Deep Learning (for Computer Vision)

Arjun Jain | 7 April 2017

Sources

A lot of the material has been shamelessly and gratefully collected from:

- http://cs231n.stanford.edu/
- http://neuralnetworksanddeeplearning.com/chap4.html

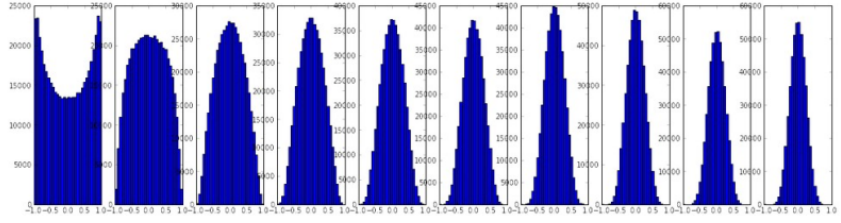
Recap

- Building blocks
  - Linear
  - Convolution
  - Max Pooling
  - Cross Entropy
  - Dropout
  - Batch Normalization
- Weight initializations, data preprocessing
- Choosing hyper-parameters, baby sitting the learning process
- Applications of ConvNets

$$\frac{\partial L}{\partial W} = \text{Correlation}(\text{I}, \text{LO}) \qquad \frac{\partial L}{\partial I} = \text{Correlation}(W_{pad}, \text{LO}_{flip})$$

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}$$



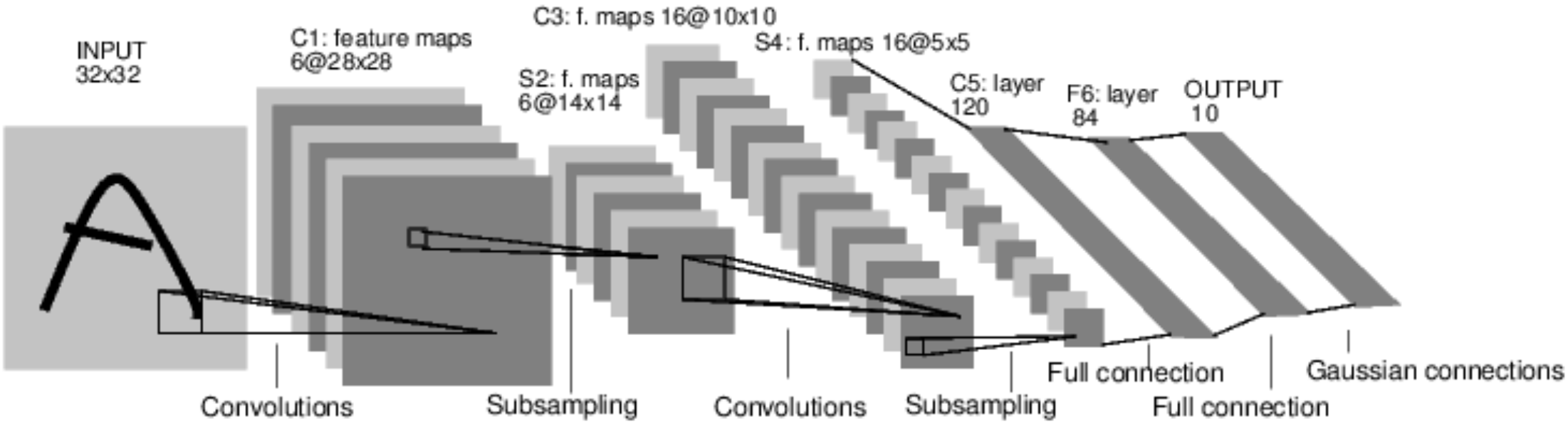
Agenda this Week

- CNN Case Studies
  - LeNet
  - AlexNet
  - …
- Myth buster: You need a lot of data to train ConvNets (false!)
- Visualizing and understanding ConvNets
- CNN Compression

A Few CNN Case Studies

Case Study: LeNet-5

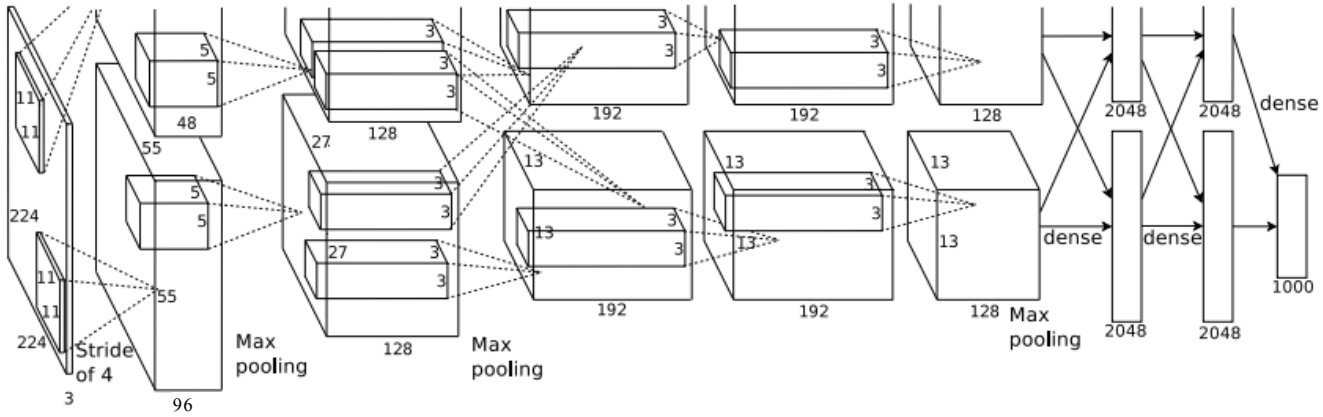
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

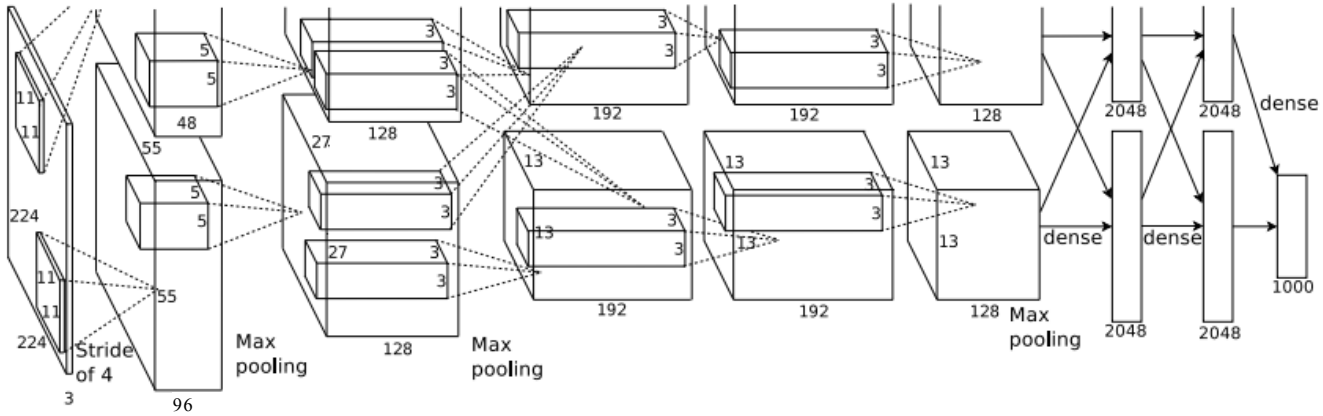
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

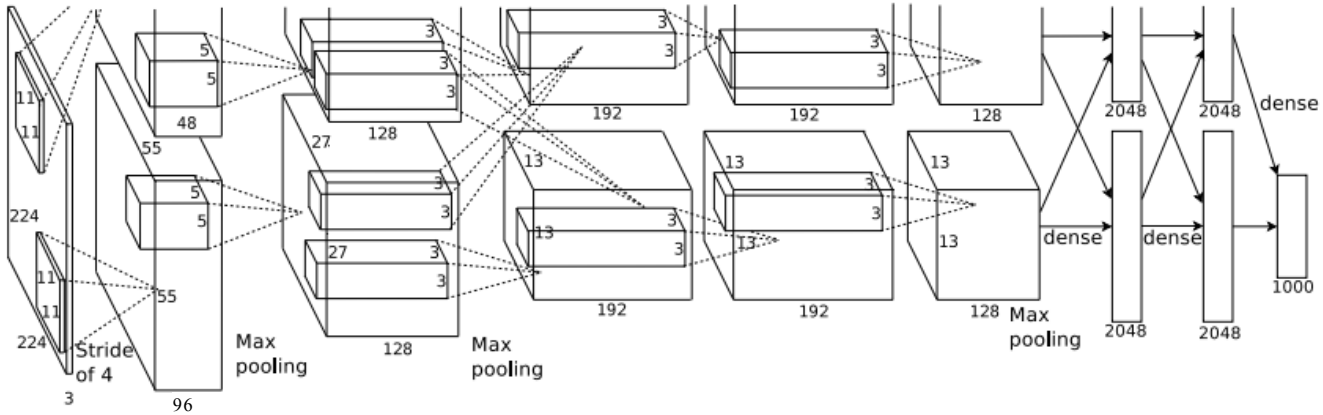
=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

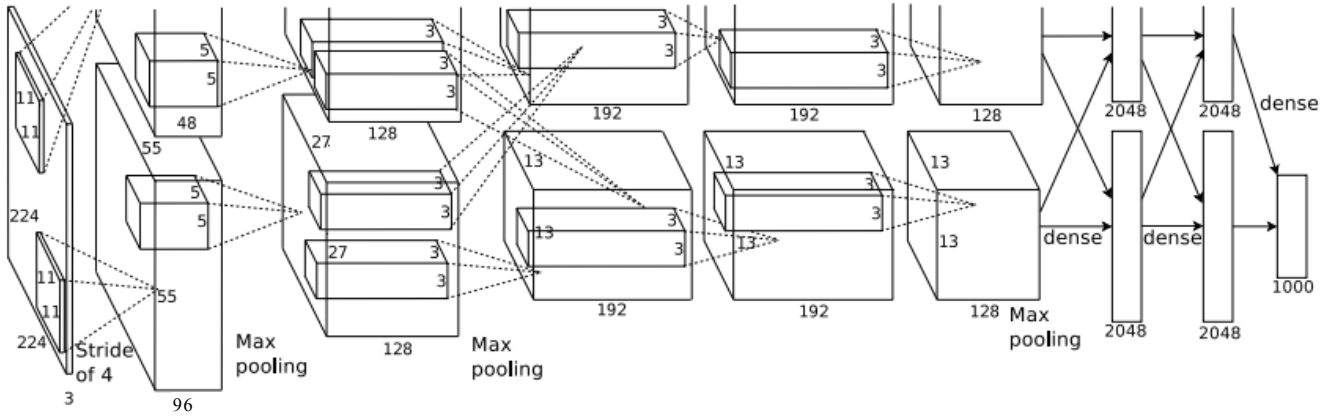


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

[Krizhevsky et al. 2012]



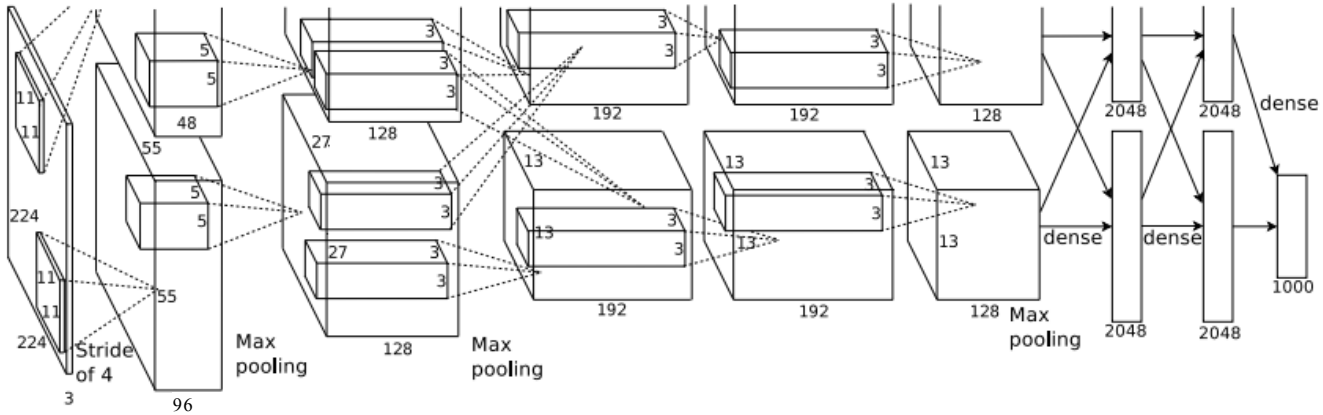
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

Case Study: AlexNet

[Krizhevsky et al. 2012]



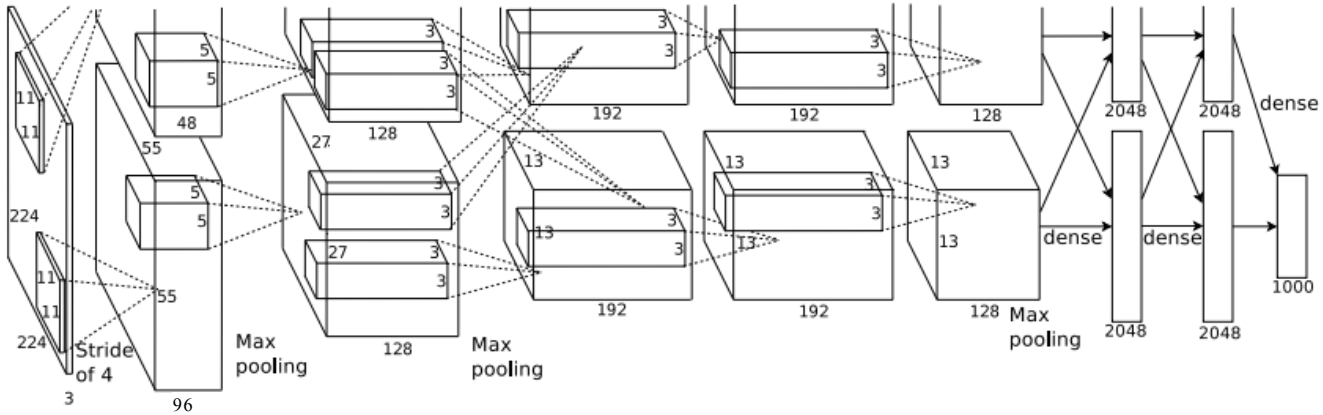
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]

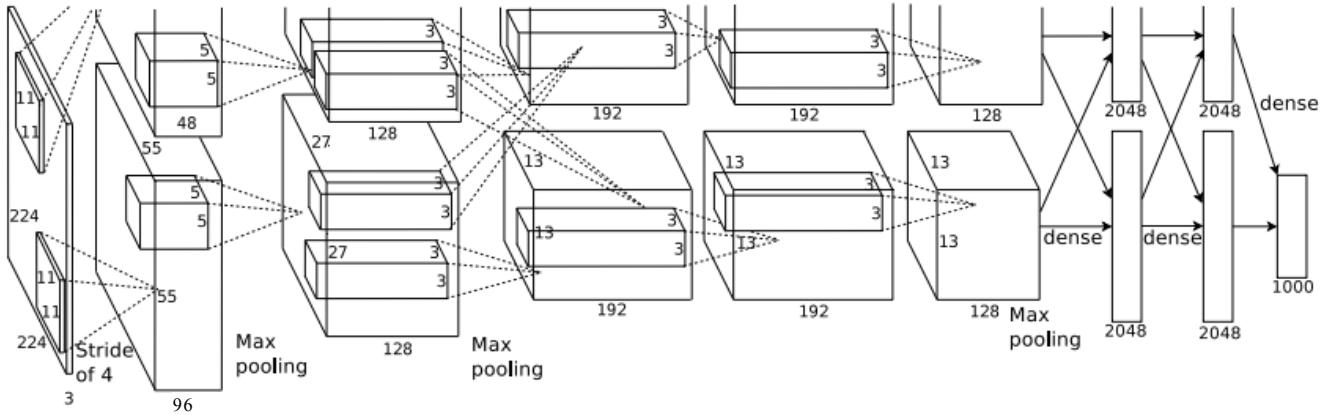


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

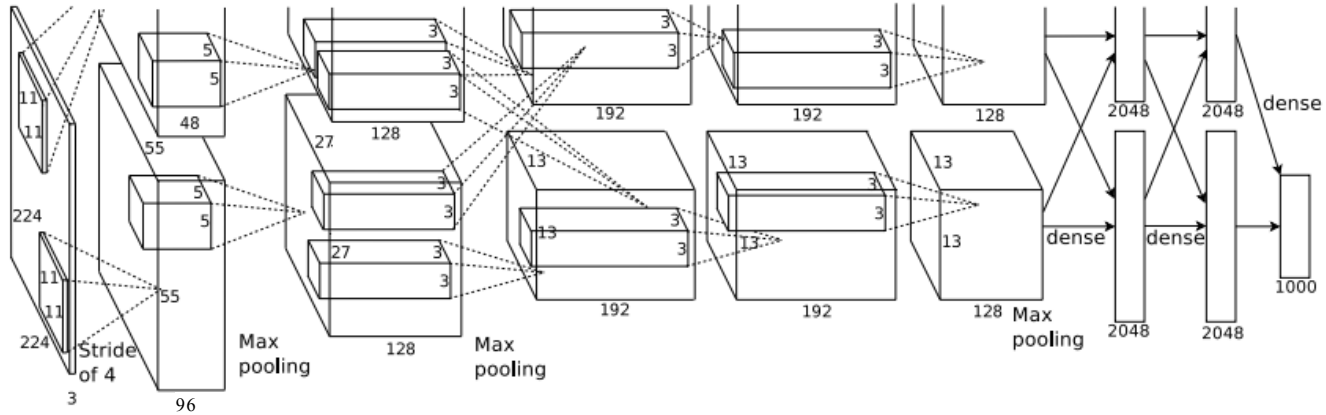
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
…

Case Study: AlexNet

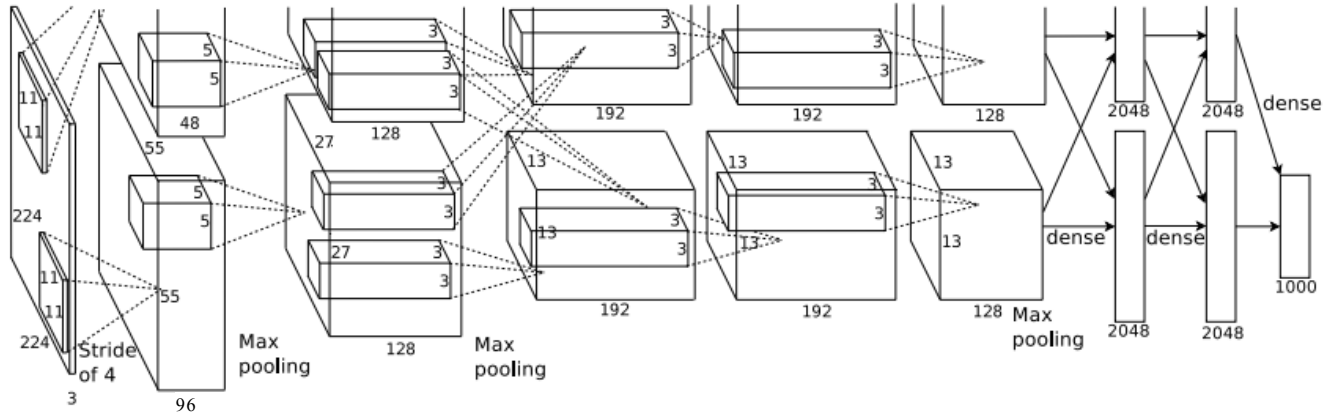
[Krizhevsky et al. 2012]



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)

Case Study: AlexNet

[Krizhevsky et al. 2012]

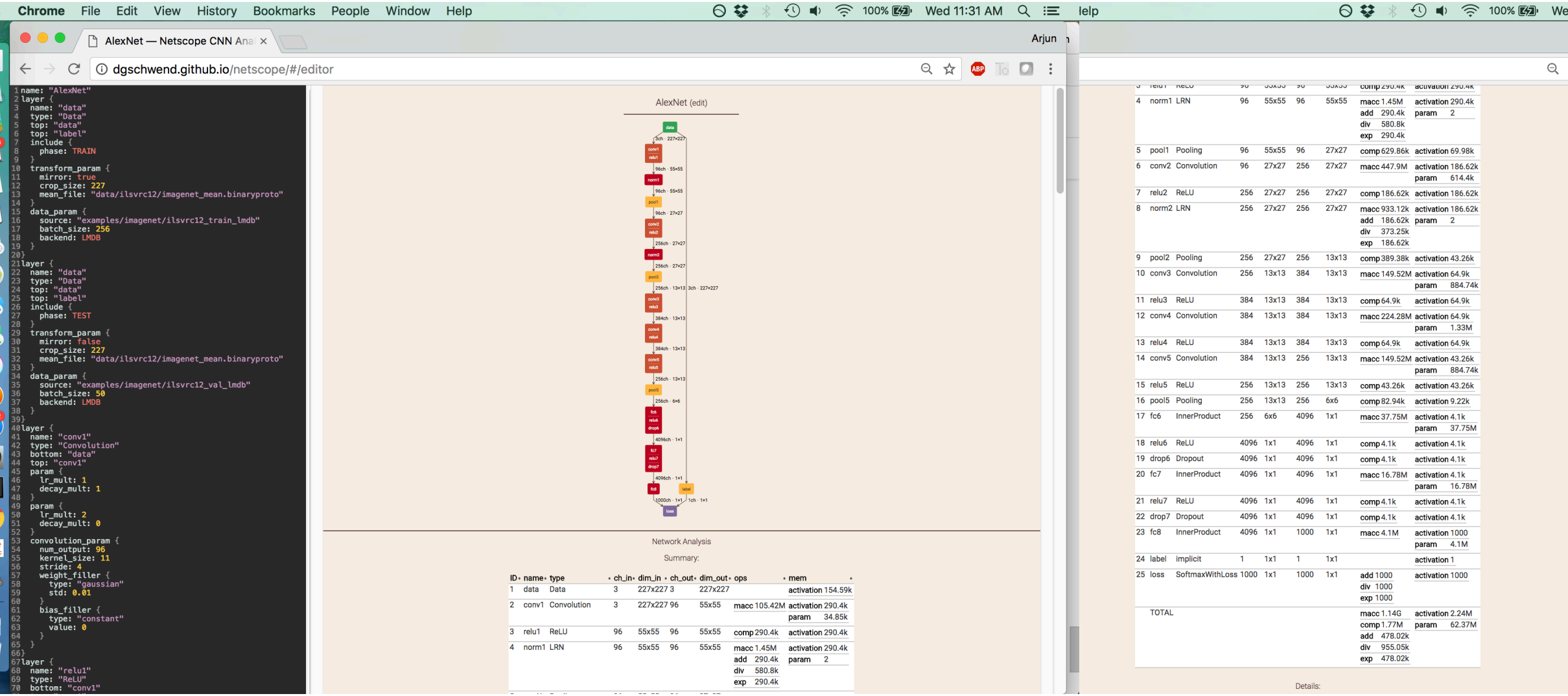


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)

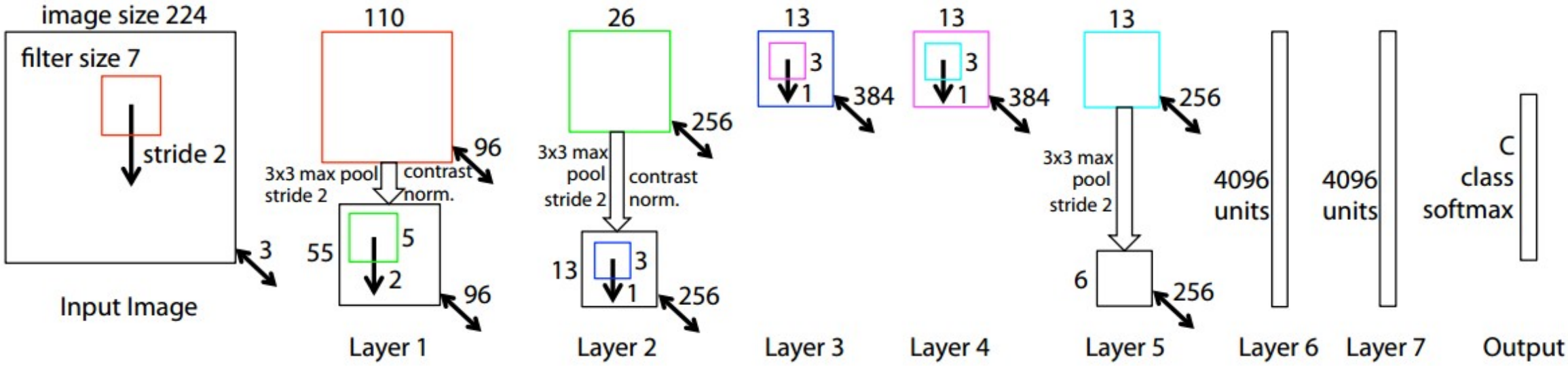
Details/Retrospectives:

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

Aside: Useful Tool http://dgschwend.github.io/netscope/#/editor



Case Study: ZFNet *[Zeiler and Fergus, 2013]*



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

ImageNet top 5 error: 15.4% -> 14.8 (and later 11.2)%

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error (did not win!)

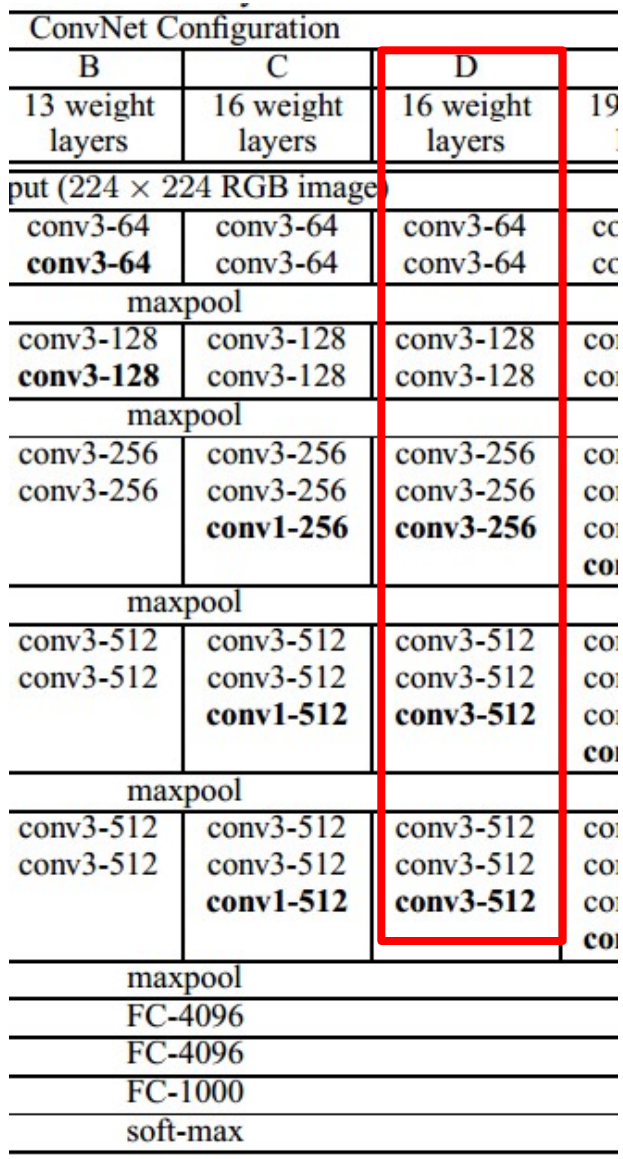
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128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

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

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

(params not counting biases)



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

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

(params not counting biases)

TOTAL memory: 24M * 4 bytes ~= 93MB / image
(only forward! Backward?)
TOTAL params: 138M parameters

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

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

(params not counting biases)

Note:

Most memory is in early CONV

Most params are in late FC

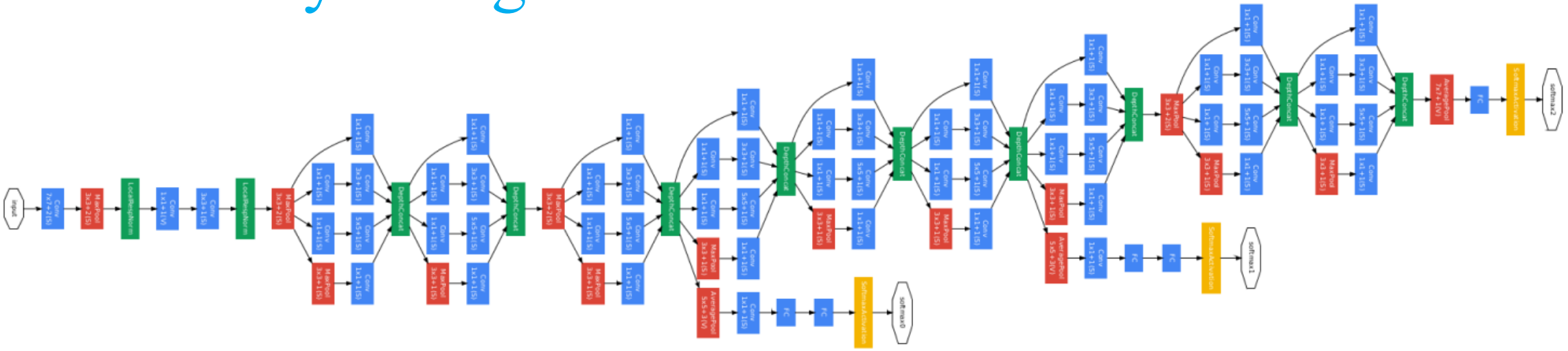
(Avg. Pool)

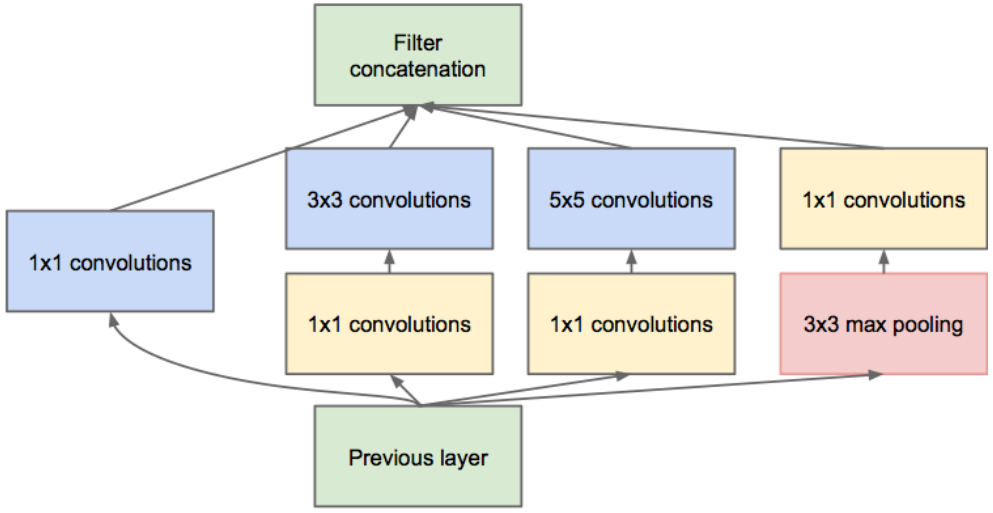
TOTAL memory: 24M * 4 bytes ~= 93MB / image

(only forward! ~*2 for bwd)

TOTAL params: 138M parameters

Case Study: GoogLeNet *[Szegedy et al., 2014]*





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Case Study: GoogLeNet *[Szegedy et al., 2014]*

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Fun features:

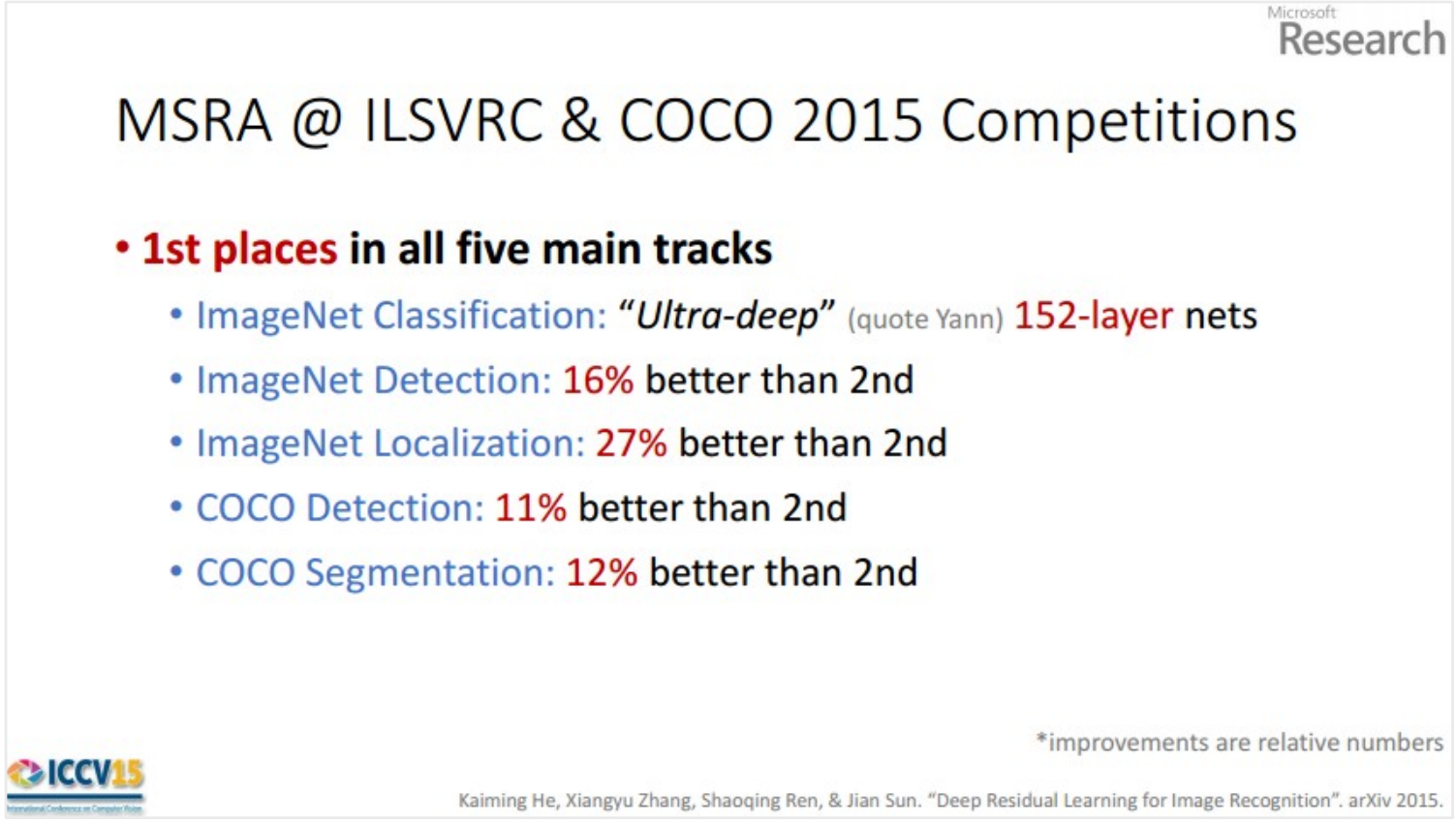
- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

Case Study: ResNet

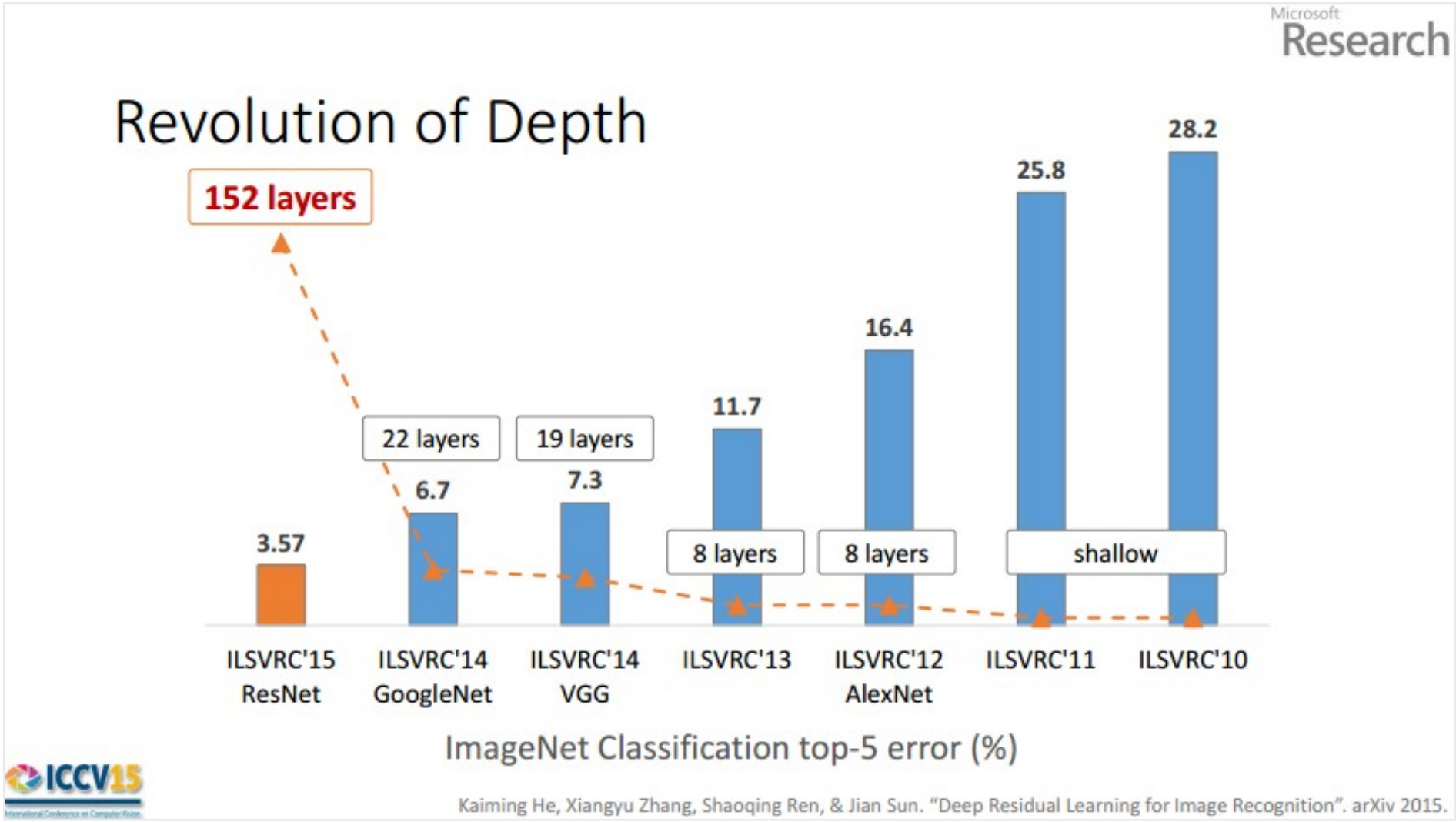
[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



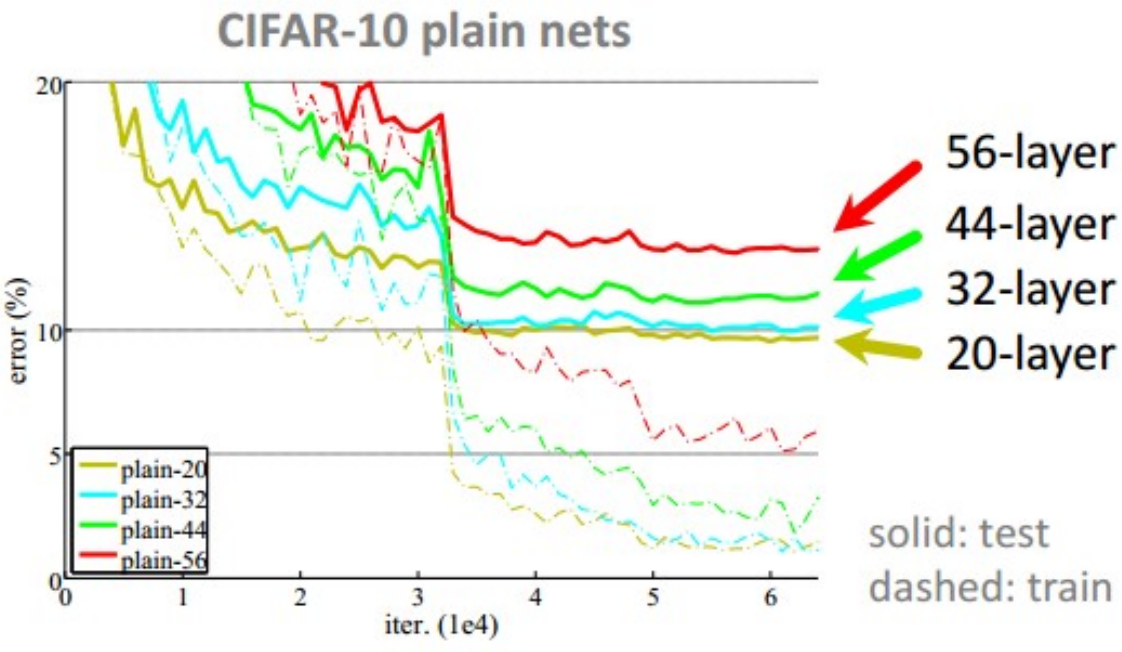
Slide from Kaiming He's recent presentation

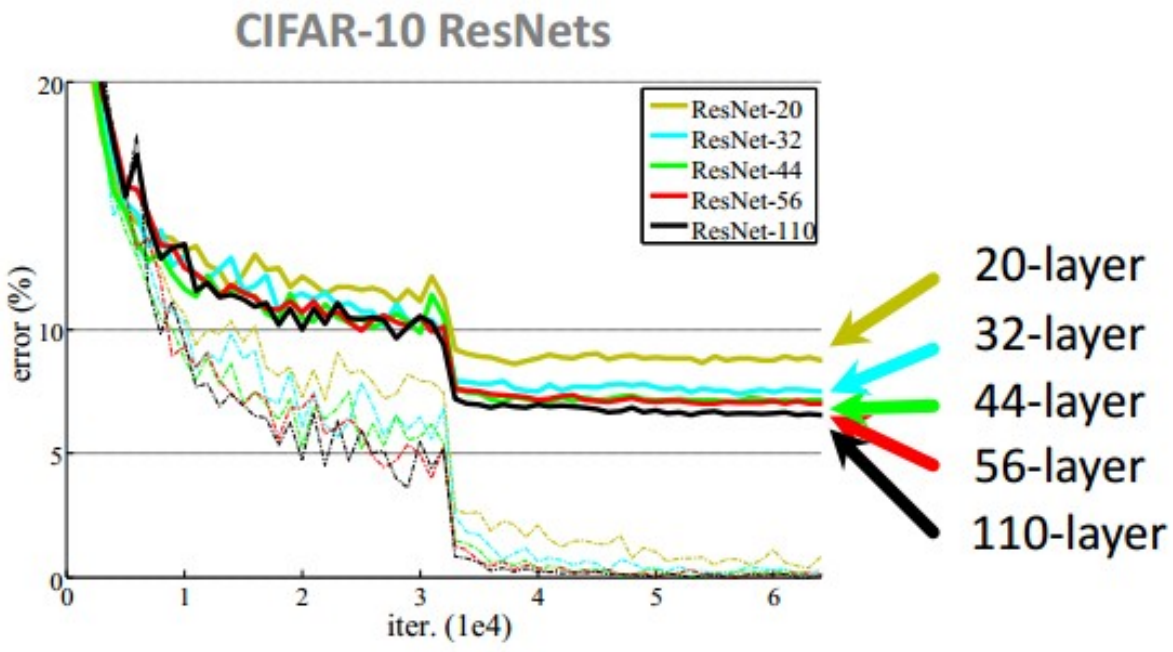
https://www.youtube.com/watch?v=1PGLj-uKT1w



(slide from Kaiming He's recent presentation)

CIFAR-10 experiments

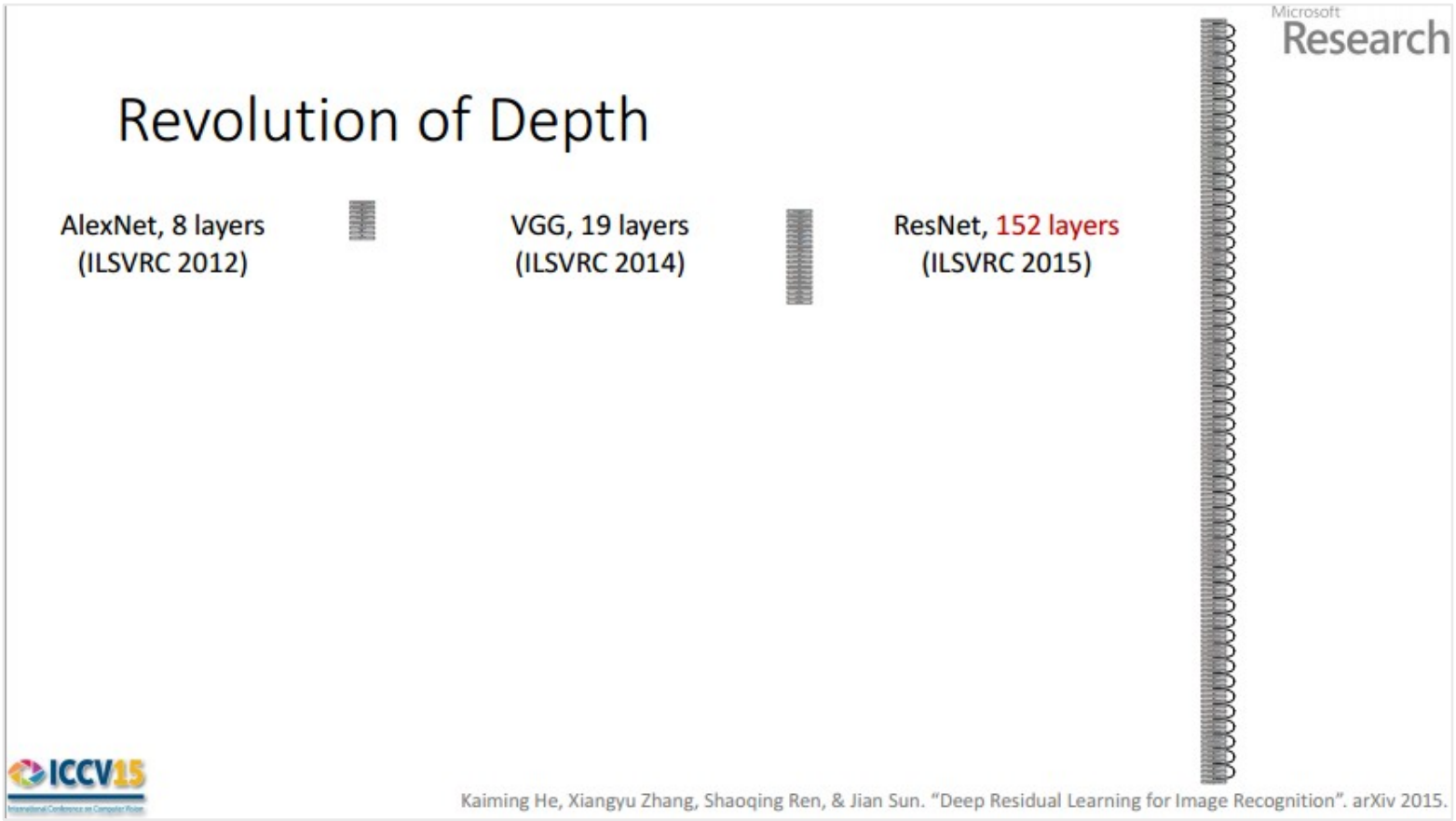




Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



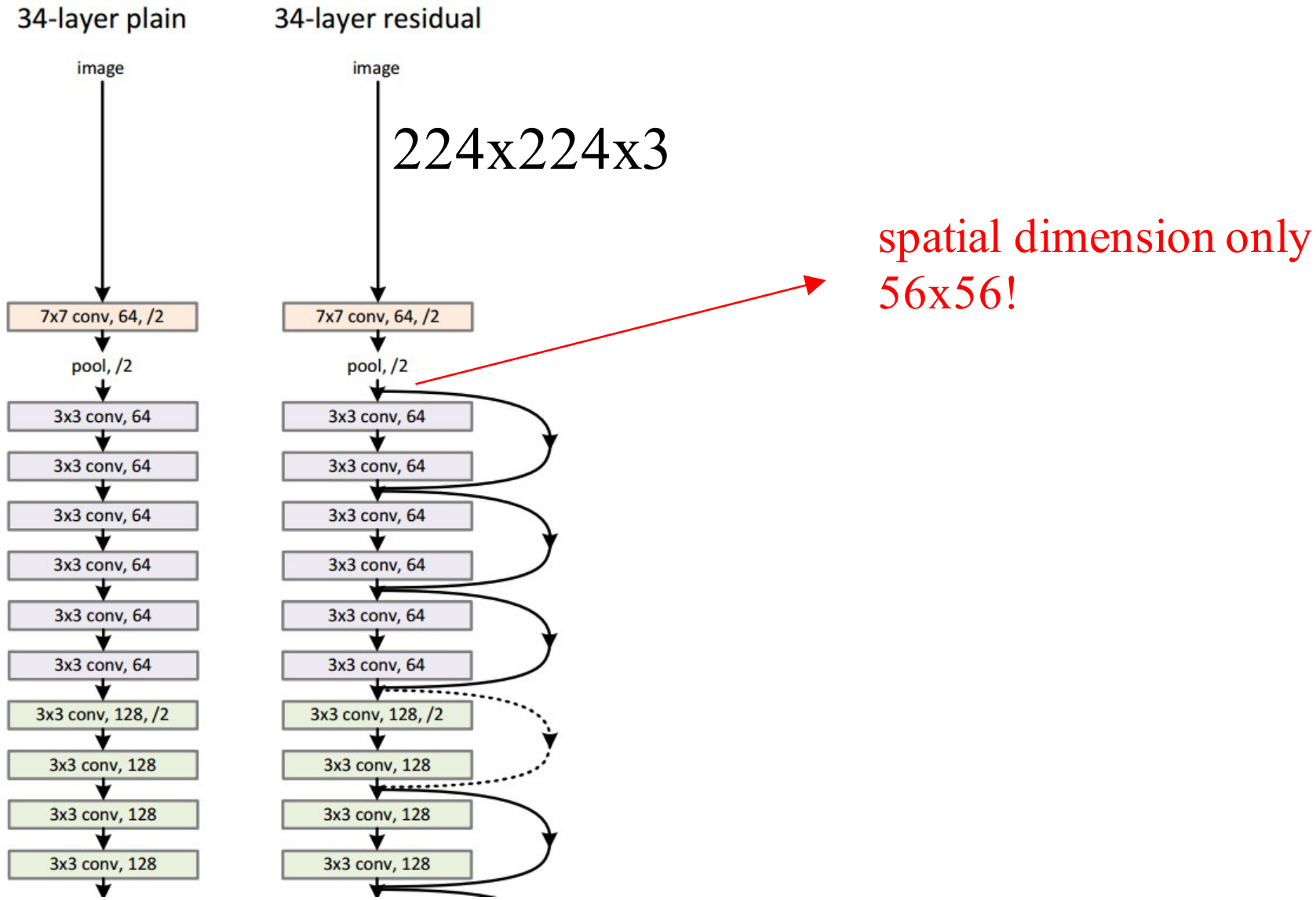
2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

(slide from Kaiming He's recent presentation)

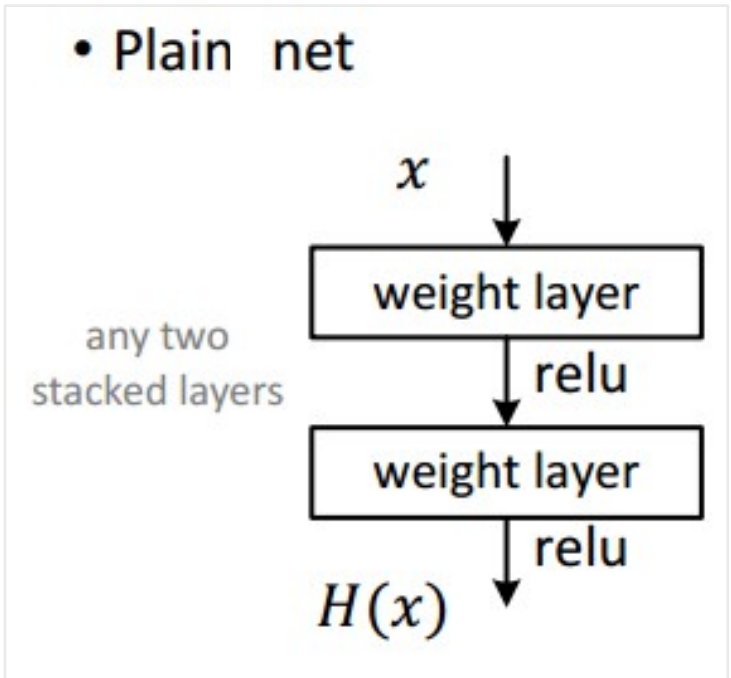
Case Study: ResNet

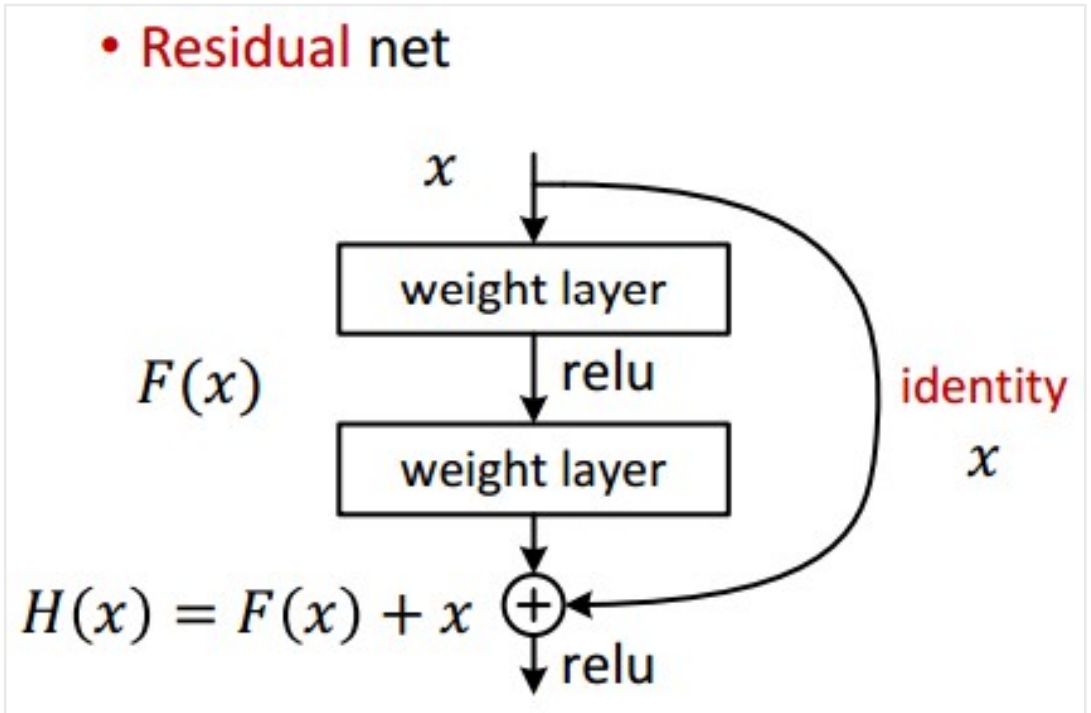
[He et al., 2015]



Case Study: ResNet

[He et al., 2015]





Case Study: ResNet

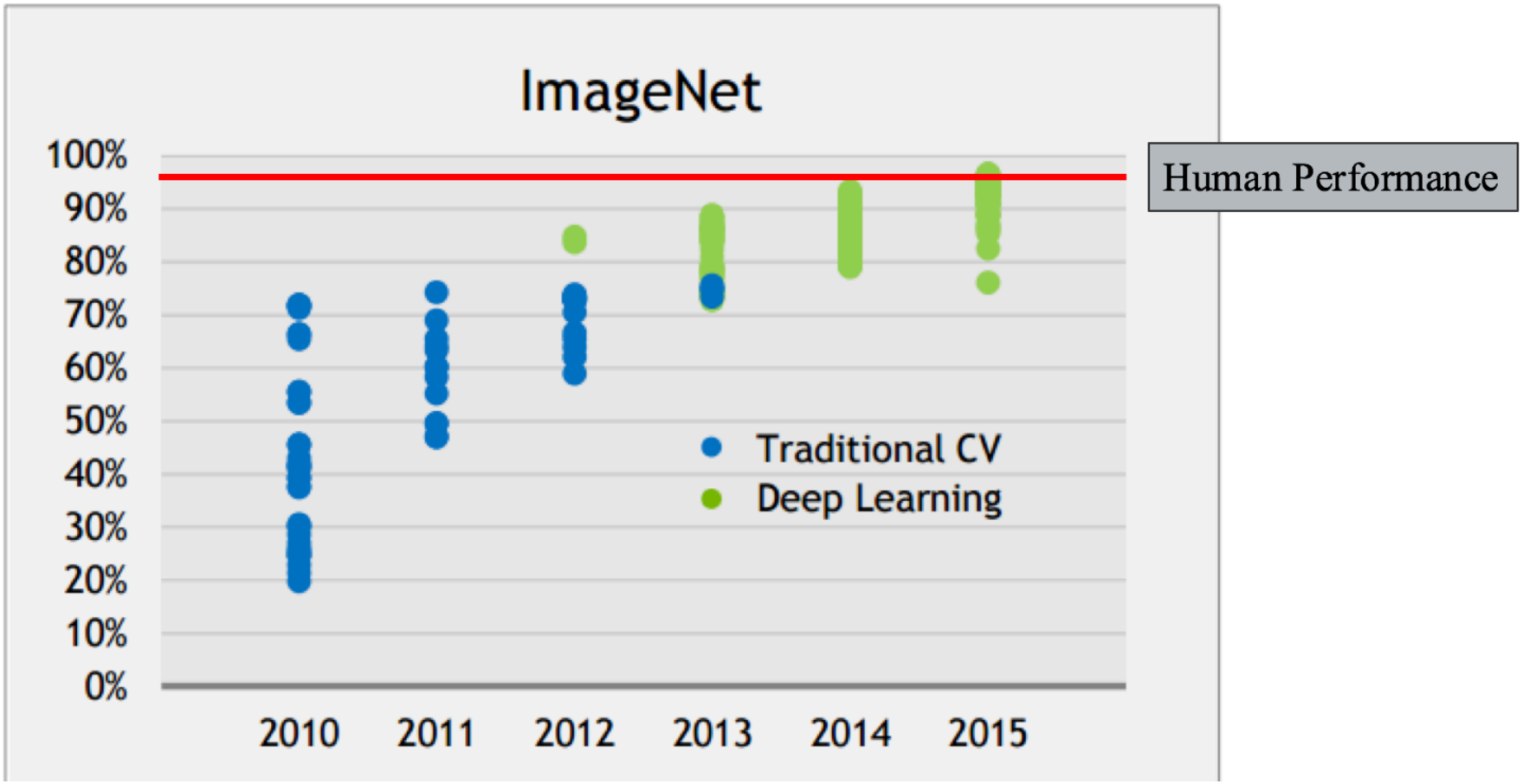
[He et al., 2015]

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

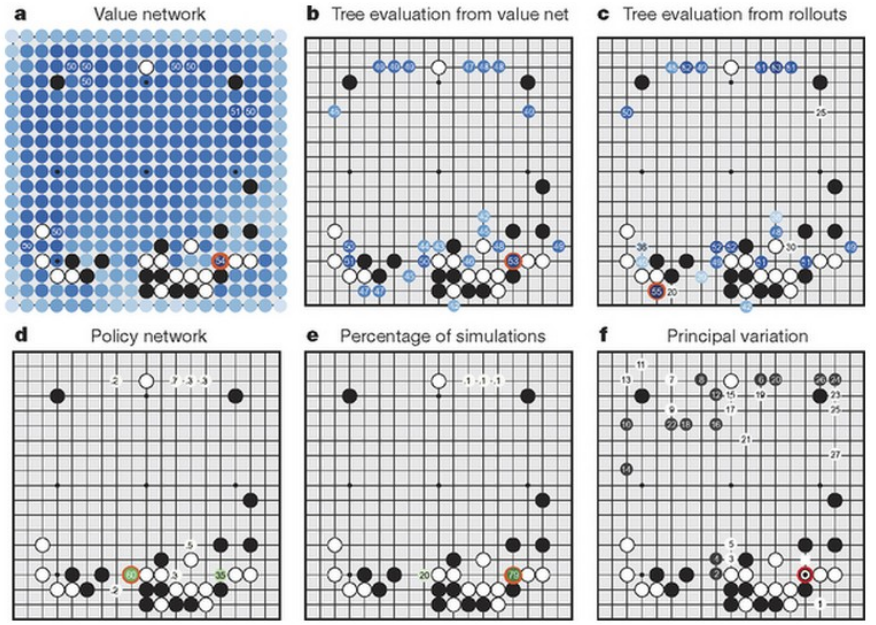
ILVRC 2016

- Classification error is down to 3.0% from 3.6% last year.
- Pretty boring, best model is just an ensemble
- https://www.reddit.com/r/MachineLearning/comments/54jiyy/large_scale_visual_recognition_challenge_2016/
- Are we plateauing?

Why ConvNets?



Case Study Bonus: DeepMind's AlphaGo



The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:

[19x19x48] Input

CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]

CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]

CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like **[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K-SOFTMAX**
  where N is usually up to ~5, M is large, 0 <= K <= 2.
- but recent advances such as ResNet/GoogLeNet challenge this paradigm

Things you should know:

"ConvNets need a lot of data to train"

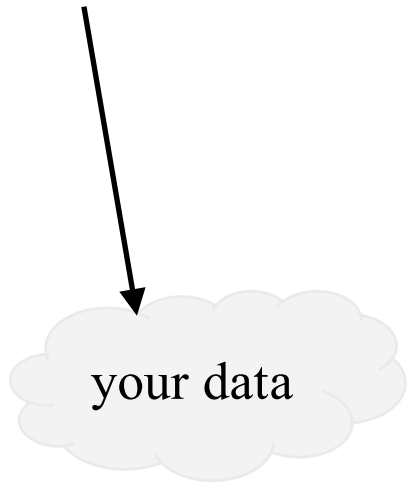
Things you should know:

"ConvNets need a lot of data to train"

finetuning! we don't always need to train ConvNets from scratch.

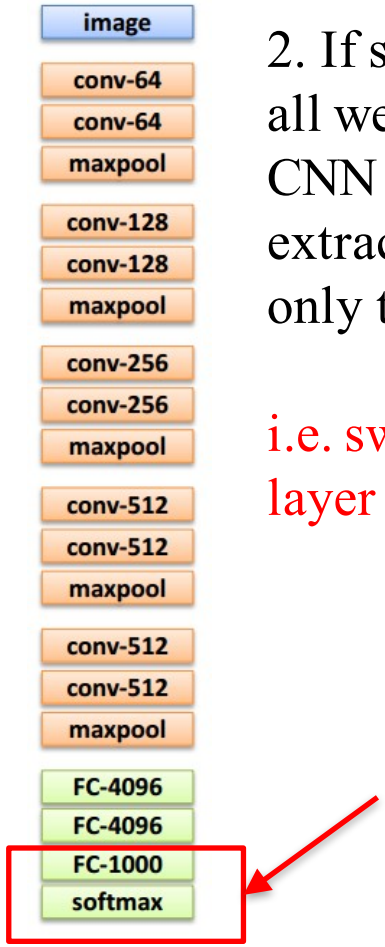
1. Train on ImageNet

2. Finetune network on your own data



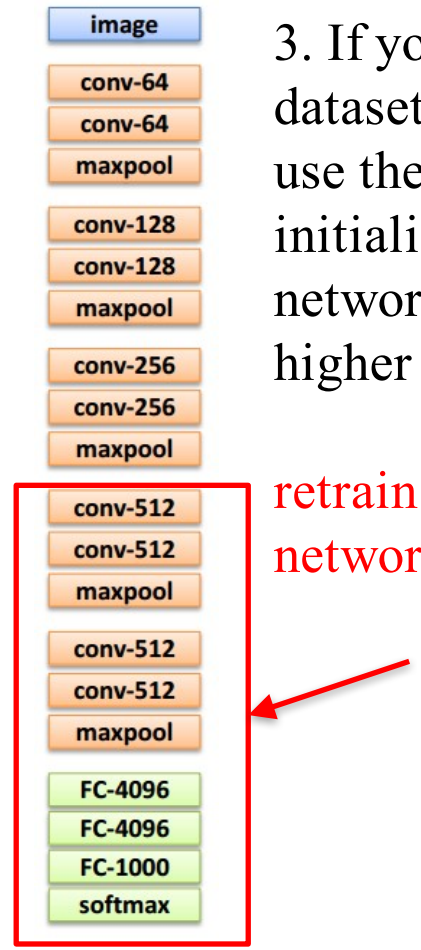
Transfer Learning with CNNs

1. Train on ImageNet



2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

Transfer Learning with CNNs

1. Train on Imagenet

Transfer Learning with CNNs

Transfer Learning with CNNs

1. Train on Imagenet

2. Small dataset: **feature extractor**

Freeze these

Train this

3. Medium dataset: **finetuning**

more data = retrain more of the network (or all of it)

Freeze these

Train this

Transfer Learning with CNNs

1. Train on Imagenet

2. Small dataset: **feature extractor**

Freeze these

Train this

3. Medium dataset: **finetuning**

more data = retrain more of the network (or all of it)

Freeze these

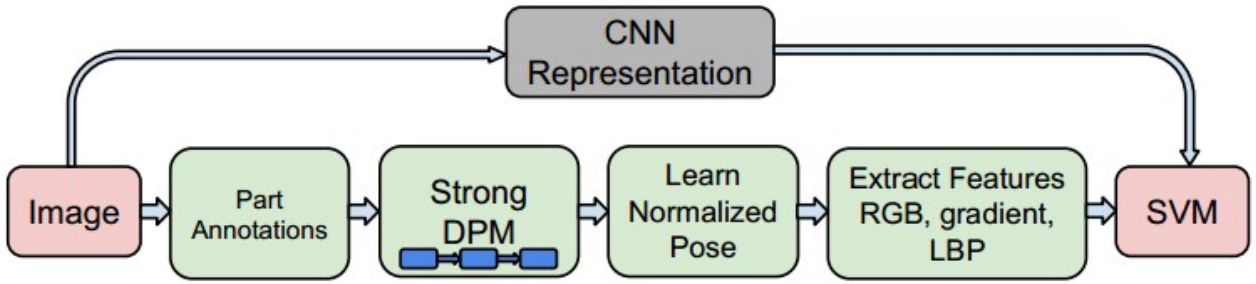
tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers

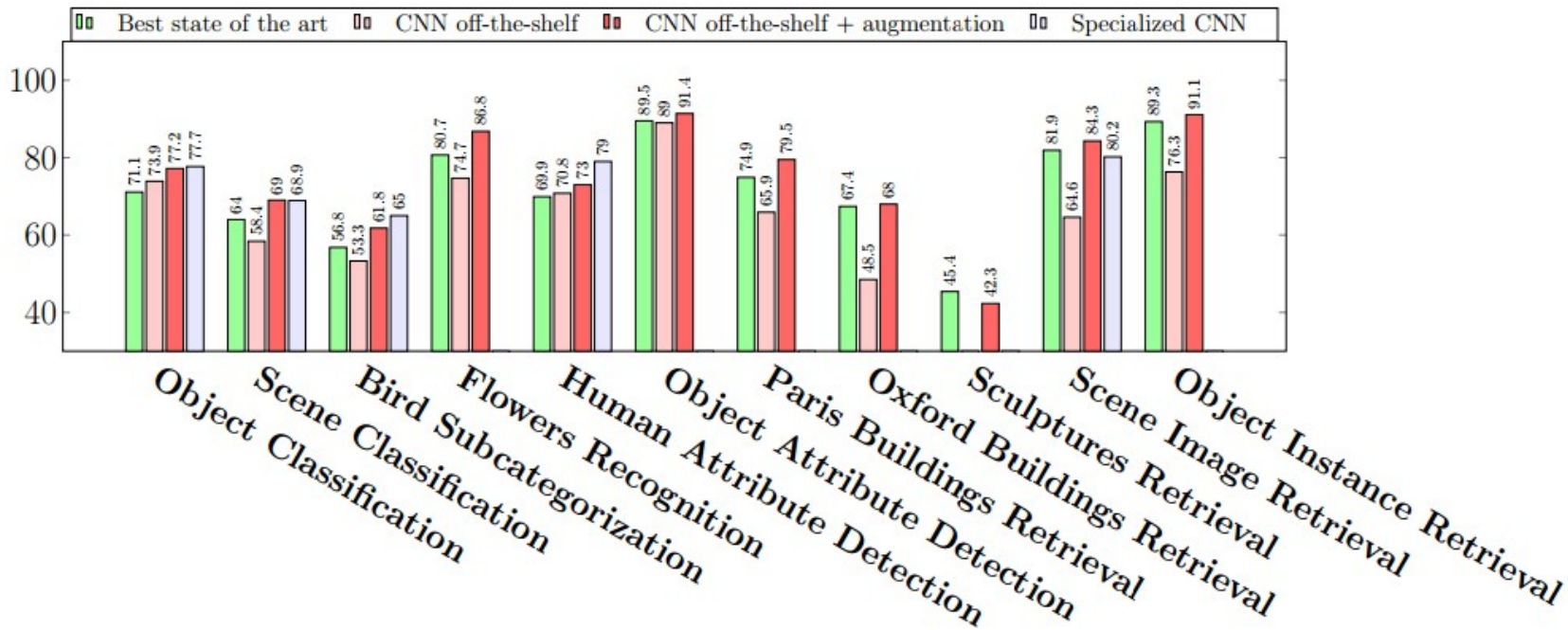
Train this

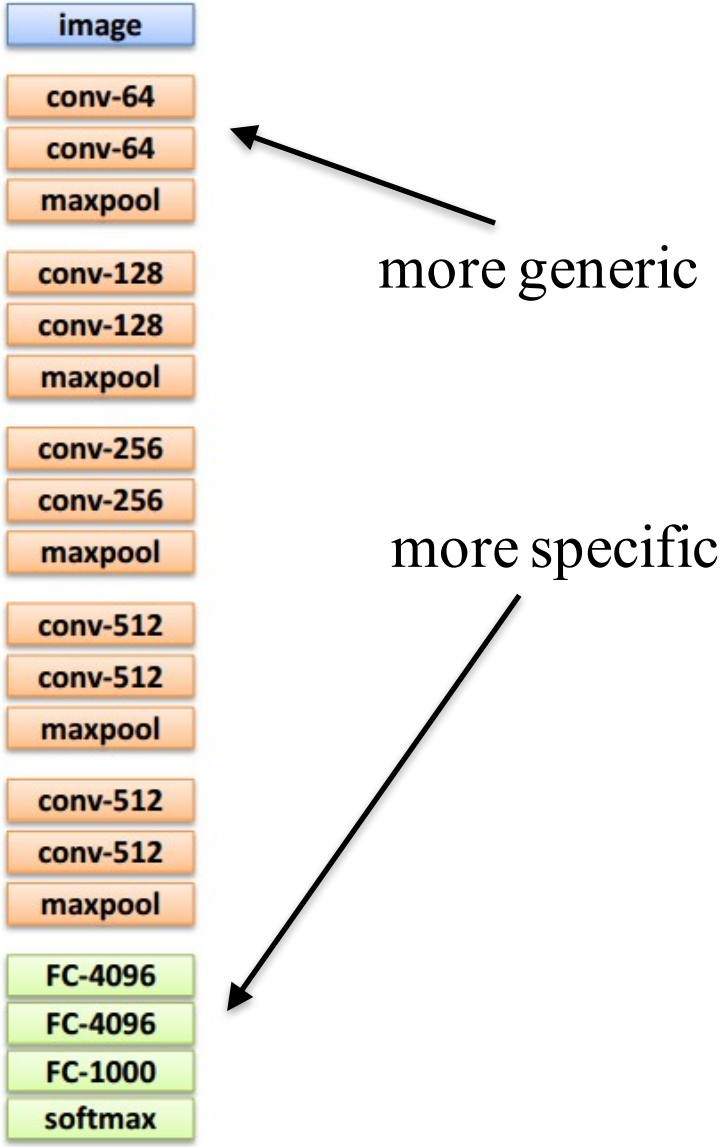
CNN Features off-the-shelf: an Astounding Baseline for Recognition [Razavian et al, 2014]

DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition [Donahue, Jia*, et al., 2013]*

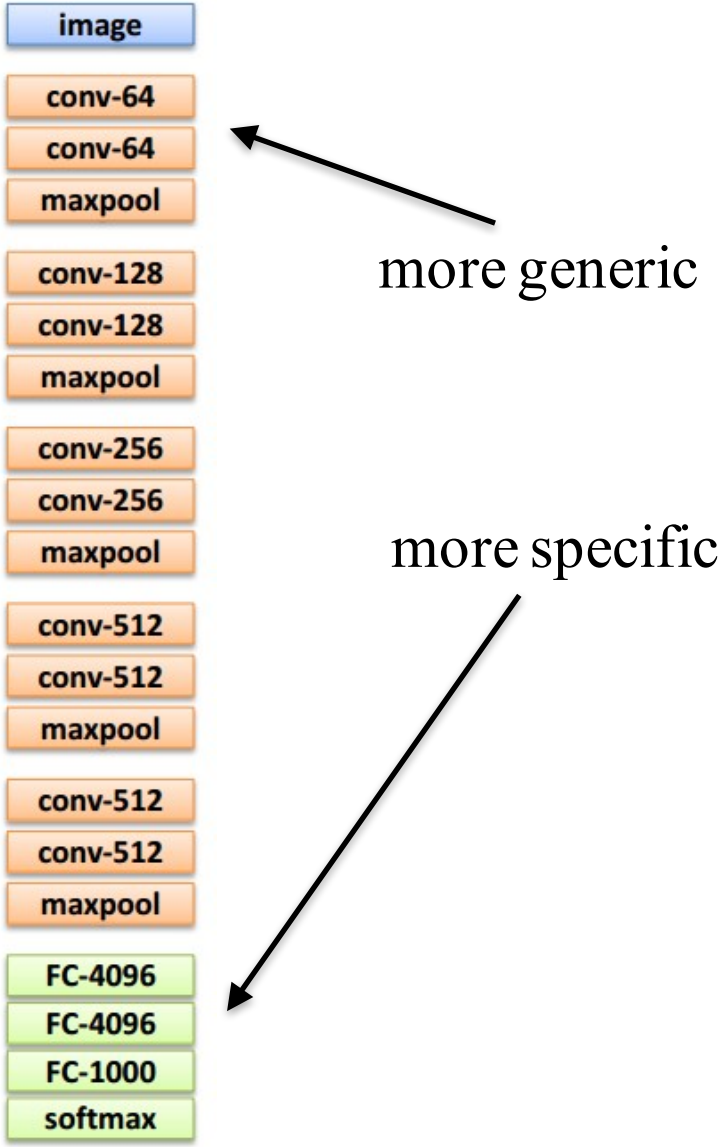
	$DeCAF_6$	$DeCAF_7$
LogReg	**40.94 ± 0.3**	40.84 ± 0.3
SVM	39.36 ± 0.3	40.66 ± 0.3
Xiao et al. (2010)	38.0	



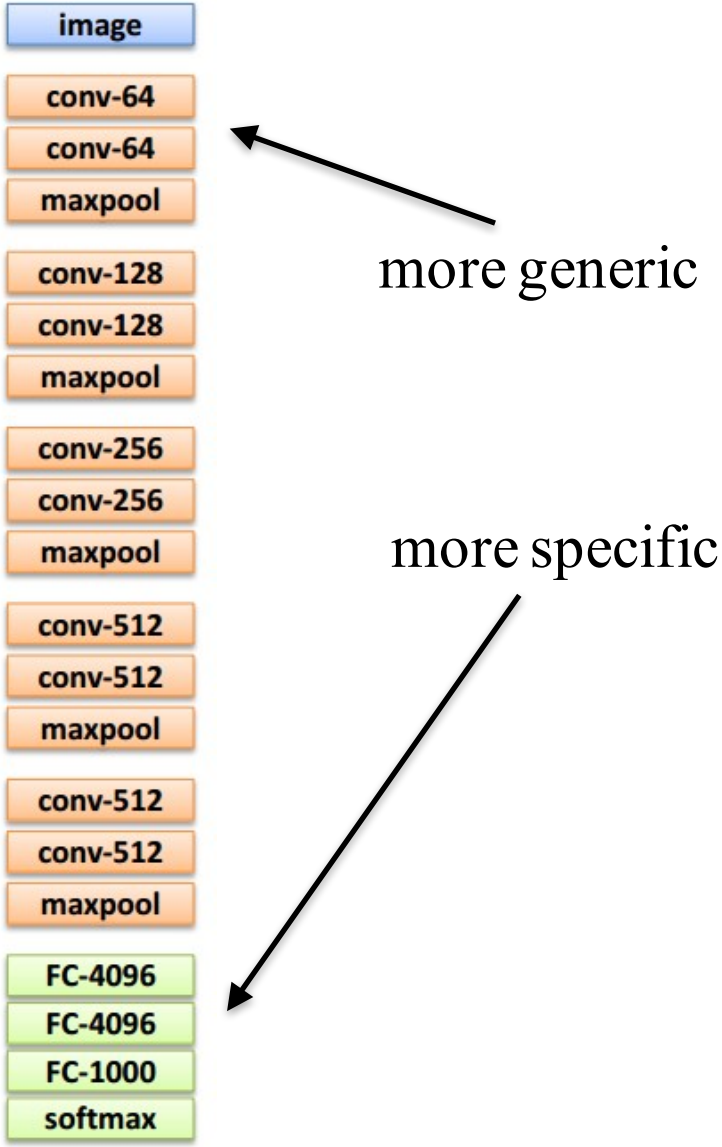




	very similar dataset	very different dataset
very little data	?	?
quite a lot of data	?	?

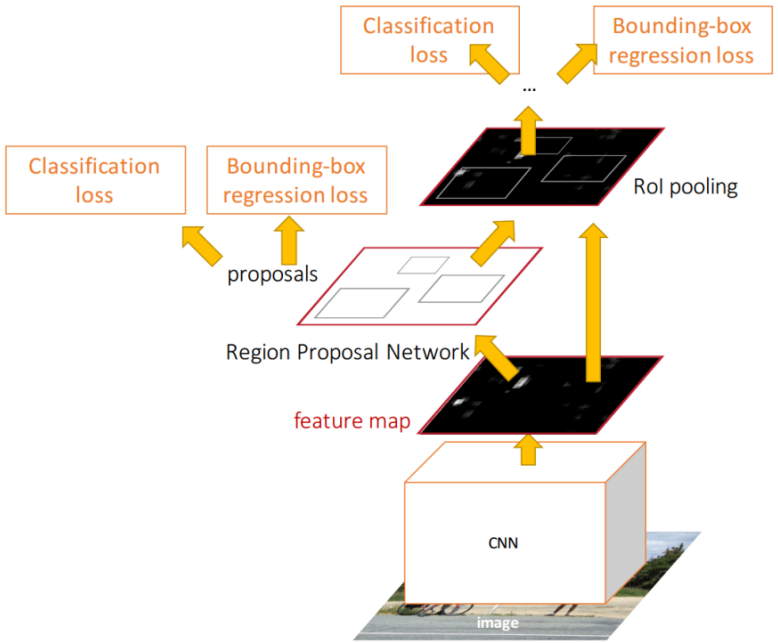


	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?



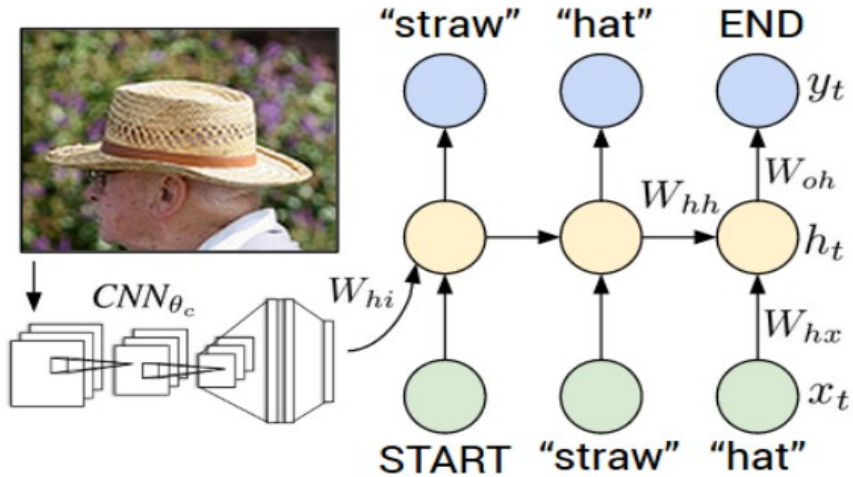
	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)

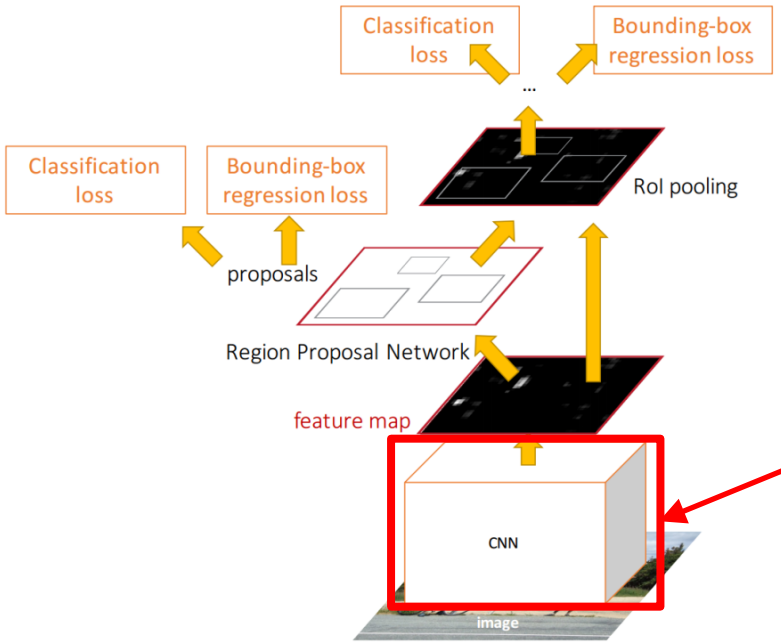


Object Detection
(Faster R-CNN)

Image Captioning: CNN + RNN

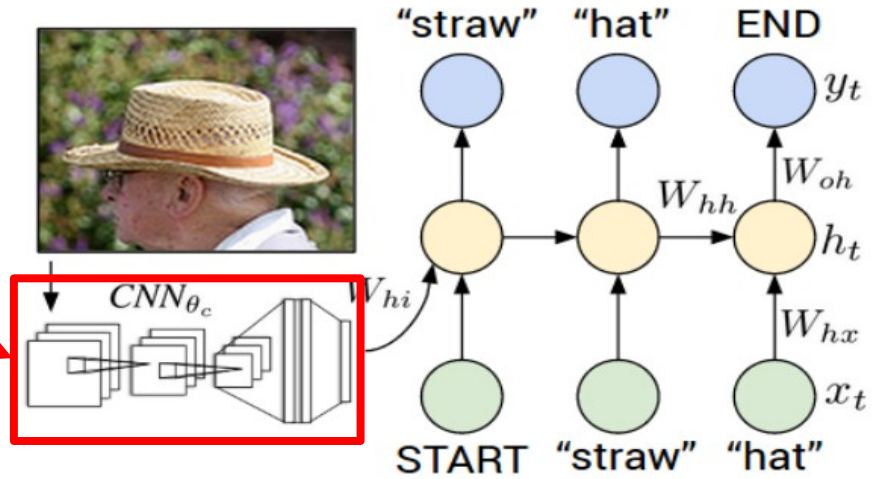


Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)



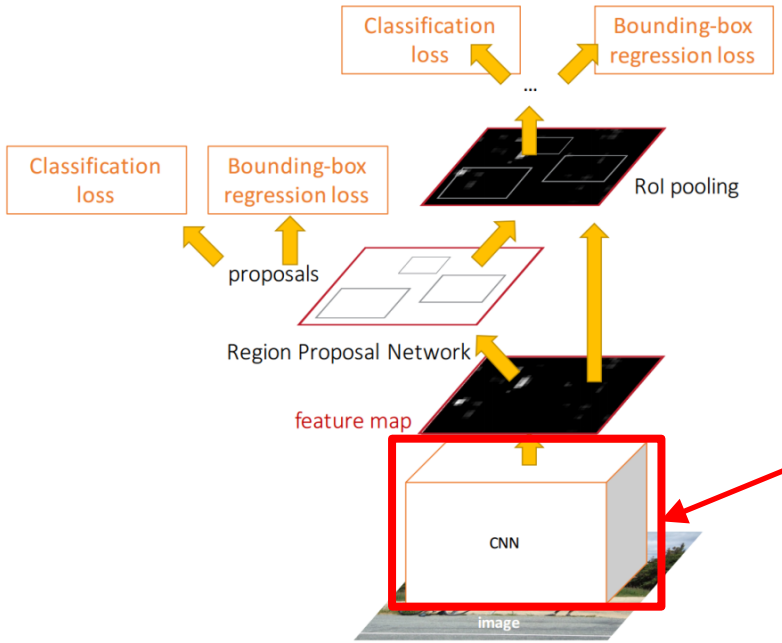
CNN pretrained on ImageNet

Image Captioning: CNN + RNN



Object Detection (Faster R-CNN)

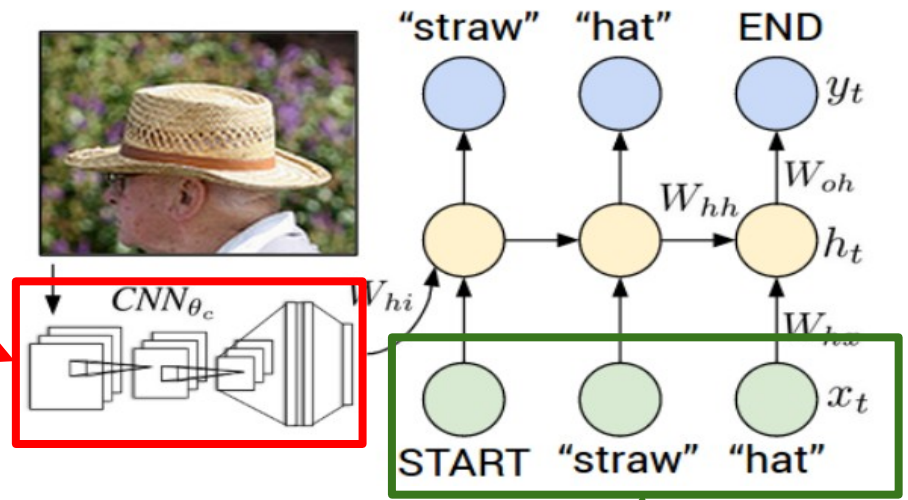
Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)



Object Detection
(Faster R-CNN)

CNN pretrained
on ImageNet

Image Captioning: CNN
+ RNN

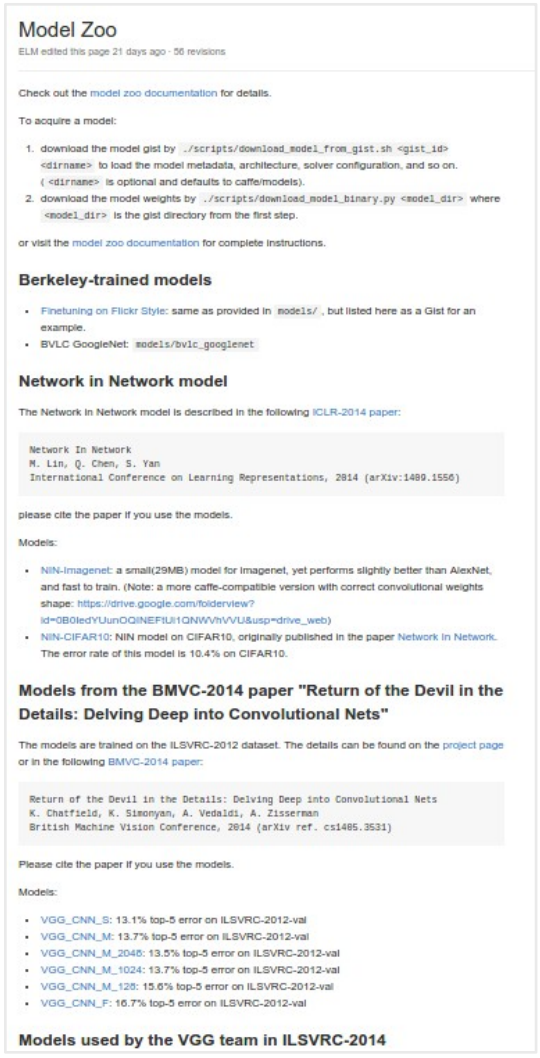


Word vectors pretrained
from word2vec

E.g. Caffe Model Zoo: Lots of pretrained ConvNets

https://github.com/BVLC/caffe/wiki/Model-Zoo

https://github.com/szagoruyko/loadcaffe



Model Zoo

ELM edited this page 21 days ago · 56 revisions

Check out the model zoo documentation for details.

To acquire a model:

1. download the model gist by `./scripts/download_model_from_gist.sh <gist_id> <dirname>` to load the model metadata, architecture, solver configuration, and so on. (`<dirname>` is optional and defaults to caffe/models).
2. download the model weights by `./scripts/download_model_binary.py <model_dir>` where `<model_dir>` is the gist directory from the first step.

or visit the model zoo documentation for complete instructions.

Berkeley-trained models

- Finetuning on Flickr Style: same as provided in `models/`, but listed here as a Gist for an example.
- BVLC GoogleNet: `models/bvlc_googlenet`

Network in Network model

The Network in Network model is described in the following ICLR-2014 paper:

```
Network In Network
M. Lin, Q. Chen, S. Yan
International Conference on Learning Representations, 2014 (arXiv:1409.1556)
```

please cite the paper if you use the models.

Models:

- NIN-Imagenet: a small(29MB) model for Imagenet, yet performs slightly better than AlexNet, and fast to train. (Note: a more caffe-compatible version with correct convolutional weights shape: https://drive.google.com/folderview?id=0B0IedYUunOQlNEFfUl1QNWVhVVU&usp=drive_web)
- NIN-CIFAR10: NIN model on CIFAR10, originally published in the paper Network In Network. The error rate of this model is 10.4% on CIFAR10.

Models from the BMVC-2014 paper "Return of the Devil in the Details: Delving Deep into Convolutional Nets"

The models are trained on the ILSVRC-2012 dataset. The details can be found on the project page or in the following BMVC-2014 paper:

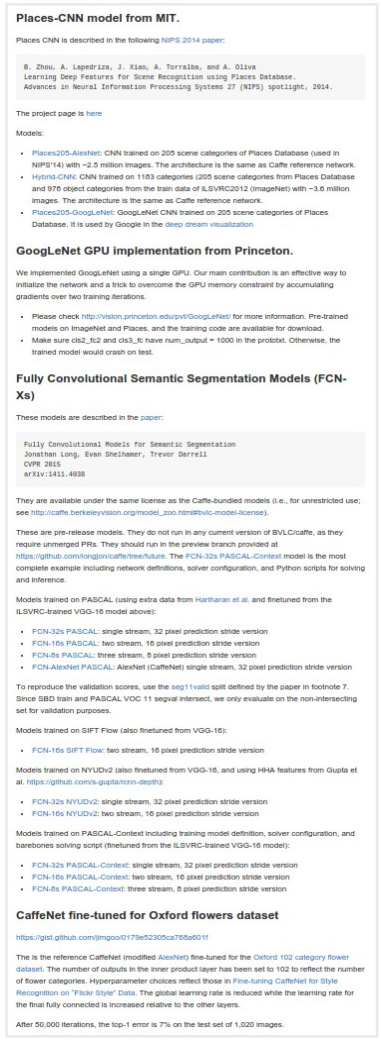
```
Return of the Devil in the Details: Delving Deep into Convolutional Nets
K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman
British Machine Vision Conference, 2014 (arXiv ref. cs1405.3531)
```

Please cite the paper if you use the models.

Models:

- VGG_CNN_S: 13.1% top-5 error on ILSVRC-2012-val
- VGG_CNN_M: 13.7% top-5 error on ILSVRC-2012-val
- VGG_CNN_M_2048: 13.5% top-5 error on ILSVRC-2012-val
- VGG_CNN_M_1024: 13.7% top-5 error on ILSVRC-2012-val
- VGG_CNN_M_128: 15.6% top-5 error on ILSVRC-2012-val
- VGG_CNN_F: 16.7% top-5 error on ILSVRC-2012-val

Models used by the VGG team in ILSVRC-2014



Places-CNN model from MIT.

Places CNN is described in the following NIPS 2014 paper:

```
B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva
Learning Deep Features for Scene Recognition using Places Database.
Advances in Neural Information Processing Systems 27 (NIPS) spotlight, 2014.
```

The project page is here

Models:

- Places205-AlexNet: CNN trained on 205 scene categories of Places Database (used in NIPS'14) with ~2.5 million images. The architecture is the same as Caffe reference network.
- Hybrid-CNN: CNN trained on 1183 categories (205 scene categories from Places Database and 978 object categories from the train data of ILSVRC2012 (ImageNet)) with ~3.6 million images. The architecture is the same as Caffe reference network.
- Places205-GoogLeNet: GoogLeNet CNN trained on 205 scene categories of Places Database. It is used by Google in the deep dream visualization

GoogLeNet GPU implementation from Princeton.

We implemented GoogLeNet using a single GPU. Our main contribution is an effective way to initialize the network and a trick to overcome the GPU memory constraint by accumulating gradients over two training iterations.

- Please check http://vision.princeton.edu/pvt/GoogLeNet/ for more information. Pre-trained models on ImageNet and Places, and the training code are available for download.
- Make sure cls2_fc2 and cls3_fc have num_output = 1000 in the prototxt. Otherwise, the trained model would crash on test.

Fully Convolutional Semantic Segmentation Models (FCN-Xs)

These models are described in the paper:

```
Fully Convolutional Models for Semantic Segmentation
Jonathan Long, Evan Shelhamer, Trevor Darrell
CVPR 2015
arXiv:1411.4038
```

They are available under the same license as the Caffe-bundled models (i.e., for unrestricted use; see http://caffe.berkeleyvision.org/model_zoo.html#bvlc-model-license).

These are pre-release models. They do not run in any current version of BVLC/caffe, as they require unmerged PRs. They should run in the preview branch provided at https://github.com/longjon/caffe/tree/future. The FCN-32s PASCAL-Context model is the most complete example including network definitions, solver configuration, and Python scripts for solving and inference.

Models trained on PASCAL (using extra data from Hariharan et al. and finetuned from the ILSVRC-trained VGG-16 model above):

- FCN-32s PASCAL: single stream, 32 pixel prediction stride version
- FCN-16s PASCAL: two stream, 16 pixel prediction stride version
- FCN-8s PASCAL: three stream, 8 pixel prediction stride version
- FCN-AlexNet PASCAL: AlexNet (CaffeNet) single stream, 32 pixel prediction stride version

To reproduce the validation scores, use the seg11valid split defined by the paper in footnote 7. Since SBD train and PASCAL VOC 11 segval intersect, we only evaluate on the non-intersecting set for validation purposes.

Models trained on SIFT Flow (also finetuned from VGG-16):

- FCN-16s SIFT Flow: two stream, 16 pixel prediction stride version

Models trained on NYUDv2 (also finetuned from VGG-16, and using HHA features from Gupta et al. https://github.com/s-gupta/rcnn-depth):

- FCN-32s NYUDv2: single stream, 32 pixel prediction stride version
- FCN-16s NYUDv2: two stream, 16 pixel prediction stride version

Models trained on PASCAL-Context including training model definition, solver configuration, and barebones solving script (finetuned from the ILSVRC-trained VGG-16 model):

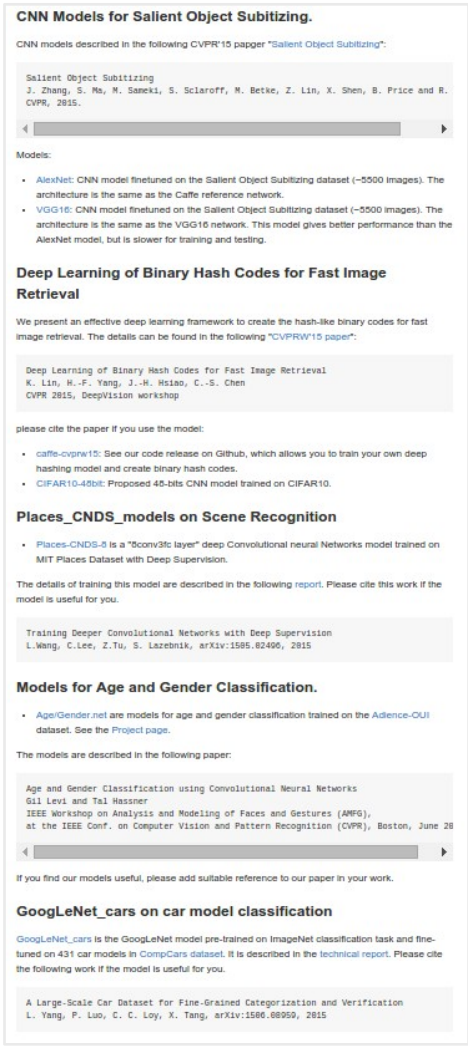
- FCN-32s PASCAL-Context: single stream, 32 pixel prediction stride version
- FCN-16s PASCAL-Context: two stream, 16 pixel prediction stride version
- FCN-8s PASCAL-Context: three stream, 8 pixel prediction stride version

CaffeNet fine-tuned for Oxford flowers dataset

https://gist.github.com/jimgoo/0175e52305ca766a601f

The is the reference CaffeNet (modified AlexNet) fine-tuned for the Oxford 102 category flower dataset. The number of outputs in the inner product layer has been set to 102 to reflect the number of flower categories. Hyperparameter choices reflect those in Fine-tuning CaffeNet for Style Recognition on "Flickr Style" Data. The global learning rate is reduced while the learning rate for the final fully connected is increased relative to the other layers.

After 50,000 iterations, the top-1 error is 7% on the test set of 1,020 images.



CNN Models for Salient Object Subitizing.

CNN models described in the following CVPR'15 papger "Salient Object Subitizing":

```
Salient Object Subitizing
J. Zhang, S. Ma, M. Sameki, S. Sclaroff, M. Betke, Z. Lin, X. Shen, B. Price and R.
CVPR, 2015.
```

Models:

- AlexNet: CNN model finetuned on the Salient Object Subitizing dataset (~5500 images). The architecture is the same as the Caffe reference network.
- VGG16: CNN model finetuned on the Salient Object Subitizing dataset (~5500 images). The architecture is the same as the VGG16 network. This model gives better performance than the AlexNet model, but is slower for training and testing.

Deep Learning of Binary Hash Codes for Fast Image Retrieval

We present an effective deep learning framework to create the hash-like binary codes for fast image retrieval. The details can be found in the following "CVPRW'15 paper":

```
Deep Learning of Binary Hash Codes for Fast Image Retrieval
K. Lin, H.-F. Yang, J.-H. Hsiao, C.-S. Chen
CVPR 2015, DeepVision workshop
```

please cite the paper if you use the model:

- caffe-cvprw15: See our code release on Github, which allows you to train your own deep hashing model and create binary hash codes.
- CIFAR10-48bit: Proposed 48-bits CNN model trained on CIFAR10.

Places_CNDS_models on Scene Recognition

- Places-CNDS-8 is a "8conv3fc layer" deep Convolutional neural Networks model trained on MIT Places Dataset with Deep Supervision.

The details of training this model are described in the following report. Please cite this work if the model is useful for you.

```
Training Deeper Convolutional Networks with Deep Supervision
L.Wang, C.Lee, Z.Tu, S. Lazebnik, arXiv:1505.02496, 2015
```

Models for Age and Gender Classification.

- Age/Gender.net are models for age and gender classification trained on the Adience-OUI dataset. See the Project page.

The models are described in the following paper:

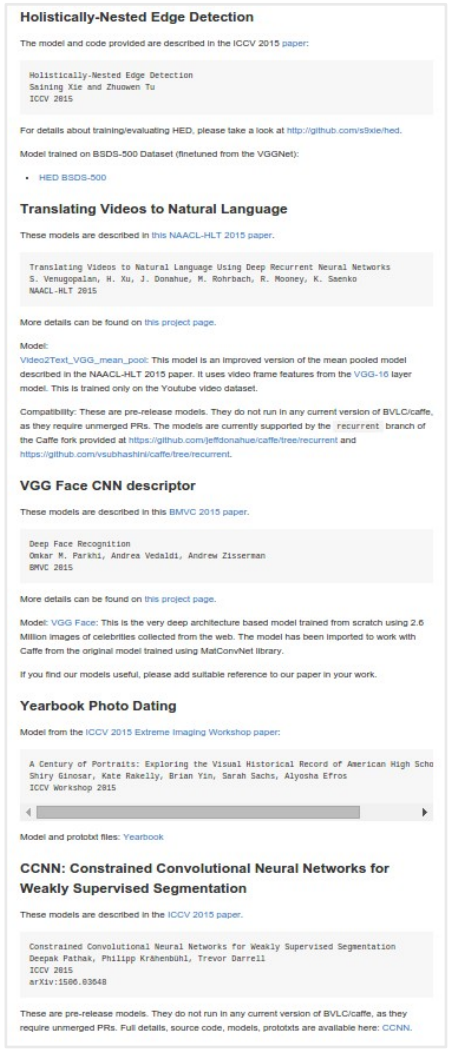
```
Age and Gender Classification using Convolutional Neural Networks
Gil Levi and Tal Hassner
IEEE Workshop on Analysis and Modeling of Faces and Gestures (AMFG),
at the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston, June 20
```

If you find our models useful, please add suitable reference to our paper in your work.

GoogLeNet_cars on car model classification

GoogLeNet_cars is the GoogLeNet model pre-trained on ImageNet classification task and fine-tuned on 431 car models in CompCars dataset. It is described in the technical report. Please cite the following work if the model is useful for you.

```
A Large-Scale Car Dataset for Fine-Grained Categorization and Verification
L. Yang, P. Luo, C. C. Loy, X. Tang, arXiv:1506.08959, 2015
```



Holistically-Nested Edge Detection

The model and code provided are described in the ICCV 2015 paper:

```
Holistically-Nested Edge Detection
Saining Xie and Zhuowen Tu
ICCV 2015
```

For details about training/evaluating HED, please take a look at http://github.com/s9xie/hed.

Model trained on BSDS-500 Dataset (finetuned from the VGGNet):

- HED BSDS-500

Translating Videos to Natural Language

These models are described in this NAACL-HLT 2015 paper.

```
Translating Videos to Natural Language Using Deep Recurrent Neural Networks
S. Venugopalan, H. Xu, J. Donahue, M. Rohrbach, R. Mooney, K. Saenko
NAACL-HLT 2015
```

More details can be found on this project page.

Model:

Video2Text_VGG_mean_pool: This model is an improved version of the mean pooled model described in the NAACL-HLT 2015 paper. It uses video frame features from the VGG-16 layer model. This is trained only on the Youtube video dataset.

Compatibility: These are pre-release models. They do not run in any current version of BVLC/caffe, as they require unmerged PRs. The models are currently supported by the `recurrent` branch of the Caffe fork provided at https://github.com/jeffdonahue/caffe/tree/recurrent and https://github.com/vsubhashini/caffe/tree/recurrent.

VGG Face CNN descriptor

These models are described in this BMVC 2015 paper.

```
Deep Face Recognition
Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman
BMVC 2015
```

More details can be found on this project page.

Model: VGG Face: This is the very deep architecture based model trained from scratch using 2.6 Million images of celebrities collected from the web. The model has been imported to work with Caffe from the original model trained using MatConvNet library.

If you find our models useful, please add suitable reference to our paper in your work.

Yearbook Photo Dating

Model from the ICCV 2015 Extreme Imaging Workshop paper:

```
A Century of Portraits: Exploring the Visual Historical Record of American High Scho
Shiry Ginosar, Kate Rakelly, Brian Yin, Sarah Sachs, Alyosha Efros
ICCV Workshop 2015
```

Model and prototxt files: Yearbook

CCNN: Constrained Convolutional Neural Networks for Weakly Supervised Segmentation

These models are described in the ICCV 2015 paper.

```
Constrained Convolutional Neural Networks for Weakly Supervised Segmentation
Deepak Pathak, Philipp Krähenbühl, Trevor Darrell
ICCV 2015
arXiv:1506.03648
```

These are pre-release models. They do not run in any current version of BVLC/caffe, as they require unmerged PRs. Full details, source code, models, prototxts are available here: CCNN.

• • •

Visualizing and Understanding ConvNets

Understanding ConvNets

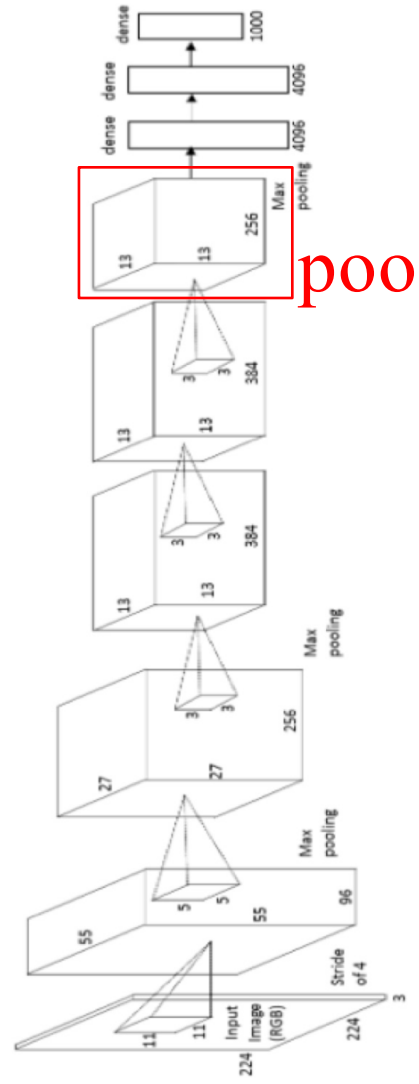
- Visualize patches that maximally activate neurons
- Visualize the weights
- Visualize the representation space
- Occlusion experiments
- Human experiment comparisons
- Deconv approaches (single backward pass)
- Optimization over image approaches (optimization)

Visualize patches that maximally activate neurons

Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).

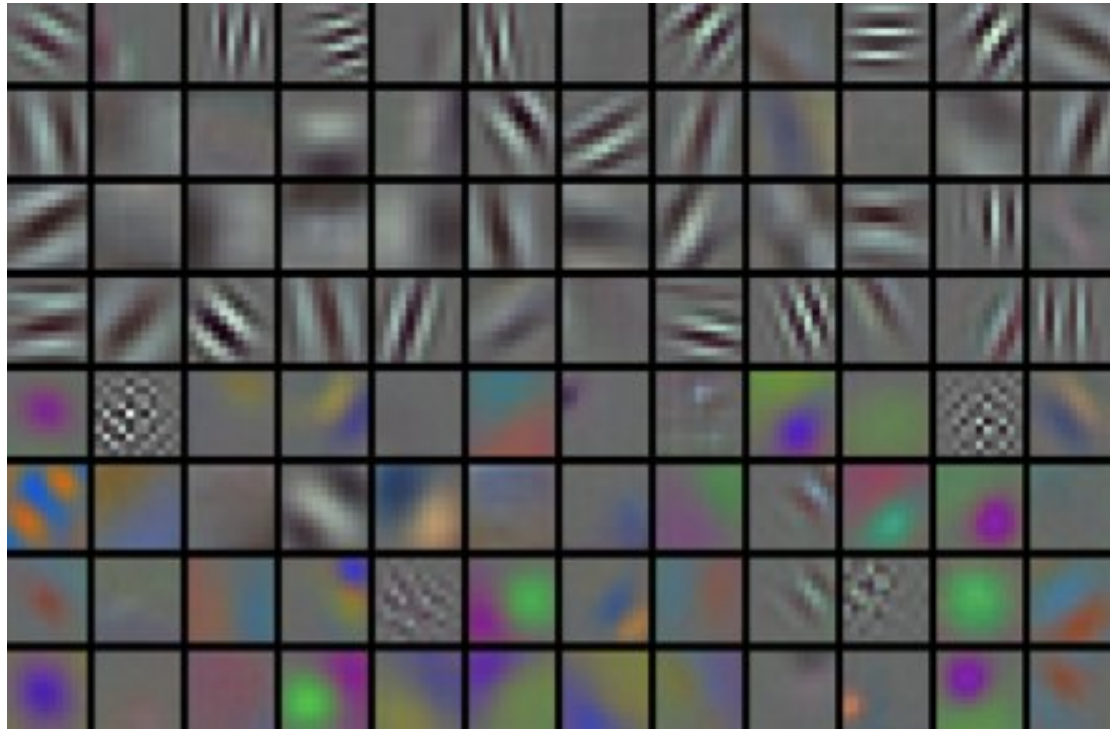
Rich feature hierarchies for accurate object detection and semantic segmentation
[Girshick, Donahue, Darrell, Malik]

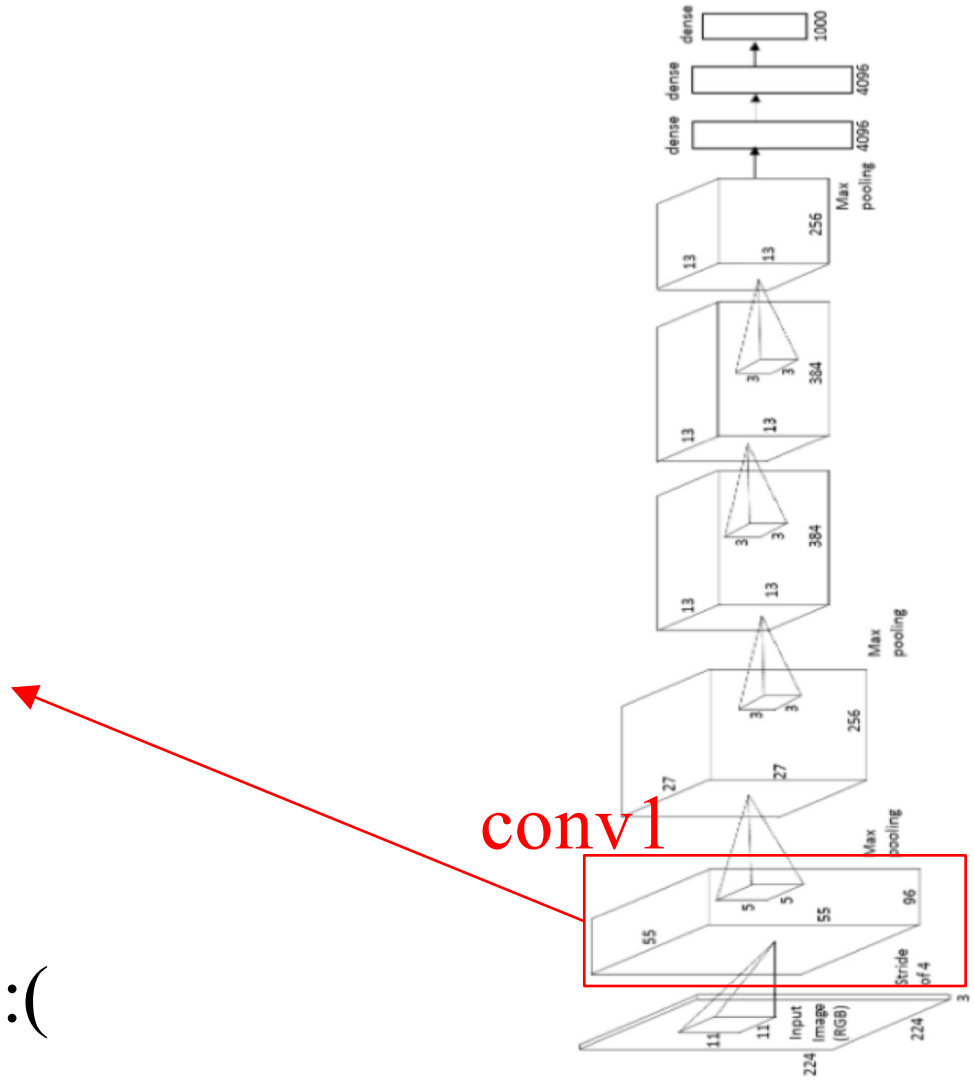
one-stream AlexNet



Visualize the filters/kernels (raw weights)

one-stream AlexNet





only interpretable on the first layer :(

Visualize the filters/kernels (raw weights)

you can still do it for higher layers, it's just not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)

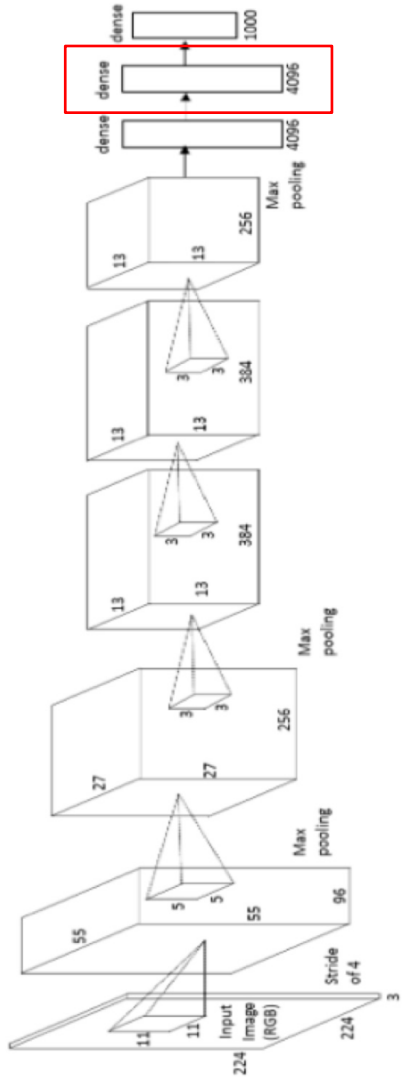
layer 1 weights

layer 2 weights

layer 3 weights

Visualizing the representation

fc7 layer



4096-dimensional "code" for an image (layer immediately before the classifier)

can collect the code for many images

Visualizing the representation

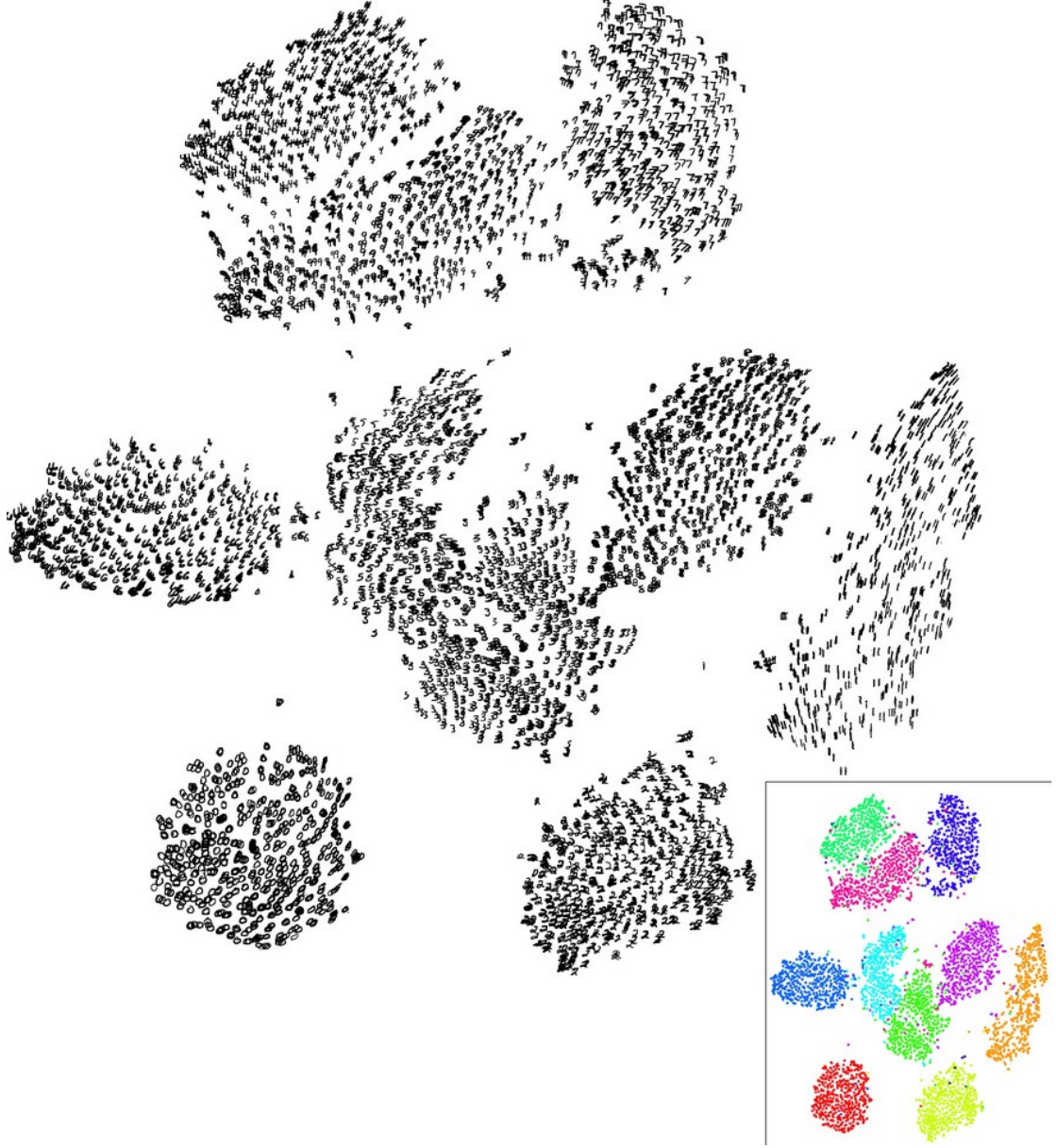
t-SNE visualization

[van der Maaten & Hinton]

Embed high-dimensional points so that locally, pairwise distances are conserved

i.e. similar things end up in similar places. dissimilar things end up wherever

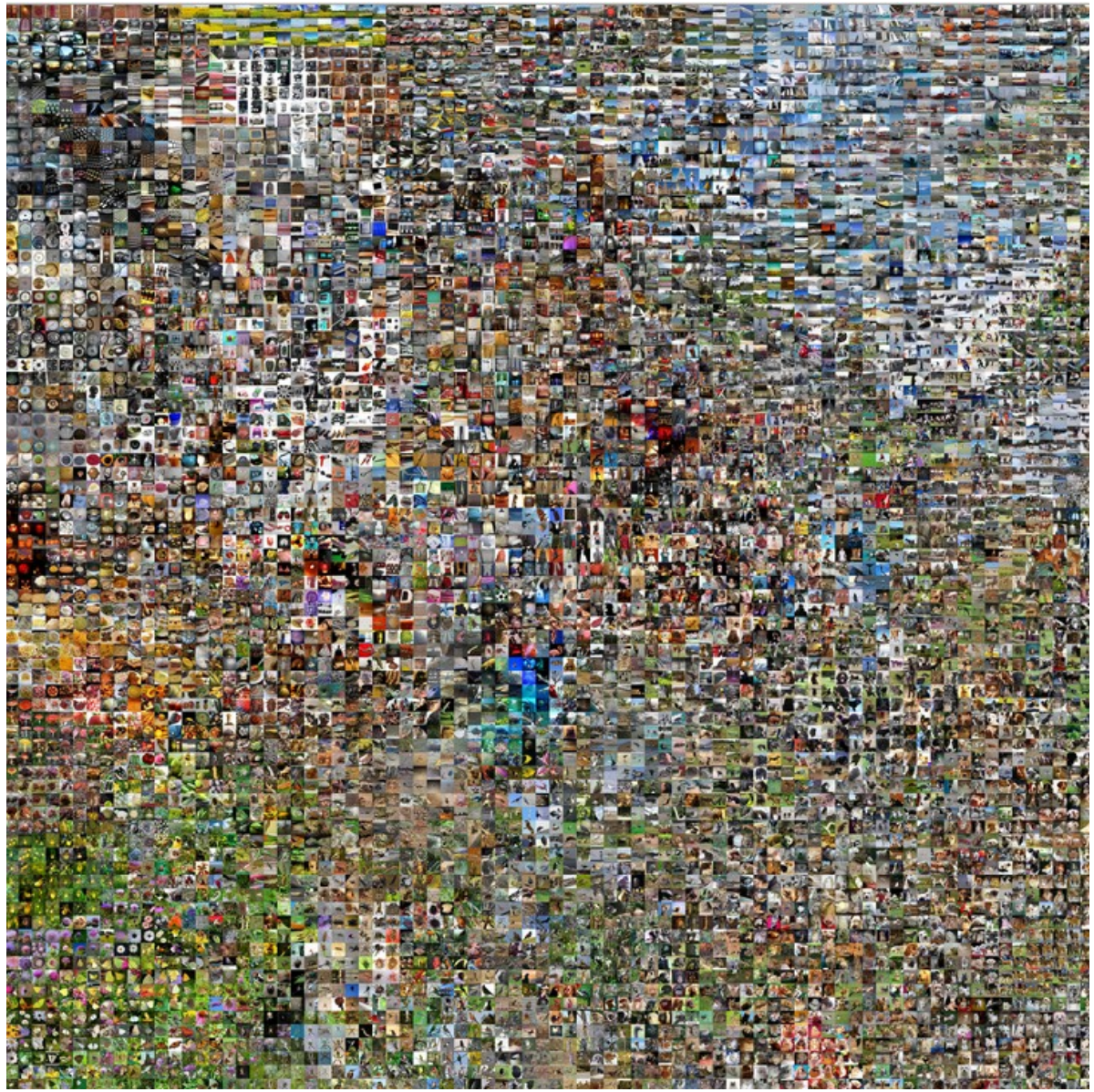
Right: Example embedding of MNIST digits (0-9) in 2D



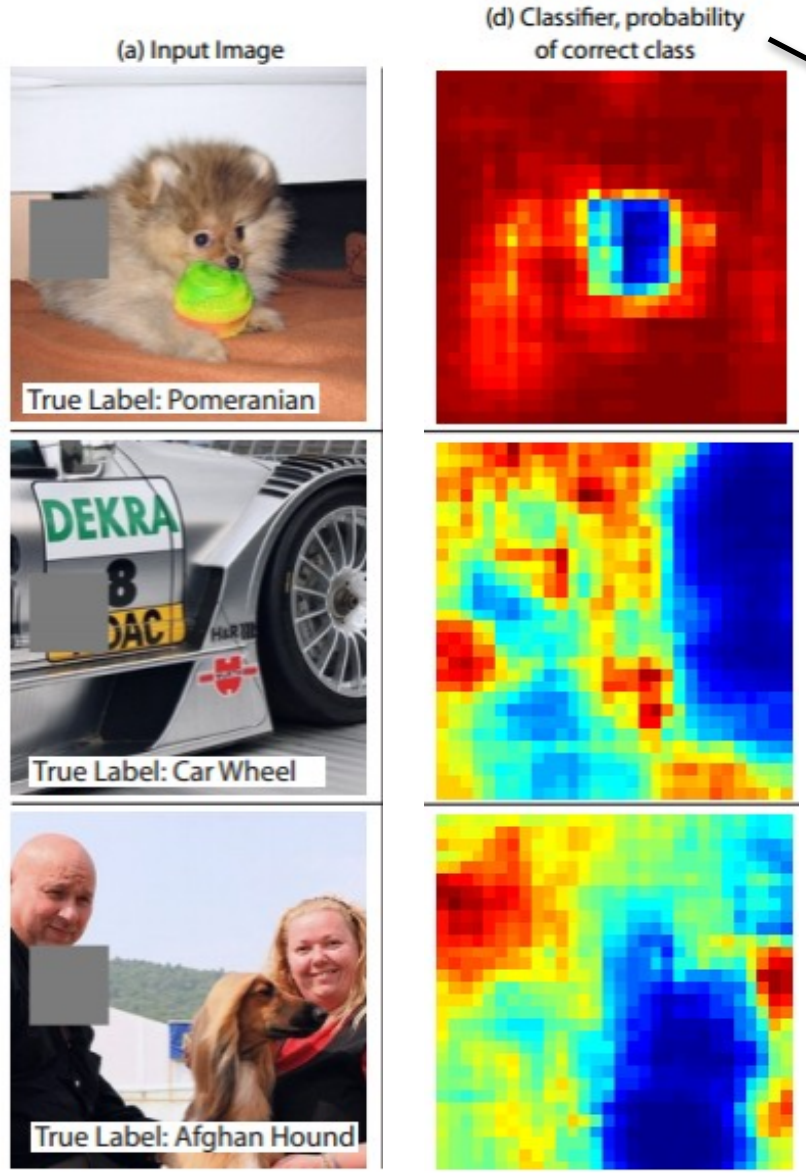
t-SNE visualization:

two images are placed nearby if their CNN codes are close. See more:

http://cs.stanford.edu/people/karpathy/cnnembed/

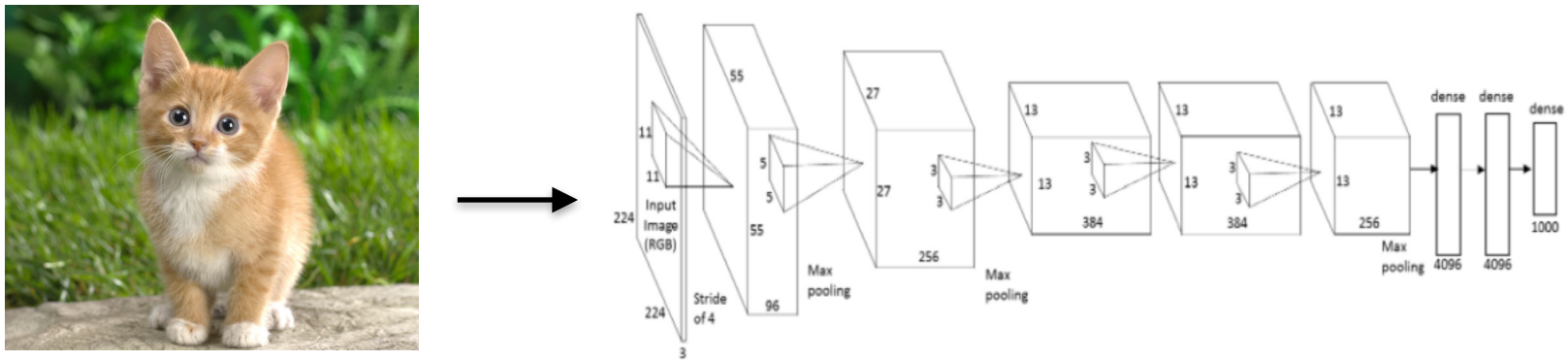


Occlusion experiments [Zeiler & Fergus 2013]



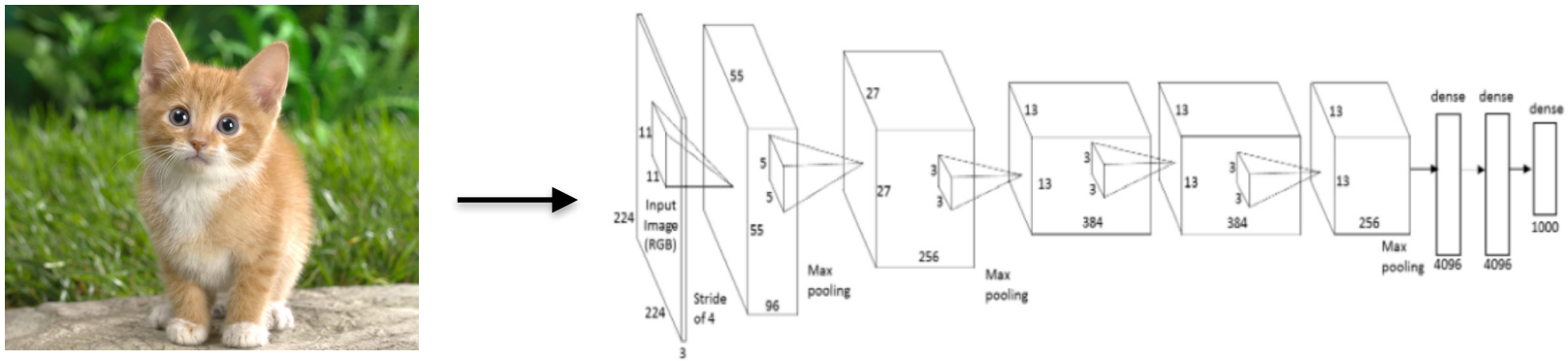
(as a function of the position of the square of zeros in the original image)

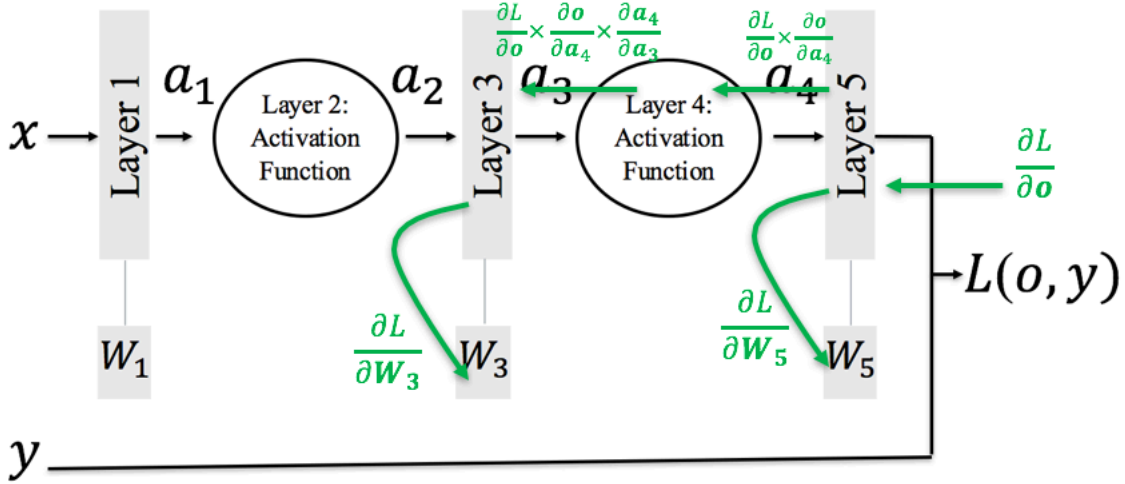
Deconv approaches



Q: how can we compute the gradient of any arbitrary neuron in the network w.r.t. the image?

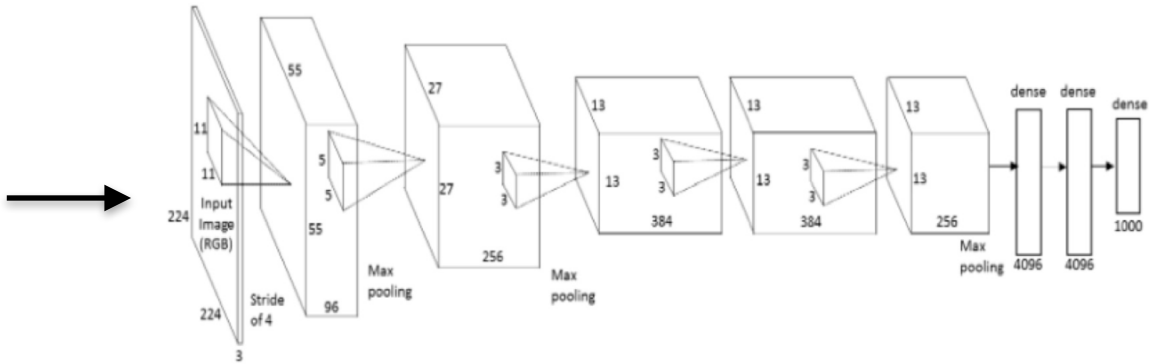
Deconv approaches



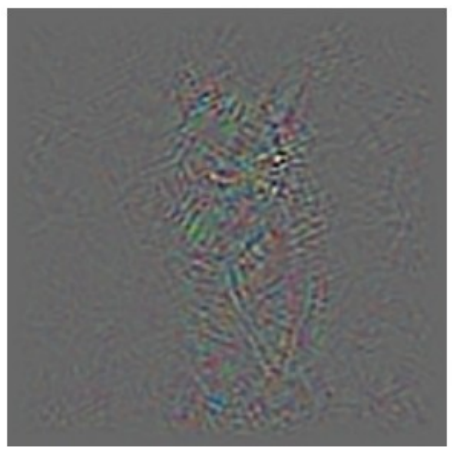


Deconv approaches

1. Feed image into net

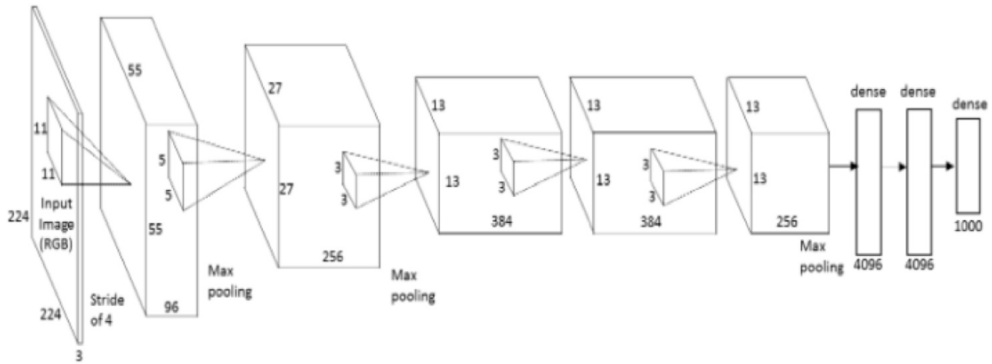


2. Pick a layer, set the gradient there to be all zero except for one 1 for some neuron of interest
3. Backprop to image:

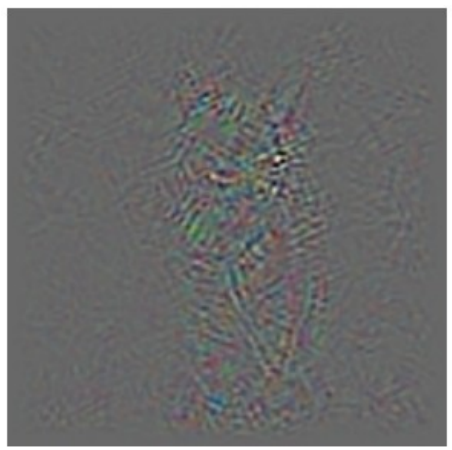


Deconv approaches

1. Feed image into net



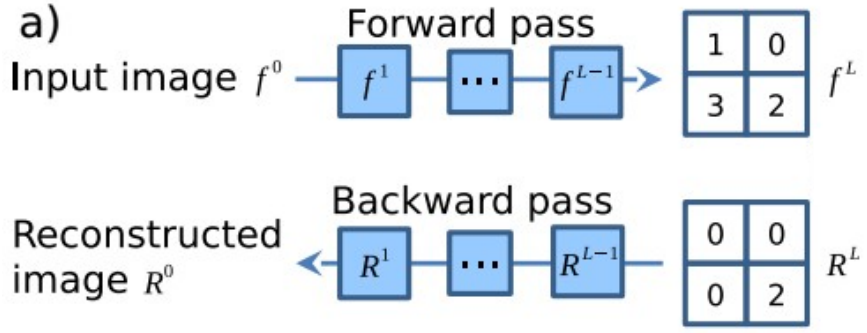
2. Pick a layer, set the gradient there to be all zero except for one 1 for some neuron of interest
3. Backprop to image:



"Guided backpropagation:" instead

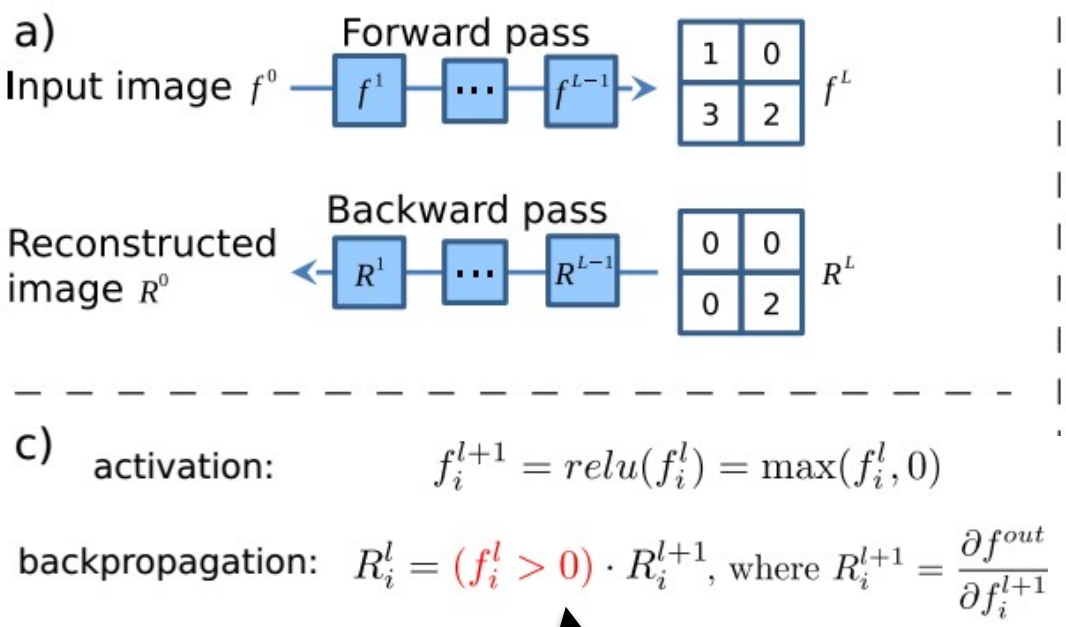
Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



Deconv approaches

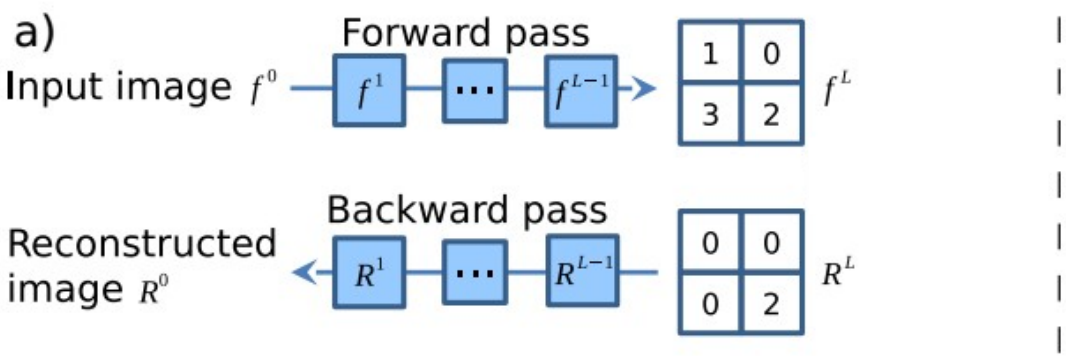
[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



Backward pass for a ReLU (will be changed in Guided Backprop)

Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]

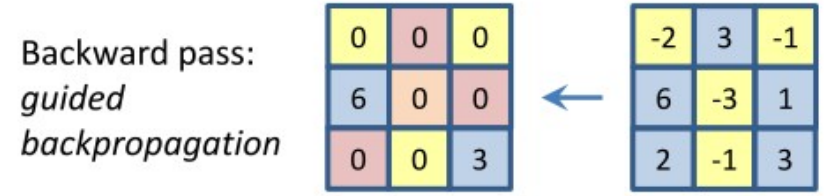


c)

activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$

backpropagation: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$

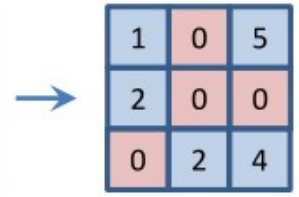
guided backpropagation: $R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$

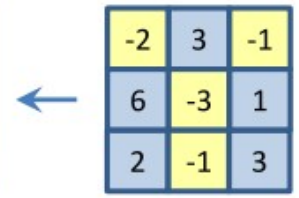


Deconv approaches

```
function GuidedReLU:updateOutput(input)
  self.output = self.relu:forward(input)
  return self.output
end

function GuidedReLU:updateGradInput(input, gradOutput)
  self.gradInput = self.relu:backward(input, gradOutput)
  local positive_mask = gradOutput:gt(0):double()
  self.gradInput = self.gradInput:cmul(positive_mask):cmul(gradOutput)
  return self.gradInput
end
```



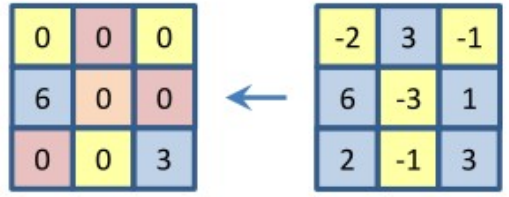


c) activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$

backpropagation: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$

guided backpropagation: $R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$

Backward pass: *guided backpropagation*

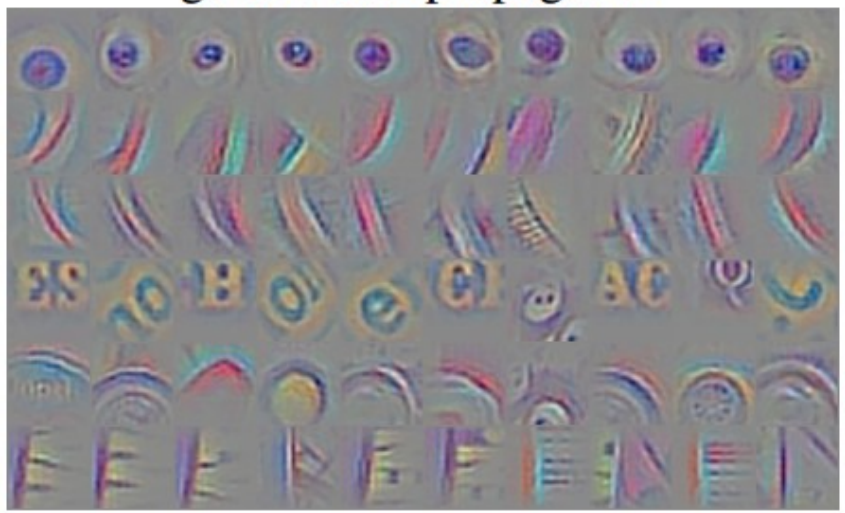


Visualization of patterns learned by the layer **conv6** (top) and layer **conv9** (bottom) of the network trained on ImageNet.

Each row corresponds to one filter.

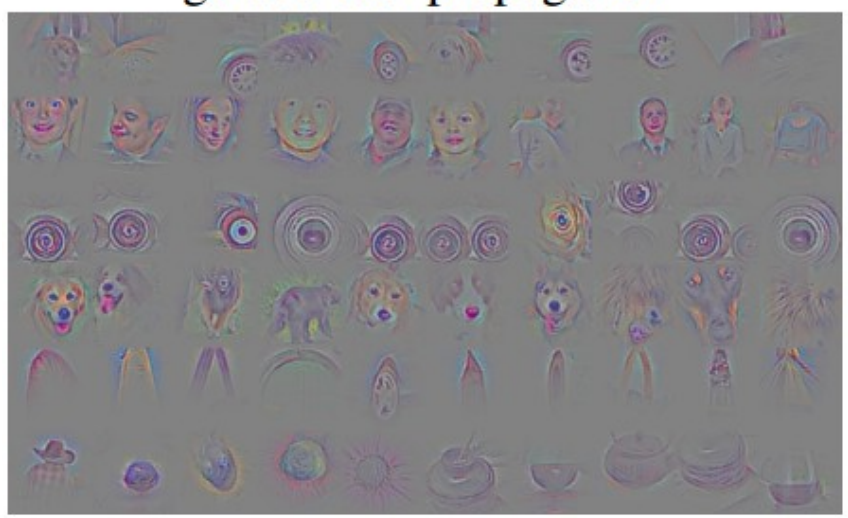
The visualization using "guided backpropagation" is based on the top 10 image patches activating this filter taken from the ImageNet dataset.

guided backpropagation

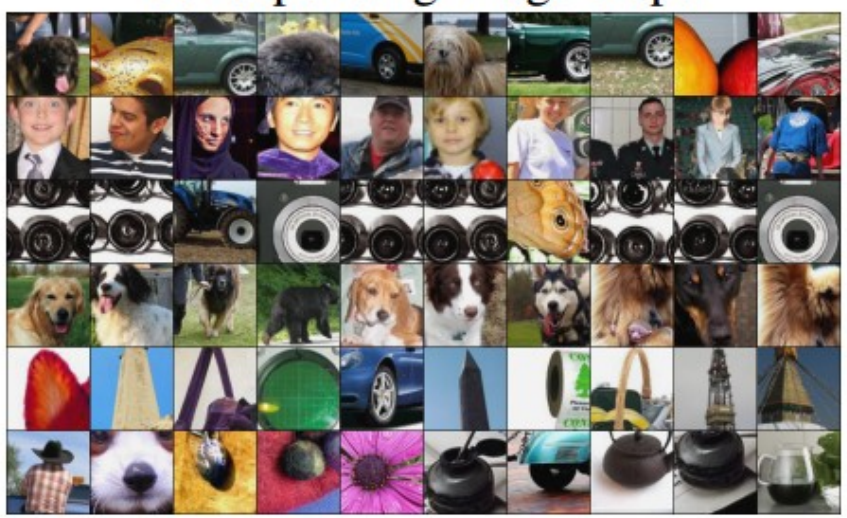


corresponding image crops

guided backpropagation



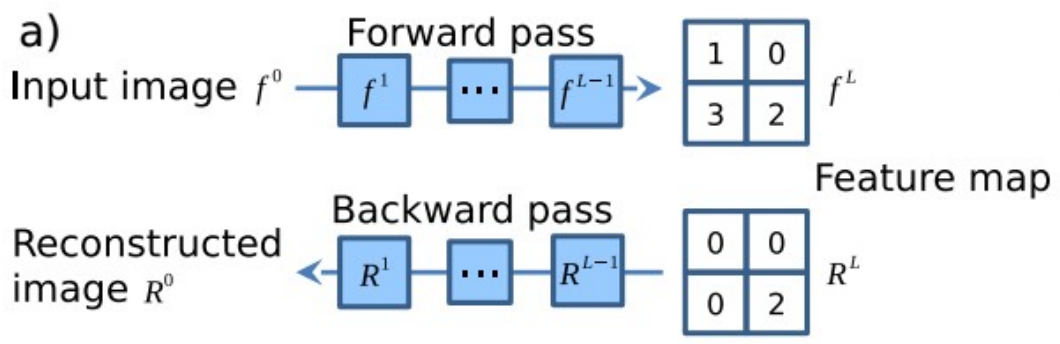
corresponding image crops



[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]

Deconv approaches

[Visualizing and Understanding Convolutional Networks, Zeiler and Fergus 2013]
[Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan et al., 2014]
[Striving for Simplicity: The all convolutional net, Springenberg, Dosovitskiy, et al., 2015]



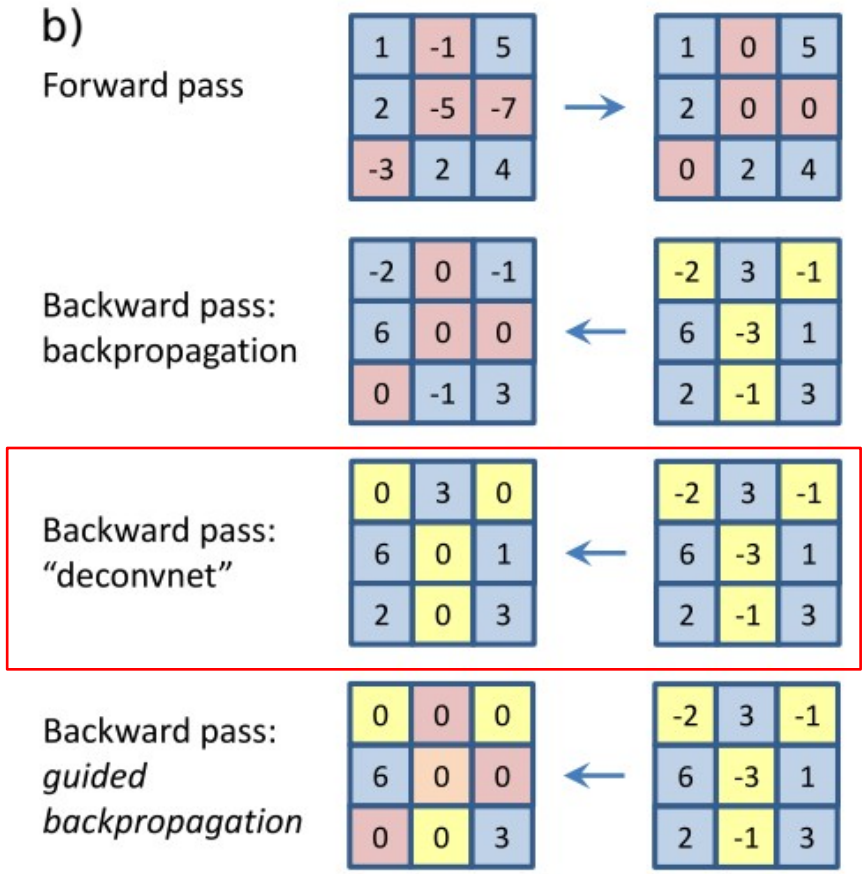
c)

activation: $f_i^{l+1} = relu(f_i^l) = \max(f_i^l, 0)$

backpropagation: $R_i^l = (f_i^l > 0) \cdot R_i^{l+1}$, where $R_i^{l+1} = \frac{\partial f^{out}}{\partial f_i^{l+1}}$

backward 'deconvnet': $R_i^l = (R_i^{l+1} > 0) \cdot R_i^{l+1}$

guided backpropagation: $R_i^l = (f_i^l > 0) \cdot (R_i^{l+1} > 0) \cdot R_i^{l+1}$



bit weird

Universal approximation theorem

From Wikipedia, the free encyclopedia (https://en.wikipedia.org/wiki/Universal_approximation_theorem)

In the mathematical theory of artificial neural networks, the **universal approximation theorem** states[1] that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function. The theorem thus states that simple neural networks can *represent* a wide variety of interesting functions when given appropriate parameters.

Universal approximation theorem

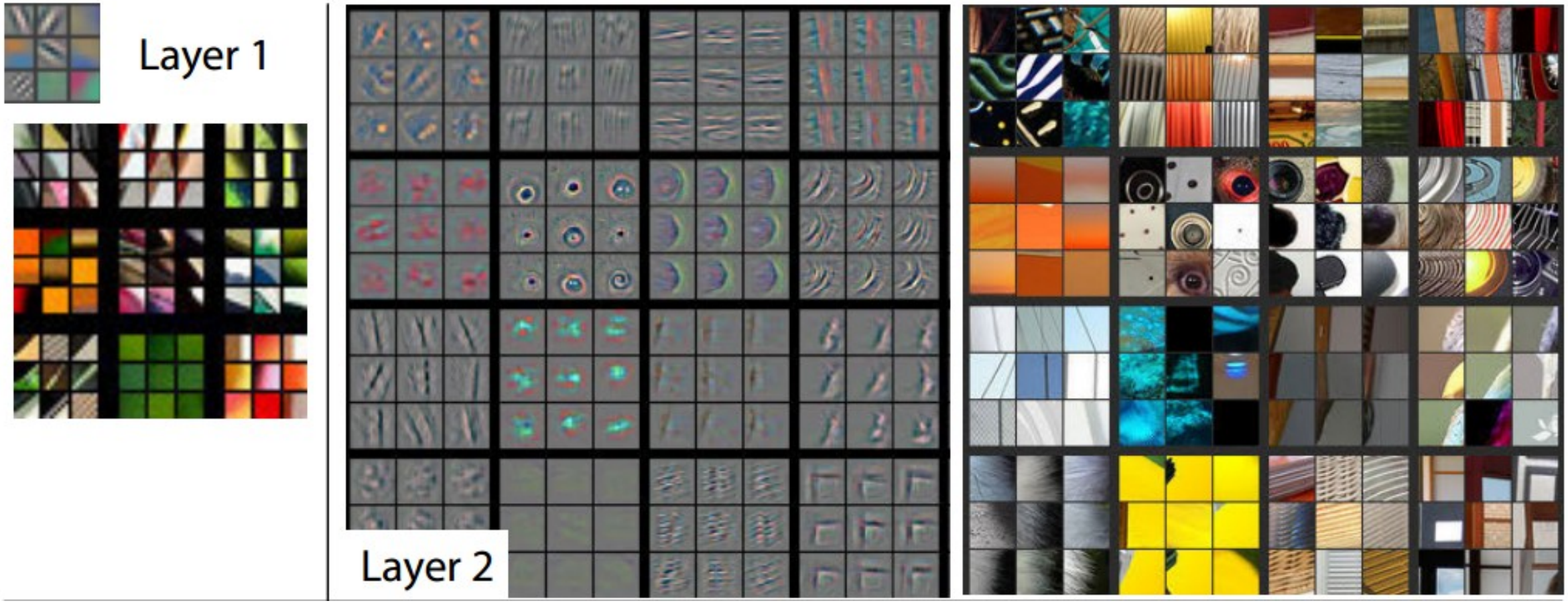
From Wikipedia, the free encyclopedia (https://en.wikipedia.org/wiki/Universal_approximation_theorem)

In the mathematical theory of artificial neural networks, the **universal approximation theorem** states[1] that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbf{R}^n$, under mild assumptions on the activation function. The theorem thus states that simple neural networks can *represent* a wide variety of interesting functions when given appropriate parameters.

Then Why So Deep??

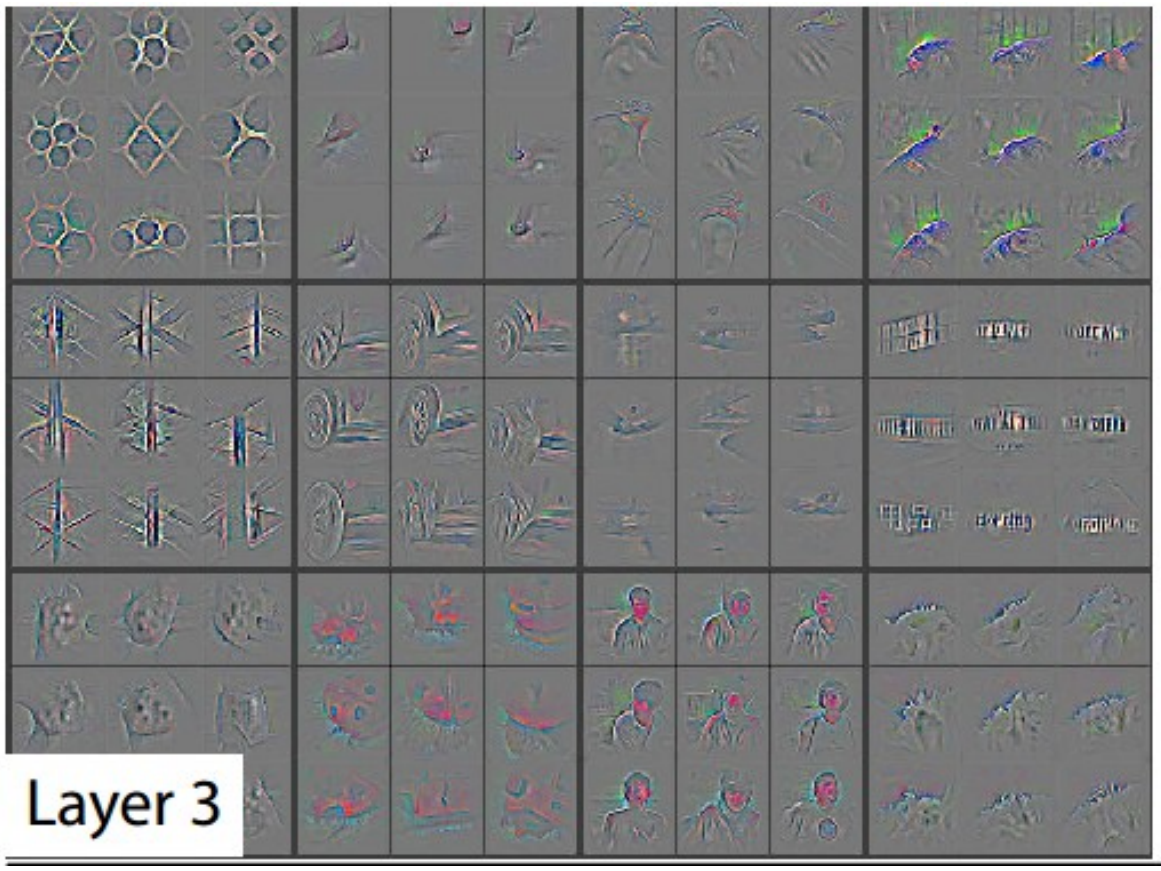
Visualizing and Understanding Convolutional Networks

Visualizing arbitrary neurons along the way to the top...

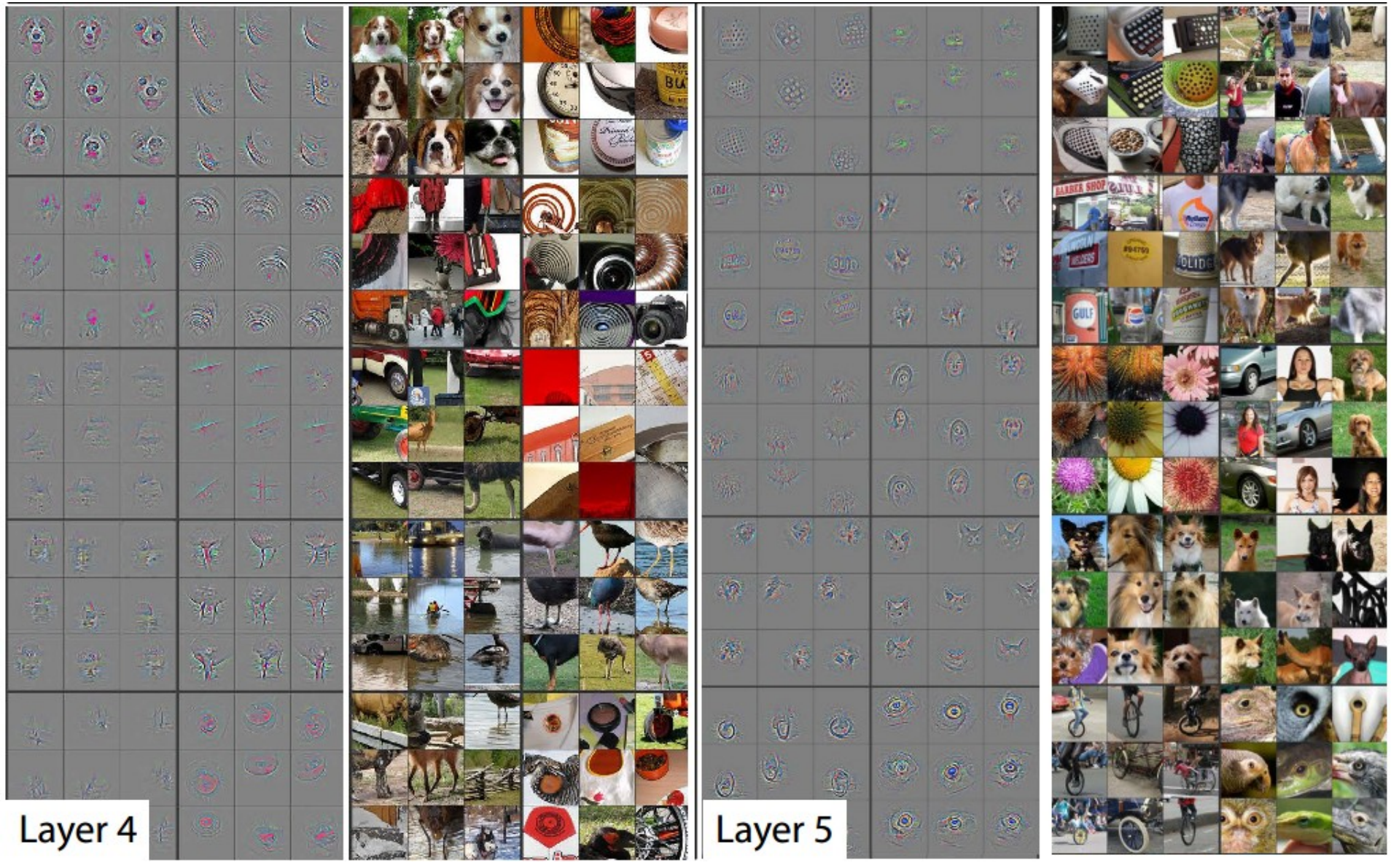


[Zeiler & Fergus, 2013]

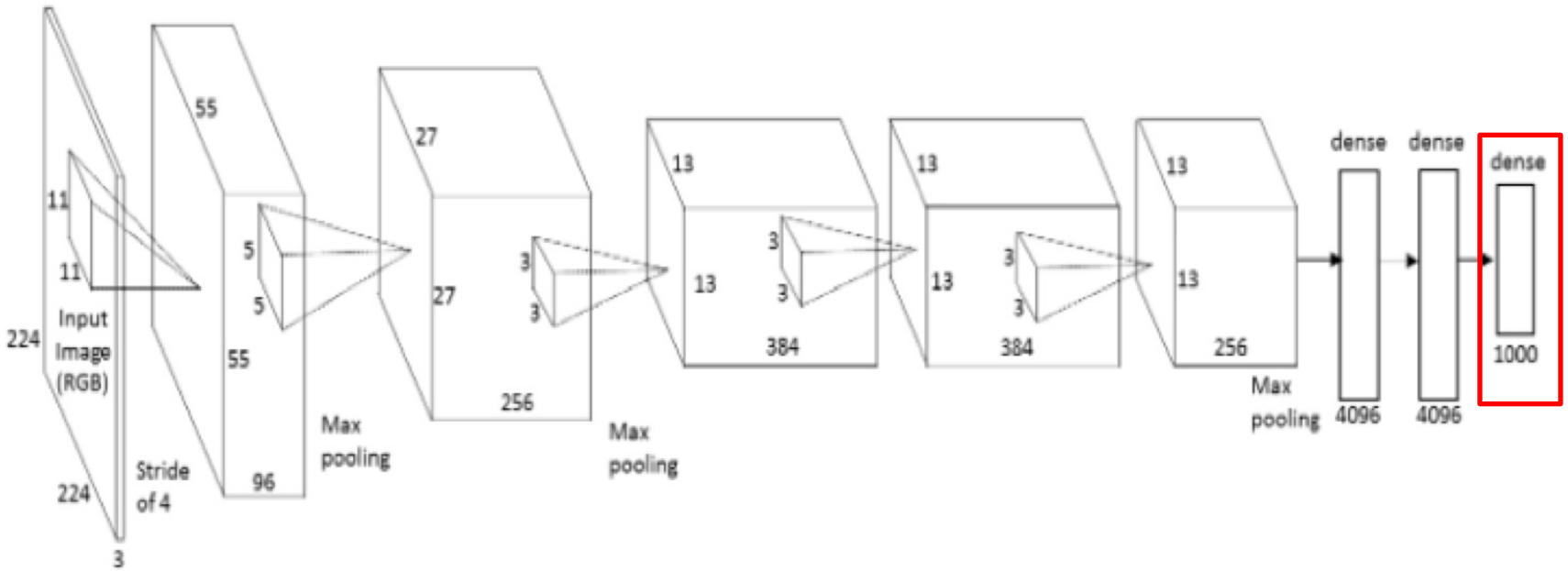
Visualizing arbitrary neurons along the way to the top...



Visualizing arbitrary neurons along the way to the top...



Optimization to Image

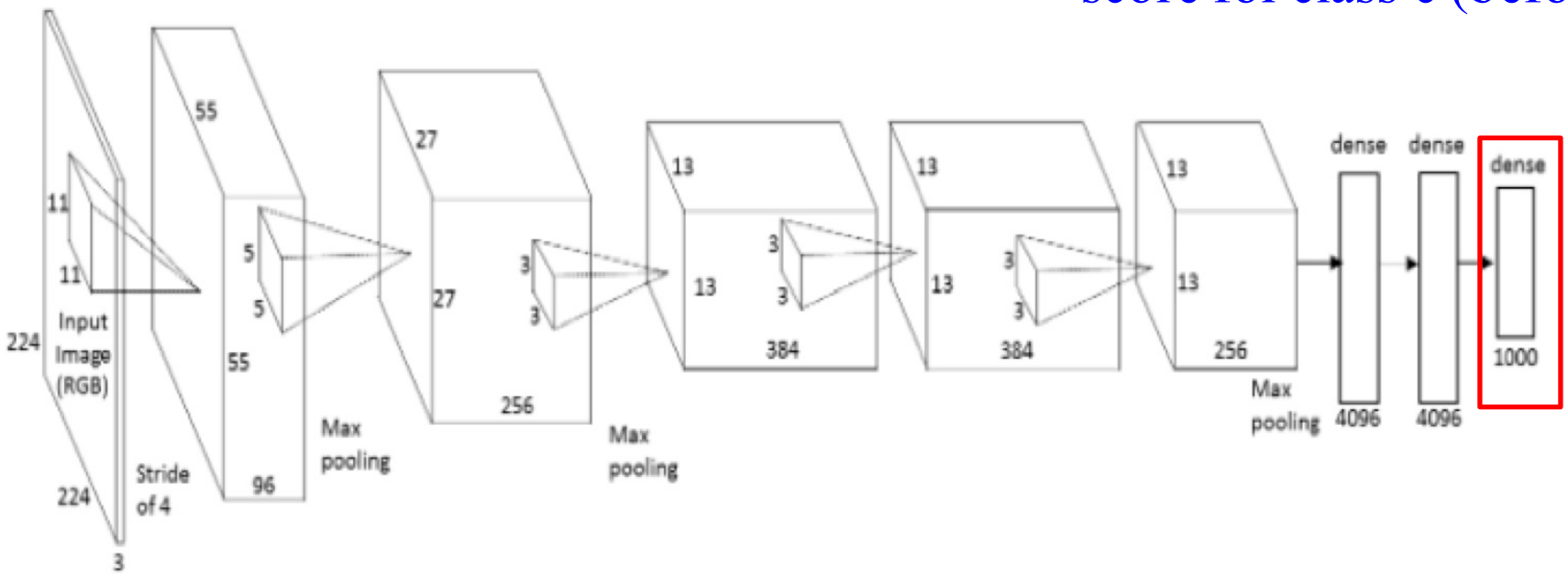


Q: can we find an image that maximizes some class score?

Optimization to Image

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

score for class c (before Softmax)

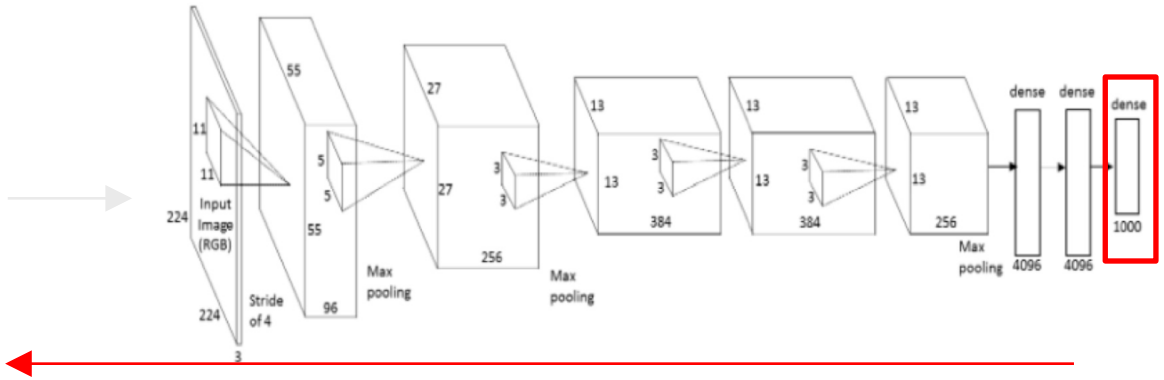


Q: can we find an image that maximizes some class score?

Optimization to Image

1. feed in zeros.

zero image

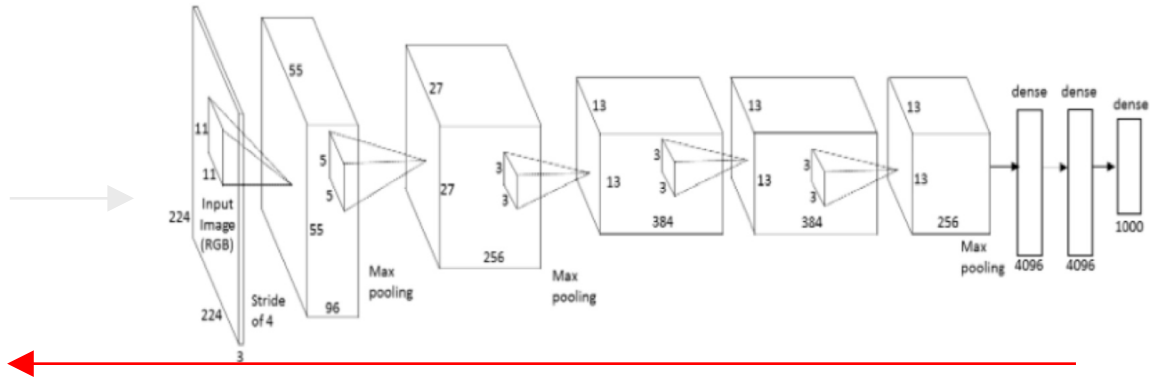


2. set the gradient of the scores vector to be [0,0,....1,....,0], then backprop to image

Optimization to Image

1. feed in zeros.

zero image



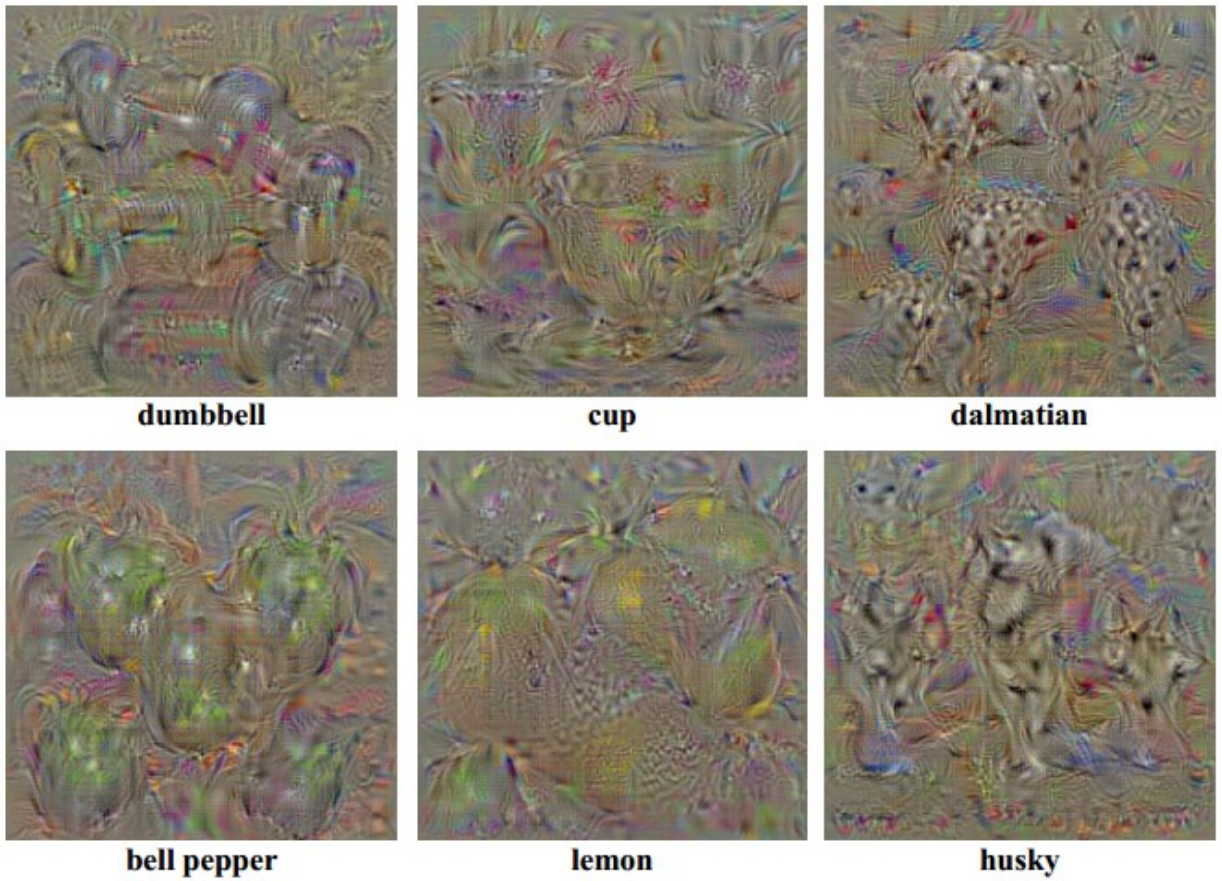
2. set the gradient of the scores vector to be [0,0,....1,....,0], then backprop to image
3. do a small "image update"
4. forward the image through the network.
5. go back to 2.

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

score for class c (before Softmax)

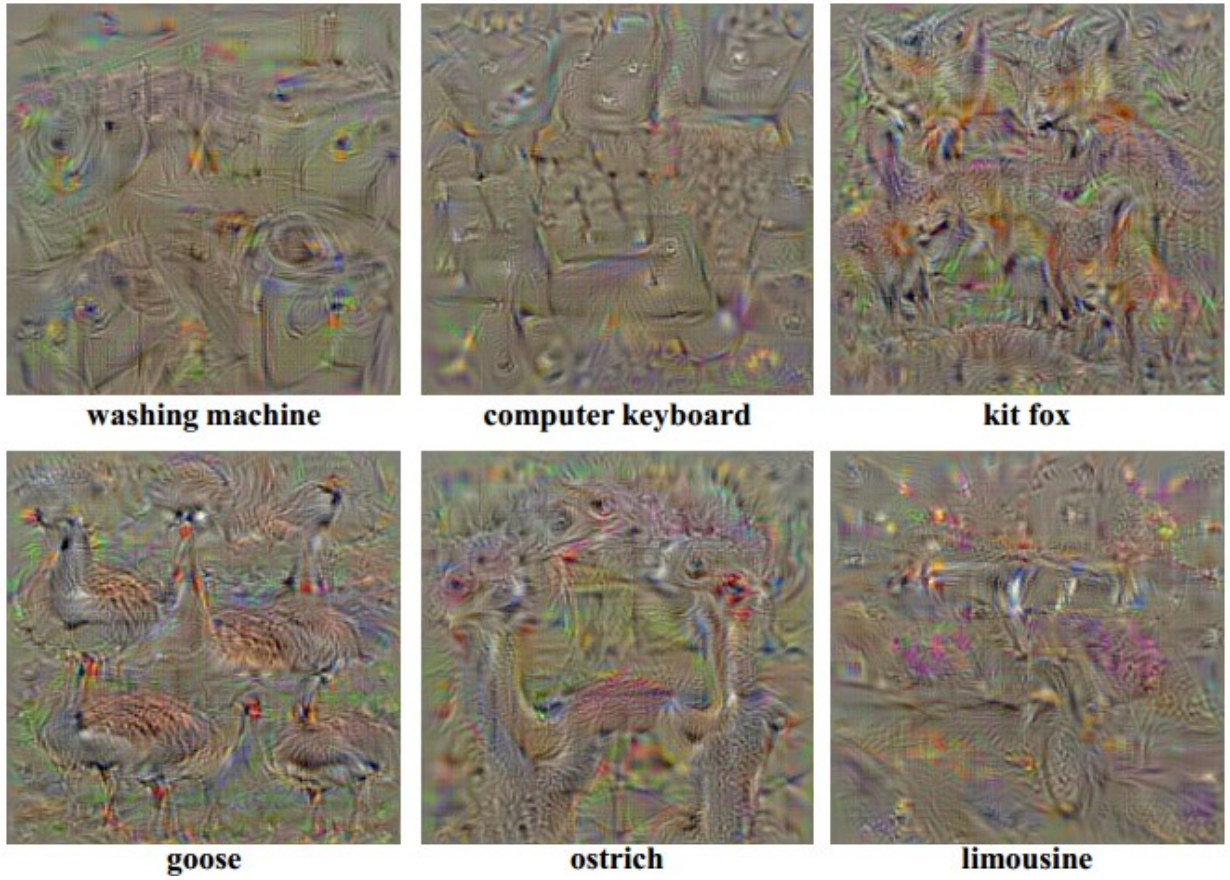
Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

1. Find images that maximize some class score:



Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

1. Find images that maximize some class score:



[Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015]

Proposed a different form of regularizing the image

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

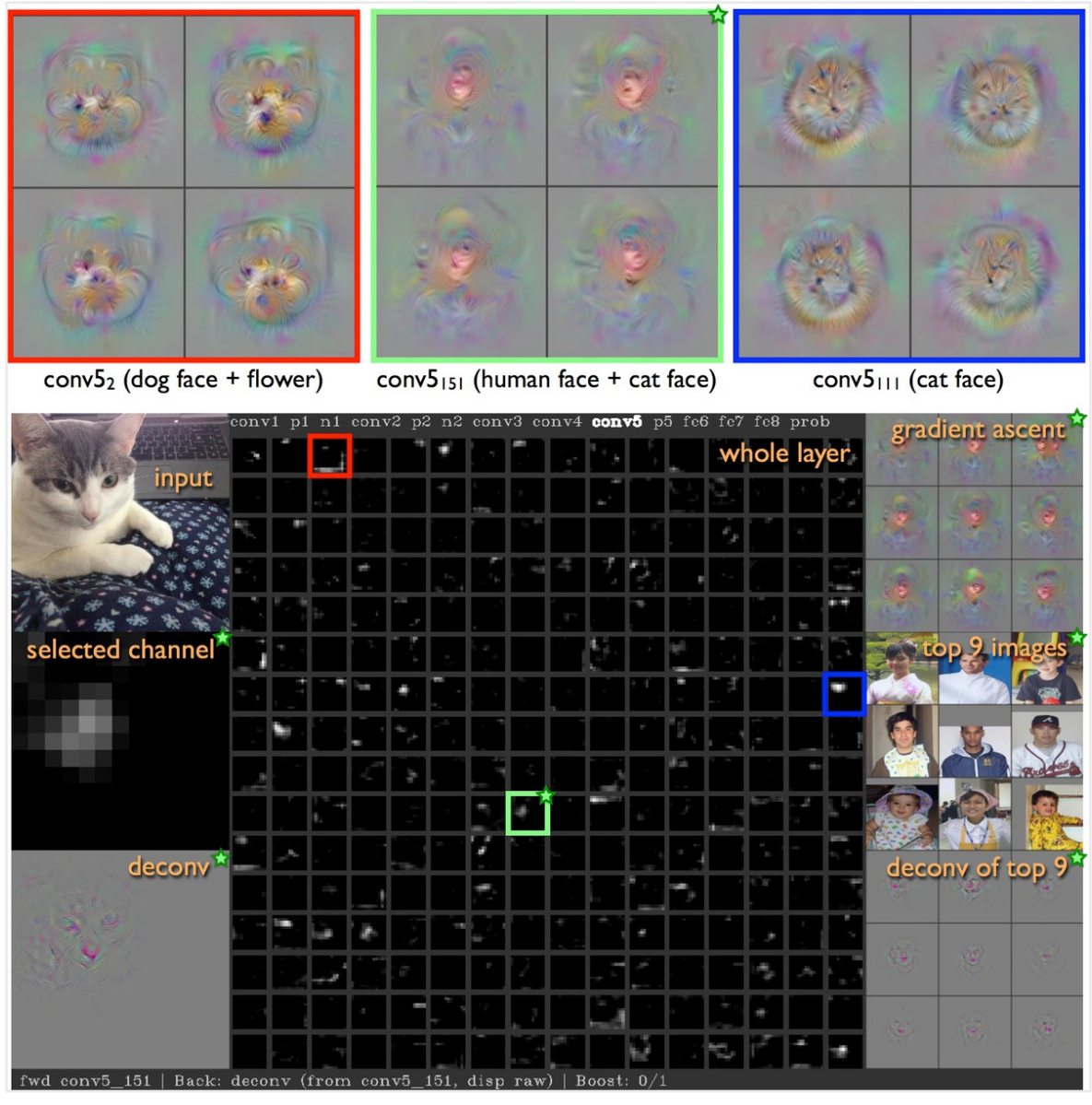
More explicit scheme:

Repeat:

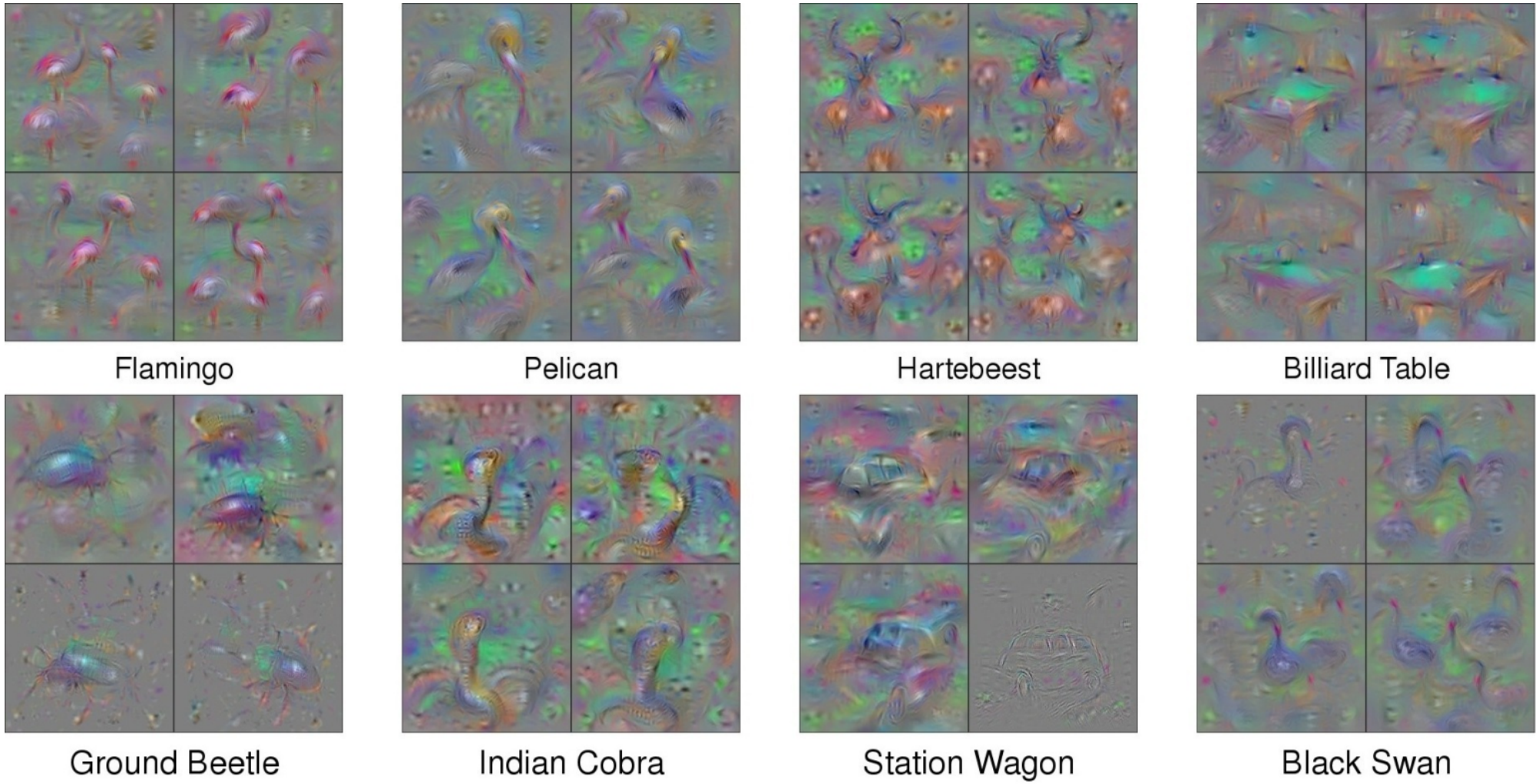
- Update the image **x** with gradient from some unit of interest
- Blur x a bit
- Take any pixel with small norm to zero (to encourage sparsity)

http://yosinski.com/deepvis

YouTube video
https://www.youtube.com/watch?v=AgkfIQ4IGaM
(4min)

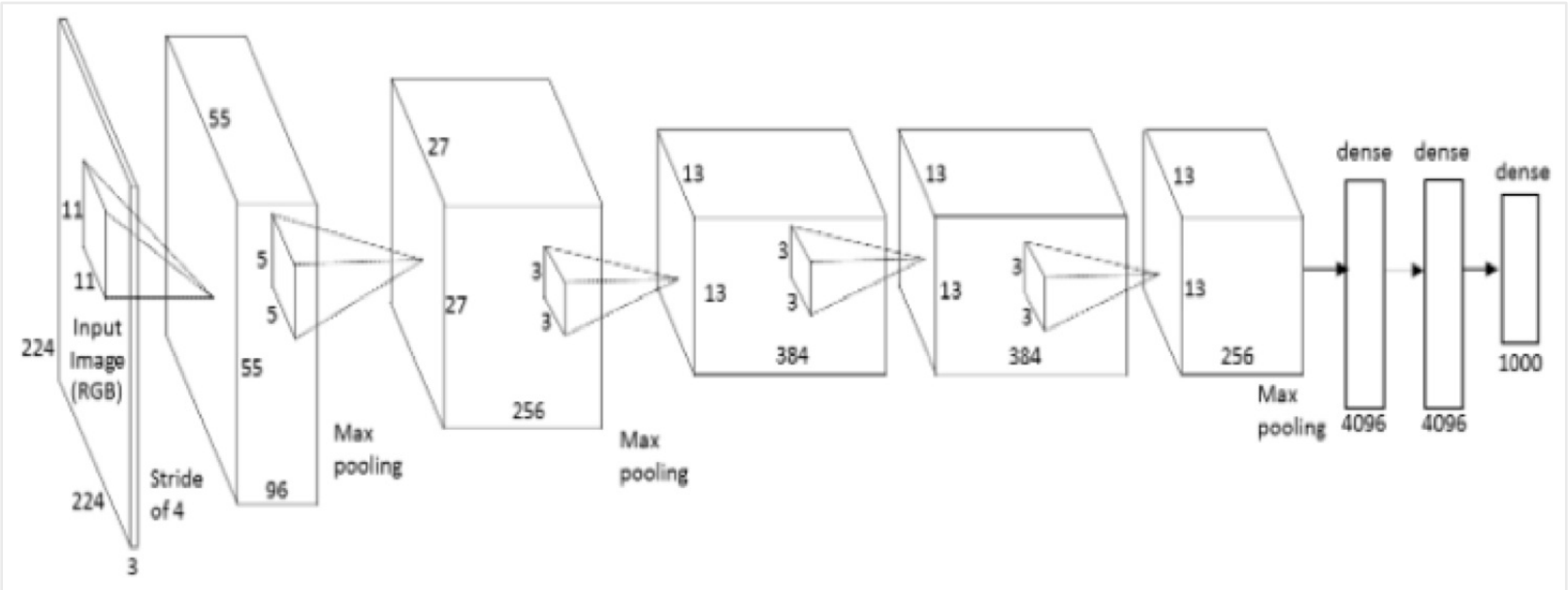


[Understanding Neural Networks Through Deep Visualization, Yosinski et al. , 2015]



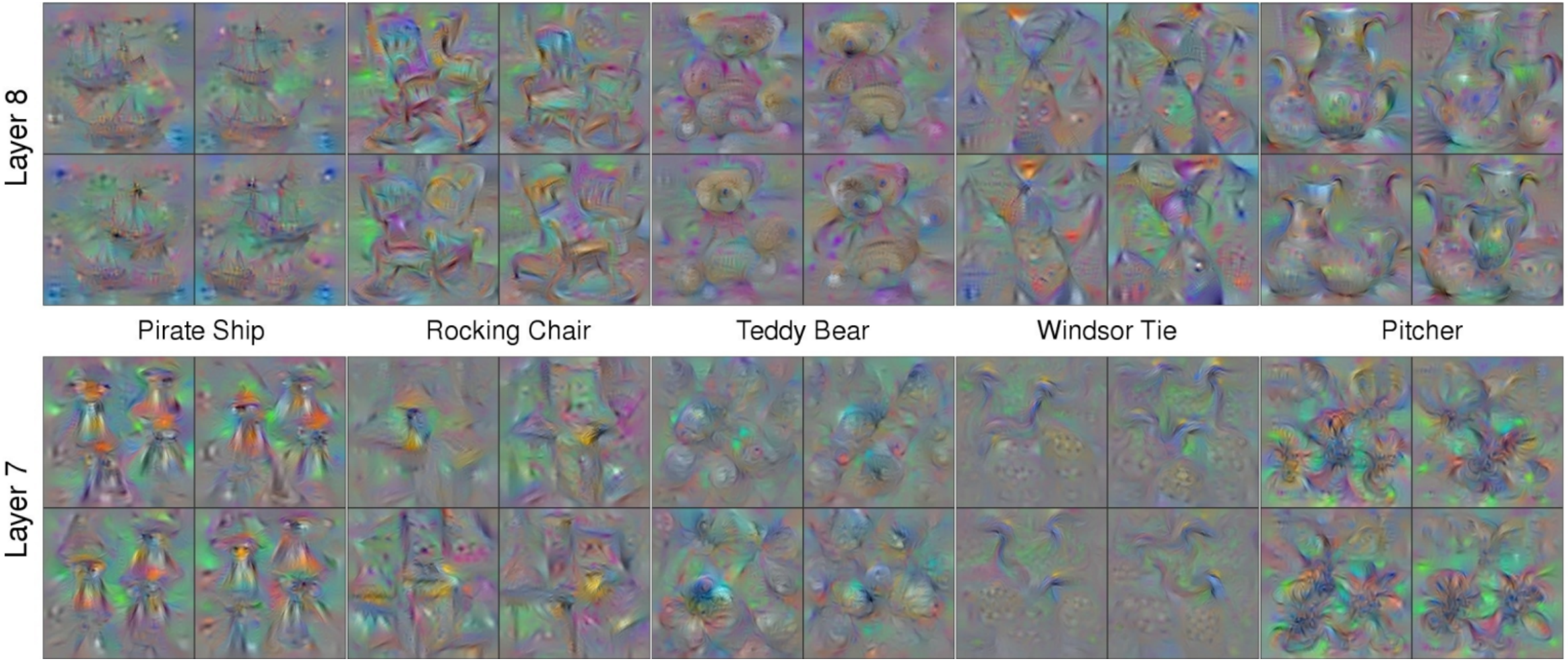
http://yosinski.com/deepvis

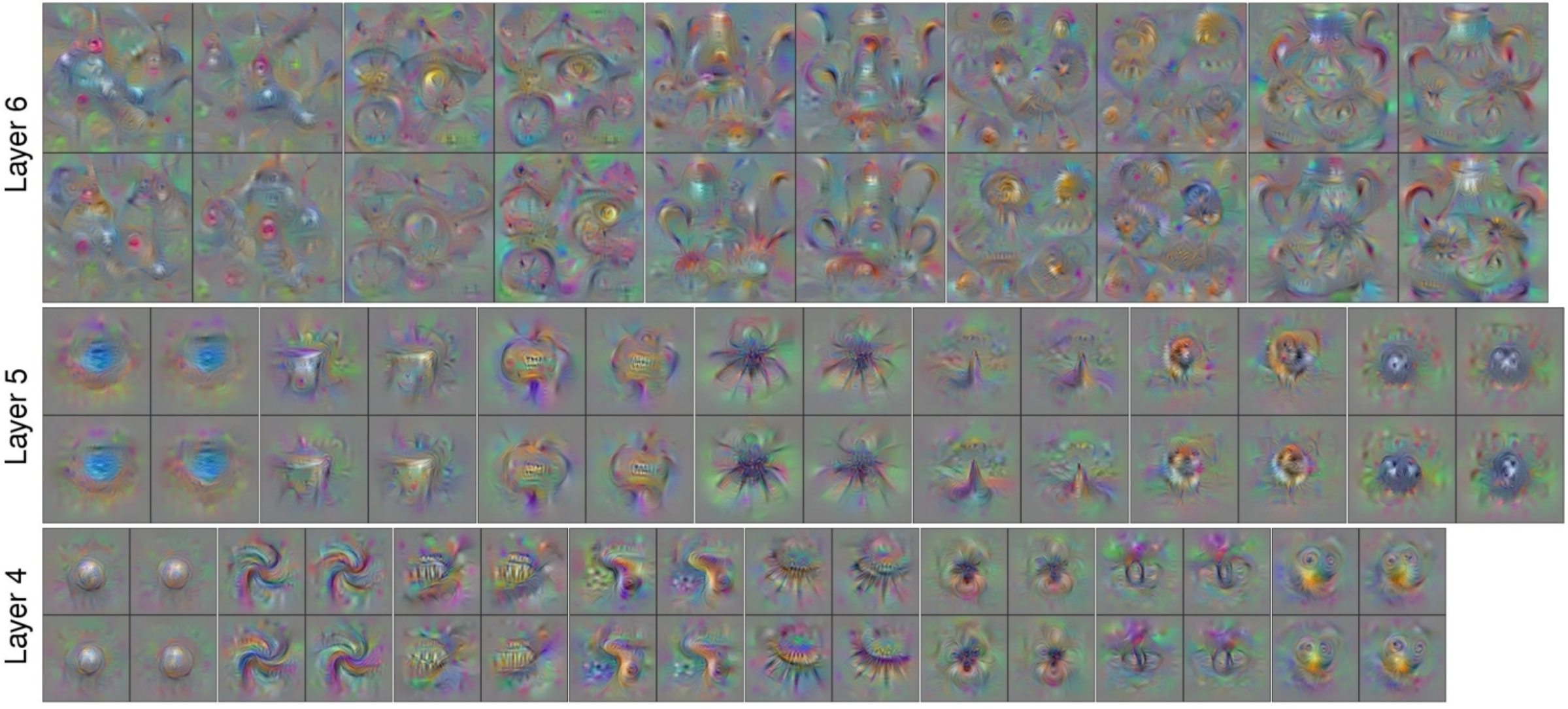
We can in fact do this for arbitrary neurons along the ConvNet

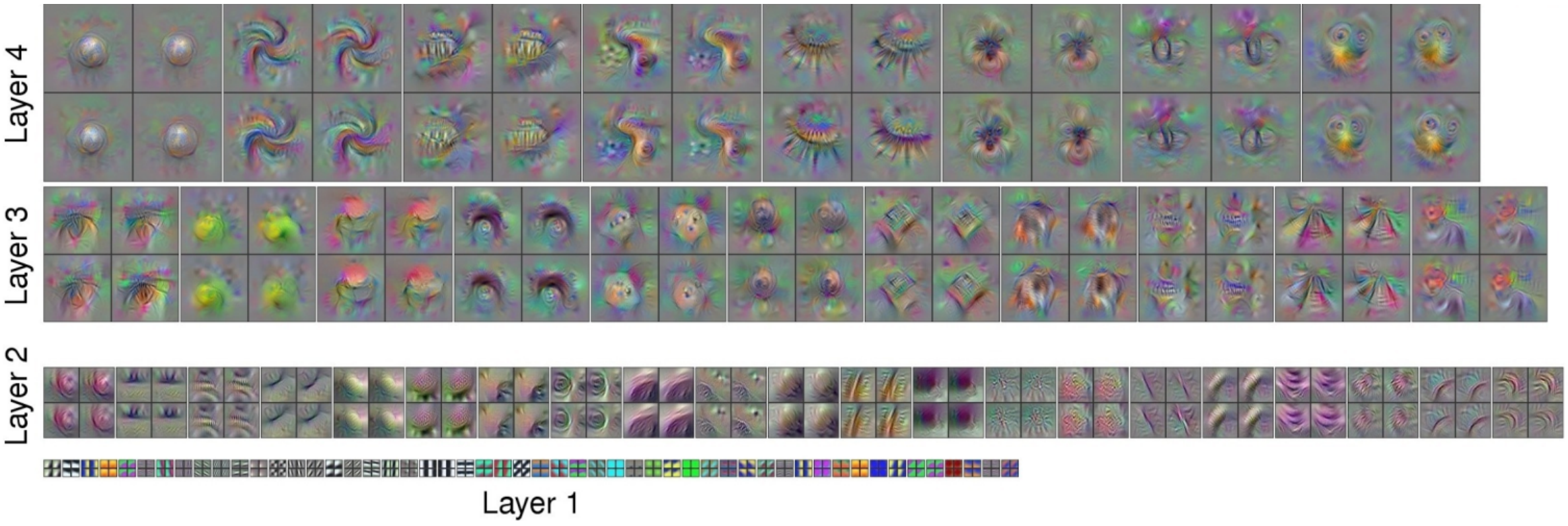


Repeat:

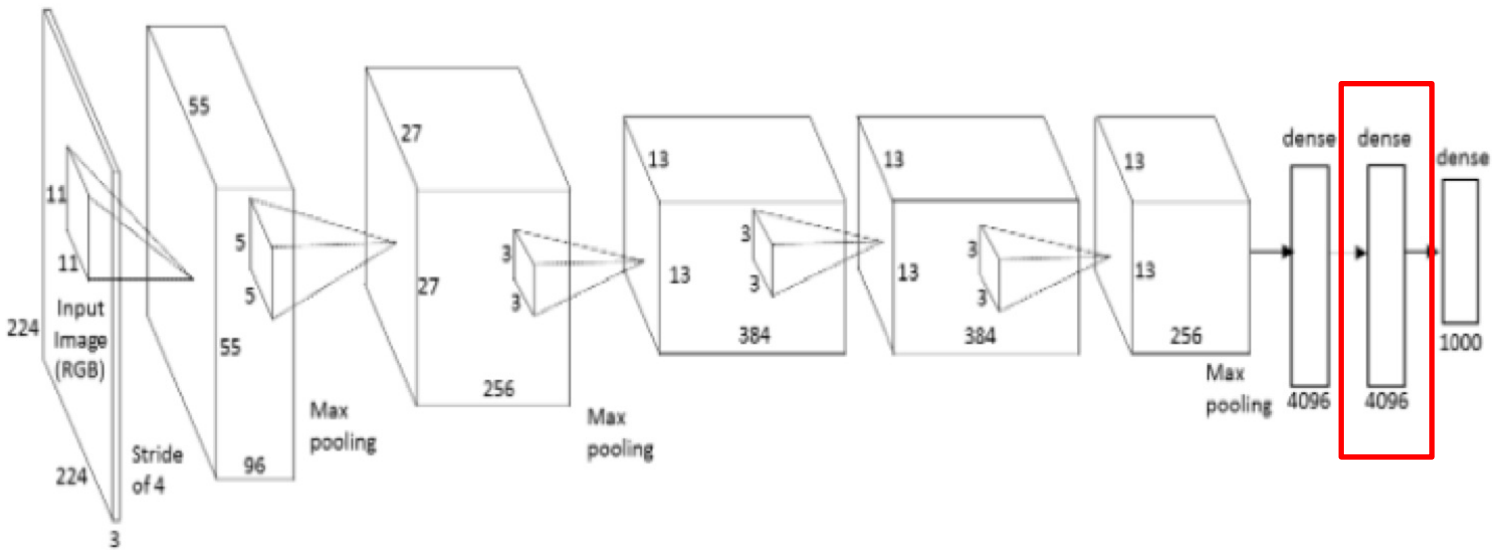
1. Forward an image
2. Set activations in layer of interest to all zero, except for a 1.0 for a neuron of interest
3. Backprop to image
4. Do an “image update”


Layer 8
Pirate Ship
Rocking Chair
Teddy Bear
Windsor Tie
Pitcher
Layer 7


Layer 6
Layer 5
Layer 4


Layer 4
Layer 3
Layer 2
Layer 1

Question: Given a CNN code, is it possible to reconstruct the original image?



Find an image such that:

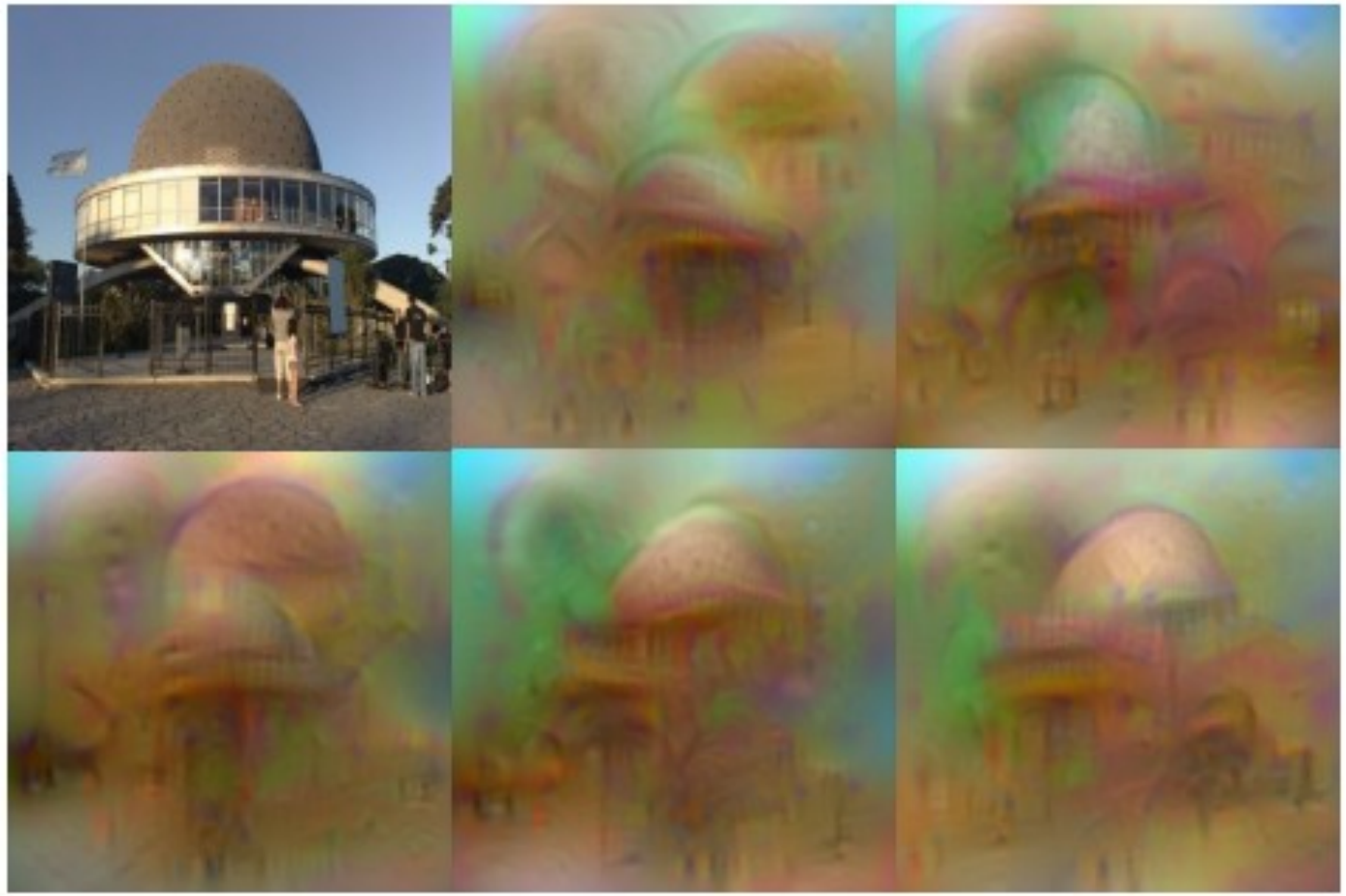
- Its code is similar to a given code
- It “looks natural” (image prior regularization)

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

Understanding Deep Image Representations by Inverting Them
[Mahendran and Vedaldi, 2014]

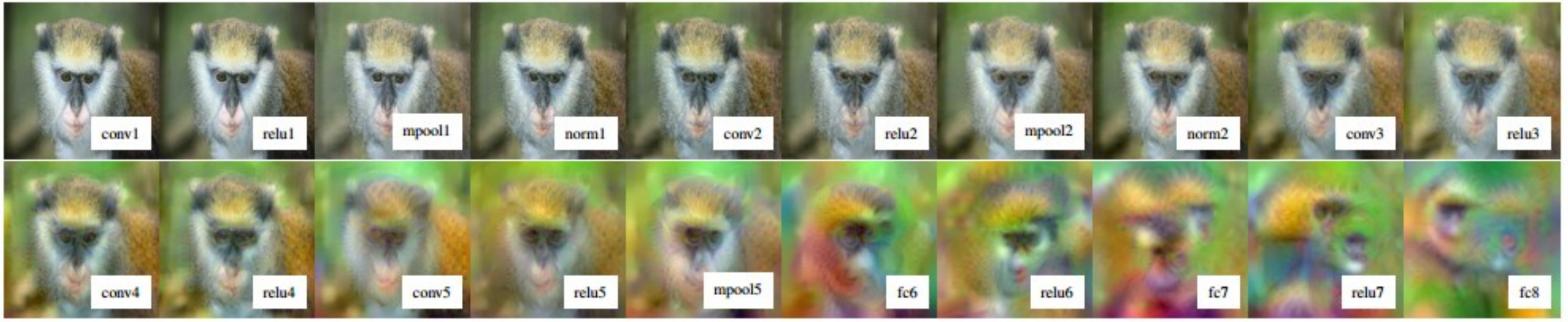
original image



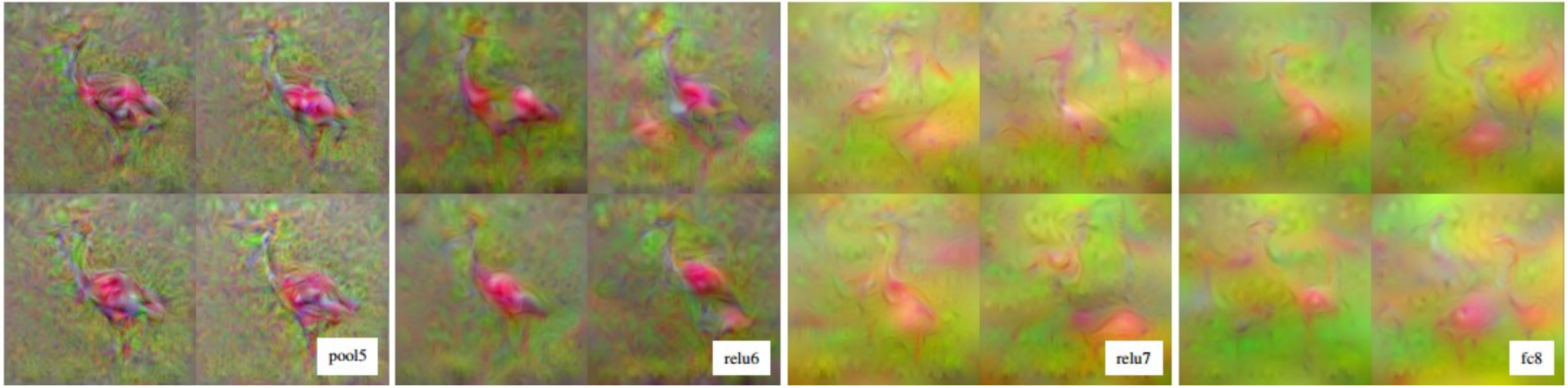
reconstructions from the 1000 log probabilities for ImageNet (ILSVRC) classes

Reconstructions from the representation after last last pooling layer (immediately before the first Fully Connected layer)

Reconstructions from intermediate layers



Multiple reconstructions. Images in quadrants all "look" the same to the CNN (same code)



Thank you!