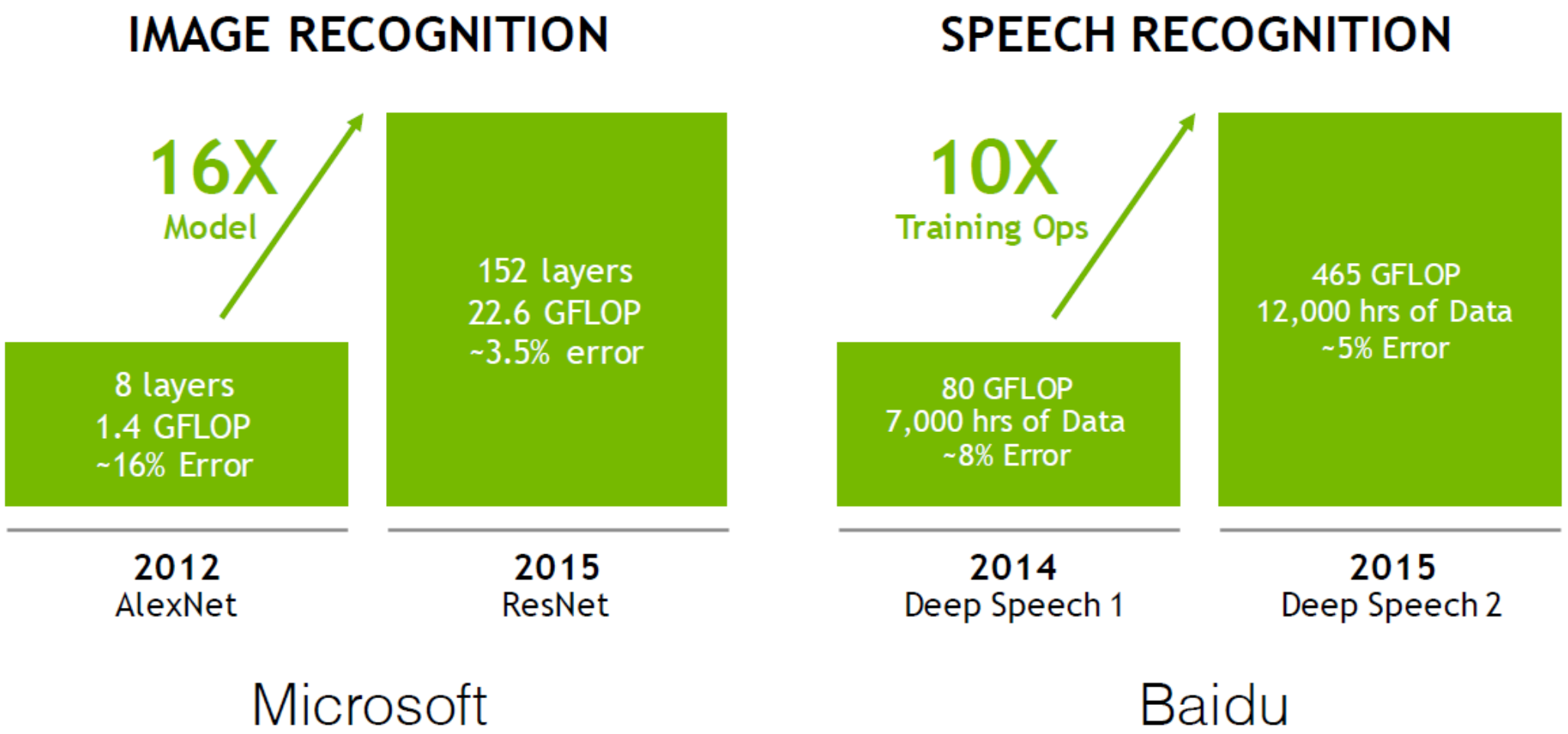
Modern CNNs II

Small Neural Networks

Fast Campus

Start Deep Learning with TensorFlow

Deep Learning Models are Getting Larger

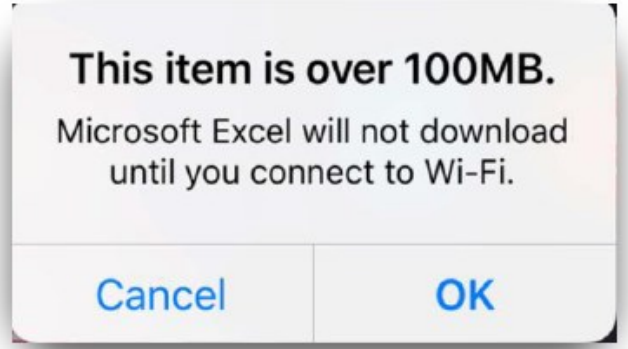


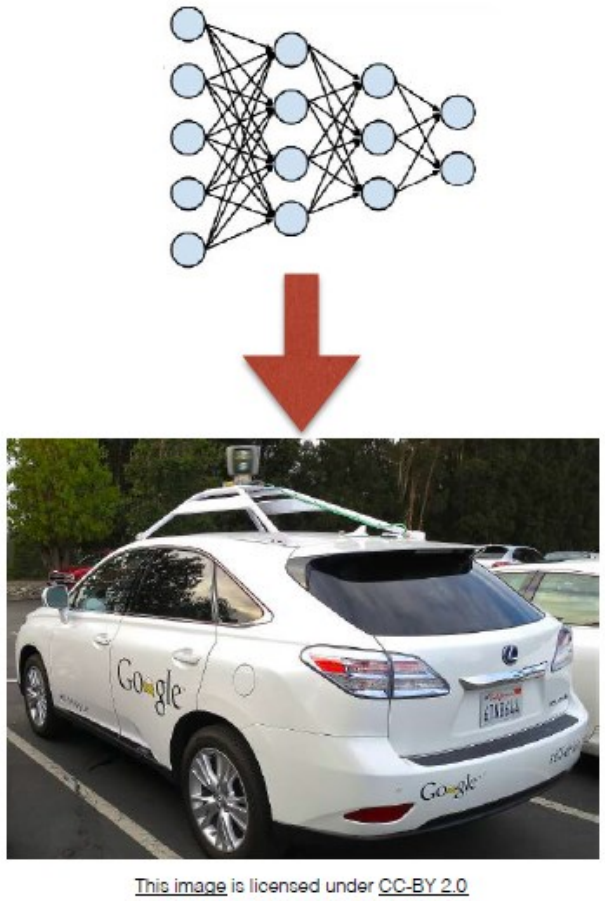
Dally, NIPS'2016 workshop on Efficient Methods for Deep Neural Networks

The First Challenge: Model Size

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

App icon is in the public domain
Phone image is licensed under CC-BY 2.0





This image is licensed under CC-BY 2.0

Slide Credit : CS231n 2017 lecture15 – Efficient Methods and hardware for Deep Learning

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

Slide Credit : CS231n 2017 lecture15 – Efficient Methods and hardware for Deep Learning

The Third Challenge: Energy Efficiency

AlphaGo: 1920 CPUs and 280 GPUs,

$3000 electric bill per game

on mobile: drains battery

on data-center: increases TCO

Slide Credit : CS231n 2017 lecture15 – Efficient Methods and hardware for Deep Learning

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

Slide Credit : Small Deep Neural Networks - Their Advantages, and Their Design by Forrest Iandola

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

Slide Credit : Small Deep Neural Networks - Their Advantages, and Their Design by Forrest Iandola

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)

Slide Credit : Small Deep Neural Networks - Their Advantages, and Their Design by Forrest Iandola

Techniques for Small Deep Neural Networks

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

Slide Credit : Small Deep Neural Networks - Their Advantages, and Their Design by Forrest Iandola

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"

Most of these papers are reviewed by PR12 members

SqueezeNet

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

Forrest N. Iandola[1], Song Han[2], Matthew W. Moskewicz[1], Khalid Ashraf[1], William J. Dally[2], Kurt Keutzer[1]
[1]DeepScale* & UC Berkeley [2]Stanford 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× smaller than AlexNet).

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

SqueezeNet

- Architectural Design Strategies
  - Replace 3x3 filters with 1x1 filters
  - Decrease the number of input channels to 3x3 filters
    - Total quantity of parameters in 3x3 conv layer is (number of input channels) x (number of filters) x (3x3)
  - 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

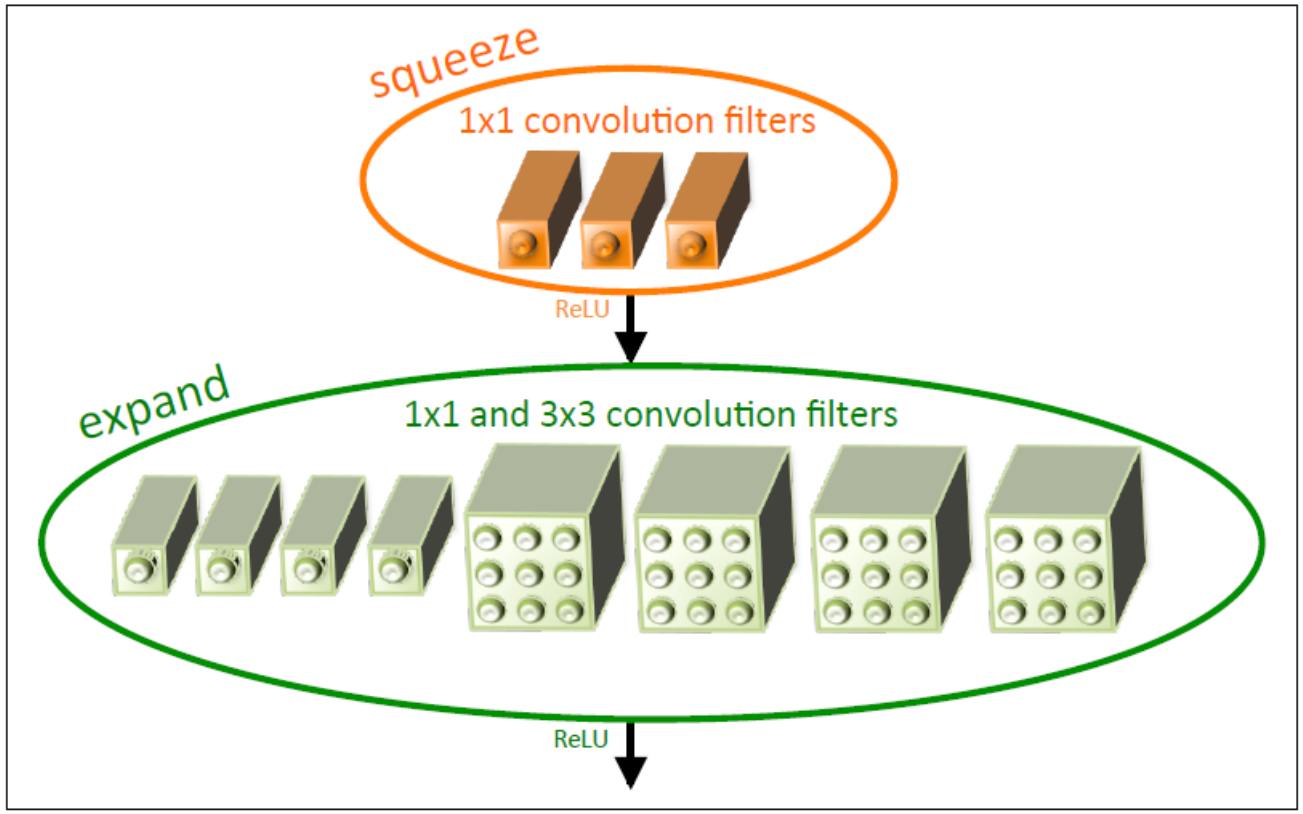
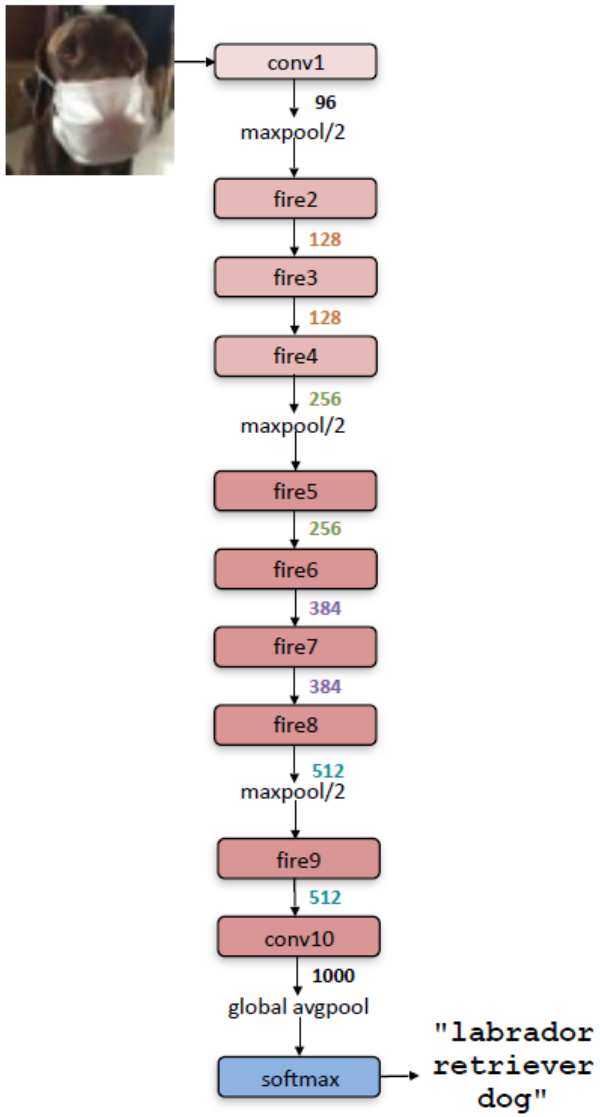
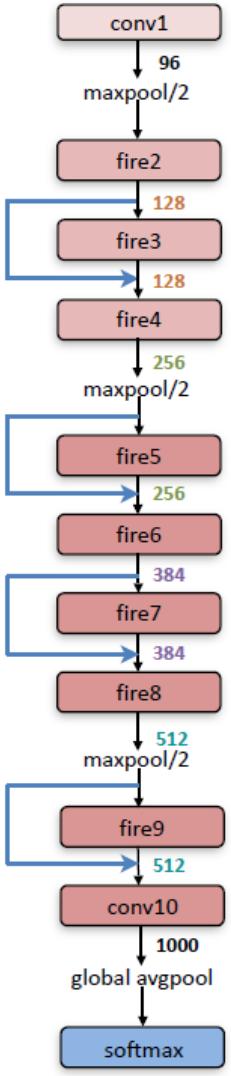
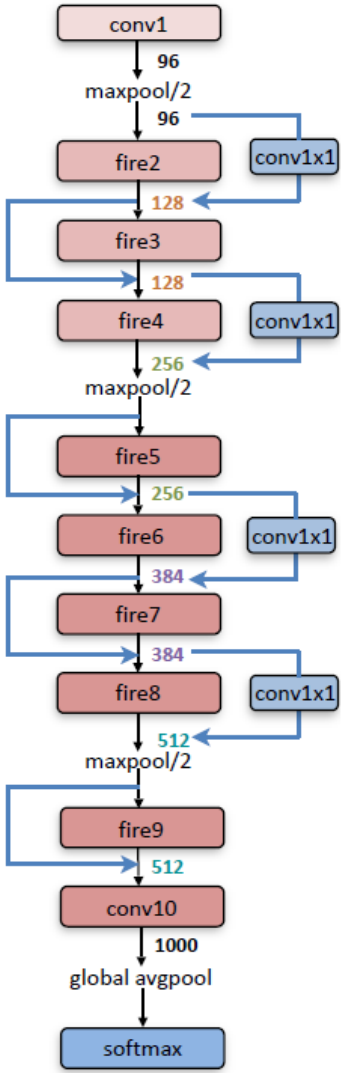


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

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

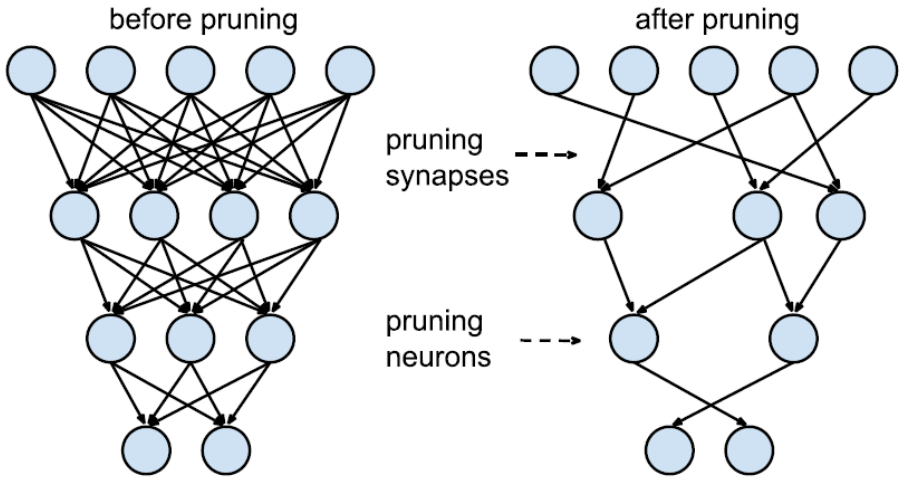
Table 2: Comparing SqueezeNet to model compression approaches. By *model size*, we mean the number of bytes required to store all of the parameters in the trained model.

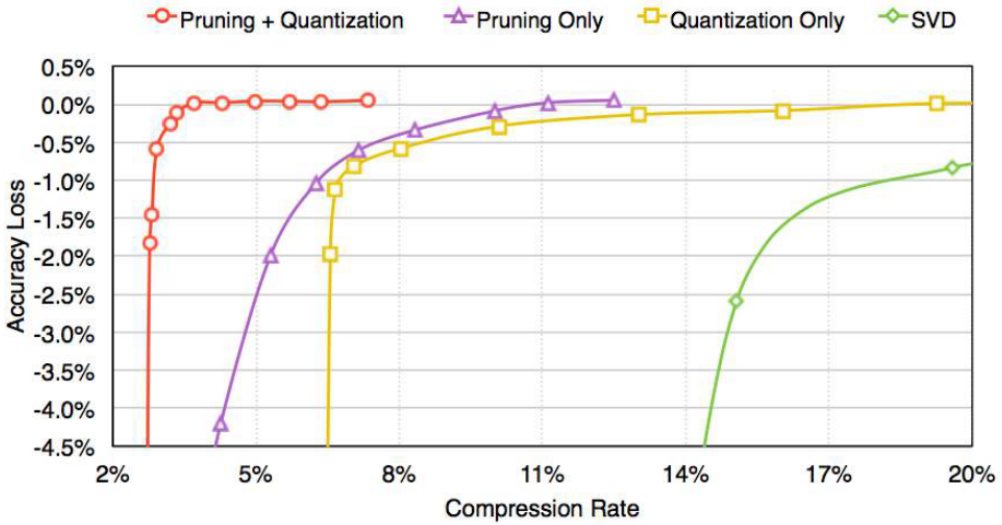
CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network 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

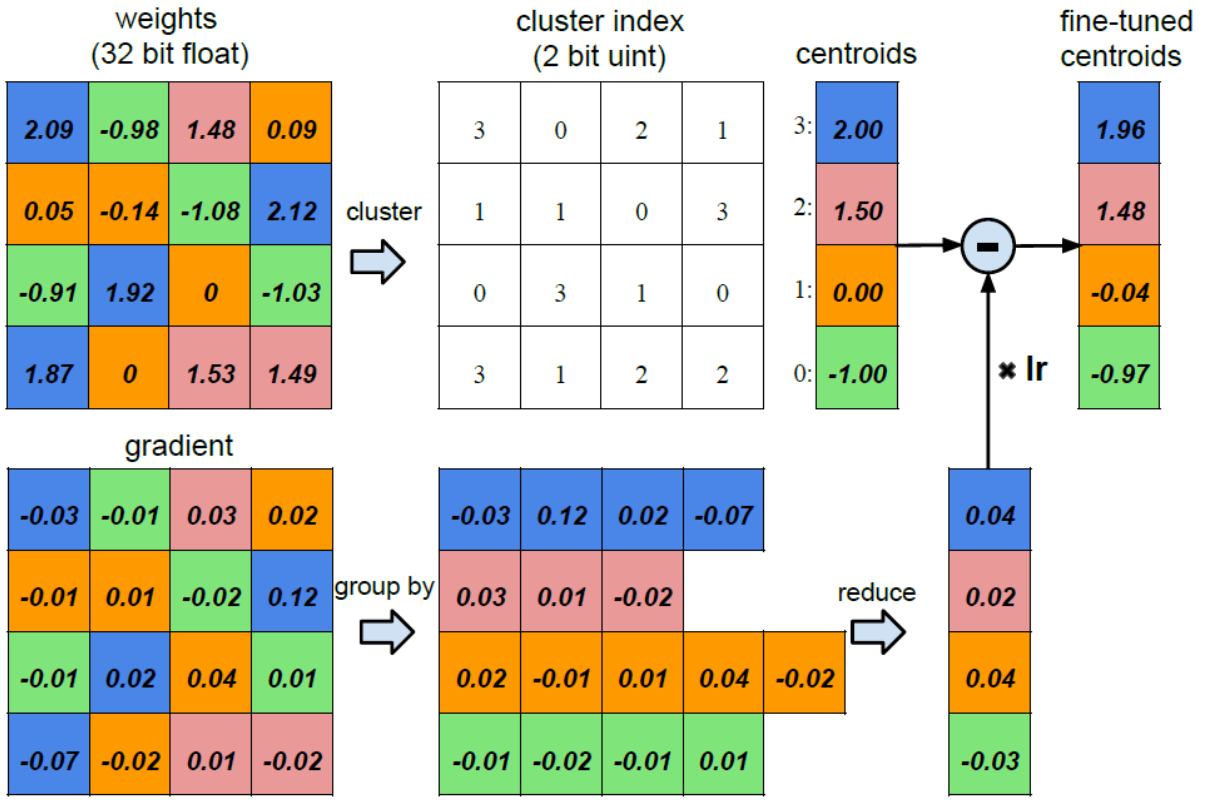
Architecture	Top-1 Accuracy	Top-5 Accuracy	Model Size
Vanilla SqueezeNet	57.5%	80.3%	4.8MB
SqueezeNet + Simple Bypass	**60.4%**	**82.5%**	4.8MB
SqueezeNet + Complex Bypass	58.8%	82.0%	7.7MB

Network Pruning & Deep Compression





AlexNet on ImageNet



Xception

Xception: Deep Learning with Depthwise Separable Convolutions

François Chollet
Google, Inc.
fchollet@google.com

Abstract

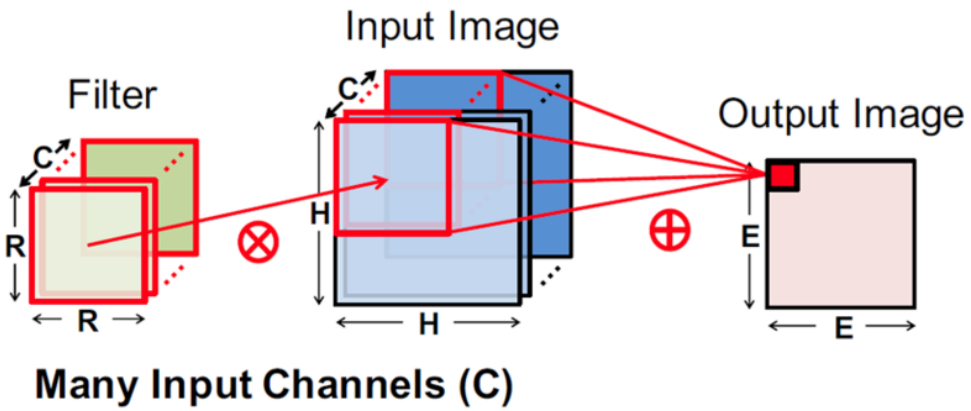
We present an interpretation of Inception modules in convolutional neural networks as being an intermediate step in-between regular convolution and the depthwise separable convolution operation (a depthwise convolution followed by a pointwise convolution). In this light, a depthwise separable convolution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

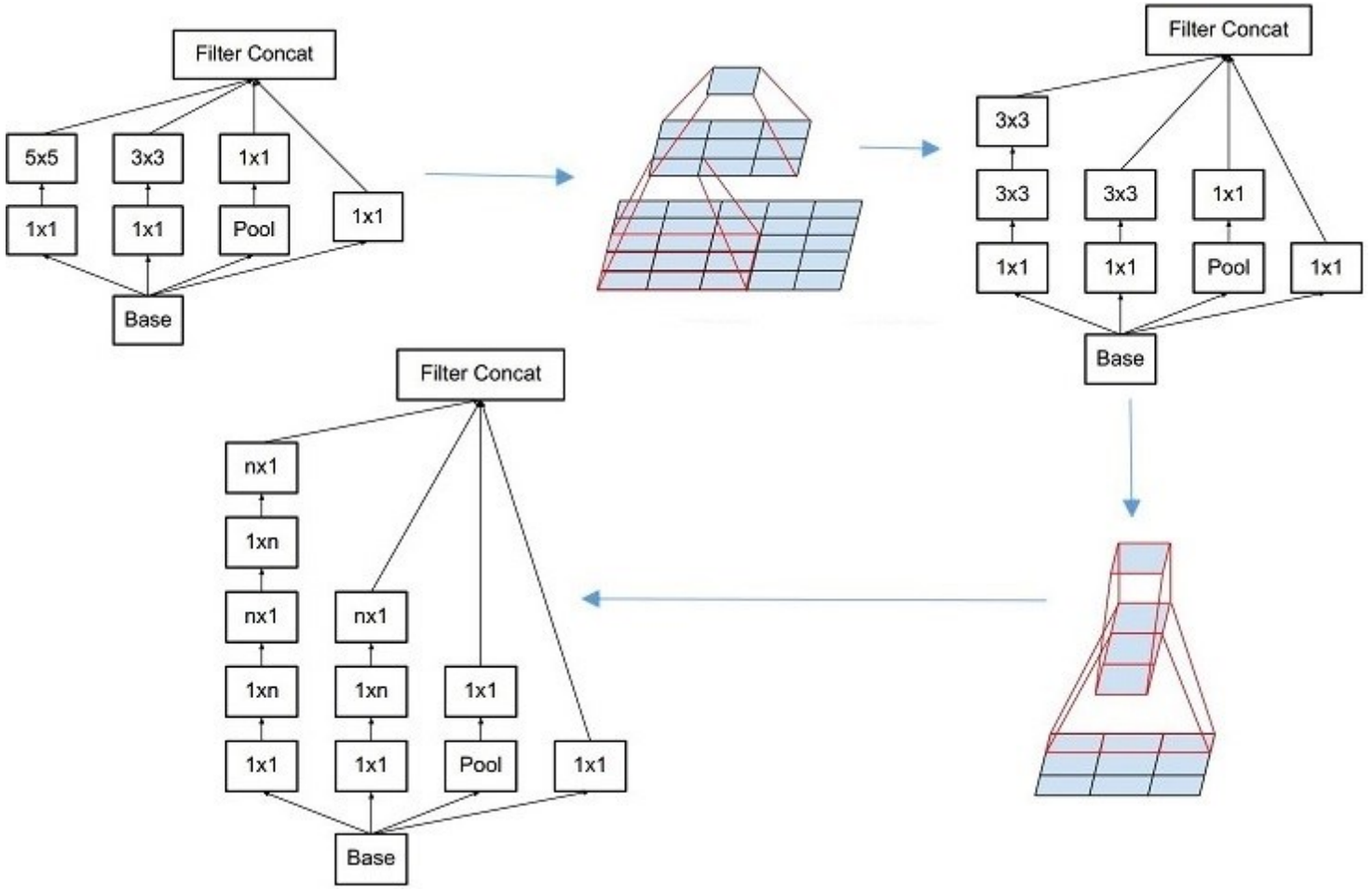
as GoogLeNet (Inception V1), later refined as Inception V2 [7], Inception V3 [21], and most recently Inception-ResNet [19]. Inception itself was inspired by the earlier Network-In-Network architecture [11]. Since its first introduction, Inception has been one of the best performing family of models on 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).

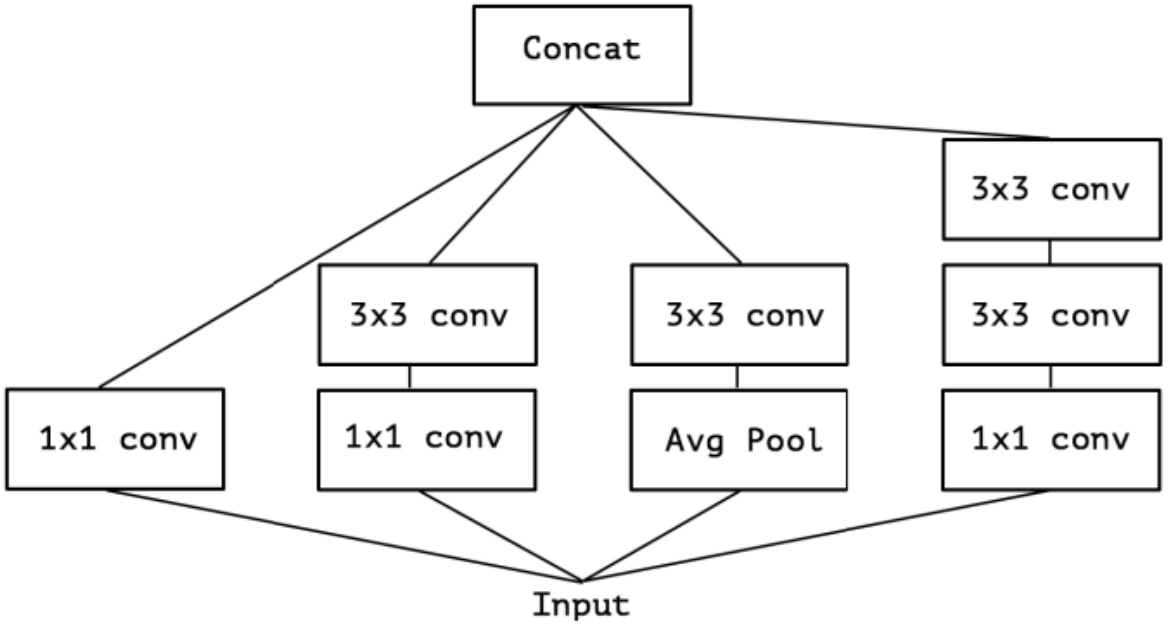
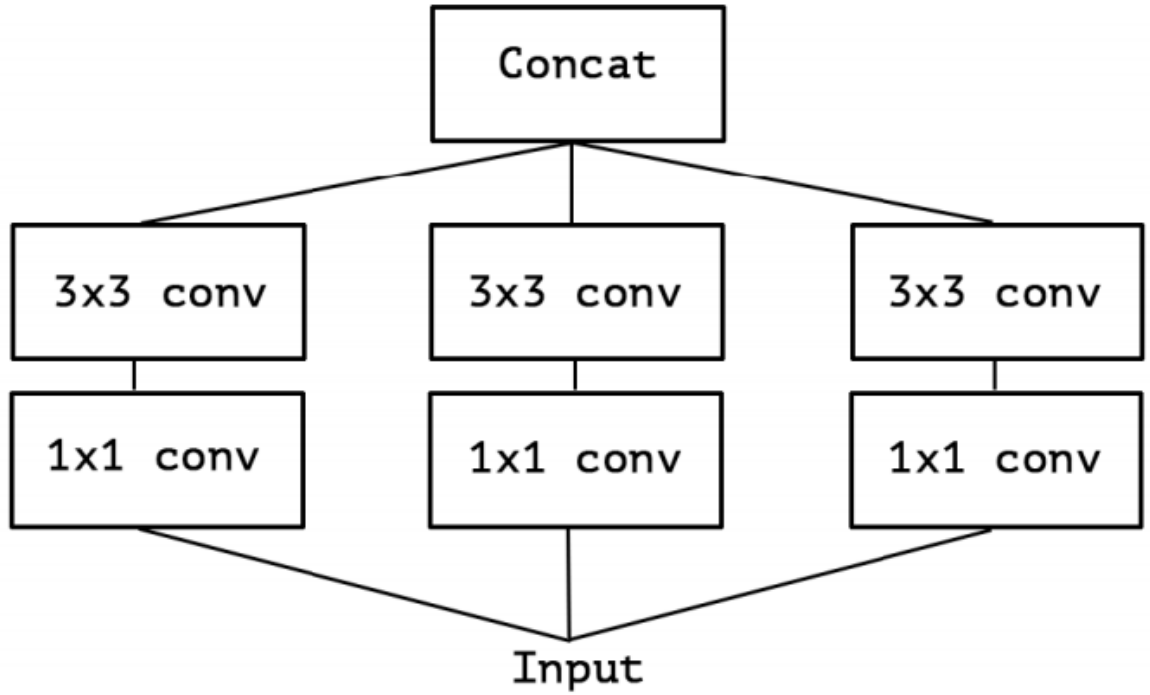


Figure 2. A simplified Inception module.



Equivalent Reformulation

Figure 2. A simplified Inception module.

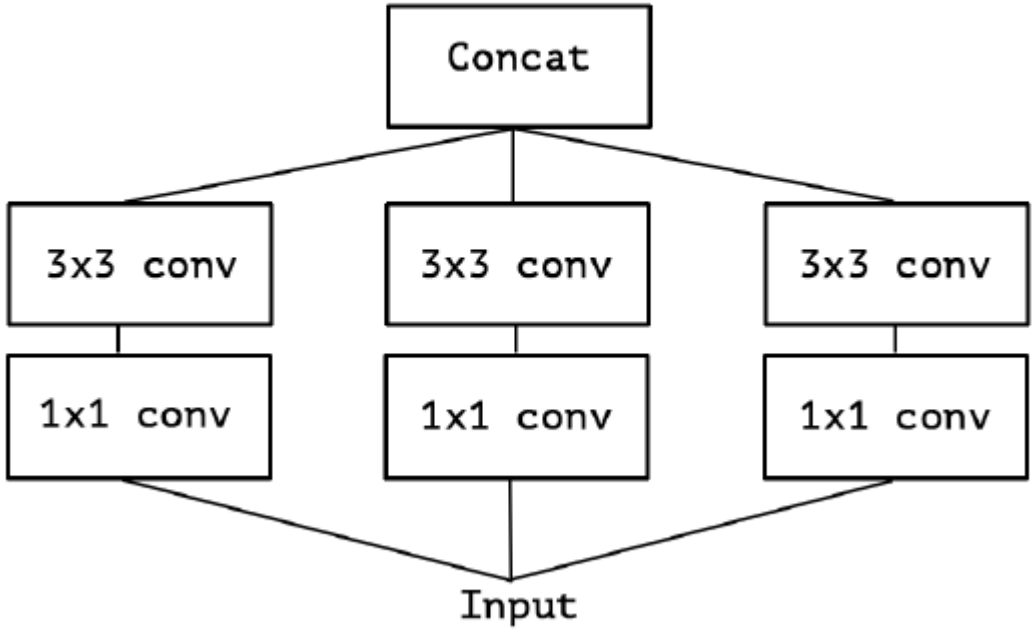
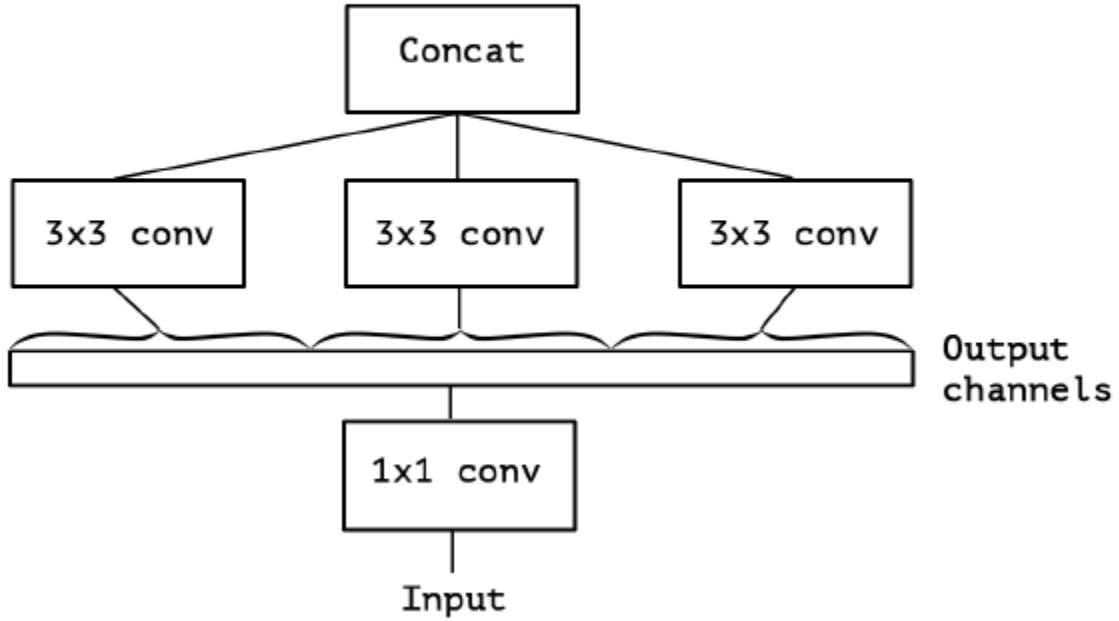
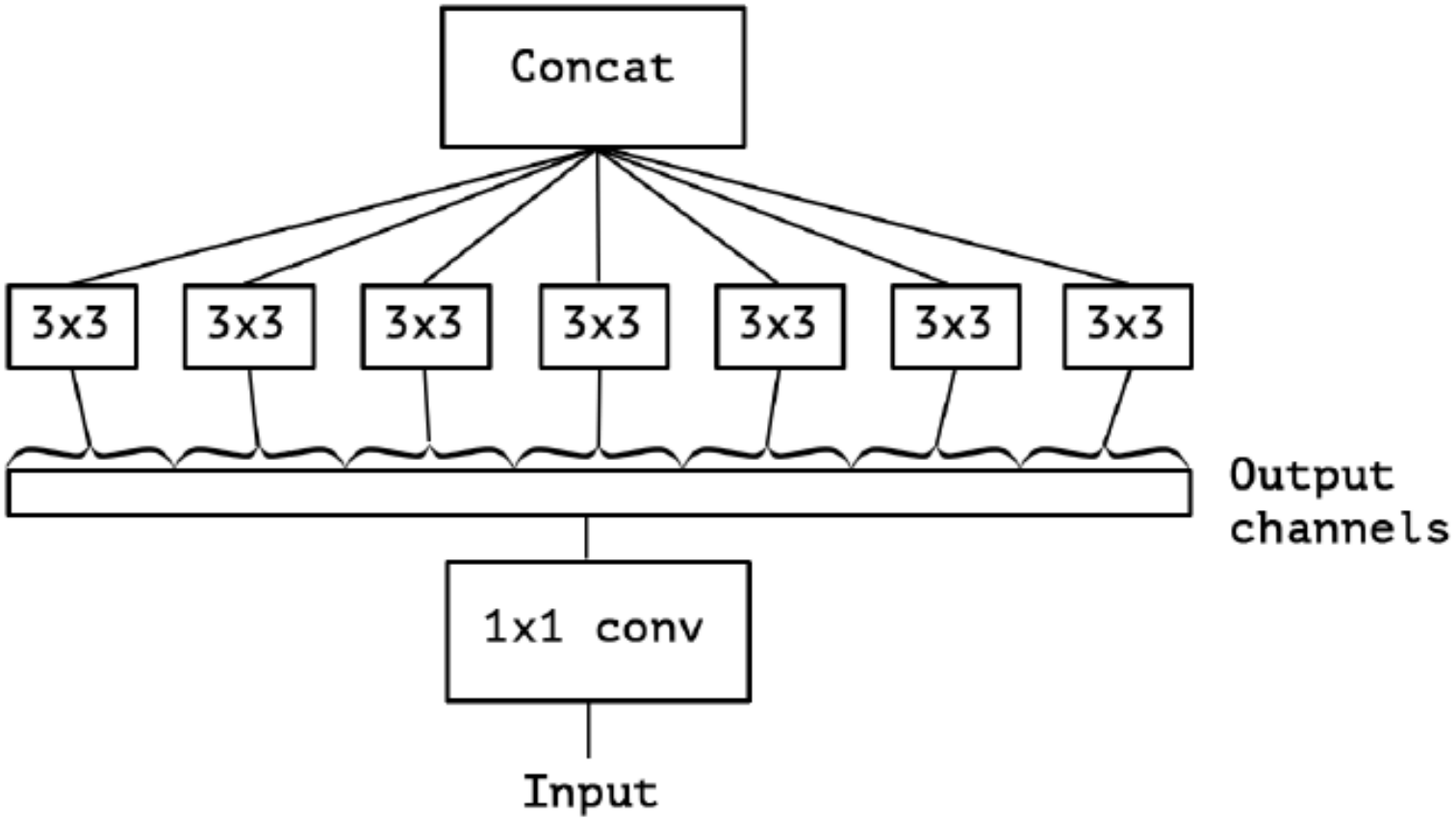


Figure 3. A strictly equivalent reformulation of the simplified Inception module.



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 Menglong Zhu Bo Chen Dmitry Kalenichenko
Weijun Wang Tobias Weyand Marco Andreetto 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 depth-wise separable convolutions to build light weight deep neural networks. We introduce two simple global hyper-parameters 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

MobileNets

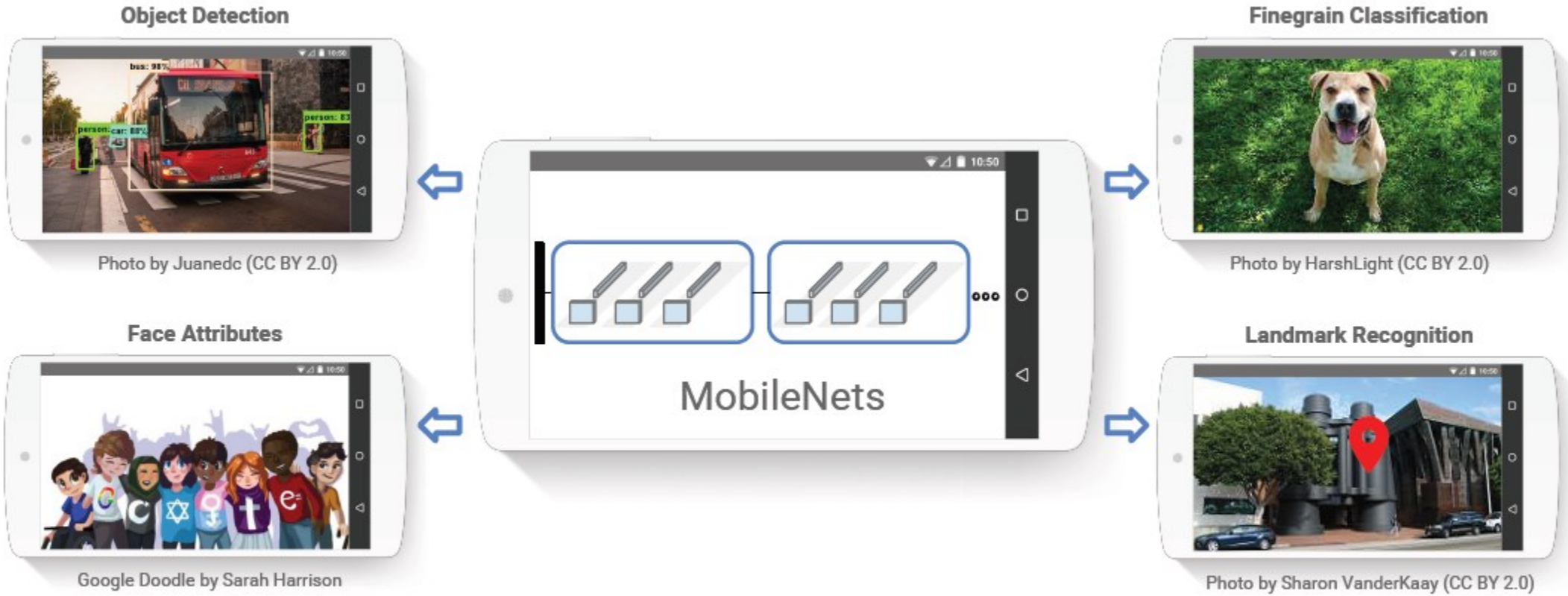
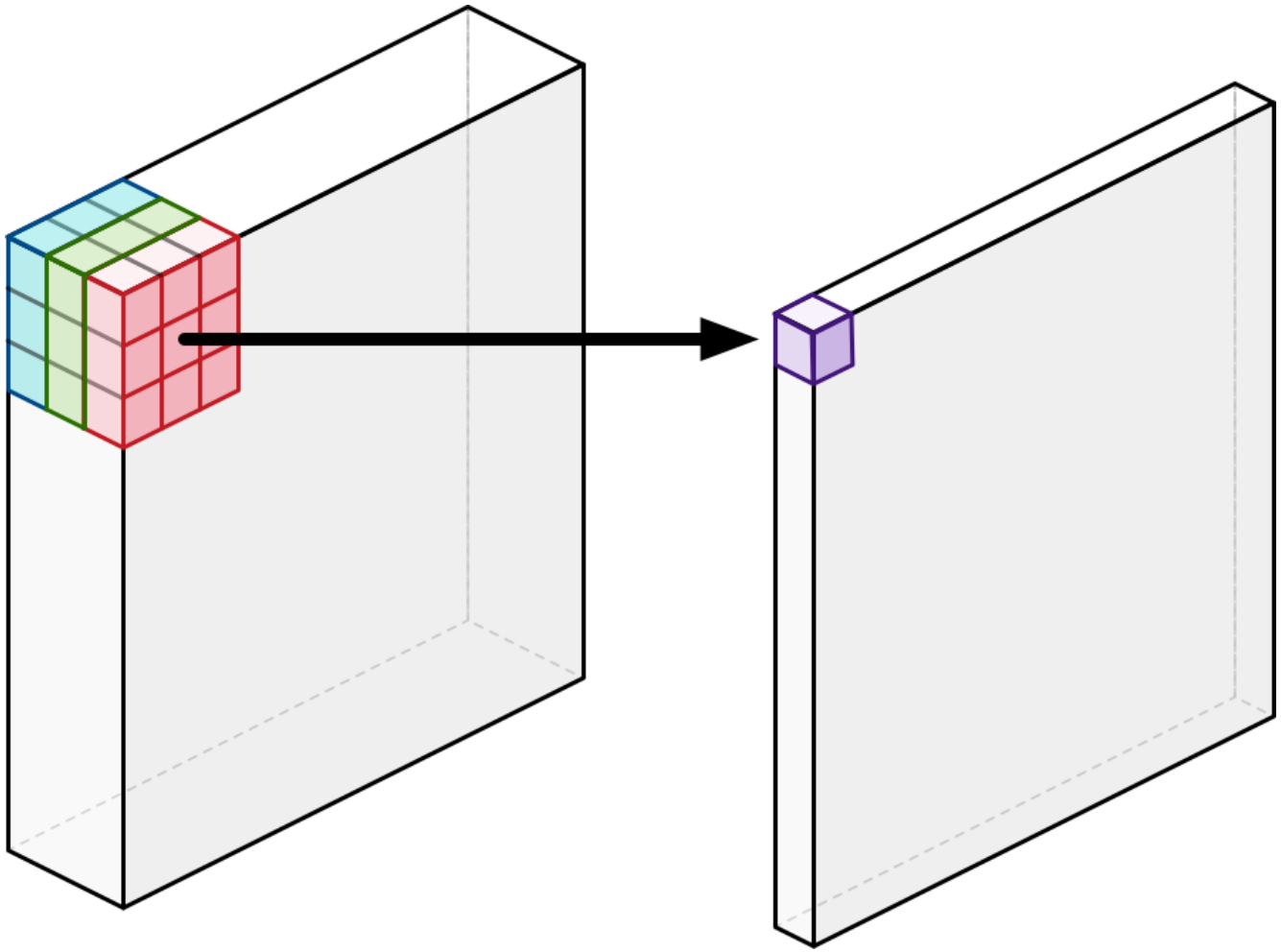


Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

Standard Convolution

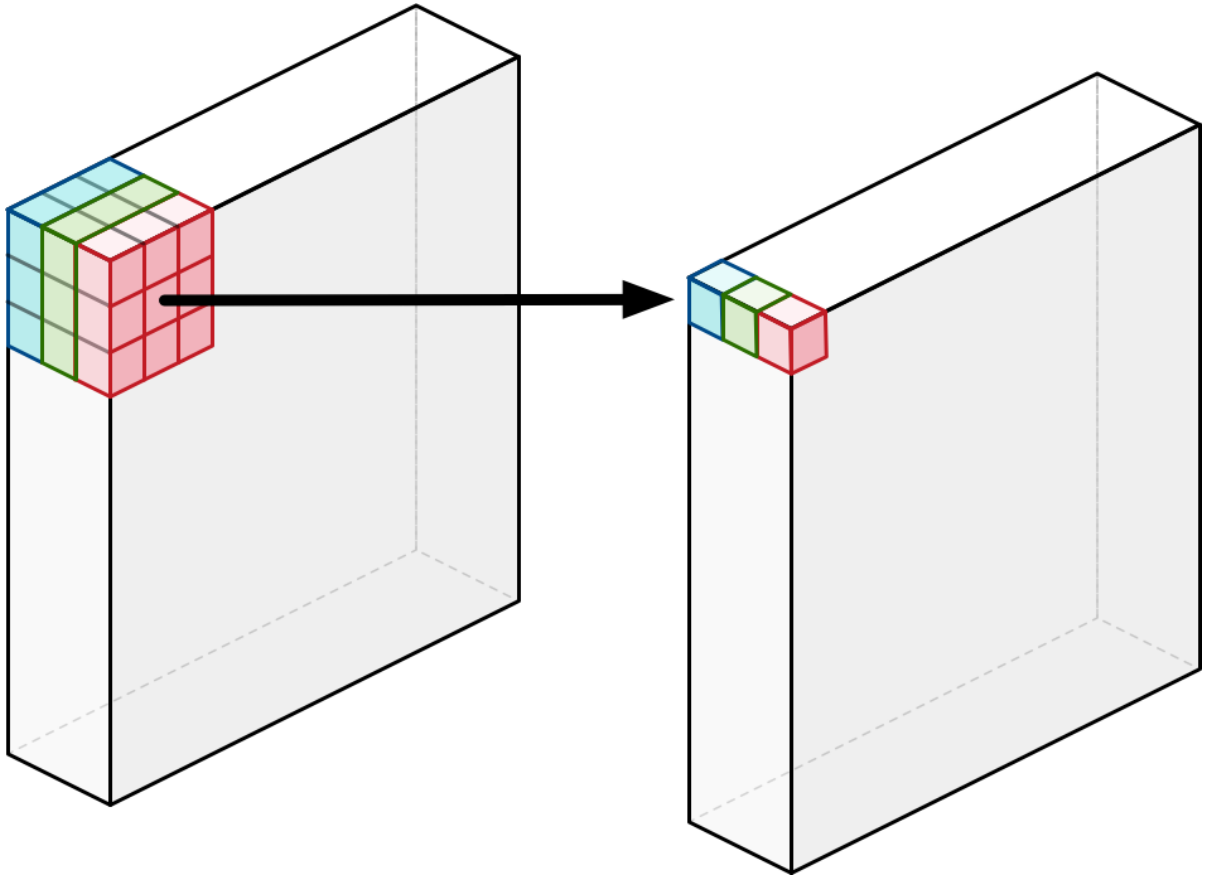


Standard convolution

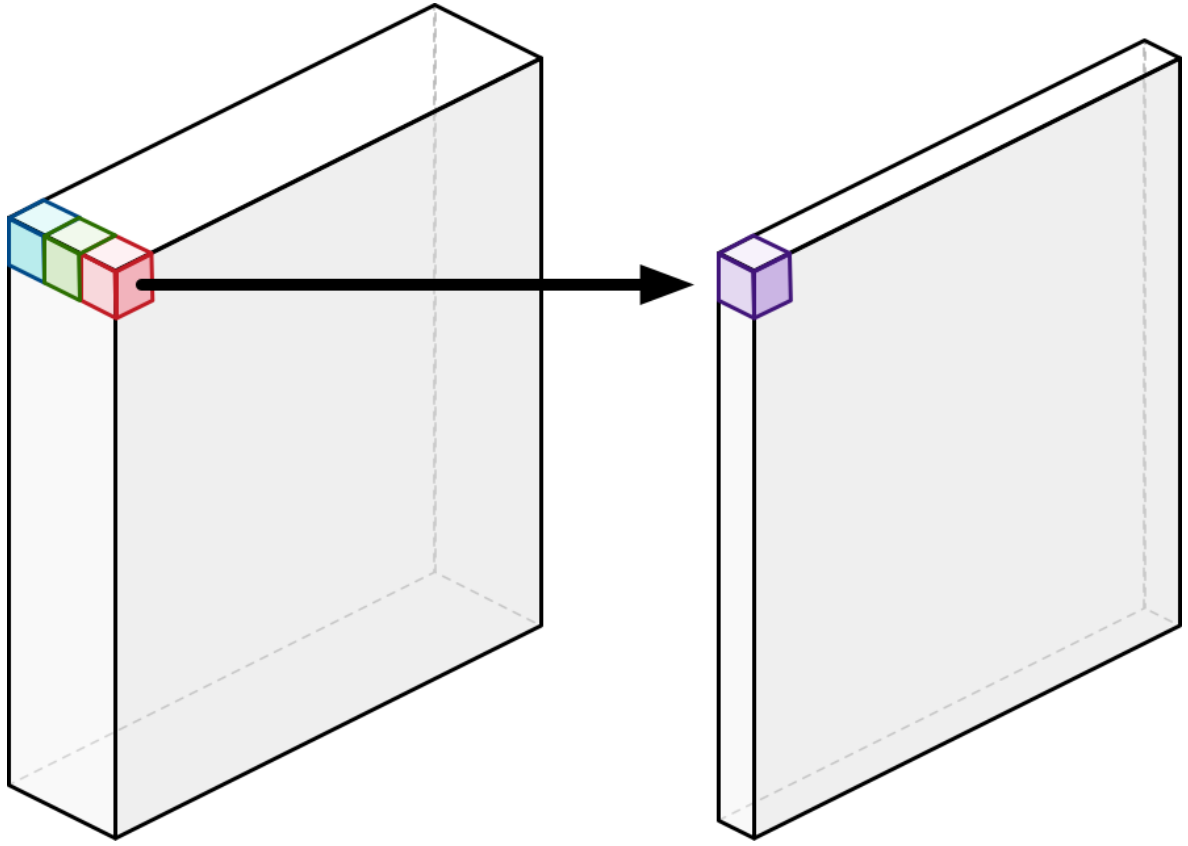
Figures from http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

Depthwise Separable Convolution

- Depthwise Convolution + Pointwise Convolution(1x1 convolution)



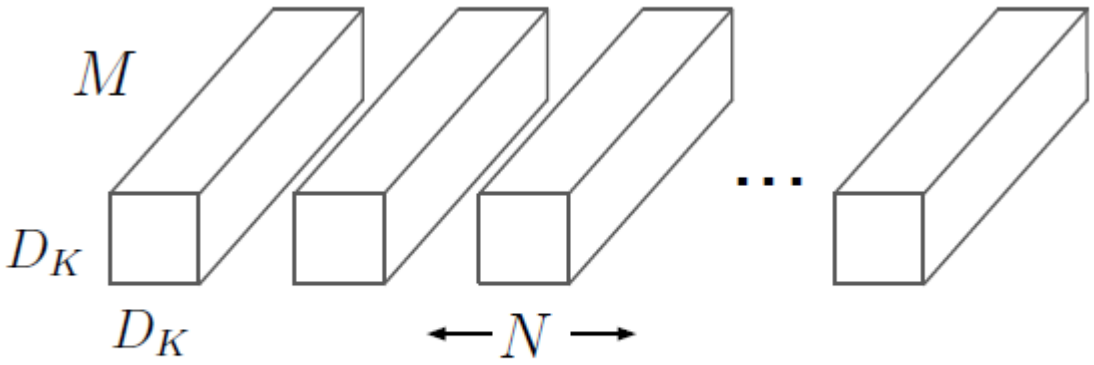
Depthwise convolution



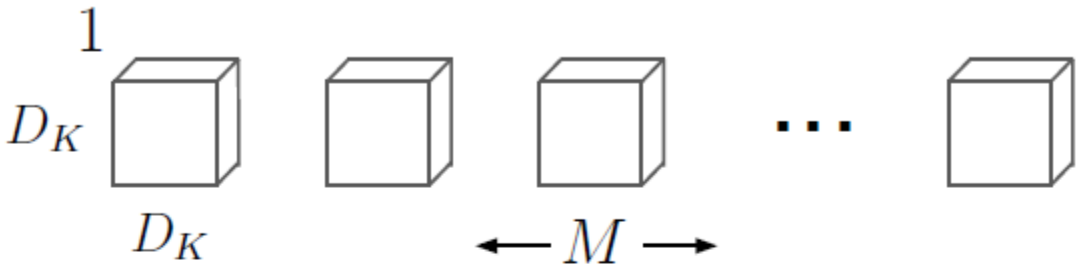
Pointwise convolution

Figures from http://machinethink.net/blog/googles-mobile-net-architecture-on-iphone/

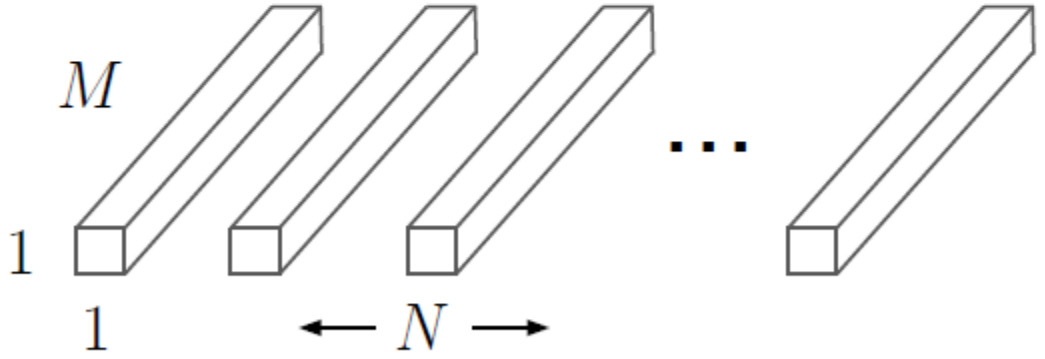
Standard Convolution vs Depthwise Separable Convolution



(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 3x3 depthwise separable convolutions, we get between 8 to 9 times less computations

Depthwise Separable Convolutions

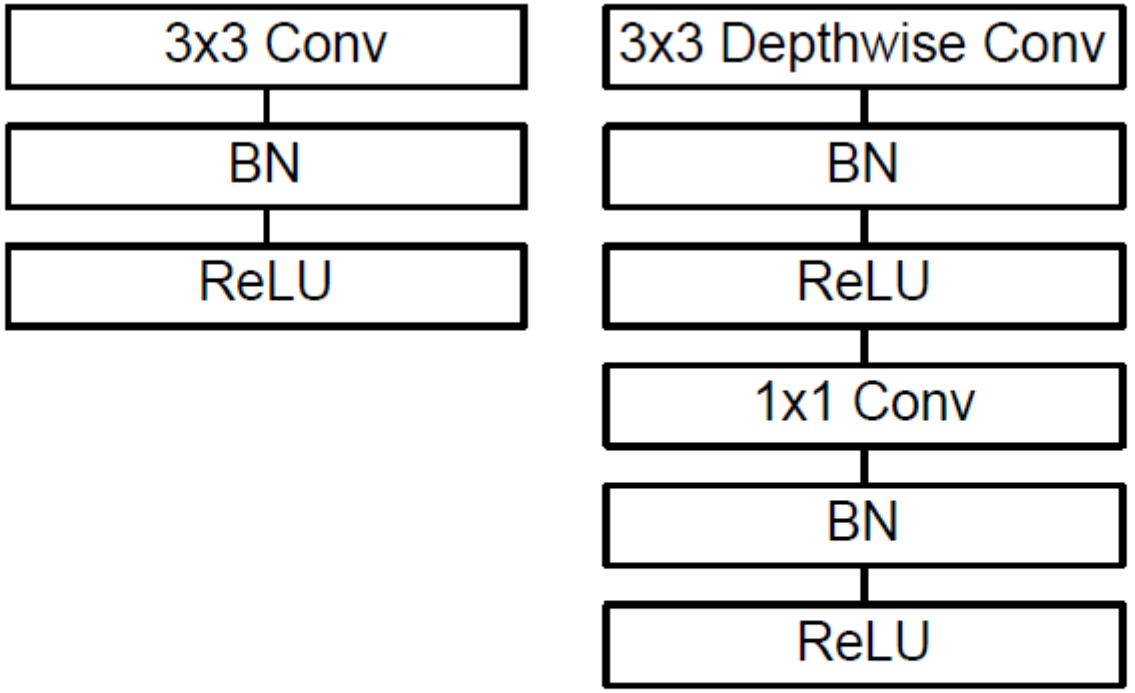


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

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	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Table 2. Resource Per Layer Type

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

Width Multiplier & Resolution Multiplier

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

$$D_K \times D_K \times \alpha M \times \rho D_F \times \rho D_F + \alpha M \times \alpha N \times \rho D_F \times \rho D_F$$

Width Multiplier & Resolution Multiplier

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

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

Experiments – Model Choices

Table 4. Depthwise Separable vs Full Convolution MobileNet

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

Table 5. Narrow vs Shallow MobileNet

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

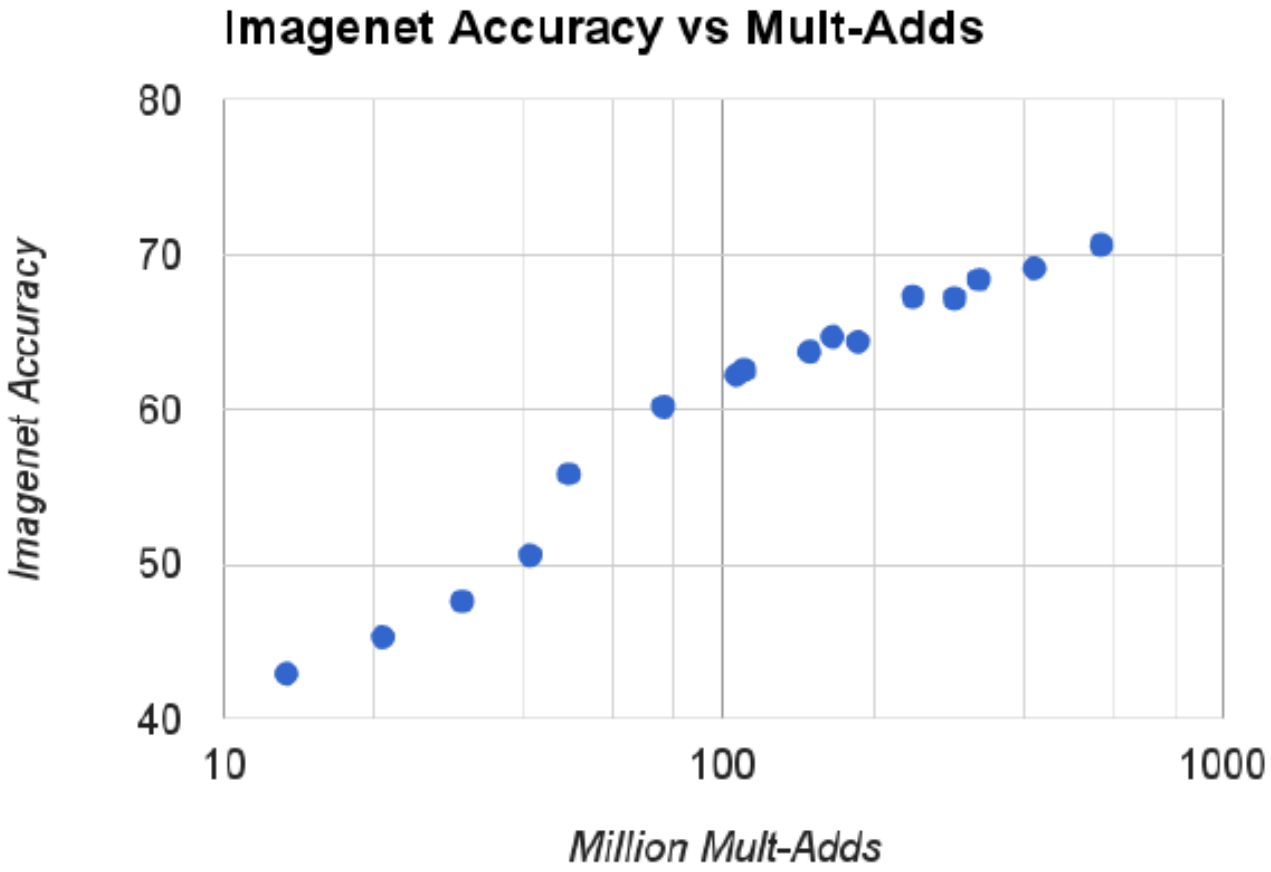
Table 6. MobileNet Width Multiplier

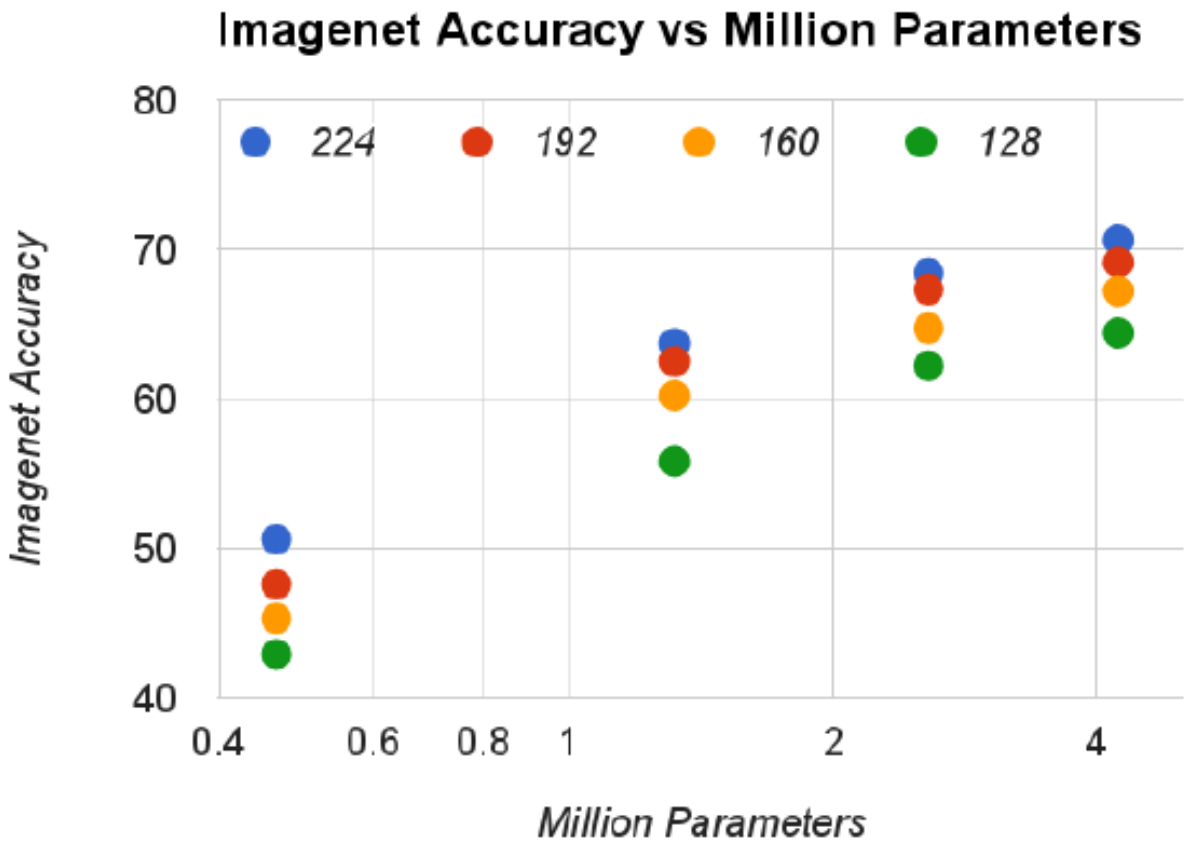
Width Multiplier	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
0.75 MobileNet-224	68.4%	325	2.6
0.5 MobileNet-224	63.7%	149	1.3
0.25 MobileNet-224	50.6%	41	0.5

Table 7. MobileNet Resolution

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

Model Shrinking Hyperparameters





Results

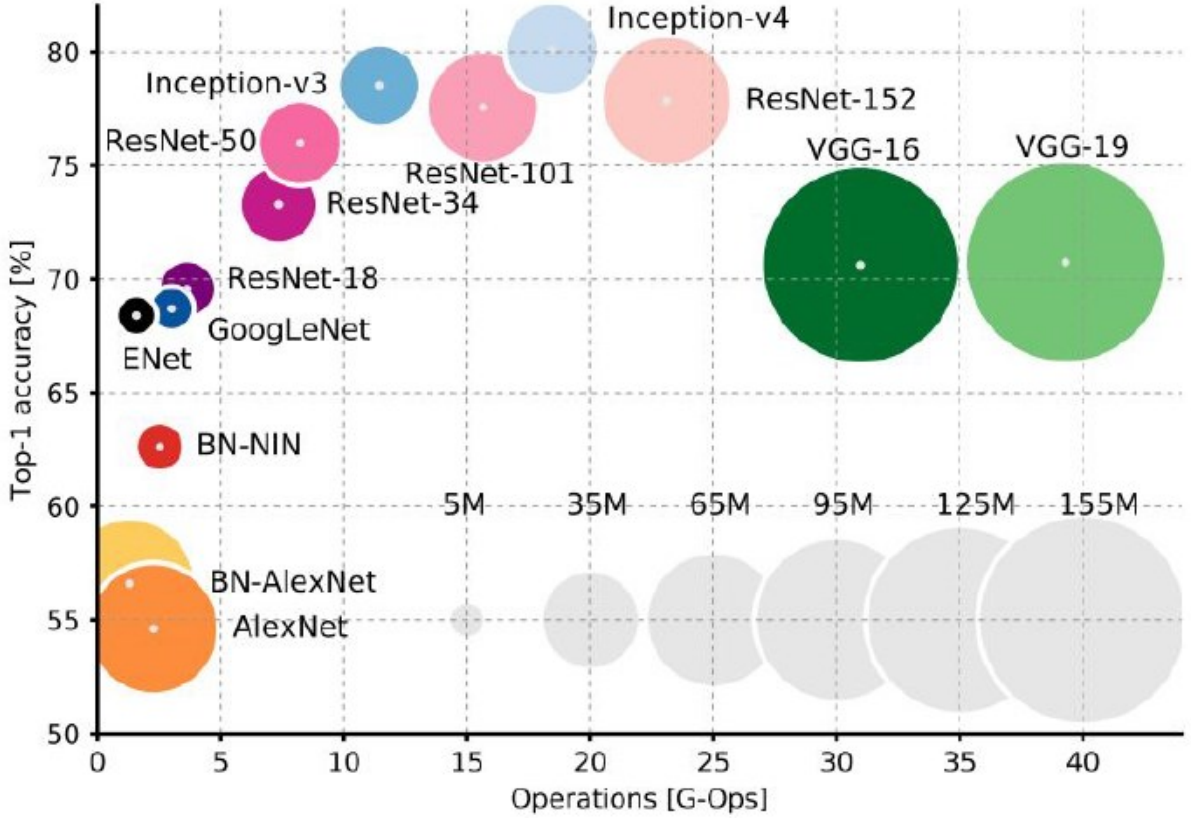
Table 8. MobileNet Comparison to Popular Models

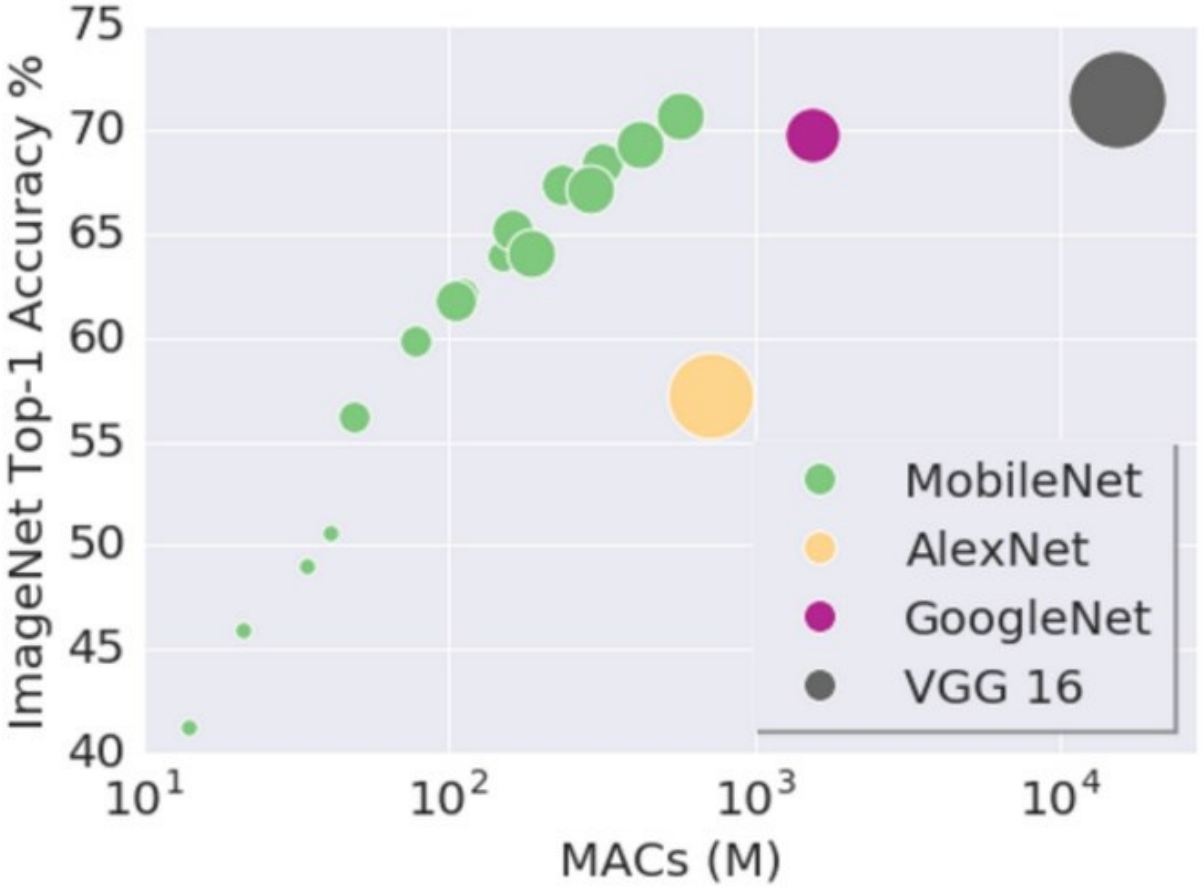
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

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 ~13× 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

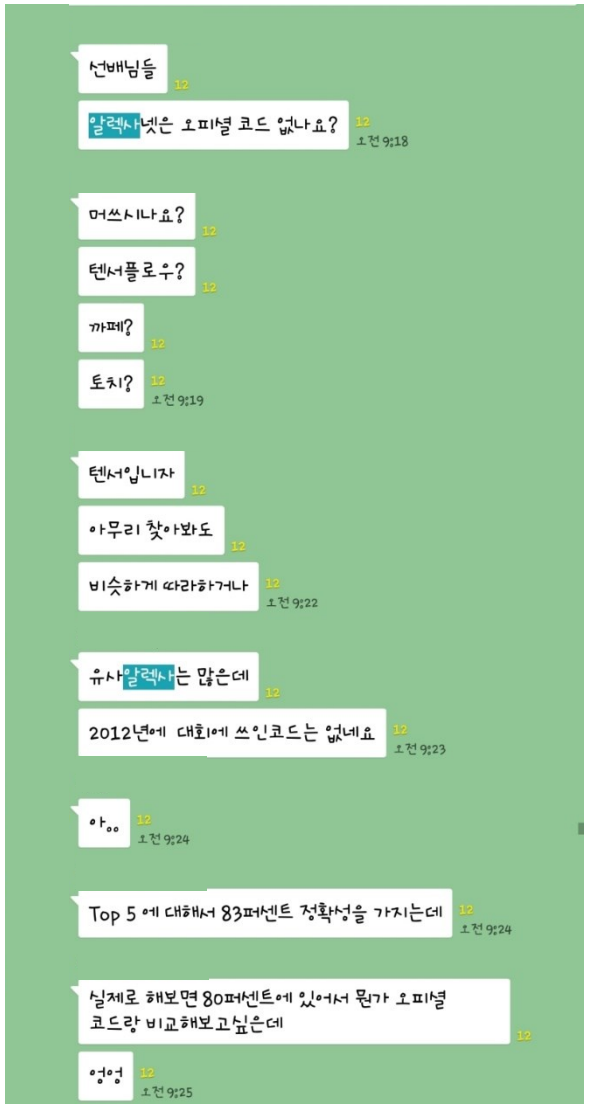
- The 1x1 convolution accounts for most of the computation

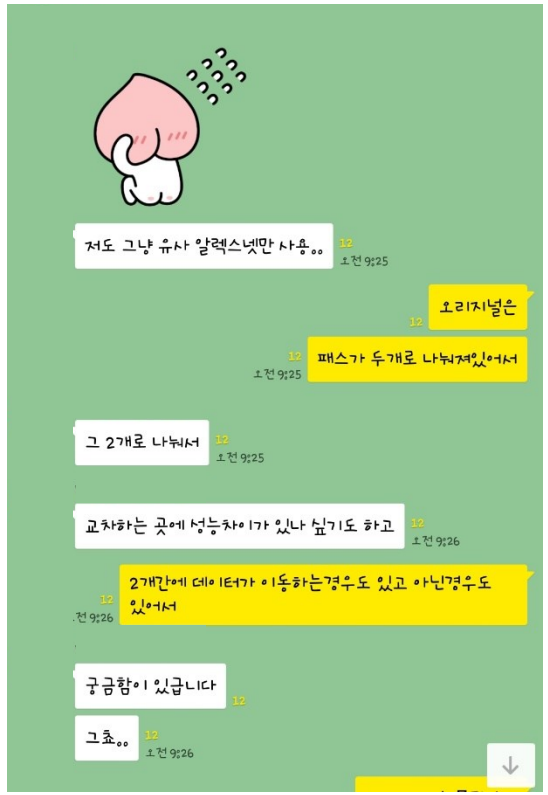
Table 2. Resource Per Layer Type

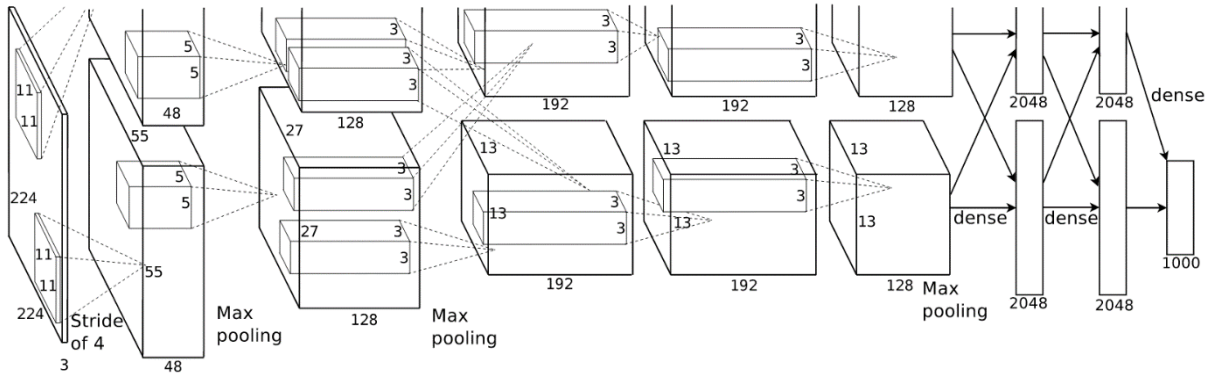
Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

- Can we reduce more?

A Secret of AlexNet



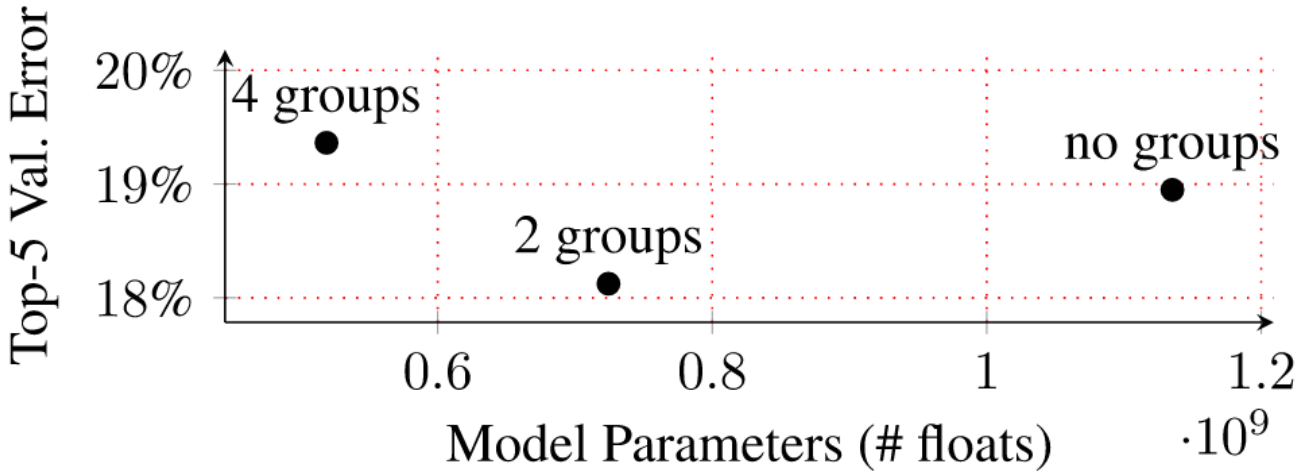




Grouped Convolution!

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!



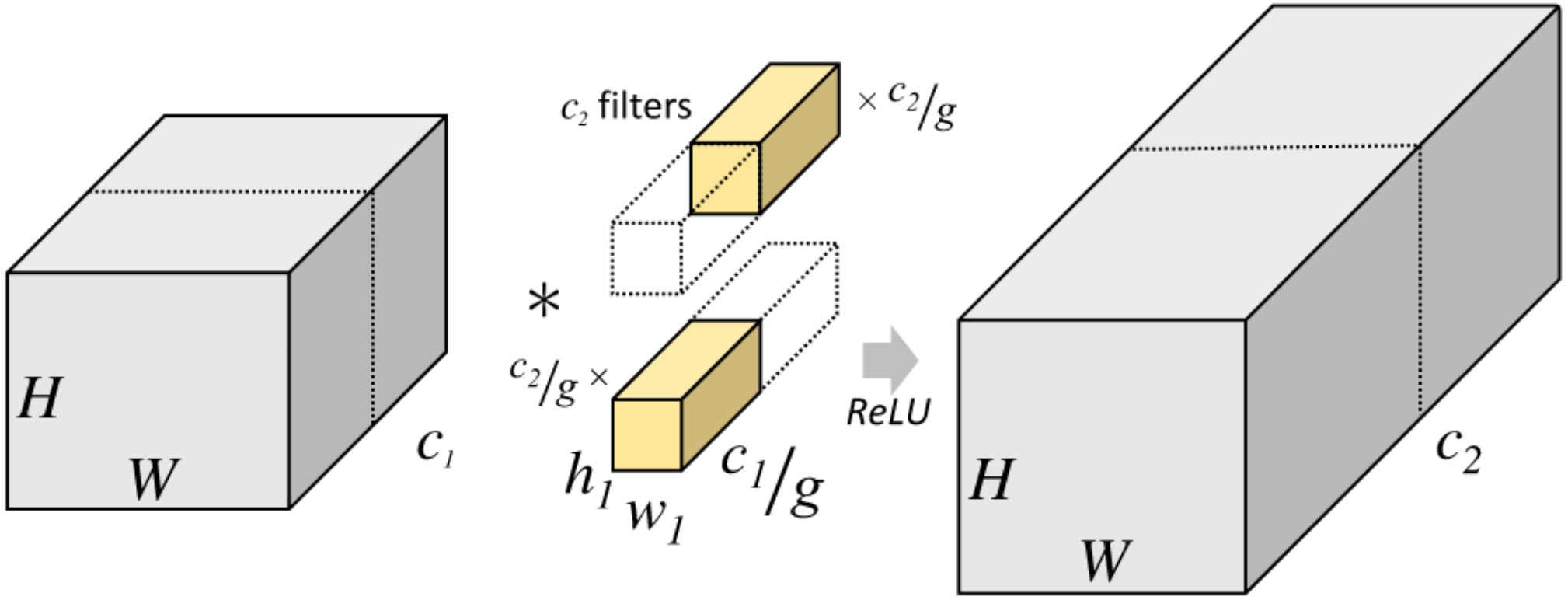
AlexNet trained with varying numbers of filter groups, from 1 (i.e. no filter groups), to 4. When trained with 2 filter groups, AlexNet is more efficient and yet achieves the same if not lower validation error.

Figure from https://blog.yani.io/filter-group-tutorial/

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

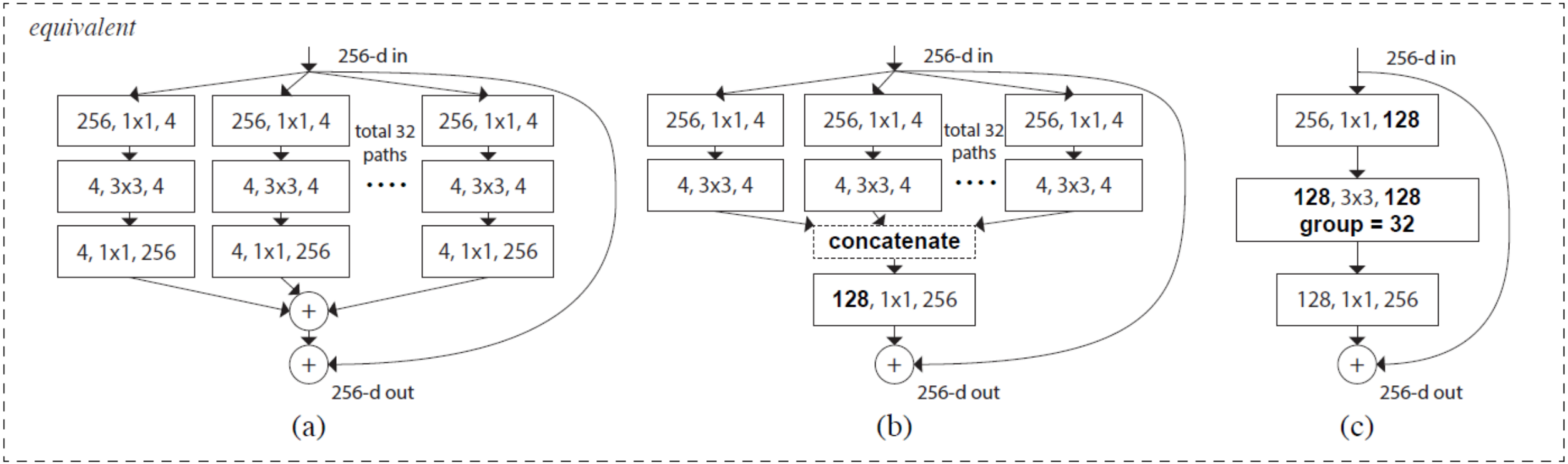


A convolutional layer with 2 filter groups. Note that each of the filters in the grouped convolutional layer is now exactly half the depth, i.e. half the parameters and half the compute as the original filter.

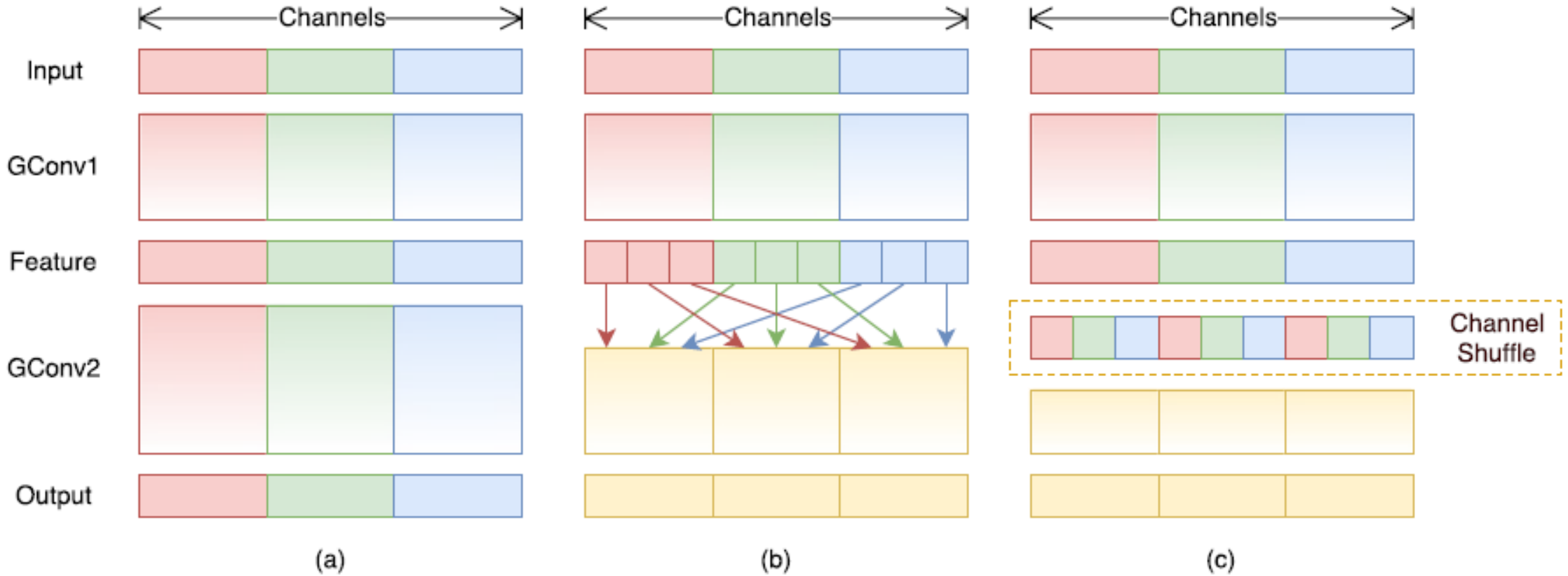
Figure from https://blog.yani.io/filter-group-tutorial/

ResNeXt

- Equivalent building blocks of RexNext (cardinality=32)



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 outputchannels will be fully related.

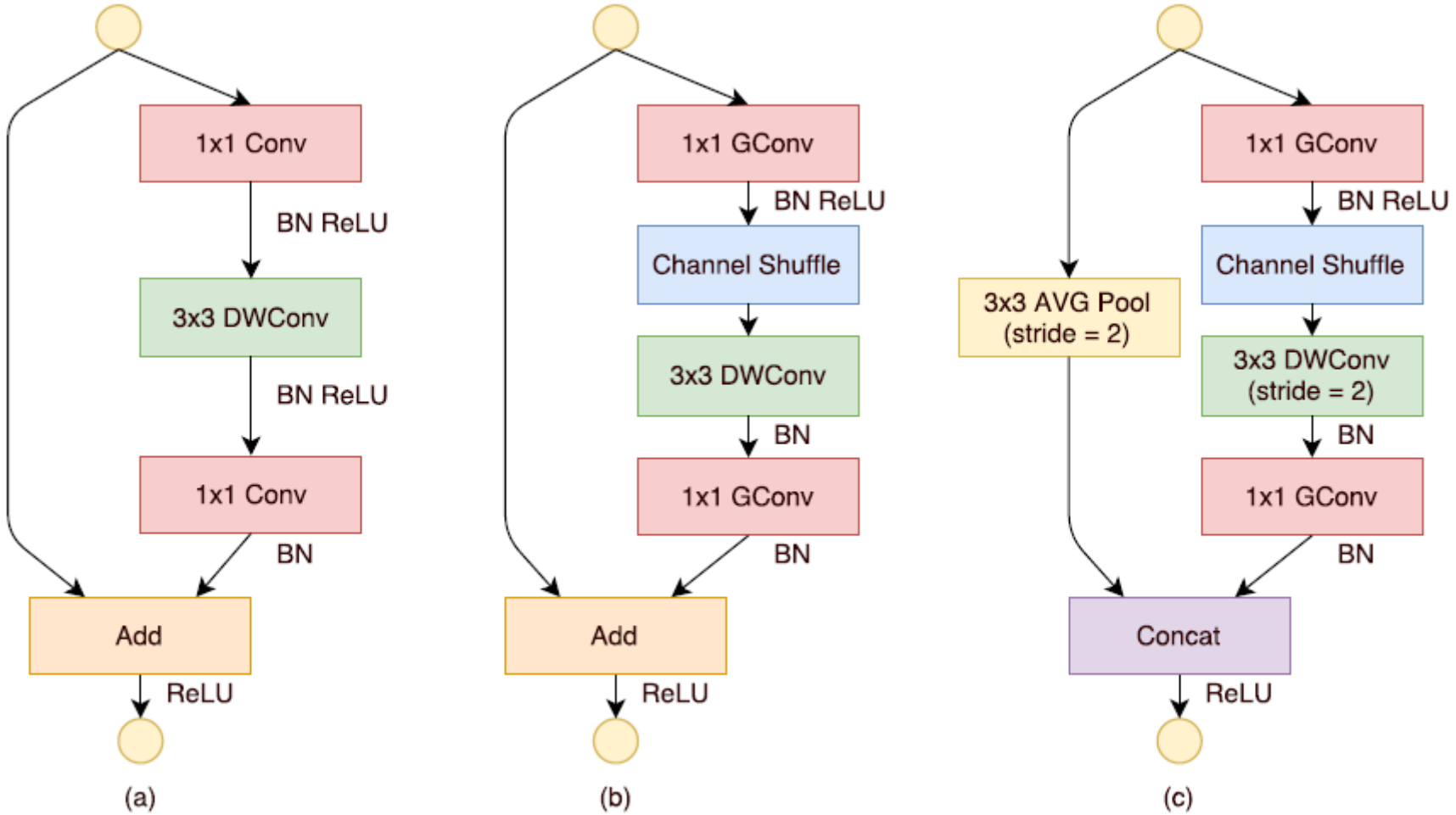
Channel Shuffle Operation

- Suppose a convolutional layer with g groups whose output has g x 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

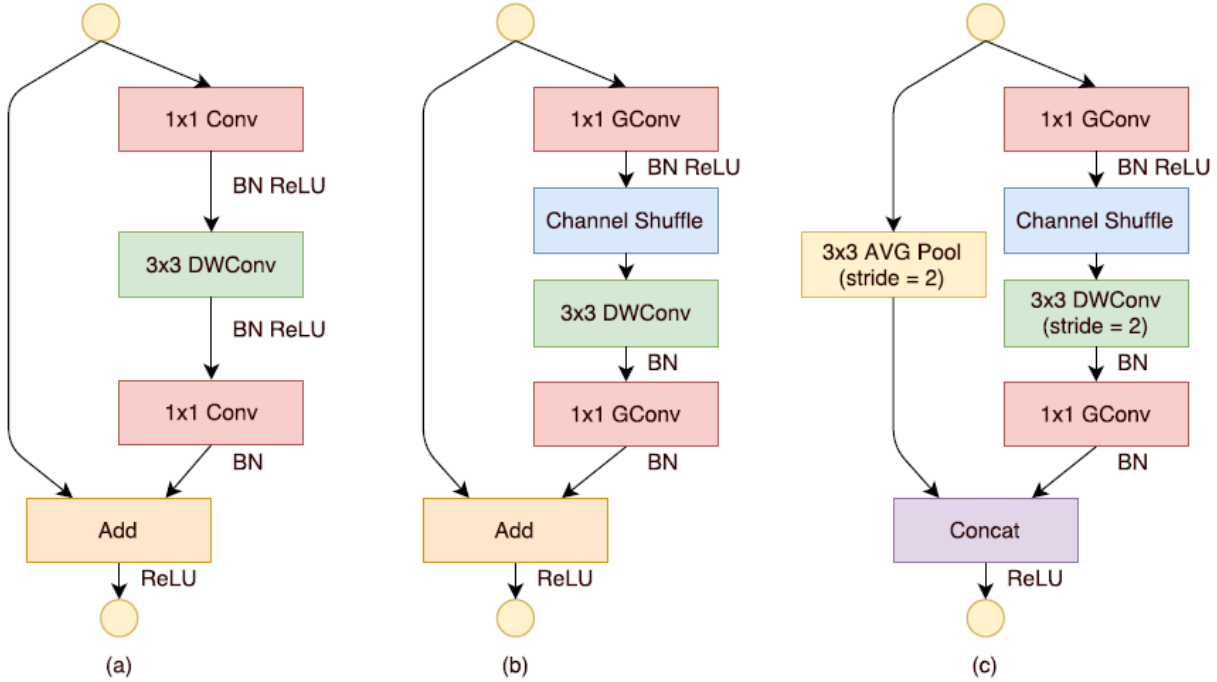
```
def channel_shuffle(name, x, num_groups):
    with tf.variable_scope(name) as scope:
        n, h, w, c = x.shape.as_list()
        x_reshaped = tf.reshape(x, [-1, h, w, num_groups, c // num_groups])
        x_transposed = tf.transpose(x_reshaped, [0, 1, 2, 4, 3])
        output = tf.reshape(x_transposed,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

Tensorflow Implementation from https://github.com/MG2033/ShuffleNet

ShuffleNet Units



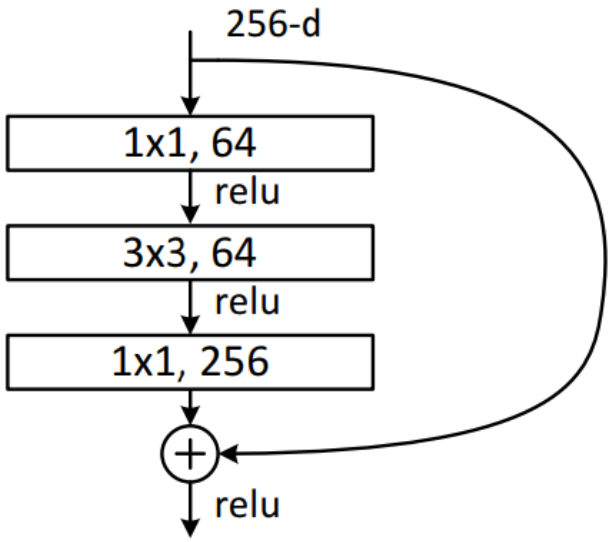
ShuffleNet Units

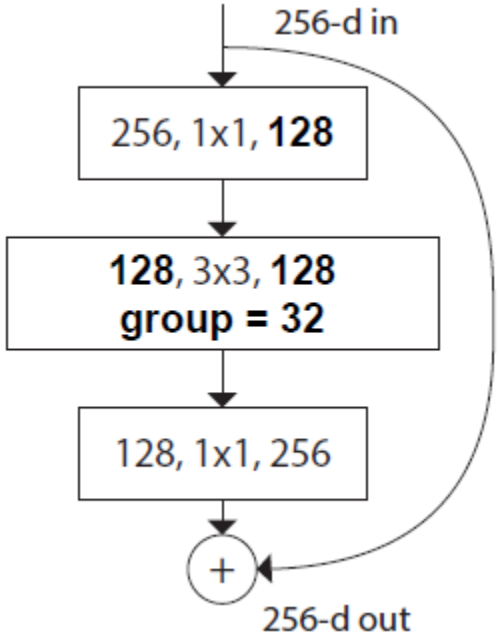


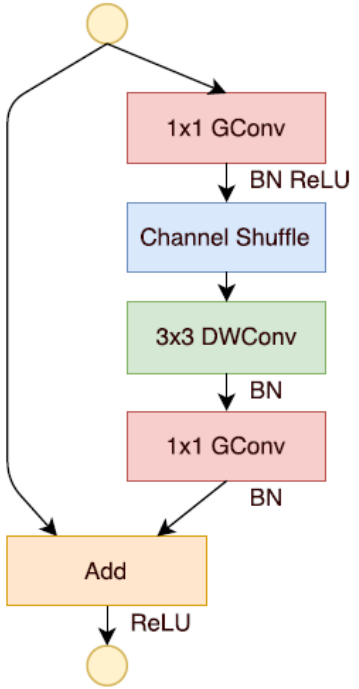
- From (a), replace the first 1x1 layer with pointwise group convolution followed by a channel shuffle operation
- ReLU is not applied to 3x3 DWConv
- As for the case where ShuffleNet is applied with stride, simply make to modifications
  - Add 3x3 average pooling on the shortcut path
  - Replace element-wise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

Complexity

- For example, given the input size c x h x w and the bottleneck channel m







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

<Number of Operations>

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× means scaling the number of filters in ShuffleNet 1× by s times thus overall complexity will be roughly s^2 times of ShuffleNet 1×
- 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×	140	33.6	32.7	32.6	32.8	**32.4**
ShuffleNet 0.5×	38	45.1	44.4	43.2	**41.6**	42.3
ShuffleNet 0.25×	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× ($g = 3$)	524	**26.3**	3.1
ShuffleNet 2× (with *SE*[13], $g = 3$)	527	**24.7**	4.7
0.75 MobileNet-224	325	31.6	-
ShuffleNet 1.5× ($g = 3$)	292	**28.5**	3.1
0.5 MobileNet-224	149	36.3	-
ShuffleNet 1× ($g = 8$)	140	**32.4**	3.9
0.25 MobileNet-224	41	49.4	-
ShuffleNet 0.5× ($g = 4$)	38	**41.6**	7.8
ShuffleNet 0.5× (shallow, $g = 3$)	40	42.8	6.6

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

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

MobileNet V2

MobileNetV2: Inverted Residuals and Linear Bottlenecks

Mark Sandler Andrew Howard Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen
Google Inc.
{sandler, howarda, menglong, azhmogin, lcchen}@google.com

Abstract

In this paper we describe a new mobile architecture, MobileNetV2, that improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. We also describe efficient ways of applying these mobile models to object detection in a novel framework we call SSDLite. Additionally, we demonstrate how to build mobile semantic segmentation models through a reduced form of DeepLabv3 which we call Mobile DeepLabv3.

is based on an inverted residual structure where the shortcut connections are between the thin bottleneck layers. The intermediate expansion layer uses lightweight depthwise convolutions to filter features as

applications.

This paper introduces a new neural network architecture that is specifically tailored for mobile and resource constrained environments. Our network pushes the state of the art for mobile tailored computer vision models, by significantly decreasing the number of operations and memory needed while retaining the same accuracy.

Our main contribution is a novel layer module: the inverted residual with linear bottleneck. This module takes as an input a low-dimensional compressed representation which is first expanded to high dimension and filtered with a lightweight depthwise convolution. Features are subsequently projected back to a low-dimensional representation with a *linear convolution*. The official implementation is available as part of TensorFlow-Slim model library in [4].

Key Features

- Depthwise Separable Convolutions
- Linear Bottlenecks
- Inverted Residuals

Linear Bottlenecks

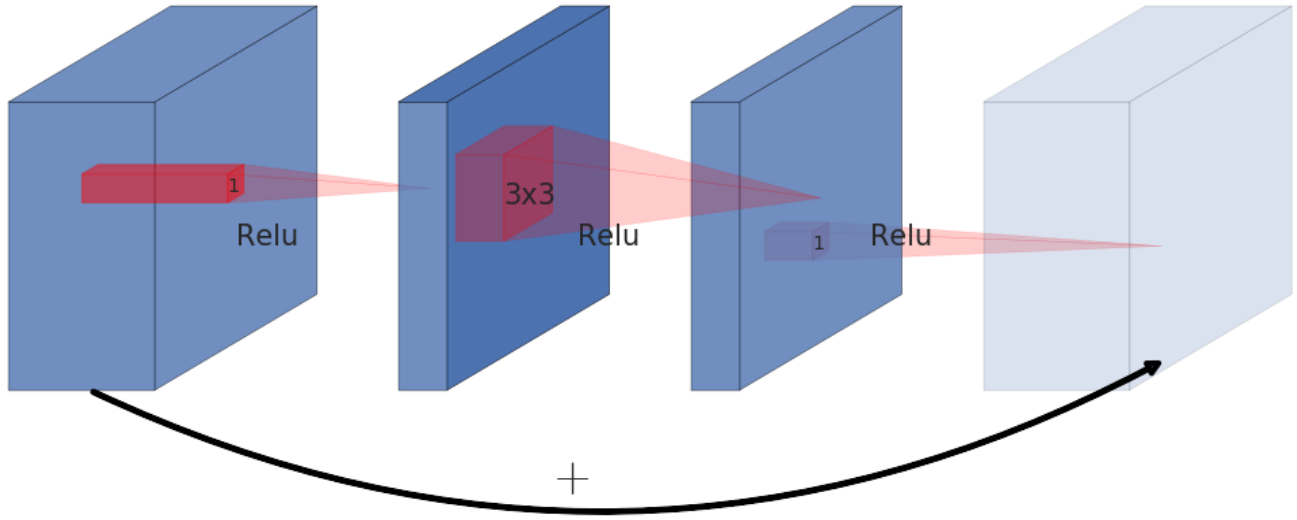
- Informally, for an input set of real images, we say that the set of layer activations forms a “manifold of interest”
- It has been long assumed that manifolds of interest in neural networks could be embedded in low-dimensional subspaces

Linear Bottlenecks

- The authors have highlighted two properties that are indicative of the requirement that the manifold of interest should lie in a low-dimensional subspace of the higher-dimensional activation space
  1. If the manifold of interest remains non-zero volume after ReLU transformation, it corresponds to a linear transformation.
  2. ReLU is capable of preserving complete information about the input manifold, but only if the input manifold lies in a low-dimensional subspace of the input space.
- Assuming the manifold of interest is low-dimensional we can capture this by inserting linear bottleneck layers into the convolutional blocks

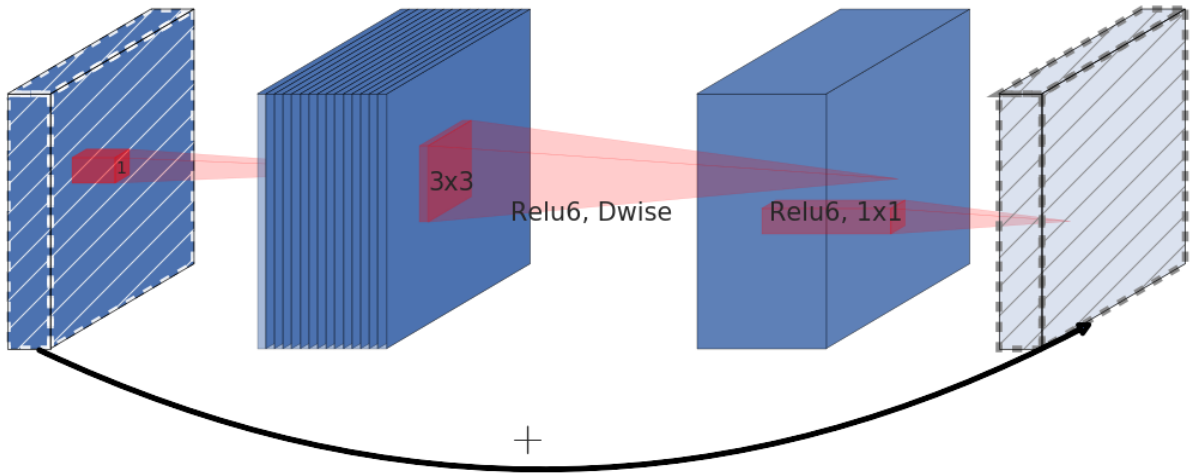
Residual Blocks

- Residual blocks connect the beginning and end of a convolutional block with a shortcut connection. By adding these two states the network has the opportunity of accessing earlier activations that weren't modified in the convolutional block.
- wide → narrow(bottleneck) → wide approach



Inverted Residuals

- Inspired by the intuition that the bottlenecks actually contain all the necessary information, while an expansion layer acts merely as an implementation detail that accompanies a non-linear transformation of the tensor, the authors use shortcuts directly between the bottlenecks
- narrow → wide → narrow approach

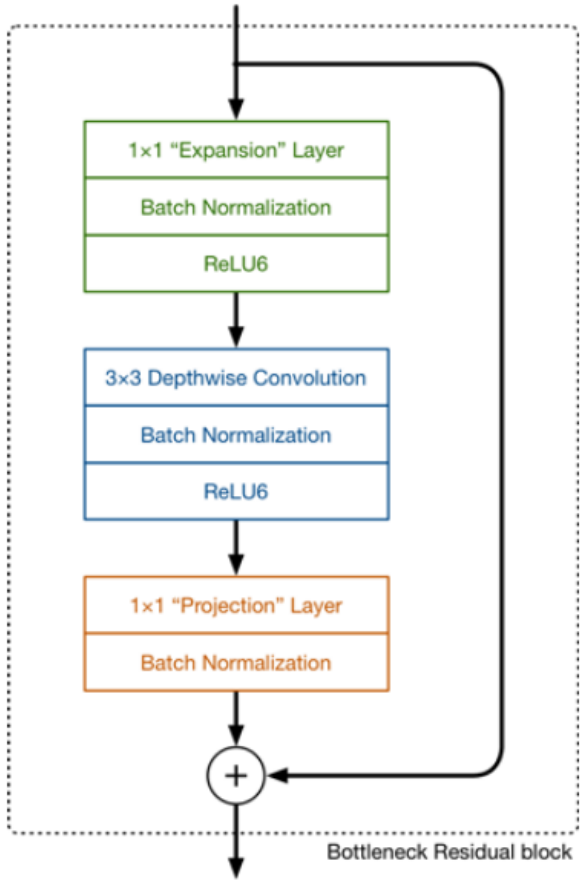


Bottleneck Residual Blocks

- The basic building block is a bottleneck depth-separable convolution with residuals

Input	Operator	Output
$h \times w \times k$	1x1 conv2d , ReLU6	$h \times w \times (tk)$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk)$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

Table 1: *Bottleneck residual block* transforming from k to k' channels, with stride s, and expansion factor t.



Figures from http://machinethink.net/blog/mobilenet-v2/

of Operations

- For a block of size $h \times w$, expansion factor t and kernel size k with d' input channels and d'' output channels,

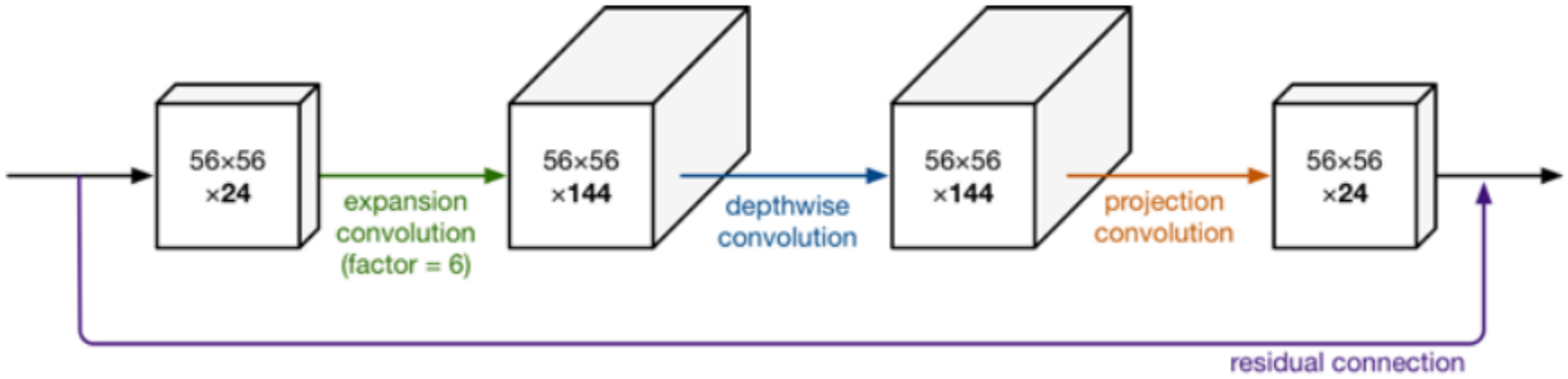
- The total number of multiply add required is

$$h \times w \times t \times d' \times d' + h \times w \times t \times d' \times k \times k + h \times w \times d'' \times t \times d'$$

$$= h \cdot w \cdot d' \cdot t(d' + k^2 + d'')$$

Information Flow Interpretation

- The proposed convolutional block has a unique property that allows to separate the network expressiveness (encoded by expansion layers) from its capacity (encoded by bottleneck inputs).



Figures from http://machinethink.net/blog/mobilenet-v2/

The Architecture of MobileNetV2

- The architecture of MobileNetV2 contains the initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers described in the Table 2.

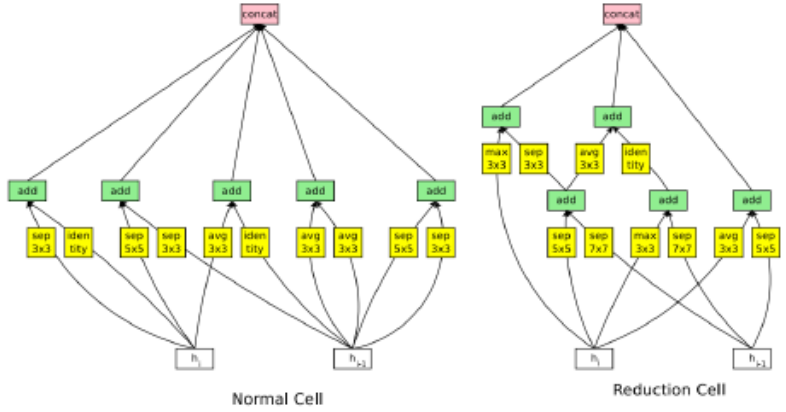
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck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

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

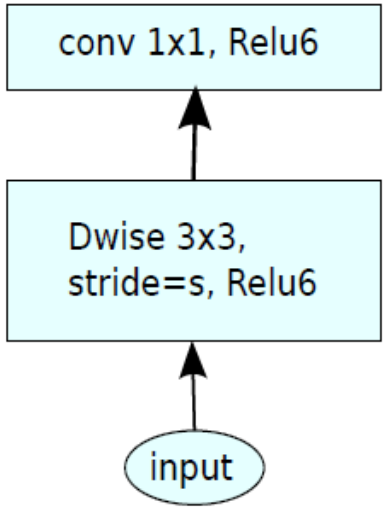
Trade-off Hyper Parameters

- Input Resolution
  - From 96 to 224
- Width Multiplier
  - From 0.35 to 1.4

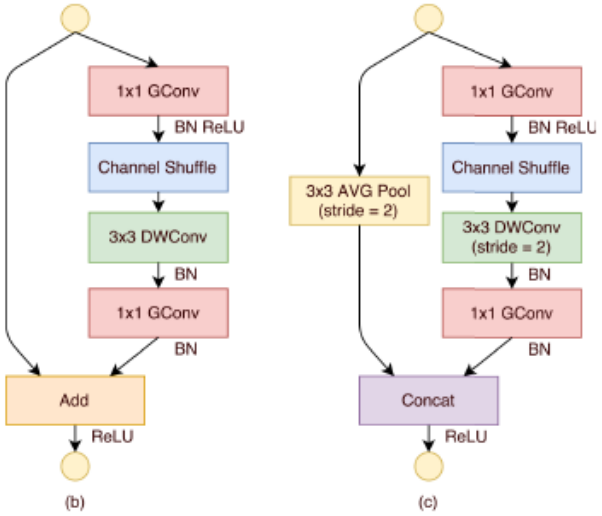
Comparison of Convolutional Blocks for Different Architectures



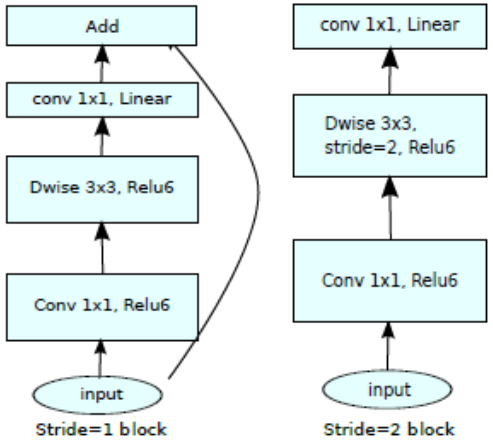
(a) NasNet[23]



(b) MobileNet[27]



(c) ShuffleNet [20]



(d) Mobilenet V2

The Max Number of Channels/Memory(in Kb)

Size	MobileNetV1	MobileNetV2	ShuffleNet (2x,g=3)
112x112	1/O(1)	1/O(1)	1/O(1)
56x56	128/800	32/200	48/300
28x28	256/400	64/100	400/600K
14x14	512/200	160/62	800/310
7x7	1024/199	320/32	1600/156
1x1	1024/2	1280/2	1600/3
max	800K	**200K**	600K

ImageNet Classification Results

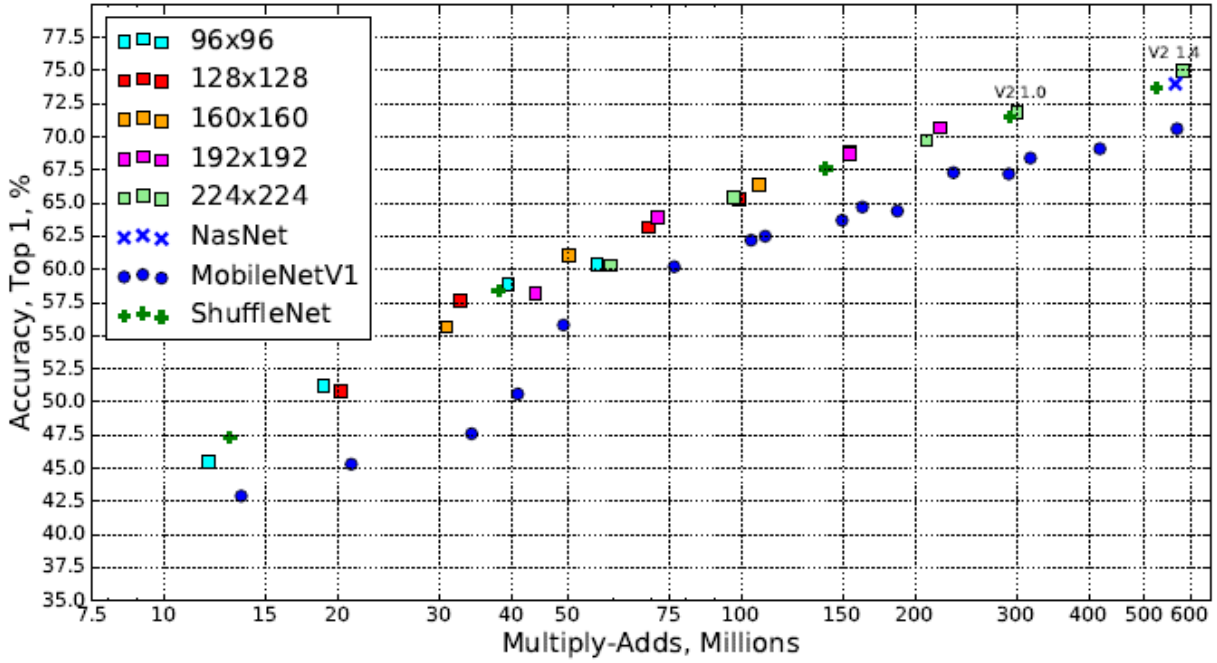


Figure 5: Performance curve of MobileNetV2 vs MobileNetV1, ShuffleNet, NAS. For our networks we use multipliers 0.35, 0.5, 0.75, 1.0 for all resolutions, and additional 1.4 for for 224. Best viewed in color.

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

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

Vs MobileNetV1

Version	MACs (millions)	Parameters (millions)
MobileNet V1	569	4.24
MobileNet V2	300	3.47

Version	iPhone 7	iPhone X	iPad Pro 10.5
MobileNet V1	118	162	204
MobileNet V2	145	233	220

Version	Top-1 Accuracy	Top-5 Accuracy
MobileNet V1	70.9	89.9
MobileNet V2	71.8	91.0

Data from http://machinethink.net/blog/mobilenet-v2/

SqueezeNext

SqueezeNext: Hardware-Aware Neural Network Design

Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai,
Xiangyu Yue, Peter Jin, Sicheng Zhao, Kurt Keutzer
EECS, UC Berkeley
{amirgh,kiseok.kwon,bichen,zizheng,xyyue,phj,schzhao,keutzer}@berkeley.edu

Abstract

One of the main barriers for deploying neural networks on embedded systems has been large memory and power consumption of existing neural networks. In this work, we introduce SqueezeNext, a new family of neural network architectures whose design was guided by considering previous architectures such as SqueezeNet, as well as by simulation results on a neural network accelerator. This new network is able to match AlexNet's accuracy on the ImageNet benchmark with 112× *fewer parameters, and one of its deeper variants is able to achieve VGG-19 accuracy with only 4.4 Million parameters, (*31× *smaller than VGG-19). SqueezeNext also achieves better top-5 classification accuracy with* 1.3× *fewer parameters as compared to MobileNet, but avoids using depthwise-separable convolutions*

size of the network, the original model had to be trained on two GPUs with a model parallel approach, where the filters were distributed to these GPUs. Moreover, dropout was required to avoid overfitting using such a large model size. The next major milestone in ImageNet classification was made by VGG-Net family [23], which exclusively uses 3×3 convolutions. The main ideas here were usage of 3×3 convolutions to approximate 7×7 filter's receptive field, along with a deeper network. However, the model size of VGG-19 with 138 million parameters is even larger than AlexNet and not suitable for real-time applications. Another step forward in architecture design was the ResNet family [9], which incorporates a repetitive structure of 1×1 and 3×3 convolutions along with a skip connection. By changing the depth of the networks, the authors showed competitive performance for multiple learning tasks.

Motivation

- A general trend of neural network design has been to find larger and deeper models to get better accuracy without considering the memory or power budget.
- However, increase in transistor speed due to semiconductor process improvements has slowed dramatically, and it seems unlikely that mobile processors will meet computational requirements on a limited power budget.

Contributions

- Use a more aggressive channel reduction by incorporating a two-stage squeeze module.
- Use separable 3x3 convolutions to further reduce the model size, and remove the additional 1x1 branch after the squeeze module.
- Use an element-wise addition skip connection similar to that of ResNet architecture
- Optimize the baseline SqueezeNext architecture by simulating its performance on a multi-processor embedded system.

Design – Low Rank Filters

- Decompose the K x K convolutions into two separable convolutions of size 1 x K and K x 1
- This effectively reduces the number of parameters from K^2 to 2K, and also increases the depth of the network.

Design – Bottleneck Module

- Use a variation of bottleneck approach by using a two stage squeeze layer
- Use two bottleneck modules each reducing the channel size by a factor of 2, which is followed by two separable convolutions
- Also incorporate a final 1 x 1 expansion module, which further reduces the number of output channels for the separable convolutions.

Design – Fully Connected Layers

- In the case of AlexNet, the majority of the network parameters are in Fully Connected layers, accounting for 96% of the total model size.
- SqueezeNext incorporates a final bottleneck layer to reduce the input channel size to the last fully connected layer, which considerably reduces the total number of model parameters.

Comparison of Building Blocks

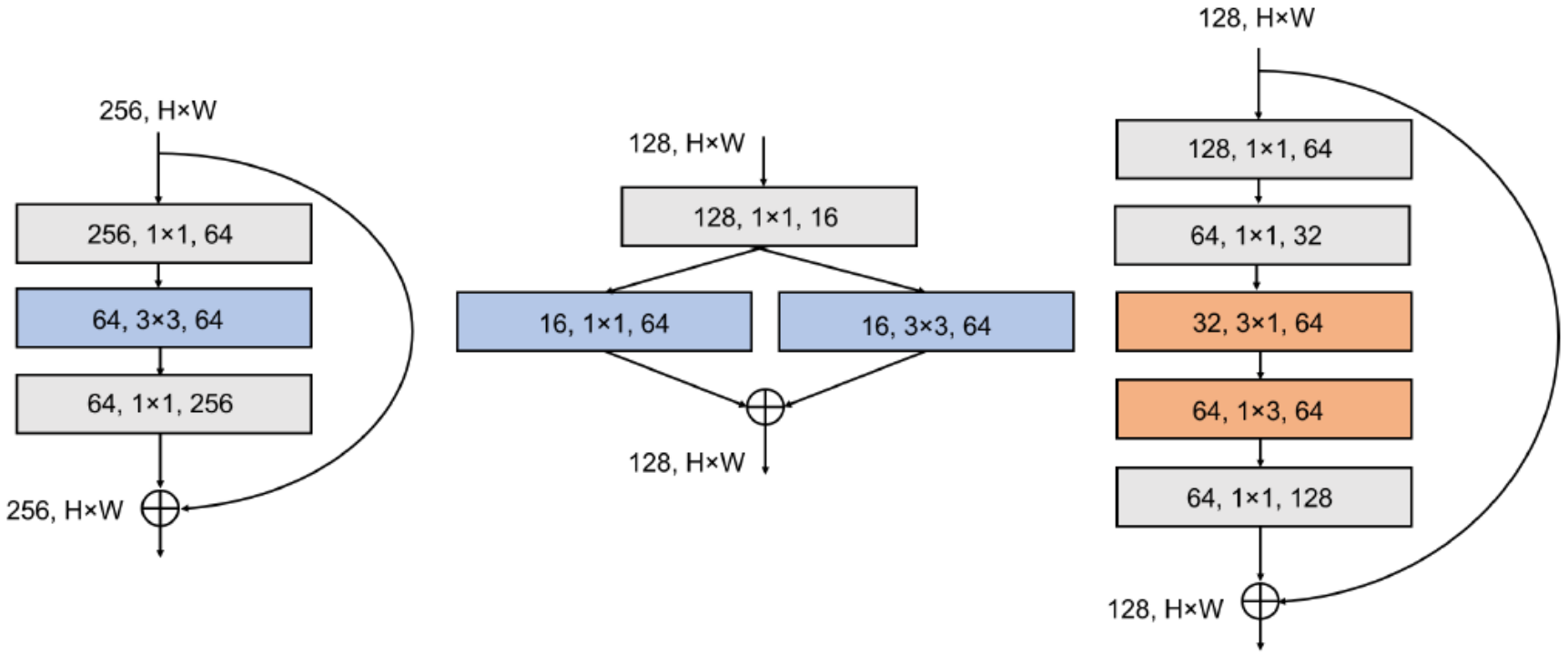
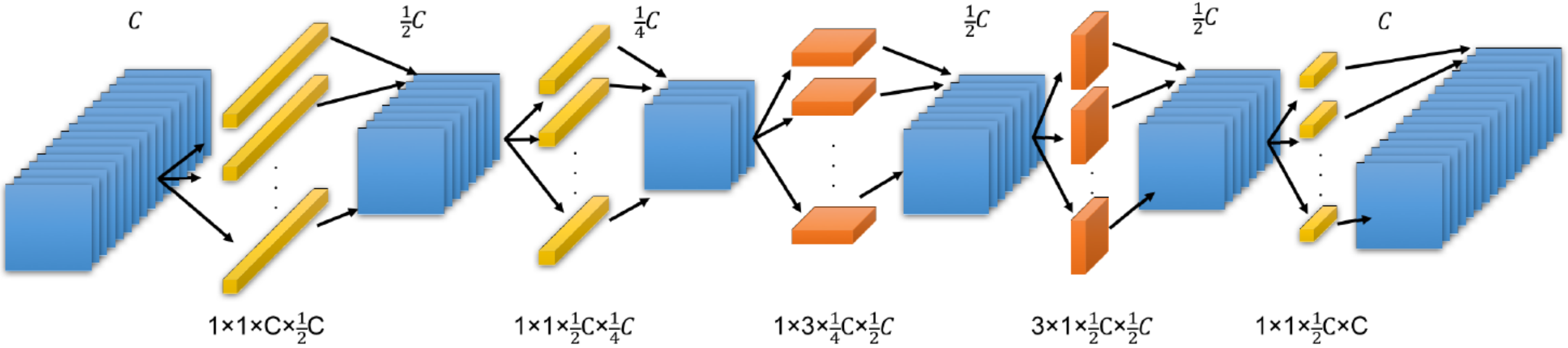


Figure 1: *Illustration of a ResNet block on the left, a SqueezeNet block in the middle, and a SqueezeNext (SqNxt) block on the right. SqueezeNext uses a two-stage bottleneck module to reduce the number of input channels to the* 3×3 *convolution. The latter is further decomposed into separable convolutions to further reduce the number of parameters (orange parts), followed by a* 1×1 *expansion module.*

SqueezeNext Block



Block Arrangement in 1.0-SqNxt-23

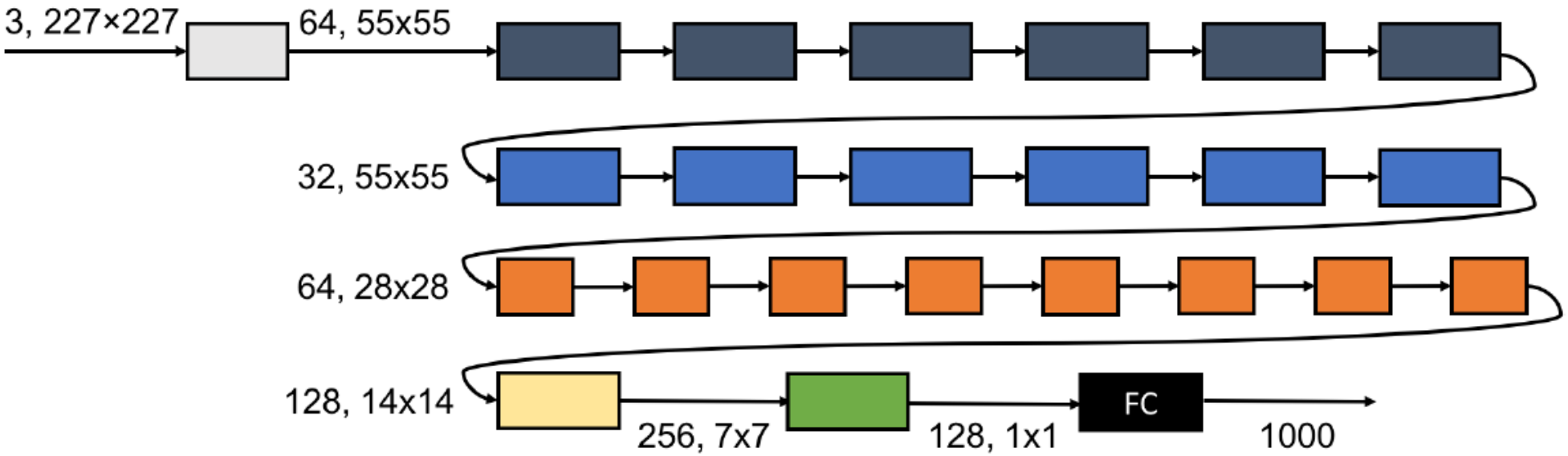


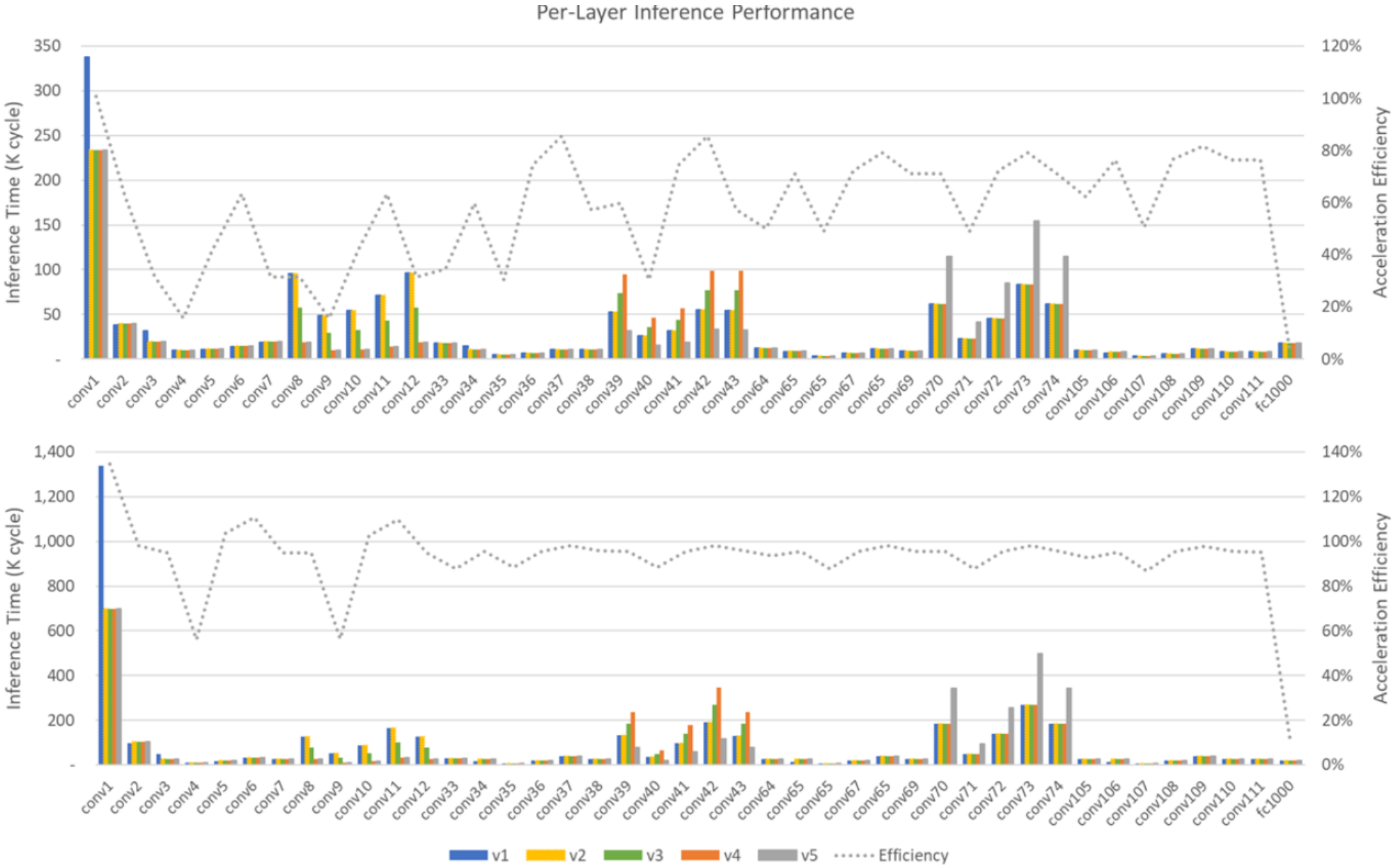
Figure 3: *Illustration of block arrangement in 1.0-SqNxt-23. Each color change corresponds to a change in input feature map's resolution. The number of blocks after the first convolution/pooling layer is* $Depth = [6, 6, 8, 1]$*, where the last number refers to the yellow box. This block is followed by a bottleneck module with average pooling to reduce the channel size and spatial resolution (green box), followed by a fully connected layer (black box). In optimized variations of the baseline, we change this depth distribution by decreasing the number of blocks in early stages (dark blue), and instead assign more blocks to later stages (Fig. 9). This increases hardware performance as early layers have poor compute efficiency.*

Results – Classification

Model	Top-1	Top-5	# Params	Comp.
AlexNet	57.10	80.30	60.9M	1×
SqueezeNet	57.50	80.30	1.2M	51×
1.0-SqNxt-23	59.05	82.60	0.72M	84×
1.0-G-SqNxt-23	**57.16**	**80.23**	**0.54M**	**112×**
1.0-SqNxt-23-IDA	60.35	83.56	0.9M	68×
1.0-SqNxt-34	61.39	84.31	1.0M	61×
1.0-SqNxt-34-IDA	62.56	84.93	1.3	47×
1.0-SqNxt-44	**62.64**	**85.15**	**1.2M**	51×
1.0-SqNxt-44-IDA	63.75	85.97	1.5M	41×

Model	Top-1	Top-5	Params
1.5-SqNxt-23	63.52	85.66	1.4M
1.5-SqNxt-34	66.00	87.40	2.1M
1.5-SqNxt-44	67.28	88.15	2.6M
VGG-19	**68.50**	**88.50**	**138M**
2.0-SqNxt-23	67.18	88.17	2.4M
2.0-SqNxt-34	68.46	88.78	3.8M
2.0-SqNxt-44	**69.59**	**89.53**	**4.4M**
MobileNet	67.50 (70.9)	86.59 (89.9)	4.2M
2.0-SqNxt-23v5	67.44 (69.8)	**88.20(89.5)**	**3.2M**

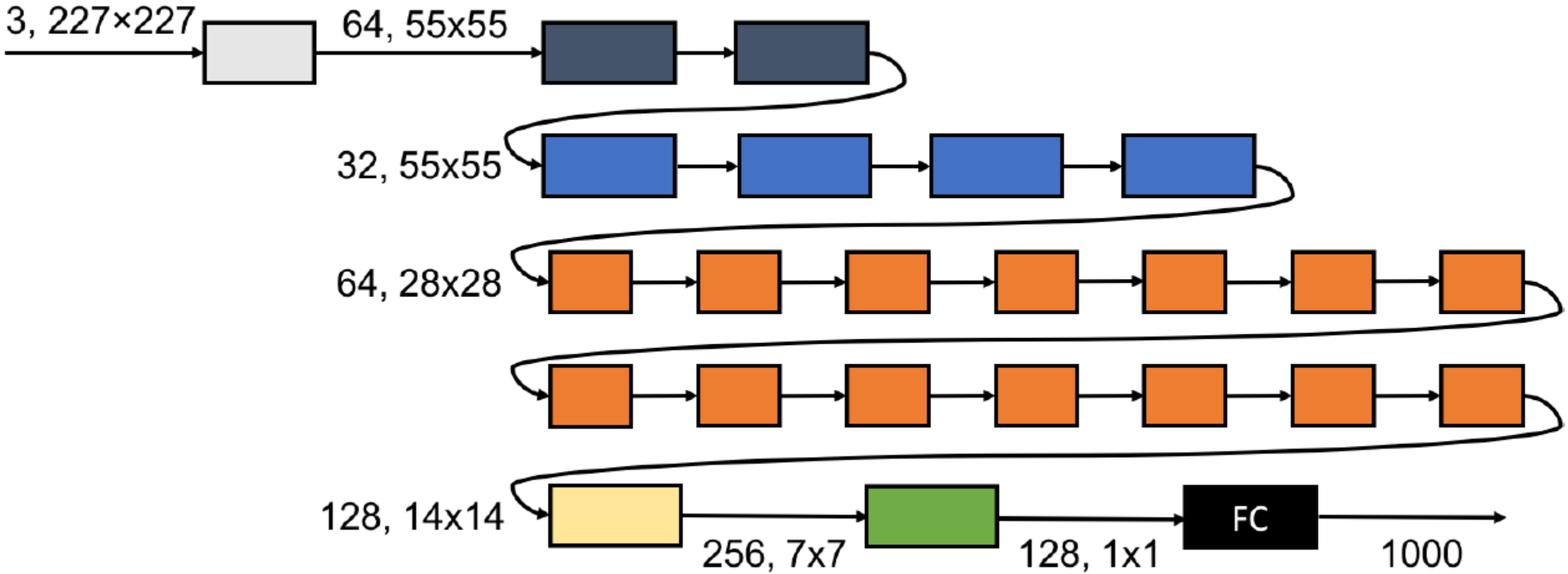
Results – Hardware Performance



SqueezeNext v2~v5

- In the 1.0-SqNxt-23, the first 7 x 7 convolutional layer accounts for 26% of the total inference time.
- Therefore, the first optimization we make is replacing this 7 x 7 layer with a 5 x 5 convolution, and construct 1.0-SqNxt-23-v2 model.
- Note the significant drop in efficiency for the layers in the first module. The reason for this drop is that the initial layers have very small number of channels which needs to be applied a large input activation map.
- In the v3/v4 variation, authors reduce the number of the blocks in the first module by 2/4 and instead add it to the second module, respectively. In the v5 variation, authors reduce the blocks of the first two modules and instead increase the blocks in the third module.

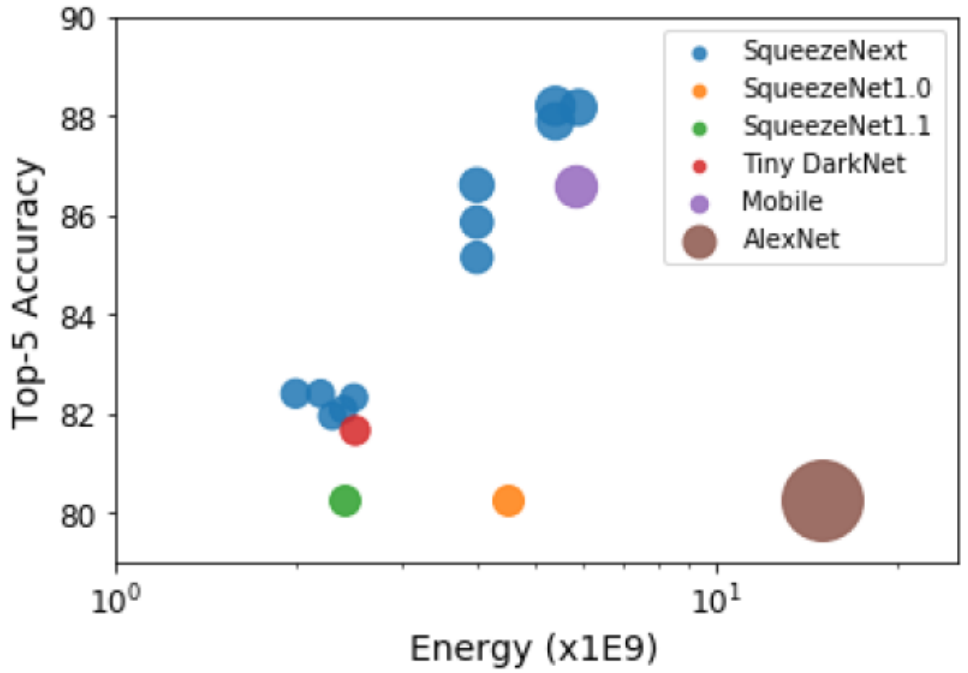
1.0-SqNxt-23v5

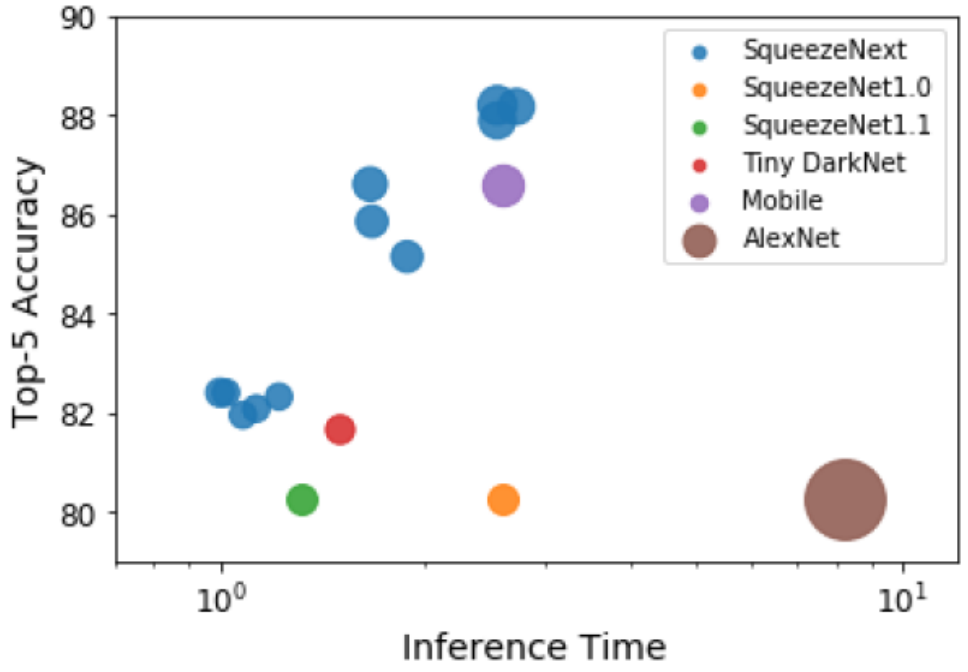


Results

Model	Params (× 1E+6)	MAC	Top-1	Top-5	Depth	8x8, 32KB Time	8x8, 32KB Energy	16x16, 128KB Time	16x16, 128KB Energy
AlexNet	60.9	725M	57.10	80.30	—	x5.46	1.6E+10	x8.26	1.5E+10
SqueezeNet v1.0	1.2	837M	57.50	80.30	—	x3.42	6.7E+09	x2.59	4.5E+09
SqueezeNet v1.1	1.2	352M	57.10	80.30	—	x1.60	3.3E+09	x1.31	2.4E+09
Tiny Darknet	1.0	495M	58.70	81.70	—	x1.92	3.8E+09	x1.50	2.5E+09
1.0-SqNxt-23	0.72	282M	59.05	82.60	[6,6,8,1]	x1.17	3.2E+09	x1.22	2.5E+09
1.0-SqNxt-23v2	0.74	228M	58.55	82.09	[6,6,8,1]	**x1.00**	2.8E+09	x1.13	2.4E+09
1.0-SqNxt-23v3	0.74	228M	58.18	81.96	[4,8,8,1]	x1.00	2.7E+09	x1.08	2.3E+09
1.0-SqNxt-23v4	0.77	228M	59.09	82.41	[2,10,8,1]	x1.00	2.6E+09	x1.02	2.2E+09
1.0-SqNxt-23v5	0.94	**228M**	**59.24**	82.41	[2,4,14,1]	x1.00	**2.6E+09**	**x1.00**	**2.0E+09**
MobileNet	4.2	574M	67.50(70.9)	86.59(89.9)	—	**x2.94**	9.1E+09	x2.60	5.8E+09
2.0-SqNxt-23	2.4	749M	67.18	88.17	[6,6,8,1]	x3.24	8.1E+09	x2.72	5.9E+09
2.0-SqNxt-23v4	2.56	708M	66.95	87.89	[2,10,8,1]	x3.17	7.5E+09	x2.55	5.4E+09
2.0-SqNxt-23v5	**3.23**	708M	67.44	**88.20**	[2,4,14,1]	x3.17	**7.4E+09**	**x2.55**	**5.4E+09**

Results


90
88
86
84
82
80
Top-5 Accuracy
10^0
10^1
Energy (x1E9)
SqueezeNext
SqueezeNet1.0
SqueezeNet1.1
Tiny DarkNet
Mobile
AlexNet


90
88
86
84
82
80
Top-5 Accuracy
10^0
10^1
Inference Time
SqueezeNext
SqueezeNet1.0
SqueezeNet1.1
Tiny DarkNet
Mobile
AlexNet

ShuffleNet V2

ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design

Ningning Ma *[1,2] Xiangyu Zhang *[1] Hai-Tao Zheng[2] Jian Sun[1]

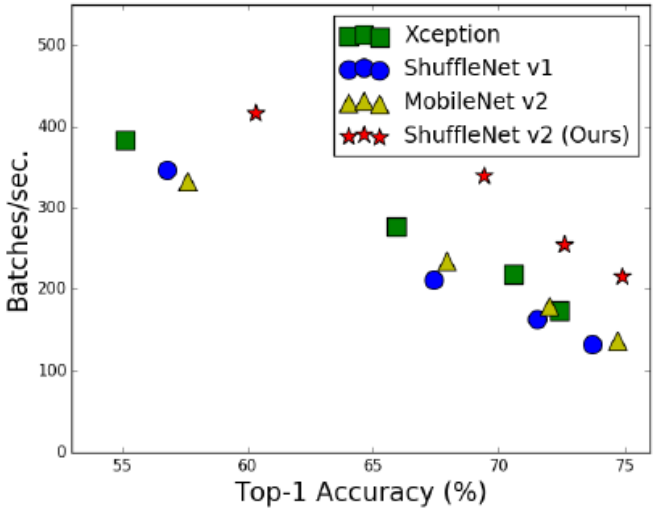
[1] Megvii Inc (Face++) [2] Tsinghua University
{maningning,zhangxiangyu,sunjian}@megvii.com
zheng.haitao@sz.tsinghua.edu.cn

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 characterics. 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.

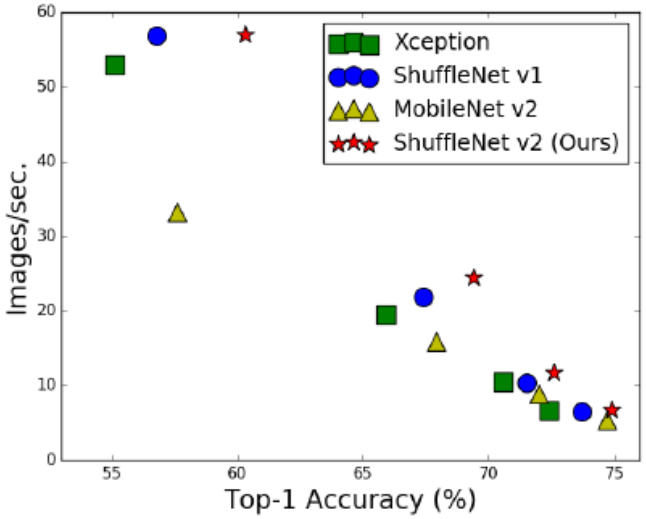
Motivation

- 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.

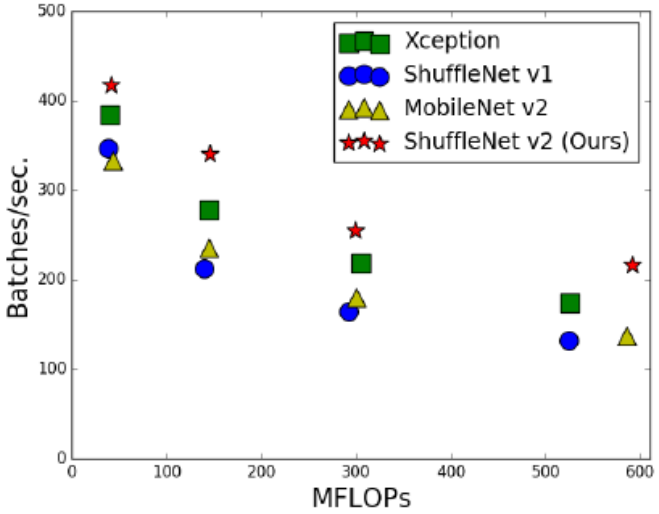
Accuracy, Speed and FLOPs



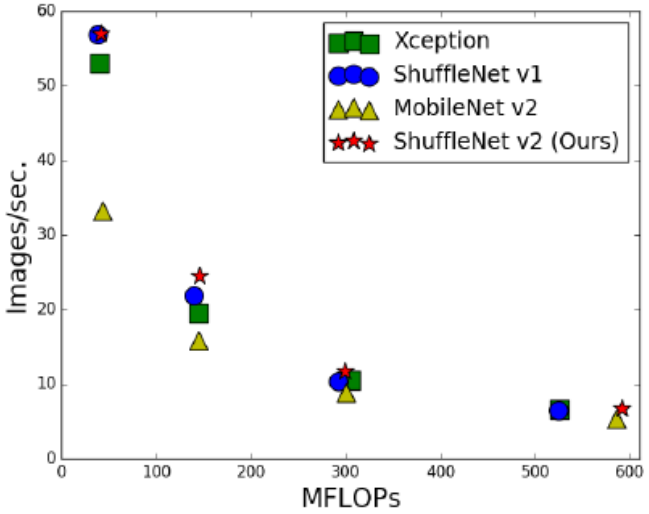
(a) GPU



(b) ARM



(c) GPU



(d) ARM

Two Reasons of the Discrepancy

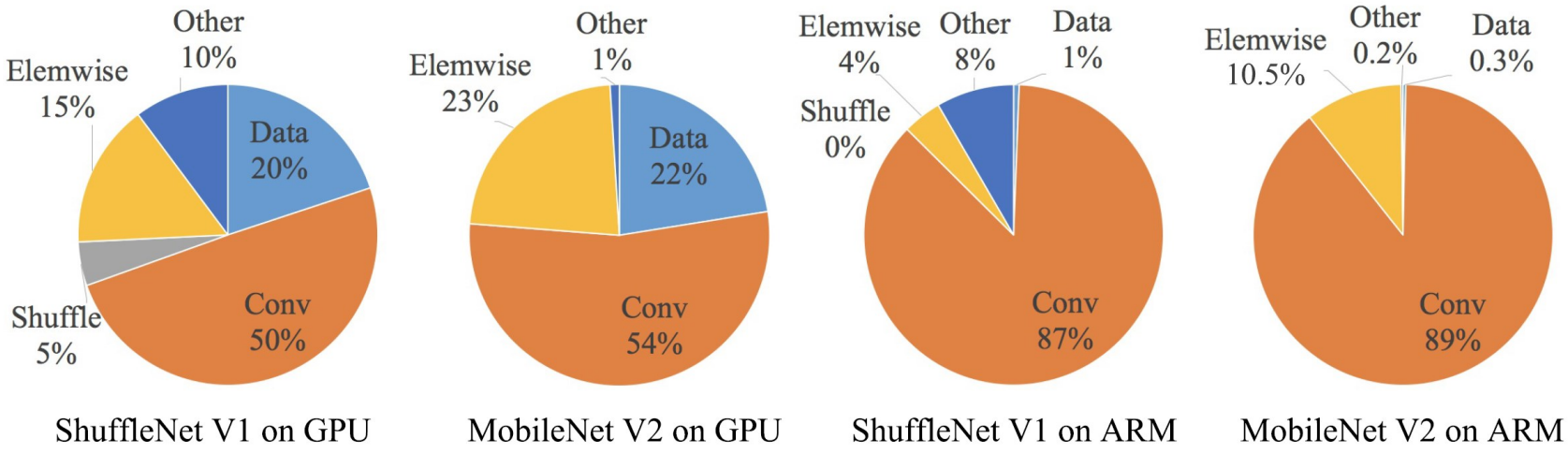
- First, several important factors that have considerable affection on speed are not taken into account by FLOPs. One such factor is memory access cost (MAC).
- Second, operations with the same FLOPs could have different running time, depending on the platform.

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.

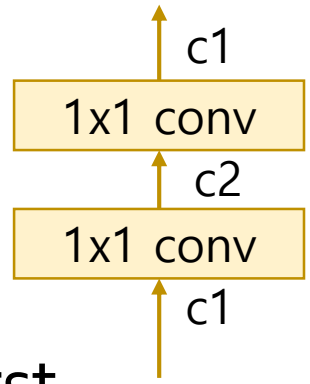


Guide 1

- **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 x 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 c1 input channels and c2 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 ($c1$ and $c2$) are tested, while the total FLOPs under the four ratios is fixed by varying the number of channels. Input image size is 56×56.

Guide 2

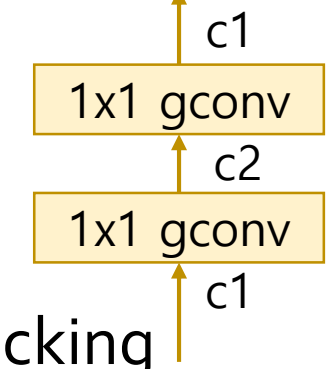
- **Excessive group convolution increases MAC.**
- Formally, following the notations in G1 and Eq. 1, the relation between MAC and FLOPs for 1 x 1 group convolution is

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

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

- Given the fixed input shape c1 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.

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

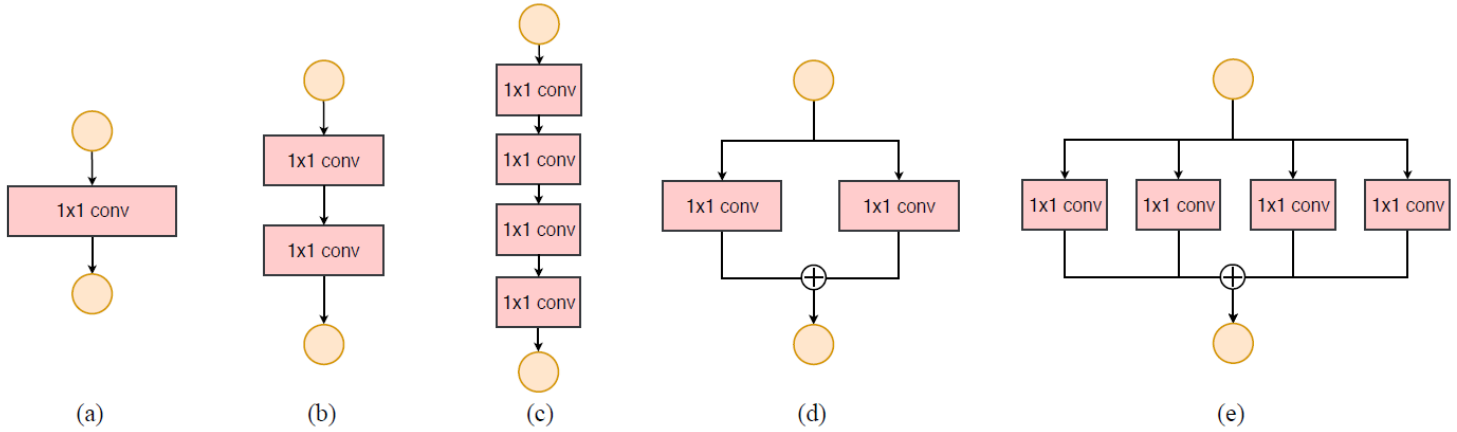
Table 2: Validation experiment for **Guideline 2**. Four values of group number g are tested, while the total FLOPs under the four values is fixed by varying the total channel number c. Input image size is 56×56.

- *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



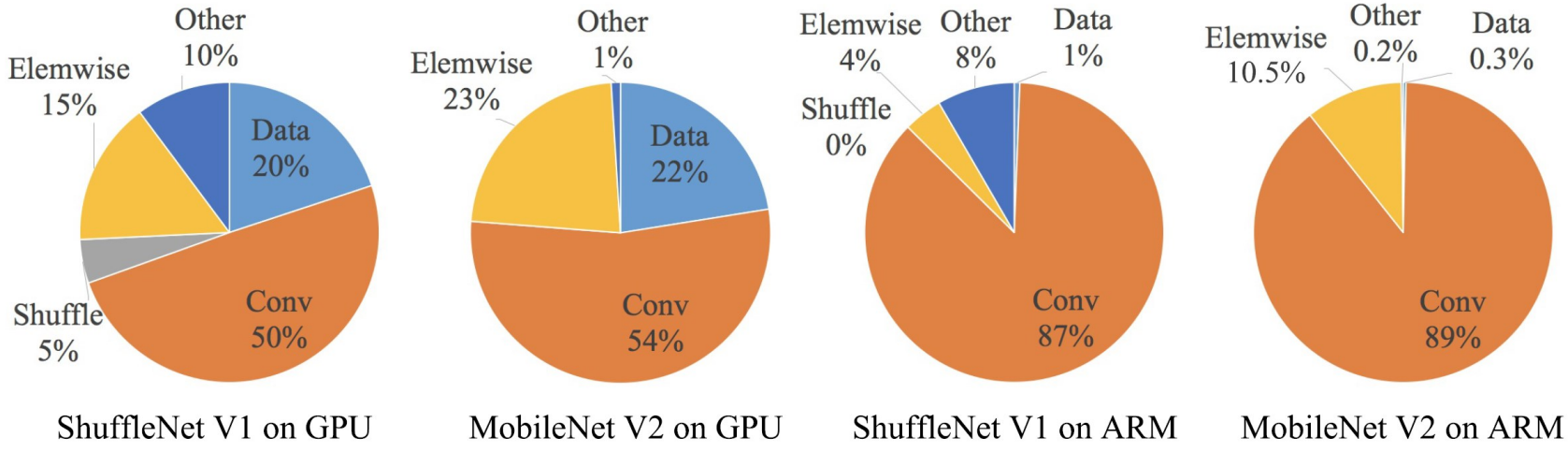
Appendix Fig. 1: Building blocks used in experiments for guideline 3. (a) *1-fragment*. (b) *2-fragment-series*. (c) *4-fragment-series*. (d) *2-fragment-parallel*. (e) *4-fragment-parallel*.

	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



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.

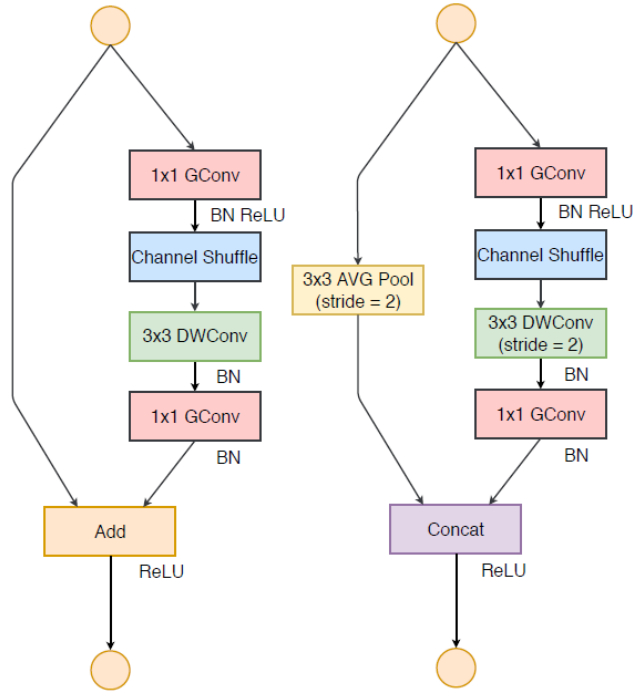
Conclusion and Discussion

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 (G1 and G2). This cost is non-negligible, especially for light-weight models. Also, using too many groups violates G3. The element-wise "Add" operation in the shortcut connection is also undesirable (G4).
- 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

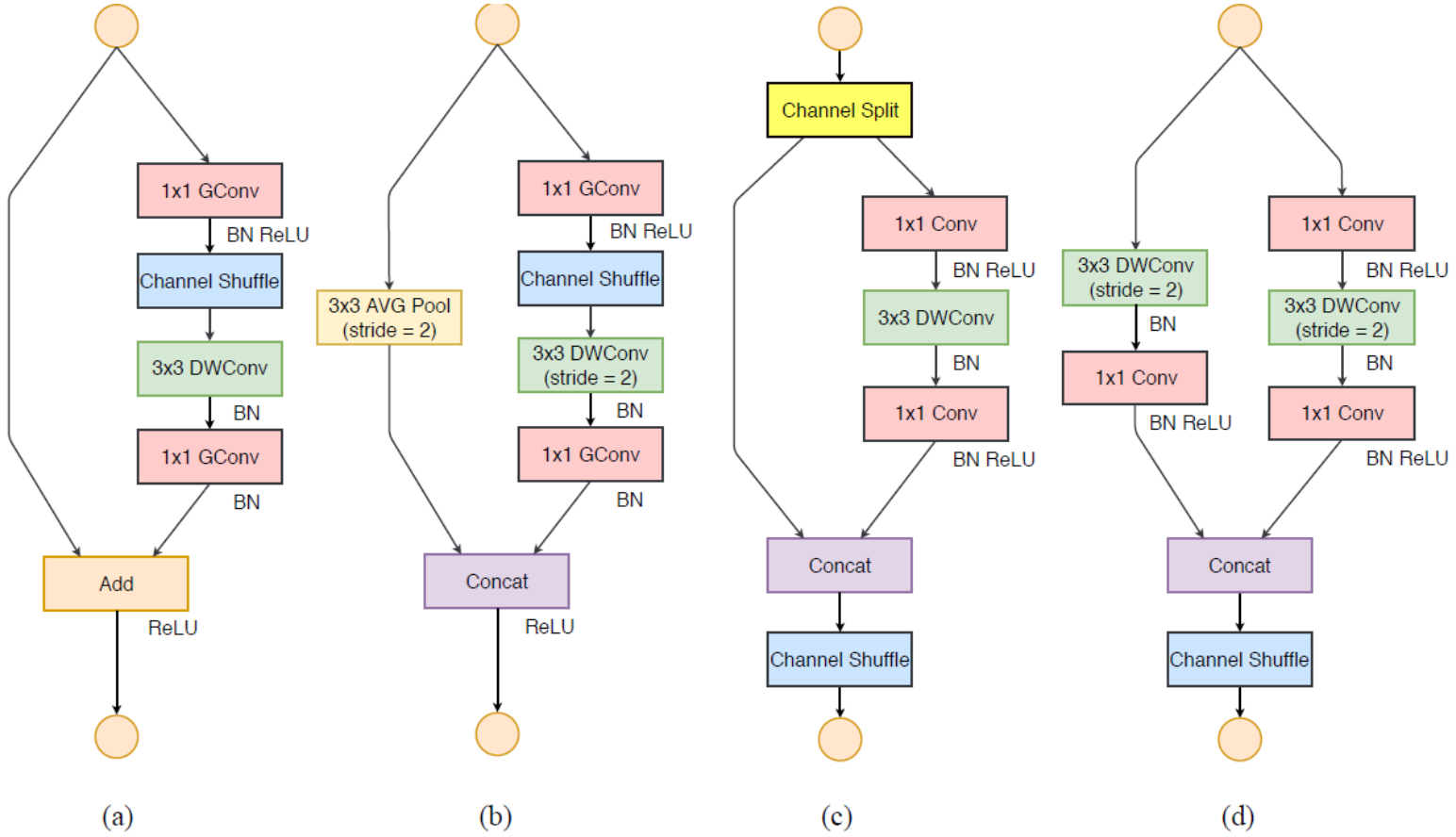


Fig. 3: Building blocks of ShuffleNet v1 [15] and this work. (a): the basic ShuffleNet unit; (b) the ShuffleNet unit for spatial down sampling (2×); (c) our basic unit; (d) our unit for spatial down sampling (2×). **DWConv**: depthwise convolution. **GConv**: group convolution.

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 x 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.

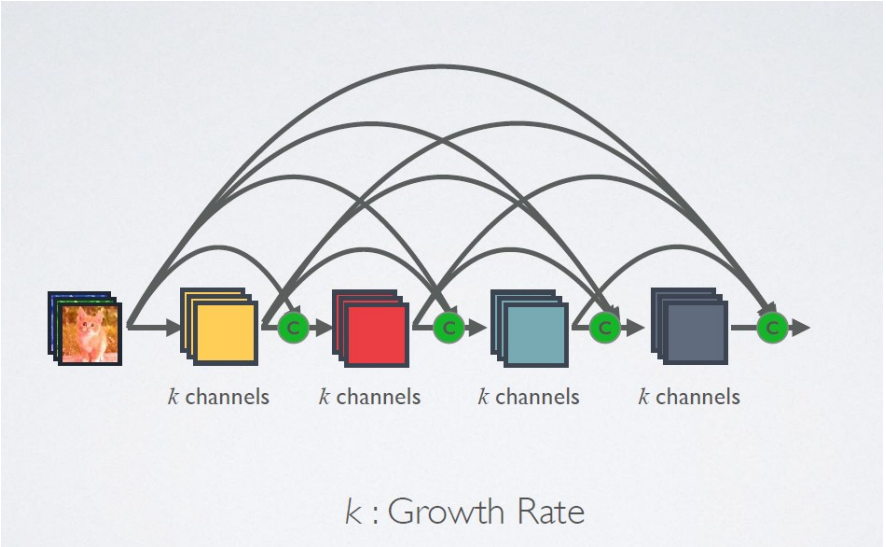
ShuffleNet V2

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	1	24	24	24	24
MaxPool	56×56	3×3	2					
Stage2	28×28		2	1	48	116	176	244
	28×28		1	3				
Stage3	14×14		2	1	96	232	352	488
	14×14		1	7				
Stage4	7×7		2	1	192	464	704	976
	7×7		1	3				
Conv5	7×7	1×1	1	1	1024	1024	1024	2048
GlobalPool	1×1	7×7						
FC					1000	1000	1000	1000
FLOPs					41M	146M	299M	591M
# of Weights					1.4M	2.3M	3.5M	7.4M

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 l1-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.

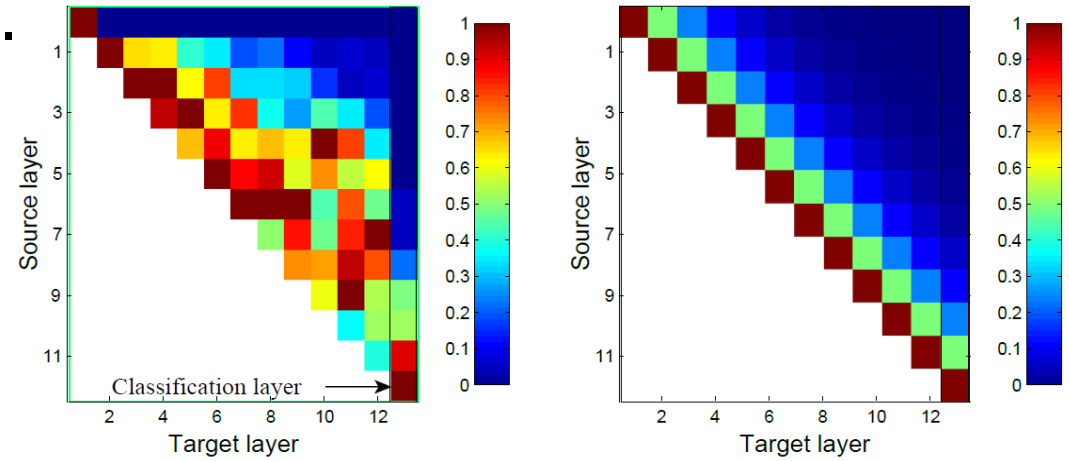


Fig. 4: Illustration of the patterns in feature reuse for *DenseNet* [6] and ShuffleNet V2. (a) (courtesy of [6]) the average absolute filter weight of convolutional layers in a model. The color of pixel (s, l) encodes the average $l1$-norm of weights connecting layer s to l. (b) The color of pixel (s, 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)	299	**27.4**	255	**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)	591	**25.1**	217	**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])	597	**24.6**	**161**	**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

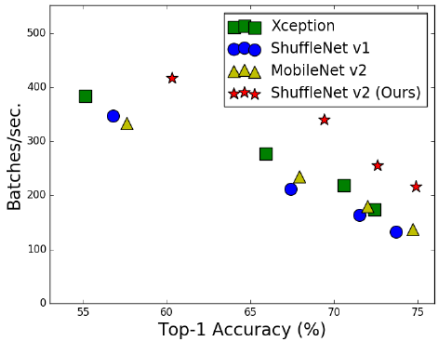
Accuracy vs FLOPs

- 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 x 224 image size. This is probably caused by too few channels.

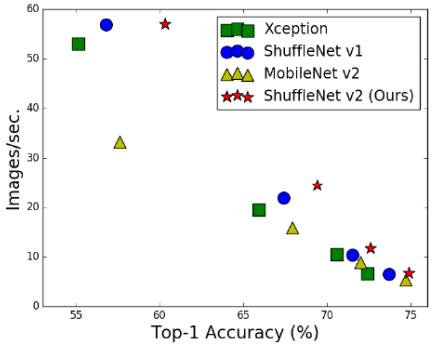
Inference Speed vs FLOPs/Accuracy

- 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.)

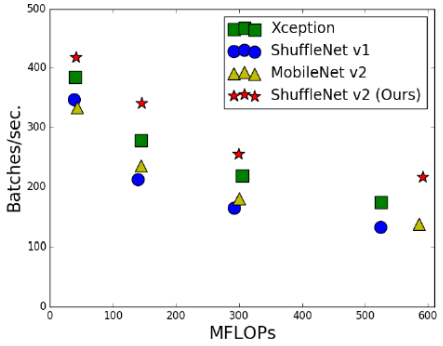
		GPU (Batches/sec.)				CPU (Images/sec.)			
Input size	FLOPs	40M	140M	300M	500M	40M	140M	300M	500M
320x320	ShuffleNet v2	315*	525	474	422	28.1	12.5	6.1	3.4
	ShuffleNet v1	236*	414	344	275	27.2	11.4	5.1	3.1
	MobileNet v2	187*	460	389	335	11.4	6.4	4.6	2.7
	Xception	279*	463	408	350	31.1	10.1	5.6	3.5
640x480	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



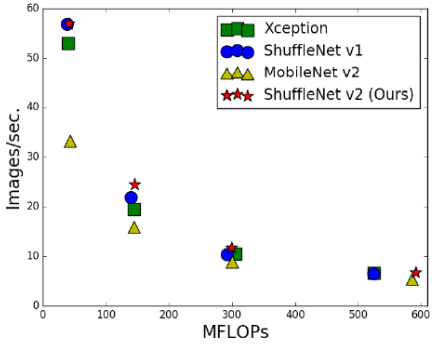
(a) GPU



(b) ARM



(c) GPU



(d) ARM

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 *SE* [8])	597	**24.6**	**161**	**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 x 3 depthwise convolution before the first pointwise convolution in each building block.(ShuffleNet V2*)

Model	mmAP(%)				GPU Speed (Images/sec.)			
FLOPs	40M	140M	300M	500M	40M	140M	300M	500M
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