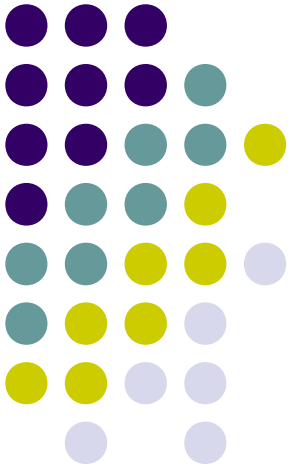
Pattern Recognition
Convolutional Neural Networks

Francesco Tortorella

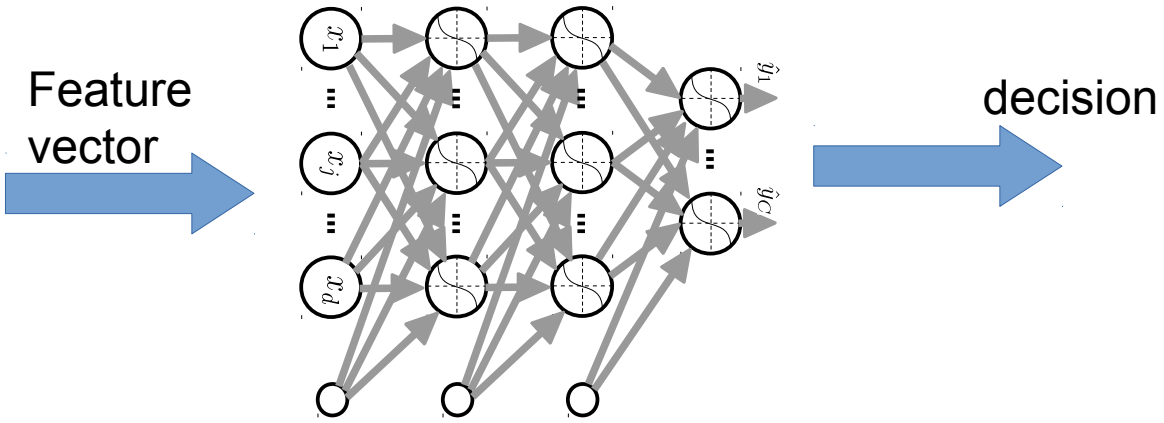
University of Cassino and Southern Latium

Cassino, Italy



Neural networks as classifiers

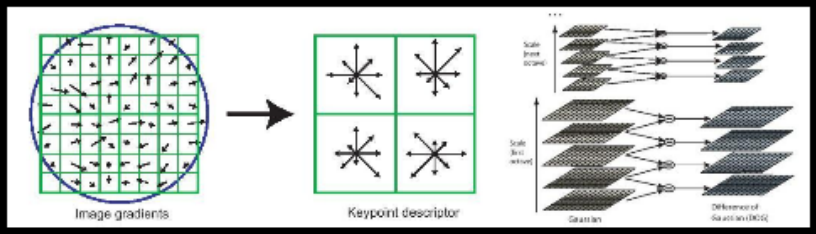
- Neural networks: another classification system
- The (deep) structure able to build a complex decision system made of a hierarchy of simpler, interconnected decision nodes.
- In principle, we have still the “classical” PR scheme:



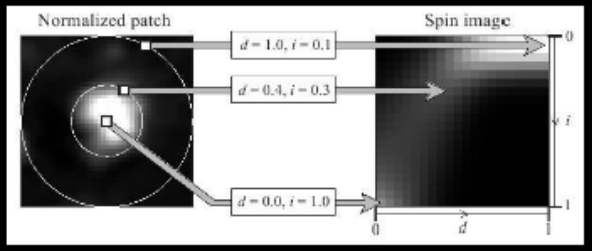
Feature design

- Features are tipically hand crafted.
- Tuning the features is difficult, time-consuming and requires expert knowledge

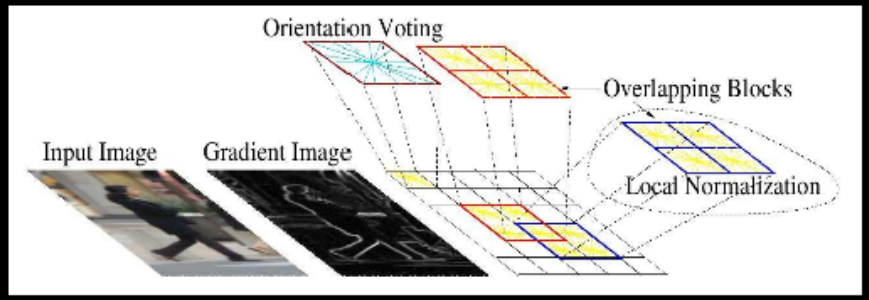
Feature design: computer vision



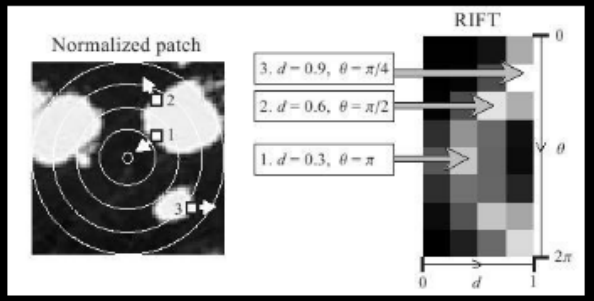
SIFT



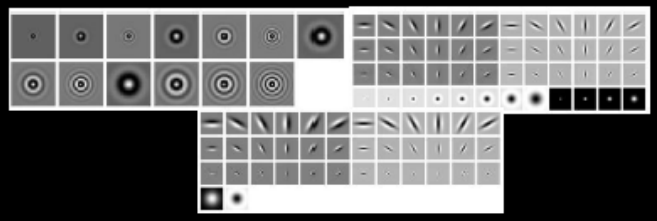
Spin image



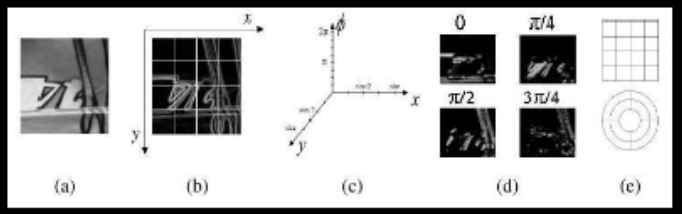
HoG



RIFT



Textons

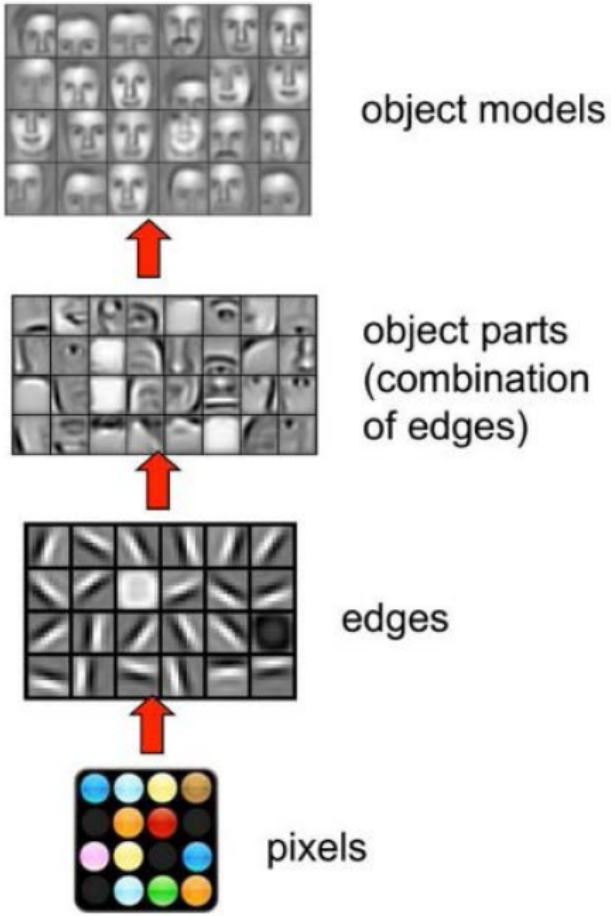


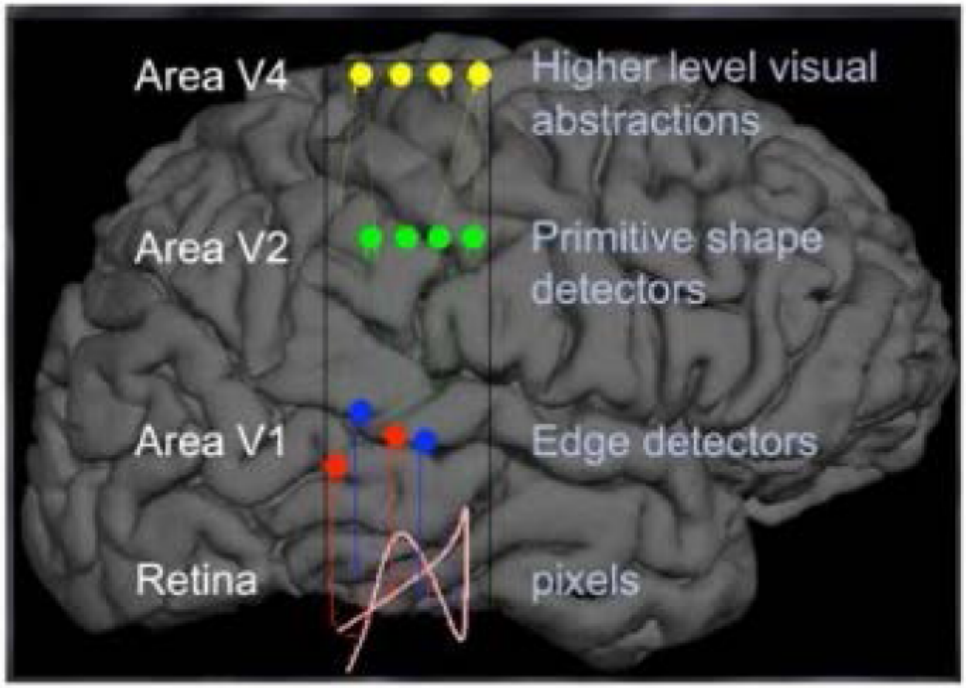
GLOH

From feature design to feature learning

- Possible to devise a system for learning the features?
- Possible to design feature learners instead of features?
- Hierarchical approach still valid?
  - It seems so. Why?

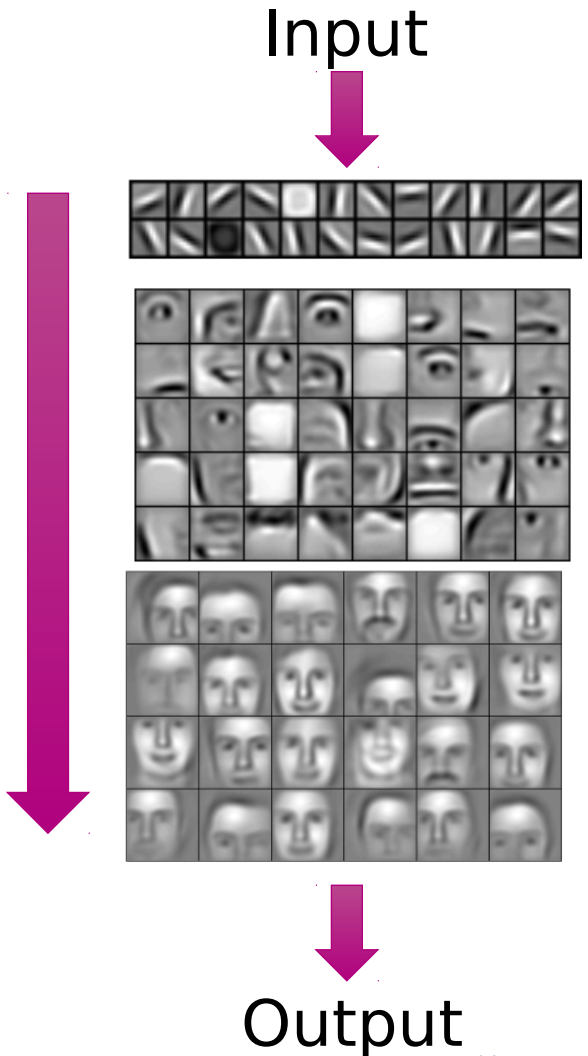
Biologically founded

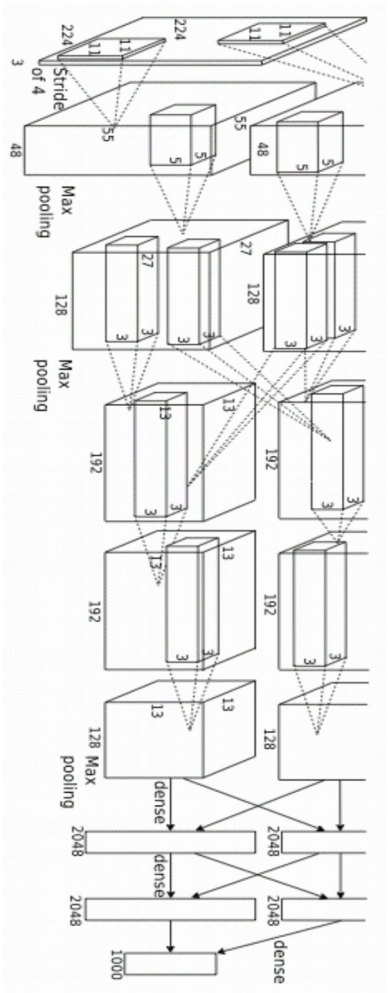




© Andrew Ng

Using neural networks to learn features





Convolutional Neural Networks

- Convolutiona Neural Network (CNN, ConvNet) architectures make the explicit assumption that the inputs are images
- This allows us to encode certain properties into the architecture and make more efficient the processing by reducing the amount of parameters in the network.

CNN architecture

- Three different kinds of layer:
  - Convolutional Layer
  - Pooling Layer
  - Fully-Connected Layer (exactly as seen in regular Neural Networks)
- We will stack these layers to form a full ConvNet architecture.

CNN architecture: the convolutional layer

- The ***convolution*** of f and g is defined as the integral of the product of the two functions after one is reversed and shifted (Wikipedia)
- Continuous

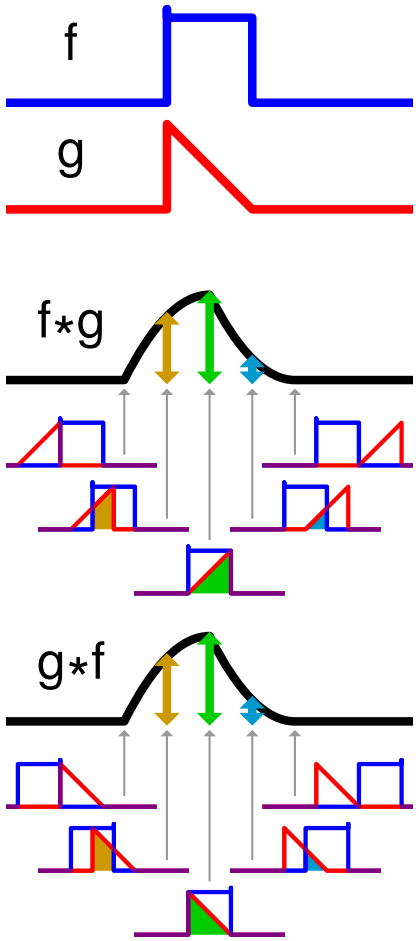
$$(f * g)(t) = \int f(\tau)g(t-\tau)d\tau = \int f(t-\tau)g(\tau)d\tau$$

- Discrete

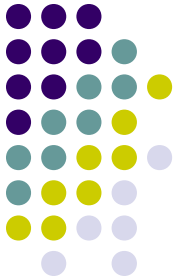
$$(f * g)[n] = \sum_m f[m]g[n-m] = \sum_m f[n-m]g[m]$$

CNN architecture: the convolutional layer

- In $(f * g)(t) = \int f(\tau)g(t-\tau)d\tau$ g is referred to as the kernel.
- Note that the kernel is flipped.



CNN architecture: the convolutional layer



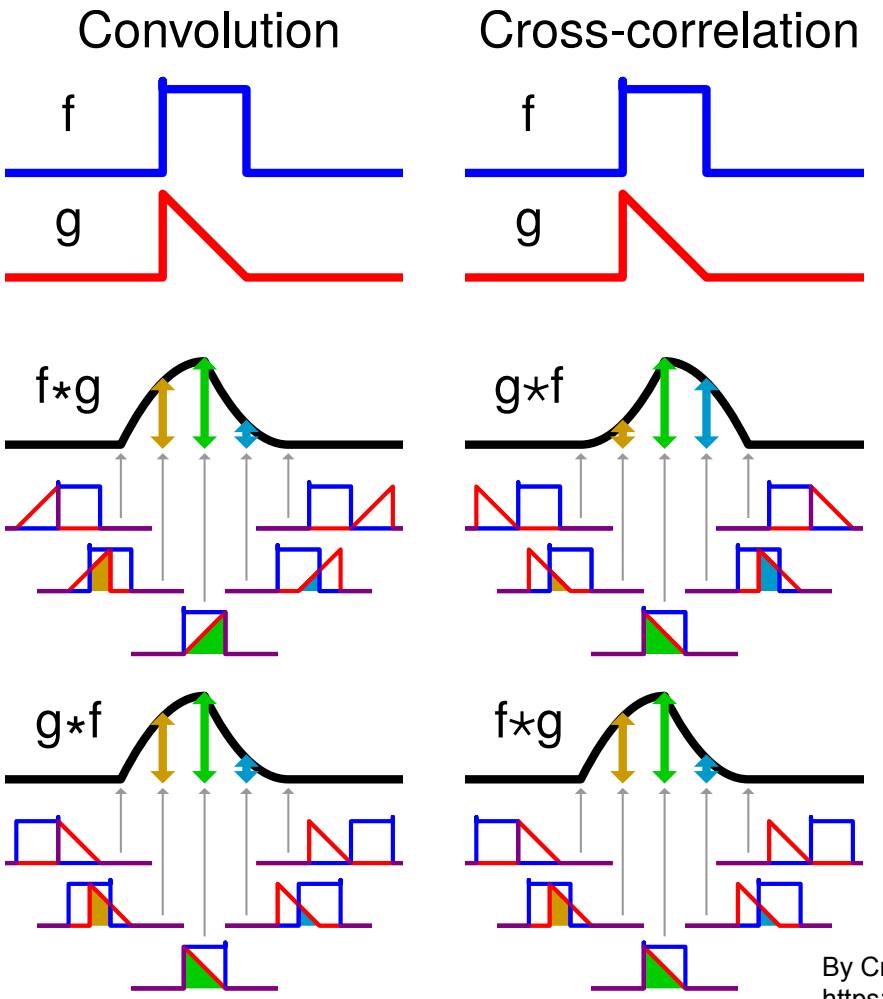
- ***Cross-correlation*** is a related operation which is the same as convolution, but without flipping the kernel
- Continuous

$$(f \star g)(t) = \int f(\tau) g(t + \tau) d\tau$$

- Discrete

$$(f \star g)[n] = \sum_m f[m] g[n + m]$$

CNN architecture: the convolutional layer



By Cmglee - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=20206883

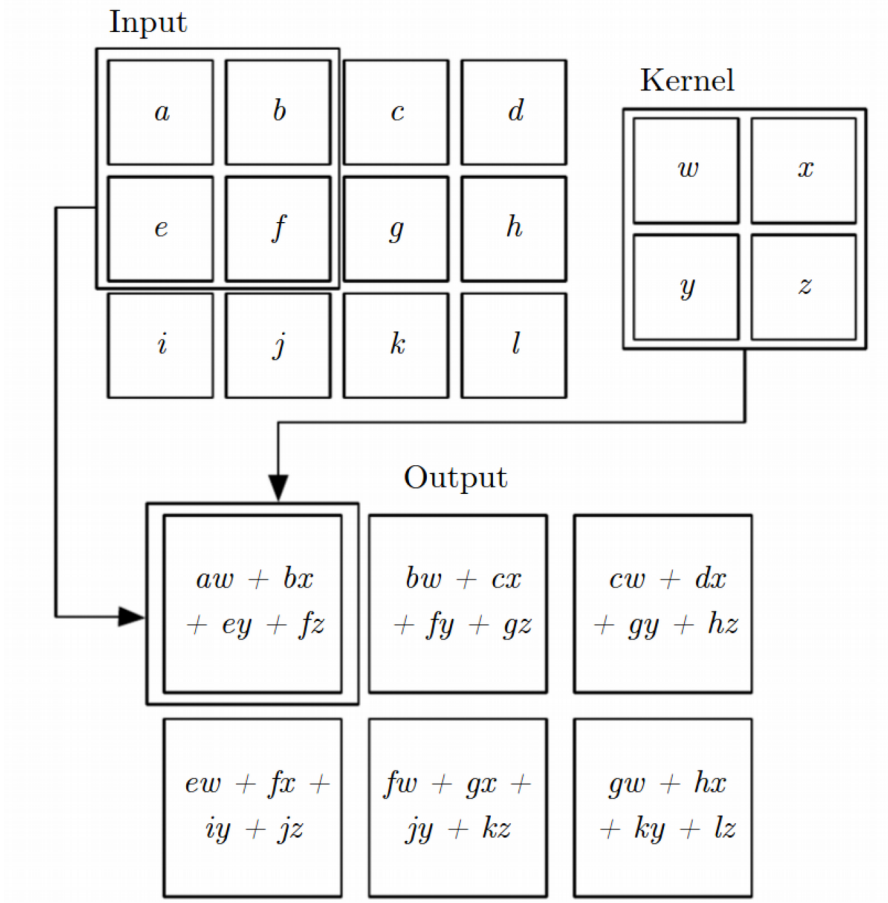
CNN architecture: the convolutional layer

- Many machine learning libraries implement cross-correlation but call it convolution.
- The convolutional layer accomplishes a cross-correlation between the input image and one or more 2D kernels with finite size

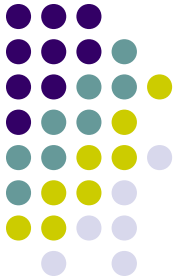
$$S(i,j) = (I \star K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

CNN architecture: the convolutional layer

- Example
  - Input 3x4
  - Kernel 2x2
  - Output 2x3



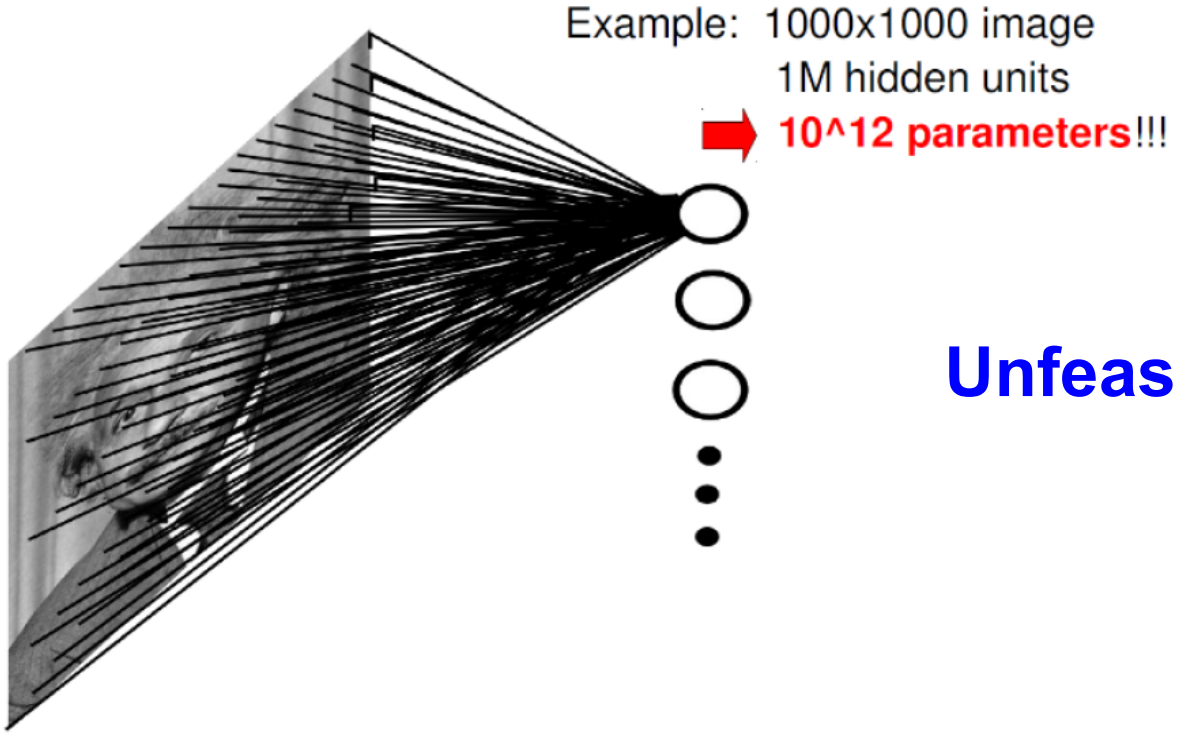
Convolutional layer
Distinguished features



- Locally Receptive Fields
- Shared Weights
- Sub-sampling

Locally receptive field

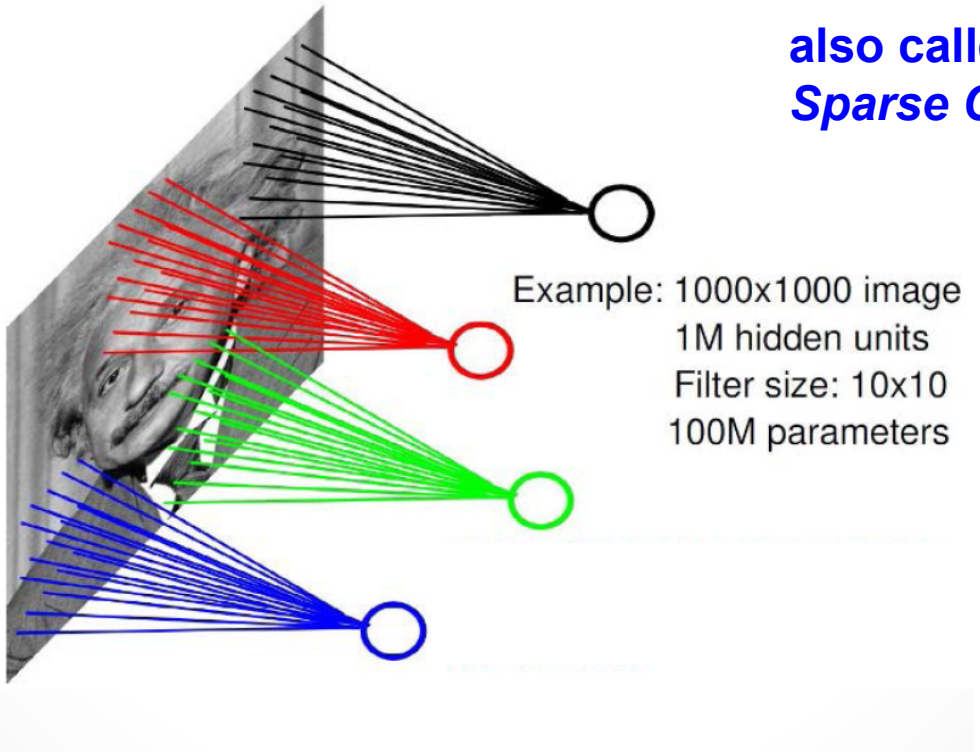
In traditional NN layers every output unit interacts with every input unit.



Unfeasible!

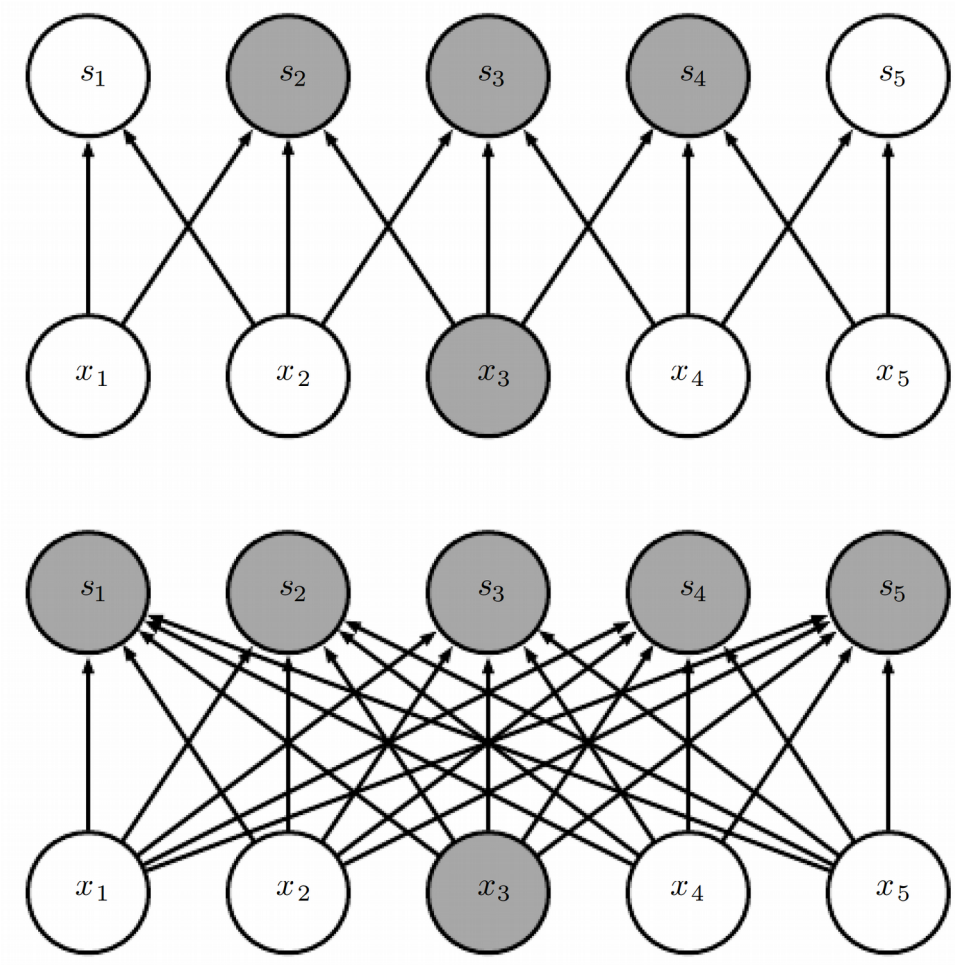
Locally receptive field

Each neuron is connected to only a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the ***receptive field*** of the neuron (equivalently this is the kernel/filter size).

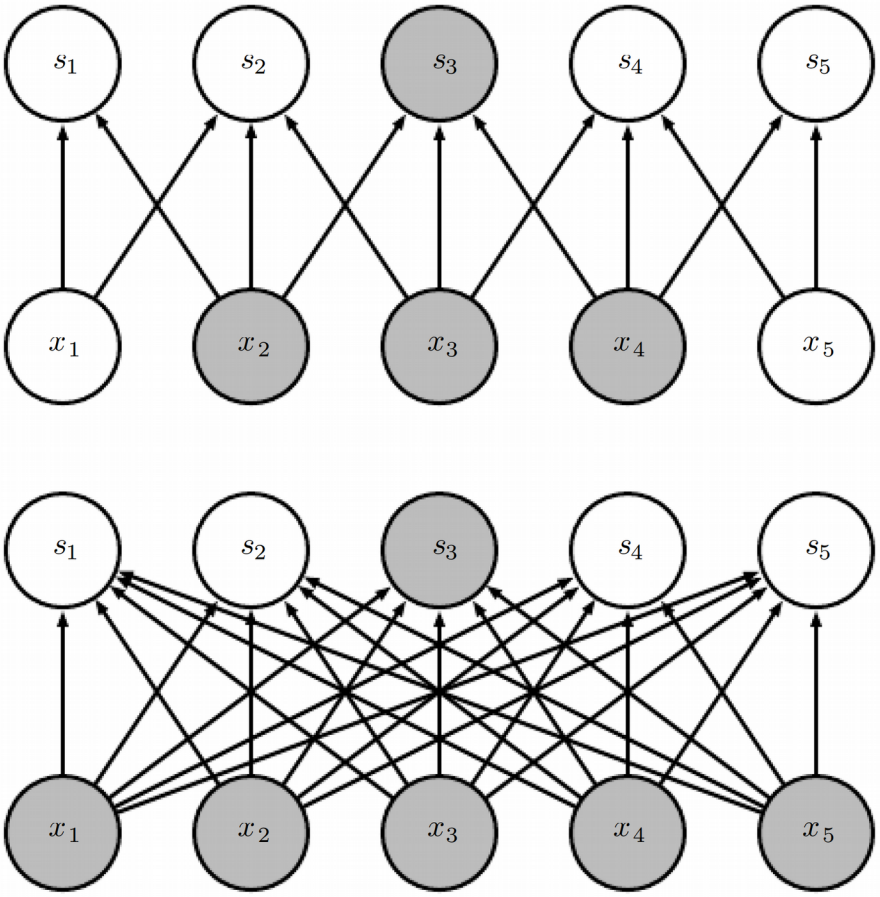


also called
Sparse Connectivity

Sparse connectivity (viewed from below)

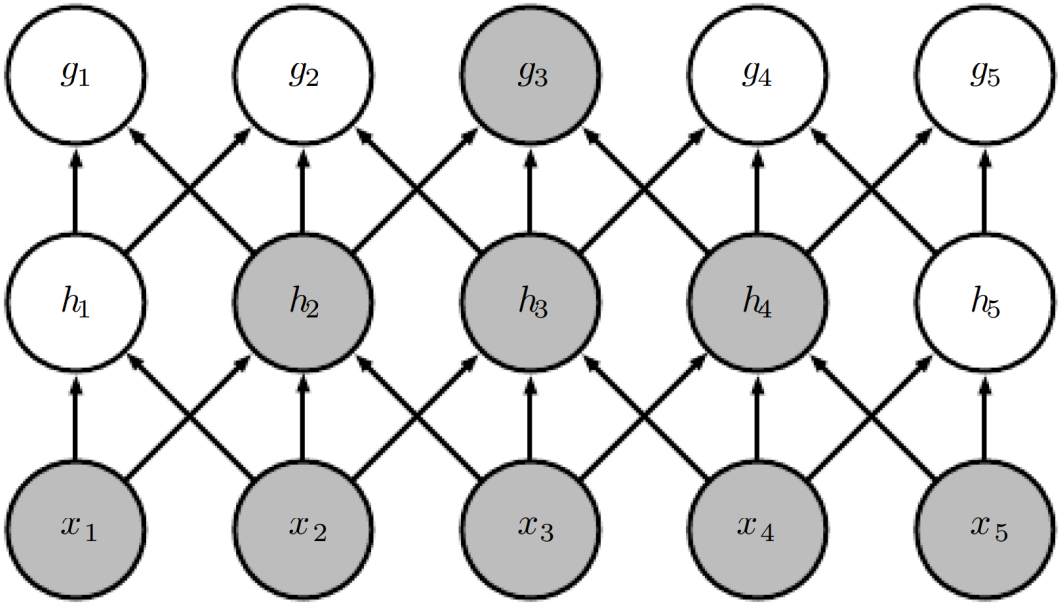


Sparse connectivity (viewed from above)



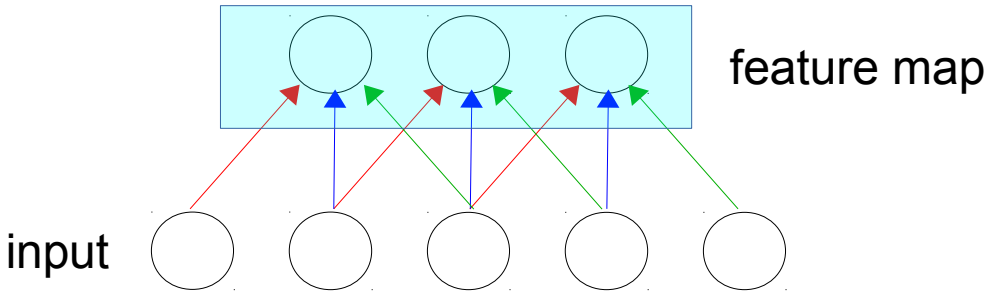
Receptive field in more layers

- Receptive fields of units in deeper layers larger than shallow layers
- Though direct connections are very sparse, deeper layers indirectly connected to most of the input image
- Effect increases with strided convolution or pooling



Shared weights/ Parameter sharing

- All neuron in the hidden layer share the same parameterization (weight vector and bias) forming a 'Feature Map‘

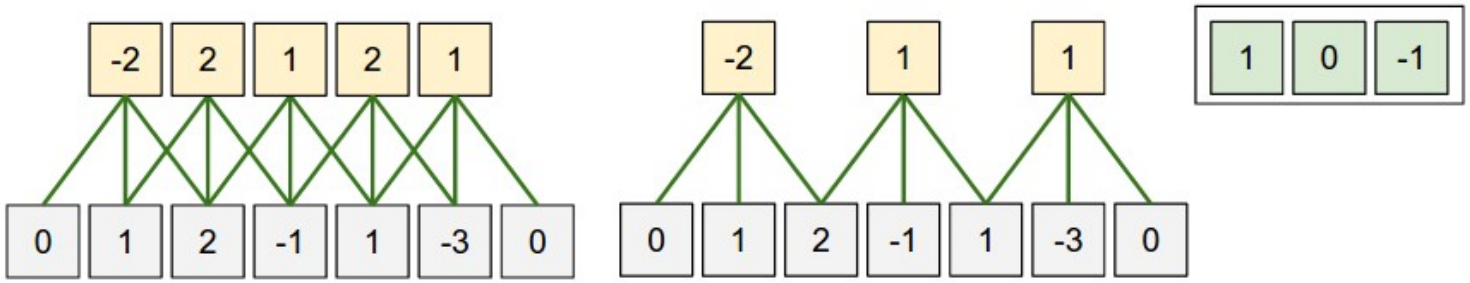


Shared weights/ Parameter sharing

- This causes the layer to have **equivariance to translation**
- Allows features to be detected regardless of their position in the visual field. (Feature is a kind of input pattern that will cause a neuron to activate, eg. an edge)
- All neurons in the first hidden layer detect exactly the same feature, just at different locations.
- CNNs are robust to the translation of objects in the image.
- Further reduces the number of free parameters, achieving better generalization and computational performance.

Downsampling: stride

- We may want to skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting our features as finely).
- We can fix a **stride** ≥ 1 with which we slide the kernel over the image.

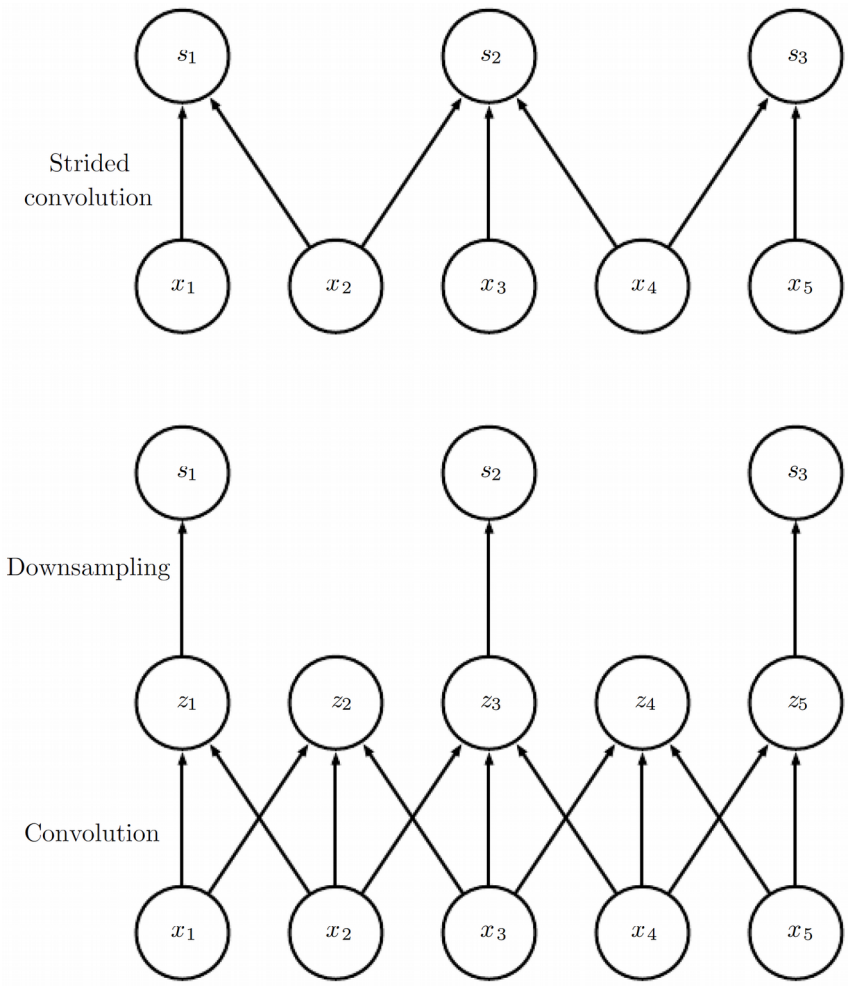


stride = 1

stride = 2

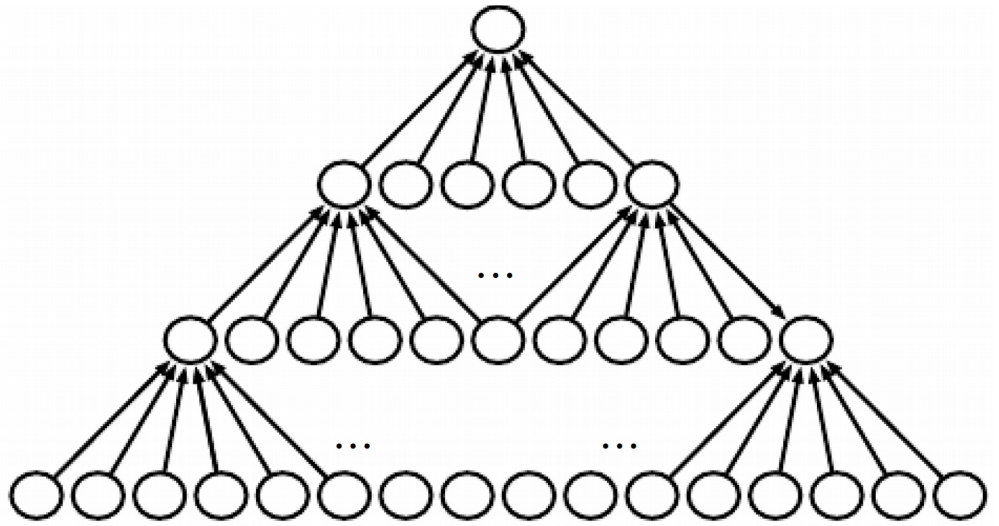
Downsampling: stride

We can think of this as downsampling the output of the full convolution function.



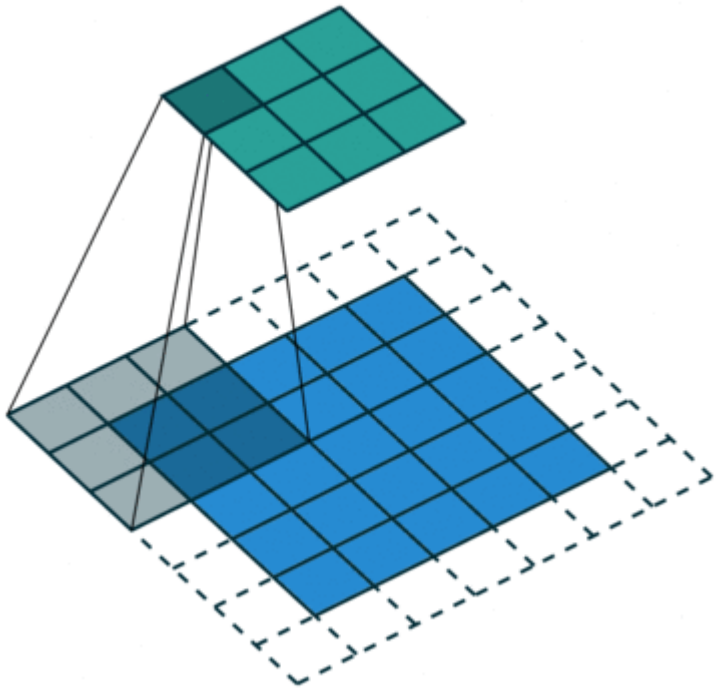
Padding

- When a convolution is accomplished, the output is smaller than the input. The representation is shrinked by some pixels at each layer. This could lead to an output too small.
- The rate of shrinking can be mitigated by using smaller kernels.
- Not always possible.



Padding

- We can add some implicit zeros to each layer to prevent the representation from shrinking with depth (**padding**).



Padding variants

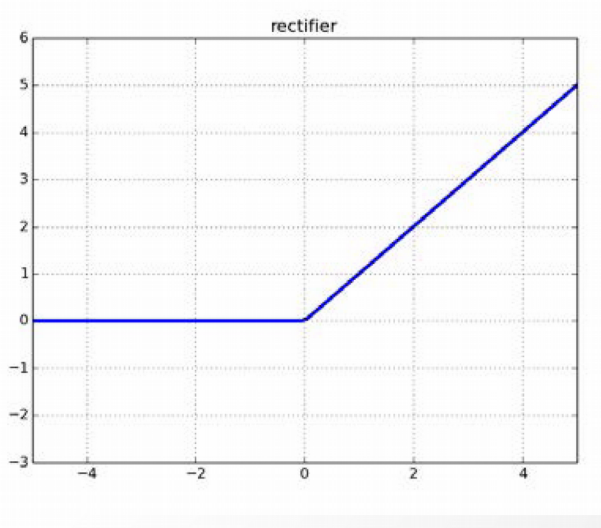
- **Valid**
  - No zero-padding, kernel is restricted to traverse only within the image
    output size: (m - k + 1) x (m - k + 1)
- **Same**
  - Add zero-padding to the image to have the output of the same size as the image
    output size: m x m
- **Full**
  - Add zero-padding to the image enough for every pixel to be visited k times in each direction
    output size: (m + k - 1) x (m + k - 1)

Spatial arrangement

- We can compute the spatial size of the output layer as a function of the input layer size (W), the receptive field size of the Conv Layer neurons (F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border.
- The correct formula for calculating how many neurons "fit" is given by **(W−F+2P)/S+1**.
- For example for a 7x7 input and a 3x3 filter with stride 1 and pad 0 we would get a 5x5 output. With stride 2 we would get a 3x3 output.

Non linear activation function

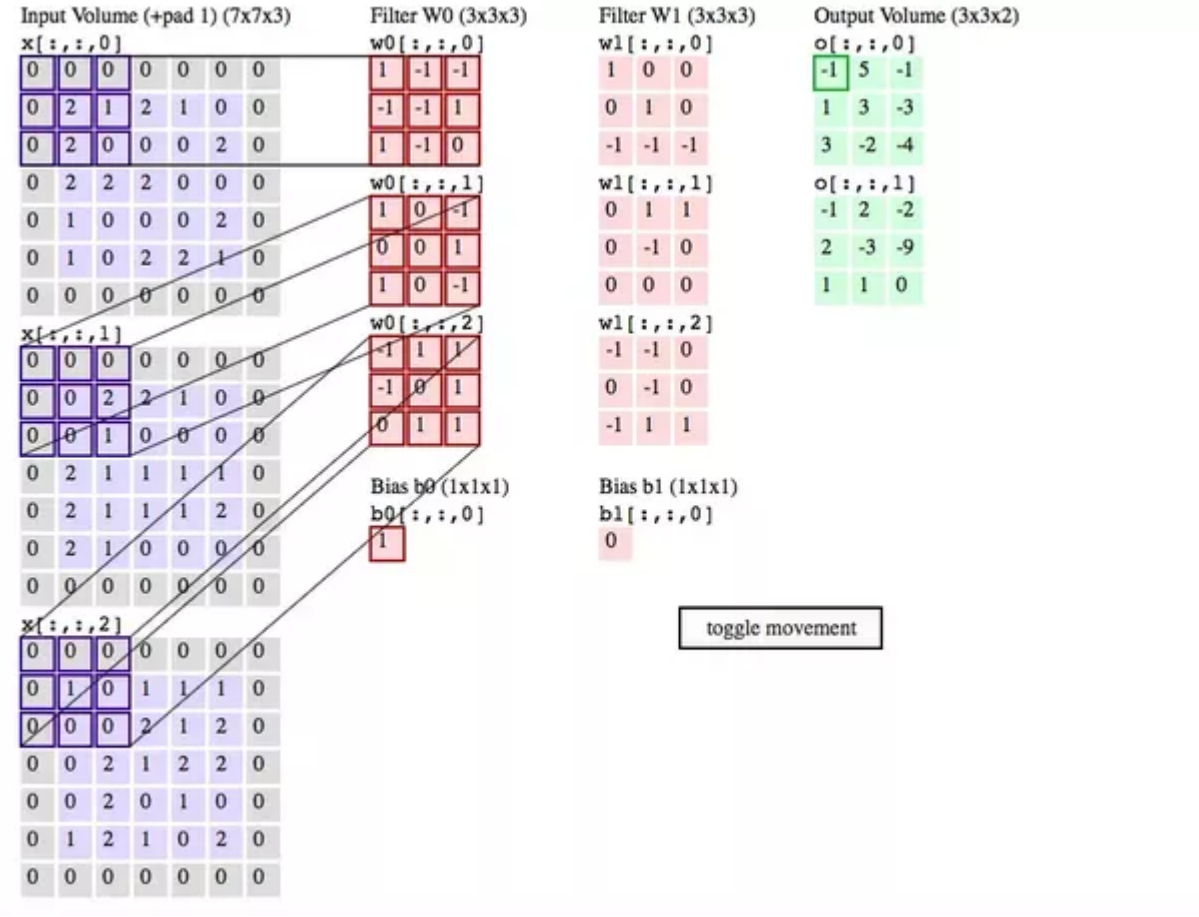
- Feature Map - Obtained by convolution of the image with a linear filter, adding a bias term and applying a non-linear function
- Most popular activation function:
  Rectified Linear Unit (ReLU)



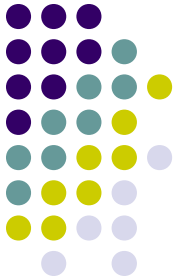
General organization

- We made two implicit assumptions
  - The input is 2D (only one slice)
  - The kernel to apply is only one
- Actually we can have
  - An input with 3 or more dimensions (e.g. Image RGB)
  - More kernels to apply in the same layer

General organization

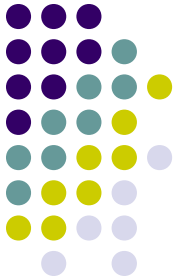


Pooling Layer



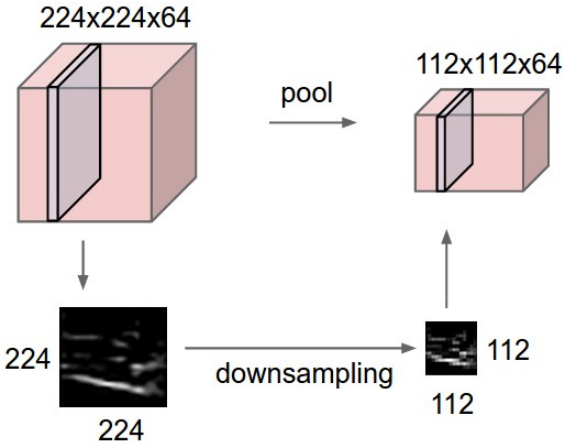
- It is common to insert a Pooling layer in-between successive Conv layers. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a MAX pooling layer with filters of size 2x2 applied with a stride of 2.
- It downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged.

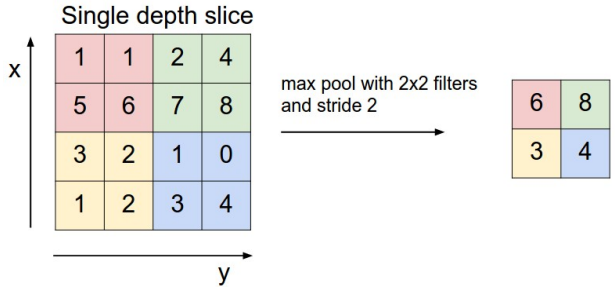
Pooling Layer



- More generally, the pooling layer:
  - Accepts a volume of size W1×H1×D1W1×H1×D1
  - Requires two hyperparameters:
    - the spatial extent F,
    - the stride S,
- Produces a volume of size W2×H2×D2 where:
  - W2=(W1−F)/S+1
  - H2=(H1−F)/S+1
  - D2=D1
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers
- It is worth noting that there are only two commonly seen variations of the max pooling layer found in practice: A pooling layer with F=3,S=2F=3,S=2 (also called overlapping pooling), and more commonly F=2,S=2F=2,S=2. Pooling sizes with larger receptive fields are too destructive.

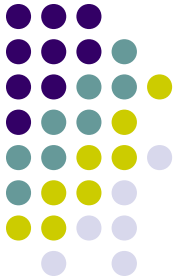
Pooling layer





Pooling with
F=2 S=2

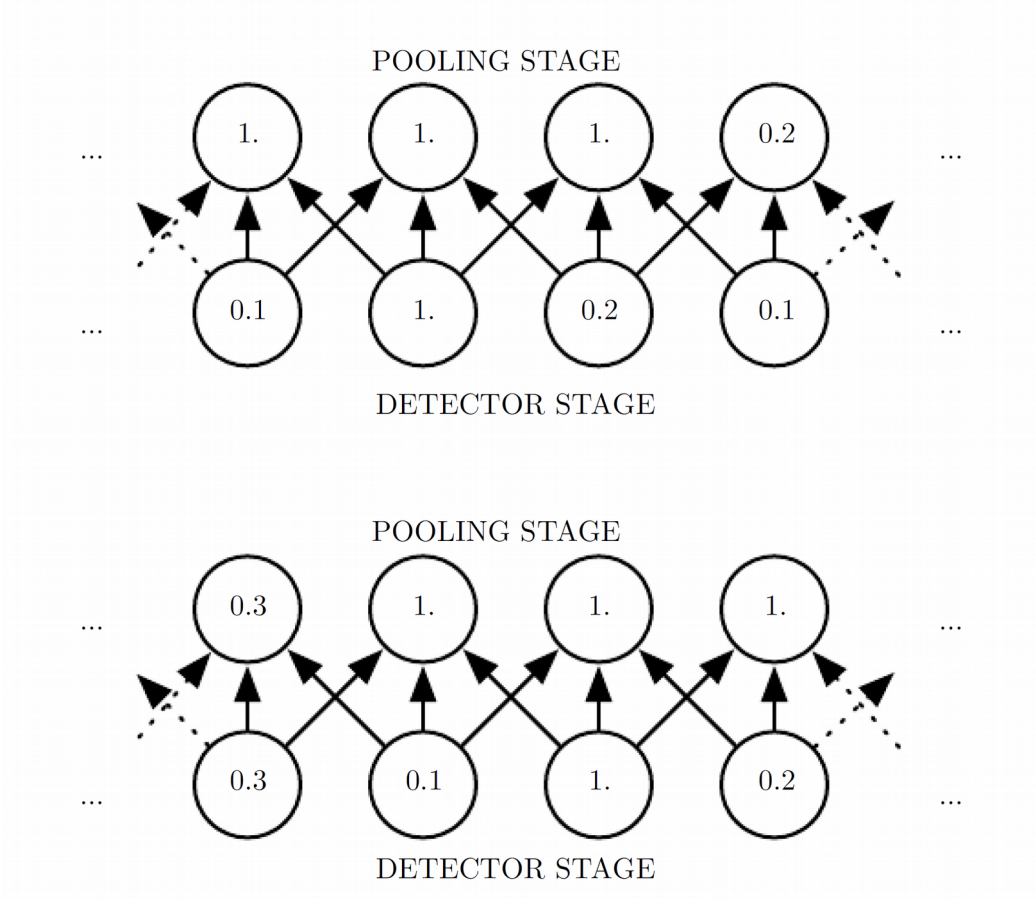
Pooling Layer



- Variants:
  - Max pooling (popular)
  - Weighted average based on distance
  - L2 norm of neighborhood
- Reduces computation for upper layers by reporting summary statistics (only with stride > 1)
- Provides translation invariance (learning must be invariant to small translations)
- Useful property, if we care more about whether some feature is present than exactly where it is (adds robustness to position)

Pooling layer

MAX pooling introduces invariance



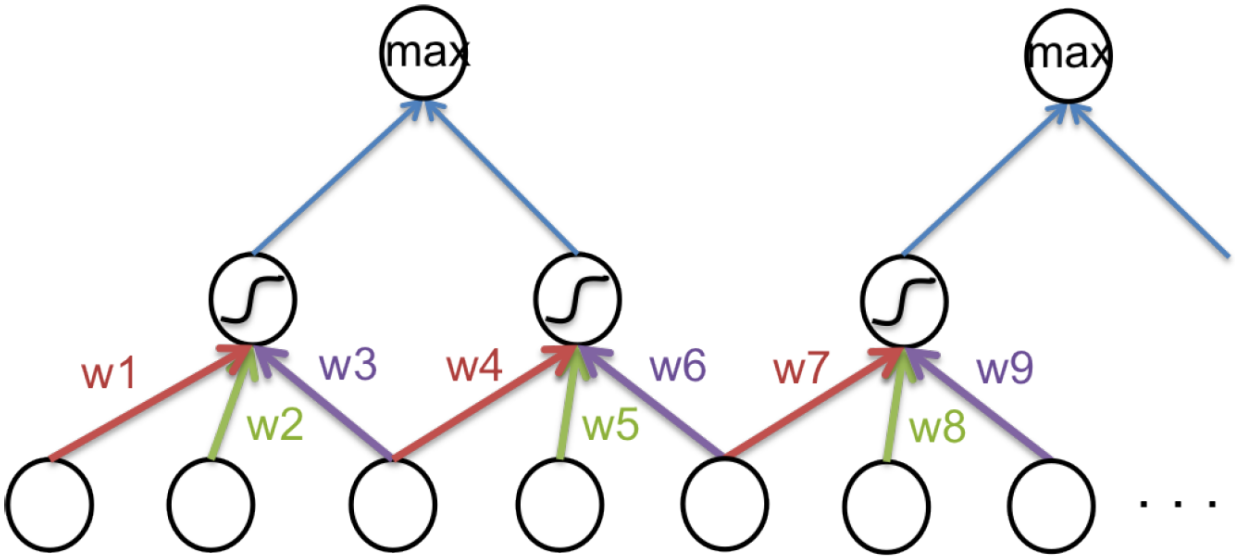
Fully-connected layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

Back propagation

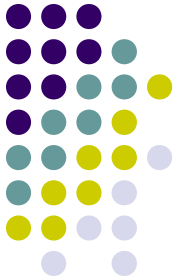
- How should be modified the backpropagation algorithm to work with these layers?
- In the forward step, the information flows as usual from input layer toward outpuyt layer.
  - For the max-pooling layer we need to remember what branch gives the max value
- During the backward step, on the pooling layer we only compute the gradient for that branch.
- As for the convolutional layer, every neuron in the layer will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.

Back propagation



Compute the gradient for all the weights: $\frac{\partial J}{\partial w_1} \ \ldots \ \frac{\partial J}{\partial w_9}$

Average the gradient on the shared weights: $w_1 = w_1 - \alpha \left(\frac{\partial J}{\partial w_1} + \frac{\partial J}{\partial w_4} + \frac{\partial J}{\partial w_7} \right)$



- **Backpropagation**. Recall from the backpropagation chapter that the backward pass for a max(x, y) operation has a simple interpretation as only routing the gradient to the input that had the highest value in the forward pass. Hence, during the forward pass of a pooling layer it is common to keep track of the index of the max activation (sometimes also called the switches) so that gradient routing is efficient during backpropagation.
-
- **Getting rid of pooling.** Many people dislike the pooling operation and think that we can get away without it. For example, Striving for Simplicity: The All Convolutional Net proposes to discard the pooling layer in favor of architecture that only consists of repeated CONV layers. To reduce the size of the representation they suggest using larger stride in CONV layer once in a while. Discarding pooling layers has also been found to be important in training good generative models, such as variational autoencoders (VAEs) or generative adversarial networks (GANs). It seems likely that future architectures will feature very few to no pooling layers.