Deep Learning Optimization - Representative CNNs

Feb 27, March 6, 2023

Eunhyeok Park

Class Overview

- Goal: Understanding the evolution history of convolutional neural networks

- The structure and the key idea of CNN designs are reused for various applications
  - AlexNet, VGG, GoogLeNet, ResNet, SqueezeNet, MobileNet-v1~v3

Large Training Data: ImageNet

ImageNet dataset

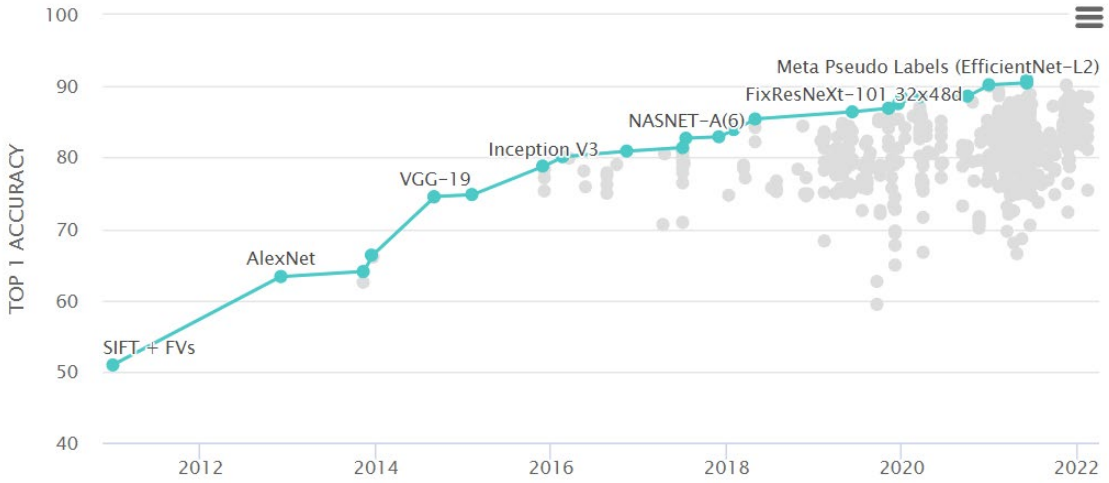
- ~15 million labeled high-resolution images, overall 22k categories
- Collected from the web, annotated by human using Amazon's Mechanical Turk

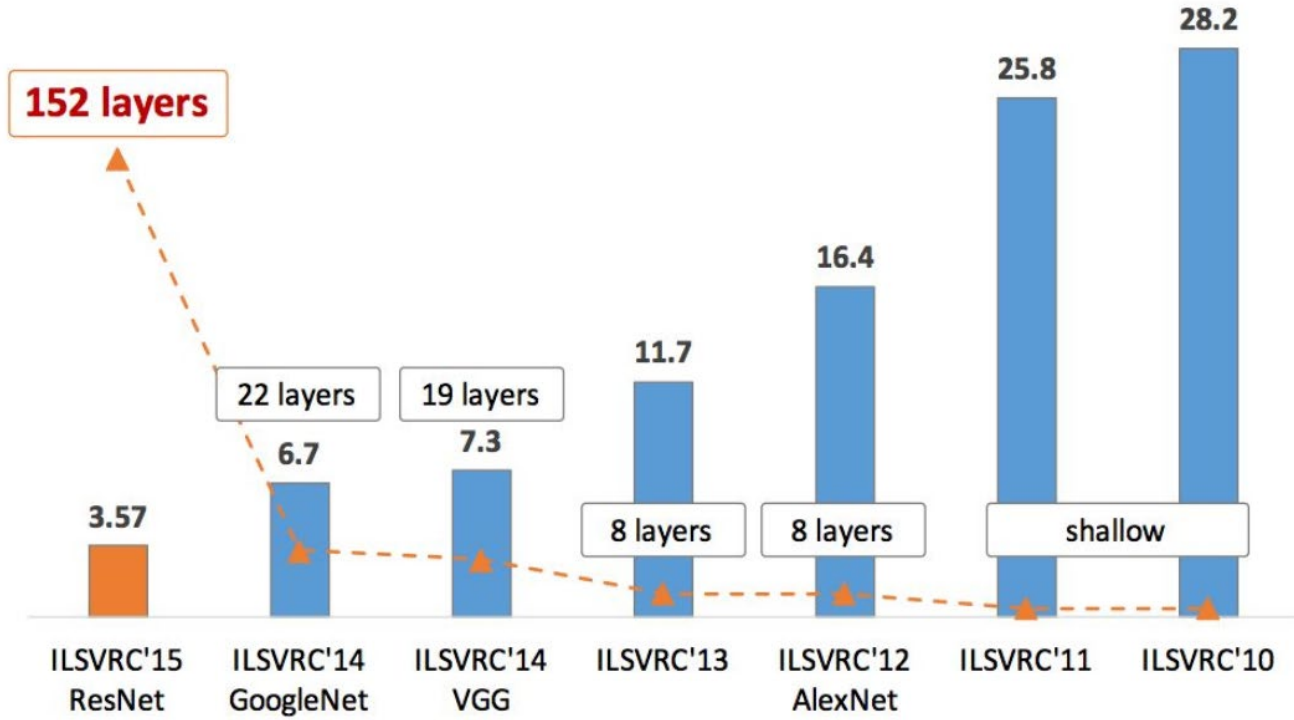
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- 1.2 million training images, 50k validation images, 1k categories
- Top-1, Top-5 accuracy for 150k test images (unknown label)

Evolution of CNN networks

- Accuracy of CNNs is continuously increased with advanced ideas, e.g., residual connection, autoML, transfer learning, …

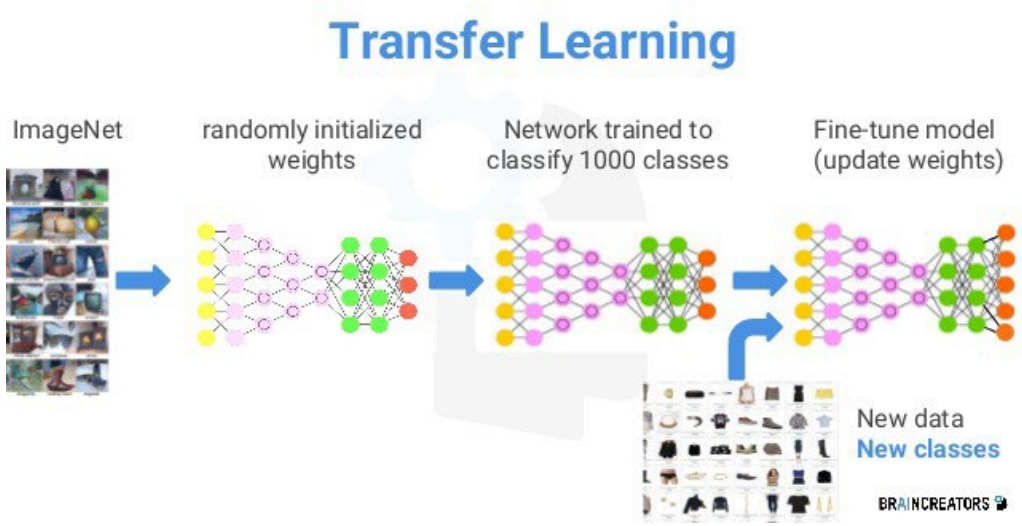


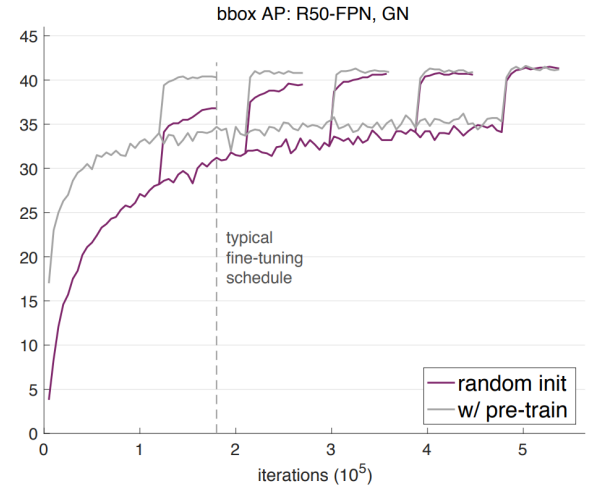


[https://paperswithcode.com/sota/image-classification-on-imagenet]

Why Is It Important? - 1

- This is a difficult problem and is used as a benchmark to improve network performance
- Datasets have enough data to learn the Large network to enable a variety of applications
  - Transfer learning
    - Well-trained network for ImageNet has abundant expressive power and it can be used for other tasks via transfer learning
    - It is useful to enable training with smaller dataset and minimize computation cost



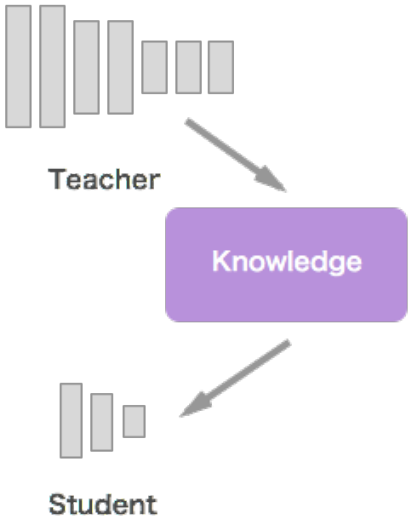


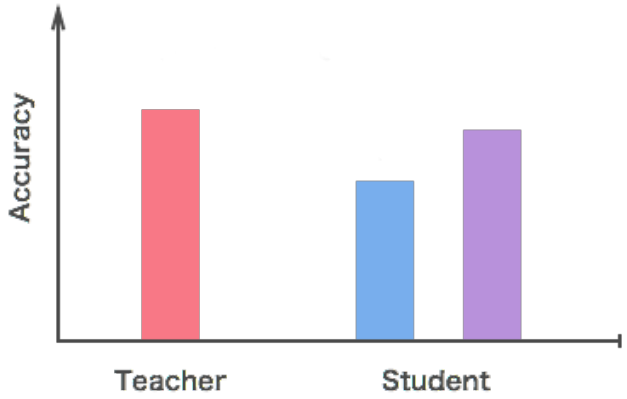
[https://mc.ai/what-you-must-know-about-transfer-learning-2/]
[Rethinking ImageNet Pre-training, He, 2018]

Why Is It Important? - 2

- This is a difficult problem and is used as a benchmark to improve network performance
- Datasets have enough data to learn the Large network to enable a variety of applications
  - Knowledge distillation
    - The enriched feature of a large network can be used to boost the accuracy of smaller network

$$\mathcal{L}_{KD}(\mathbf{W_S}) = \mathcal{H}(\mathbf{y_{true}}, \mathrm{P_S}) + \lambda \mathcal{H}(\mathrm{P_T^\tau}, \mathrm{P_S^\tau})$$



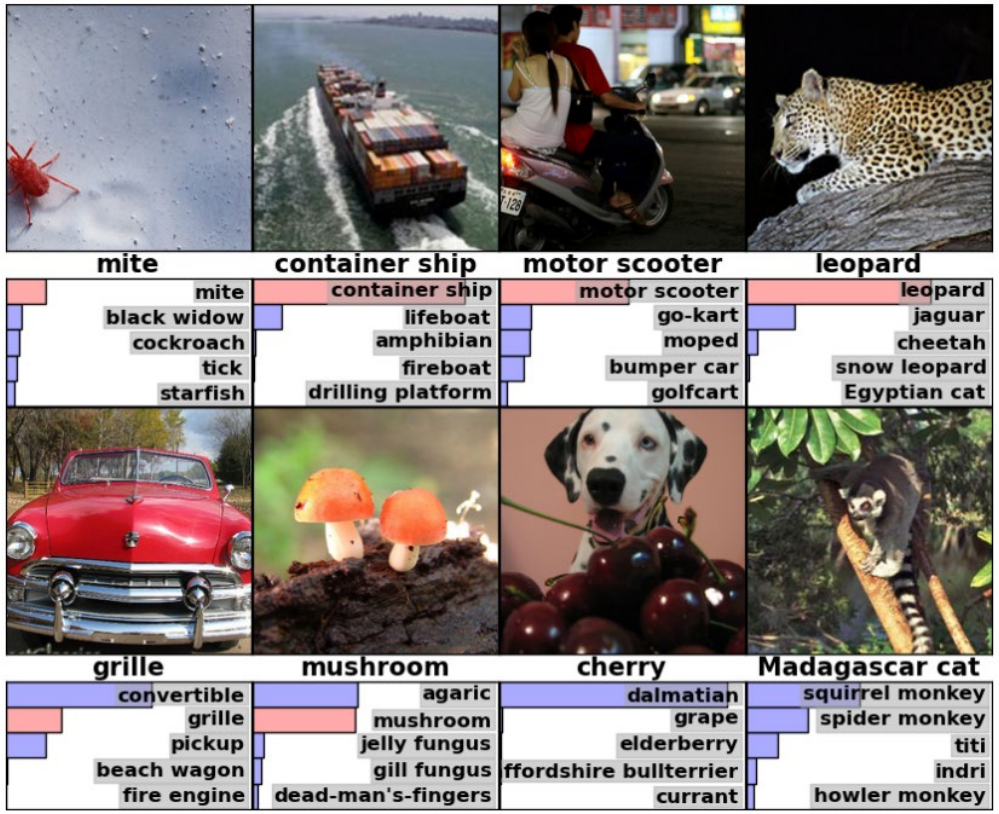


[https://medium.com/neuralmachine/knowledge-distillation-dc241d7c2322]

AlexNet & VGG

AlexNet – winner of ILSVRC'12

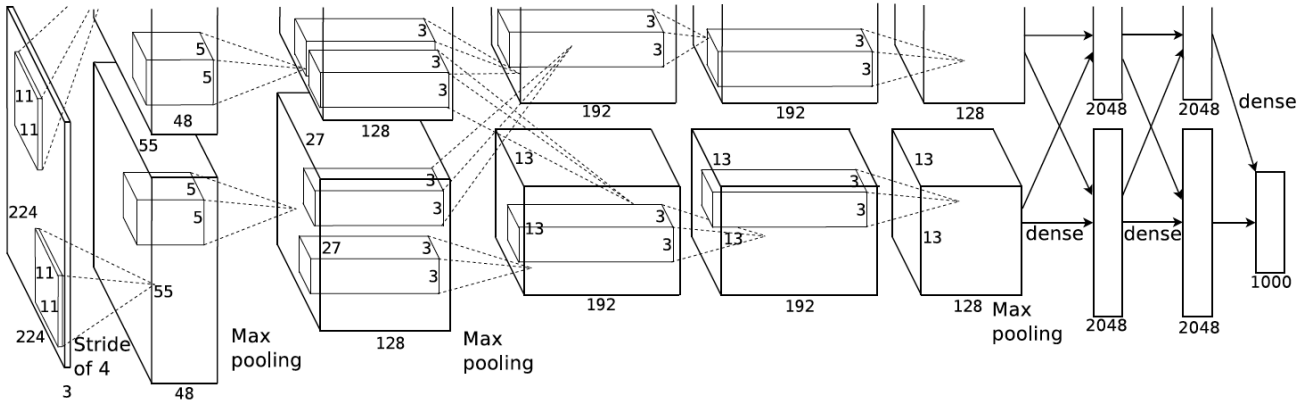
- Adopt large-scale CNN 650k neurons, 60 million parameters
- 5 convolution layer + 3 fc layers. Top-5 error: 15.3 % (2nd best: 26.2 %)



Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

[ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky]

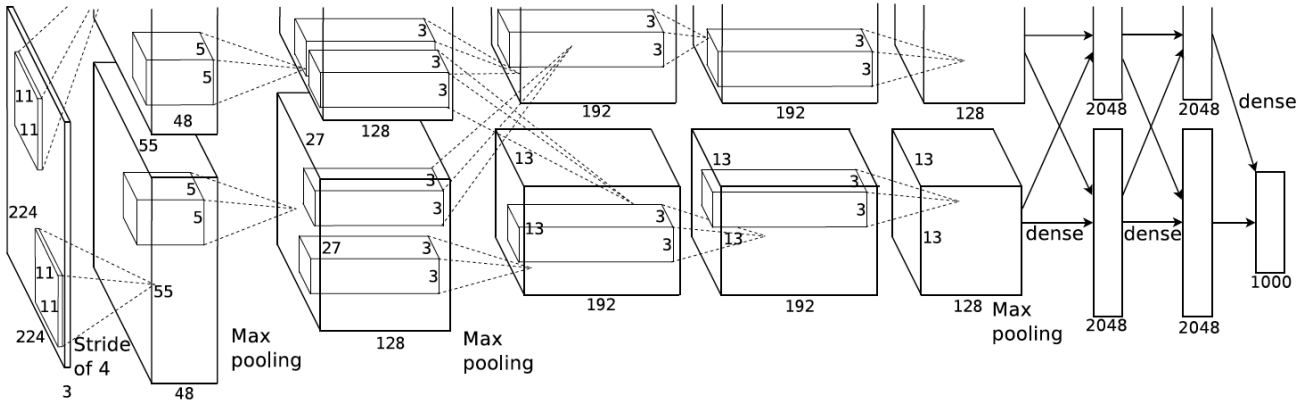
Architecture Details



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

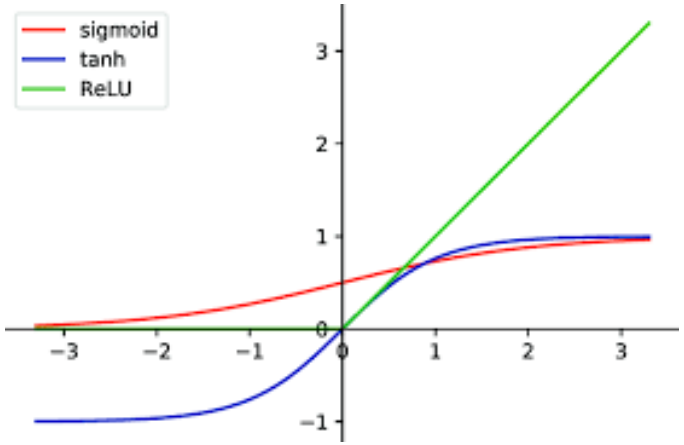
[ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky]
[cs231n]

Architecture Details



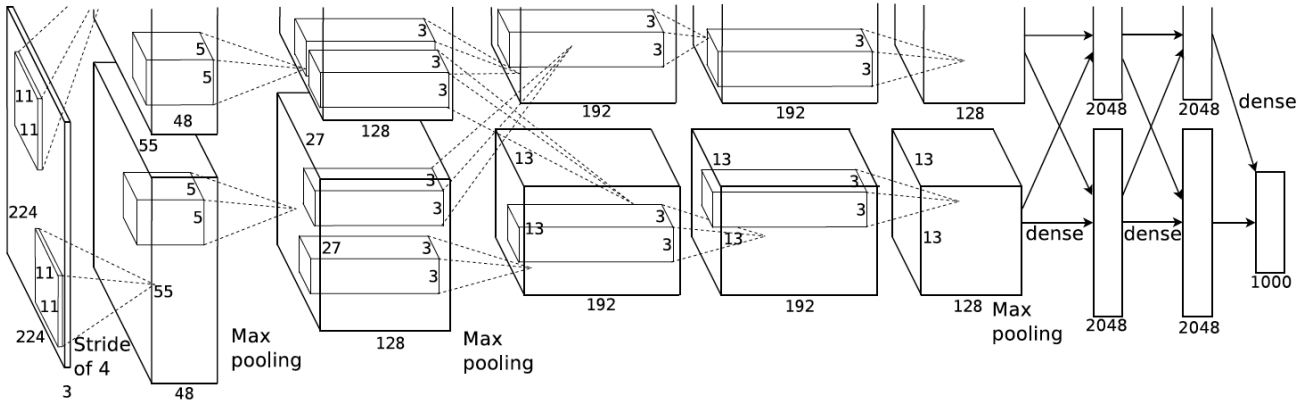
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU



[ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky]
[cs231n]

Architecture Details



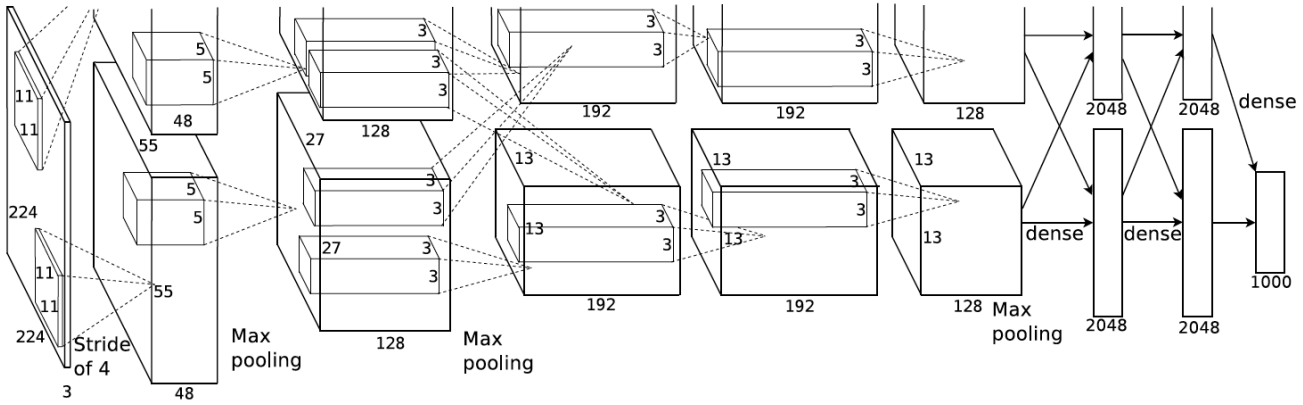
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

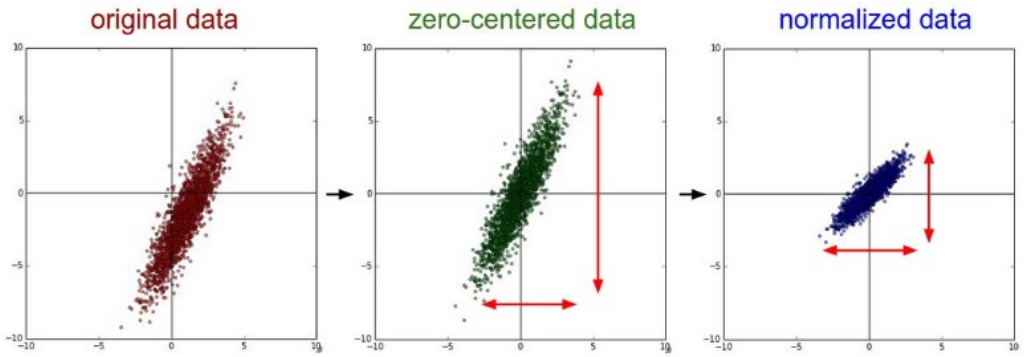
[ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky
[cs231n]

Architecture Details



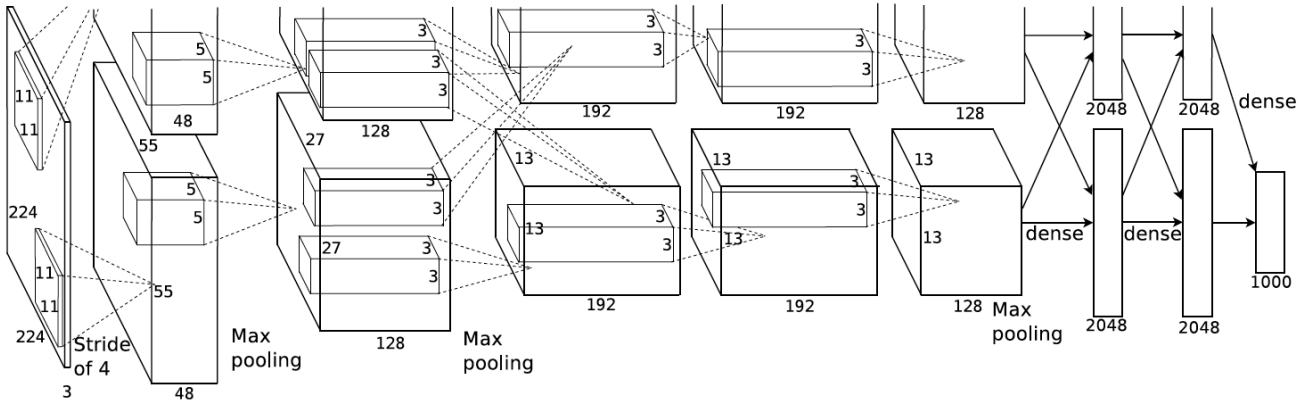
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation



[ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky]
[cs231n]

Architecture Details



Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

[ImageNet Classification with Deep Convolutional Neural Networks, Krizhevsky] [cs231n]

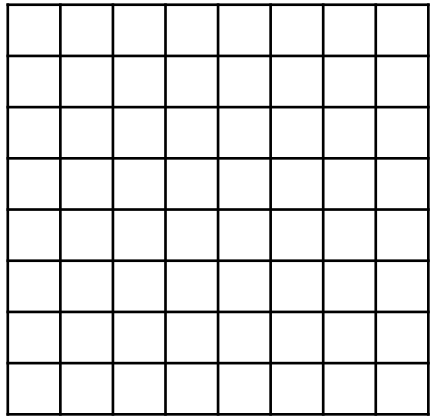
of Parameters & Operations

- $input: C_i \times H_i \times W_i$
- $kernel: K_h \times K_w$
- $output: C_o \times H_o \times W_o$
- $groups: G$
- $stride: S_h \times S_w$
- $padding: P_h \times P_w$
- $dilation: D_h \times D_w$

- $H_o = \frac{H_i + 2P_h - D_h(K_h - 1) - 1}{S_h}$

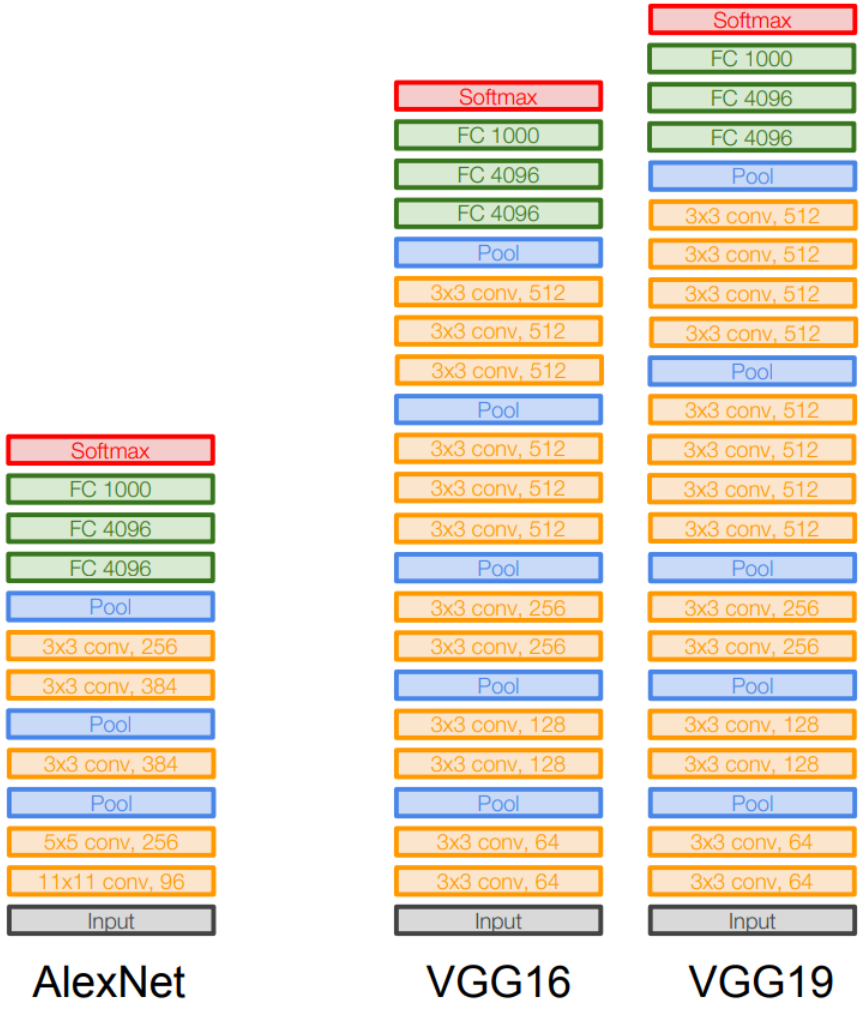
- # of parameters
  - $= \frac{C_o C_i K_h K_w}{G} + C_o$

- # of operations
  - $= \frac{H_o W_o C_o C_i K_h K_w}{G} + H_o W_o C_o$



VGG: Homogeneous 3x3 Conv Layers

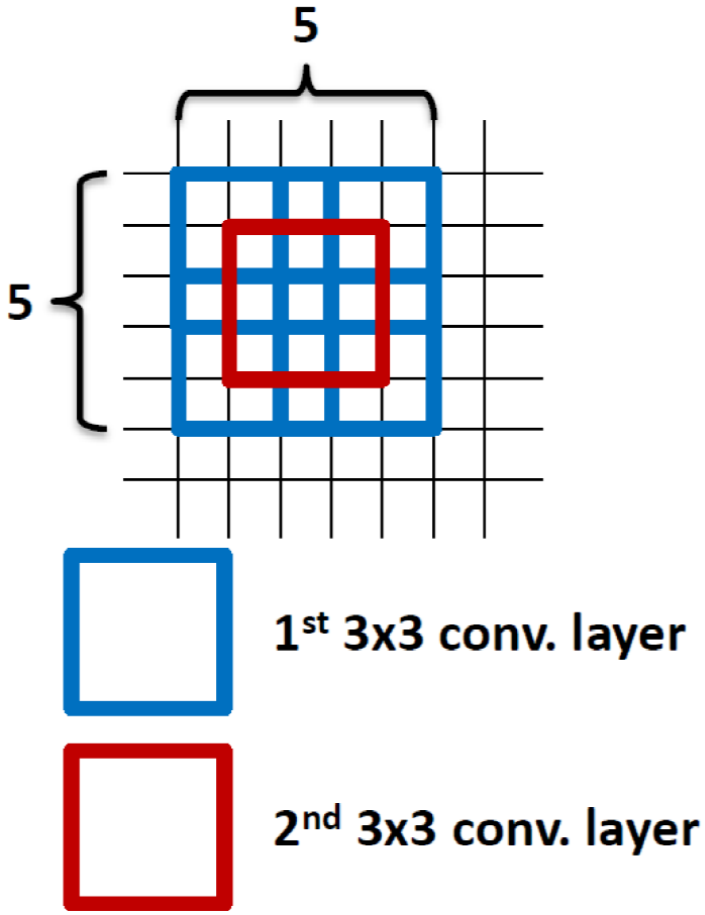
- Key design choices
  - Stacking multiple 3x3 kernels
  - Only 3x3 conv with stride 1, pad 1 + 2x2 max pooling with stride 2
  - 7.3 % Top-5 error @ ILSVRC'14



[cs231n]

Receptive Fields & 3x3 Layers

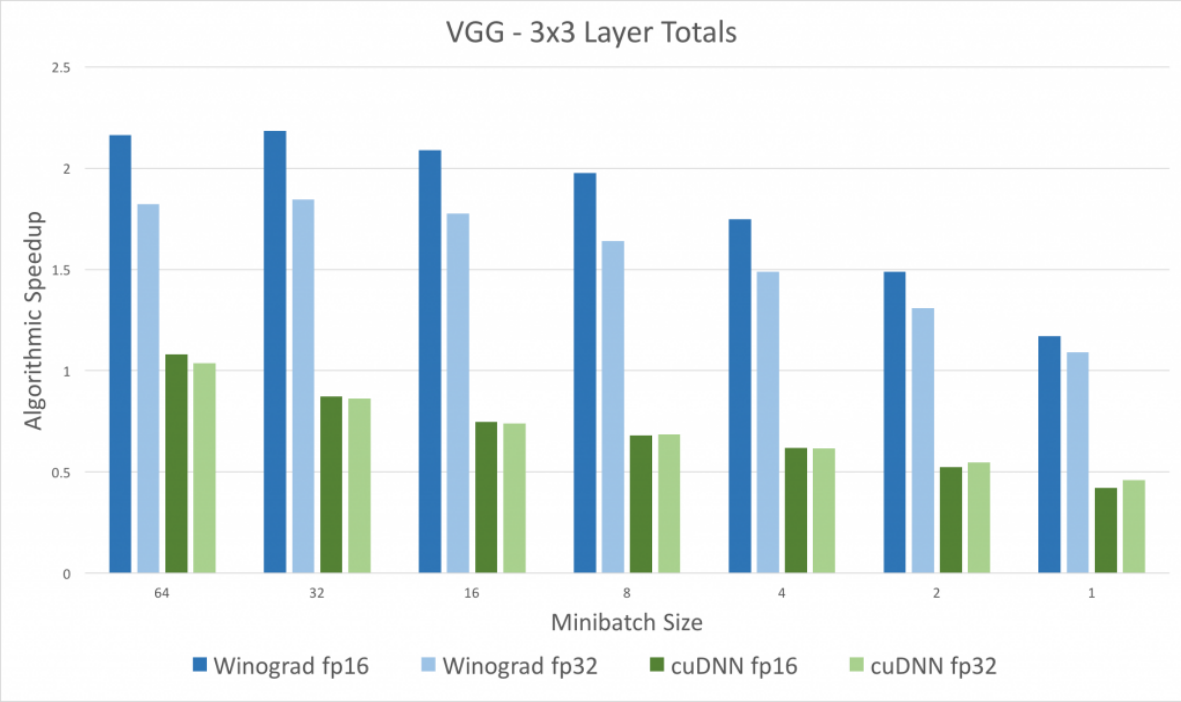
- Why 3x3 layers?
  - Stacked conv layers have a large receptive field
    - 2 x (3x3 conv) = 5x5 receptive field
    - 3 x (3x3 conv) = 7x7 receptive field
  - More non-linearity
  - Less parameters



[Simonyan]

Pros & Cons of VGG structure

- Hardware-friendly structure
  - Simple feature extractor – 3x3 convolution + 2x2 max-pooling
  - Appropriate for optimization techniques, e.g. Winograd convolution
  - Large memory reuse – high cache efficiency



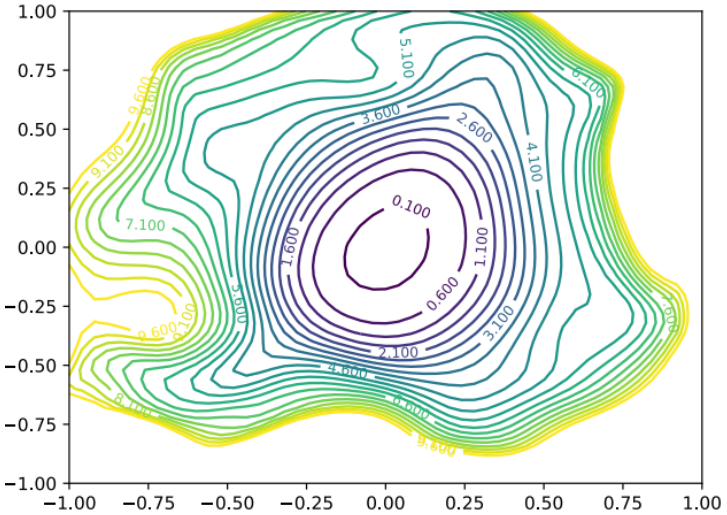
[Visualizing the Loss Landscape of Neural Nets, Li]

Pros & Cons of VGG structure

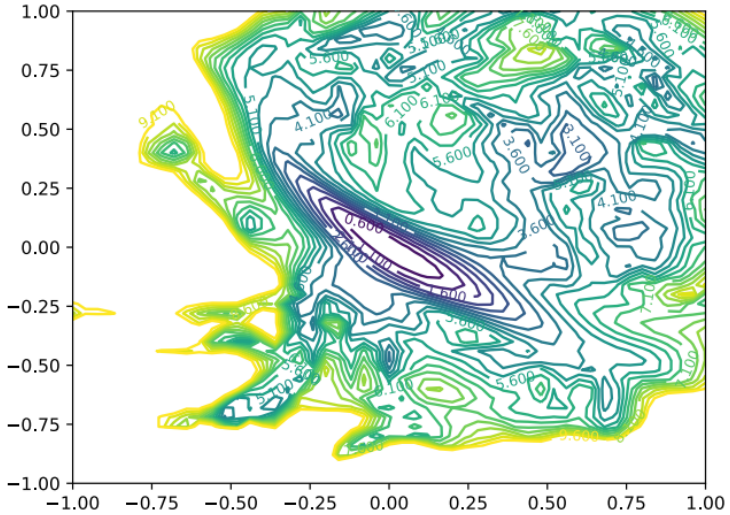
- Hard to train
  - Require proper initialization method
  - Inappropriate hyper-parameters, e.g., learning rate, cause divergent of the network

Contour Plots & Random Directions To use this approach, one chooses a center point θ^* in the graph, and chooses two direction vectors, δ and η. One then plots a function of the form $f(\alpha) = L(\theta^* + \alpha\delta)$ in the 1D (line) case, or

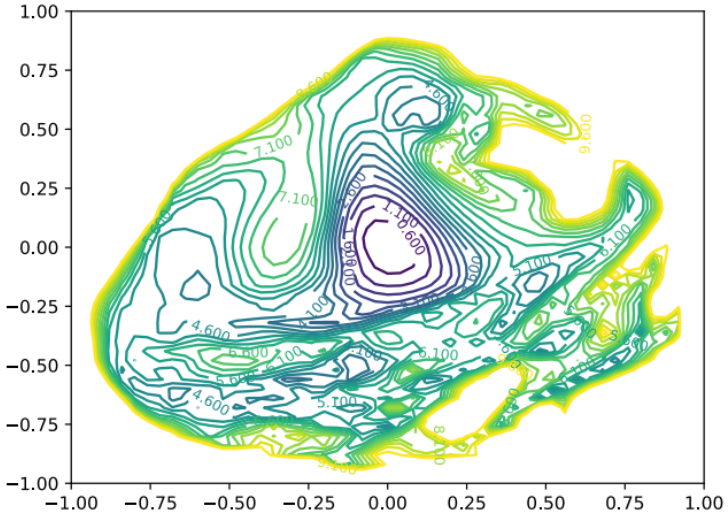
$$f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\eta) \quad (1)$$



20 layers



56 layers

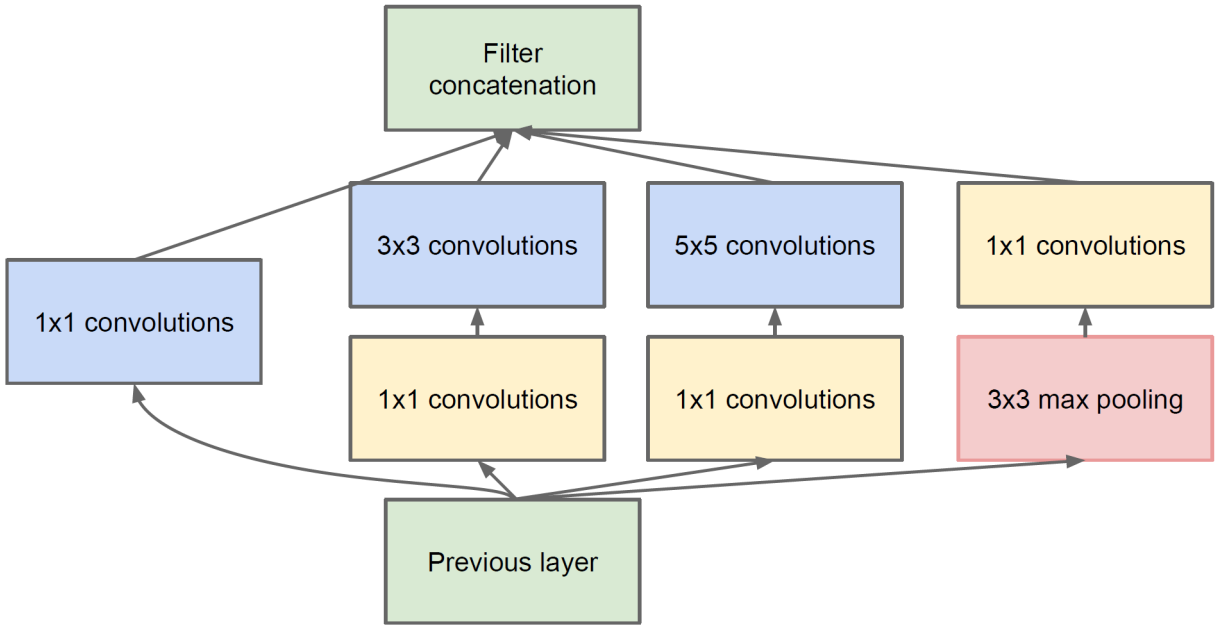


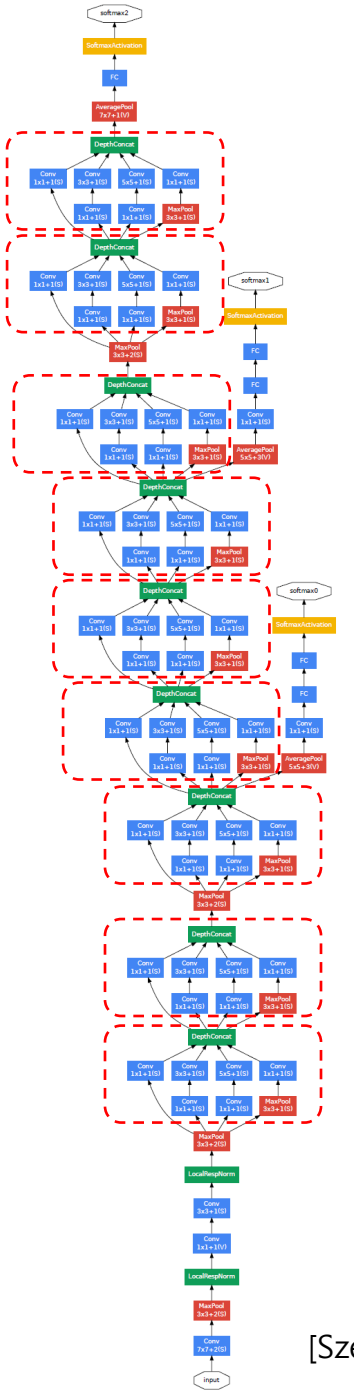
110 layers

GoogLeNet (Inception-vx)

GoogLeNet

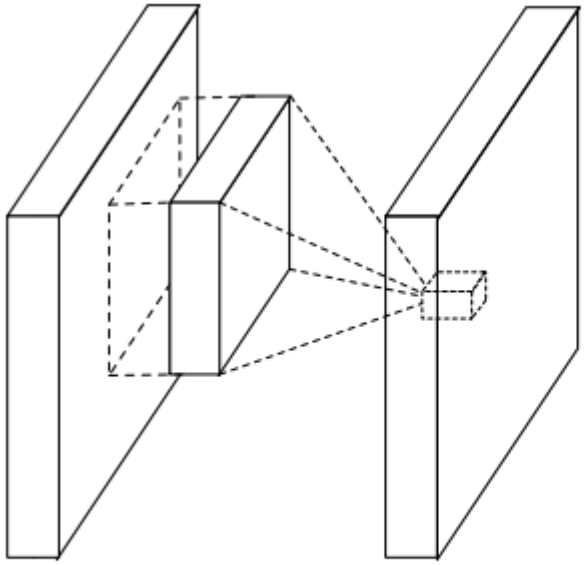
- Repetition of the same structure
  - Inception module
- Mixture of multi-scale convolution kernels





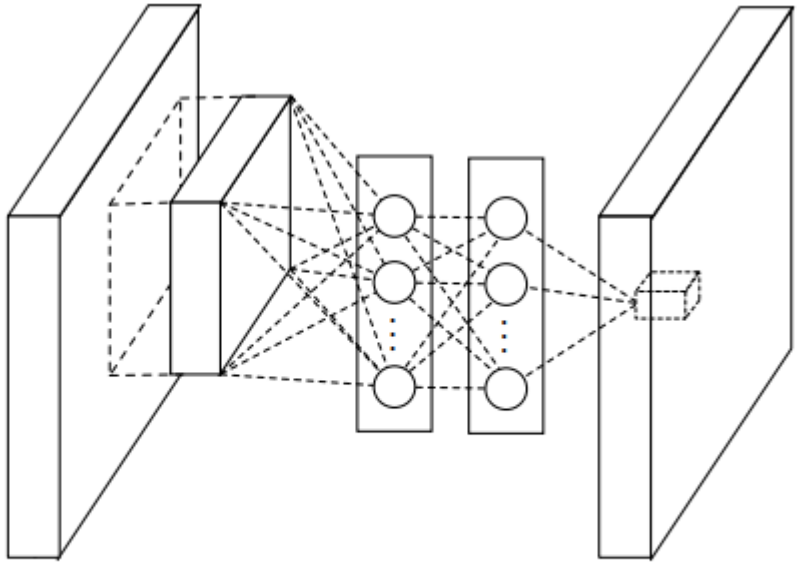
[Szegedy 2014]

Initial Thought: Network-in-Network



(a) Linear convolution layer

Linear feature extractor (dot-product)

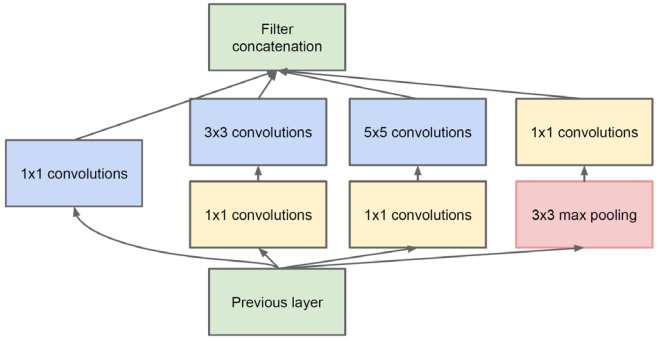


(b) Mlpconv layer

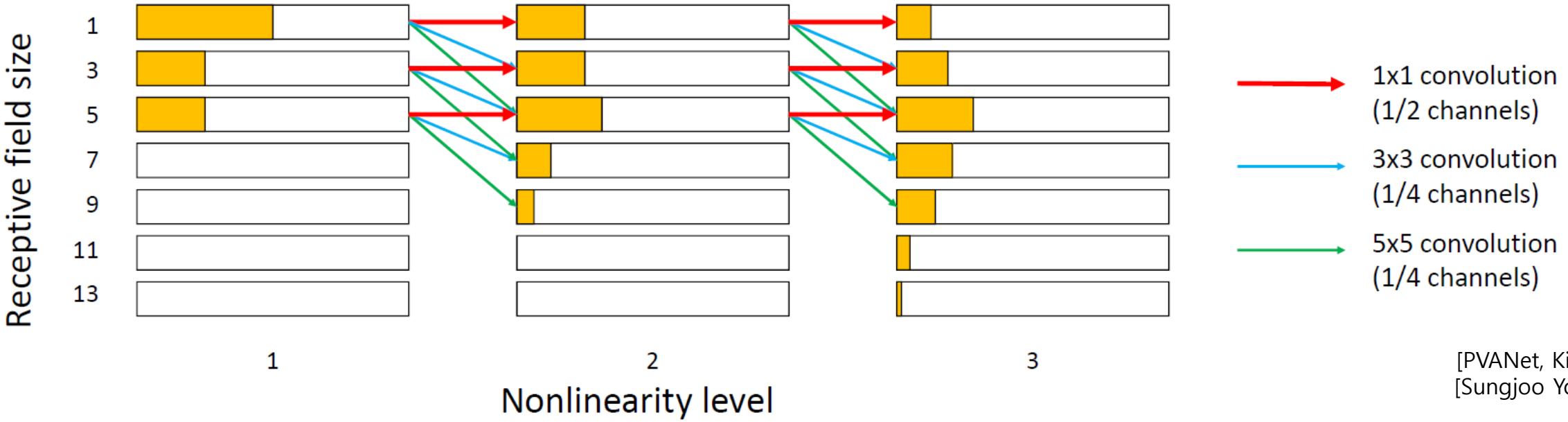
Complex multi-layer feature extractor

[Network in Network, Lin]

Receptive Field Size



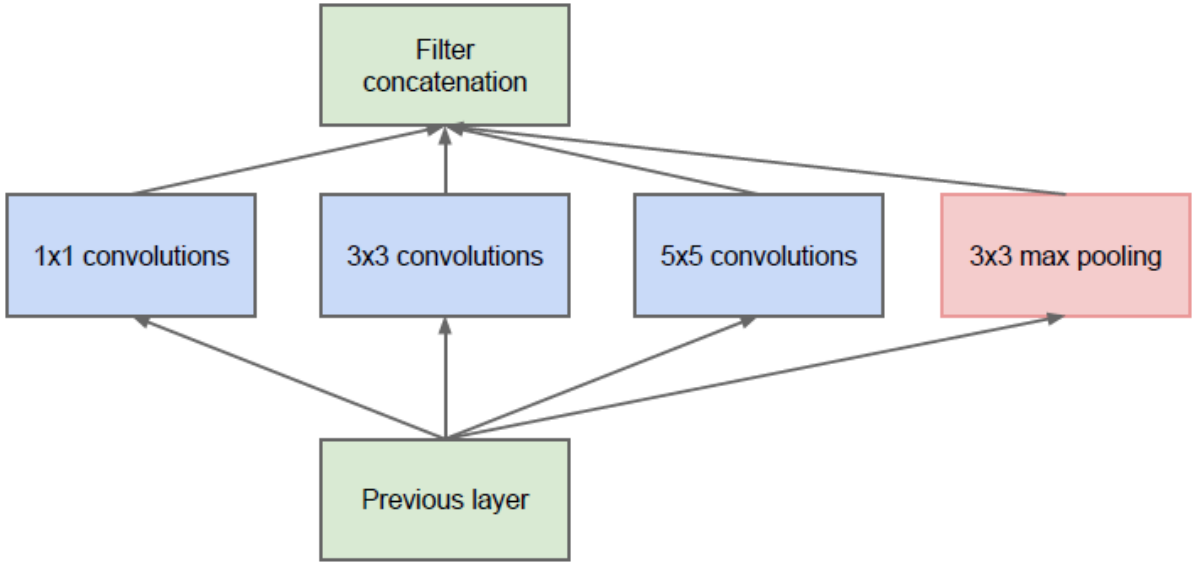
- An Inception module produces output activations of different sizes of receptive fields, so that increases the variety of receptive field sizes in the previous layer
- Very useful in recognizing both large and small objects

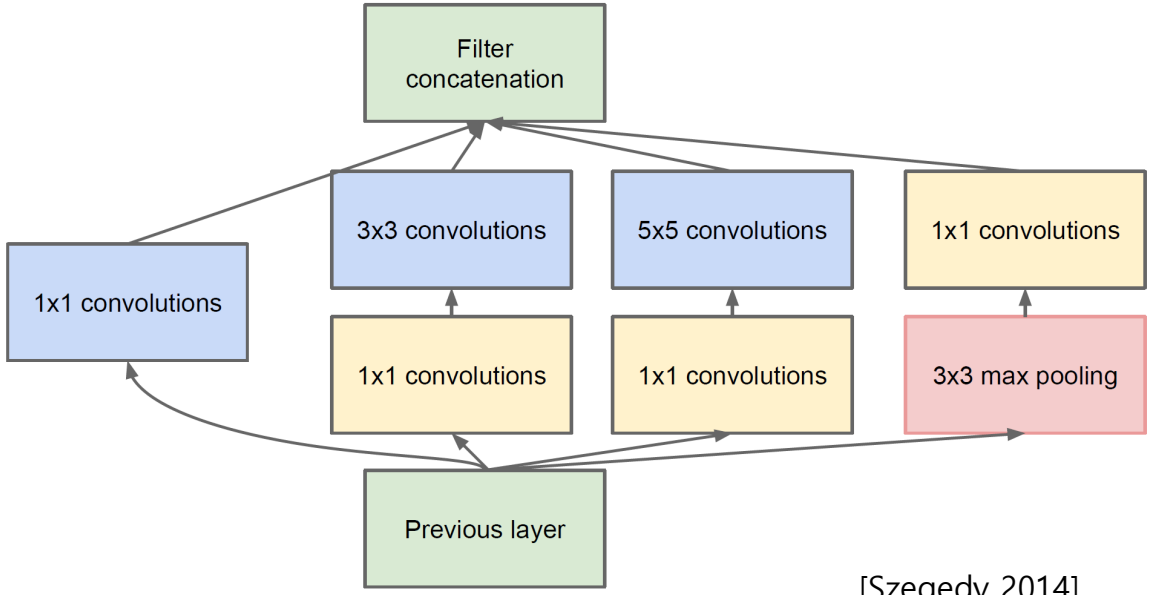


[PVANet, Kim]
[Sungjoo Yoo]

Inception-v1 Module

- Mixture of multi-scale convolution kernels: 1x1, 3x3, and 5x5
- 1x1 convolution before 3x3 and 5x5
  - Dimensionality reduction to reduce # of parameters and computation
  - Increase nonlinearity

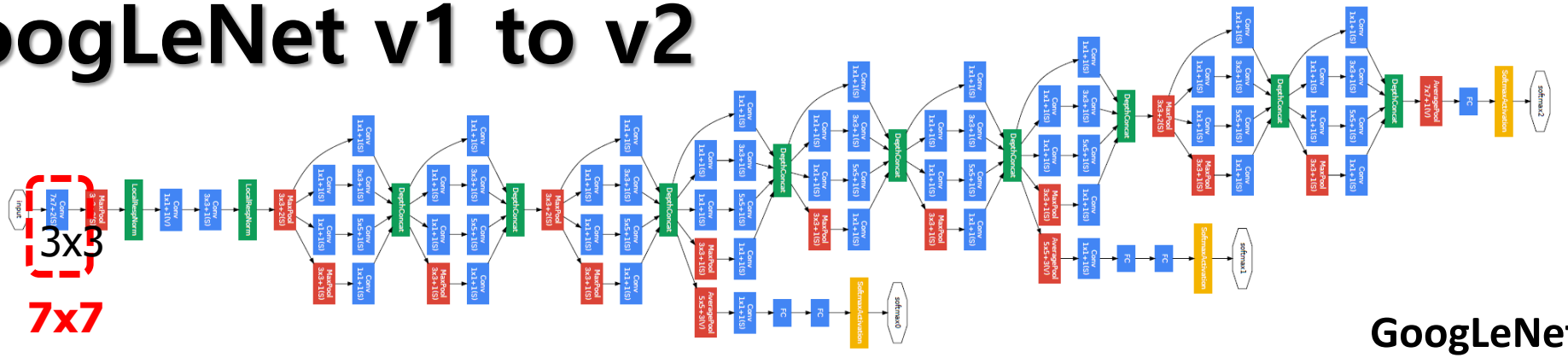




[Szegedy 2014]

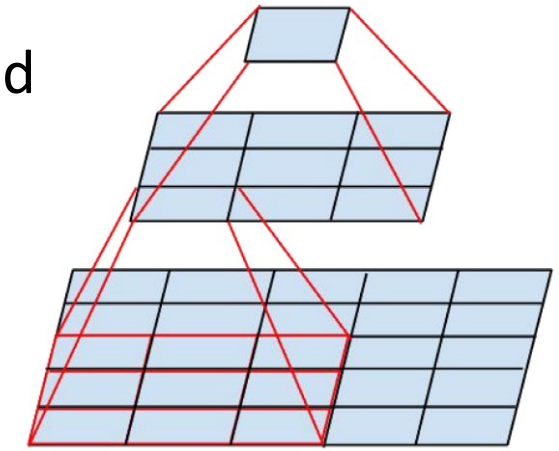
[Szegedy 2014]
[Sungjoo Yoo]

GoogLeNet v1 to v2



7x7 kernel incurs high overhead in computation/size

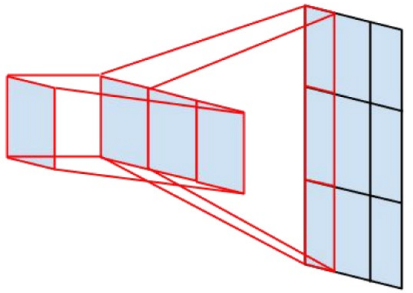
7x7 can be replaced with a cascade of smaller kernels



GoogLeNet v2

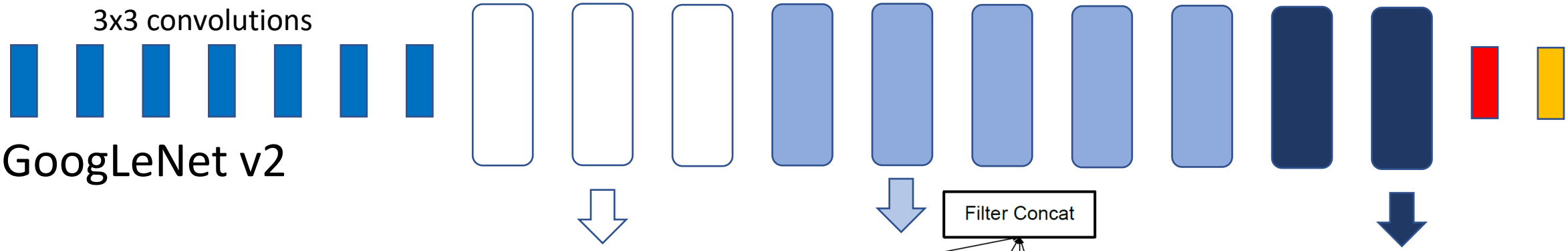
type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

GoogLeNet v1 to v2

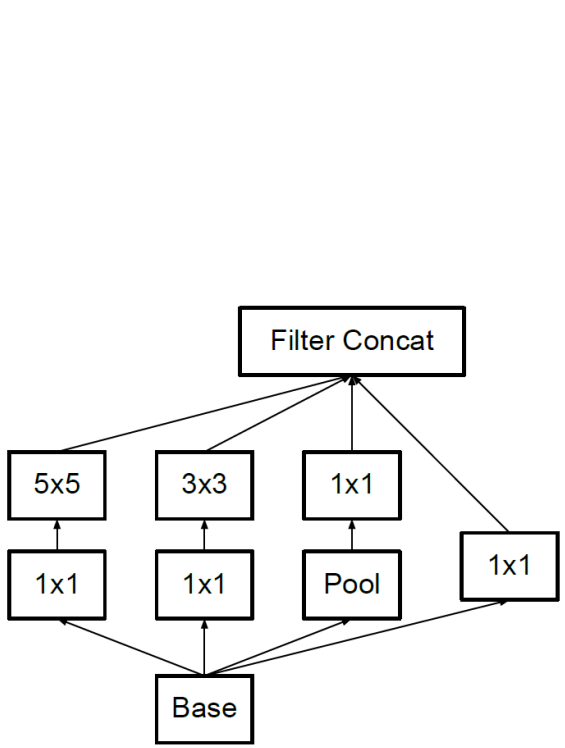


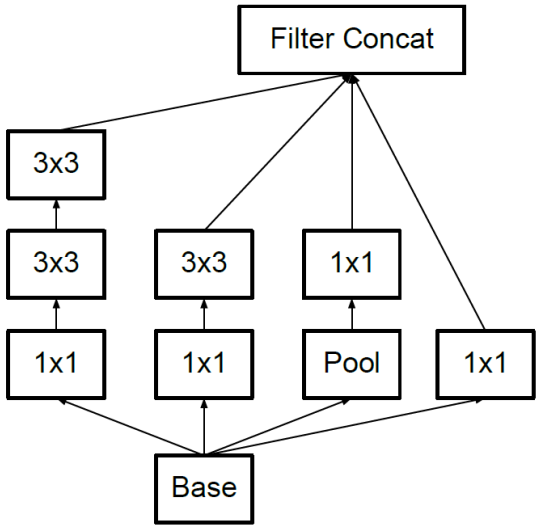
[Szegedy 2014]
[Sungjoo Yoo]

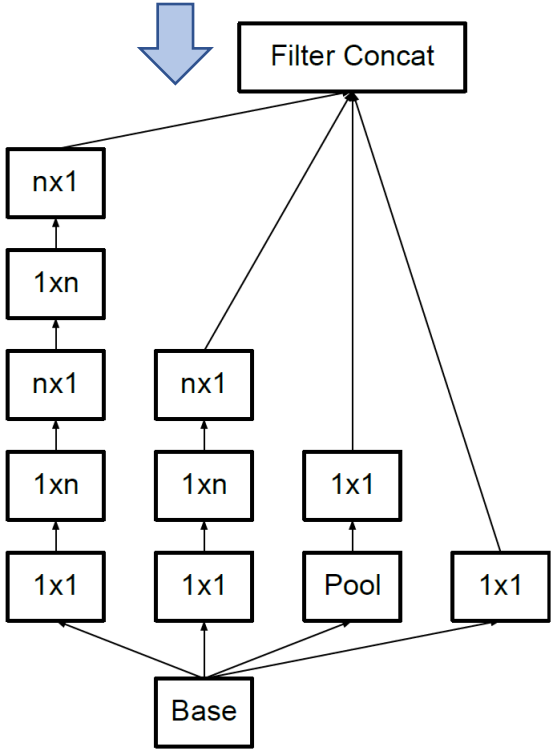
3x3 convolutions

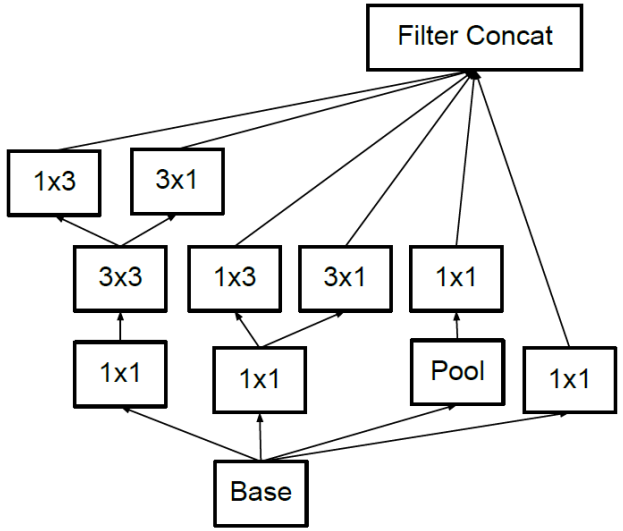


GoogLeNet v2









[Szegedy 2014]
[Sungjoo Yoo]

Accuracy of GoogLeNet v1-v2

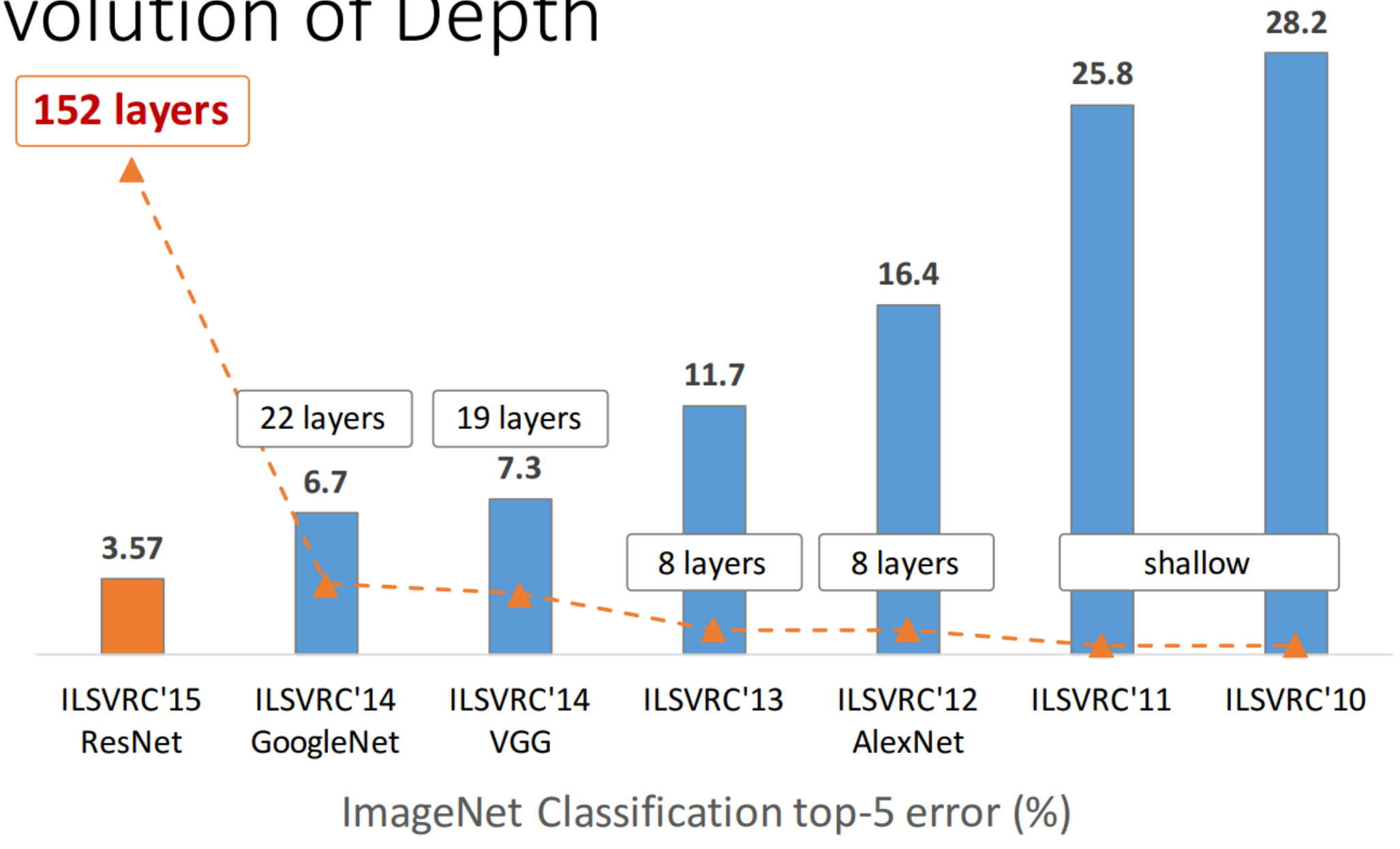
Network	Top-1 Error	Top-5 Error	Cost Bn Ops
GoogLeNet [20]	29%	9.2%	1.5
BN-GoogLeNet	26.8%	-	**1.5**
BN-Inception [7]	25.2%	7.8	2.0
Inception-v2	23.4%	-	3.8
Inception-v2 RMSProp	23.1%	6.3	3.8
Inception-v2 Label Smoothing	22.8%	6.1	3.8
Inception-v2 Factorized 7×7	21.6%	5.8	4.8
Inception-v2 BN-auxiliary	**21.2%**	**5.6%**	4.8

2.5X FLOPS
6% better accuracy

type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure 5	$35\times35\times288$
$5\times$Inception	As in figure 6	$17\times17\times768$
$2\times$Inception	As in figure 7	$8\times8\times1280$
pool	8×8	$8 \times 8 \times 2048$
linear	logits	$1 \times 1 \times 2048$
softmax	classifier	$1 \times 1 \times 1000$

ResNet

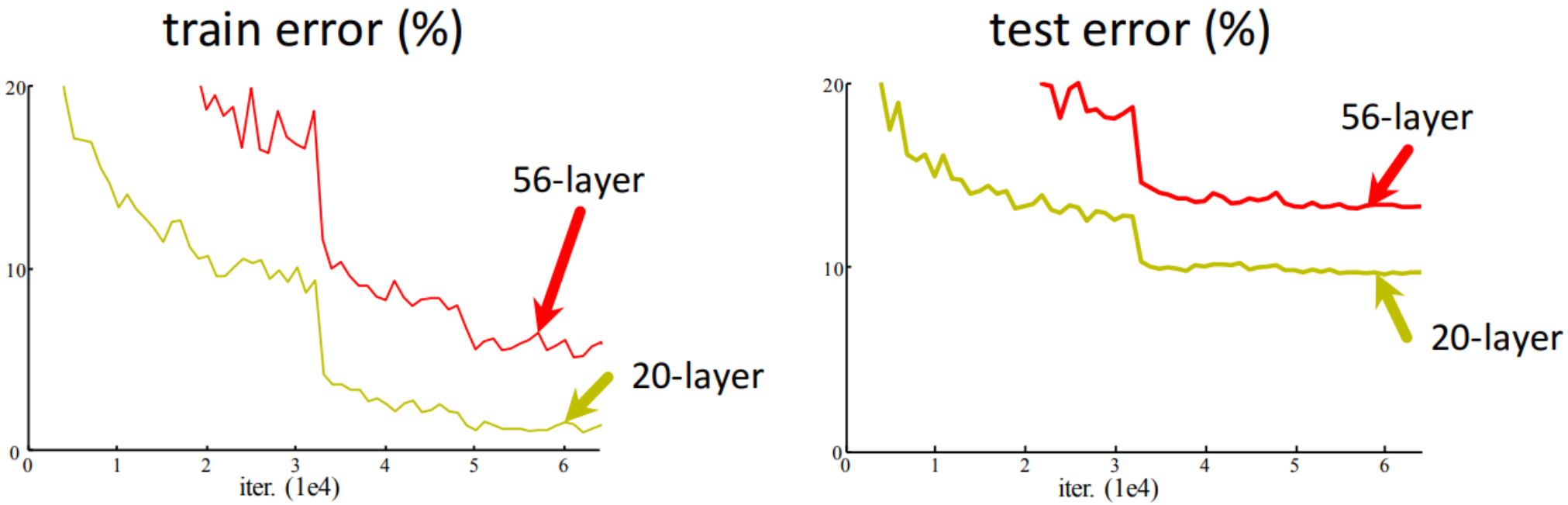
Revolution of Depth



[He, 2016]

Simply stacking layers?

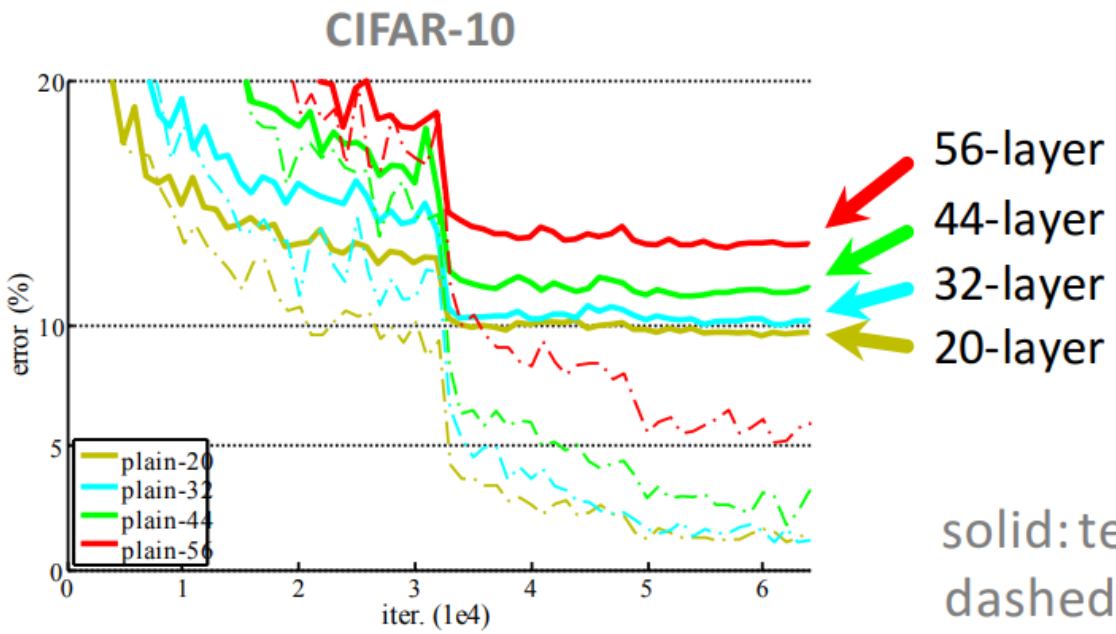
CIFAR-10

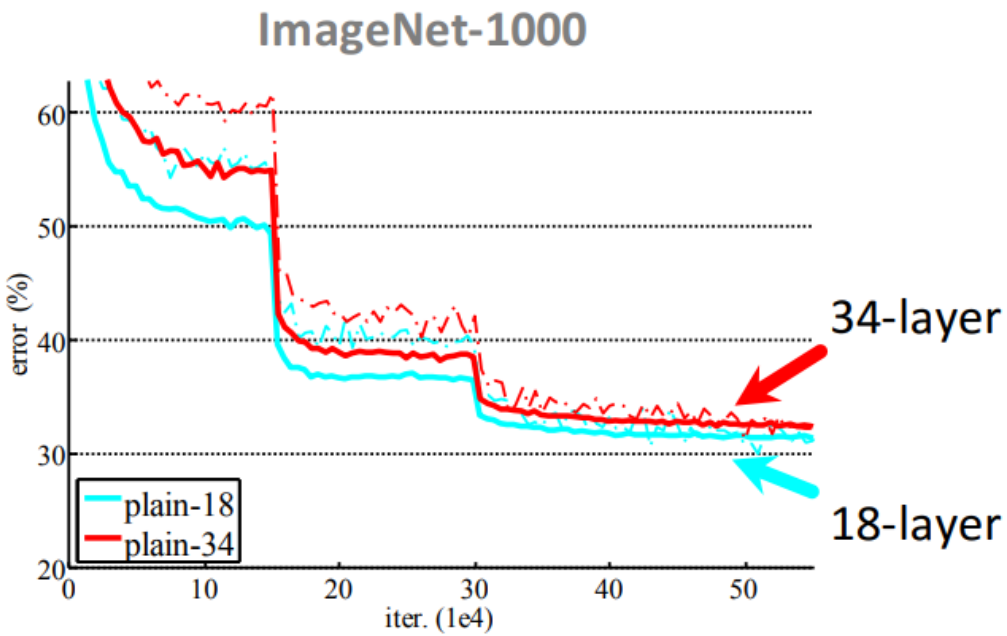


- *Plain* nets: stacking 3x3 conv layers…
- 56-layer net has **higher training error** and test error than 20-layer net

[He, 2016]

Simply stacking layers?

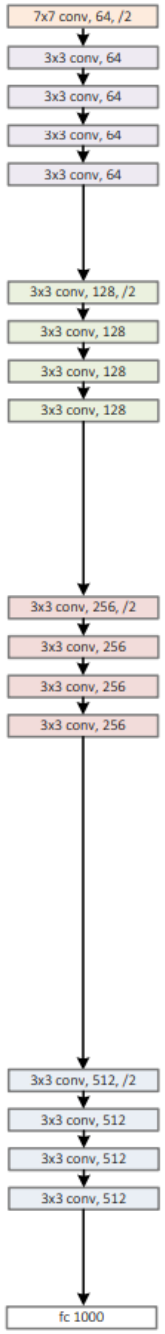


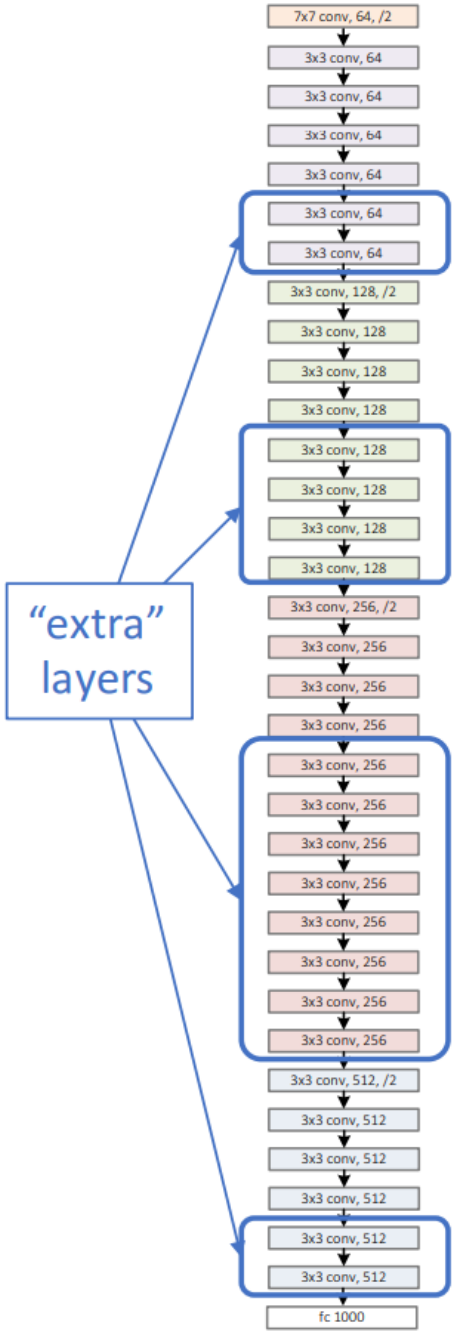


- "Overly deep" plain nets have **higher training error**
- A general phenomenon, observed in many datasets

[He, 2016]

a shallower model (18 layers)





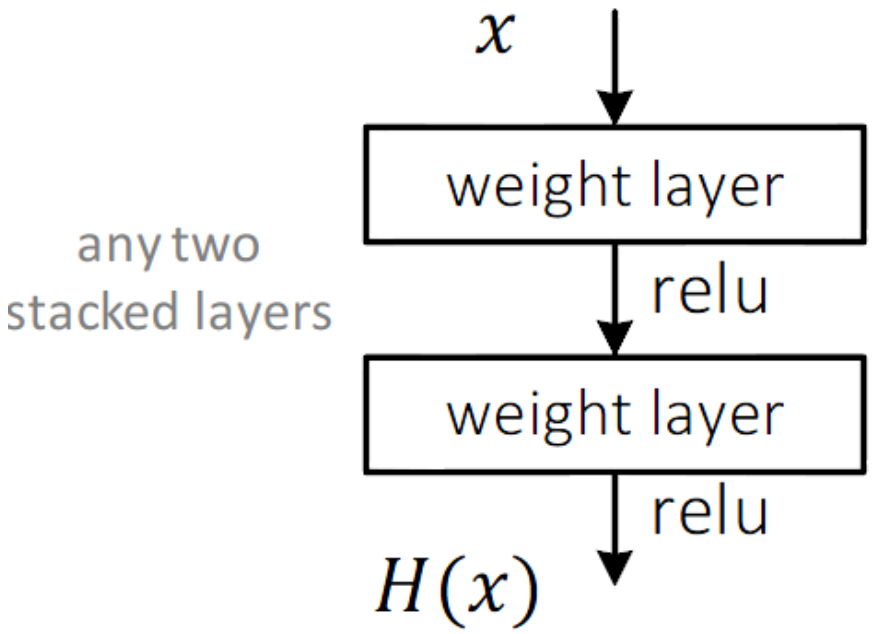
a deeper counterpart (34 layers)

- Richer solution space
- A deeper model should not have **higher training error**
- A solution *by construction*:
  - original layers: copied from a learned shallower model
  - extra layers: set as identity
  - at least the same training error
- Optimization difficulties: solvers cannot find the solution when going deeper…

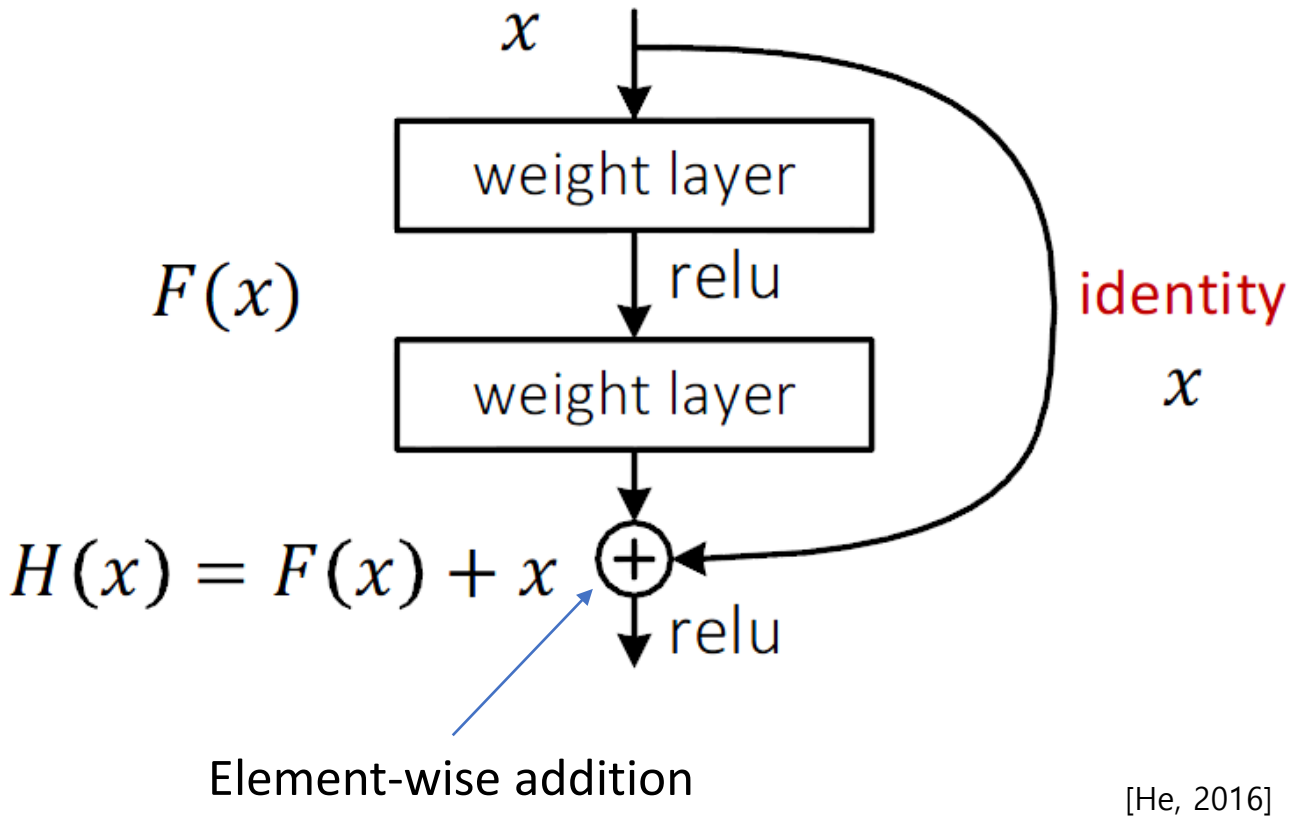
[He, 2016]

Plain vs. Residual Net

- Plain net

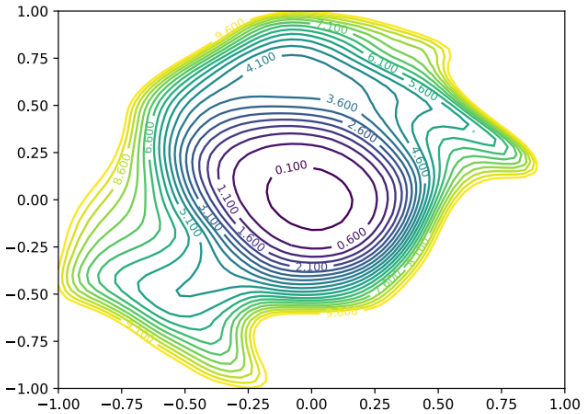


- Residual net

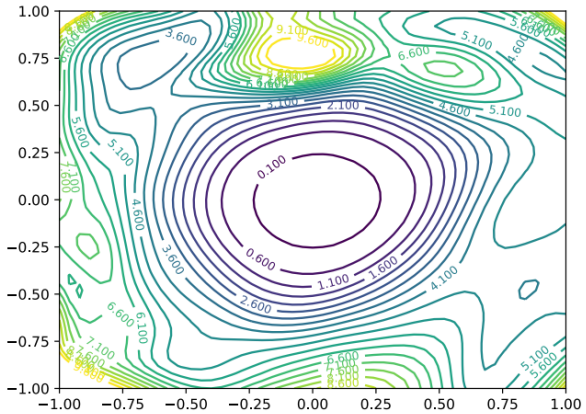


[He, 2016]

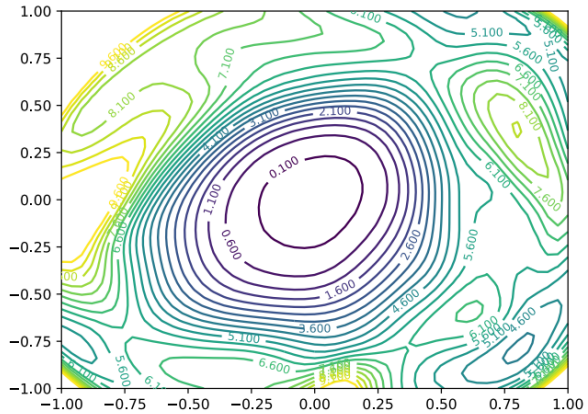
Plain vs. Residual Net



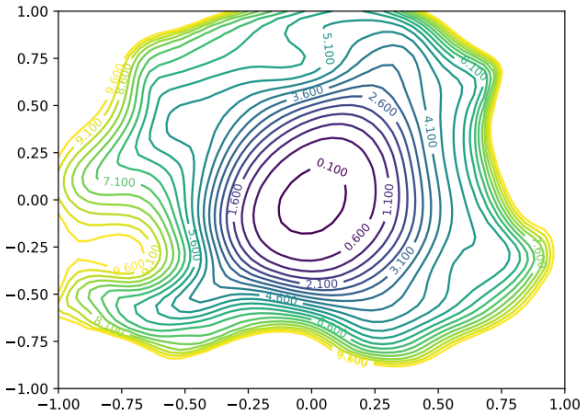
(a) ResNet-20, 7.37%



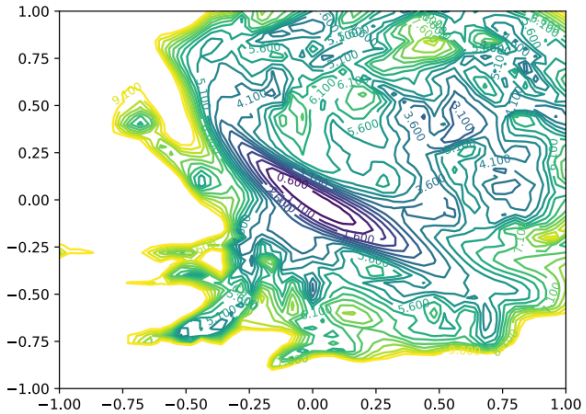
(b) ResNet-56, 5.89%



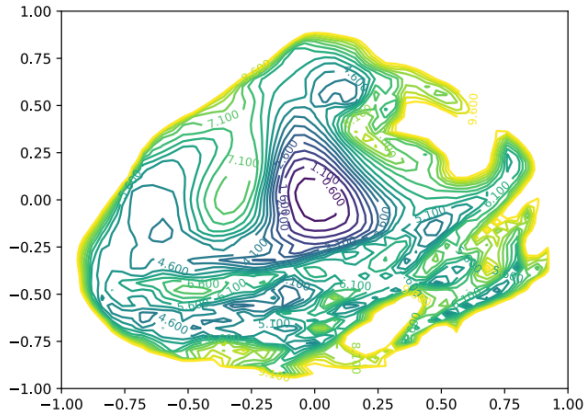
(c) ResNet-110, 5.79%



(d) ResNet-20-NS, 8.18%



(e) ResNet-56-NS, 13.31%

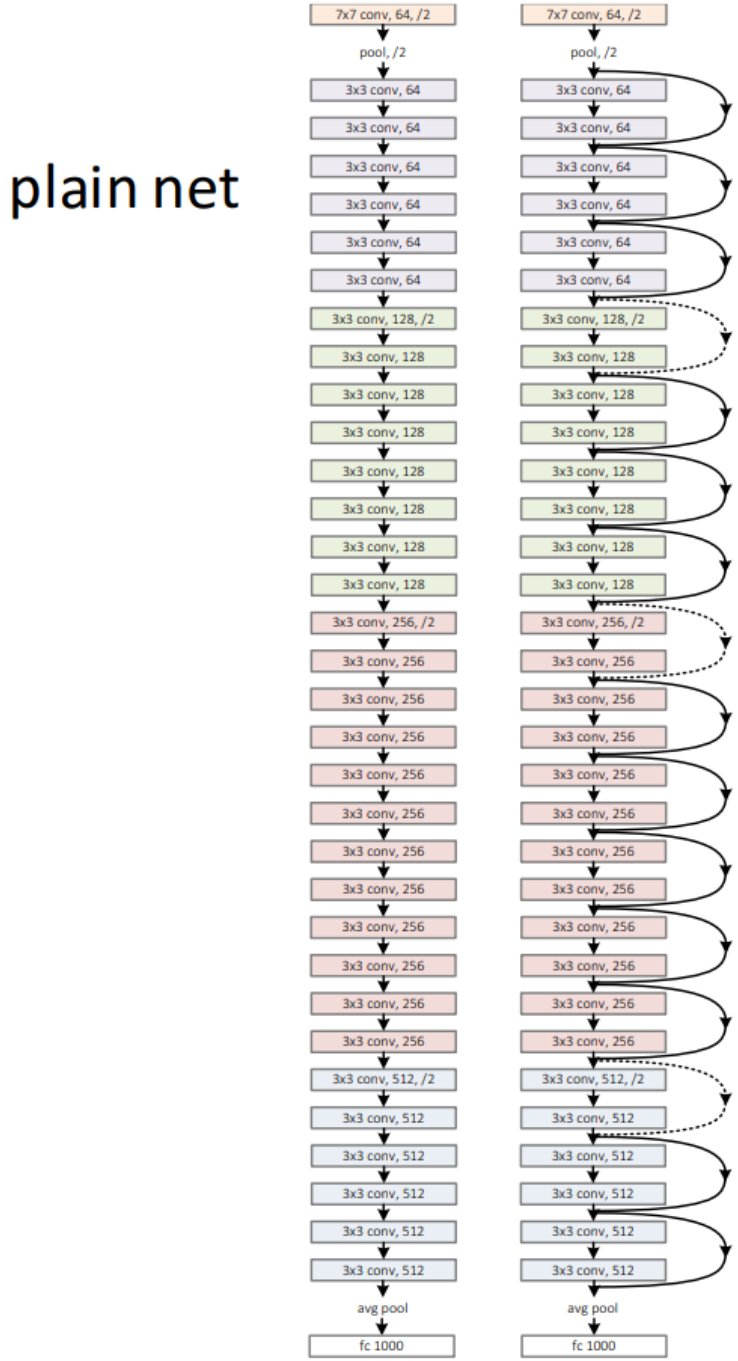


(f) ResNet-110-NS, 16.44%

[Visualizing the Loss Landscape of Neural Nets, Li]

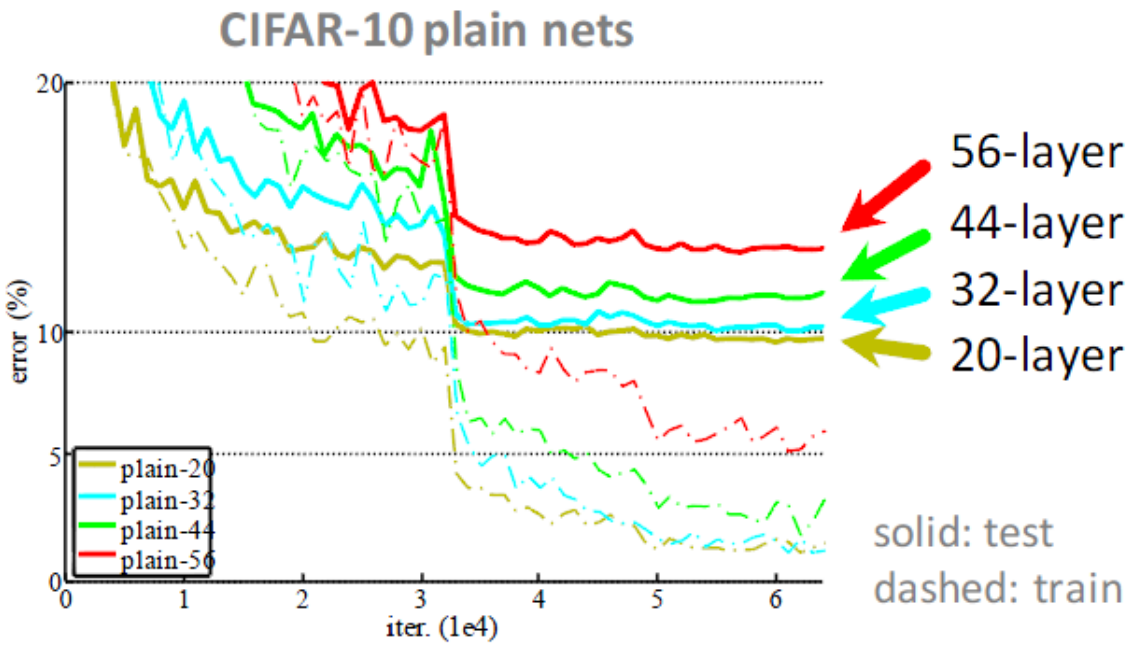
Network “Design”

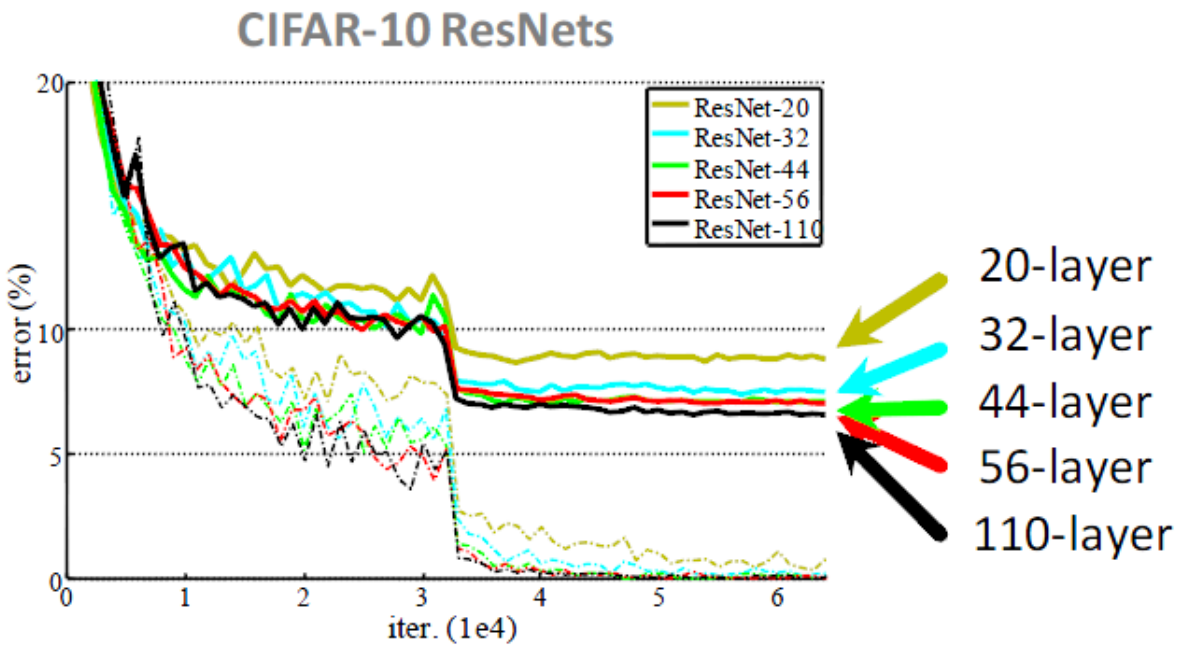
- Keep it simple
- Our basic design (VGG-style)
  - all 3x3 conv (almost)
  - spatial size /2 => # filters x2
  - Simple design; just deep!



[He, 2016]

CIFAR-10 experiments

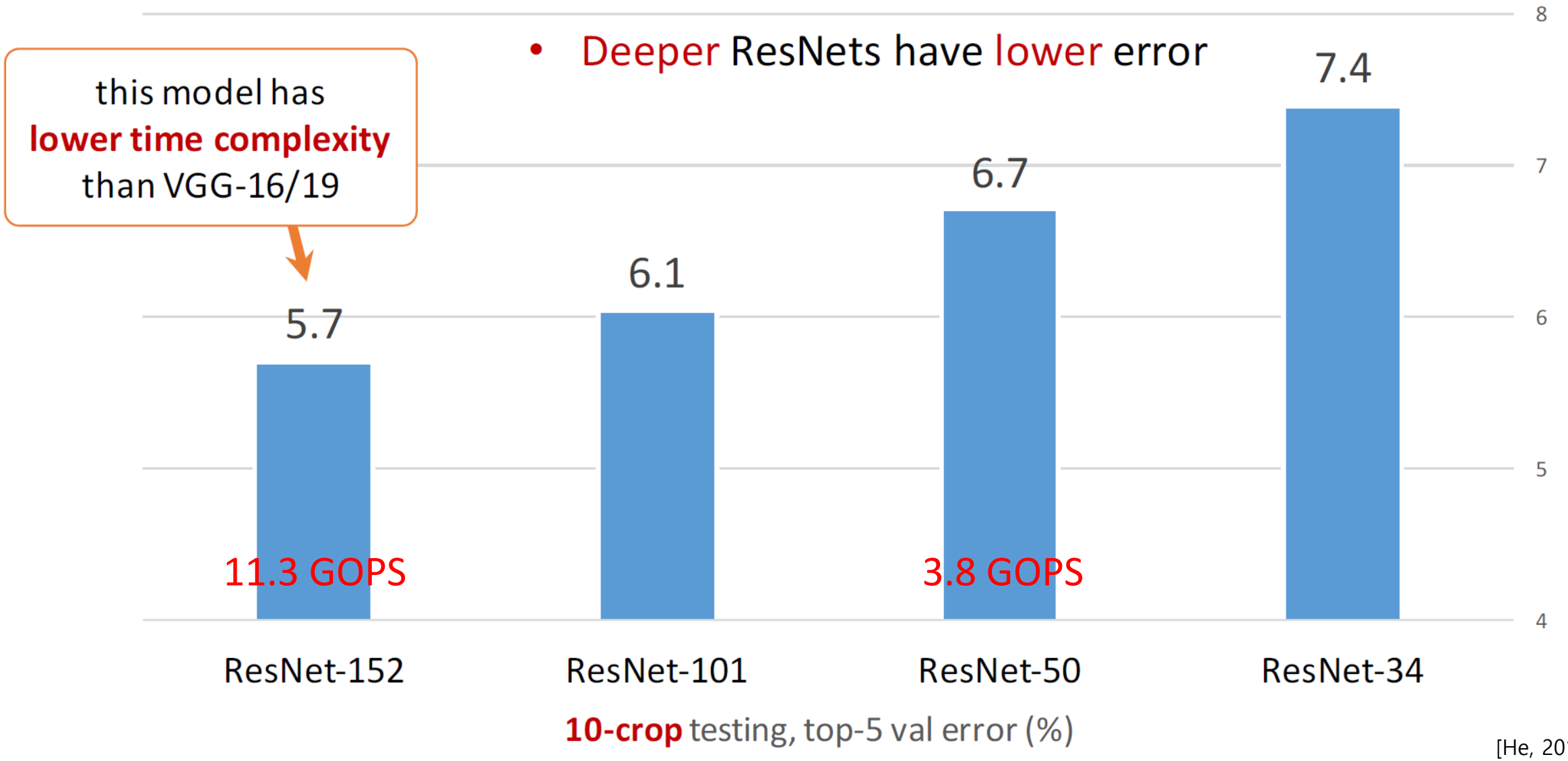




- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error

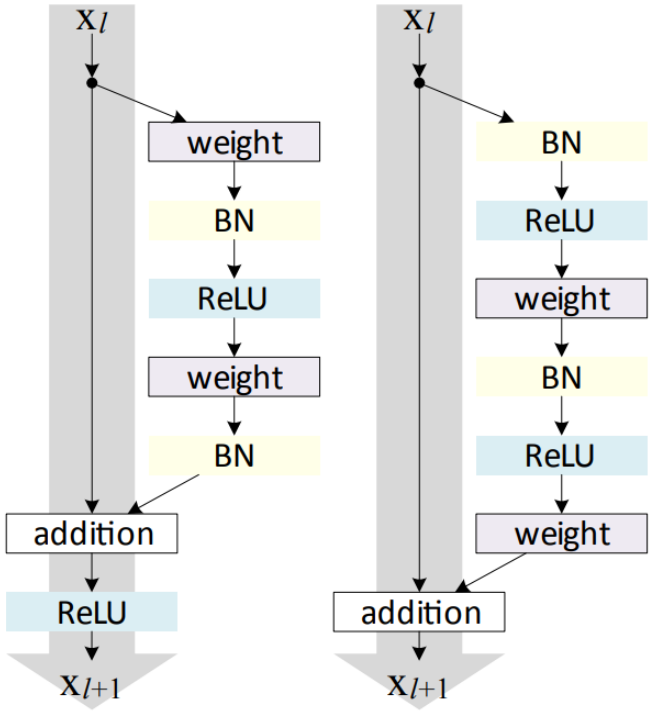
[He, 2016]

ImageNet experiments

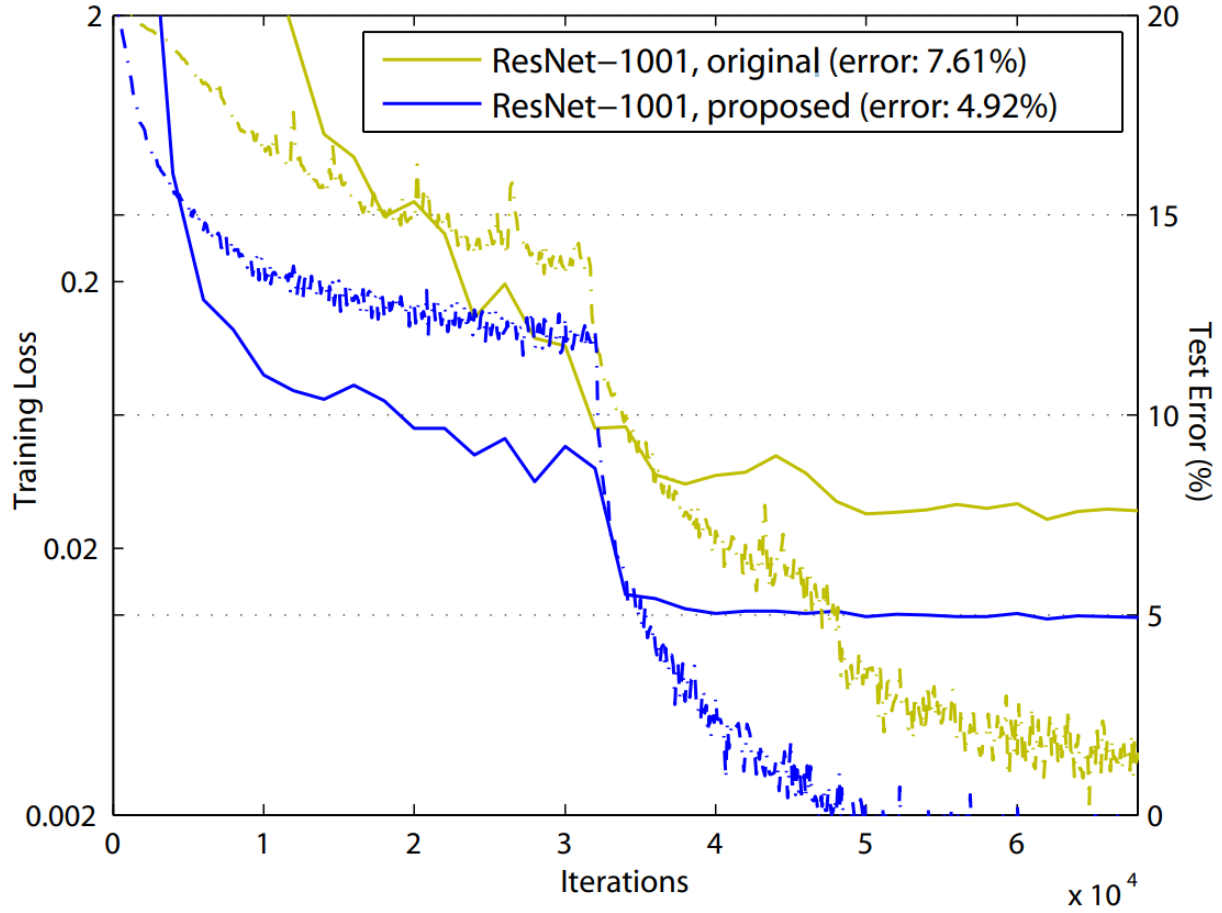


[He, 2016]

Original ResNet still has a limitation

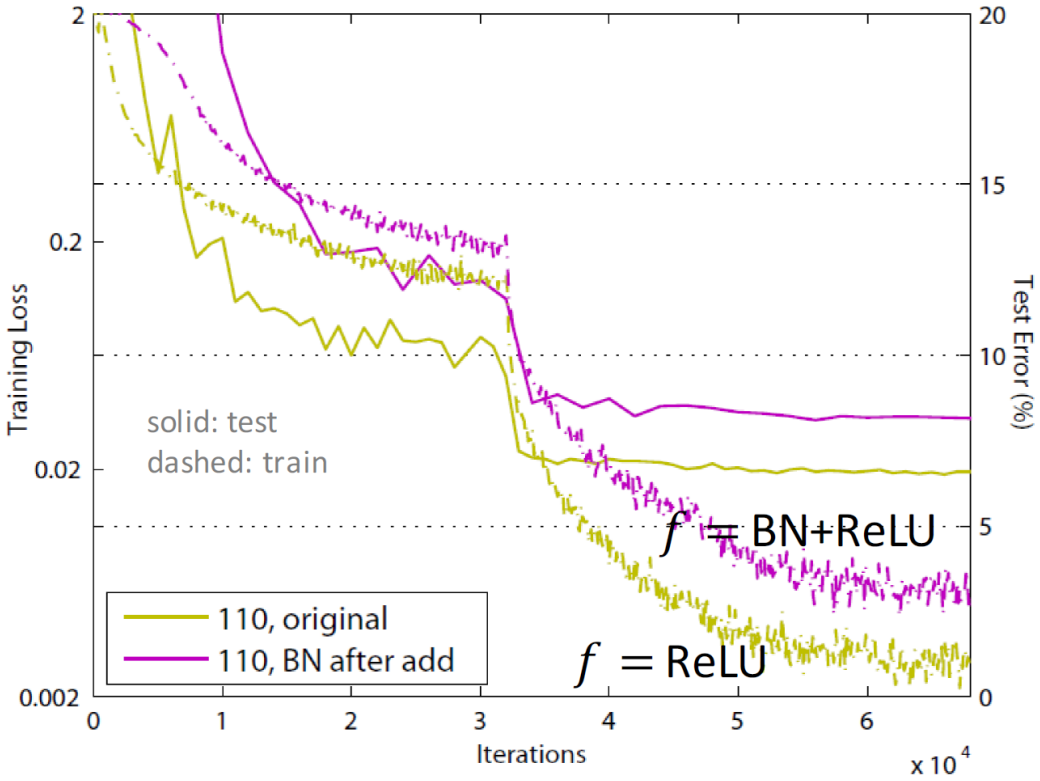


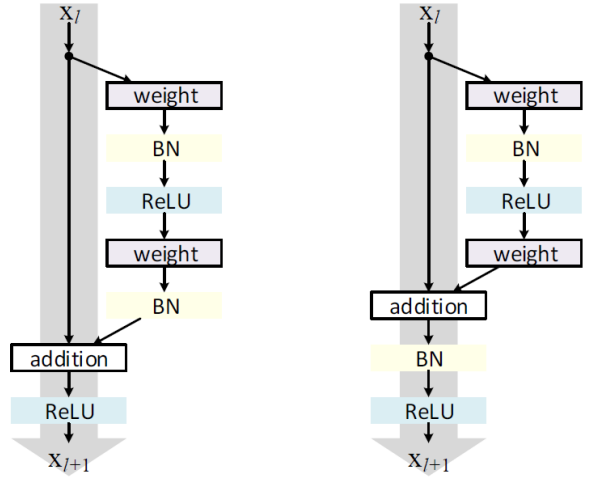
(a) original (b) proposed



[Identity Mappings in Deep Residual Networks, He]

Complex Shortcut Gives Poor Results



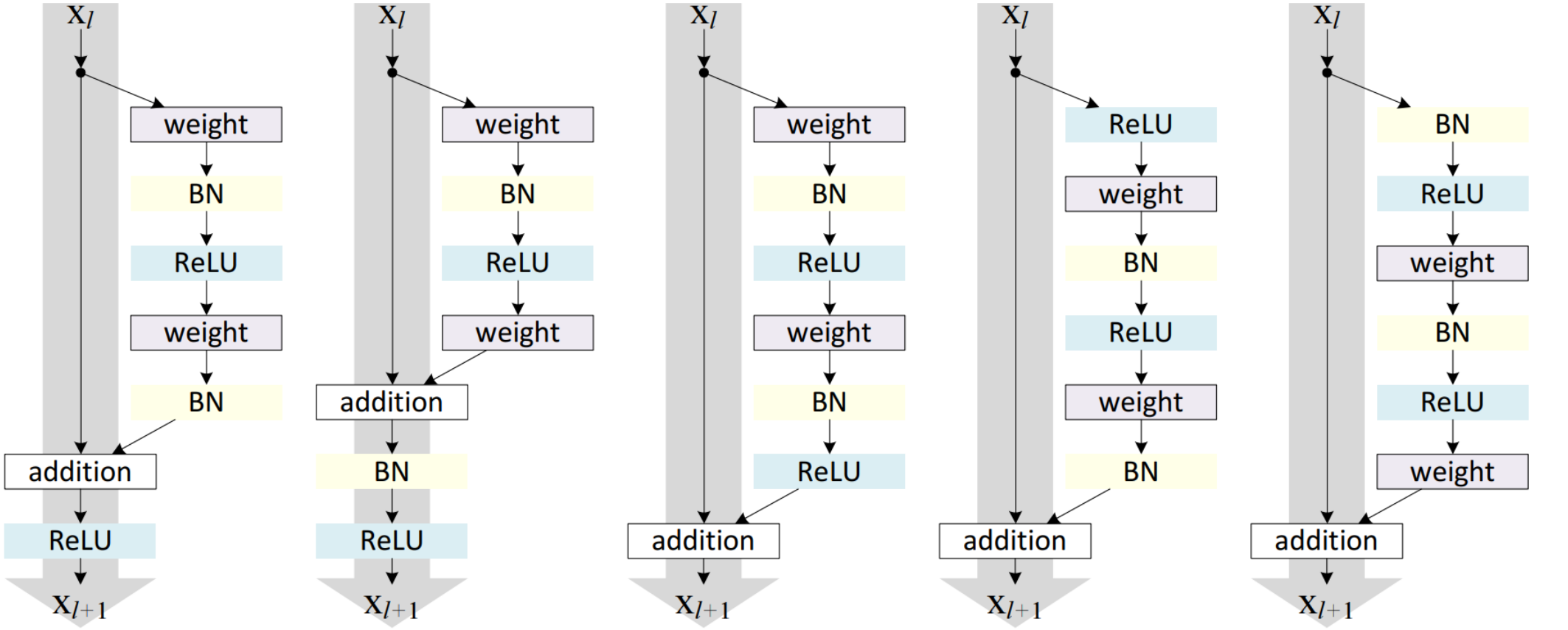


$f = \text{ReLU}$ $f = \text{BN+ReLU}$

- BN could block prop
- Keep the shortest pass as smooth as possible

[Identity Mappings in Deep Residual Networks, He]
[Sungjoo Yoo]

Various Identity Design



(a) original

(b) BN after addition

(c) ReLU before addition

(d) ReLU-only pre-activation

(e) **full pre-activation**

[Identity Mappings in Deep Residual Networks, He]

CIFAR Reults

CIFAR-10	error (%)
NIN [15]	8.81
DSN [16]	8.22
FitNet [17]	8.39
Highway [7]	7.72
All-CNN [14]	7.25
ELU [12]	6.55
FitResNet, LSUV [18]	5.84
ResNet-110 [1] (1.7M)	6.61
ResNet-1202 [1] (19.4M)	7.93
ResNet-164 [ours] (1.7M)	5.46
ResNet-1001 [ours] (10.2M)	4.92 (4.89±0.14)
ResNet-1001 [ours] (10.2M)†	**4.62** (4.69±0.20)

CIFAR-100	error (%)
NIN [15]	35.68
DSN [16]	34.57
FitNet [17]	35.04
Highway [7]	32.39
All-CNN [14]	33.71
ELU [12]	24.28
FitNet, LSUV [18]	27.66
ResNet-164 [1] (1.7M)	25.16
ResNet-1001 [1] (10.2M)	27.82
ResNet-164 [ours] (1.7M)	24.33
ResNet-1001 [ours] (10.2M)	**22.71** (22.68±0.22)

[Identity Mappings in Deep Residual Networks, He]

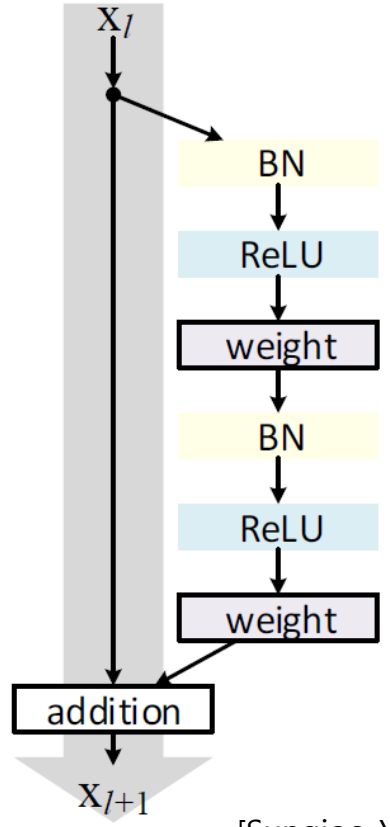
ImageNet Results

method	augmentation	train crop	test crop	top-1	top-5
ResNet-152, original Residual Unit [1]	scale	224×224	224×224	23.0	6.7
ResNet-152, original Residual Unit [1]	scale	224×224	320×320	21.3	5.5
ResNet-152, **pre-act** Residual Unit	scale	224×224	320×320	21.1	5.5
ResNet-200, original Residual Unit [1]	scale	224×224	320×320	21.8	6.0
ResNet-200, **pre-act** Residual Unit	scale	224×224	320×320	**20.7**	**5.3**
ResNet-200, **pre-act** Residual Unit	scale+asp ratio	224×224	320×320	**20.1**†	**4.8**†
Inception v3 [19]	scale+asp ratio	299×299	299×299	21.2	5.6

[Identity Mappings in Deep Residual Networks, He]

ResNet: Summary

- Keep the shortest path as smooth as possible
  - by making h and f identity
  - forward/backward signals directly flow through this path
- Features of any layers are additive outcomes
- 1000-layer ResNets can be easily trained and have better accuracy

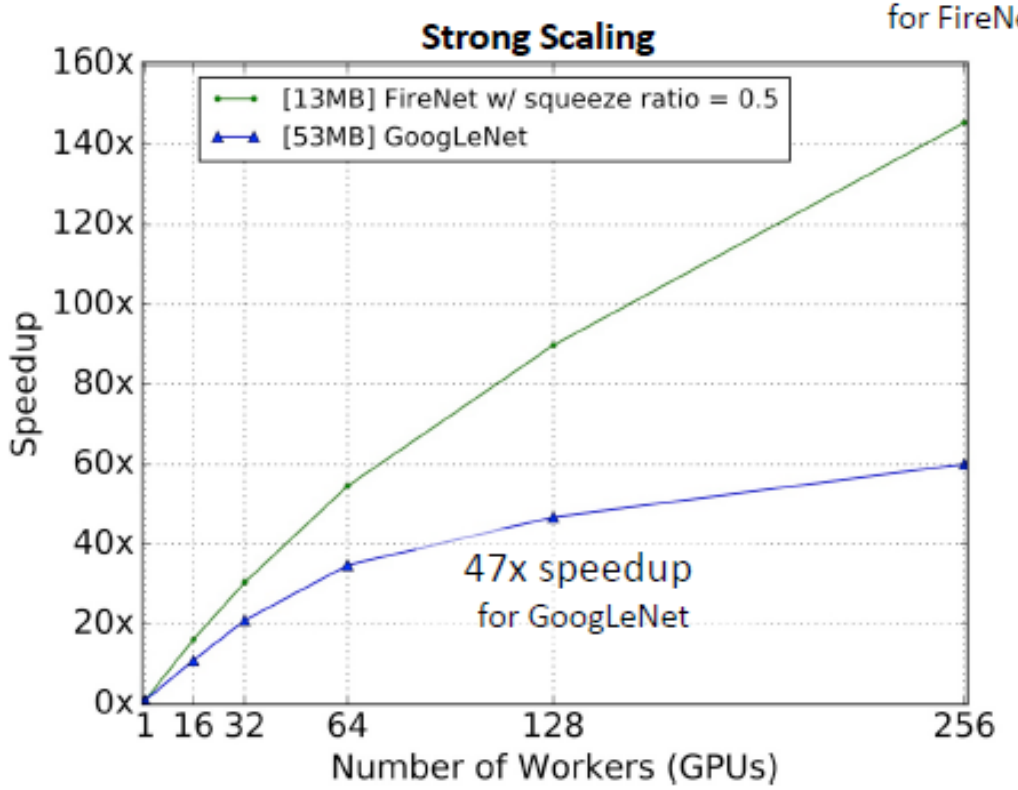


[Sungjoo Yoo]

Efficient CNNs

Small Model Advantages - 1

- Fewer parameter weights means bigger opportunities for scaling training – 145X speedup on 256 GPUs

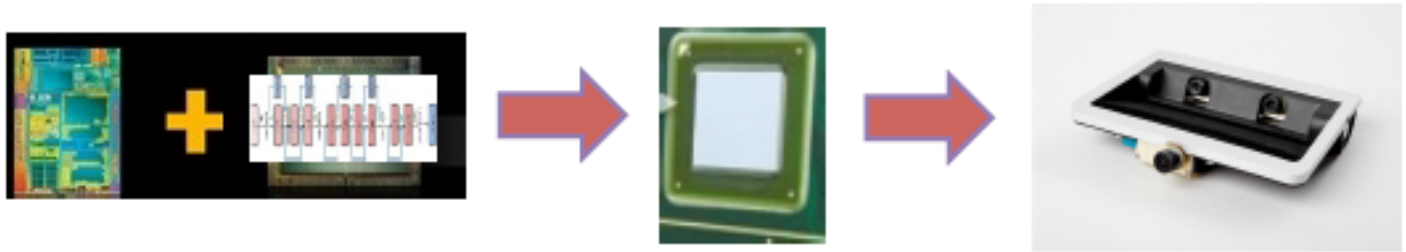


Using FireCaffe on the Titan cluster CVPR 2016

[Forrest Iandola and Kurt Keutzer]

Small Model Advantages - 2

- SqueezeNet's smaller number of weights enables complete on-chip integration of CNN model with weights – no need for off-chip memory
  - Dramatically reduces the energy for computing inference
  - Gives the potential for pushing the data processing (i.e. CNN model) up-close and personal to the data gathering (e.g. onboard cameras and other sensors)



Single-chip Integration of CNN Model

Closer integration with sensor (e.g. Camera)

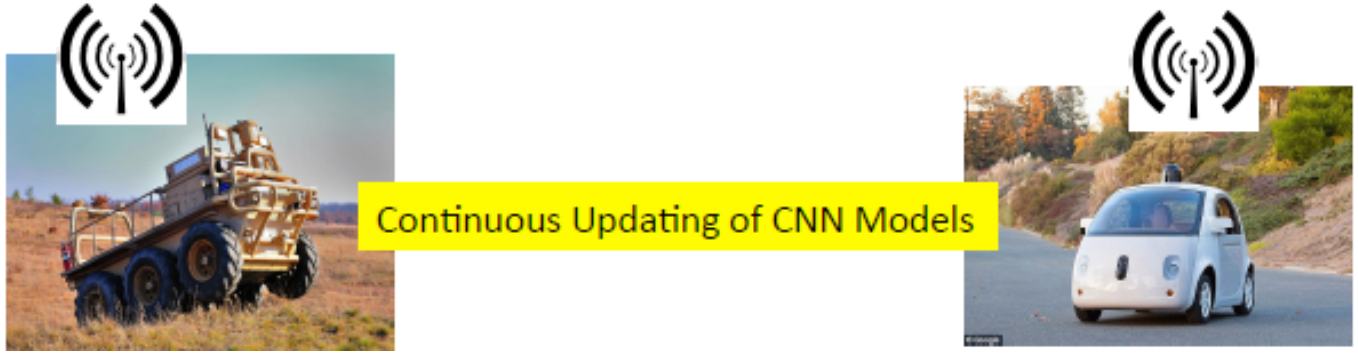
Limited memory of embedded devices makes small models absolutely essential for many applications

•

[Forrest Iandola and Kurt Keutzer]

Small Model Advantages - 3

- Small models enable continuous wireless updates of models
- Each time any sensor discovers a new image/situation that requires retraining, all models should be updated
- Data is uploaded to cloud and used for retraining
- But … how to update all the vehicles that are running the model?
- At 500KB downloading new model parameters is easy.

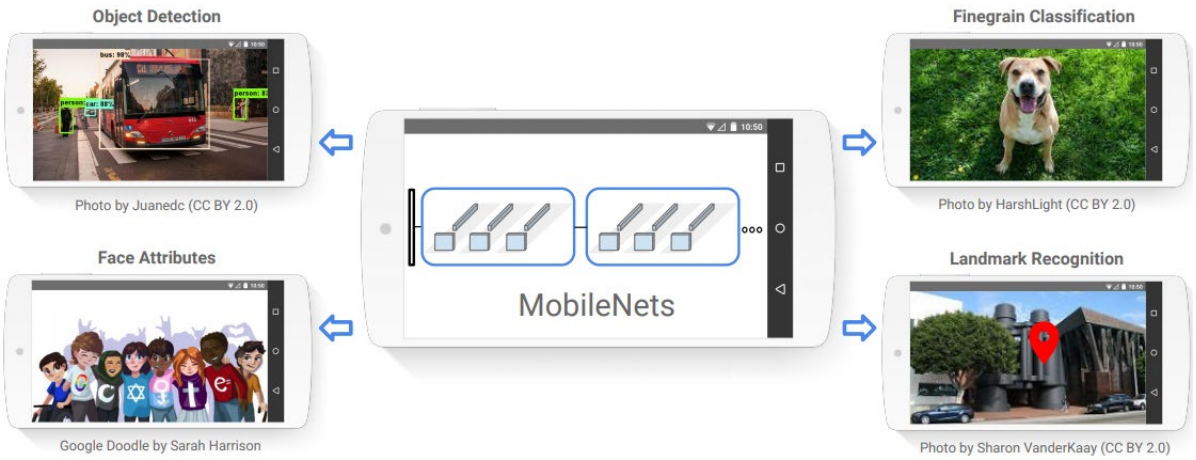


[Forrest Iandola and Kurt Keutzer]

Efficient CNNs?

- Best CNN ≠ the most accurate CNN

- ‘Practically’ best?
  - High-quality output, or abundant information within feature map
  - Small neural network size
    - Typical App size should be smaller than 100 Mb
  - Low computation overhead
    - Snapdragon 865 GPU~1250 Gflops
    - Real-time constraint
  - Low battery consumption



Way to Make Efficient

- Improving operator design
  - Depthwise-seperable convolution
  - Shift-based convolution

- Improving architecture design
  - Residual structure
  - Inverted-residual structure
  - Squeeze-excitation module

- Applying optimization techniques
  - Low-precision computation
  - Pruning
  - Low-rank approximation

Squeeze-Net

SqueezeNet

- **Strategy 1:**

  Replace 3x3 filters with 1x1 filters:

  - 9X fewer parameters

- **Strategy 2:**

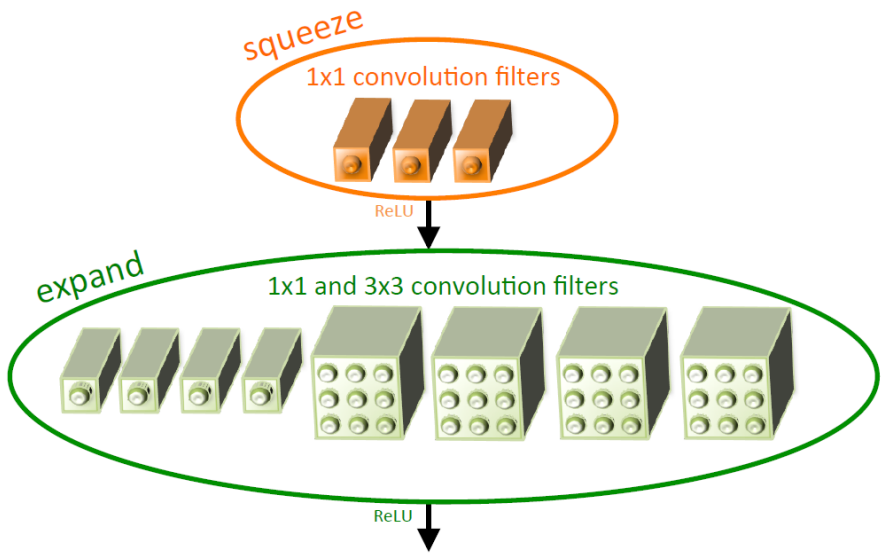
  Decrease the number of input channels to 3x3 filters

  - (number of input channels) * (number of filters) * (3*3).

- **Strategy 3:**

  Downsample late in the network so that convolution layers have large activation maps

  - Intuition: delayed downsampling => large activation maps => higher classification accuracy, with all else held equal



Fire module

```
class Fire(nn.Module):

    def __init__(self, inplanes, squeeze_planes,
                 expand1x1_planes, expand3x3_planes):
        super(Fire, self).__init__()
        self.inplanes = inplanes
        self.squeeze = nn.Conv2d(inplanes, squeeze_planes, kernel_size=1)
        self.squeeze_activation = nn.ReLU(inplace=True)
        self.expand1x1 = nn.Conv2d(squeeze_planes, expand1x1_planes,
                                   kernel_size=1)
        self.expand1x1_activation = nn.ReLU(inplace=True)
        self.expand3x3 = nn.Conv2d(squeeze_planes, expand3x3_planes,
                                   kernel_size=3, padding=1)
        self.expand3x3_activation = nn.ReLU(inplace=True)

    def forward(self, x):
        x = self.squeeze_activation(self.squeeze(x))
        return torch.cat([
            self.expand1x1_activation(self.expand1x1(x)),
            self.expand3x3_activation(self.expand3x3(x))
        ], 1)
```

[Iandola, 2016]
[Mingyang Zheng]

SqueezeNet

- **Strategy 1:**

  Replace 3x3 filters with 1x1 filters:

  - 9X fewer parameters

- **Strategy 2:**

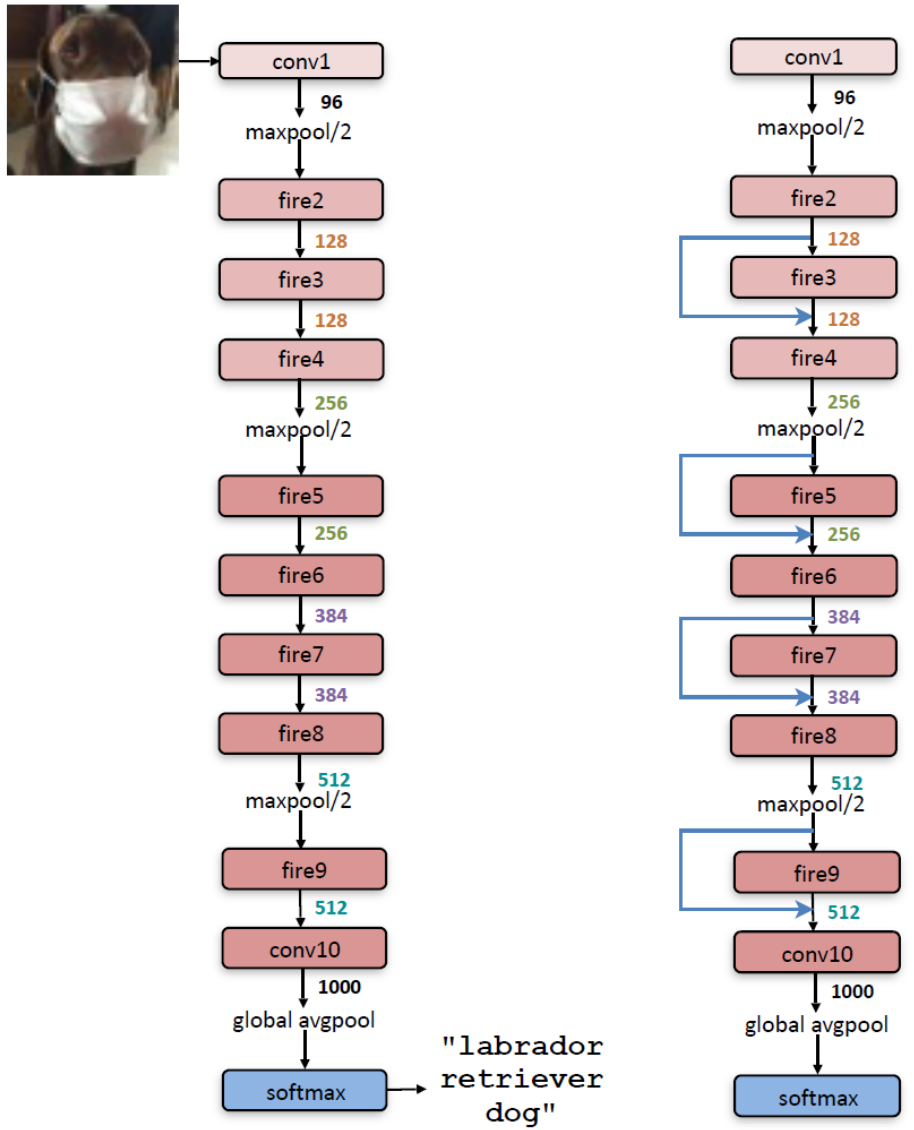
  Decrease the number of input channels to 3x3 filters

  - (number of input channels) * (number of filters) * (3*3).

- **Strategy 3:**

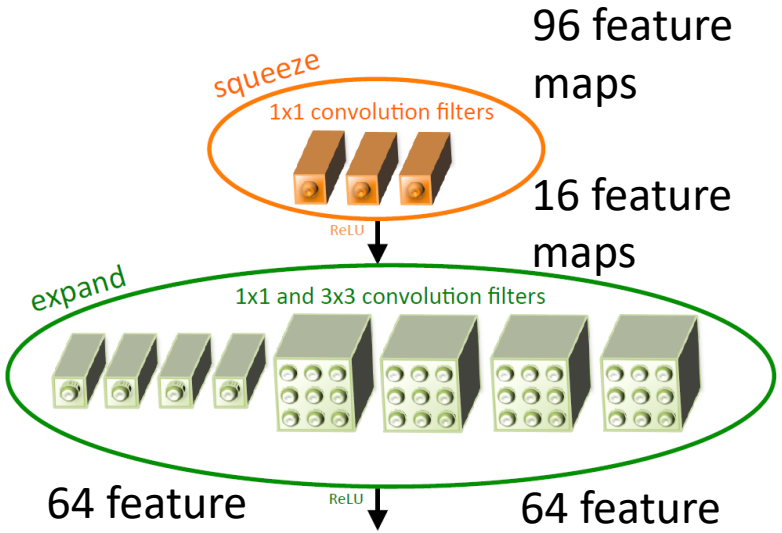
  Downsample late in the network so that convolution layers have large activation maps

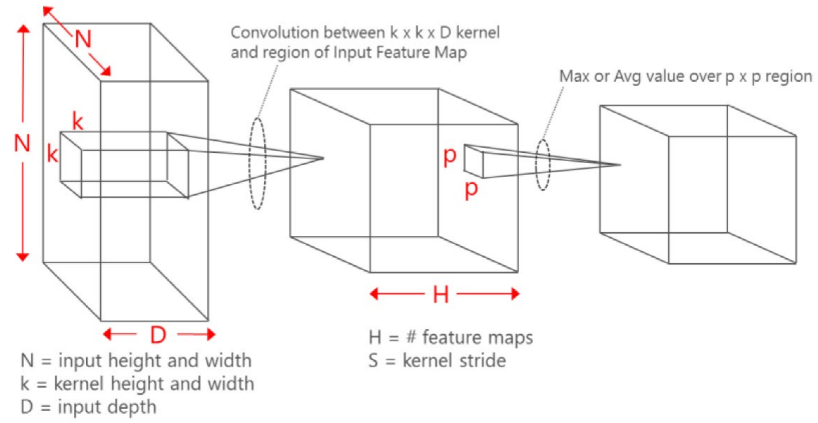
  - Intuition: delayed downsampling => large activation maps => higher classification accuracy, with all else held equal



[Iandola, 2016]
[Mingyang Zheng]

SqueezeNet gives 1.2M parameters (vs. 61M)





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)

[Iandola, 2016]
[Sungjoo Yoo]

Accuracy of SqueezeNet

- Compared to AlexNet, SqueezeNet offers comparable accuracy with much smaller parameters
- By applying additional compression technique (Pruning + Quantization + non-lossy compression), we could further minimize accuracy.

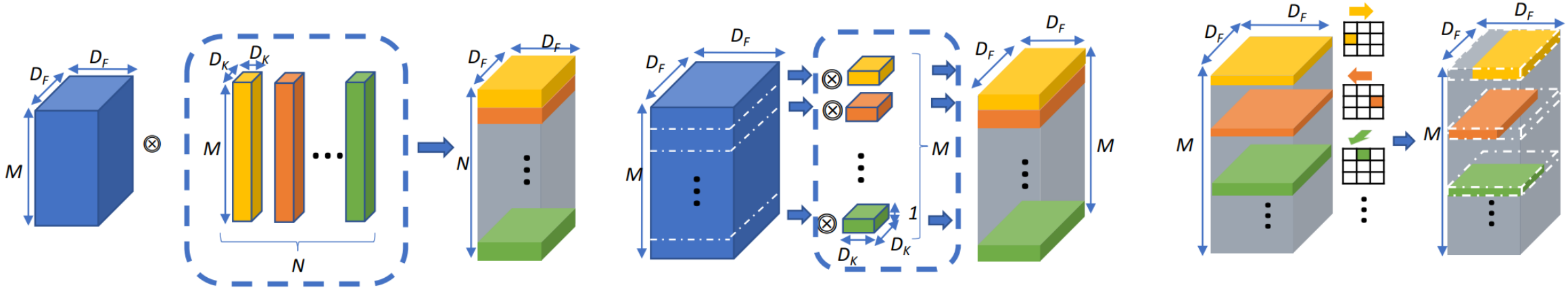
CNN architecture	Compression Approach	Data Type	Original → Compressed Model Size	Reduction in Model Size vs. AlexNet	Top-1 ImageNet Accuracy	Top-5 ImageNet Accuracy
AlexNet	None (baseline)	32 bit	240MB	1x	57.2%	80.3%
AlexNet	SVD (Denton et al., 2014)	32 bit	240MB → 48MB	5x	56.0%	79.4%
AlexNet	Network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

[Iandola, 2016]

Shift Operation

Spatial Convolution May Not Necessary

- Basic idea: replace spatial convolution by shift operation and only rely on 1x1 convolution for feature extraction



(a) Spatial Convolution

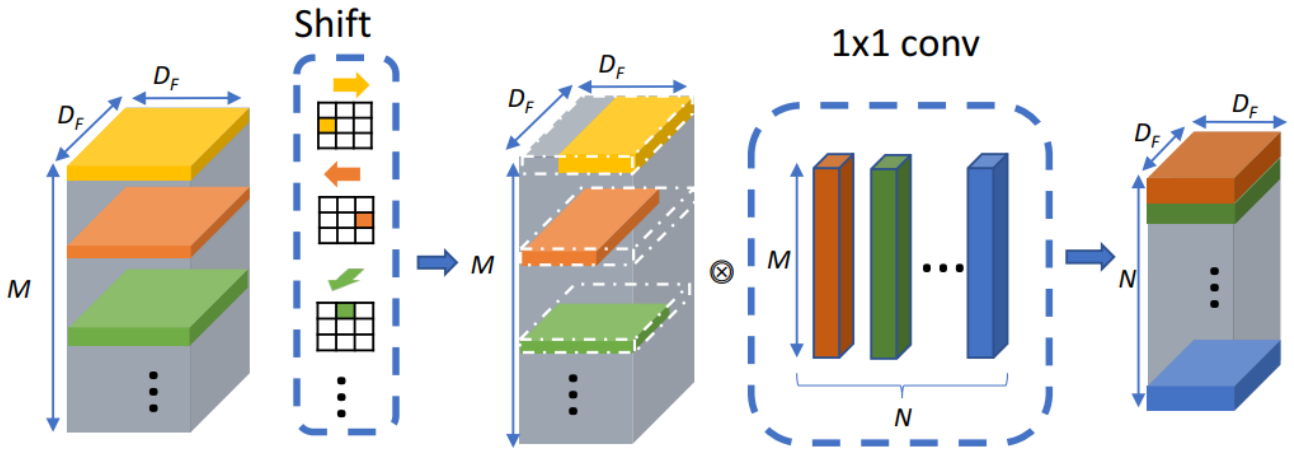
(b) Depth-wise convolution

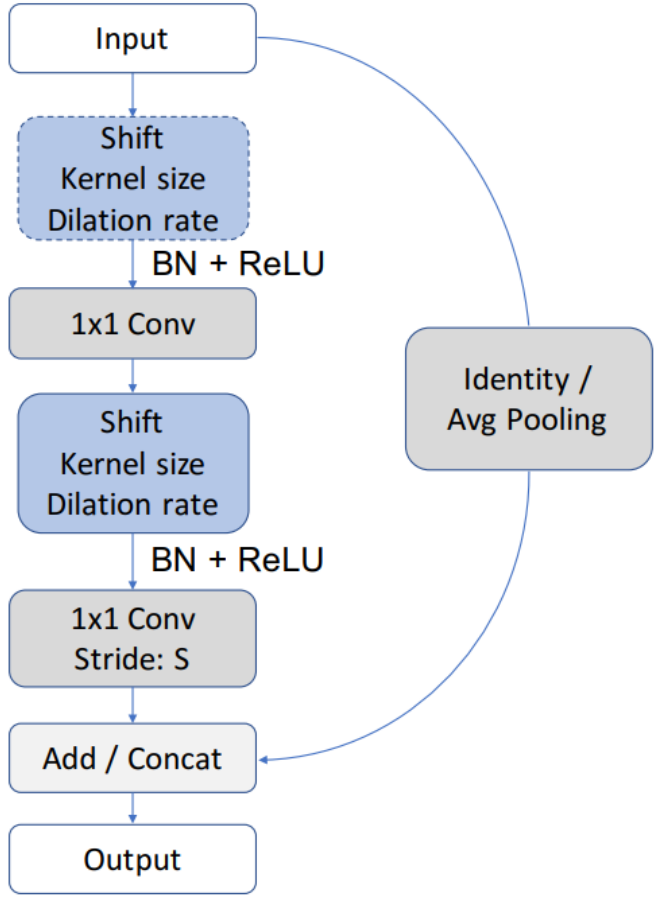
(c) Shift

[Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions]

Shift is Efficient

- Hardware is well-optimized for vector/matrix operation
  - SIMD pipeline / systolic array
- High-dimension operation is hard to handle
  - Data duplication / replacement is required
- With shift operation, spatial operation is not necessary
  - Minimize hardware circuit for high-order tensor handling





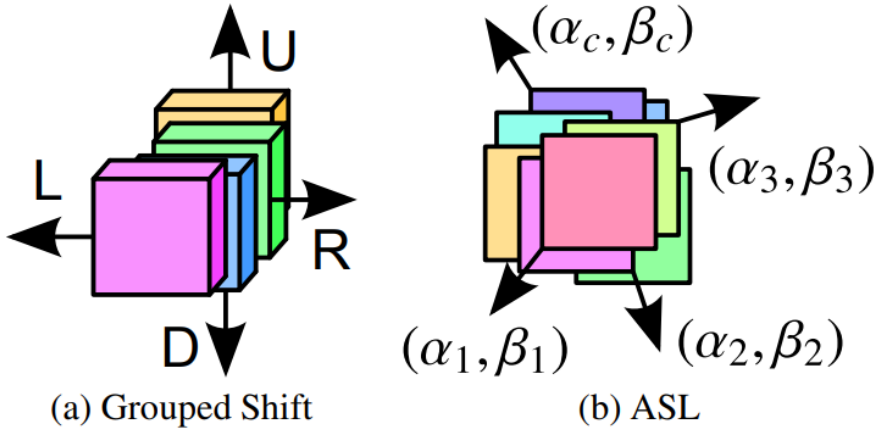
[Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions]

Active Shift

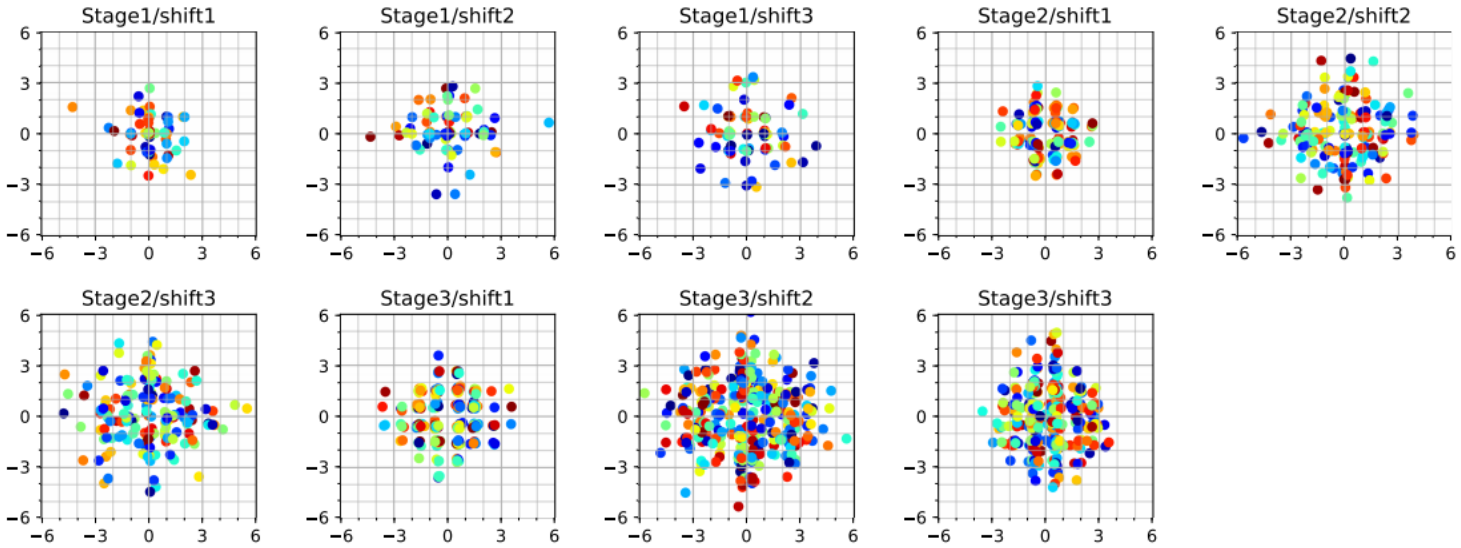
- Relaxation for grouped shift: active shift

$$O_{c,i,j} = \sum_{(n,m)\in\Omega} I_{c,n,m} \cdot (1-|i+\alpha_c - n|)(1-|j+\beta_c - m|)$$

  - Problem? Unrealistic implementation due to bilinear interpolation

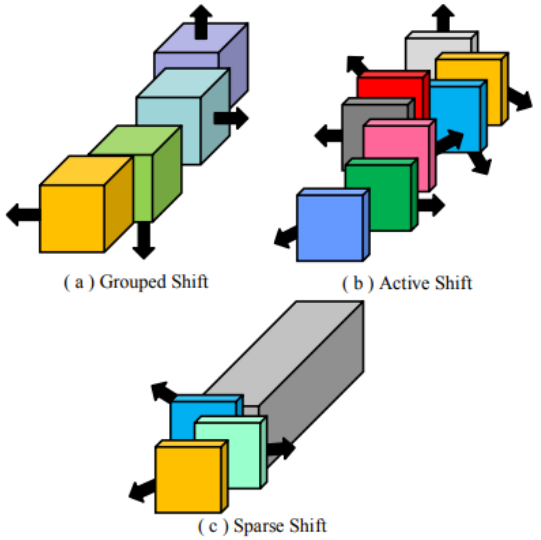


(a) Grouped Shift (b) ASL



[Constructing Fast Network through Deconstruction of Convolution]

Active Shift + Quantization



- Adopt round approximation for shift parameters
  - $O_{c,i,j} = I_{c,i+|\alpha_c|^*,j+|\beta_c|^*}$

- Gradient approximation: Straight-through estimation

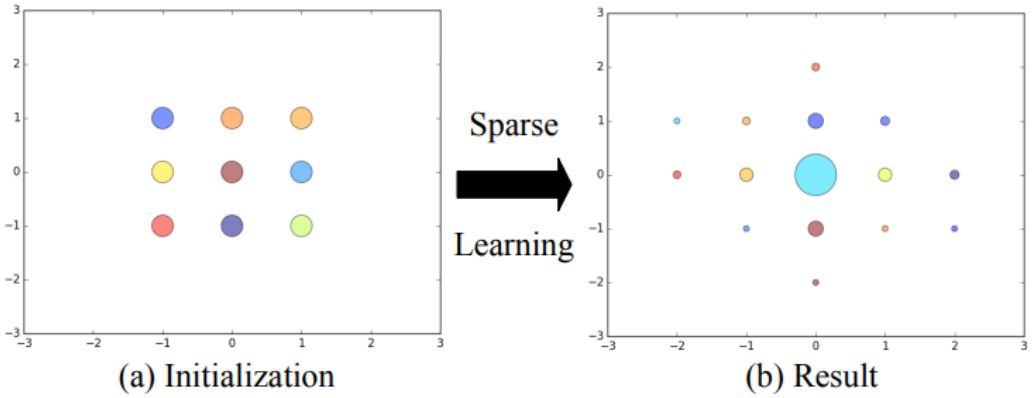
$$\frac{\partial \mathcal{L}}{\partial \alpha_c} = \sum_{i}^{w} \sum_{j}^{n} \frac{\partial \mathcal{L}}{\partial O_{c,i,j}} \sum_{(n,m)\in\Omega} I_{c,n,m} \cdot (1 - |j + \beta_c - m|) \cdot \mathrm{Sign}(n - i - \alpha_c)$$

$$\frac{\partial \mathcal{L}}{\partial \beta_c} = \sum_{i}^{w} \sum_{j}^{h} \frac{\partial \mathcal{L}}{\partial O_{c,i,j}} \sum_{(n,m)\in\Omega} I_{c,n,m} \cdot (1 - |i + \alpha_c - n|) \cdot \mathrm{Sign}(m - j - \beta_c)$$

$$\frac{\partial \mathcal{L}}{\partial I_{c,i,j}} = \frac{\partial \mathcal{L}}{\partial O_{c,i-|\alpha_c|_\dagger,j-|\beta_c|_\dagger}}$$

- Apply L1 regularization for sparsity

$$\mathcal{L}_{total} = \sum_{(x,y)} \mathcal{L}(f(x \mid W, \alpha, \beta), y) + \lambda \mathcal{R}(\alpha, \beta)$$
$$\mathcal{R}(\alpha, \beta) = \| \alpha \|_1 + \| \beta \|_1$$



[All You Need is a Few Shifts: Designing Efficient Convolutional Neural Networks for Image Classification]

Results

ShiftNet

Active Shift

Active Shift + Quantization

Networks	MAdds	Params	Top-1
MobileNetV1 0.75x [10]	325M	2.6M	68.4%
MobileNetV2 1.0x[30]	300M	3.4M	72.0%
ShuffleNetV1 1.5x(g=3)[40]	292M	3.4M	69.0%
ShuffleNetV2 1.5x[26]	299M	3.5M	72.6%
IGCV3-D [32]	318M	3.6M	72.2%
CondenseNet(G=C=8)[12]	274M	2.9M	71.0%
ShiftNet-B [37]	371M	1.1M	61.2%
AS-ResNet-w50 [16]	404M	1.96M	69.9%
FE-Net (ours) 1.0x	301M	3.7M	**72.9%**
MobileNetV1 1.0x[10]	569M	4.2M	70.6%
MobileNetV2 1.4x[30]	585M	6.9M	74.7%
ShuffleNetV1 2x[40]	524M	5.4M	70.9%
ShuffleNetV2 2x[26]	591M	7.4M	74.9%
IGCV3-D 1.4x[32]	610M	7.2M	74.55%
CondenseNet(G=C=4)[12]	529M	4.8M	73.8%
PNASNet[22]	588M	5.1M	74.2%
DARTS [23]	595M	4.9M	73.1%
ShiftNet-A [37]	1400M	4.1M	70.1%
AS-ResNet-w68 [16]	729M	3.42M	72.2%
FE-Net (ours) 1.375x	563M	5.9M	**75.0%**

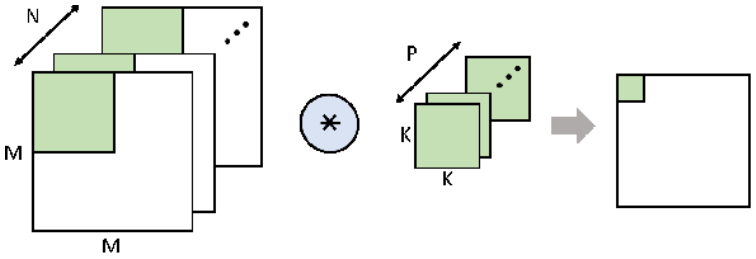
[All You Need is a Few Shifts: Designing Efficient Convolutional Neural Networks for Image Classification]

MobileNet-v1

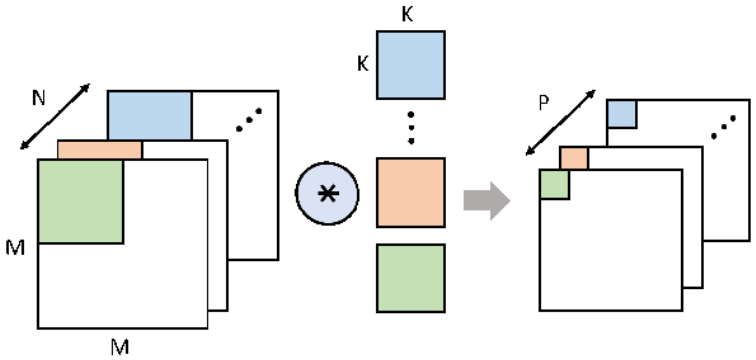
Depthwise-separable Convolution

- Conventional full-convolution covers entire channels across receptive field input
  - Some feature maps are redundant & highly correlated
  - Do we need all channels?

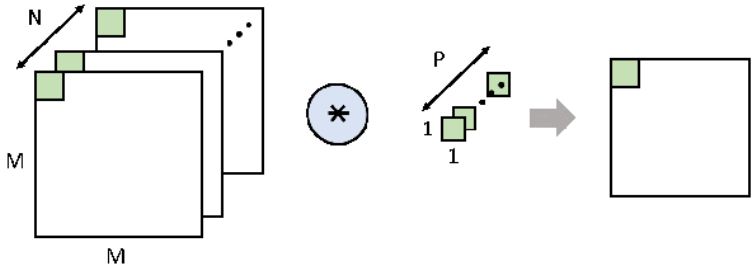
- Decoupling channel-wise feature extractor & spatial feature extractor



(a) standard convolution



(b) depthwise convolution

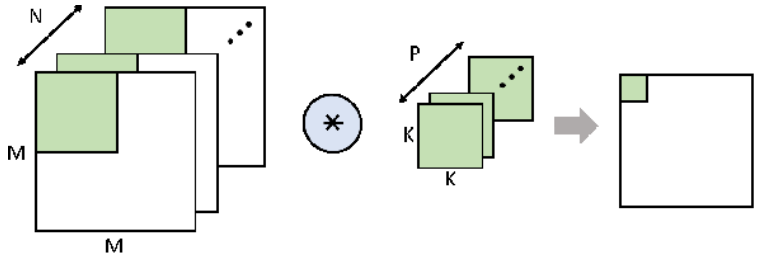


(c) pointwise convolution

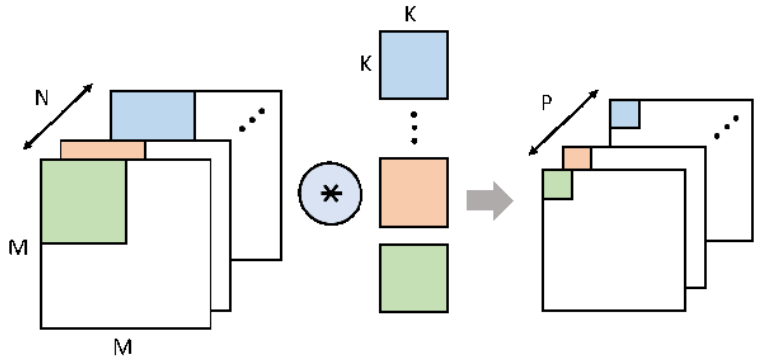
[Bai, 2018]

Depthwise-separable Convolution

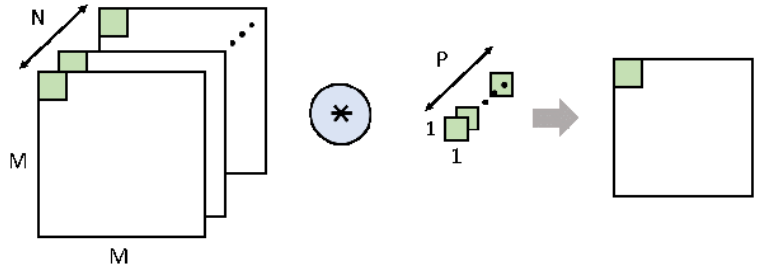
- Standard convolution
  - $Computaion\ cost = C_o C_i H_o W_o K_h K_w$
- Depthwise-separable convolution
  - $Computation\ cost = C_i H_o W_o K_h K_w + C_i C_o H_o W_o$
- Computation gain
  - $= \frac{Cost_{dw}}{Cost_{full}} = \frac{1}{C_i} + \frac{1}{K_h K_w}$



(a) standard convolution



(b) depthwise convolution



(c) pointwise convolution

[Bai, 2018]

MobileNet Structure

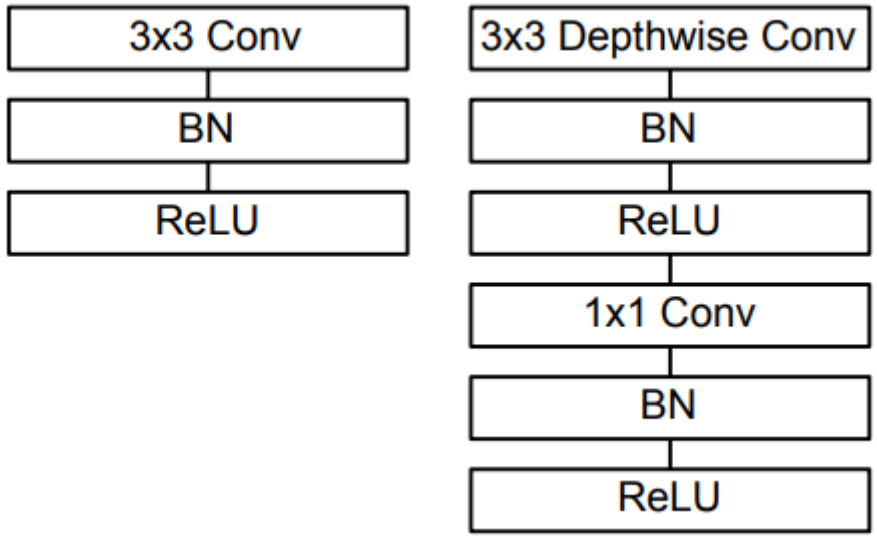


Table 2. Resource Per Layer Type

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%

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	$1 \times 1 \times 1000$

[MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Howard]

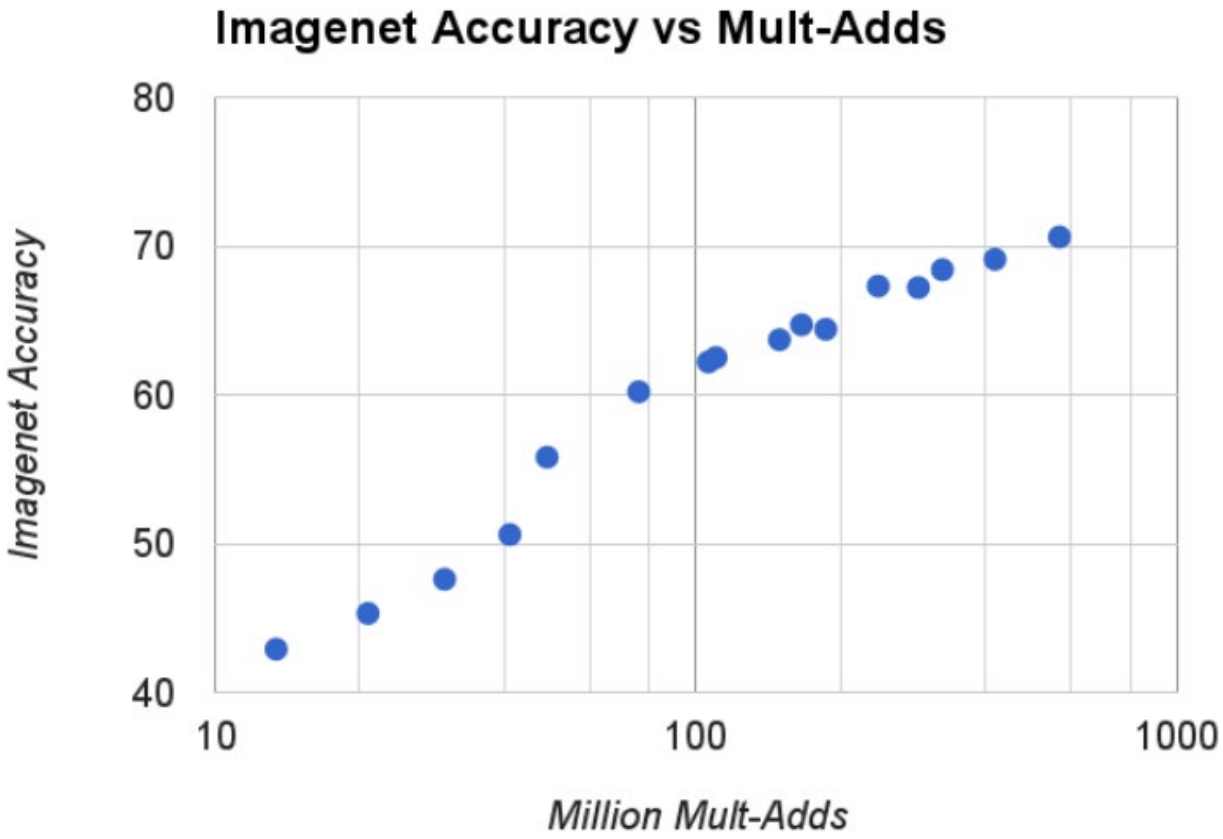
Results

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



[MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Howard]

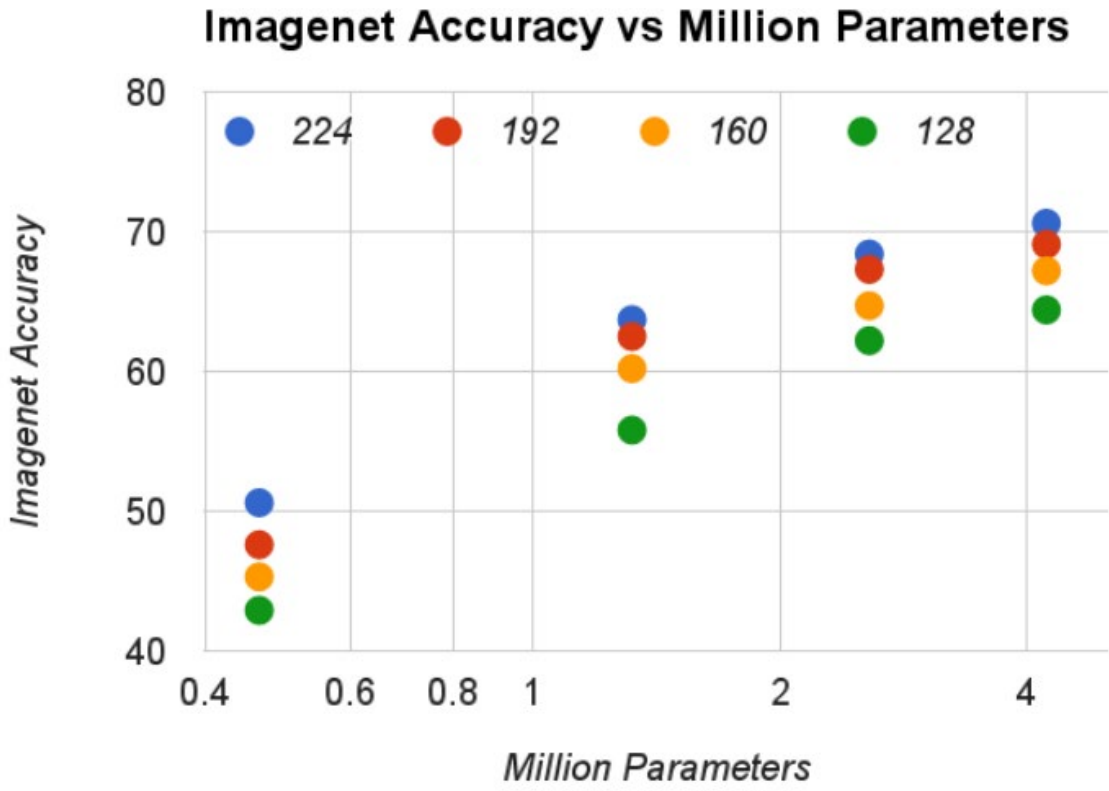
Results

- Width vs Resolution

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

Table 7. MobileNet Resolution

Resolution	ImageNet 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



[MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Howard]

ShuffleNet

1x1 is Still Expensive...

- The 1x1 convolution takes most of the computation

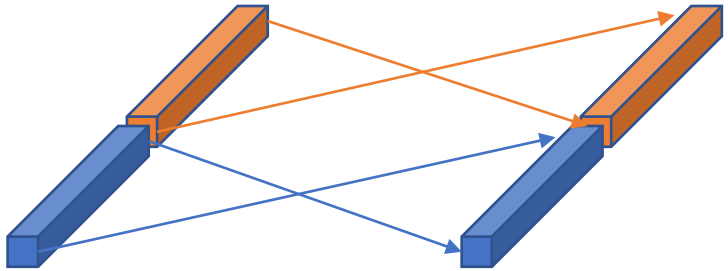
Table 2. Resource Per Layer Type

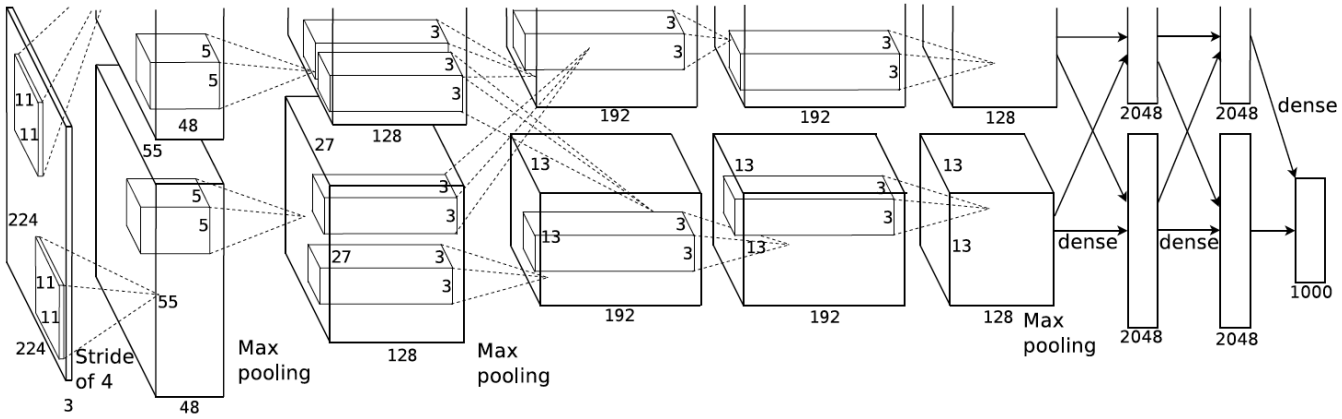
Type	Mult-Adds	Parameters
Conv 1×1	94.86%	74.59%
Conv DW 3×3	3.06%	1.06%
Conv 3×3	1.19%	0.02%
Fully Connected	0.18%	24.33%

- Is it possible to minimize this overhead?

Group Convolution…?

- "G" grouped convolution = G separable convolutions





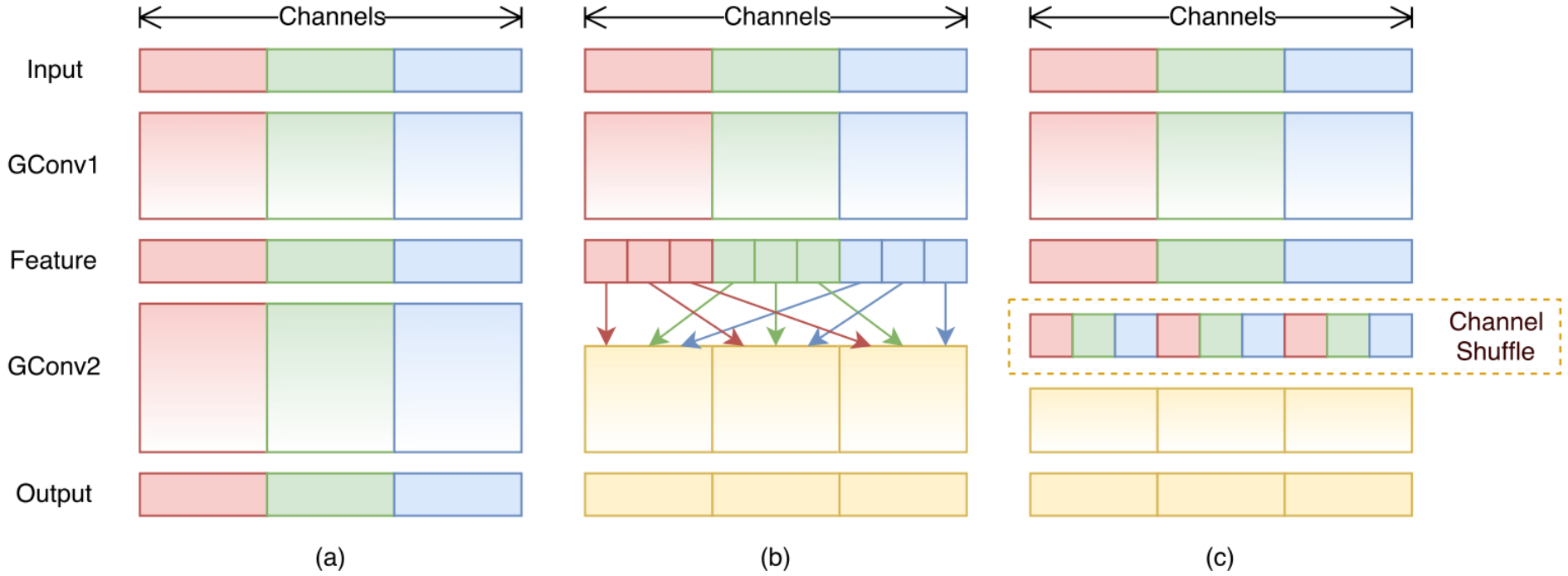
- Initially proposed at AlexNet to train the network with limited GPU memory

- With depthwise-separable convolution?
  - There is no opportunity to exchange information between different channels.

ShuffleNet Idea

```
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
```

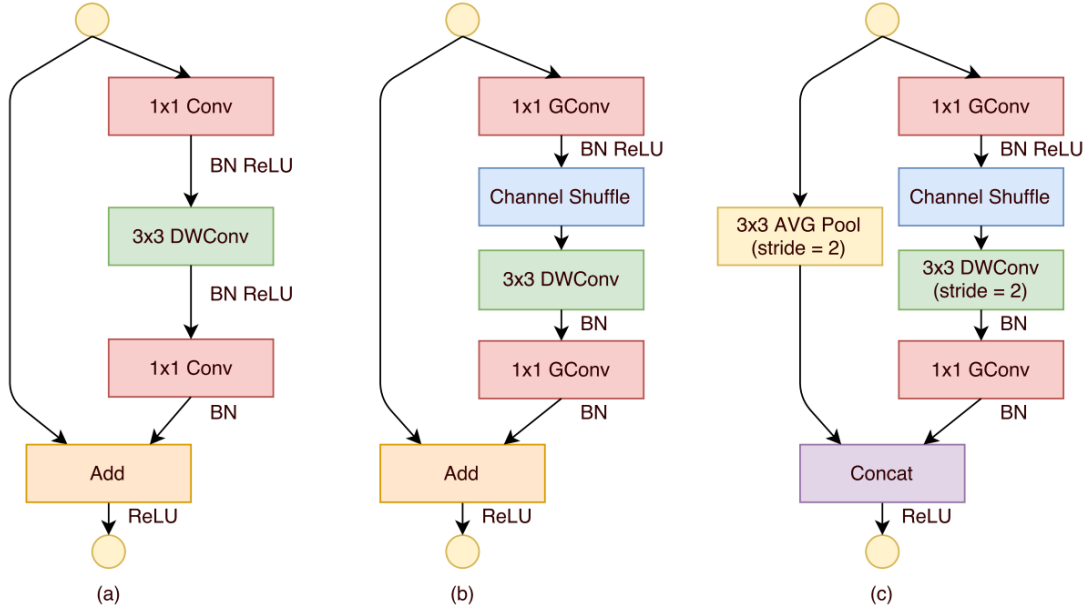
- By applying shuffle information, there is a path to exchange information across all different channels

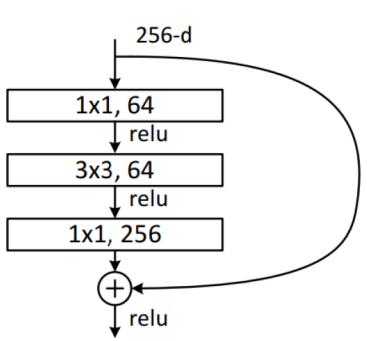


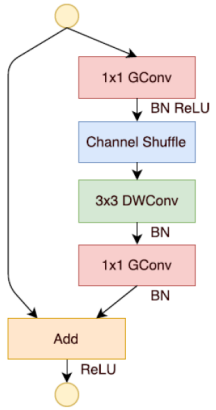
[ShuffleNet: An Extremely Efficient Convolutional Neural network for Mobile Devices, Zhang]

ShuffleNet Design

- First 1x1 conv is replaced by group conv with shuffle
- ReLU is not applied after DW
- Replace element-wise addition with concatenation to enlarge channel







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

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

[Jinwon Lee]

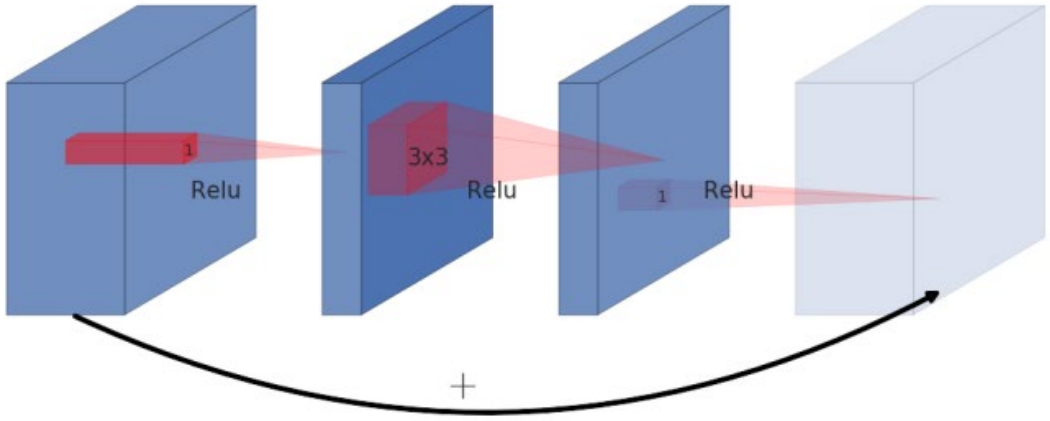
[ShuffleNet: An Extremely Efficient Convolutional Neural network for Mobile Devices, Zhang]

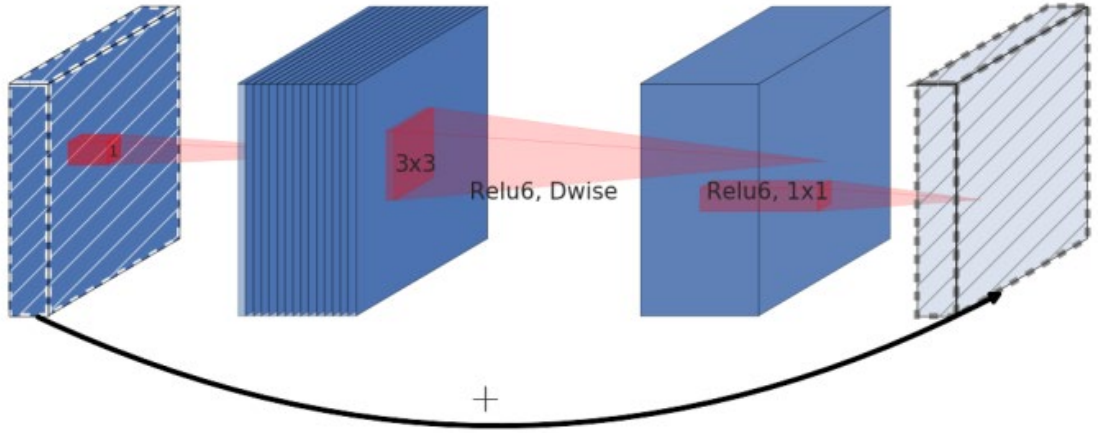
MobileNet-v2

Inverted Residual Module

- Problem of Residual block with DW convolution
  - We should hold the large intermediate activation during inference
  - Spatial feature extractor capacity may not enough

  - What about inverted residual structure?



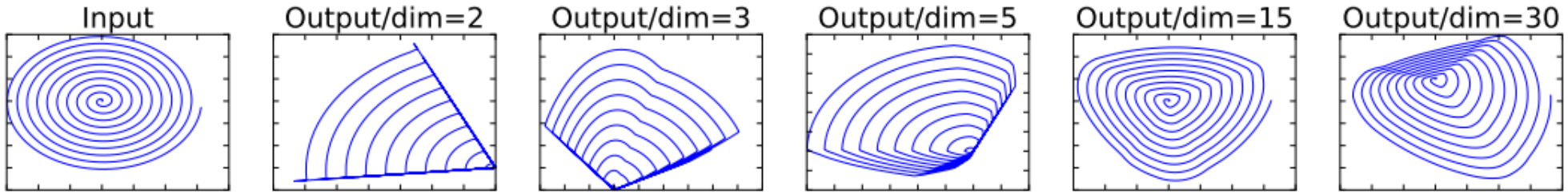


[MobileNetV2: Inverted Residuals and Linear Bottlenecks, Sandler]

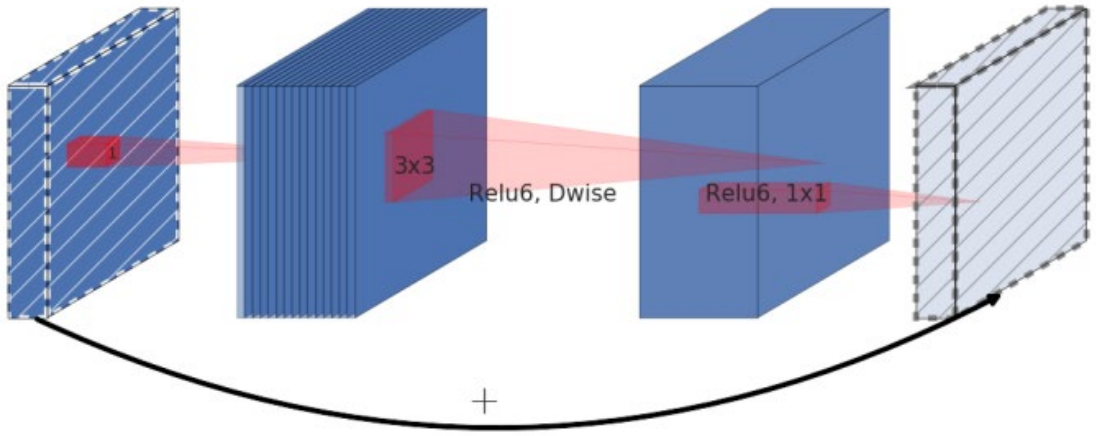
ReLU & information lost

- High-dimension tensor, $h_i \times w_i \times d_i$ feature map can be seen as $h_i \times w_i$ with d_i dimension images
- Manifold of interest, or feature map, is assumed to be embedded into low-dimensional subspace
  - Reduce dimension without information lost

- Not true with non-linear function like ReLU
  - If the manifold of interest remains non-zero volume after ReLU transformation, it corresponds to a linear transformation
  - 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.



Inverted Residual with Linear Bottleneck

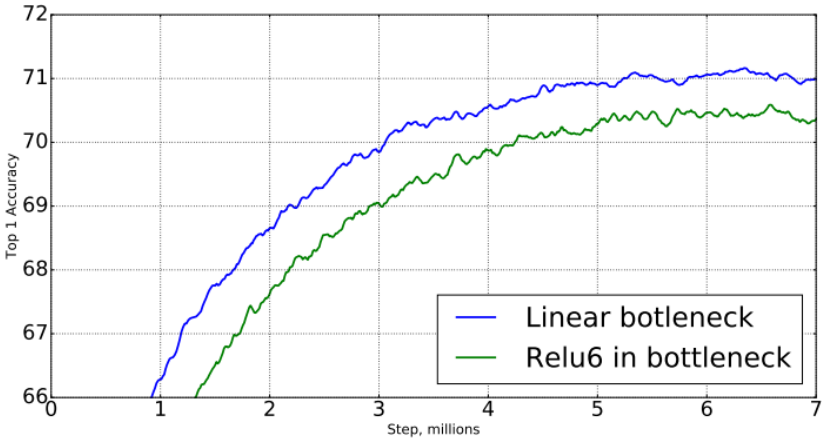



```python
class ConvBNReLU(nn.Sequential):
    def __init__(self, in_planes, out_planes, kernel_size=3, stride=1, groups=1, norm_layer=None):
        padding = (kernel_size - 1) // 2
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        super(ConvBNReLU, self).__init__(
            nn.Conv2d(in_planes, out_planes, kernel_size, stride, padding, groups=groups, bias=False),
            norm_layer(out_planes),
            nn.ReLU6(inplace=True)
        )
```

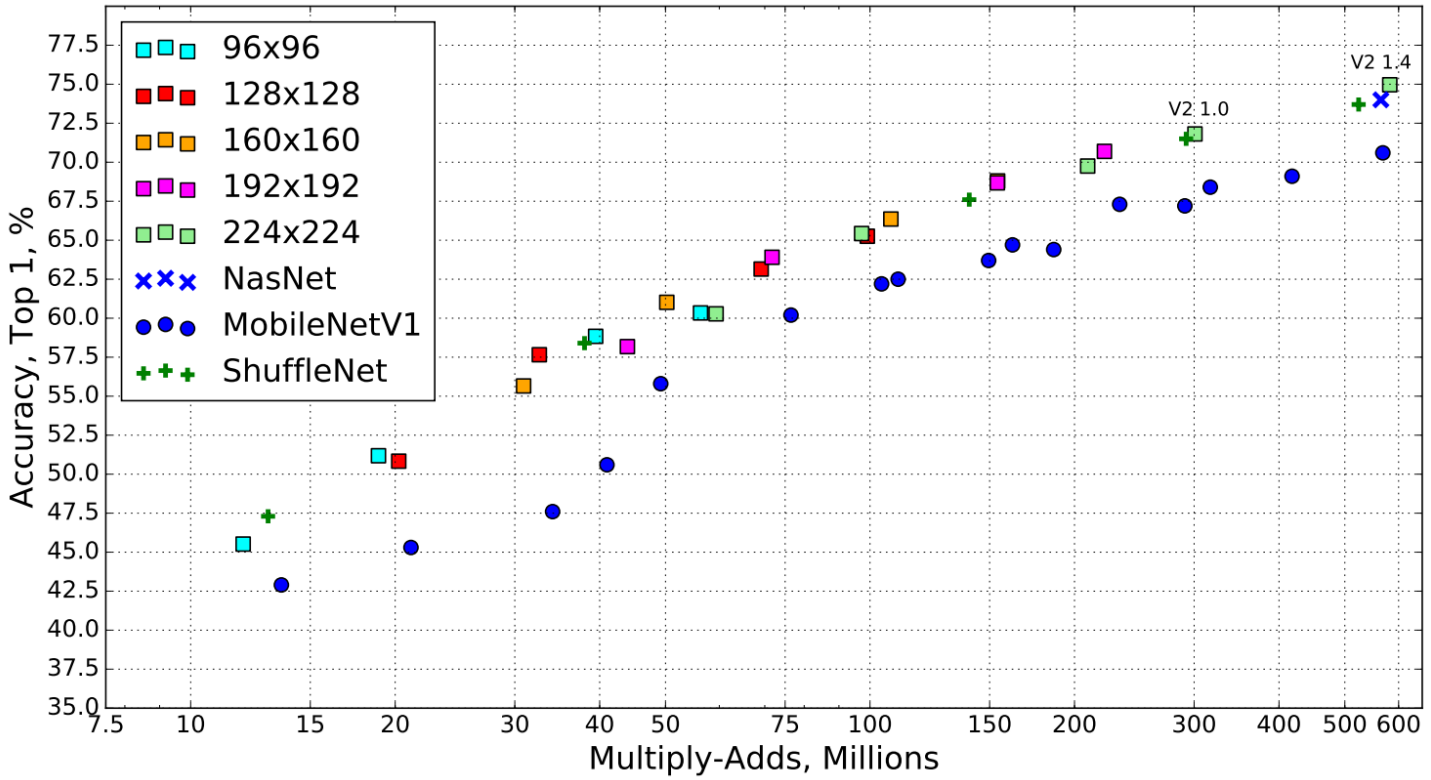
```python
class InvertedResidual(nn.Module):
    def __init__(self, inp, oup, stride, expand_ratio, norm_layer=None):
        super(InvertedResidual, self).__init__()
        self.stride = stride
        assert stride in [1, 2]

        if norm_layer is None:
            norm_layer = nn.BatchNorm2d

        hidden_dim = int(round(inp * expand_ratio))
        self.use_res_connect = self.stride == 1 and inp == oup

        layers = []
        if expand_ratio != 1:
            # pw
            layers.append(ConvBNReLU(inp, hidden_dim, kernel_size=1, norm_layer=norm_layer))
        layers.extend([
            # dw
            ConvBNReLU(hidden_dim, hidden_dim, stride=stride, groups=hidden_dim, norm_layer=norm_layer)
            # pw-linear
            nn.Conv2d(hidden_dim, oup, 1, 1, 0, bias=False),
            norm_layer(oup),
        ])
        self.conv = nn.Sequential(*layers)

    def forward(self, x):
        if self.use_res_connect:
            return x + self.conv(x)
        else:
            return self.conv(x)
```

[MobileNetV2: Inverted Residuals and Linear Bottlenecks, Sandler]

Results



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**



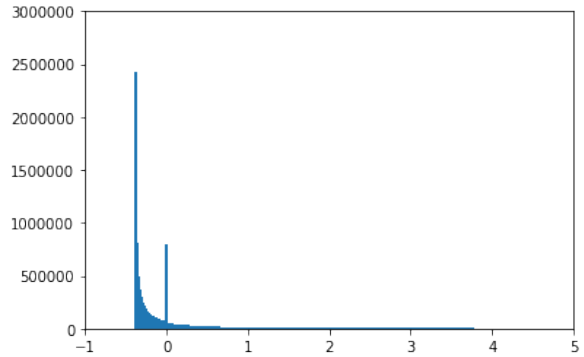
[MobileNetV2: Inverted Residuals and Linear Bottlenecks, Sandler]

MobileNet-v3

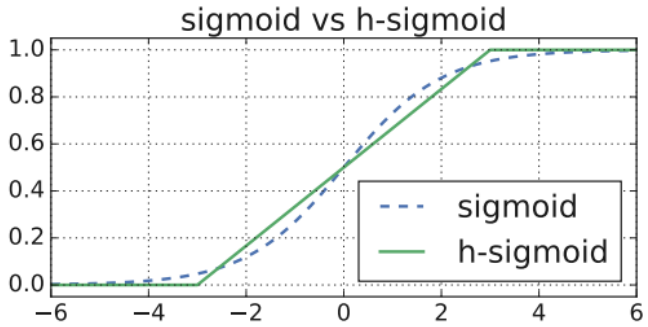
3 Key Component for MV3 Design

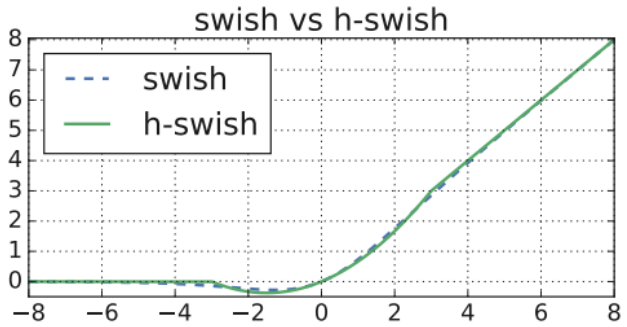
- New non-linear activation, h-swish
- Squeeze-excitation module
- Automated architecture tuning

H-swish function



- VS ReLU: has small range of negative output
- Simplified version of $swish(x) = x \cdot \sigma(x)$
  - $h - swish(x) = x \cdot \frac{ReLU6(x+3)}{6}$



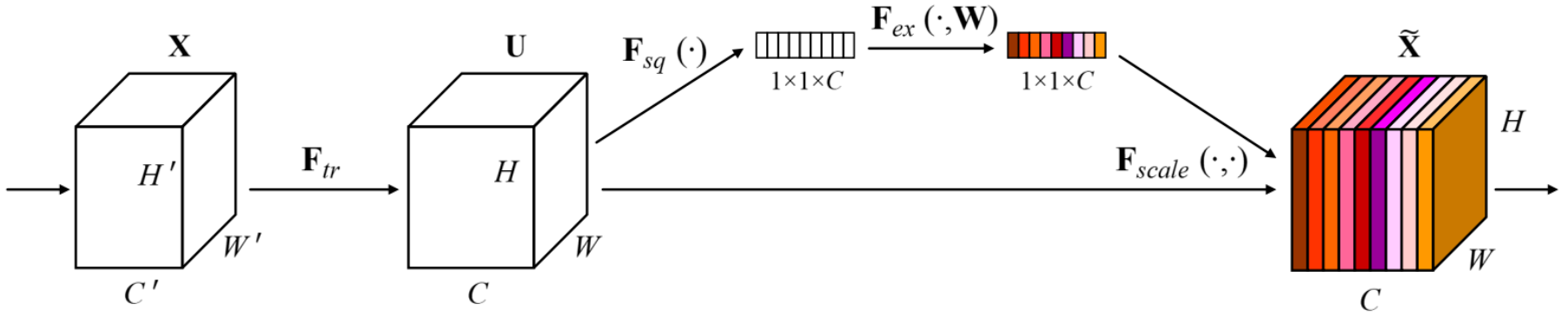


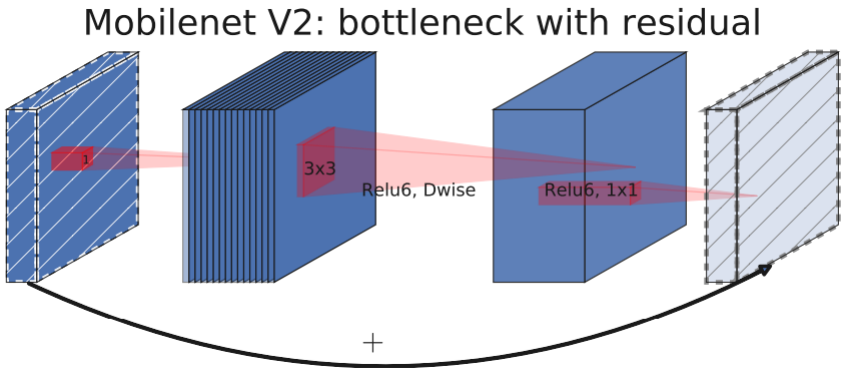
- According to observation, most of the benefits swish are realized only in the deeper layers
  - Use h-swish at the second half of the model

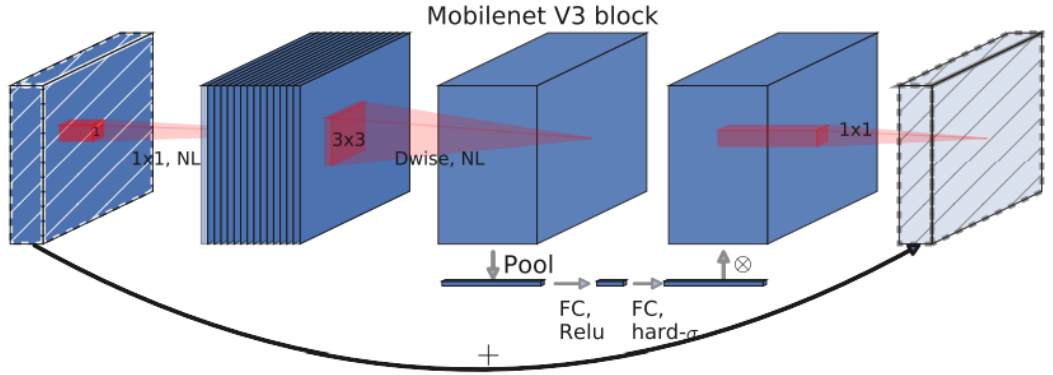
[Searching for MobileNetV3, Howard]

Squeeze-Excitation Module

- Give attention to the feature map with channel-size scaling
- Squeeze-excitation is beneficial for various of networks







[Squeeze-and-Excitation Networks, Hu]

Automated Architecture Tuning

- Important question: how to find out optimal channel for each block?
  - 2-step approach
    - 1st step: Platform-Aware NAS for Block-wise Search
      - Block-level optimization to find out the best global structure
      - Try to find the best model around the target latency
      - Use multi-object reward $ACC(m) \times \left[\frac{LAT(m)}{TAR}\right]^{w}$
    - 2nd step: NetAdapt for Layer-wise Search
      - Layer-wise fine-tuning
- We will cover other examples in the future…

1. Starts with a seed network architecture found by platform-aware NAS.
2. For each step:
   (a) Generate a set of new *proposals*. Each proposal represents a modification of an architecture that generates at least δ reduction in latency compared to the previous step.
   (b) For each proposal we use the pre-trained model from the previous step and populate the new proposed architecture, truncating and randomly initializing missing weights as appropriate. Fine-tune each proposal for T steps to get a coarse estimate of the accuracy.
   (c) Selected best proposal according to some metric.
3. Iterate previous step until target latency is reached.

[Searching for MobileNetV3, Howard]

Results

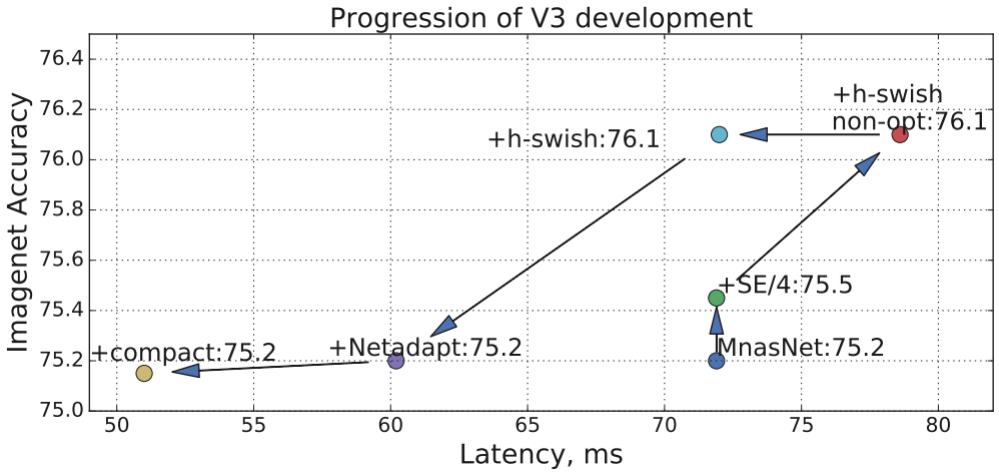
Network	Top-1	MAdds	Params	P-1	P-2	P-3
V3-Large 1.0	**75.2**	**219**	5.4M	**51**	**61**	**44**
V3-Large 0.75	73.3	155	4.0M	39	46	40
MnasNet-A1	75.2	315	3.9M	71	86	61
Proxyless[5]	74.6	320	4.0M	72	84	60
V2 1.0	72.0	300	3.4M	64	76	56
V3-Small 1.0	**67.4**	**56**	2.5M	**15.8**	**19.4**	**14.4**
V3-Small 0.75	65.4	44	2.0M	12.8	15.6	11.7
Mnas-small [43]	64.9	65.1	1.9M	20.3	24.2	17.2
V2 0.35	60.8	59.2	1.6M	16.6	19.6	13.9

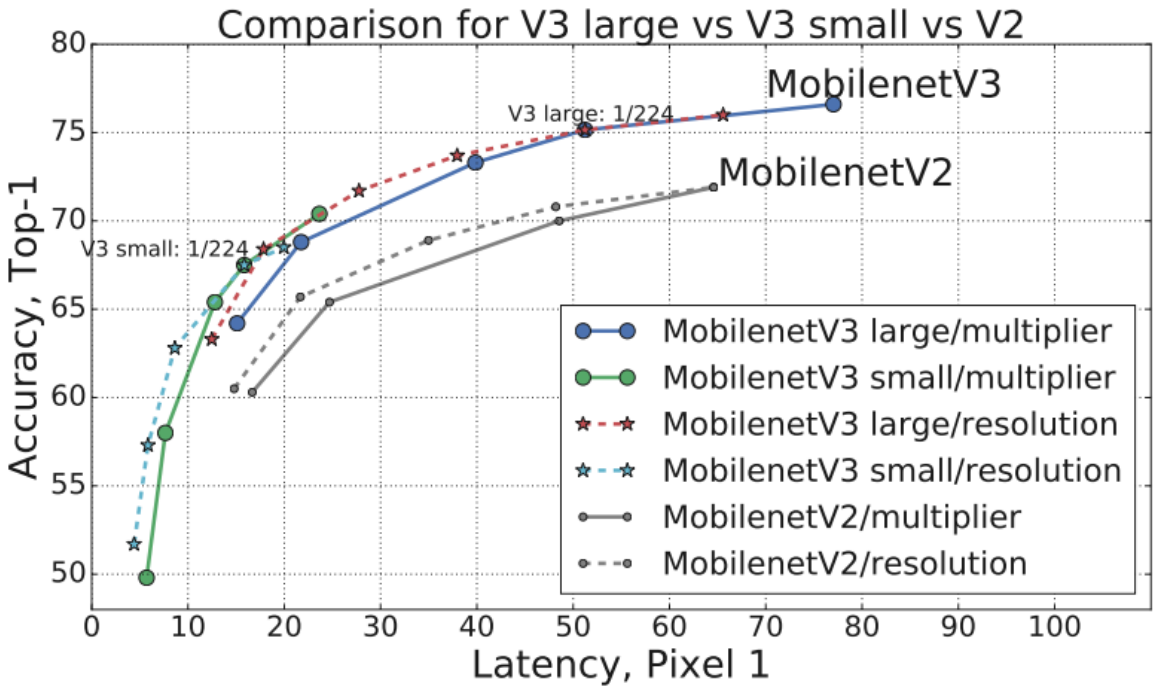
Table 3. Floating point performance on the Pixel family of phones (P-n denotes a Pixel-n phone). All latencies are in ms and are measured using a single large core with a batch size of one. Top-1 accuracy is on ImageNet.

Input	Operator	exp size	$\#out$	SE	NL	s
$224^2 \times 3$	conv2d	-	16	-	HS	2
$112^2 \times 16$	bneck, 3x3	16	16	-	RE	1
$112^2 \times 16$	bneck, 3x3	64	24	-	RE	2
$56^2 \times 24$	bneck, 3x3	72	24	-	RE	1
$56^2 \times 24$	bneck, 5x5	72	40	✓	RE	2
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 5x5	120	40	✓	RE	1
$28^2 \times 40$	bneck, 3x3	240	80	-	HS	2
$14^2 \times 80$	bneck, 3x3	200	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	184	80	-	HS	1
$14^2 \times 80$	bneck, 3x3	480	112	✓	HS	1
$14^2 \times 112$	bneck, 3x3	672	112	✓	HS	1
$14^2 \times 112$	bneck, 5x5	672	160	✓	HS	2
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	bneck, 5x5	960	160	✓	HS	1
$7^2 \times 160$	conv2d, 1x1	-	960	-	HS	1
$7^2 \times 960$	pool, 7x7	-	-	-	-	1
$1^2 \times 960$	conv2d 1x1, NBN	-	1280	-	HS	1
$1^2 \times 1280$	conv2d 1x1, NBN	-	k	-	-	1

[Searching for MobileNetV3, Howard]

Results



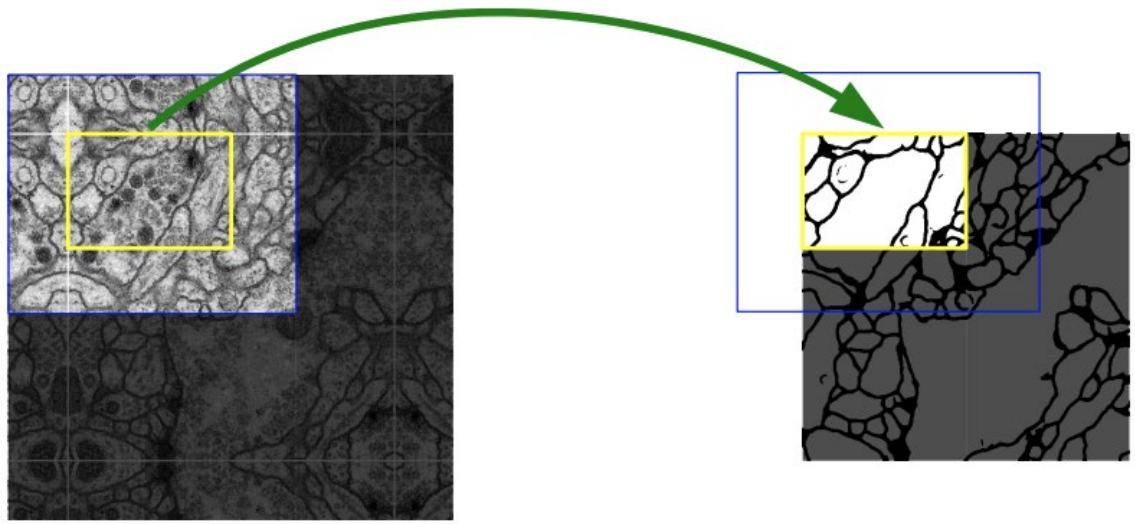


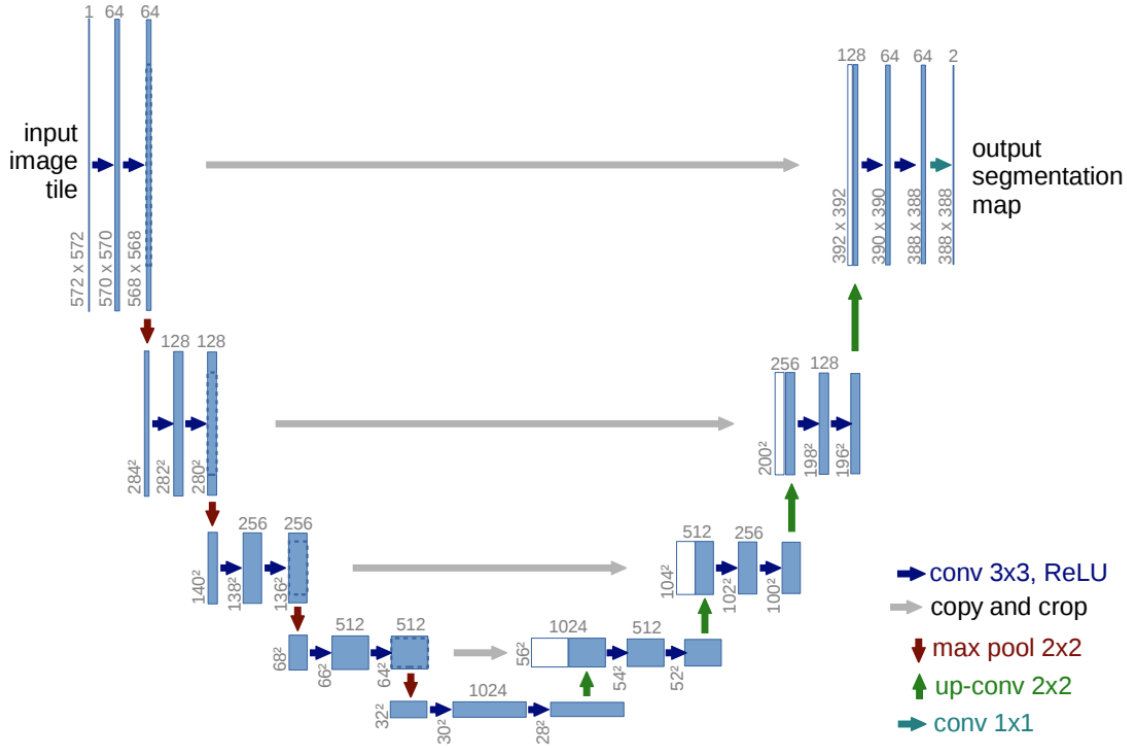
[Searching for MobileNetV3, Howard]

U-Net and Feature Pyramid Network

U-net Architecture for Segmentation

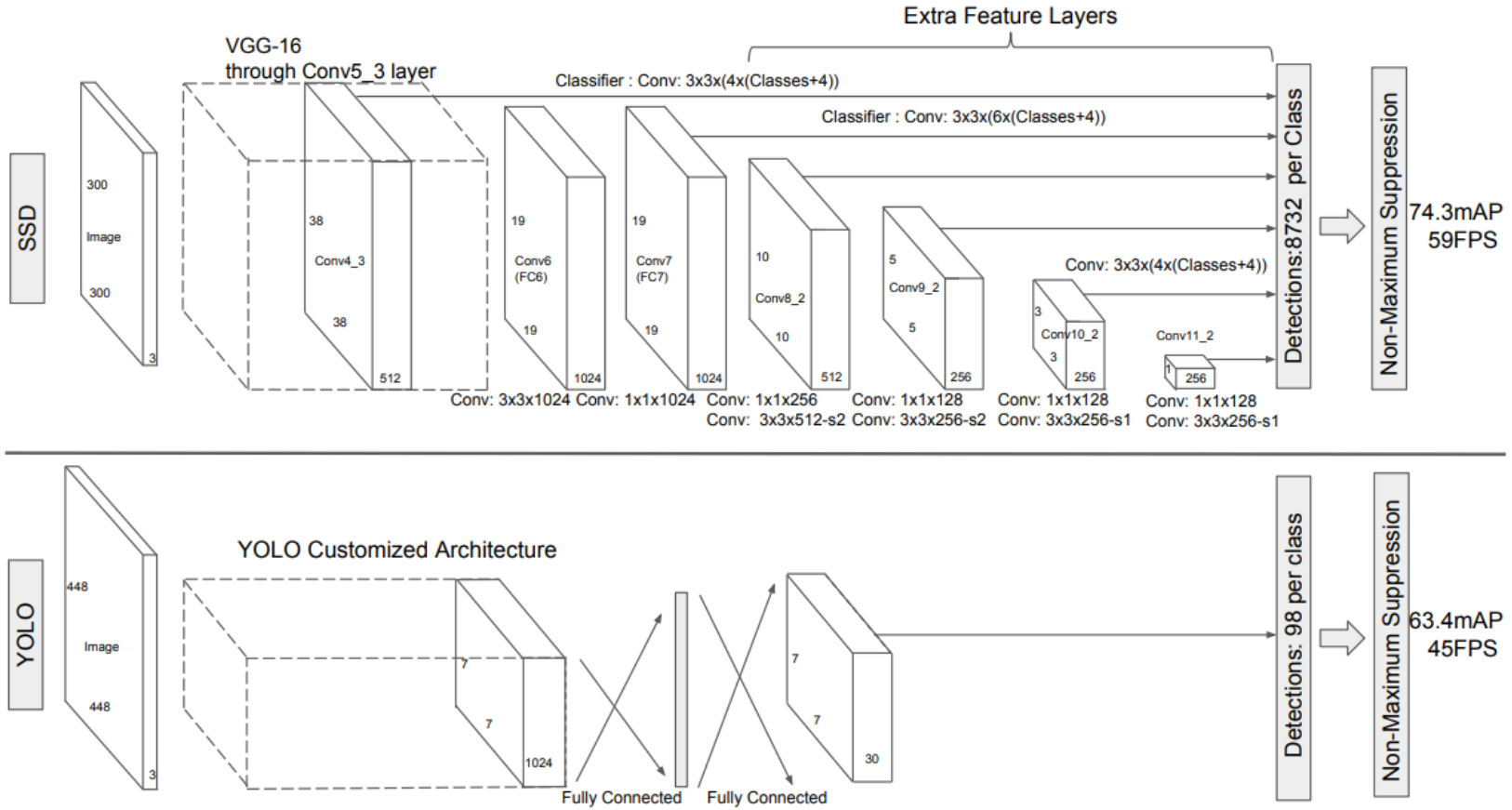
- U-net is initially proposed for biomedical image segmentation
- Prediction of the segmentation in the yellow area, requires image data within the blue area as input



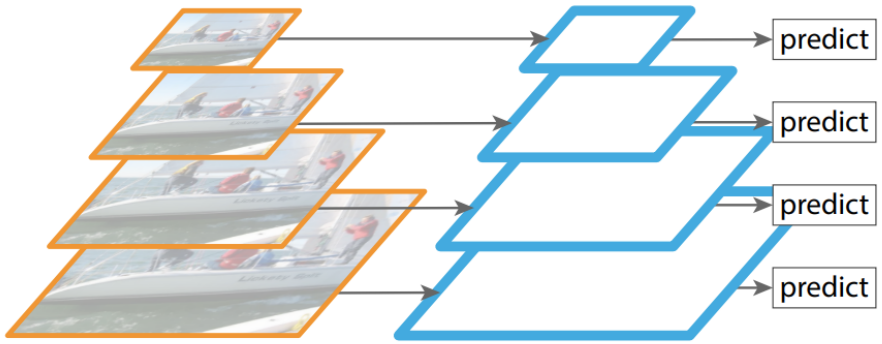


Multi-scale Inference for Object Detection

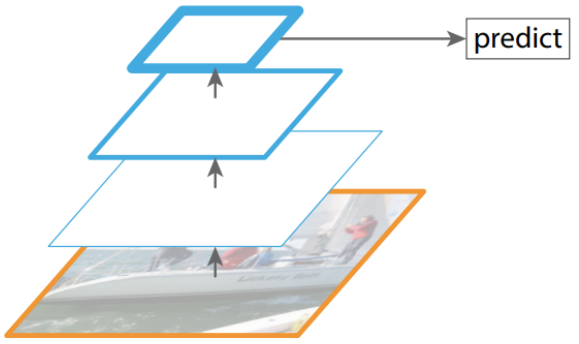
- Multi-scale object detection is crucial to detect the objects having various scales



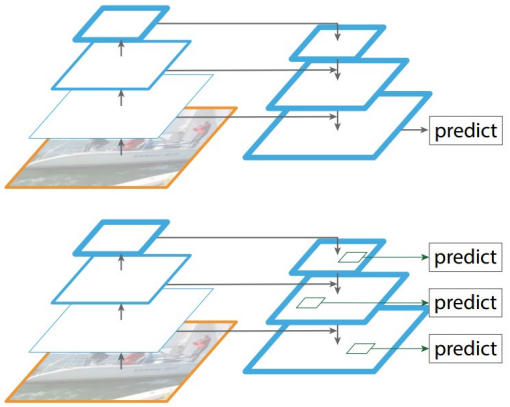
Feature Pyramid Network Structure

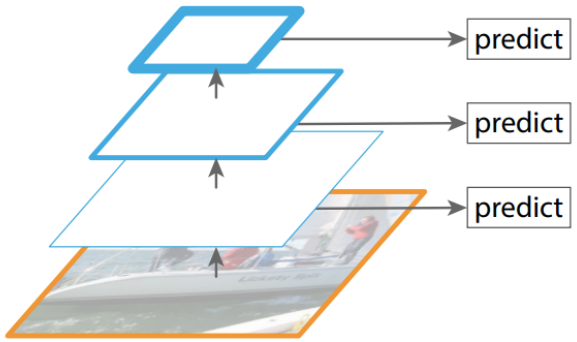


(a) Featurized image pyramid

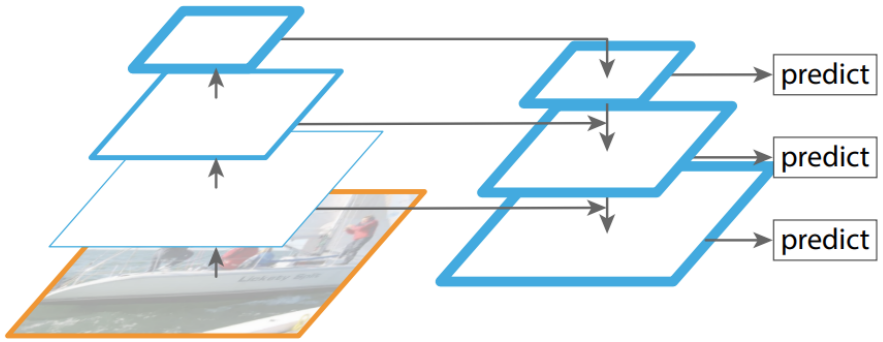


(b) Single feature map

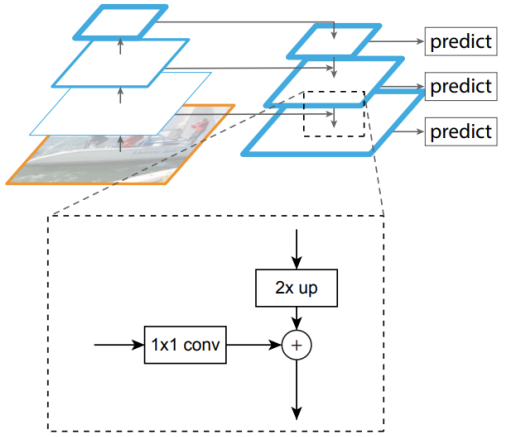




(c) Pyramidal feature hierarchy



(d) Feature Pyramid Network



Results

RPN	feature	# anchors	lateral?	top-down?	AR^{100}	AR^{1k}	AR^{1k}_s	AR^{1k}_m	AR^{1k}_l
(a) baseline on conv4	C_4	47k			36.1	48.3	32.0	58.7	62.2
(b) baseline on conv5	C_5	12k			36.3	44.9	25.3	55.5	64.2
(c) **FPN**	$\{P_k\}$	200k	✓	✓	**44.0**	**56.3**	**44.9**	**63.4**	66.2
Ablation experiments follow:									
(d) bottom-up pyramid	$\{P_k\}$	200k	✓		37.4	49.5	30.5	59.9	**68.0**
(e) top-down pyramid, w/o lateral	$\{P_k\}$	200k		✓	34.5	46.1	26.5	57.4	64.7
(f) only finest level	P_2	750k	✓	✓	38.4	51.3	35.1	59.7	67.6

Table 1. Bounding box proposal results using RPN [29], evaluated on the COCO `minival` set. All models are trained on `trainval35k`. The columns "lateral" and "top-down" denote the presence of lateral and top-down connections, respectively. The column "feature" denotes the feature maps on which the heads are attached. All results are based on ResNet-50 and share the same hyper-parameters.

Fast R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(a) baseline on conv4	RPN, $\{P_k\}$	C_4	conv5			54.7	31.9	15.7	36.5	45.5
(b) baseline on conv5	RPN, $\{P_k\}$	C_5	2*fc*			52.9	28.8	11.9	32.4	43.4
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	**45.8**
Ablation experiments follow:										
(d) bottom-up pyramid	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓		44.9	24.9	10.9	24.4	38.5
(e) top-down pyramid, w/o lateral	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*		✓	54.0	31.3	13.3	35.2	45.3
(f) only finest level	RPN, $\{P_k\}$	P_2	2*fc*	✓	✓	56.3	33.4	17.3	37.3	45.6

Table 2. Object detection results using **Fast R-CNN** [11] on *a fixed set of proposals* (RPN, $\{P_k\}$, Table 1(c)), evaluated on the COCO `minival` set. Models are trained on the `trainval35k` set. All results are based on ResNet-50 and share the same hyper-parameters.

Faster R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP_s	AP_m	AP_l
(*) baseline from He *et al.* [16]†	RPN, C_4	C_4	conv5			47.3	26.3	-	-	-
(a) baseline on conv4	RPN, C_4	C_4	conv5			53.1	31.6	13.2	35.6	**47.1**
(b) baseline on conv5	RPN, C_5	C_5	2*fc*			51.7	28.0	9.6	31.9	43.1
(c) **FPN**	RPN, $\{P_k\}$	$\{P_k\}$	2*fc*	✓	✓	**56.9**	**33.9**	**17.8**	**37.7**	45.8

Table 3. Object detection results using **Faster R-CNN** [29] evaluated on the COCO `minival` set. *The backbone network for RPN are consistent with Fast R-CNN.* Models are trained on the `trainval35k` set and use ResNet-50. †Provided by authors of [16].

Reading Assignment

- ShuffleNet v2 [https://arxiv.org/abs/1807.11164]
- EfficientNet [https://arxiv.org/abs/1905.11946]
- EfficientNet v2 [https://arxiv.org/abs/2104.00298]