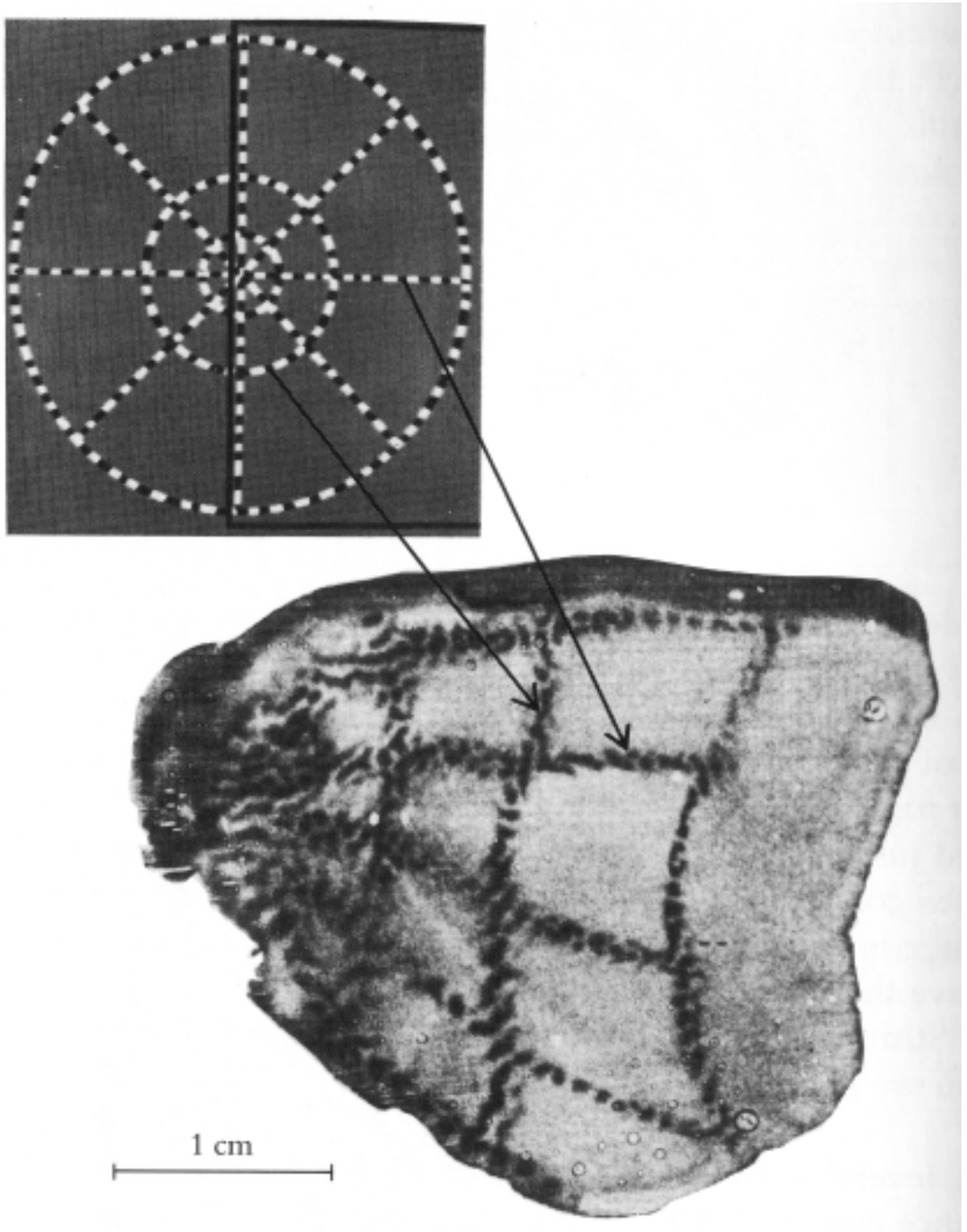
Learning in Convolutional Neural Networks (CNNs)

Retinotopic Map

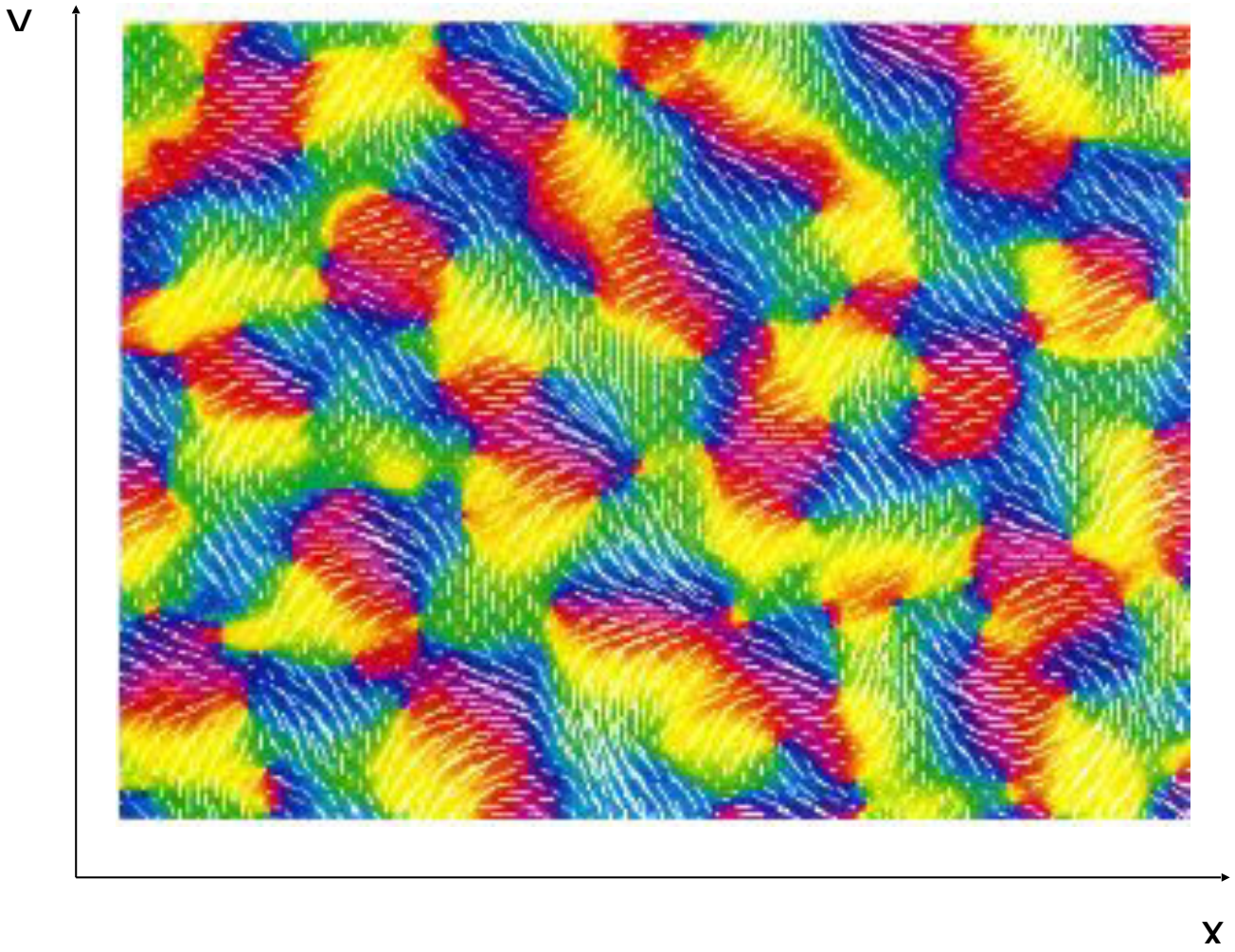
A flickering visual pattern excites cells across the visual field

Each mini dark region represents 150,000 cells



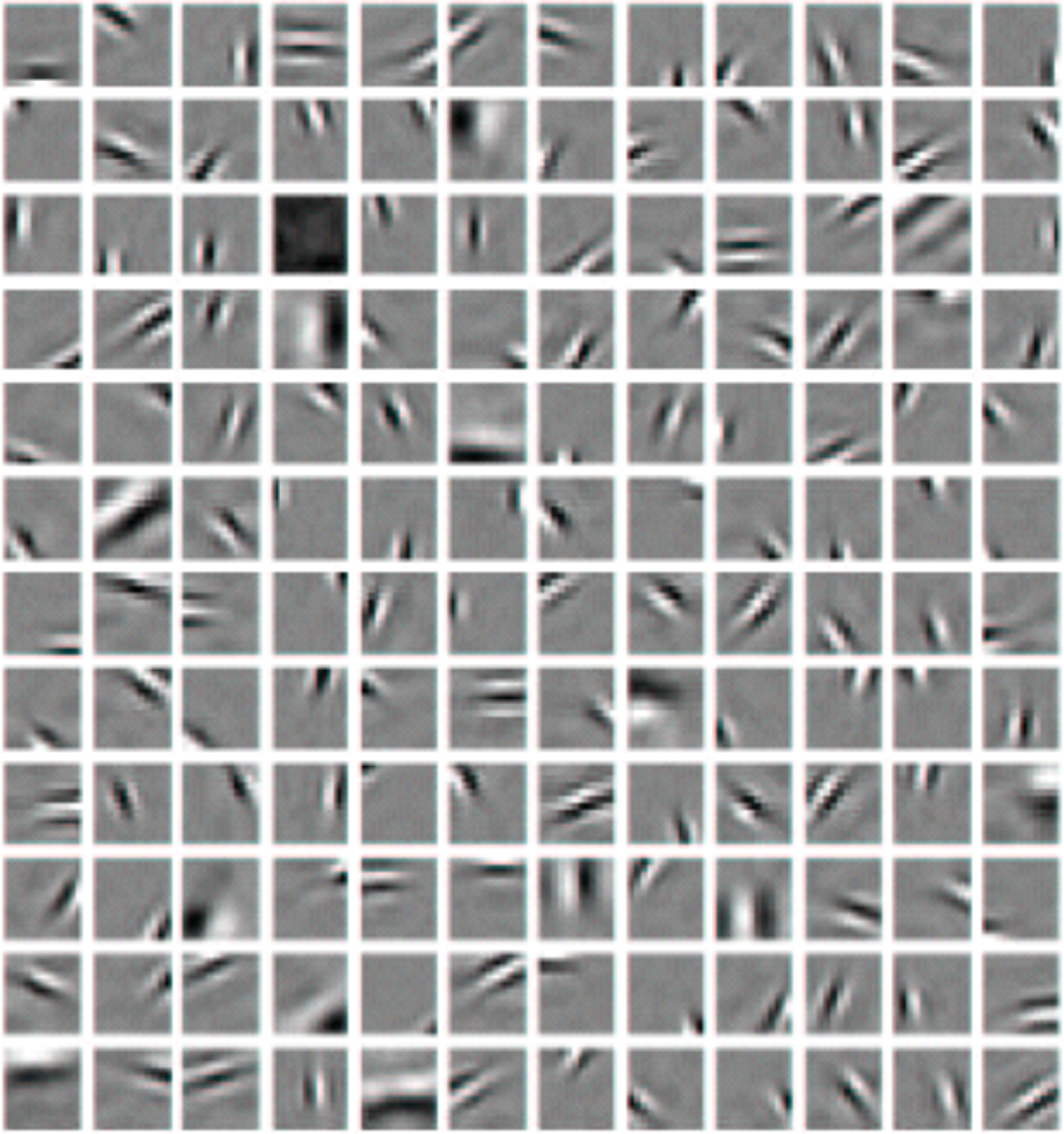
Orientation Maps in Striate Cortex

Across visual coordinates, cells responding to local photometric “edges” at different orientation (here denoted by different colors) are clumped into groups



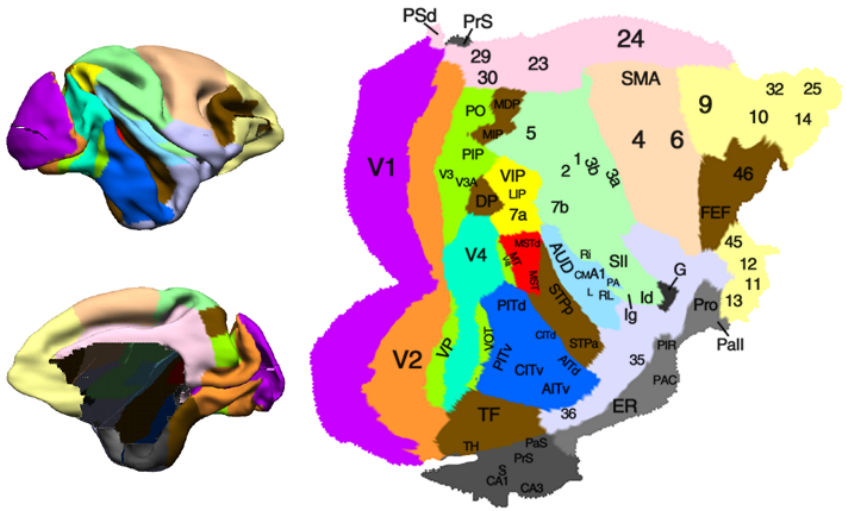
Different features

Receptive fields (here 144) of cells at a given location can be learned

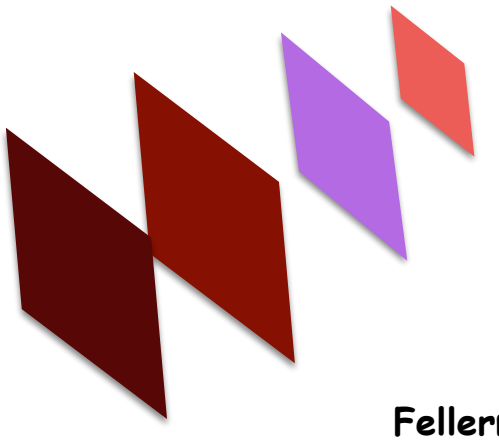


Flattened Representation of Monkey Cerebral Cortex

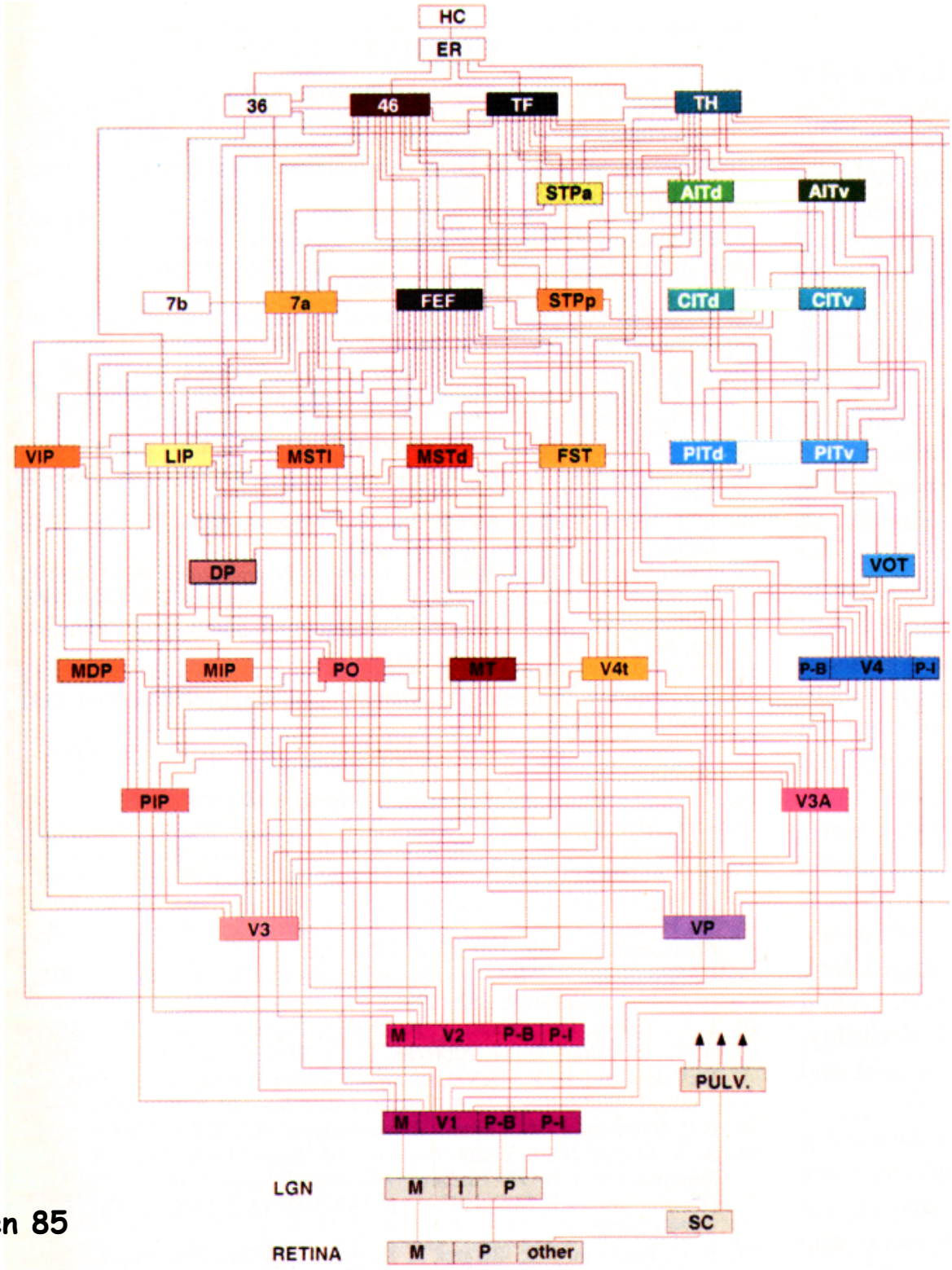
Van Essen and Drury 1998



Successive maps are connected to each other

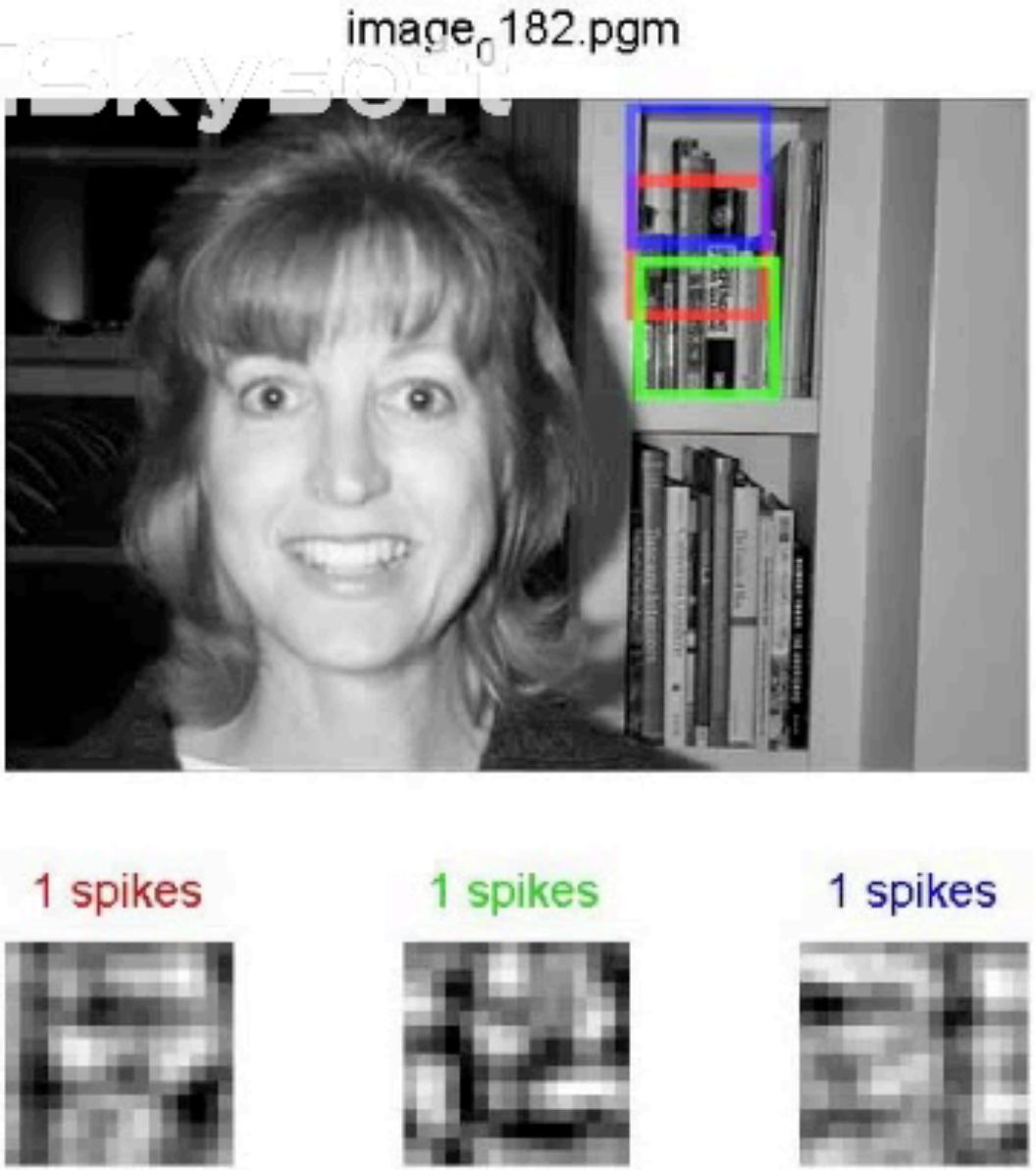


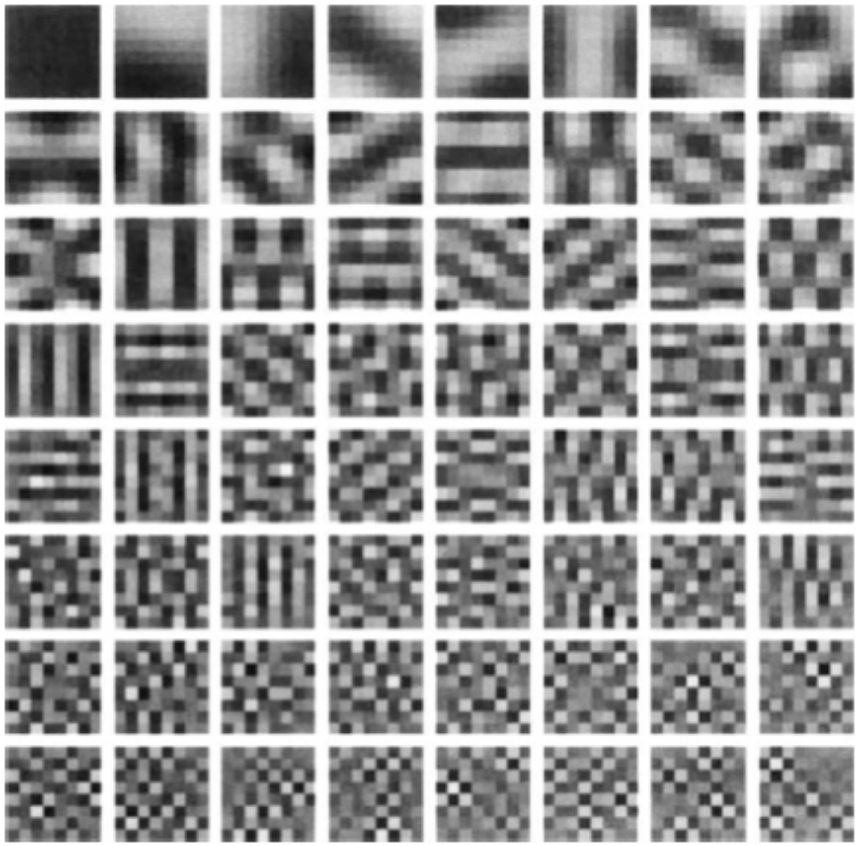
Fellerman and Van Essen 85



Bio-inspired convolution-like algorithm learns receptive fields.

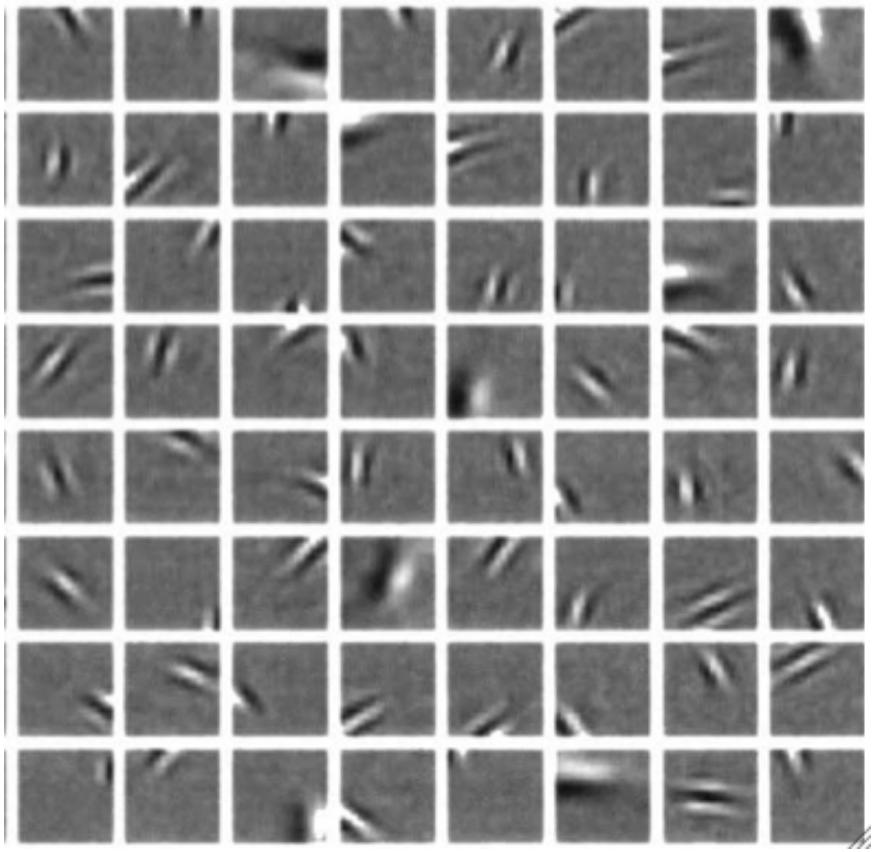
1. Templates tile the image
2. Best match template adds its input into itself and all its copies





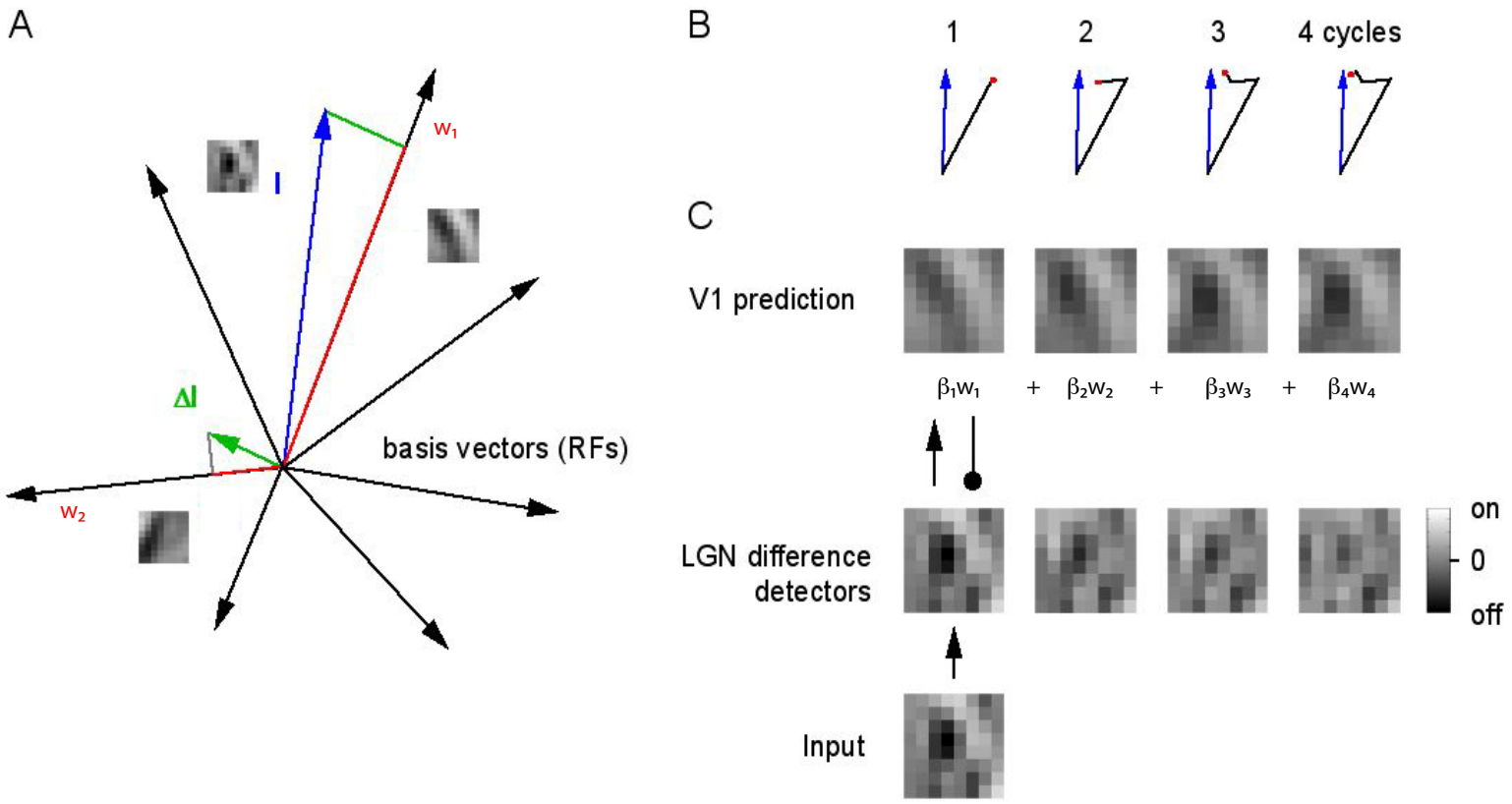
$$\min_{\boldsymbol{r}} \quad ||\boldsymbol{I} - U\boldsymbol{r}||_2$$

Learn receptive fields
by minimizing the fit only.

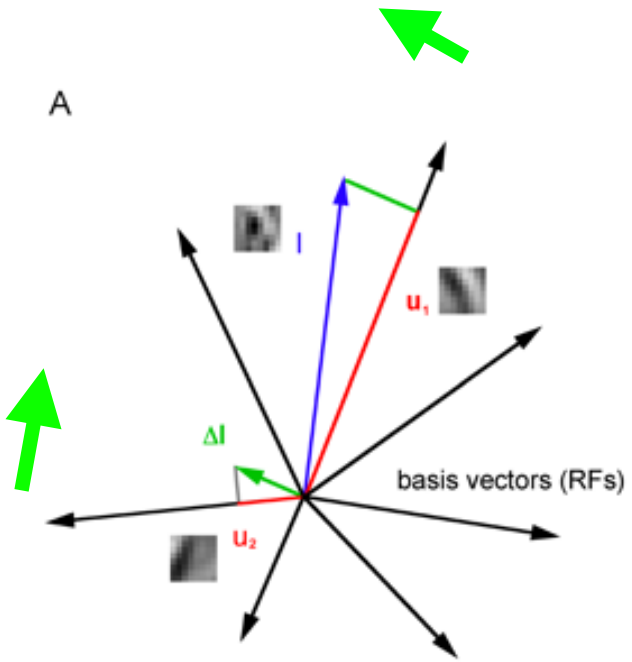


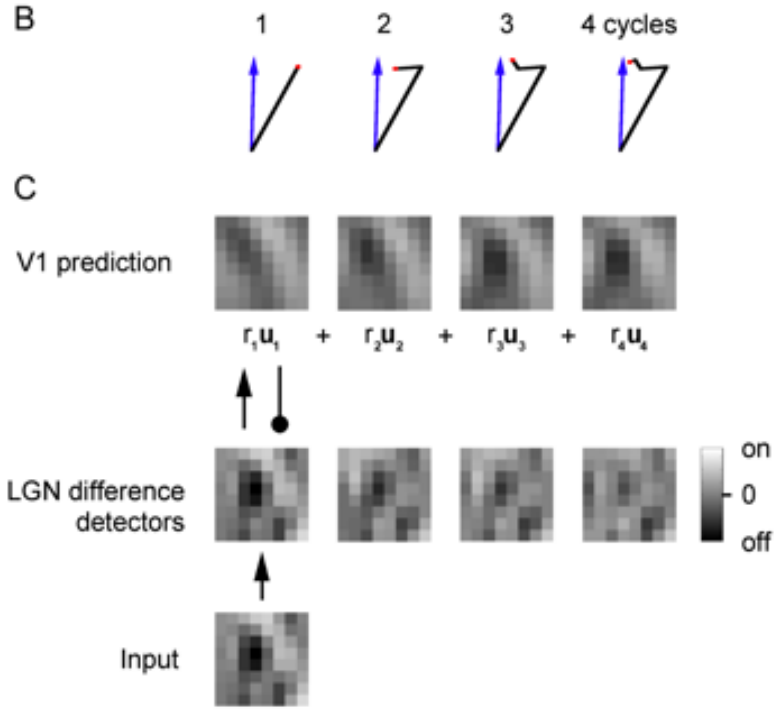
$$\min_{\boldsymbol{r}} \quad ||\boldsymbol{I} - U\boldsymbol{r}||_2 + \lambda||\boldsymbol{r}||_1$$

Learn receptive fields
by adding a term that penalizes large synapses
$||_1$ means absolute value


A
I
w1
ΔI
w2
basis vectors (RFs)
B
1
2
3
4 cycles
C
V1 prediction
β1w1 + β2w2 + β3w3 + β4w4
LGN difference detectors
on
0
off
Input

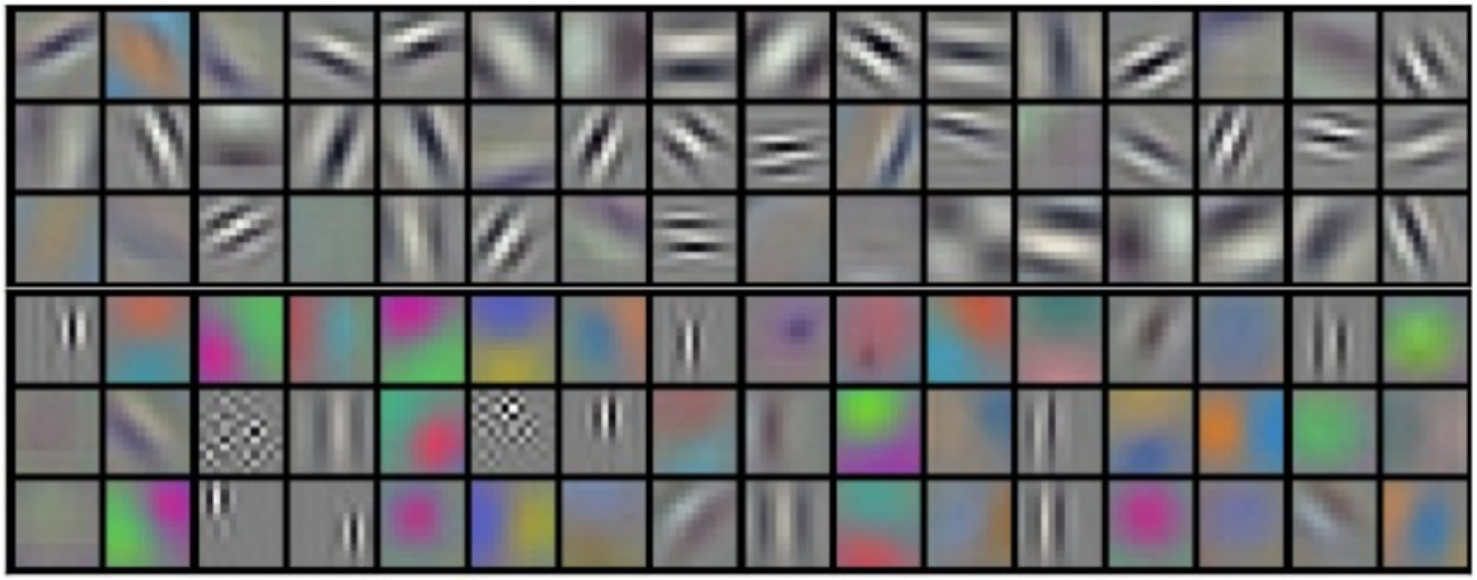
A standard Algorithm: Matching Pursuit



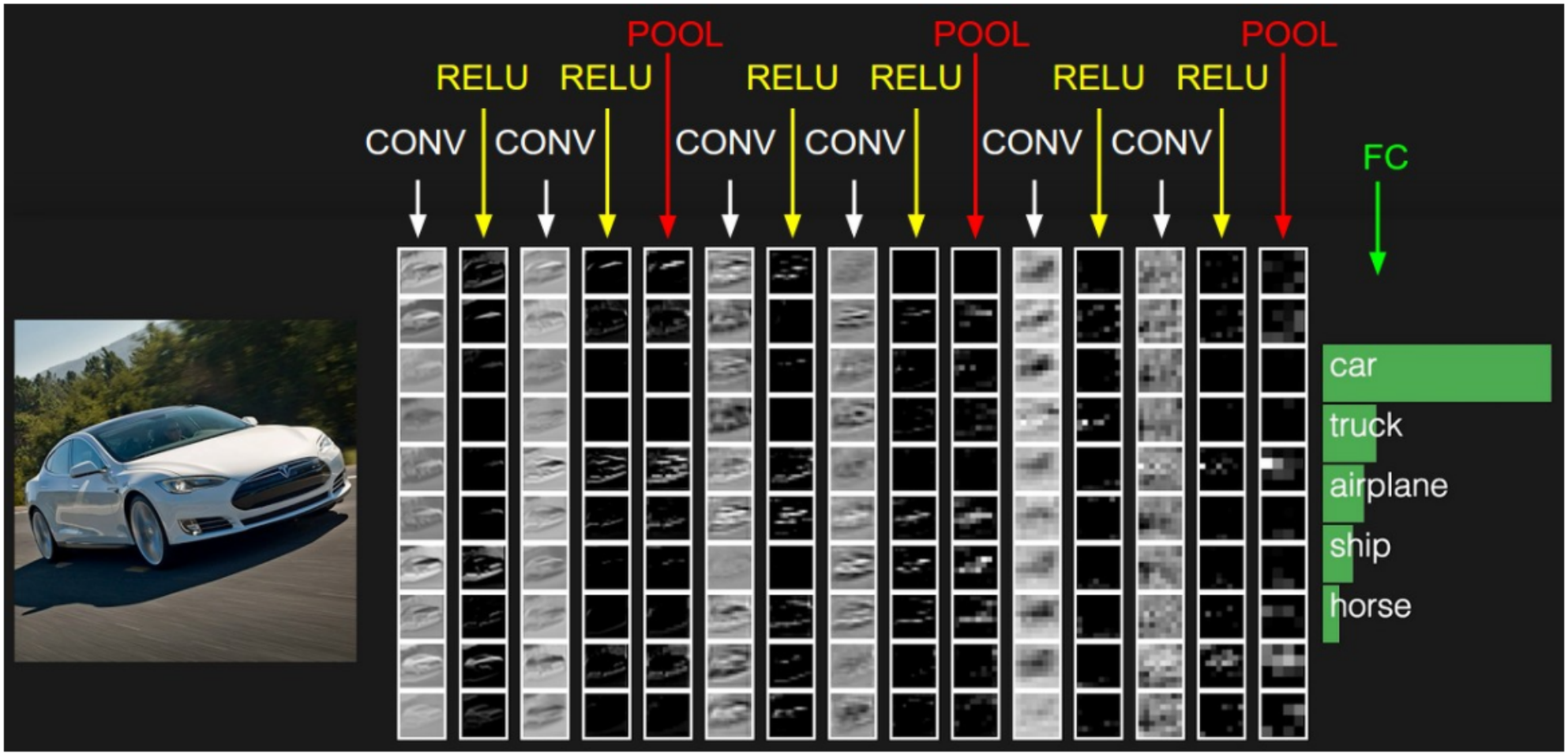


Learning algorithm: move RFs of winning neurons towards inputs

Learned filters



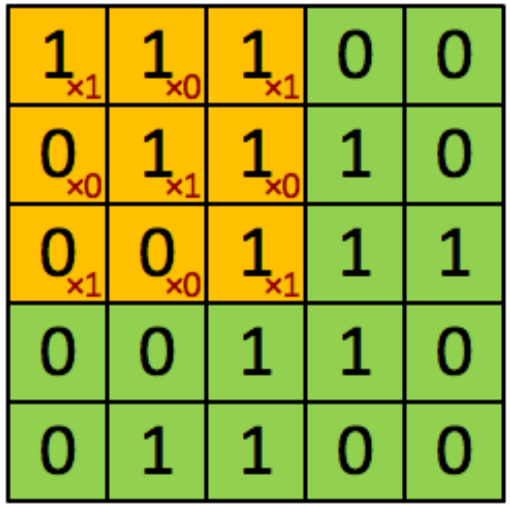
Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.


POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

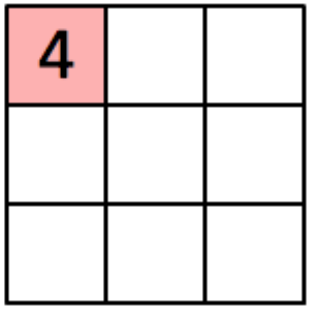
How convolution operator works

1	0	1
0	1	0
1	0	1

Convolution Template



Image



Convolved Feature

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	1	1	1	2	2	0
0	2	0	2	1	1	0
0	1	2	2	2	1	0
0	1	0	2	1	1	0
0	0	1	1	1	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	0	2	2	0	0
0	0	0	1	2	2	0
0	1	0	2	2	1	0
0	0	0	2	0	2	0
0	0	2	2	2	2	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	1	2	0	0	2	0
0	2	0	0	1	0	0
0	1	0	0	2	2	0
0	2	1	1	0	0	0
0	2	1	1	0	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	0	1
1	1	1
0	-1	0

w0[:,:,1]

1	1	0
0	-1	-1
-1	-1	-1

w0[:,:,2]

0	0	0
-1	1	1
-1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

-1	0	-1
1	-1	1
-1	1	-1

w1[:,:,1]

1	0	1
1	1	-1
-1	1	0

w1[:,:,2]

-1	0	0
1	1	0
-1	0	-1

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

0	-7	1
0	1	3
3	3	5

o[:,:,1]

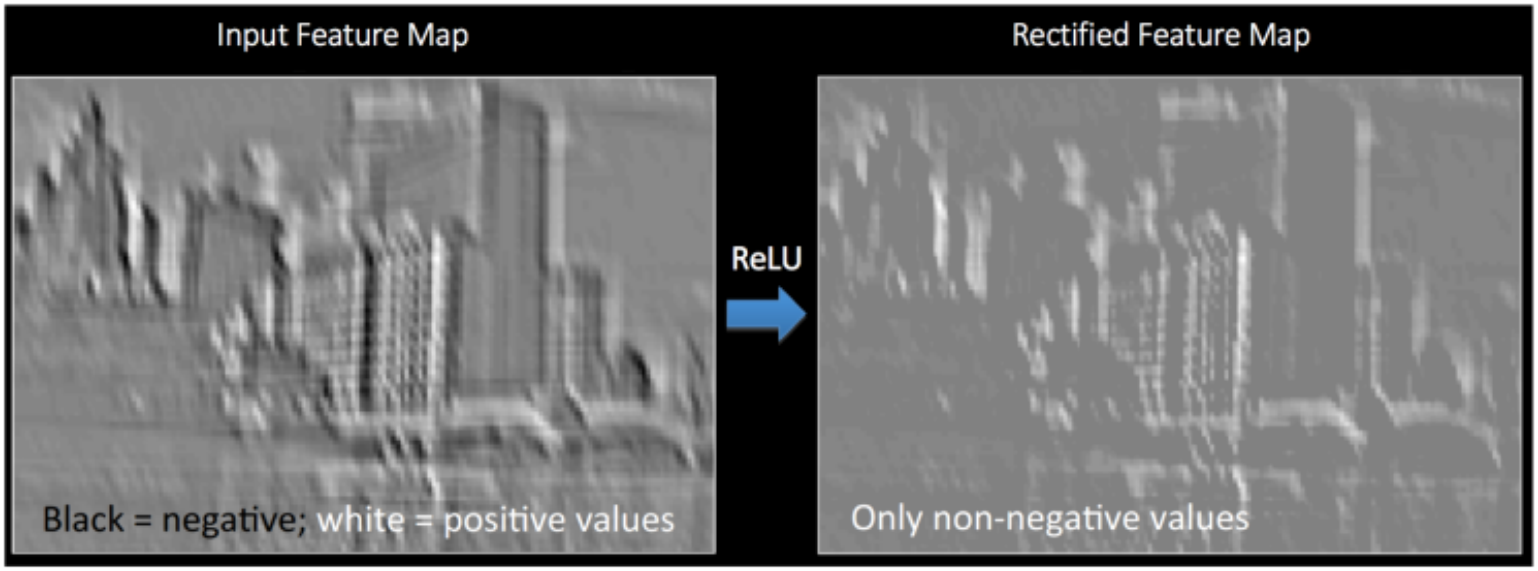
5	5	3
3	5	10
1	3	4

toggle movement

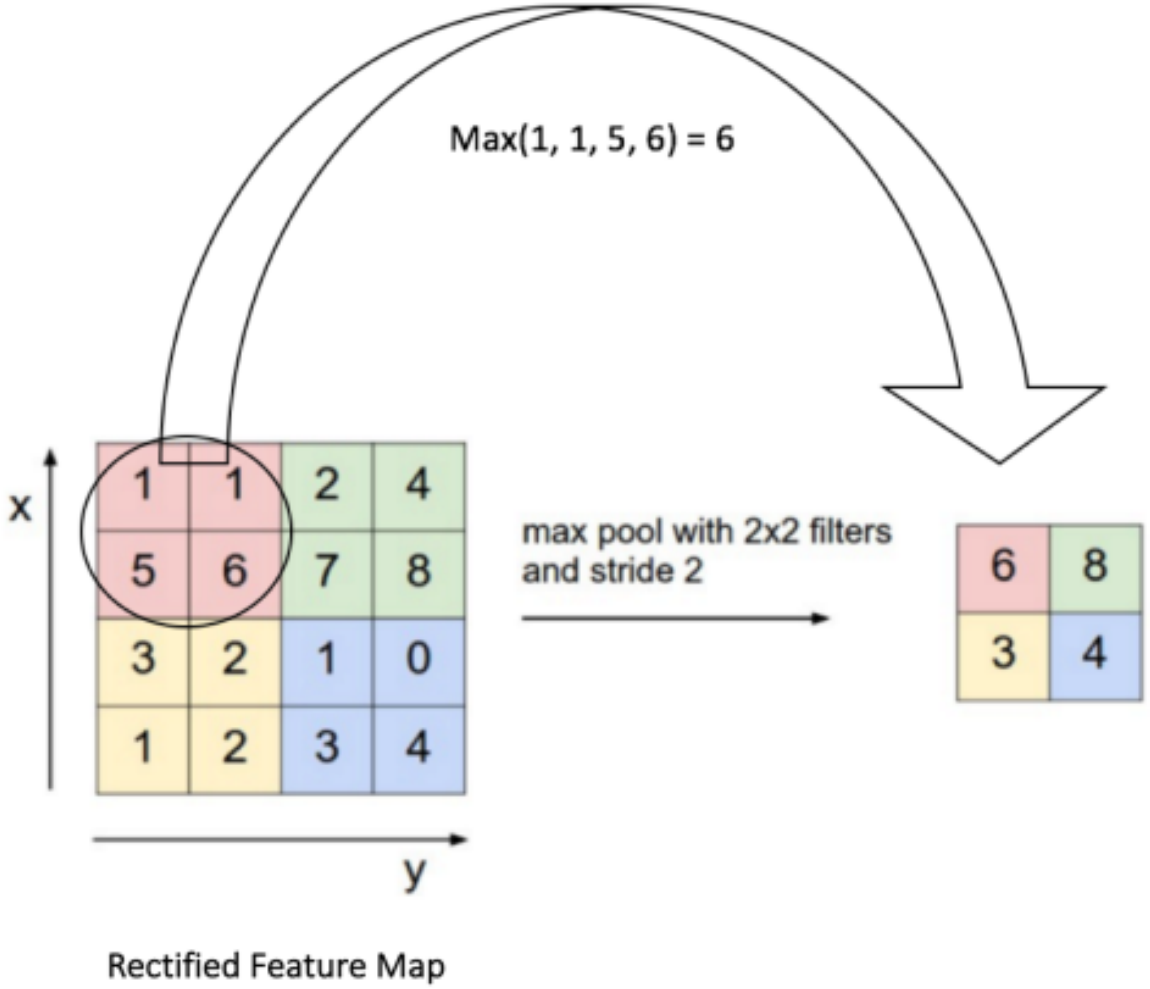
Example for a larger image with two feature templates

Figures taken from: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

RELUs [Rectified Linear Units] introduce nonlinearity



Stride and Max Pooling



Backpropagation

To keep things simple, let us just work with one pattern. In that case the objective function is defined to be

$$E = \frac{1}{2}\|\boldsymbol{x}^K - \boldsymbol{d}\|^2 \tag{1}$$

where K is an index denoting the last layer in the network and d is the desired output.

The equation for updating the states is given by

$$\boldsymbol{x}^{k+1} = g(W^k \boldsymbol{x}^k) \tag{2}$$

where W^k is a matrix used to store the weights between layer k and layer $k+1$,

$$W^k = \begin{bmatrix} w_{11}^k & w_{12}^k & \dots \\ w_{21}^k & \ddots & \\ \vdots & & w_{nn}^k \end{bmatrix} = \begin{pmatrix} \boldsymbol{w}_1^k \\ \boldsymbol{w}_2^k \\ \vdots \\ \boldsymbol{w}_n^k \end{pmatrix}$$

and the special understanding we shall have is that the function g applied to a vector is just that function applied to its elements; that is,

$$g(W\boldsymbol{x}) = \begin{pmatrix} g(\boldsymbol{w}_1 \cdot \boldsymbol{x}) \\ g(\boldsymbol{w}_2 \cdot \boldsymbol{x}) \\ \vdots \end{pmatrix}$$

This is just a version of the optimal control problem where Equation 1 is the optimand and Equation 2 is the dynamics. Here instead of the control $\boldsymbol{u}$, the matrices $W^k, k = 1, \dots, K$ represent the "control" variables. One difference, of course, is the dimensionality: In networks the dimension of the state space may range into the thousands! But the important thing to realize is that the mathematical treatment is the same.

Generating the Solution

To generate the solution, assume an initial W matrix. Then solve the dynamic equations going forward in levels in the network. This solution provides a value for $\boldsymbol{x}^K$. This value allows the adjoint system to be solved backward for $k = K$ to $k = 0$:

$$\boldsymbol{\lambda}^K = \boldsymbol{x}^K - \boldsymbol{d} \text{ for } k = K$$

$$\boldsymbol{\lambda}^k = -W^T \Lambda^{k+1} g'(W\boldsymbol{x}^k) \text{ for } k = 0, \ldots, K-1$$

where g' denotes differentiation with respect to its argument. Now improve the estimate for W using gradient descent. To do so, calculate the adjustment for the weights as $\frac{\partial H}{\partial w_{ij}}$:

$$\frac{\partial H}{\partial w_{ij}^k} = \lambda_i^{k+1} x_j^k g'(\boldsymbol{w}_i^k \cdot \boldsymbol{x}^k) \tag{5}$$

Equation 5 is very compact and so is worth unpacking in order to understand it. The multiplier λ_i^{k+1} can be understood by realizing that at level K it just keeps track of the output error. This is its role for the internal units also; it appropriately keeps track of how the internal weight change affects output error. The derivative g' is also simple, especially when using

$$g(u) = \frac{1}{1 + e^{-u}}$$

since it can be easily verified that

$$g'(u) = g(1 - g)$$

Thus to calculate g' for a given unit, just determine how much input the x_j^kth unit receives, and apply that as an argument to $g'()$.

Biological features that have inspired CNN architecture are mostly based on how the brain's cortex handles vision

Retinopathy: visual features are represented across the visual field *explicitly*

Specific properties: visual features are *repeated* across the visual field explicitly

Feature maps are connected to a layer above and down sampled to create a hierarchy of maps with properties of increasing abstraction

All these properties are incorporated into CNNs

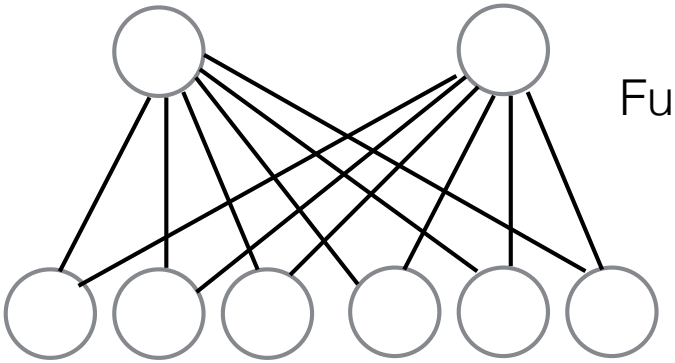
Now back to Backpropagation

For the fully connected network …

For the weights: $$\Delta w_{ij}^k \propto \lambda_i^{k+1} x_j^k g'(\mathbf{w}_i^k \cdot \mathbf{x}^k)$$

For the errors:
$$\boldsymbol{\lambda}^K = \boldsymbol{x}^K - \boldsymbol{d} \text{ for } k = K$$
$$\boldsymbol{\lambda}^k = -W^T \Lambda^{k+1} g'(W\boldsymbol{x}^k) \text{ for } k = 0, \ldots, K-1$$

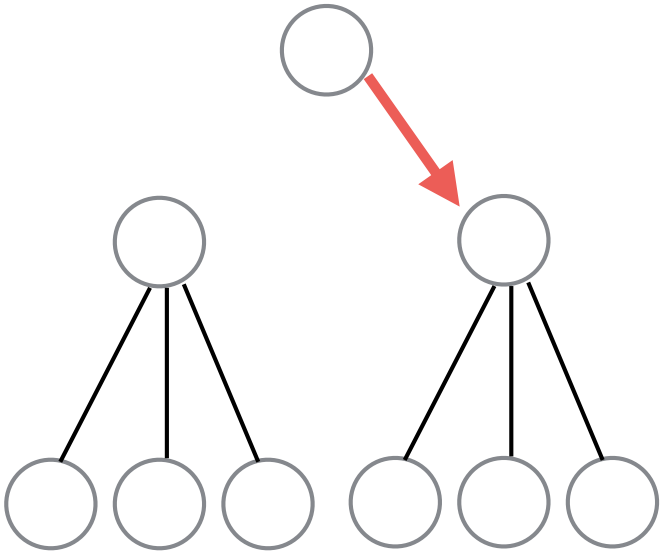
For the CNN … $\boldsymbol{\lambda}^k$ =upsample ($-W^T \Lambda^{k+1} g'(W\boldsymbol{x}^k)$)



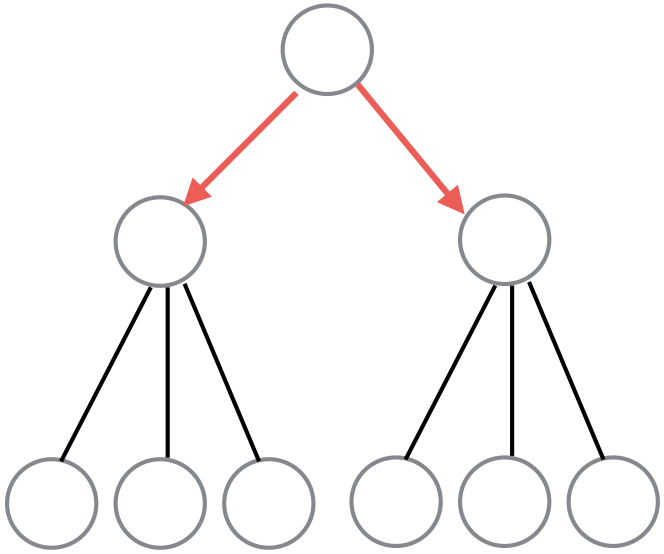
Fully-connected network

CNNs

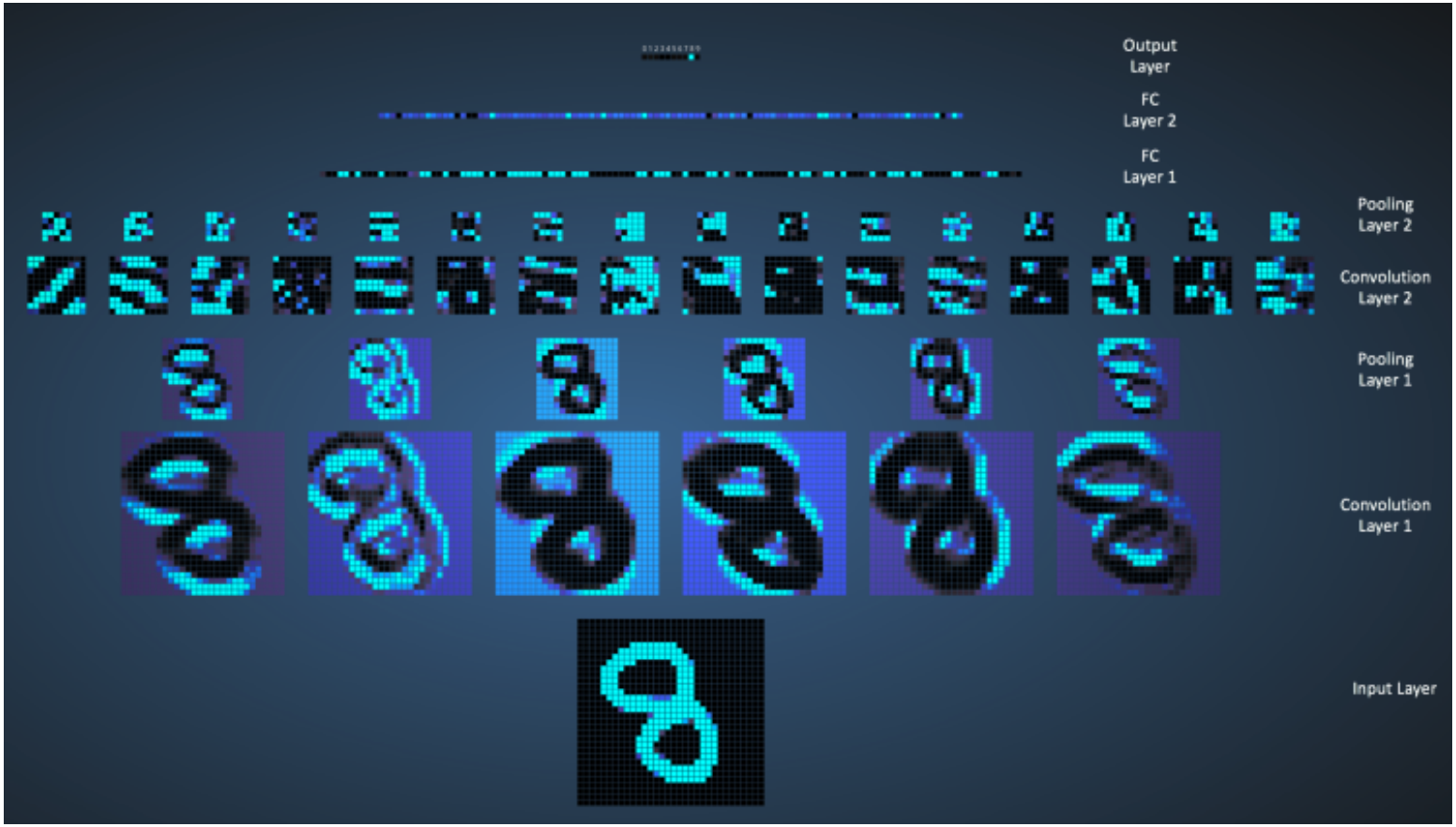
Max pooling



Averaging

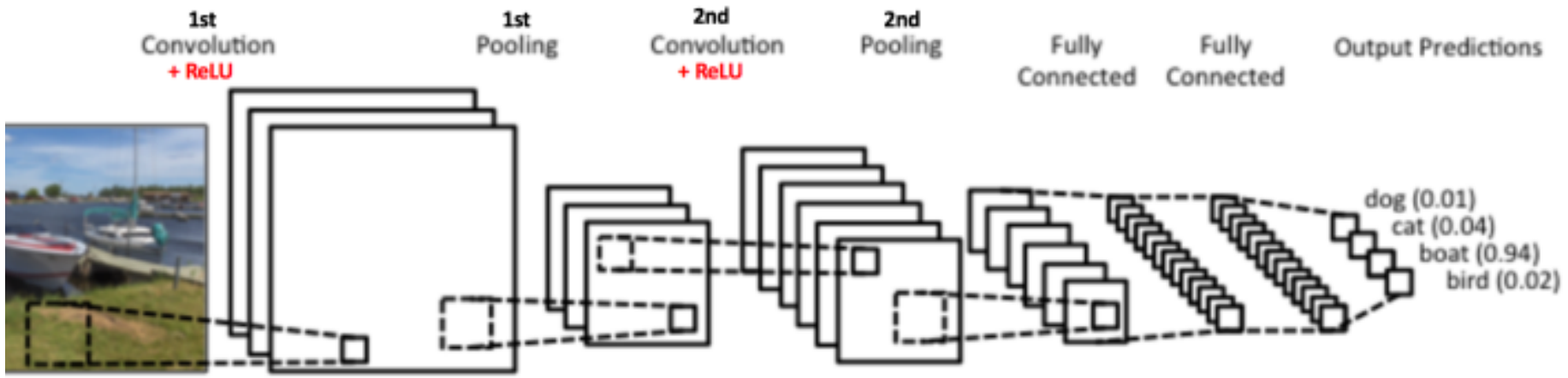


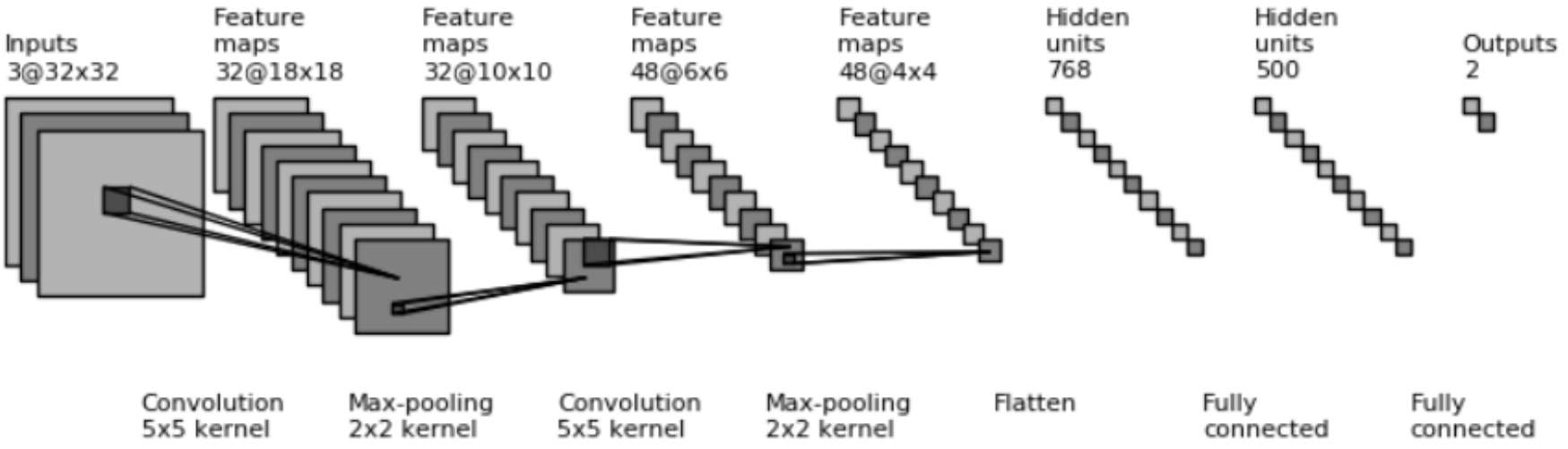
Visualizations of a deep neural network with CNNs



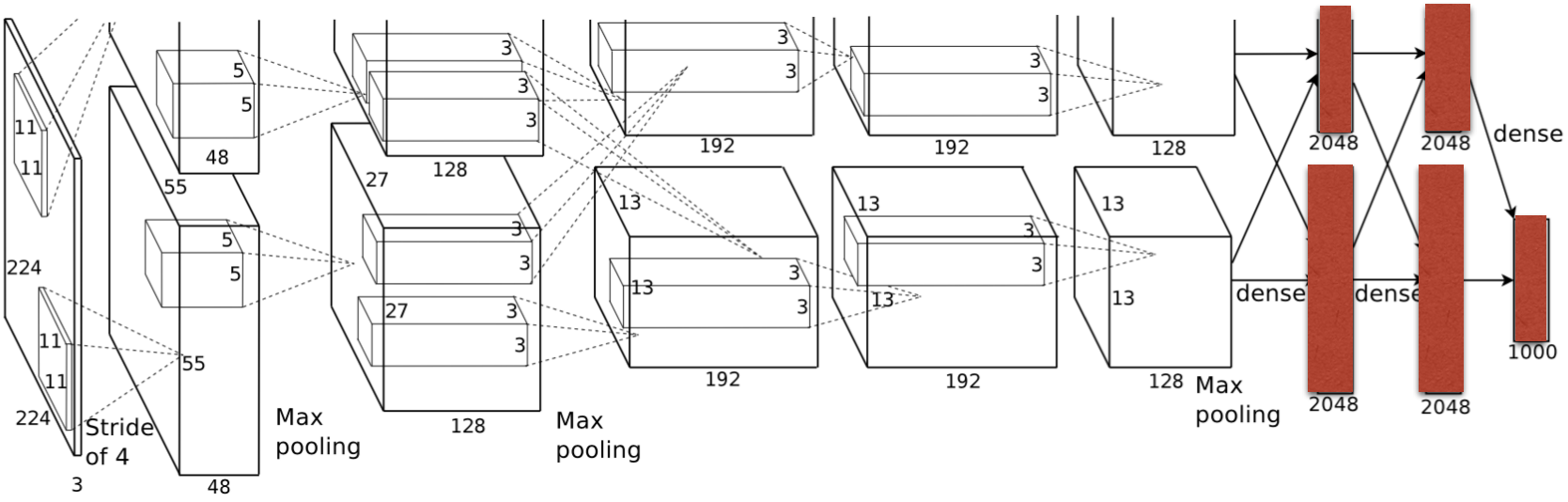
A. W. Harley, “An Interactive Node-Link Visualization of Convolutional Neural Networks,” in ISVC, pages 867-877, 2015

Summary: An example with two CNNs and two fully-connected layers




Inputs
3@32x32
Feature
maps
32@18x18
Feature
maps
32@10x10
Feature
maps
48@6x6
Feature
maps
48@4x4
Hidden
units
768
Hidden
units
500
Outputs
2
Convolution
5x5 kernel
Max-pooling
2x2 kernel
Convolution
5x5 kernel
Max-pooling
2x2 kernel
Flatten
Fully
connected
Fully
connected

A Deep Learning Network handcrafted for image classification



Convolution nets

regular
completely connected nets

Krizhevsky, A., Sutskever, I. and Hinton, G. E.
ImageNet Classification with Deep Convolutional Neural Networks
Advances in Neural Information Processing 25, MIT Press, Cambridge, MA