Computer Vision

CS-E4850, 5 study credits

Lecturer: Juho Kannala

Lecture 8: Deep convolutional neural networks & image classification

- **Deep convolutional neural networks** are trainable computational models with multiple processing layers and many parameters. They allow end-to-end learning of both image features and task-specific decisions (classifiers, regressors, etc.).

- **Image classification** is a task where a given image is assigned a class label that describes its content (e.g. “person”, “dog”, “car”, “building”).

Acknowledgement: many slides from Svetlana Lazebnik, Rob Fergus, and Andrej Karpathy

Further reading

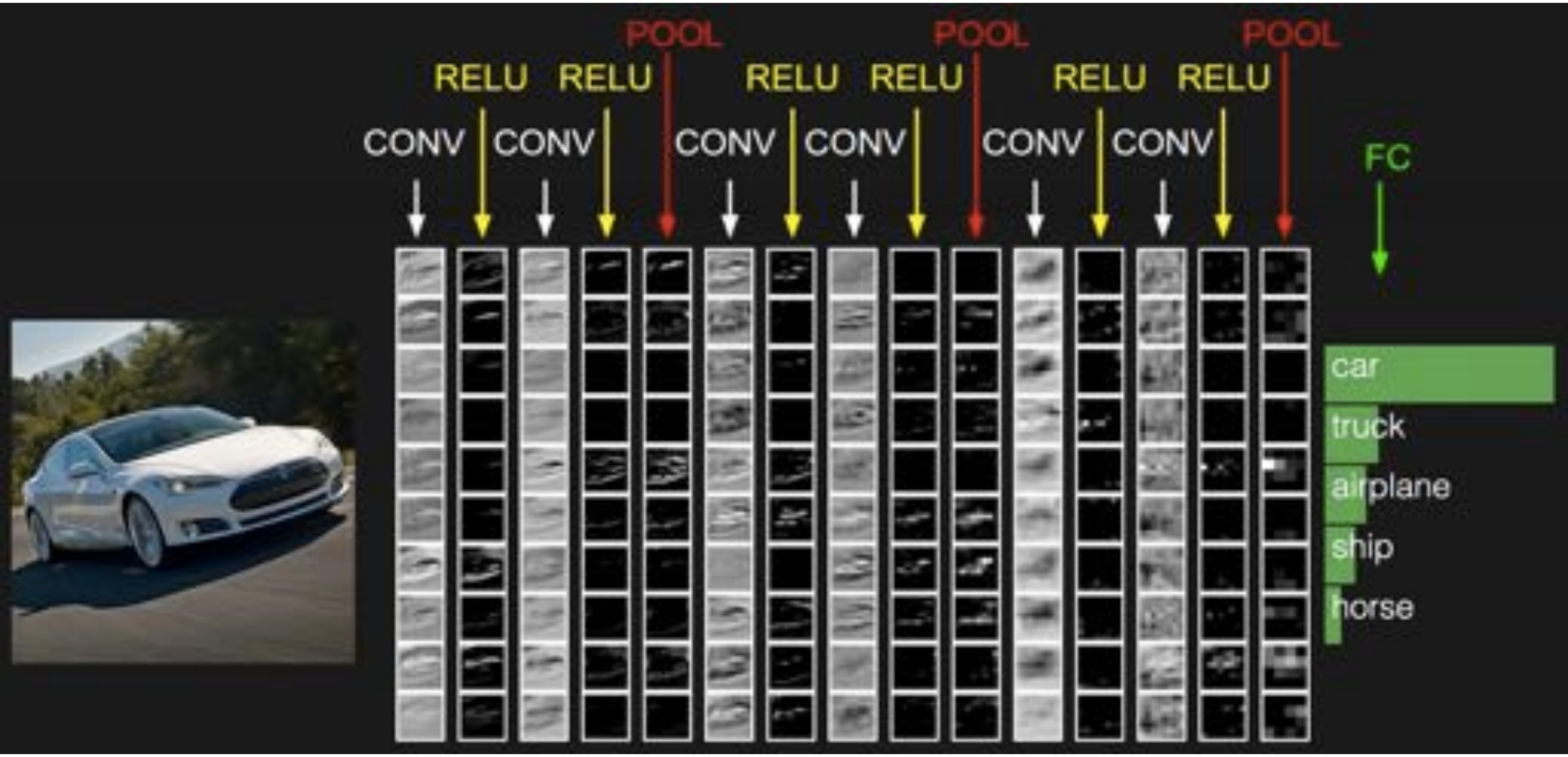
- LeCun, Bengio, Hinton: Deep learning. Nature 2015.

http://pages.cs.wisc.edu/~dyer/cs540/handouts/deep-learning-nature2015.pdf

- Goodfellow, Bengio, Courville:

http://www.deeplearningbook.org/

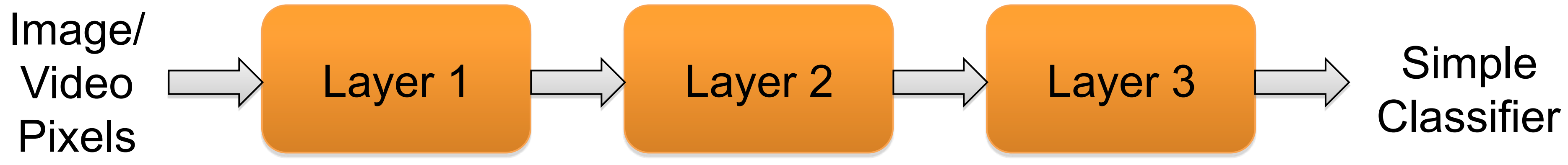
Deep Convolutional Neural Networks for Image Classification



Many slides from Rob Fergus, Andrej Karpathy

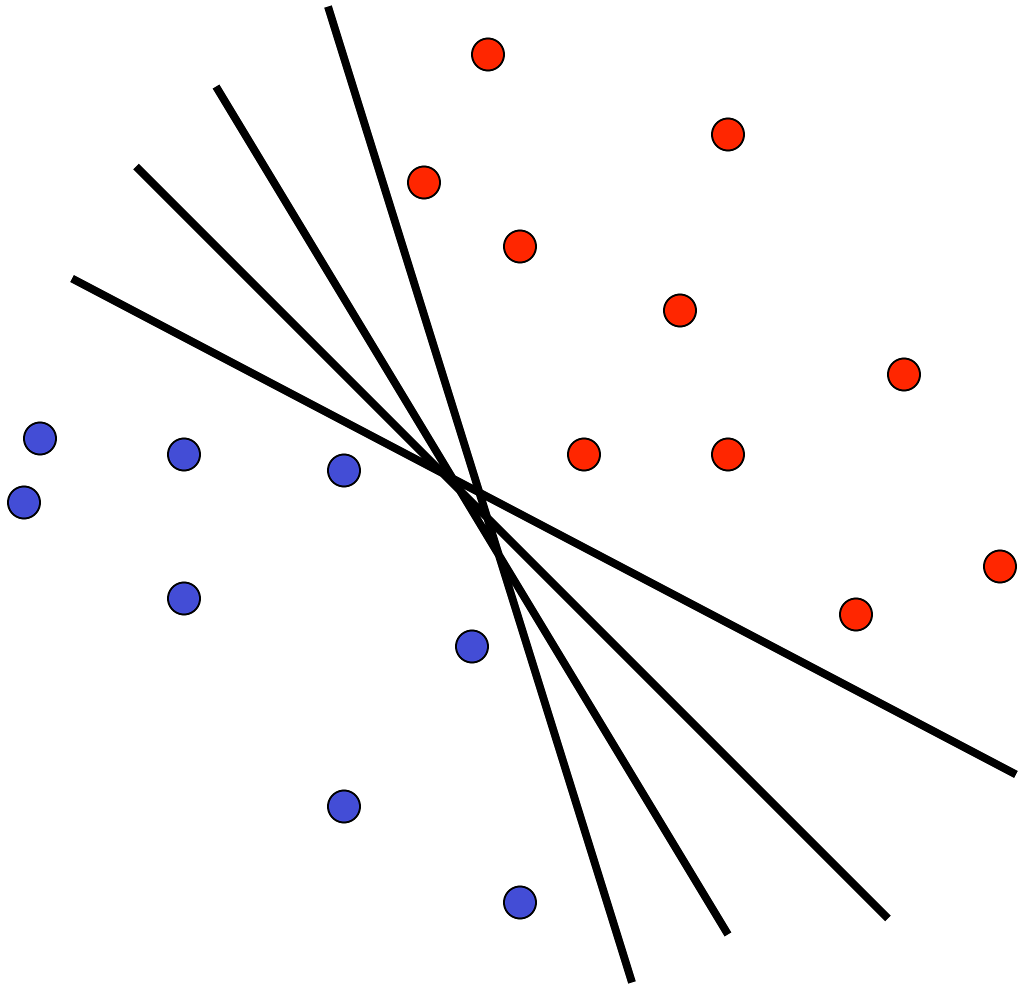
Deep learning

- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly



Background: Linear classifiers

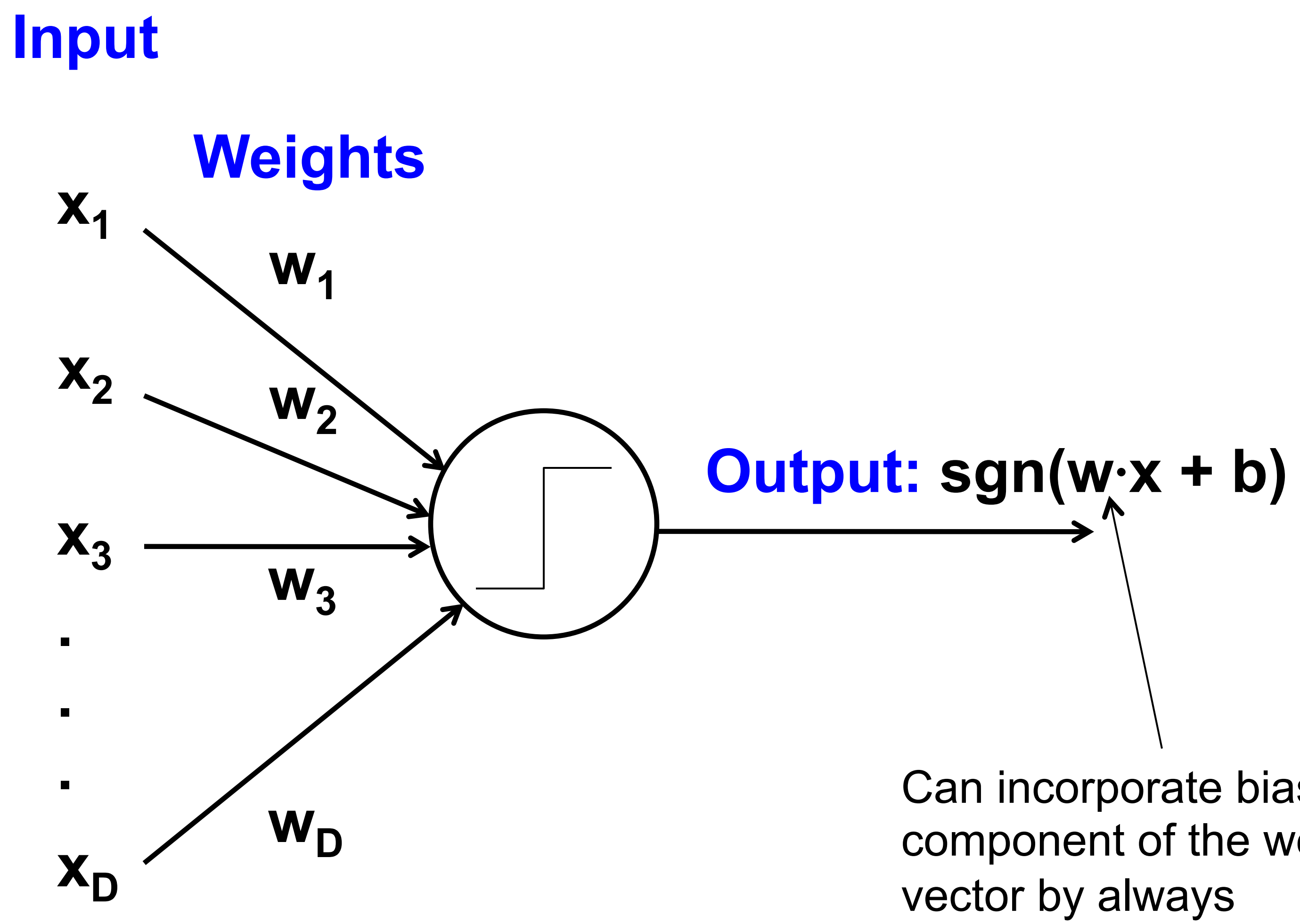
- When the data is linearly separable, there may be more than one separator (hyperplane)



Which separator is best?

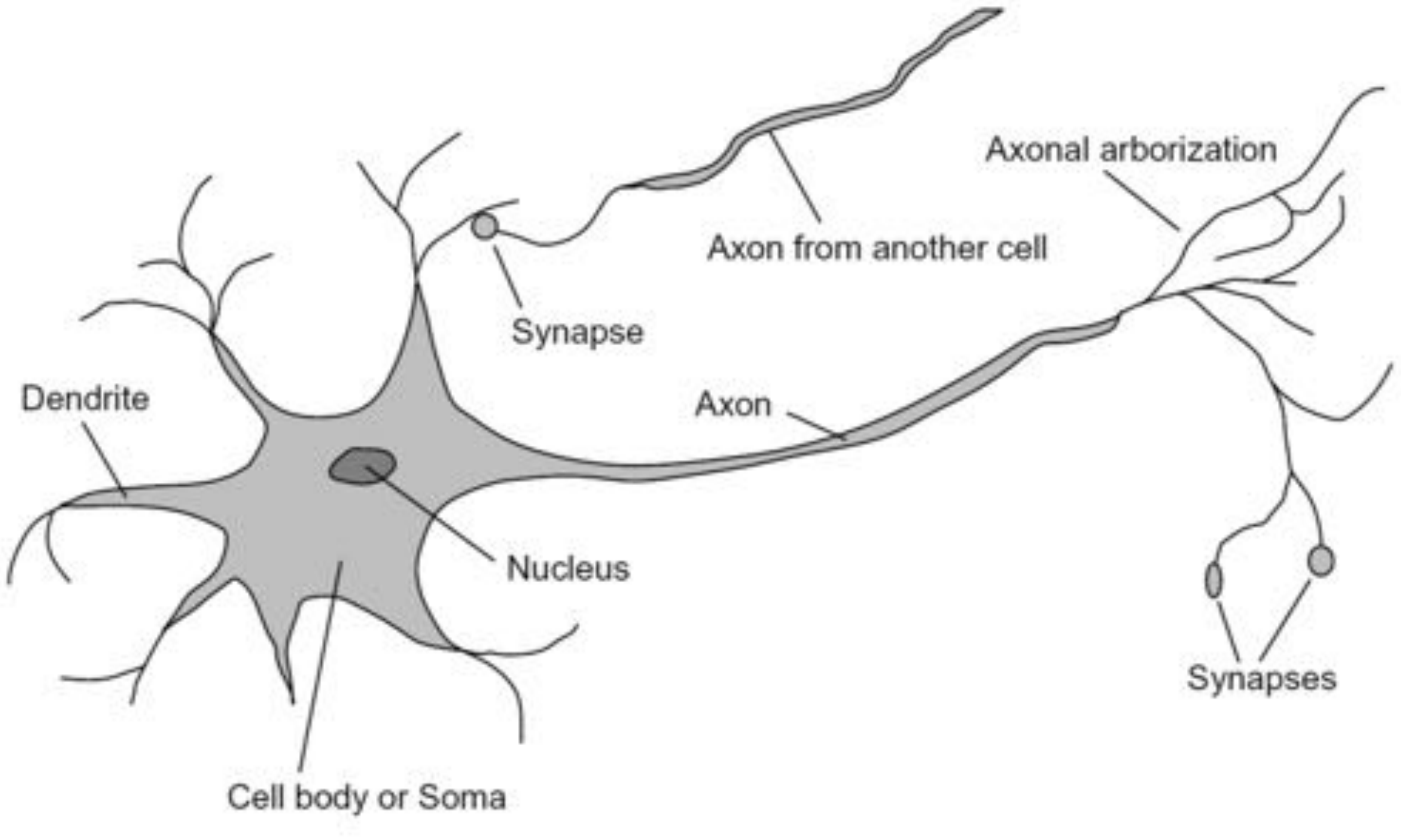
Perceptron

Perceptron

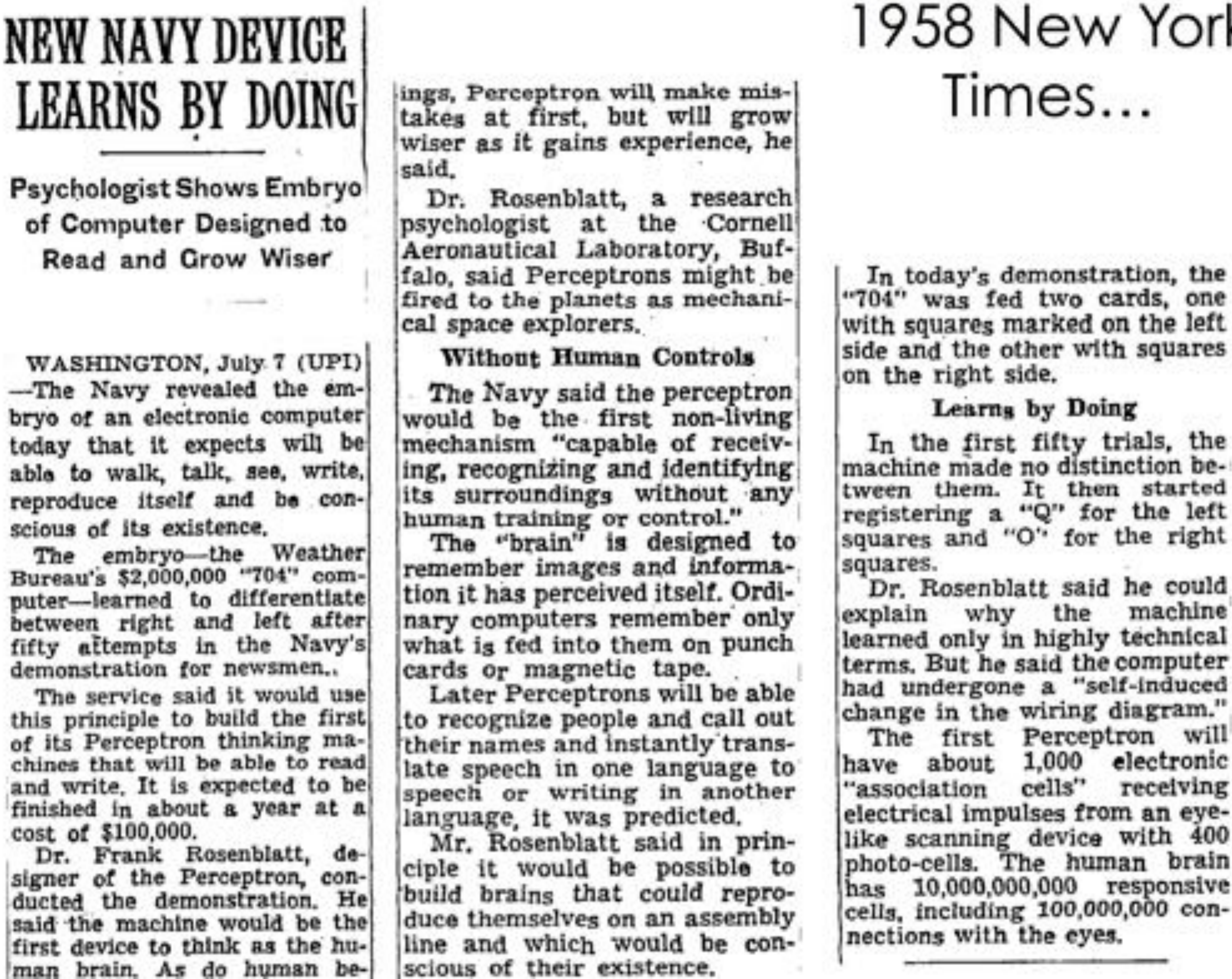


Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Loose inspiration: Human neurons



1958 New York Times...



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI)—The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen..

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Perceptron update rule

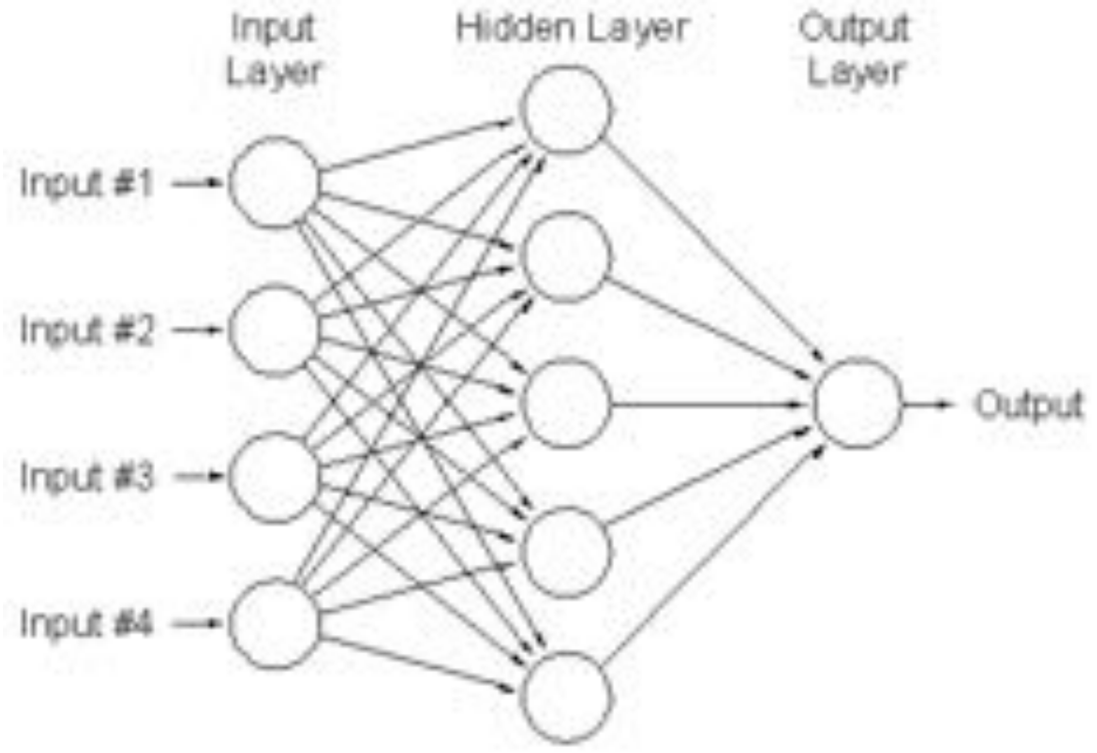
- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance **x** with label y:
  - Classify with current weights: $y' = \mathrm{sgn}(\mathbf{w} \cdot \mathbf{x})$
  - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y-y')\mathbf{x}$
  - α is a learning rate that should decay as 1/t (t is the epoch)
  - What happens if y' is correct?
  - Otherwise, consider what happens to individual weights $w_i \leftarrow w_i + \alpha(y-y')x_i$
    - If y = 1 and y' = -1, w_i will be increased if x_i is positive or decreased if x_i is negative $\rightarrow$ **w·x** will get bigger
    - If y = -1 and y' = 1, w_i will be decreased if x_i is positive or increased if x_i is negative $\rightarrow$ **w·x** will get smaller

Convergence of perceptron update rule

- **Linearly separable data:** converges to a perfect solution
- **Non-separable data:** converges to a minimum-error solution assuming learning rate decays as O(1/t) and examples are presented in random sequence

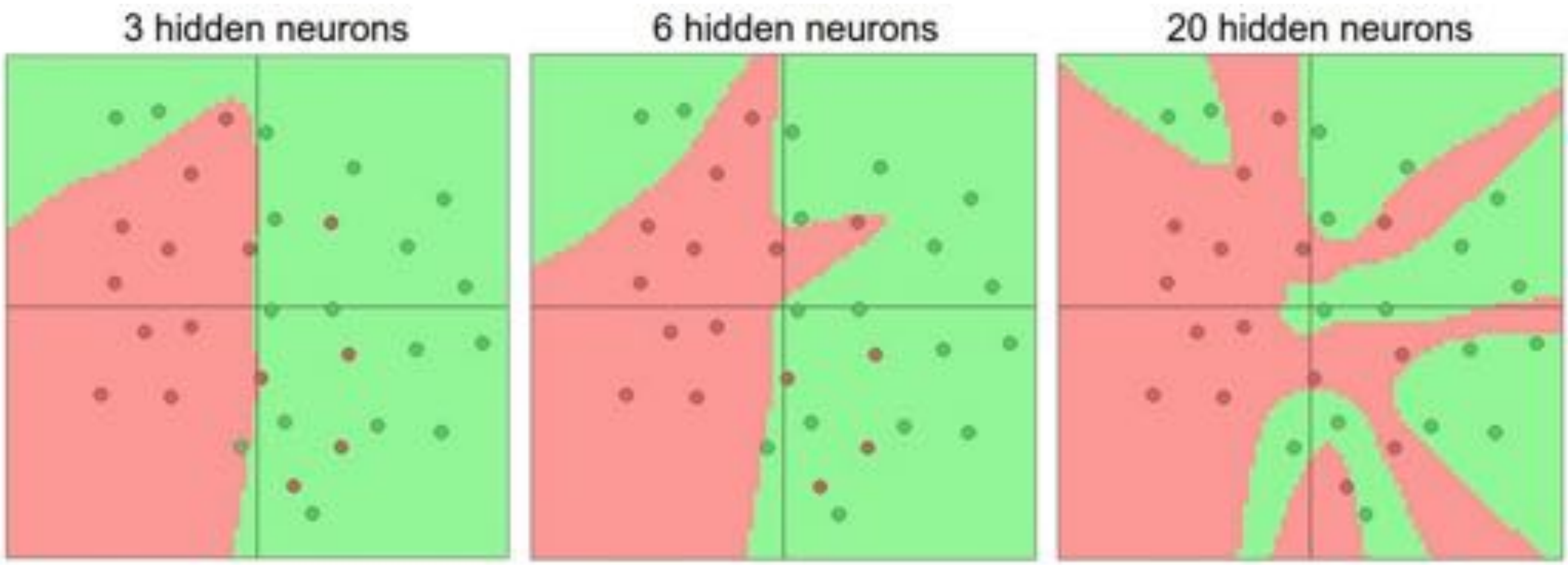
Multi-Layer Neural Networks

- Network with a hidden layer:



- Can represent nonlinear functions (provided each perceptron has a nonlinearity)

Multi-Layer Neural Networks



Source: http://cs231n.github.io/neural-networks-1/

Multi-Layer Neural Networks

- Beyond a single hidden layer:

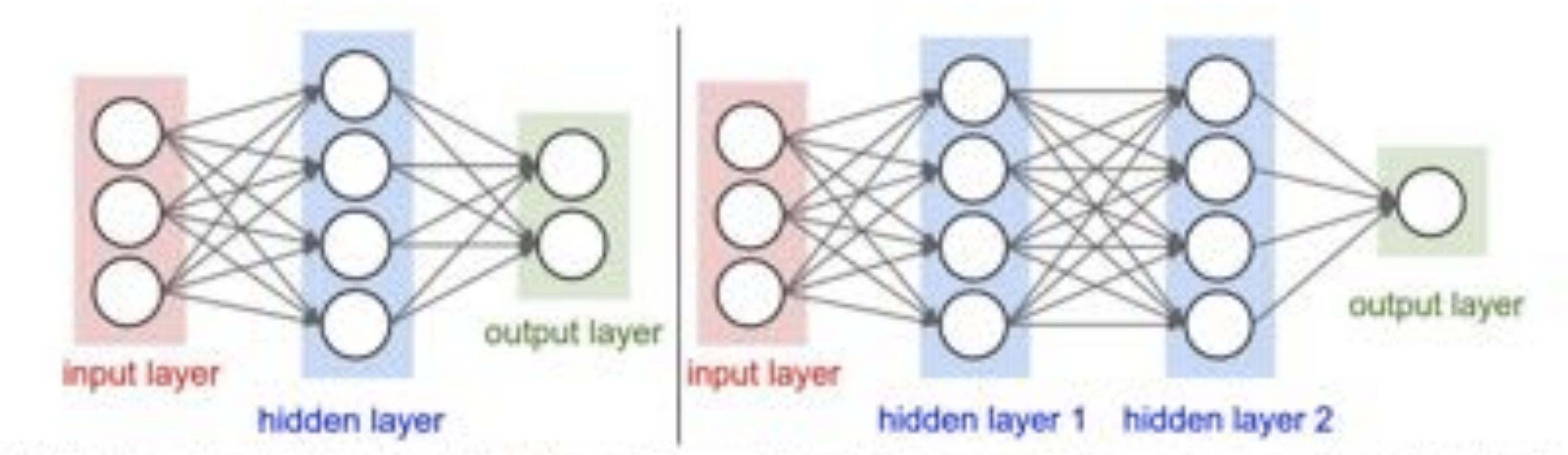


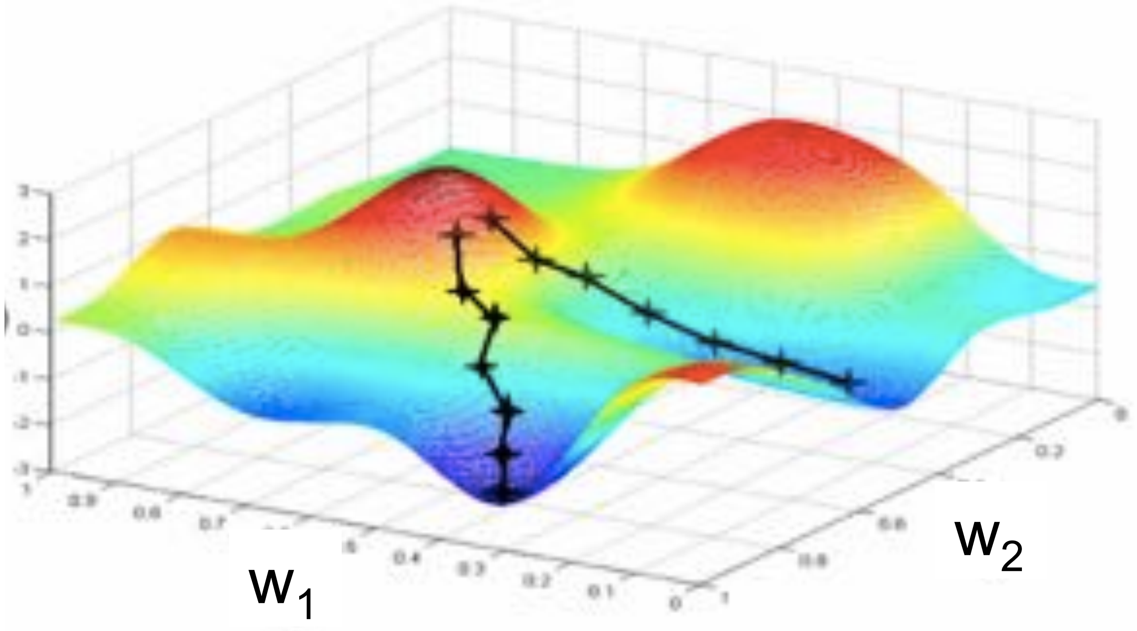
Figure source: http://cs231n.github.io/neural-networks-1/

Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$



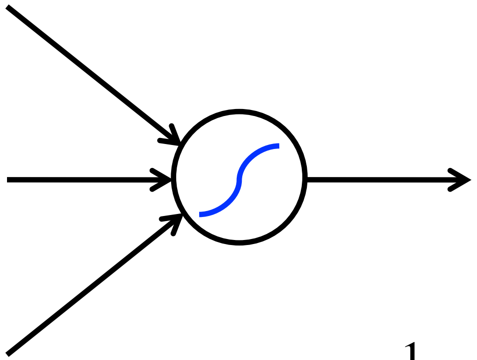
Training of multi-layer networks

- Find network weights to minimize the error between true and estimated labels of training examples:

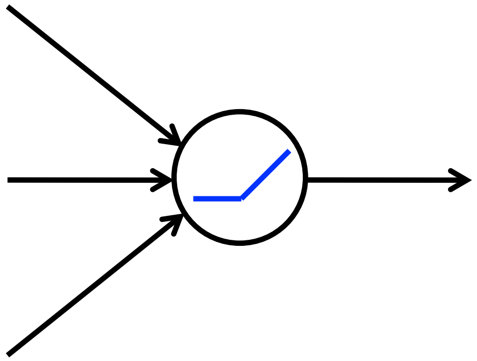
$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

- This requires perceptrons with a differentiable nonlinearity



Sigmoid: $g(t) = \frac{1}{1+e^{-t}}$



Rectified linear unit (ReLU): $g(t) = \max(0,t)$

Training of multi-layer networks

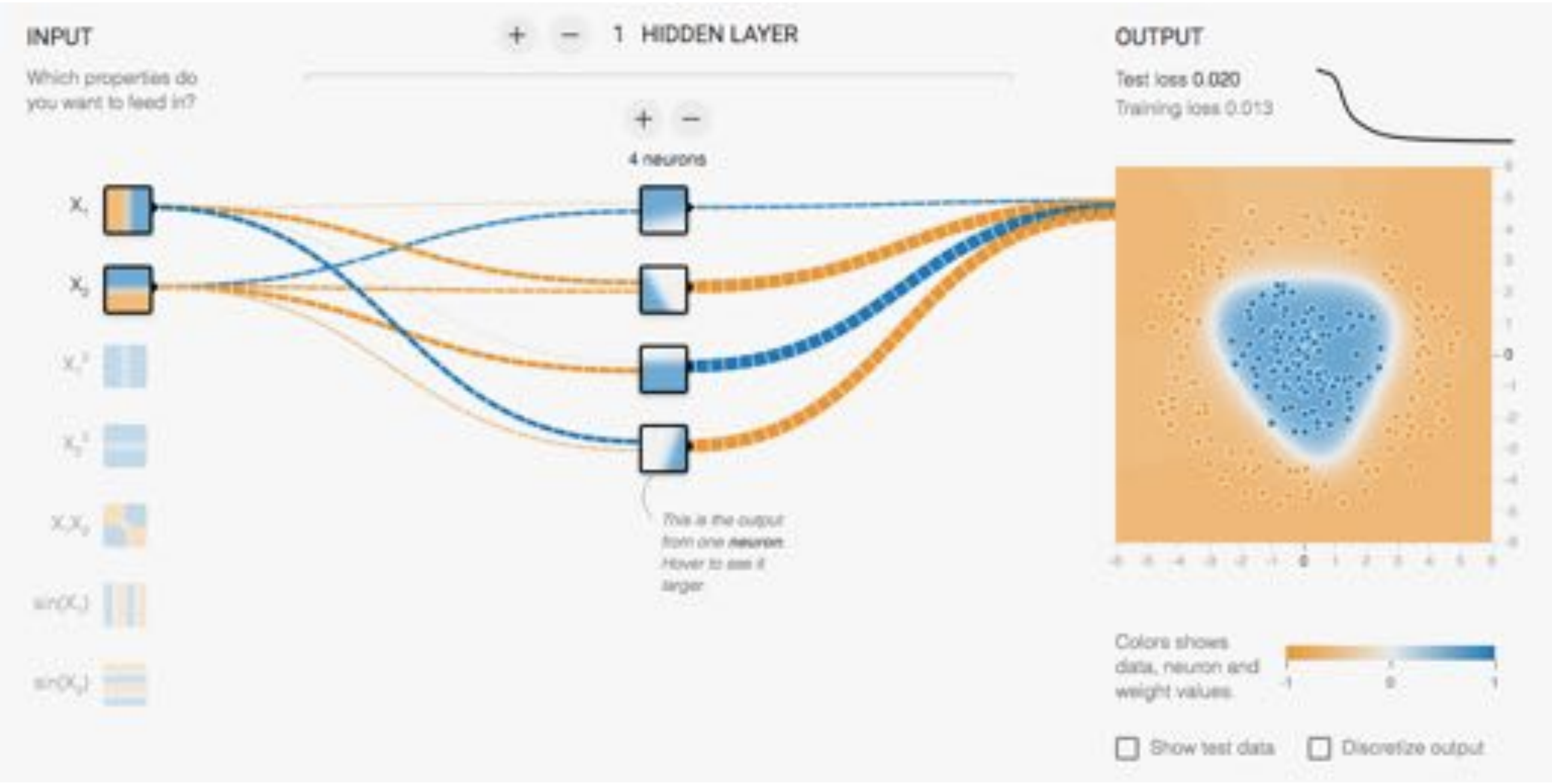
- Find network weights to minimize the error between true and estimated labels of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \dfrac{\partial E}{\partial \mathbf{w}}$

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule
- **Stochastic gradient descent:** compute the weight update w.r.t. one training example (or a small batch of examples) at a time, cycle through training examples in random order in multiple epochs

Multi-Layer Network Demo



Which properties do you want to feed in?

This is the output from one neuron. Hover to see it larger.

Test loss 0.020
Training loss 0.013

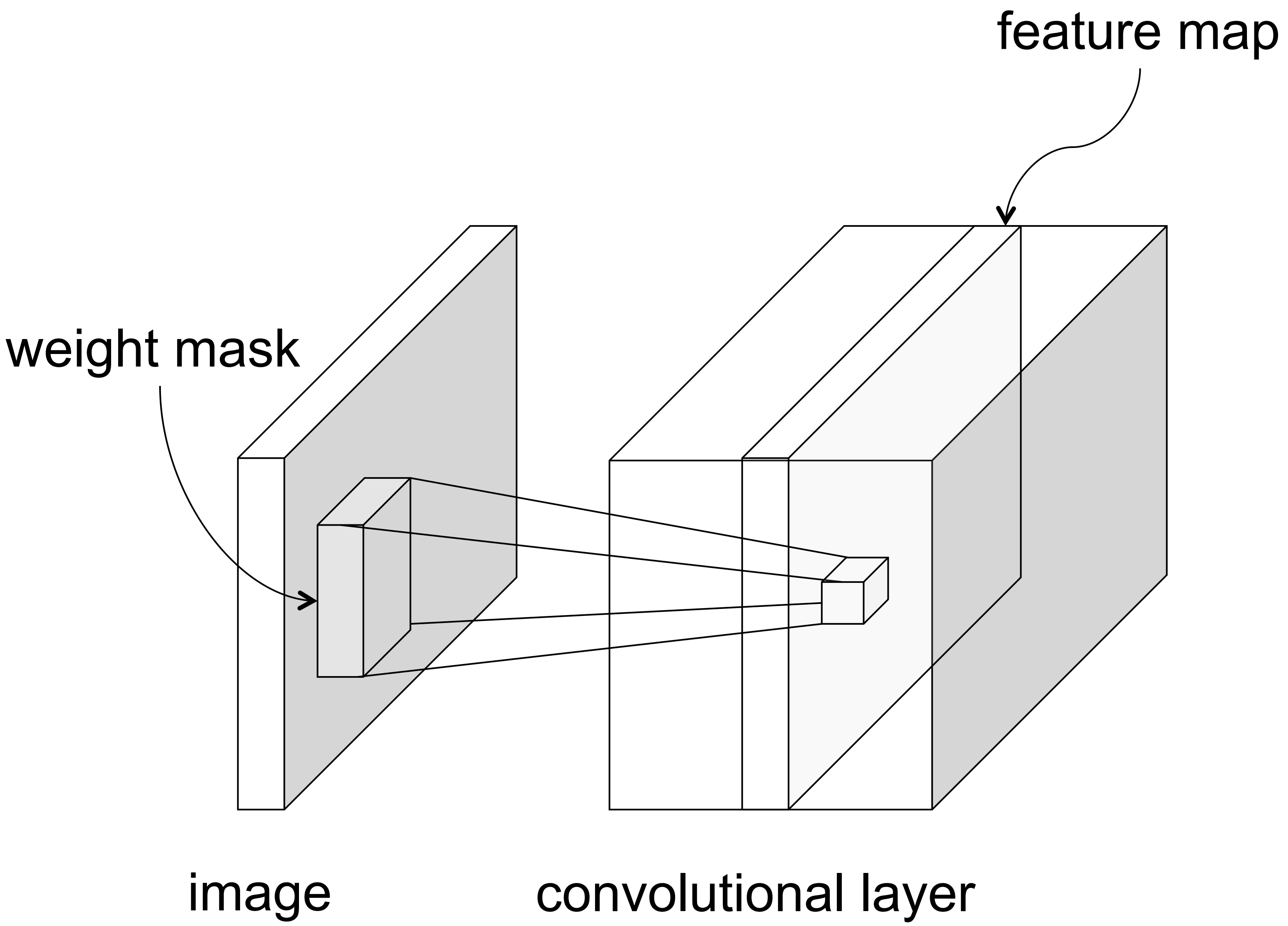
Colors shows data, neuron and weight values.

http://playground.tensorflow.org/

Neural networks: Pros and cons

- Pros
  - Flexible and general function approximation framework
  - Can build extremely powerful models by adding more layers
- Cons
  - Hard to analyze theoretically (e.g., training is prone to local optima)
  - Huge amount of training data, computing power may be required to get good performance
  - The space of implementation choices is huge (network architectures, parameters)

Neural networks for images



Neural networks for images

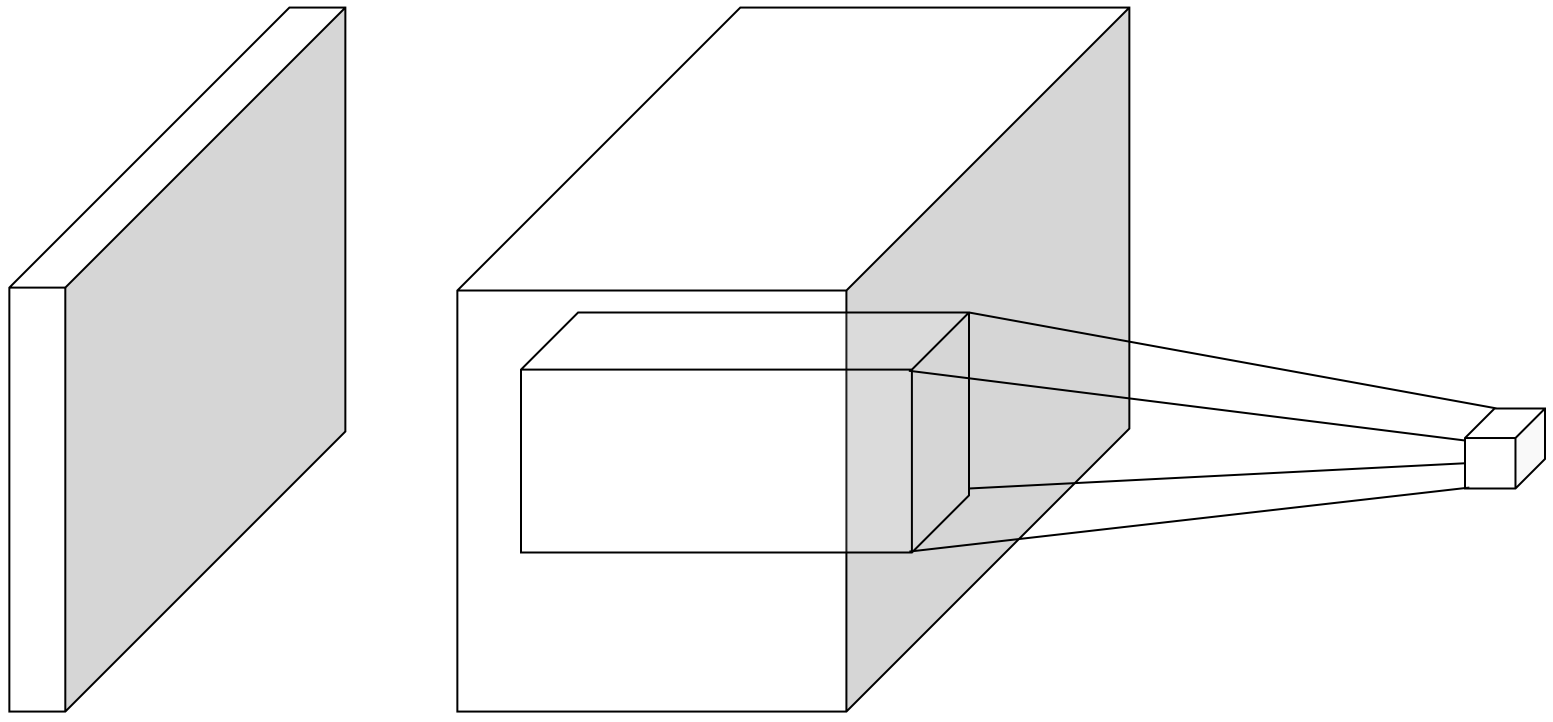
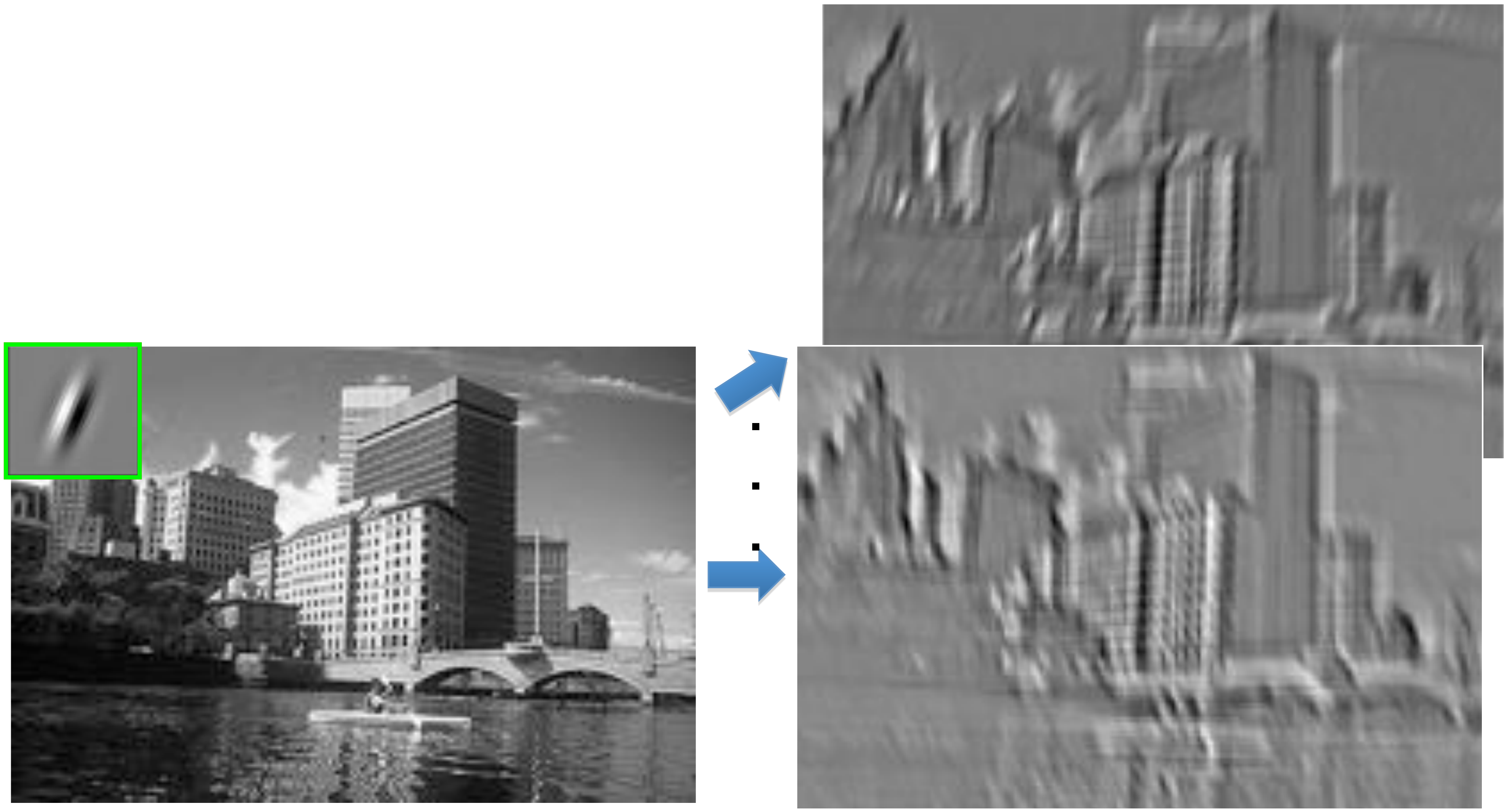


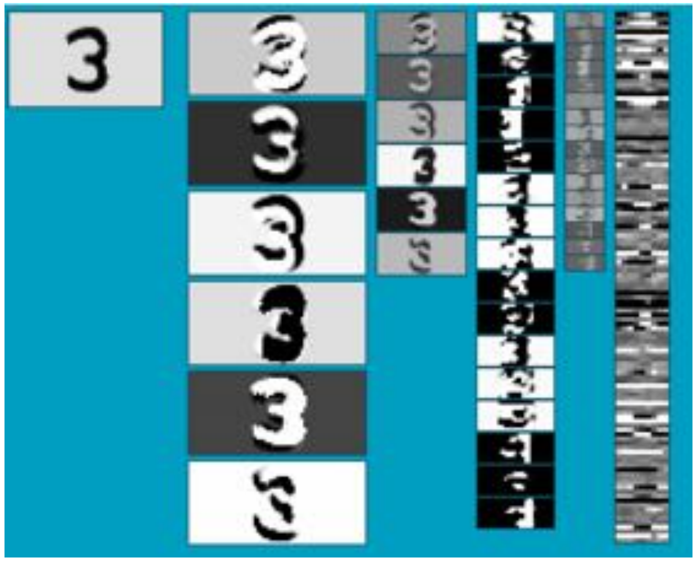
image convolutional layer

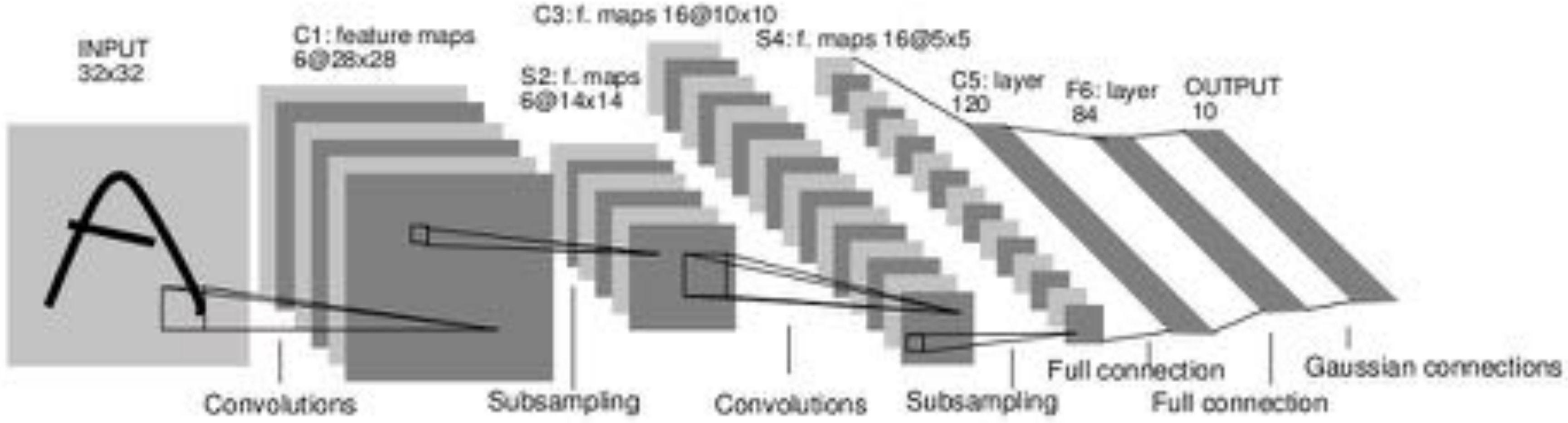
Convolution as feature extraction



Convolutional Neural Networks

- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end

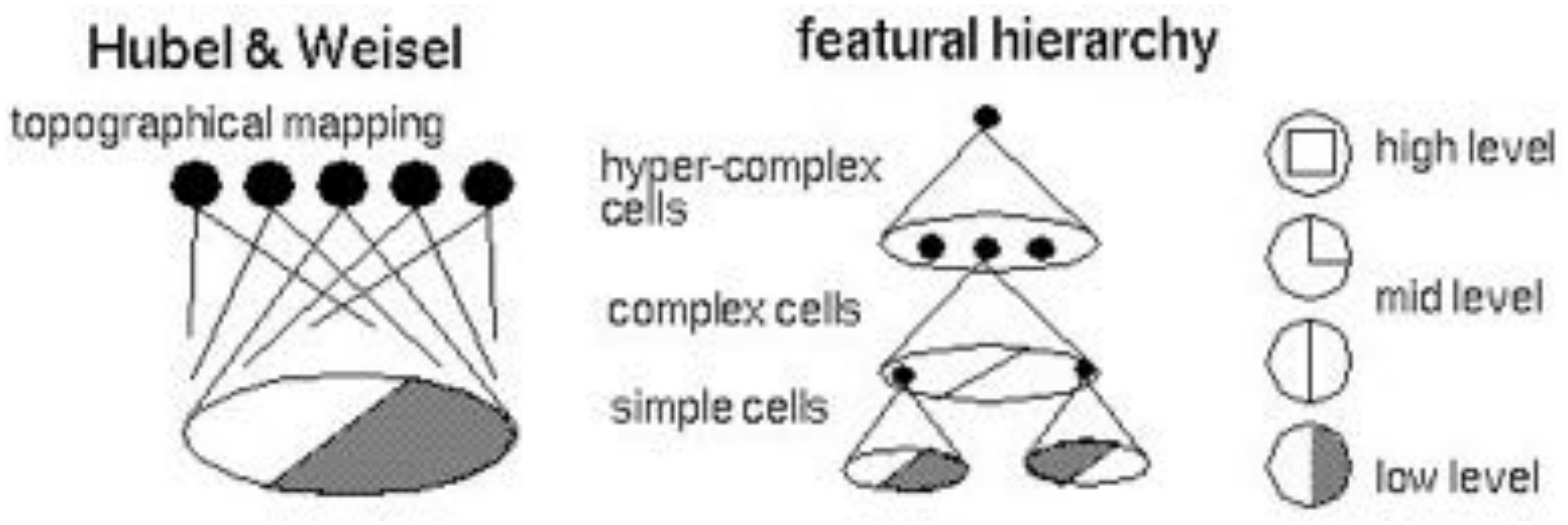




Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

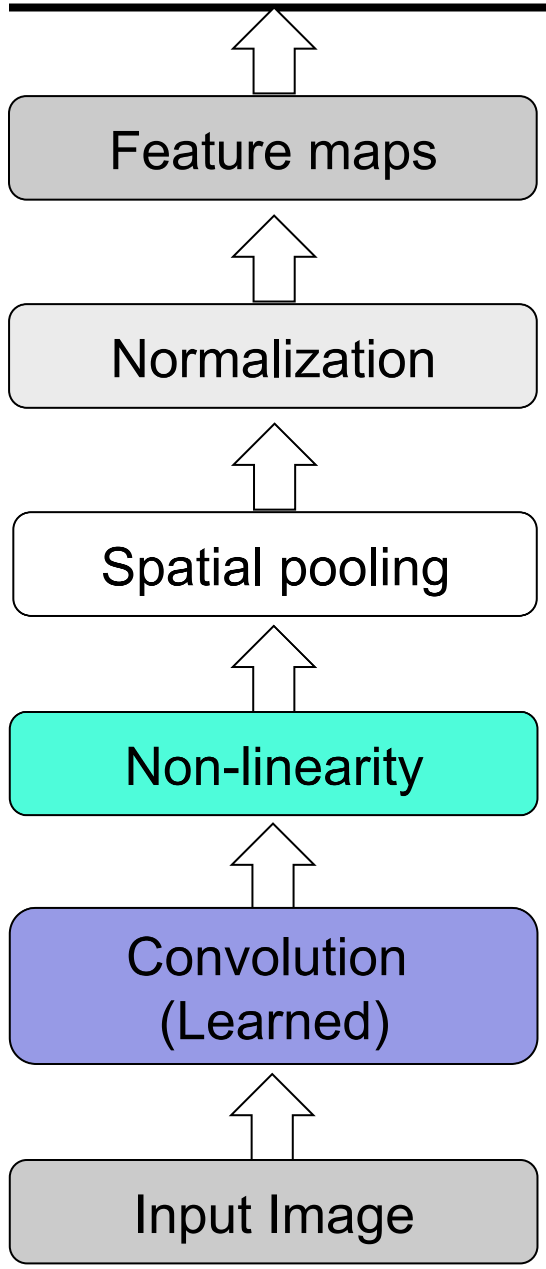
Biological inspiration

- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)
  - Visual cortex consists of a hierarchy of *simple*, *complex*, and *hyper-complex* cells

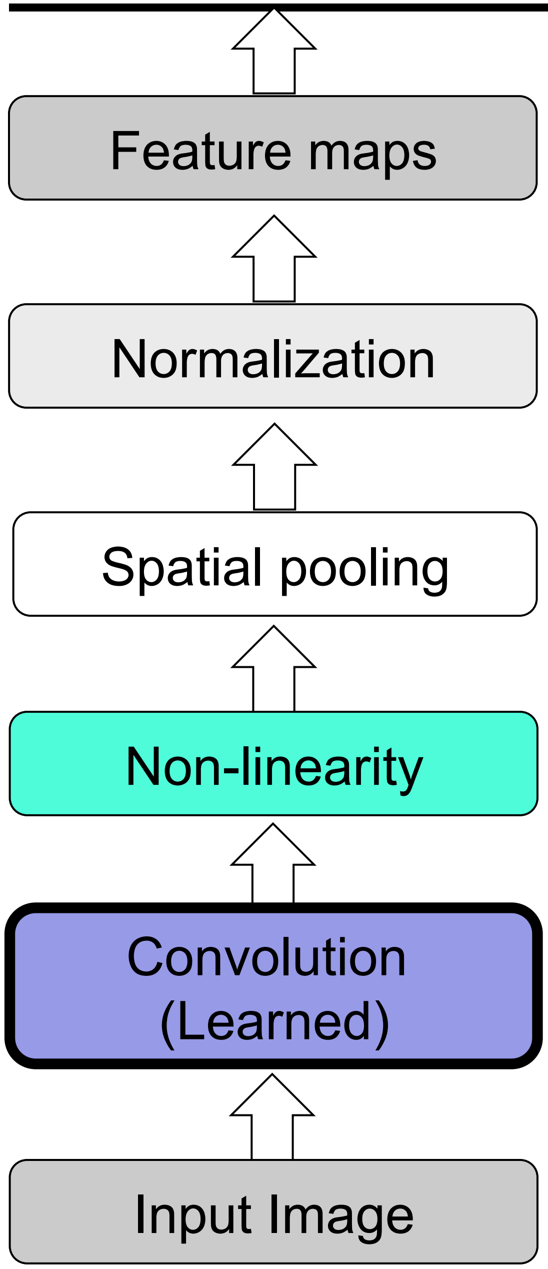


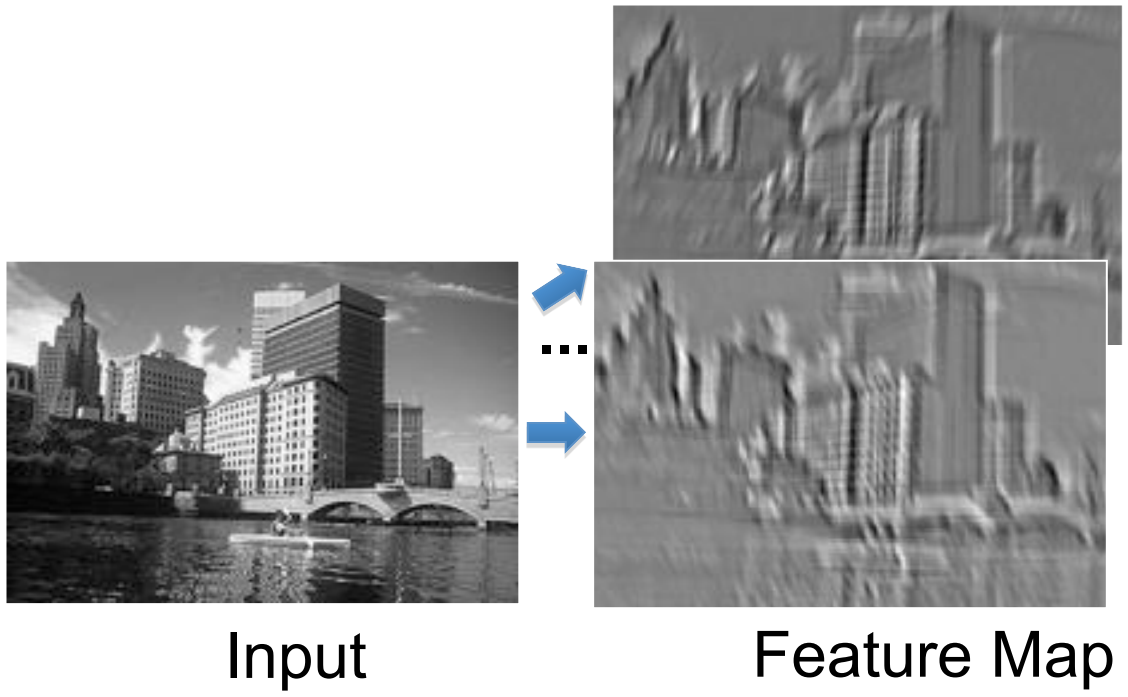
[Source]

Convolutional Neural Networks

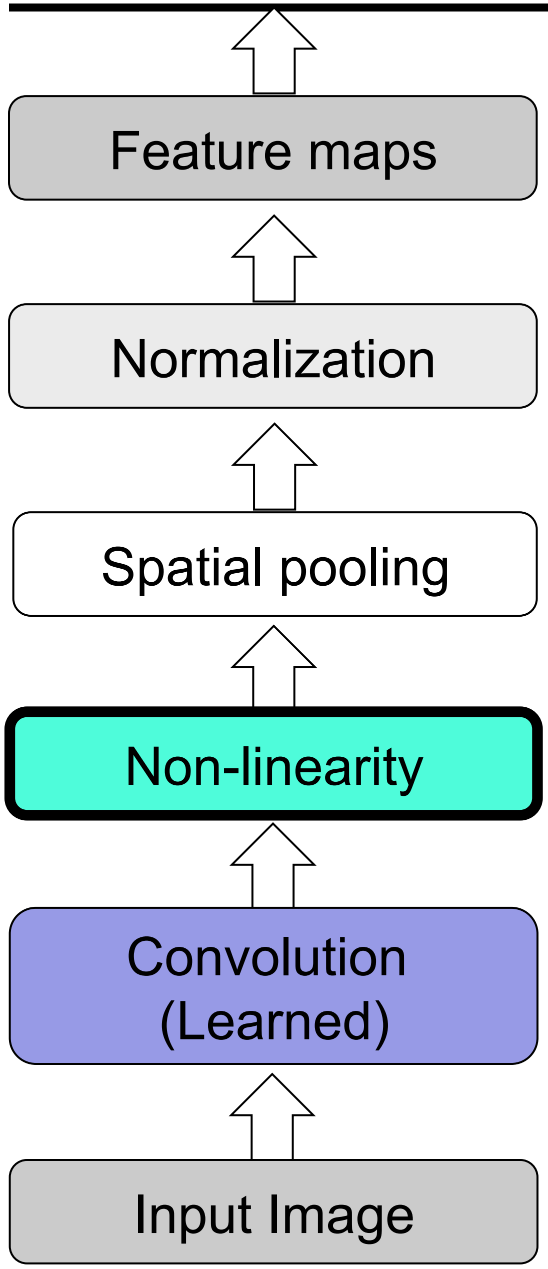


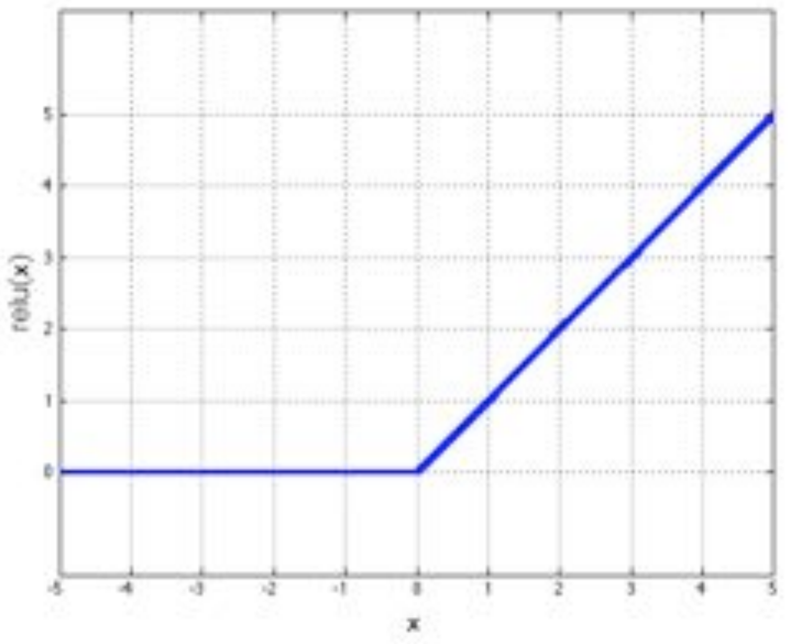
Convolutional Neural Networks



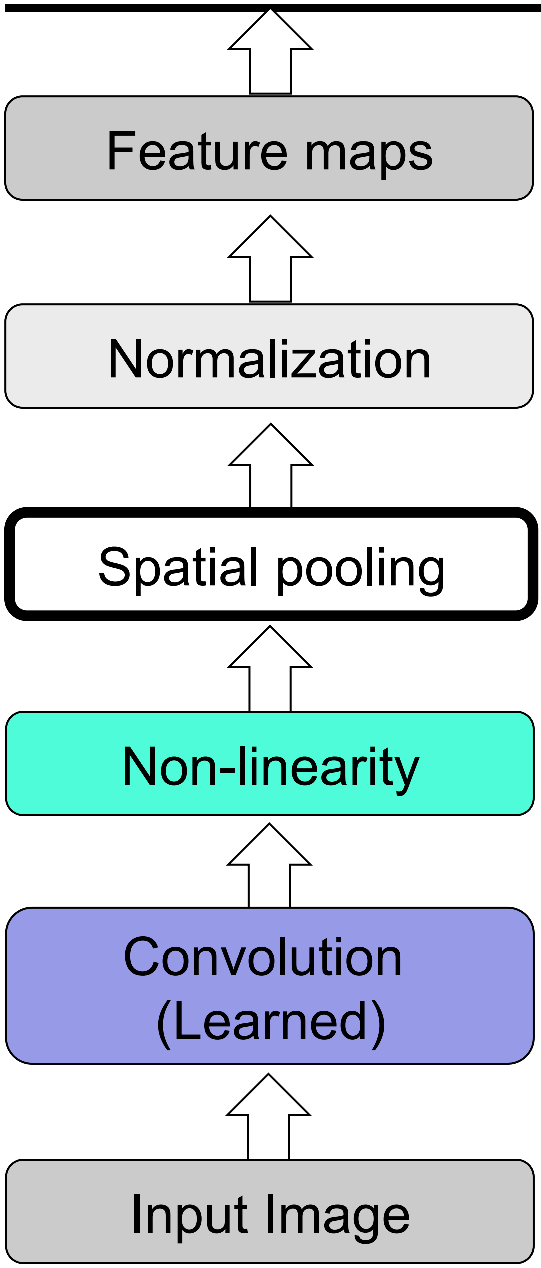


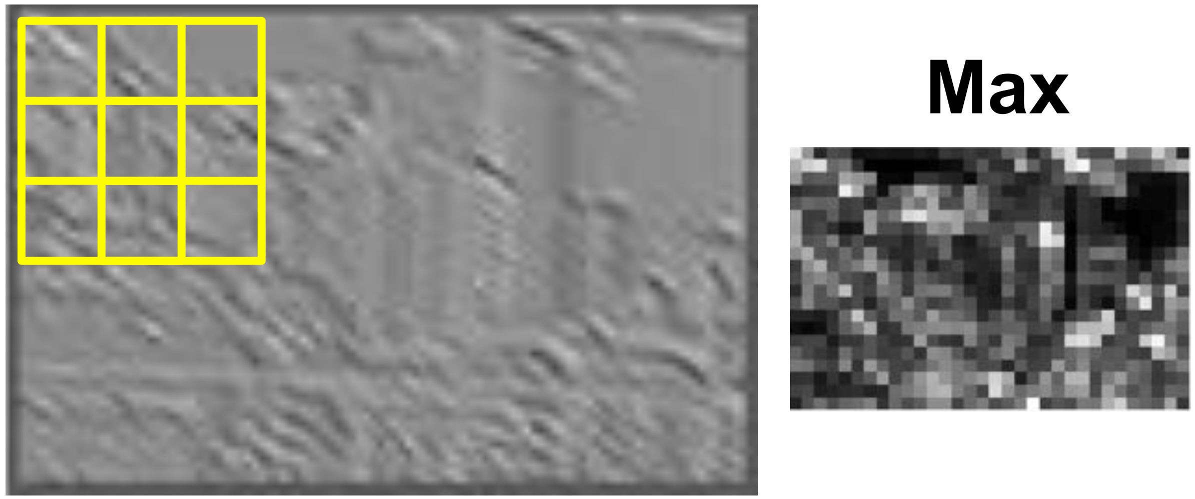
Convolutional Neural Networks



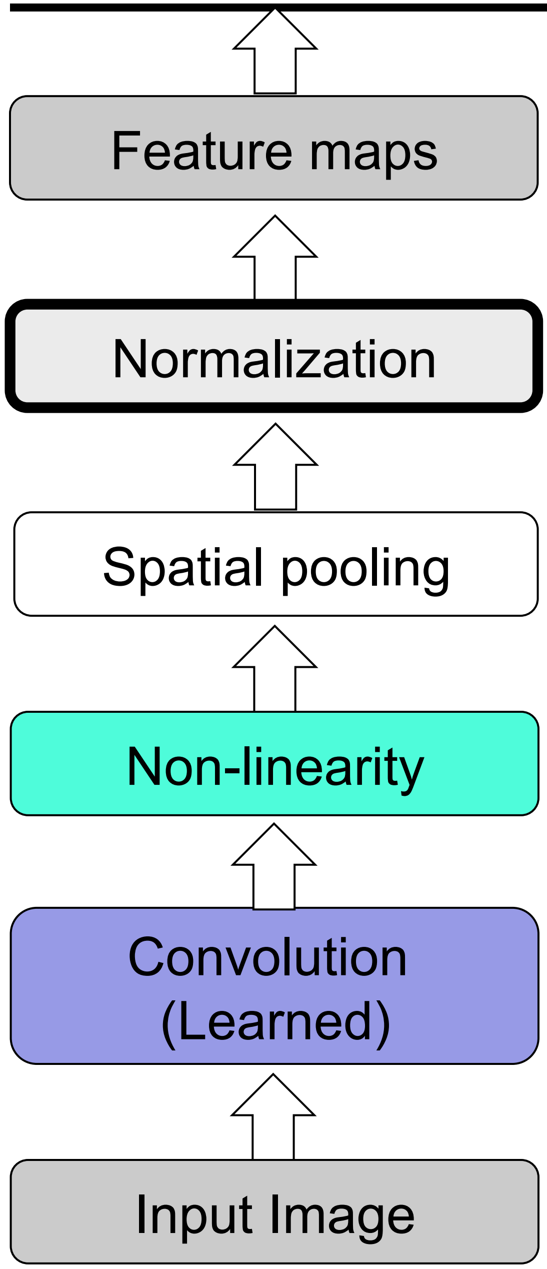


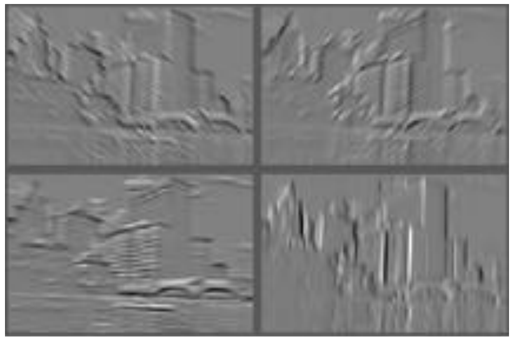
Convolutional Neural Networks

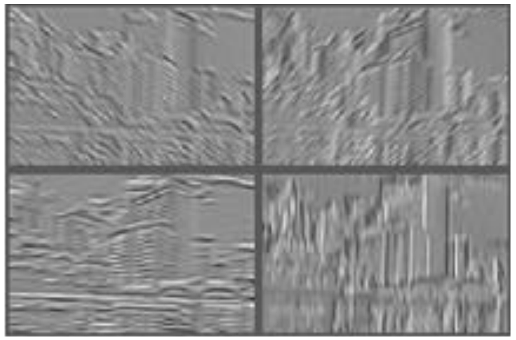




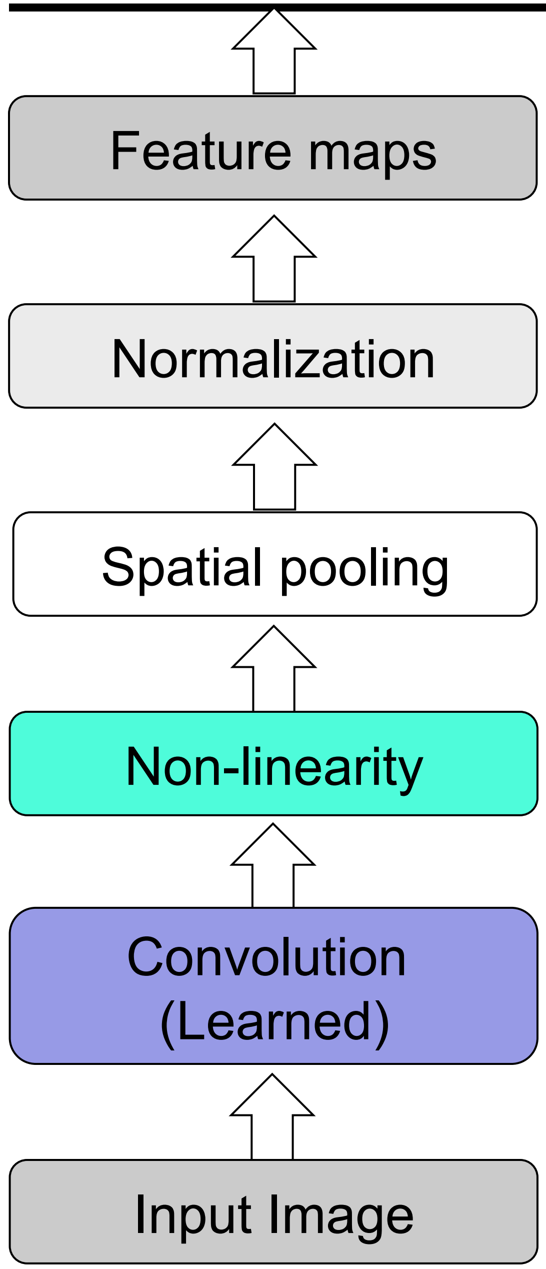
Convolutional Neural Networks




Feature Maps

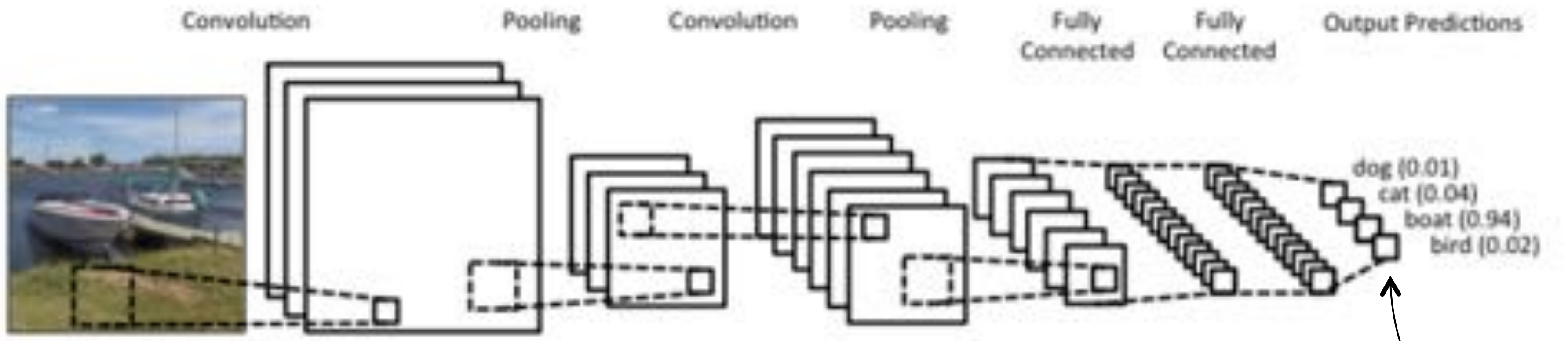

Feature Maps After Contrast Normalization

Convolutional Neural Networks



Convolutional filters are trained in a supervised manner by back-propagating classification error

Simplified architecture



Softmax layer:

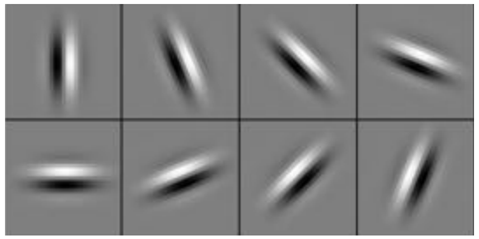
$$P(c \mid \mathbf{x}) = \frac{\exp(\mathbf{w}_c \cdot \mathbf{x})}{\sum_{k=1}^{C} \exp(\mathbf{w}_k \cdot \mathbf{x})}$$

Compare: SIFT Descriptor

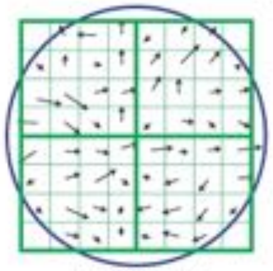
Lowe
[IJCV 2004]

Image Pixels

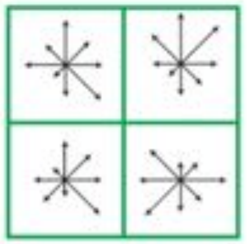
Apply oriented filters



Take max filter response (L-inf normalization)



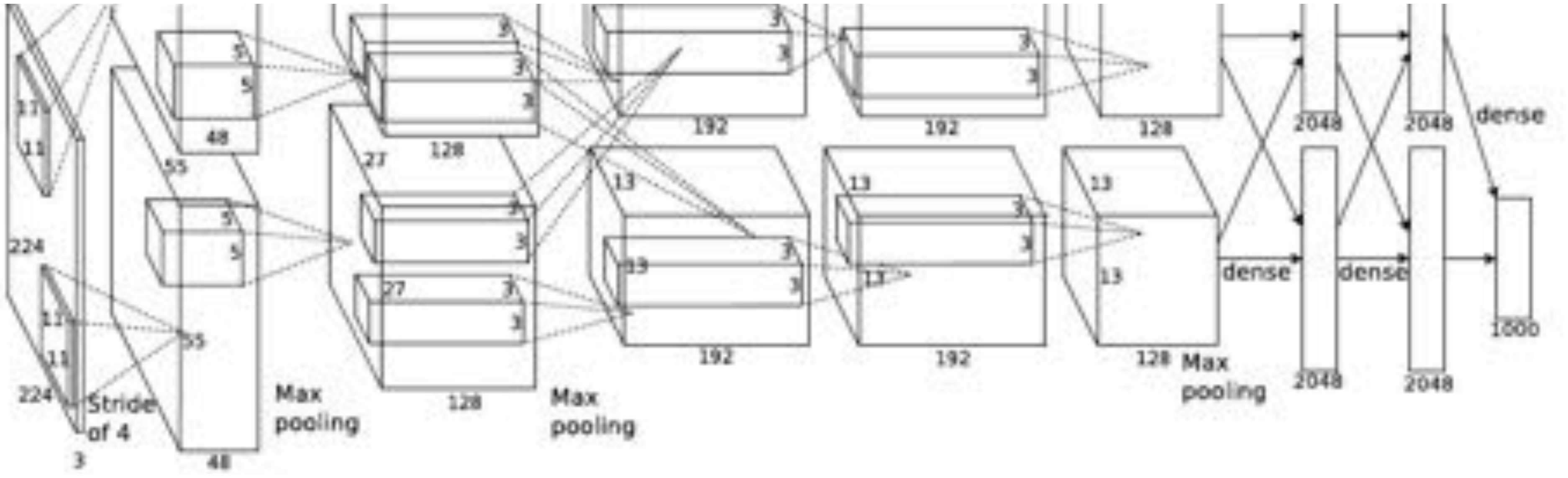
Spatial pool (Sum), L2 normalization



Feature Vector

AlexNet

- Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data (10^6 vs. 10^3 images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week



A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

Refinement of AlexNet architecture

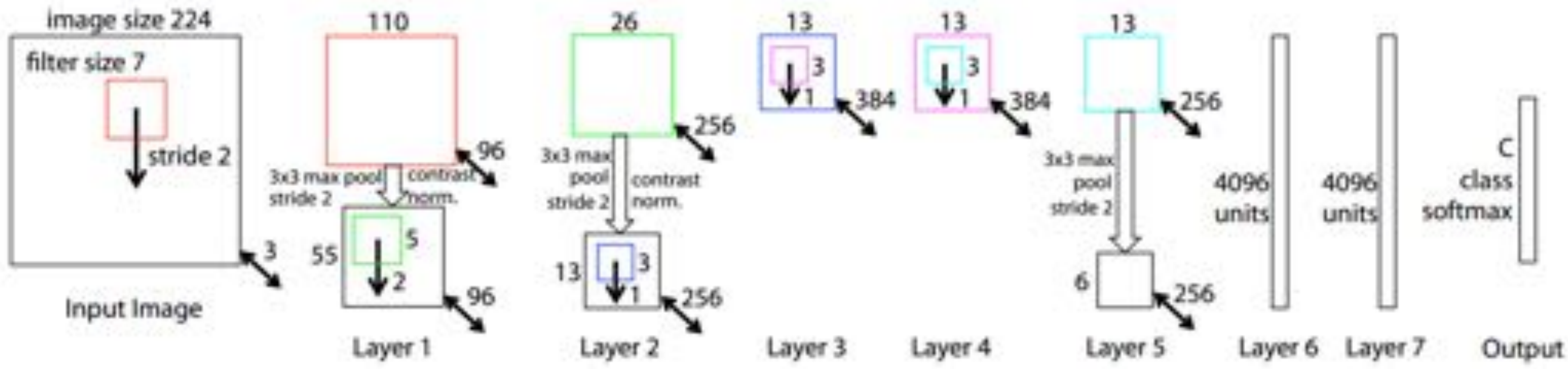
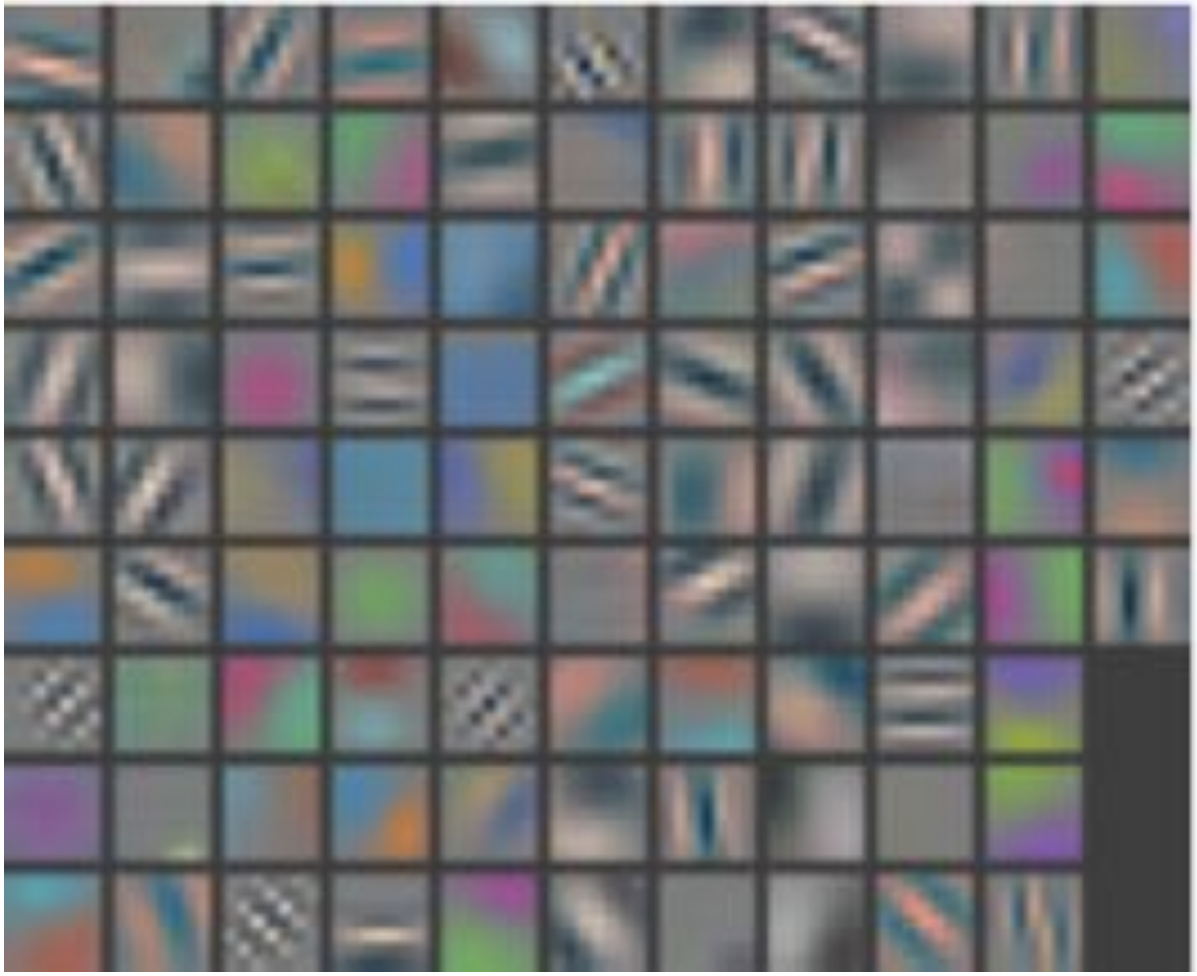


Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through 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.

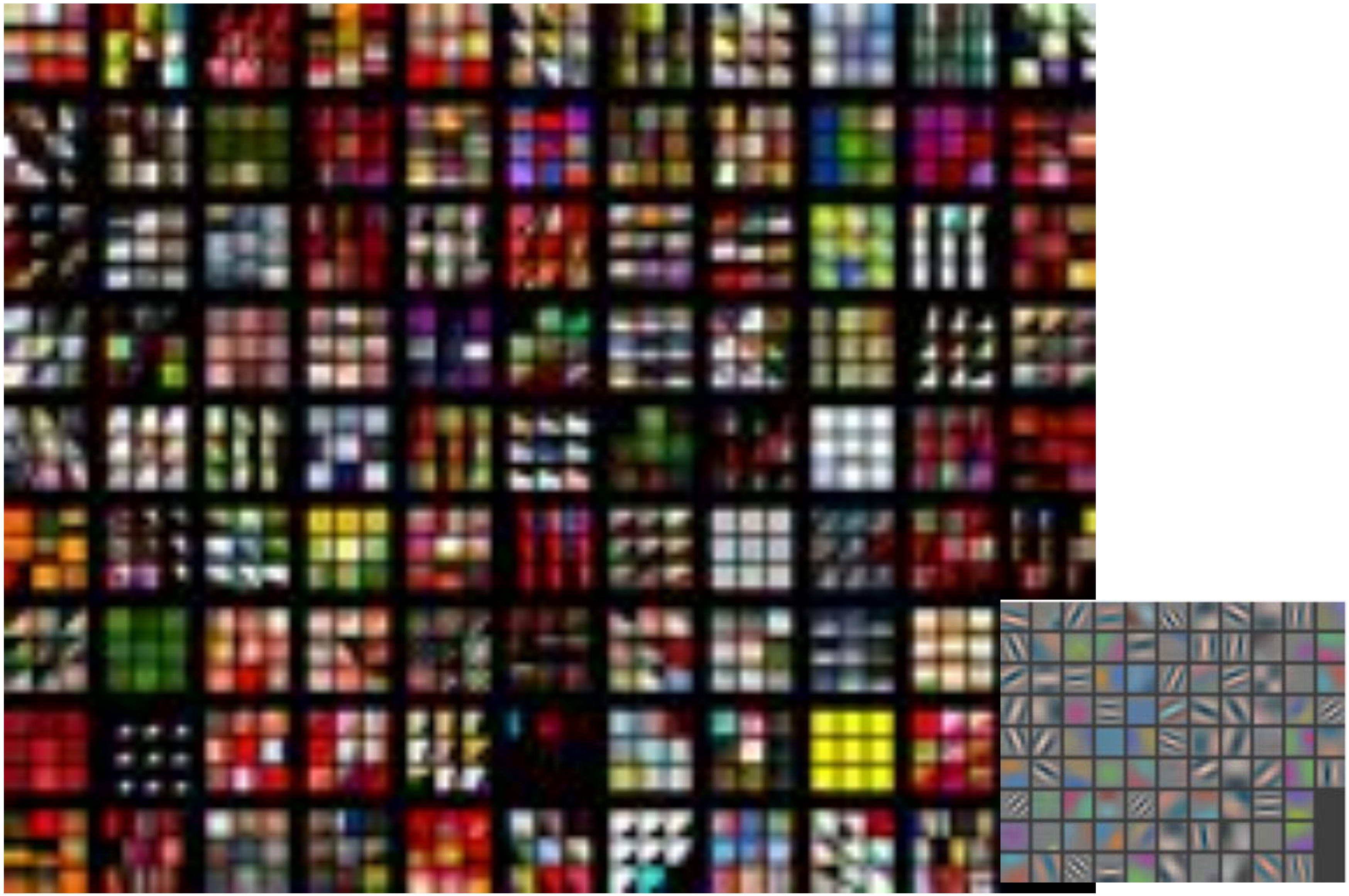
M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, arXiv preprint, 2013

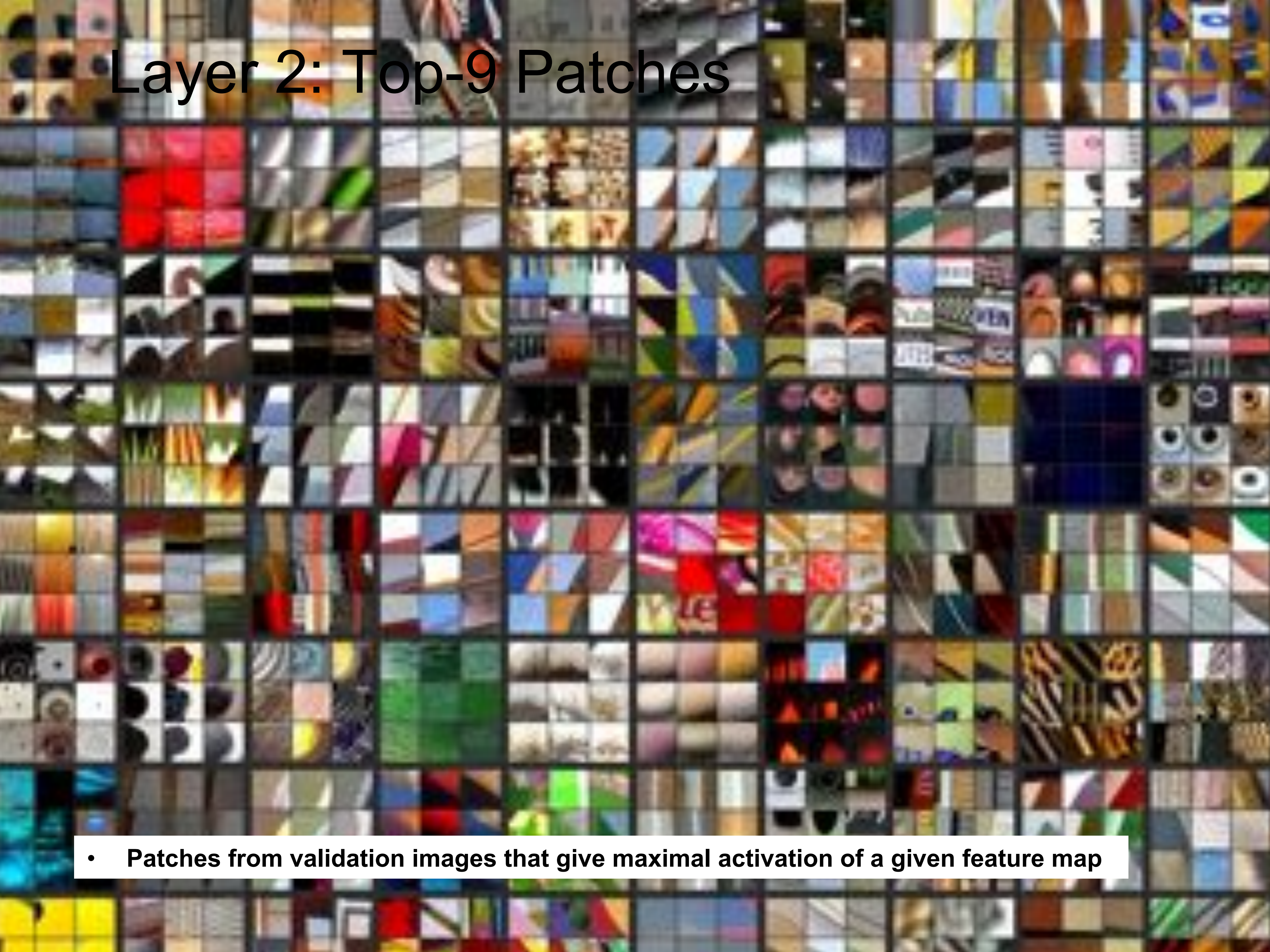
Layer 1 Filters



M. Zeiler and R. Fergus, [Visualizing and Understanding Convolutional Networks](), ECCV 2014 (Best Paper Award winner)

Layer 1: Top-9 Patches

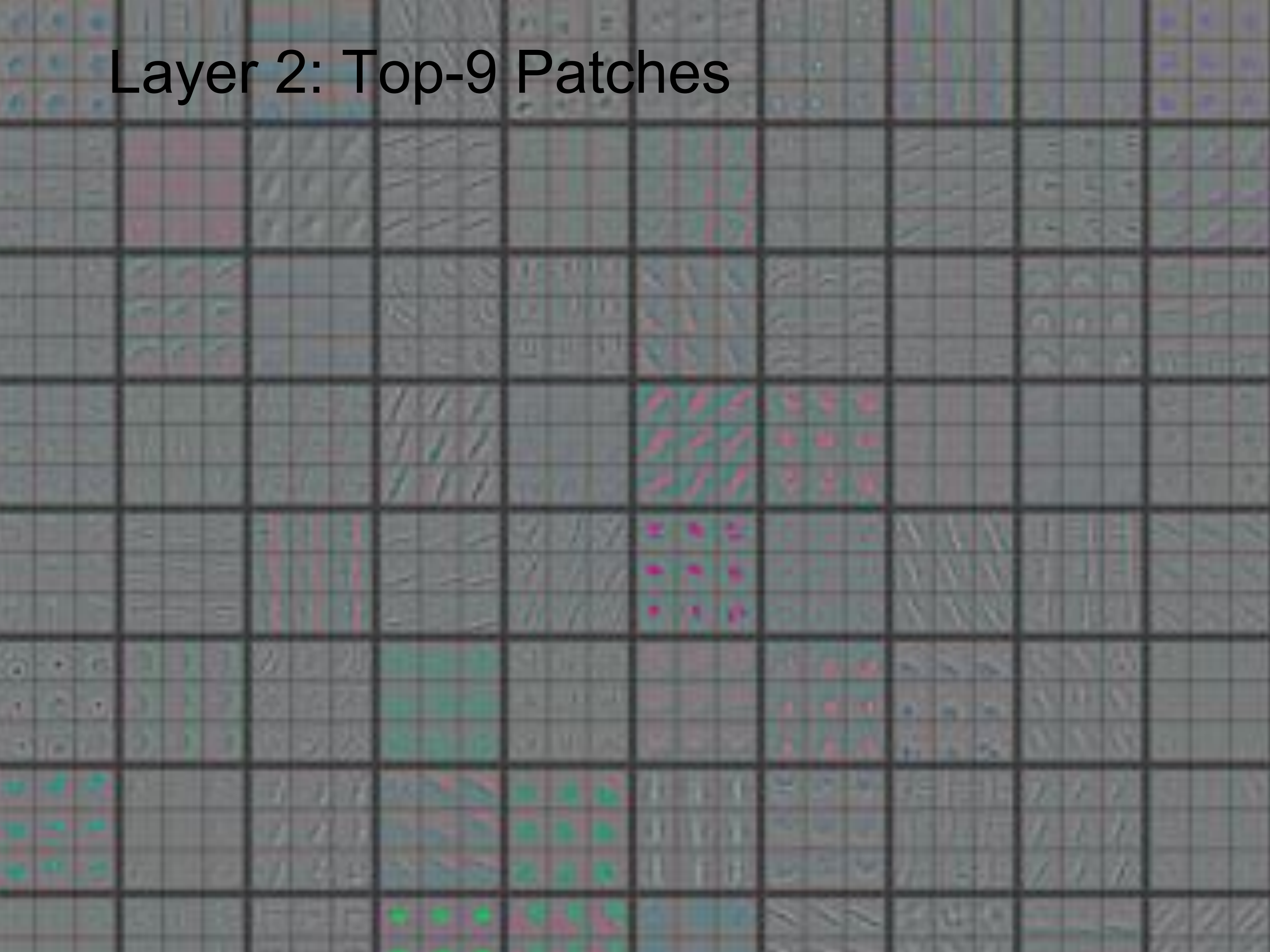




Layer 2: Top-9 Patches

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

Layer 2: Top-9 Patches



Layer 5: Top-9 Patches

Layer 5: Top-9 Patches

ImageNet Challenge

IMAGENET

- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- Challenge: 1.2 million training images, 1000 classes

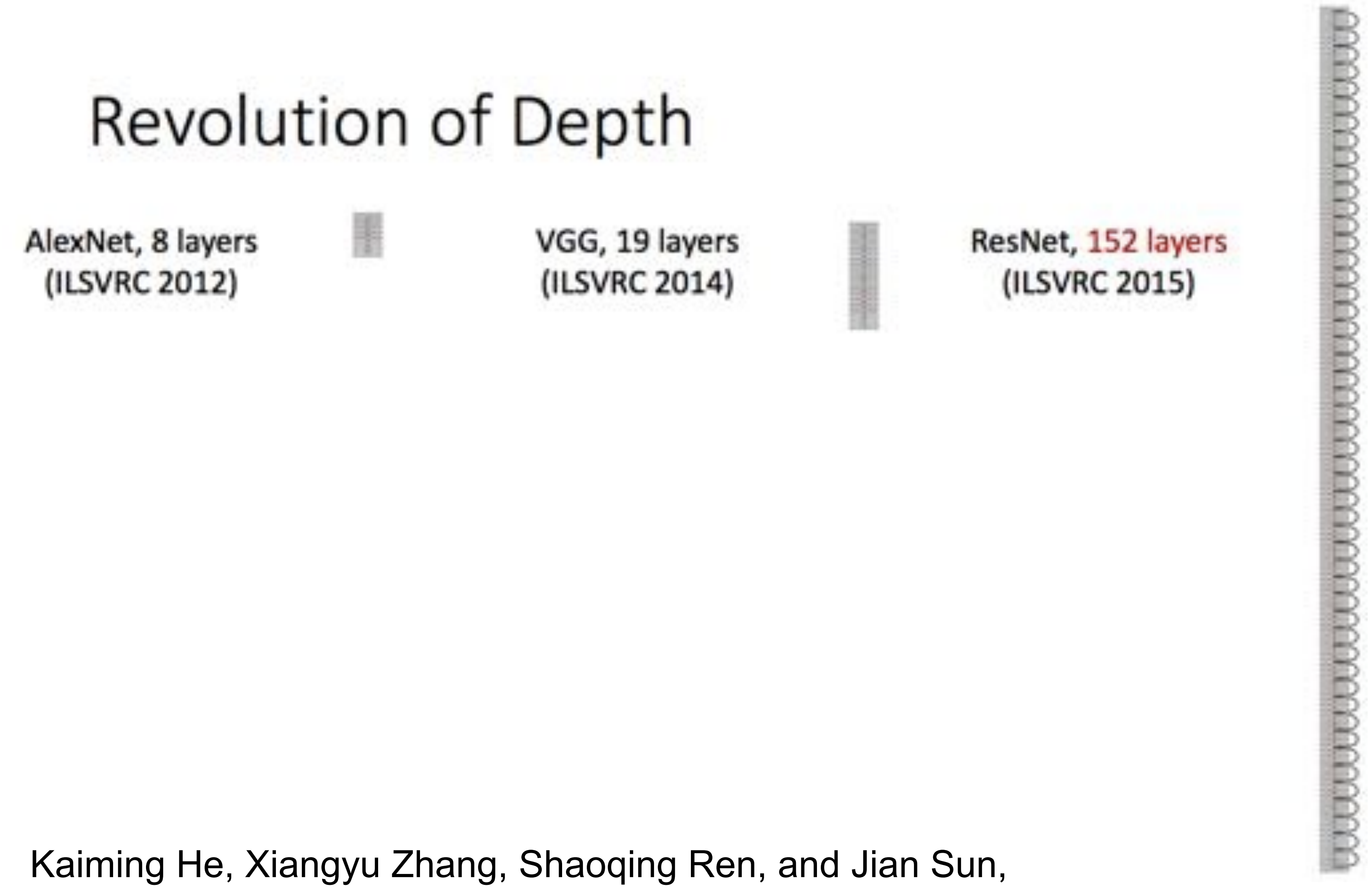
www.image-net.org/challenges/LSVRC/

ImageNet Challenge 2012-2014

Team	**Year**	**Place**	**Error (top-5)**	**External data**
SuperVision – Toronto (7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
Human expert*			5.1%	

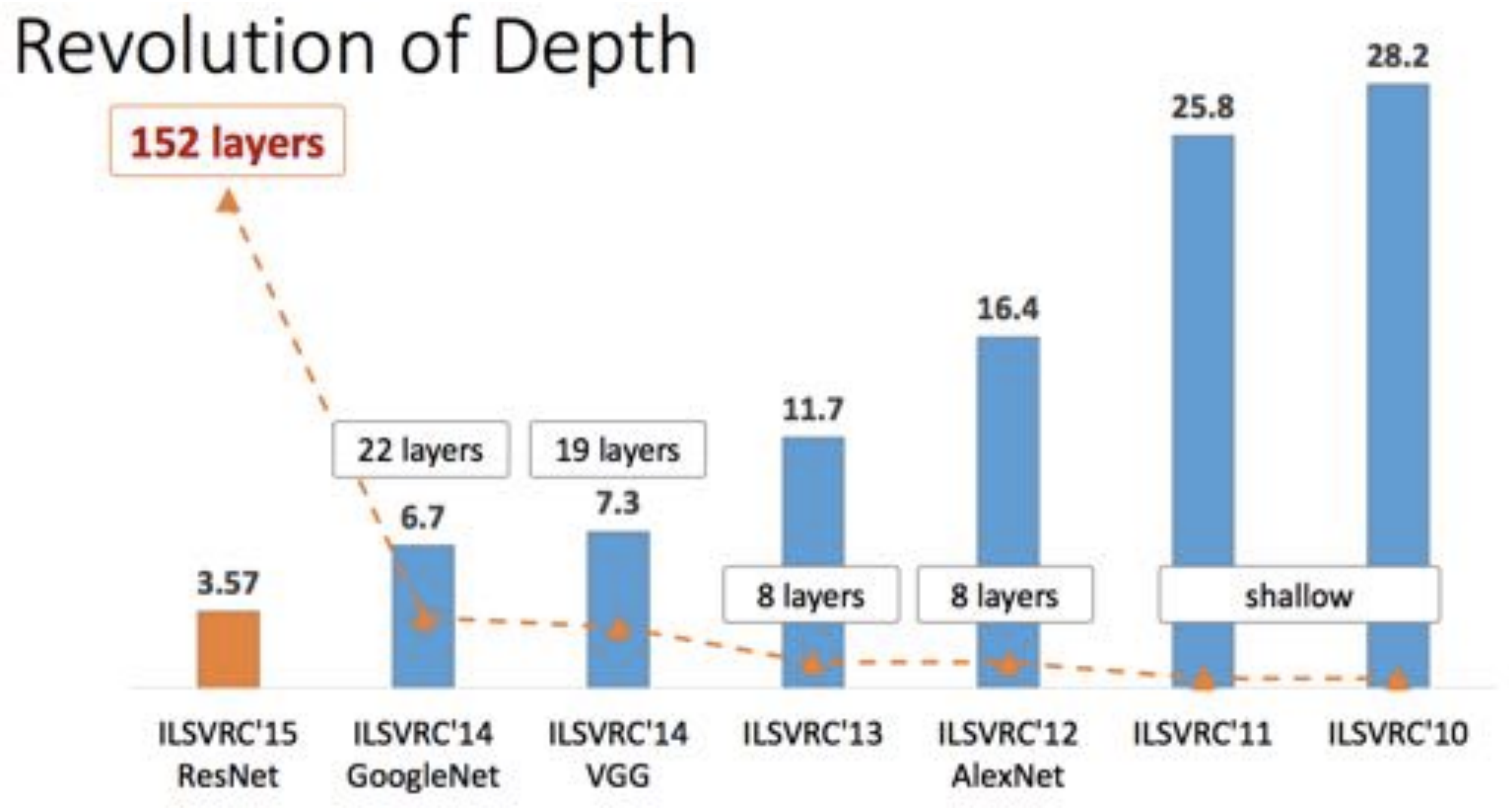
http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

Deep Residual Nets



Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun,
Deep Residual Learning for Image Recognition, arXiv 2015

Deep Residual Nets



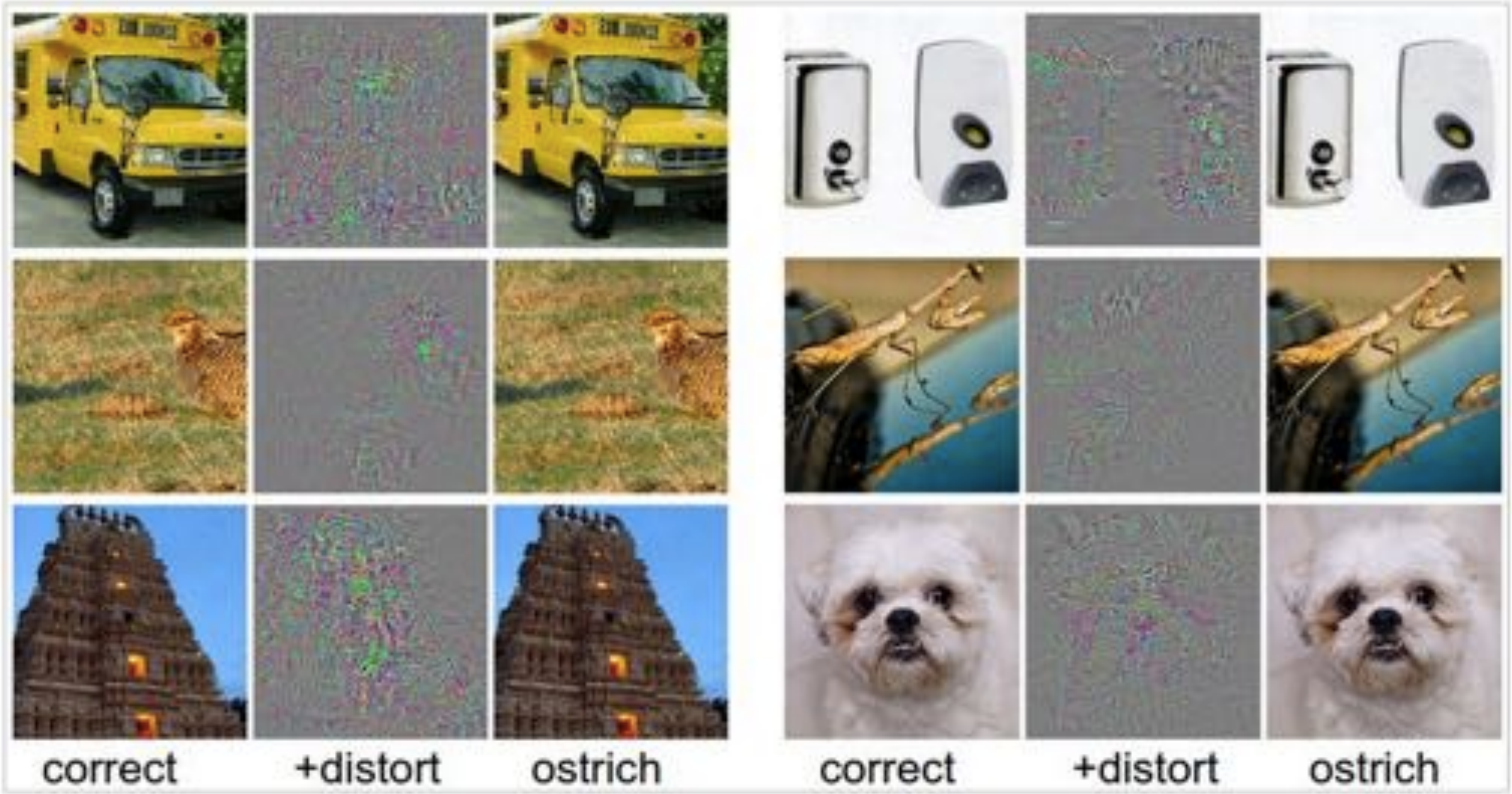
Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, arXiv 2015

Deep learning packages

- Caffe
- Torch
- Theano
- TensorFlow
- Matconvnet
- …

http://deeplearning.net/software_links/

Breaking CNNs

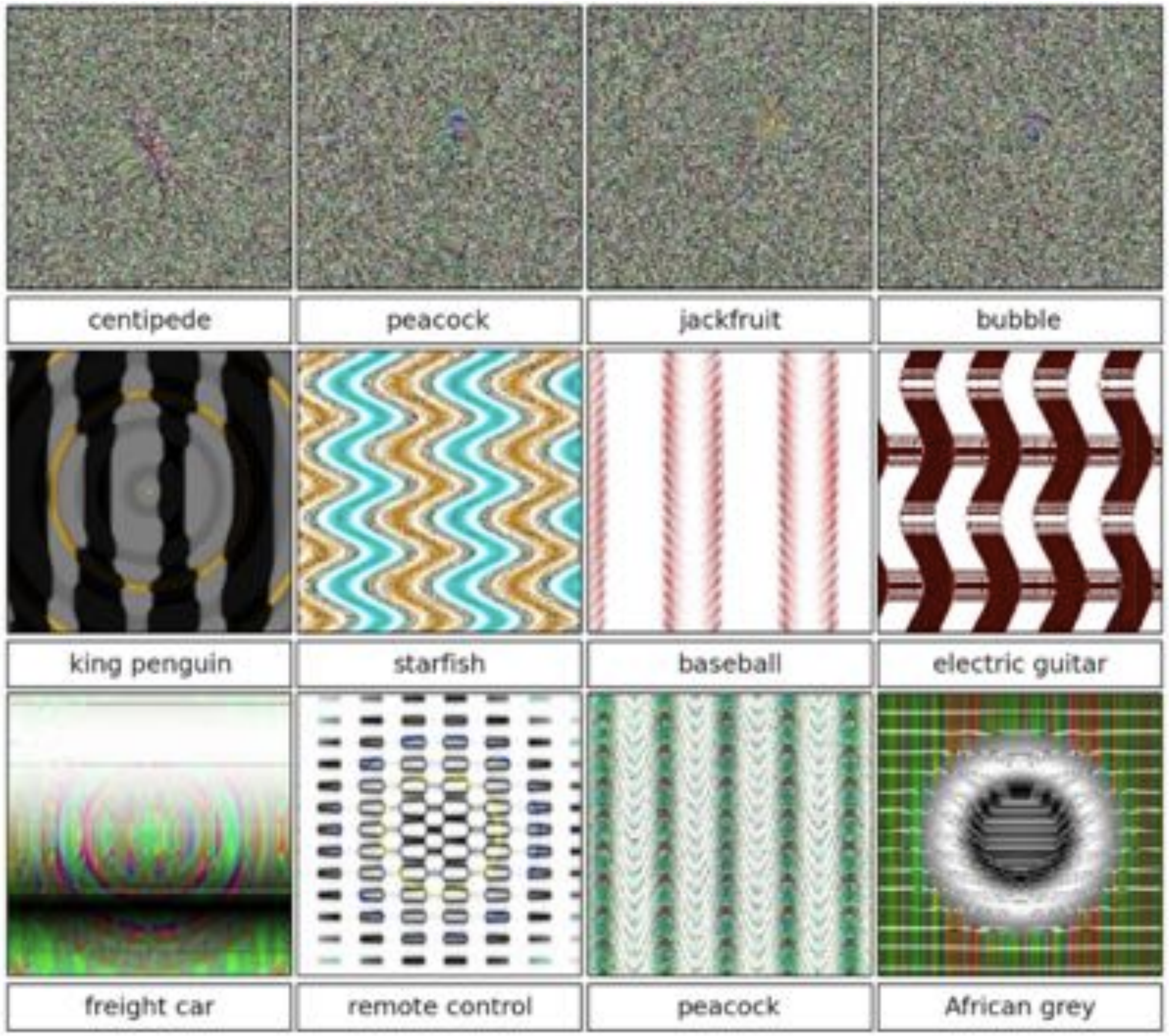


Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

http://arxiv.org/abs/1312.6199

http://karpathy.github.io/2015/03/30/breaking-convnets/

Breaking CNNs



http://arxiv.org/abs/1412.1897

http://karpathy.github.io/2015/03/30/breaking-convnets/

What is going on?

- Recall gradient descent training: modify the weights to reduce classifier error

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$$

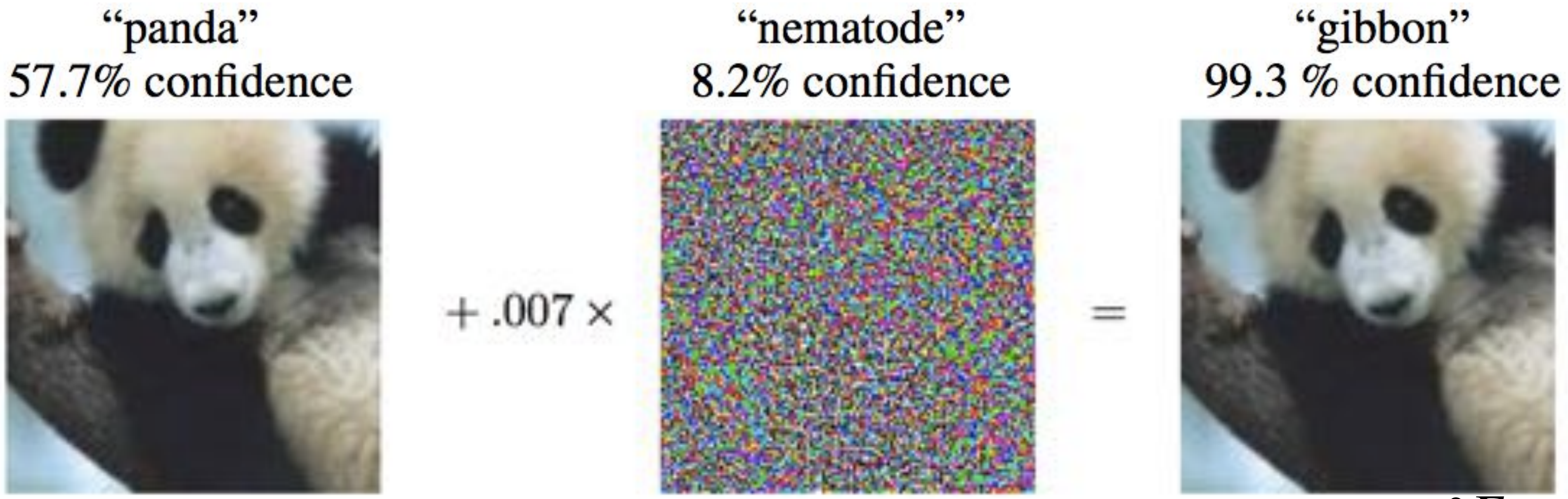
- Adversarial examples: modify the *image* to *increase* classifier error

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \frac{\partial E}{\partial \mathbf{x}}$$

http://arxiv.org/abs/1412.6572

http://karpathy.github.io/2015/03/30/breaking-convnets/

What is going on?



x

$$\frac{\partial E}{\partial \mathbf{x}}$$

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \frac{\partial E}{\partial \mathbf{x}}$$

http://arxiv.org/abs/1412.6572

http://karpathy.github.io/2015/03/30/breaking-convnets/

Fooling a linear classifier

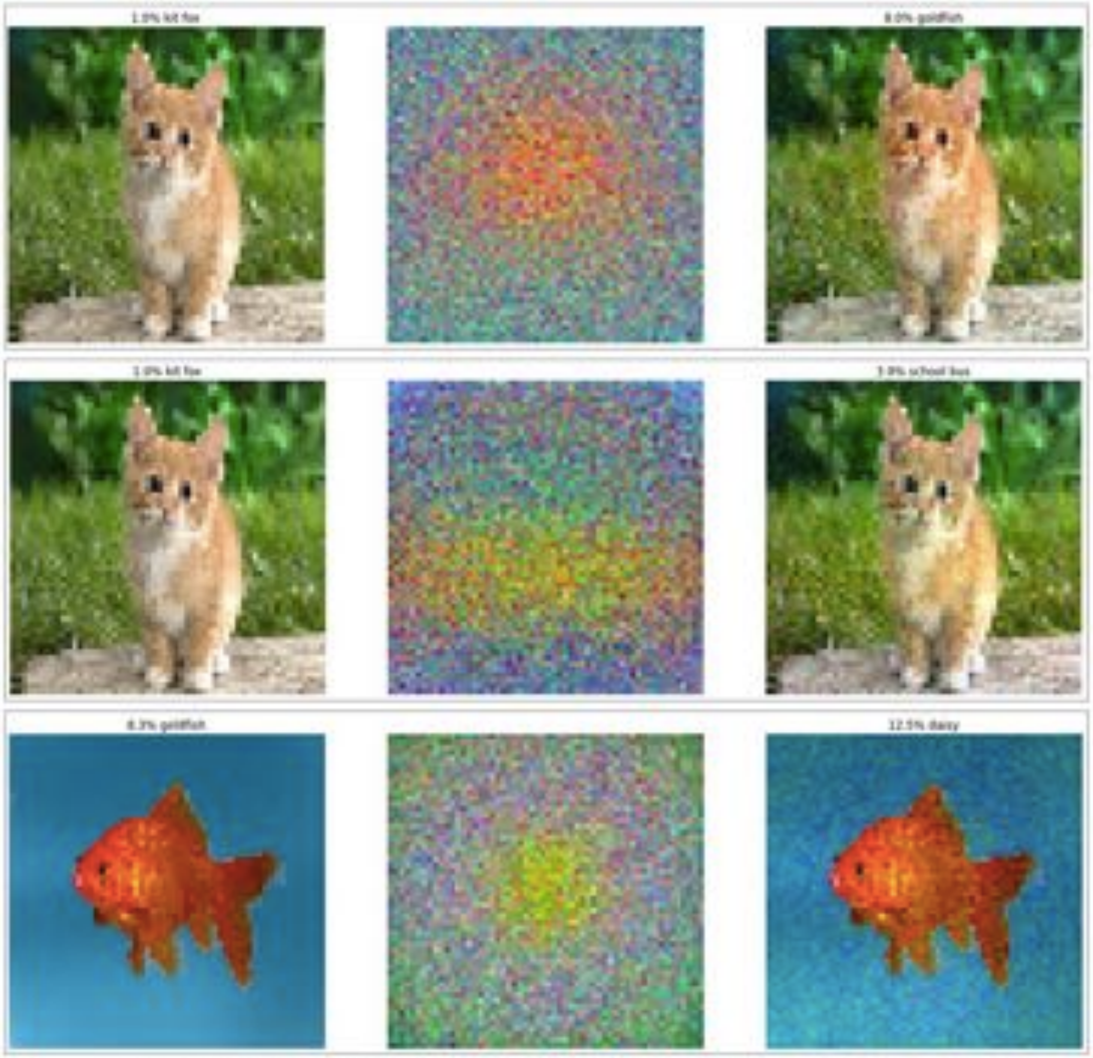
- Perceptron weight update: add a small multiple of the example to the weight vector:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \mathbf{x}$$

- To fool a linear classifier, add a small multiple of the weight vector to the training example:

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \mathbf{w}$$

Fooling a linear classifier



Fooled linear classifier: The starting image (left) is classified as a kit fox. That's incorrect, but then what can you expect from a linear classifier? However, if we add a small amount "goldfish" weights to the image (top row, middle), suddenly the classifier is convinced that it's looking at one with high confidence. We can distort it with the school bus template instead if we wanted to.

http://karpathy.github.io/2015/03/30/breaking-convnets/

Summary

- Currently most applications of deep neural networks in computer vision are based on supervised learning
- Need lots of annotated training data (and GPUs)
- Training via back-propagation takes time and is prone to local minima (many tips and tricks)
- Manual design of feature detectors and descriptors is replaced by manual design of network architectures (-> better automation needed)
- Lately there has been progress in weakly supervised and unsupervised learning (e.g. generative adversarial networks)