MIT CSAIL

6.869: Advances in Computer Vision

Antonio Torralba, 2016

MIT COMPUTER VISION

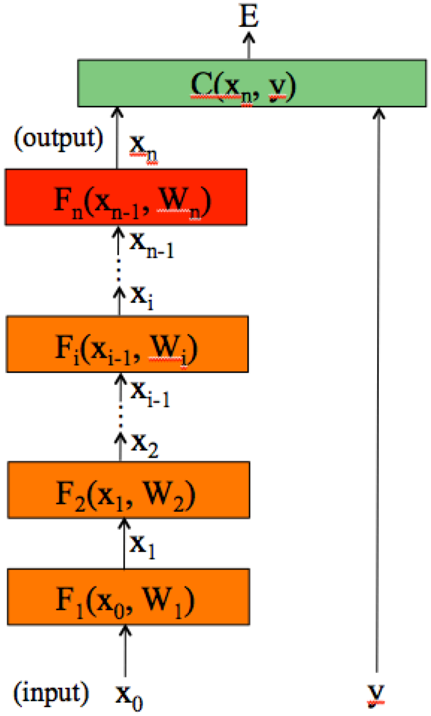
Lecture 7

Intro to convolutional neural networks

Computing gradients

To compute the gradients, we could start by wring the full energy E as a function of the network parameters.

$$E(\theta) = \sum_{m=1}^{M} C\left(F_n\left(F_{n-1}\left(F_2\left(F_1\left(x_0^m, w_1\right), w_2\right), w_{n-1}\right), w_n\right), y^m\right)$$



And then compute the partial derivatives… instead, we can use the chain rule to derive a compact algorithm: **back-propagation**

Computing gradients

The energy E is the sum of the costs associated to each training example x^m, y^m

$$E(\theta) = \sum_{m=1}^{M} C\left(x_n^m, y^m; \theta\right)$$

Computing gradients

The energy E is the sum of the costs associated to each training example x^m, y^m

$$E(\theta) = \sum_{m=1}^{M} C\left(x_n^m, y^m; \theta\right)$$

Its gradient with respect to each of the networks parameters θ_i is:

$$\frac{\partial E}{\partial \theta_i} = \sum_{m=1}^{M} \frac{C\left(x_n^m, y^m; \theta\right)}{\partial \theta_i}$$

is how much E varies when the parameter θ_i is varied.

Computing gradients

We could write the cost function to get the gradients:

$$C(x_n, y;\ \theta) = C(F_n(x_{n-1}, w_n), y)$$

with $\theta = [w_1, w_2, \cdots, w_n]$

If we compute the gradient with respect to the parameters of the last layer (output layer) w_n, using the chain rule:

$$\frac{\partial C}{\partial w_n} = \frac{\partial C}{\partial x_n} \cdot \frac{\partial x_n}{\partial w_n} = \frac{\partial C}{\partial x_n} \cdot \frac{\partial F_n(x_{n-1}, w_n)}{\partial w_n}$$

(how much the cost changes when we change w_n: is the product between how much the cost changes when we change the output of the last layer and how much the output changes when we change the layer parameters.)

Computing gradients: cost layer

If we compute the gradient with respect to the parameters of the last layer (output layer) w_n, using the chain rule:

$$\frac{\partial C}{\partial w_n} = \frac{\partial C}{\partial x_n} \cdot \frac{\partial x_n}{\partial w_n} = \frac{\partial C}{\partial x_n} \cdot \frac{\partial F_n\left(x_{n-1}, w_n\right)}{\partial w_n}$$

Will depend on the layer structure and non-linearity.

For example, for an Euclidean loss:

$$C(x_n, y) = \frac{1}{2}\left\|x_n - y\right\|^2$$

The gradient is:

$$\frac{\partial C}{\partial x_n} = x_n - y$$

Computing gradients: layer i

We could write the full cost function to get the gradients:

$$C(x_n, y;\ \theta) = C\left(F_n\left(F_{n-1}\left(F_2\left(F_1\left(x_0, w_1\right), w_2\right), w_{n-1}\right), w_n\right), y\right)$$

If we compute the gradient with respect to w_i, using the chain rule:

$$\frac{\partial C}{\partial w_i} = \underbrace{\frac{\partial C}{\partial x_n} \cdot \frac{\partial x_n}{\partial x_{n-1}} \cdot \frac{\partial x_{n-1}}{\partial x_{n-2}} \cdot \ldots \cdot \frac{\partial x_{i+1}}{\partial x_i}}_{\frac{\partial C}{\partial x_i}} \cdot \underset{\uparrow}{\frac{\partial x_i}{\partial w_i}}$$

$$\frac{\partial C}{\partial x_i}$$

And this can be computed iteratively!

$$\frac{\partial F_i\left(x_{i-1}, w_i\right)}{\partial w_i}$$

This is easy.

Backpropagation

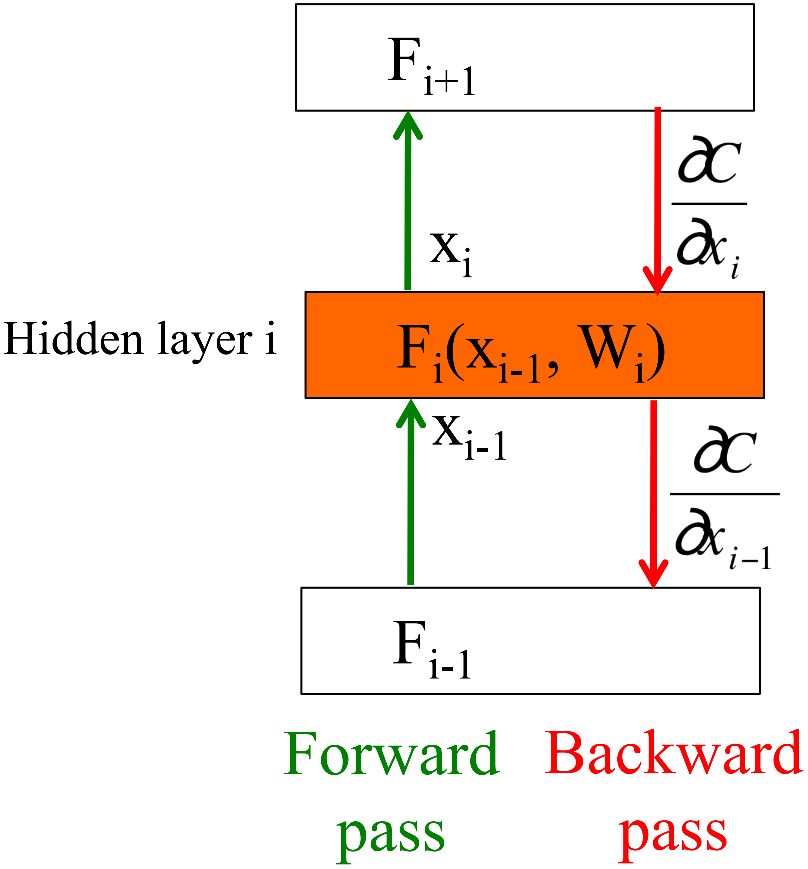
$$\frac{\partial C}{\partial w_i} = \underbrace{\frac{\partial C}{\partial x_n}\cdot\frac{\partial x_n}{\partial x_{n-1}}\cdot\frac{\partial x_{n-1}}{\partial x_{n-2}}\cdot\ldots\cdot\frac{\partial x_{i+1}}{\partial x_i}}_{\frac{\partial C}{\partial x_i}}\cdot\underset{\overset{\uparrow}{\frac{\partial F_i(x_{i-1},w_i)}{\partial w_i}}}{\frac{\partial x_i}{\partial w_i}}$$

If we have the value of $\frac{\partial C}{\partial x_i}$ we can compute the gradient at the layer bellow as:

$$\underset{\overset{\nearrow}{\text{Gradient layer i-1}}}{\frac{\partial C}{\partial x_{i-1}}} = \underset{\overset{\nearrow}{\text{Gradient layer i}}}{\frac{\partial C}{\partial x_i}}\cdot\underset{\overset{\nwarrow}{\frac{\partial F_i(x_{i-1},w_i)}{\partial x_{i-1}}}}{\frac{\partial x_i}{\partial x_{i-1}}}$$

Backpropagation

Goal: to update parameters of layer i



- Layer i has two inputs (during training)

$$x_{i-1} \qquad \frac{\partial C}{\partial x_i}$$

- For layer i, we need the derivatives:

$$\frac{\partial F_i(x_{i-1}, w_i)}{\partial x_{i-1}} \qquad \frac{\partial F_i(x_{i-1}, w_i)}{\partial w_i}$$

- We compute the outputs

$$x_i = F_i(x_{i-1}, w_i)$$

$$\frac{\partial C}{\partial x_{i-1}} = \frac{\partial C}{\partial x_i} \cdot \frac{\partial F_i(x_{i-1}, w_i)}{\partial x_{i-1}}$$

- The weight update equation is:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial x_i} \cdot \frac{\partial F_i(x_{i-1}, w_i)}{\partial w_i}$$

$$\boxed{w_i^{k+1} \leftarrow w_i^k + \eta \frac{\partial E}{\partial w_i}}$$

(sum over all training examples to get E)

Backpropagation: summary

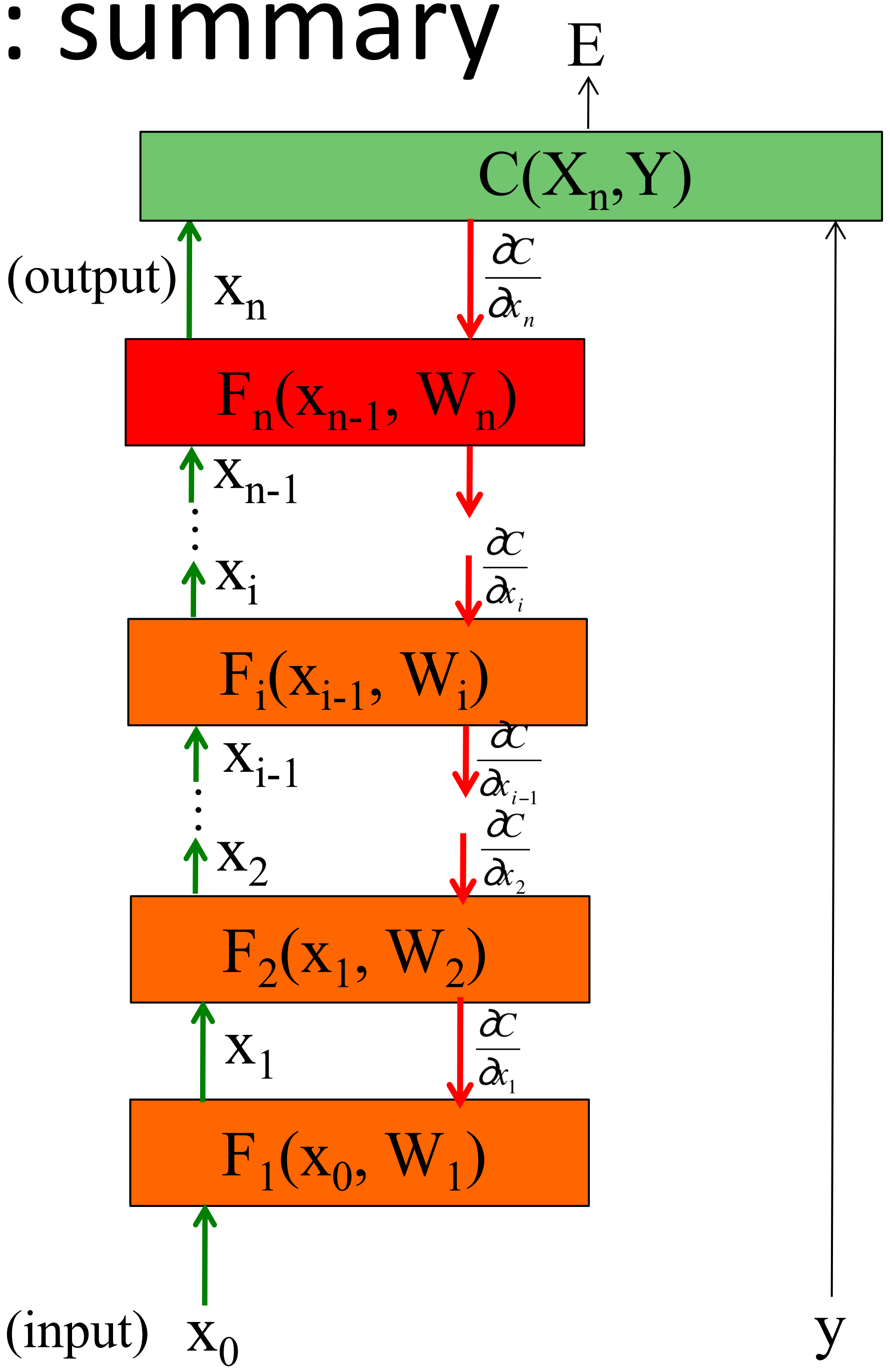
- Forward pass: for each training example. Compute the outputs for all layers

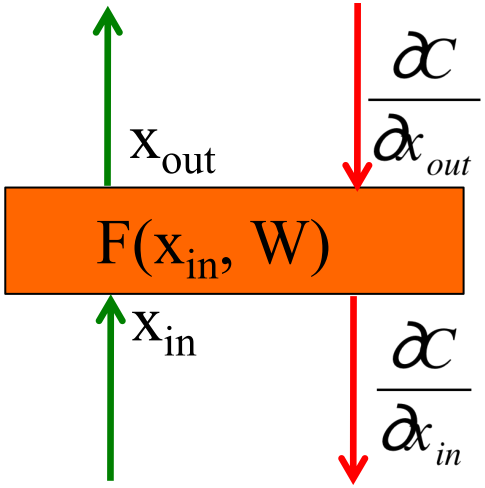
$$x_i = F_i(x_{i-1}, w_i)$$

- Backwards pass: compute cost derivatives iteratively from top to bottom:

$$\frac{\partial C}{\partial x_{i-1}} = \frac{\partial C}{\partial x_i} \cdot \frac{\partial F_i(x_{i-1}, w_i)}{\partial x_{i-1}}$$

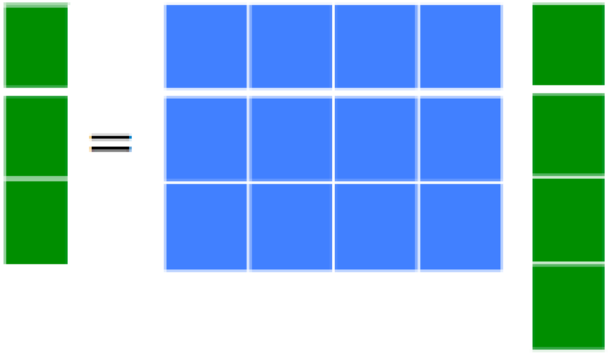
- Compute gradients and update weights.


E
$C(X_n, Y)$
(output)
x_n
$\frac{\partial C}{\partial x_n}$
$F_n(x_{n-1}, W_n)$
x_{n-1}
x_i
$\frac{\partial C}{\partial x_i}$
$F_i(x_{i-1}, W_i)$
x_{i-1}
$\frac{\partial C}{\partial x_{i-1}}$
x_2
$\frac{\partial C}{\partial x_2}$
$F_2(x_1, W_2)$
x_1
$\frac{\partial C}{\partial x_1}$
$F_1(x_0, W_1)$
(input)
x_0
y


x_{out}
$\frac{\partial C}{\partial x_{out}}$
$F(x_{in}, W)$
x_{in}
$\frac{\partial C}{\partial x_{in}}$

Linear Module

- Forward propagation: $x_{out} = F(x_{in}, W) = Wx_{in}$



With W being a matrix of size $|x_{out}| \times |x_{in}|$

- Backprop to input:

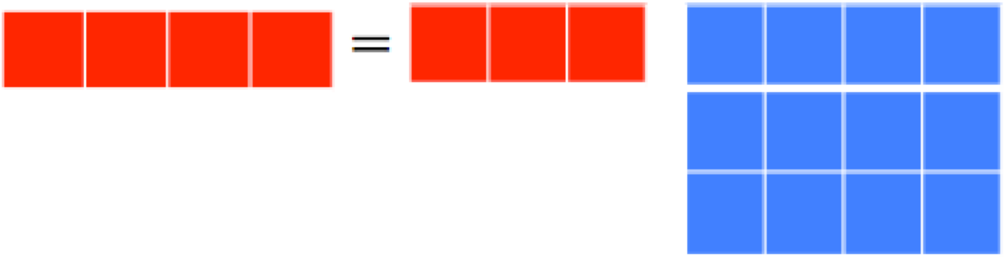
$$\frac{\partial C}{\partial x_{in}} = \frac{\partial C}{\partial x_{out}} \cdot \frac{\partial F(x_{in}, W)}{\partial x_{in}} = \frac{\partial C}{\partial x_{out}} \cdot \frac{\partial x_{out}}{\partial x_{in}}$$

If we look at the j component of output x_{out}, with respect to the i component of the input, x_{in}:

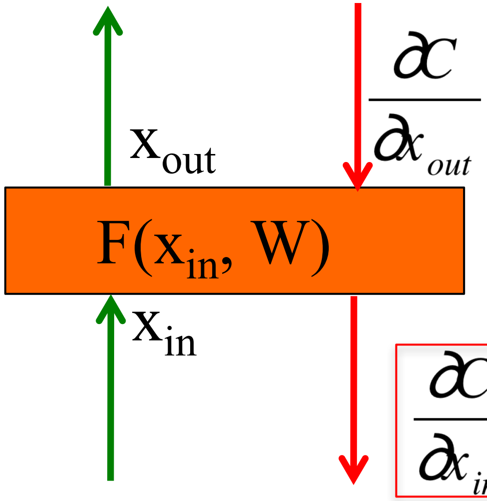
$$\frac{\partial x_{out_i}}{\partial x_{in_j}} = W_{ij} \quad \longrightarrow \quad \frac{\partial F(x_{in}, W)}{\partial x_{in}} = W$$

Therefore:

$$\boxed{\frac{\partial C}{\partial x_{in}} = \frac{\partial C}{\partial x_{out}} \cdot W}$$

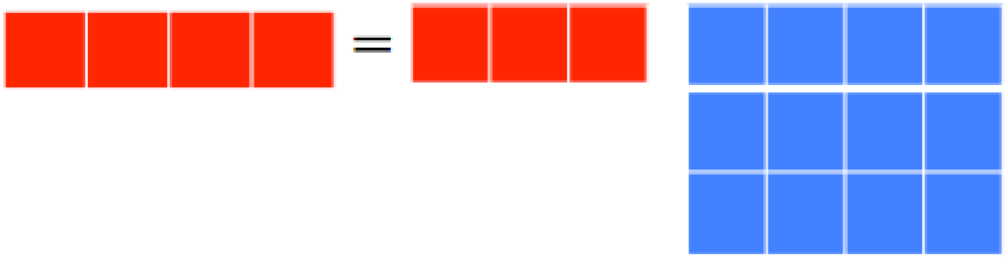


Linear Module

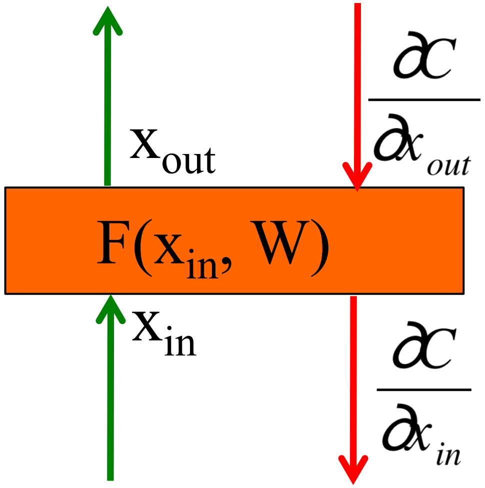


- Forward propagation: $x_{out} = F(x_{in}, W) = Wx_{in}$
- Backprop to input:

$$\boxed{\frac{\partial C}{\partial x_{in}} = \frac{\partial C}{\partial x_{out}} \cdot W}$$



Now let’s see how we use the set of outputs to compute the weights update equation (backprop to the weights).


x_{out}
$\frac{\partial C}{\partial x_{out}}$
F(x_{in}, W)
x_{in}
$\frac{\partial C}{\partial x_{in}}$

Linear Module

- Forward propagation: $x_{out} = F(x_{in}, W) = W x_{in}$
- Backprop to weights:

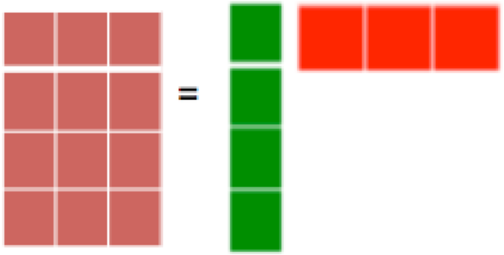
$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial x_{out}} \cdot \frac{\partial F(x_{in}, W)}{\partial W} = \frac{\partial C}{\partial x_{out}} \cdot \frac{\partial x_{out}}{\partial W}$$

If we look at how the parameter W_{ij} changes the cost, only the i component of the output will change, therefore:

$$\frac{\partial C}{\partial W_{ij}} = \frac{\partial C}{\partial x_{out_i}} \cdot \frac{\partial x_{out_i}}{\partial W_{ij}} = \frac{\partial C}{\partial x_{out_i}} \cdot x_{in_j}$$

$$\frac{\partial x_{out_i}}{\partial W_{ij}} = x_{in_j}$$

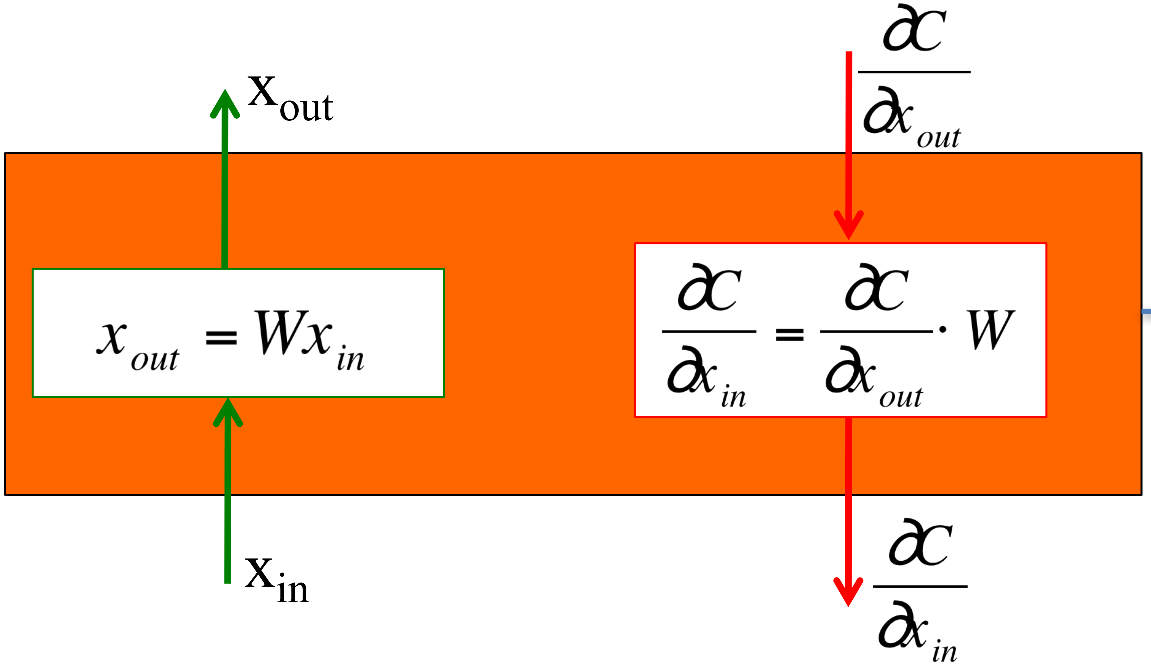
$$\boxed{\frac{\partial C}{\partial W} = x_{in} \cdot \frac{\partial C}{\partial x_{out}}}$$



And now we can update the weights (by summing over all the training examples):

$$\boxed{W_{ij}^{k+1} \leftarrow W_{ij}^{k} + \eta \frac{\partial E}{\partial W_{ij}} \quad \text{(sum over all training examples to get E)}}$$

Linear Module



x_{out}

$$x_{out} = Wx_{in}$$

x_{in}

$$\frac{\partial C}{\partial x_{out}}$$

$$\frac{\partial C}{\partial x_{in}} = \frac{\partial C}{\partial x_{out}} \cdot W$$

$$\frac{\partial C}{\partial x_{in}}$$

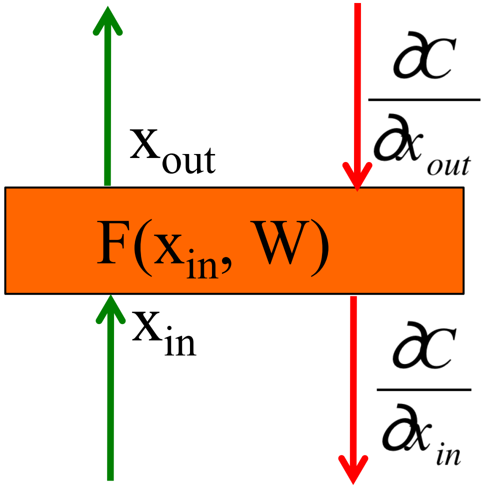
$$\frac{\partial C}{\partial W} = x_{in} \cdot \frac{\partial C}{\partial x_{out}}$$

Weight updates

$$W^{k+1} \leftarrow W^{k} + \eta \left(\frac{\partial E}{\partial W} \right)^{T}$$

$$E = \sum_{m=1}^{M} C\left(x_n^m, y^m\right) \longrightarrow \frac{\partial E}{\partial W} = \sum_{m=1}^{M} x_{in} \cdot \frac{\partial C}{\partial x_{out}} \bigg|_{\left(x_0^m, y^m\right)}$$

Sum over M training pairs



x_{out}
$\frac{\partial C}{\partial x_{out}}$
$F(x_{in}, W)$
x_{in}
$\frac{\partial C}{\partial x_{in}}$

Pointwise function

- Forward propagation:

$$x_{out_i} = h(x_{in_i} + b_i)$$

h = an arbitrary function, b_i is a bias term.

- Backprop to input: $\frac{\partial C}{\partial x_{in_i}} = \frac{\partial C}{\partial x_{out_i}} \cdot \frac{\partial x_{out_i}}{\partial x_{in_i}} = \frac{\partial C}{\partial x_{out_i}} \cdot h'(x_{in_i} + b_i)$

- Backprop to bias: $\frac{\partial C}{\partial b_i} = \frac{\partial C}{\partial x_{out_i}} \cdot \frac{\partial x_{out_i}}{\partial b_i} = \frac{\partial C}{\partial x_{out_i}} \cdot h'(x_{in_i} + b_i)$

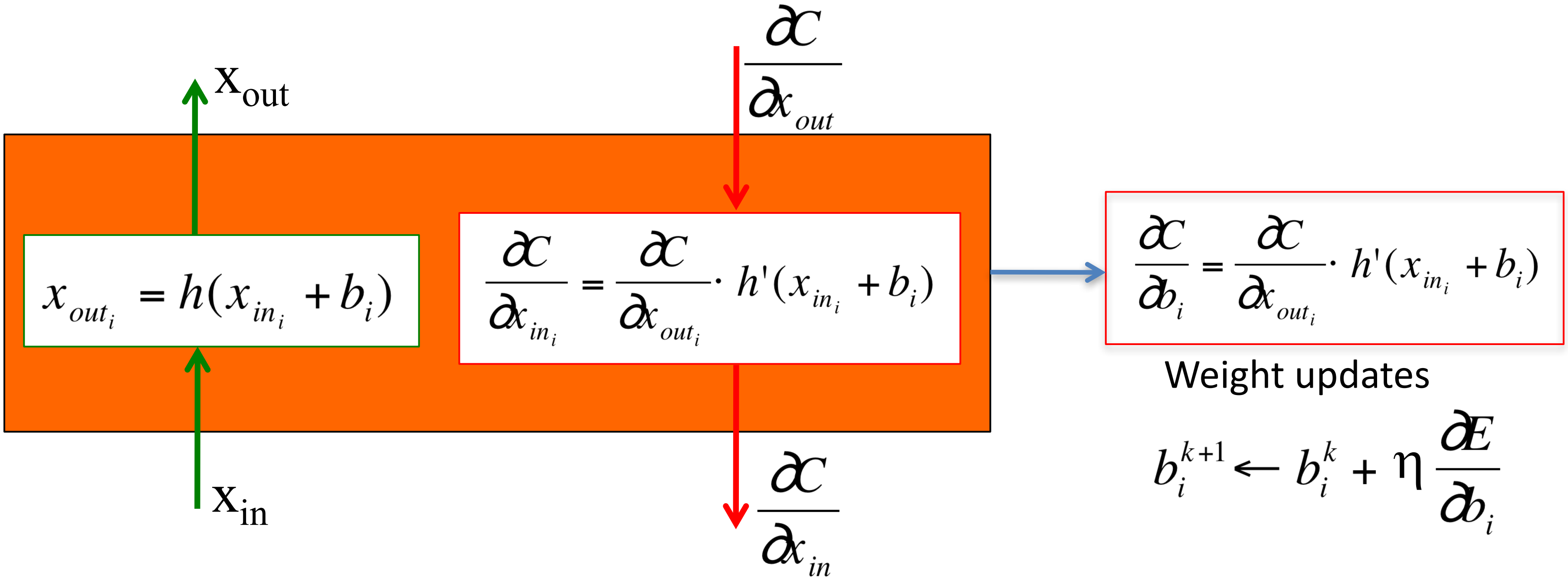
We use this last expression to update the bias.

Some useful derivatives:

For hyperbolic tangent: $\tanh'(x) = 1 - \tanh^2(x)$

For ReLU: h(x) = max(0,x) h'(x) = 1 [x>0]

Pointwise function



$\mathrm{x}_{\mathrm{out}}$

$$\frac{\partial C}{\partial x_{out}}$$

$$x_{out_i} = h(x_{in_i} + b_i)$$

$$\frac{\partial C}{\partial x_{in_i}} = \frac{\partial C}{\partial x_{out_i}} \cdot h'(x_{in_i} + b_i)$$

$$\frac{\partial C}{\partial b_i} = \frac{\partial C}{\partial x_{out_i}} \cdot h'(x_{in_i} + b_i)$$

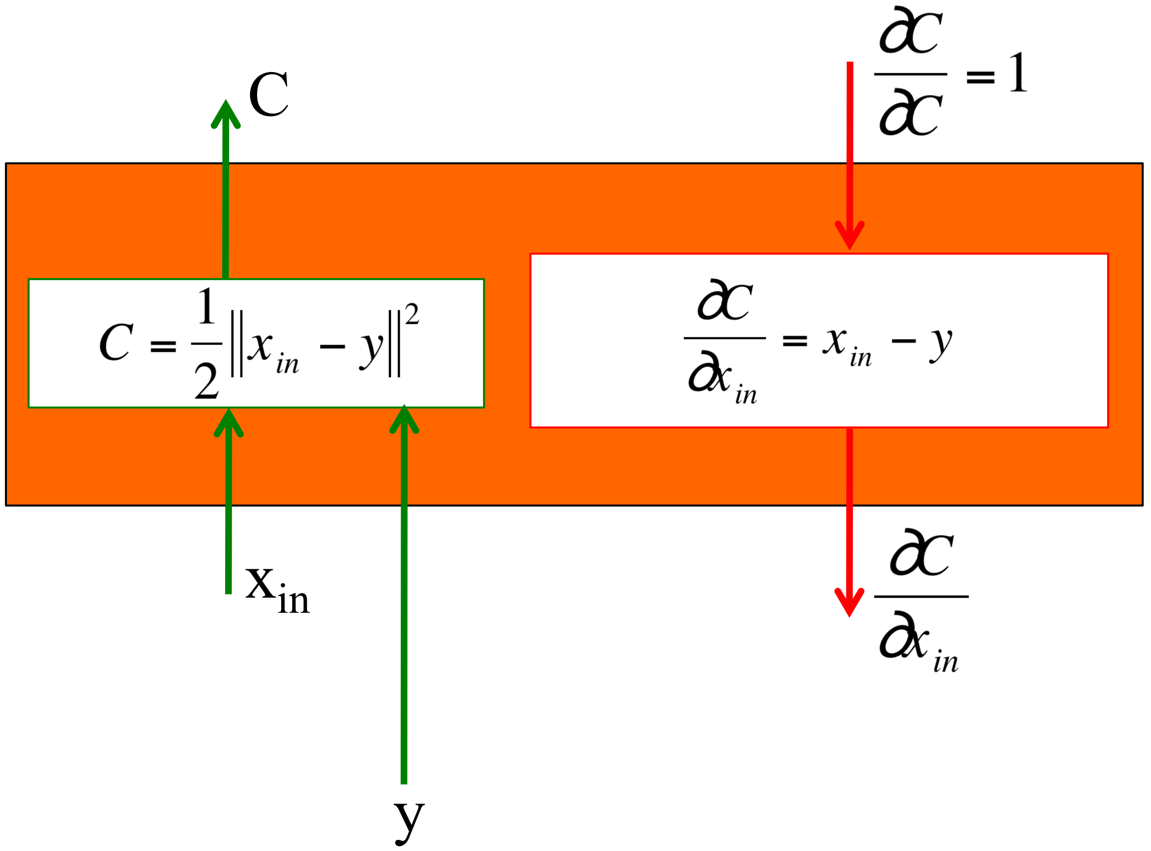
x_{in}

$$\frac{\partial C}{\partial x_{in}}$$

Weight updates

$$b_i^{k+1} \leftarrow b_i^k + \eta \frac{\partial E}{\partial b_i}$$

Euclidean cost module



C

$\frac{\partial C}{\partial C} = 1$

$C = \frac{1}{2}\|x_{in} - y\|^2$

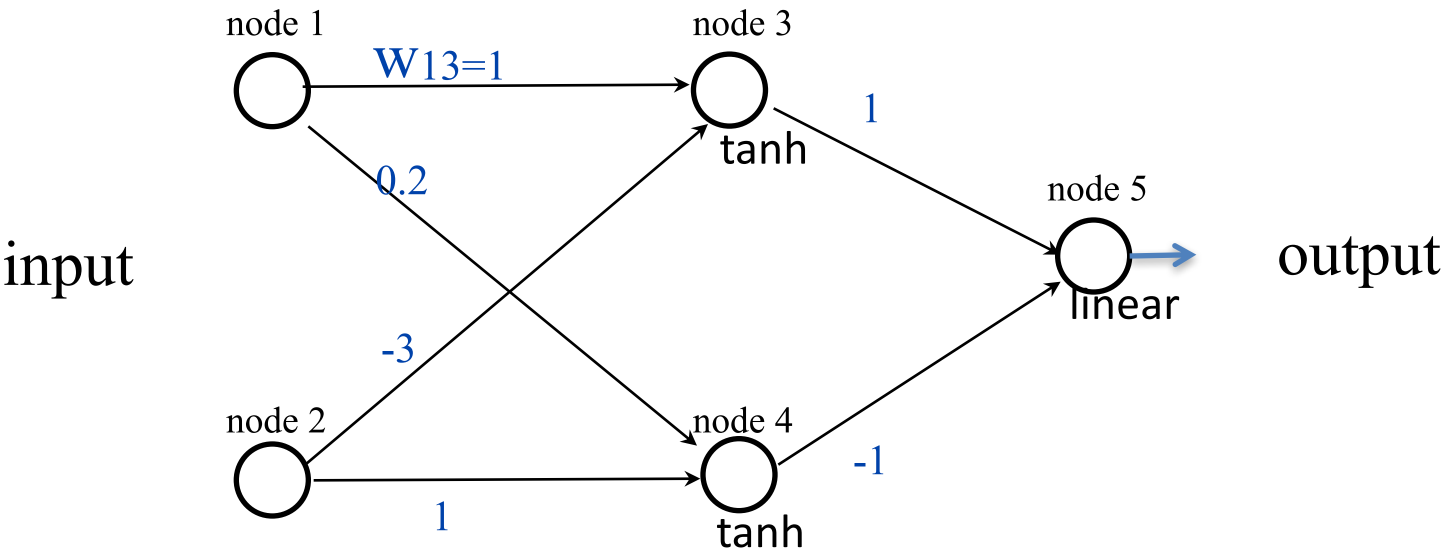
$\frac{\partial C}{\partial x_{in}} = x_{in} - y$

x_{in}

y

$\frac{\partial C}{\partial x_{in}}$

Back propagation example



Learning rate η = -0.2 (because we used positive increments)

Euclidean loss

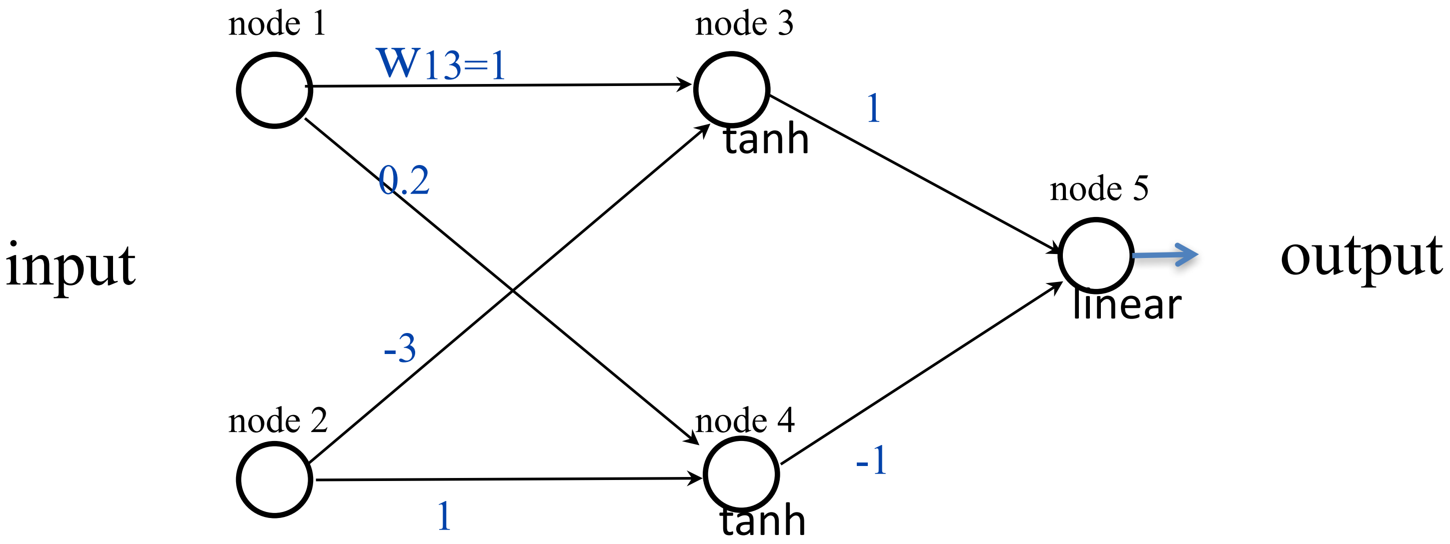
Training data:

	input		desired output
	node 1	node 2	node 5
	1.0	0.1	0.5

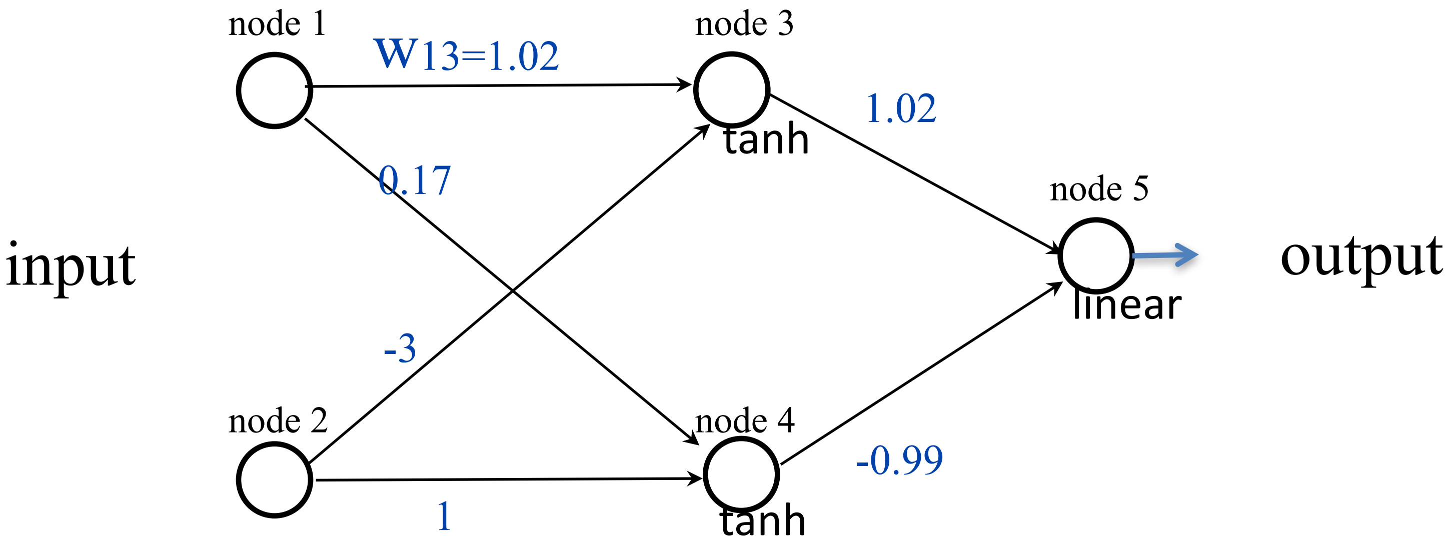
Exercise: run one iteration of back propagation

- How many boxes? 1, 2, 3, 4, 5?
- Let’s compute the forward pass
- Let’s compute the backwards pass
- Let’s update the weights!

Back propagation example



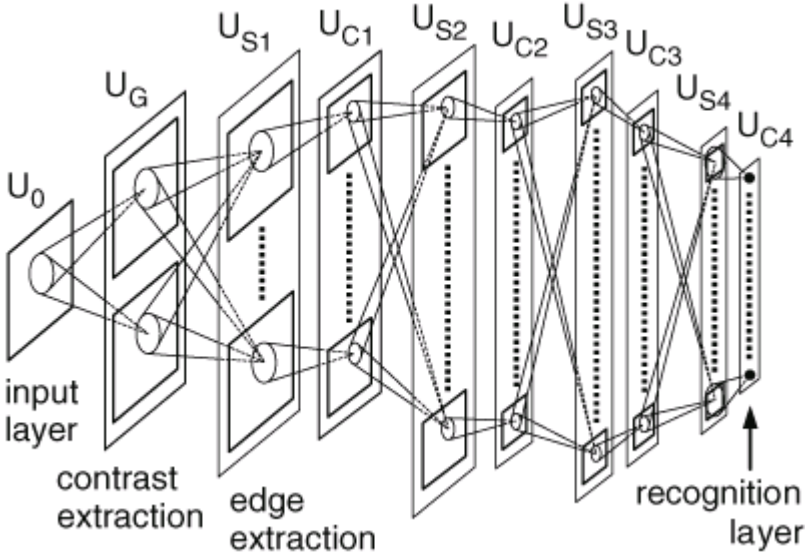
After one iteration (rounding to two digits):



Dealing with images

Neocognitron

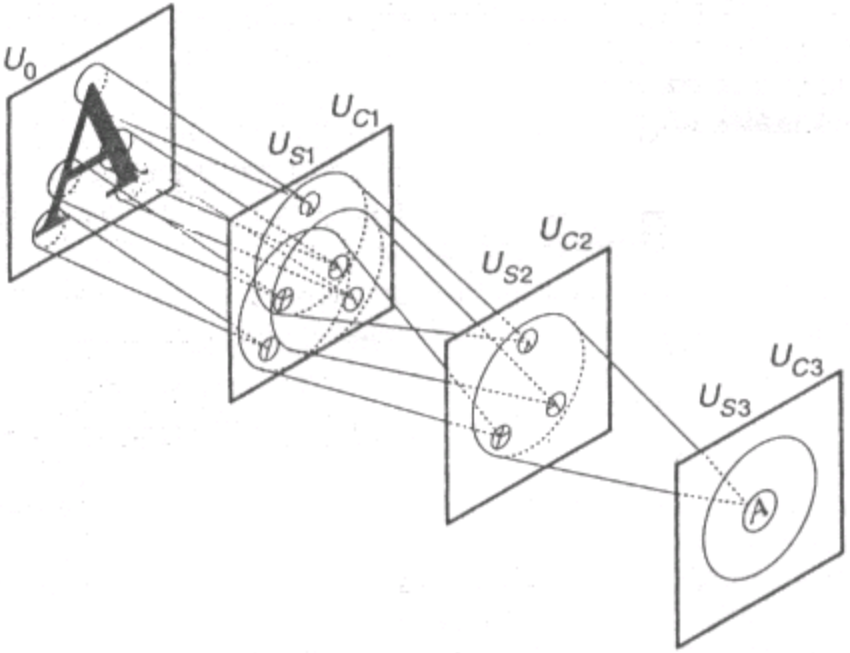
Fukushima (1980). Hierarchical multilayered neural network

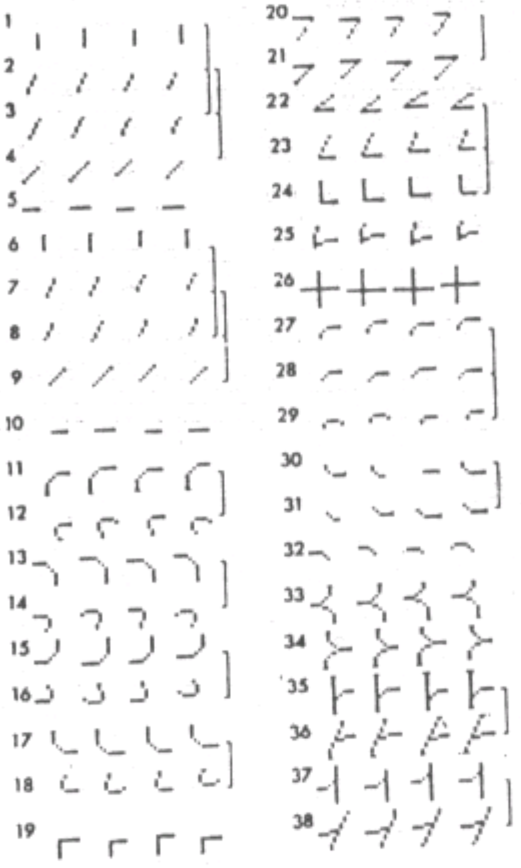


S-cells work as feature-extracting cells. They resemble simple cells of the primary visual cortex in their response.

C-cells, which resembles complex cells in the visual cortex, are inserted in the network to allow for positional errors in the features of the stimulus. The input connections of C-cells, which come from S-cells of the preceding layer, are fixed and invariable. Each C-cell receives excitatory input connections from a group of S-cells that extract the same feature, but from slightly different positions. The C-cell responds if at least one of these S-cells yield an output.

Neocognitron





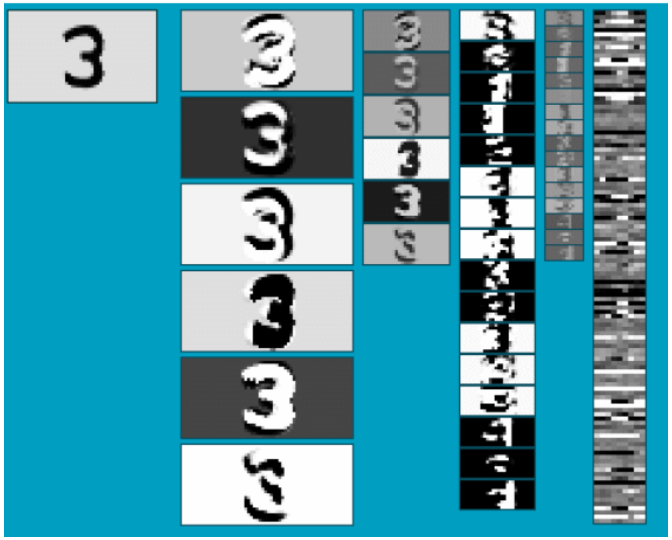
Learning is done greedily for each layer

Multistage Hubel-Wiesel Architecture

- Stack multiple stages of simple cells / complex cells layers
- Higher stages compute more global, more invariant features
- Classification layer on top

History:

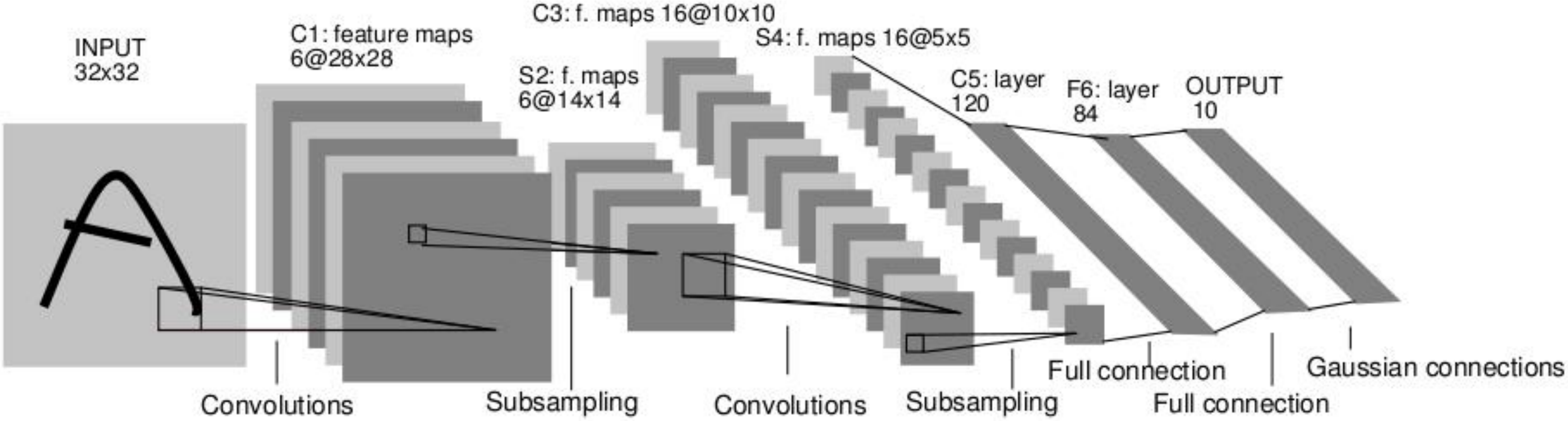
- Neocognitron [Fukushima 1971-1982]
- Convolutional Nets [LeCun 1988-2007]
- HMAX [Poggio 2002-2006]
- Many others….



Slide: Y. LeCun

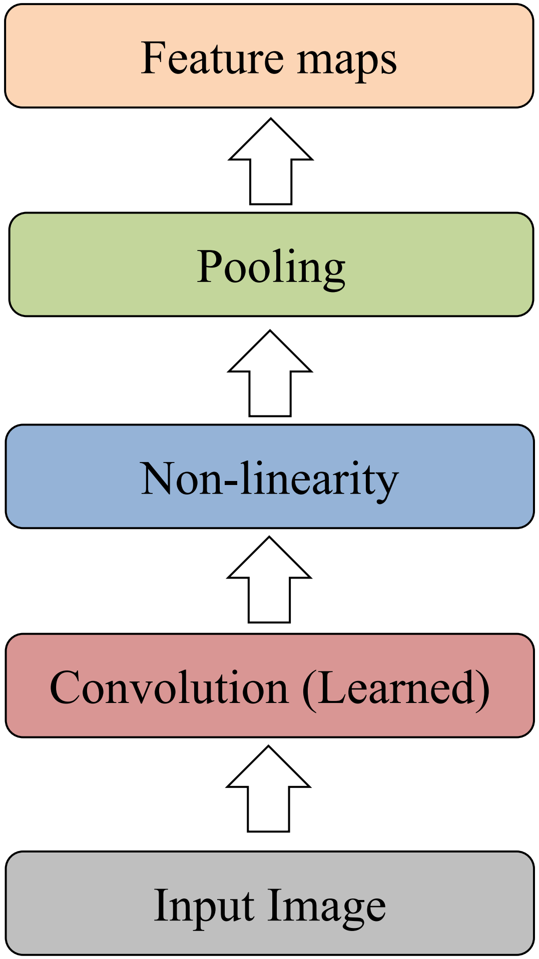
Convolutional Neural Networks

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



Overview of Convnets

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



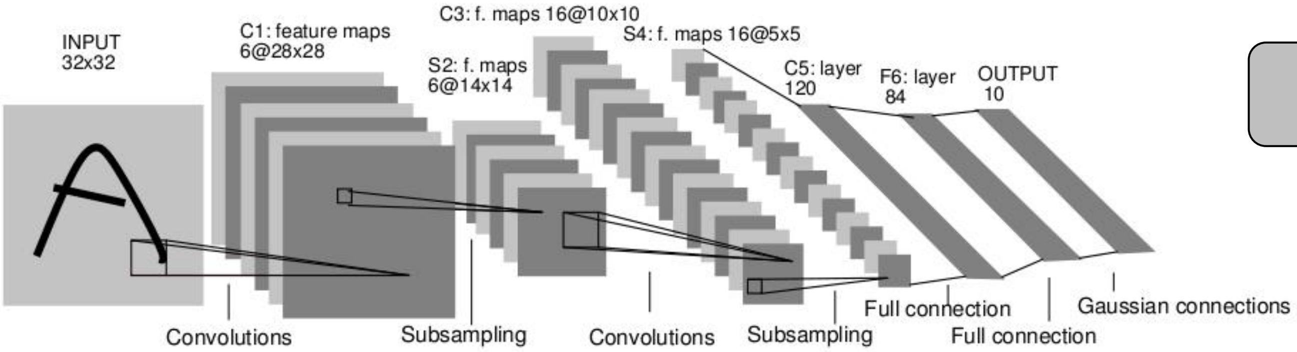
Feature maps

Pooling

Non-linearity

Convolution (Learned)

Input Image

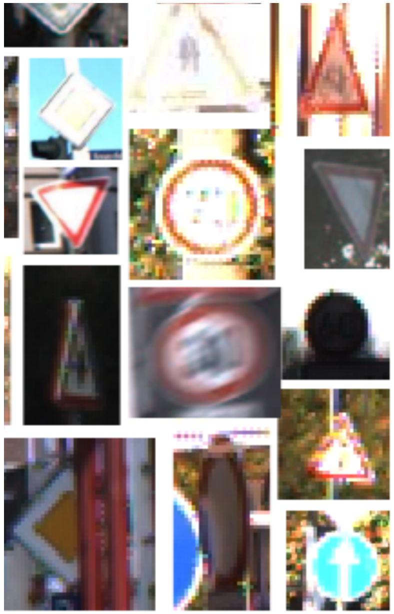


LeCun et al. 1998

Convnet Successes

- Handwritten text/digits
  - MNIST (0.17% error [Ciresan et al. 2011])
  - Arabic & Chinese [Ciresan et al. 2012]

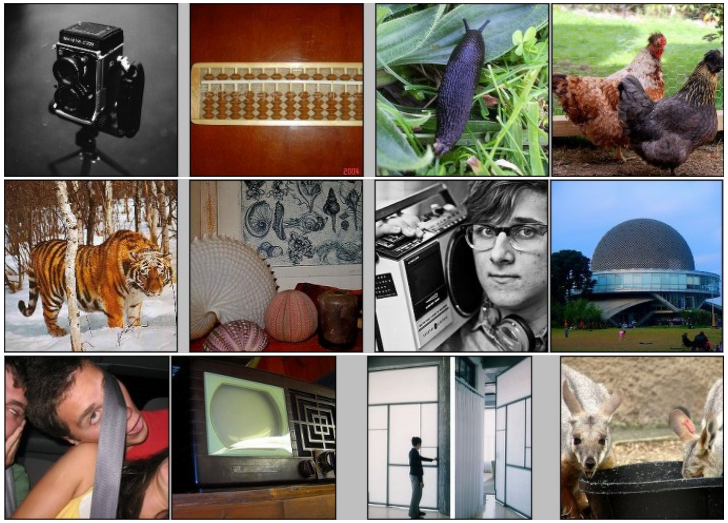
- Simpler recognition benchmarks
  - CIFAR-10 (9.3% error [Wan et al. 2013])
  - Traffic sign recognition
    - 0.56% error vs 1.16% for humans [Ciresan et al. 2011]



- But less good at more complex datasets
  - E.g. Caltech-101/256 (few training examples)

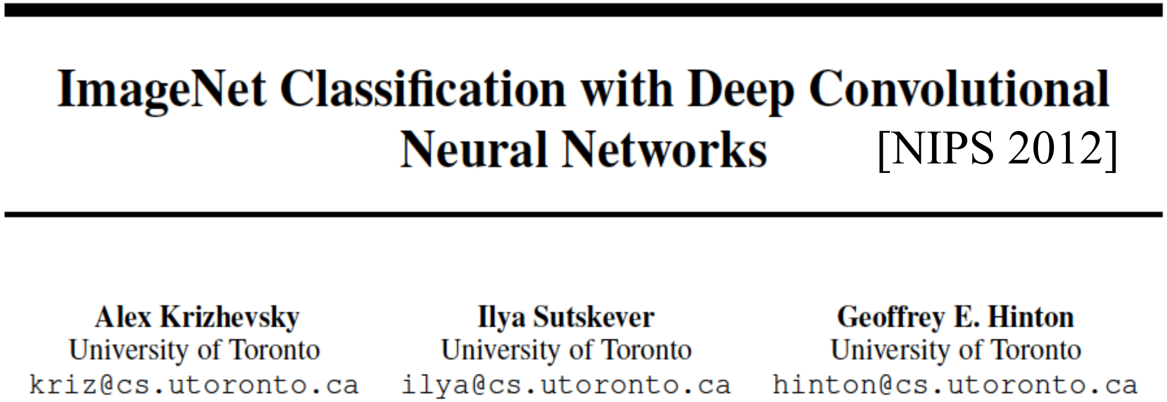
Application to ImageNet

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]

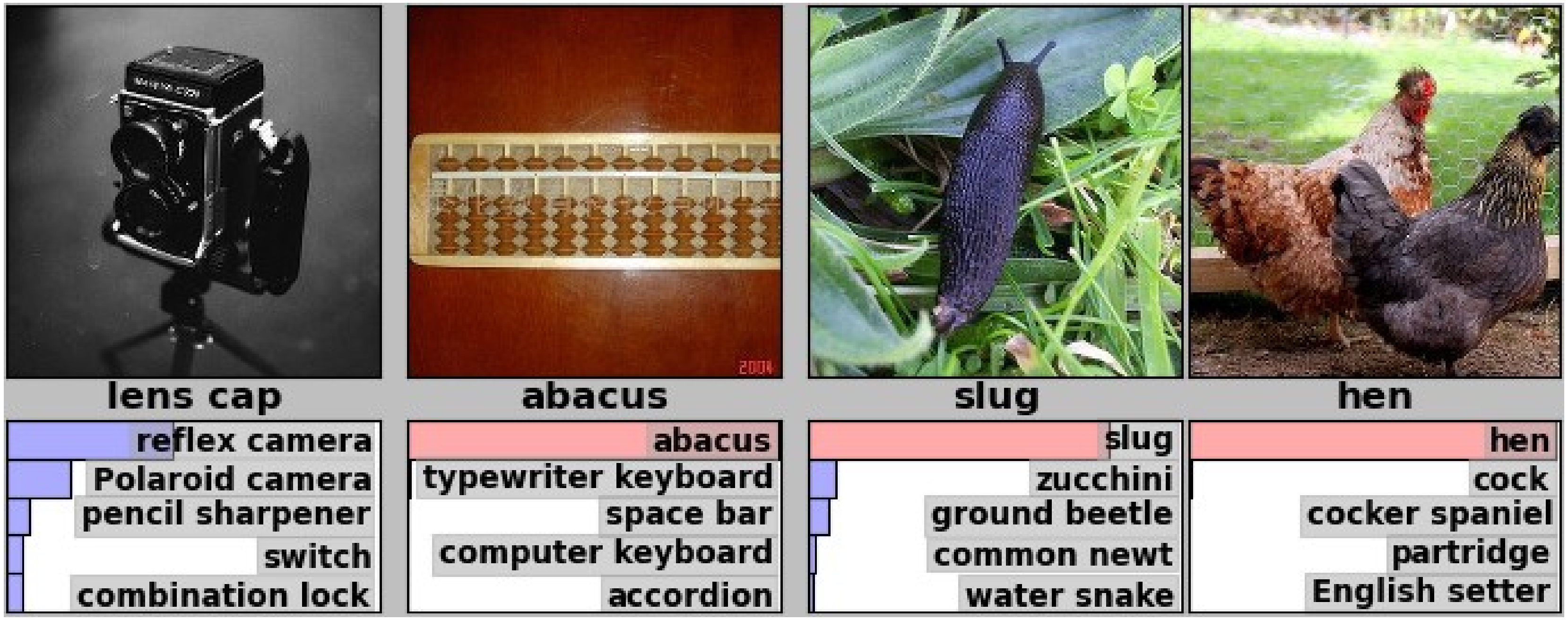
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Goal

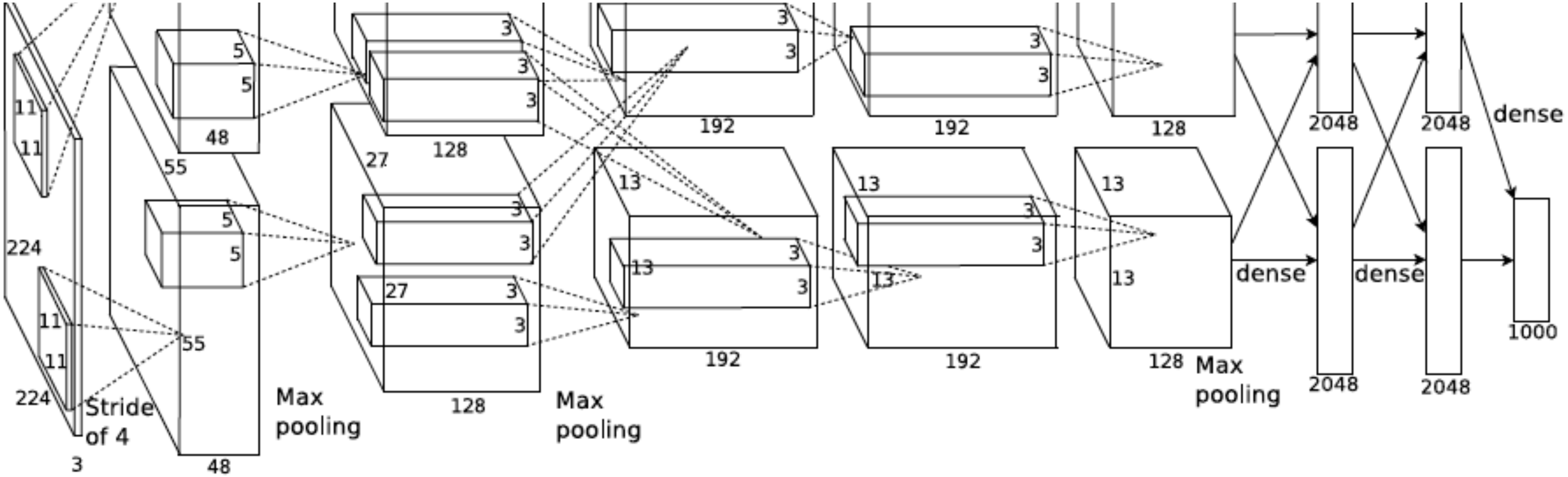
- Image Recognition
  - Pixels → Class Label



[Krizhevsky et al. NIPS 2012]

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

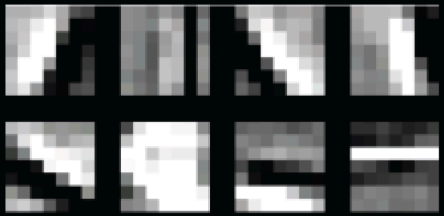
How convnets work

- Operations in each layer
- Architecture
- Training
- Results

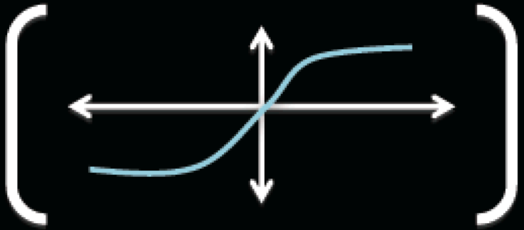
Components of Each Layer

Pixels / Features

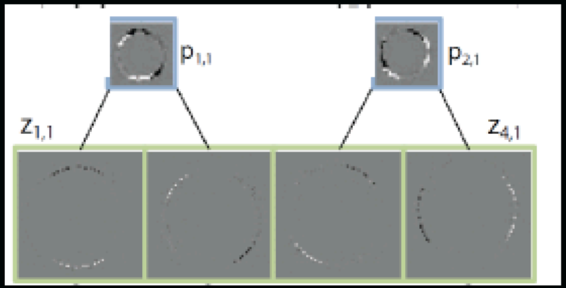
Filter with learned dictionary



Non-linearity



Spatial local max pooling



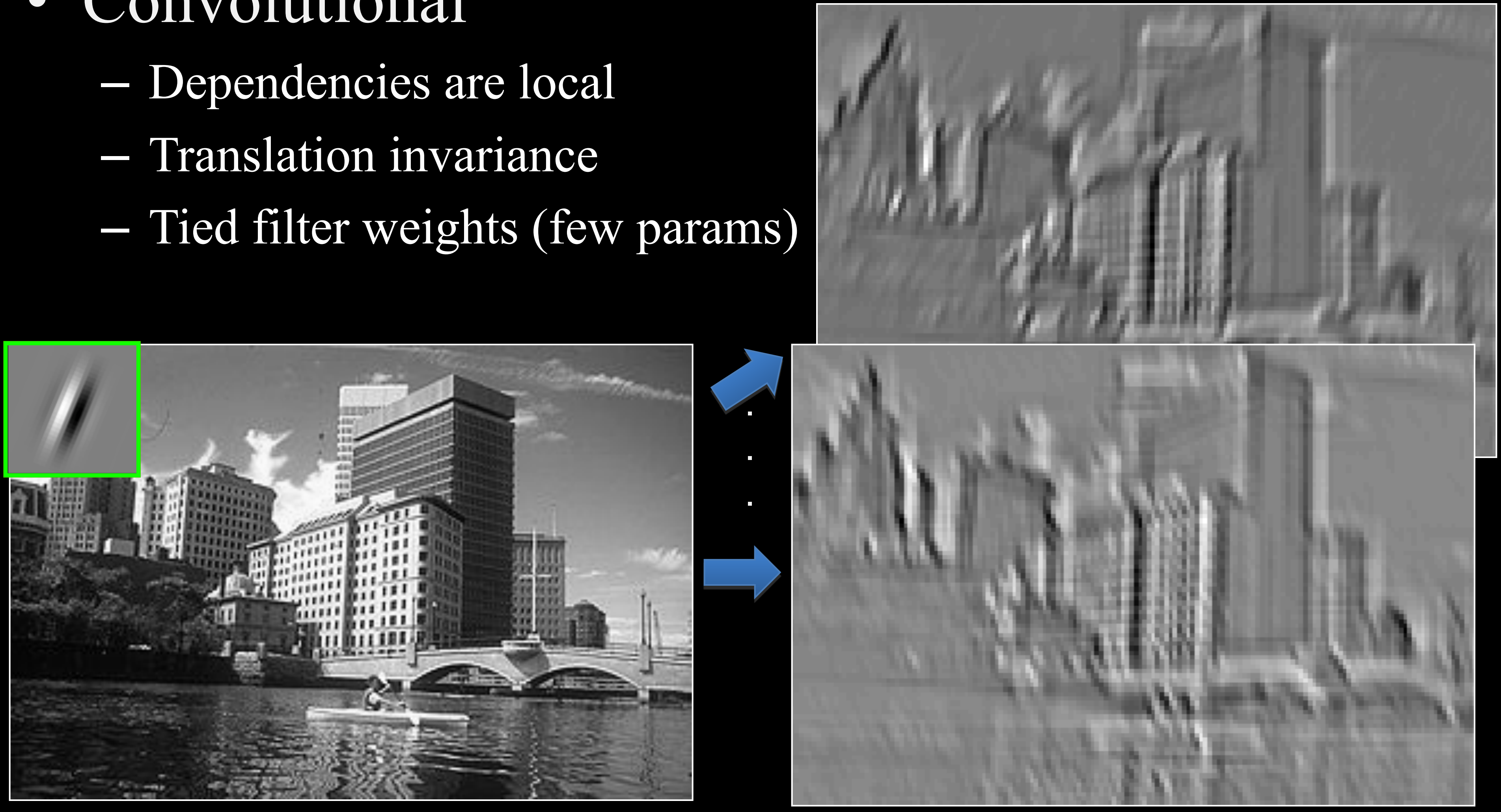
[Optional] Normalization across data/features

Output Features

Slide: R. Fergus

Filtering

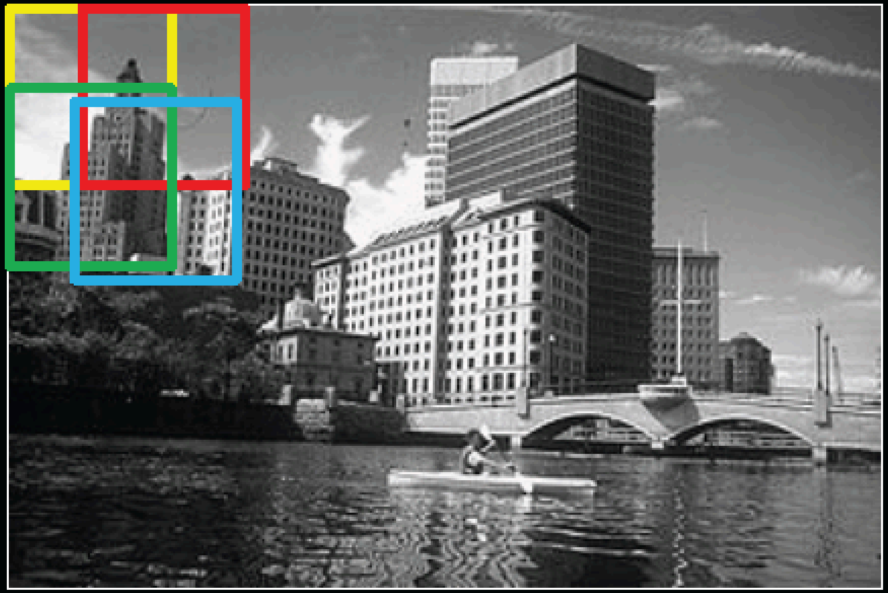
- Convolutional
  - Dependencies are local
  - Translation invariance
  - Tied filter weights (few params)



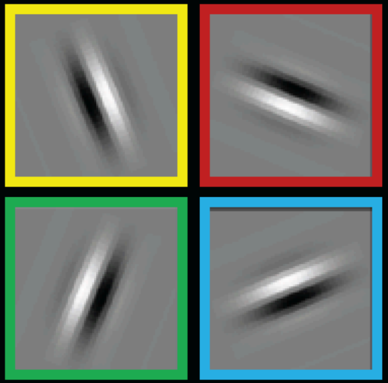
Slide: R. Fergus

Filtering

- Local
  - Each unit layer above look at local window
  - But no weight tying

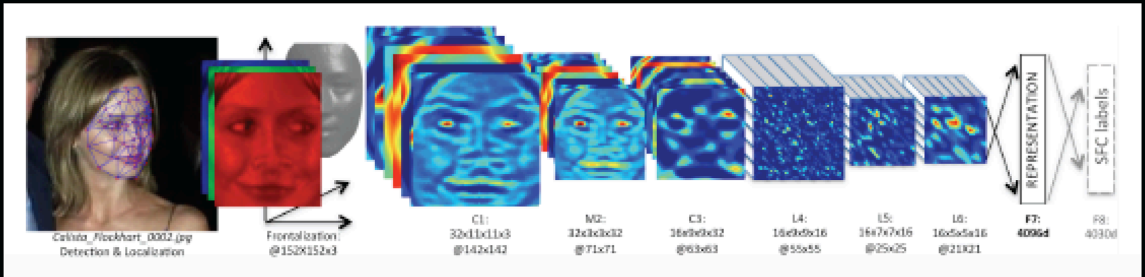


Input

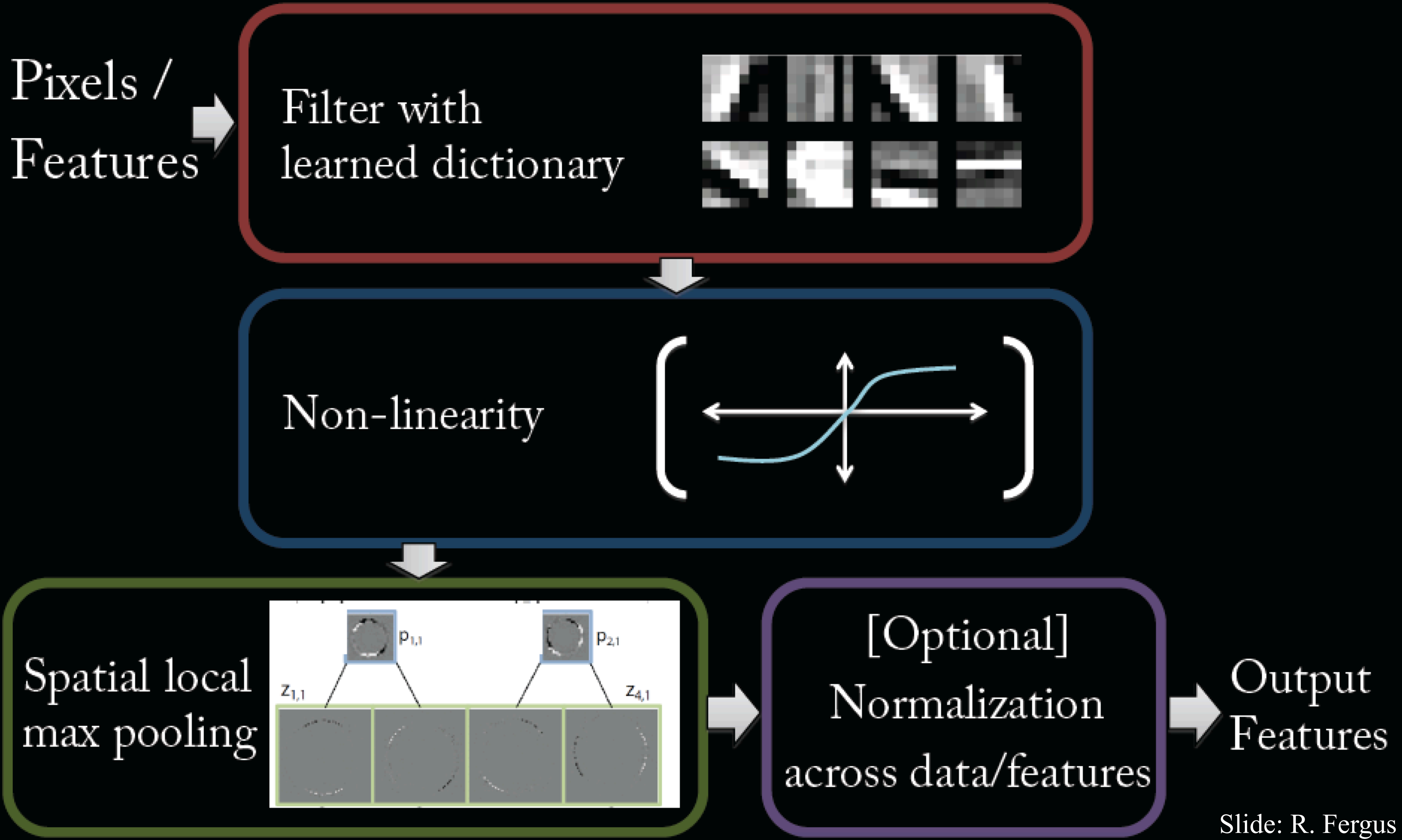


Filters

- E.g. face recognition



Components of Each Layer



Pixels / Features

Filter with learned dictionary

Non-linearity

Spatial local max pooling

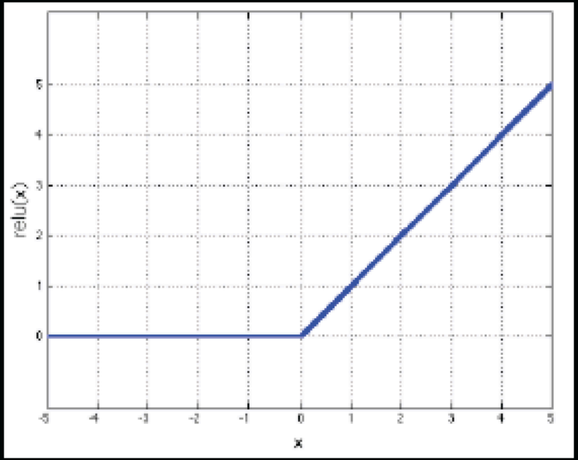
[Optional] Normalization across data/features

Output Features

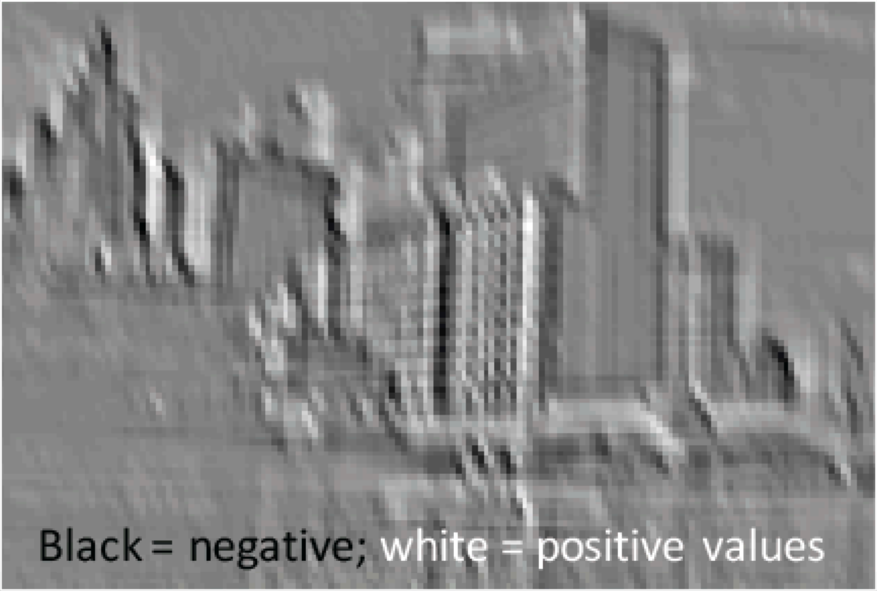
Slide: R. Fergus

Non-Linearity

- Rectified linear function
  - Applied per-pixel
  - output = max(0,input)

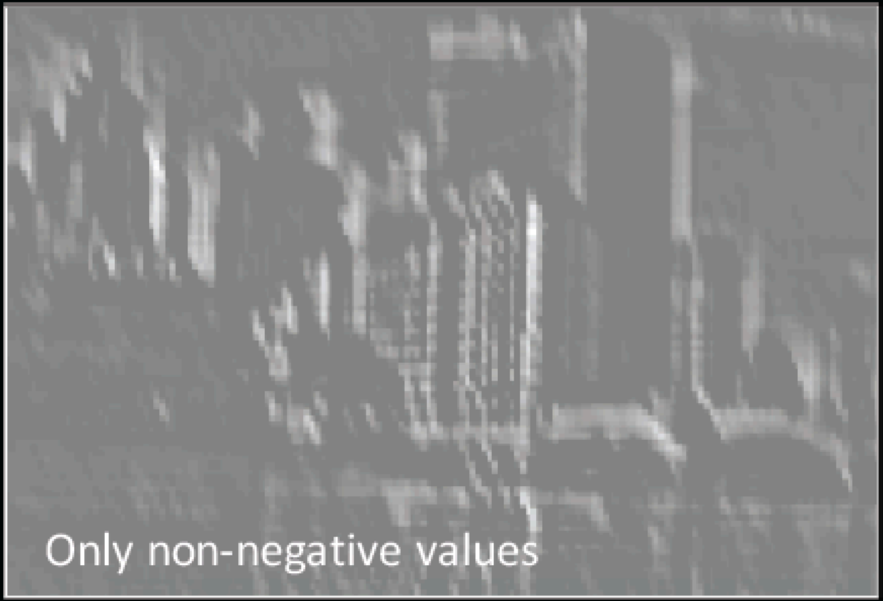


Input feature map



Black = negative; white = positive values

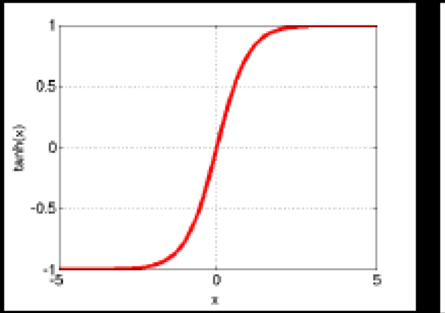
Output feature map

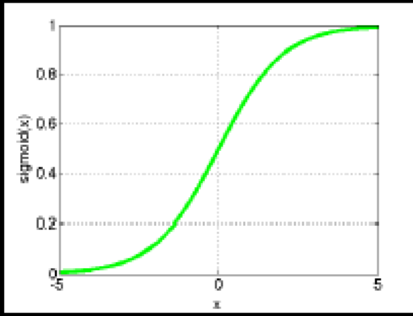


Only non-negative values

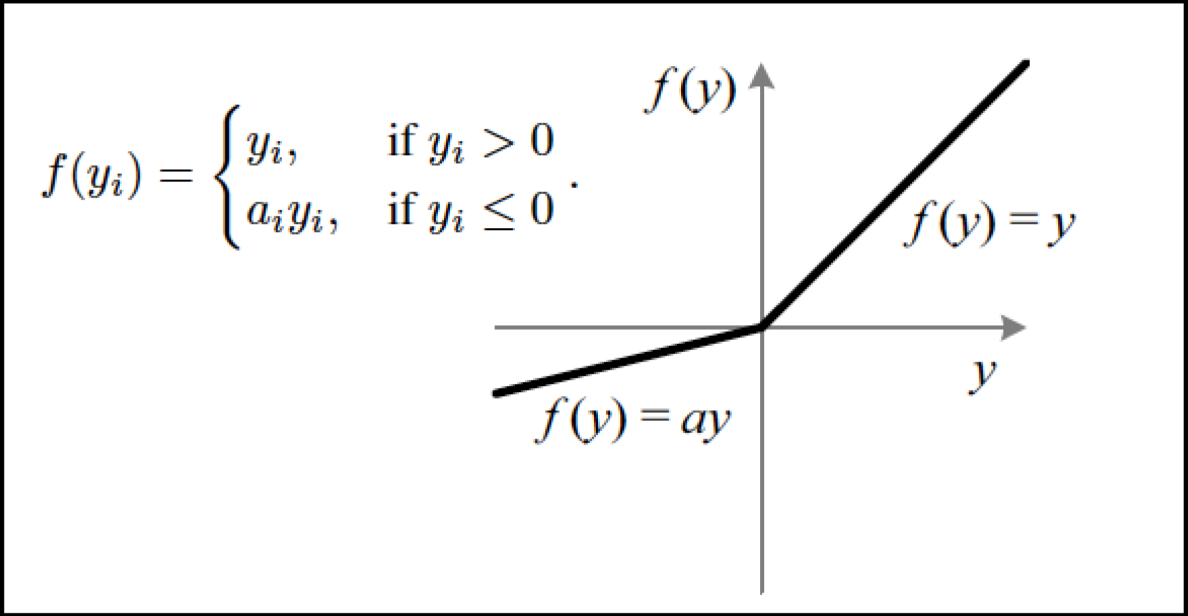
Non-Linearity

- Other choices:
  - Tanh
  - Sigmoid: 1/(1+exp(-x))
  - PReLU





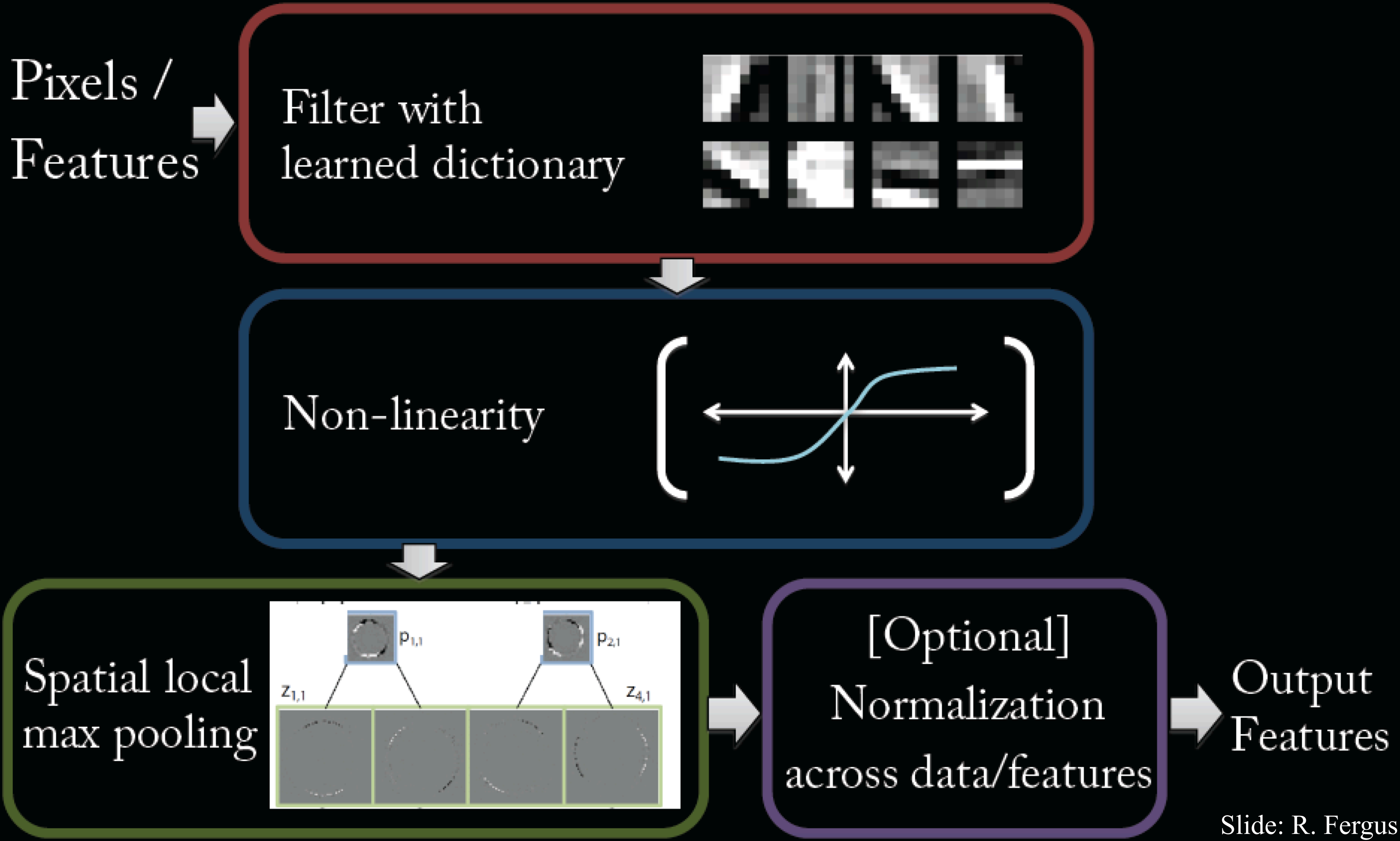
[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, Kaiming He et al. arXiv:1502.01852v1.pdf, Feb 2015]



$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \leq 0 \end{cases}.$$

Slide: R. Fergus

Components of Each Layer



Pixels / Features

Filter with learned dictionary

Non-linearity

Spatial local max pooling

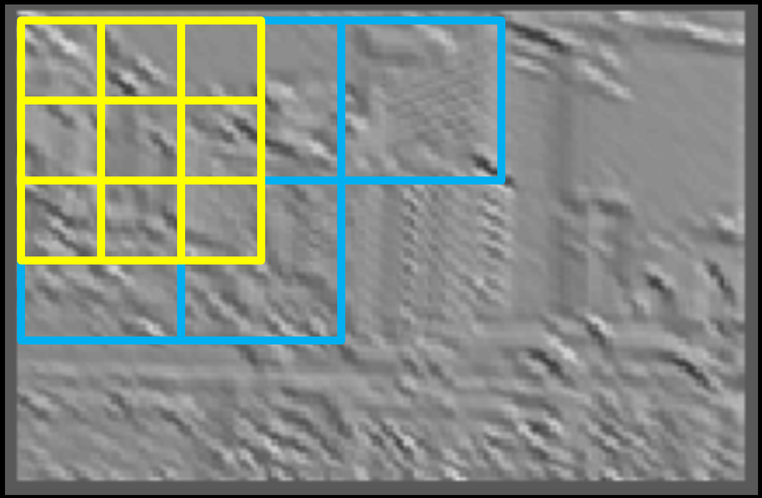
[Optional] Normalization across data/features

Output Features

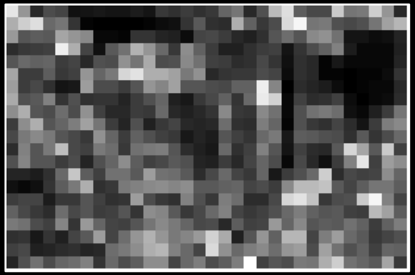
Slide: R. Fergus

Pooling

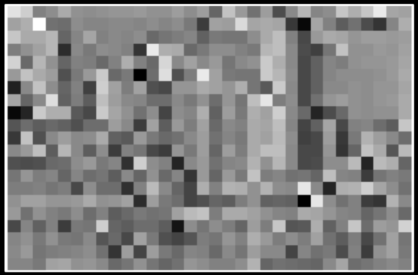
- Spatial Pooling
  - Non-overlapping / overlapping regions
  - Sum or max
  - Boureau et al. ICML'10 for theoretical analysis



Max



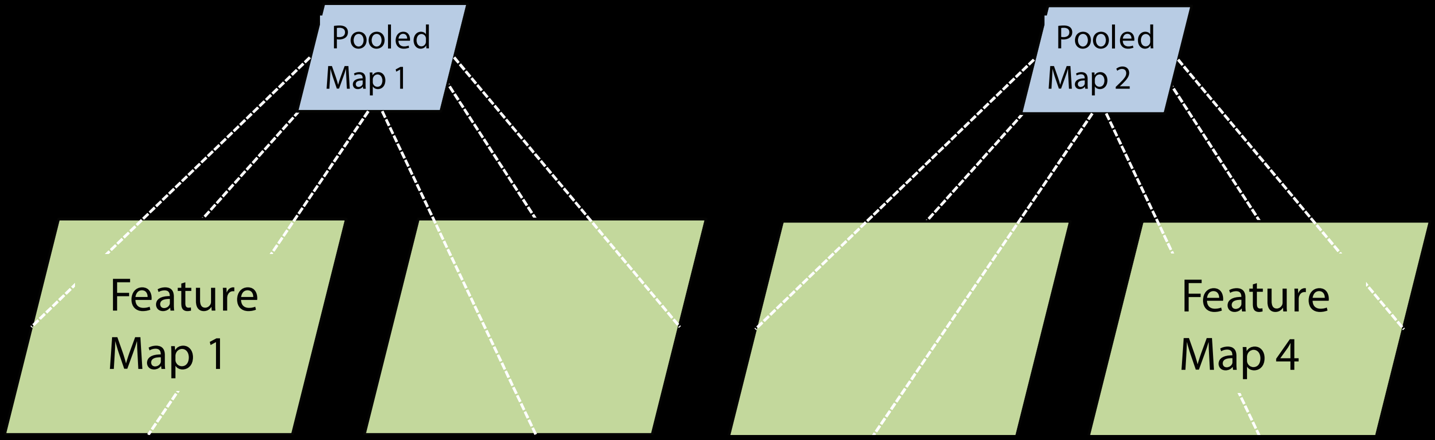
Sum



Slide: R. Fergus

Pooling

- Pooling across feature groups
  - Additional form of inter-feature competition
  - MaxOut Networks [Goodfellow et al. ICML 2013]

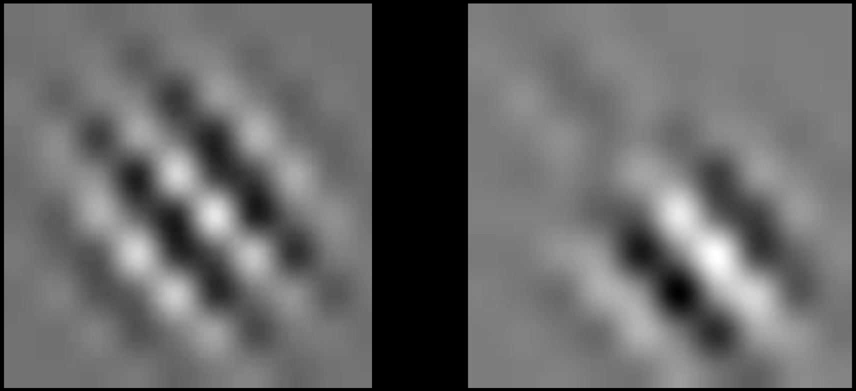


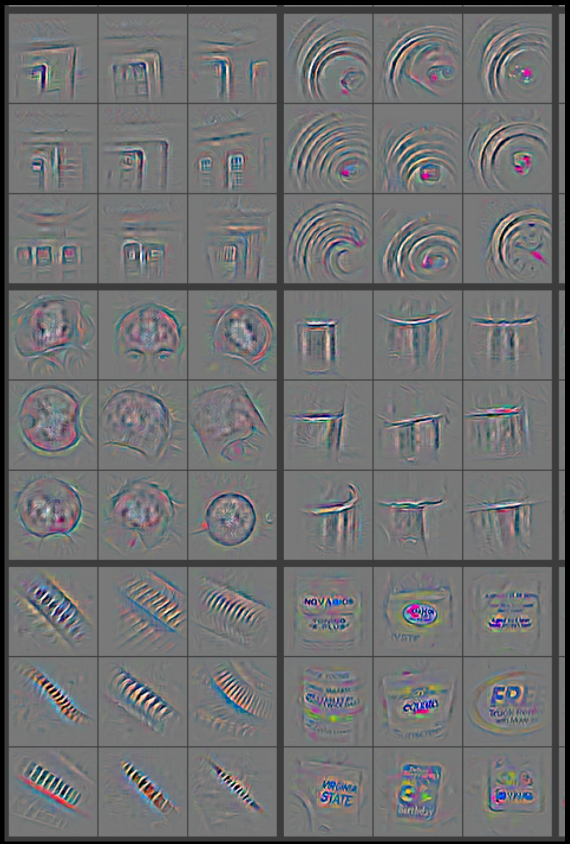
Slide: R. Fergus

Role of Pooling

- Spatial pooling
  - Invariance to small transformations
  - Larger receptive fields (see more of input)

Visualization technique from [Le et al. NIPS’10]:





Zeiler, Fergus [arXiv 2013]

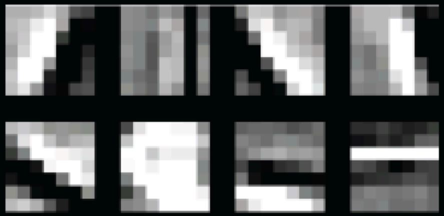
Videos from: http://ai.stanford.edu/~quocle/TCNNweb

Slide: R. Fergus

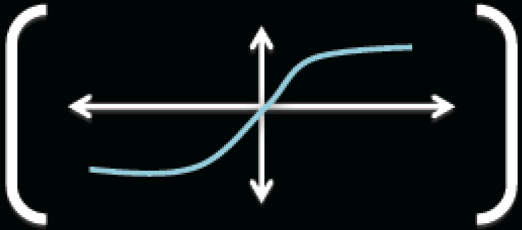
Components of Each Layer

Pixels / Features

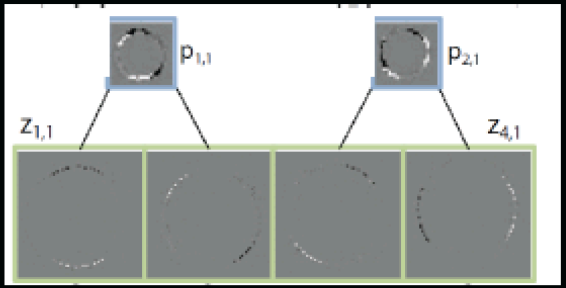
Filter with learned dictionary



Non-linearity



Spatial local max pooling



[Optional] Normalization across data/features

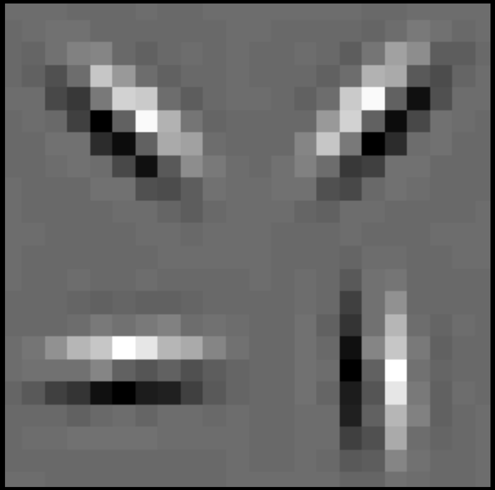
Output Features

Slide: R. Fergus

Normalization

- Contrast normalization
  - See Divisive Normalization in Neuroscience

Input

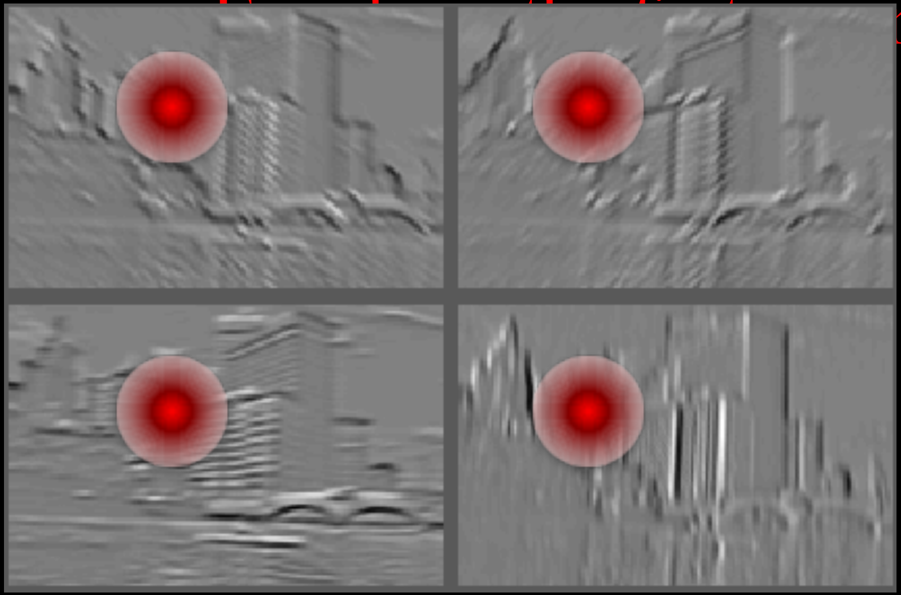


Filters

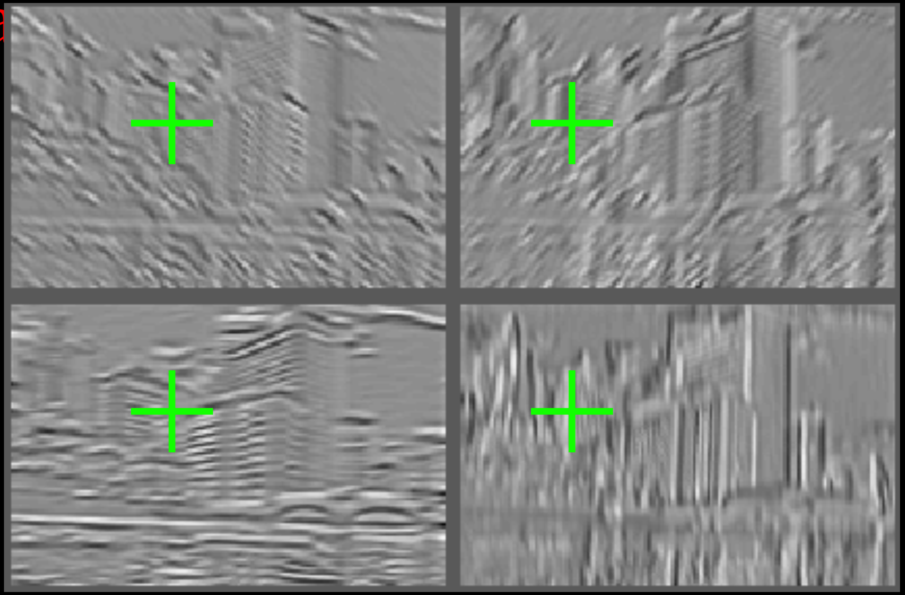
Slide: R. Fergus

Normalization

- Contrast normalization (across feature maps)
  - Local mean = 0, local std. = 1, "Local" → 7x7 Gaussian
  - Equalizes the feature maps



Feature Maps



Feature Maps
After Contrast Normalization

Slide: R. Fergus

Role of Normalization

- Introduces local competition between features
  - "Explaining away" in graphical models
  - Just like top-down models
  - But more local mechanism

- Also helps to scale activations at each layer better for learning
  - Makes energy surface more isotropic
  - So each gradient step makes more progress

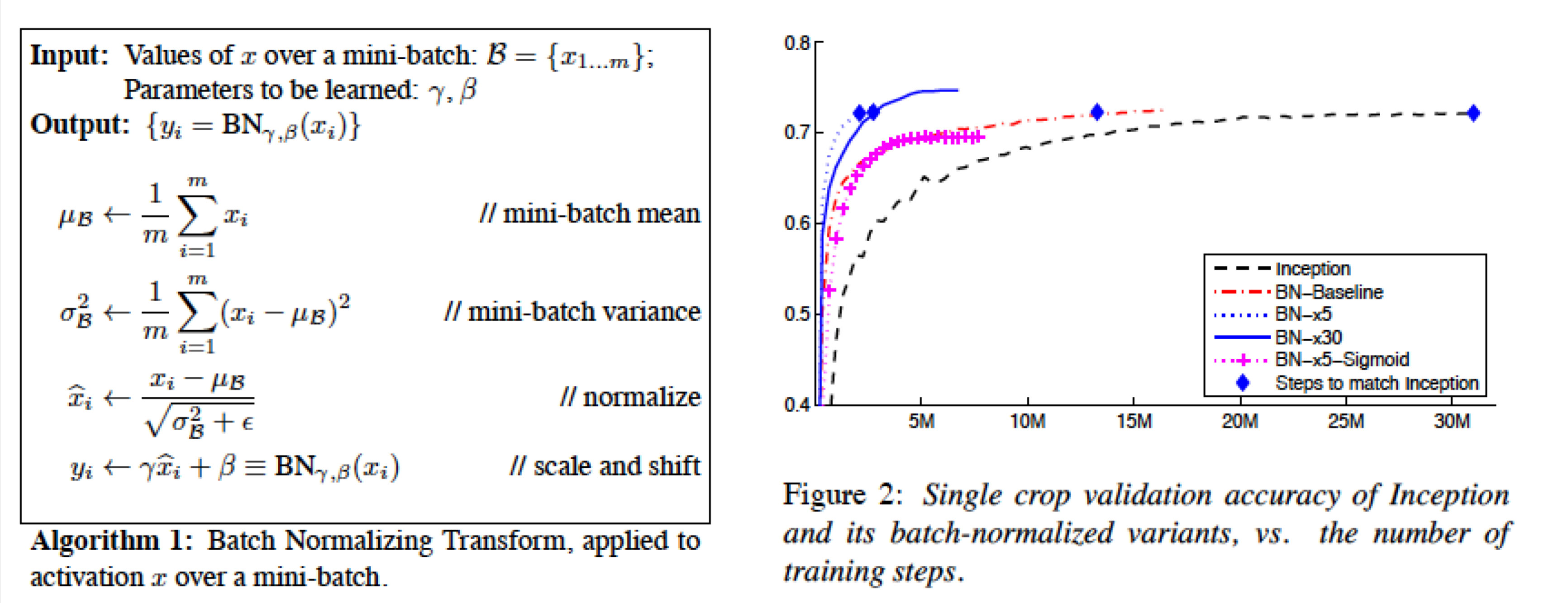
- Empirically, seems to help a bit (1-2%) on ImageNet

- Recent models do not use normalization

Normalization across Data

- Batch Normalization

[Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Sergey Ioffe, Christian Szegedy, arXiv:1502.03167]



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

```
$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$    // mini-batch mean
$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$    // mini-batch variance
$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$    // normalize
$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$    // scale and shift
```

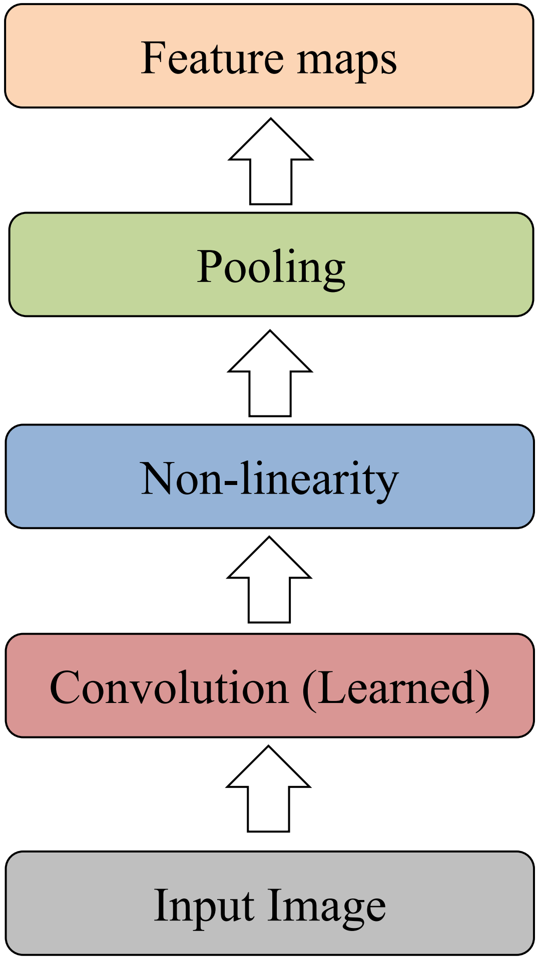
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Slide: R. Fergus

Overview of Convnets

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



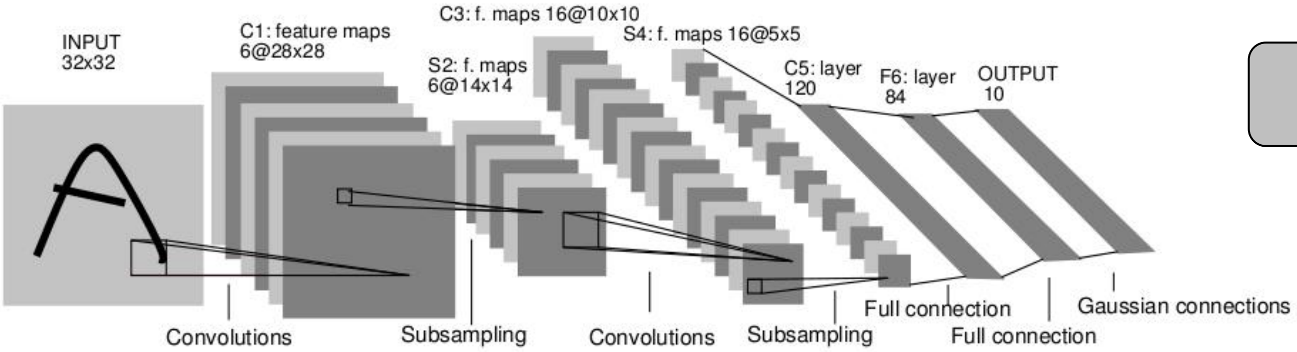
Feature maps

Pooling

Non-linearity

Convolution (Learned)

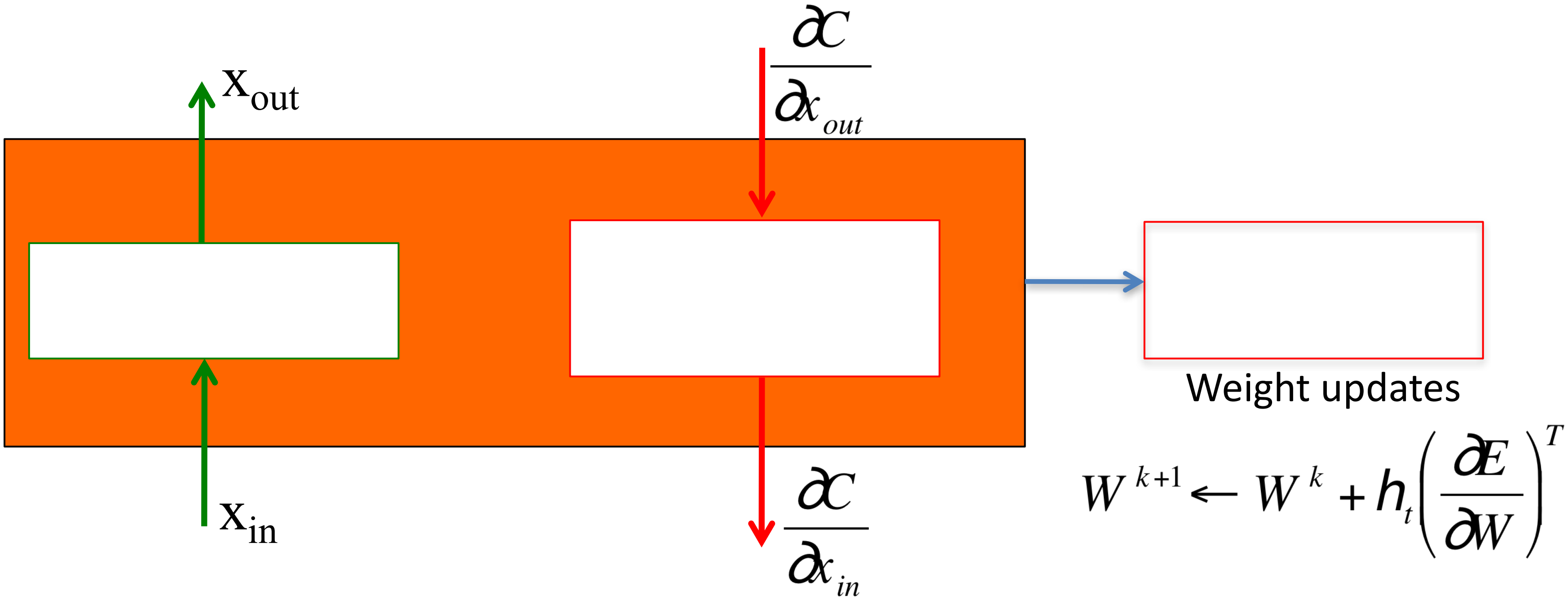
Input Image



LeCun et al. 1998

Pset: Convolution Module

Assume the input x_{in} and output x_{out} are 1D signals of the same length N.
The convolution kernel is w_i, and has length M < N


x_{out}
$\frac{\partial C}{\partial x_{out}}$
x_{in}
$\frac{\partial C}{\partial x_{in}}$
Weight updates

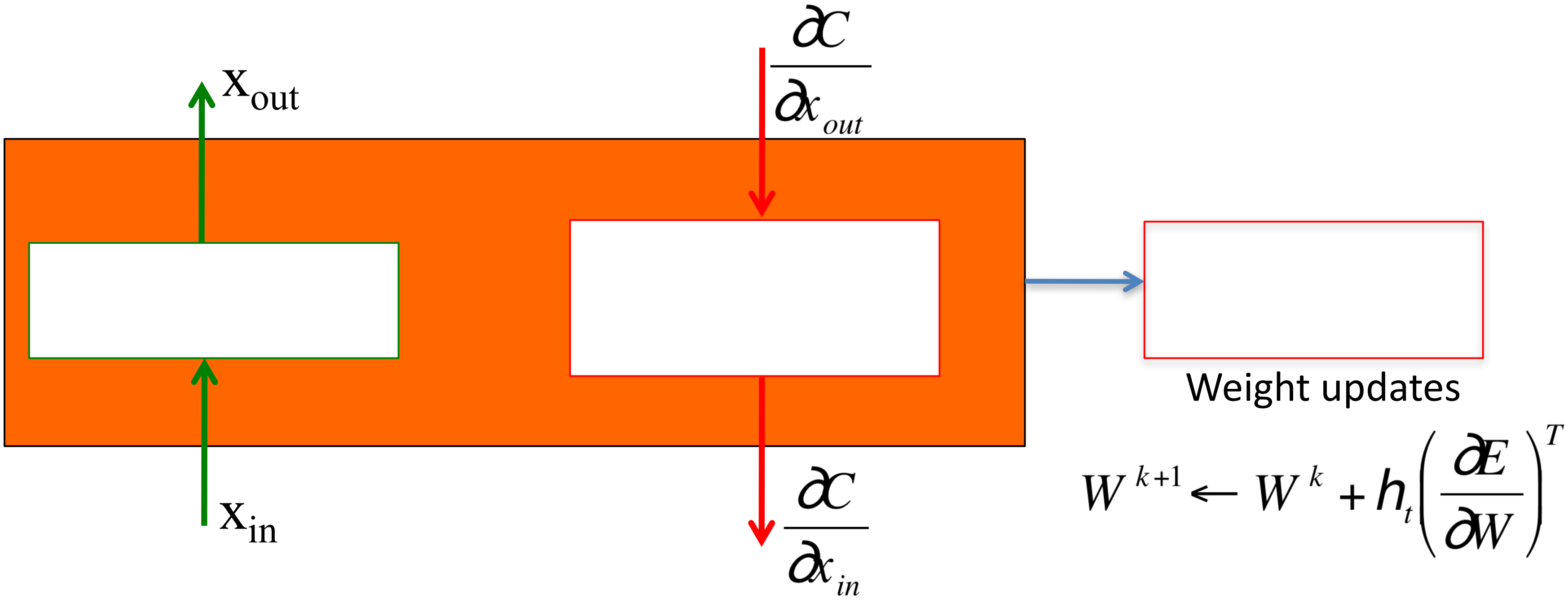
$$W^{k+1} \leftarrow W^{k} + h_t \left(\frac{\partial E}{\partial W} \right)^{T}$$

Derive the equations that go inside each box.
Discuss how you handle the boundaries.

Pset: max pooling Module

(grad course, optional for undergrads)

Assume the input x_{in} and output x_{out} are 1D signals of different lengths.



x_{out}
$\frac{\partial C}{\partial x_{out}}$
x_{in}
$\frac{\partial C}{\partial x_{in}}$
Weight updates

$$W^{k+1} \leftarrow W^{k} + h_t \left(\frac{\partial E}{\partial W} \right)^T$$

Derive the equations that go inside each box.
Discuss how you handle the boundaries.

Architecture

- Big issue: how to select
  - Depth
  - Width
  - Parameter count

- Manual tuning of features has turn into manual tuning of Architectures

How we choose the architecture?

- Many hyper-parameters:
- – # layers, # feature maps
- Cross-validation
- Grid search (need lots of GPUs)
- Smarter strategies:

  – Random [Bergstra & Bengio JMLR 2012]

  – Gaussian processes [Hinton]

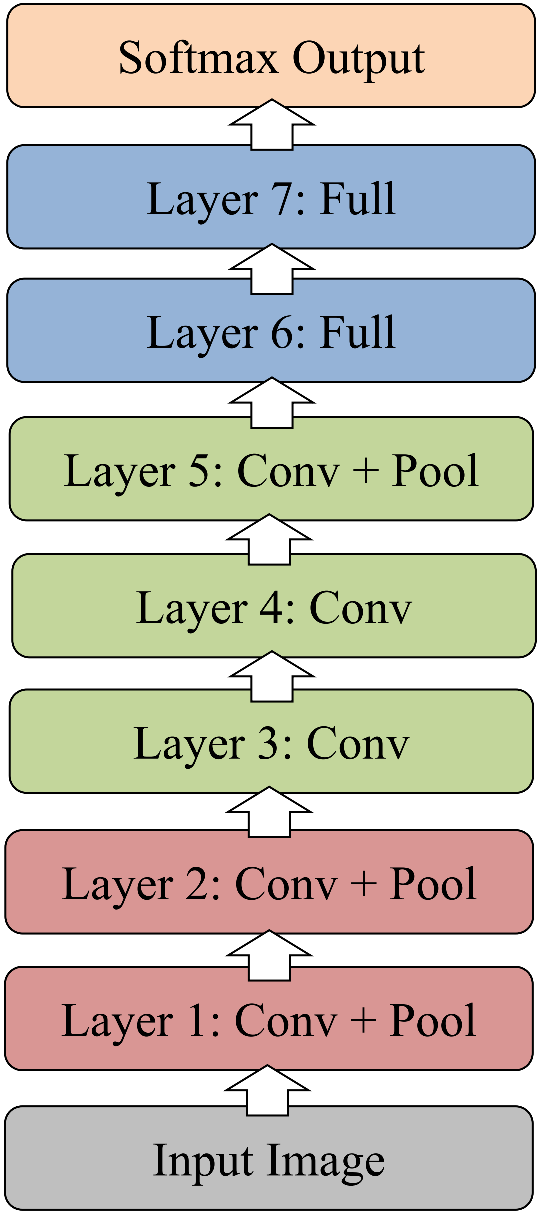
How important is Depth

- “Deep” in Deep Learning
- Ablation study
- Tap off features

Architecture of Krizhevsky et al.

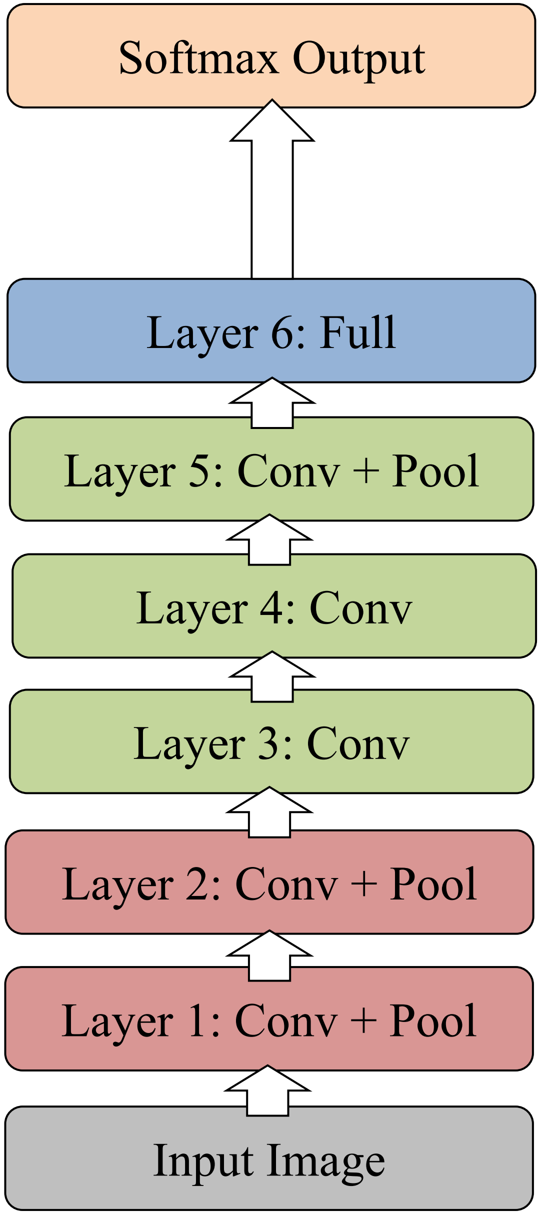
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:

  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!

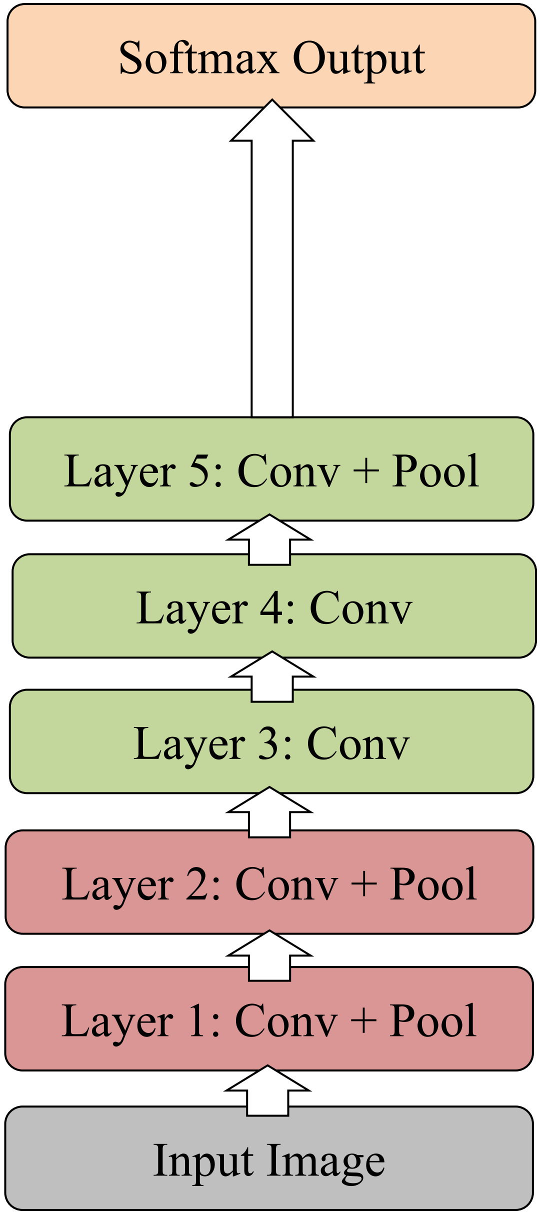


Architecture of Krizhevsky et al.

- Remove both fully connected layers
  - Layer 6 & 7

- Drop ~50 million parameters

- 5.7% drop in performance

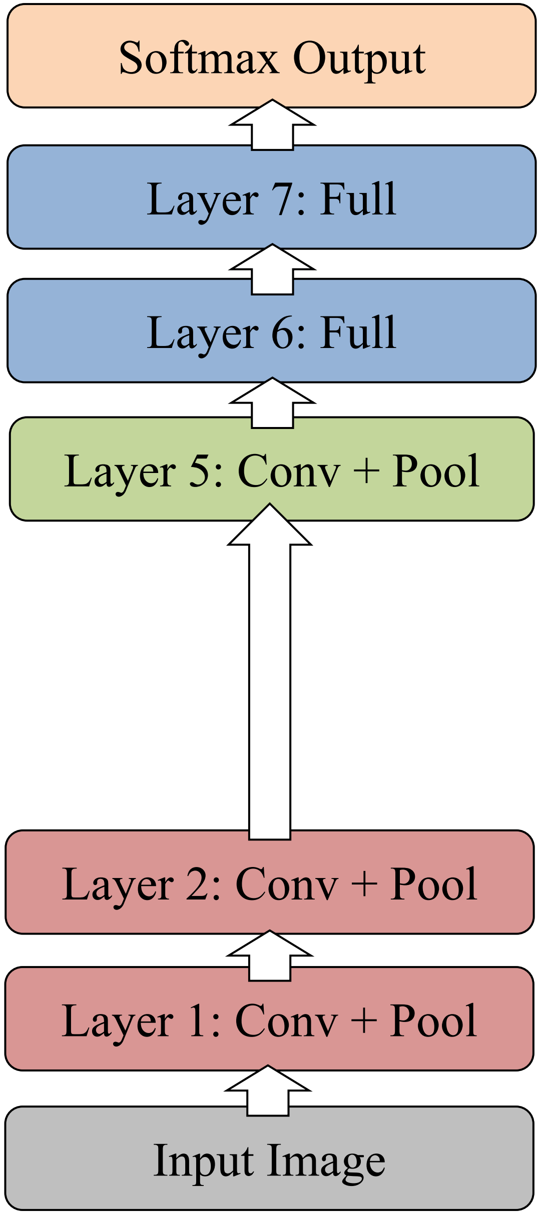


Architecture of Krizhevsky et al.

- Now try removing upper feature extractor layers:
  - Layers 3 & 4

- Drop ~1 million parameters

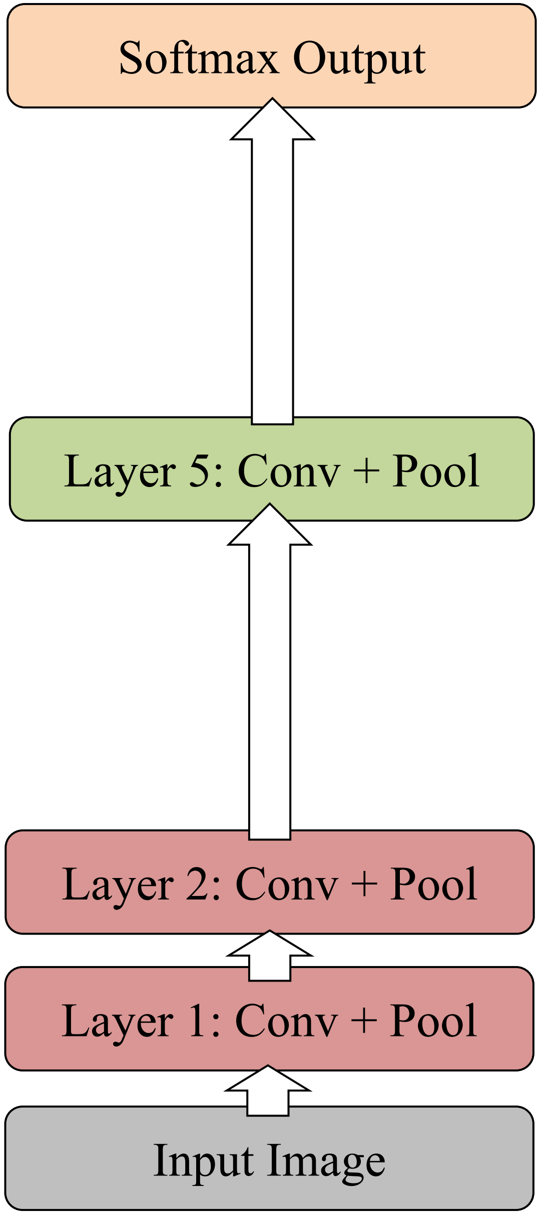
- 3.0% drop in performance



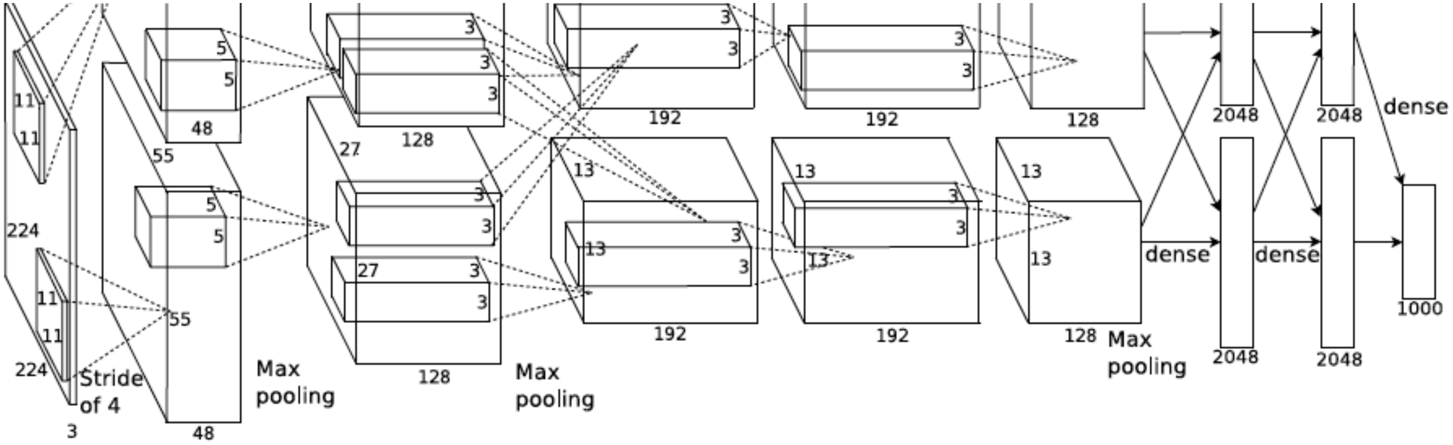
Architecture of Krizhevsky et al.

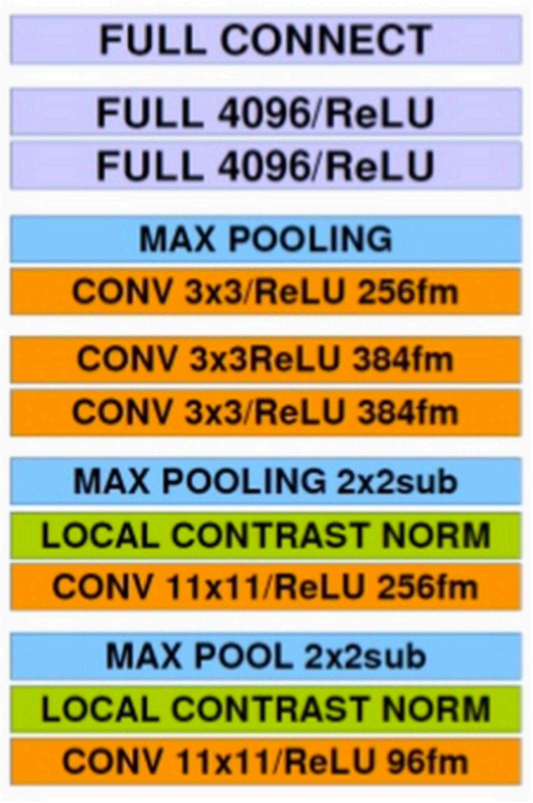
- Now try removing upper feature extractor layers & fully connected:
  - Layers 3, 4, 6 ,7
- Now only 4 layers
- 33.5% drop in performance

→Depth of network is key



Krizhevsky et al. [NIPS2012]





FULL CONNECT

FULL 4096/ReLU

FULL 4096/ReLU

MAX POOLING

CONV 3x3/ReLU 256fm

CONV 3x3ReLU 384fm

CONV 3x3/ReLU 384fm

MAX POOLING 2x2sub

LOCAL CONTRAST NORM

CONV 11x11/ReLU 256fm

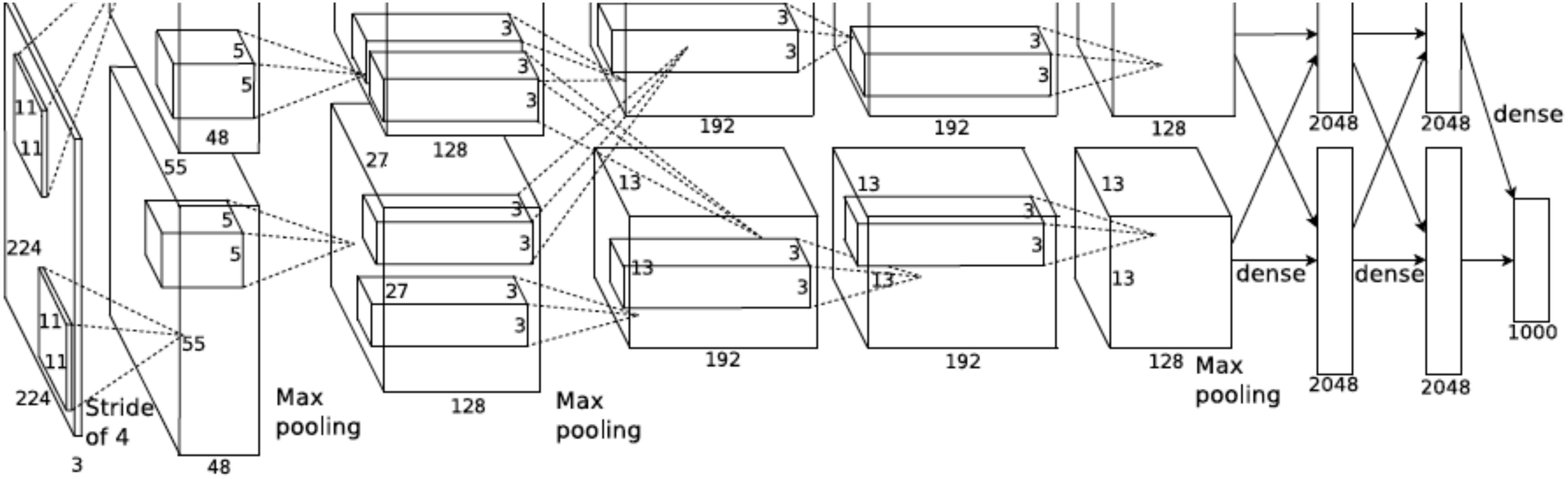
MAX POOL 2x2sub

LOCAL CONTRAST NORM

CONV 11x11/ReLU 96fm

AlexNet architecture:

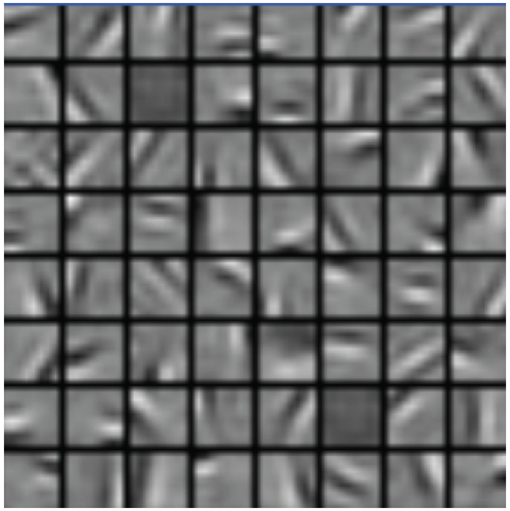
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)



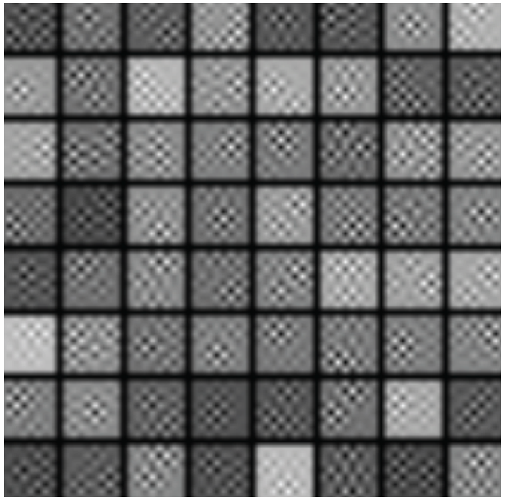
What filters are learned?

What filters are learned?

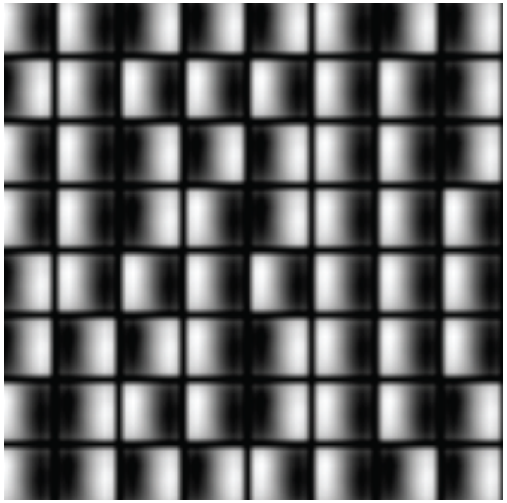
A



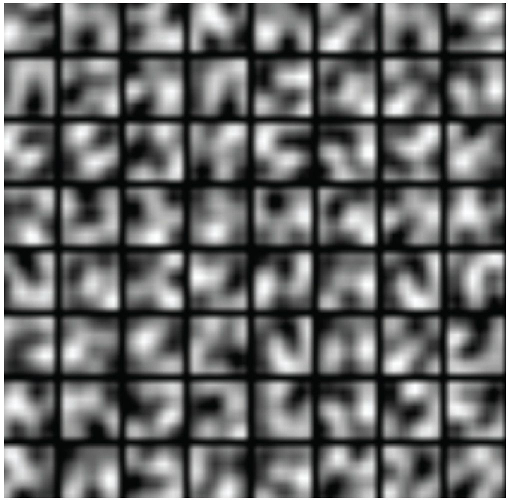
B



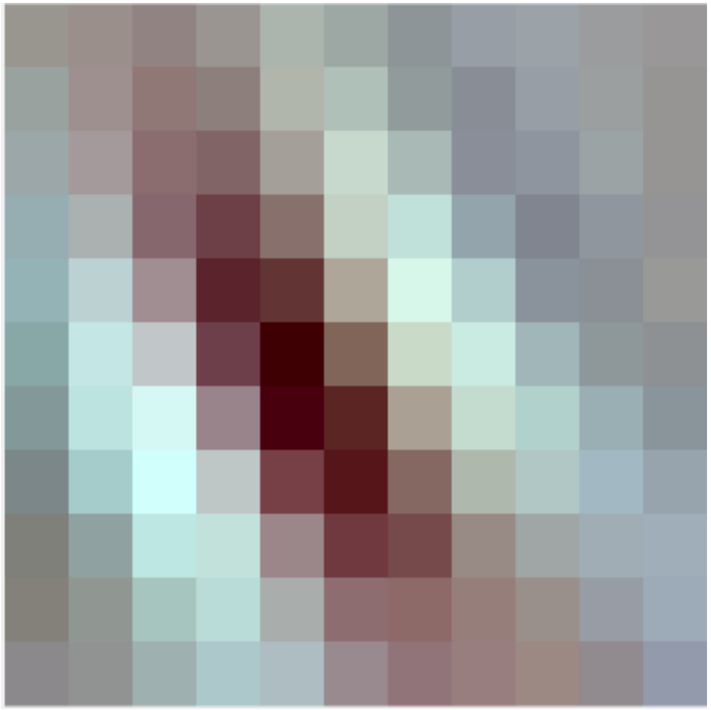
C

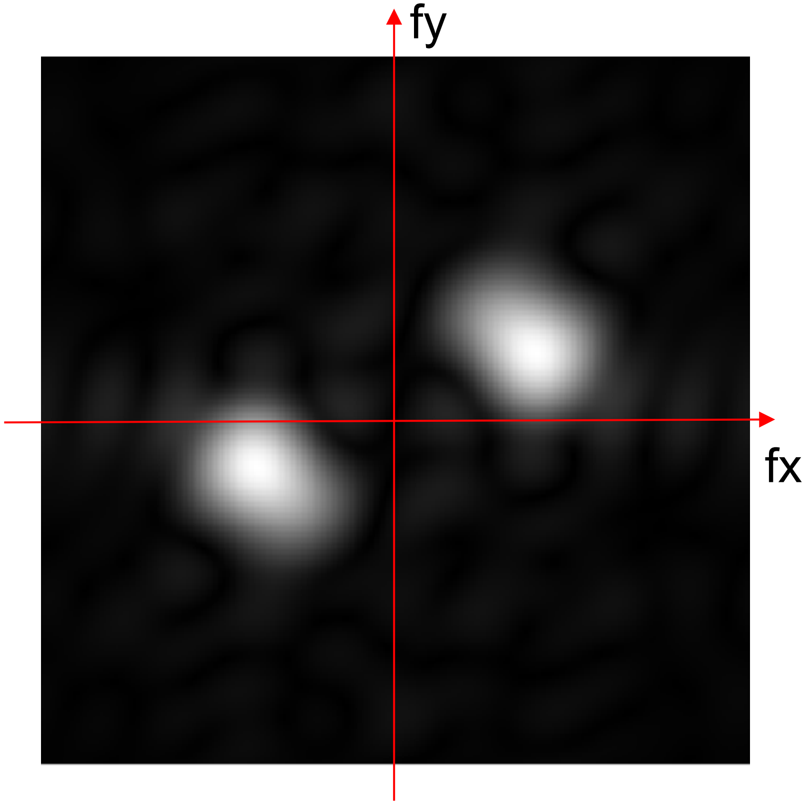


D



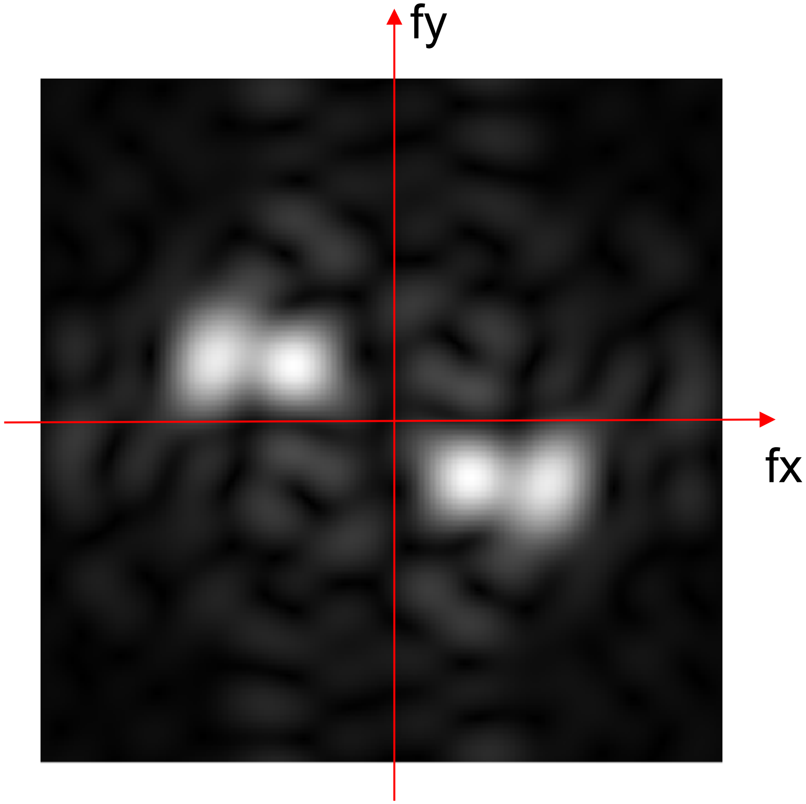
Get to know your units



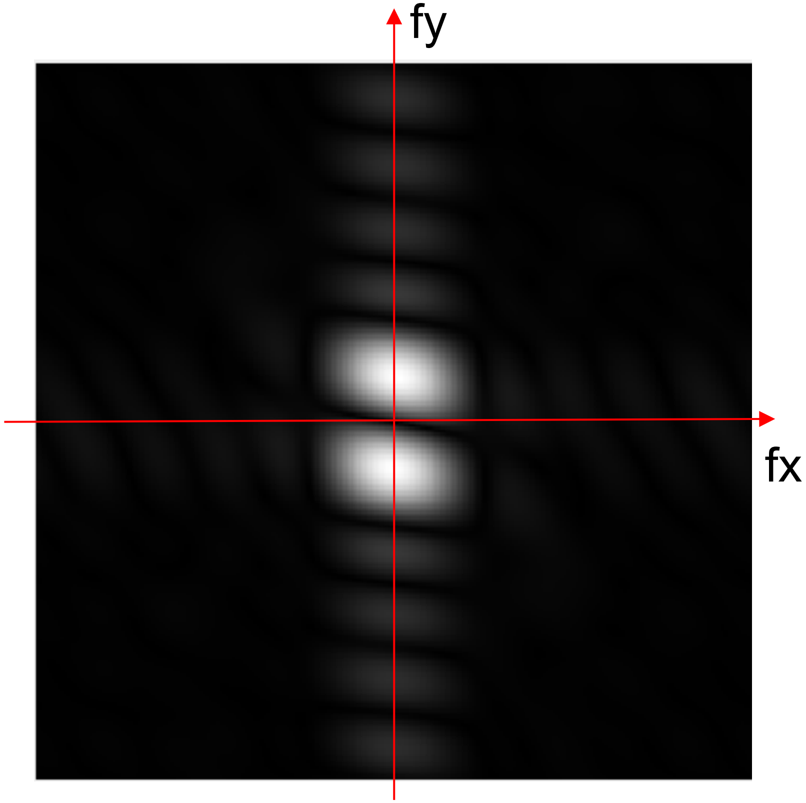


11x11 convolution kernel
(3 color channels)

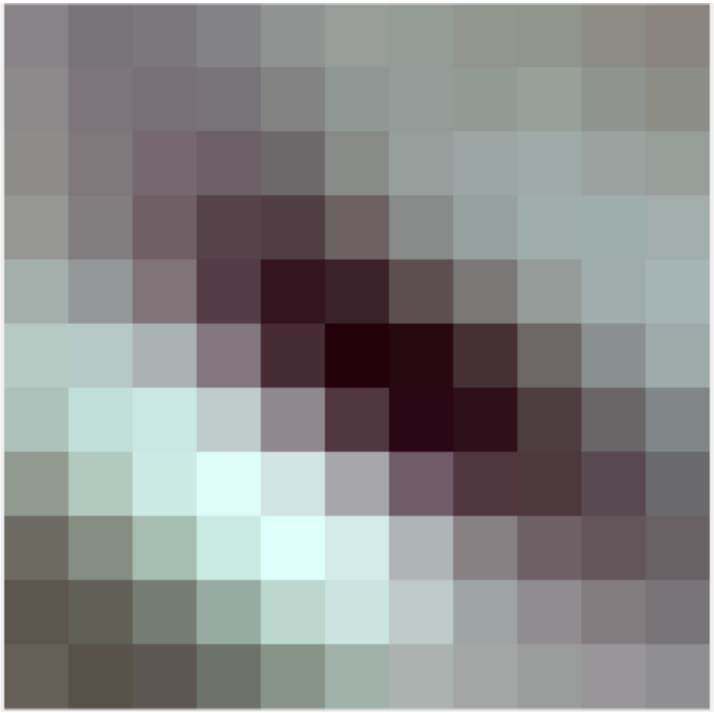
Get to know your units

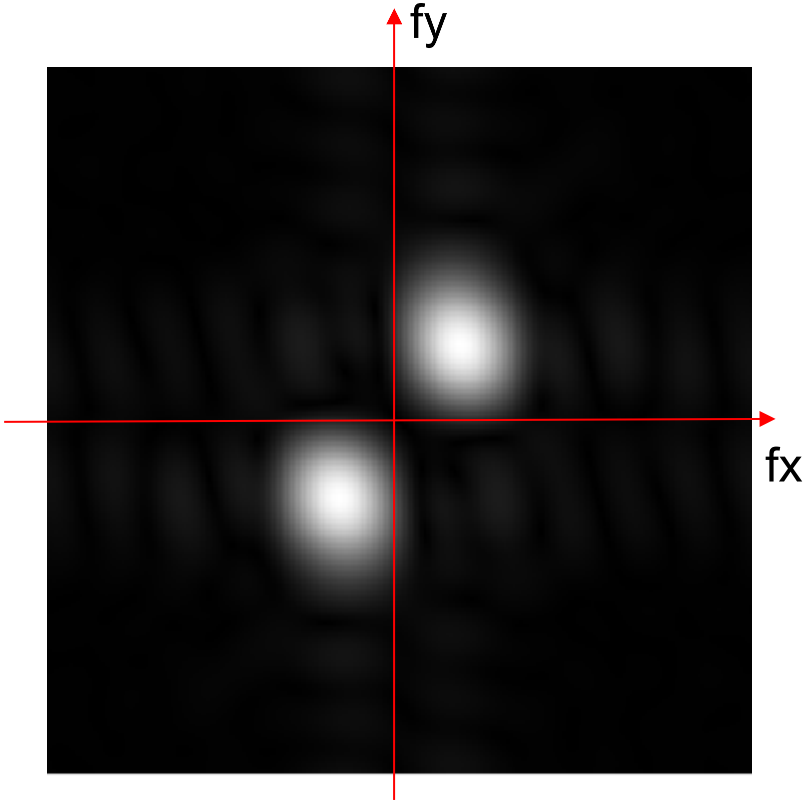


Get to know your units

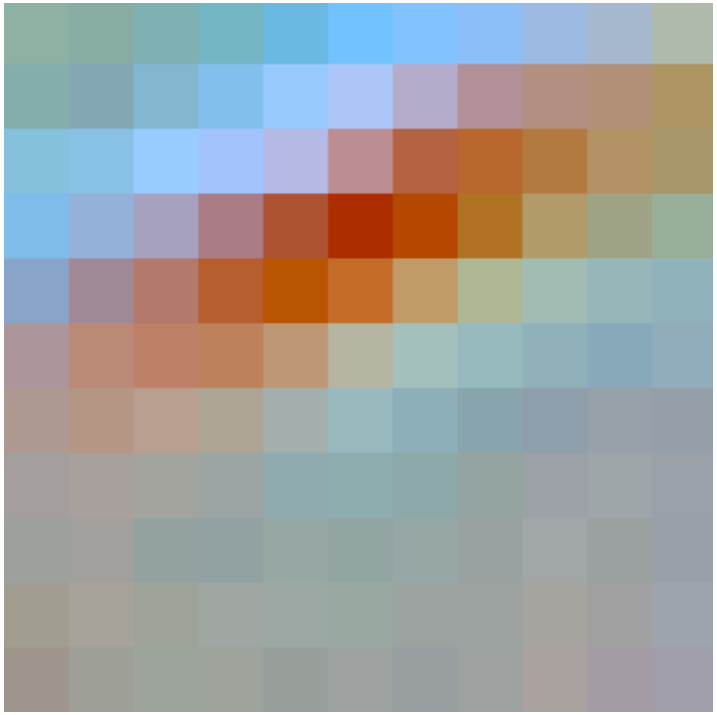


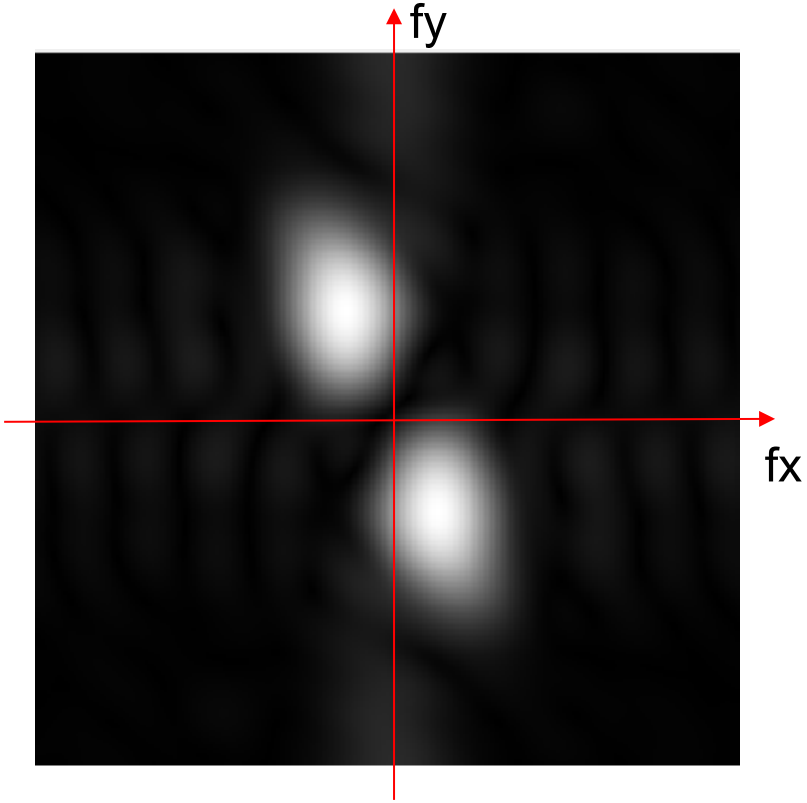
Get to know your units



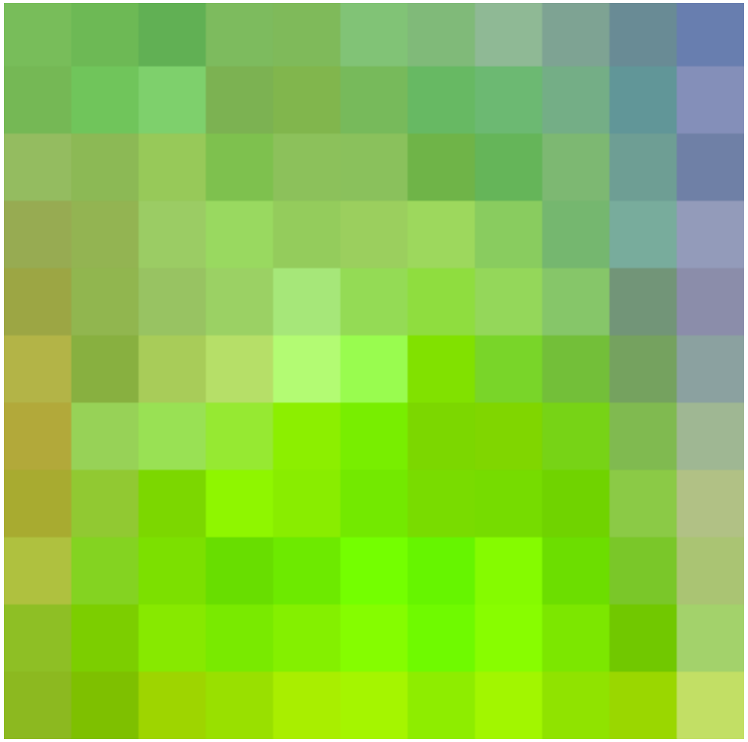


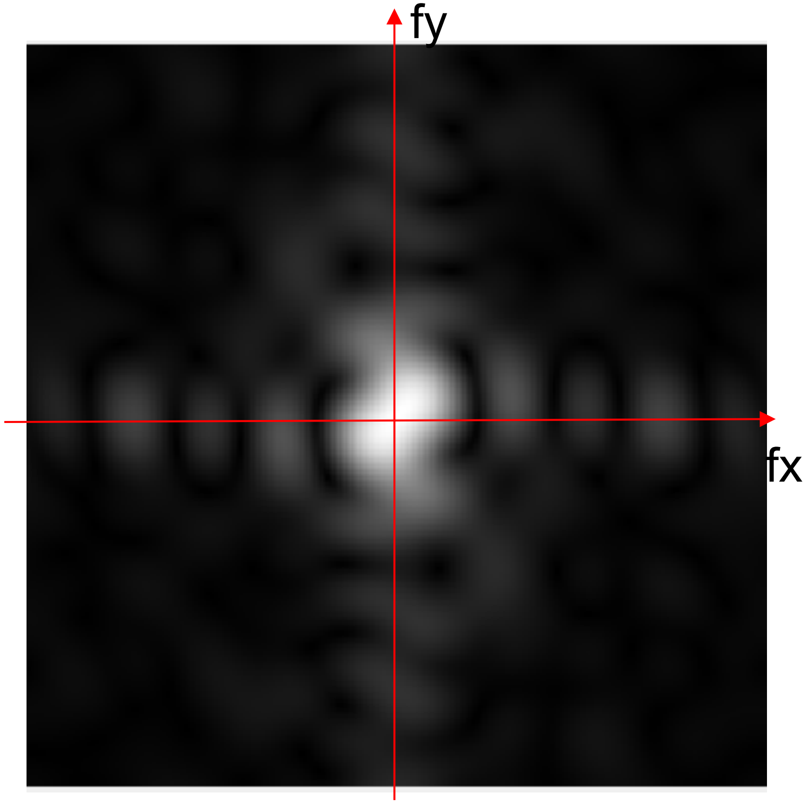
Get to know your units



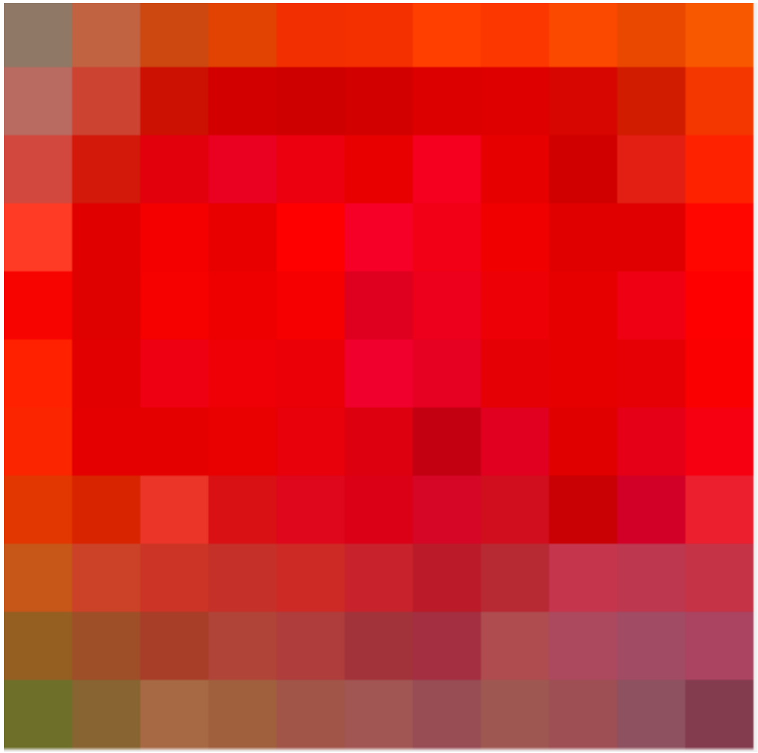


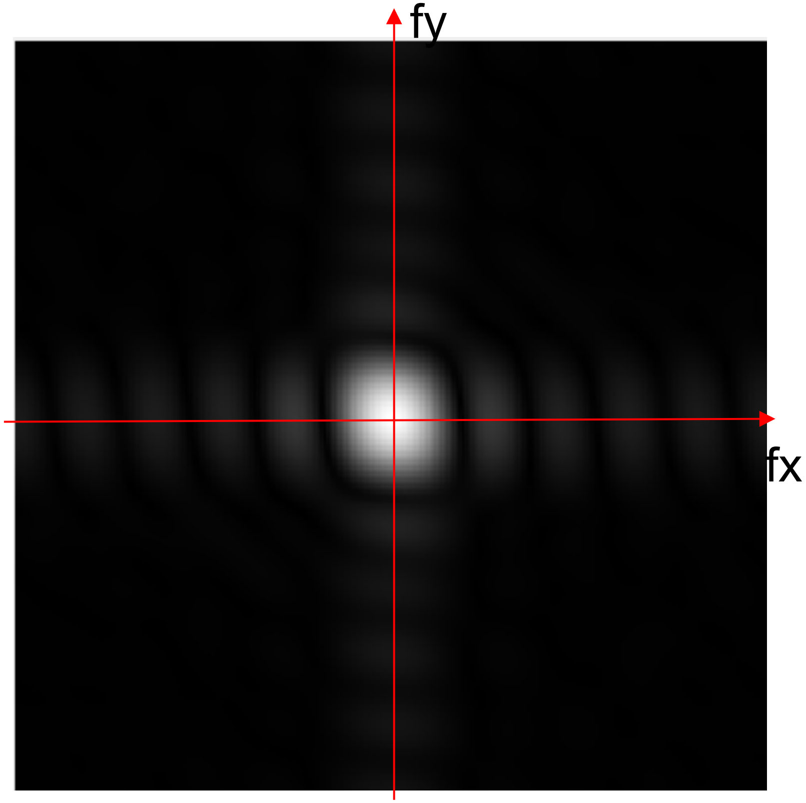
Get to know your units



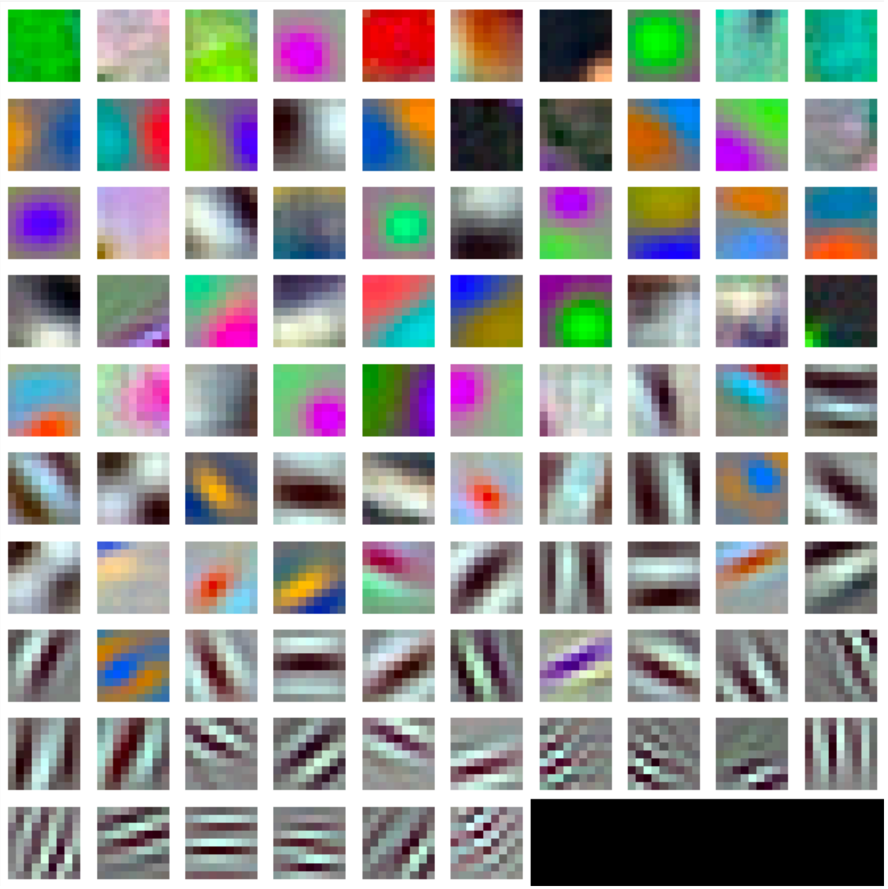


Get to know your units

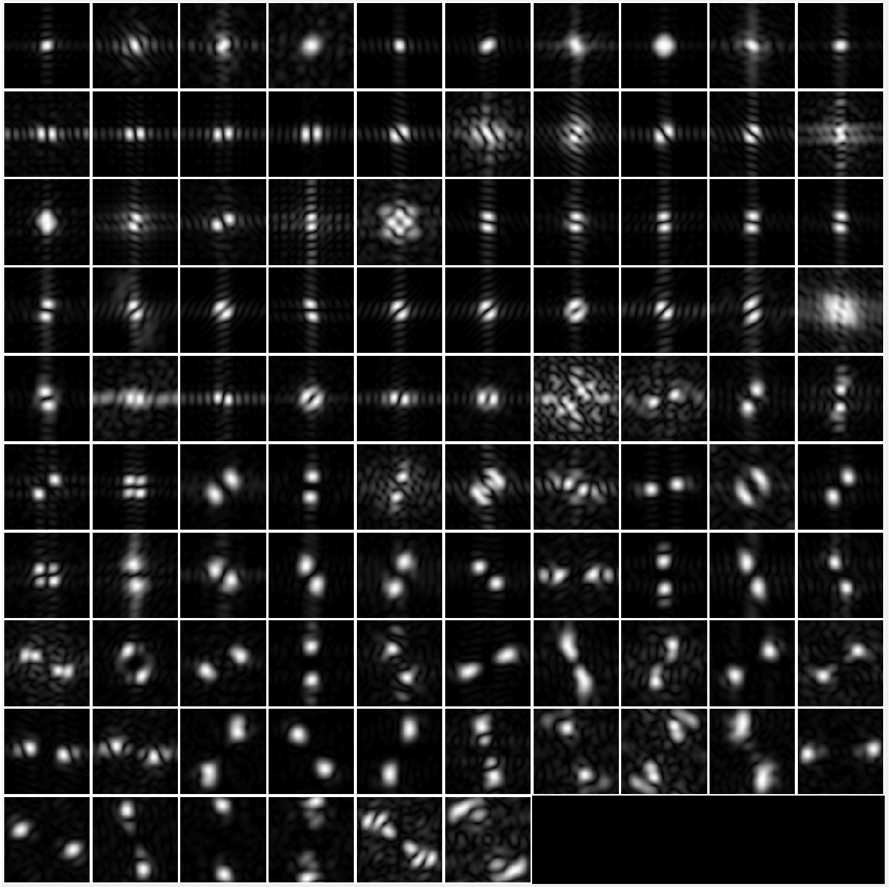




Get to know your units

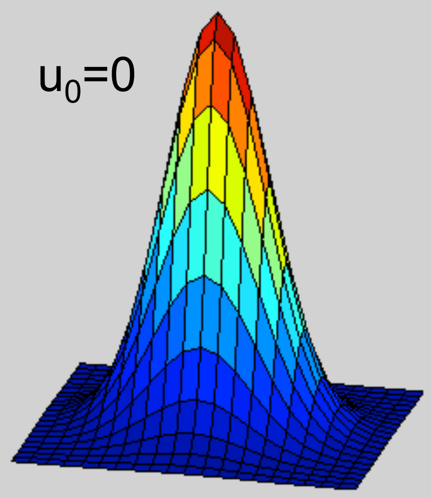


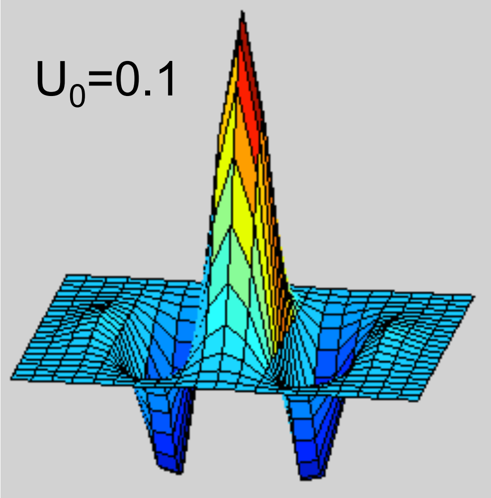
96 Units in conv1

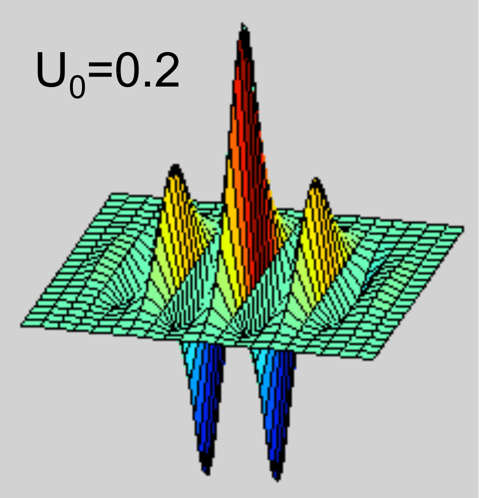


Gabor wavelets

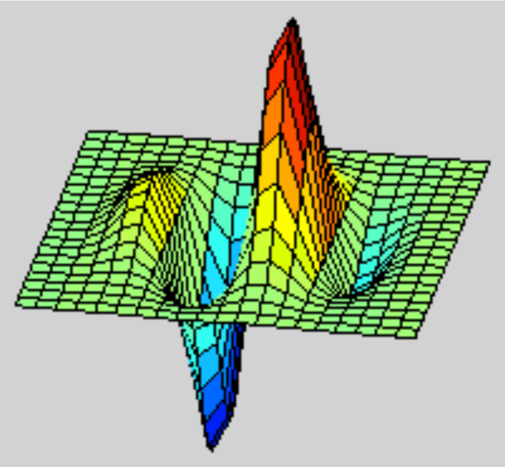
$$\psi_c(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \cos\left(2\pi u_0 x\right)$$

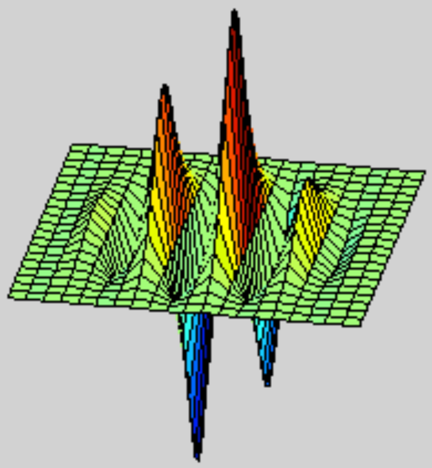




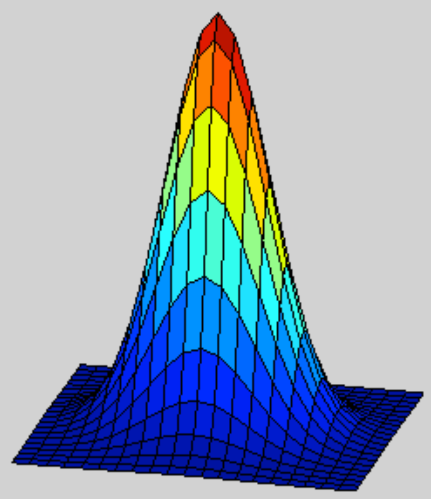


$$\psi_s(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \sin\left(2\pi u_0 x\right)$$

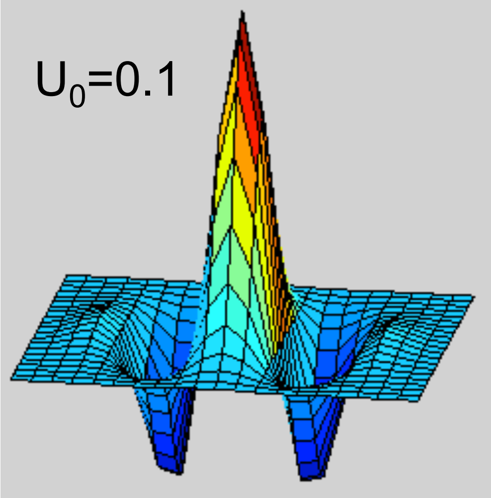


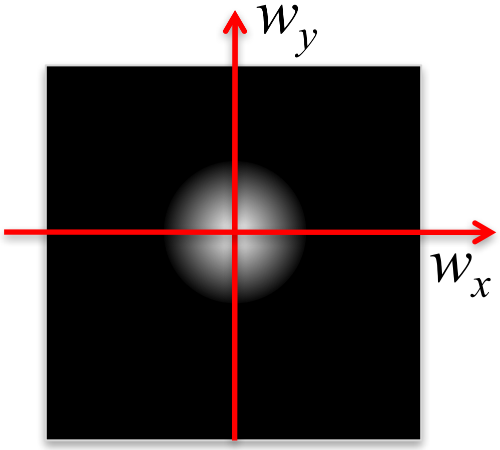


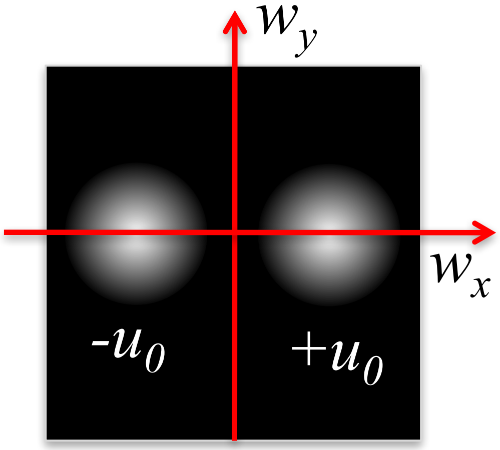
Fourier transform of a Gabor wavelet

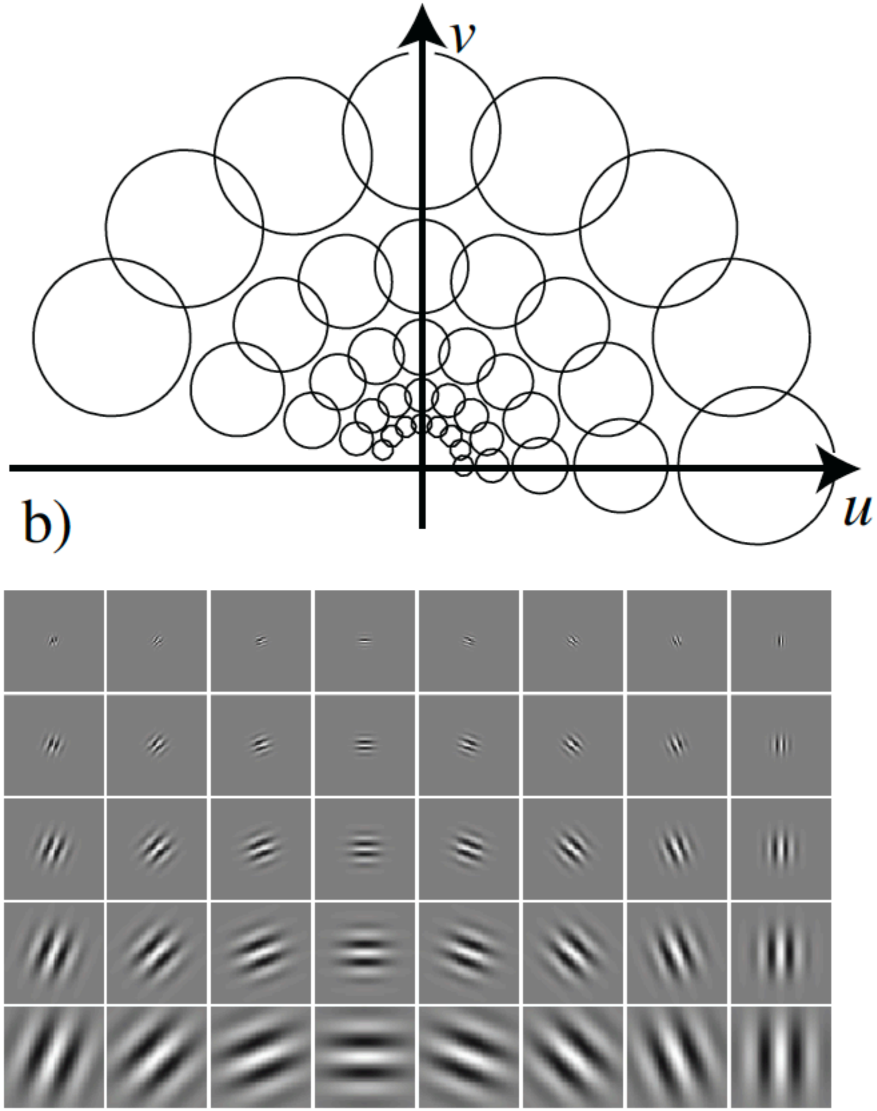


$$\psi_c(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \cos\left(2\pi u_0 x\right)$$

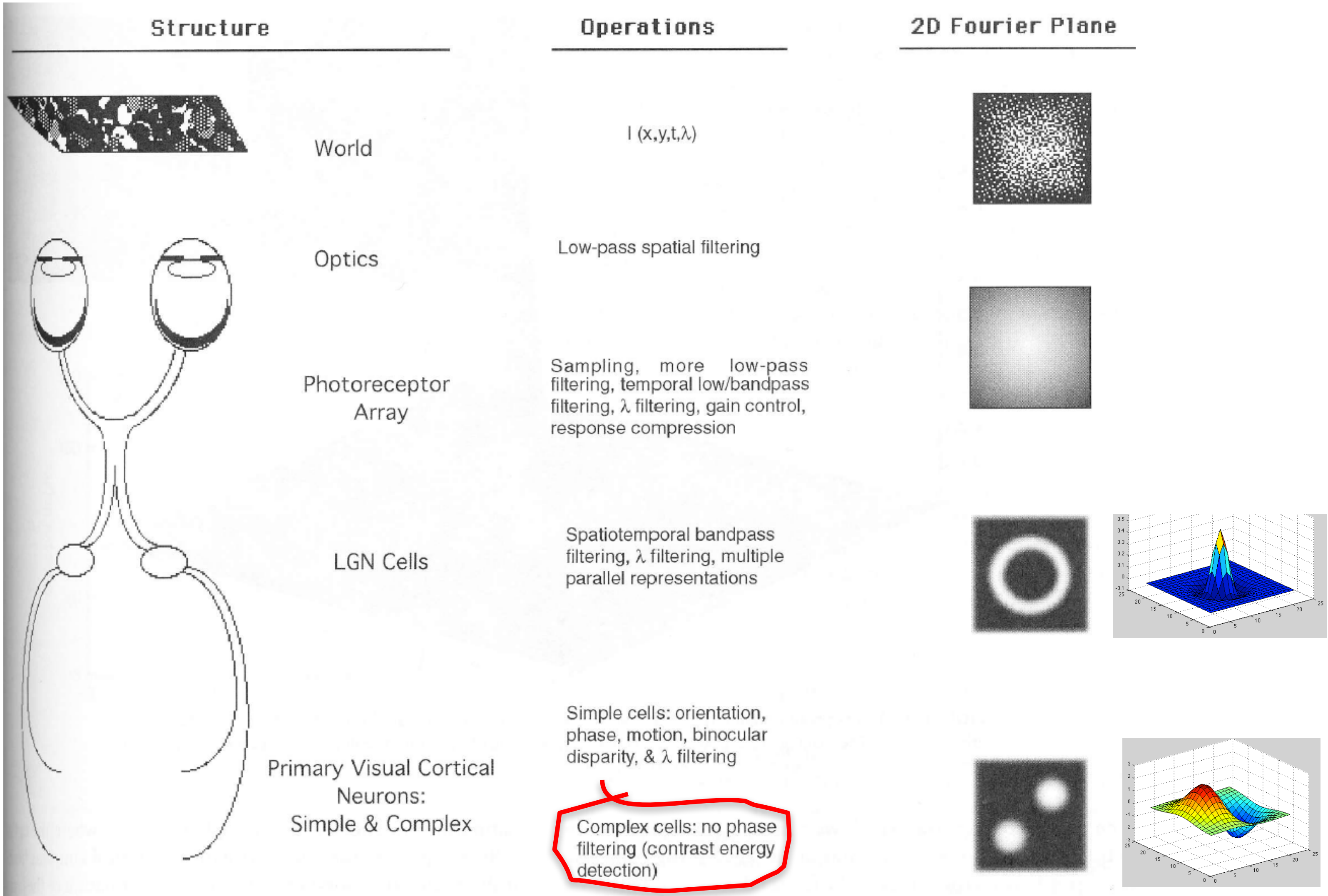







v
u
b)

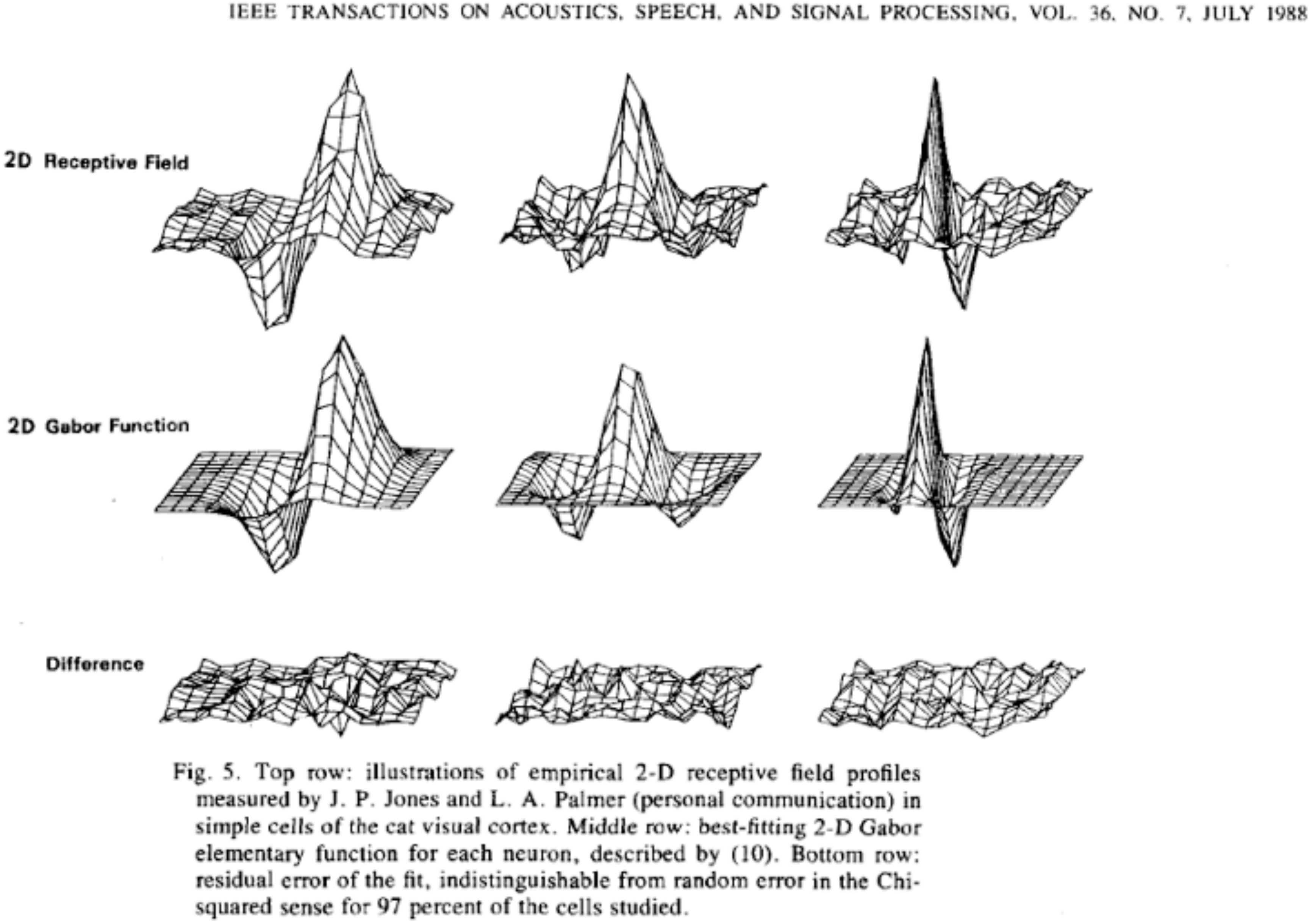
Comparing Human and Machine Perception



Structure	Operations	2D Fourier Plane
World	I (x,y,t,λ)	
Optics	Low-pass spatial filtering	
Photoreceptor Array	Sampling, more low-pass filtering, temporal low/bandpass filtering, λ filtering, gain control, response compression	
LGN Cells	Spatiotemporal bandpass filtering, λ filtering, multiple parallel representations	
Primary Visual Cortical Neurons: Simple & Complex	Simple cells: orientation, phase, motion, binocular disparity, & λ filtering Complex cells: no phase filtering (contrast energy detection)	

FIGURE 1 Schematic overview of the processing done by the early visual system. On the left, are some of the major structures to be discussed; in the middle, are some of the major operations done at the associated structure; in the right, are the 2-D Fourier representations of the world, retinal image, and sensitivities typical of a ganglion and cortical cell.

John Daugman, 1988

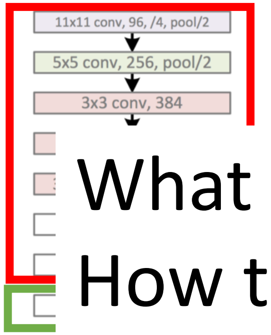


IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING, VOL. 36, NO. 7, JULY 1988

Fig. 5. Top row: illustrations of empirical 2-D receptive field profiles measured by J. P. Jones and L. A. Palmer (personal communication) in simple cells of the cat visual cortex. Middle row: best-fitting 2-D Gabor elementary function for each neuron, described by (10). Bottom row: residual error of the fit, indistinguishable from random error in the Chi-squared sense for 97 percent of the cells studied.

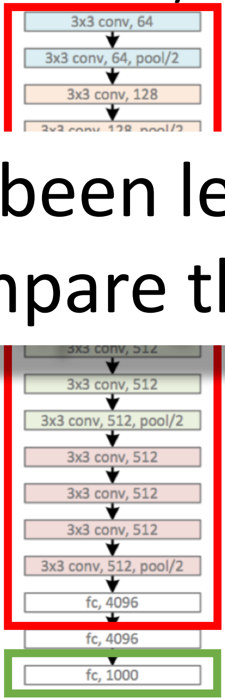
Deep Neural Networks for Visual Recognition

2012: AlexNet
5 conv. layers



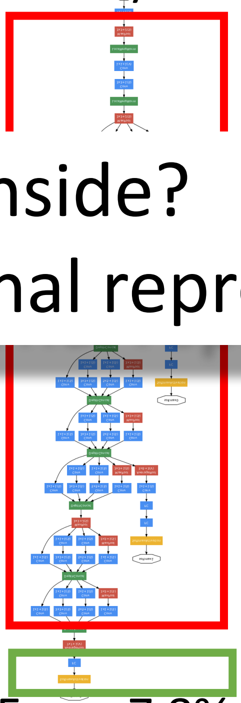
Error: 15.3%

2014: VGG
16 conv. layers



Error: 8.5%

2015: GoogLeNet
22 conv. layers



Error: 7.8%

2016: ResNet
>100 conv. layers

Error: 4.4%

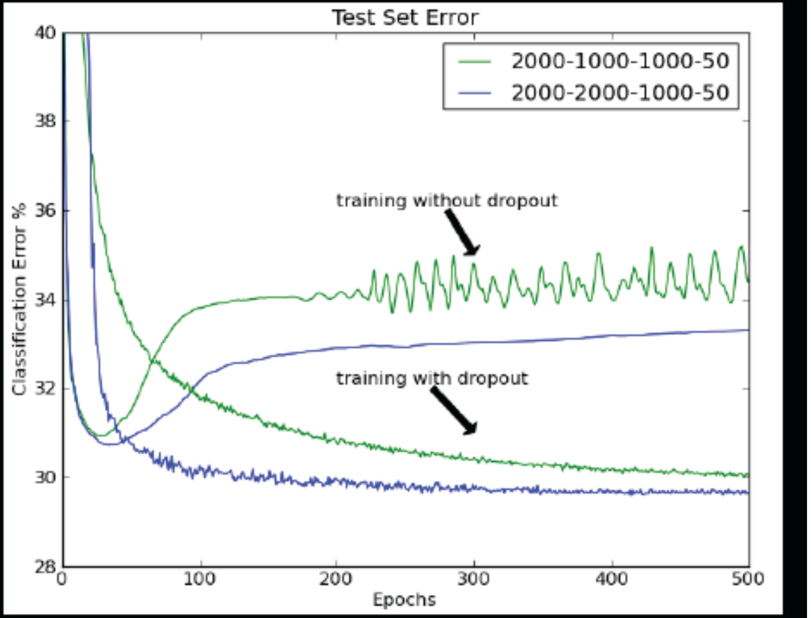
What have been learned inside?

How to compare the internal representations?

DropOut

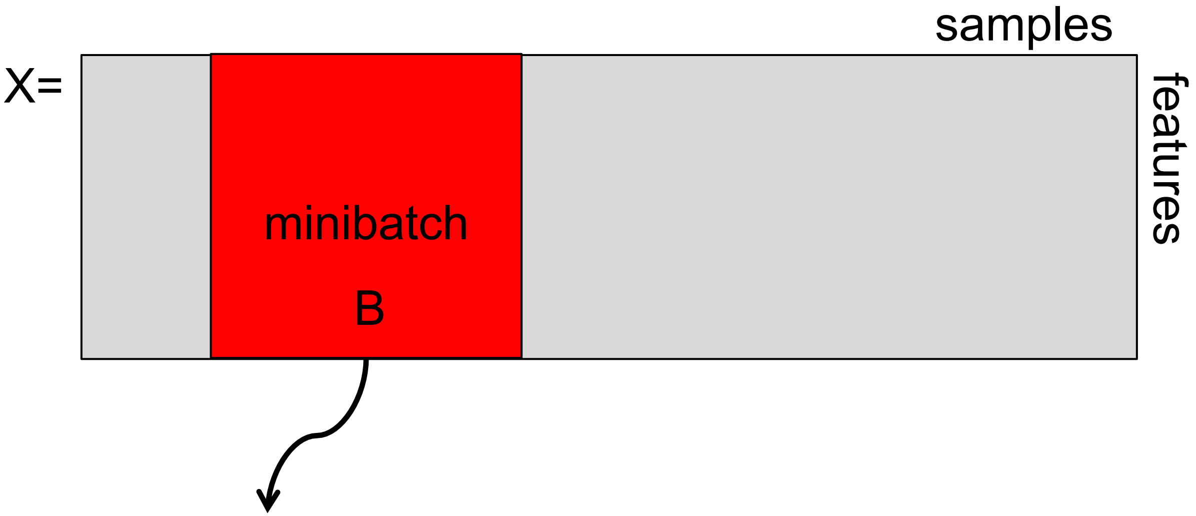
- G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, arXiv:1207.0580 2012

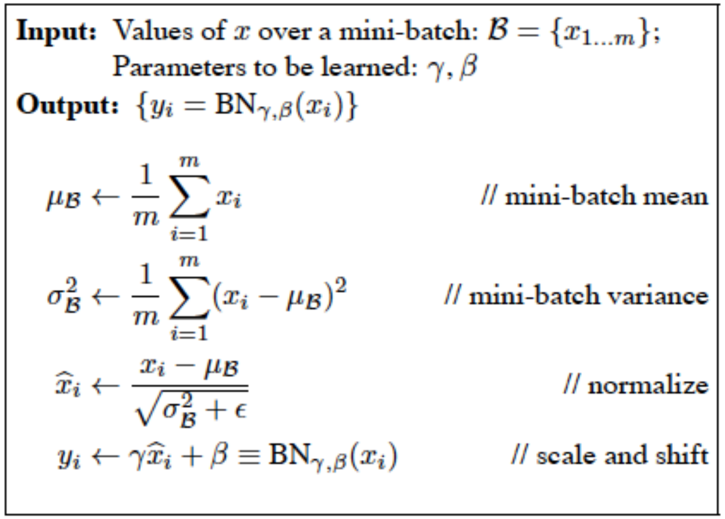
- Fully connected layers only
- Randomly set activations in layer to zero
- Gives ensemble of models
- Similar to bagging [Breiman'94], but differs in that parameters are shared.



Slide Rob Fergus

Batch normalization





```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean

    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$       // mini-batch variance

    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                 // normalize

    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$          // scale and shift
```

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

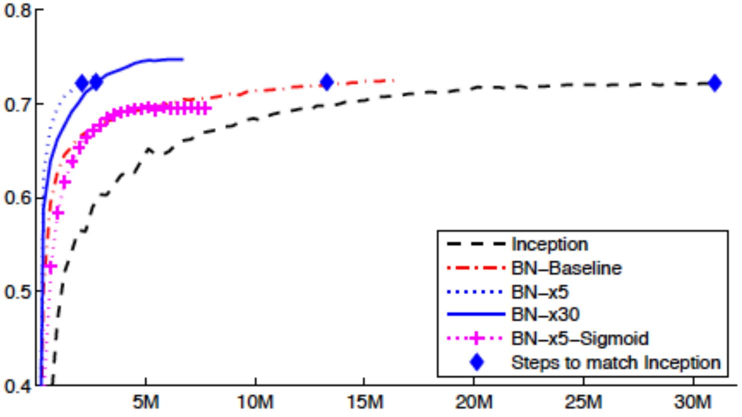


Figure 2: *Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.*

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Sergey Ioffe, Christian Szegedy, arXiv:1502.03167

Batch normalization

- Training: take into account the normalization in backdrop

  Derivative wrt x_i depends on the partial derivative of the mean and stddev

  Must also update γ and β

- Test time: use the global mean stddev at test time

  Removes the stochasticity of the mean and stddev

  Requires a final phase where, from the first to the last hidden layer

  1. propagate all training data to that layer
  2. compute and store the global mean and stddev of each unit

Fooling Convnets

- Search for images that are misclassified by the network
- Intriguing properties of neural networks, Christian Szegedy et al. arXiv 1312.6199, 2013
- Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images, Anh Nguyen, Jason Yosinski, Jeff Clune, arXiv 1412.1897.
- Problem common to any discriminative method

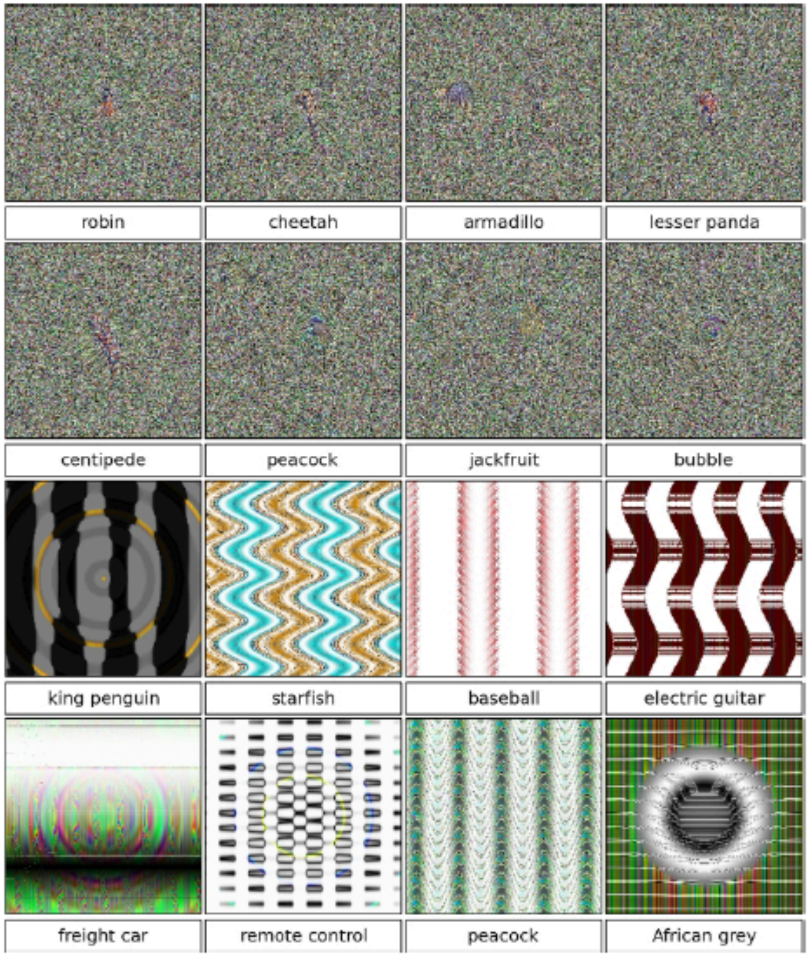
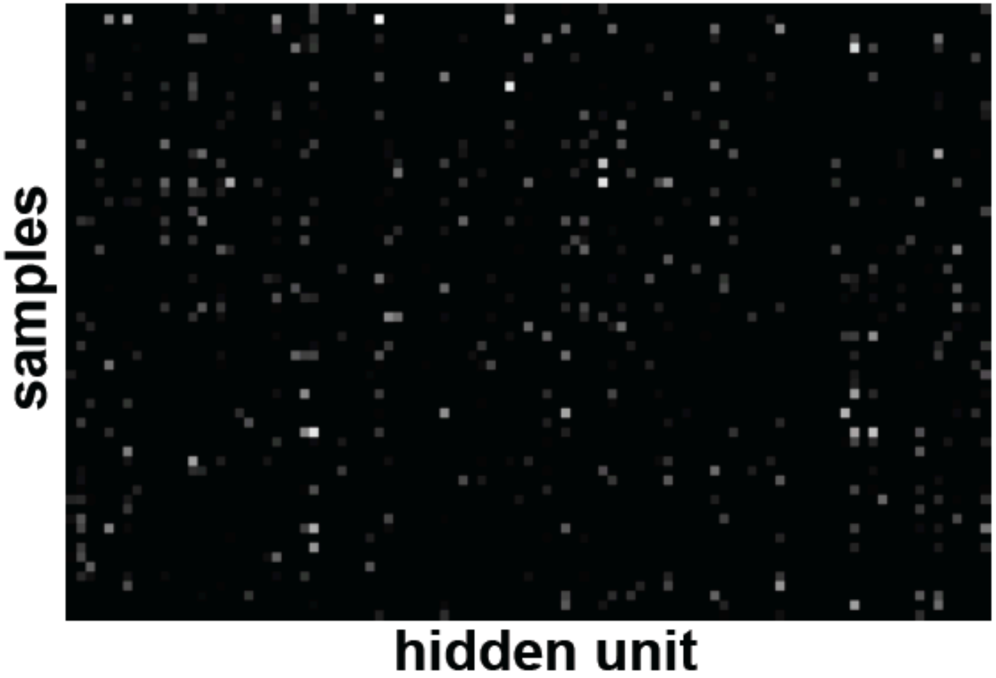


Figure 1. Evolved images that are unrecognizable to humans, but that state-of-the-art DNNs trained on ImageNet believe with $\geq$ 99.6% certainty to be a familiar object. This result highlights differences between how DNNs and humans recognize objects.

Slide Rob Fergus

OTHER THINGS GOOD TO KNOW

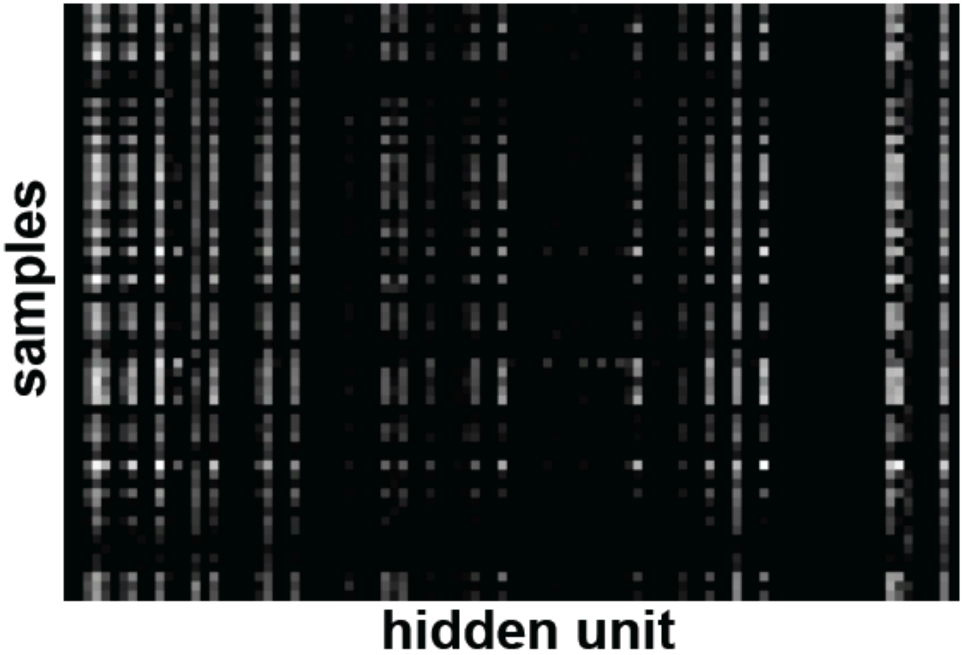
- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.



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

OTHER THINGS GOOD TO KNOW

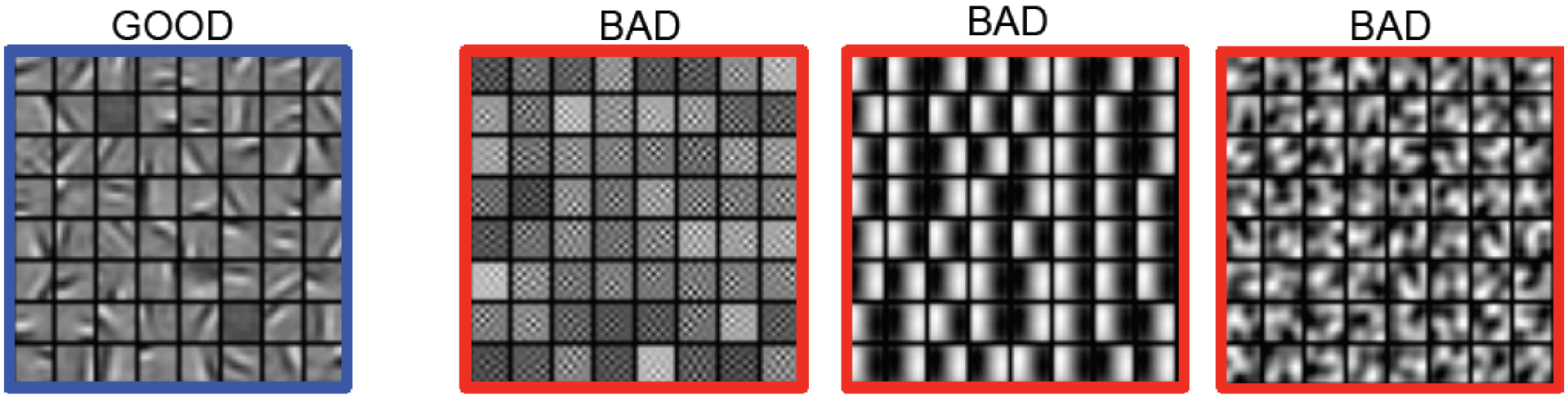
- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.



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

OTHER THINGS GOOD TO KNOW

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



GOOD

BAD

too noisy

BAD

too correlated

BAD

lack structure

Good training: learned filters exhibit structure and are uncorrelated.

OTHER THINGS GOOD TO KNOW

- Check gradients numerically by finite differences
- Visualize features (feature maps need to be uncorrelated) and have high variance.
- Visualize parameters
- Measure error on both training and validation set.
- Test on a small subset of the data and check the error → 0.