Introduction to RL - EECE695

Lecture Note 9
Convolutional Neural Networks

Convolution in Machine Learning

Convolution in continuous-time domain:

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

- x(t): input, w(t): kernel

Convolution in discrete-time domain:

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

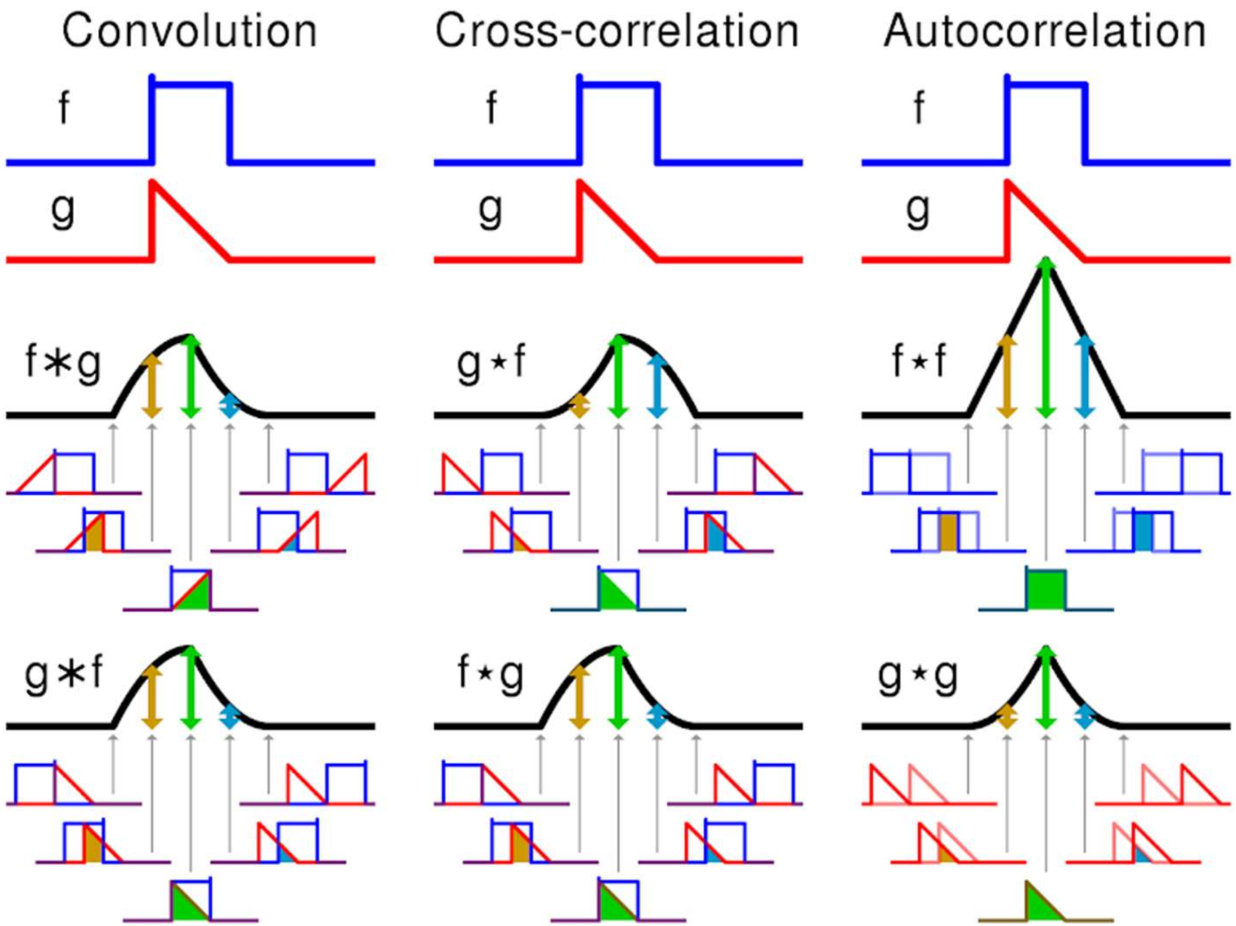
Two-dimensional convolution:

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

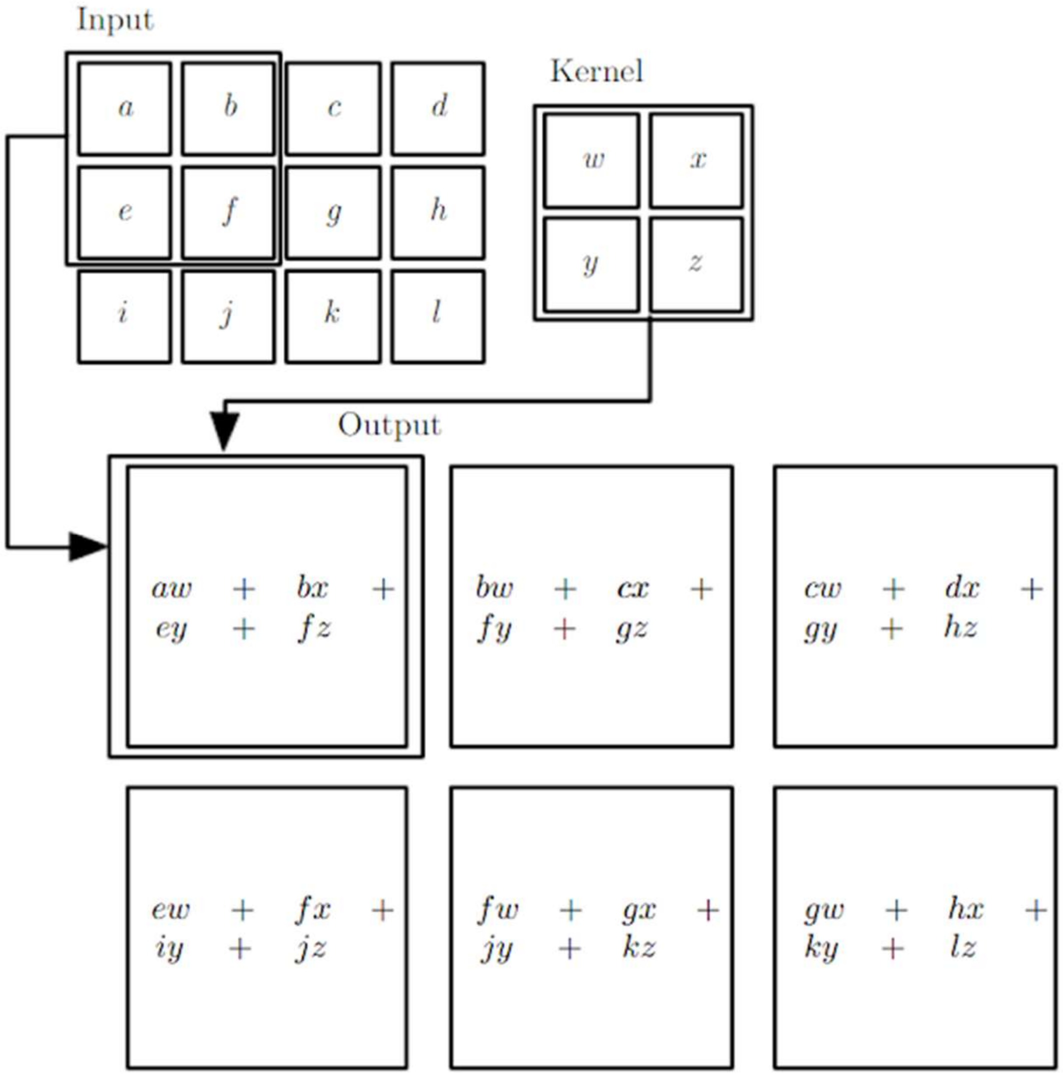
cf) Cross-correlation: $$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Visualization

$f * g = \sum f[a]g[-(a-t)]$ $g * f = \sum f[a]g[a+t]$ $f * f = \sum f[a]f[a+t]$



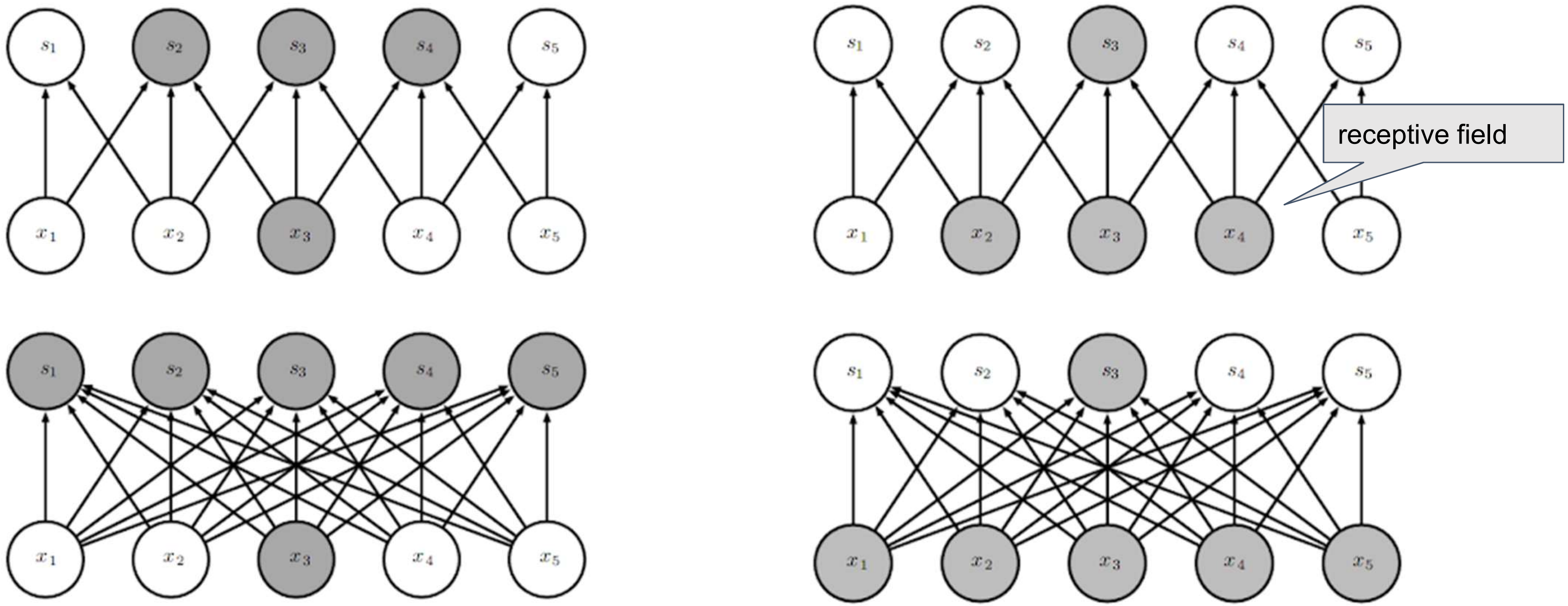
2-D Convolution



Sparse Interactions of Convolution

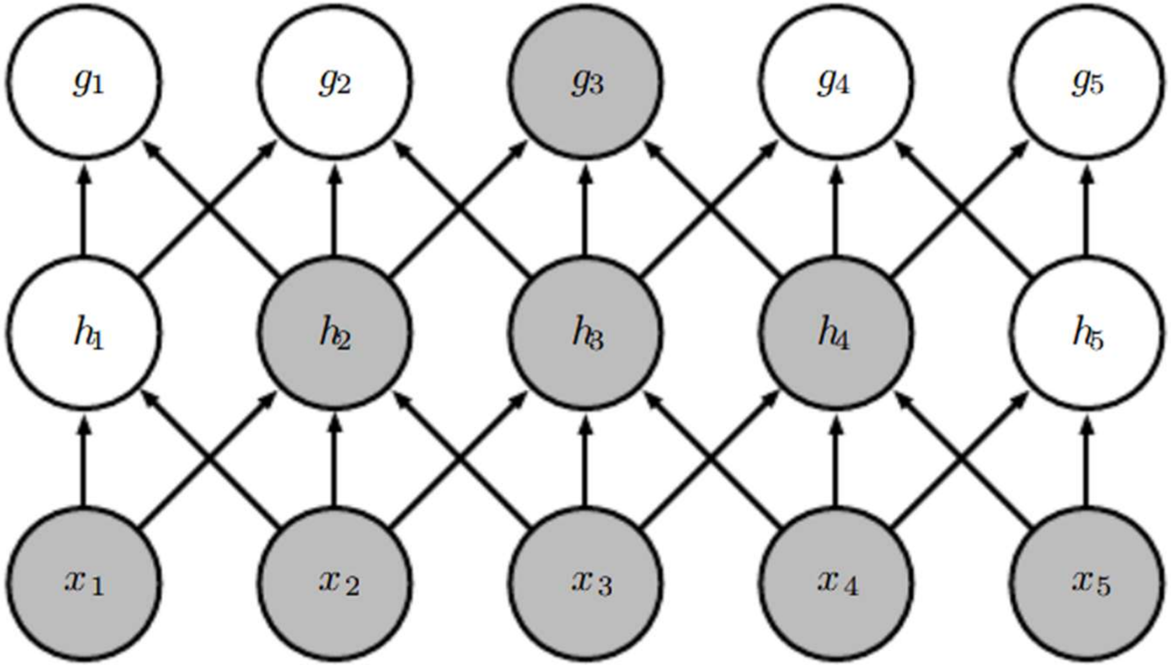
Via convolution, we can

- detect small, meaningful features such as edges
- reduce the memory requirements of the model
- improve the statistical efficiency
- lower the computational complexity

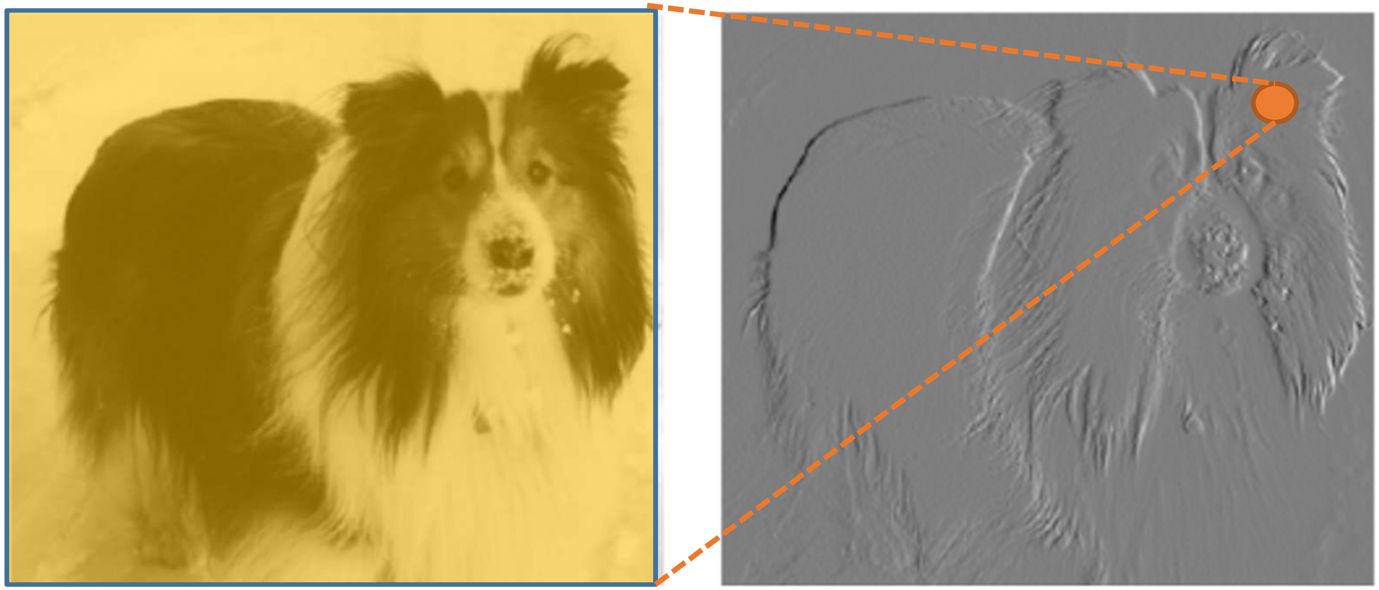


Sparse Interactions of Convolution

As the model becomes deeper…



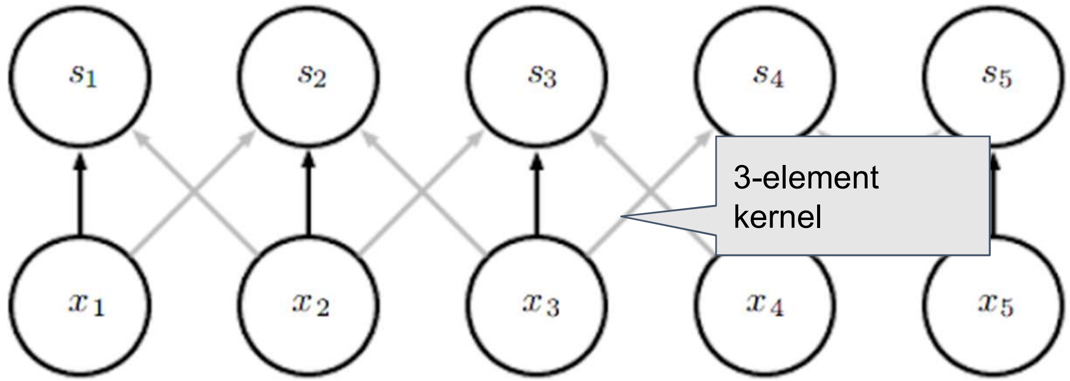
Sparse Interactions of Convolution

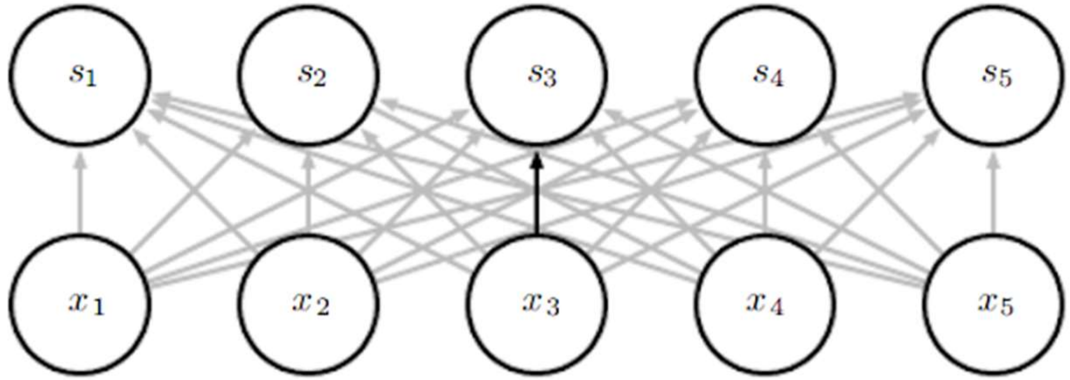


Parameter Sharing in Convolution

Because of the parameter sharing,

- we are learning only one set of parameters
- we can reduce the storage requirement
- we can lower the computational complexity
- we can improve the statistical efficiency

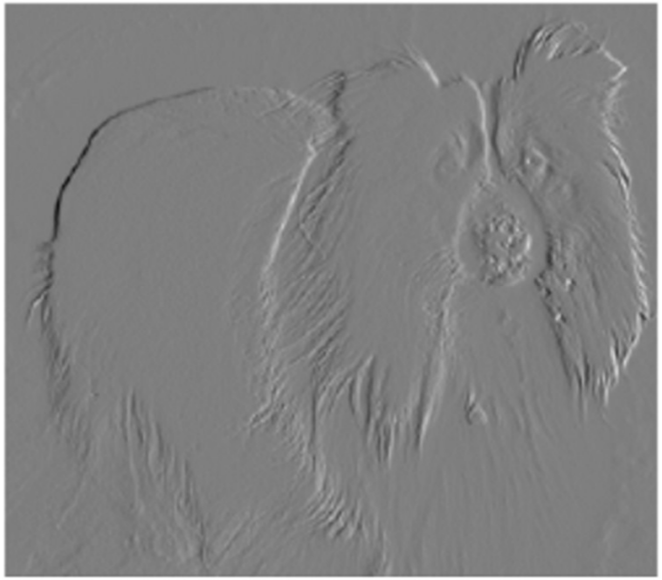




Extracting Essential Features

Efficiency of edge detection

- the right image was formed by taking each pixel and subtracting the value of each pixel's left-pixel ⇒ special case of convolution
- much reduced number of parameters, and computational complexity



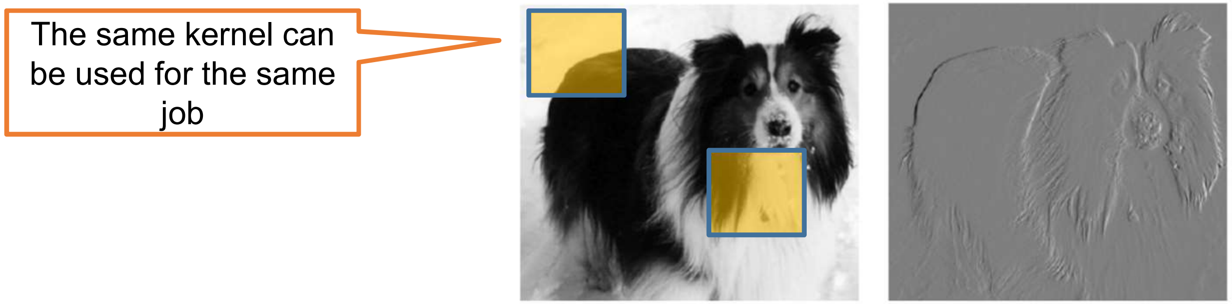
Equivariance to Translation of Convolution

Definition of equivariance:

$$\boxed{\text{a function } f(x) \text{ is equivariant to a function } g \text{ if } f(g(x)) = g(f(x))}$$

Convolution is equivariant to the shifting function

- Let the shifting function be defined by $I'(x, y) = I(x-1, y)$.
- (image → shifting → convolution) = (image → convolution → shifting).
- If we move the object in the input → its representation will move the same amount in the output.
- useful when applied to multiple locations to do the same job, e.g., finding edges.

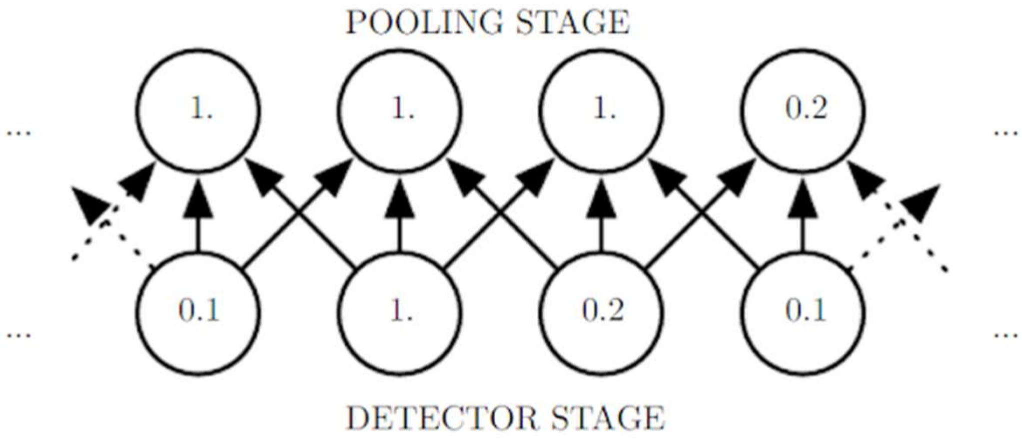


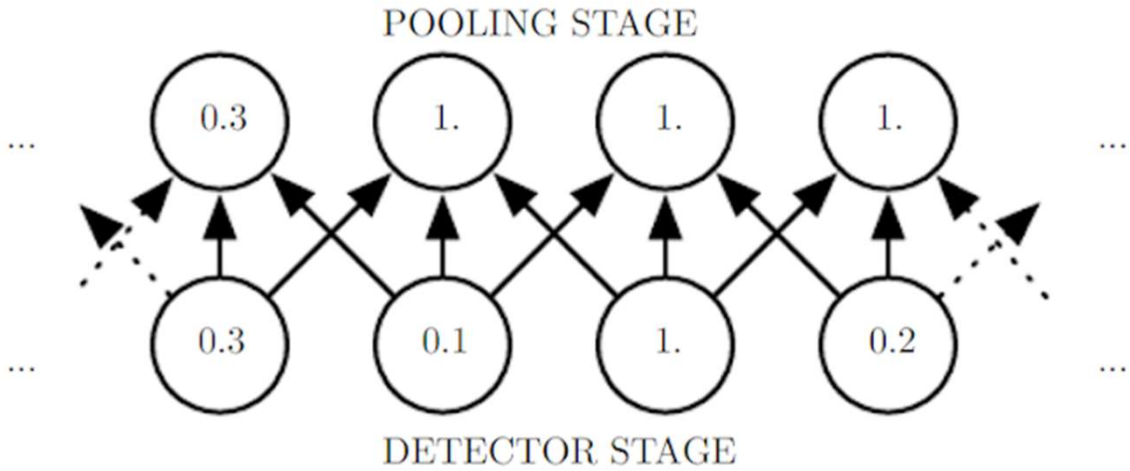
Convolution is not naturally equivariant to some other transformations, such as scale changes or rotation changes..

Pooling

Max pooling

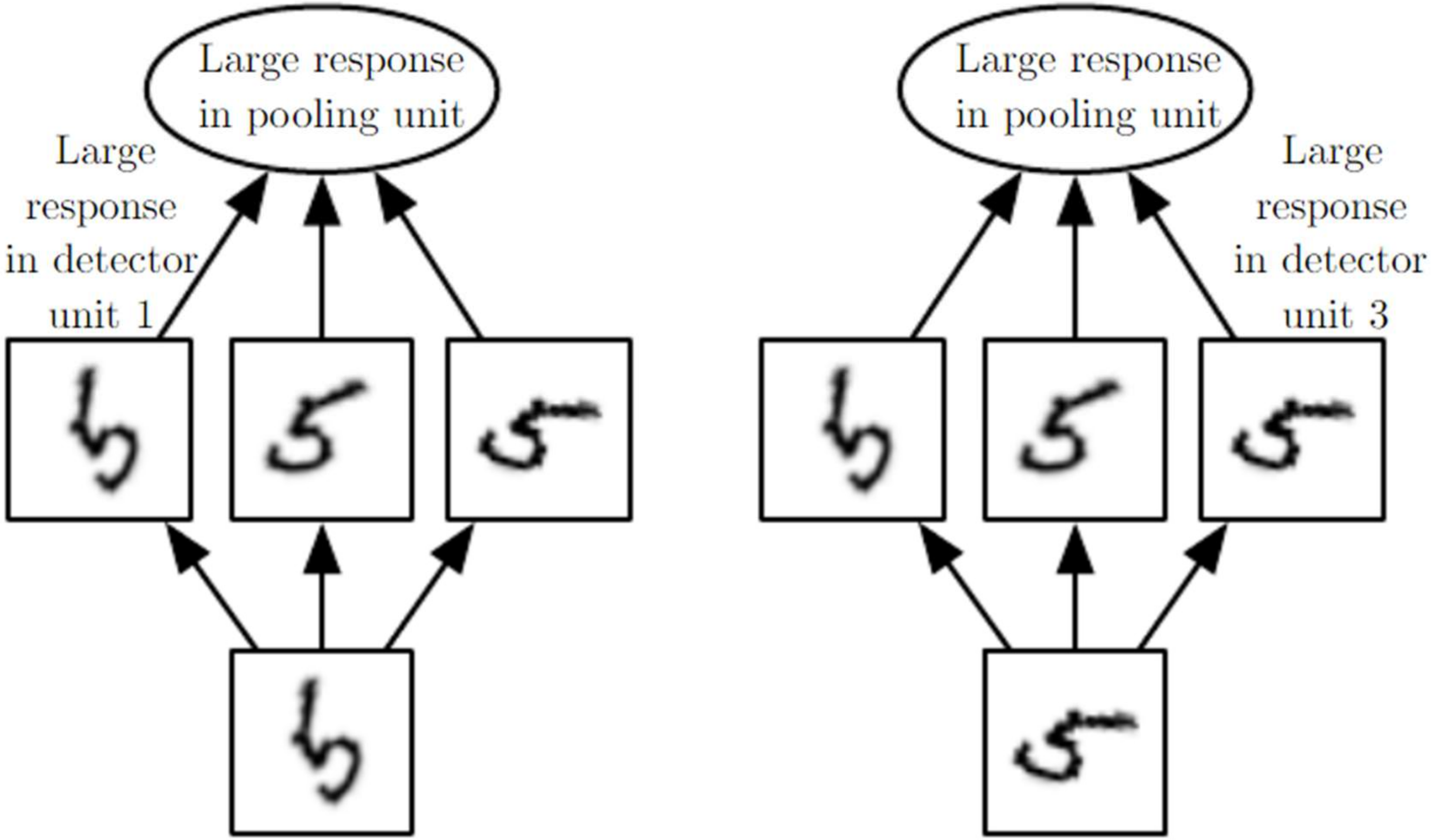
- inputs become invariant to small translations
- improves computational efficiency





Pooling

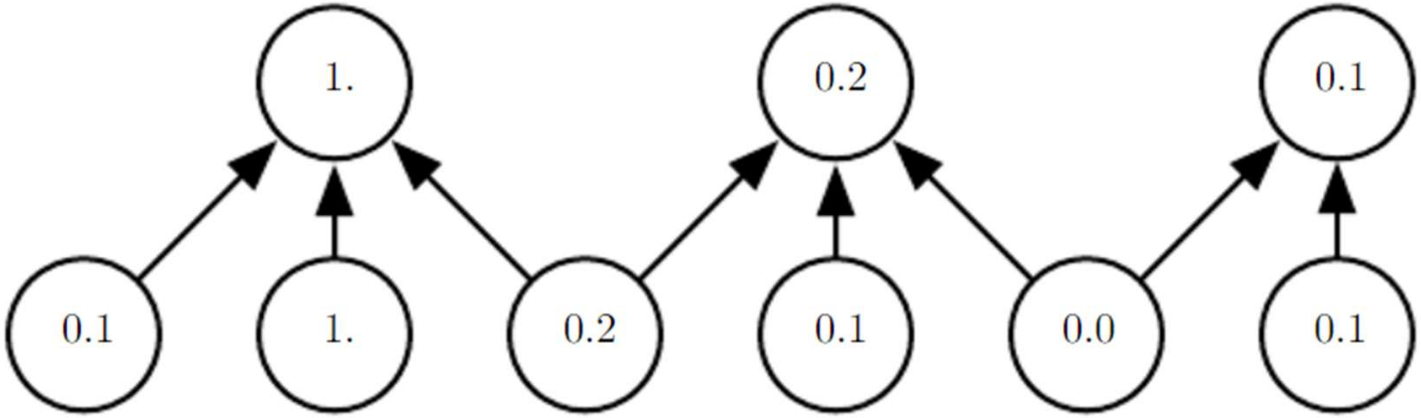
Example of learned invariances



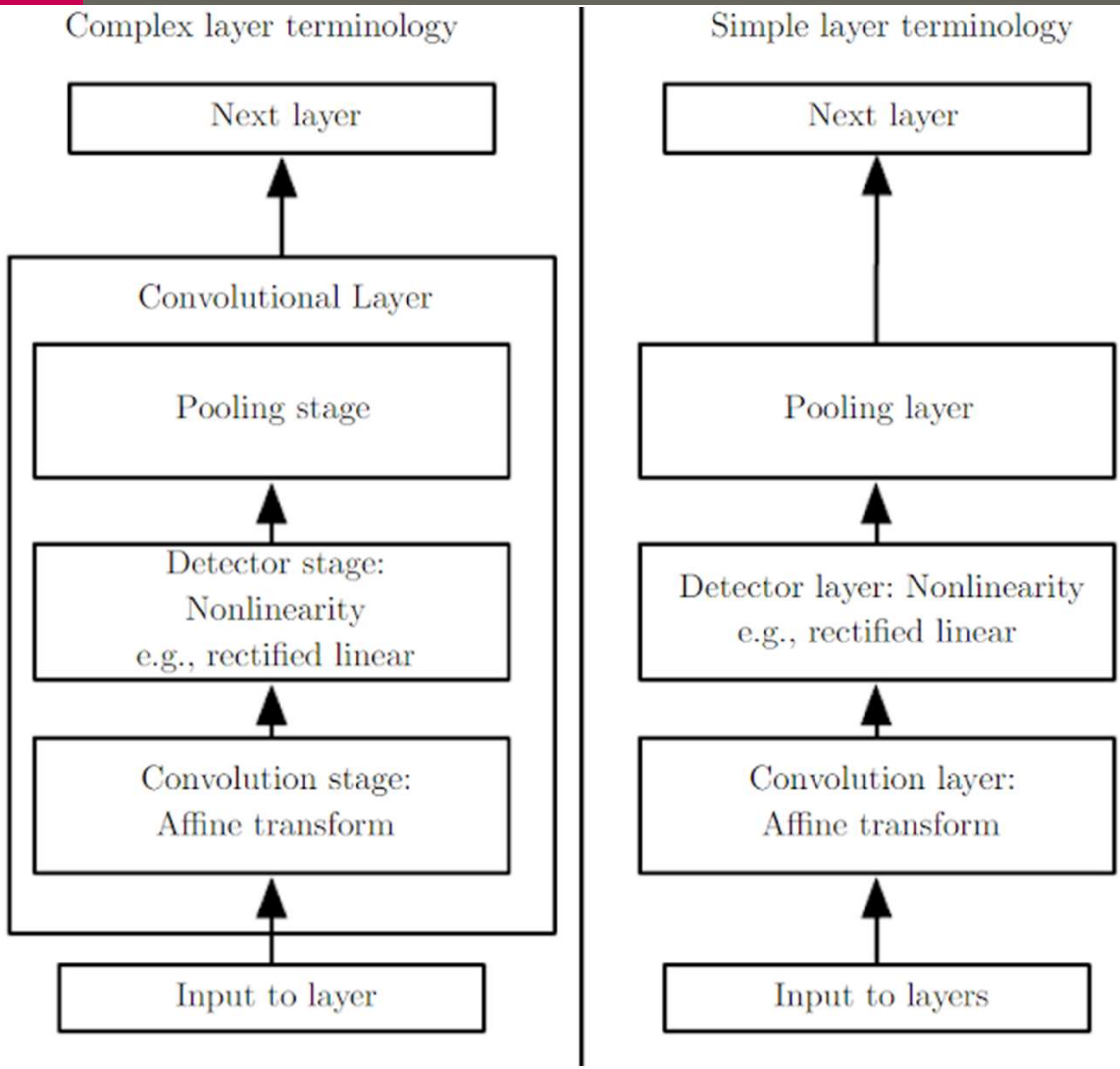
Pooling

Pooling with downsasmpling

- improve the statistical efficiency and reduce memory requirements

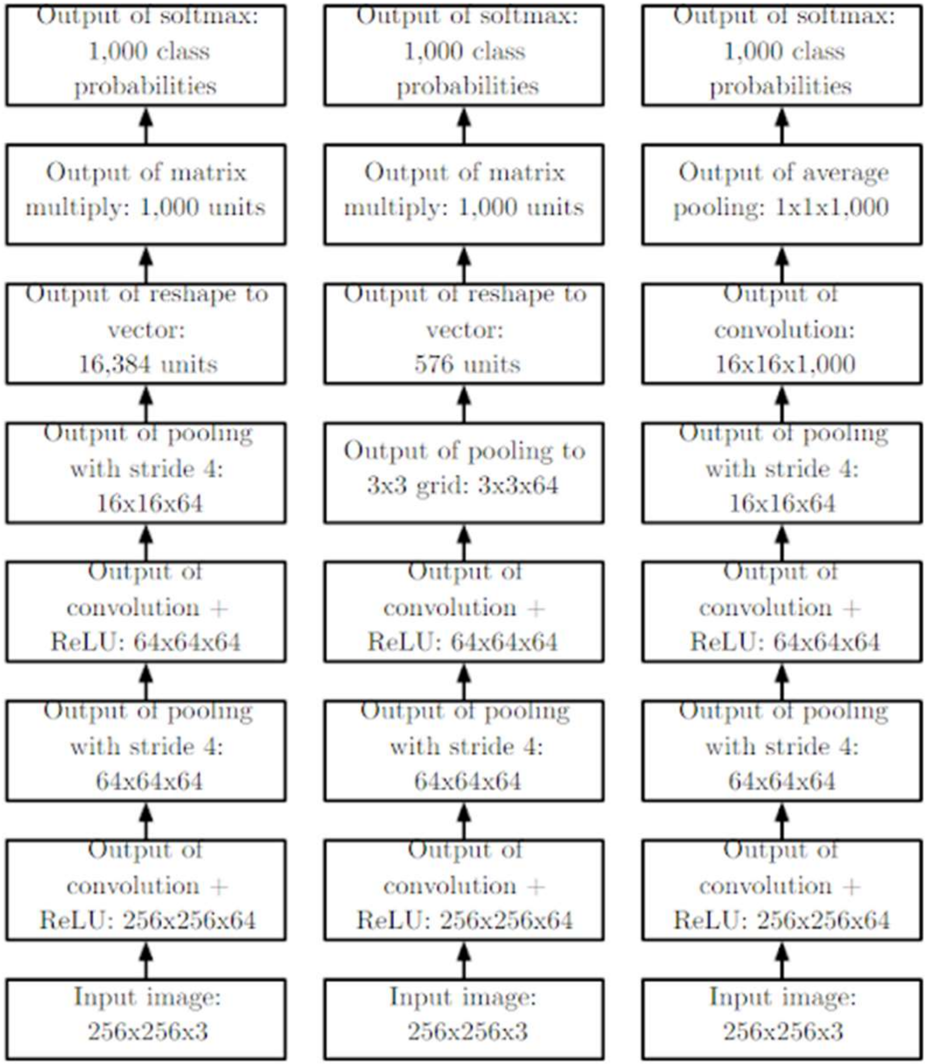


Examples



Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage:
Nonlinearity
e.g., rectified linear
Convolution stage:
Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity
e.g., rectified linear
Convolution layer:
Affine transform
Input to layers

Examples

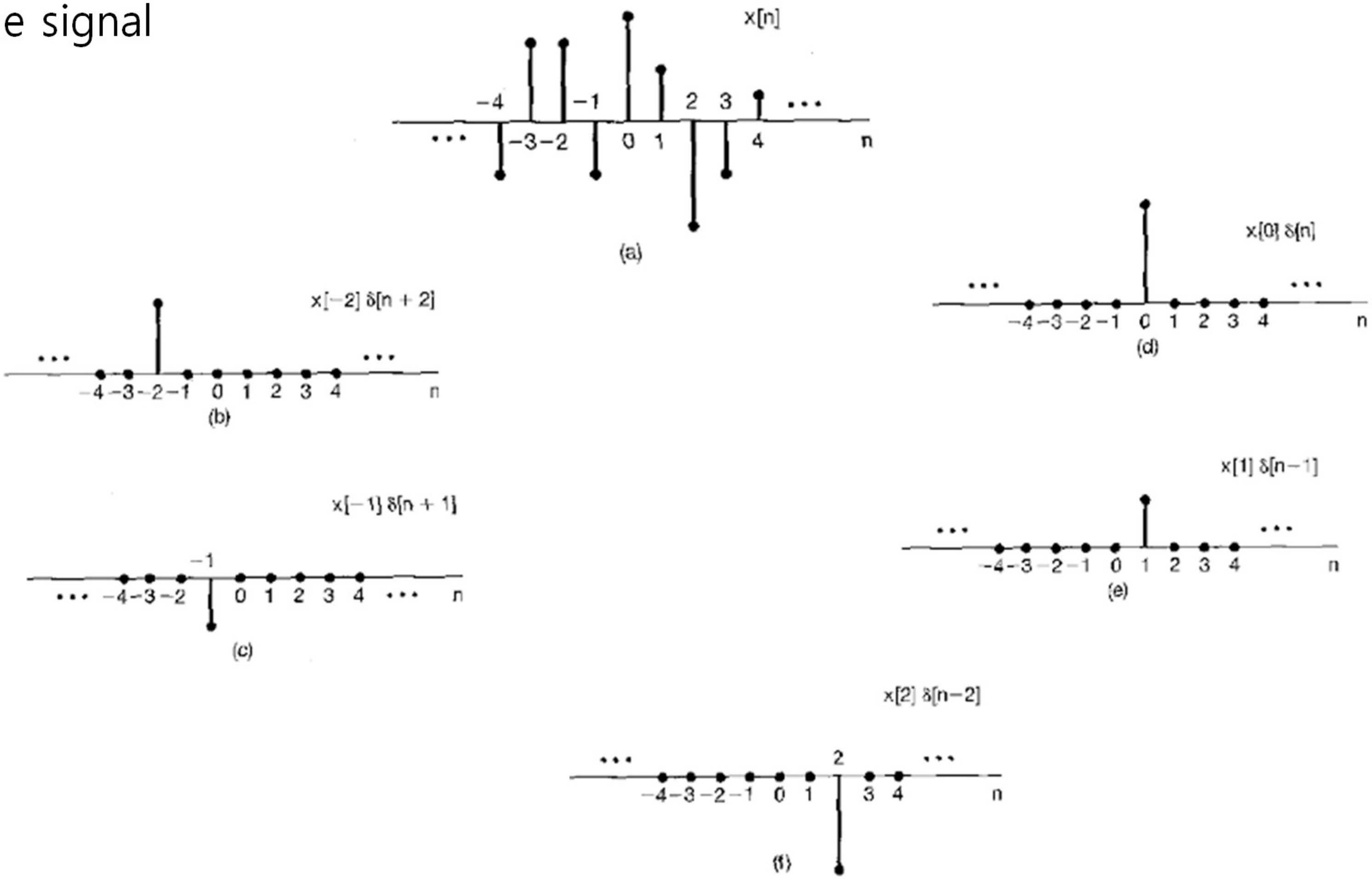

Output of softmax: 1,000 class probabilities
Output of matrix multiply: 1,000 units
Output of reshape to vector: 16,384 units
Output of pooling with stride 4: 16x16x64
Output of convolution + ReLU: 64x64x64
Output of pooling with stride 4: 64x64x64
Output of convolution + ReLU: 256x256x64
Input image: 256x256x3
Output of softmax: 1,000 class probabilities
Output of matrix multiply: 1,000 units
Output of reshape to vector: 576 units
Output of pooling to 3x3 grid: 3x3x64
Output of convolution + ReLU: 64x64x64
Output of pooling with stride 4: 64x64x64
Output of convolution + ReLU: 256x256x64
Input image: 256x256x3
Output of softmax: 1,000 class probabilities
Output of average pooling: 1x1x1,000
Output of convolution: 16x16x1,000
Output of pooling with stride 4: 16x16x64
Output of convolution + ReLU: 64x64x64
Output of pooling with stride 4: 64x64x64
Output of convolution + ReLU: 256x256x64
Input image: 256x256x3

Appendix

Convolution

Representation of Discrete-Time Signals in Terms of Impulses

Consider the signal



Representation of Discrete-Time Signals in Terms of Impulses

Thus, the signal can be represented by

$$x[n] = \ldots + x[-3]\delta[n+3] + x[-2]\delta[n+2] + x[-1]\delta[n+1] + x[0]\delta[n] + x[1]\delta[n-1] + x[2]\delta[n-2] + x[3]\delta[n-3] + \ldots.$$

Or more compactly,

$$x[n] = \sum_{k=-\infty}^{+\infty} x[k]\delta[n-k].$$

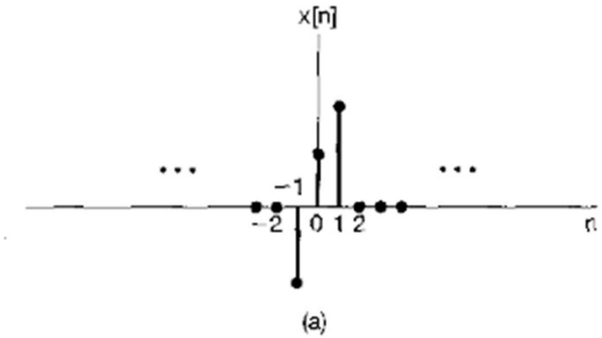
Therefore, an arbitrary discrete-time signal can be represented by a linear combination of shifted unit impulses $\delta[n-k]$, where the weights in this linear combination are $x[k]$

For example, the unit step function can be written by

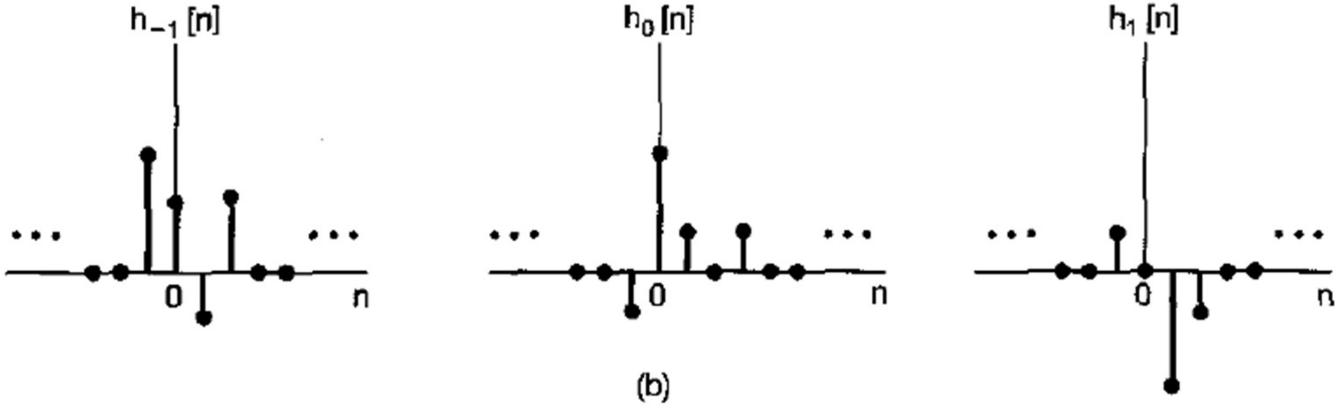
$$u[n] = \sum_{k=0}^{+\infty} \delta[n-k],$$

Convolution-Sum Response of Linear Systems

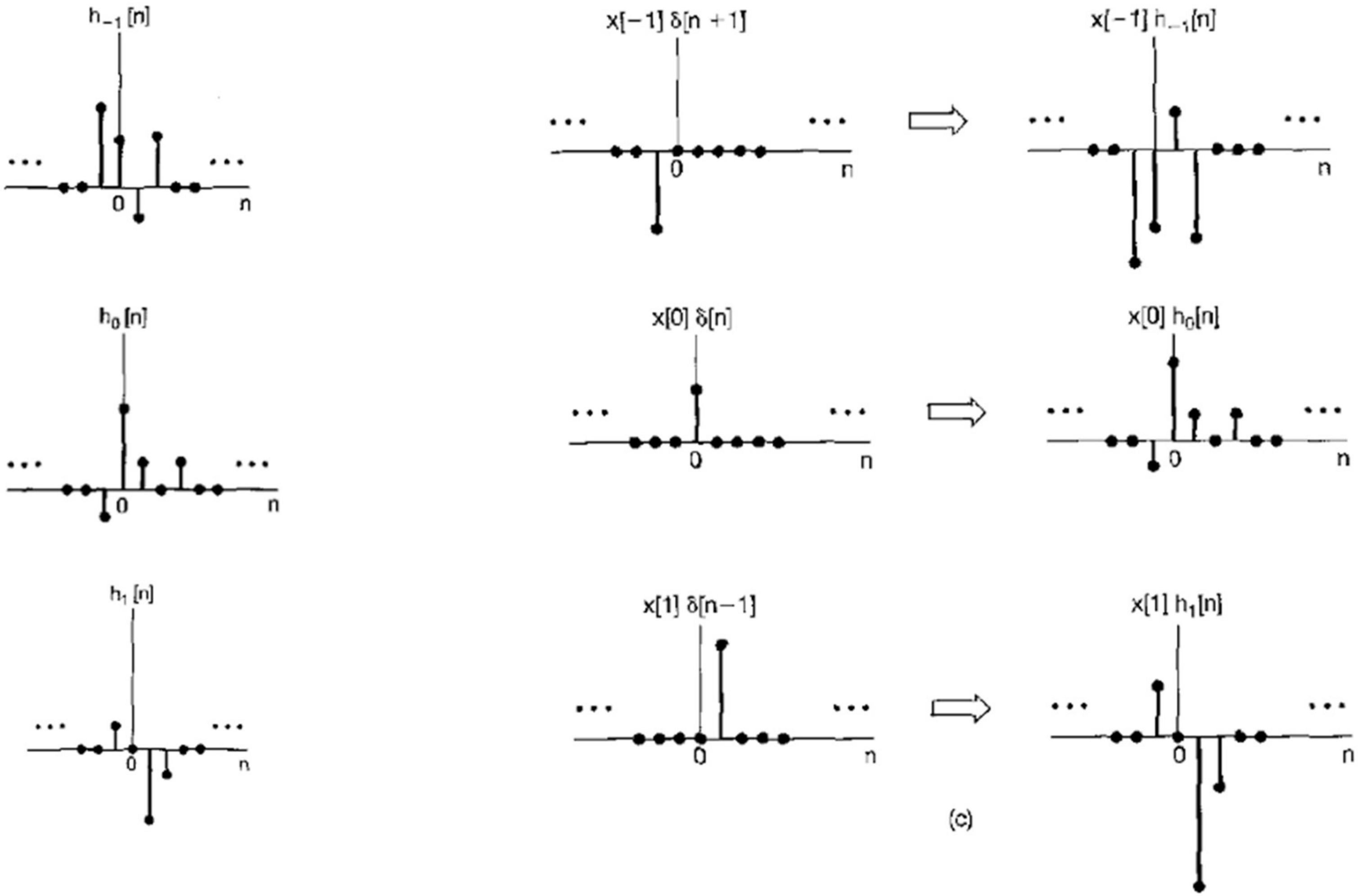
Consider an input



Let h_k $[n]$ denote the response of the linear system to the shifted unit impulse $\delta[n-k]$.

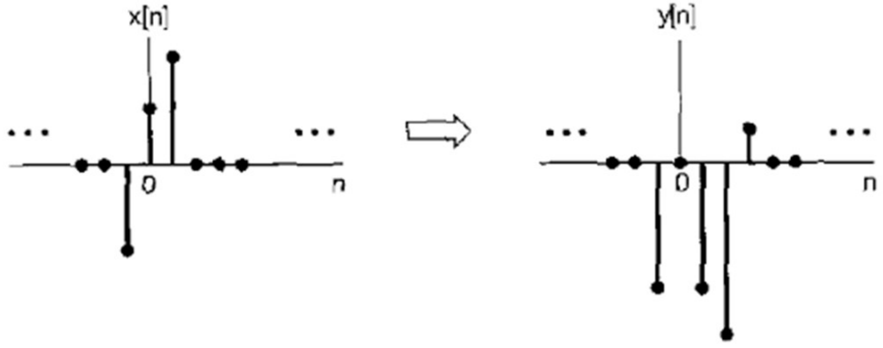


Convolution-Sum Response of Linear Systems



Convolution-Sum Response of Linear Systems

Then, the output can be expressed as



(d)

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h_k[n].$$

Thus, if we know the response of a linear system to the set of shifted unit impulses, we can construct the response to an arbitrary input

In time-invariant systems, h_k $[n]$ to time-shifted unit impulses are all time-shifted versions of each other

$$h_k[n] = h_0[n-k].$$

$$h[n] = h_0[n].$$

For an LTI system,

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h_k[n]. \longrightarrow \boxed{y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k].}$$

⇒ Convolution sum (superposition sum)

We will represent this by

$$y[n] = x[n] * h[n].$$