THE UNIVERSITY OF BRITISH COLUMBIA

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

Lecture 5: Convolutional Neural Networks (Part 2)

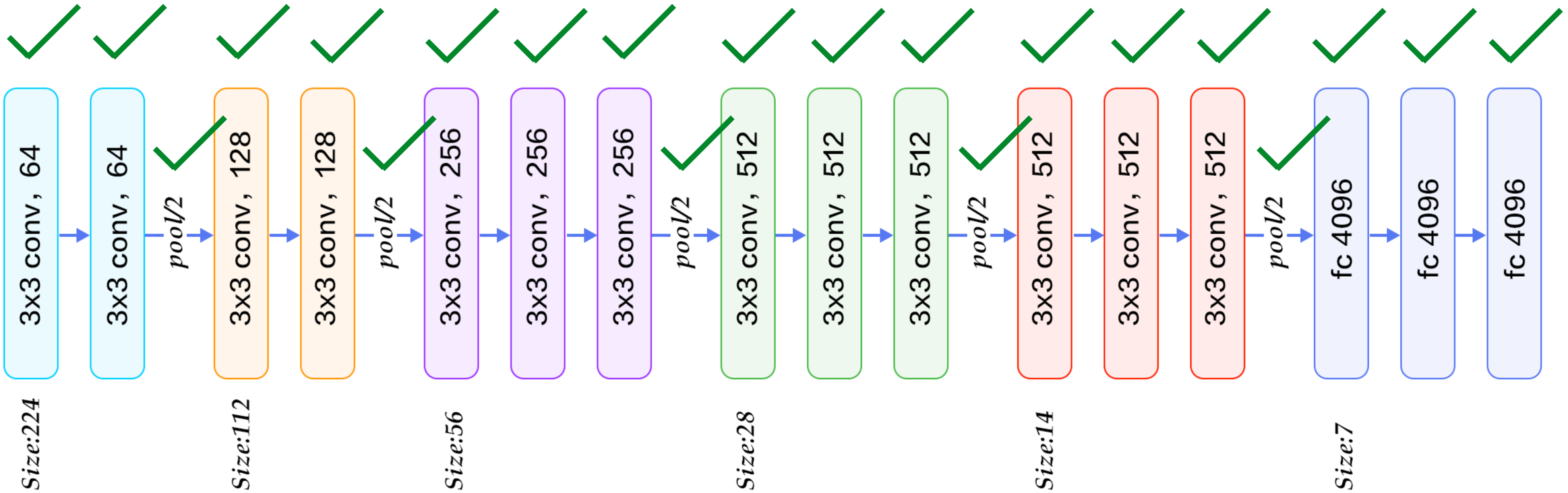
Logistics:

Assignment 2 is out

Groups + idea for project by **January 31st**

Paper list to be posted soon-ish

Last Time: Convolutional Neural Networks

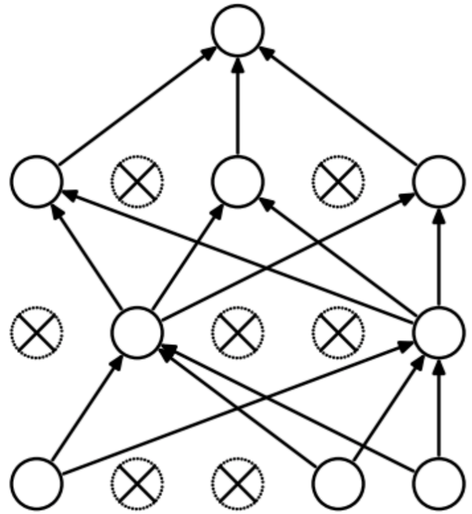


VGG-16 Network

Improving **Single Model**

Regularization

- L2, L1
- Dropout / Inverted Dropout
- Data augmentation



Dropout

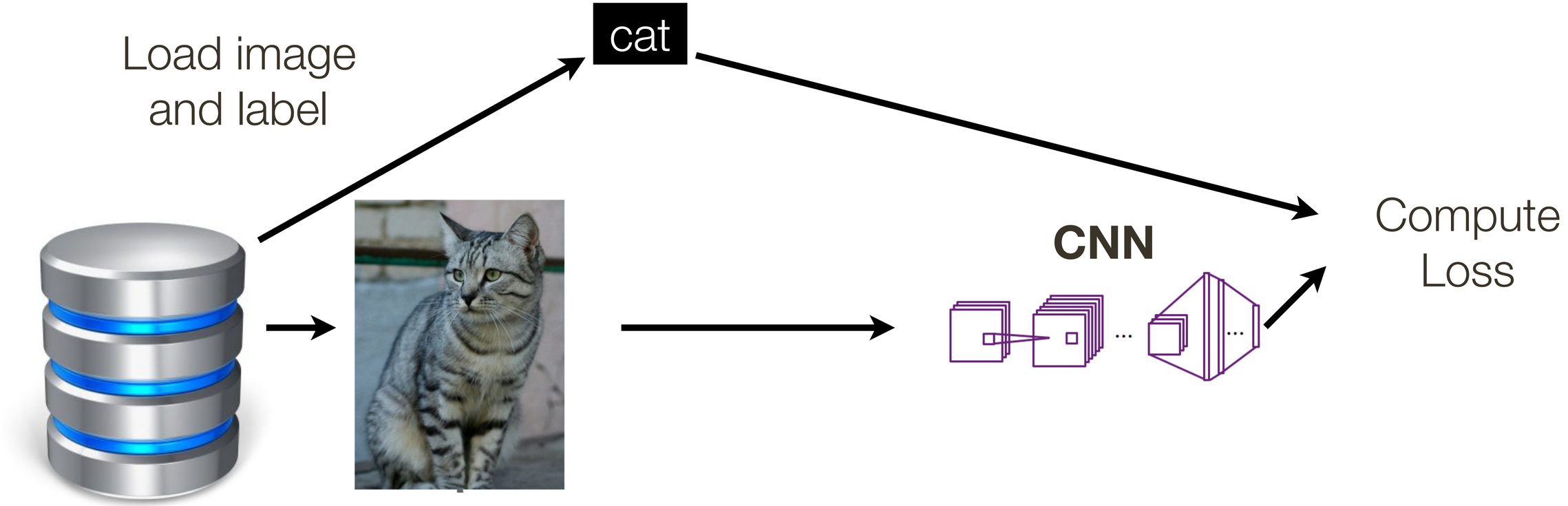
L2 Regularization: Learn a more (dense) distributed representation

$$R(\mathbf{W}) = ||\mathbf{W}||_2 = \sum_i \sum_j \mathbf{W}_{i,j}^2$$

L1 Regularization: Learn a sparse representation (few non-zero wight elements)

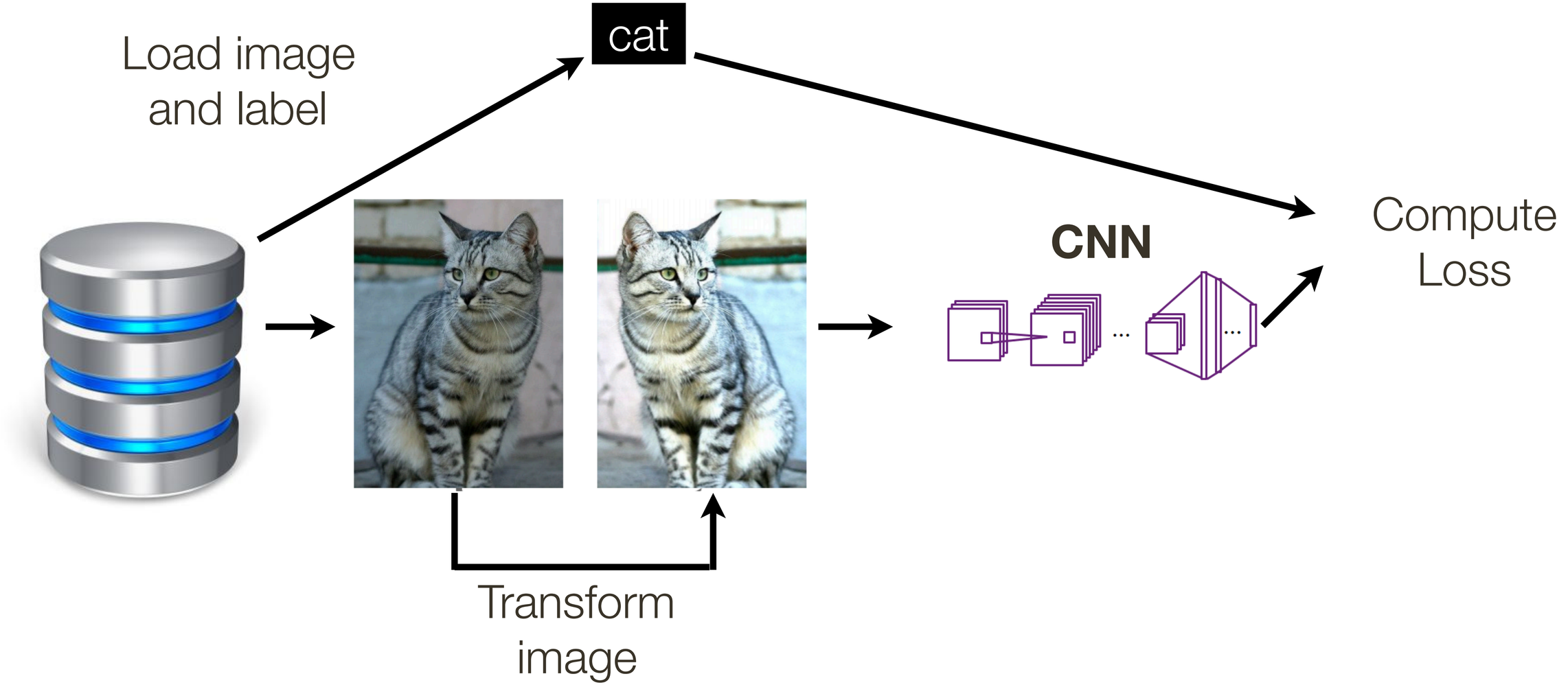
$$R(\mathbf{W}) = ||\mathbf{W}||_1 = \sum_i \sum_j |\mathbf{W}_{i,j}|$$

Regularization: Data Augmentation



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

Regularization: Data Augmentation



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

Regularization: Data Augmentation

Horizontal flips

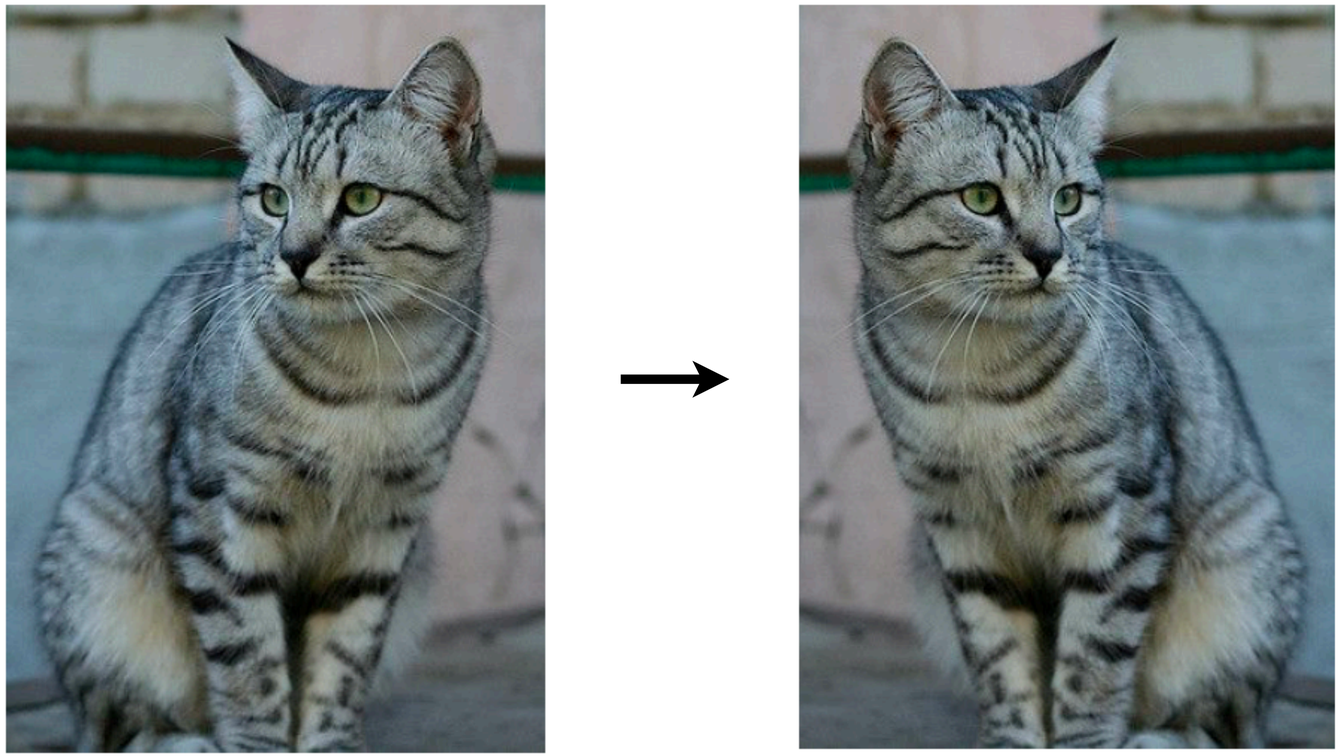
Random crops & scales

Color Jitter

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

Regularization: Data Augmentation

Horizontal flips Random crops & scales Color Jitter



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

Regularization: Data Augmentation

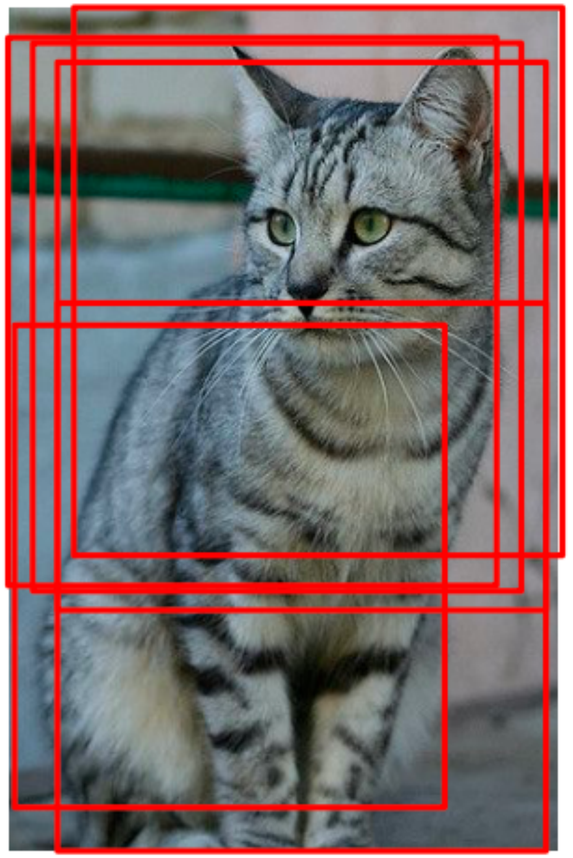
Horizontal flips **Random crops & scales** Color Jitter

Training: sample random crops and scales
e.g., ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short size = L
3. Sample random 224x224 patch

Testing: average a fix set of crops
e.g., ResNet:

1. Resize image to 5 scales (224, 256, 384, 480, 640)
2. For each image use 10 224x224 crops: 4 corners + center, + flips



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

Regularization: Data Augmentation

Horizontal flips

Random crops & scales

Color Jitter

Random perturbations in contrast and brightness

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

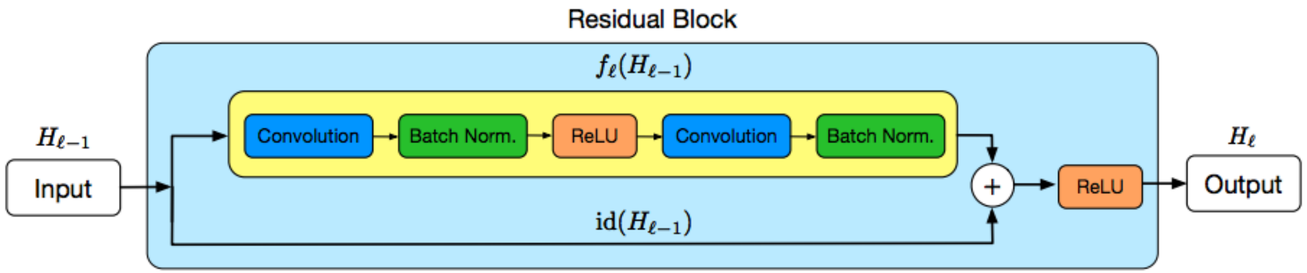
Regularization: Stochastic Depth

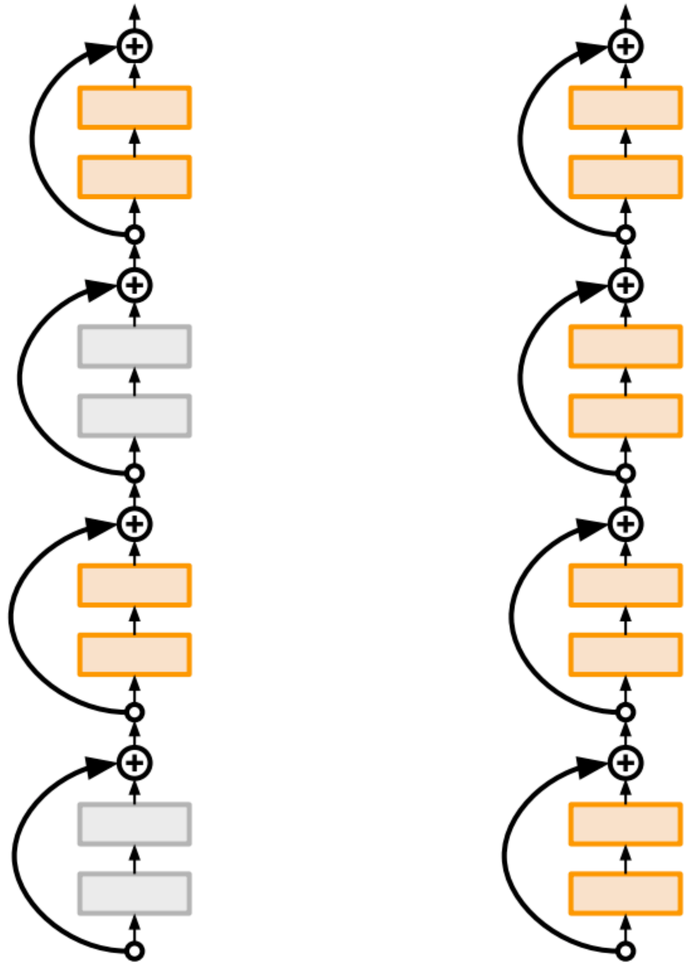
[Huang et al., ECCV 2016]

Effectively “dropout” but for layers

Stochastically with some probability **turn off some layer** (for each batch)

Effectively trains a collection of neural networks





* 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

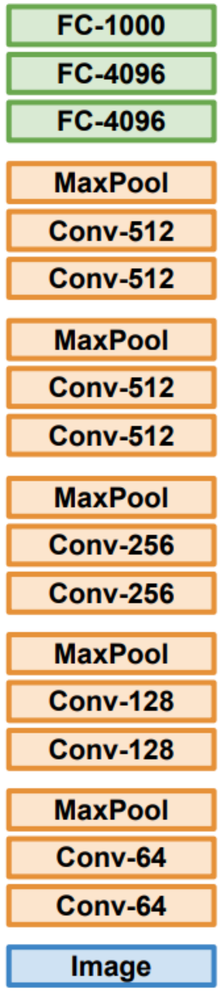
This strategy is PERVASIVE.

* 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**



Why on **ImageNet**?

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

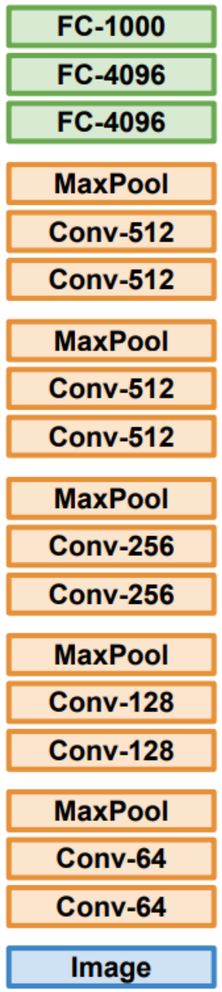
However, for some tasks we would need to start with something else (e.g., videos for optical flow)

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

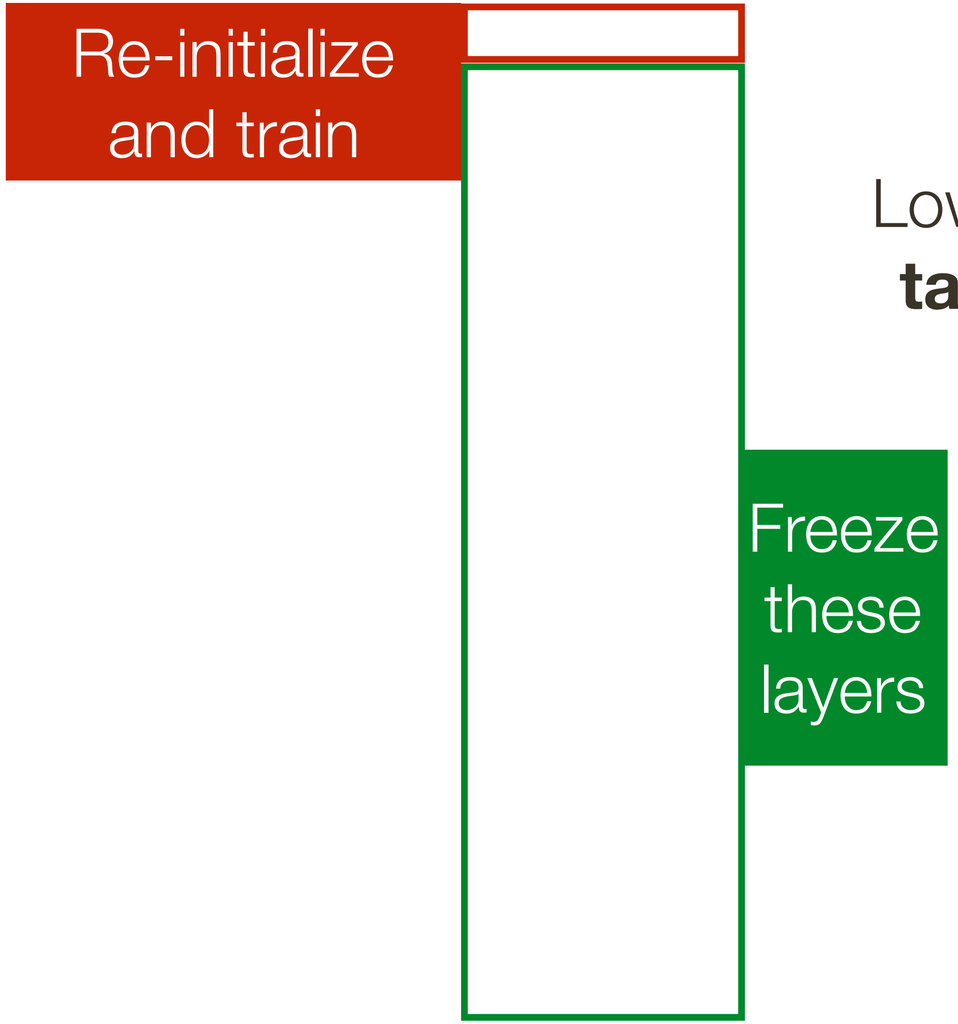
Transfer Learning with CNNs

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

Train on **ImageNet**



Small dataset with C classes



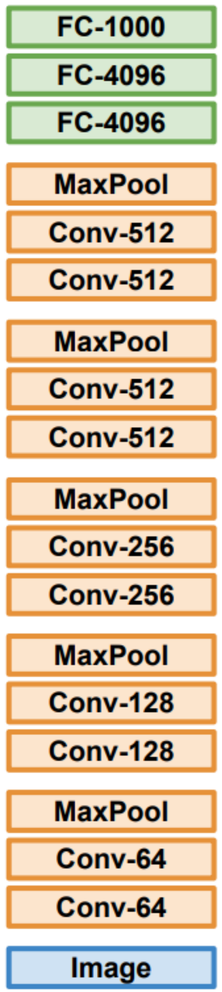
Lower levels of the CNN are at **task independent** anyways

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

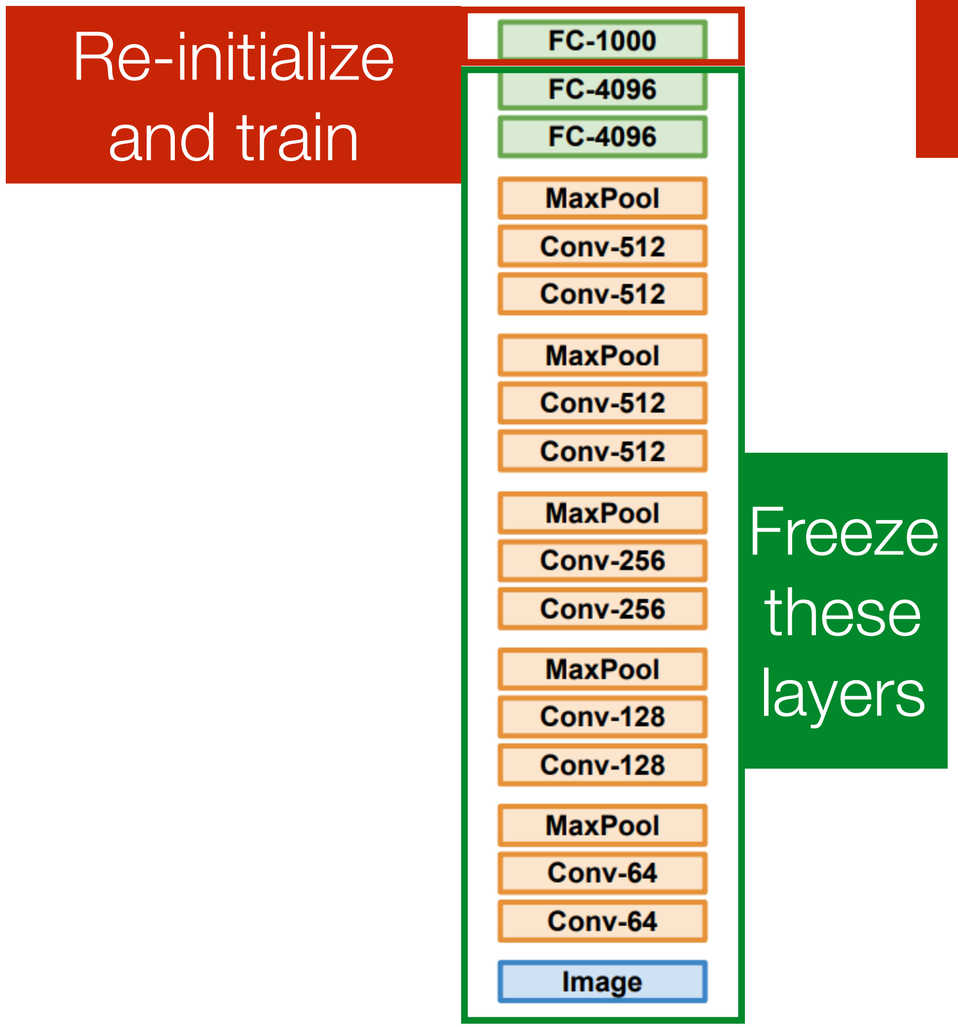
Transfer Learning with CNNs

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

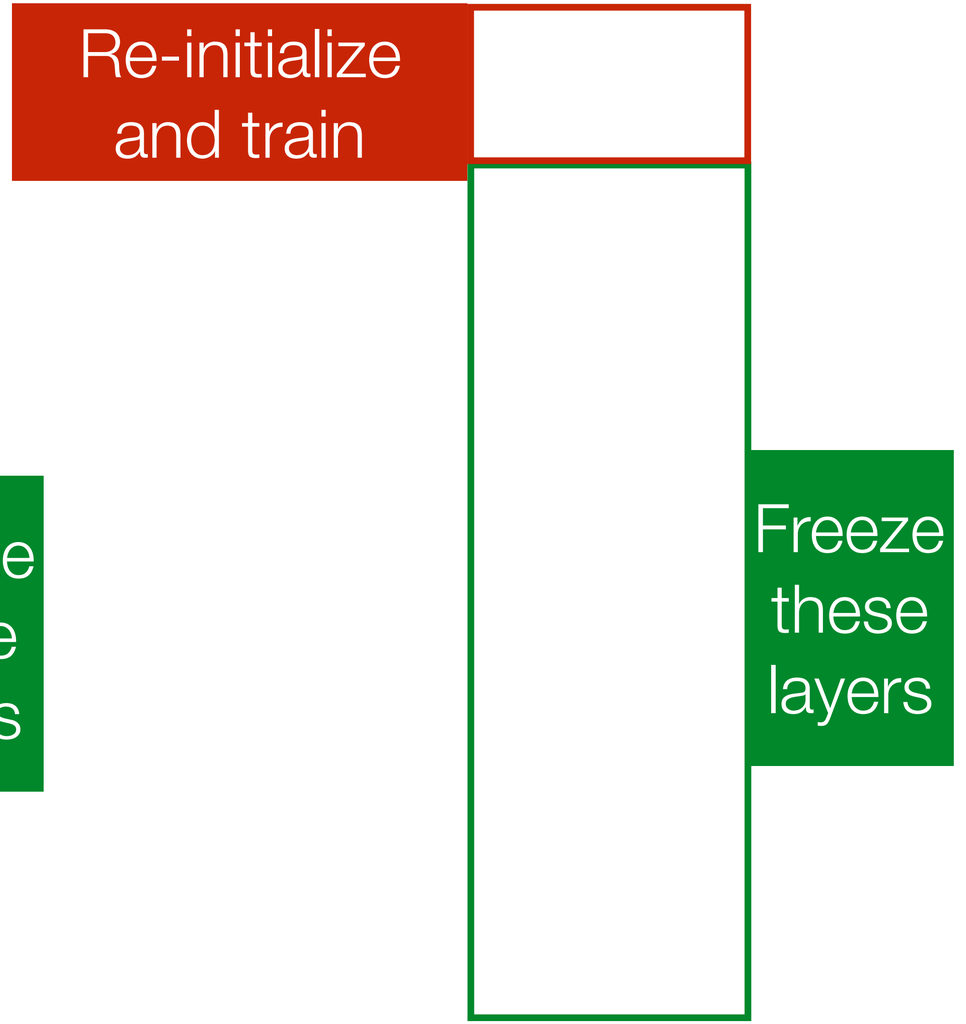
Train on **ImageNet**



Small dataset with C classes

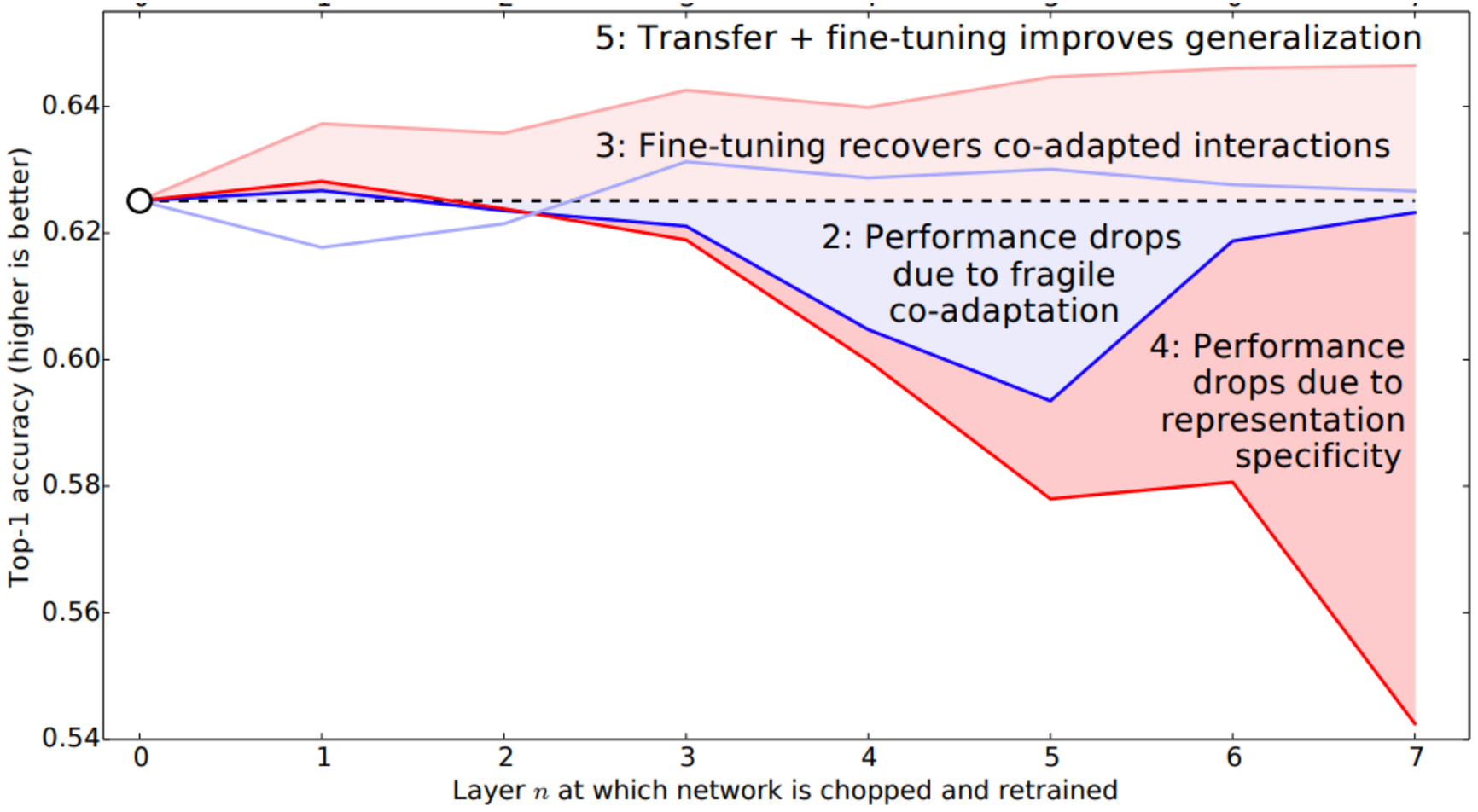


Larger dataset



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

Transfer Learning with CNNs



[Yosinski et al., NIPS 2014]

Model **Ensemble**

Training: Train multiple independent models

Test: Average their results

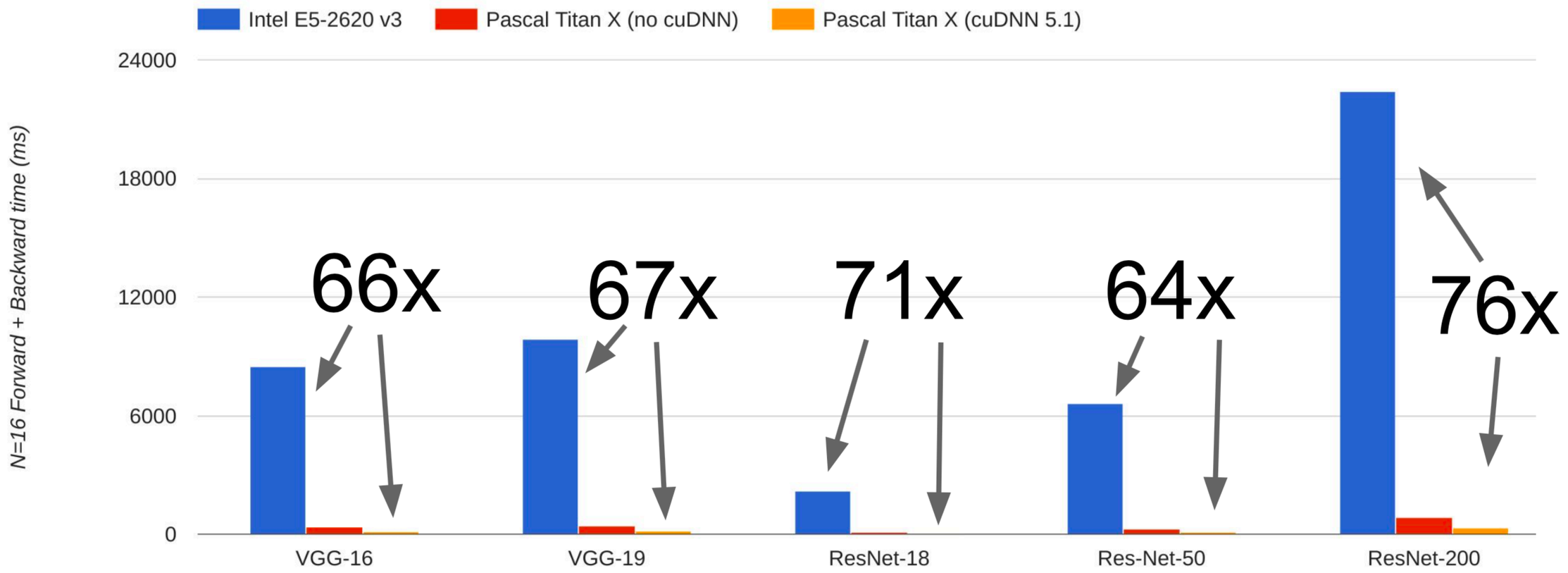
~ 2% improved performance in practice

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**

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.)

* 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

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

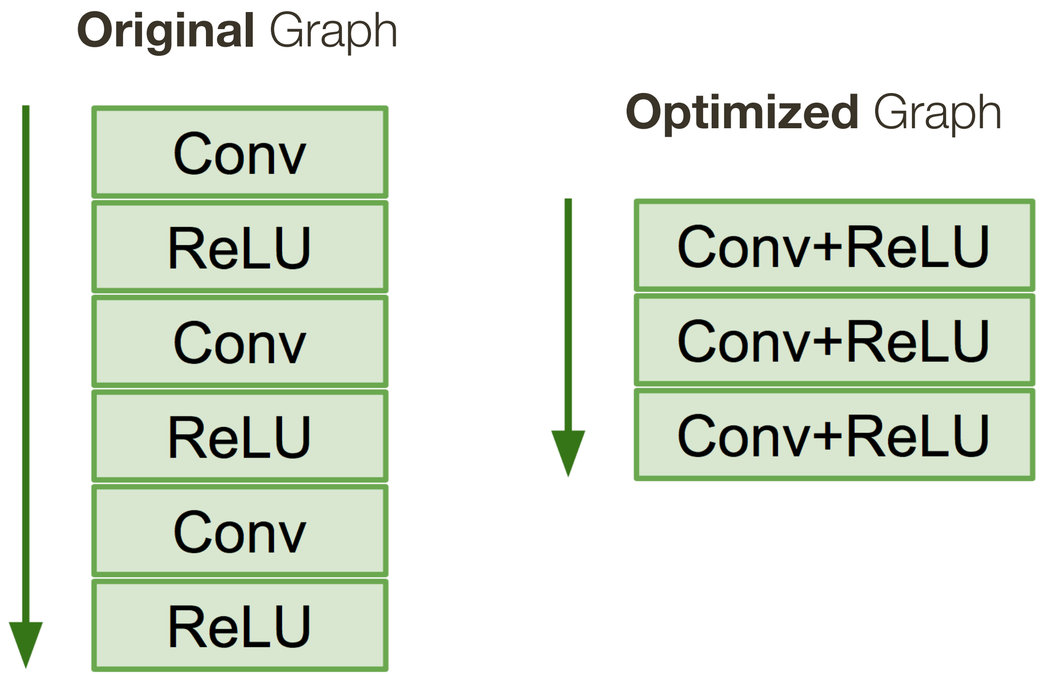
Frameworks: PyTorch vs. TensorFlow

Dynamic vs. Static computational graphs

Frameworks: PyTorch vs. TensorFlow

Dynamic vs. **Static** computational graphs

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

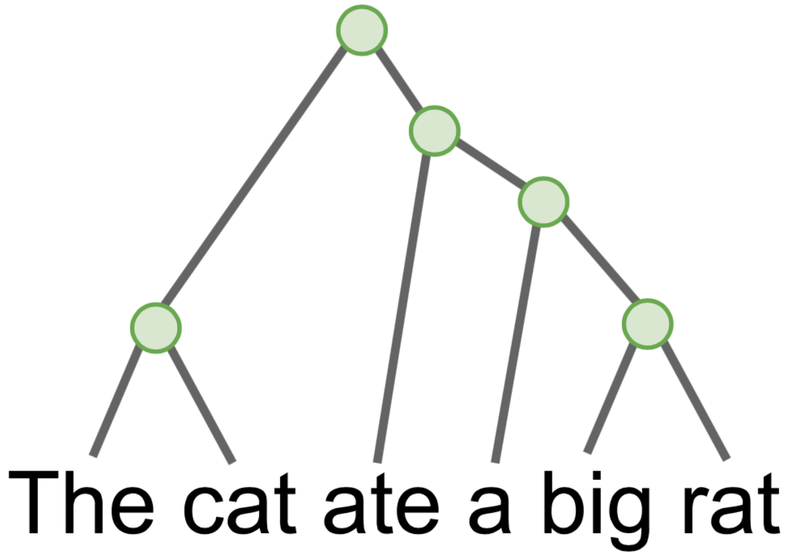


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Frameworks: PyTorch vs. TensorFlow

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

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

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

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)

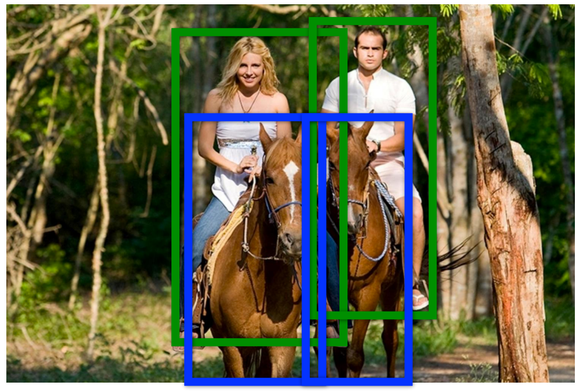
Categorization

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)

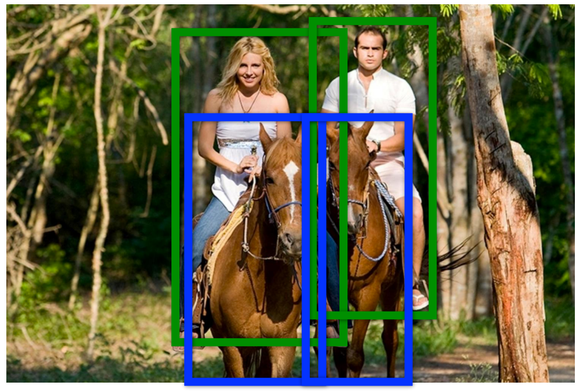
Categorization

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)

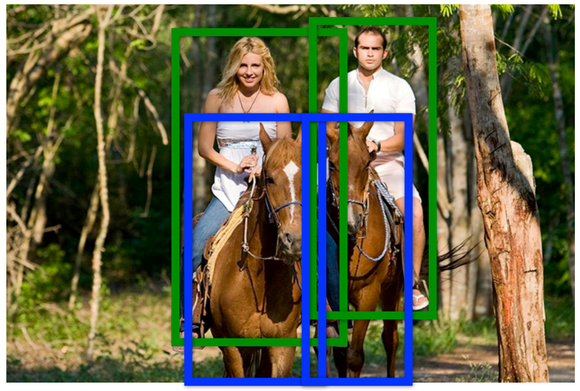
Categorization

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

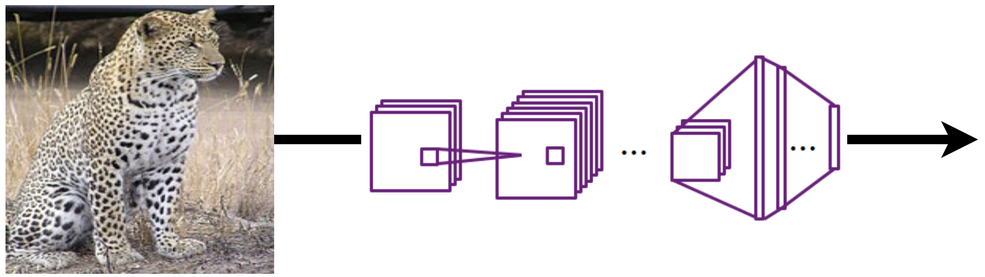
Horse1
Horse2
Person1
Person2

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

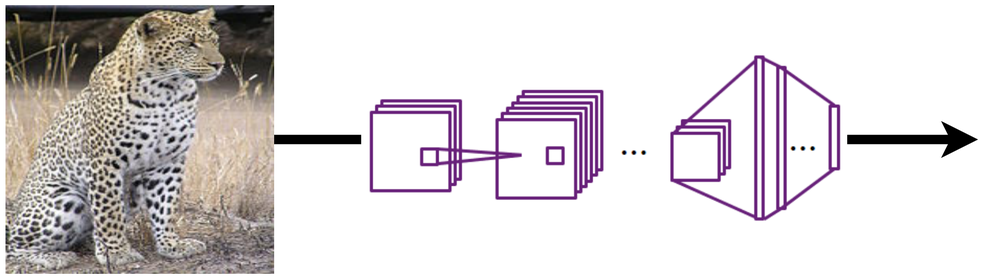
Object **Classification**

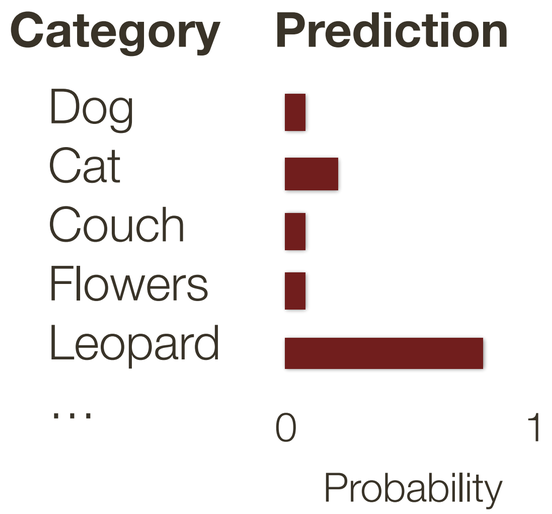


Category	Prediction
Dog	No
Cat	No
Couch	No
Flowers	No
Leopard	**Yes**
…	…

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

Object **Classification**

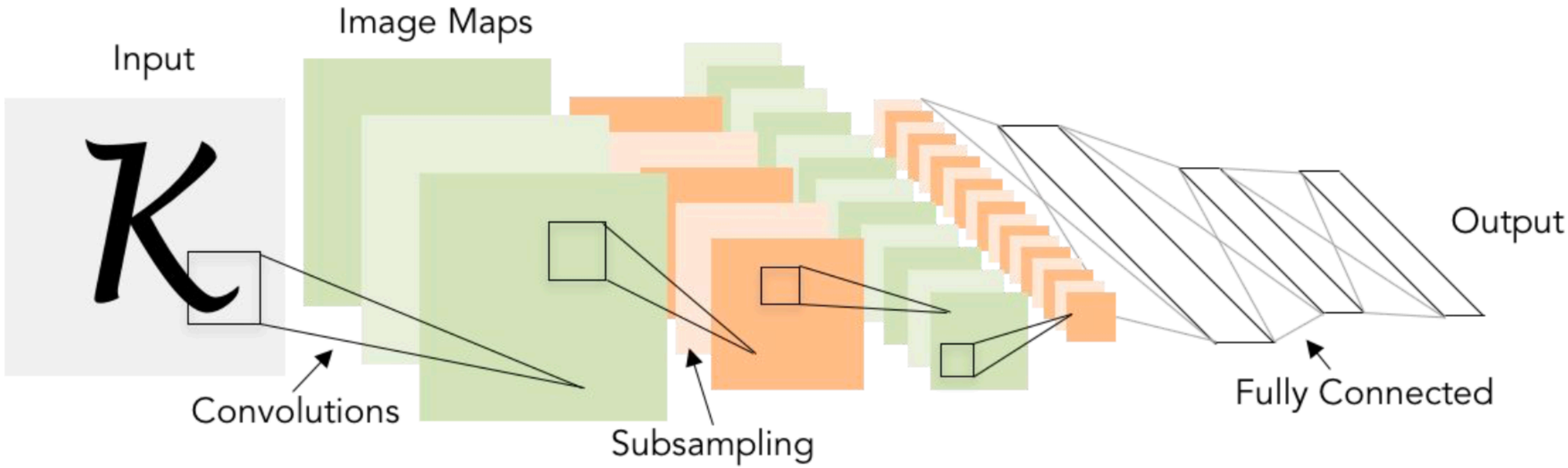




Problem: For each image predict which category it belongs to out of a fixed set

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

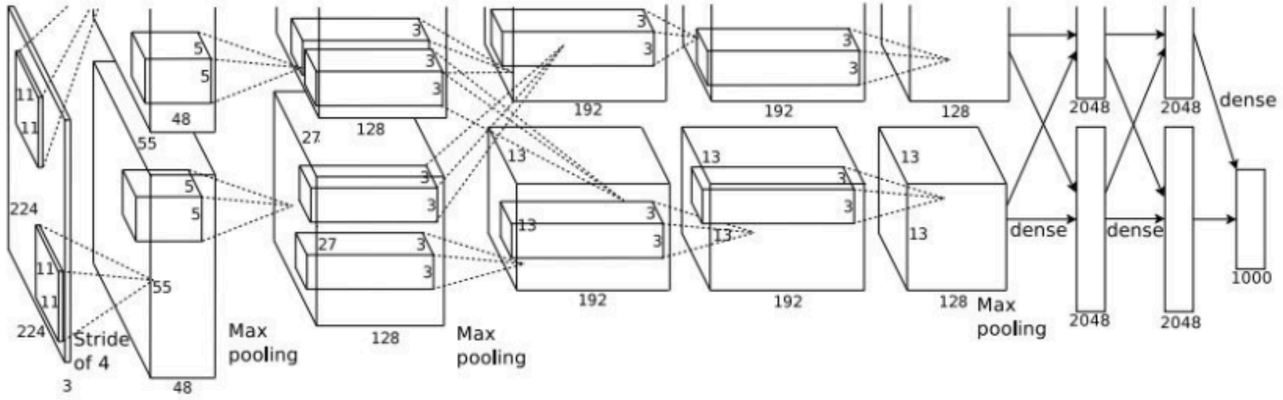
ImageNet **Competition** (ILSVRC)

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

CONV1: 96 11 x 11 filters applied at stride 4

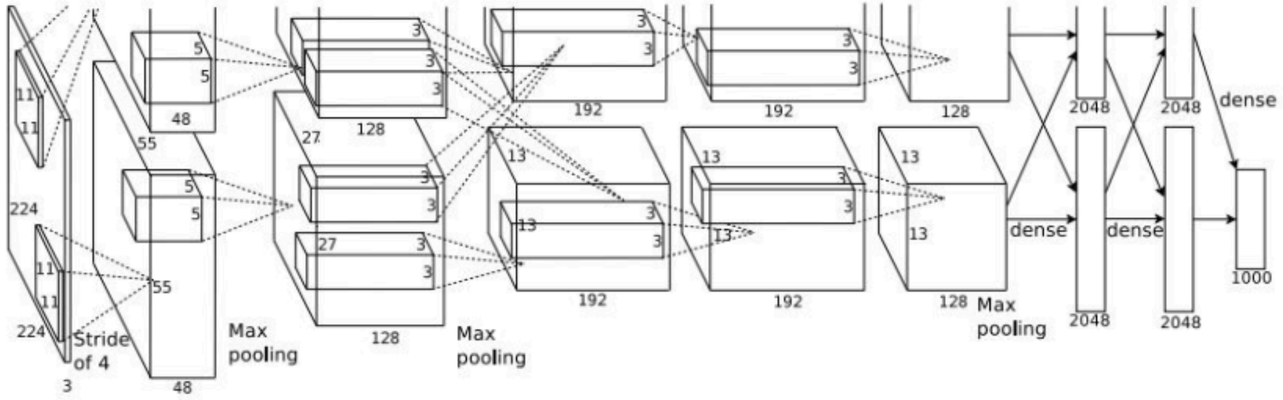
Output: 55 x 55 x 96 **Parameters:** 35K

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

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

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

AlexNet



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)

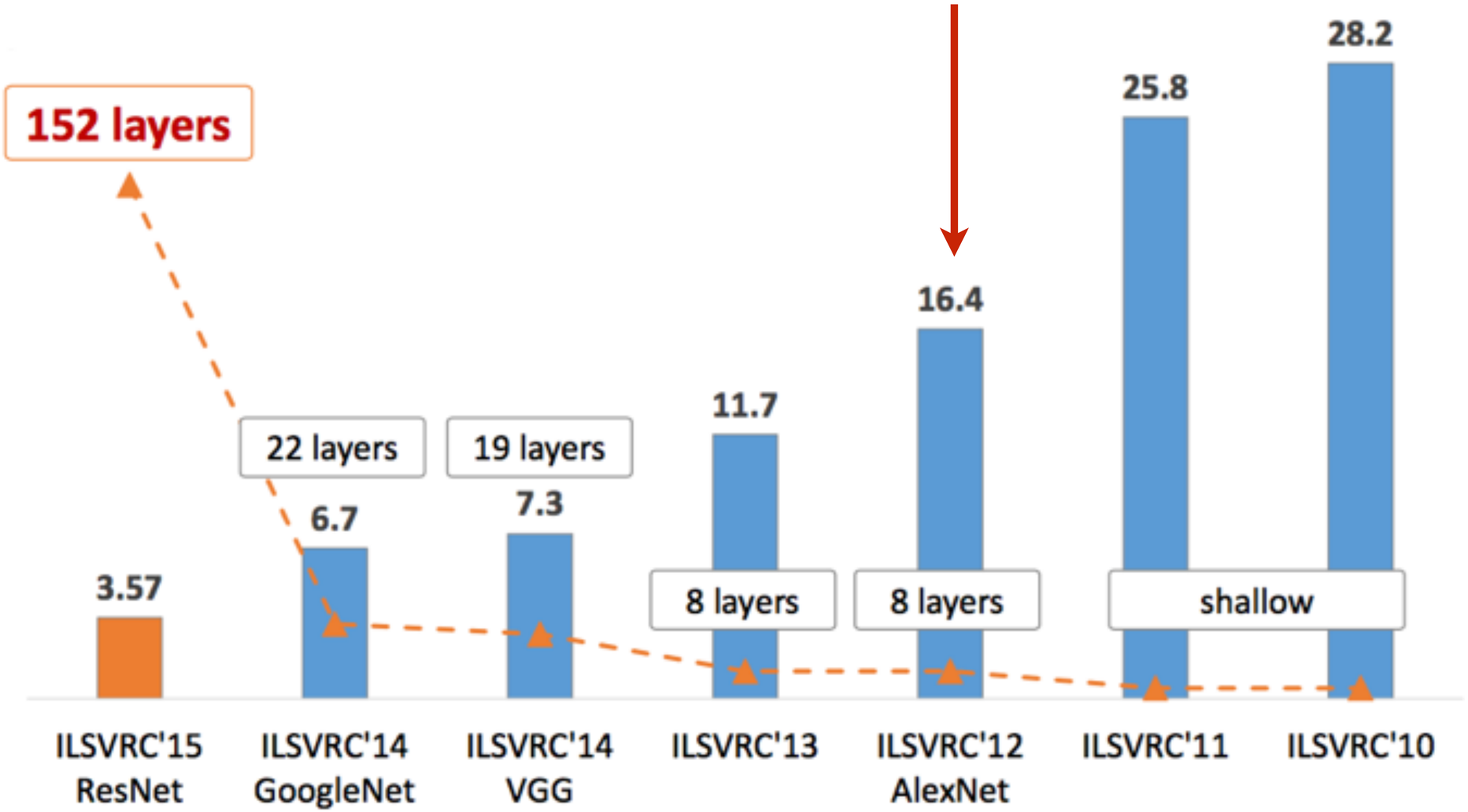
[Krizhevsky et al., 2012]

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%

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

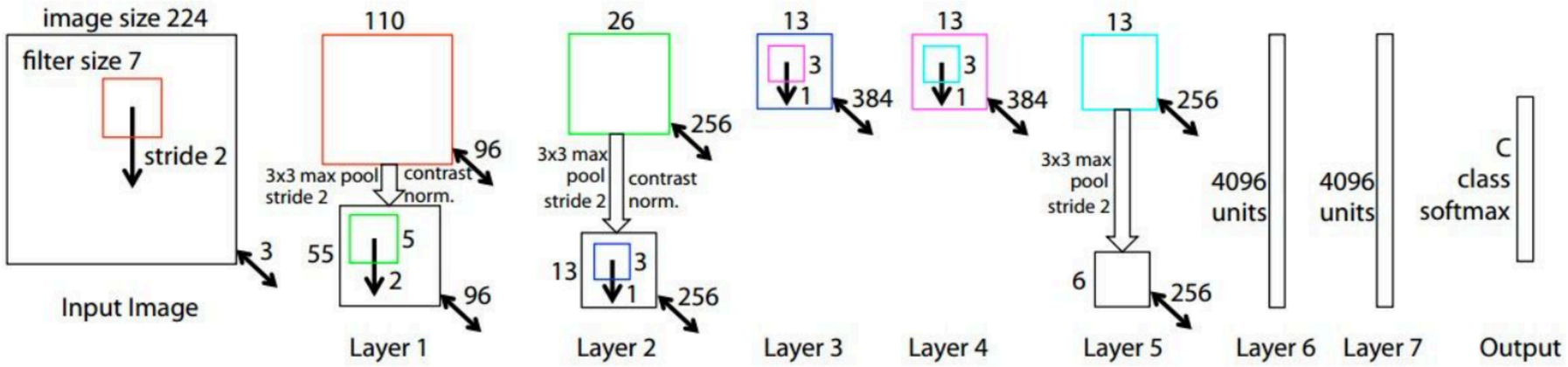
ILSVRC winner 2012



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

ZF Net

[Zeiler and Fergus, 2013]

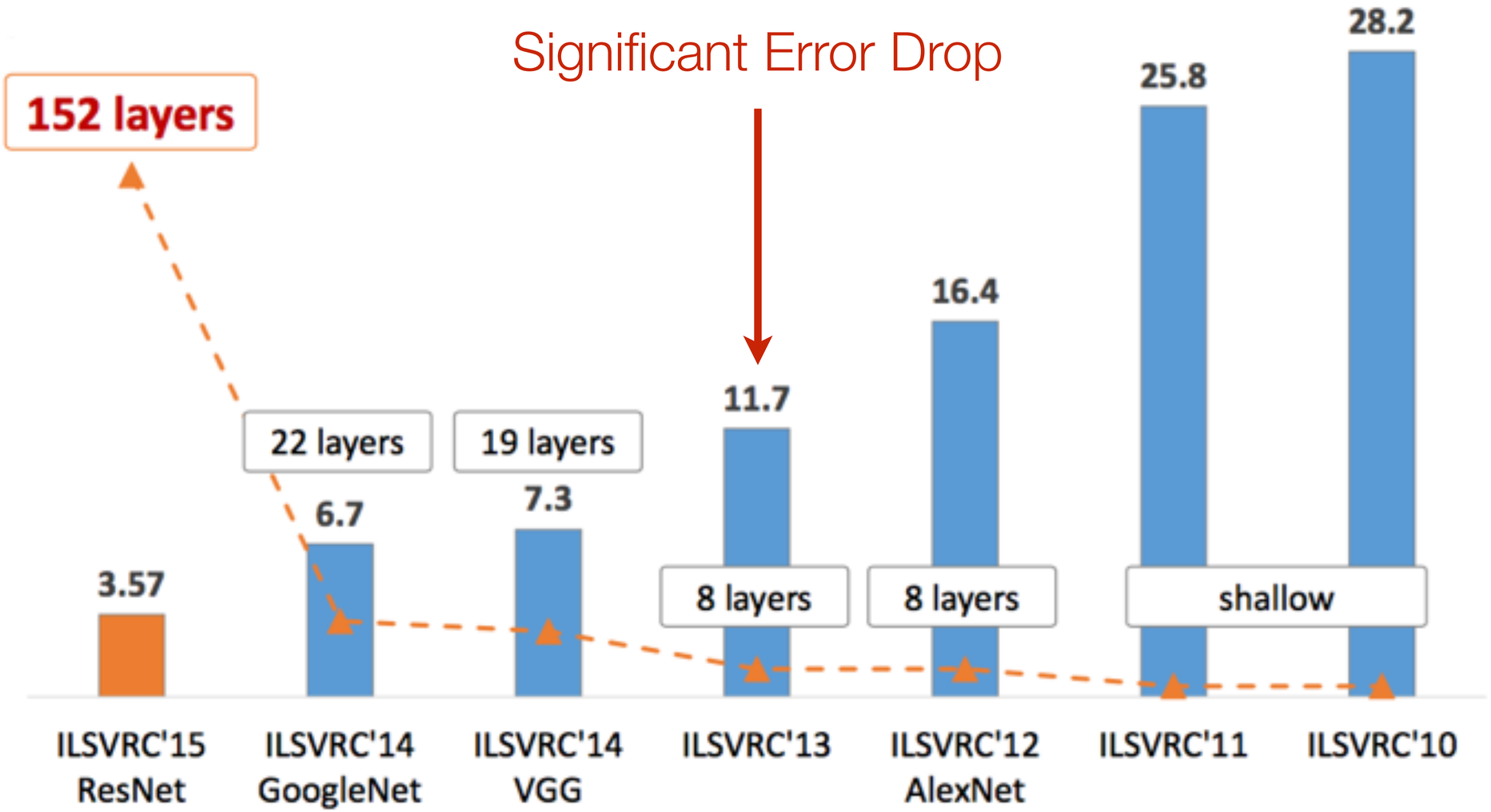


AlexNet with small modifications:

— 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

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

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)

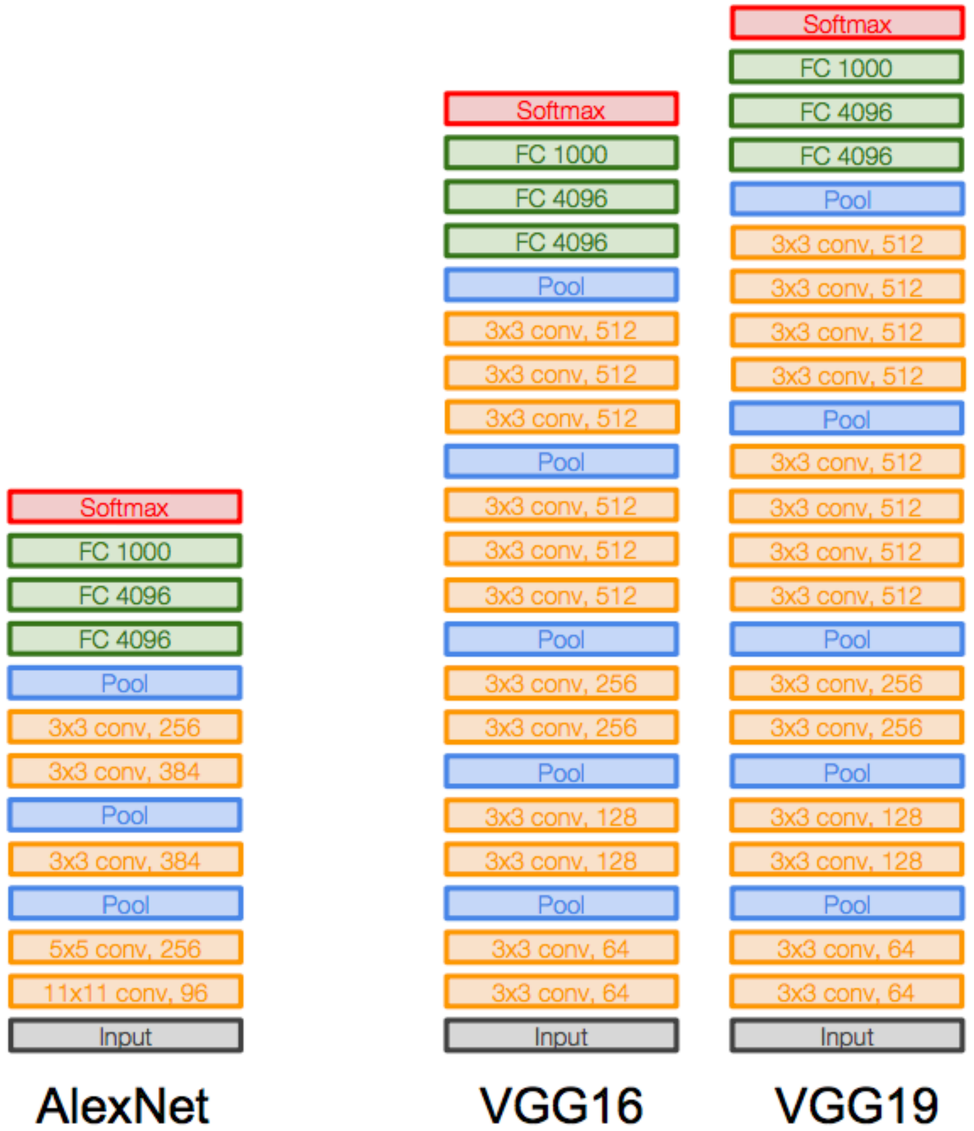
— deeper network (16 or 19 vs. 8 in AlexNet)

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**



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

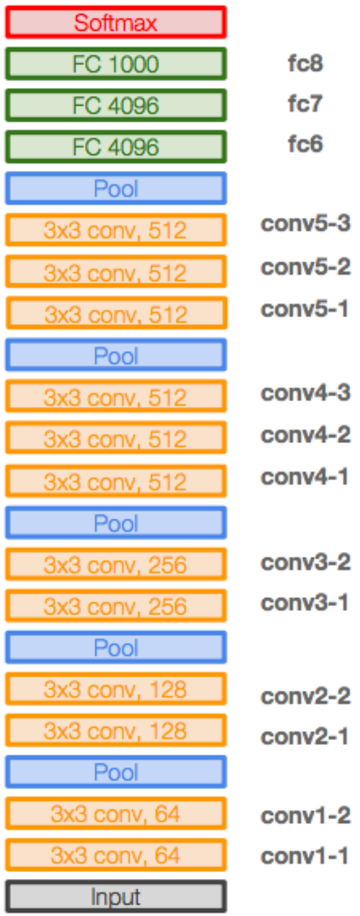
VGG Net

[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

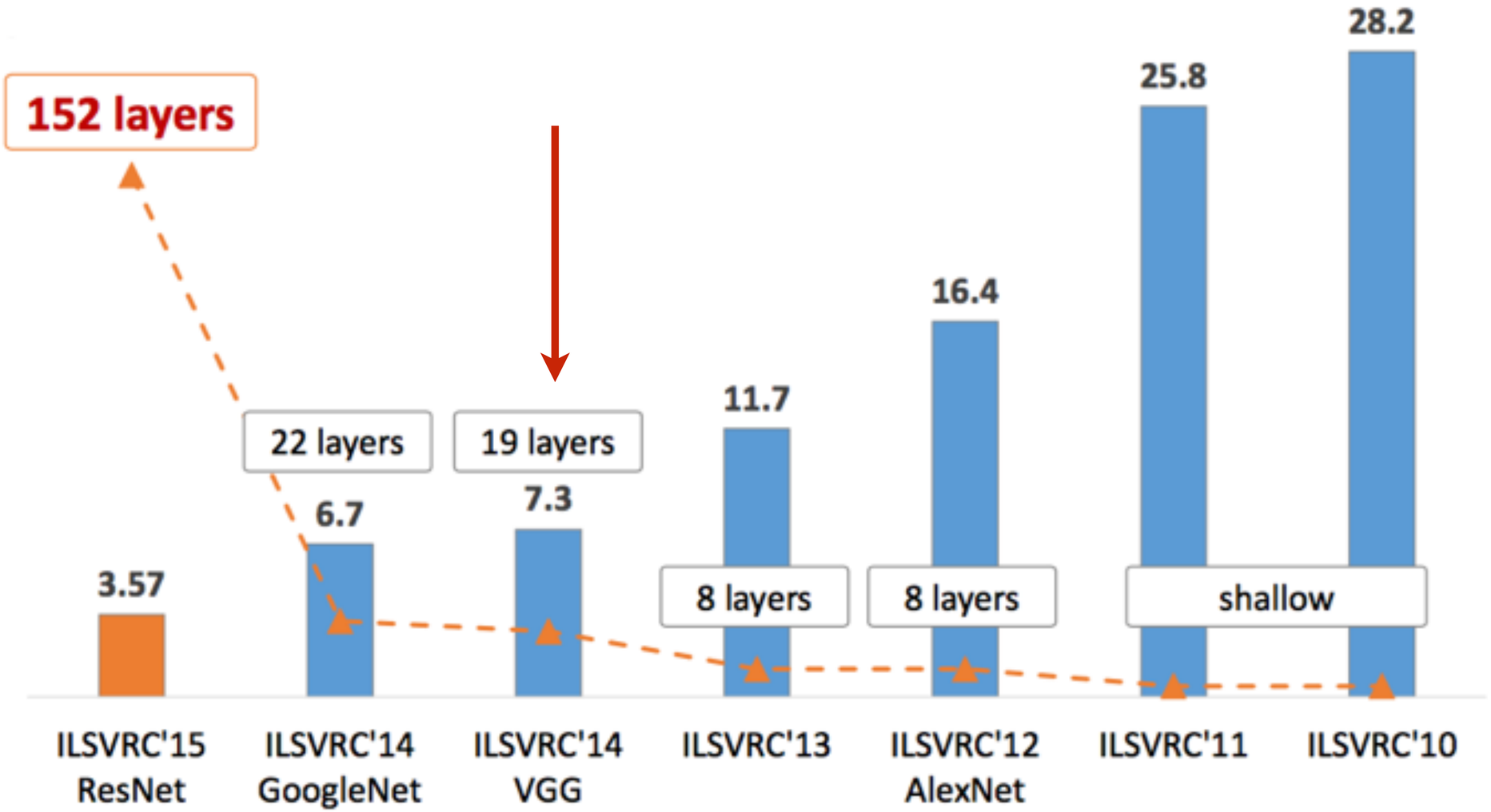
TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



VGG16

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

ILSVRC winner 2012



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

GoogleLeNet

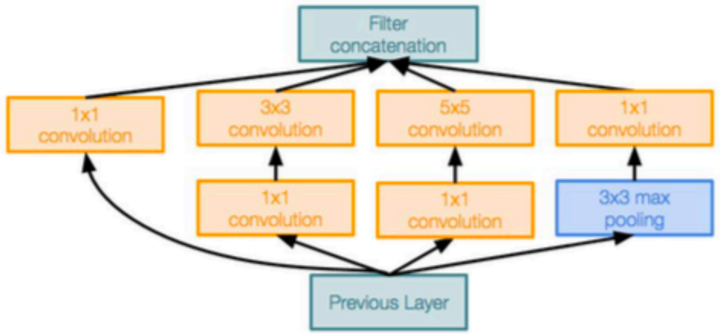
[Szegedy et al., 2014]

even deeper network with **computational efficiency**

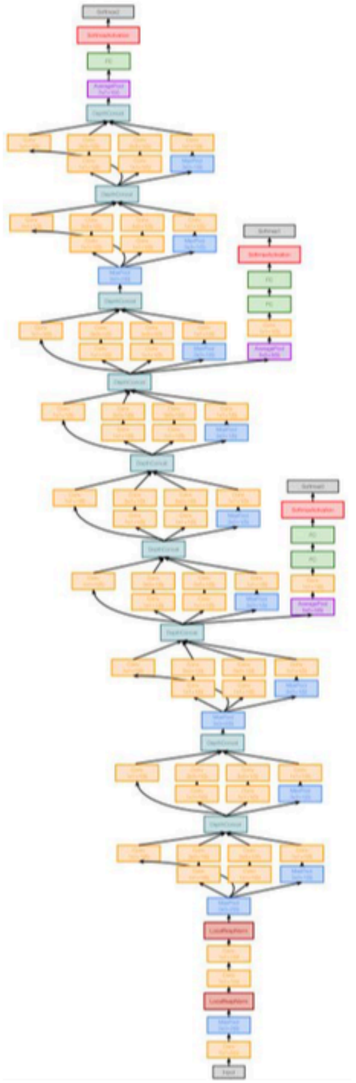
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!

(12x less than AlexNet!)

- Better performance (@6.7 top 5 error)



Inception module

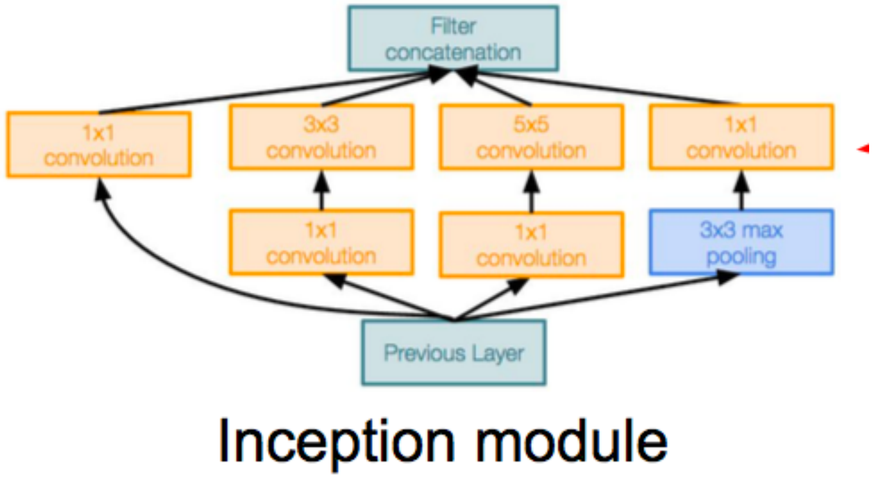


* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

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



Inception module

* 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

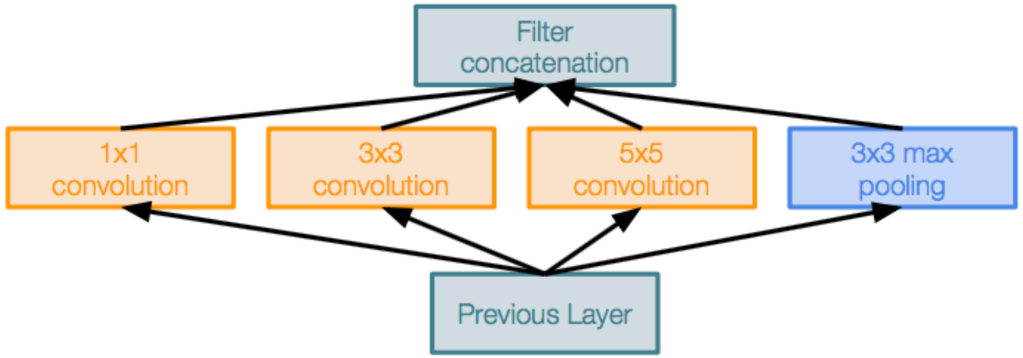
Apply **parallel filter operations** on the input from previous layer

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

— Pooling operation (3x3)

Concatenate all filter outputs together at output depth-wise

What’s the problem?



Naive Inception module

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

GoogleLeNet: Inception Module

[Szegedy et al., 2014]

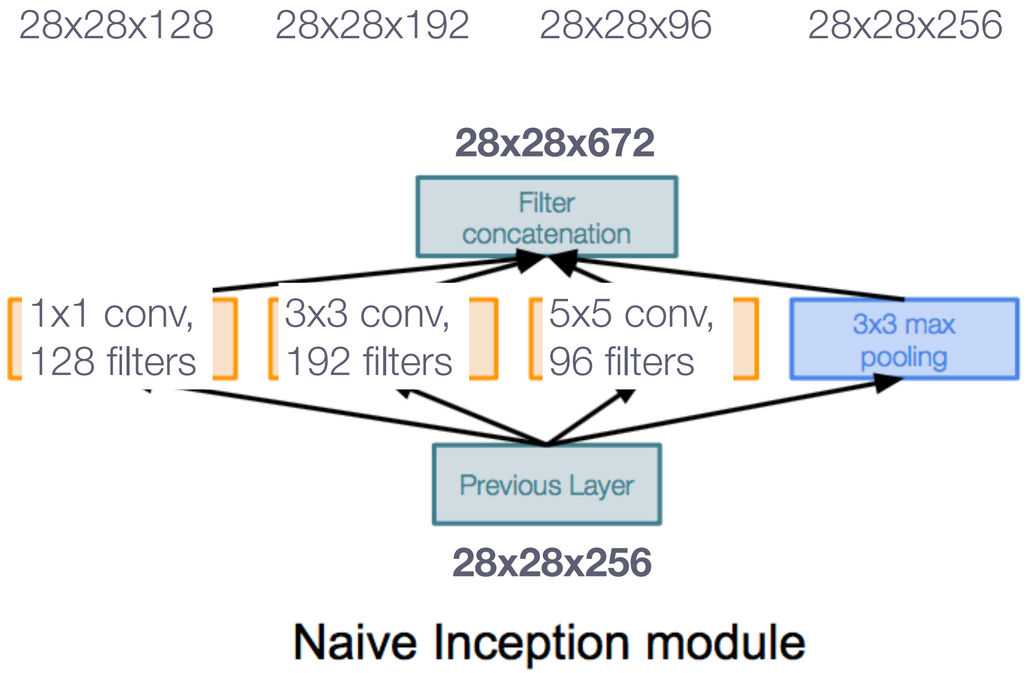
Idea: design good local topology (“network within network”) and then stack these modules

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

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

— Pooling operation (3x3)

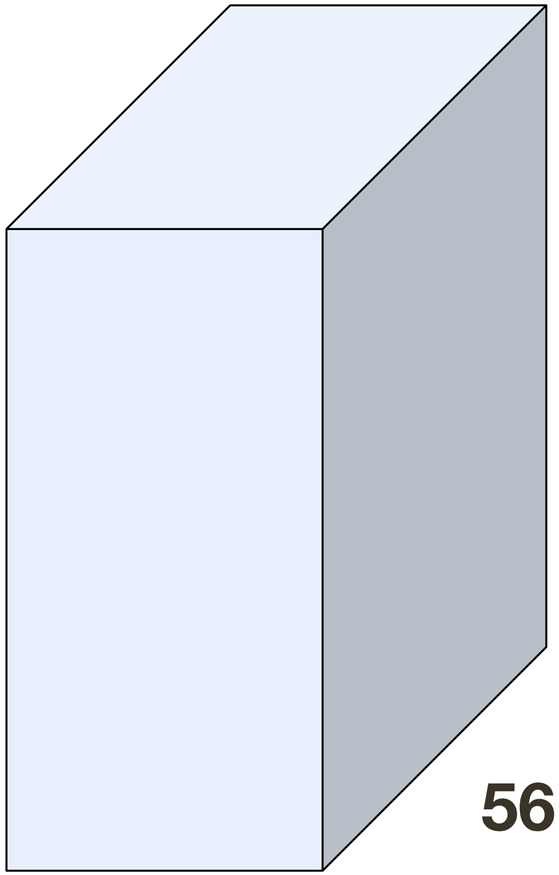
Concatenate all filter outputs together at output depth-wise



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

Convolutional Layer: **1x1 convolutions**

56 x 56 x 64 **image**



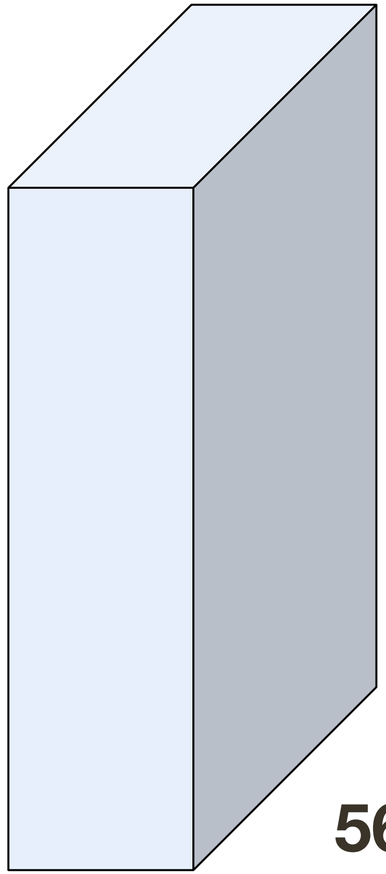
56 height

56 width

64 depth

32 **filters** of size, 1 x 1 x 64

56 x 56 x 32 **image**



56 height

56 width

32 depth

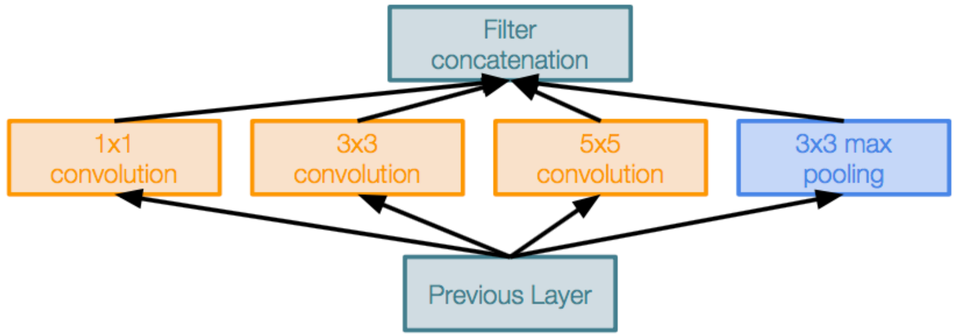
* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

GoogleLeNet: Inception Module

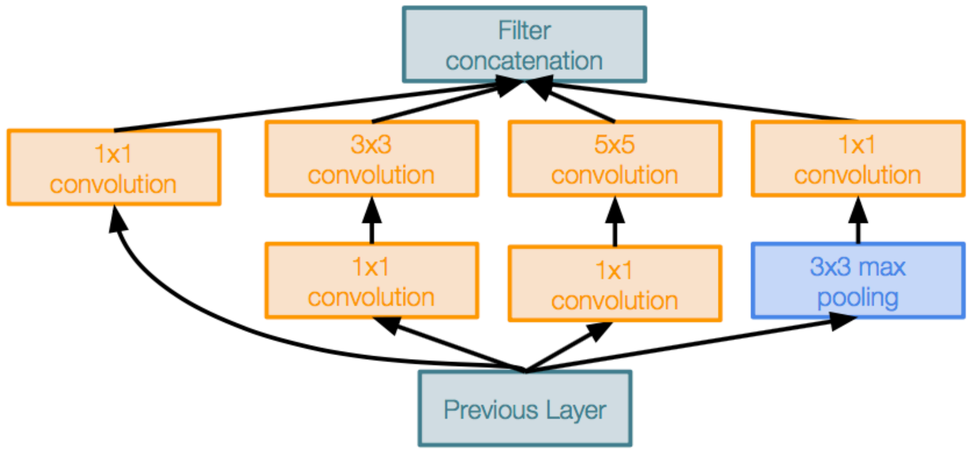
[Szegedy et al., 2014]

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

1x1 “bottleneck” layers



Naive Inception module



Inception module with dimension reduction

saves approximately 60% of computations

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

GoogleLeNet

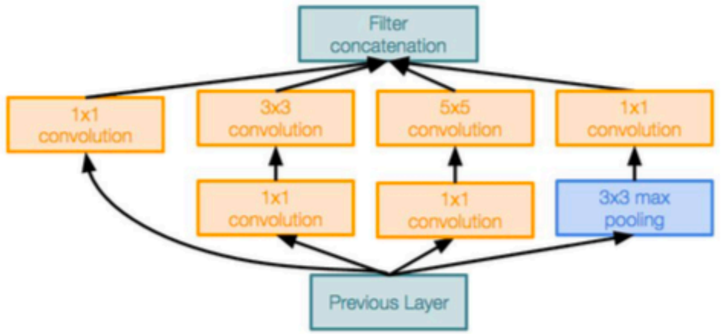
[Szegedy et al., 2014]

even deeper network with **computational efficiency**

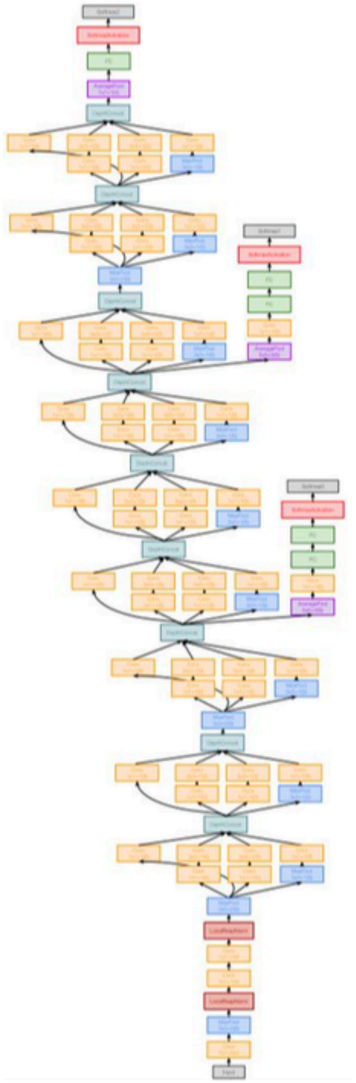
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!

(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

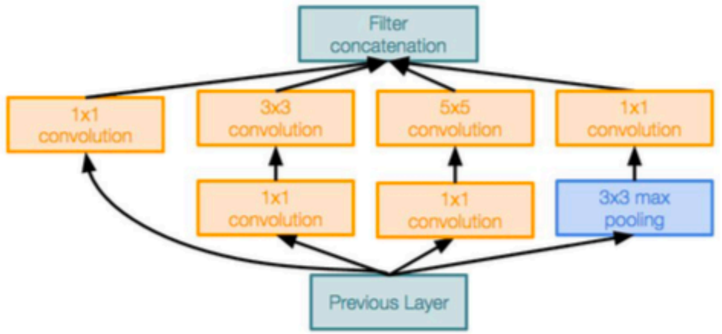
[Szegedy et al., 2014]

even deeper network with **computational efficiency**

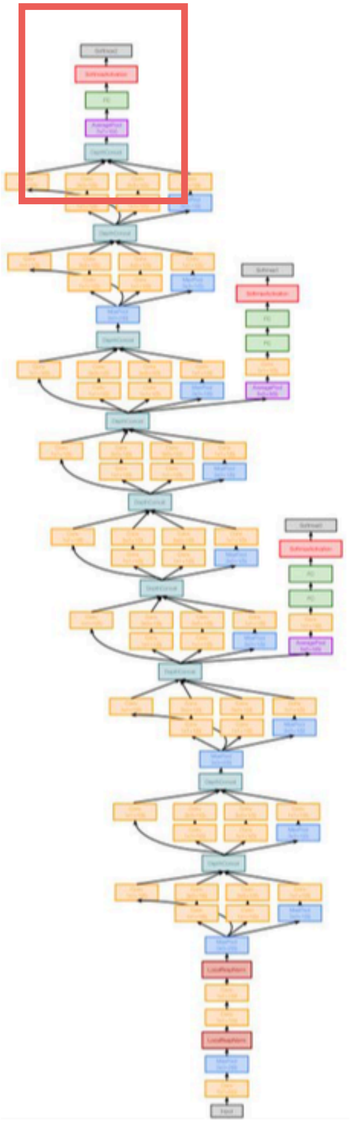
- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!

(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

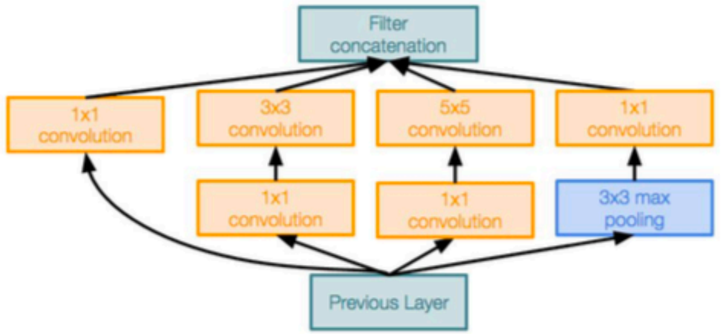
— Efficient “Inception” module

— No FC layers

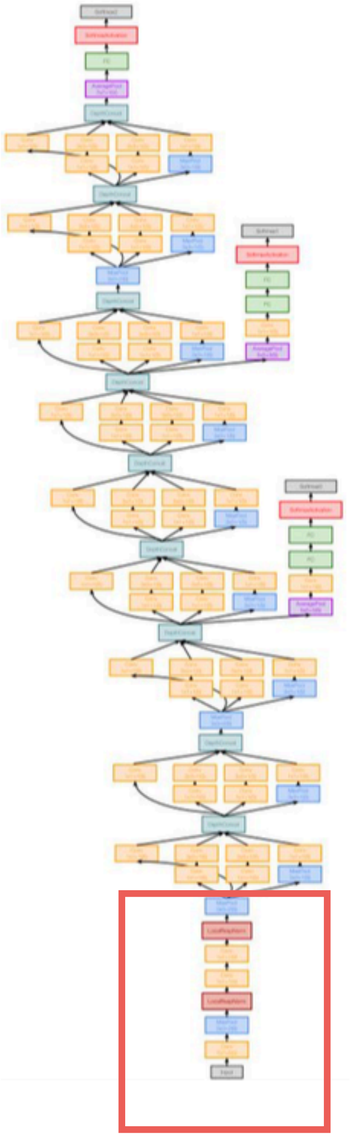
— Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)



Inception module



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

GoogleLeNet

[Szegedy et al., 2014]

even deeper network with **computational efficiency**

— 22 layers

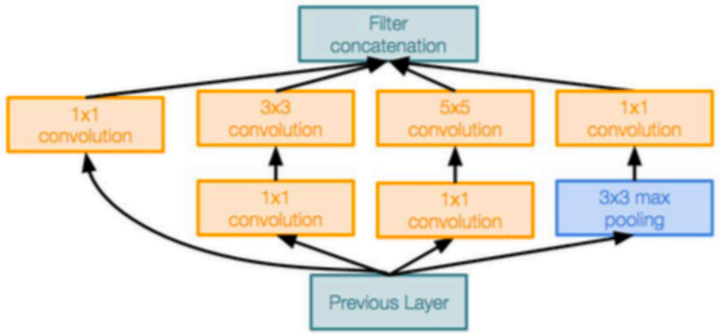
— Efficient “Inception” module

— No FC layers

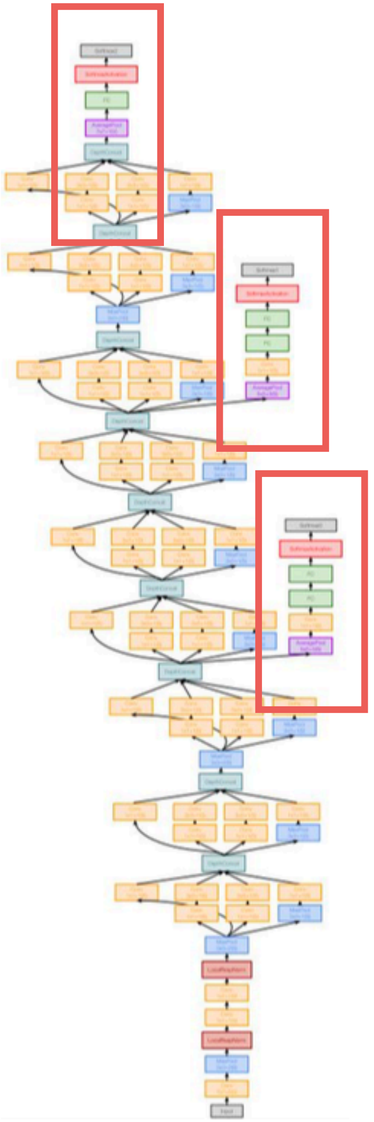
— Only 5 million parameters!

(12x less than AlexNet!)

— Better performance (@6.7 top 5 error)

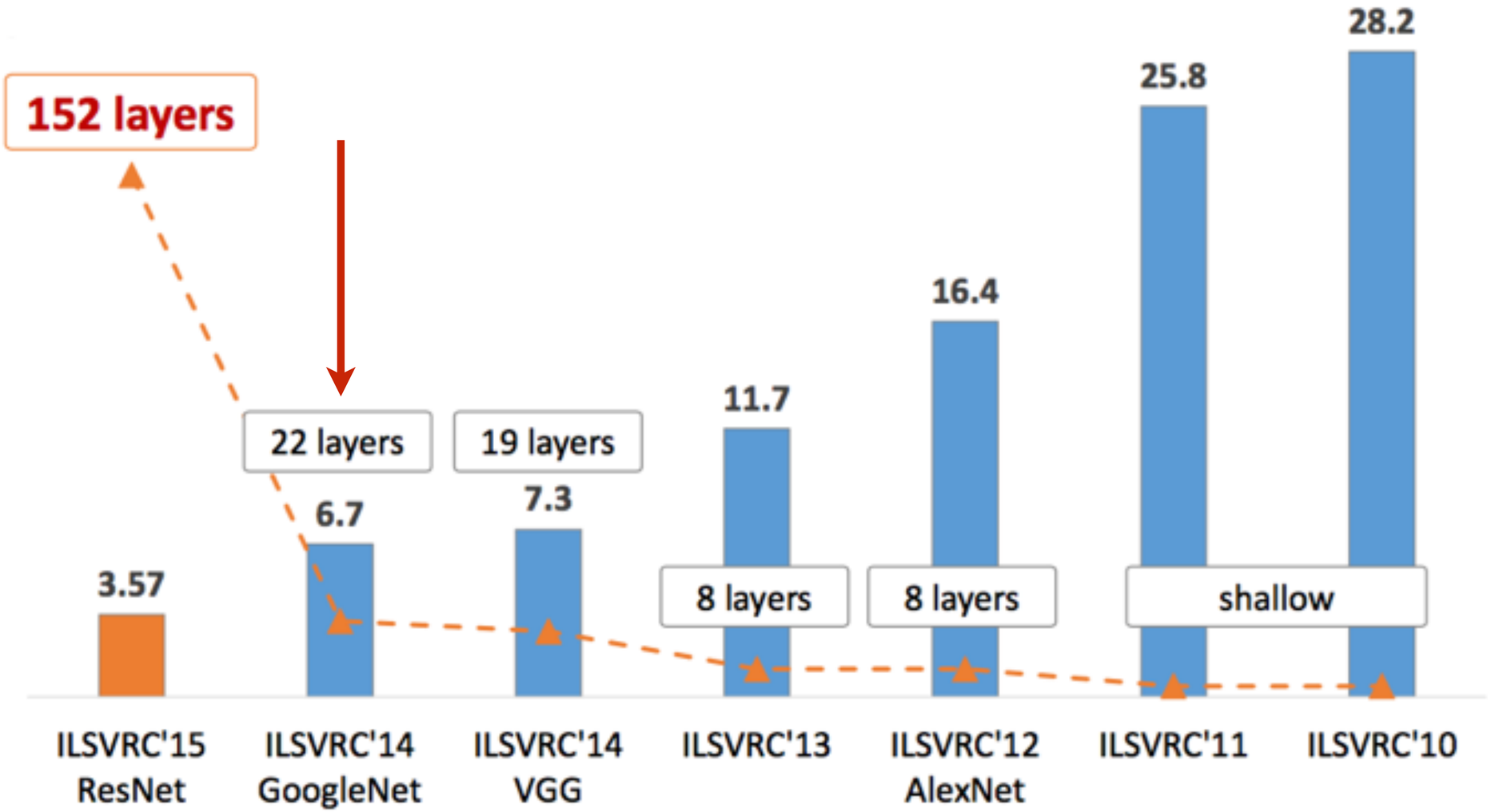


Inception module



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

ILSVRC winner 2012



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