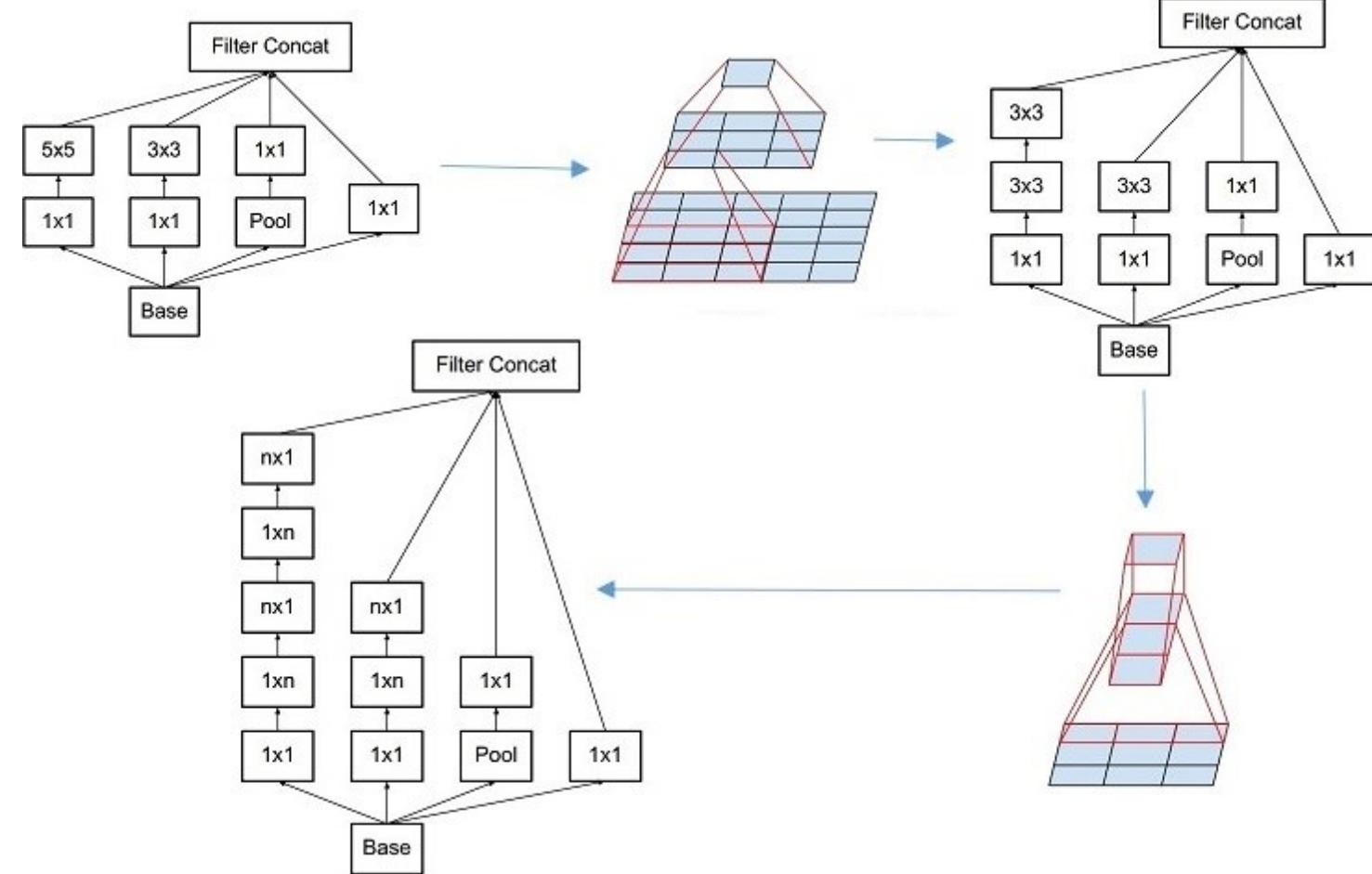
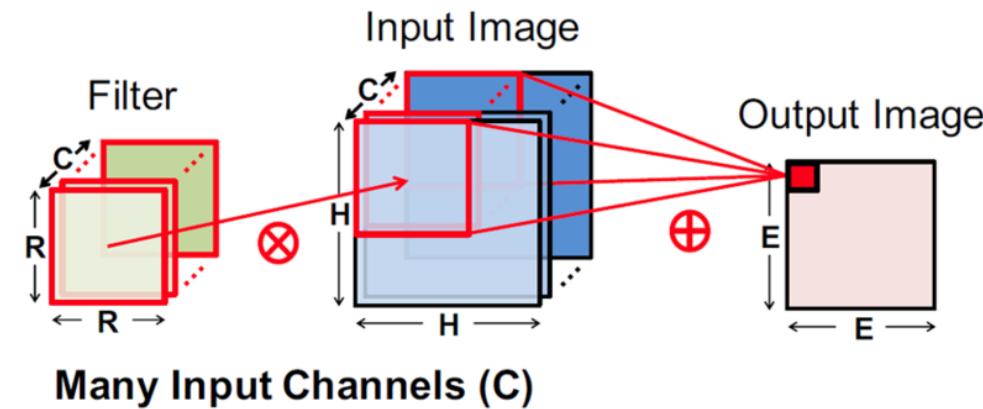
We present an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution 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?

Filter Factorization of CNN



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).

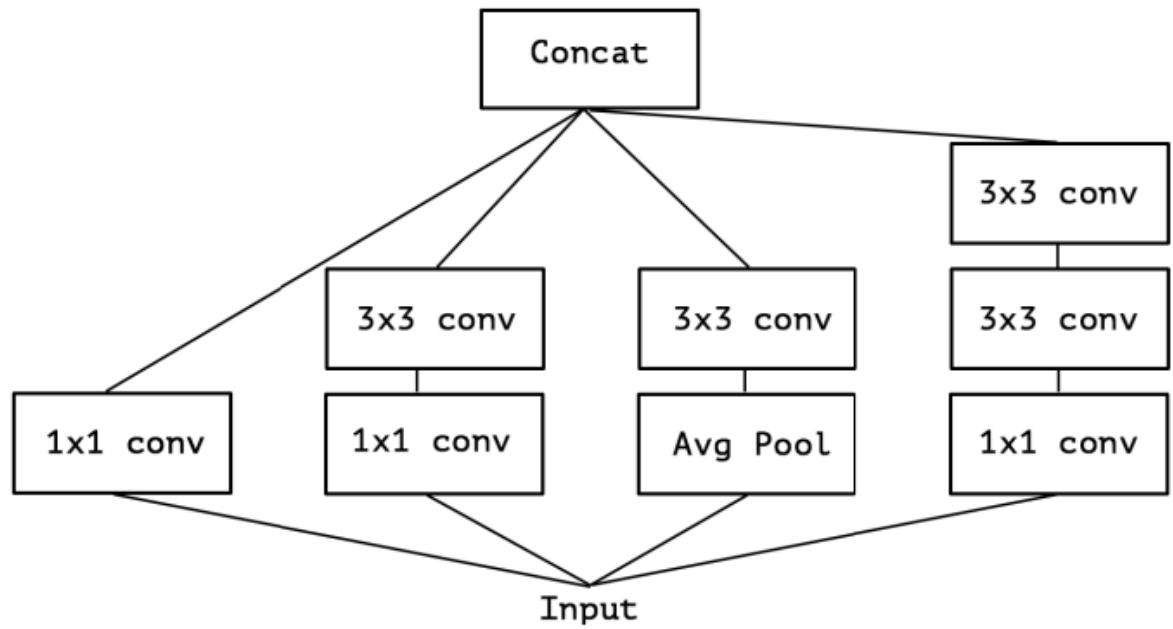
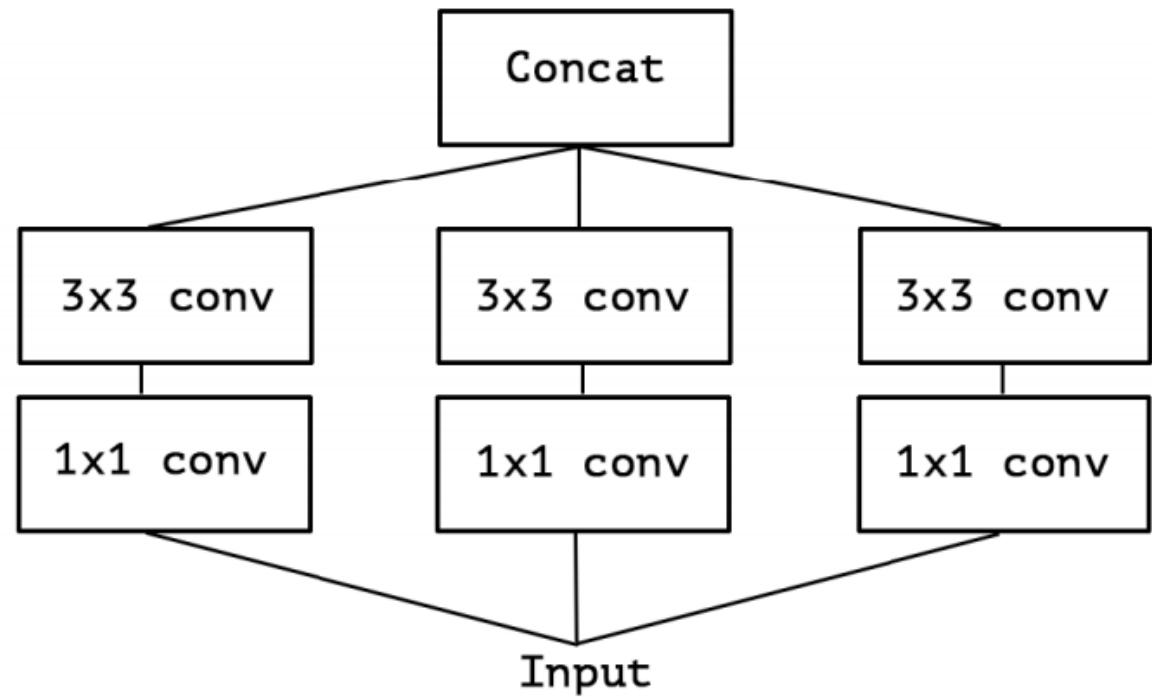


Figure 2. A simplified Inception module.



Equivalent Reformulation

Figure 2. A simplified Inception module.

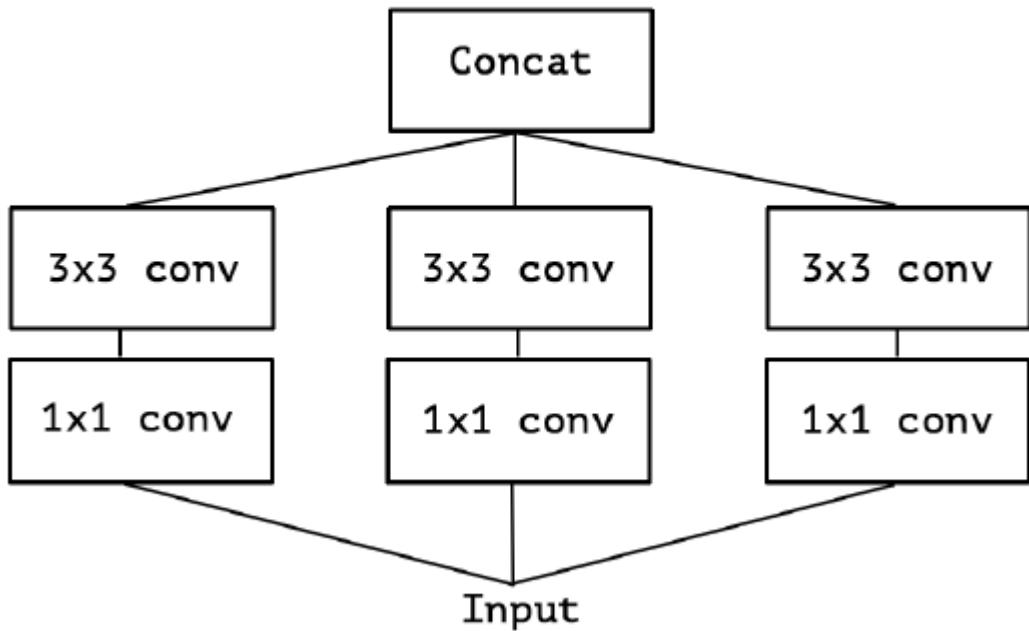
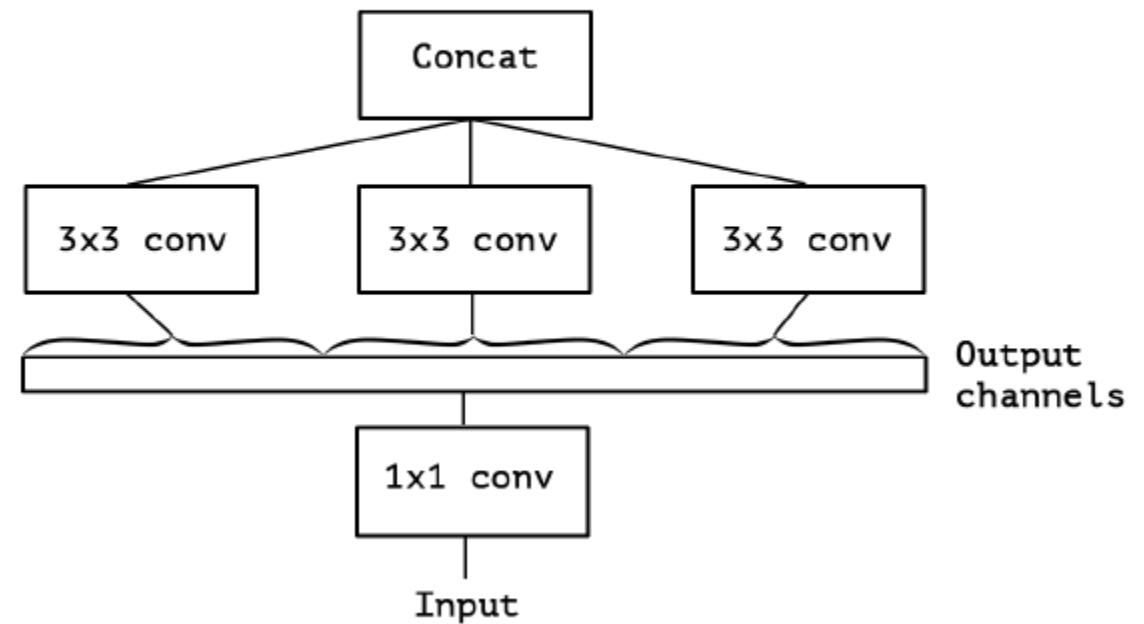
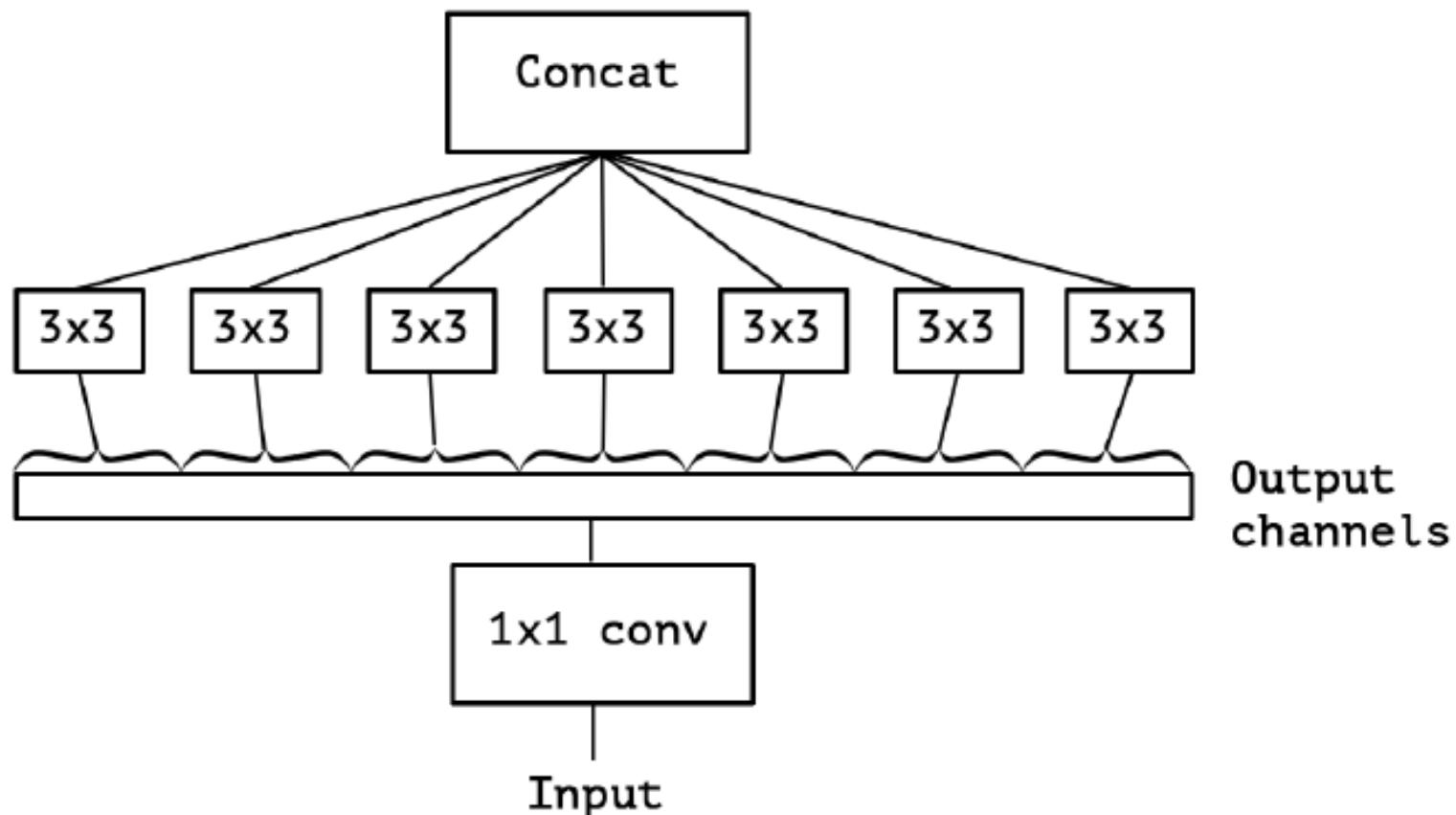


Figure 3. A strictly equivalent reformulation of the simplified Inception module.



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

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

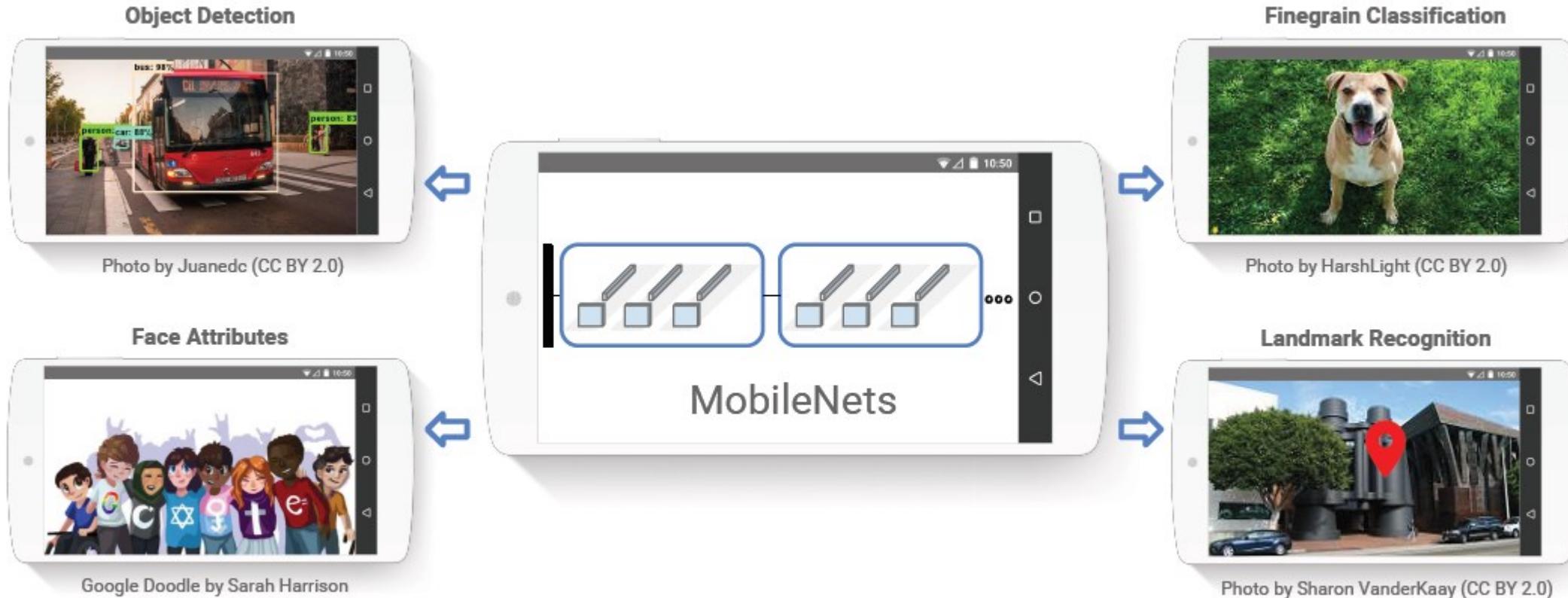
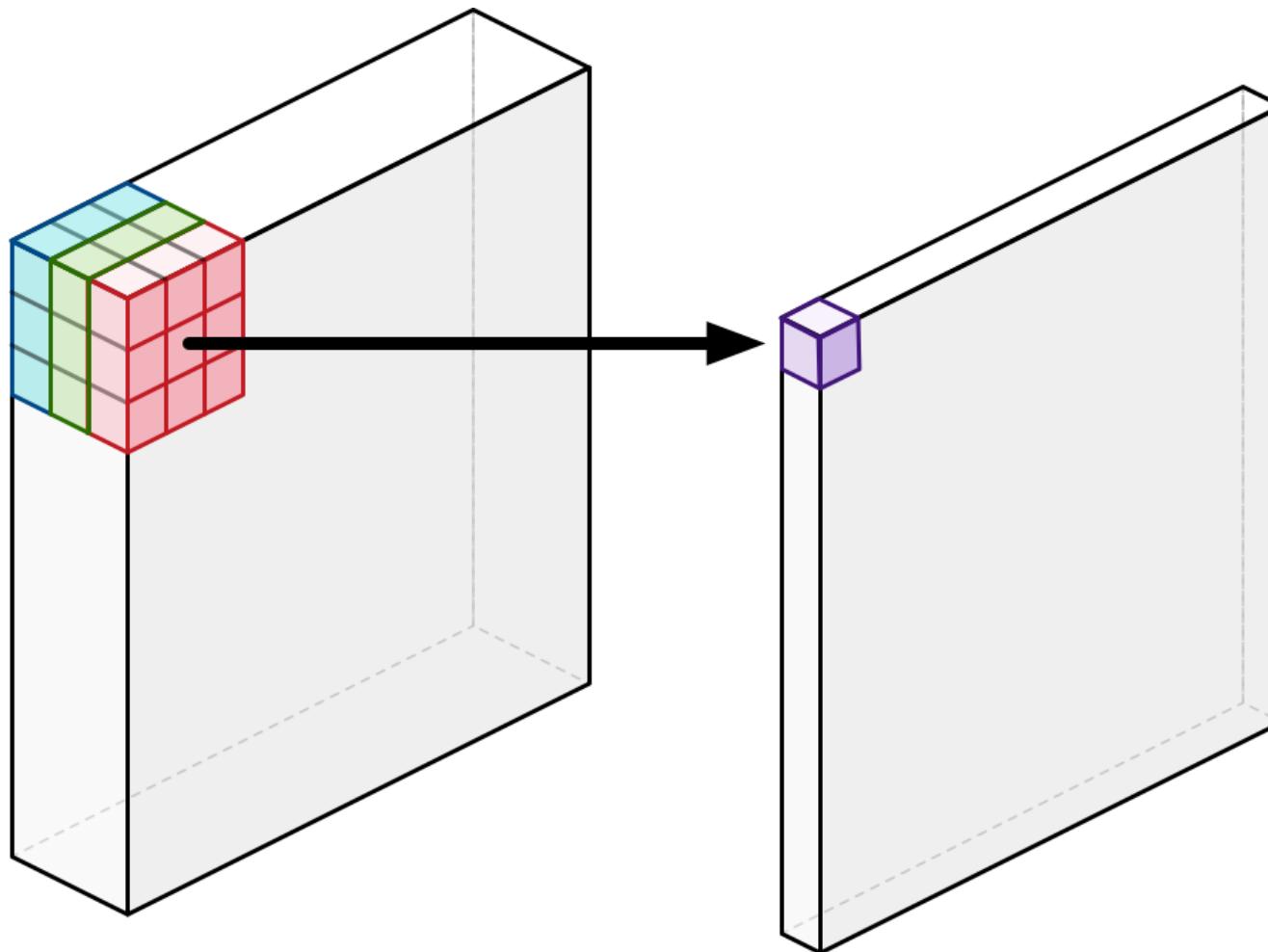


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

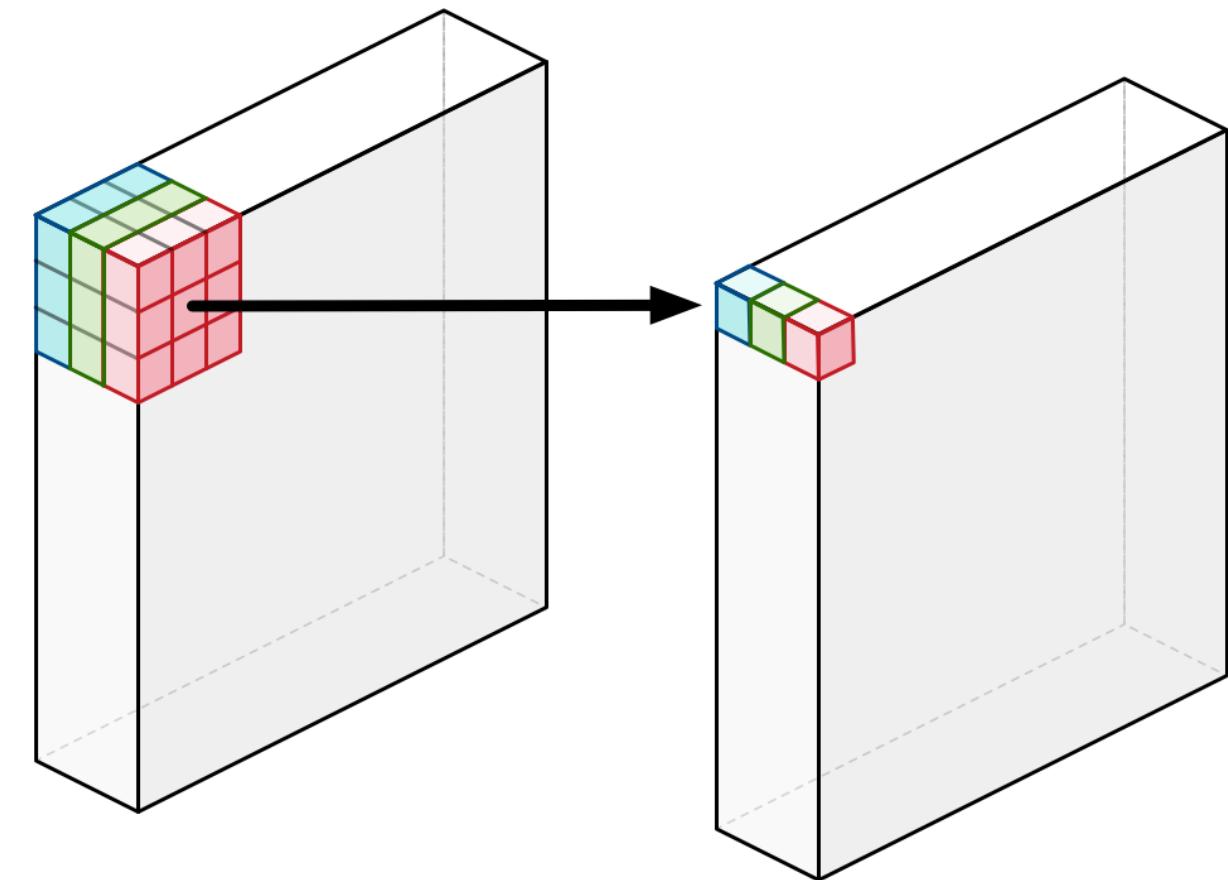
Standard Convolution



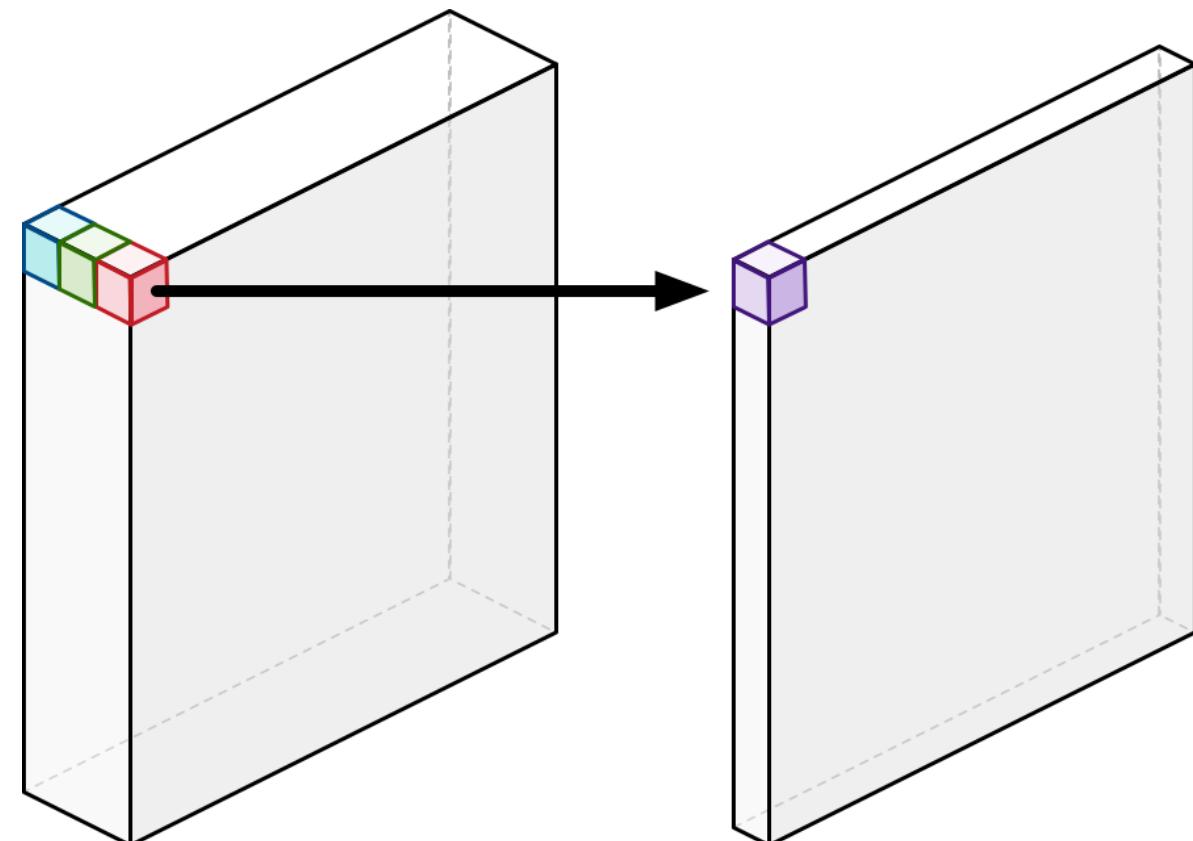
Standard convolution

Depthwise Separable Convolution

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

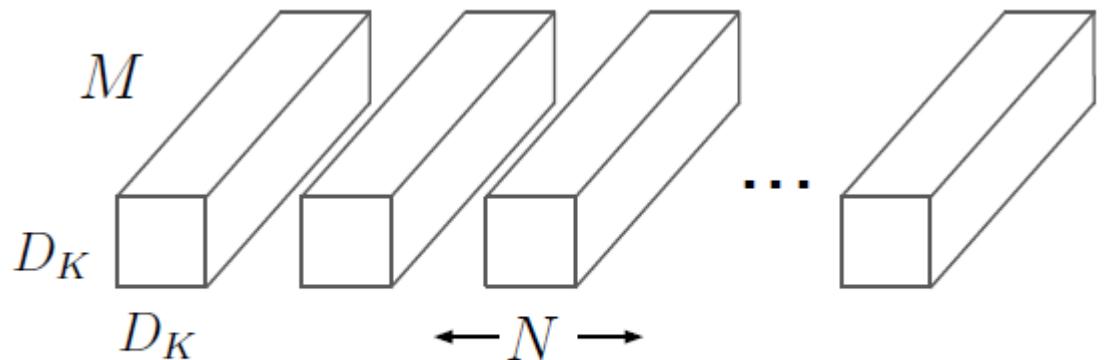


Depthwise convolution

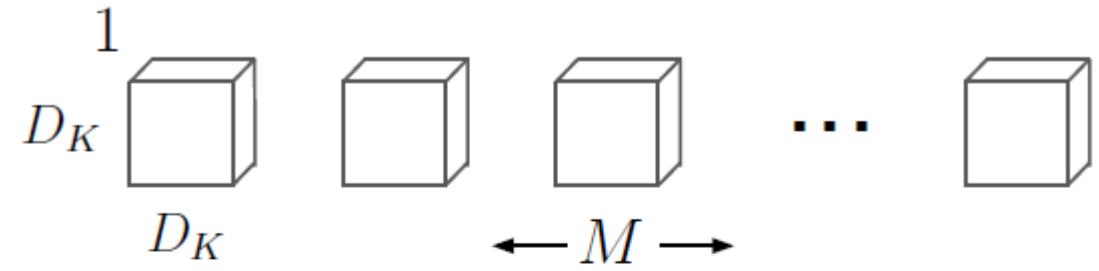


Pointwise convolution

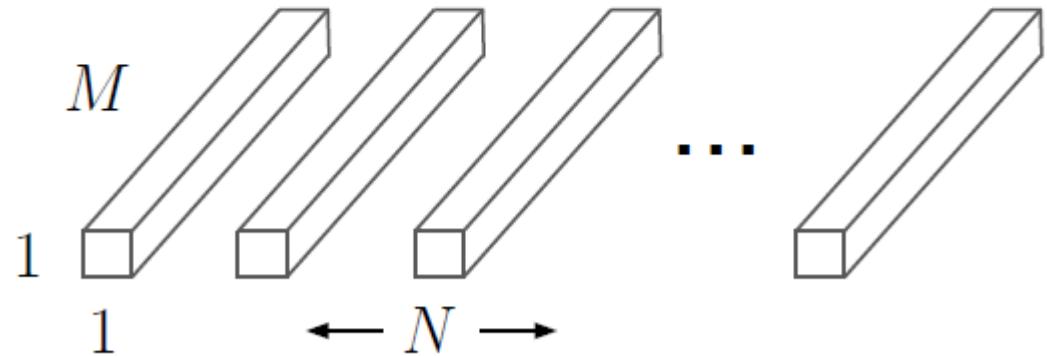
Standard Convolution vs Depthwise Separable Convolution



(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

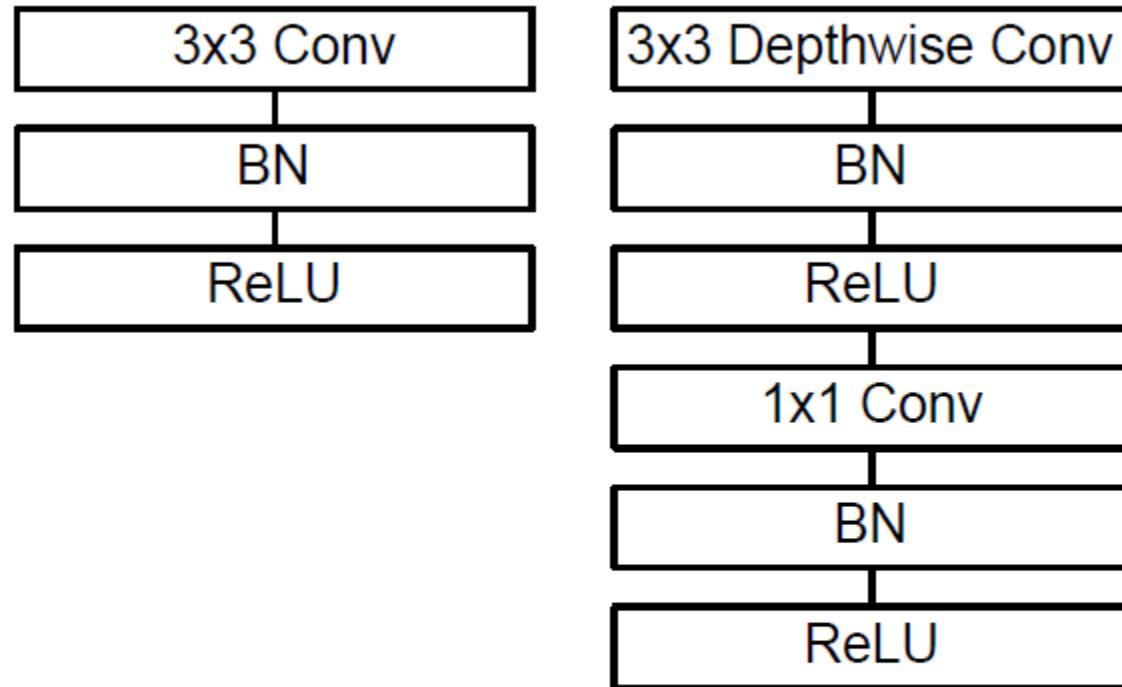


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

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× 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 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

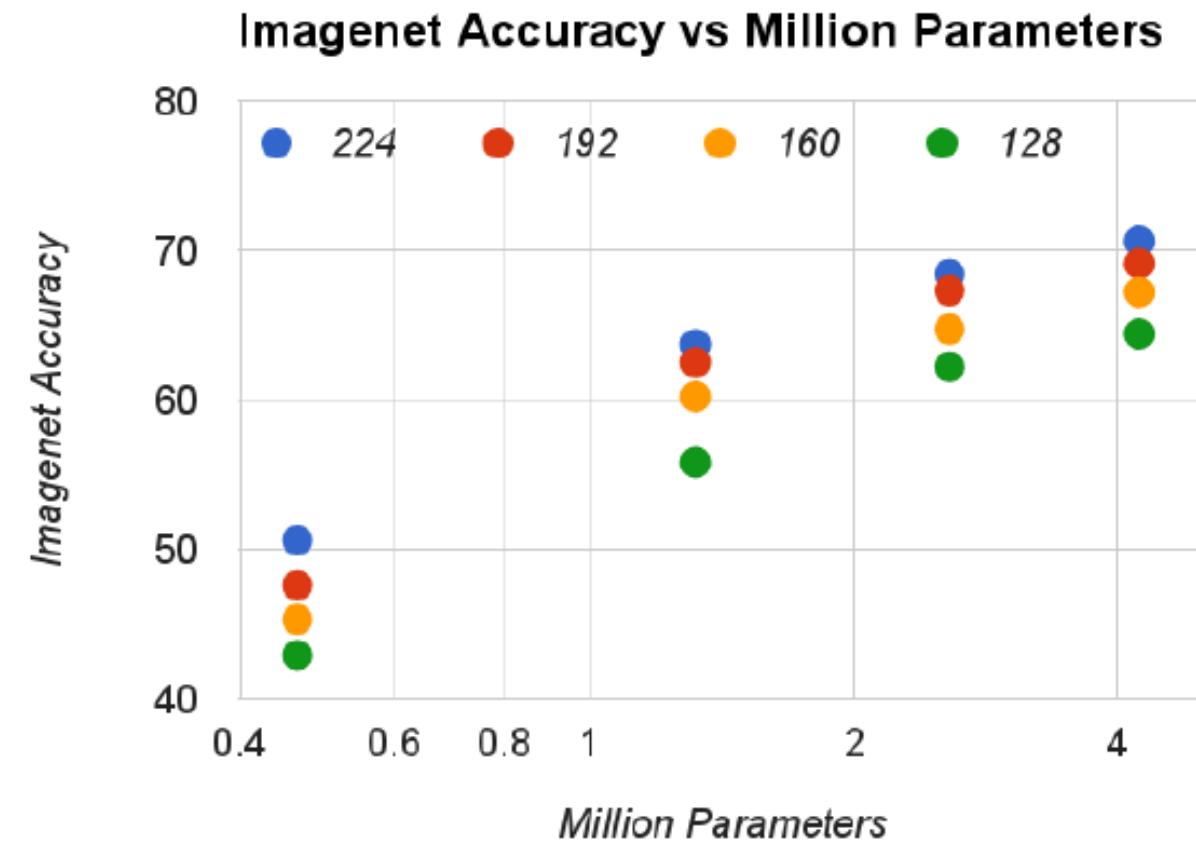
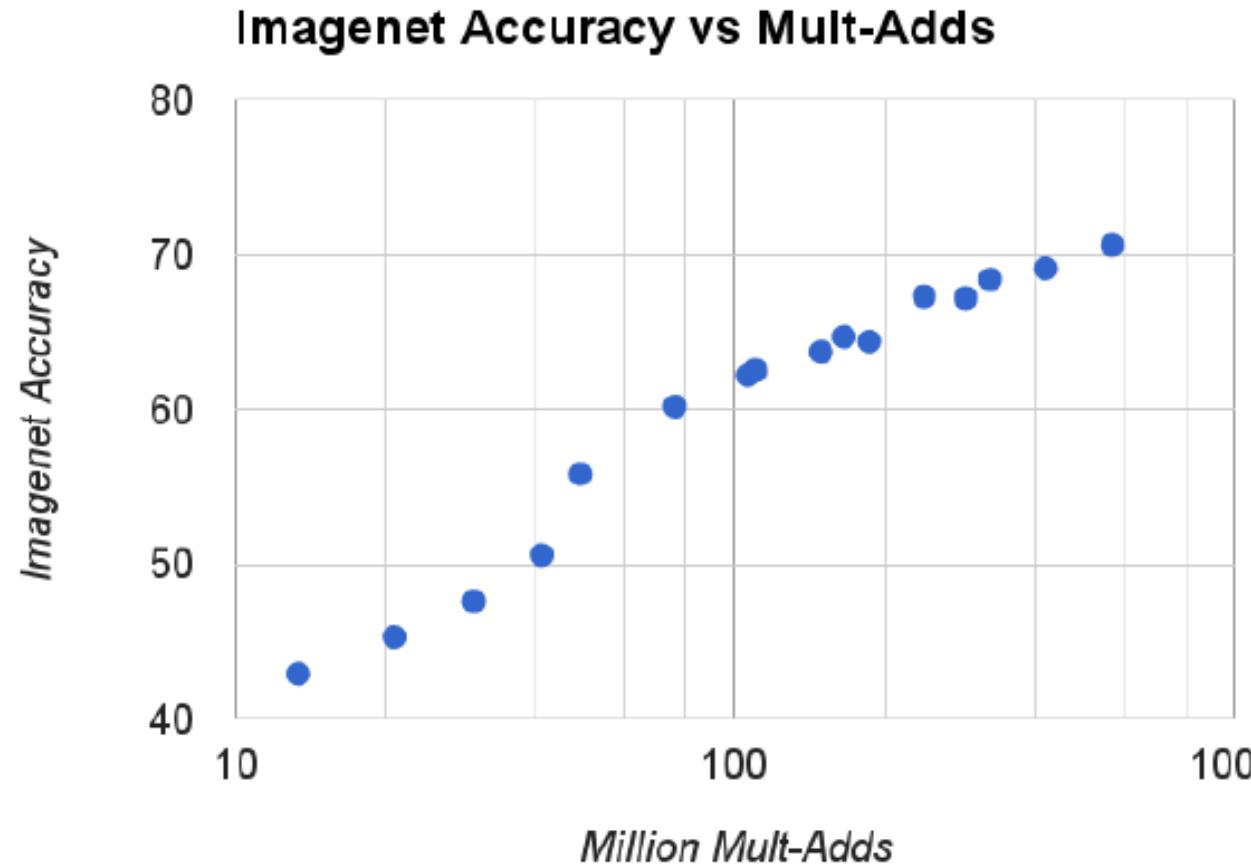
Table 5. Narrow vs Shallow MobileNet

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

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



Results

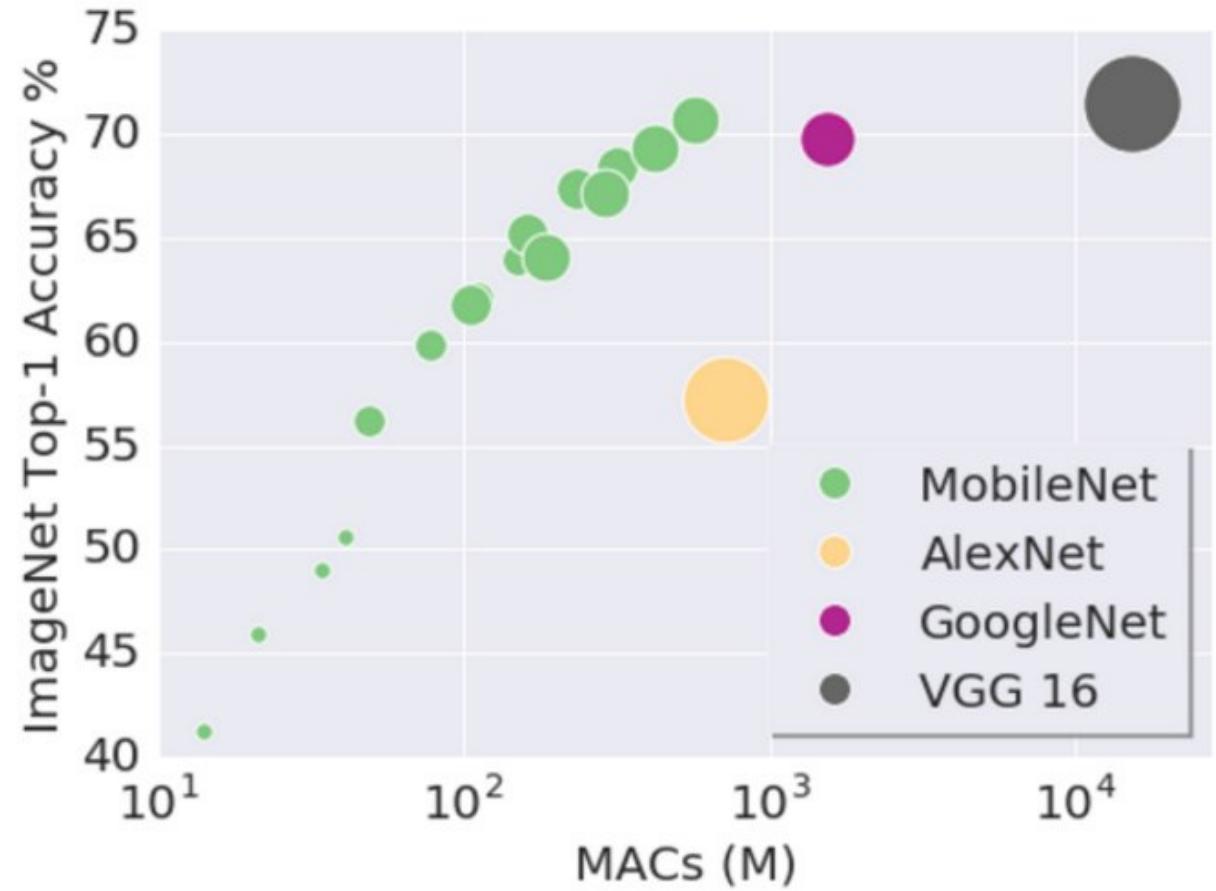
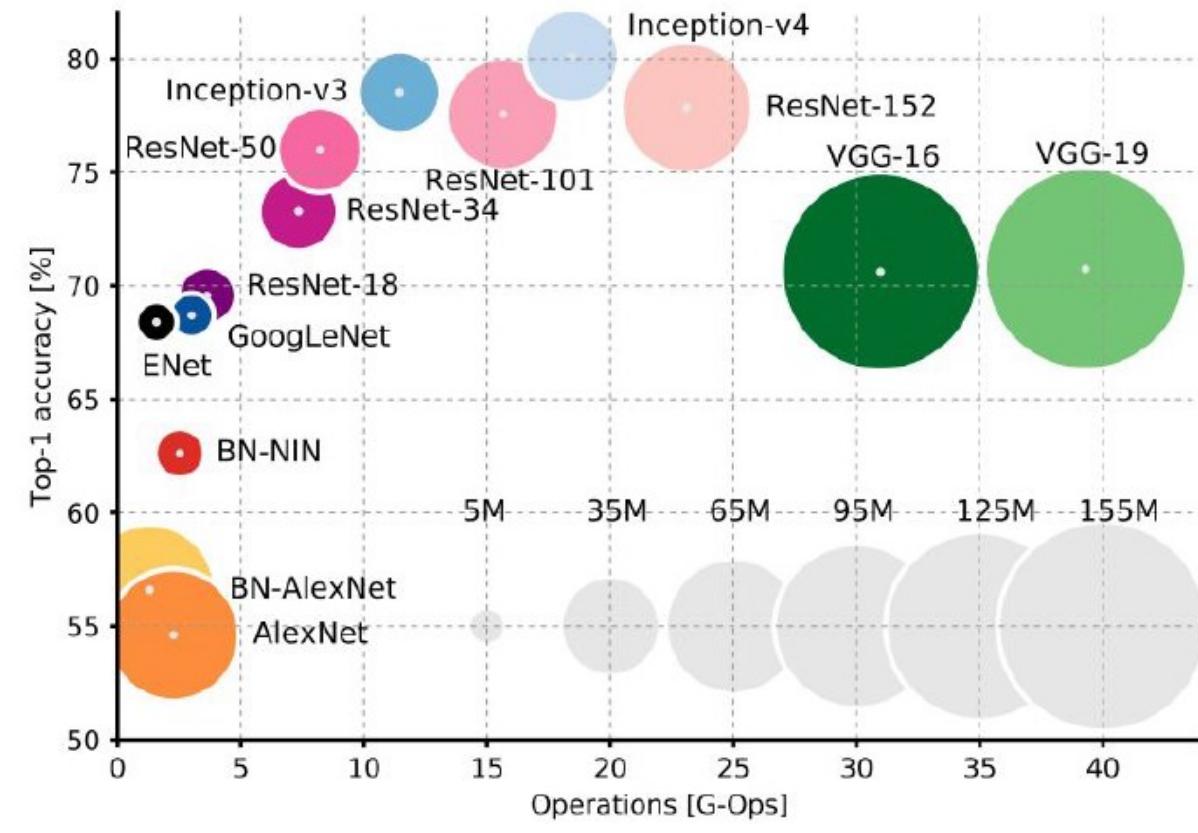
Table 8. MobileNet Comparison to Popular Models

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

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

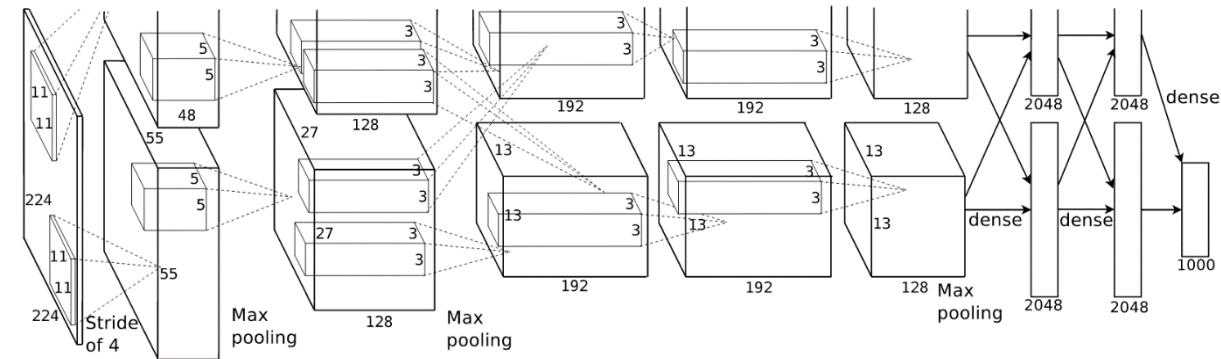
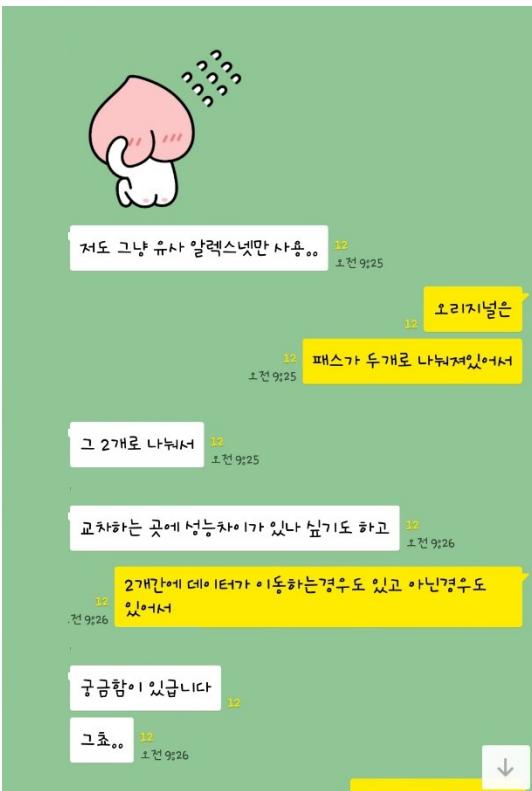
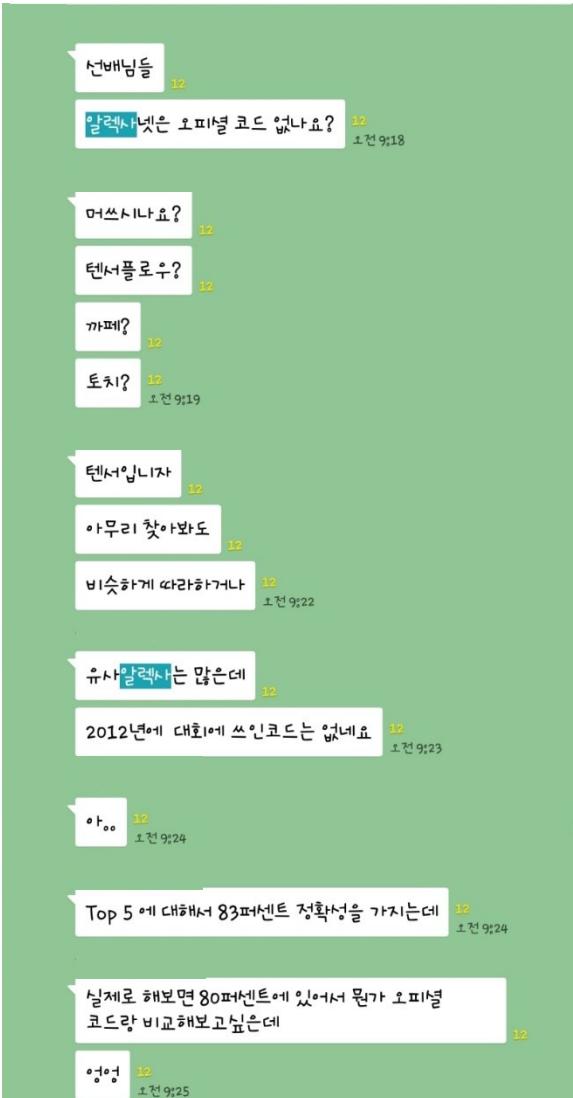
- The 1×1 convolution accounts for most of the computation

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?

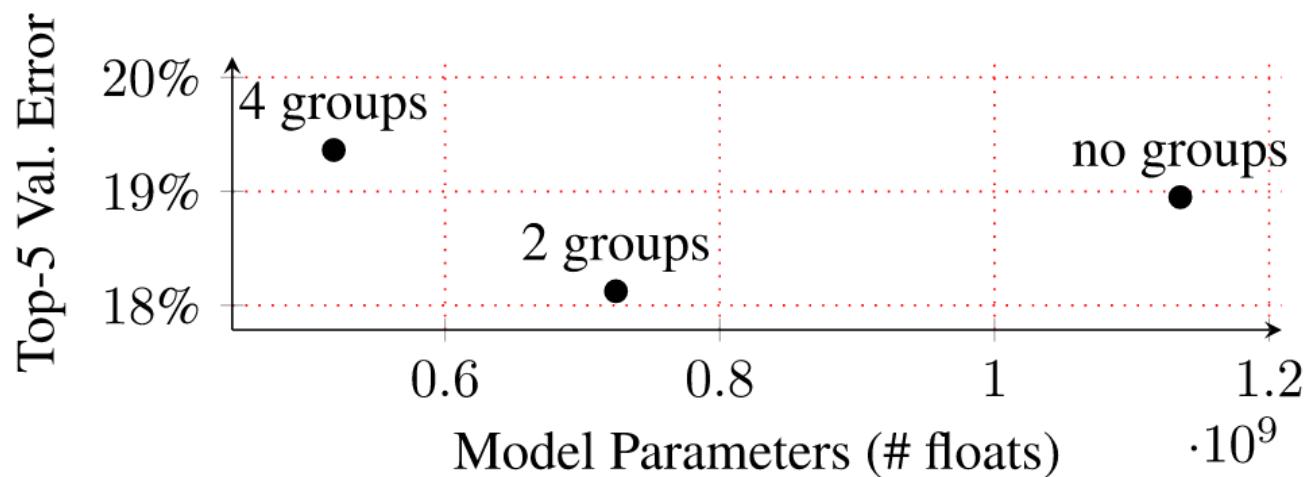
A Secret of AlexNet



Grouped Convolution!

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!

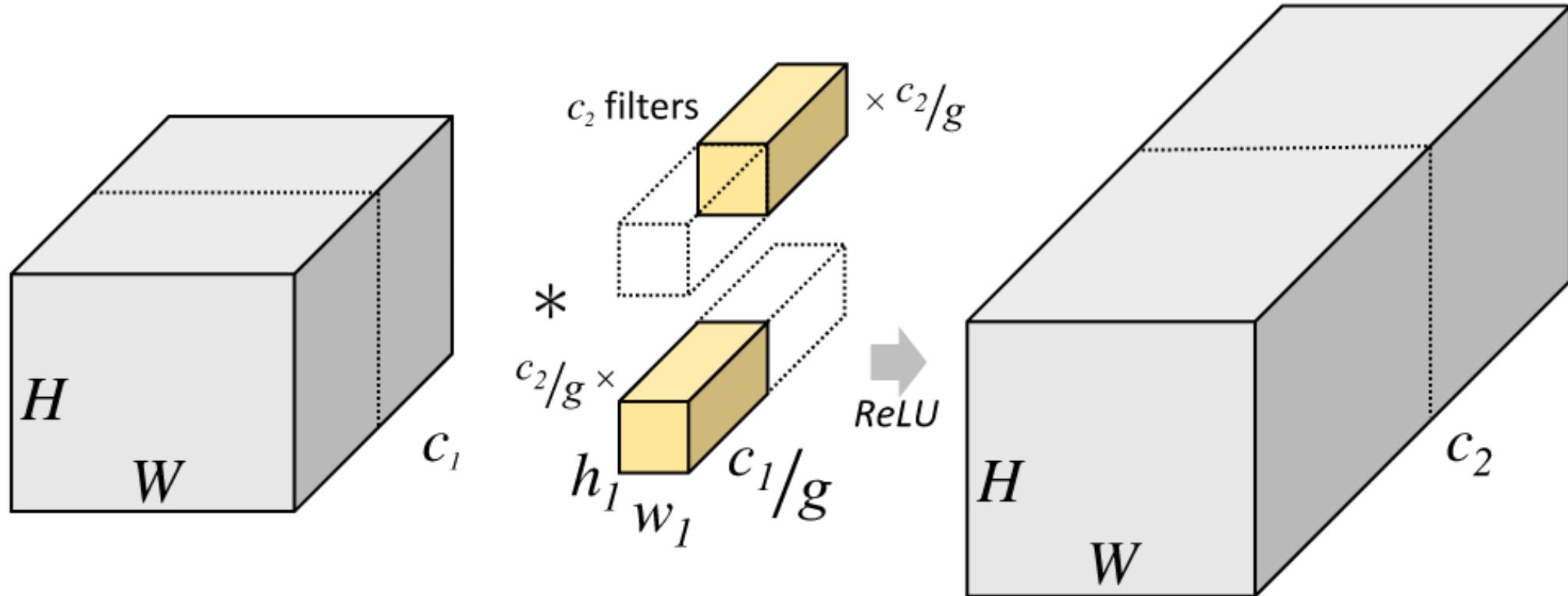


AlexNet trained with varying numbers of filter groups, from 1 (i.e. no filter groups), to 4. When trained with 2 filter groups, AlexNet is more efficient and yet achieves the same if not lower validation error.

Main Ideas of ShuffleNet

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

Grouped Convolution

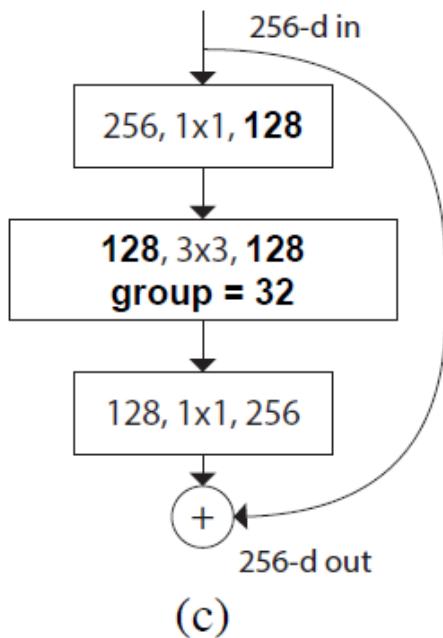
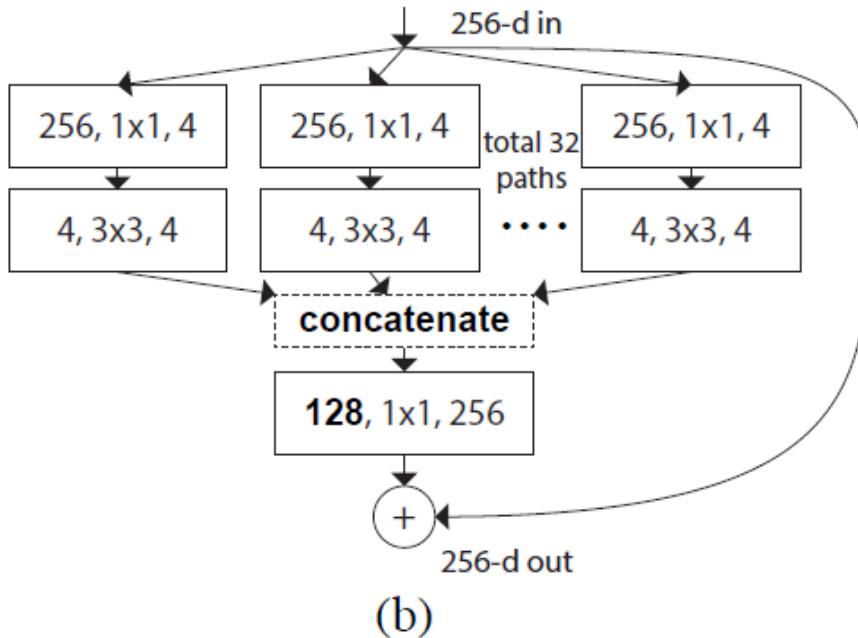
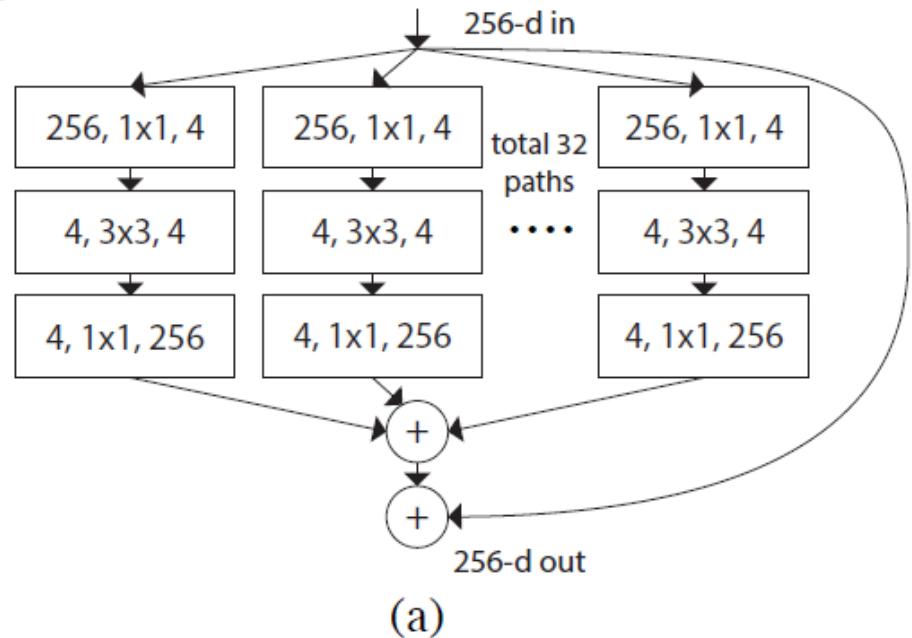


A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

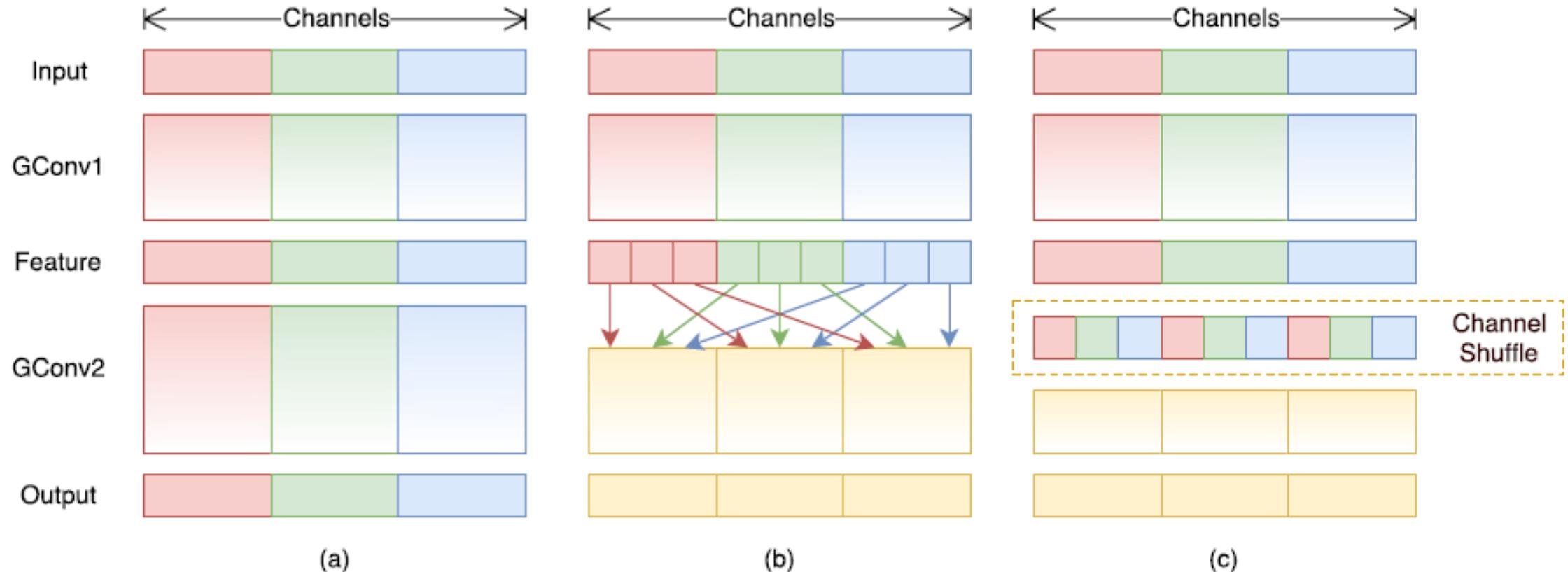
ResNeXt

- Equivalent building blocks of RexNext (cardinality=32)

equivalent



1x1 Grouped Convolution with Channel Shuffling



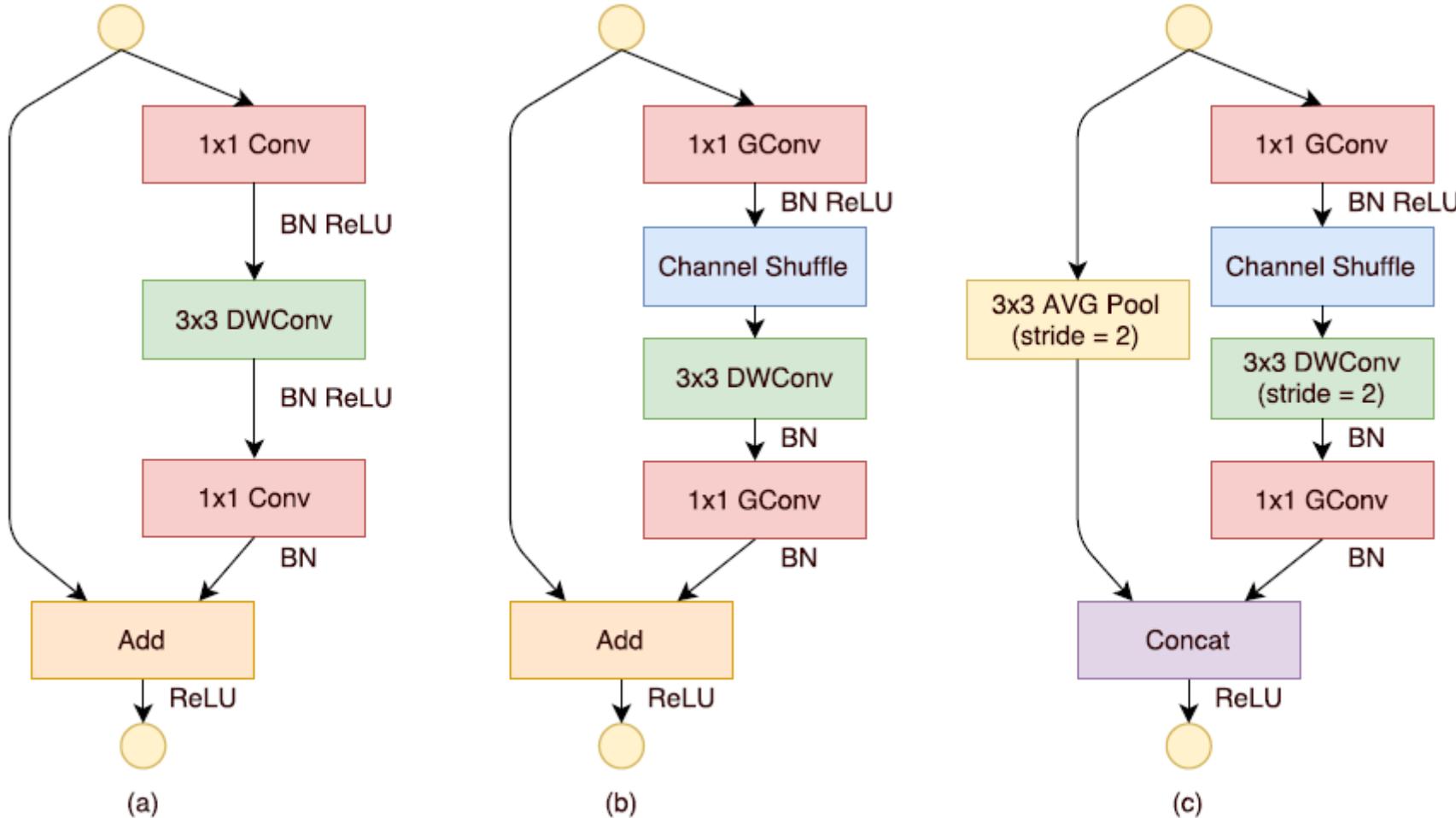
- If multiple group convolutions stack together, there is one side effect(a)
 - Outputs from a certain channel are only derived from a small fraction of input channels
- If we allow group convolution to obtain input data from different groups, the input and output channels will be fully related.

Channel Shuffle Operation

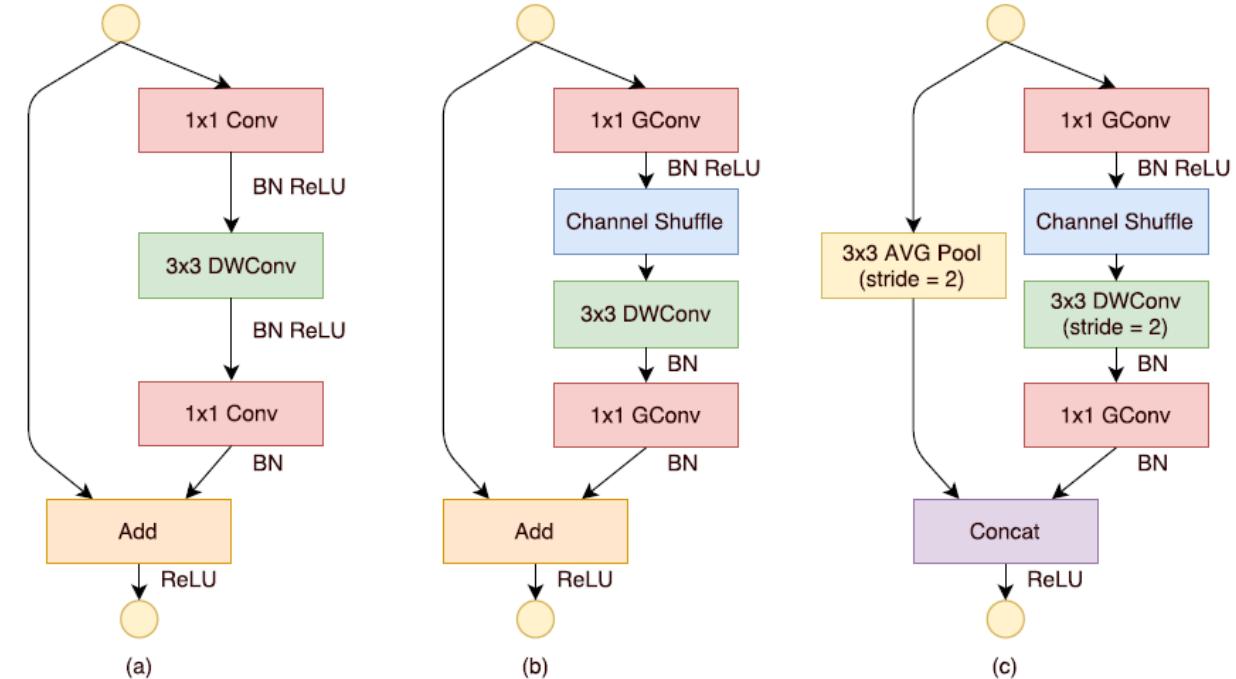
- Suppose a convolutional layer with g groups whose output has $g \times n$ channels; we first reshape the output channel dimension into (g, n) , transposing and then flattening it back as the input of next layer.
- Channel shuffle operation is also differentiable

```
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



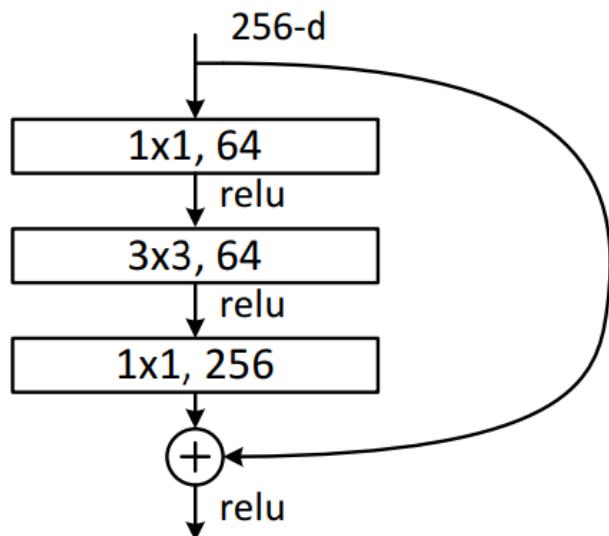
ShuffleNet Units



- 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

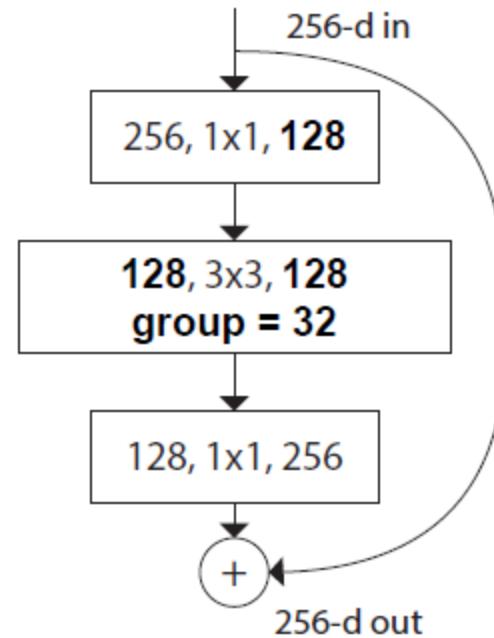
Complexity

- For example, given the input size $c \times h \times w$ and the bottleneck channel m



ResNet

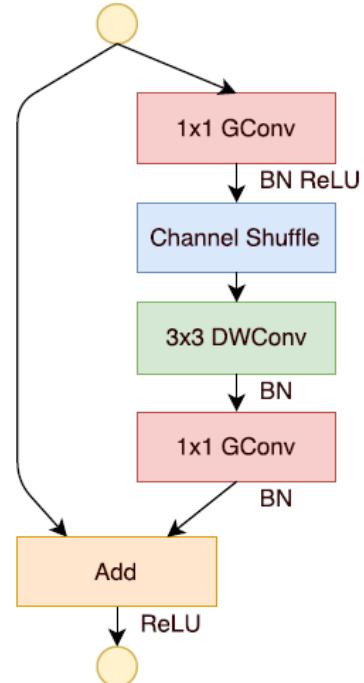
$$hw(2cm + 9m^2)$$



ResNeXt

$$hw(2cm + 9m^2/g)$$

<Number of Operations>



ShuffleNet

$$hw(2cm/g + 9m)$$

ShuffleNet Architecture

Table 1: ShuffleNet architecture

Layer	Output size	KSize	Stride	Repeat	Output channels (g groups)				
					$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
Image	224×224				3	3	3	3	3
Conv1	112×112	3×3	2	1	24	24	24	24	24
MaxPool	56×56	3×3	2						
Stage2 ¹	28×28		2	1	144	200	240	272	384
	28×28		1	3	144	200	240	272	384
Stage3	14×14		2	1	288	400	480	544	768
	14×14		1	7	288	400	480	544	768
Stage4	7×7		2	1	576	800	960	1088	1536
	7×7		1	3	576	800	960	1088	1536
GlobalPool	1×1	7×7							
FC					1000	1000	1000	1000	1000
Complexity ²					143M	140M	137M	133M	137M

Experimental Results

- To customize the network to a desired complexity, a scale factor s on the number of channels is applied
 - ShuffleNet $s \times$ means scaling the number of filters in ShuffleNet $1 \times$ by s times thus overall complexity will be roughly s^2 times of ShuffleNet $1 \times$
- Grouped convolutions ($g > 1$) consistently perform better than the counter parts without pointwise group convolutions ($g=1$). Smaller models tend to benefit more from groups

Model	Complexity (MFLOPs)	Classification error (%)				
		$g = 1$	$g = 2$	$g = 3$	$g = 4$	$g = 8$
ShuffleNet $1 \times$	140	33.6	32.7	32.6	32.8	32.4
ShuffleNet $0.5 \times$	38	45.1	44.4	43.2	41.6	42.3
ShuffleNet $0.25 \times$	13	57.1	56.8	55.0	54.2	52.7

Table 2. Classification error vs. number of groups g (smaller number represents better performance)

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)

ShuffleNet V2

ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design

Ningning Ma ^{*1,2} Xiangyu Zhang ^{*1} Hai-Tao Zheng² Jian Sun¹

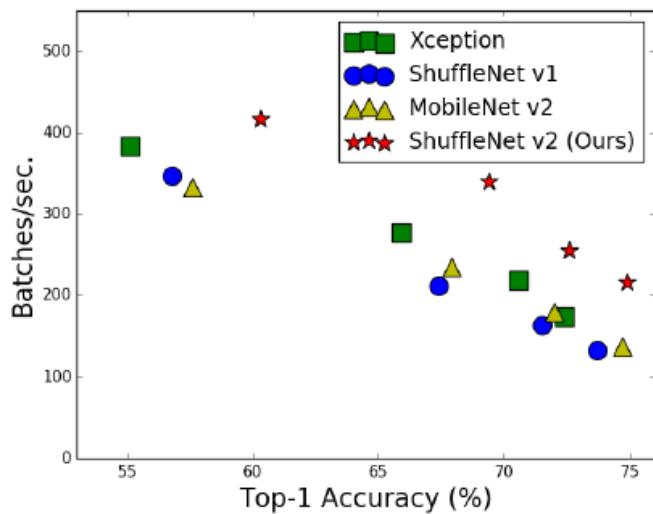
¹ Megvii Inc (Face++) ² Tsinghua University
`{maningning,zhangxiangyu,sunjian}@megvii.com`
`zheng.haitao@sz.tsinghua.edu.cn`

Abstract. Currently, the neural network architecture design is mostly guided by the *indirect* metric of computation complexity, i.e., FLOPs. However, the *direct* metric, e.g., speed, also depends on the other factors such as memory access cost and platform characteristics. Thus, this work proposes to evaluate the direct metric on the target platform, beyond only considering FLOPs. Based on a series of controlled experiments, this work derives several practical *guidelines* for efficient network design. Accordingly, a new architecture is presented, called *ShuffleNet V2*. Comprehensive ablation experiments verify that our model is the state-of-the-art in terms of speed and accuracy tradeoff.

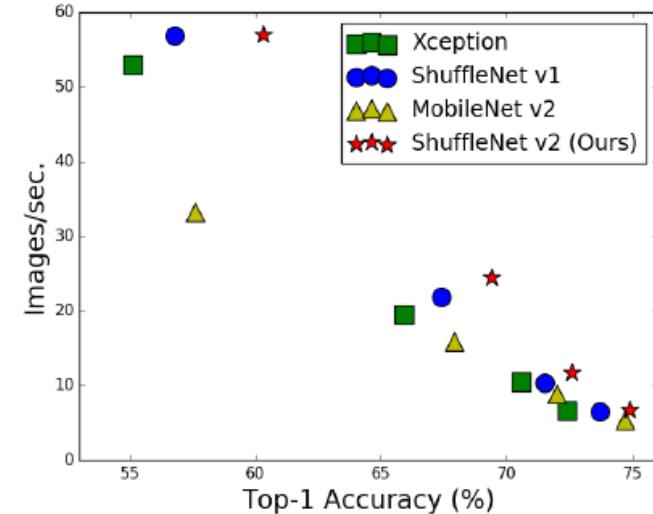
Motivation

- Currently, the neural network architecture design is mostly guided by **the indirect metric of computation complexity**, i.e., FLOPs.
- However, **the direct metric**, e.g., speed, also depends on the other factors such as memory access cost and platform characteristics.
- Indirect metric is usually not equivalent to the direct metric that we really care about, such as speed or latency.
- For example, MobileNet v2 is much faster than NASNET-A but they have comparable FLOPs.

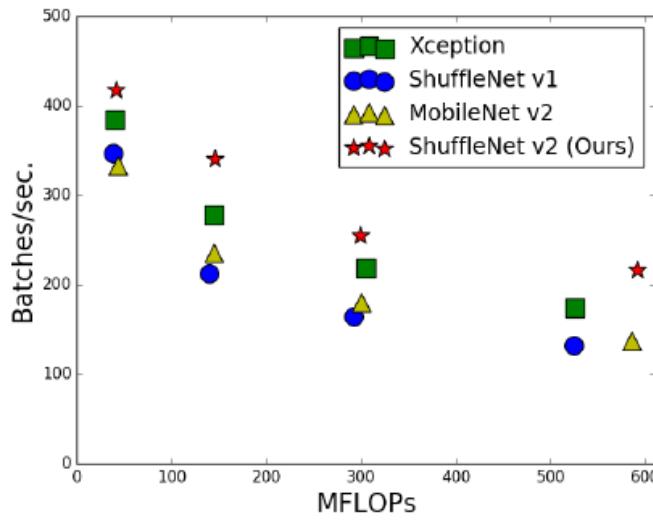
Accuracy, Speed and FLOPs



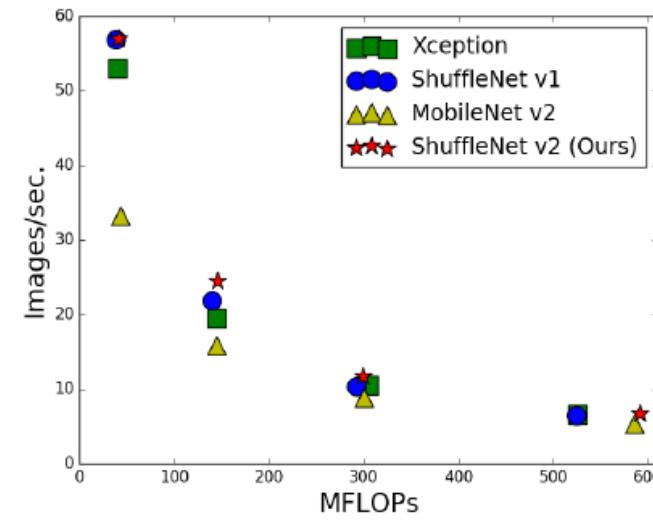
(a) GPU



(b) ARM



(c) GPU



(d) ARM

Two Reasons of the Discrepancy

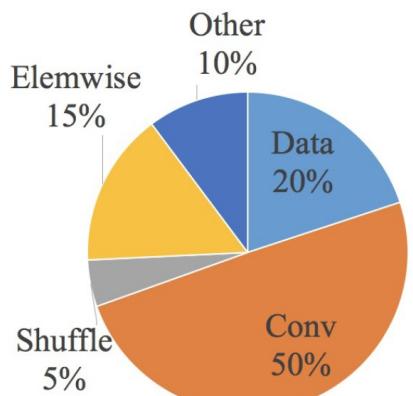
- First, several important factors that have considerable affection on speed are not taken into account by FLOPs. One such factor is memory access cost (MAC).
- Second, operations with the same FLOPs could have different running time, depending on the platform.

Hardware Platform for Experiments

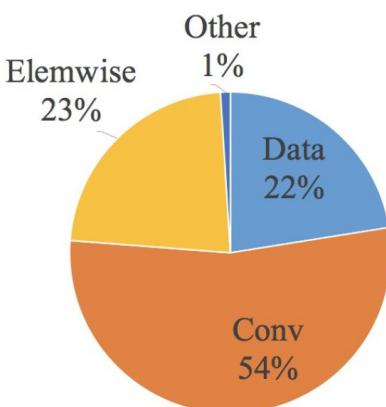
- GPU – A single NVIDIA GeForce GTX 1080Ti is used. The convolution library is CUDNN 7.0.
- ARM. A Qualcomm Snapdragon 810. We use a highly-optimized Neon-based implementation. A single thread is used for evaluation.

Analysis of the Runtime Performance (ShuffleNet v1 and MobileNet v2)

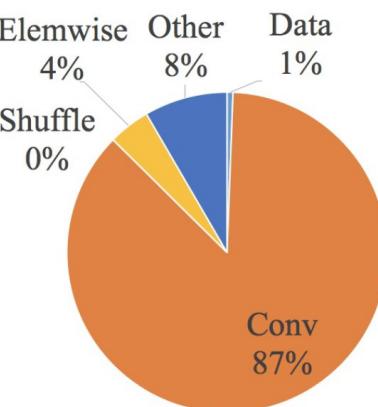
- FLOPs metric only account for the convolution part.
- Although this part consumes most time, the other operations including data I/O, data shuffle and element-wise operations also occupy considerable amount of time.



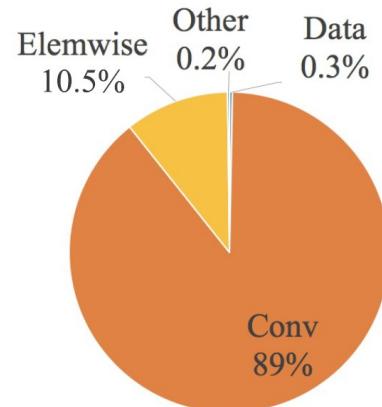
ShuffleNet V1 on GPU



MobileNet V2 on GPU



ShuffleNet V1 on ARM



MobileNet V2 on ARM

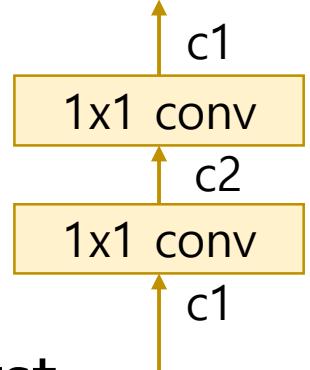
Guide 1

- 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.

$$MAC \geq 2\sqrt{hwB} + \frac{B}{hw} \quad (1)$$

Guide 1

- A benchmark network is built by stacking **10 building blocks repeatedly**. Each block contains two convolution layers. The first contains c_1 input channels and c_2 output channels, and the second otherwise.



		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 (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 .

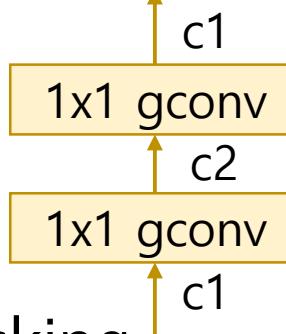
Guide 2

- **Excessive group convolution increases MAC.**
- Formally, following the notations in G1 and Eq. 1, the relation between MAC and FLOPs for 1×1 group convolution is

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

- Given the fixed input shape $c_1 \text{ h w}$ and the computational cost B , MAC increases with the growth of g .

Guide 2



- To study the affection in practice, a benchmark network is built by stacking 10 pointwise group convolution layers.

g	c for $\times 1$	GPU (Batches/sec.)			c for $\times 1$	CPU (Images/sec.)		
		$\times 1$	$\times 2$	$\times 4$		$\times 1$	$\times 2$	$\times 4$
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

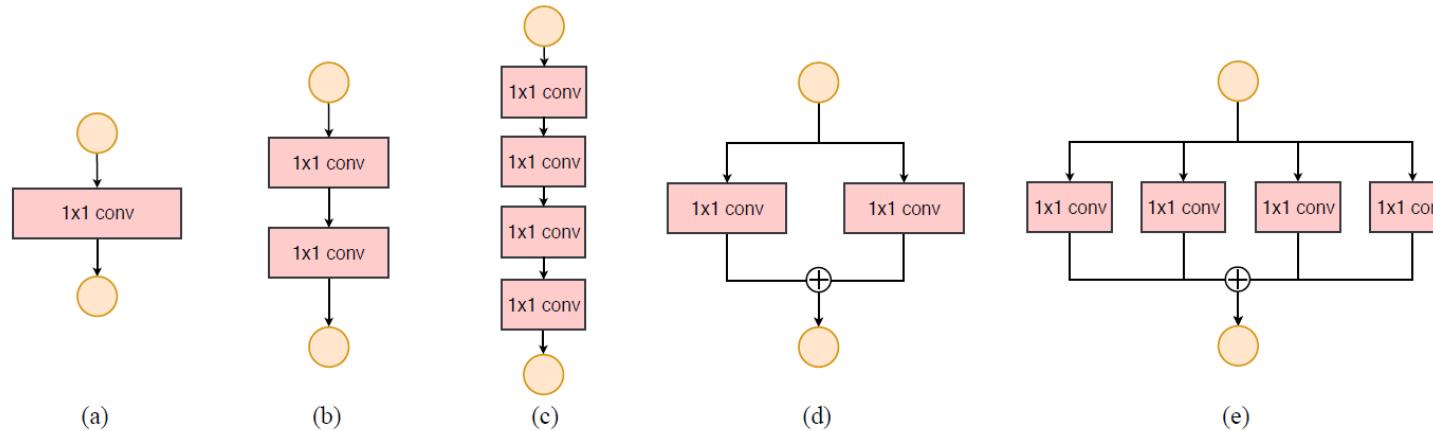
Table 2: Validation experiment for **Guideline 2**. Four values of group number g are tested, while the total FLOPs under the four values is fixed by varying the total channel number c . Input image size is 56×56 .

- The group number should be carefully chosen based on the target platform and task. It is unwise to use a large group number simply because this may enable using more channels, because the benefit of accuracy increase can easily be outweighed by the rapidly increasing computational cost.*

Guide 3

- **Network fragmentation reduces degree of parallelism.**
- In the GoogLeNet series and auto-generated architectures a “multi-path” structure is widely adopted in each network block.
- A lot of small operators (called “fragmented operators” here) are used instead of a few large ones.
- Though such fragmented structure has been shown beneficial for accuracy, **it could decrease efficiency because it is unfriendly for devices with strong parallel computing powers like GPU**. It also introduces extra overheads such as kernel launching and synchronization.

Guide 3



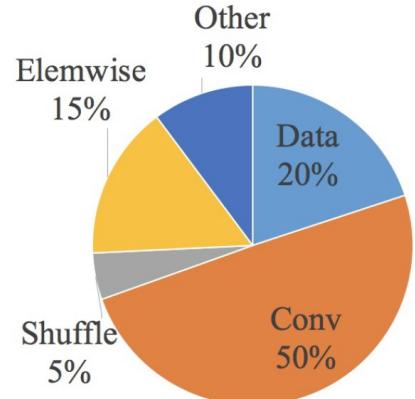
Appendix Fig. 1: Building blocks used in experiments for guideline 3. (a) *1-fragment*. (b) *2-fragment-series*. (c) *4-fragment-series*. (d) *2-fragment-parallel*. (e) *4-fragment-parallel*.

	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

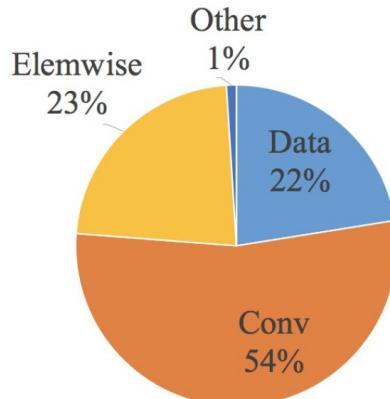
Table 3: Validation experiment for **Guideline 3**. c denotes the number of channels for *1-fragment*. The channel number in other fragmented structures is adjusted so that the FLOPs is the same as *1-fragment*. Input image size is 56×56 .

Guide 4

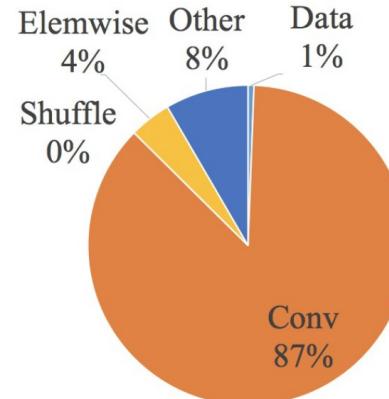
- **Element-wise operations are non-negligible.**
- Here, the element-wise operators include ReLU, AddTensor, AddBias, etc. They have small FLOPs but relatively heavy MAC.
- The authors also consider depthwise convolution as an element-wise operator as it also has a high MAC/FLOPs ratio



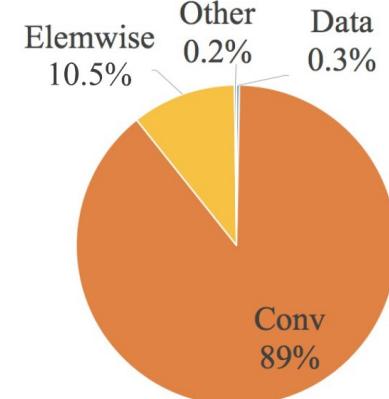
ShuffleNet V1 on GPU



MobileNet V2 on GPU



ShuffleNet V1 on ARM



MobileNet V2 on ARM

Guide 4

- Around 20% speedup is obtained on both GPU and ARM, after ReLU and shortcut are removed.

		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

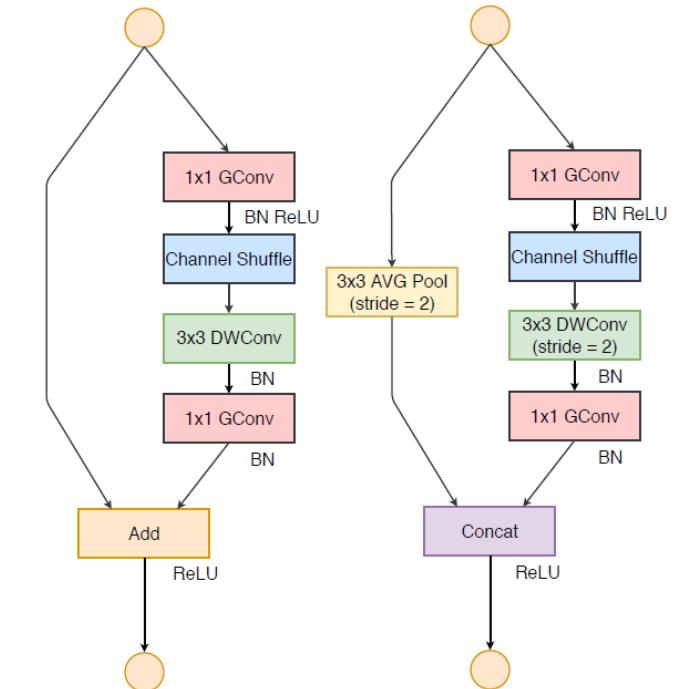
Table 4: Validation experiment for **Guideline 4**. The ReLU and shortcut operations are removed from the “bottleneck” unit [4], separately. c is the number of channels in unit. The unit is stacked repeatedly for 10 times to benchmark the speed.

Conclusion and Discussion

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.
-
- ShuffleNet v1 heavily depends group convolutions (against G2) and bottleneck-like building blocks (against G1).
 - MobileNet v2 uses an inverted bottleneck structure that violates G1. It uses depthwise convolutions and ReLUs on "thick" feature maps. This violates G4.
 - The auto-generated structures are highly fragmented and violate G3.

Review of ShuffleNet V1

- Both **pointwise group convolutions and bottleneck structures increase MAC** (G_1 and G_2). This cost is non-negligible, especially for light-weight models. Also, **using too many groups violates G_3** . The **element-wise "Add"** operation in the shortcut connection is also undesirable (G_4).
- Therefore, in order to achieve high model capacity and efficiency, the key issue is how to maintain a large number and equally wide channels with neither dense convolution nor too many groups.



Building Blocks of ShuffleNet V1 and V2

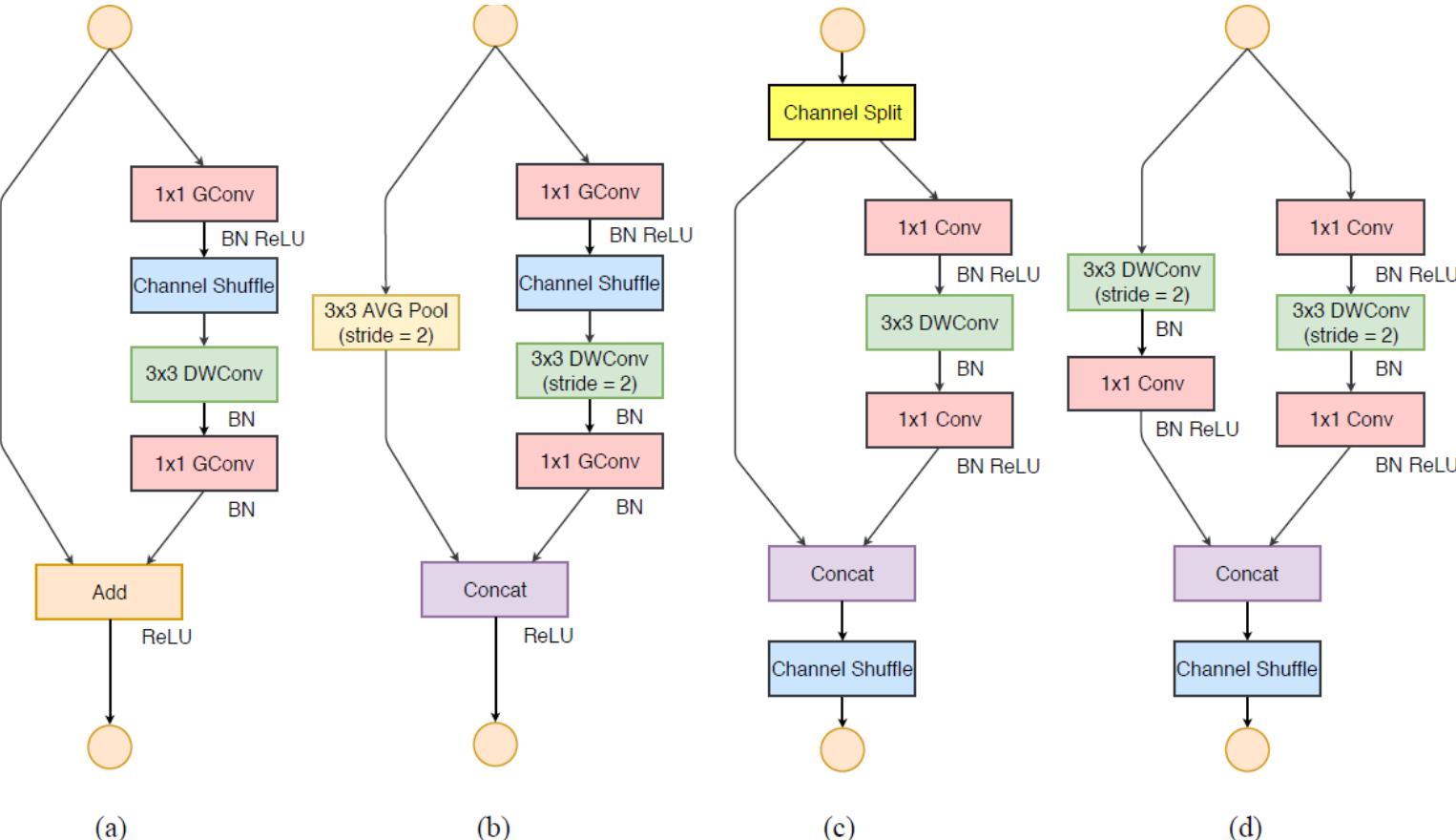


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling ($2\times$); (c) our basic unit; (d) our unit for spatial down sampling ($2\times$). **DWConv**: depthwise convolution. **GConv**: group convolution.

Building Blocks of ShuffleNet V2

- At the beginning of each unit, the input of **c feature channels are split into two branches with $c - c'$ and c' channels**, respectively. ($c' = c/2$ in this paper)
- Following G₃, **one branch remains as identity**. The other branch consists of **three convolutions with the same input and output channels** to satisfy G₁.
- **The two 1×1 convolutions are no longer group-wise**. This is partially to follow G₂, and partially because the split operation already produces two groups.
- After convolution, the two branches are concatenated. So, **the number of channels keeps the same (G₁)**.
- Note that **the "Add" operation in ShuffleNet v1 no longer exists**. Element-wise operations like ReLU and depthwise convolutions exist only in one branch.
- For spatial down sampling, the unit is slightly modified. The channel split operator is removed.

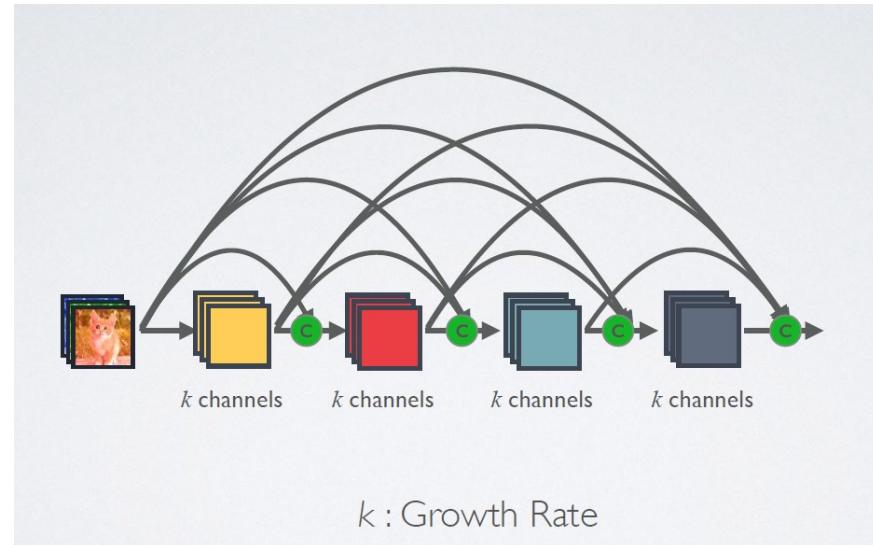
ShuffleNet V2

Layer	Output size	KSize	Stride	Repeat	Output channels			
					0.5×	1×	1.5×	2×
Image	224×224				3	3	3	3
Conv1	112×112	3×3	2		24	24	24	24
MaxPool	56×56	3×3	2	1				
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

Table 5: Overall architecture of ShuffleNet v2, for four different levels of complexities.

Analysis of Network Accuracy

- ShuffleNet v2 is not only efficient, but also accurate.
- The high efficiency in each building block enables using more feature channels and larger network capacity.
- In each block, half of feature channels(when $c'=c/2$) directly go through the block and join the next block. This can be regarded as a kind of feature reuse, in a similar spirit as in DenseNet.



Similar to DenseNet

- In DenseNet, to analyze the feature reuse pattern, the l_1 -norm of the weights between layers are plotted.
- In shuffleNet V2, it is easy to prove that the number of “directly-connected” channels between i -th and $(i+j)$ -th building block is $r^j c$, where $r = (1 - c')/c$.

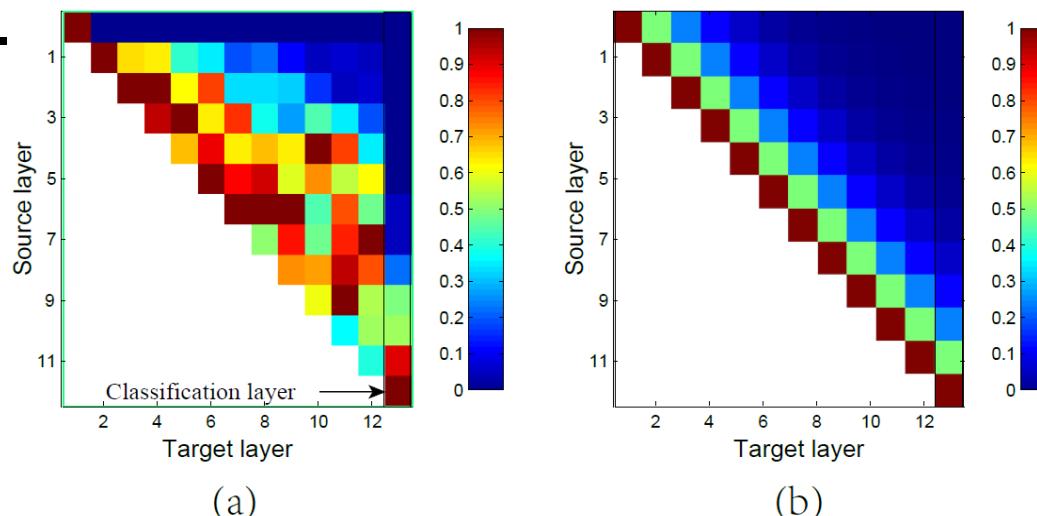


Fig. 4: Illustration of the patterns in feature reuse for *DenseNet* [6] and *ShuffleNet V2*. (a) (courtesy of [6]) the average absolute filter weight of convolutional layers in a model. The color of pixel (s, l) encodes the average l_1 -norm of weights connecting layer s to l . (b) The color of pixel (s, l) means the number of channels *directly* connecting block s to block l in *ShuffleNet v2*. All pixel values are normalized to $[0, 1]$.

Experiment

- Compared Networks
 - ShuffleNet V1
 - MobileNet V2
 - Xception
 - DenseNet

Results

Model	Complexity (MFLOPs)	Top-1 err. (%)	GPU Speed (Batches/sec.)	ARM Speed (Images/sec.)
ShuffleNet v2 0.5× (ours)	41	39.7	417	57.0
0.25 MobileNet v1 [13]	41	49.4	502	36.4
0.4 MobileNet v2 [14] (our impl.)*	43	43.4	333	33.2
0.15 MobileNet v2 [14] (our impl.)	39	55.1	351	33.6
ShuffleNet v1 0.5× (g=3) [15]	38	43.2	347	56.8
DenseNet 0.5× [6] (our impl.)	42	58.6	366	39.7
Xception 0.5× [12] (our impl.)	40	44.9	384	52.9
IGCV2-0.25 [27]	46	45.1	183	31.5
ShuffleNet v2 1× (ours)	146	30.6	341	24.4
0.5 MobileNet v1 [13]	149	36.3	382	16.5
0.75 MobileNet v2 [14] (our impl.)**	145	32.1	235	15.9
0.6 MobileNet v2 [14] (our impl.)	141	33.3	249	14.9
ShuffleNet v1 1× (g=3) [15]	140	32.6	213	21.8
DenseNet 1× [6] (our impl.)	142	45.2	279	15.8
Xception 1× [12] (our impl.)	145	34.1	278	19.5
IGCV2-0.5 [27]	156	34.5	132	15.5
IGCV3-D (0.7) [28]	210	31.5	143	11.7

ShuffleNet v2 1.5× (ours)	<u>299</u>	27.4	<u>255</u>	11.8
0.75 MobileNet v1 [13]	325	31.6	314	10.6
1.0 MobileNet v2 [14]	300	28.0	180	8.9
1.0 MobileNet v2 [14] (our impl.)	301	28.3	180	8.9
ShuffleNet v1 1.5× (g=3) [15]	292	28.5	164	10.3
DenseNet 1.5× [6] (our impl.)	295	39.9	274	9.7
CondenseNet (G=C=8) [16]	274	29.0	-	-
Xception 1.5× [12] (our impl.)	305	29.4	219	10.5
IGCV3-D [28]	318	27.8	102	6.3
ShuffleNet v2 2× (ours)	<u>591</u>	25.1	<u>217</u>	6.7
1.0 MobileNet v1 [13]	569	29.4	247	6.5
1.4 MobileNet v2 [14]	585	25.3	137	5.4
1.4 MobileNet v2 [14] (our impl.)	587	26.7	137	5.4
ShuffleNet v1 2× (g=3) [15]	524	26.3	133	6.4
DenseNet 2× [6] (our impl.)	519	34.6	197	6.1
CondenseNet (G=C=4) [16]	529	26.2	-	-
Xception 2× [12] (our impl.)	525	27.6	174	6.7
IGCV2-1.0 [27]	564	29.3	81	4.9
IGCV3-D (1.4) [28]	610	25.5	82	4.5
ShuffleNet v2 2x (ours, with SE [8])	<u>597</u>	24.6	<u>161</u>	5.6
NASNet-A [9] (4 @ 1056, our impl.)	564	26.0	130	4.6
PNASNet-5 [10] (our impl.)	588	25.8	115	4.1

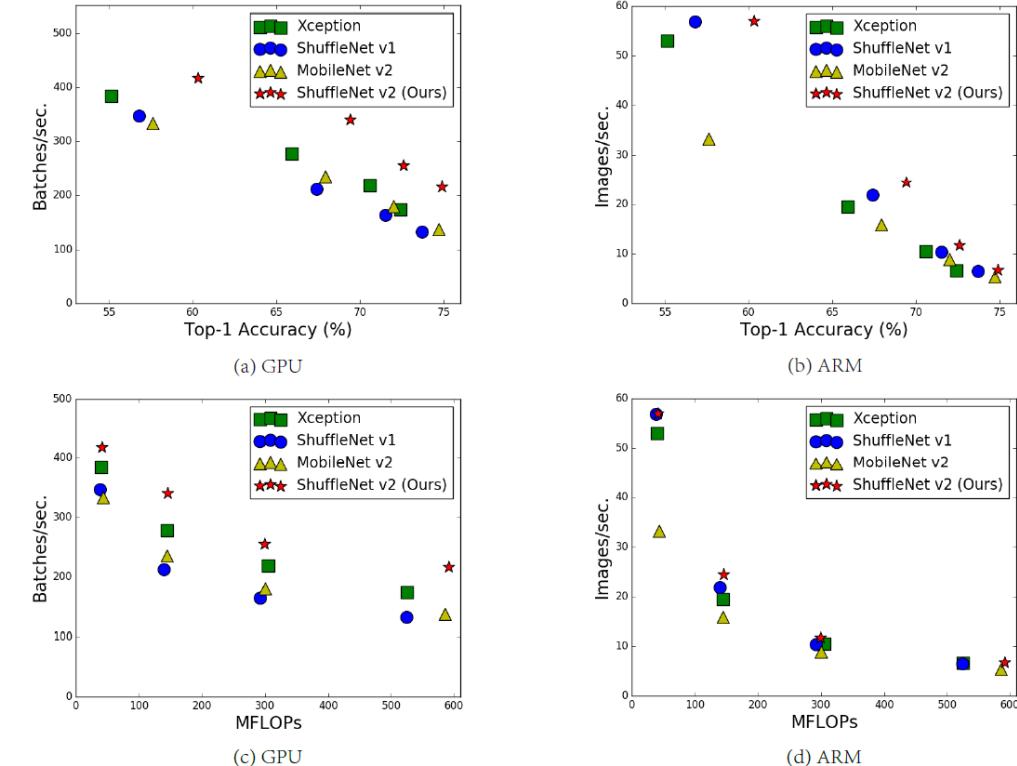
Accuracy vs FLOPs

- ShuffleNet V2 models outperform all other networks by a large margin
- Also, we note that MobileNet V2 performs poorly at 40 MFLOPs level with 224×224 image size. This is probably caused by too few channels.

Inference Speed vs FLOPs/Accuracy

- ShuffleNet V2 is clearly faster than the other three networks, especially on GPU.
- MobileNet V2 is much slower, especially on smaller FLOPs. (because of higher MAC which is significant on mobile devices.)

Input size	FLOPs	GPU (Batches/sec.)				CPU (Images/sec.)			
		40M	140M	300M	500M	40M	140M	300M	500M
320x320	ShuffleNet v2	315*	525	474	422	28.1	12.5	6.1	3.4
	ShuffleNet v1	236*	414	344	275	27.2	11.4	5.1	3.1
	MobileNet v2	187*	460	389	335	11.4	6.4	4.6	2.7
	Xception	279*	463	408	350	31.1	10.1	5.6	3.5
640x480	ShuffleNet v2	424	394	297	250	9.3	4.0	1.9	1.1
	ShuffleNet v1	396	269	198	156	8.0	3.7	1.6	1.0
	MobileNet v2	338	248	208	165	3.8	2.0	1.4	0.8
	Xception	399	326	244	209	9.6	3.2	1.7	1.1
1080x720	ShuffleNet v2	248	197	141	115	3.5	1.5	0.7	0.4
	ShuffleNet v1	203	131	96	77	2.9	1.4	0.4	0.3
	MobileNet v2	159	117	99	78	1.4	0.7	0.3	0.3
	Xception	232	160	124	106	3.6	1.2	0.5	0.4



Compared with MobileNet V1

- Although the accuracy of MobileNet V1 is not as good, its speed on GPU is faster than all the counterparts, including ShuffleNet V2.
- This is because its structure satisfies most of proposed guidelines (e.g. for G₃, the fragments of MobileNet V1 are even fewer than ShuffleNet v2).

Compare with AutoML Models

- AutoML models' speeds are relatively slow.
- This is mainly due to the usage of too many fragments.
- If model search algorithms are combined with proposed guidelines, and the direct metric(speed) is evaluated on the target platform.

ShuffleNet v2 2x (ours, with <i>SE</i> [8])	<u>597</u>	24.6	<u>161</u>	5.6
NASNet-A [9] (4 @ 1056, our impl.)	564	26.0	130	4.6
PNASNet-5 [10] (our impl.)	588	25.8	115	4.1

Generalization to Large Models

- ShuffleNet V2 still outperforms ShuffleNet V1 at 2.3GFLOPs and surpasses ResNet-50 with 40% fewer FLOPs.
- For very deep ShuffleNet V2 (e.g. over 100 layers), for the training to converge faster, authors **slightly modify the basic ShuffleNet V2 unit by adding a residual path**.
- Below table presents a ShuffleNet V2 model of 164 layers equipped with SE components. It obtains superior accuracy over the previous state-of-the-art models with much fewer FLOPs.

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

Object Detection

- Xception is good on detection task. This is probably due to the larger receptive field of Xception building blocks than the other counterparts.
- The authors also enlarge the receptive field of ShuffleNet v2 by introducing an additional 3×3 depthwise convolution before the first pointwise convolution in each building block.(ShuffleNet V2*)

Model	mmAP(%)				GPU Speed (Images/sec.)			
	40M	140M	300M	500M	40M	140M	300M	500M
FLOPs								
Xception	21.9	29.0	31.3	32.9	178	131	101	83
ShuffleNet v1	20.9	27.0	29.9	32.9	152	85	76	60
MobileNet v2	20.7	24.4	30.0	30.6	146	111	94	72
ShuffleNet v2 (ours)	22.5	29.0	31.8	33.3	188	146	109	87
ShuffleNet v2* (ours)	23.7	29.6	32.2	34.2	183	138	105	83