Today: Outline

- **Neural networks cont'd**

- **Types of networks:** Feed-forward networks, convolutional networks, recurrent networks

- **ConvNets**: multiplication vs convolution; filters (or kernels); convolutional layers; 1D and 2D convolution; pooling layers; LeNet, CIFAR10Net

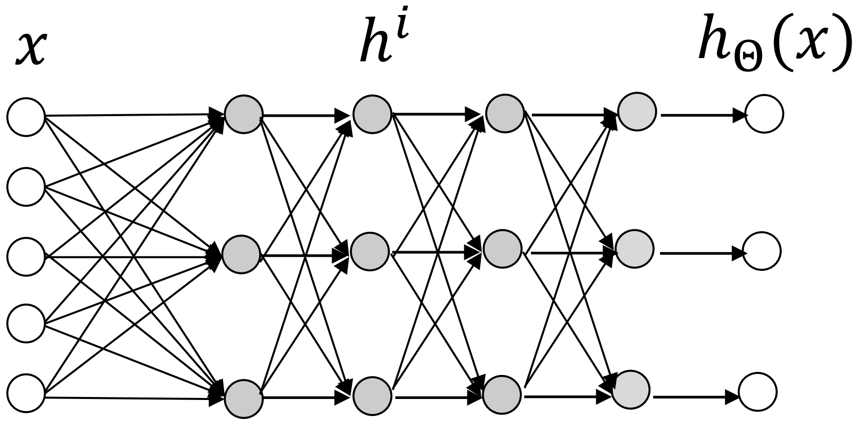
- **Reminder:** Pre-lecture material due Fri 28
  PS2 Self Score due Mar 3

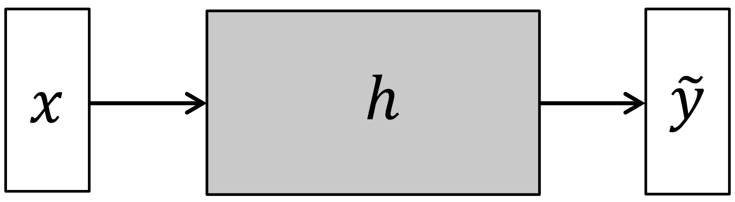
- **Announcement:** BU Productions visiting next class

Neural Networks III

Network Architectures

Neural networks: recap





Learn parameters via gradient descent

$$\min_{\Theta} J(\Theta)$$

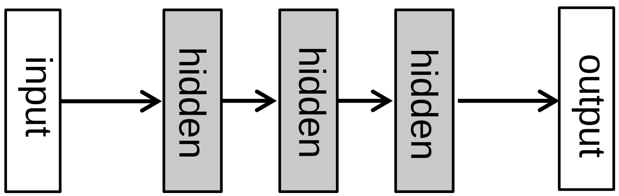
Backpropagation efficiently computes cost (forward pass) and gradient (backward pass)

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$$

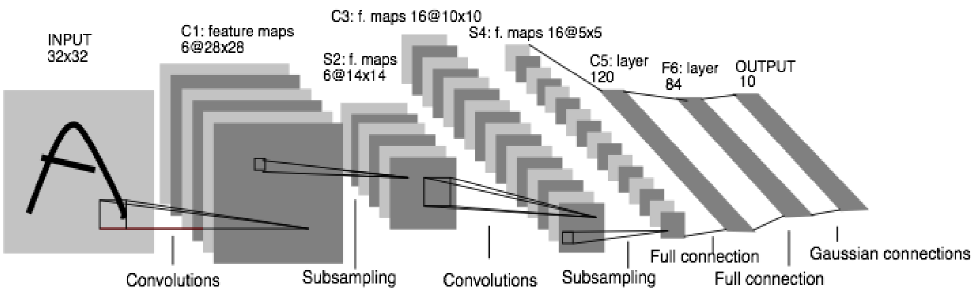
Network architectures

Feed-forward

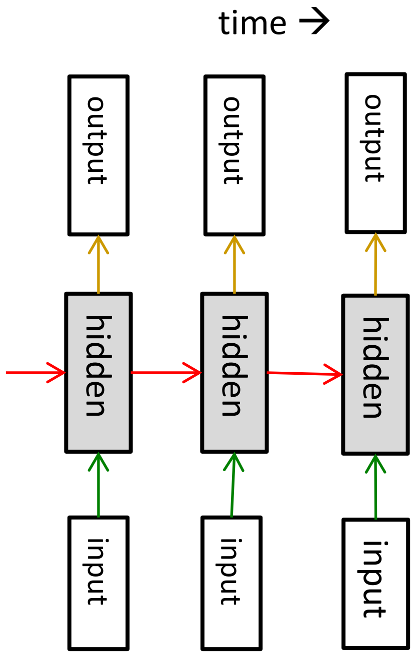
Fully connected



Convolutional



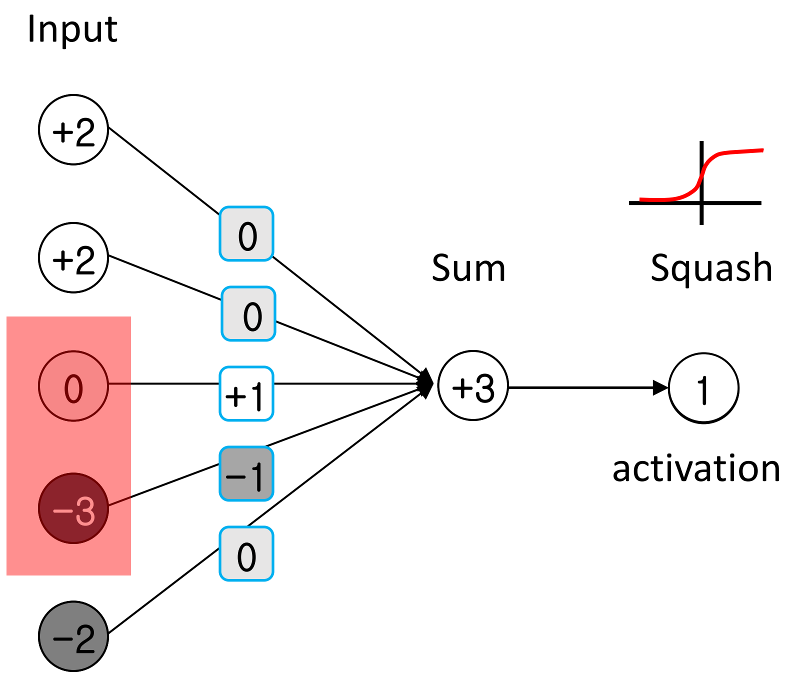
Recurrent



Neural Networks III

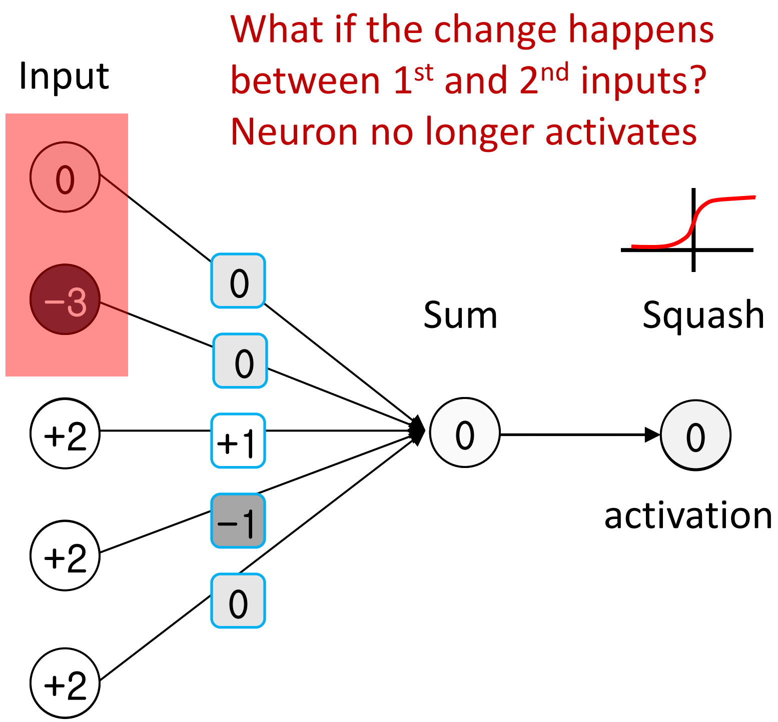
Convolutional Architectures

Multiplication vs convolution



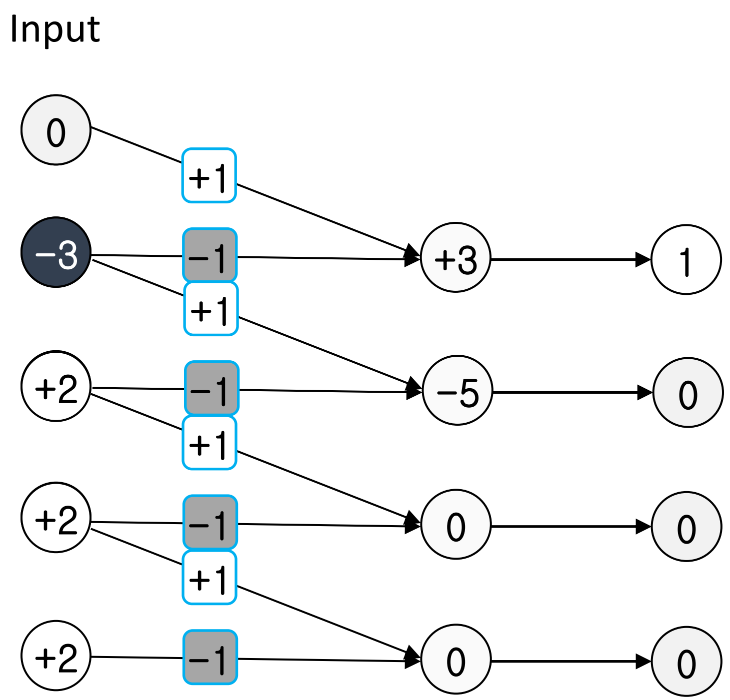
- Recall, a neuron can be thought of as learning to spot certain features in the input
- E.g., this neuron detects change from high to low (light to dark) between 3rd and 4th inputs

Multiplication vs convolution



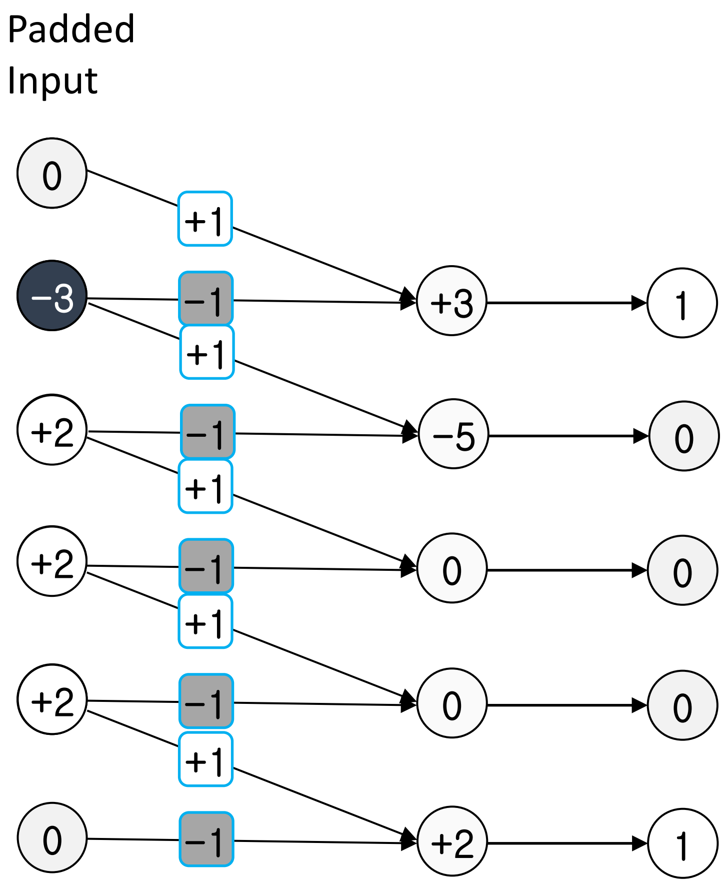
- Must have a new neuron for each new location of pattern???
- This is not efficient
- Solution: use convolution instead of multiplication

Multiplication vs convolution



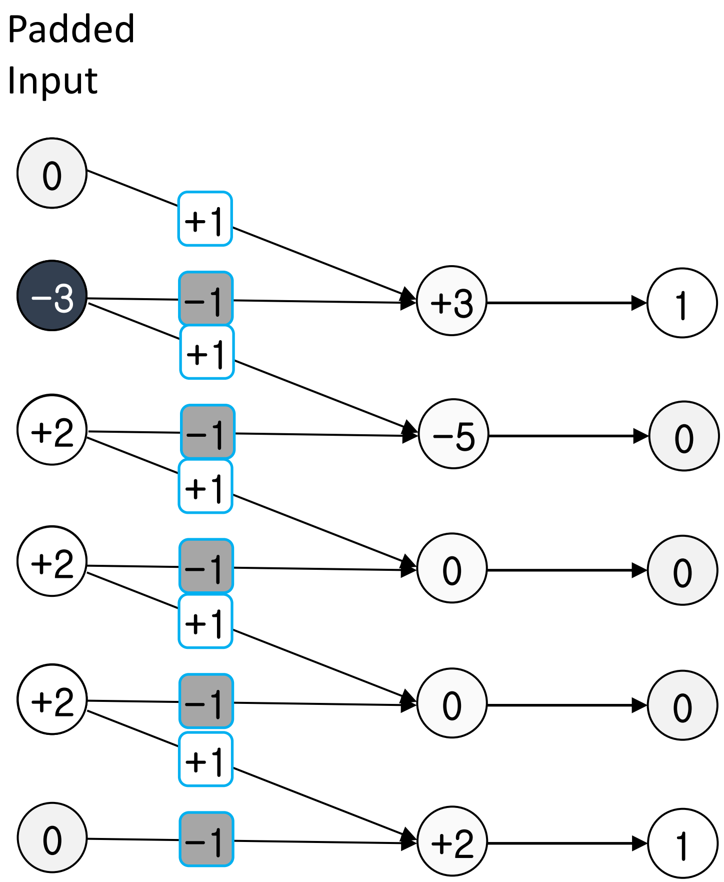
- New weights are of size 2 x 1; called filter, or kernel
- New output is the size of input minus 1 because of boundary
- New convolutional neurons all share the same weights! This is much more efficient; we learn the weights once instead of many times for each position

Multiplication vs convolution



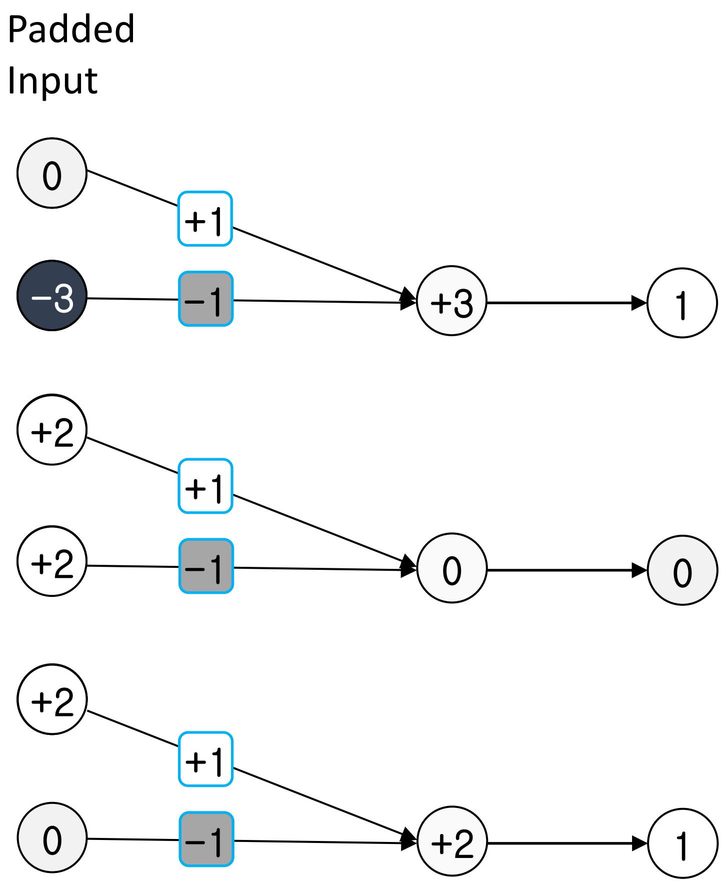
- New output is the size of input minus 1 because of boundary
- We can fix the boundary effect by padding the input with 0 and adding one more neuron

Multiplication vs convolution



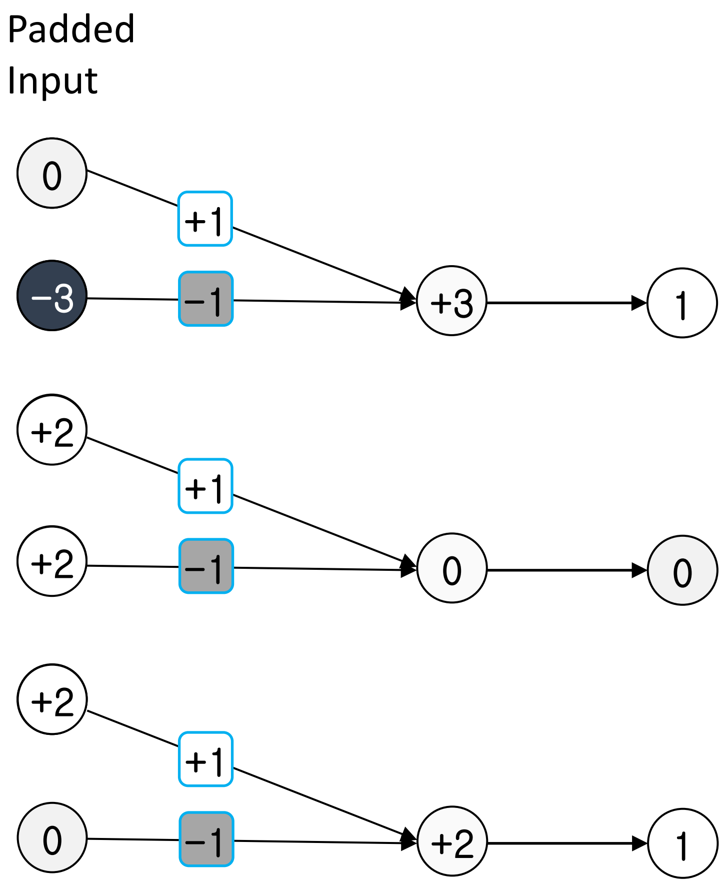
- Note, we move the filter by 1 each time, this is called stride

Multiplication vs convolution



- Note, we move the filter by 1 each time, this is called stride
- Stride can be larger, e.g. here is stride 2

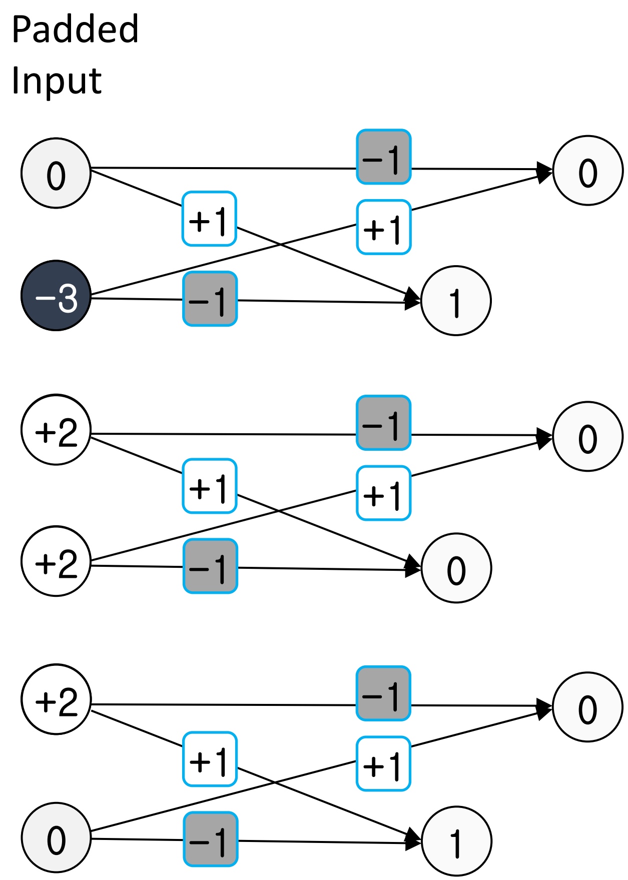
Multiplication vs convolution



To summarize, this layer has

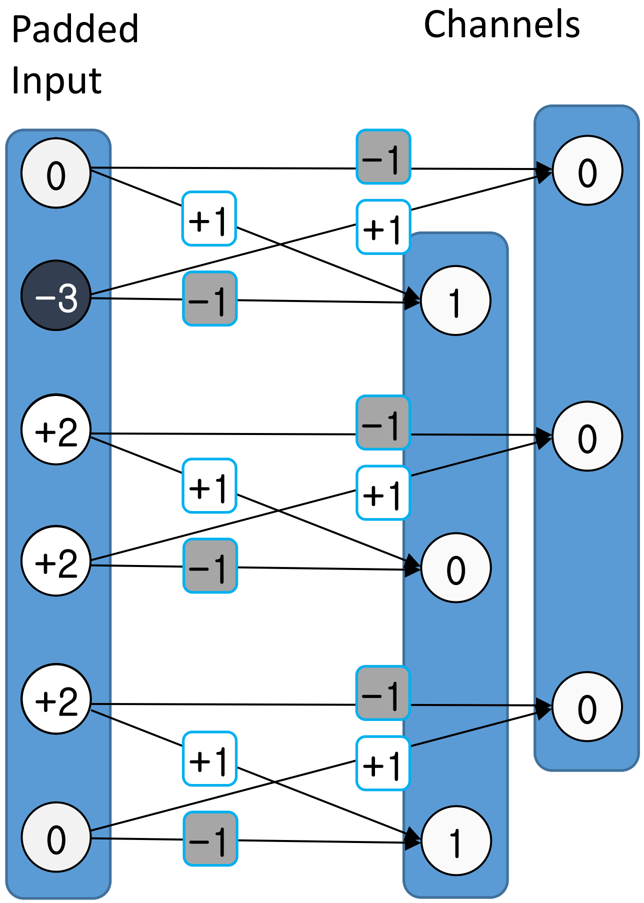
- `Input 5 x 1, padded to 6 x 1`
- `Kernel 2 x 1 with weights [+1,-1]`
- `Stride 2`
- `Output 3 x 1`

Multiplication vs convolution



- We can add another filter, this time to detect opposite change with weights [-1 +1]
- Unique filters are called channels

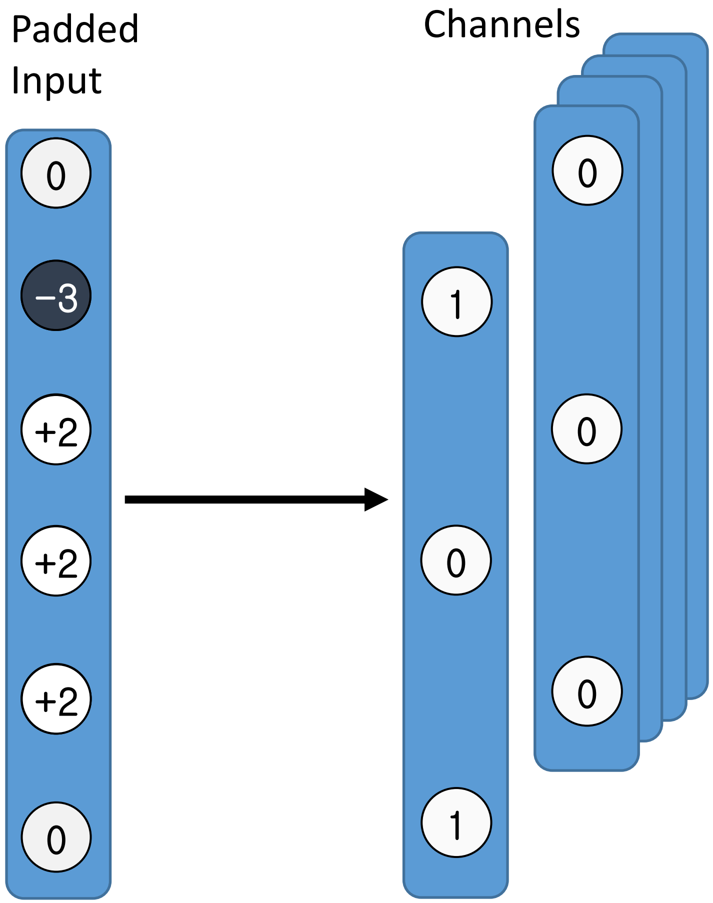
Multiplication vs convolution



- We can add another filter, this time to detect opposite change with weights [-1 +1]
- Unique filters are called channels

Multiplication vs convolution

simplified view



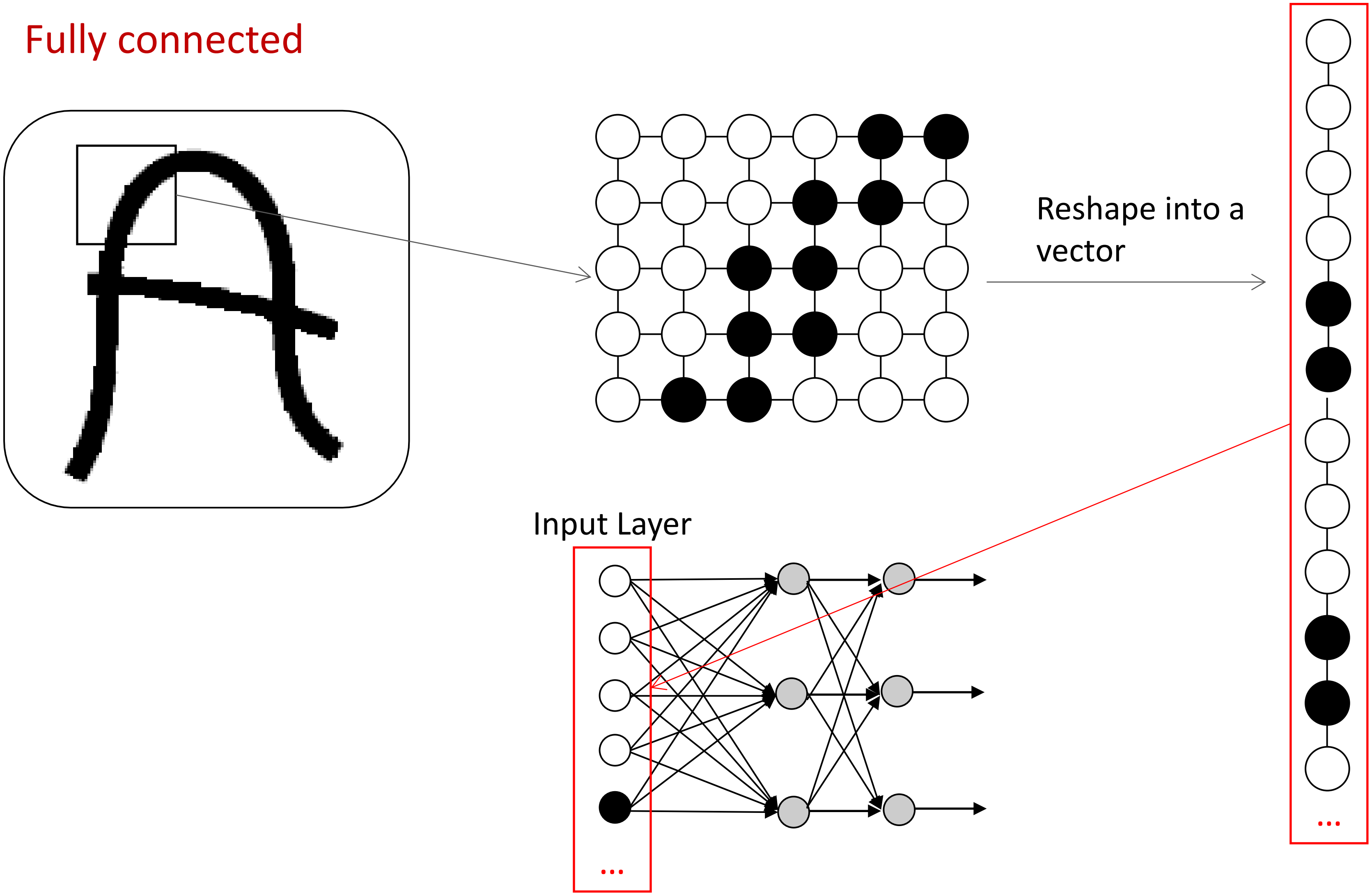
- We can add another filter, this time to detect opposite change with weights [-1 +1]
- Unique filters are called channels

Convolutional Neural Networks

For images and other 2-D signals

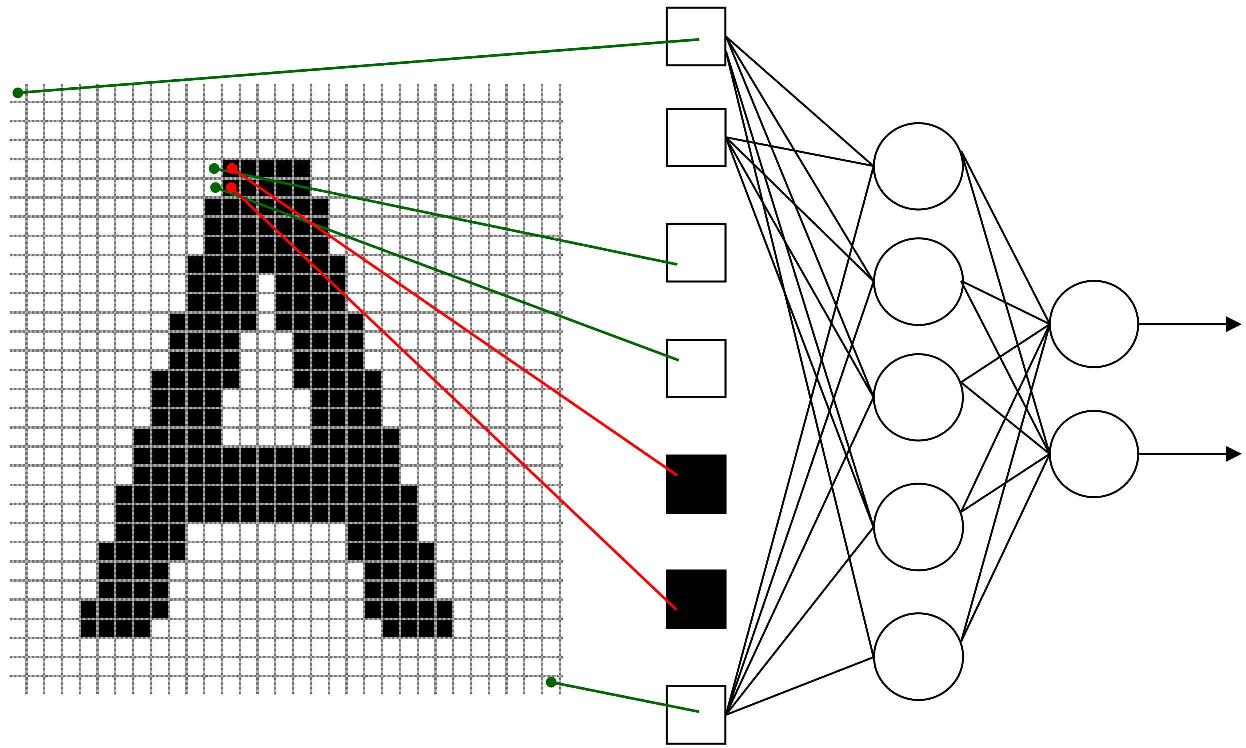
Representing images

Fully connected



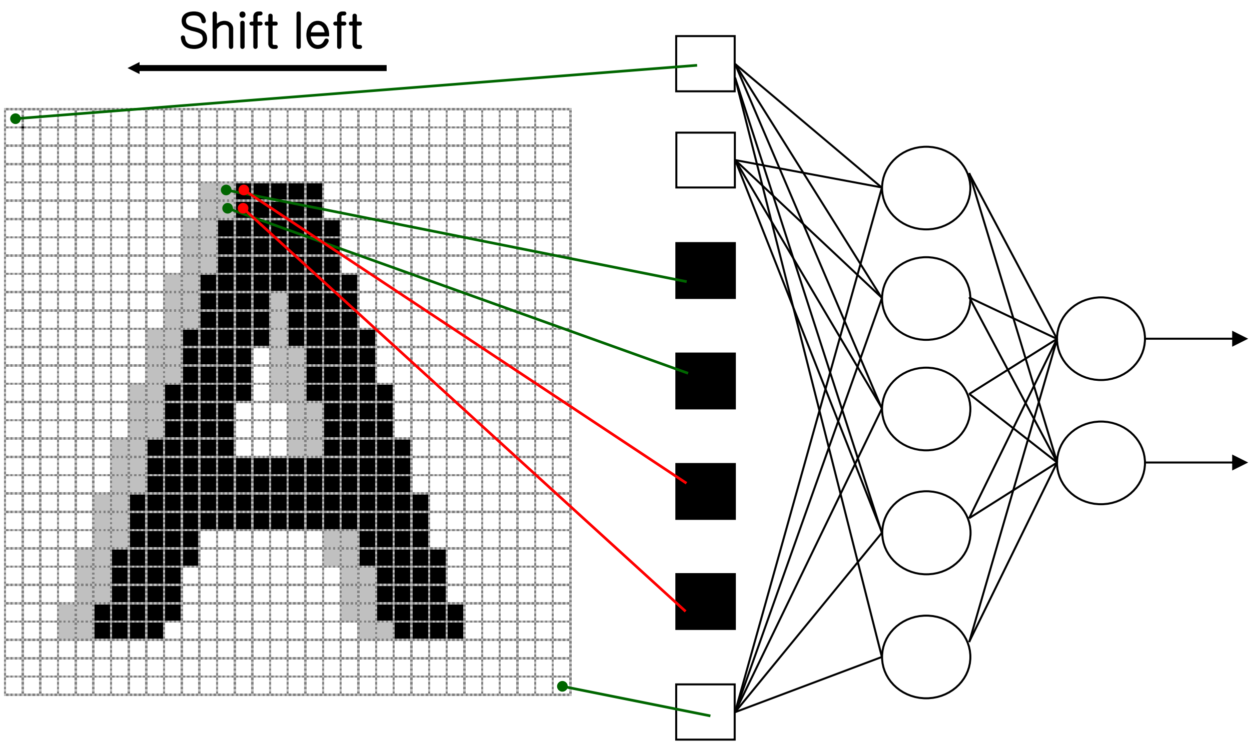
2D Input: fully connected network

Vectorize input by copying rows into a single column



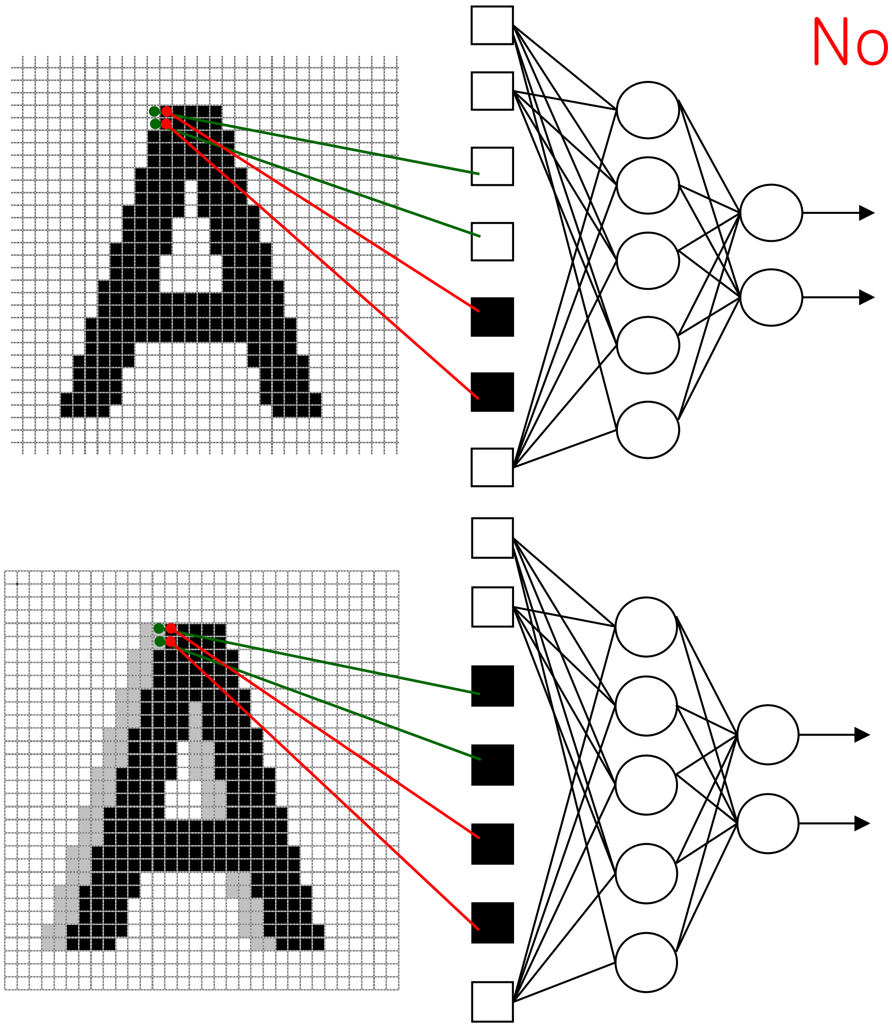
2D Input: fully connected network

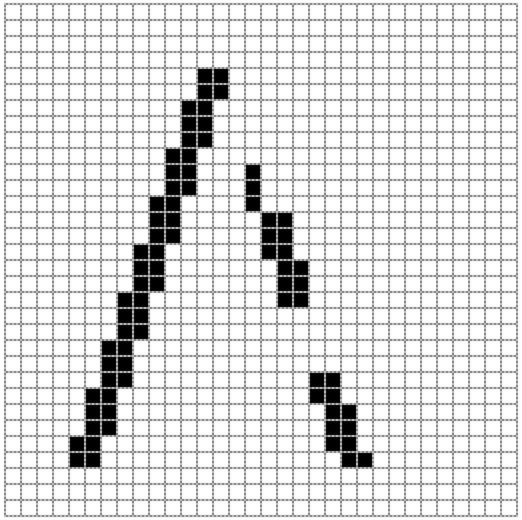
Problem: shifting, scaling, and other distortion changes location of features



2D Input: fully connected network

Not invariant to translation!

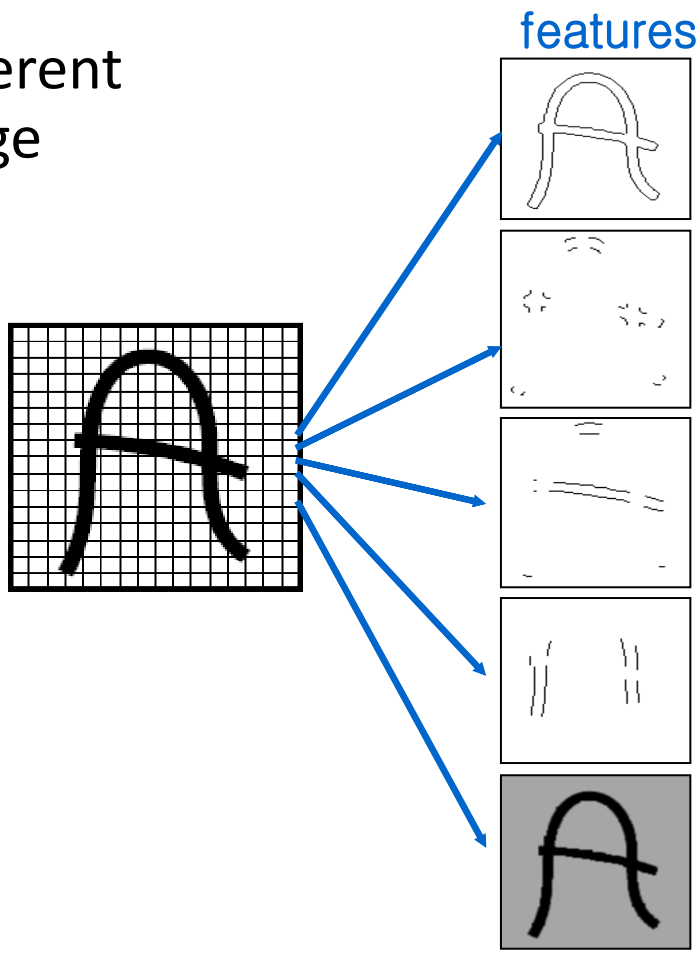




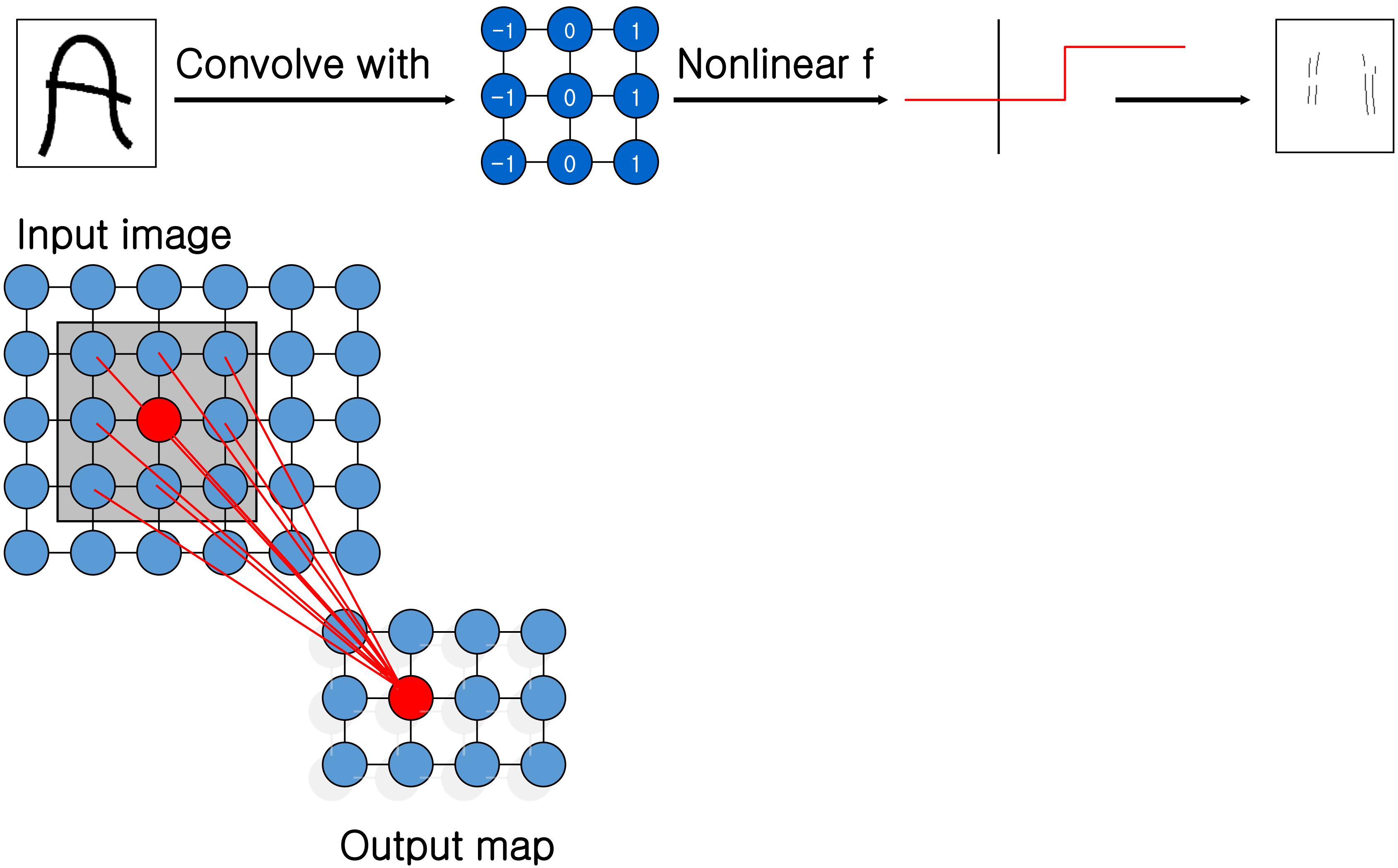
154 input change
from 2 shift left
77 : black to white
77 : white to black

Convolution layer in 2D

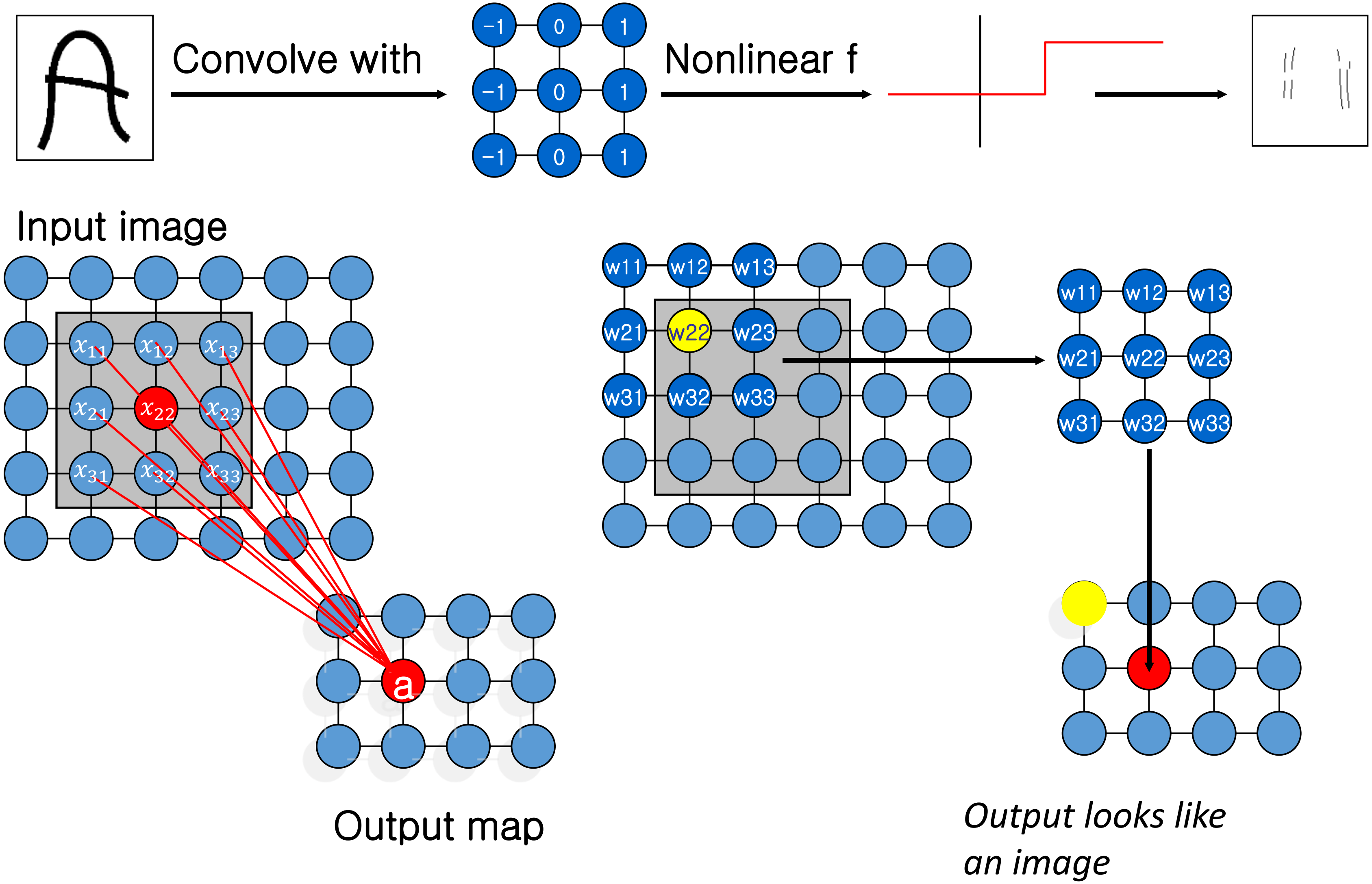
- detect the same feature at different positions in the input, e.g. image
- preserve input topology



Convolution layer in 2D



Convolution layer in 2D



$$a = f(w_{11}x_{11} + w_{12}x_{12} + w_{13}x_{13} + \cdots w_{33}x_{33})$$

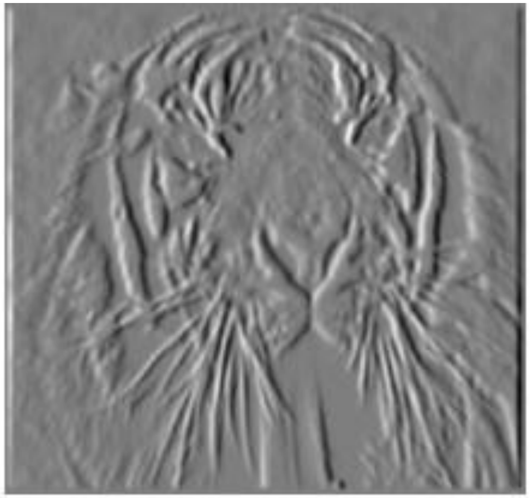
What weights correspond to these output maps?

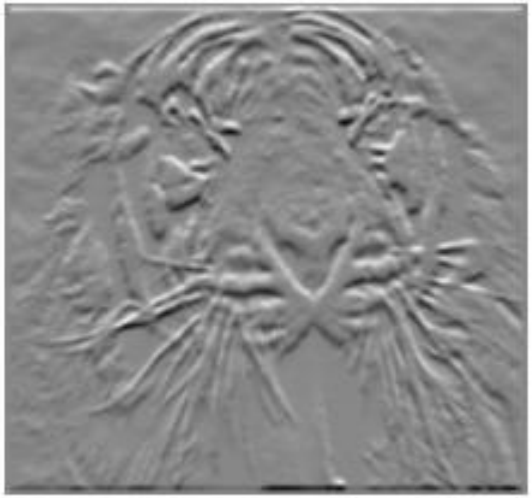
These are output maps before thresholding

Hint: filters look like the input they fire on

$$\frac{\partial f(x,y)}{\partial x}$$

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$



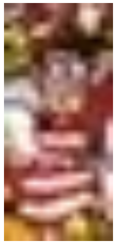


$$\frac{\partial f(x,y)}{\partial y}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Where is Waldo?

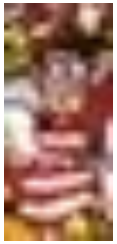
Input



filter

What will the output map look like?

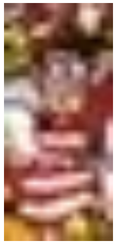
Input



filter

What will the output map look like?

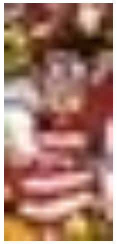
Output



filter

Here is Waldo

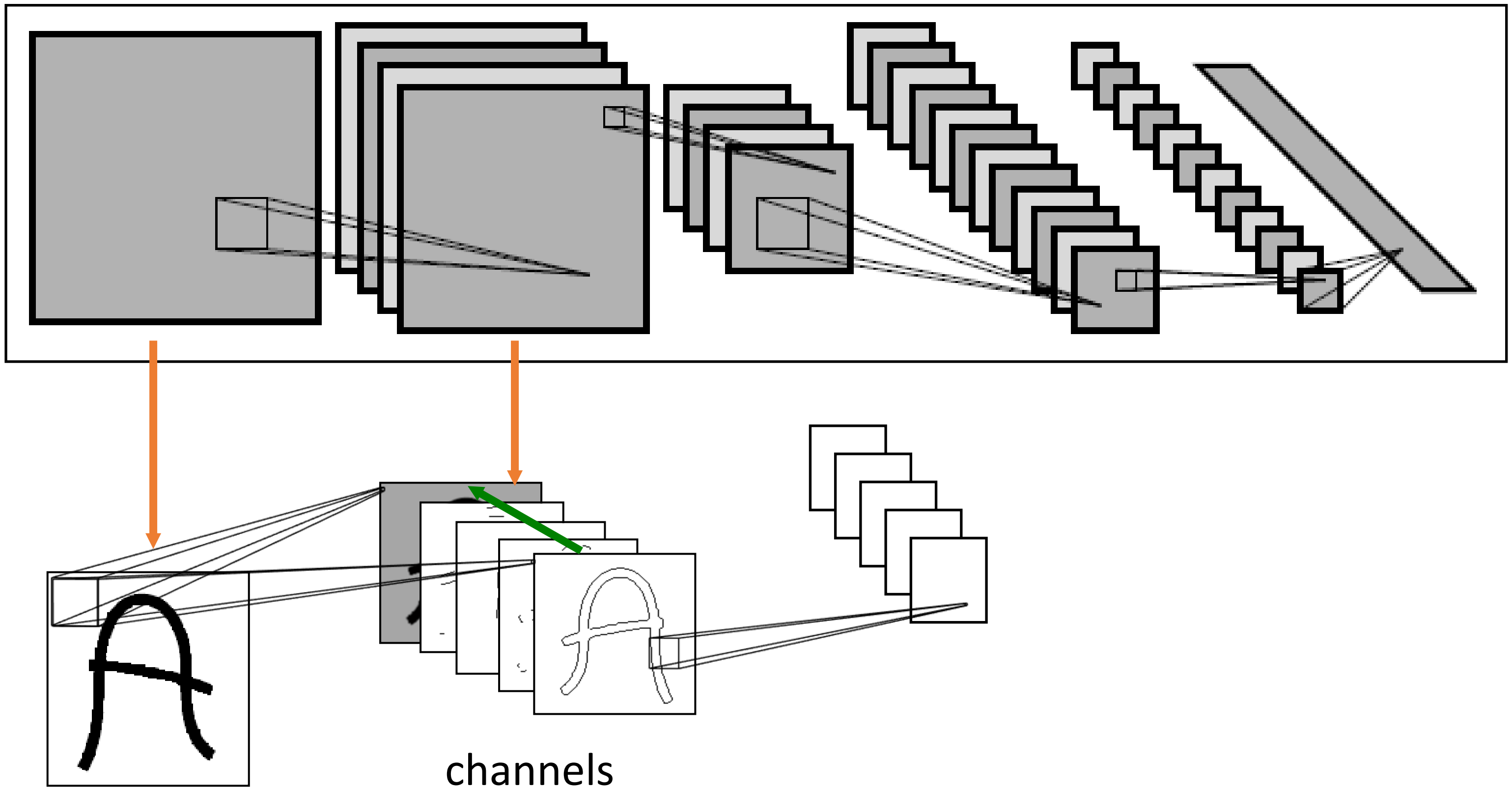
Input



filter

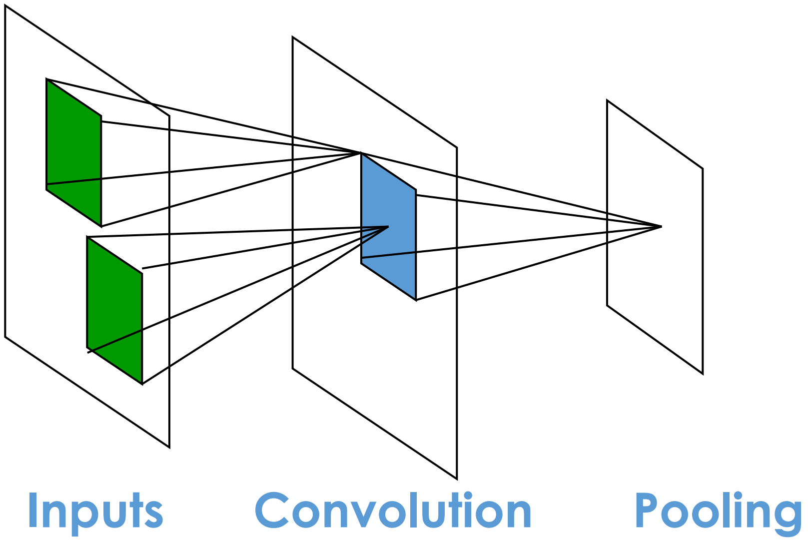
Stacking convolutional layers

- Each layer outputs multi-channel feature maps (like images)
- Next layer learns filters on previous layer's feature maps



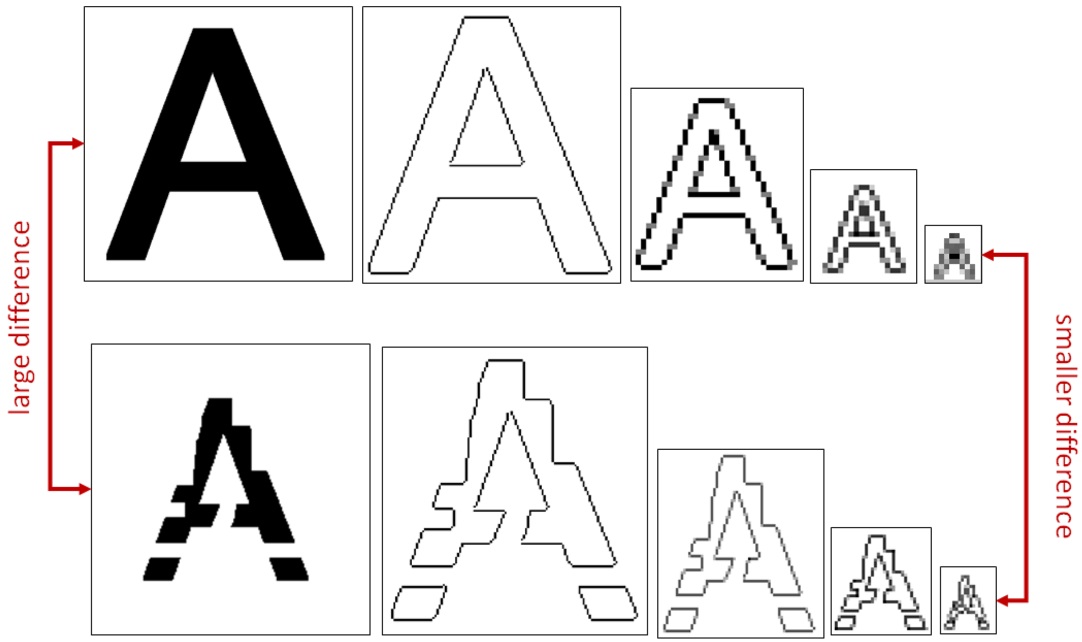
Pooling layers

- Convolution with stride > 1 reduces the size of the input
- Another way to downsize the feature map is with pooling
- A pooling layer subsamples the input in each sub-window
  - max-pooling: chose the max in a window
  - mean-pooling: take the average



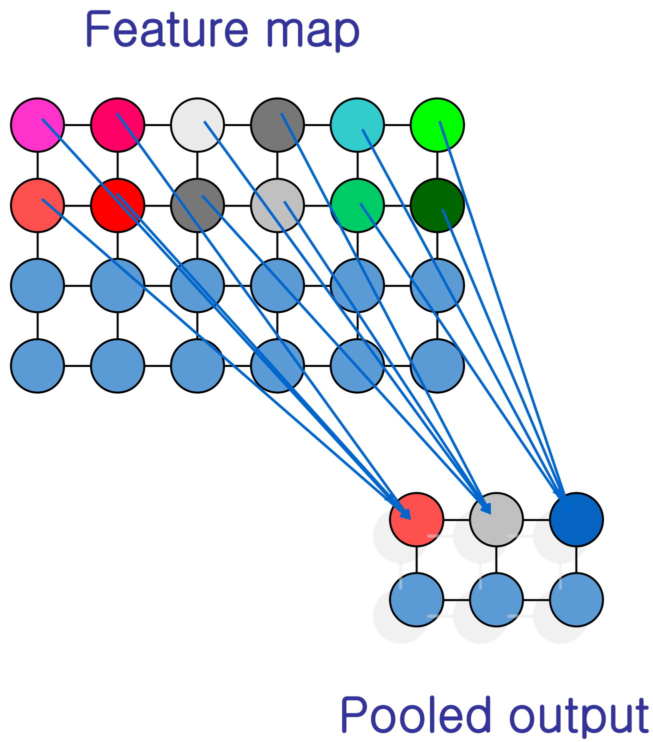
Pooling layer

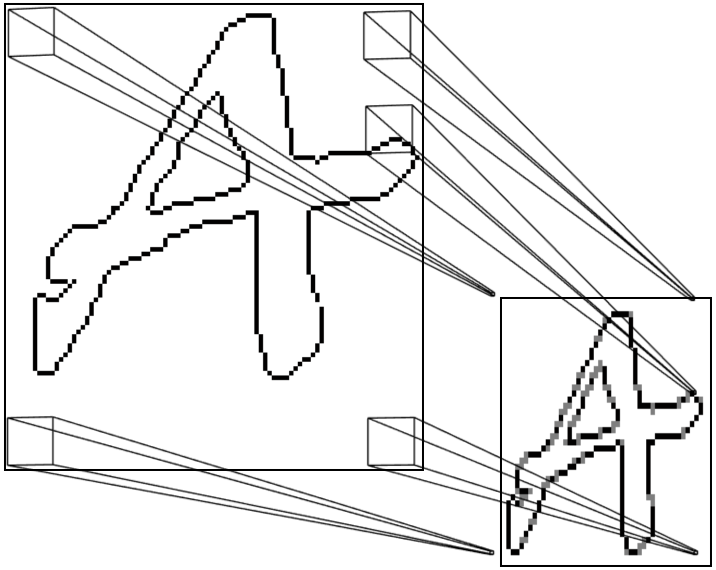
- the pooling layers reduce the spatial resolution of each feature map
- Goal is to get a certain degree of shift and distortion invariance



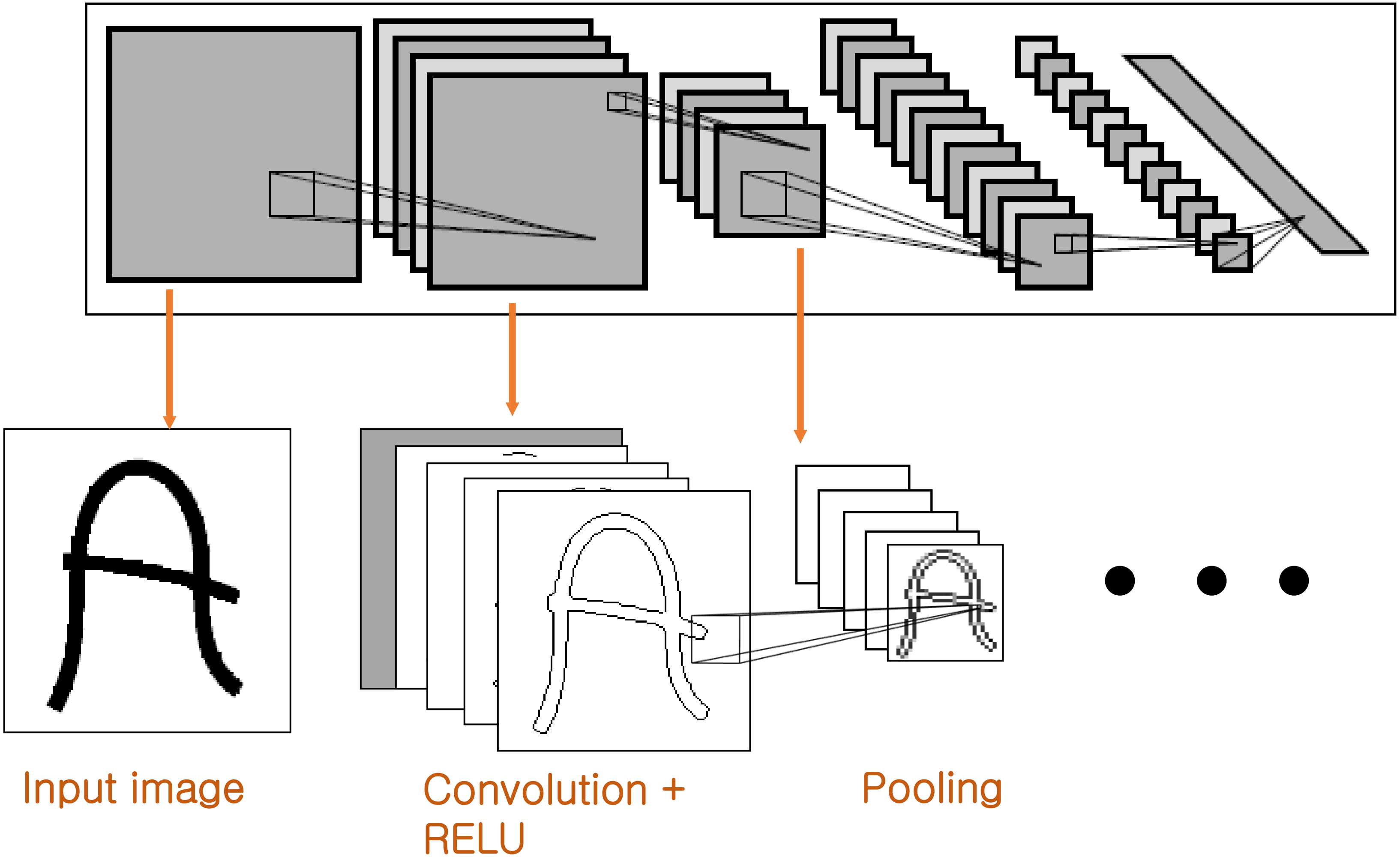
Pooling layer

- the weight sharing is also applied in pooling layers
- for mean/max pooling, no weights are needed



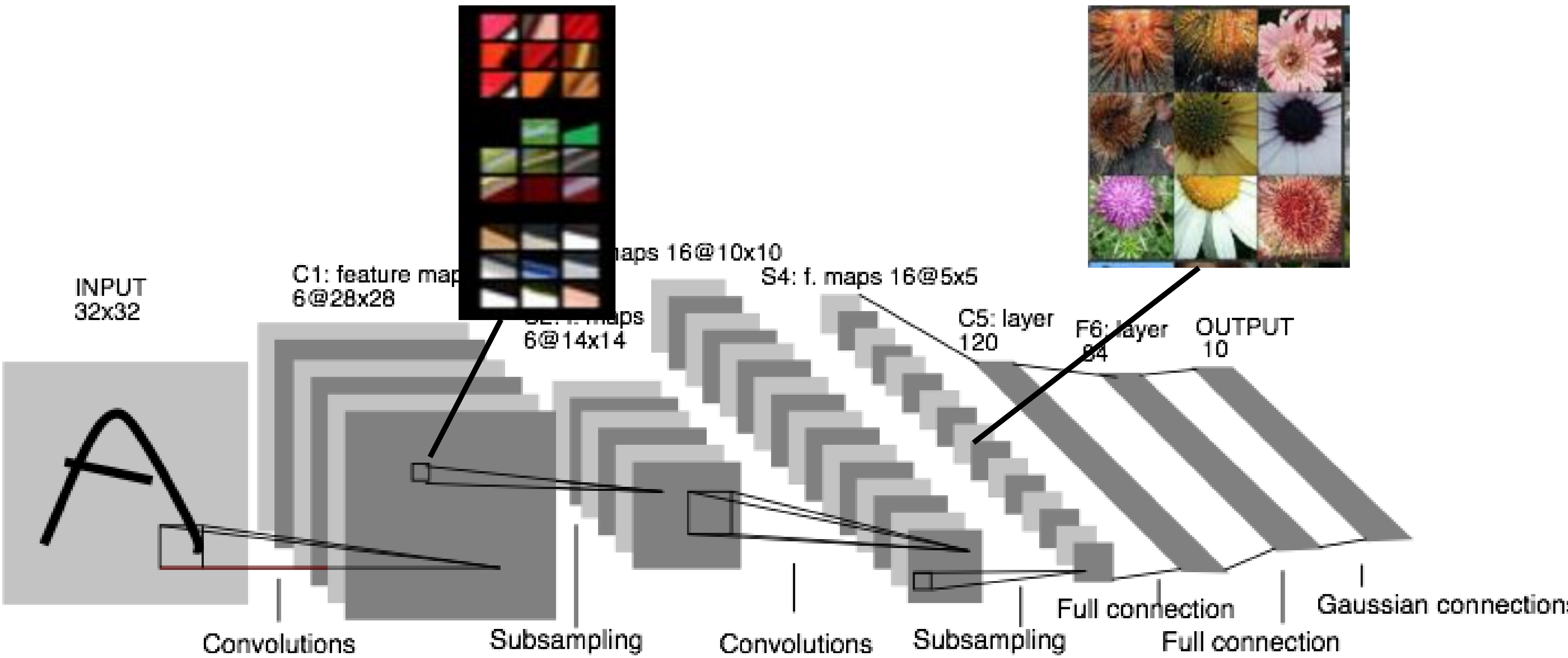


Putting it all together…



Convolutional Neural Network

A CNN is a better architecture for 2D signals



LeNet

Deep Convolutional Networks

The Unreasonable Effectiveness of Deep Features

Maximal activations of $pool_5$ units [R-CNN]

$conv_5$ DeConv visualization
[Zeiler-Fergus]

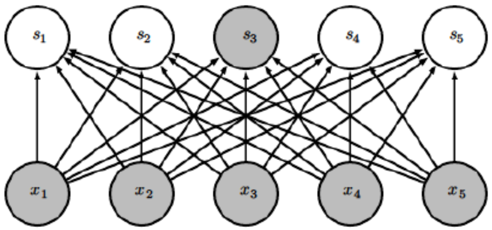
Rich visual structure of features deep in hierarchy.

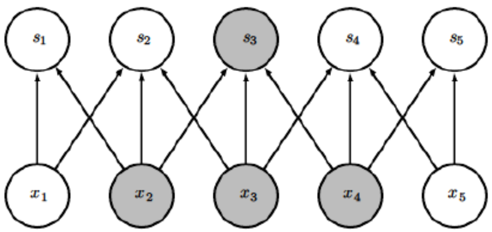
Convolutional Neural Nets

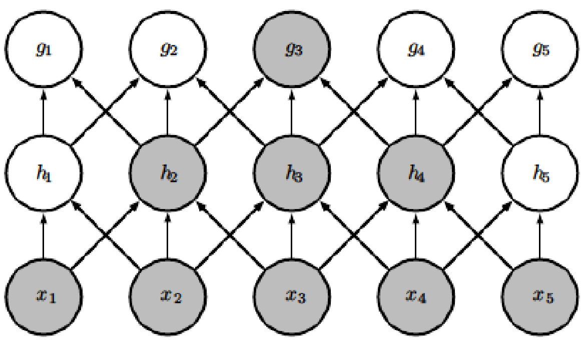
Why they rule

Why CNNs rule: Sparsity

- CNNs have sparse interactions, because the kernel is smaller than the input
- E.g. in thousands or millions pixel image, can detect small meaningful features such as edges
- Very efficient computation!
  - For m inputs and n outputs, matrix multiplication requires $O(m \times n)$ runtime (per example)
  - For k connections to each output, need only $O(k \times n)$ runtime
- Deep layers have larger effective inputs, or receptive fields

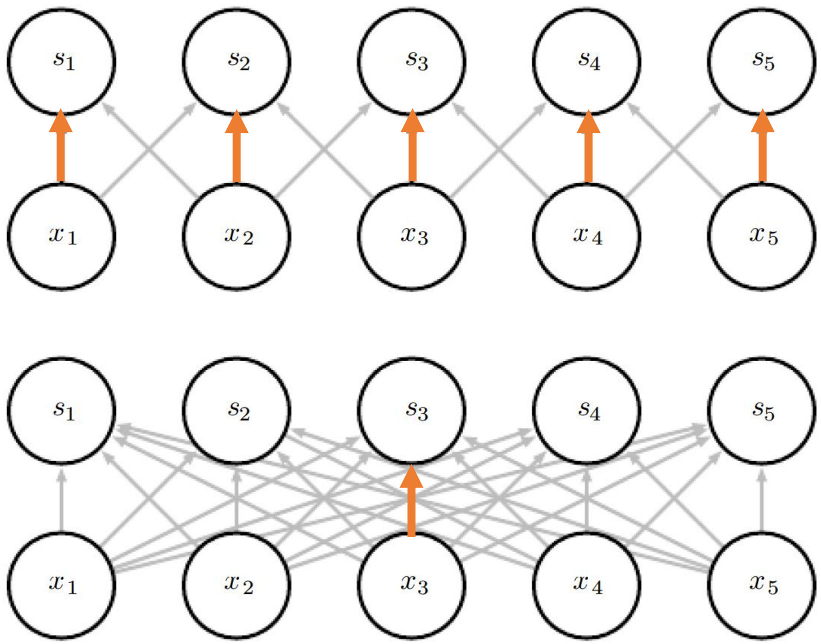






Why CNNs rule: Parameter sharing

- Kernel weights are shared across all locations
- Statistically efficient – learn from more data
- Memory efficient – store only *k* parameters, since $k << m$, this is much smaller than $m \times n$.

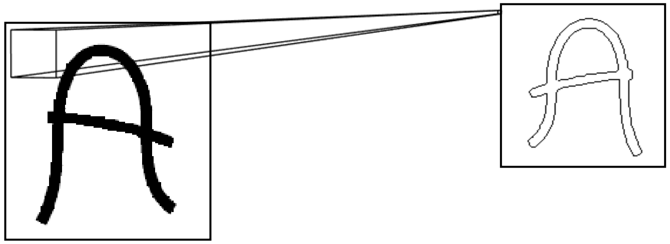


parameter used multiple times

parameter used only once

Why CNNs rule: Translation invariance

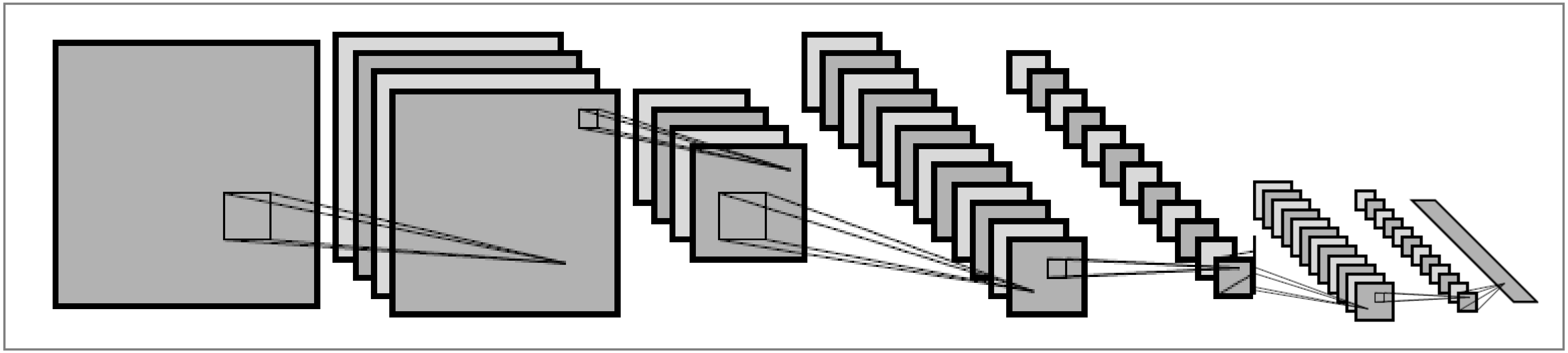
- Output is invariant to translation of input
  - spatial translation for images
  - temporal translation for time sequences
- useful when some function of a small local window is useful when applied to multiple input locations
- Note, not invariant to other transformations of input, such as large image rotation
- Pooling provides additional invariance to distortions



Convolutional Neural Nets

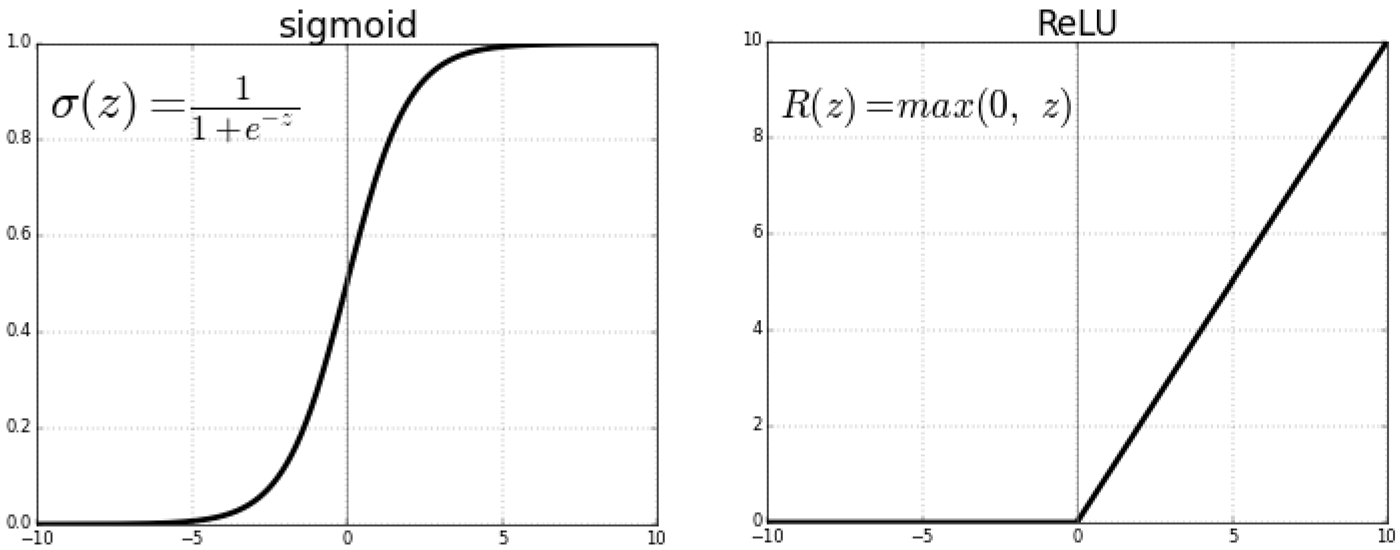
Example

CIFAR-10 Demo ConvJS Network

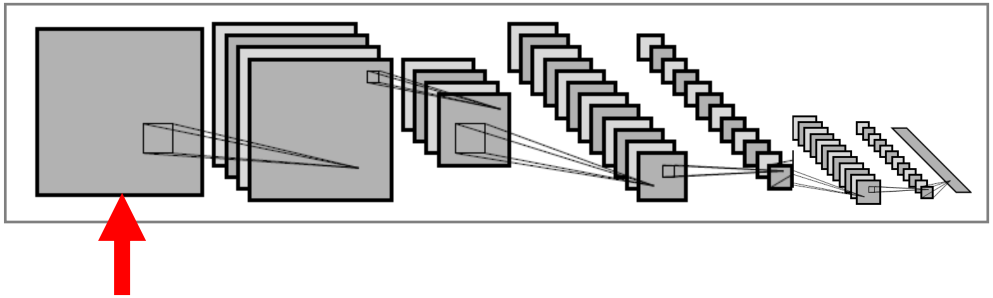


input image conv RELU pool conv RELU pool conv RELU pool fc

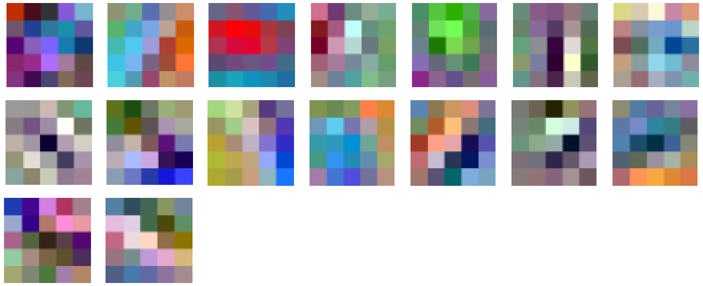
RELU: rectified linear unit



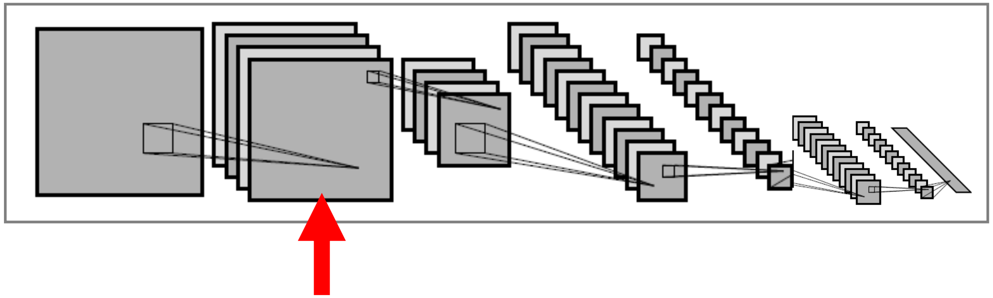
RELU function $g(x) = \max(0, x)$



filter size 5x5x3, stride 1



input (32x32x3)

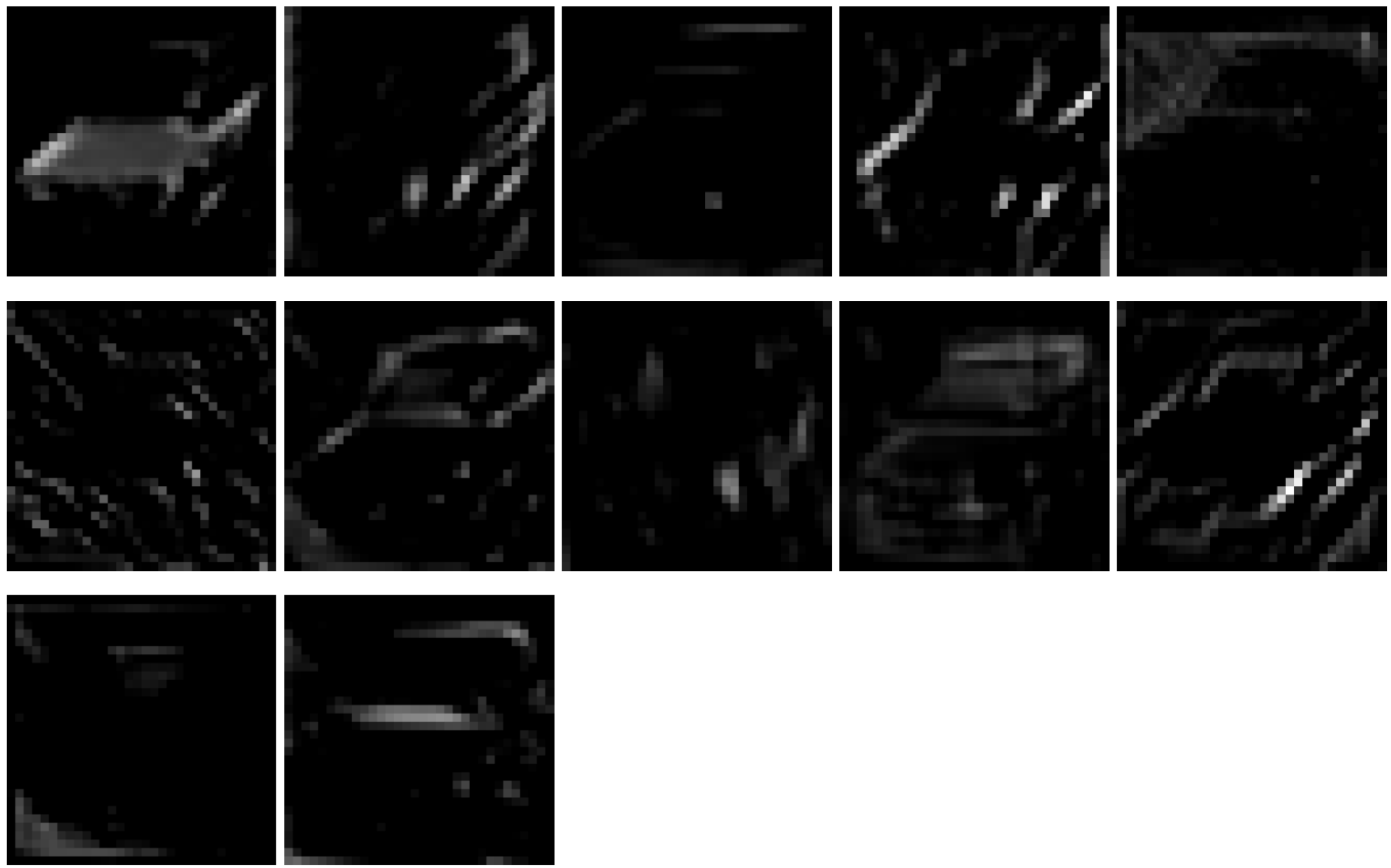


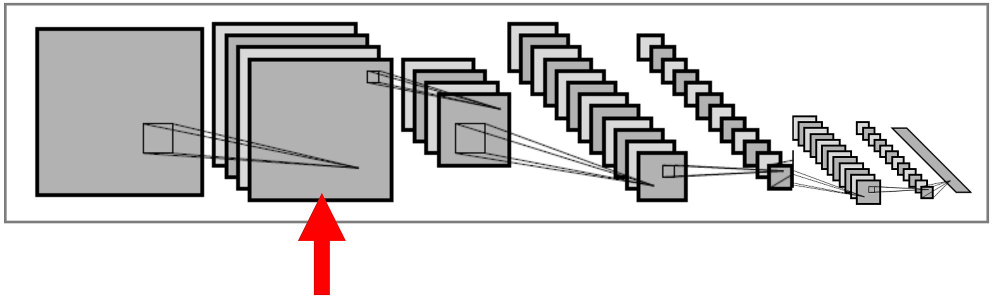
filter size 5x5x3, stride 1

input (32x32x3)

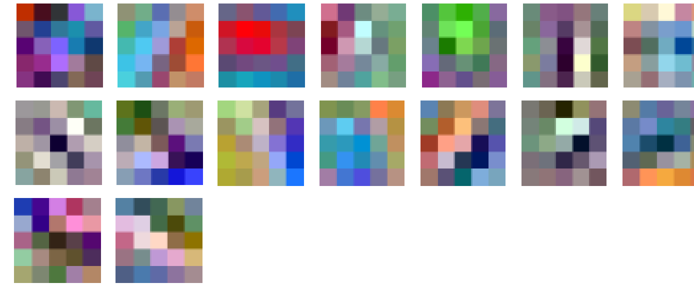
RELU

conv (32x32x16) params: 16x5x5x3+16 = 1216



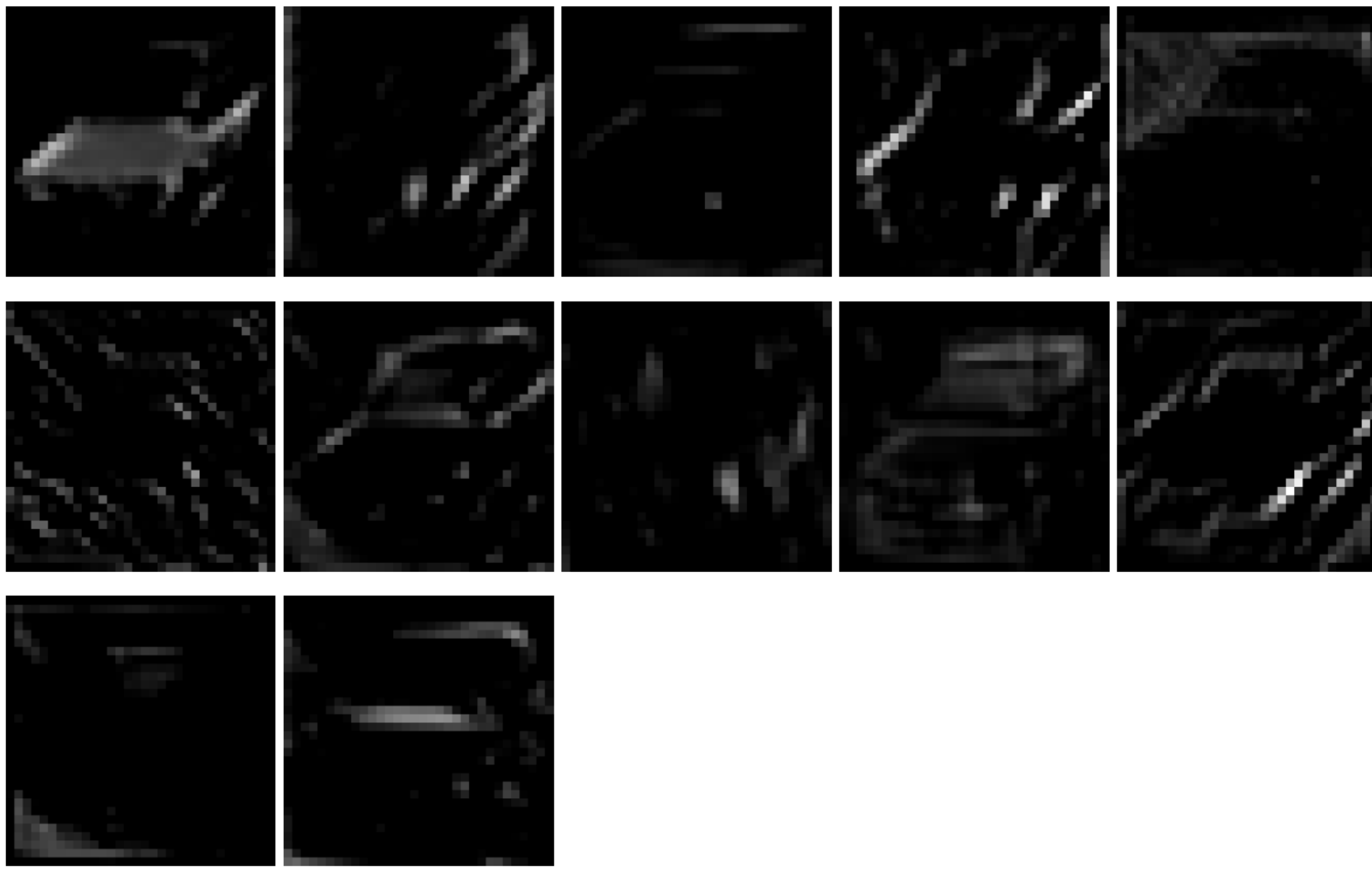


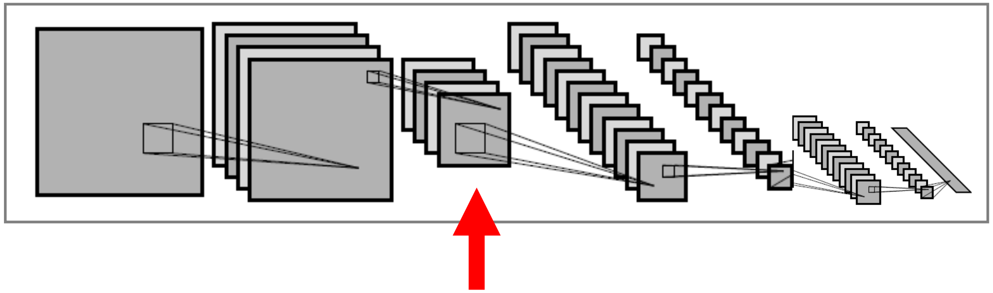
filter size 5x5x3, stride 1



input (32x32x3)

conv (32x32x16) params: 16x5x5x3+16 = 1216

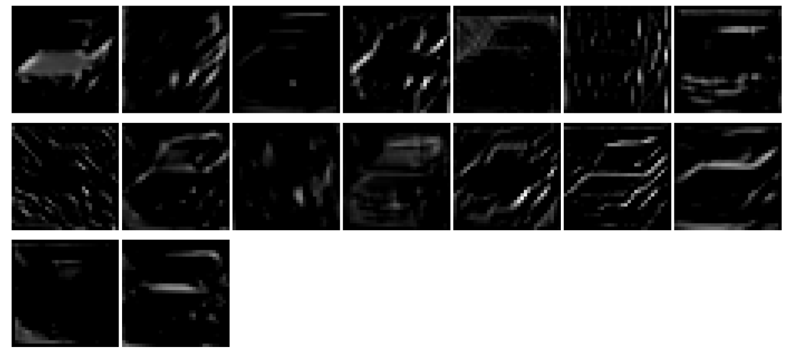




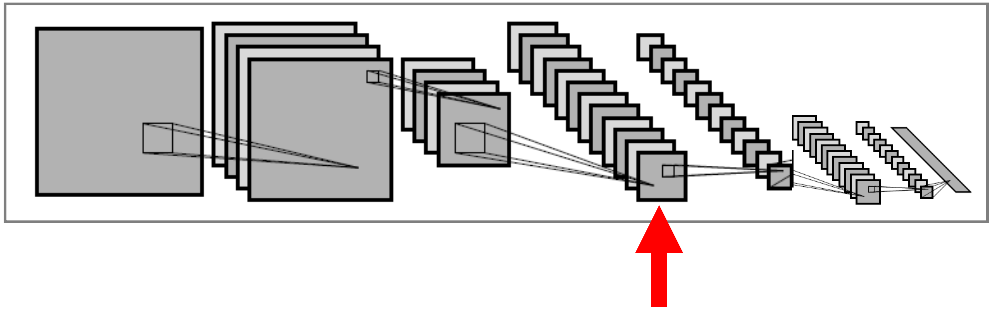
filter size 5x5x3, stride 1

input (32x32x3)

conv (32x32x16) params: 16x5x5x3+16 = 1216



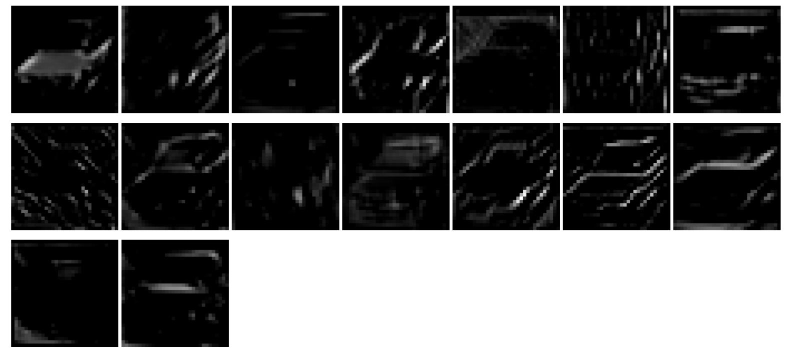
pool (16x16x16)
pooling size 2x2, stride 2



filter size 5x5x3, stride 1

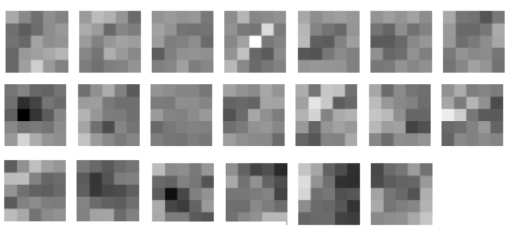
input (32x32x3)

conv (32x32x16) params: 16x5x5x3+16 = 1216



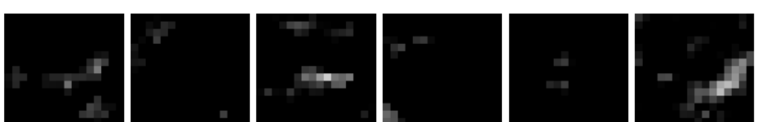
pool (16x16x16)
pooling size 2x2, stride 2

filter size 5x5x16, stride 1

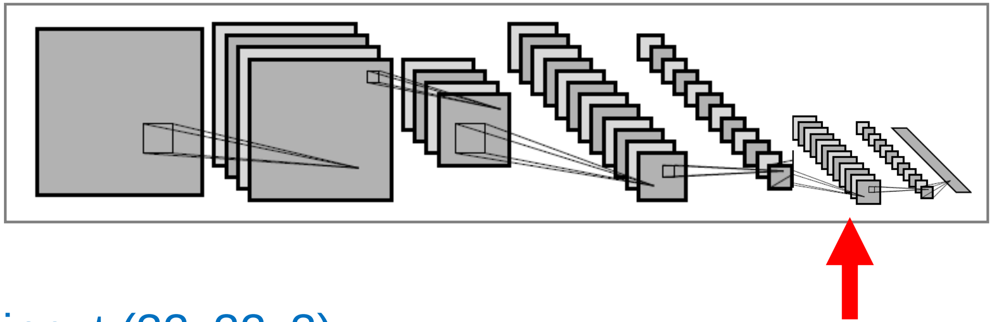


RELU

conv (16x16x20) params: 20x5x5x16+20 = 8020



pool (8x8x20)
pooling size 2x2, stride 2



One more conv+RELU+pool:

conv (8x8x20)
filter size 5x5x20, stride 1
relu (8x8x20)
pool (4x4x20)
pooling size 2x2, stride 2
parameters: 20x5x5x20+20 = 10020

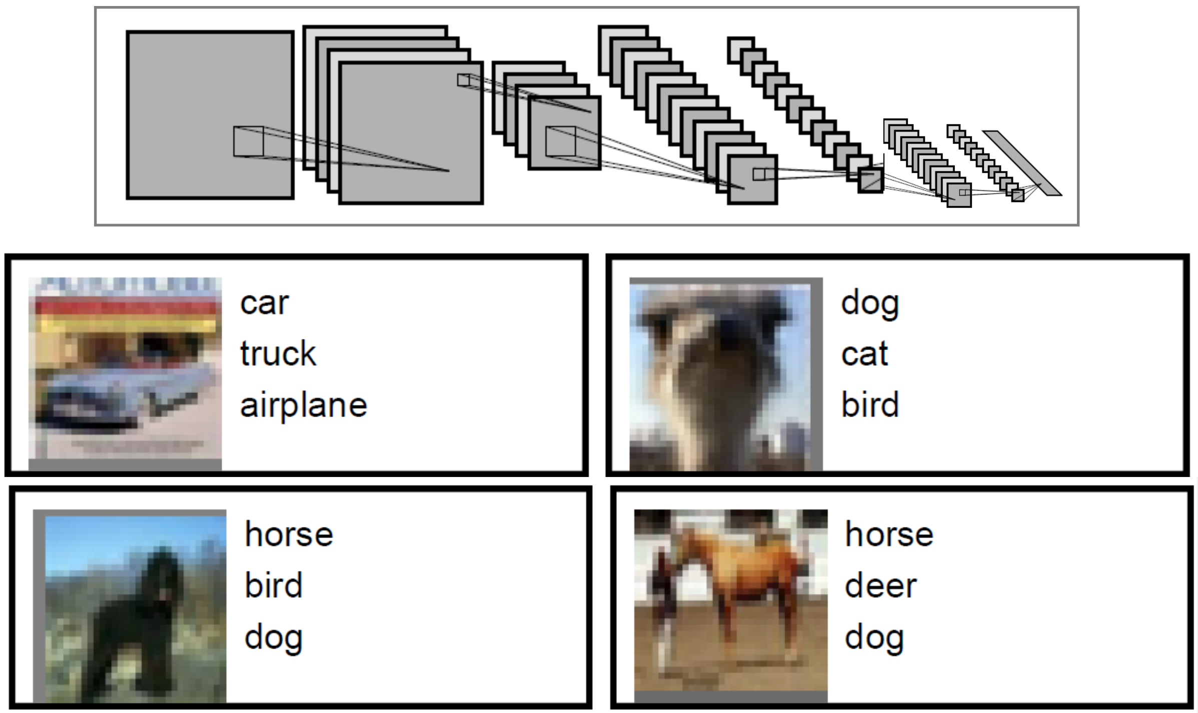
input (32x32x3)

fc (1x1x10); parameters: 10x320+10 = 3210

softmax (1x1x10)

Testing the network

- Show top three most likely classes



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html