Convolutional Networks Overview

Sargur Srihari

Topics

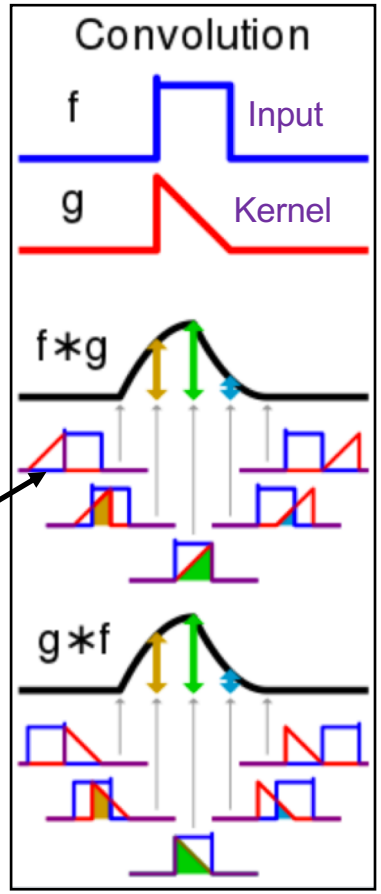
- The Convolution operation
- Neural nets and matrix multiplication
- Limitations of Simple Neural Networks
- Convolutional Networks
- Pooling
- Convolutional Network Architecture
- Backpropagation in a CNN
- Advantages of CNN architectures

Convolution: 1-D continuous case

- One-dimensional continuous case
  - Input $f(t)$ is convolved with a kernel $g(t)$

$$(f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau) g(t-\tau)\, d\tau$$

Note that $(f * g)(t) = (g * f)(t)$

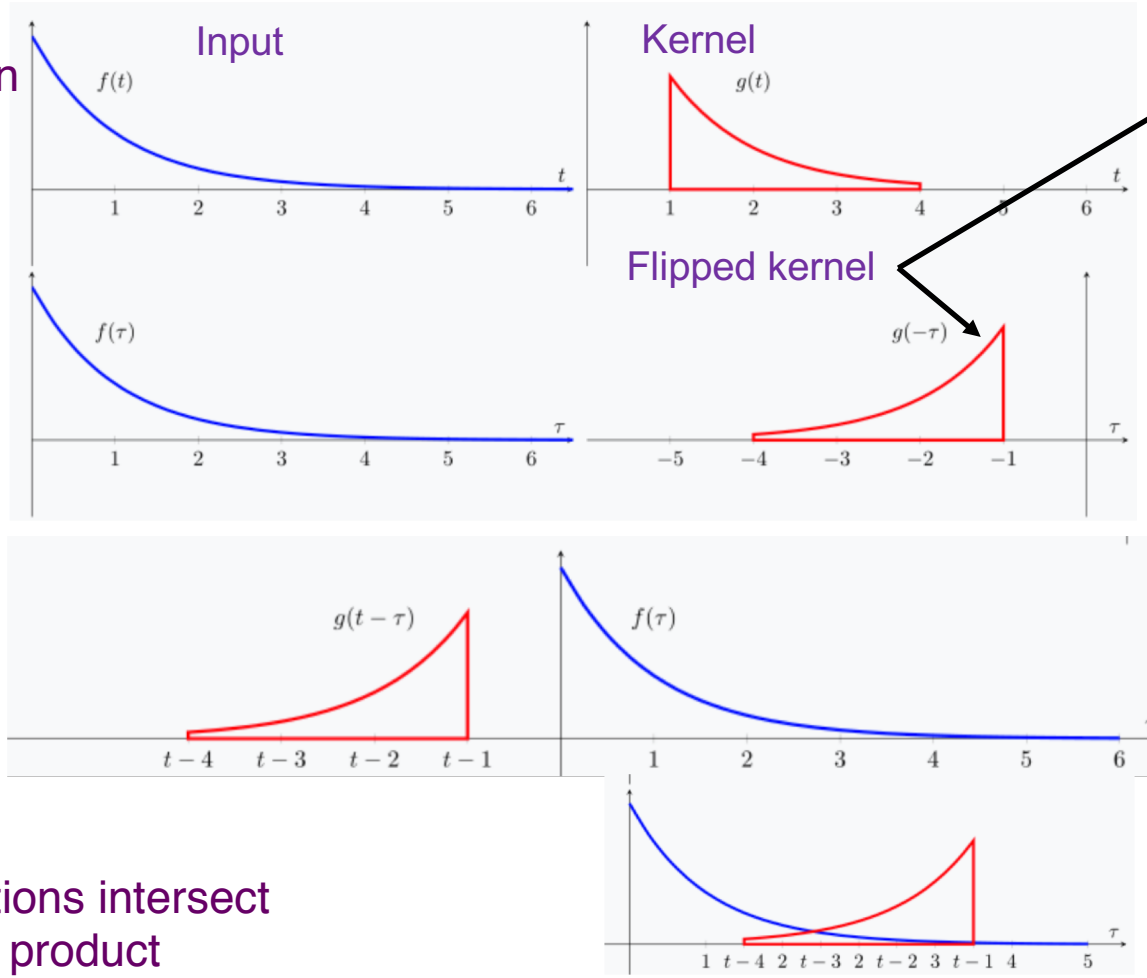


1. Express each function in terms of a dummy variable τ

2. Reflect one of the functions $g(\tau) \rightarrow g(-\tau)$

3. Add a time offset t, which allows $g(t-\tau)$ to slide along the τ axis

4. Start t at $-\infty$ and slide it all the way to $+\infty$
Wherever the two functions intersect find the integral of their product

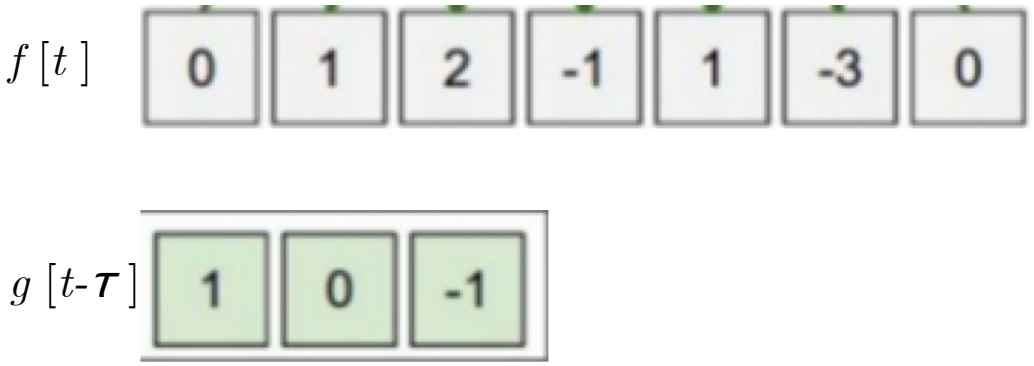


https://en.wikipedia.org

Convolution: 1-D discrete case

- Here we have discrete functions f and g

$$(f * g)[t] = \sum_{\tau=-\infty}^{\infty} f[\tau] \cdot g[t-\tau]$$



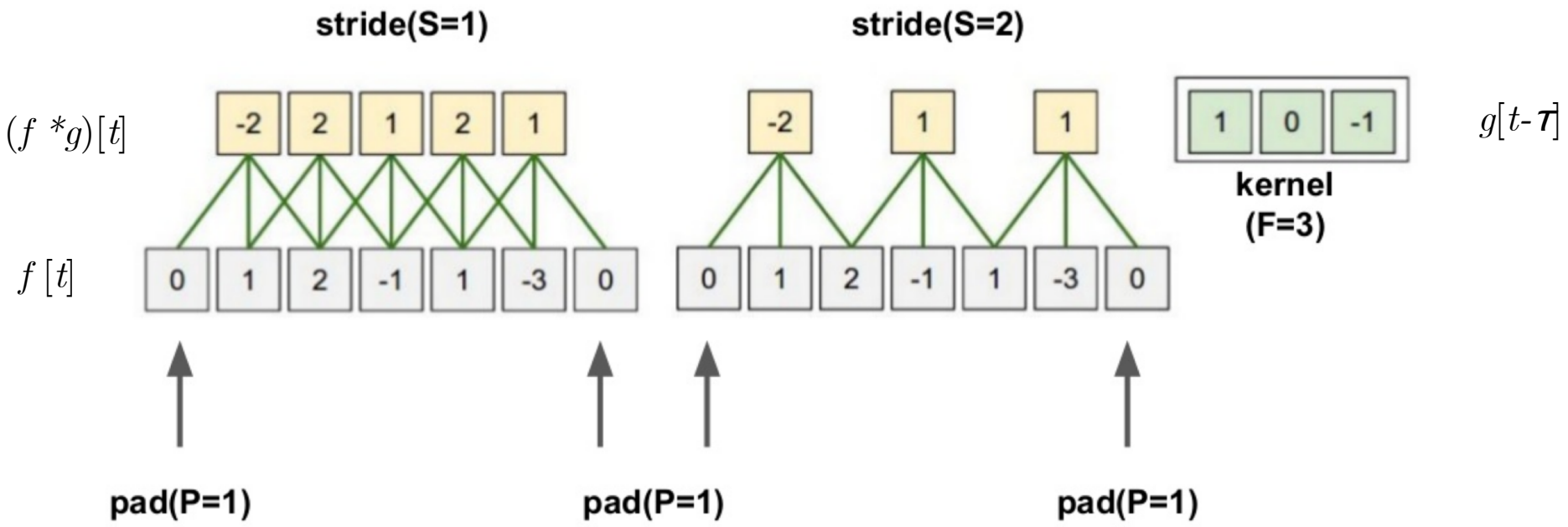
Computation of 1-D discrete convolution

Parameters of convolution:

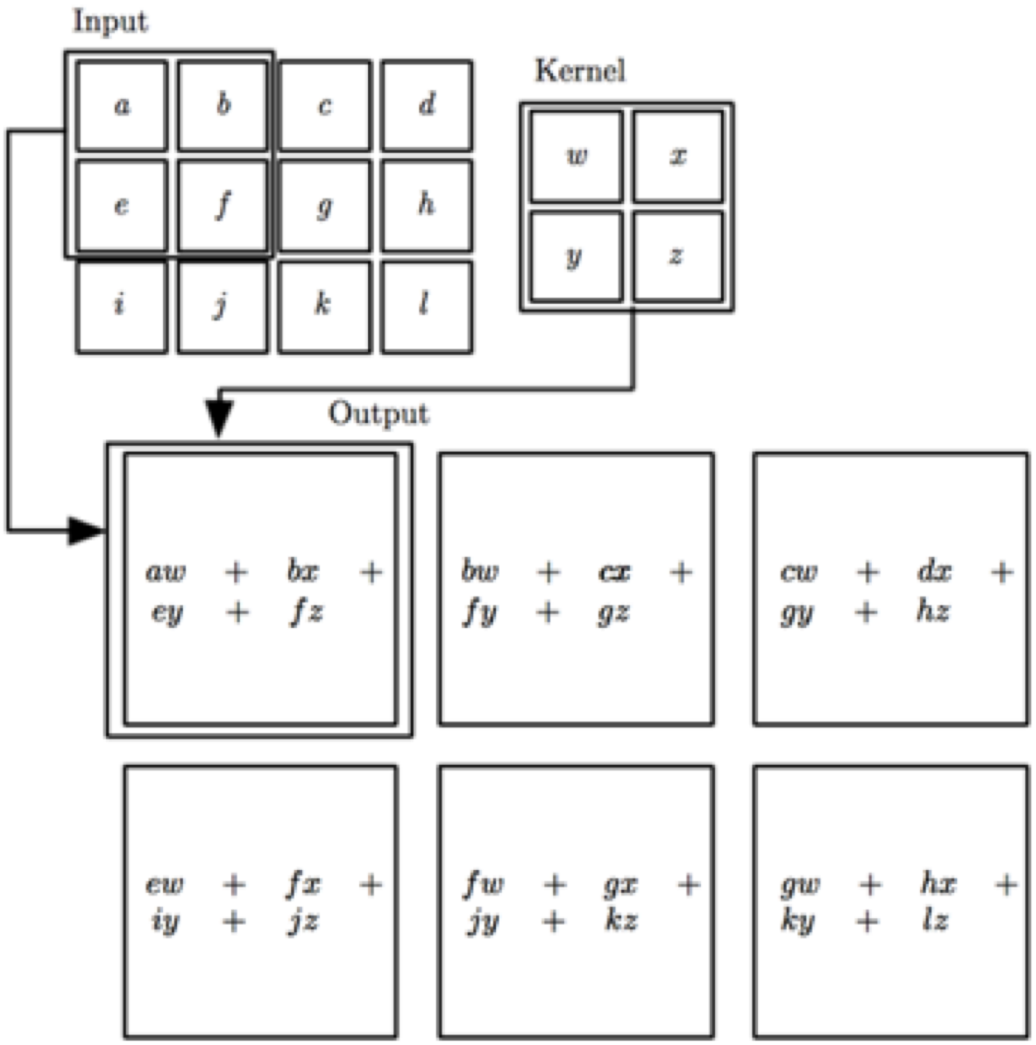
Kernel size (F)

Padding (P)

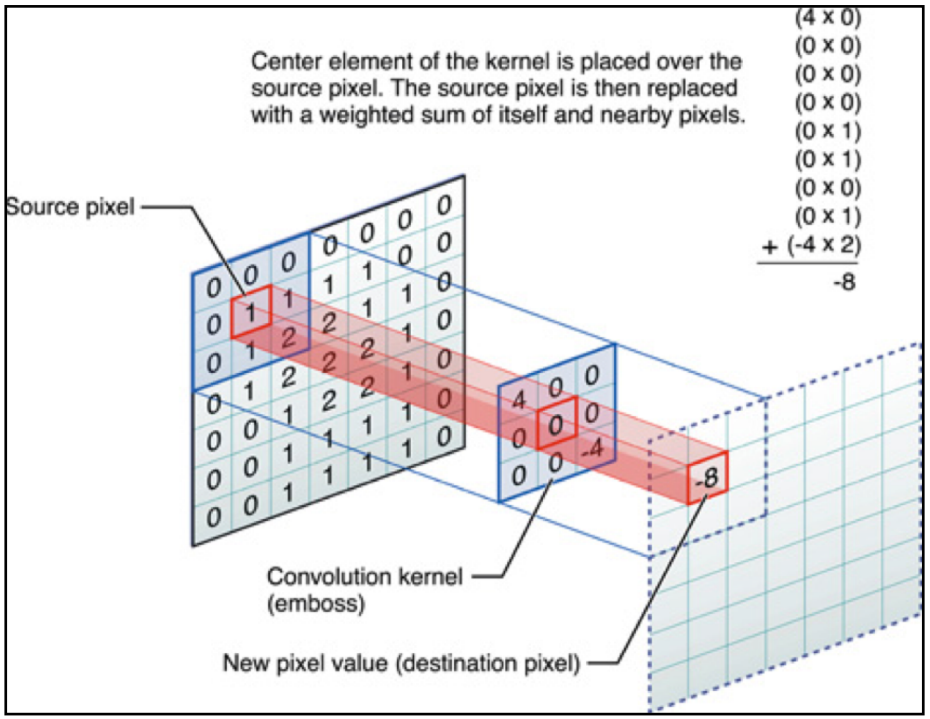
Stride (S)



2-D discrete convolution



Uses of 2-D Image Convolution



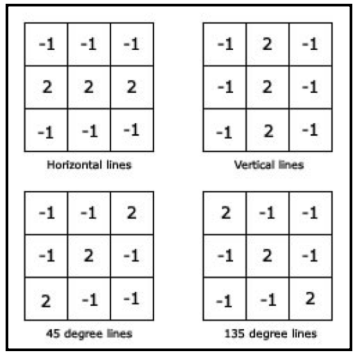
Kernel for blurring
Neighborhood average

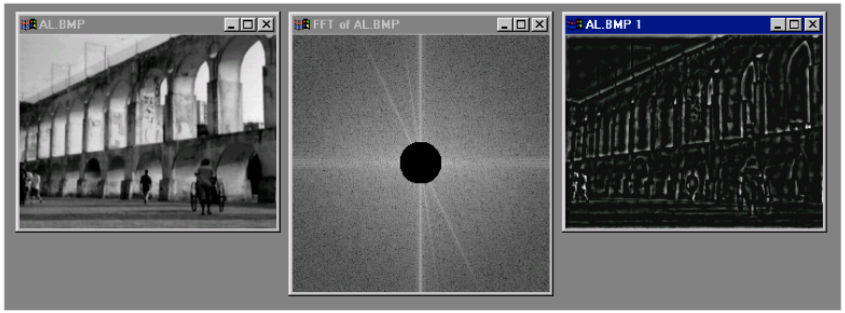
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

Kernel for edge detection
Neighborhood difference

	0	1	0	
	1	-4	1	
	0	1	0	

Kernels for line detection

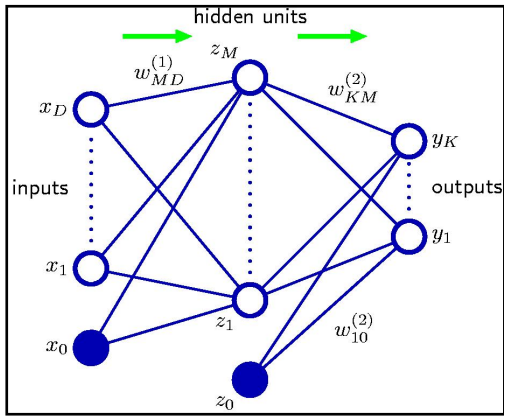




Neural net performs matrix multiplications

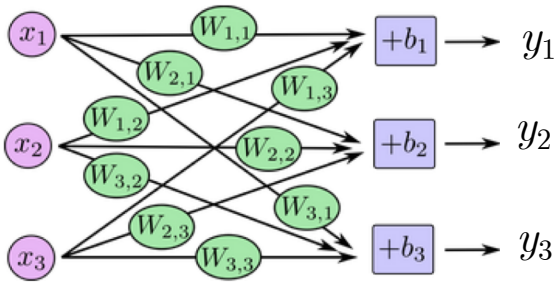
- Each layer produces values that are obtained from previous layer by performing a matrix-multiplication
  - In an unaugmented network
    - Hidden layer produces values $\boldsymbol{z} = h(\boldsymbol{W}^{(1)\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}^{(1)})$
    - Output layer produces values $\boldsymbol{y} = \sigma(\boldsymbol{W}^{(2)\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}^{(2)})$
  - Note: $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$ are matrices rather than vectors
    - Example with $D=3,\ M=3 \quad \boldsymbol{x}=[x_1,x_2,x_3]^{\mathrm{T}}$
    - We have two weight matrices $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$

Augmented network



$$\boldsymbol{w} = \left\{ \begin{array}{l} \boldsymbol{W}^{(1)}\text{columns}: \boldsymbol{W}^{(1)}_1 = \left[W^{(1)}_{11}W^{(1)}_{12}W^{(1)}_{13}\right]^T, \boldsymbol{W}^{(1)}_2 = \left[W^{(1)}_{21}W^{(1)}_{22}W^{(1)}_{23}\right]^T, \boldsymbol{W}^{(1)}_3 = \left[W^{(1)}_{31}W^{(1)}_{32}W^{(1)}_{33}\right]^T \\ \boldsymbol{W}^{(2)}\text{columns}: \boldsymbol{W}^{(2)}_1 = \left[W^{(2)}_{11}W^{(2)}_{12}W^{(2)}_{13}\right]^T, \boldsymbol{W}^{(2)}_2 = \left[W^{(2)}_{21}W^{(2)}_{22}W^{(2)}_{23}\right]^T, \boldsymbol{W}^{(2)}_3 = \left[W^{(2)}_{31}W^{(2)}_{32}W^{(2)}_{33}\right]^T \end{array} \right.$$

First Network layer



Network layer output

$$\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

In matrix multiplication notation

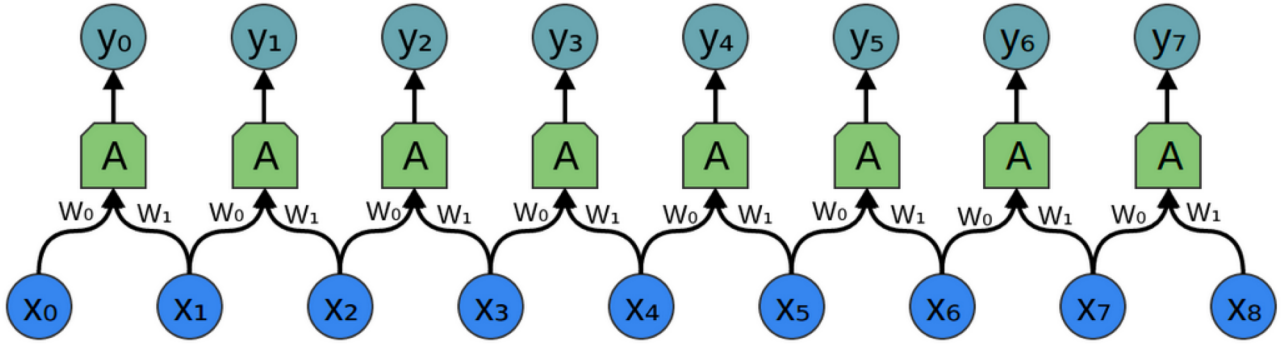
$$\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Limitations of Simple Neural Networks

- Need substantial number of training samples
- Slow learning (convergence times)
- Inadequate parameter selection techniques that lead to poor minima
- Network should exhibit invariance to translation, scaling and elastic deformations
  - A large training set can take care of this
- It ignores a key property of images
  - Nearby pixels are more strongly correlated than distant ones
  - Modern computer vision approaches exploit this property
- Information can be merged at later stages to get higher order features and about whole image

Neural network performing 1-D convolution

$f[t]$



Kernel $g(t)$:

$[\ldots 0, w_1, w_0, 0 \ldots]$

Equations for outputs of this network:

$$y_0 = \sigma(W_0 x_0 + W_1 x_1 - b)$$

$$y_1 = \sigma(W_0 x_1 + W_1 x_2 - b)$$

etc. upto y_7

We can also write the equations in terms of elements of a general 8×8 weight matrix W as:

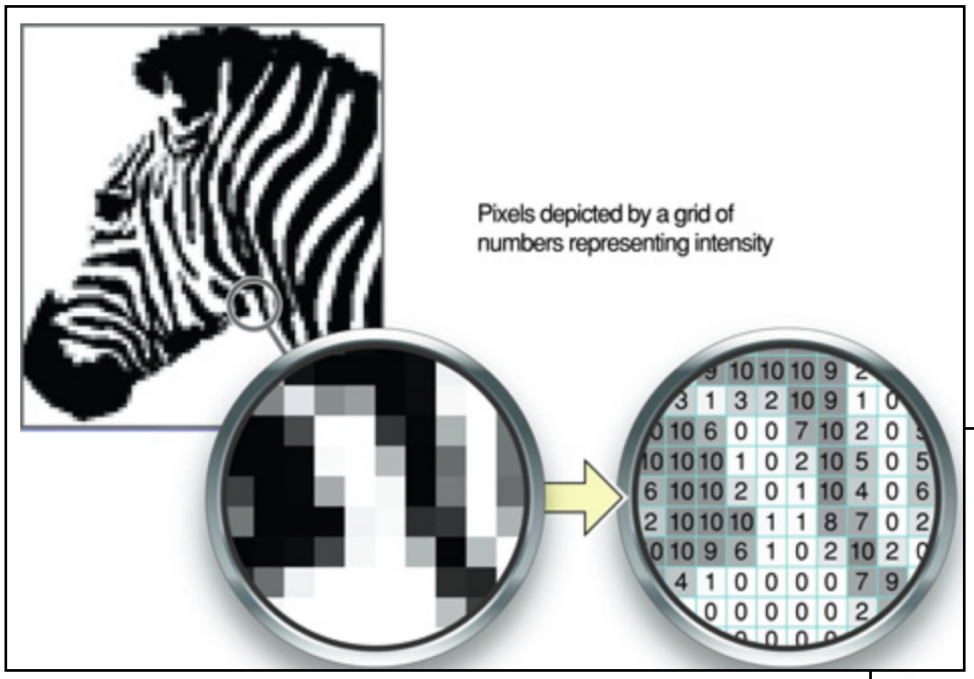
$$y_0 = \sigma(W_{0,0} x_0 + W_{0,1} x_1 + W_{0,2} x_2 \ldots)$$

$$y_1 = \sigma(W_{1,0} x_0 + W_{1,1} x_1 + W_{1,2} x_2 \ldots)$$

where $W = \begin{bmatrix} w_0 & w_1 & 0 & 0 & \ldots \\ 0 & w_0 & w_1 & 0 & \ldots \\ 0 & 0 & w_0 & w_1 & \ldots \\ 0 & 0 & 0 & w_0 & \ldots \\ \ldots & \ldots & \ldots & \ldots & \ldots \end{bmatrix}$

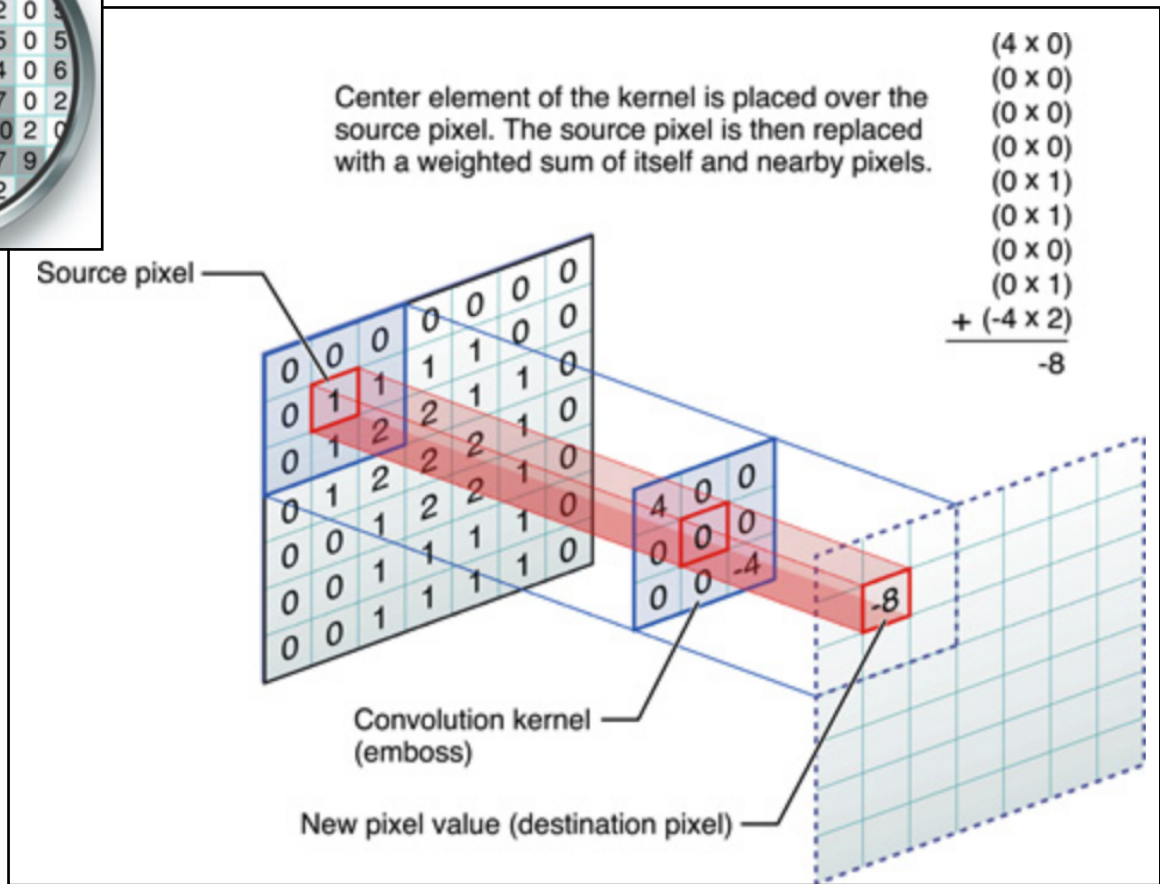
http://colah.github.io/posts/2014-07-Understanding-Convolutions/

Sparse connectivity due to Image Convolution



- Input image may have *millions* of pixels,
- But we can detect edges with kernels of *hundreds* of pixels
- If we limit no of connections for each input to k
  - we need $k \times n$ parameters and $O(k \times n)$ runtime
- It is possible to get good performance with $k << n$

- Convolutional networks have sparse interactions
- Accomplished by making the kernel smaller than the input
  - Next slide shows graphical depiction



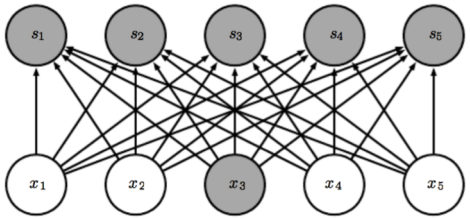
Three Mechanisms of Convolutional Neural Networks

1. Local Receptive Fields
2. Subsampling
3. Weight Sharing

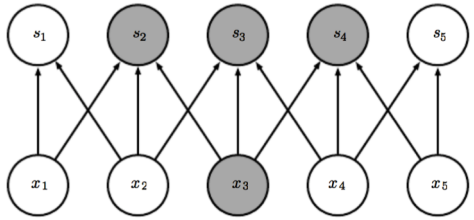
Traditional vs Convolutional Networks

- Traditional neural network layers use matrix multiplication by a matrix of parameters with a separate parameter describing the interaction between each input unit and each output unit

$$\boldsymbol{s} = g(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{x})$$

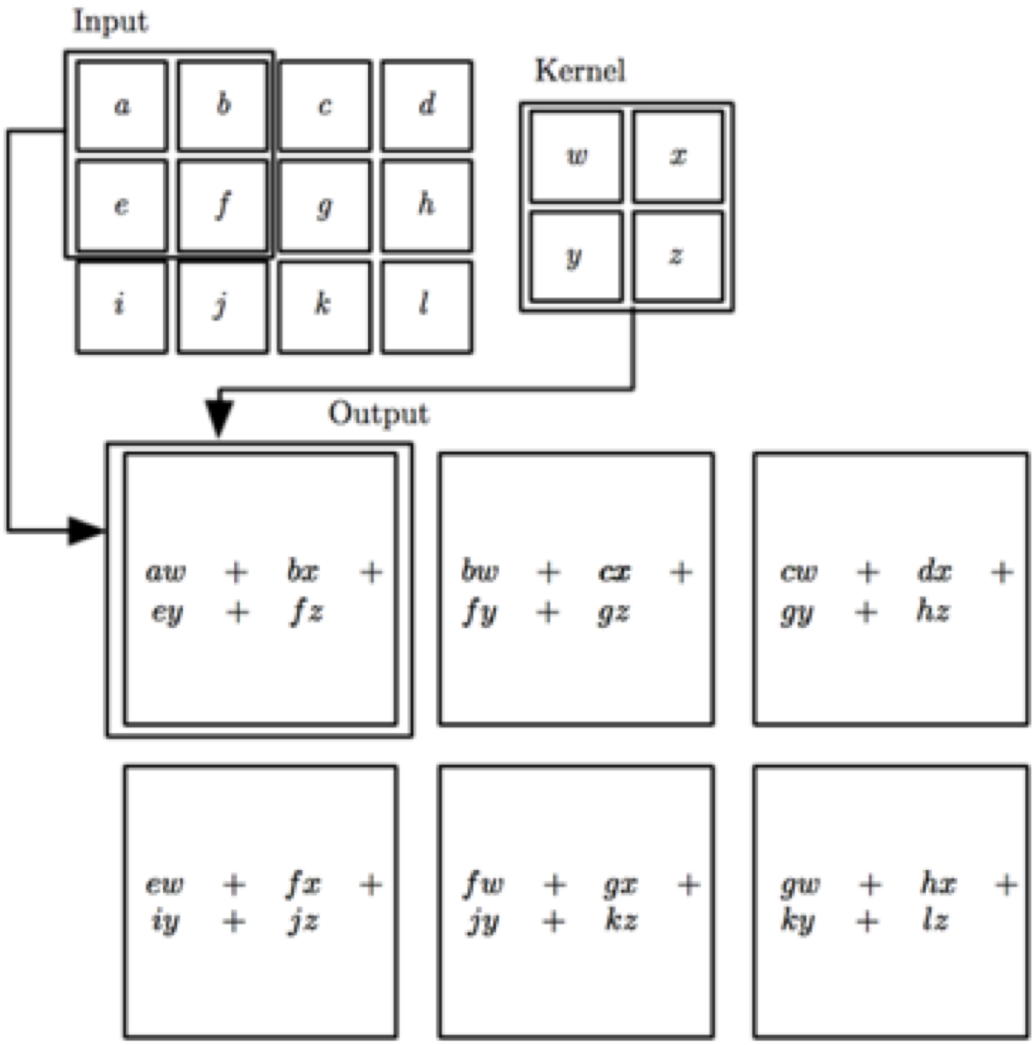


  - With m inputs and n outputs, matrix multiplication requires $m \times n$ parameters and $O(m \times n)$ runtime per example
  - This means every output unit interacts with every input unit
- Convolutional network layers have sparse interactions



  - If we limit no of connections for each input to k we need $k \times n$ parameters and $O(k \times n)$ runtime

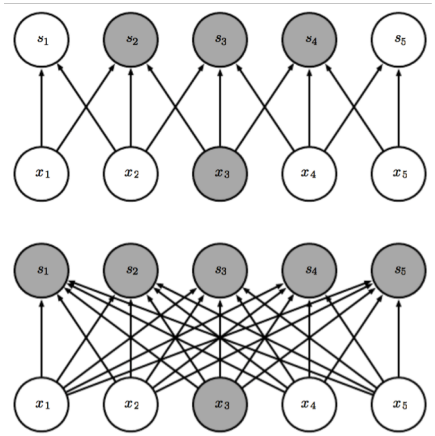
CNN matrix multiplication



Far fewer weights needed than full matrix multiplication

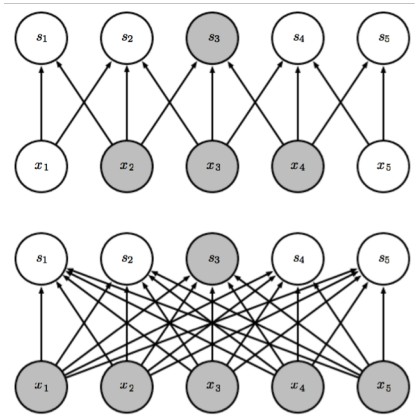
Views of sparsity of CNN vs full connectivity

Sparsity viewed from below



- Highlight one input x_3 and output units s affected by it
- *Top*: when s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3
- *Bottom:* when s is formed by matrix multiplication connectivity is no longer sparse
  - So all outputs are affected by x_3

Sparsity viewed from above



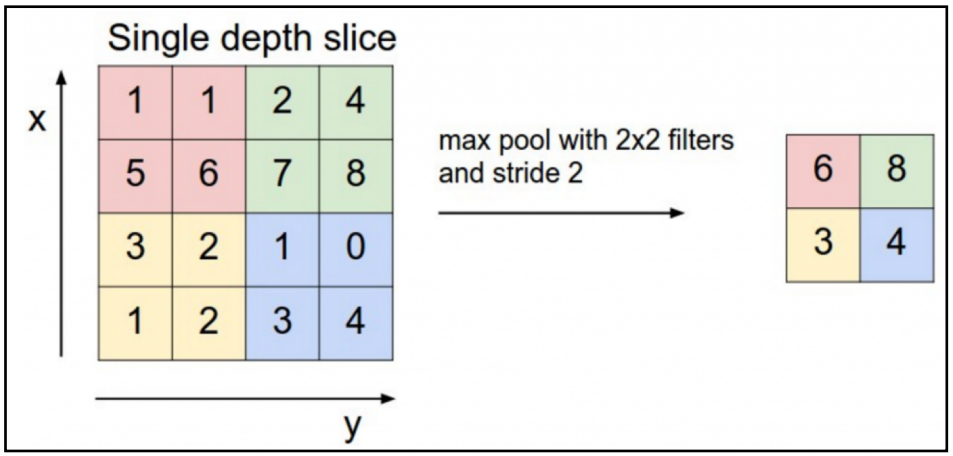
- Highlight one output s_3 and
- inputs x that affect this unit
  - These units are known as the receptive field of s_3

Pooling

- A key aspect of CNNs are *pooling layers*
- Typically applied after the convolutional layers.
- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby inputs
  - Pooling layers subsample their input
  - Example on next slide

Pooling functions

- Popular pooling functions are:
  1. *max pooling* operation reports the maximum output within a rectangular neighborhood



$6,8,3,4$ are the maximum values in each of the 2×2 regions of same color

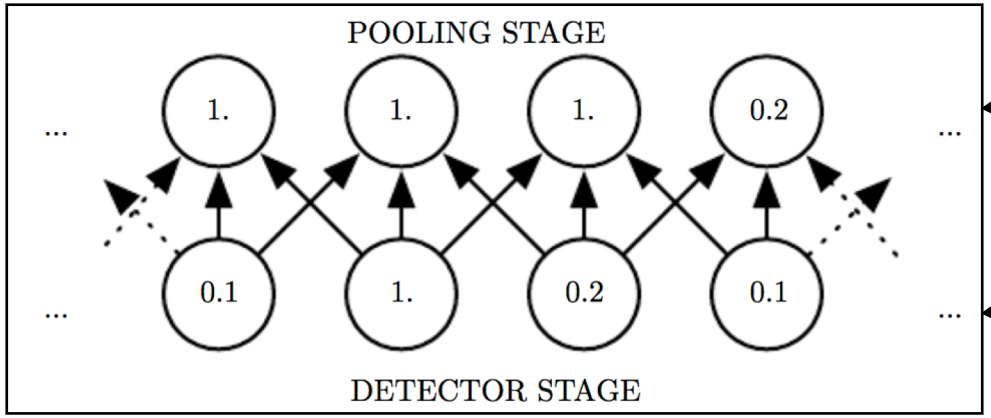
  2. Average of a rectangular neighborhood
  3. L^2 norm of a rectangular neighborhood
  4. Weighted average based on the distance from the central pixel

Why pooling?

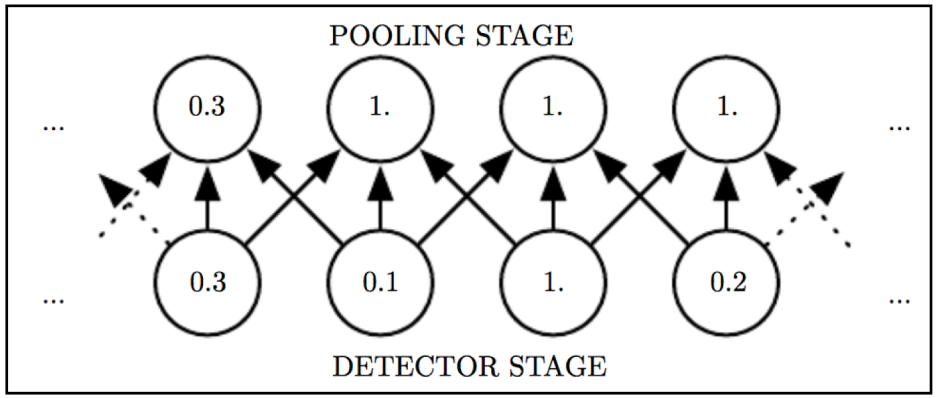
- It provides a fixed size output matrix, which typically is required for classification.
  - E.g., with $1{,}000$ filters and max pooling to each, we get a 1000-dimensional output, regardless of the size of filters, or size of input
    - This allows you to use variable size sentences, and variable size filters, but always get the same output dimensions to feed into a classifier
- Pooling also provides basic invariance to translating (shifting) and rotation
  - When pooling over a region, output will stay approximately the same even if you shift/rotate the image by a few pixels
    - because the max operations will pick out the same value regardless

Max pooling introduces invariance to translation

- View of middle of output of a convolutional layer

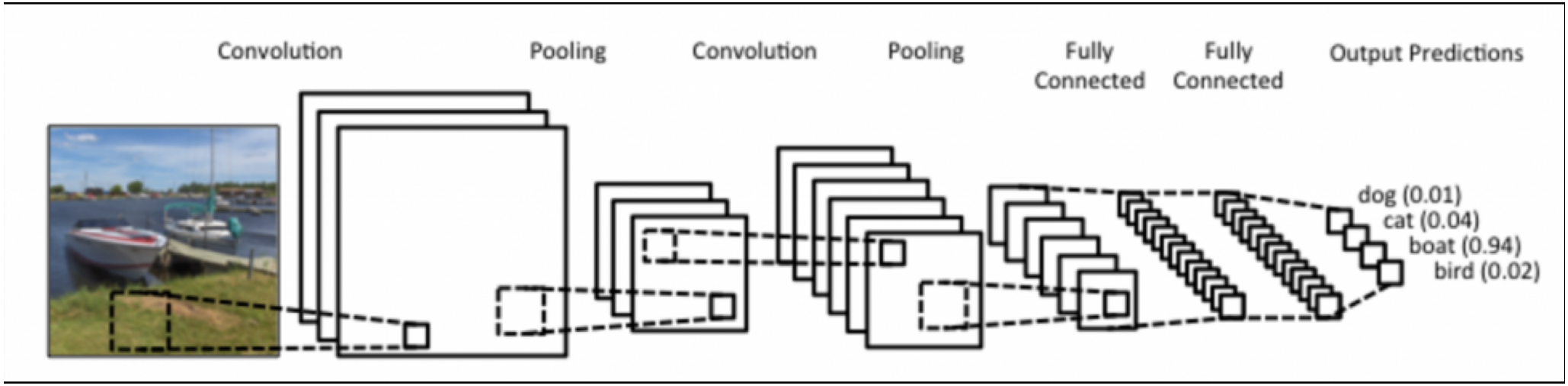


- Same network after the input has been shifted by one pixel



- Every input value has changed, but only half the values of output have changed because maxpooling units are only sensitive to maximum value in neighborhood not exact value

Convolutional Network Architecture



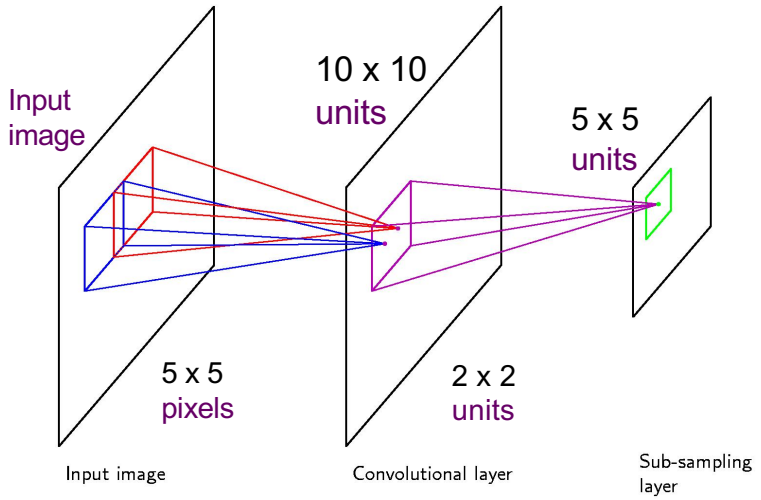
Three kernels

Pooling Reduces size

Six kernels

Convolution and Sub-sampling

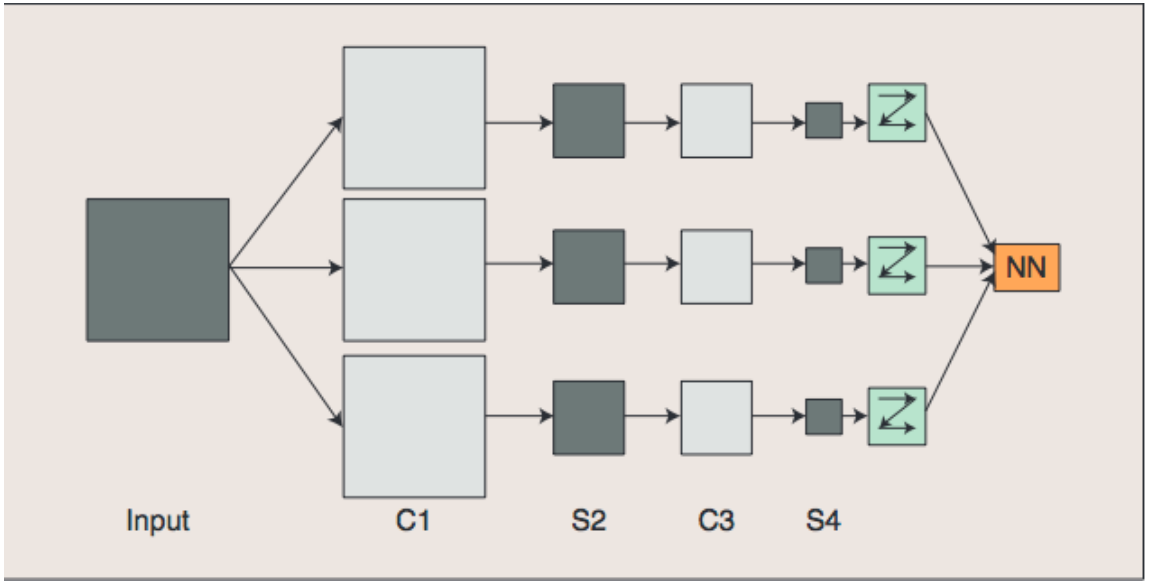
- Instead of treating input to a fully connected network
- Two layers of Neural networks are used
  1. Layer of convolutional units
     - which consider overlapping regions
  2. Layer of subsampling units
     - Also called "pooling"
- Several feature maps and sub-sampling
  - Gradual reduction of spatial resolution compensated by increasing no. of features
- Final layer has softmax output
- Whole network trained using backpropagation
  - Including those for convolution and sub-sampling



Each pixel patch is 5 x 5

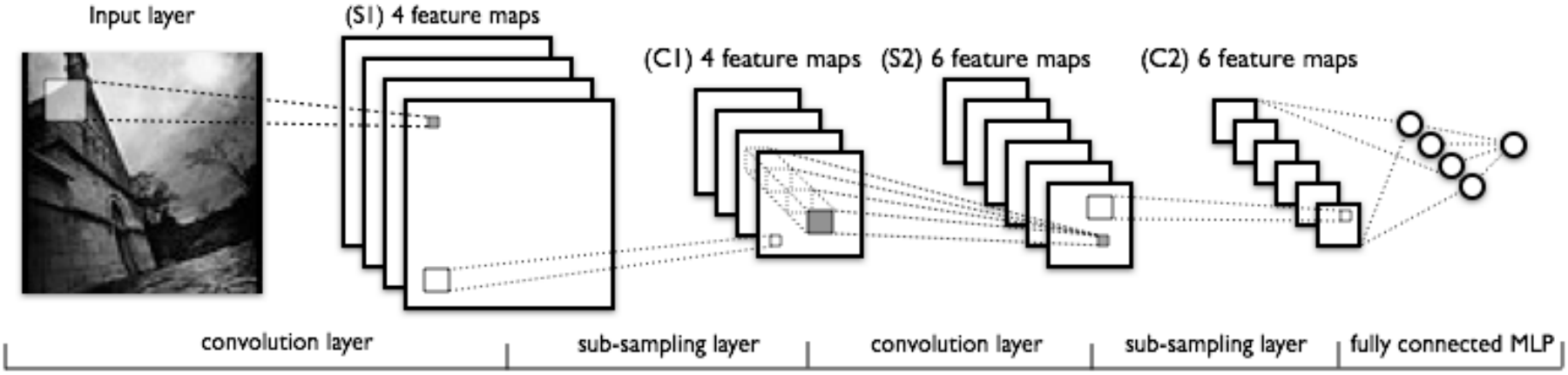
This plane has $10 \times 10 = 100$ neural network units (called a feature map). Weights are same for different planes. So only 25 weights are needed. Due to weight sharing this is equivalent to convolution. Different features have different feature maps

Two layers of convolution and sub-sampling



1. Convolve Input image with three trainable filters and biases to produce three feature maps at the C1 level
2. Each group of four pixels in the feature maps are added, weighted, combined with a bias, and passed through a sigmoid to produce feature maps at S2.
3. These are again filtered to produce the C3 level.
4. The hierarchy then produces S4 in a manner analogous to S2
5. Finally, rasterized pixel values are presented as a vector to a "conventional" neural network

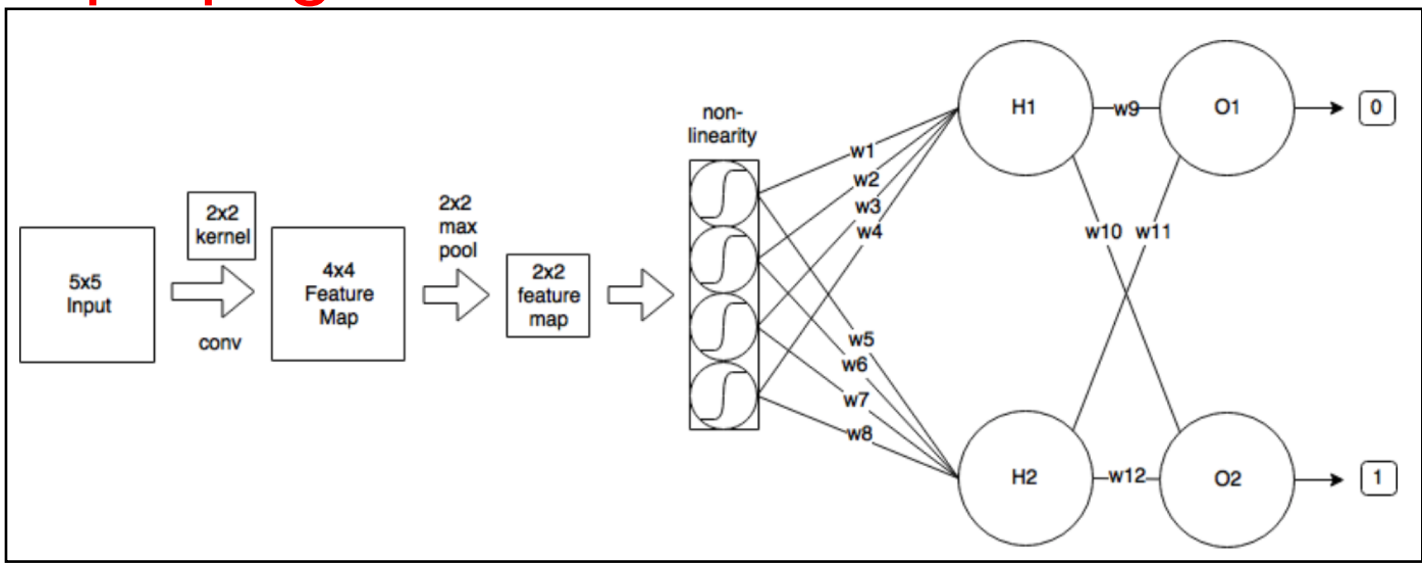
Two layers of convolution and sub-sampling



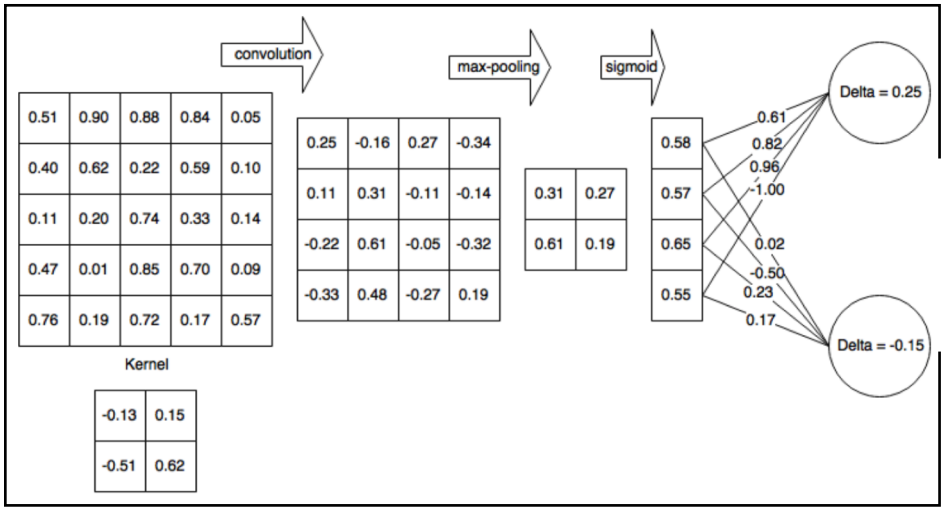
- By weight sharing, invariance to small transformations (translation, rotation achieved)
  - Regularization
- Similar to biological networks
  - Local receptive fields
- Smart way of reducing dimensionality before applying a full neural network

Backpropagation in a CNN

1. Input image size $5x5$
2. Apply Convolution with $2x2$ kernel
3. Apply $2x2$ maxpooling stride$=2$ reduces feature map to $2x2$
4. Apply logistic sigmoid
5. FC layer with 2 neurons
6. Then an output layer



After complete forward pass and partial backward pass:



Assume we have already completed forward pass and computed $\delta H1 = 0.25$ and $\delta H2 = 0.15$

$$\delta_{11} = (0.25 * 0.61 + -0.15 * 0.02) * 0.58 * (1 - 0.58) = 0.0364182$$
$$\delta_{12} = (0.25 * 0.82 + -0.15 * -0.50) * 0.57 * (1 - 0.57) = 0.068628$$
$$\delta_{21} = (0.25 * 0.96 + -0.15 * 0.23) * 0.65 * (1 - 0.65) = 0.04675125$$
$$\delta_{22} = (0.25 * -1.00 + -0.15 * 0.17) * 0.55 * (1 - 0.55) = -0.06818625$$

Then propagate δs to the $4x4$ layer
Set all values filtered out by maxpooling to zero
Gradient map:

0	0	0.0686	0
0	0.0364	0	0
0	0.0467	0	0
0	0	0	-0.0681

Advantages of Convolutional Network Architecture

- Minimize computation compared to a regular neural network
- Convolution simplifies computation to a great extent without losing the essence of the data
- They are great at handling image classification
- They use the same knowledge across all image locations