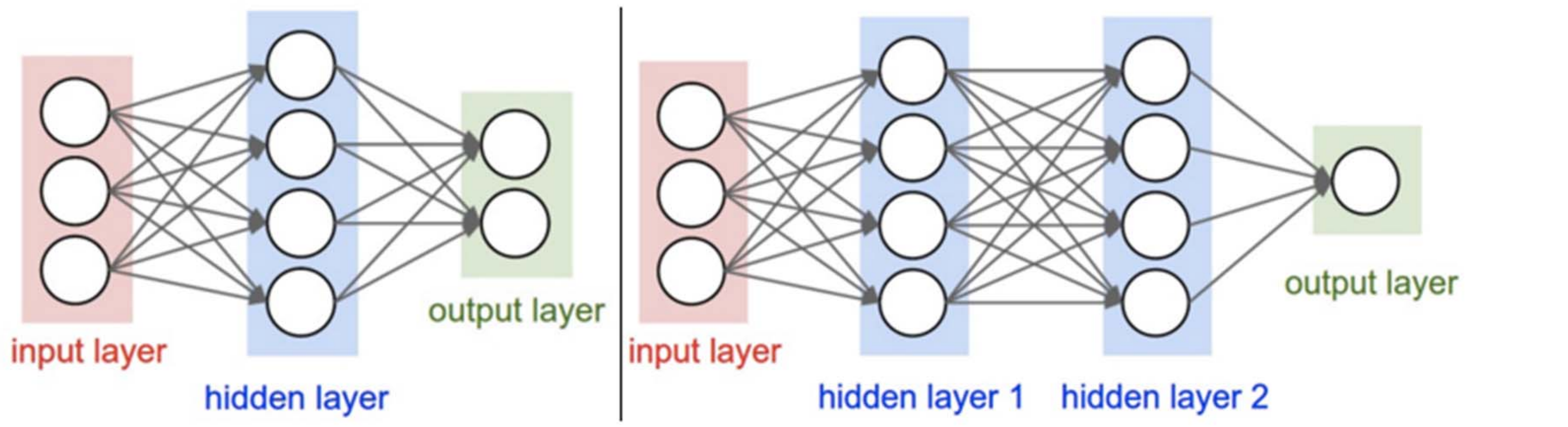
Lecture 23: Deep Learning

Slides by Svetlana Lazebnik, 11/2016

Modified by Mark Hasegawa-Johnson, 11/2017

Beyond a single hidden layer



Left: A 2-layer Neural Network (one hidden layer of 4 neurons (or units) and one output layer with 2 neurons), and three inputs. Right: A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer.

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

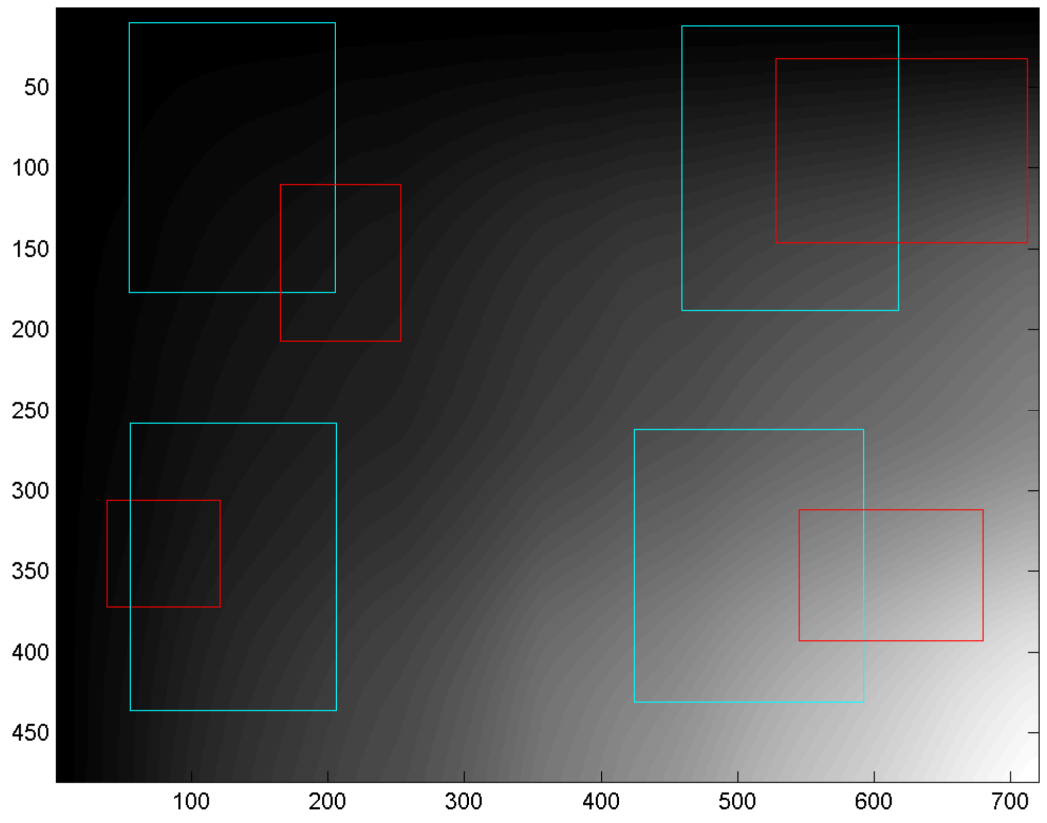
Deep networks basic idea

- A linear classifier performs OK when you apply it to raw pixels
- It works much better if you apply it to higher-level features, e.g., detected edges, corners, patterns, etc.
- Pre-deep learning: detect features, then give them to a linear classifier
- Deep learning: learn the features and the classifier at the same time

Outline

- Adaboost: learn the features, then learn the classifier
- Convolutional neural networks
- Adversarial inputs

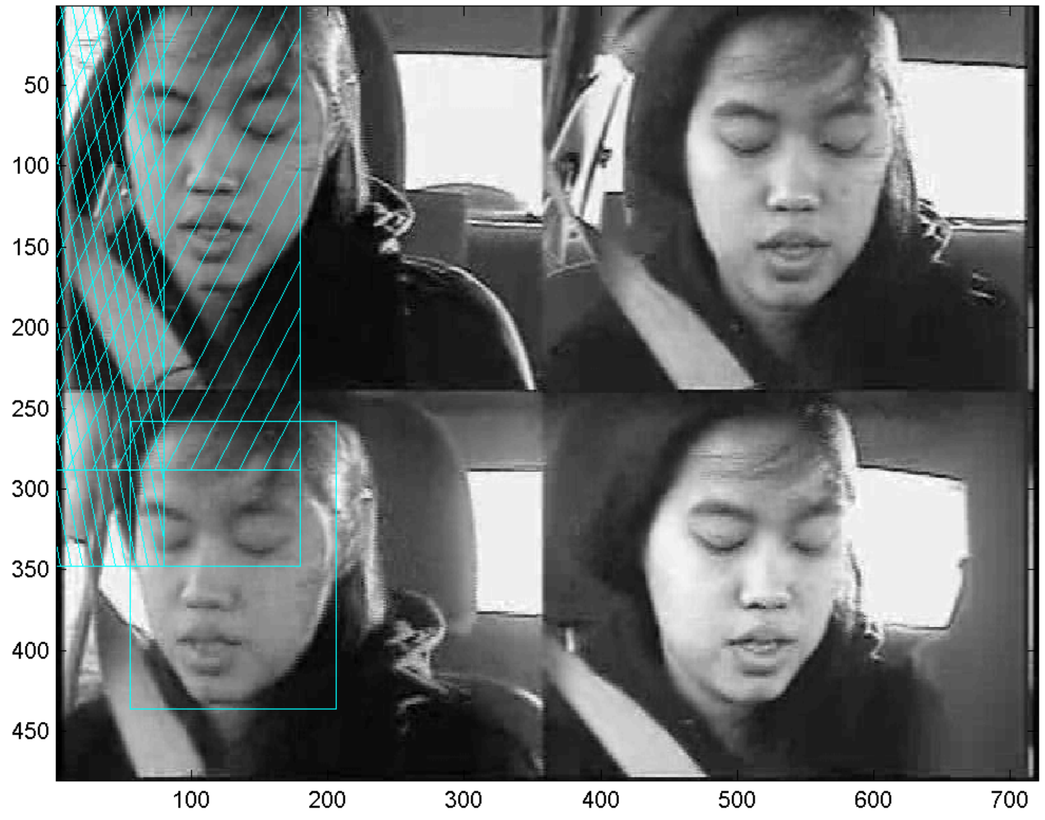
Integral image



$$ii(y,x) = \sum_{x' \le x, y' \le y} i(y',x')$$

```
ii = cumsum(cumsum(i,2),1);
imagesc(ii);
```

Efficiently computing the sum

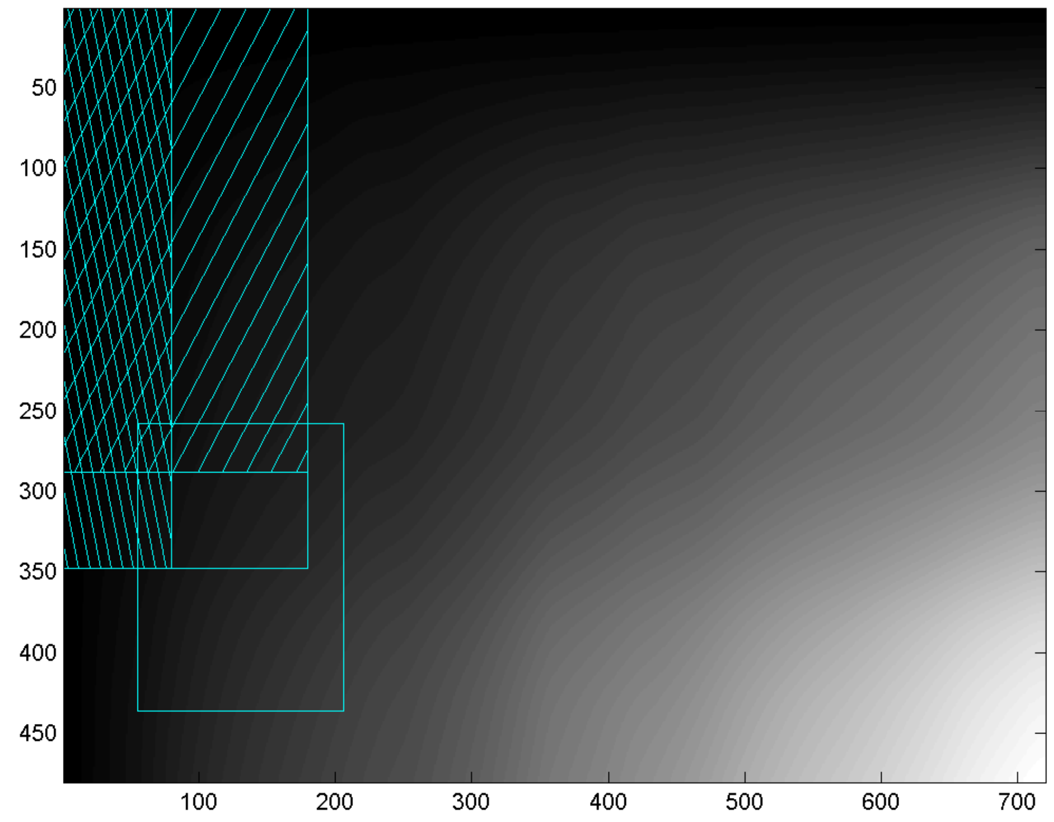


The sum within the subrectangle is:

sum(open pixels) –
sum(/// pixels) –
sum(\\\ pixels) +
sum(### pixels)

The last term is necessary because, by subtracting the two previous terms, we have subtracted the ### pixels twice; it is necessary to compensate.

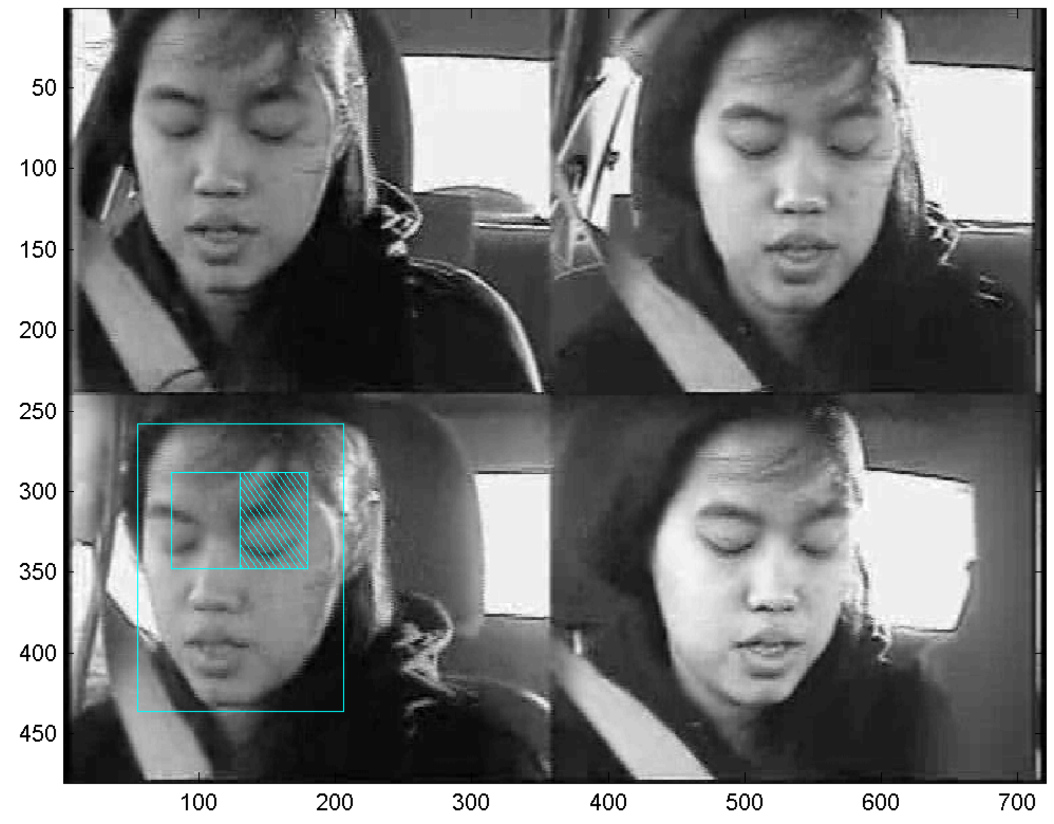
Efficiently computing the sum



In the integral image, each point is a sum! Thus the feature we want is just

ii(y2,x2) –
ii(y1,x2) –
ii(y2,x1) +
ii(y1,x1)

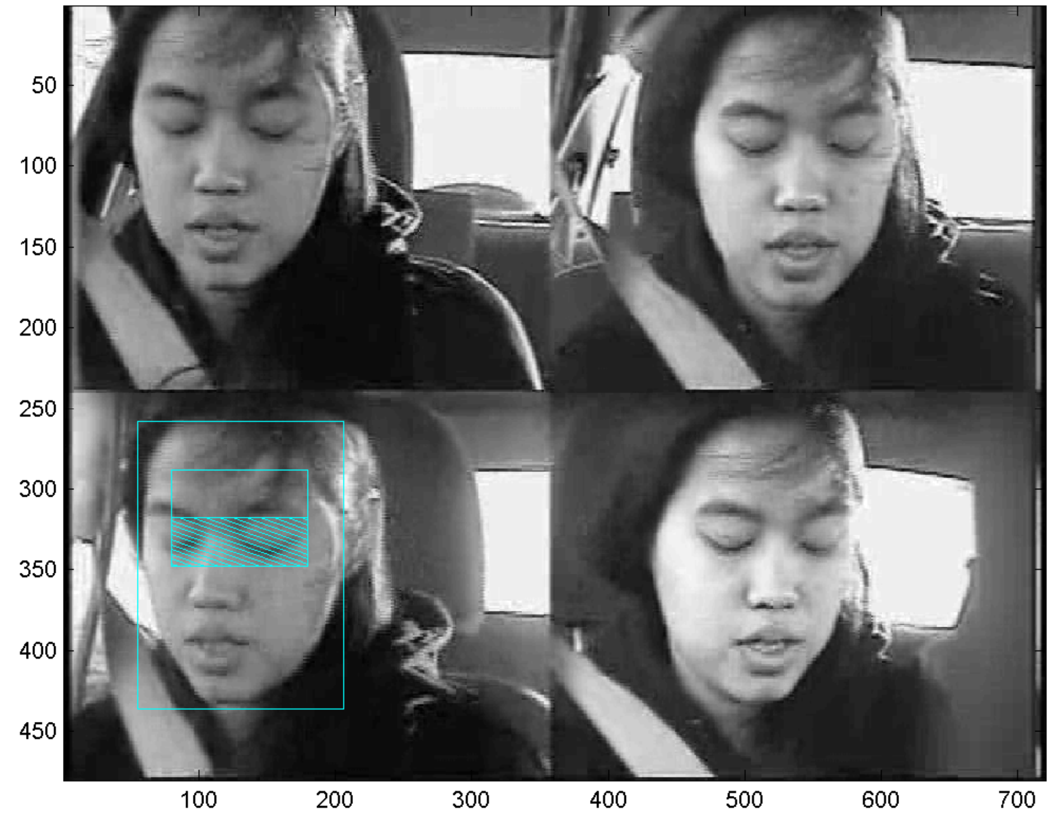
Other useful features: order 2, horizontal



Feature f(x;fr,q=2,v=0)

An order-2 horizontal feature is the sum of the right half, minus the sum of the left half.

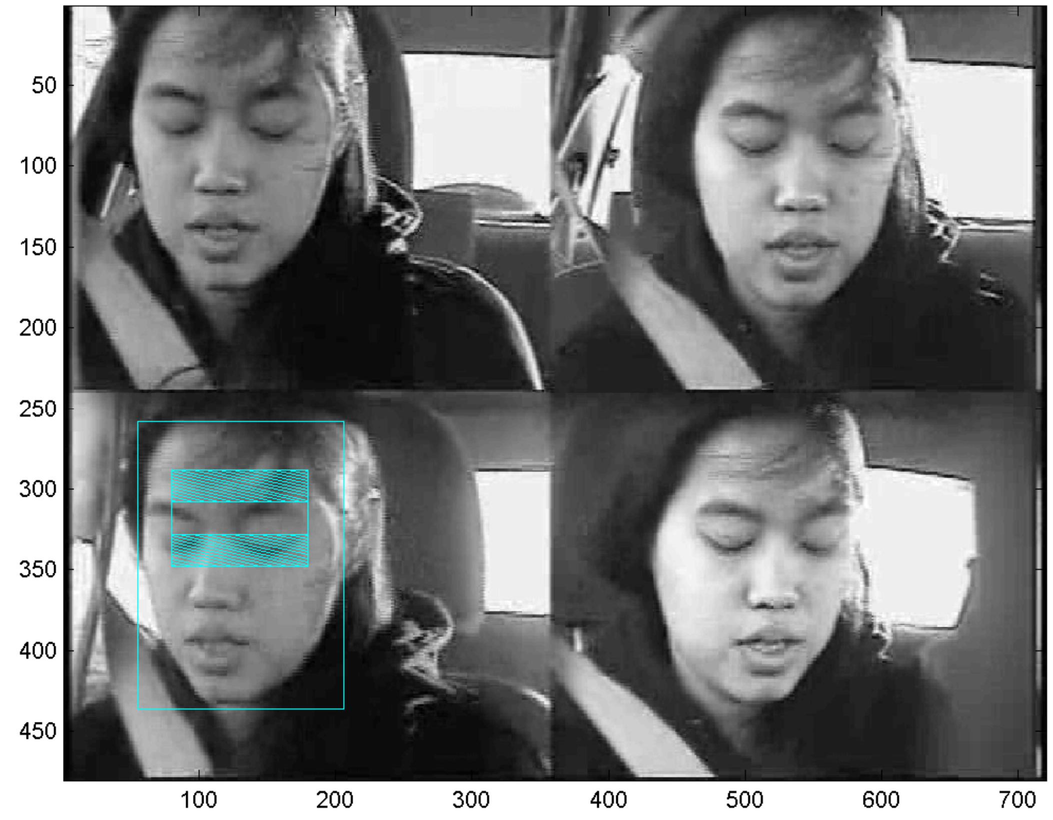
Other useful features: order 2, vertical



Feature f(x;fr,q=2,v=1)

An order-2 vertical feature is the sum of the bottom half, minus the sum of the top half.

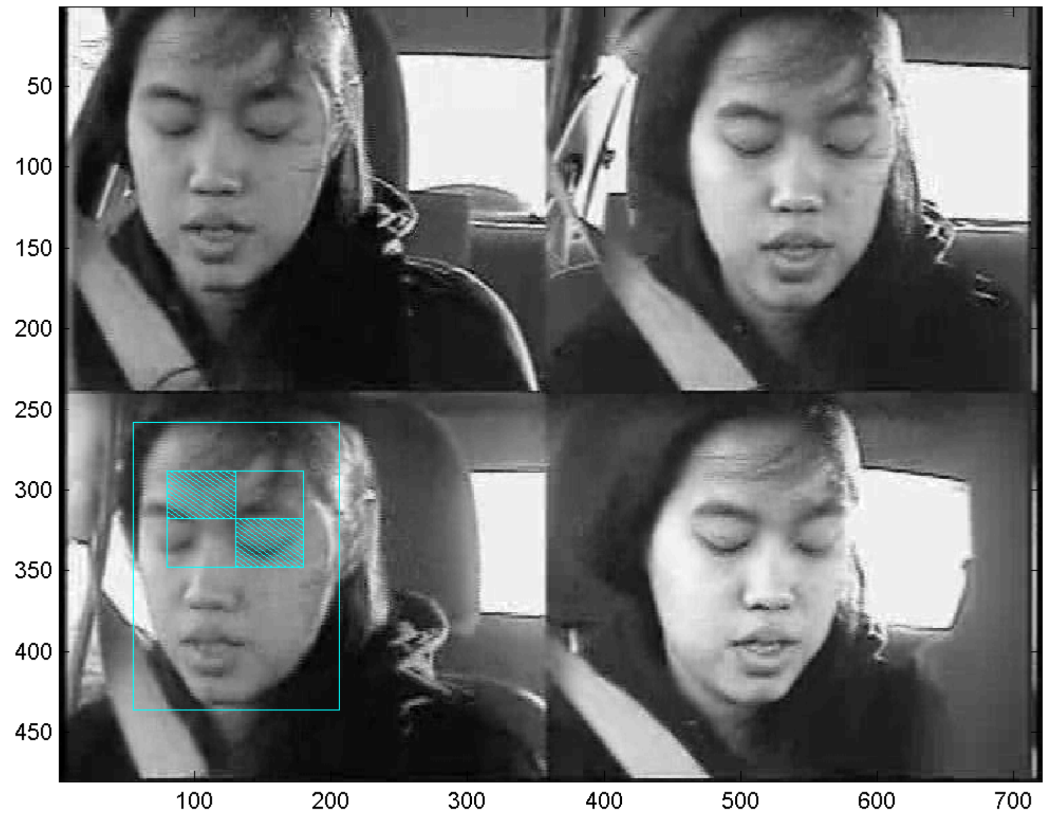
Other useful features: order 3, vertical



Feature f(x;fr,q=3,v=1)

An order-3 vertical feature is the sum of the outer thirds, minus the sum of the middle third.

Other useful features: order 4



Feature f(x;fr,q=4)

An order-4 feature is the sum of the main diagonal quadrants, minus the sum of the off-diagonal quadrants.

Scalar Classifier

$$h(x; fr, q, v, p, \theta) = \begin{cases} 1 & f(x; fr, q, v) < \theta \\ 0 & otherwise \end{cases}$$

Recap: finding the best scalar classifier out of a set containing tens of thousands of possible scalar classifiers

1. For every possible feature (fx,fy,fw,fh,q,v),
2. … compute the feature for the whole database…
3. … compute the probability of error…
4. … if the answer to #3 is the best one you've found so far, keep it.

Adaboost

Suppose somebody told you: I’m going to take a whole bunch of scalar classifiers. Let’s use $h_t(x)$ to mean the classifier computed in the t’th training iteration; remember that $h_t(x)$ is either 0 or 1. Then I’m going to add them all together, and the final classifier will be

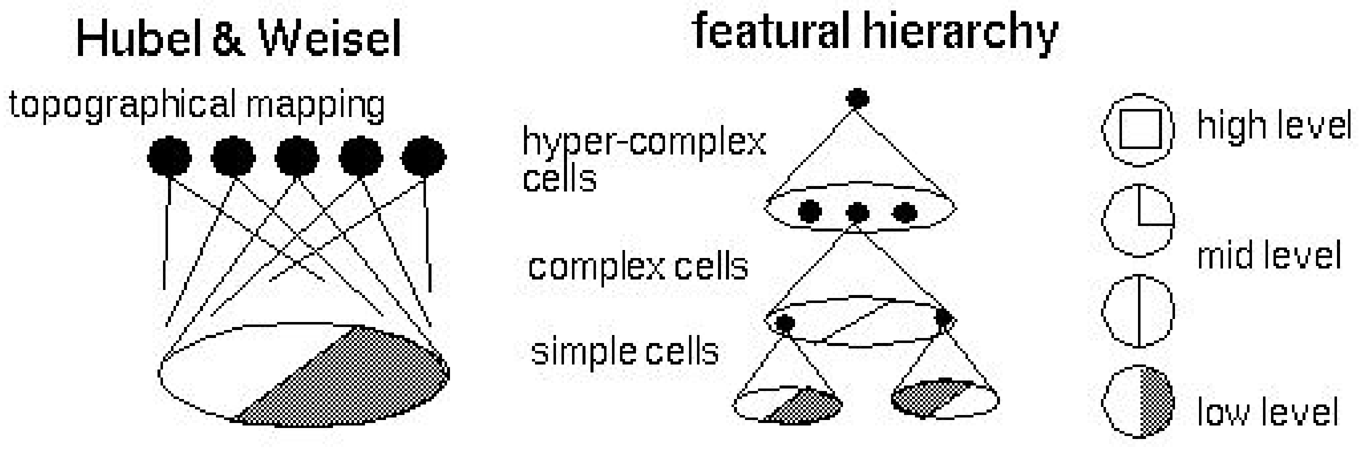
$$h(x) = \begin{cases} 1 \;\; if \; \sum_t \alpha_t (2h_t(x) - 1) \; > 0 \\ 0 \;\; if \; \sum_t \alpha_t (2h_t(x) - 1) < 0 \end{cases}$$

Outline

- Adaboost: learn the features, then learn the classifier
- Convolutional neural networks
- Adversarial inputs

Biological inspiration

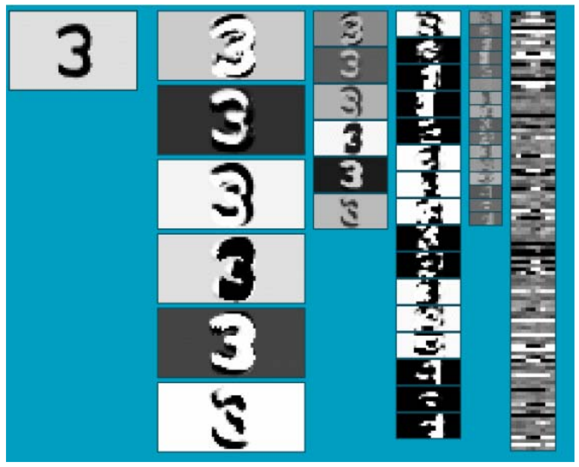
- D. Hubel and T. Wiesel (1959, 1962, Nobel Prize 1981)
  - Visual cortex consists of a hierarchy of *simple*, *complex*, and *hyper-complex* cells

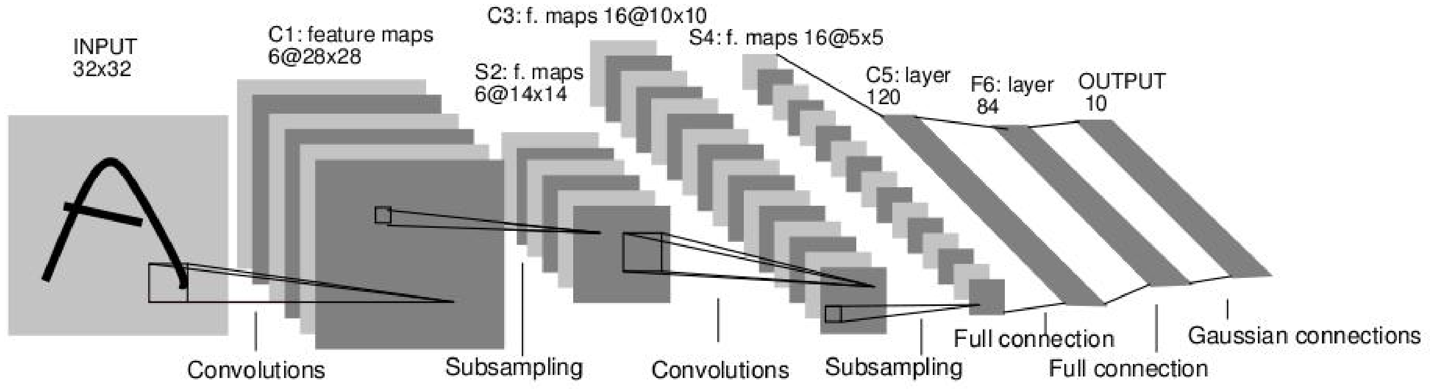


Source

Convolutional Neural Networks

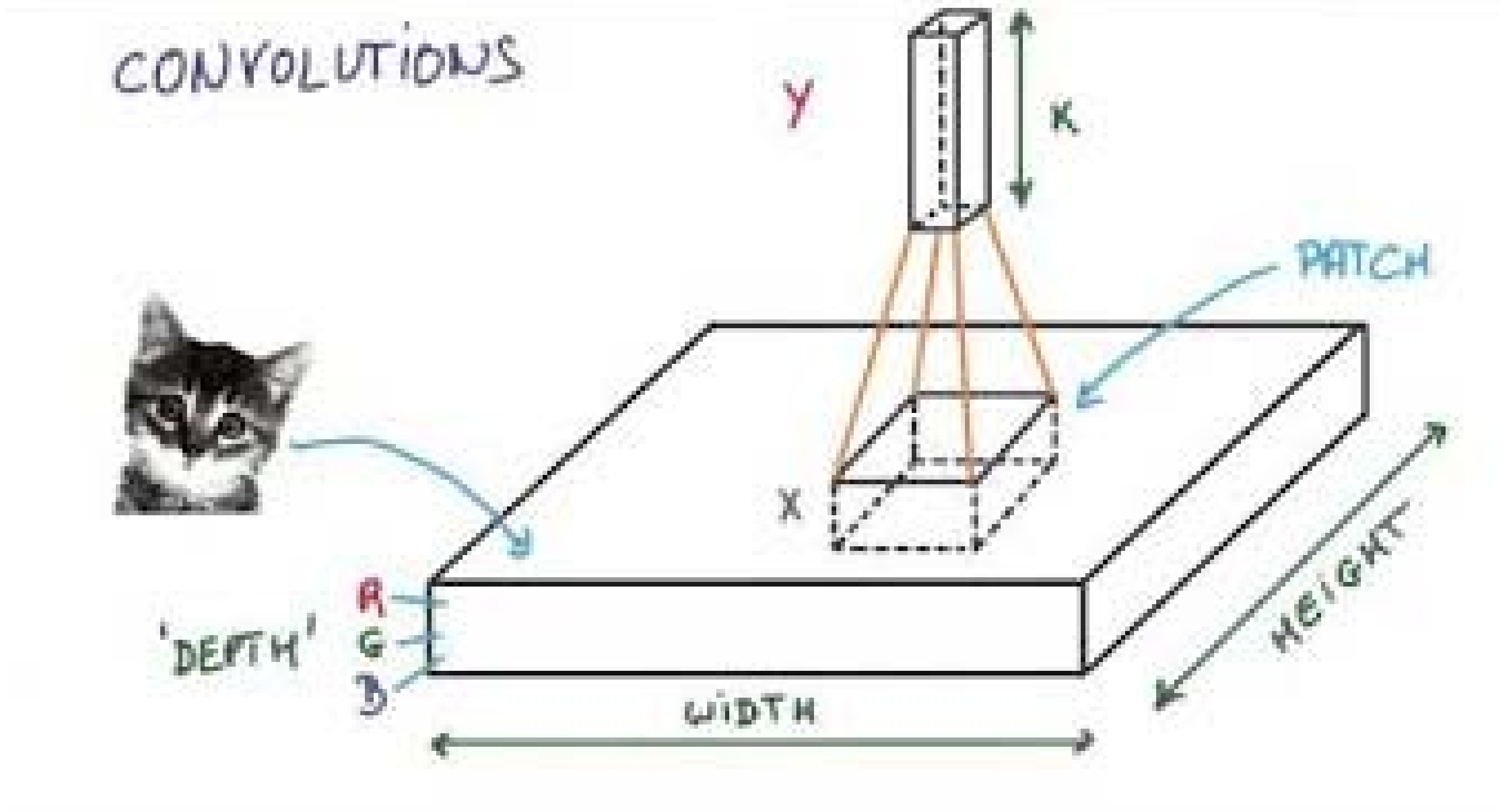
- Neural network with specialized connectivity structure
- Stack multiple stages of feature extractors
- Higher stages compute more global, more invariant features
- Classification layer at the end





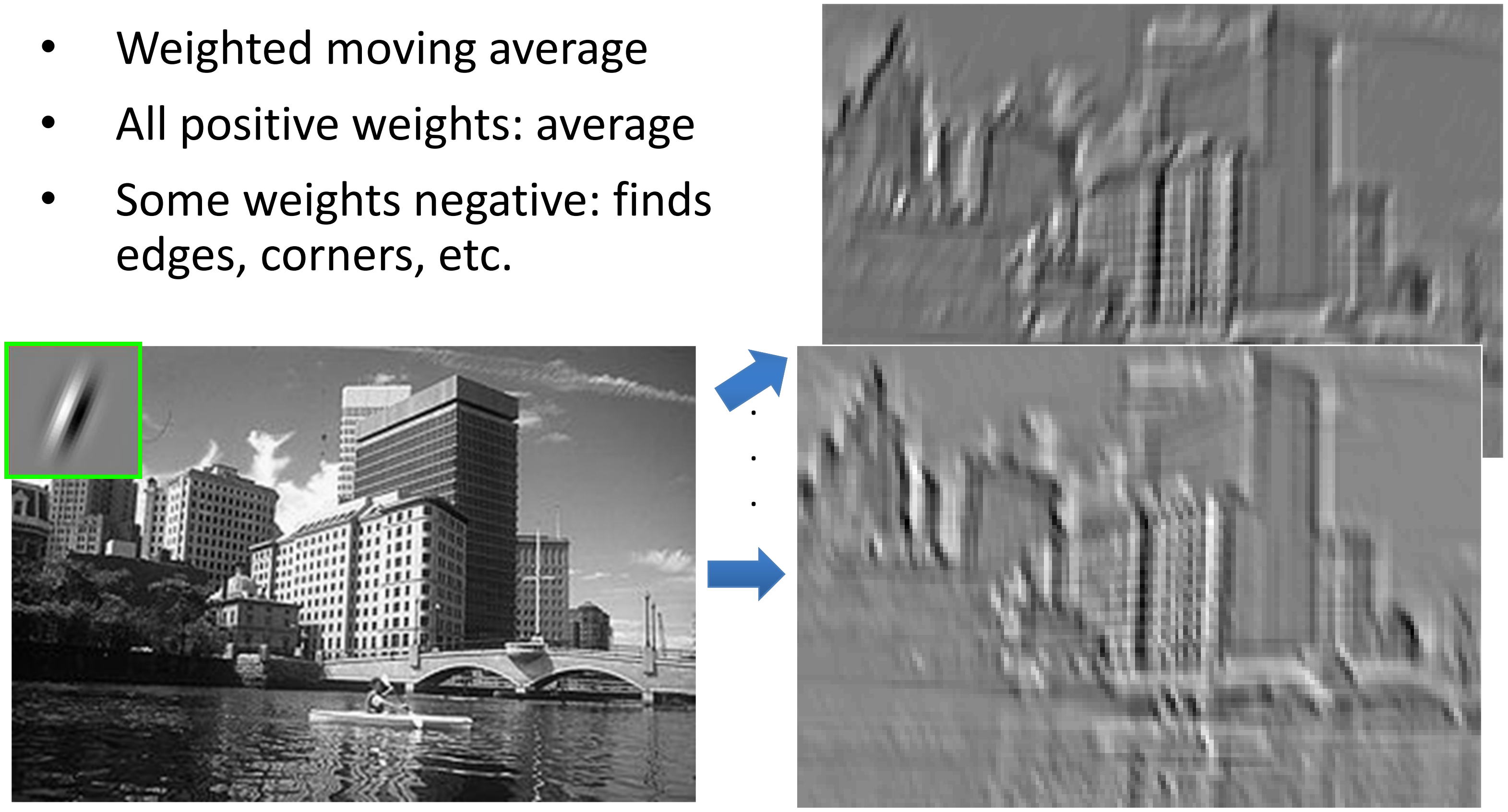
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

CNN (Convolutional Neural Networks)

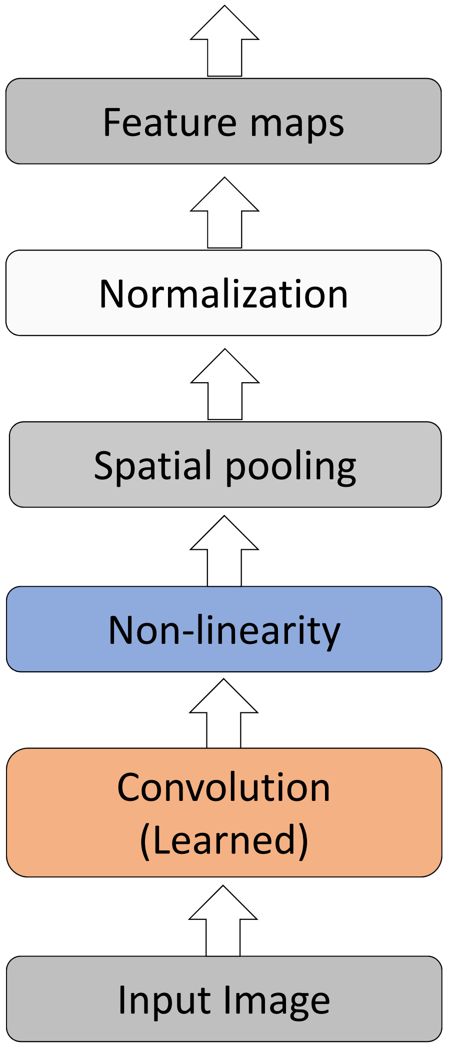


What is a convolution?

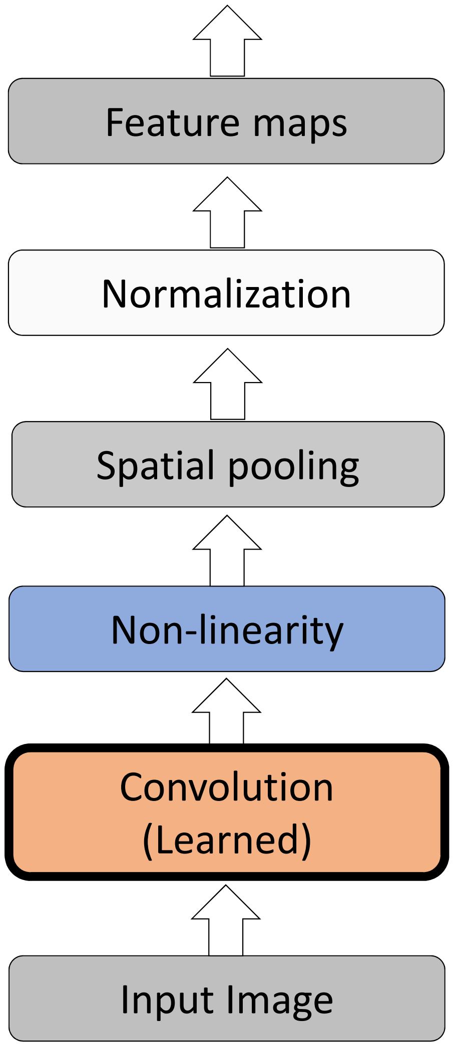
- Weighted moving average
- All positive weights: average
- Some weights negative: finds edges, corners, etc.

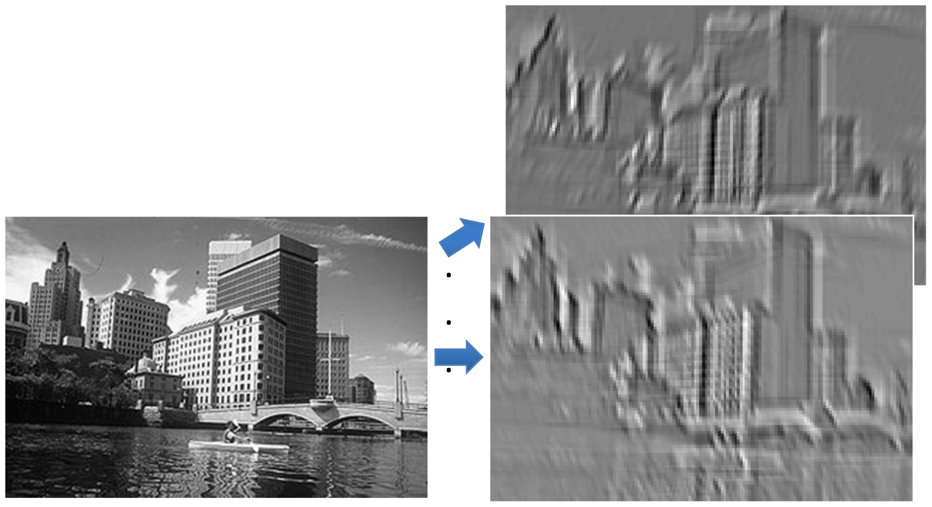


Convolutional Neural Networks

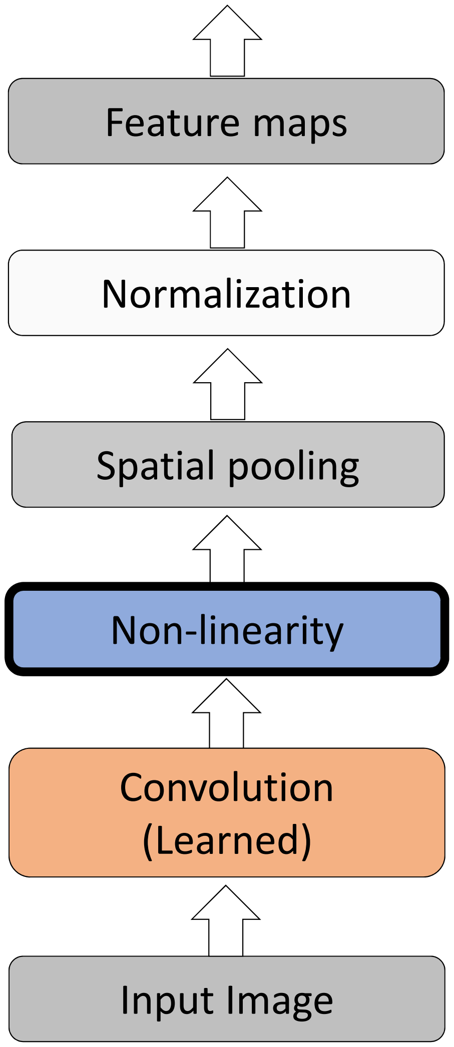


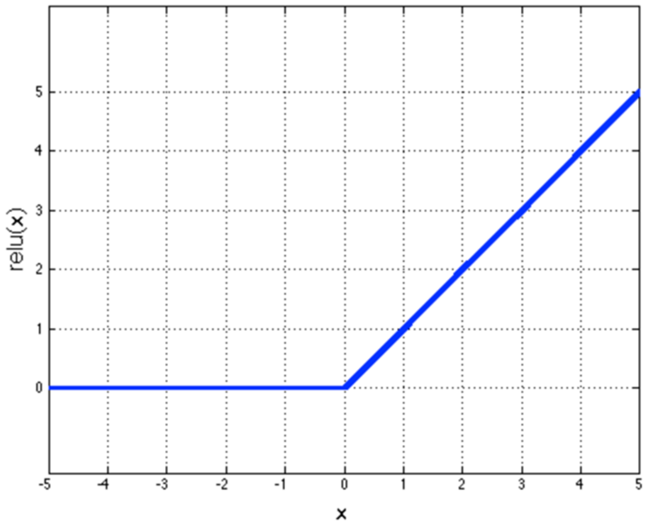
Convolutional Neural Networks



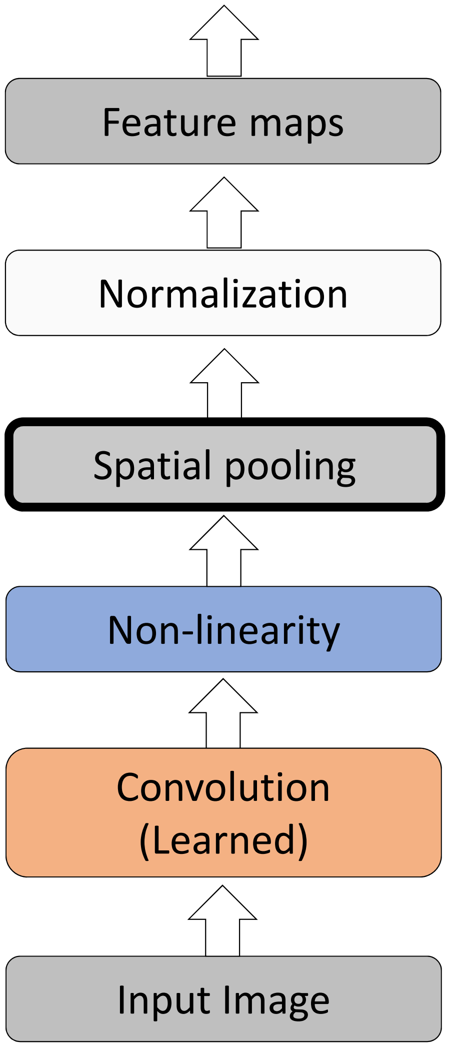


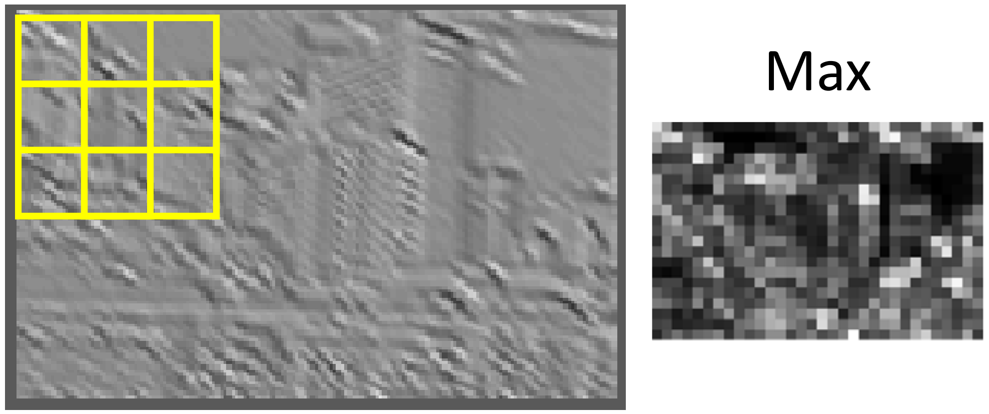
Convolutional Neural Networks



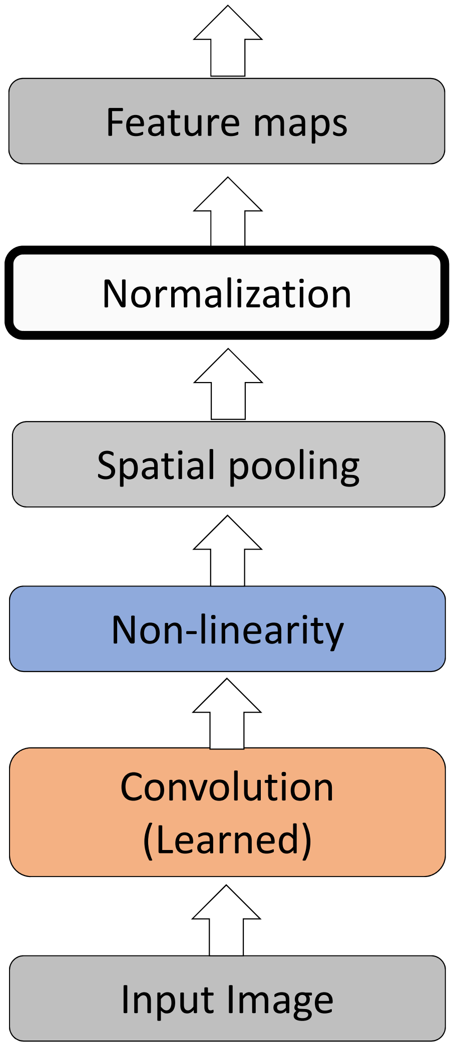


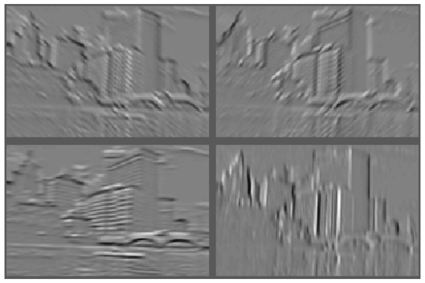
Convolutional Neural Networks

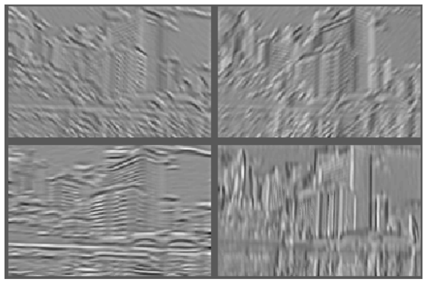




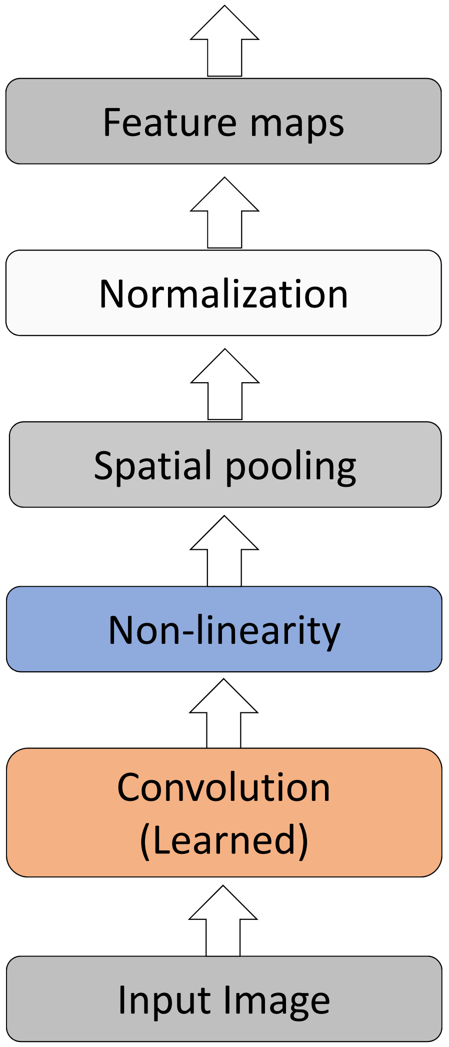
Convolutional Neural Networks




Feature Maps


Feature Maps
After Contrast
Normalization

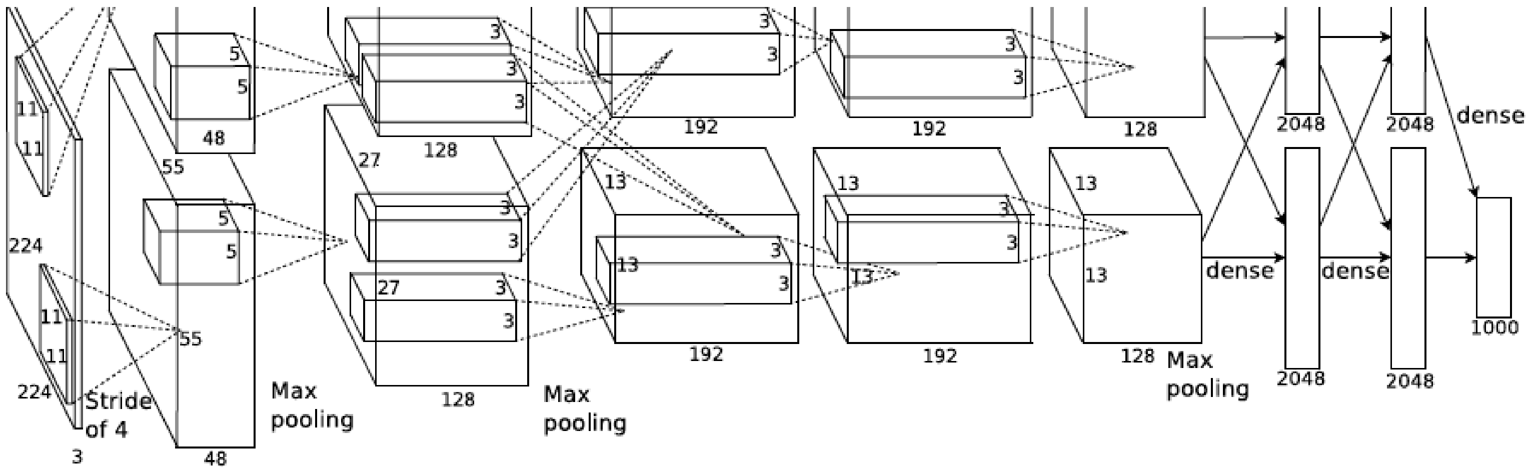
Convolutional Neural Networks



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

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

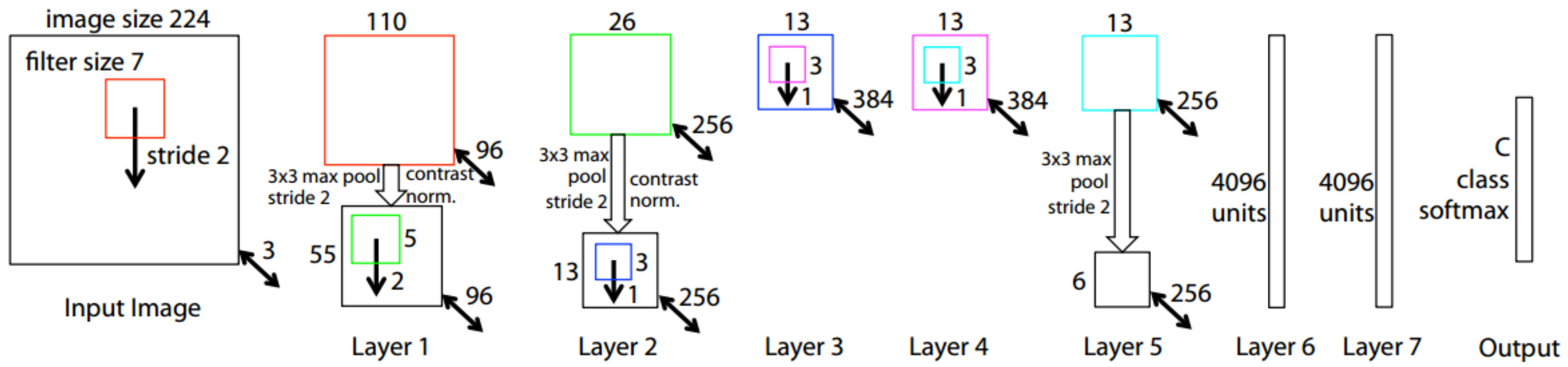
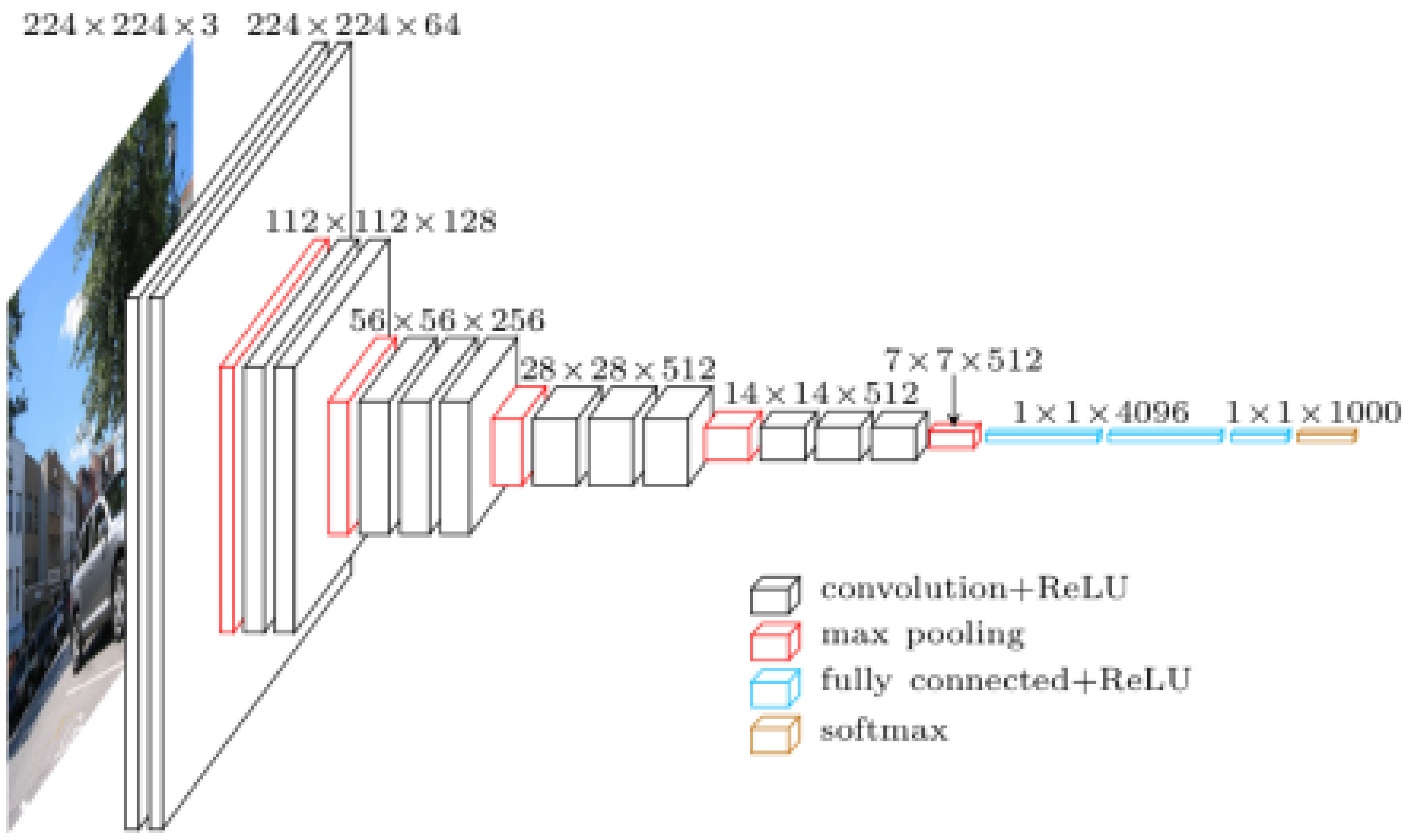


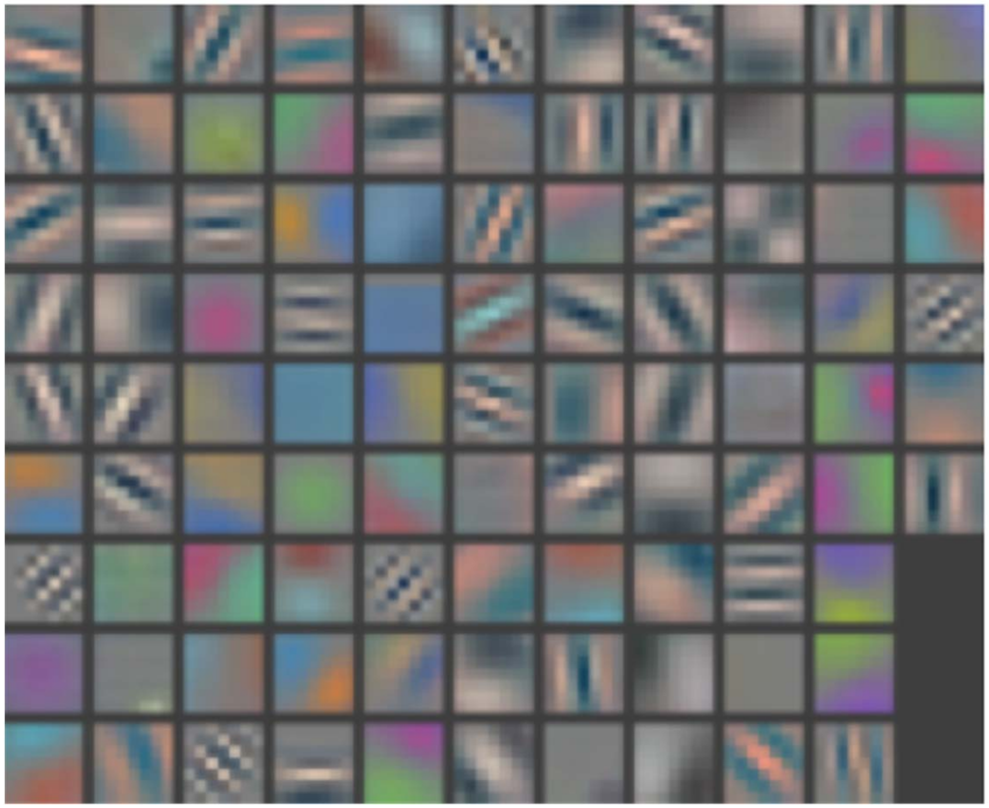
Figure 3. Architecture of our 8 layer convnet model. A 224 by 224 crop of an image (with 3 color planes) is presented as the input. This is convolved with 96 different 1st layer filters (red), each of size 7 by 7, using a stride of 2 in both x and y. The resulting feature maps are then: (i) passed through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

CNN for Image Classification: VGG16

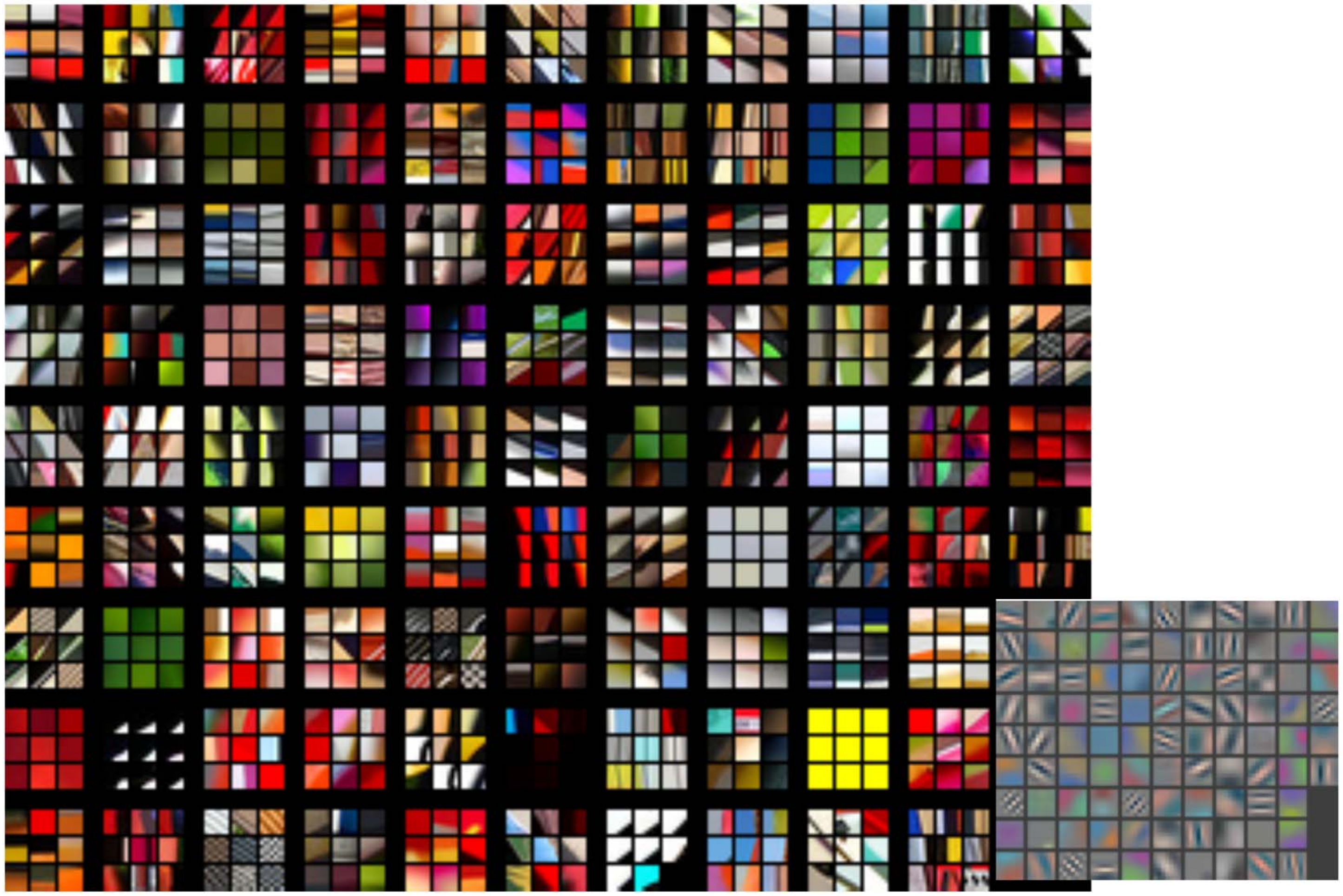


Layer 1 Filters

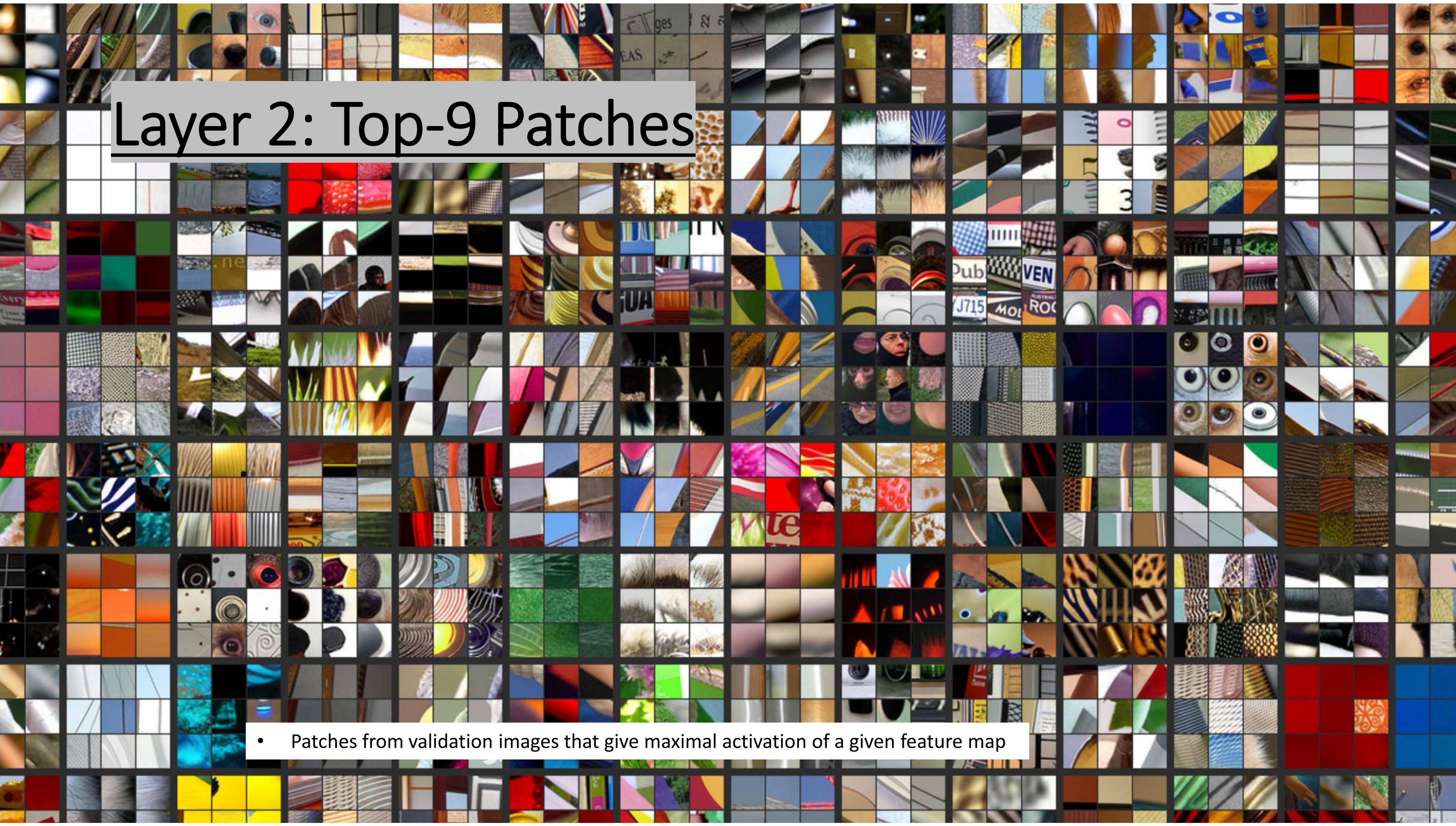


M. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, arXiv preprint, 2013

Layer 1: Top-9 Patches

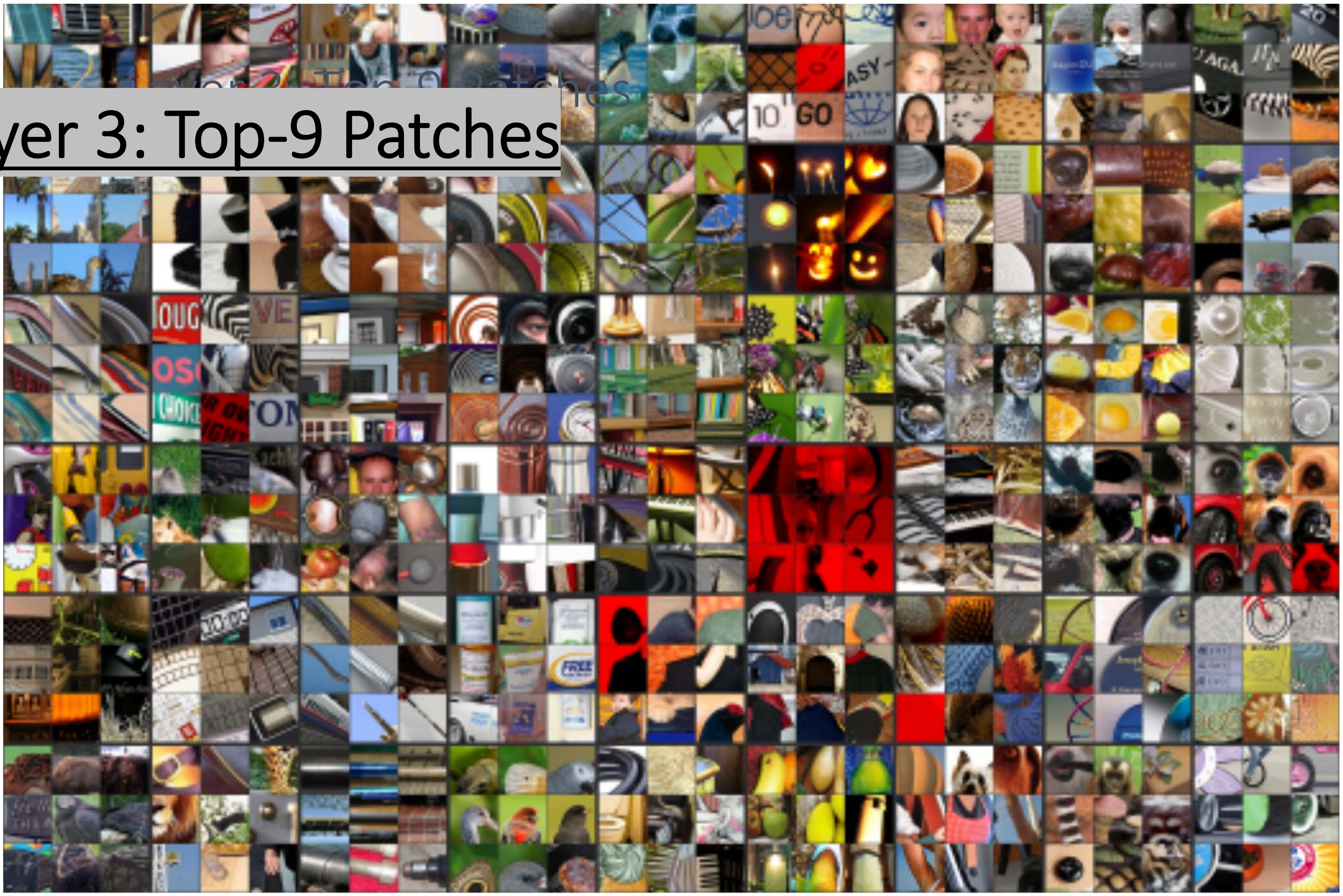


Layer 2: Top-9 Patches

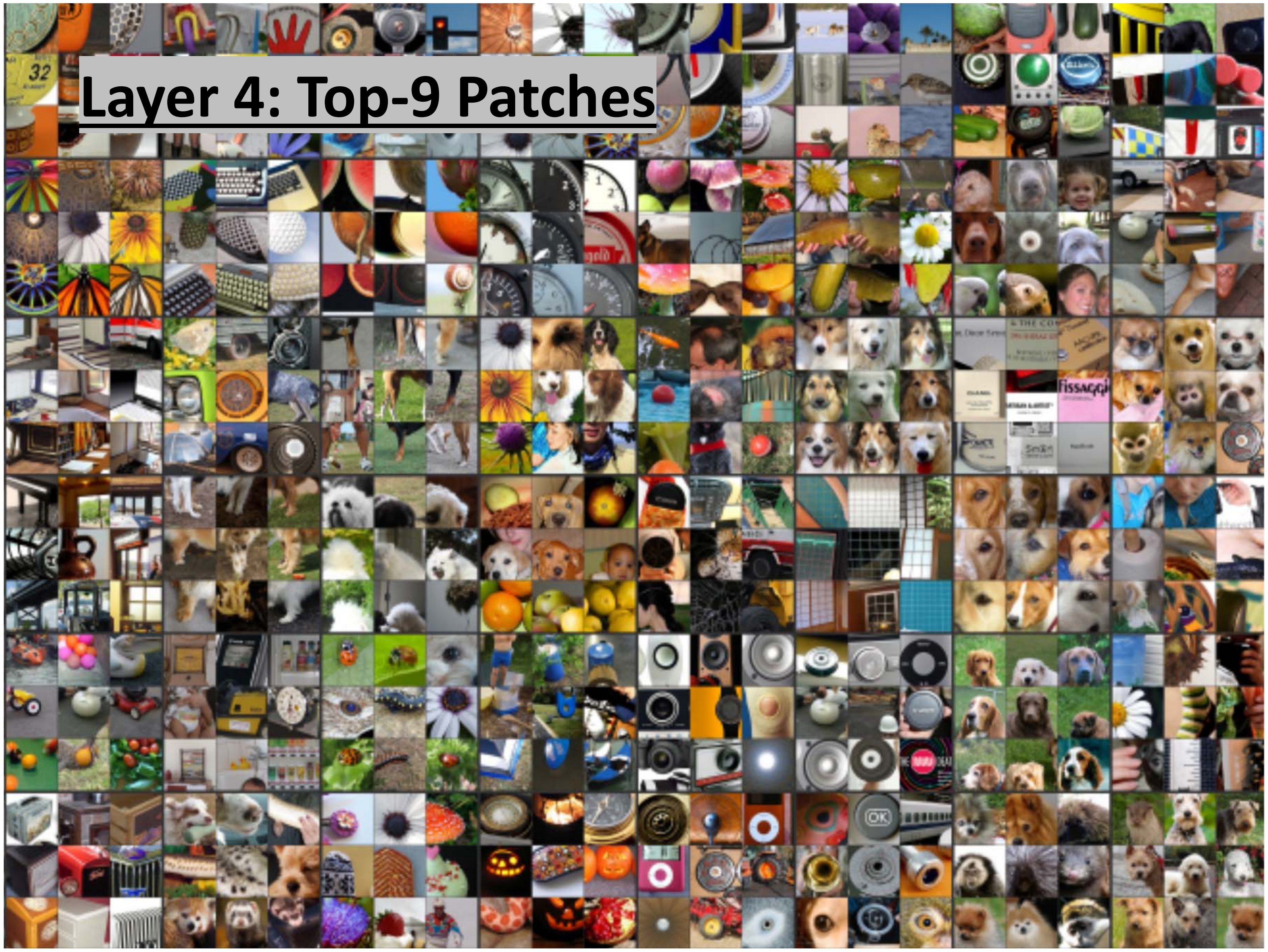


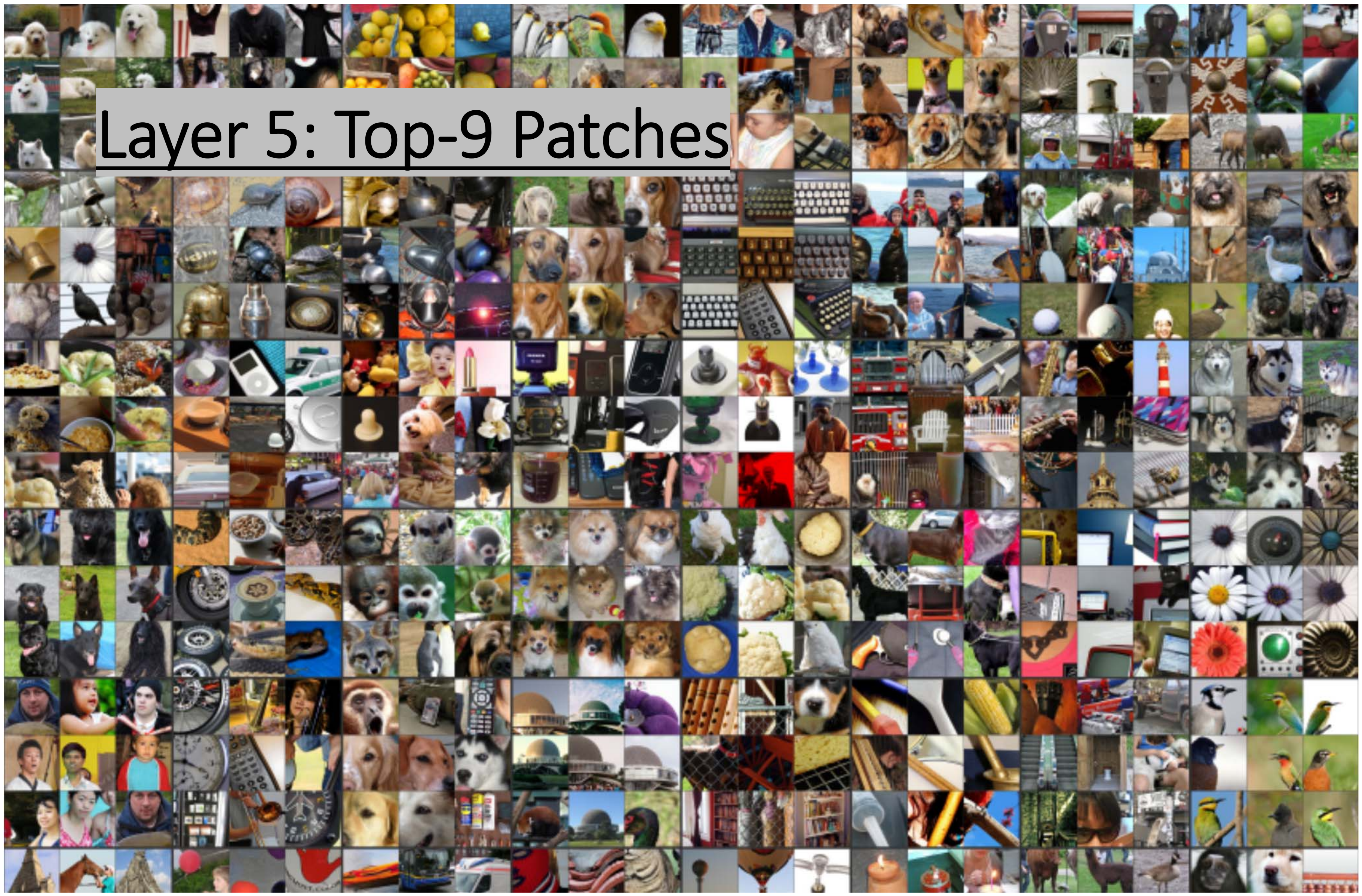
- Patches from validation images that give maximal activation of a given feature map

Layer 3: Top-9 Patches



Layer 4: Top-9 Patches





Layer 5: Top-9 Patches

ImageNet Challenge

[Deng et al. CVPR 2009]

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

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

ImageNet Challenge 2012-2014

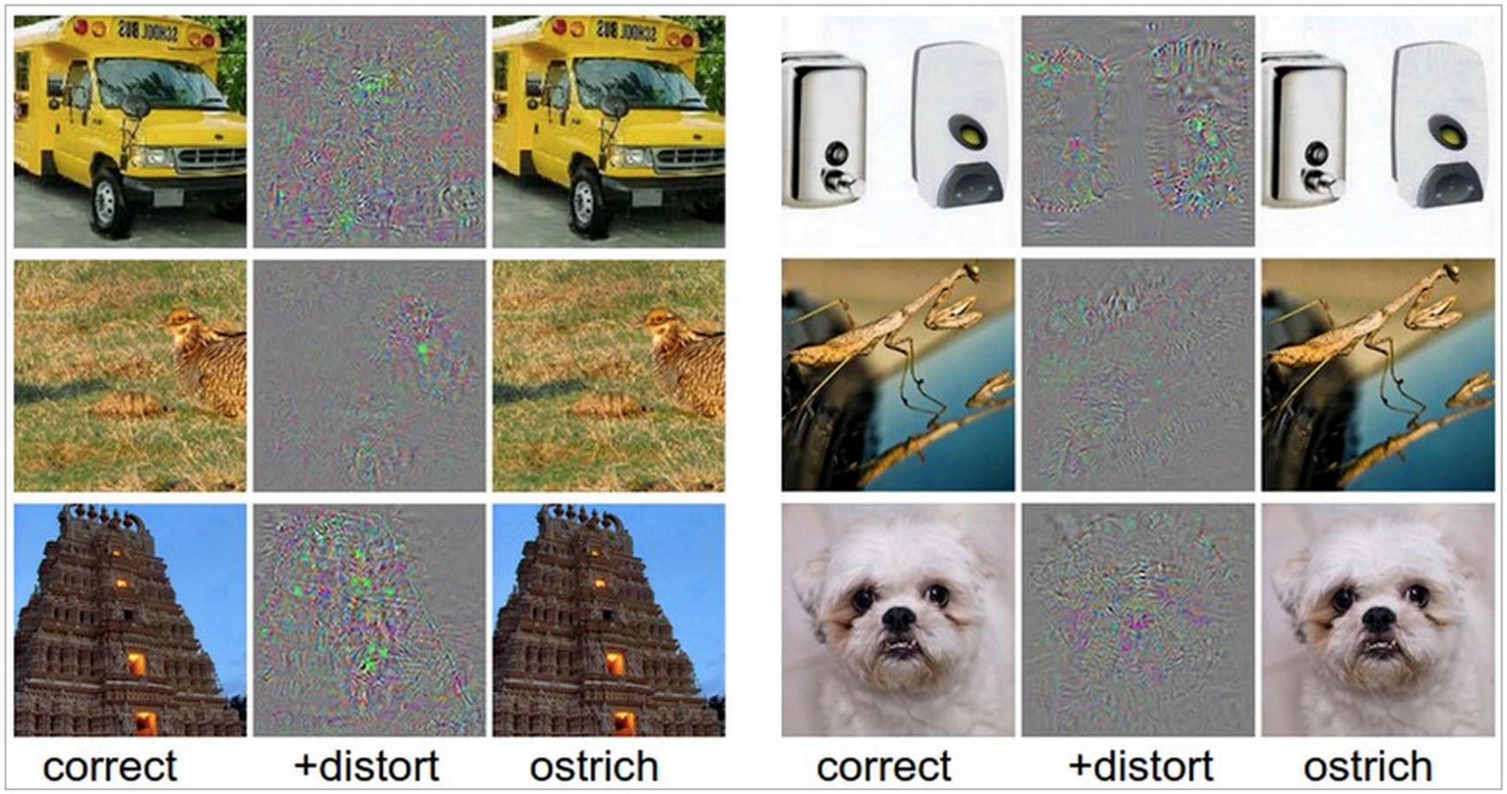
Team	Year	Place	Error (top-5)	External data
SuperVision – Toronto (7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
Human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

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

Outline

- Adaboost: learn the features, then learn the classifier
- Convolutional neural networks
- Adversarial inputs

Breaking CNNs

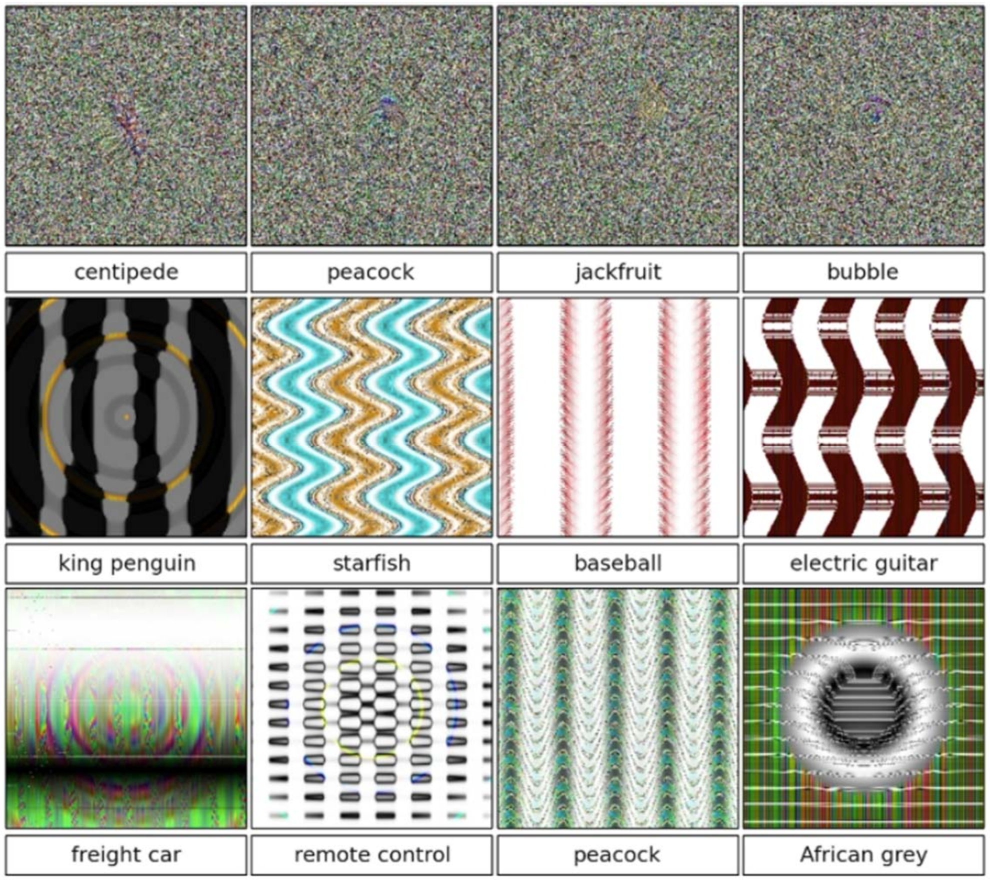


Take a correctly classified image (left image in both columns), and add a tiny distortion (middle) to fool the ConvNet with the resulting image (right).

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

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

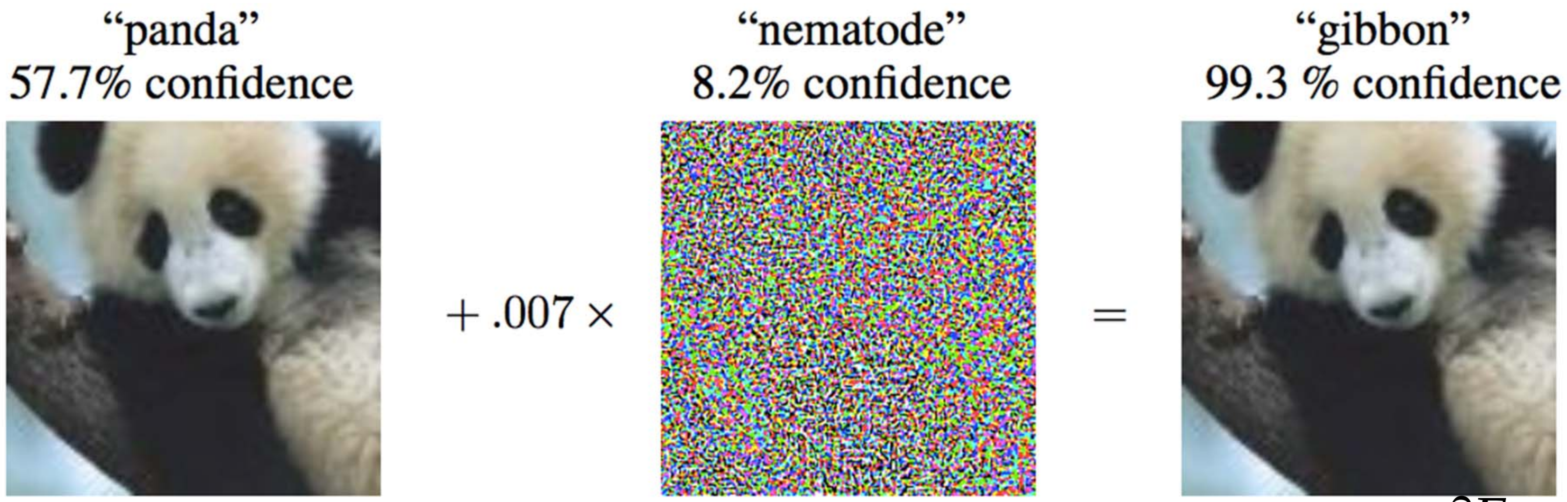
- Adversarial examples: modify the *image* to *increase* classifier error

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

http://arxiv.org/abs/1412.6572

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

What is going on?



x

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

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

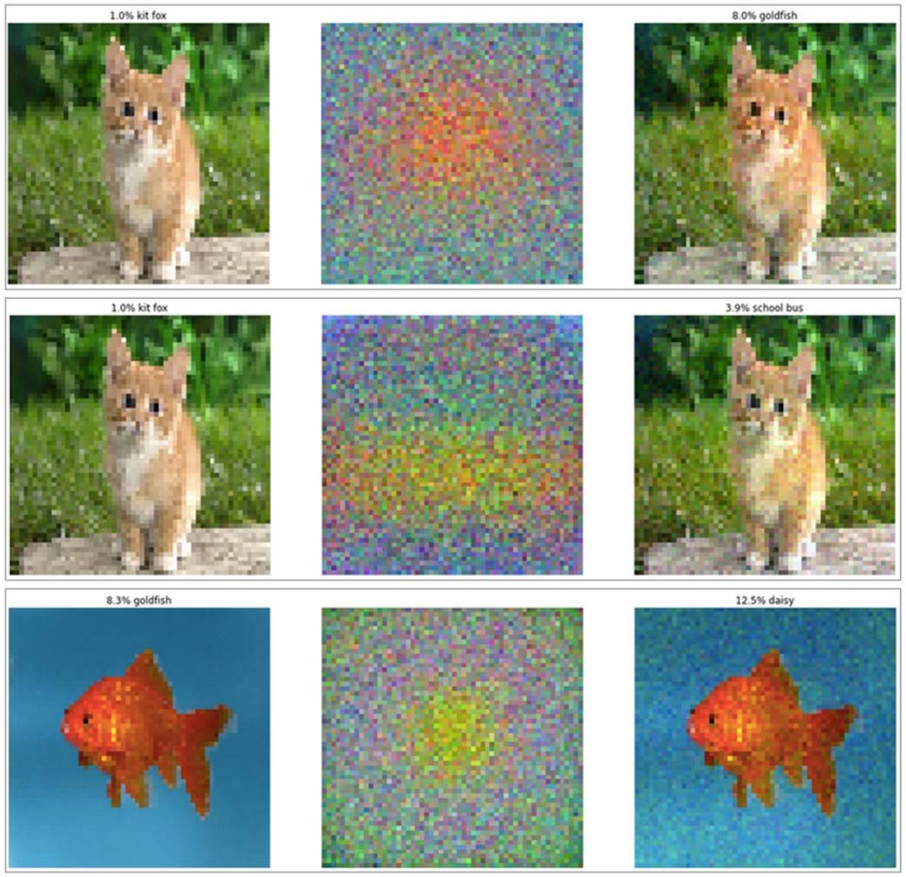
http://arxiv.org/abs/1412.6572

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/