High Level Computer Vision

Intro to Deep Learning for Computer Vision

Bernt Schiele - schiele@mpi-inf.mpg.de
Mario Fritz - mfritz@mpi-inf.mpg.de

https://www.mpi-inf.mpg.de/hlcv

most slides from: Rob Fergus & Marc'Aurelio Ranzato

Deep Learning for Computer Vision

NIPS 2013 Tutorial

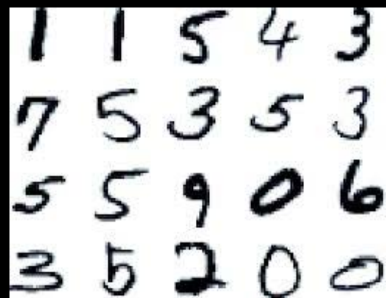
Rob Fergus

Dept. of Computer Science
New York University

Overview

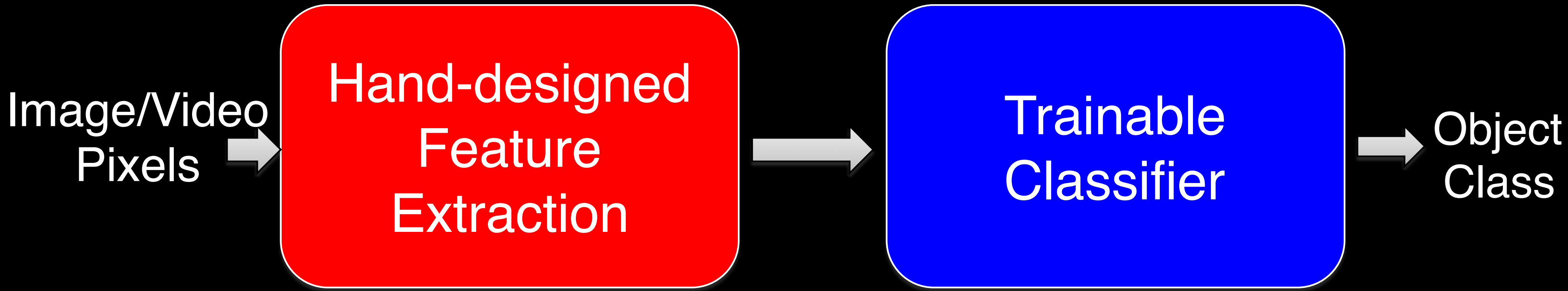
- Primarily about object recognition, using supervised ConvNet models
- Focus on natural images
  - Rather than digits
  - Classification & Detection
- Brief discussion of other vision problems

instead of



Motivation

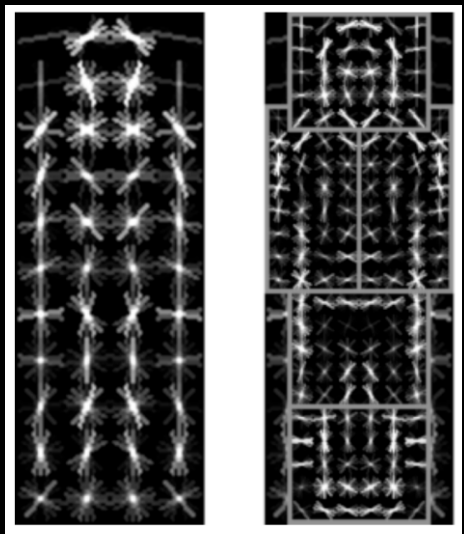
Existing Recognition Approach



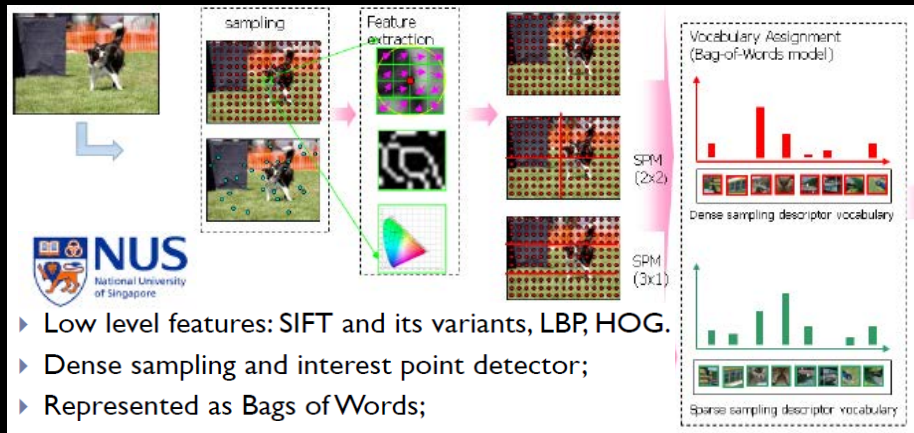
- Features are not learned
- Trainable classifier is often generic (e.g. SVM)

Motivation

- Features are key to recent progress in recognition
- Multitude of hand-designed features currently in use
  - SIFT, HOG, LBP, MSER, Color-SIFT………….
- Where next? Better classifiers? Or keep building more features?



Felzenszwalb, Girshick, McAllester and Ramanan, PAMI 2007

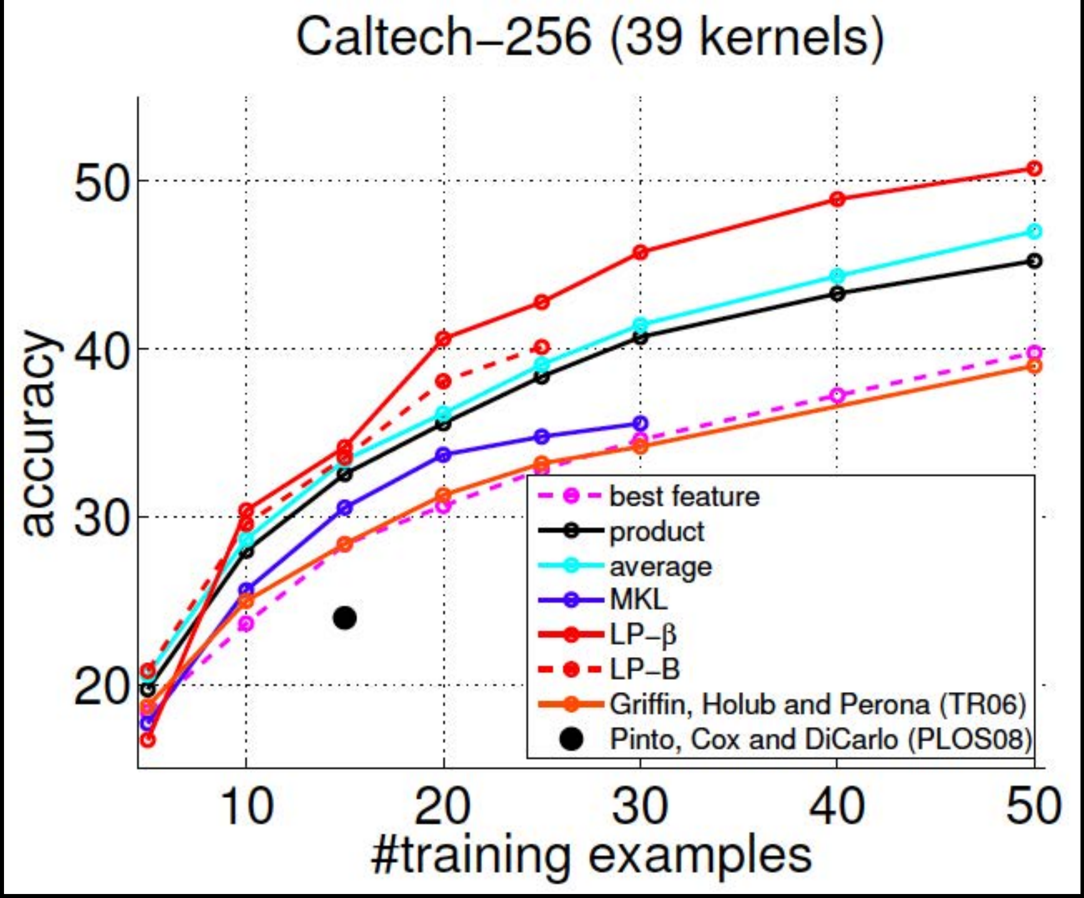


Yan & Huang
(Winner of PASCAL 2010 classification competition)

Hand-Crafted Features

- LP-β Multiple Kernel Learning (MKL)
  - Gehler and Nowozin, On Feature Combination for Multiclass Object Classification, ICCV'09
- 39 different kernels
  - PHOG, SIFT, V1S+, Region Cov. Etc.
- MKL only gets few % gain over averaging features

→ Features are doing the work

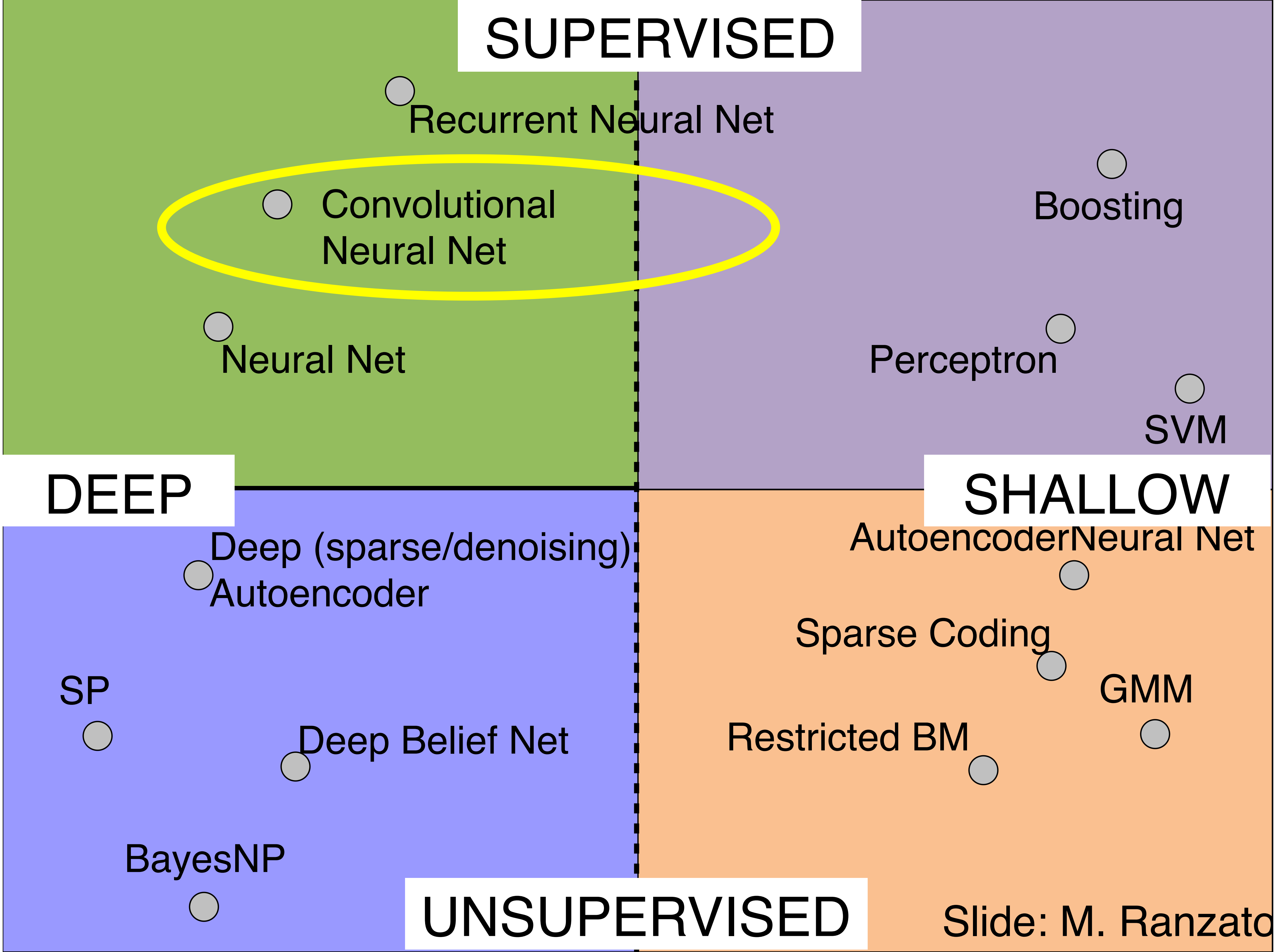


What about Learning the Features?

- Perhaps get better performance?
- Deep models: hierarchy of feature extractors
- All the way from pixels → classifier
- One layer extracts features from output of previous layer

- Train all layers jointly

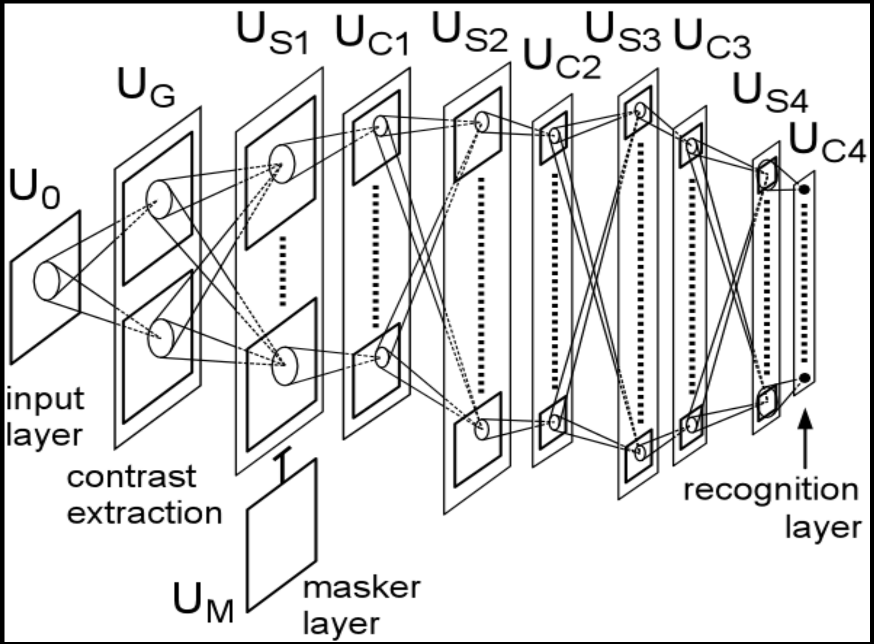
Deep Learning


SUPERVISED
Recurrent Neural Net
Convolutional
Neural Net
Boosting
Neural Net
Perceptron
SVM
DEEP
SHALLOW
Deep (sparse/denoising)
Autoencoder
AutoencoderNeural Net
Sparse Coding
GMM
SP
Deep Belief Net
Restricted BM
BayesNP
UNSUPERVISED
Slide: M. Ranzato

Multistage Hubel&Wiesel Architecture

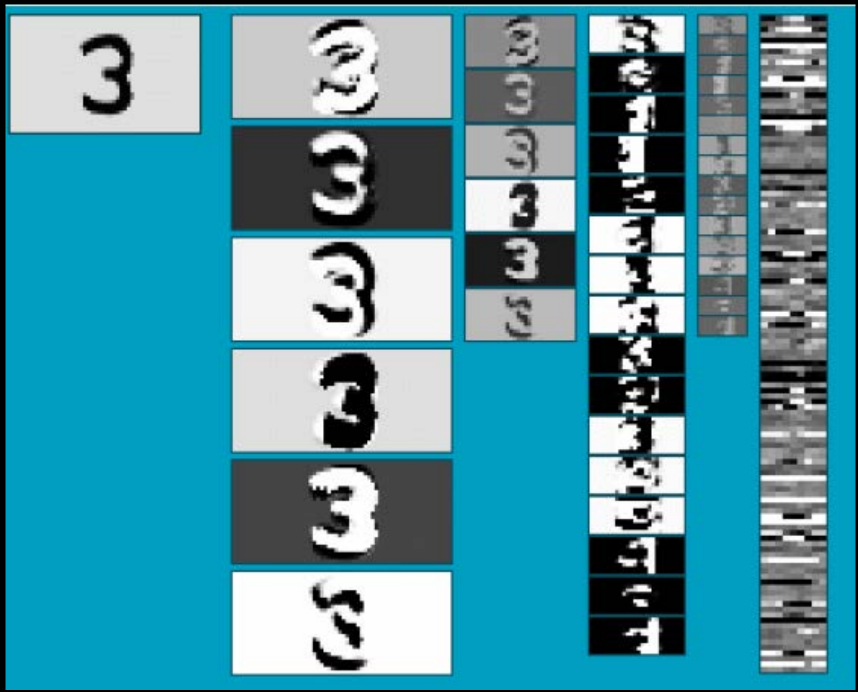
Slide: Y.LeCun

- [Hubel & Wiesel 1962]
  - simple cells detect local features
  - complex cells “pool” the outputs of simple cells within a retinotopic neighborhood.



Cognitron / Neocognitron [Fukushima 1971-1982]

- Also HMAX [Poggio 2002-2006]

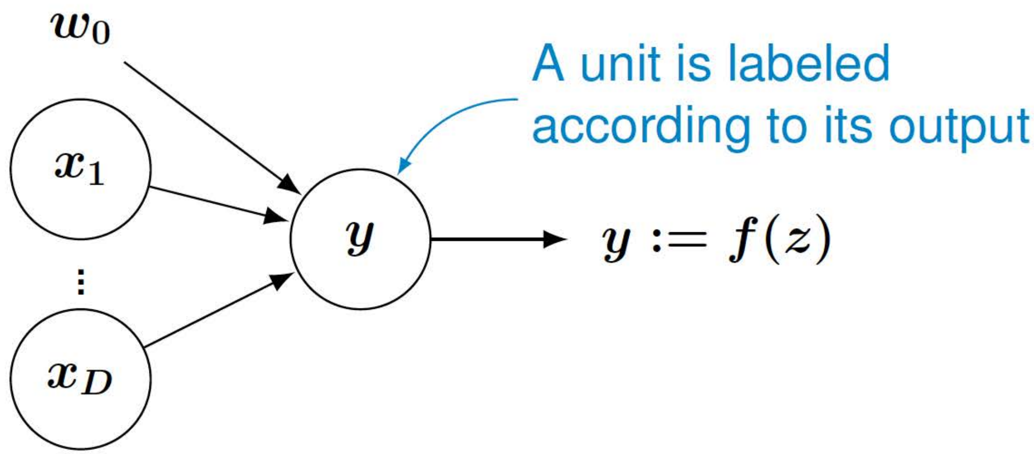


Convolutional Networks [LeCun 1988-present]

Short Intro: "Standard" Neural Networks

Core component of a neural network: *processing unit* $=$ neuron of the human brain.

A processing unit maps multiple input values onto one output value y:



- $x_1, \ldots, x_D$ are inputs, e.g. from other processing units within the network.
- w_0 is an external input called *bias*.
- The *propagation rule* maps all input values onto the actual input z.
- The *activation function* is applied to obtain $y = f(z)$.

slide taken from David Stutz (Aachen)

Short Intro: Perceptron

Introduced by Rosenblatt in [Rosenblatt 58].

The (single-layer) *perceptron* consists of D input units and C output units.

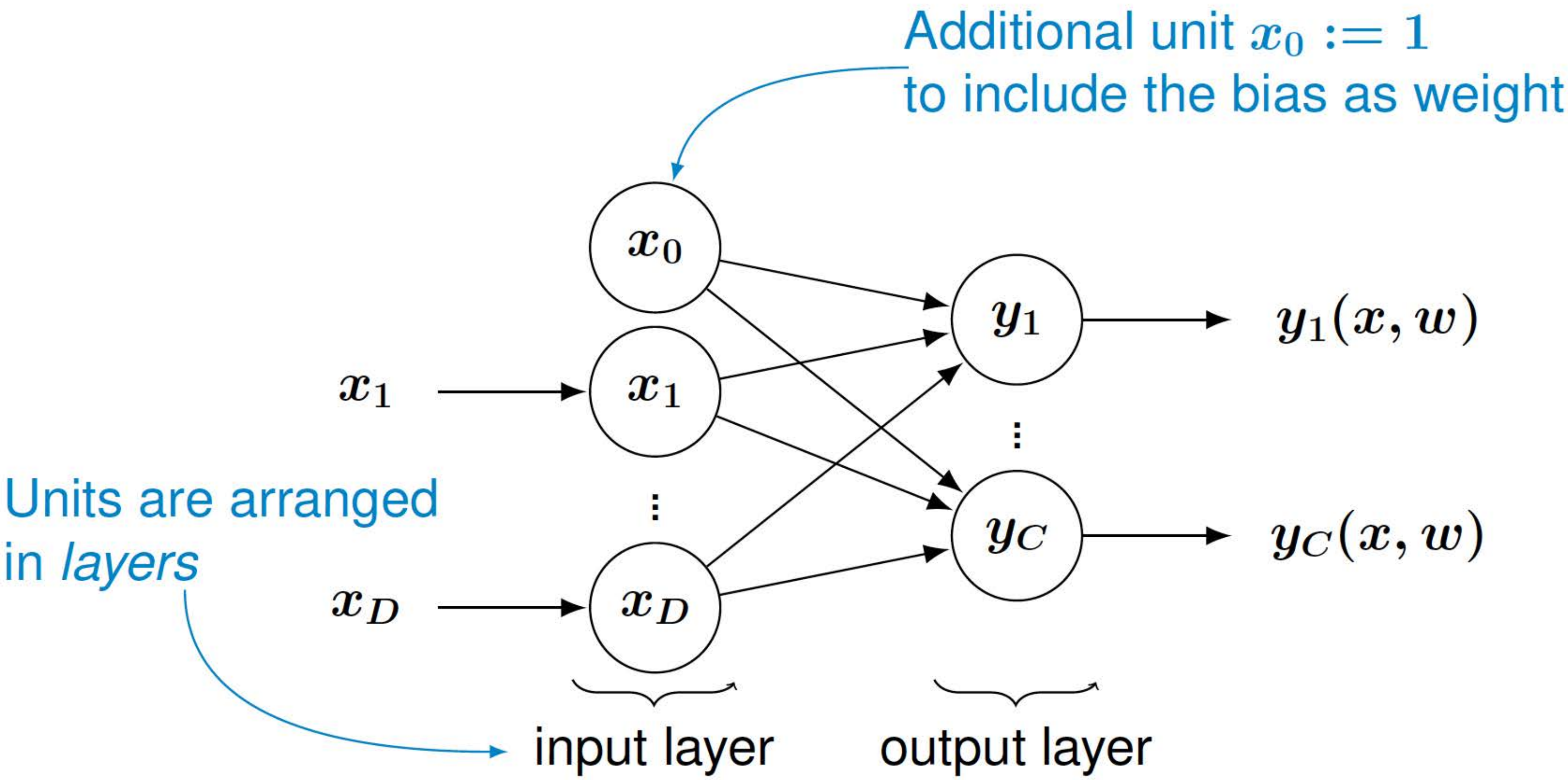
- Propagation rule: weighted sum over inputs x_i with weights w_{ij}.
- Input unit i: single input value $z = x_i$ and identity activation function.
- Output unit j calculates the output

$$y_j(x, w) = f(z_j) = f\left(\sum_{k=1}^{D} w_{jk}x_k + w_{j0}\right) \overset{x_0:=1}{=} f\left(\sum_{k=0}^{D} w_{jk}x_k\right).$$

propagation rule with additional bias w_{j0}

slide taken from David Stutz (Aachen)

Short Intro: Perceptron



slide taken from David Stutz (Aachen)

Short Intro: Perceptron - Activation Functions

Used propagation rule: weighted sum over all inputs.

How to choose the activation function $f(z)$?

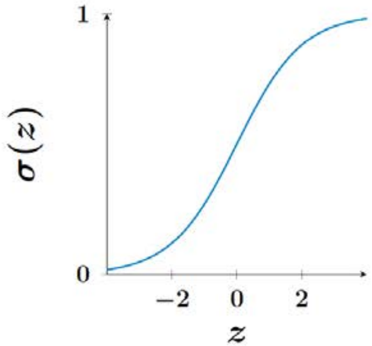
- **Heaviside function $h(z)$ models the electrical impulse of neurons in the human brain:**

$$h(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}.$$

In general we prefer monotonic, differentiable activation functions.

- **Logistic sigmoid $\sigma(z)$ as differentiable version of the Heaviside function:**

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



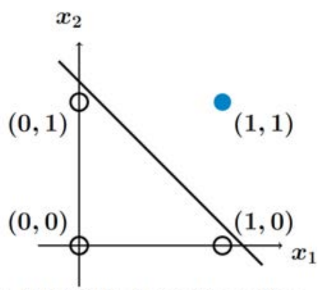
- **Or its extension for multiple output units, the softmax activation function:**

$$\sigma(z, i) = \frac{\exp(z_i)}{\sum_{k=1}^{C} \exp(z_k)}.$$

Single Layer Perceptron

Which target functions can be modeled using a single-layer perceptron?

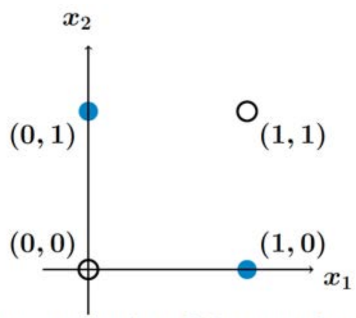
- **A single-layer perceptron represents a hyperplane in multidimensional space.**



Modeling boolean AND with target function $g(x_1, x_2) \in \{0, 1\}$.

Problem: How to model boolean exclusive OR (XOR) using a line in two-dimensional space?

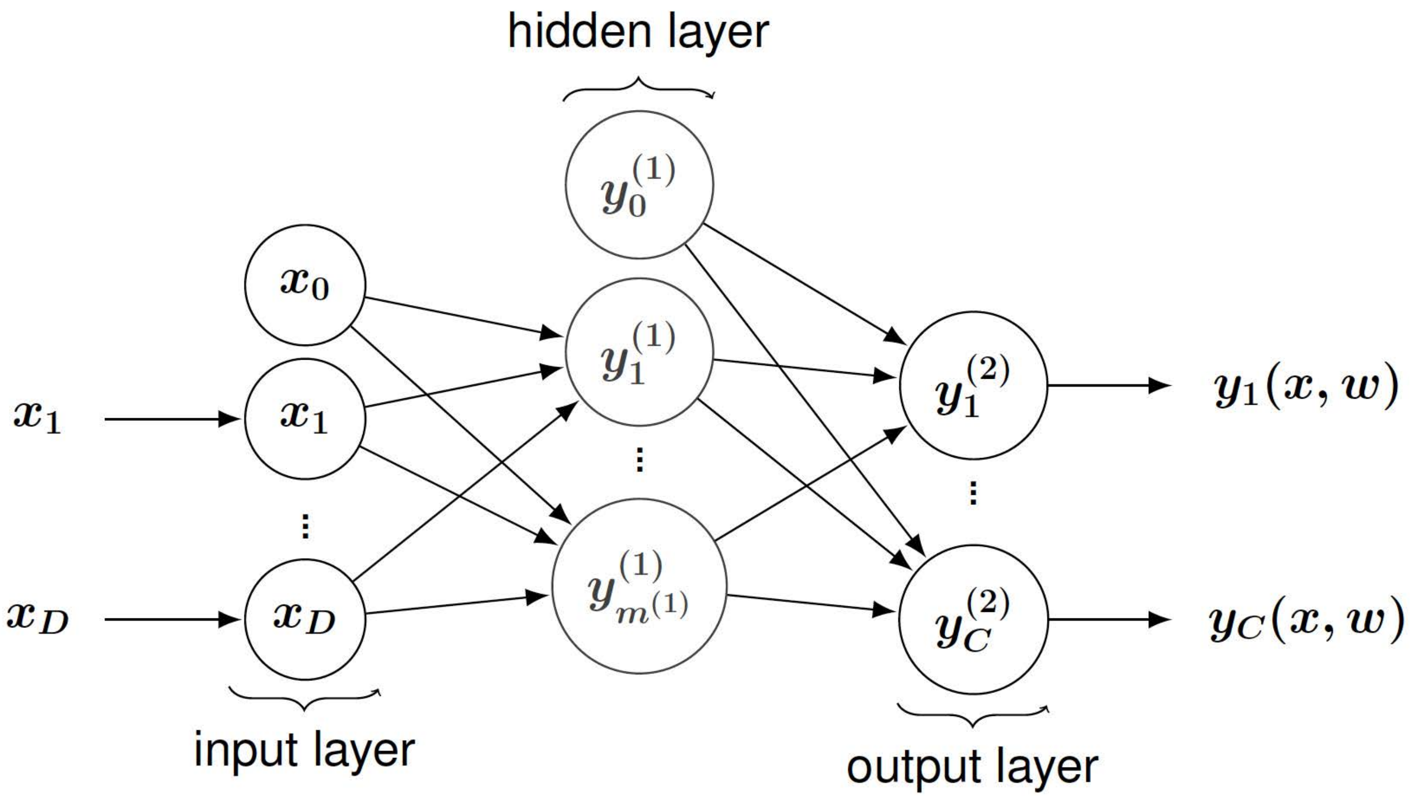
- **Boolean XOR cannot be modeled using a single-layer perceptron.**



Boolean exclusive OR target function.

slide taken from David Stutz (Aachen)

Short Intro: Two-Layer Perceptron



slide taken from David Stutz (Aachen)

Short Intro: Multi-Layer Perceptron (MLP)

Idea: Add additional $L > 0$ *hidden* layers in between the input and output layer.

- **$m^{(l)}$ hidden units in layer (l) with $m^{(0)} := D$ and $m^{(L+1)} := C$.**
- **Hidden unit i in layer l calculates the output**

layer

$$y_i^{(l)} = f\left(\sum_{k=0}^{m^{(l-1)}} w_{ik} y_k^{(l-1)}\right).$$

unit

A *multilayer perceptron* models a function

$$y(\cdot, w) : \mathbb{R}^D \mapsto \mathbb{R}^C, x \mapsto y(x, w) = \begin{pmatrix} y_1(x, w) \\ \vdots \\ y_C(x, w) \end{pmatrix} = \begin{pmatrix} y_1^{(L+1)} \\ \vdots \\ y_C^{(L+1)} \end{pmatrix}$$

where $y_i^{(L+1)}$ is the output of the i-th output unit.

slide taken from David Stutz (Aachen)

Network Training

Training a neural network means adjusting the weights to get a good approximation of the target function.

How **does a neural network learn?**

- ***Supervised learning*: *Training set* T provides both input values and the corresponding target values:**

input value – *pattern*

$$T := \{(x_n, t_n) : 1 \leq n \leq N\}. \tag{6}$$

target value

- **Approximation performance of the neural network can be evaluated using a distance measure between approximation and target function.**

slide taken from David Stutz (Aachen)

Network Training - Error Measures

Sum-of-squared error function:

weight vector

k-th component of modeled function y

$$E(w) = \sum_{n=1}^{N} E_n(w) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{C} (y_k(x_n, w) - t_{nk})^2.$$

k-th entry of t_n

Cross-entropy error function:

$$E(w) = \sum_{n=1}^{N} E_n(w) = - \sum_{n=1}^{N} \sum_{k=1}^{C} t_{nk} \log y_k(x_n, w).$$

slide taken from David Stutz (Aachen)

Network Training - Approaches

Idea: Adjust the weights such that the error is minimized.

Stochastic training Randomly choose an input value x_n and update the weights based on the error $E_n(w)$.

Mini-batch training Process a subset $M \subseteq \{1, \ldots, N\}$ of all input values and update the weights based on the error $\sum_{n \in M} E_n(w)$.

Batch training Process all input values x_n, $1 \leq n \leq N$ and update the weights based on the overall error $E(w) = \sum_{n=1}^{N} E_n(w)$.

slide taken from David Stutz (Aachen)

Network Training - Parameter Optimization

How to minimize the error $E(w)$?

Problem: $E(w)$ can be nonlinear and may have multiple local minima.

Iterative optimization algorithms:

- Let $w[0]$ be a starting vector for the weights.
- $w[t]$ is the weight vector in the t-th iteration of the optimization algorithm.
- In iteration $[t+1]$ choose a *weight update* $\Delta w[t]$ and set

$$w[t+1] = w[t] + \Delta w[t].$$

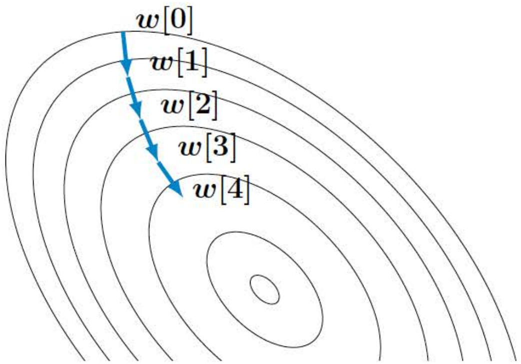
- Different optimization algorithms choose different weight updates.

slide taken from David Stutz (Aachen)

Parameter Optimization by Gradient Descent

Idea: In each iteration take a step in the direction of the negative gradient.

- **The direction of the steepest descent.**



- **Weight update $\Delta w[t]$ is given by**

$$\Delta w[t] = -\gamma \frac{\partial E}{\partial w[t]}.$$

learning rate – *step size*

slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

Summary: We want to minimize the error $E(w)$ on the training set T to get a good approximation of the target function.

Using gradient descent and stochastic learning, the weight update in iteration $[t+1]$ is given by

$$w[t+1]_{ij}^{(l)} = w[t]_{ij}^{(l)} - \gamma \frac{\partial E_n}{\partial w[t]_{ij}^{(l)}}. \quad (11)$$

How to evaluate the gradient $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ of the error function with respect to the current weight vector?

Using the chain rule we can write:

$$\frac{\partial E_n}{\partial w_{ij}^{(l)}} = \frac{\partial E_n}{\partial z_i^{(l)}} \underbrace{\frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}}_{=y_j^{(l-1)}}. \quad (12)$$

slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

Error backpropagation allows to evaluate $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ for each weight in $\mathcal{O}(W)$ where W is the total number of weights:

(1) Calculate the *errors* $\delta_i^{(L+1)}$ for the output layer:

$$\delta_i^{(L+1)} := \frac{\partial E_n}{\partial z_i^{(L+1)}} = \frac{\partial E_n}{\partial y_i^{(L+1)}} f'\left(z_i^{(L+1)}\right). \tag{13}$$

- The output errors are often easy to calculate.
  - For example using the sum-of-squared error function and the identity as output activation function:

$$\delta_i^{(L+1)} = \frac{\partial\left[\frac{1}{2}\sum_{k=1}^{C}(y_k^{(L+1)} - t_{nk})^2\right]}{\partial y_i^{(L+1)}} \cdot 1 = y_i(x_n, w) - t_{ni}. \tag{14}$$

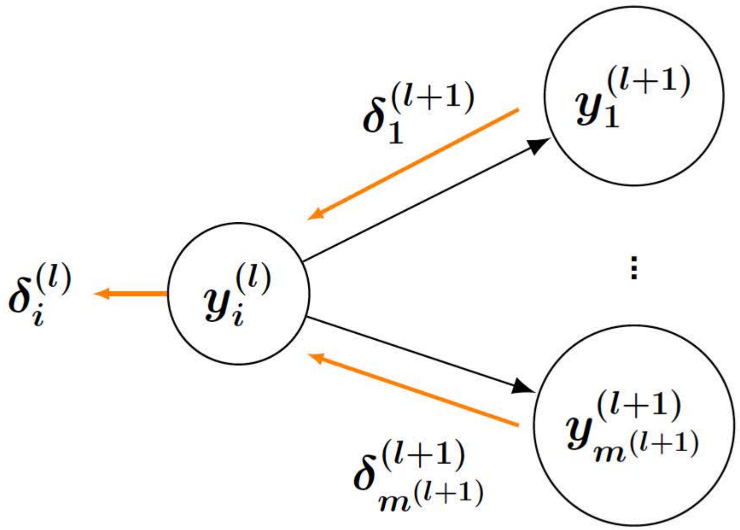
slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

(2) Backpropagate the errors $\delta_i^{(L+1)}$ through the network using

$$\delta_i^{(l)} := \frac{\partial E_n}{\partial z_i^{(l)}} = f'\left(z_i^{(l)}\right) \sum_{k=1}^{m^{(l+1)}} w_{ik}^{(l+1)} \delta_k^{(l+1)}. \quad (15)$$

- **This can be evaluated recursively for each layer after determining the errors $\delta_i^{(L+1)}$ for the output layer.**



slide taken from David Stutz (Aachen)

Backpropagation = Parameter Optimization by Gradient Descent

(3) Determine the needed derivatives using

$$\frac{\partial E_n}{\partial w_{ij}^{(l)}} = \frac{\partial E_n}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} y_j^{(l-1)}.$$

Now use the derivatives $\frac{\partial E_n}{\partial w_{ij}^{(l)}}$ to update the weights in each iteration.

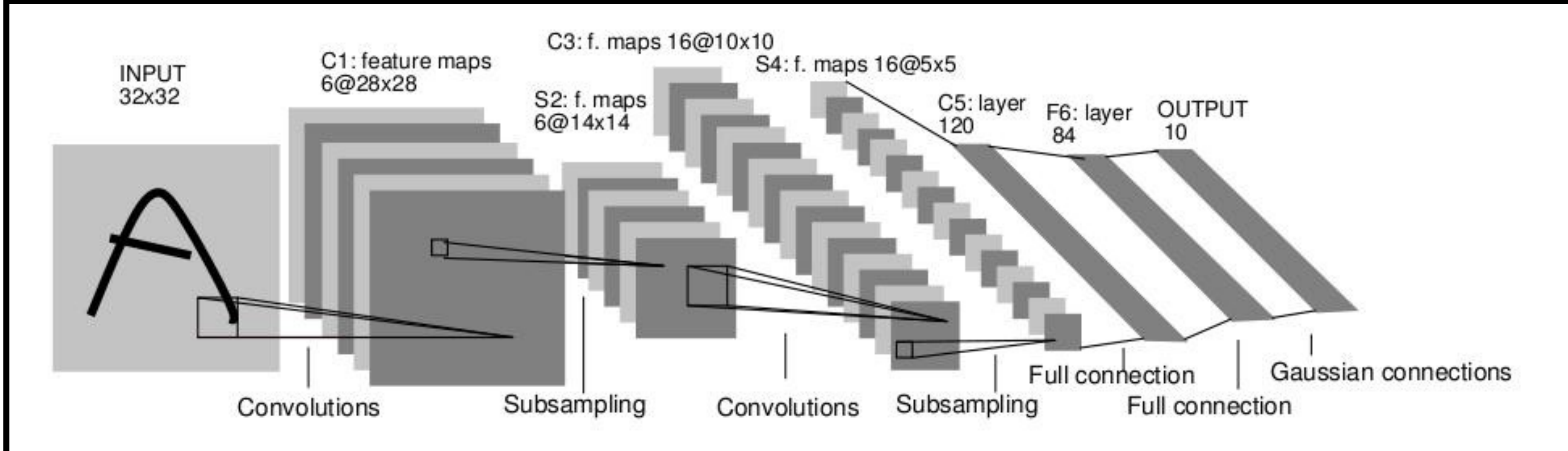
- In iteration step $[t+1]$ set

$$w[t+1]_{ij}^{(l)} = w[t]_{ij}^{(l)} - \gamma \frac{\partial E_n}{\partial w[t]_{ij}^{(l)}}.$$

slide taken from David Stutz (Aachen)

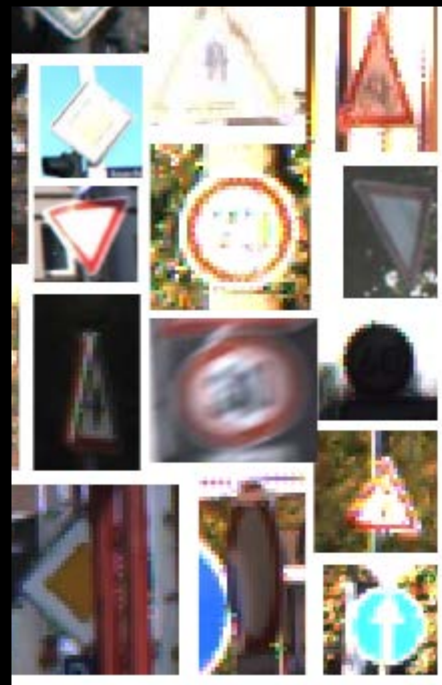
Convolutional Neural Networks

- LeCun et al. 1989
- Neural network with specialized connectivity structure



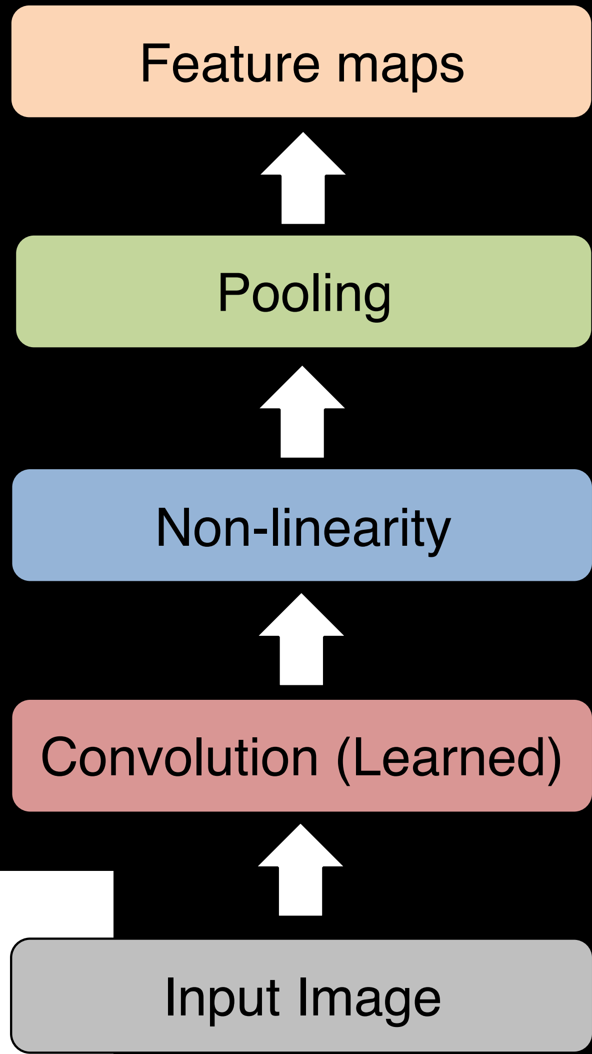
Convnet Successes

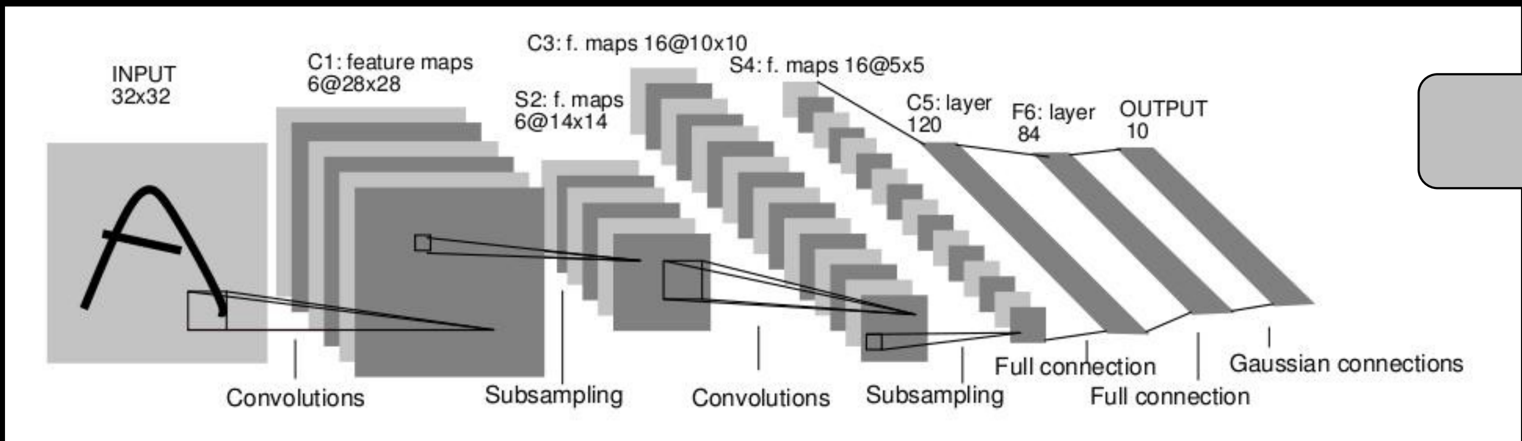
- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese   [Ciresan et al. 2012]
- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition
    - 0.56% error vs 1.16% for humans [Ciresan et al. 2011]
- But (until recently) less good at more complex datasets
  - E.g. Caltech-101/256 (few training examples)



Characteristics of Convnets

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





[LeCun et al. 1989]

Application to ImageNet

IMAGENET [Deng et al. CVPR 2009]

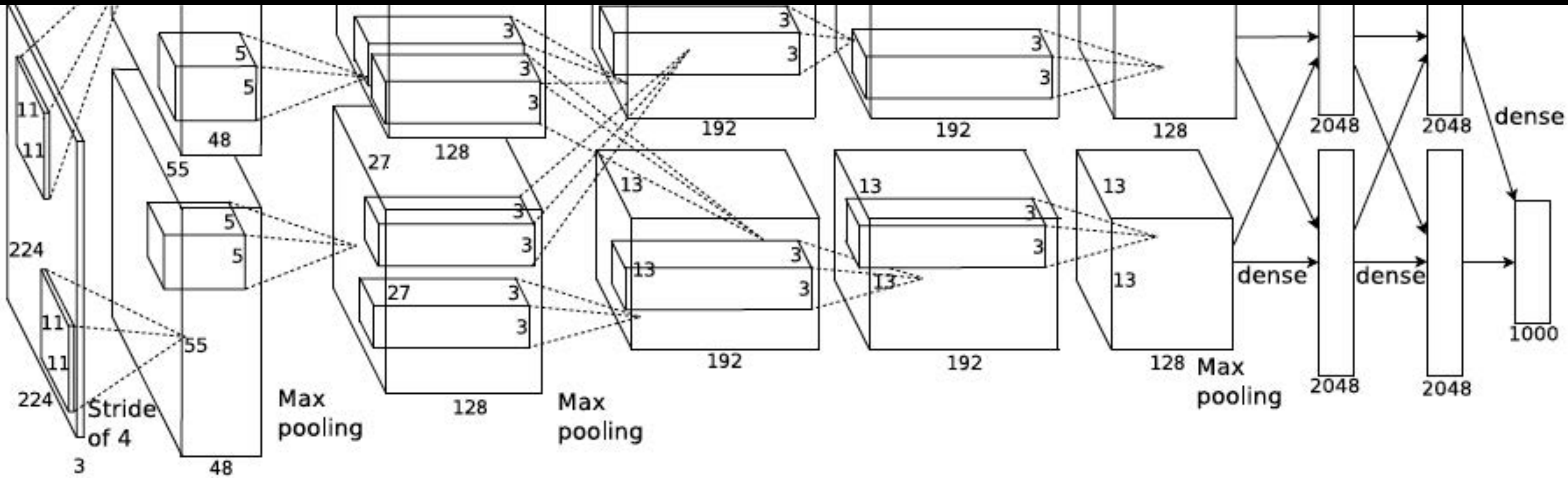
- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon Turk

ImageNet Classification with Deep Convolutional Neural Networks [NIPS 2012]

Alex Krizhevsky	Ilya Sutskever	Geoffrey E. Hinton
University of Toronto	University of Toronto	University of Toronto
kriz@cs.utoronto.ca	ilya@cs.utoronto.ca	hinton@cs.utoronto.ca

Krizhevsky et al. [NIPS2012]

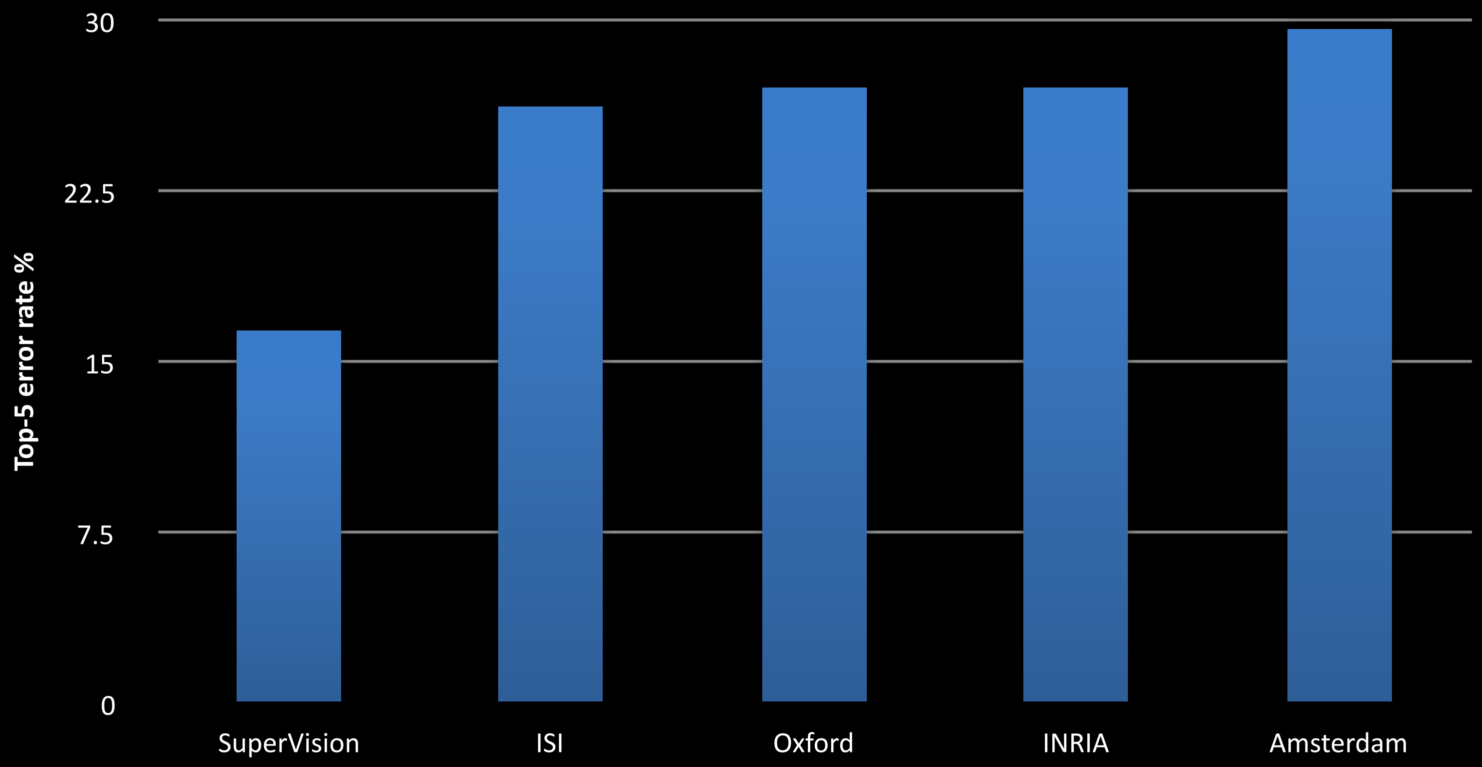
- Same model as LeCun'98 but:
  - Bigger model (8 layers)
  - More data (10^6 vs 10^3 images)
  - GPU implementation (50x speedup over CPU)
  - Better regularization (DropOut)



- 7 hidden layers, 650,000 neurons, 60,000,000 parameters
- Trained on 2 GPUs for a week

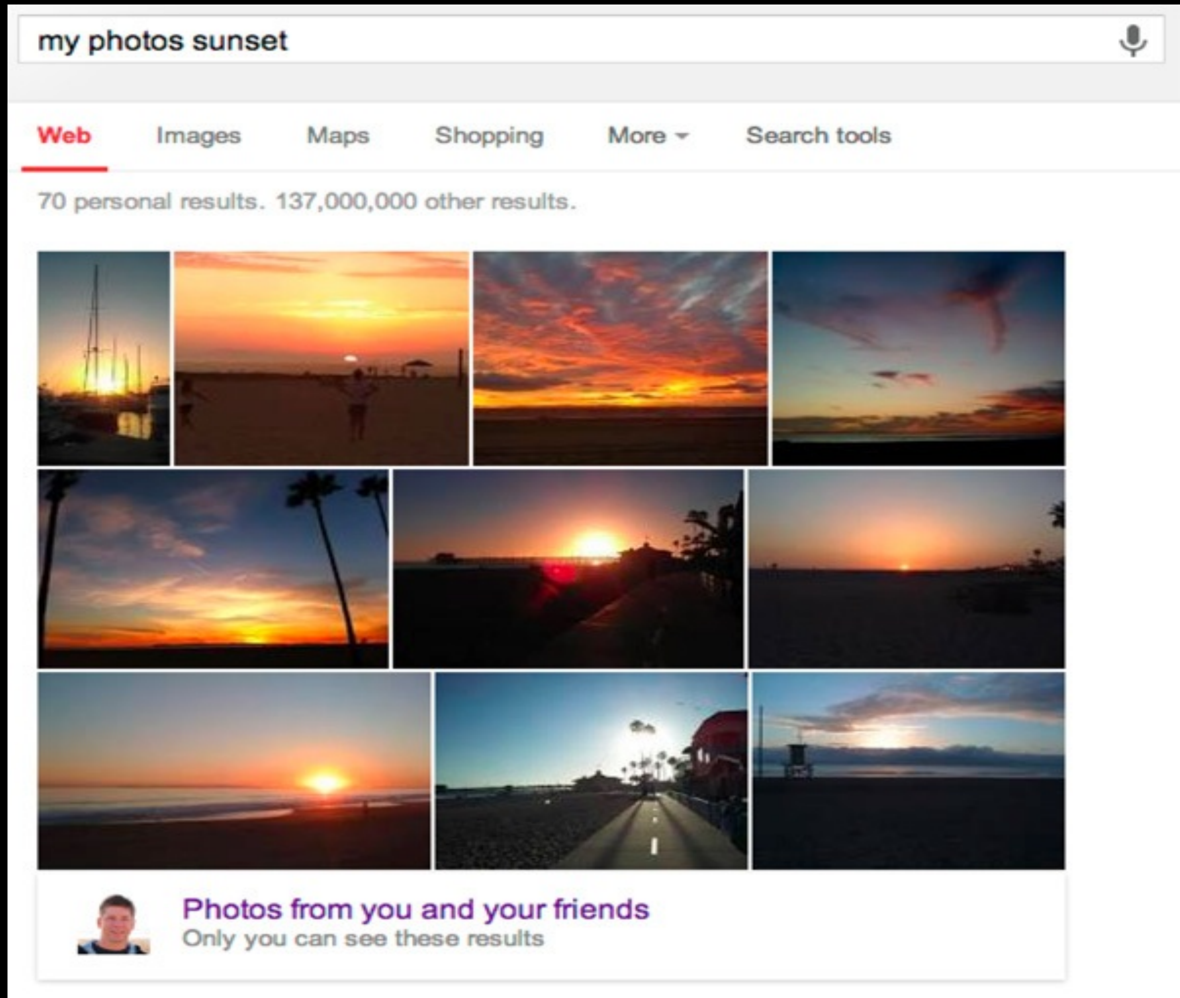
ImageNet Classification 2012

- Krizhevsky et al. - 16.4% error (top-5)
- Next best (non-convnet) – 26.2% error



Commercial Deployment

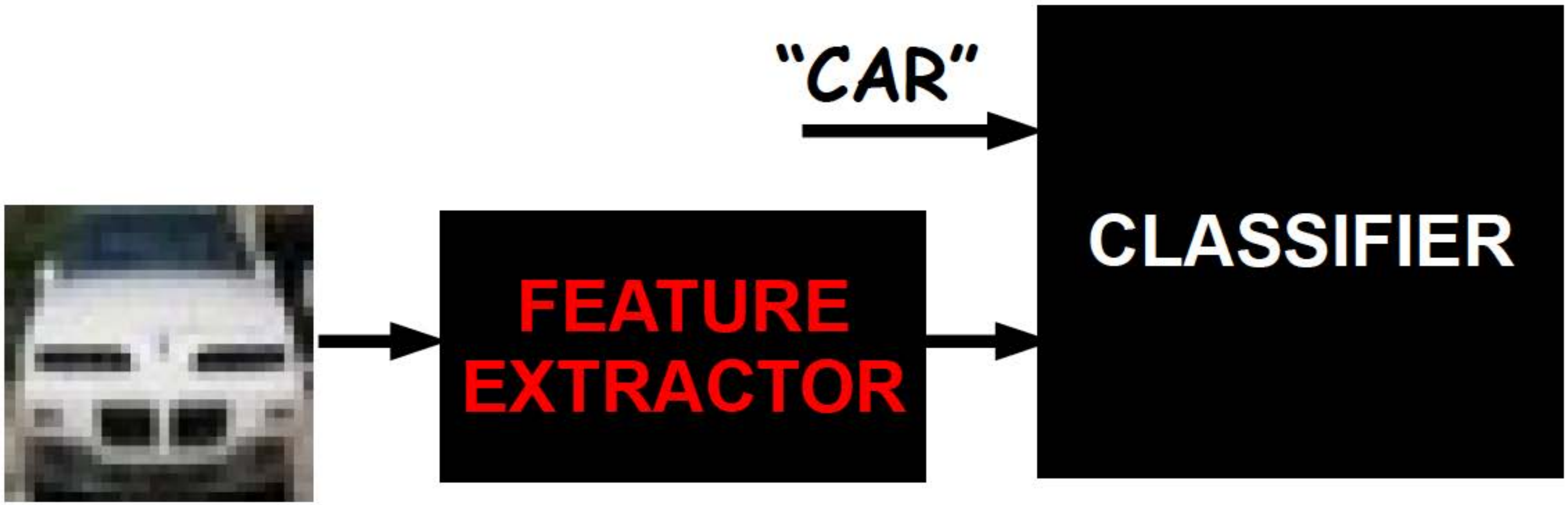
- Google & Baidu, Spring 2013 for personal image search



Intuitions Behind Deep Networks

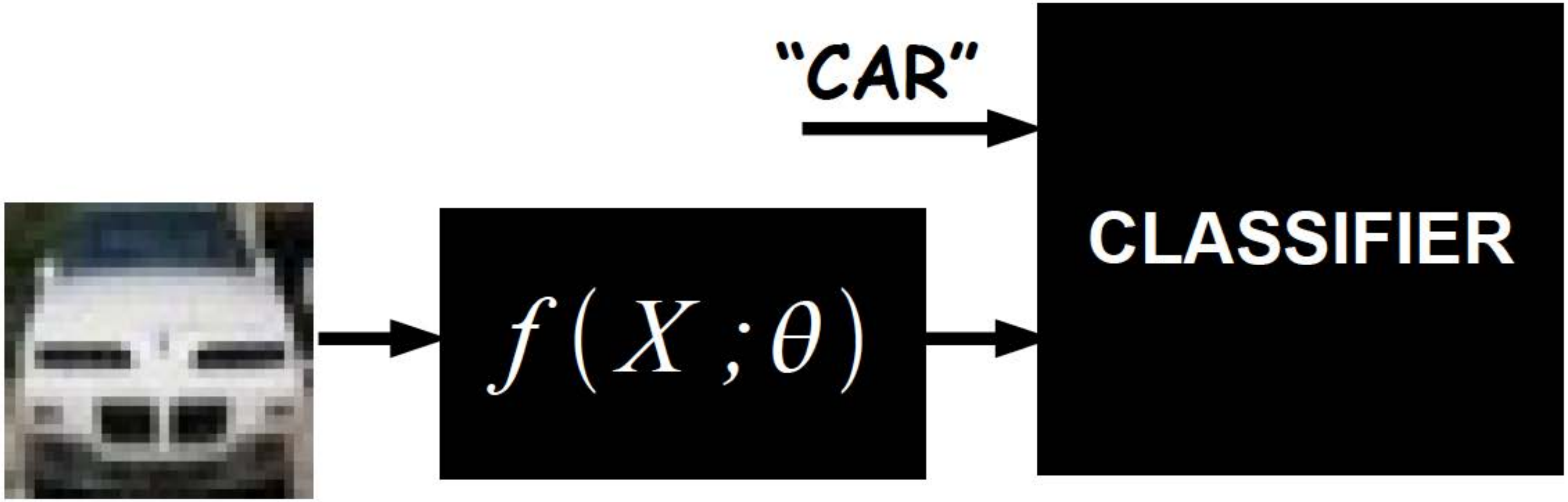
(following slides from Marc Aurelio Ranzato - Google)

Building an Object Recognition System



IDEA: Use data to optimize features for the given task.

Building an Object Recognition System



What we want: Use parameterized function such that
a) features are computed efficiently
b) features can be trained efficiently

Building an Object Recognition System

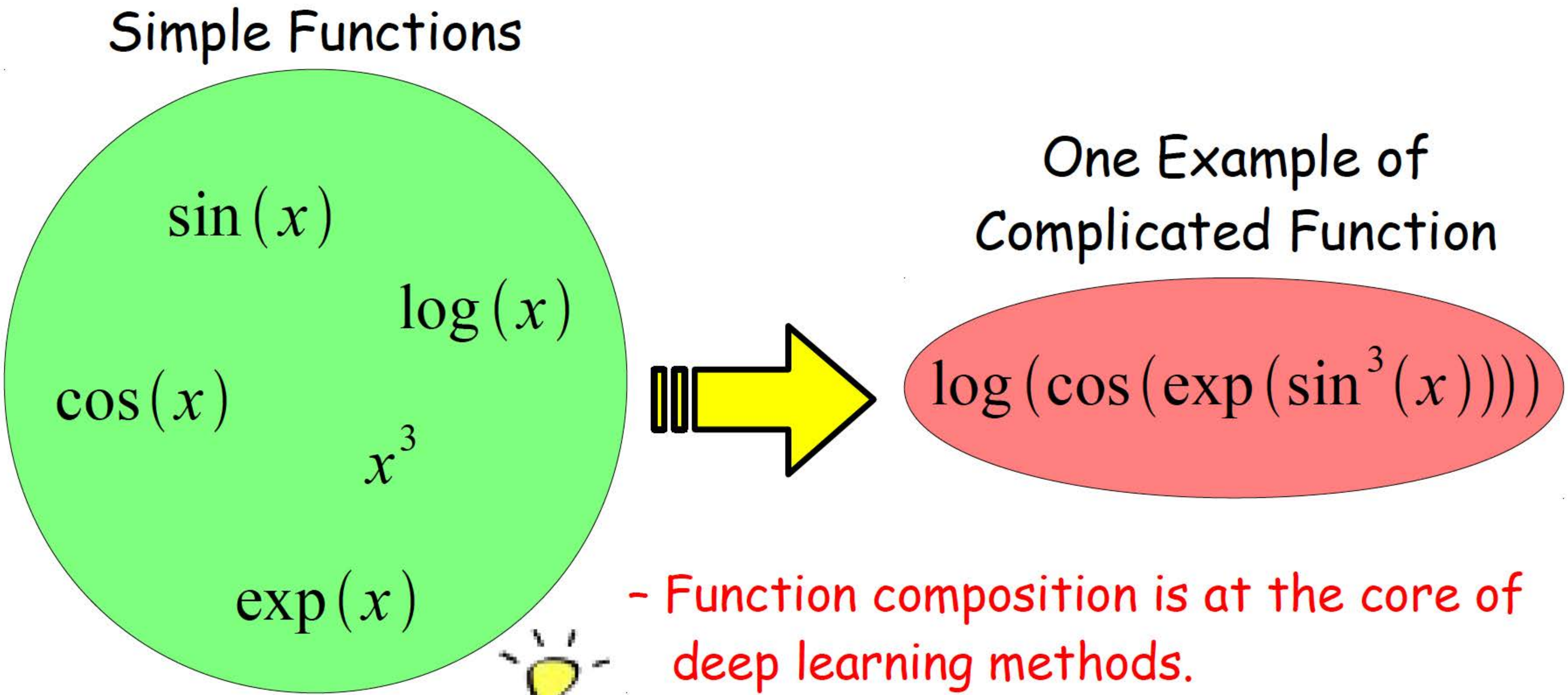
- Everything becomes adaptive.
- No distiction between feature extractor and classifier.
- Big non-linear system trained from raw pixels to labels.

Building an Object Recognition System

Q: How can we build such a highly non-linear system?

A: By combining simple building blocks we can make more and more complex systems.

Building A Complicated Function

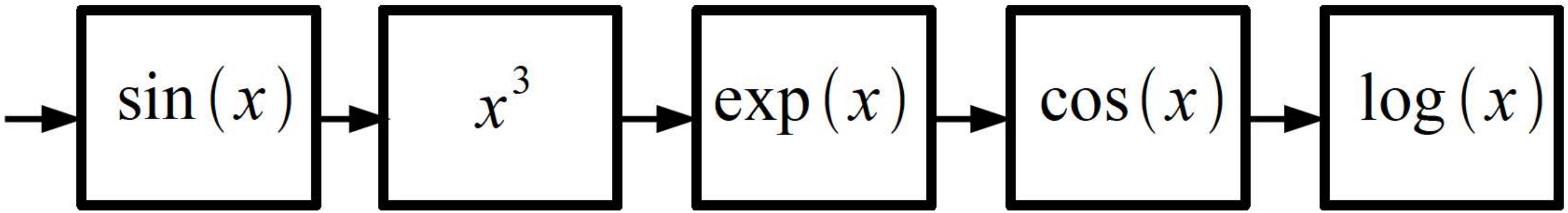


- Function composition is at the core of deep learning methods.
- Each "simple function" will have parameters subject to training.

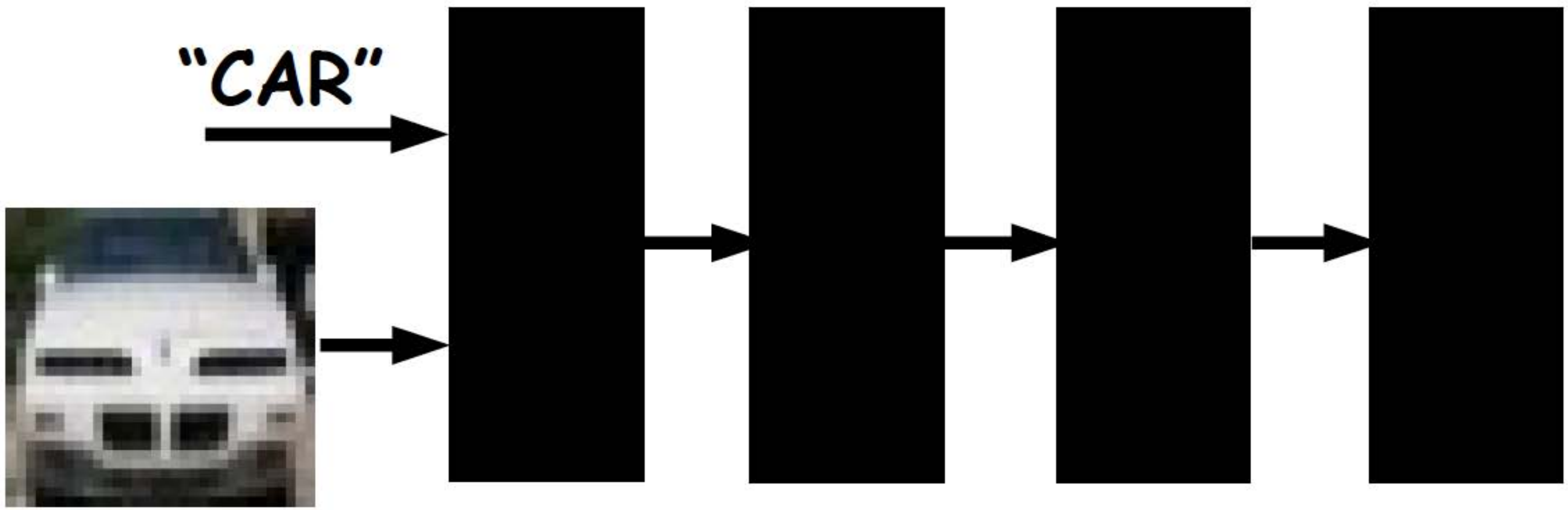
Implementing A Complicated Function

Complicated Function

$$\log(\cos(\exp(\sin^3(x))))$$

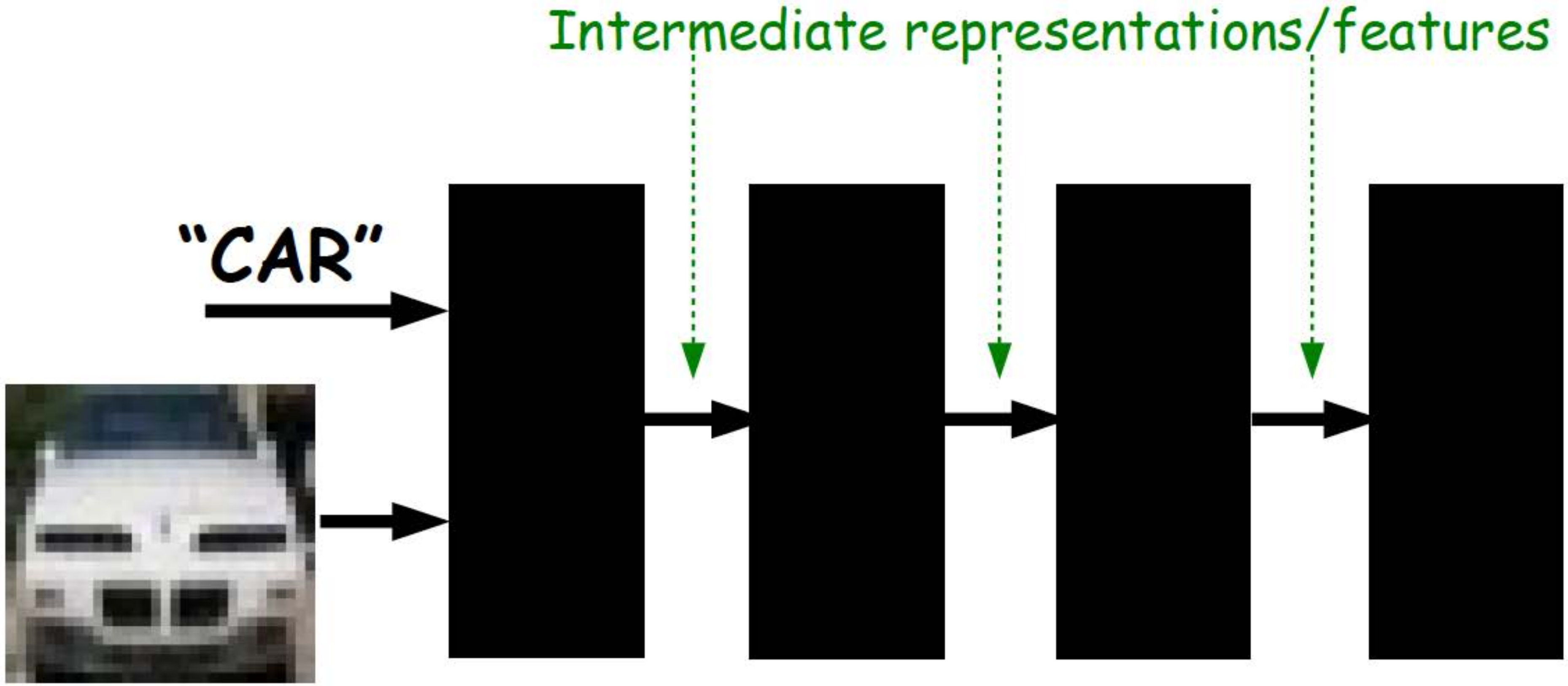


Intuition Behind Deep Neural Nets



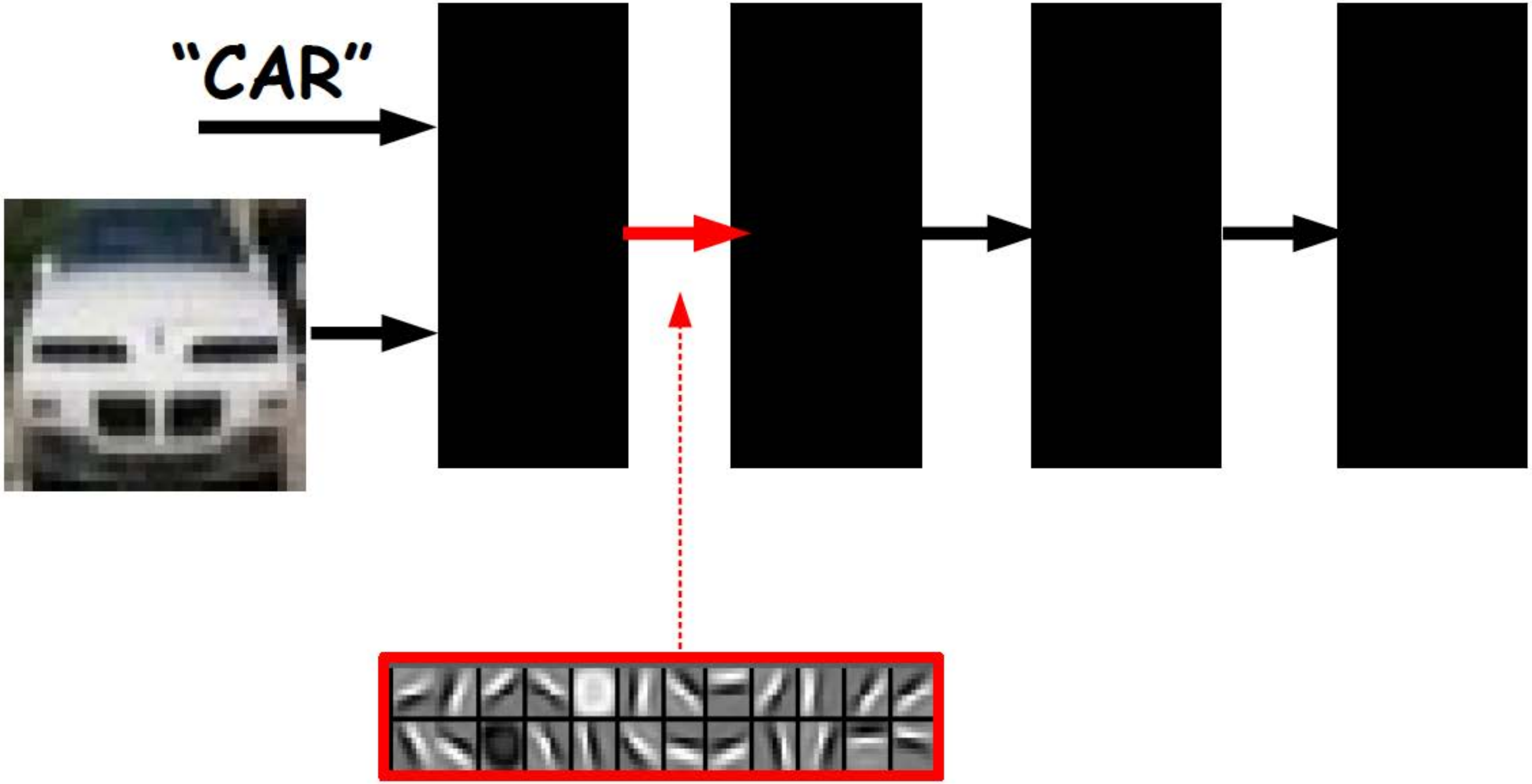
NOTE: Each black box can have trainable parameters.
Their composition makes a highly non-linear system.

Intuition Behind Deep Neural Nets



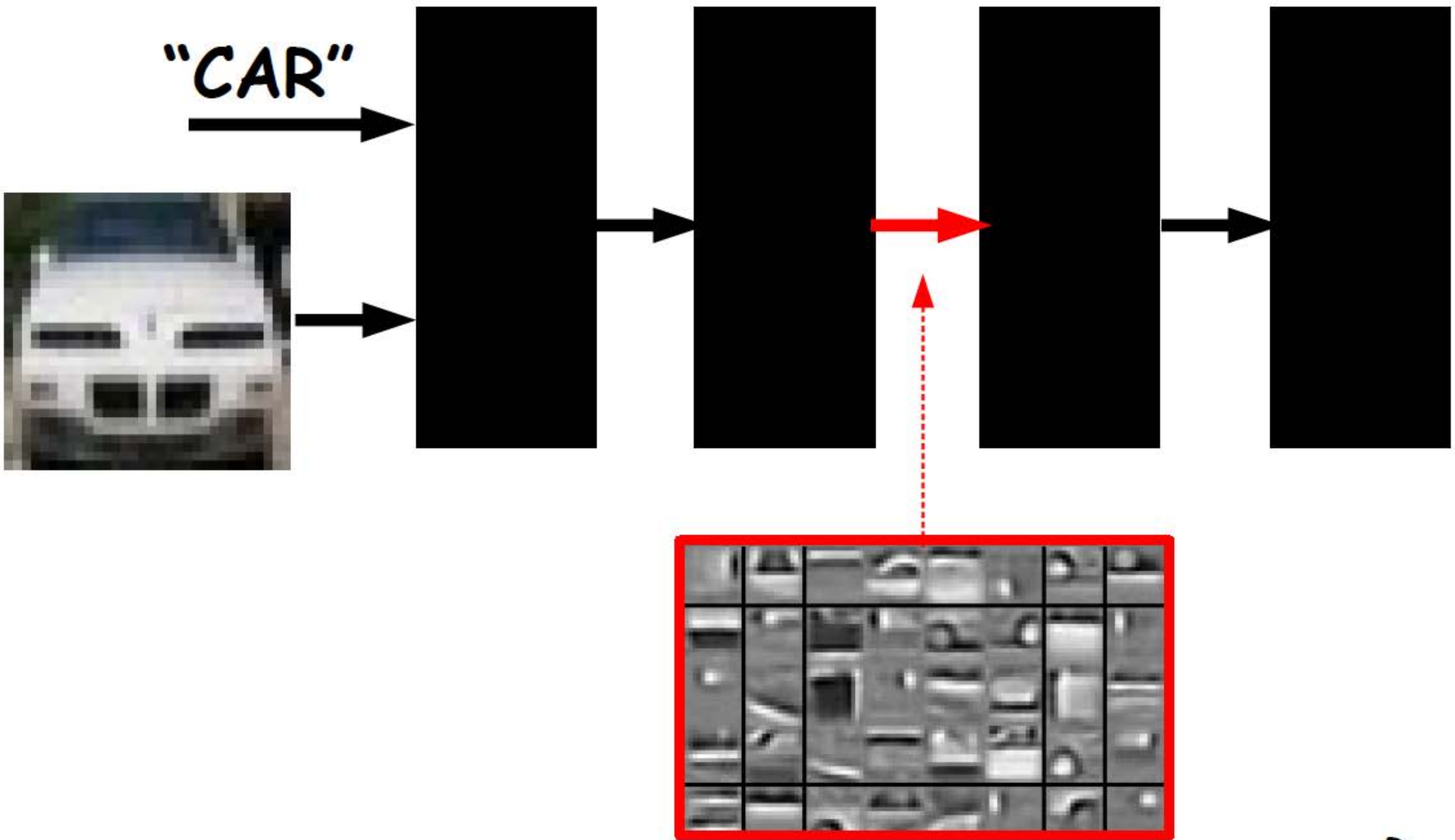
NOTE: System produces a hierarchy of features.

Intuition Behind Deep Neural Nets



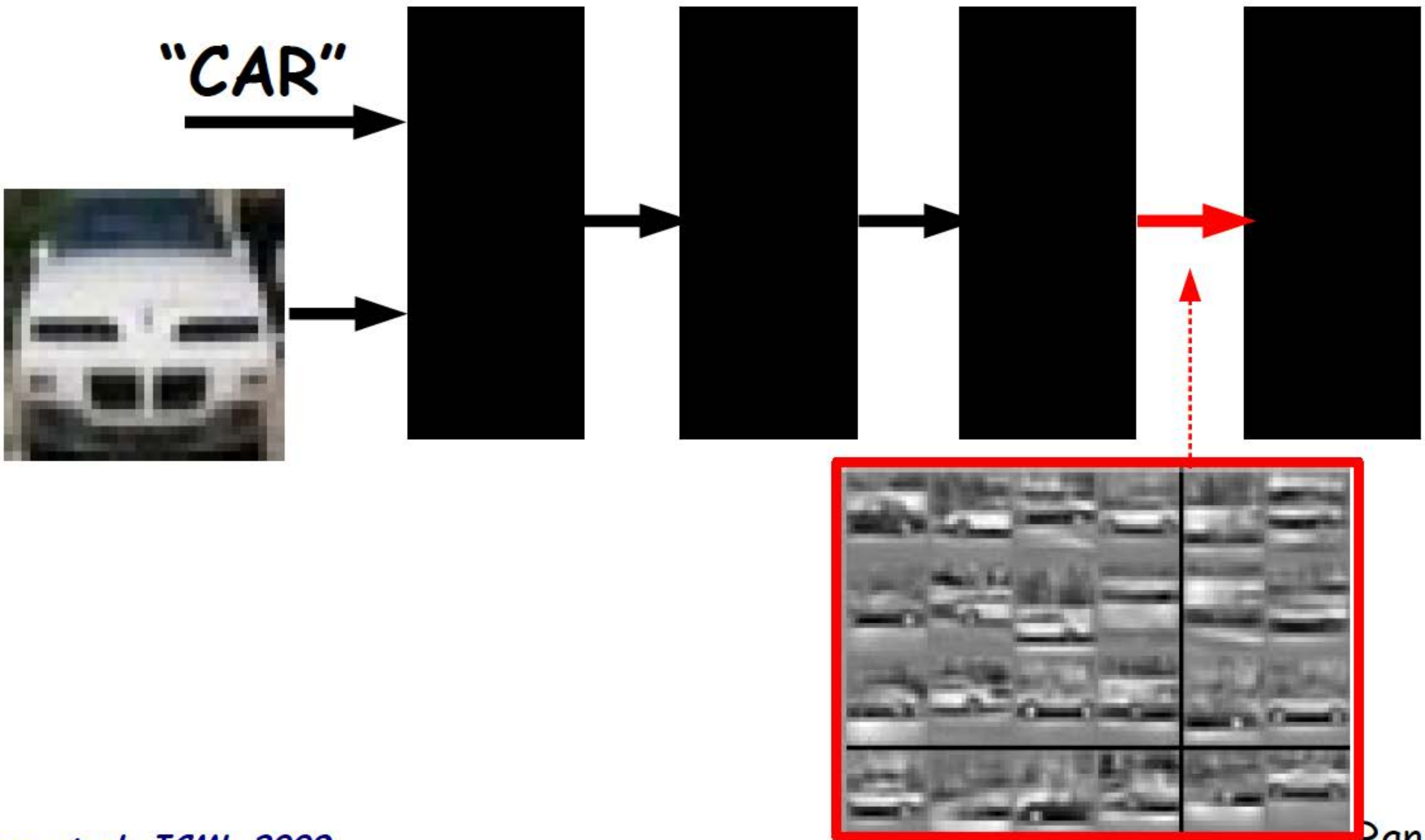
Lee et al. "Convolutional DBN's for scalable unsup. learning..." ICML 2009

Intuition Behind Deep Neural Nets



Lee et al. ICML 2009

Intuition Behind Deep Neural Nets



Lee et al. ICML 2009

KEY IDEAS OF NEURAL NETS

IDEA # 1

Learn features from data

IDEA # 2

Use differentiable functions that produce features efficiently

IDEA # 3

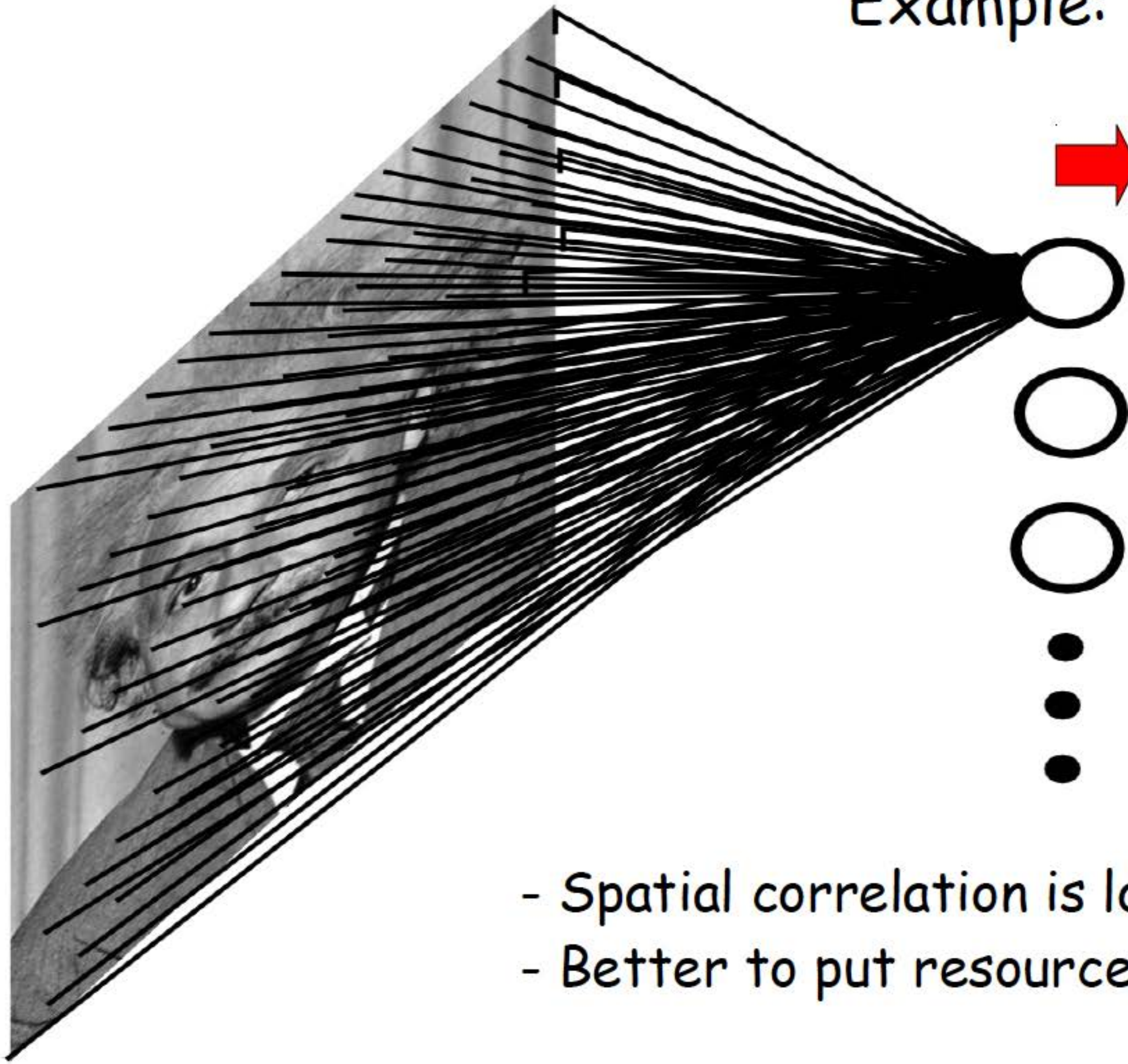
End-to-end learning:
no distinction between feature extractor and classifier

IDEA # 4

"Deep" architectures:
cascade of simpler non-linear modules

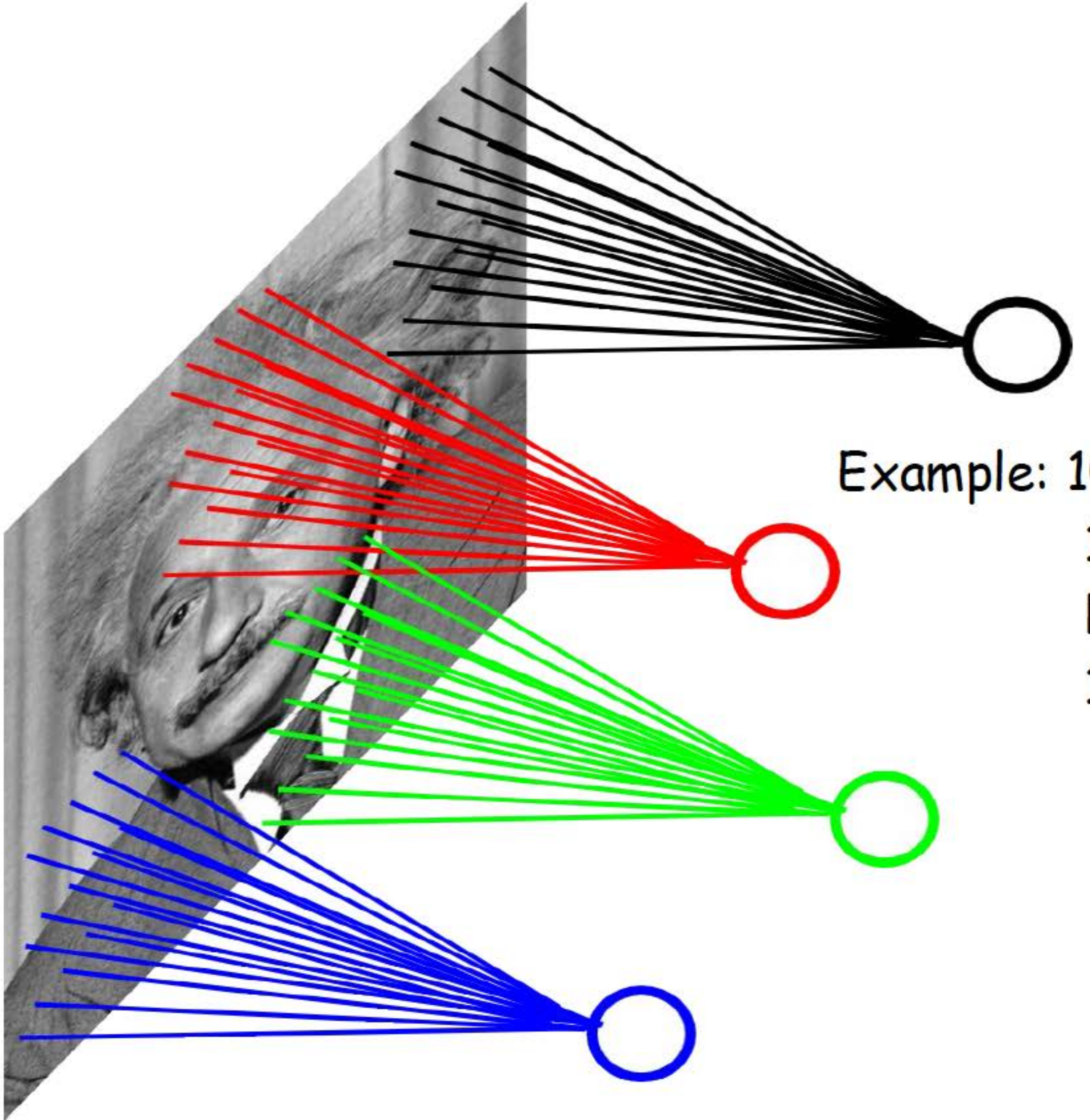
FULLY CONNECTED NEURAL NET

Example: 1000x1000 image
1M hidden units
→ 10^12 parameters!!!



- Spatial correlation is local
- Better to put resources elsewhere!

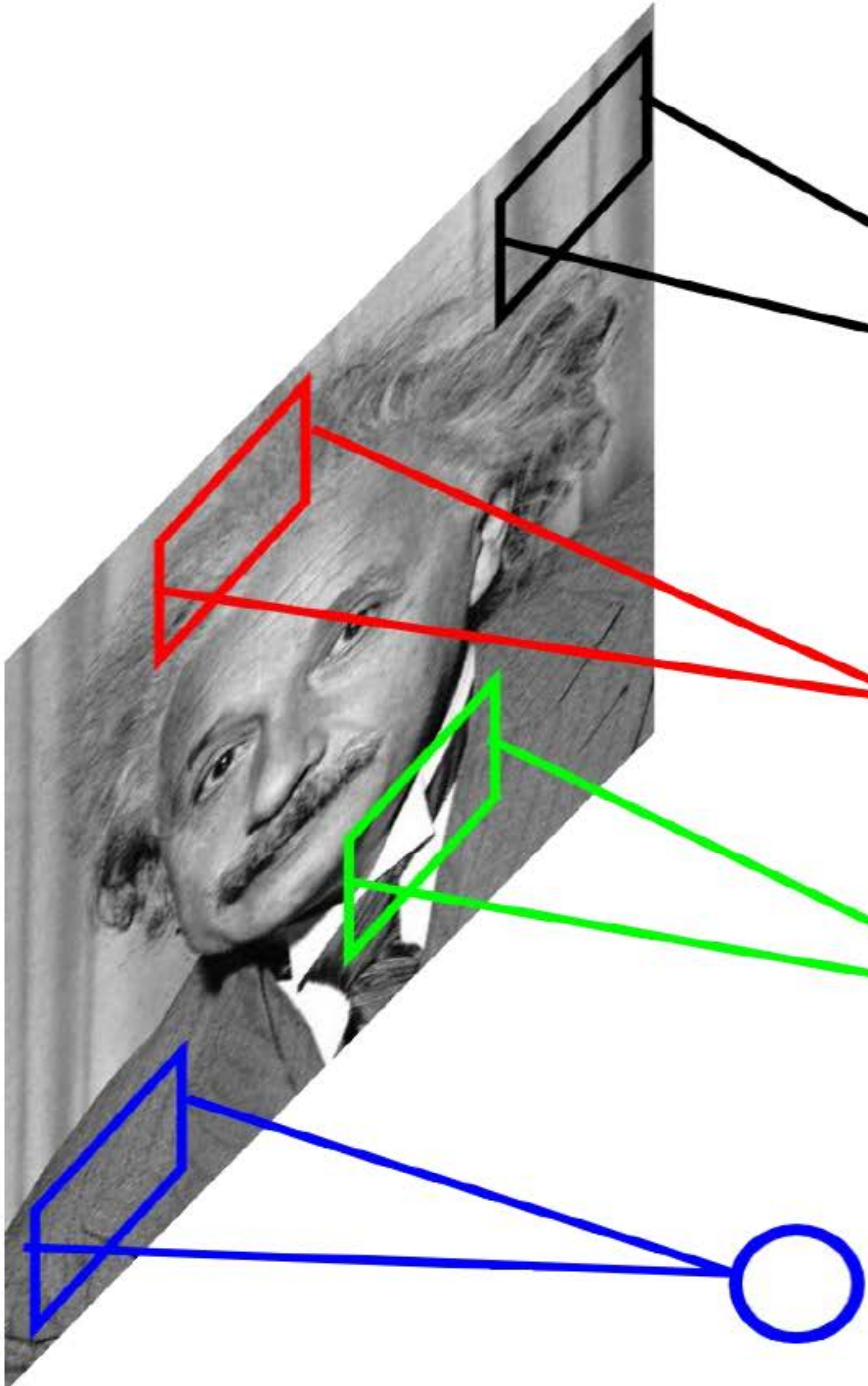
LOCALLY CONNECTED NEURAL NET



Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Ranzato

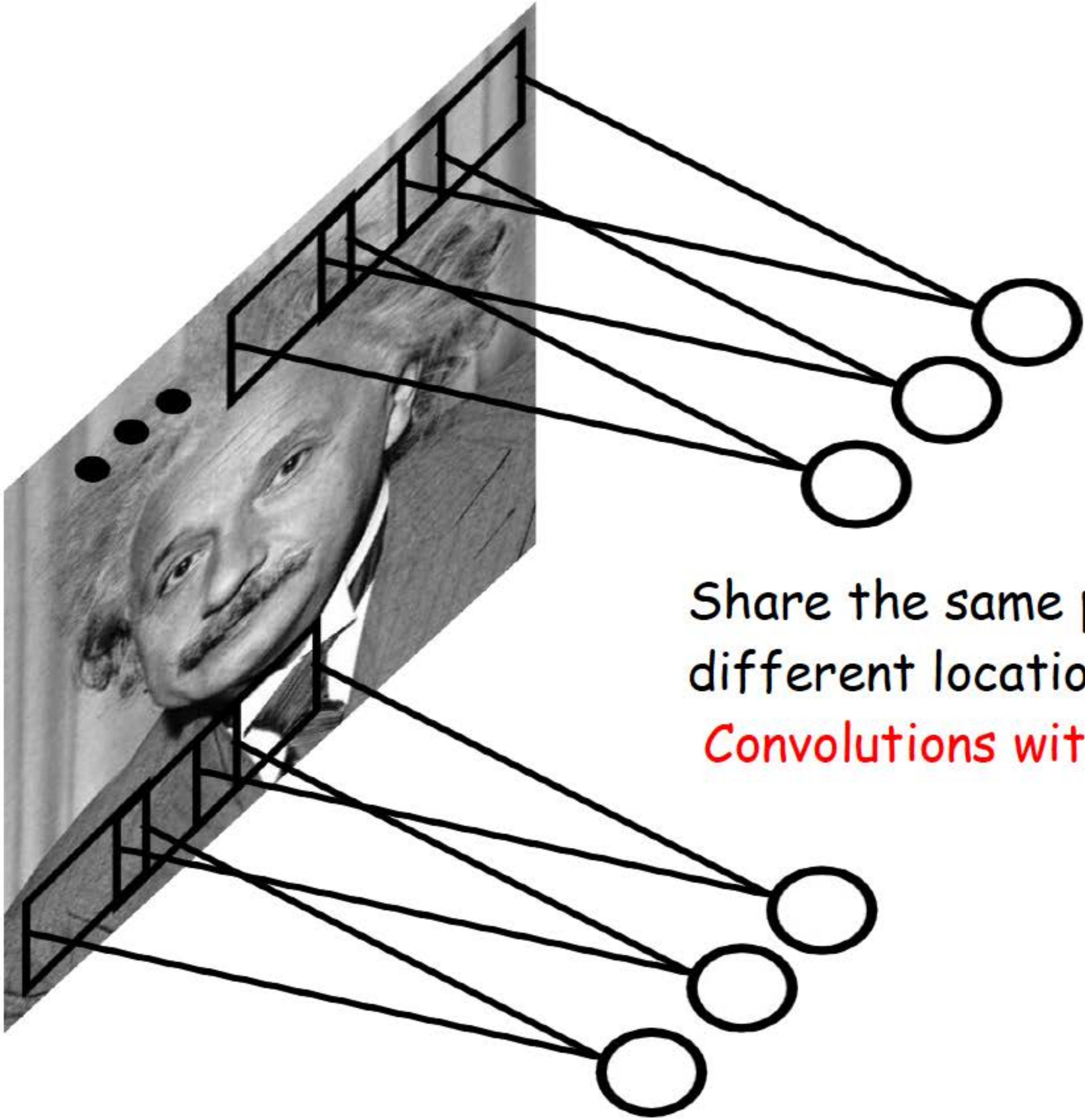
LOCALLY CONNECTED NEURAL NET



STATIONARITY? Statistics is similar at different locations

Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

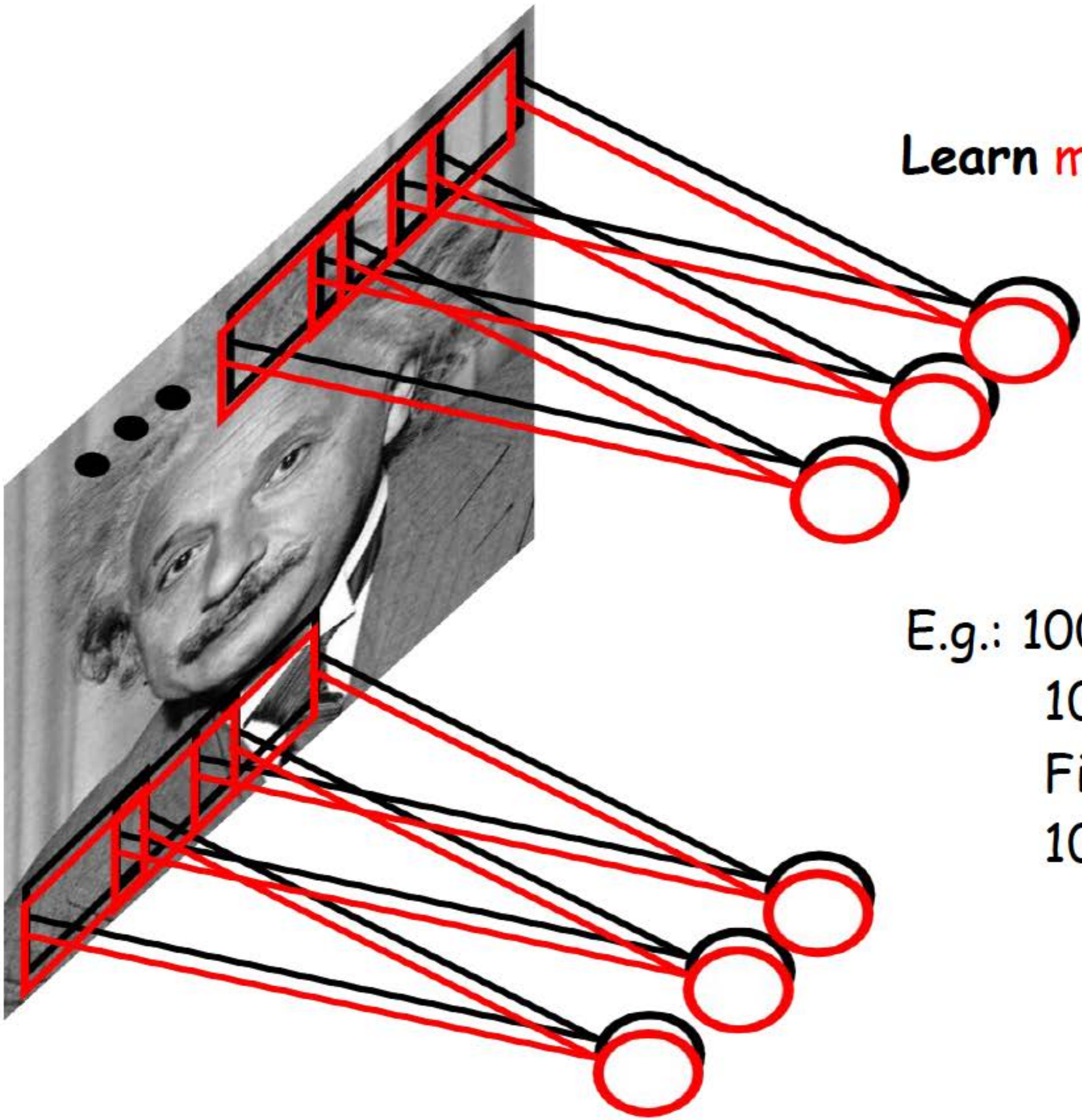
CONVOLUTIONAL NET



Share the same parameters across different locations:

Convolutions with learned kernels

CONVOLUTIONAL NET



Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

NEURAL NETS FOR VISION

A standard neural net applied to images:

- scales quadratically with the size of the input
- does not leverage stationarity

Solution:

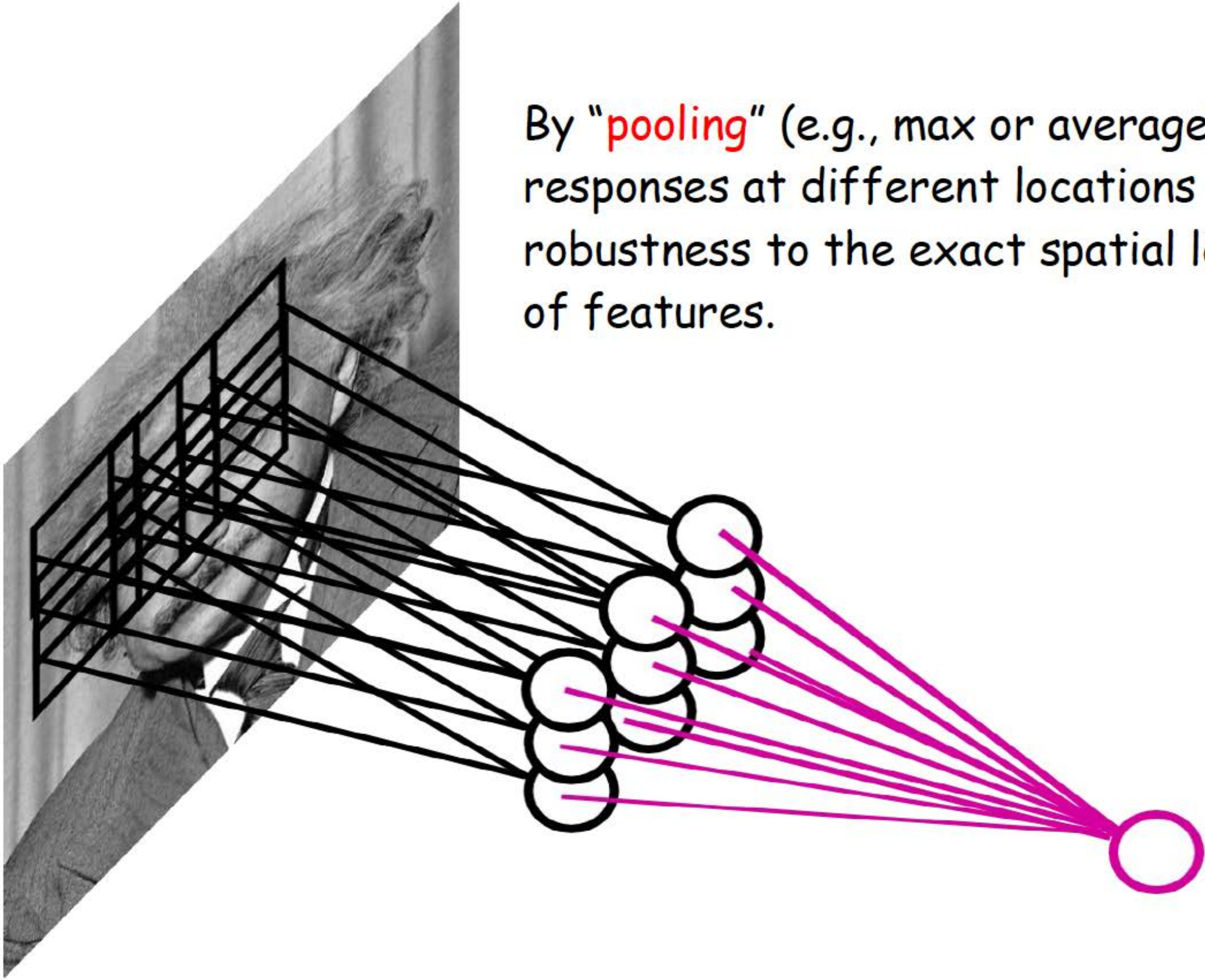
- connect each hidden unit to a small patch of the input
- share the weight across hidden units

This is called: **convolutional network.**

LeCun et al. "Gradient-based learning applied to document recognition" IEEE 1998

CONVOLUTIONAL NET

By "pooling" (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features.



Large Convnets for Image Classification

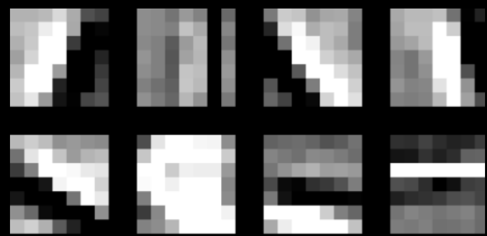
Large Convnets for Image Classification

- Operations in each layer
- Architecture
- Training
- Results

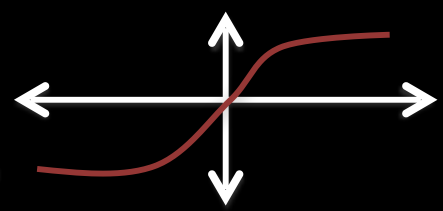
Components of Each Layer

Pixels / Features

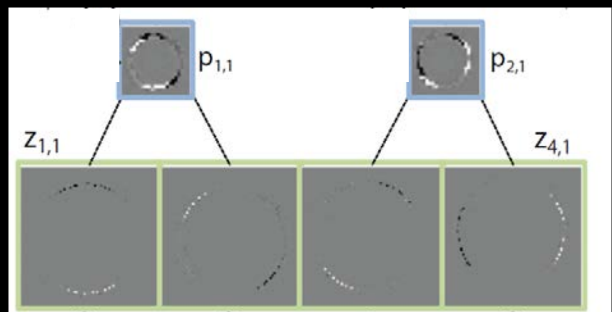
Filter with Dictionary (convolutional or tiled)



+ Non-linearity



Spatial/Feature (Sum or Max)



[Optional]

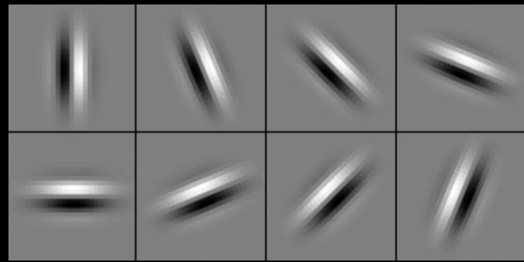
Normalization between feature responses

Output Features

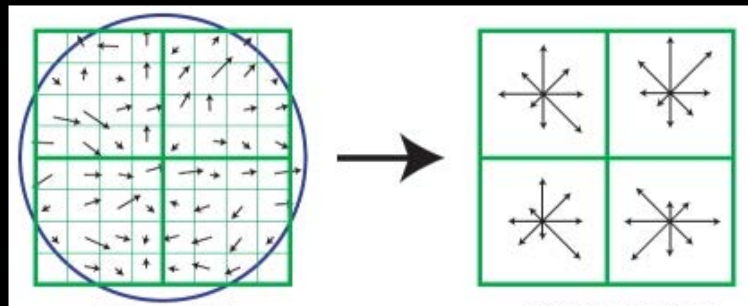
Compare: SIFT Descriptor

Image Pixels

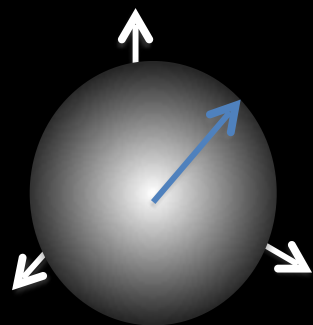
Apply Gabor filters



Spatial pool (Sum)



Normalize to unit length

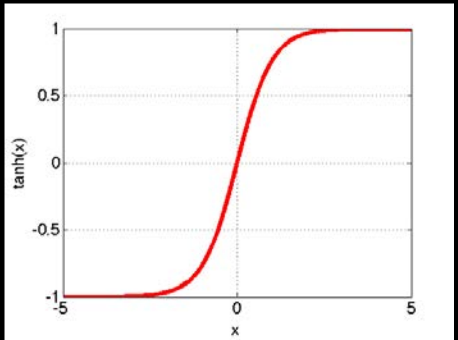


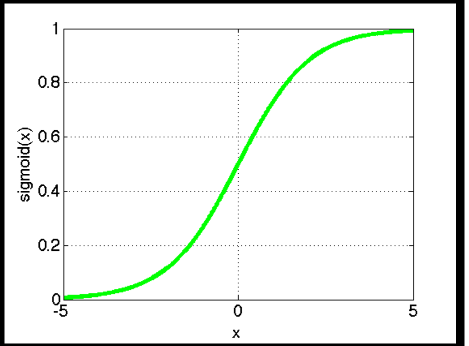
Feature Vector

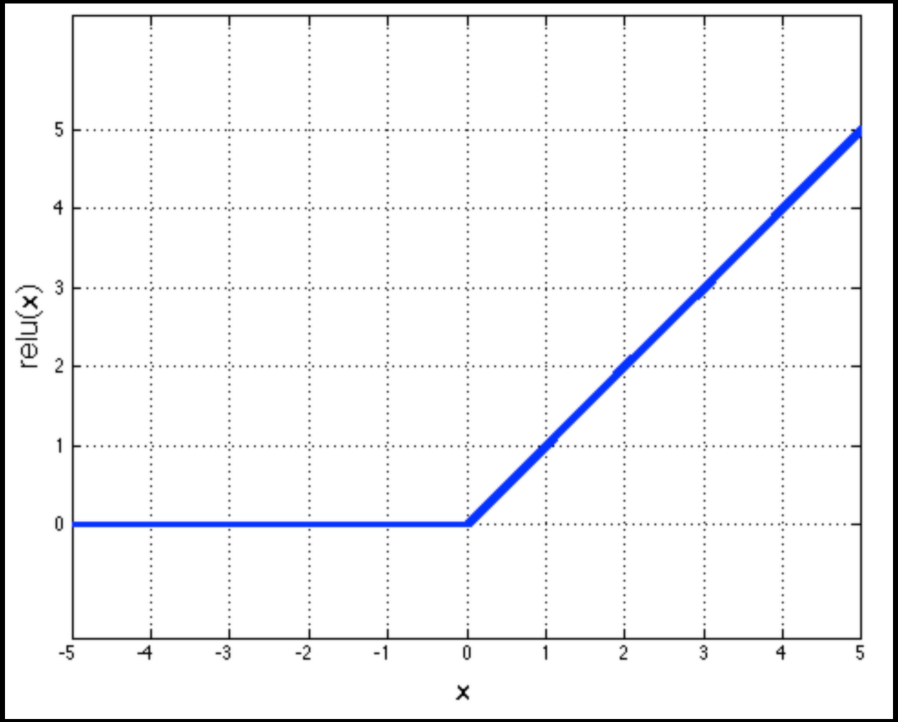
Non-Linearity

- Non-linearity
  - Per-feature independent
  - Tanh
  - Sigmoid: 1/(1+exp(-x))
  - Rectified linear
    - Simplifies backprop
    - Makes learning faster
    - Avoids saturation issues

  → Preferred option

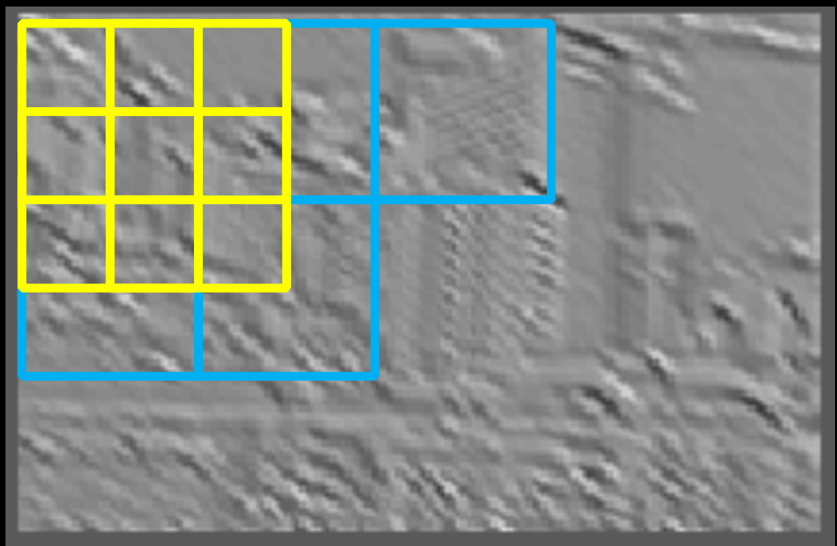




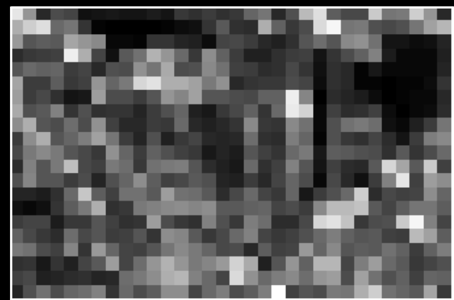


Pooling

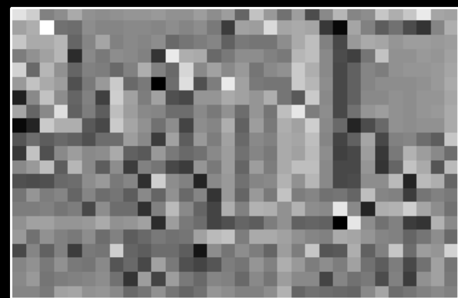
- Spatial Pooling
  - Non-overlapping / overlapping regions
  - Sum or max
  - Boureau et al. ICML’10 for theoretical analysis



Max



Sum



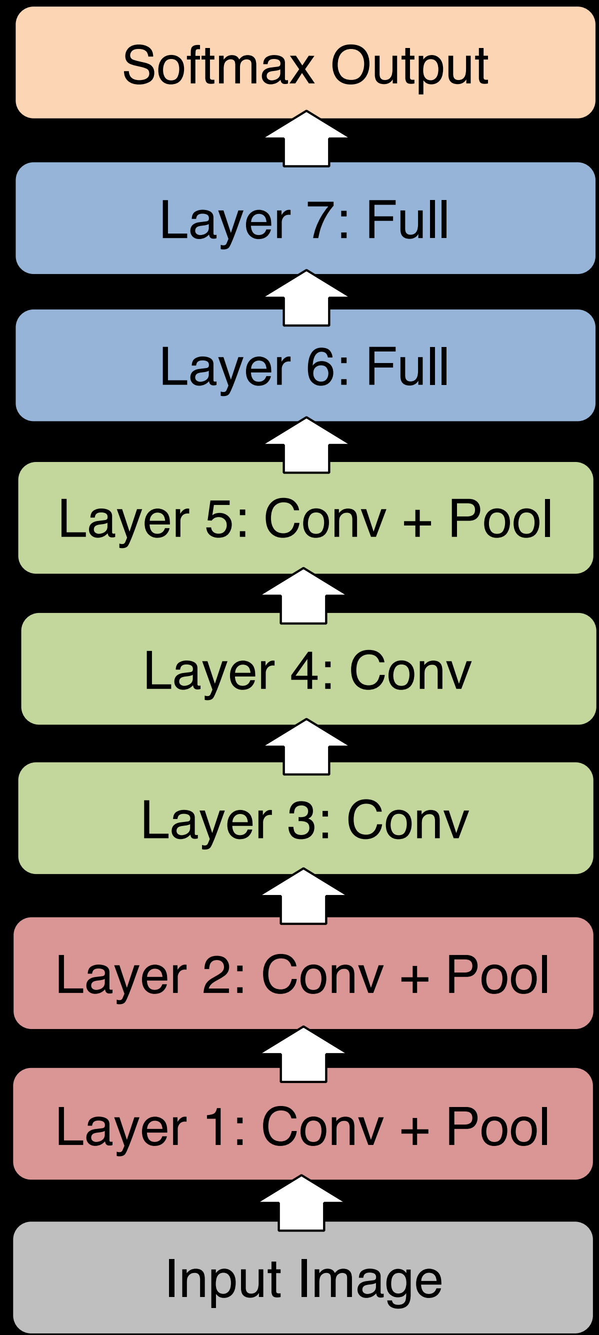
Architecture

Importance of Depth

Architecture of Krizhevsky et al.

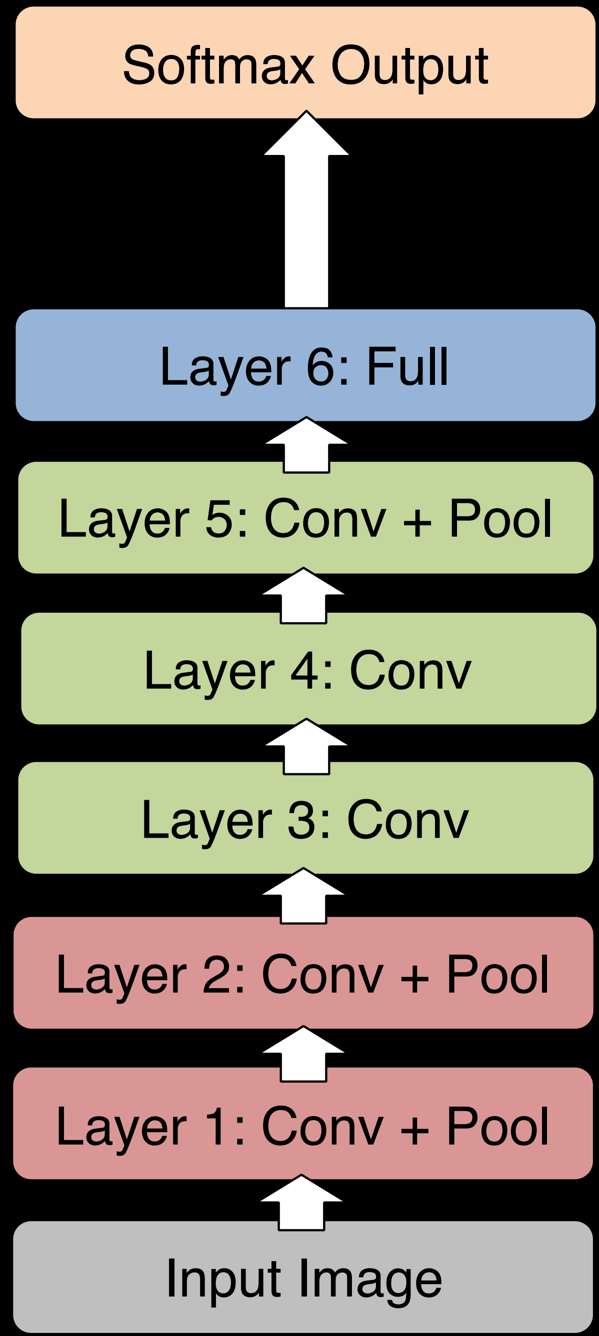
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:

  18.1% top-5 error



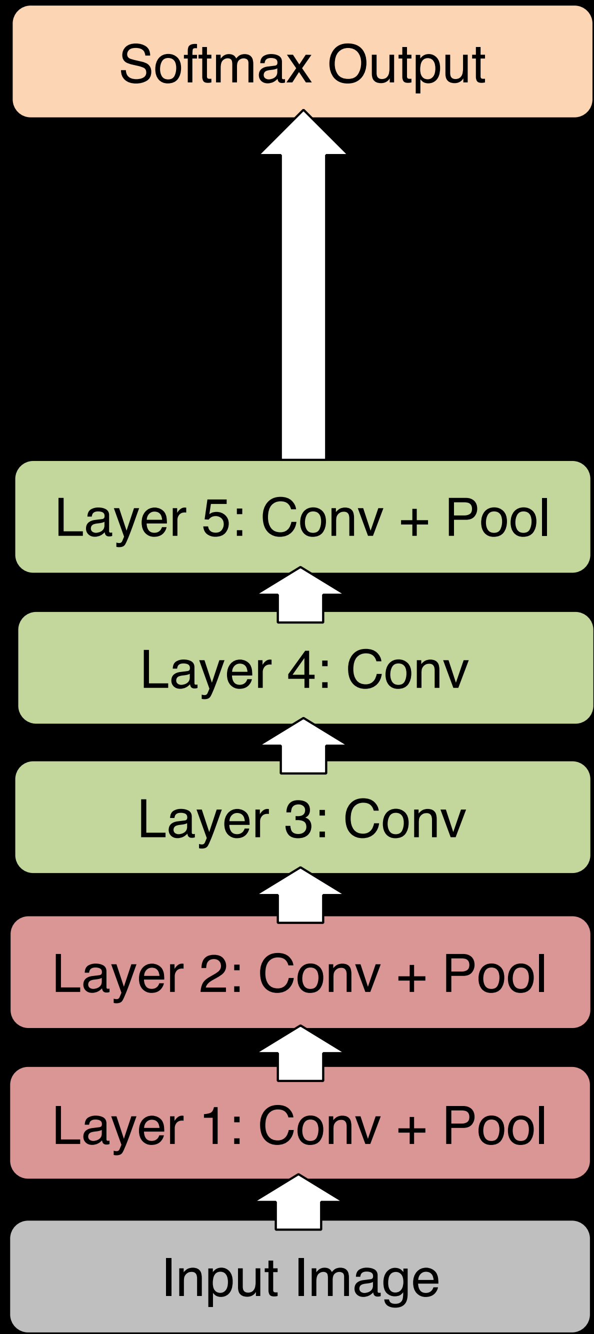
Architecture of Krizhevsky et al.

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



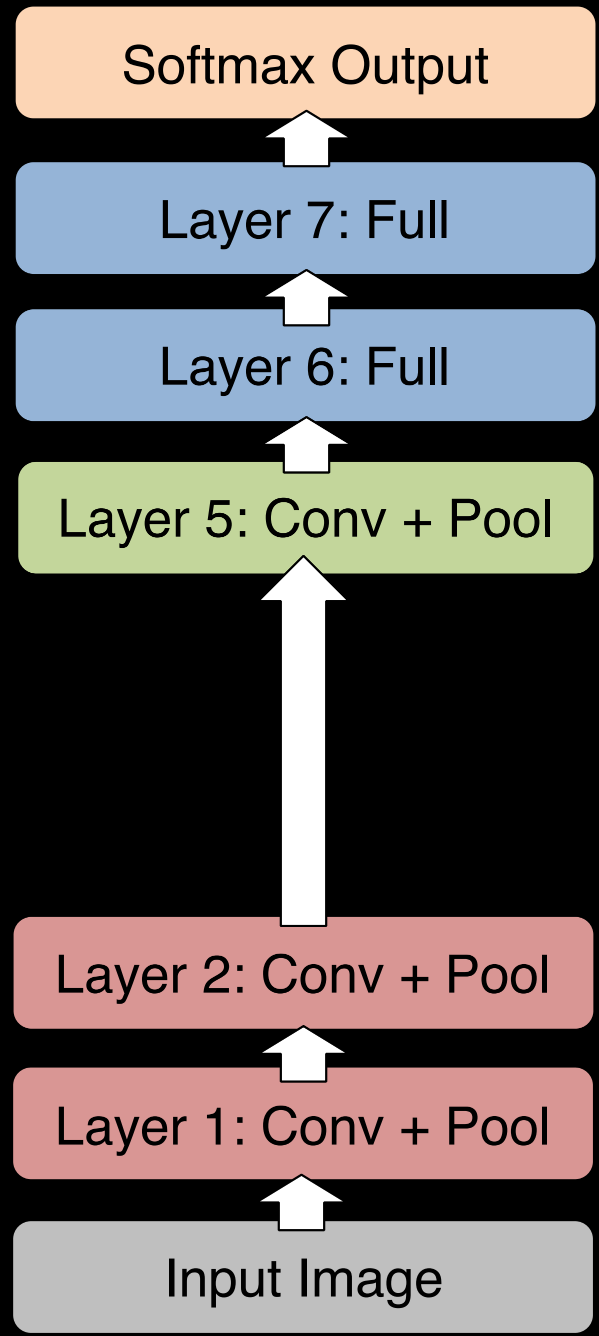
Architecture of Krizhevsky et al.

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



Architecture of Krizhevsky et al.

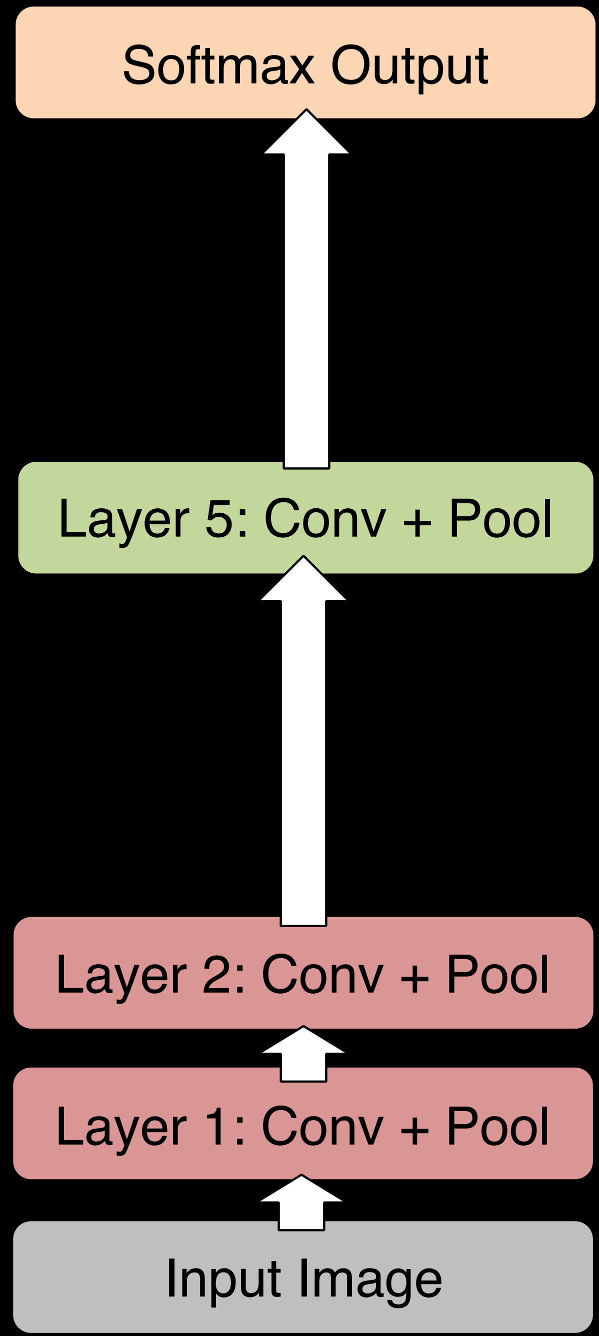
- Now try removing upper feature extractor layers:
  - Layers 3 & 4
- Drop ~1 million parameters
- 3.0% drop in performance



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

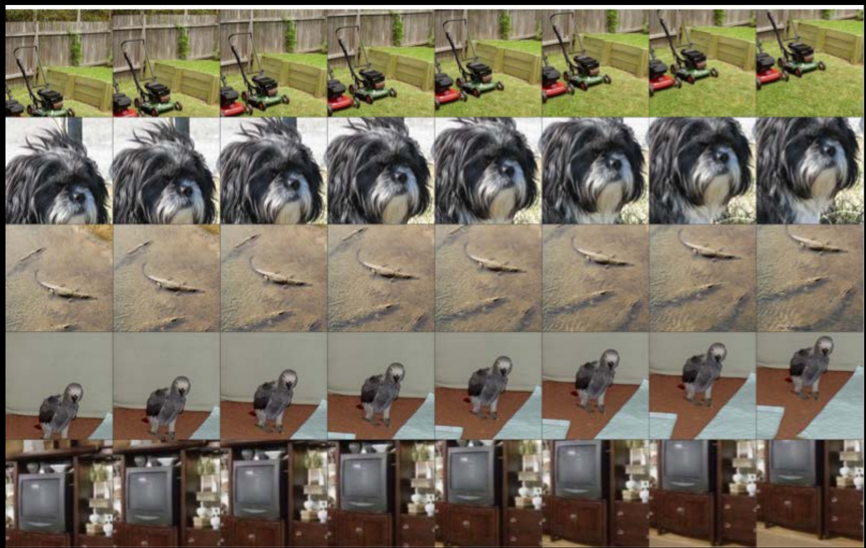


Tapping off Features at each Layer

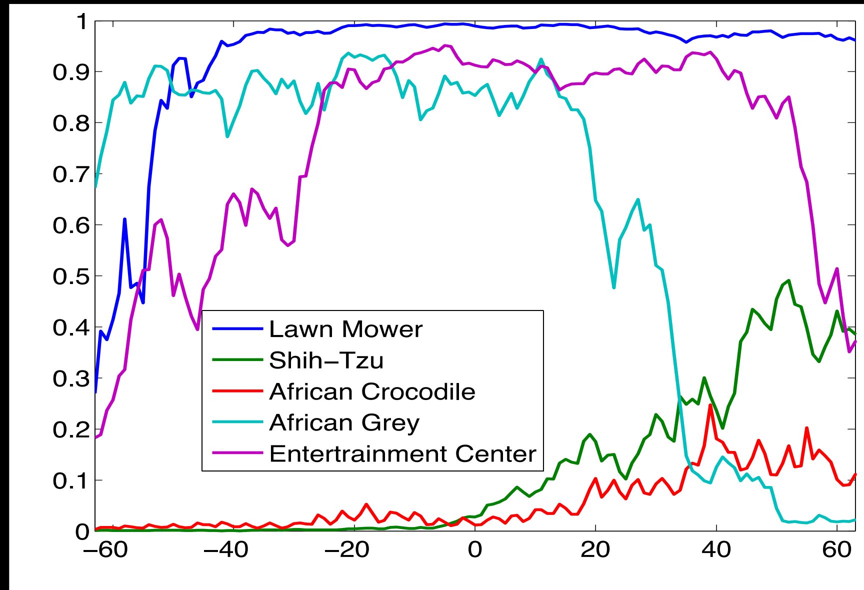
Plug features from each layer into linear SVM or soft-max

	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)	$\mathbf{85.4 \pm 0.4}$	$\mathbf{72.6 \pm 0.1}$

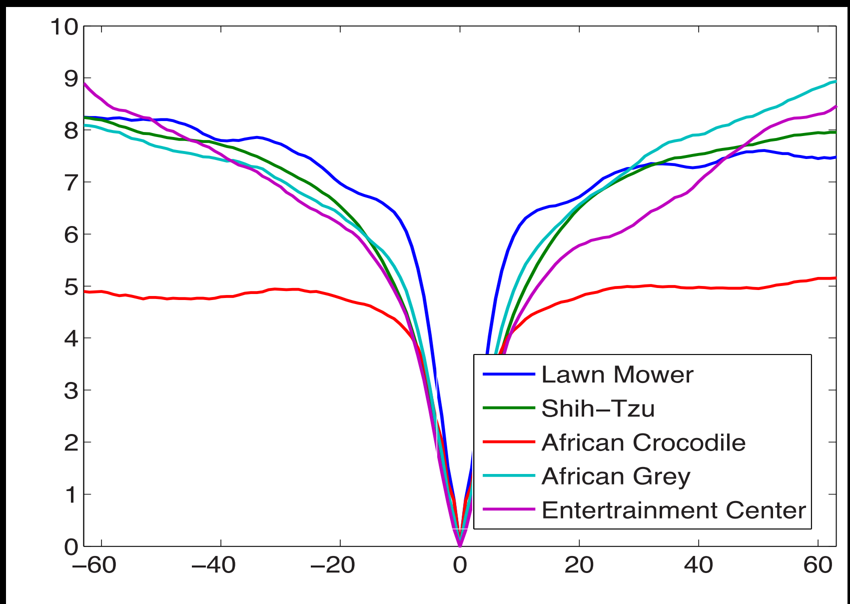
Translation (Vertical)



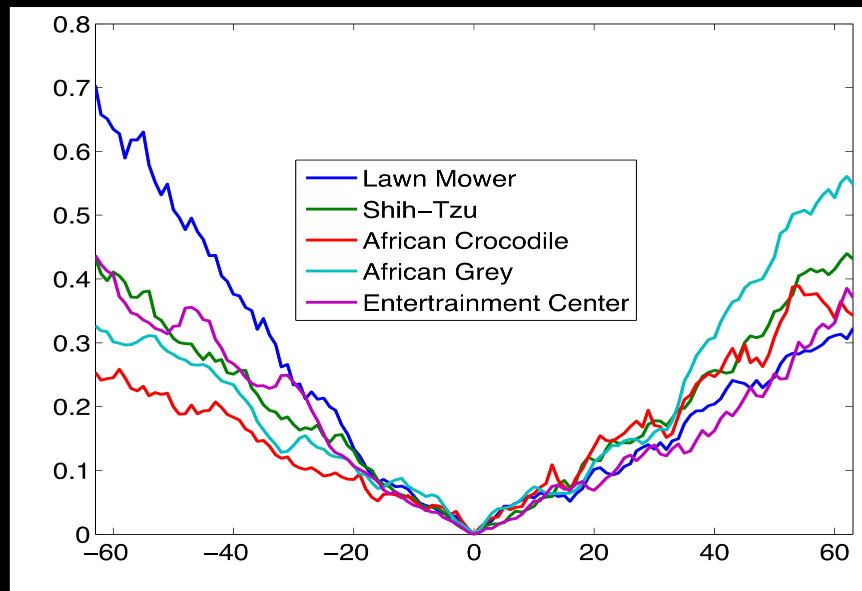
Output



Layer 1

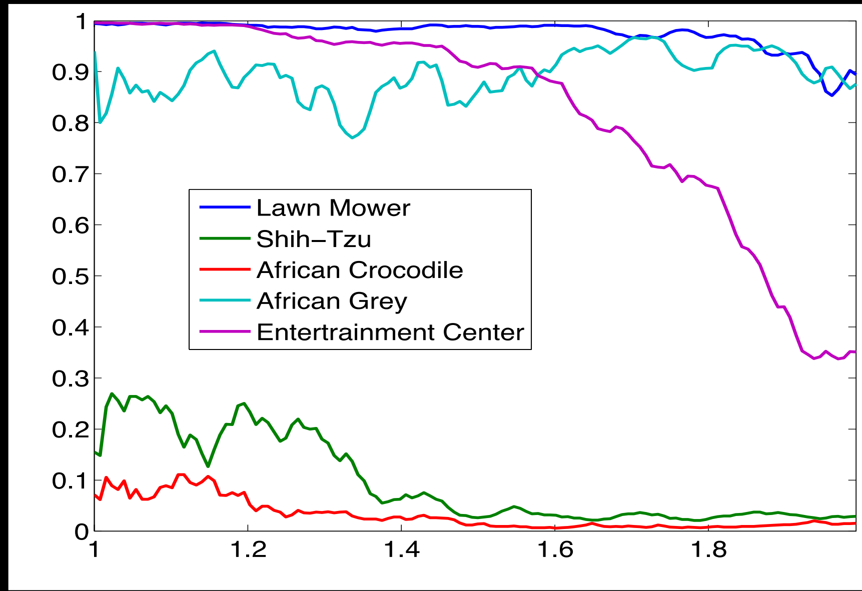


Layer 7

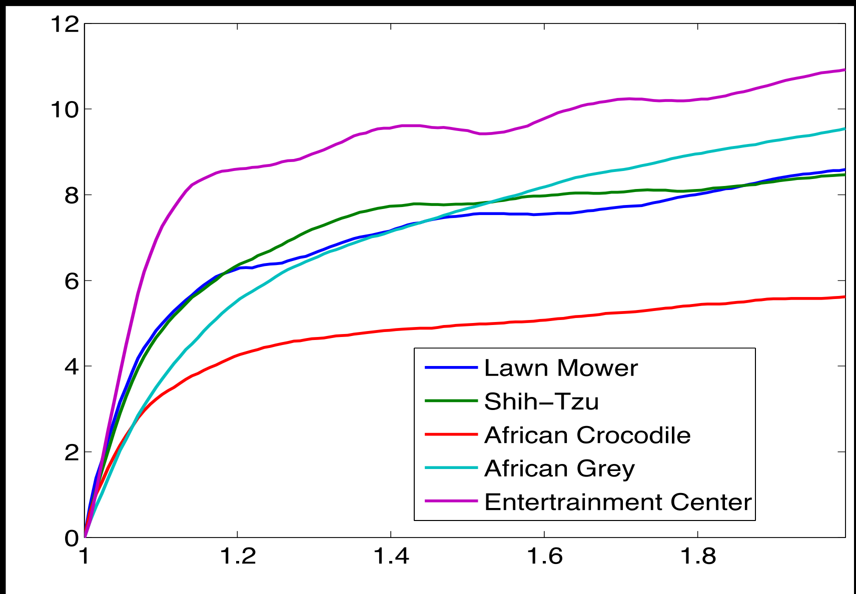


Scale Invariance

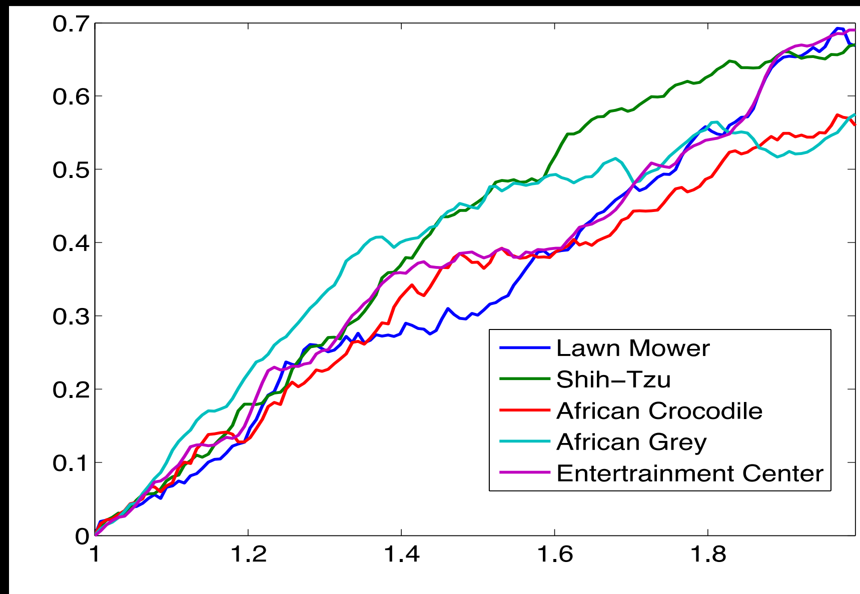
Output



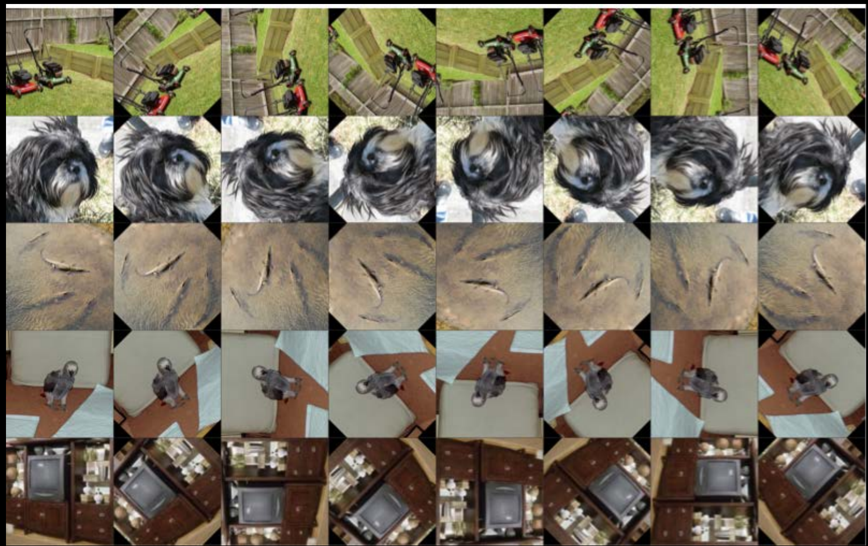
Layer 1



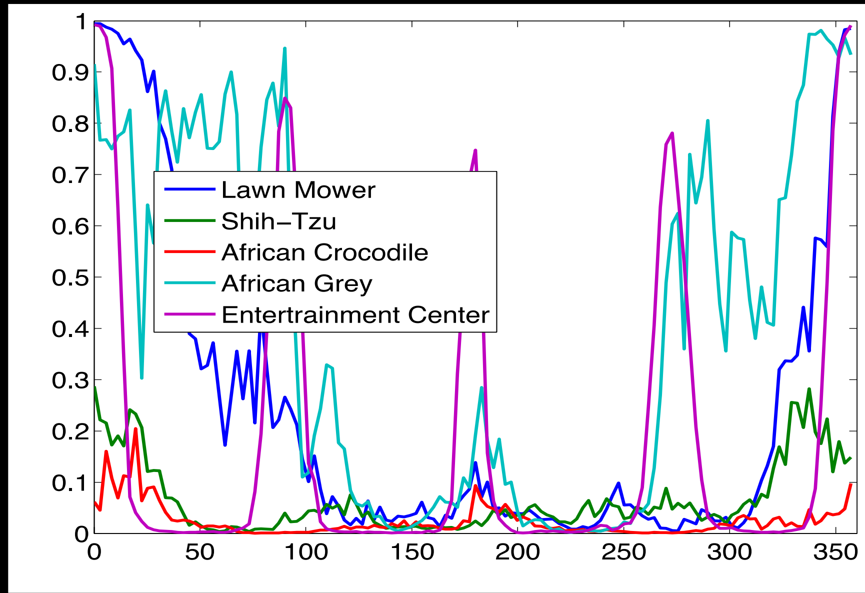
Layer 7



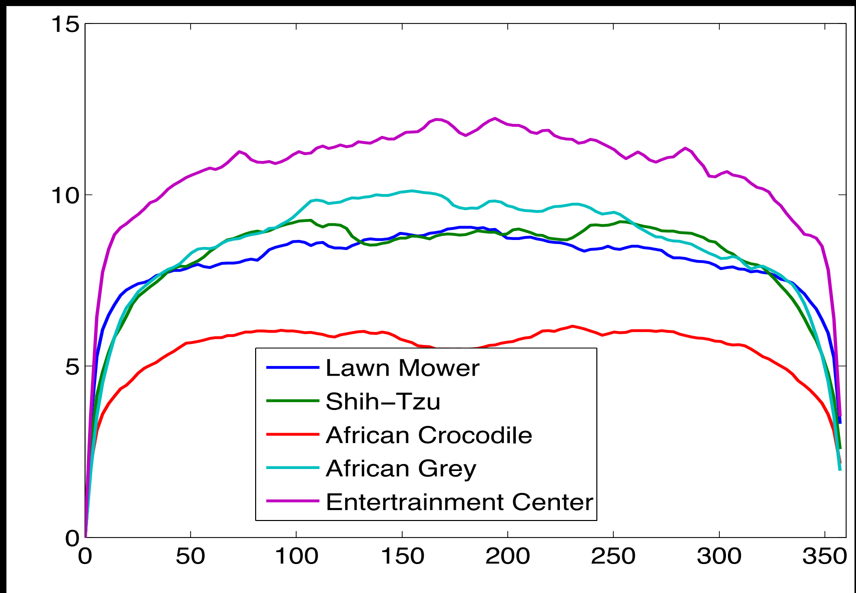
Rotation Invariance



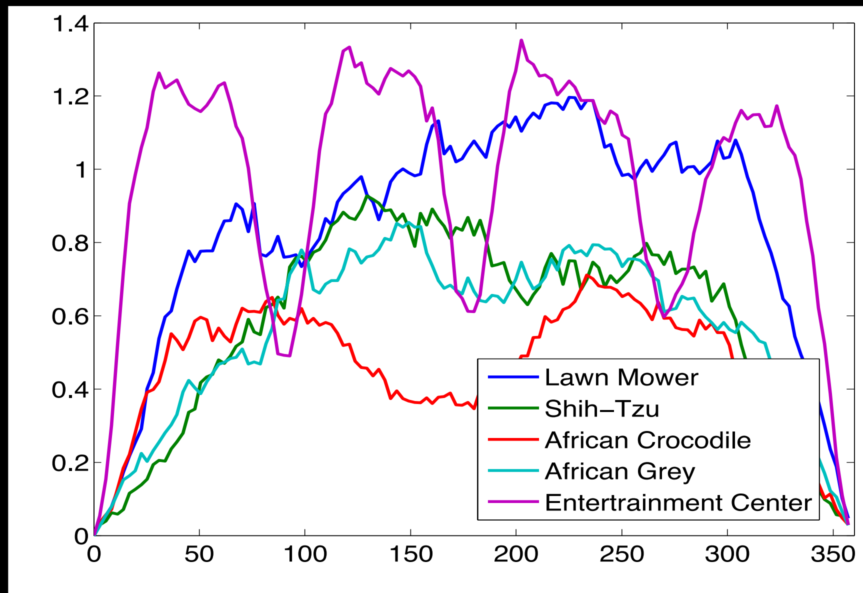
Output



Layer 1



Layer 7



Visualizing ConvNets

Visualizing Convnets

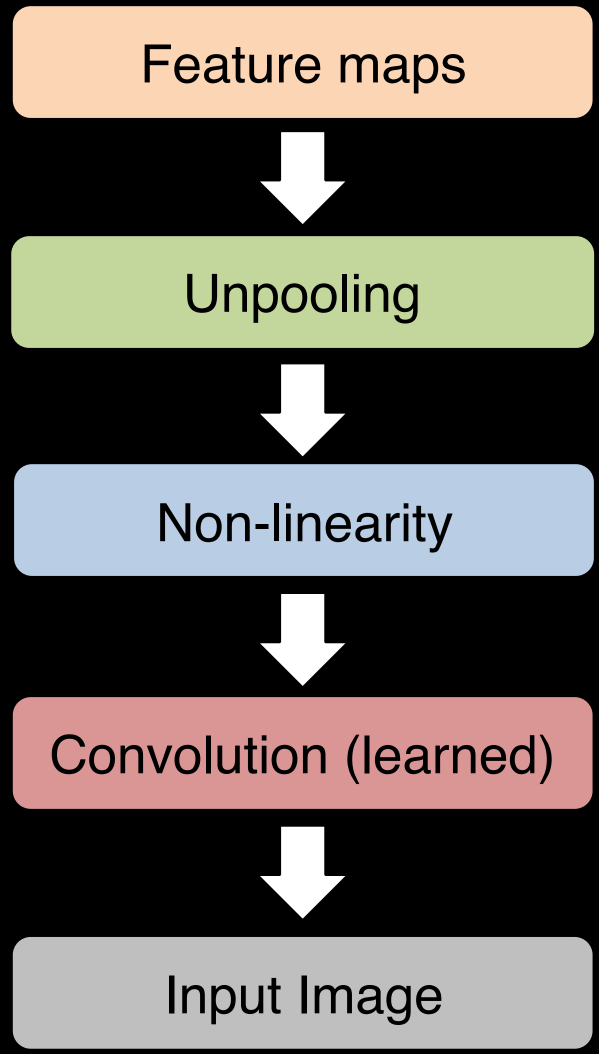
- Raw coefficients of learned filters in higher layers difficult to interpret

- Several approaches look to optimize input to maximize activity in a high-level feature
  - Erhan et al. [Tech Report 2009]
  - Le et al. [NIPS 2010]
  - Depend on initialization
  - Model invariance with Hessian about (locally) optimal stimulus

Visualization using Deconvolutional Networks

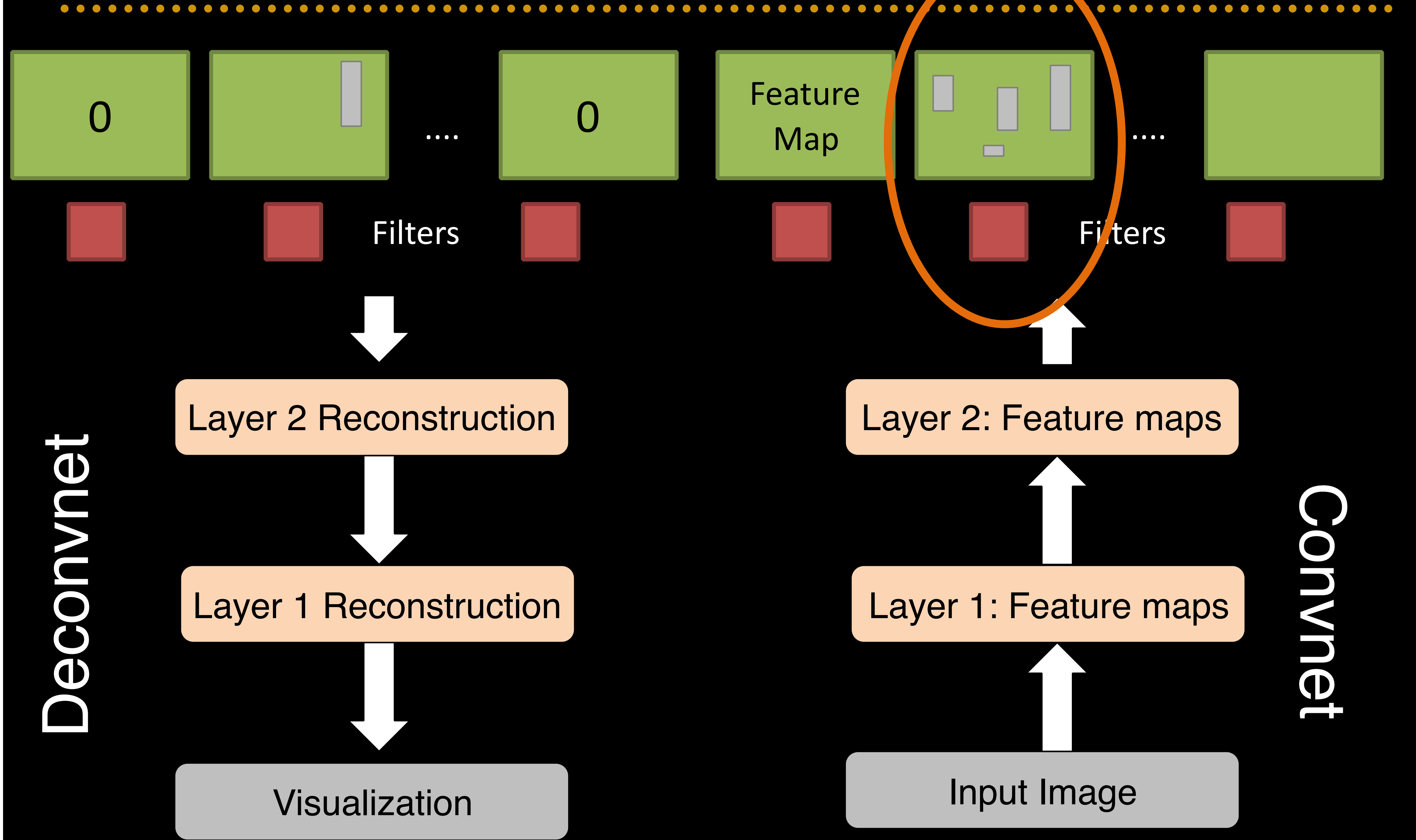
[Zeiler et al. CVPR'10, ICCV'11, arXiv'13]

- Provide way to map activations at high layers back to the input
- Same operations as Convnet, but in reverse:
  - Unpool feature maps
  - Convolve unpooled maps
    - Filters copied from Convnet
- Used here purely as a probe
  - Originally proposed as unsupervised learning method
  - No inference, no learning

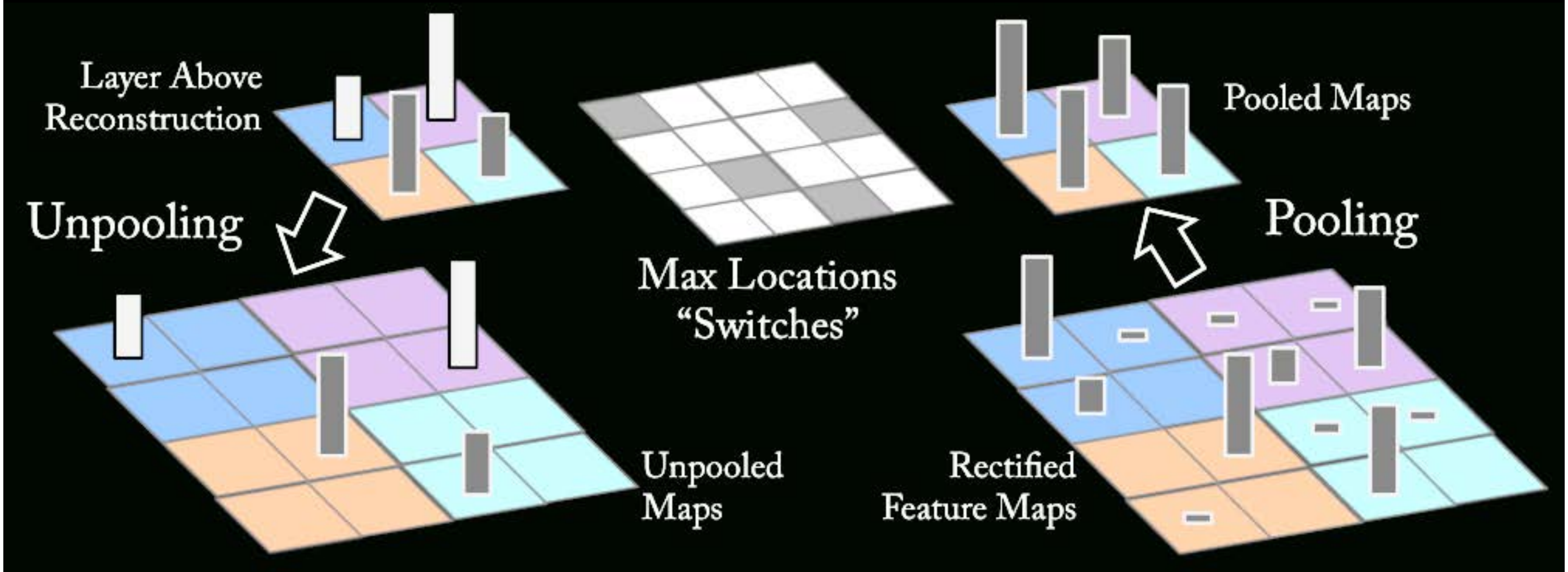


Deconvnet Projection from Higher Layers

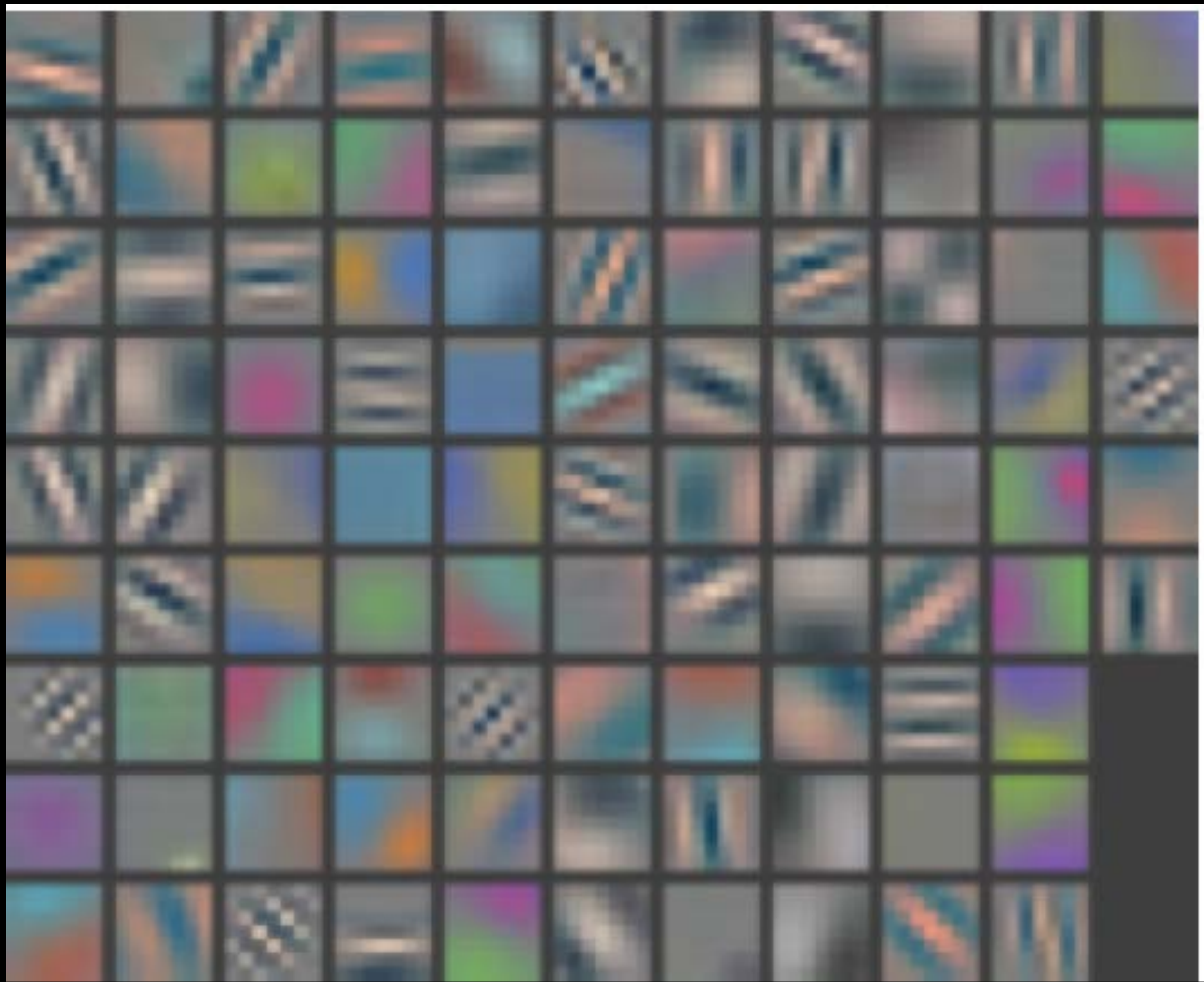
[Zeiler and Fergus. arXiv'13]



Unpooling Operation



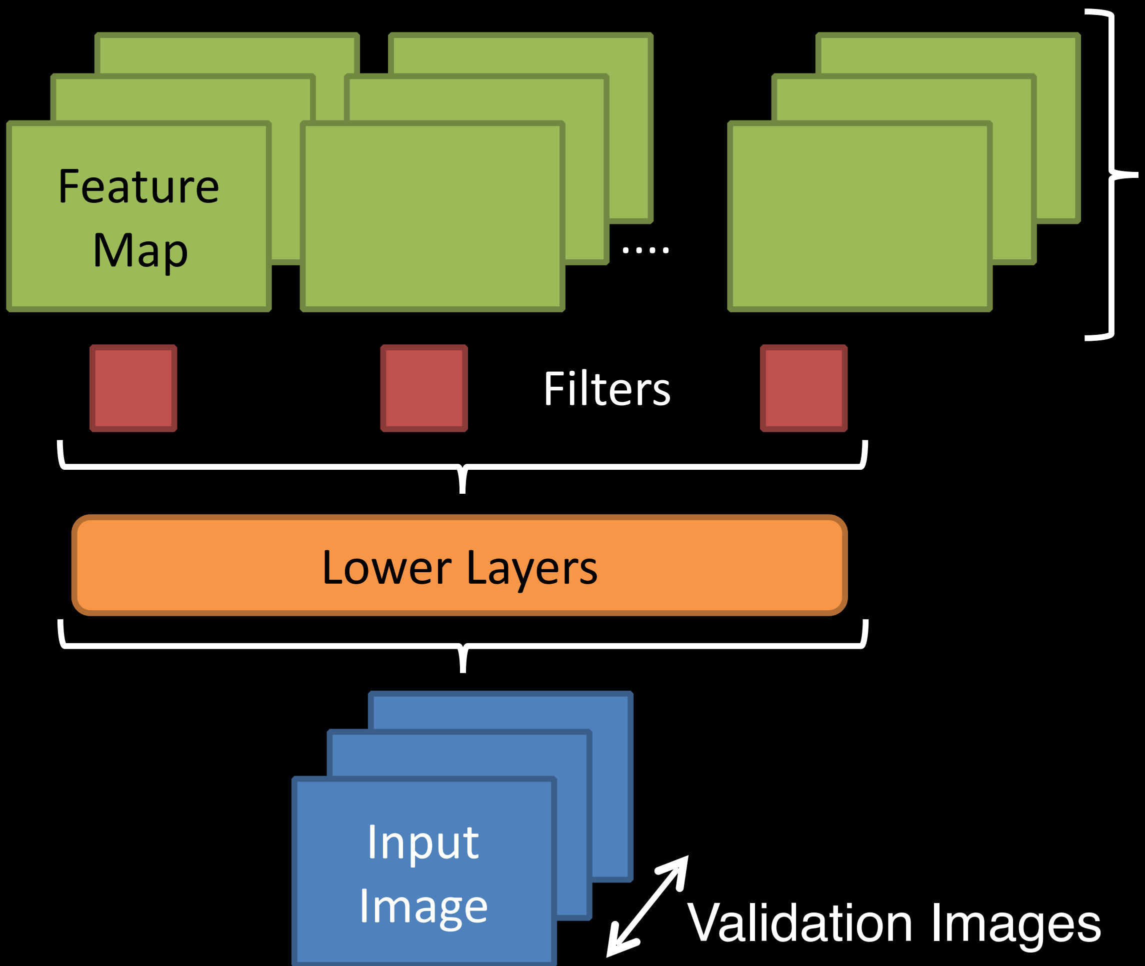
Layer 1 Filters



Visualizations of Higher Layers

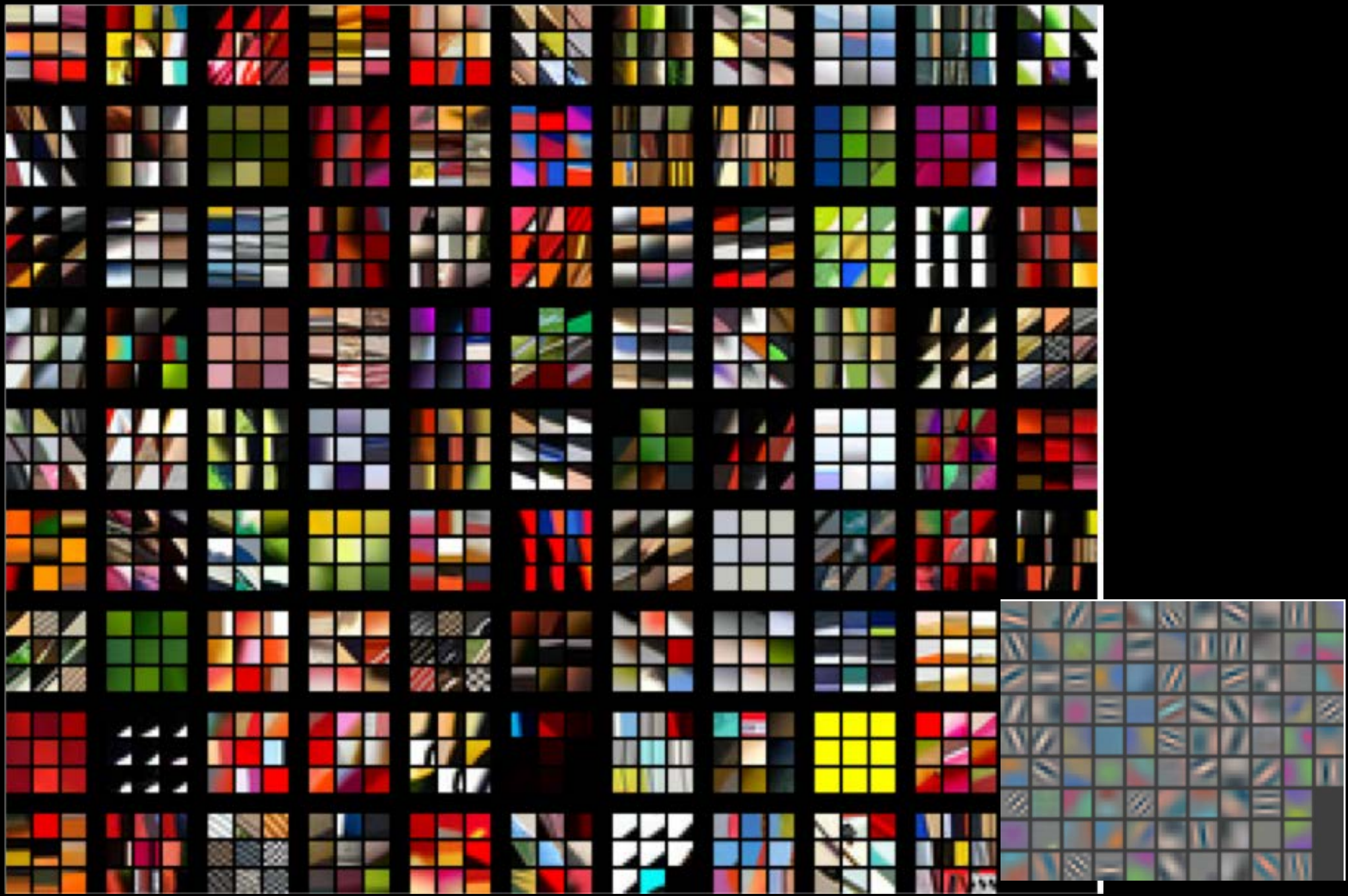
[Zeiler and Fergus. arXiv'13]

- Use ImageNet 2012 validation set
- Push each image through network

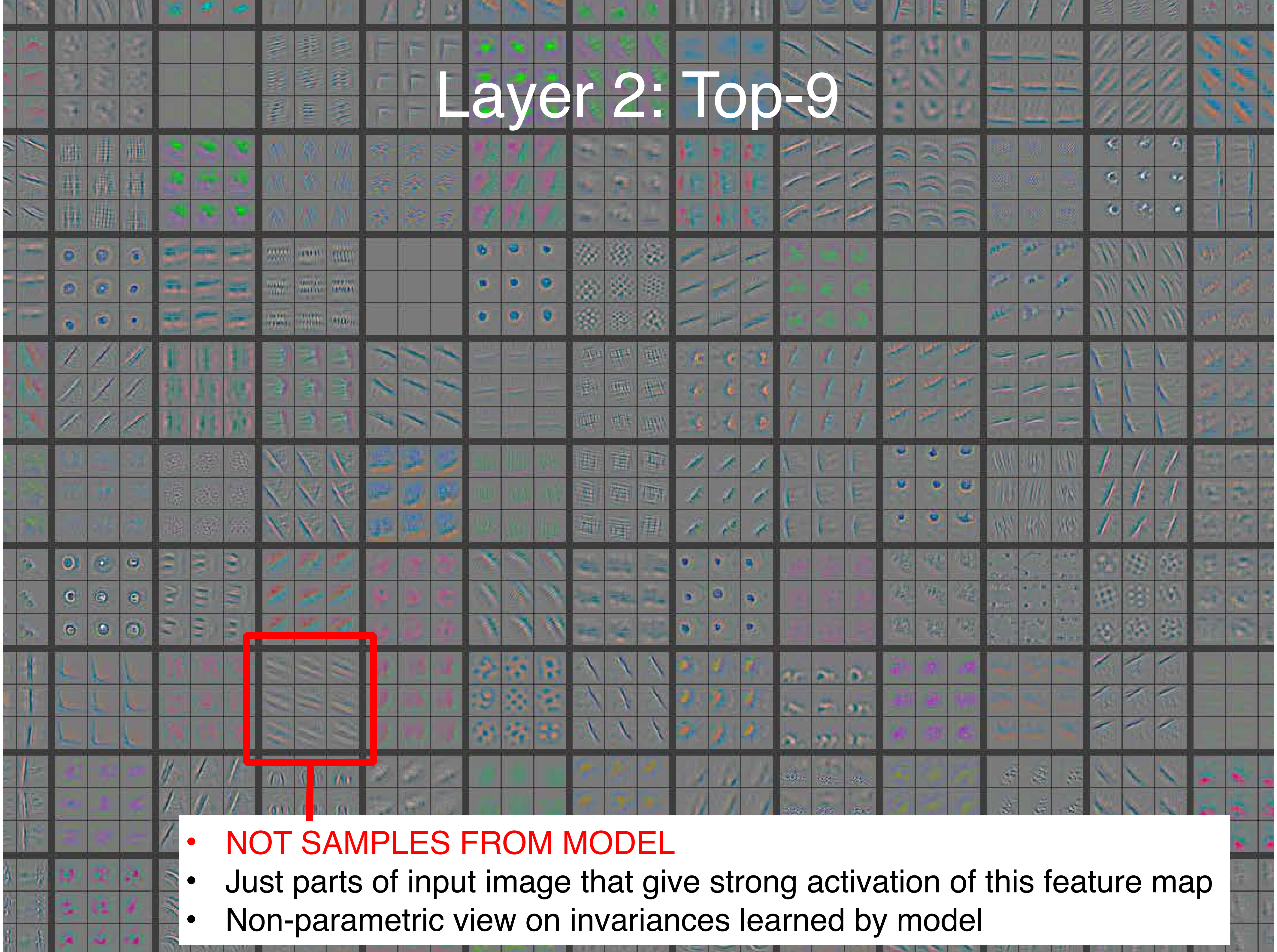


- Take max activation from feature map associated with each filter
- Use Deconvnet to project back to pixel space
- Use pooling "switches" peculiar to that activation

Layer 1: Top-9 Patches



Layer 2: Top-9

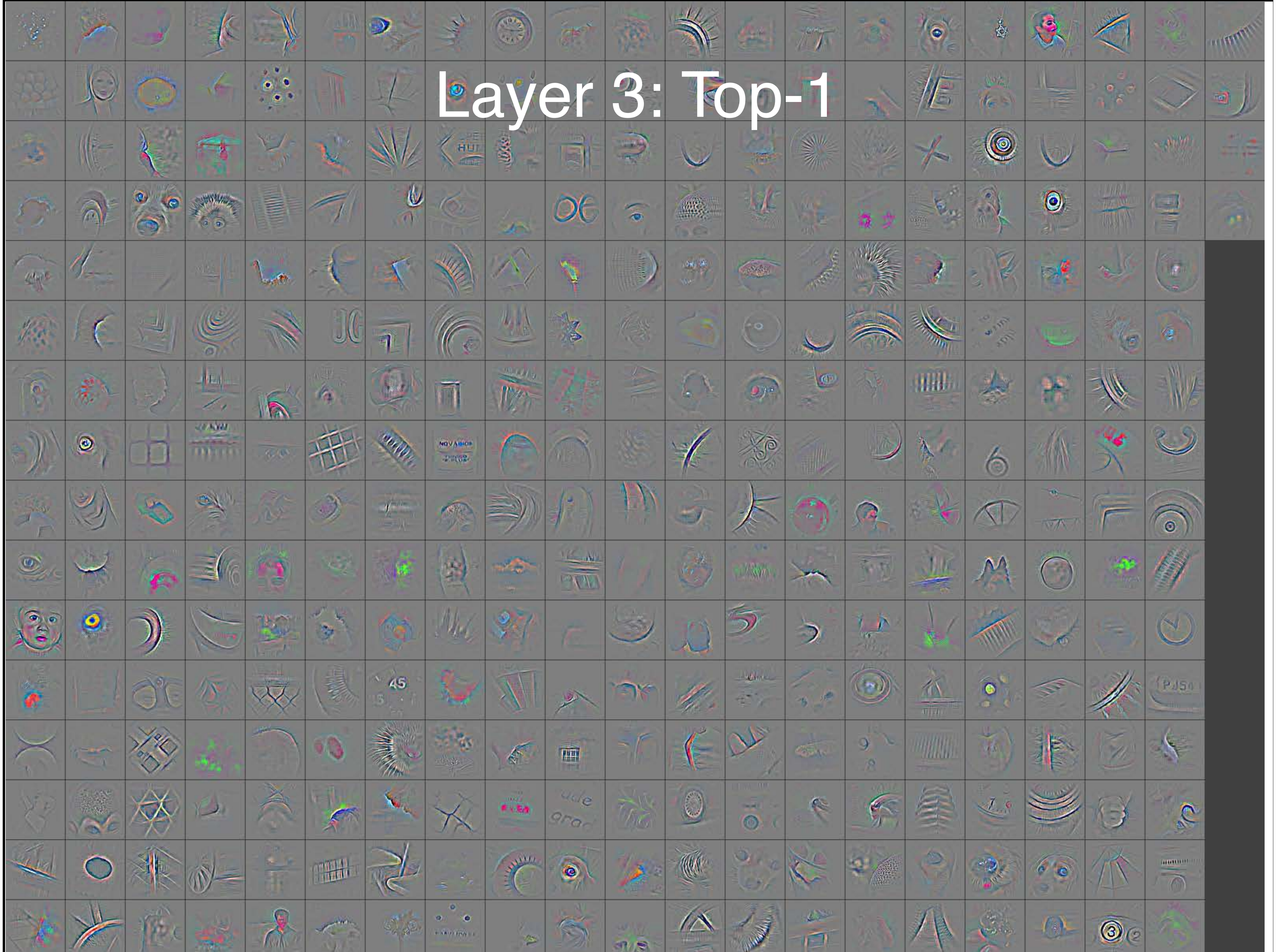


- NOT SAMPLES FROM MODEL
- Just parts of input image that give strong activation of this feature map
- Non-parametric view on invariances learned by model

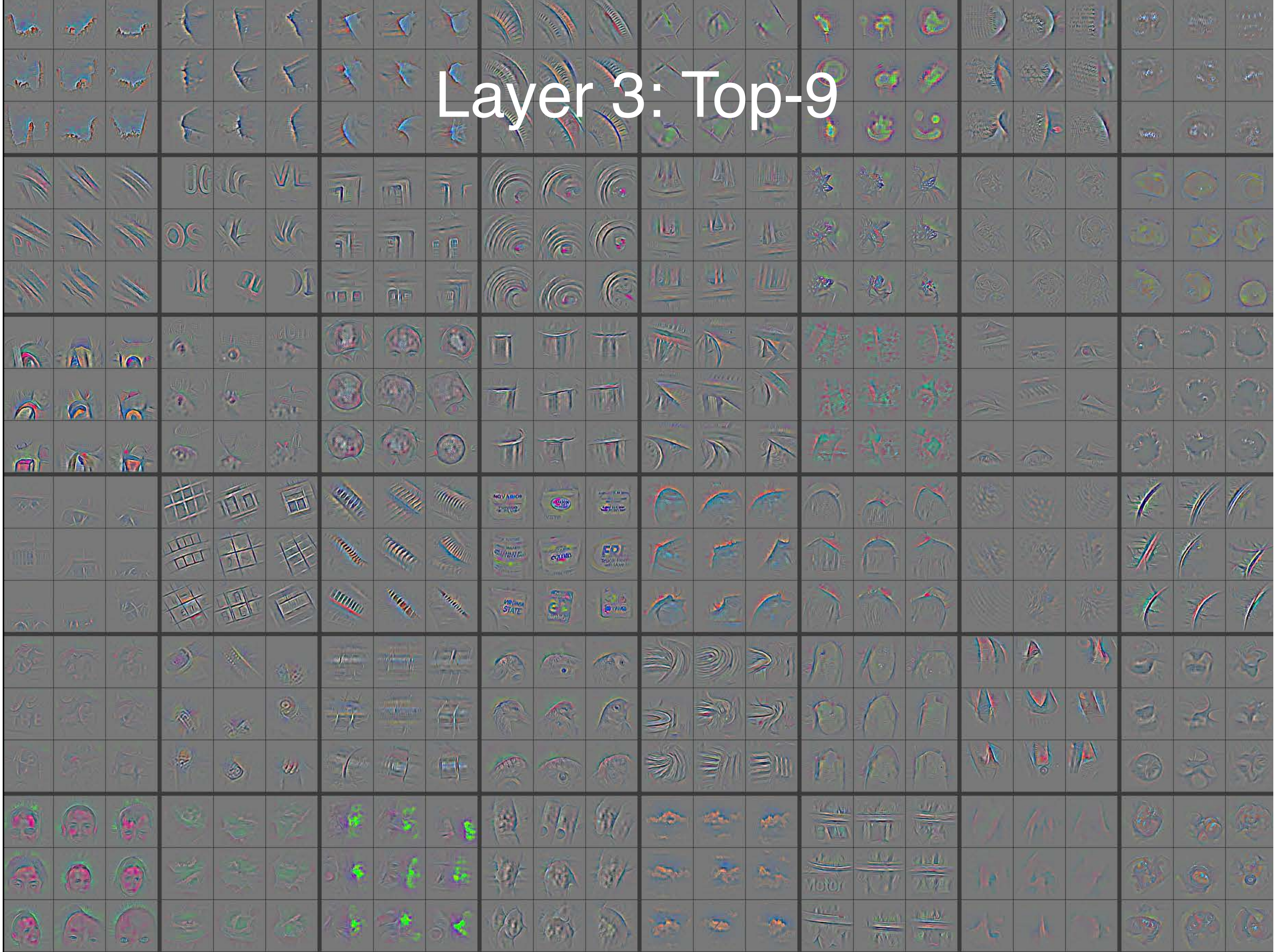
Layer 2: Top-9 Patches

- Patches from validation images that give maximal activation of a given feature map

Layer 3: Top-1

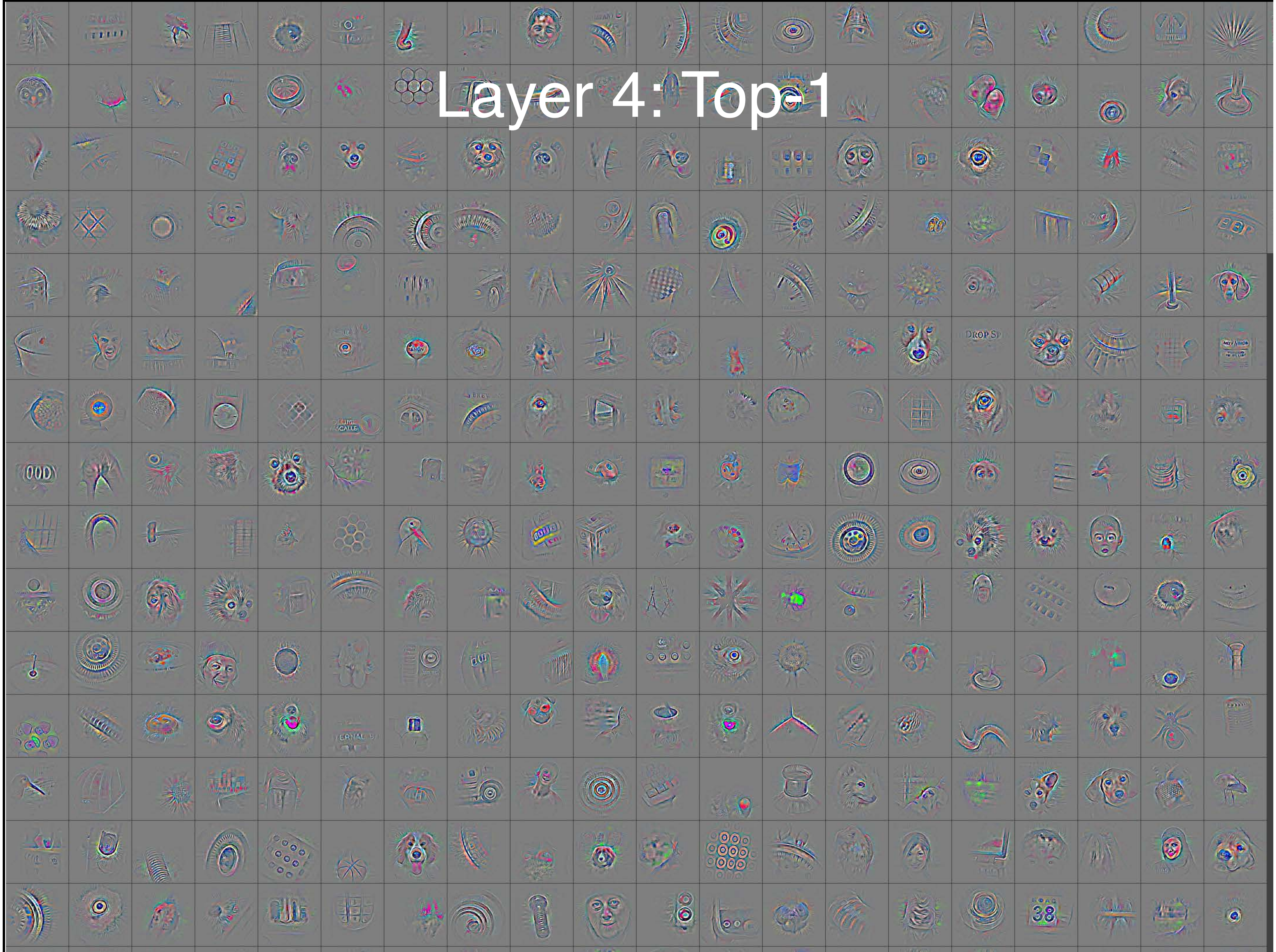


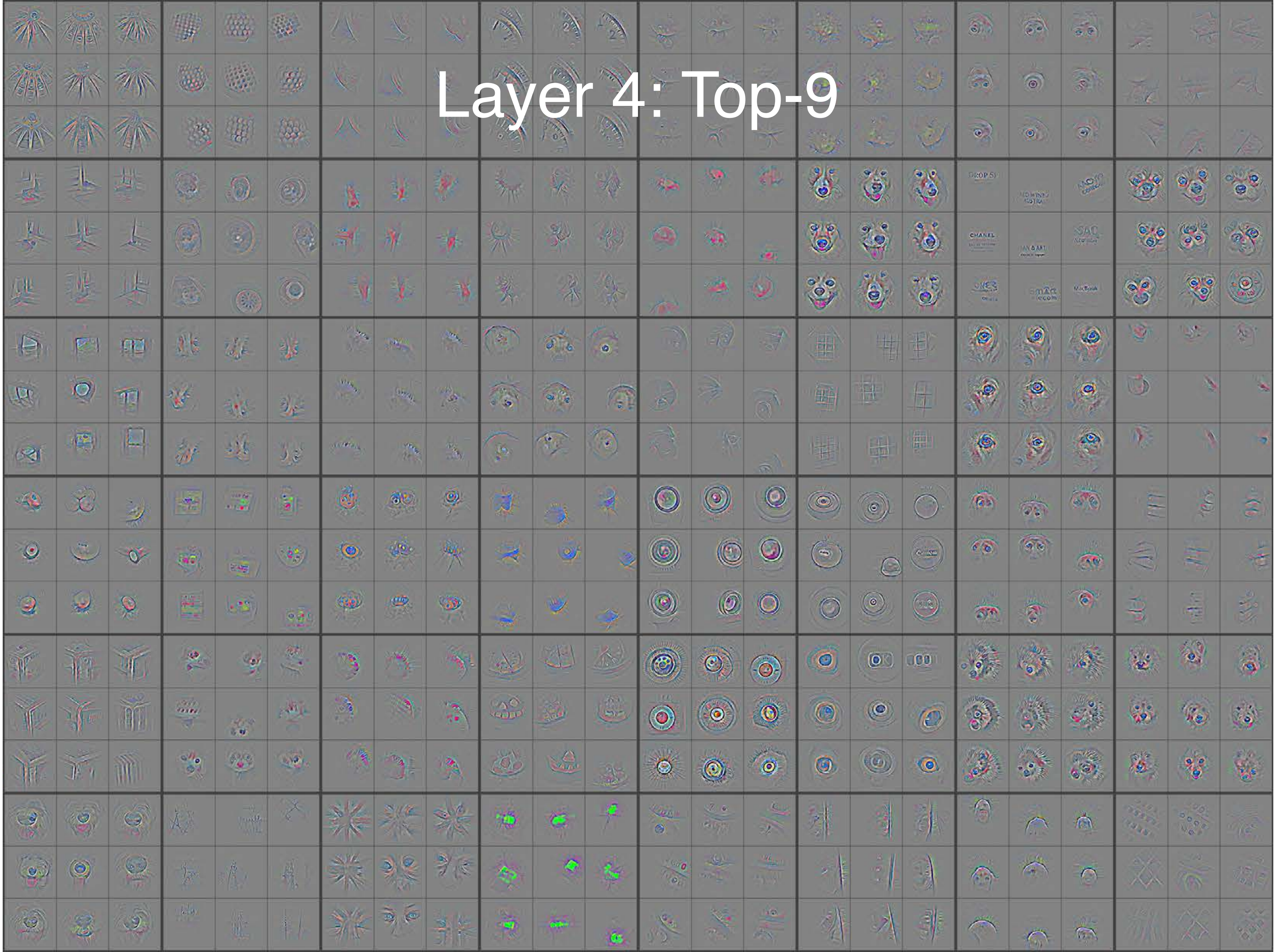
Layer 3: Top-9



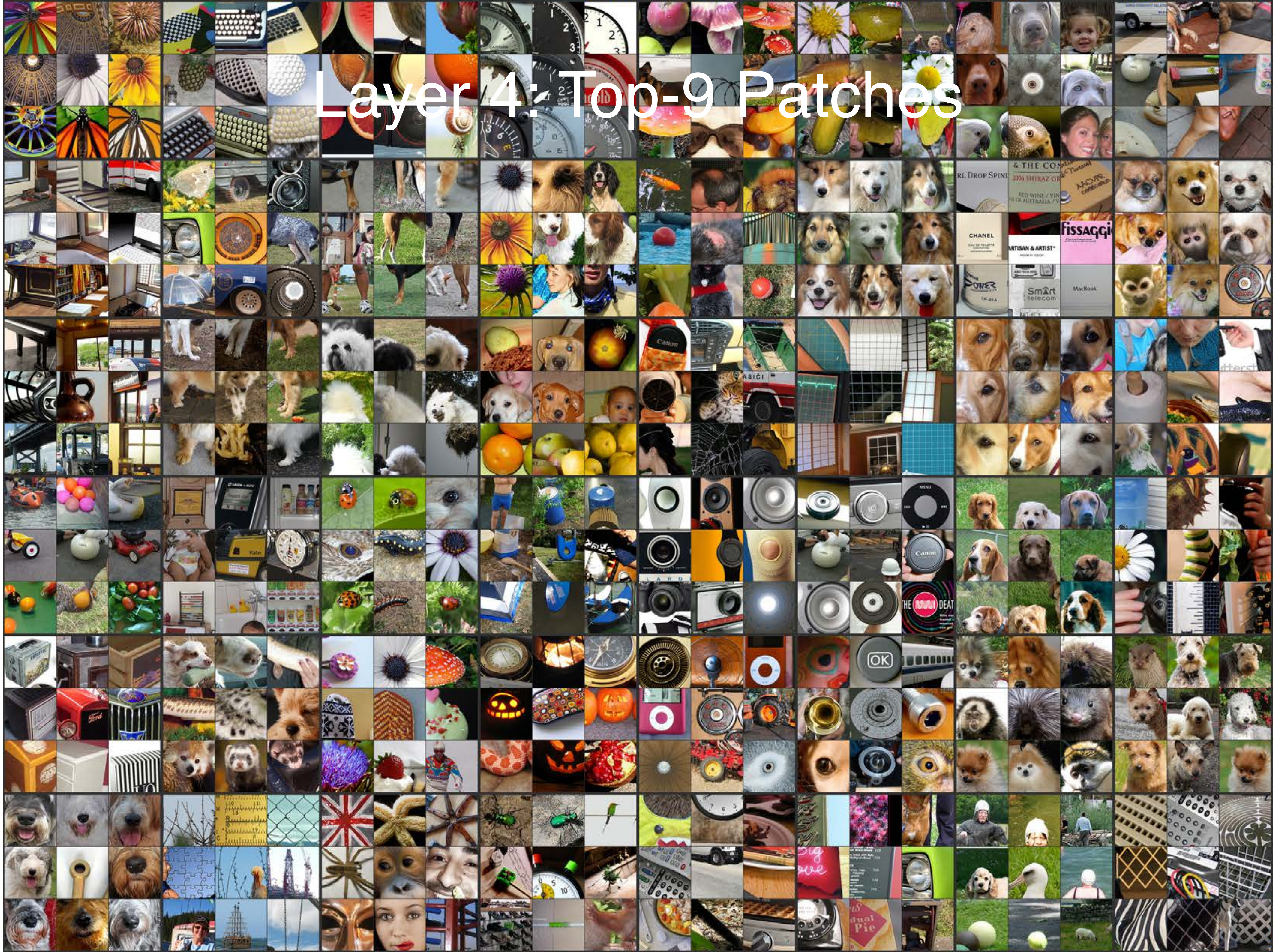
Layer 3: Top-9 Patches

Layer 4: Top-1

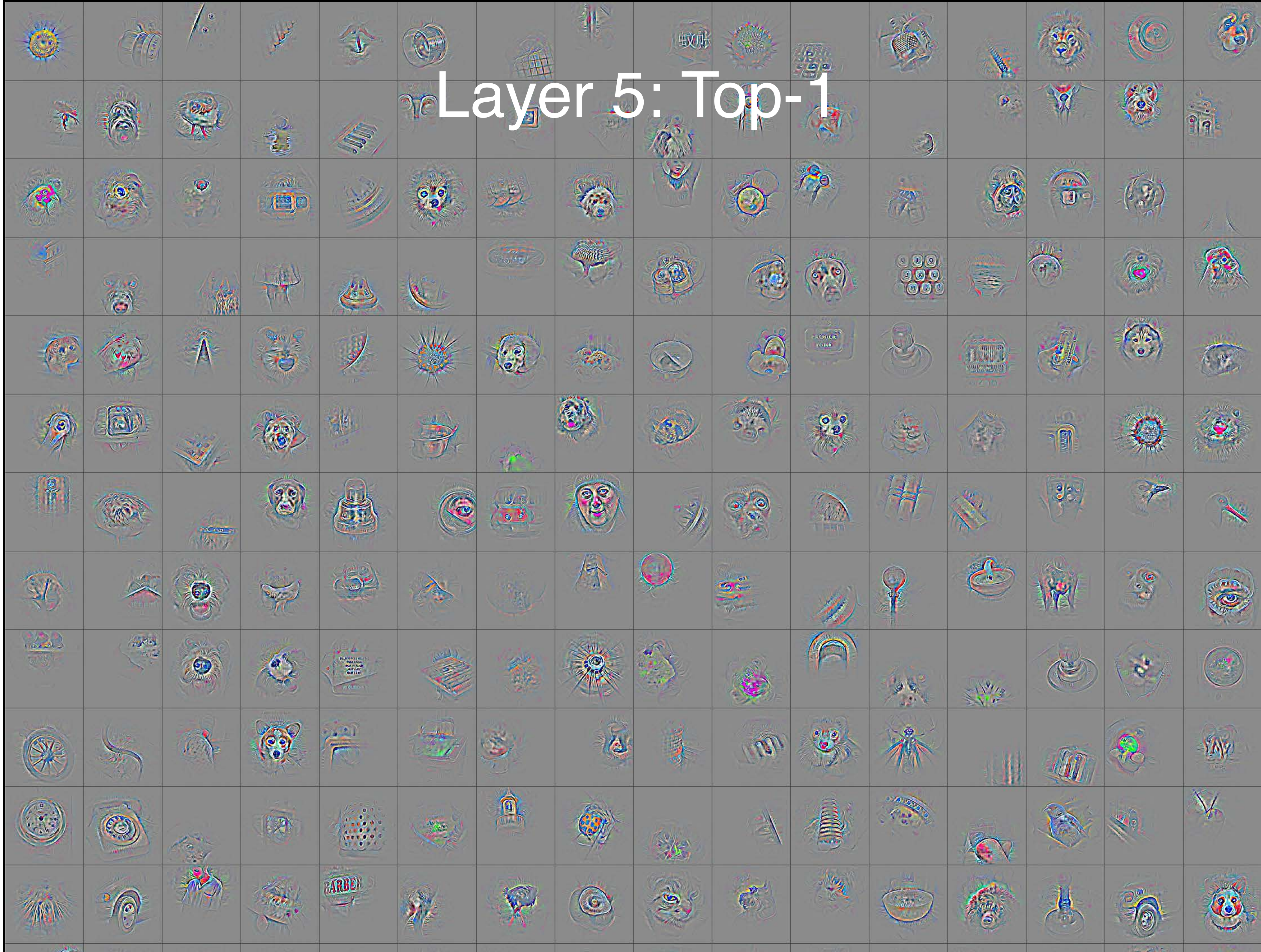


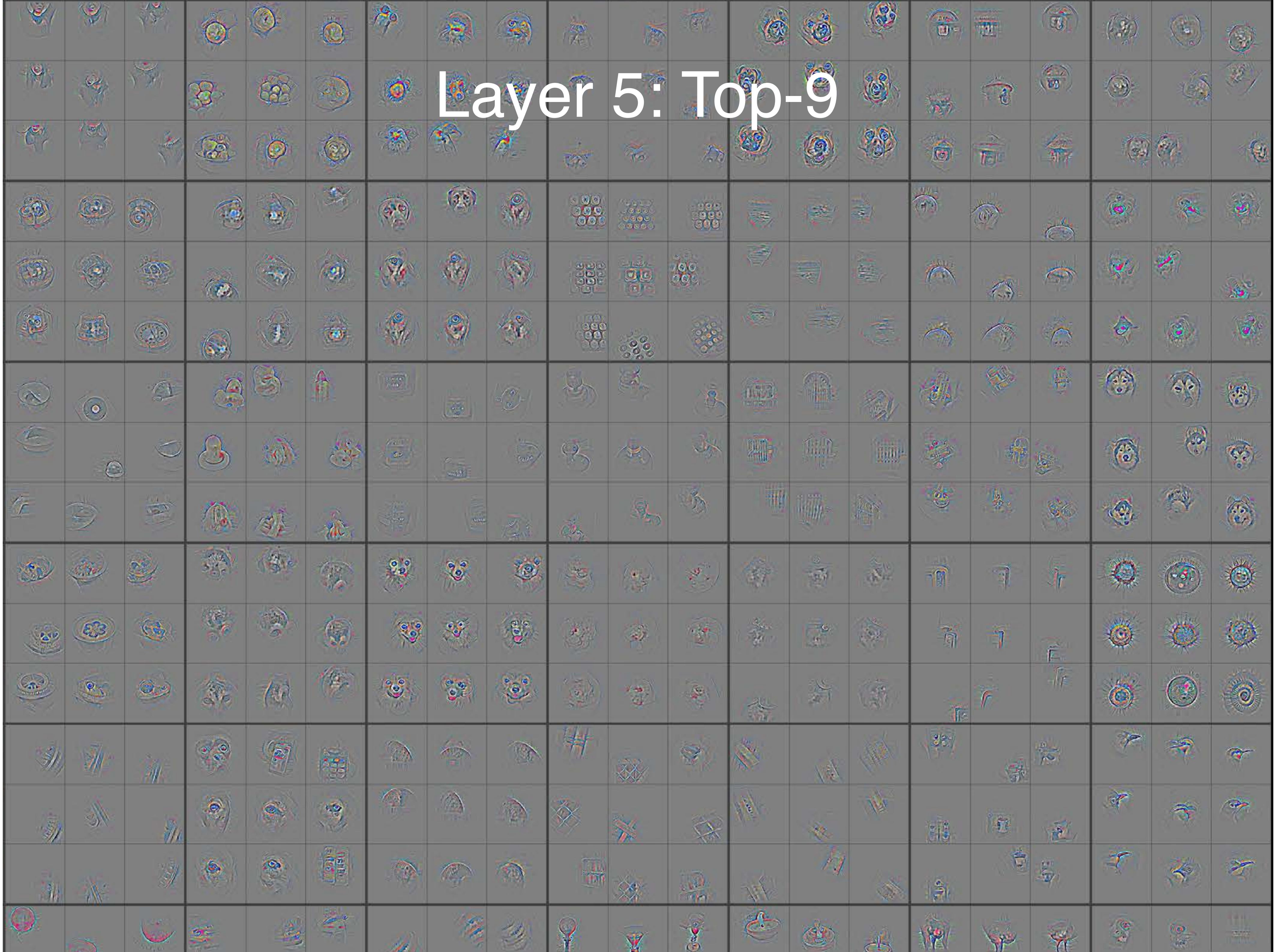

Layer 4: Top-9

Layer 4: Top-9 Patches

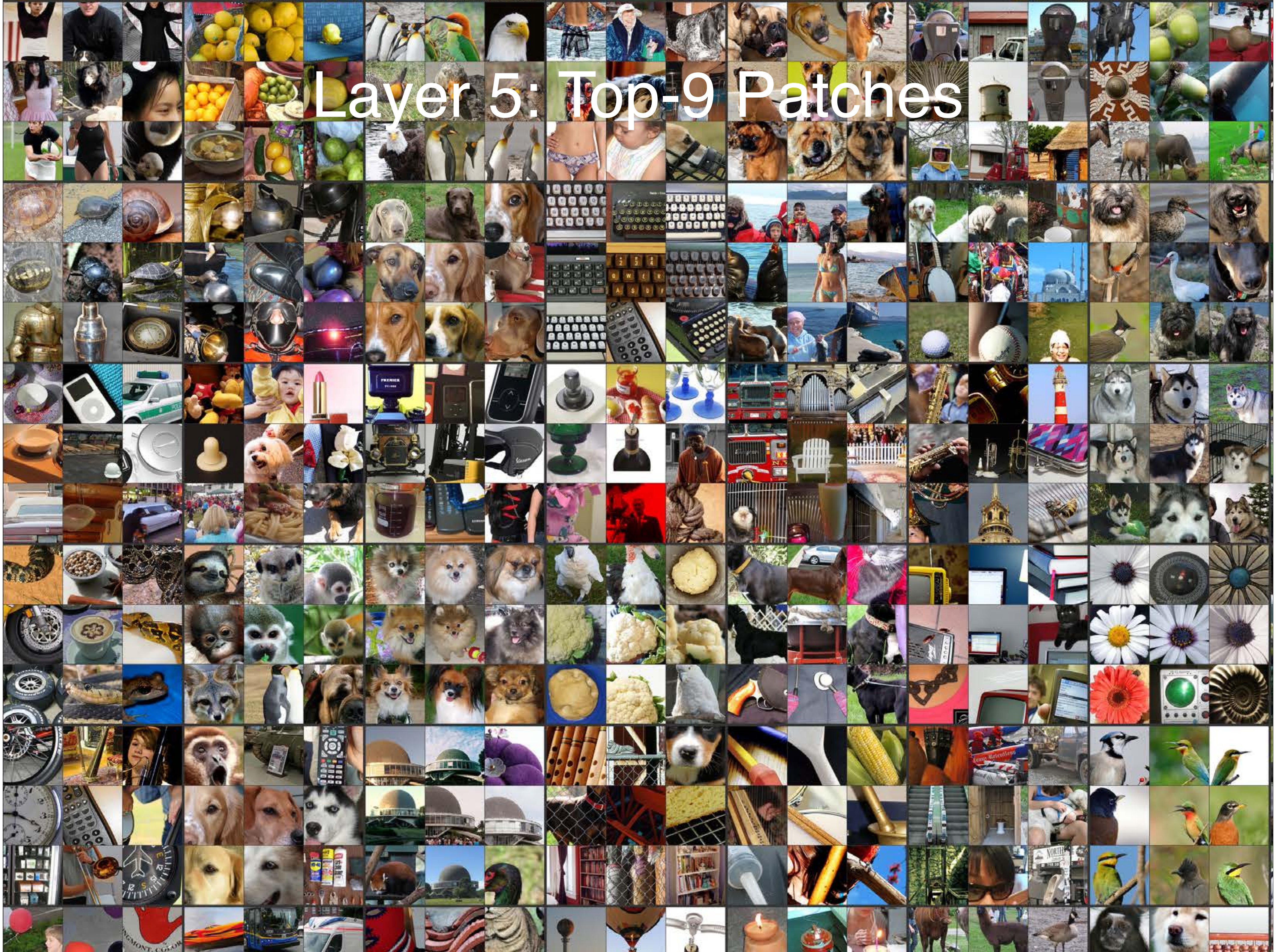


Layer 5: Top-1



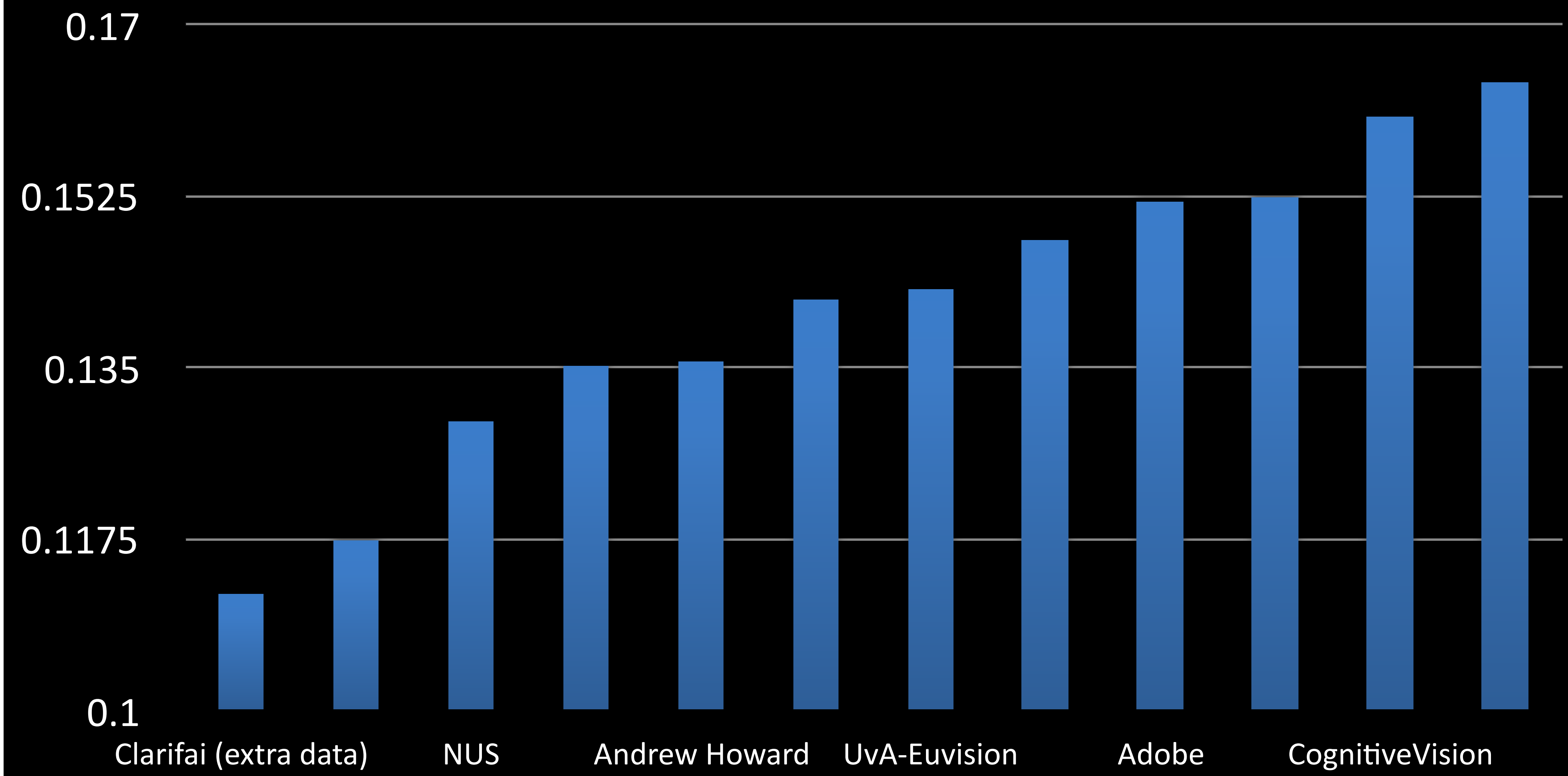

Layer 5: Top-9

Layer 5: Top-9 Patches



ImageNet Classification 2013 Results

- http://www.image-net.org/challenges/LSVRC/2013/results.php



- Pre-2012: 26.2% error → 2012: 16.5% error → 2013: 11.2% error

Sample Classification Results

[Krizhevsky et al. NIPS'12]