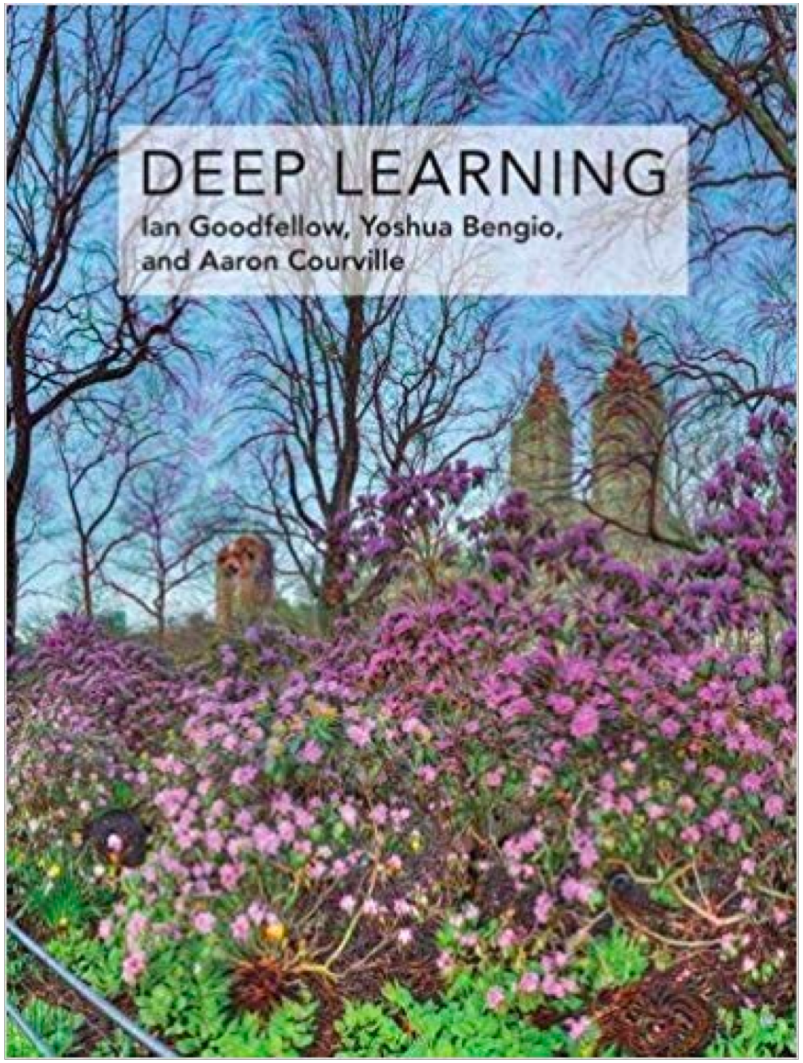
國立交通大學
National Chiao Tung University

Deep Learning
深度學習
Fall 2018

Convolutional Networks
(Chapter 9)

Prof. Chia-Han Lee
李佳翰 副教授



Convolutional networks

- Convolutional networks, also known as convolutional neural networks, or CNNs, are a specialized kind of neural network for processing data that has a known grid-like topology.

- Examples include time-series data, which can be thought of as a 1-D grid taking samples at regular time intervals, and image data, which can be thought of as a 2-D grid of pixels.

- The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

The convolution operation

- Suppose we are tracking the location of a spaceship with a laser sensor. Our laser sensor provides a single output $x(t)$, the position of the spaceship at time t. Both x and t are real valued, that is, we can get a different reading from the laser sensor at any instant in time.

- Suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average several measurements. Of course, more recent measurements are more relevant, so we will want this to be a weighted average that gives more weight to recent measurements. We can do this with a weighting function $\omega(a)$, where a is the age of a measurement.

The convolution operation

- If we apply such a weighted average operation at every moment, we obtain a new function s providing a smoothed estimate of the position of the spaceship:

$$s(t) = \int x(a)w(t-a)da. \tag{9.1}$$

This operation is called convolution, typically denoted as

$$s(t) = (x * w)(t). \tag{9.2}$$

- In our example, ω needs to be a valid probability density function, or the output will not be a weighted average. Also, ω needs to be 0 for all negative arguments, or it will look into the future. These limitations are particular to our example. In general, convolution is defined for any functions for which the above integral is defined and may be used for other purposes.

The convolution operation

- In convolutional network terminology, the first argument (the function x) to the convolution is often referred to as the input, and the second argument (the function ω) as the kernel. The output is sometimes referred to as the feature map.

- In our example, it might be more realistic to assume that our laser provides a measurement once per second. The time index t can then take on only integer values. If we now assume that x and ω are defined only on integer t, we can define the discrete convolution:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{9.3}$$

The convolution operation

- In machine learning applications, the input is usually a multidimensional array of data, and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. We will refer to these multidimensional arrays as tensors.

- Because each element of the input and kernel must be explicitly stored separately, we usually assume that these functions are zero everywhere but in the finite set of points for which we store the values. This means that in practice, we can implement the infinite summation as a summation over a finite number of array elements.

The convolution operation

- Finally, we often use convolutions over more than one axis at a time. For example, if we use a two-dimensional image I as our input, we probably also want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n). \tag{9.4}$$

- Convolution is commutative, meaning we can equivalently write

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{9.5}$$

Usually this formula is more straightforward to implement in a machine learning library, because there is less variation in the range of valid values of m and n.

The convolution operation

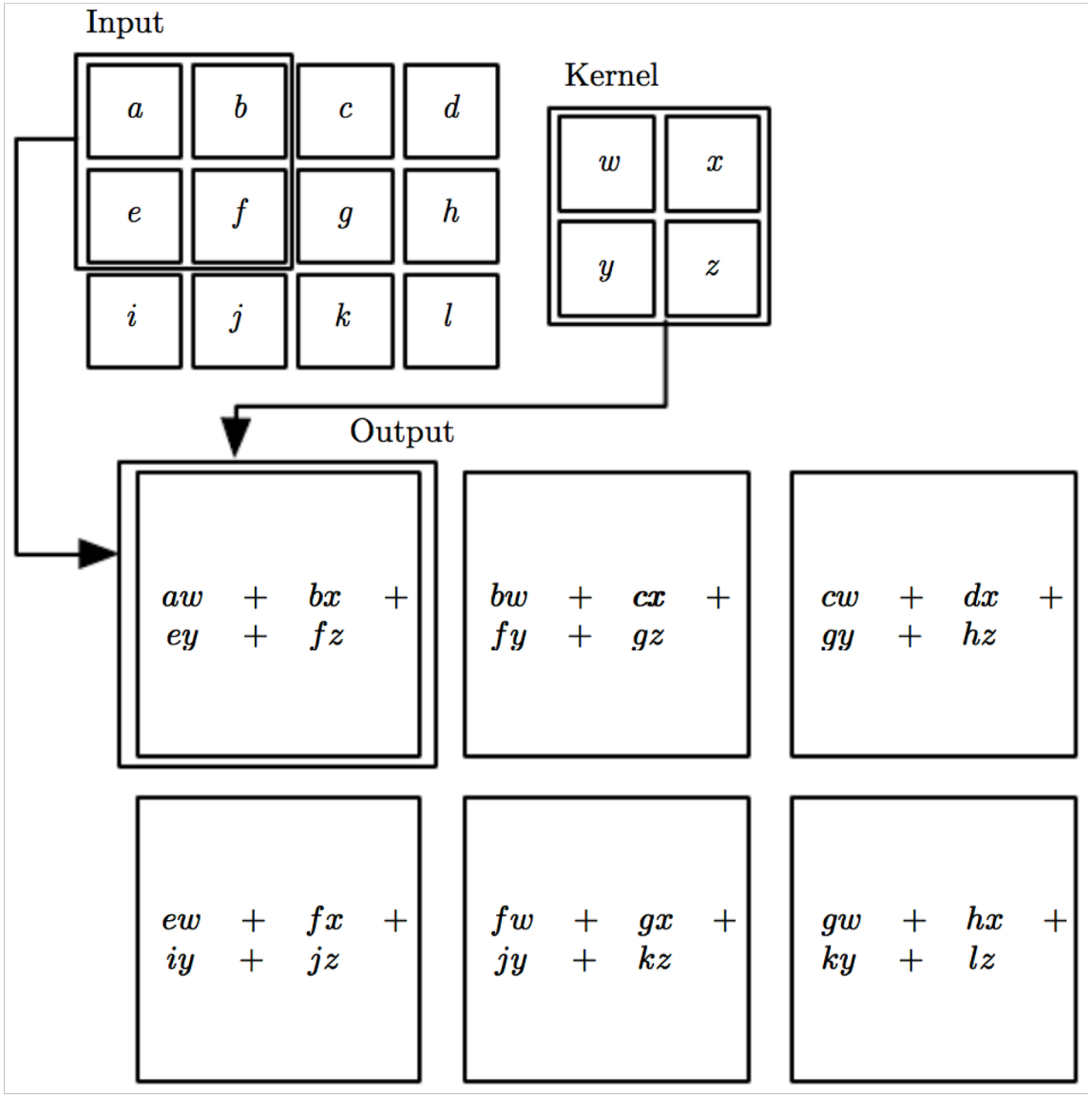
- While the commutative property is useful for writing proofs, it is not usually an important property of a neural network implementation. Instead, many neural network libraries implement a related function called the cross-correlation, which is the same as convolution but without flipping the kernel:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \quad (9.6)$$

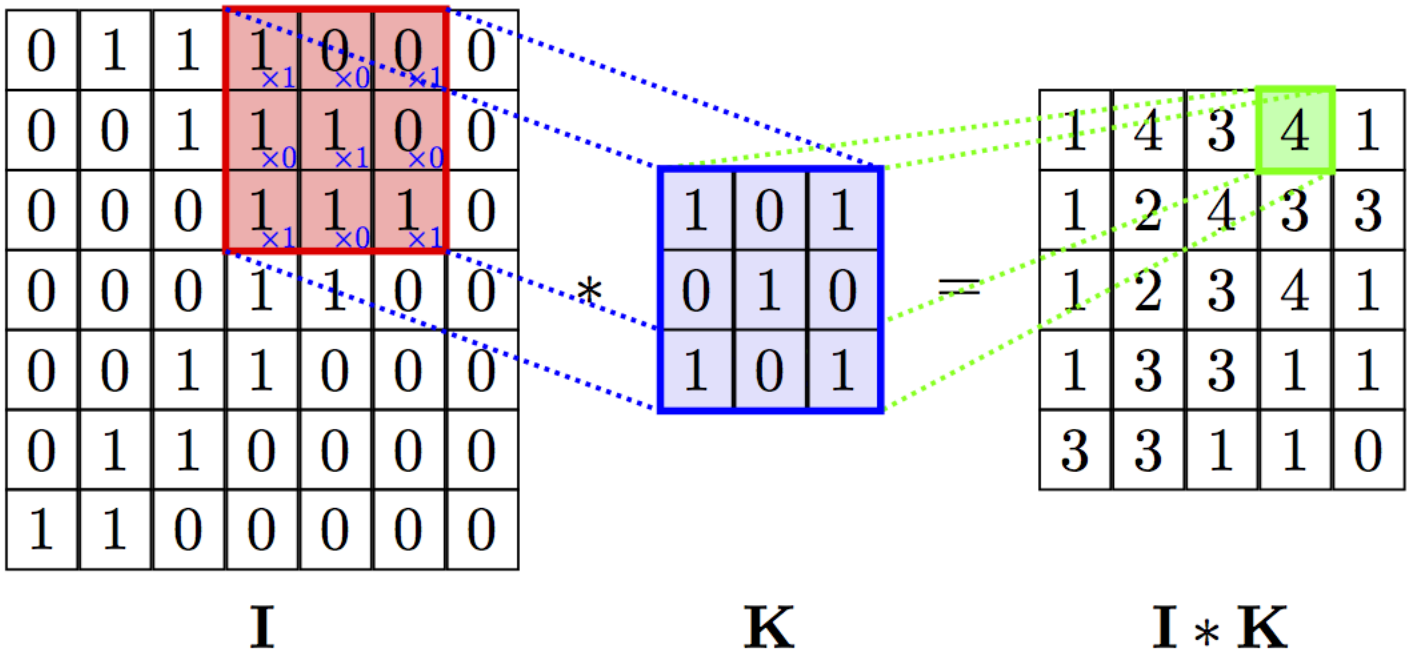
- Many machine learning libraries implement cross-correlation but call it convolution. We follow this convention of calling both operations convolution.

- Usually, convolution is used simultaneously with other functions, and the combination of these functions does not commute.

The convolution operation



The convolution operation



Deep learing for complete beginners: Using convolutional nets to recognise images, Cambridge Coding Academy

The convolution operation

- Discrete convolution can be viewed as multiplication by a matrix, but the matrix has several entries constrained to be equal to other entries.

- For example, for univariate discrete convolution, each row of the matrix is constrained to be equal to the row above shifted by one element. This is known as a Toeplitz matrix.

- In two dimensions, a doubly block circulant matrix corresponds to convolution.

$$A = \begin{bmatrix} a_0 & a_{-1} & a_{-2} & \dots & \dots & a_{-(n-1)} \\ a_1 & a_0 & a_{-1} & \ddots & & \vdots \\ a_2 & a_1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & a_{-1} & a_{-2} \\ \vdots & & \ddots & a_1 & a_0 & a_{-1} \\ a_{n-1} & \dots & \dots & a_2 & a_1 & a_0 \end{bmatrix} \quad C = \begin{bmatrix} c_0 & c_{n-1} & \dots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \dots & c_1 & c_0 \end{bmatrix}.$$

https://en.wikipedia.org/wiki/Toeplitz_matrix https://en.wikipedia.org/wiki/Circulant_matrix

The convolution operation

- In addition to these constraints that several elements be equal to each other, convolution usually corresponds to a very sparse matrix. This is because the kernel is usually much smaller than the input image.

- Any neural network algorithm that works with matrix multiplication and does not depend on specific properties of the matrix structure should work with convolution, without requiring any further changes to the neural network.

Motivation

- Convolution leverages three important ideas that can help improve a machine learning system: sparse interactions, parameter sharing and equivariant representations. Moreover, convolution provides a means for working with inputs of variable size.

- In traditional neural networks, every output unit interacts with every input unit. Convolutional networks, however, typically have sparse interactions (also called sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input.

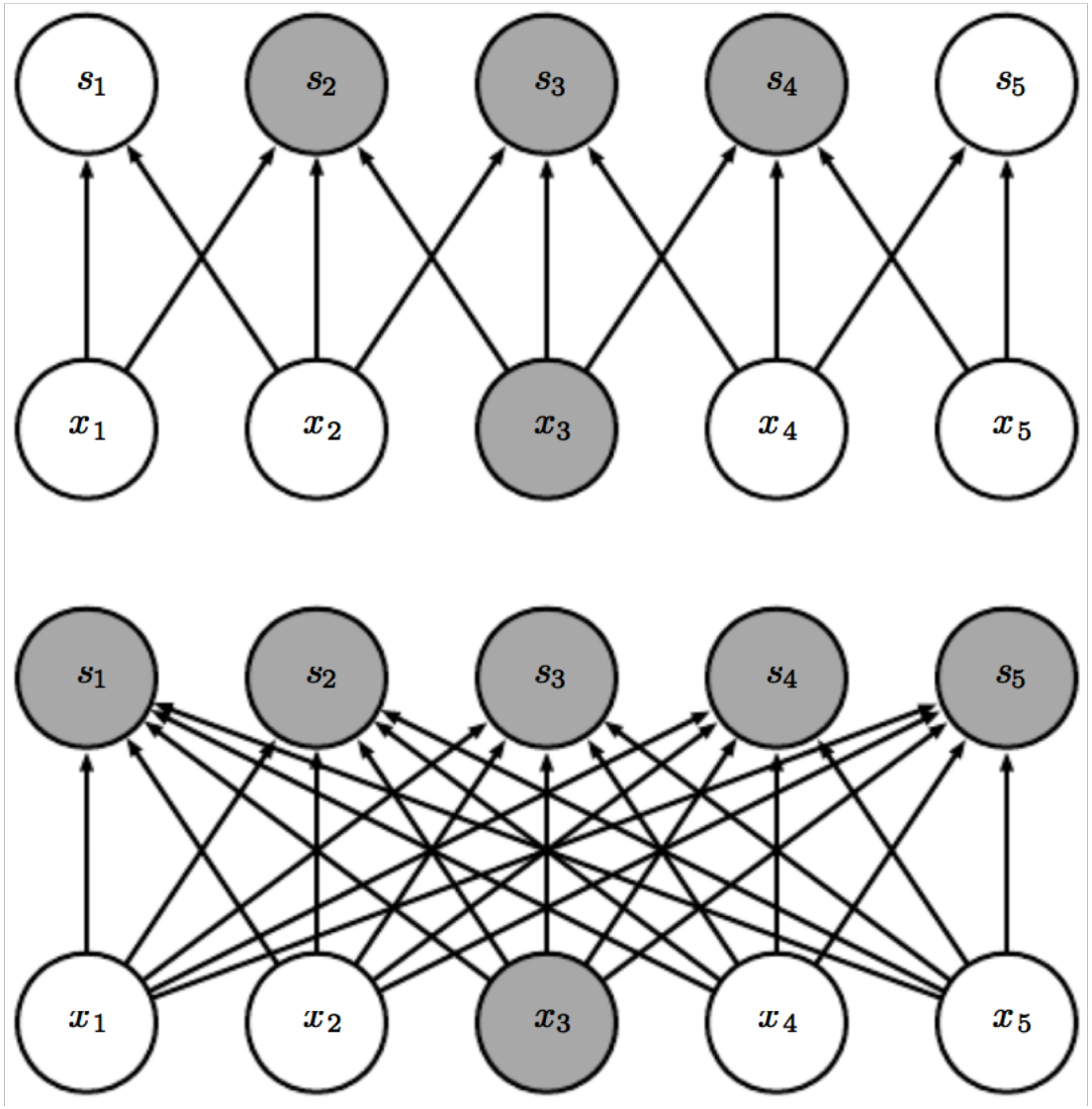
- For example, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels.

Motivation

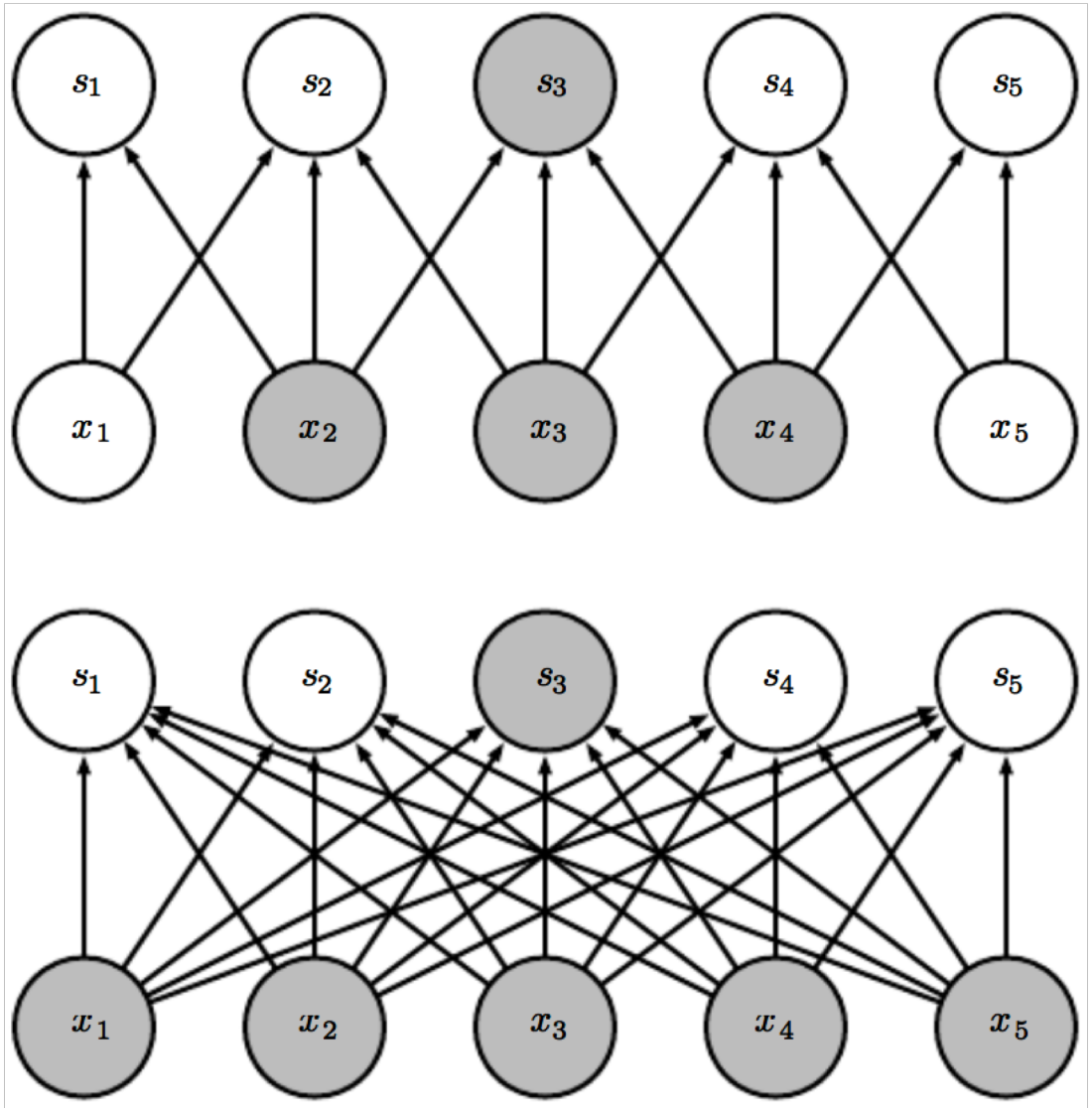
- This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. It also means that computing the output requires fewer operations.

- If there are m inputs and n outputs, then matrix multiplication requires $m \times n$ parameters, and the algorithms used in practice have $O(m \times n)$ runtime (per example). If we limit the number of connections each output may have to k, then the sparsely connected approach requires only $k \times n$ parameters and $O(k \times n)$ runtime. For practical applications, it is possible to obtain good performance on the machine learning task while keeping k several orders of magnitude smaller than m.

Motivation



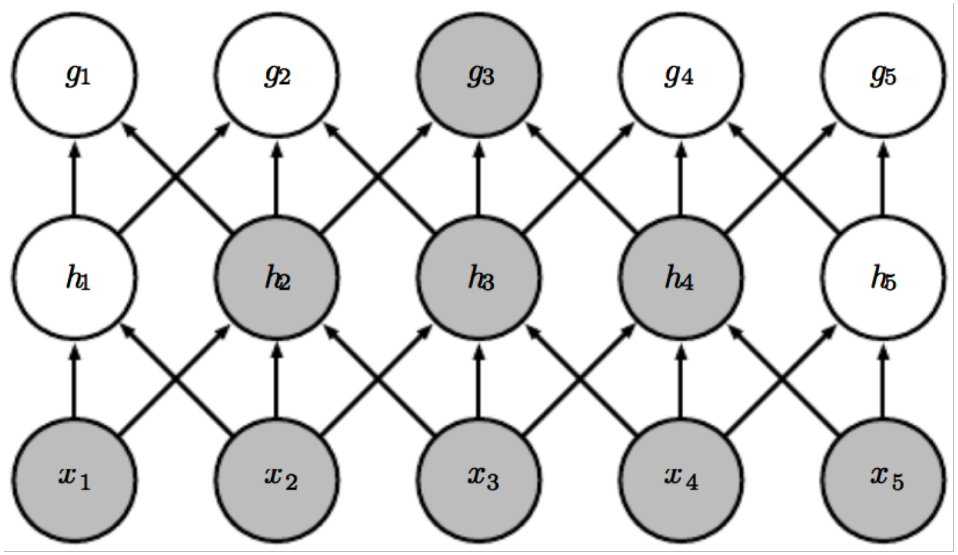
Motivation



Motivation

- In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input.

- This allows the network to efficiently describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions.



Motivation

- Parameter sharing refers to using the same parameter for more than one function in a model.
- In a traditional neural net, each element of the weight matrix is used exactly once when computing the output of a layer.
- In a convolutional neural net, each member of the kernel is used at every position of the input (except perhaps some of the boundary pixels, depending on the design decisions regarding the boundary).

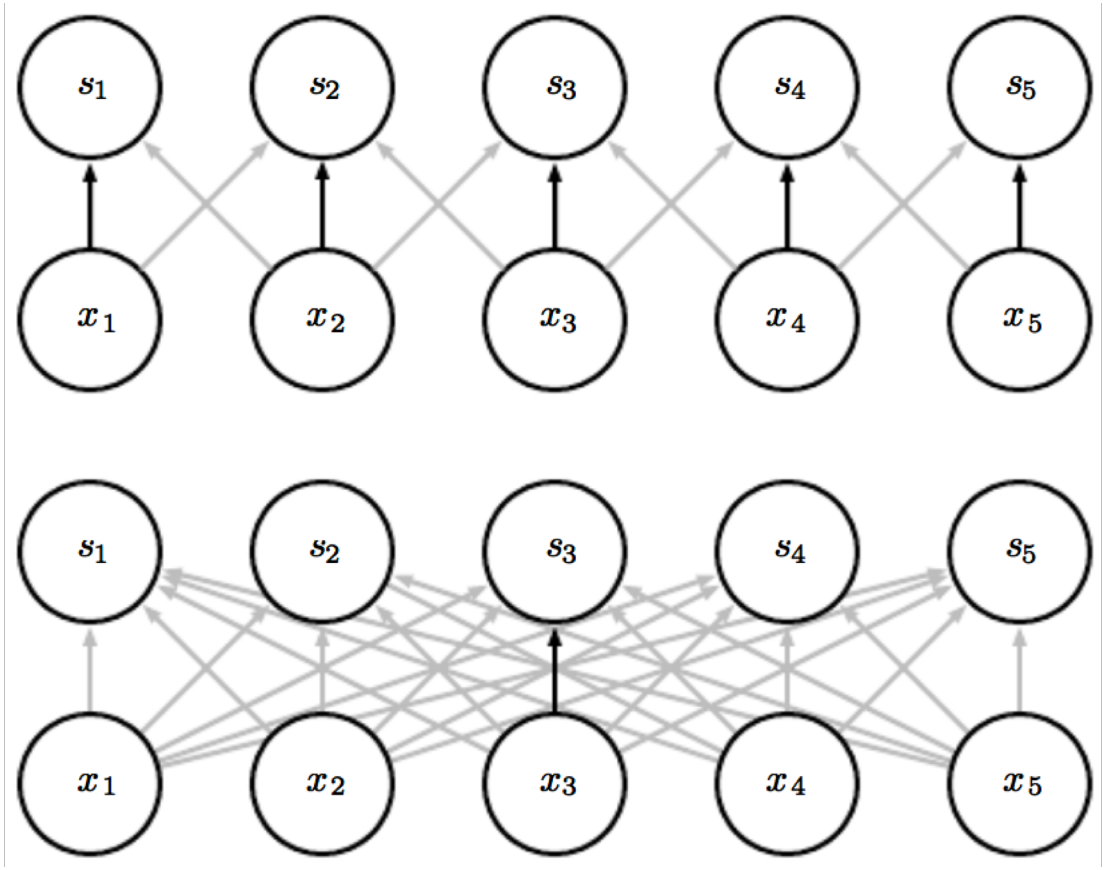
Motivation

- The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set.

- This does not affect the runtime of forward propagation—it is still $O(k \times n)$—but it does further reduce the storage requirements of the model to k parameters. Recall that k is usually several orders of magnitude smaller than m.

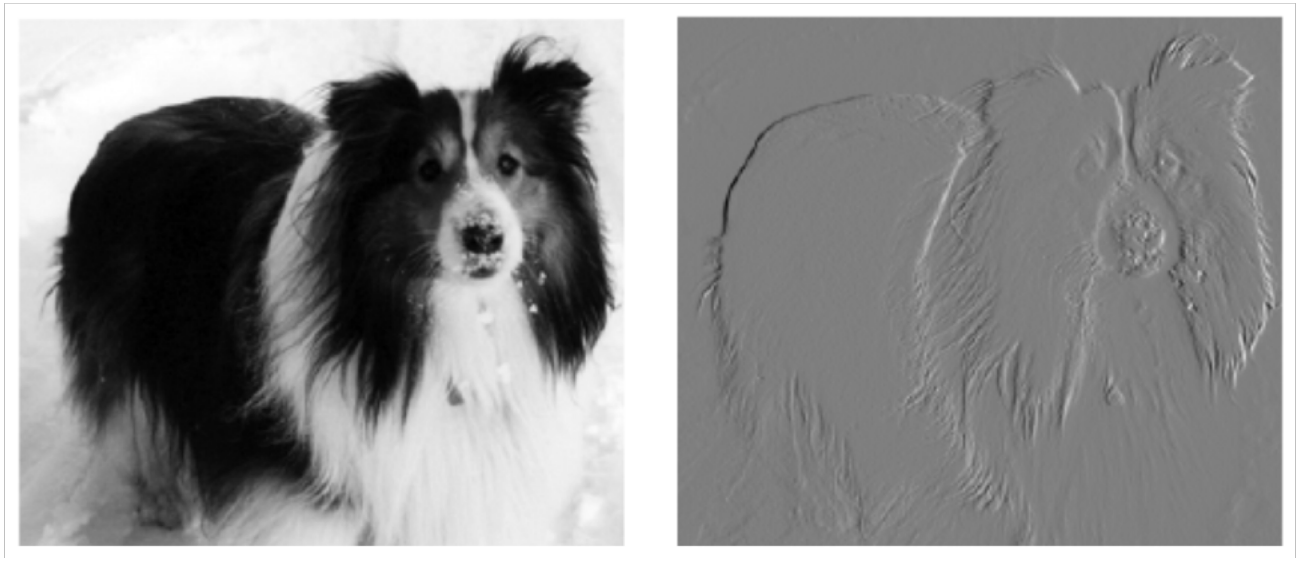
- Since m and n are usually roughly the same size, k is practically insignificant compared to $m \times n$. Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

Motivation



Motivation

As an example of both of these first two principles in action, figure 9.6 shows how sparse connectivity and parameter sharing can dramatically improve the efficiency of a linear function for detecting edges in an image.



Motivation

- In the case of convolution, the particular form of parameter sharing causes the layer to have a property called equivariance to translation.

- To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$.

- In the case of convolution, if we let g be any function that translates the input, that is, shifts it, then the convolution function is equivariant to g.

Motivation

- For example, let I be a function giving image brightness at integer coordinates. Let g be a function mapping one image function to another image function, such that $I' = g(I)$ is the image function with $I'(x, y) = I(x - 1, y)$. This shifts every pixel of I one unit to the right. If we apply this transformation to I, then apply convolution, the result will be the same as if we applied convolution to I', then applied the transformation g to the output.

- When processing time-series data, this means that convolution produces a sort of timeline that shows when different features appear in the input. If we move an event later in time in the input, the exact same representation of it will appear in the output, just later.

Motivation

- Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.

- This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations. For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image.

Motivation

- In some cases, we may not wish to share parameters across the entire image. For example, if we are processing images that are cropped to be centered on an individual's face, we probably want to extract different features at different locations—the part of the network processing the top of the face needs to look for eyebrows, while the part of the network processing the bottom of the face needs to look for a chin.

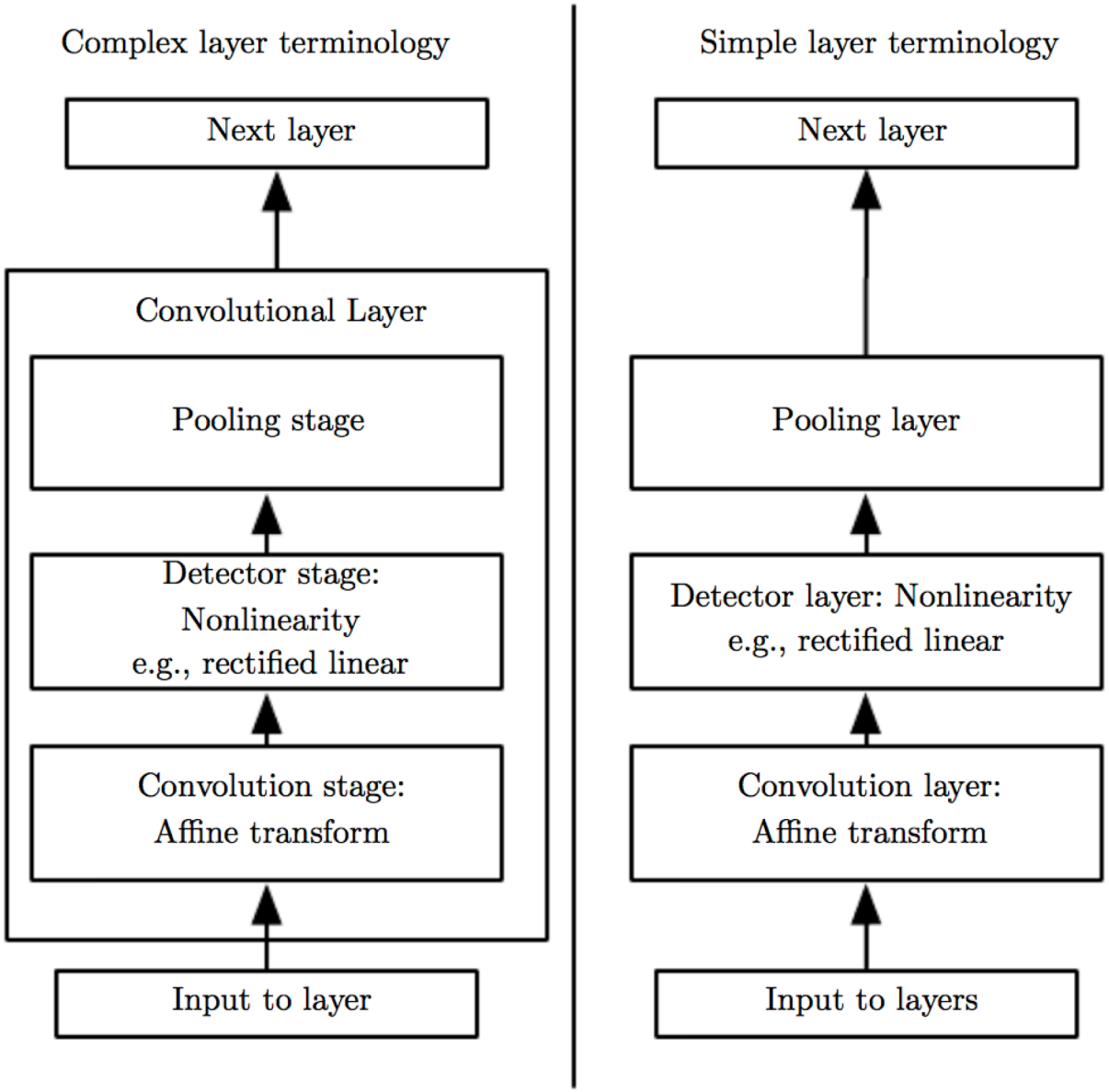
- Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image.

- Convolution enables processing of some kinds of data that cannot be processed by neural networks defined by matrix multiplication with a fixed-shape matrix.

Pooling

- A typical layer of a convolutional network consists of three stages.
- In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations.
- In the second stage, each linear activation is run through a nonlinear activation function, such as the rectified linear activation function. This stage is sometimes called the detector stage.
- In the third stage, we use a pooling function to modify the output of the layer further.

Pooling

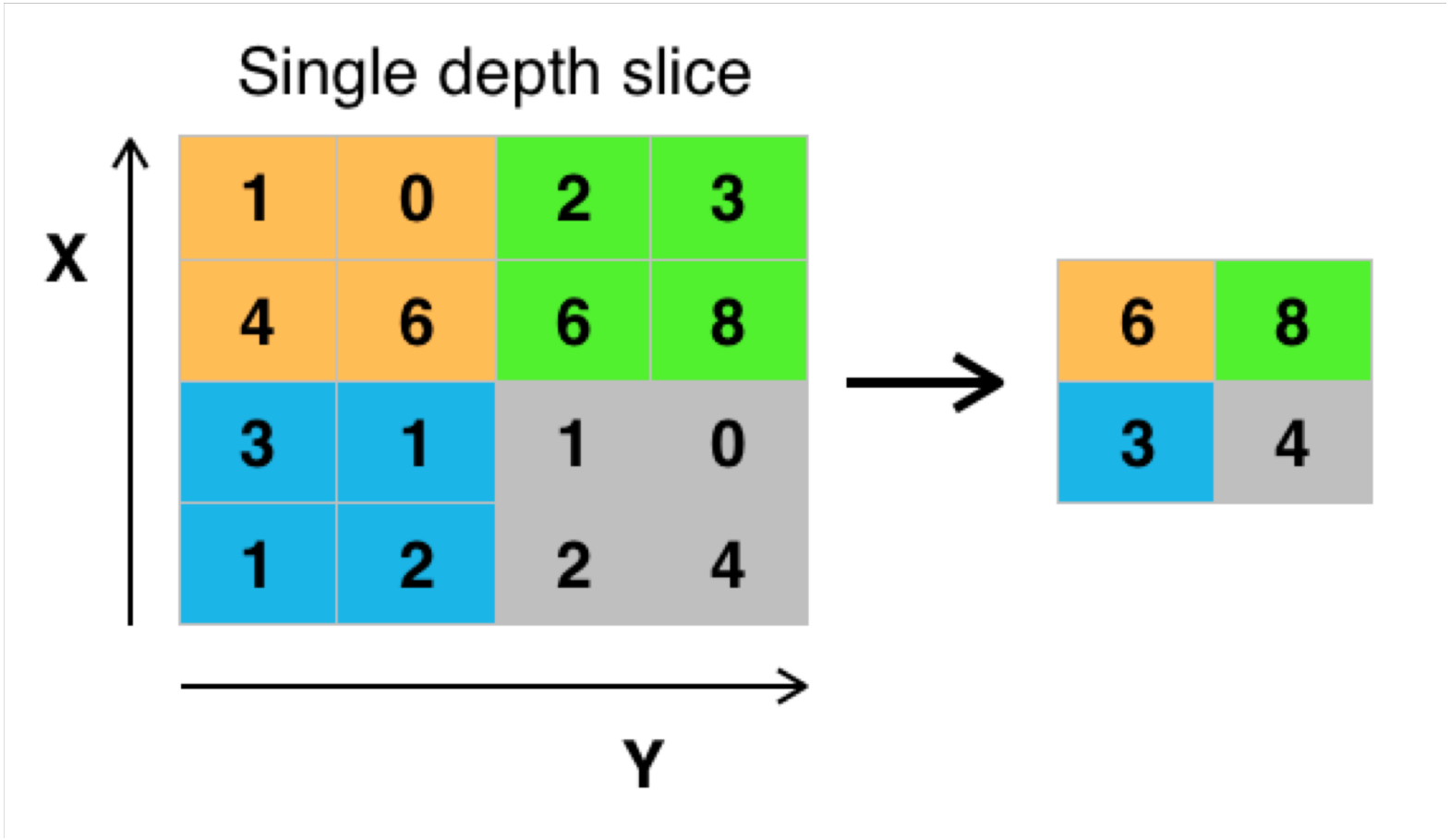

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage:
Nonlinearity
e.g., rectified linear
Convolution stage:
Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity
e.g., rectified linear
Convolution layer:
Affine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

Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- For example, the max pooling operation reports the maximum output within a rectangular neighborhood.
- Other popular pooling functions include the average of a rectangular neighborhood, the L^2 norm of a rectangular neighborhood, or a weighted average based on the distance from the central pixel.

Pooling

Max pooling



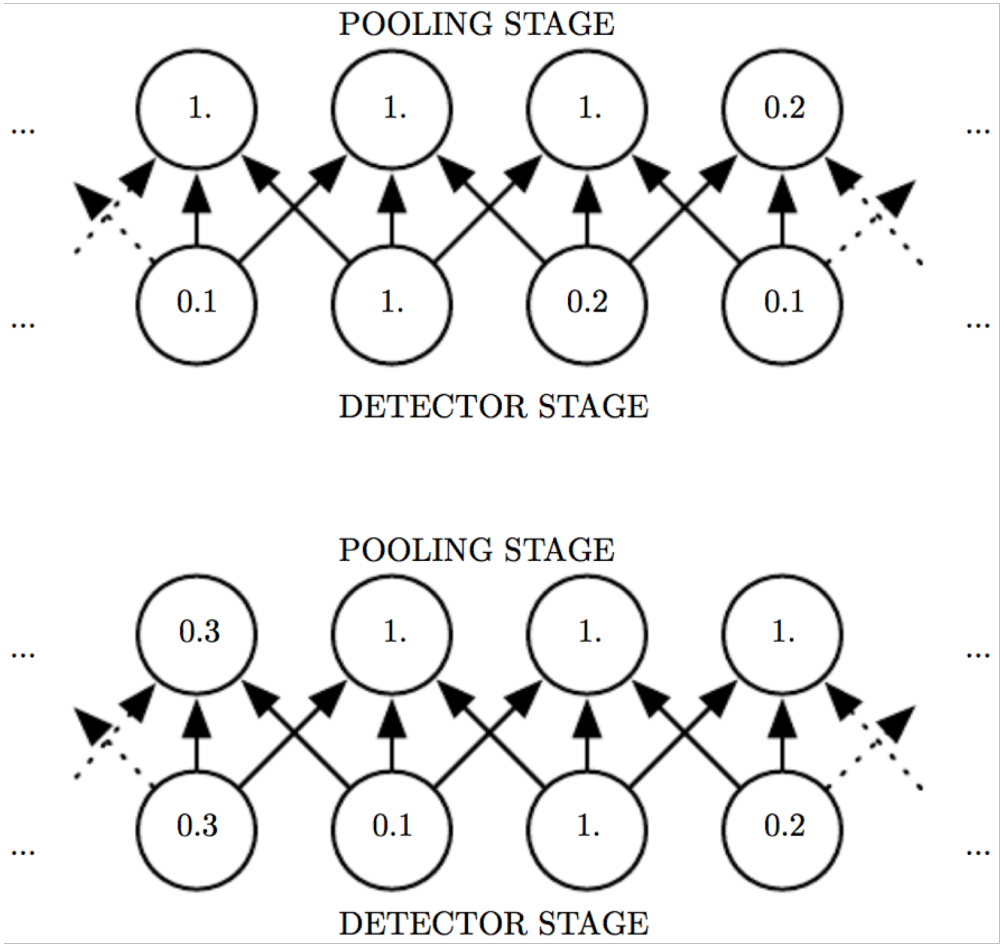
https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Max_pooling.png

Pooling

- In all cases, pooling helps to make the representation approximately invariant to small translations of the input.
- Invariance to translation means that if we translate the input by a small amount, the values of most of the pooled outputs do not change.
- Invariance to local translation can be a useful property if we care more about whether some feature is present than exactly where it is. For example, when determining whether an image contains a face, we need not know the location of the eyes with pixel-perfect accuracy, we just need to know that there is an eye on the left side of the face and an eye on the right side of the face.

Pooling

After the input has been shifted to the right by one pixel. Every value in the bottom row has changed, but only half of the values in the top row have changed, because the max pooling units are sensitive only to the maximum value in the neighborhood, not its exact location



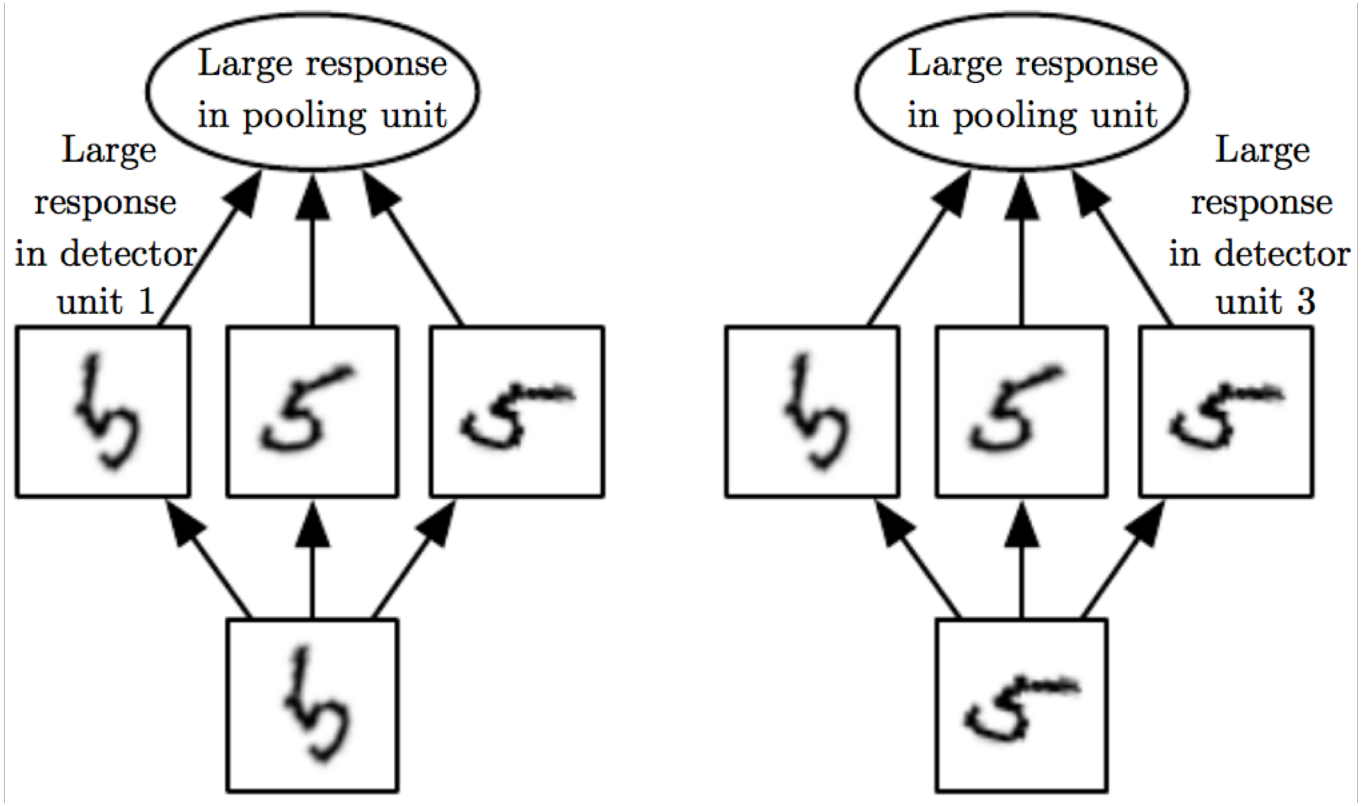
Pooling

- The use of pooling can be viewed as adding an infinitely strong prior that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency.

- In other contexts, it is more important to preserve the location of a feature. For example, if we want to find a corner defined by two edges meeting at a specific orientation, we need to preserve the location of the edges well enough to test whether they meet.

Pooling

- Pooling over spatial regions produces invariance to translation, but if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to.



Pooling

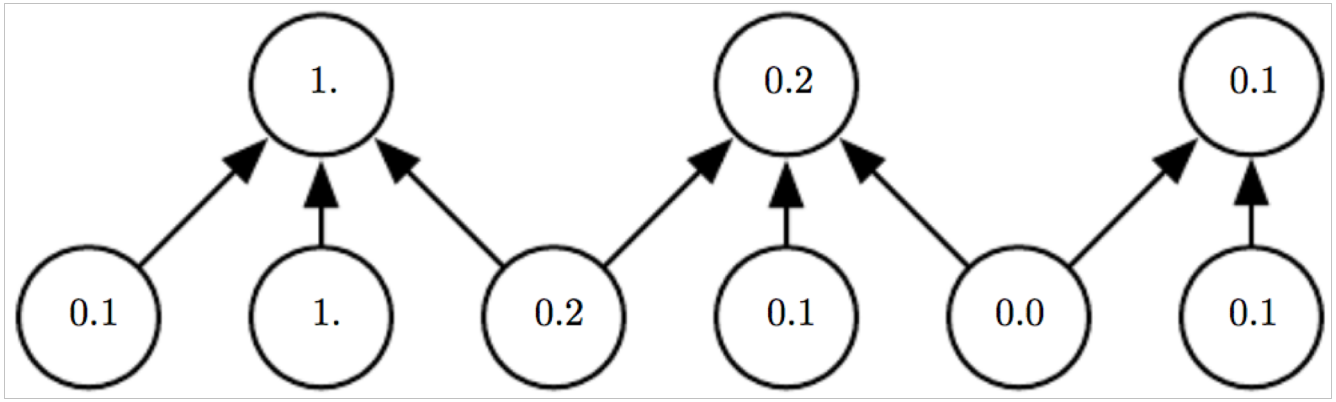
- Because pooling summarizes the responses over a whole neighborhood, it is possible to use fewer pooling units than detector units, by reporting summary statistics for pooling regions spaced k pixels apart rather than 1 pixel apart.

- This improves the computational efficiency of the network because the next layer has roughly k times fewer inputs.

- When the number of parameters in the next layer is a function of its input size (such as when the next layer is fully connected and based on matrix multiplication), this reduction in the input size can also result in improved statistical efficiency and reduced memory requirements for storing the parameters.

Pooling

Pooling with downsampling. Here we use max pooling with a pool width of three and a stride between pools of two.



Pooling

- For many tasks, pooling is essential for handling inputs of varying size.

- For example, if we want to classify images of variable size, the input to the classification layer must have a fixed size. This is usually accomplished by varying the size of an offset between pooling regions so that the classification layer always receives the same number of summary statistics regardless of the input size.

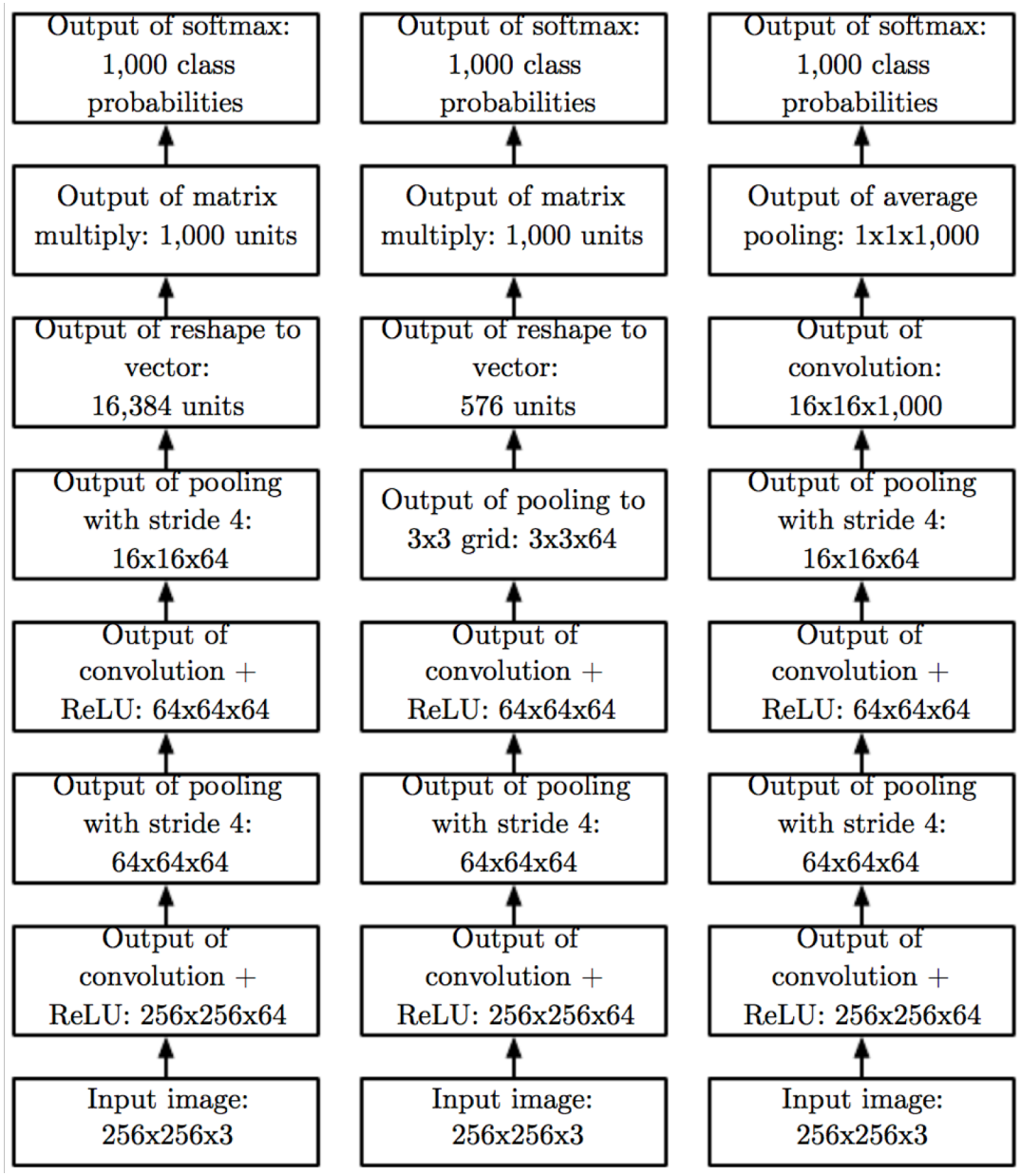
- For example, the final pooling layer of the network may be defined to output four sets of summary statistics, one for each quadrant of an image, regardless of the image size.

Pooling

- It is also possible to dynamically pool features together, for example, by running a clustering algorithm on the locations of interesting features. This approach yields a different set of pooling regions for each image.

- Another approach is to learn a single pooling structure that is then applied to all images.

Pooling


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

Convolution and pooling as an infinitely strong prior

- An infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data give to those values.

- We can imagine a convolutional net as being similar to a fully connected net, but with an infinitely strong prior over its weights.

- This infinitely strong prior says that the weights for one hidden unit must be identical to the weights of its neighbor but shifted in space. The prior also says that the weights must be zero, except for in the small, spatially contiguous receptive field assigned to that hidden unit.

Convolution and pooling as an infinitely strong prior

- Overall, we can think of the use of convolution as introducing an infinitely strong prior probability distribution over the parameters of a layer.

- This prior says that the function the layer should learn contains only local interactions and is equivariant to translation. Likewise, the use of pooling is an infinitely strong prior that each unit should be invariant to small translations.

Convolution and pooling as an infinitely strong prior

- Convolution and pooling can cause underfitting. If a task relies on preserving precise spatial information, then using pooling on all features can increase the training error.

- Some convolutional network architectures are designed to use pooling on some channels but not on others, in order to get both highly invariant features and features that will not underfit when the translation invariance prior is incorrect.

- When a task involves incorporating information from very distant locations in the input, then the prior imposed by convolution may be inappropriate.

Variants of the basic convolution function

- When we refer to convolution in the context of neural networks, we usually actually mean an operation that consists of many applications of convolution in parallel.

- This is because convolution with a single kernel can extract only one kind of feature, albeit at many spatial locations. Usually we want each layer of our network to extract many kinds of features, at many locations.

- Additionally, the input is usually not just a grid of real values. Rather, it is a grid of vector-valued observations. For example, a color image has a red, green and blue intensity at each pixel.

Variants of the basic convolution function

- When working with images, we usually think of the input and output of the convolution as being 3-D tensors, with one index into the different channels and two indices into the spatial coordinates of each channel.

- Software implementations usually work in batch mode, so they will actually use 4-D tensors, with the fourth axis indexing different examples in the batch, but we will omit the batch axis in our description here for simplicity.

- Because convolutional networks usually use multichannel convolution, the linear operations they are based on are not guaranteed to be commutative, even if kernel flipping is used.

Variants of the basic convolution function

- Assume we have a 4-D kernel tensor $\mathbf{K}$ with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit.

- Assume our input consists of observed data $\mathbf{V}$ with element $V_{i,j,k}$ giving the value of the input unit within channel i at row j and column k.

- Assume our output consists of $\mathbf{Z}$ with the same format as $\mathbf{V}$.

Variants of the basic convolution function

- If $\mathbf{Z}$ is produced by convolving $\mathbf{K}$ across $\mathbf{V}$ without flipping $\mathbf{K}$, then

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}, \tag{9.7}$$

where the summation over l, m and n is over all values for which the tensor indexing operations inside the summation are valid.

- In linear algebra notation, we index into arrays using a 1 for the first entry. This necessitates the −1 in the above formula. Programming languages such as C and Python index starting from 0, rendering the above expression even simpler.

Variants of the basic convolution function

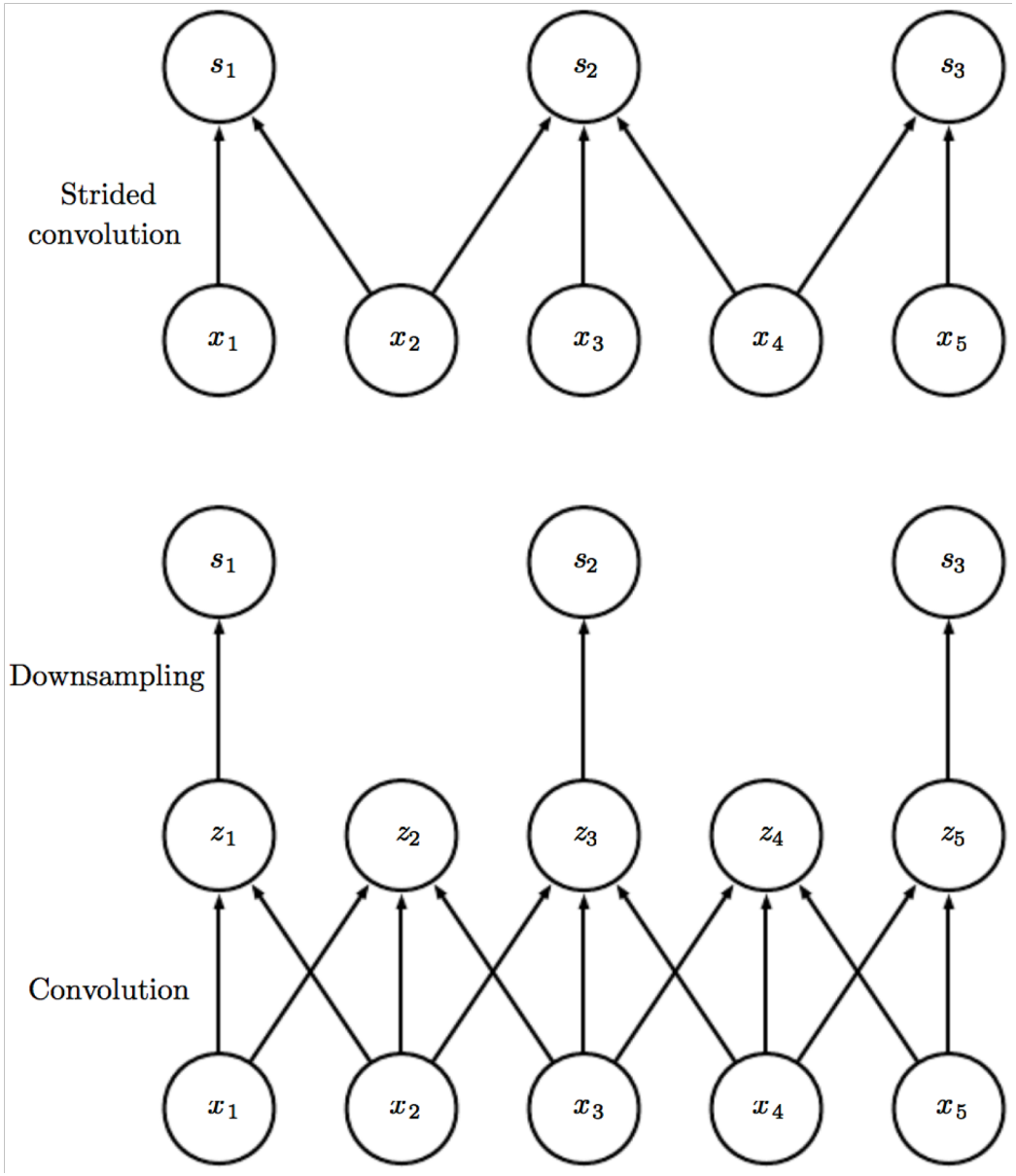
- We may want to skip over some positions of the kernel to reduce the computational cost (at the expense of not extracting our features as finely). We can think of this as downsampling the output of the full convolution function.

- If we want to sample only every s pixels in each direction in the output, then we can define a downsampled convolution function c such that

$$Z_{i,j,k} = c(\mathbf{K}, \mathbf{V}, s)_{i,j,k} = \sum_{l,m,n} \left[V_{l,(j-1)\times s+m,(k-1)\times s+n} K_{i,l,m,n} \right]. \quad (9.8)$$

- We refer to s as the stride of this downsampled convolution. It is also possible to define a separate stride for each direction of motion.

Variants of the basic convolution function



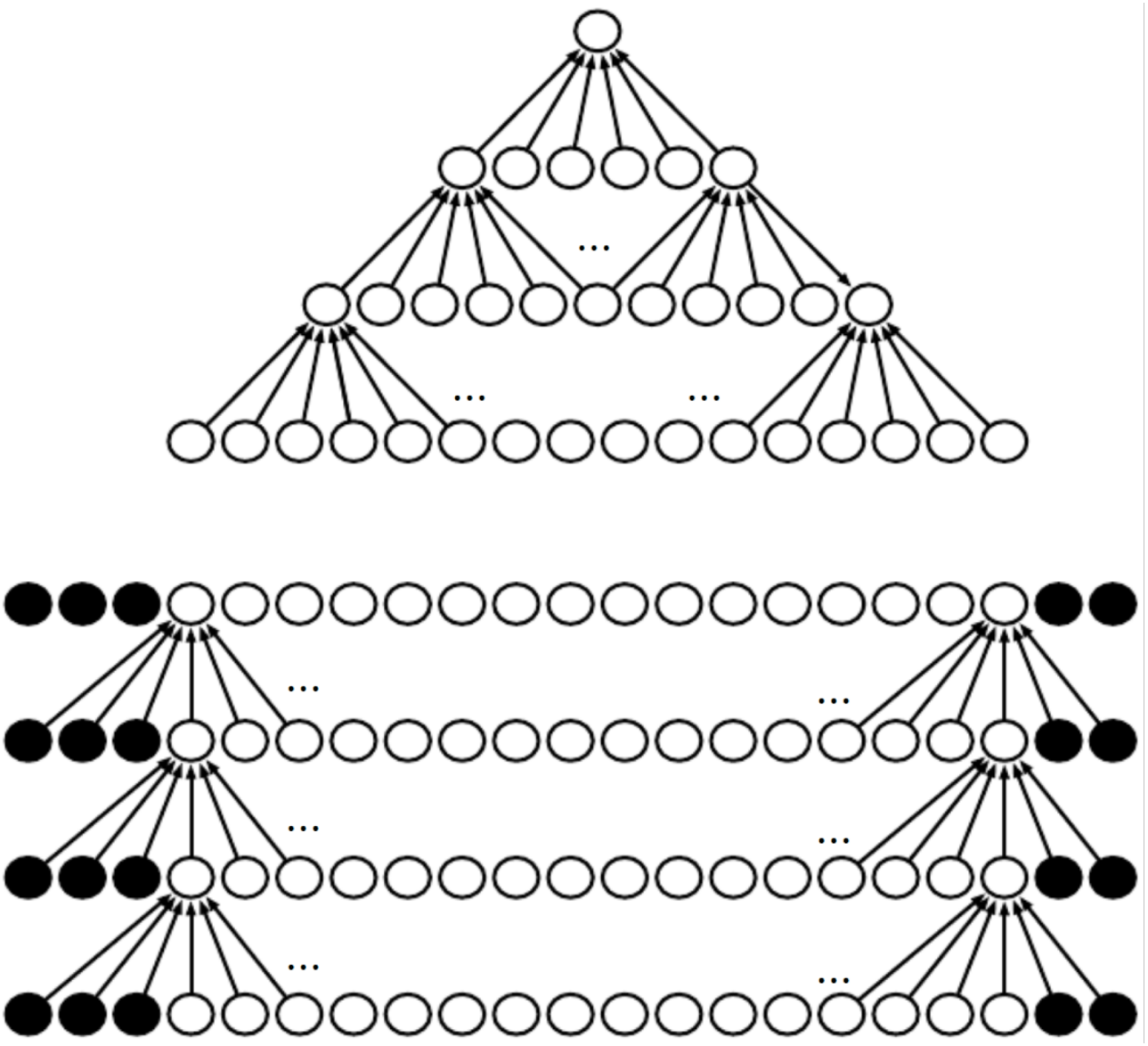
Variants of the basic convolution function

- One essential feature of any convolutional network implementation is the ability to implicitly zero pad the input $\mathbf{V}$ to make it wider. Without this feature, the width of the representation shrinks by one pixel less than the kernel width at each layer.

- Zero padding the input allows us to control the kernel width and the size of the output independently.

- Without zero padding, we are forced to choose between shrinking the spatial extent of the network rapidly and using small kernels—both scenarios that significantly limit the expressive power of the network.

Variants of the basic convolution function



Variants of the basic convolution function

- There are three special cases of zero-padding.
- One is the extreme case in which no zero padding is used whatsoever, and the convolution kernel is allowed to visit only positions where the entire kernel is contained entirely within the image.
- In MATLAB terminology, this is called valid convolution. In this case, all pixels in the output are a function of the same number of pixels in the input, so the behavior of an output pixel is somewhat more regular.
- However, the size of the output shrinks at each layer. If the input image has width m and the kernel has width k, the output will be of width $m - k + 1$.

Variants of the basic convolution function

- The rate of this shrinkage can be dramatic if the kernels used are large. Since the shrinkage is greater than 0, it limits the number of convolutional layers that can be included in the network. As layers are added, the spatial dimension of the network will eventually drop to 1×1.

- Another special case of the zero-padding setting is when just enough zero padding is added to keep the size of the output equal to the size of the input. MATLAB calls this same convolution. In this case, the network can contain as many convolutional layers as the available hardware can support, since the operation of convolution does not modify the architectural possibilities available to the next layer.

Variants of the basic convolution function

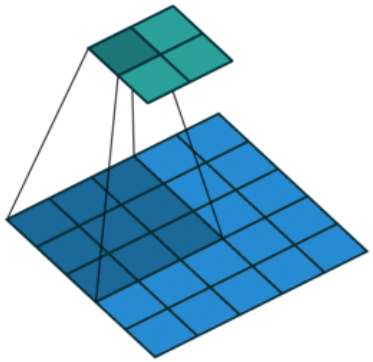
- The input pixels near the border, however, influence fewer output pixels than the input pixels near the center. This can make the border pixels somewhat underrepresented in the model.

- This motivates the other extreme case, which MATLAB refers to as full convolution, in which enough zeros are added for every pixel to be visited k times in each direction, resulting in an output image of width $m + k - 1$. The output pixels near the border are a function of fewer pixels than the output pixels near the center. This can make it difficult to learn a single kernel that performs well at all positions in the convolutional feature map.

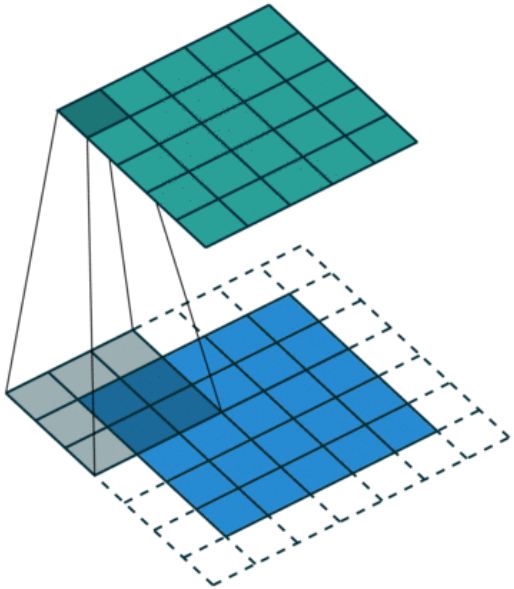
- Usually the optimal amount of zero padding lies between "valid" and "same" convolution.

Variants of the basic convolution function

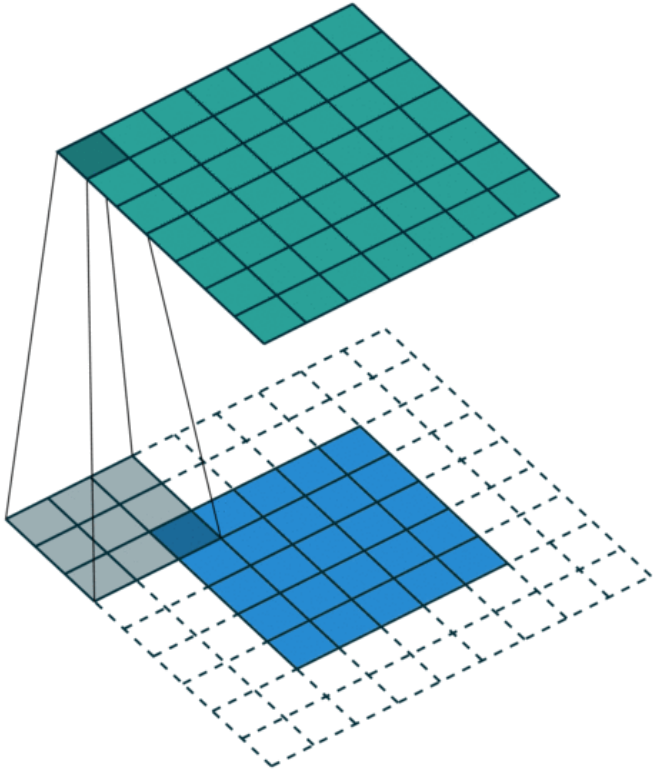
valid convolution



same convolution



full convolution



http://deeplearning.net/software/theano_versions/dev/tutorial/conv_arithmetic.html

Variants of the basic convolution function

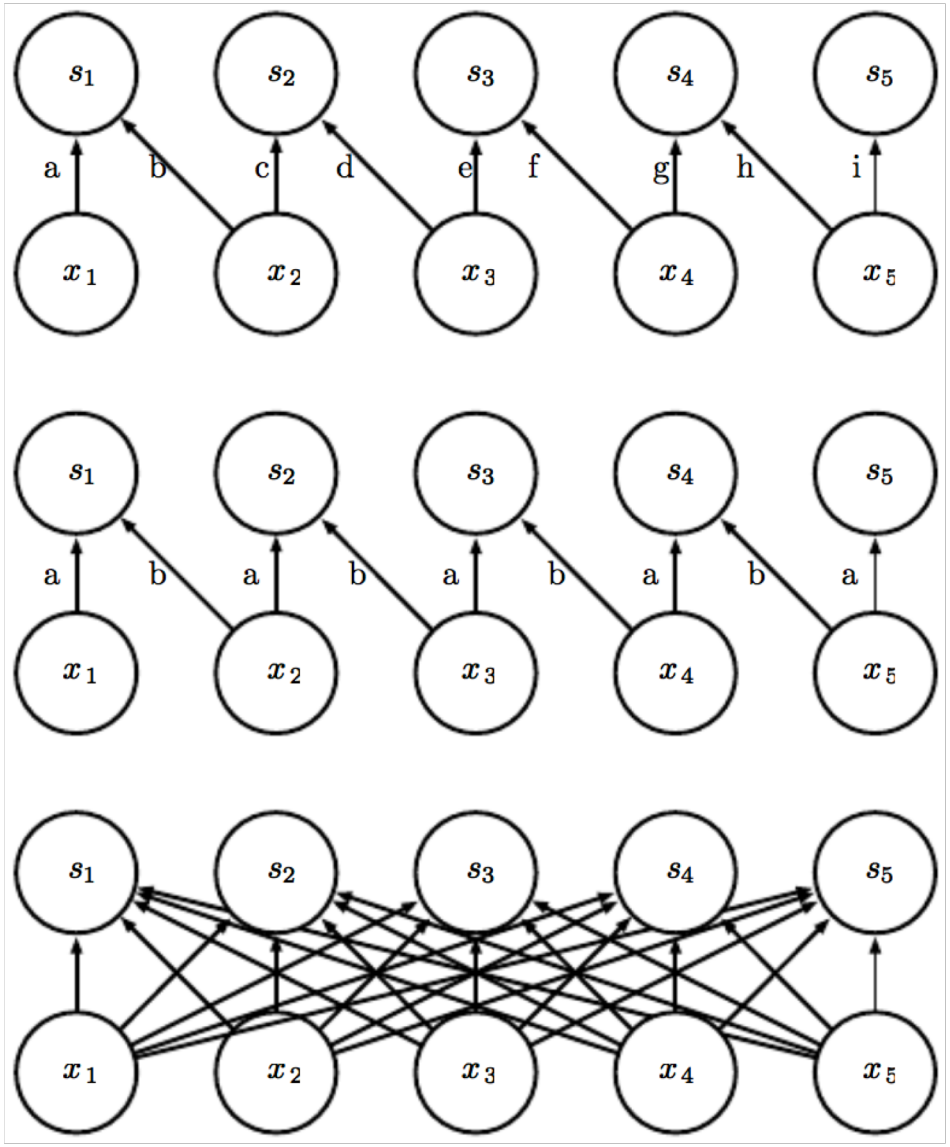
- In some cases, we do not actually want to use convolution, but want to use locally connected layers instead. The adjacency matrix in the graph of our MLP is the same, but every connection has its own weight, specified by a 6-D tensor $\mathbf{W}$. The indices into $\mathbf{W}$ are respectively: i; the output channel; j, the output row; k, the output column; l, the input channel; m, the row offset within the input; and n, the column offset within the input. The linear part of a locally connected layer is given by

$$Z_{i,j,k} = \sum_{l,m,n} \left[V_{l,j+m-1,k+n-1} w_{i,j,k,l,m,n} \right]. \tag{9.9}$$

- This is sometimes also called unshared convolution, because it is a similar operation to discrete convolution with a small kernel, but without sharing parameters.

Variants of the basic convolution function

Comparison of local connections, convolution, and full connections



Variants of the basic convolution function

- Locally connected layers are useful when we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space.

- For example, if we want to tell if an image is a picture of a face, we only need to look for the mouth in the bottom half of the image.

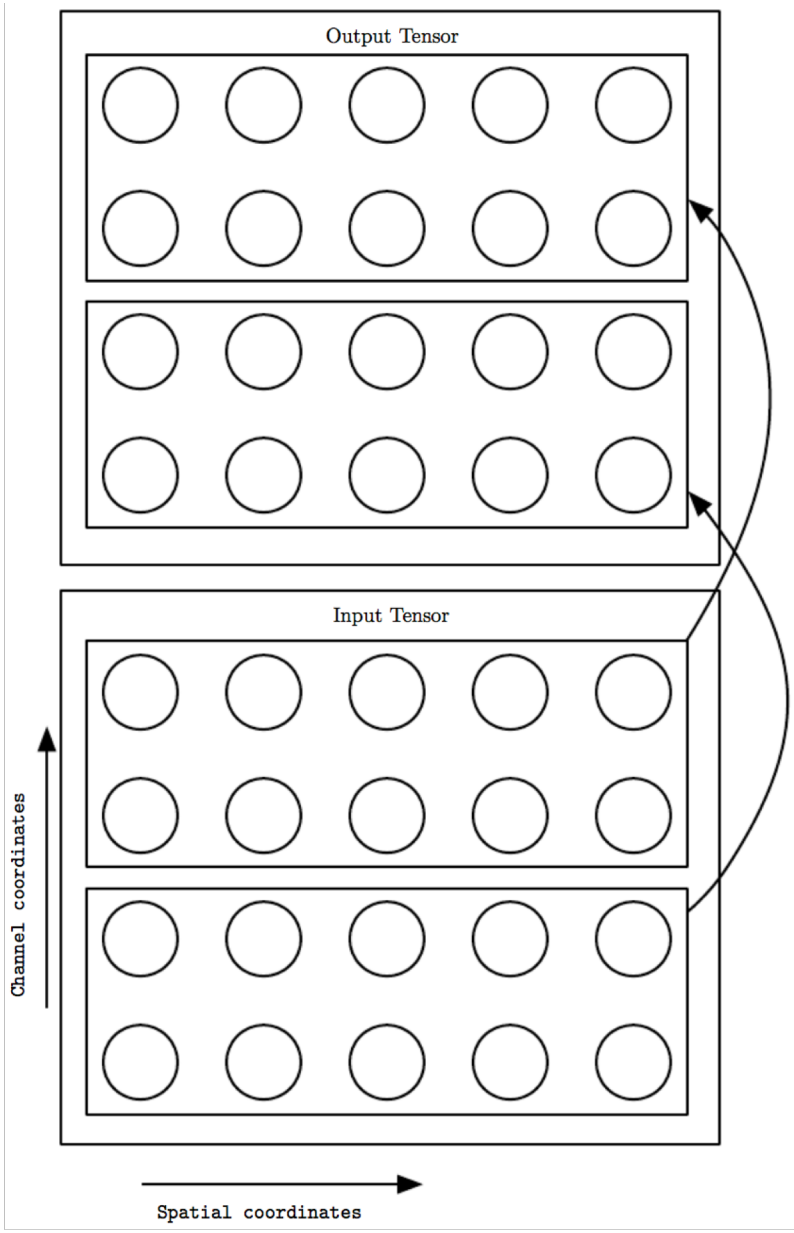
- It can also be useful to make versions of convolution or locally connected layers in which the connectivity is further restricted, for example to constrain each output channel i to be a function of only a subset of the input channels l.

Variants of the basic convolution function

- A common way to do this is to make the first m output channels connect to only the first n input channels, the second m output channels connect to only the second n input channels, and so on.

- Modeling interactions between few channels allows the network to have fewer parameters, reducing memory consumption, increasing statistical efficiency, and reducing the amount of computation needed to perform forward and back-propagation. It accomplishes these goals without reducing the number of hidden units.

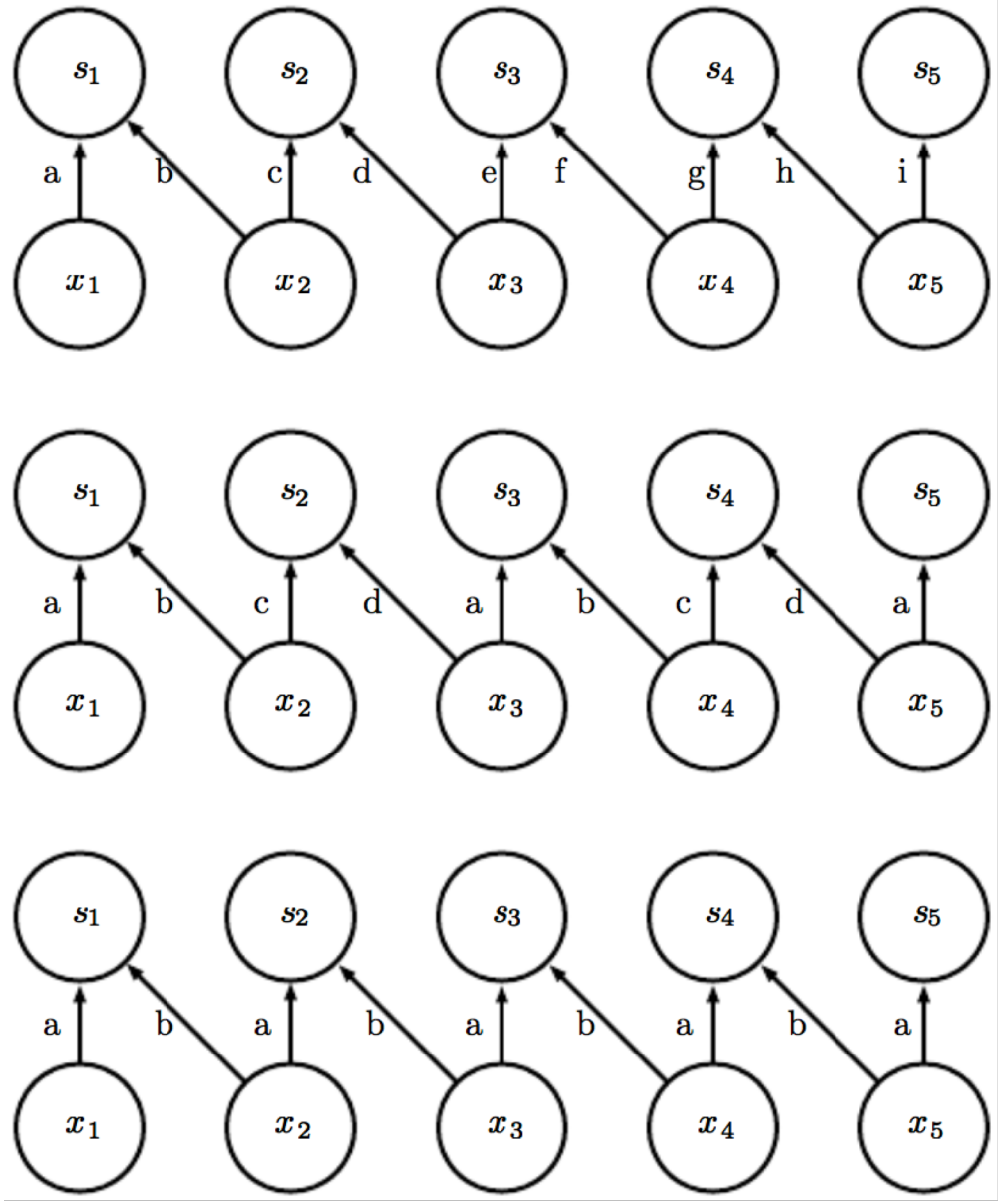
Variants of the basic convolution function



Variants of the basic convolution function

- Tiled convolution offers a compromise between a convolutional layer and a locally connected layer.
- Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space.
- This means that immediately neighboring locations will have different filters, as in a locally connected layer, but the memory requirements for storing the parameters will increase only by a factor of the size of this set of kernels, rather than by the size of the entire output feature map.

Variants of the basic convolution function



Locally connected layers

Tiled convolution

Standard convolution

Variants of the basic convolution function

- To define tiled convolution algebraically, let k be a 6-D tensor, where two of the dimensions correspond to different locations in the output map.

- Rather than having a separate index for each location in the output map, output locations cycle through a set of t different choices of kernel stack in each direction. If t is equal to the output width, this is the same as a locally connected layer.

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n,j\%t+1,k\%t+1}, \quad (9.10)$$

where percent % is the modulo operation, with $t\%t = 0$, $(t+1)\%t = 1$, and so on.

Variants of the basic convolution function

- Locally connected layers and tiled convolutional layers have both an interesting interaction with max pooling: the detector units of these layers are driven by different filters.

- If these filters learn to detect different transformed versions of the same underlying features, then the max-pooled units become invariant to the learned transformation. Convolutional layers are hard coded to be invariant specifically to translation.

Variants of the basic convolution function

- Other operations besides convolution are usually necessary to implement a convolutional network.

- To perform learning, one must be able to compute the gradient with respect to the kernel, given the gradient with respect to the outputs. In some simple cases, this operation can be performed using the convolution operation, but many cases of interest, including the case of stride greater than 1, do not have this property.

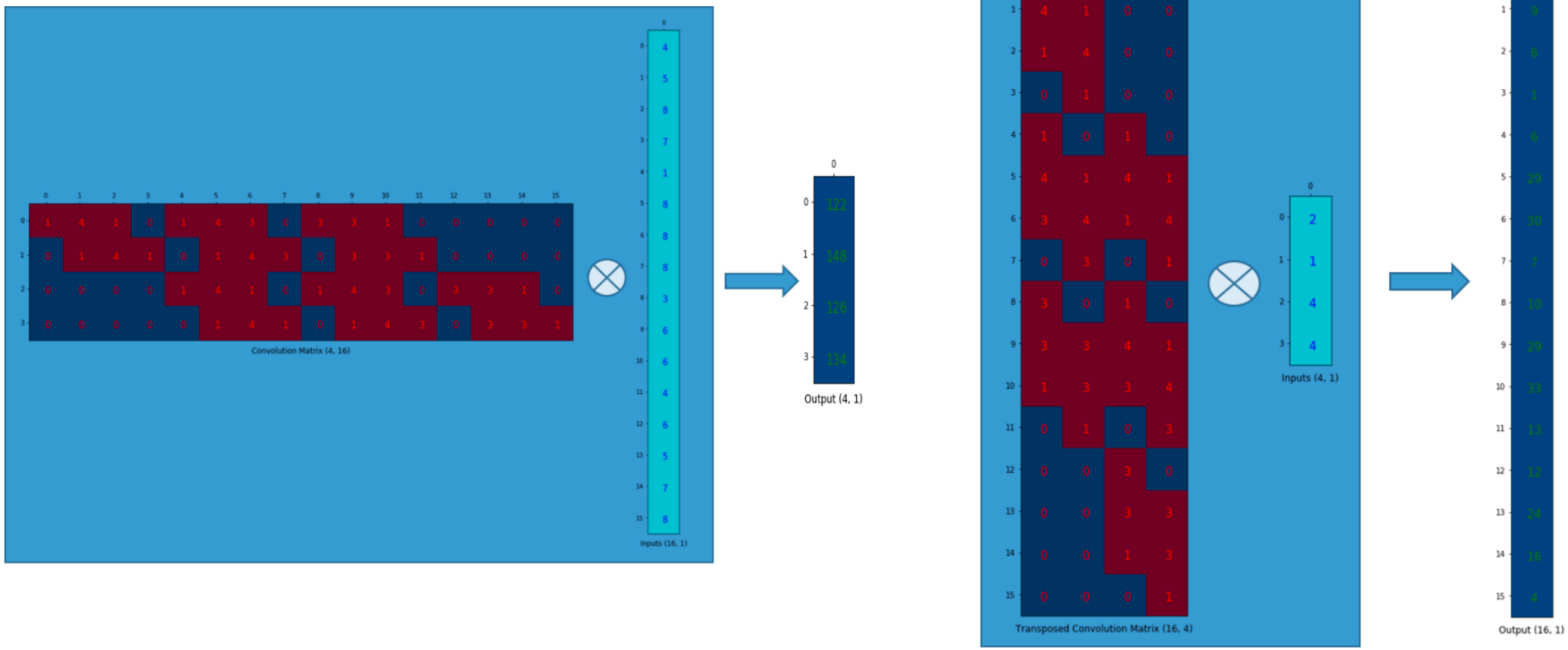
- Recall that convolution can be described as a matrix multiplication. The matrix involved is a function of the convolution kernel. The matrix is sparse, and each element of the kernel is copied to several elements of the matrix.

Variants of the basic convolution function

- Multiplication by the transpose of the matrix defined by convolution is the operation needed to back-propagate error derivatives through a convolutional layer.

- Care must be taken to coordinate this transpose operation with the forward propagation. The size of the output that the transpose operation should return depends on the zero-padding policy and stride of the forward propagation operation, as well as the size of the forward propagation's output map. In some cases, multiple sizes of input to forward propagation can result in the same size of output map, so the transpose operation must be explicitly told what the size of the original input was.

Variants of the basic convolution function



https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0

Variants of the basic convolution function

- To give a sense of how these equations work, we present the two-dimensional, single example version here.

- Suppose we want to train a convolutional network that incorporates strided convolution of kernel stack $\mathbf{K}$ applied to multichannel image $\mathbf{V}$ with stride s as defined by $c(\mathbf{K}, \mathbf{V}, s)$, as in equation 9.8.

- Suppose we want to minimize some loss function $J(\mathbf{V}, \mathbf{K})$. During forward propagation, we will need to use c itself to output Z, which is then propagated through the rest of the network and used to compute the cost function J . During back-propagation, we will receive a tensor $\mathbf{G}$ such that $G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(\mathbf{V}, \mathbf{K})$.

Variants of the basic convolution function

- To train the network, we need to compute the derivatives with respect to the weights in the kernel. To do so, we can use a function

$$g(\mathbf{G}, \mathbf{V}, s)_{i,j,k,l} = \frac{\partial}{\partial K_{i,j,k,l}} J(\mathbf{V}, \mathbf{K}) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)\times s+k,(n-1)\times s+l}. \quad (9.11)$$

- If this layer is not the bottom layer of the network, we will need to compute the gradient with respect to $\mathbf{V}$ to back-propagate the error further down. To do so, we can use a function

$$h(\mathbf{K}, \mathbf{G}, s)_{i,j,k} = \frac{\partial}{\partial V_{i,j,k}} J(\mathbf{V}, \mathbf{K}) \quad (9.12)$$

$$= \sum_{\substack{l,m \\ \text{s.t.} \\ (l-1)\times s+m=j}} \sum_{\substack{n,p \\ \text{s.t.} \\ (n-1)\times s+p=k}} \sum_{q} K_{q,i,m,p} G_{q,l,n}. \quad (9.13)$$

Variants of the basic convolution function

- Generally, we do not use only a linear operation to transform from the inputs to the outputs in a convolutional layer. We generally also add some bias term to each output before applying the nonlinearity.

- This raises the question of how to share parameters among the biases.

- For locally connected layers, it is natural to give each unit its own bias, and for tiled convolution, it is natural to share the biases with the same tiling pattern as the kernels.

Variants of the basic convolution function

- For convolutional layers, it is typical to have one bias per channel of the output and share it across all locations within each convolution map.

- However, if the input is of known, fixed size, it is also possible to learn a separate bias at each location of the output map.

- Separating the biases may slightly reduce the statistical efficiency of the model, but it allows the model to correct for differences in the image statistics at different locations. For example, when using implicit zero padding, detector units at the edge of the image receive less total input and may need larger biases.

Data types

- The data used with a convolutional network usually consist of several channels, each channel being the observation of a different quantity at some point in space or time.

- So far we have discussed only the case where every example in the train and test data has the same spatial dimensions. One advantage to convolutional networks is that they can also process inputs with varying spatial extents. These kinds of input simply cannot be represented by traditional, matrix multiplication-based neural networks. This provides a compelling reason to use convolutional networks even when computational cost and overfitting are not significant issues.

Data types

	Single channel	Multichannel
1-D	Audio waveform: The axis we convolve over corresponds to time. We discretize time and measure the amplitude of the waveform once per time step.	Skeleton animation data: Animations of 3-D computer-rendered characters are generated by altering the pose of a "skeleton" over time. At each point in time, the pose of the character is described by a specification of the angles of each of the joints in the character's skeleton. Each channel in the data we feed to the convolutional model represents the angle about one axis of one joint.
2-D	Audio data that has been preprocessed with a Fourier transform: We can transform the audio waveform into a 2-D tensor with different rows corresponding to different frequencies and different columns corresponding to different points in time. Using convolution in the time makes the model equivariant to shifts in time. Using convolution across the frequency axis makes the model equivariant to frequency, so that the same melody played in a different octave produces the same representation but at a different height in the network's output.	Color image data: One channel contains the red pixels, one the green pixels, and one the blue pixels. The convolution kernel moves over both the horizontal and the vertical axes of the image, conferring translation equivariance in both directions.
3-D	Volumetric data: A common source of this kind of data is medical imaging technology, such as CT scans.	Color video data: One axis corresponds to time, one to the height of the video frame, and one to the width of the video frame.

Data types

- For example, consider a collection of images in which each image has a different width and height. It is unclear how to model such inputs with a weight matrix of fixed size.

- Convolution is straightforward to apply; the kernel is simply applied a different number of times depending on the size of the input, and the output of the convolution operation scales accordingly.

- Convolution may be viewed as matrix multiplication; the same convolution kernel induces a different size of doubly block circulant matrix for each size of input.

Data types

- Note that the use of convolution for processing variably sized inputs makes sense only for inputs that have variable size because they contain varying amounts of observation of the same kind of thing—different lengths of recordings over time, different widths of observations over space, and so forth. Convolution does not make sense if the input has variable size because it can optionally include different kinds of observations.

- For example, if we are processing college applications, and our features consist of both grades and standardized test scores, but not every applicant took the standardized test, then it does not make sense to convolve the same weights over features corresponding to the grades as well as the features corresponding to the test scores.

Efficient convolution algorithms

- When a d-dimensional kernel can be expressed as the outer product of d vectors, one vector per dimension, the kernel is called separable.

- When the kernel is separable, naive convolution is inefficient. It is equivalent to compose d one-dimensional convolutions with each of these vectors. The composed approach is significantly faster than performing one d-dimensional convolution with their outer product.

- The kernel also takes fewer parameters to represent as vectors. If the kernel is ω elements wide in each dimension, then naive multidimensional convolution requires $O(\omega^d)$ runtime and parameter storage space, while separable convolution requires $O(\omega \times d)$ runtime and parameter storage space.

Convolutional networks

- Convolutional nets were some of the first working deep networks trained with back-propagation. It is not entirely clear why convolutional networks succeeded when general back-propagation networks were considered to have failed.

- It may simply be that convolutional networks were more computationally efficient than fully connected networks, so it was easier to run multiple experiments with them and tune their implementation and hyperparameters.

Convolutional networks

- Convolutional networks provide a way to specialize neural networks to work with data that has a clear grid-structured topology and to scale such models to very large size. This approach has been the most successful on a two-dimensional image topology.

- To process one-dimensional sequential data, we turn next to another powerful specialization of the neural networks framework: recurrent neural networks.