L4: Convolutional Neural Networks Understanding and Visualization

Xiaopeng HONG

CMV, OU

Acknowledgement

- Most Slides are from or inspired by
  - Rob Fergus
  - Feifei Li & Andrei Karpathy, and
  - Marc'Aurelio Ranzato's talks

Outline

- CNN: a Revisit
- An Insight into CNN
- Visualization

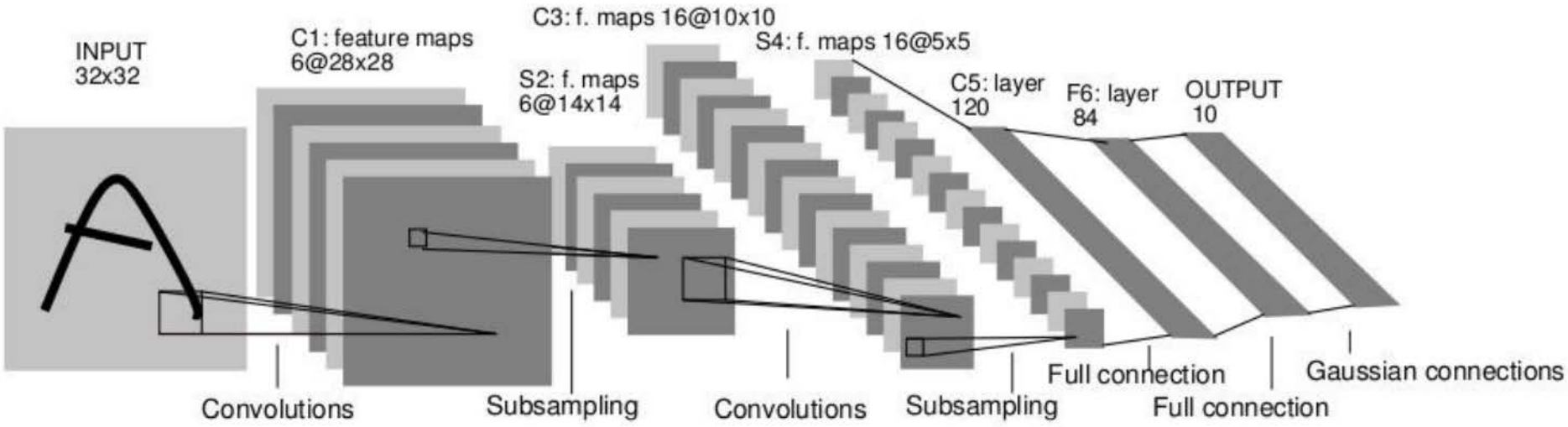
Basic features

- Supervised
- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Train convolutional filters by back-propagating classification error

Basic features

- Supervised
- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Train convolutional filters by back-propagating classification error
- Convolve input
  - Connect each hidden unit to a small patch of the input
  - Share the weight across hidden units

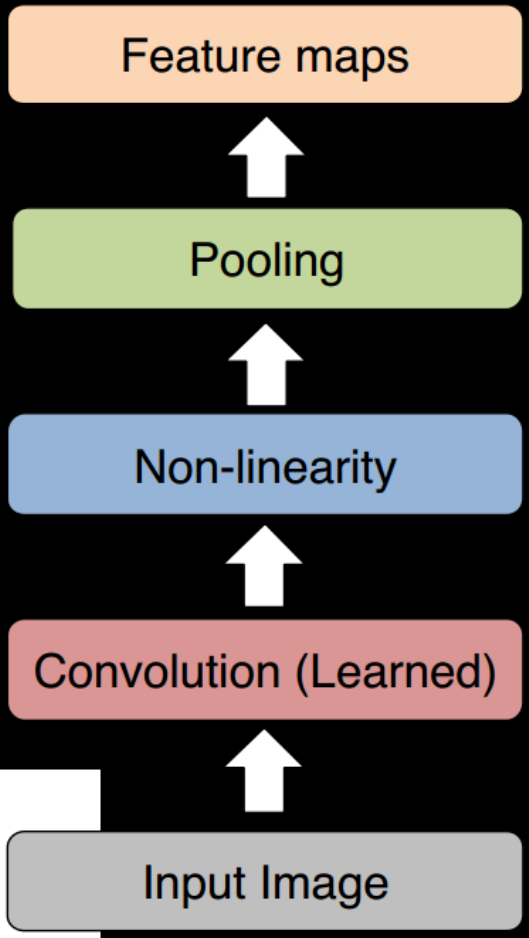
Convolutional Neural Networks

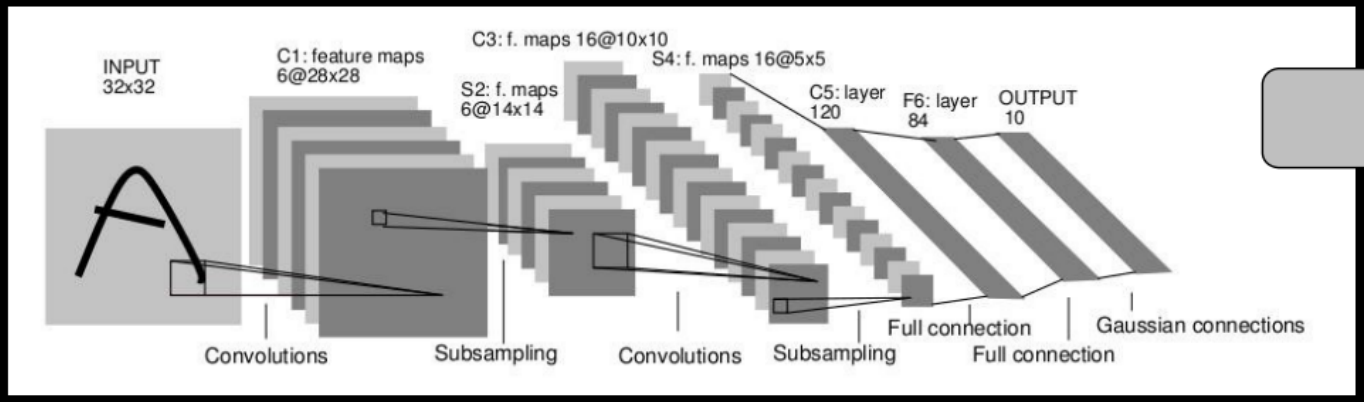


LeCun et al. 1989

Recap of Convnets

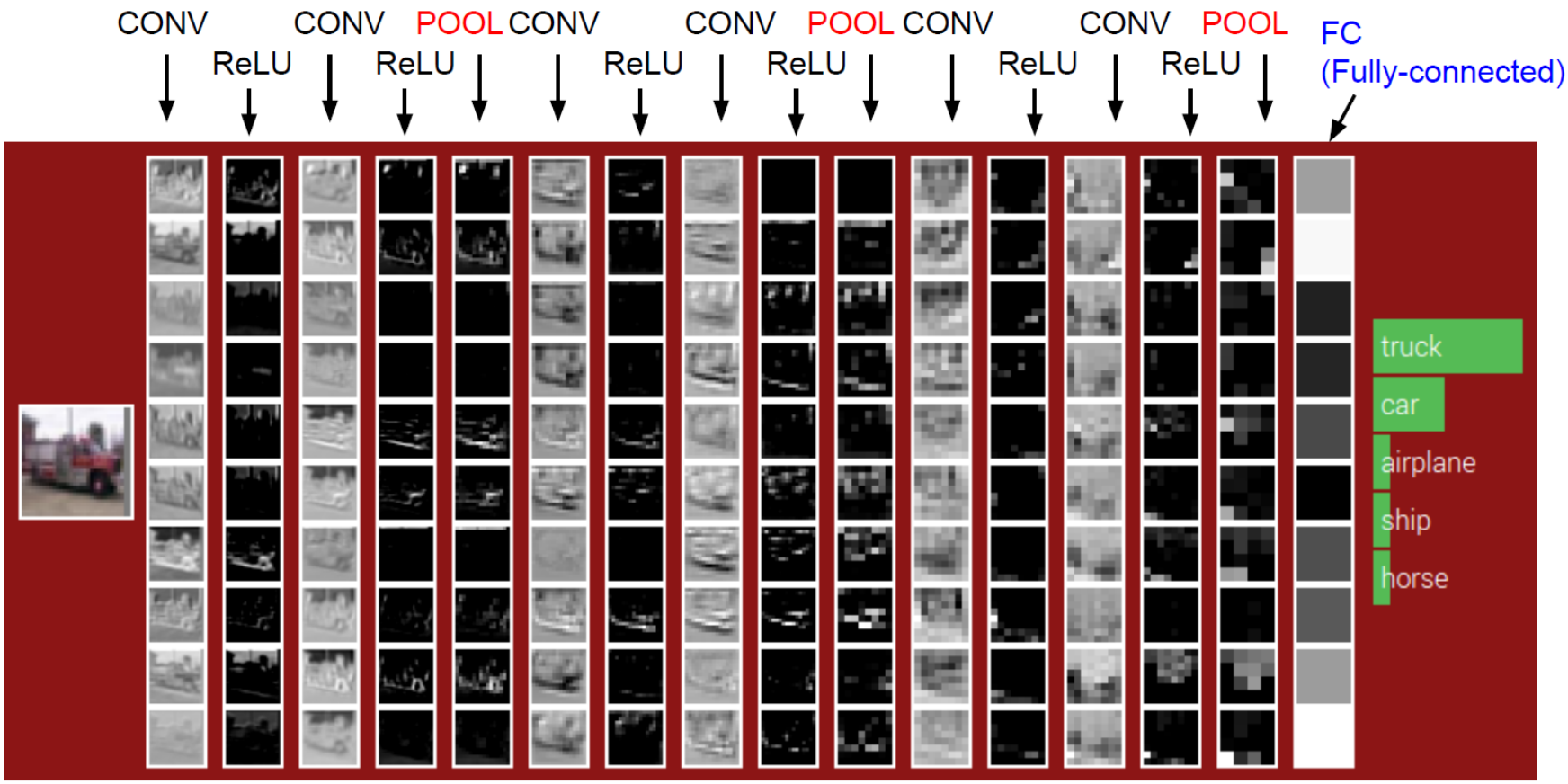
- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised
- Train convolutional filters by back-propagating classification error





[LeCun et al. 1989]

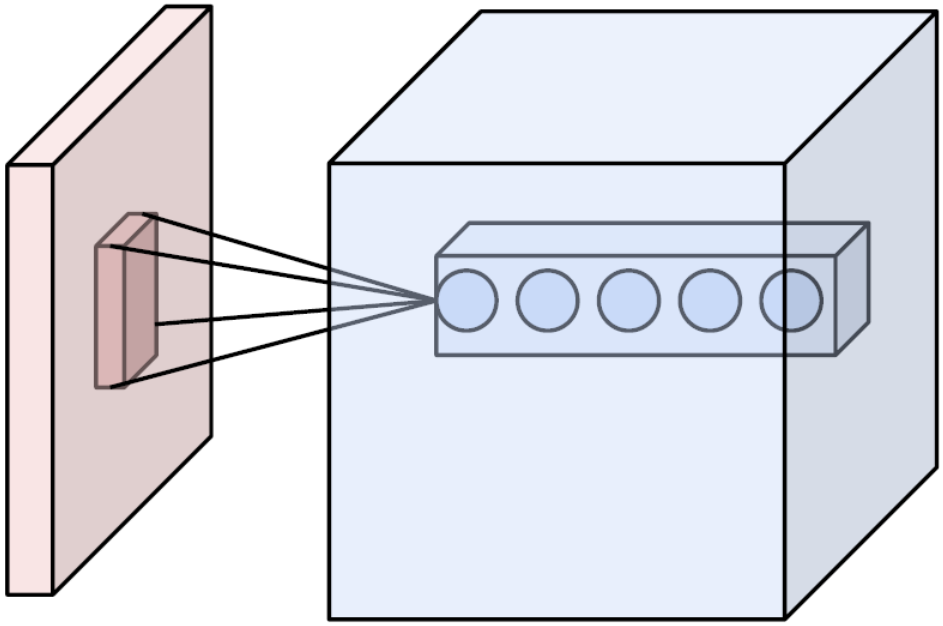
Some modern structure



A grammar-like CNN description

- [CONV-RELU-POOL]xN,[FC-RELU]xM, FC, SOFTMAX
- [CONV-RELU-CONV-RELU-POOL]xN,[FC-RELU]xM,FC,SOFTMAX
  - N >= 0, M >=0
  - last FC layer should not have RELU - these are the class scores

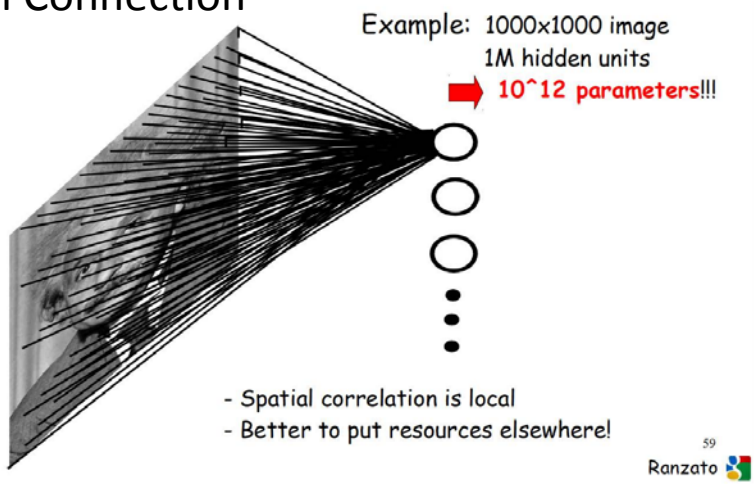
One CNN layer



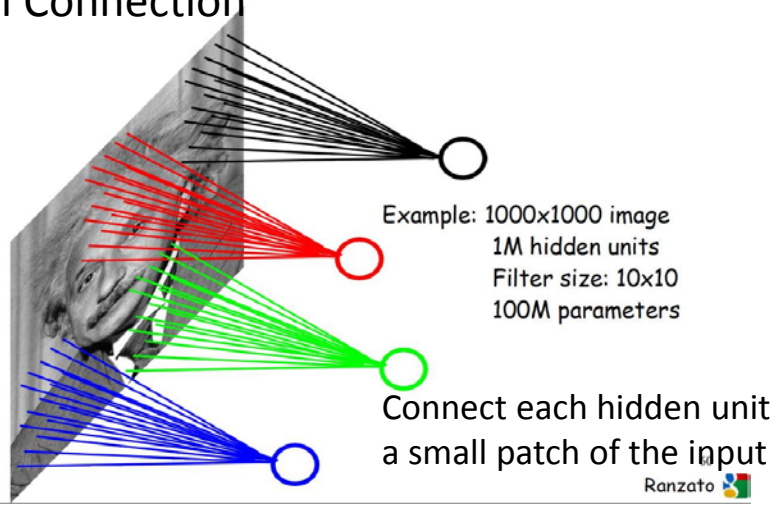
Just like normal Hidden Layer BUT:

- Connect neurons to the input in a **local receptive field**
- All neurons in a single depth slice **share weights**

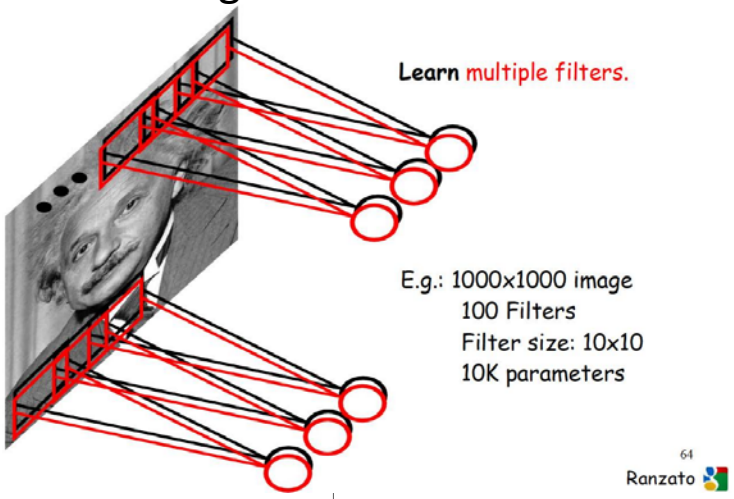
Full Connection

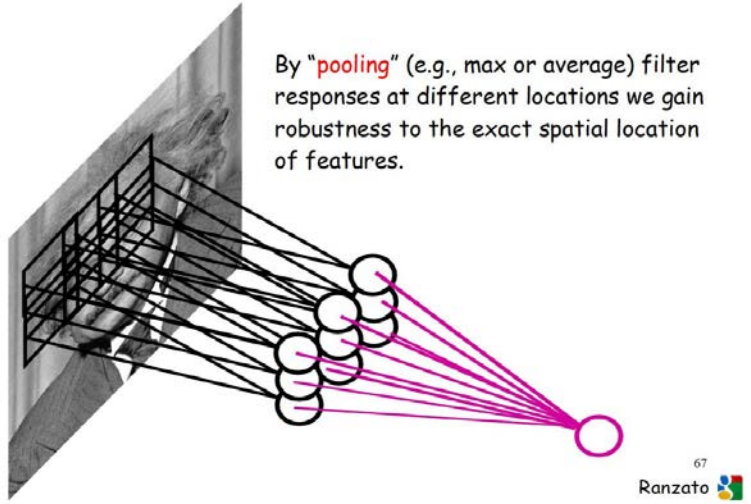


Local Connection



Share the weight across hidden units

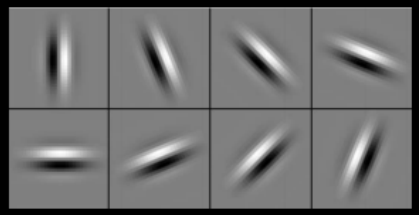




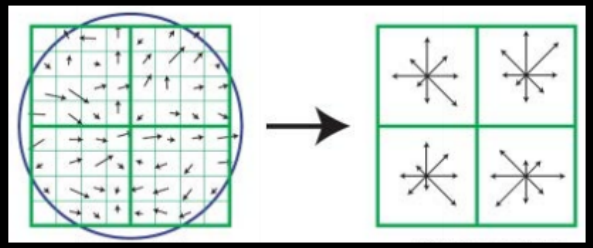
Compare: SIFT Descriptor

Image Pixels

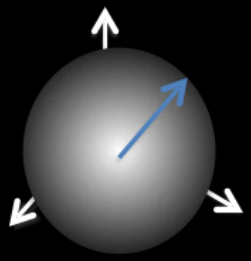
Apply Gabor filters



Spatial pool (Sum)



Normalize to unit length

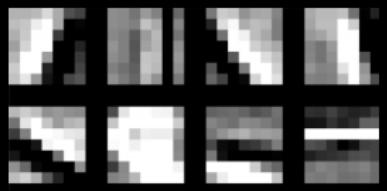


Feature Vector

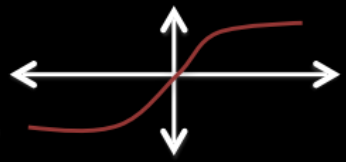
Components of Each Layer

Pixels / Features

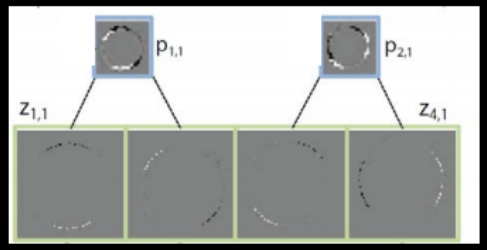
Filter with Dictionary (convolutional or tiled)



+ Non-linearity



Spatial/Feature (Sum or Max)



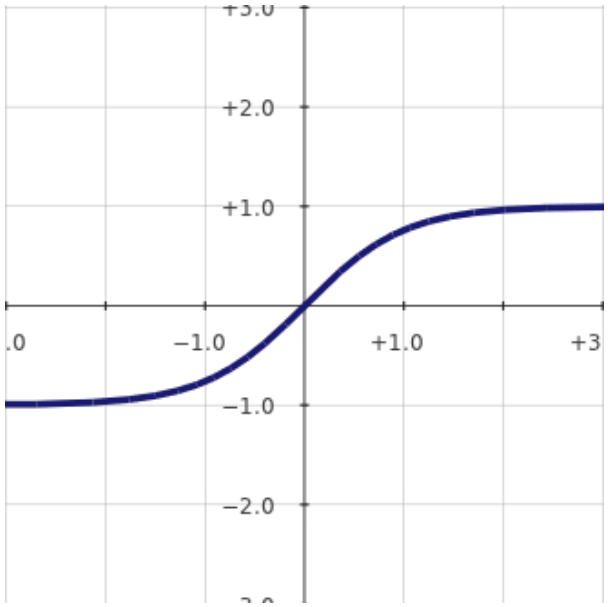
[Optional]

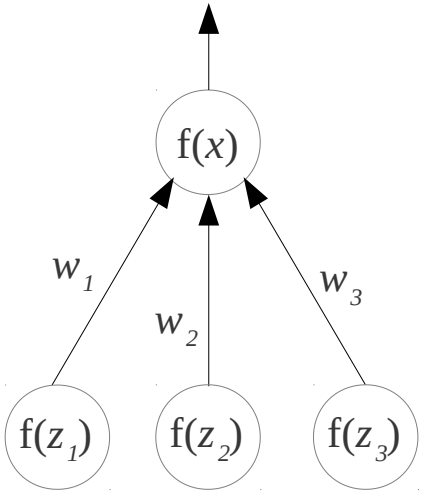
Normalization between feature responses

Output Features

Non Linearity

tanh(x)

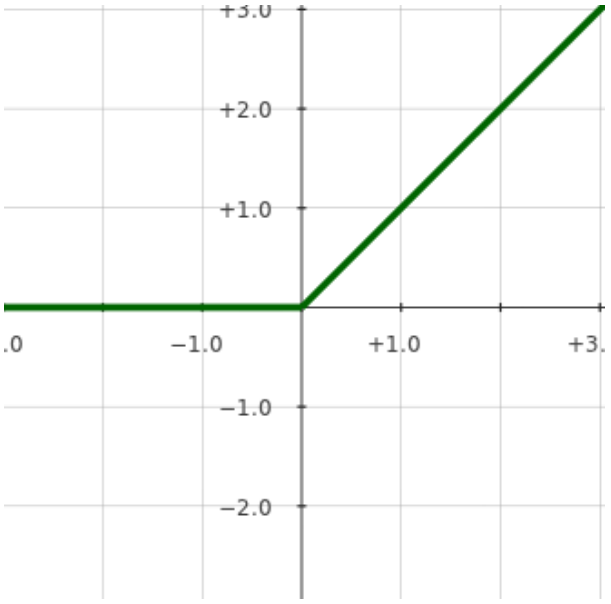




$x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3)$

x is called the total input to the neuron, and $f(x)$ is its output

f(x) = max(0, x)



Slow to train

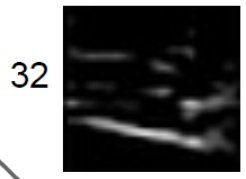
Saturation Issue

Simple

Efficient

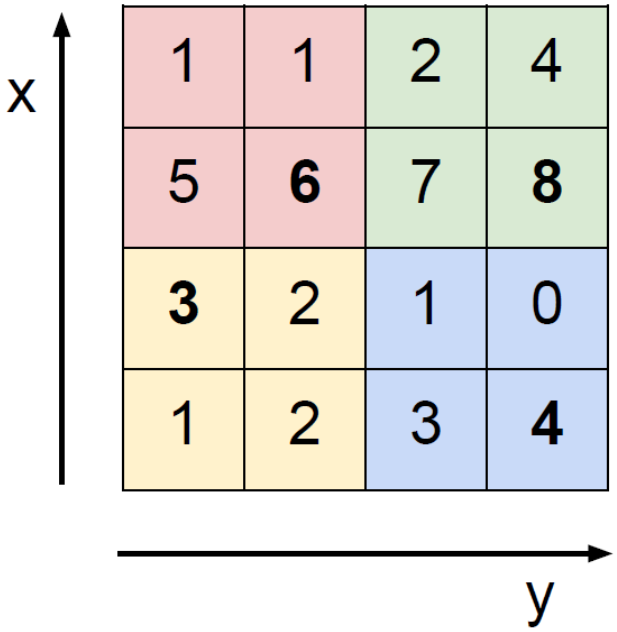
Avoid the Saturation Issue

Max Pooling Layer



downsampling

Single depth slice



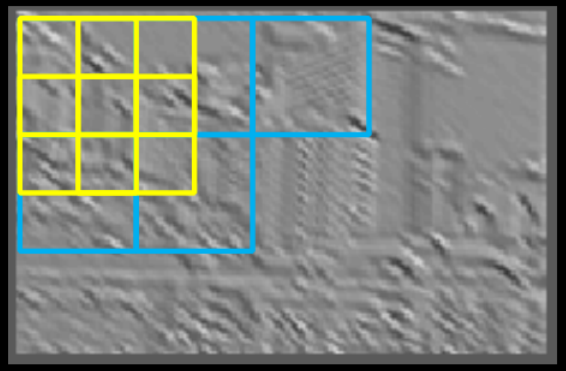
max pool with 2x2 filters and stride 2

6	8
3	4

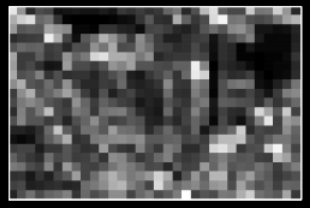
Pooling layer downsamples every activation map in the input independently with max.

Pooling

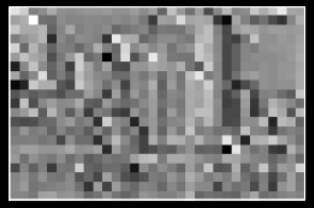
- Spatial Pooling
  - Non-overlapping / overlapping regions
  - Sum or max
  - Boureau et al. ICML'10 for theoretical analysis



Max



Sum



Modern CNN trend toward:

- - Small filter sizes (3x3 and less)
- - Small pooling sizes (2x2 and less)
- - Small strides (stride = 1, ideally)
- - Deep
- - Conv Layers should **pad with zeros** to not reduce spatial size
- - Pool Layers should reduce size once in a while
- - Eventually Fully-Connected Layers take over

(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

Most memory is in early CONV

Most params. are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

When SVM meets CNN

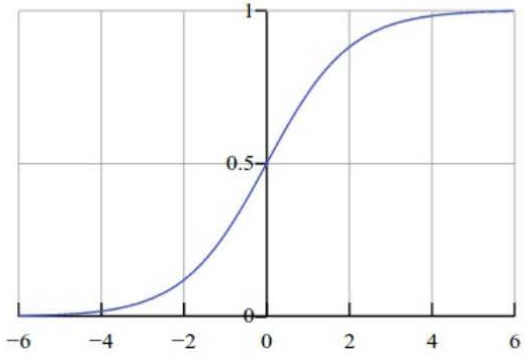
Outline

- CNN: a Revisit
- An Insight into CNN
- Visualization

Let's attack a binary linear classifier

Lets fool a binary linear classifier:
(logistic regression)

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$



Since the probabilities of class 1 and 0 sum to one, the probability for class 0 is $P(y=0 \mid x; w, b) = 1 - P(y=1 \mid x; w, b)$. Hence, an example is classified as a positive example (y = 1) if $\sigma(w^T x + b) > 0.5$, or equivalently if the score $w^T x + b > 0$.

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1	← input example
w	-1	-1	1	-1	1	-1	1	1	-1	1	← weights

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1

← input example

← weights

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1	← input example
w	-1	-1	1	-1	1	-1	1	1	-1	1	← weights
adversarial x	?	?	?	?	?	?	?	?	?	?	

class 1 score = dot product:
= -2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3
=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474
i.e. the classifier is **95%** certain that this is class 0 example.

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5

← input example

← weights

class 1 score before:

-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3

=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2

=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88

i.e. we improved the class 1 probability from 5% to 88%

$$P(y=1 \mid x; w, b) = \frac{1}{1+e^{-(w^T x+b)}} = \sigma(w^T x + b)$$

Lets fool a binary linear classifier:

x	2	-1	3	-2	2	2	1	-4	5	1
w	-1	-1	1	-1	1	-1	1	1	-1	1
adversarial x	1.5	-1.5	3.5	-2.5	2.5	1.5	1.5	-3.5	4.5	1.5

← input example

← weights

class 1 score before:

-2 + 1 + 3 + 2 + 2 - 2 + 1 - 4 - 5 + 1 = -3

=> probability of class 1 is 1/(1+e^(-(-3))) = 0.0474

-1.5+1.5+3.5+2.5+2.5-1.5+1.5-3.5-4.5+1.5 = 2

=> probability of class 1 is now 1/(1+e^(-(2))) = 0.88

i.e. we improved the class 1 probability from 5% to 88%

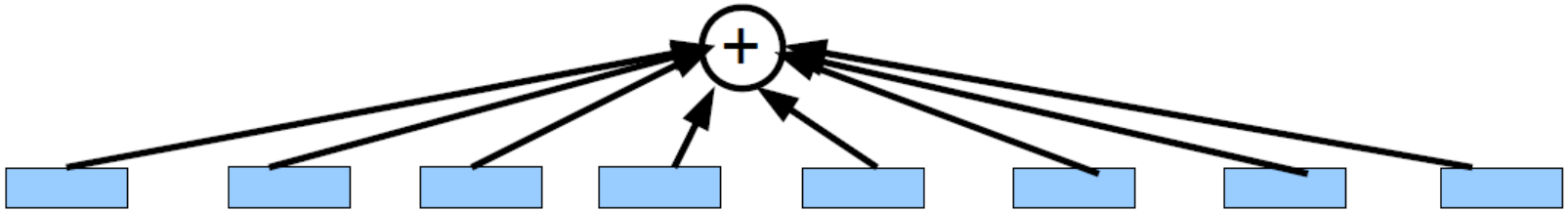
This was only with 10 input dimensions. A 224x224 input image has 150,528.

(It's significantly easier with more numbers, need smaller nudge for each)

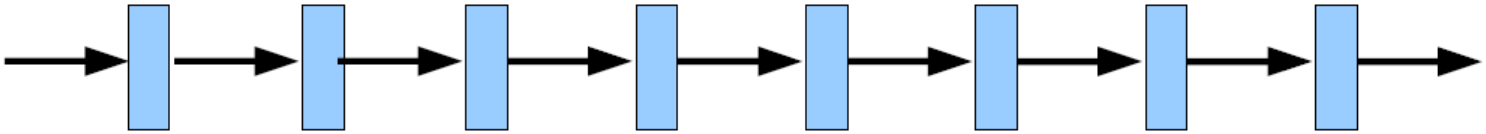
An Insight into CNN

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination $f(x) \approx \sum_j g_j$



Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



15

Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination $f(x) \approx \sum_j g_j$

- Kernel learning
- Boosting
- ...

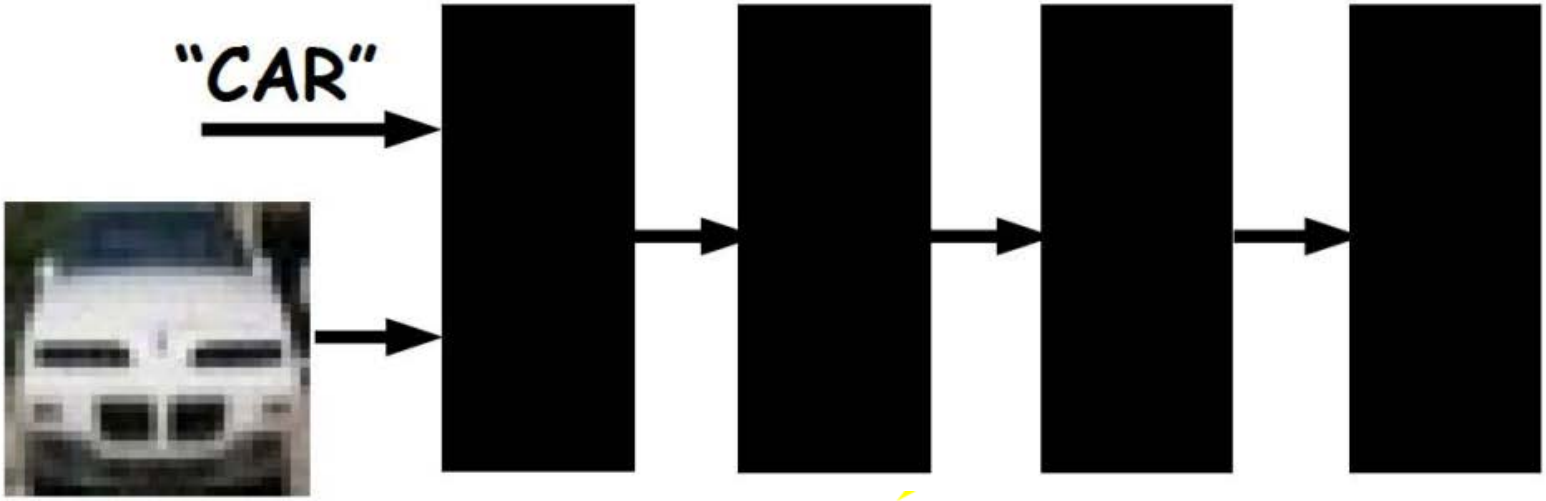
Shallow

Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$

- Deep learning
- Scattering networks (wavelet cascade)
- S.C. Zhou & D. Mumford "grammar"

Deep

16



$$f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$$

NOTE: Each black box can have trainable parameters.
Their composition makes a highly non-linear system.

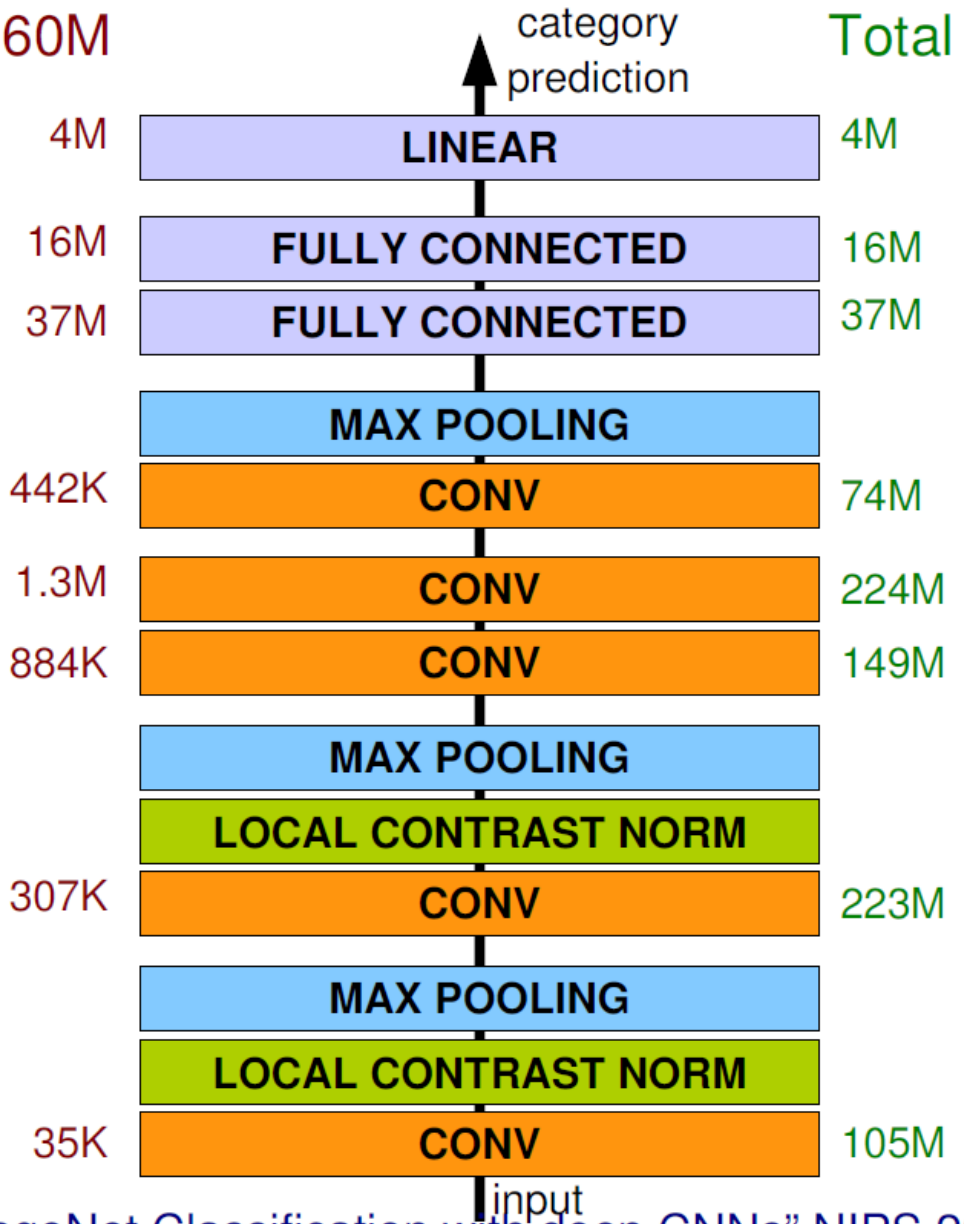
10

Ranzato

Architecture

Total nr. params: 60M

Total nr. flops: 832M

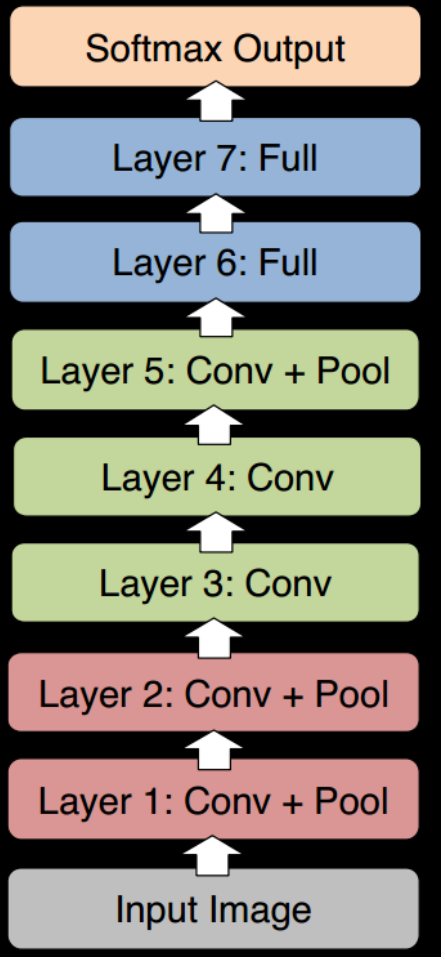


Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

- A
- Im

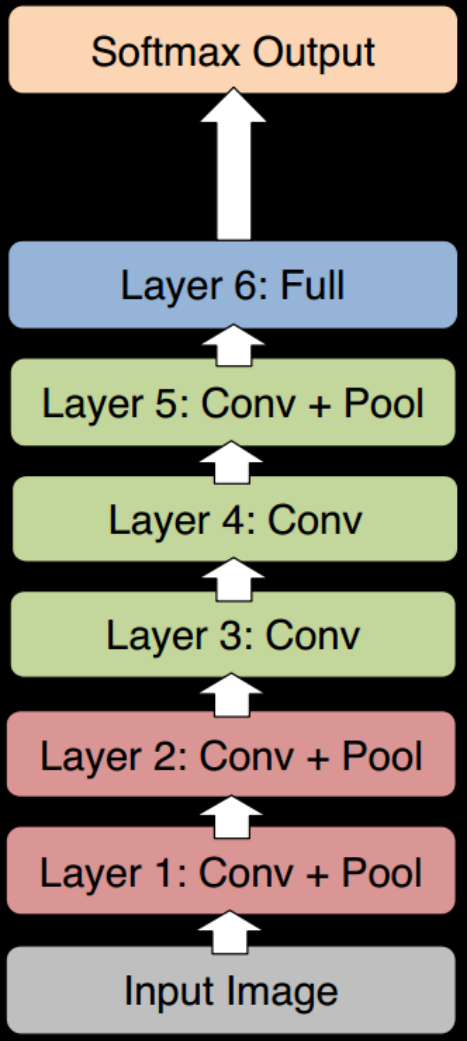
Architecture of Krizhevsky et al.

- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:
  18.1% top-5 error



Architecture of Krizhevsky et al.

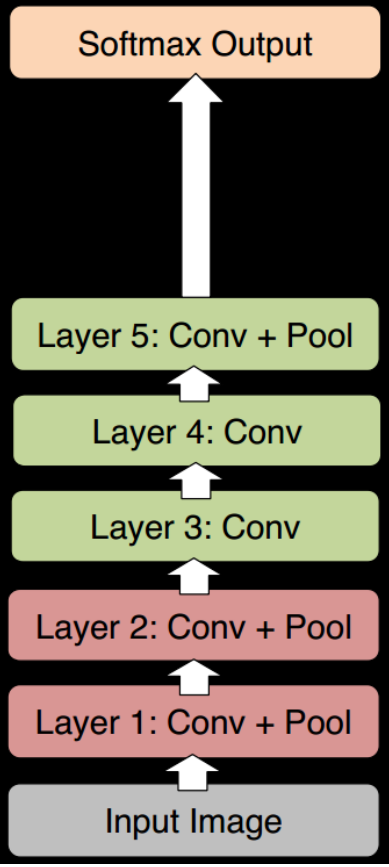
- Remove top fully connected layer
  - Layer 7
- Drop 16 million parameters
- Only 1.1% drop in performance!



Slides from Rob Fergus

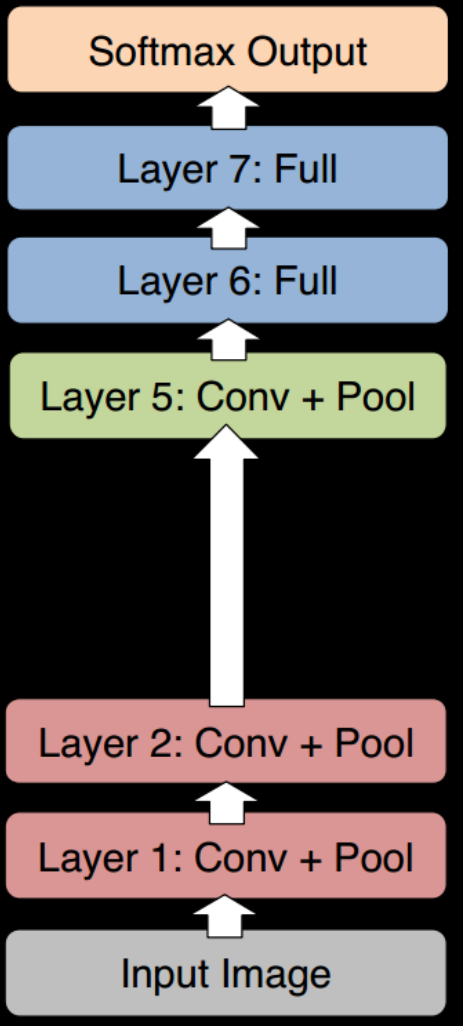
Architecture of Krizhevsky et al.

- Remove both fully connected layers
  - Layer 6 & 7
- Drop ~50 million parameters
- 5.7% drop in performance



Architecture of Krizhevsky et al.

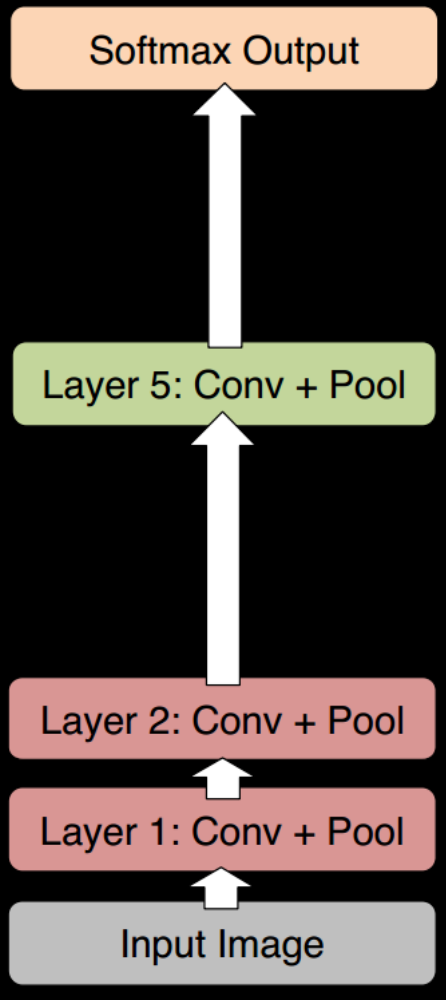
- Now try removing upper feature extractor layers:
  - Layers 3 & 4
- Drop ~1 million parameters
- 3.0% drop in performance



Architecture of Krizhevsky et al.

- Now try removing upper feature extractor layers & fully connected:
  - Layers 3, 4, 6 ,7
- Now only 4 layers
- 33.5% drop in performance

→Depth of network is key

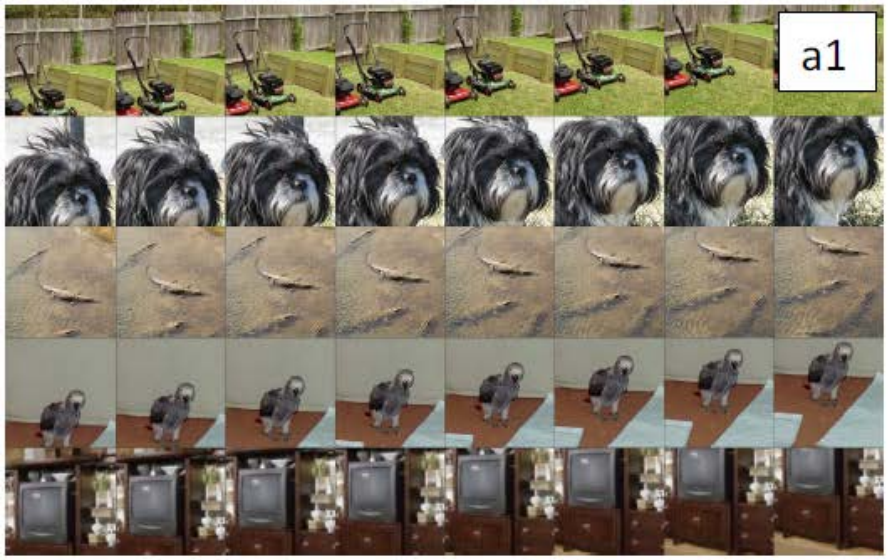


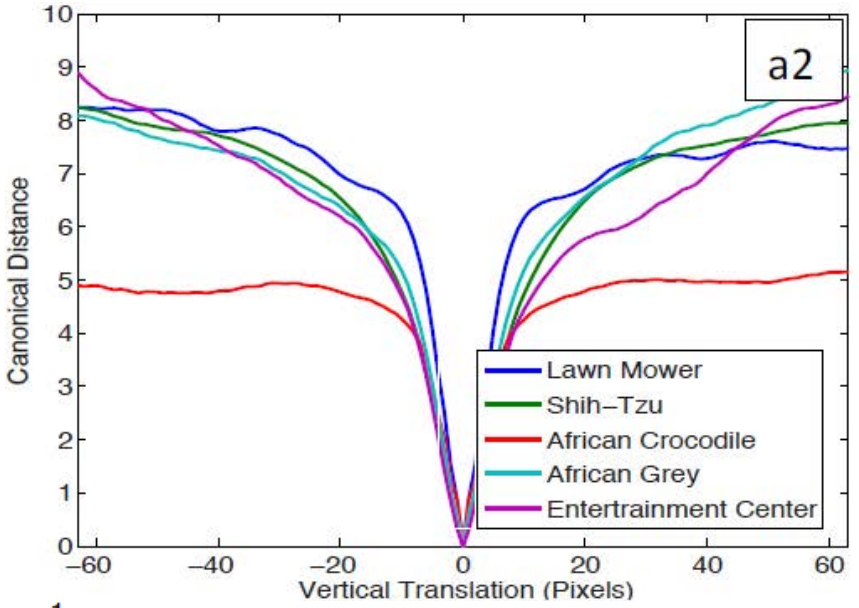
Tapping off Features @each Layer

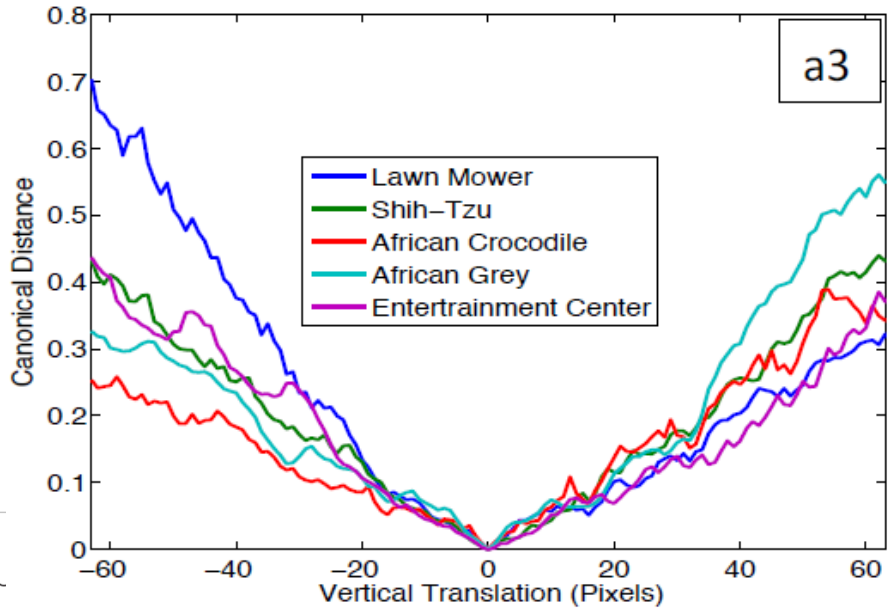
- train either a linear SVM or soft-max on features from dierent layers (as indicated in brackets) from the convnet.
- Higher layers generally produce more discriminative features.

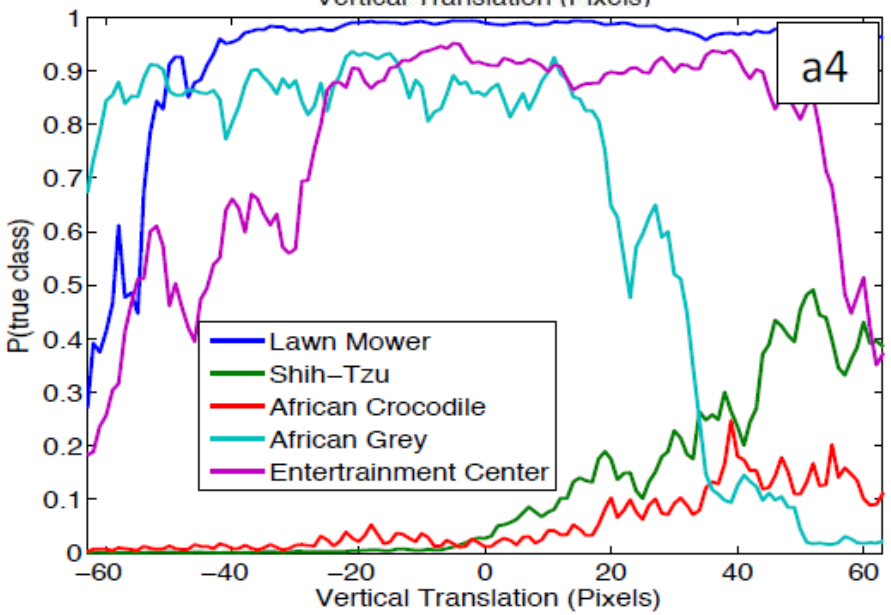
	Cal-101 (30/class)	Cal-256 (60/class)
SVM (1)	44.8 ± 0.7	24.6 ± 0.4
SVM (2)	66.2 ± 0.5	39.6 ± 0.3
SVM (3)	72.3 ± 0.4	46.0 ± 0.3
SVM (4)	76.6 ± 0.4	51.3 ± 0.1
SVM (5)	$\mathbf{86.2 \pm 0.8}$	65.6 ± 0.3
SVM (7)	$\mathbf{85.5 \pm 0.4}$	$\mathbf{71.7 \pm 0.2}$
Softmax (5)	82.9 ± 0.4	65.7 ± 0.5
Softmax (7)	$\mathbf{85.4 \pm 0.4}$	$\mathbf{72.6 \pm 0.1}$

Verticle Translation

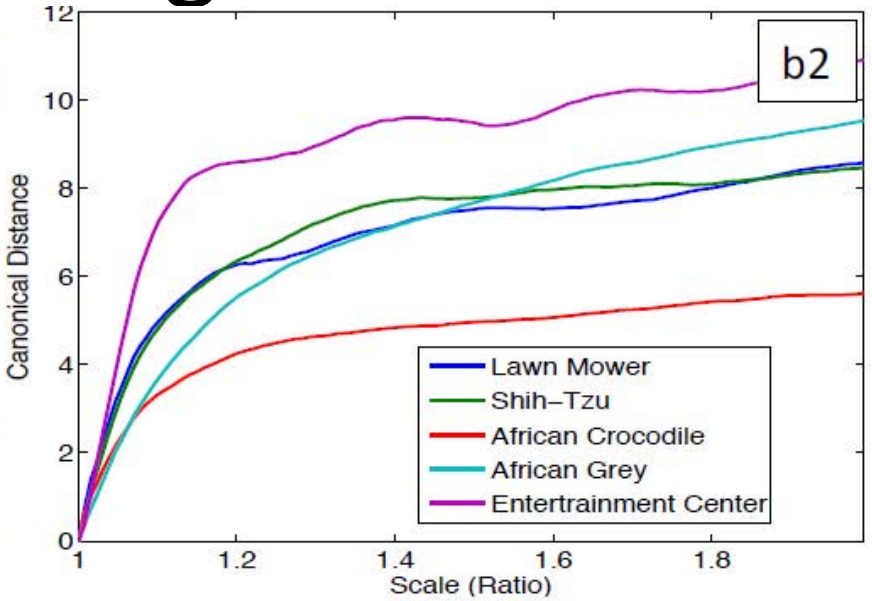


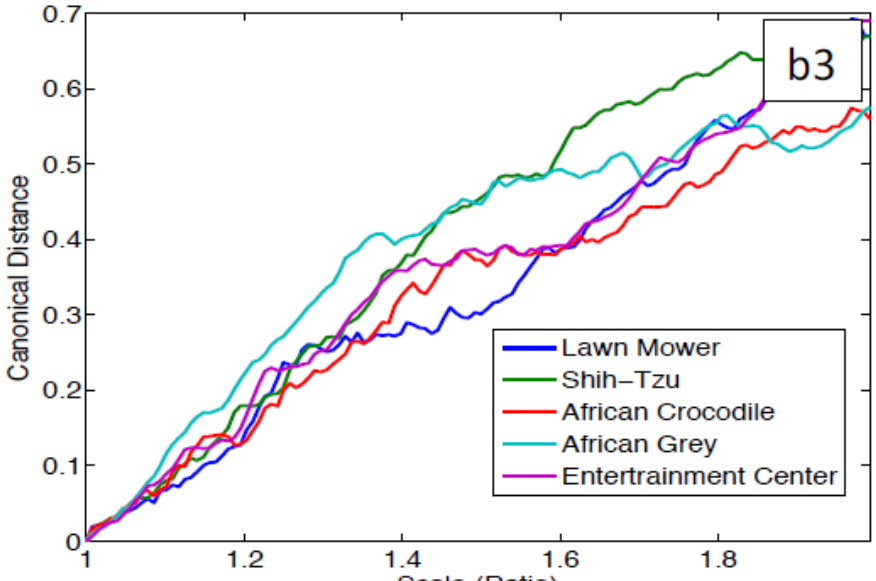


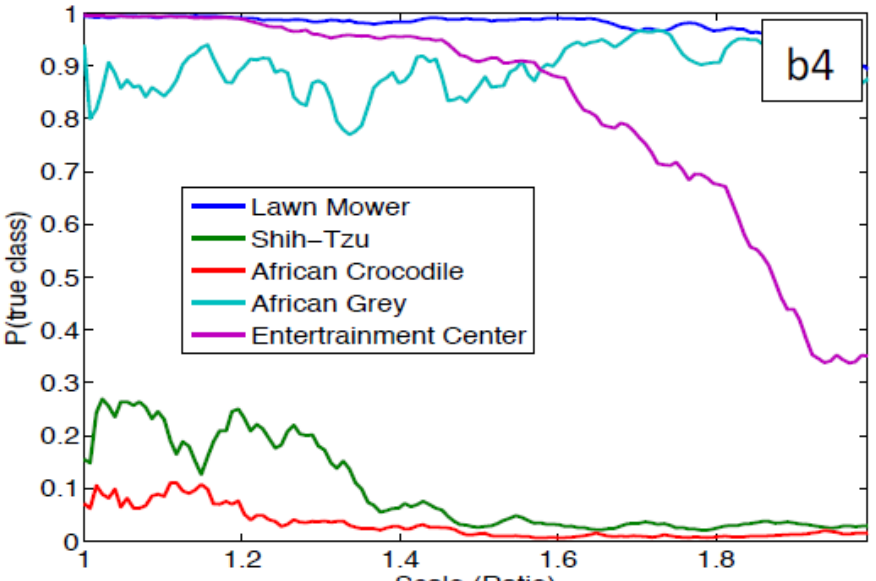




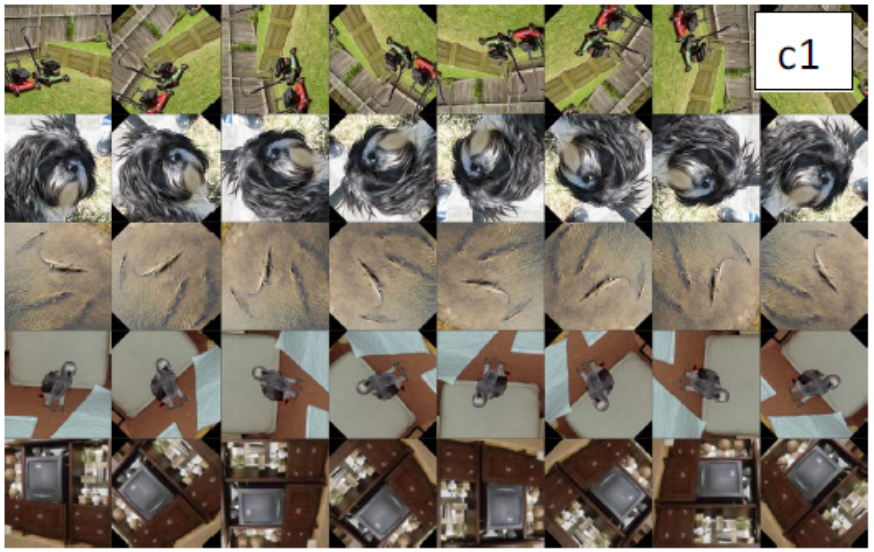
Scale changes

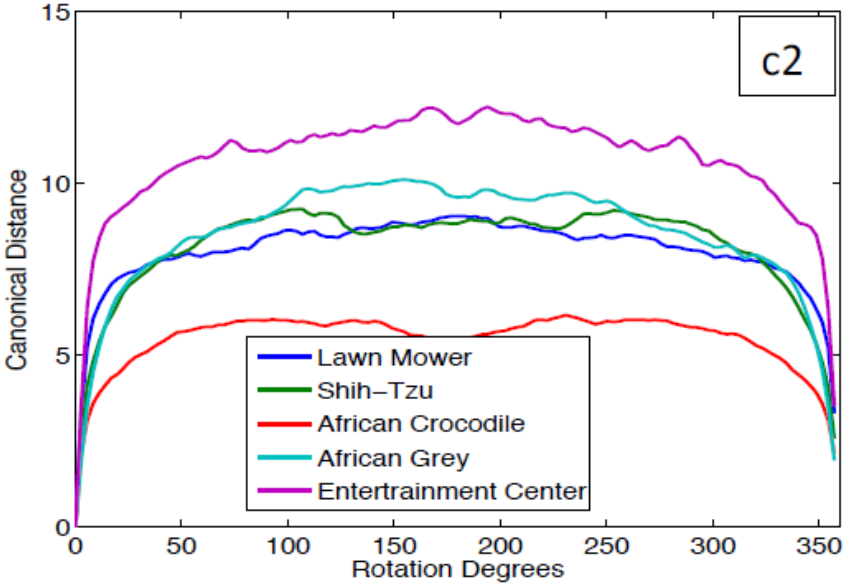


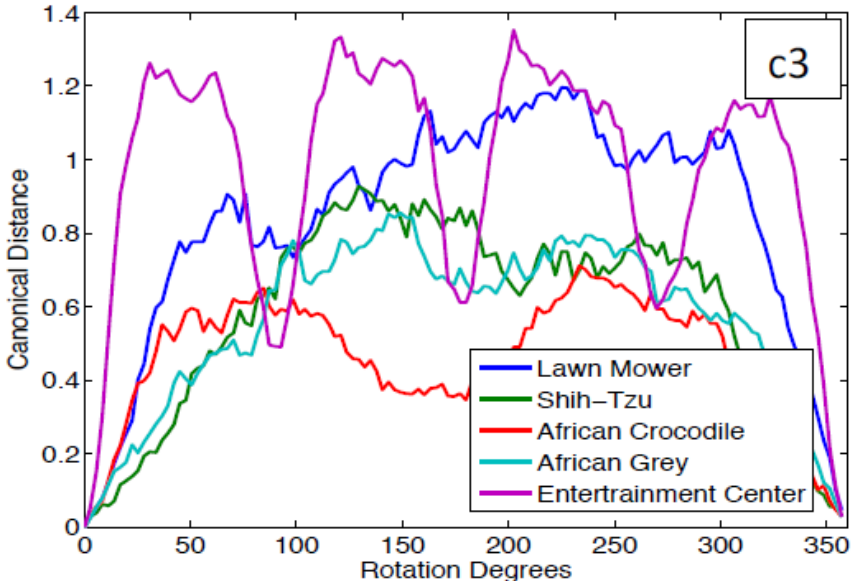


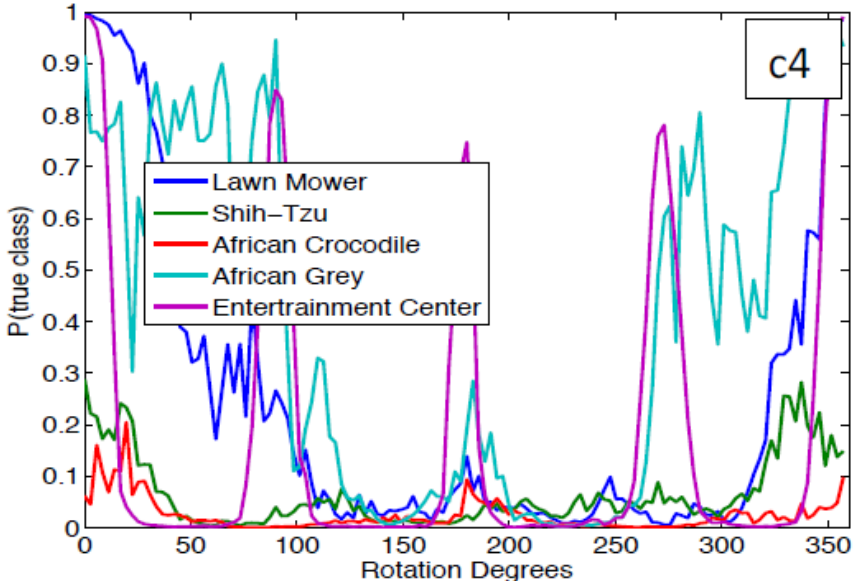


Rotation change







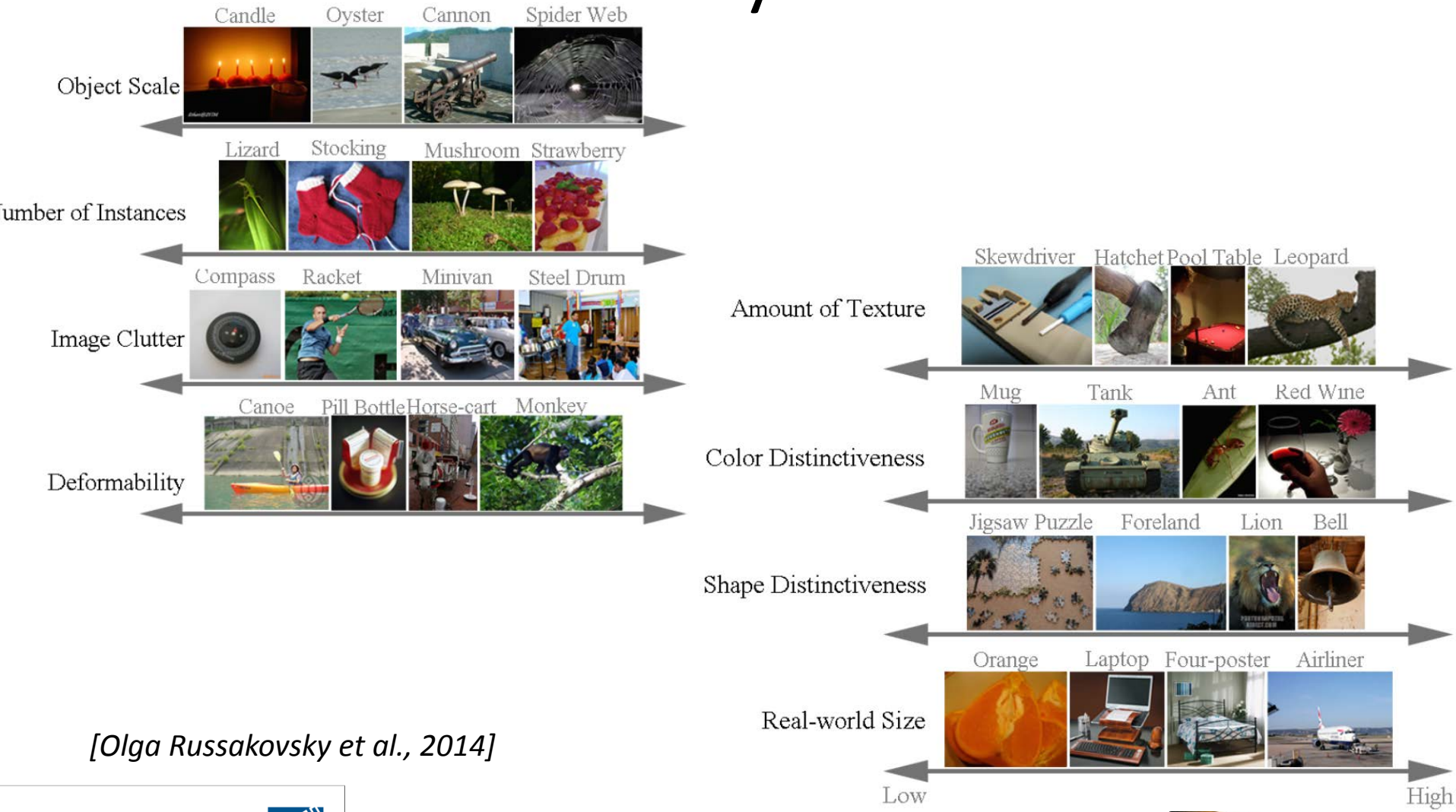


The 2014 ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- 1.2 million images in the training set, each labeled with one of 1000 categories
  - cover a wide variety of objects, animals, scenes, and even some abstract geometric concepts such as *"hook"*, or *"spiral"*.
- The 100,000 test set images are released with the dataset, but not with the labels.
- The teams have to predict 5 (out of 1000) classes and an image is considered to be correct if at least one of the predictions is the ground truth.

ILSVRC

ImageNet (ILSVRC competition) analysis



[Olga Russakovsky et al., 2014]

Image classification

Easiest classes

red fox (100) hen-of-the-woods (100) ibex (100) goldfinch (100) flat-coated retriever (100)

tiger (100) hamster (100) porcupine (100) stingray (100) Blenheim spaniel (100)

Hardest classes

muzzle (71) hatchet (68) water bottle (68) velvet (68) loupe (66)

hook (66) spotlight (66) ladle (65) restaurant (64) letter opener (59)

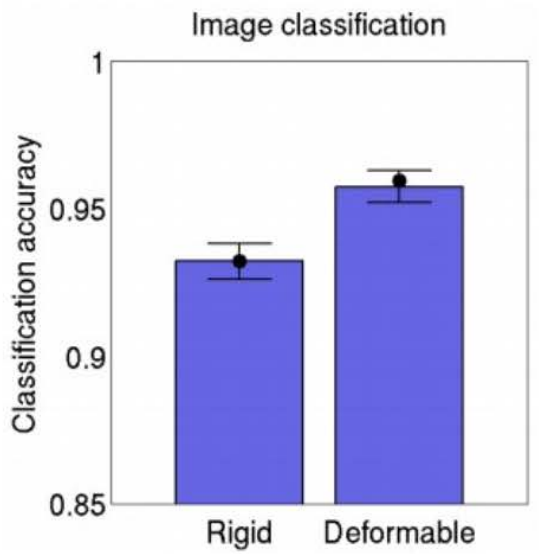


Image classification
Classification accuracy
1
0.95
0.9
0.85
Rigid
Deformable

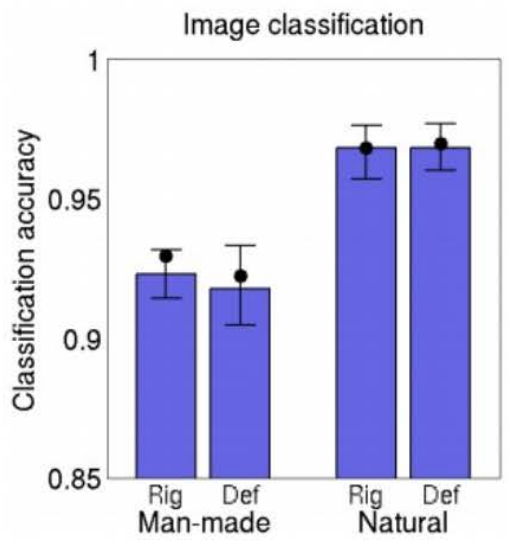


Image classification
Classification accuracy
1
0.95
0.9
0.85
Rig
Def
Rig
Def
Man-made
Natural

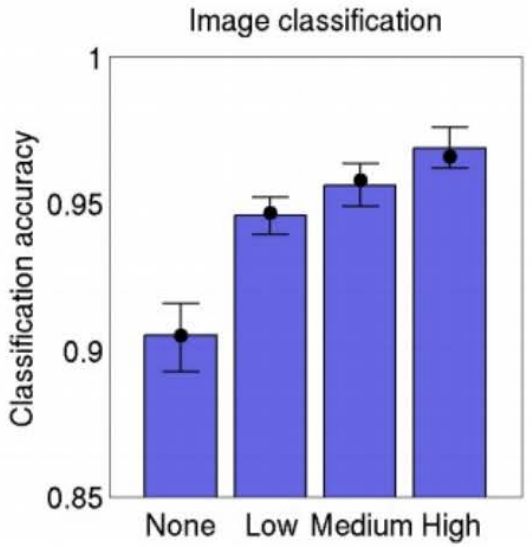


Image classification
Classification accuracy
1
0.95
0.9
0.85
None
Low
Medium
High

CNN vs. Human

Labeling Interface

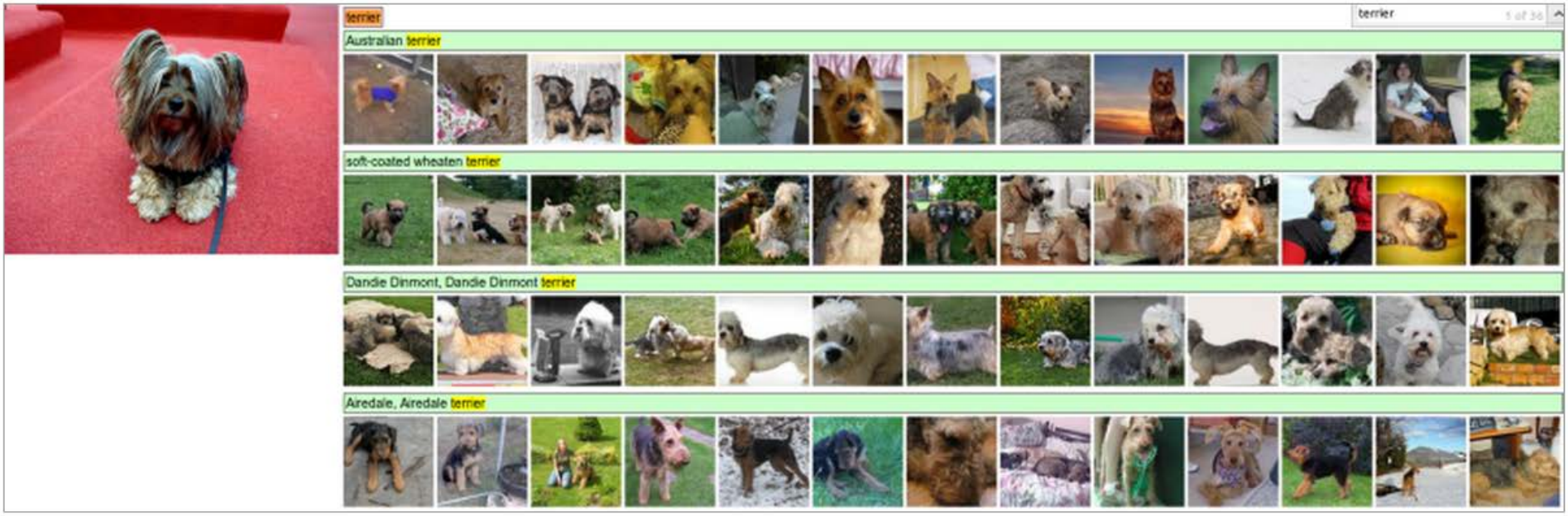
I developed a labeling interface that would help us evaluate the human performance. It looked similar to, but not identical, to the screenshot below:

A crop of a screenshot of the labeling interface for the ILSVRC validation data. Try it out for yourself.

What I learned from competing against a ConvNet on ImageNet
Karpathy, 2014: http://bit.ly/humanvsconvnet

CNN vs. Human

> *My error turned out to be 5.1%, compared to GoogLeNet error of 6.8%. Still a bit of a gap to close (and more).*



Representative example of practical frustrations of labeling ILSVRC classes. Aww, a cute dog! Would you like to spend 5 minutes scrolling through 120 breeds of dog to guess what species it is?

What I learned from competing against a ConvNet on ImageNet
Karpathy, 2014: http://bit.ly/humanvsconvnet

CNN vs. Human

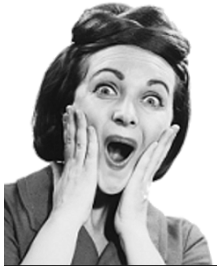
	GoogLeNet correct	GoogLeNet wrong
Human correct	1352/1500 	72/1500 • Objects very small or thin • Abstract representations • Image filters
Human wrong	46/1500 • Fine-grained recognition • Class unawareness • Insufficient training data	30/1500 • Multiple objects • Incorrect annotations

GoogLeNet: 6.8%
Andrej: 5.1% phew...

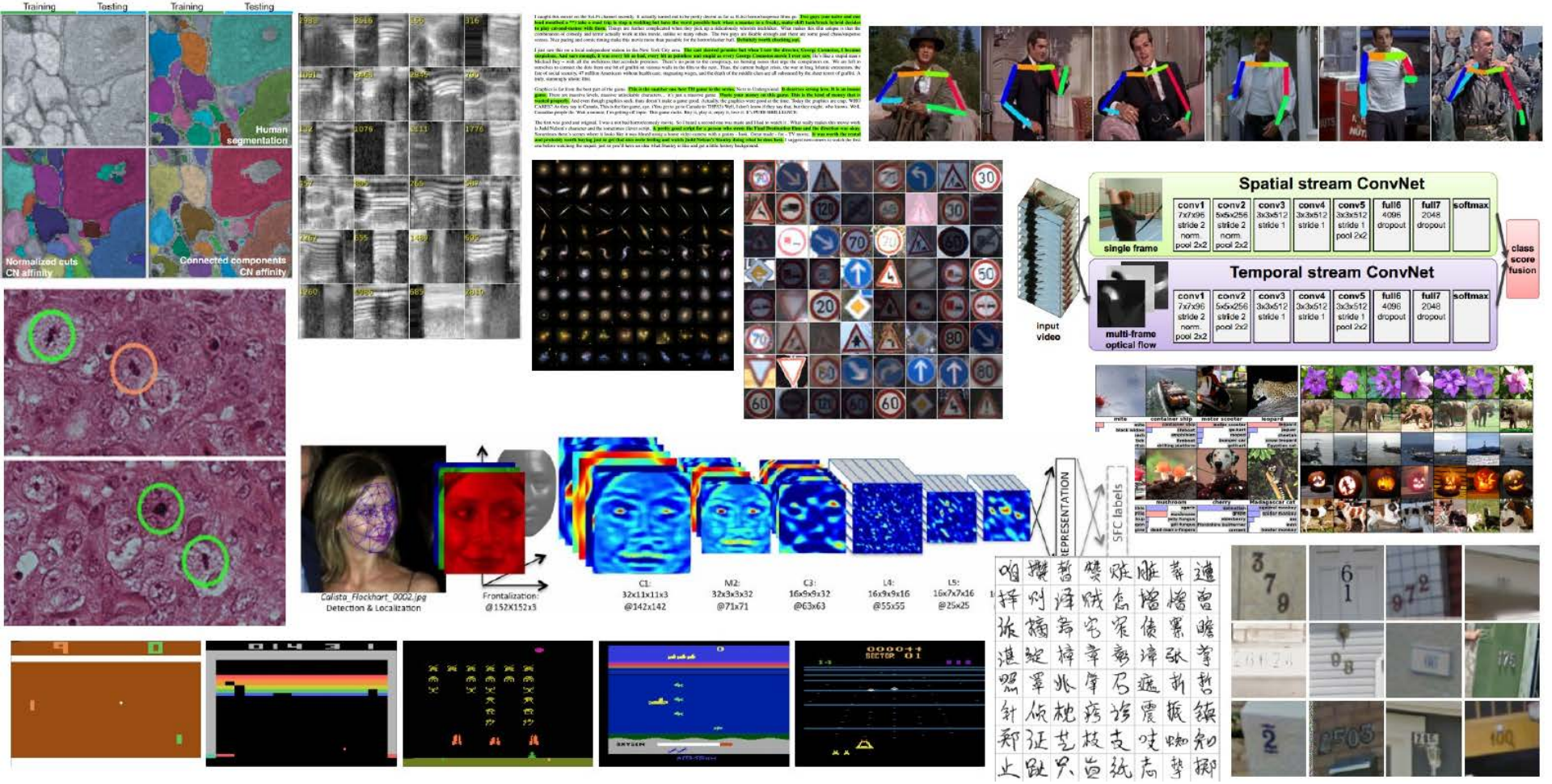
rule, ruler	king crab, Alaska crab	sidewinder	saltshaker, salt shaker	reel	hatchet	schipperke
pencil box, pencil case	pizza, pizza pie	maze, labyrinth	pill bottle	stethoscope	vase	schipperke
rubber eraser, rubber	strawberry	gar, garfish	water bottle	whistle	pitcher, ewer	groenendael
ballpoint, ballpoint pen	orange	valley, vale	lotion	ice lolly, lolly	coffeepot	doormat, welcome mat
pencil sharpener	fig	hammerhead	hair spray	hair spray	mask	teddy, teddy bear
carpenter's kit, tool kit	ice cream, icecream	sea snake	beer bottle	maypole	cup	jigsaw puzzle

CNN vs. Human

Try it out yourself: *http://cs.stanford.edu/people/karpathy/ilsvrc/*



Successful application



Visualization

- Check the (image) code space
- Visualize the filter
- What images maximize the score of some class in a ConvNet?

Check the (image) code space

...
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
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 ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

"CNN code"

A CNN transforms the image to 4096 numbers that are then linearly classified.

Check the (image) code space

("CNN code" = 4096-D vector before classifier)

query image

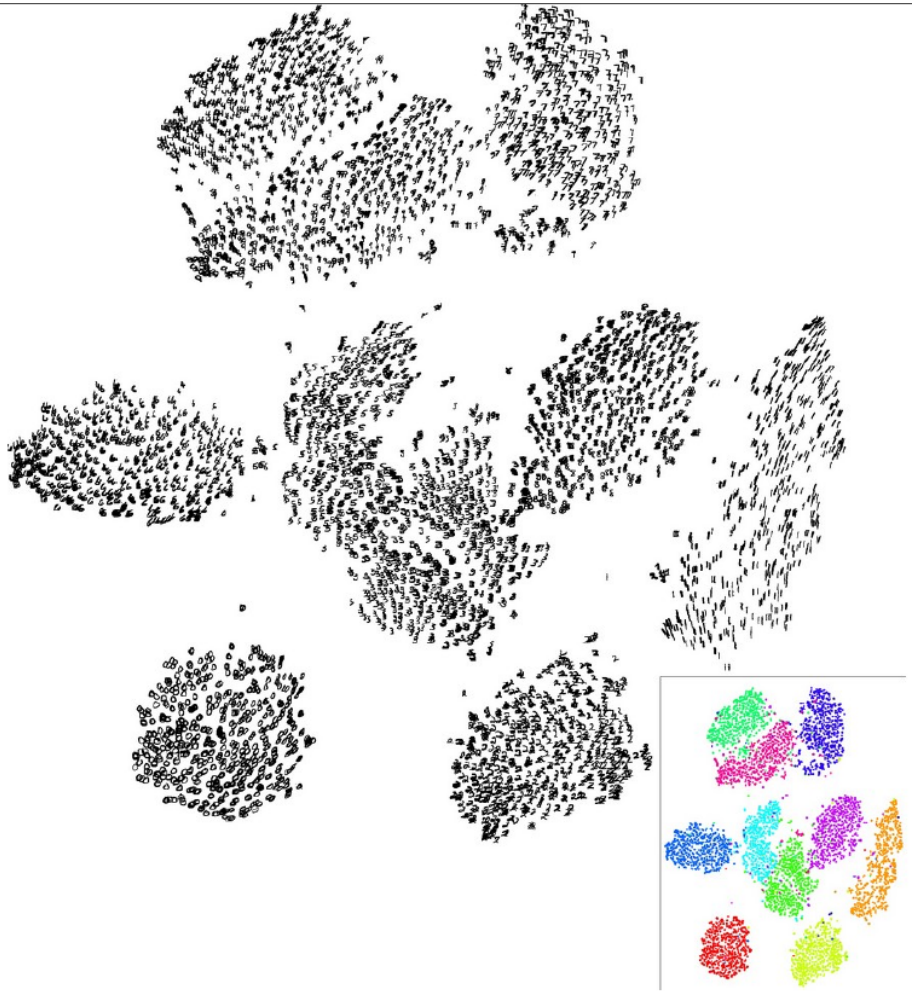
nearest neighbors in the "code" space

(But we'd like a more general (global) way to visualize the distances)

t-SNE visualization

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

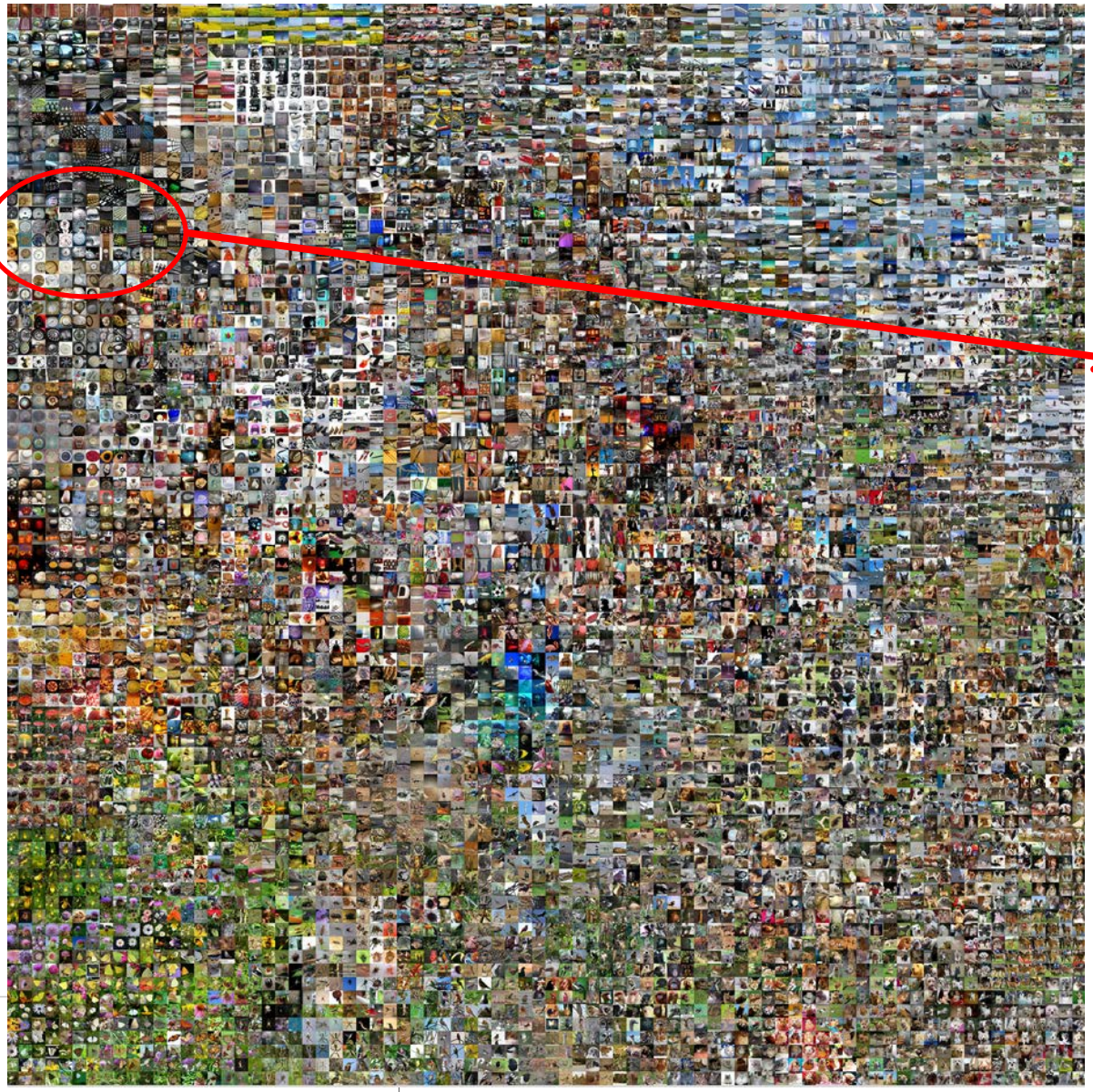
http://lvdmaaten.github.io/tsne/

ILSVRC
1k by 1k case

- two images are placed nearby if their CNN codes are close.

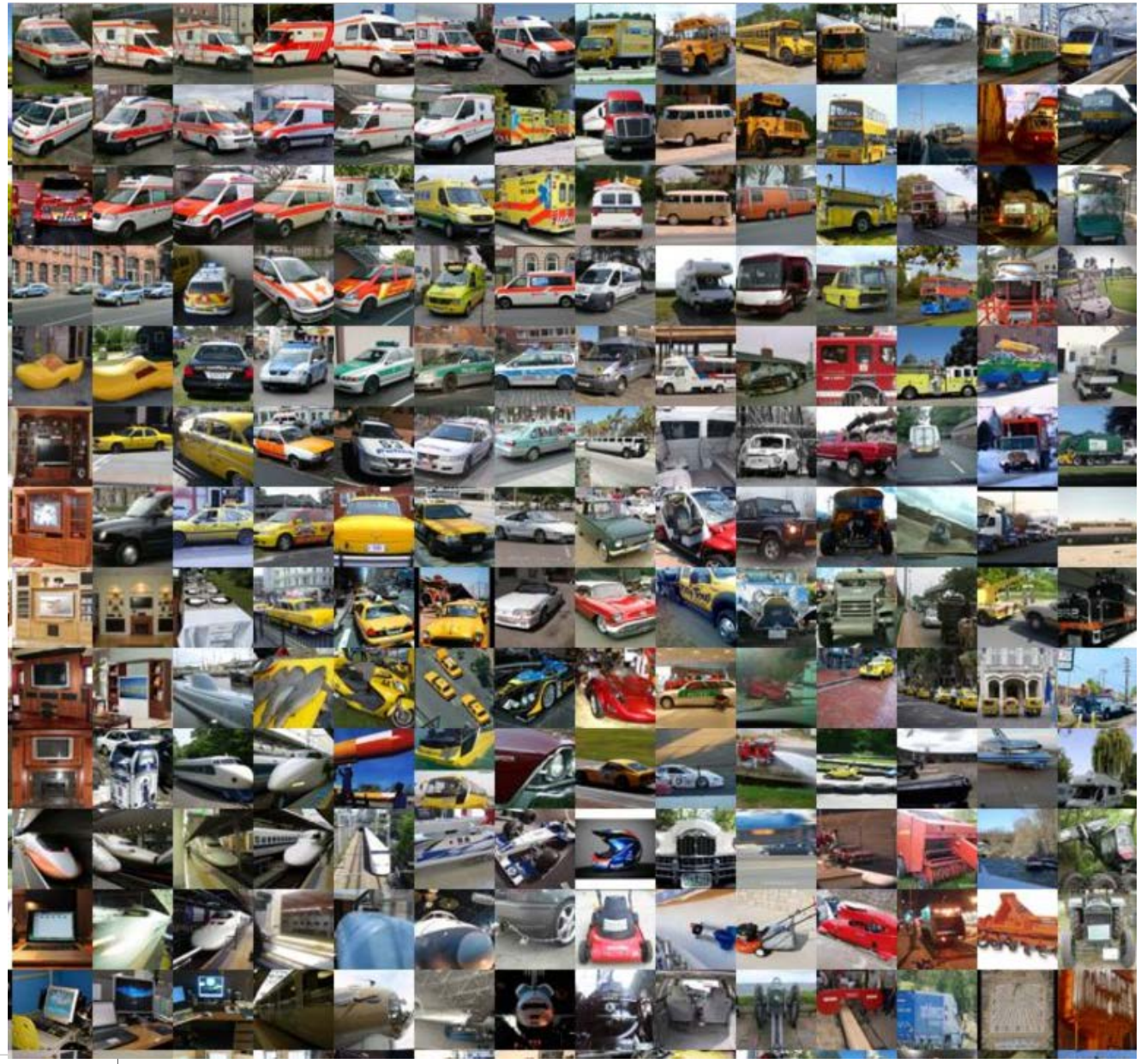
http://cs.stanford.edu/people/karpathy/cnnembed/

ILSVRC
4k by 4k case



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

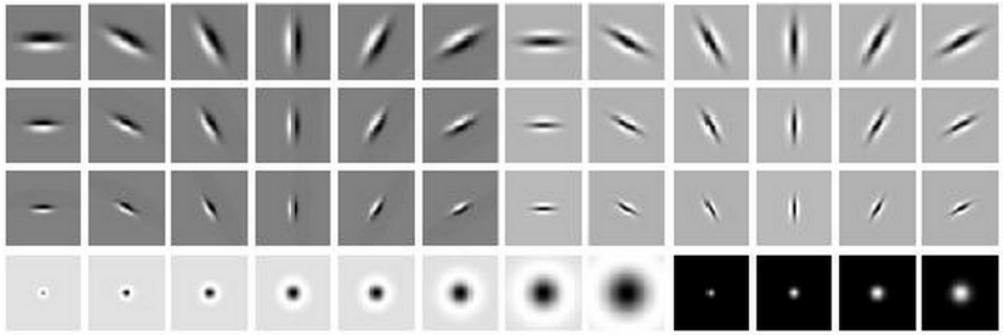
Infotech Oulu

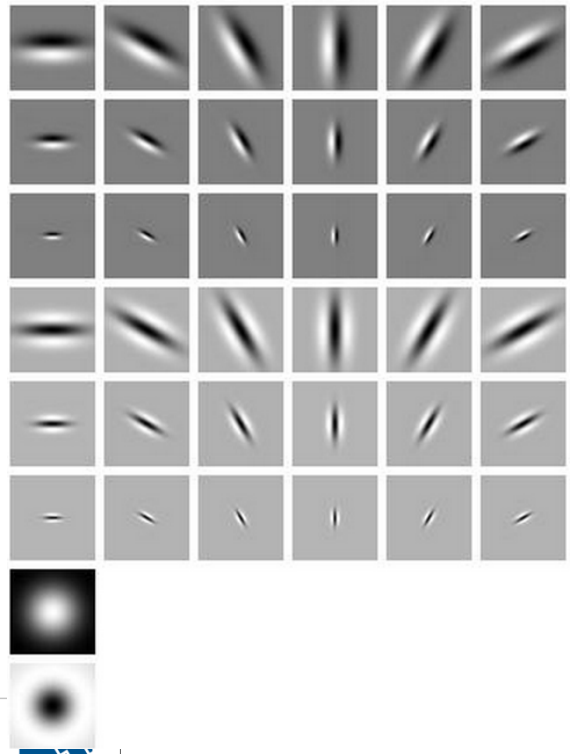


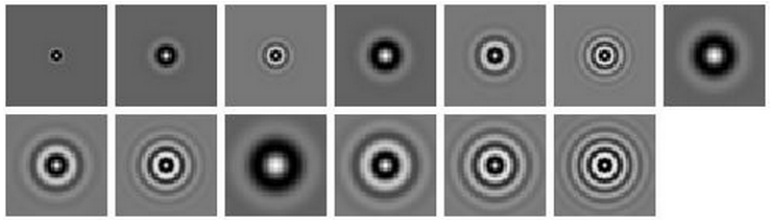
Visualization

- Check the (image) code space
- Visualize the filter
- What images maximize the score of some class in a ConvNet?

Visualize the filter

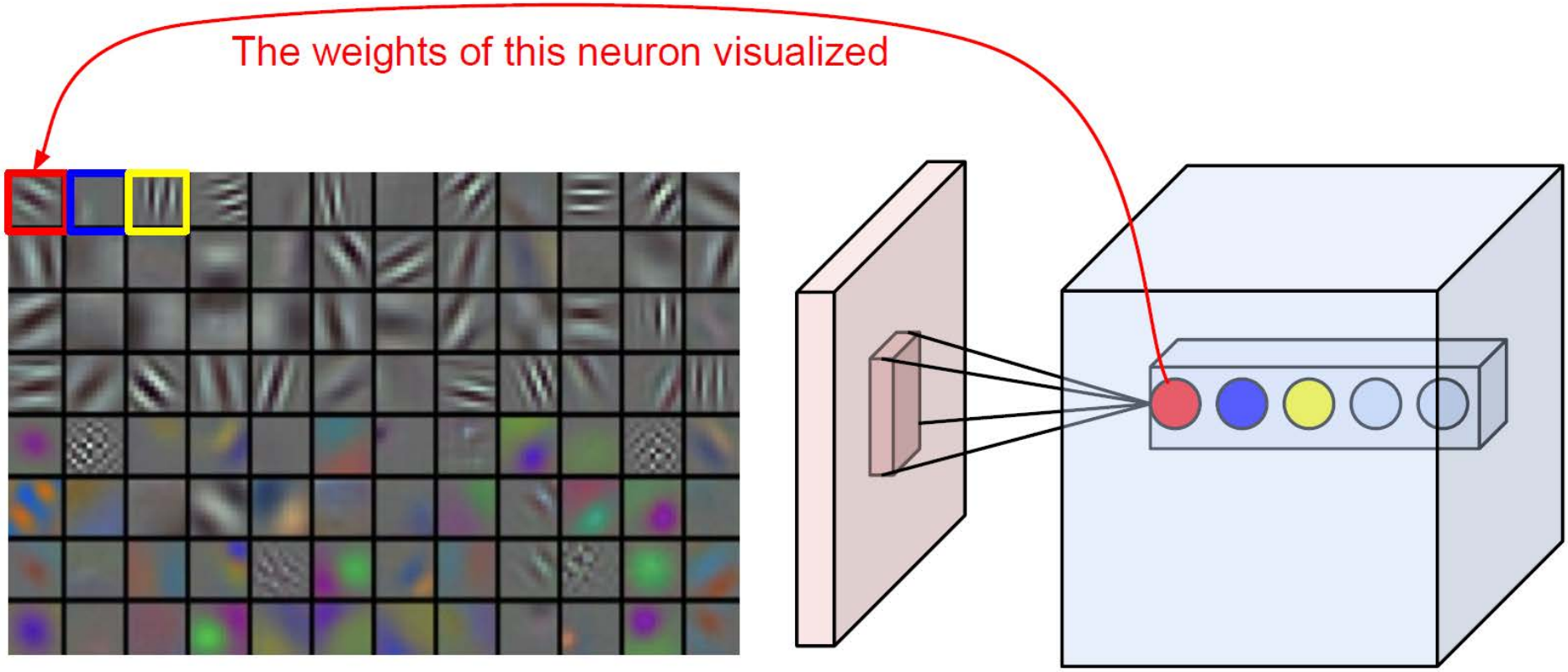






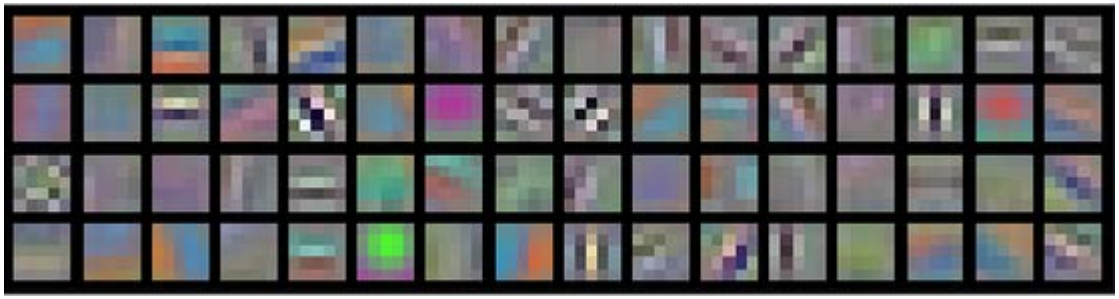
http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html

Visualize the filter



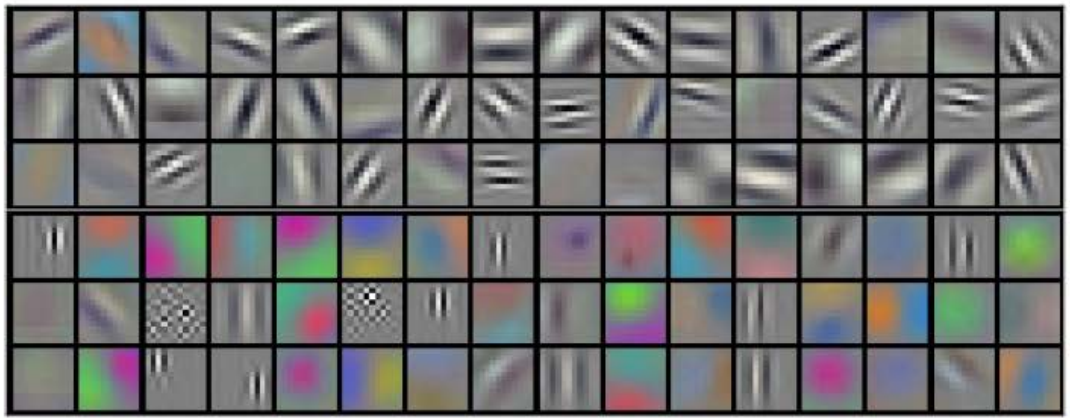
Cuda-Convnet1

- Filters learned by Cuda- Convnet 1 on CIFAR-10 :



https://code.google.com/p/cuda-convnet/

AlexNet

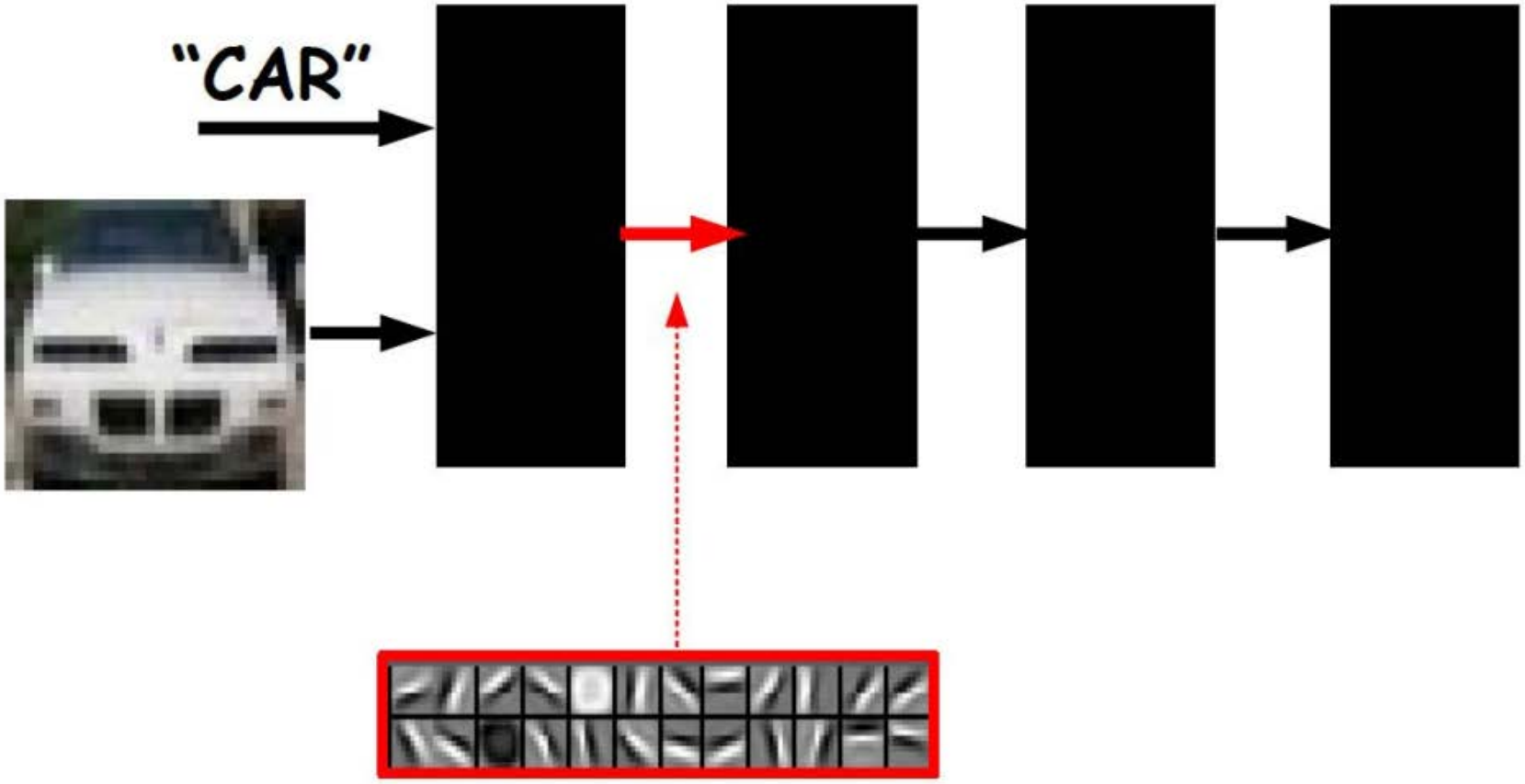


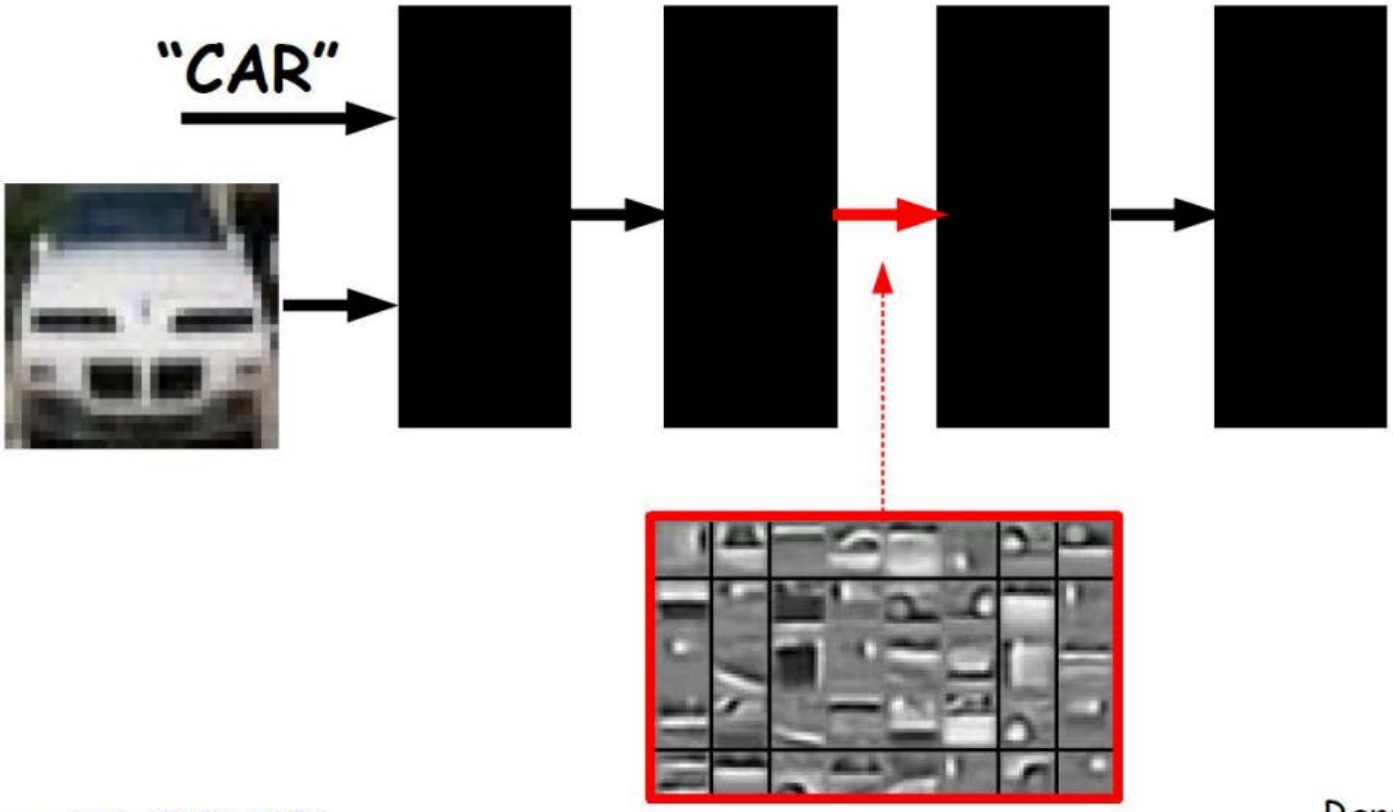
First layer learned filters (processing raw pixel values).

115

Krizhevsky et al. "ImageNet Classification with deep CNNs" NIPS 2012

Ranzato

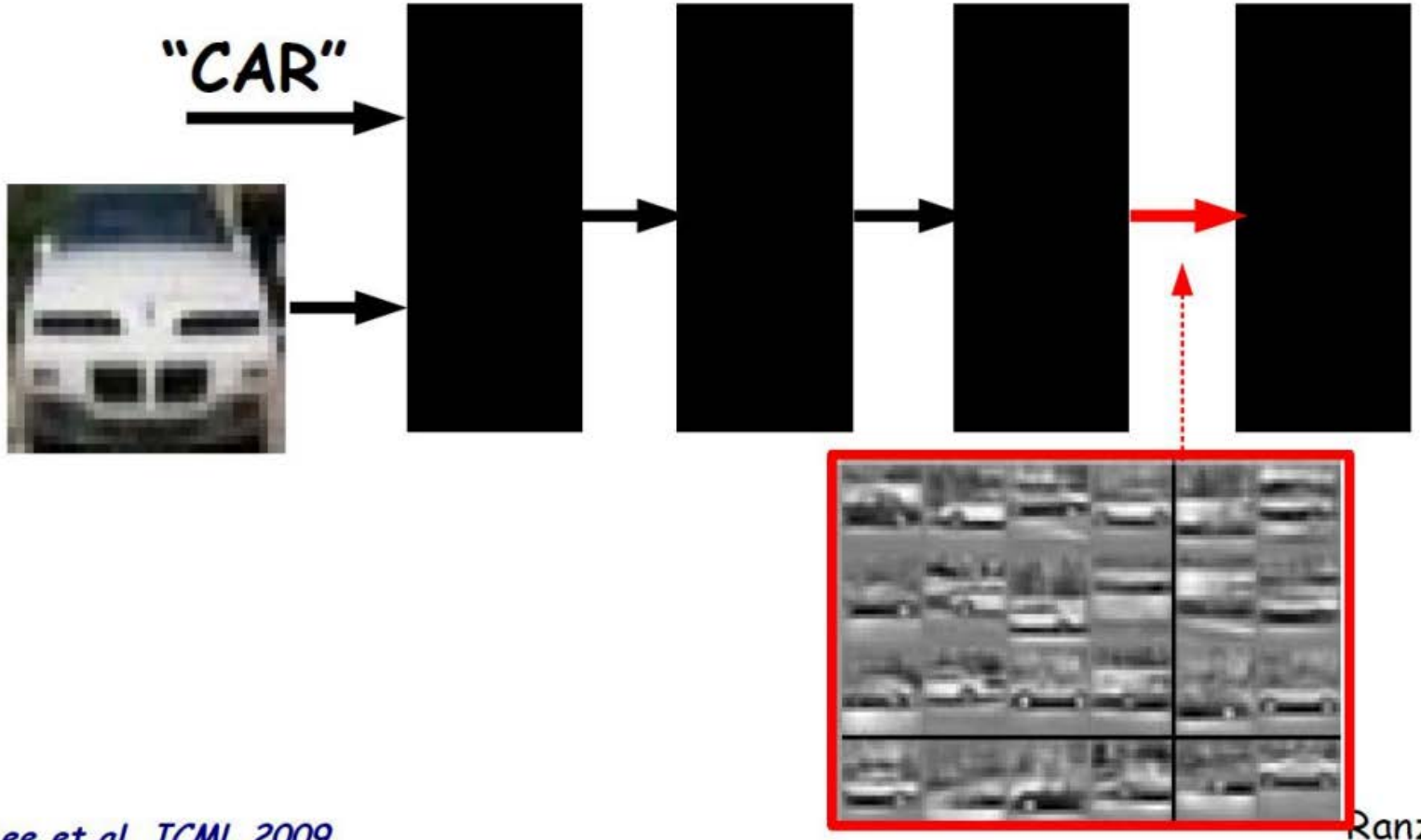

"CAR"



Lee et al. ICML 2009

13

Ranzato

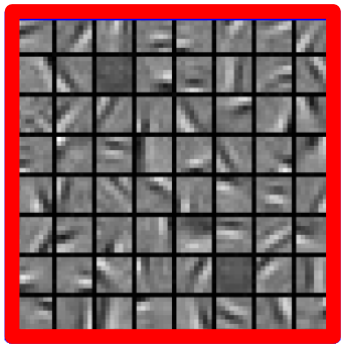
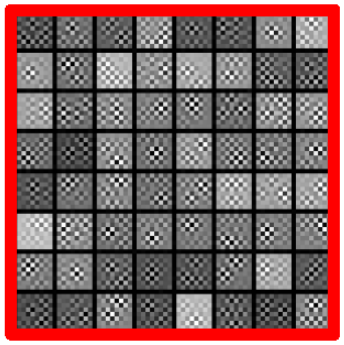
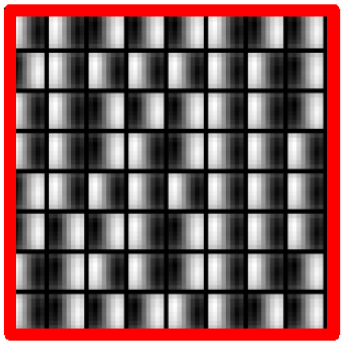
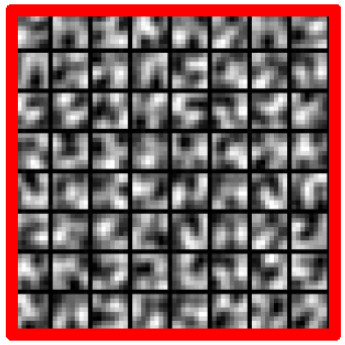


Lee et al. ICML 2009

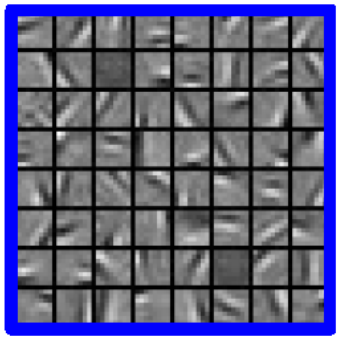
14

Ranzato

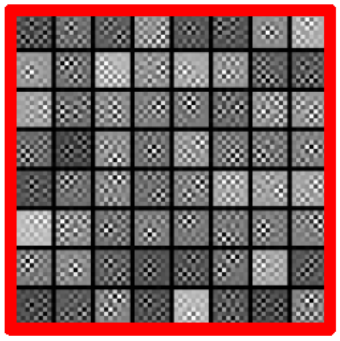
Diagnosing: A Quiz

GOOD

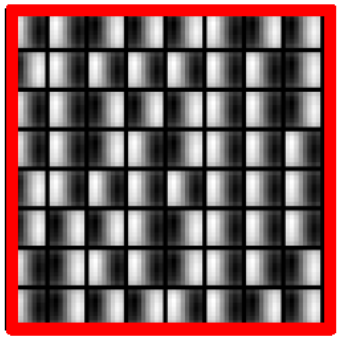


BAD



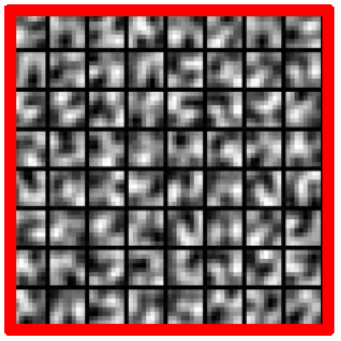
too noisy

BAD



too correlated

BAD



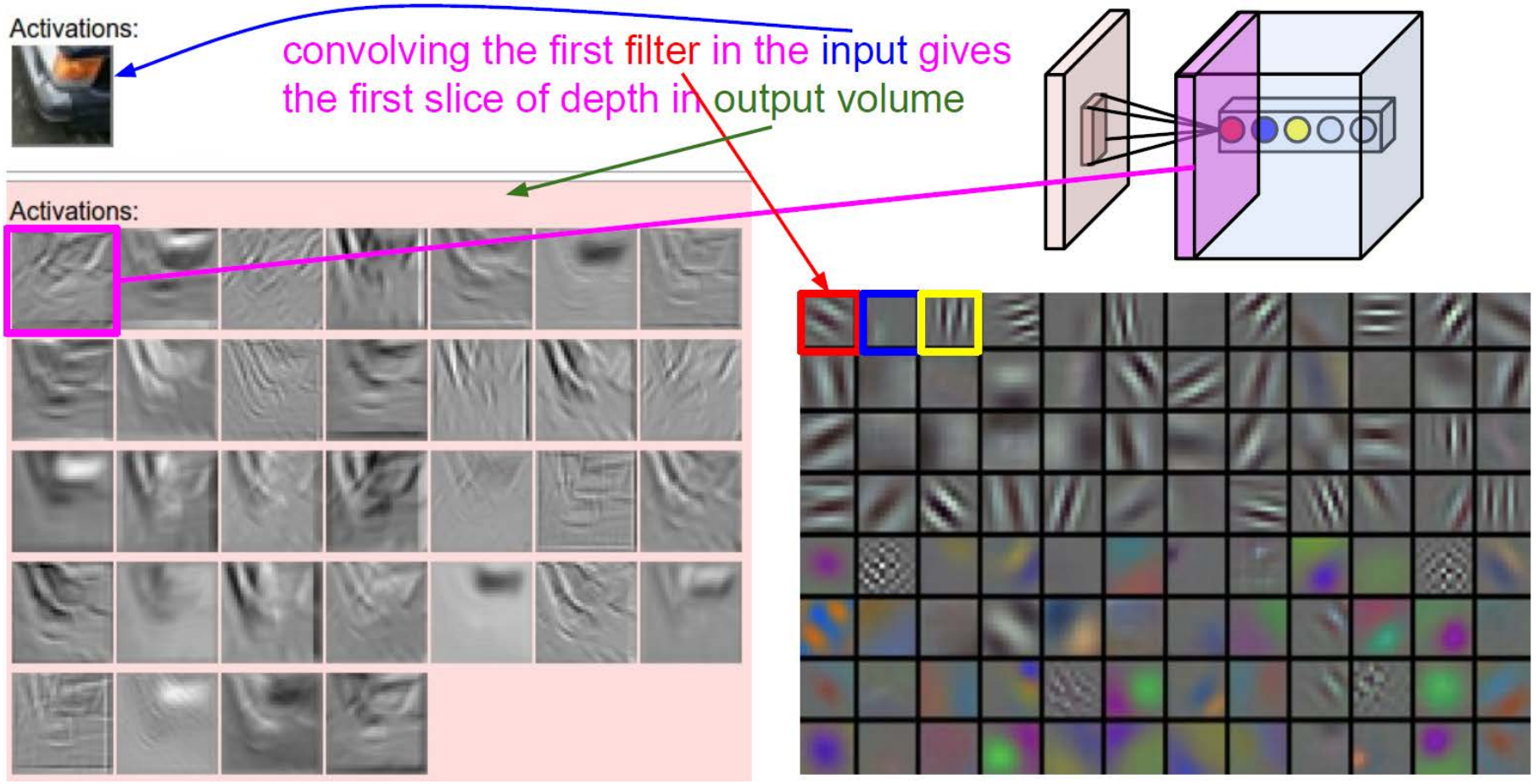
lack structure

Good training: learned filters exhibit structure and are uncorrelated.

Visualization

- Check the (image) code space
- Visualize the filter
- What images maximize the score of some class in a ConvNet?

Visualizing the input maximizing the scores



Visualizing the input maximizing the scores

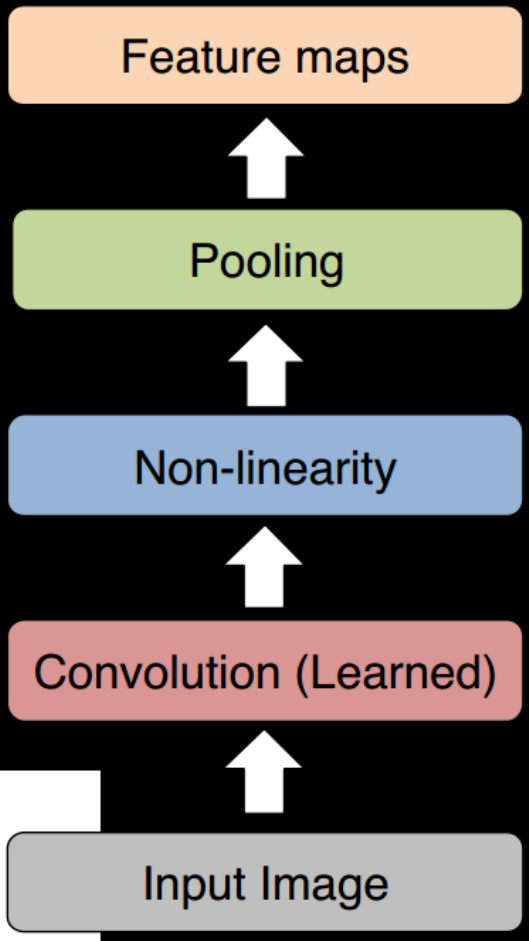
- DeconvNet based visualization
- Optimization based visualization
- Reconstruct the original image?

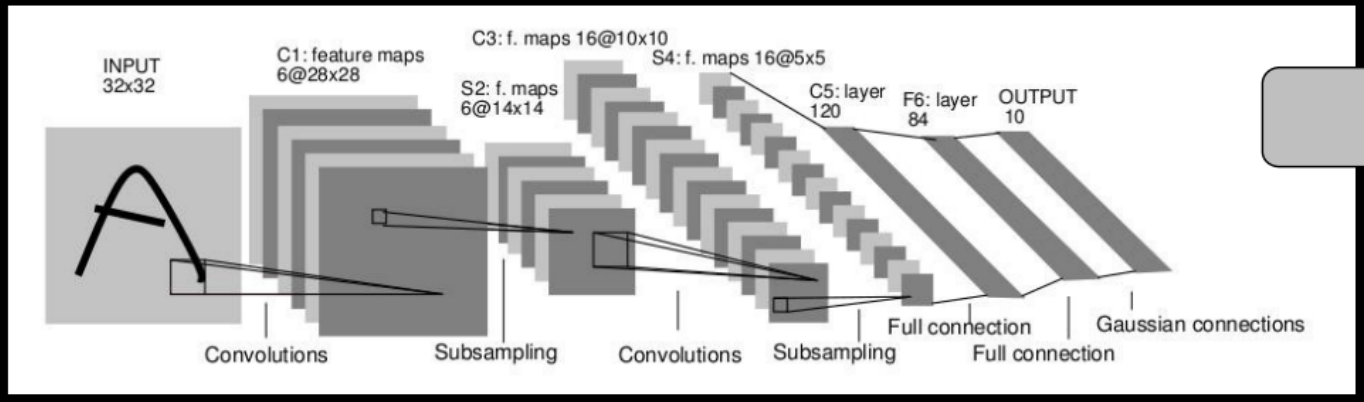
DeConvNet based Visualization

- Mapping with a Deconvolutional Network (deconvnet).
  - a convnet model that uses the same components (filtering, pooling) but in reverse, so instead of mapping pixels to features does the opposite.
- In (Zeiler et al., 2011), deconvnets were proposed as a way of performing unsupervised learning.
- Used here purely as a probe
  - No inference, no learning
  - No contrast normalization

Recap of Convnets

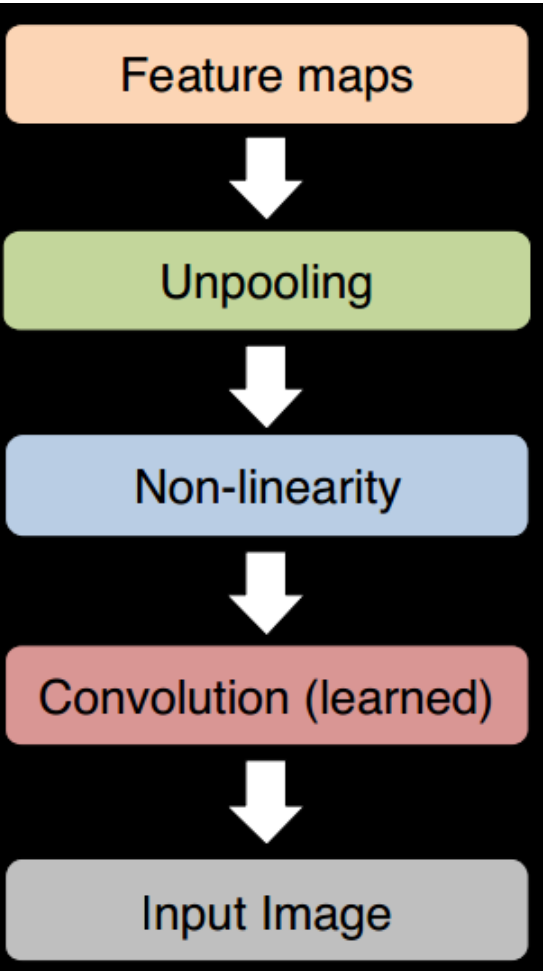
- Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised
- Train convolutional filters by back-propagating classification error





[LeCun et al. 1989]

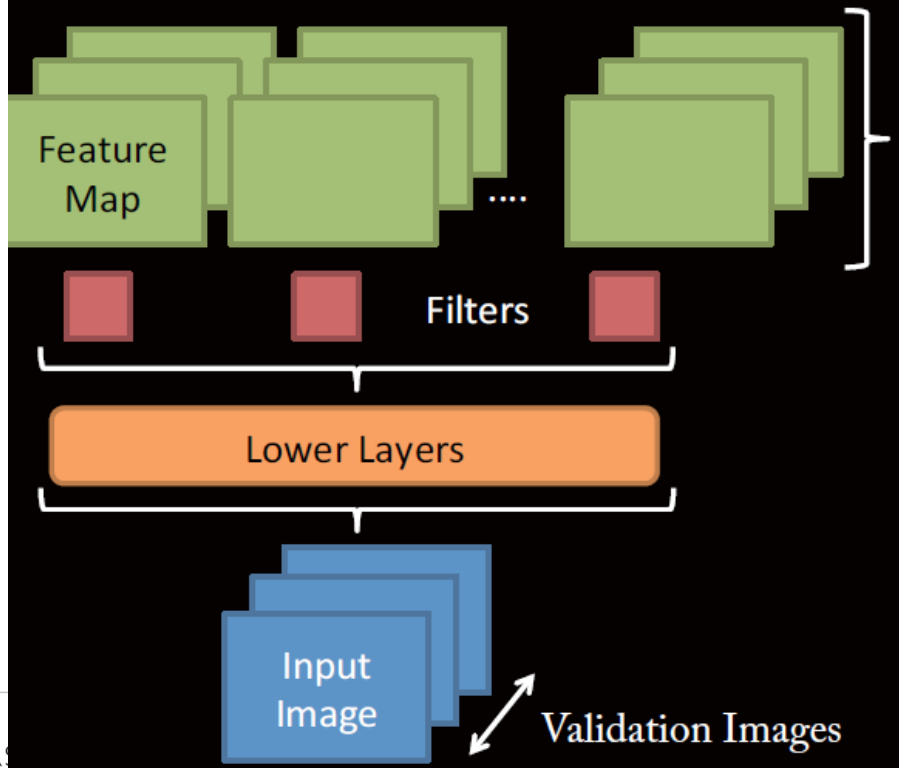
DeConvNet based Visualization



- Provides way to map activations at high layers back to the input
- Same operations as Convnet, but in reverse:
  - Unpool feature maps
  - (De-)Convolve unpooled maps
- Filters copied from Convnet
  - The transposed version of the same filter

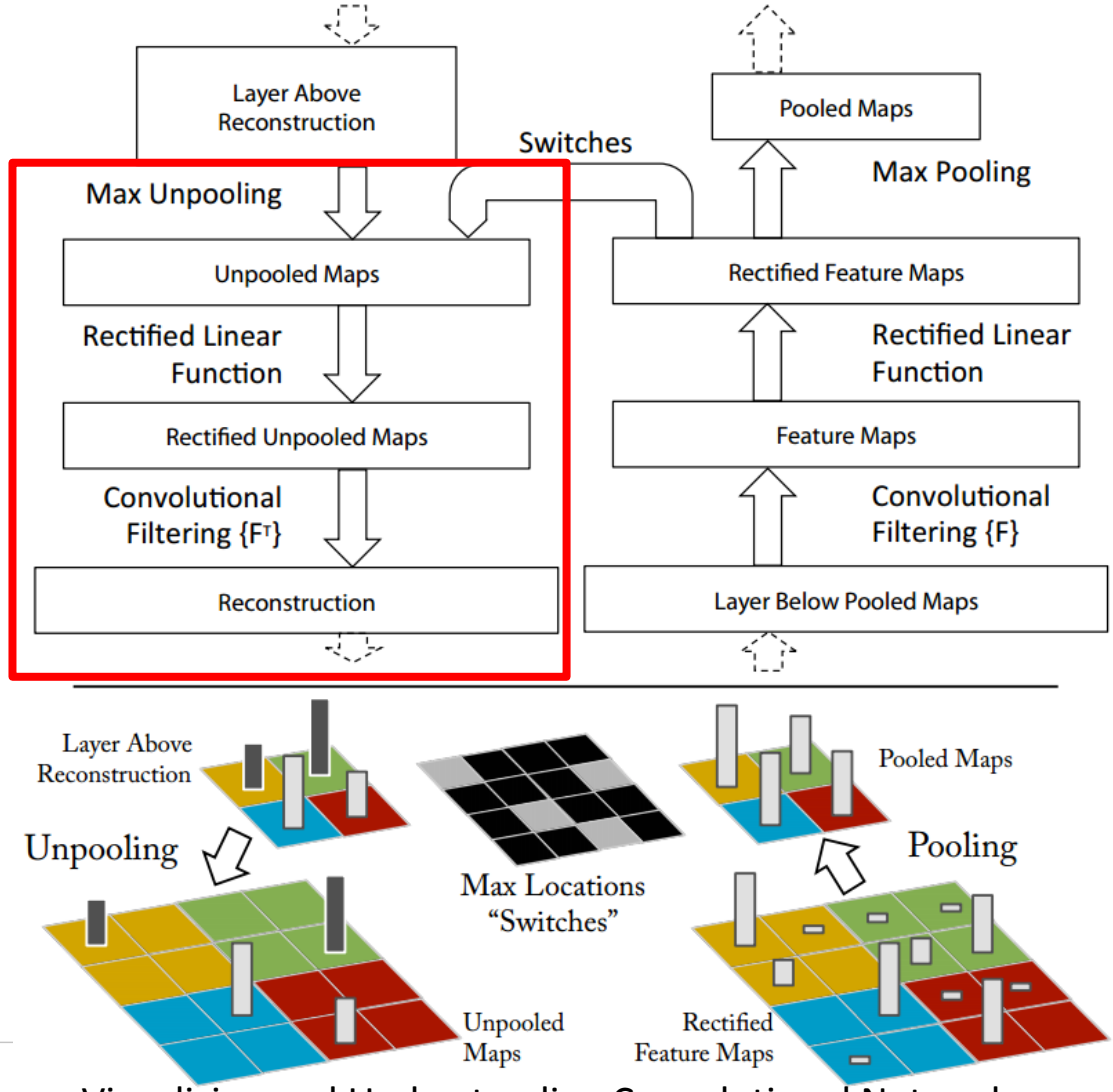
Visualizations of Higher Layers

- Use ImageNet 2012 validation set [Zeiler and Fergus. arXiv'13]
- Push each image through network



- Take max activation from feature map associated with each filter
- Use Deconvnet to project back to pixel space
- Use pooling "switches" peculiar to that activation

A DeConvNet Layer

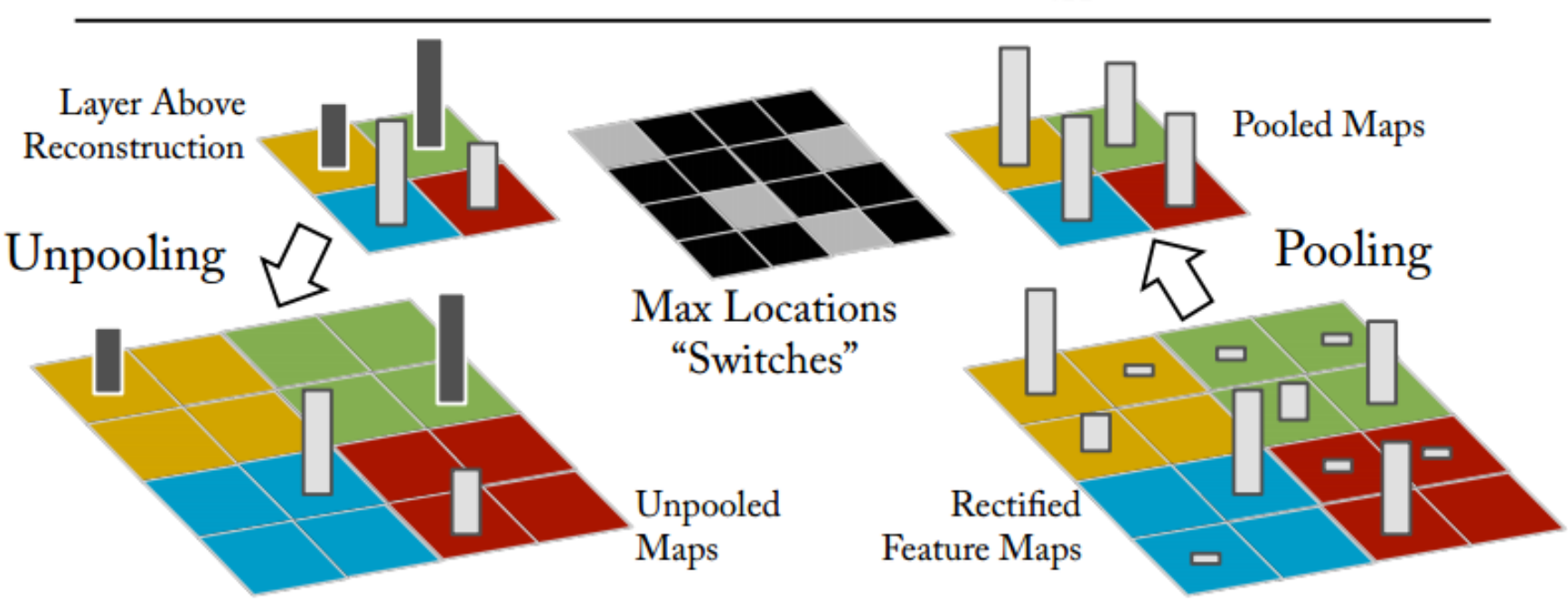


Visualizing and Understanding Convolutional Networks
by Zeiler & Fergus, 2011, 2013 & 2014

DeConvolve Operation

- The convNet uses learned filters to convolve the feature maps from the previous layer
- To approximately invert this, the deconvNet
  - uses transposed version of the same filters
    - Flipping each filter vertically and horizontally
  - But applied to the rectified maps, not the output of the layer beneath

Unpooling Operation



Training details

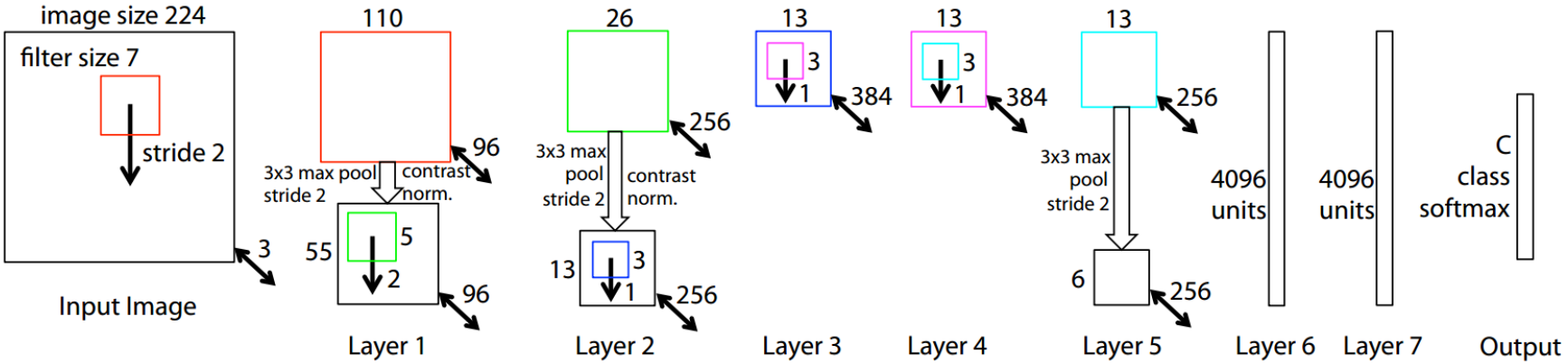
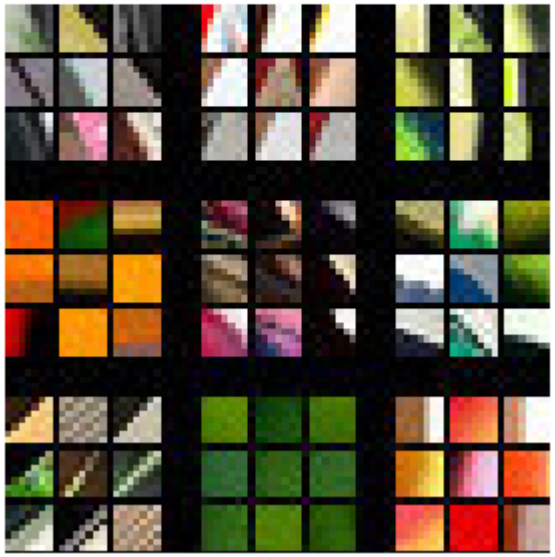


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through a rectified linear function (not shown), (ii) pooled (max within 3x3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55 by 55 element feature maps. Similar operations are repeated in layers 2,3,4,5. The last two layers are fully connected, taking features from the top convolutional layer as input in vector form ($6 \cdot 6 \cdot 256 = 9216$ dimensions). The final layer is a C-way softmax function, C being the number of classes. All filters and feature maps are square in shape.

Visualizing and Understanding Convolutional Networks
by Zeiler & Fergus, 2013 & 2014

Visualizing arbitrary neurons along the way to the top

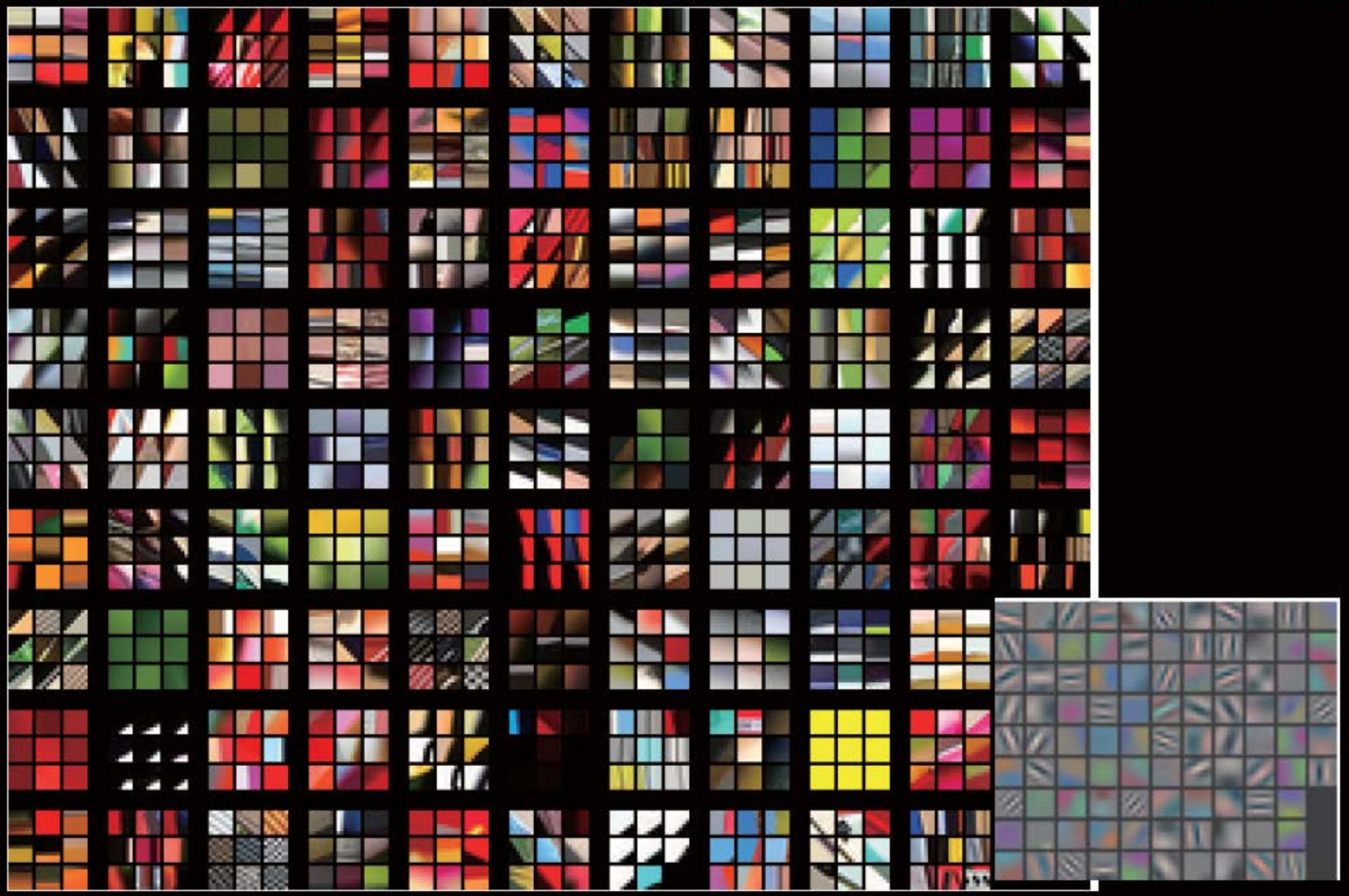
Layer 1



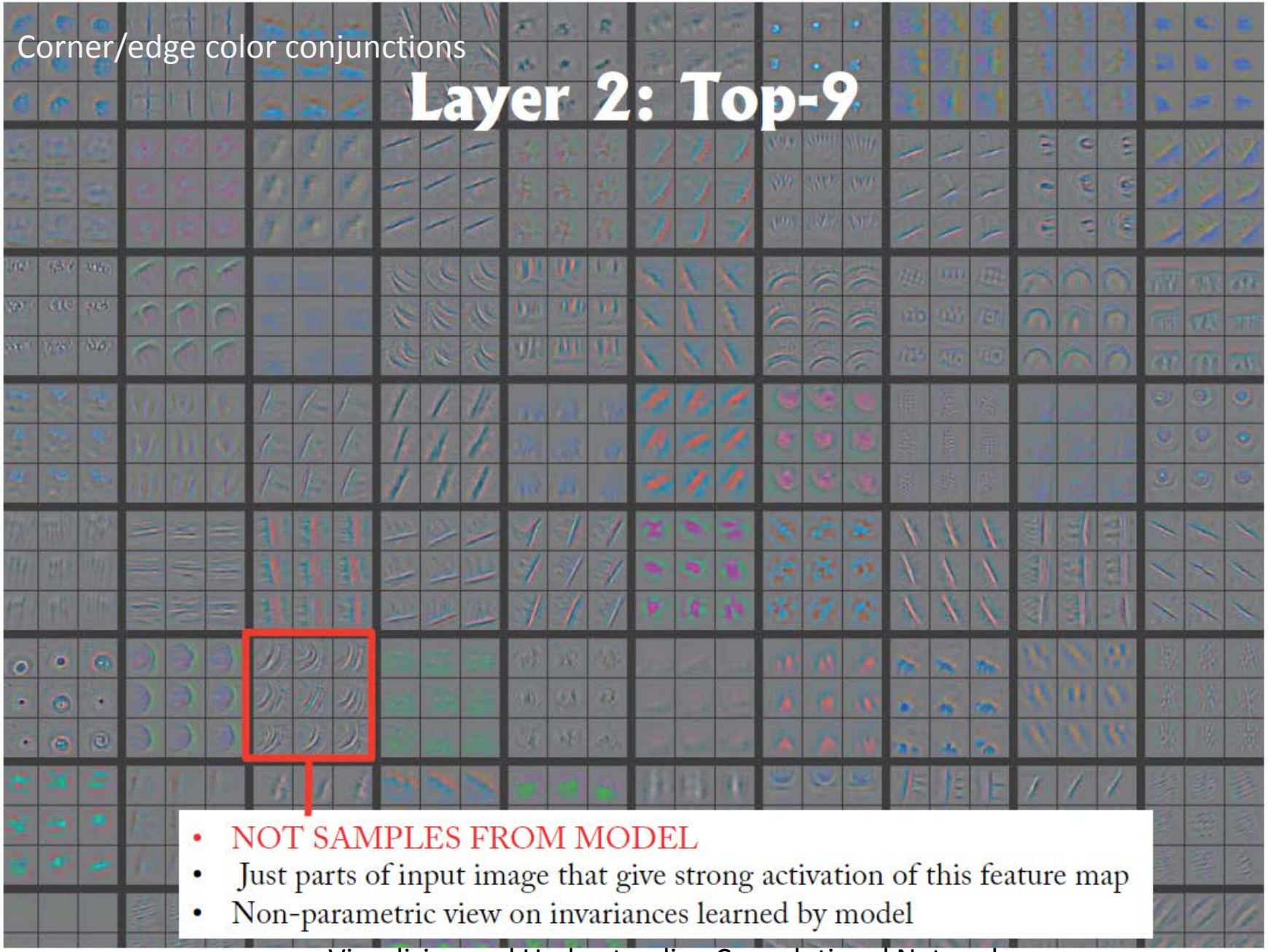
edge

Visualizing and Understanding Convolutional Networks
by Zeiler & Fergus, 2013 & 2014

Layer 1: Top-9 Patches

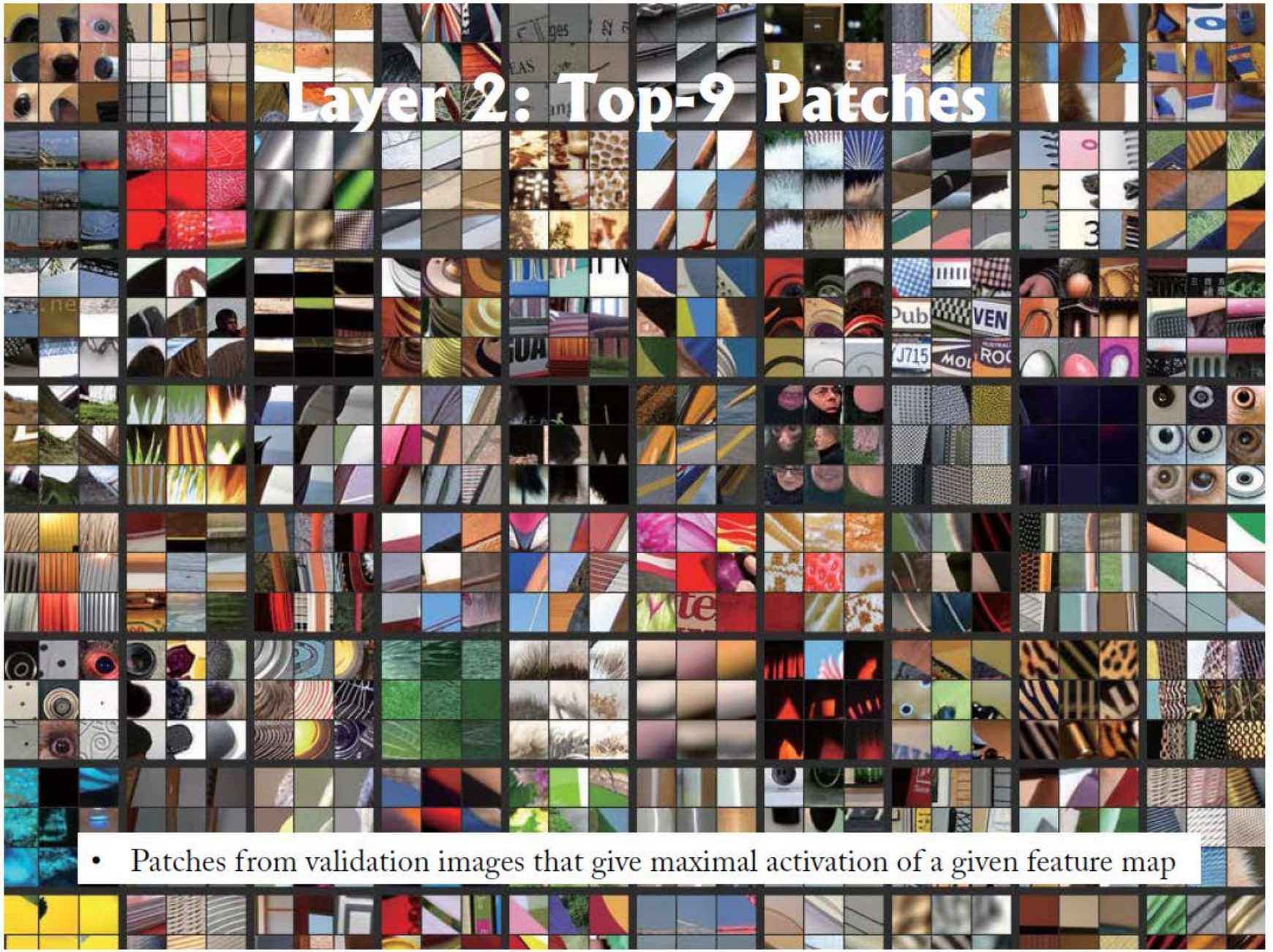


Corner/edge color conjunctions

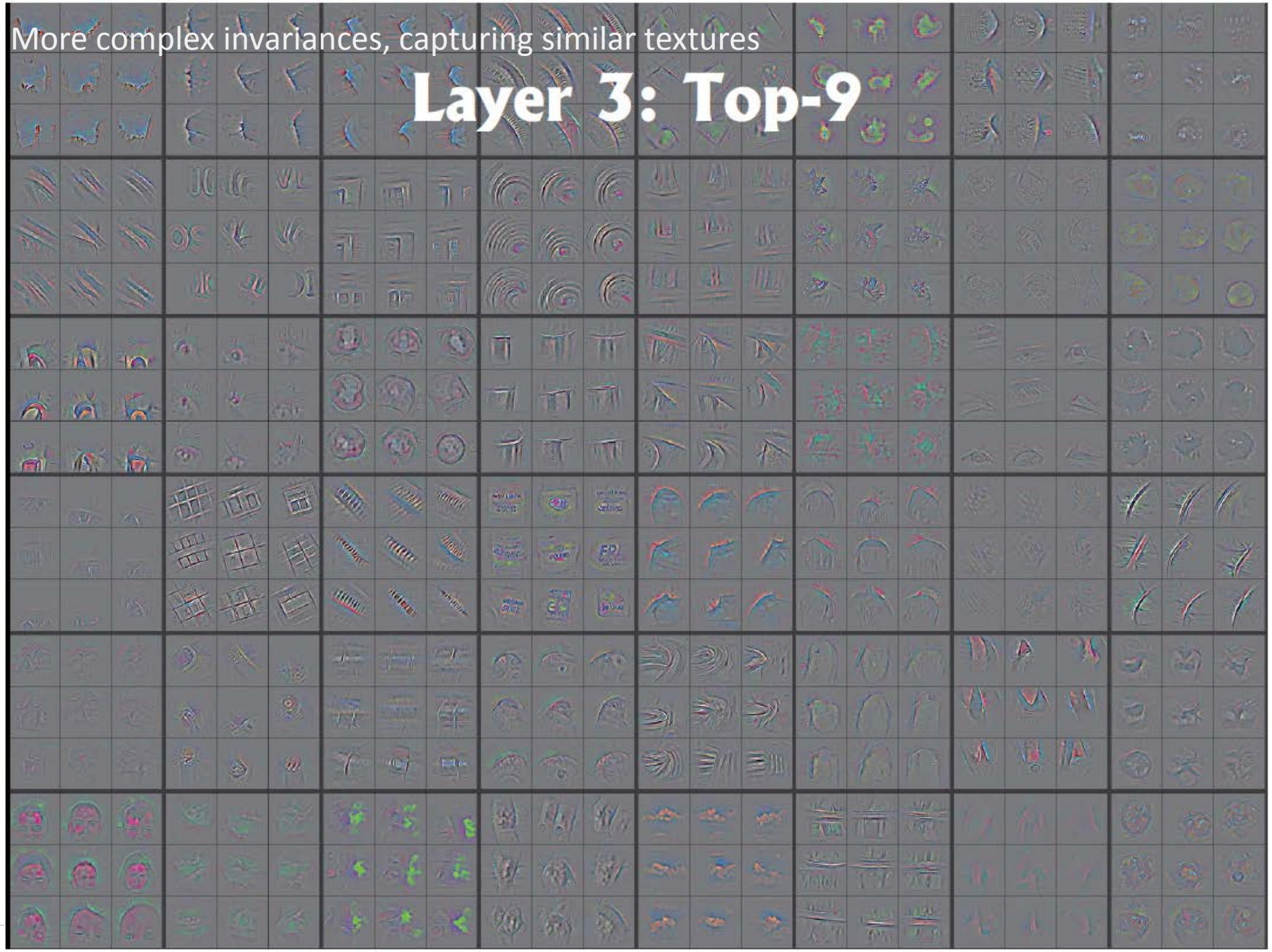


- NOT SAMPLES FROM MODEL
- Just parts of input image that give strong activation of this feature map
- Non-parametric view on invariances learned by model

Visualizing and Understanding Convolutional Networks
by Zeiler & Fergus, 2013 & 2014

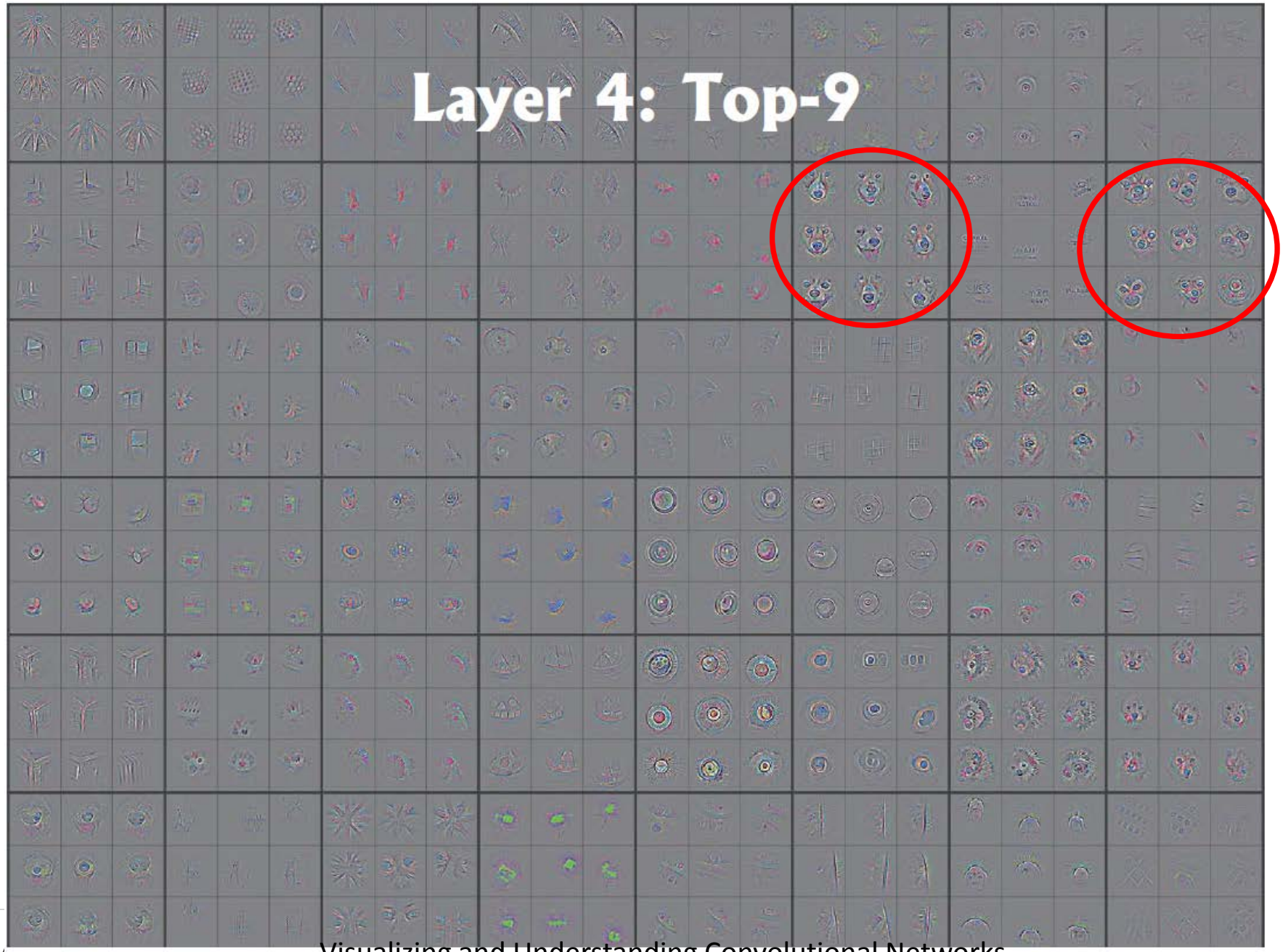

Layer 2: Top-9 Patches
• Patches from validation images that give maximal activation of a given feature map

More complex invariances, capturing similar textures



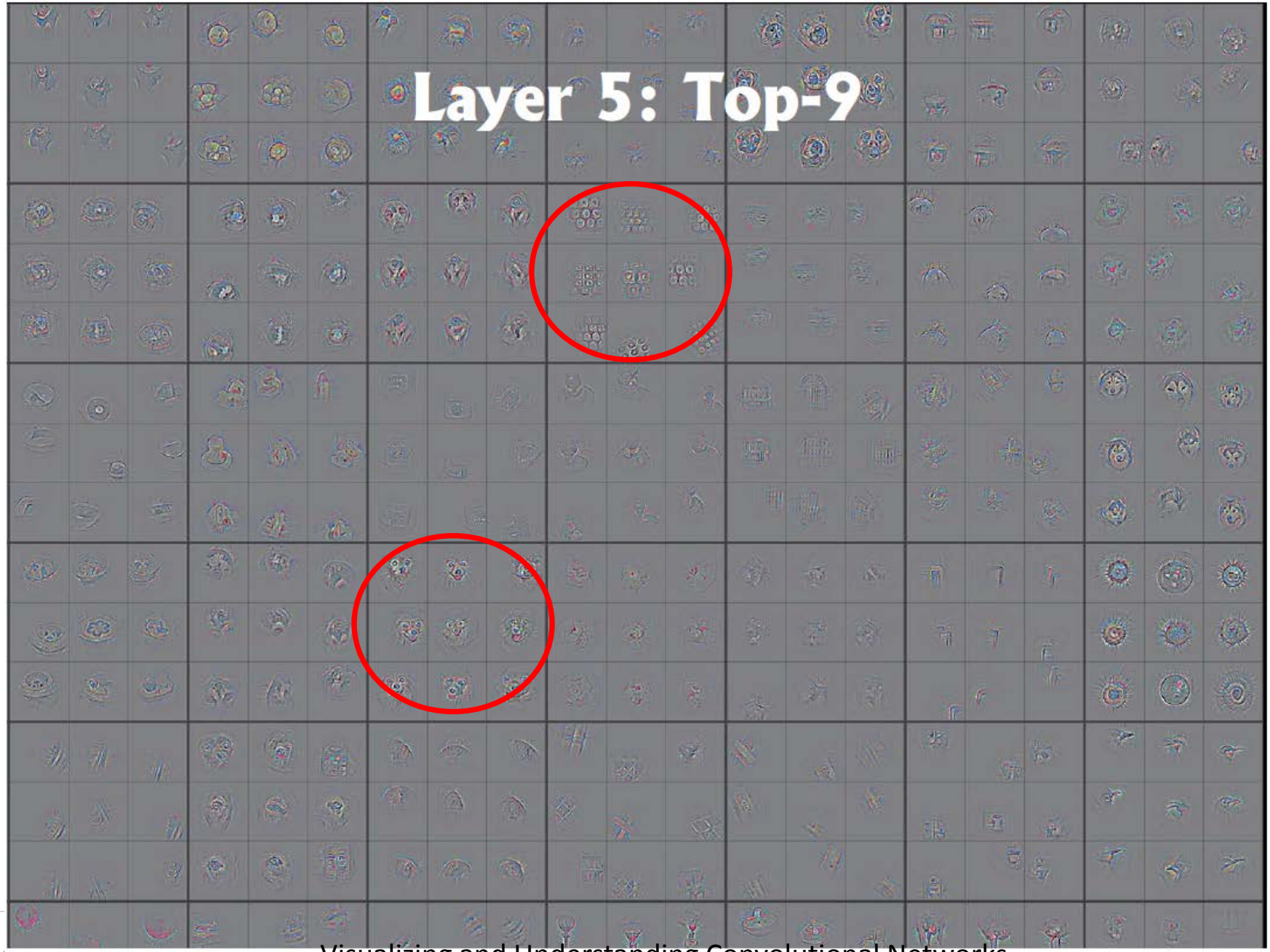
Visualizing and Understanding Convolutional Networks
by Zeiler & Fergus, 2013 & 2014

Significant variation, more class-specific




Layer 4: Top-9 Patches

Entire objects with significant pose variation



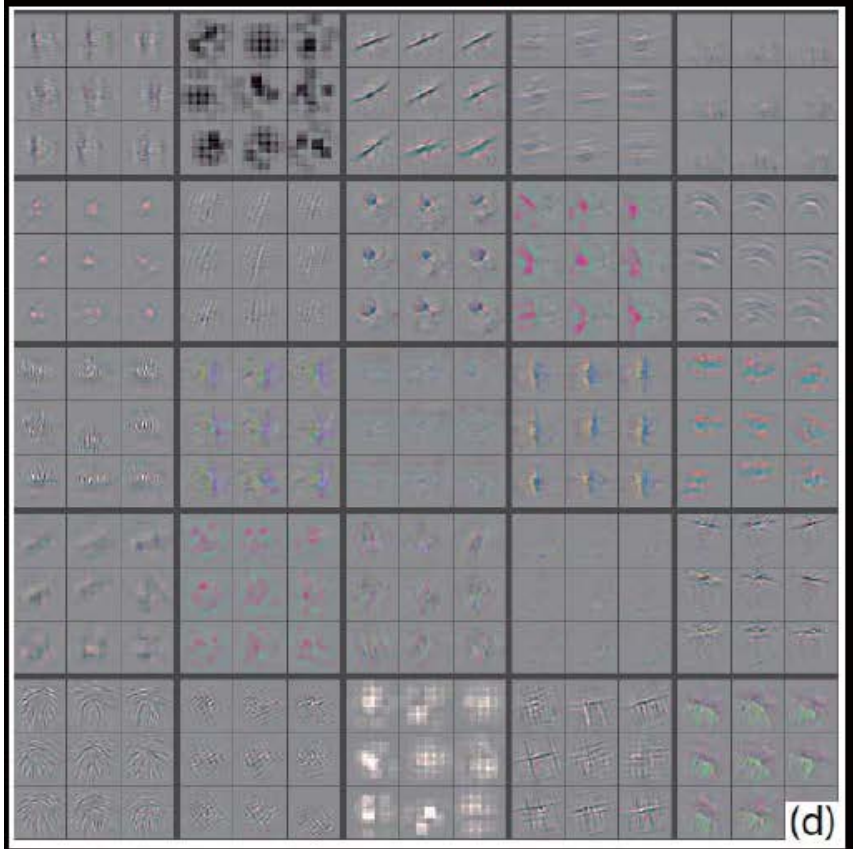

Layer 5: Top-9 Patches

Diagnosing Problems

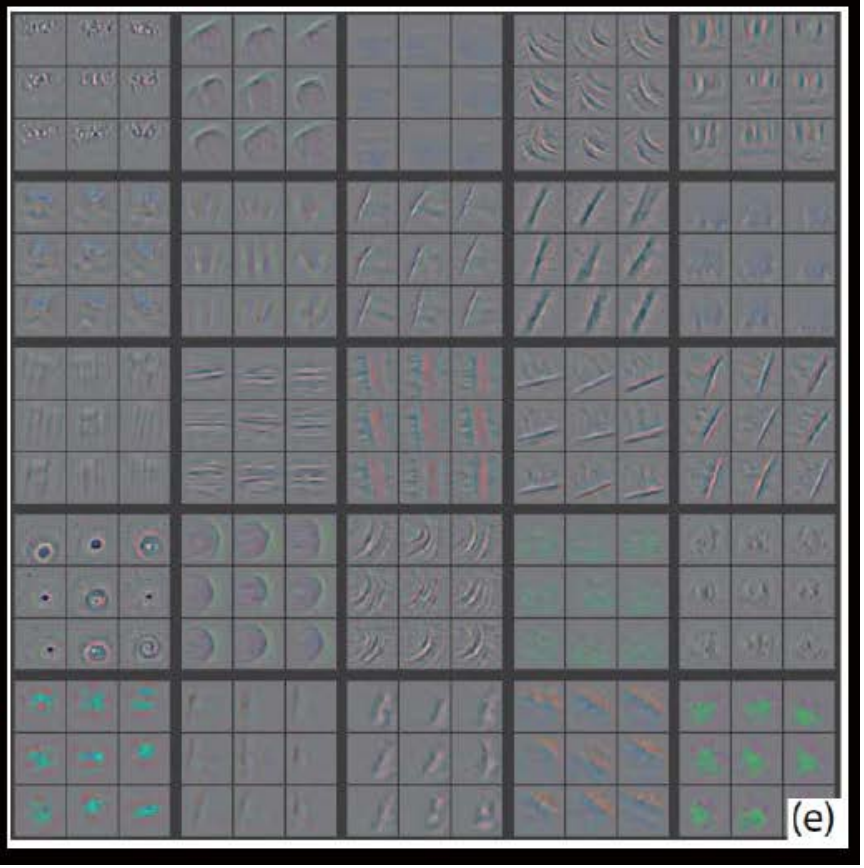
- Visualization of Krizhevsky et al.'s architecture showed some problems with layers 1 and 2
  - Large stride of 4 used
- Alter architecture: smaller stride & filter size
  - Visualizations look better
  - Performance improves

- Layer 2 visualizations

Krizhevsky et al.



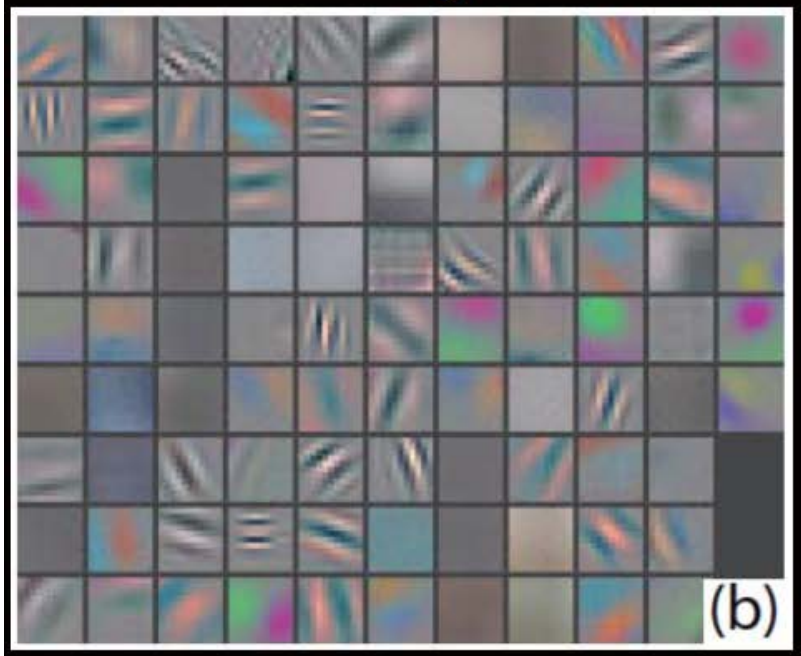
Ours



Comparison with Krizhevsky et al.

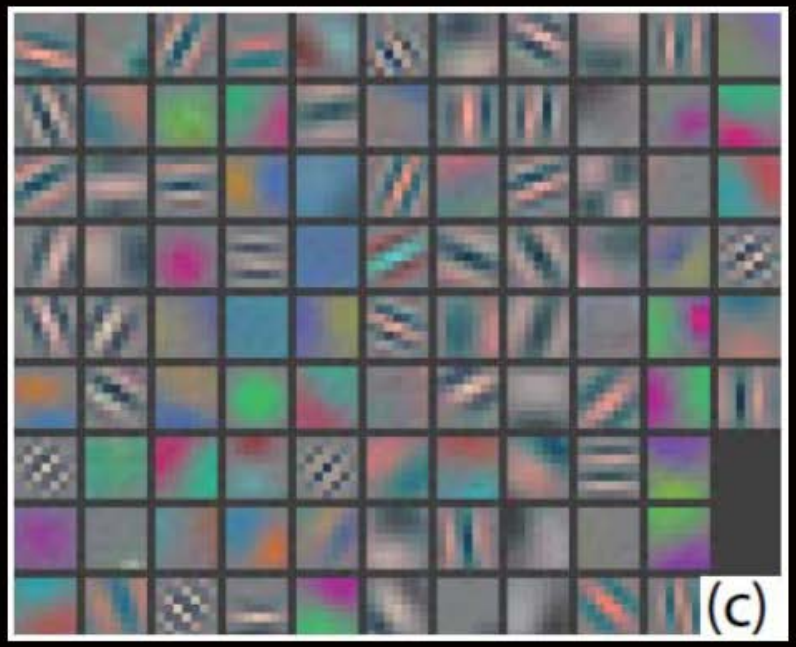
- Layer 1 filters

Krizhevsky et al.



11x11 filters, stride 4

Ours

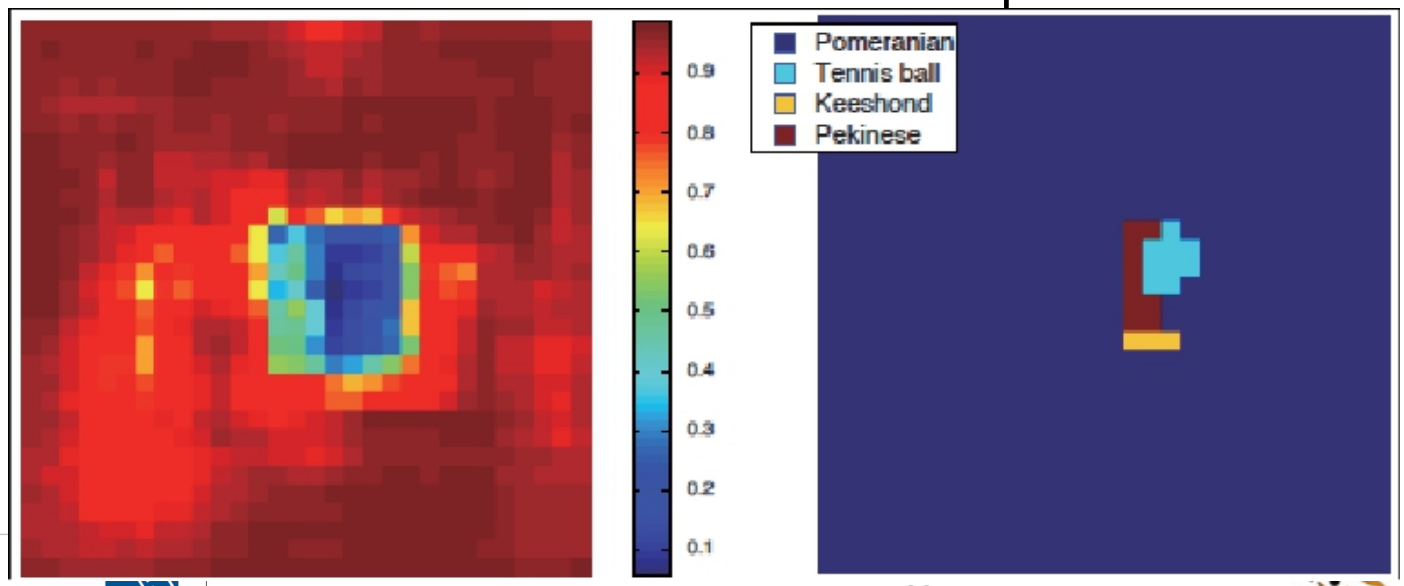


7x7 filters, stride 2

Input image

p(True class)

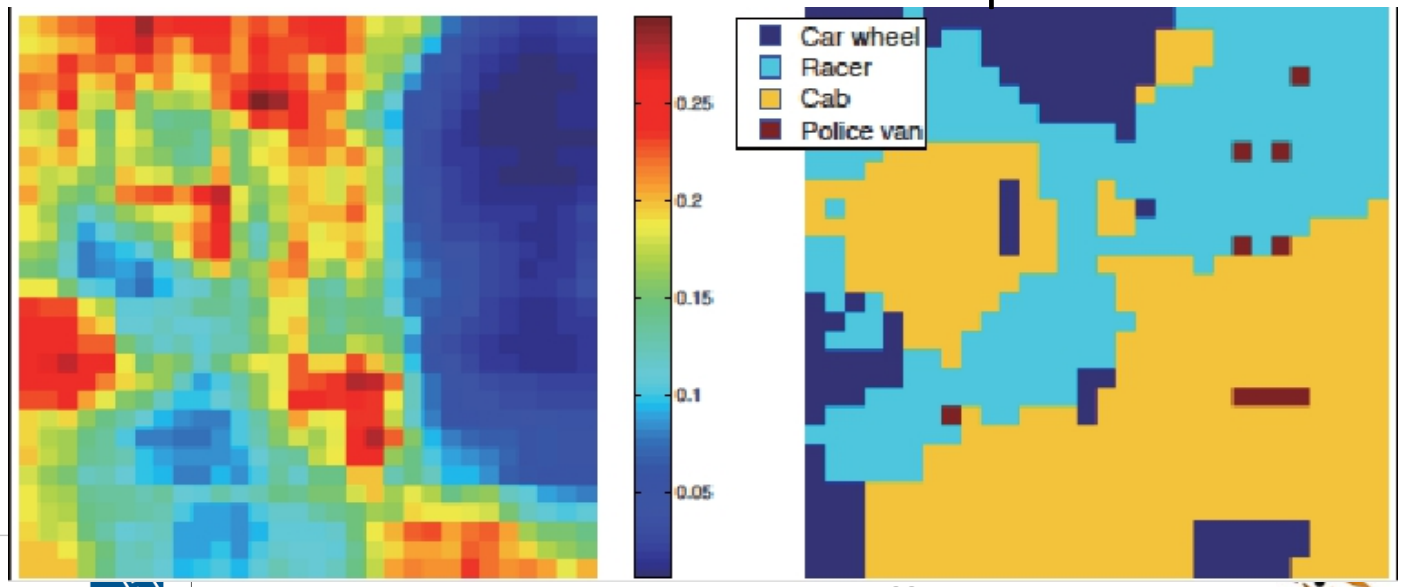
Most probable class



Input image

p(True class)

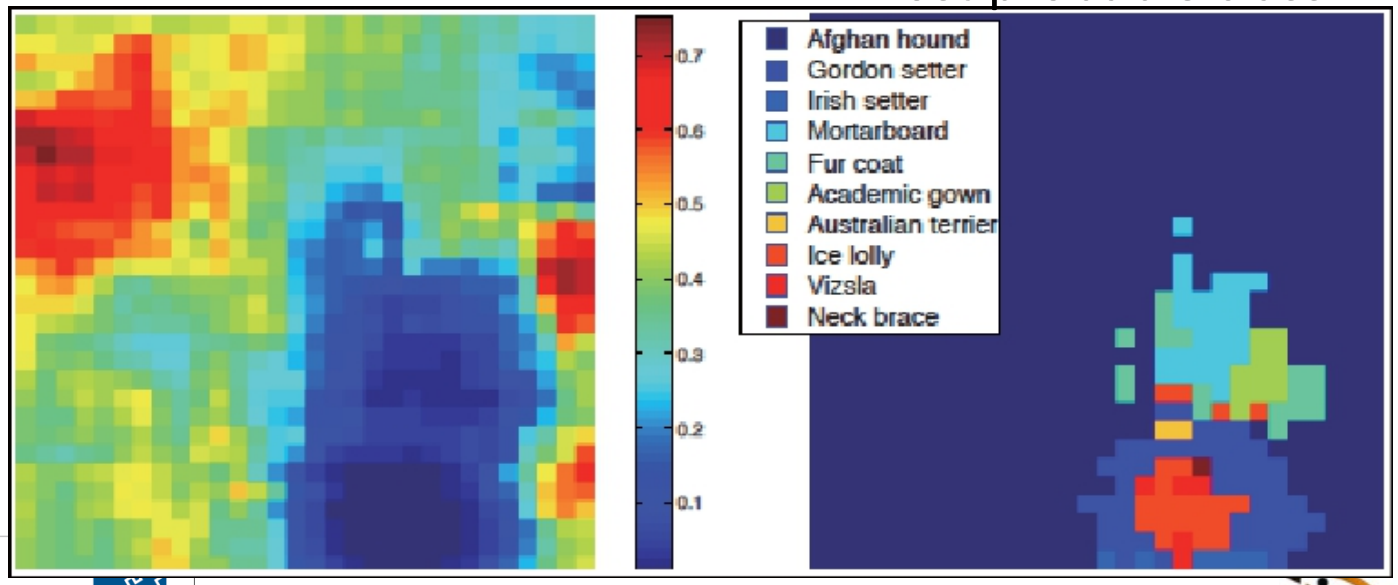
Most probable class

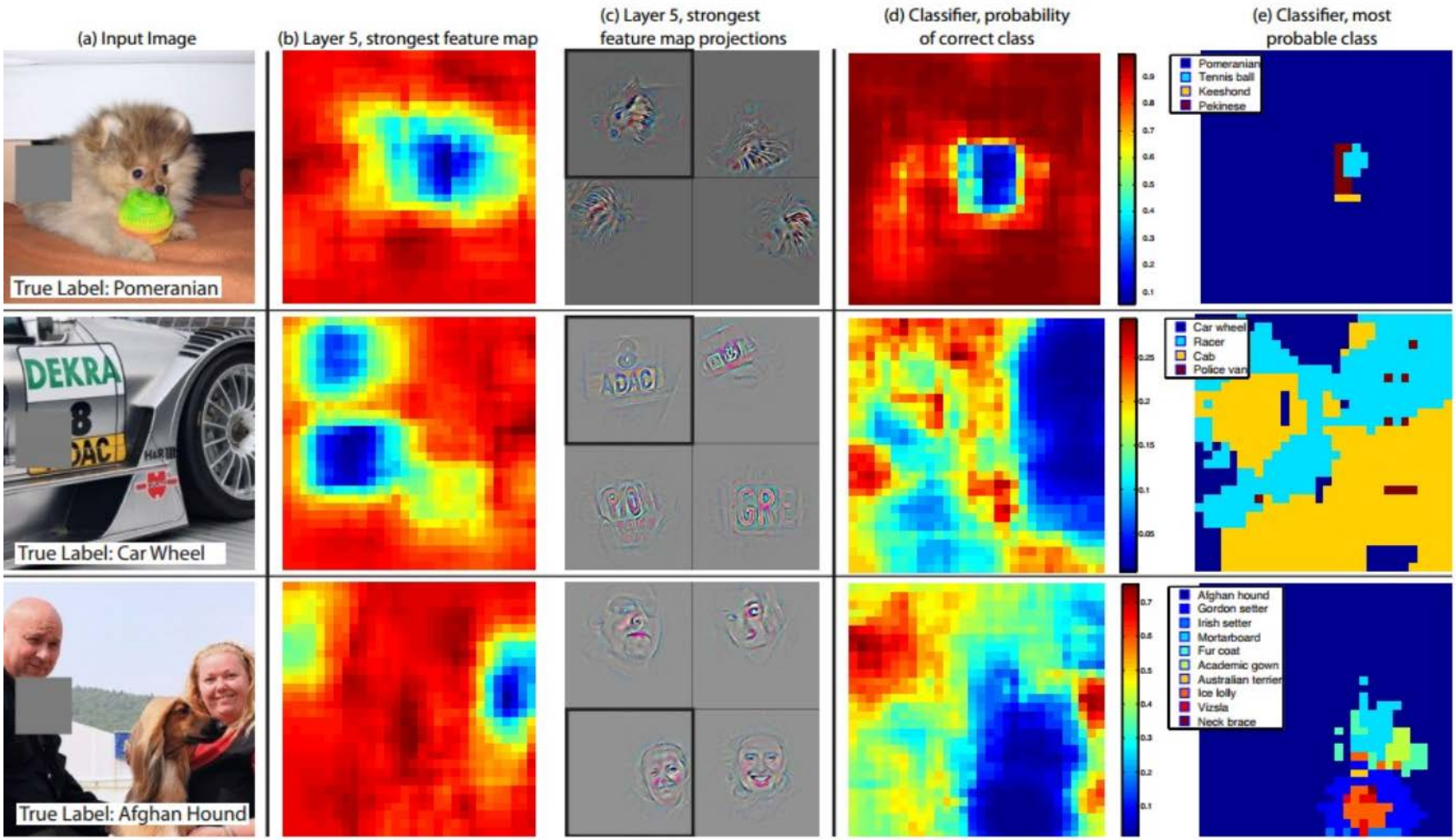


Input image

p(True class)

Most probable class




(a) Input Image
(b) Layer 5, strongest feature map
(c) Layer 5, strongest feature map projections
(d) Classifier, probability of correct class
(e) Classifier, most probable class
True Label: Pomeranian
Pomeranian
Tennis ball
Keeshond
Pekinese
DEKRA
True Label: Car Wheel
Car wheel
Racer
Cab
Police van
True Label: Afghan Hound
Afghan hound
Gordon setter
Irish setter
Mortarboard
Fur coat
Academic gown
Australian terrier
Ice lolly
Vizsla
Neck brace

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

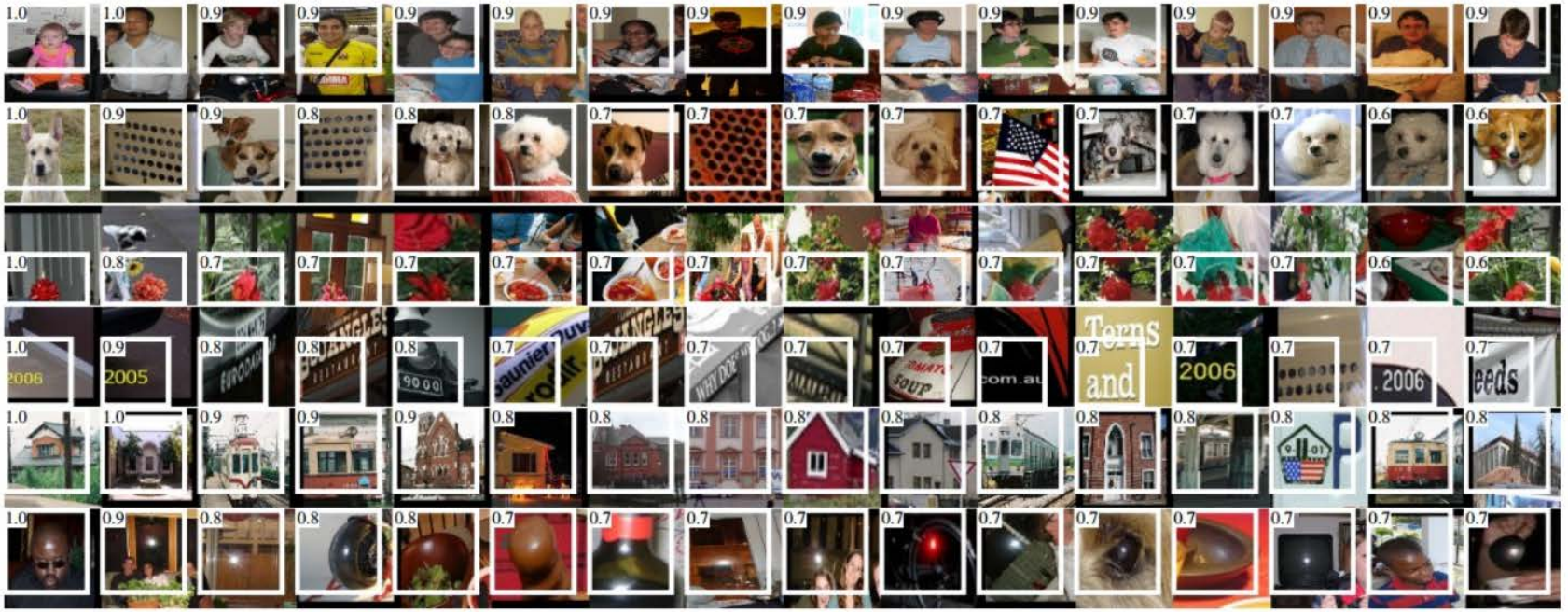


Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).

Visualizing the input maximizing the scores

- DeconvNet based visualization
- Optimization based visualization
- Reconstruct the original image?

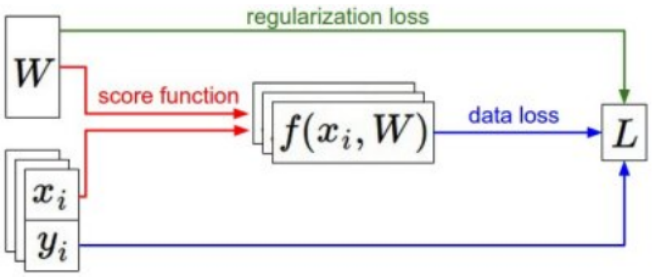
Gradient based ConvNet visulisation

1. Find images that maximize some class score:

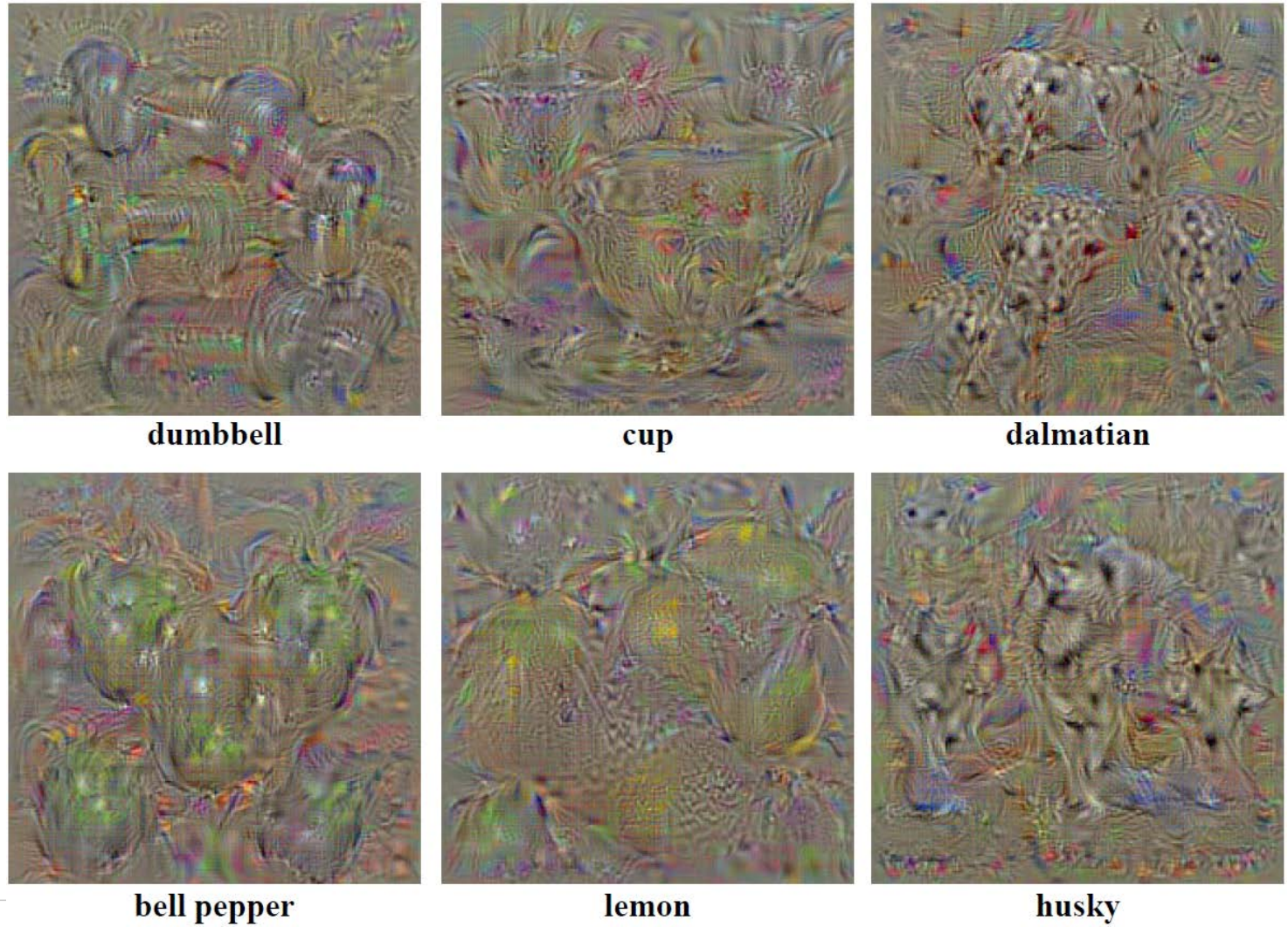
$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

Score for class c
(before Softmax)

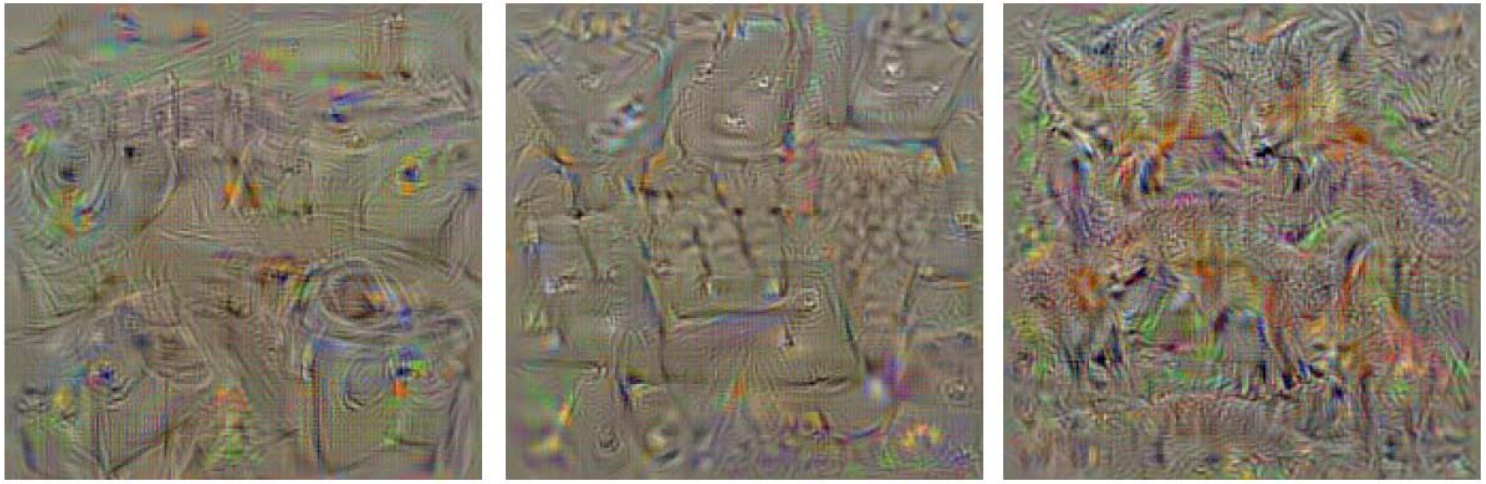
Remember:



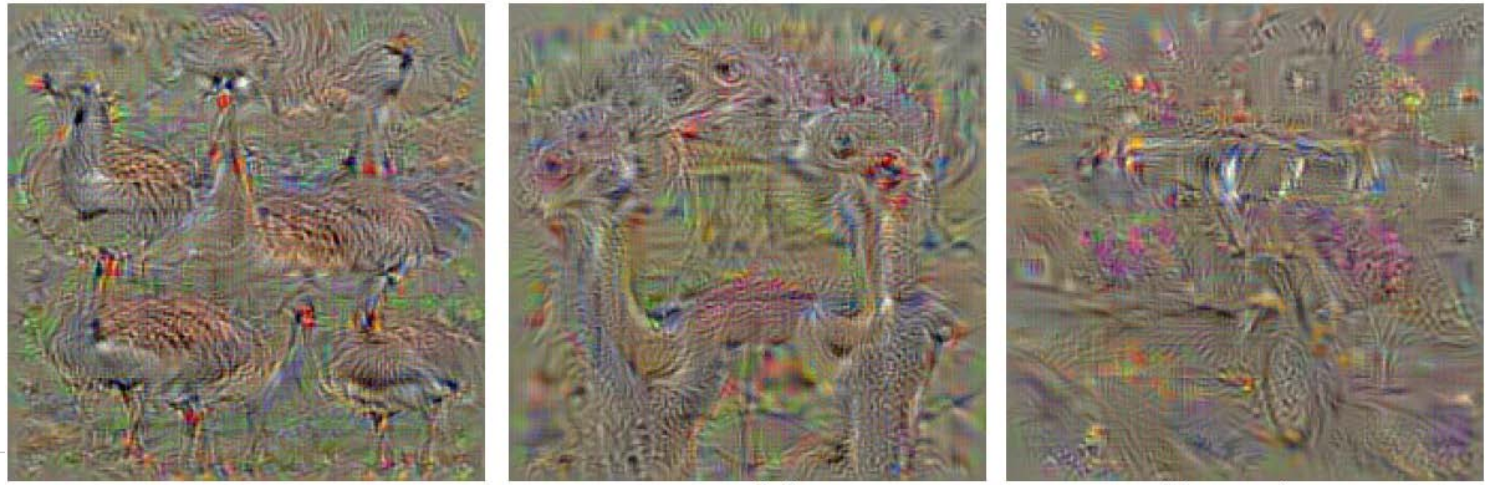
Gradient based ConvNet visualization



Gradient based ConvNet visualization



washing machine | computer keyboard | kit fox



goose | ostrich | limousine

Image Specific Class Saliency Visualization

2. Visualize the Data gradient:

$$S_c(I) \approx w^T I + b, \qquad w = \left.\frac{\partial S_c}{\partial I}\right|_{I_0}$$



M = ?

(note that the gradient on data has three channels. Here they visualize M, s.t.:

$$M_{ij} = \max_c |w_{h(i,j,c)}|$$

(at each pixel take abs val, and max over channels)

Image Specific Class Saliency Visualization

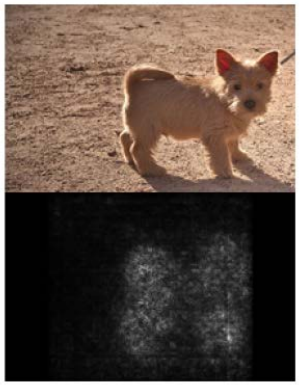
2. Visualize the Data gradient:

$$S_c(I) \approx w^T I + b, \qquad w = \frac{\partial S_c}{\partial I}\Big|_{I_0}$$

(note that the gradient on data has three channels. Here they visualize M, s.t.:

$$M_{ij} = \max_c |w_{h(i,j,c)}|$$

(at each pixel take abs val, and max over channels)

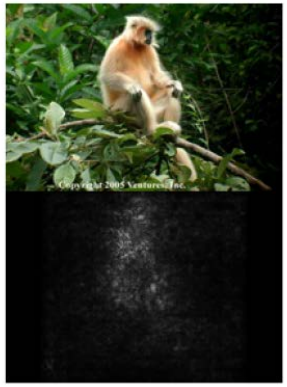
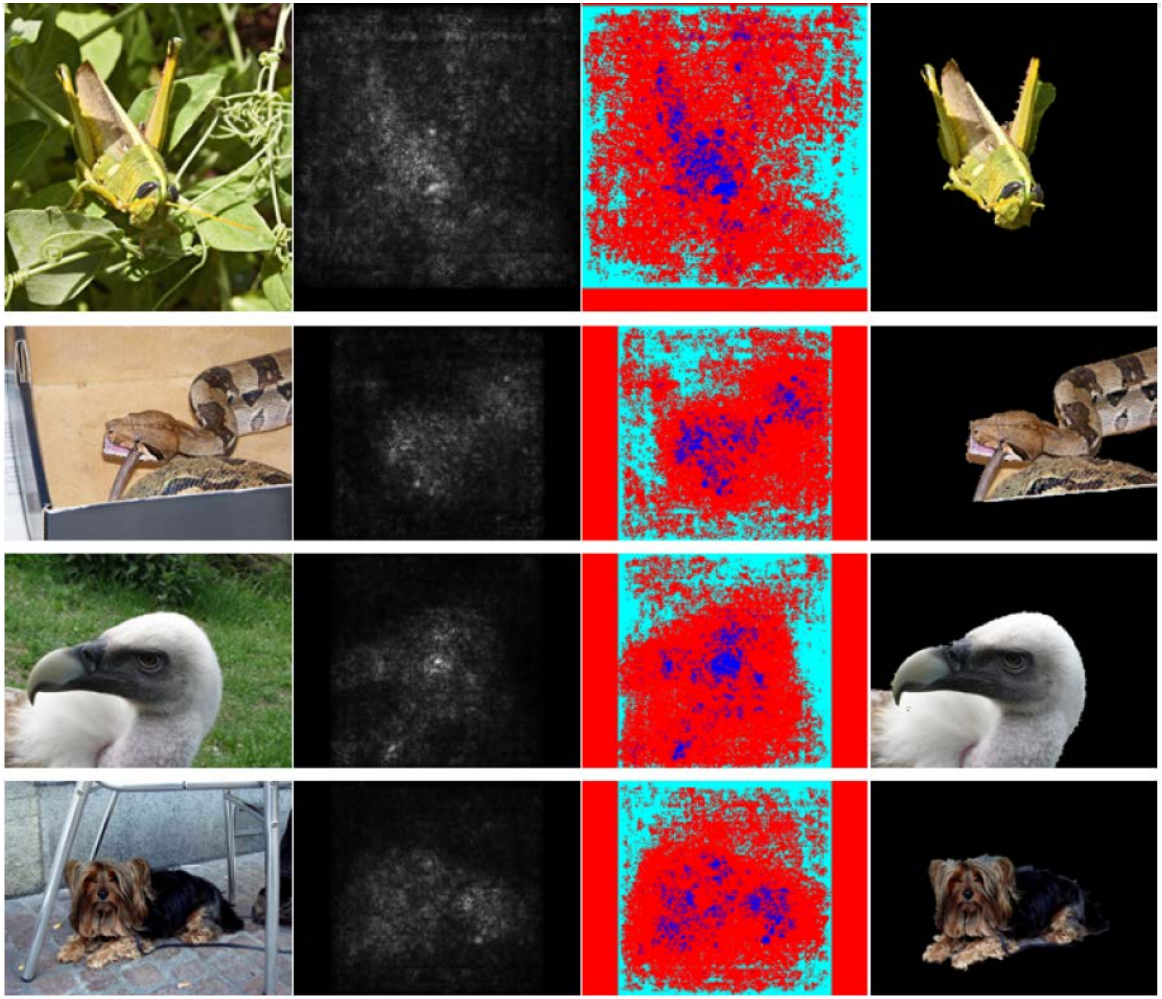




Image Specific Class Saliency Visualization

- Graph-cut based Segmentation



Slides from Feifei Li & Andrei Karpathy

Other methods

- Erhan et al. [Tech Report 2009]
- Le et al. [NIPS 2010]
- Depend on initialization
- Model invariance with Hessian about (locally) optimal stimulus

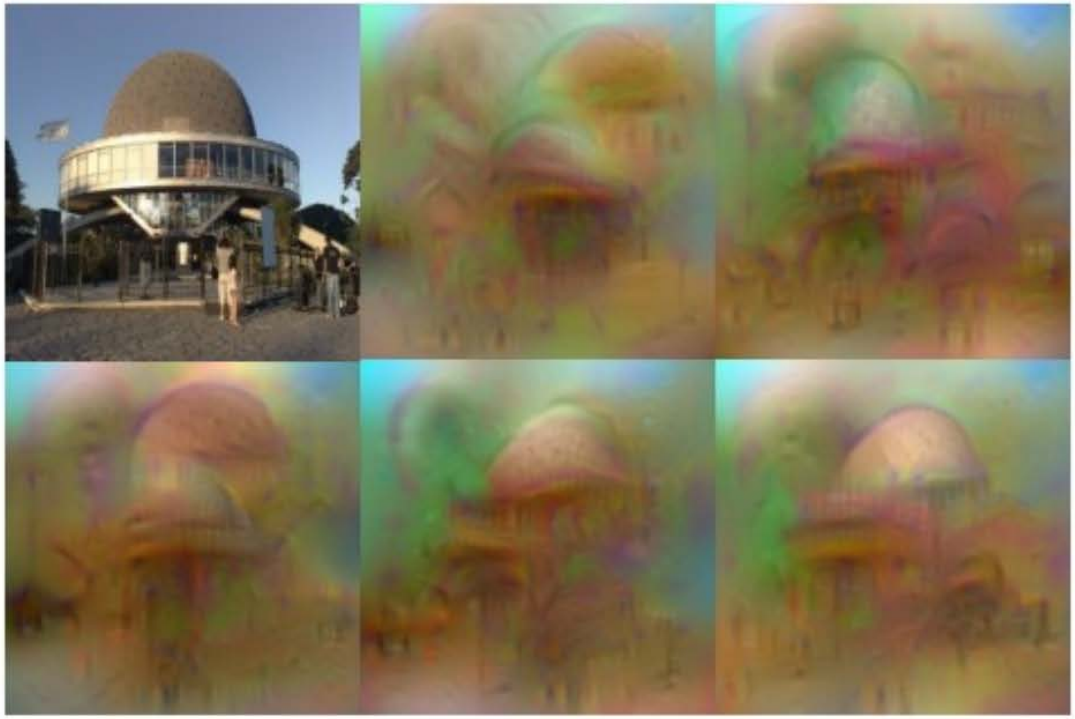
Visualizing the input maximizing the scores

- DeconvNet based visualization
- Optimization based visualization
- Reconstruct the original image?

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

Inverse of CNN image representation

original image



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

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

Find an image such that:

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

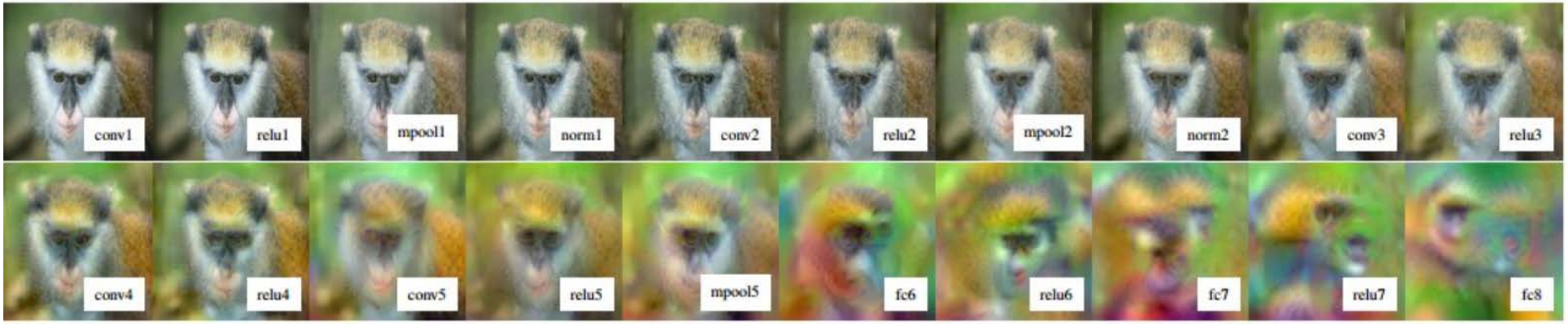
$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

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

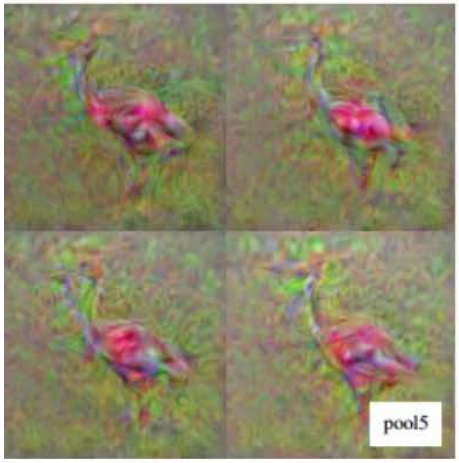
Solve using SGD + Momentum

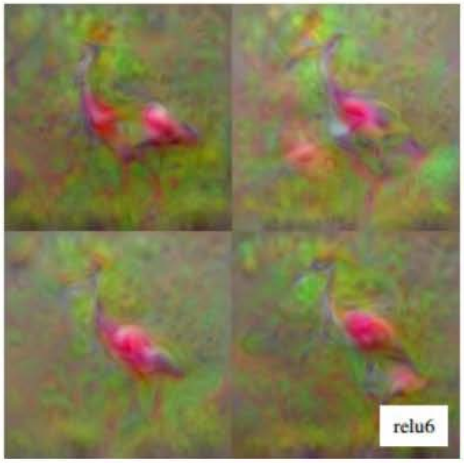
Reconstructions from the representation after last last pooling layer (immediately before the first Fully Connected layer)

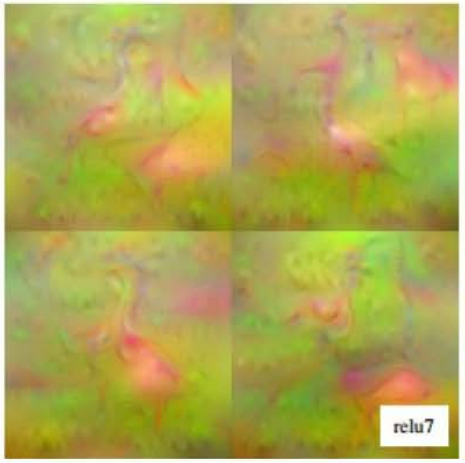
Reconstructions from intermediate layers

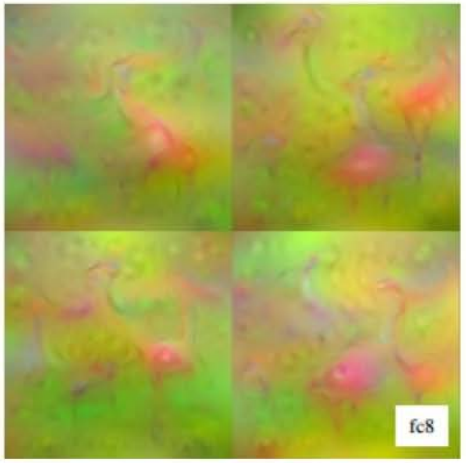


Multiple reconstructions. Images in quadrants all “look” the same to the CNN (same code)



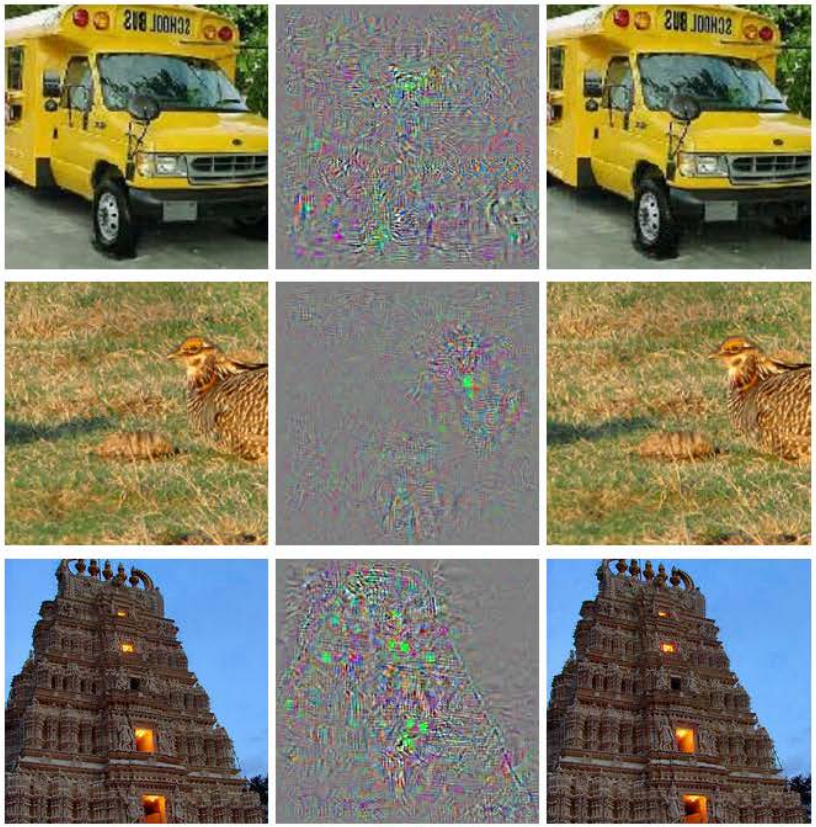


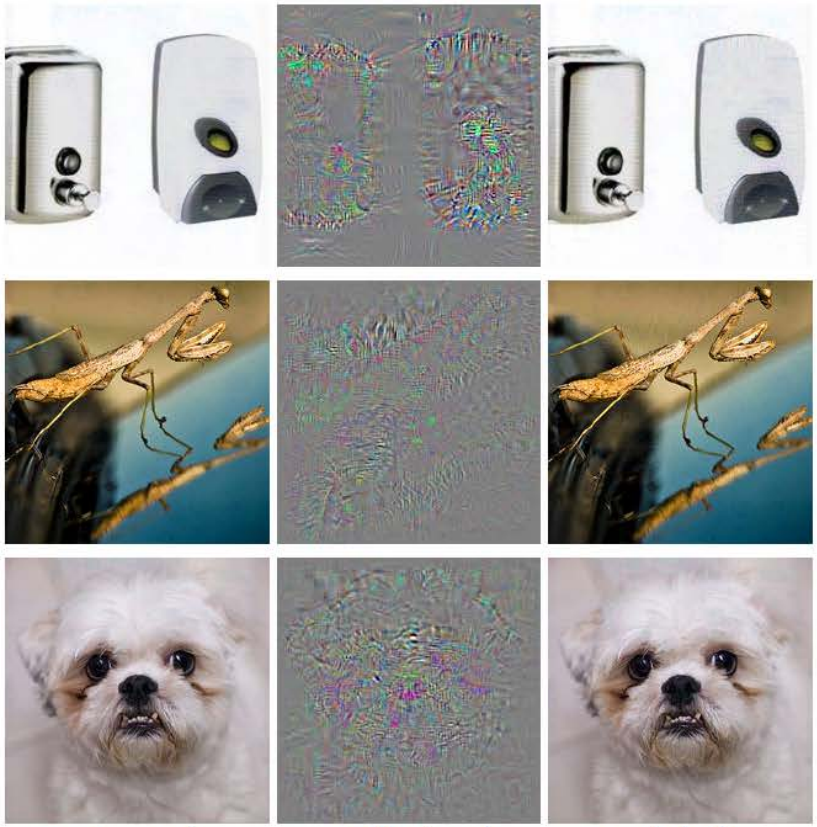




- We can pose an optimization over the input image to maximize any class score. That seems useful.
- Question: Can we use this to “fool” ConvNets?

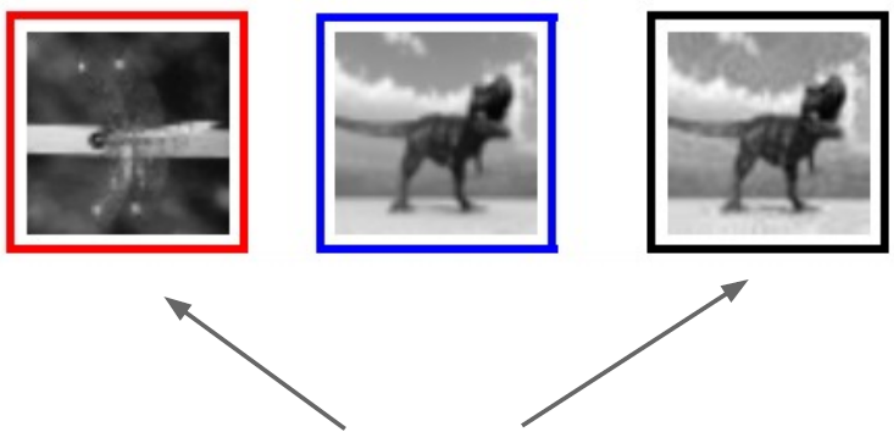
Intriguing properties of neural networks
[Szegedy et al.]

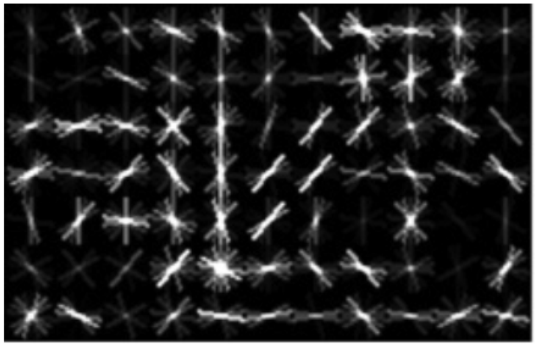




These kinds of results were around even before ConvNets…

Exploring the Representation Capabilities of the HOG Descriptor
[Tatu et al., 2011]

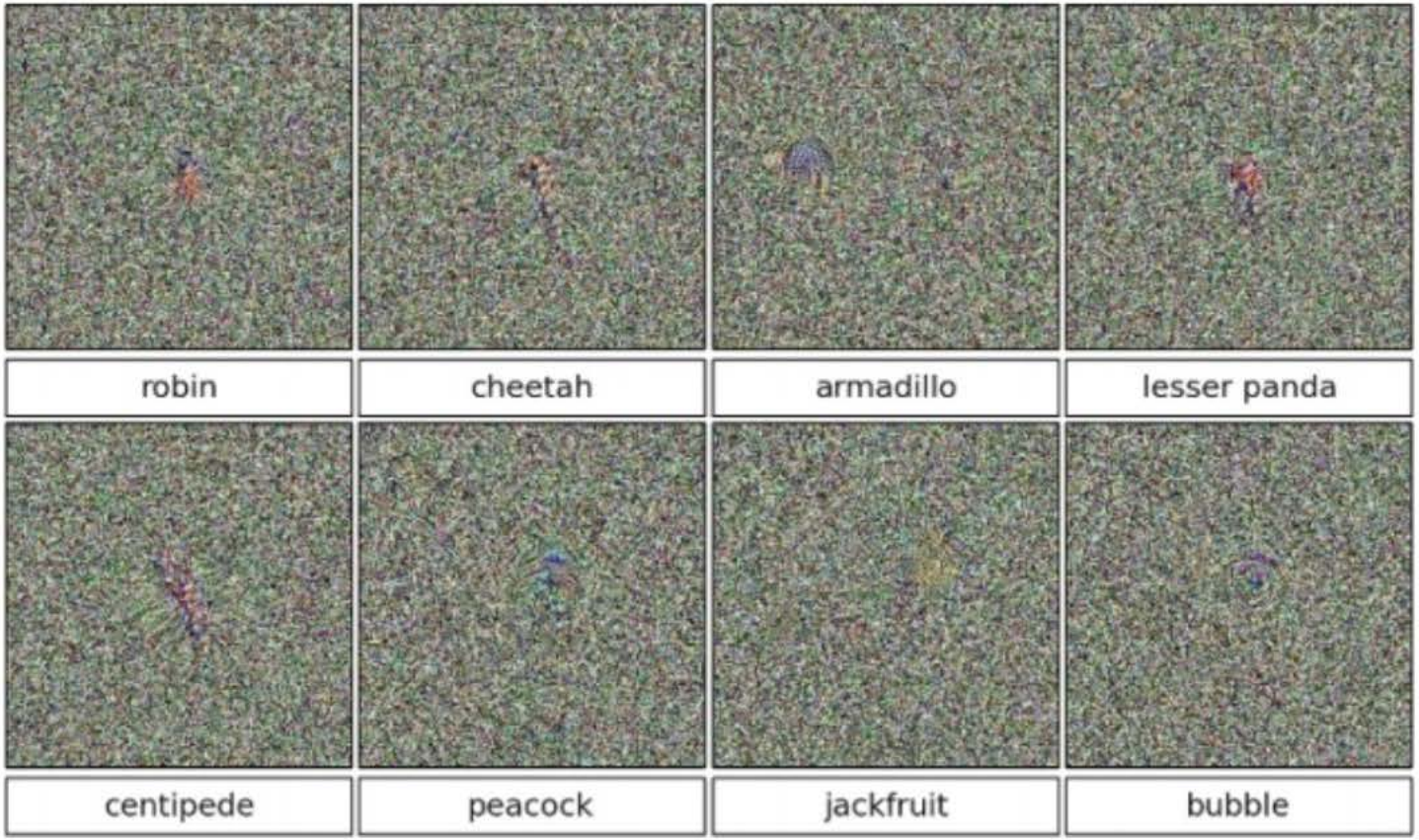




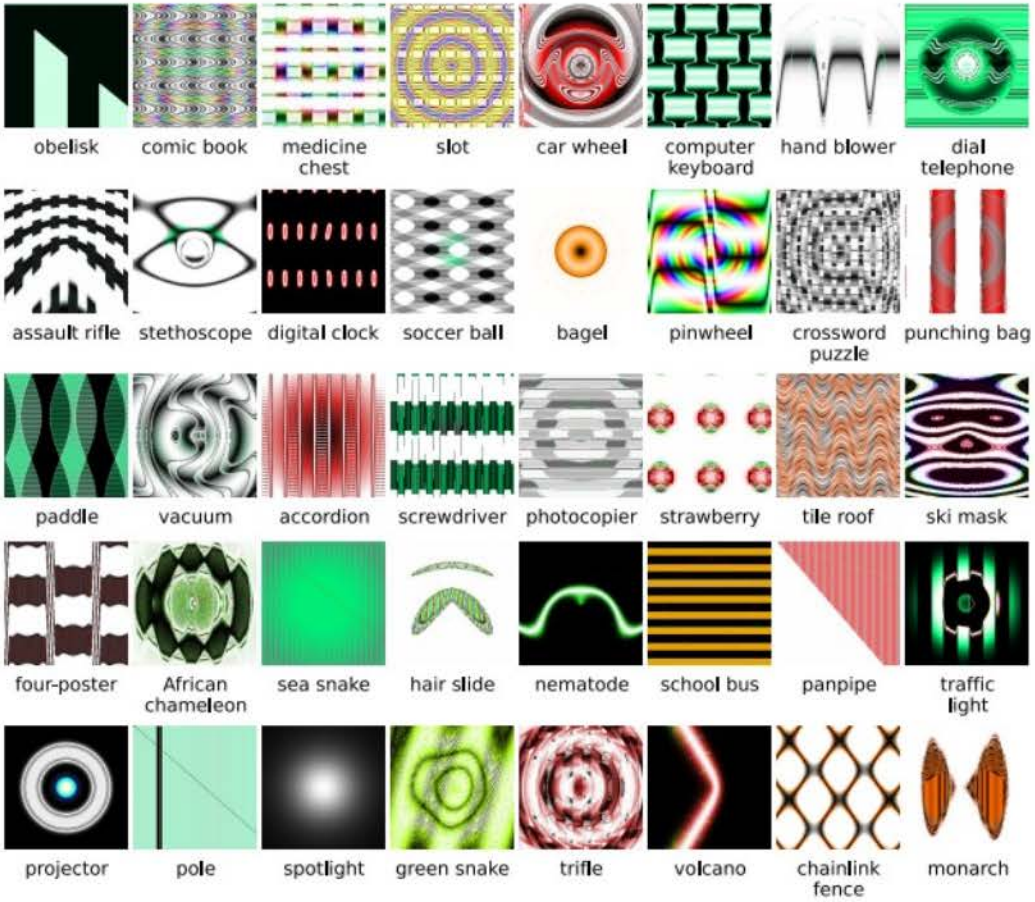
Identical HOG represention

Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [Nguyen, Yosinski, Clune]

>99.6% confidences



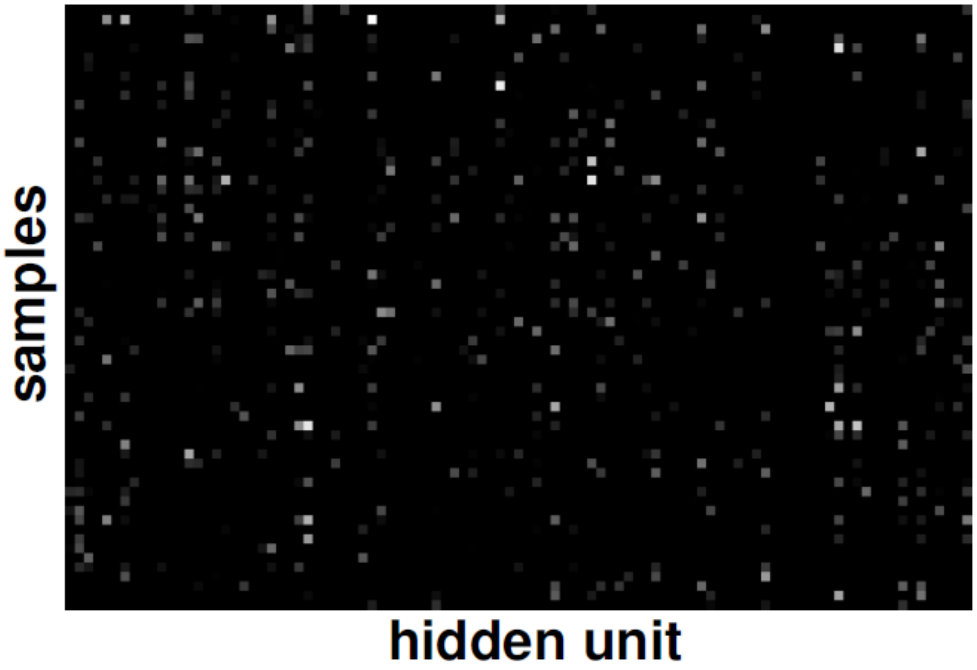
Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images [Nguyen, Yosinski, Clune]



>99.12% confidences

OTHER THINGS GOOD TO KNOW

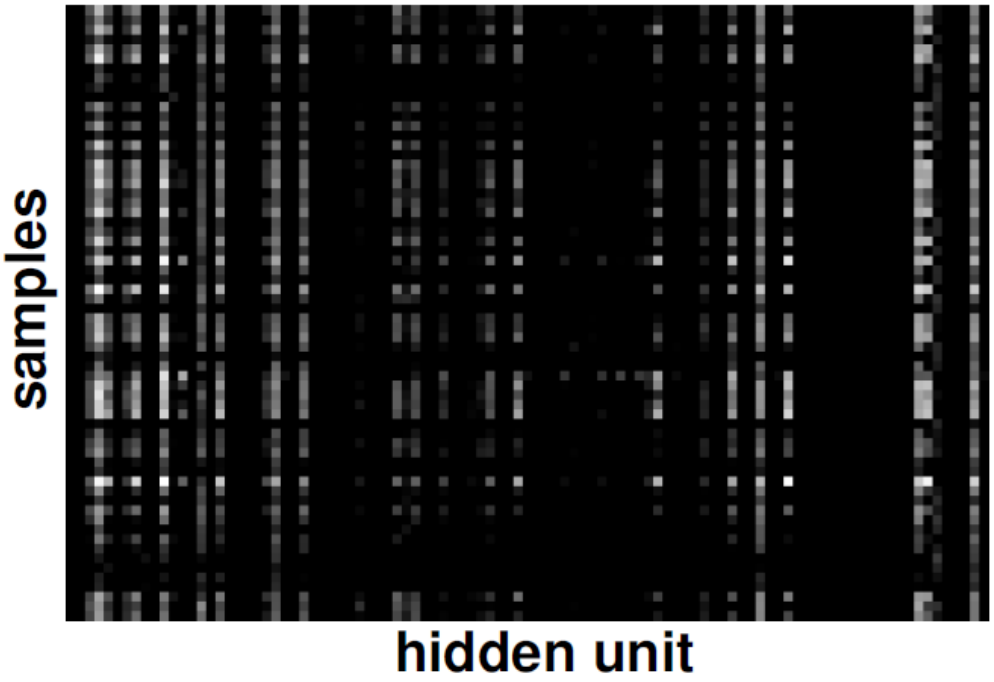
- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.



Good training: hidden units are sparse across samples and across features.

OTHER THINGS GOOD TO KNOW

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.



Bad training: many hidden units ignore the input and/or exhibit strong correlations.

Summary

- We get insight into the CNN structure
  - How different layers contribute to the final performance
- We looked at several works that try to visualize how ConvNets work and what they learn
  - Code space
  - Filter Visualization
  - Finding the input maximizing the scores