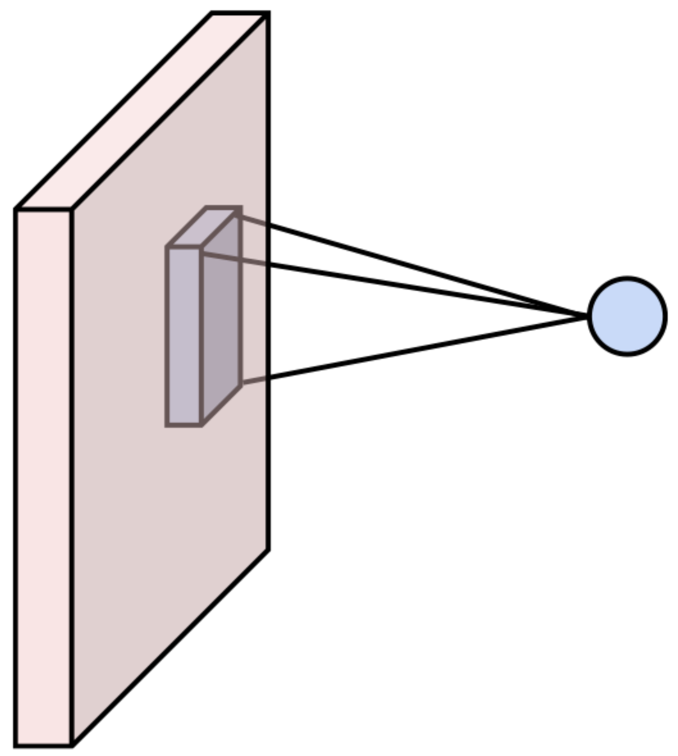
DeepRob

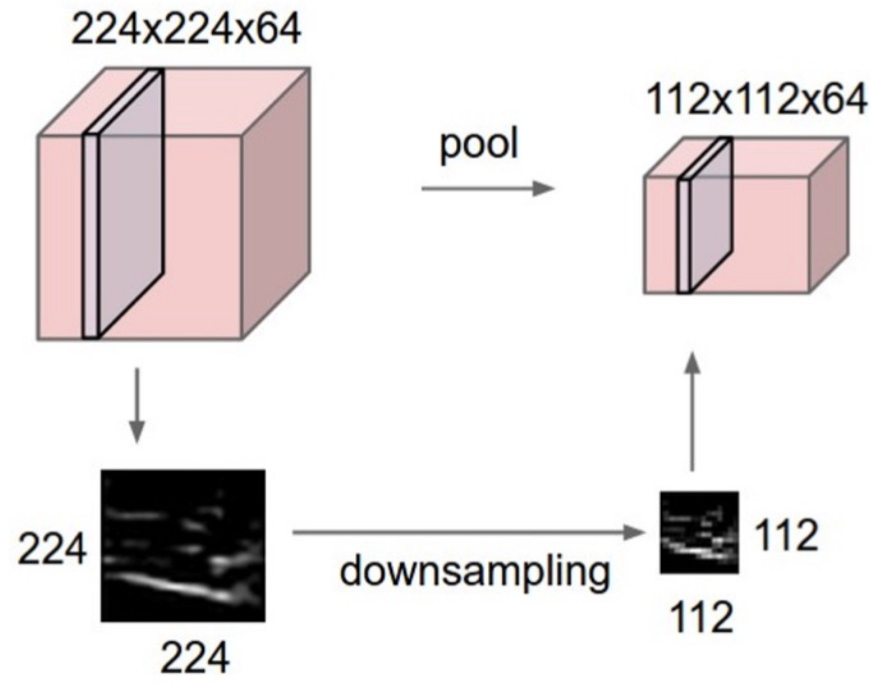
Lecture 8
CNN Architectures and Object Detection
University of Michigan | Department of Robotics

Recap: Convolutional Neural Networks

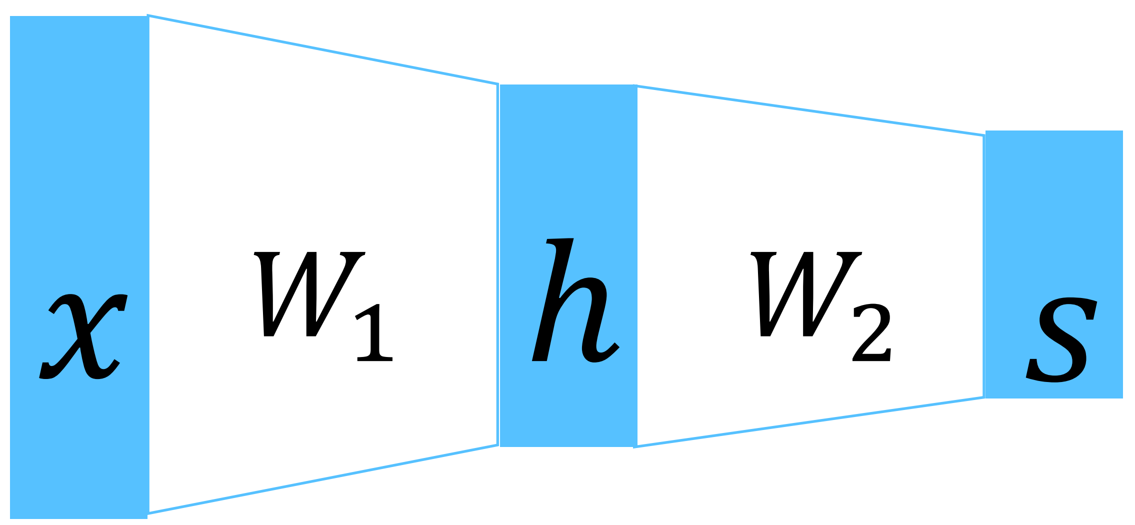
Convolution Layers



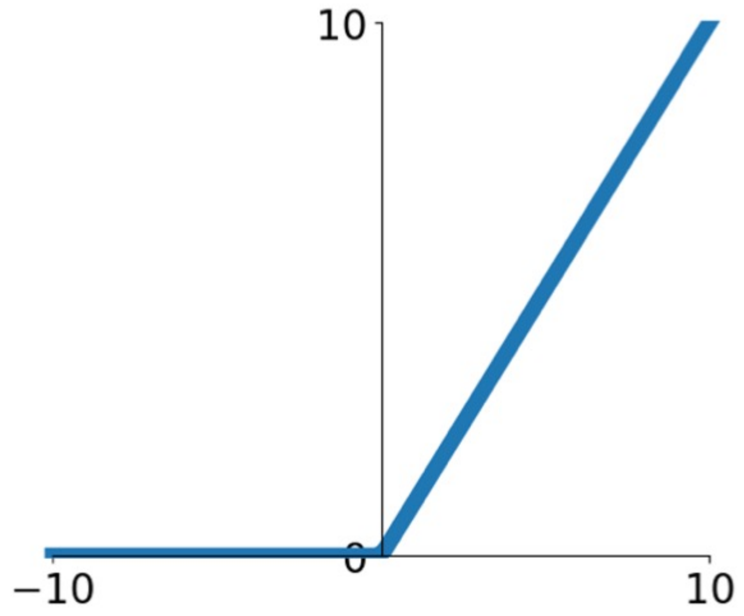
Pooling Layers



Fully-Connected Layers



Activation Function



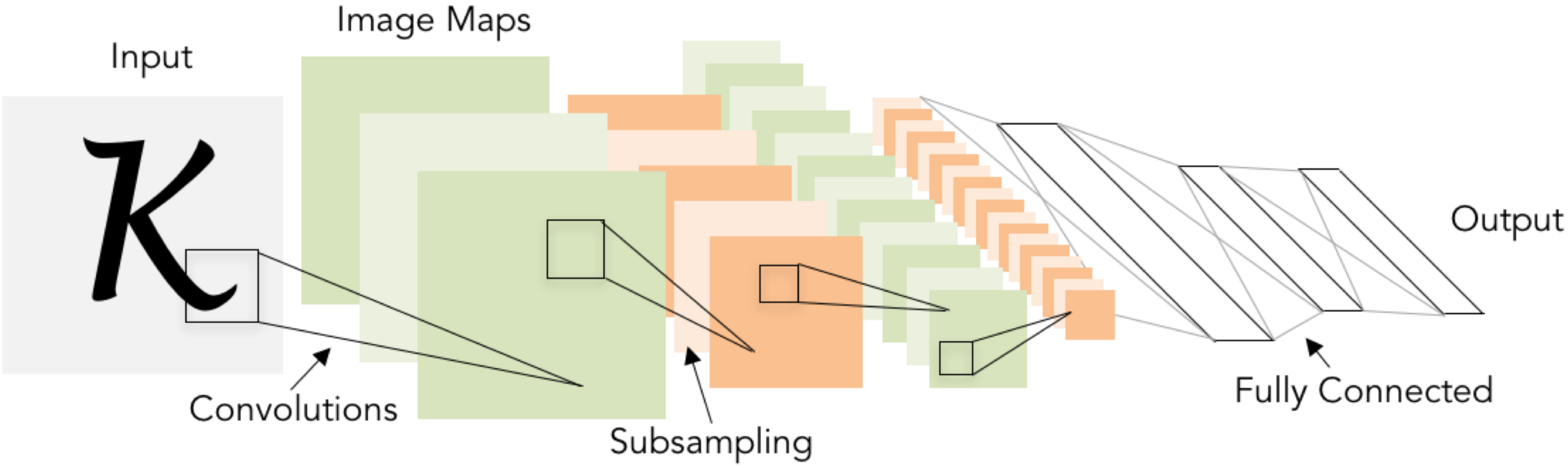
Normalization

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Recap: Classic CNN Architectures

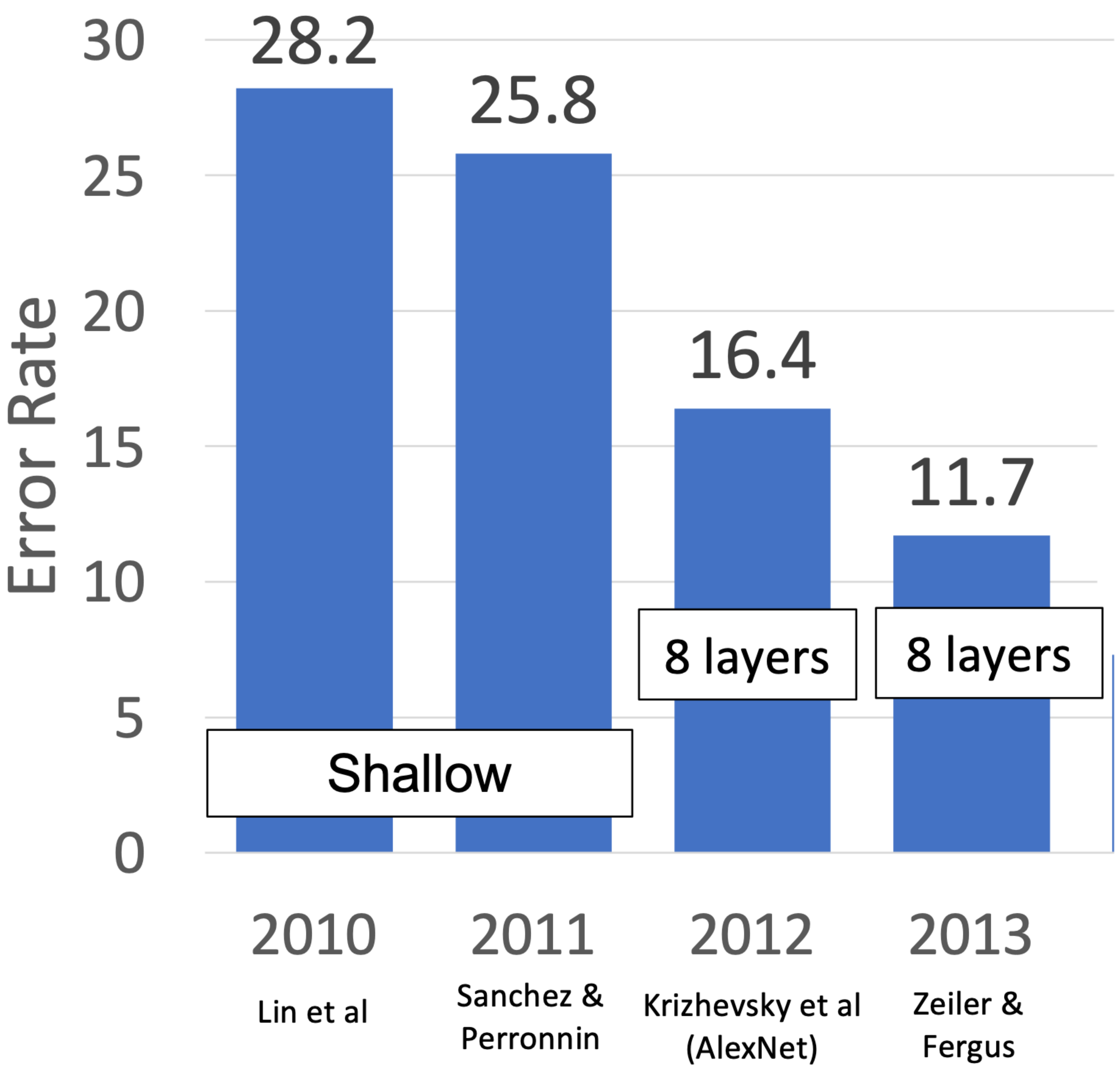
Classic architecture: [Conv, ReLU, Pool] x N, flatten, [FC, ReLU] x N, FC

Example: LeNet-5

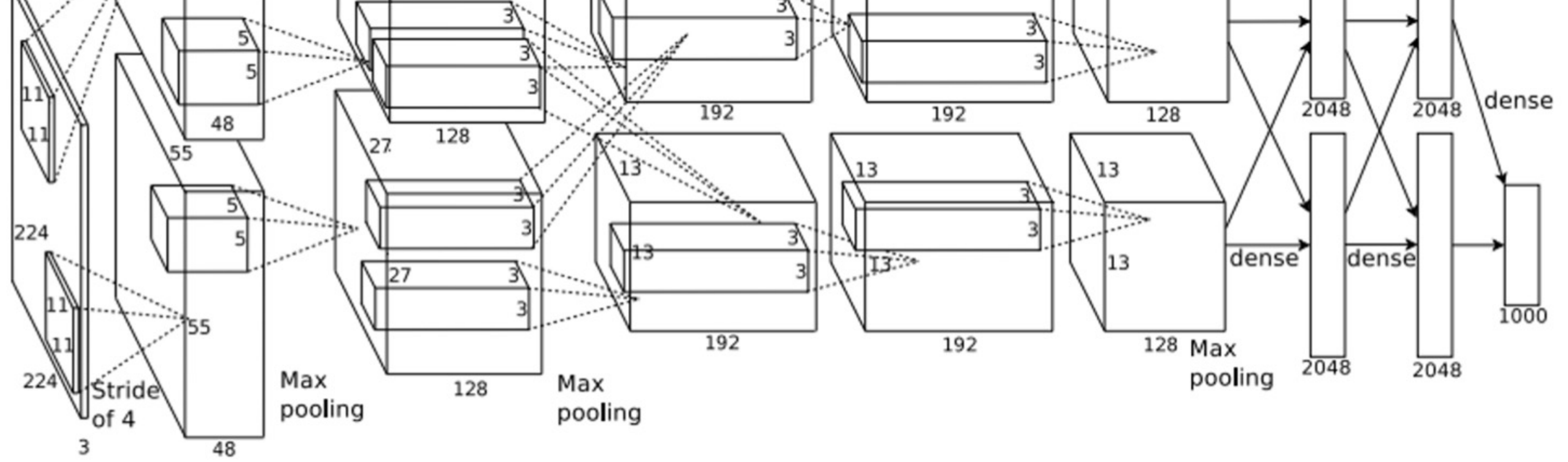


Lecun et al., “Gradient-based learning applied to document recognition”, 1998

Recap: ImageNet Classification Challenge



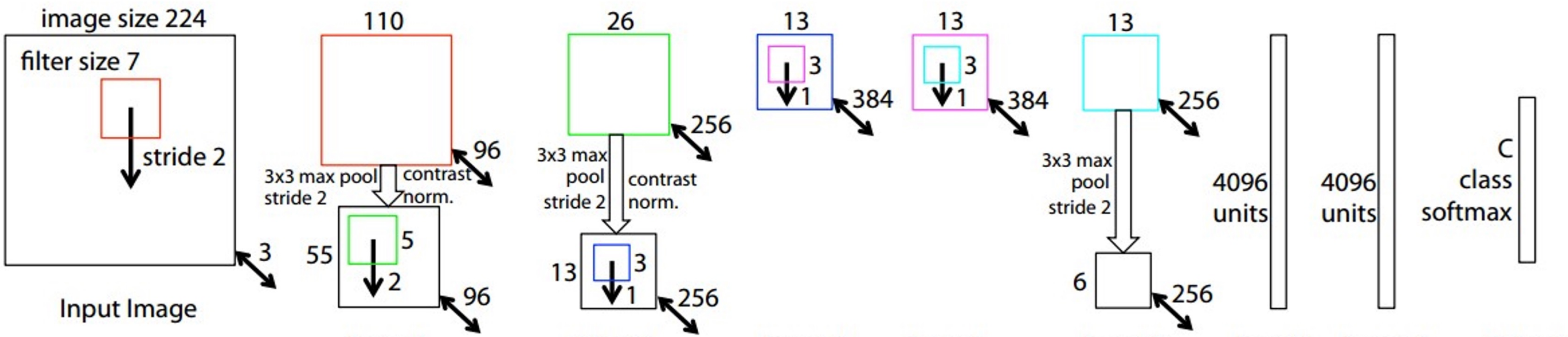
Example: AlexNet



	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params (k)	Flop (M)
Conv1	3	227	64	11	4	2	64	56	784	23	73
Pool1	64	56		3	2	0	64	27	182	0	0
Conv2	64	27	192	5	1	2	192	27	547	307	224
Pool2	192	27		3	2	0	192	13	127	0	0
Conv3	192	13	384	3	1	1	384	13	254	664	112
Conv4	384	13	256	3	1	1	256	13	169	885	145
Conv5	256	13	256	3	1	1	256	13	169	590	100
Pool5	256	13		3	2	0	256	6	36	0	0
Flatten	256	6					9216		36	0	0
FC6	9216		4096				4096		16	37749	38
FC7	4096		4096				4096		16	16777	17
FC8	4096		1000				1000		4	4096	4

Example: ZFNet (a larger AlexNet)

ImageNet top 5 error: 16.4% -> 11.7%



AlexNet but:

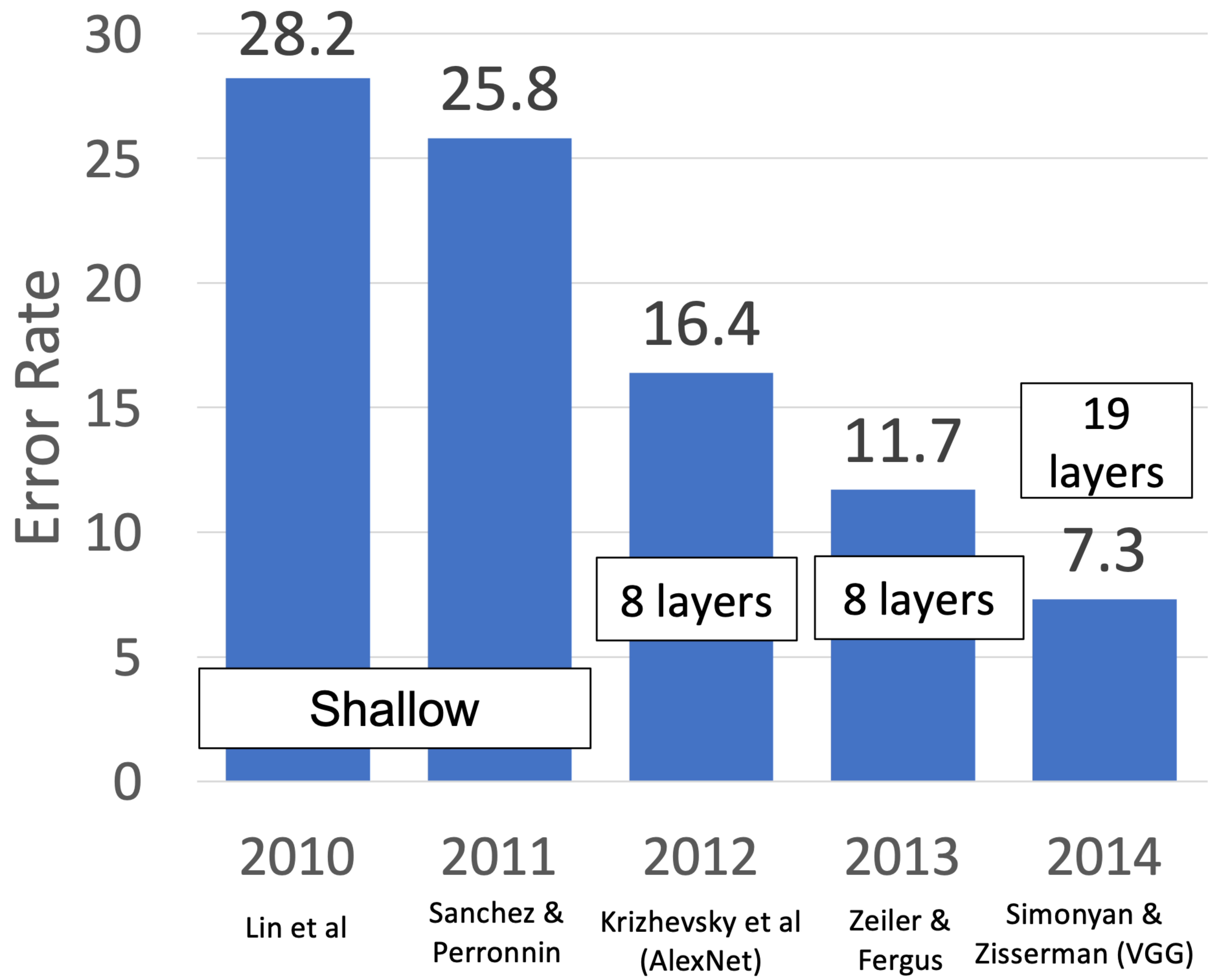
Conv1: change from (11x11 stride 4) to (7x7 stride 2)

Conv3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

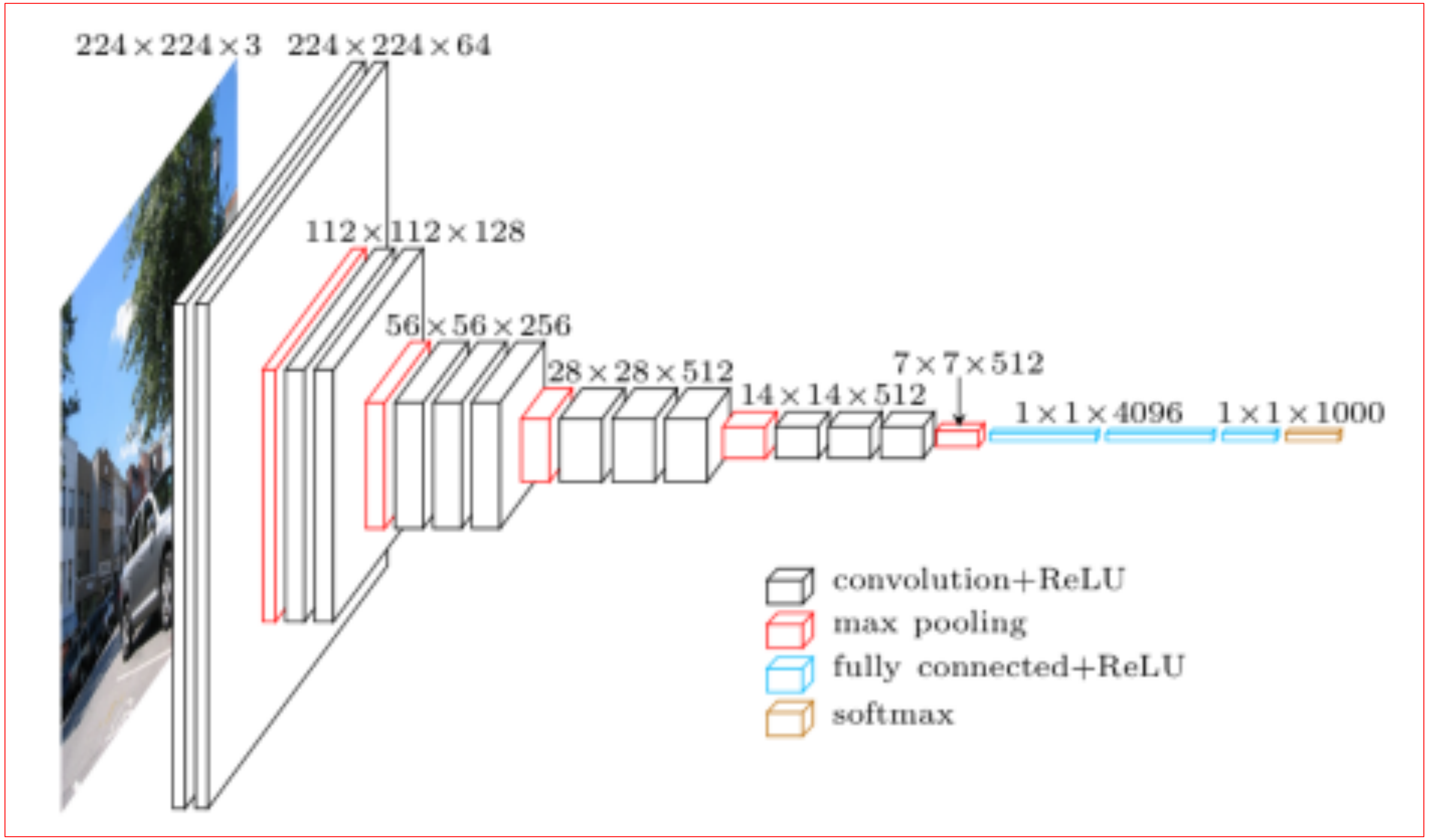
More trial and error :(

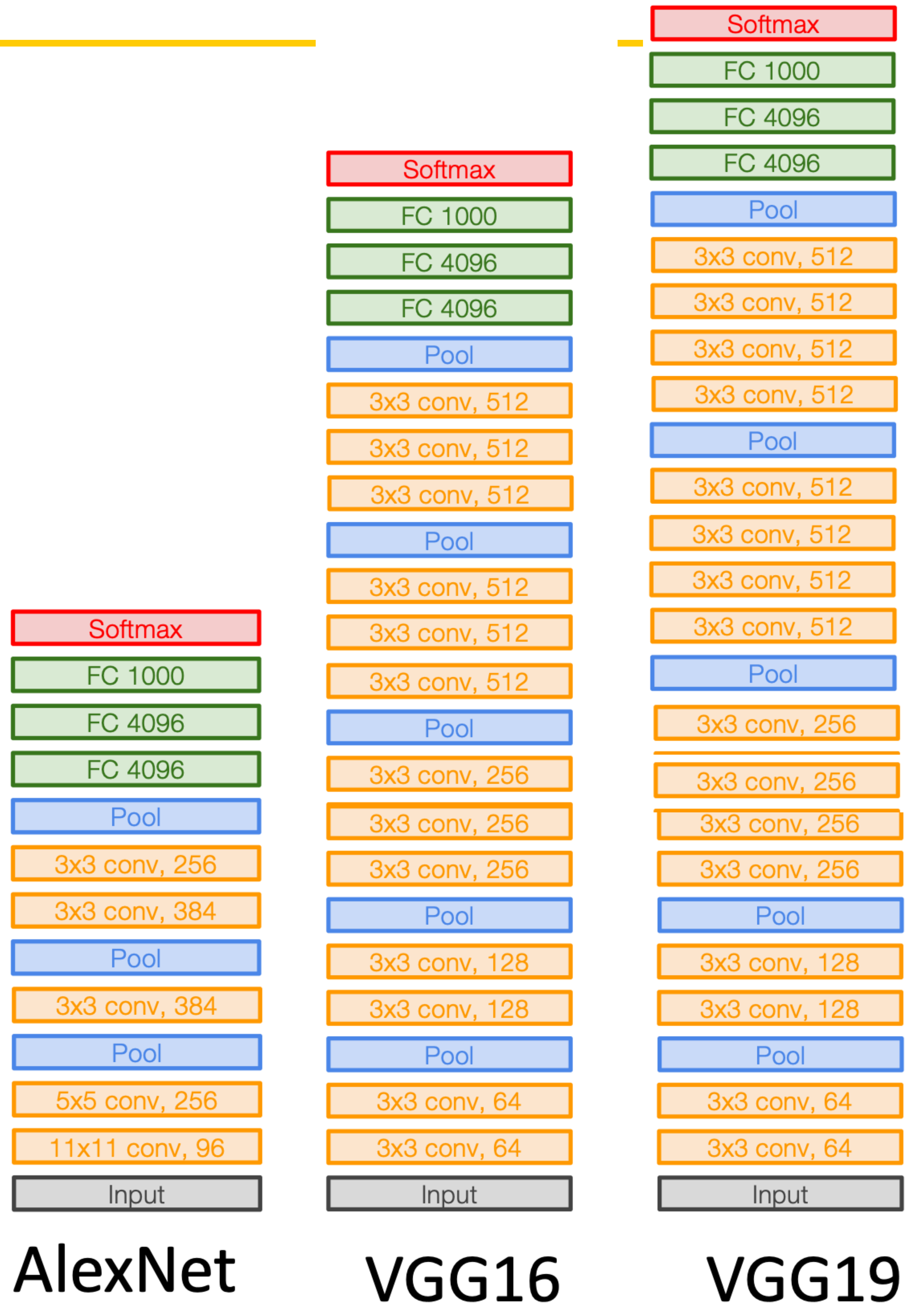
Zeiler and Fergus, "Visualizing and Understanding Convolutional Networks", ECCV 2014

ImageNet Classification Challenge



VGG: Deeper Networks, Regular Design





Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

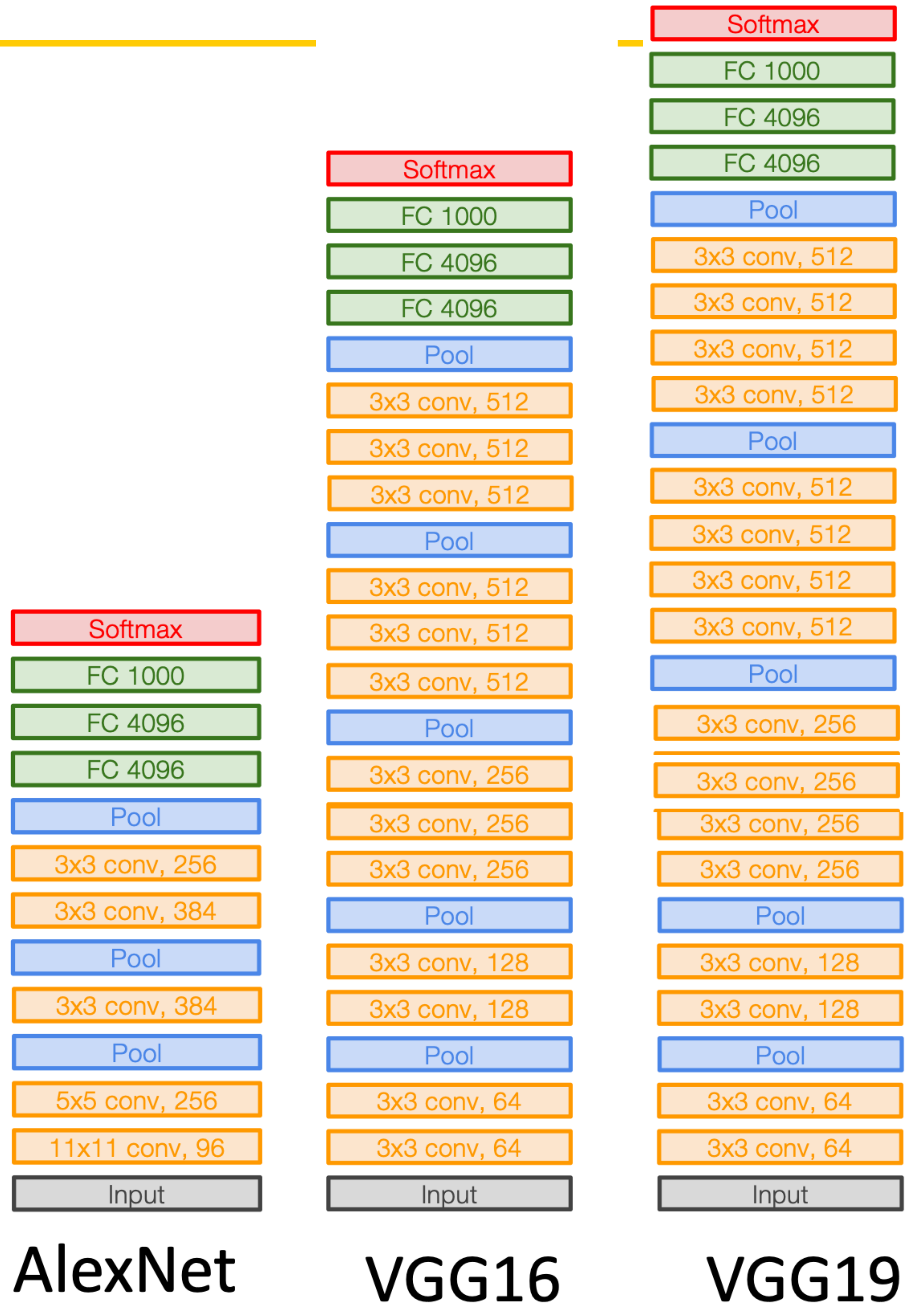
VGG: Deeper Networks, Regular Design

<u>VGG Design rules:</u>

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

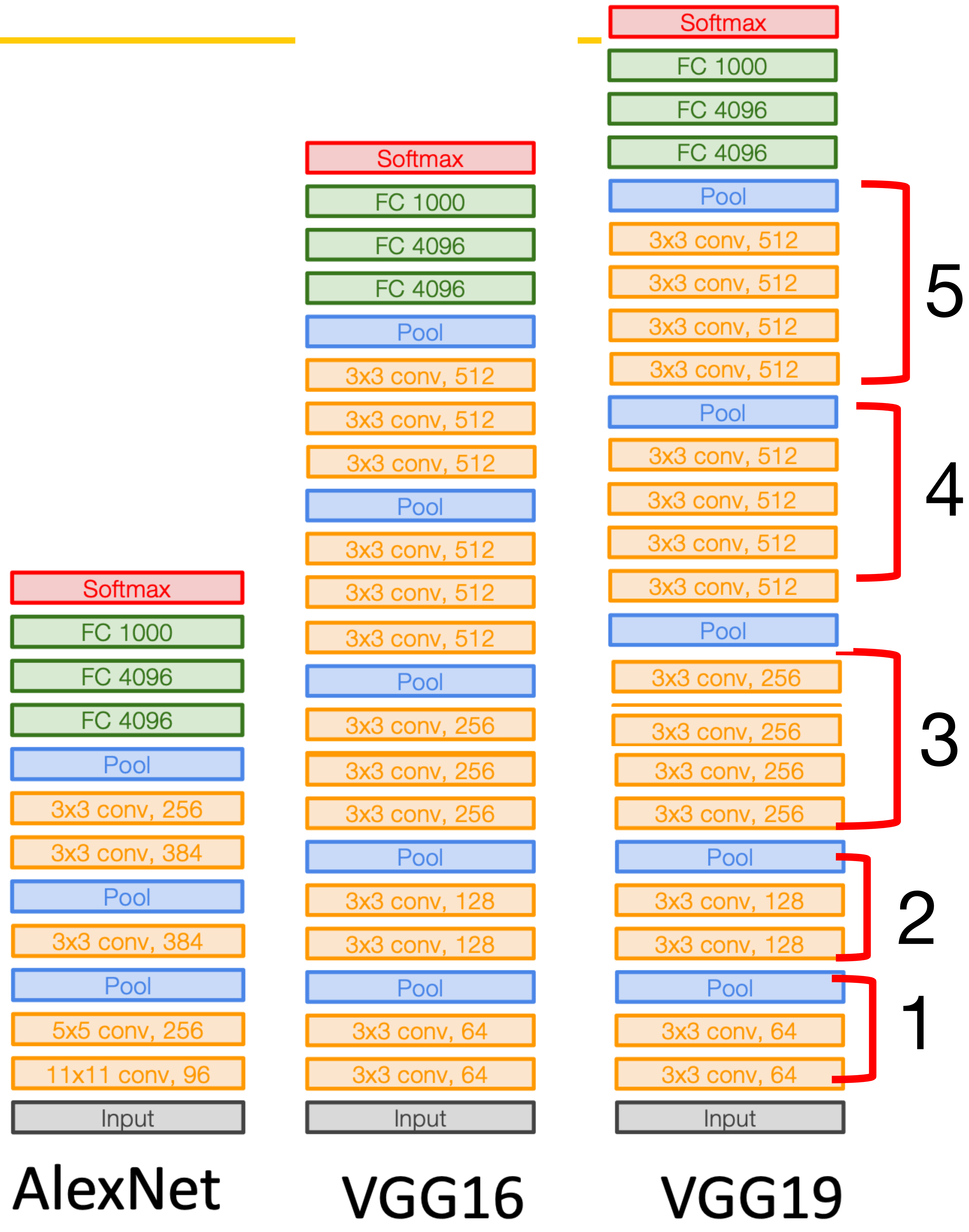


Simonyan and Zissermann, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, ICLR 2015

VGG: Deeper Networks, Regular Design

Network has 5 convolution **stages**:
Stage 1: conv-conv-pool
Stage 2: conv-conv-pool
Stage 3: conv-conv-conv-[conv]-pool
Stage 4: conv-conv-conv-[conv]-pool
Stage 5: conv-conv-conv-[conv]-pool

There are other variations, see Simonyan and Zissermann paper



Simonyan and Zissermann, "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

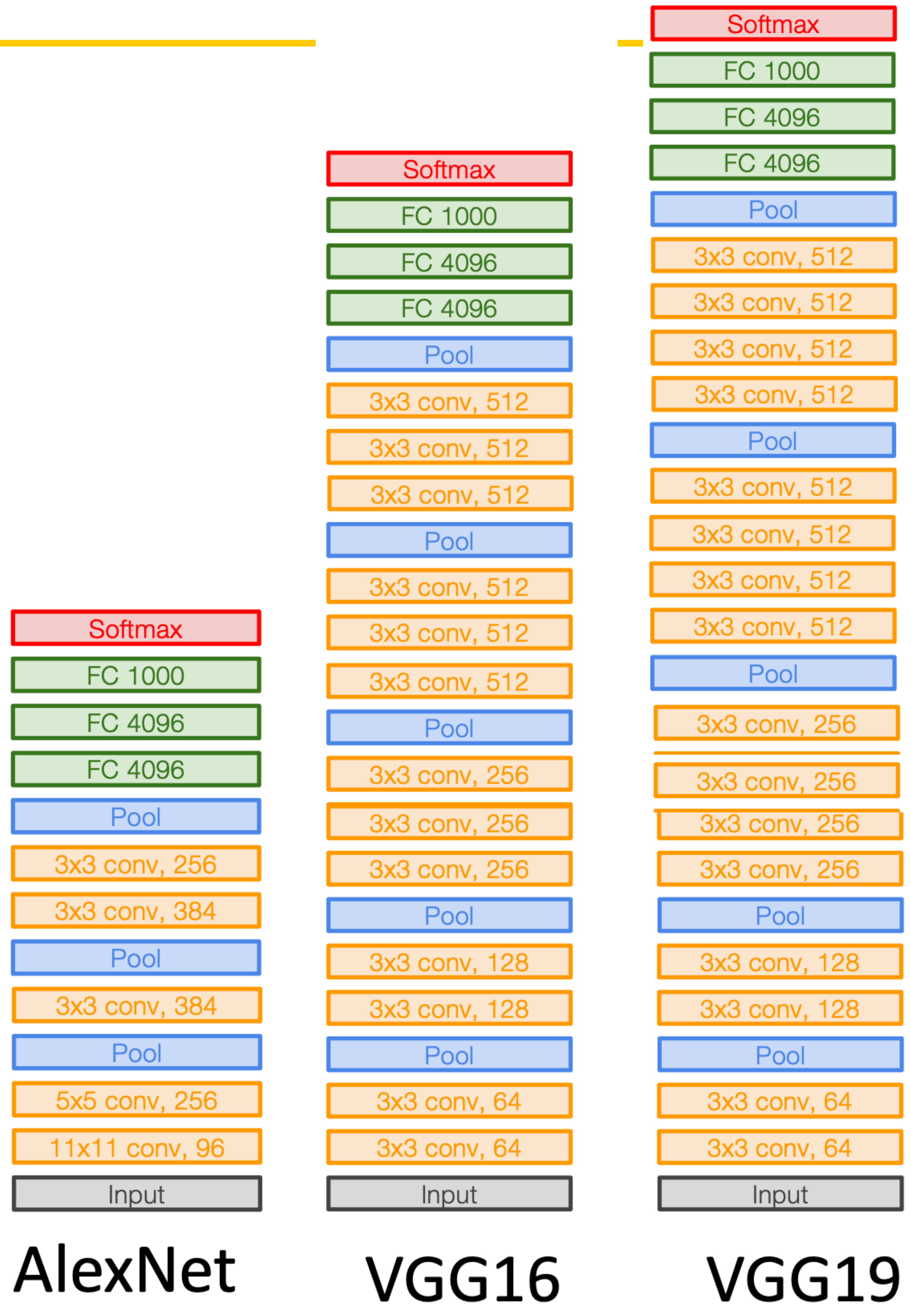
After pool, double #channels

Option 1:

Conv(5x5, C->C)

Params: $25C^2$

FLOPs: $25C^2HW$



VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Option 1:

Conv(5x5, C->C)

Params: $25C^2$

FLOPs: $25C^2HW$

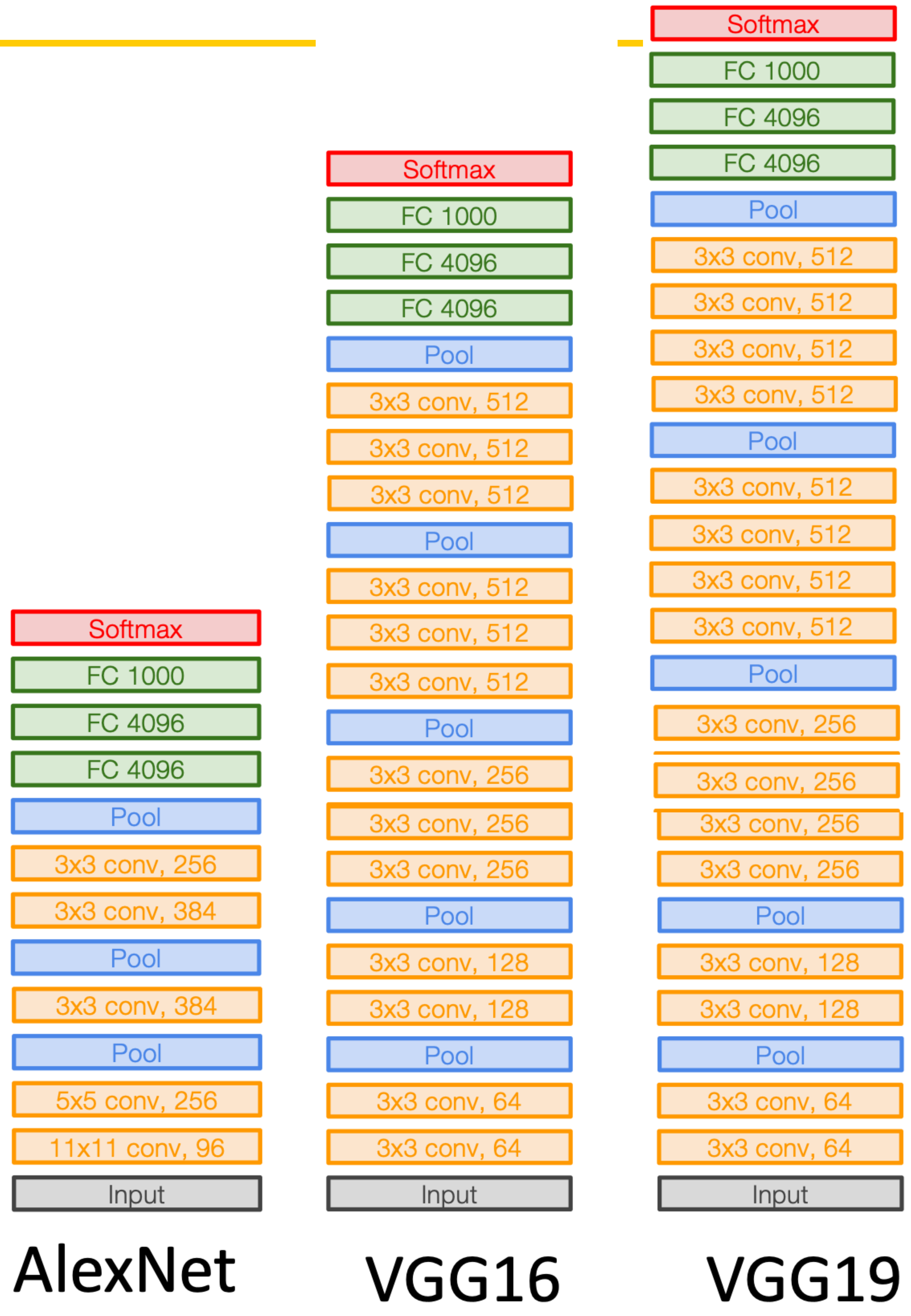
Option 2:

Conv(3x3, C->C)

Conv(3x3, C->C)

Params: $9C^2 + 9C^2 = 18C^2$

FLOPs: $18C^2HW$



VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Two 3x3 conv has same receptive field as a single 5x5 conv, but has fewer parameters and takes less computation!

Option 1:

Conv(5x5, C->C)

Params: $25C^2$

FLOPs: $25C^2HW$

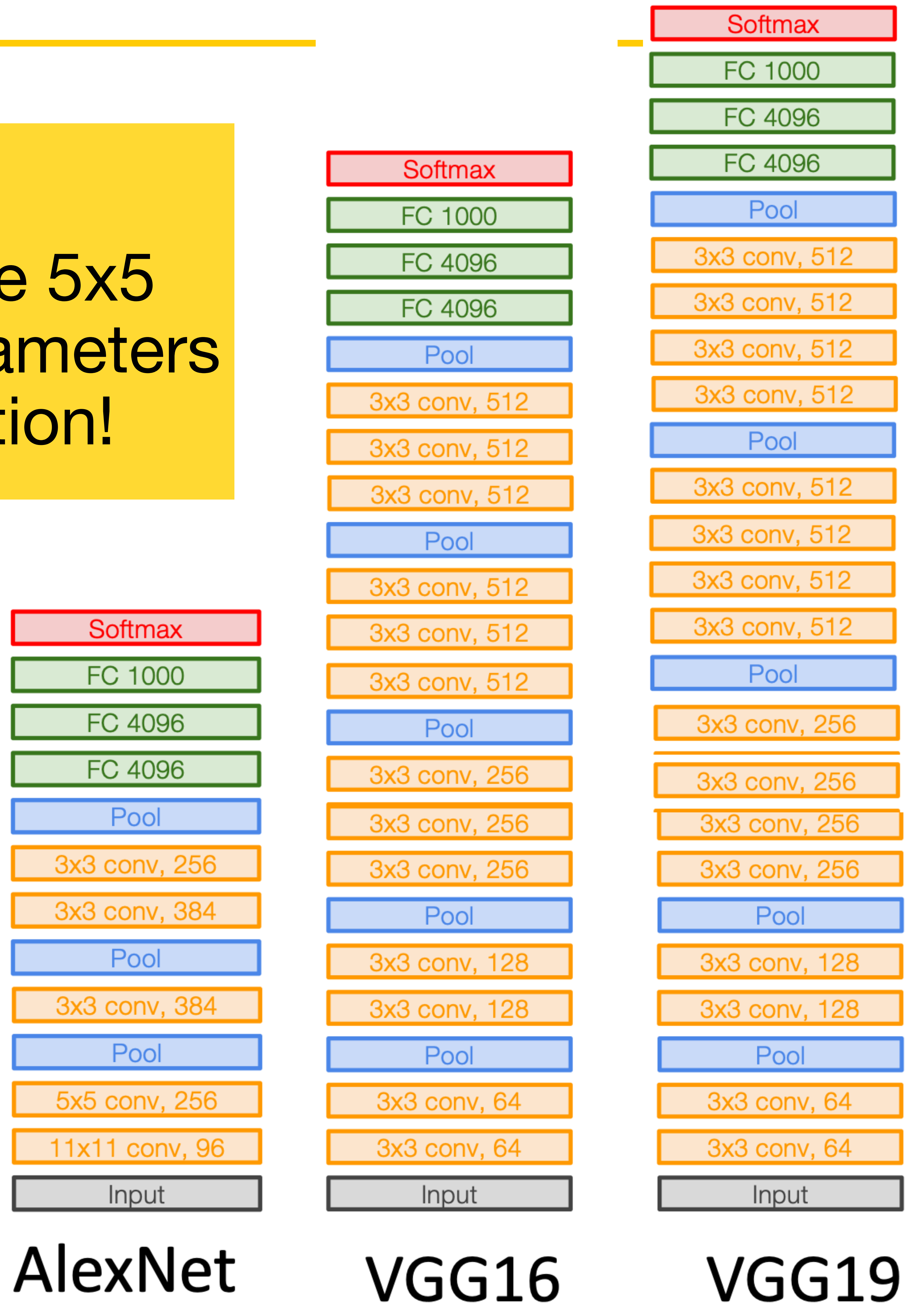
Option 2:

Conv(3x3, C->C)

Conv(3x3, C->C)

Params: $18C^2$

FLOPs: $18C^2HW$



VGG: Deeper Networks, Regular Design

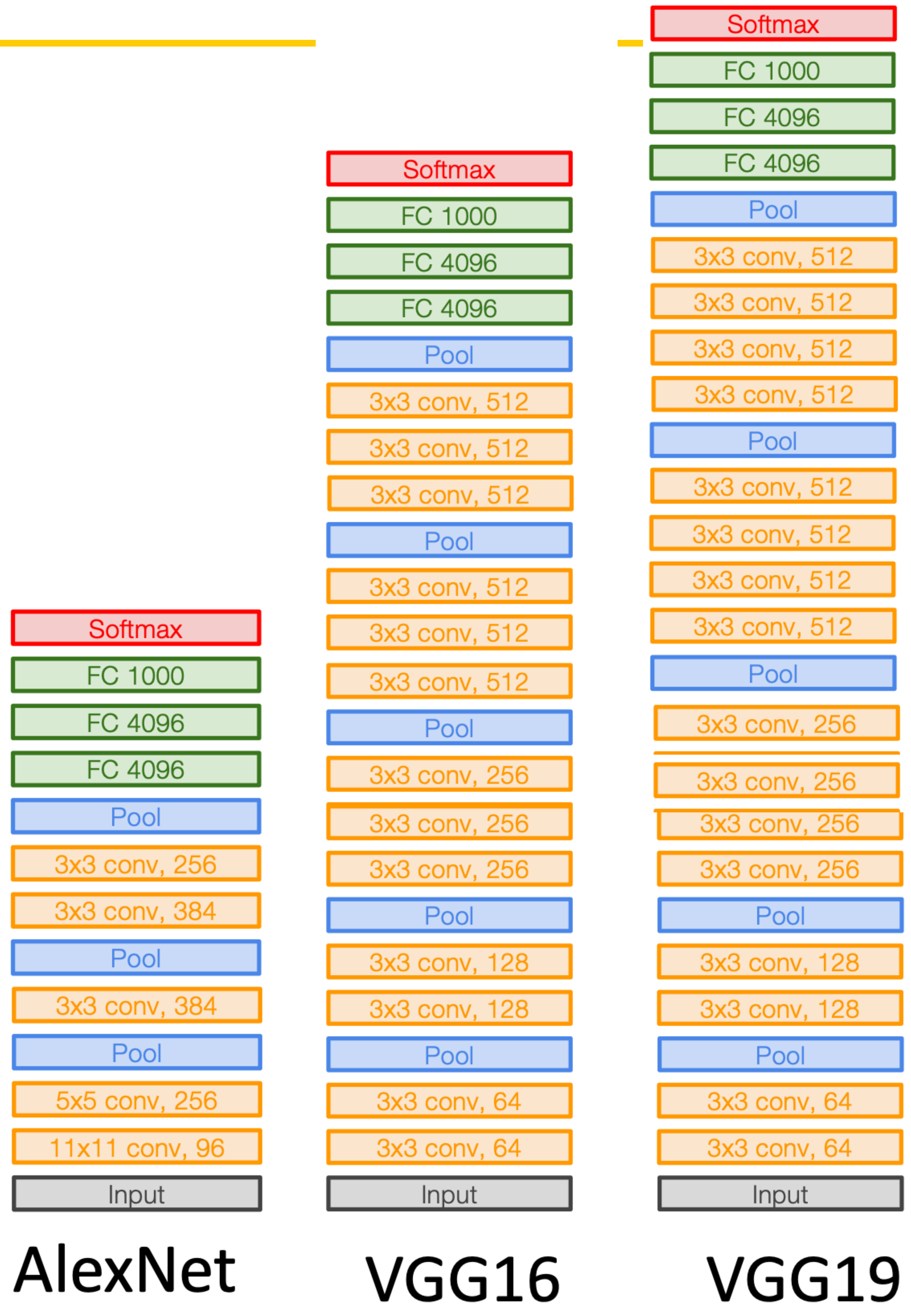
VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

After pool, double #channels

Option 1:
Input: C x 2H x 2W
Layer: Conv(3x3, C->C)
Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$



VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

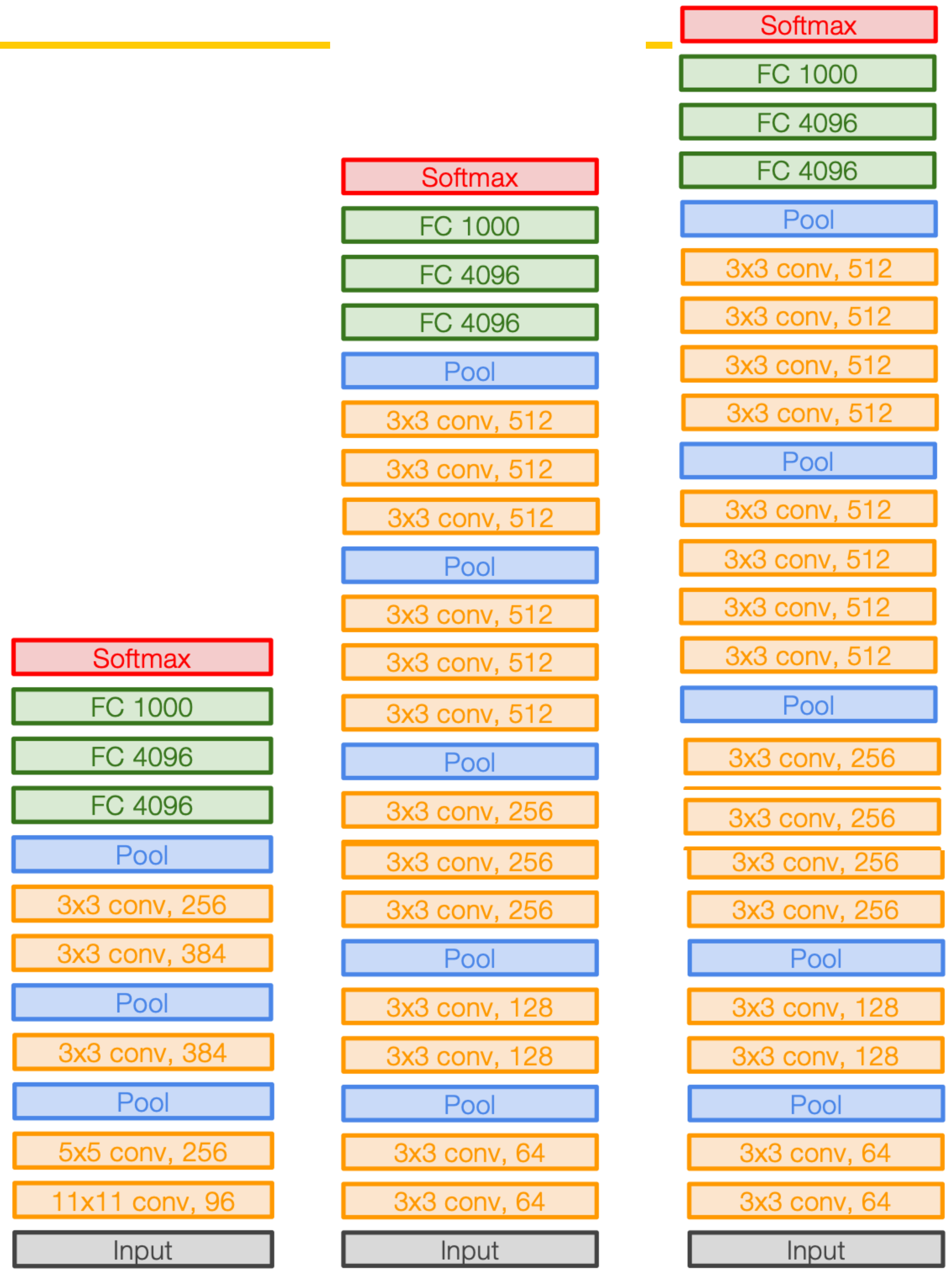
All max pool are 2x2 stride 2

After pool, double #channels

Option 1:
Input: C x 2H x 2W
Layer: Conv(3x3, C->C)
Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$

Option 2:
Input: 2C x H x W
Layer: Conv(3x3, 2C->2C)

Memory: 2HWC
Params: $36C^2$
FLOPs: $36HWC^2$



AlexNet VGG16 VGG19

VGG: Deeper Networks, Regular Design

VGG Design rules:

All conv are 3x3 stride 1 pad 1

All max pool are 2x2 stride 2

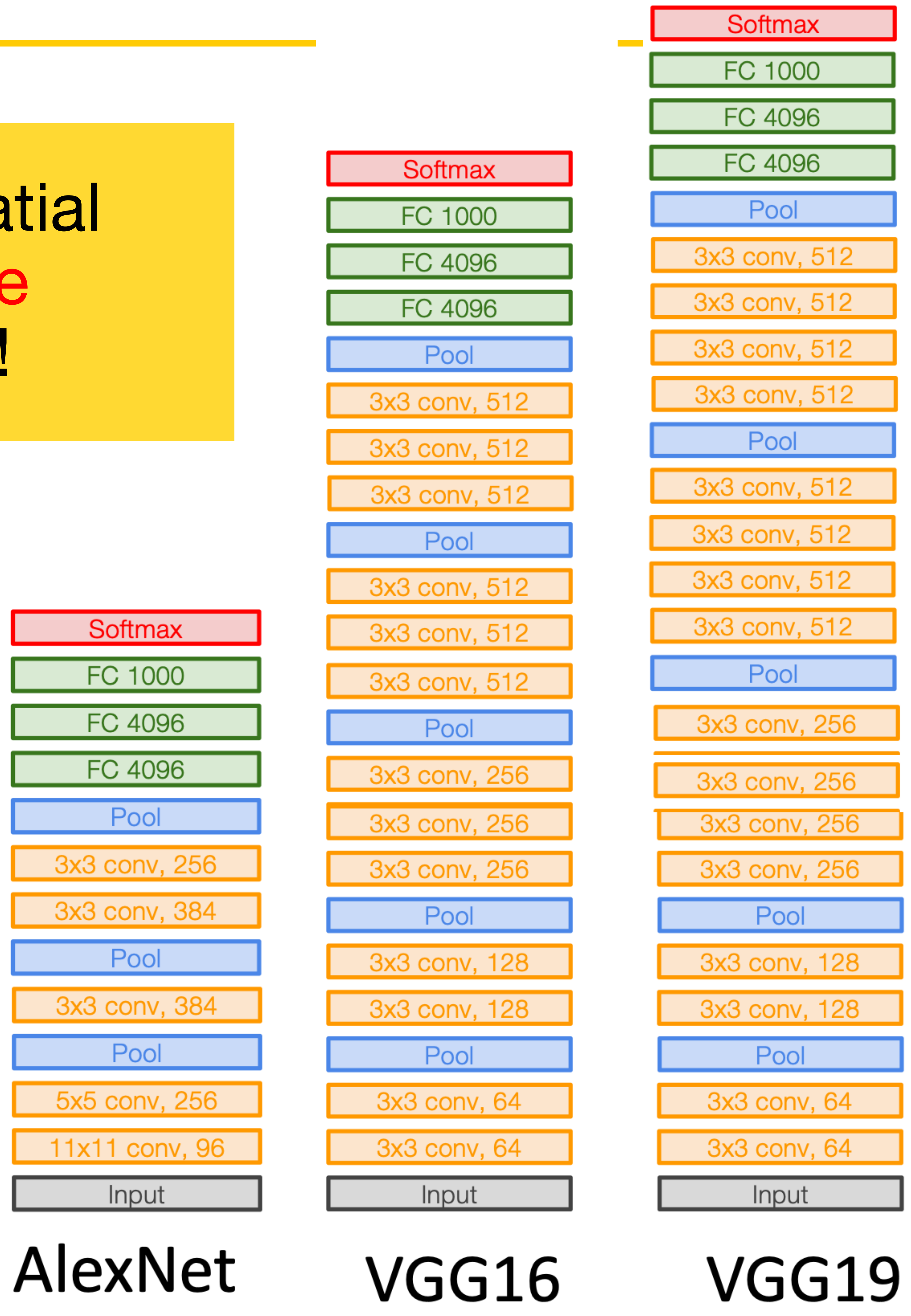
After pool, double #channels

Conv layers at each spatial resolution take the same amount of computation!

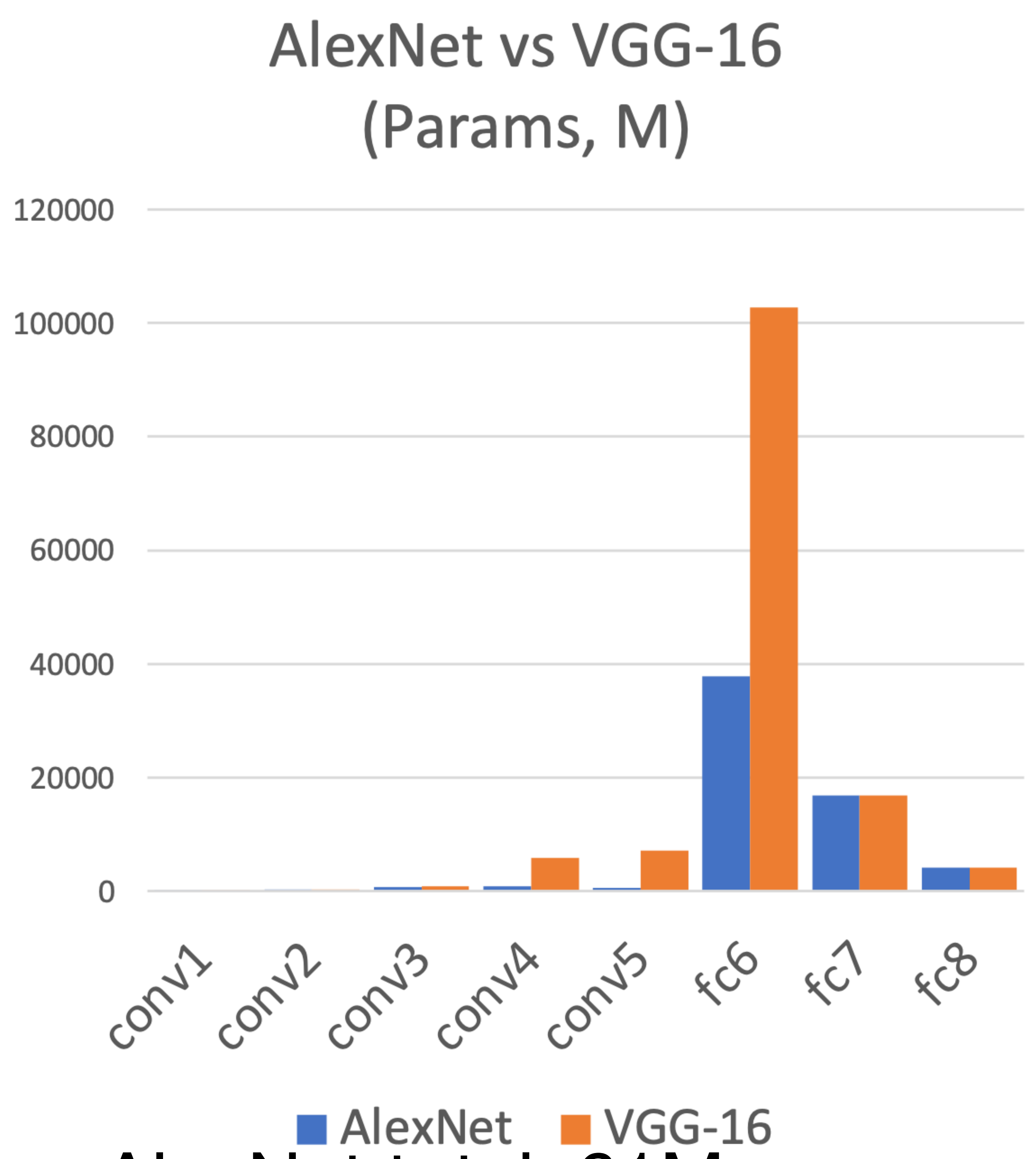
Option 1:
Input: C x 2H x 2W
Layer: Conv(3x3, C->C)
Memory: 4HWC
Params: $9C^2$
FLOPs: $36HWC^2$

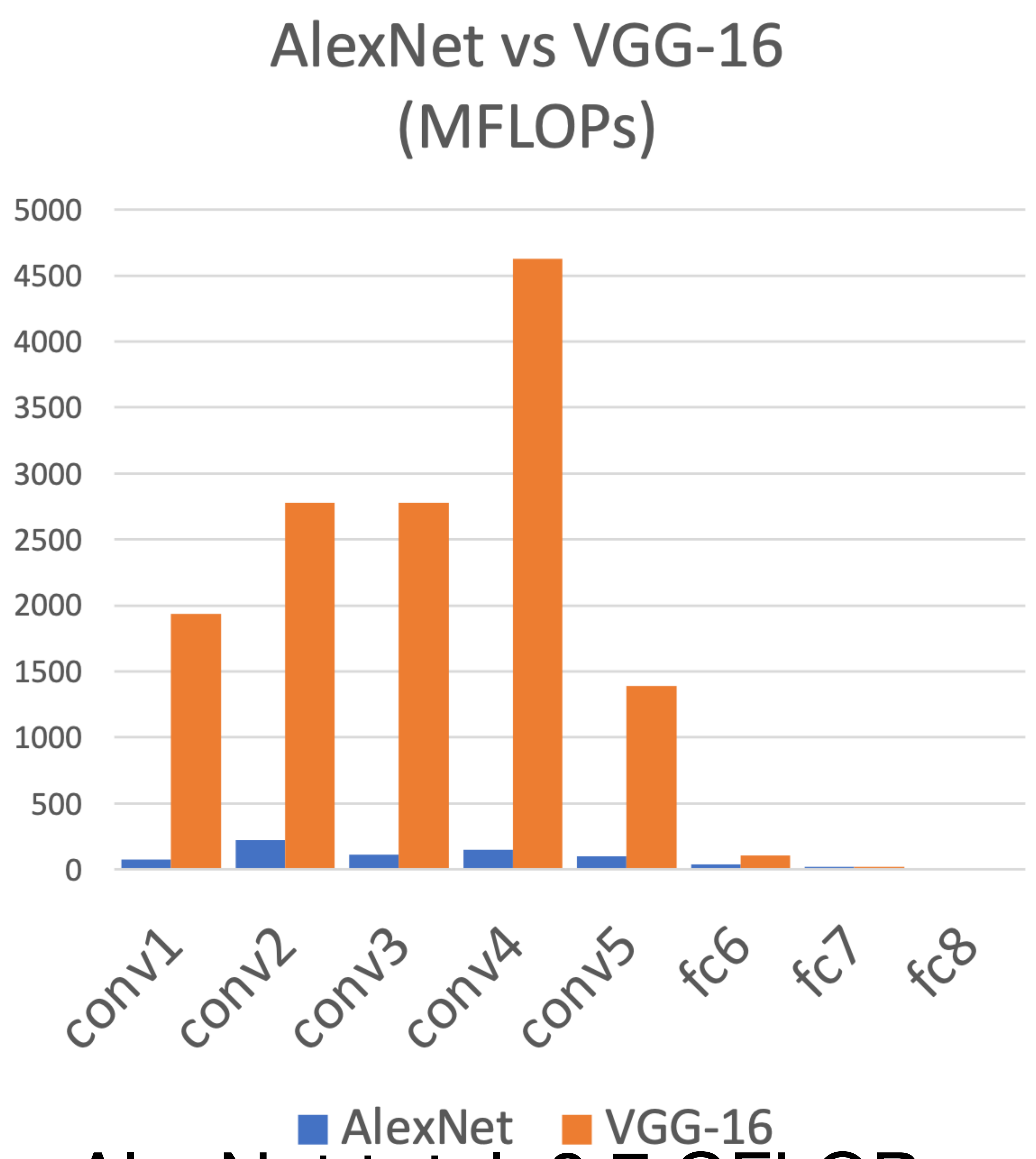
Option 2:
Input: 2C x H x W
Layer: Conv(3x3, 2C->2C)

Memory: 2HWC
Params: $36C^2$
FLOPs: $36HWC^2$



AlexNet vs VGG-16: Much bigger network!





AlexNet total: 1.9MB

VGG-16 total: 48.6MB (25x)

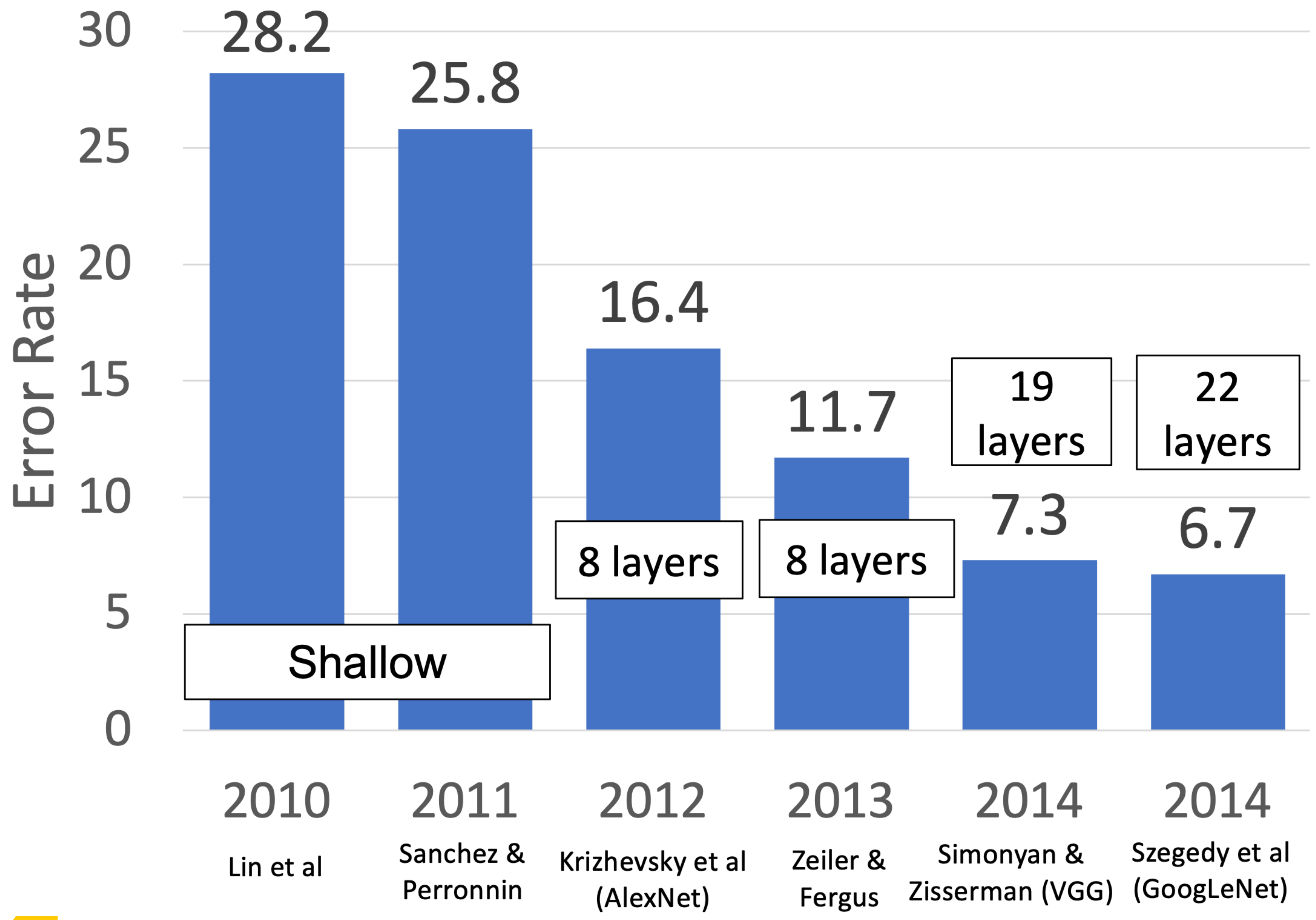
AlexNet total: 61M

VGG-16 total: 138M (2.3x)

AlexNet total: 0.7 GFLOP

VGG-16 total: 13.6 GFLOP (19.4x)

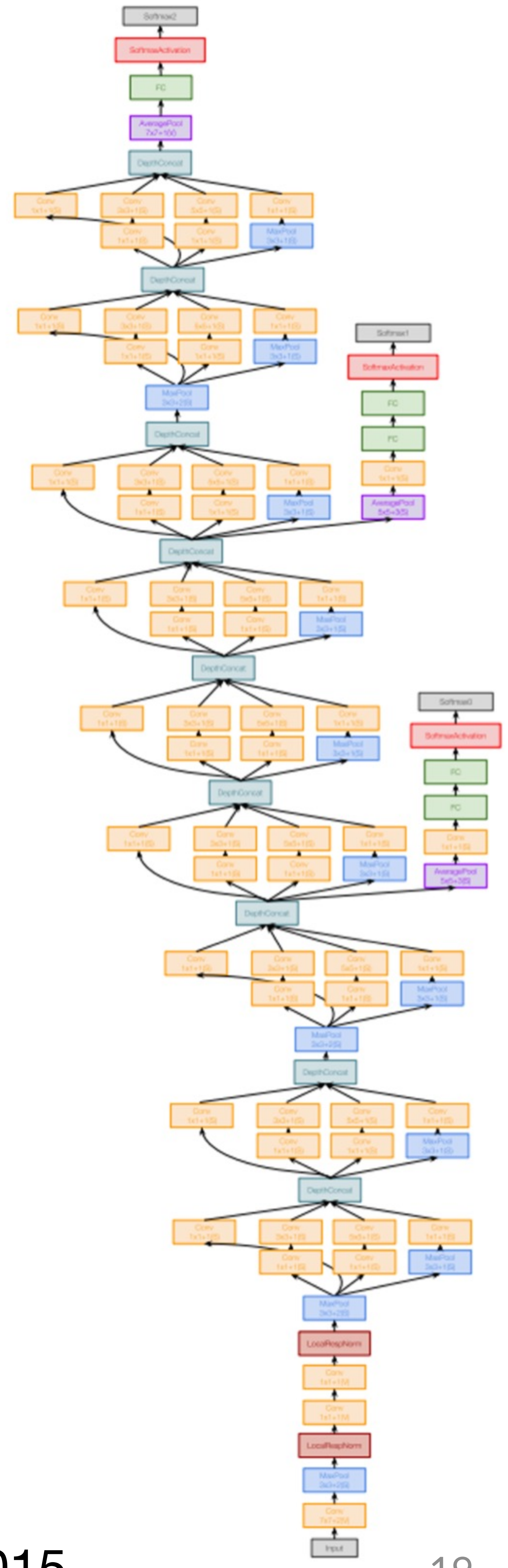
ImageNet Classification Challenge



GoogLeNet: Focus on Efficiency

"Inception v1"

Many innovations for efficiency: reduce parameter count, memory usage, and computation

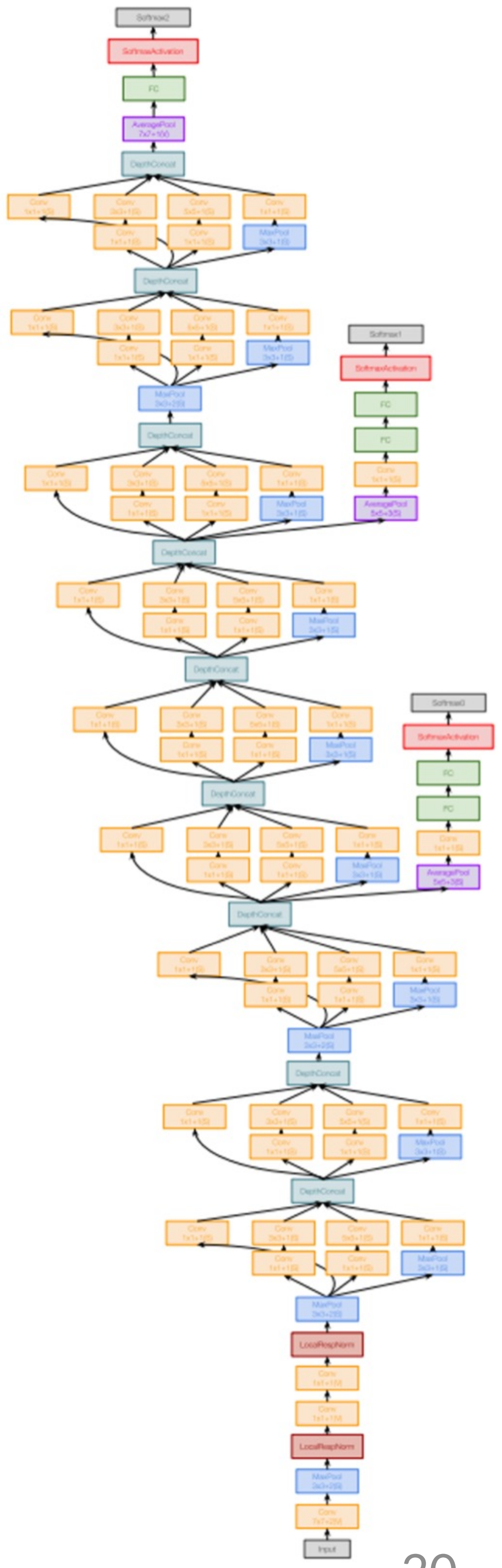


Szegedy et al, "Going deeper with convolutions", CVPR 2015

GoogLeNet: Aggressive Stem

Multi-Branch Networks

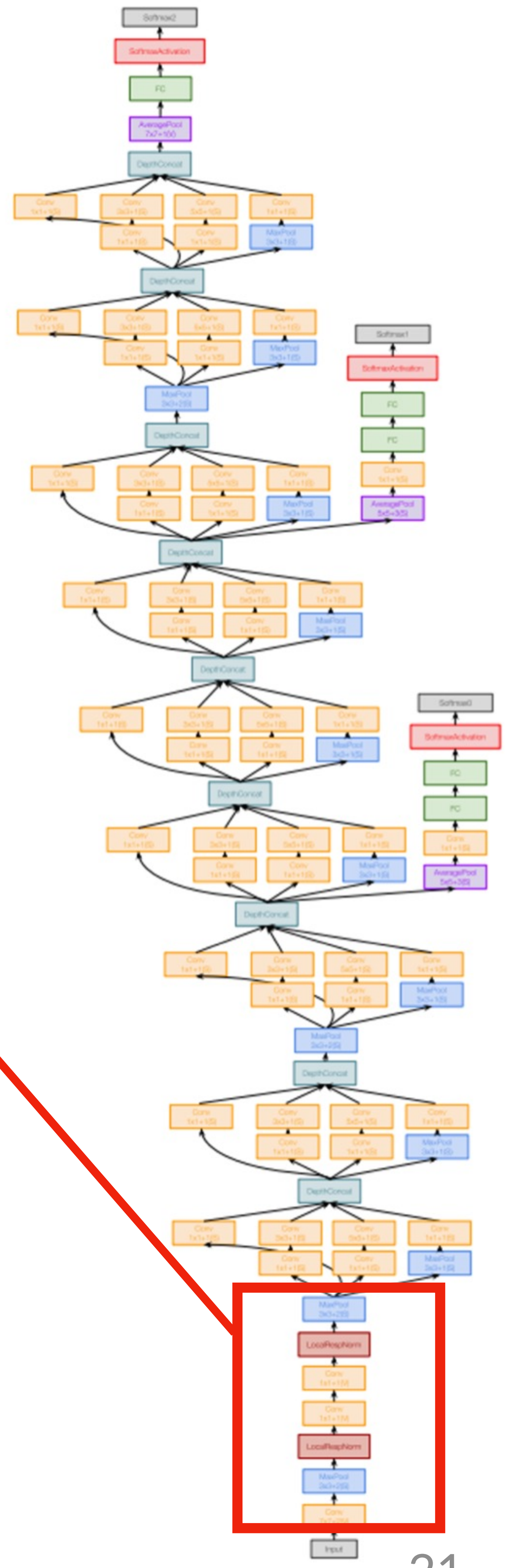
Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)



GoogLeNet: Aggressive Stem

Stem network at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Strid	Pad	C	H/W	Memory	Params	Flop (M)
Conv	3	224	64	7	2	3	64	112	3136	9	118
Max-pool	64	112		3	2	1	64	56	784	0	2
Conv	64	56	64	1	1	0	64	56	784	4	13
Conv	64	56	192	3	1	1	192	56	2352	111	347
Max-pool	192	56		3	2	1	192	28	588	0	1



Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

GoogLeNet: Inception Module

Inception module: Local unit with parallel branches

Local structure repeated many times throughout the network

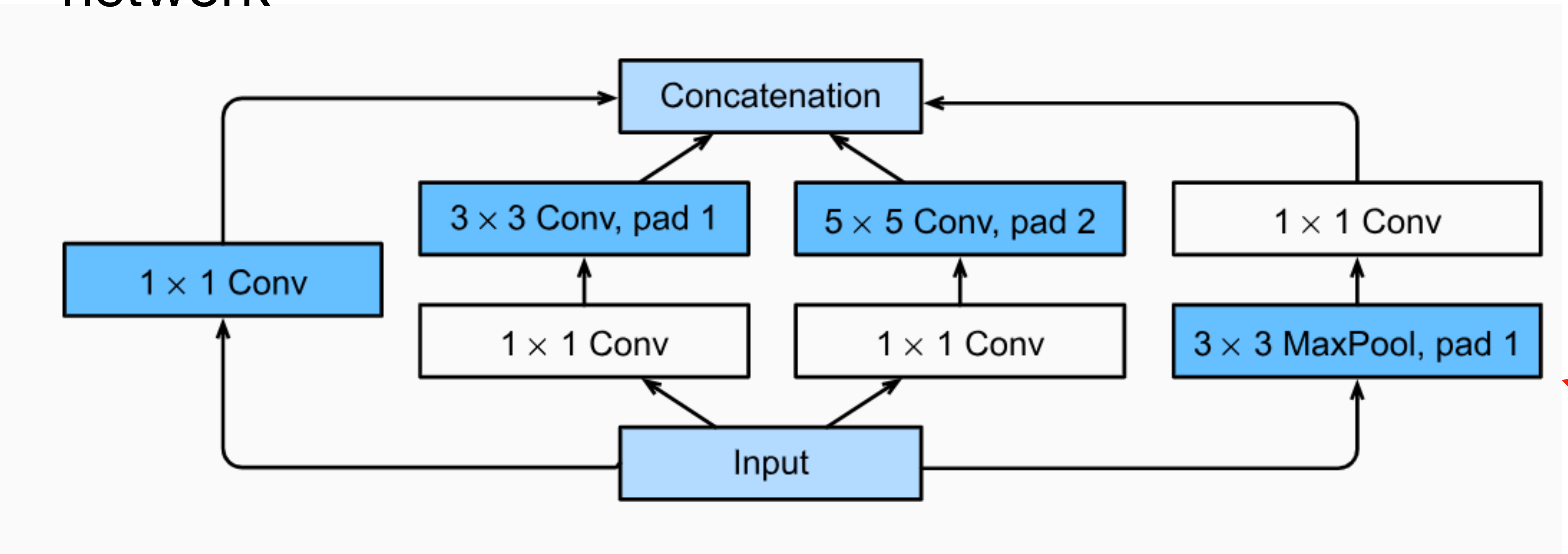
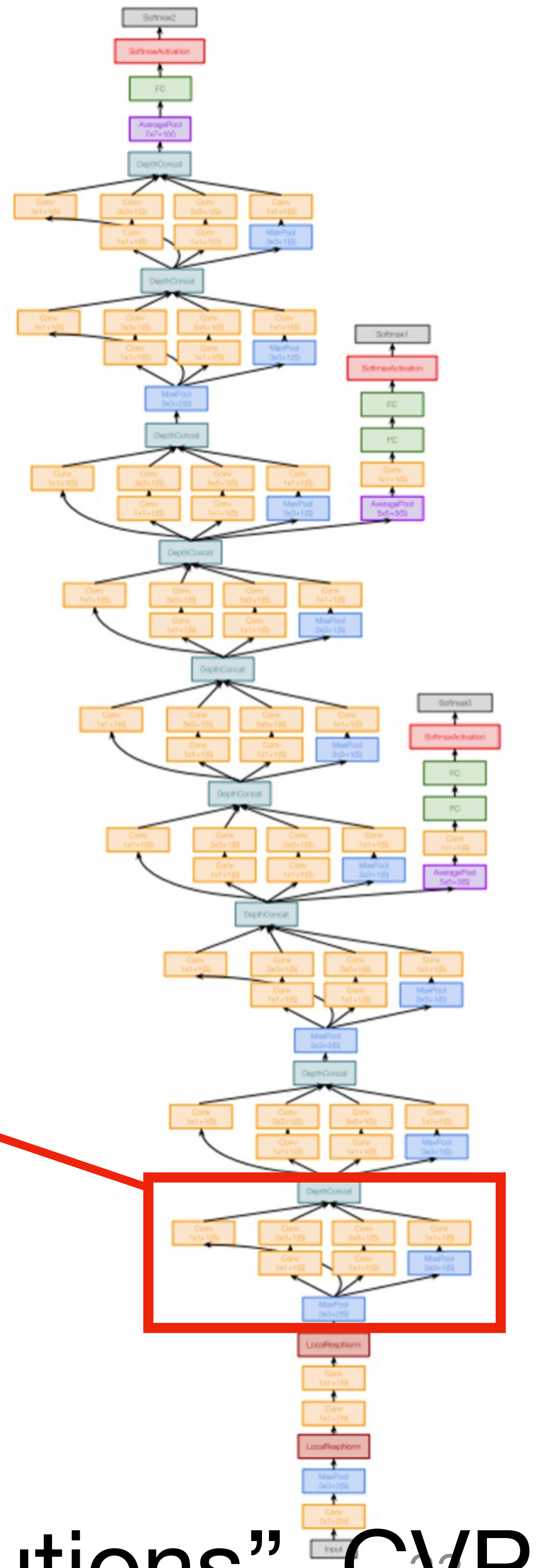


Fig. 8.4.1 Structure of the Inception block.



Szegedy et al, "Going deeper with convolutions", CVPR 201

GoogLeNet: Inception Module

Inception module: Local unit with parallel branches

Local structure repeated many times throughout the network

Uses 1x1 "Bottleneck" layers to reduce channel dimension before expensive conv (we will revisit this with ResNet!)

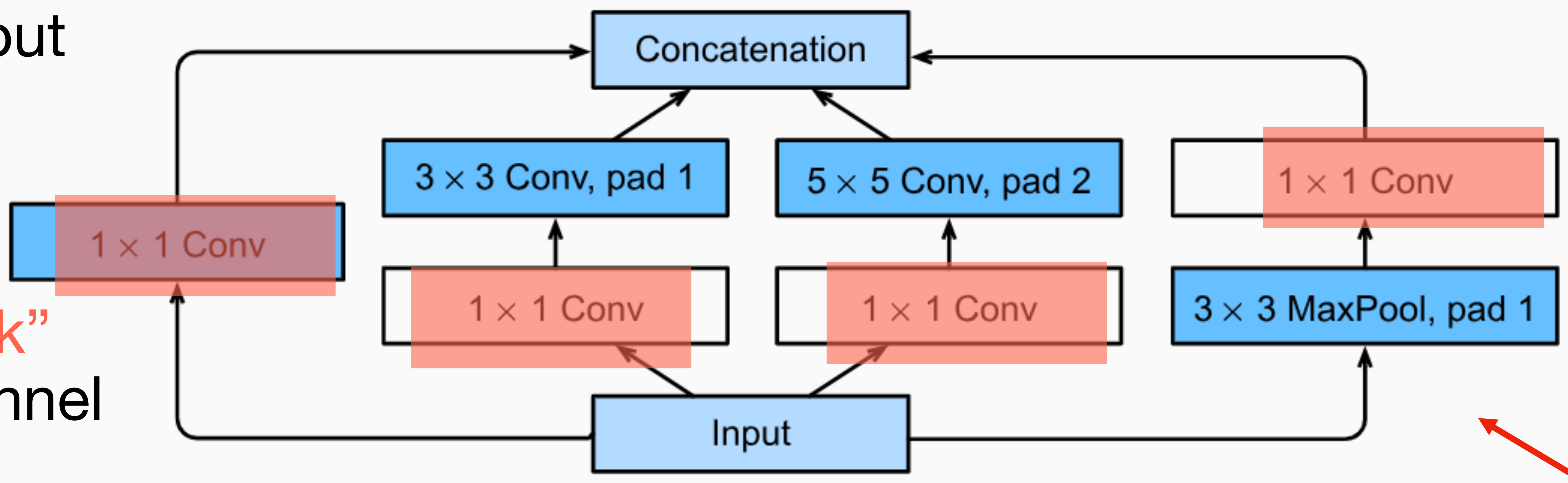
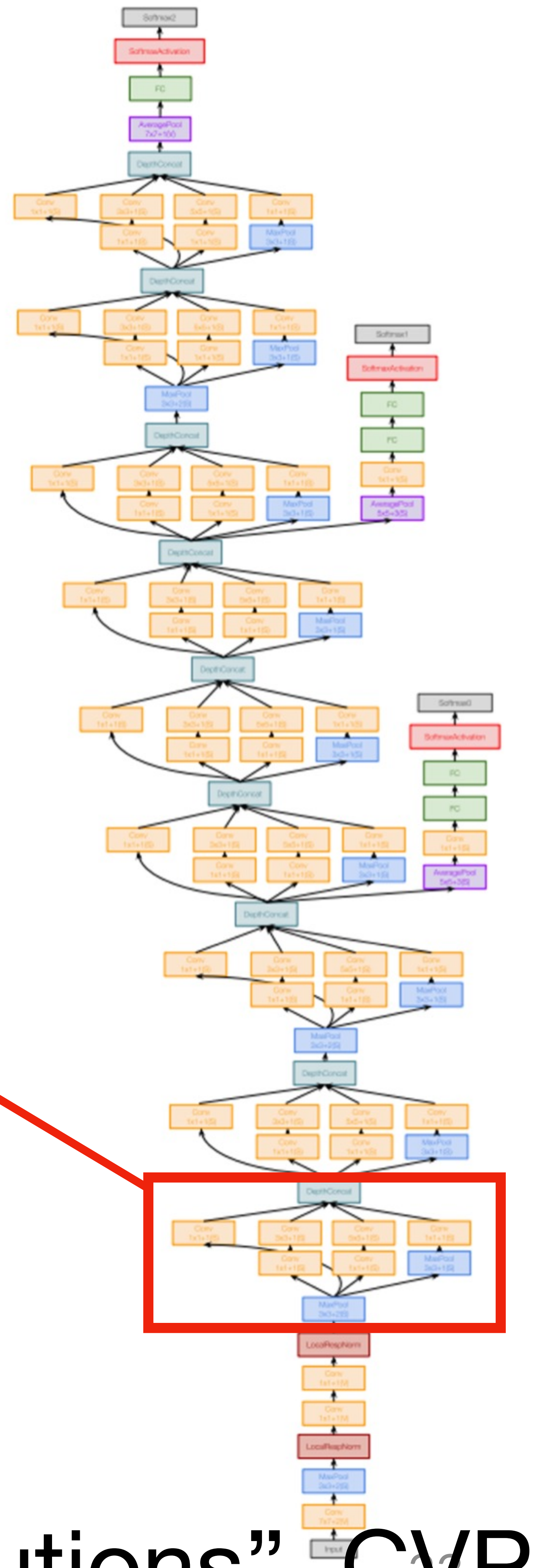


Fig. 8.4.1 Structure of the Inception block.



Szegedy et al, "Going deeper with convolutions", CVPR 201

GoogLeNet: Inception Module

Inception modules throughout the network

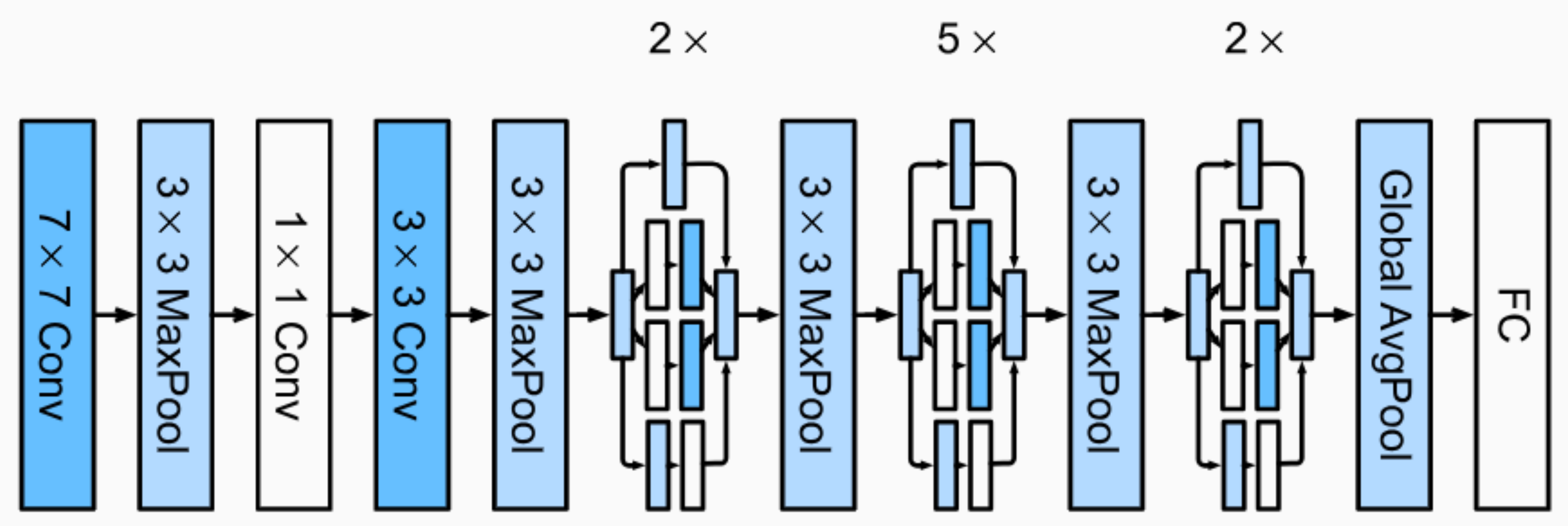
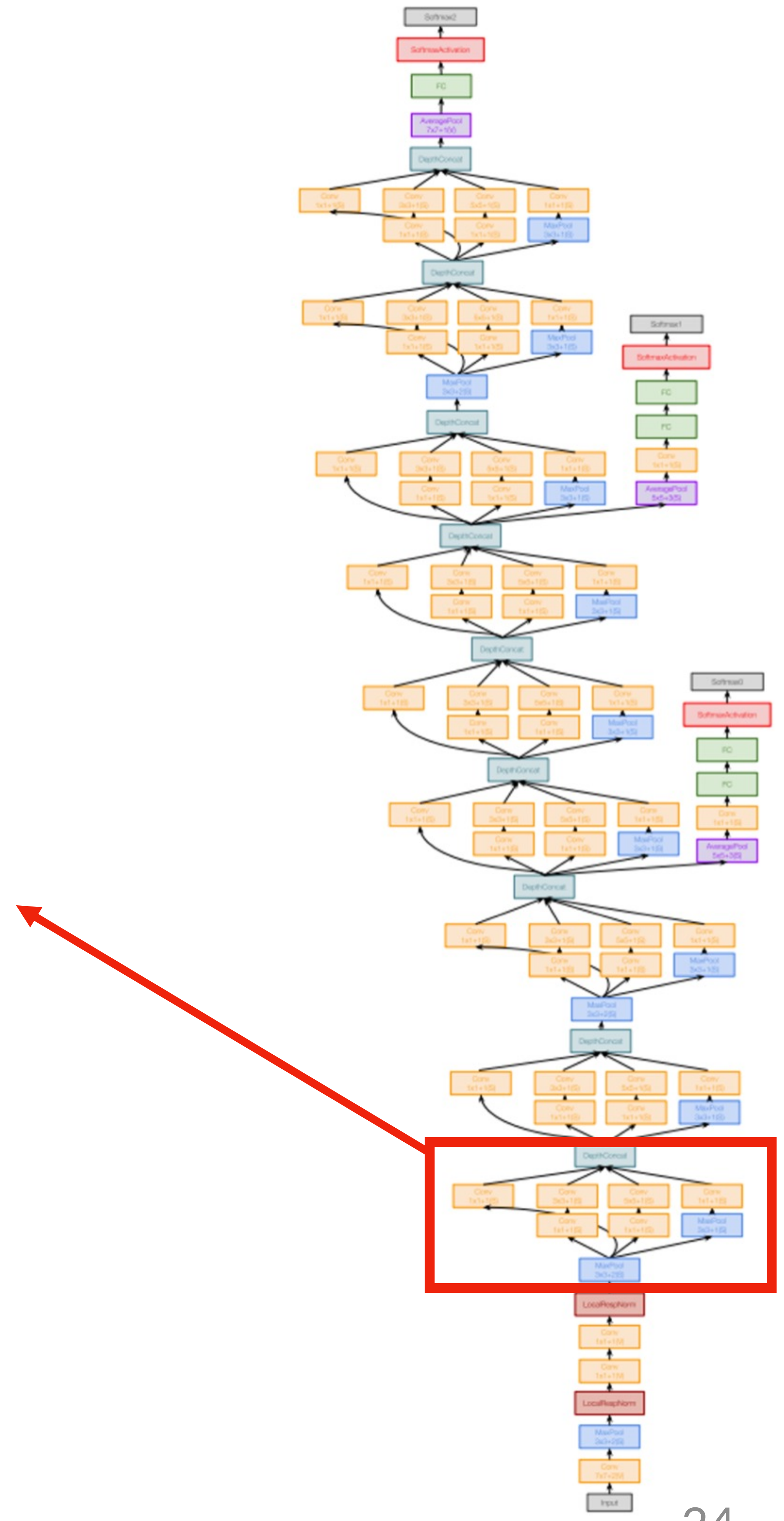


Fig. 8.4.2 The GoogLeNet architecture.



https://d2l.ai/chapter_convolutional-modern/googlenet.html

GoogLeNet: Global Average Pooling

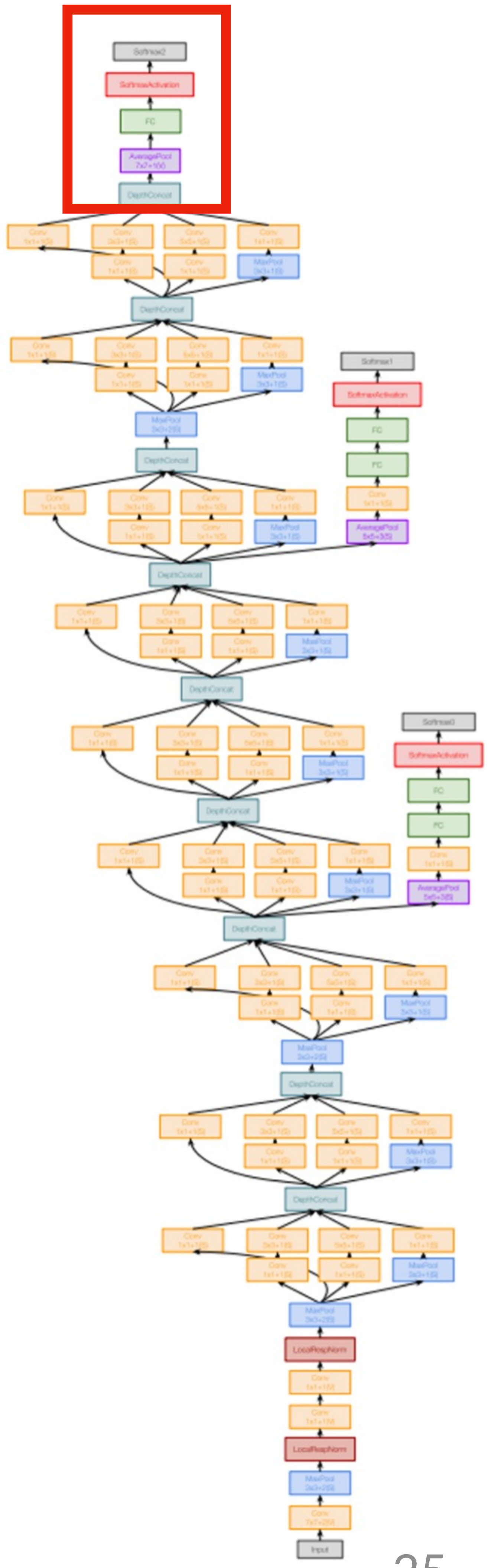
No large FC layers at the end!

Instead use **global average pooling** to collapse spatial dimensions, and one linear layer to produce class scores
(Recall VGG-16: Most parameters were in the FC layers!)

	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params	Flop (M)
avg-pool	1024	7		7	1	0	1024	1	4	0	0
fc	1024		1000				1000	0	0	1025	1

Compare with VGG-16:

	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params	Flop (M)
Flatten	512	7					25088		98		
FC6	25088			4096			4096		16	102760	103
FC7	4096			4096			4096		16	16777	17
FC8	4096			1000			1000		4	4096	4

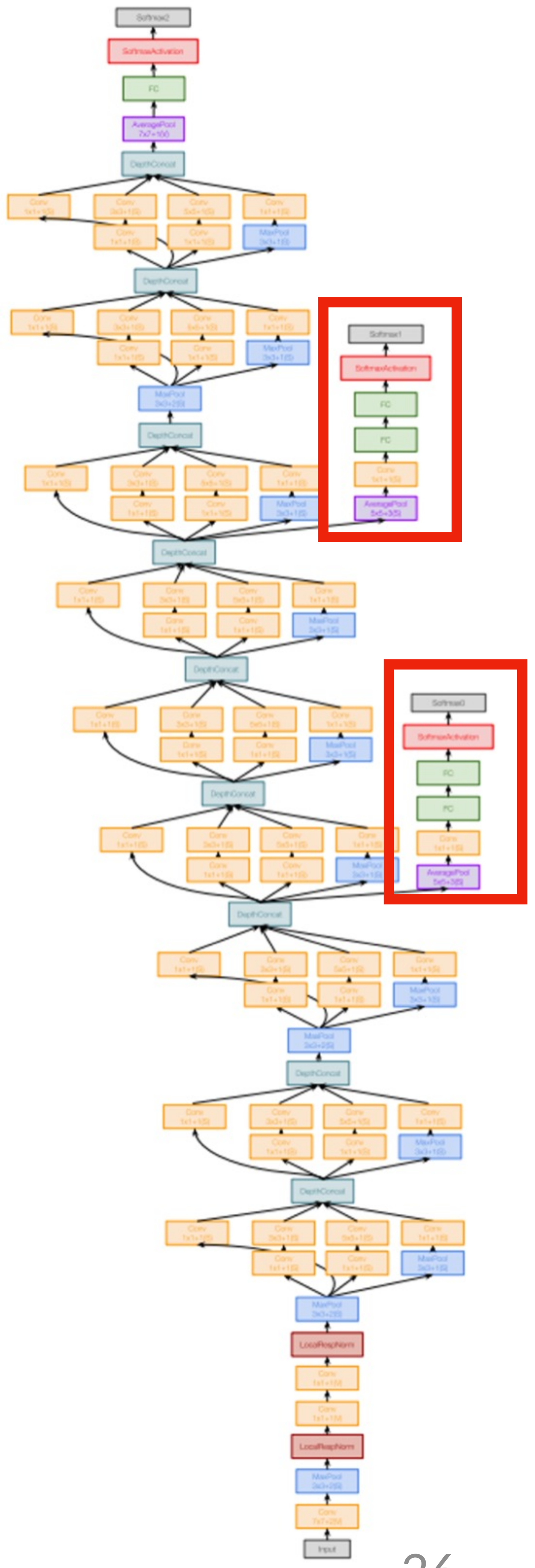


GoogLeNet: Auxiliary Classifiers

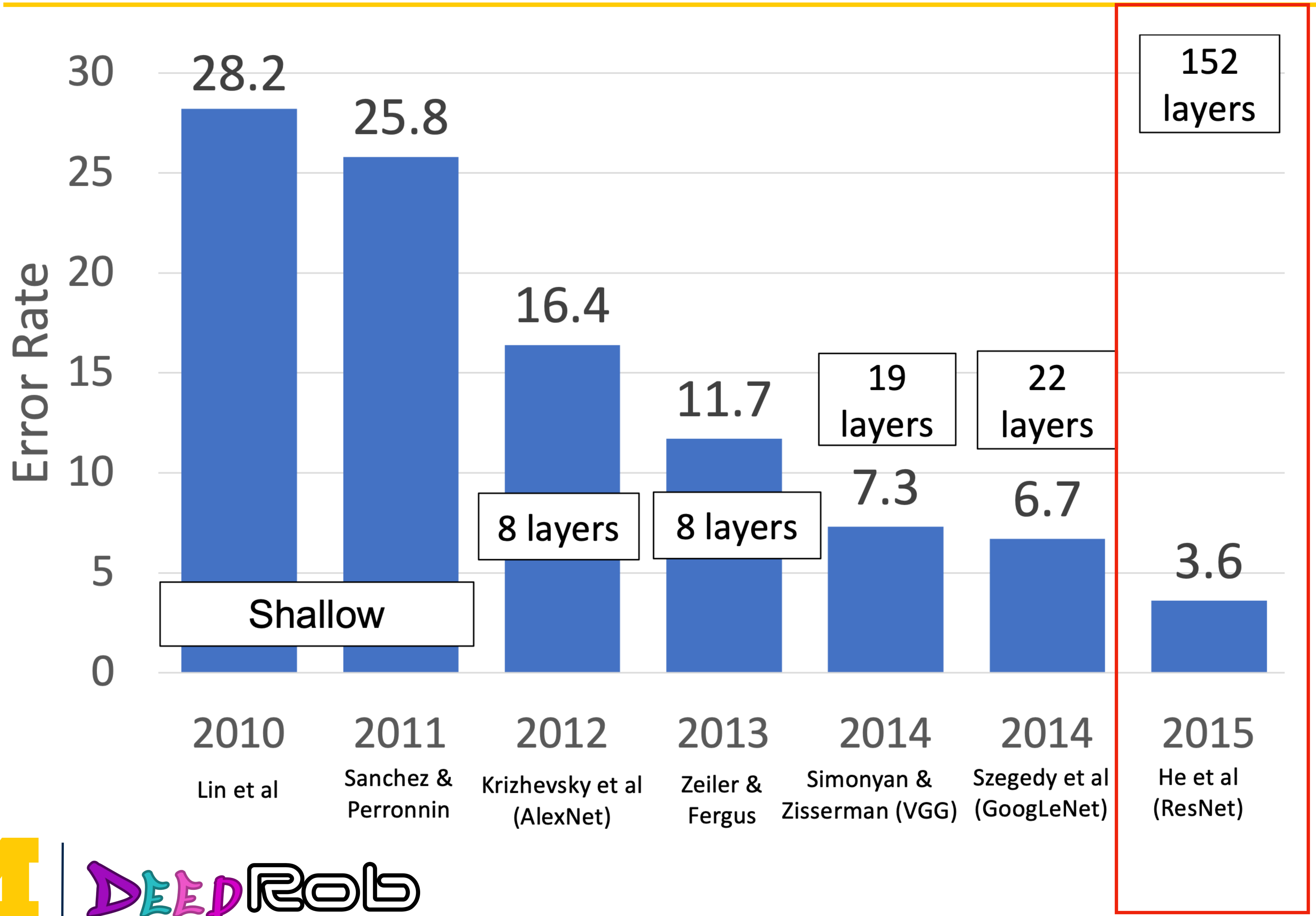
Training using loss at the end of the network didn't work well: Network is too deep, gradients don't propagate cleanly

As a hack, attach "auxiliary classifiers" at several intermediate points in the network that also try to classify the image and receive loss

GoogLeNet was before batch normalization! With BatchNorm, we no longer need to use this trick



ImageNet Classification Challenge



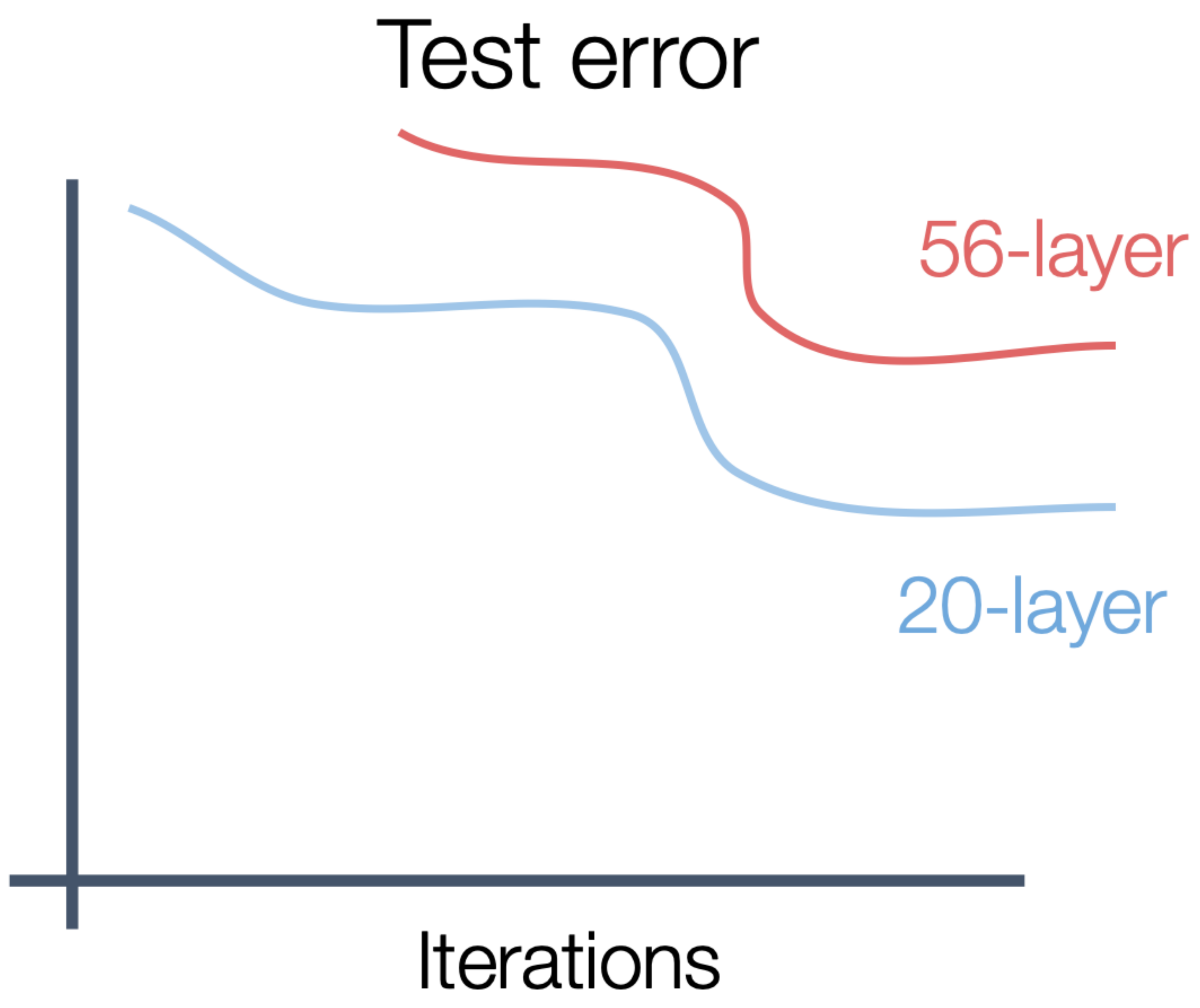
Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers.

What happens as we go deeper?

Deeper model does worse than shallow model!

Initial guess: Deep model is **overfitting** since it is much bigger than the other model



He et al, “Deep Residual Learning for Image Recognition”, CVPR 2016

Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers.

What happens as we go deeper?

In fact the deep model seems to be **underfitting** since it also performs worse than the shallow model on the training set! It is actually **underfitting**

Residual Networks

A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

Hypothesis: This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Residual Networks

A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

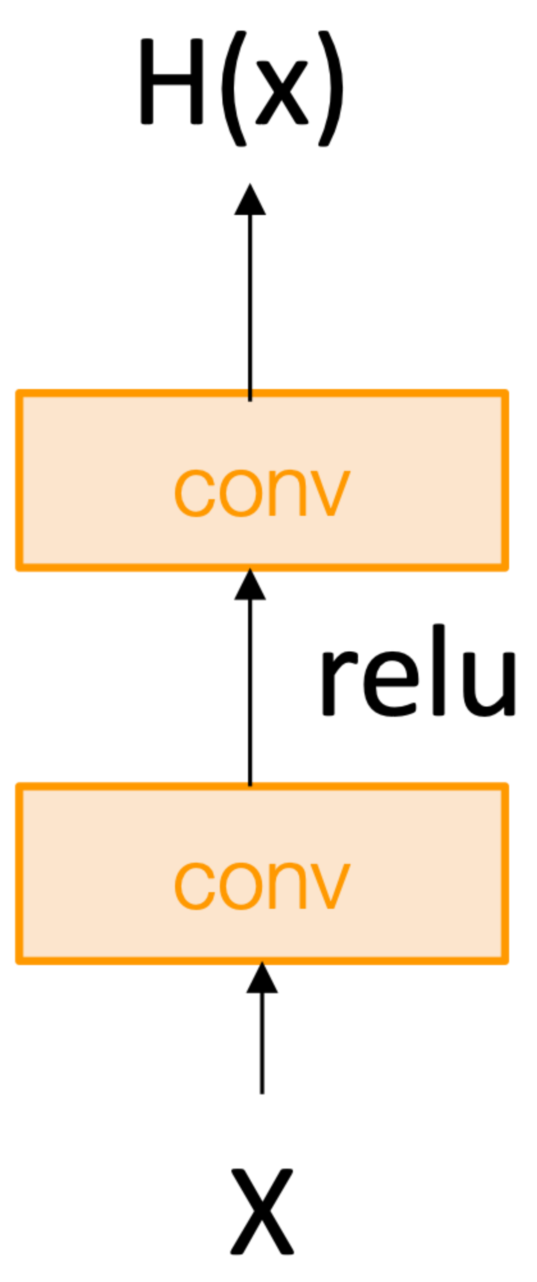
Thus deeper models should do at least as good as shallow models

Hypothesis: This is an optimization problem. Deeper models are harder to optimize, and in particular don't learn identity functions to emulate shallow models

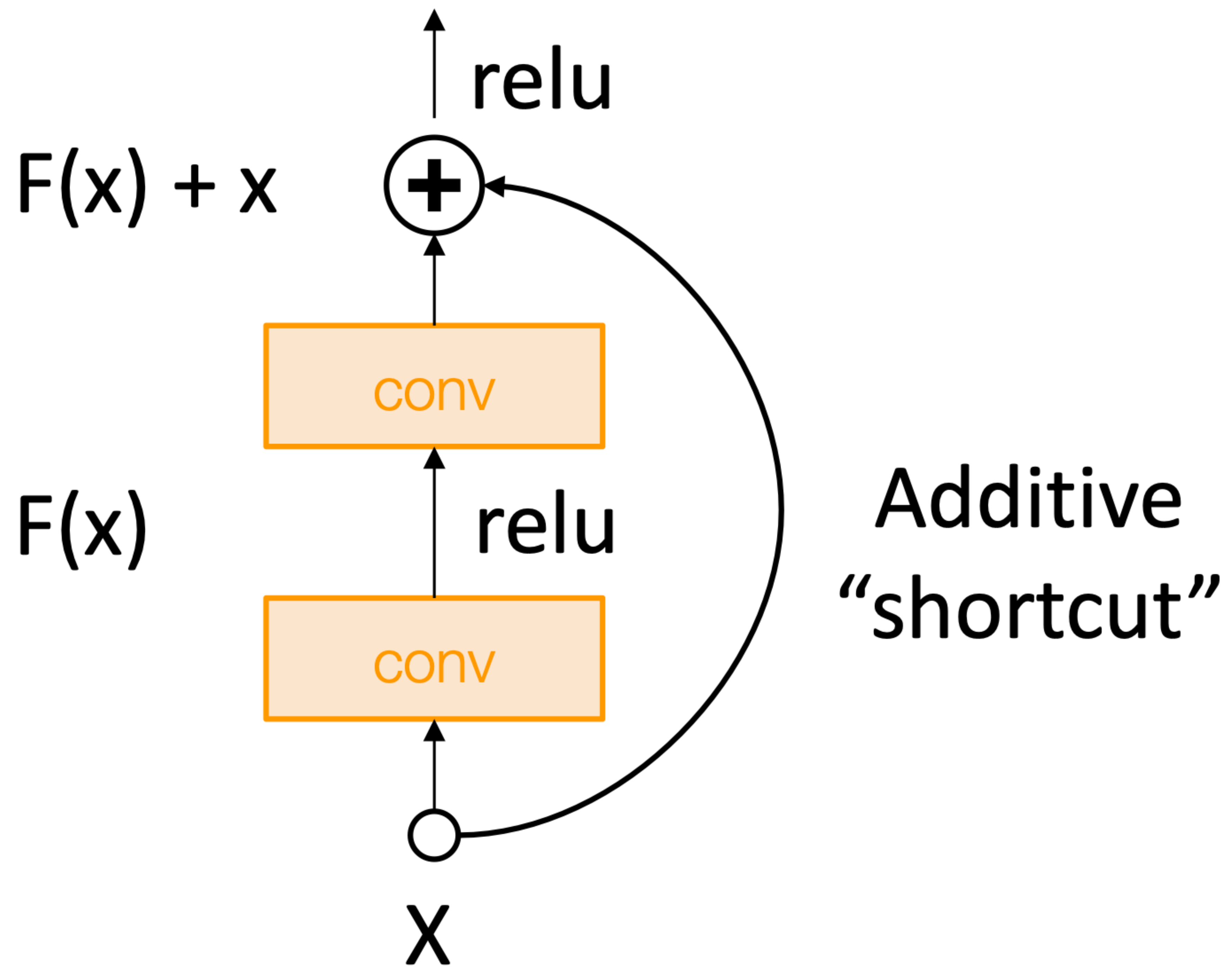
Solution: Change the network so learning identity functions with extra layers is easy!

Residual Networks

Solution: Change the network so learning identity functions with extra layers is easy!



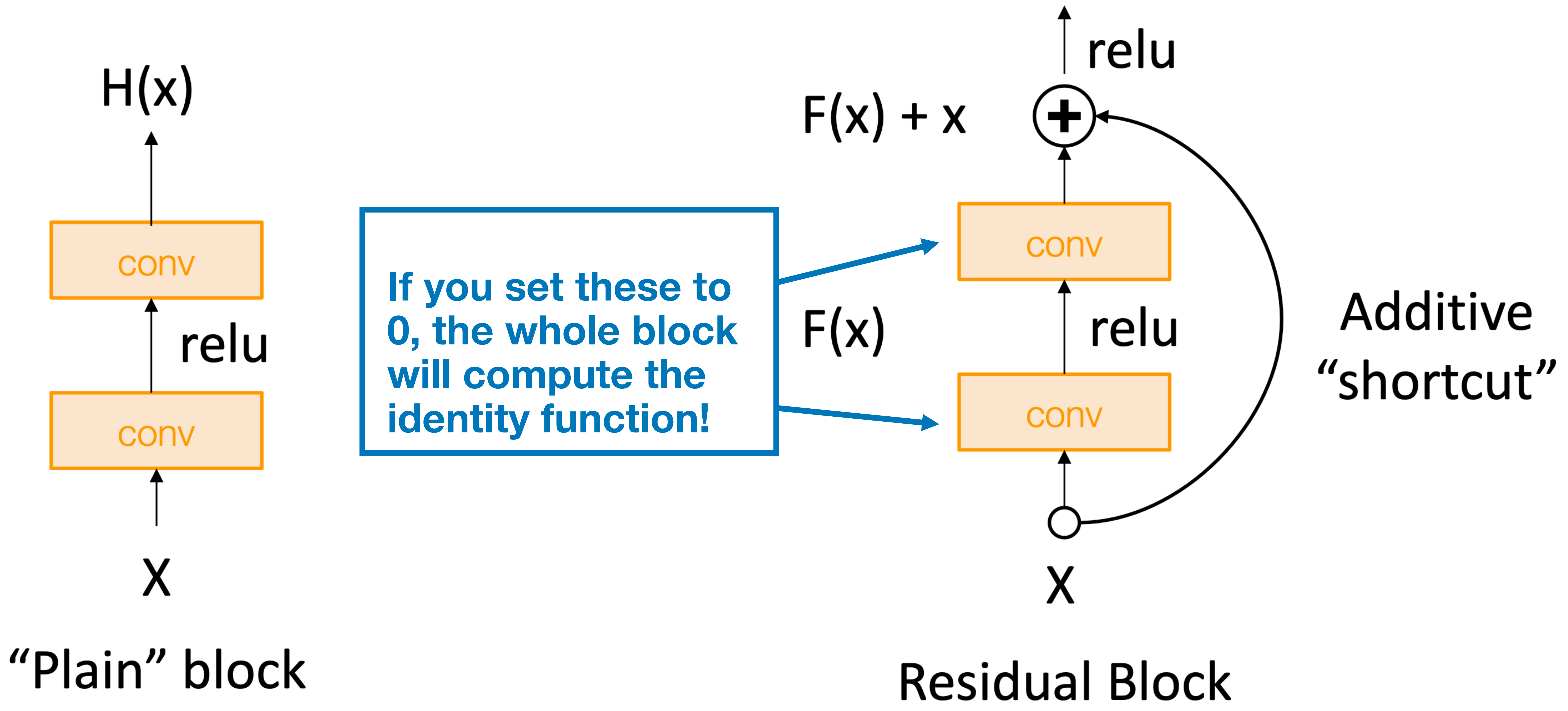
"Plain" block



Residual Block

Residual Networks

Solution: Change the network so learning identity functions with extra layers is easy!

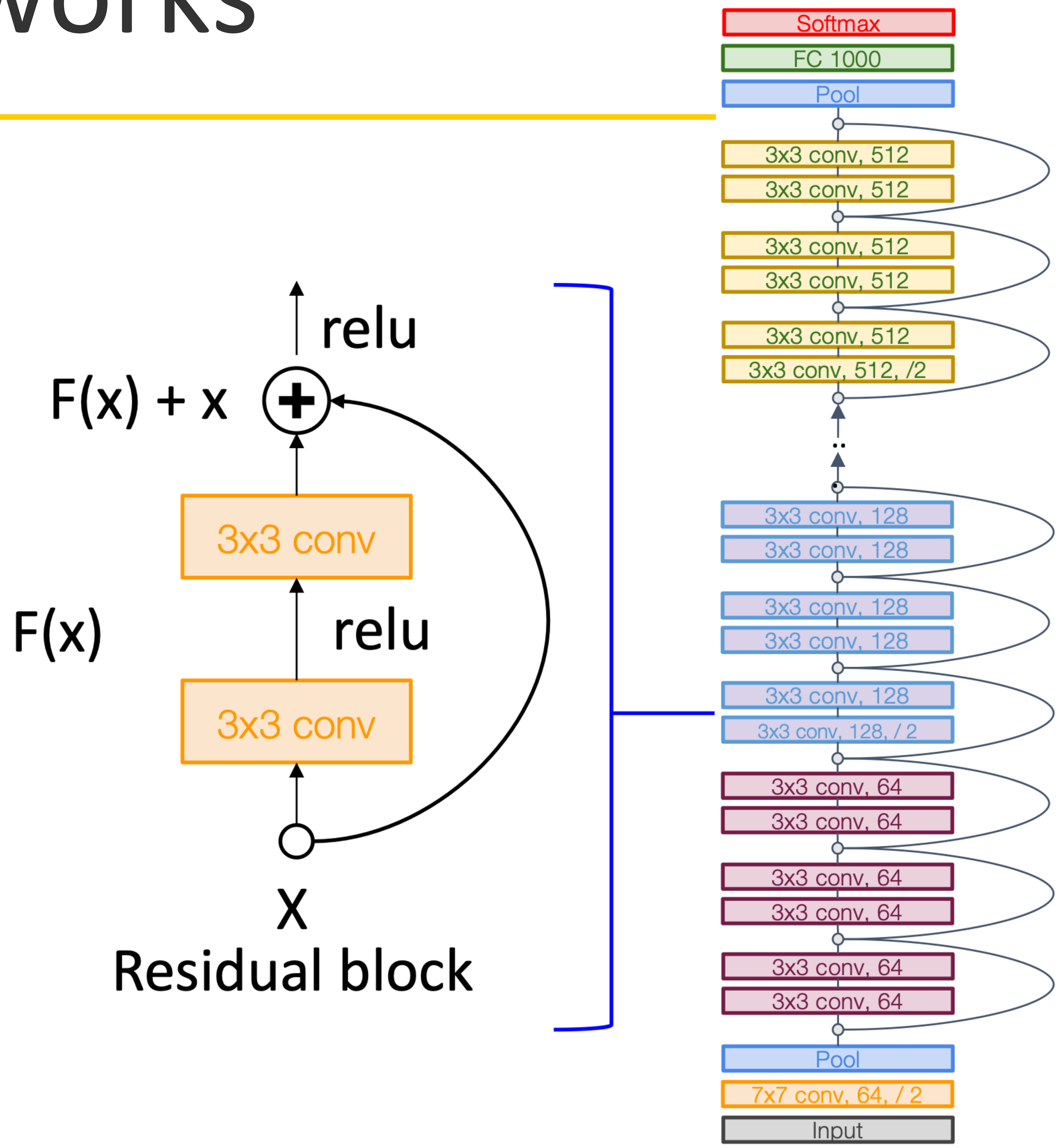


Residual Networks

A residual network is a stack of many residual blocks

Regular design, like VGG: each residual block has two 3x3 conv

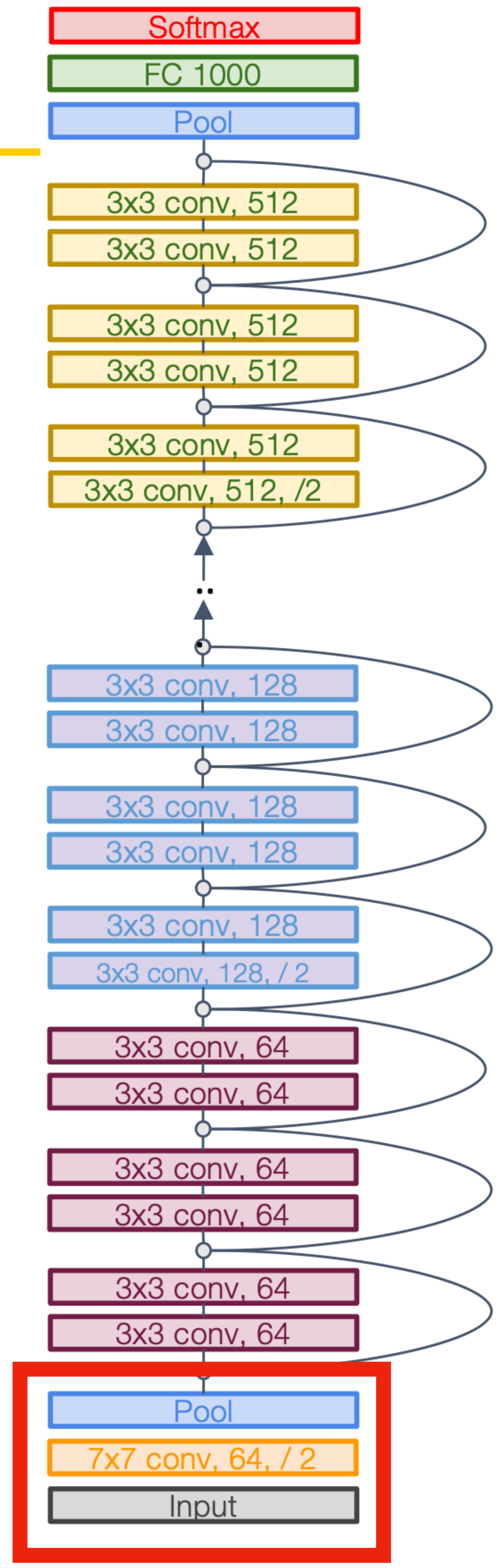
Network is divided into **stages**: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels



Residual Networks

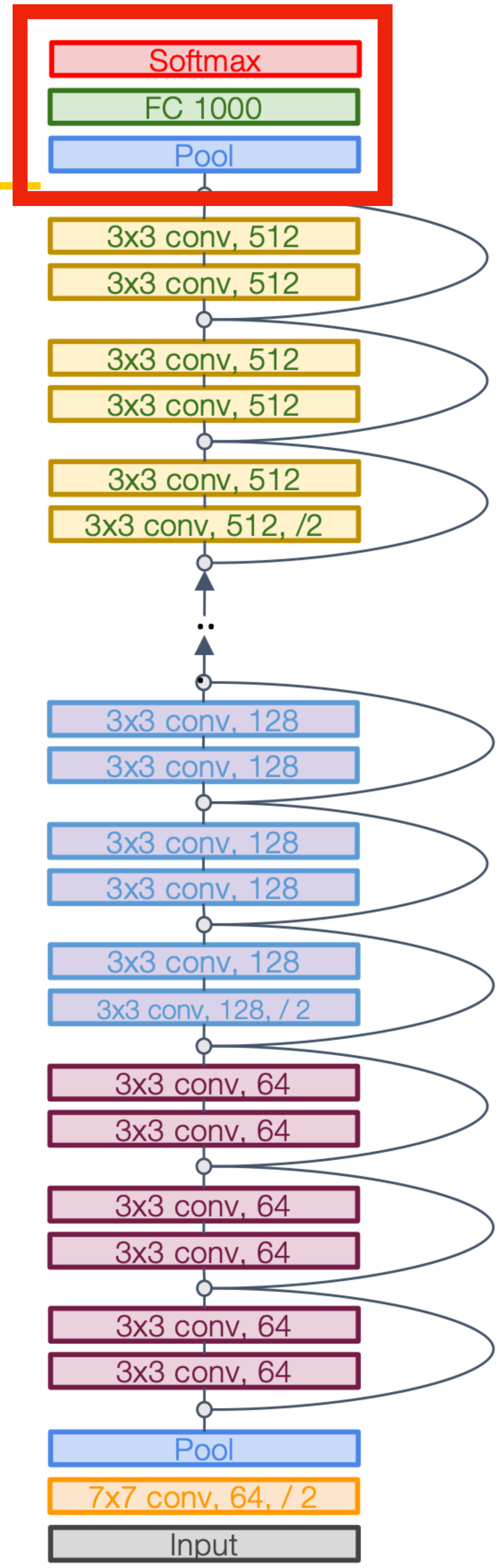
Uses the same aggressive **stem** as GoogleNet to downsample the input 4x before applying residual blocks:

	Input size		Layer				Output size				
Layer	C	H/W	Filters	Kernel	Stride	Pad	C	H/W	Memory (KB)	Params (k)	Flop (M)
Conv Poo	3	224	64	7	2	3	64	112	3136	9	118
Max-pool	64	112		3	2	1	64	56	784	0	2



Residual Networks

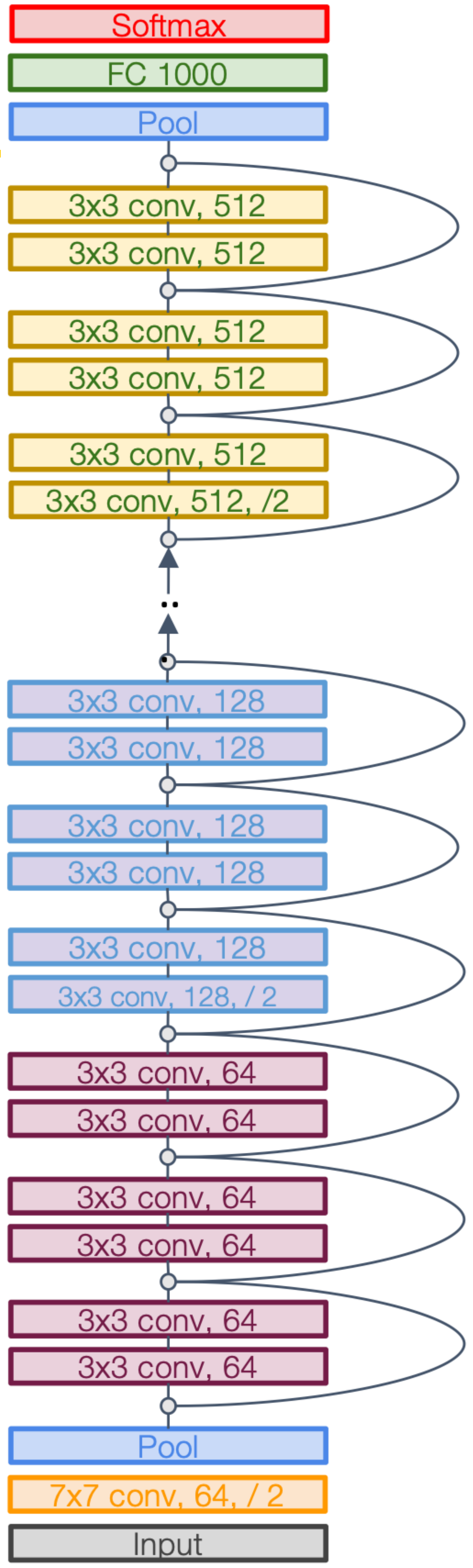
Like GoogLeNet, no big fully-connected-layers: Instead use **global average pooling** and a single linear layer at the end



Residual Networks

ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8



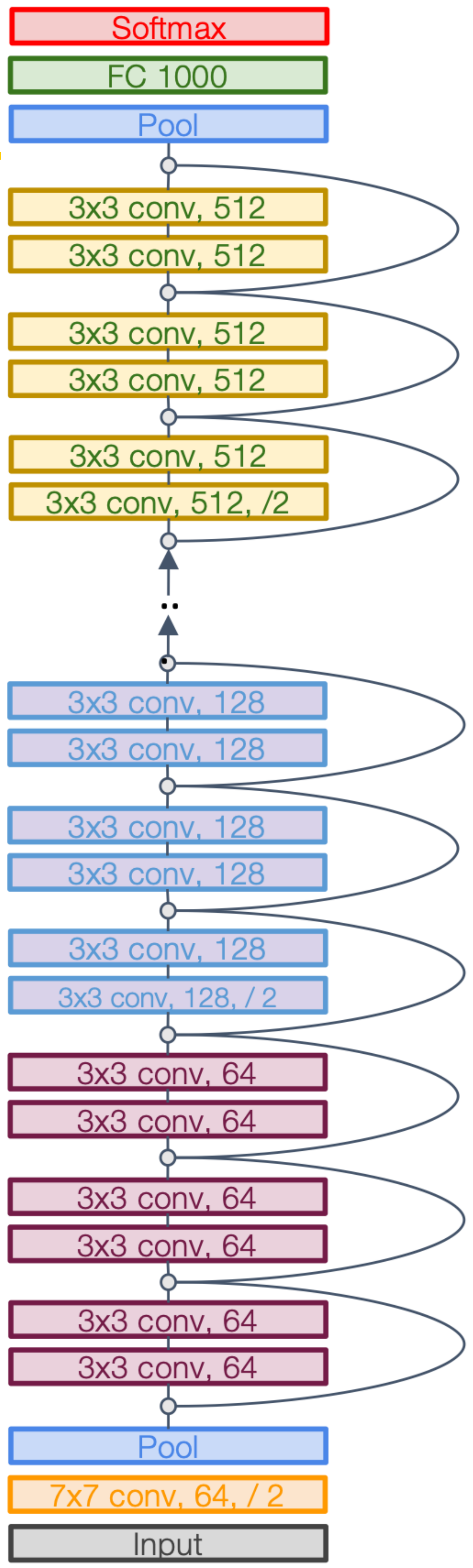
Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks

ResNet-18:
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear
ImageNet top-5 error: 10.92
GFLOP: 1.8

ResNet-34:
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear
ImageNet top-5 error: 8.58
GFLOP: 3.6

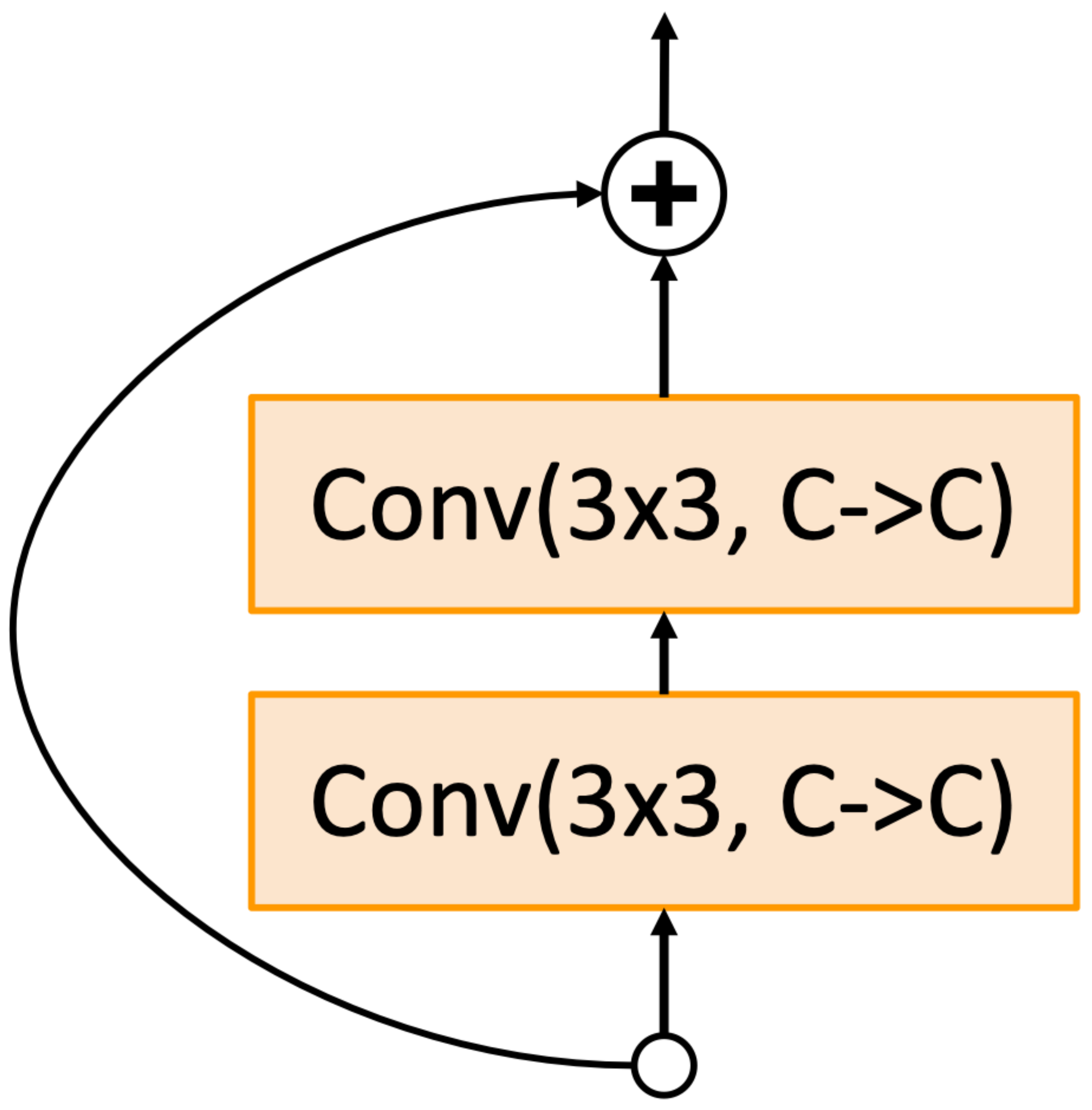
VGG-16:
ImageNet top-5 error: 9.62
GFLOP: 13.6



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Error rates are 224x224 single-crop testing, reported by torchvision

Residual Networks: Basic Block



“Basic”
Residual block

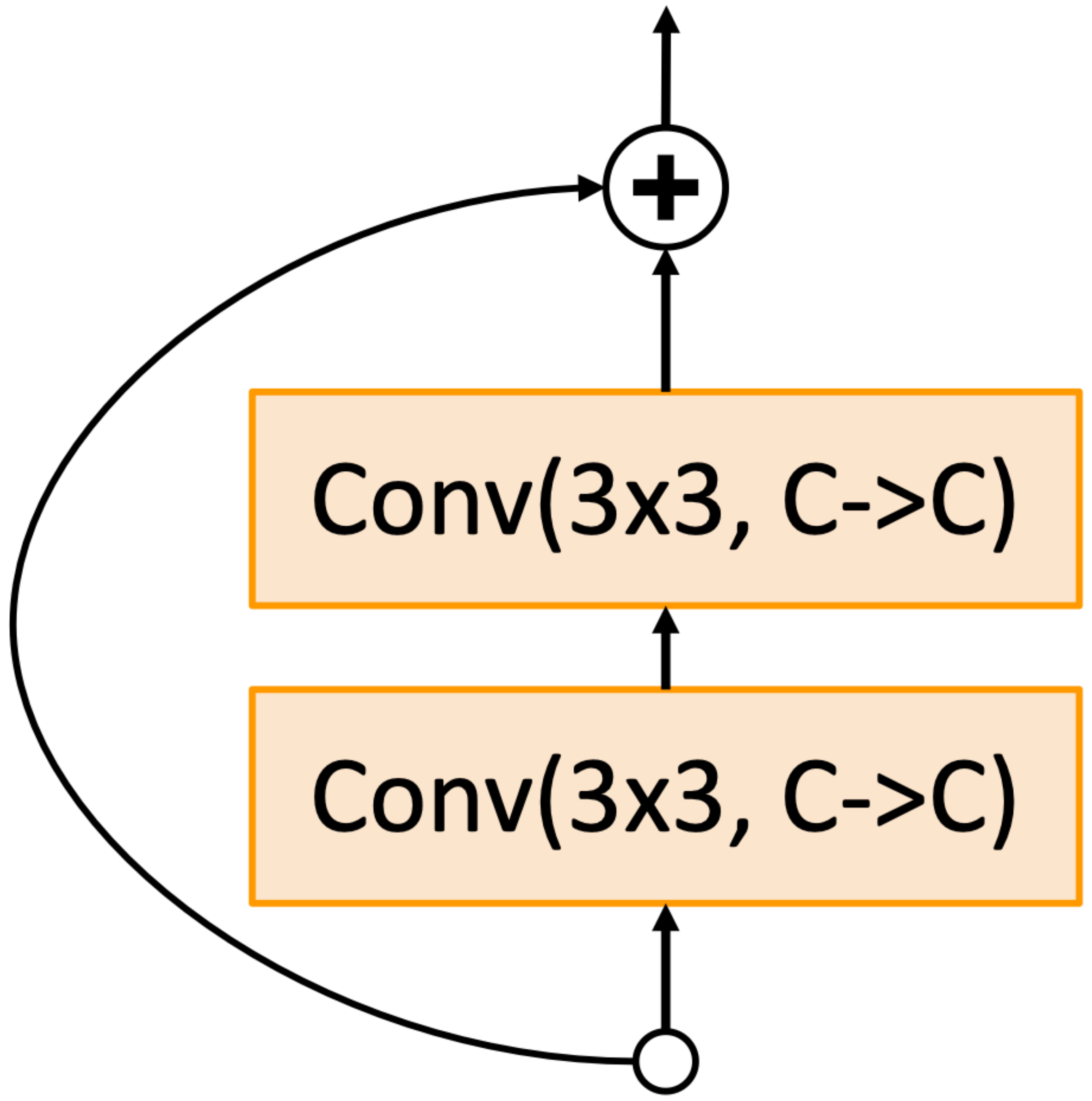
FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

Total FLOPs:

$18HWC^2$

Residual Networks: Bottleneck Block



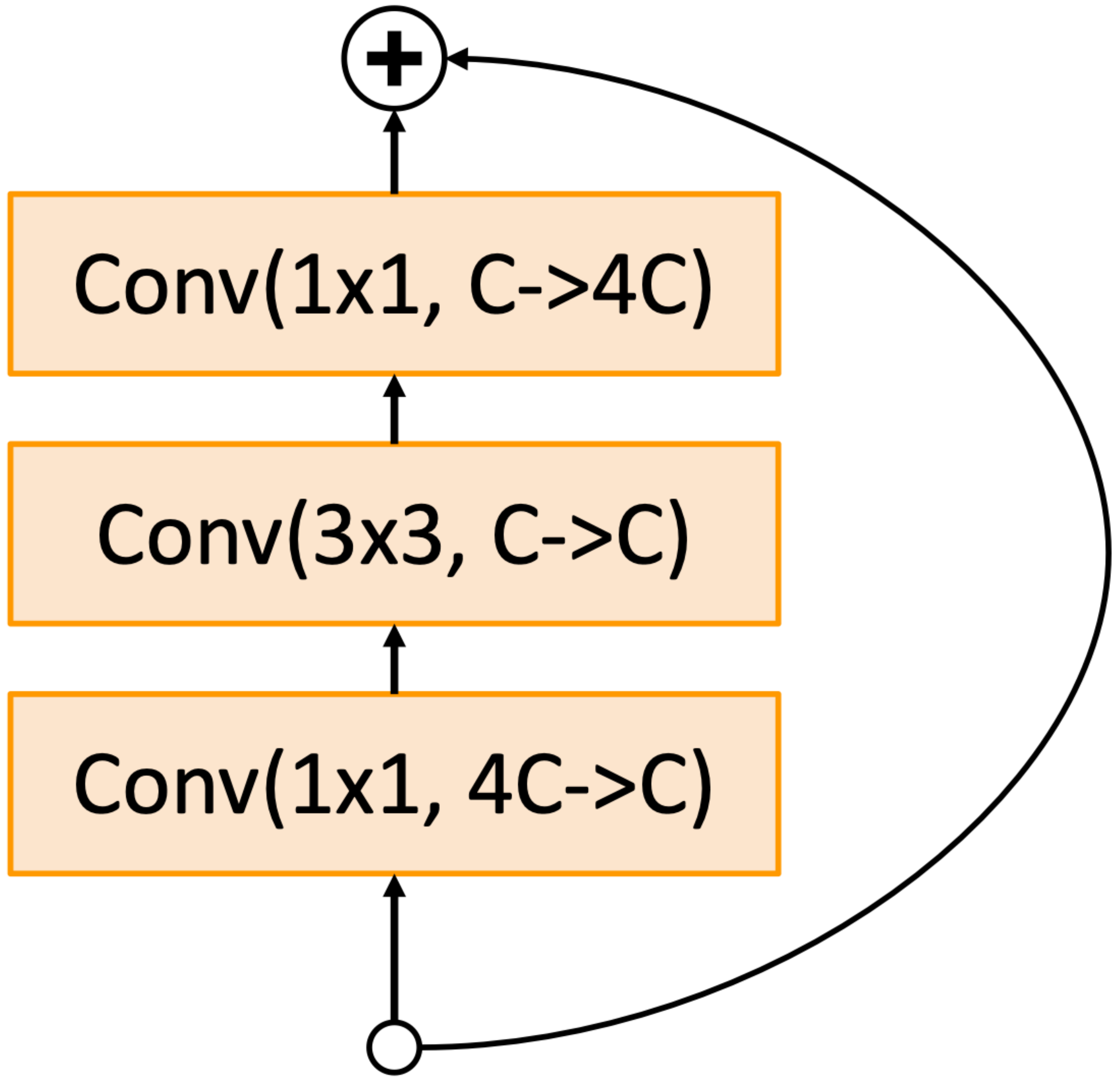
"Basic"
Residual block

FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

Total FLOPs:

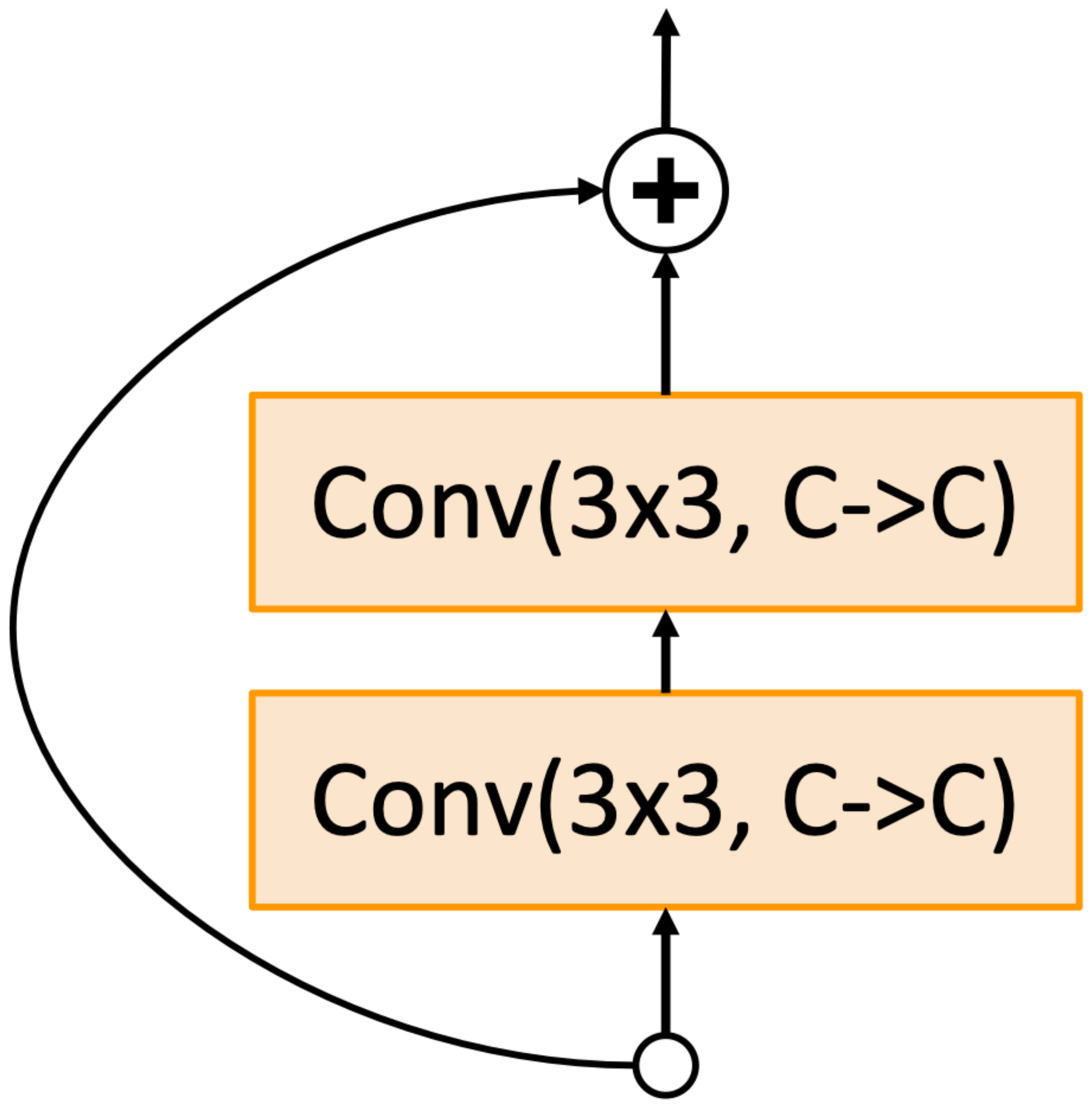
$18HWC^2$



"Bottleneck"
Residual block

Residual Networks: Bottleneck Block

More layers, less computational cost!



"Basic"
Residual block

FLOPs: $9HWC^2$

FLOPs: $9HWC^2$

Total FLOPs:

$18HWC^2$

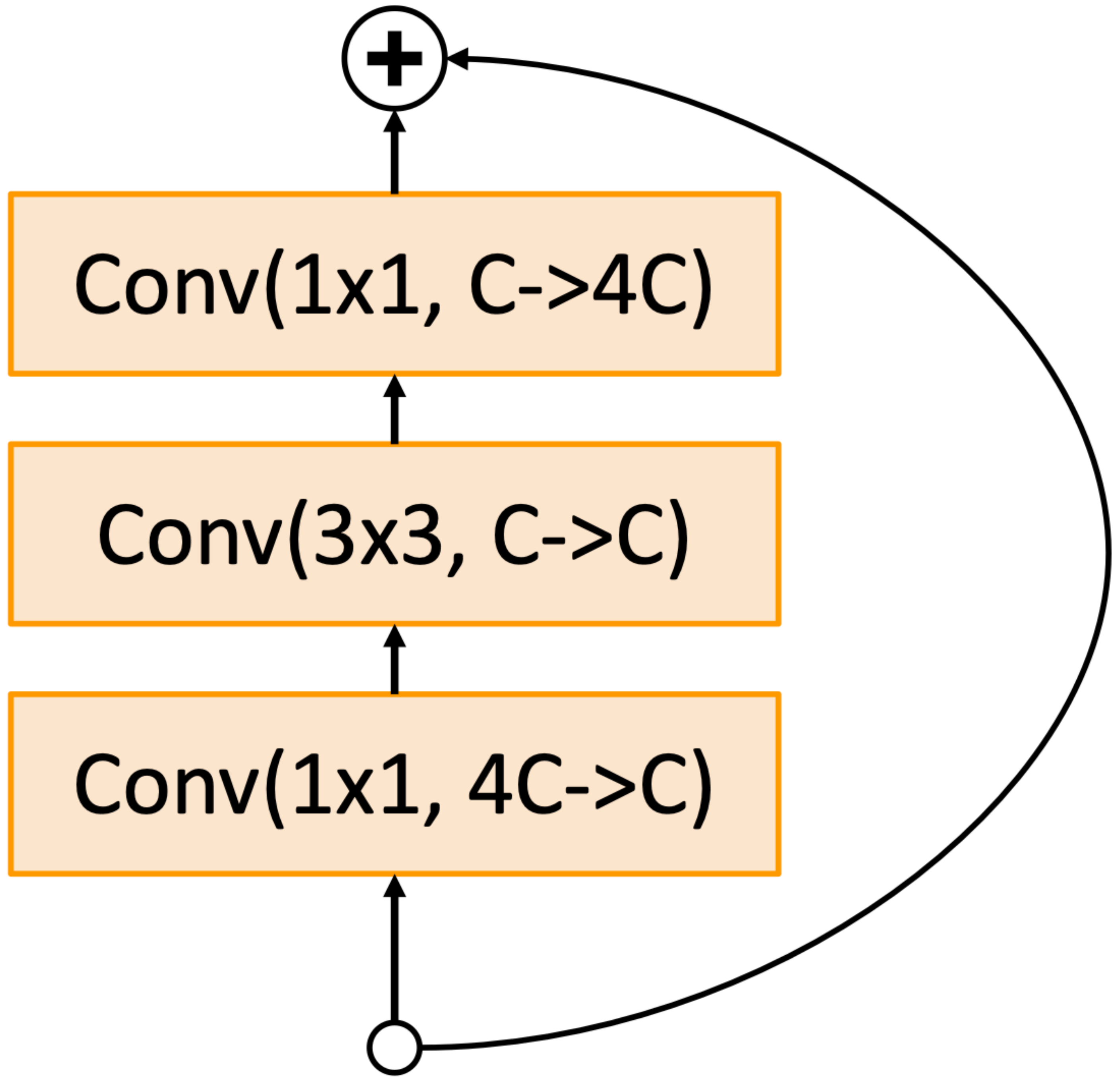
FLOPs: $4HWC^2$

FLOPs: $9HWC^2$

FLOPs: $4HWC^2$

Total FLOPs:

$17HWC^2$



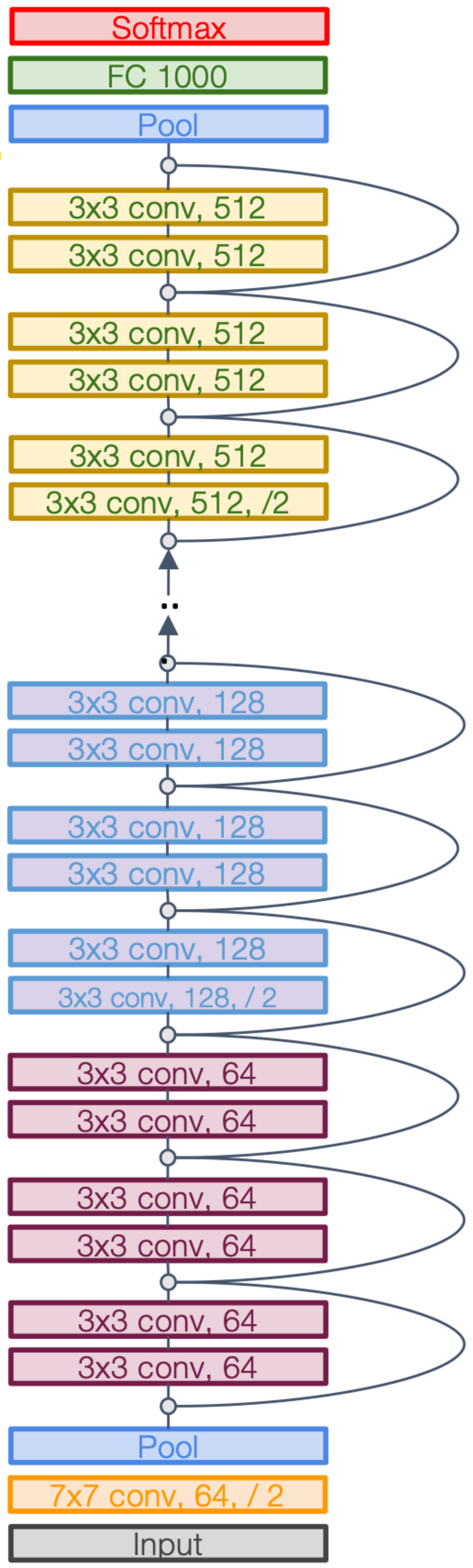
"Bottleneck"
Residual block

Residual Networks

ResNet-50 is the same as ResNet-34, but replaces Basic blocks with Bottleneck Blocks. This is a great baseline architecture for many tasks even today!

Deeper ResNet-101 and ResNet-152 models are more accurate, but also more computationally heavy

			Stage 1		Stage 2		Stage 3		Stage 4				
	Block type	Stem layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	Blocks	Layers	FC Layers	GFLOP	ImageNet
ResNet-18	Basic	1	2	4	2	4	2	4	2	4	1	1.8	10.92
ResNet-34	Basic	1	3	6	4	8	6	12	3	6	1	3.6	8.58
ResNet-50	Bottle	1	3	9	4	12	6	18	3	9	1	3.8	7.13
ResNet-101	Bottle	1	3	9	4	12	23	69	3	9	1	7.6	6.44
ResNet-152	Bottle	1	3	9	8	24	36	108	3	9	1	11.3	5.94



Residual Networks

- Able to train very deep networks
- Deeper networks do better than shallow networks (as expected)
- Swept 1st place in all ILSVRC and COCO 2015 competitions
- Still widely used today

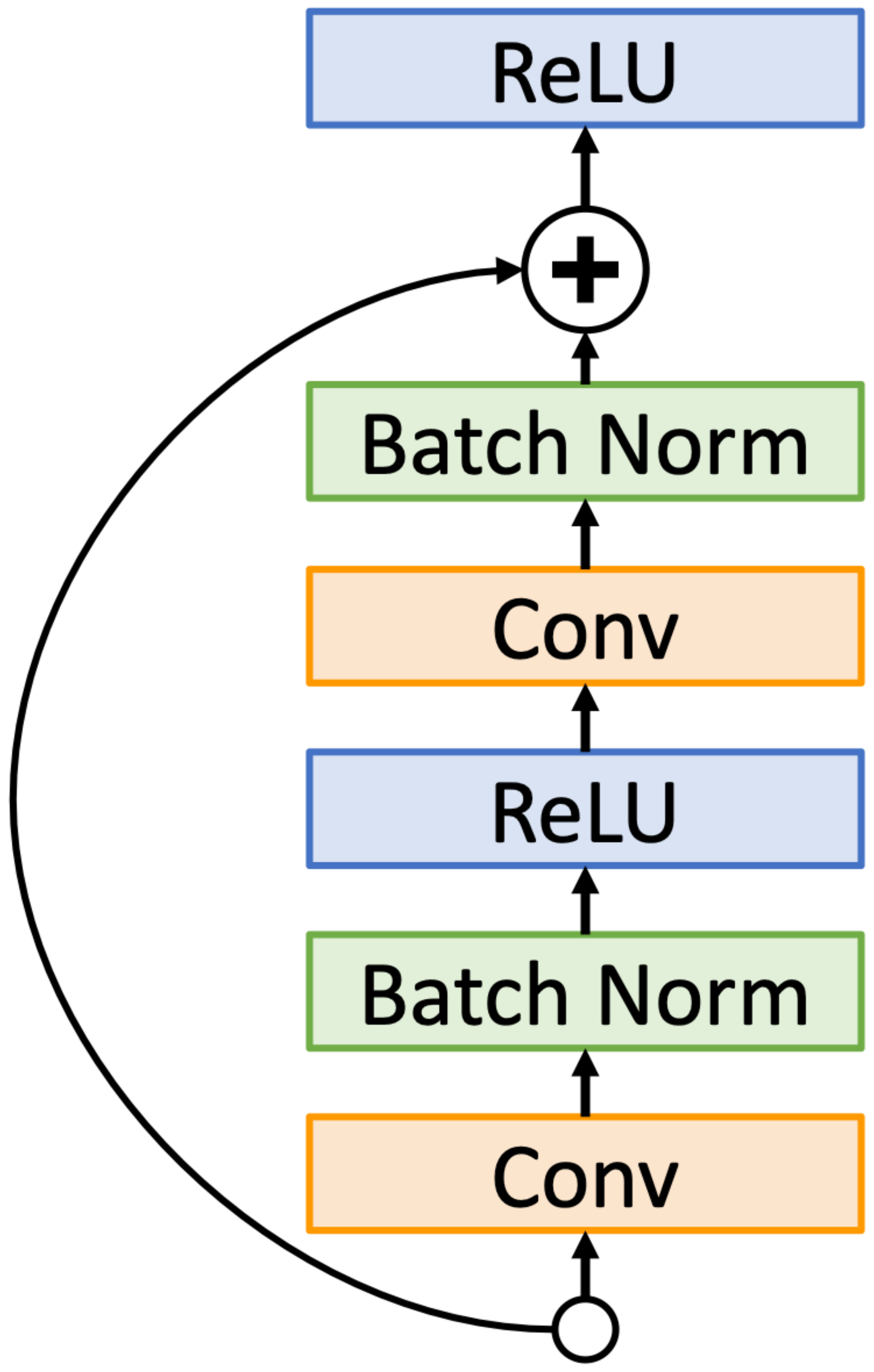
MSRA @ ILSVRC & COCO 2015 Competitions

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

He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Improving Residual Networks: Block Design

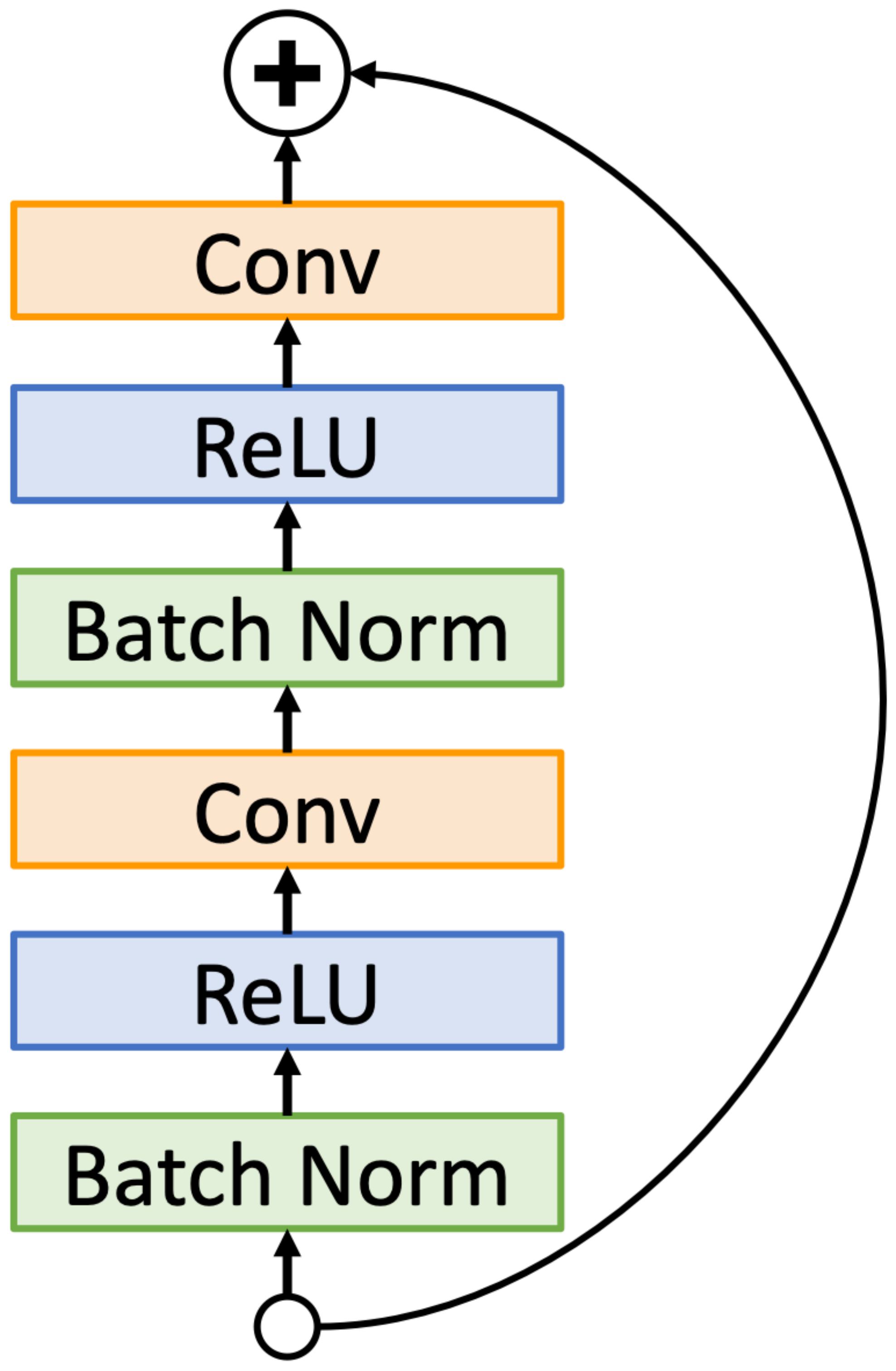
Original ResNet block



Note ReLU **after** residual:

Cannot actually learn identity function since outputs are nonnegative!

"Pre-Activation" ResNet Block



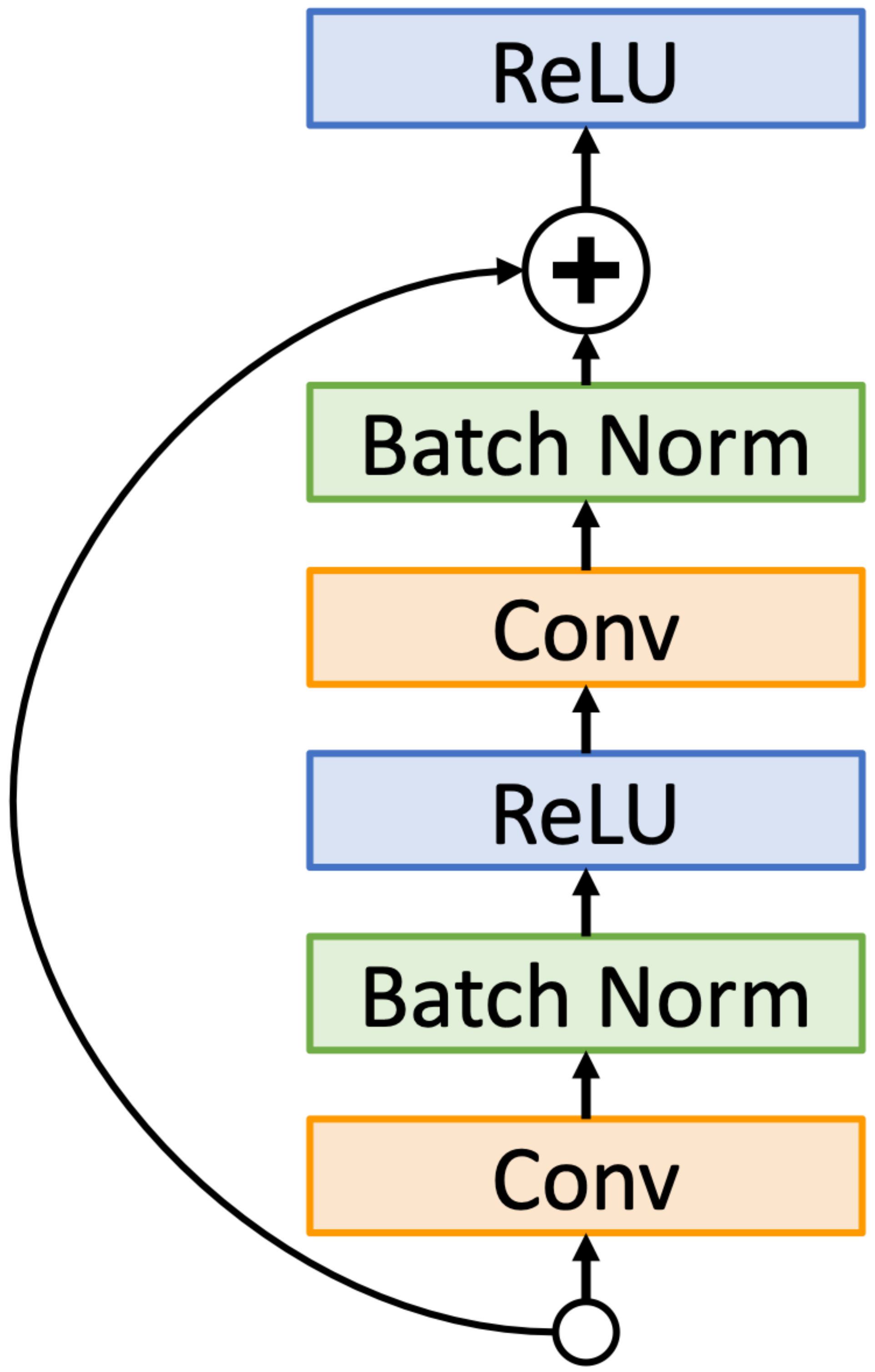
Note ReLU **inside** residual:

Can learn identity function by setting Conv weights to zero

He et al, "Identity mappings in deep residual networks", ECCV 2016

Improving Residual Networks: Block Design

Original ResNet block



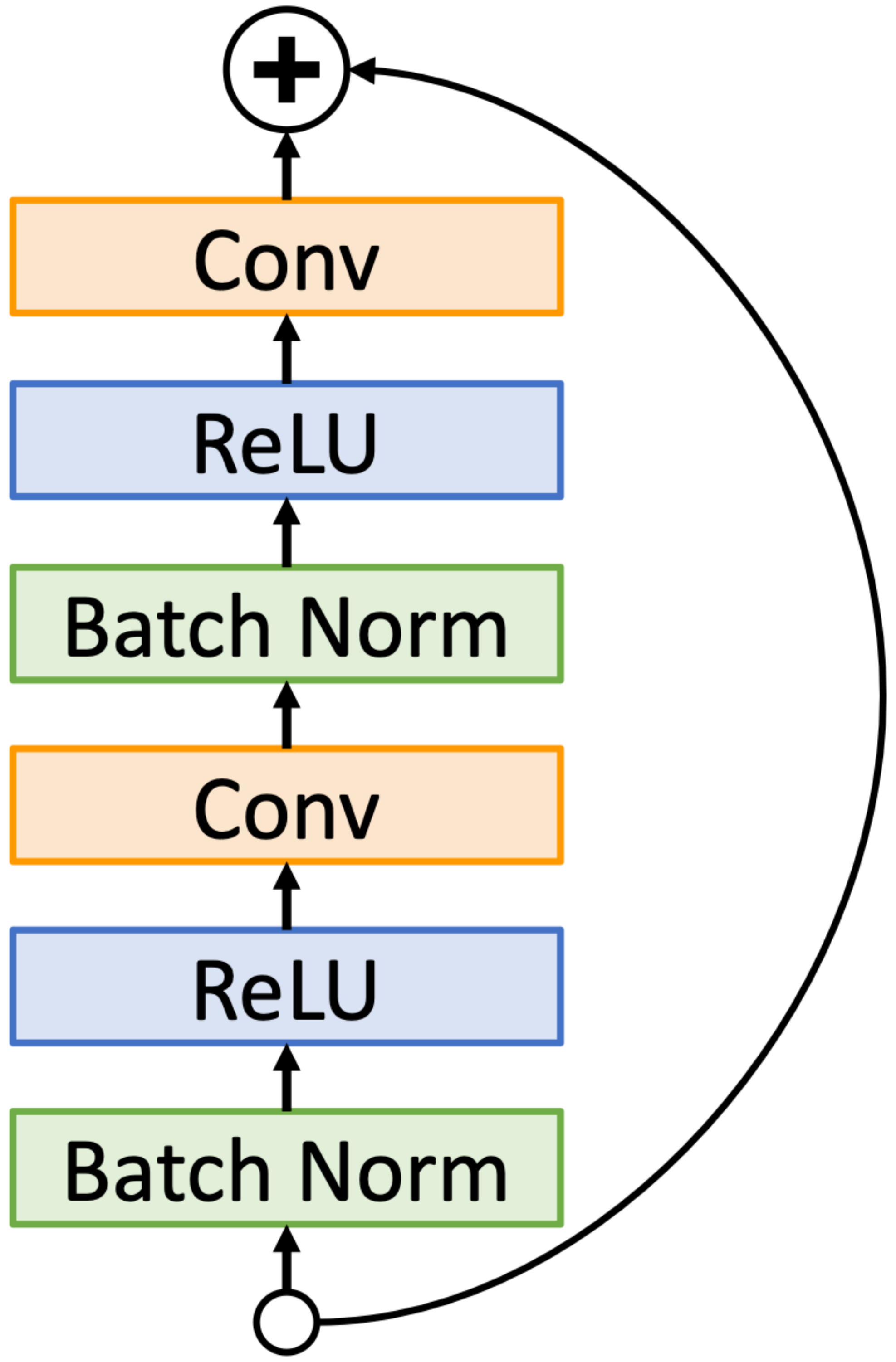
Slight improvement in accuracy (ImageNet top-1 error)

ResNet-152: 21.3 vs **21.1**

ResNet-200: 21.8 vs **20.7**

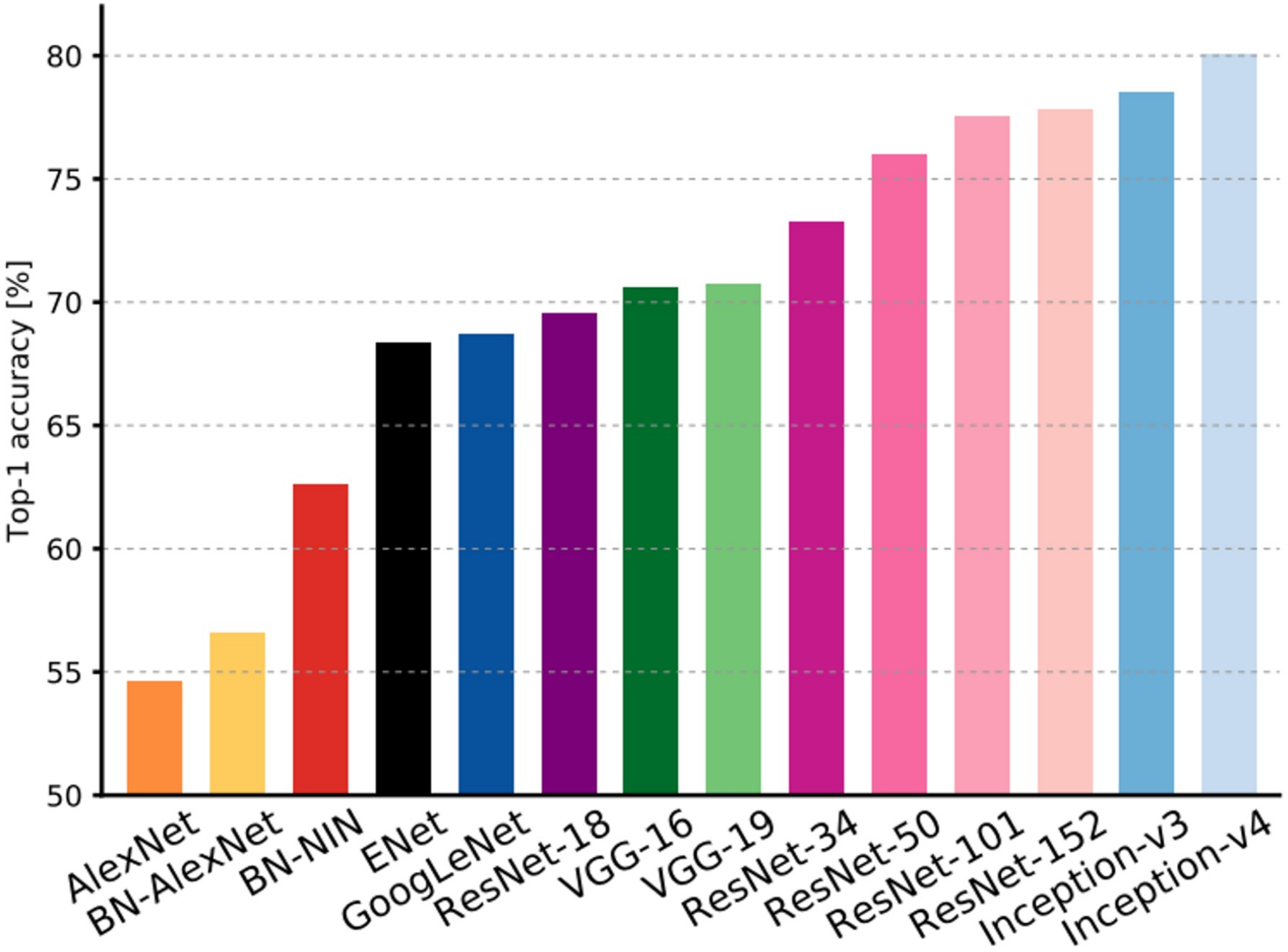
Not actually used that much in practice

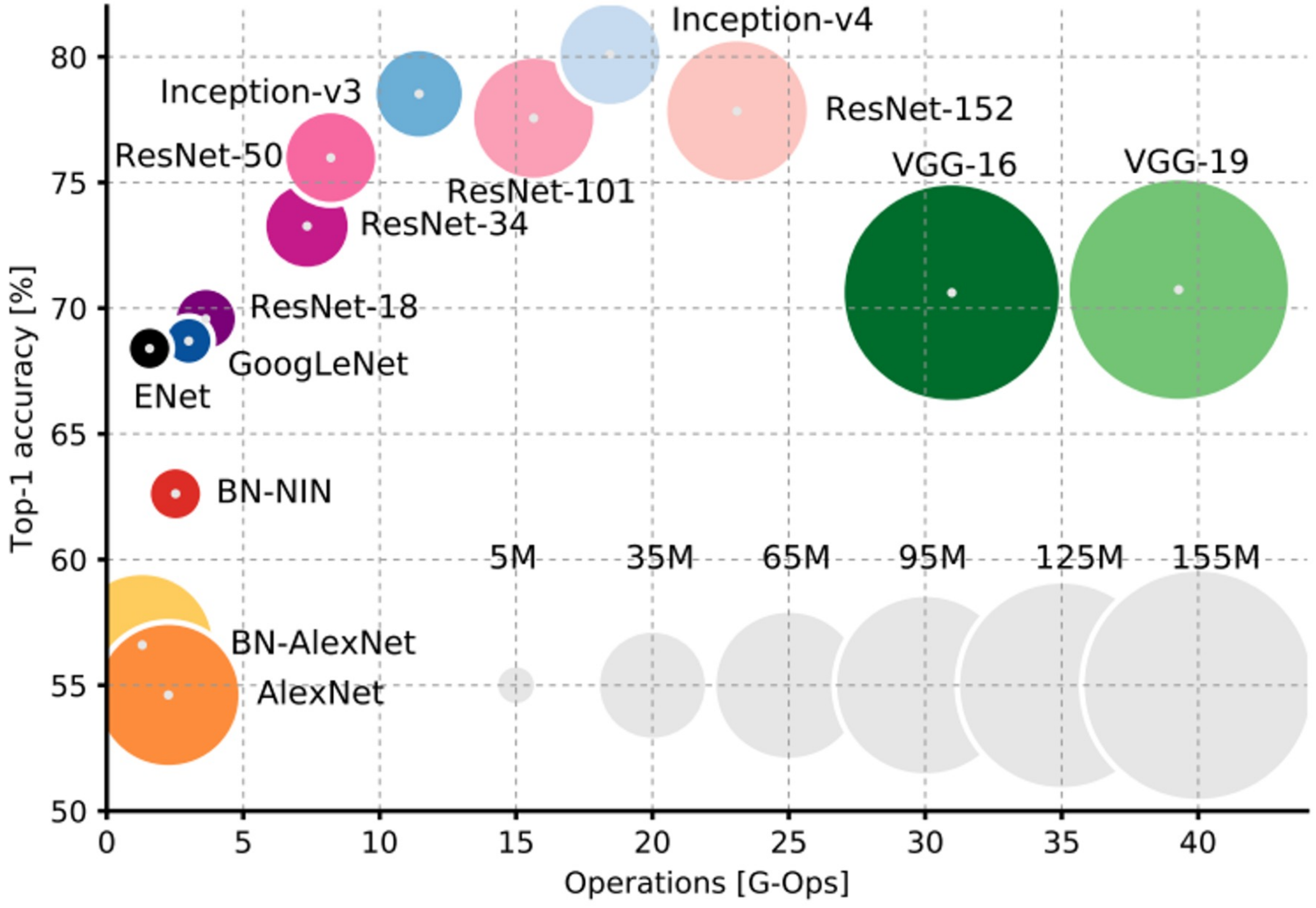
"Pre-Activation" ResNet Block



He et al, "Deep Residual Learning for Image Recognition", CVPR 2016

Comparing Complexity

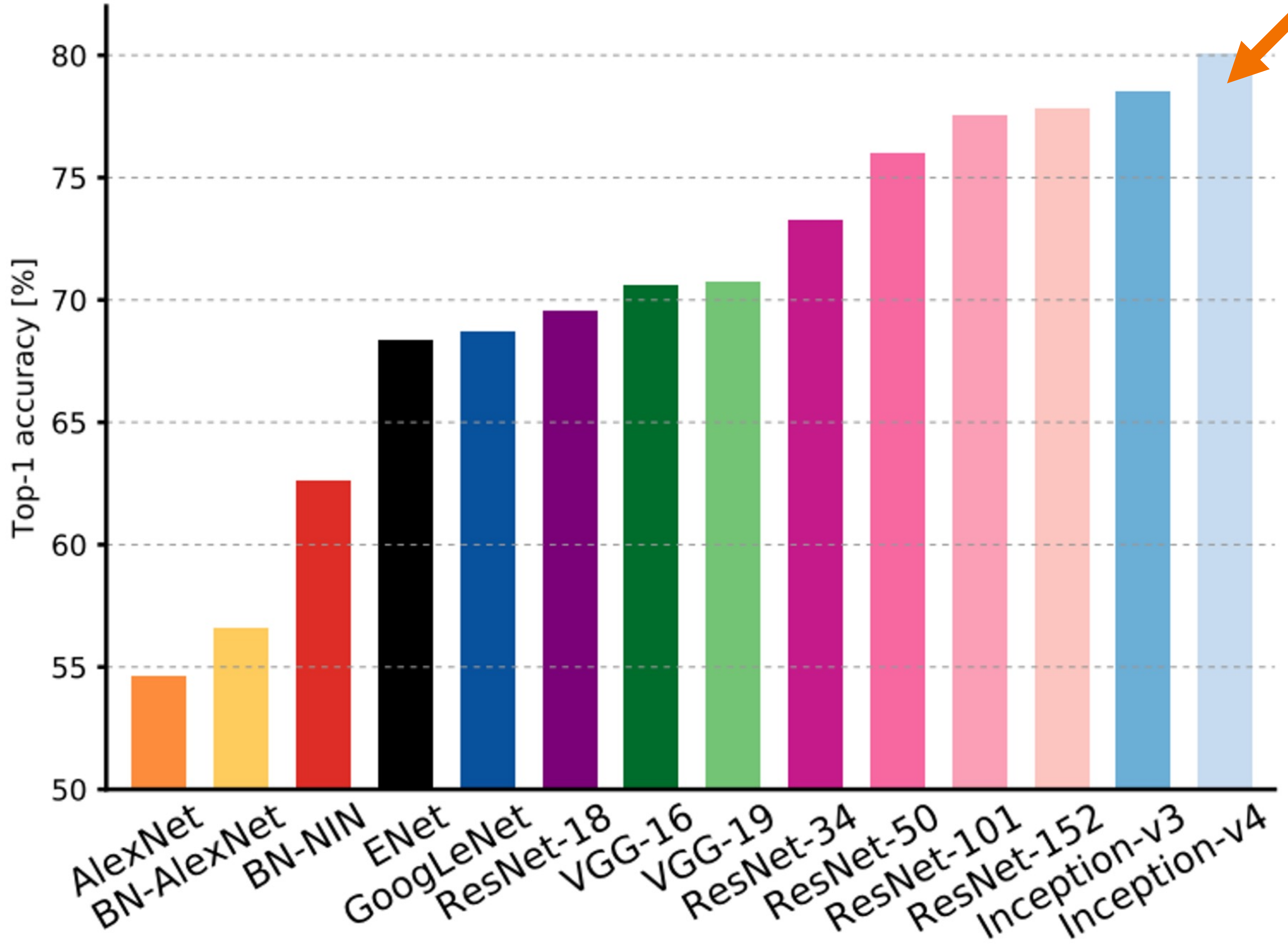


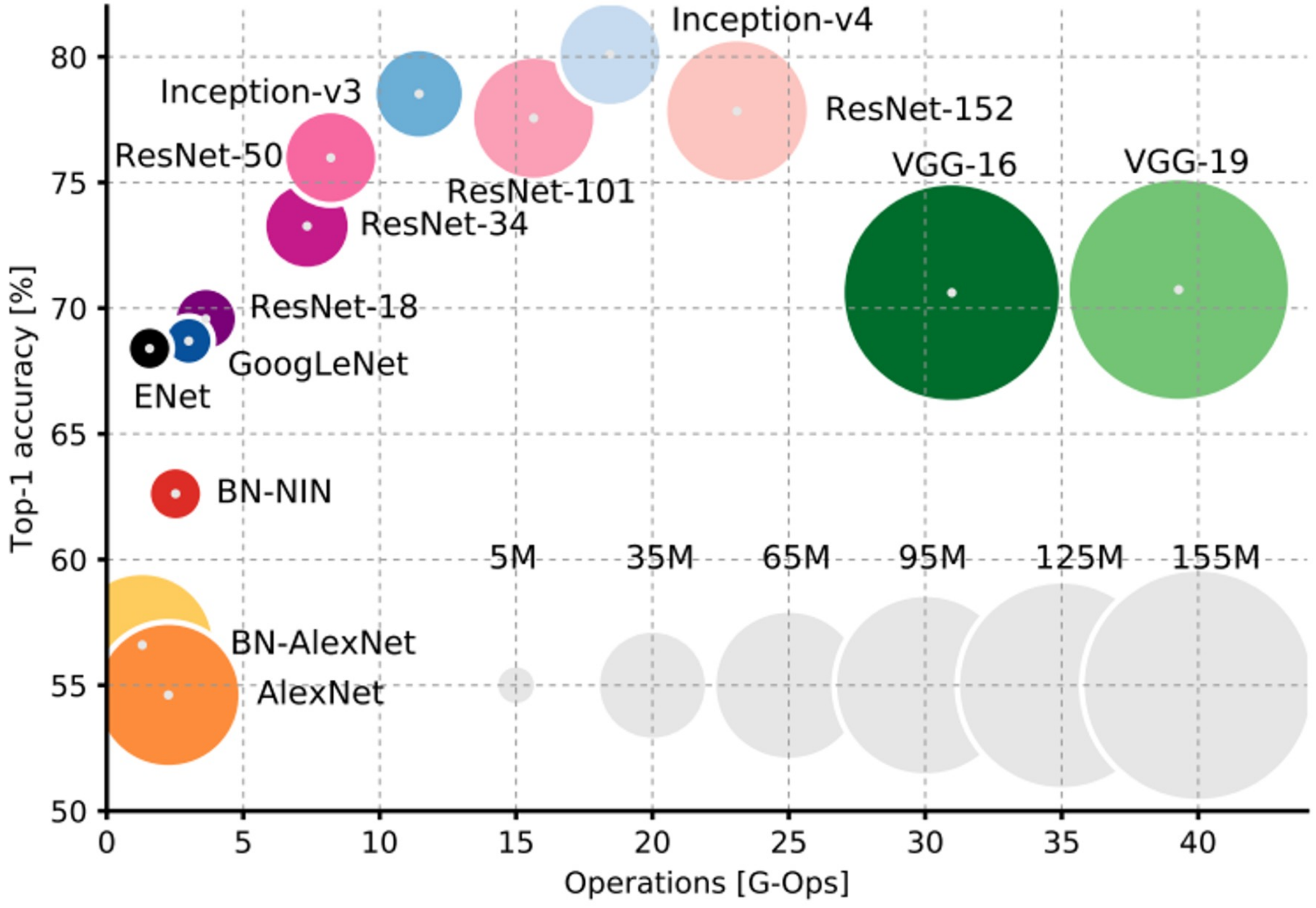


Canziani et al, "An analysis of deep neural network models for practical applications", 2017

Comparing Complexity

Inception-v4: ResNet + Inception!



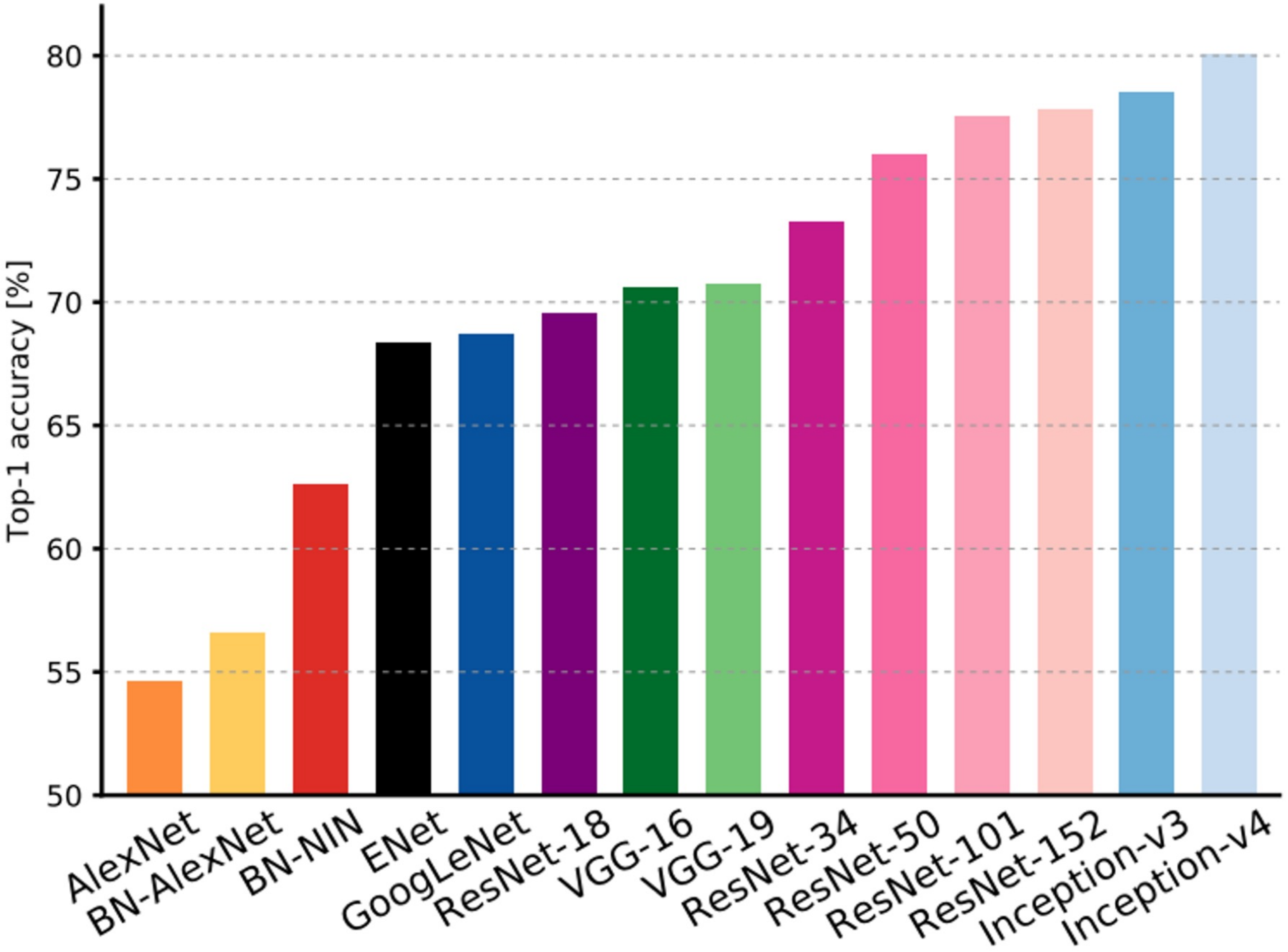


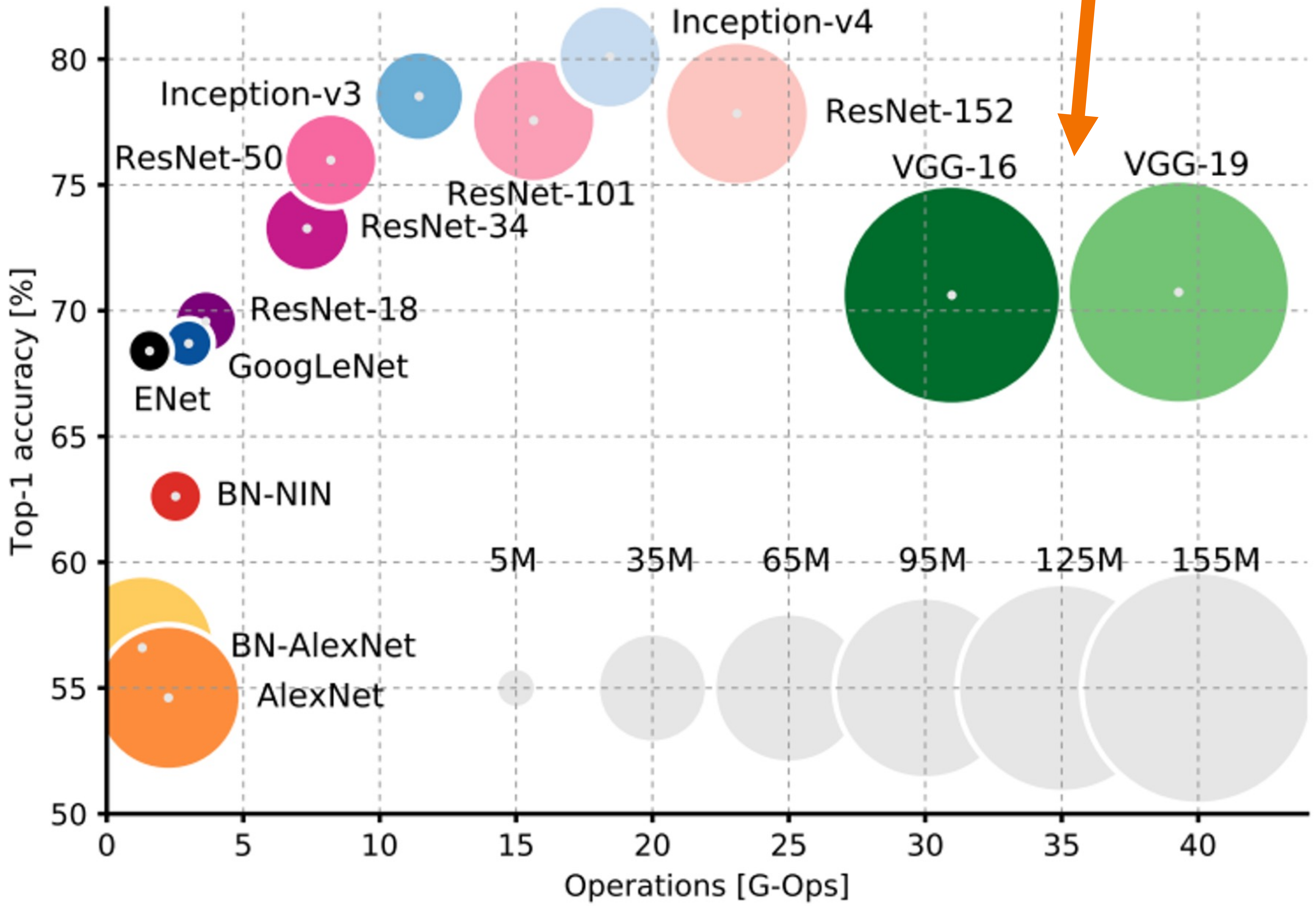
Comparing Complexity

VGG:

Highest memory,

most operations

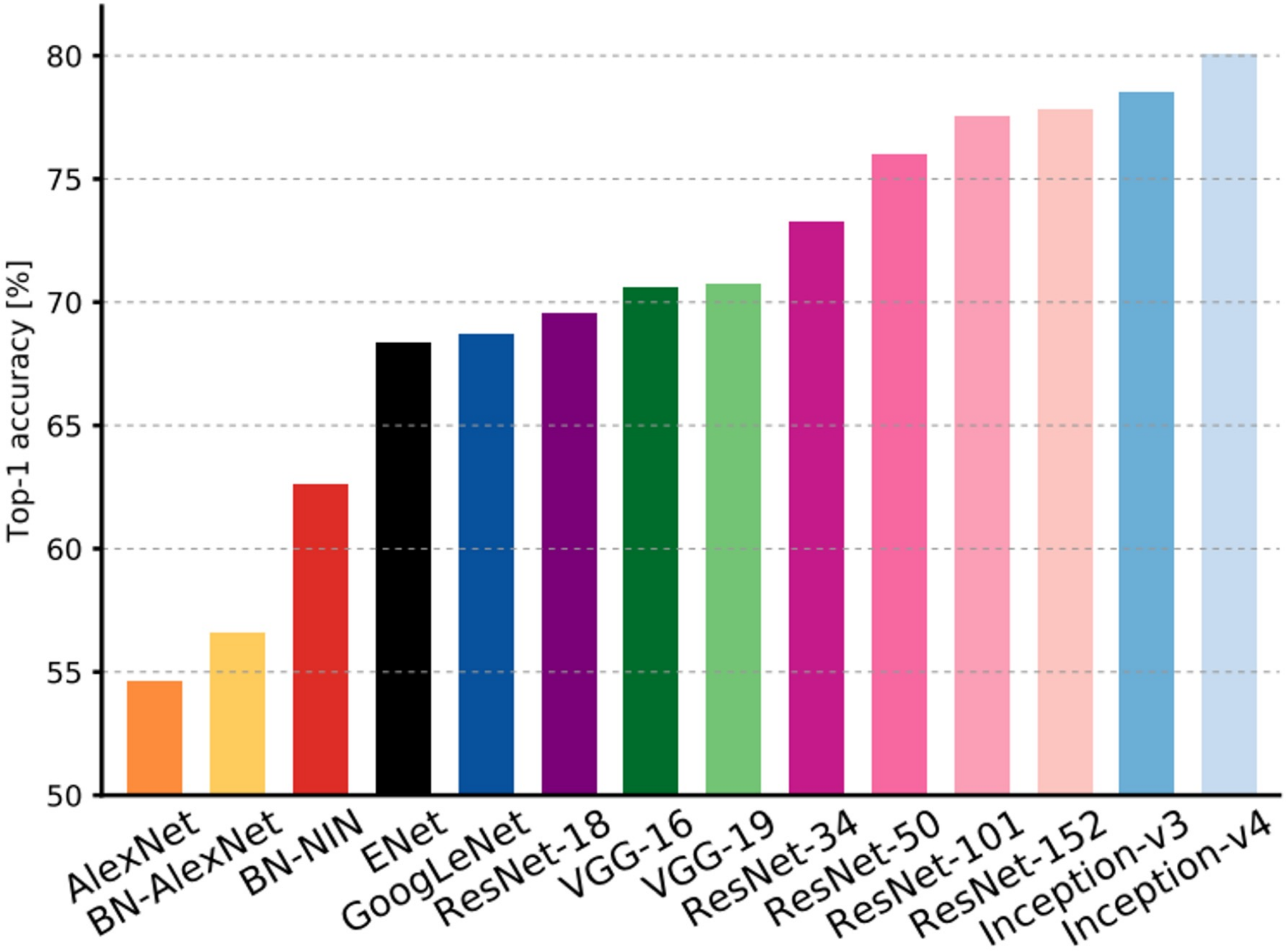


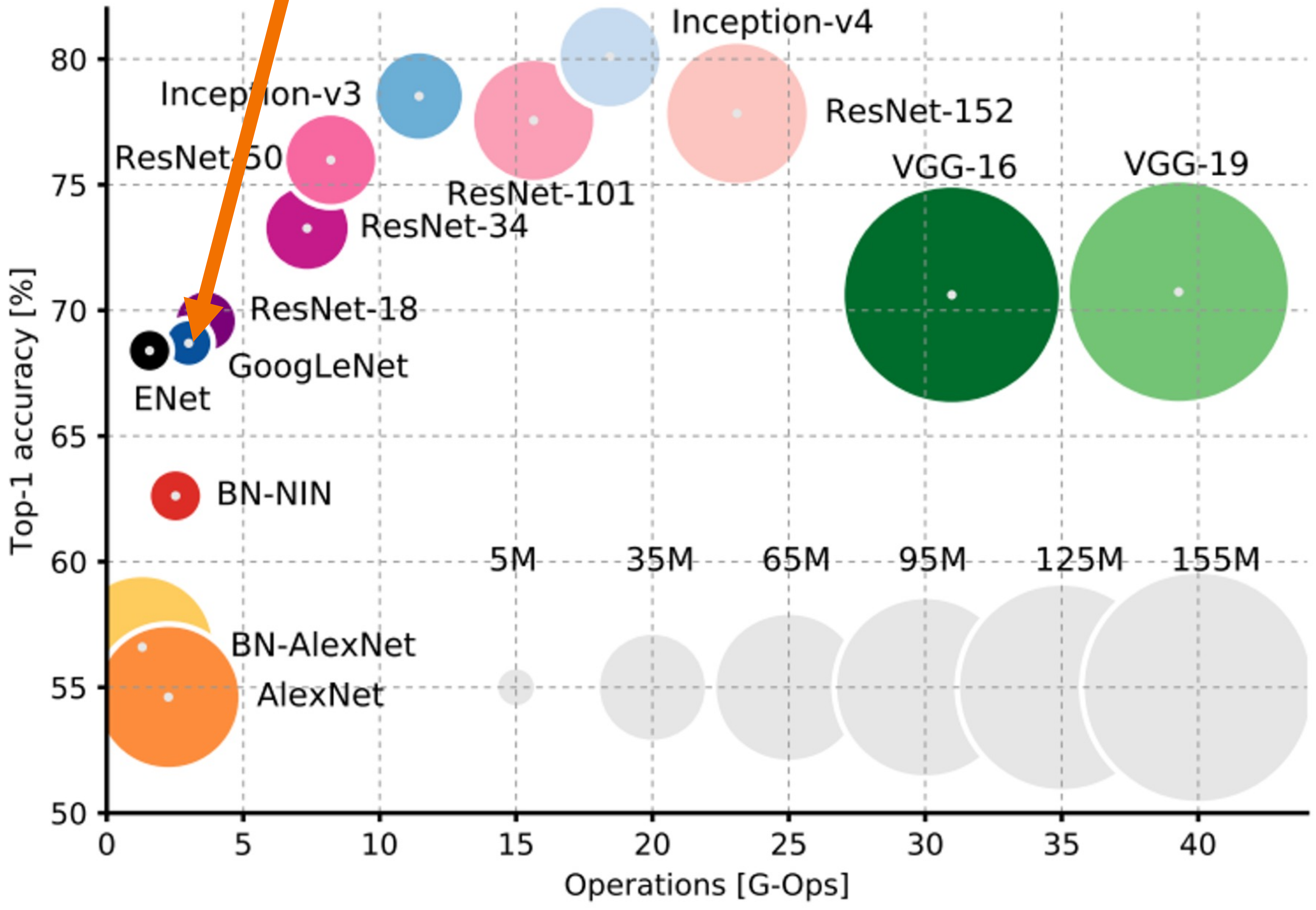


Comparing Complexity

GoogLeNet:

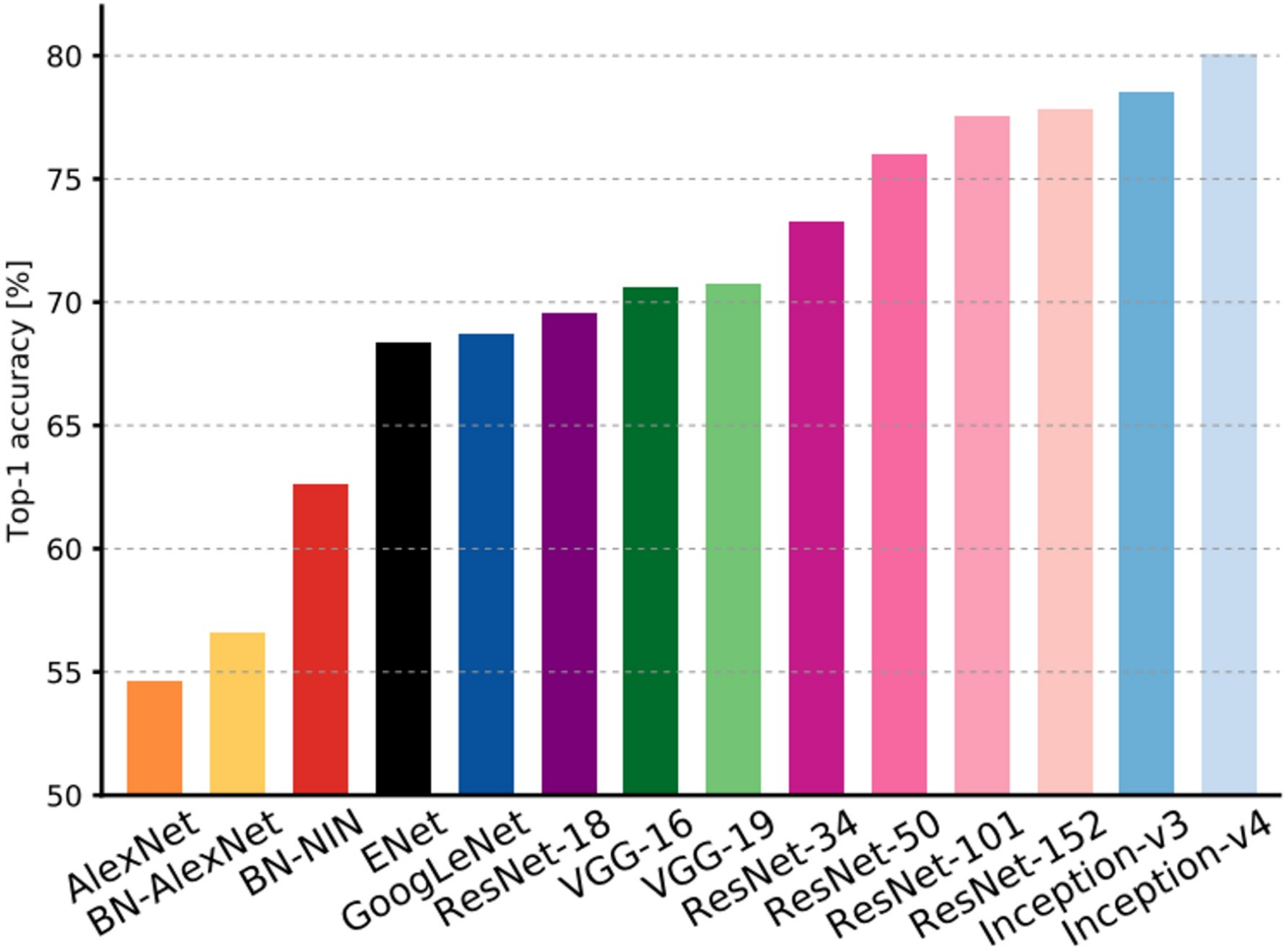
Very efficient!

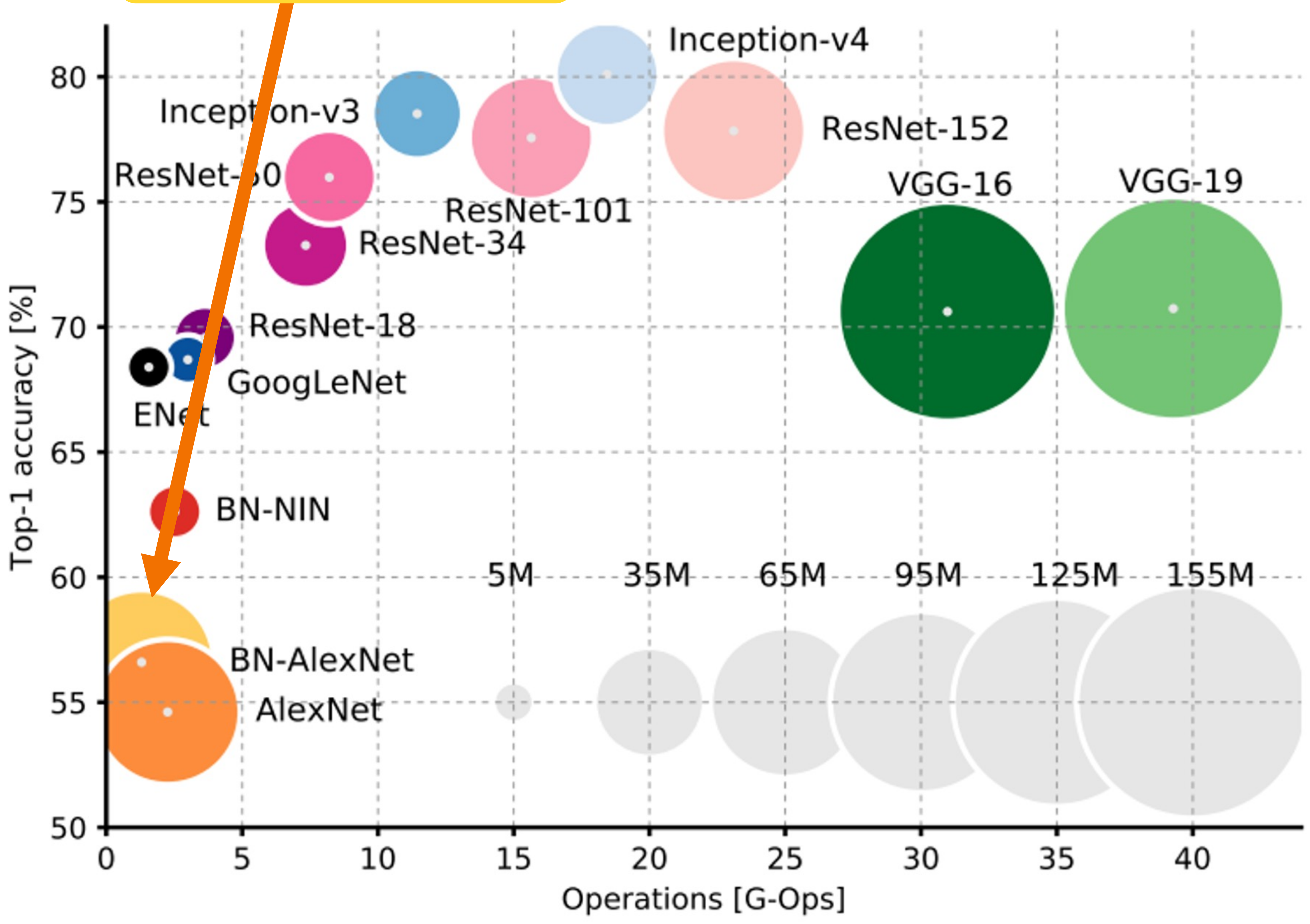




Comparing Complexity

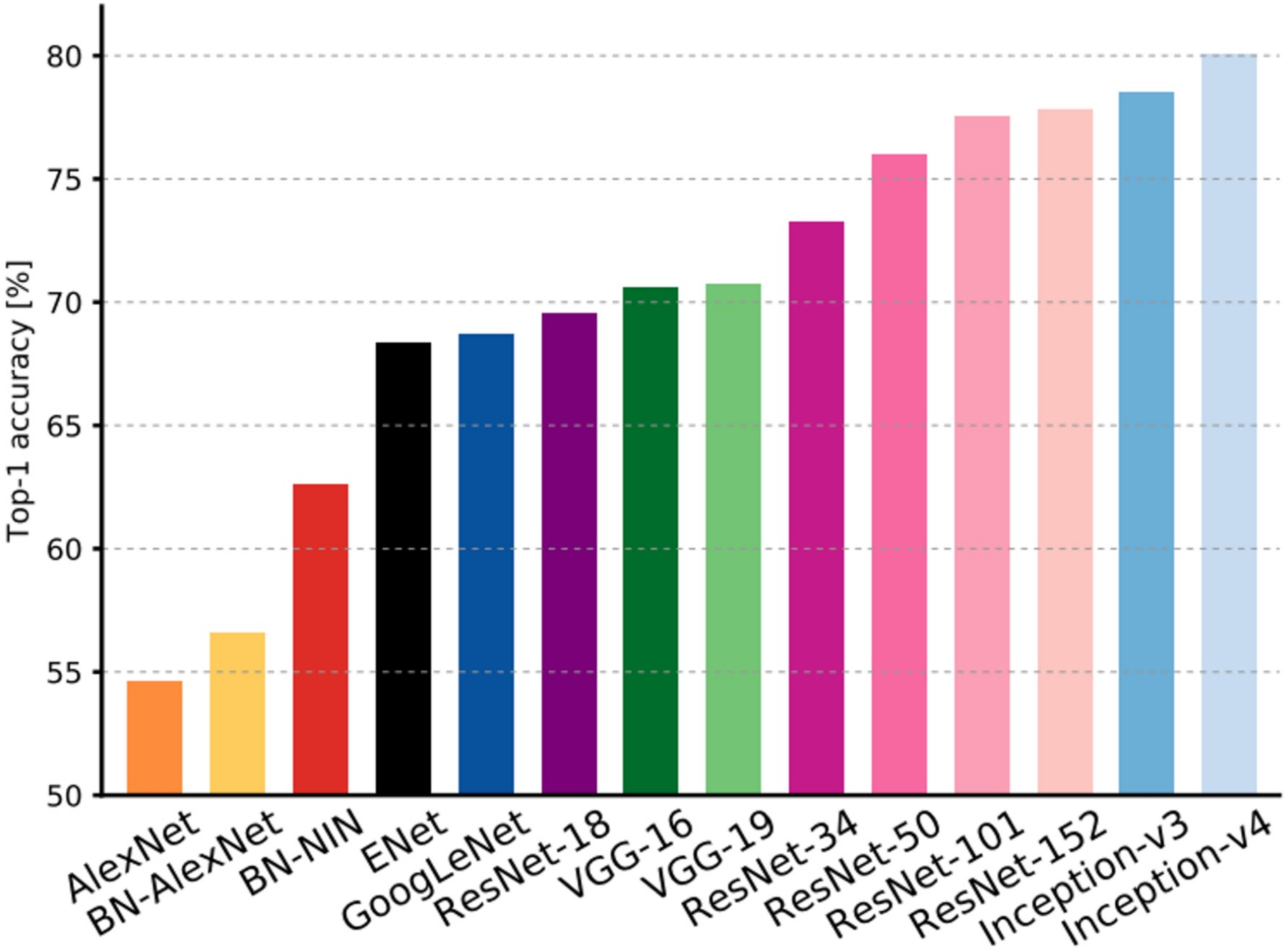
AlexNet: Low compute, lots of parameters

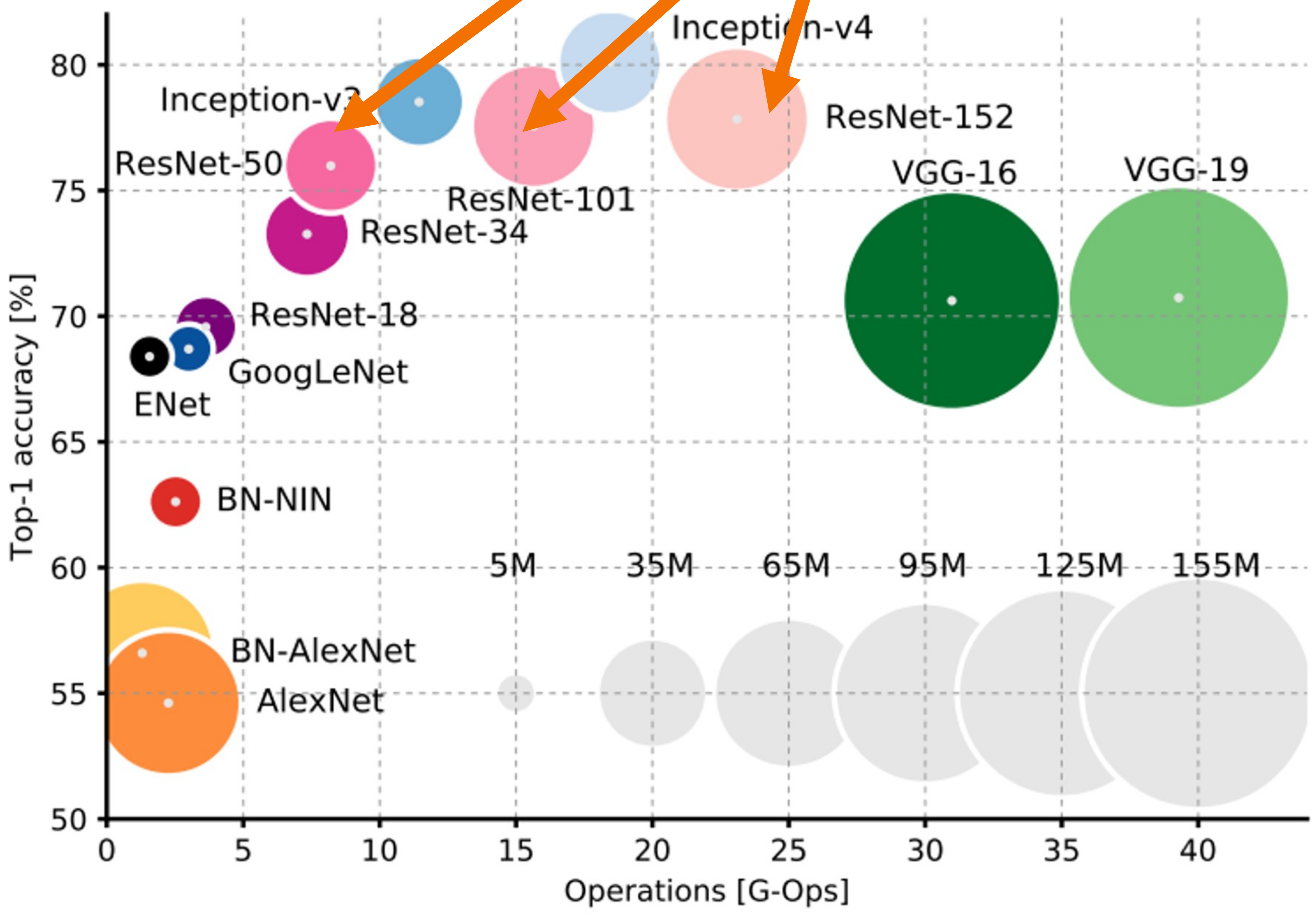




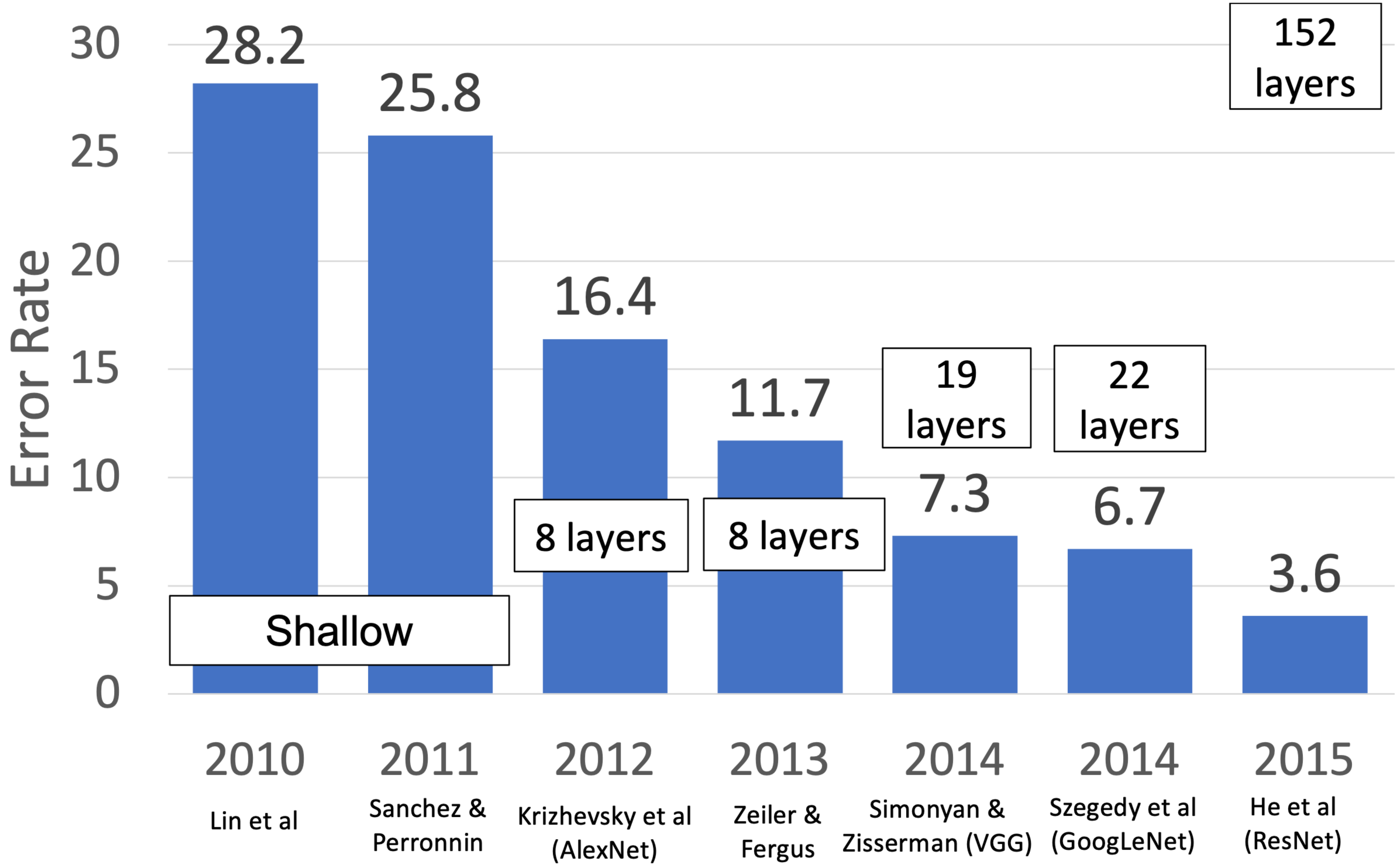
Comparing Complexity

ResNet: Simple design, moderate efficiency, high accuracy





ImageNet Classification Challenge



CNN architectures have continued to evolve!

So far: Image Classification

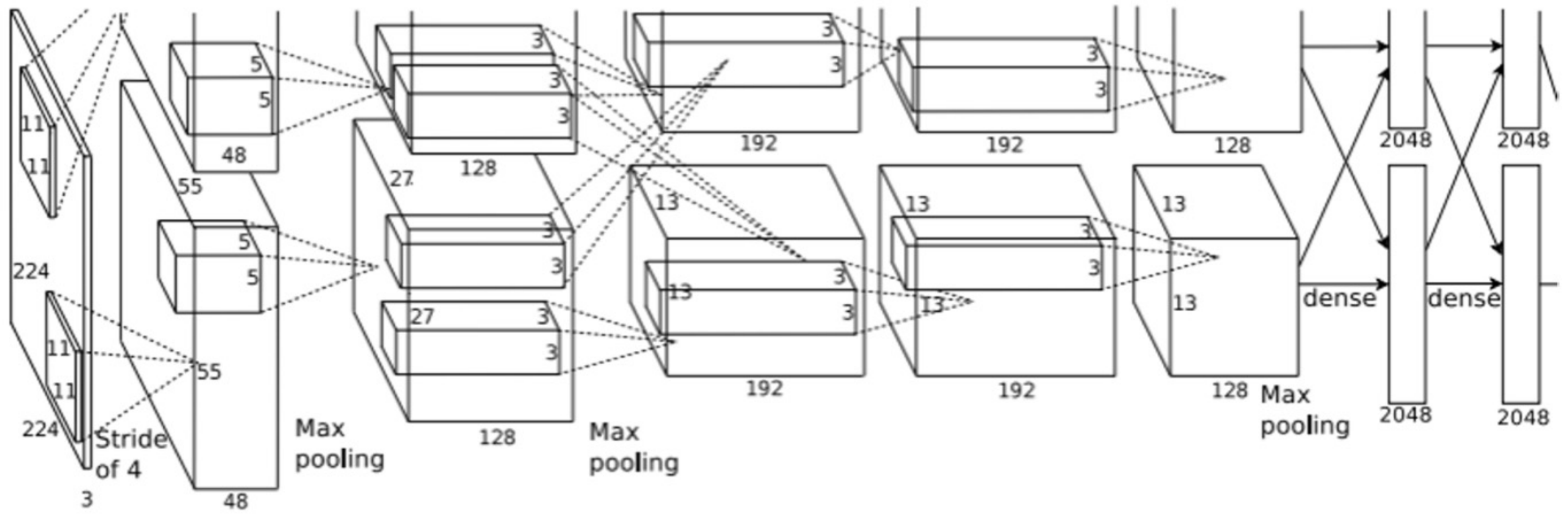


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:

4096

Fully connected:

4096 to 10

Chocolate Pretzels

Granola Bar

Potato Chips

Water Bottle

Popcorn

Computer Vision Tasks

Classification

“Chocolate Pretzels”

No spatial extent

Semantic Segmentation

Chocolate Pretzels,

Shelf

No objects, just pixels

Object Detection

Instance Segmentation

Flipz, Hershey’s, Keese's

Multiple objects

P2 due date: Feb.22, 2024

Today: Object Detection (used in P2)

Classification

"Chocolate Pretzels"

No spatial extent

Semantic Segmentation

Chocolate Pretzels,

Shelf

No objects, just pixels

Object Detection

Flipz, Hershey's, Keese's

Instance Segmentation

Multiple objects

Object Detection: Task definition

Input: Single RGB image

Output: A set of detected objects;

For each object predict:

1. Category label (from a fixed set of labels)
2. Bounding box (four numbers: x, y, width, height)

Object Detection: Challenges

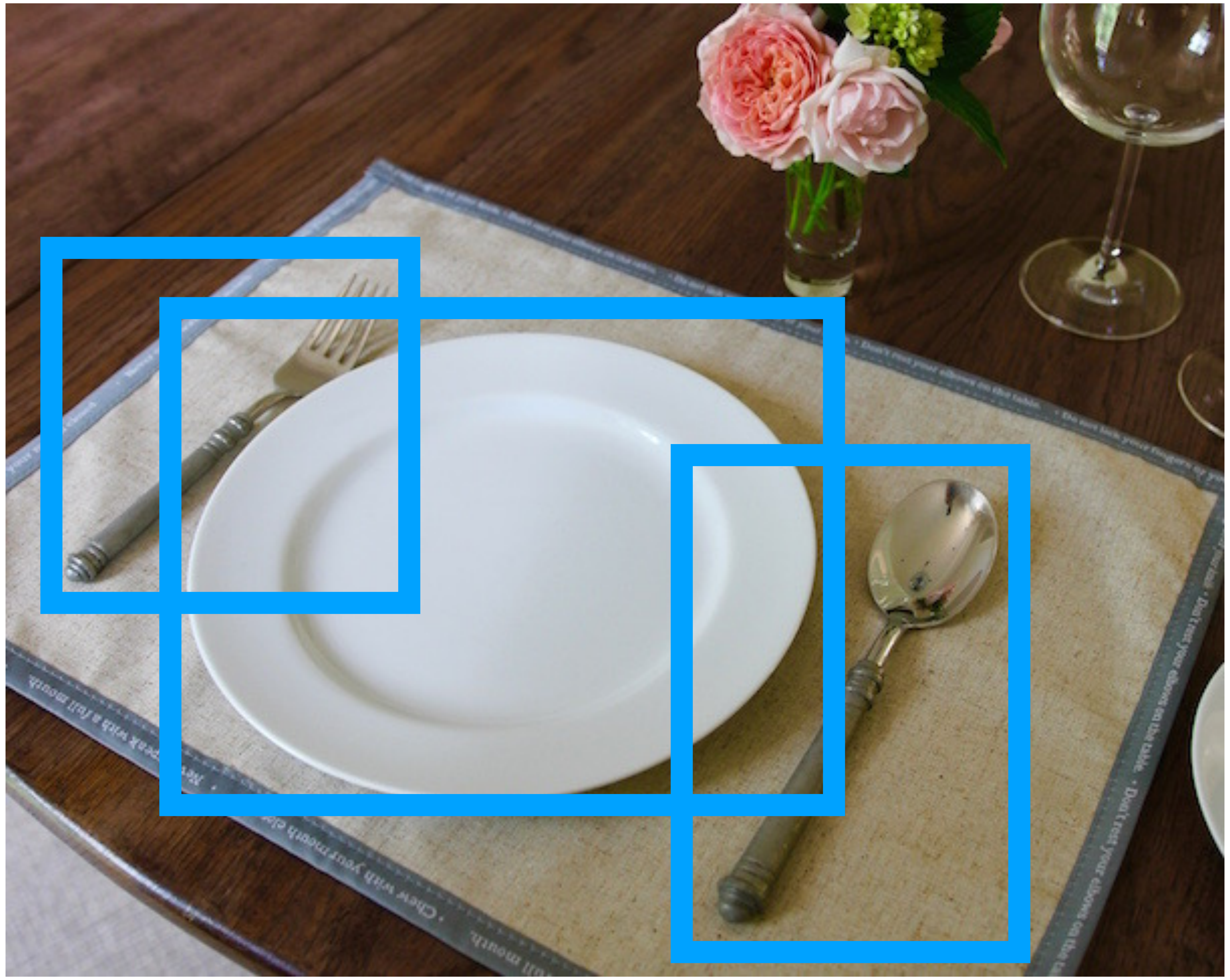
Multiple outputs: Need to output variable numbers of objects per image

Multiple types of output: Need to predict ”what” (category label) as well as “where” (bounding box)

Large images: Classification works at 224x224; need higher resolution for detection, often ~800x600

Bounding Boxes

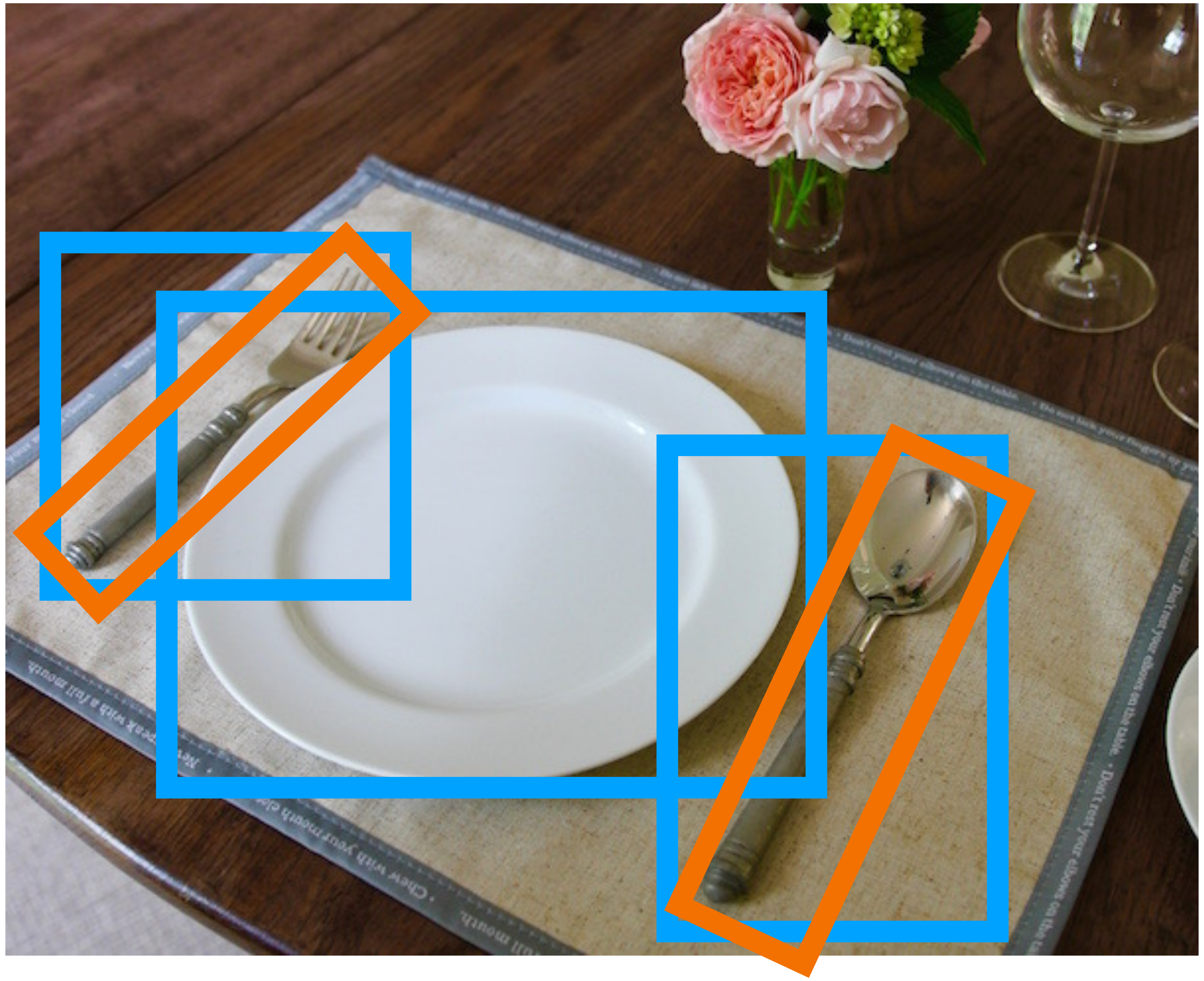
Bounding boxes are typically *axis-aligned*



Bounding Boxes

Bounding boxes are typically *axis-aligned*

Oriented boxes are much less common



Object Detection: Modal vs Amodal Boxes

Bounding boxes cover only the visible portion of the object

Zhu et al, "Semantic Amodal Segmentation", CVPR 2017

Object Detection: Modal vs Amodal Boxes

Bounding boxes cover only the visible portion of the object

Amodal detection: box covers the entire extent of the object, even occluded parts

Object Detection: Modal vs Amodal Boxes

"Modal" detection: Bounding boxes (usually) cover only the visible portion of the object

Amodal detection: box covers the entire extent of the object, even occluded parts

Comparing Boxes: Intersection over Union (IoU)

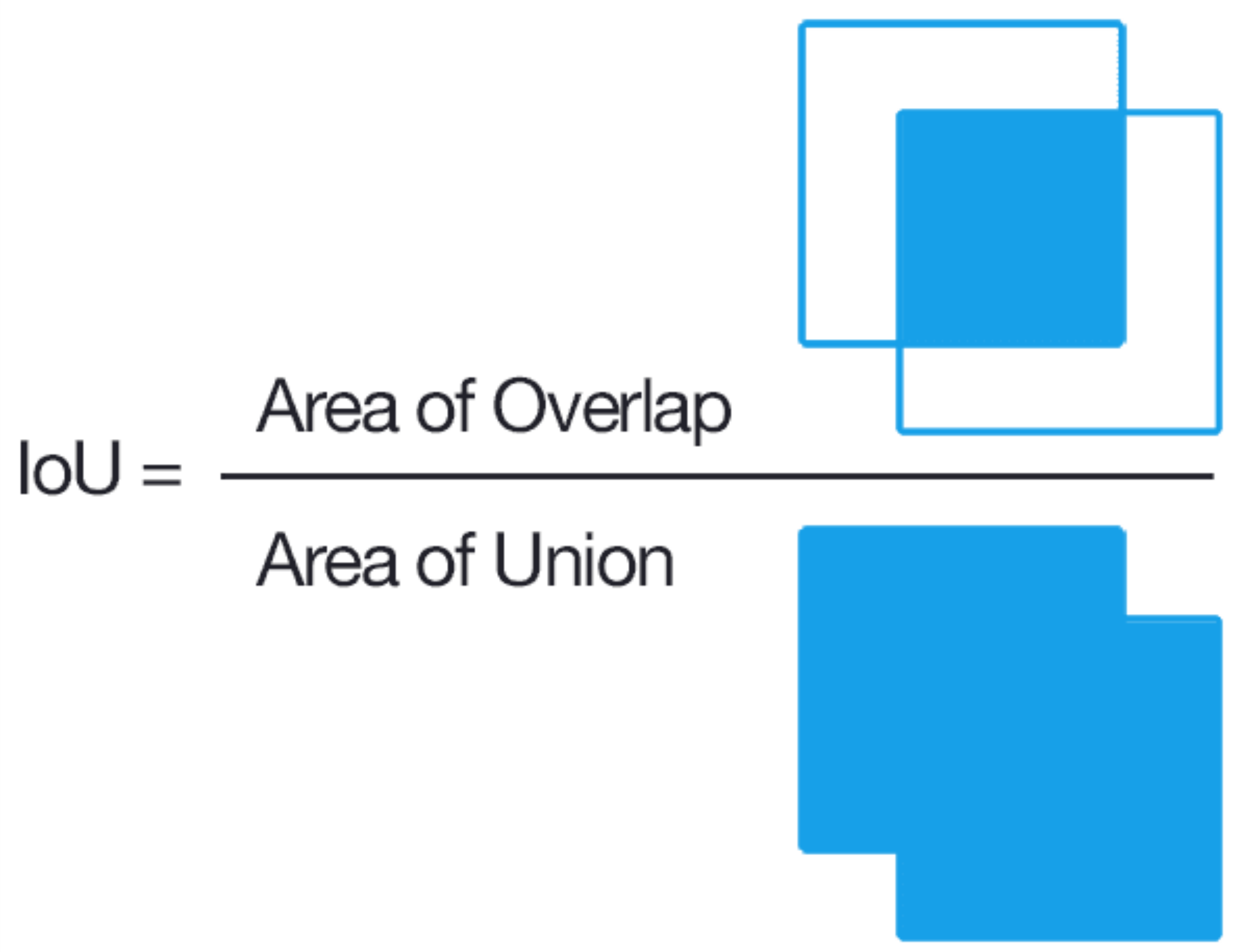
How can we compare our prediction to the ground-truth box?

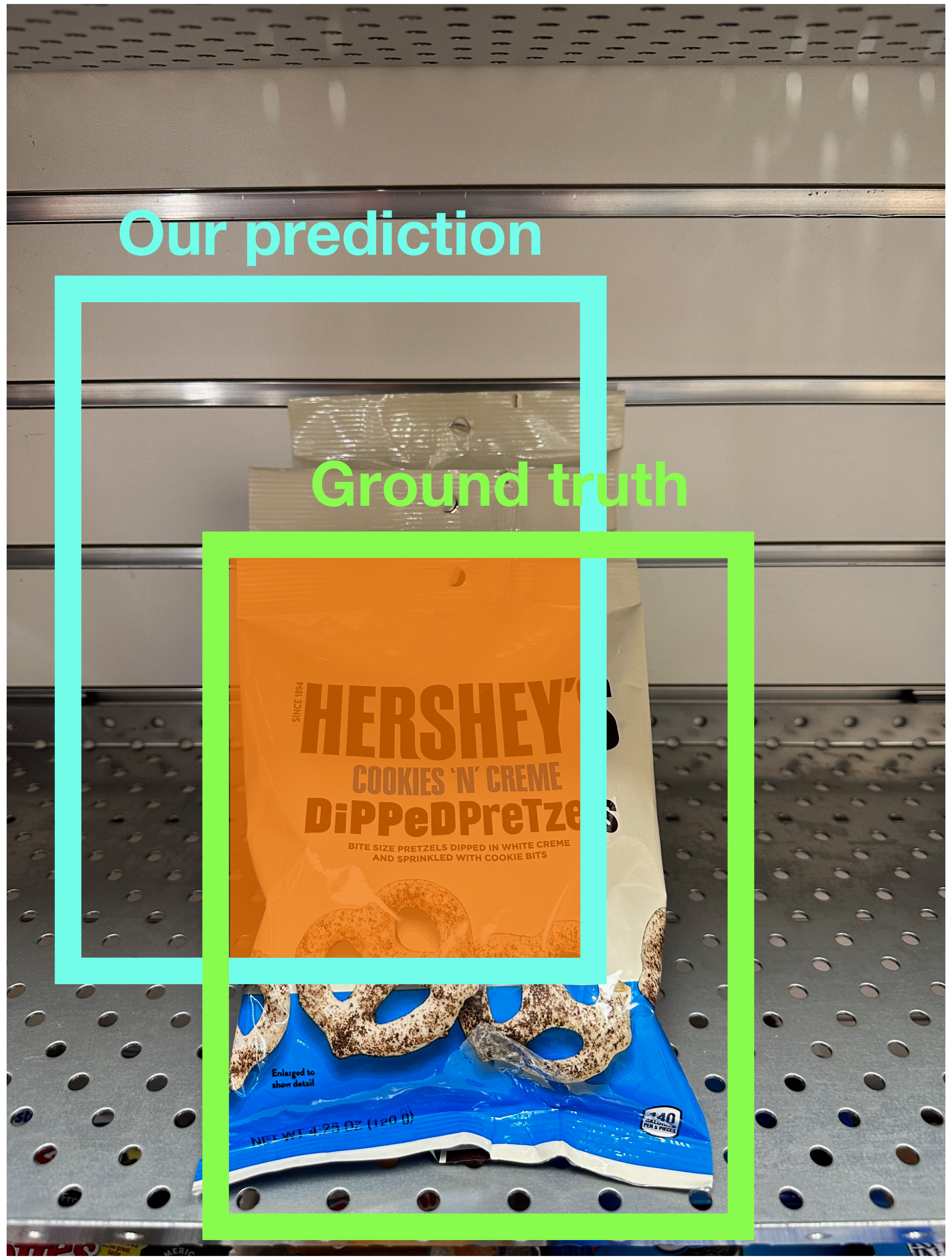
Evaluation Metric

Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU) (Also called “Jaccard similarity” or “Jaccard index”):



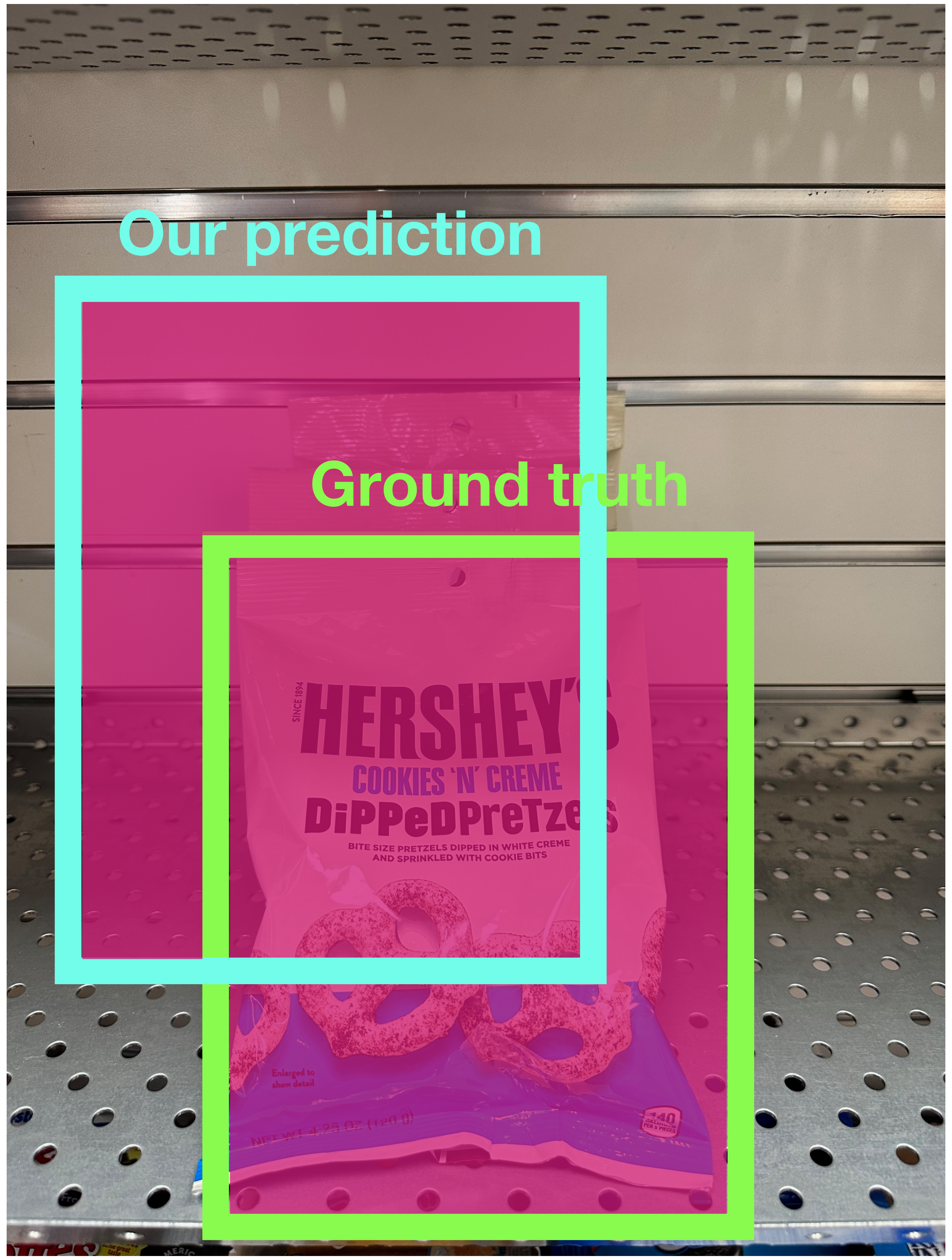


Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU) (Also called "Jaccard similarity" or "Jaccard index"):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$



Comparing Boxes: Intersection over Union (IoU)

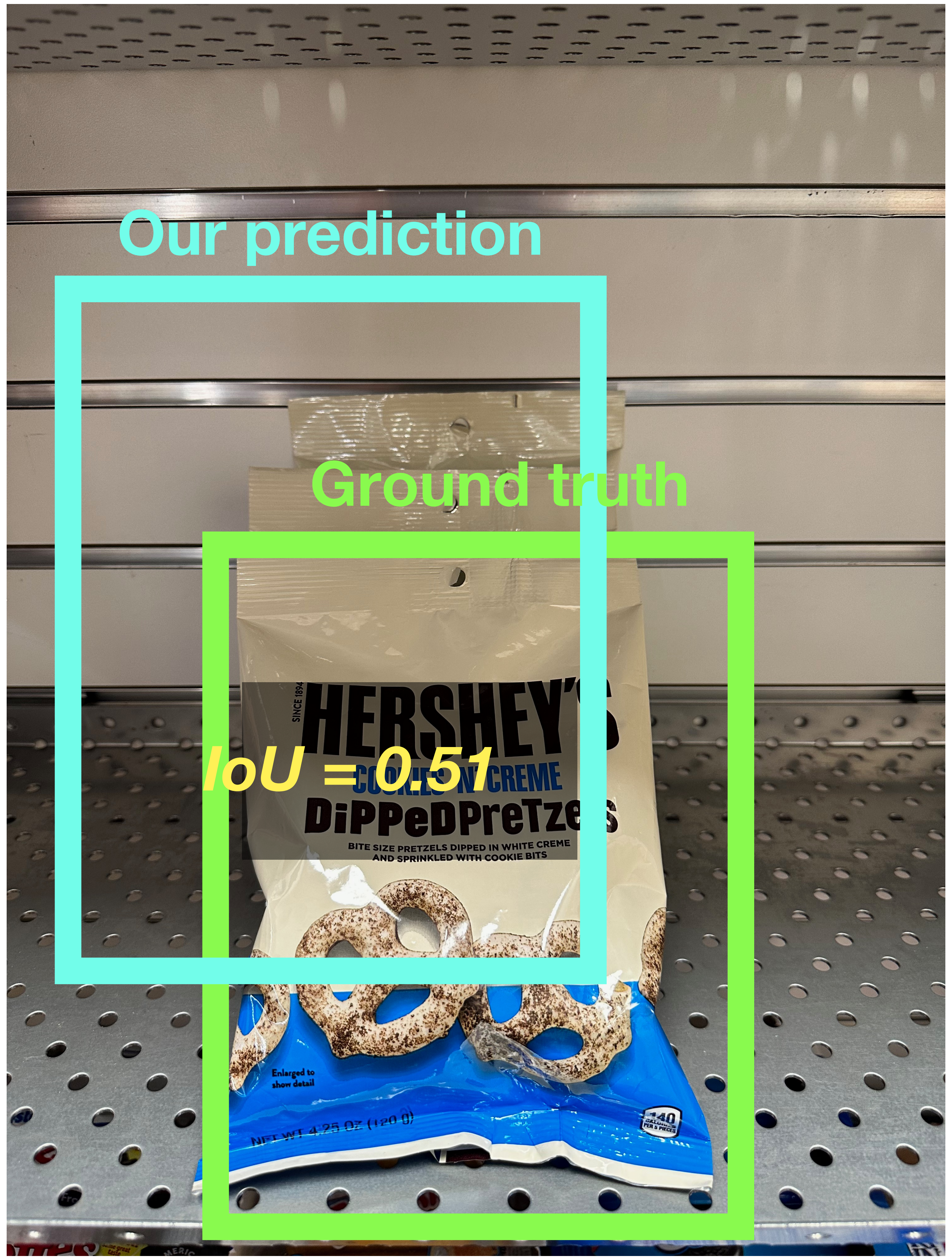
How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU) (Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

IoU > 0.5 is “decent”,
IoU > 0.7 is “pretty good”,

IoU > 0.9 is “almost perfect”



Comparing Boxes: Intersection over Union (IoU)

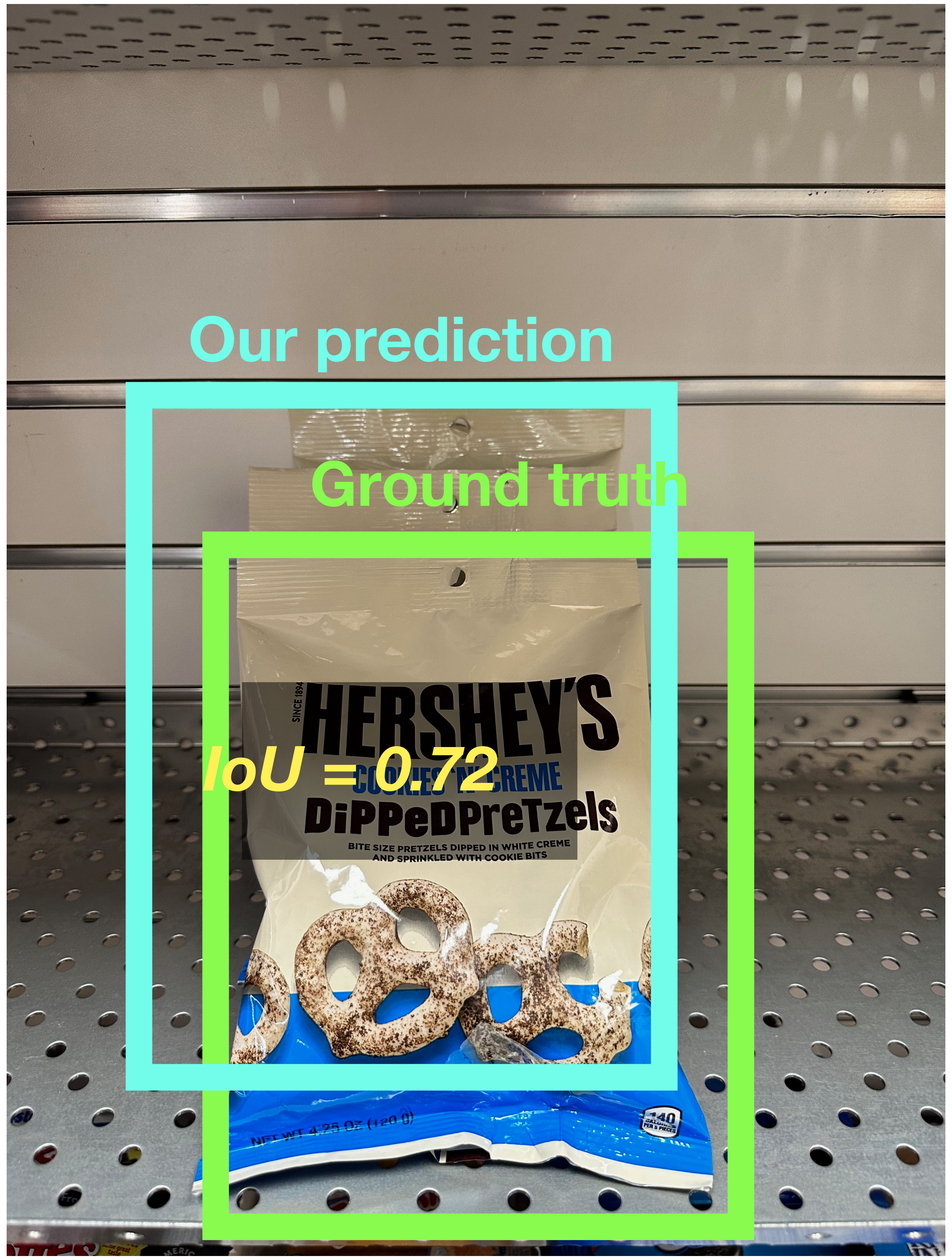
How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU) (Also called "Jaccard similarity" or "Jaccard index"):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

IoU > 0.5 is "decent",
IoU > 0.7 is "pretty good",

IoU > 0.9 is "almost perfect"



Comparing Boxes: Intersection over Union (IoU)

How can we compare our prediction to the ground-truth box?

Intersection over Union (IoU) (Also called “Jaccard similarity” or “Jaccard index”):

$$\frac{\textit{Area of Intersection}}{\textit{Area of Union}}$$

IoU > 0.5 is “decent”,
IoU > 0.7 is “pretty good”,

IoU > 0.9 is “almost perfect”

Detecting a single object

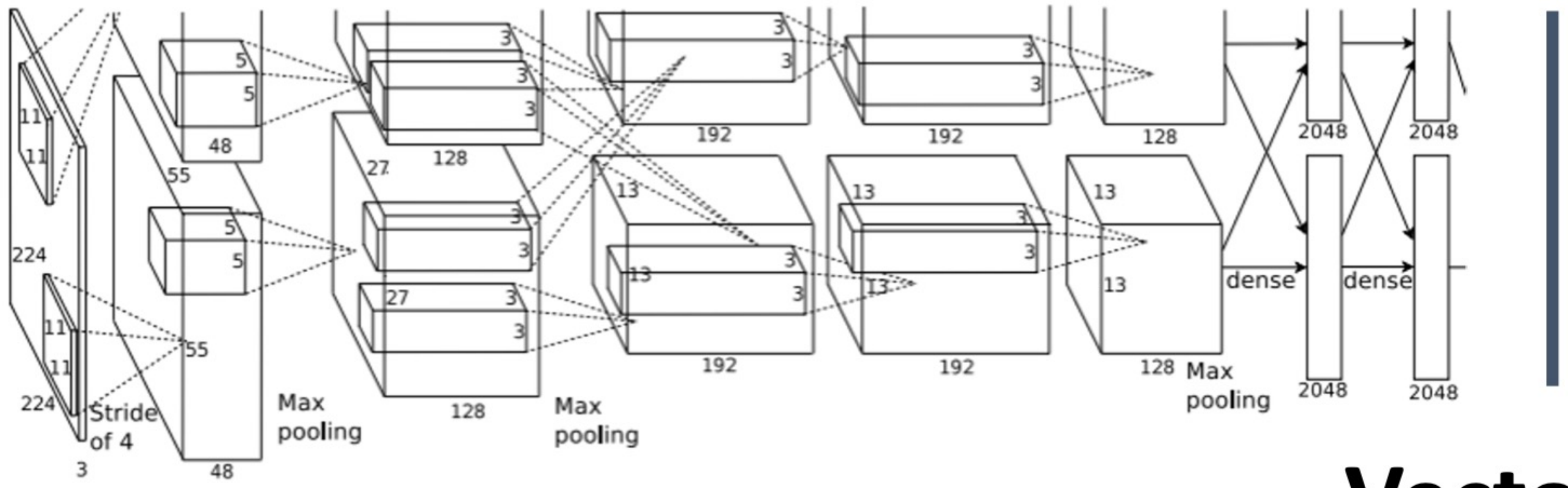


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Loss

Treat localization as a regression problem!

Detecting a single object

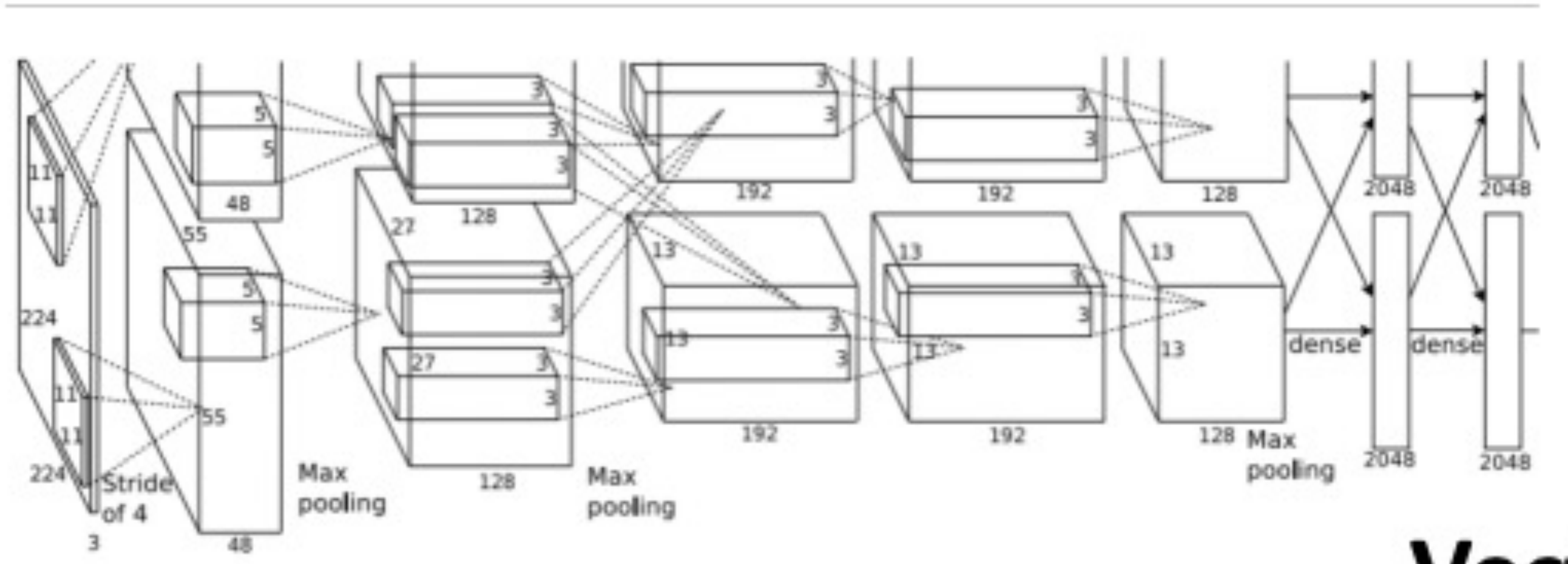


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully connected:
4096 to 10

What??

Class scores:
Chocolate Pretzels: 0.9
Granola Bar: 0.02
Potato Chips: 0.02
Water Bottle: 0.02
Popcorn: 0.01
....

Correct Label:
Chocolate Pretzels

Softmax Loss

Treat localization as a regression problem!

Detecting a single object

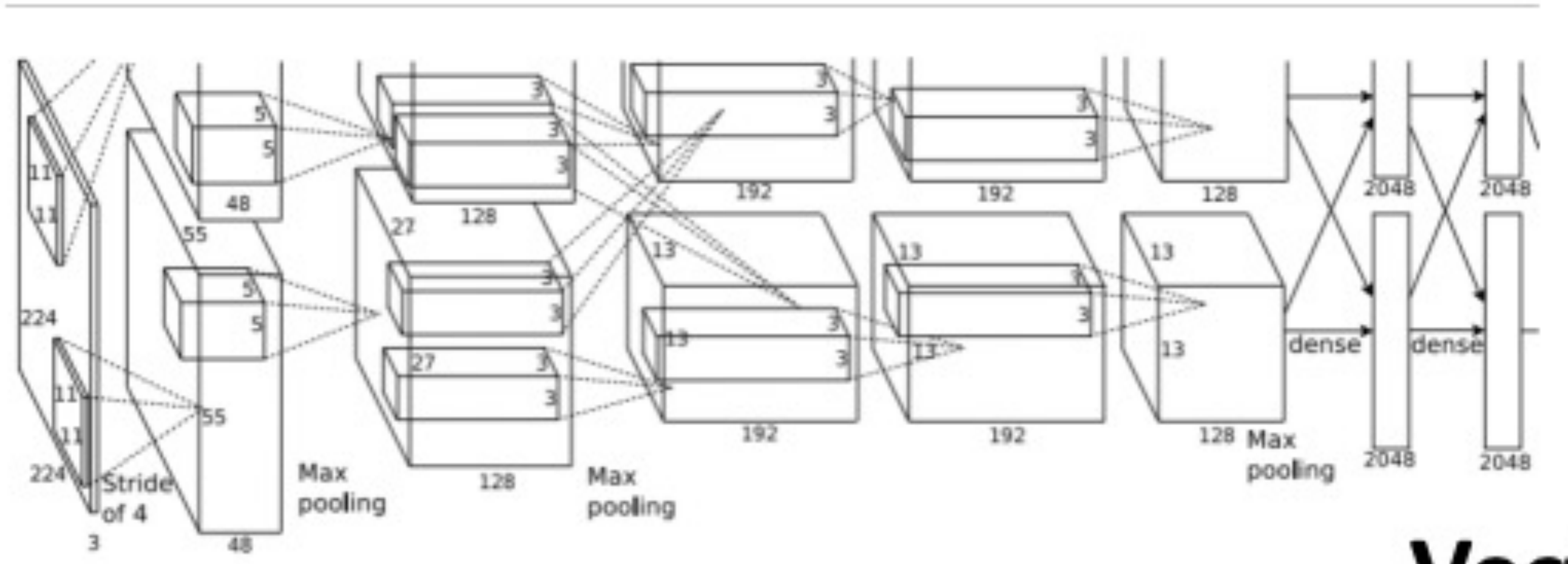


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector: 4096

What??

Fully connected: 4096 to 10

Class scores:
Chocolate Pretzels: 0.9
Granola Bar: 0.02
Potato Chips: 0.02
Water Bottle: 0.02
Popcorn: 0.01
....

Correct Label:
Chocolate Pretzels

Softmax Loss

Fully connected: 4096 to 10

Box coordinates:
(x, y, w, h)

L2 Loss

Correct coordinates:
(x', y', w', h')

Where??

Treat localization as a regression problem!

Detecting a single object

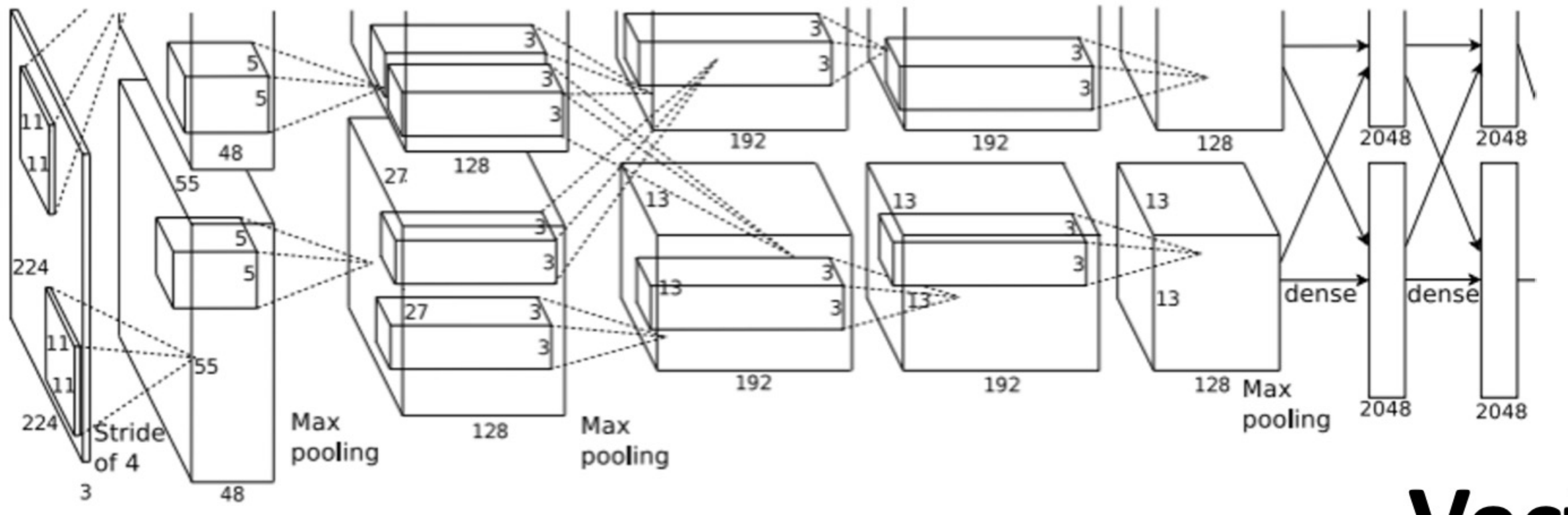


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Treat localization as a regression problem!

Vector: 4096

Fully connected:

4096 to 10

Class scores:

Chocolate Pretzels: 0.9

What??

Granola Bar: 0.02

Potato Chips: 0.02

Water Bottle: 0.02

Popcorn: 0.01

....

Correct Label:

Chocolate Pretzels

Softmax Loss

Multitask Loss

Weighted Sum

Loss

$$L = L_{cls} + \lambda L_{reg}$$

Fully connected:

4096 to 10

Box coordinates:

(x, y, w, h)

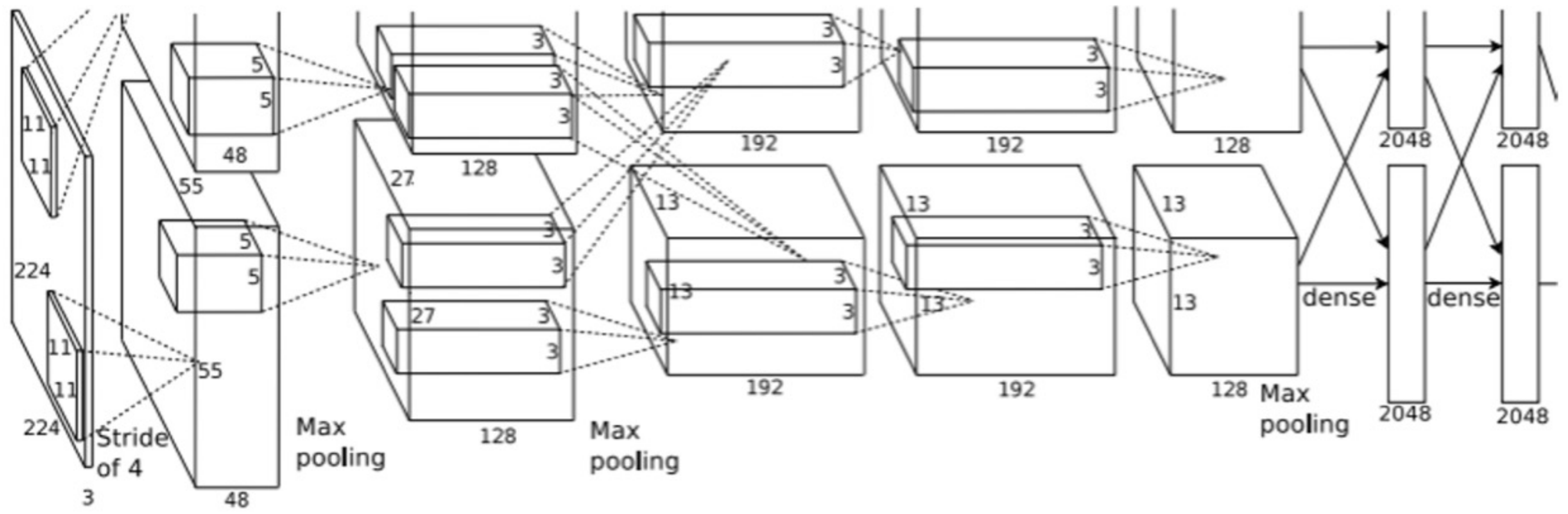
Where??

L2 Loss

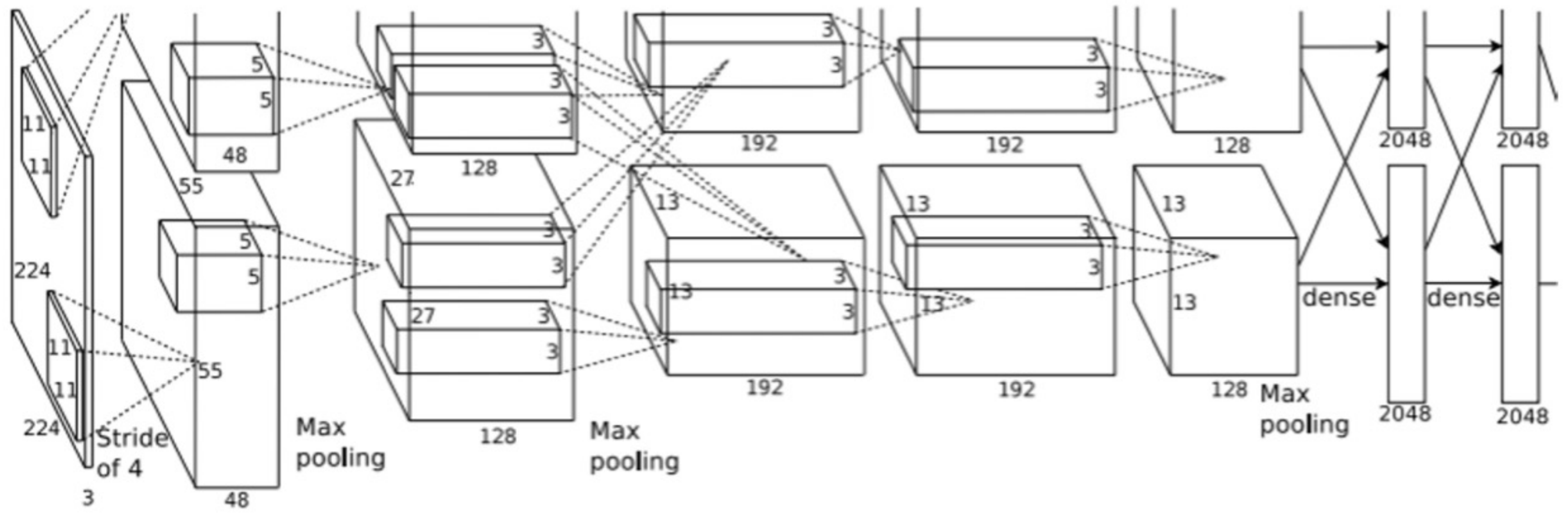
Correct coordinates:

(x’, y', w’, h’)

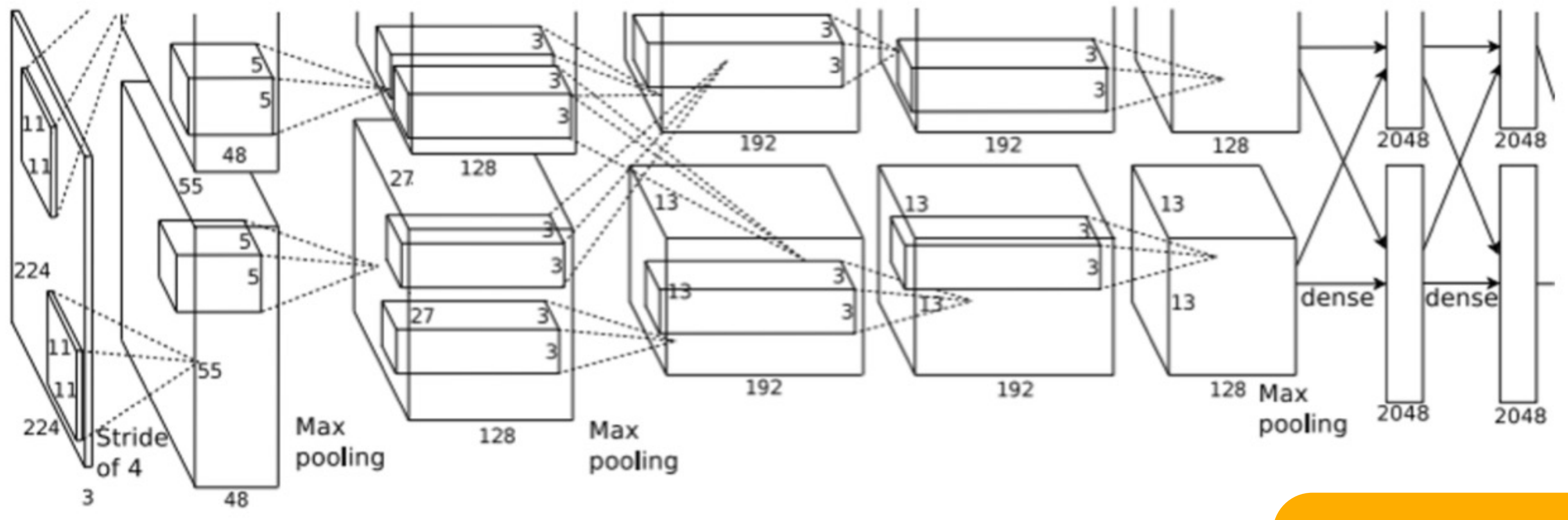
Detecting Multiple Objects



Hershey's: (x, y, w, h)
4 numbers



Hershey's: (x, y, w, h)
Flipz: (x, y, w, h)
Reese's (x, y, w, h)
12 numbers

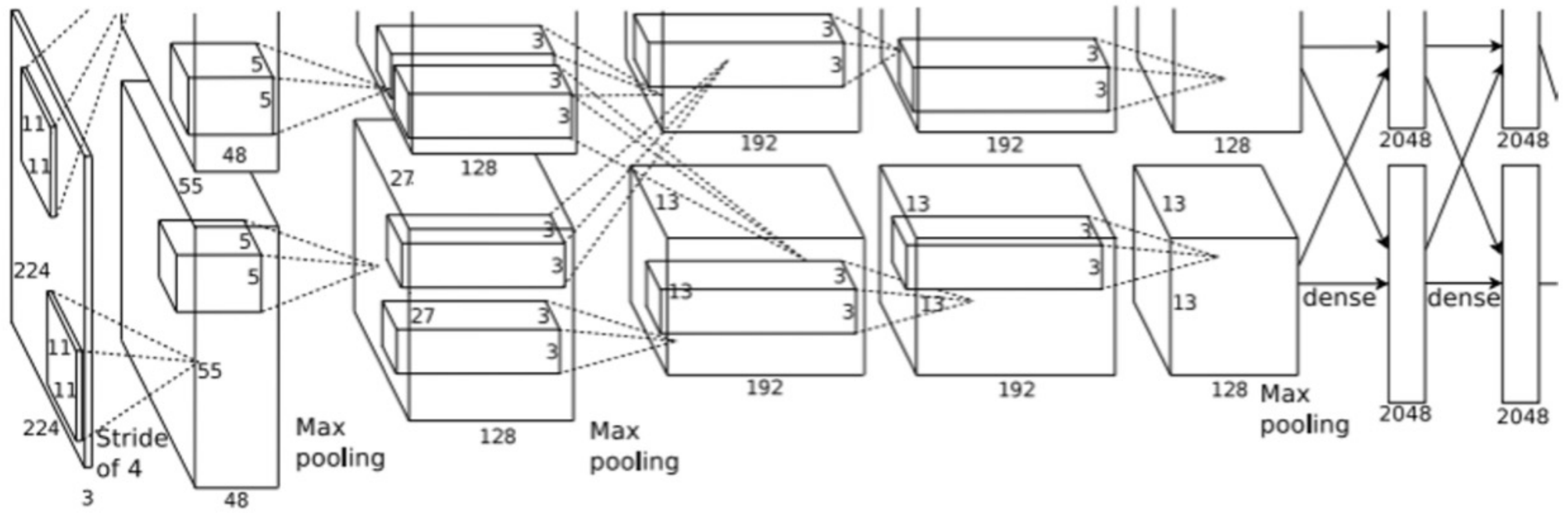


Chips: (x, y, w, h)
Chips: (x, y, w, h)
…..
Many numbers!

Need different numbers of output per image

Detecting Multiple Objects: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Hershey's: No

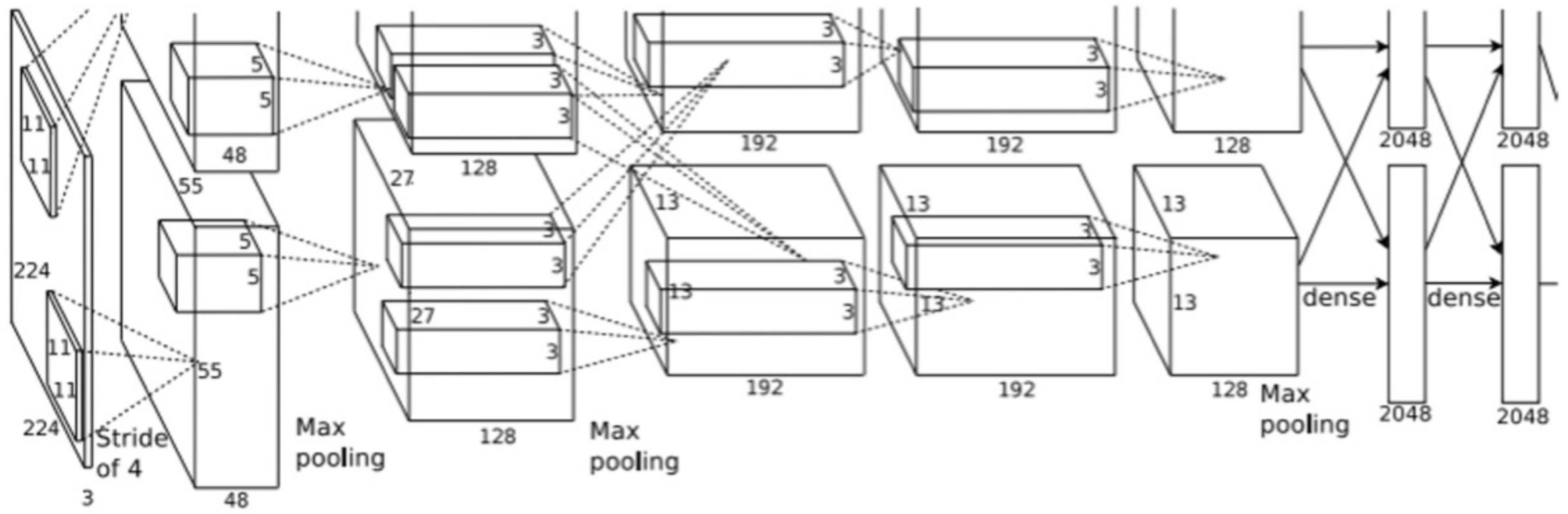
Flipz: No

Reese's: No

Background: Yes

Detecting Multiple Objects: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Hershey's: No

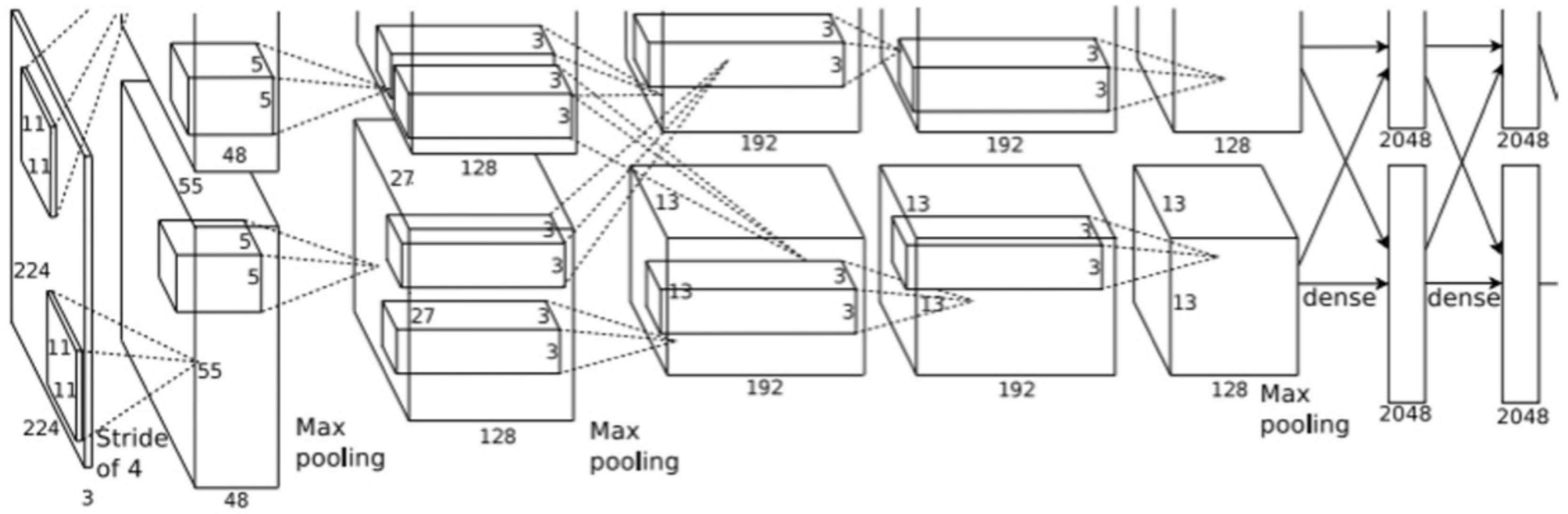
Flipz: Yes

Reese's: No

Background: No

Detecting Multiple Objects: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Hershey's: No

Flipz: No

Reese's: Yes

Background: No

Detecting Multiple Objects: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

Question: How many possible boxes are there in an image of size H x W?

Consider box of size h x w:
Possible x positions: W - w + 1
Possible y positions: H - h + 1
Possible positions:
(W-w+1) x (H-h+1)

Total possible boxes:

$$\sum_{h=1}^{H}\sum_{w=1}^{W}(W-w+1)(H-h+1)$$

$$=\frac{H(H+1)}{2}\frac{W(W+1)}{2}$$

Detecting Multiple Objects: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

Question: How many possible boxes are there in an image of size H x W?

Consider box of size h x w:
Possible x positions: W - w + 1
Possible y positions: H - h + 1
Possible positions:
(W-w+1) x (H-h+1)

800 x 600 image has ~58M boxes. No way we can evaluate them all

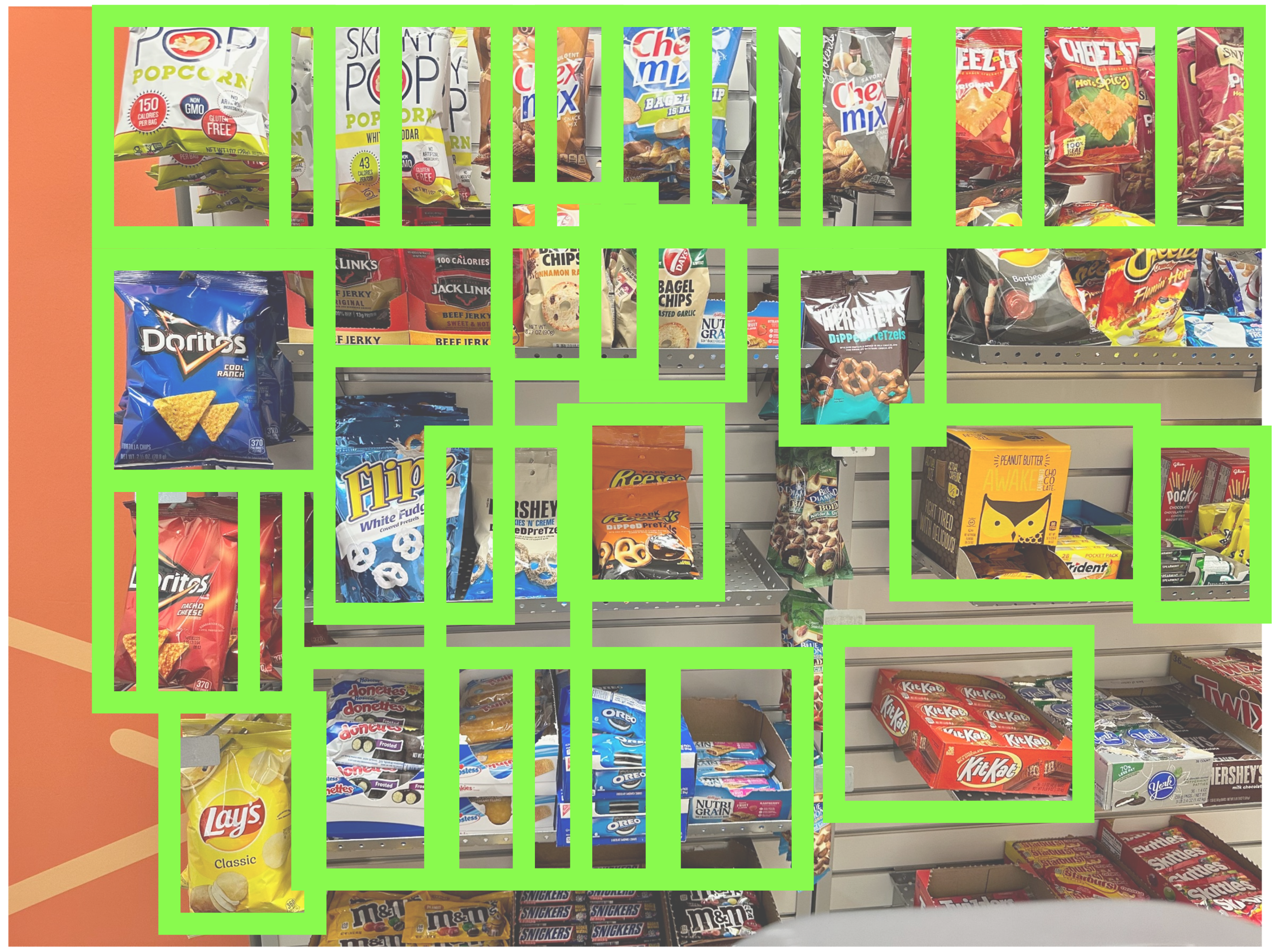
Total possible boxes:

$$\sum_{h=1}^{H}\sum_{w=1}^{W}(W-w+1)(H-h+1)$$

$$= \frac{H(H+1)}{2}\frac{W(W+1)}{2}$$

Region Proposals

- Find a small set of boxes that are likely to cover all objects
- Often based on heuristics: e.g. look for "blob-like" image regions
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, "Measuring the objectness of image windows", TPAMI 2012
Uijlings et al, "Selective Search for Object Recognition", IJCV 2013
Cheng et al, "BING: Binarized normed gradients for objectness estimation at 300fps", CVPR 2014
Zitnick and Dollar, "Edge boxes: Locating object proposals from edges", ECCV 2014

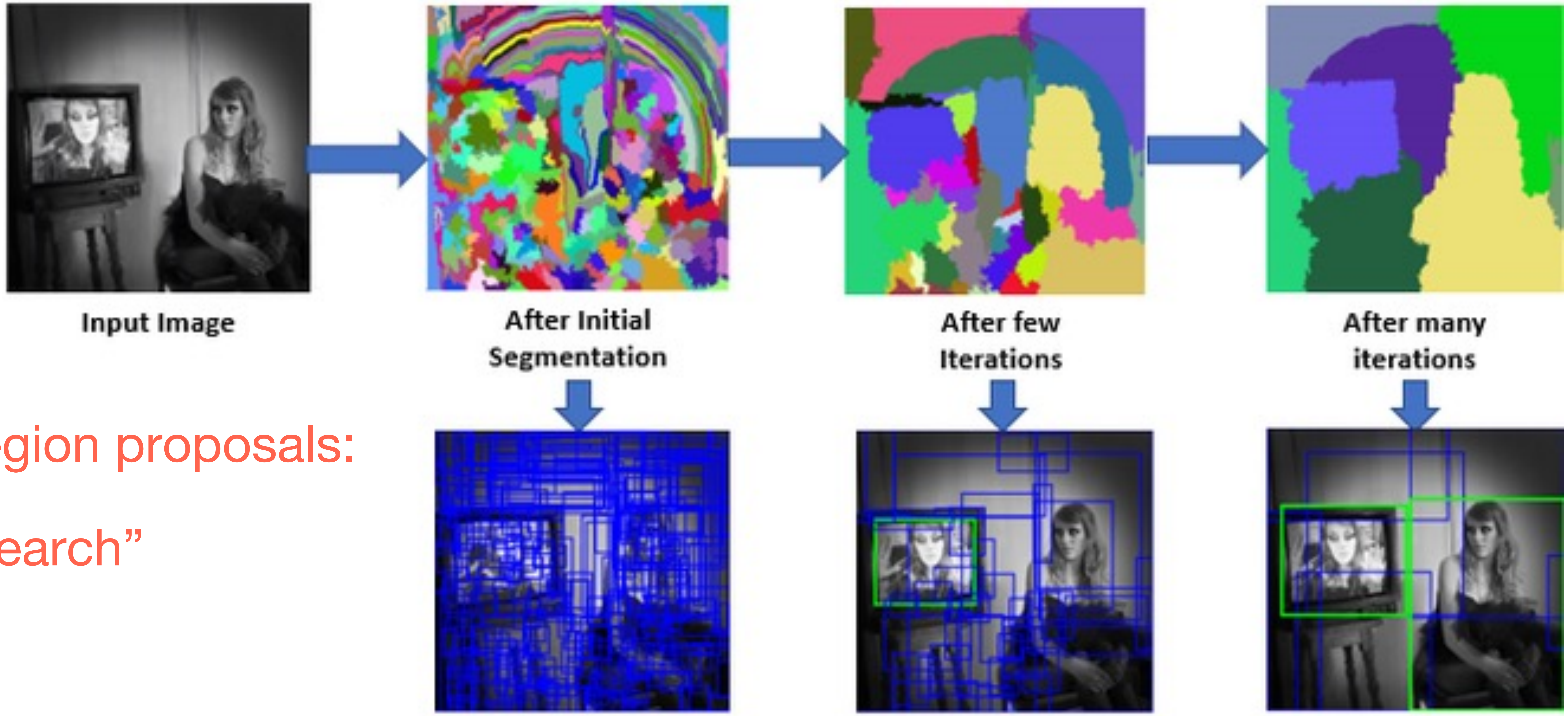
R-CNN: Region-Based CNN

R-CNN: Region-Based CNN

Input image

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

R-CNN: Region-Based CNN



Generate region proposals:

"selective search"

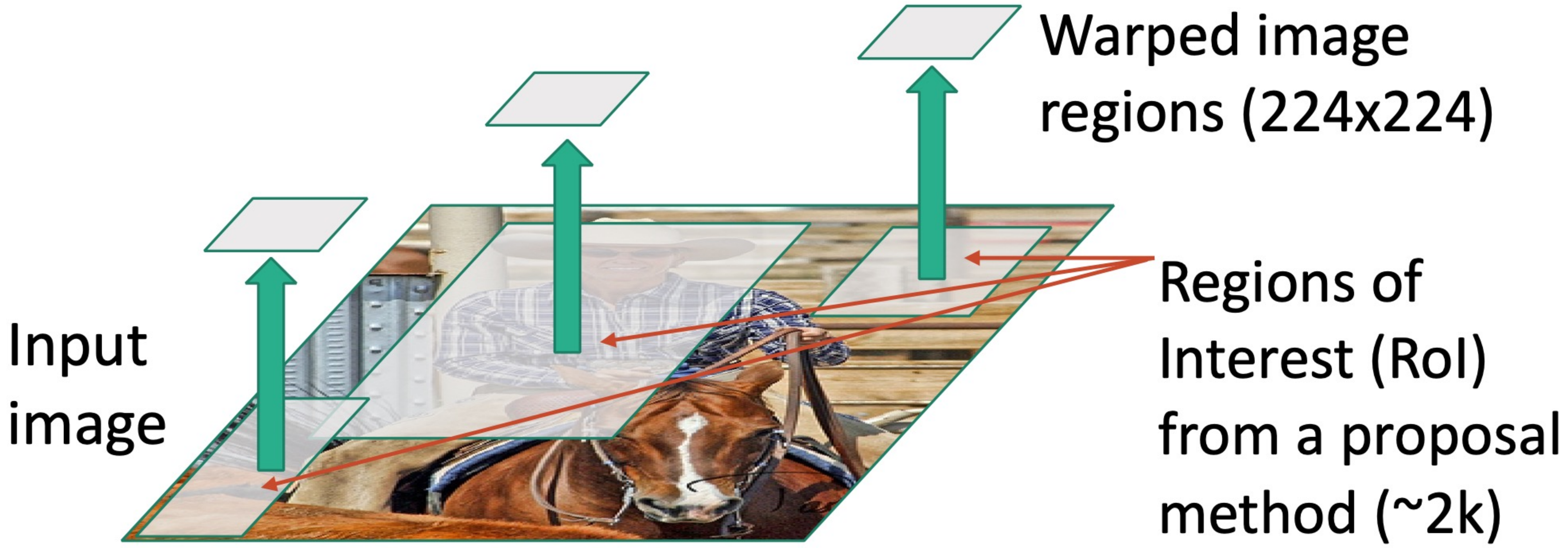
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

R-CNN: Region-Based CNN

R-CNN: Region-Based CNN

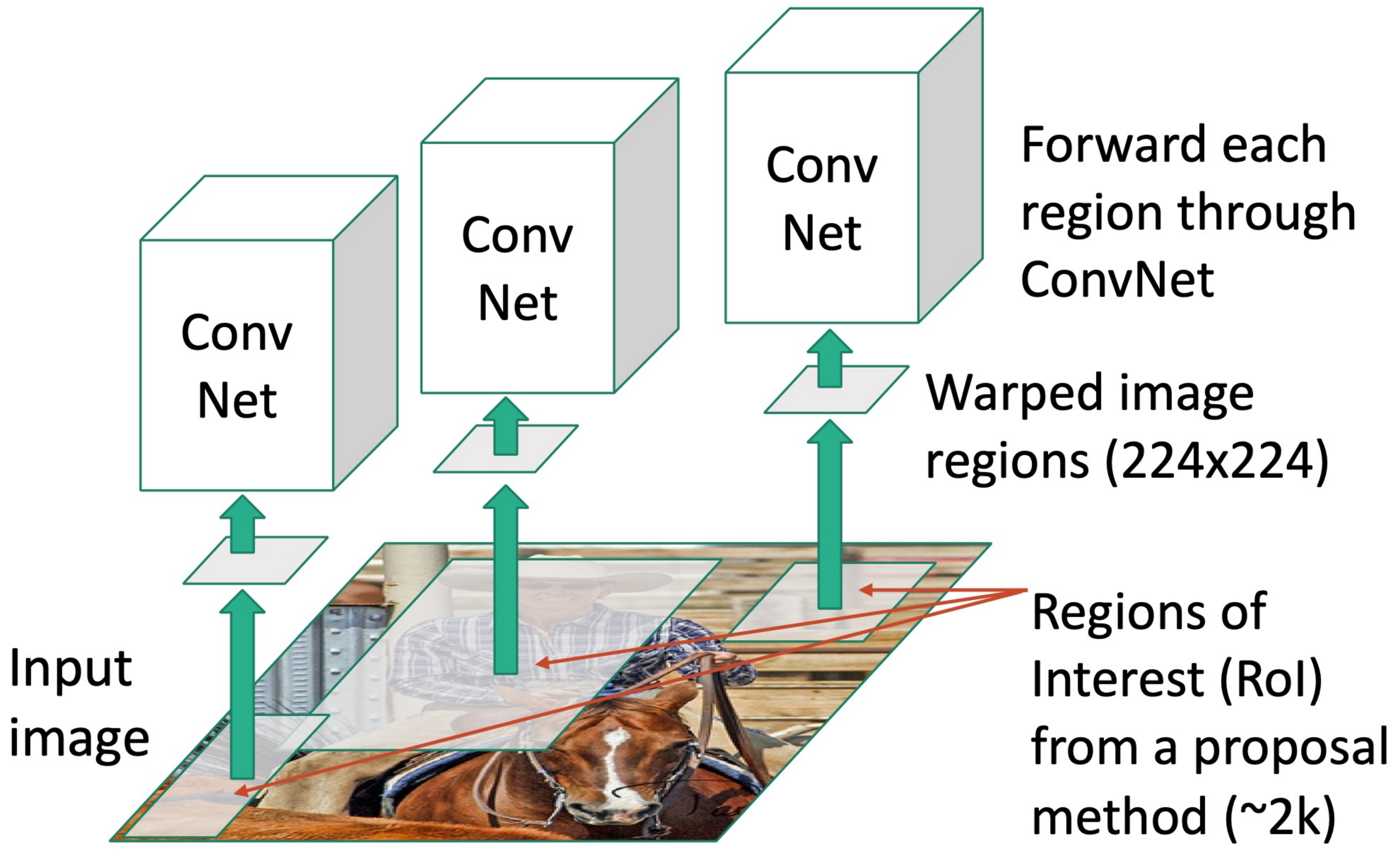
R-CNN: Region-Based CNN

R-CNN: Region-Based CNN



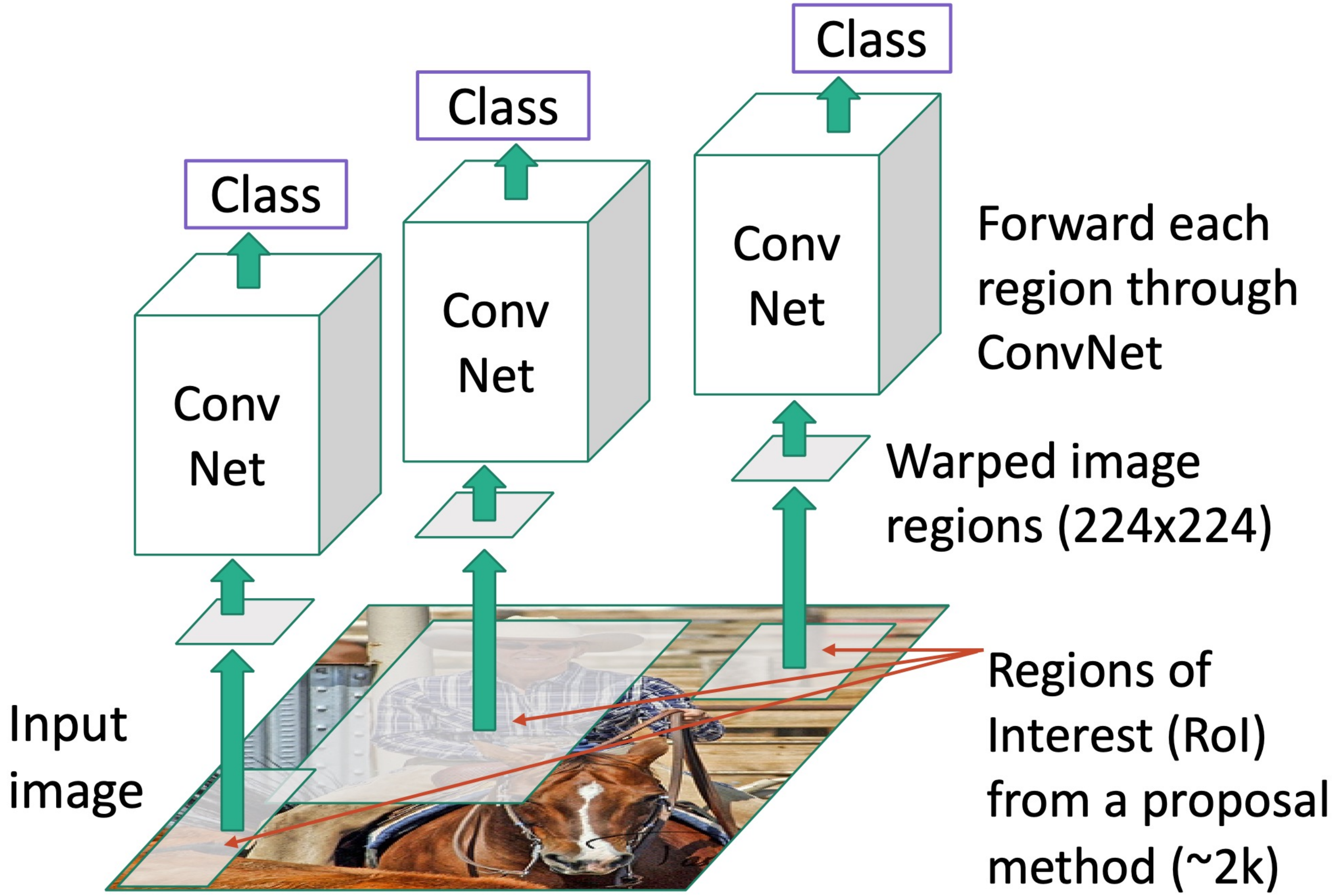
R-CNN: Region-Based CNN

R-CNN: Region-Based CNN



R-CNN: Region-Based CNN

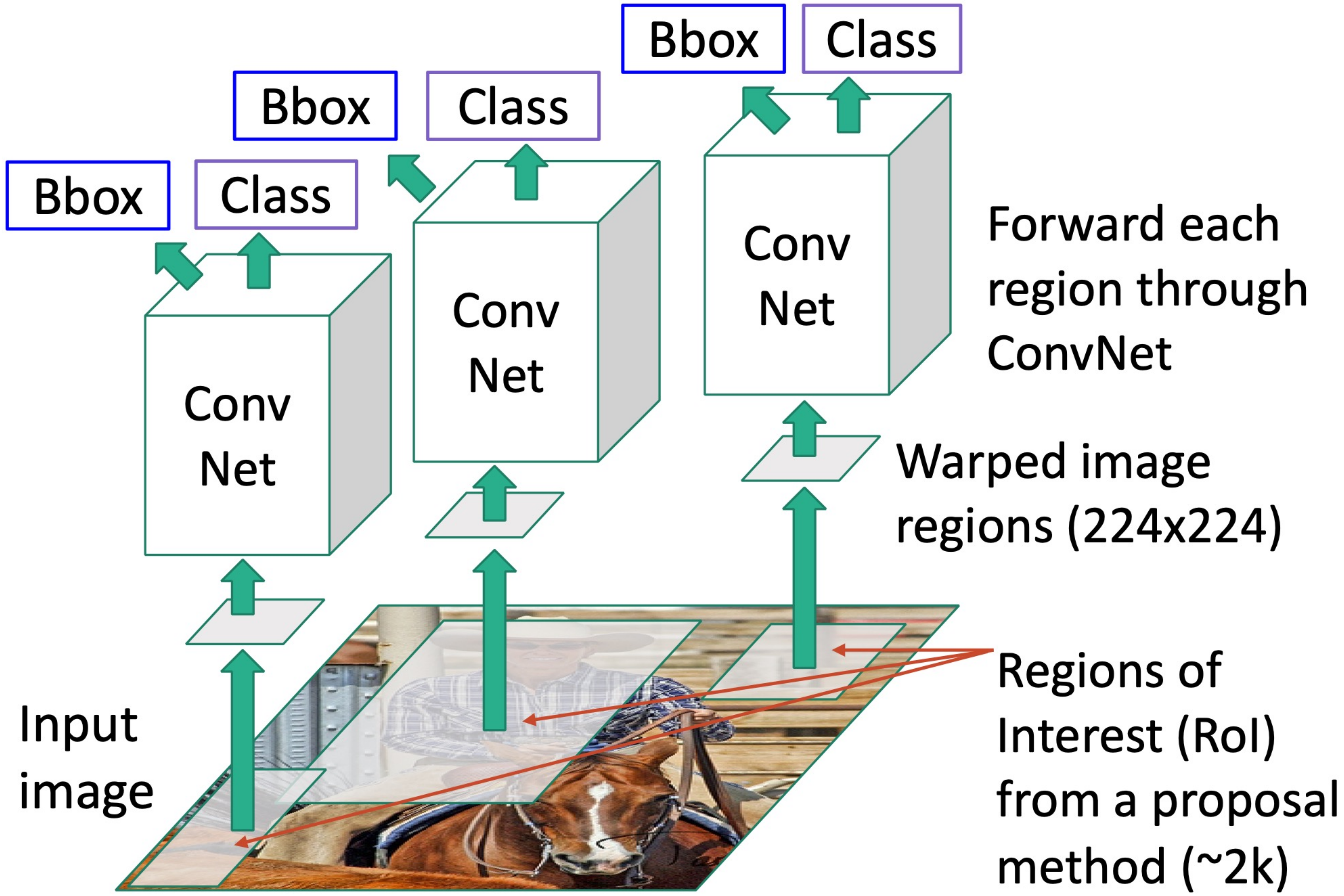
R-CNN: Region-Based CNN



Classify each region

R-CNN: Region-Based CNN

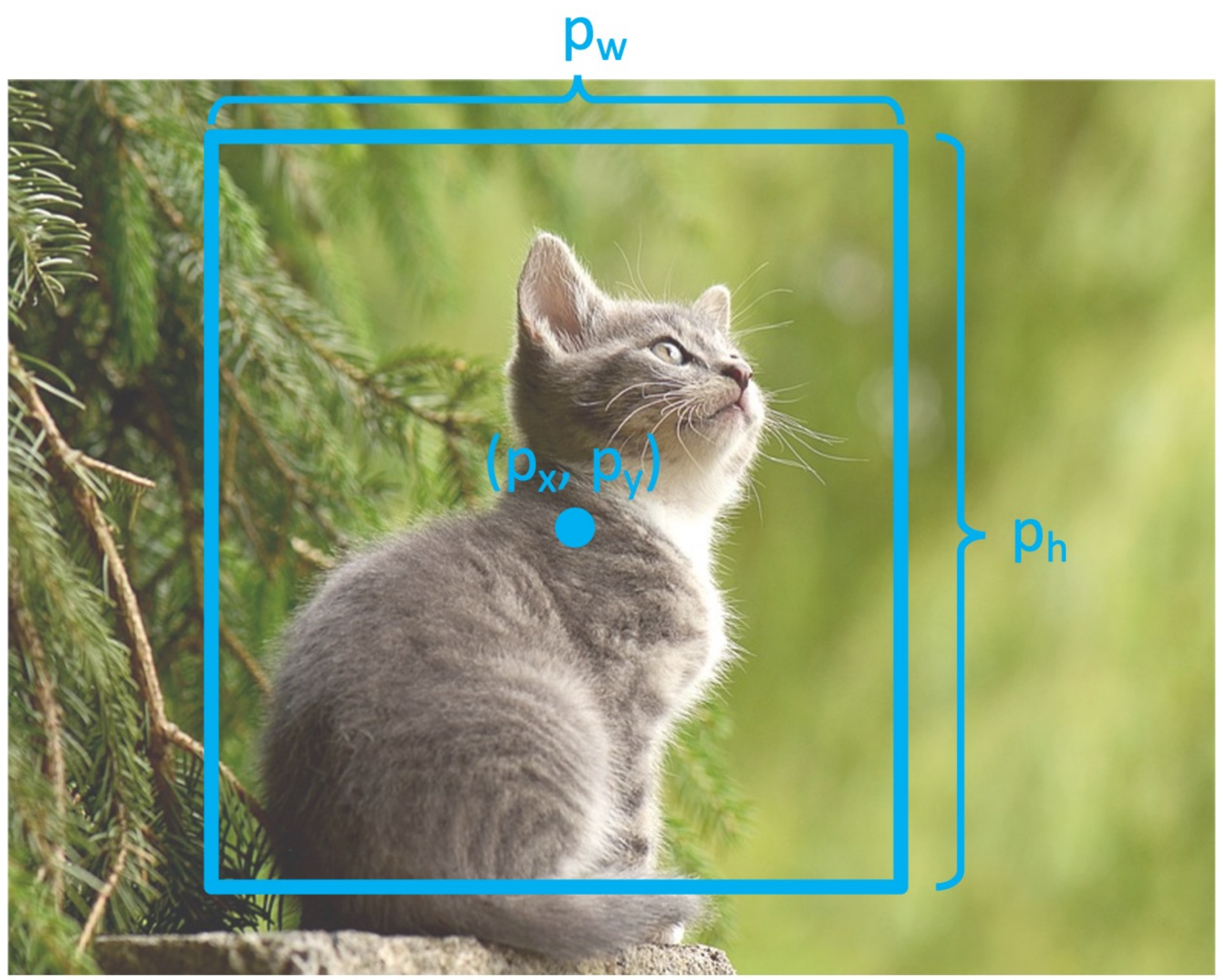
R-CNN: Region-Based CNN



Classify each region

Bounding box regression:
Predict "transform" to correct the RoI: 4 numbers (t_x, t_y, t_h, t_w)

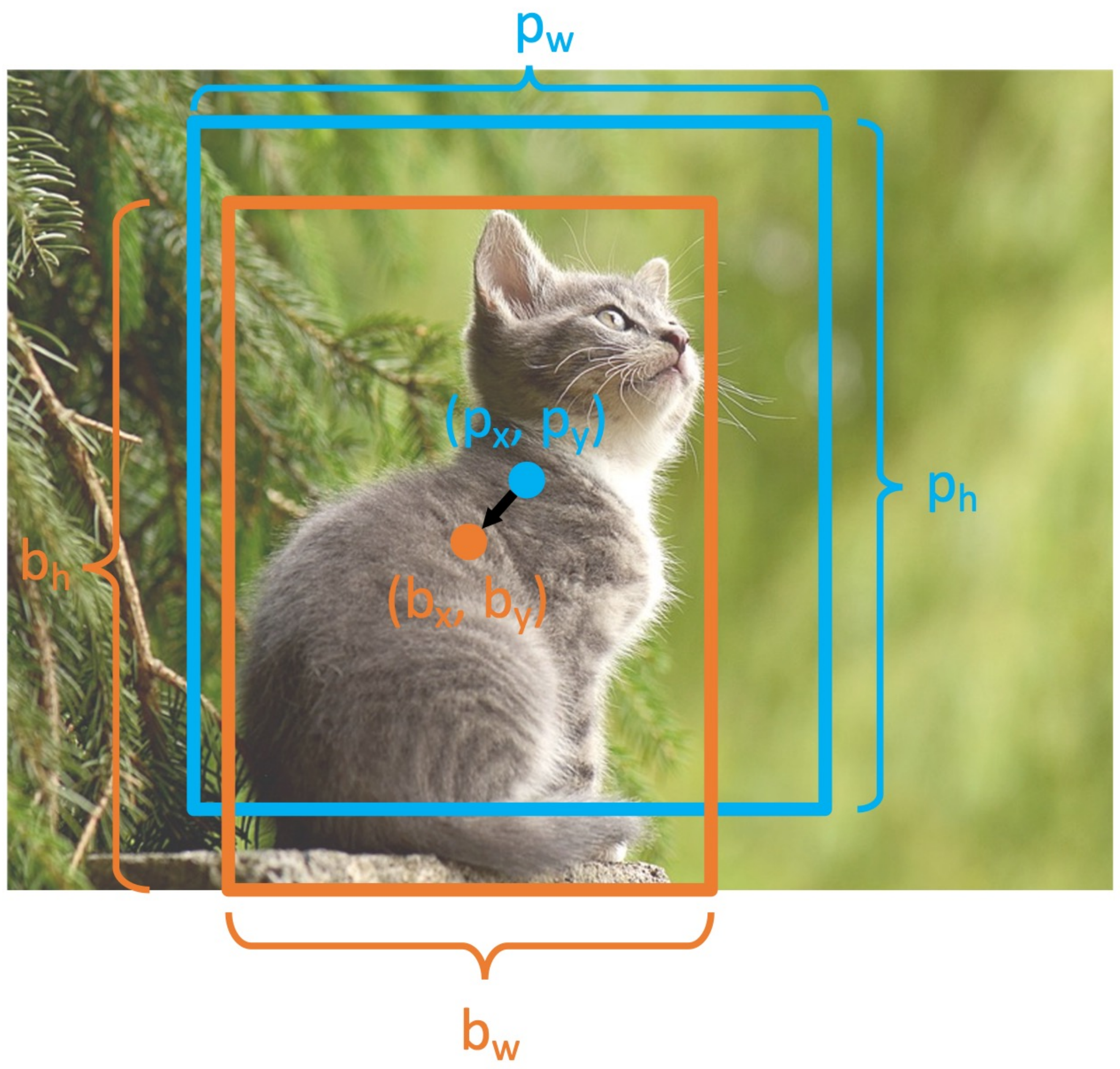
R-CNN: Box Regression



Consider a region proposal with center (p_x, p_y), width p_w, height p_h

Model predicts a transform (t_x, t_y, t_w, t_h) to correct the region proposal

R-CNN: Box Regression



Consider a region proposal with center (p_x, p_y), width p_w, height p_h

Model predicts a transform (t_x, t_y, t_w, t_h) to correct the region proposal

The output box is defined by:

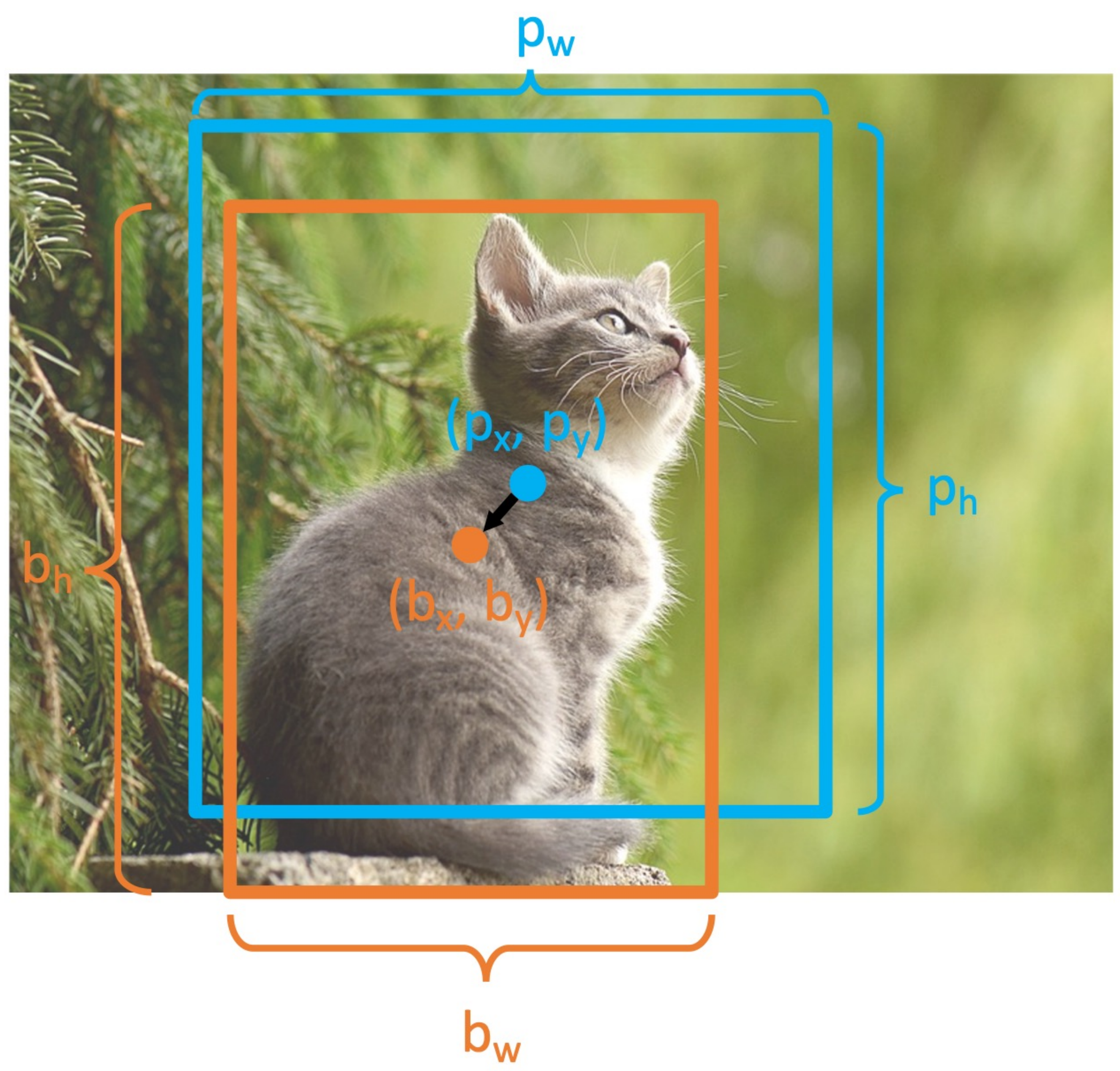
$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$

Shift center by amount relative to proposal size

$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

Scale proposal; exp ensures that scaling factor is > 0

R-CNN: Box Regression



Consider a region proposal with center (p_x, p_y), width p_w, height p_h

Model predicts a transform (t_x, t_y, t_w, t_h) to correct the region proposal

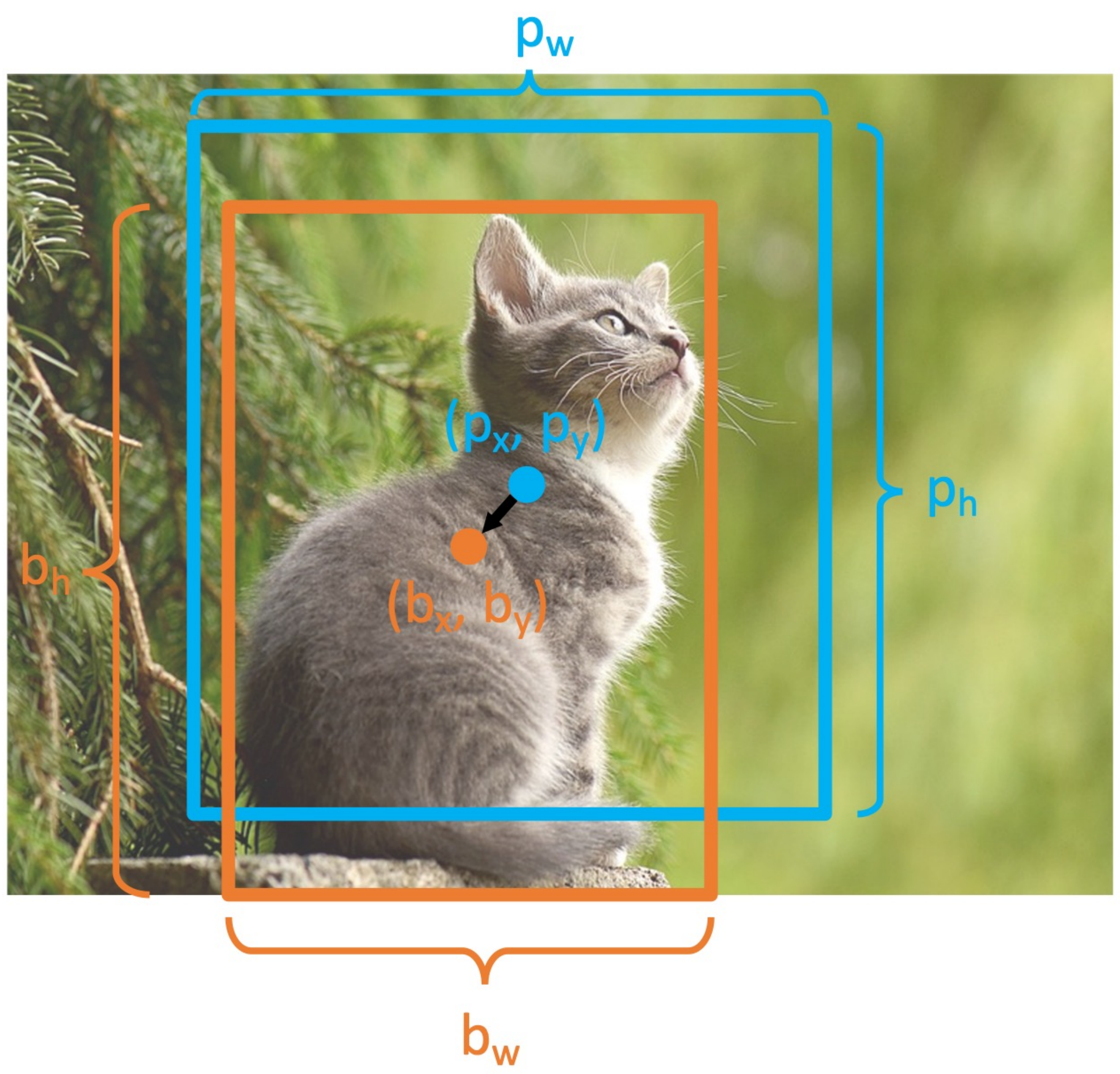
The output box is defined by:

$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$
$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

When transform is 0, output = proposal

L2 regularization encourages leaving proposal unchanged

R-CNN: Box Regression



Consider a region proposal with center (p_x, p_y), width p_w, height p_h

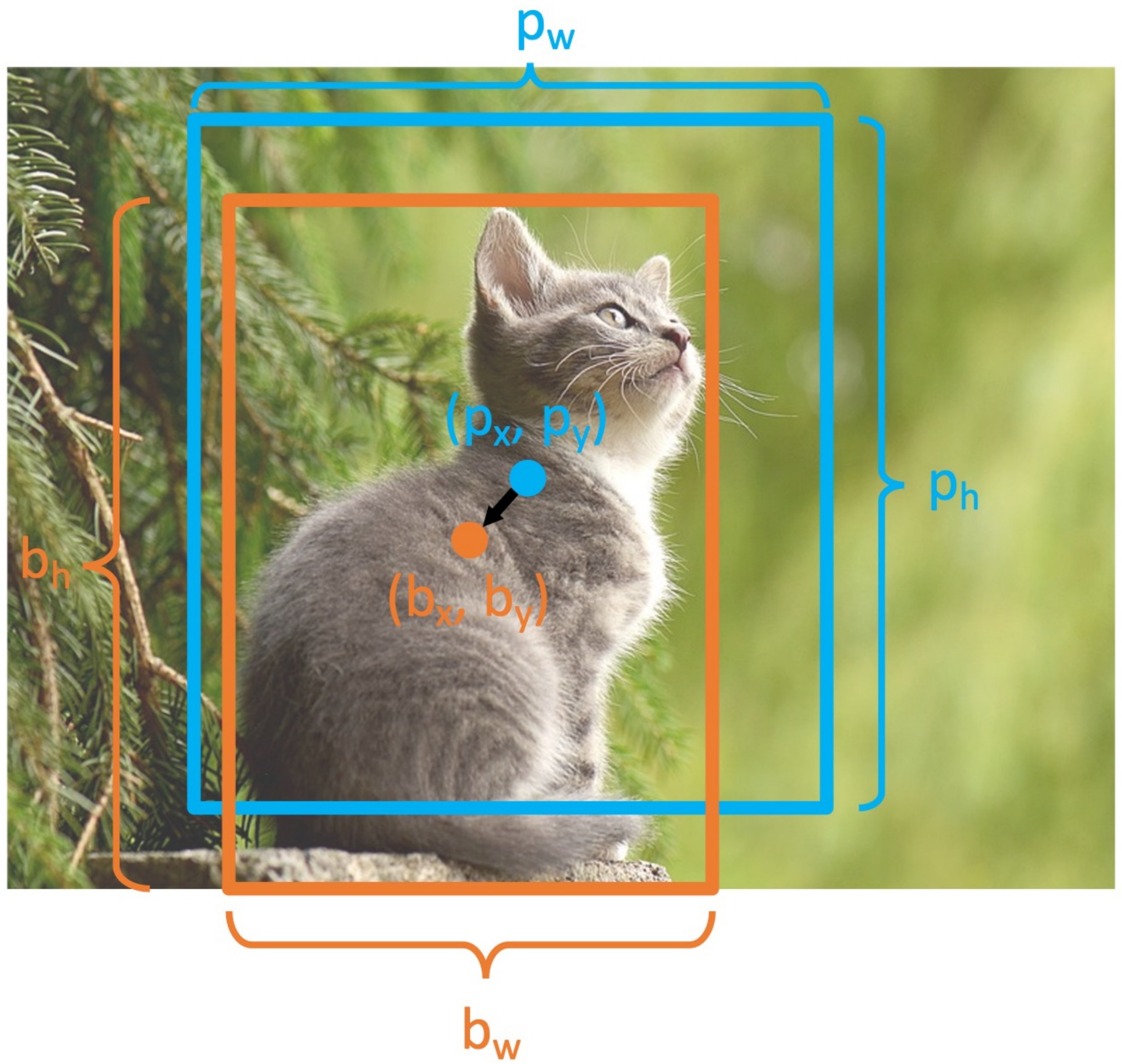
Model predicts a transform (t_x, t_y, t_w, t_h) to correct the region proposal

The output box is defined by:

$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$
$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

Scale / Translation invariance: Transform encodes *relative* difference between proposal and output; important since CNN doesn't see absolute size or position after cropping

R-CNN: Box Regression



Consider a region proposal with center (p_x, p_y), width p_w, height p_h

Model predicts a transform (t_x, t_y, t_w, t_h) to correct the region proposal

The output box is defined by:

$$b_x = p_x + p_w t_x$$
$$b_y = p_y + p_h t_y$$
$$b_w = p_w \exp(t_w)$$
$$b_h = p_h \exp(t_h)$$

Given proposal and target output, we can solve for the transform the network should output:

$$t_x = (b_x - p_x)/p_w$$
$$t_y = (b_y - p_y)/p_h$$
$$t_w = \log(b_w/p_w)$$
$$t_h = \log(b_h/p_h)$$

R-CNN: Training

Input Image

Ground Truth

R-CNN: Training

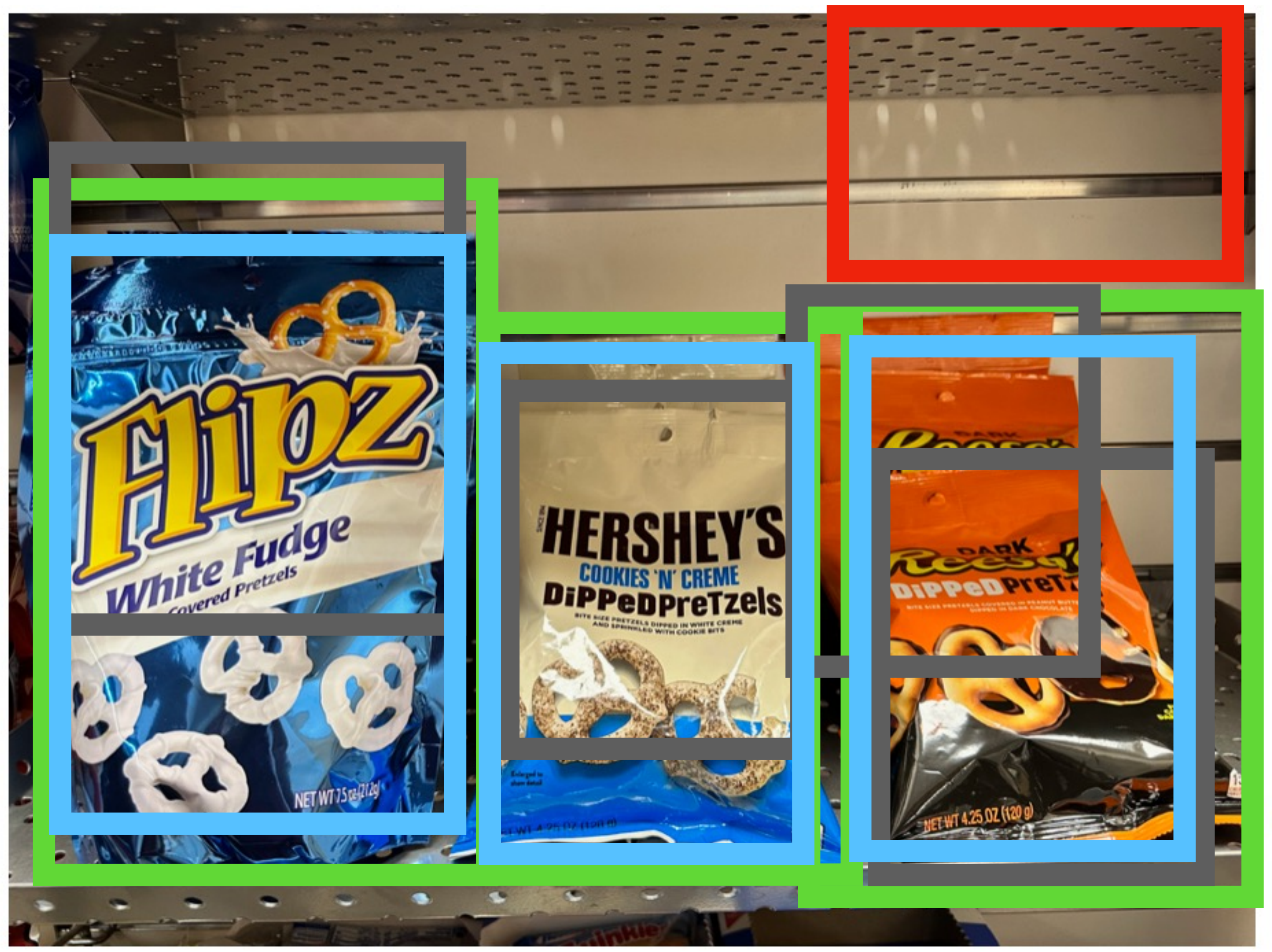
Input Image

Ground Truth

Region Proposals

R-CNN: Training

Input Image

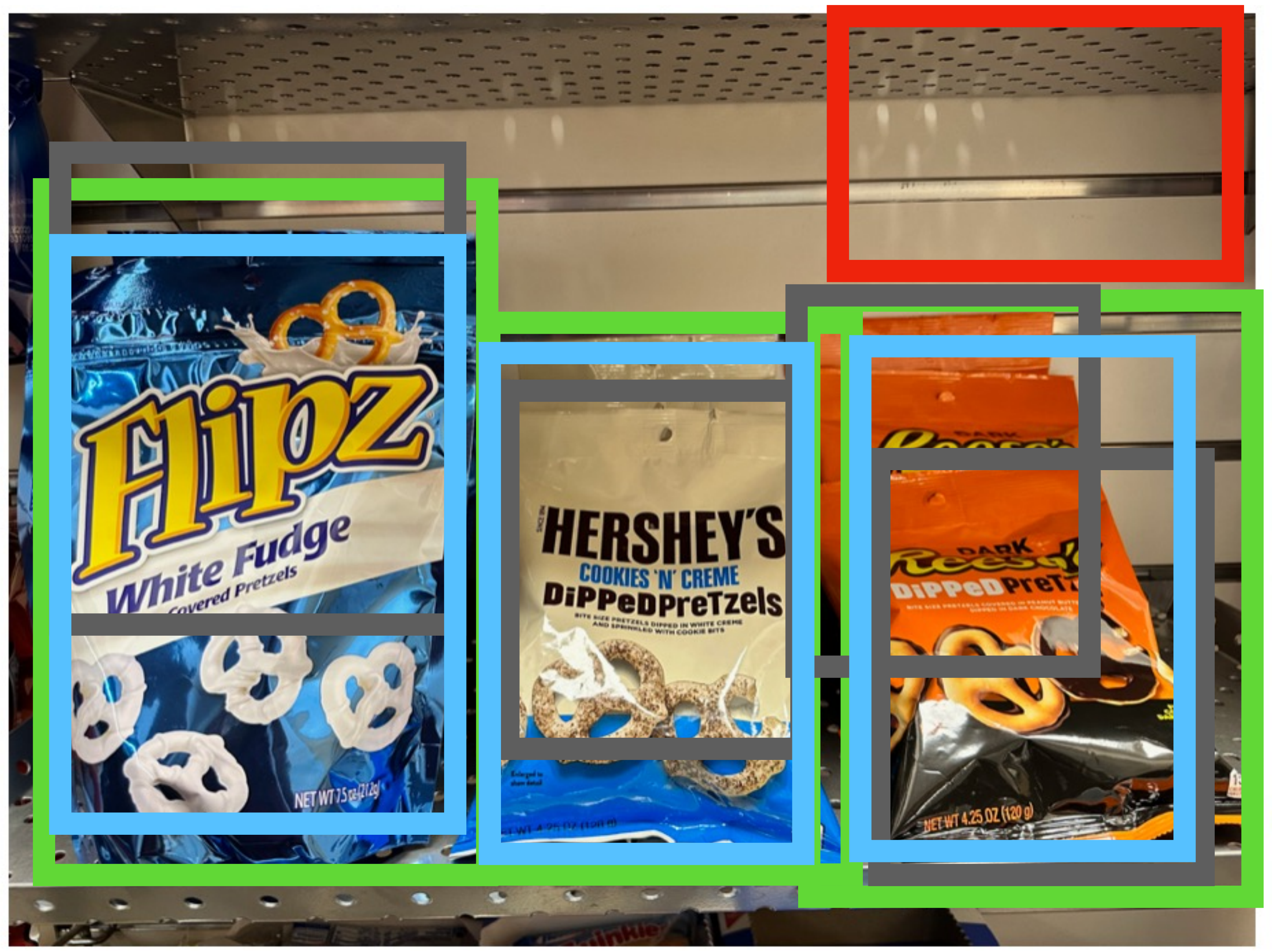


Ground Truth | Positive

Neutral | Negative

R-CNN: Training

Input Image



Ground Truth | Positive

Neutral | Negative

Categorize each region proposal as positive, negative or neutral based on overlap with the Ground truth boxes:

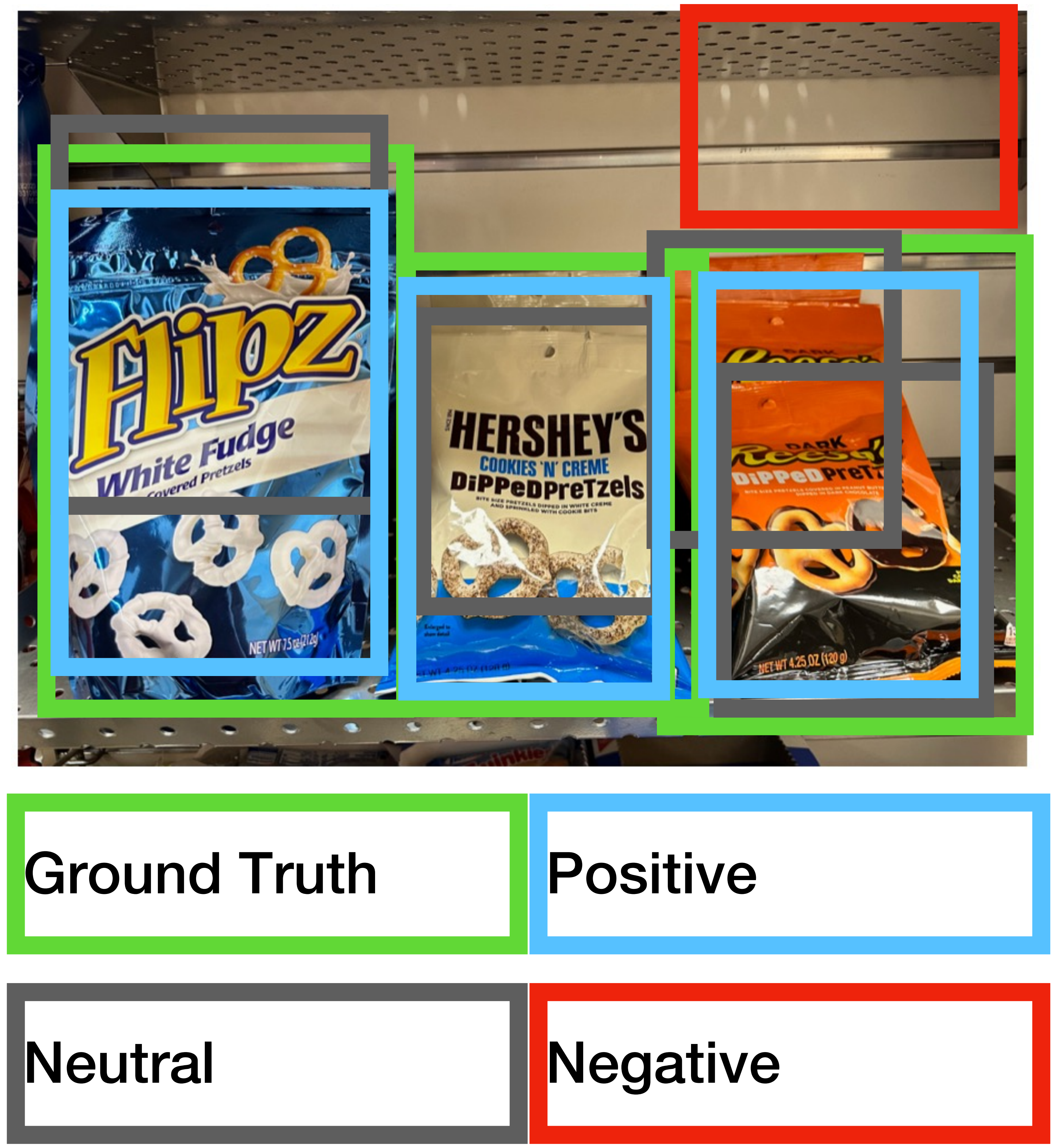
Positive: > 0.5 IoU with a GT box

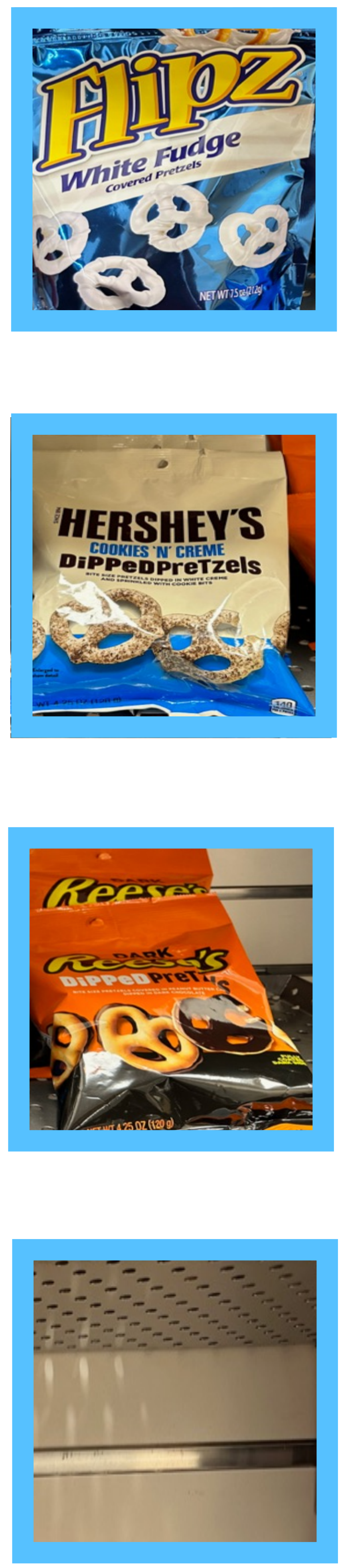
Negative: < 0.3 IoU with all GT boxes

Neutral: between 0.3 and 0.5 IoU with GT boxes

R-CNN: Training

Input Image



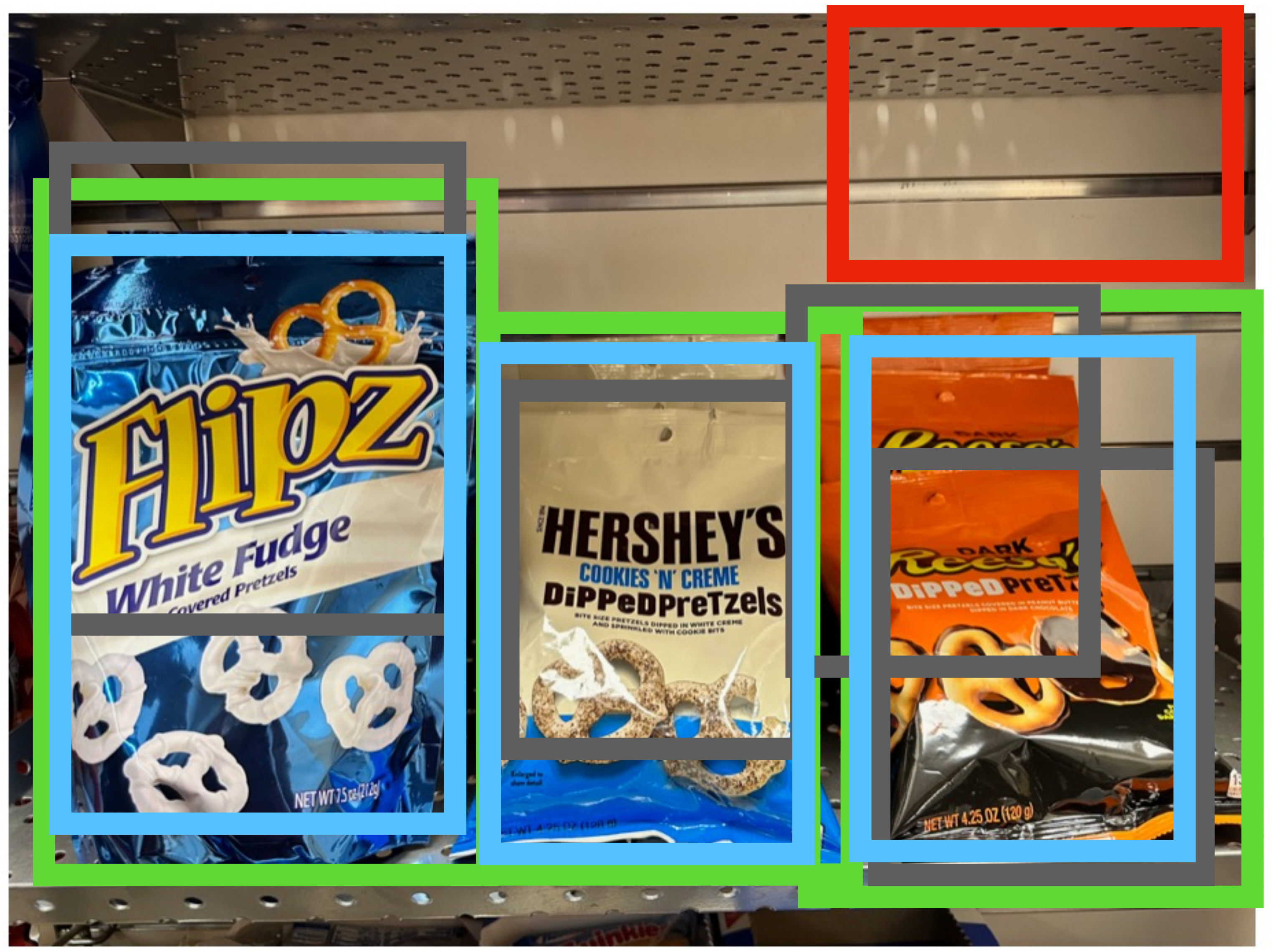


Crop pixels from each positive and negative proposal, resize to 224 x 224

Run each region through CNN
Positive regions: predict class and transform
Negative regions: just predict class

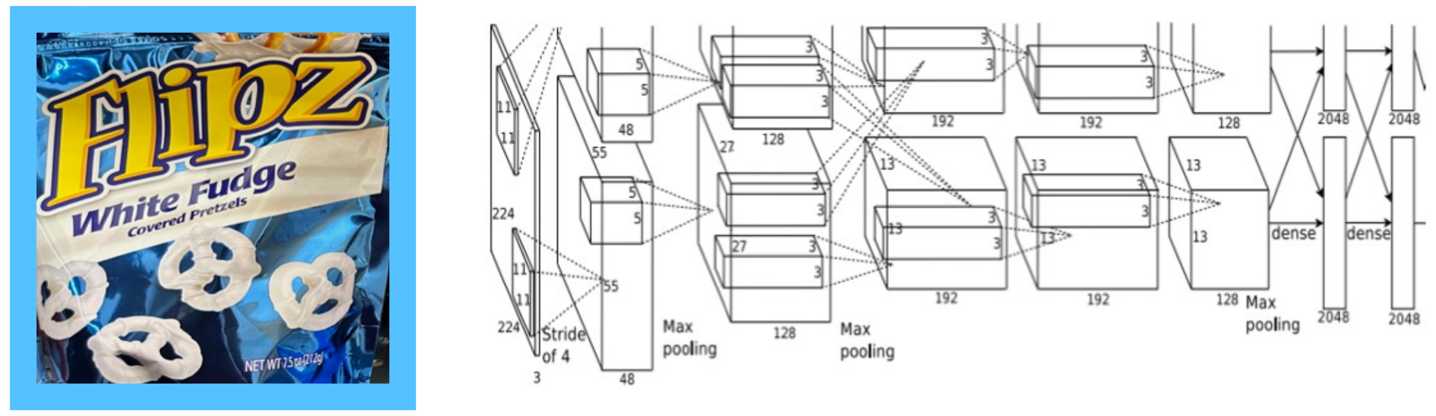
R-CNN: Training

Input Image



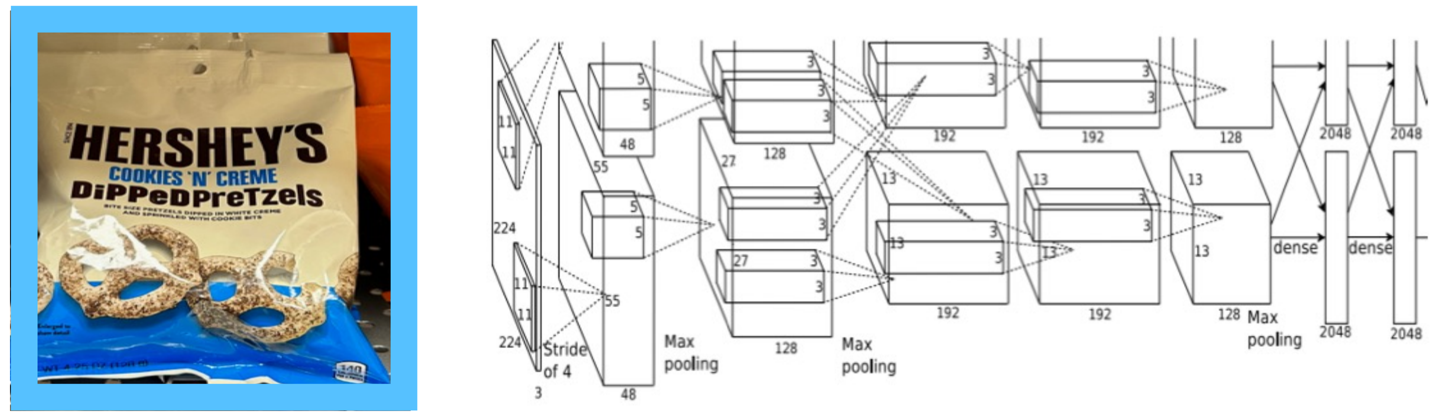
Ground Truth | Positive

Neutral | Negative



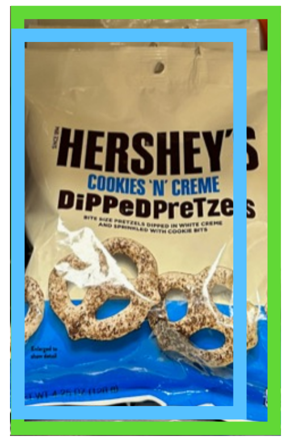
Class target: Flipz

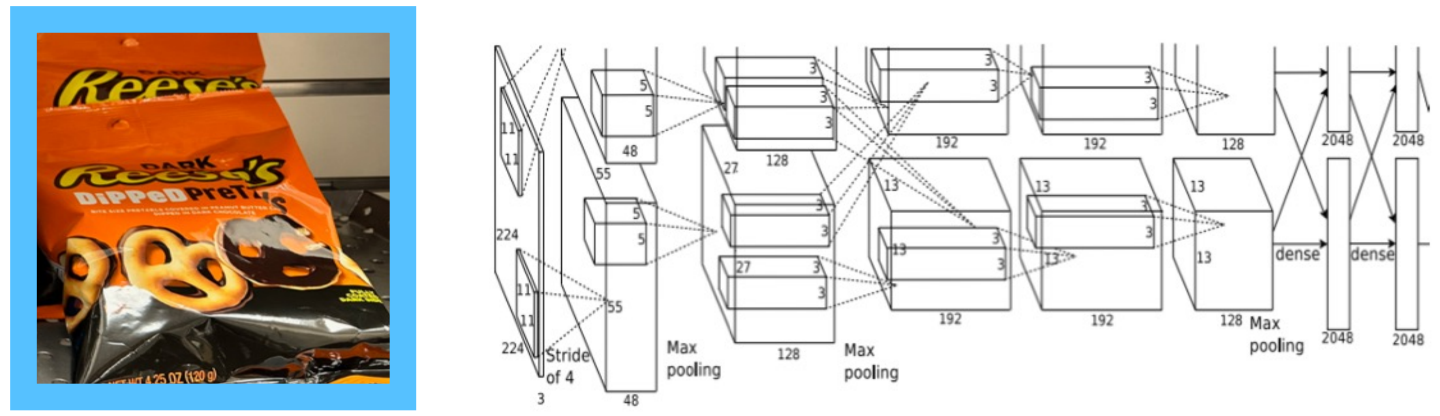
Box target:



Class target: Hershey's

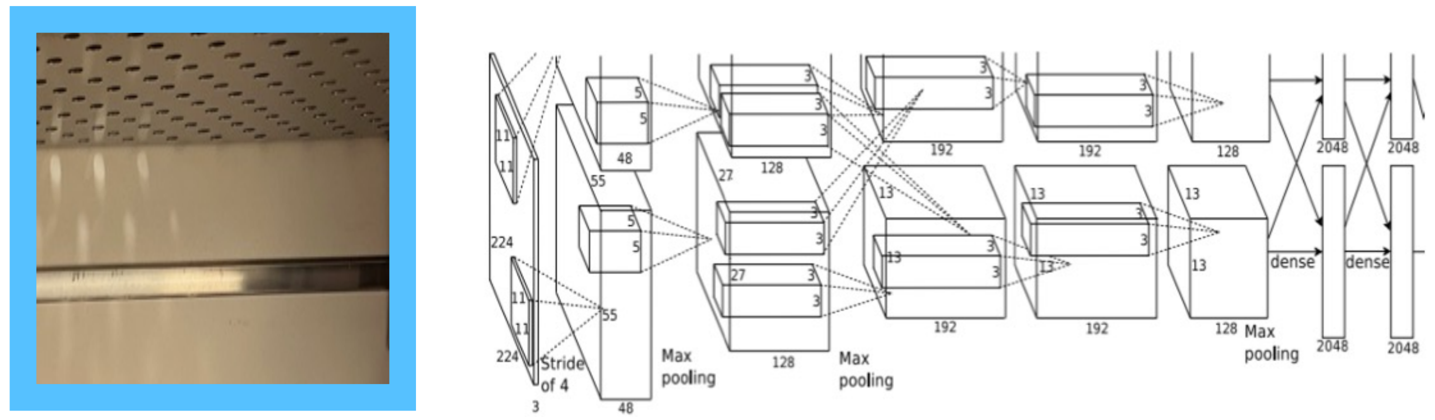
Box target:





Class target: Reese's

Box target:



Class target: Background

Box target: None

Run each region through CNN
Positive regions: predict class and transform
Negative regions: just predict class

R-CNN: Test time

Input Image

Region Proposals

Run proposal method:

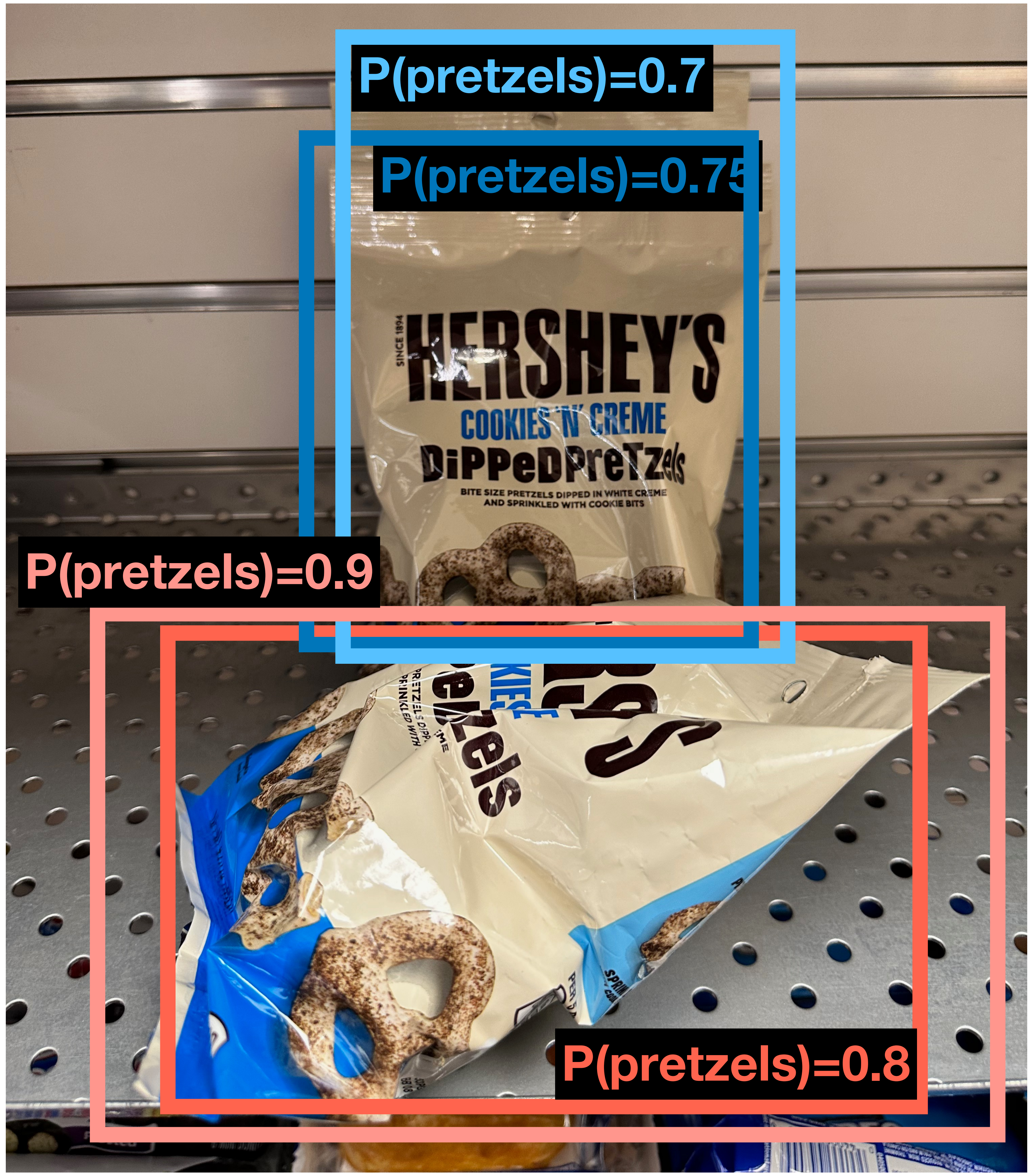
1. Run CNN on each proposal to get class scores, transforms
2. Threshold class scores to get a set of detections

2 Problems:

1. CNN often outputs overlapping boxes
2. How to set thresholds?

Overlapping Boxes

Problem: Object detectors often output many overlapping detections



Overlapping Boxes: Non-Max Suppression (NMS)

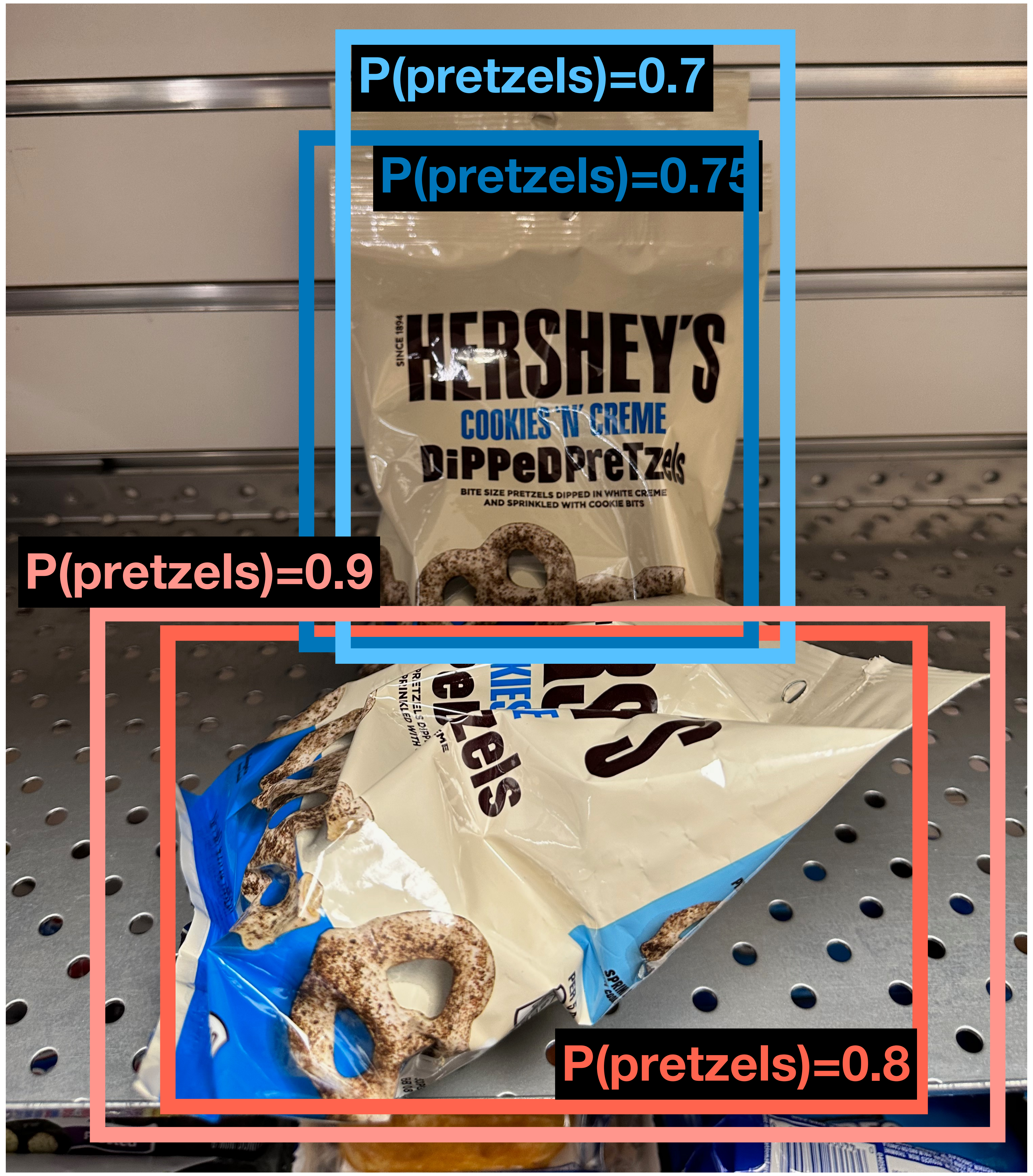
Problem: Object detectors often output many overlapping detections

Solution: Post-process raw detections using Non-Max Suppression (NMS)

1. Select next highest-scoring box

2. Eliminate lower-scoring boxes with IoU> threshold (e.g. 0.7)

3. If any boxes remain, GOTO 1



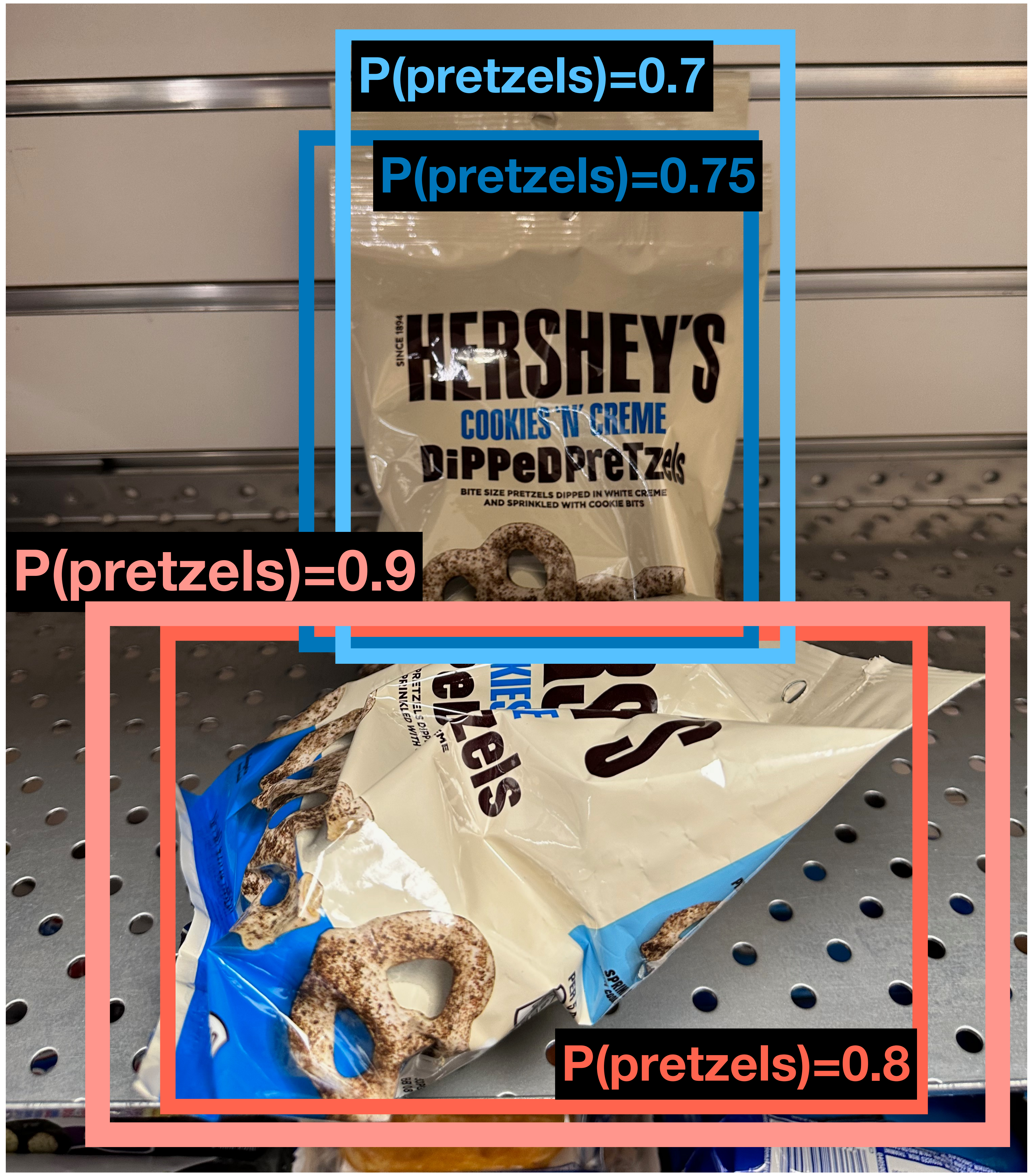
Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections

Solution: Post-process raw detections using Non-Max Suppression (NMS)

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU> threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(,) = 0.8
IoU(,) = 0.03
IoU(,) = 0.05



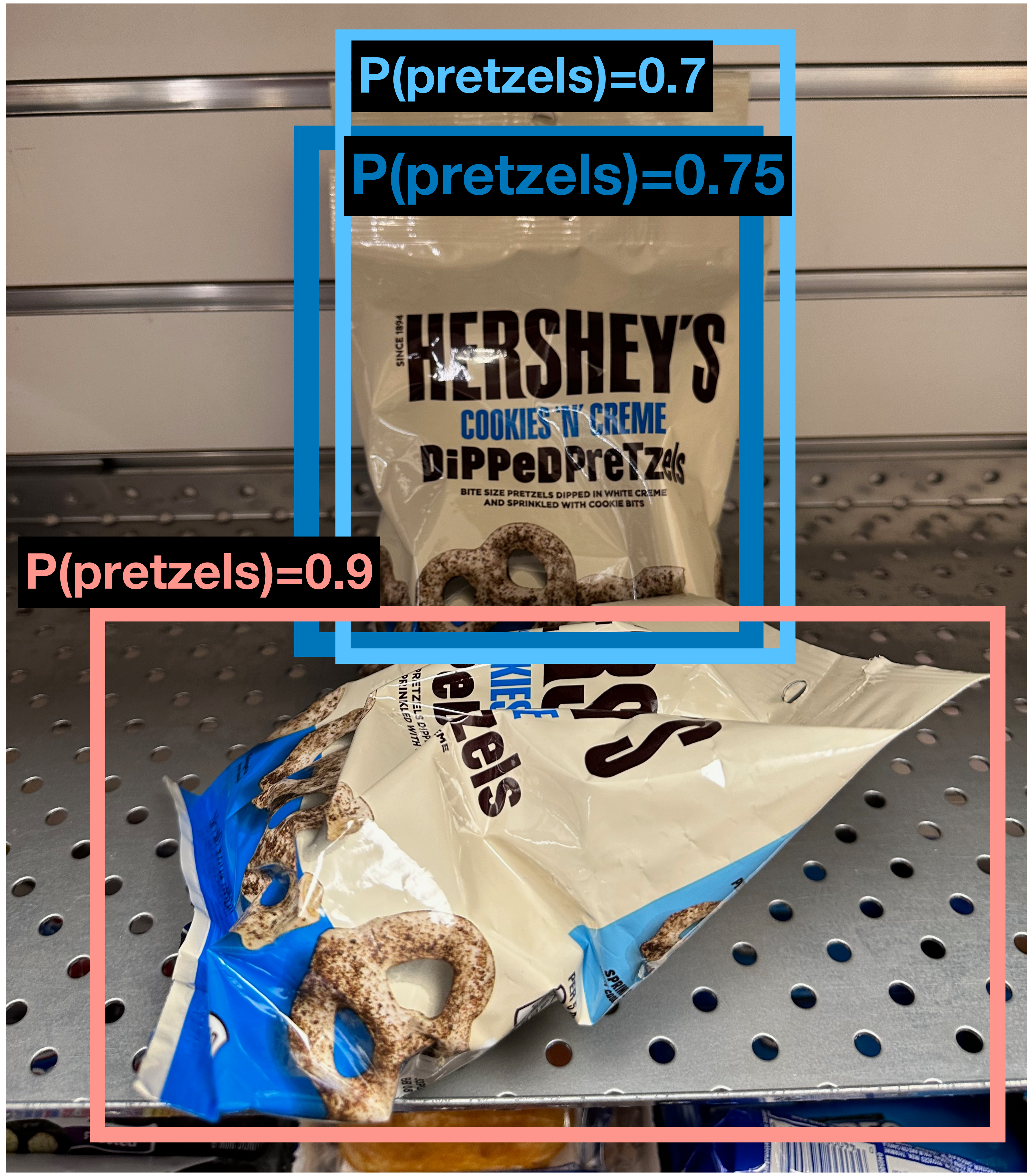
Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections

Solution: Post-process raw detections using Non-Max Suppression (NMS)

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU> threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

IoU(■, ■) = 0.85

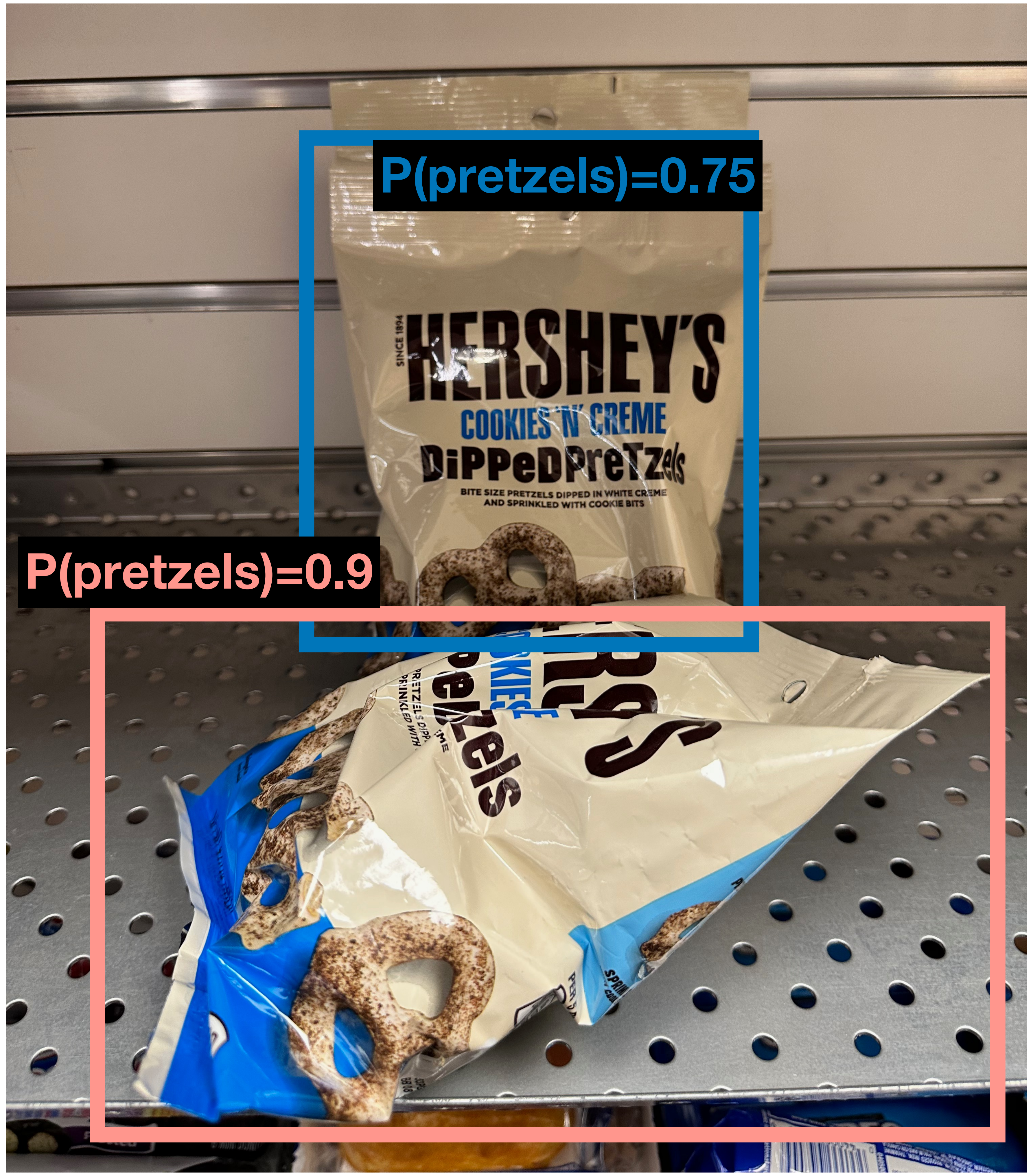


Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections

Solution: Post-process raw detections using Non-Max Suppression (NMS)

1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU> threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1



Overlapping Boxes: Non-Max Suppression (NMS)

Problem: Object detectors often output many overlapping detections

Solution: Post-process raw detections using Non-Max Suppression (NMS)
1. Select next highest-scoring box
2. Eliminate lower-scoring boxes with IoU> threshold (e.g. 0.7)
3. If any boxes remain, GOTO 1

Problem: NMS may eliminate "good" boxes when objects are highly overlapping… no good solution

DeepRob

Lecture 8
CNN Architectures and Object Detection
University of Michigan | Department of Robotics

DeepRob