Deep Learning

Chapter 9 - Convolutional Networks

Cherif Jazra
@jazracherif
YouTube Part 1 Part 2

Comments section is open

Chapter Content

Part 1:

9 Intro
9.1 The Convolution Operation
9.2 Motivation
9.3 Pooling
9.4 Convolution and Pooling as an Infinitely Strong Prior
9.5 Variants of the Basic Convolution Function

Part 2:

9.6 Structured Outputs
9.7 Data Types
9.8 Efficient Convolution Algorithms
9.9 Random or Unsupervised Features
9.10 The Neuroscientific Basis for Convolutional Networks
9.11 Convolutional Networks and the History of Deep Learning

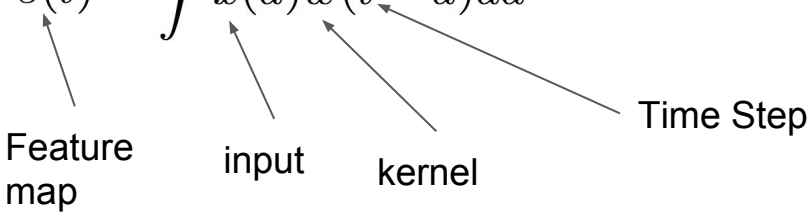
Introduction

A special kind of neural networks adapted to grid like input:
1-D Time Series
2-D Image data

Uses a mathematical operation **Convolution** instead of plain **matrix multiplication** in at least 1 layer of the network.

A weighted average of a function at every point:

$$s(t) = \int x(a)w(t-a)da \qquad s(t) = (x * w)(t)$$



Ex: A noisy laser sensor used to estimate a spaceship position.

→ Do a weight average of measurements that gives more weight to recent ones

1-D Discrete Version, t is an integer.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

2-D version

Kernel

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n).$$

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n).$$

Commutative Property: K and I are interchangeable. True because the Kernel is be **flipped**

In practice, the **Cross-correlation** is used, Kernel is **not** flipped

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n).$$

2D-Example

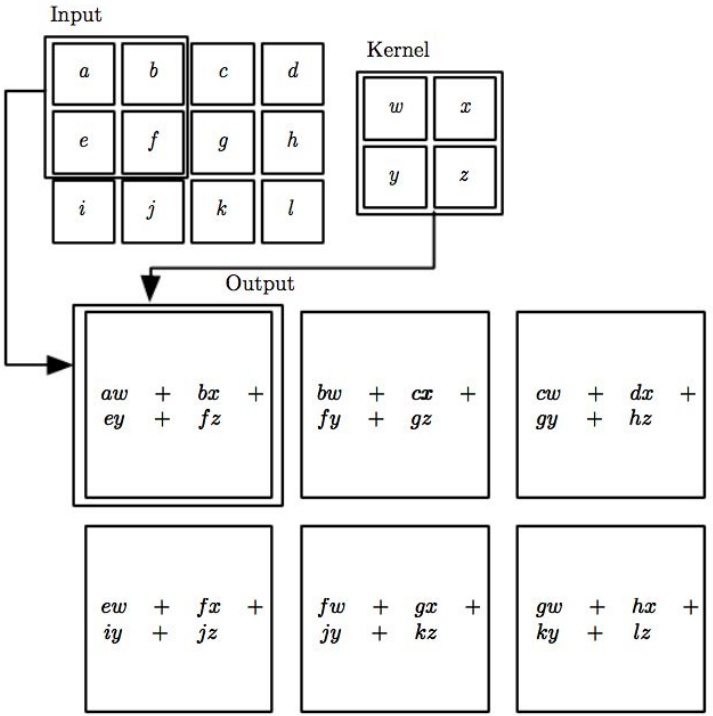
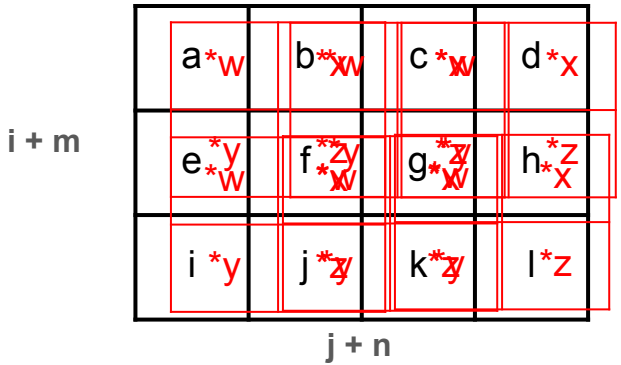


Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called "valid" convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

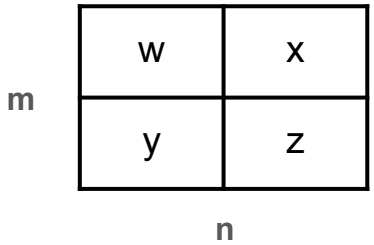
9.1 The Convolution Operation

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n).$$

Input (3x4)

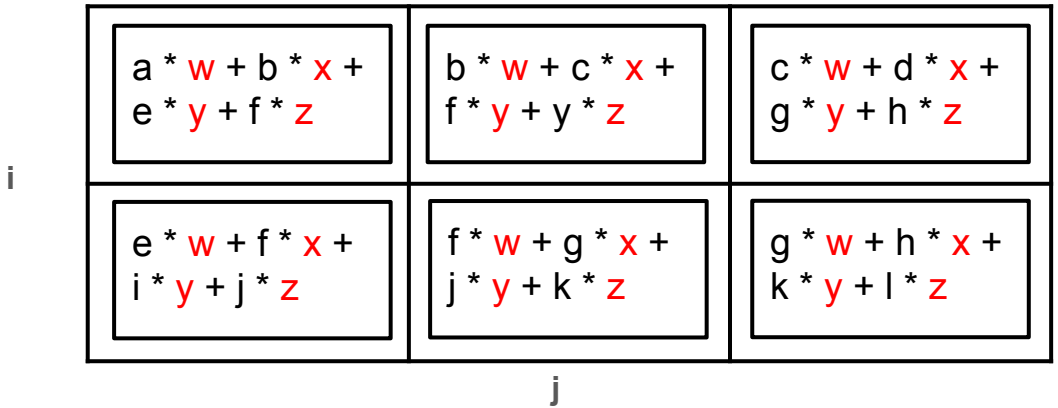


Kernel (2x2)



The VALID Convolution: Output is restricted to positions where the kernel lies entirely within the image

Feature map S (2 x 3)



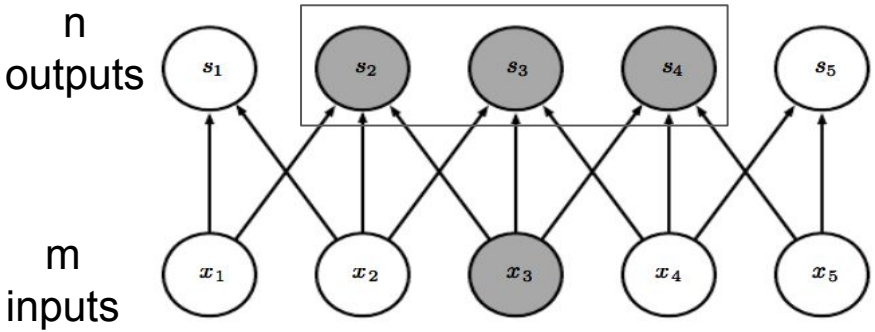
Why use a convolution?

1) Sparse Interactions
2) Parameter Sharing
3) Equivariant Representations

1) Sparse Interactions / Sparse Connectivity / Sparse weights

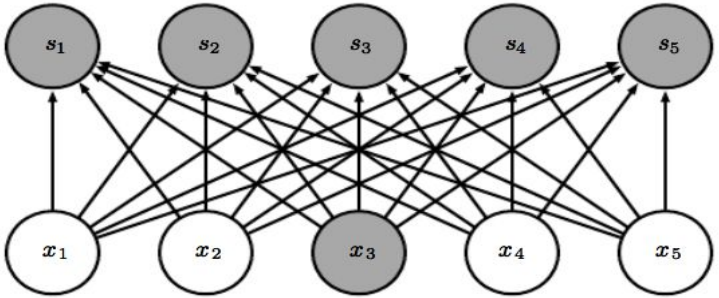
Output units are not connected to all input units since kernel size is smaller than input
⇒ Fewer Parameters!
⇒ Fewer Computations!

Sparse (k =3)



O(k * n) k<<m

Dense

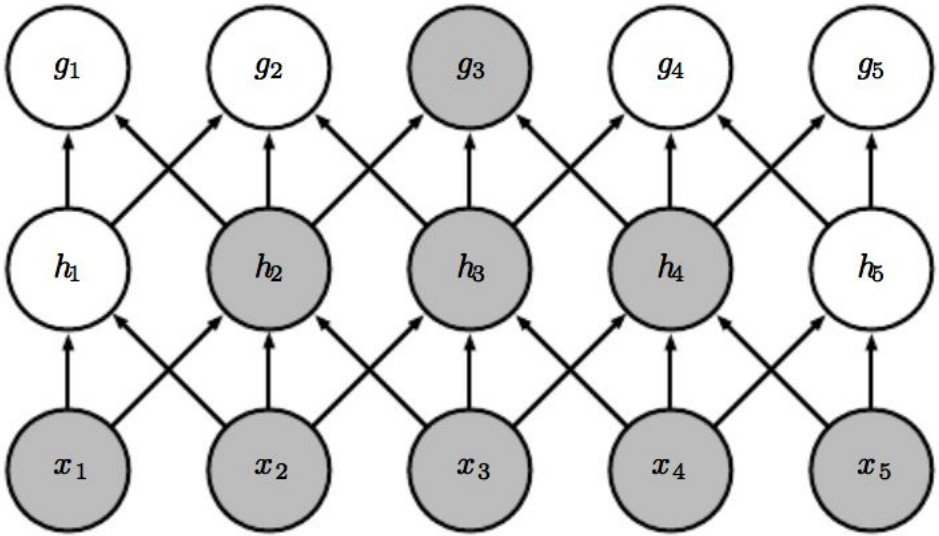


O(m * n)

Each edge is 1 parameter

1) Sparse Interactions / Sparse Connectivity / Sparse weights

Receptive Field: units in deeper unit are indirectly connected to most of the input image

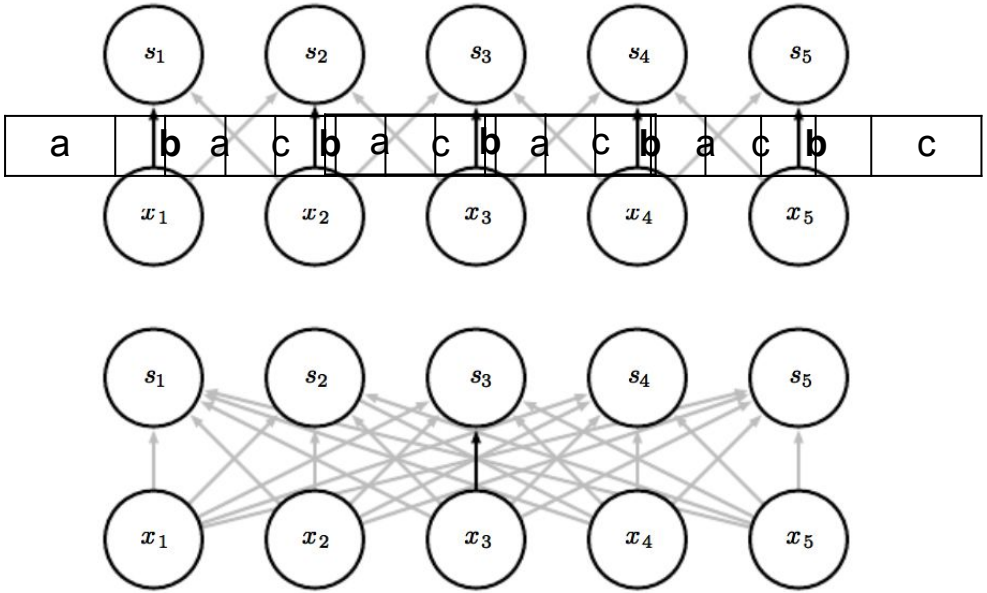


2) Parameter Sharing / tied weights

The same parameter is used multiple time: each member of the kernel is used at each location of the input pixel

Ex: Kernel of size 3

Black line = the **central** parameter of the kernel



3) Equivariance to Translation

The convolution transformation is equivariant to **translation**: “If the input changes, the output changes in the same way”

Convolving + shifting ⇔ Shifting + Convolving

1-D ex: “if we move an event later in time in the input, the exact same representation of it will appear in the output, but later in time”

2-D ex: “if we move the object in the input, its representation will move the same amount in the output, at a different location”
Ex: detecting edges in the lower level of an image

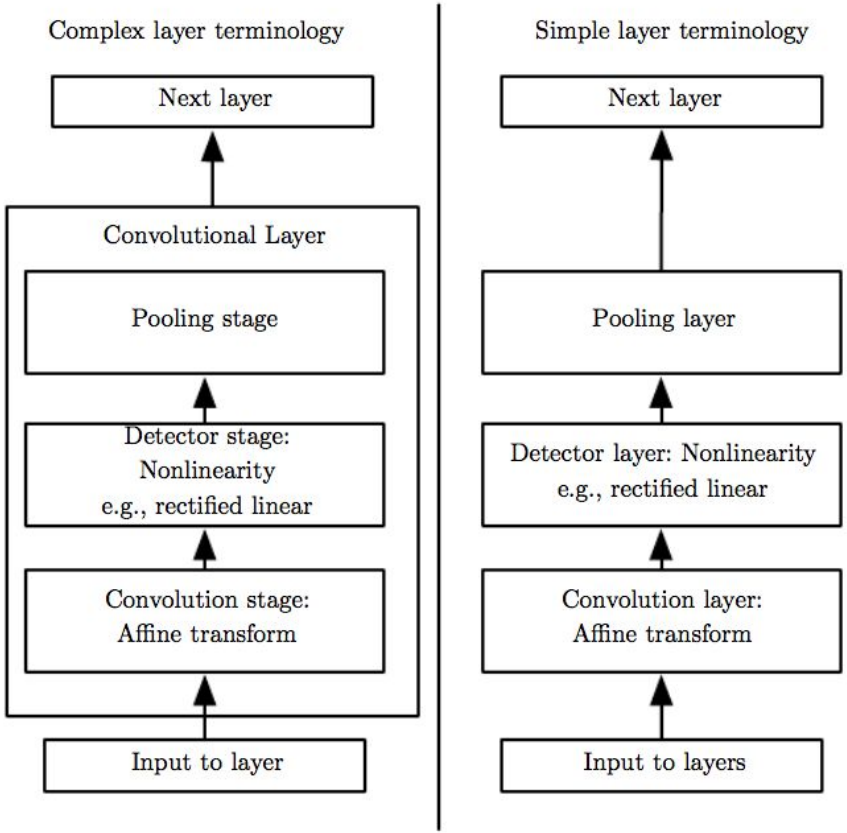
⇒ This characteristic allows the network to represent in a similar way, pictures with object translated in them, therefore facilitating pattern identification in the higher levels

NOTE: Convolution is NOT equivariant with regards to **Rotation** or **scaling**.

The **Convolution layer** typically refers to 3 phases:

1. Convolution Stage
2. Detector Stage
3. Pooling Stage

Left terminology used in the book



What is Pooling?

Conceptually, the pooling operation consists of replacing a certain area of the input with summary statistics of the area. Ex: **max-pooling** replace a region with the maximum value within that region
Other types: **Avg Pool**, **L^2 norm**, **Weight average**.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

max-pool

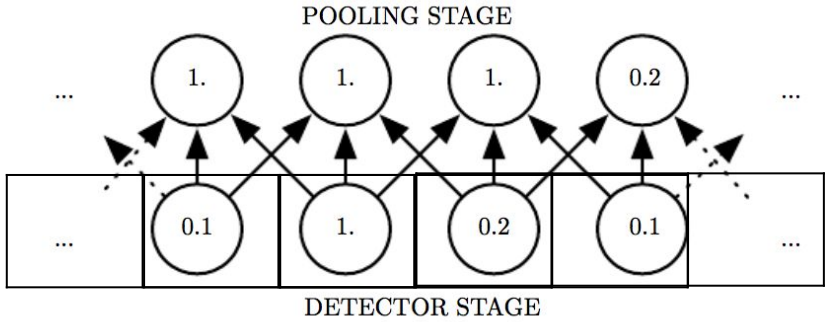
6	8
14	16

Why Pooling?

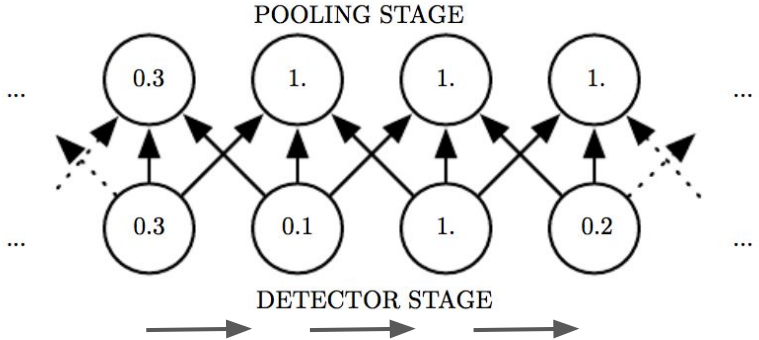
To make the representation approximately **invariant** to small **translations.**

MAX-POOL-3

Example 1

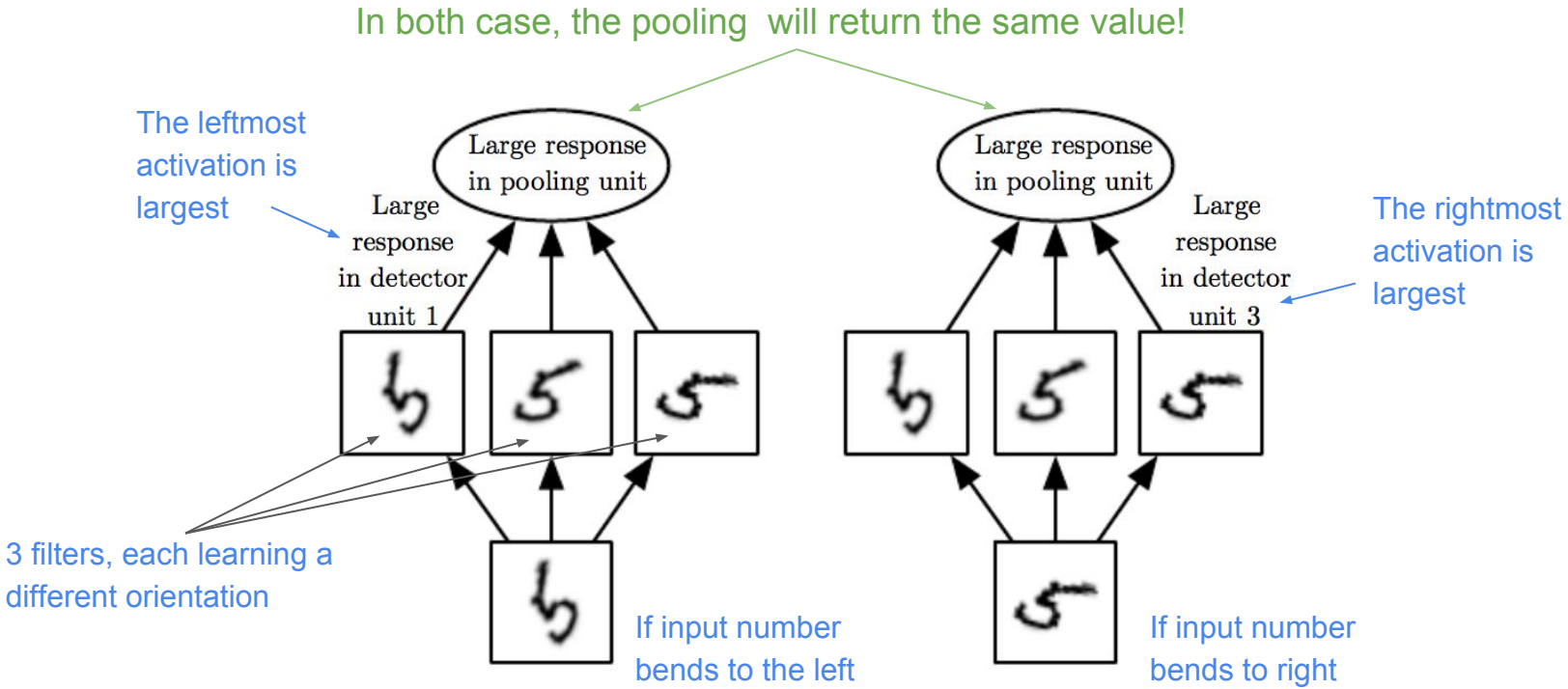


Example 2



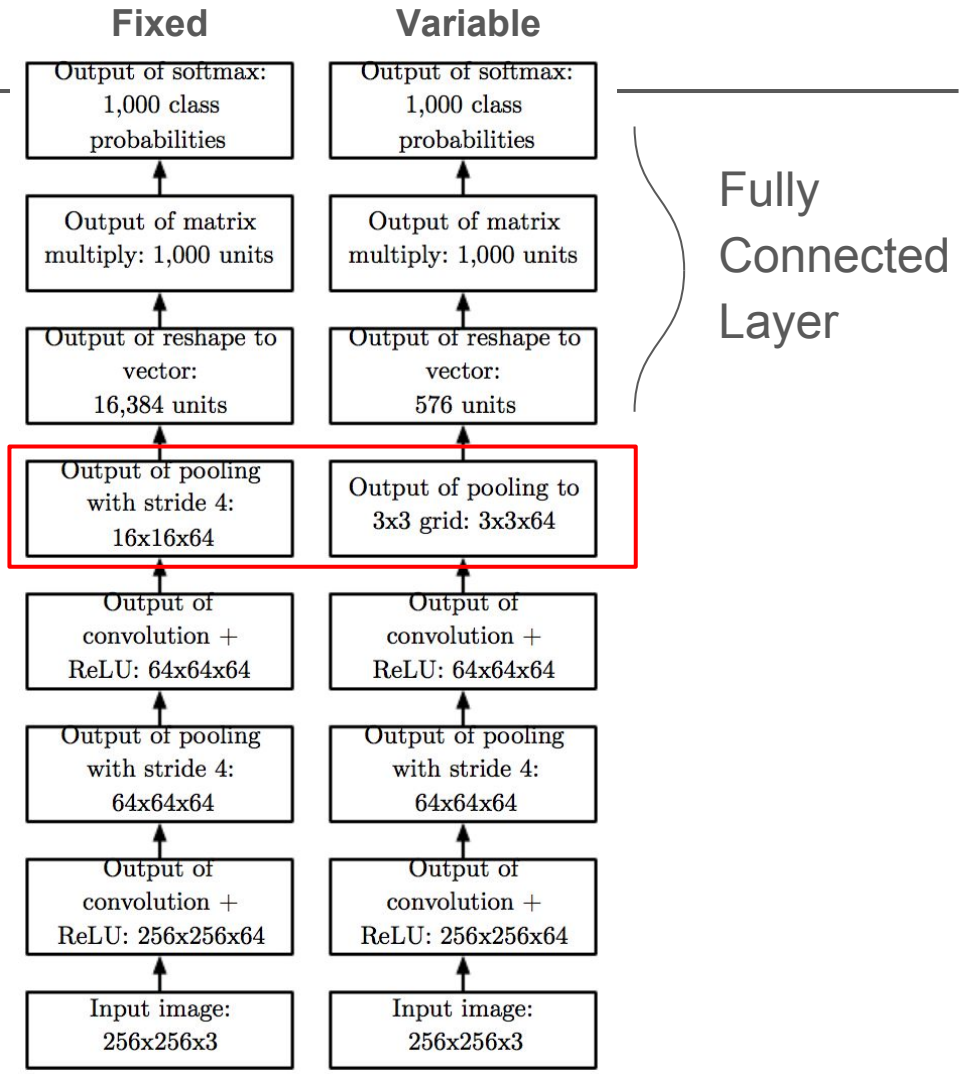
Shift input one node to the right

If we pool over the output of different kernels, the features can learn which transformation to become invariant to



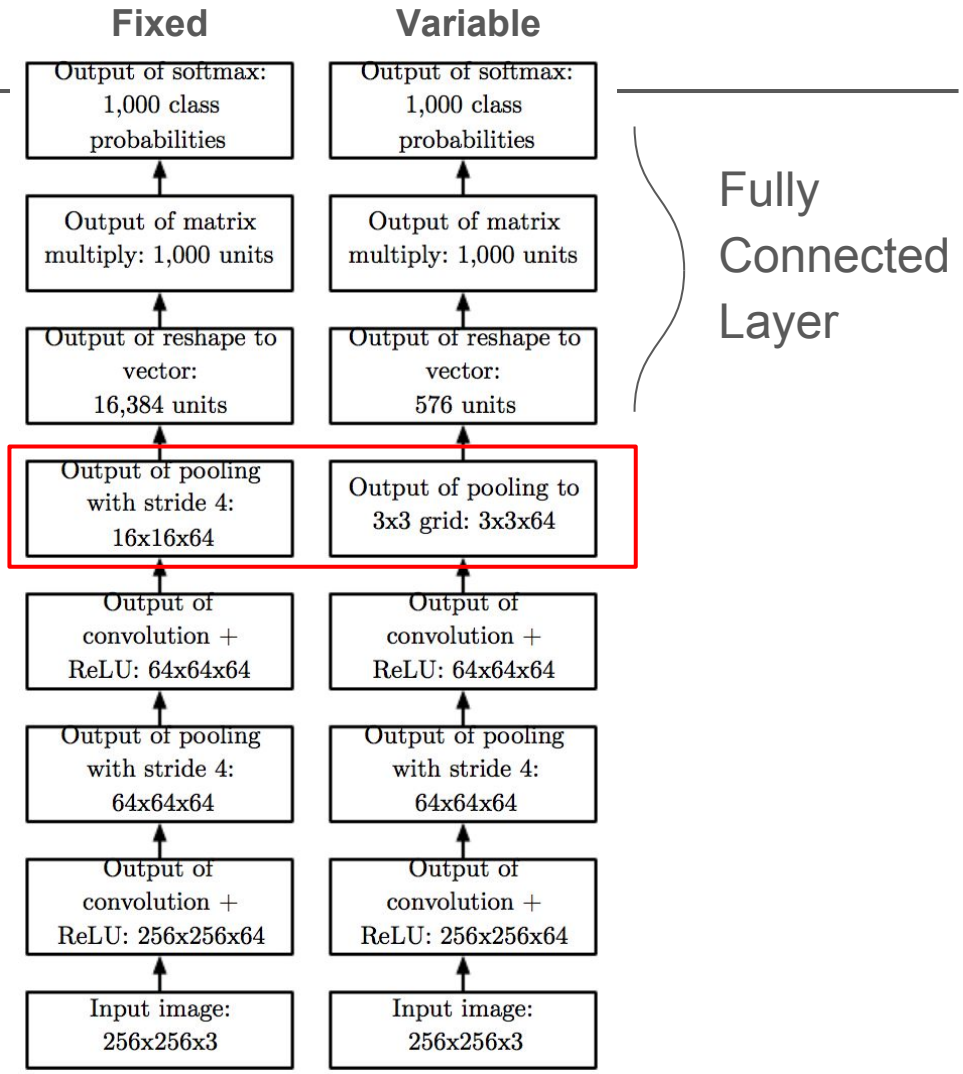
9.3 Pooling

Pooling can help handle images of **varying** size. As the last layer before the classification layer, it can be set to have a fixed output grid size



9.3 Pooling

Pooling can help handle images of **varying** size. As the last layer before the classification layer, it can be set to have a fixed output grid size



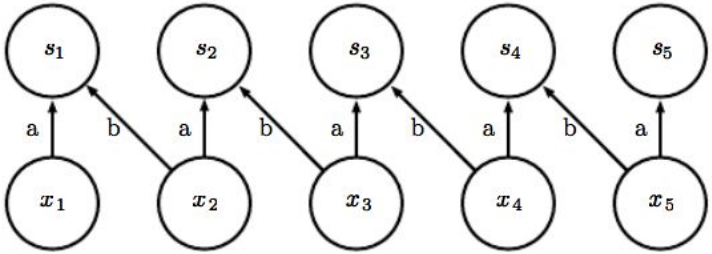
9.4 Convolution and Pooling as an Infinitely Strong Prior

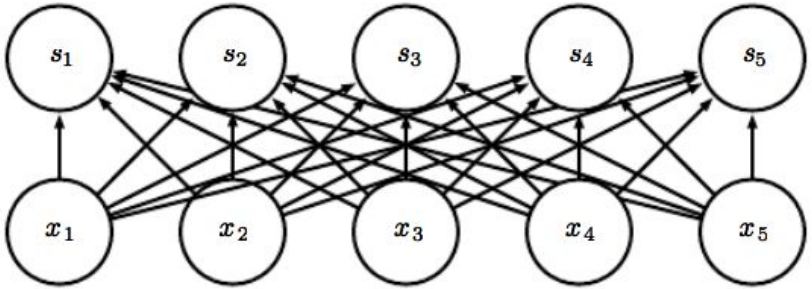
The use of pooling can be viewed as adding **infinitely strong prior** over the function that the layer learns

Weak Prior : putting a prior distribution that has high entropy, like a gaussian with high variance.
⇒ Parameters values can change more freely
Strong Prior: low entropy distribution such as Gaussian with low variance.
⇒ places zero probability on some parameters, making them forbidden

Convolutional network ⇔ a fully connected network with infinite strong prior over its weights, where

- Weights for a hidden units must be the same as weights of neighbors but shifted
- Weights must be zero except in the spatially contiguous receptive field.





9.4 Convolution and Pooling as an Infinitely Strong Prior

1) Problem **underfitting** when using convolution and pooling with those prior assumptions don't reflect reality.

Ex: a task which relies on preserving precise spatial information.

One can use pooling on some channels and not other in order to reduce underfitting

If a local input needs information from a distant location, the prior may not be appropriate.

2) Compare Conv network only to Conv networks

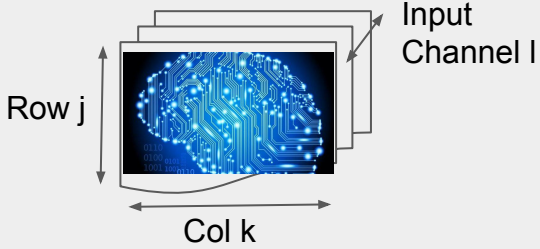
While permuting pixel would make convolutional networks fails, if could still work with other models.

9.5 Variant of the Basic Convolution Function

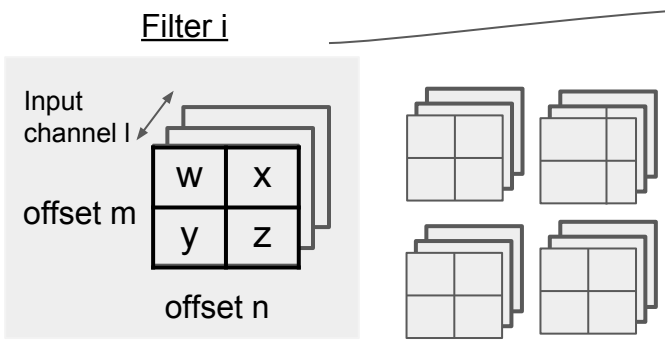
1) Multiple Channels, Multiple Kernels, stride =1

Each pixel has 3 values ⇒ 3 channels
We want to capture as many features as possible: # of Kernels

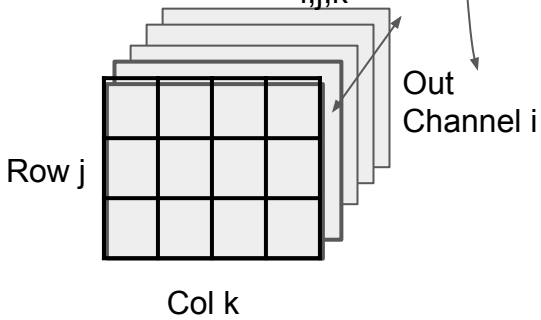
Input $V_{l,j,k}$



4-D Kernel $K_{i,l,m,n}$



Output $Z_{i,j,k}$



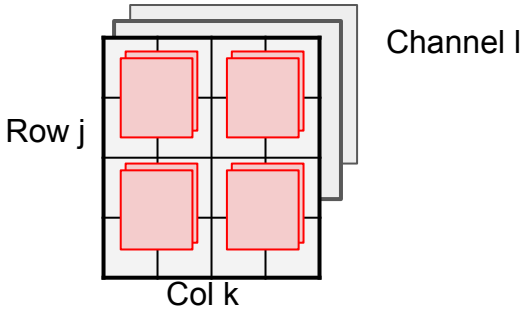
$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

2) **Stride = s, skip some positions of the kernel** $Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right]$$

Input $V_{l,j,k}$

Ex: stride s = 2



Stride can leads to downsampling.

Zero Padding allows us to control the size of the output. It's a hyperparameter

3 types:

1) Valid
2) Same
3) Full

Some animations here: https://github.com/vdumoulin/conv_arithmetic

Stride can leads to downsampling. Different Types

1) VALID Padding

Kernel only fits the image shape. Final width = m - k + 1

Input 4x4

output 3x3

k=2

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

M = 4

Stride can leads to downsampling. Different Types

1) **SAME Padding**:

Zero pad the input as much as needed in order to maintain the same shape

Input (4 x 4)

Output (4 x 4)

k=3

0	0	0	0	0	0
0	1	2	3	4	0
0	5	6	7	8	0
0	9	10	11	12	0
0	13	14	15	16	0
0	0	0	0	0	0

Stride can leads to downsampling. Different Types

1) **FULL Padding**:

Zero pad the input such that every pixel is visited k times.

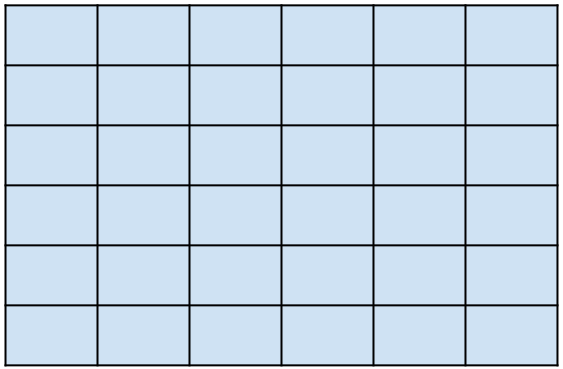
Final width = m + k + 1

Input (4 x 4)

Output (6 x 6)

k=3

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	2	3	4	0	0
0	0	5	6	7	8	0	0
0	0	9	10	11	12	0	0
0	0	13	14	15	16	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Other kinds of Connection:

1) **Locally Connected Layers** / Unshared Convolution

- Connection graph the same as in standard convolution
- Different parameters are used for each of these connections.
- Useful if we believe that each feature should be small part of a space, that is not found in other part of the image (ex: mouth or eyes in a face)

$$Z_{i,j,k} = \sum_{l,m,n} \left[V_{l,j+m-1,k+n-1} w_{i,j,k,l,m,n} \right].$$ 6D kernel

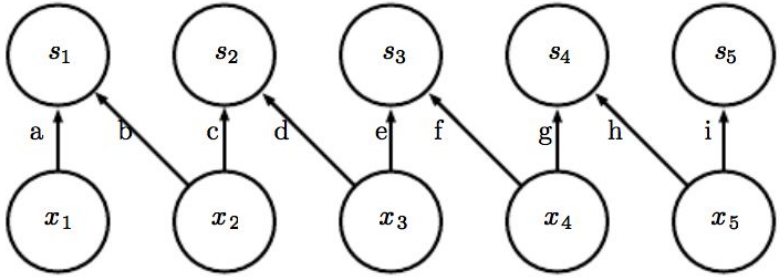
2) **Tiled Convolution**

- A compromise between Standard Conv. and Locally Connected Layer: Multiple kernels are used and repeated to fit the input grid.
- Immediate neighbors have kernels with different parameters
- Partial Parameter sharing, Lower memory requirement

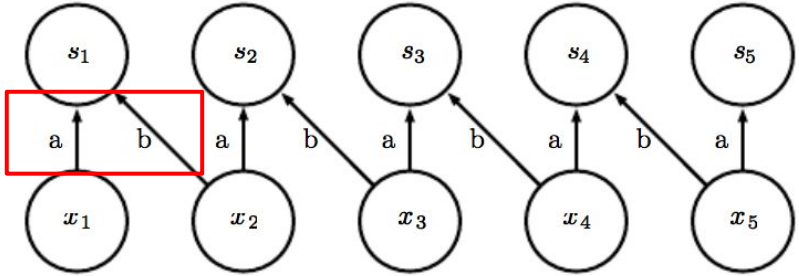
$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n,j\%t+1,k\%t+1}$$

9.5 Variant of the Basic Convolution Function

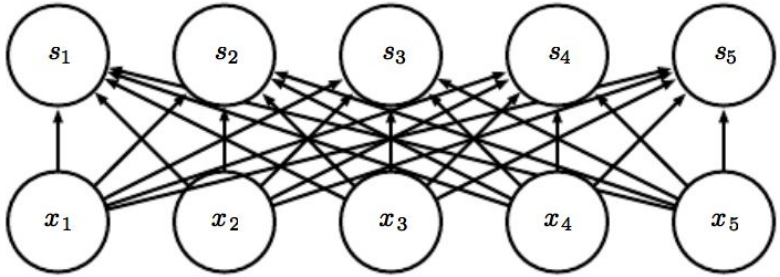
Locally-Connected



Standard Convolution k=2

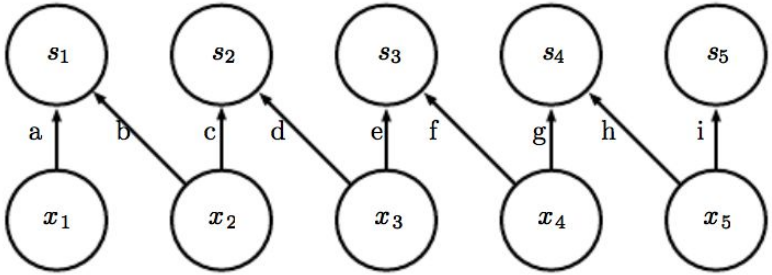


Fully-Connected

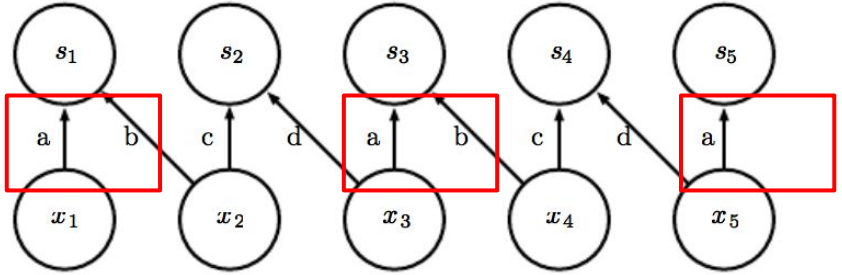


9.5 Variant of the Basic Convolution Function

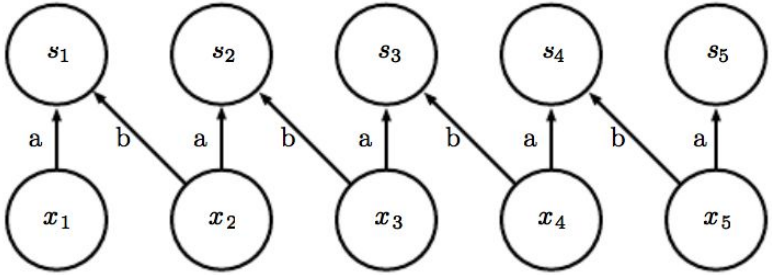
Locally-Connected



Tiled Convolution
2 kernels (a,b) and (c,d)



Traditional Convolution



Training

The convolution matrix:

- Convolutional as a linear operation, can be described by a **matrix** multiplication
- This matrix will be sparse and each elements of the kernel will be copied to several location

The transpose of the above matrix is needed to back-propagate the errors

Example: Training a strided convolution with kernel K, over multichannel input V, using stride s.

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right]$$

Loss function = J(V,K). During backprop, we receive a tensor **G** a the output of layer:

$$G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(\mathbf{V}, \mathbf{K})$$

Need to compute derivatives with respect to **weights** as well as **input**

$$G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(\mathbf{V}, \mathbf{K})$$

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right]$$

Derivatives with respect to Weights

$$g(\mathbf{G}, \mathbf{V}, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(\mathbf{V}, \mathbf{K}) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l}.$$

Derivatives with respect to Inputs

$$h(\mathbf{K}, \mathbf{G}, s)_{i,j,k} = \frac{\partial}{\partial V_{i,j,k}} J(\mathbf{V}, \mathbf{K})$$

$$= \sum_{\substack{l,m \\ \text{s.t.} \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ \text{s.t.} \\ (n-1)\times s+p=k}} \sum_{q} K_{q,i,m,p} G_{q,l,n}.$$

9.5 Variant of the Basic Convolution Function

Biases:

- **Locally connected layer**: each unit has its own bias
- **Tiled convolution**: share bias with the same tiling pattern
- **Standard Convolution**: One bias per channel

Goodfellow derivation details:
http://www.iro.umontreal.ca/~lisa/pointeurs/convolution.pdf

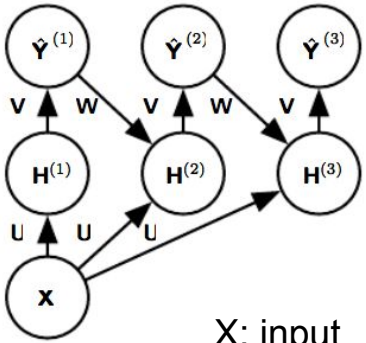
9.6 Structured Output

ConvNet can emit a multidimensional Tensor
Ex: $S_{i,j,k}$: the probability that pixel (j,k) belongs to class i

How?
1) Avoid pooling (Jain et Al 2007)
2) Emit a lower-resolution grid of labels (Pinheiro, Collobert 2014,2015)
3) Use a pooling operator with unit stride

Pixel wise labeling:
1) Produce initial guess
2) Refine the guess with interactions between neighboring pixels

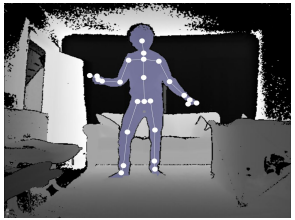
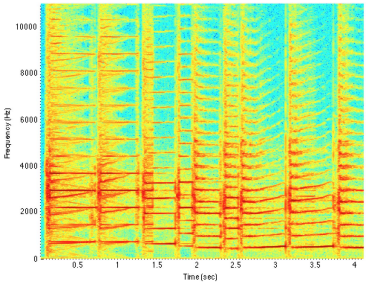
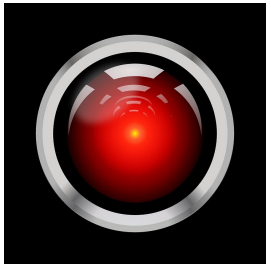
Like a Recurrent Neural Network



X: input

Y = output tensor labeling pixels, used as input iteratively
U,V,W are all conv kernels

9.7 Data Types

	Single Channel	MultiChannel
1D	Audio 	 Skeleton Animation Time t, angles
2D	Audio in freq spectrum 	 Color Image data
3D	Volumetric Data - CT Scan 	 Color Video Data

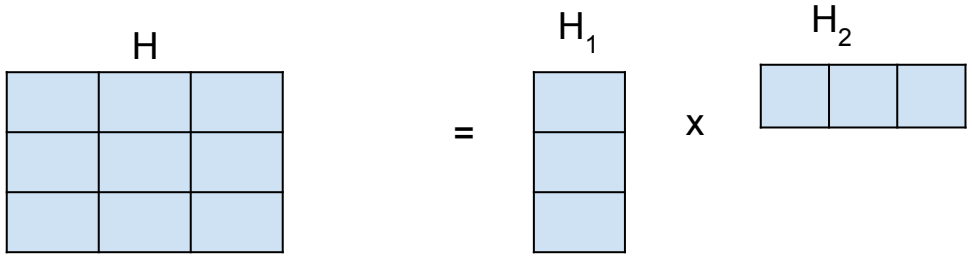
9.8 Efficient Convolution Algorithms

1) Convolution is ⇔ point wise multiplication of the input + kernel in the Frequency domain, followed by converting the result back to the time domain with inverse Fourier Transform. Can be faster than naive implementation for some problem sizes

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

From wikipedia: “With the help of the convolution theorem and the fast Fourier transform, the complexity of the convolution can be reduced from $O(n^2)$ to $O(n \log n)$.”

2) **Separable kernel:** If a d-dimensional kernel can be written as product of d kernels



Associative property $I * H = I * (H_1 * H_2) = (I * H_1) * H_2$

faster

Standard Conv ⇔ Compose **d** 1-dimensional convolutions with each of the vectors

Runtime and Storage space = $O(w^d) \rightarrow O(w \cdot d)$

9.9 Random or Unduspervised Features

Standard Convolution requires a complete run of forward and backprop on each gradient step.
How to reduce the cost of convolutional network training?
→ Use features that are not trained in a supervised fashion

3 strategies:

1) Initialize kernel parameters at random
2) Design kernel parameters by hand.
3) Learn kernels with an unsupervised criterion (ex: Coates et al 2011, apply k-mean do small image patches and used learned centroid as kernel)

On the random filter approach:

1) Select multiple randomly initialised network and train only higher layers.
2) Pick the best and then train the entire network

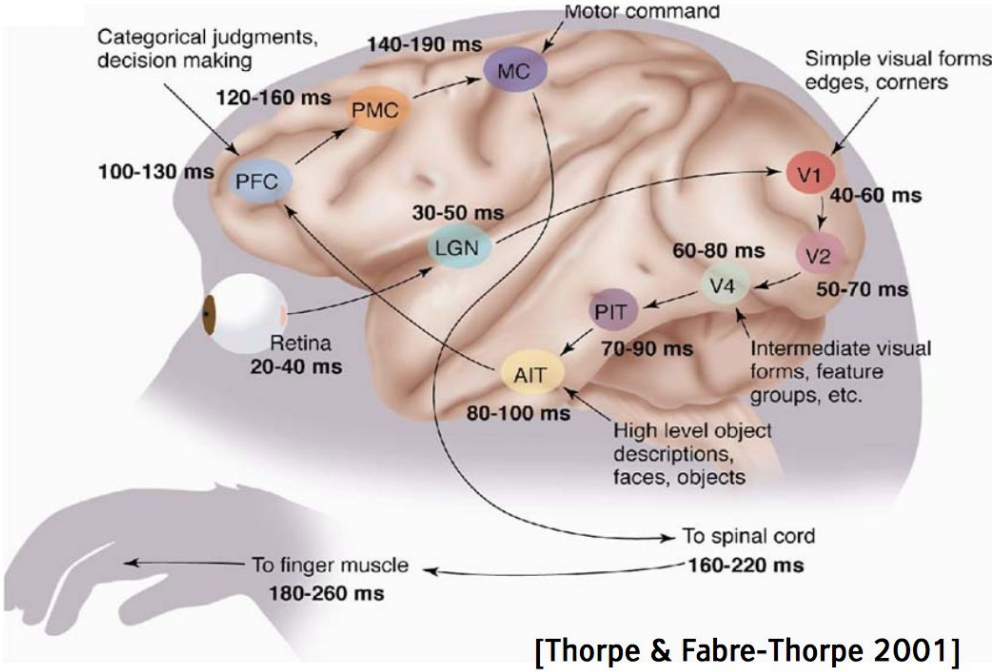
Other methods that don't require full forward and back-prop
Greedy Layer-wise pretraining: See convolutional deep Belief Network (Lee et al 2009)

9.10 The Neuroscientific Basis for Convolutional Networks

What are the scientific discoveries that served as basic design principle for convnets?

Hubel/Wisel Experiments: Neurons in early visual system are more strongly receptive to visual patterns such as precisely oriented bars.

Light travels accross following regions:
Retina → LGN → V1 → V4 → IT



9.10 The Neuroscientific Basis for Convolutional Networks

V1: Primary Visual Cortex

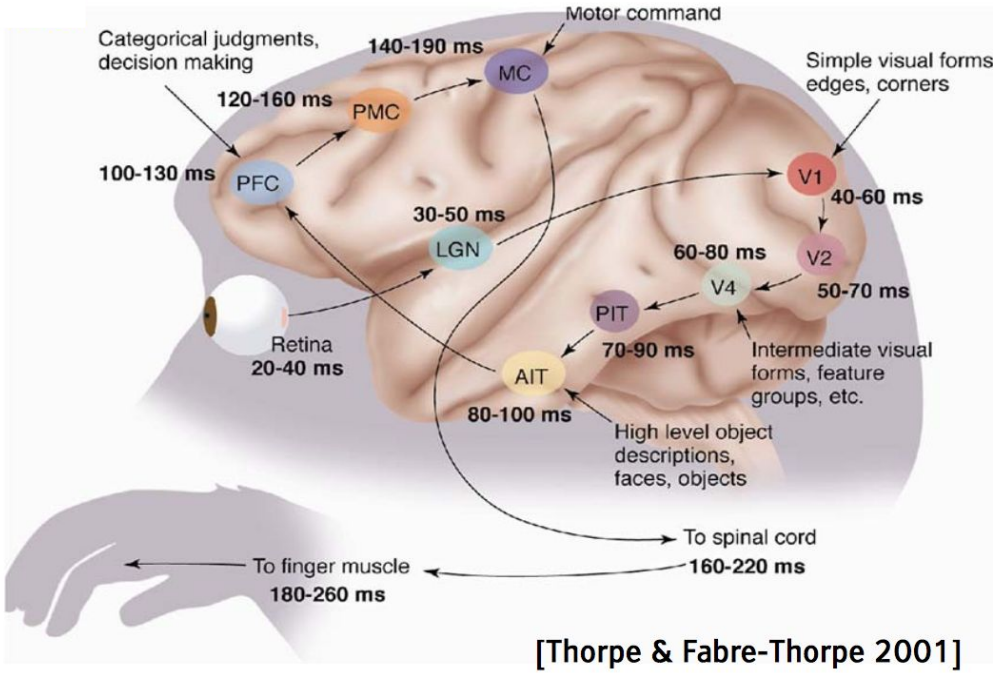
Cartoon view: Convnet captures 3 properties of V1

1) V1 cells arranged in a spatial map
2) Many **simple** cells, behave as detector units
3) Many **Complex** Cells, behave as pooling cells

Simple cells: activity ⇔ simple linear function

Complex cells:

- invariance to small shift in the position of the features
- Invariant to changes in lighting that cannot be captured by pooling (max-out)

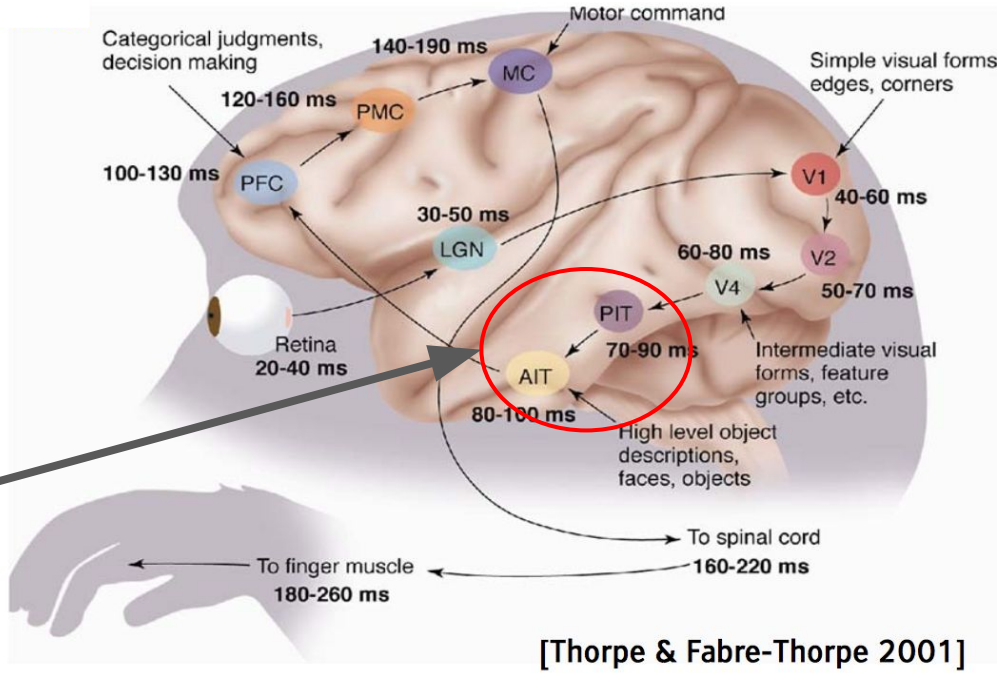


9.10 The Neuroscientific Basis for Convolutional Networks

Same concepts apply in higher layers, and eventually we reach cells that capture whole concepts: **Grandmother Cells**
= Cells that are invariant to concept, such as grandmother, however the person is represented in a picture.

Do these cells really exist?

It seems they have been identified in the **medial temporal lobe**. (Quiroga 2005)
Ex: Halle Berry Neuron, Bill Clinton Neuron, Jennifer Aniston neuron etc..



http://suns.mit.edu/articles/Nature.pdf

9.10 The Neuroscientific Basis for Convolutional Networks

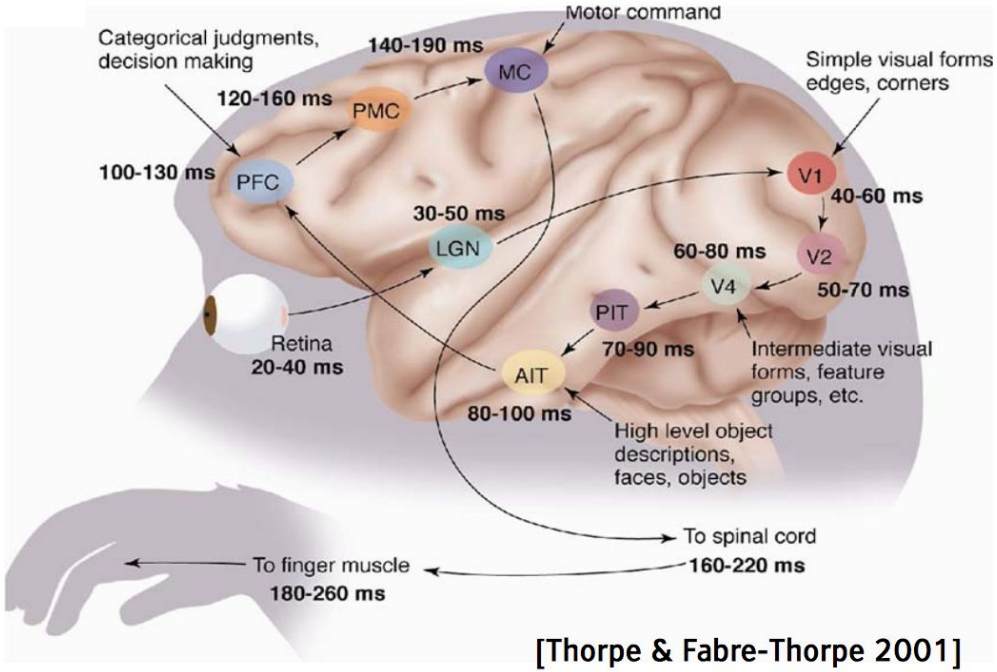
IT: Inferotemporal Cortex

The closest analog to a convolutional network's last layer of features

100ms: Time it take for Information to go from retina to IT. If gazing longer on an object, brain uses feedback downwards to update the activations.

It has been shown that ConvNets can predict the firing rate of IT region and perform similarly to time-limited human on object recognition task. (DiCarlo 2013)

http://media.nips.cc/Conferences/2013/Video/Tutorial3A.pdf



9.10 The Neuroscientific Basis for Convolutional Networks

Differences between ConvNet and Mammalian Visual System:

1) Human eye is very low resolution.
    a) **Fovea**: an area of the size of thumbnail at arm's length
    b) Illusion of seeing high-def, actually brain stitches together glimpses from several areas.
    c) Saccades: Eye movement that provide attention mechanism.
2) Human visual system integrated with other systems: hearing, smelling, mood, thoughts.
3) Visual system recognizes entire scenes, not just objects.
4) Low Level V1 layer have feedback from higher layers.
5) Not clear how intermediate layer between V1 and IT work. Is simple/complex cell distinction illusory, is it a continuum?

Neuroscience doesn't tell us much about how to **train** a neural network.
Early network such as **Neocognitron** already had the concept of shared parameters but did not use backpropagation, instead relied on layer-wise unsupervised clustering.

Whence backprop? Introduced by Lang/Hinton 1988 to train a Time Delay Neural Networks(TDNN) applied to times series.
Lecun in 1989 developed convolutional networks with backprop on 2D images

9.10 The Neuroscientific Basis for Convolutional Networks

What exactly do the **simple** neural cells detect?

In a ANN, we can look at the kernel of the first layer as images to see what is learned.

In biological neural network, used electrodes to detect spikes, and fit a linear model: **Reverse Correlation**.

⇒ What do we learn? Most V1 weights can be described by **Gabor Functions**. A function that described the weights of a 2D point in the image.

Ex: if I(x,y): pixel intensity at position (x,y), and w(x,y) the weights, then the response of the a cell to an image is given by

$$s(I) = \sum_{x \in \mathbb{X}} \sum_{y \in \mathbb{Y}} w(x,y) I(x,y).$$

$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp\left(-\beta_x x'^2 - \beta_y y'^2\right) \cos(f x' + \phi),$$

$$x' = (x - x_0)\cos(\tau) + (y - y_0)\sin(\tau)$$

$$y' = -(x - x_0)\sin(\tau) + (y - y_0)\cos(\tau).$$

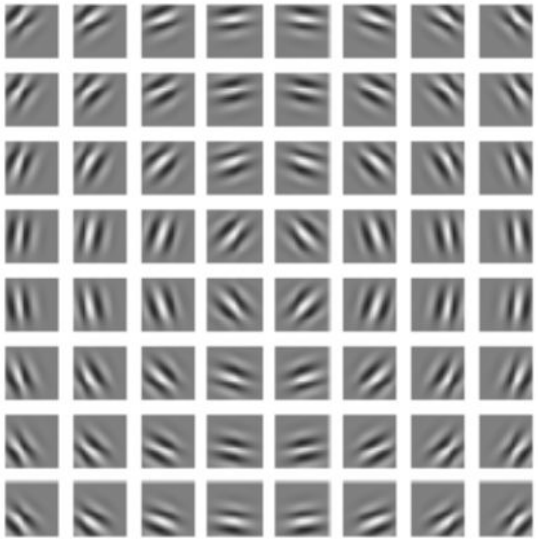
x_0, y_0 , τ : a coordinate system.

- The cell will respond to images features entered at the point (x_0, y_0)
- Cell will respond to changes in brightness as we move along a line rotate τ radians from the horizontal

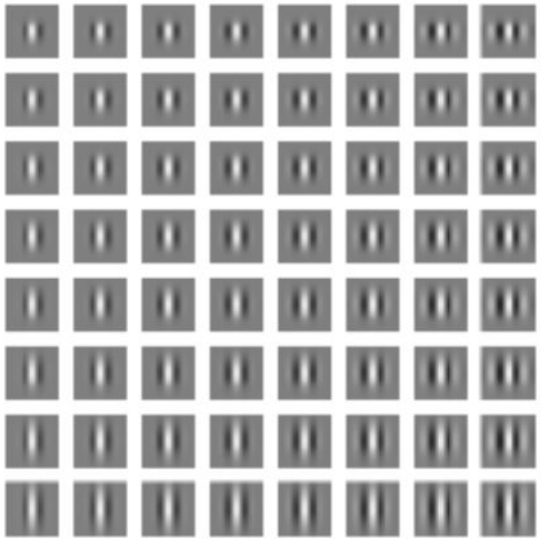
9.10 The Neuroscientific Basis for Convolutional Networks

$$w(x, y; \alpha, \beta_x, \beta_y, f, \phi, x_0, y_0, \tau) = \alpha \exp\left(-\beta_x x'^2 - \beta_y y'^2\right) \cos(f x' + \phi),$$

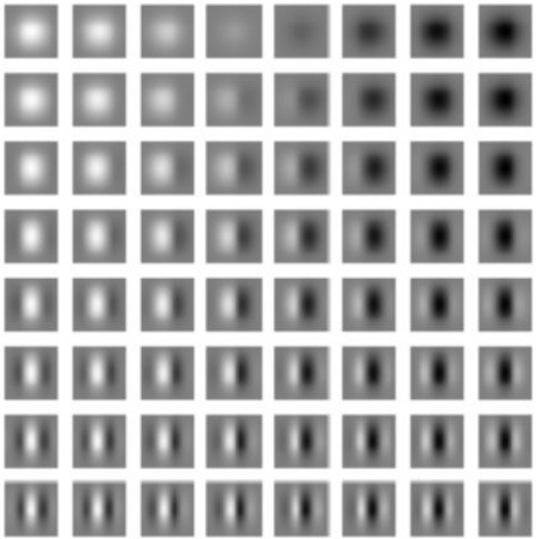
White = large positive values



Different values of x_0, y_0 , τ



Different Gaussian scale parameter β_x and β_y (decreasing left to right, top to bottom



Different Sinusoidal parameter f and φ. (f increase top to bottom, φ left to right)

Many algorithm learn Gabor like weights: ex: Sparse Coding (Olshauser 1996), Convnets

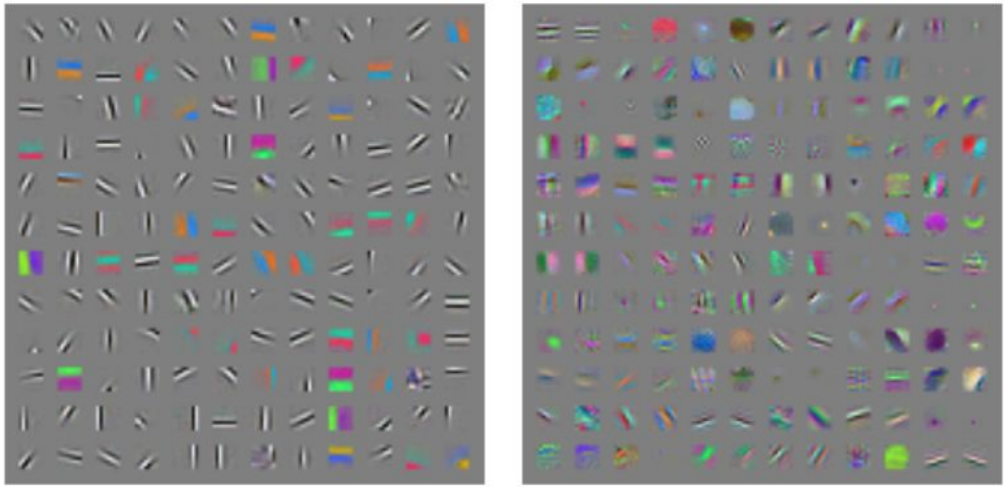


Figure 9.19: Many machine learning algorithms learn features that detect edges or specific colors of edges when applied to natural images. These feature detectors are reminiscent of the Gabor functions known to be present in primary visual cortex. *(Left)*Weights learned by an unsupervised learning algorithm (spike and slab sparse coding) applied to small image patches. *(Right)*Convolution kernels learned by the first layer of a fully supervised convolutional maxout network. Neighboring pairs of filters drive the same maxout unit.

How about **complex** cells?

Cartoon version is that it computes the L2 norm of two simple cells

$$c(I) = \sqrt{s_0(I)^2 + s_1(I)^2}$$

9.11 Convolutional Networks and the History of Deep Learning

Long history of Convolutional neural networks.

1) 1990s AT&T convNet to read checks (Lecun 1998b)
2) End of 90s: NEC system read 10% of US checks
3) Microsoft OCR handwriting recognition system (Simard 2003)

ConvNEt in competitions: ImageNET 2012 won by ConvNet developed by Krizhevsky et al. 2012.

Why were ConvNets successful early on? Not sure, Perhaps less computationally expensive than fully connected networks.

Currently DL methods enabled by modern hardware and test shows they perform well even if using technology available early in the 1990s. Psychological barrier to their use?