CNN for CV

AI for CV Group
2019

Week 7.
CNN II. Network Structure

Augmentation

Selection Strategy

Data

Initialization Strategy

Learning Strategy

Optimization Strategy

Framework

Modules

Layers

Network

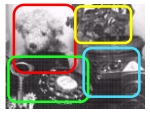
Loss Type

Loss Selection

Update Strategy

Evaluation

Loss



Outline:

I. Net Framework

A. Traditional Classic Network Frameworks

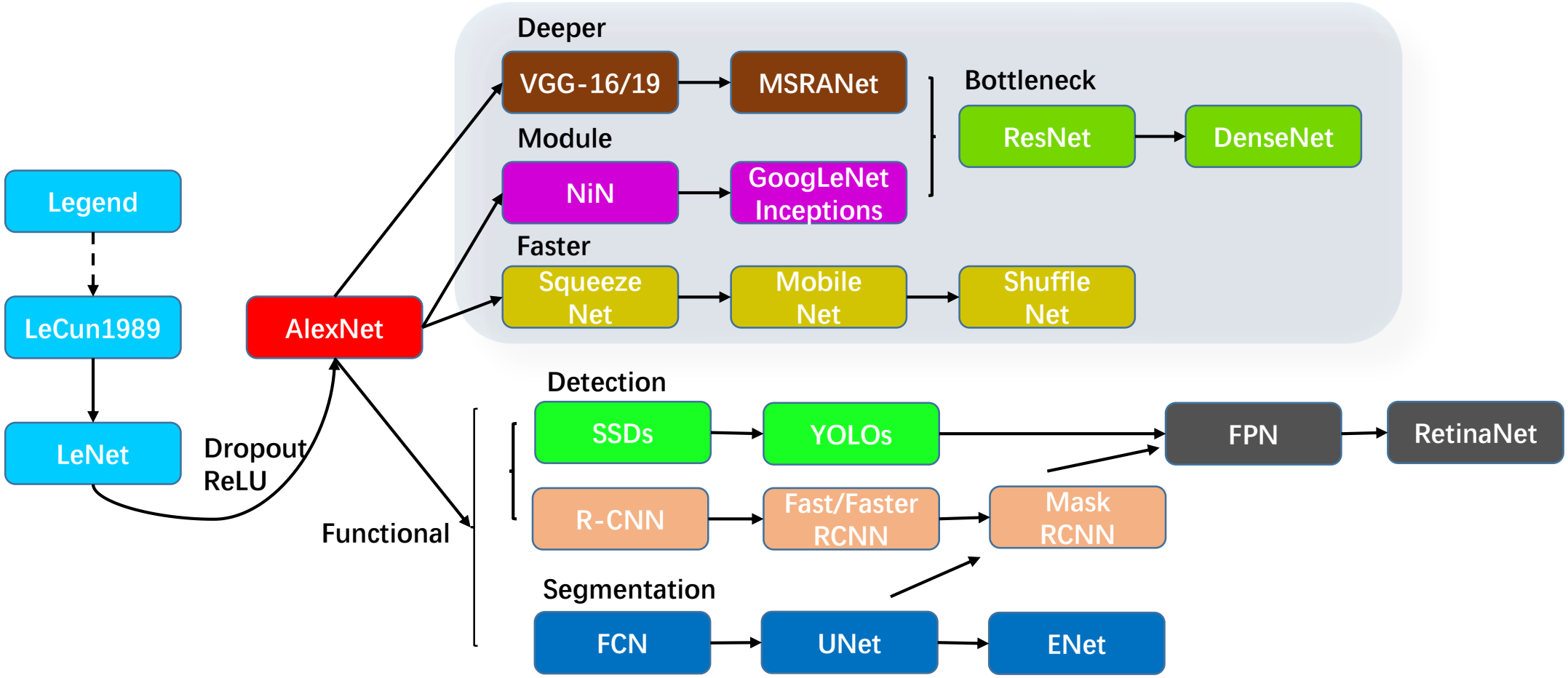
B. Light Frameworks

II. FLOPs

C. Concept & Computation Of FLOPs

I. Net Framework

I. Net Framework



I. Net Framework

A. Classic Frameworks:

History:

[0-1st generation]

LeNet-5 (1998) -> AlexNet (2012) -> ZF (2013) [Before History]

[2nd generation]

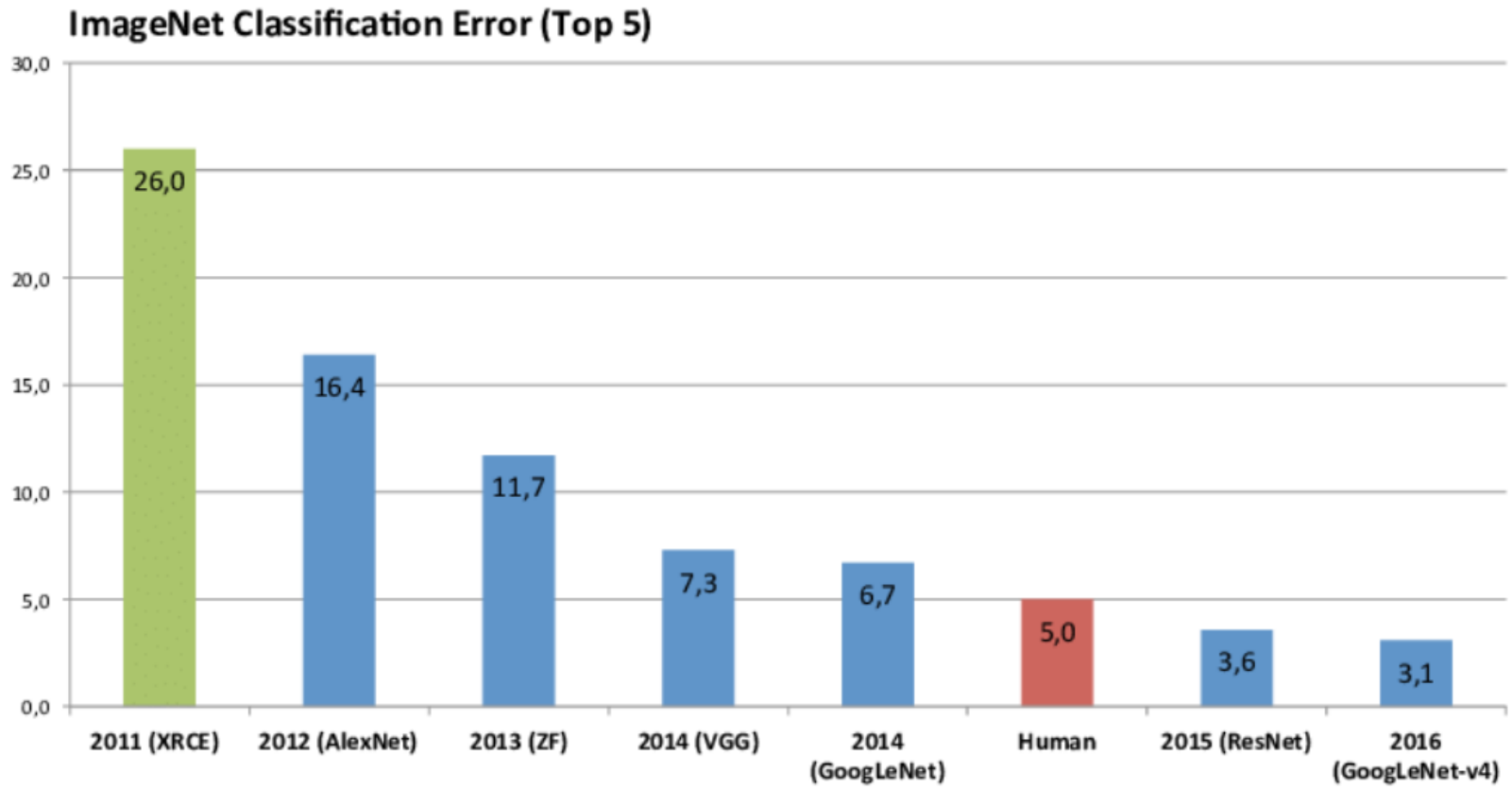
VGGNet (2014) -> GoogLeNet (2014) -> [Ancient History]

[3rd generation]

ResNet (2015) -> DenseNet (2017) -> CapsNet (2017) [Modern History]

I. Net Framework

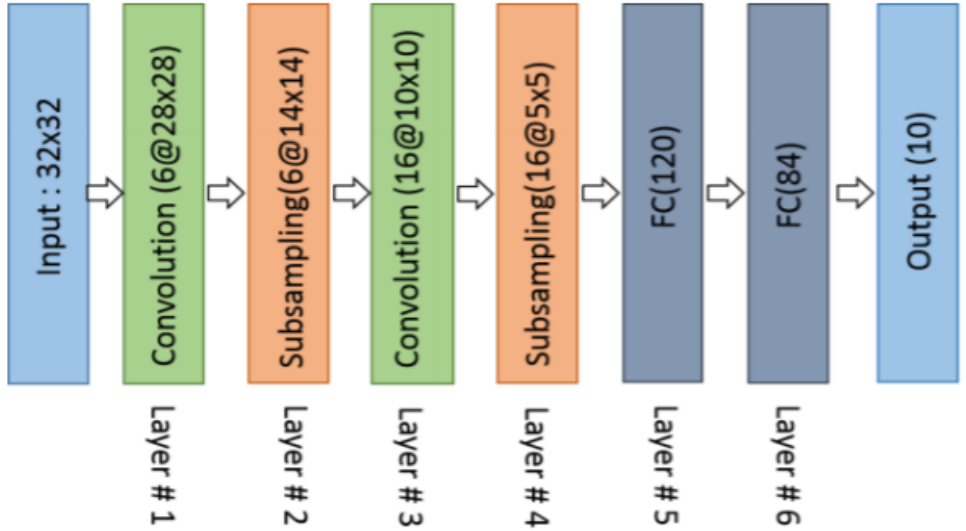
A. Classic Frameworks:

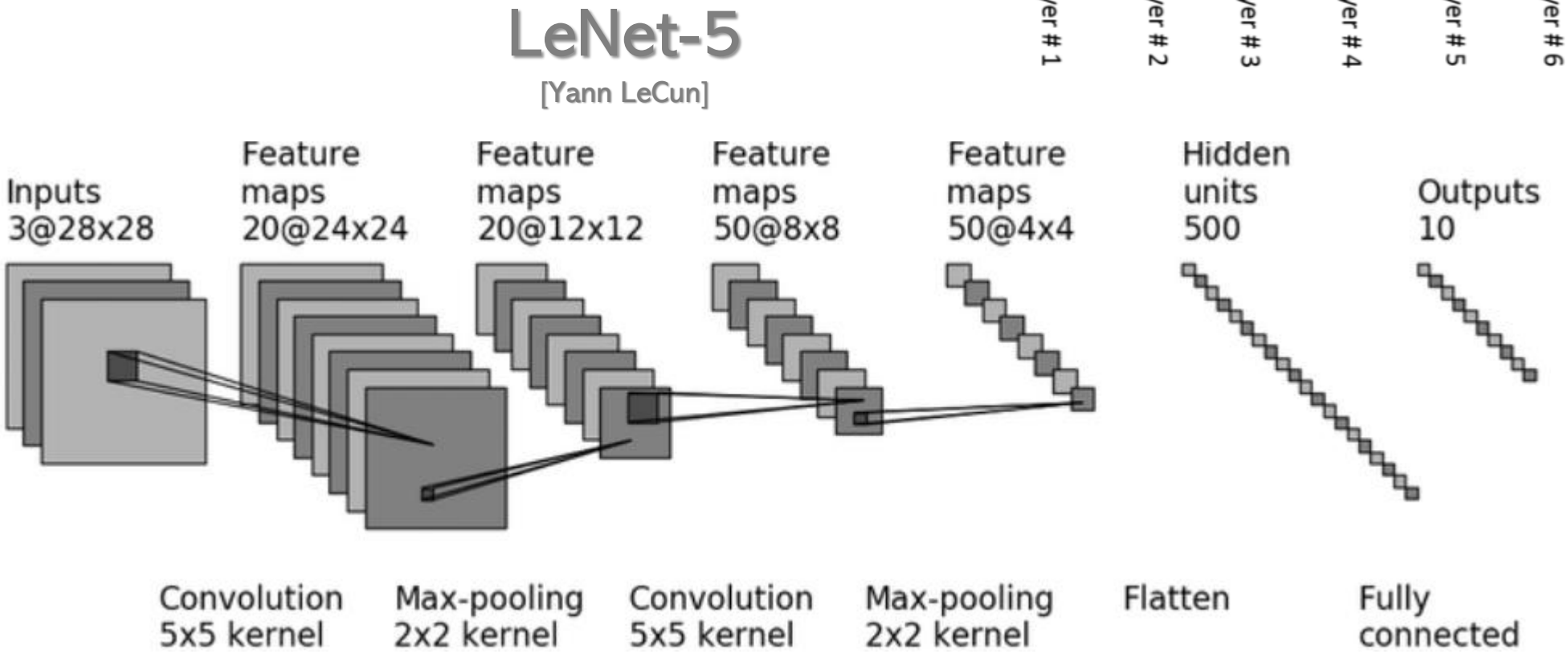


I. Net Framework

A. Classic Frameworks:

History:

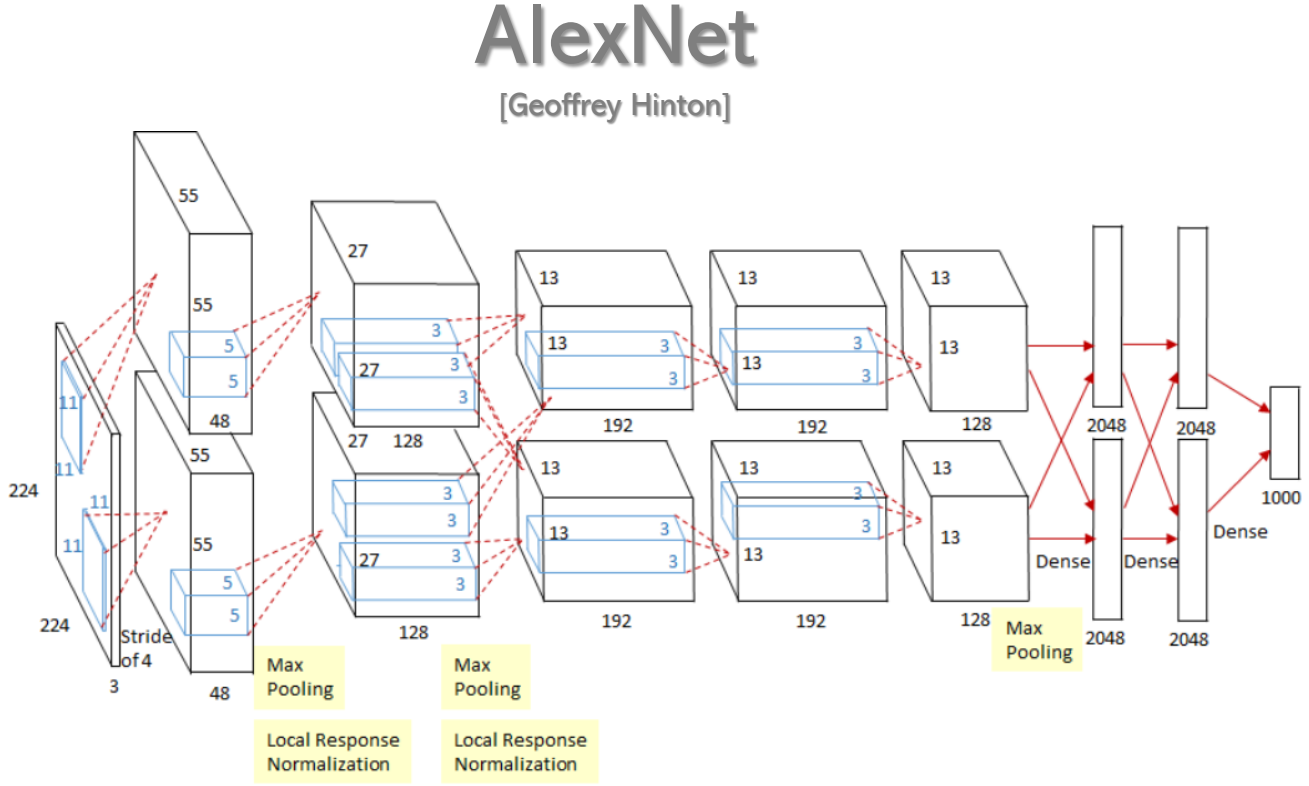




I. Net Framework

A. Classic Frameworks:

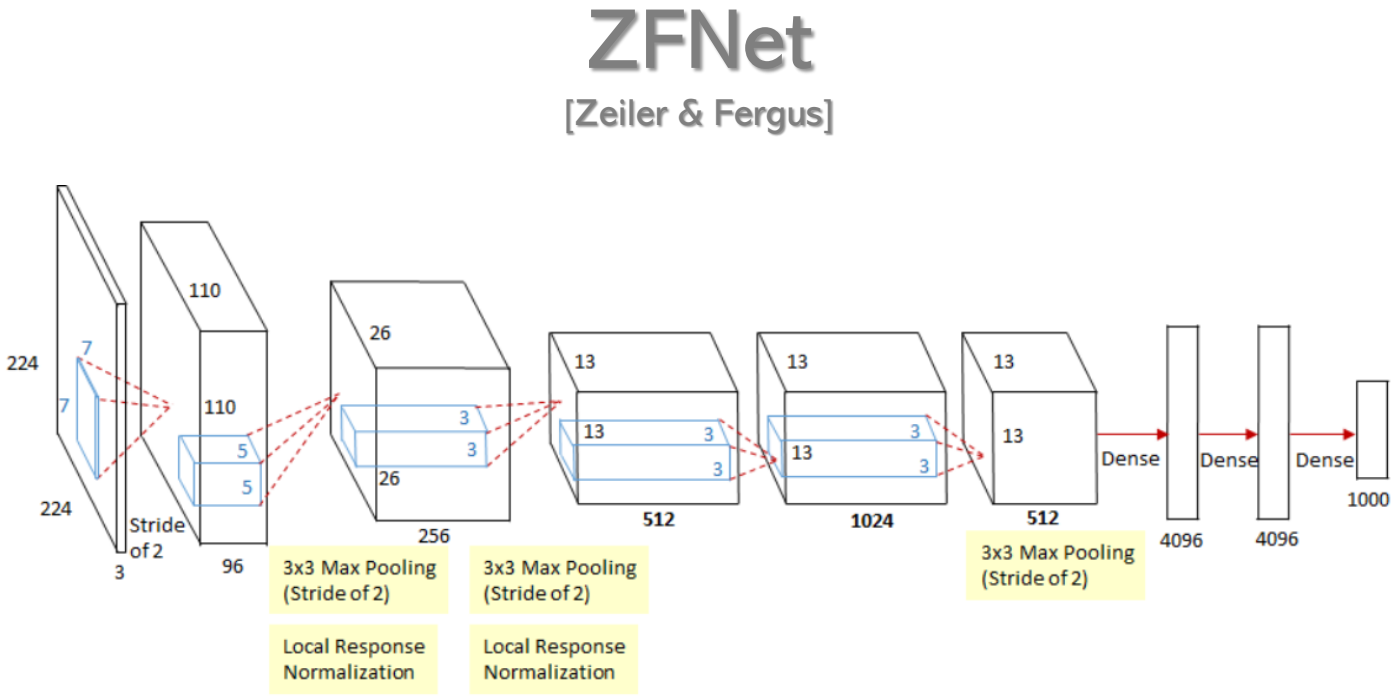
History:



I. Net Framework

A. Classic Frameworks:

History:



I. Net Framework

A. Classic Frameworks:

History:

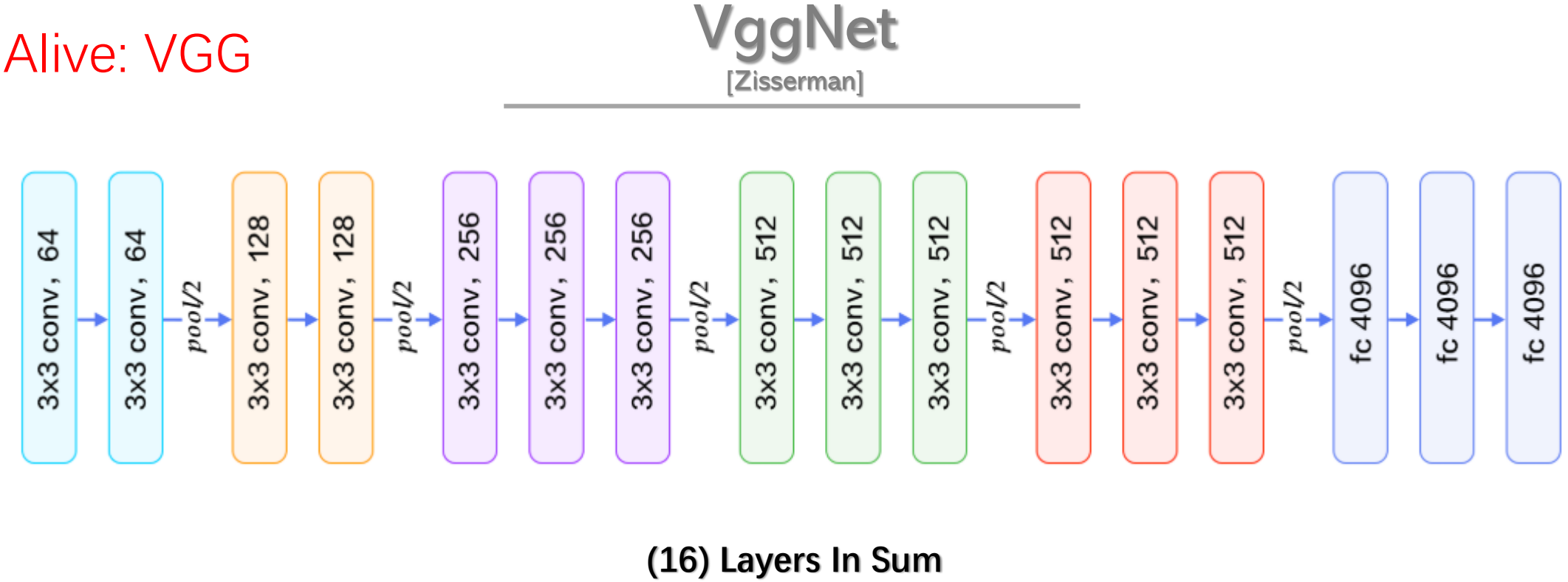
Empirical Formula:

Input – n x (Conv – ReLU – Max Pool) – 1|2 x FC – Output

I. Net Framework

A. Classic Frameworks:

Alive: VGG

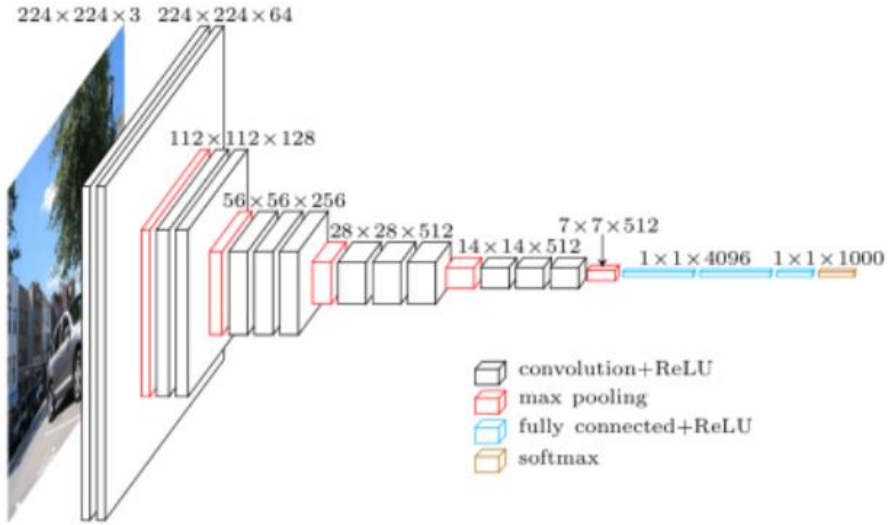


I. Net Framework

A. Classic Frameworks:

Alive: VGG **VggNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

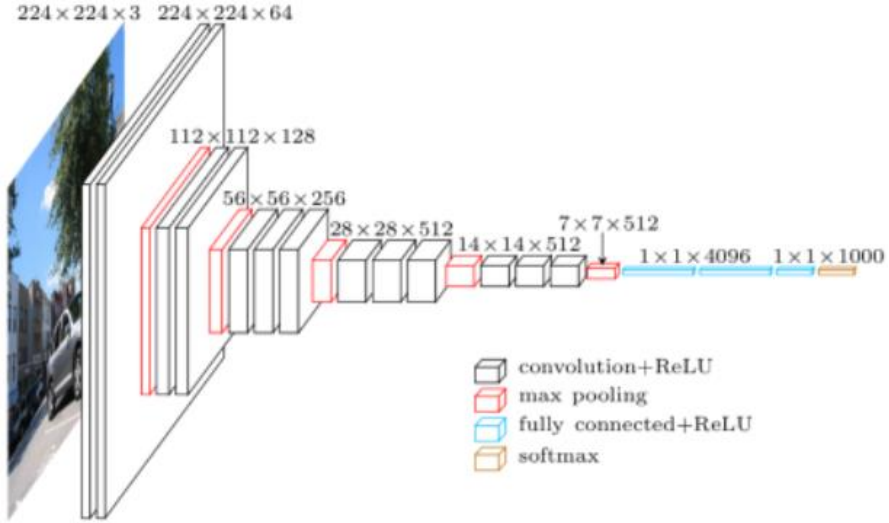


I. Net Framework

A. Classic Frameworks:

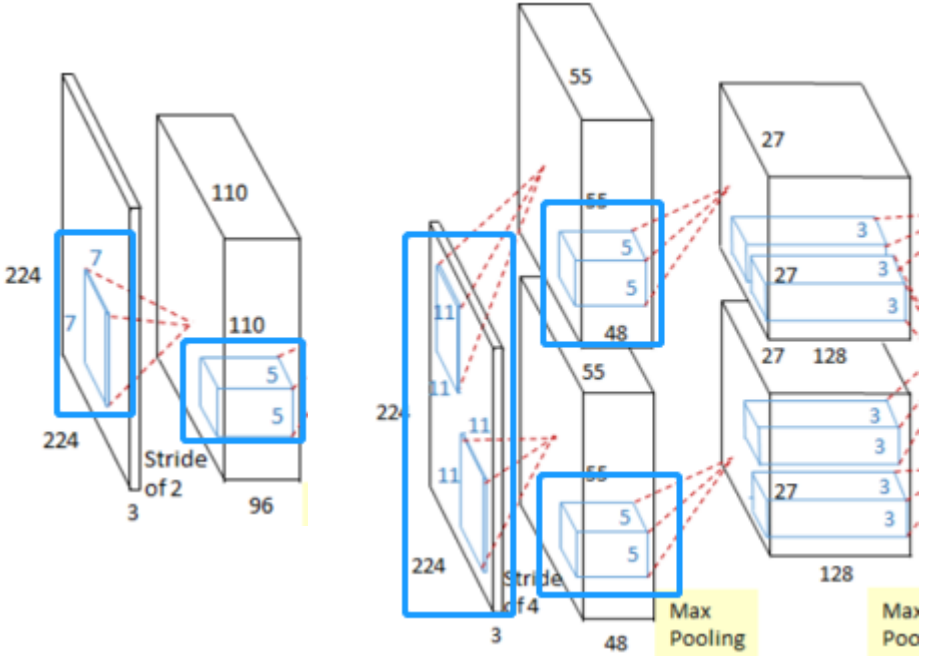
Alive: VGG **VggNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



?

Why 3?

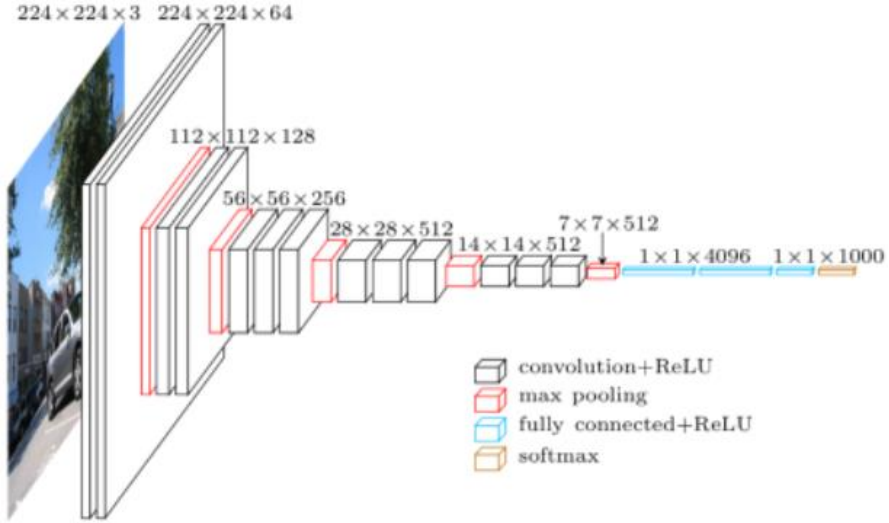


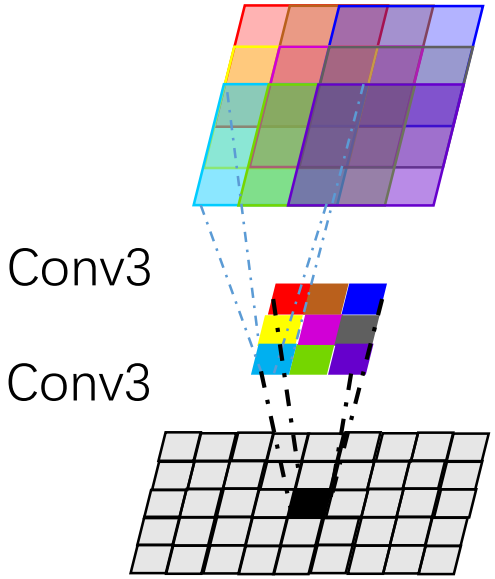
I. Net Framework

A. Classic Frameworks:

Alive: VGG **VggNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					





Receptive Field:
(感受野)

The region of the input space that affects a particular unit of the network.

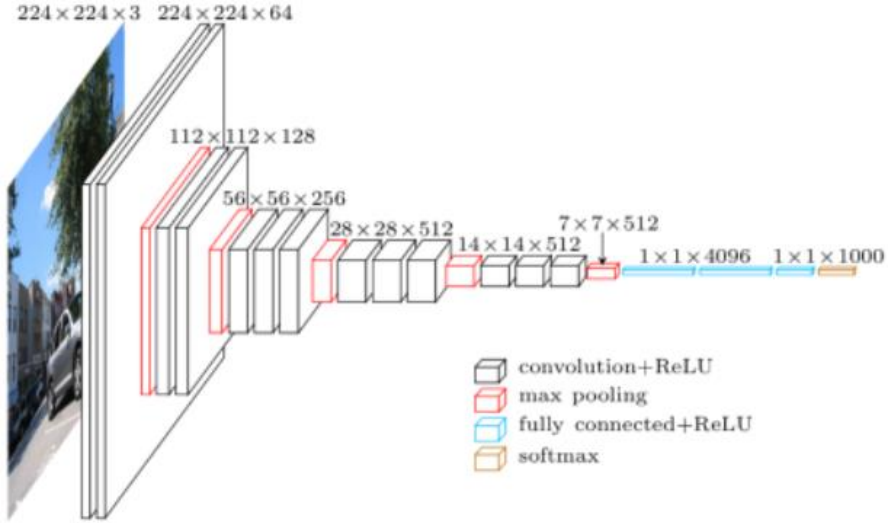
当前元素代表上层特征图的分辨率

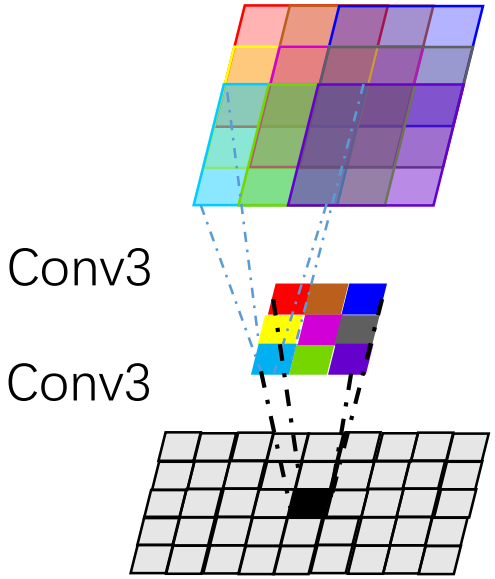
I. Net Framework

A. Classic Frameworks:

Alive: VGG **VggNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					





Upper Layer

Get Receptive Field

Get Feature Map

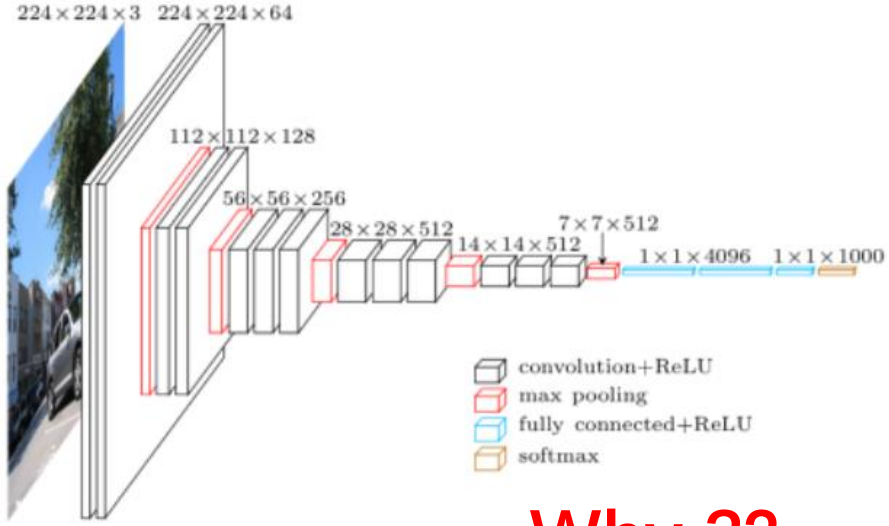
Lower Layer

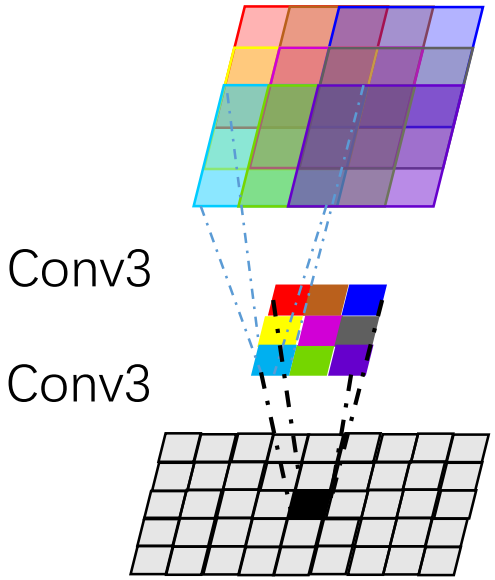
I. Net Framework

A. Classic Frameworks:

Alive: VGG **VggNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					





Why 3?

RF(2 x Conv3) = RF(1 x Conv5)
RF(3 x Conv3) = RF(1 x Conv7)

To save the kernel:

3xConv3 = 3x3xCx3 = 27C
1xConv7 = 7x7xCx1 = 49C

Conclusion:

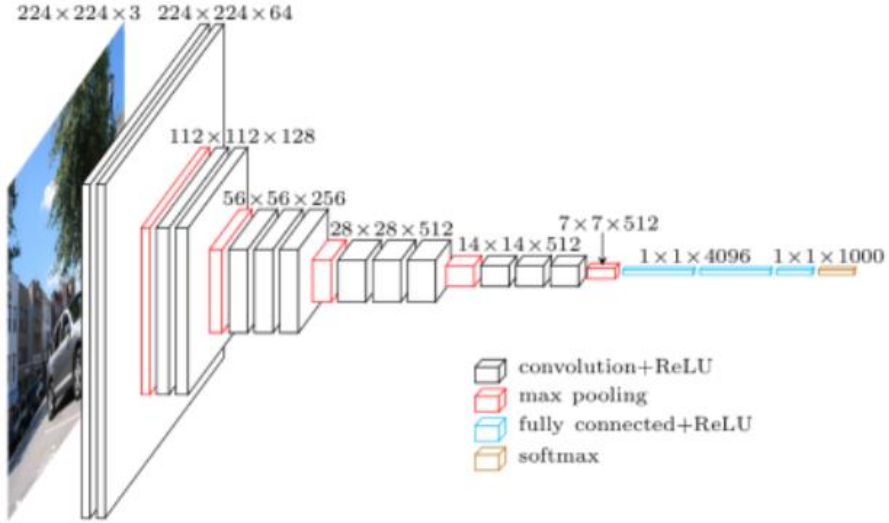
Small kernel save space;
Cheaper to use;
Has same RF with big kernel;
Friendly to Winograd

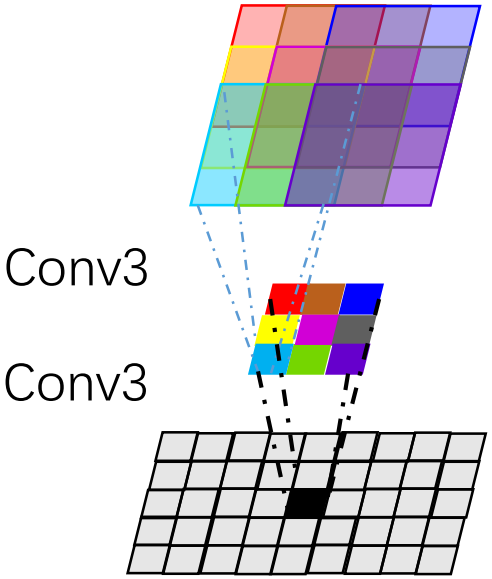
I. Net Framework

A. Classic Frameworks:

Alive: VGG **VggNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					





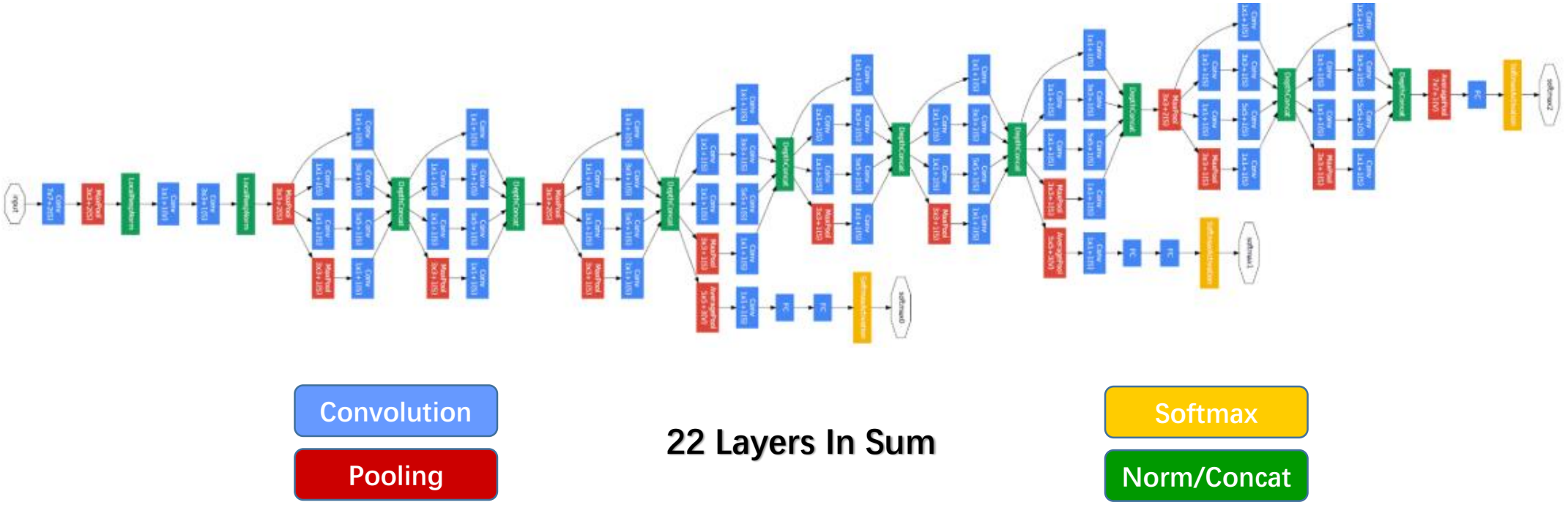
Small kernel always better?

We'll answer this in ResNet.

I. Net Framework

A. Classic Frameworks:

Alive: GoogLeNet

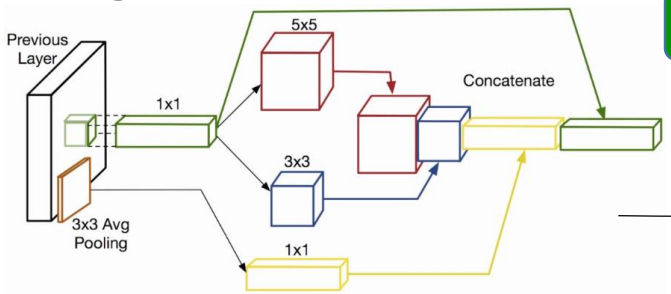


I. Net Framework

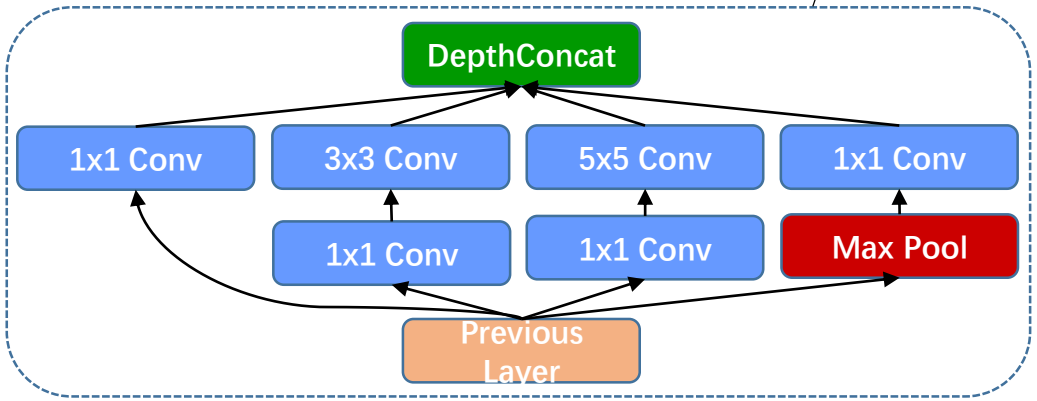
A. Classic Frameworks:

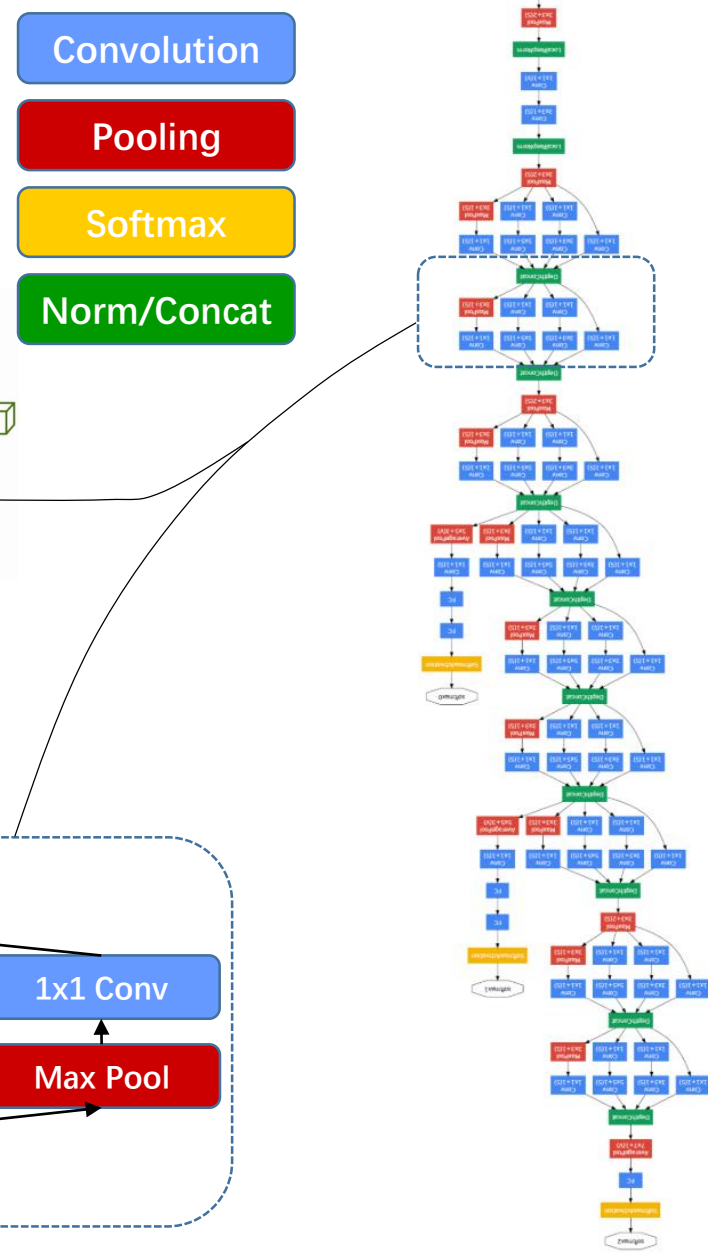
Alive: GoogLeNet

1. **1x1 Conv:**
2. **Inception Module:**
3. **Global Ave Pool:**
4. **Auxiliary Softmax:**



GoogLeNet





I. Net Framework

A. Classic Frameworks:

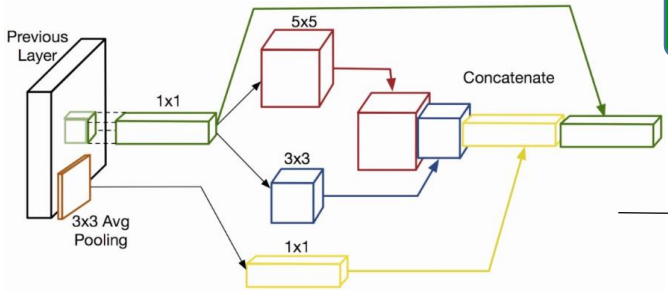
Alive: GoogLeNet

1. **1x1 Conv:**
   reduce dimension

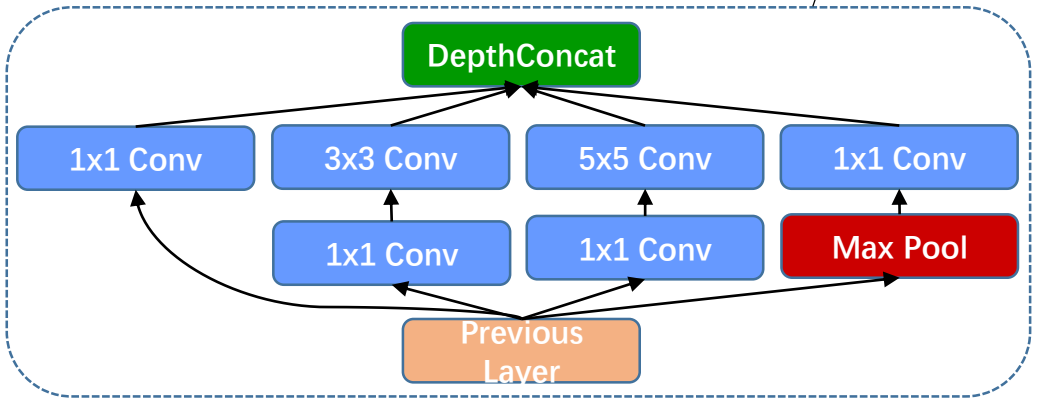
2. **Inception Module:**
   multiple resolution

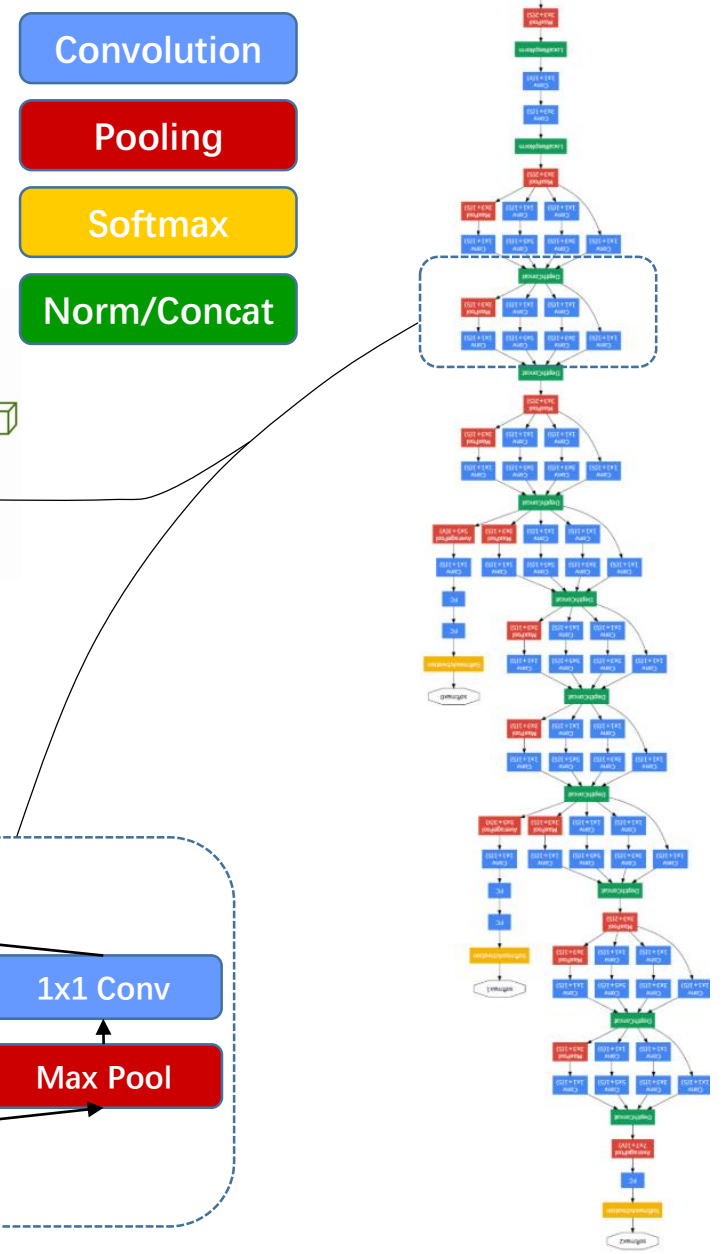
3. **Global Ave Pool:**
   better result than FC

4. **Auxiliary Softmax:**
   weighted assemble loss
   less overfit



GoogLeNet





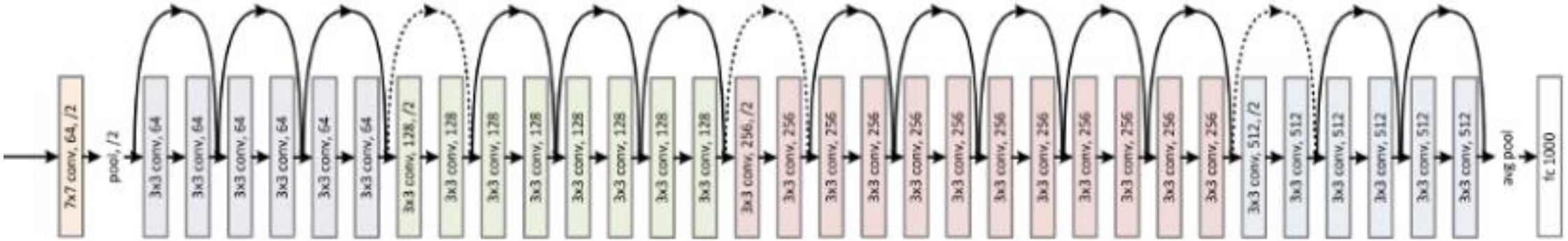
I. Net Framework

A. Classic Frameworks:

Alive: ResNet

ResNet

[Kaiming He]



(34) Layers In Sum

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

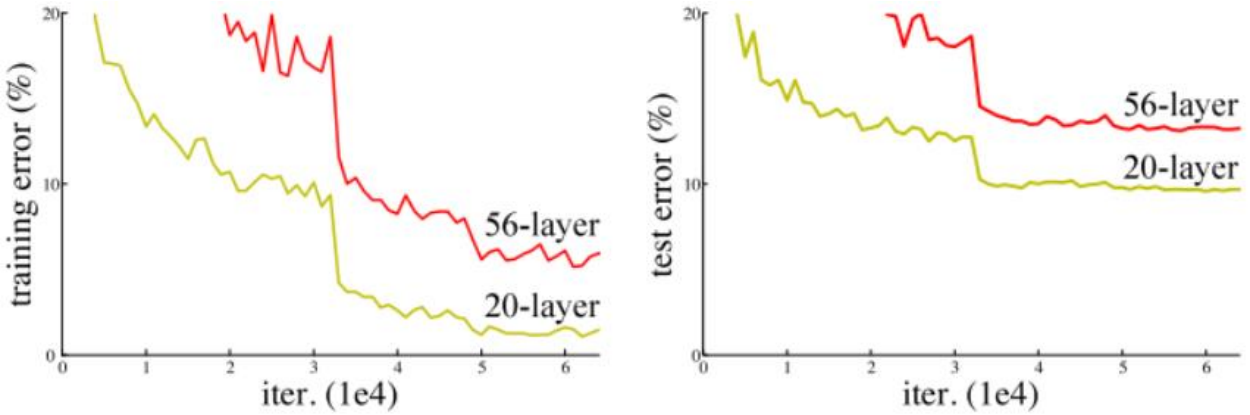
Why ResNet

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Why ResNet



The deeper the network,
The more parameters,
The worse results we'll get

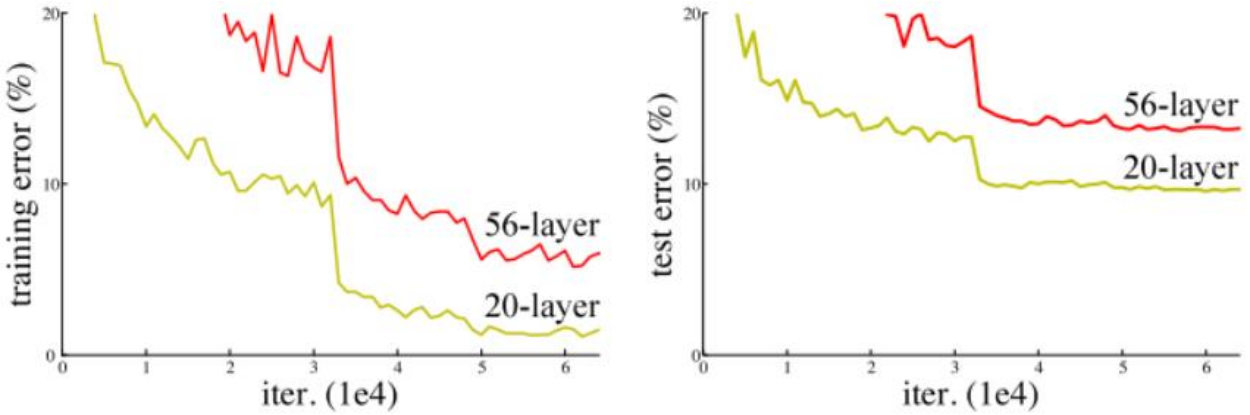
WERIED!!!

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Why ResNet



The deeper the network,
The more parameters,
The worse results we'll get

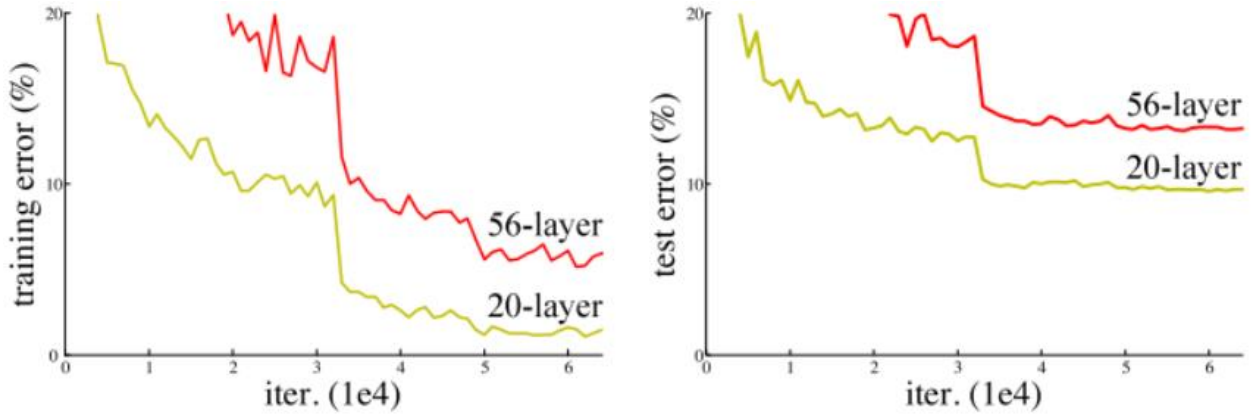
WHY???

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Why ResNet



The deeper the network,
The more parameters,
The worse results we'll get

Too deep, accuracy saturated

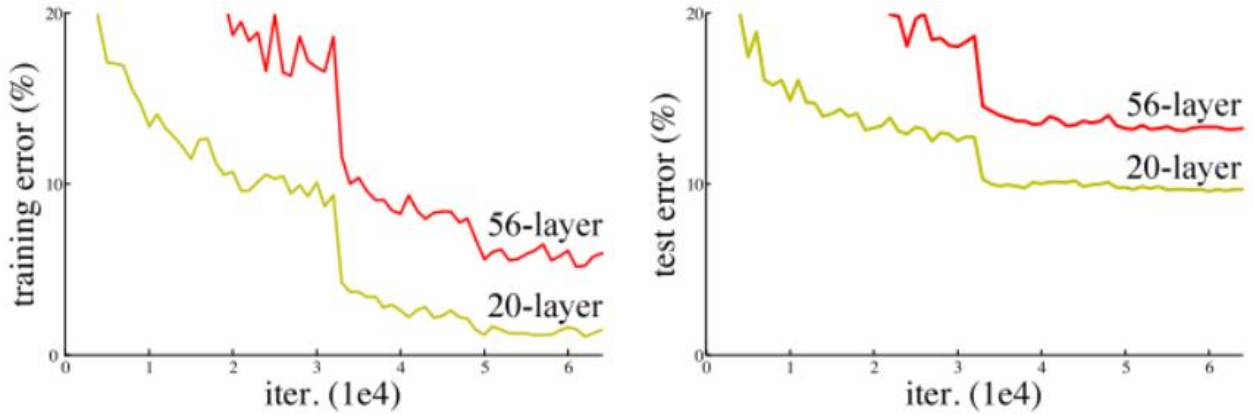
Gradient vanished

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Why ResNet



The deeper the network,
The more parameters,
The worse results we'll get

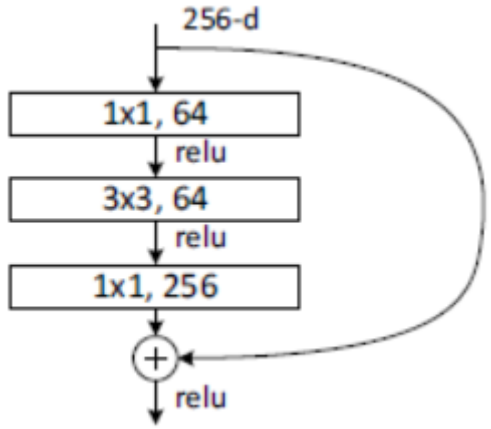
How???

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Botteneck

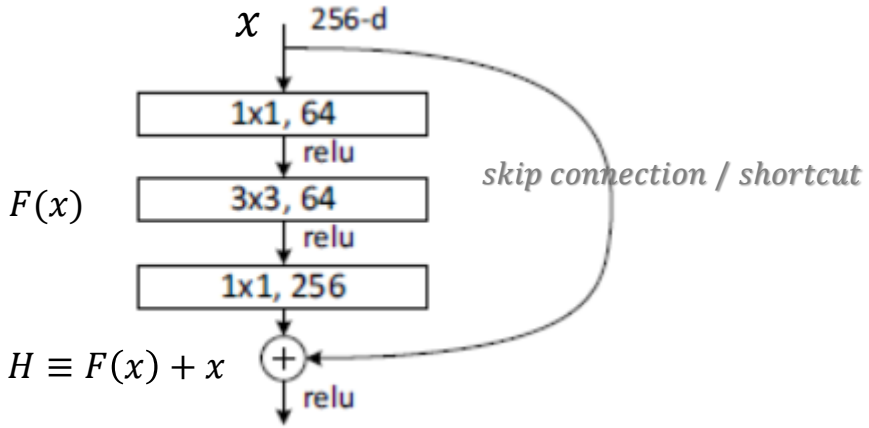


I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Botteneck



$$\frac{\partial H}{\partial x} = F'(x) + \mathbf{1}$$

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Structure

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Structure: 2 Qs

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

1. How

I. Net Framework

A. Classic Frameworks:

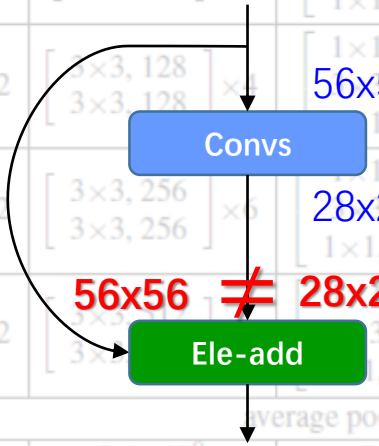
Alive: ResNet

Structure: 2 Qs

We can use conv/pool to reduce the size, What about the shortcut?

How can we add 2 parts without same resolution?

1. How



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

I. Net Framework

A. Classic Frameworks:

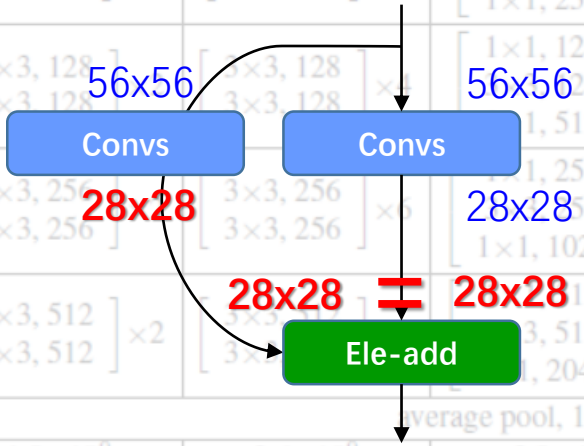
Alive: ResNet

Structure: 2 Qs

We can use conv/pool to reduce the size,
What about the shortcut?

How can we add 2 parts without same resolution?

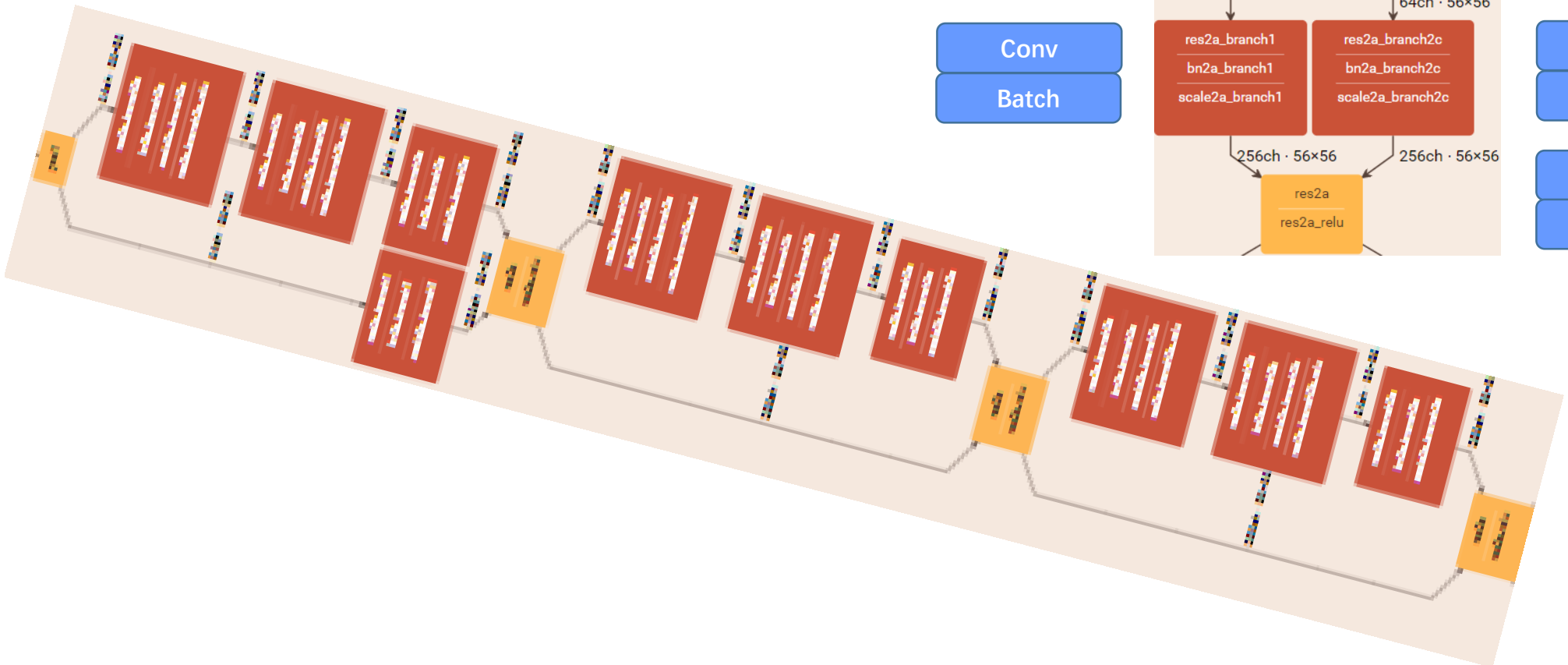
1. How

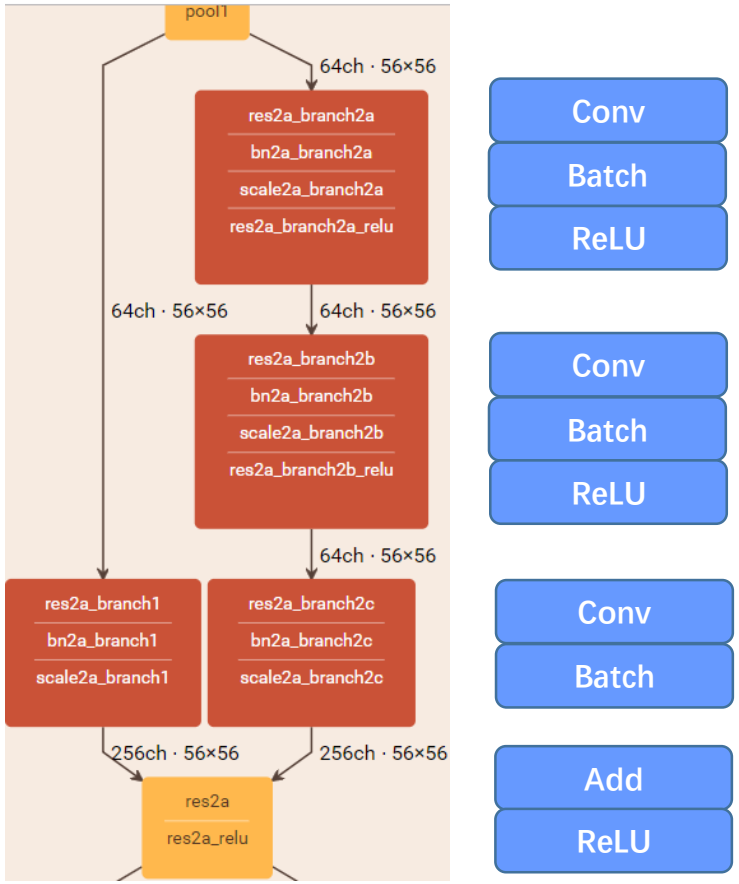


I. Net Framework

A. Classic Frameworks:

Alive: ResNet





Conv
Batch
ReLU

Conv
Batch
ReLU

Conv
Batch

Conv
Batch

Add
ReLU

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Structure: 2 Qs

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

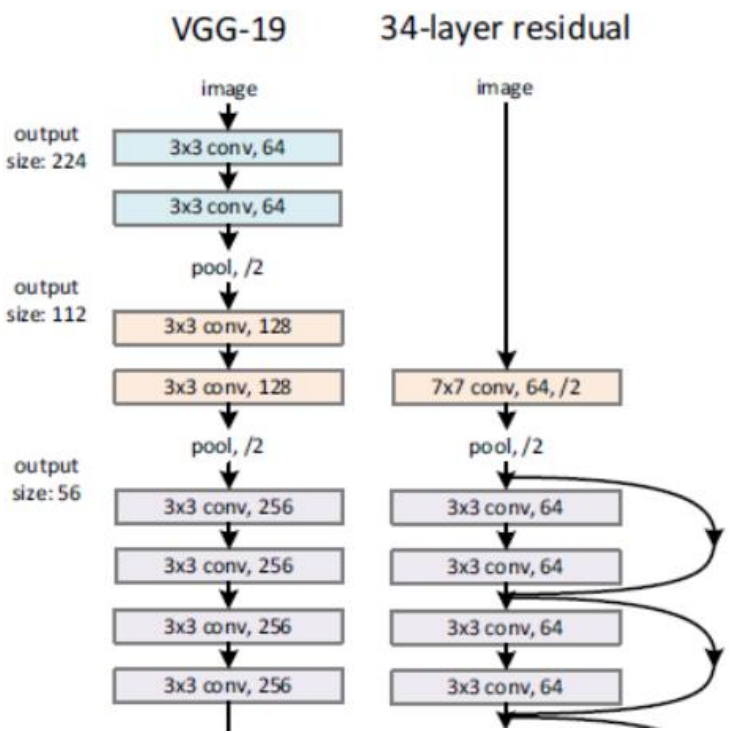
2. Why

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Structure: 2 Qs



1. **Get rich set of primary features**

2. **Channel of input layer is less,**
   big kernel doesn't have to mean great # of params

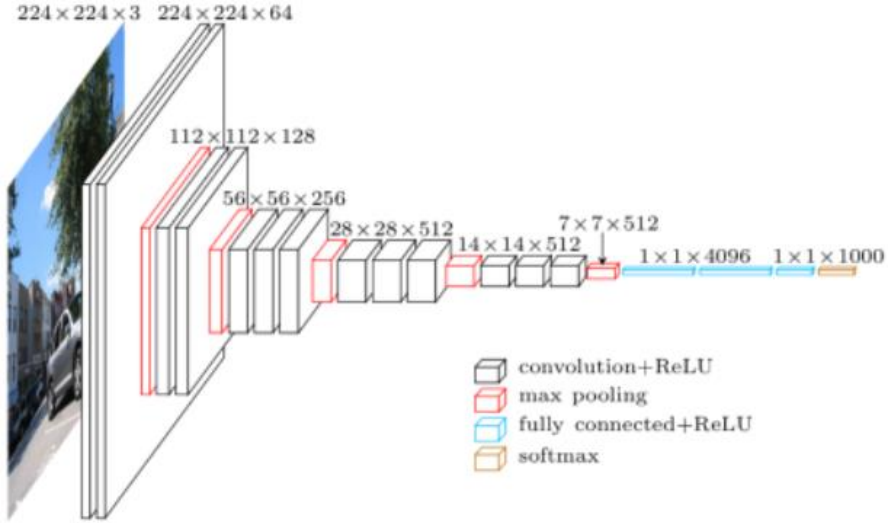
3. **Computing reason:**
   Res: 7x7x3x64x112x112 = 120M
   VGG: 3x3x3x64x224x224+
   3x3x64x64x224x224+
   3x3x64x128x112x112+
   3x3x128x128x56x56
   = x >> 120M x 2

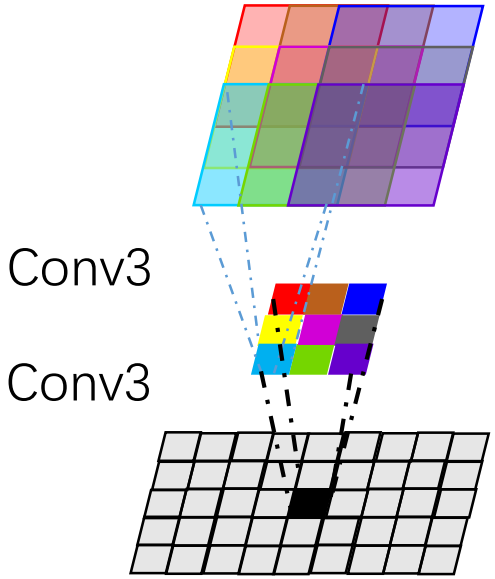
I. Net Framework

A. Classic Frameworks:

Alive: VGG **VggNet**

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					





Small kernel always better?

NO

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Structure: Revisit

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

I. Net Framework

A. Classic Frameworks:

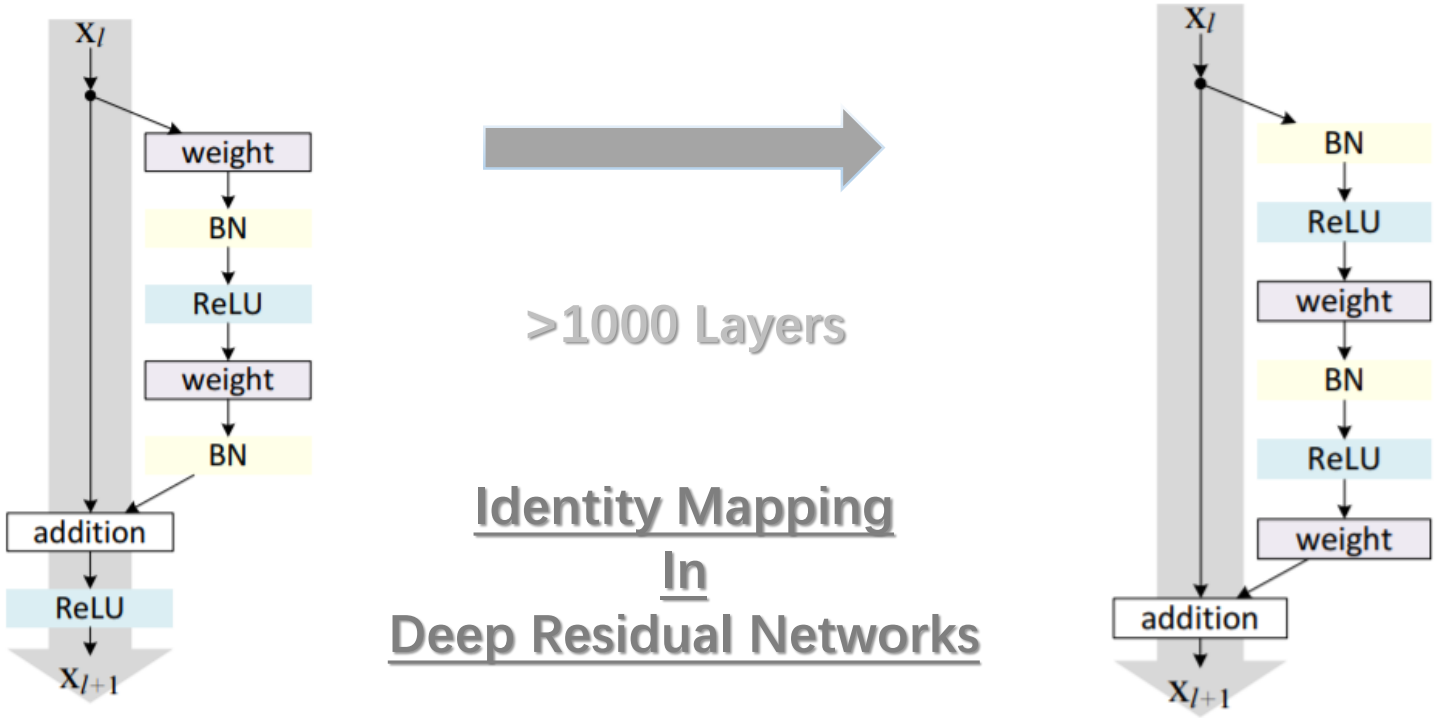
Alive: ResNet **Structure: Advanced?**

I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Structure: Advanced?

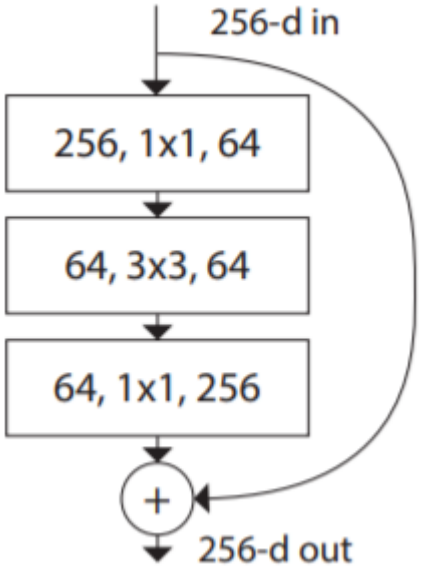


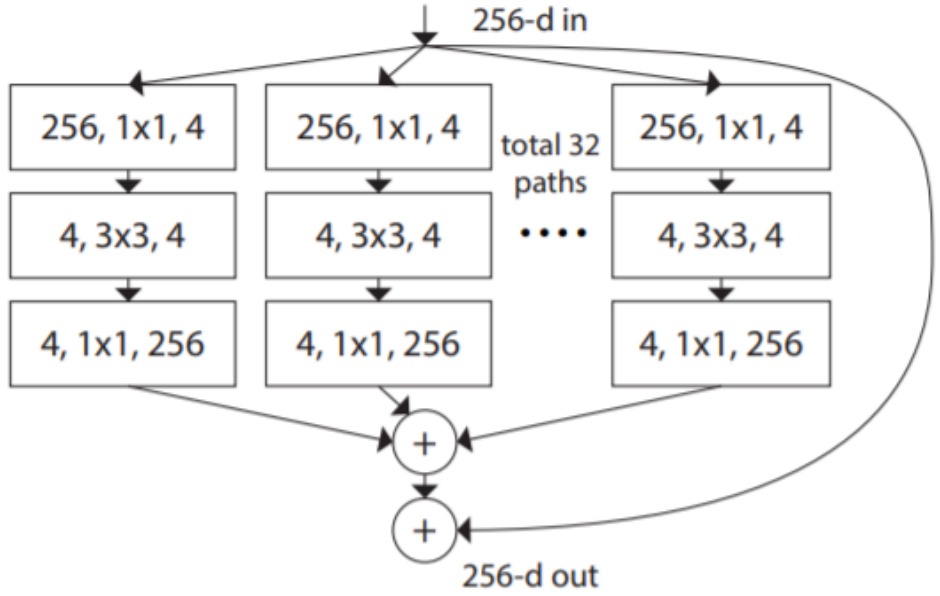
I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Structure: Advanced?





ResNeXt

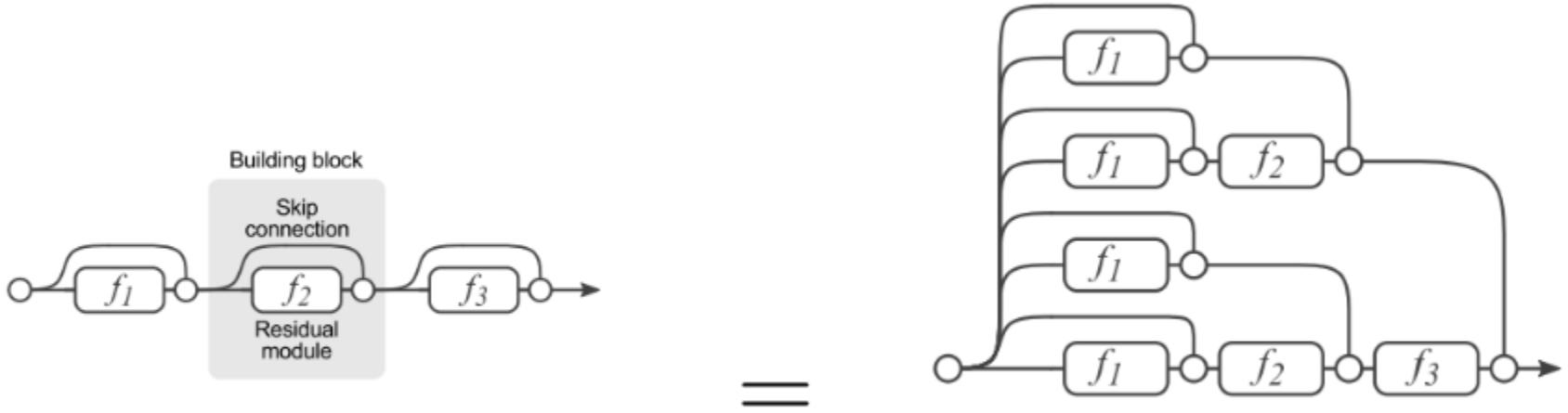
I. Net Framework

A. Classic Frameworks:

Alive: ResNet

Why better?

1. **Solved gradient vanishing by using shortcut**
2. **Can be seen as assembled models**



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

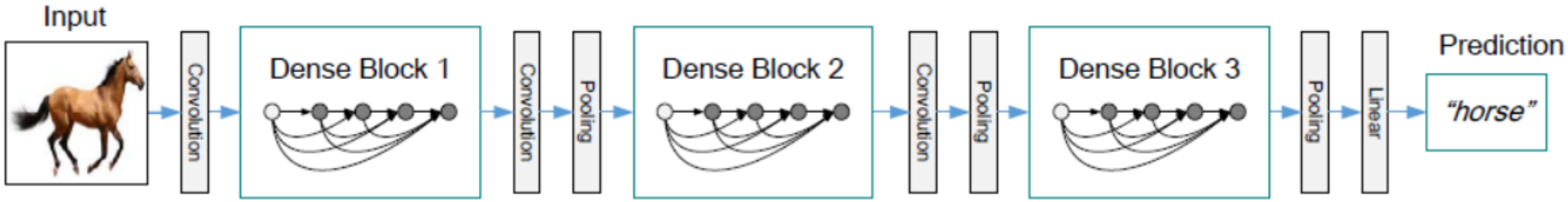
I. Net Framework

A. Classic Frameworks:

Alive: DenseNet

DenseNet

[Gao Huang & Zhuang Liu]

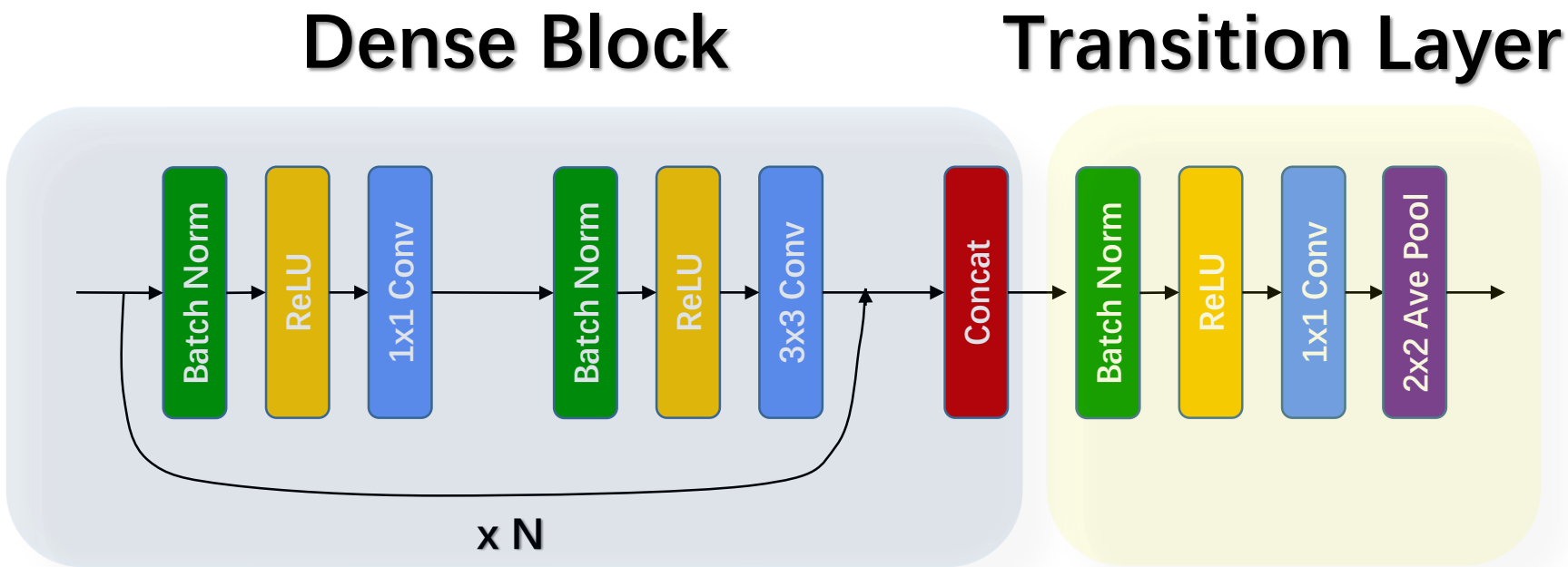


(121) Layers In Sum

I. Net Framework

A. Classic Frameworks:

Alive: DenseNet



I. Net Framework

A. Classic Frameworks:

Alive: DenseNet **Structure**

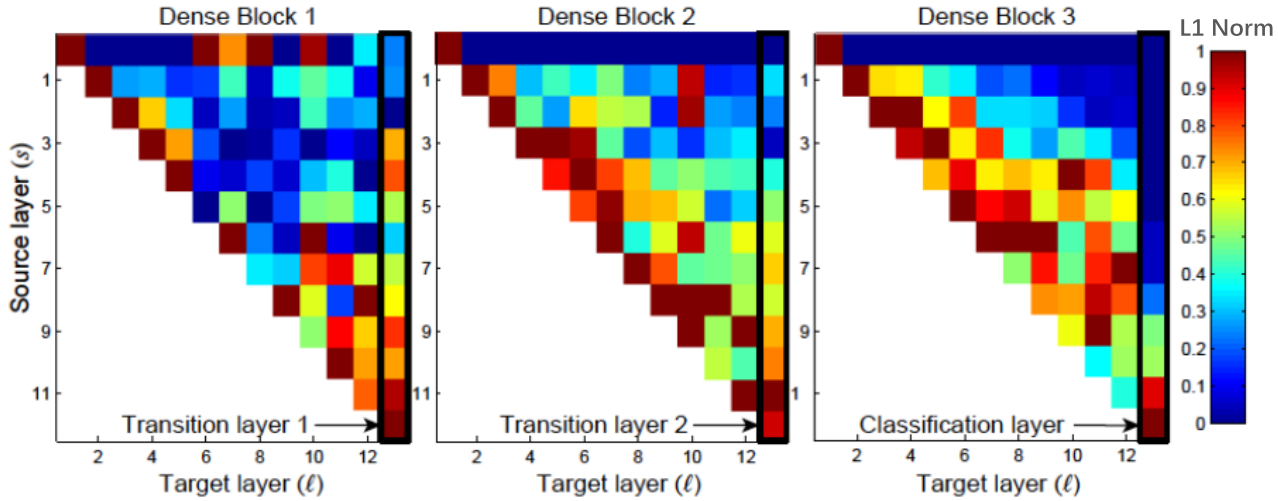
Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

I. Net Framework

A. Classic Frameworks:

Alive: DenseNet

Analysis



The closer the previous layer is,
The more important the layer will be,
The more uses the layer will have.

I. Net Framework

A. Classic Frameworks:

Alive: DenseNet

Pros & Cons

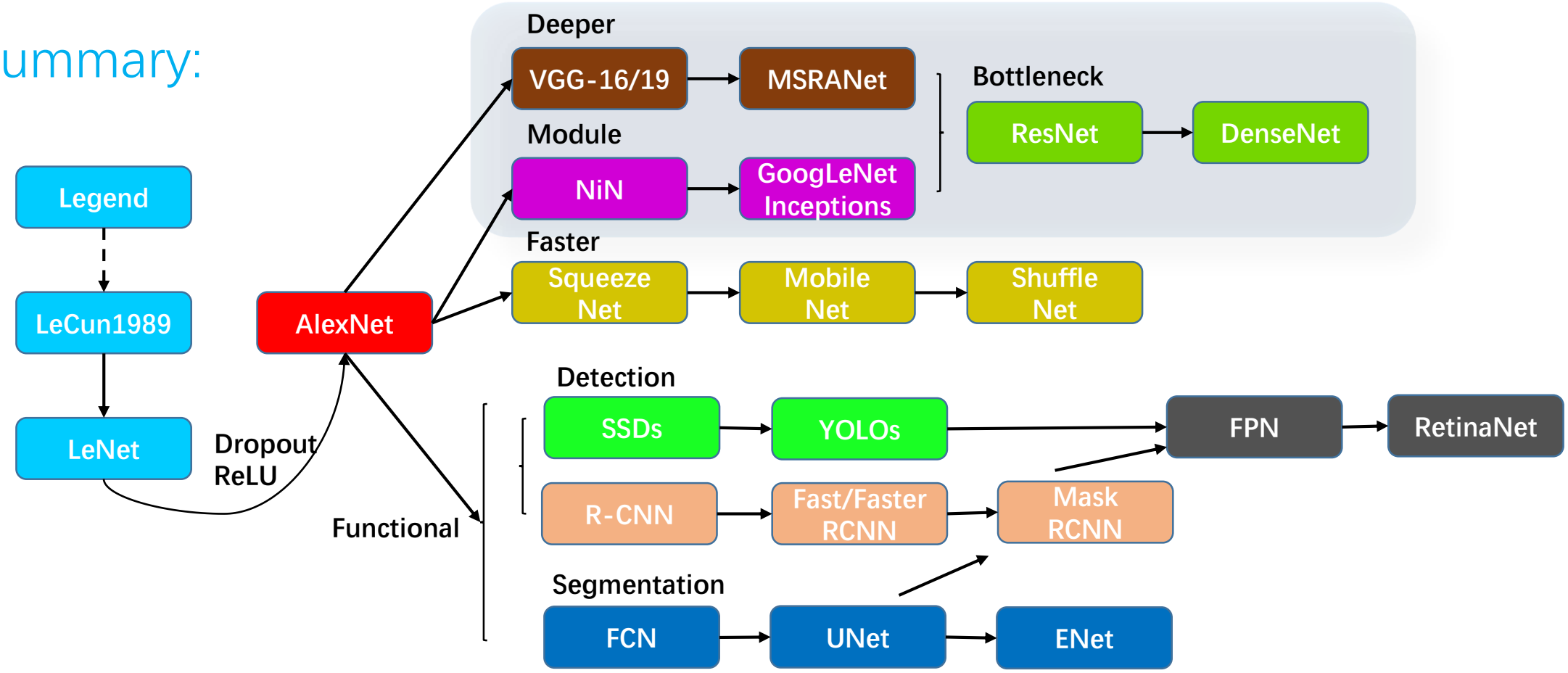
- **Deeper**
- **Fewer params**
- **Better results**

- **Much more memories needed**
  (better not try on your own device)
  [BP needs all layers output]

I. Net Framework

A. Classic Frameworks:

Summary:



I. Net Framework

B. Light Frameworks :

Problems: Too big (Memory)
Too slow (Speed)

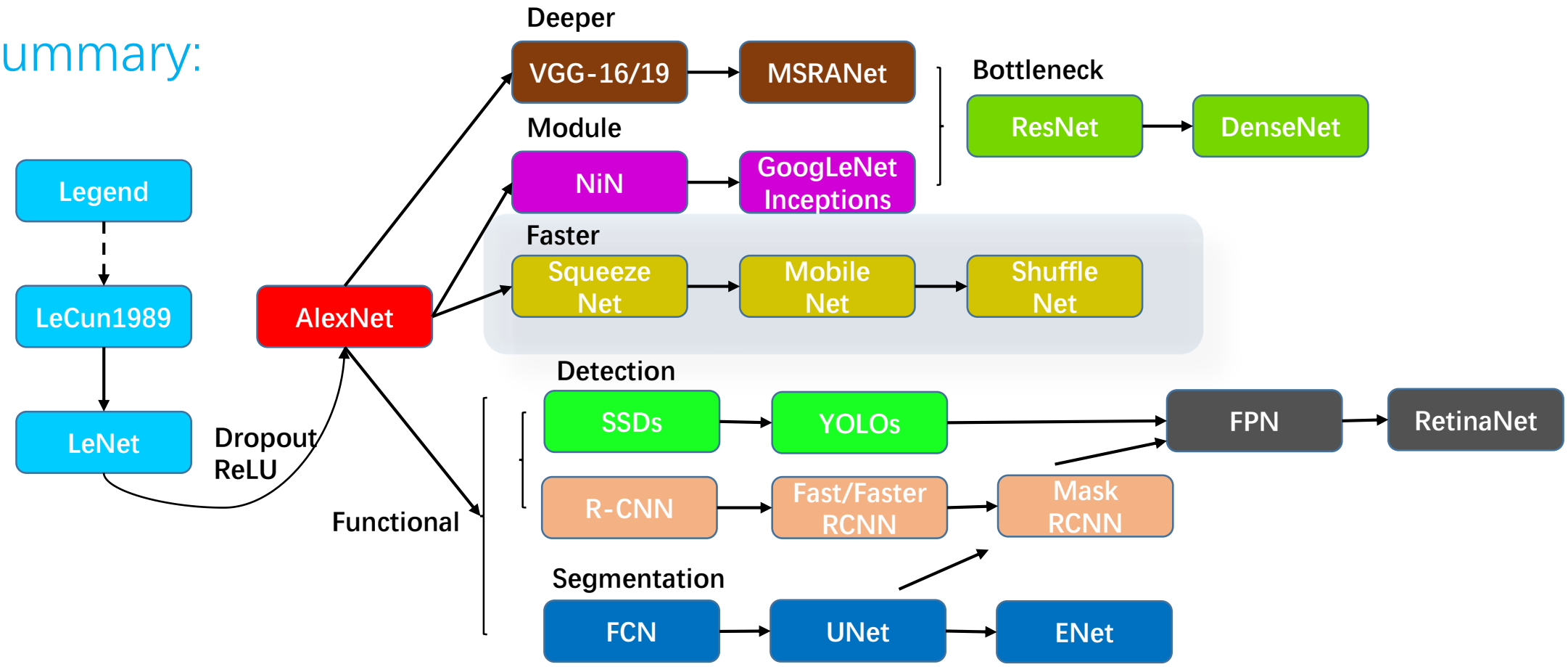
Features: Much less parameters
Much faster
Focus on end devices (i.e. no GPU, mobile phones)

How: Shrink the model
Change the structure

I. Net Framework

A. Classic Frameworks:

Summary:



I. Network Framework

B. Light Frameworks :

History:

[Version 1]

SqueezeNet -> MobileNet -> ShuffleNet -> Xception -> [2016-2017]

[Version 2]

SqueezeNext -> MobileNetV2 -> ShuffleNetV2 -> [2017-2018]

[Version 3]

AutoShuffleNet [2018-2019]

I. Network Framework

B. Light Frameworks :

History:

[Version 1]

SqueezeNet -> MobileNet -> ShuffleNet -> Xception -> [2016-2017]

[Version 2]

SqueezeNext -> MobileNetV2 -> ShuffleNetV2 -> [2017-2018]

[Version 3]

AutoShuffleNet [2018-2019]

I. Network Framework

B. Light Frameworks :

Ideas for acceleration:

- Optimize net framework: Shuffle Net
- Decrease # of parameters: Squeeze Net
- Optimize Conv:
  - Order of computation: Mobile Net
  - Method of computation: Winograd
- Delete FC: Squeeze Net, LightCNN

I. Net Framework

B. Light Frameworks :

B1. SqueezeNet

Ideas (from GoogLeNet):

1. Use more 1 x 1 kernel, LESS AMOUNT of 3 x 3 kernels.
   *. 1x1 can reduce the channel while maintaining the size
2. LESS CHANNEL NUMBER of 3 x 3 kernels.
   *. Reduce parameters' number directly
3. Delay to down sample [晚点儿用pooling]
   *. Increase visual field to get more info.
   A tradeoff for the less amount of parameters.

I. Net Framework

B. Light Frameworks :

B1. SqueezeNet

Structure – Fire Module:

Some Details:

1. $S_1 < E_1 = E_3$

   SR: $S_1/(E_1 + E_3)$

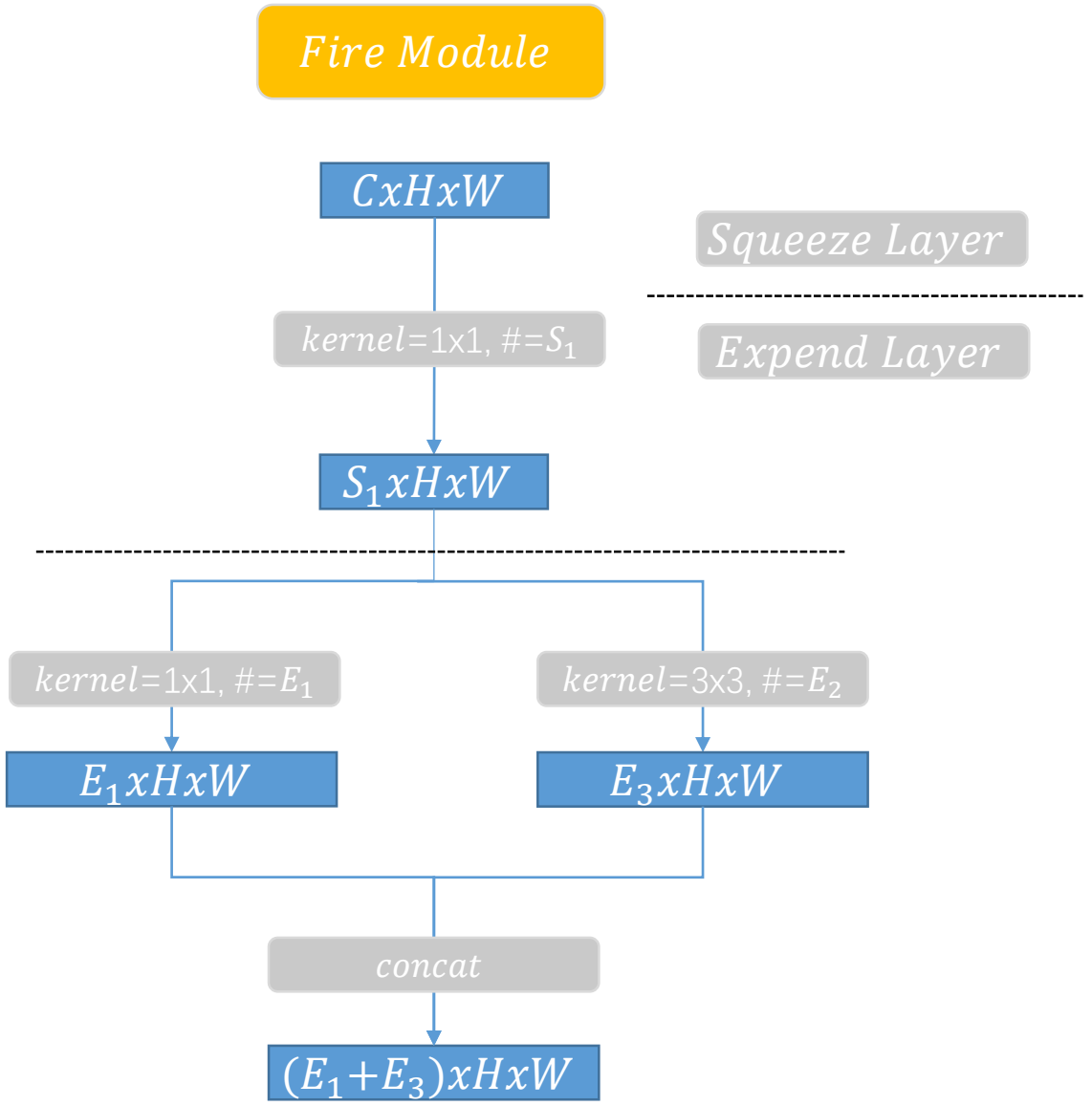
   In paper: $E_1 = E_3 = 4S_1$ [SR=0.125]

   In Experiments: Best in SR=0.75

   (i.e. Increase # of S_1)

2. pct_{3x3}: $E_3/(E_1 + E_3)$

   Best begin from SR=0.5, E_3=0.5



I. Net Framework

B. Light Frameworks :

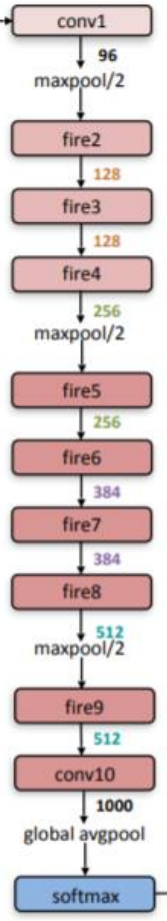
B1. SqueezeNet

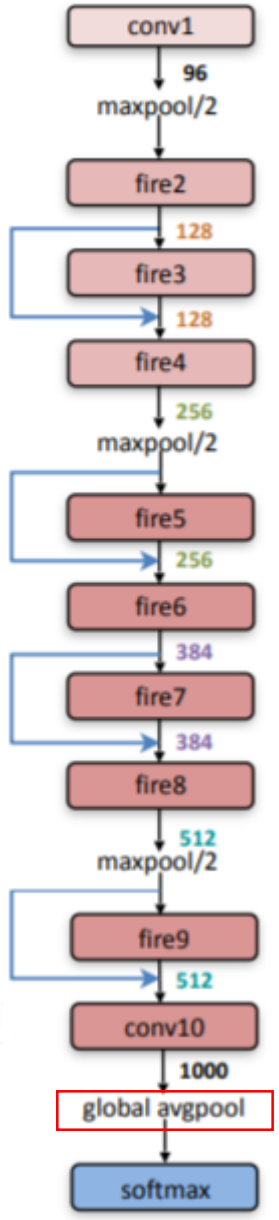
Structure – Macro:

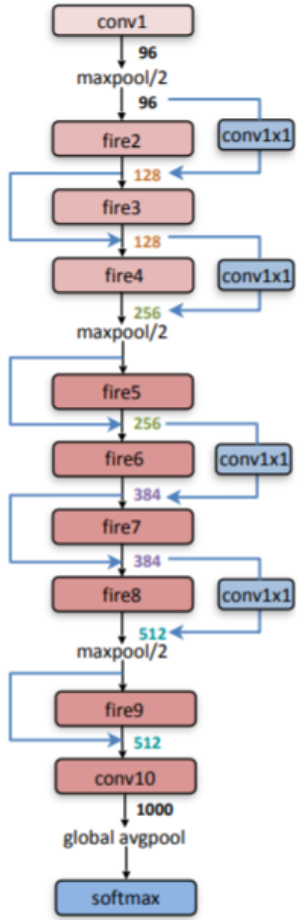
Bottleneck structure is better!

SqueezeNet accuracy and model size using different microarchitecture 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







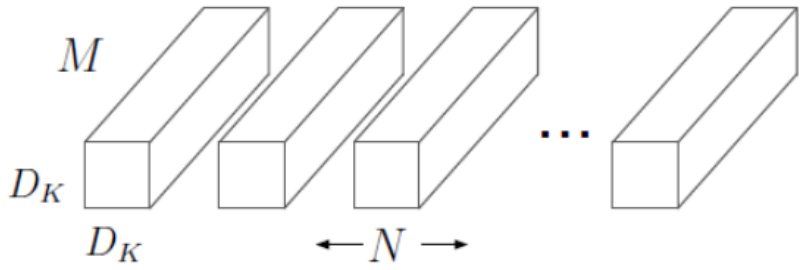
I. Net Framework

B. Light Frameworks :

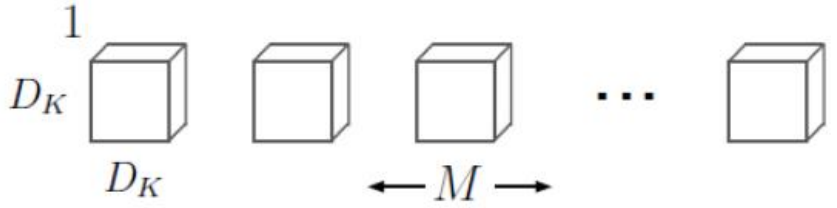
B2.1 MobileNet-V1

Ideas – Depthwise Convolution:

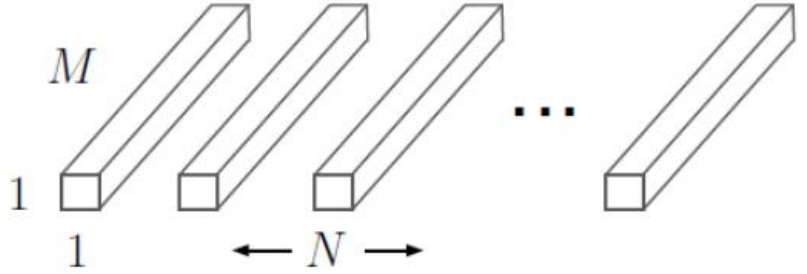
1. Conv in separated channel.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



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

I. Net Framework

B. Light Frameworks :

B2.1 MobileNet-V1

Ideas – Depthwise Convolution:

1. Conv in separated channel.

I. Net Framework

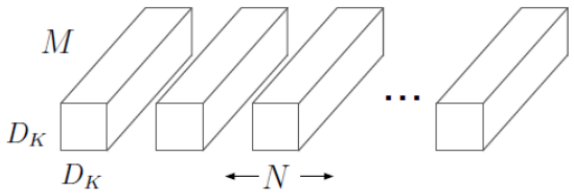
B. Light Frameworks :

B2.1 MobileNet-V1

Ideas – Depthwise Convolution:

1. Conv in separated channel.

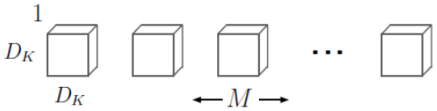
Traditional Convolution:



(a) Standard Convolution Filters

Image size: $D_F \cdot D_F \cdot M$
Kernel size: $D_k \cdot D_k \cdot M \cdot N$
Operations: $D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F \equiv Ori$

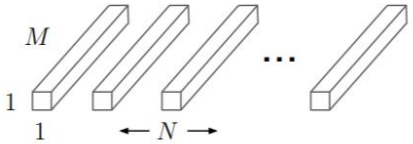
Depthwise Convolution:



(b) Depthwise Convolutional Filters

Step1:
Image size: $D_F \cdot D_F \cdot M$
Depthwise Kernel size: $D_k \cdot D_k \cdot 1 \cdot M$
Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F$

This step "only filters input channels, it does not combine them to create new features." So a 1x1 conv is needed to create new features by computing a linear combination.



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

Step2:
Input size: $D_F \cdot D_F \cdot M$
Pointwise Kernel size: $1 \cdot 1 \cdot M \cdot N$
Operations: $M \cdot N \cdot D_F \cdot D_F$

Total:
Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \equiv Dep$
How Small:
$$\frac{Dep}{Ori} = \frac{1}{N} + \frac{1}{D_k^2}$$

I. Net Framework

B. Light Frameworks :

B2.1 MobileNet-V1

Ideas – Depthwise Convolution:

1. Conv in separated channel.

Traditional Convolution:

D_F=112, M=64, N=128, $D_k = 3$
Image size: $D_F \cdot D_F \cdot M$
Kernel size: $D_k \cdot D_k \cdot M \cdot N$
Operations:
$D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F = 924844032$

Depthwise Convolution:

Operations:
$D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$=109985792

$$\frac{Dep}{Ori} = \frac{1}{N} + \frac{1}{D_k^2}$$

$$=\frac{109985792}{924844032}$$

=0.1189

Usually, it can be 8~9 times less computation than standard convolutions with only about 1% loss of accuracy.

I. Net Framework

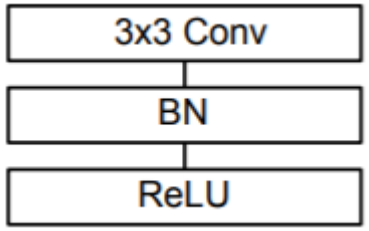
B. Light Frameworks :

B2.1 MobileNet-V1

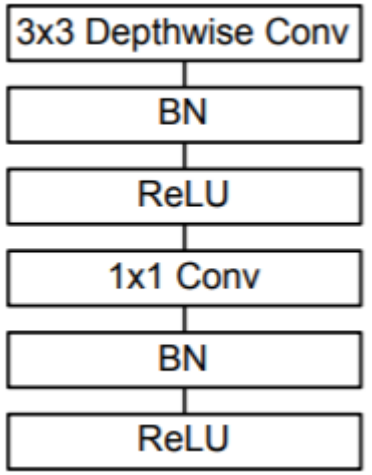
Ideas – Depthwise Convolution:

1. Conv in separated channel.

Traditional Convolution:



Depthwise Convolution:



I. Net Framework

B. Light Frameworks :

B2.1 MobileNet-V1

Structure – Macro:

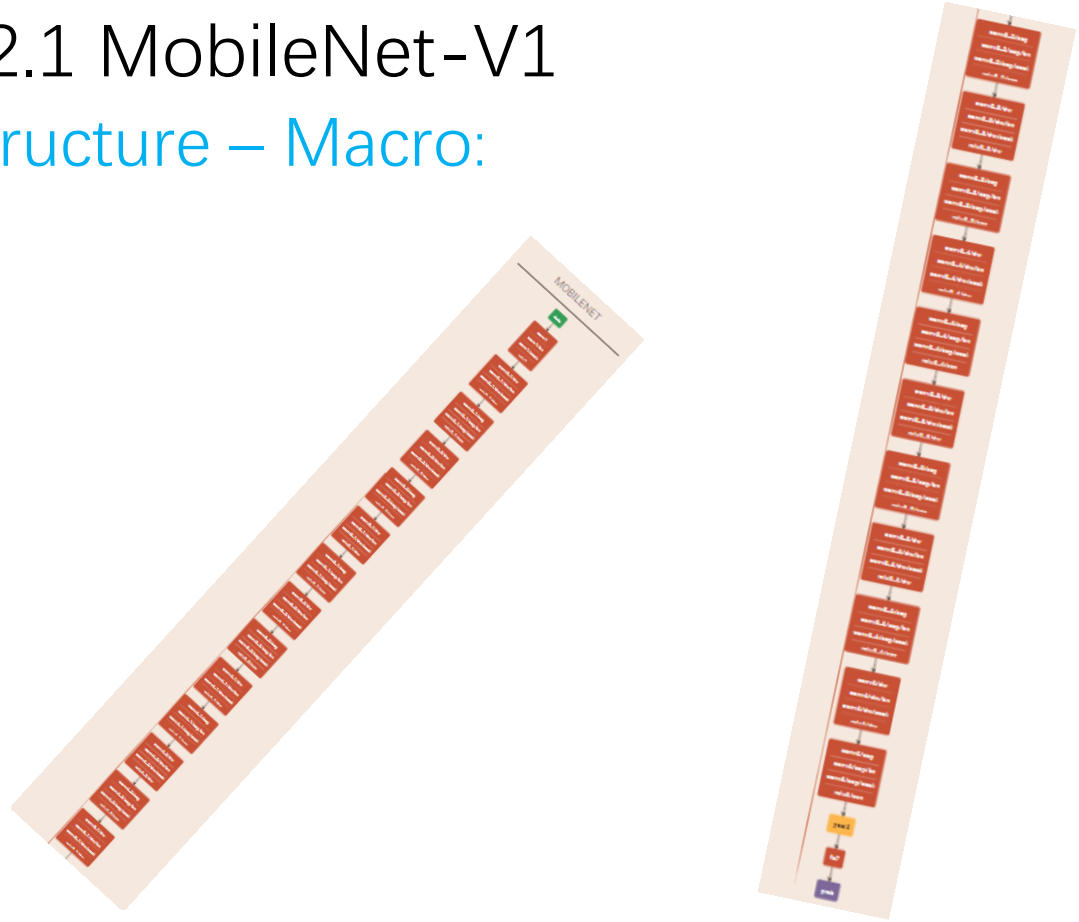


Table 1. MobileNet Body Architecture

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 Conv / s1	$3 \times 3 \times 512$ dw $1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$ $14 \times 14 \times 512$
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	$1 \times 1 \times 1000$

I. Net Framework

B. Light Frameworks :

B2.1 MobileNet-V1

Structure – Hyperparameters:

1. Width Multiplier $\boldsymbol{\alpha}$: Thinner Models

$D_k \cdot D_k \cdot \boldsymbol{\alpha}M \cdot D_F \cdot D_F +$
$\boldsymbol{\alpha}M \cdot \boldsymbol{\alpha}N \cdot D_F \cdot D_F, \quad \boldsymbol{\alpha} \in (0,1]$
Roughly reduce computations by α^2

2. Resolution Multiplier $\boldsymbol{\rho}$: Reduced Representation

$D_k \cdot D_k \cdot \boldsymbol{\alpha}M \cdot \boldsymbol{\rho}D_F \cdot \boldsymbol{\rho}D_F +$
$\boldsymbol{\alpha}M \cdot \boldsymbol{\alpha}N \cdot \boldsymbol{\rho}D_F \cdot \boldsymbol{\rho}D_F, \quad \boldsymbol{\alpha}, \boldsymbol{\rho} \in (0,1]$
Roughly reduce computations by ρ^2 more

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

I. Net Framework

B. Light Frameworks :

B2.1 MobileNet-V1

Training – Some Details:

1. RMSprop

2. Less regularization and data augmentation considering small models have less trouble overfitting

3. Very Little or no weight decay (l2 regularization) on depthwise filters since there are so few parameters in them.

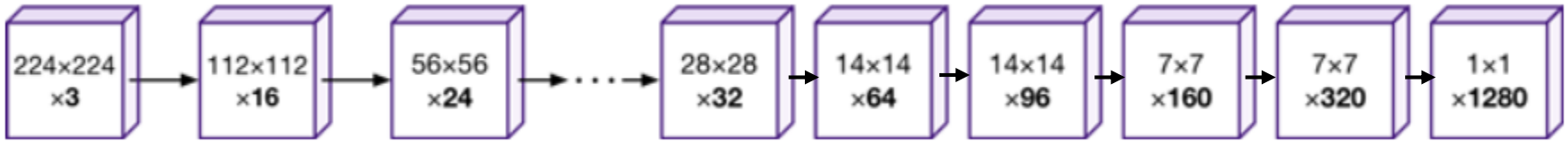
Figure 6. Example objection detection results using MobileNet SSD.

I. Net Framework

B. Light Frameworks :

B2.2 MobileNet-V2

Ideas – Inverted Residuals & Linear Bottlenecks:



First impression: very little parameters, very slim blobs

心灵拷问: Then how could it be better?

人话版:
肯定在每层中间增加channel数来增加参数数量，然后在下一层前再缩回去。

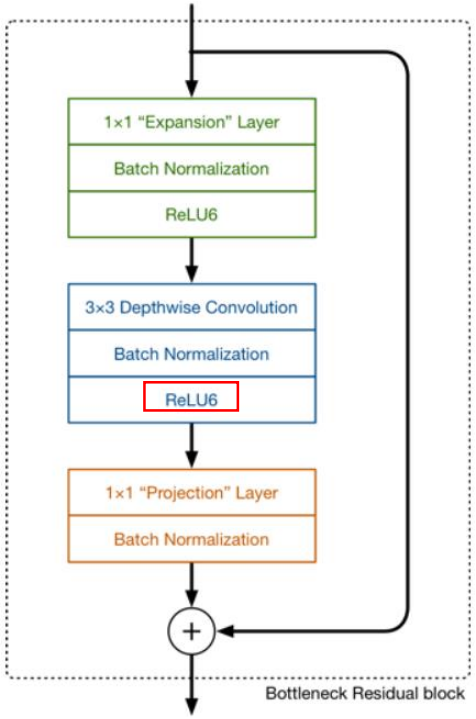
高逼格版:
Inverted Residuals
+
Linear Bottlenecks

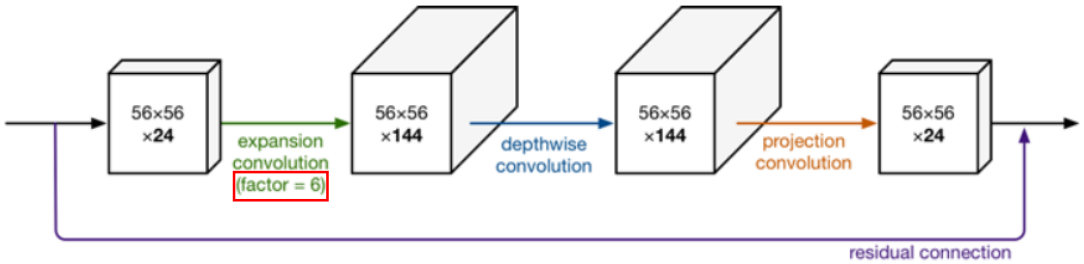
I. Net Framework

B. Light Frameworks :

B2.2 MobileNet-V2

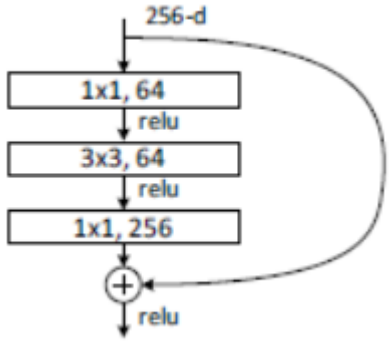
Ideas – Inverted Residuals & Linear Bottlenecks:





Inverted residual: expand - transfer – reduce

Linear Bottlenecks: Eltwise+ with NO ReLU
at the end of a bottleneck



Original residual:
reduce - transfer - expand

Original bottlenecks:
Eltwise+ with ReLU at
the end of a bottleneck

I. Net Framework

B. Light Frameworks :

Manifold：流形
局部具有欧式空间性质的空间

B2.2 MobileNet-V2

Ideas – Inverted Residuals & Linear Bottlenecks:

人话解释：

Inverted Residuals:
Skip connection 这种bottleneck的结构被证明很有效，所以想用；
但是如果像以前那样先压缩channel，channel数本来自就少，再压没了，所以不如先增大再减少。

Linear Bottlenecks:
ReLU让负半轴为0。本来我参数就不多，学习能力就有限，这一下再让一些参数为0了，就更学不着什么东西了。干脆在eltwise+那里不要ReLU了

品质解释：

Inverted Residuals:
作用是uncompress数据，使得我们感兴趣的低维流形(manifold of interest)能够包含于我们的高维空间中。

Linear Bottlenecks:
神经网络被认为可以使MOI嵌入低维空间。如果当前激活空间MOI完整度较高，ReLU会让空间坍塌，丢失信息；并且，其非0部分是做线性变换，实为一个线性分类器。因而采用线性bottleneck

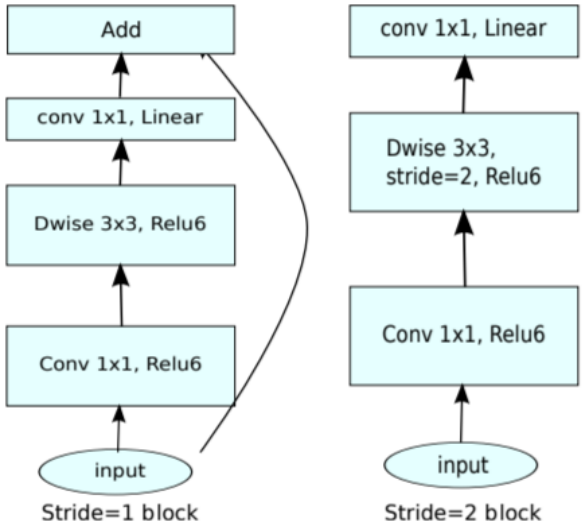
I. Net Framework

B. Light Frameworks :

B2.2 MobileNet-V2

Use "stride 2 block" to reduce dimension. So no skip connection.

Structure:



Use 1x1 conv to reduce dimension. What about pooling along channel?

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	-	

I. Net Framework

B. Light Frameworks :

MAC: Multiply-Accumulates
(Accumulation of + & x)

1.4: width multiplier

B2.2 MobileNet-V2

Effect:

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

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**

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

I. Net Framework

B. Light Frameworks :

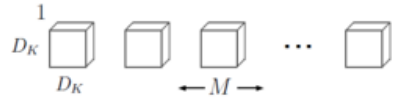
B2.2 MobileNet-V2

Little Question:

How to code with width multiplier parameter?

What is $\alpha = 1.4$ where 1.4M is thus larger than M.

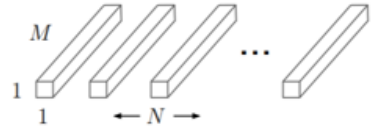
Depthwise Convolution:



(b) Depthwise Convolutional Filters

Step1: Image size: $D_F \cdot D_F \cdot M$
Depthwise Kernel size: $D_k \cdot D_k \cdot 1 \cdot M$
Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F$

This step "only filters input channels, it does not combine them to create new features." So a 1x1 conv is needed to create new features by computing a linear combination.



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

Step2: Input size: $D_F \cdot D_F \cdot M$
Pointwise Kernel size: $1 \cdot 1 \cdot M \cdot N$
Operations: $M \cdot N \cdot D_F \cdot D_F$

Total: Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \equiv Dep$
How Small:
$\frac{Dep}{Ori} = \frac{1}{N} + \frac{1}{D_k^2}$

$$D_k \cdot D_k \cdot \boldsymbol{\alpha} M \cdot D_F \cdot D_F + \boldsymbol{\alpha} M \cdot \boldsymbol{\alpha} N \cdot D_F \cdot D_F, \qquad \boldsymbol{\alpha} \in (0,1]$$

I. Net Framework

B. Light Frameworks :

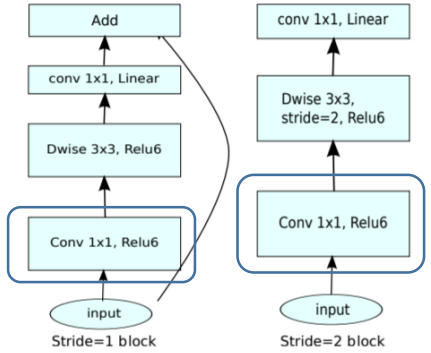
B2.2 MobileNet-V2

Little Question:

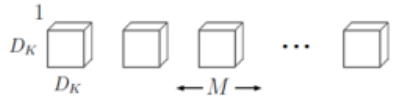
How to code with width multiplier parameter?

What if $\alpha = 1.4$ where 1.4M is thus larger than M.

Answer: Never mind. We have 1x1 conv that we can change to any dimension as we want.



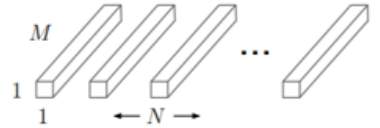
Depthwise Convolution:



(b) Depthwise Convolutional Filters

Step1: Image size: $D_F \cdot D_F \cdot M$
Depthwise Kernel size: $D_k \cdot D_k \cdot 1 \cdot M$
Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F$

This step "only filters input channels, it does not combine them to create new features." So a 1x1 conv is needed to create new features by computing a linear combination.



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

Step2: Input size: $D_F \cdot D_F \cdot M$
Pointwise Kernel size: $1 \cdot 1 \cdot M \cdot N$
Operations: $M \cdot N \cdot D_F \cdot D_F$

Total: Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \equiv Dep$
How Small:
$$\frac{Dep}{Ori} = \frac{1}{N} + \frac{1}{D_k^2}$$

$$D_k \cdot D_k \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F, \qquad \alpha \in (0,1]$$

I. Net Framework

B. Light Frameworks :

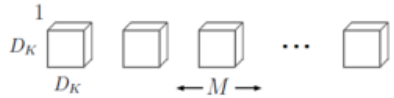
B2.2 MobileNet-V2

Little Question:

How to code with width multiplier parameter?
What if $\alpha = 0.8$ when it's in V1.

Answer: We can change the # of channel at pointwise (2nd) step.

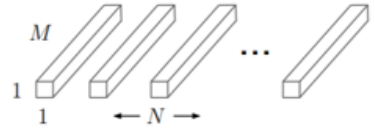
Depthwise Convolution:



(b) Depthwise Convolutional Filters

Step1: Image size: $D_F \cdot D_F \cdot M$
Depthwise Kernel size: $D_k \cdot D_k \cdot 1 \cdot M$
Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F$

This step "only filters input channels, it does not combine them to create new features." So a 1x1 conv is needed to create new features by computing a linear combination.



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

Step2: Input size: $D_F \cdot D_F \cdot M$
Pointwise Kernel size: $1 \cdot 1 \cdot M \cdot N$
Operations: $M \cdot N \cdot D_F \cdot D_F$

Total: Operations: $D_k \cdot D_k \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \equiv Dep$
How Small:
$$\frac{Dep}{Ori} = \frac{1}{N} + \frac{1}{D_k^2}$$

$$D_k \cdot D_k \cdot \boldsymbol{\alpha} M \cdot D_F \cdot D_F + \boldsymbol{\alpha} M \cdot \boldsymbol{\alpha} N \cdot D_F \cdot D_F, \qquad \boldsymbol{\alpha} \in (0,1]$$

I. Net Framework

B. Light Frameworks :

B2.3 MobileNet Summary

Summary:

MobileNet – V1

Mainly separate traditional conv into 2 steps: Depthwise + Pointwise

Accelerate conv operation

MobileNet – V2:

Inverted residual: expand – transfer – reduce

+

Linear bottleneck: remove ReLU after Eltwise+

Add more useful parameters

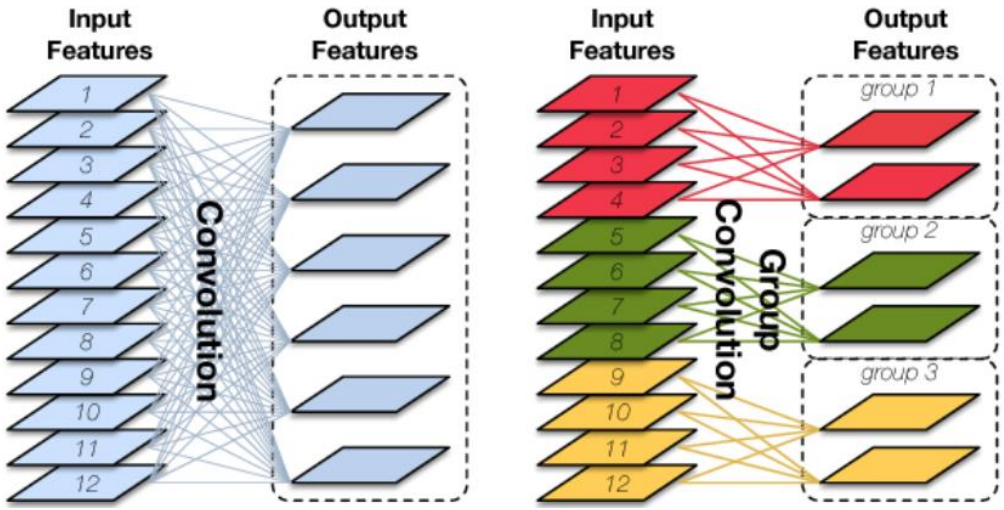
I. Net Framework

B. Light Frameworks :

B3.1 ShuffleNet – V1

Ideas: 1x1 Group Conv

Channel Shuffle



Why Group Conv?

A. Because of less computation

Standard Conv:
Image size: $D_F \cdot D_F \cdot M$
Kernel size: $D_k \cdot D_k \cdot M \cdot N$
Operations: $D_k \cdot D_k \cdot M \cdot N \cdot D_F \cdot D_F \equiv Ori$

Group Conv:
Image size: $D_F \cdot D_F \cdot M$
Kernel size: $D_k \cdot D_k \cdot \frac{M}{g} \cdot \frac{N}{g}$
Operations: $D_k \cdot D_k \cdot \frac{M}{g} \cdot \frac{N}{g} \cdot D_F \cdot D_F \cdot g = \frac{Ori}{g} \equiv GConv$

Why 1x1 Conv?

A. For channel combining

Why 1x1 GConv?

A. Faster combining channels

Anything need to be improved?

A. Yes. Because each group of feature is only a fractional response of the input. So we need to fix it.

I. Net Framework

B. Light Frameworks :

B3.1 ShuffleNet – V1

Ideas: 1x1 Group Conv

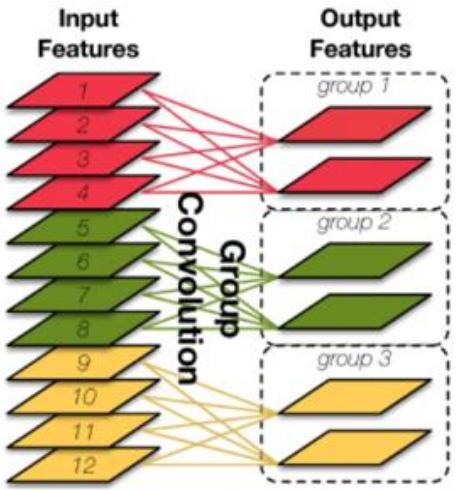
Channel Shuffle

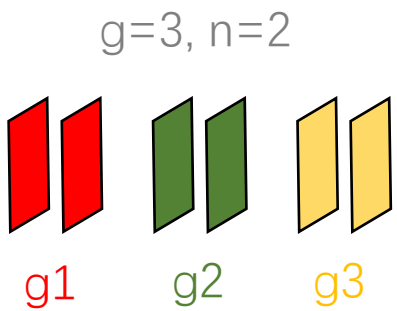
Anything need to be improved?

A. Yes. Because each group of feature is only a fractional response of the input. So we need to fix it.

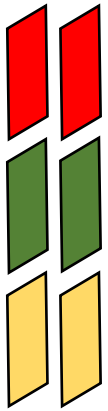
How to improve? / to shuffle channel?

A. Use channel shuffle tech. After doing GConv, we shuffle the feature maps

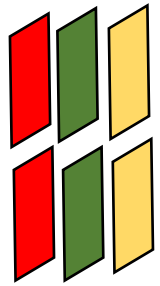




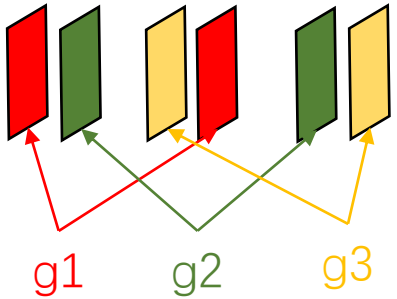
s1: channel=gxn



s2: reshape to [g,n]



s3: transpose to [n,g]



s4: flatten [n,g]

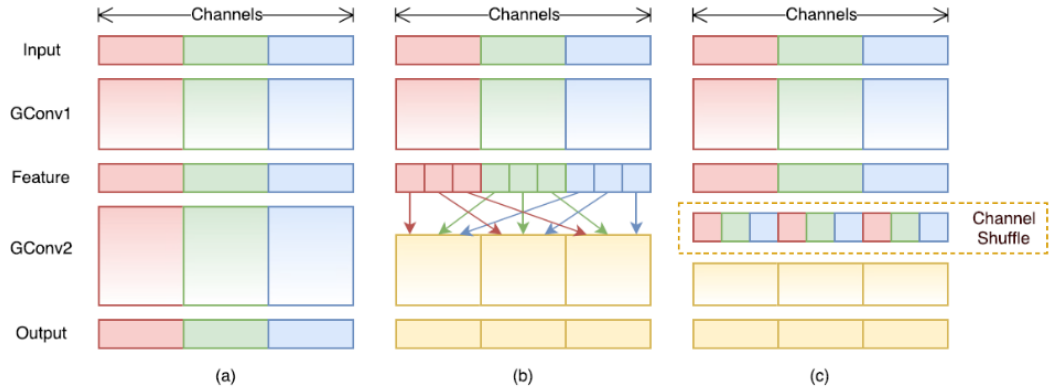
I. Net Framework

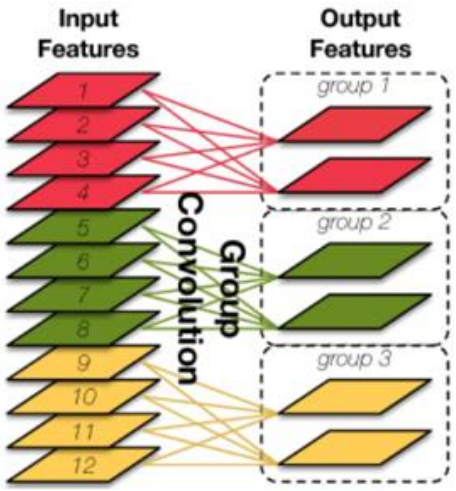
B. Light Frameworks :

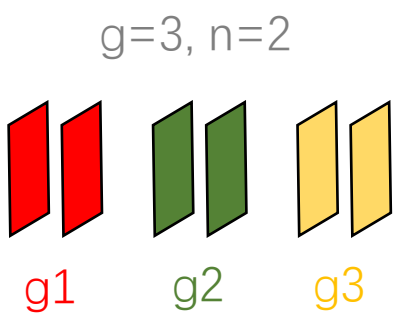
B3.1 ShuffleNet – V1

Ideas: 1x1 Group Conv

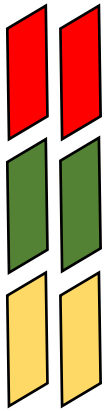
Channel Shuffle



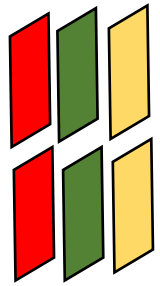




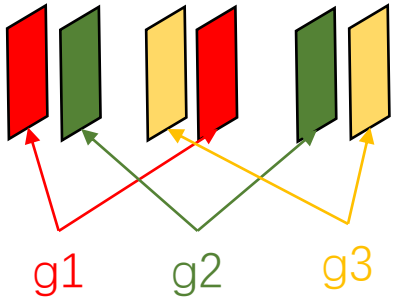
s1: channel=gxn



s2: reshape to [g,n]



s3: transpose to [n,g]



s4: flatten [n,g]

I. Net Framework

B. Light Frameworks :

B3.1 ShuffleNet – V1

Structure: Comparison:

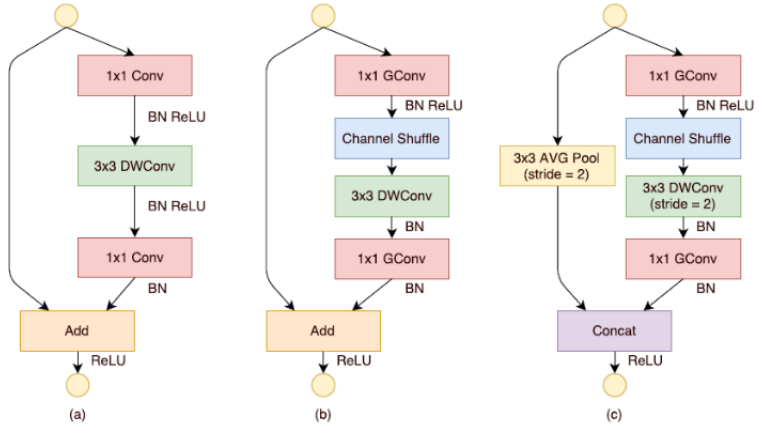


Figure 2: ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

With Resnet (Similarity)

A. Use skip connection
   1x1 – 3x3 – 1x1 module

With Resnet (Difference)

A. Use DConv

With MobileNet - V2 (Similarity)

A. Use DConv
   1x1 – 3x3 – 1x1 module
   2 bottlenecks

With MobileNet - V2 (Difference)

A. 1x1 Gconv
   It has ReLU as activation of a bottleneck
   It has FC layer

I. Net Framework

B. Light Frameworks :

B3.1 ShuffleNet – V1

Structure: Shuffle Unit

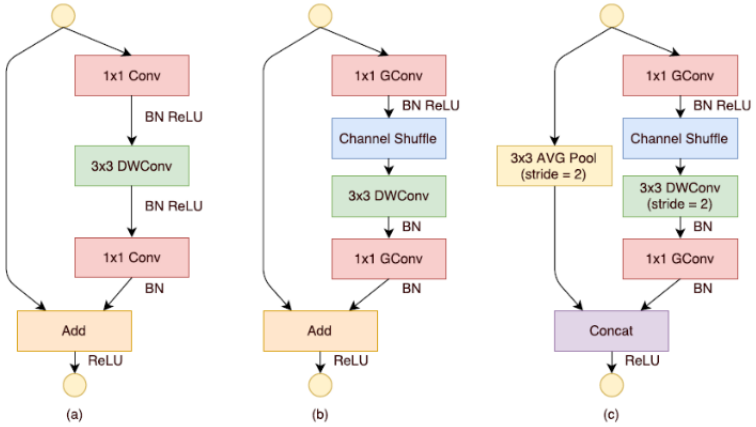


Figure 2: ShuffleNet Units. a) bottleneck unit [9] with depthwise convolution (DWConv) [3, 12]; b) ShuffleNet unit with pointwise group convolution (GConv) and channel shuffle; c) ShuffleNet unit with stride = 2.

ShuffleNet framework

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	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					143M	140M	137M	133M	137M

Table 1. ShuffleNet architecture. The complexity is evaluated with FLOPs, i.e. the number of floating-point multiplication-adds. Note that for Stage 2, we do not apply group convolution on the first pointwise layer because the number of input channels is relatively small.

I. Net Framework

- B. Light Frameworks :

- B3.1 ShuffleNet – V1
- Shuffle code:

```
def channel_shuffle(x, groups):
    """
    Parameters
        x: Input tensor of with `channels_last` data format
        groups: int number of groups per channel
    Returns
        channel shuffled output tensor
    Examples
        Example for a 1D Array with 3 groups
        >>> d = np.array([0,1,2,3,4,5,6,7,8])
        >>> x = np.reshape(d, (3,3))
        >>> x = np.transpose(x, [1,0])
        >>> x = np.reshape(x, (9,))
        '[0 1 2 3 4 5 6 7 8] --> [0 3 6 1 4 7 2 5 8]'
    """
    height, width, in_channels = x.shape.as_list()[1:]
    channels_per_group = in_channels // groups
    x = K.reshape(x, [-1, height, width, groups, channels_per_grou
    x = K.permute_dimensions(x, (0, 1, 2, 4, 3))  # transpose
    x = K.reshape(x, [-1, height, width, in_channels])
    return x
```

I. Net Framework

B. Light Frameworks :

B3.2 ShuffleNet – V2

Ideas: Time=T(FLOPS)+T(I/O)

4 Guidelines:

G1. MAC becomes minimal when input/output has same the size.

G2. Excessive GConv increases MAC.

G3. Network fragmentation reduces degree of parallelism

G4. Eltwise operations are non-negligible

Explain: (assume 1x1 kernel)

Feature map size (In):	$w \times h \times c_1$
Feature map size (Out):	$w \times h \times c_2$
Conv FLOPs (B):	whc_1c_2
(In) memory:	whc_1
(Out) memory:	whc_2
Kernels (In+Out) memory:	c_1c_2

MAC: Memory Access Cost

$$MAC = hw(c_1 + c_2) + c_1c_2$$
$$= \sqrt{(hw(c_1 + c_2))^2} + \frac{B}{hw}$$
$$= \sqrt{(hw)^2 \cdot (c_1 + c_2)^2} + \frac{B}{hw}$$
$$\geq \sqrt{(hw)^2 \cdot 4c_1c_2} + \frac{B}{hw}$$
$$= 2\sqrt{hw \cdot (hwc_1c_2)} + \frac{B}{hw}$$
$$= 2\sqrt{hwB} + \frac{B}{hw}$$

$\boldsymbol{MAC_{min}\ can\ be\ gotten\ when\ c_1 = c_2}$

I. Net Framework

B. Light Frameworks :

B3.2 ShuffleNet – V2

Ideas: Time=T(FLOPS)+T(I/O)

4 Guidelines:

G1. MAC becomes minimal when input/output has same the size.

G2. Excessive GConv increases MAC.

G3. Network fragmentation reduces degree of parallelism

G4. Eltwise operations are non-negligible

Explain:

Feature map size (In):	$w \times h \times c_1$
Feature map size (Out):	$w \times h \times c_2$
Conv FLOPs (B):	$wh\frac{c_1}{g}\frac{c_2}{g}g$
(In) memory:	whc_1
(Out) memory:	whc_2
Kernels (In+Out) memory:	$\frac{c_1}{g}\frac{c_2}{g}g$
Group number:	g

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

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

$$= Bg(\frac{1}{c_1} + \frac{1}{c_2}) + \frac{B}{hw}$$

MAC is getting bigger with the increase of group number.

So g shouldn't be too big.

I. Net Framework

B. Light Frameworks :

B3.2 ShuffleNet – V2

Ideas: Time=T(FLOPS)+T(I/O)

4 Guidelines:

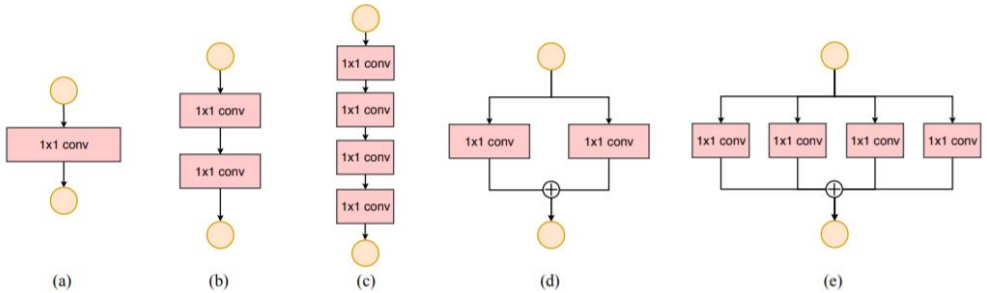
G1. MAC becomes minimal when input/output has same the size.

G2. Excessive GConv increases MAC.

G3. Network fragmentation reduces degree of parallelism

G4. Eltwise operations are non-negligible

Explain:



	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

$Speed$: $a > b > d > c > e$

The more branches, the slower the system.

The more fragments, the slower the system.

I. Net Framework

B. Light Frameworks :

B3.2 ShuffleNet – V2

Ideas: Time=T(FLOPS)+T(I/O)

4 Guidelines:

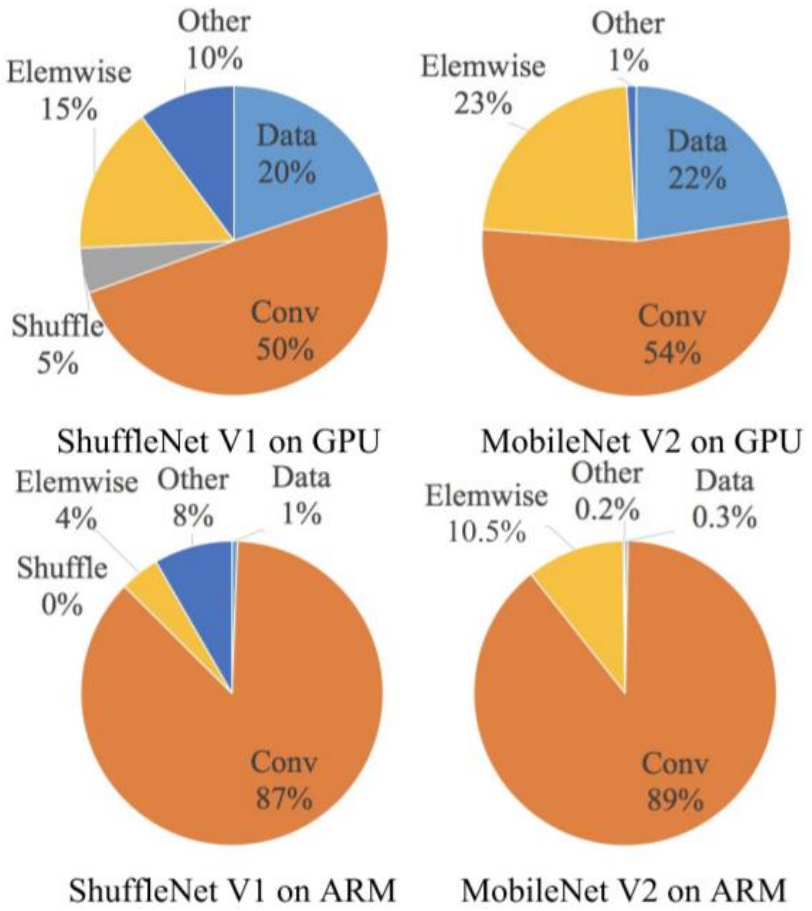
G1. MAC becomes minimal when input/output has same the size.

G2. Excessive GConv increases MAC.

G3. Network fragmentation reduces degree of parallelism

G4. Eltwise operations are non-negligible

Explain:



Reduce the usage of Eltwise

I. Net Framework

B. Light Frameworks :

B3.2 ShuffleNet – V2

Ideas: Time=T(FLOPS)+T(I/O)

4 Guidelines:

G1. MAC becomes minimal when input/output has same the size.

G2. Excessive GConv increases MAC.

G3. Network fragmentation reduces degree of parallelism

G4. Eltwise operations are non-negligible

E.G.
Q. How other networks break the rules?

ShuffleNet – V1:
X G2: rely on GConv heavily
X G1: it's bottleneck-like building blocks

MobileNet – V2:
X G1: inverted bottleneck structure
X G4: Eltwise+ at thick feature

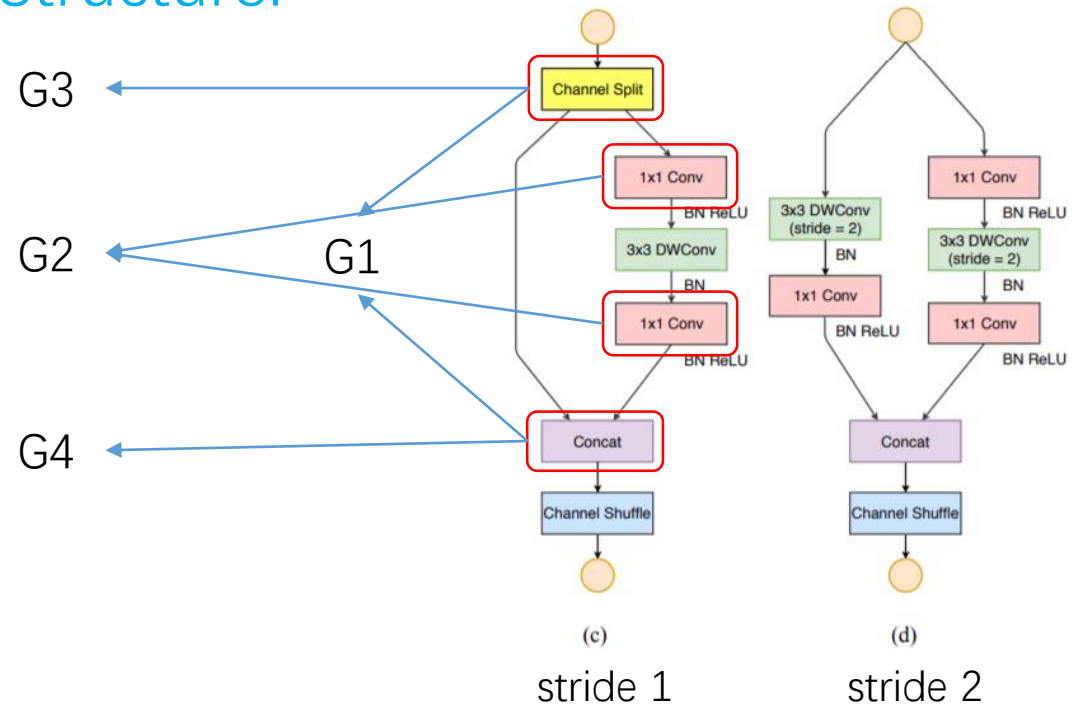
Auto-generated structures:
X G3: highly fragmented

I. Net Framework

B. Light Frameworks :

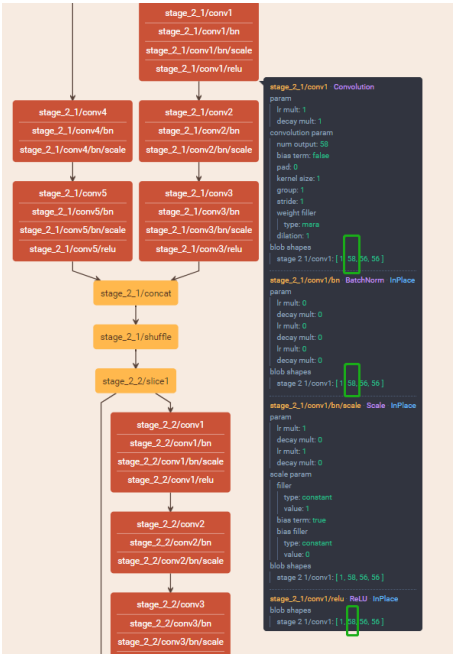
B3.2 ShuffleNet – V2

Structure:



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.



I. Net Framework

B. Light Frameworks :

B3.2 ShuffleNet – V2

Comparison:

It skip connection is very much like DenseNet.

The difference is it just map half the feature.

ShuffleNet – V2 has better result.

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

I. Net Framework

B. Light Frameworks :

B4. Summary

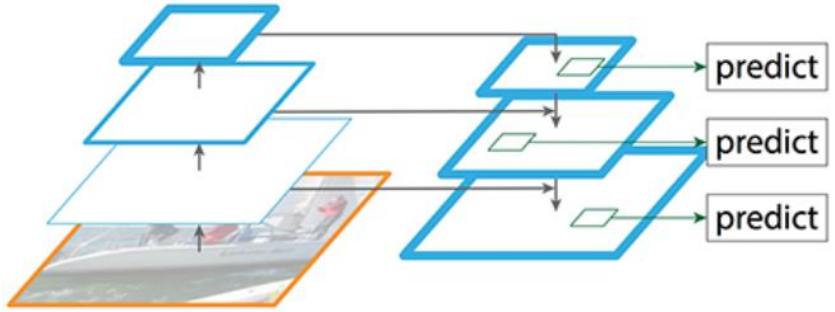
ShuffleNet – V2 > MobileNet – V2 > ShuffleNet > MobileNet > SqueezeNet

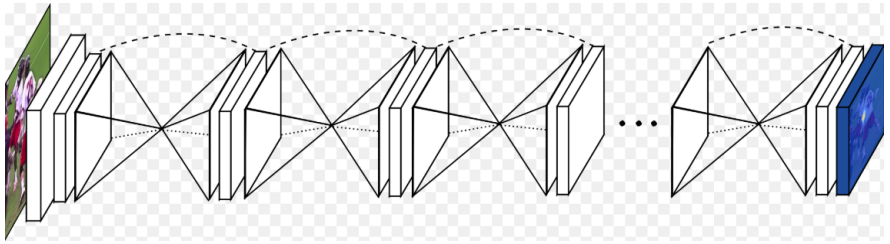
---

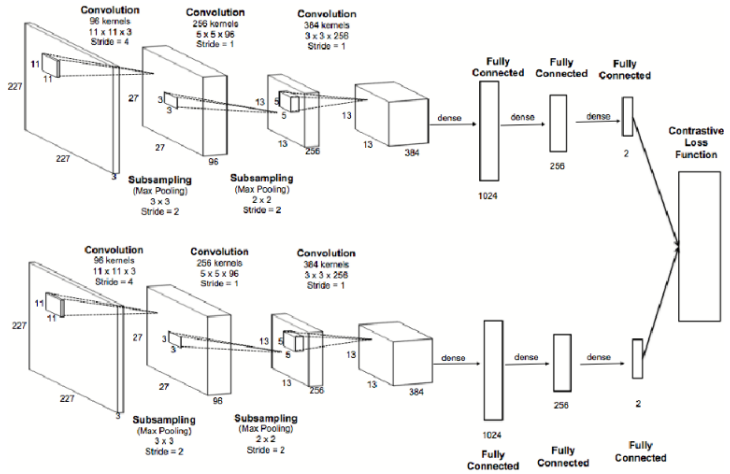
1st Choice: ShuffleNet/MobileNet-V2
2nd Choice: MobileNet

Please remember the improvement among each family;
Please remember the differences between each family.

If you are interested in this topic, here are some other papers worth reading:
Xception / ResNeXt / MobileID …







I. Net Framework

Summary:

Development:	deeper
	skip-connection
	module
	light but accurate
Trend:	multi-scale (pyramid)
	auto
Others:	hourglass
	Siamese

II. FLOPs

II. FLOPs

C. **Concept** & Computation

Meaning:

FLOPs: floating-point operations

浮点运算次数

FLOPS: floating –point operations per second

每秒浮点数运算次数 / 每秒峰值速度

II. FLOPs

C. **Concept** & Computation

Meaning:

FLOPs: floating-point operations

浮点运算次数

FLOPS: floating-point operations per second

每秒浮点数运算次数 / 每秒峰值速度

II. FLOPs

C. Concept & **Computation**

Meaning: floating-point operations

E.g.
Input Tensor: $H \times W \times C_{in}$
Output Tensor: $H \times W \times C_{out}$
Kernel Size: $k_w \times k_h$
# of parameters: $\underbrace{(k_w \times k_h \times C_{in}) \times C_{out}}_{\boldsymbol{W}} \underbrace{+}_{\boldsymbol{b}} \equiv N_p$

II. FLOPs

C. Concept & **Computation**

Meaning: floating-point operations

E.g.

Input Tensor: $H \times W \times C_{in}$

Output Tensor: $H \times W \times C_{out}$

Kernel Size: $k_w \times k_h$

# of parameters: $\underbrace{(k_w \times k_h \times C_{in}) \times C_{out}}_{\boldsymbol{W}} + \underbrace{C_{out}}_{\boldsymbol{b}} \equiv N_p$

FLOPs: $[(k_w \times k_h \times C_{in}) \times C_{out} + C_{out}] \times H \times W = N_p \times H \times W$

II. FLOPs

C. Concept & **Computation**

Meaning: floating-point operations

E.g.
Input Tensor: $H \times W \times C_{in}$
Output Tensor: $H \times W \times C_{out}$
Kernel Size: $k_w \times k_h$

# of parameters: $\underbrace{(k_w \times k_h \times C_{in}) \times C_{out}}_{\boldsymbol{W}} + \underbrace{C_{out}}_{\boldsymbol{b}} \equiv N_p$ 【For Conv】

FLOPs: $[(k_w \times k_h \times C_{in}) \times C_{out} + C_{out}] \times H \times W = N_p \times H \times W$

# of parameters: $\underbrace{N_{in} \times N_{out}}_{\boldsymbol{W}} + \underbrace{N_{out}}_{\boldsymbol{b}}$ 【For FC】

FLOPs: = # of parameters

II. FLOPs

C. Concept & **Computation**

Meaning: floating-point operations

E.g.

Input Tensor: $H \times W \times C_{in}$

Output Tensor: $H \times W \times C_{out}$

Kernel Size: $k_w \times k_h$

# of parameters: $\underbrace{(k_w \times k_h \times C_{in}) \times C_{out}}_{\boldsymbol{W}} + \underbrace{C_{out}}_{\boldsymbol{b}} \equiv N_p$

FLOPs: $[(k_w \times k_h \times C_{in}) \times C_{out} + C_{out}] \times H \times W = N_p \times H \times W$

$= (2 \times \text{FLOPs})$

[1 Mac = 2 FLOPs = 1 “+” + 1 “*”]

II. FLOPs

C. Concept & **Computation**

E.G: Real cases

Case 1: VGG16

Total: $\sum_{c=1}^{5} stage_n + \sum_{f=1}^{3} fc_f$
$= 15.5$ GFLOPs
$= 1.547 \times 10^{10}$ FLOPs

Stage 1: 3x3x3x64x224x224 +
3x3x64x64x224x24 = 1.93×10^9

Stage 2: 3x3x64x128x112x112 +
3x3x128x128x112x112 = 2.78×10^9

Stage 3: 3x3x128x256x56x56 +
3x3x256x256x56x56x2 = 4.63×10^9

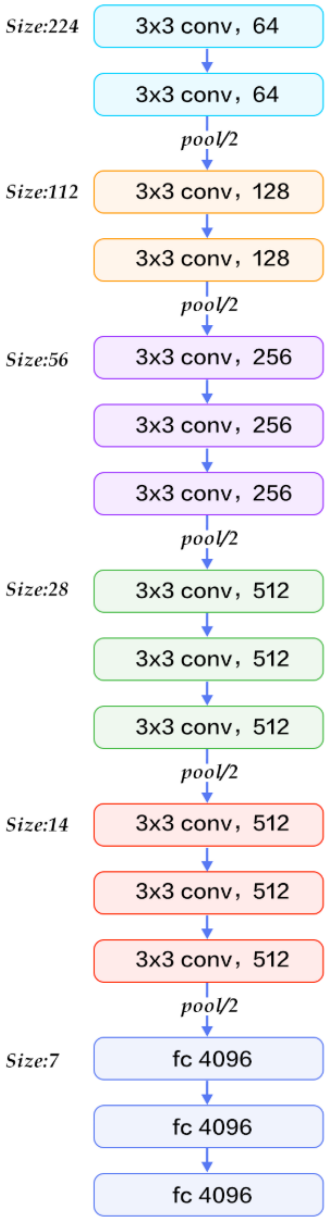
Stage 4: 3x3x256x512x28x28 +
3x3x256x256x28x28x2 = 4.63×10^9

Stage 5: 3x3x512x512x14x14x3= 1.39×10^9

fc 1: 7x7x512x4096= 1.03×10^8

fc 2: 4096x4096= 1.68×10^7

fc 3: 4096x1000= 4.10×10^6



II. FLOPs

C. Concept & **Computation**

E.G: Real cases

Case 2: Comparisons

Resnet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

MobileNet: 0.573 GFLOPs

ShuffleNet: 0.136GFLOPs