Convolutional Networks

Tripp Deep Learning F22

Figures from Goodfellow et al., Deep Learning

TODAY'S GOAL

By the end of the class, you should be able to explain the main elements of convolutional networks, and the advantages, disadvantages, and applicability of convolutional layers.

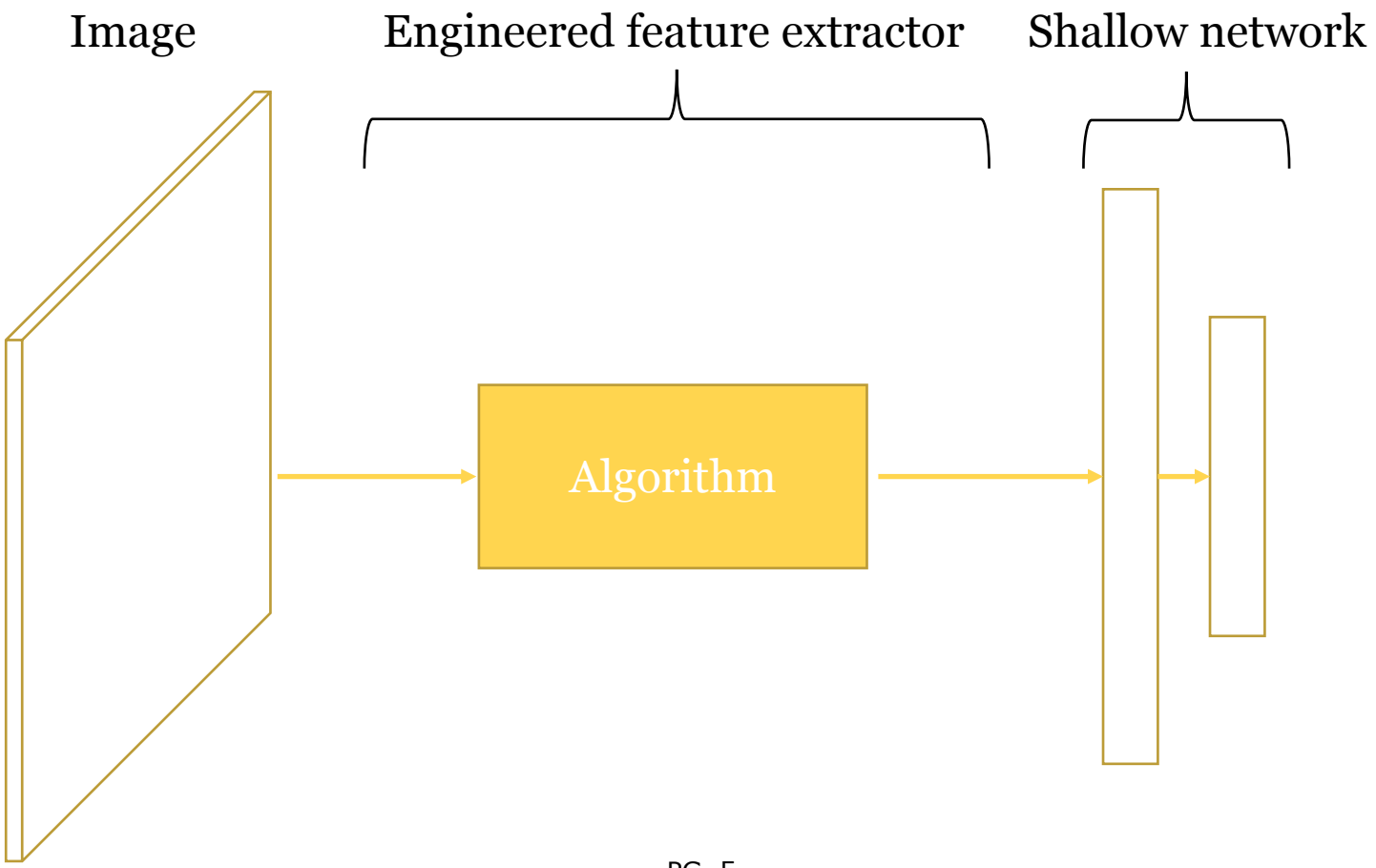
Summary

1. Convolution is related to correlation
2. Convolution emphasizes features that are like the kernel
3. Convolution is linear
4. Convolution is less general than matrix multiplication
5. Convolutional layers work with signals, images, and volumes
6. Convolutional layers have multiple channels
7. Something must be done about the edges
8. Convolutional layers are efficient in terms of computation and parameters
9. Stride and dilation subsample in different ways
10. Pooling layers introduce translation invariance

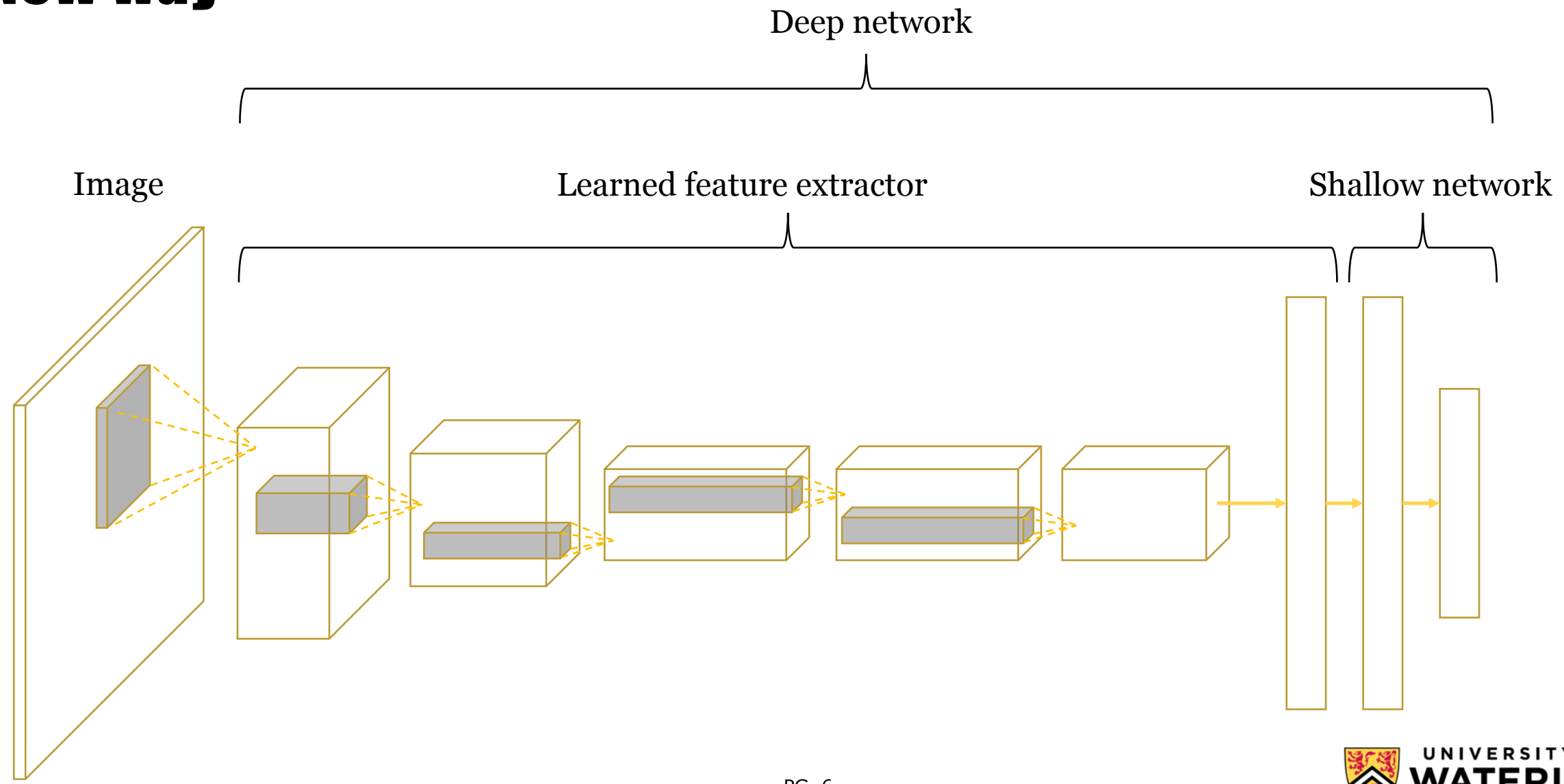
Motivation

- Many problems, including vision problems such as object recognition, require complex networks
- Standard networks that are made complex enough simply by making them wider and/or deeper do not generalize well
- This is mainly because they have too many parameters to be constrained well enough by available data
- Standard practice in 1990s was to pre-process data to extract manually-defined features (labour-intensive and non-optimal)
  - E.g., Histogram of oriented gradients (HOG) for image recognition
- Modern methods focus on more effective approaches to end-to-end learning

Old way



New way



Convolutional networks

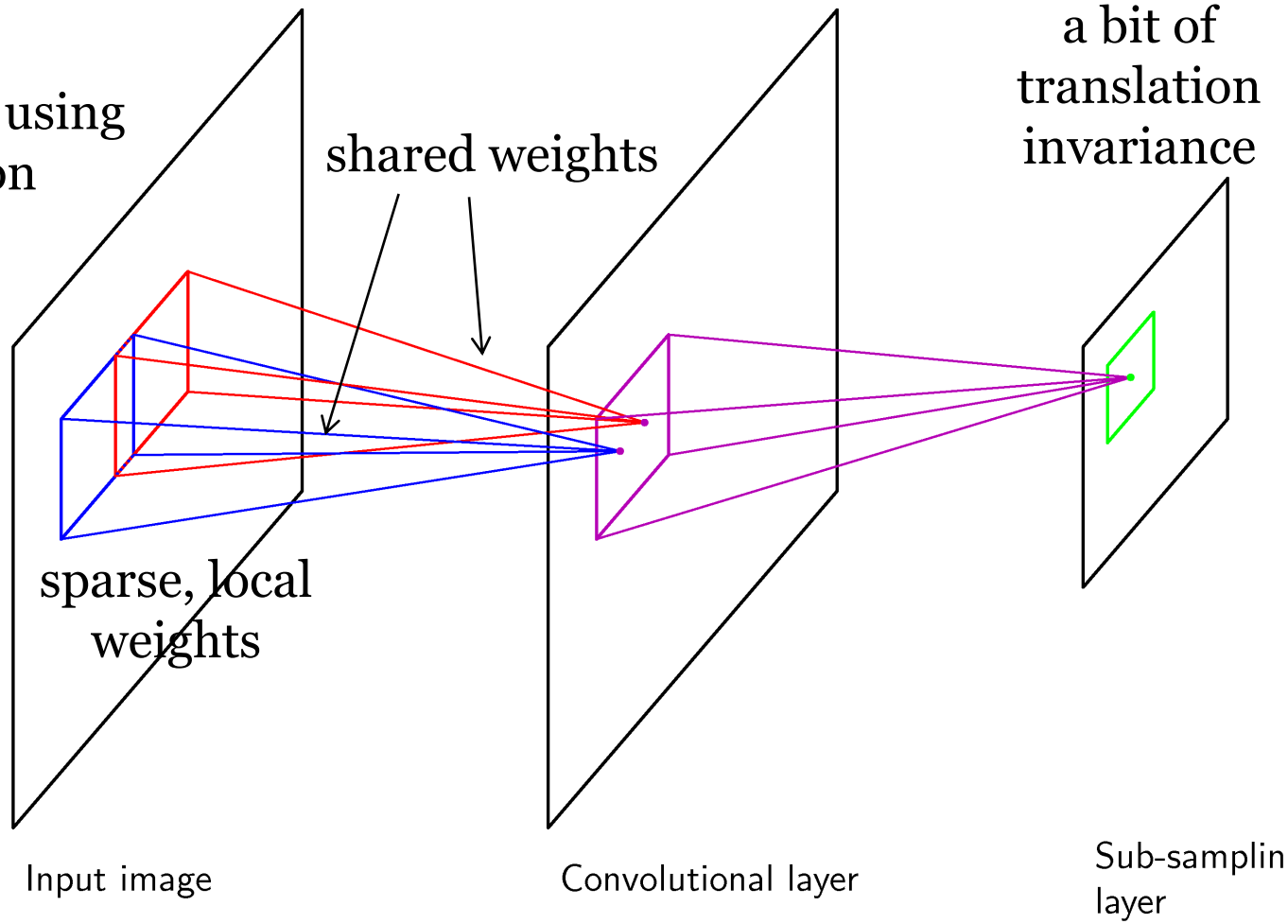
- Definition: A convolutional network is a neural network that uses convolution as its linear operation in at least one layer (in place of the weight matrix)
- They make mild assumptions about the required features that greatly reduce the numbers of parameters needed
- These were the leading approach throughout computer vision from roughly 2012 until 2022 (recently often outperformed by transformers, although some of these also incorporate convolutional layers)
- They also work well for one-dimensional signals and three-dimensional volumes

Convolutional networks – key ideas

Bishop, *Pattern Recognition and Machine Learning*

- Inputs to a neuron only come from a small part of the image (or previous layer)
- Weights are shared across many neurons that see different parts of the image
- Outputs of multiple neurons are often pooled over a local region to enforce translation invariance

Results of using convolution



UNIVERSITY OF WATERLOO

CONVOLUTION IS RELATED TO CORRELATION

Discrete convolution with finite support

Recall the standard one-dimensional convolution of a continuous-time signal,

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

where f is the signal, g is the kernel, and t is time. Similarly, convolution in discrete time is,

$$(f * g)[k] = \sum_{m=-\infty}^{\infty} f[k-m]g[m]\,.$$

If the kernel has finite support in the domain $-M$ to M then,

$$(f * g)[k] = \sum_{m=-M}^{M} f[k-m]g[m]$$

Cross-correlation vs. convolution

- Usually, “convolutional” networks really use cross-correlation, which is a measure of the **similarity** between a signal and a kernel (or between two signals)
- The cross-correlation of real-valued discrete-time signals f and g is,

$$(f \star g)[k] = \sum_{m=-\infty}^{\infty} f[k+m]g[m]$$

- The cross-correlation of a signal f with a kernel g that has finite support over $-M$ to M is,

$$(f \star g)[k] = \sum_{m=-M}^{M} f[k+m]g[m]$$

Like convolution but $\star$ instead of $*$ and $+$ instead of -

Cross-correlation vs. Pearson correlation

The cross-correlation is related to the Pearson product-moment correlation coefficient,

$$\rho_{X,Y} = \frac{E[(X - \mu_x)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

Given N samples samples, (x_i, y_i), of these variables, $\rho_{X,Y}$ can be estimated with the sample correlation coefficient,

$$r_{xy} = \frac{1}{N} \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{s_x s_y},$$

where $\bar{x}$ and $\bar{y}$ are the sample means and s_x and s_y are the sample standard deviations.

Cross-correlation vs. correlation

A related measure is the sample covariance, which omits normalization by s_x and s_y,

$$q_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

Let's imagine an over-simplified version of sample covariance, q^o_{xy}, that ignores the means and doesn't normalize by N: $q^o_{xy} = \sum_{i=1}^{N}(x_i)(y_i)$.

This is the basis of cross-correlation,

$$(f \star g)[k] = \sum_{m=-M}^{M} f[k+m]g[m] = q^o_{f_k g},$$

where f_k indicates the signal f offset by k samples, $N = 2M + 1$, and $i = m + M + 1$.

Cross-correlation vs. correlation

The cross-correlation equation,

$$(f \star g)[k] = \sum_{m=-M}^{M} f[k+m]g[m]\,,$$

is consistent with the sample correlation coefficient if one considers f and g as signals that are preprocessed in the following ways:

- Centered around zero (mean subtracted from the signal)
- Divided by their standard deviations
- Divided by $2M + 1$

Engineers think of this equation as cross-correlation regardless of whether these steps are performed and may or may not perform each of them depending on the context.

Cross-correlation vs. correlation

- Note that for the cross-correlation to be fully consistent with sample correlation, each segment $f[k - M : k + M]$ must be independently normalized by its own standard deviation

When convolution and cross-correlation are the same

- They are equivalent when the kernel is symmetric, i.e., $g[m] = g[-m]$
- Also, the distinction doesn't matter when the kernel is learned
  - E.g., suppose convolution is used and a learning process leads to $g[3] = 1.2$
  - If cross-correlation were used instead and everything else were the same, the learning process would lead to $g[-3] = 1.2$
- Convolutional networks normally use cross-correlation rather than convolution, but it doesn't matter because the kernels are learned

$$(f \star g)[k] = \sum_{m=-M}^{M} f[k+m]g[m]$$

$$(f * g)[k] = \sum_{m=-M}^{M} f[k-m]g[m]$$

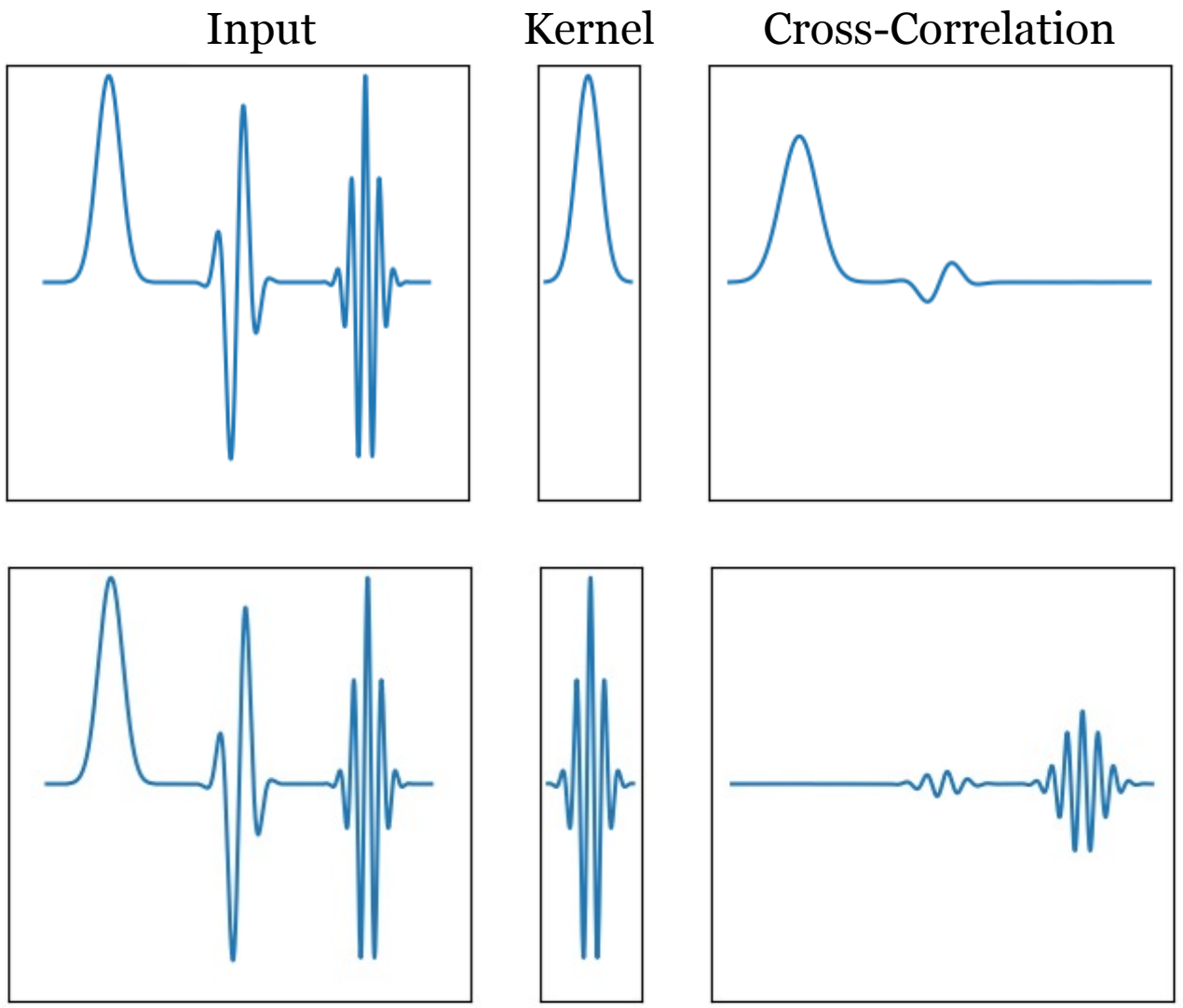
CONVOLUTION EMPHASIZES FEATURES THAT ARE LIKE THE KERNEL

Correlation is high when input has same shape as kernel

- $(f \star g)[k]$ uses a segment of the input signal around the k^{th} sample, specifically $f[k - M : k + M]$. Let's call this segment f_k.
- Given a kernel g, for what possible signal segment f_k is the cross-correlation $(f \star g)[k]$ highest?
- Recall that the correlation between two variables is highest when they are linearly related, specifically if $f_k = \alpha g + \beta$ then $r_{f_k g} = 1$ (e.g., $f_k = g$)
- Similarly, cross-correlation is highest when f_k and g have the same shape **if** all f_k have the same mean and standard deviation as each other
  - An f_k with a somewhat different shape and a larger magnitude could produce a higher cross-correlation

Convolutional layers prefer kernel-like inputs

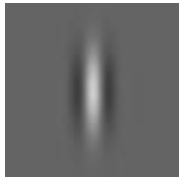
- For this reason, convolutional layers output higher values in response to inputs that match the shape of their kernels more closely
- In other words, they emphasize regions of the input that are shaped like the kernel

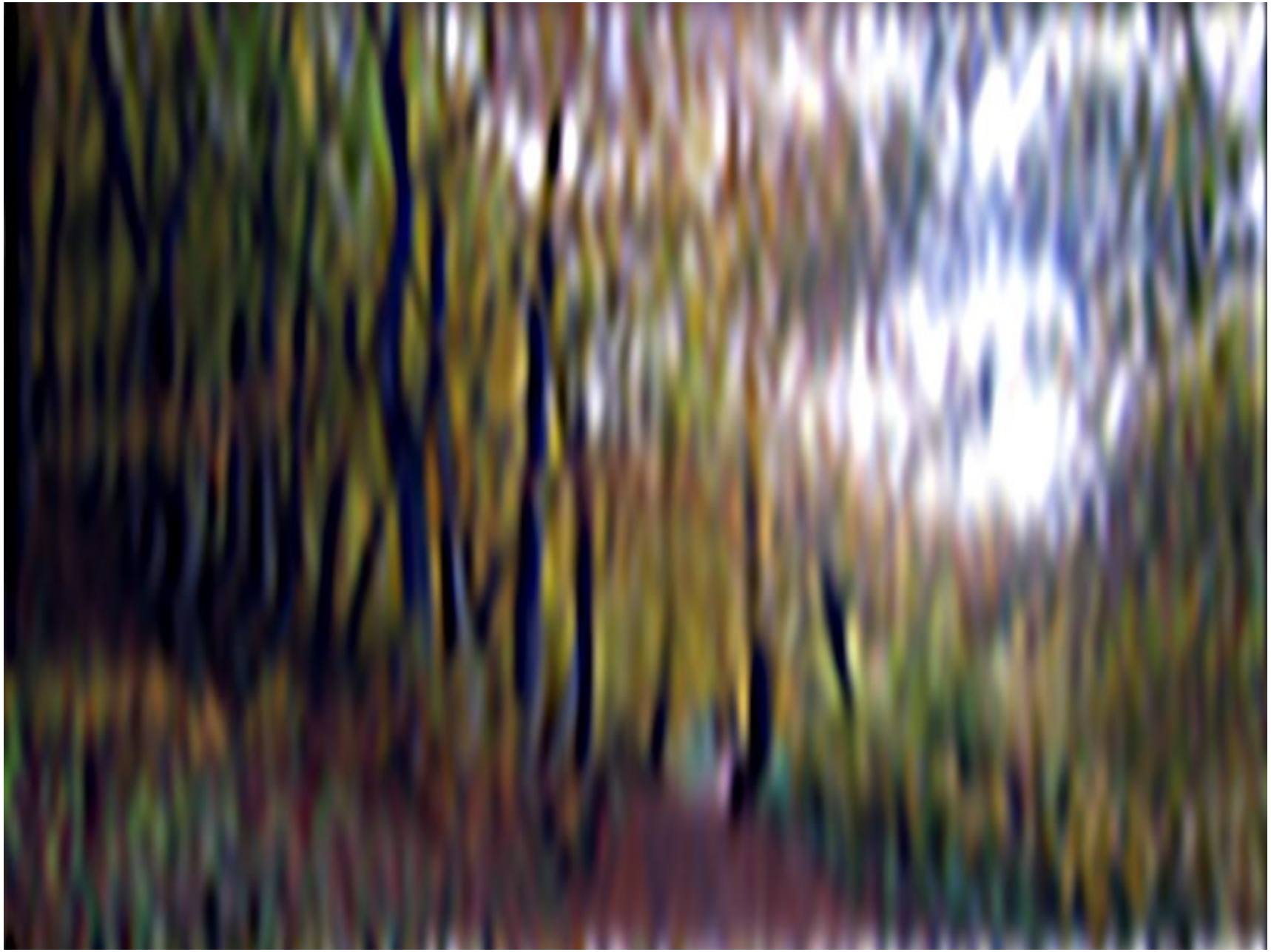


Input image

http://en.wikipedia.org/wiki/File:Stara_planina_suma.jpg

Kernel

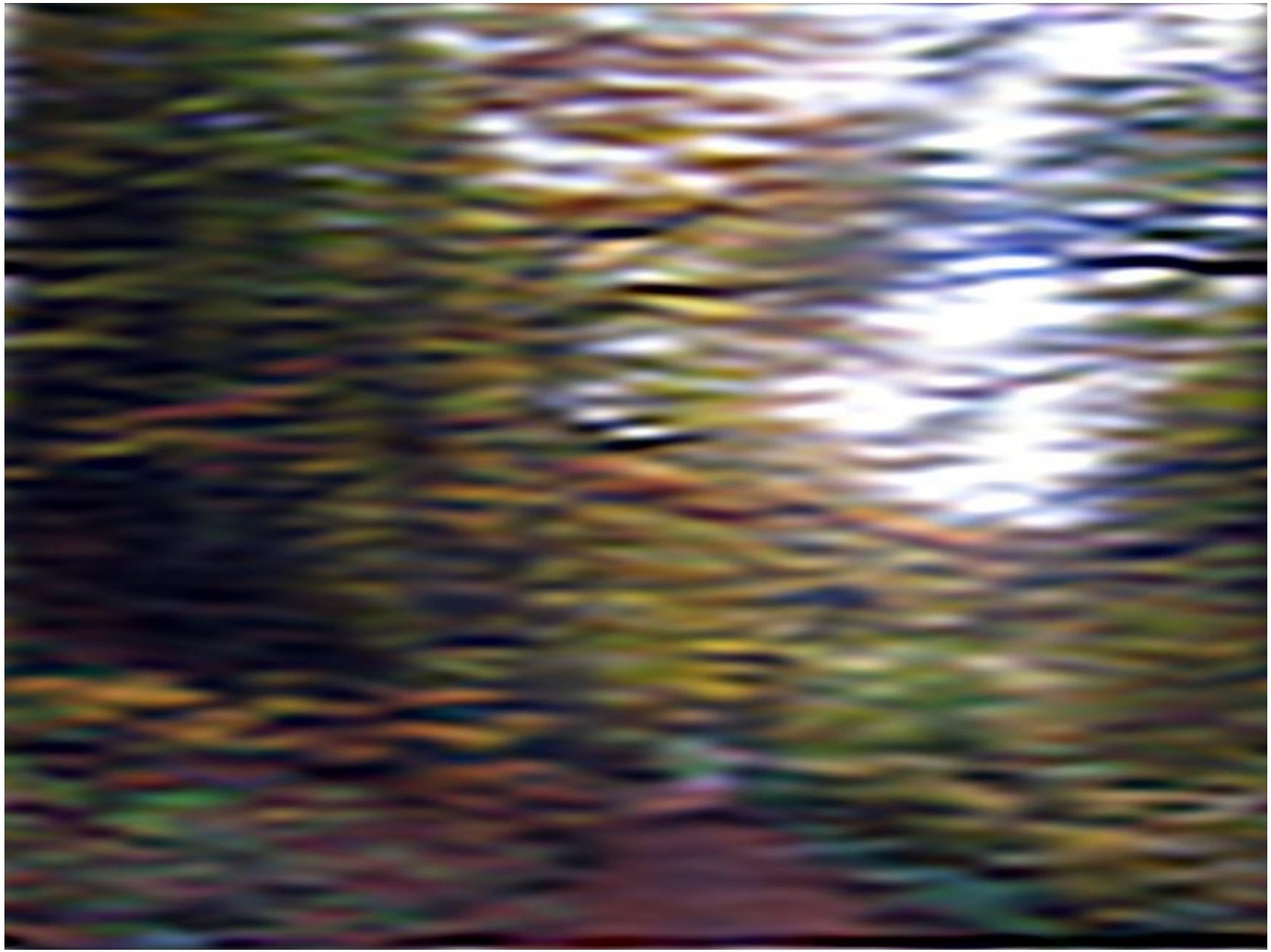




Input image

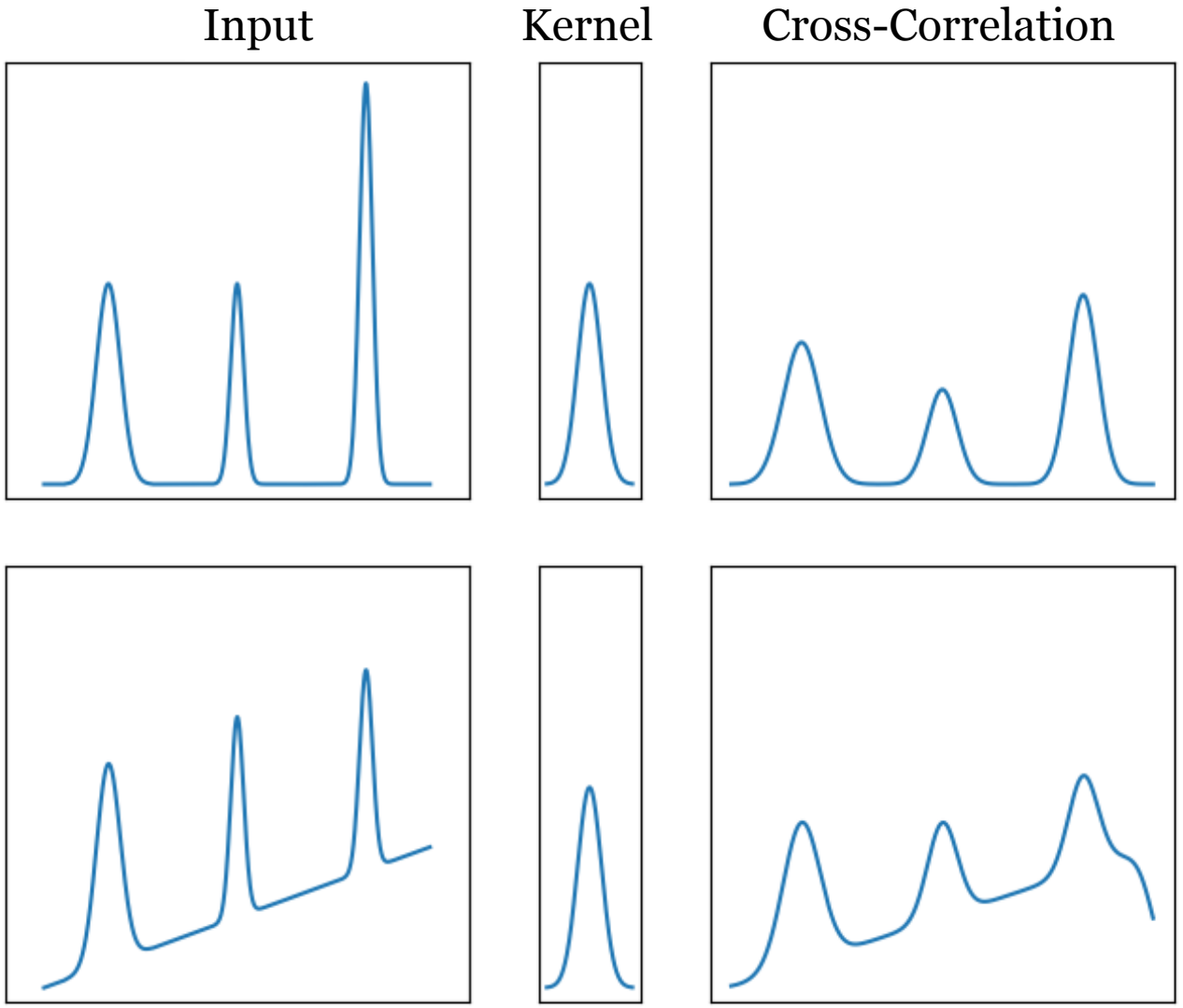
http://en.wikipedia.org/wiki/File:Stara_planina_suma.jpg

Kernel



Convolutional layers also respond to amplitude and bias

- An input that is highly correlated with the kernel can produce a strong response
- However, less-correlated inputs can also produce a strong response
  - A less-correlated input with larger amplitude can result in larger output
  - If the kernel mean is not zero, an input with large mean can also result in larger output



CONVOLUTION IS LINEAR

Homogeneity property

- A system is linear if it satisfies both the homogeneity and additivity properties
- Let S be a system and let c be a scalar constant
- S satisfies the homogeneity property if $S(x) = y$ implies that $S(cx) = cy$
- Convolution satisfies this property:

$$(cf * g)[k] = \sum_{m=-M}^{M} cf[k-m]g[m] = c \sum_{m=-M}^{M} f[k-m]g[m] = c(f * g)[k]$$

Additivity property

- S satisfies the additivity property if $S(x_1) = y_1$ and $S(x_2) = y_2$ implies that $S(x_1 + x_2) = y_1 + y_2$
- Convolution satisfies this property:

$$\begin{aligned}((f_1 + f_2) * g)[k] &= \sum_{m=-M}^{M} (f_1 + f_2)[k - m]g[m] \\ &= \sum_{m=-M}^{M} f_1[k - m]g[m] + \sum_{m=-M}^{M} f_2[k - m]g[m] \\ &= (f_1 * g)[k] + (f_2 * g)[k]\end{aligned}$$

CONVOLUTION IS LESS GENERAL THAN MATRIX MULTIPLICATION

Matrix multiplication can perform any linear map

- Any linear map from a n-dimensional space to a m-dimensional space can be performed by multiplication with a corresponding m by n matrix

Matrix multiplication can perform any linear map

- Let's show this …
- Consider a linear map f from vector space V to W
- Let $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\}$ be a basis of V and $\{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_m\}$ be a basis of W
  - Every $\boldsymbol{v} \in V$ is determined by coefficients as $\boldsymbol{v} = a_1\boldsymbol{v}_1 + \cdots + a_n\boldsymbol{v}_n$
  - Every $\boldsymbol{w} \in W$ is determined by coefficients as $\boldsymbol{w} = b_1\boldsymbol{w}_1 + \cdots + b_m\boldsymbol{w}_m$
- From the superposition principle,

$$f(\boldsymbol{v}) = f(a_1\boldsymbol{v}_1 + \cdots + a_n\boldsymbol{v}_n) = a_1 f(\boldsymbol{v}_1) + \cdots + a_n f(\boldsymbol{v}_n)$$

Matrix multiplication can perform any linear map

$$f(\boldsymbol{v}) = f(a_1\boldsymbol{v}_1 + \cdots + a_n\boldsymbol{v}_n) = a_1 f(\boldsymbol{v}_1) + \cdots + a_n f(\boldsymbol{v}_n)$$

- Each $f(\boldsymbol{v}_j)$ is a vector in W, so it can be written $f(\boldsymbol{v}_j) = b_{1j}\boldsymbol{w}_1 + \cdots + b_{mj}\boldsymbol{w}_m$, and

$$f(\boldsymbol{v}) = a_1(b_{11}\boldsymbol{w}_1 + \cdots + b_{m1}\boldsymbol{w}_m) + \cdots + a_n(b_{1n}\boldsymbol{w}_1 + \cdots + b_{mn}\boldsymbol{w}_m)$$

- Collecting terms and writing $f(\boldsymbol{v})$ in terms of its basis as $f(\boldsymbol{v}) = c_1\boldsymbol{w}_1 + \cdots + c_m\boldsymbol{w}_m$, we see that $c_1 = a_1 b_{11} + \cdots + a_n b_{1n}$, etc.
- Writing these terms in matrix-vector form,

$$\begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix}$$

Matrix multiplication can perform convolution

- Because convolution is linear, it can be performed by a matrix; for example, suppose signal f has length 5 and kernel g has length 3
- The corresponding matrix multiplication is,

$$\begin{bmatrix} (f*g)[1] \\ (f*g)[2] \\ (f*g)[3] \\ (\boldsymbol{f}*\boldsymbol{g})[\mathbf{4}] \\ (f*g)[5] \end{bmatrix} = \begin{bmatrix} g[0] & g[-1] & 0 & 0 & 0 \\ g[1] & g[0] & g[-1] & 0 & 0 \\ 0 & g[1] & g[0] & g[-1] & 0 \\ 0 & 0 & \boldsymbol{g}[\mathbf{1}] & \boldsymbol{g}[\mathbf{0}] & \boldsymbol{g}[\mathbf{-1}] \\ 0 & 0 & 0 & g[0] & g[-1] \end{bmatrix} \begin{bmatrix} f[1] \\ f[2] \\ \boldsymbol{f}[\mathbf{3}] \\ \boldsymbol{f}[\mathbf{4}] \\ \boldsymbol{f}[\mathbf{5}] \end{bmatrix}$$

- The bolded elements highlight calculation of the 4th element of the convolution result, $(f*g)[4] = \sum_{m=-1}^{1} f[4-m]g[m]$

Matrix multiplication can perform convolution

- For 1D convolution the matrix is banded, like this:

$$\begin{bmatrix} l & e & n & r & e & k & & & \\ & l & e & n & r & e & k & & \\ & & l & e & n & r & e & k & \\ & & & l & e & n & r & e & \\ & & & & \ddots & & & & \\ & & & & & & l & e & n & r & e & k \end{bmatrix}$$

"lenrek" is kernel spelled backwards, to remind us that the elements go in reverse order

- Convolution can't perform all linear maps because convolution matrices are constrained
  - Each row has the same entries (the kernel, in reverse), and each row's non-zero entries are shifted one position to the right compared to the row above
- Convolution performs a time-invariant (or space-invariant) linear map rather than a general linear map

CONVOLUTIONAL LAYERS WORK WITH SIGNALS, IMAGES, AND VOLUMES

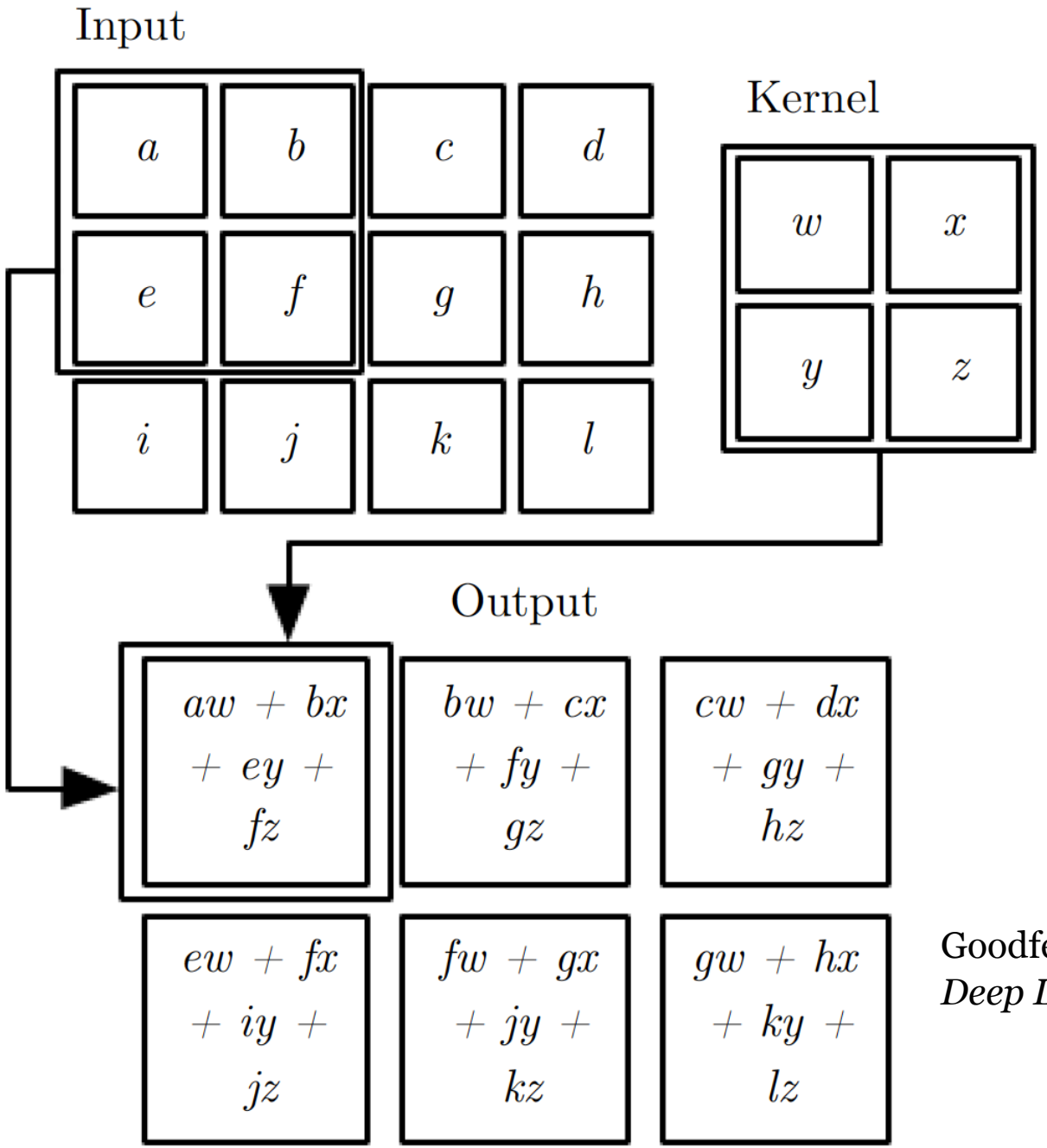
Convolution can be done in any number of dimensions

$$(f * g)[k] = \sum_{m=-M}^{M} f[k-m]g[m] \qquad \text{(1D)}$$

$$(f * g)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} f[i-m, j-n]g[m,n] \qquad \text{(2D)}$$

$$(f * g)[i,j,k] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} \sum_{o=-O}^{O} f[i-m, j-n, k-o]g[m,n,o] \qquad \text{(3D)}$$

- Example in two dimensions
- Question: Is this cross-correlation or convolution?



Goodfellow et al., *Deep Learning*

Library support

- Libraries such as PyTorch support one, two, and three-dimensional convolutional layers

`nn.Conv1d`	Applies a 1D convolution over an input signal composed of several input planes.
`nn.Conv2d`	Applies a 2D convolution over an input signal composed of several input planes.
`nn.Conv3d`	Applies a 3D convolution over an input signal composed of several input planes.

https://pytorch.org/docs/stable/nn.html#convolution-layers

CONVOLUTIONAL LAYERS HAVE MULTIPLE CHANNELS

Multiple channels per layer

- The input to a network is usually not a grid of scalars (such as a greyscale image) but a grid of vectors (such as colour image with red, green, and blue “channels”)
- The activation of a unit in the next layer is a **sum of convolutions** of each channel with a different kernel
- We typically do this multiple times to create multiple output channels, which are also called “feature maps”
- These provide multiple channels of input for the layer after that

Multiple channels per layer

3D input (input channel, vertical position, horizontal position)

$$S_{l,i,j} = \sum_{k=1}^{K} (I_k \star G_{k,l})_{i,j} = \sum_{k=1}^{K} \sum_{m=-M}^{M} \sum_{n=-N}^{N} I_{k,i+m,j+n} \, G_{k,l,m,n}$$

3D activation (output channel, vertical position, horizontal position)

4D kernel (input channel, output channel, vertical position, horizontal position)

Multiple channels per layer

2D convolution of k^{th} image and kernel channels

$$S_{l,i,j} = \sum_{k=1}^{K} (I_k \star G_{k,l})_{i,j} = \sum_{k=1}^{K} \sum_{m=-M}^{M} \sum_{n=-N}^{N} I_{k,i+m,j+n} \, G_{k,l,m,n}$$

Sum of 2D convolutions

Multiple channels per layer

Docs > torch.nn > Conv2d

CONV2D

CLASS `torch.nn.Conv2d(`*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

SOMETHING MUST BE DONE ABOUT THE EDGES

The problem

- Suppose a one-dimensional input has length K and elements are indexed from 1 to K
- This equation works fine for the middle of the signal, specifically $M < k \leq K - M$

$$(f \star g)[k] = \sum_{m=-M}^{M} f[k+m]g[m]$$

- For k at the edges of the signal, such as $k = 1$, we can't do the whole sum
- We must decide how to handle the edges

Full, same, and valid

- The obvious options are called valid, same, and full
- These names were popularized by Matlab

conv

Convolution and polynomial multiplication

R2023b

collapse all in page

Syntax

```
w = conv(u,v)
w = conv(u,v,shape)
```

shape — Subsection of convolution

`'full'` (default) | `'same'` | `'valid'`

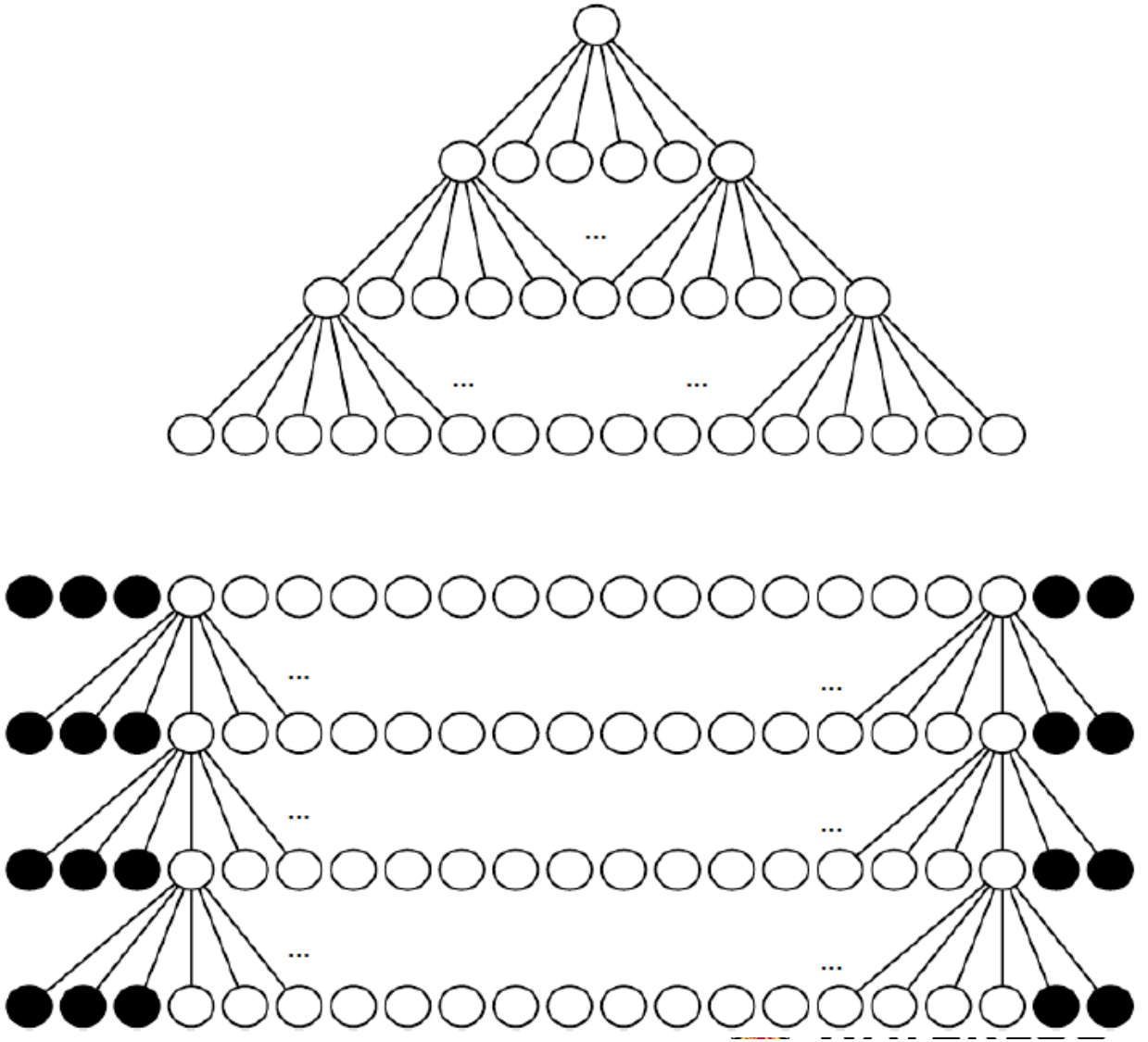
Subsection of the convolution, specified as `'full'`, `'same'`, or `'valid'`.

`'full'`	Full convolution (default).
`'same'`	Central part of the convolution of the same size as u.
`'valid'`	Only those parts of the convolution that are computed without the zero-padded edges. Using this option, `length(w)` is `max(length(u)-length(v)+1,0)`, except when `length(v)` is zero. If `length(v) = 0`, then `length(w) = length(u)`.

https://www.mathworks.com/help/matlab/ref/conv.html

Convolutional networks usually use valid or same

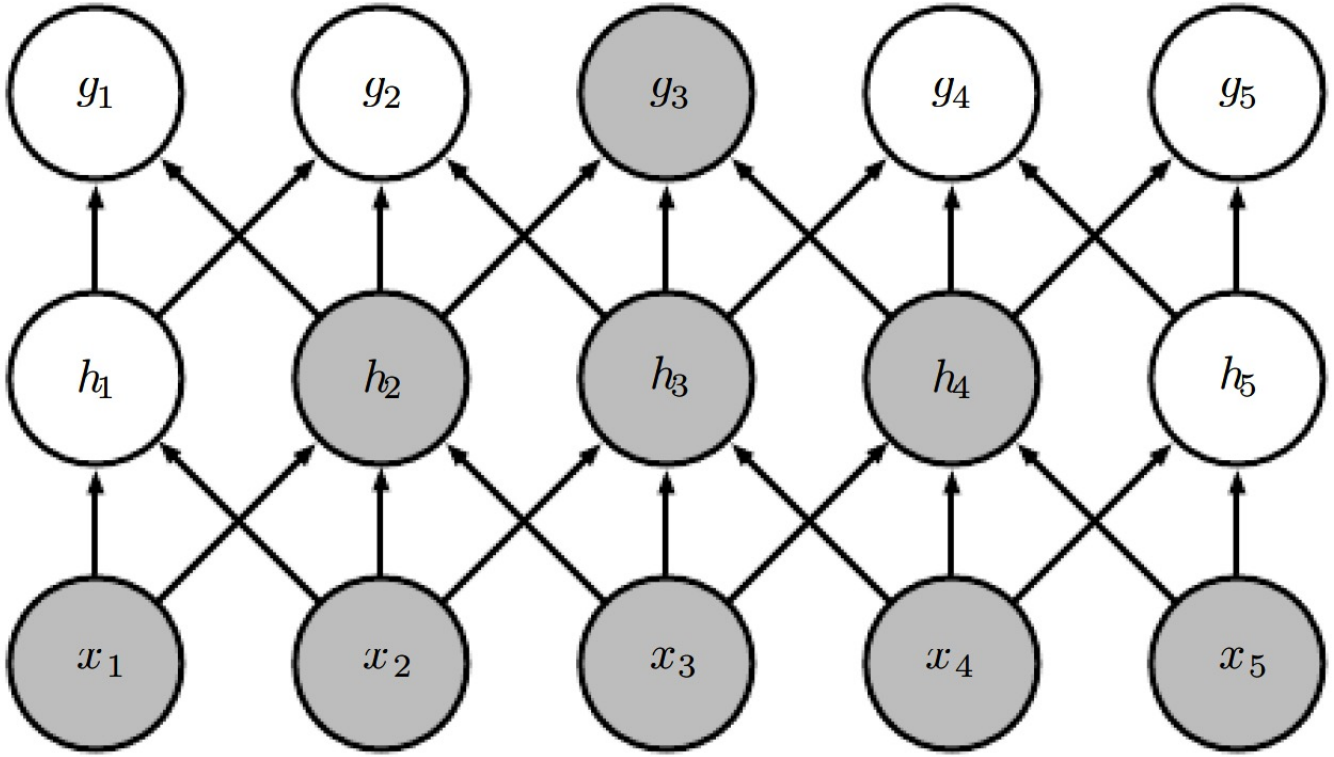
- Valid:
  - Only calculate result for $M < k \leq K - M$
  - Limited depth and/or small kernels (both may limit the power of the network).
- Same:
  - Pad edges of input with enough zeros that output is same size as input
  - Edge units have less input than centre units



Goodfellow et al., *Deep Learning*

Receptive fields

- Many units deeper in the network can be affected by edges
- This is because nodes farther from the input have larger “receptive fields” (regions of input that affect their activation)
- They are affected by a larger area of the input, despite having local direct connections with the immediately previous layer



Goodfellow et al., *Deep Learning*

CONVOLUTIONAL LAYERS ARE EFFICIENT IN TERMS OF COMPUTATION AND PARAMETERS

Parameter efficiency

- A convolutional layer has far fewer parameters than a fully-connected layer with the same number of neurons
- E.g., suppose 250x250 image input and one 250x250 feature map in the first layer
- If the layer were fully connected there would be $250^4 \approx 4$ billion weights, which would require a great deal of data to set without overfitting
- If the layer were not convolutional but had local connectivity in a 5x5 region, there would be $250^2 5^2 \approx 1.5$ million weights
- A convolutional layer with a single 5x5 kernel producing a 250x250 feature map would have 25 weights

Limitations

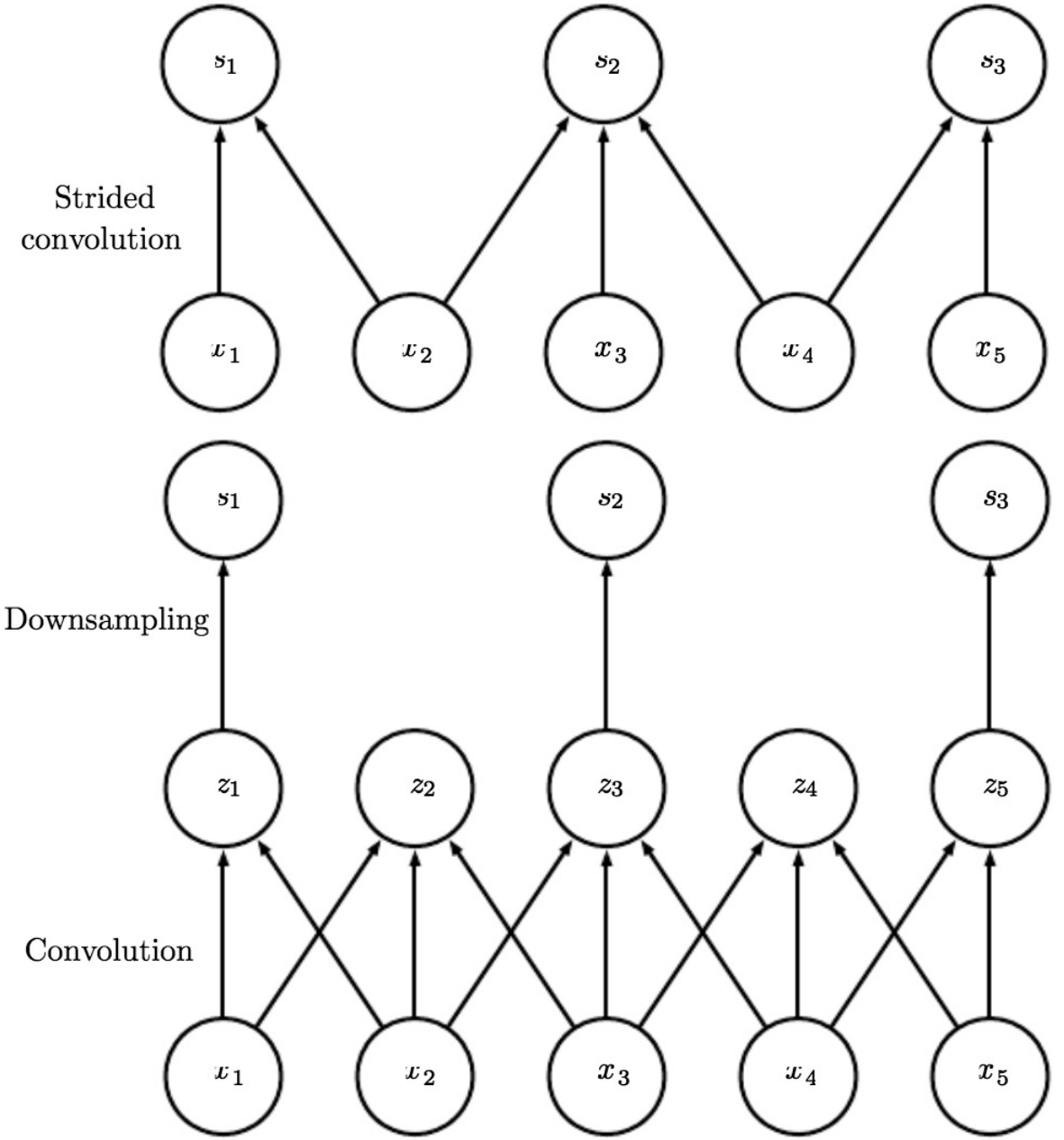
- This approach implicitly assumes that:
  - The most useful and/or consistent low-level features of an image are small
  - Input statistics are similar in different parts of the input

https://www.britannica.com

STRIDE AND DILATION SUBSAMPLE IN DIFFERENT WAYS

Stride

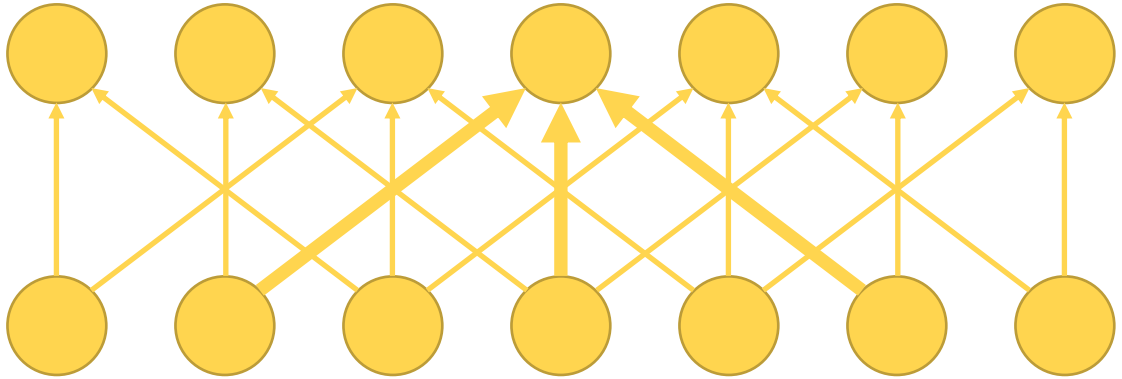
- The “stride” of an operation is the subsampling ratio of the output
- For example, a stride of two:
  - Calculates only every second element of the result
  - Reduces the size of the result by a factor of two in each dimension.
- Equivalent to (but more efficient than) calculating each element and then downsampling.
- Every input is used unless stride > kernel size (unusual)



Goodfellow et al., *Deep Learning*

Dilation

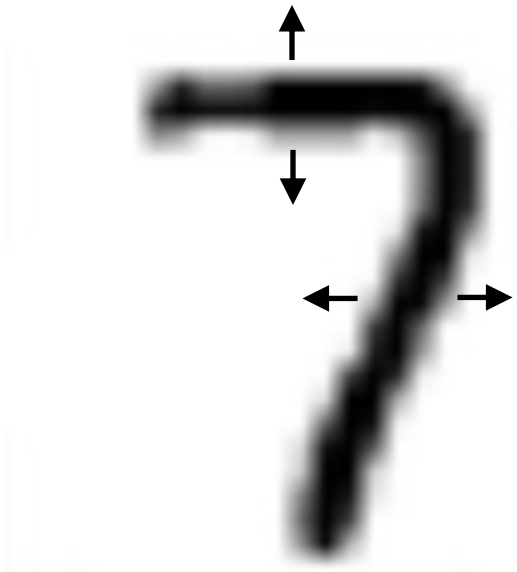
- The dilation factor of a convolution is the subsampling ratio of the input to each neuron
- Dilation allows a larger receptive field (broader context) without more parameters
- Shown here is a convolution with kernel size 3 and dilation factor 2
- Every input is used unless stride > 1



POOLING LAYERS INTRODUCE TRANSLATION INVARIANCE

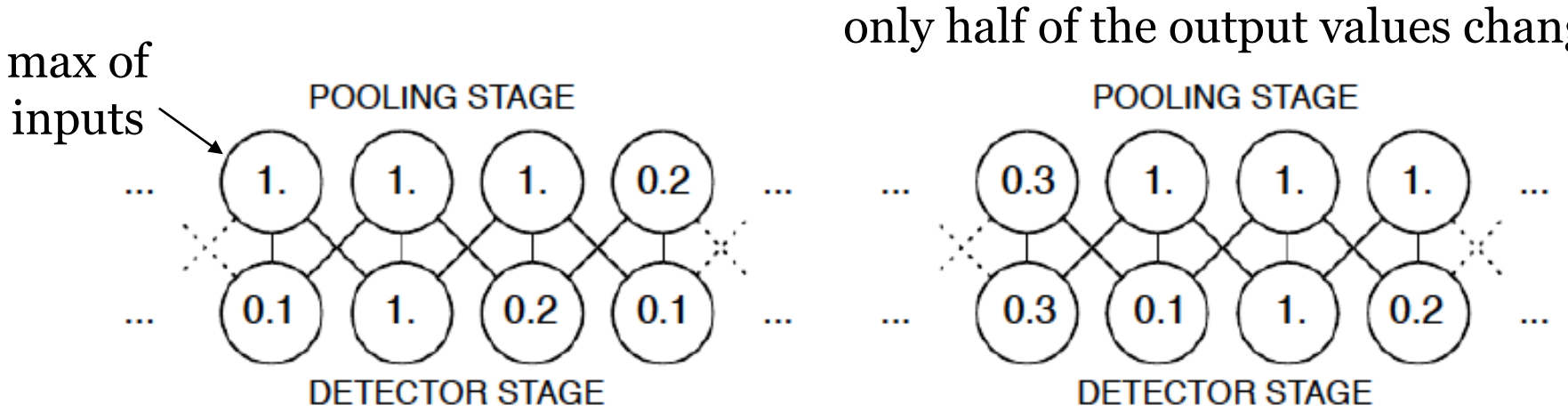
Image meaning is often invariant to small translations

- Example: This would be a seven regardless of small translations of the strokes (one or two pixels each way)



Max pooling introduces translation invariance

- Max pooling produces activity that is somewhat invariant to small translations
- In a max-pooling layer, each neuron gets input from a small region of the previous layer and its output is the maximum of its inputs
- Useful if the presence of a feature is more relevant than its precise location



Max pooling is usually strided

- Max pooling doesn't reduce the size of subsequent feature maps, but it is usually combined with a stride that does
- E.g., a 2x2 pool with stride of 2 in each dimension (horizontal & vertical) would reduce the next feature map resolution by a factor of 2 in each dimension

Docs > torch.nn > MaxPool2d

MAXPOOL2D

CLASS `torch.nn.MaxPool2d(`*`kernel_size`*`, `*`stride=None`*`, `*`padding=0`*`, `*`dilation=1`*`, `*`return_indices=False`*`, `*`ceil_mode=False`*`)` [SOURCE]

Applies a 2D max pooling over an input signal composed of several input planes.

Parameters

- **kernel_size** – the size of the window to take a max over
- **stride** – the stride of the window. Default value is `kernel_size`

Backpropagation through a max pooling layer

- Note that to backpropagate errors, the code must keep track of the index of the maximum input, e.g.,

$$z = \max(x_1, x_2)$$

$$\frac{\partial z}{\partial x_1} = \begin{cases} 1, & x_1 \geq x_2 \\ 0, & x_1 < x_2 \end{cases}$$

Summary

1. Convolution is related to correlation
2. Convolution emphasizes features that are like the kernel
3. Convolution is linear
4. Convolution is less general than matrix multiplication
5. Convolutional layers work with signals, images, and volumes
6. Convolutional layers have multiple channels
7. Something must be done about the edges
8. Convolutional layers are efficient in terms of computation and parameters
9. Stride and dilation subsample in different ways
10. Pooling layers introduce translation invariance