CSE 5526: Introduction to Neural Networks

Convolutional Networks

Motivation

- Convolutional neural networks (CNNs) are biologically-inspired variants of MLPs
- Motivated by properties of the visual cortex
  - A neuron in the visual cortex is sensitive only to stimuli in a local region of the visual field, called the receptive field of the neuron.
  - The visual cortex contains simple and complex neurons
    - Simple neurons act as local filters to detect features over the input space
    - Complex neurons integrate the outputs of a group of simple neurons, showing a level of invariance to feature locations
- Key idea: Replace layerwise matrix multiplication in MLP with convolution

Basic structure

- Three kinds of layers to build a CNN architecture
  - Convolutional layer
  - Pooling layer
  - Fully connected layer, like in conventional MLP
- Three operations
  - Convolution (correlation)
  - Max pooling
  - Rectified linear unit (ReLU) as activation function

Conventional matrix multiplication

- In a conventional MLP, the activity of a layer can be viewed as the matrix multiplication of the output of the previous layer and the weight matrix

$$\mathbf{v}^{(l)} = \mathbf{W}\mathbf{y}^{(l-1)}$$

- If layers are two-dimensional, **W** has a four-dimensional structure, called a tensor

Convolution operator

- The convolution (denoted by symbol $*$) between two functions $x(t)$ and $w(t)$ is defined

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

- In digital (discrete) representation

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

  - The first function, $x(t)$, is called the input, while the second is called a kernel
  - Note that the kernel is flipped relative to the input

Properties of convolution

- Convolution is commutative, i.e. $x(t) * w(t) = w(t) * x(t)$
- Convolution in time is equivalent to multiplication in frequency (via Fourier transform)
- A convolution operation is commonly called filtering
  - Widely used in signal processing and other engineering disciplines
  - In this case, kernel $w(t)$ is called a filter
  - Hence convolution is often referred to as feature extraction, and $s(t)$ is called a feature map

Convolution and correlation

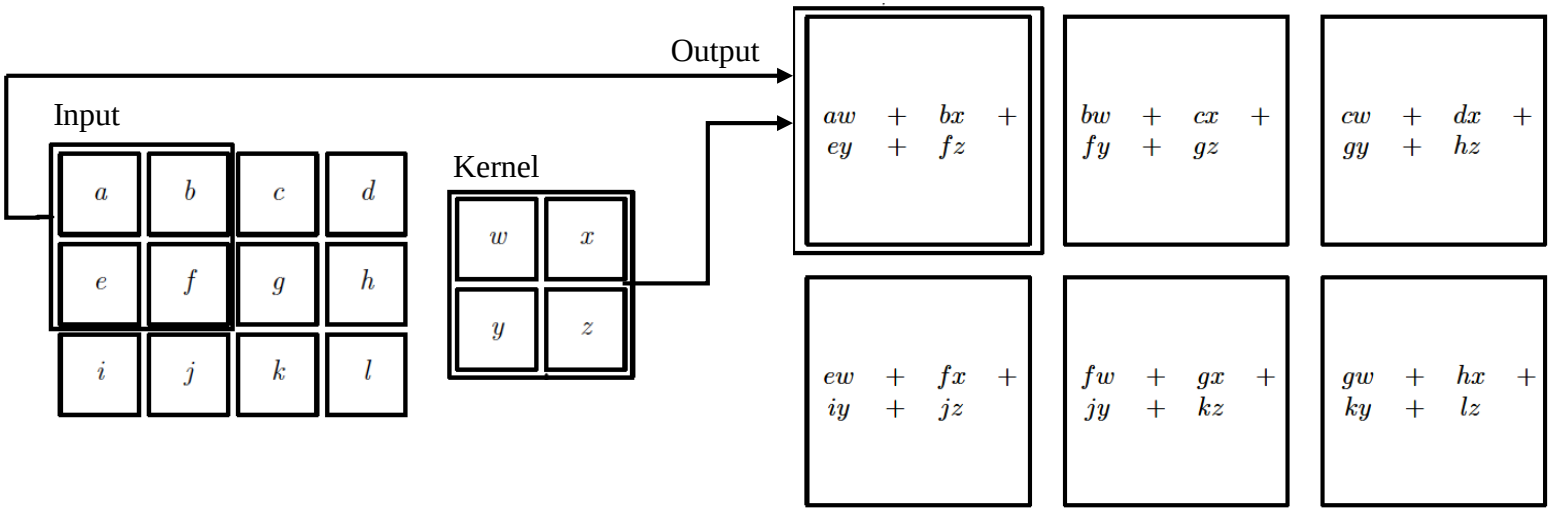
- Two-dimensional convolution with a 2-D kernel K:

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

- While kernel flipping is needed for commutativity, it is usually not important for neural network operations
- Thus, CNN uses (cross) correlation instead, which is more intuitive

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

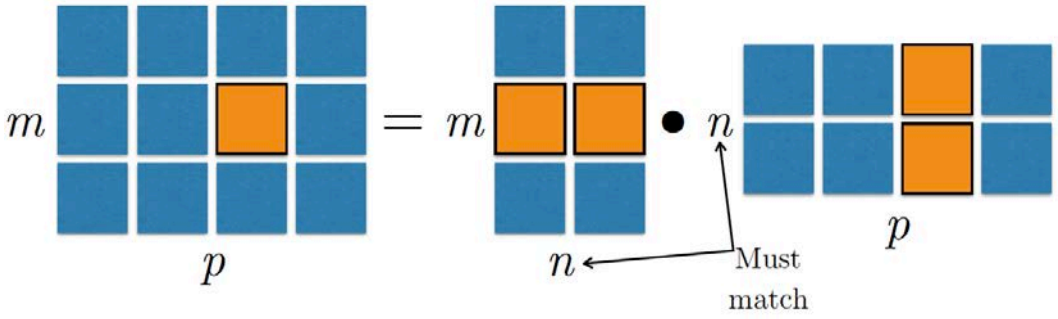
2-D convolution



- It resembles matrix multiplication (with a transpose)

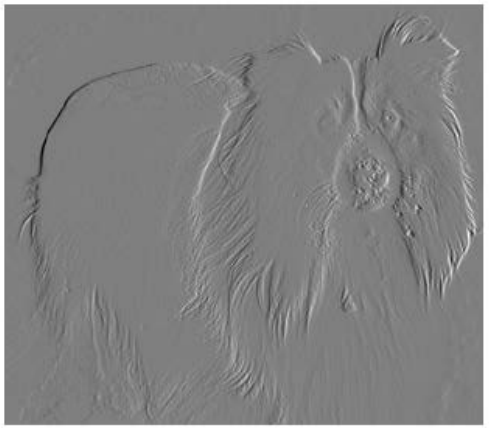
$$\boldsymbol{C} = \boldsymbol{AB}.$$

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}.$$



Example: edge detection by convolution

Input

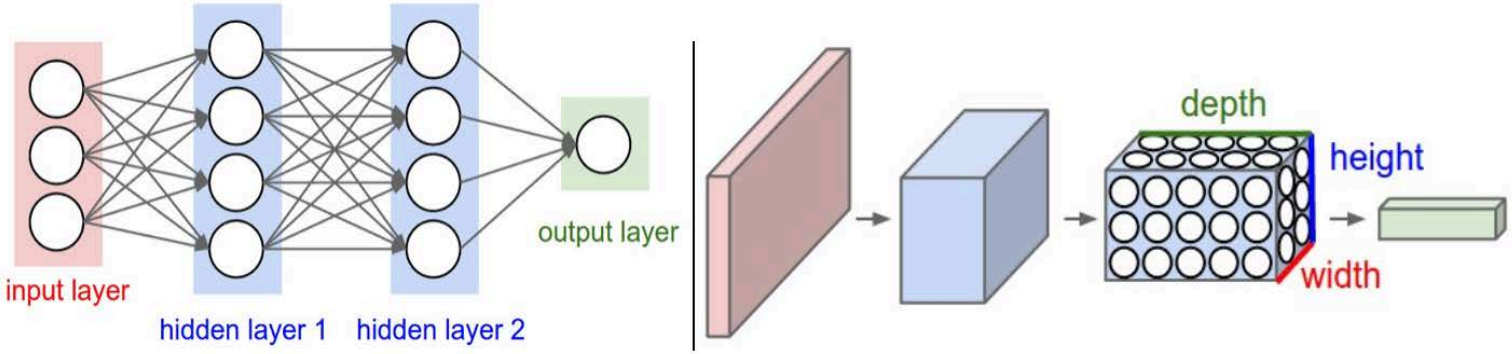


Output

1	-1

Kernel

From fully connected to convolutional networks



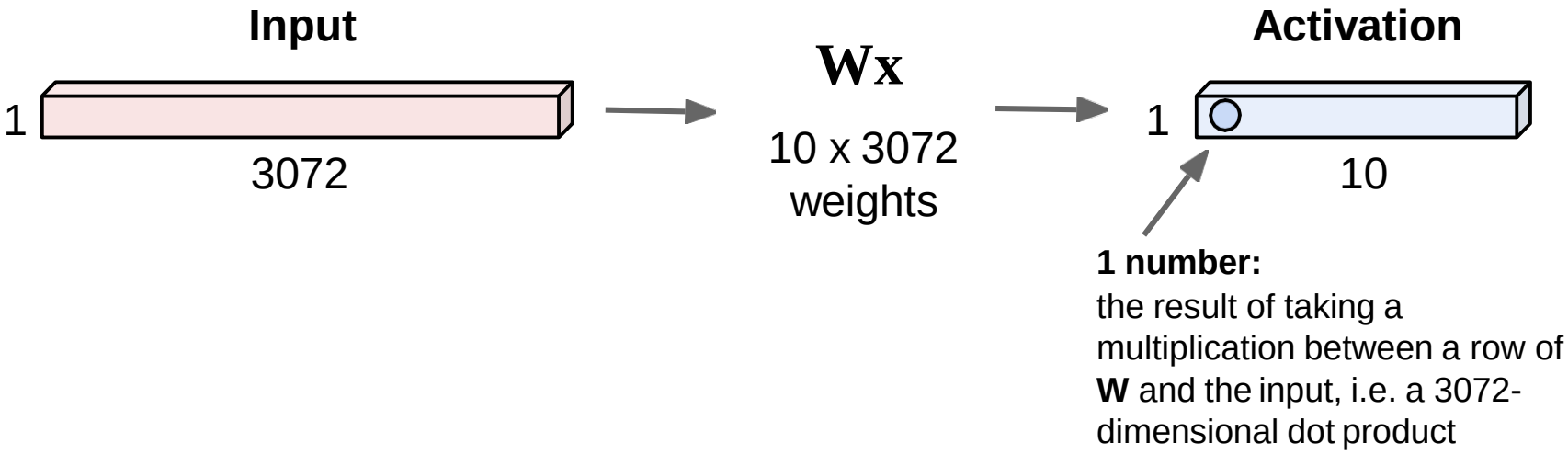
Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Fully connected layer

- An example: An input volume with dimensions 32x32x3 (width, height, and depth, respectively)

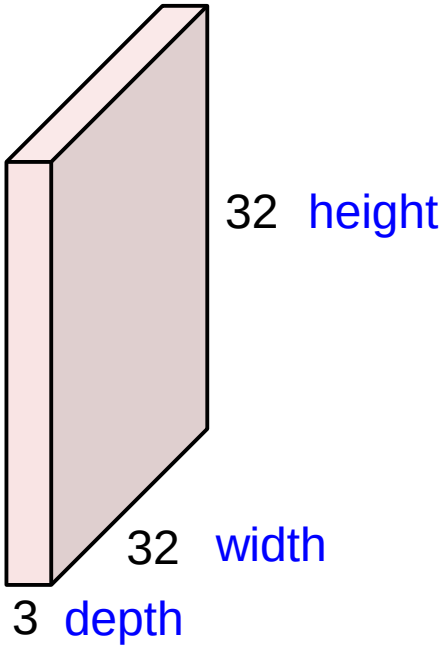
32x32x3 image: stretch to 3072 x 1

Each output neuron looks at the full input volume



Convolution layer

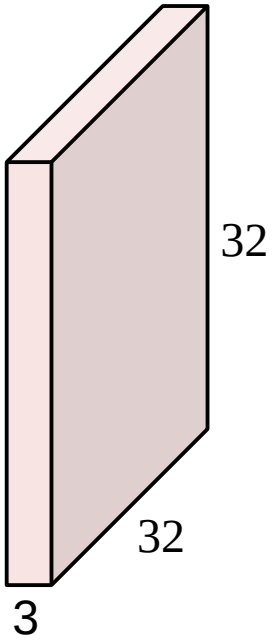
- 32x32x3 image, with spatial structure



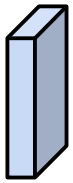
Convolution layer (cont.)

Filters extend the full depth of the input volume

- 32x32x3 image

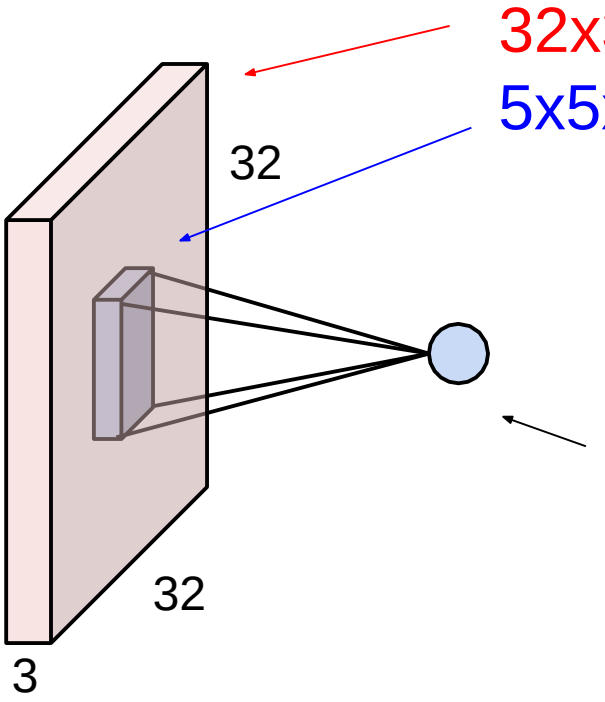


- 5x5x3 filter



- Convolve the filter with the image
- That is, “slide over the image spatially, computing dot products”

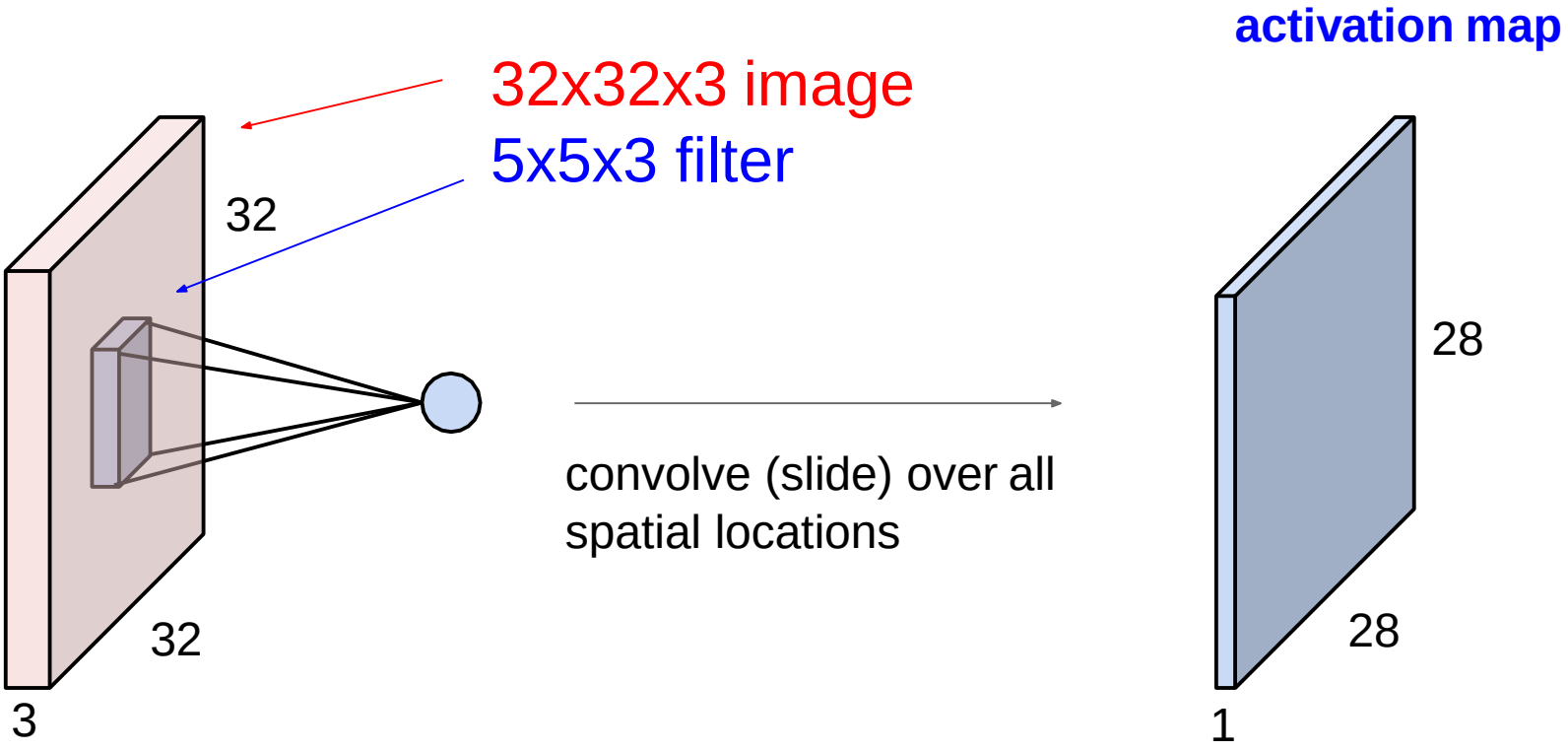
Convolution layer (cont.)



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5x5x3 = 75-dimensional dot product + bias)

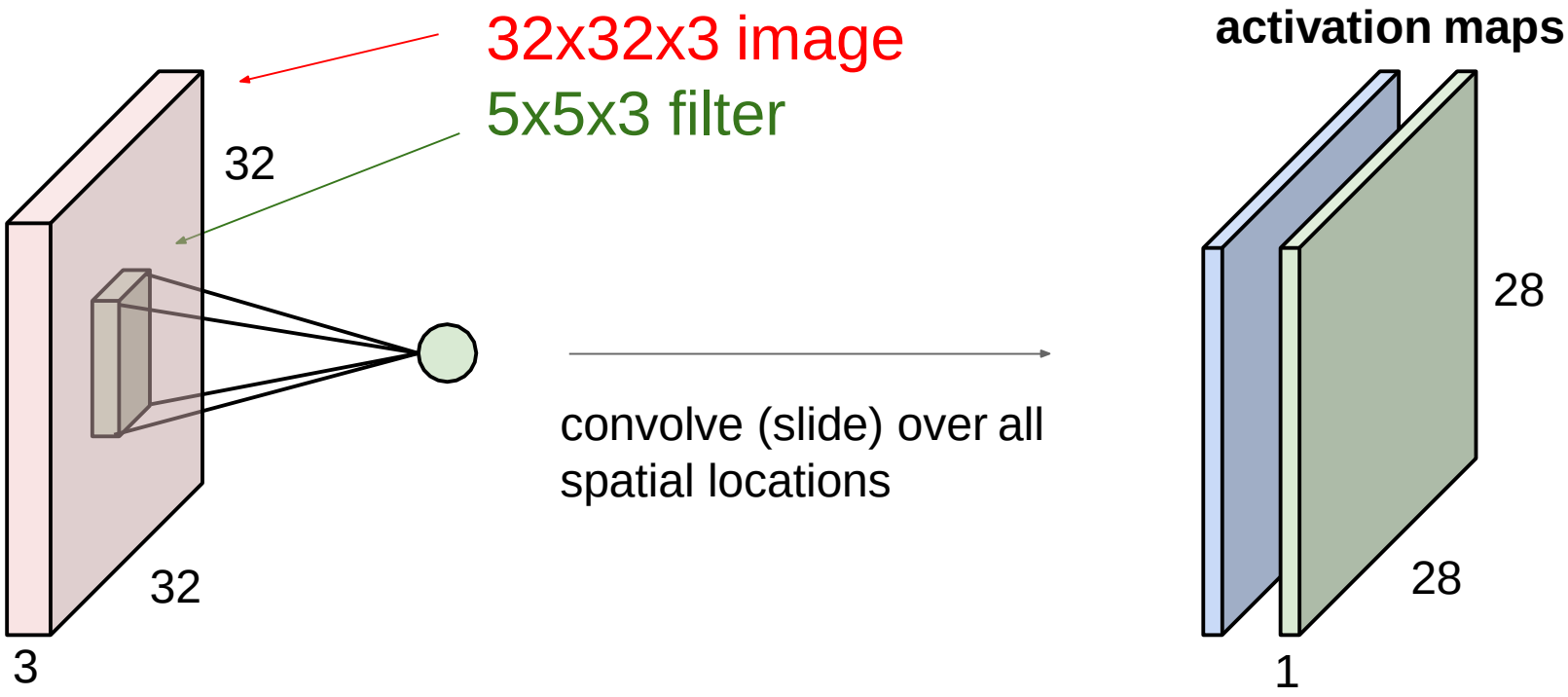
$$\mathbf{W}^T\mathbf{x} + b$$

Convolution layer (cont.)

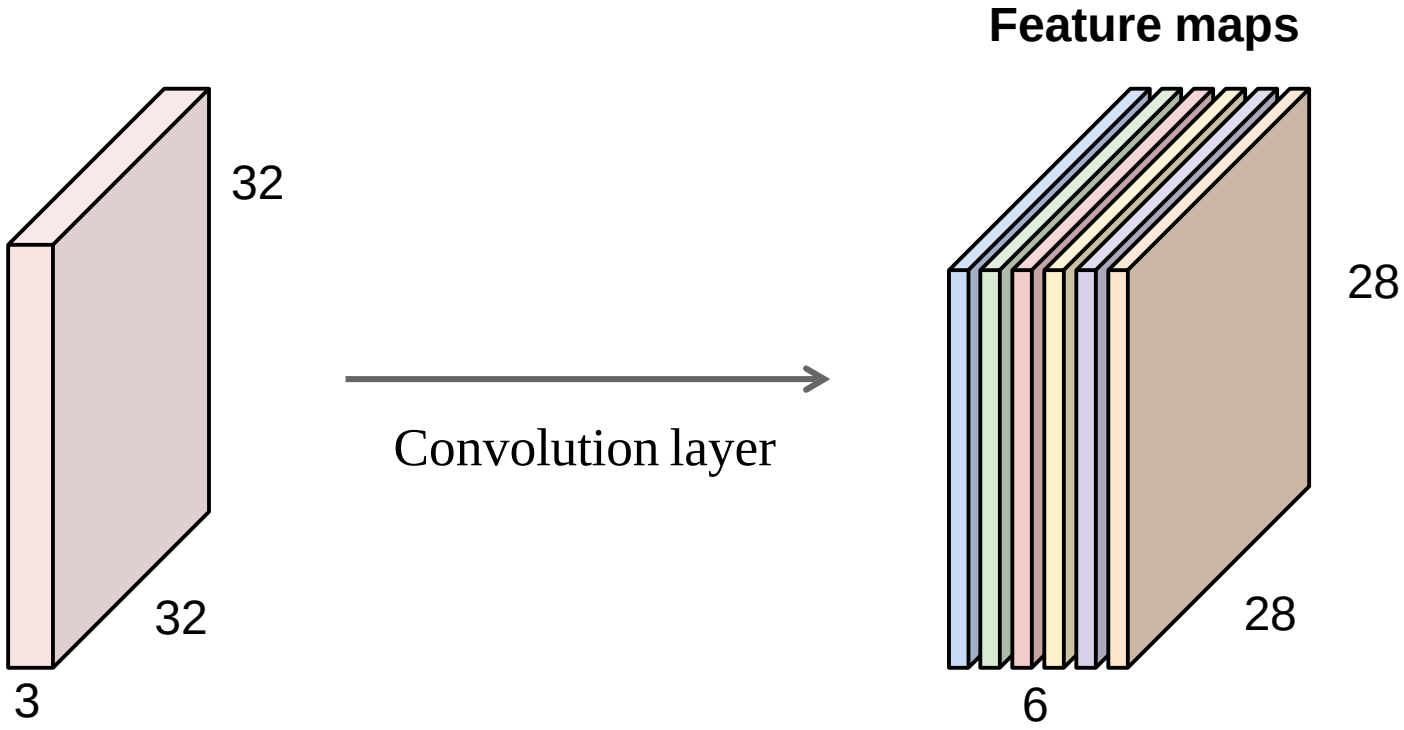


Convolution layer (cont.)

Consider a second, green filter

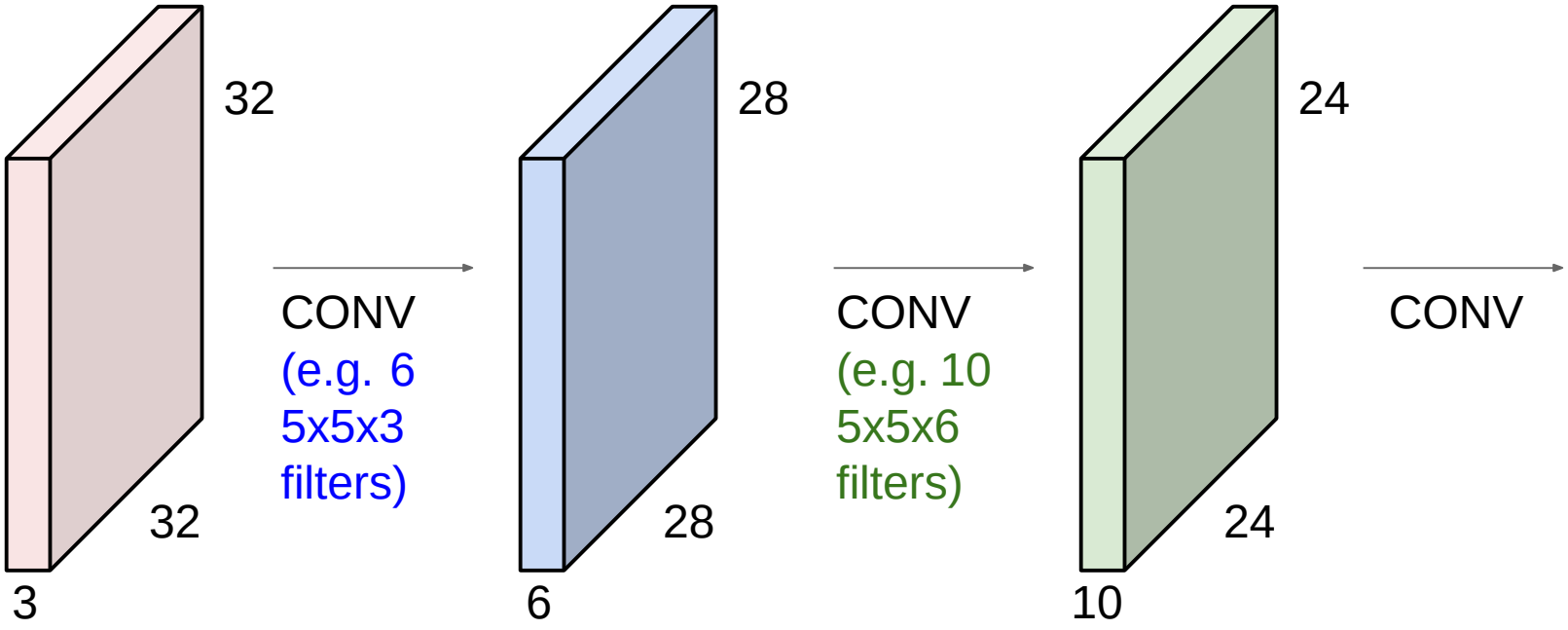


Convolution layer (cont.)



We stack these to get a new “image” of size 28x28x6

CNN

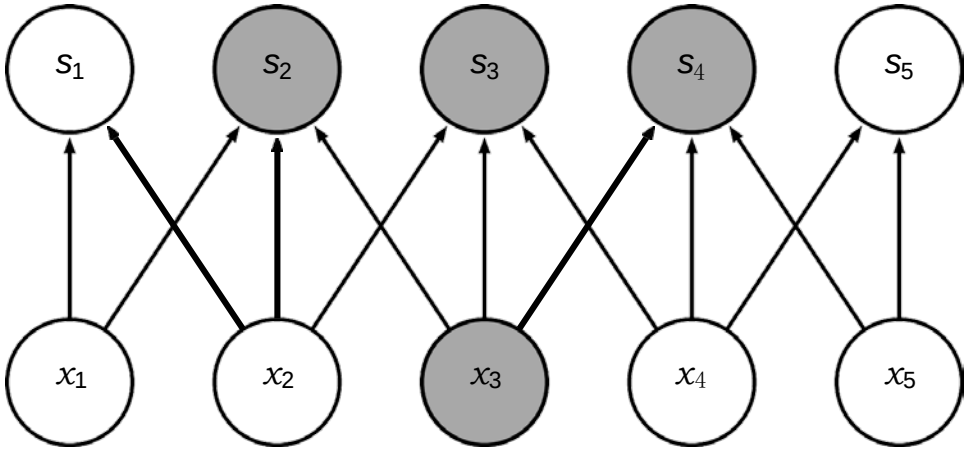

32
32
3
CONV
(e.g. 6
5x5x3
filters)
28
28
6
CONV
(e.g. 10
5x5x6
filters)
24
24
10
CONV

Understanding convolution

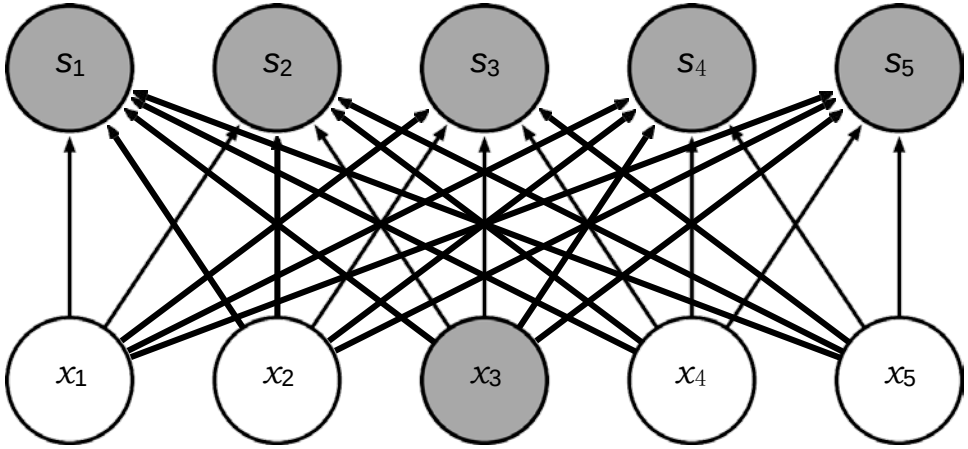
- Sparse connectivity because we create a feature from a local region in the input image that has the same size as the kernel (or filter)
  - To explore small regions of the image to find local features, such as edge and orientation
- Shared weights because we use the same filter across the entire image
  - Detect the same feature in various parts of the image in the same way

Sparse connectivity

Sparse connections due to small convolution kernel

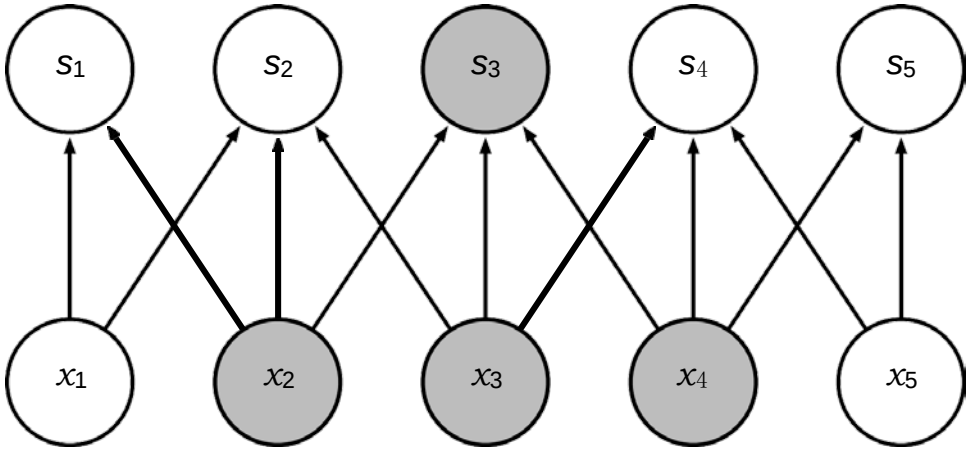


Dense connections

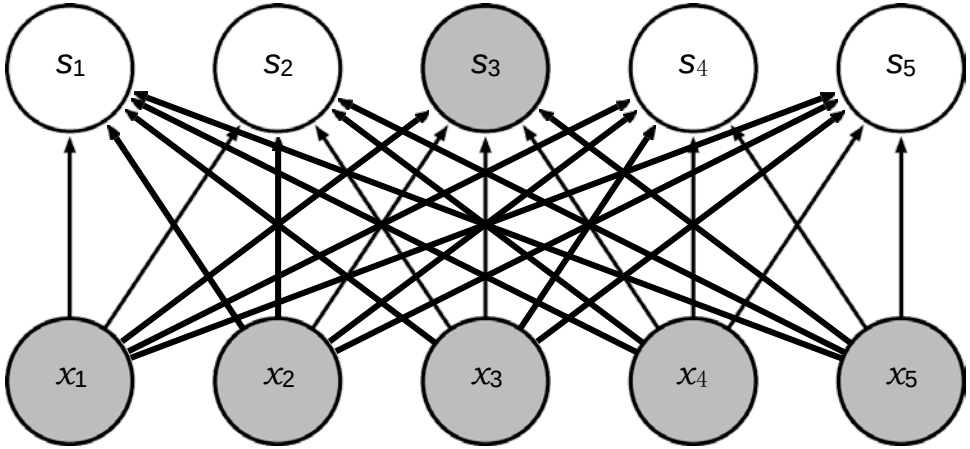


Sparse connectivity (cont.)

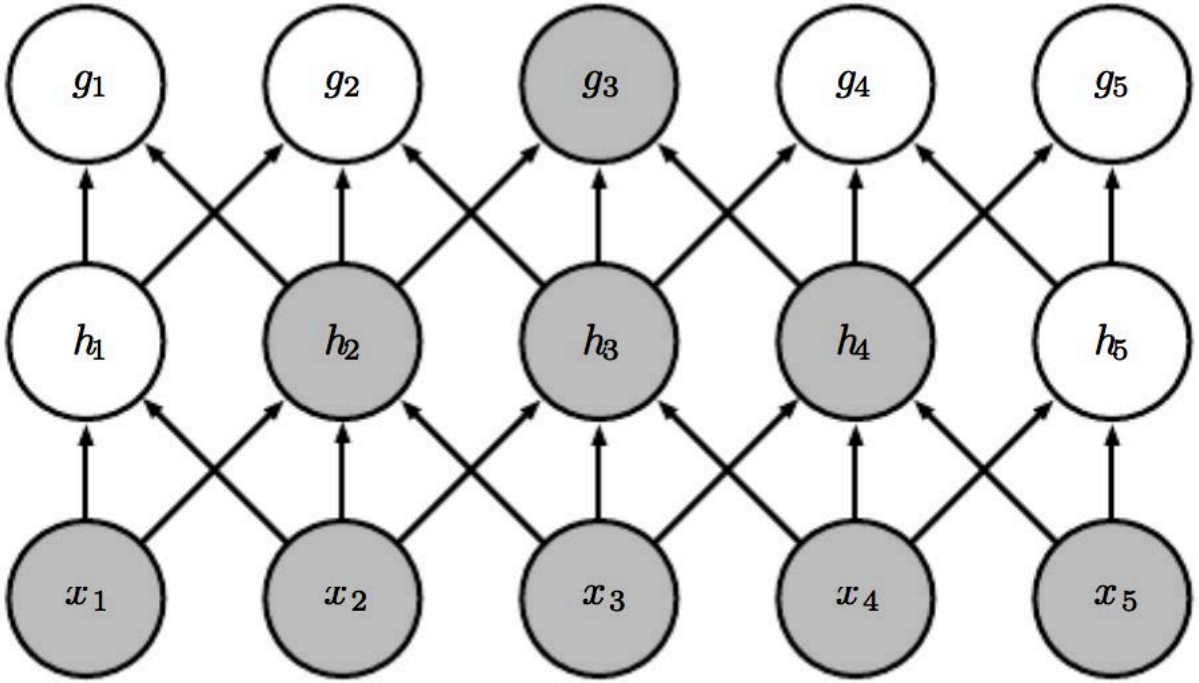
Sparse connections due to small convolution kernel



Dense connections



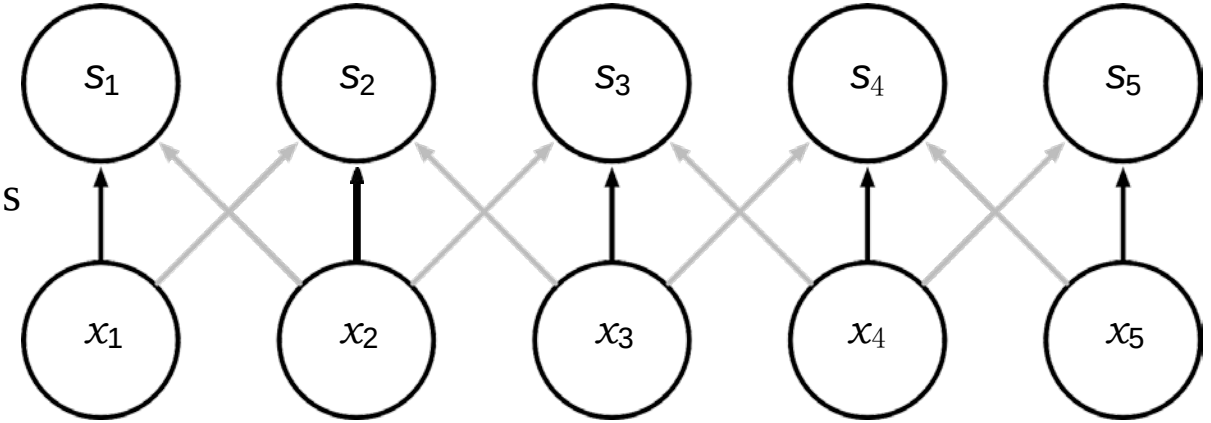
Growing receptive fields



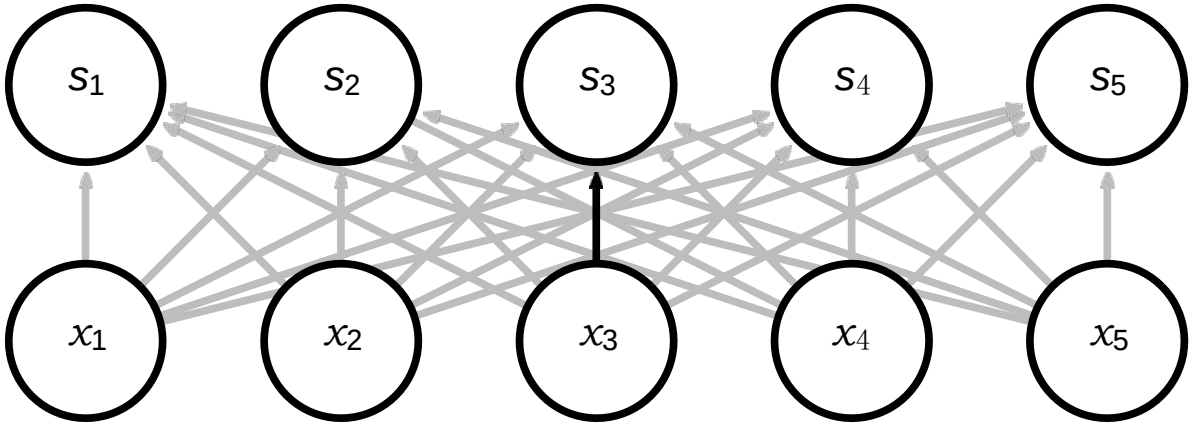
- The receptive fields of the units in deeper layers are larger than those in shallow layers
- In other words, even though direct connections between consecutive layers are sparse, units in deeper layers can be indirectly connected to large parts of the input image

Parameter sharing

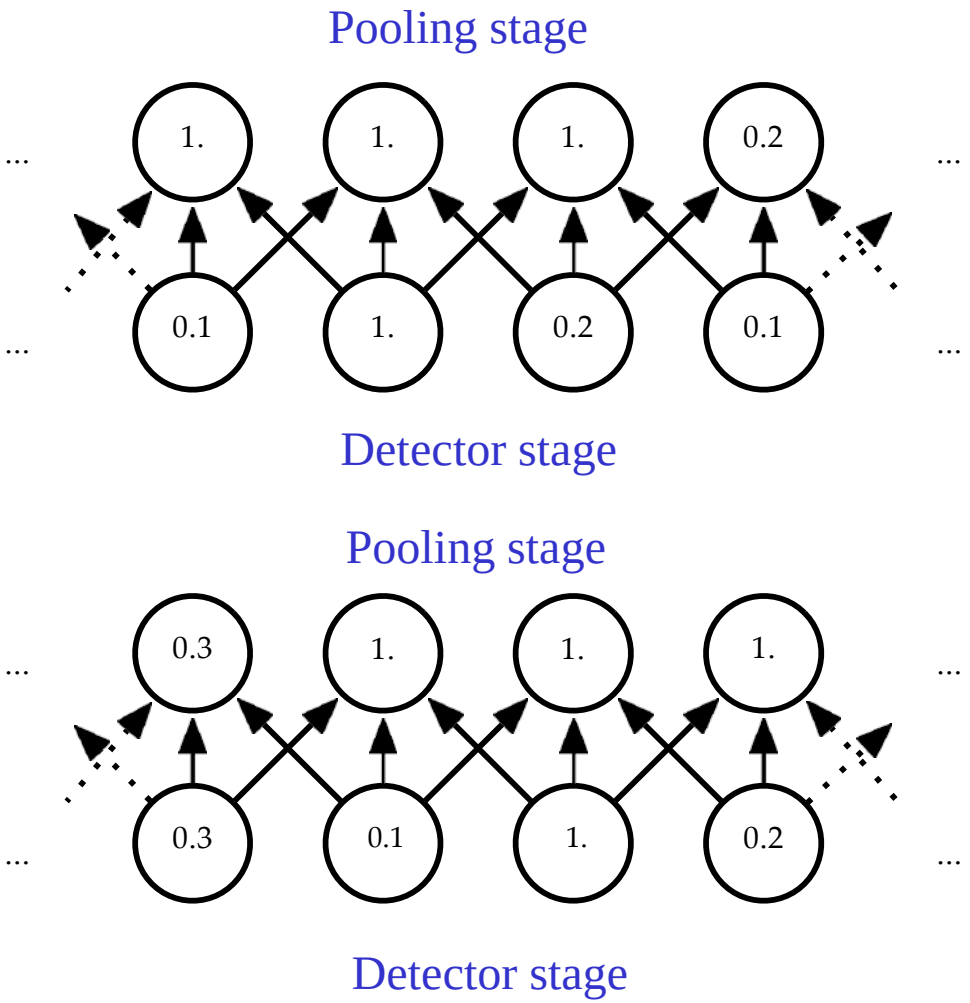
Convolution shares the same parameters across all spatial locations



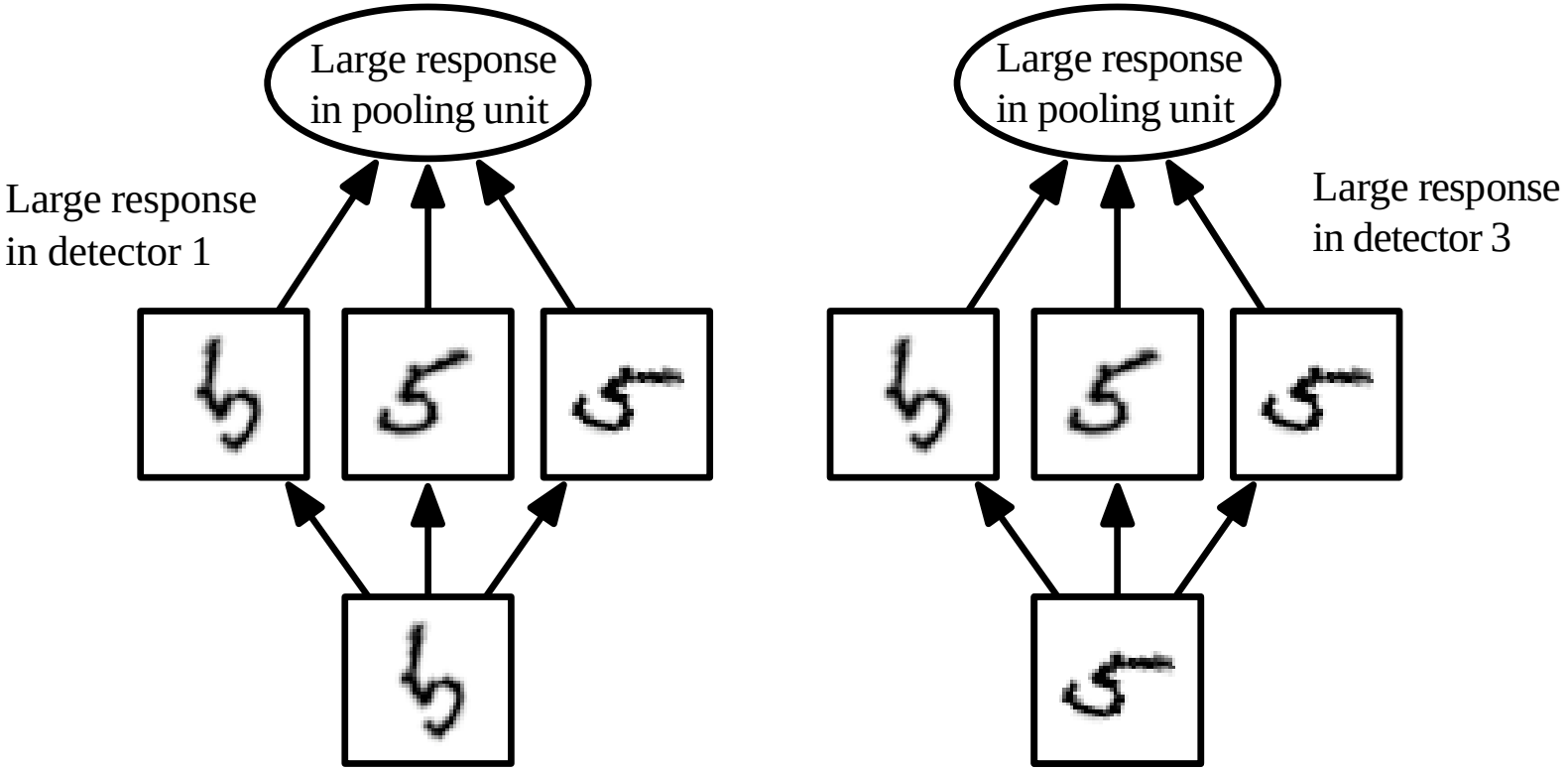
Traditional matrix multiplication does not share any parameters



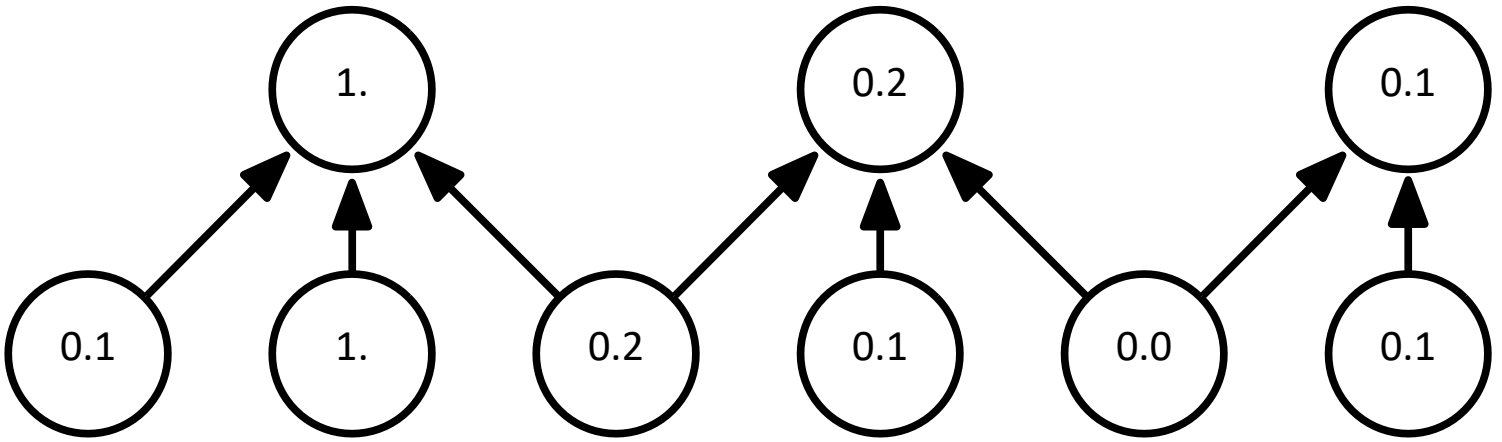
Max pooling and invariance to translation



Cross channel (filter) pooling and invariance to learned detectors

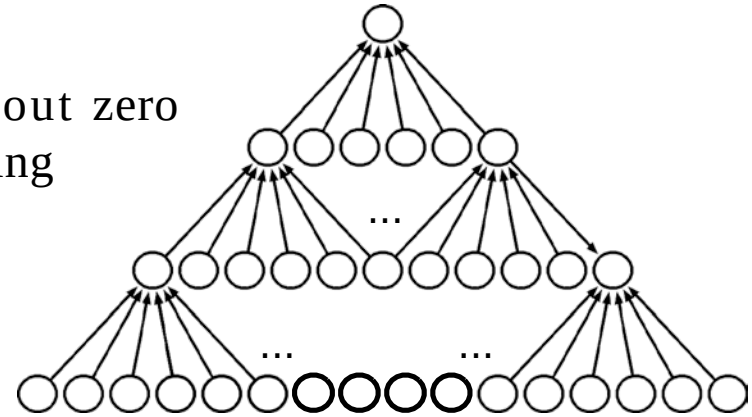


Pooling with downsampling

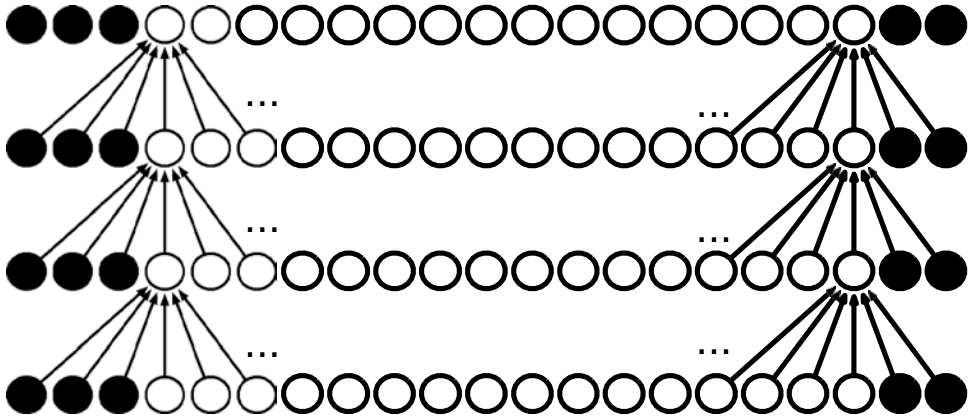


Zero padding controls network size

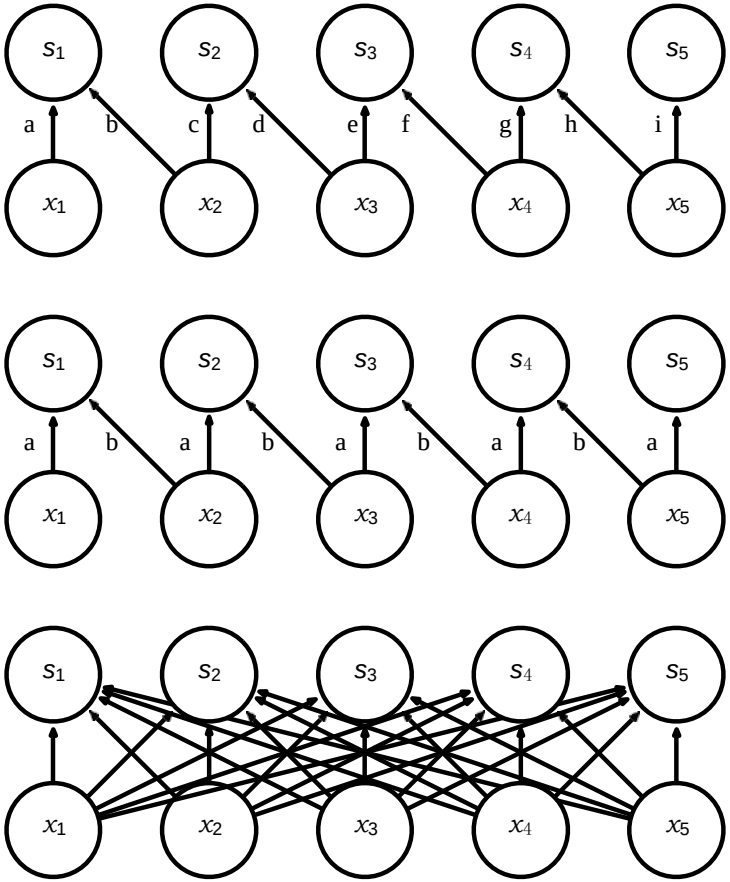
Without zero padding



With zero padding



Kinds of connectivity

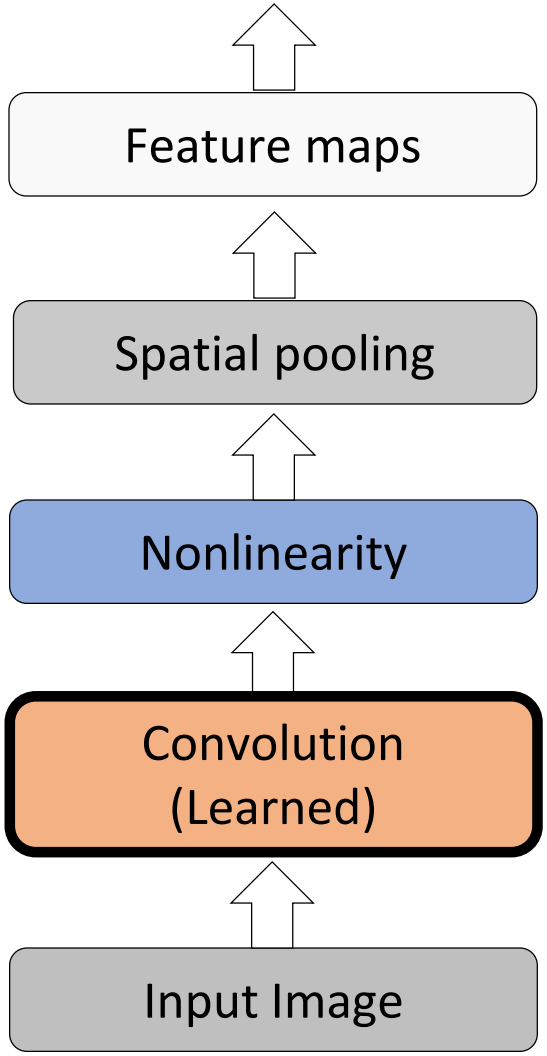


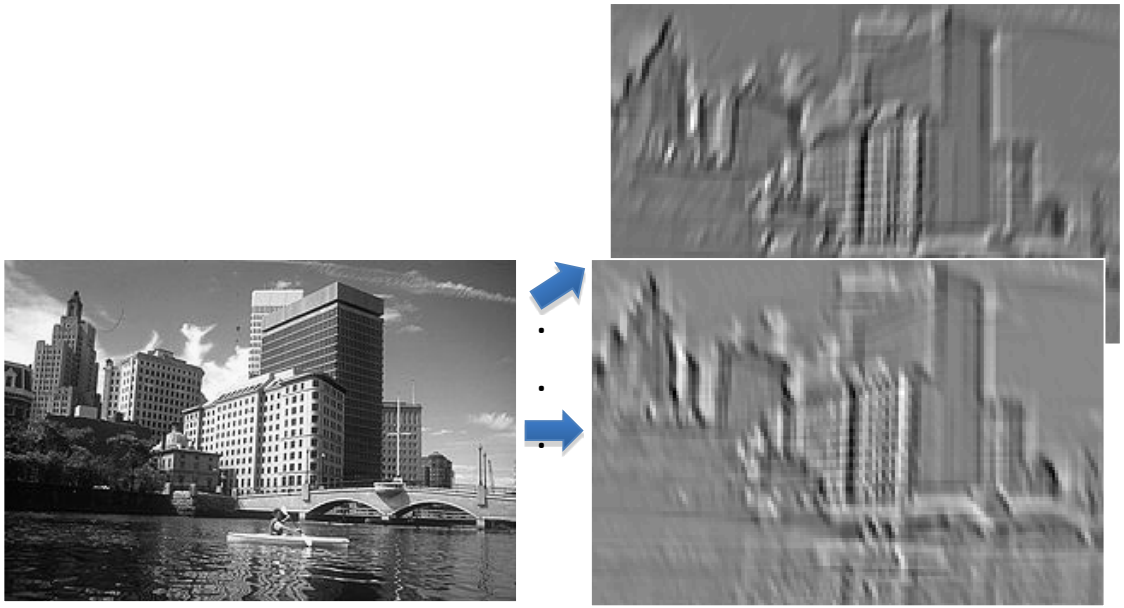
Local connection (like convolution but no sharing)

Convolution

Full connection

Typical CNN operations

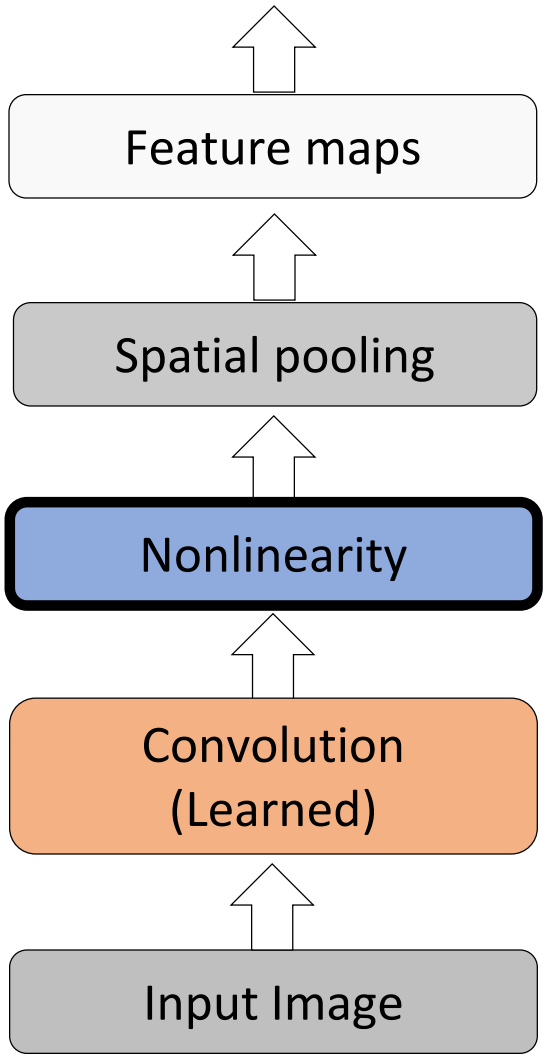




Input

Feature maps

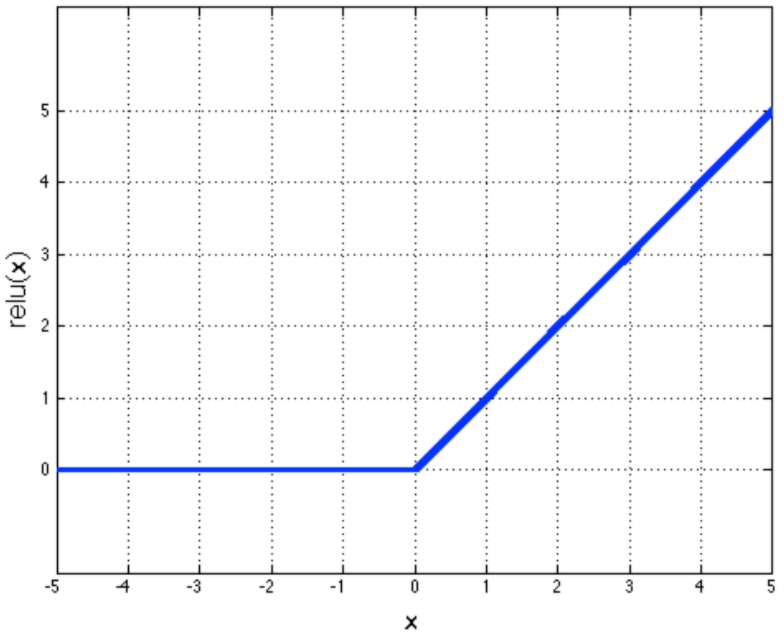
Typical CNN operations



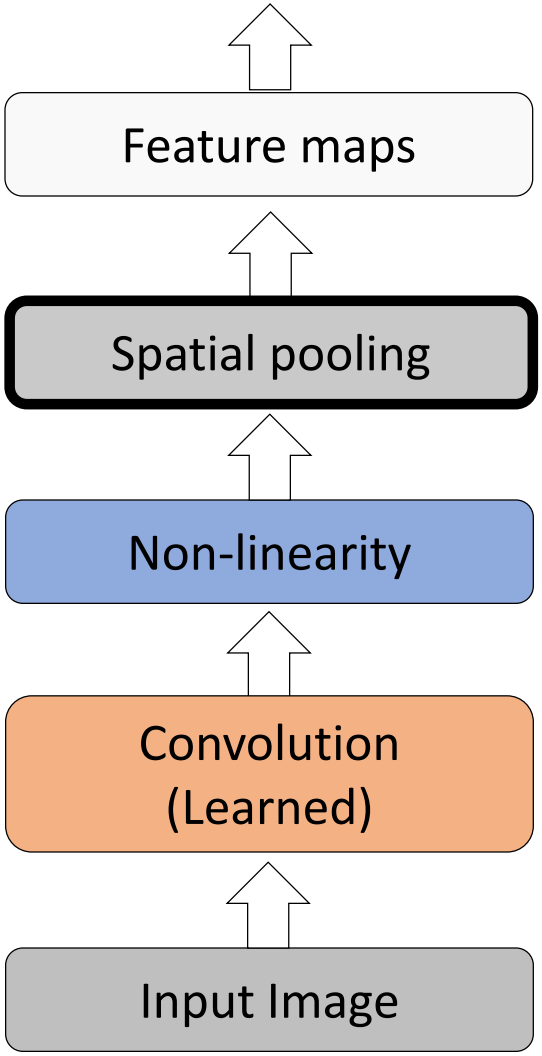
- Modern activation function or nonlinearity: Rectified Linear Unit (ReLU)

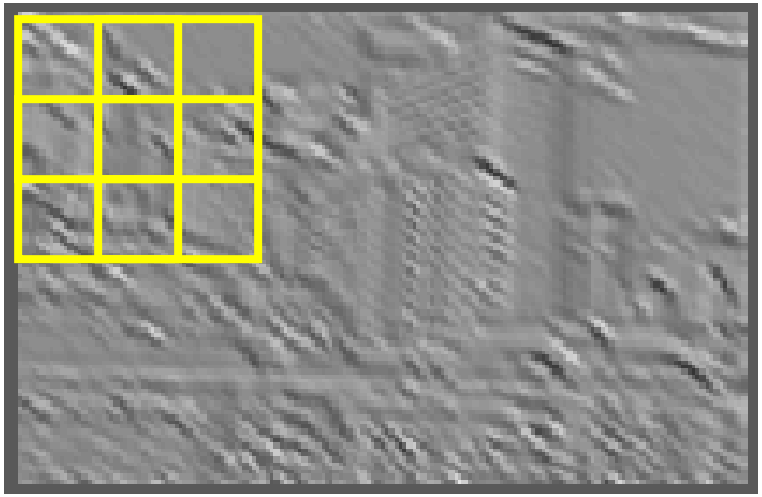
$$ReLU(x) = \begin{cases} 0 & if\ x < 0 \\ x & if\ x \geq 0 \end{cases}$$

Rectified Linear Unit

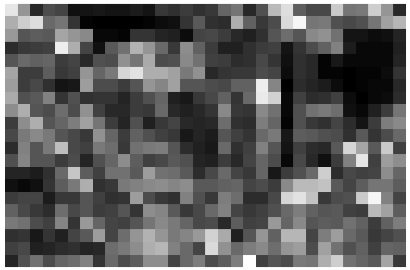


Typical CNN operations

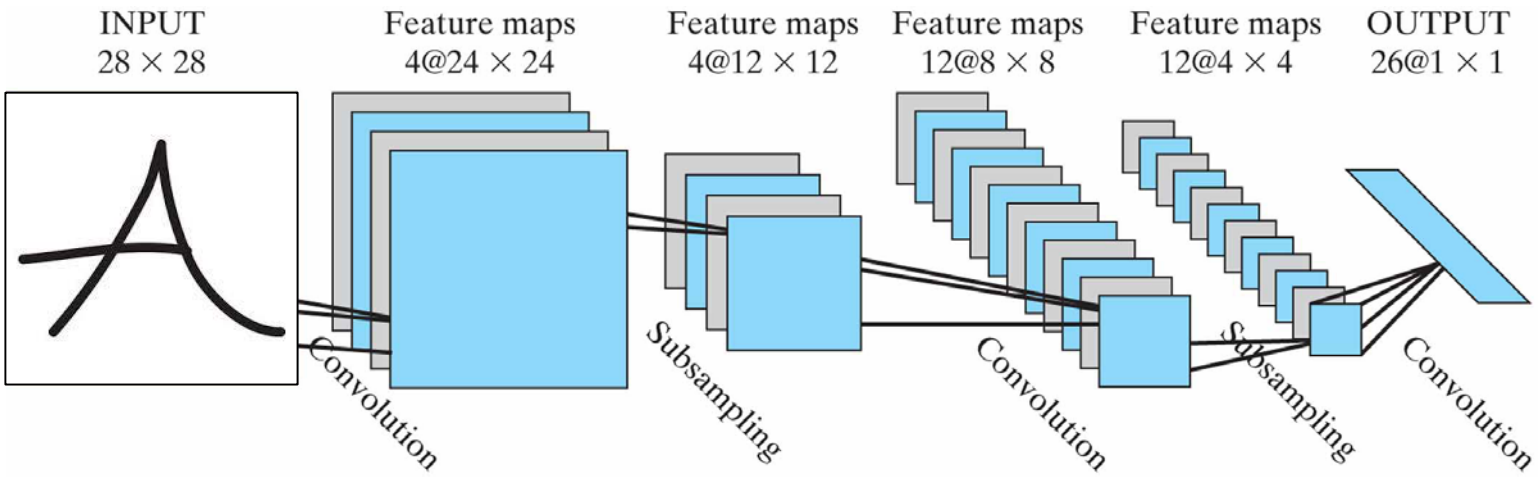




Max



Example: CNN for letter recognition



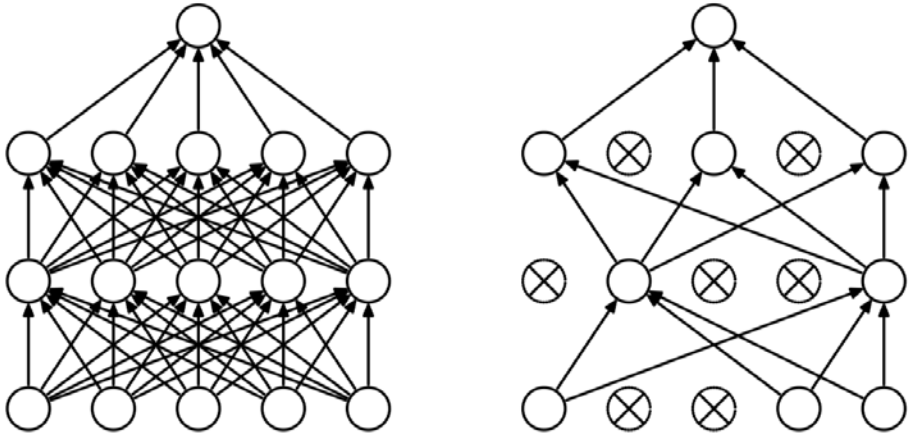
An early CNN for handwritten letter recognition

Backpropagation in CNN

- Three operations: convolution, max-pooling, and ReLU
- ReLU backpropagation is the same as any other network
  - Passes gradients to a previous layer only if the original input value is positive
- Max-pooling passes the gradient flow through the neuron with the largest response in the input volume
- Main complexity is in backpropagation through convolutions
  - Backpropagation through convolutions amounts to convolution with inverted filters (kernels)

Dropout

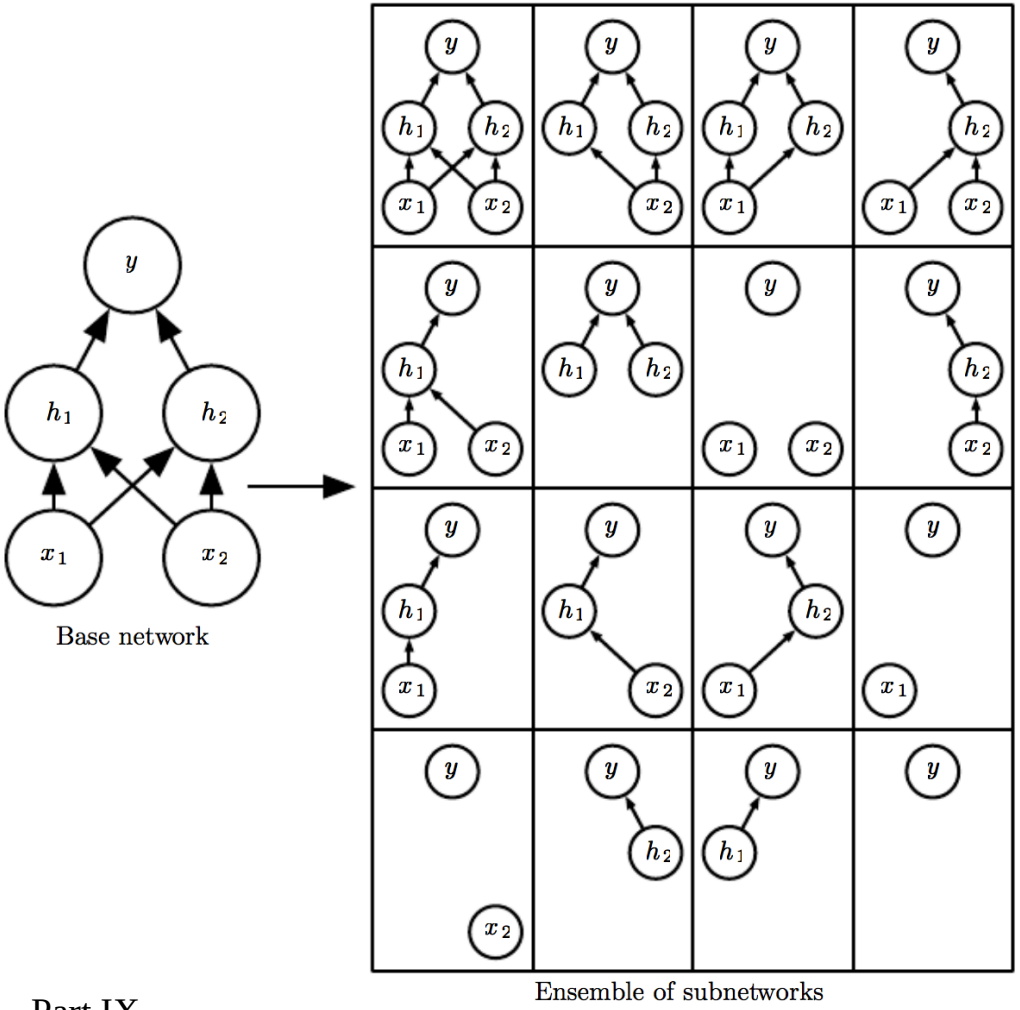
- Dropout is a regularization technique for reducing overfitting in neural network training
- The term “dropout” refers to dropping out randomly chosen units in a layer (both hidden and visible)
- Applying dropout amounts to sampling a “thinned” network



(a) Standard Neural Net (b) After applying dropout.

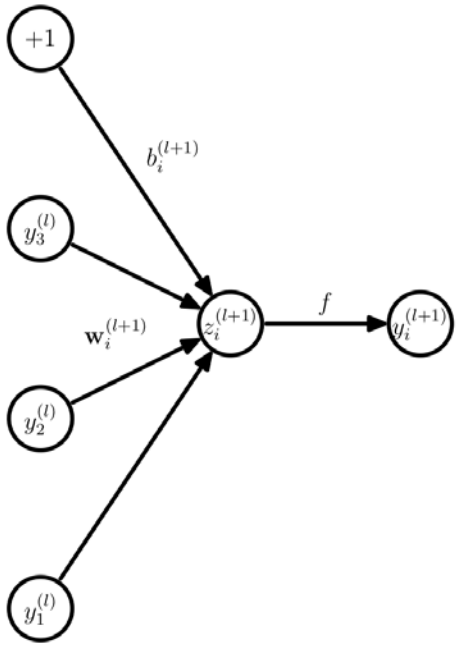
Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Dropout (cont.)

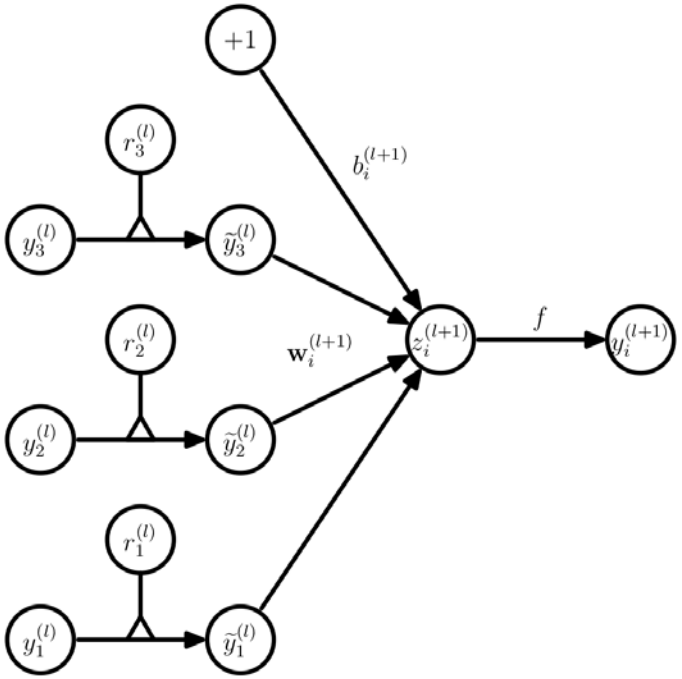


- We begin with a base network with two visible units and two hidden units
- There are sixteen possible subsets that may be formed by dropping out different subsets of units from the original network

Dropout (cont.)



(a) Standard network



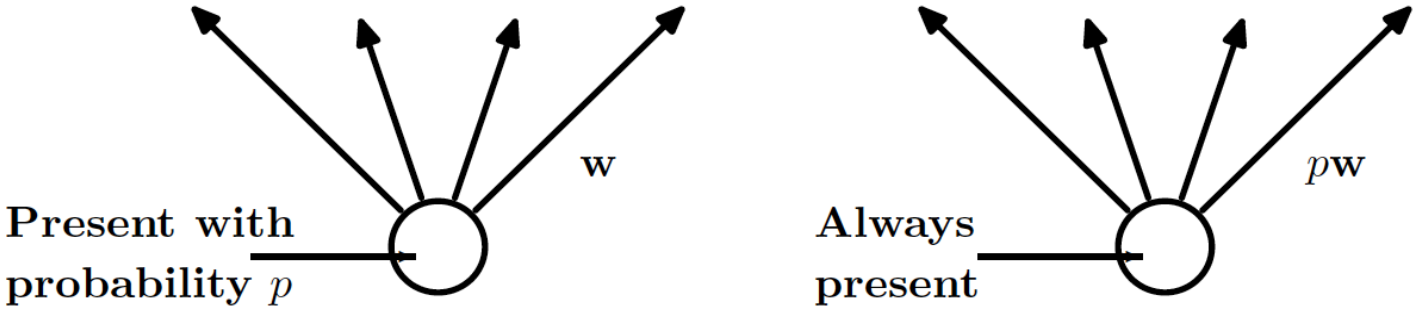
(b) Dropout network

$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)}\mathbf{y}^{(l)} + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f(z_i^{(l+1)})$$

$$r_j^{(l)} \sim Bernoulli(p)$$
$$\tilde{\mathbf{y}}^{(l)} = \mathbf{r}^{(l)} \odot \mathbf{y}^{(l)}$$
$$z_i^{(l+1)} = \mathbf{w}_i^{(l+1)}\tilde{\mathbf{y}}^{(l)} + b_i^{(l+1)}$$
$$y_i^{(l+1)} = f(z_i^{(l+1)})$$

Symbol $\odot$: elementwise product

Dropout (cont.)



(a) At training time (b) At test time

Left: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Summary

- CNNs are tremendously useful in practical applications
- Dropout is commonly used in DNN training
- Backpropagation remains the workhorse of modern deep learning