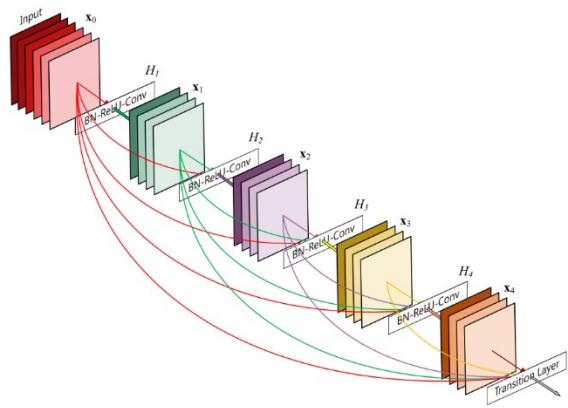


Modern CNNs

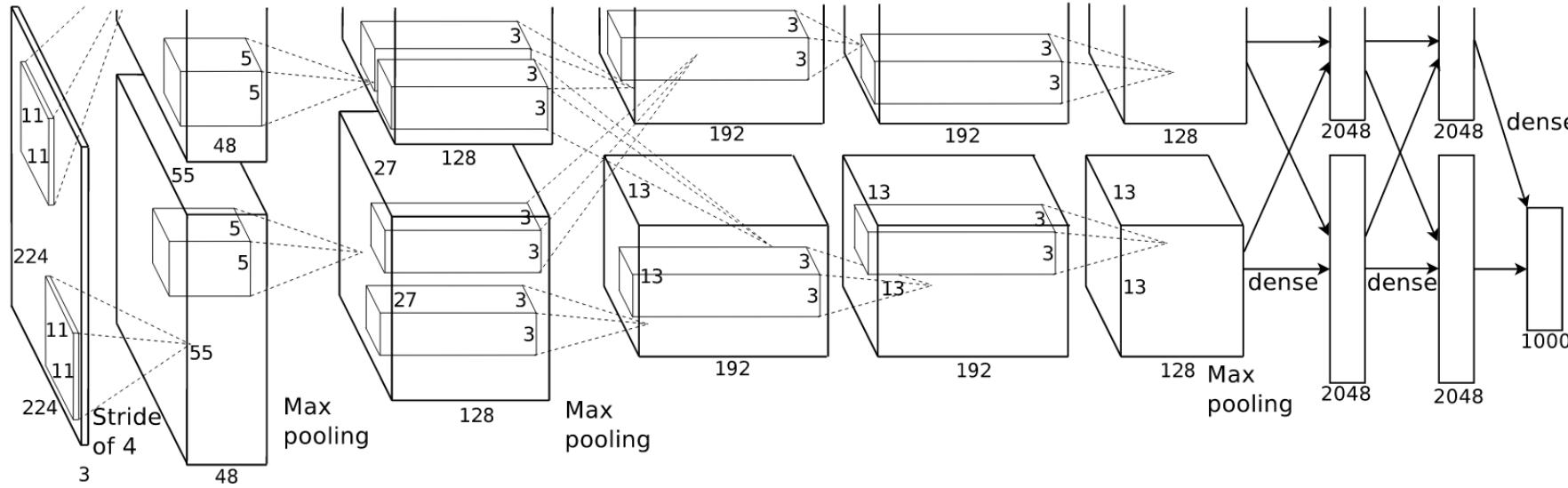


Fast Campus
Start Deep Learning with Tensorflow

Recap – History(?) of CNN

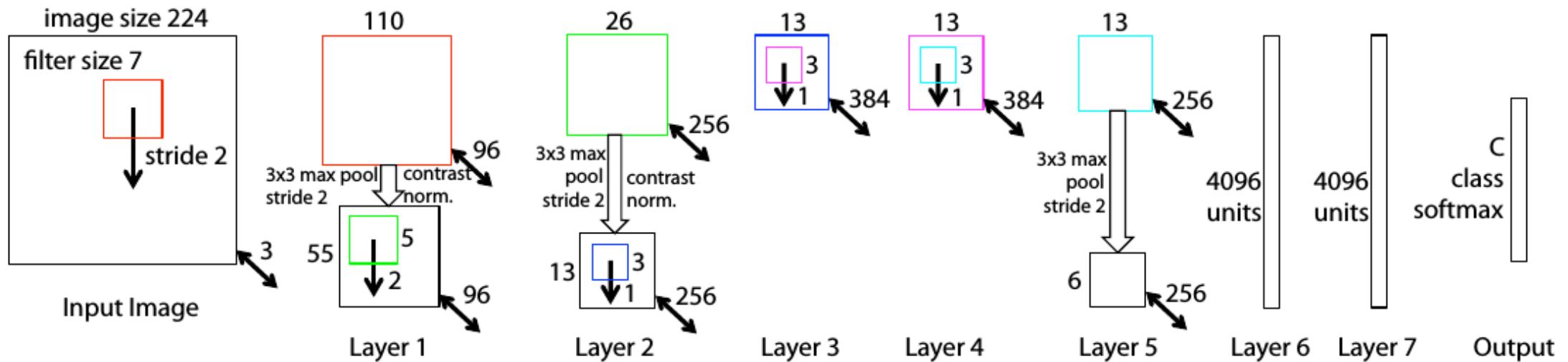
- AlexNet
- ZFNet
- VGGNet
- GoogLeNet
- ResNet

Recap - AlexNet (Krizhevsky, 2012)



- First use of ReLU
- Used norm layers(not common anymore)
- Data Augmentation
- Dropout 0.5
- Batch size 128
- SGD Momentum 0.9
- Learning rate 0.01, reduced by 10
- L₂ weight decay 5e-4
- 7 CNN ensemble: 18.2% → 15.4%

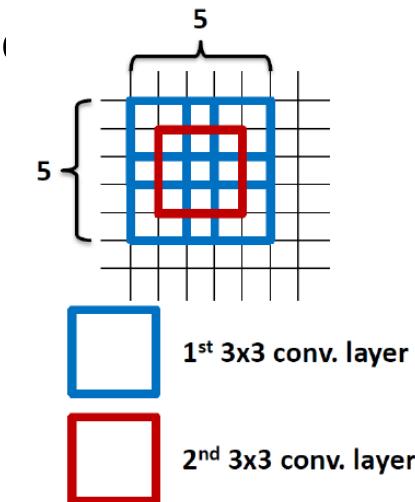
Recap - ZFNet (Zeiler and Fergus, 2013)



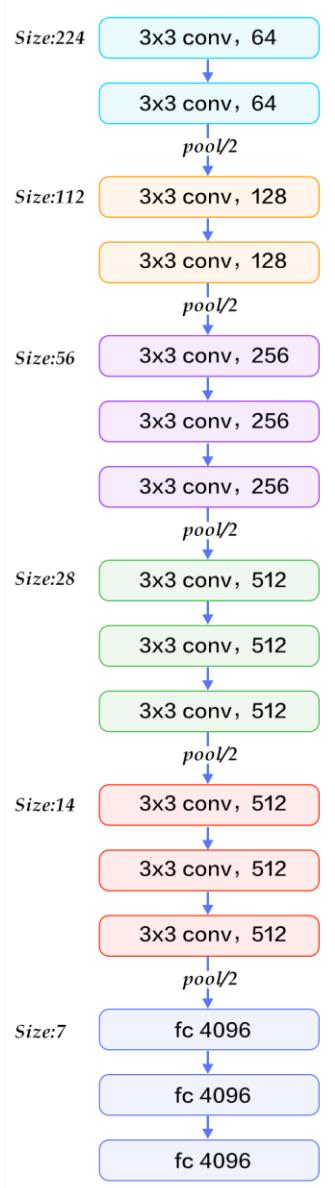
- Similar to AlexNet except:
 - Conv1 – change from (11x11 stride 4) to (7x7 stride 2)
 - Conv3, 4, 5: instead of 384, 384, 256 filters use 512, 1024, 512
 - Top 5 error : 16.4% → 11.7%

Recap - VGGNet (Simonyan and Zisserman, 2014)

- ILSVRC'14 2nd in classification, 1st in localization
- Small filters, Deeper networks
 - 8 layers(AlexNet) → 16~19 layers(VGG16, VGG19)
 - Only use 3x3 conv stride 1, pad 1 & 2x2 maxpool stride 2
 - 11.7% top 5 error(ZFNet) → 7.3% top 5 error
- Why use only 3x3 filters?
 - Stack of 3x3 conv layers has same effective receptive field as 7x7 conv layer
 - Deeper means more non-linearities
 - Fewer parameters: $2 \times (3 \times 3 \times C)$ vs $(5 \times 5 \times C)$
→ regularization effect

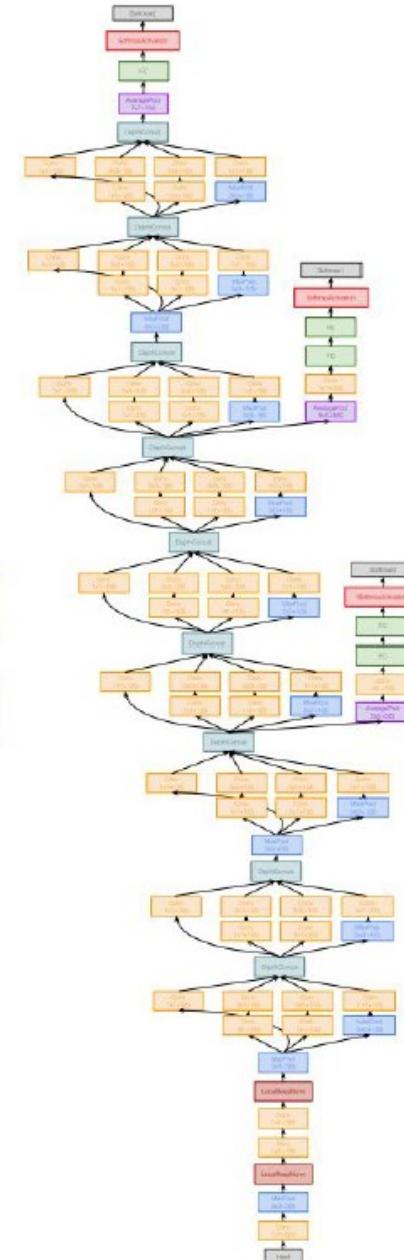
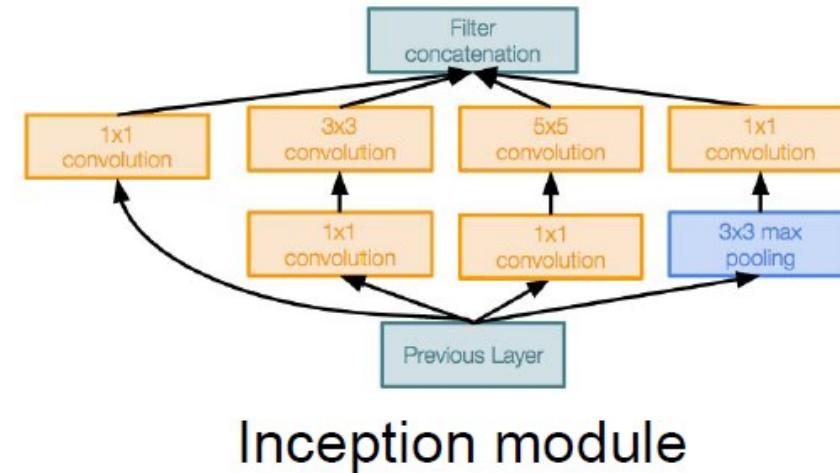


5 or



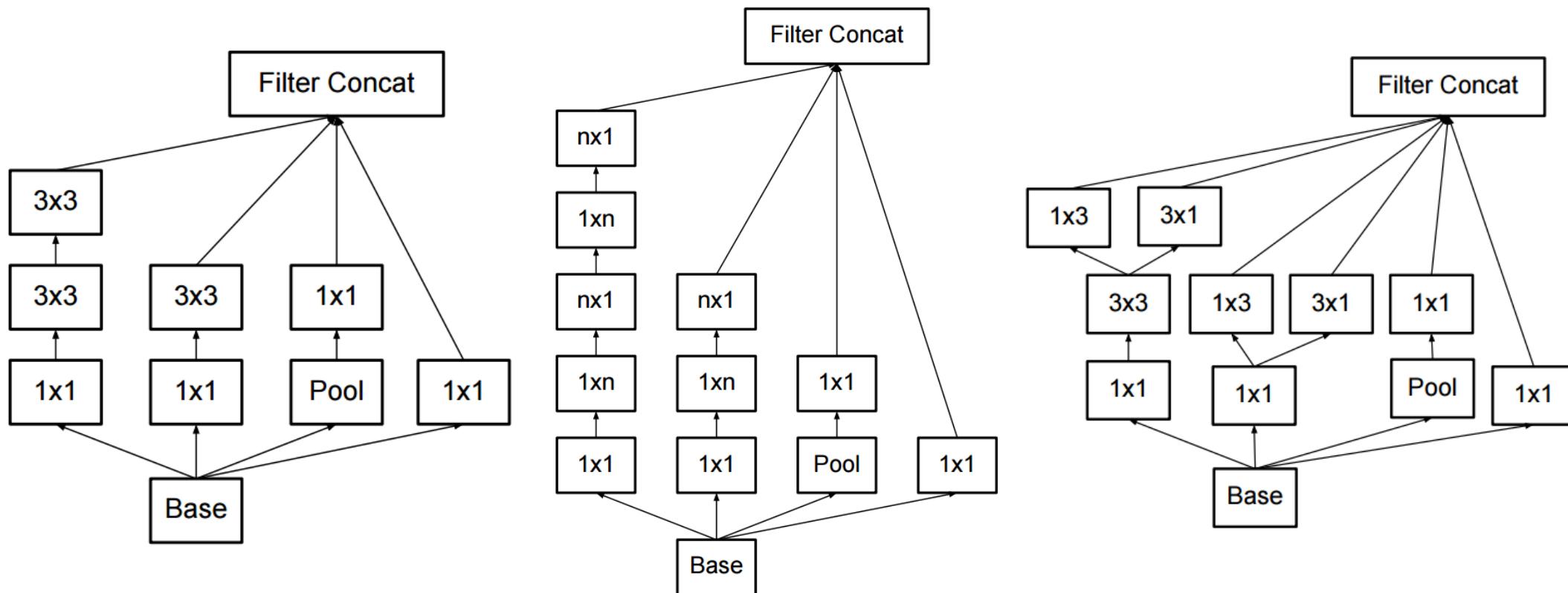
Recap - GoogLeNet (Szegedy, 2014)

- Deeper networks, with computational efficiency
 - 22 layers
 - Efficient “Inception” module
 - Global average pooling
 - Only 5 million parameters
 - ILSVRC’14 classification winner
➤ 6.7% top 5 error



Recap - Inception-v3

- Factorization of filters



Recap - ResNet

- Swept 1st place in all ILSVRC and COCO 2015 competitions

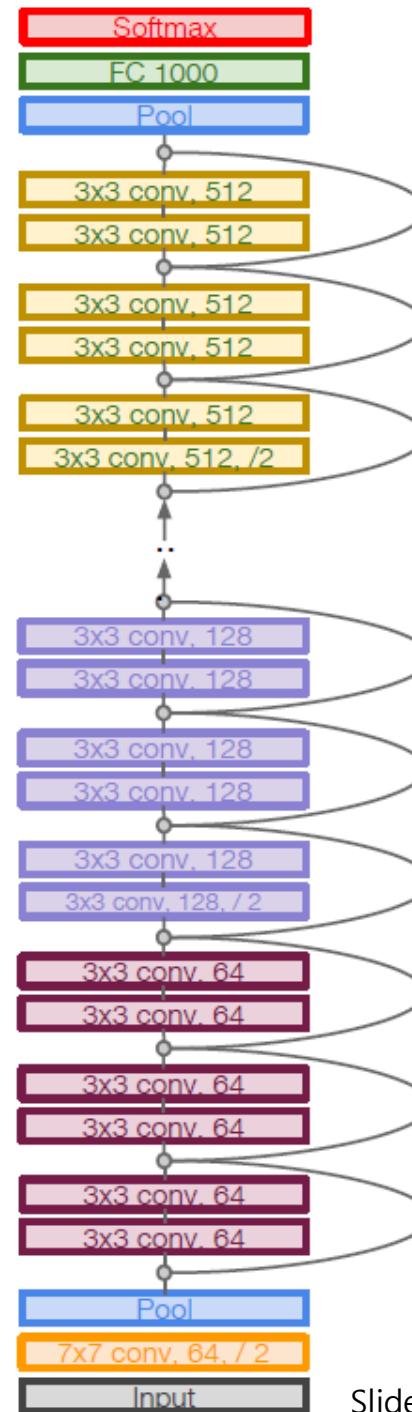
MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places in all five main tracks**
 - ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
 - ImageNet Detection: **16%** better than 2nd
 - ImageNet Localization: **27%** better than 2nd
 - COCO Detection: **11%** better than 2nd
 - COCO Segmentation: **12%** better than 2nd

- ILSVRC'15 classification winner(3.6% top 5 error) – better than “human performance” (Russakovsky 2014)

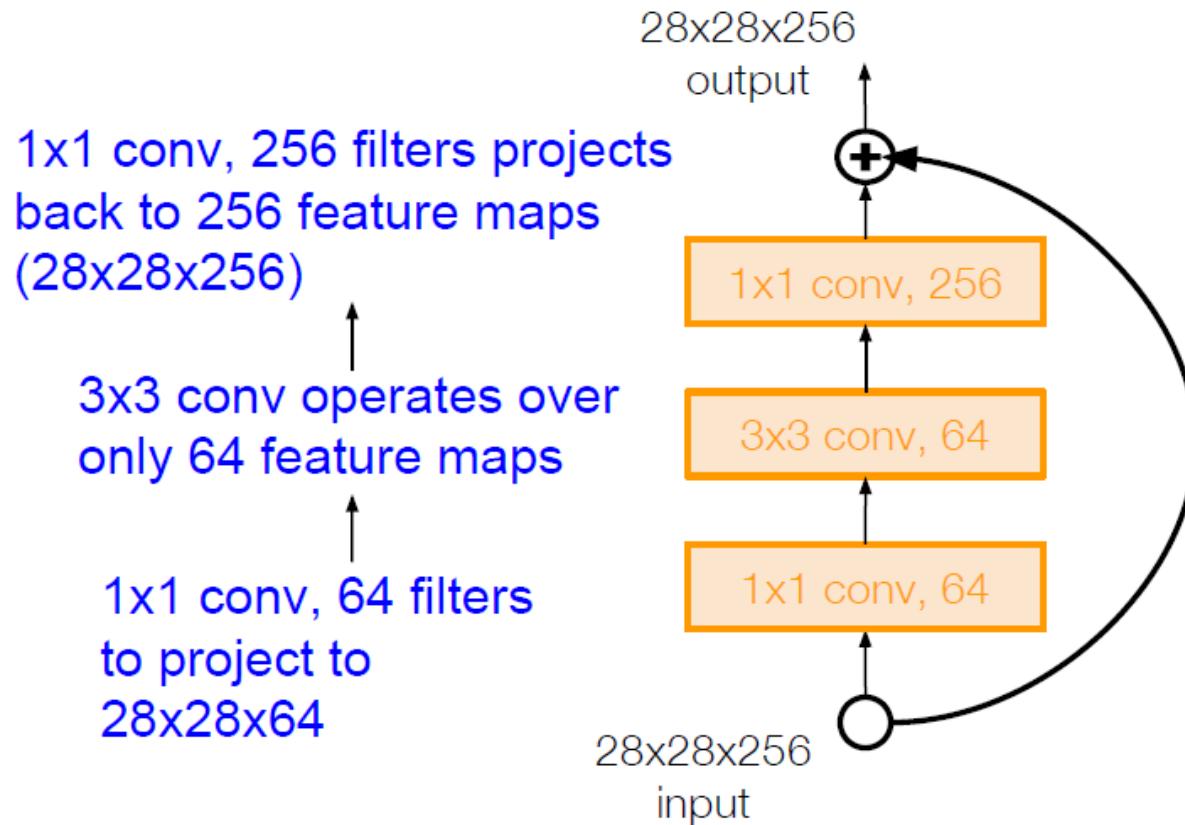
Recap - ResNet

- Full ResNet architecture
 - Stack residual blocks
 - Every residual block has two 3×3 conv layers
 - Periodically double # of filters and downsample spatially using stride 2
 - Additional conv layer at the beginning
 - No FC layers at the end(only FC 1000 to output classes)



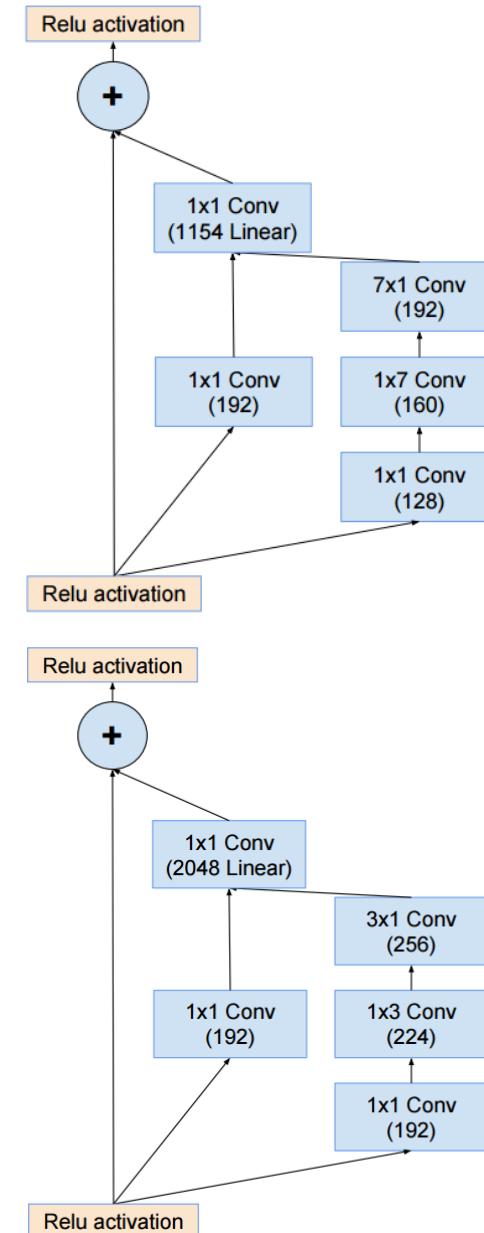
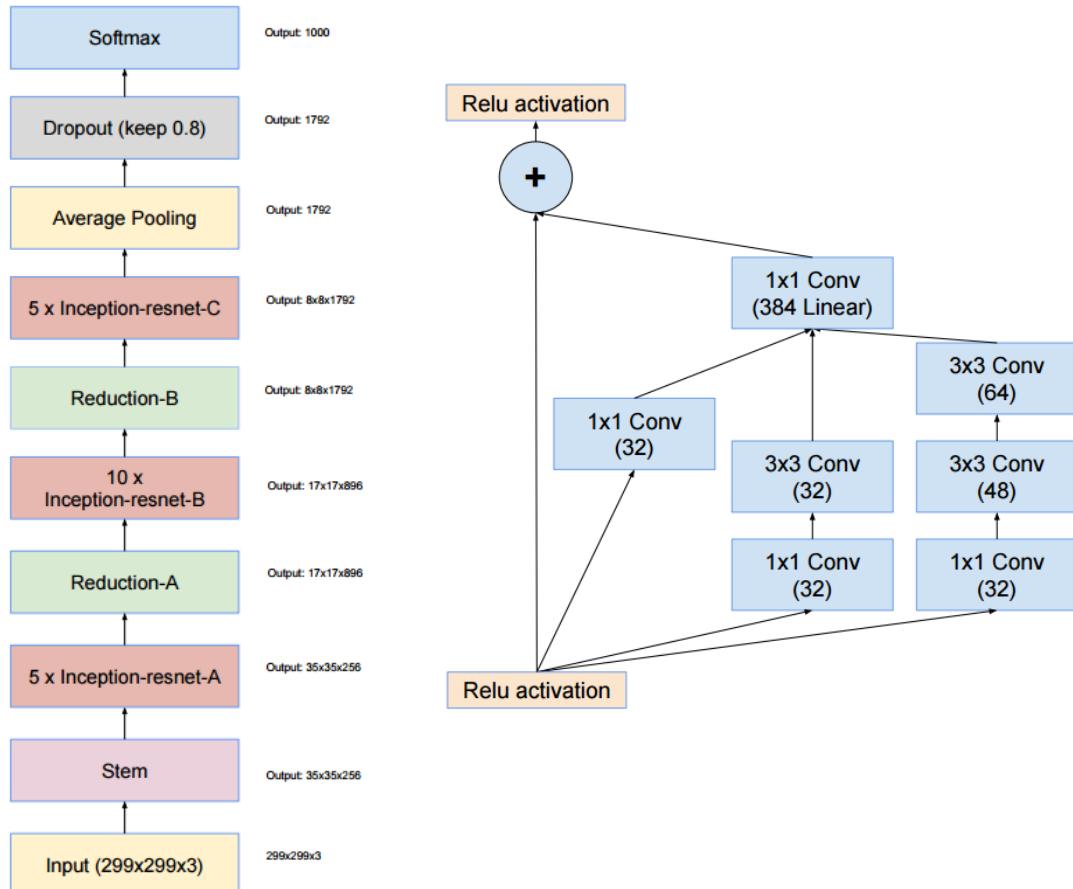
Recap - Bottleneck Architecture

- For deeper networks(ResNet-50+), use “bottleneck” layer to improve efficiency(similar to GoogLeNet)

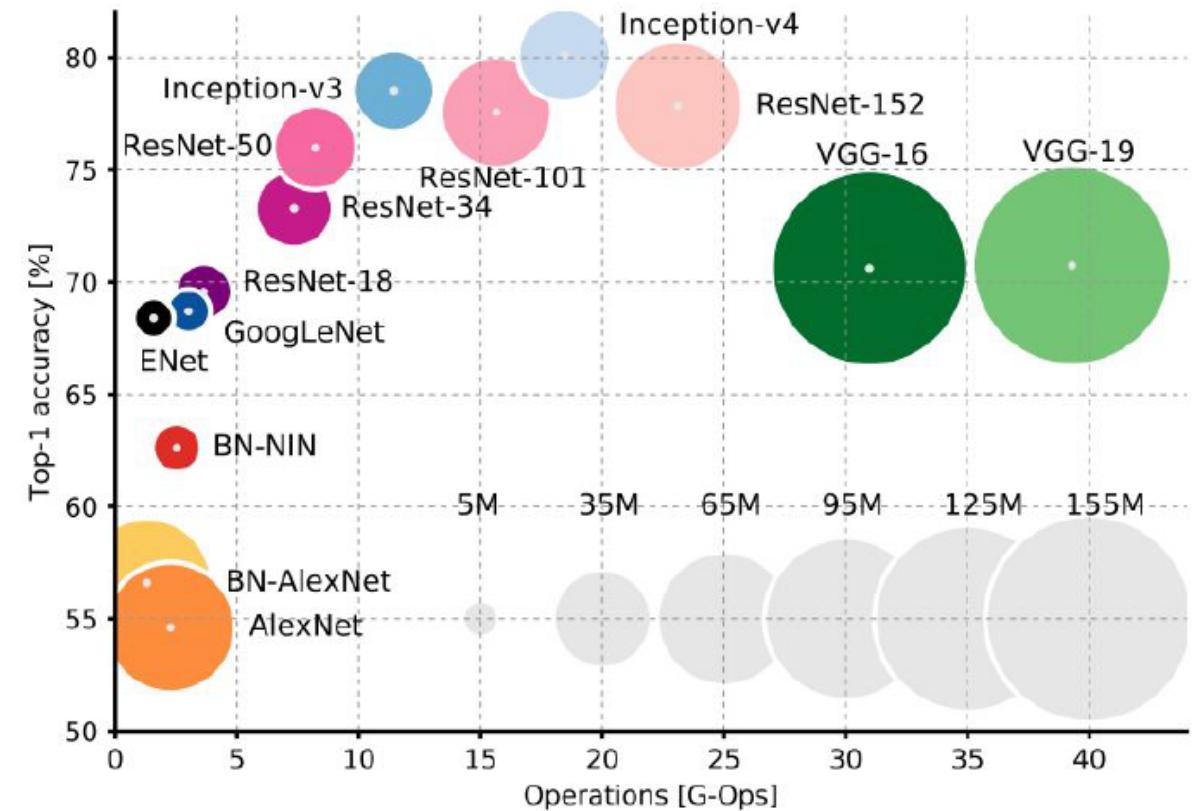
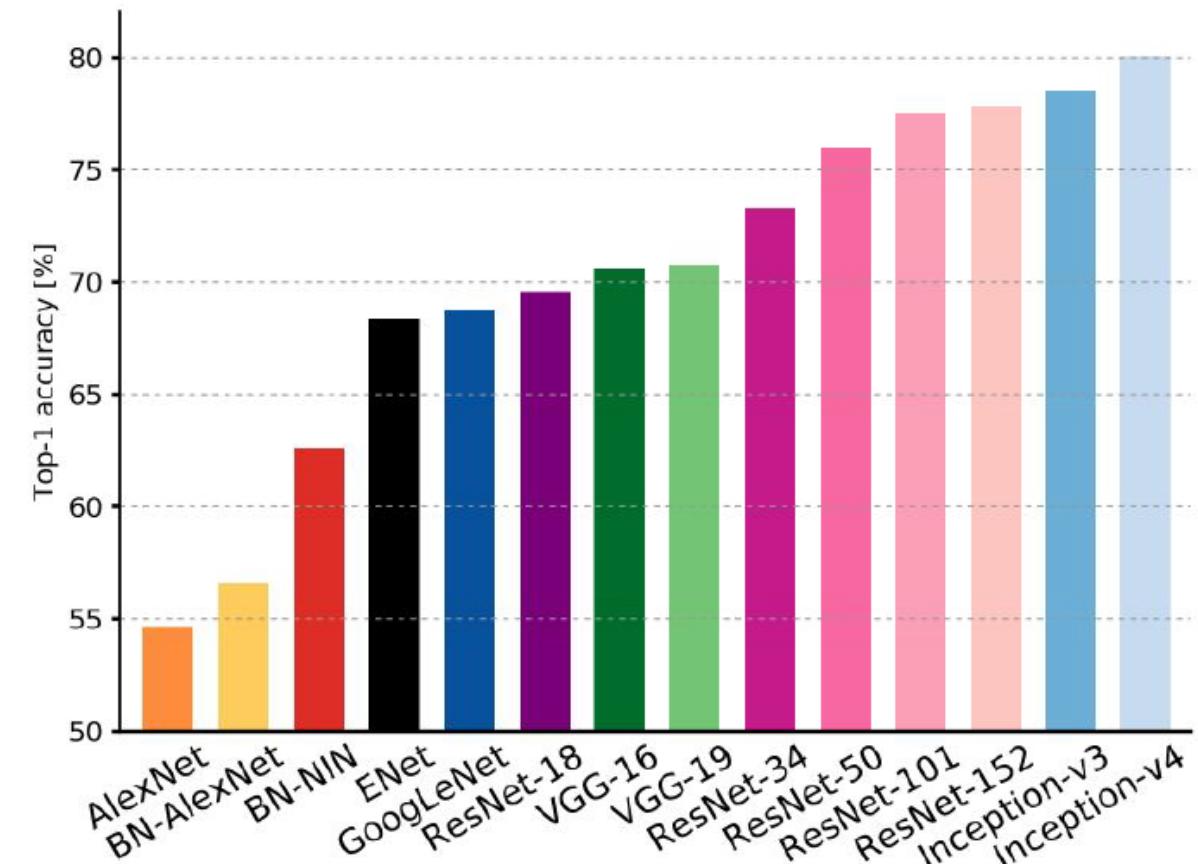


Recap - Inception-ResNet

- Inception + ResNet



Recap - Comparing Complexity



An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Key Requirements for Commercial Computer Vision Usage

- Data-centers(Clouds)
 - Rarely safety-critical
 - Low power is nice to have
 - Real-time is preferable
- Gadgets – Smartphones, Self-driving cars, Drones, etc.
 - Usually safety-critical(except smartphones)
 - Low power is must-have
 - Real-time is required

What's the “Right” Neural Network for Use in a Gadget?

- Desirable Properties
 - Sufficiently high accuracy
 - Low computational complexity
 - Low energy usage
 - Small model size

Why Small Deep Neural Networks?

- Small DNNs train faster on distributed hardware
- Small DNNs are more deployable on embedded processors
- Small DNNs are easily updatable Over-The-Air(OTA)

Techniques for Small Deep Neural Networks

- Remove Fully-Connected Layers
- Kernel Reduction
- Channel Reduction
- Evenly Spaced Downsampling
- Depthwise Separable Convolutions
- Shuffle Operations
- Distillation & Compression

List of Papers

- “**Densely Connected Convolutional Networks**” – CVPR 2017 best paper
- “**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”
- “Aggregated Residual Transformations for Deep Neural Networks” - **ResNeXt**
- “Learning Transferable Architectures for Scalable Image Recognition” - **NASNet**

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-

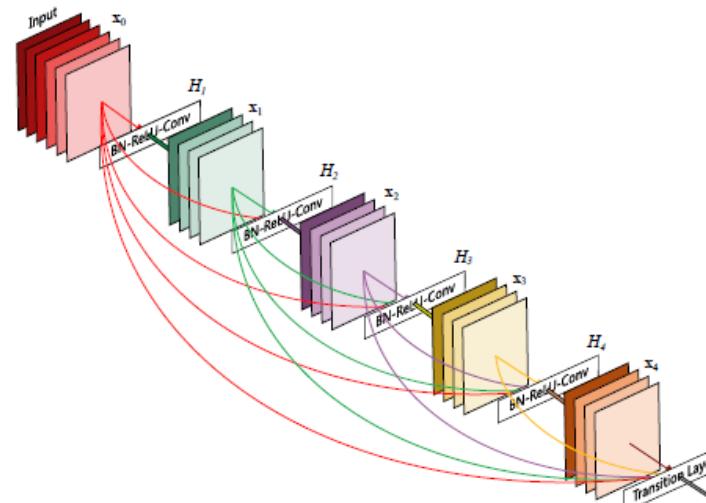
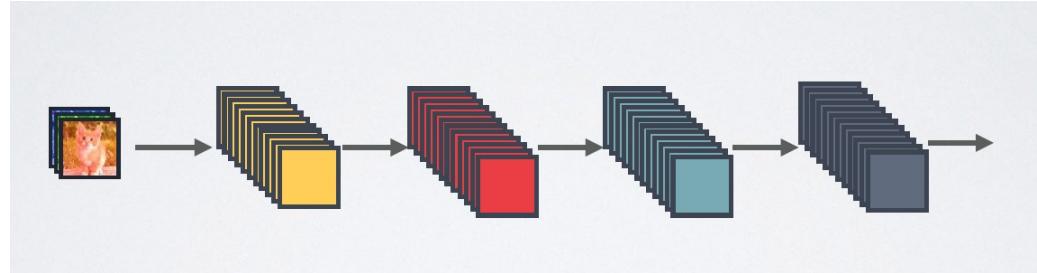


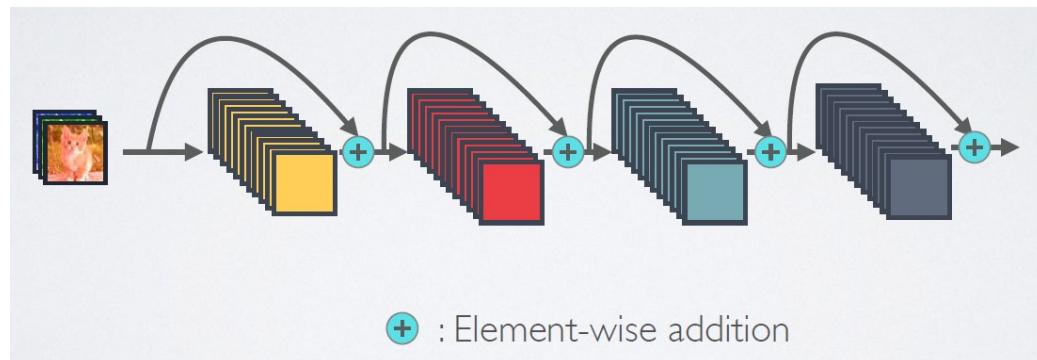
Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Dense Connectivity

- Standard Connectivity

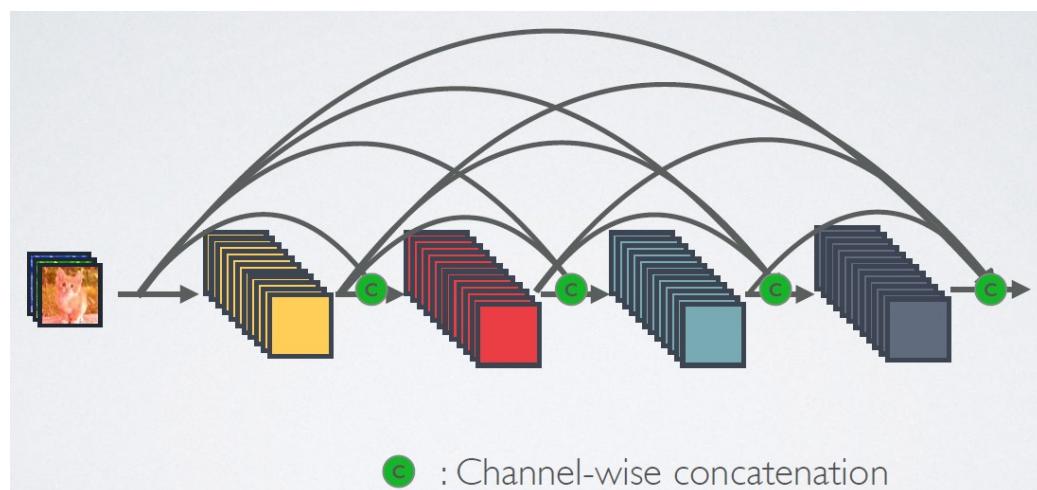


- ResNet Connectivity



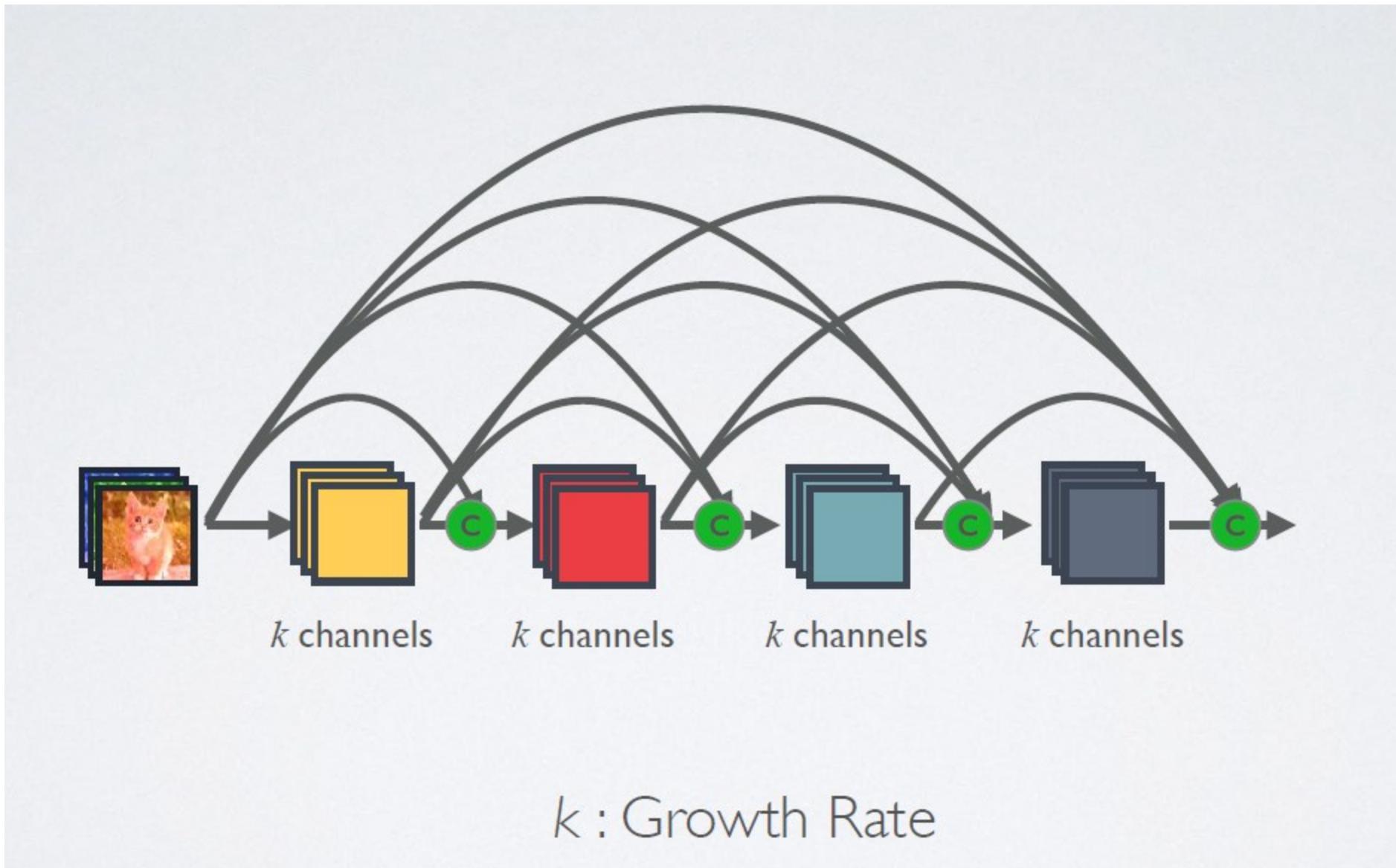
+ : Element-wise addition

- Dense Connectivity

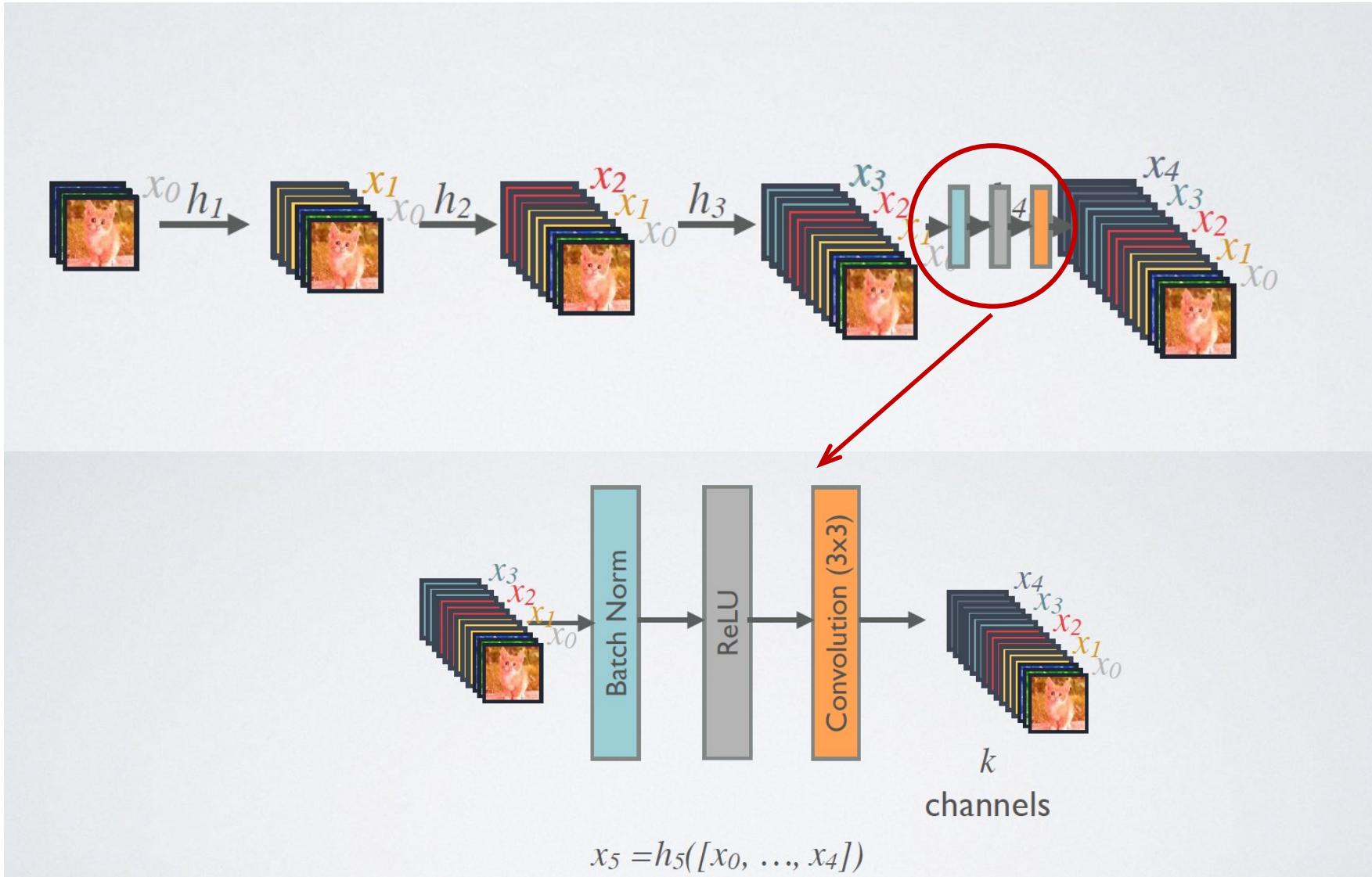


C : Channel-wise concatenation

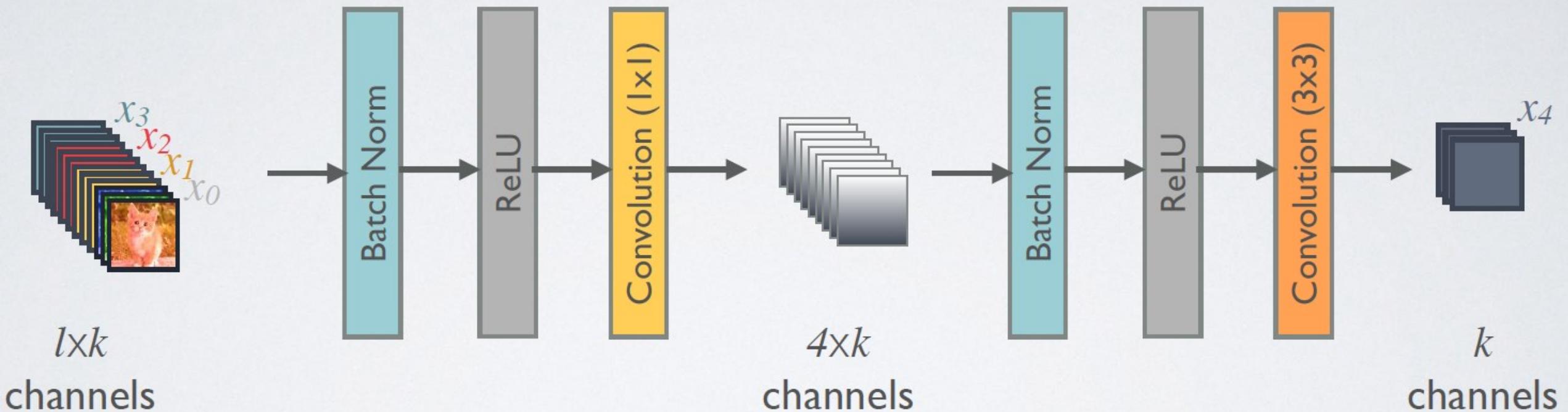
Dense and Slim



Forward Propagation

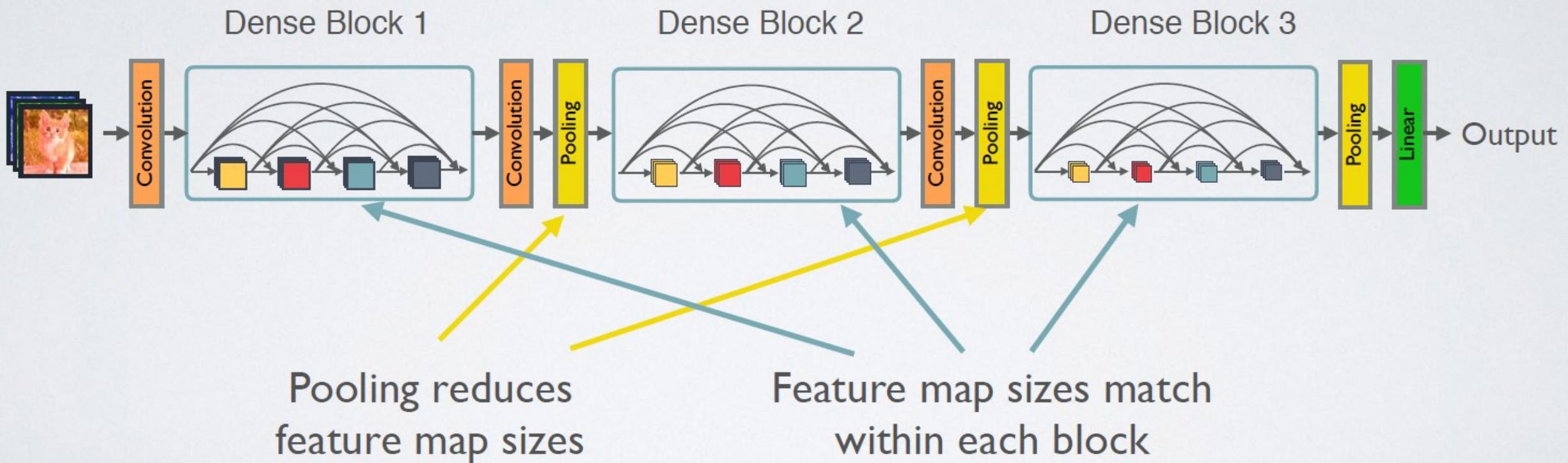


Composite Layer in DenseNet with Bottleneck Layer

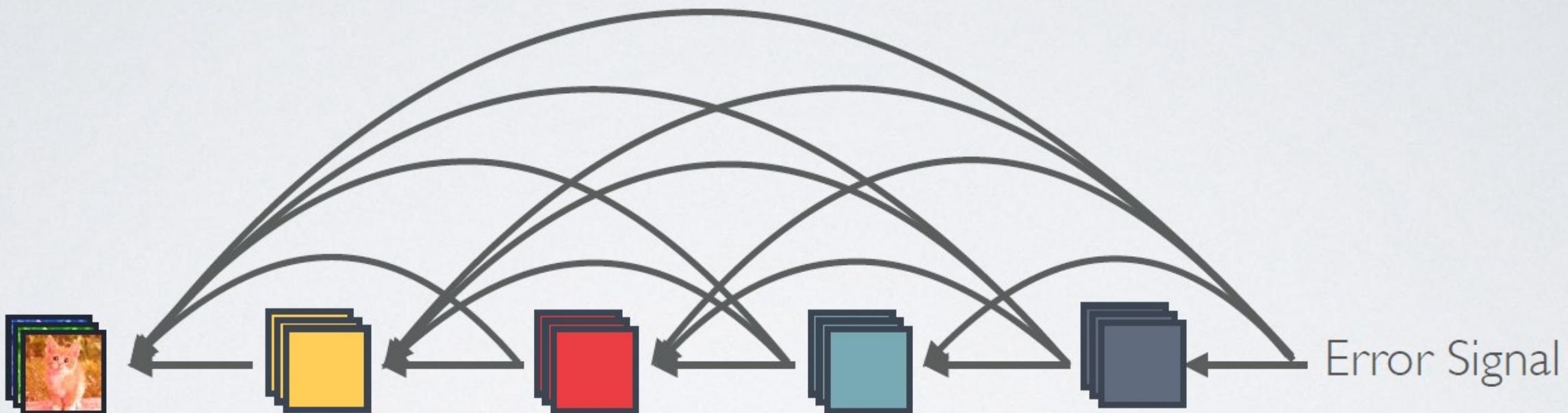


Higher parameter and computational efficiency

DenseNet



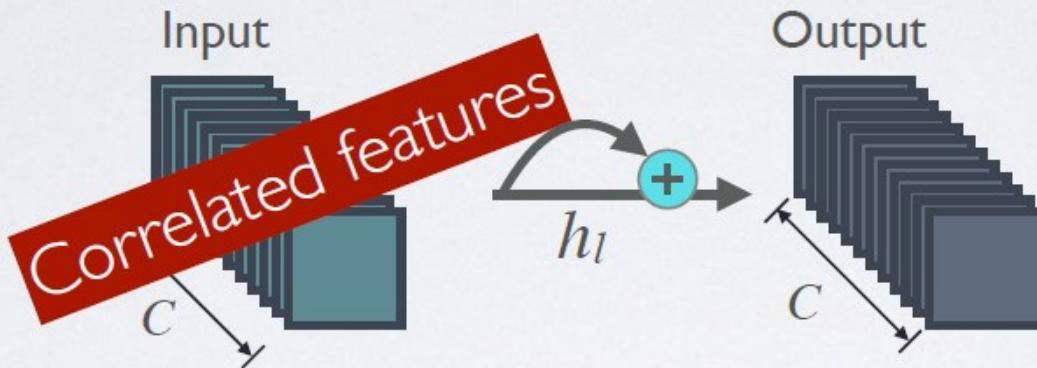
Advantage 1 : Strong Gradient Flow



Implicit “deep supervision”

Advantage 2 : Parameter & Computational Efficiency

ResNet connectivity:

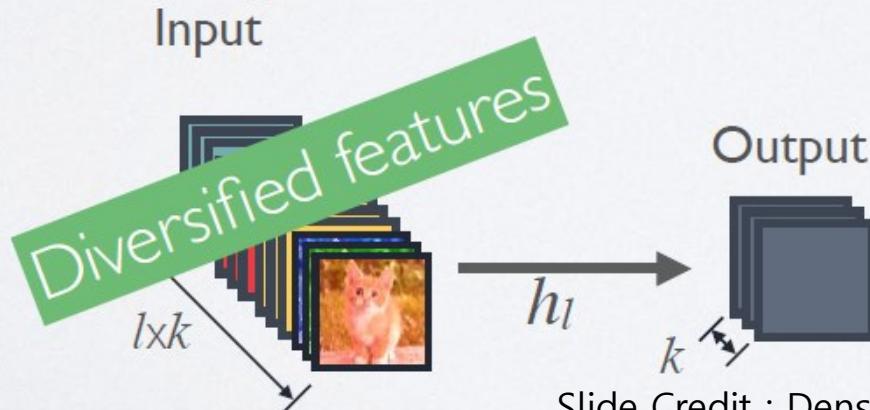


#parameters:

$$O(C \times C)$$

$k \ll C$

DenseNet connectivity:



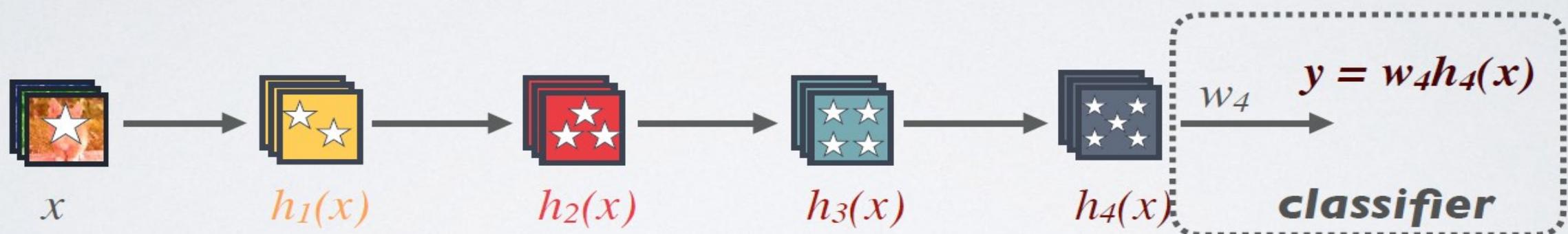
$$O(l \times k \times k)$$

k : Growth rate

Advantage 3 : Maintains Low Complexity Features

Standard Connectivity:

Classifier uses most complex (high level) features

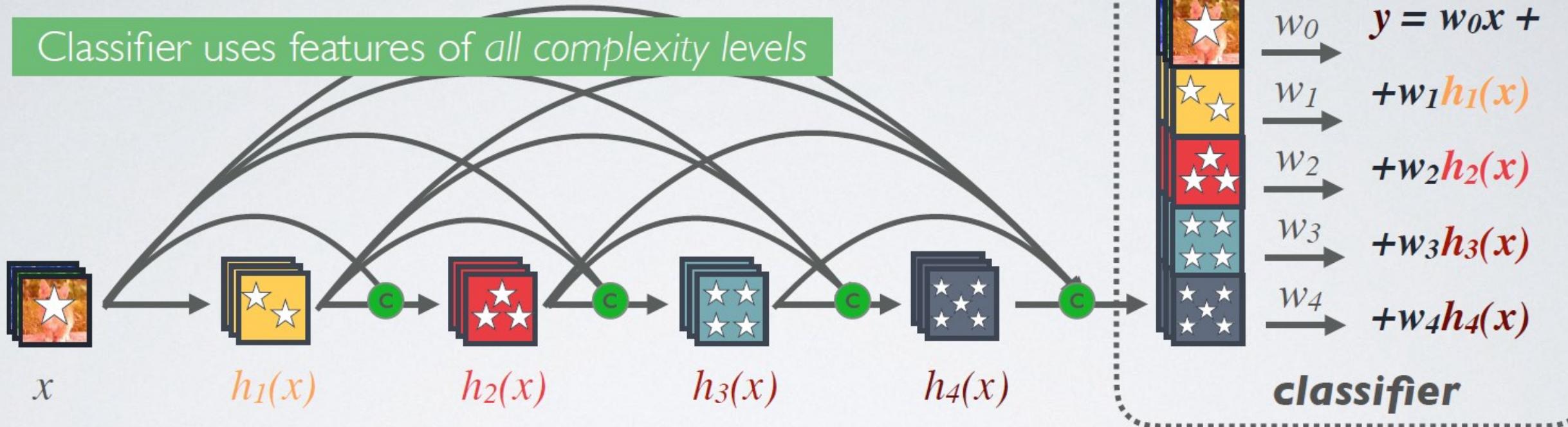


Increasingly complex features



Advantage 3 : Maintains Low Complexity Features

Dense Connectivity:

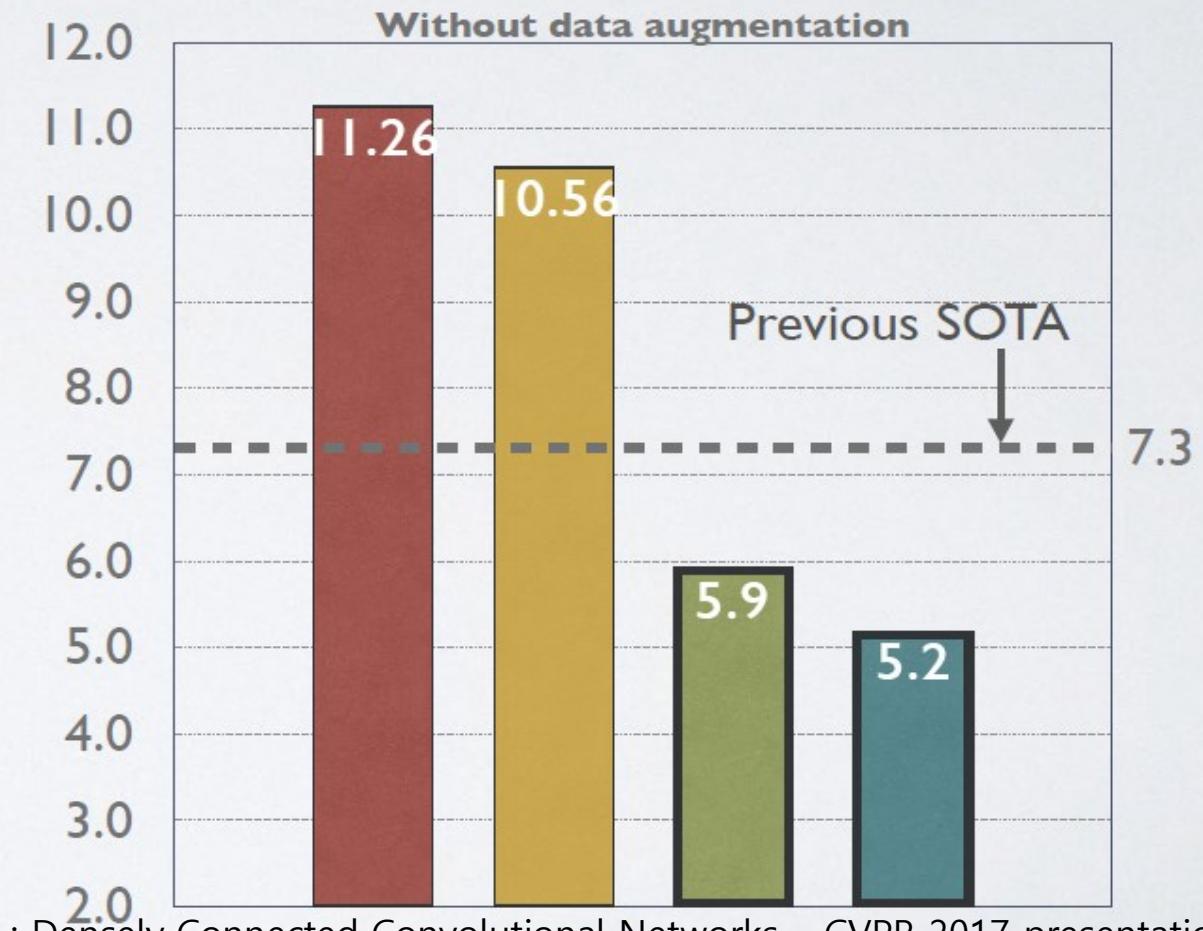
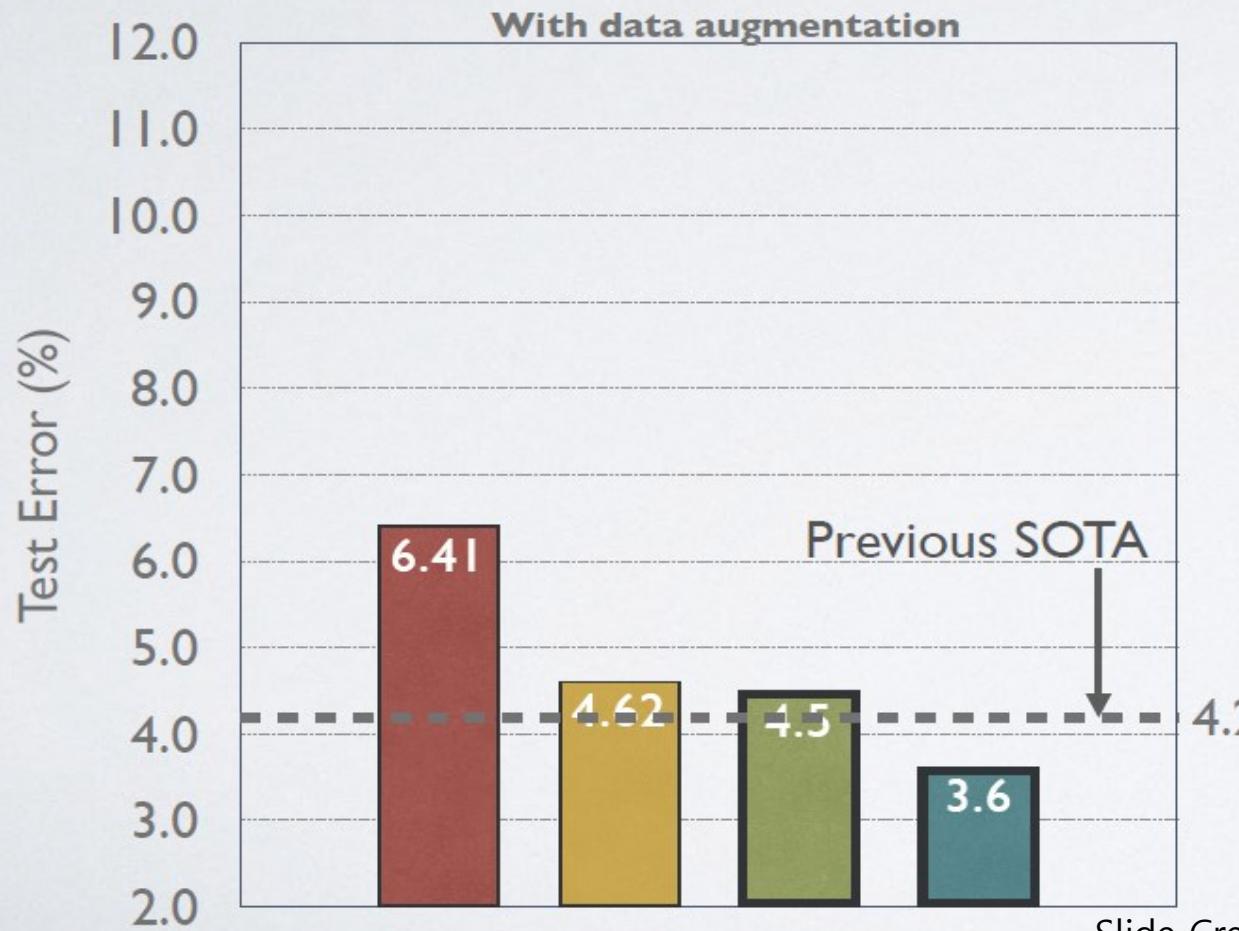


Increasingly complex features



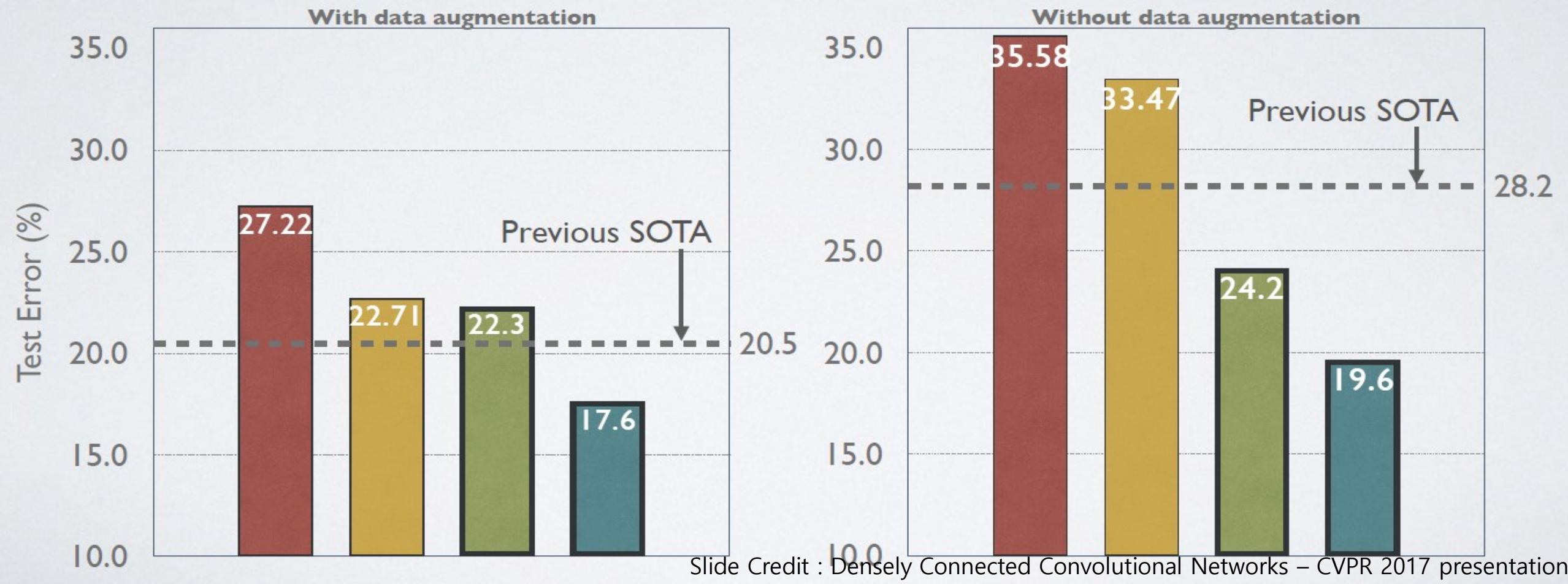
Results on CIFAR-10

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

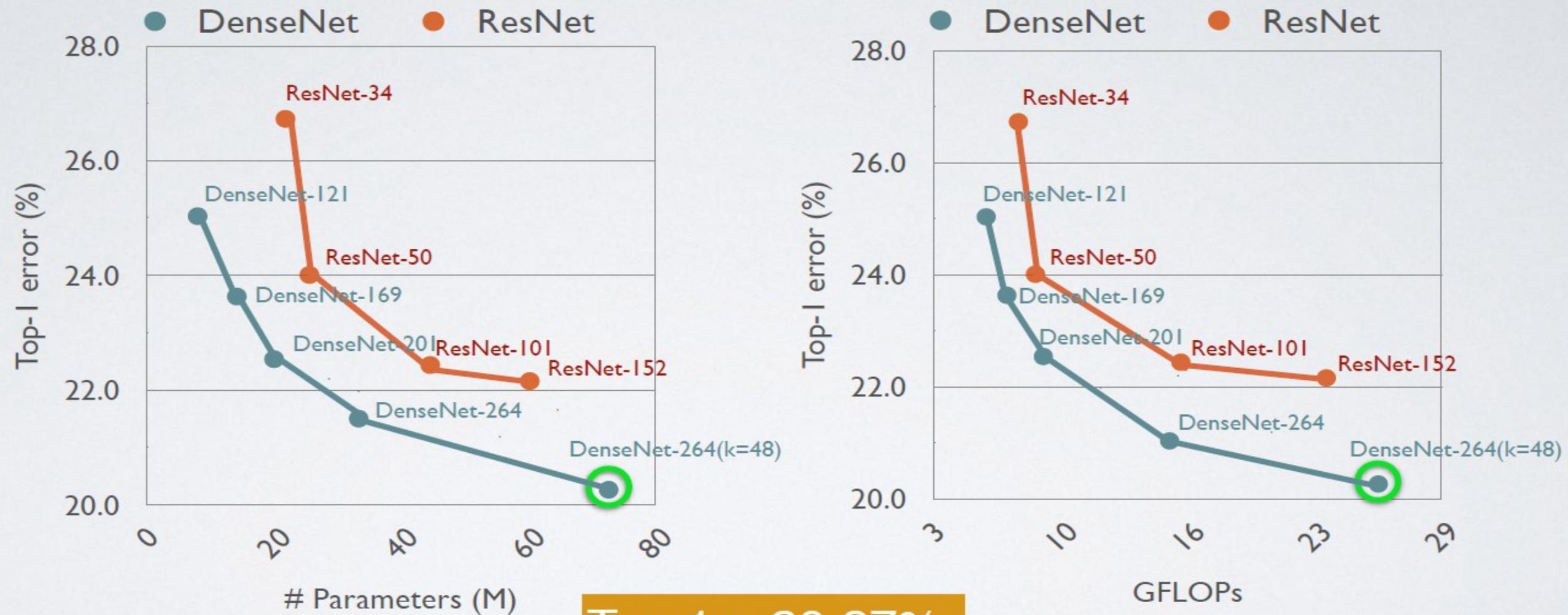


Results on CIFAR-100

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%

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

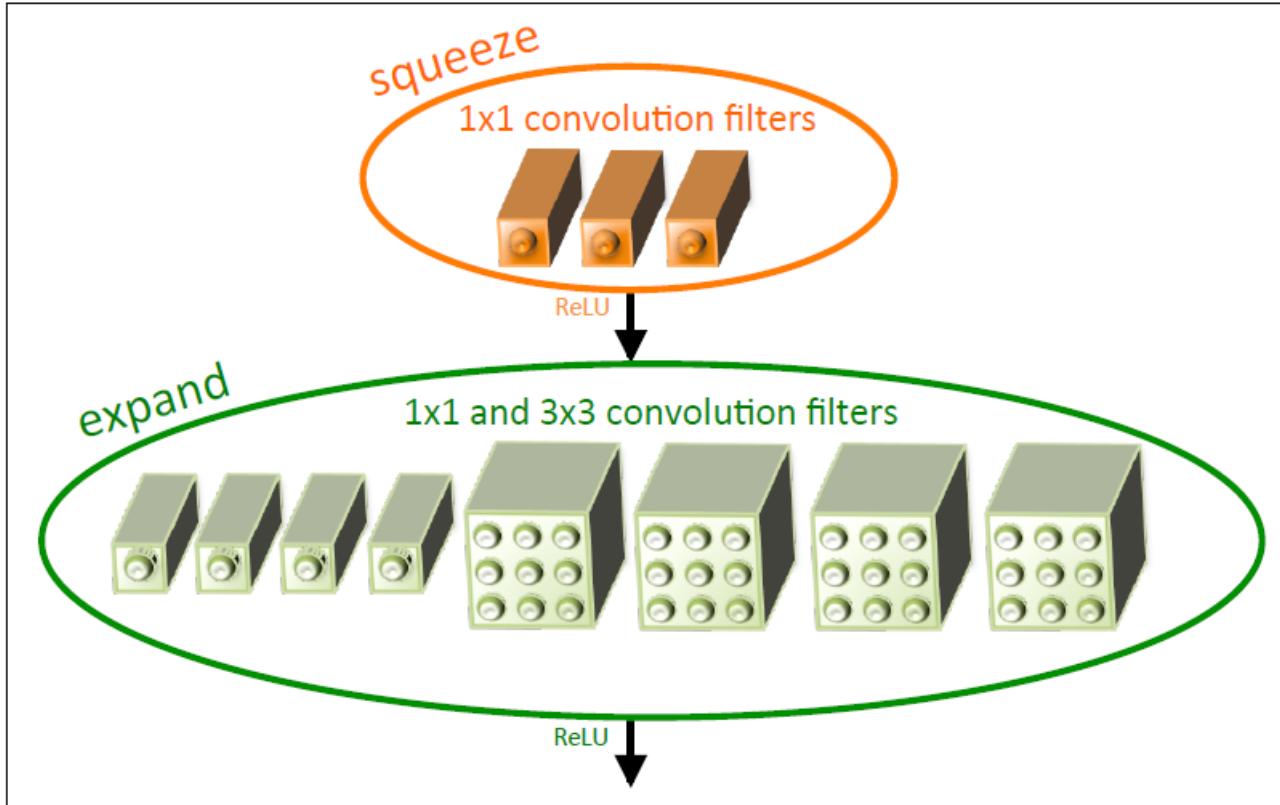
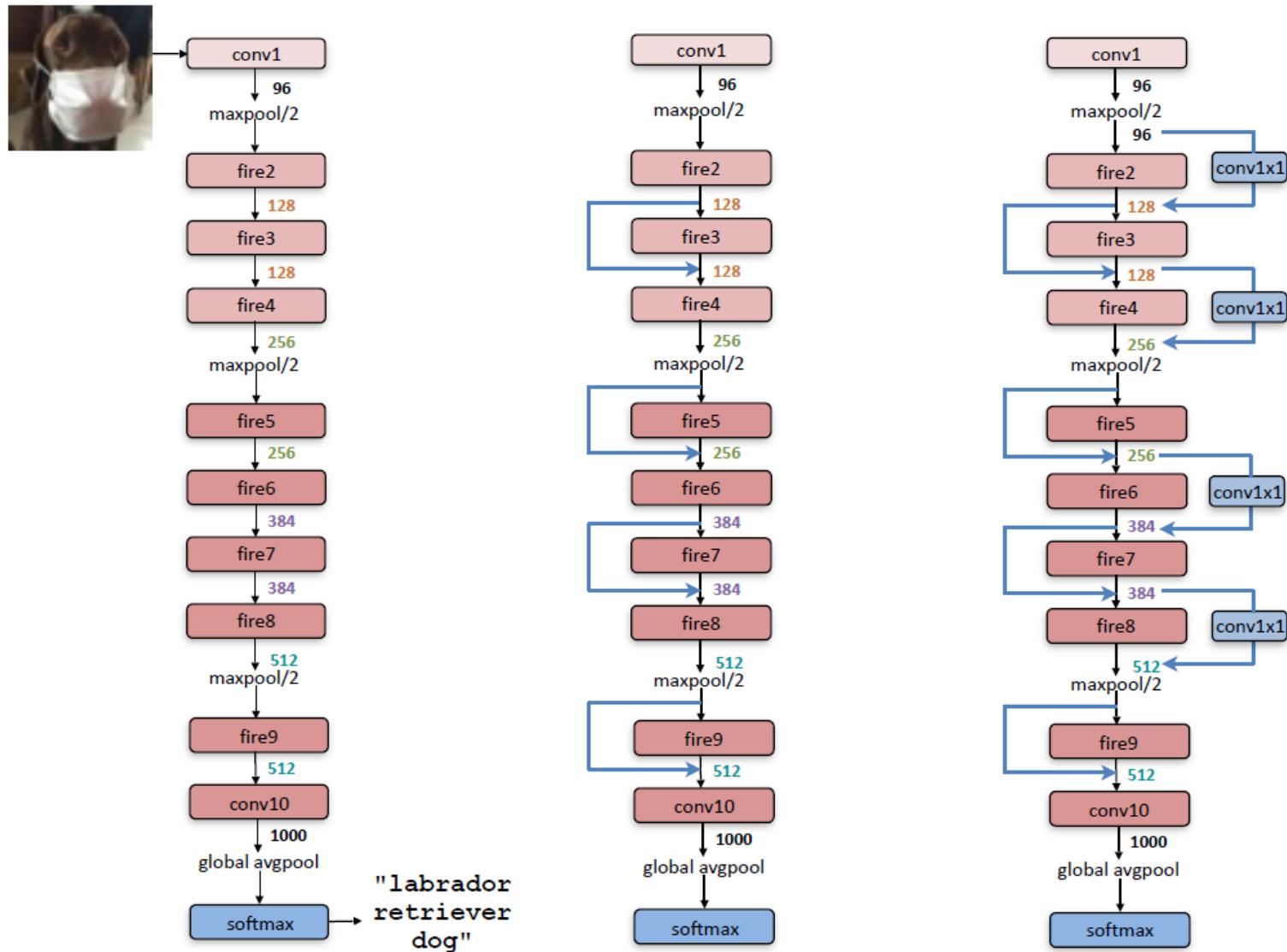


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

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)

Other SqueezeNet Details

- 1-pixel border of zero-padding in the input data to 3x3 filters
- ReLU is applied
- Dropout ratio of 50% after the fire9 module
- Lack of fully-connected layers
- Initial learning rate : 0.04
- Implemented on Caffe

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

Xception

.02357v3 [cs.CV] 4 Apr 2017

Xception: Deep Learning with Depthwise Separable Convolutions

François Chollet

Google, Inc.

fchollet@google.com

Abstract

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

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

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

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

Xception

- Problems
 - Bigger model typically means a larger number of parameters
→ Overfitting
 - Increased use of computational resources
 - e.g. quadratic increase of computation
 $3 \times 3 \times C \rightarrow 3 \times 3 \times C : C^2$ computations

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).

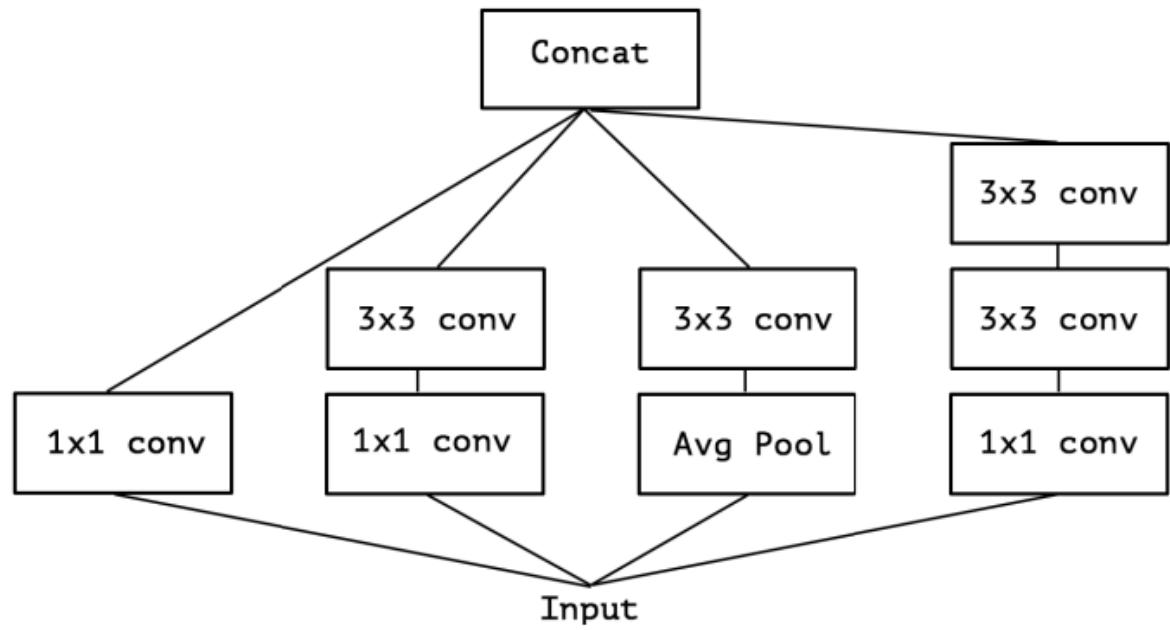
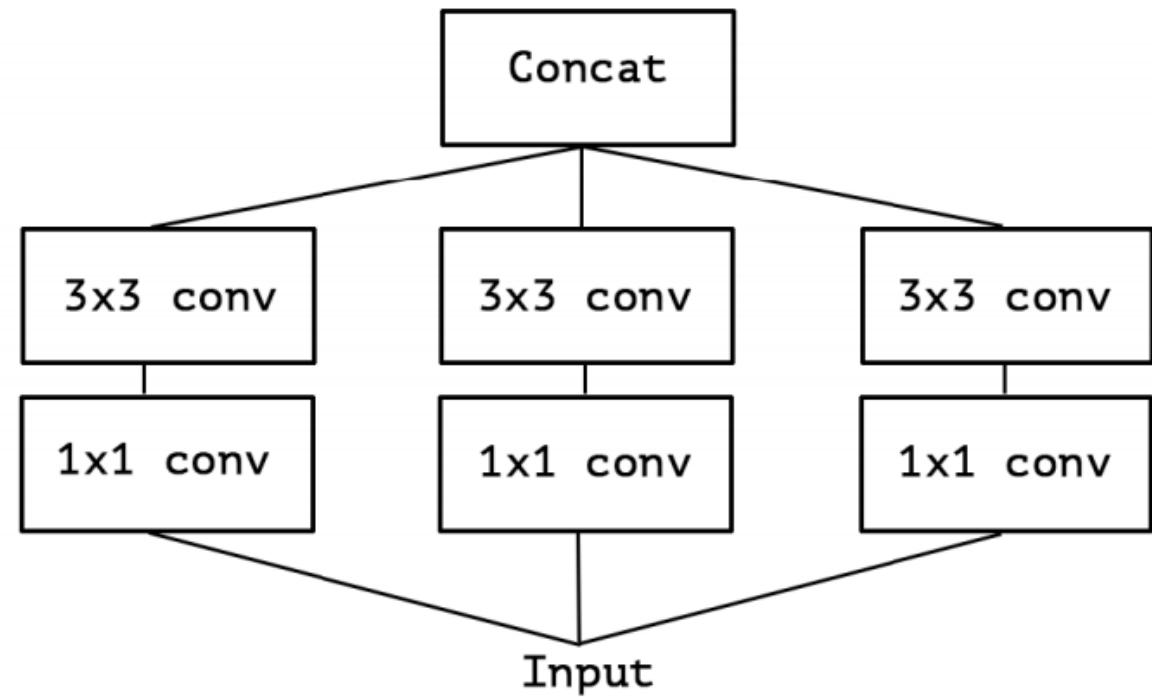


Figure 2. A simplified Inception module.



Equivalent Reformulation

Figure 2. A simplified Inception module.

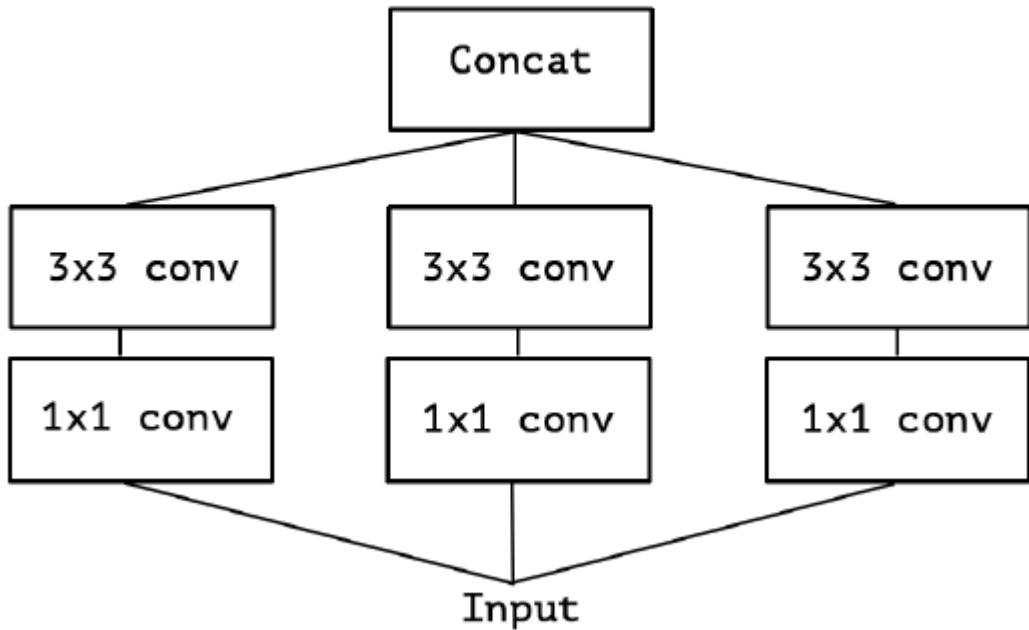
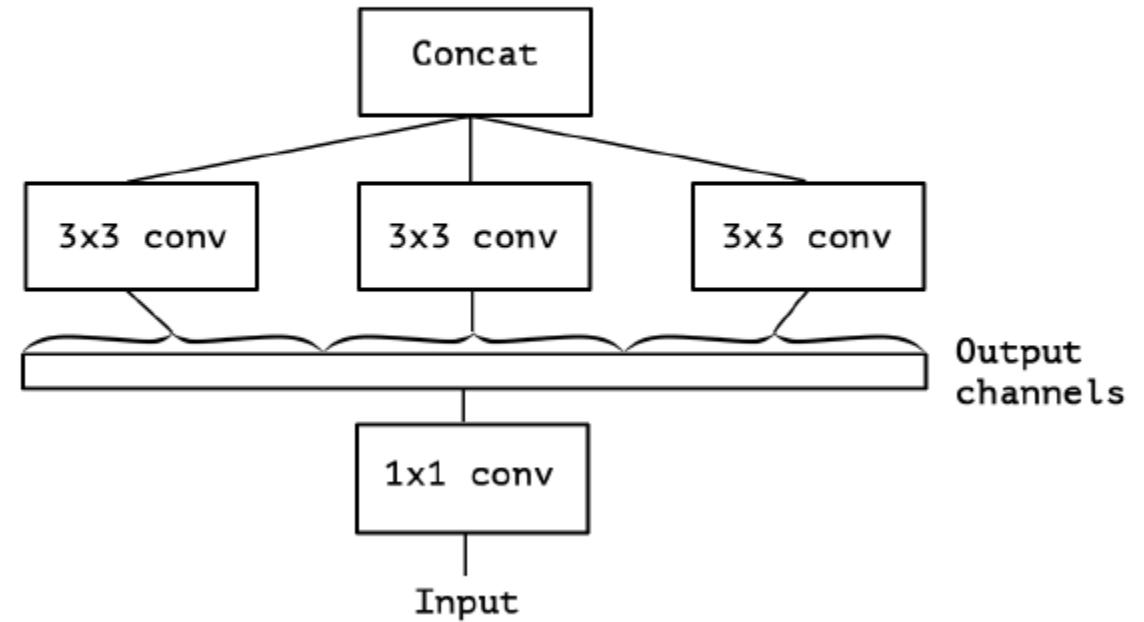
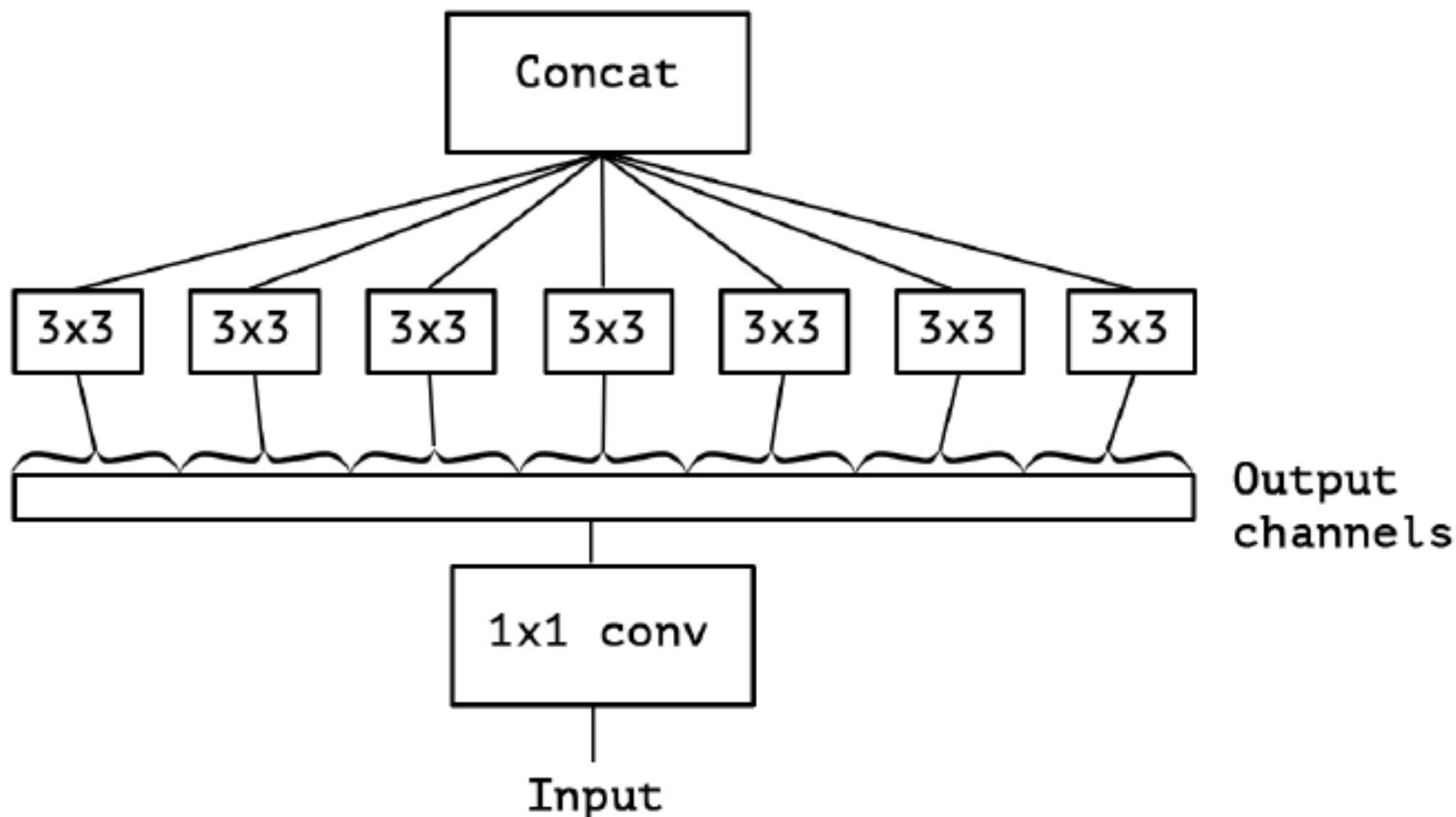


Figure 3. A strictly equivalent reformulation of the simplified Inception module.



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.



Depthwise Separable Convolution

Commonly called “separable convolution” in deep learning

frameworks such as TF and Keras; a spatial convolution performed

independently over each channel of an input followed by a pointwise

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

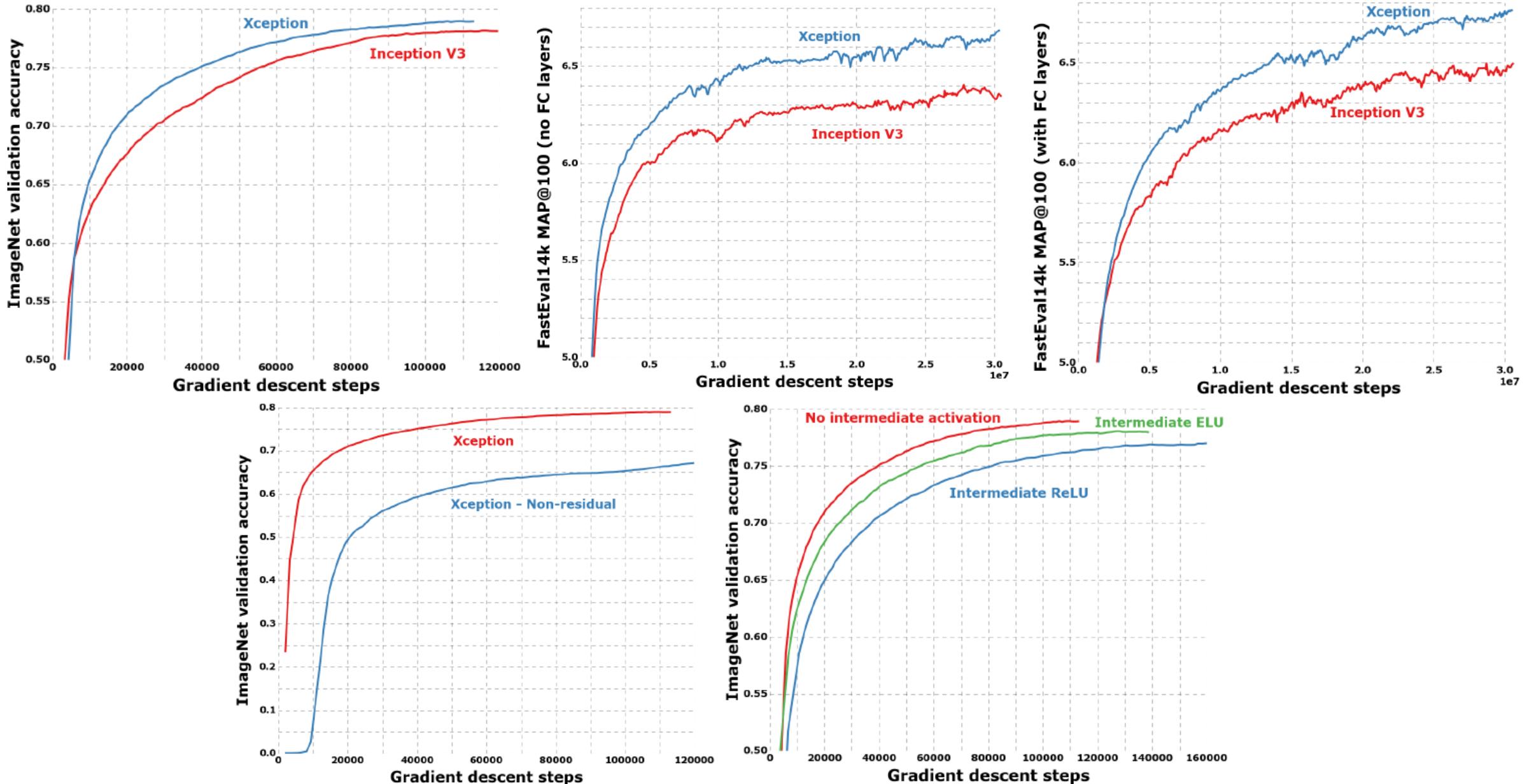
ImageNet

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

Results



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

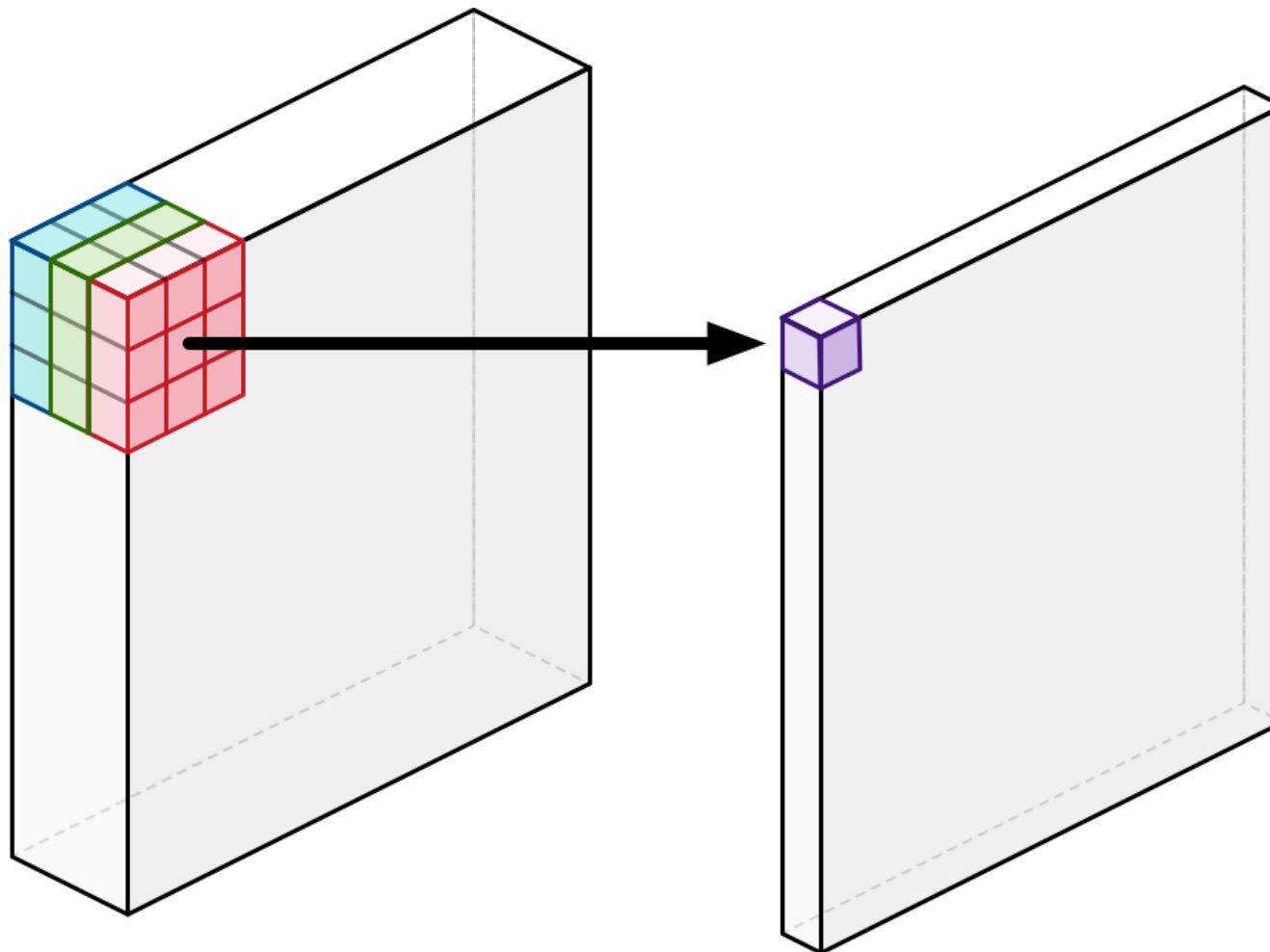
There has been rising interest in building small and efficient neural networks in the recent literature, e.g. [16, 34, 12, 36, 22]. Many different approaches can be generally classified into three main categories: 1) pruning

MobileNets



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

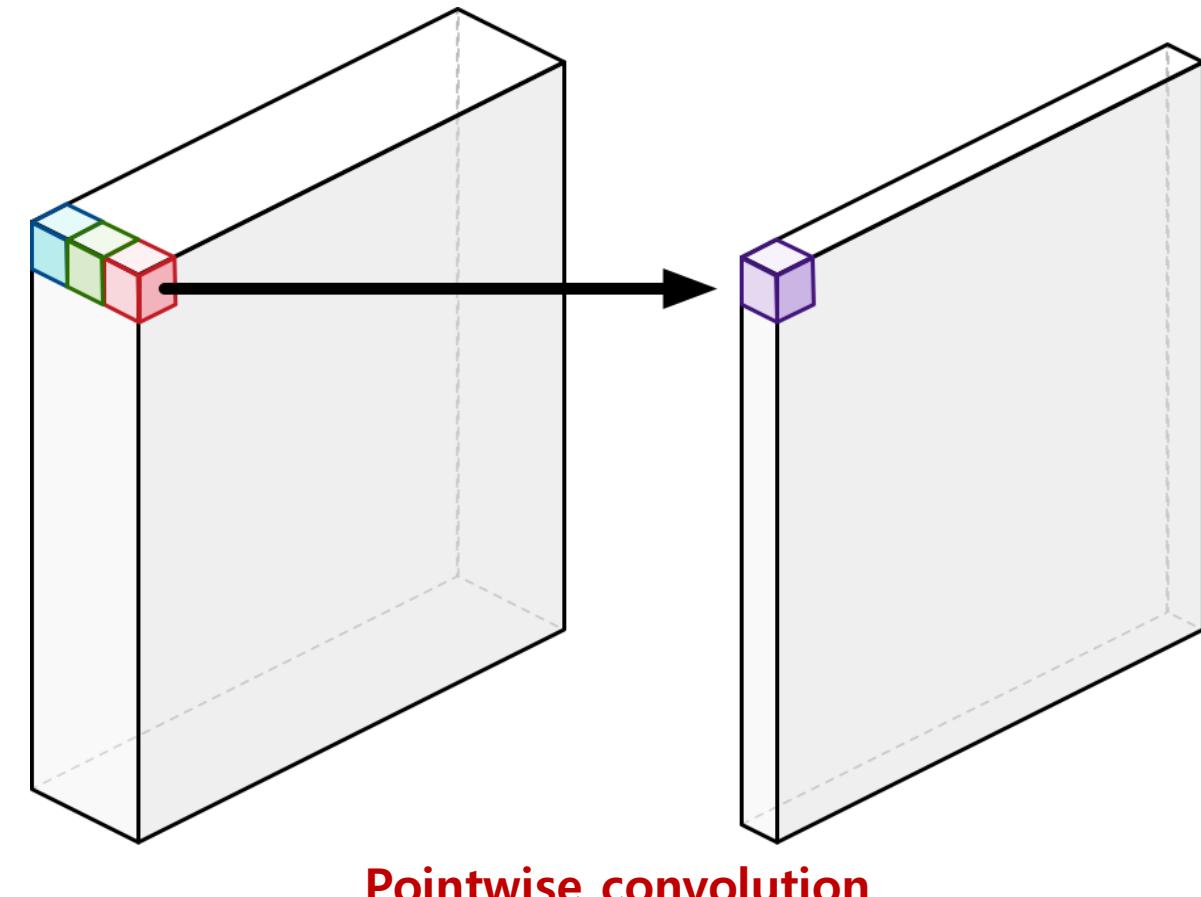
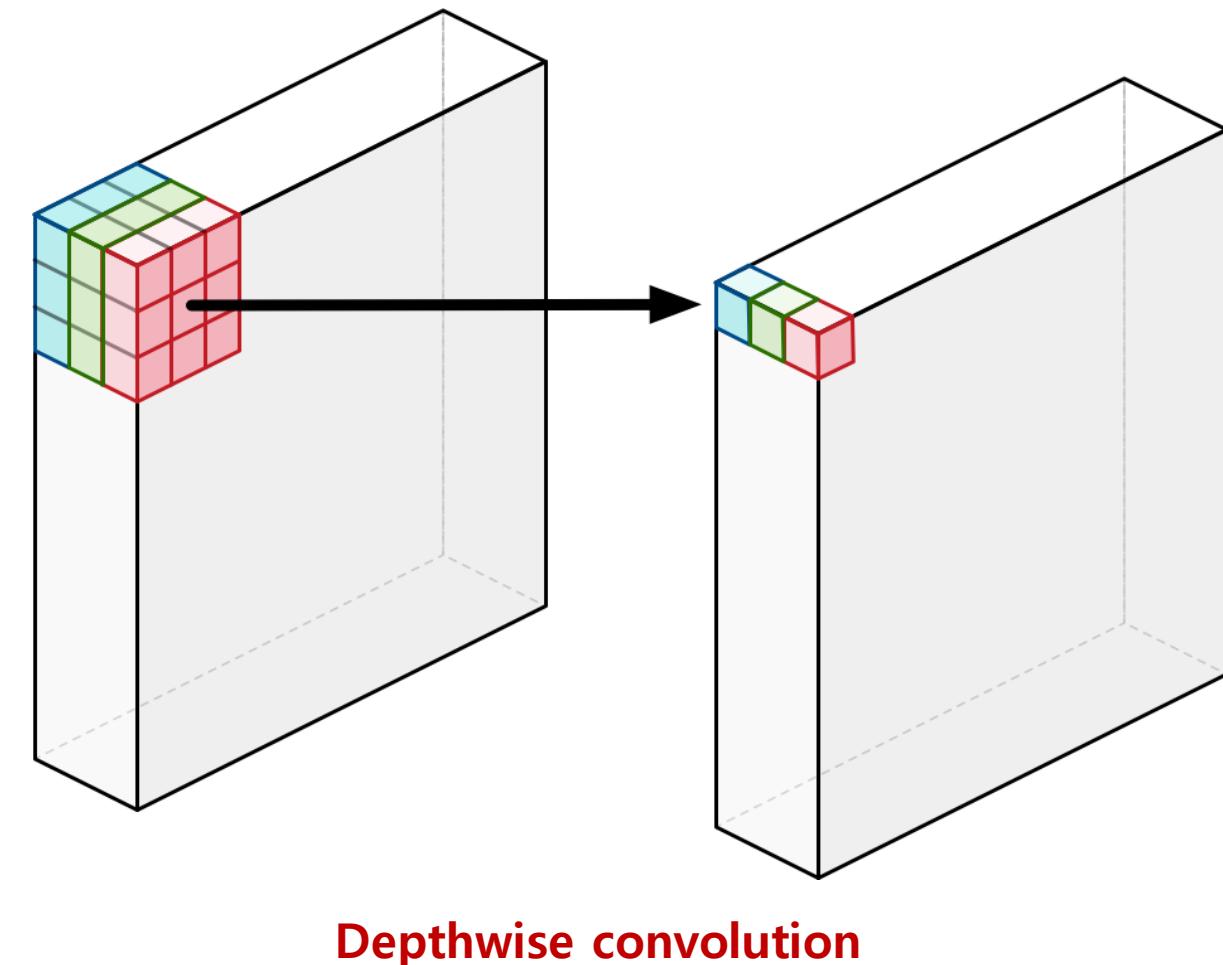
Standard Convolution



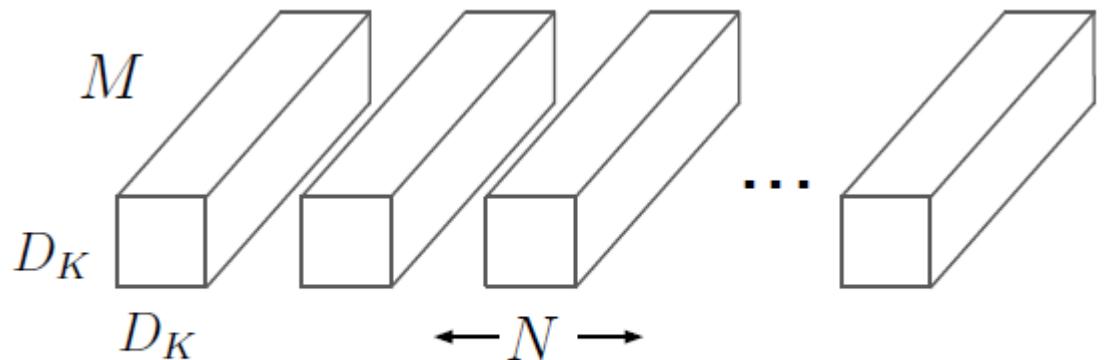
Depthwise convolution

Depthwise Separable Convolution

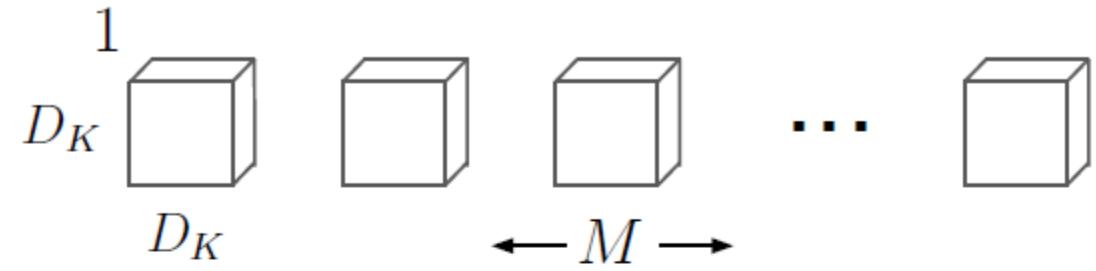
- Depthwise Convolution + Pointwise Convolution(1×1 convolution)



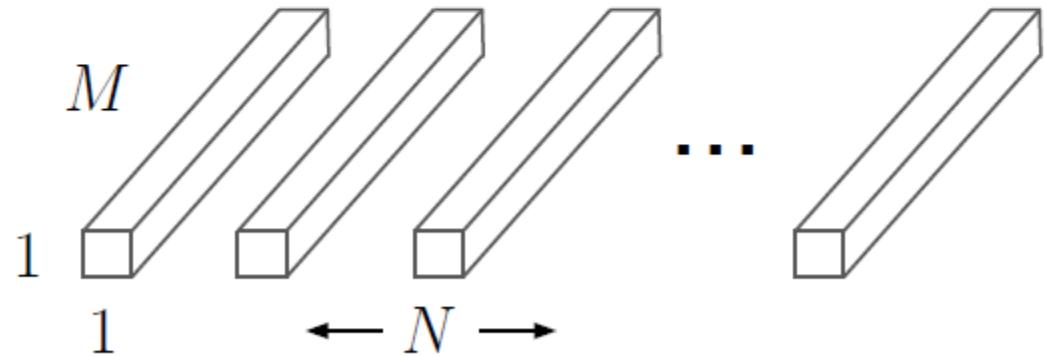
Standard Convolution vs Depthwise Separable Convolution



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



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

Standard Convolution vs Depthwise Separable Convolution

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

Depthwise Separable Convolutions

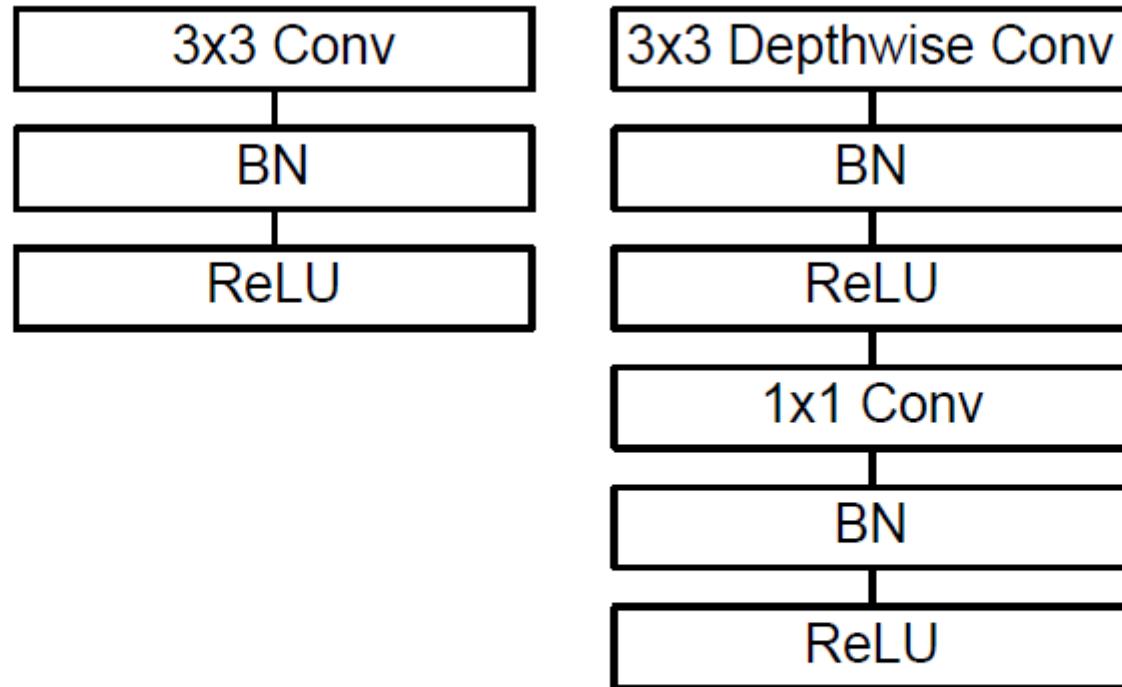


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Model Structure

Table 1. MobileNet Body Architecture

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

Table 2. Resource Per Layer Type

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

Width Multiplier & Resolution Multiplier

- For a given layer and width multiplier α , the number of input channels M becomes αM and the number of output channels N becomes αN – where α with typical settings of 1, 0.75, 0.6 and 0.25
- The second hyper-parameter to reduce the computational cost of a neural network is a resolution multiplier ρ
- Computational cost:
$$D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$$

Width Multiplier & Resolution Multiplier

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

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

Experiments – Model Choices

Table 4. Depthwise Separable vs Full Convolution MobileNet

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

Table 5. Narrow vs Shallow MobileNet

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

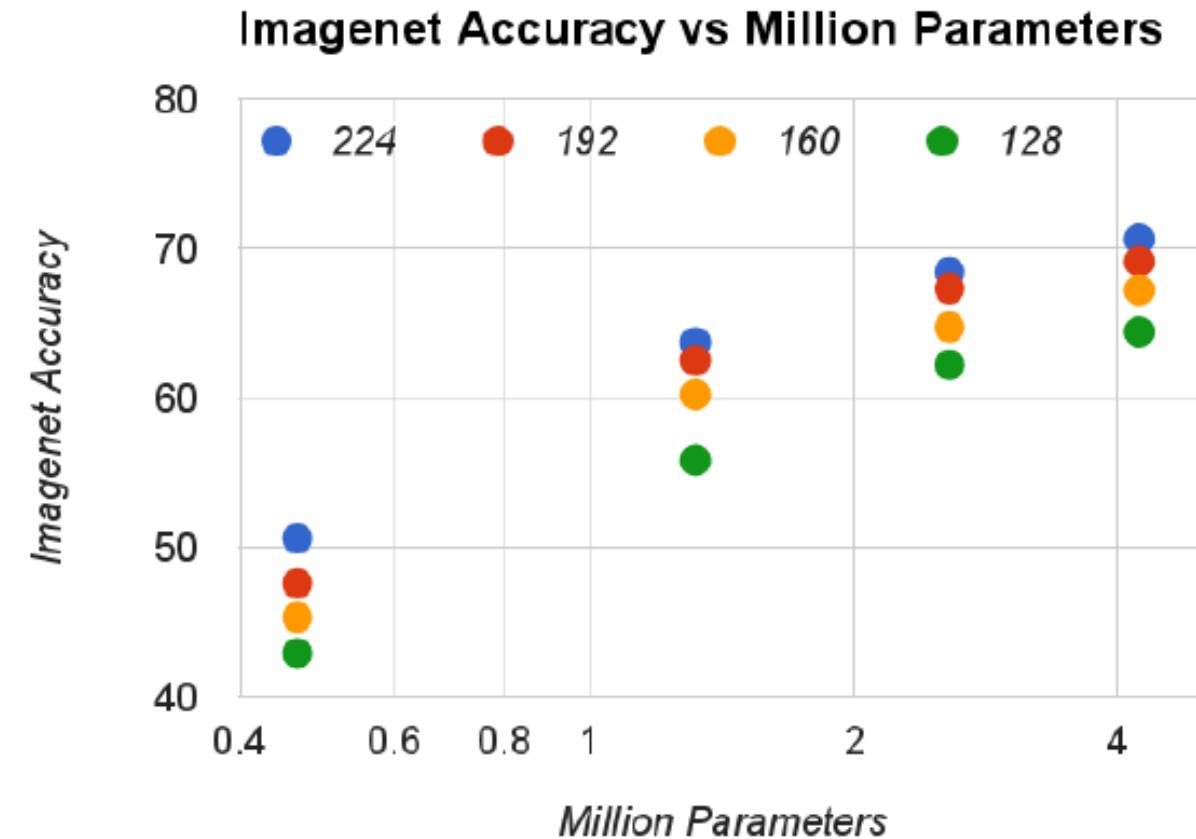
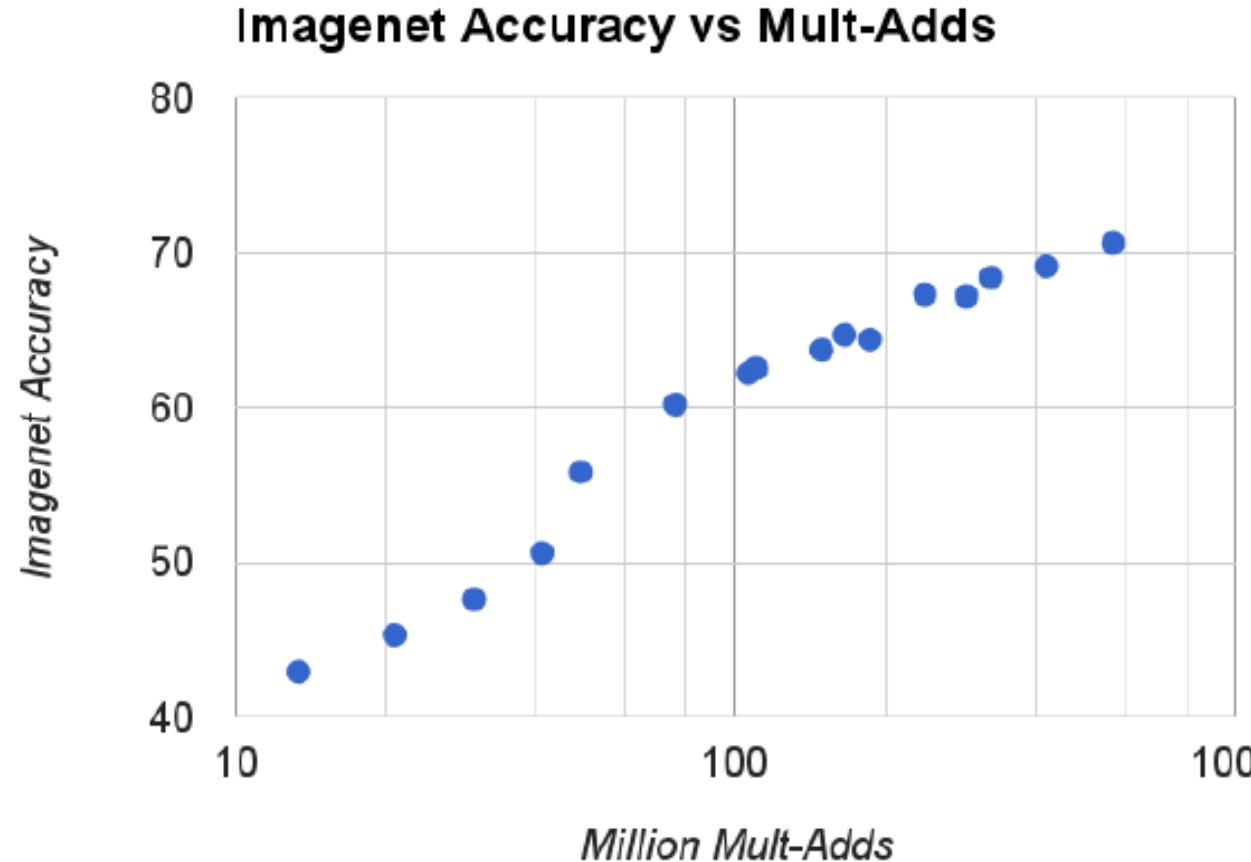
Table 6. MobileNet Width Multiplier

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

Table 7. MobileNet Resolution

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

Model Shrinking Hyperparameters



Results

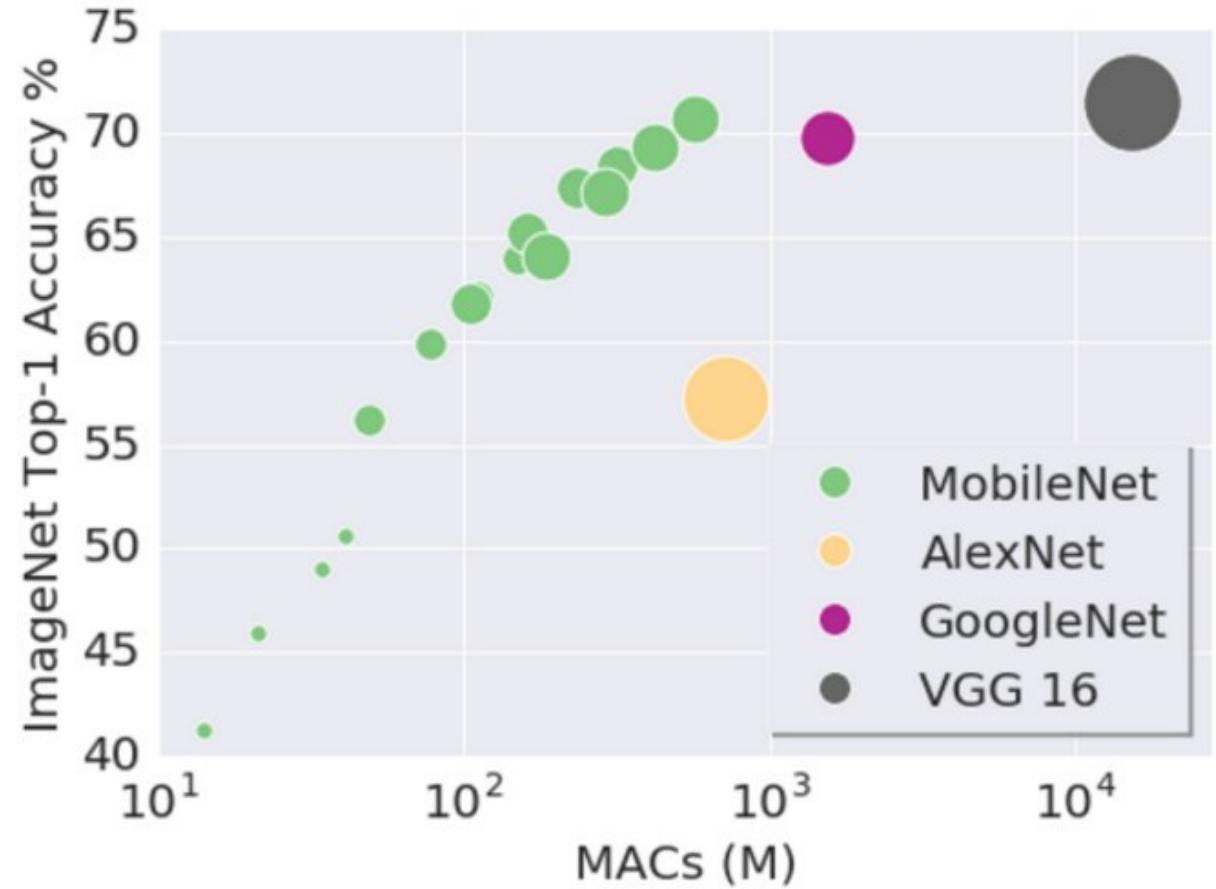
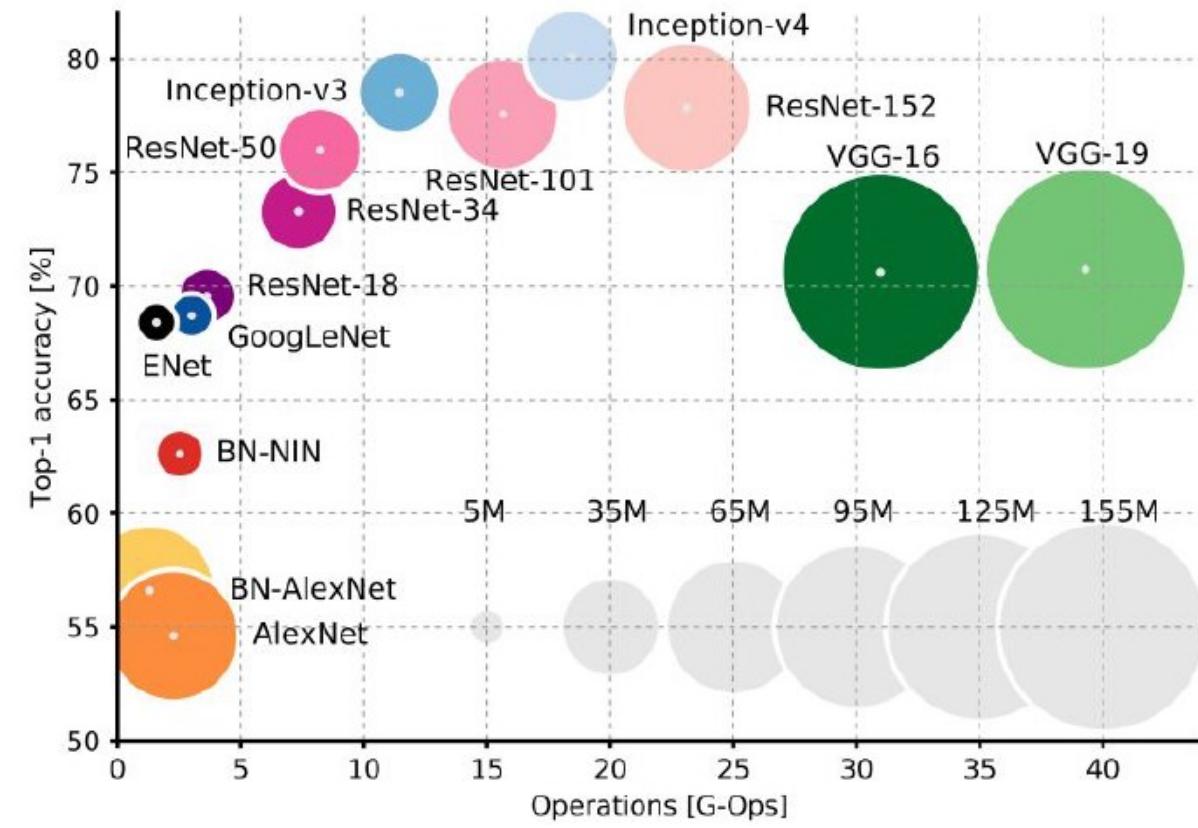
Table 8. MobileNet Comparison to Popular Models

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



Results

Table 10. MobileNet for Stanford Dogs

Model	Top-1 Accuracy	Million Mult-Adds	Million Parameters
Inception V3 [18]	84%	5000	23.2
1.0 MobileNet-224	83.3%	569	3.3
0.75 MobileNet-224	81.9%	325	1.9
1.0 MobileNet-192	81.9%	418	3.3
0.75 MobileNet-192	80.5%	239	1.9

Table 11. Performance of PlaNet using the MobileNet architecture. Percentages are the fraction of the Im2GPS test dataset that were localized within a certain distance from the ground truth. The numbers for the original PlaNet model are based on an updated version that has an improved architecture and training dataset.

Scale	Im2GPS [7]	PlaNet [35]	PlaNet MobileNet
Continent (2500 km)	51.9%	77.6%	79.3%
Country (750 km)	35.4%	64.0%	60.3%
Region (200 km)	32.1%	51.1%	45.2%
City (25 km)	21.9%	31.7%	31.7%
Street (1 km)	2.5%	11.0%	11.4%

Results

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

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



Figure 6. Example objection detection results using MobileNet SSD.

Results

Table 12. Face attribute classification using the MobileNet architecture. Each row corresponds to a different hyper-parameter setting (width multiplier α and image resolution).

Width Multiplier / Resolution	Mean AP	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	88.7%	568	3.2
0.5 MobileNet-224	88.1%	149	0.8
0.25 MobileNet-224	87.2%	45	0.2
1.0 MobileNet-128	88.1%	185	3.2
0.5 MobileNet-128	87.7%	48	0.8
0.25 MobileNet-128	86.4%	15	0.2
Baseline	86.9%	1600	7.5

Table 14. MobileNet Distilled from FaceNet

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

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

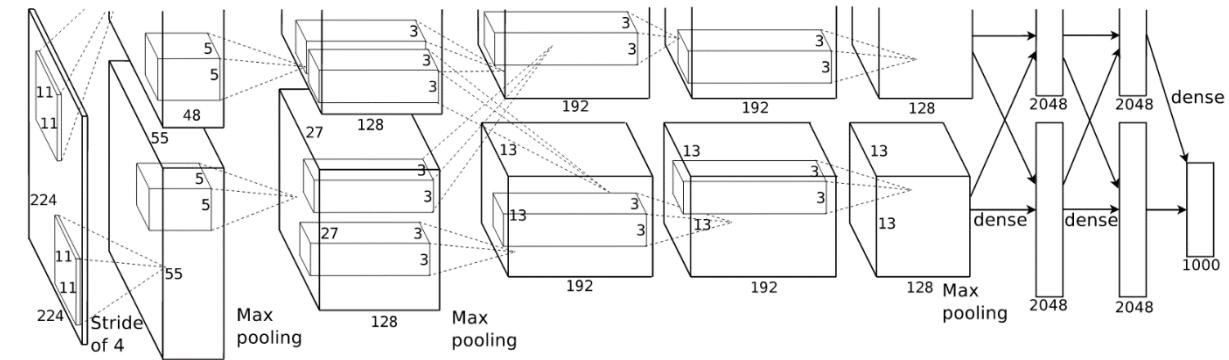
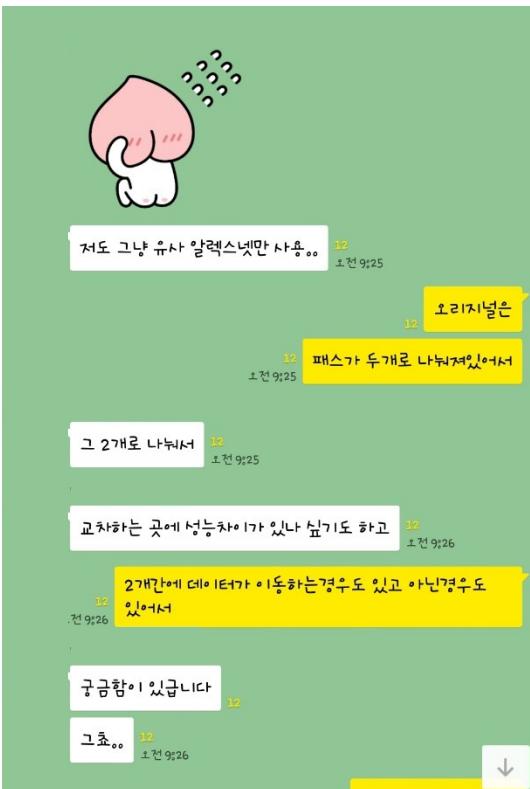
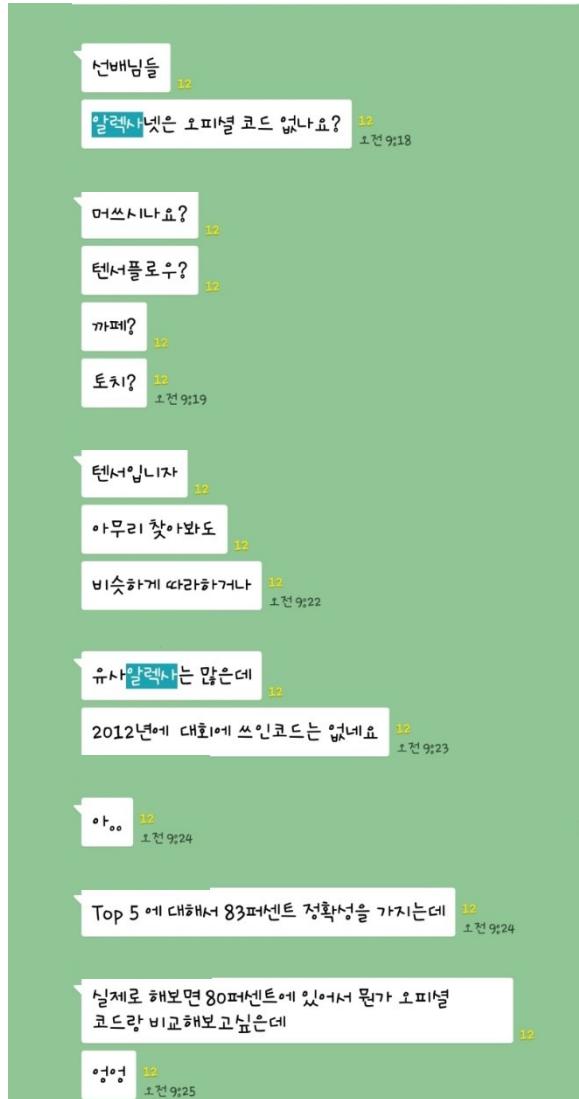
- The 1×1 convolution accounts for most of the computation

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?

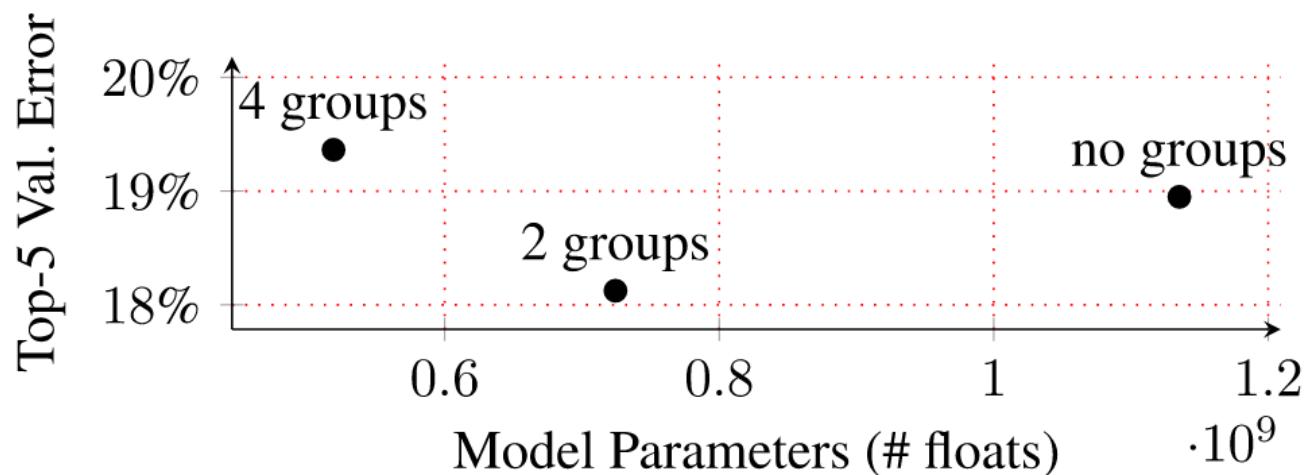
A Secret of AlexNet



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!

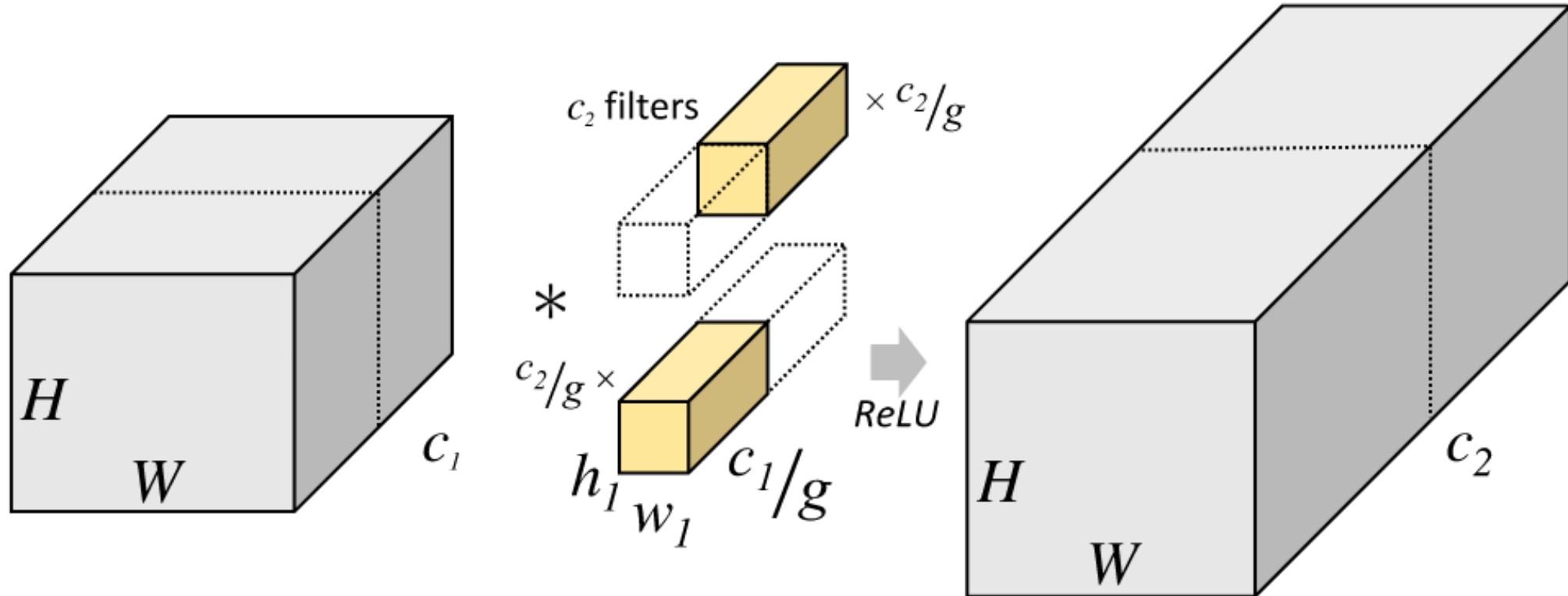


AlexNet trained with varying numbers of filter groups, from 1 (i.e. no filter groups), to 4. When trained with 2 filter groups, AlexNet is more efficient and yet achieves the same if not lower validation error.

Main Ideas of ShuffleNet

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

Grouped Convolution

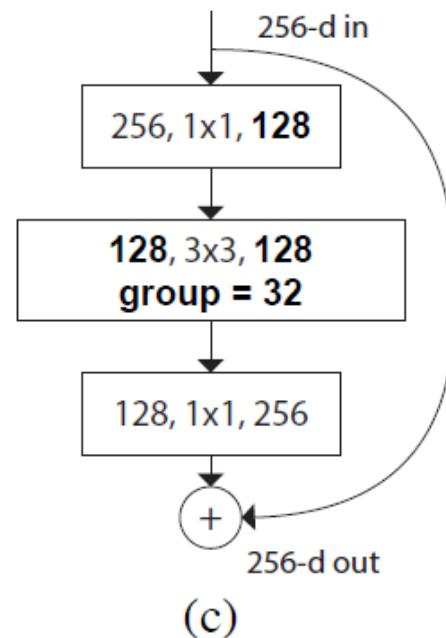
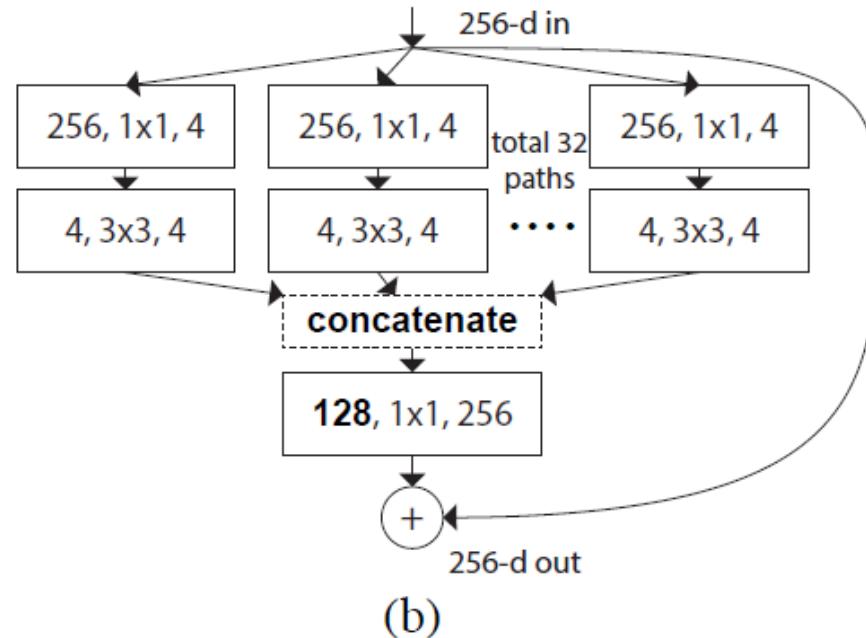
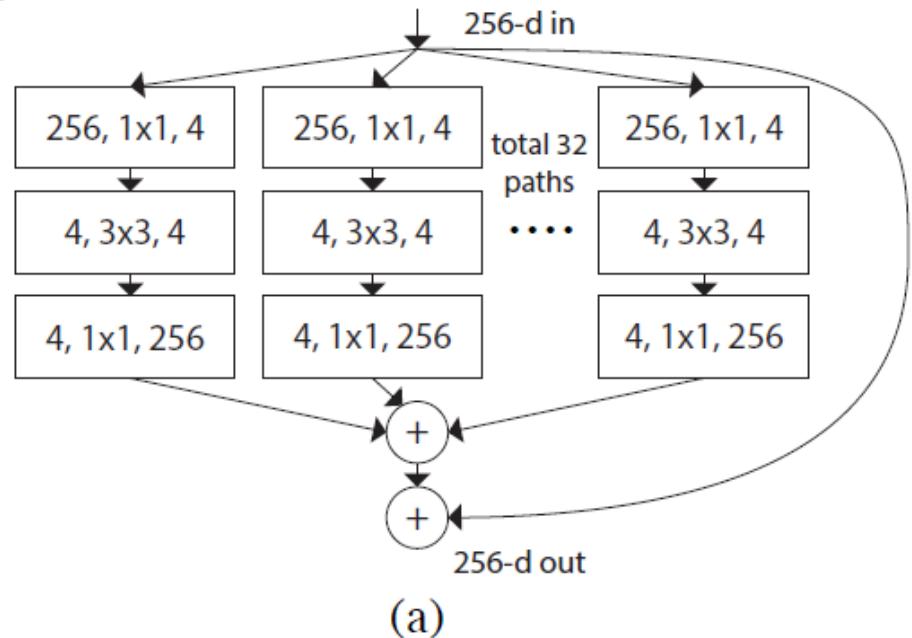


A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

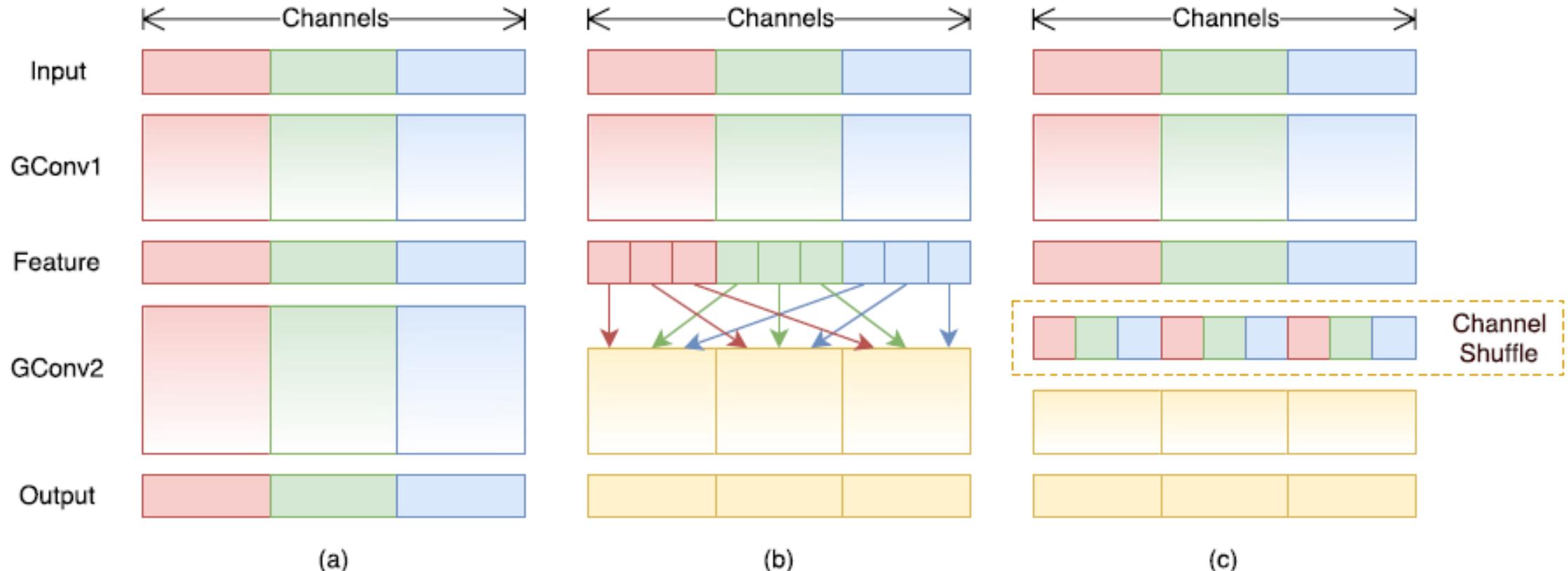
ResNeXt

- Equivalent building blocks of RexNext (cardinality=32)

equivalent



1x1 Grouped Convolution with Channel Shuffling



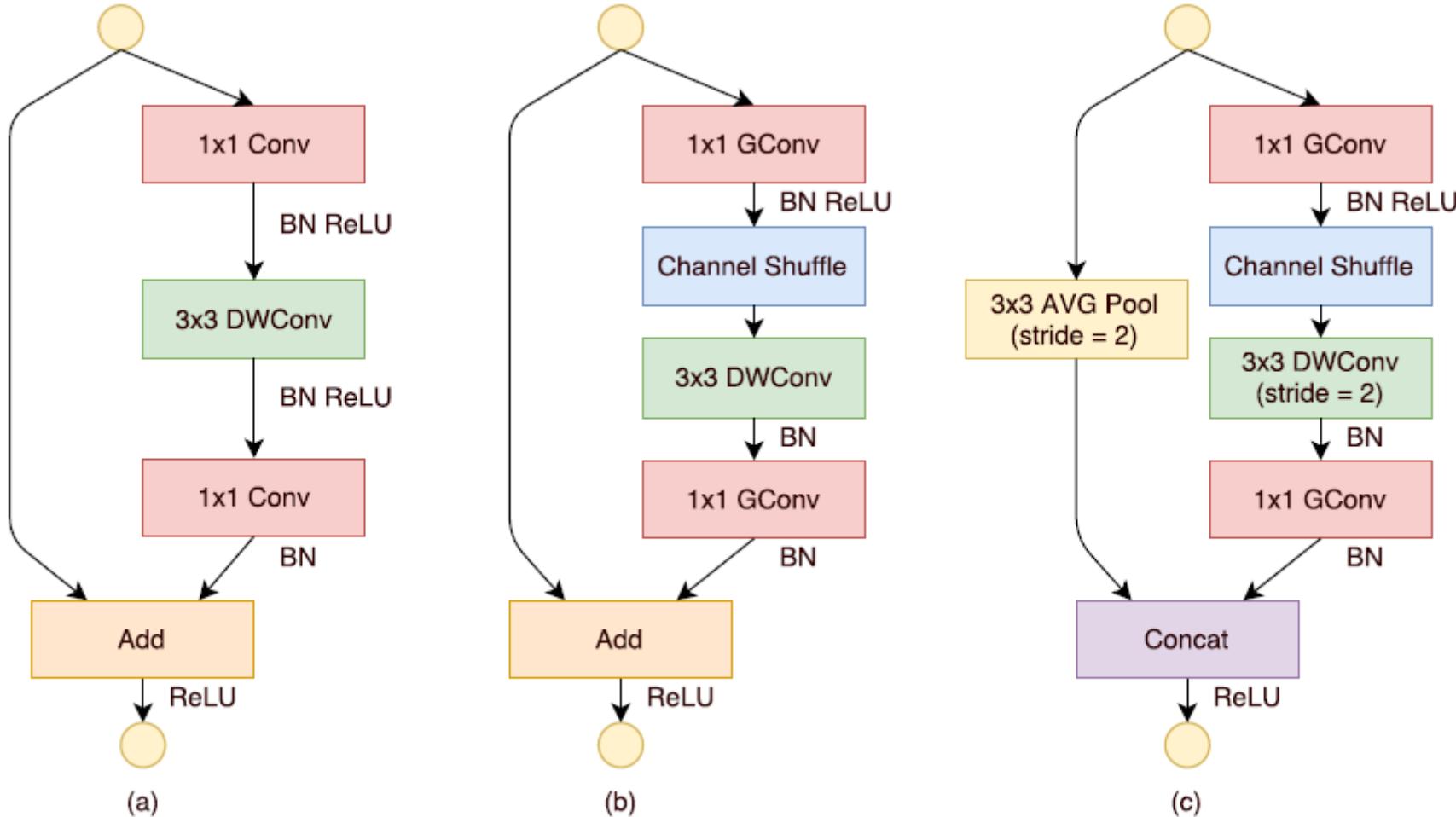
- If multiple group convolutions stack together, there is one side effect(a)
 - Outputs from a certain channel are only derived from a small fraction of input channels
- If we allow group convolution to obtain input data from different groups, the input and output channels will be fully related.

Channel Shuffle Operation

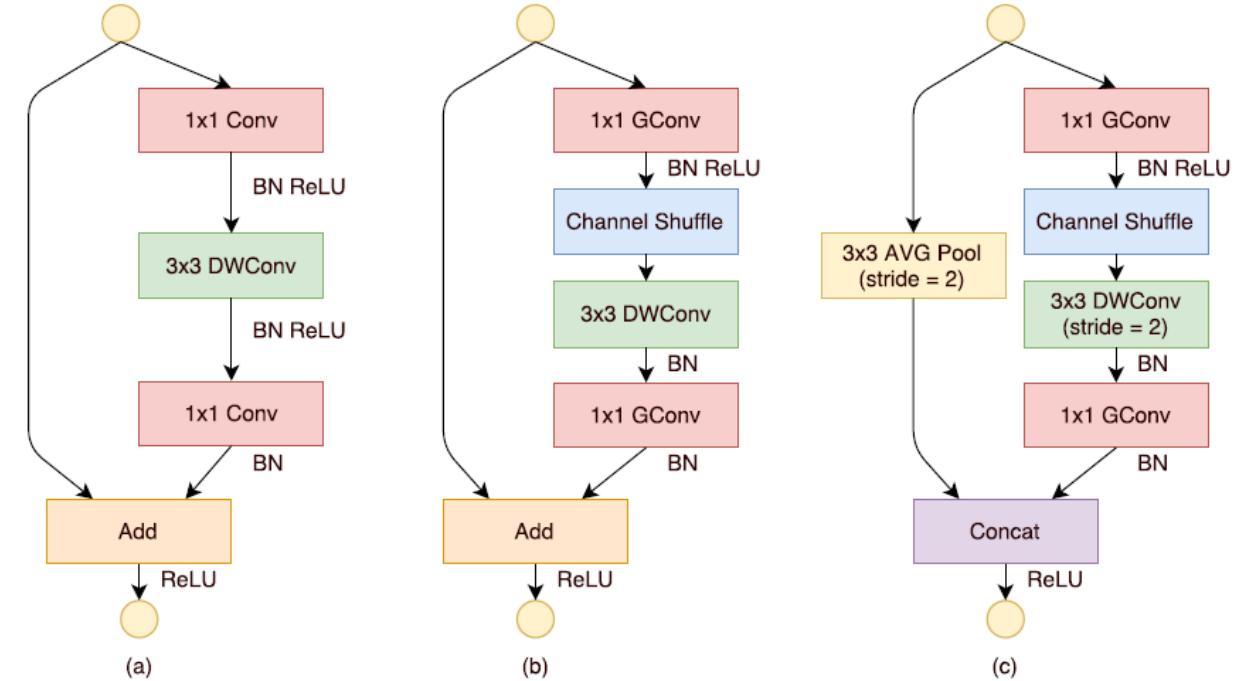
- Suppose a convolutional layer with g groups whose output has $g \times n$ channels; we first reshape the output channel dimension into (g, n) , transposing and then flattening it back as the input of next layer.
- Channel shuffle operation is also differentiable

```
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



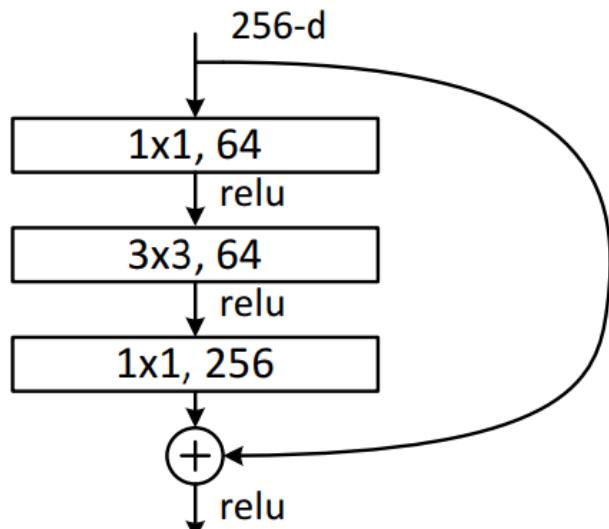
ShuffleNet Units



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

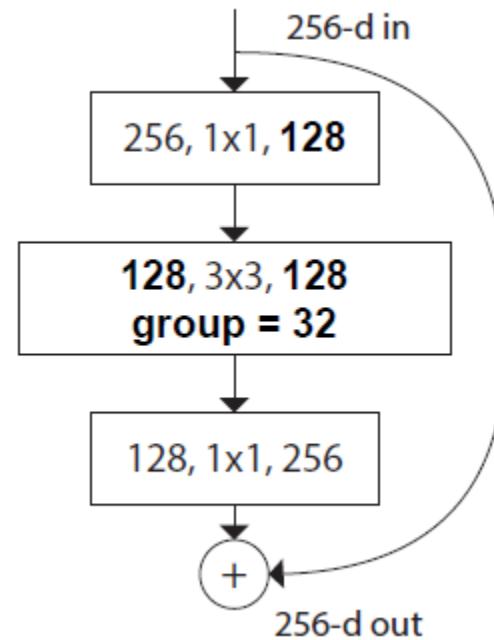
Complexity

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



ResNet

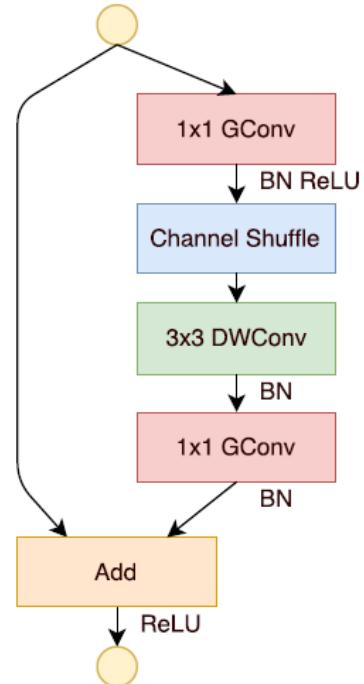
$$hw(2cm + 9m^2)$$



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 channel shuffle consistently boosts classification scores for different settings

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

Table 3. ShuffleNet with/without channel shuffle (*smaller number represents better performance*)

Experimental Results

Complexity (MFLOPs)	VGG-like	ResNet	Xception-like	ResNeXt	ShuffleNet (ours)
140	50.7	37.3	33.6	33.3	32.4 ($1\times, g = 8$)
38	-	48.8	45.1	46.0	41.6 ($0.5\times, g = 4$)
13	-	63.7	57.1	65.2	52.7 ($0.25\times, g = 8$)

Table 4. Classification error vs. various structures (% , smaller number represents better performance). We do not report VGG-like structure on smaller networks because the accuracy is significantly worse.

- ShuffleNet models outperform most others by a significant margin under different complexities
- Interestingly, they found an empirical relationship between feature map channels and classification accuracy
 - Under the complexity of 38 MFLOPs, output channels of Stage 4 for VGG-like, ResNet, ResNeXt, Xception-like. ShuffleNet models are 50, 192, 192, 288, 576 respectively
- PVANet has 29.7% classification errors with 557MFLOPs / ShuffleNet 2x model($g=3$) gets 26.3% with 524MFLOPs

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)

Experimental Results

Model	mAP [.5, .95] (300× image)	mAP [.5, .95] (600× image)
ShuffleNet 2× ($g = 3$)	18.7%	25.0%
ShuffleNet 1× ($g = 3$)	14.5%	19.8%
1.0 MobileNet-224 [12]	16.4%	19.8%
1.0 MobileNet-224 (our impl.)	14.9%	19.3%

Table 7. Object detection results on MS COCO (*larger numbers represents better performance*). For MobileNets we compare two results: 1) COCO detection scores reported by [12]; 2) finetuning from our reimplemented MobileNets, whose training and finetuning settings are exactly the same as that for ShuffleNets.

Model	Cls err. (%)	FLOPs	224 × 224	480 × 640	720 × 1280
ShuffleNet 0.5× ($g = 3$)	43.2	38M	15.2ms	87.4ms	260.1ms
ShuffleNet 1× ($g = 3$)	32.6	140M	37.8ms	222.2ms	684.5ms
ShuffleNet 2× ($g = 3$)	26.3	524M	108.8ms	617.0ms	1857.6ms
AlexNet [21]	42.8	720M	184.0ms	1156.7ms	3633.9ms
1.0 MobileNet-224 [12]	29.4	569M	110.0ms	612.0ms	1879.2ms

Table 8. Actual inference time on mobile device (*smaller number represents better performance*). The platform is based on a single Qualcomm Snapdragon 820 processor. All results are evaluated with **single thread**.

- Due to memory access and other overheads, every 4x theoretical complexity reduction usually result in ~2.6x actual speedup. Compared with AlexNet achieving ~13x actual speedup(the theoretical speedup is 18x)

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

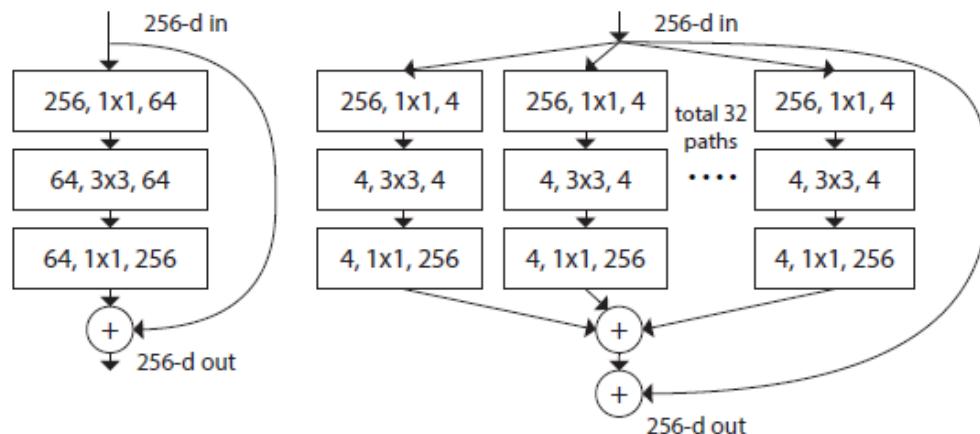


Figure 1. **Left:** A block of ResNet [14]. **Right:** A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

ImageNet 2016 Results

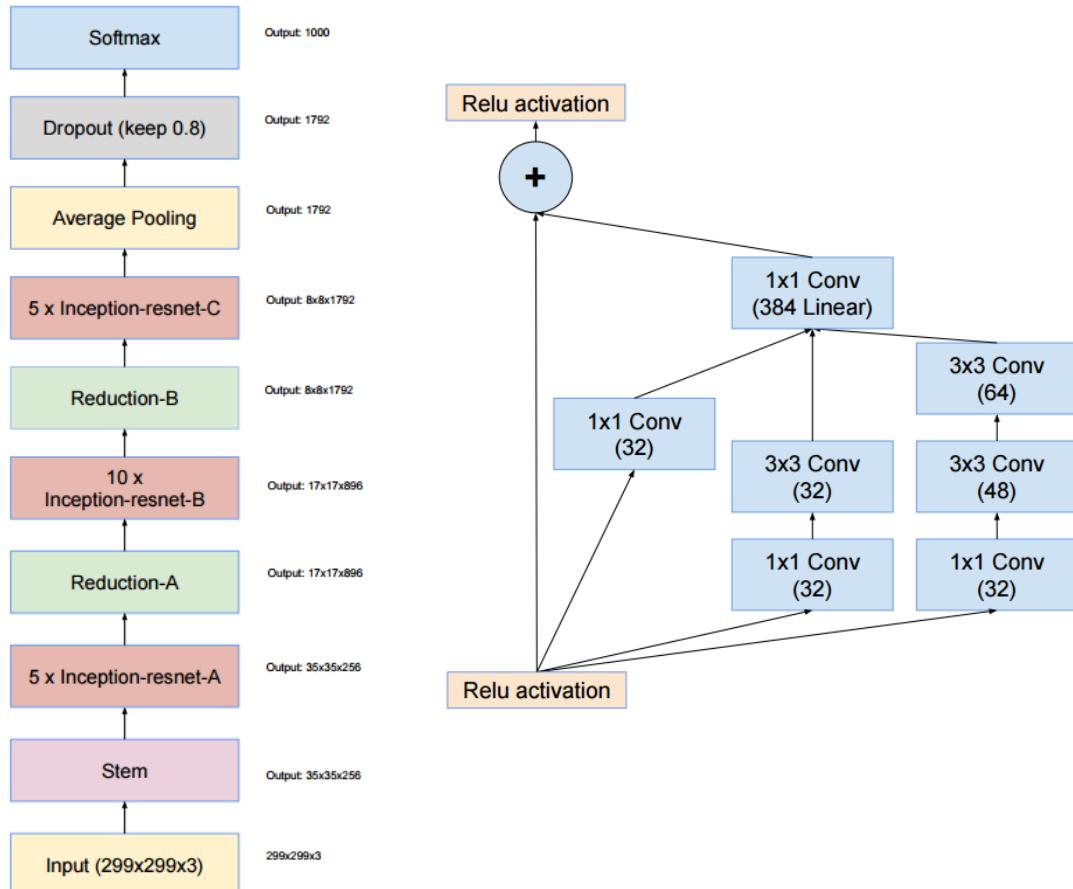
Team name	Entry description	Classification error	Localization error
Trimps-Soushen	Ensemble 2	0.02991	0.077668
Trimps-Soushen	Ensemble 3	0.02991	0.077087
Trimps-Soushen	Ensemble 4	0.02991	0.077429
ResNeXt	Ensemble C, weighted average, tuned on val. [No bounding box results]	0.03031	0.737308
CU-DeepLink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

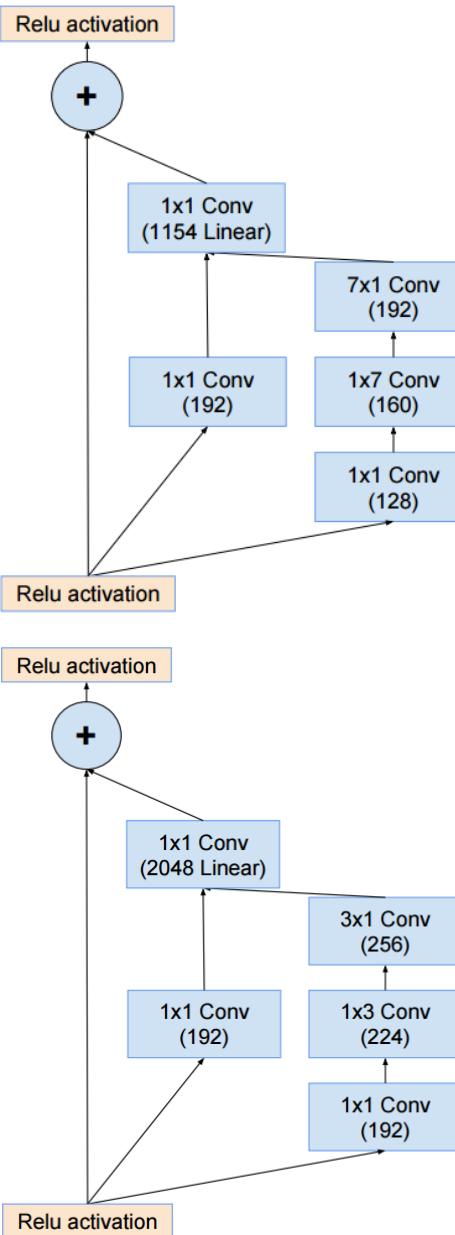
- VGGNet exhibit a simple yet effective strategy of constructing very deep network – **stacking building blocks of the same shape**
- ResNet inherit this strategy with **stacking modules of the same topology**
- Unlike VGGNet, the family of Inception models have demonstrated that carefully designed topologies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

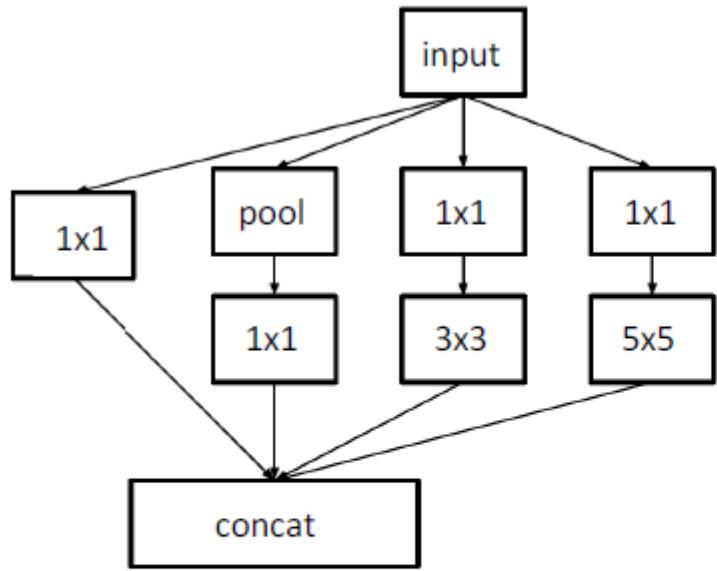


"Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning"



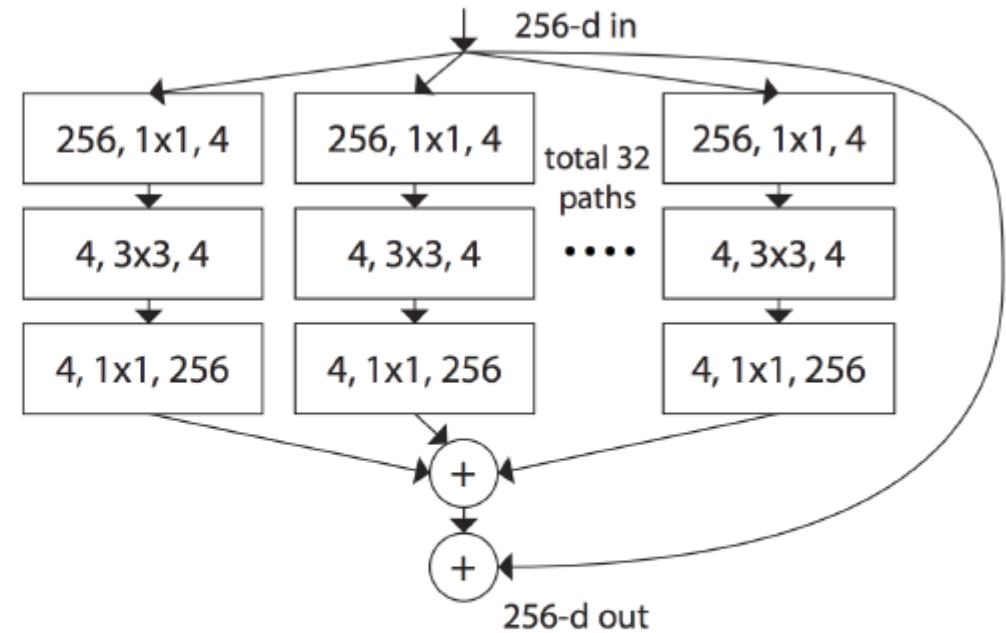
ResNet Learns Inception??

- Multi-branch + ResNet



Inception:

heterogeneous multi-branch

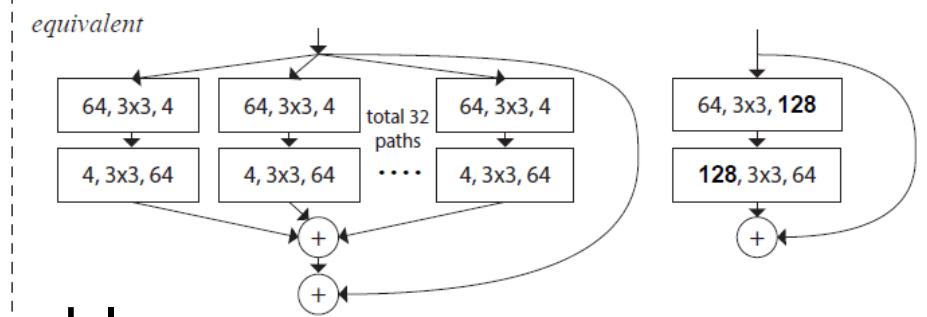


ResNeXt:

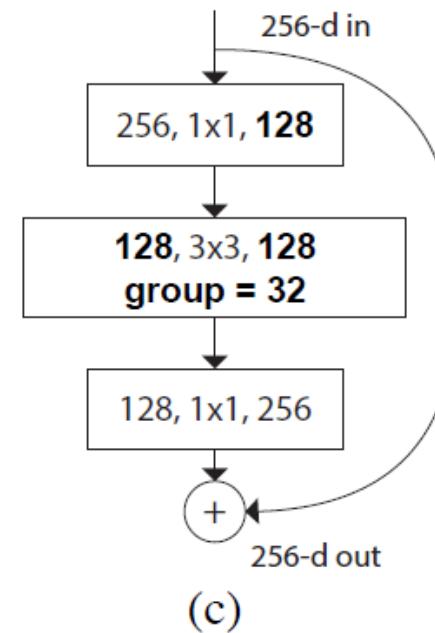
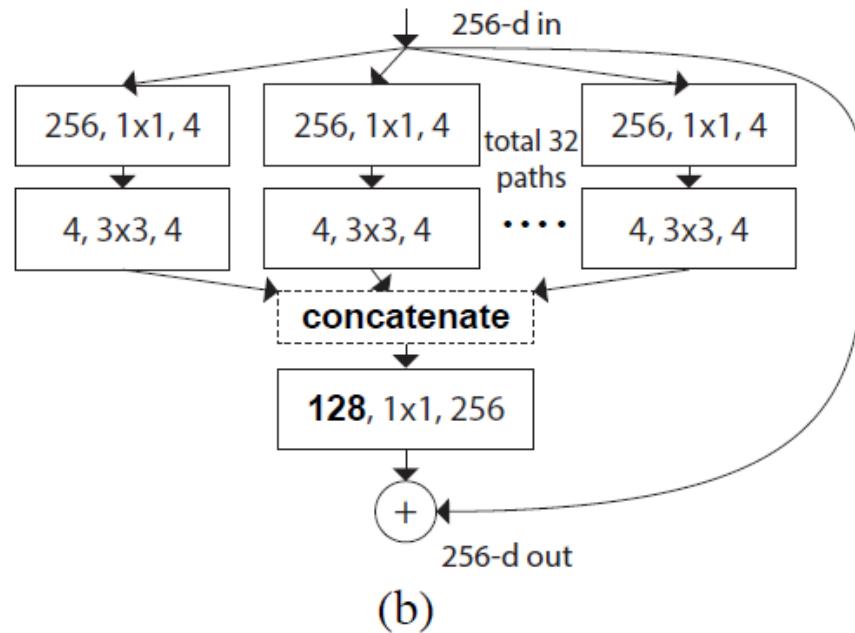
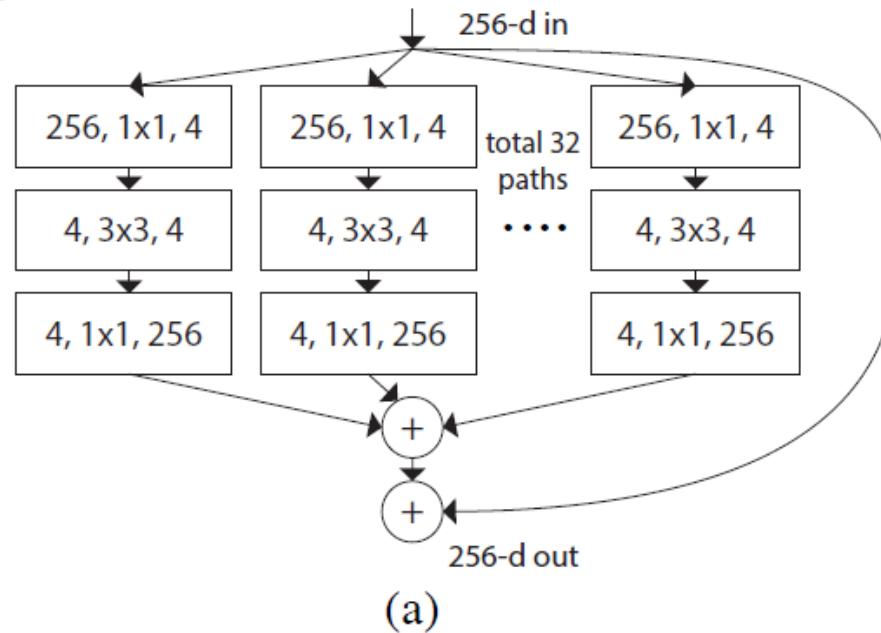
uniform multi-branch

ResNeXt

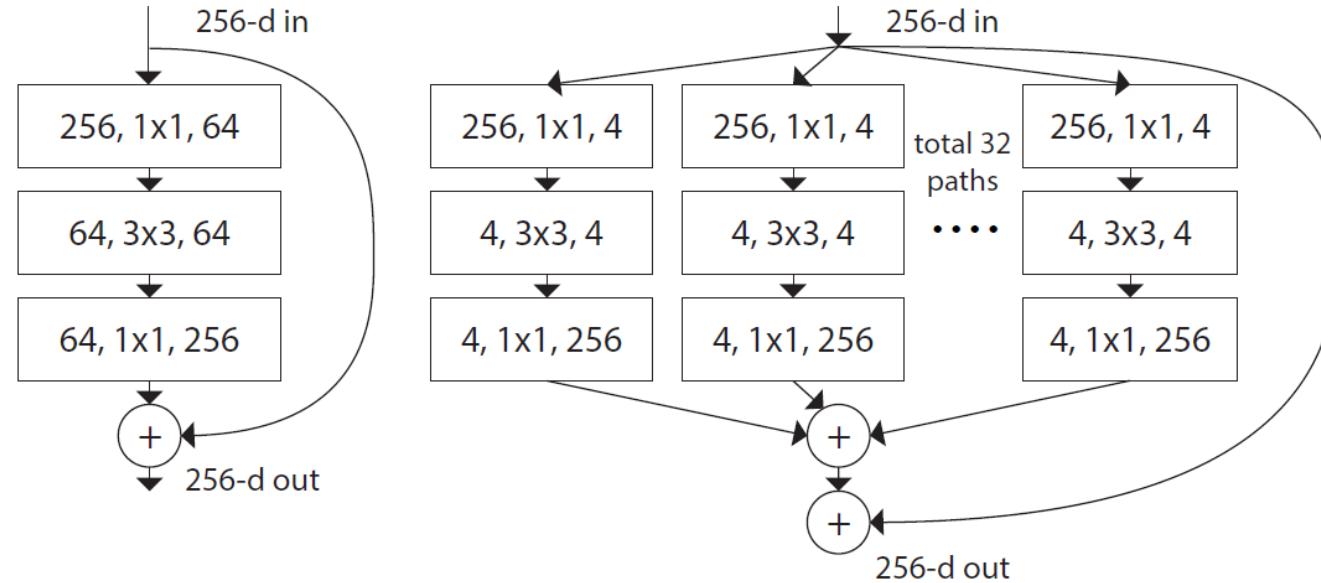
- Concatenation and Addition are interchangeable
- Uniform multi-branching can be done by group-conv



equivalent

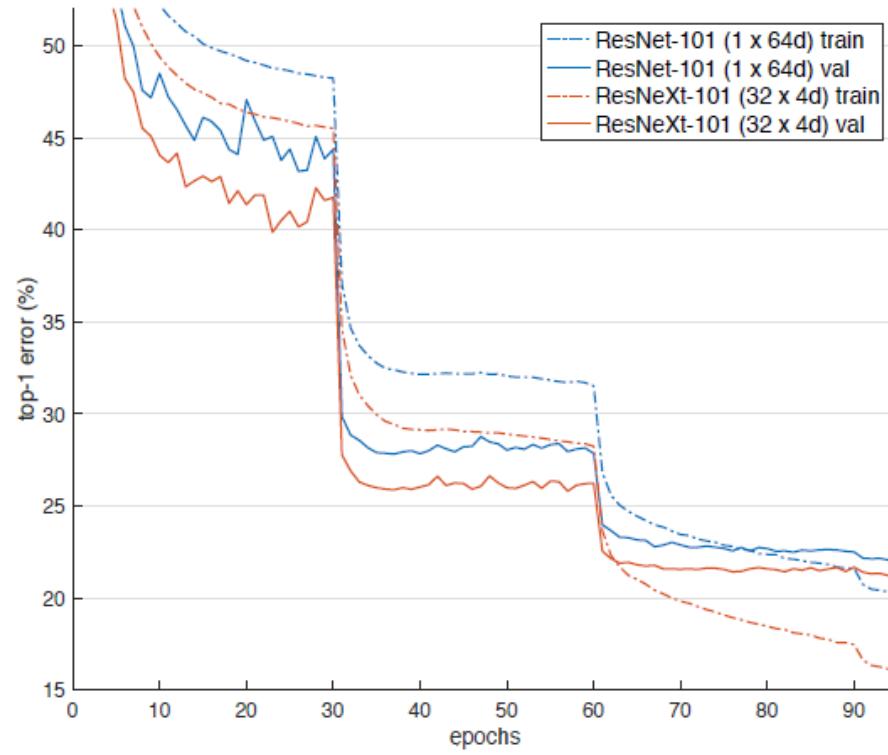
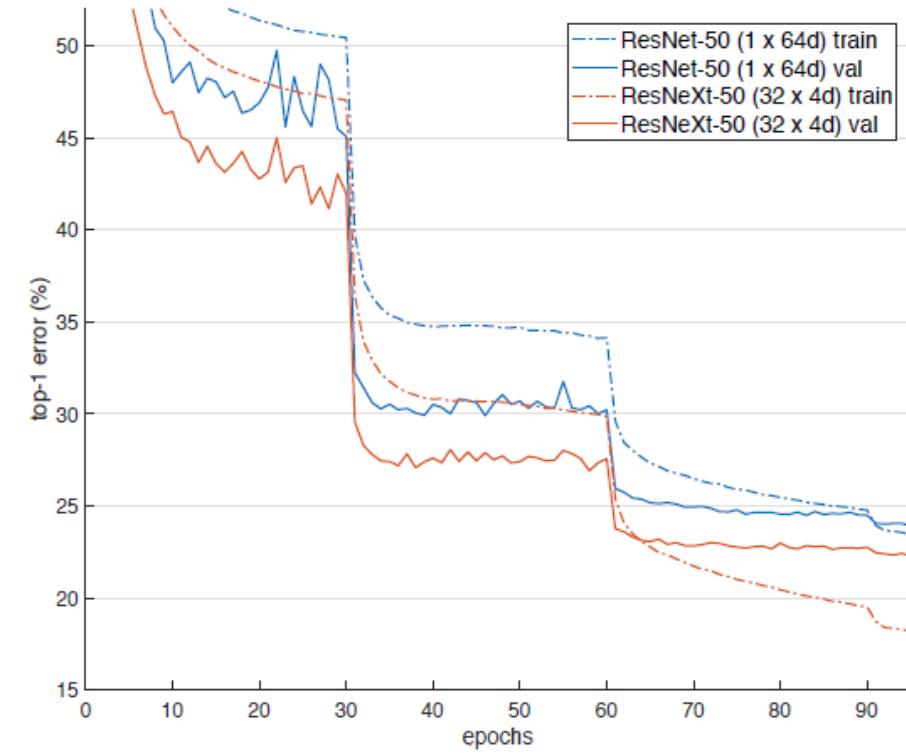


Model Capacity (# of parameters)



- 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

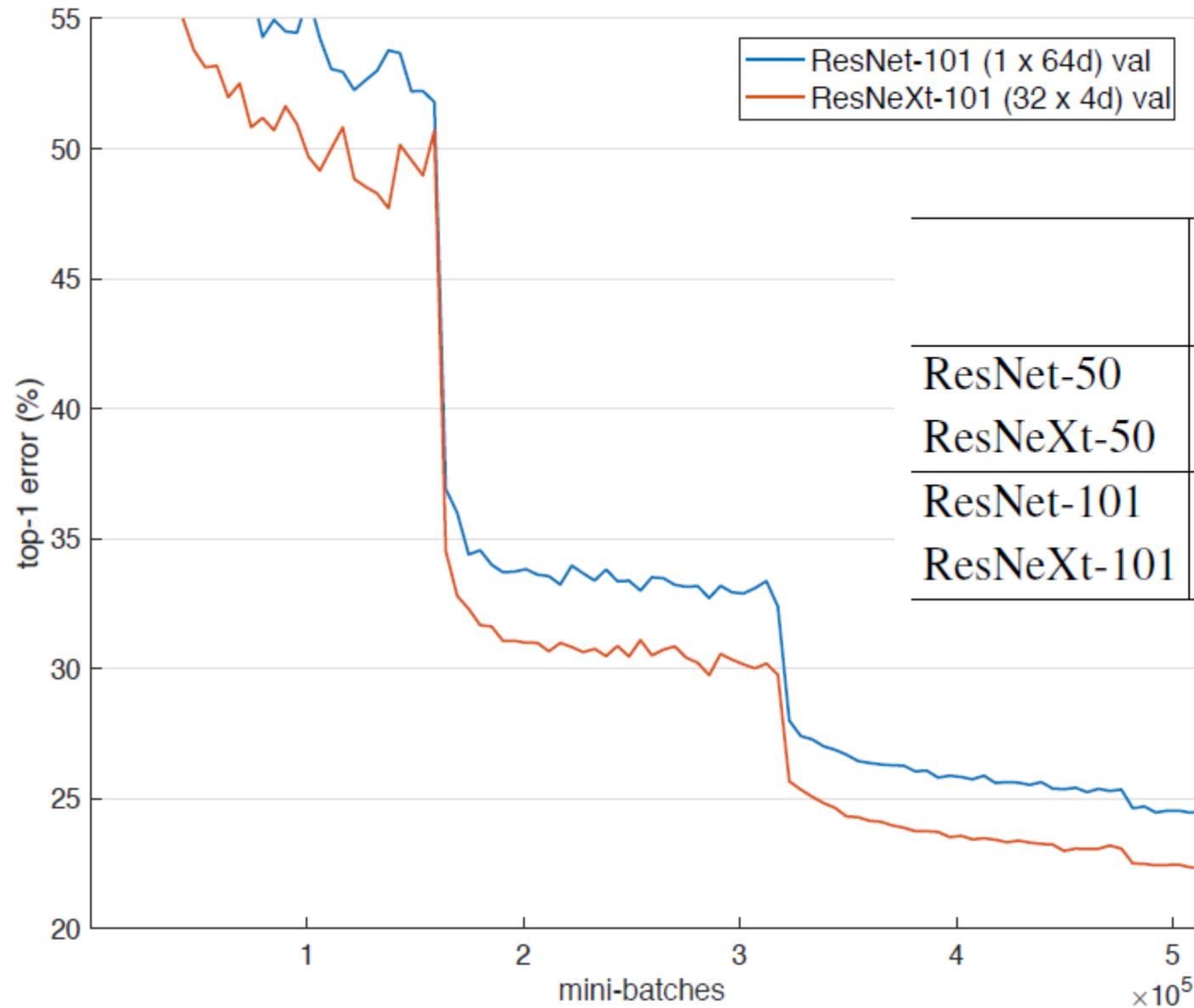
Results – Increasing Cardinality vs Deeper/Wider

	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

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].

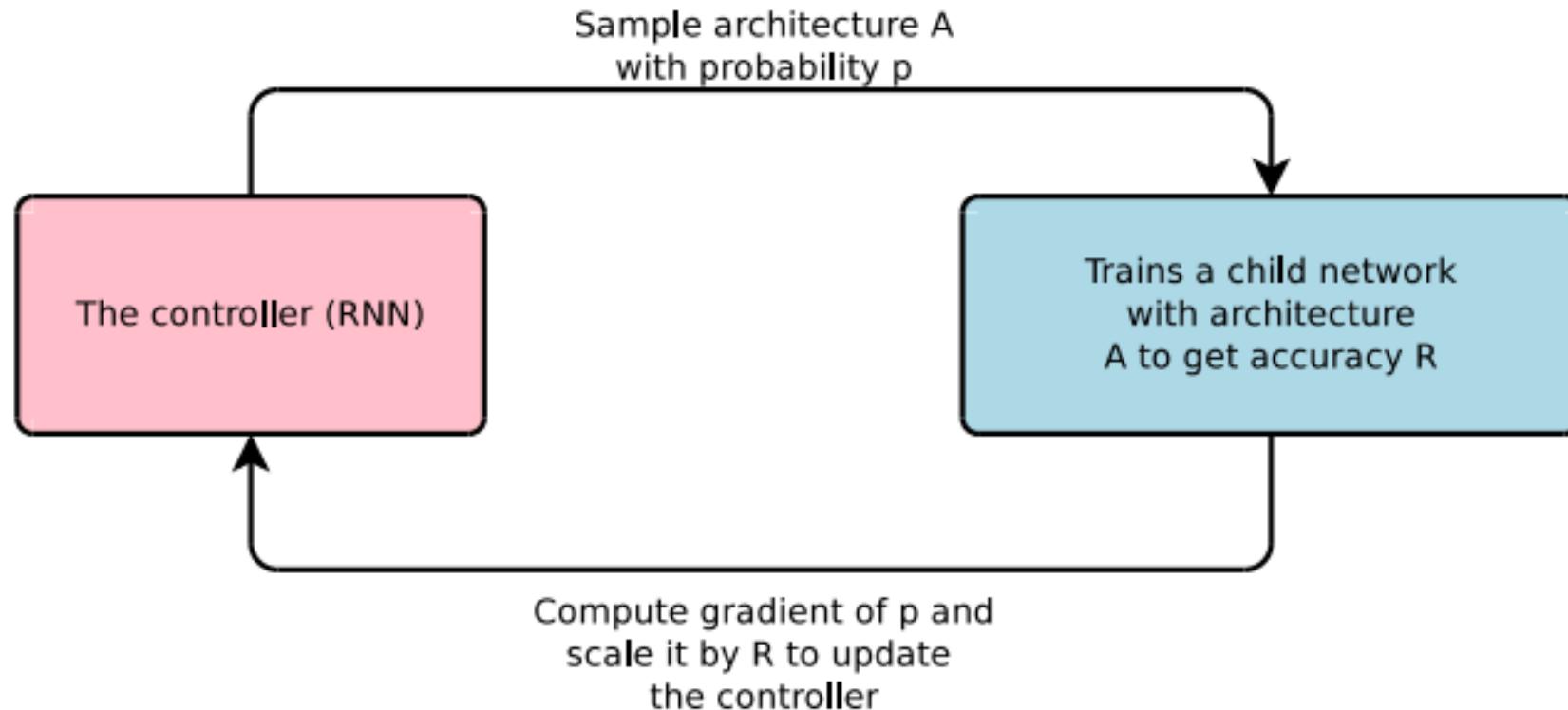
Tesorflow Pre-trained Models

- <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

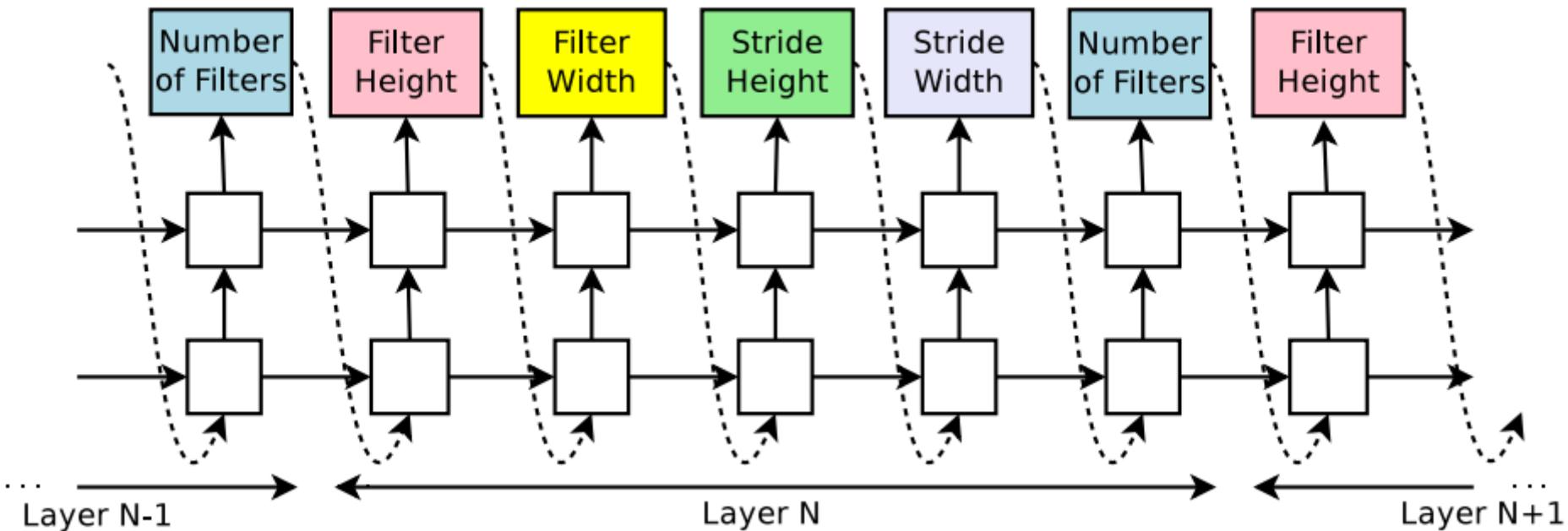
Neural Architecture Search

- 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



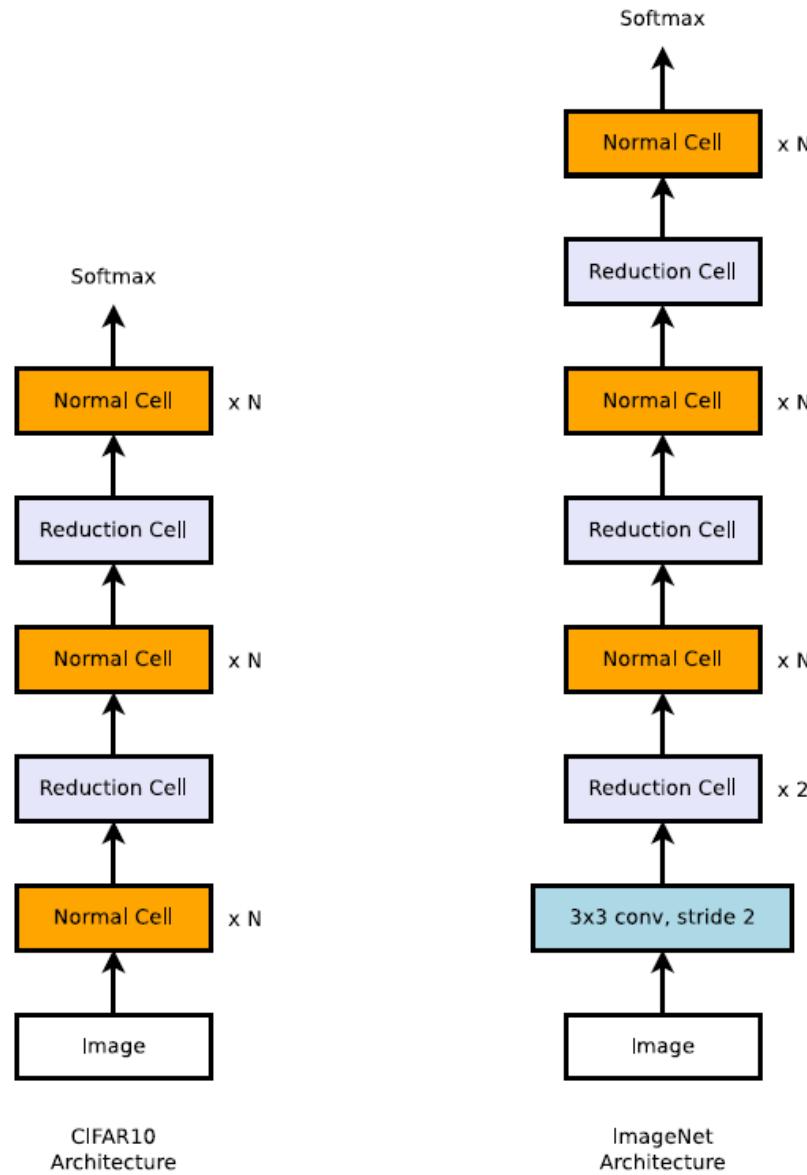
Motivation & Idea

- 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

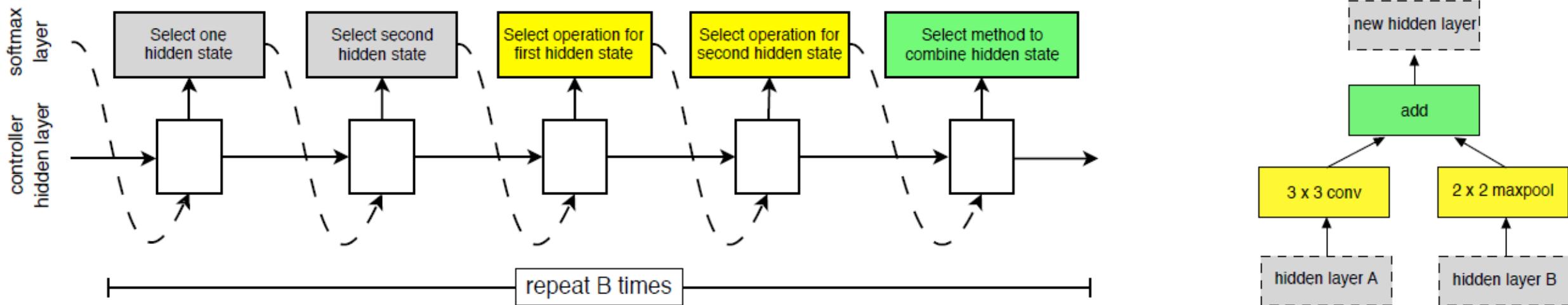
Method

- 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

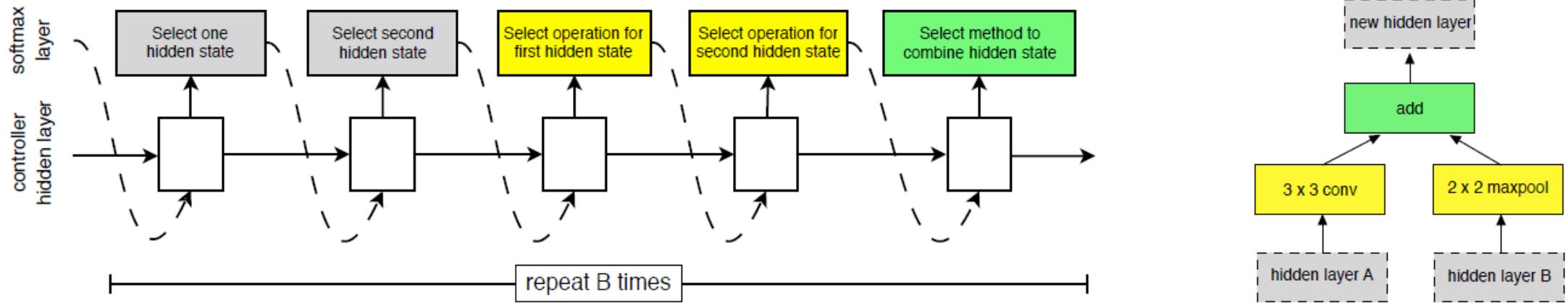


Models and Algorithms



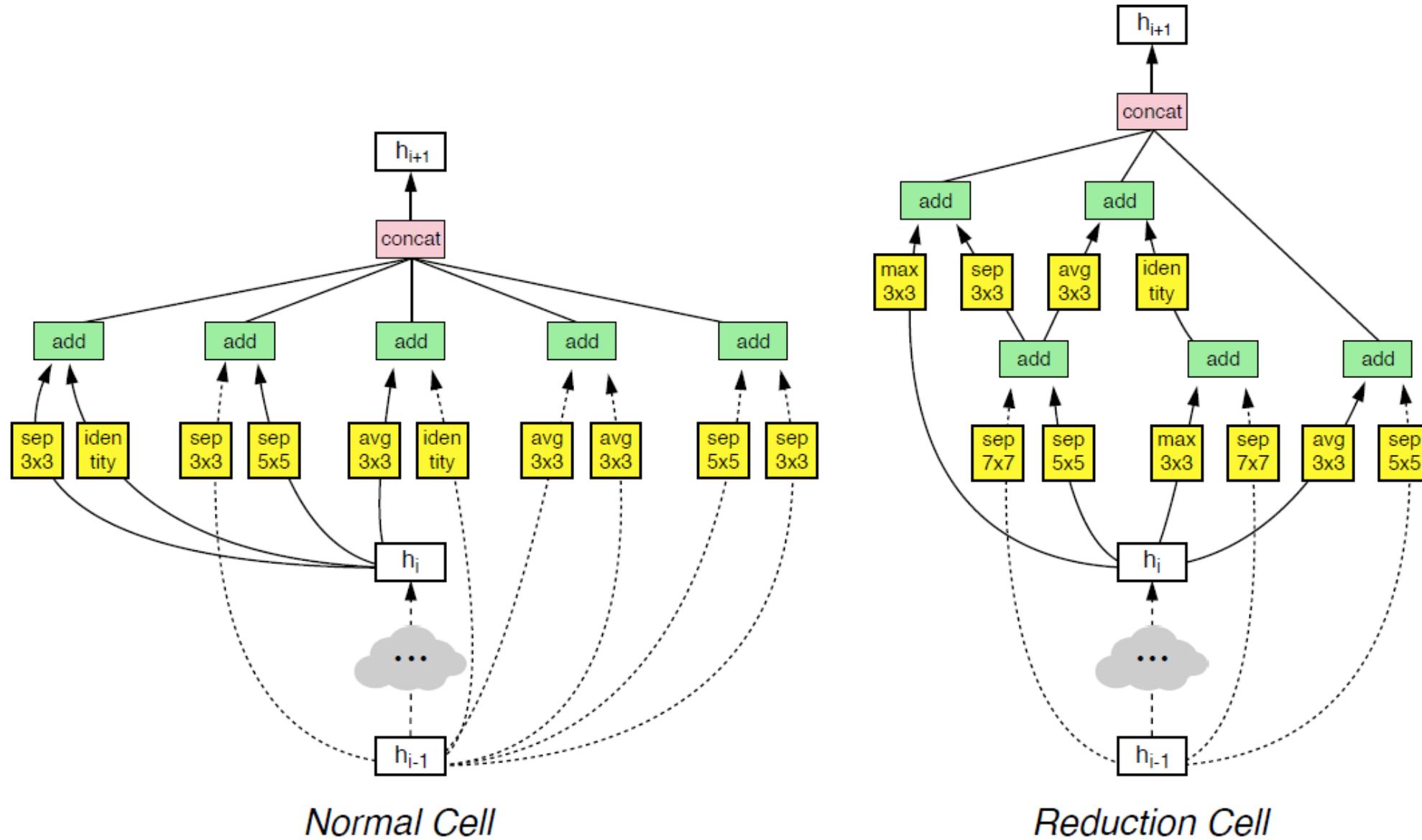
- Step 1.** Select a hidden state from h_i, h_{i-1} or from the set of hidden states created in previous blocks.
- Step 2.** Select a second hidden state from the same options as in Step 1.
- Step 3.** Select an operation to apply to the hidden state selected in Step 1.
- Step 4.** Select an operation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

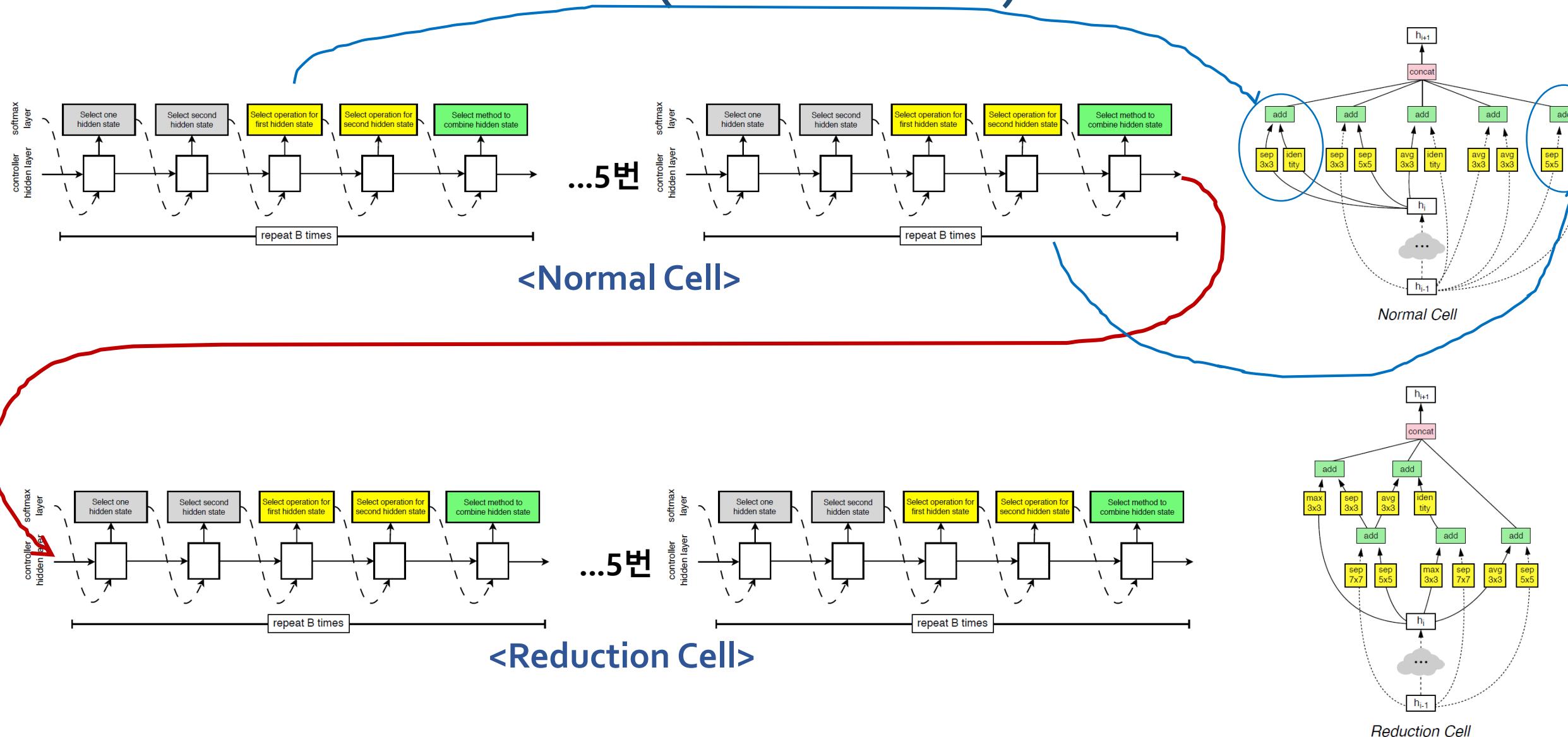


- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-separable conv

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

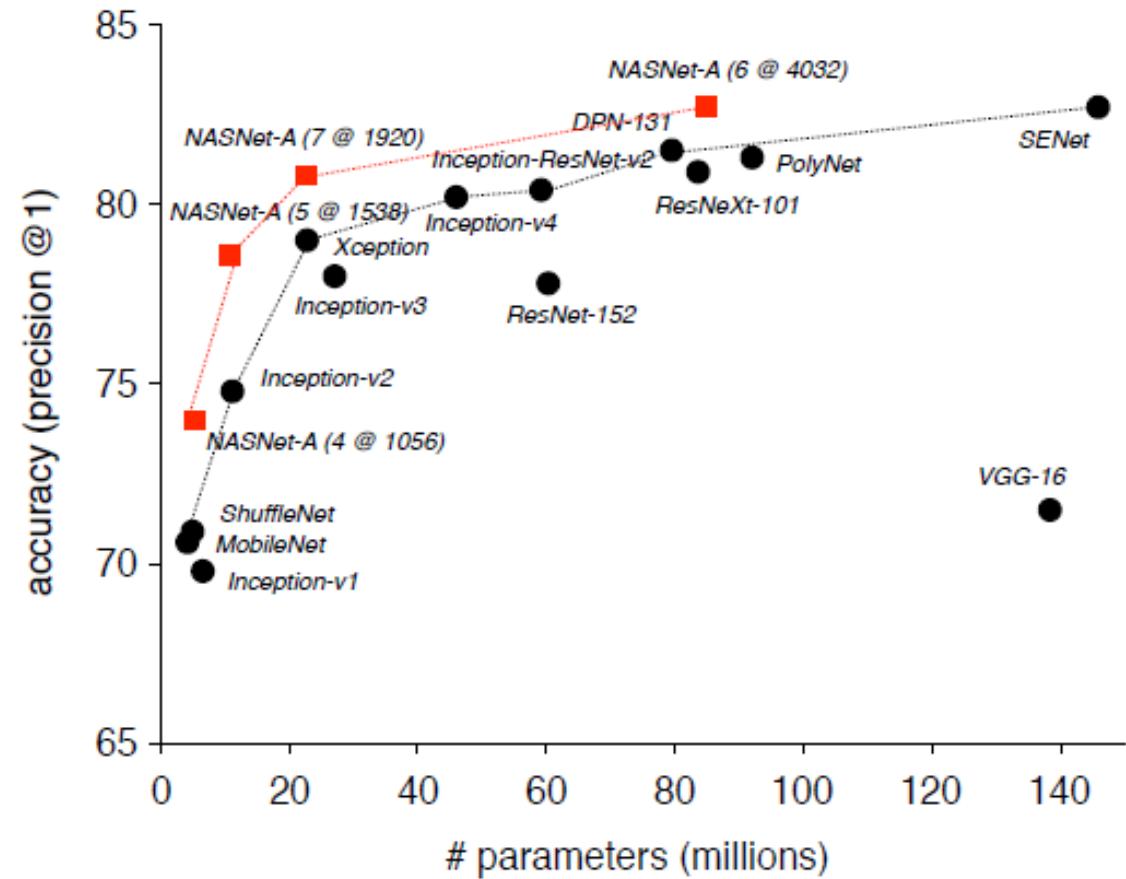
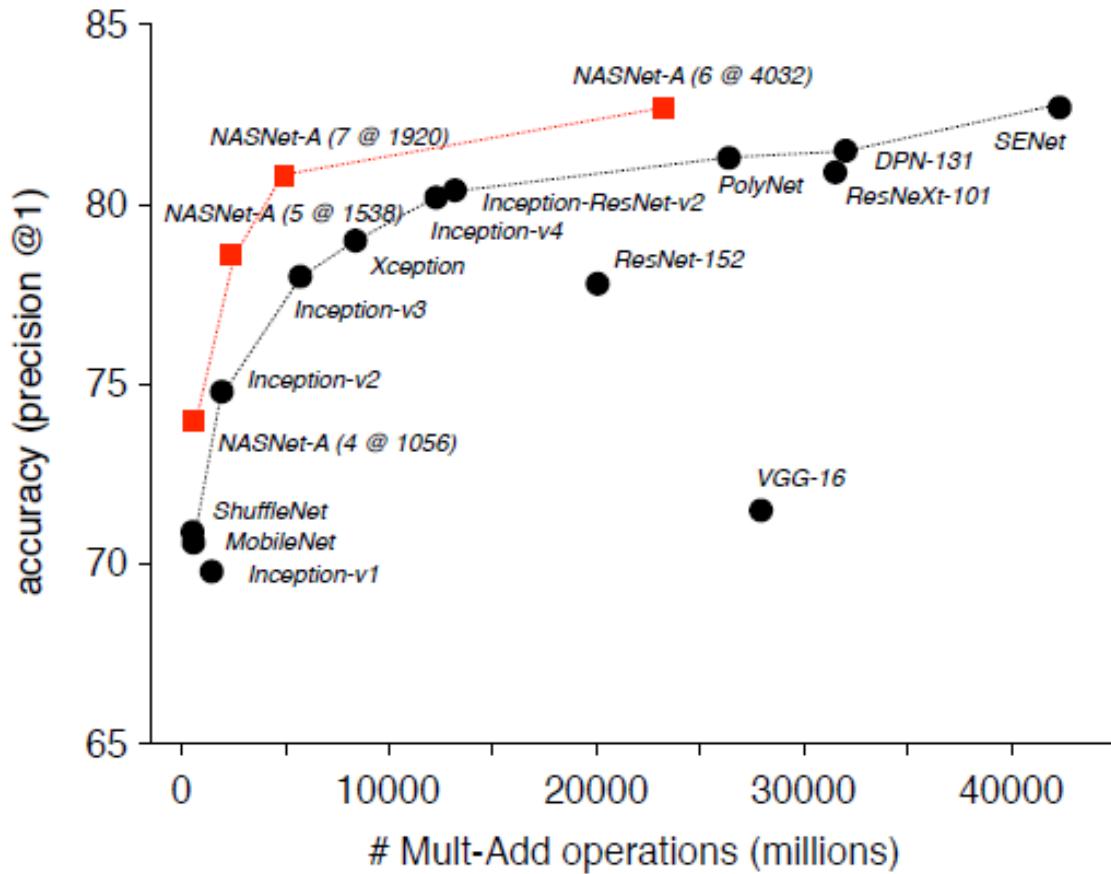
Results on ImageNet (vs SOTA)

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

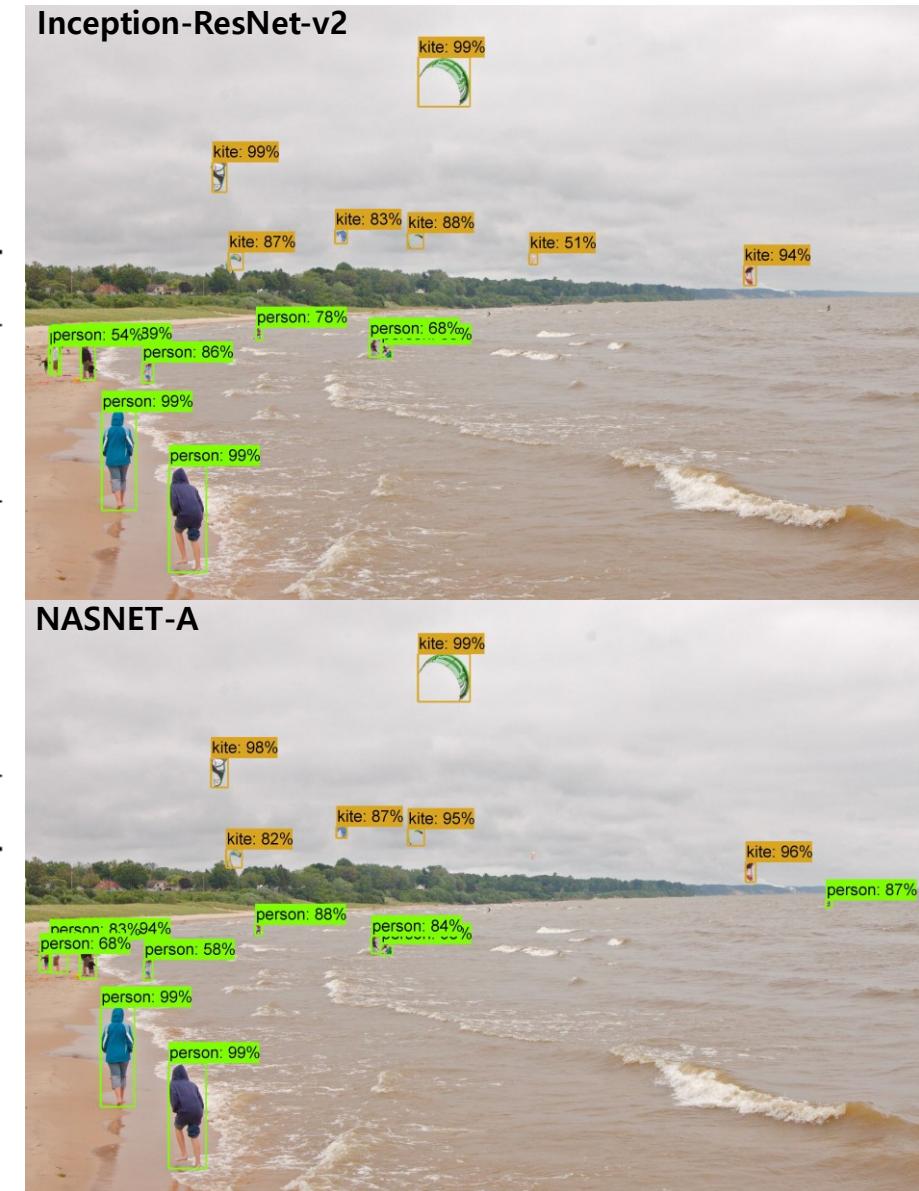
Model	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V1 [58]	6.6M	1,448 M	69.8	89.9
MobileNet-224 [24]	4.2 M	569 M	70.6	89.5
ShuffleNet (2x) [69]	~ 5M	524 M	70.9	89.8
NASNet-A (4 @ 1056)	5.3 M	564 M	74.0	91.6
NASNet-B (4 @ 1536)	5.3M	488 M	72.8	91.3
NASNet-C (3 @ 960)	4.9M	558 M	72.5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

Results on ImageNet

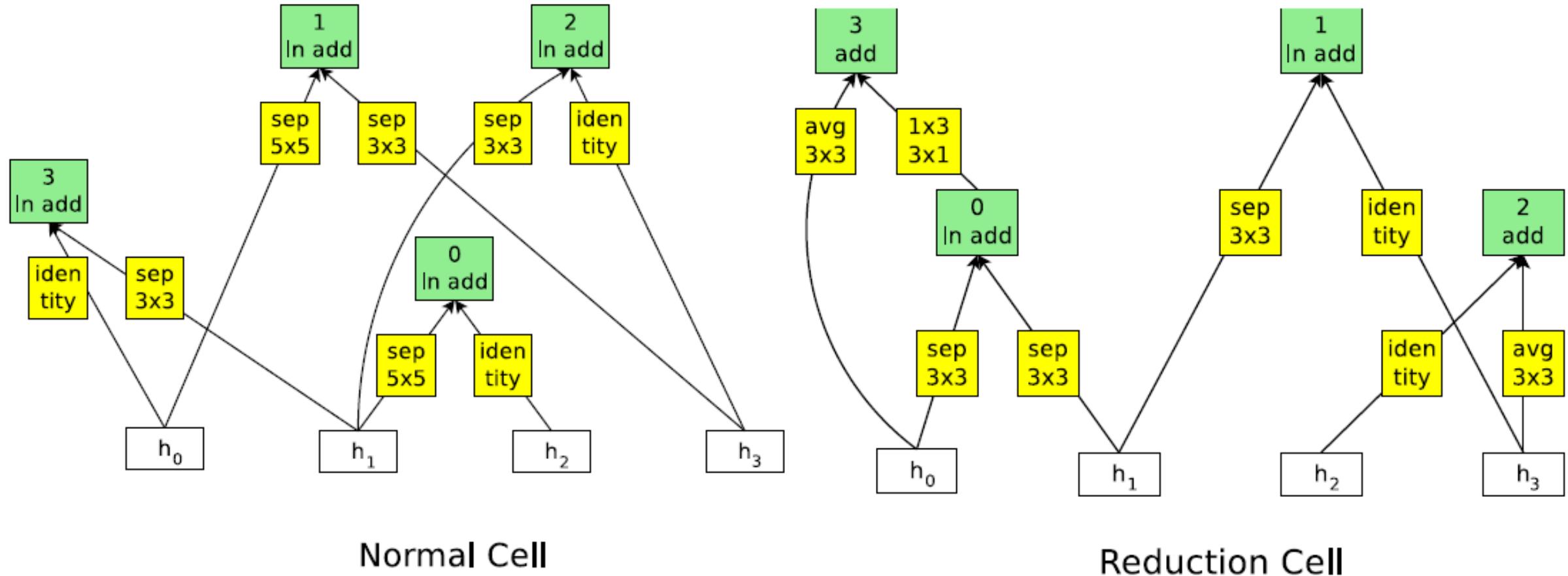


Object Detection Performance on COCO

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

