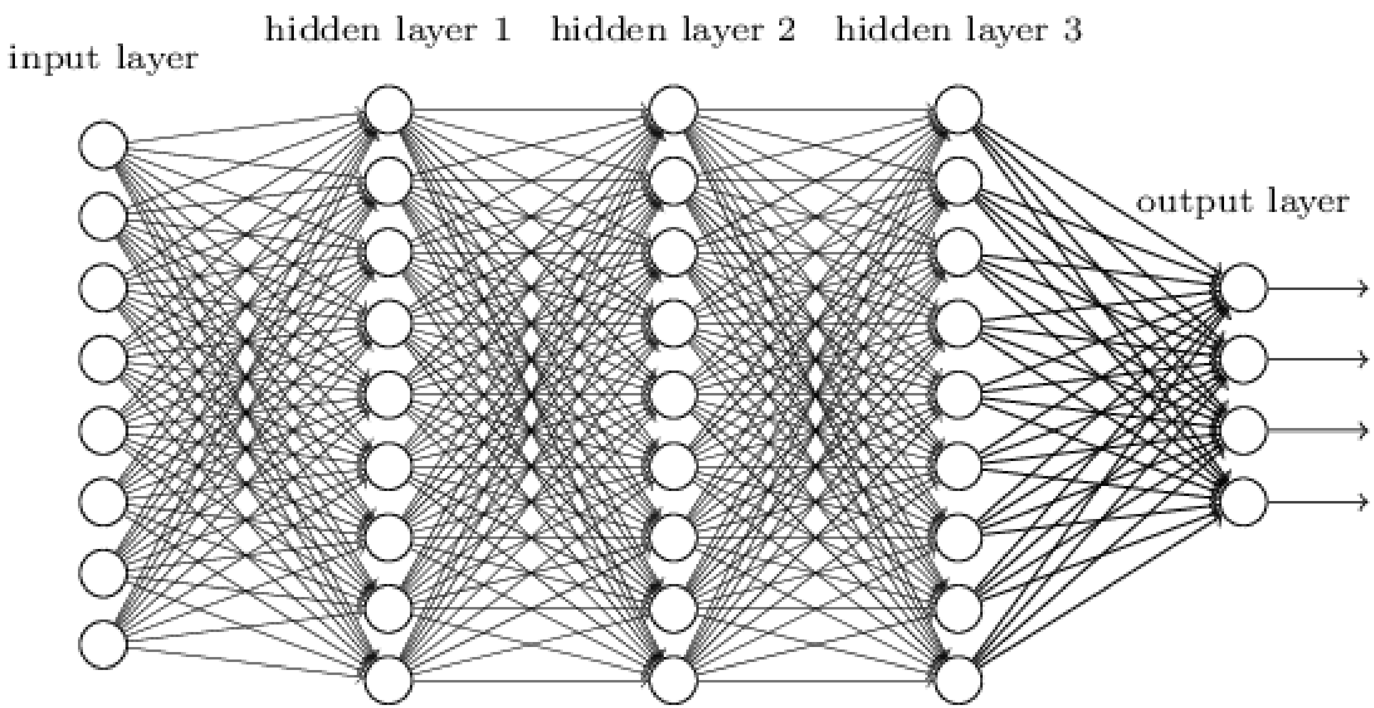
# Convolutional Neural Networks

Muhammad Atif Tahir

Some Slides from
Waterloo (CS)
Andrew Ng (Coursera)

# CNN

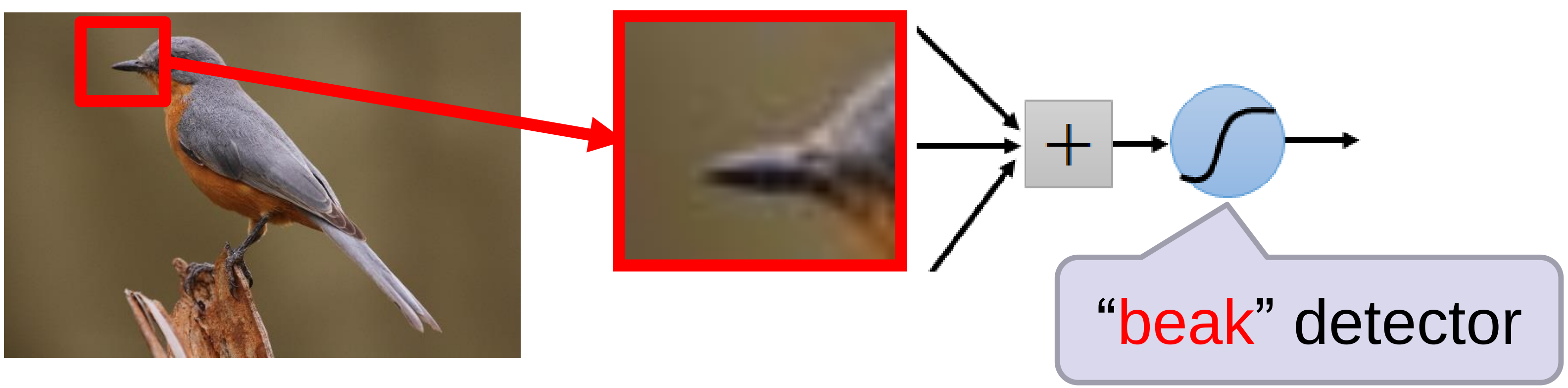
- We know it is good to learn a small model.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?



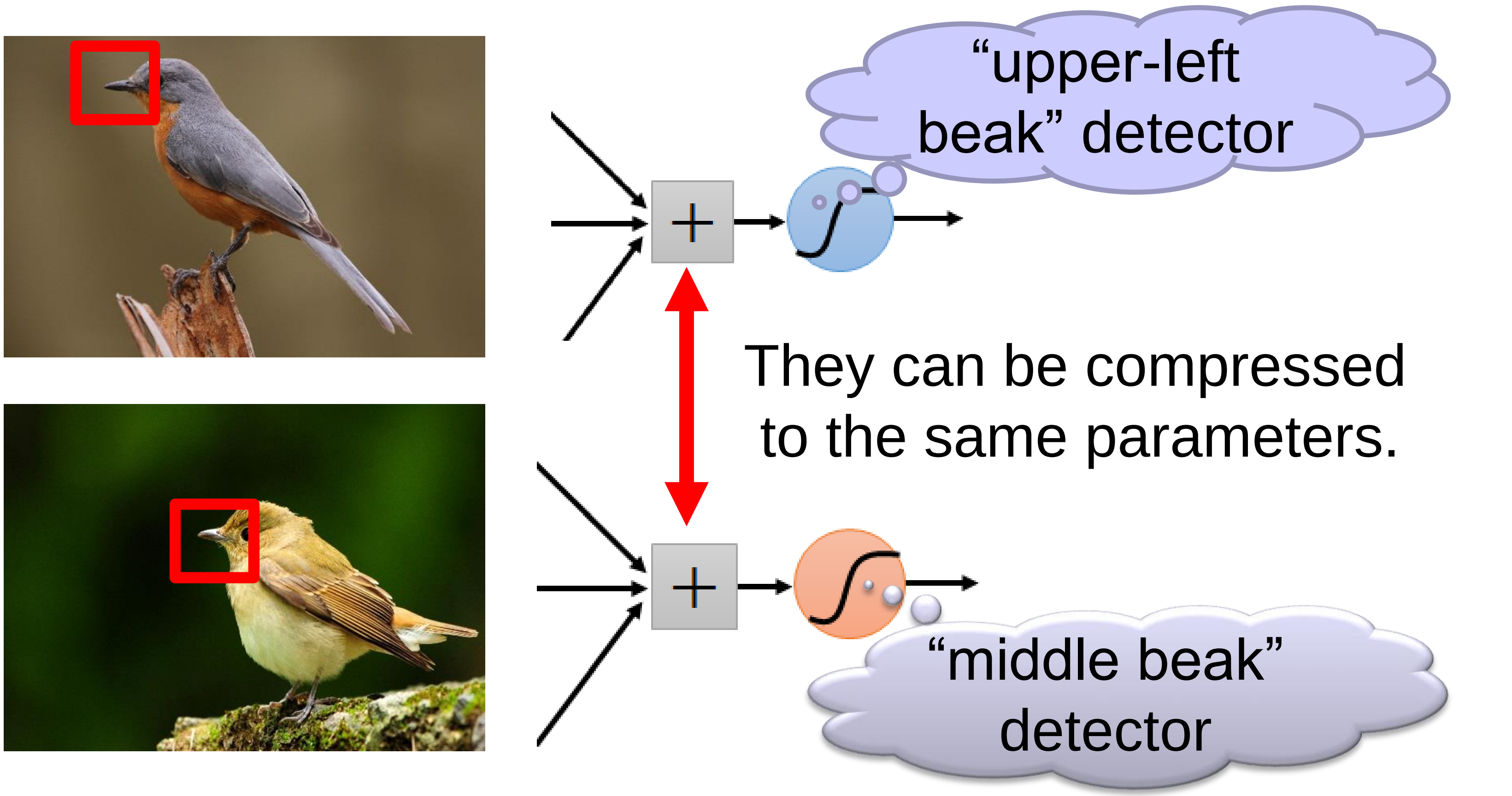
# Consider learning an image:

- Some patterns are much smaller than the whole image

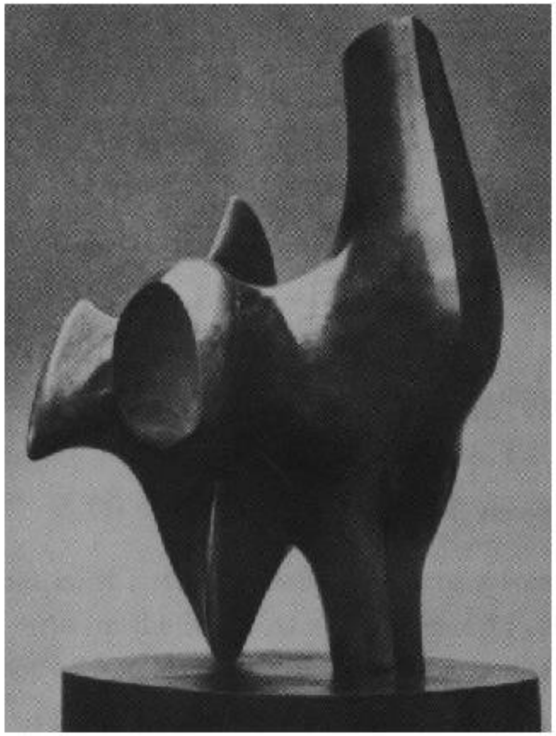
Can represent a small region with fewer parameters

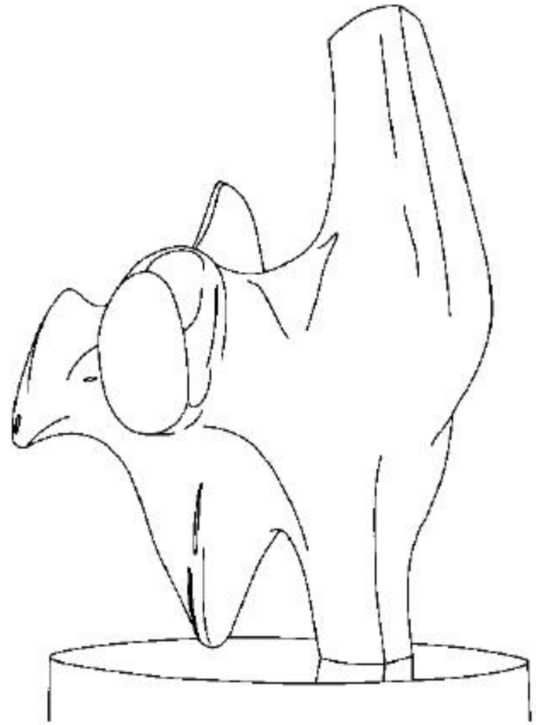


Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors and each detector must “move around”.



# Edge detection

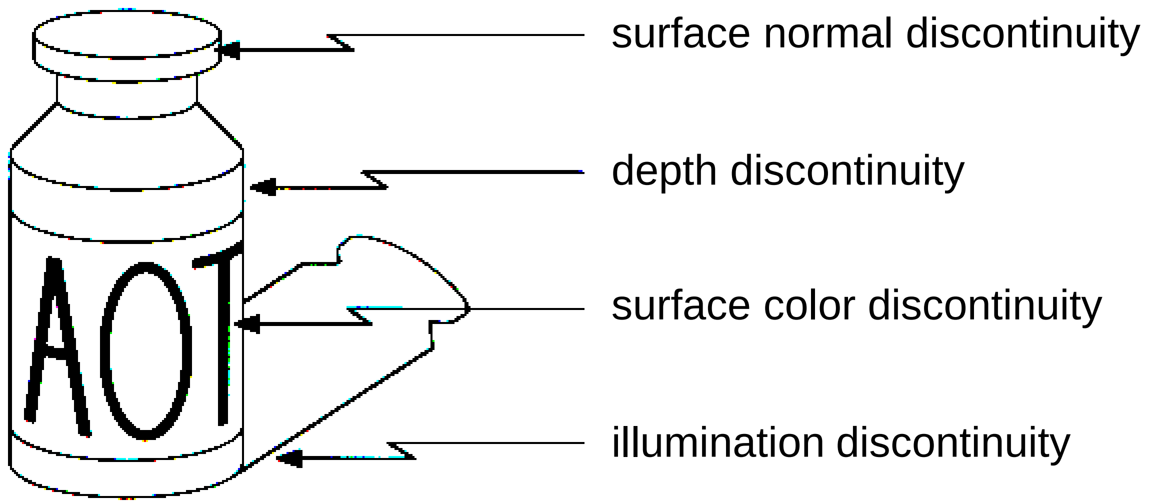




Convert a 2D image into a set of curves

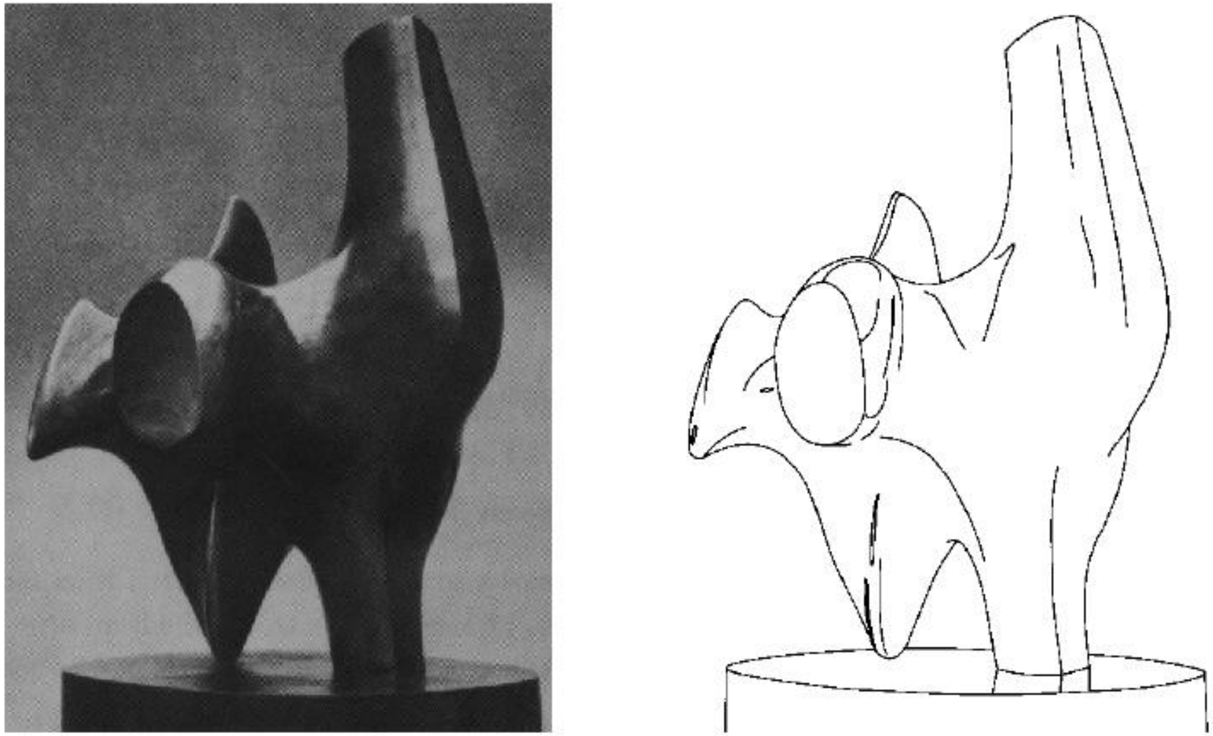
- Extracts salient features of the scene
- More compact than pixels

# Origin of Edges



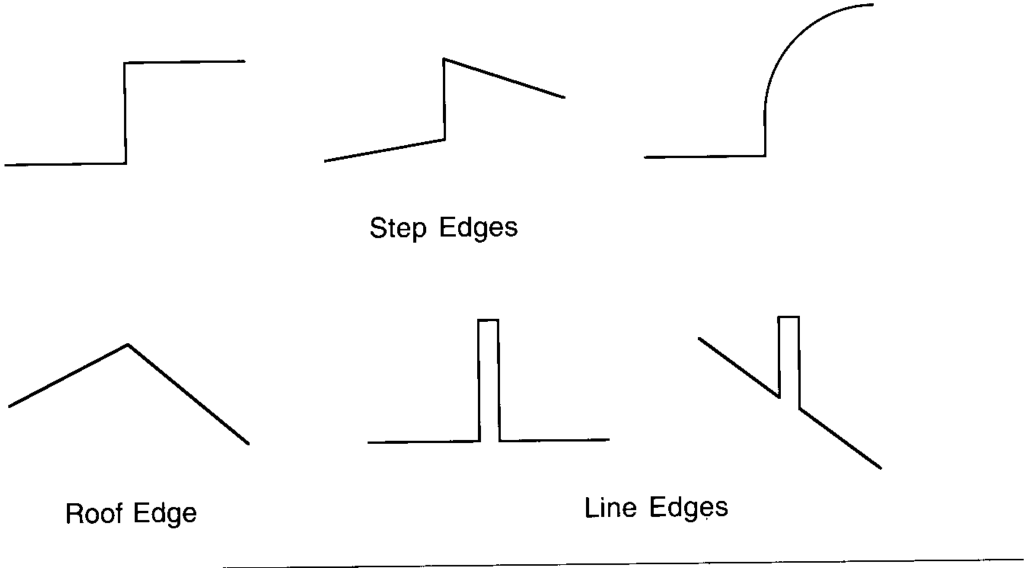

Edges are caused by a variety of factors

# Edge detection



How can you tell that a pixel is on an edge?

# Profiles of image intensity edges

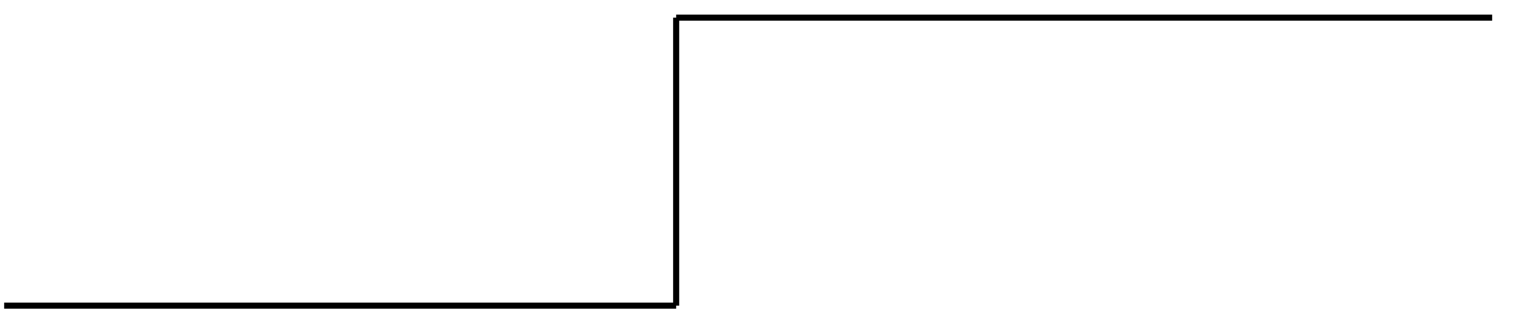


# Edge is Where Change Occurs

Change is measured by derivative in 1D

Biggest change, derivative has maximum magnitude

Or 2nd derivative is zero.

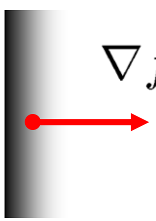


# Image gradient

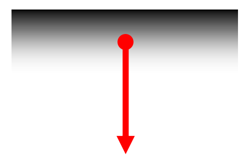
The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

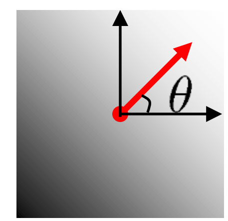
The gradient points in the direction of most rapid change in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$




$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient direction is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right)$$

- how does this relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

# Gradient operators

$$\Delta_1 = \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix} \qquad \Delta_2 = \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$$

(a)

$$\Delta_1 = \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \qquad \Delta_2 = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

(b)

$$\Delta_1 = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad \Delta_2 = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

(c)

$$\Delta_1 = \begin{matrix} -3 & -1 & 1 & 3 \\ -3 & -1 & 1 & 3 \\ -3 & -1 & 1 & 3 \\ -3 & -1 & 1 & 3 \end{matrix} \qquad \Delta_2 = \begin{matrix} 3 & 3 & 3 & 3 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -3 & -3 & -3 & -3 \end{matrix}$$

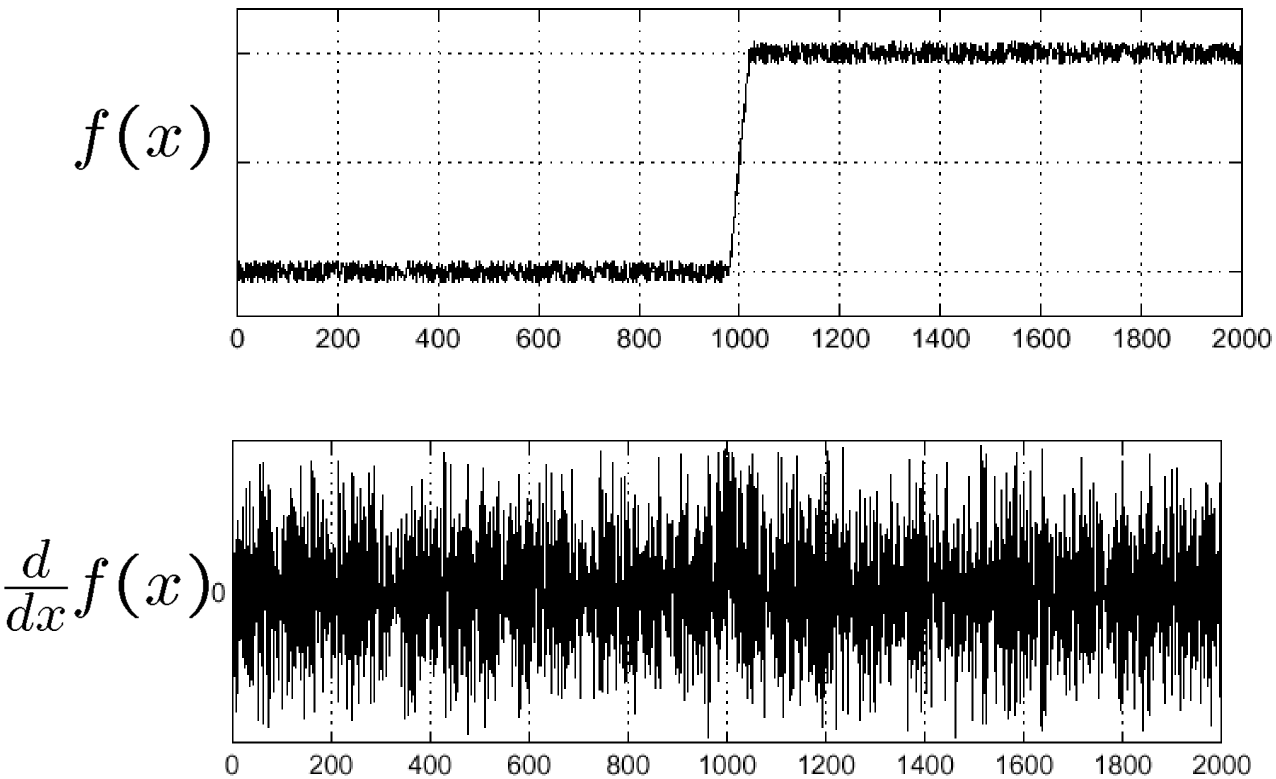
(d)

(a): Roberts' cross operator (b): 3x3 Prewitt operator
(c): Sobel operator (d) 4x4 Prewitt operator

# Effects of noise

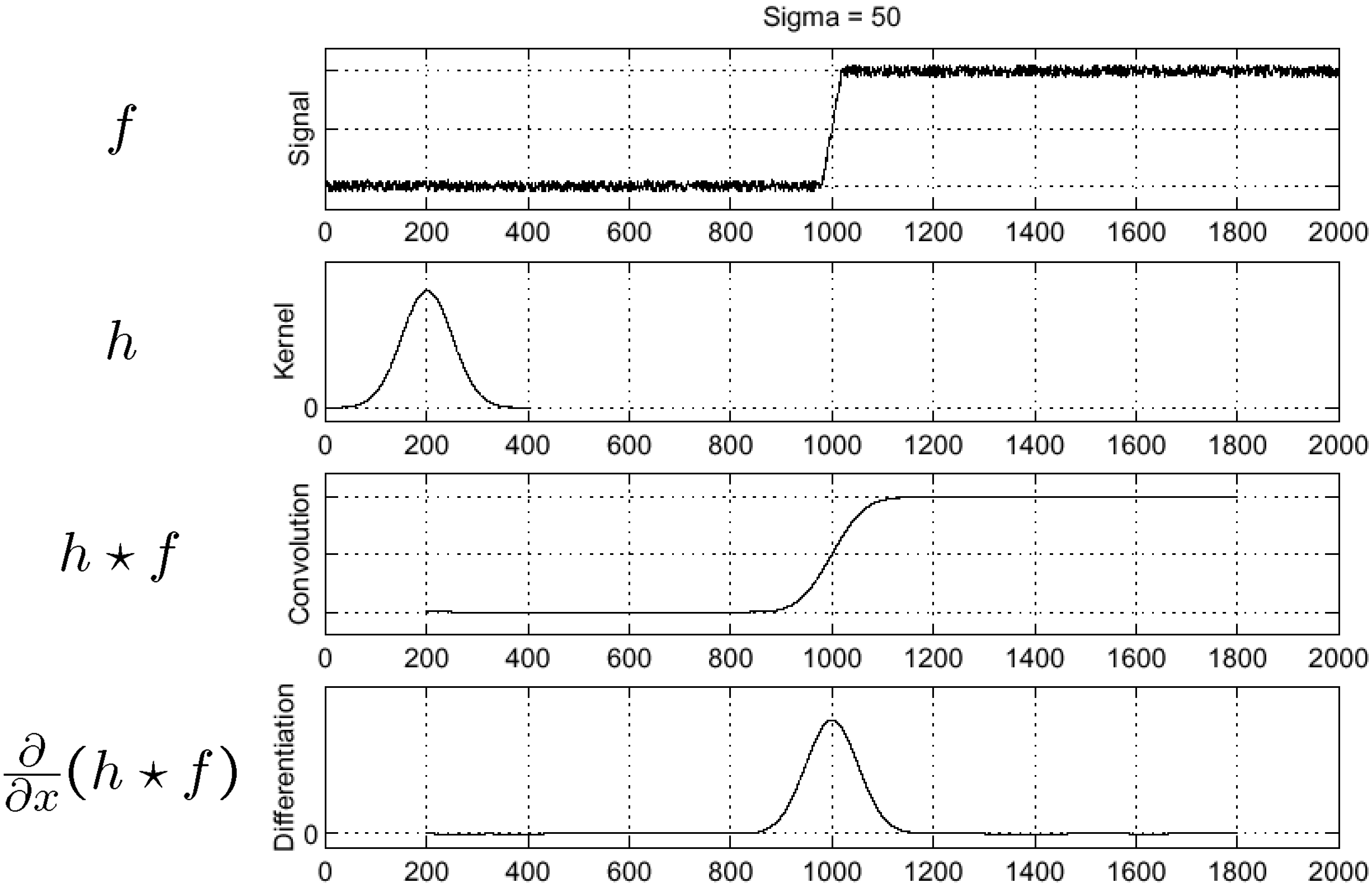
Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal




Where is the edge?

# Solution:  smooth first




Where is the edge?   Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

# Derivative theorem of convolution

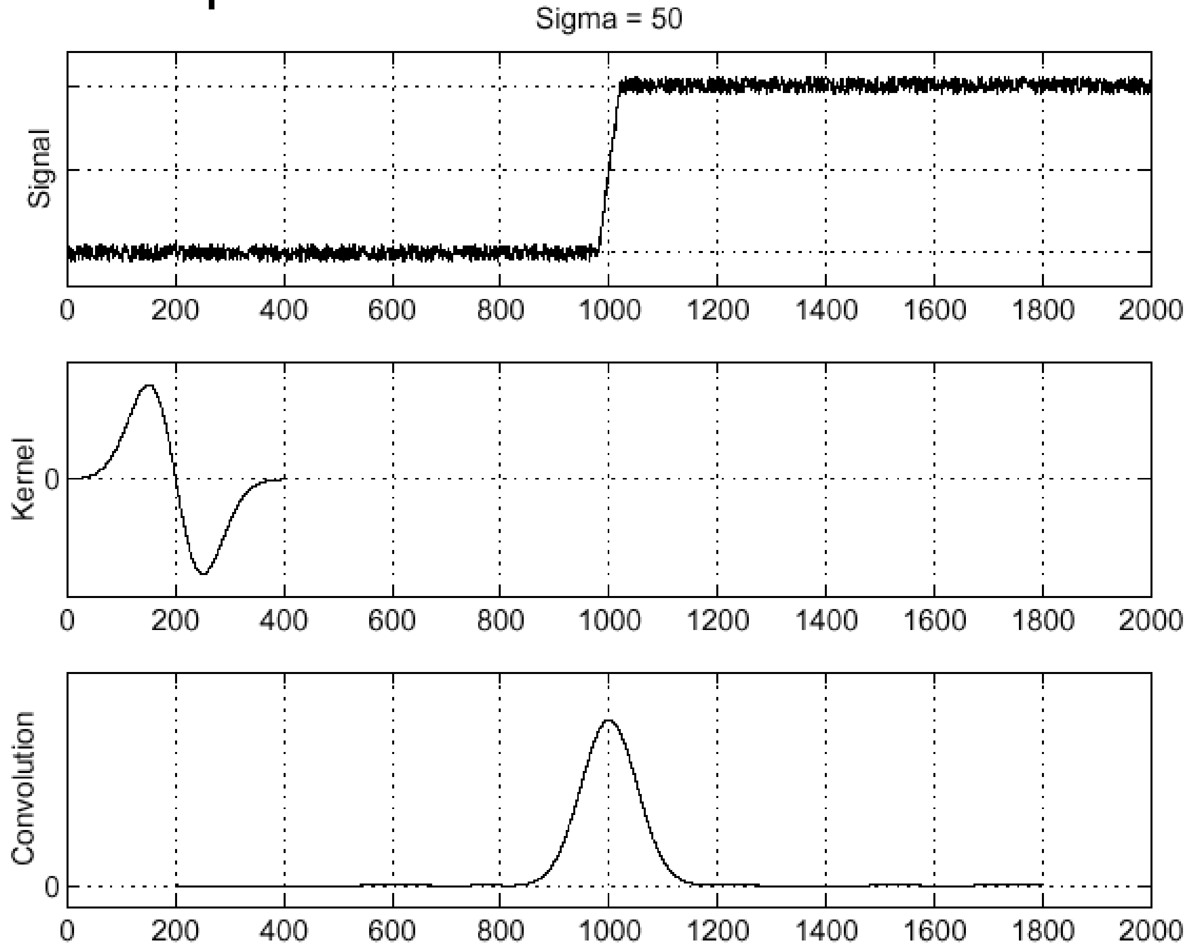
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

This saves us one operation:

$f$

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



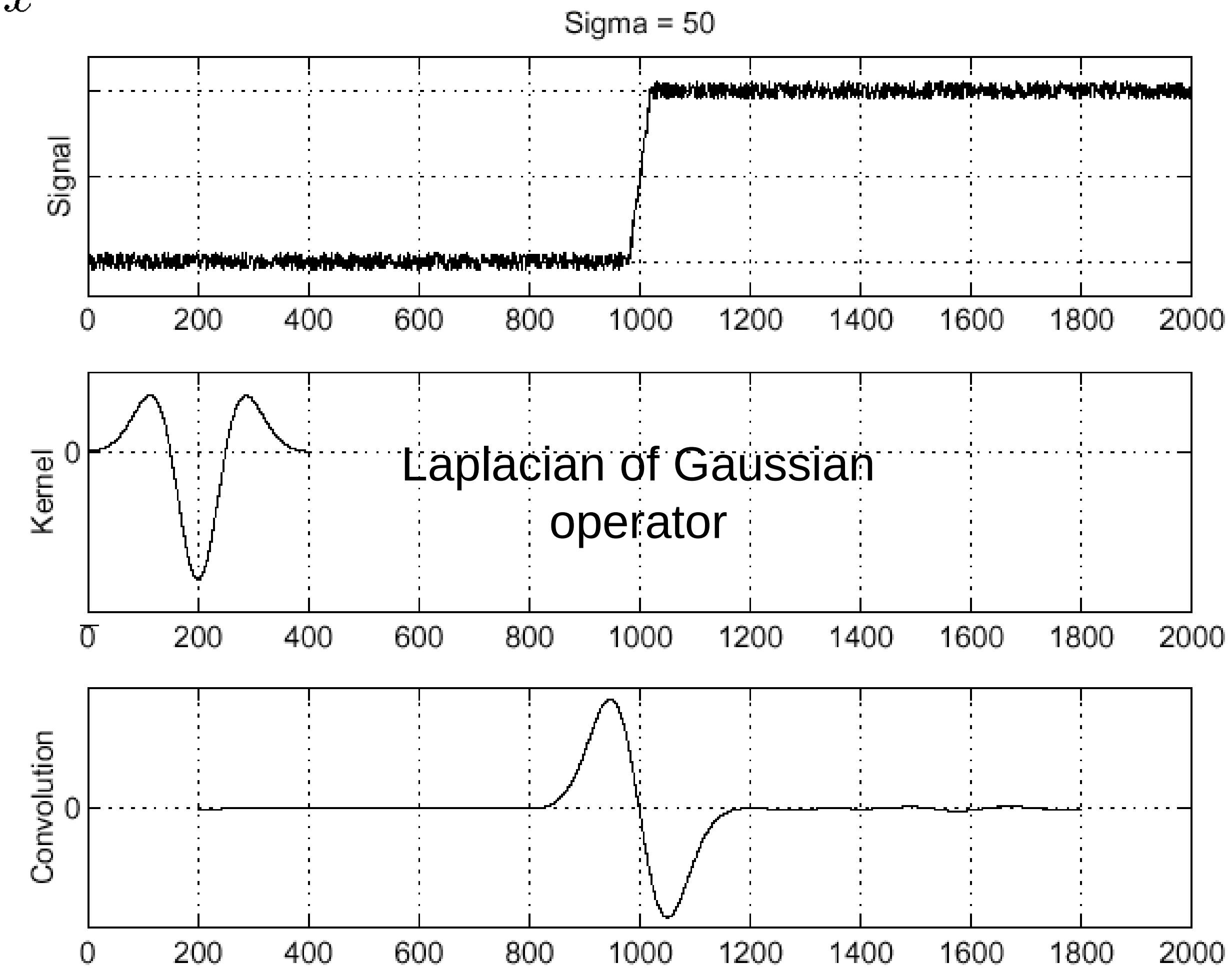
# Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

$f$

$\frac{\partial^2}{\partial x^2}h$

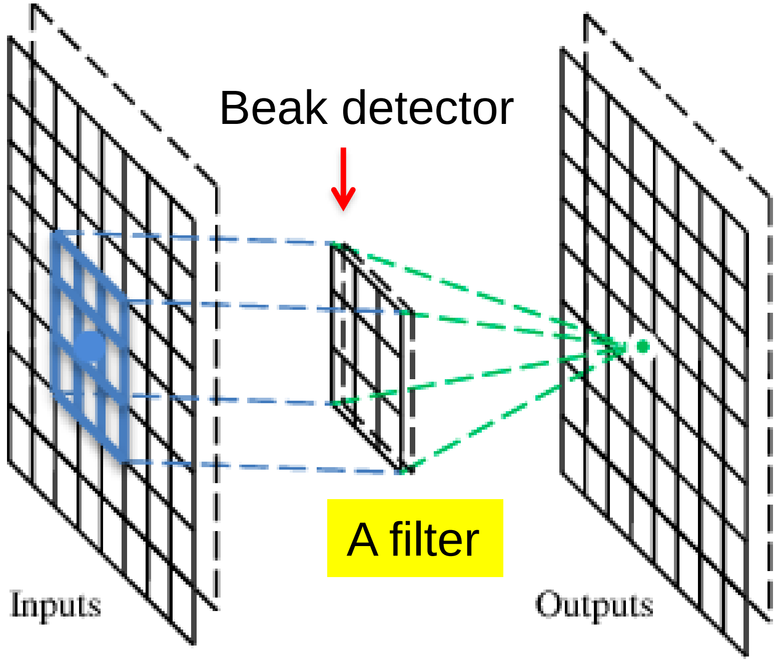
$(\frac{\partial^2}{\partial x^2}h) \star f$




Where is the edge? Zero-crossings of bottom graph

# A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



# Convolution

**These are the network parameters to be learned.**

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

# Convolution

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

stride=1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

Dot product

3 -1

6 x 6 image

# Convolution

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

If stride=2

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

3 -3

# Convolution

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

stride=1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

| | | | |
|---|---|---|---|
| 3 | -1 | -3 | -1 |
| -3 | 1 | 0 | -3 |
| -3 | -3 | 0 | 1 |
| 3 | -2 | -2 | -1 |

# Convolution

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

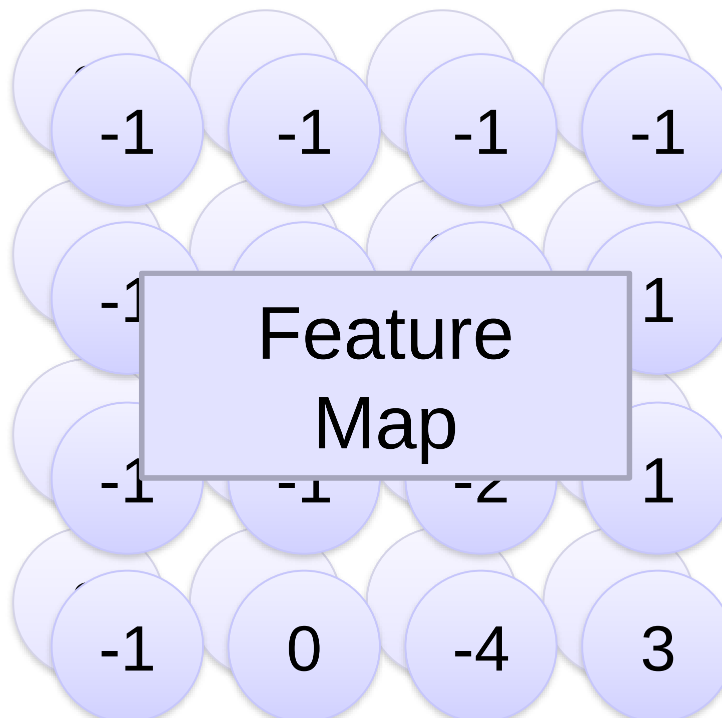
Filter 2

stride=1

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

Repeat this for each filter




Two 4 x 4 images
Forming 2 x 4 x 4 matrix

# Color image: RGB 3 channels

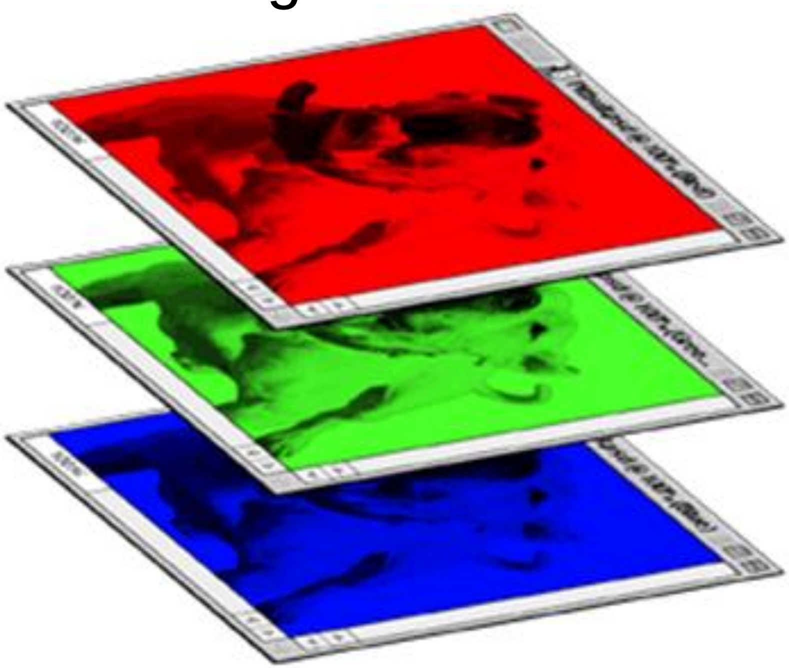
Filter 1

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 2

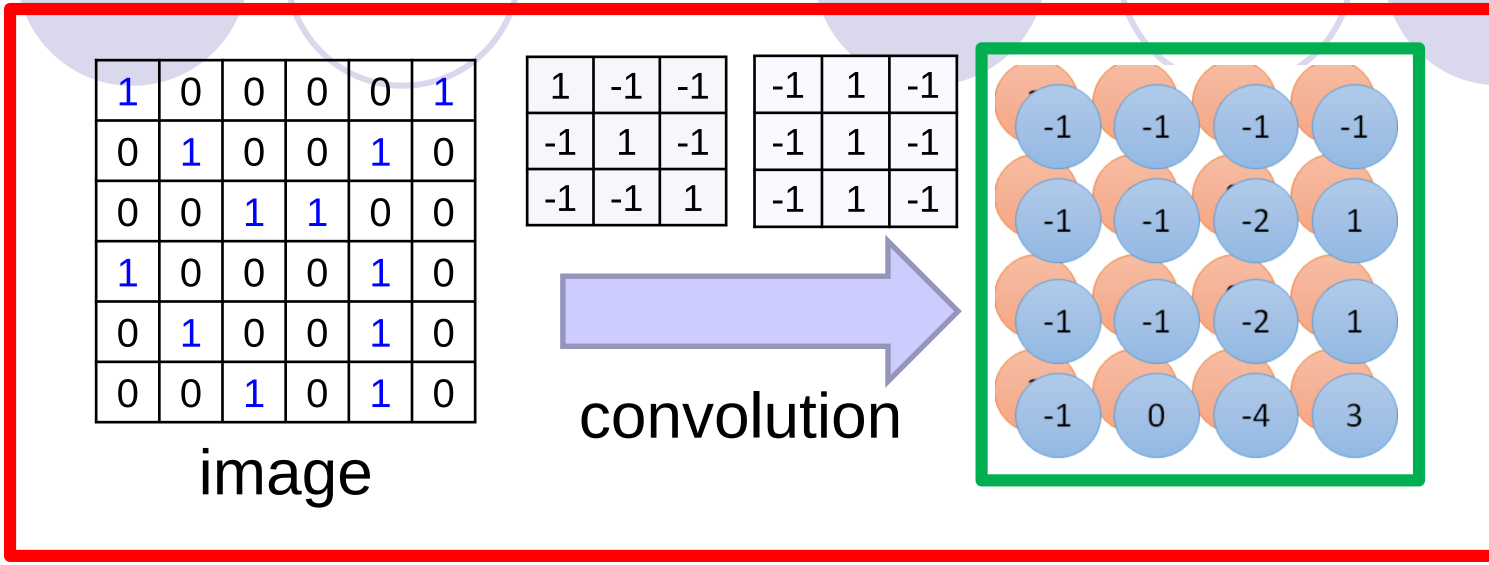
| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

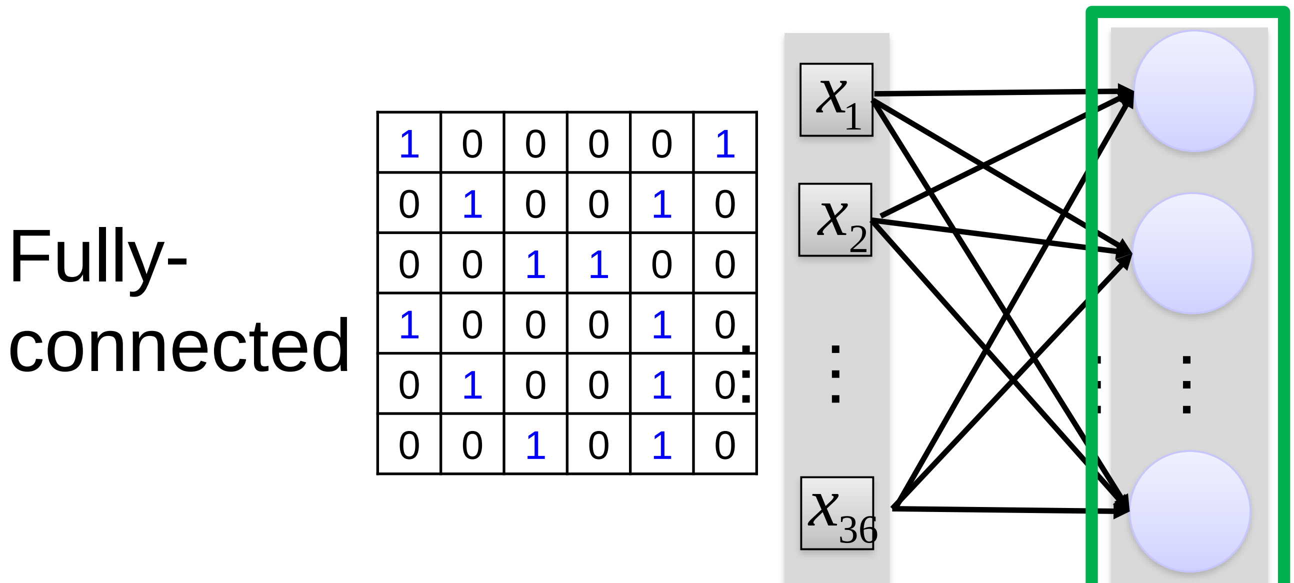
Color image

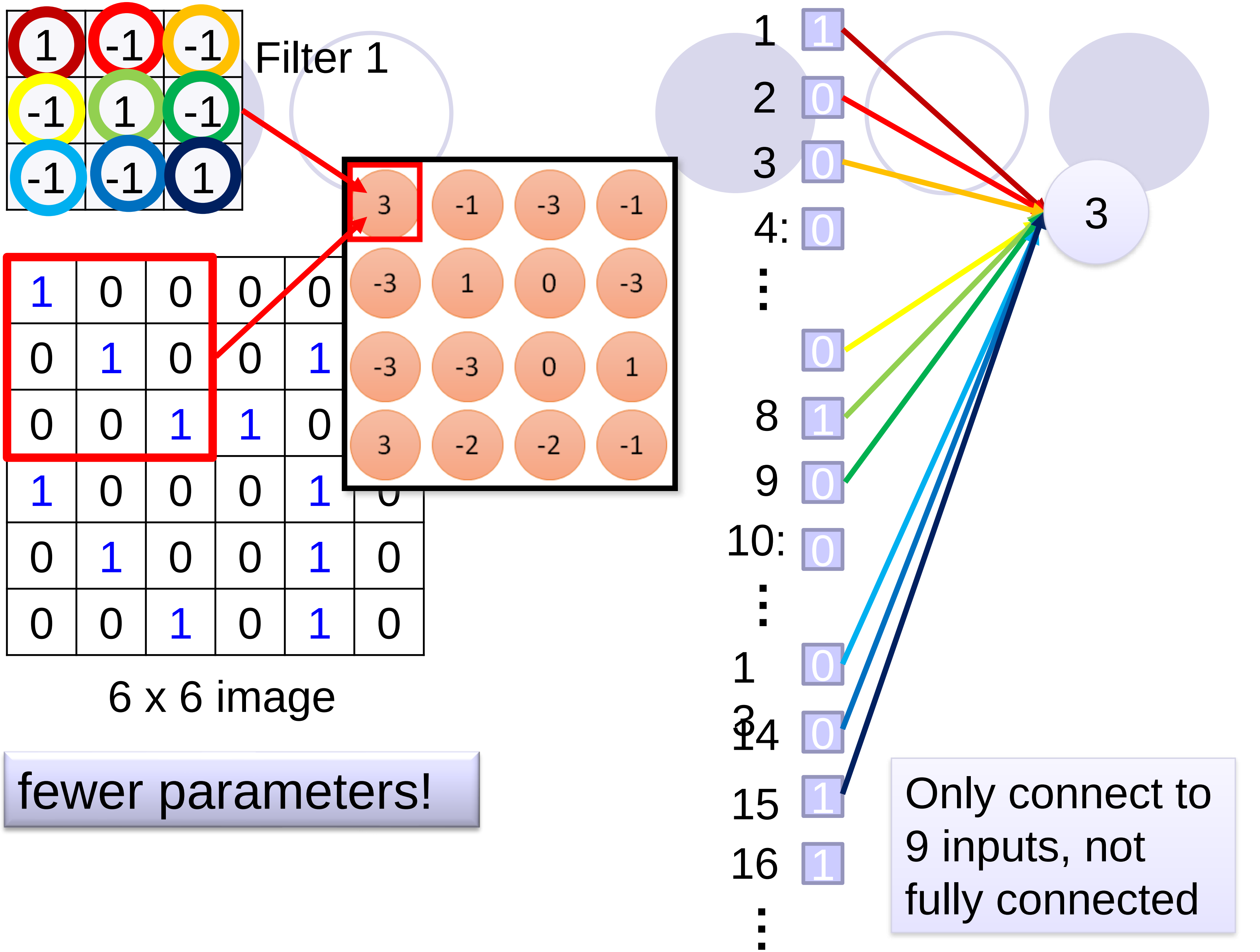


| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

# ***Convolution v.s. Fully Connected***

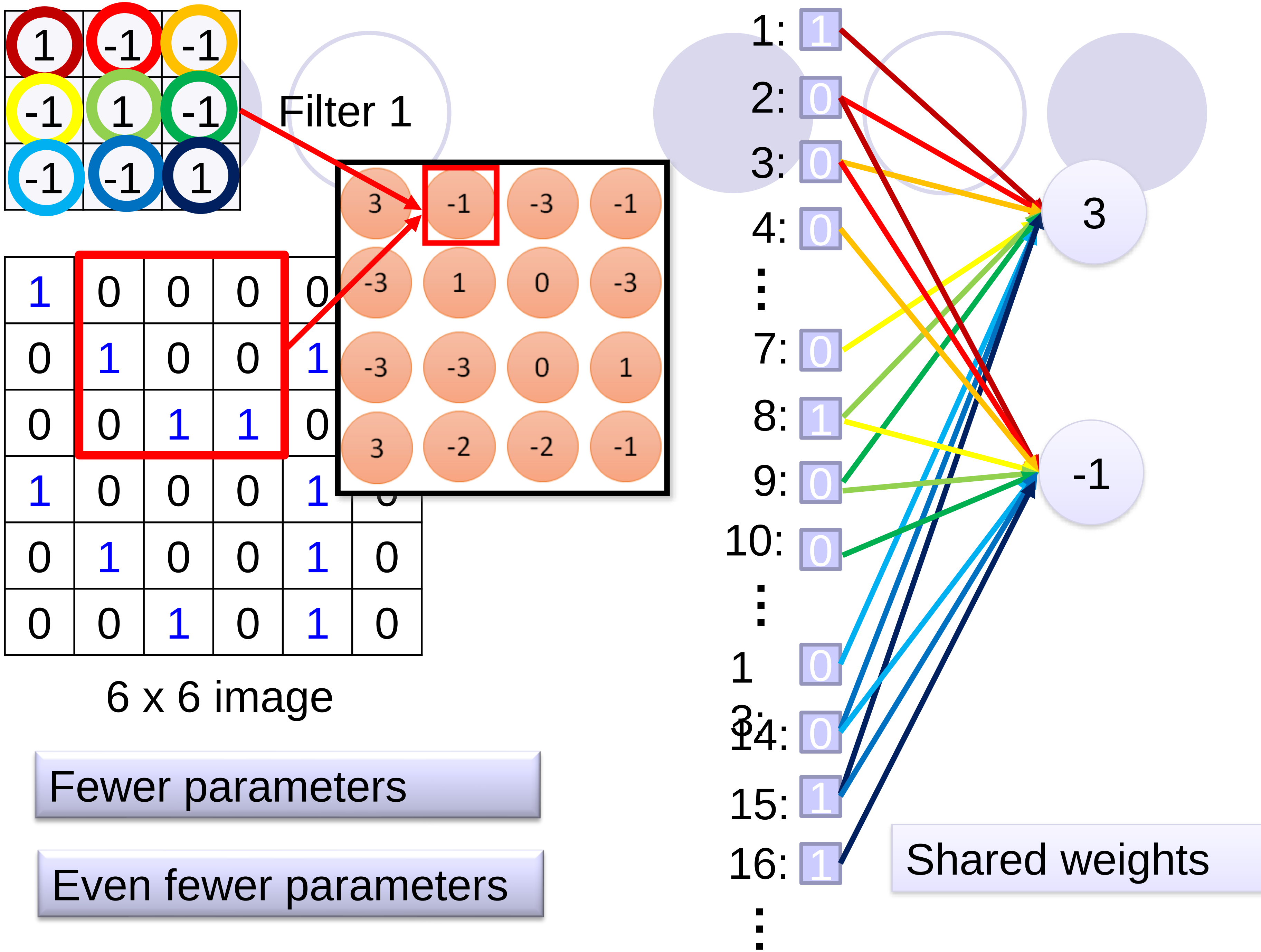




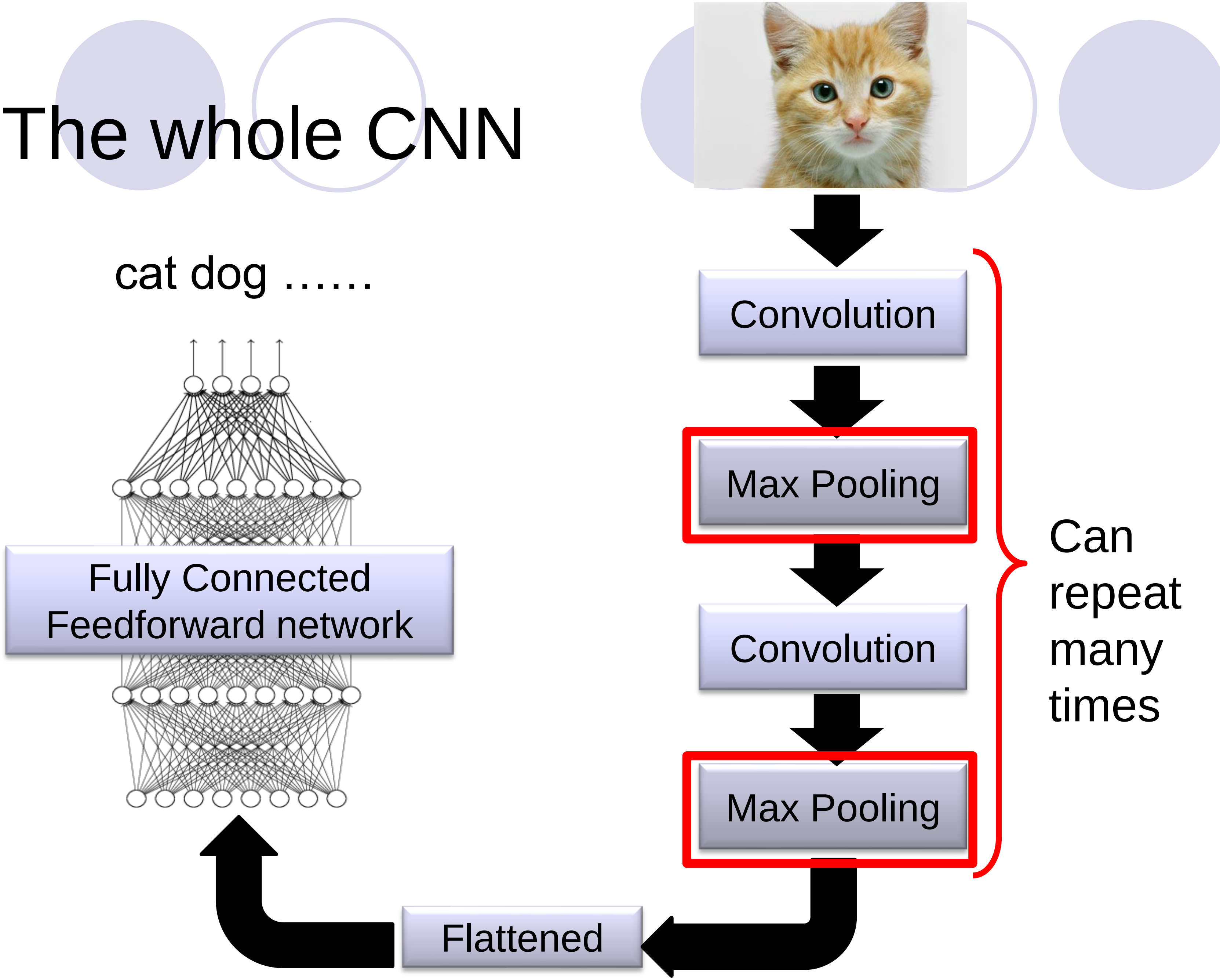

fewer parameters!

Only connect to 9 inputs, not fully connected



Filter 1
6 x 6 image
Fewer parameters
Even fewer parameters
Shared weights
3
-1

# The whole CNN



# Max Pooling

| | | |
|---|---|---|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Filter 1

| | | |
|---|---|---|
| -1 | 1 | -1 |
| -1 | 1 | -1 |
| -1 | 1 | -1 |

Filter 2

| | | | |
|---|---|---|---|
| 3 | -1 | -3 | -1 |
| -3 | 1 | 0 | -3 |
| -3 | -3 | 0 | 1 |
| 3 | -2 | -2 | -1 |

| | | | |
|---|---|---|---|
| -1 | -1 | -1 | -1 |
| -1 | -1 | -2 | 1 |
| -1 | -1 | -2 | 1 |
| -1 | 0 | -4 | 3 |

# Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

fewer parameters to characterize the image

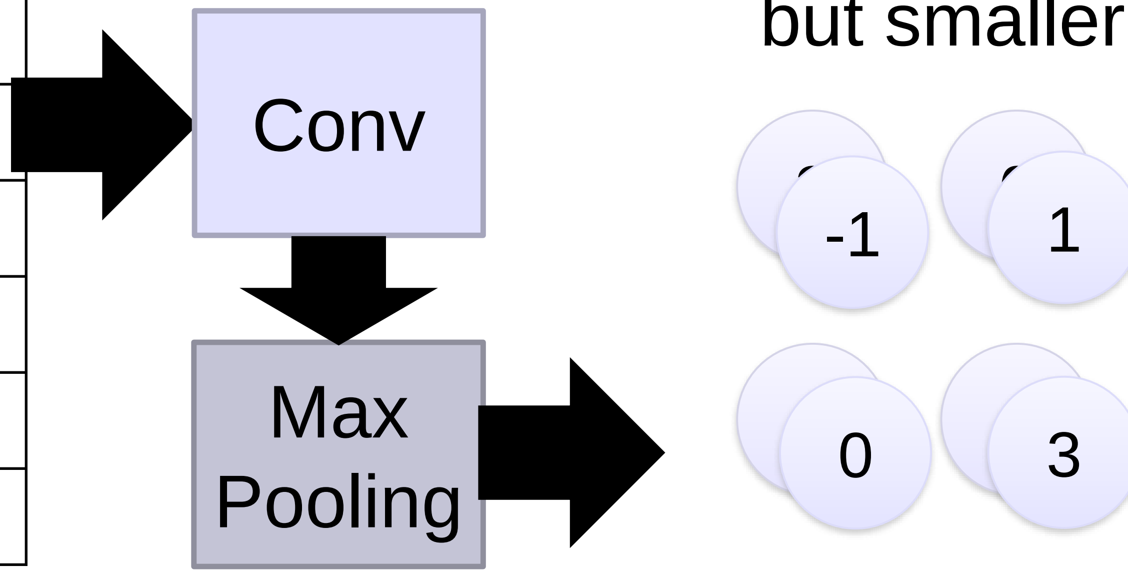
# A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

# Max Pooling

| | | | | | |
|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |

6 x 6 image

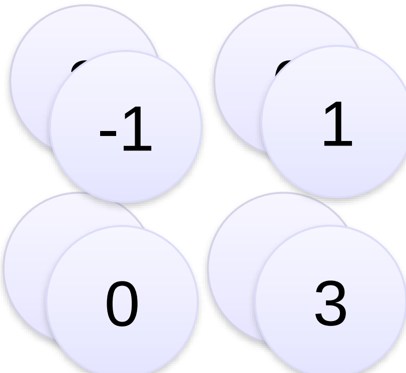



New image
but smaller

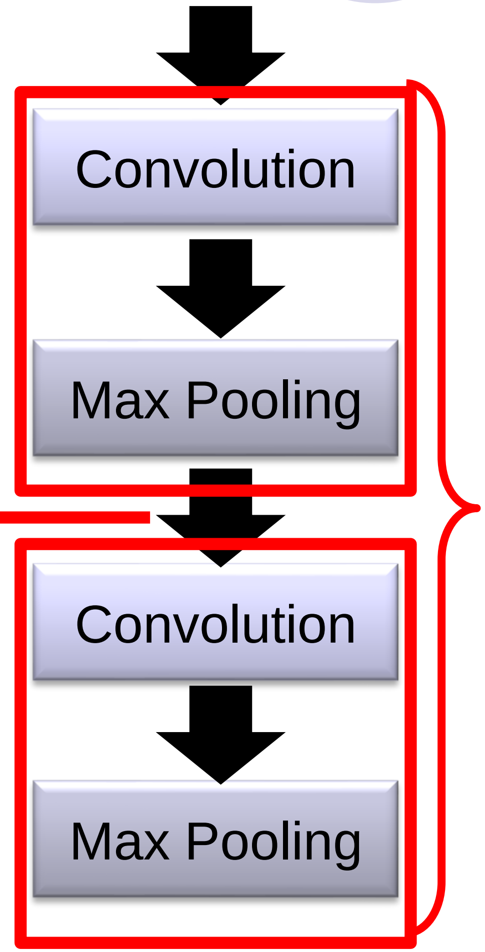
2 x 2 image

Each filter
is a channel

# The whole CNN



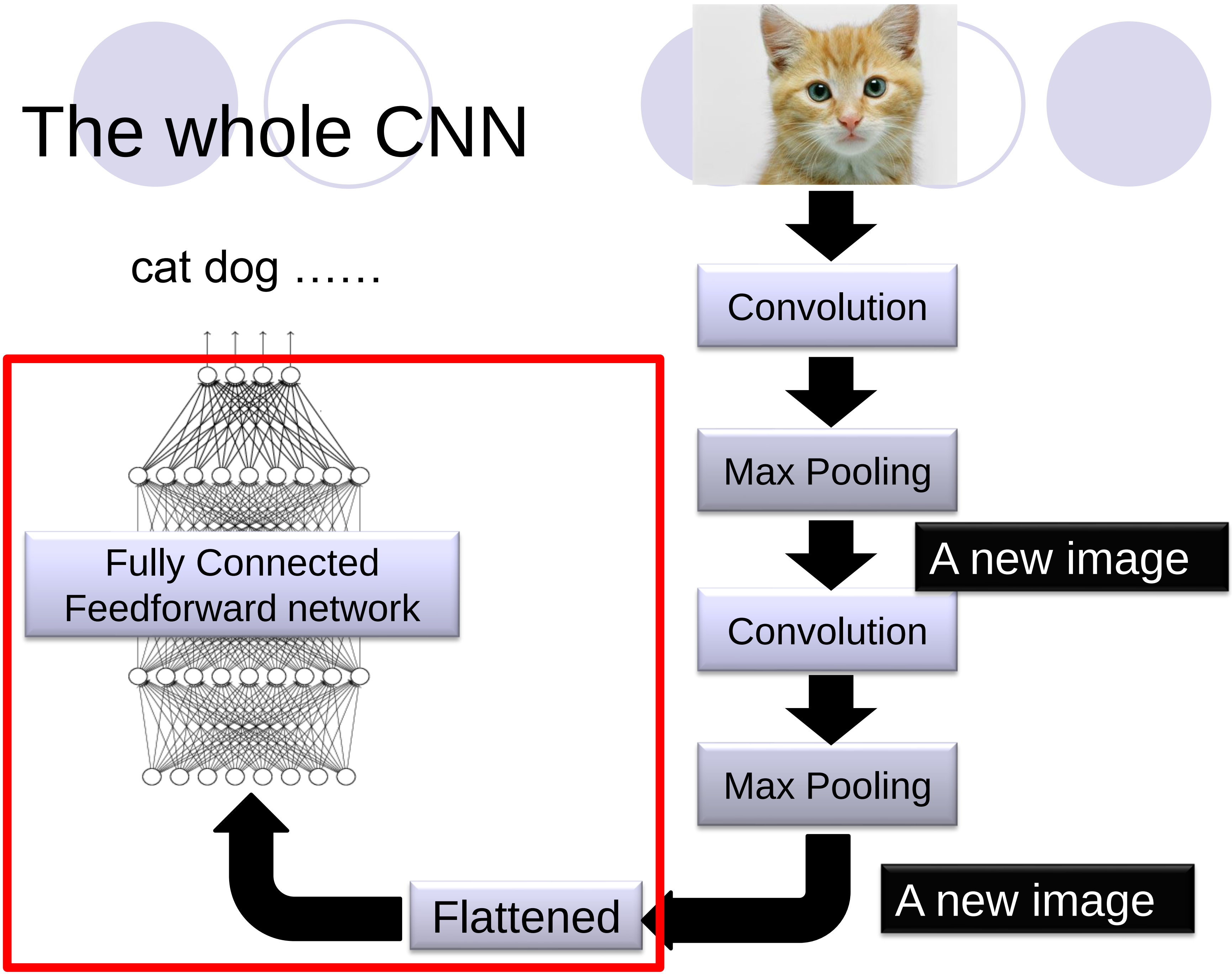

A new image



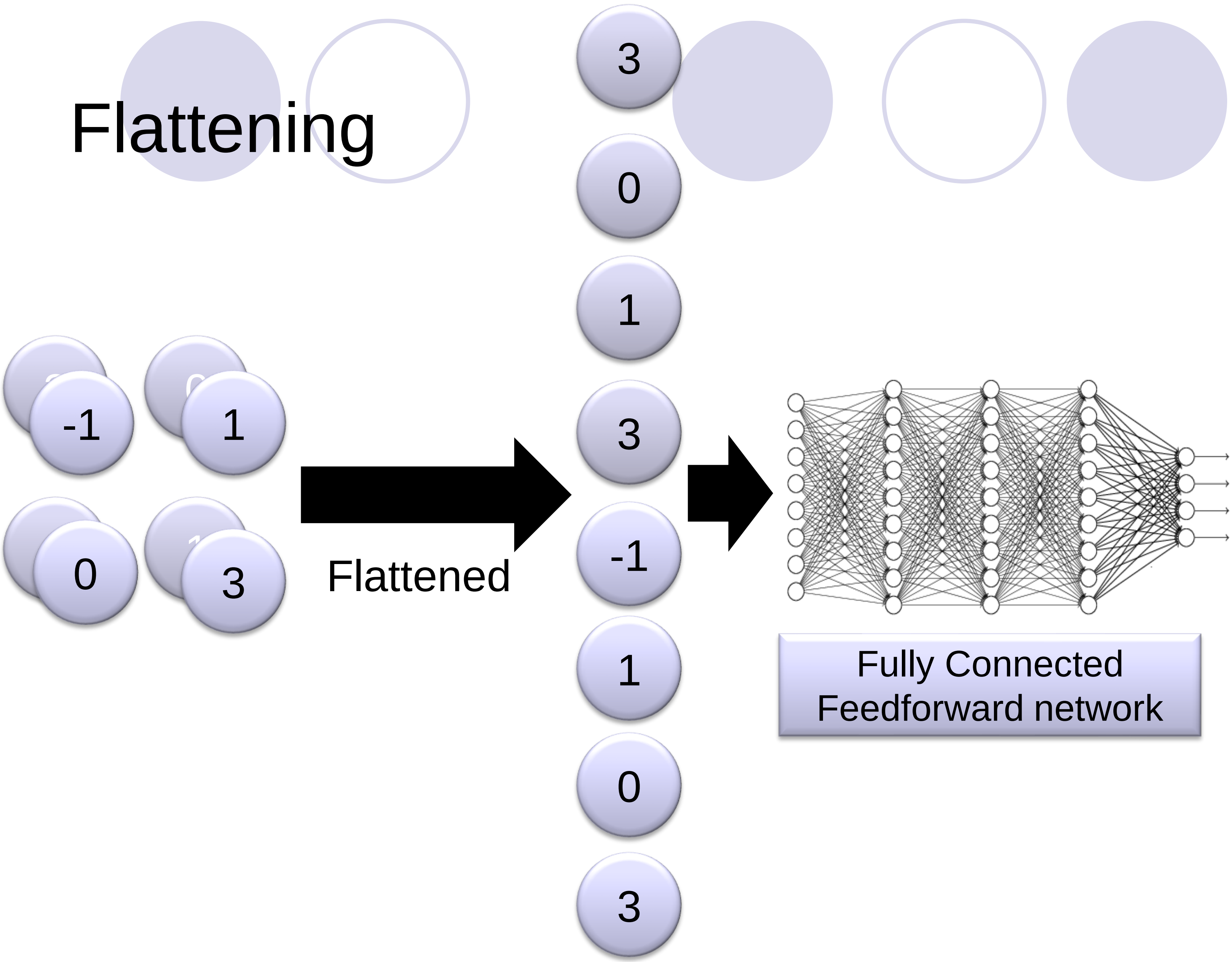

Smaller than the original image

The number of channels is the number of filters

# The whole CNN



# Flattening

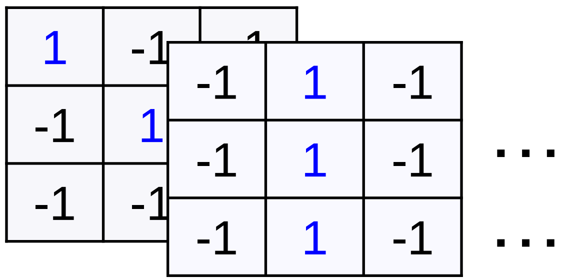


-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

# CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
                input_shape=(28,28,1)) )
```



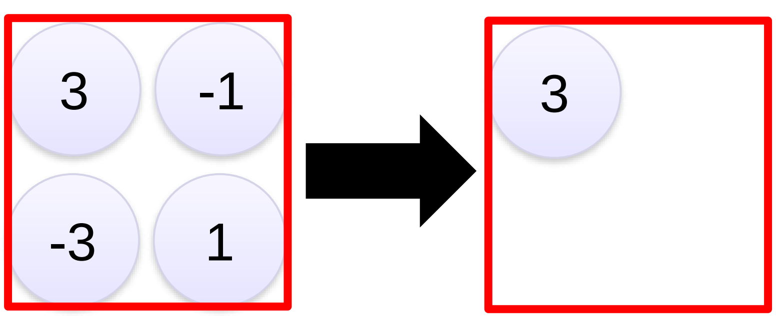

There are 25 3x3 filters.

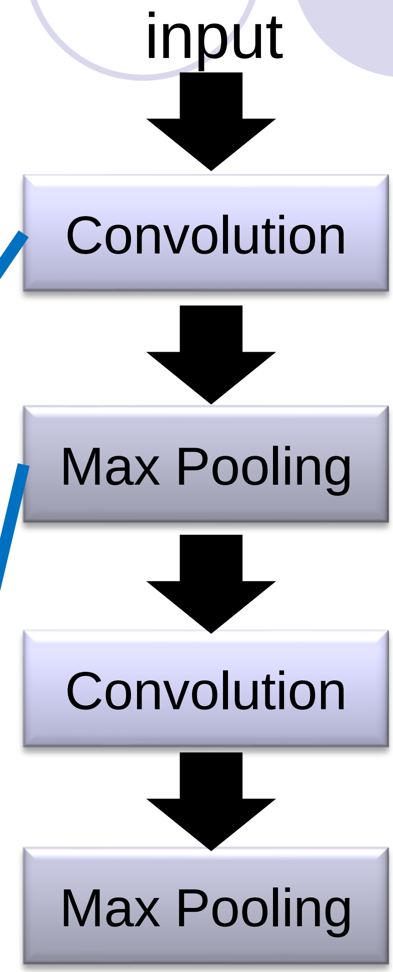
Input_shape = ( 28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

```
model2.add(MaxPooling2D((2,2)))
```



# CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***

Input

1 x 28 x 28

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(28,28,1)) )
```

Convolution

How many parameters for each filter?

9

25 x 26 x 26

```
model2.add(MaxPooling2D((2,2)))
```

Max Pooling

25 x 13 x 13

```
model2.add(Convolution2D(50,3,3))
```

Convolution

How many parameters for each filter?

225=
25x9

50 x 11 x 11

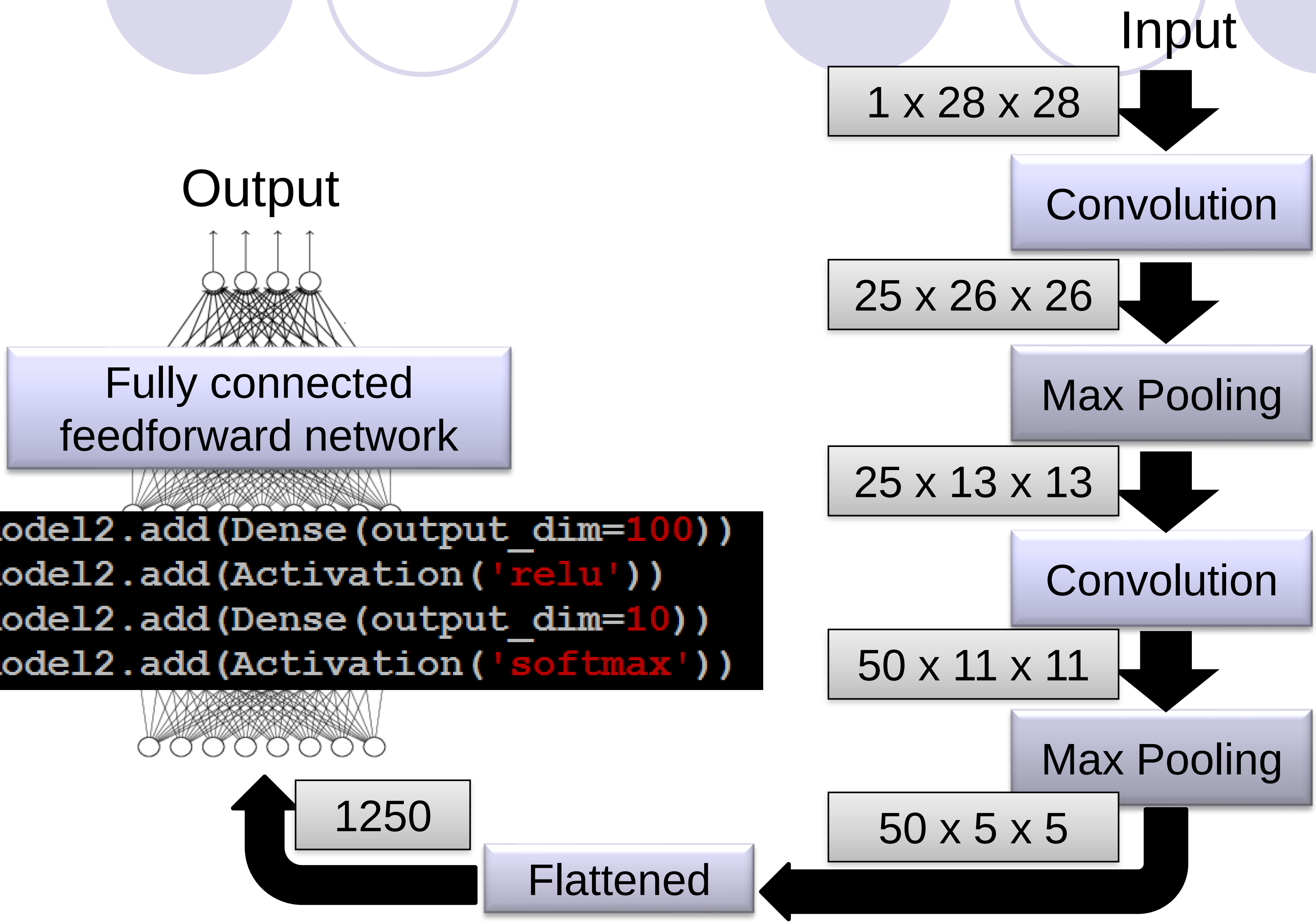
```
model2.add(MaxPooling2D((2,2)))
```

Max Pooling

50 x 5 x 5

# *CNN in Keras*

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***




```
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
```

```
model2.add(Flatten())
```

# AlphaGo

19 x 19 matrix

Black: 1

white: -1

none: 0

Neural Network

Next move

(19 x 19 positions)

Fully-connected feedforward network can be used

But CNN performs much better

# AlphaGo's policy network

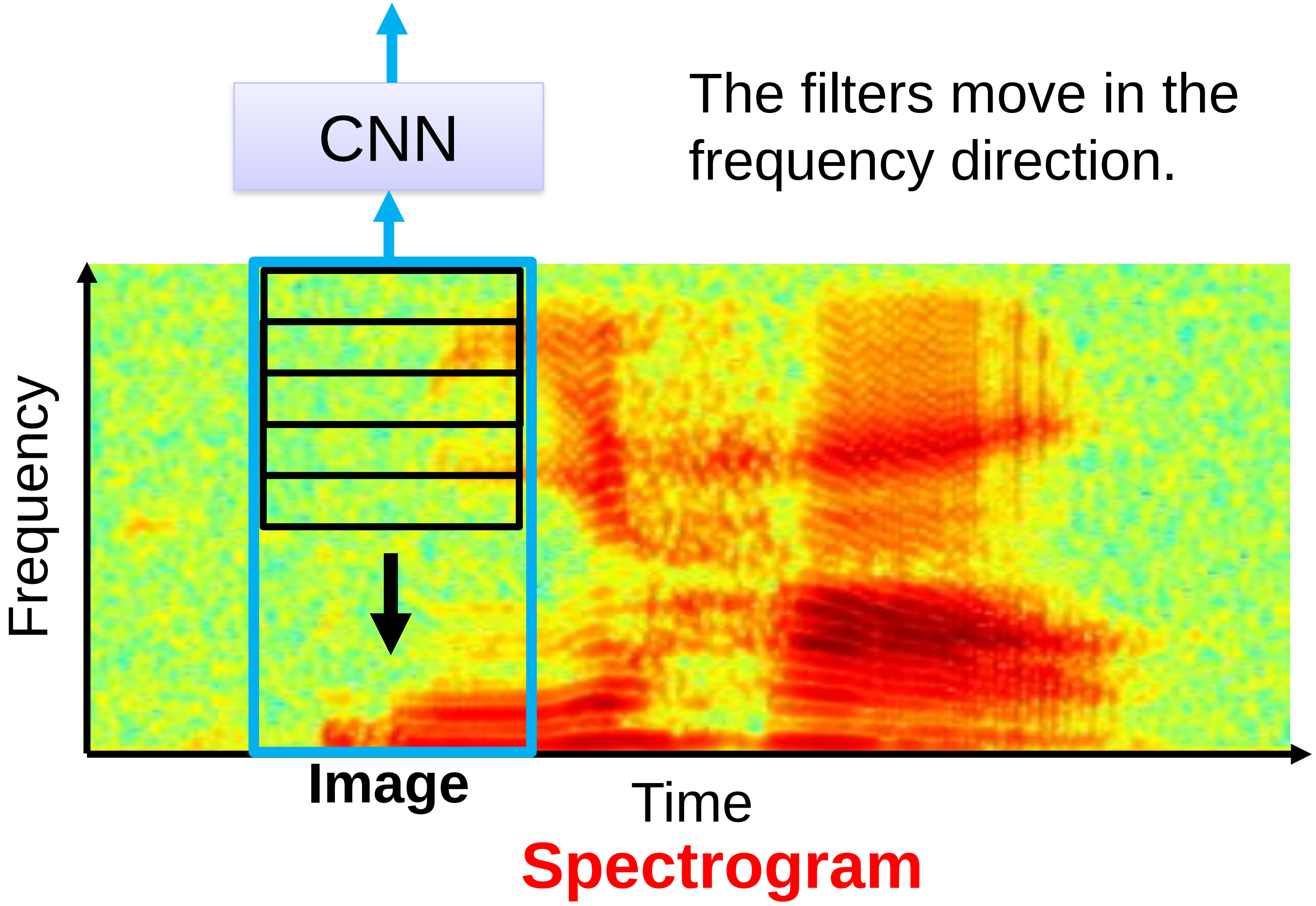
The following is quotation from their Nature article:

Note: AlphaGo does not use Max Pooling.

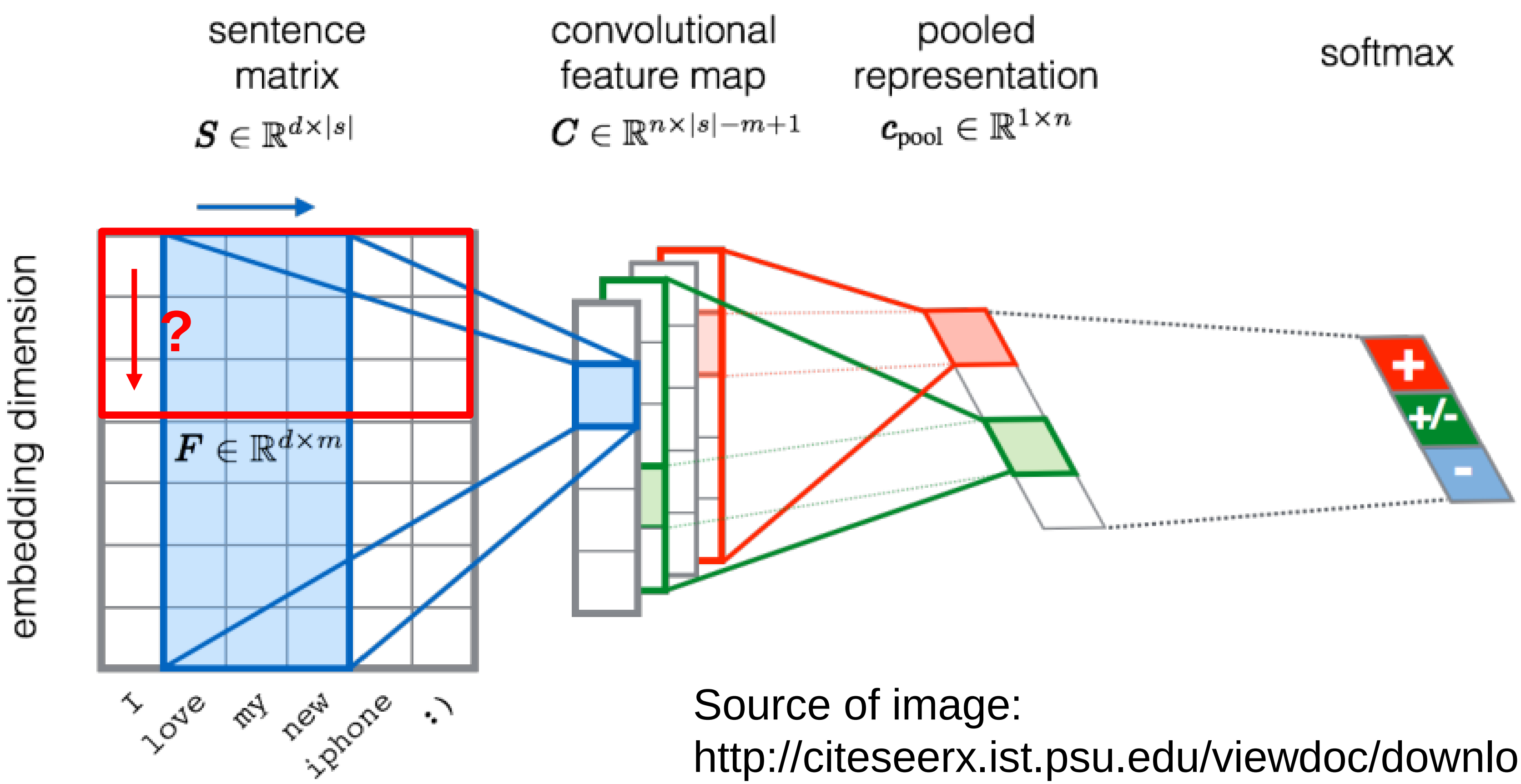
**Neural network architecture.** The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a $23 \times 23$ image, then convolves $k$ filters of kernel size $5 \times 5$ with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a $21 \times 21$ image, then convolves $k$ filters of kernel size $3 \times 3$ with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size $1 \times 1$ with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

# CNN in speech recognition

The filters move in the frequency direction.



# CNN in text classification




Source of image:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

# Convolutional NN

Convolutional Neural Networks is extension of traditional Multi-layer Perceptron, based on 3 ideas:

1. Local receive fields
2. Shared weights
3. Spatial / temporal sub-sampling

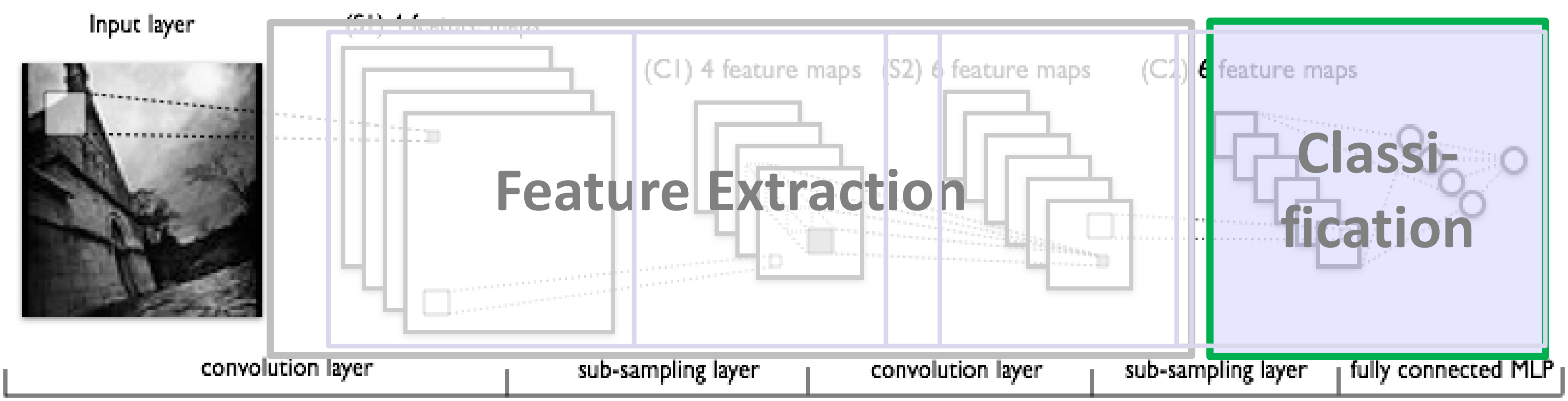
See LeCun paper (1998) on text recognition:

http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

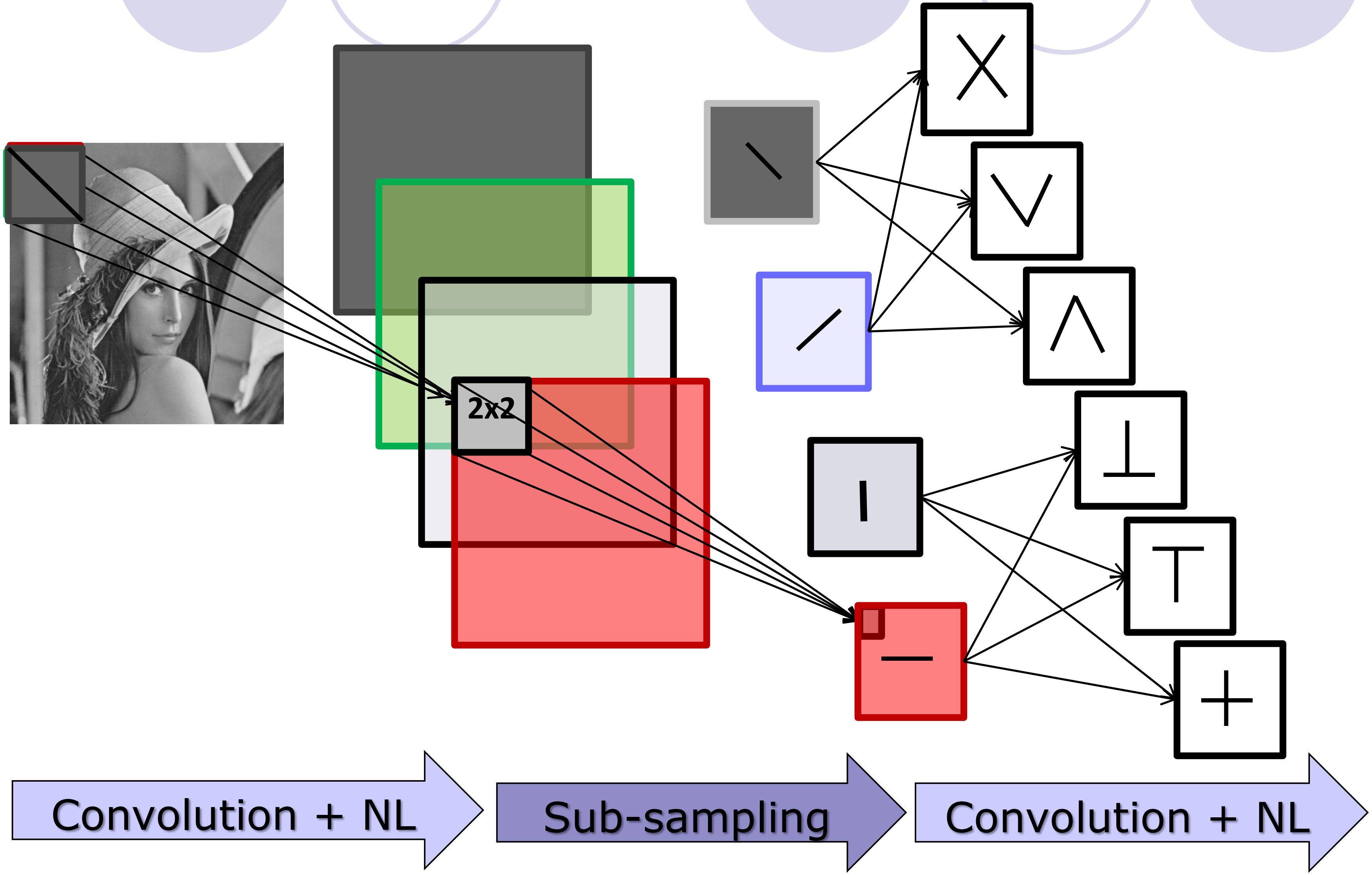
# What is Convolutional NN ?

## CNN - multi-layer NN architecture

  - Convolutional + Non-Linear Layer
  - Sub-sampling Layer
  - Convolutional +Non-L inear Layer
  - Fully connected layers
- Supervised

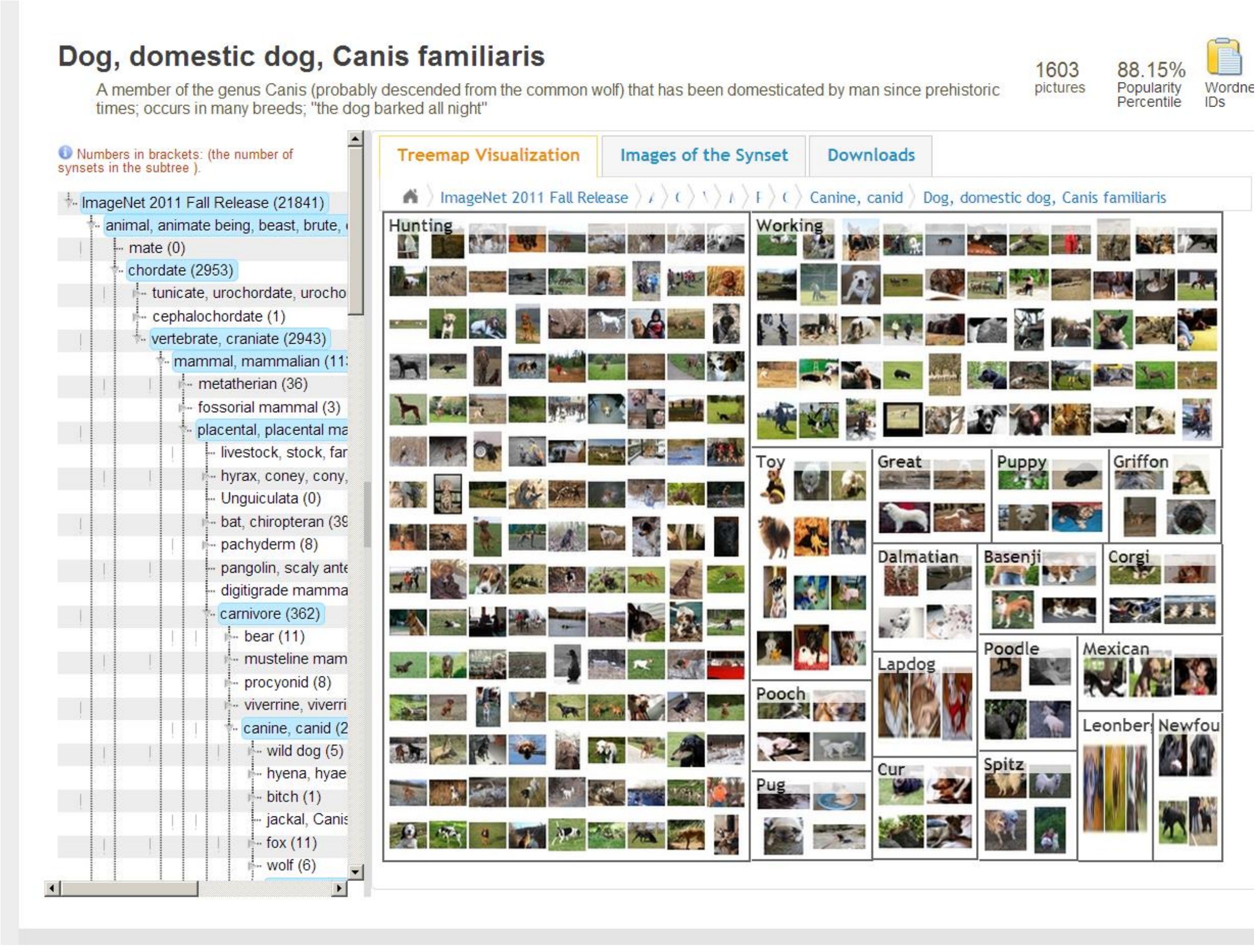


# What is Convolutional NN ?

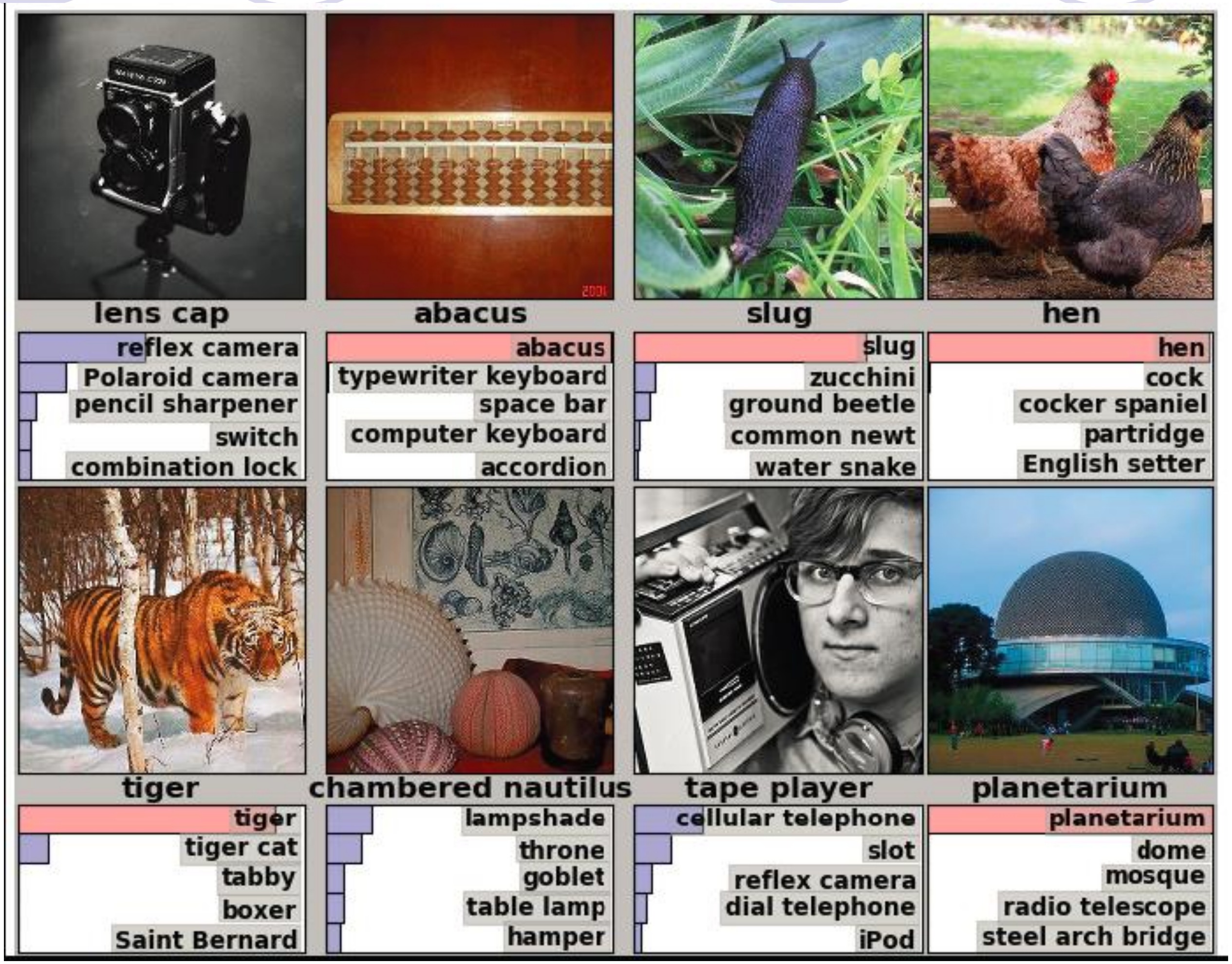


# CNN success story: ILSVRC 2012

Imagenet data base:  14 mln labeled images, 20K categories

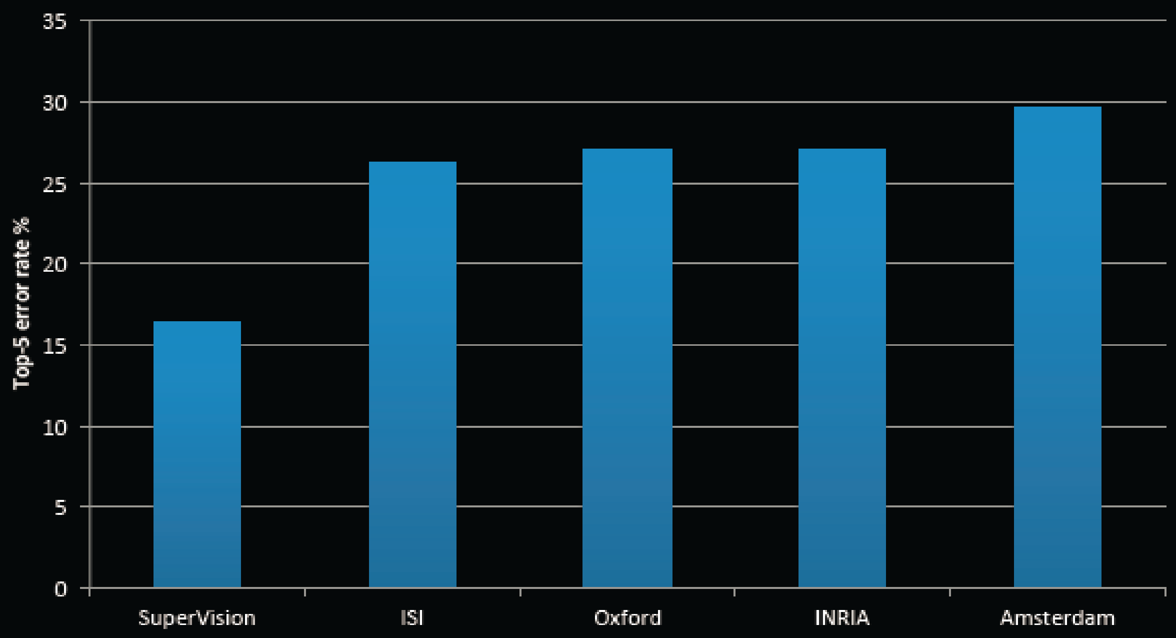


# ILSVRC: Classification



# Imagenet Classifications 2012

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



# ILSVRC 2012: top rankers

http://www.image-net.org/challenges/LSVRC/2012/results.html

| N | Error-5 | Algorithm | Team | Authors |
|---|---|---|---|---|
| 1 | 0.153 | Deep Conv. Neural Network | Univ. of Toronto | Krizhevsky et al |
| 2 | 0.262 | Features + Fisher Vectors + Linear classifier | ISI | Gunji et al |
| 3 | 0.270 | Features + FV + SVM | OXFORD_VGG | Simonyan et al |
| 4 | 0.271 | SIFT + FV + PQ + SVM | XRCE/INRIA | Perronin et al |
| 5 | 0.300 | Color desc. + SVM | Univ. of Amsterdam | van de Sande et al |

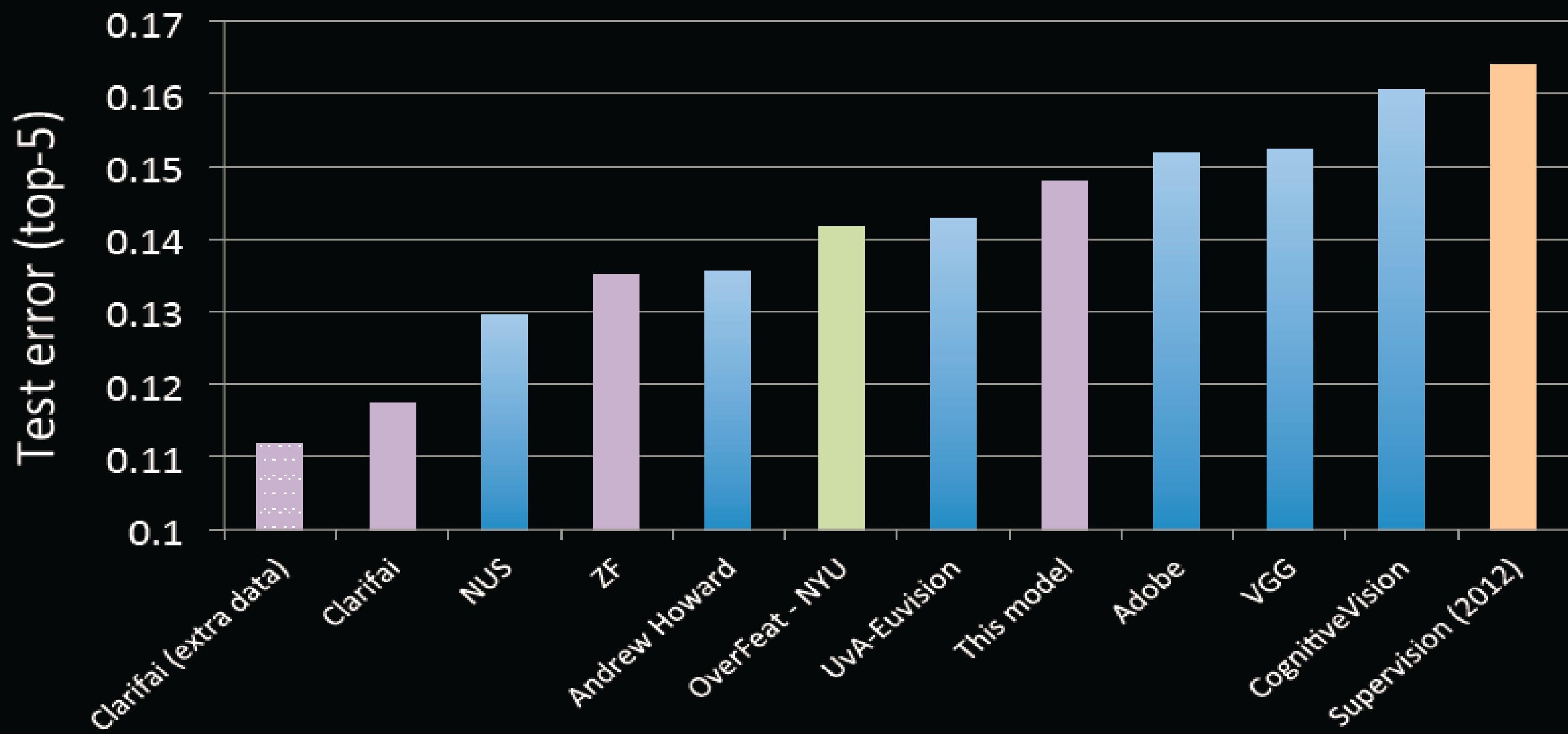
# Imagenet 2013: top rankers

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

| N | Error-5 | Algorithm | Team | Authors |
|---|---|---|---|---|
| 1 | 0.117 | Deep Convolutional Neural Network | Clarifi | Zeiler |
| 2 | 0.129 | Deep Convolutional Neural Networks | Nat.Univ Singapore | Min LIN |
| 3 | 0.135 | Deep Convolutional Neural Networks | NYU | Zeiler Fergus |
| 4 | 0.135 | Deep Convolutional Neural Networks | | Andrew Howard |
| 5 | 0.137 | Deep Convolutional Neural Networks | Overfeat NYU | Pierre Sermanet et al |

# Imagenet Classifications 2013

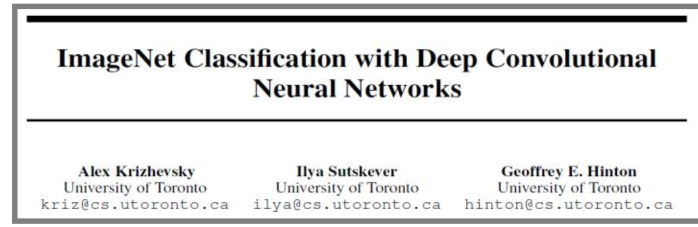
- http://www.image-net.org/challenges/LSVRC/2013/results.php




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

# Conv Net Topology

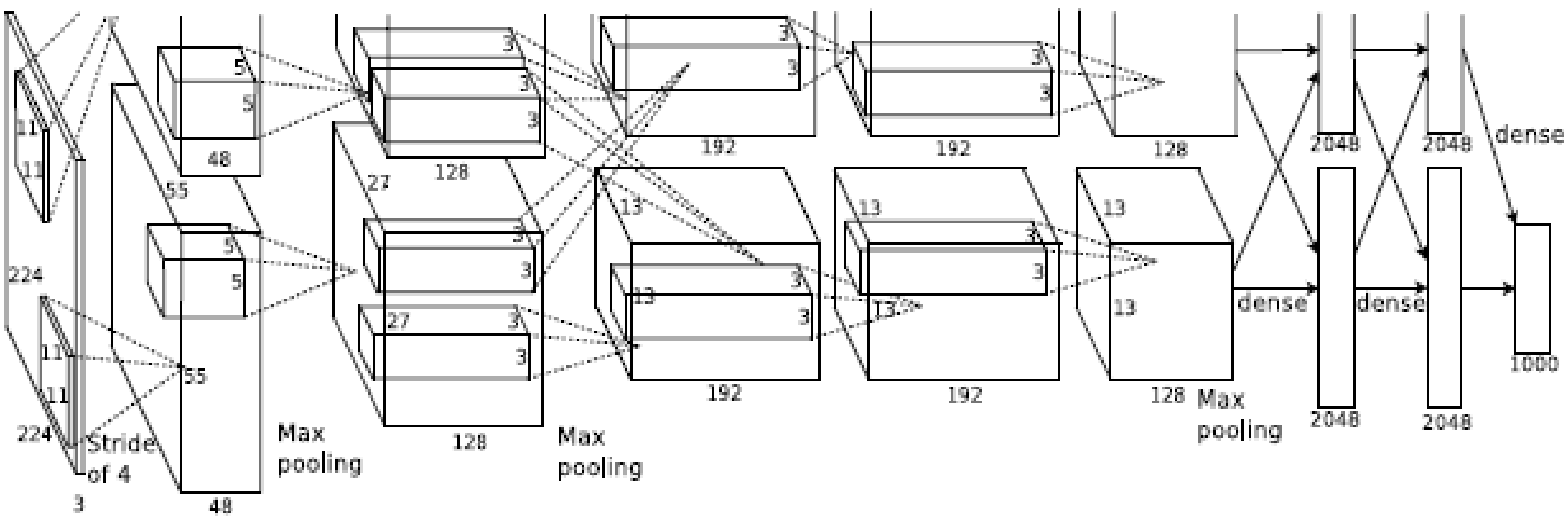
- 5 convolutional layers
- 3 fully connected layers + soft-max
- 650K neurons , 60 Mln weights


**ImageNet Classification with Deep Convolutional Neural Networks**


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

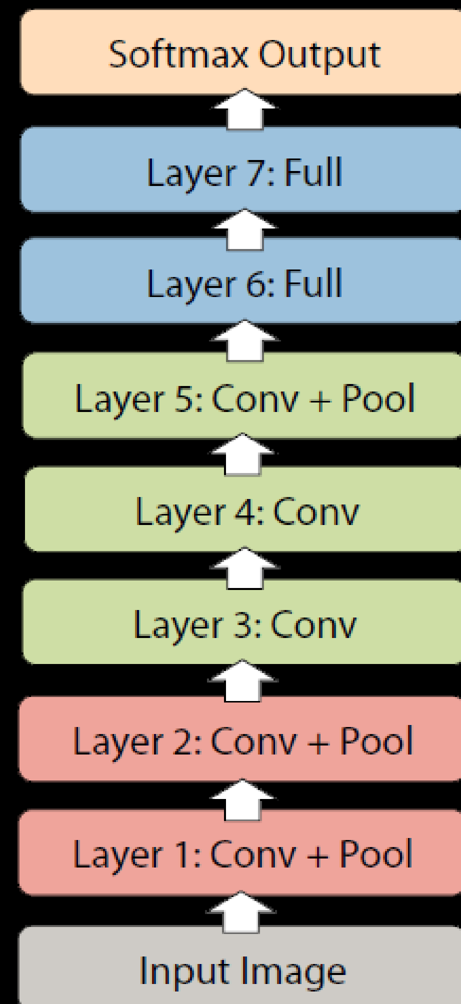
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

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

# Why ConvNet should be Deep?

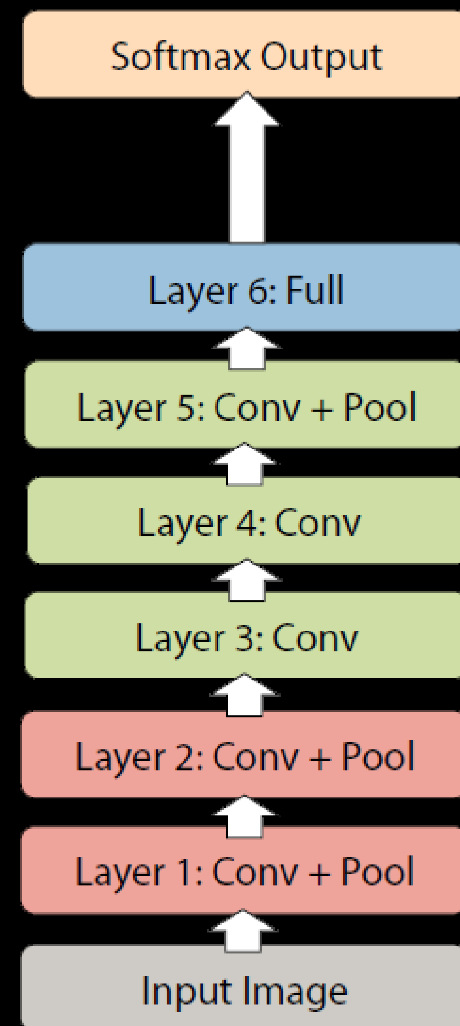
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:
  18.1% top-5 error




Rob Fergus, NIPS 2013

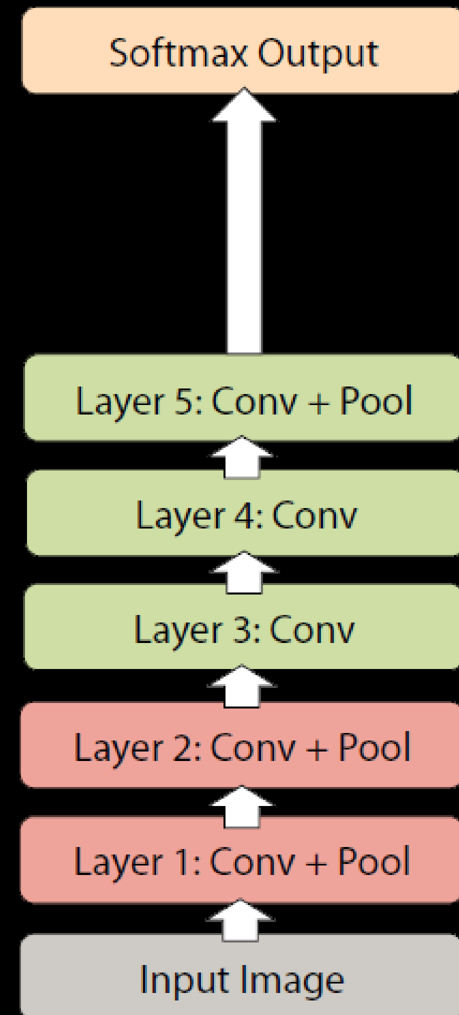
# Why ConvNet should be Deep?

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



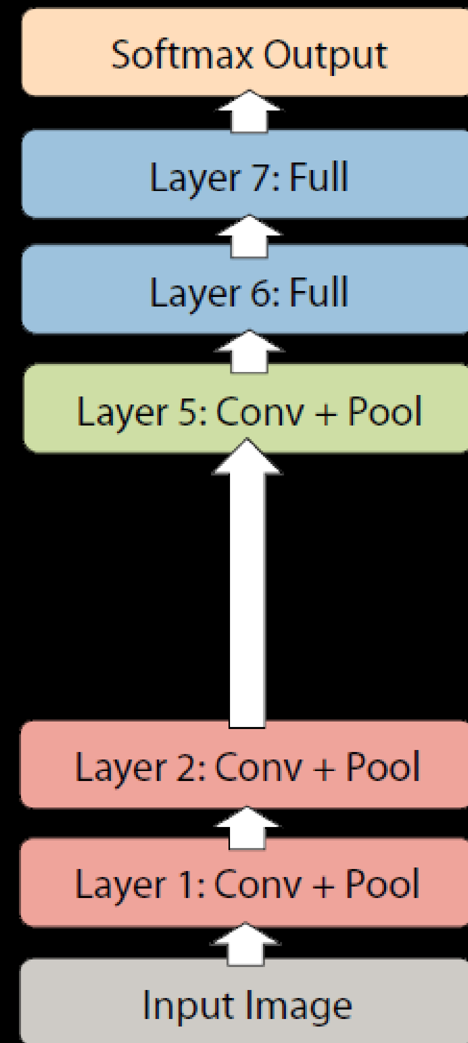
# Why ConvNet should be Deep?

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



# Why ConvNet should be Deep?

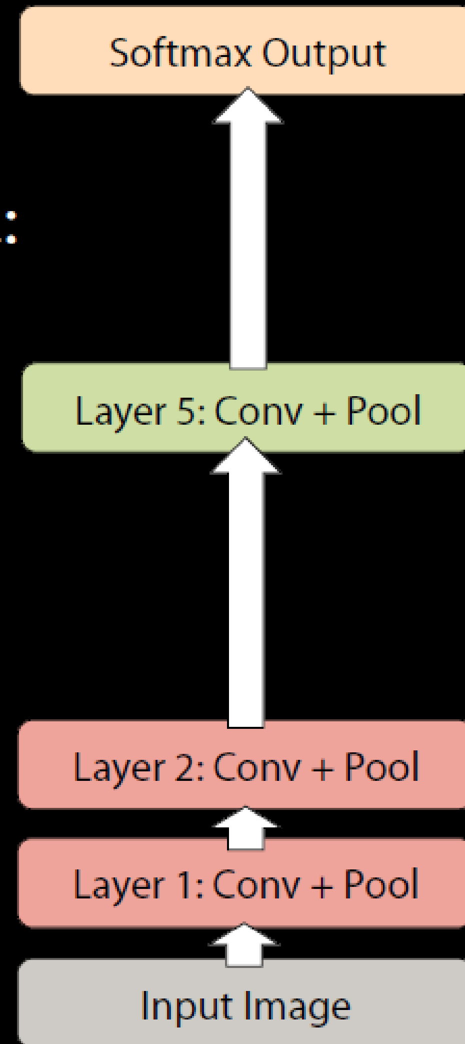
- Now try removing upper feature extractor layers:
  - Layers 3 & 4
- Drop ~1 million parameters
- 3.0% drop in performance



# Why ConvNet should be Deep?

- 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



# Conv Nets:

# beyond Visual Classification

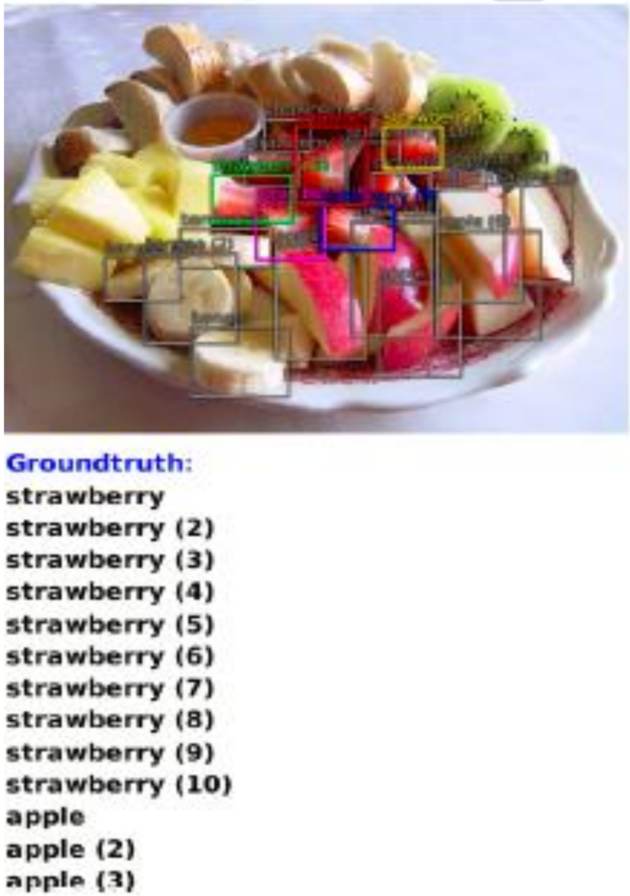
# CNN applications

CNN is a big hammer

Plenty low hanging fruits

You need just a right nail!

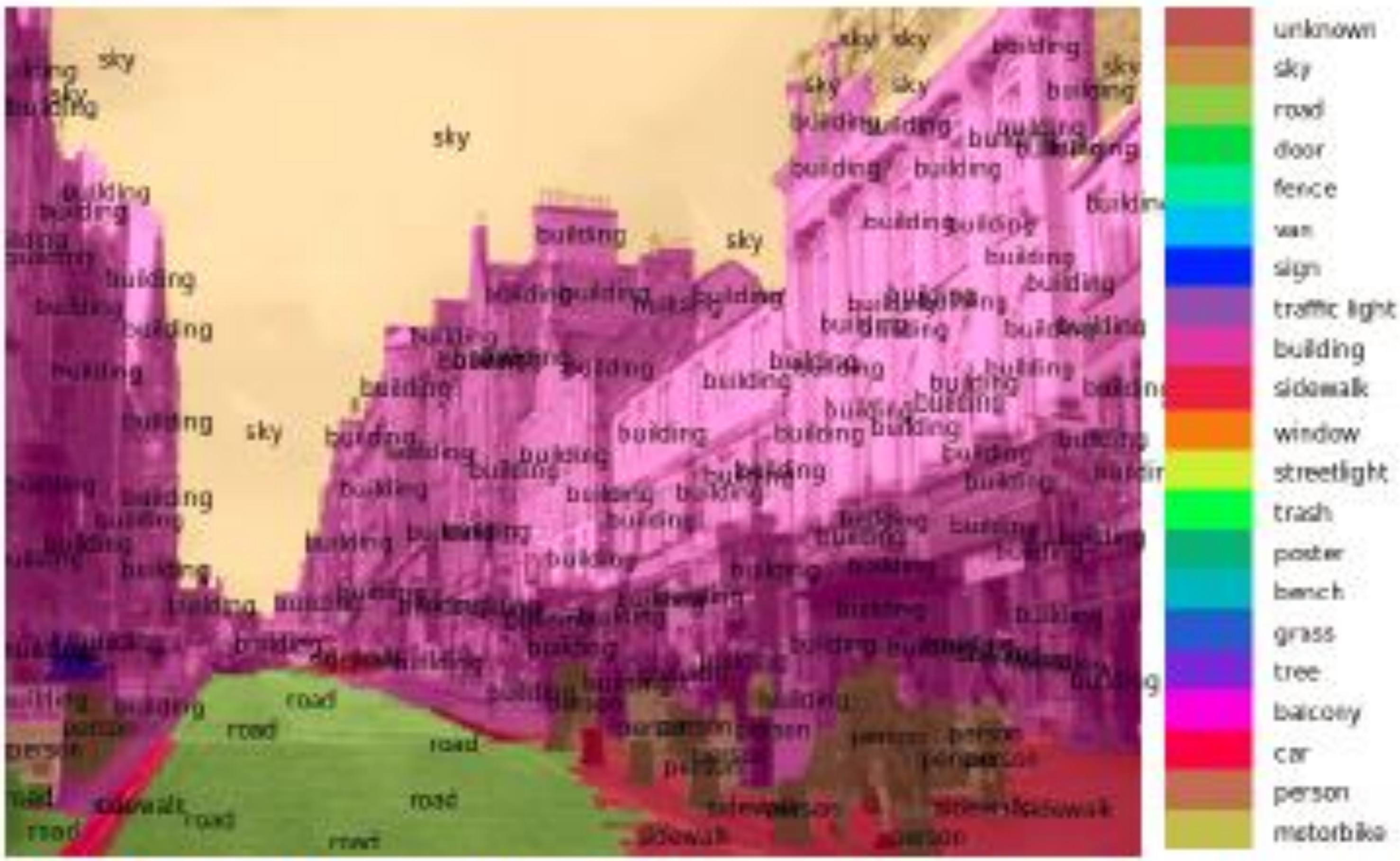
# Conv NN: Detection





Sermanet, CVPR 2014

# Conv NN: Scene parsing




Farabet, PAMI 2013

# CNN: indoor semantic labeling RGBD

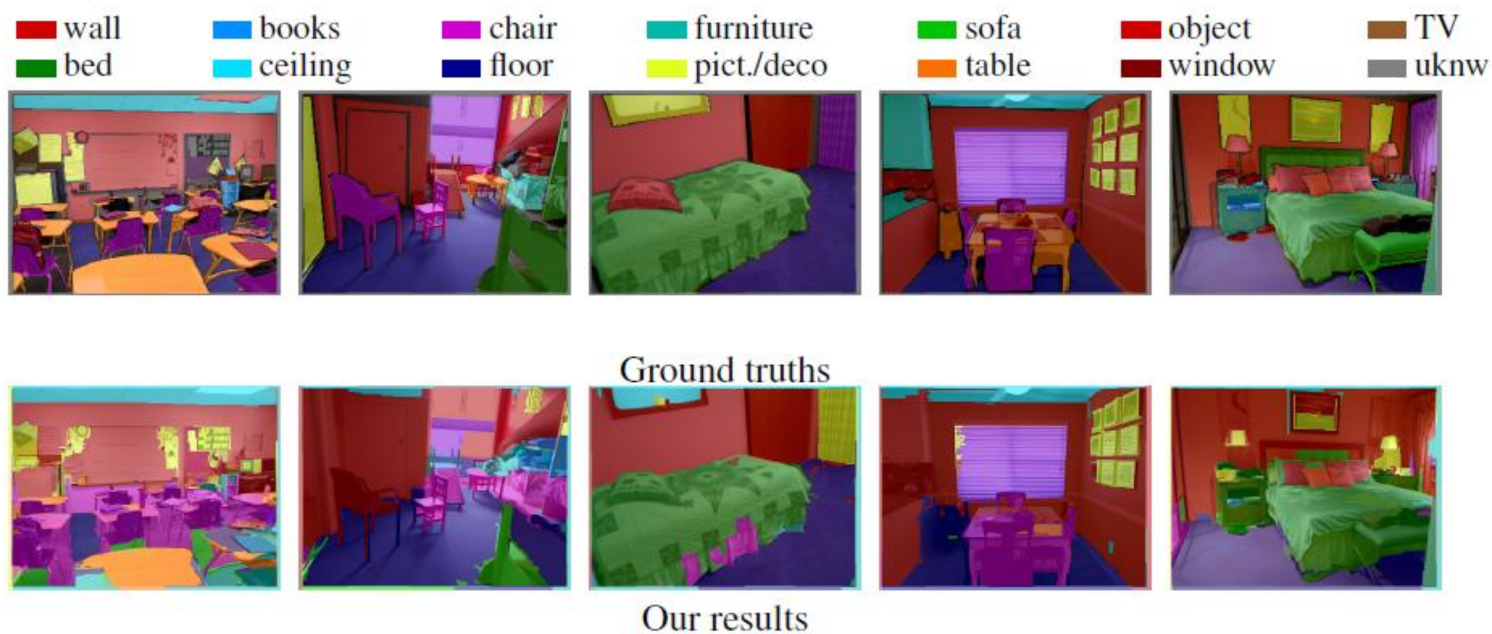



Figure 2: Some scene labelings using our Multiscale Convolutional Network trained on RGBD images.

Farabet, 2013

# Conv NN: Action Detection

Taylor, ECCV 2010