Convolutional Neural Networks

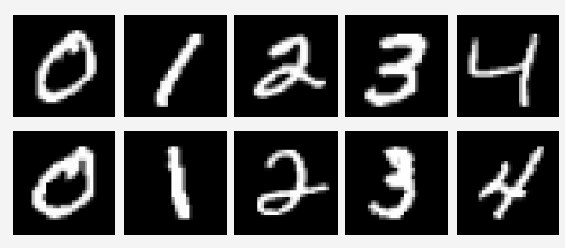
Zhiyao Duan
Associate Professor of ECE and CS
University of Rochester

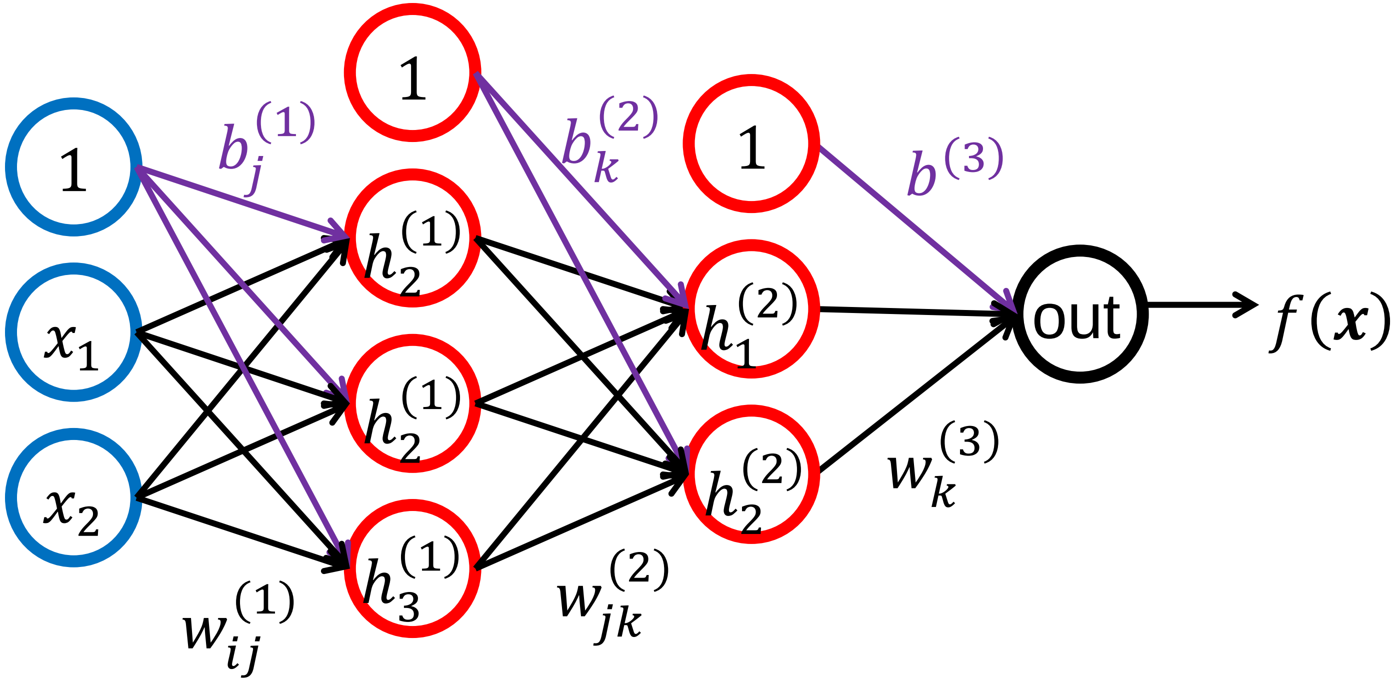
Some figures are copied from the following books

- **LWLS** - Andreas Lindholm, Niklas Wahlström, Fredrik Lindsten, Thomas B. Schön, *Machine Learning: A First Course for Engineers and Scientists*, Cambridge University Press, 2022.
- **GBC** - Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep Learning, MIT Press.

Let's start from Multi-Layer Perceptron

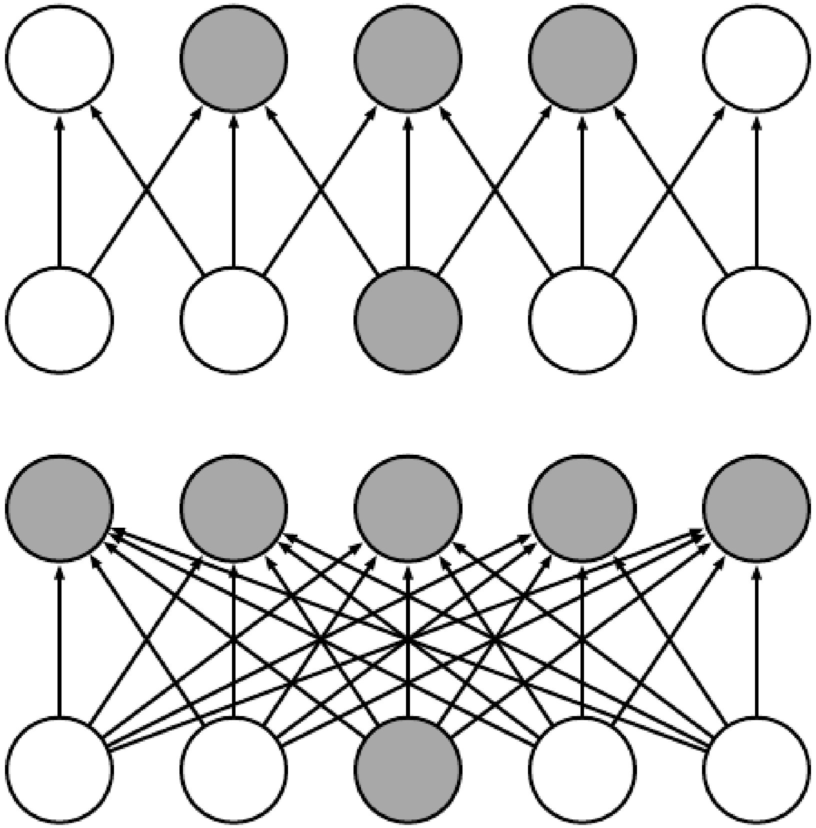
- Fully connected between adjacent layers
  - Many parameters → prone to overfitting
  - Some connections may be unnecessary
  - Not robust to shifts of input





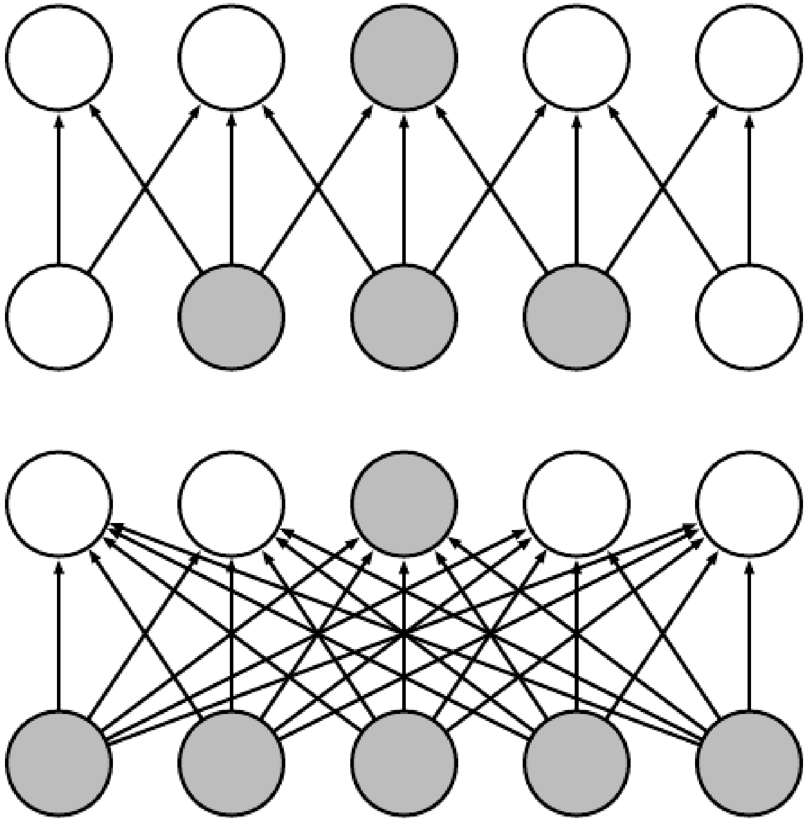
Full Connection → Sparse Connection

- Only keep local connections
  - Assuming nearby inputs have stronger correlations



(Fig. 9.2 in GBC)

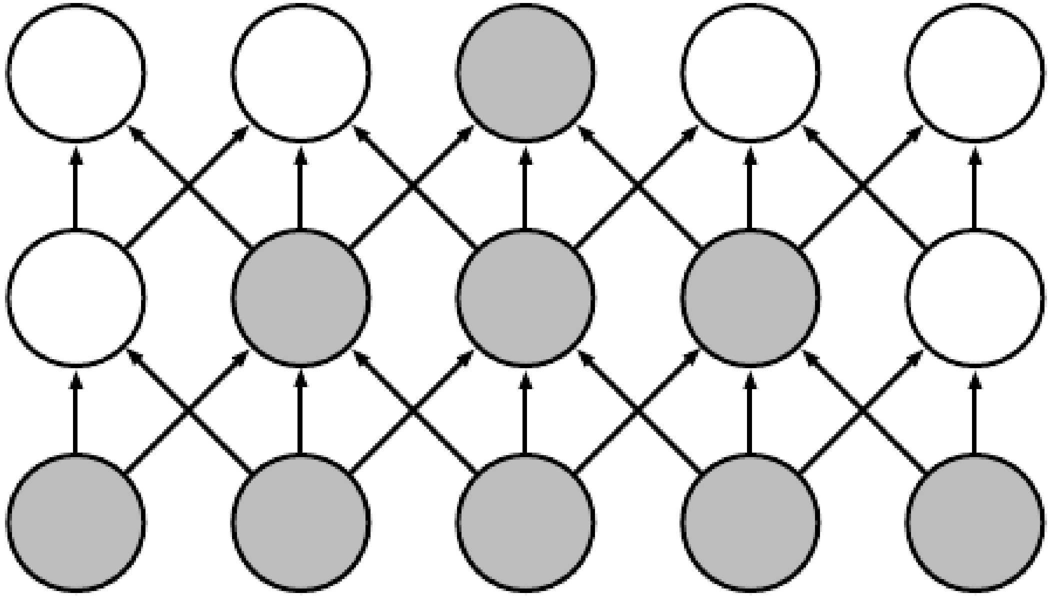
Receptive field of a neuron



(Fig. 9.3 in GBC)

Receptive Field at a Deeper Layer

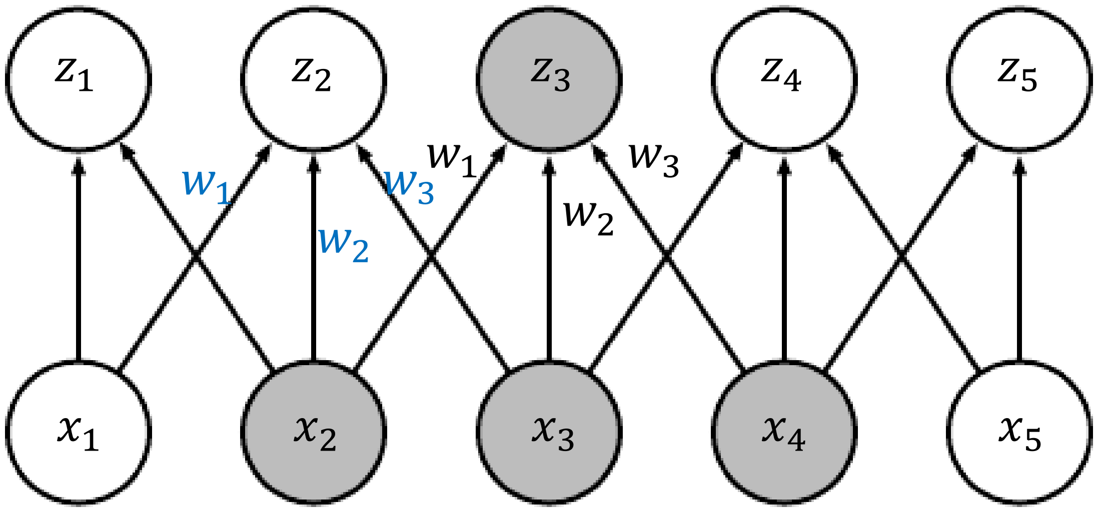
- With sparse connections, nodes at a deeper layer can still have a large receptive field, and global patterns could still be captured



(Fig. 9.4 in GBC)

Independent Weights → Shared Weights

- Assuming neurons at different locations process their inputs in the same way, we can let them share weights



(Adapted from Fig. 9.3 in GBC)

Much Fewer Parameters!

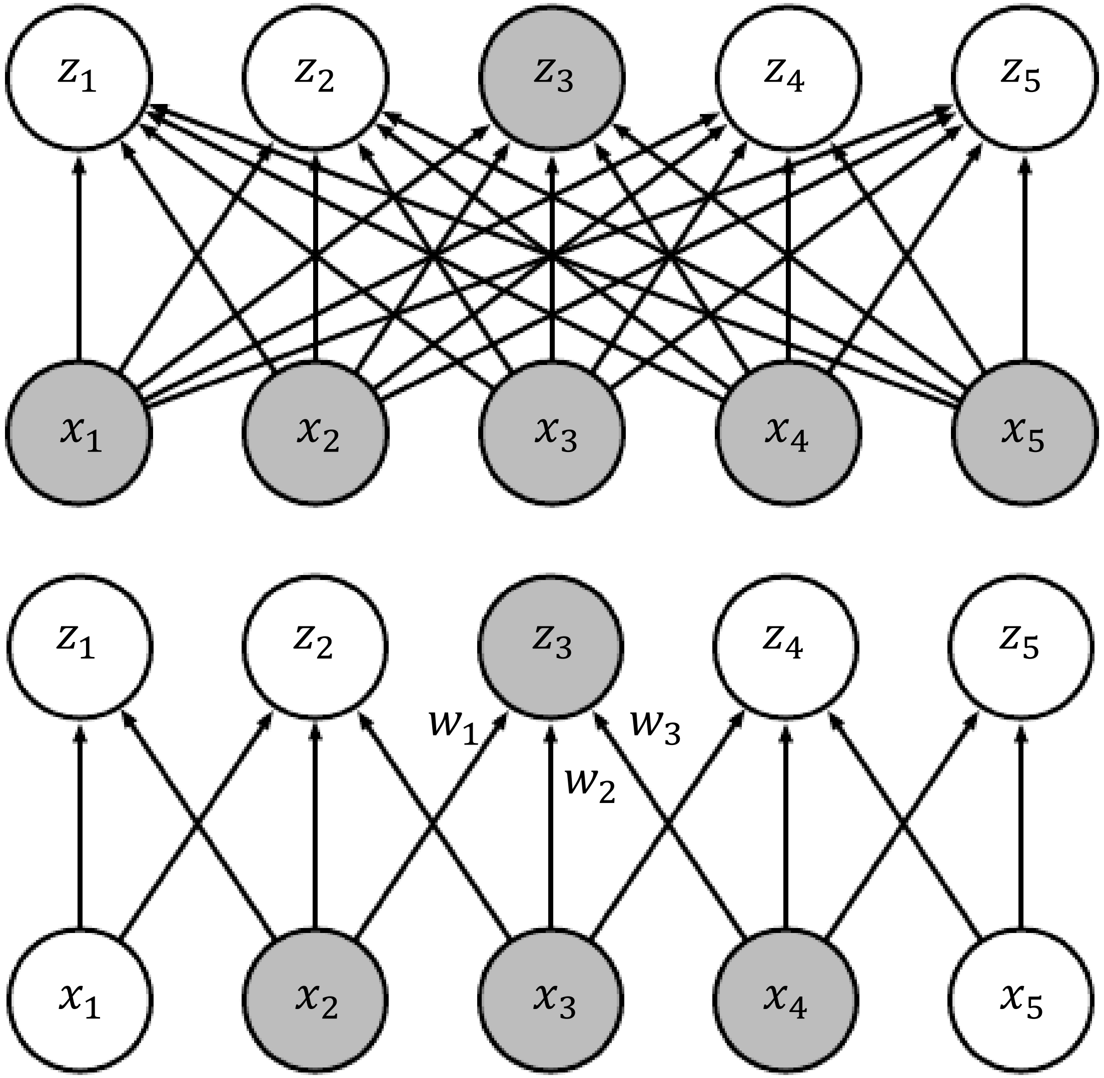
$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{51} \\ \vdots & \vdots & \vdots \\ w_{15} & \cdots & w_{55} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

- 5*5+5 parameters (biases are omitted in figures)

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = \begin{bmatrix} w_2 & w_3 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \\ 0 & 0 & 0 & w_1 & w_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

- 3+1 parameters

$$z_n = \sum_m w_m x_{m+n-2}$$



(Adapted from Fig. 9.3 in GBC)

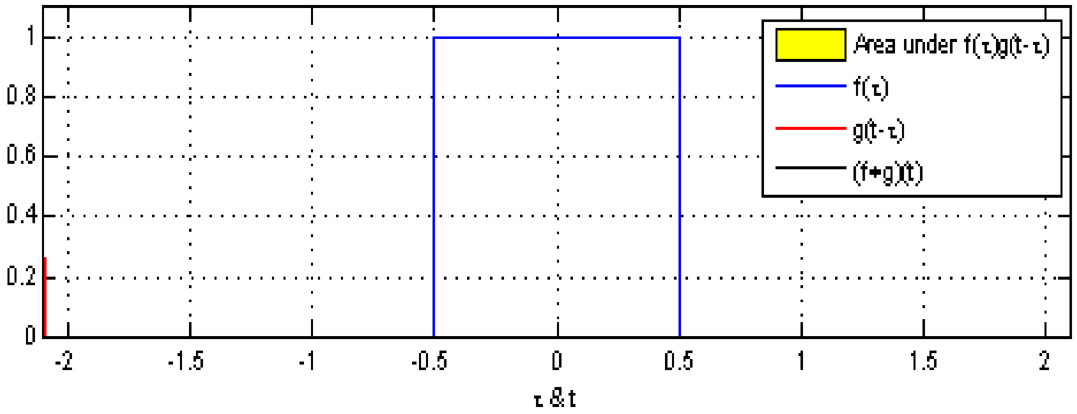
This is basically convolution

- Continuous-time signals

$$z(t) = (x * w)(t) = \int x(\tau)w(t-\tau)d\tau = \int x(t-\tau)w(\tau)d\tau = (w * x)(t)$$

- Discrete-time signals

$$z[n] = x[n] * w[n] = \sum_m x[m]w[n-m] = \sum_m x[n-m]w[m] = w[n] * x[n]$$



- Cross convolution: no flipping, but is the convolution referred to in deep learning

$$z[n] = \sum_m x[m]w[n+m]$$

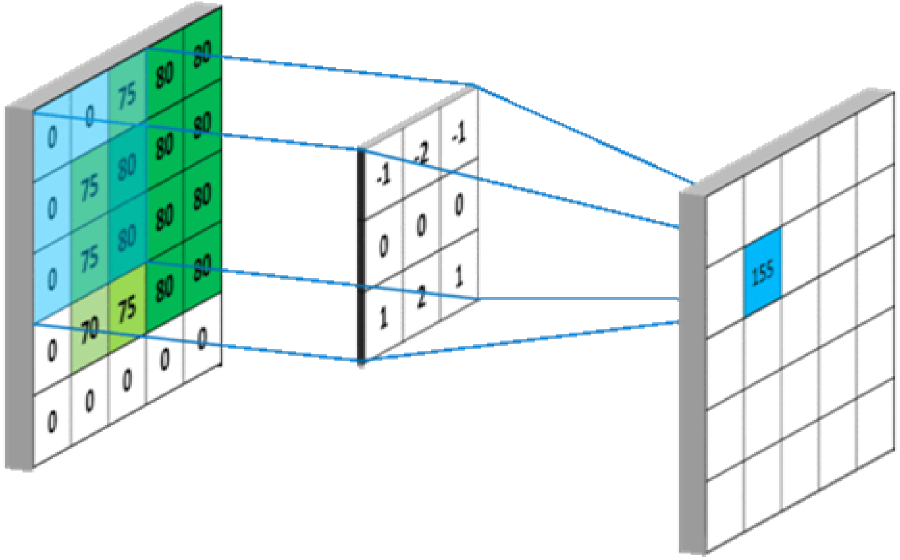
2D Convolution

2D convolution $$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n) K(m,n)$$

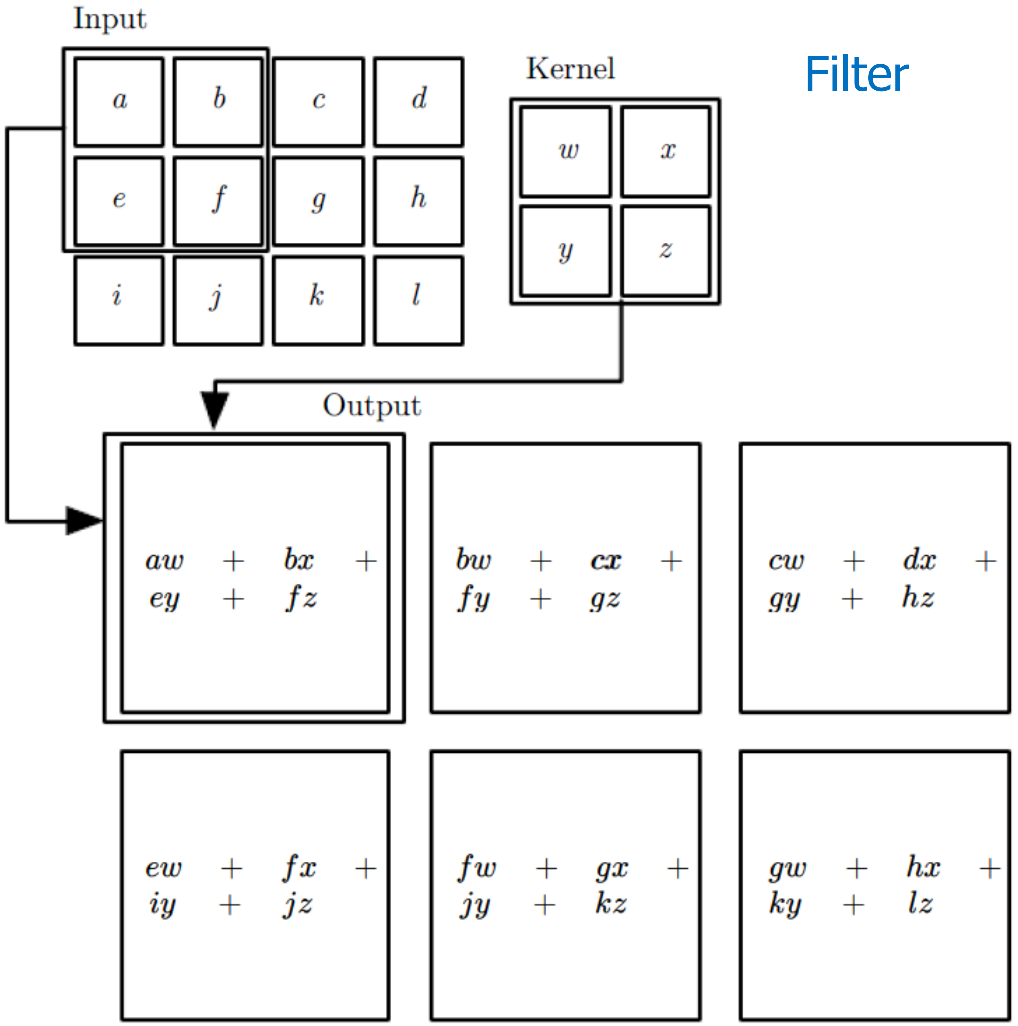
2D cross-correlation $$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

Input Signal 2D
(i.e Image)

Filter/Kernel 2D



2D Convolution



Filter

Feature map

(Fig. 9.1 in GBC)

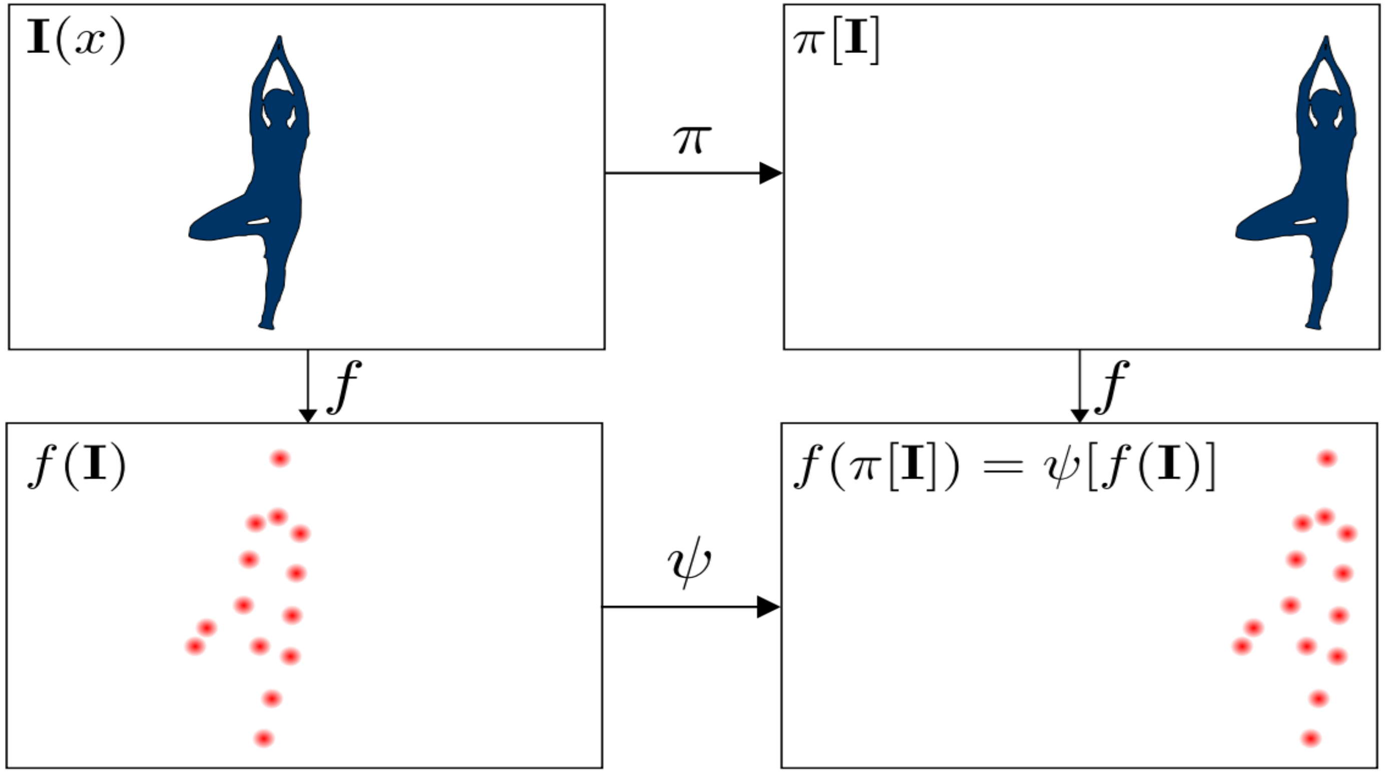
2D Convolution Example

- Vertical edge detection using a 1*2 kernel [-1, 1]
- (Cross-)convolving a gray-scale image with this kernel computes the intensity difference between two horizontally adjacent pixels

(Fig. 9.6 in GBC)

Equivariance to Translation

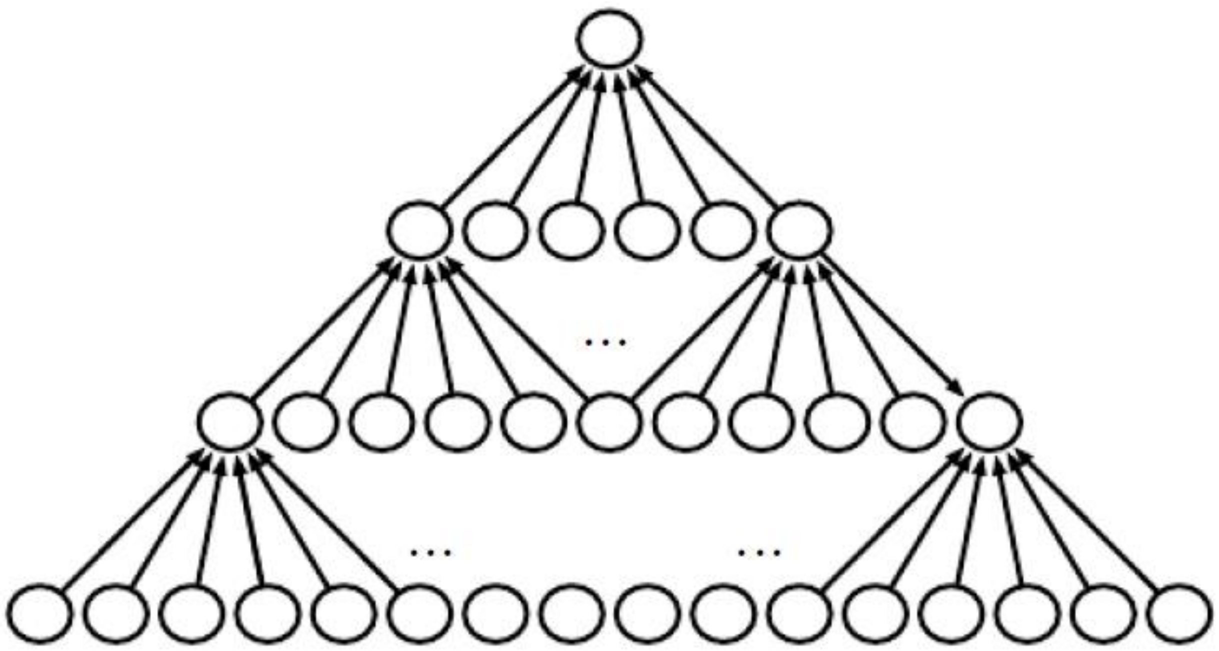
- Shifting input results in the same feature map, but at a correspondingly shifted position



http://visual.cs.ucl.ac.uk/pubs/harmonicNets/pdfs/worrallEtAl2017.pdf

Zero Padding Before Convolution

- Convolution reduces size if no zero padding
  - Called "valid convolution"

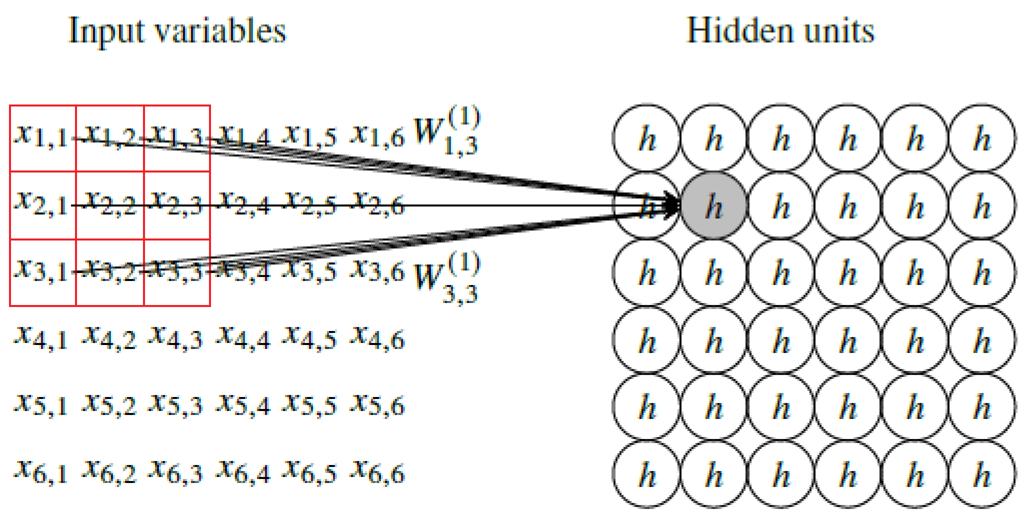


- Use zero padding to maintain size
  - Called "same convolution" (preferred)

- Pad more zeros to make edge nodes have the same number of connections as internal nodes
  - Called "full convolution"

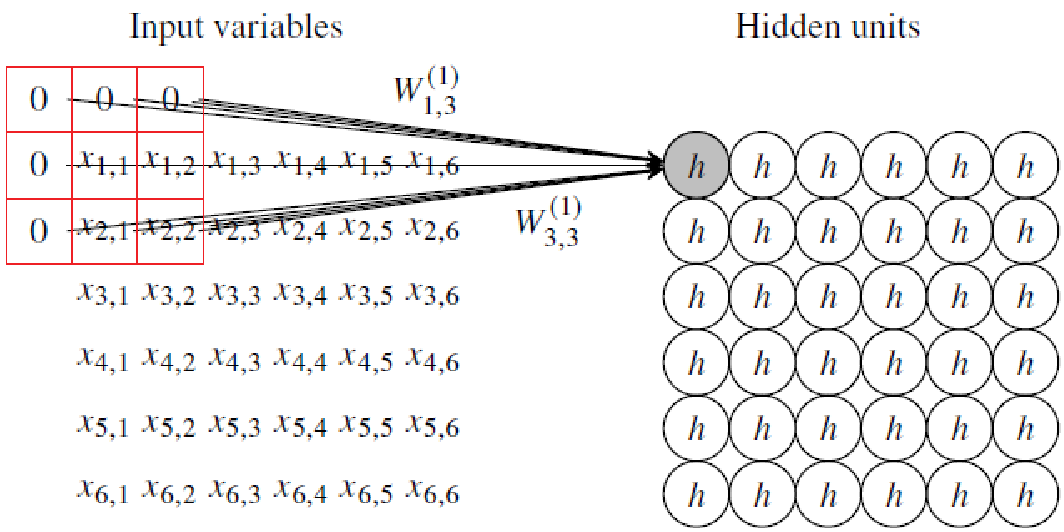
2D Zero Padding

Without zero padding



(Fig. 6.10 in LWLS)

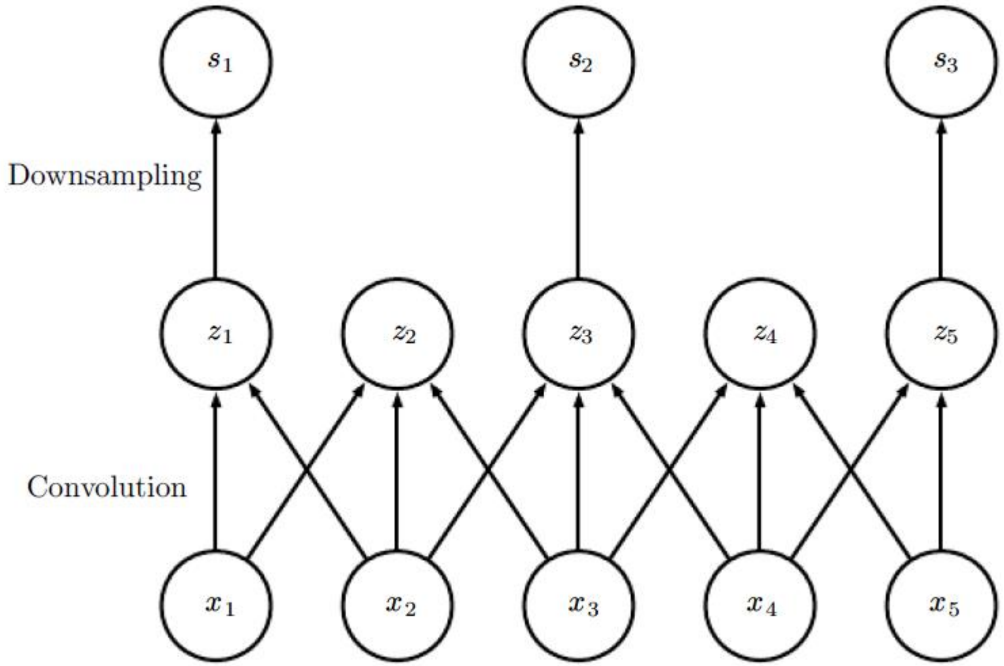
With "same" zero padding



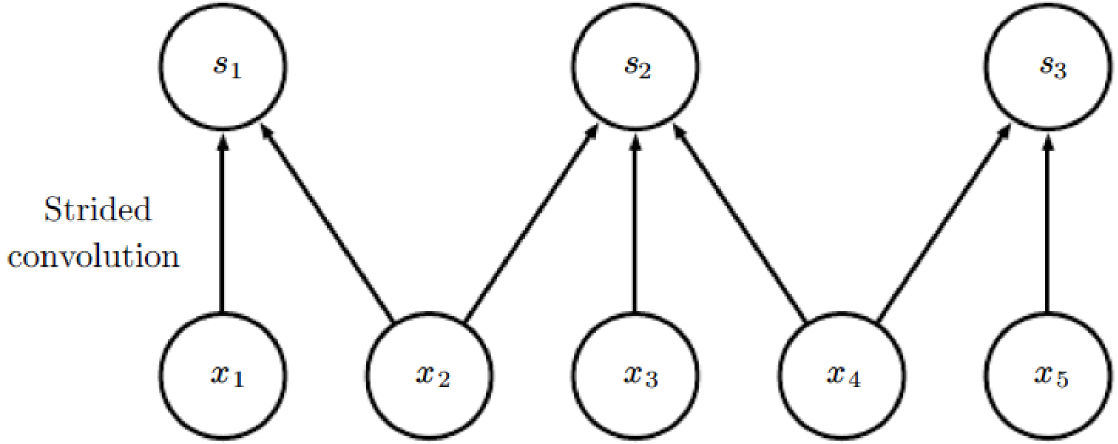
(Fig. 6.11 in LWLS)

Convolution with Strides

- Downsampling after convolution



Stride size = 2



(Fig. 9.12 in GBC)

2D Convolution with Strides

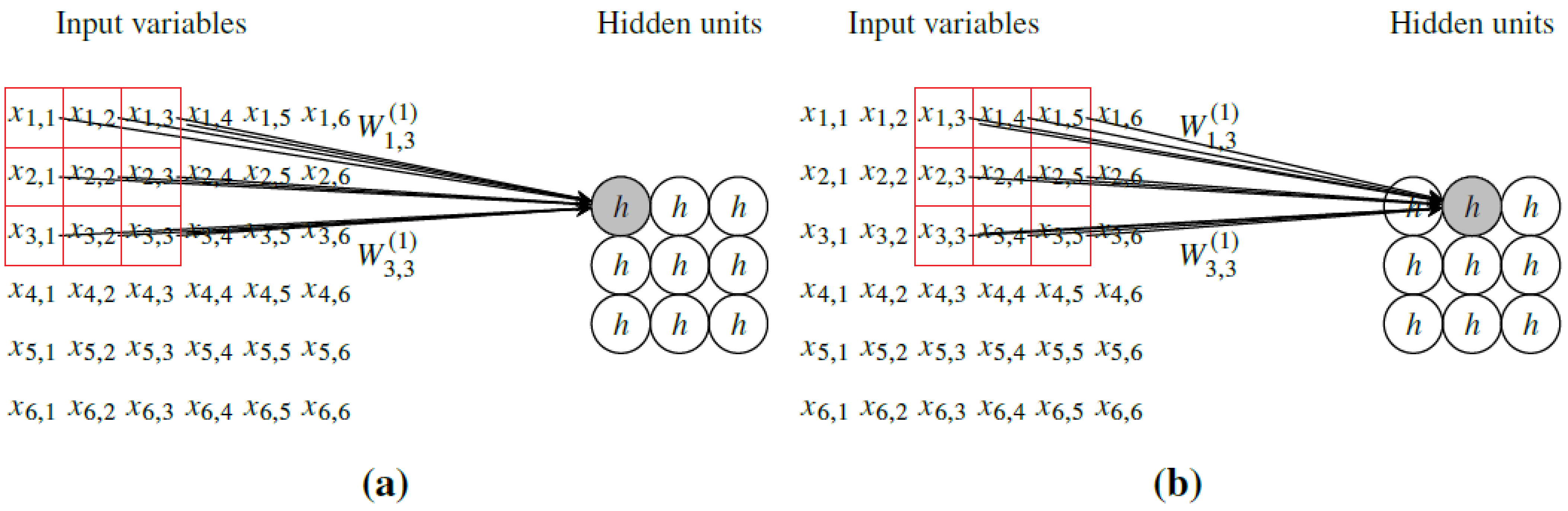
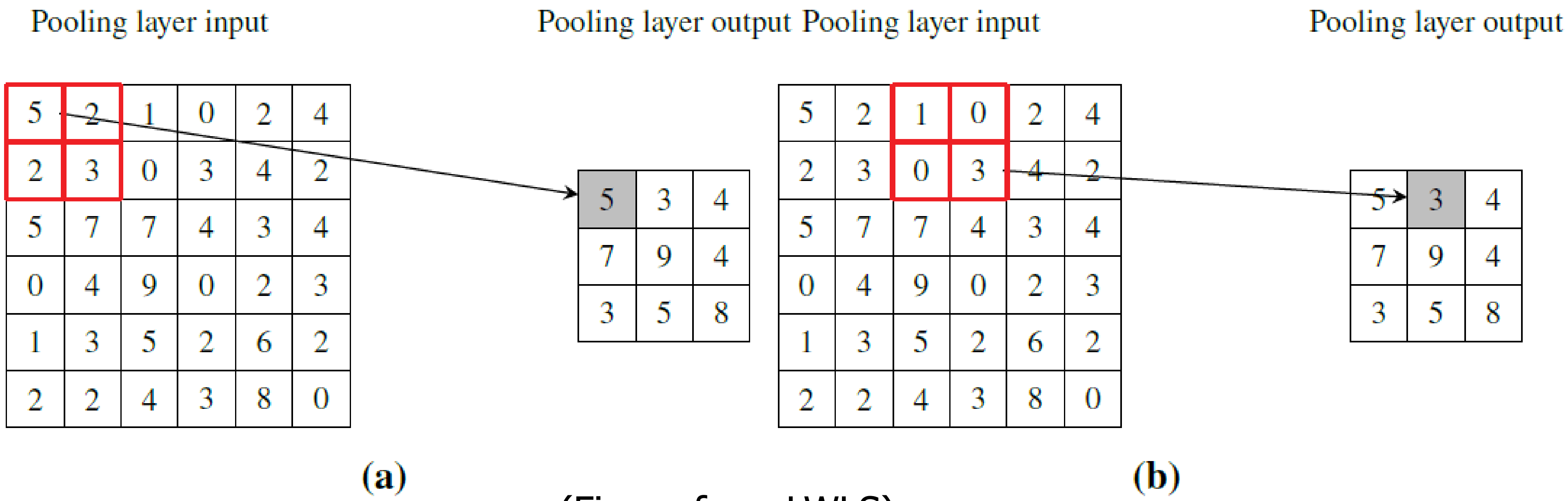


Figure 6.12: A convolutional layer with stride 2 and filter size 3 × 3.

(Figure from LWLS)

Pooling

- Pooling is another way to reduce the size of feature maps
  - Max pooling: taking the max → result is invariant to small shifts
  - Average pooling: taking the average
- No trainable parameters

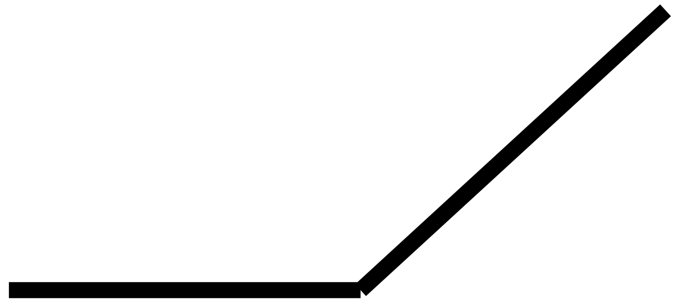


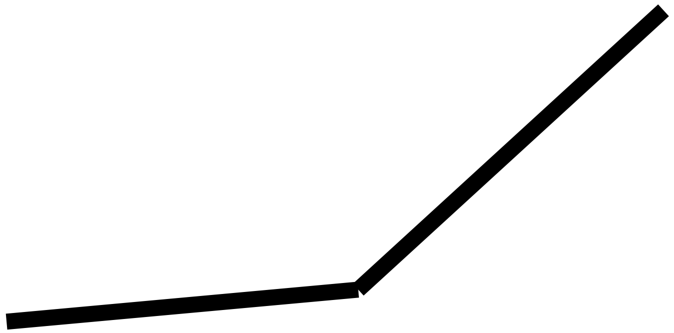
(Figure from LWLS)

Figure 6.13: A max pooling layer with stride 2 and pooling filter size 2×2.

Nonlinear Activation

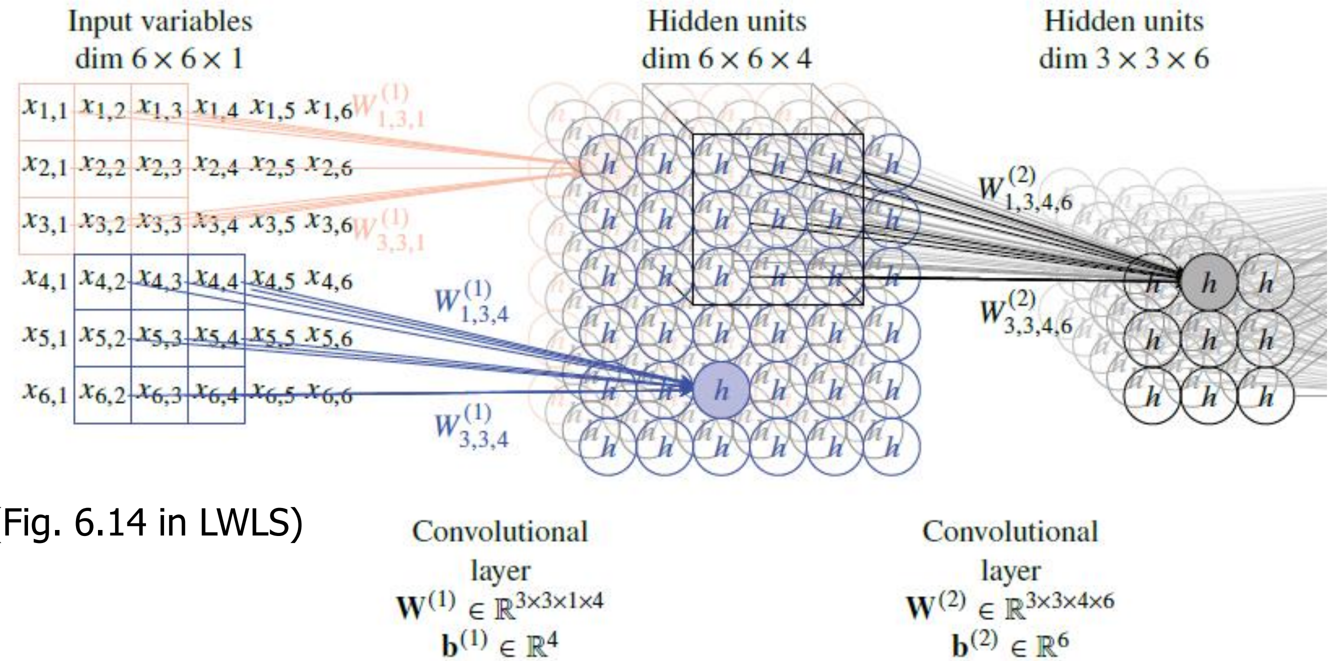
- As discussed before, convolution is a linear operation
- We need a nonlinear activation after convolution to build deep nets
- Rectified Linear Unit (ReLU) and Leaky ReLU is most used





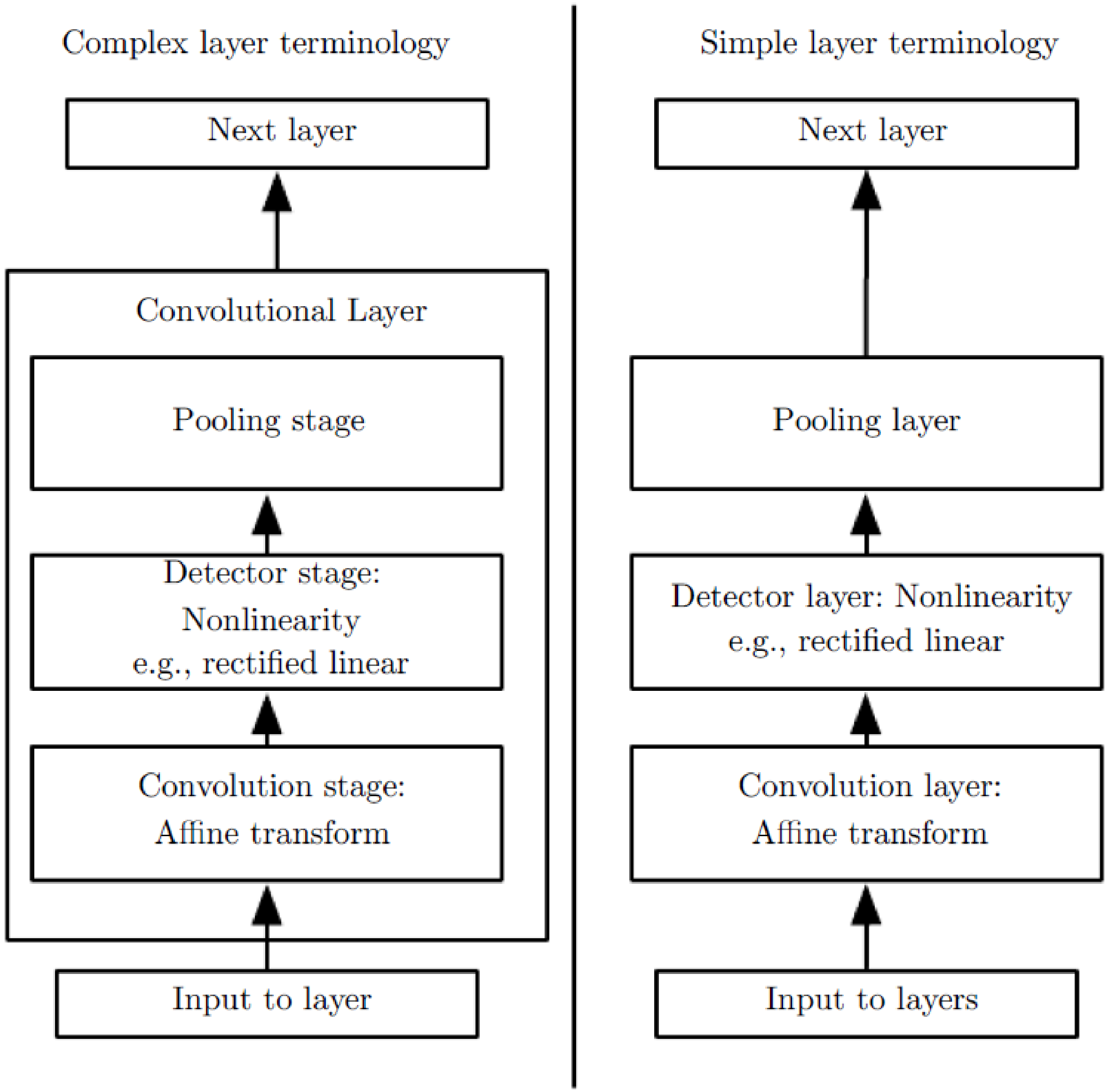
Multiple Channels

- Convolution with a single filter (kernel) detects only one pattern (e.g., vertical edges)
- Use multiple filters to detect more patterns
  - Each filter results in one feature map
  - Multiple filter result in multiple feature maps, stacked as channels
  - When input is 2D with multiple channels, each filter becomes a 3D tensor



(Fig. 6.14 in LWLS)

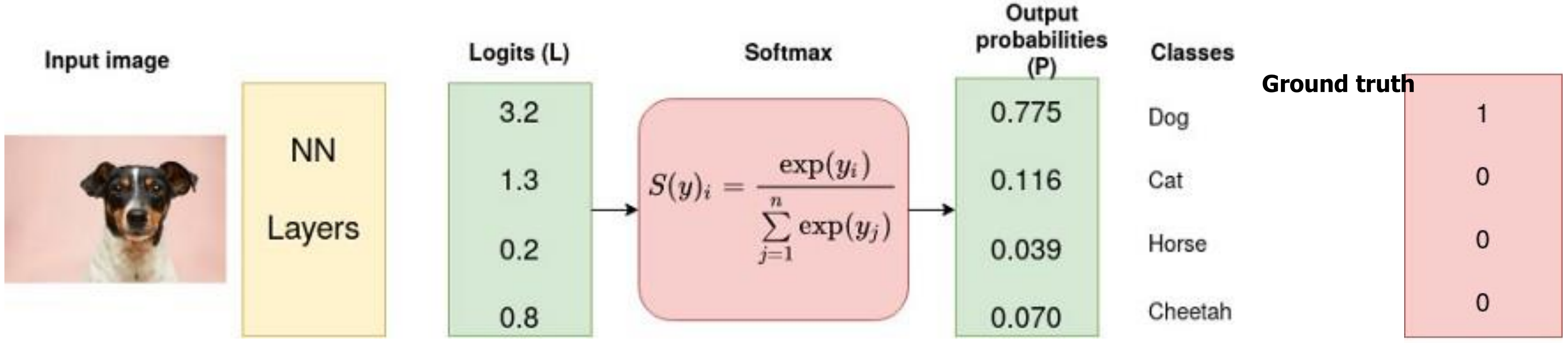
Convolution Layer



(Fig. 9.7 in GBC)

Typical Output Layer

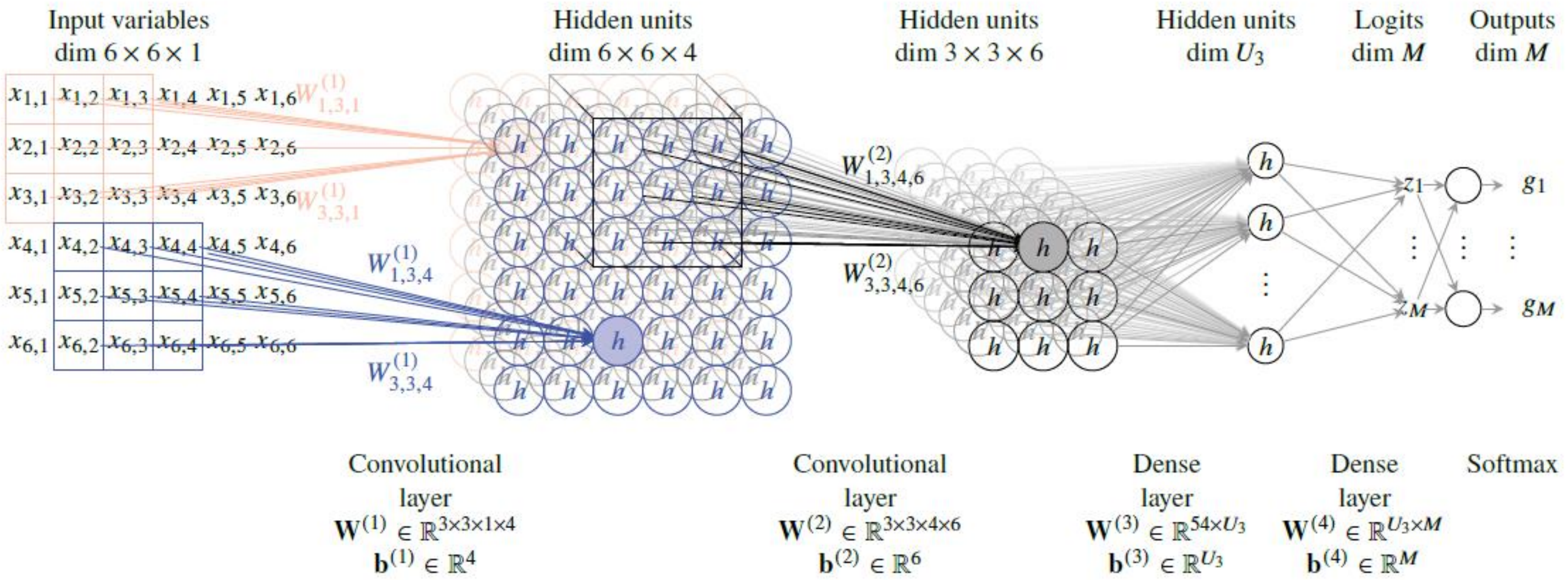
- After a stack of convolutional layers, a few fully connected layers often follow to give the output
  - The last convolutional layer's feature map is reshaped to a vector
- M-Class Classification:
  - Use M output nodes
  - Softmax activation (probability): $\hat{y}_i = \frac{e^{h_i}}{\sum_{j=0}^{M-1} e^{h_j}}, \forall i = 0, \cdots, M-1$
  - Cross entropy loss: $L_{CE} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$



(Figure from https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e)

Full CNN Architecture

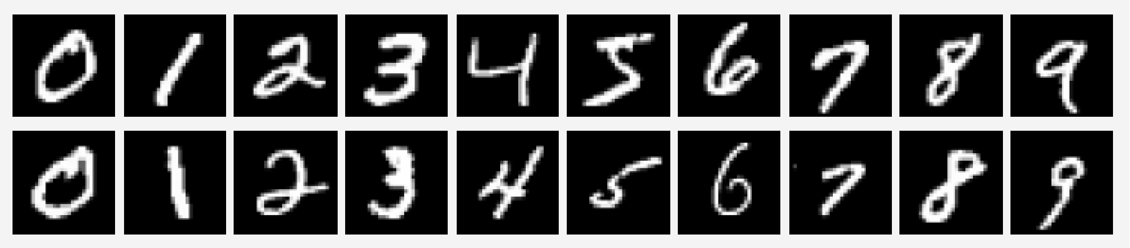
- M-class classification on single-channel 2D input



(Fig. 6.14 in LWLS)

Full CNN Architecture

- Input: 28*28=784-d gray-scale (i.e., 1-channel) hand-written digits



	Convolutional layers			Dense layers	
	Layer 1	**Layer 2**	**Layer 3**	**Layer 4**	**Layer 5**
Number of filters/output channels	4	8	12	–	–
Filter rows and columns	(5×5)	(5×5)	(4×4)	–	–
Stride	1	2	2	–	–
Number of hidden units	3 136	1 568	588	200	10
Number of parameters (including offset vector)	104	808	1 548	117 800	2 010

(Example 6.3 in LWLS) 784*4=3136 784/4*8=1568 784/4/4*12=588

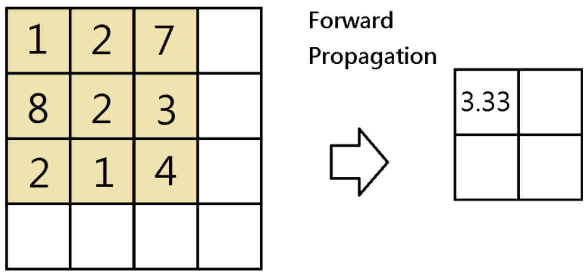
Network Training

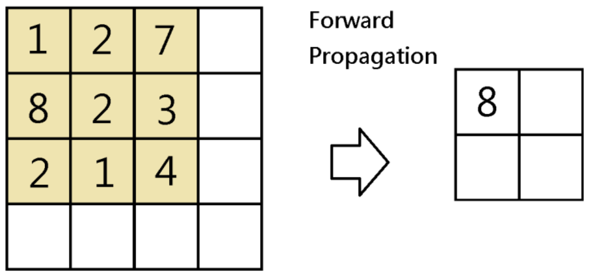
- Define a loss function
  - Classification: cross entropy for softmax output
  - Regression: mean squared error
- Stochastic gradient descent
  - Randomly picking training samples to form a mini-batch
  - Compute gradient of loss function w.r.t. weights through backpropagation
  - Update weights along negative gradient with some (adaptive) learning rate
- Different optimizers
  - Adam: adaptive moment estimation – uses running averages on gradients and second order moments
  - Adagrad: adaptive gradient – uses different learning rates at different iterations
  - RMSprop: root mean square propagation – exponentially weighted average of squared gradient to adapt learning rate

Backpropagation for CNN

- BP through nonlinear activation
  - Same as before

- BP through pooling
  - Average pooling: gradient is equally distributed to all inputs
  - Max pooling: gradient is solely assigned to the max input





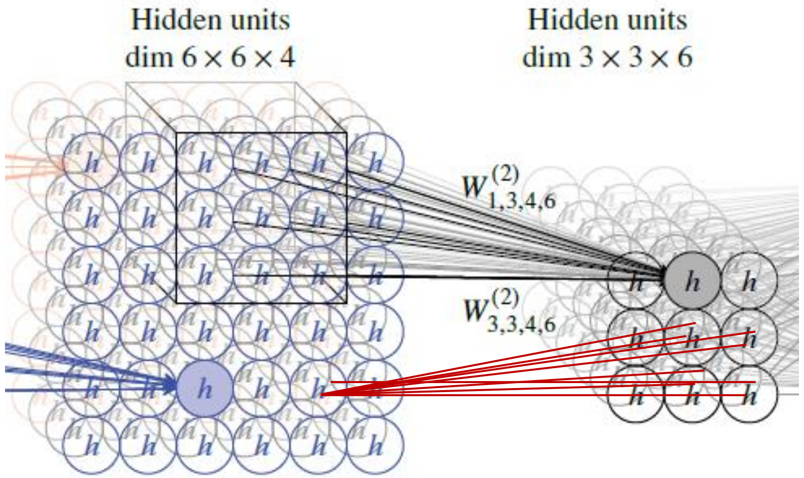
(Figures from https://lanstonchu.wordpress.com/2018/09/01/convolutional-neural-network-cnn-backward-propagation-of-the-pooling-layers/)

Backpropagation for CNN

- Convolution is a linear operation between the input tensor and a kernel, and it results in an output tensor

- BP through convolution to layer input
  - Each element of the input tensor affects multiple channels of the output tensor through different filters

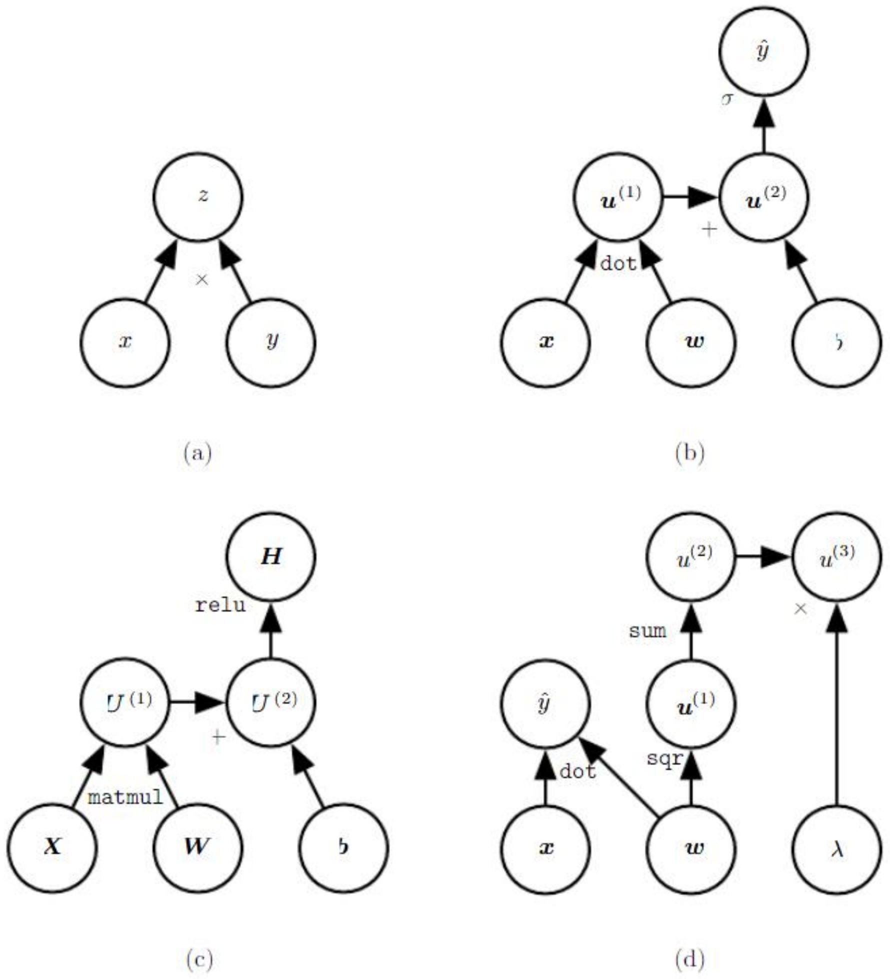
- BP through convolution to layer weights
  - Each weight affects all elements of one output channel through one channel of previous layer's output



(Adapted from Fig. 6.14 in LWLS)

Computational Graph

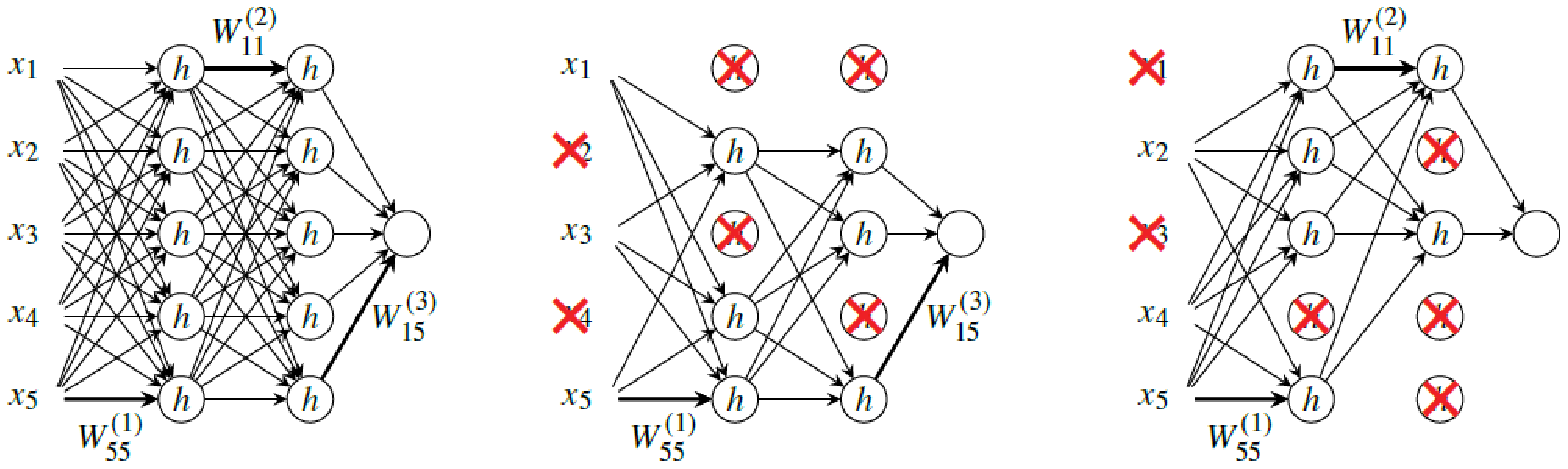
- Node: a variable (e.g., scalar, vector, matrix, tensor)
- Operation: a simple function of one or more variables, outputting a single variable
  - More complicated functions can be realized through composing these operations
- Chain rule of calculus
  - Gradient backpropagates through the graph using derivatives (Jacobian matrices) of operations



(Fig. 6.8 in GBC)

Dropout

- An important technique to alleviate overfitting
- Randomly (with probability 1-r) dropout some neurons/filters in each iteration of training
  - They do not participate in either forward computation or backpropagation
- During inference (i.e., predicting on unseen data), multiple network weights with r
- Conceptually, the learned model is like an ensemble of networks that share some weights
- Practically very effective; theoretically unclear why



(a) A standard neural network (Fig. 6.18 in LWLS) **(b)** Two sub-networks

Batch Normalization

- Internal Covariance Shift: inputs to internal layers have random shifts on means and variances due to the unexpected weight updates of previous layers
  - Remember that gradient w.r.t. a weight is computed assuming all the other weights are static (partial derivative), but in practice all weights are updated simultaneously in one backward pass

- Idea: normalize net input to each internal node (or output from its previous node) using mean and variance computed in a mini-batch

$$\hat{h} = \frac{h - \mu}{\sigma}$$

where $\mu = \frac{1}{M}\sum_{i=1}^{M} h^{(i)}$ and $\sigma = \sqrt{\delta + \frac{1}{M}\sum_{i=1}^{M}(h^{(i)} - \mu)^2}$. M is the mini-batch size, δ is very small

- Backprop needs to go through this operation, i.e., $\frac{\partial \hat{h}^{(i)}}{\partial h^{(i)}}$ is computed. Note $h^{(i)}$ affects $\hat{h}^{(i)}$ through the computation of μ, σ, and the normalization operation.

Batch Normalization

- For CNNs: use the same mean and variance for outputs at different spatial locations of the same filter
  - This is to preserve the statistics of the relations among different spatial locations

- During inference: apply normalization the same way on test data, but using a running average of means and variances of training mini-batches

- Batch normalization removes influences on means and variances (i.e., first- and second-order moments) of internal layer inputs from previous layers, but still preserves their influences on higher order moments
  - This is a type of regularization, reducing the expressive power of the network

CNNs for Different Types of Input

	Single-Channel	**Multi-Channel**
1-D	Audio waveforms	Skeleton animation data: Each channel represents one angle of one joint
2-D	Audio spectrograms; gray-scale images	Color images: RGB channels
3-D	Volumetric data, e.g., CT scans	Color video data

(Adapted from Table 9.1 in GBC)

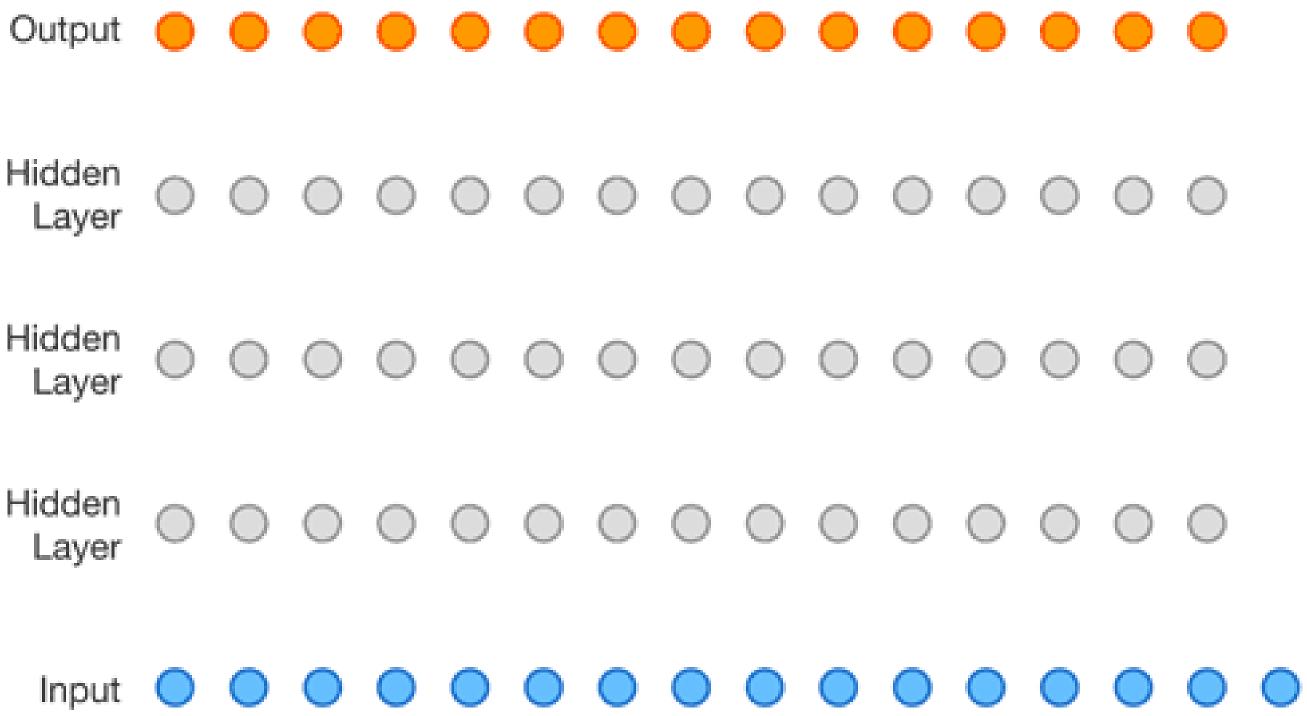
1D CNN for Audio Generation

- WaveNet [van den Oord et al., 2016]
- Dilated causal convolution

free generation (speech)

text-to-speech

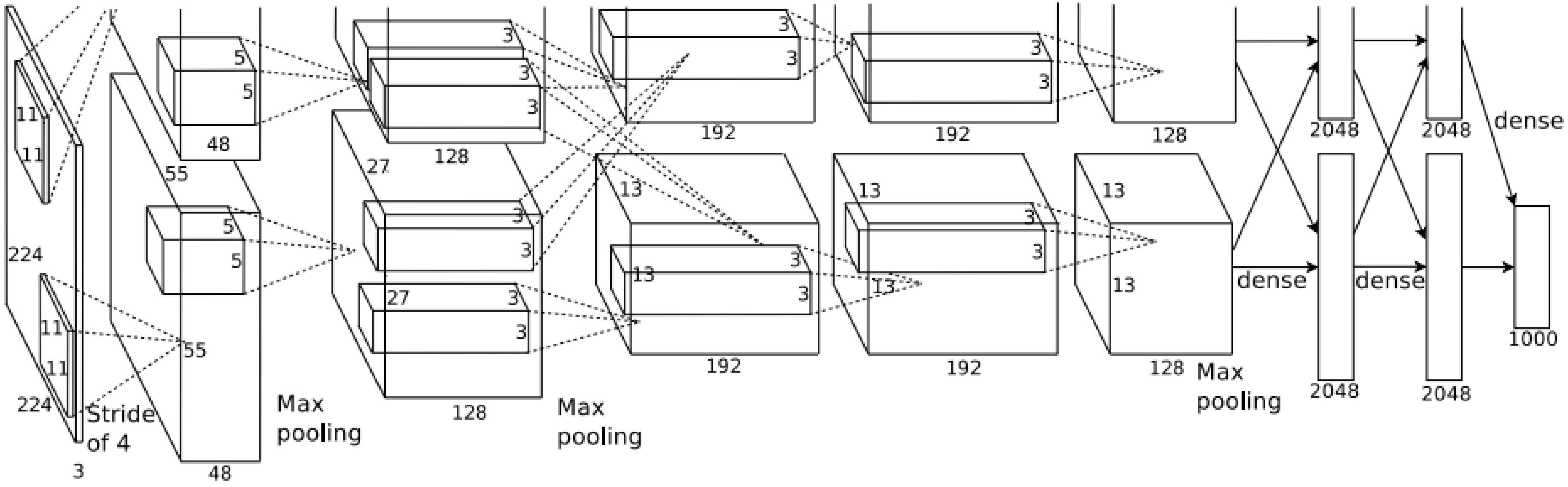
Free generation (piano music)



https://www.deepmind.com/blog/wavenet-a-generative-model-for-raw-audio

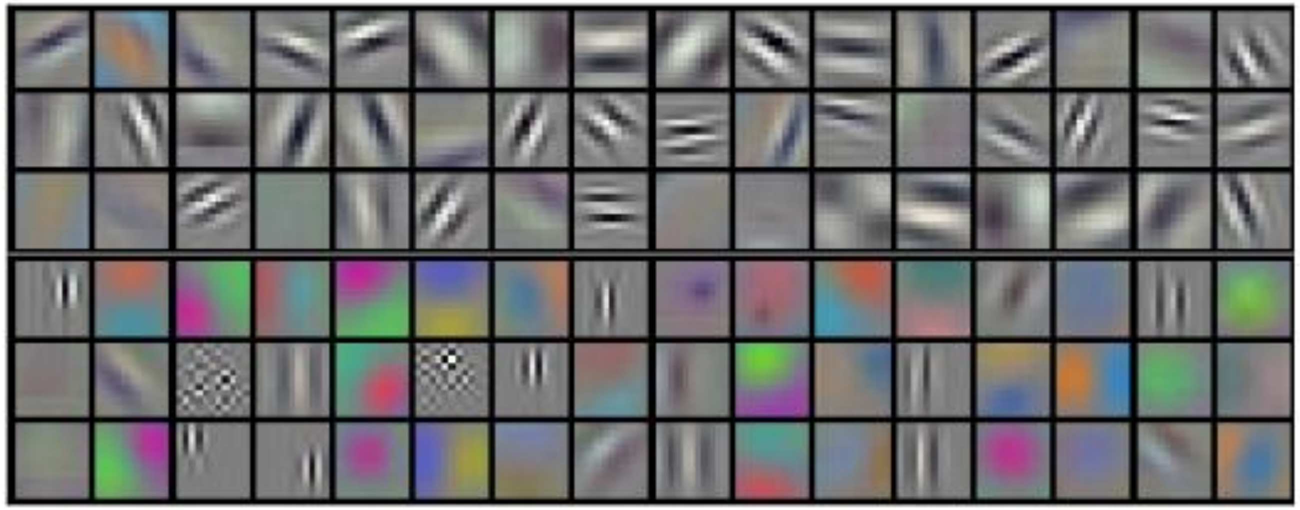
2D CNN for Image Classification

- AlexNet [Krizhevsky et al., 2012]



Filter Visualization of AlexNet

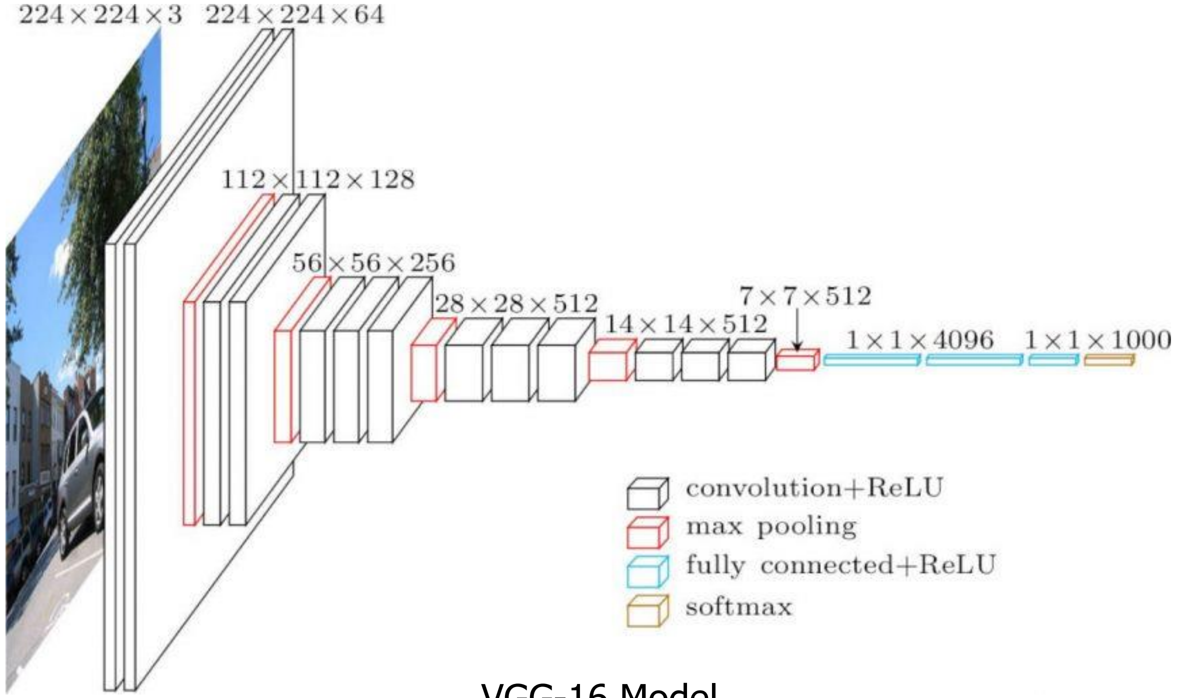
- Learned filters of the 1st convolutional layer
  - 96 filters with size of 11*11*3



[Krizhevsky et al., 2012]

Transfer Learning with Pretrained Networks

- First layers (features extractors) learned from one task (e.g., natural image classification) can be useful for another relevant task (e.g., medical image classification)
- Use a pre-trained model (on big data tasks) to build a new model (for small data tasks)
  - Remove last few layers (e.g., the last dense layer), which are usually task-specific
  - Use the remaining layers to build a new network by adding a couple of layers for the new task
  - Train new layers (or fine tune the entire network) on the new task



VGG-16 Model

ImageNet

- 1.3 M images from 1000 classes

Summary

- Key properties of CNNs
  - Sparse (local) connection
  - Shared weights
  - Equivariance to translation
- Important components
  - Convolution
  - Pooling: max pooling, average pooling
  - Activation: ReLU
- Important concepts
  - Filter, receptive field, channel, tensor
- Applications
  - Classification, regression, generation
  - 1D, 2D, 3D
- Think: what problems/data are not appropriate for CNNs?