THE UNIVERSITY OF BRITISH COLUMBIA

Topics in AI (CPSC 532S):
Multimodal Learning with Vision, Language and Sound

Lecture 6: Convolutional Neural Networks (Part 3)

Logistics:

Assignment 2 is due on **Friday** (will postpone to Monday)

Assignment 2 check out Piazza for debugging hints and some guides

TA office hours are Tuesdays (today) @ 3pm

My office hours are Fridays

Will make slides available **later today**

Revisit **Layers** we Learned About

Fully Connected:

— **Not** invariant to any transformations

— **Not** equivariant to any transformations

Convolutional:

— **Not** invariant to any transformations

— Convolution is **translation equivariant**

Note: convolution can “learn” to encode positional information when padding is used

Revisit **Layers** we Learned About

Input to Layer 1:

0	0	0	0	0	0	0	0	0
0	23	25	67	89	13	64	35	0
0	74	15	46	67	64	36	14	0
0	67	14	46	86	75	43	16	0
0	67	69	69	74	34	56	15	0
0	46	37	95	72	27	35	45	0
0	15	26	28	16	48	89	12	0
0	23	11	46	78	18	23	12	0
0	0	0	0	0	0	0	0	0

CNN Layer 1:

Weight

0	0	0
1	0	0
0	0	0

Kernel

Bias

1

Revisit **Layers** we Learned About

Output of Layer 1:

0	0	0	0	0	0	0	0	0
0	1	24	26	68	90	14	65	0
0	1	75	16	47	68	65	37	0
0	1	68	15	47	87	76	44	0
0	1	68	70	70	75	35	57	0
0	1	47	38	96	73	28	36	0
0	1	16	27	29	17	49	90	0
0	1	24	12	47	79	19	24	0
0	0	0	0	0	0	0	0	0

CNN Layer 1:

Weight

0	0	0
1	0	0
0	0	0

Kernel

Bias

1

Revisit **Layers** we Learned About

Input to Layer 2:

0	0	0	0	0	0	0	0	0
0	1	24	26	68	90	14	65	0
0	1	75	16	47	68	65	37	0
0	1	68	15	47	87	76	44	0
0	1	68	70	70	75	35	57	0
0	1	47	38	96	73	28	36	0
0	1	16	27	29	17	49	90	0
0	1	24	12	47	79	19	24	0
0	0	0	0	0	0	0	0	0

CNN Layer 2:

Weight

0	0	0
1	0	0
0	0	0

Kernel

Bias

1

Revisit **Layers** we Learned About

Output of Layer 2:

0	0	0	0	0	0	0	0	0
0	1	2	25	27	69	91	15	0
0	1	2	76	17	48	69	66	0
0	1	2	69	16	48	88	77	0
0	1	2	69	71	71	76	36	0
0	1	2	48	39	97	74	29	0
0	1	2	17	28	30	18	50	0
0	1	2	25	13	48	80	20	0
0	0	0	0	0	0	0	0	0

CNN Layer 2:

Weight

0	0	0
1	0	0
0	0	0

Kernel

Bias

1

Revisit **Layers** we Learned About

Output of Layer 7:

0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	0
0	1	2	3	4	5	6	7	0
0	1	2	3	4	5	6	7	0
0	1	2	3	4	5	6	7	0
0	1	2	3	4	5	6	7	0
0	1	2	3	4	5	6	7	0
0	1	2	3	4	5	6	7	0
0	0	0	0	0	0	0	0	0

CNN Layer 7:

Weight

0	0	0
1	0	0
0	0	0

Kernel

Bias

1

Revisit **Layers** we Learned About

Input to Layer 1:

0	0	0	0	0	0	0	0	0
0	23	25	67	89	13	64	35	0
0	74	15	46	67	64	36	14	0
0	67	14	46	86	75	43	16	0
0	67	69	69	74	34	56	15	0
0	46	37	95	72	27	35	45	0
0	15	26	28	16	48	89	12	0
0	23	11	46	78	18	23	12	0
0	0	0	0	0	0	0	0	0

CNN Layer 1:

Weight

0	1	0
0	0	0
0	0	0

Kernel

Bias

1

Do CNNs Capture **Positional** Information?

PosENet = Simple **1 layer** convolutional neural net with one **3 x 3 kernel**

(trained to minimize mean squared error)

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

PosENet = Simple **1 layer** convolutional neural net with one **3 x 3 kernel**

(trained to minimize mean squared error)

SPC = Spearmen Correlation

MAE = Mean Squared Error

	Model	Black	
		SPC	MAE
H	PosENet	.0	.251

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

PosENet = Simple **1 layer** convolutional neural net with one **3 x 3 kernel**

(trained to minimize mean squared error)

SPC = Spearmen Correlation

MAE = Mean Squared Error

	Model	Black	
		SPC	MAE
H	PosENet	.0	.251
V	PosENet	.0	.251

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

PosENet = Simple **1 layer** convolutional neural net with one **3 x 3 kernel**

(trained to minimize mean squared error)

SPC = Spearmen Correlation

MAE = Mean Squared Error

	Model	Black		White	
		SPC	MAE	SPC	MAE
H	PosENet	.0	.251	.0	.251
V	PosENet	.0	.251	.0	.251

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

PosENet = Simple **1 layer** convolutional neural net with one **3 x 3 kernel**

(trained to minimize mean squared error)

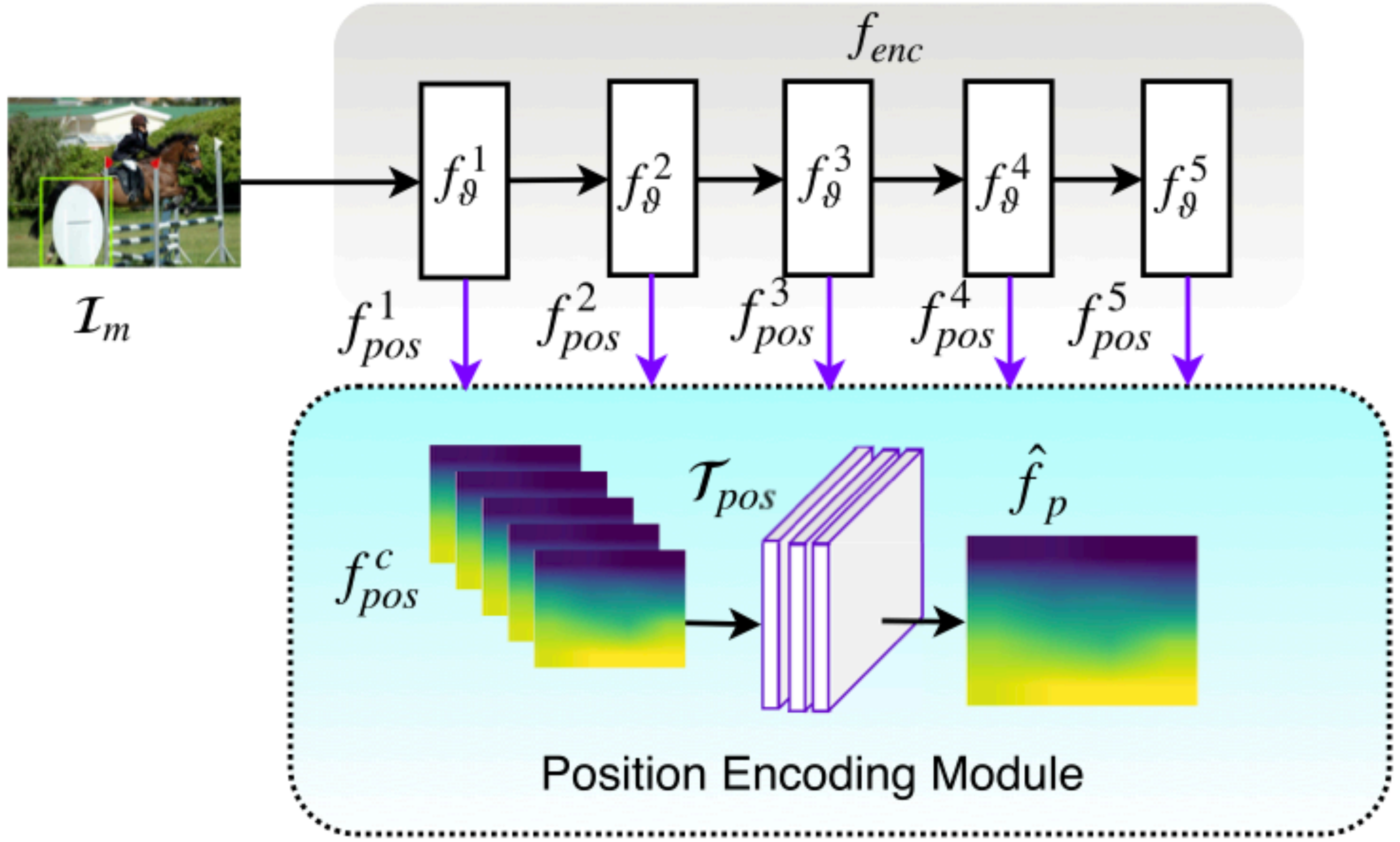
SPC = Spearmen Correlation

MAE = Mean Squared Error

	Model	PASCAL-S		Black		White	
		SPC	MAE	SPC	MAE	SPC	MAE
H	PosENet	.012	.251	.0	.251	.0	.251
V	PosENet	.131	.248	.0	.251	.0	.251

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?



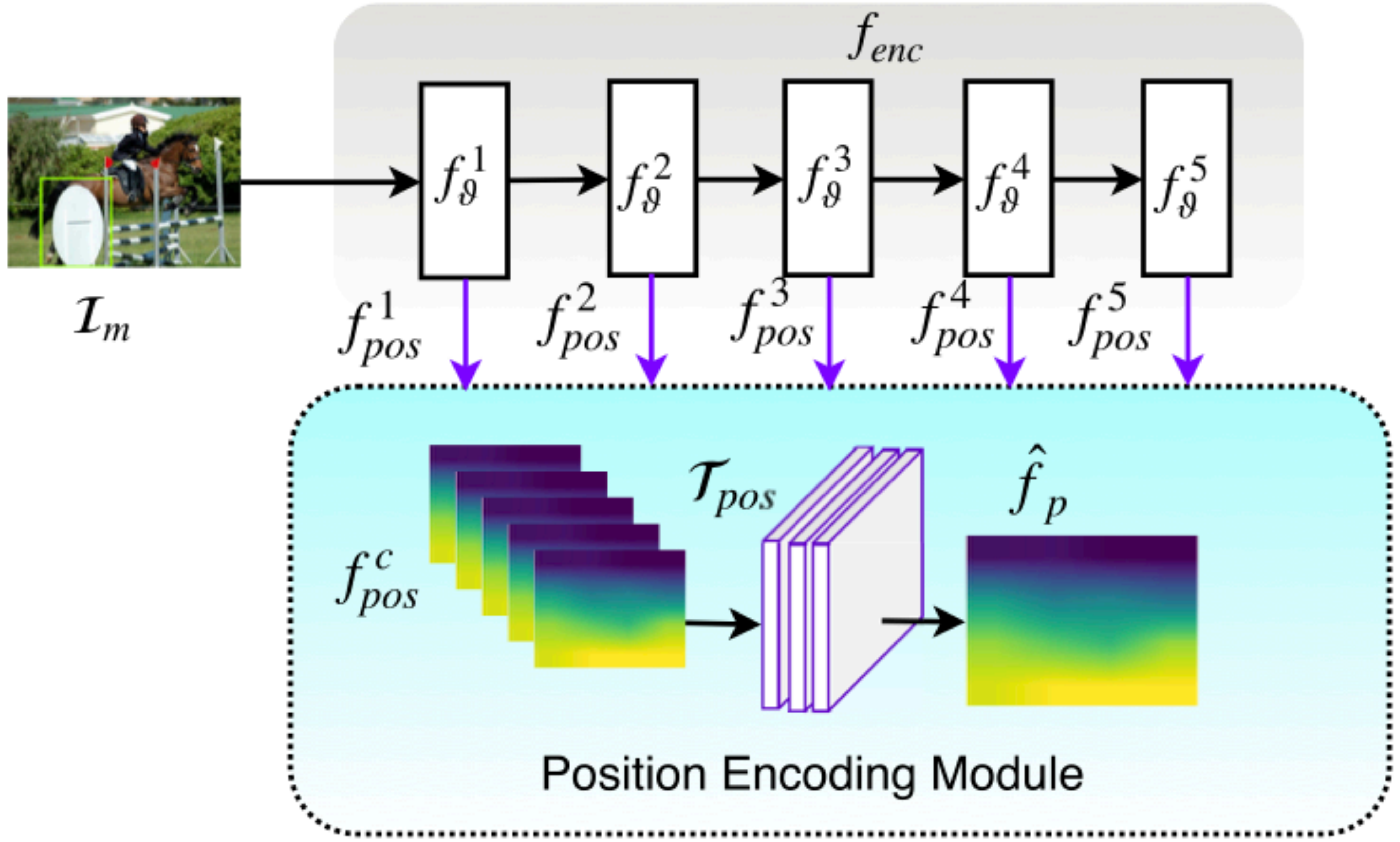
SPC = Spearmen Correlation

MAE = Mean Squared Error

	Model	PASCAL-S		Black		White	
		SPC	MAE	SPC	MAE	SPC	MAE
H	PosENet	.012	.251	.0	.251	.0	.251
V	PosENet	.131	.248	.0	.251	.0	.251

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?



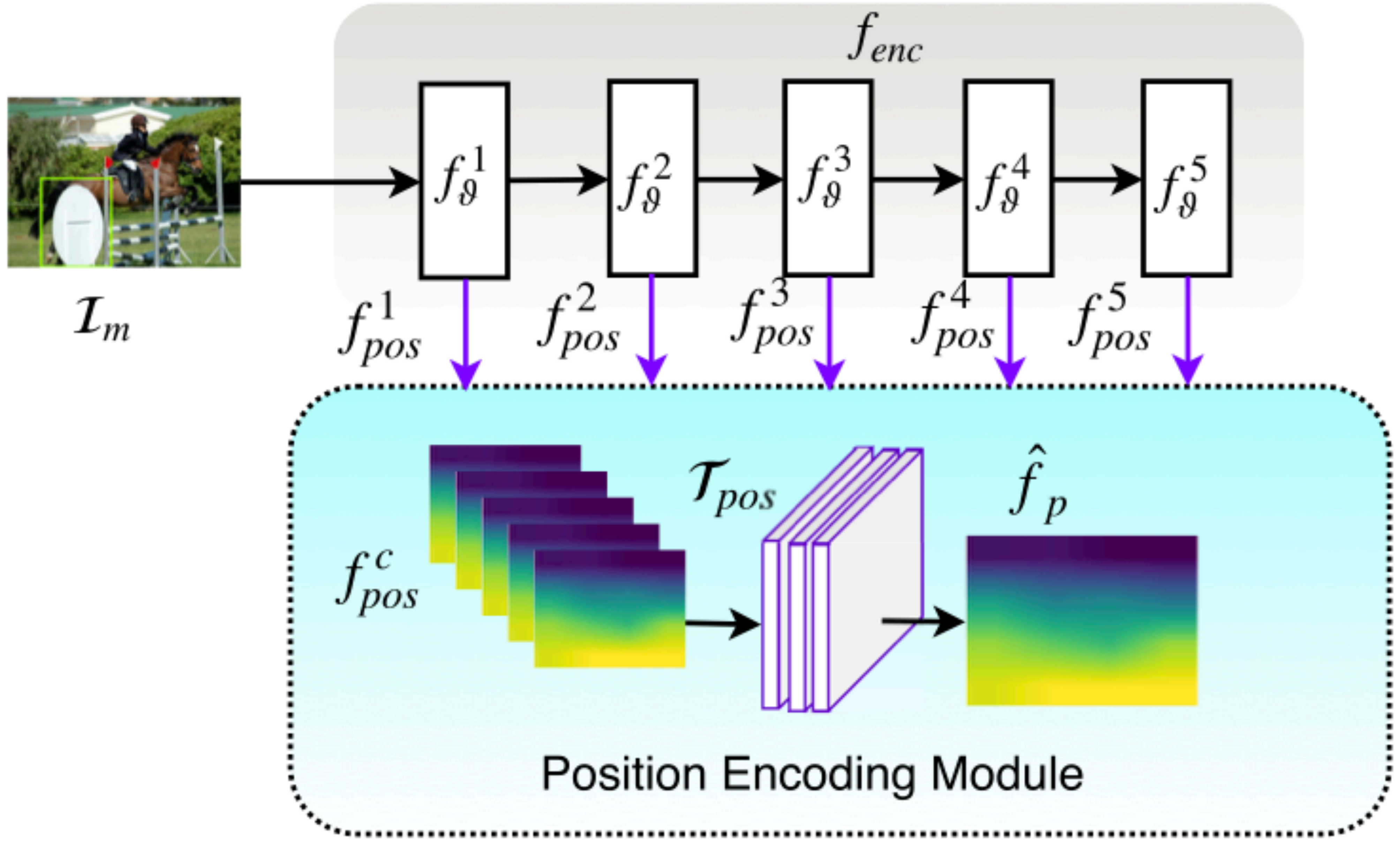
SPC = Spearmen Correlation

MAE = Mean Squared Error

	Model	PASCAL-S		Black		White	
		SPC	MAE	SPC	MAE	SPC	MAE
H	PosENet	.012	.251	.0	.251	.0	.251
	VGG	.742	.149	.751	.164	.873	.157
V	PosENet	.131	.248	.0	.251	.0	.251
	VGG	.816	.129	.846	.146	.927	.138

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?



SPC = Spearmen Correlation

MAE = Mean Squared Error

	Model	PASCAL-S		Black		White	
		SPC	MAE	SPC	MAE	SPC	MAE
H	PosENet	.012	.251	.0	.251	.0	.251
	VGG	.742	.149	.751	.164	.873	.157
	ResNet	.933	.084	.987	.080	.994	.078
V	PosENet	.131	.248	.0	.251	.0	.251
	VGG	.816	.129	.846	.146	.927	.138
	ResNet	.951	.083	.978	.069	.979	.072

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

This result is robust even if we use 1 x 1 CNN kernel

	Kernel	PosENet		VGG	
		SPC	MAE	SPC	MAE
H	1×1	.013	.251	.542	.196
	3×3	.012	.251	.742	.149
	7×7	.060	.250	.828	.120
G	1×1	.017	.188	.724	.127
	3×3	-.001	.233	.814	.109
	7×7	.068	.187	.816	.111
HS	1×1	-.004	.628	.317	.576
	3×3	-.001	.723	.405	.556
	7×7	.002	.628	.487	.532

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

More positional information is stored in **deeper** layer feature maps

	Method	f^1_{pos}	f^2_{pos}	f^3_{pos}	f^4_{pos}	f^5_{pos}	SPC	MAE
H	**VGG**	✓					.101	.249
			✓				.344	.225
				✓			.472	.203
					✓		.610	.181
						✓	.657	.177
		✓	✓	✓	✓	✓	.742	.149
G	**VGG**	✓					.241	.182
			✓				.404	.168
				✓			.588	.146
					✓		.653	.138
						✓	.693	.135
		✓	✓	✓	✓	✓	.814	.109

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

Where does positional information coming from?

Model	H		G		HS	
	SPC	MAE	SPC	MAE	SPC	MAE
VGG16	.742	.149	.814	.109	.405	.556
VGG16 w/o. *padding*	.381	.223	.359	.174	.011	.628

(padding appears to contribute significantly to encoded positional information)

[Islam et al, ICLR 2020]

Do CNNs Capture **Positional** Information?

Where does positional information coming from?

Model	H		G		HS	
	SPC	MAE	SPC	MAE	SPC	MAE
VGG16	.742	.149	.814	.109	.405	.556
VGG16 w/o. *padding*	.381	.223	.359	.174	.011	.628

(padding appears to contribute significantly to encoded positional information)

Why is it there?

Model	mIoU (%)
VGG w/o padding	12.3
VGG	23.1

(results of semantic segmentation with and without padding)

[Islam et al, ICLR 2020]

Transfer Learning with CNNs

Common "Wisdom": You need a lot of data to train a CNN

Transfer Learning with CNNs

Common “Wisdom”: You need a lot of data to train a CNN

Solution: Transfer learning — taking a model trained on the task that has lots of data and adopting it to the task that may not

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

Common “Wisdom”: You need a lot of data to train a CNN

Solution: Transfer learning — taking a model trained on the task that has lots of data and adopting it to the task that may not

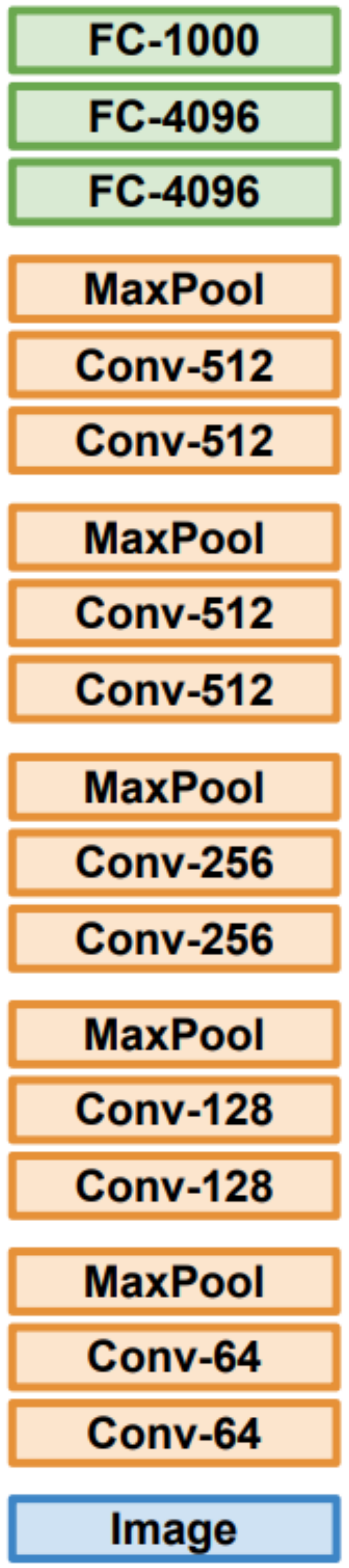
This strategy is PERVASIVE.

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

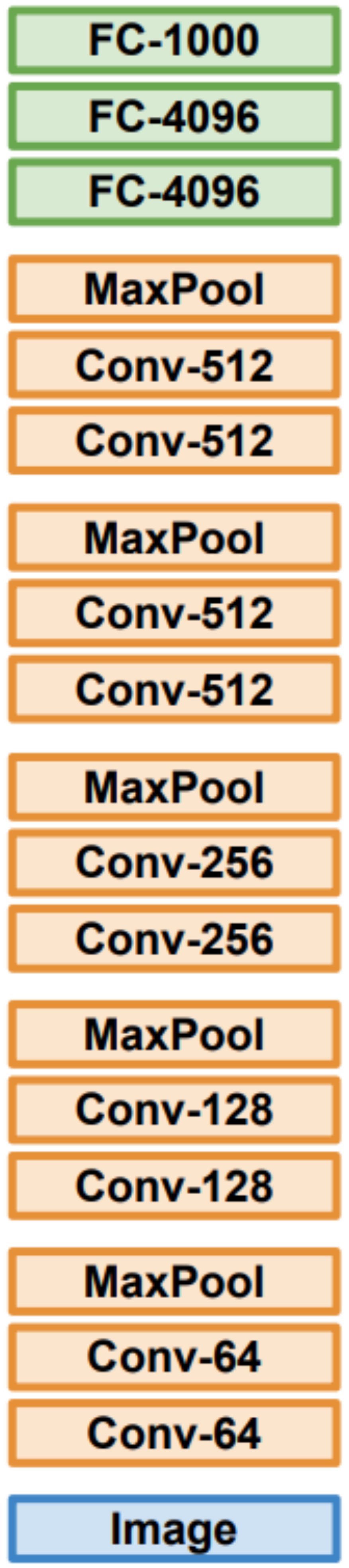
Train on **ImageNet**



Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**

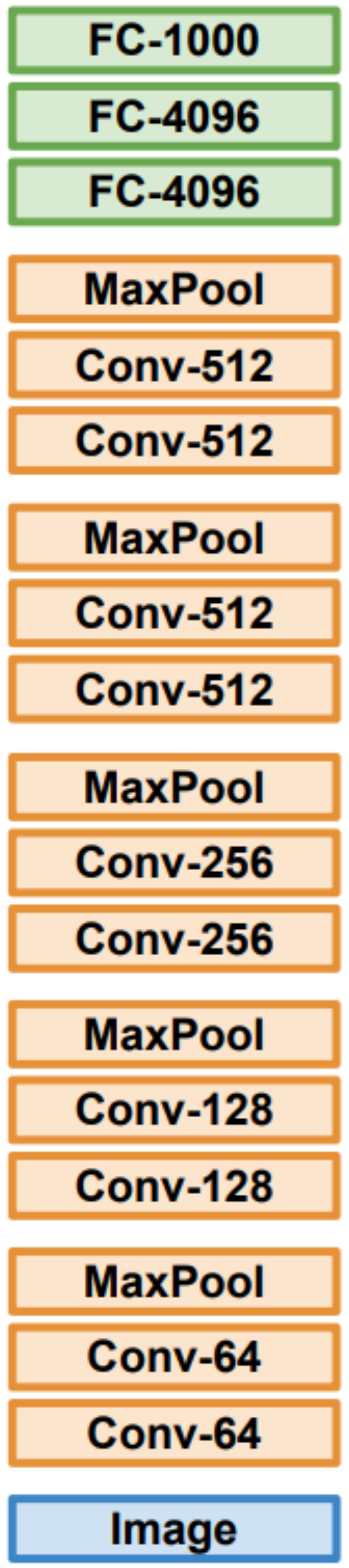


Why on **ImageNet**?

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



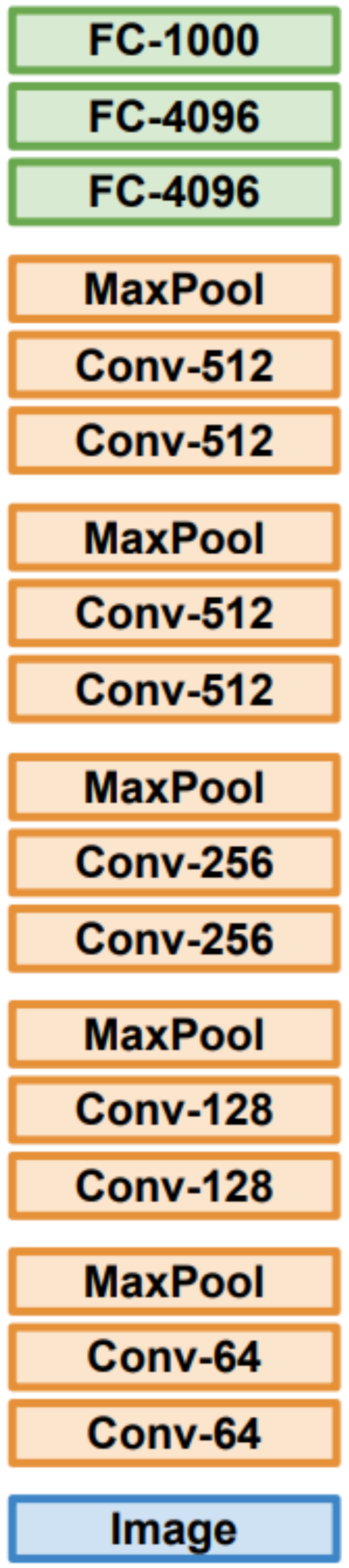
Why on **ImageNet**?

- Convenience, lots of **data**
- We know how to **train these well**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



Why on **ImageNet**?

- Convenience, lots of **data**
- We know how to **train these well**

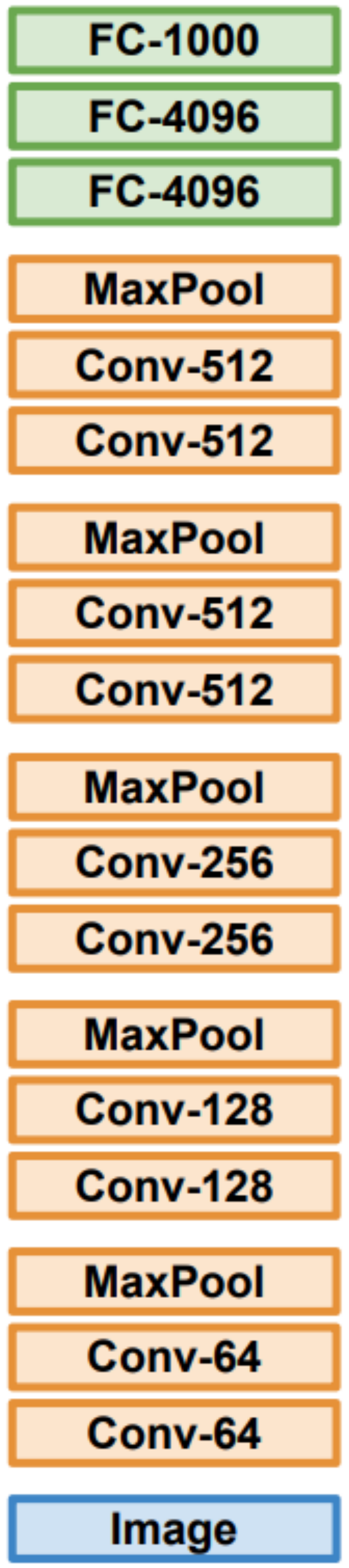
However, for some tasks we would need to start with something else (e.g., videos for optical flow)

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



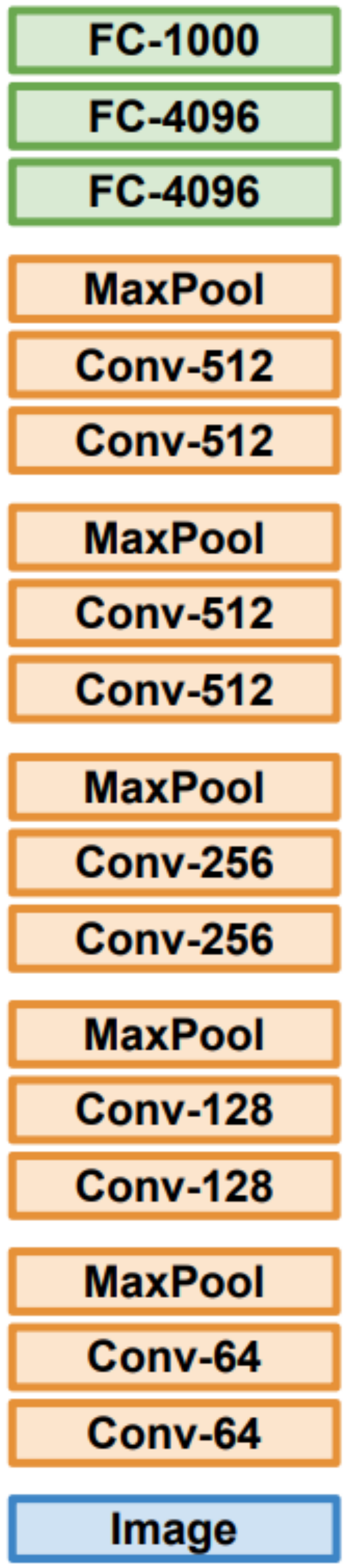
Small dataset with C classes

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

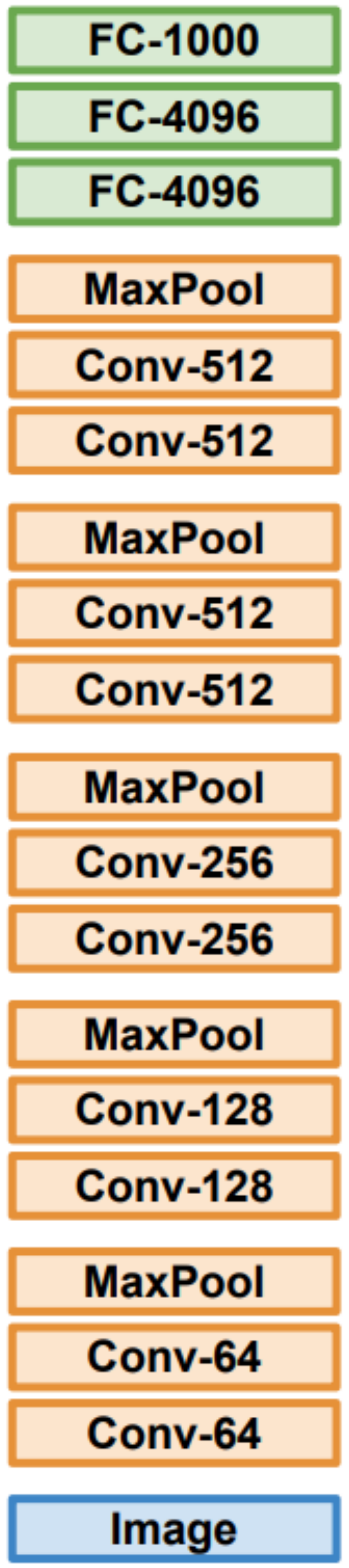
Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



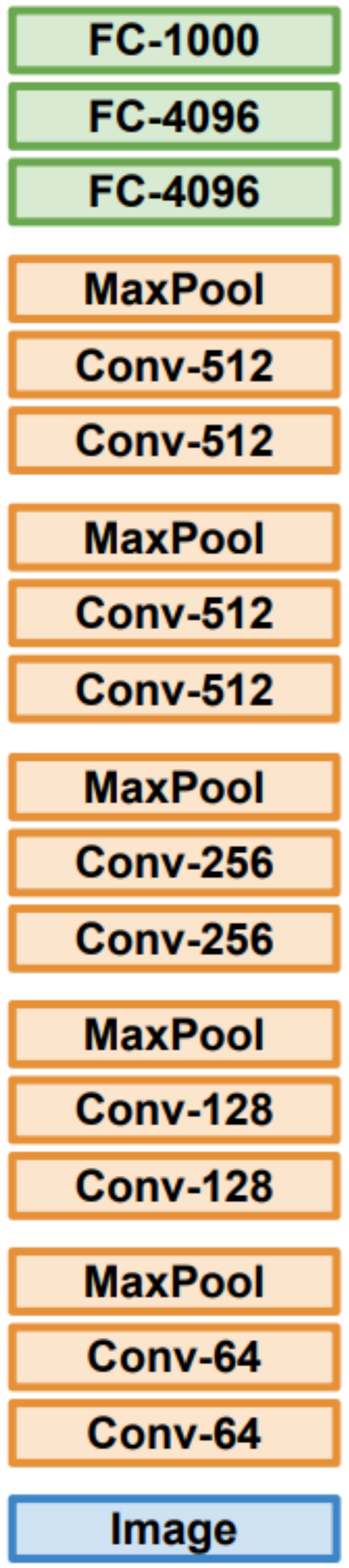
Small dataset with C classes



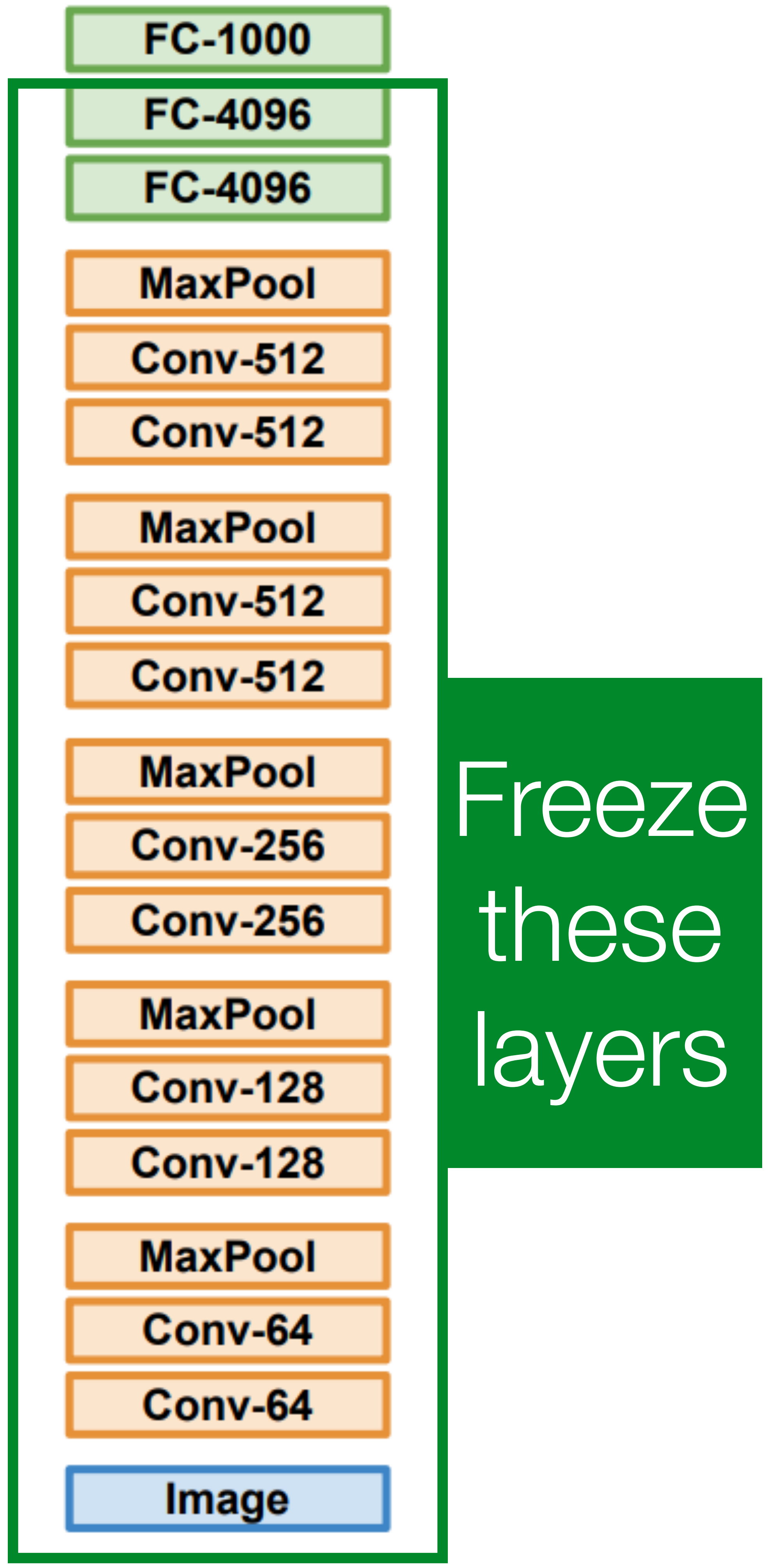
Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



Small dataset with C classes

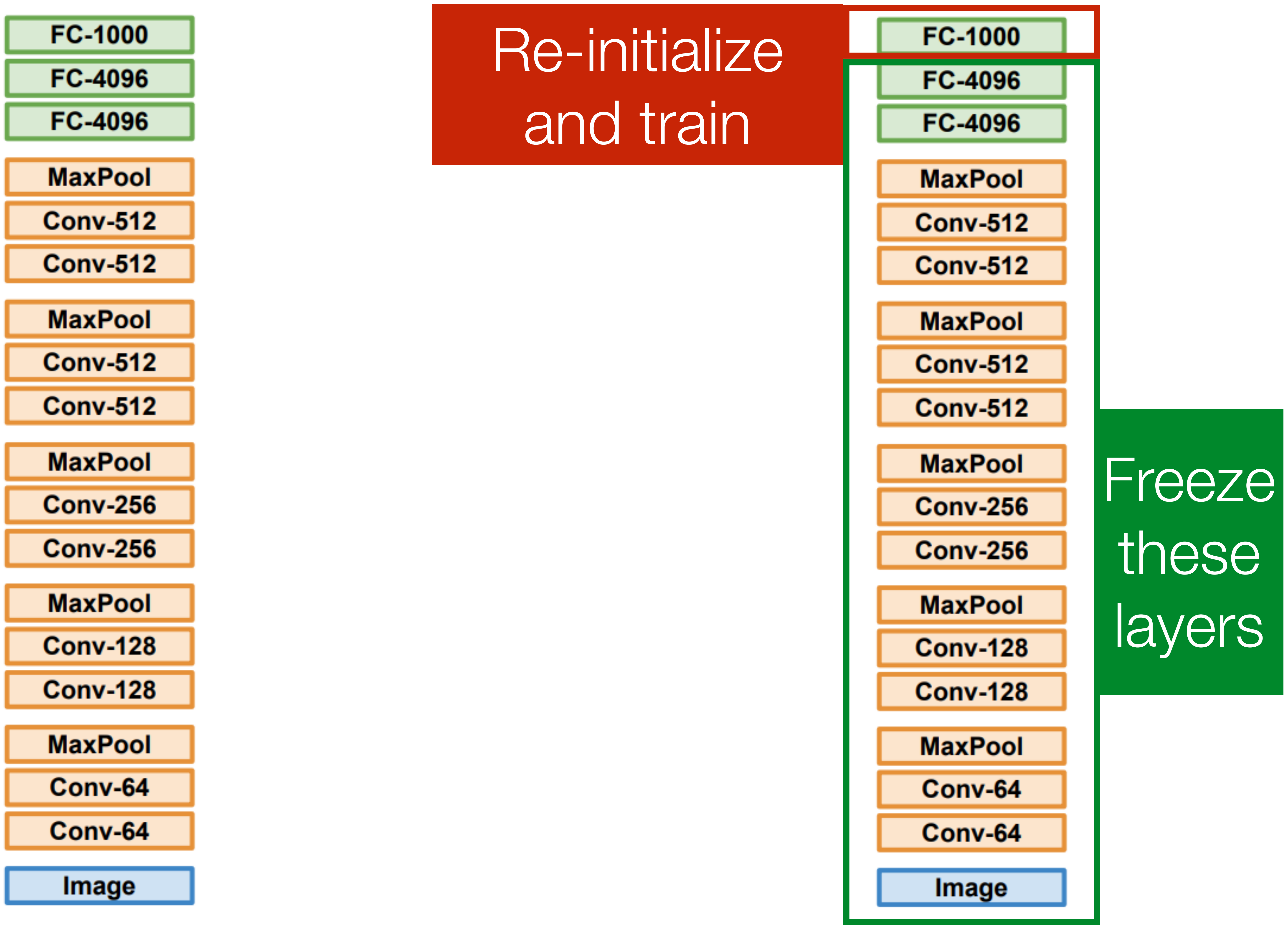


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

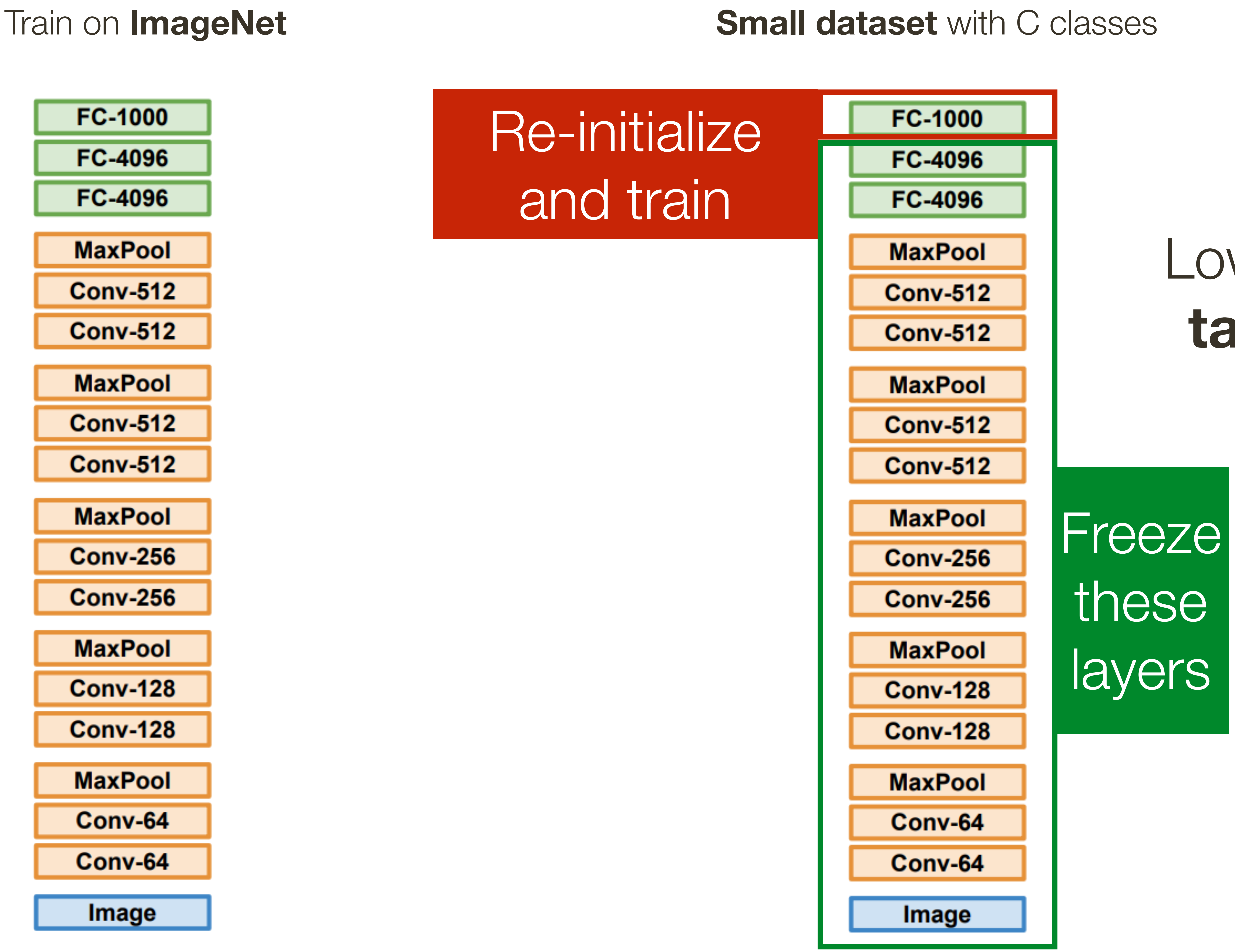
Train on **ImageNet** | **Small dataset** with C classes



* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]



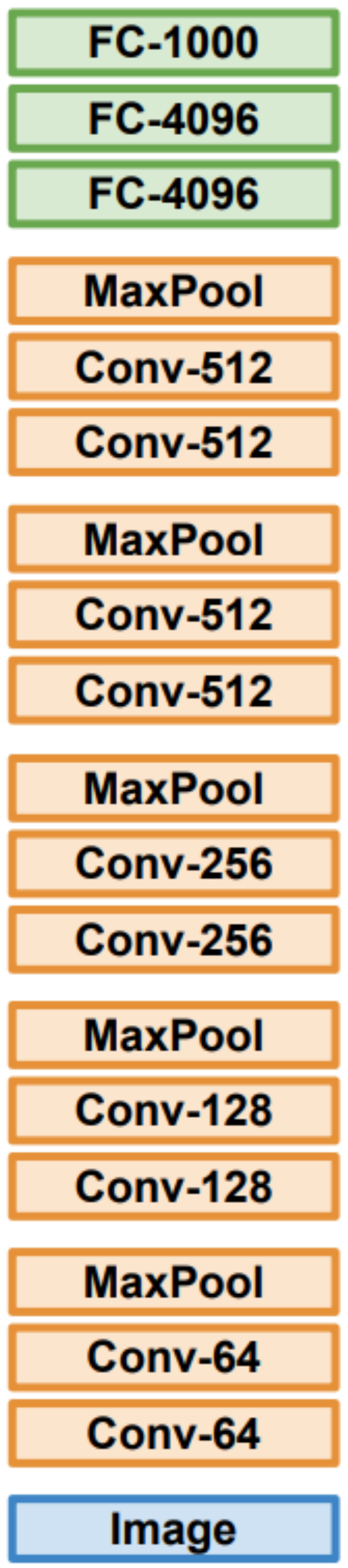
Lower levels of the CNN are at **task independent** anyways

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

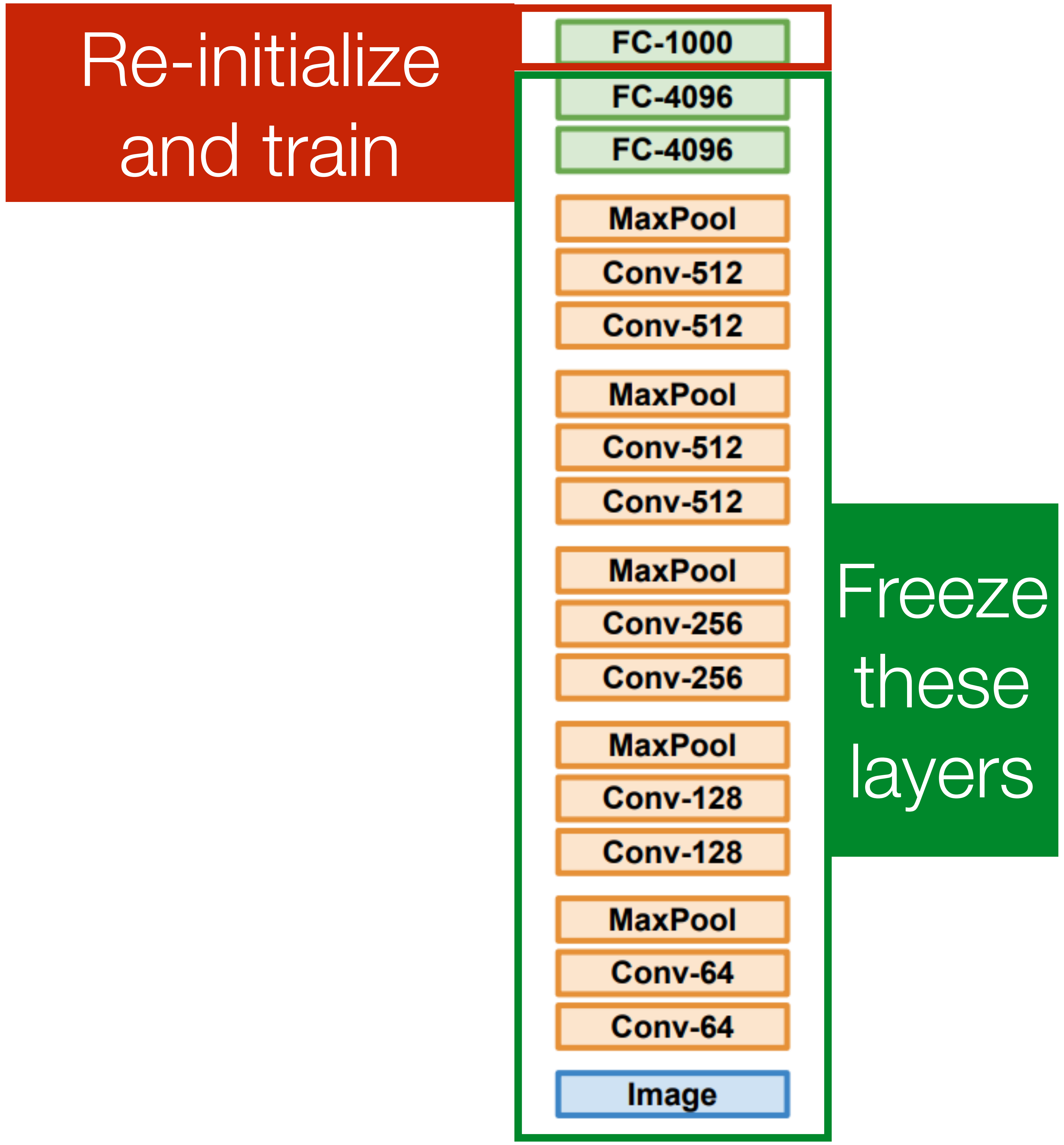
Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet**



Small dataset with C classes

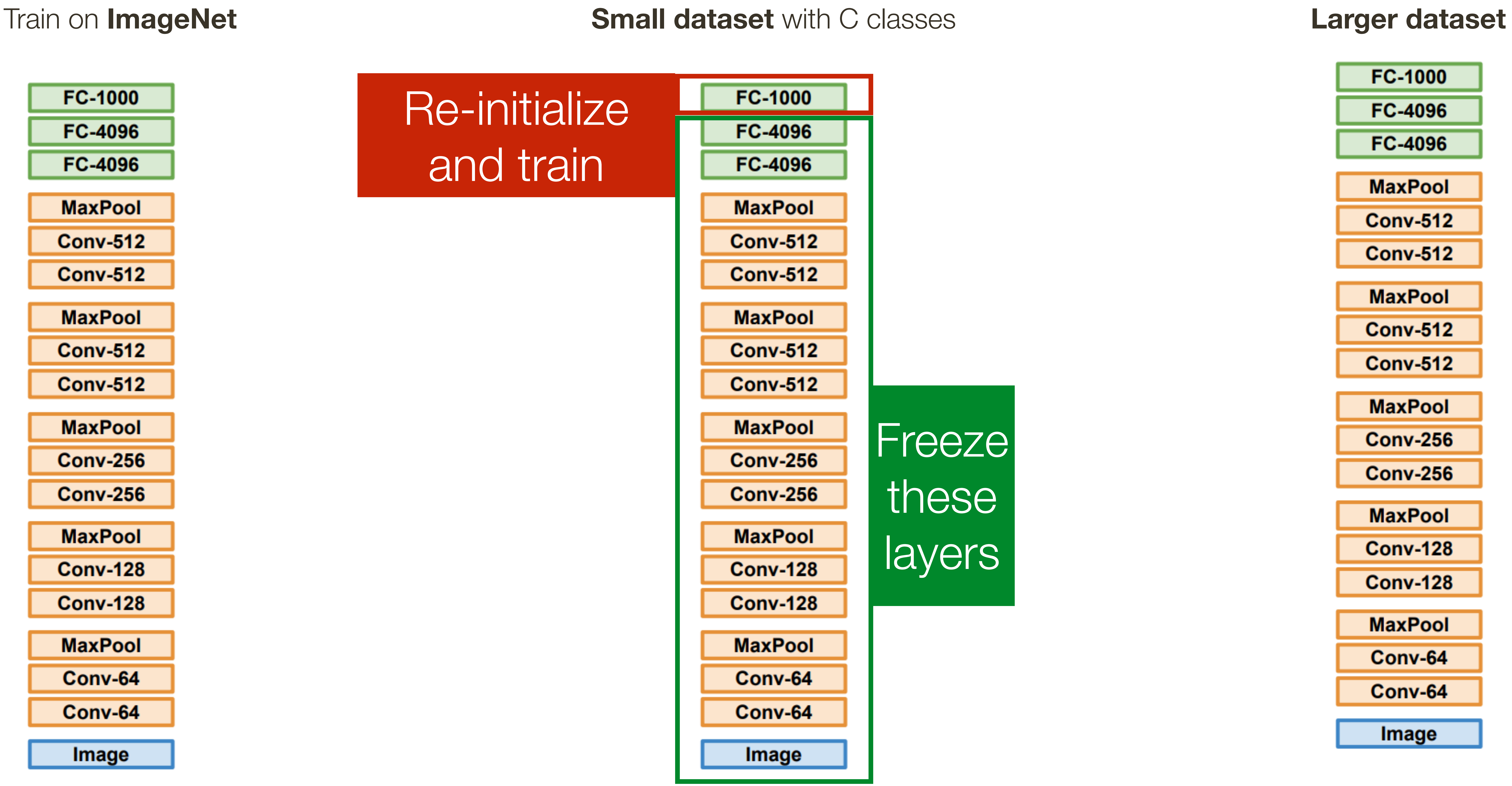


Larger dataset

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

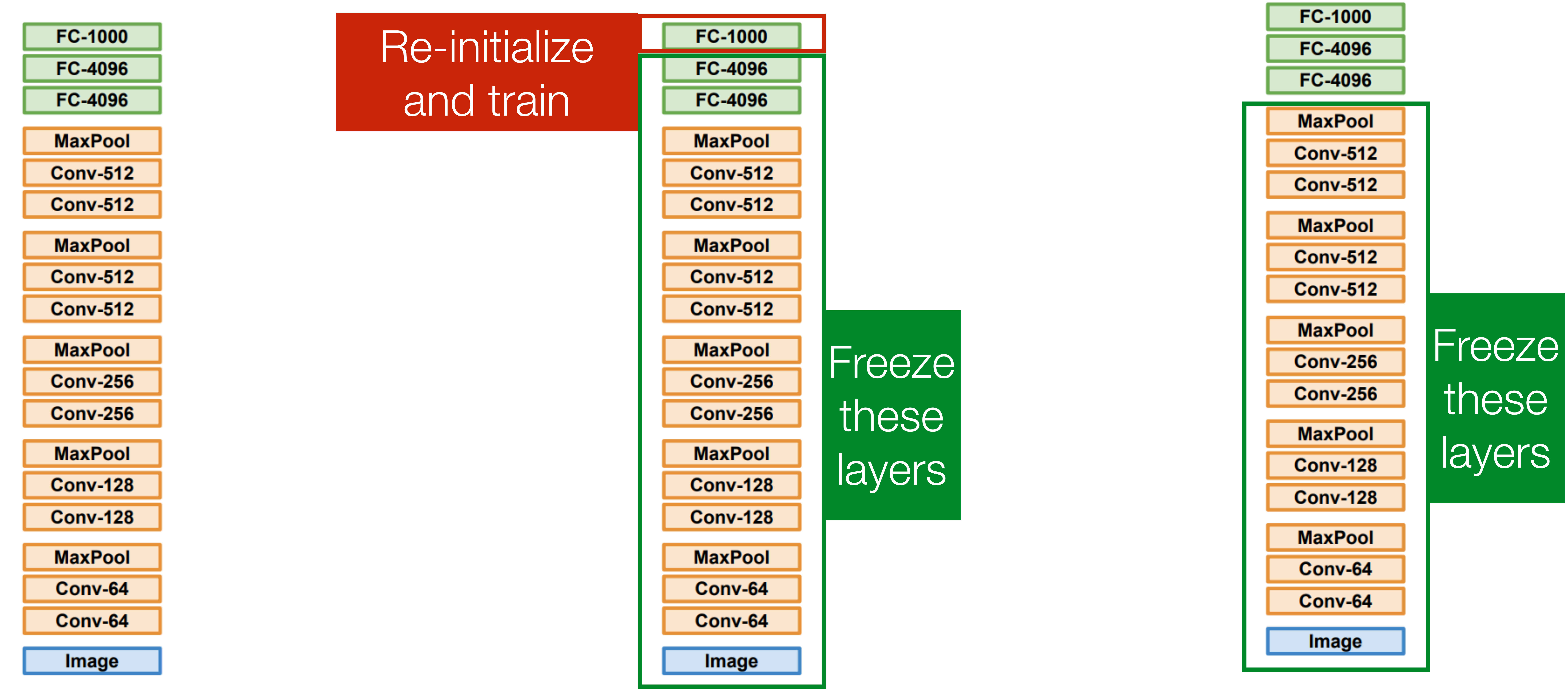


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

Train on **ImageNet** | **Small dataset** with C classes | **Larger dataset**

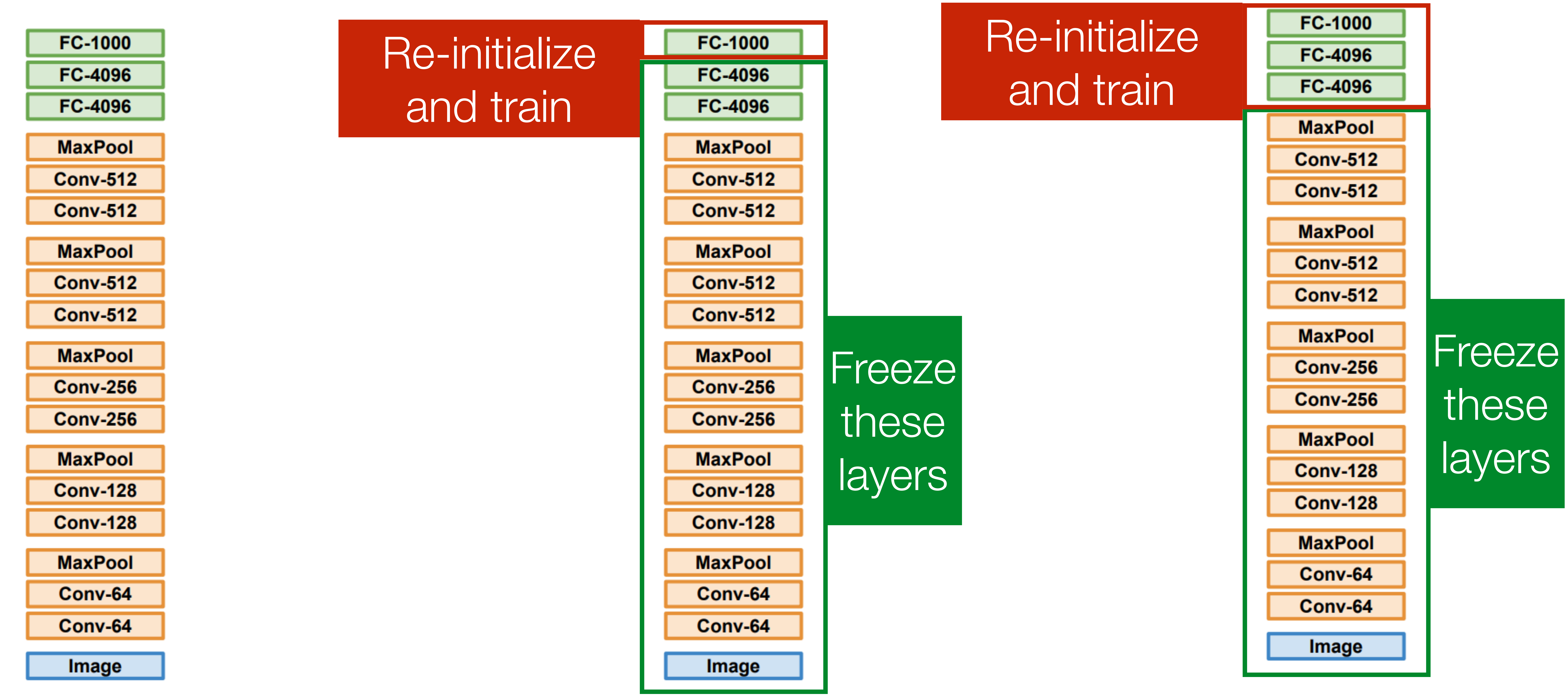


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

[Yosinski et al., NIPS 2014]
[Donahue et al., ICML 2014]
[Razavian et al., CVPR Workshop 2014]

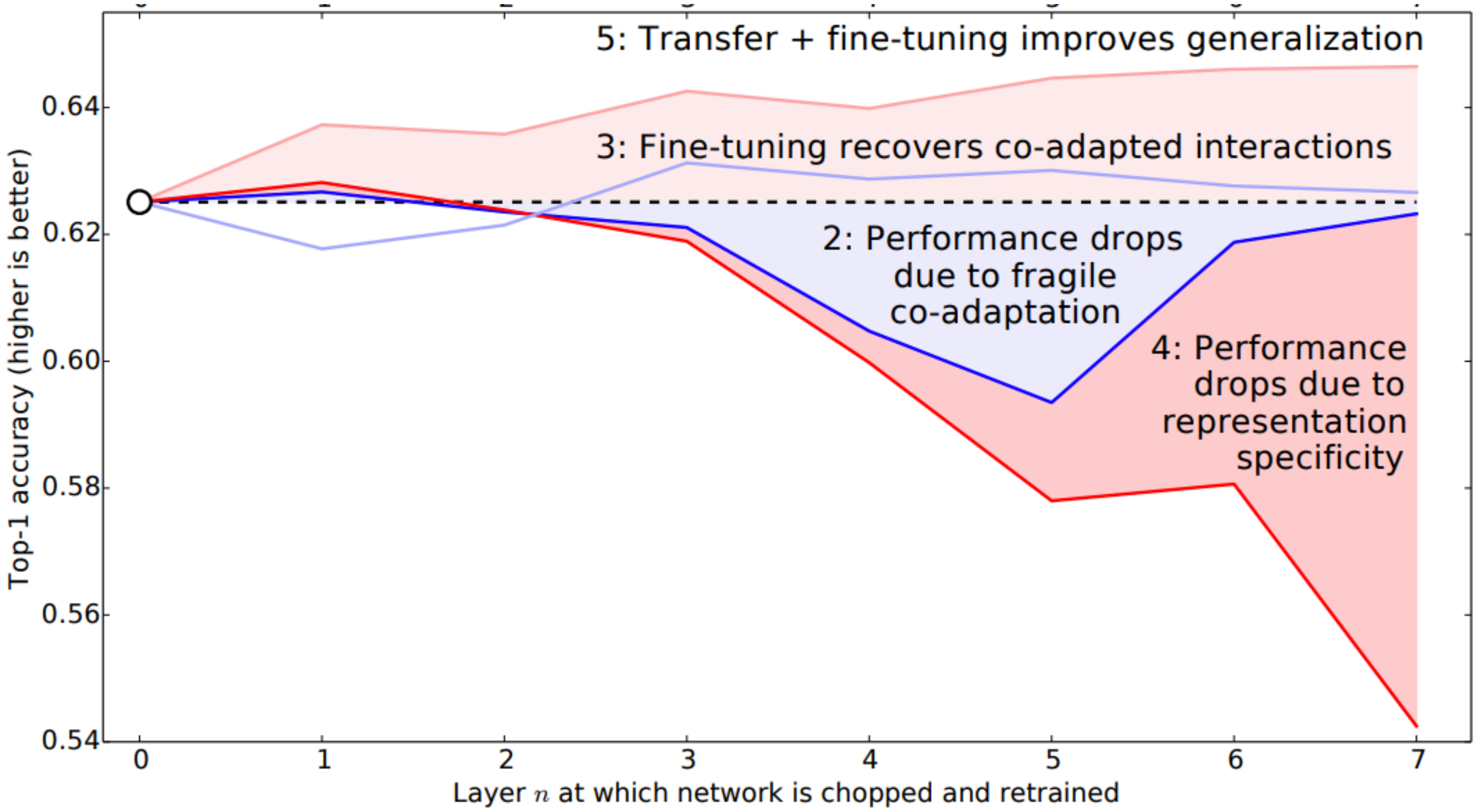
Train on **ImageNet** | **Small dataset** with C classes | **Larger dataset**



* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Transfer Learning with CNNs

Dataset A: 500 classes



Layers fine-tuned

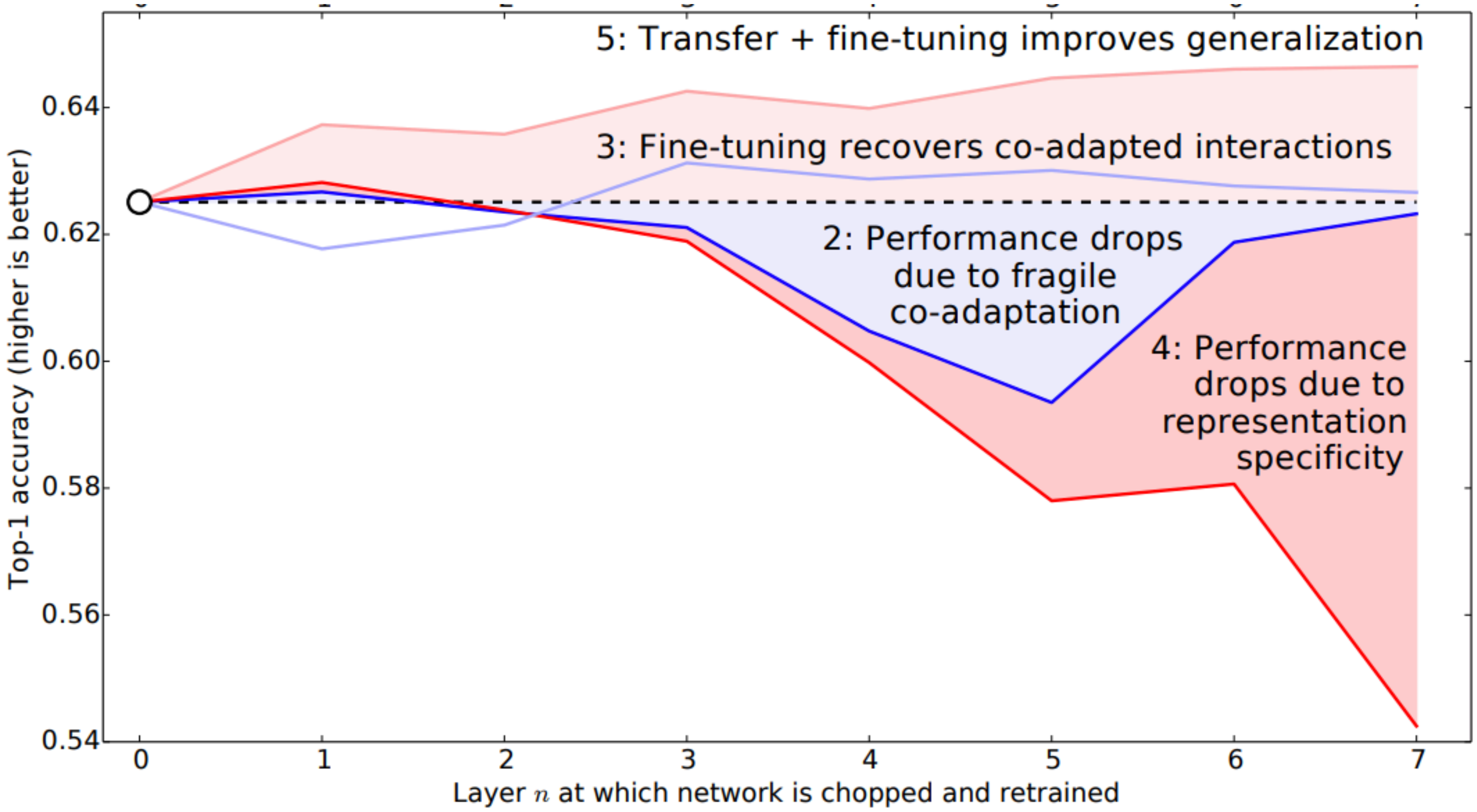
Layers fixed

[Yosinski et al., NIPS 2014]

Transfer Learning with CNNs

Dataset A: 500 classes

Dataset B: (different) 500 classes



Layers fine-tuned

Layers fixed

[Yosinski et al., NIPS 2014]

Layers fine-tuned

Layers fixed

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Alternative: Multiple snapshots of the single model during training!

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

~ 2% improved performance in practice

Alternative: Multiple snapshots of the single model during training!

Improvement: Instead of using the actual parameter vector, keep a moving average of the parameter vector and use that at test time (Polyak averaging)

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

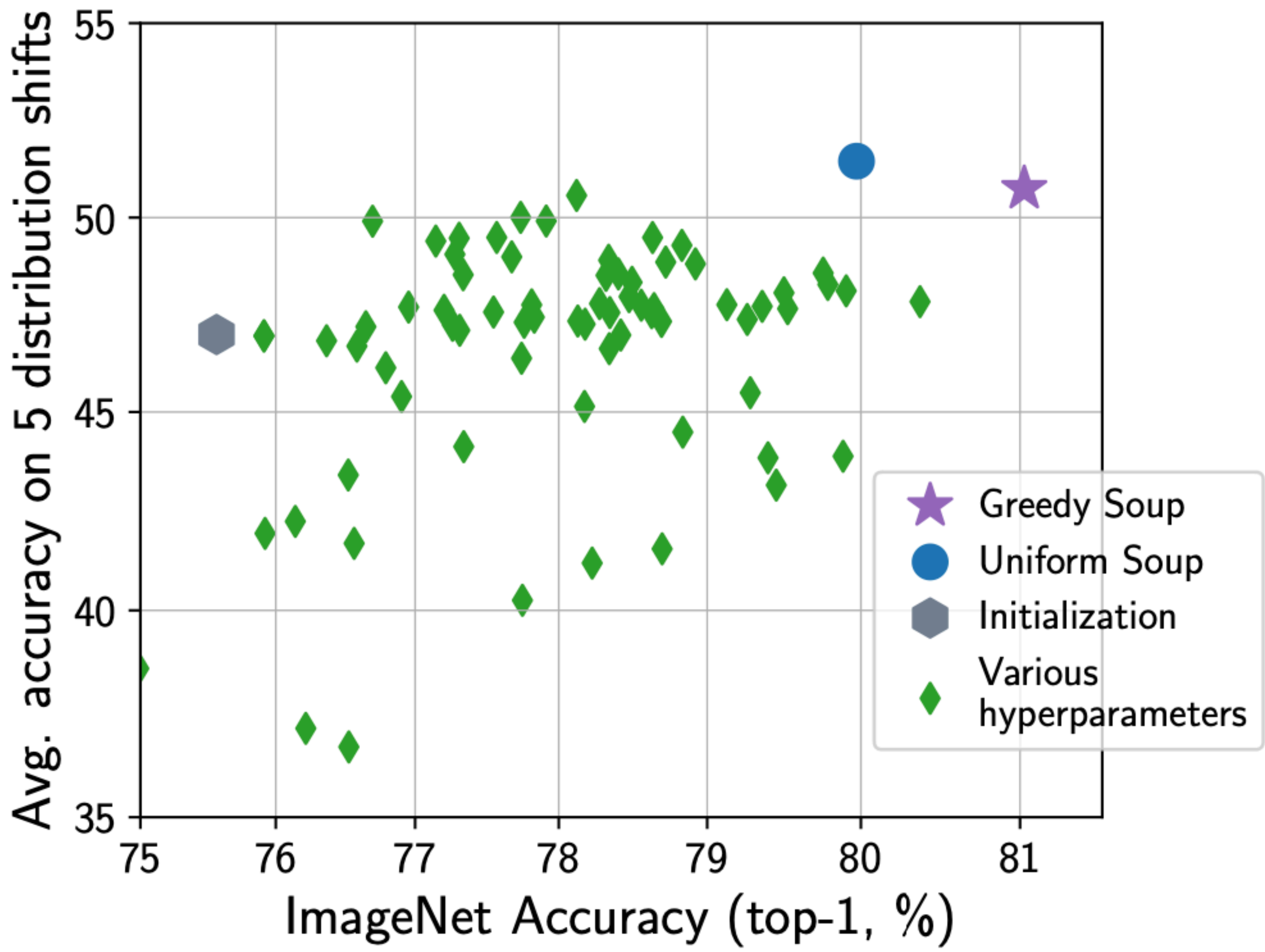
Model **Ensemble** vs. **Soup**

	Method	Cost
Best on val. set	$f(x, \arg\max_i \mathsf{ValAcc}(\theta_i))$	$\mathcal{O}(1)$
Ensemble	$\frac{1}{k}\sum_{i=1}^{k} f(x, \theta_i)$	$\mathcal{O}(k)$
Uniform soup	$f\left(x, \frac{1}{k}\sum_{i=1}^{k} \theta_i\right)$	$\mathcal{O}(1)$
Greedy soup	Recipe 1	$\mathcal{O}(1)$
Learned soup	Appendix I	$\mathcal{O}(1)$

Recipe 1 GreedySoup

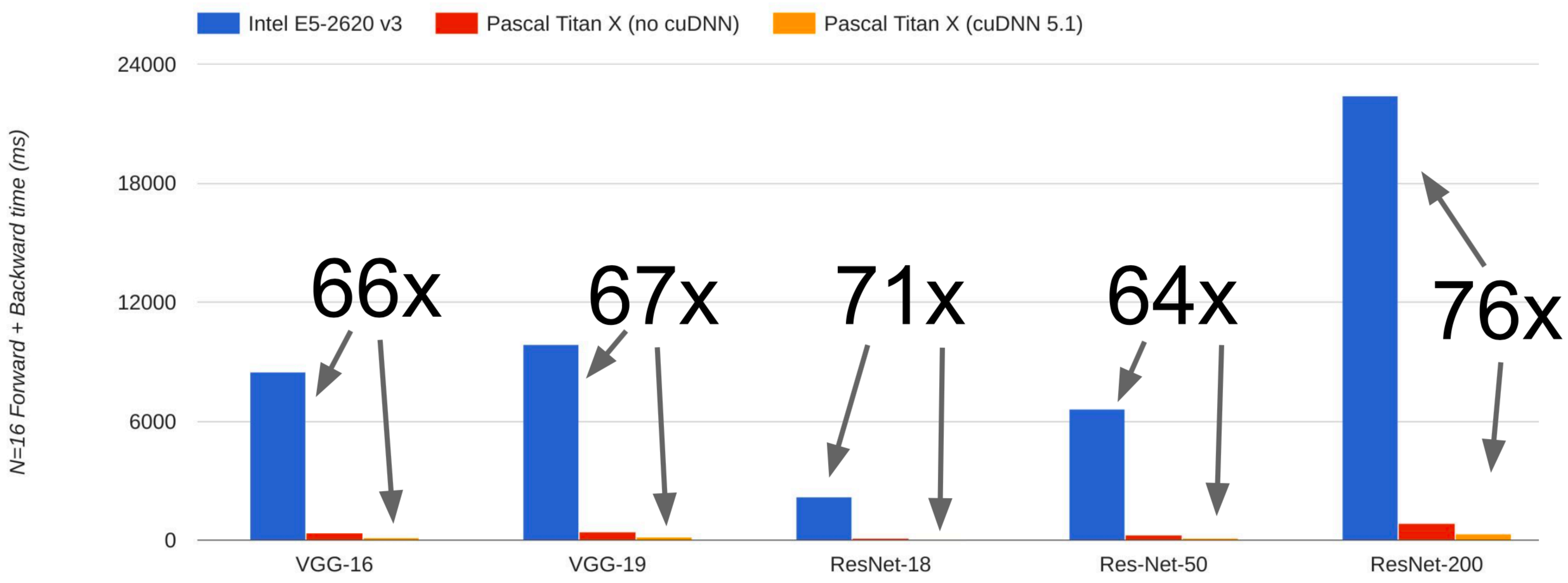
Input: Potential soup ingredients $\{\theta_1, ..., \theta_k\}$ (sorted in decreasing order of $\mathsf{ValAcc}(\theta_i)$).

```
ingredients ← {}
for i = 1 to k do
  if ValAcc(average(ingredients ∪ {θ_i})) ≥
          ValAcc(average(ingredients)) then
    ingredients ← ingredients ∪ {θ_i}
return average(ingredients)
```



[Wortsman et al., ArXiv 2022]

CPU vs. **GPU** (Why do we need Azure?)



Data from https://github.com/jcjohnson/cnn-benchmarks

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Frameworks: Super quick overview

1. Easily **build computational graphs**

2. Easily **compute gradients** in computational graphs

3. **Run it all efficiently** on a GPU (weap cuDNN, cuBLAS, etc.)

Frameworks: Super quick overview

Core DNN Frameworks		
Caffe (UC Berkeley)	Caffe 2 (Facebook)	Puddle (Baidu)
Torch (NYU/Facebook)	**PyTorch** (Facebook)	CNTK (Microsoft)
Theano (U Montreal)	**TensorFlow** (Google)	MXNet (Amazon)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Frameworks: Super quick overview

Core DNN Frameworks		
Caffe (UC Berkeley)	Caffe 2 (Facebook)	Puddle (Baidu)
Torch (NYU/Facebook)	**PyTorch** (Facebook)	CNTK (Microsoft)
Theano (U Montreal)	**TensorFlow** (Google)	MXNet (Amazon)

Wrapper Libraries
Keras
TFLearn
TensorLayer
tf.layers
TF-Slim
tf.contrib.learn
Pretty Tensor

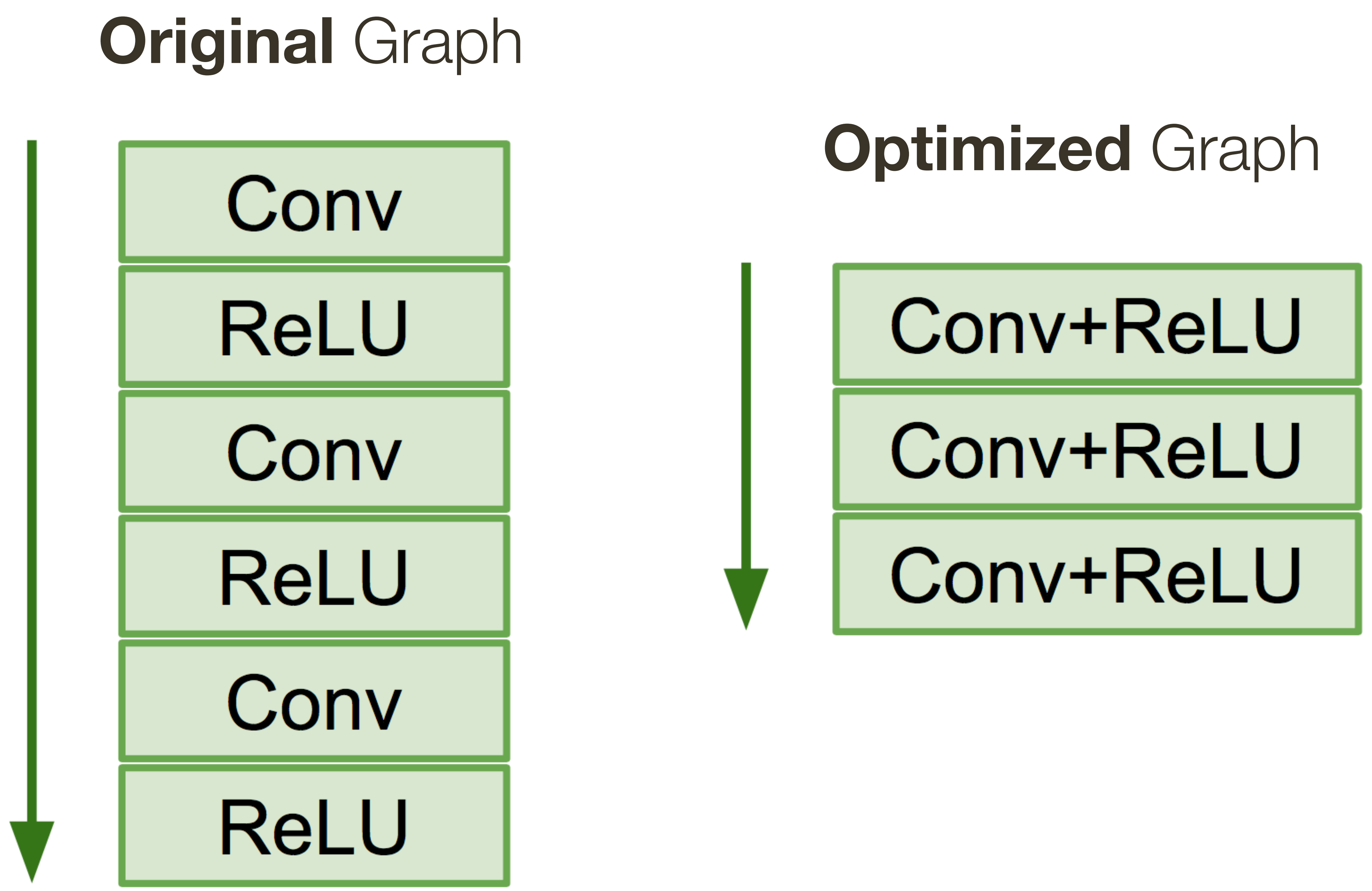
Frameworks: PyTorch vs. TensorFlow (v1)

Dynamic vs. Static computational graphs

Frameworks: PyTorch vs. TensorFlow (v1)

Dynamic vs. **Static** computational graphs

With static graphs, framework can **optimize** the graph for you before it runs!

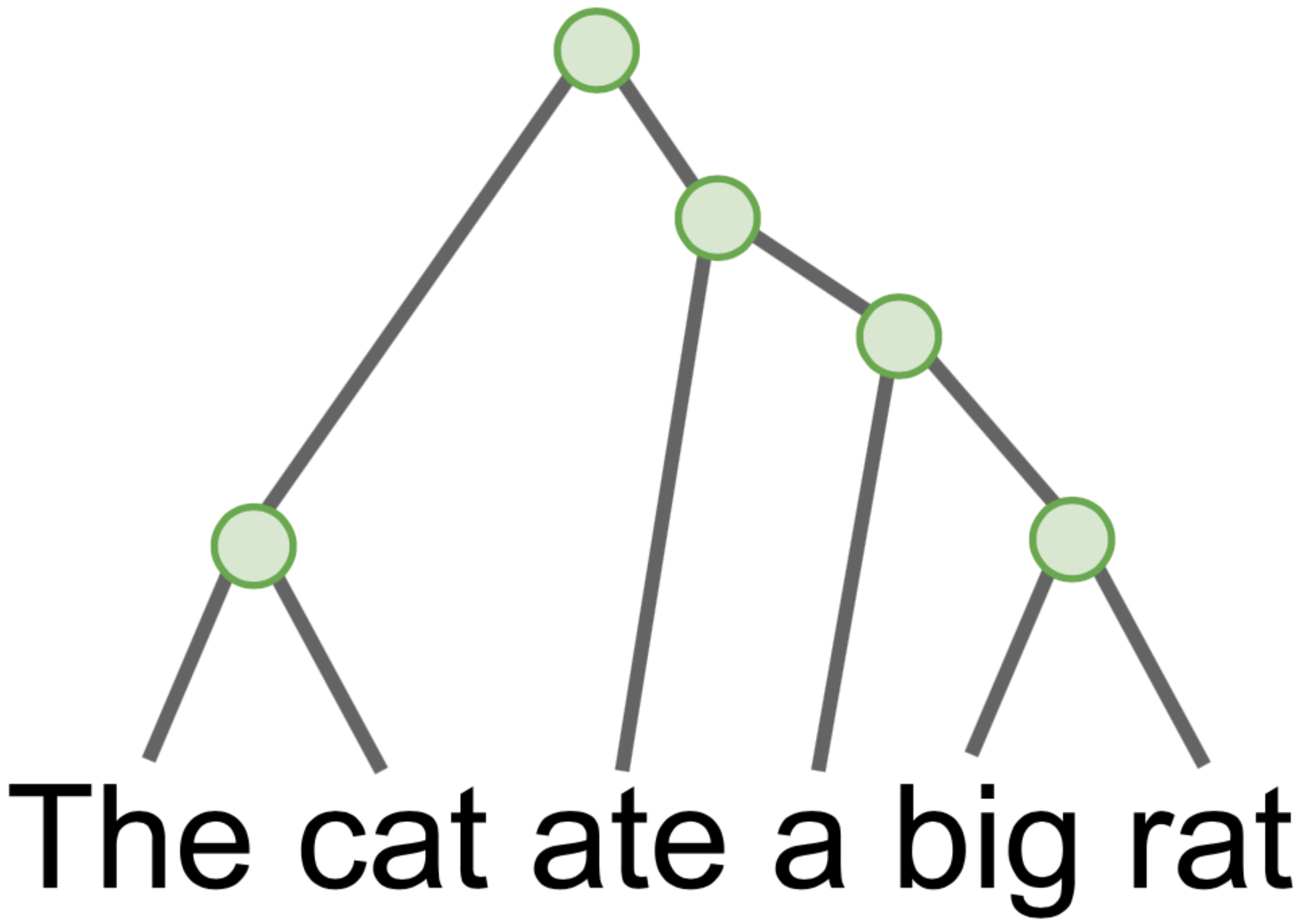


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Frameworks: PyTorch vs. TensorFlow (v1)

Dynamic vs. Static computational graphs

Graph building and execution is intertwined. Graph can be different for every sample.



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

PyTorch: Three levels of abstraction

Tensor: Imperative ndarray, but runs on GPU

Variable: Node in a computational graph; stores data and gradients

Module: A neural network layer; may store state or learnable weights

Computer **Vision Problems** (no language for now)

Computer **Vision Problems** (no language for now)

Categorization

Computer **Vision Problems** (no language for now)

Categorization

Multi-**class:** Horse
Church
Toothbrush
Person

IMAGENET

Computer **Vision Problems** (no language for now)

Categorization

Multi-**class**: Horse
Church
Toothbrush
Person

IMAGENET

Multi-**label**: **Horse**
Church
Toothbrush
Person

Computer **Vision Problems** (no language for now)

Categorization

Multi-**class:**

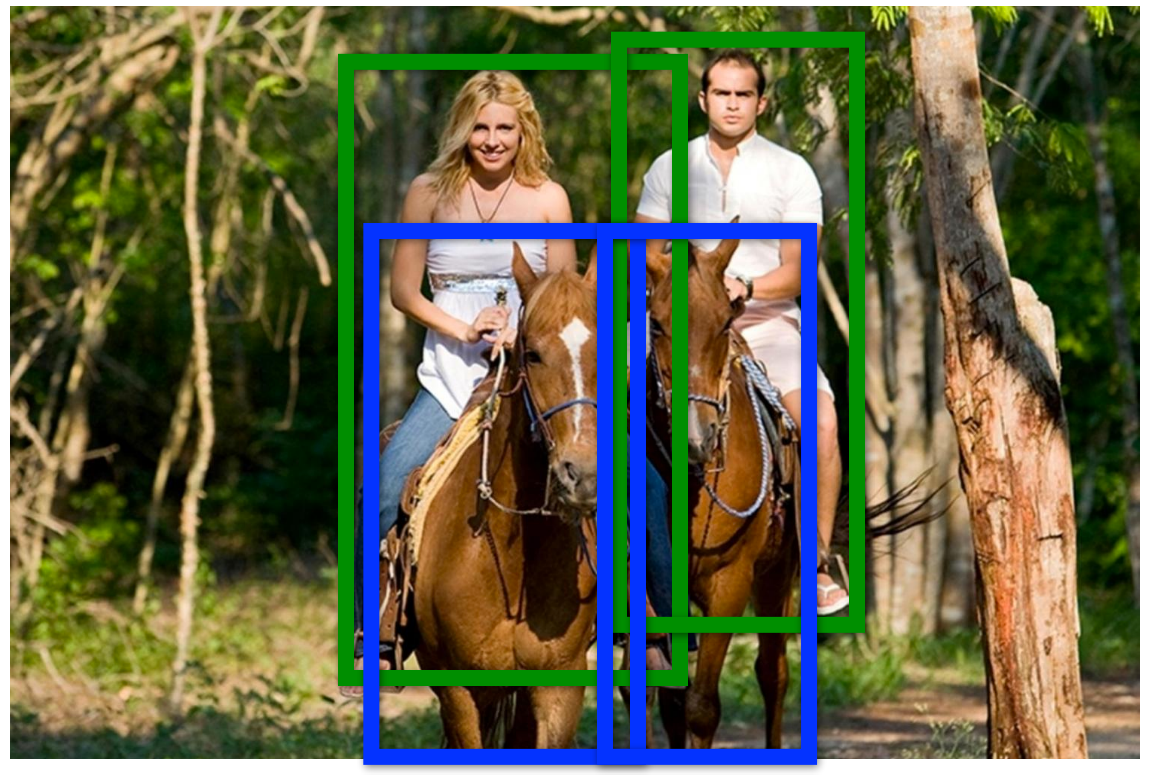
Horse
Church
Toothbrush
Person

IMAGENET

Multi-**label**:

Horse
Church
Toothbrush
Person

Detection



Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)

COCO
Common Objects in Context

Computer **Vision Problems** (no language for now)

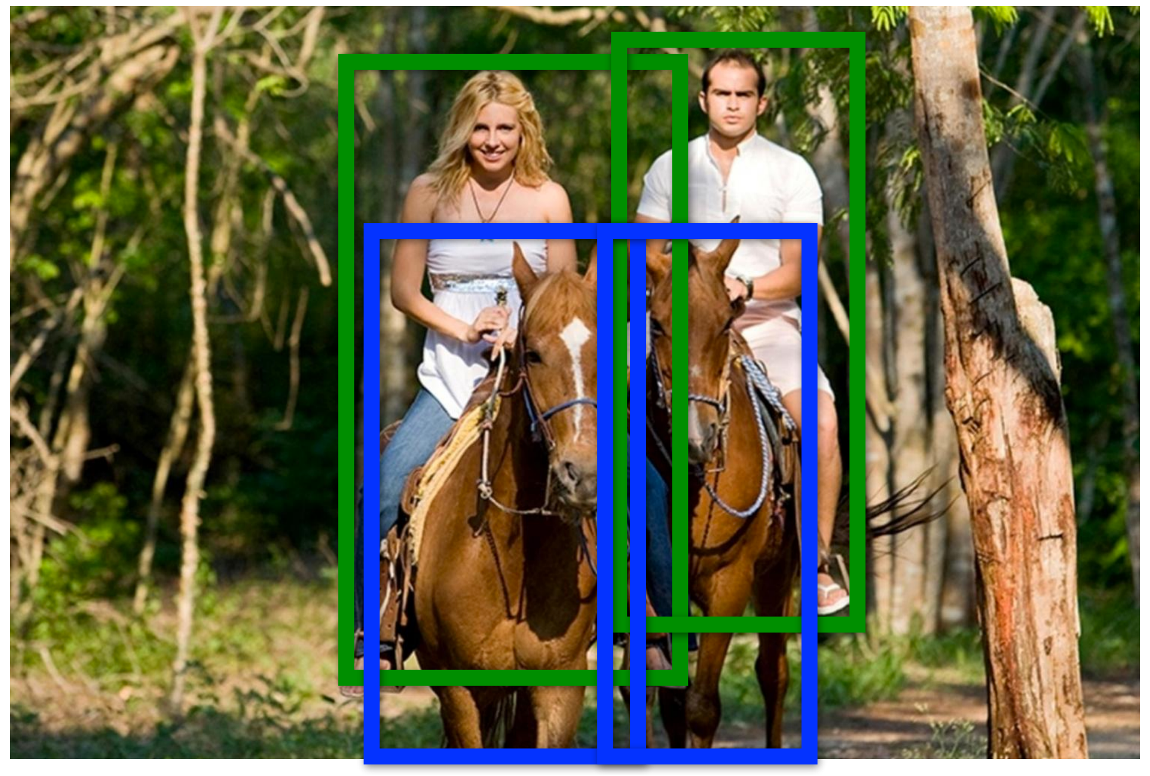
Categorization

Multi-**class**: Horse
Church
Toothbrush
Person

IMAGENET

Multi-**label**: **Horse**
Church
Toothbrush
Person

Detection



Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)

COCO
Common Objects in Context

Segmentation

Horse
Person

COCO
Common Objects in Context

Computer **Vision Problems** (no language for now)

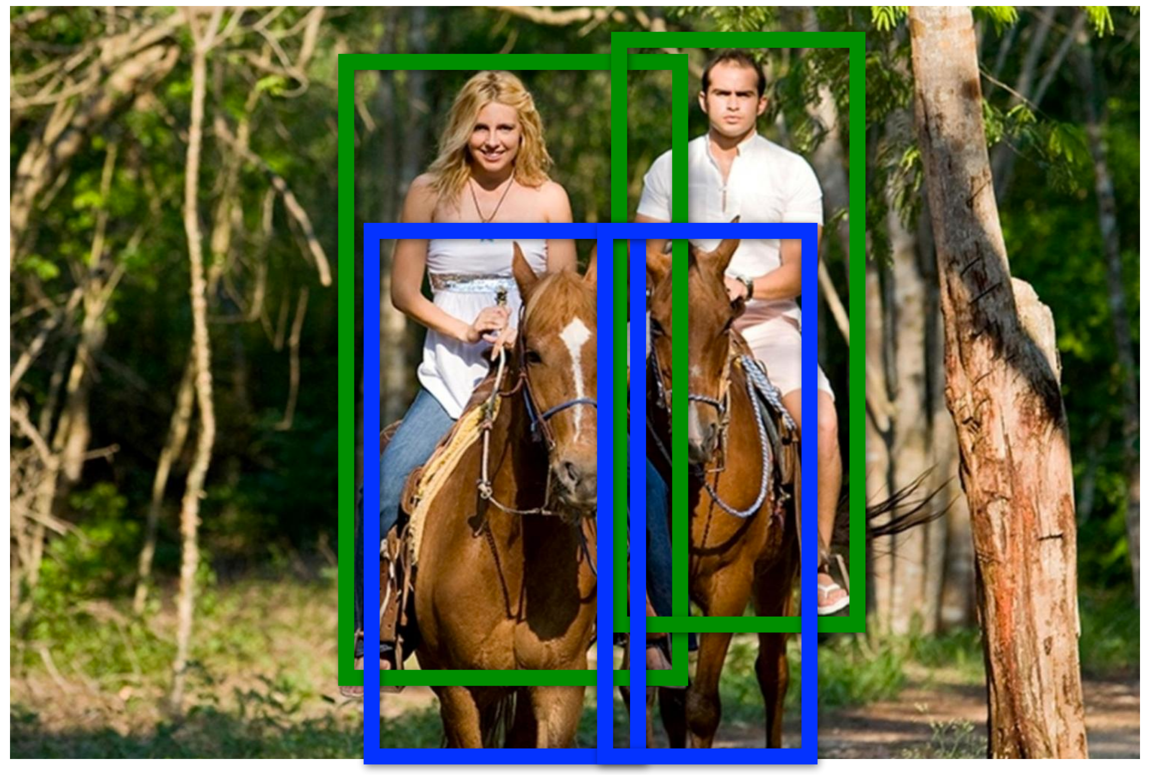
Categorization

Multi-**class**: Horse
Church
Toothbrush
Person

IMAGENET

Multi-**label**: **Horse**
Church
Toothbrush
Person

Detection



Horse (x, y, w, h)
Horse (x, y, w, h)
Person (x, y, w, h)
Person (x, y, w, h)

COCO Common Objects in Context

Segmentation

Horse
Person

COCO Common Objects in Context

Instance Segmentation

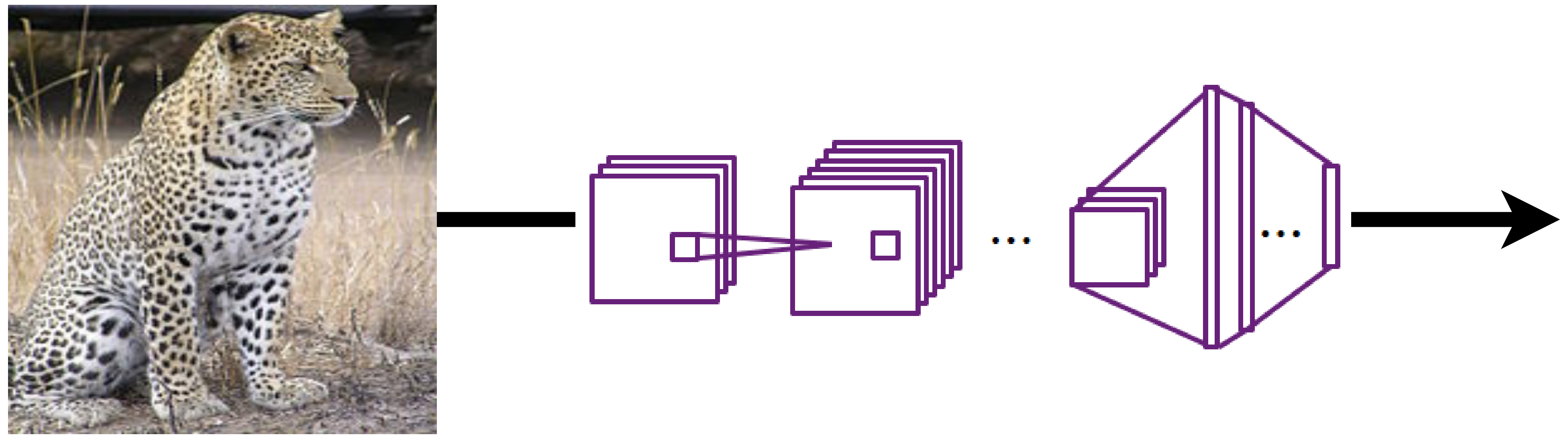
Horse1
Horse2
Person1
Person2

Object **Classification**

Category	Prediction
Dog	No
Cat	No
Couch	No
Flowers	No
Leopard	**Yes**
…	…

Problem: For each image predict which category it belongs to out of a fixed set

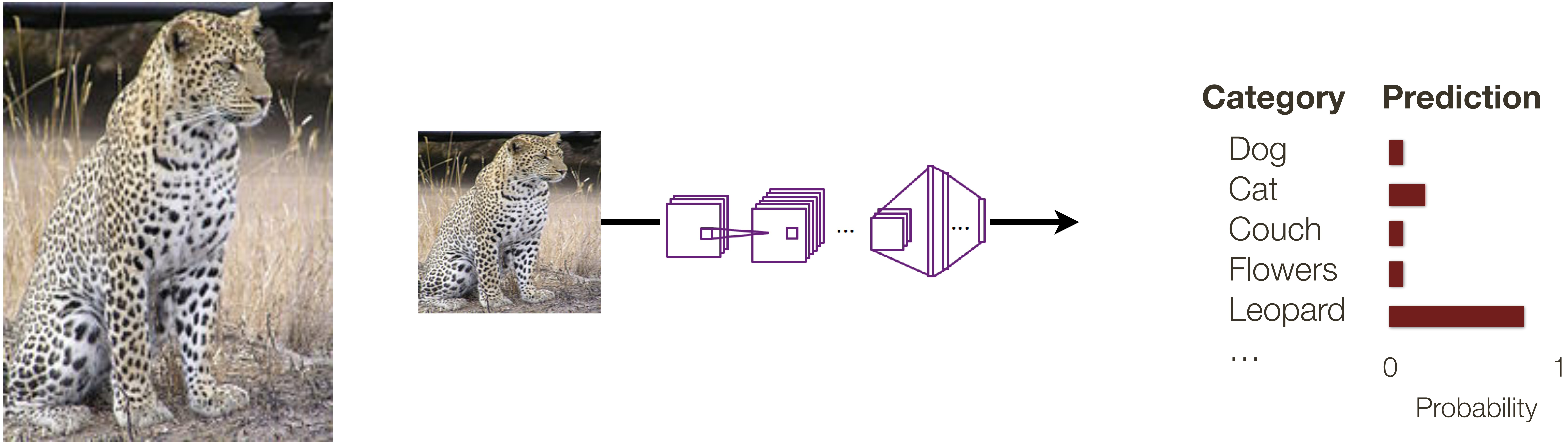
Object **Classification**



Category	**Prediction**
Dog	No
Cat	No
Couch	No
Flowers	No
Leopard	**Yes**
…	…

Problem: For each image predict which category it belongs to out of a fixed set

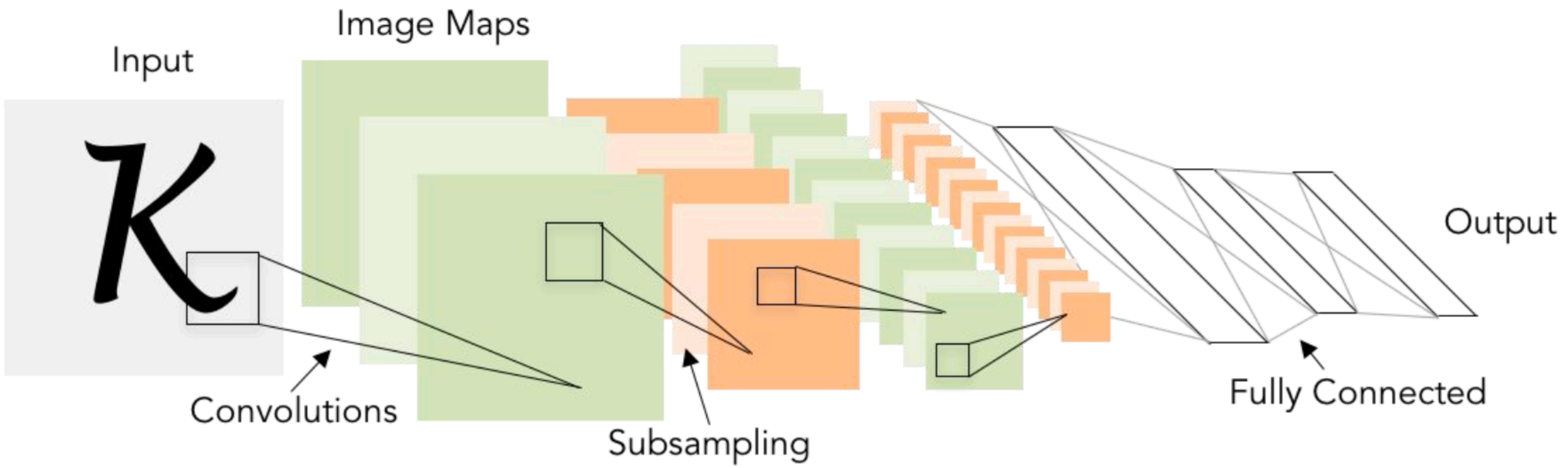
Object **Classification**



Problem: For each image predict which category it belongs to out of a fixed set

CNN Architectures: LeNet-5

[LeCun et al., 1998]



Architecture: CONV —> POOL —> CONV —> POOL —> FC —> FC

Conv filters: 5x5, Stride: 1

Pooling: 2x2, Stride: 2

ImageNet **Dataset**

Over **14 million** (high resolution) web **images**

Roughly labeled with **22K synset** categories

Labeled on Amazon Mechanical Turk (AMT)

Popular Synsets

Animal
fish
bird
mammal
invertebrate

Plant
tree
flower
vegetable

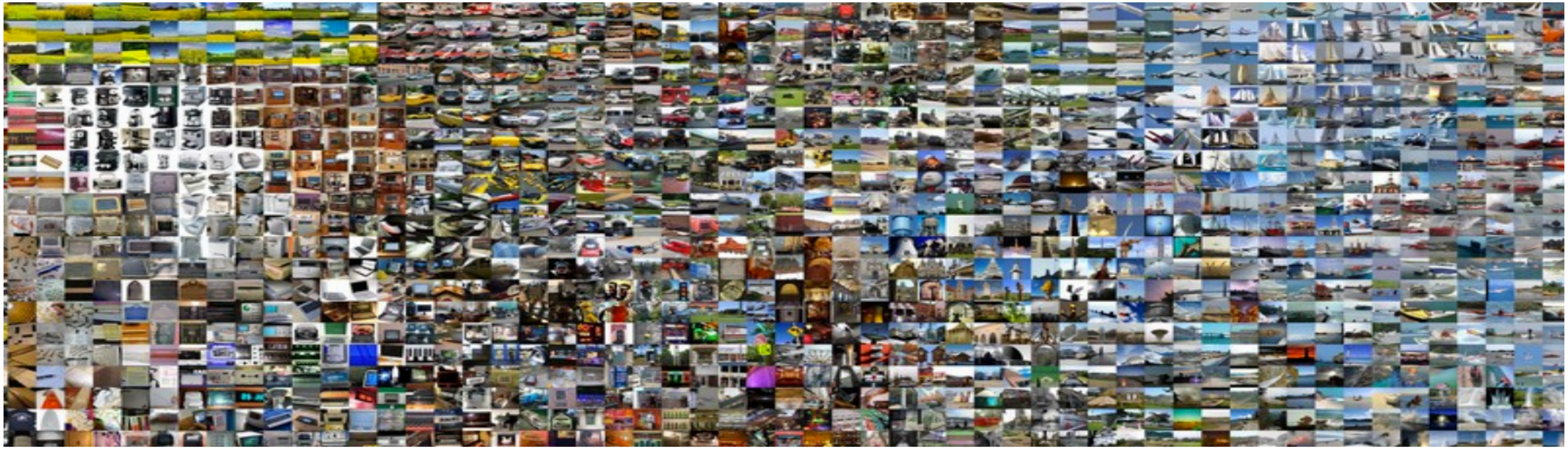
Activity
sport

Material
fabric

Instrumentation
utensil
appliance
tool
musical instrument

Scene
room
geological formation

Food
beverage

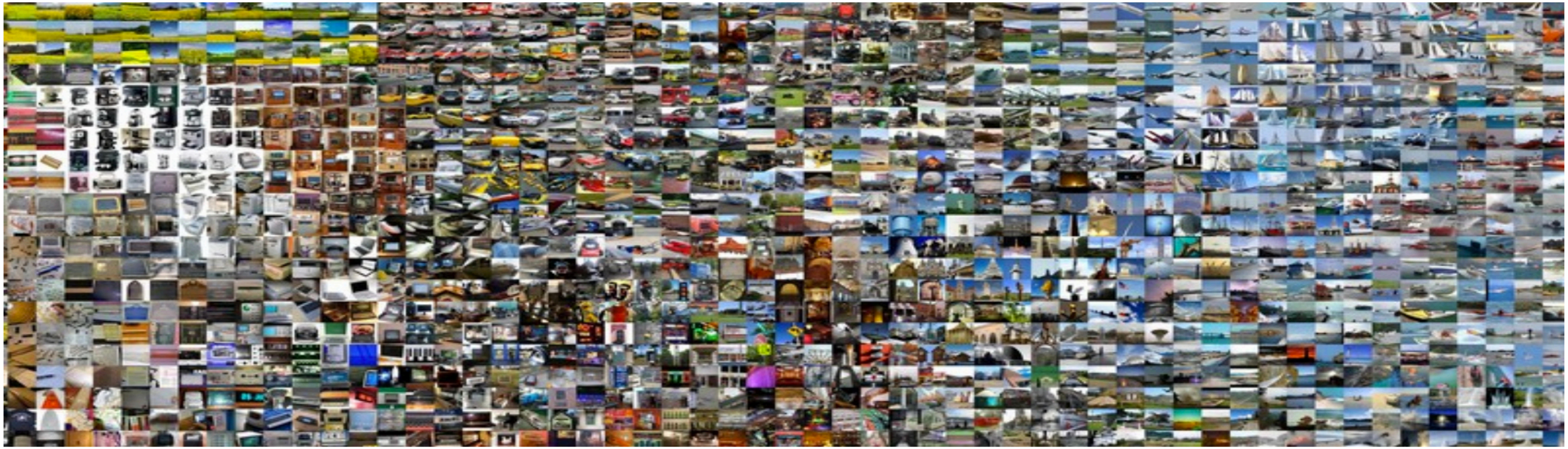


ImageNet **Competition** (ILSVRC)

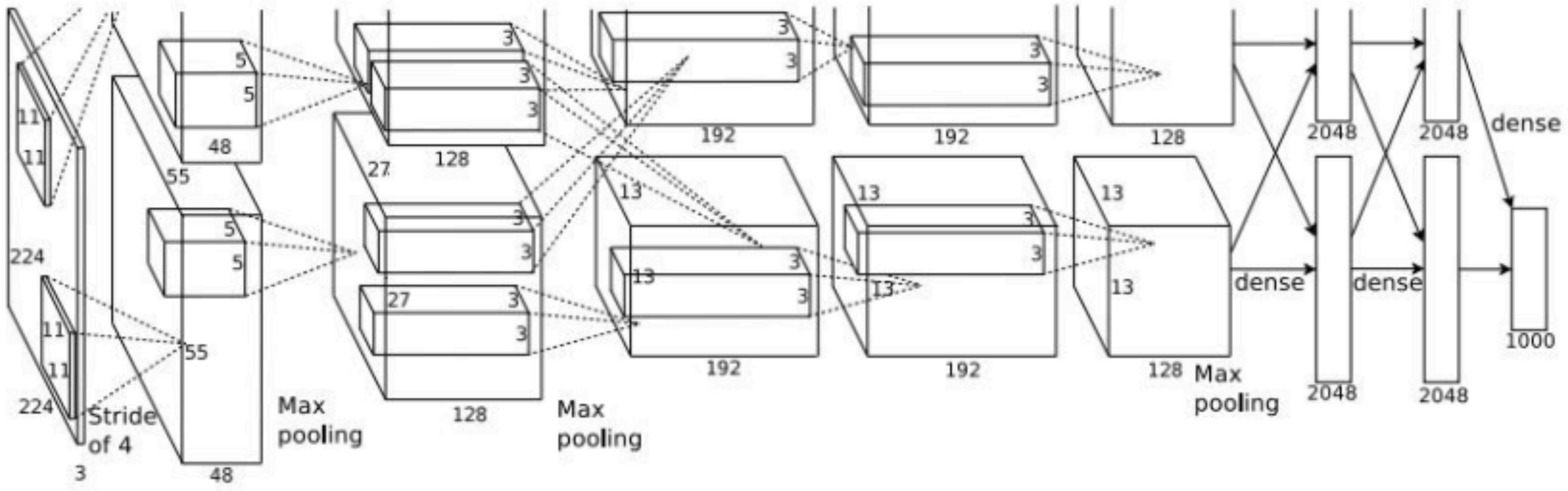
Annual competition of image classification at scale

Focuses on a subset of **1K synset** categories

Scoring: need to predict true label within top K (K=5)



AlexNet

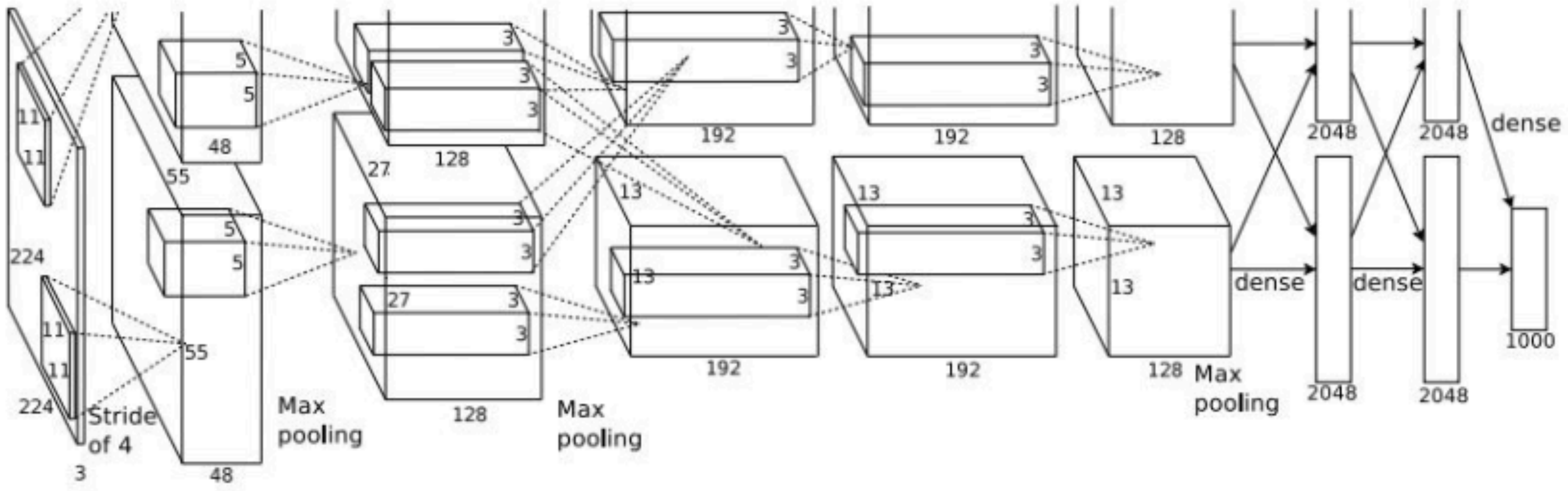


[Krizhevsky et al., 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Input: 227 x 227 x 3 images

AlexNet



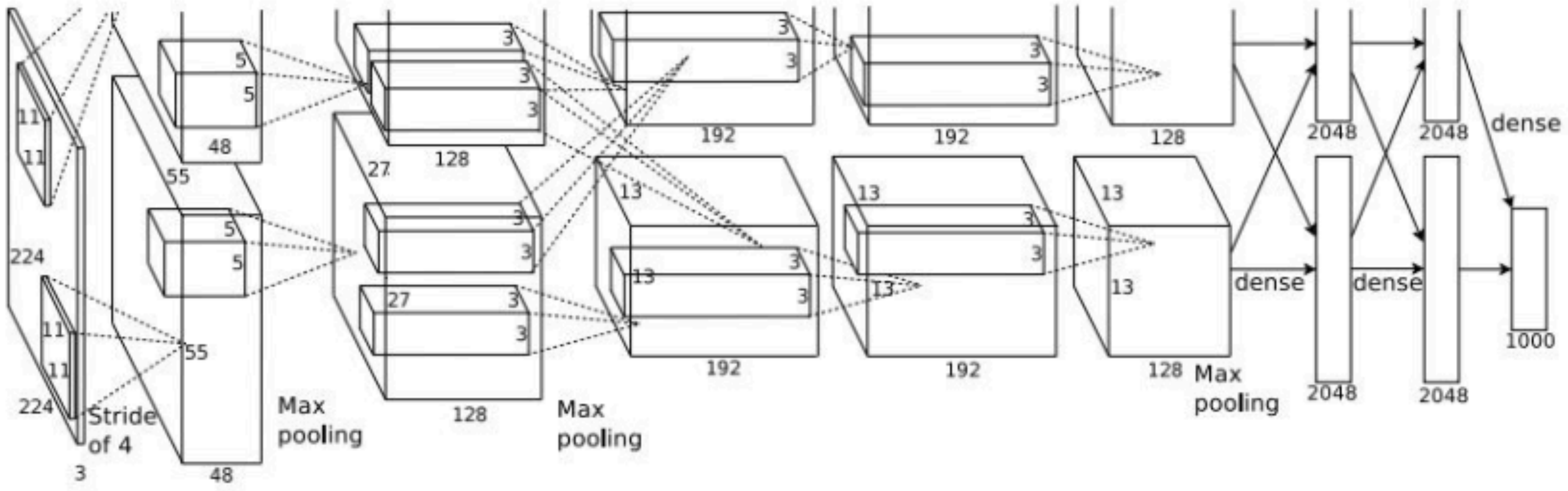
[Krizhevsky et al., 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

AlexNet



[Krizhevsky et al., 2012]

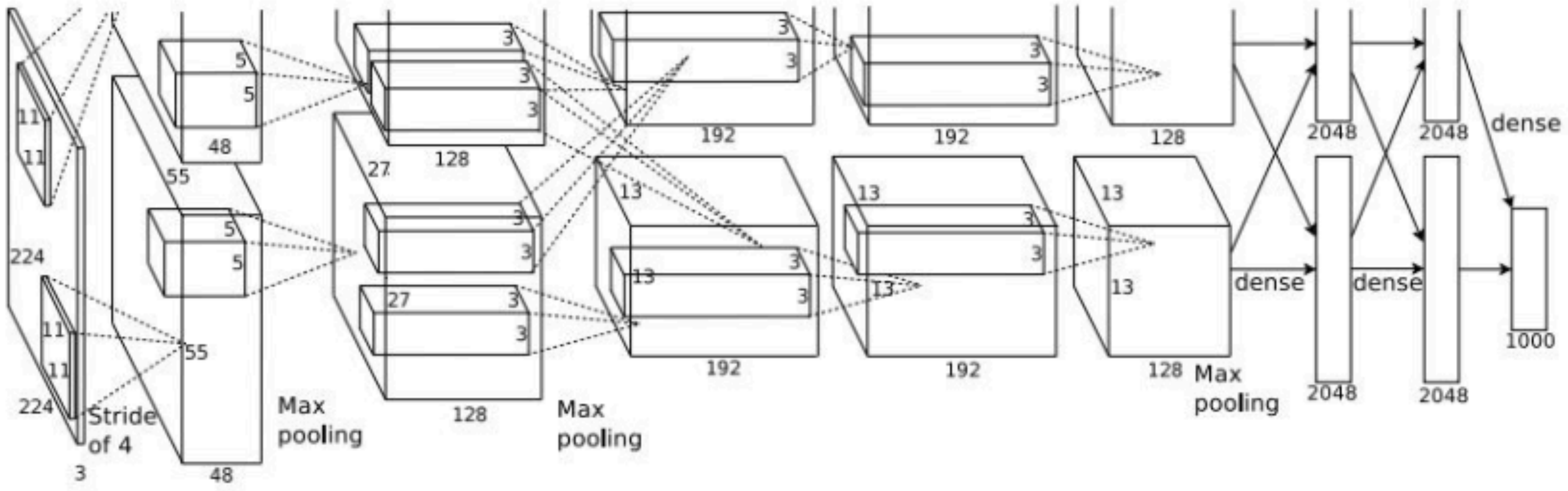
Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96

AlexNet



[Krizhevsky et al., 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

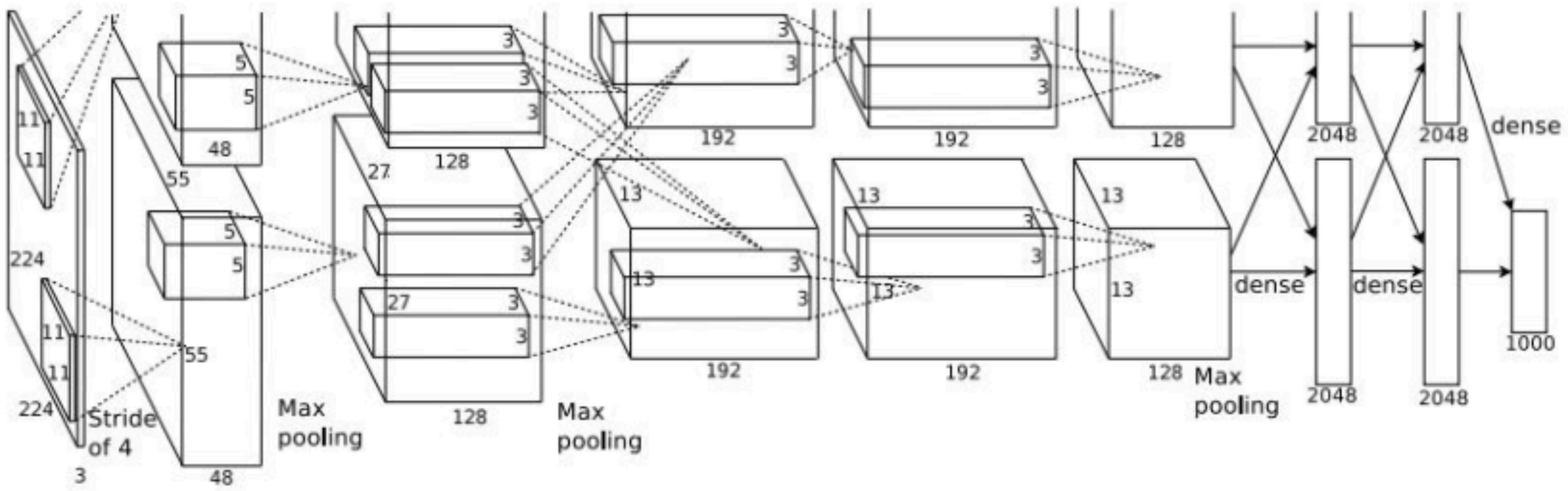
Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96

Parameters: 35K

AlexNet



[Krizhevsky et al., 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

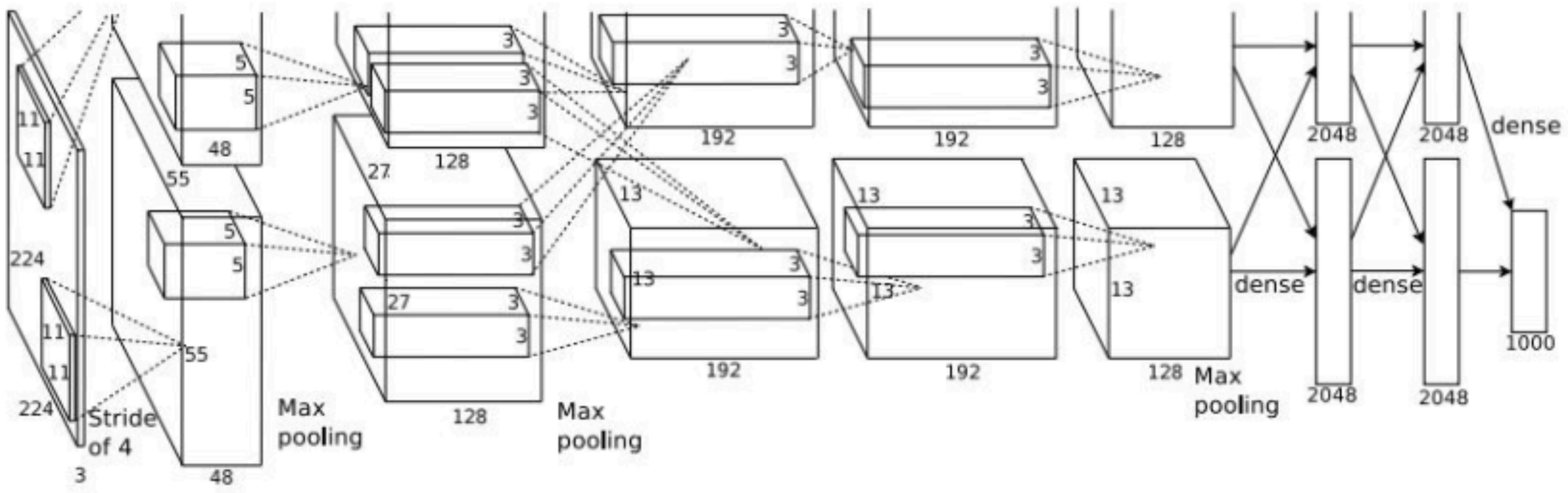
Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96 **Parameters:** 35K

MAX POOL1: 96 11 x 11 filters applied at stride 4

AlexNet



[Krizhevsky et al., 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Input: 227 x 227 x 3 images

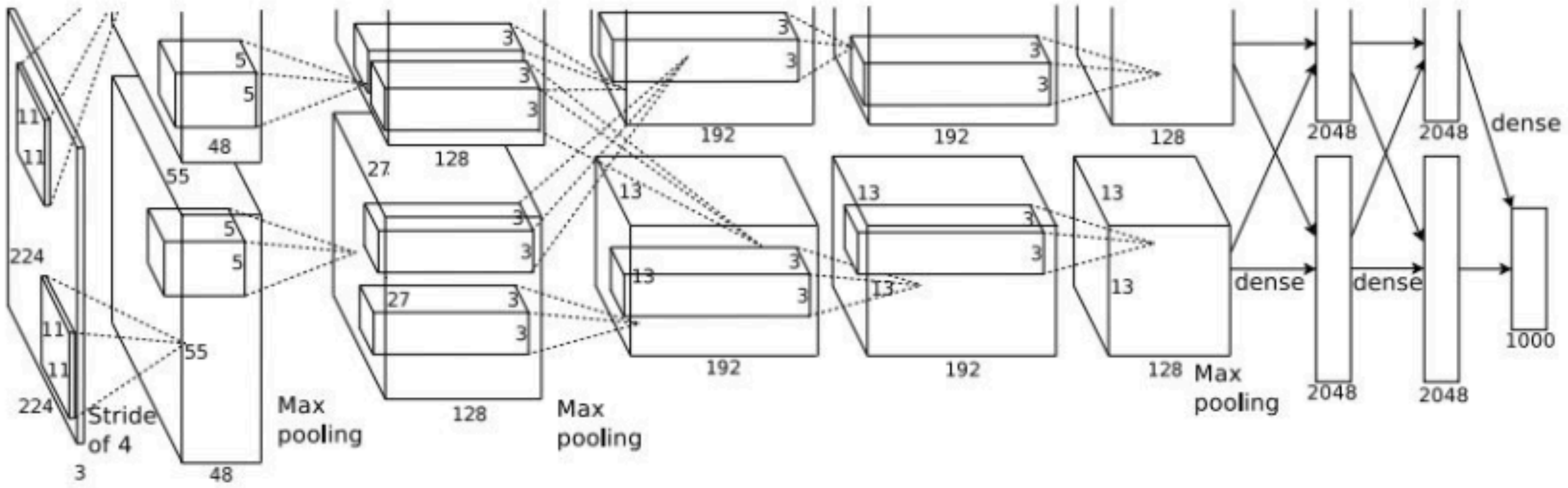
CONV1: 96 11 x 11 filters applied at stride 4

Output: 55 x 55 x 96 **Parameters:** 35K

MAX POOL1: 96 11 x 11 filters applied at stride 4

Output: 27 x 27 x 96

AlexNet



[Krizhevsky et al., 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Input: 227 x 227 x 3 images

CONV1: 96 11 x 11 filters applied at stride 4

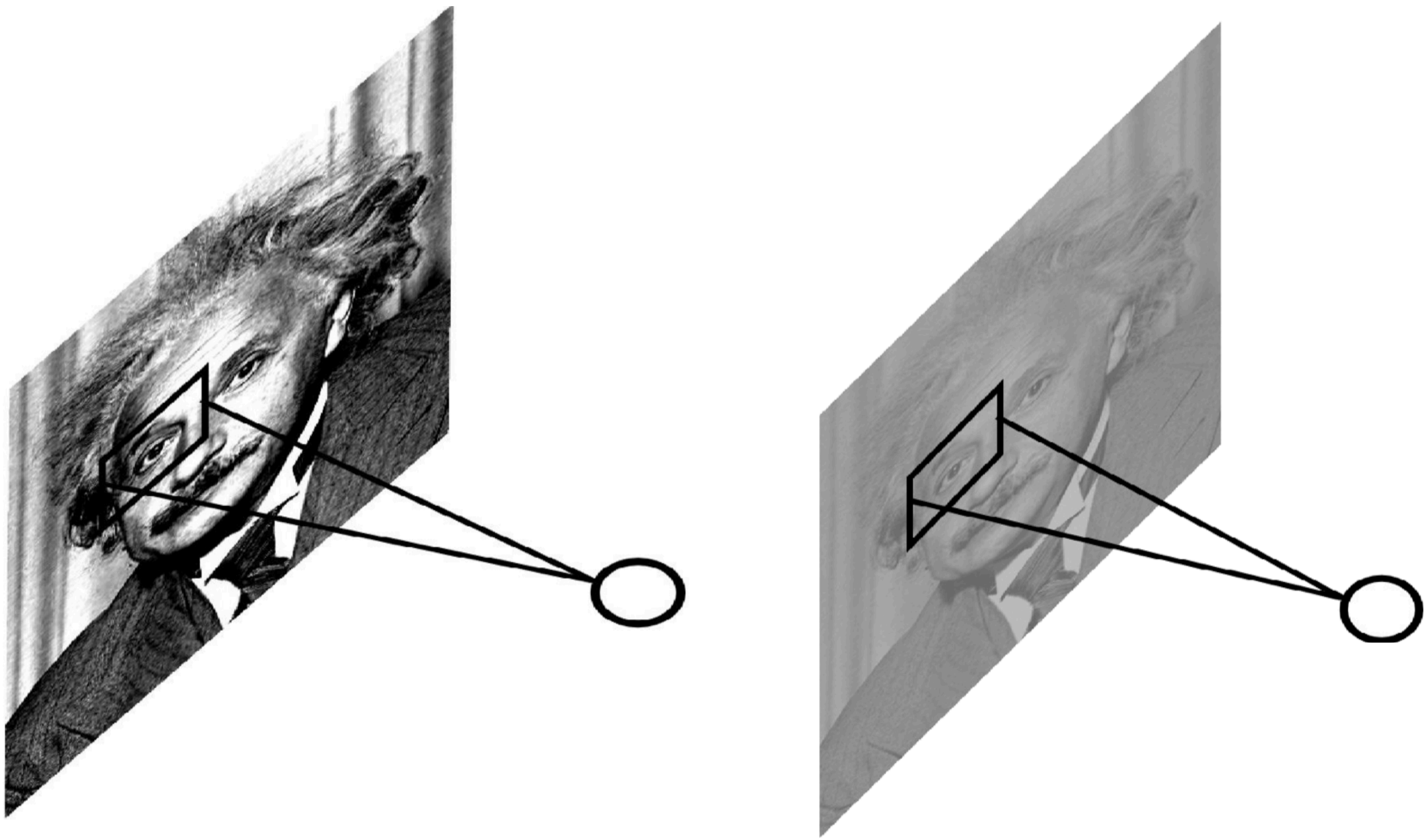
Output: 55 x 55 x 96 **Parameters:** 35K

MAX POOL1: 96 11 x 11 filters applied at stride 4

Output: 27 x 27 x 96 **Parameters:** 0

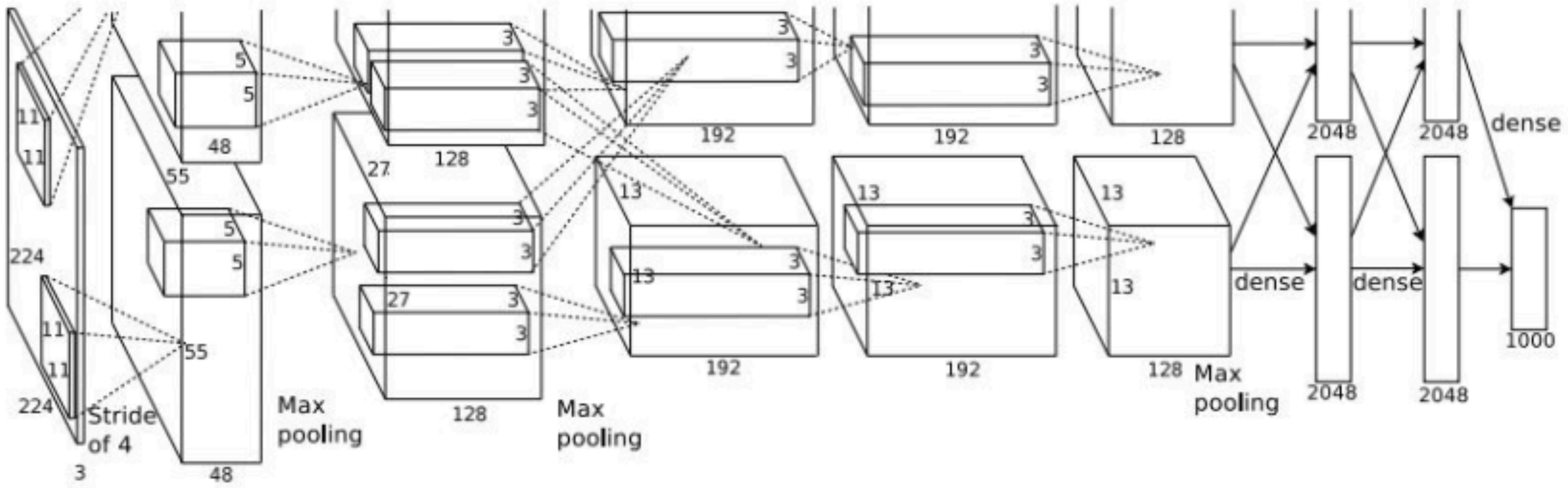
Local **Contrast Normalization** Layer

ensures response is the same in both case (details omitted, no longer popular)



* images from Marc'Aurelio Renzato

AlexNet

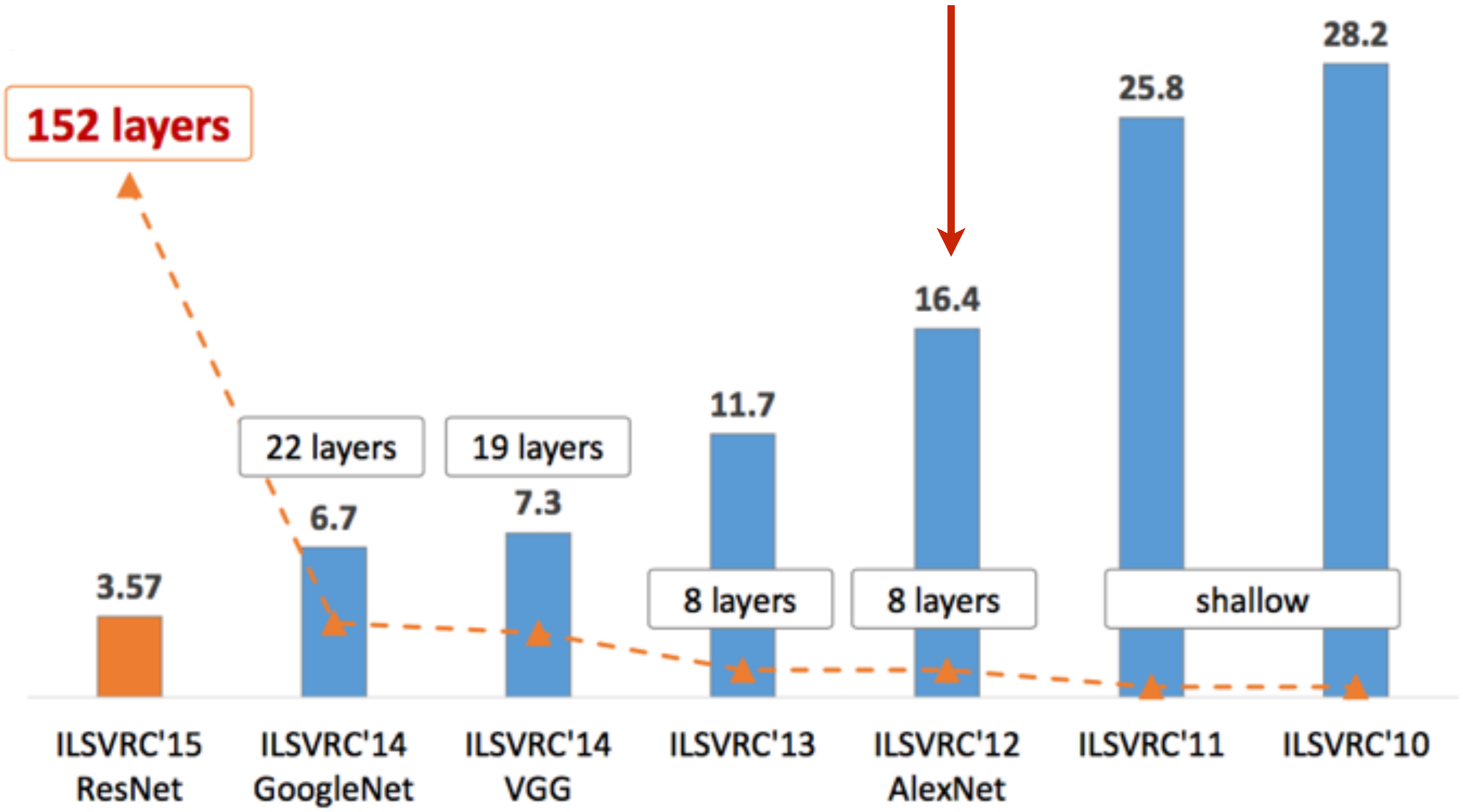


[Krizhevsky et al., 2012]

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 / Comments
— First use of ReLU
— Used contrast normalization layers
— Heavy data augmentation
— Dropout of 0.5
— Batch size of 128
— SGD Momentum (0.9)
— Learning rate (1e-2) reduced by 10 manually when validation accuracy plateaus
— L2 weight decay
— 7 CNN ensamble: 18.2% -> 15.4%

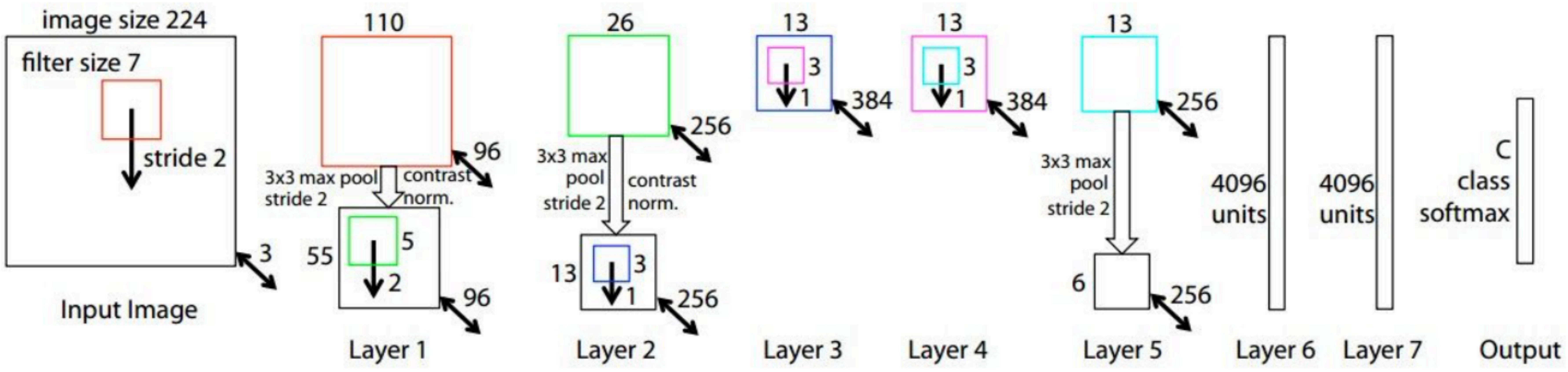
ILSVRC winner 2012



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

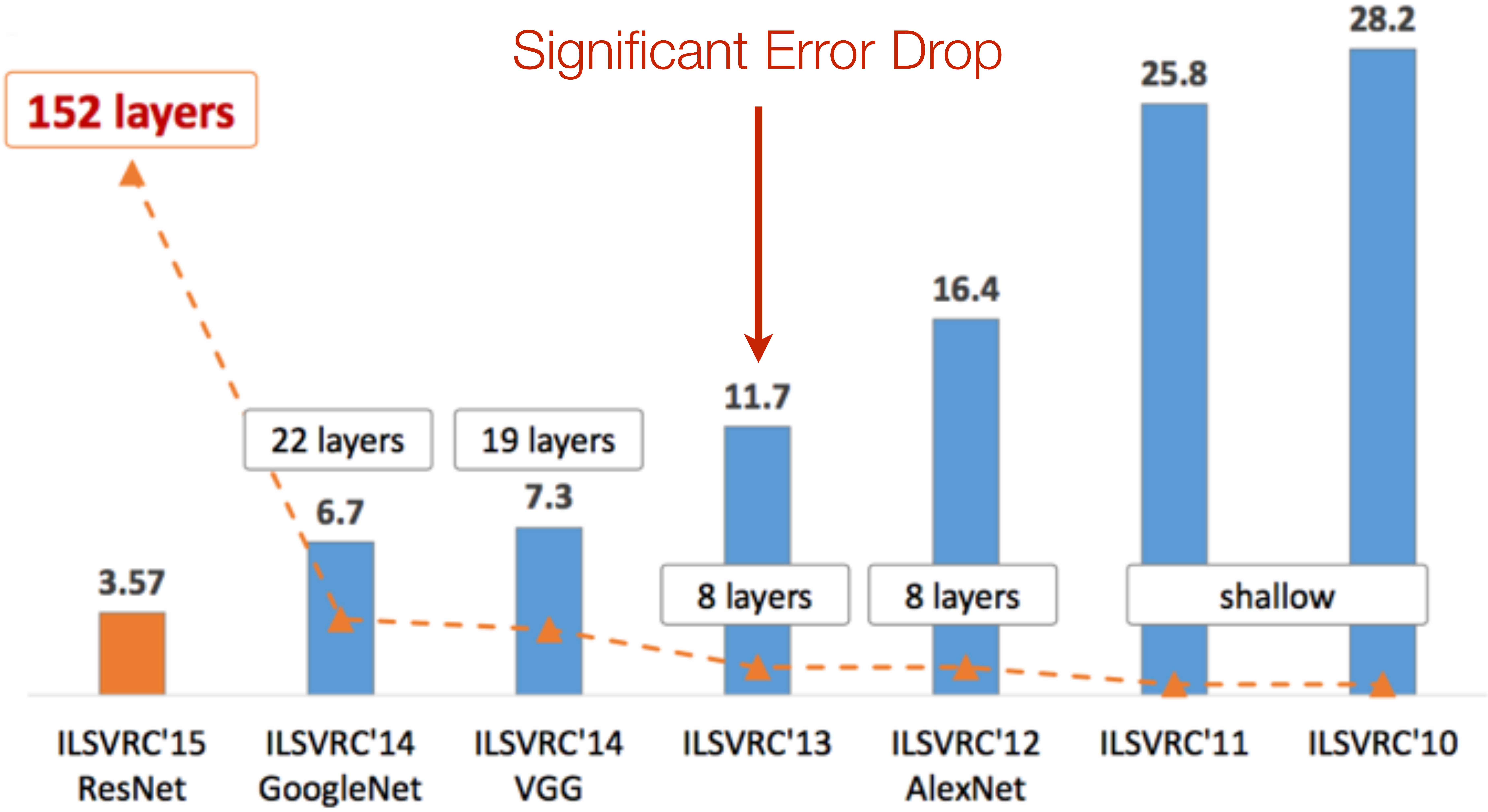
ZF Net

[Zeiler and Fergus, 2013]



AlexNet with small modifications:
— CONV1 (11 x 11 stride 4) to (7 x 7 stride 2)
— CONV3 # of filters 384 -> 512
— CONV4 # of filters 384 -> 1024
— CONV5 # of filters 256 -> 512

ILSVRC winner 2012



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

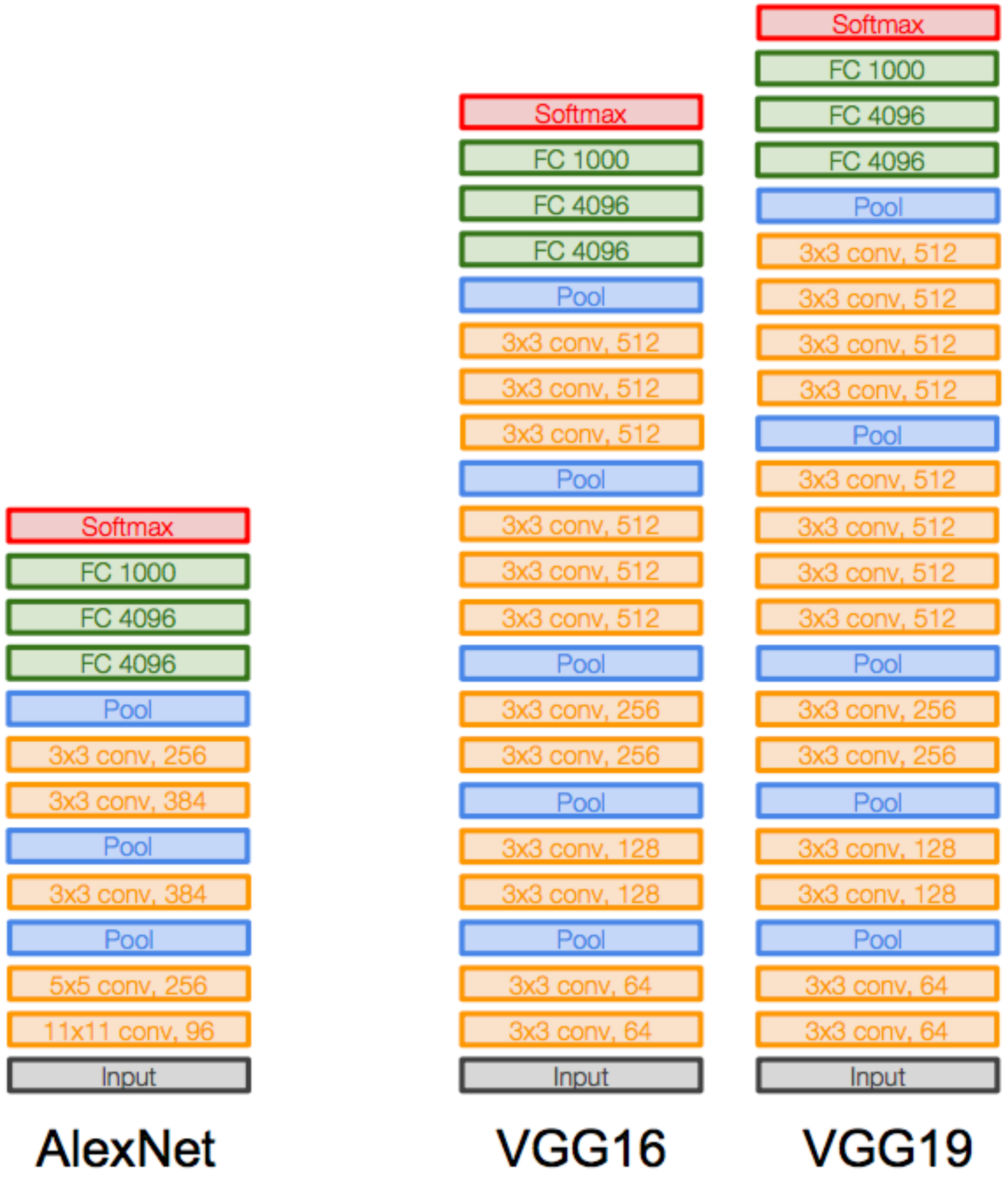
VGG Net

[Simonyan and Zisserman, 2014]

Trend:

— smaller filters (3 x 3)

— deeper network (16 or 19 vs. 8 in AlexNet)



VGG Net

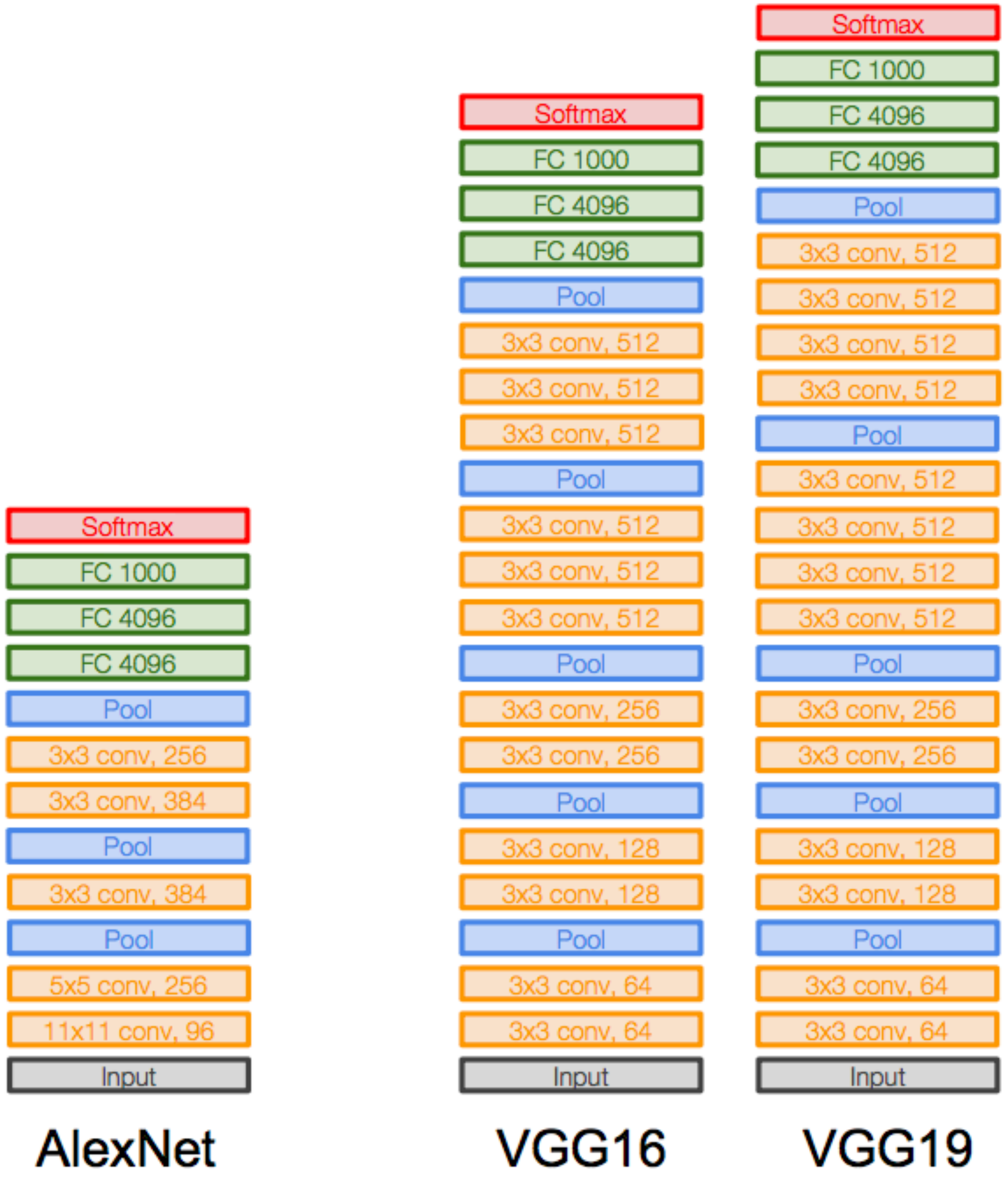
[Simonyan and Zisserman, 2014]

Trend:

— smaller filters (3 x 3)

— deeper network (16 or 19 vs. 8 in AlexNet)

Why?



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

VGG Net

[Simonyan and Zisserman, 2014]

Trend:

— smaller filters (3 x 3)

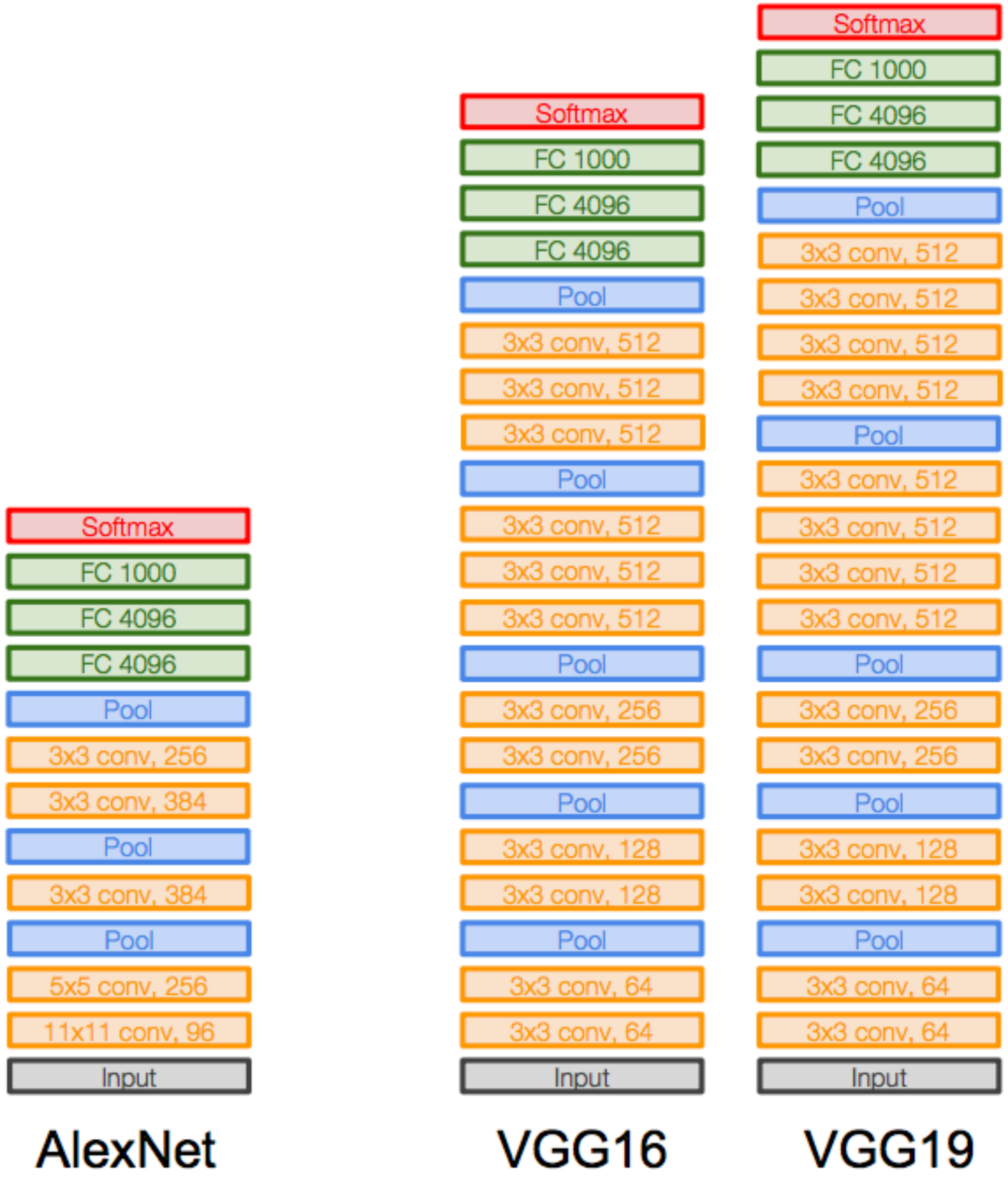
— deeper network (16 or 19 vs. 8 in AlexNet)

Why?

— **receptive field** of a 3 layer ConvNet with filter size = 3x3 is the same as 1 layer ConvNet with filter 7x7 (at stride 1)

— deeper = **more non-linearity** (so richer filters)

— **fewer parameters**

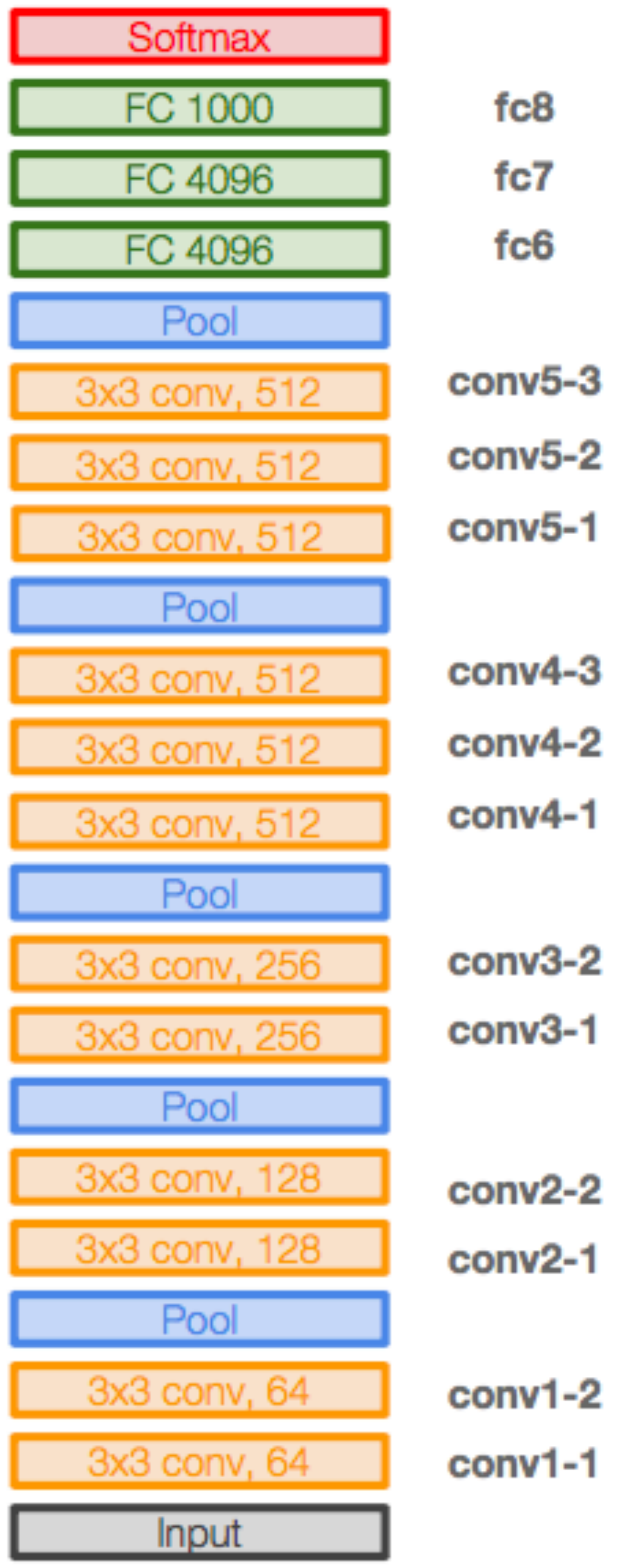


VGG Net

[Simonyan and Zisserman, 2014]

(not counting biases)

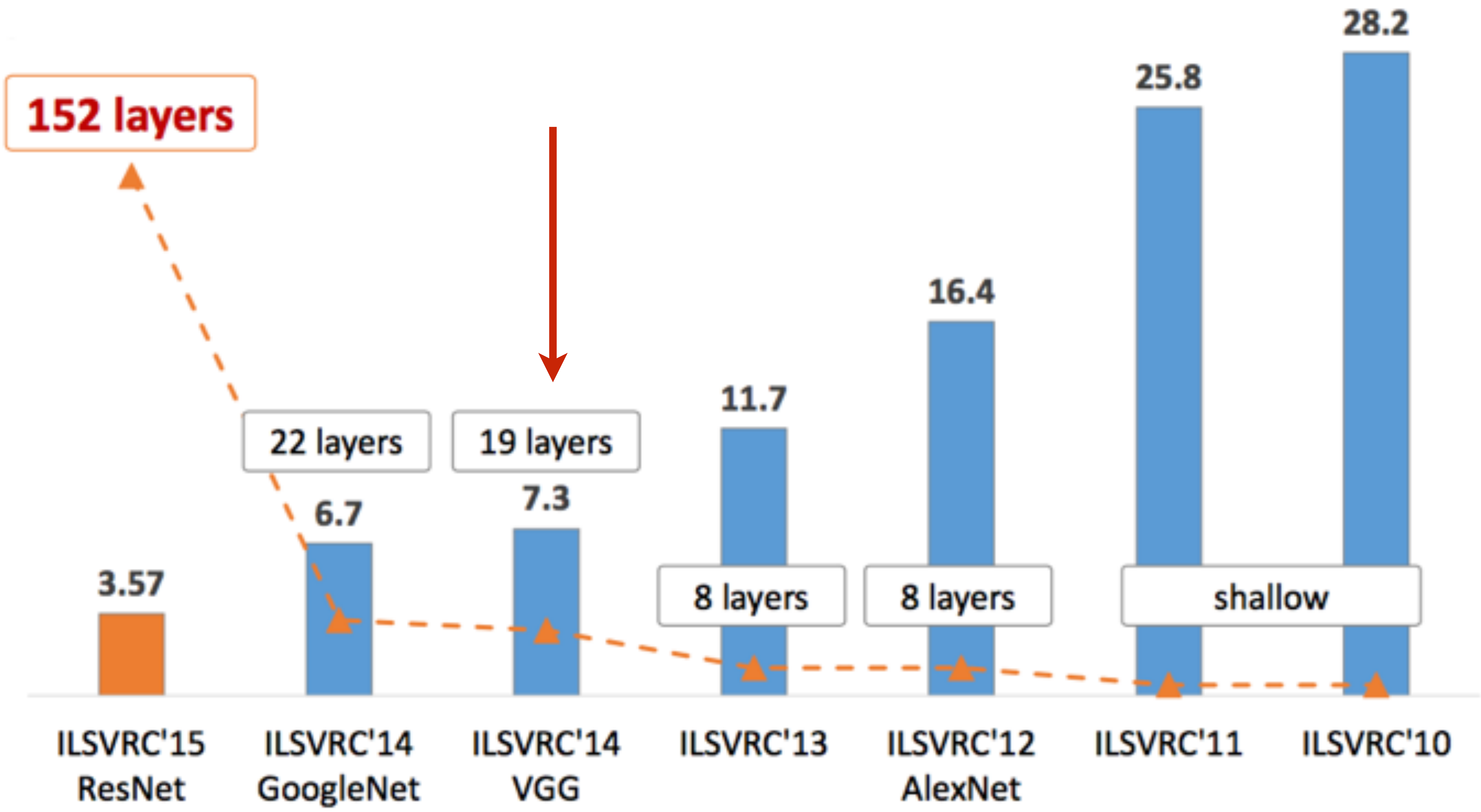
INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000



VGG16

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ILSVRC winner 2012



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet

[Szegedy et al., 2014]

even deeper network with **computational efficiency**

— 22 layers

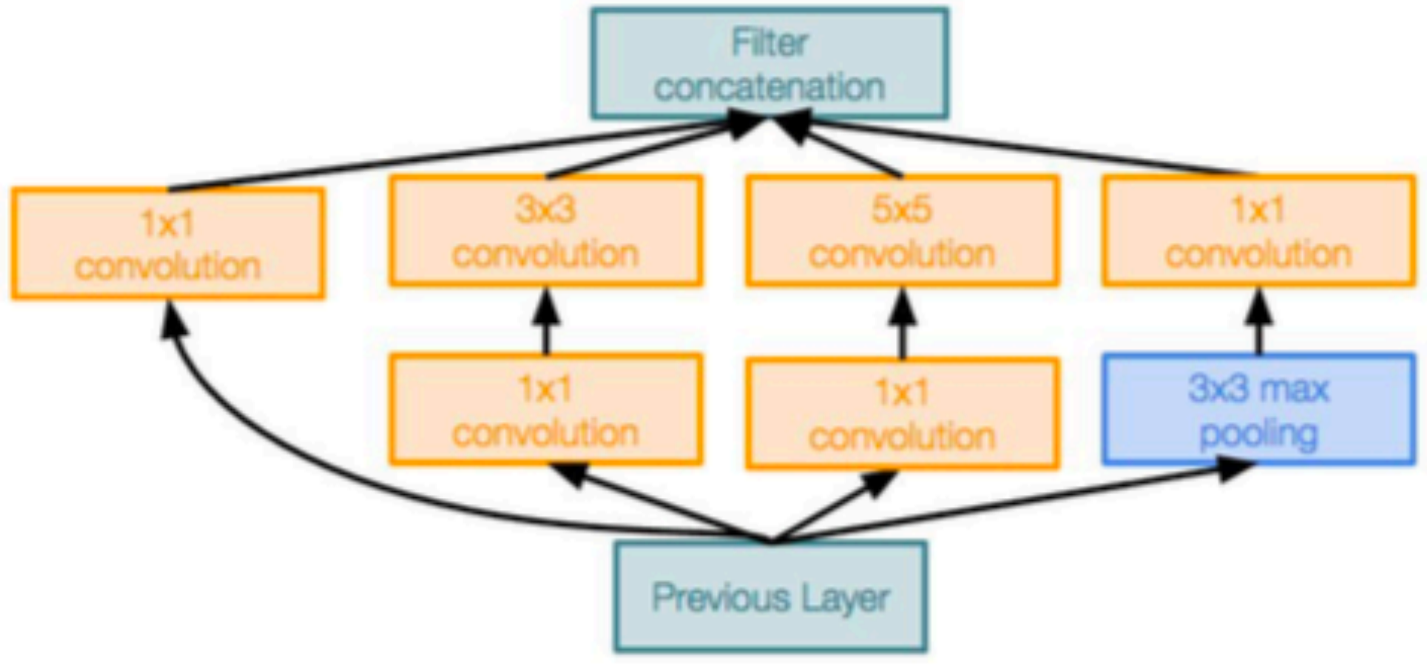
— Efficient "Inception" module

— No FC layers

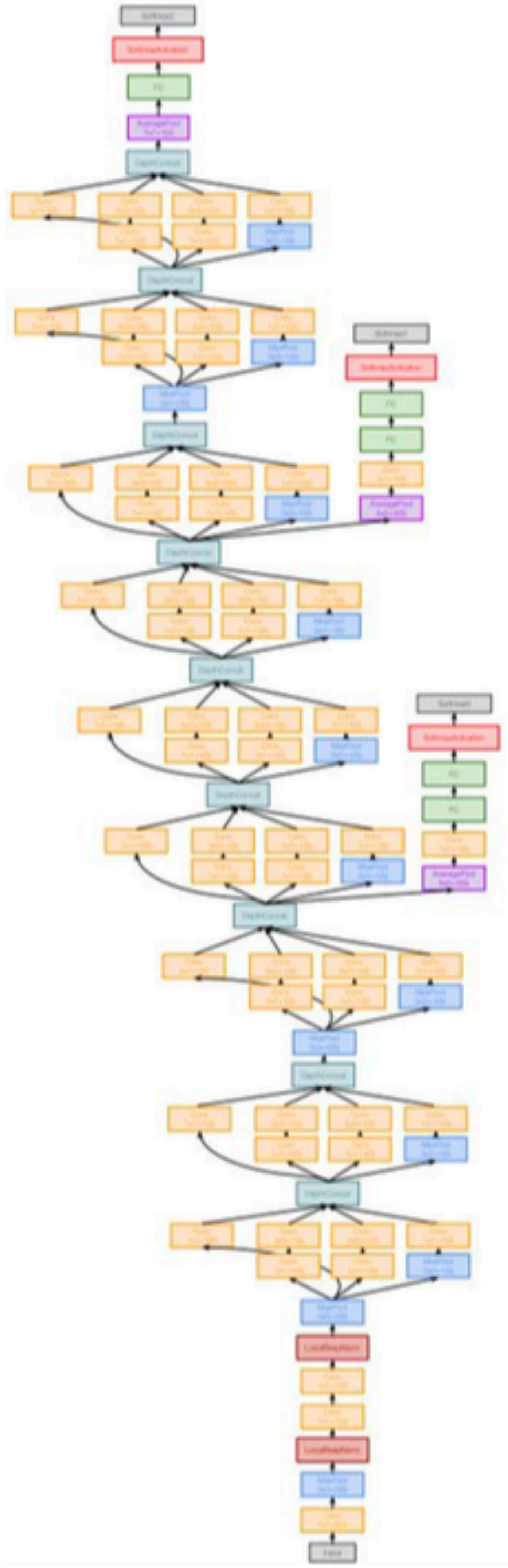
— Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)



Inception module

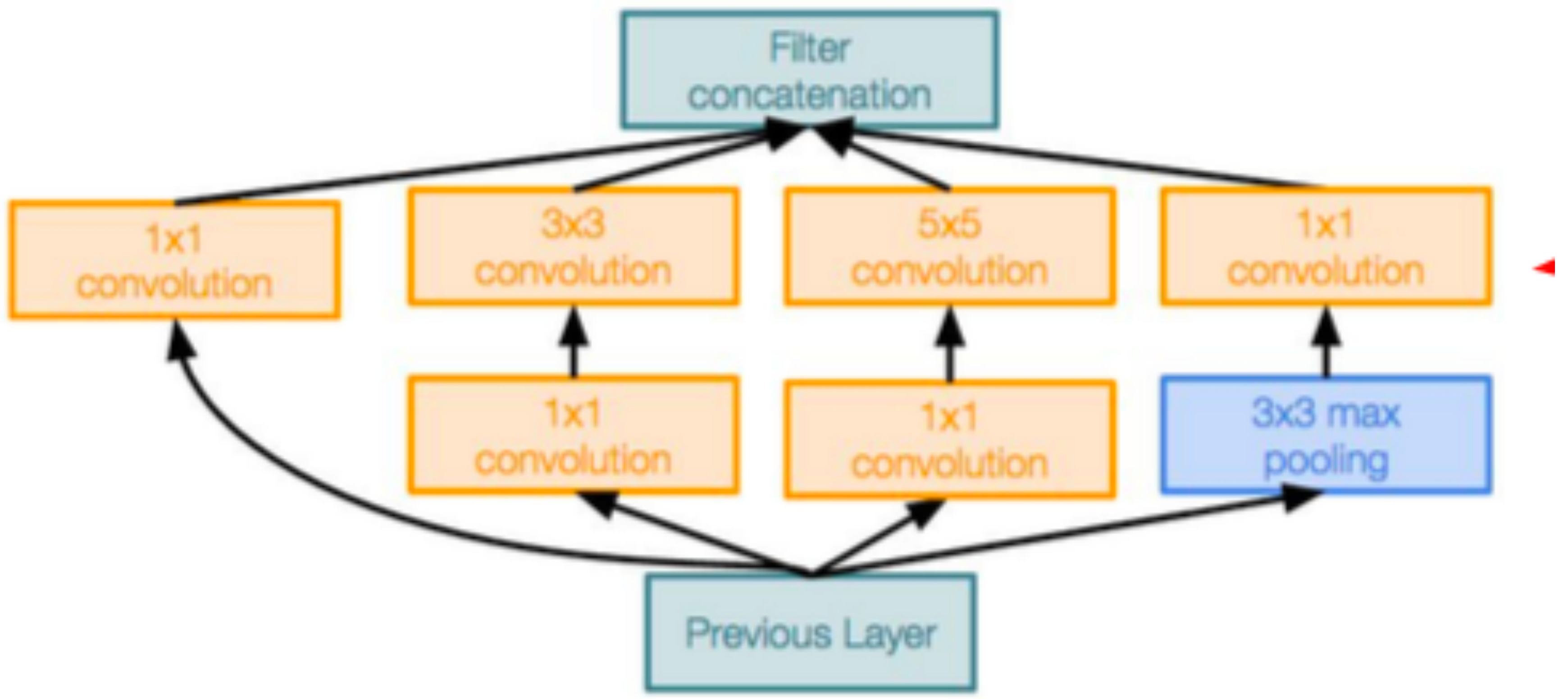


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

Idea: design good local topology ("network within network") and then stack these modules



Inception module

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

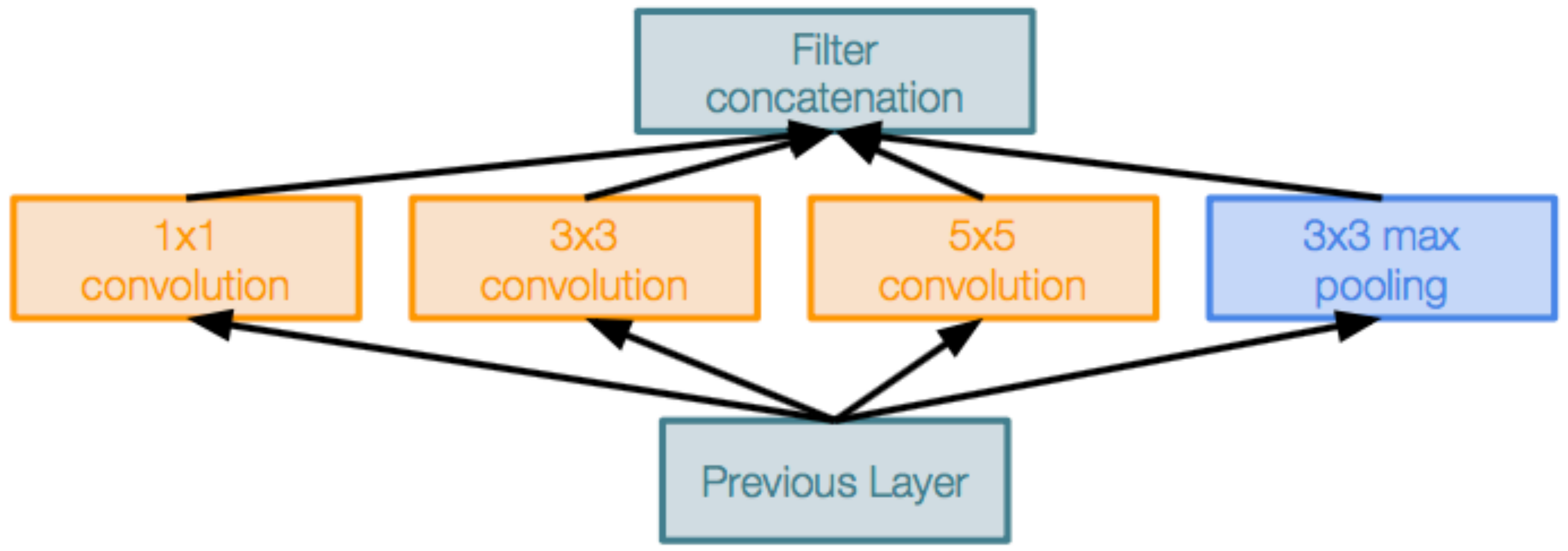
Idea: design good local topology ("network within network") and then stack these modules

Apply **parallel filter operations** on the input from previous layer

— Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)

— Pooling operation (3x3)

Concatenate all filter outputs together at output depth-wise



Naive Inception module

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

Idea: design good local topology ("network within network") and then stack these modules

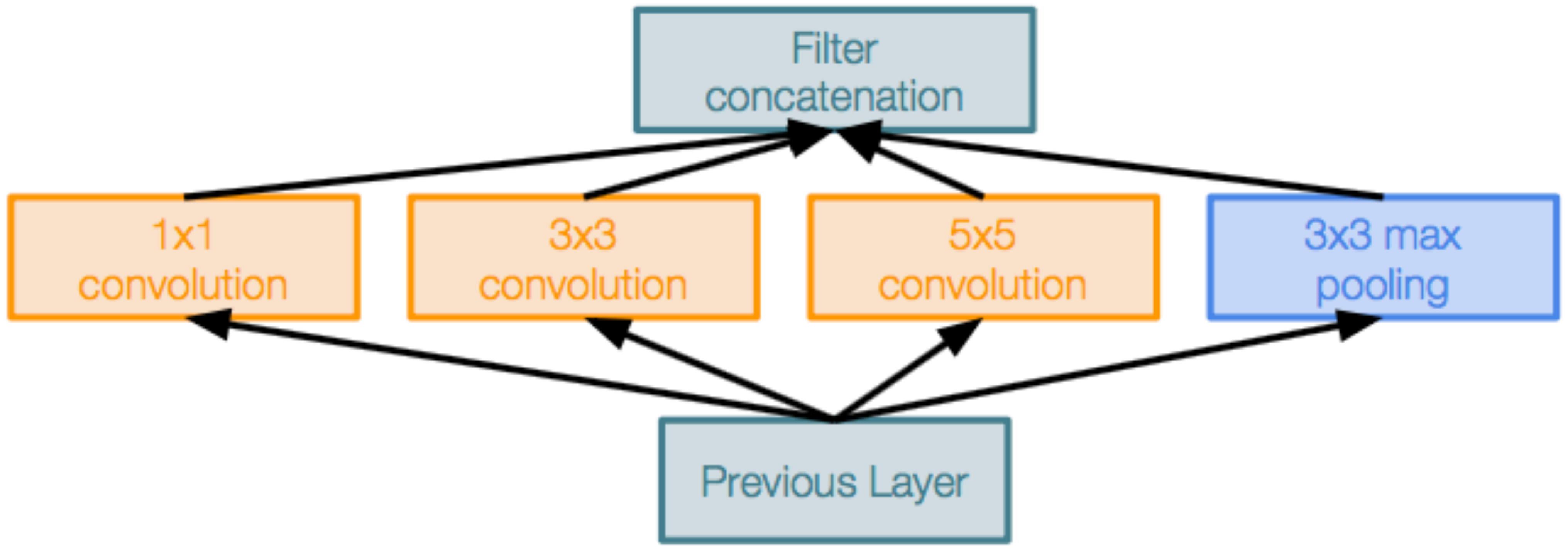
What's the problem?

Apply **parallel filter operations** on the input from previous layer

— Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)

— Pooling operation (3x3)

Concatenate all filter outputs together at output depth-wise



Naive Inception module

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

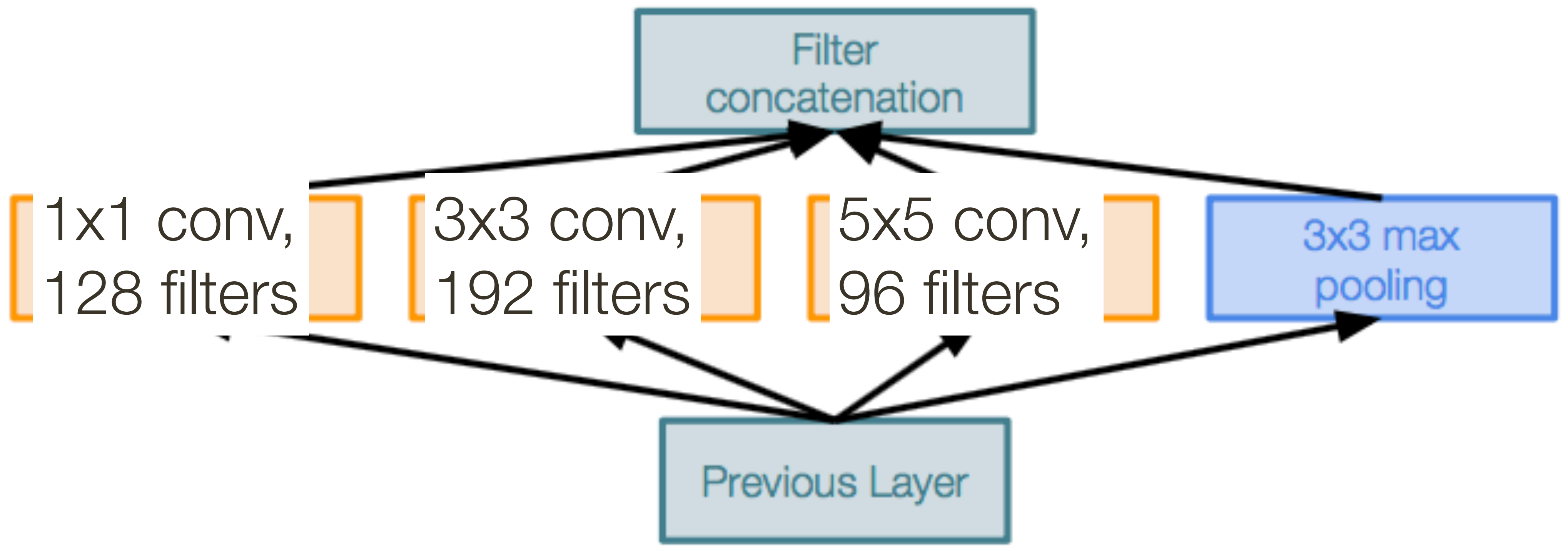
Idea: design good local topology ("network within network") and then stack these modules

Apply **parallel filter operations** on the input from previous layer

— Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)

— Pooling operation (3x3)

Concatenate all filter outputs together at output depth-wise



28x28x256

Naive Inception module

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

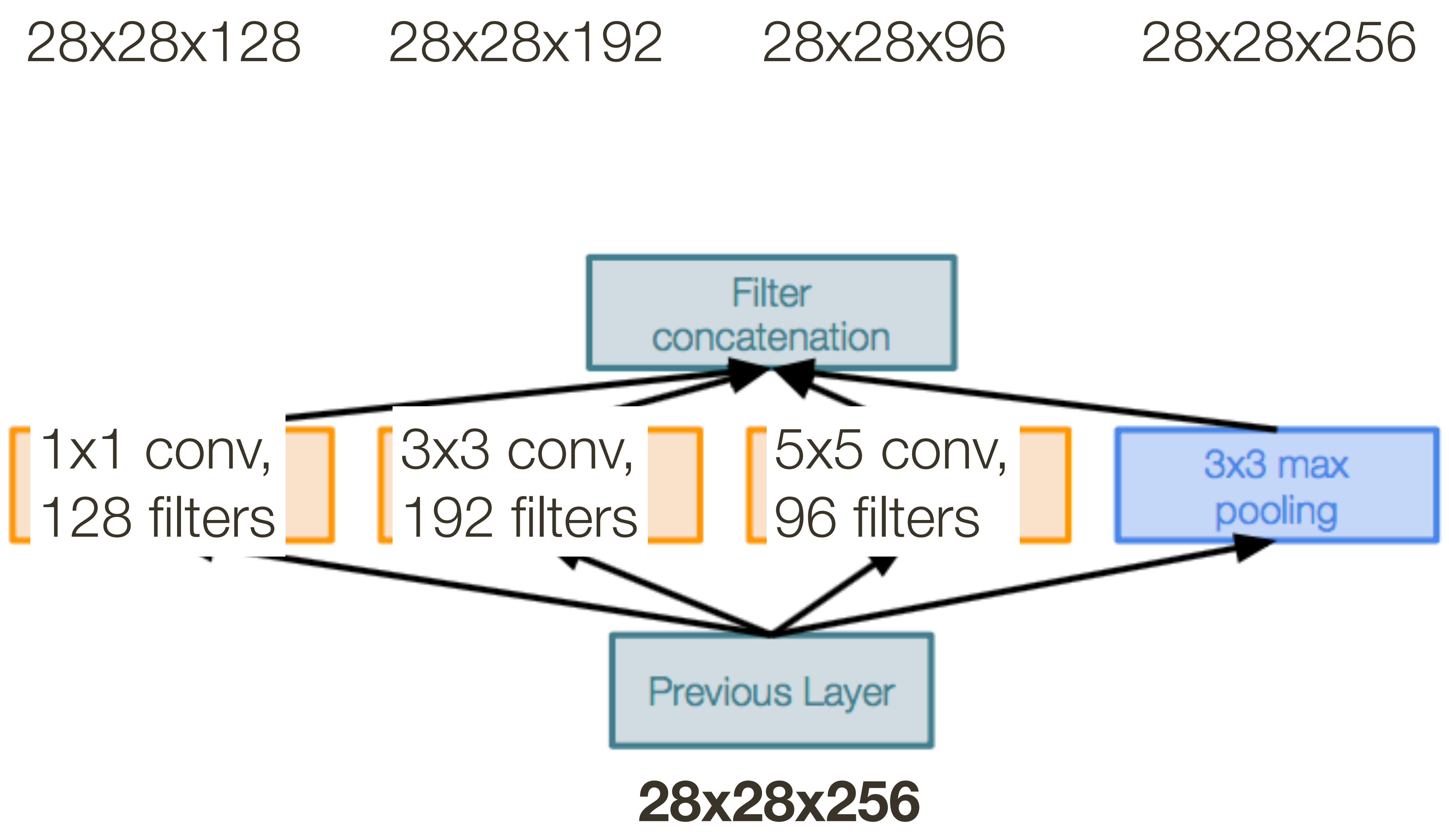
Idea: design good local topology ("network within network") and then stack these modules

Apply **parallel filter operations** on the input from previous layer

— Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)

— Pooling operation (3x3)

Concatenate all filter outputs together at output depth-wise



Naive Inception module

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

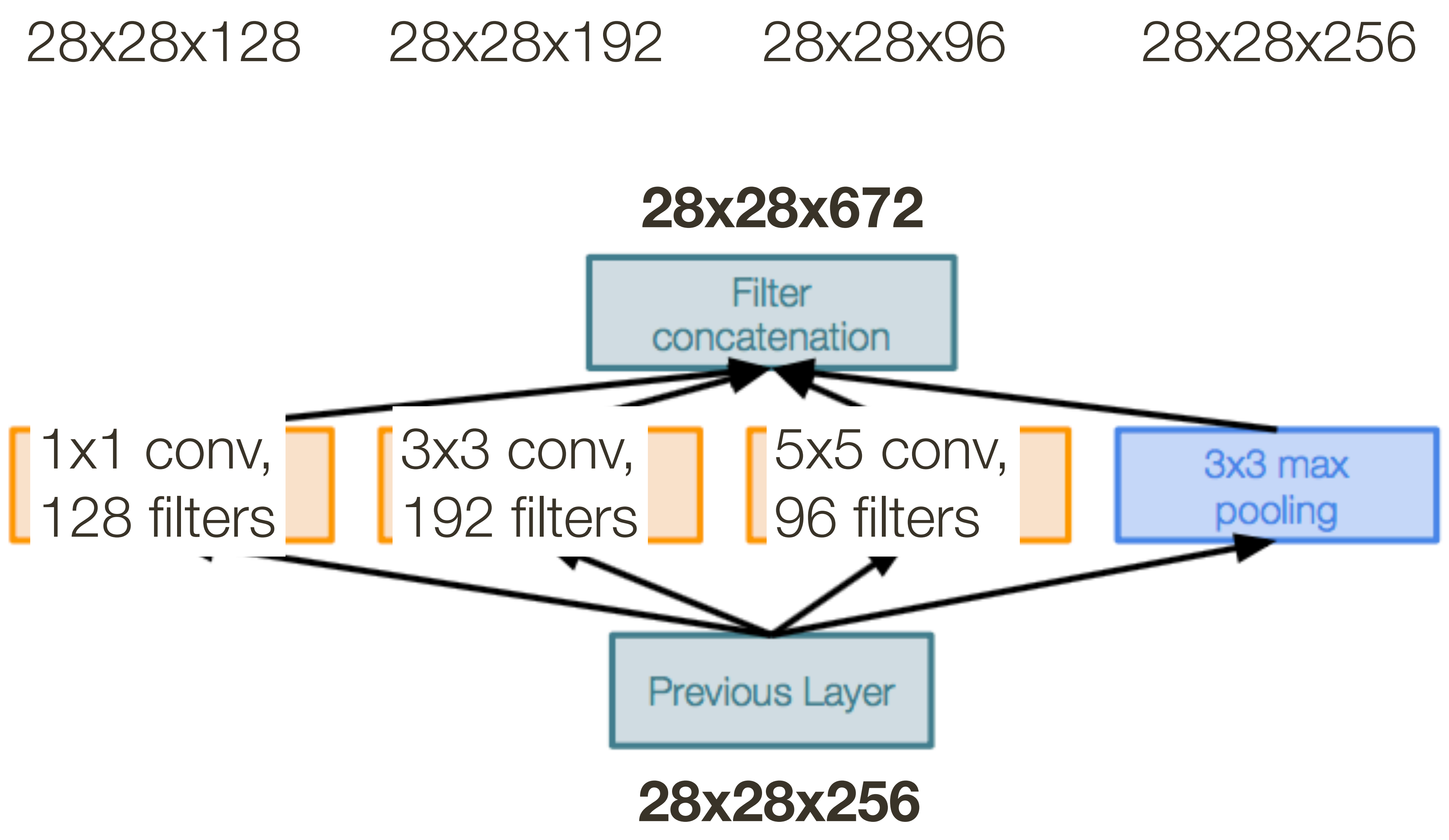
Idea: design good local topology ("network within network") and then stack these modules

Apply **parallel filter operations** on the input from previous layer

— Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)

— Pooling operation (3x3)

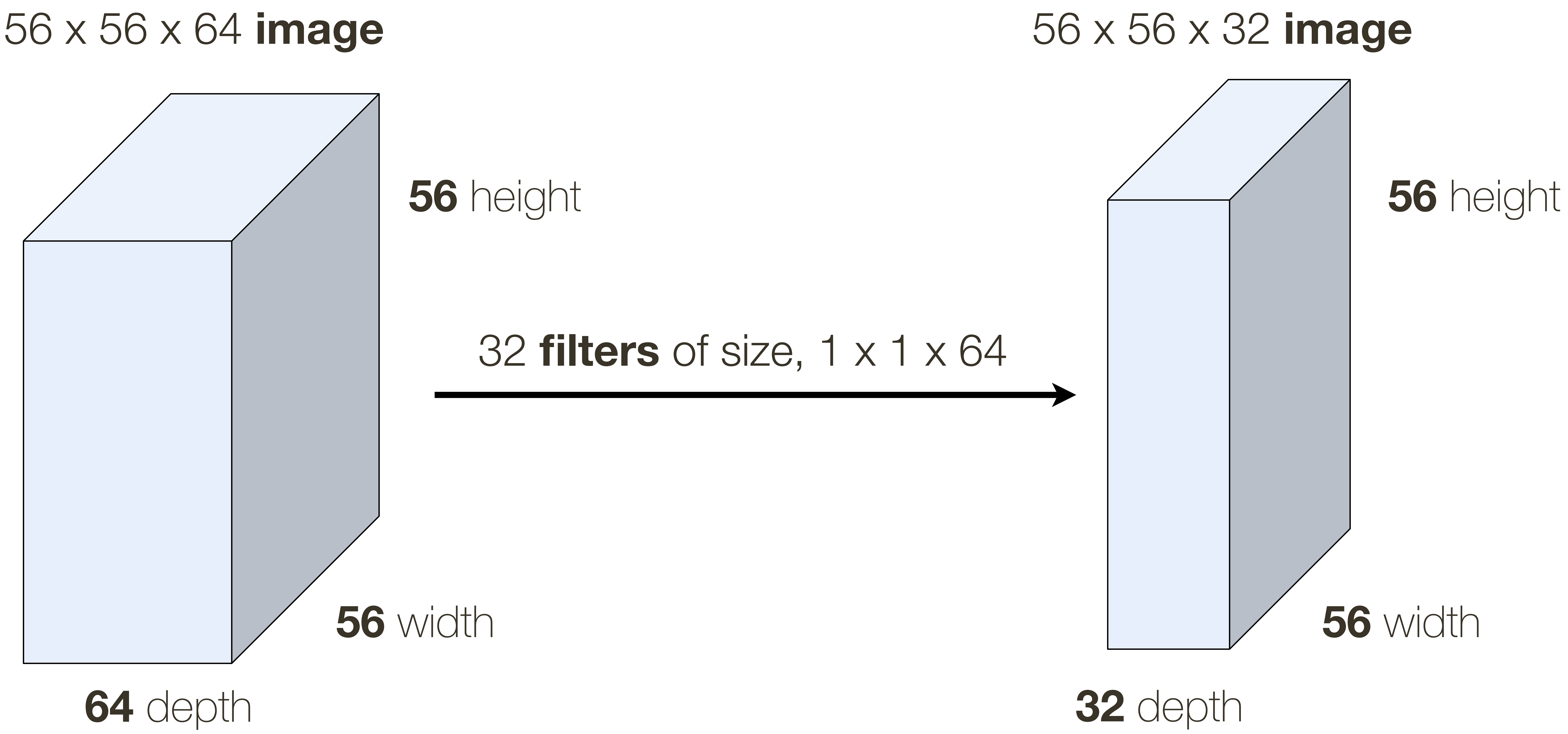
Concatenate all filter outputs together at output depth-wise



Naive Inception module

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Convolutional Layer: **1x1 convolutions**

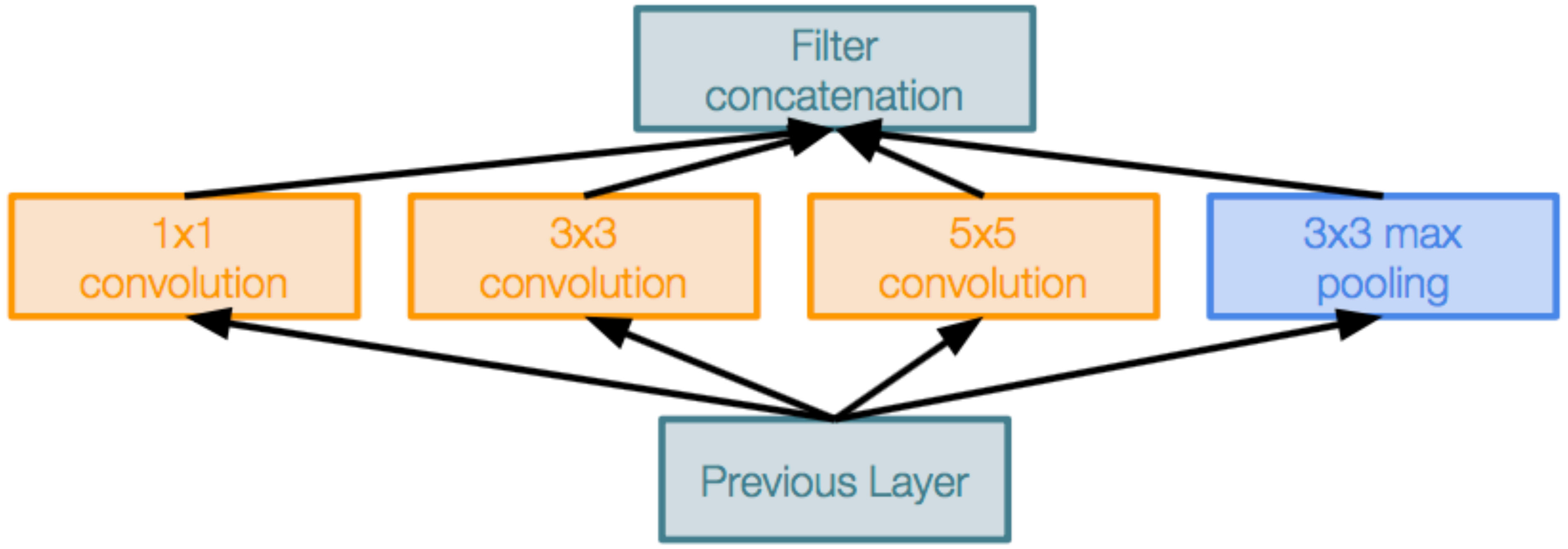


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet: Inception Module

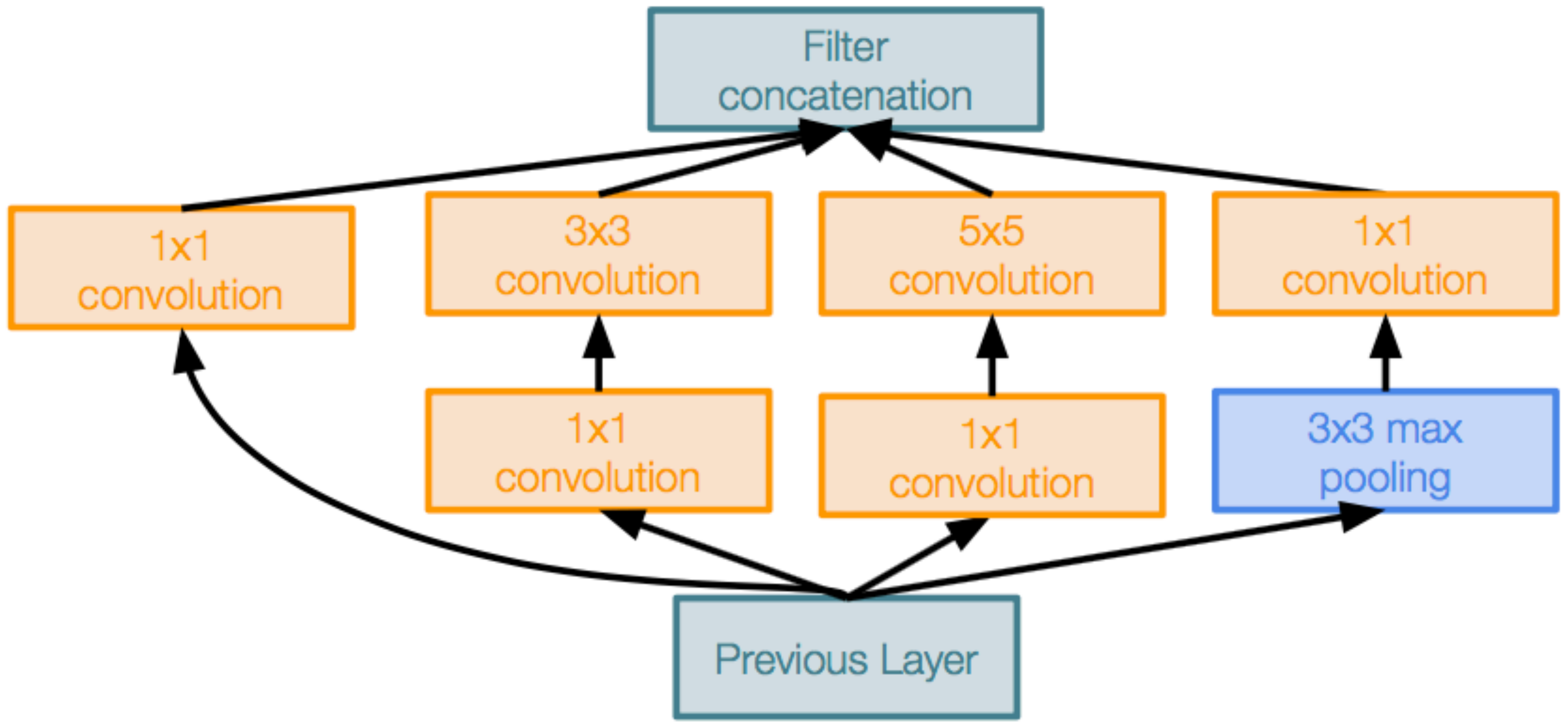
[Szegedy et al., 2014]

Idea: design good local topology ("network within network") and then stack these modules



Naive Inception module

1x1 "bottleneck" layers



Inception module with dimension reduction

saves approximately 60% of computations

GoogleLeNet

[Szegedy et al., 2014]

even deeper network with **computational efficiency**

— 22 layers

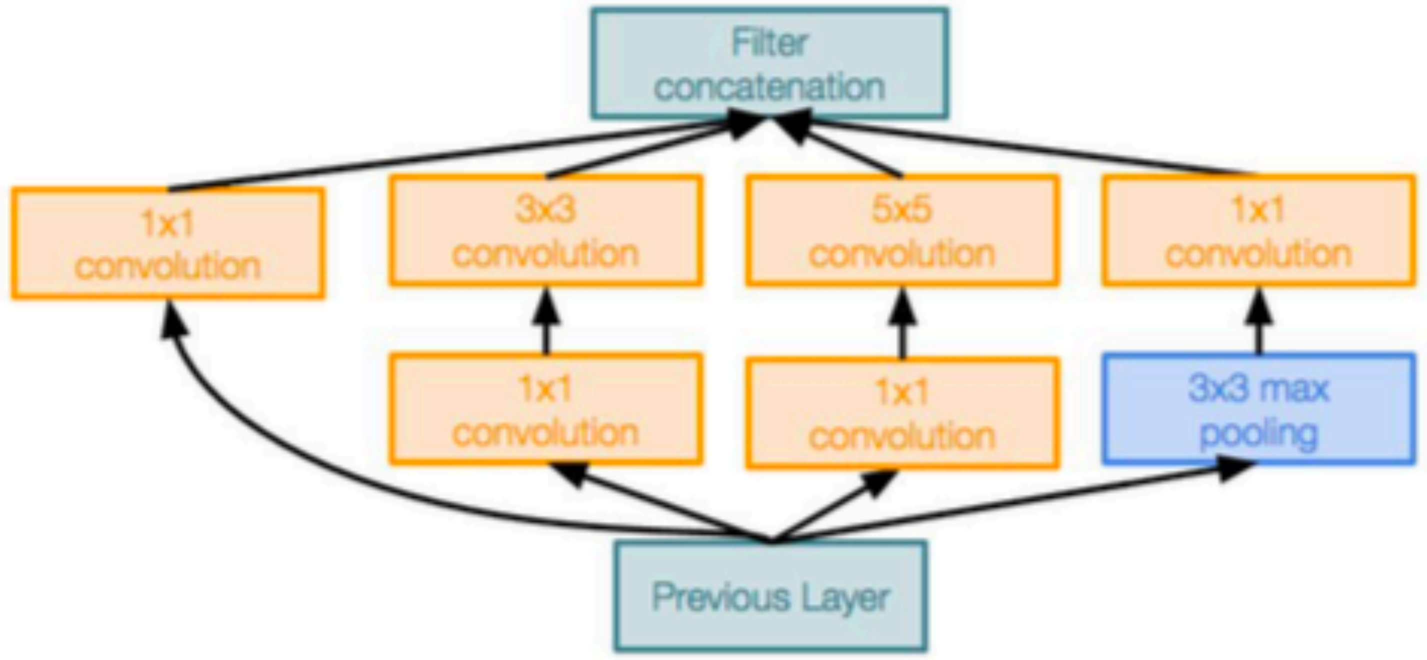
— Efficient "Inception" module

— No FC layers

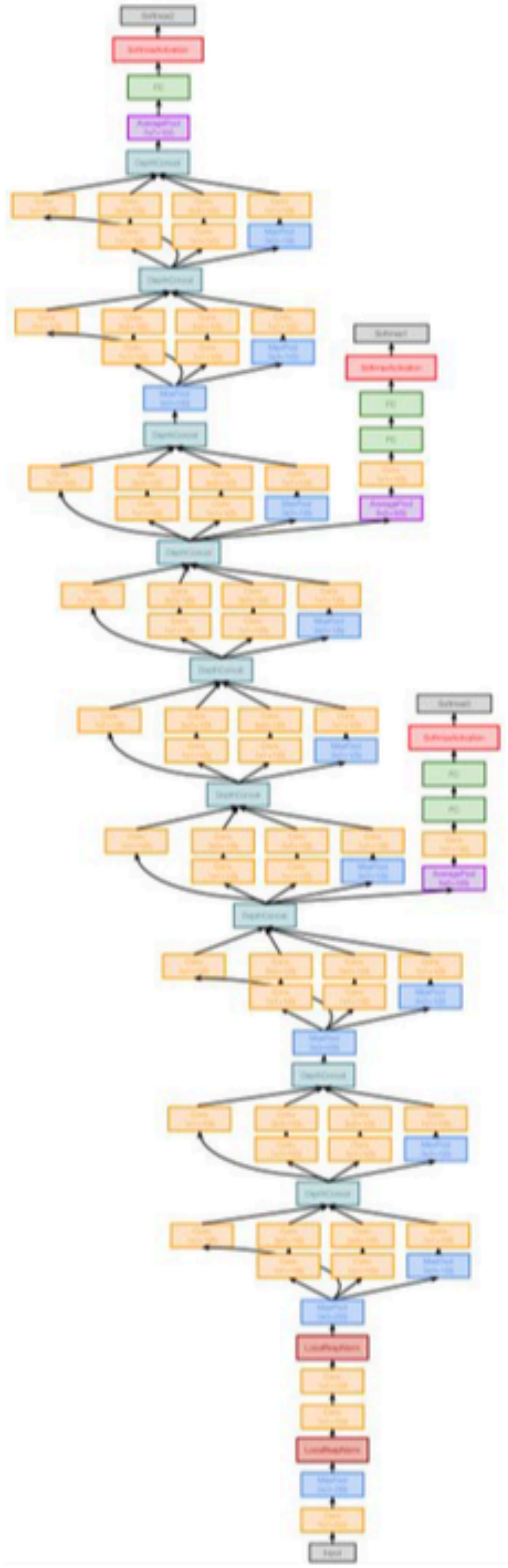
— Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)



Inception module



GoogleLeNet

[Szegedy et al., 2014]

even deeper network with **computational efficiency**

— 22 layers

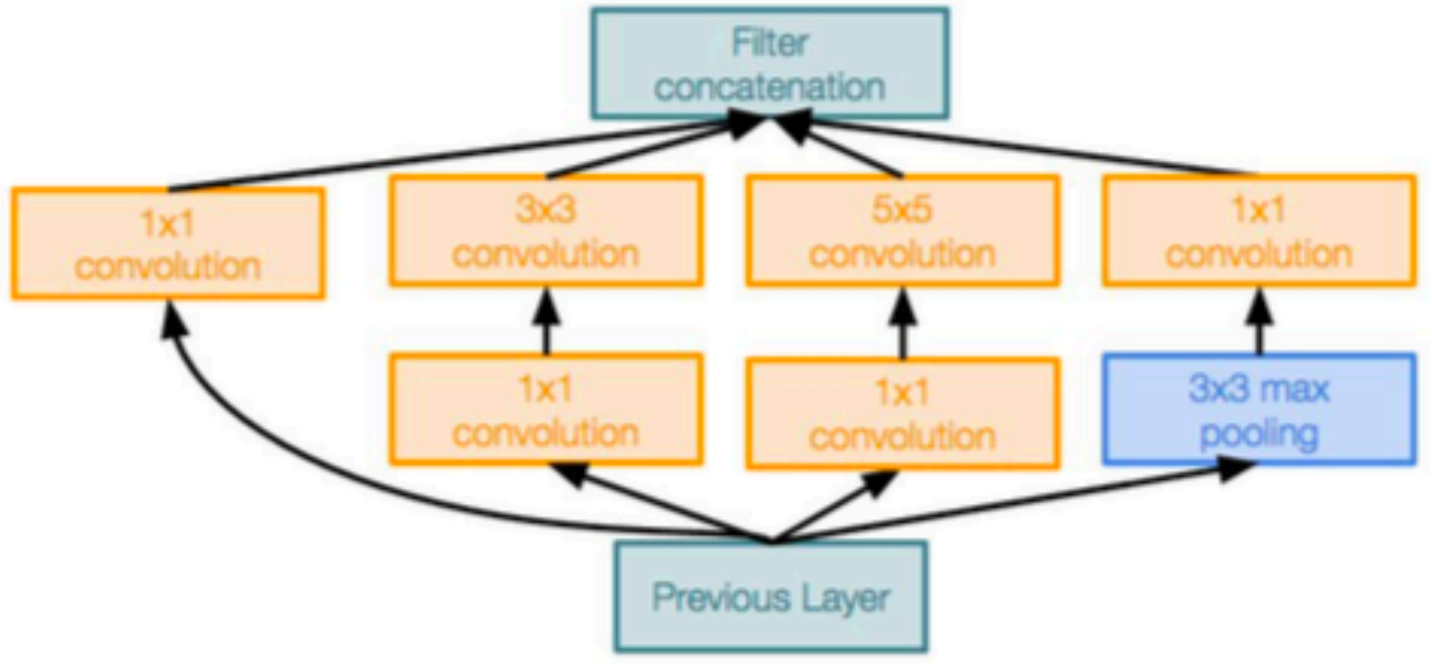
— Efficient “Inception” module

— No FC layers

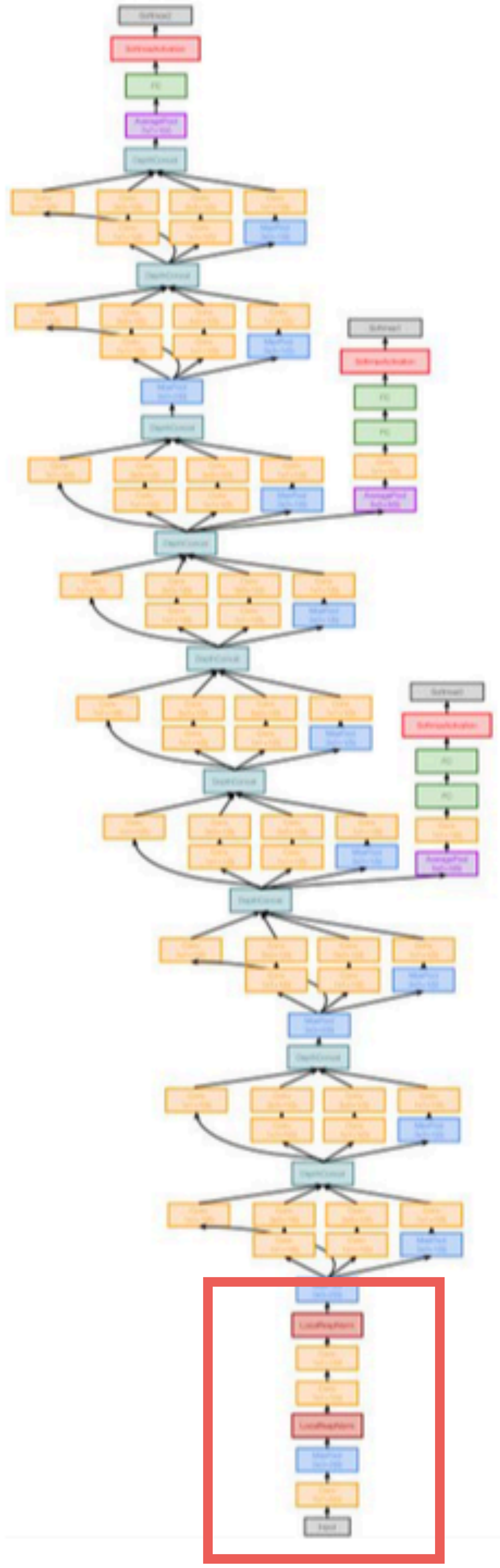
— Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)



Inception module



* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet

[Szegedy et al., 2014]

even deeper network with **computational efficiency**

— 22 layers

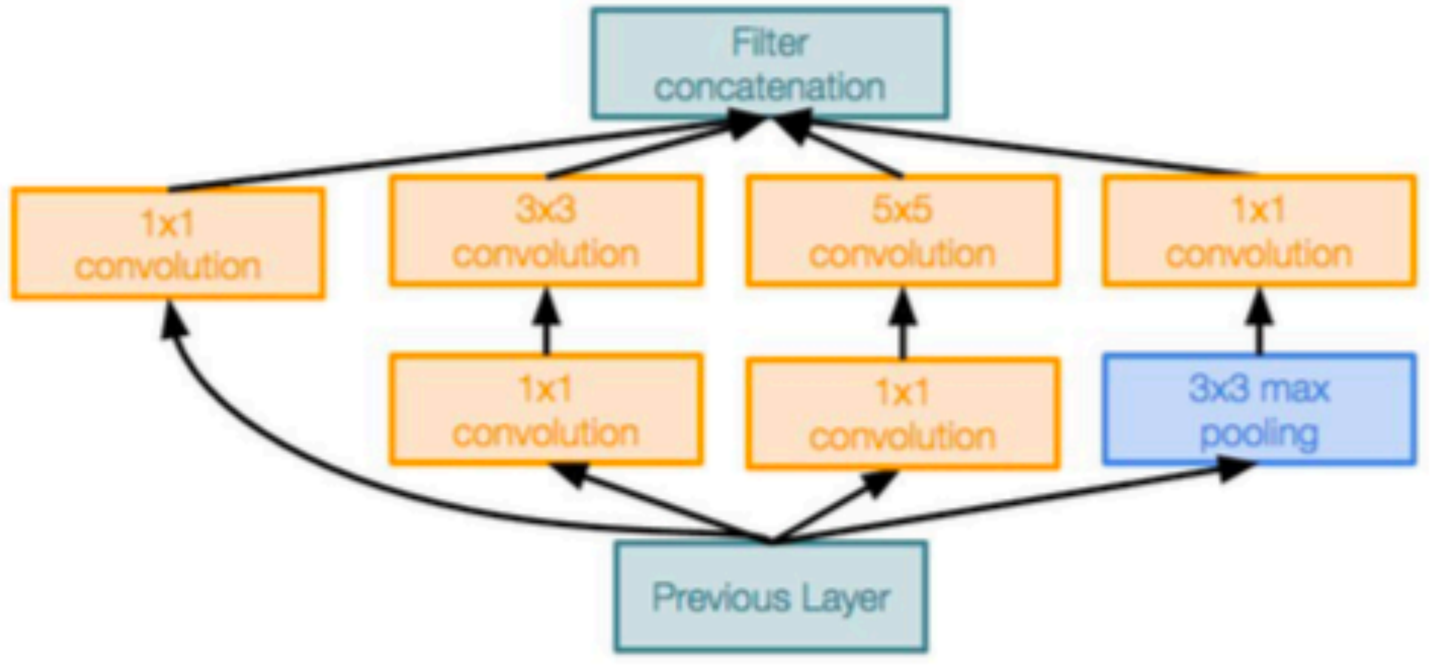
— Efficient "Inception" module

— No FC layers

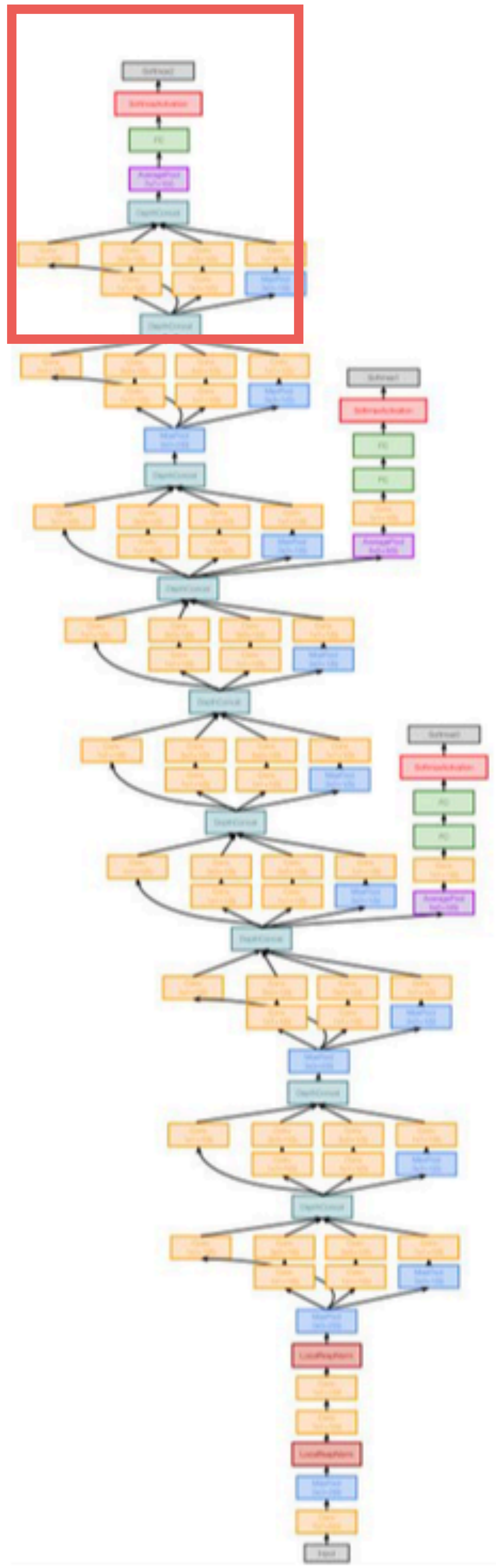
— Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)



Inception module



GoogleLeNet

[Szegedy et al., 2014]

even deeper network with **computational efficiency**

— 22 layers

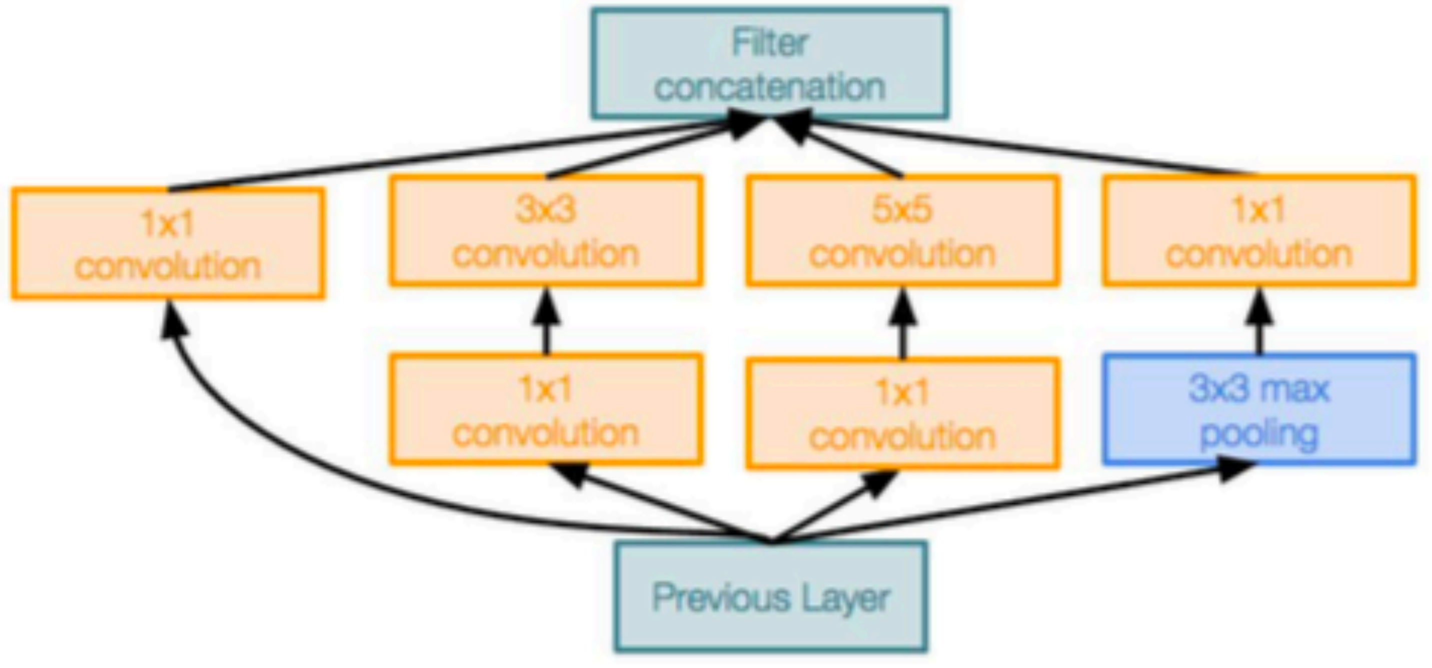
— Efficient “Inception” module

— No FC layers

— Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)



Inception module