EBERHARD KARLS UNIVERSITÄT TÜBINGEN

MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT

Deep Neural Networks

Chapter 9: Convolutional Networks

9. Convolutional Networks

- 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 viewed as a 1D grid of samples at regular time intervals, and image data, which can be viewed as a 2D grid of pixels.
- The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation.
- *Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.*

9.1 The Convolution Operation

- In its most general form, convolution is an operation on two functions of a real-valued argument.
- Suppose we are tracking the location of a spaceship with a laser sensor. Our sensor provides $x(t)$, the position of the spaceship at time t. Both x and t are real-valued.
- Now suppose that our laser sensor is somewhat noisy. To obtain a less noisy estimate of the spaceship's position, we would like to average together several measurements. More recent measurements are more relevant, so we use a weighted average that gives more weight to recent measurements. We can do this with a weighting function $w(a)$, 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 providing a smoothed estimate of the position s of the spaceship:

$$s(t) = \int x(a)w(t-a)da$$

- This operation is called a convolution. The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t)$$

- In general, convolution is defined for any functions for which the above integral is defined, and may be used for other purposes besides taking weighted averages.

The Convolution Operation

- In convolutional network terminology, the first argument (here the function x) to the convolution is often referred to as the input and the second argument (in this example, the function w) as the kernel. The output is sometimes referred to as the feature map.
- Usually, when we work with data on a computer, time will be discretized, and our sensor will provide data at regular intervals. The time index t can then take on only integer values.
- If we now assume that x and w are defined only on integers t, we can define the discrete convolution:

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

The convolution operation

- In machine learning applications, the input is usually an n-dim. array of data and the kernel is usually an n-dim. array of parameters that are adapted by the learning algorithm. We refer to these n-dim. arrays as tensors.
- Because each element of the input and kernel must be explicitly stored separately, we assume that these functions are zero everywhere but the finite set of points for which we store the values.
- This means that in practice we can implement the infinite sum as a sum over a finite number of array elements.
- Finally, we often use convolutions over more than one axis at a time.

2-dimensional Convolutions

- For example, if we use a 2-dim. image I as our input, we probably also want to use a two-dimensional kernel K:

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

- Convolution is commutative, so we can equivalently write:

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

- Usually the latter 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 commutative property of convolution arises because we have flipped the kernel relative to the input, in the sense that as m increases, the index into the input increases, but the index into the kernel decreases.
- The only reason to flip the kernel is to obtain the commutative property. 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:

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

Example of 2D Convolution

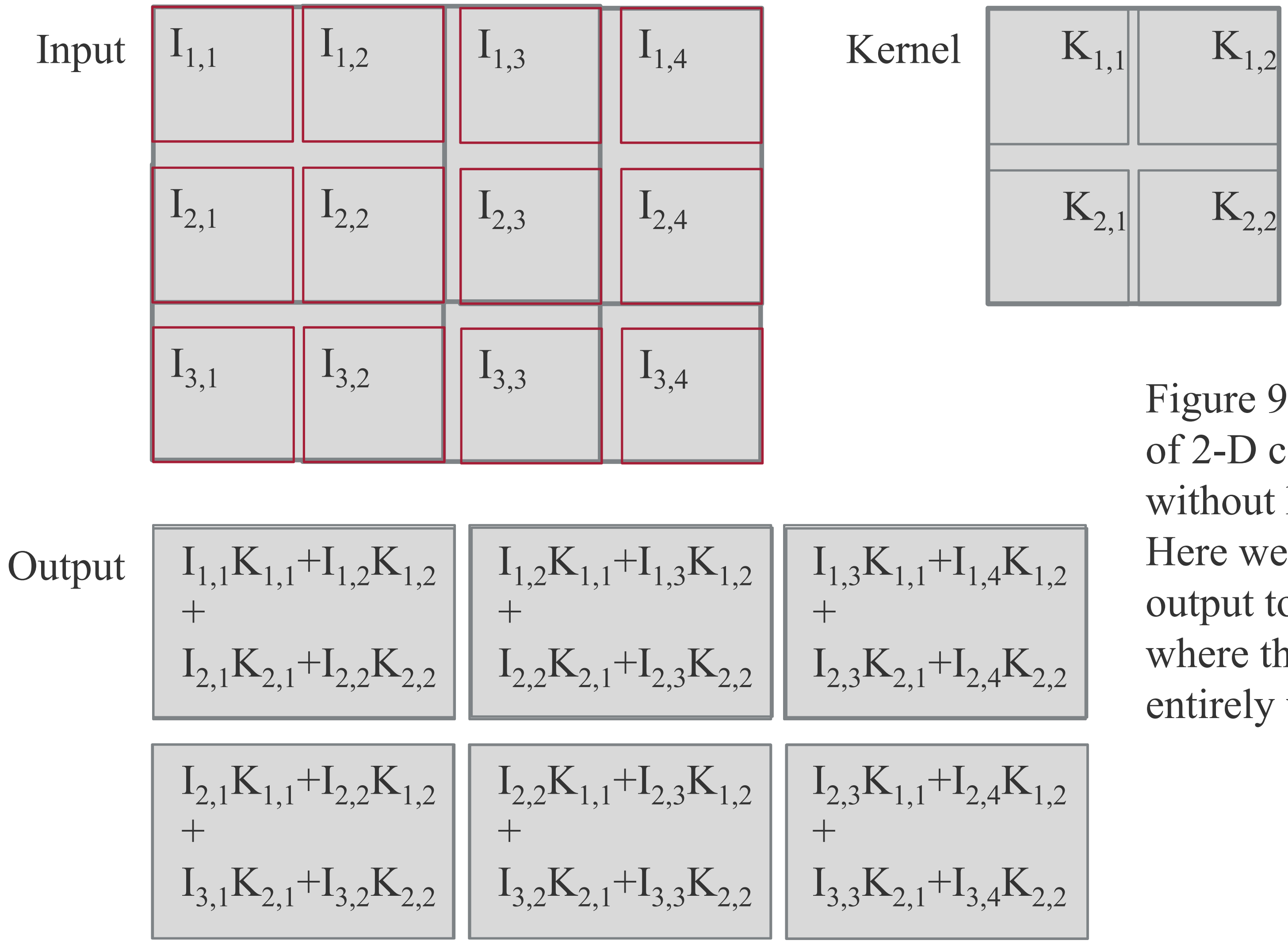

Input
$I_{1,1}$ $I_{1,2}$ $I_{1,3}$ $I_{1,4}$
$I_{2,1}$ $I_{2,2}$ $I_{2,3}$ $I_{2,4}$
$I_{3,1}$ $I_{3,2}$ $I_{3,3}$ $I_{3,4}$
Kernel
$K_{1,1}$ $K_{1,2}$
$K_{2,1}$ $K_{2,2}$
Output
$I_{1,1}K_{1,1}+I_{1,2}K_{1,2} + I_{2,1}K_{2,1}+I_{2,2}K_{2,2}$
$I_{1,2}K_{1,1}+I_{1,3}K_{1,2} + I_{2,2}K_{2,1}+I_{2,3}K_{2,2}$
$I_{1,3}K_{1,1}+I_{1,4}K_{1,2} + I_{2,3}K_{2,1}+I_{2,4}K_{2,2}$
$I_{2,1}K_{1,1}+I_{2,2}K_{1,2} + I_{3,1}K_{2,1}+I_{3,2}K_{2,2}$
$I_{2,2}K_{1,1}+I_{2,3}K_{1,2} + I_{3,2}K_{2,1}+I_{3,3}K_{2,2}$
$I_{2,3}K_{1,1}+I_{2,4}K_{1,2} + I_{3,3}K_{2,1}+I_{3,4}K_{2,2}$

Figure 9.1: An example of 2-D convolution without kernel-flipping. Here we restrict the output to only positions where the kernel lies entirely within the image

Example of 2D Convolution

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

Input

0	0	0	0	0	0
0	1	5	2	3	0
0	5	5	5	5	0
0	2	5	1	2	0
0	3	5	2	2	0
0	0	0	0	0	0

Kernel 1

-1	2	-1
-1	2	-1
-1	2	-1

Kernel 2

-1	-1	-1
2	2	2
-1	-1	-1

Output 1

0	0	0	0	0	0
0	2	7	-4	9	0
0	1	14	-4	12	0
0	5	12	-8	10	0
0	0	12	-8	5	0
0	0	0	0	0	0

Output 2

0	0	0	0	0	0
0	2	1	5	0	0
0	7				0
0					0
0					0
0	0	0	0	0	0

9.2 Motivation of Convolution

- Convolution leverages three important ideas:
  - 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 referred to as sparse connectivity or sparse weights).

Motivation of Convolution

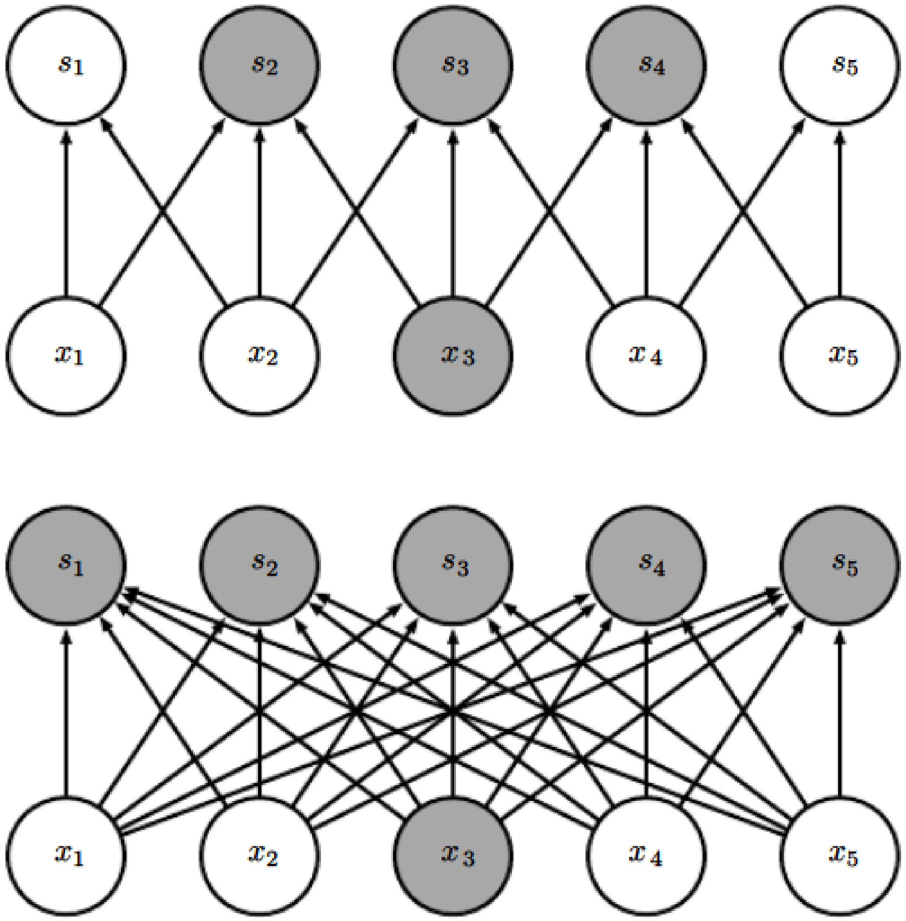
- This is accomplished by making the kernel smaller than the input.
- When processing an image, the input image might have millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels.
- This means that we need to store fewer 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.
- These improvements in efficiency are usually quite large.

Efficiency of Convolution

- If there are m inputs and n outputs, 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 many practical applications, it is possible to obtain good performance on the machine learning task while keeping k several orders of magnitude smaller than m.
- In a deep CNN, 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 from building blocks that each describe only sparse interactions.

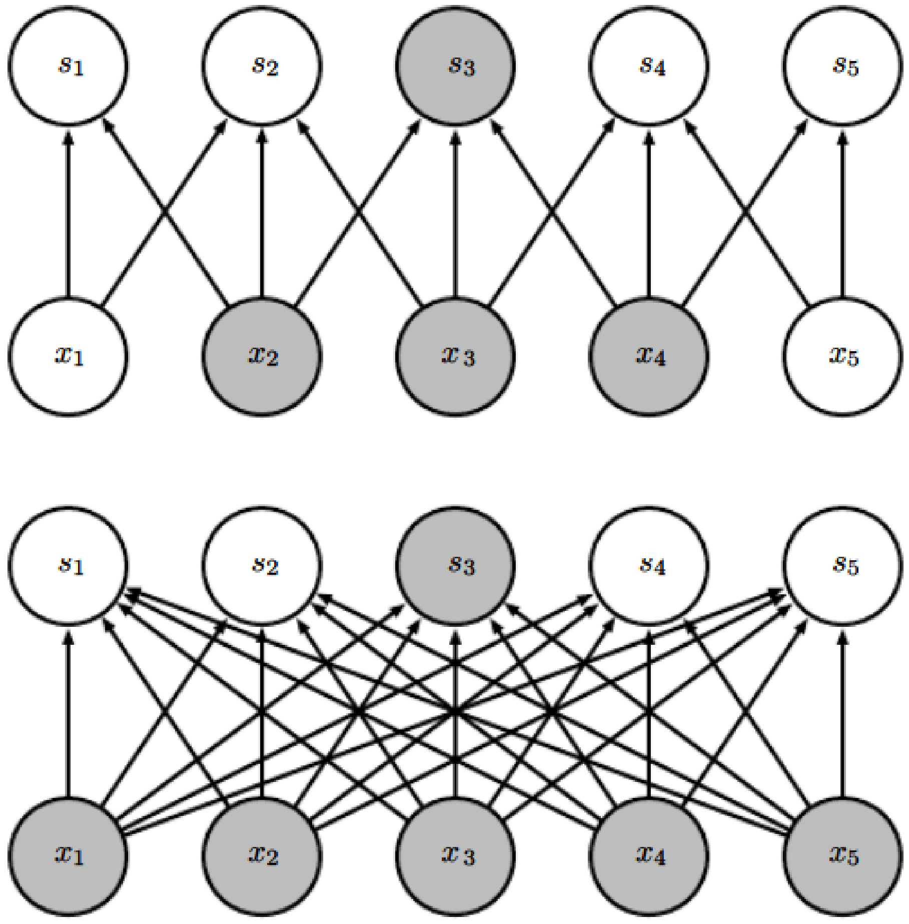
Sparse Connectivity of Convolution

Figure 9.2: *Sparse connectivity, viewed from below:* We highlight one input unit, x_3, and also highlight the output units in s that are affected by this unit. *(Top)*When s is formed by convolution with a kernel of width 3, only three outputs are affected by x_3. *(Bottom)*When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by x_3.



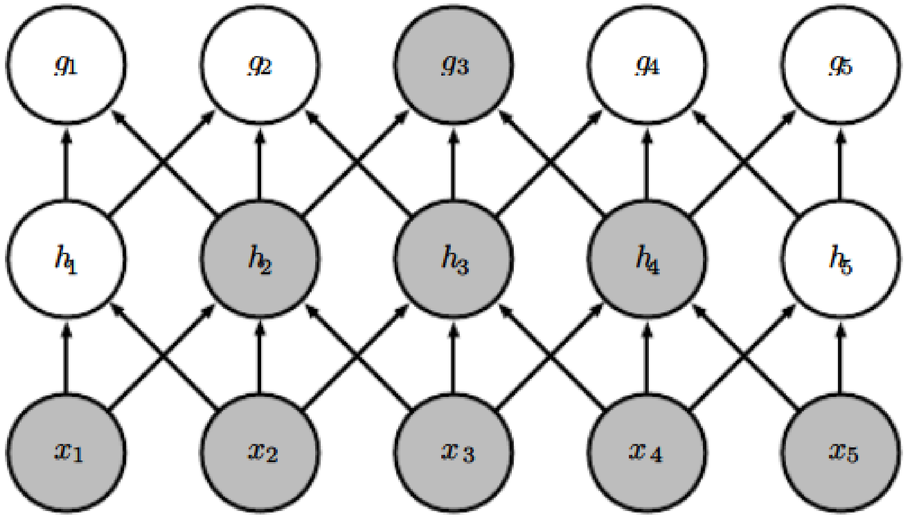
Sparse Connectivity of Convolution

Figure 9.3: *Sparse connectivity, viewed from above:* We highlight one output unit, s_3, and also highlight the input units in x that affect this unit. These units are the receptive field of s_3.
*(Top)*When s is formed by convolution with a kernel of width 3, only three inputs affect s_3.
*(Bottom)*When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the inputs affect s_3.



Indirect Connectivity of Convolution

Figure 9.4: The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution or pooling (section 9.3). This means that even though *direct* connections in a convolutional net are very sparse, units in the deeper layers can be *indirectly* connected to all or most of the input image.



Parameter Sharing of Convolution

- 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.
- As a synonym for parameter sharing, one can say that a network has shared ~~tied~~ weights, because the value of the weight applied to one input is tied to the value of a weight applied elsewhere.
- 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).

Parameter Sharing of Convolution

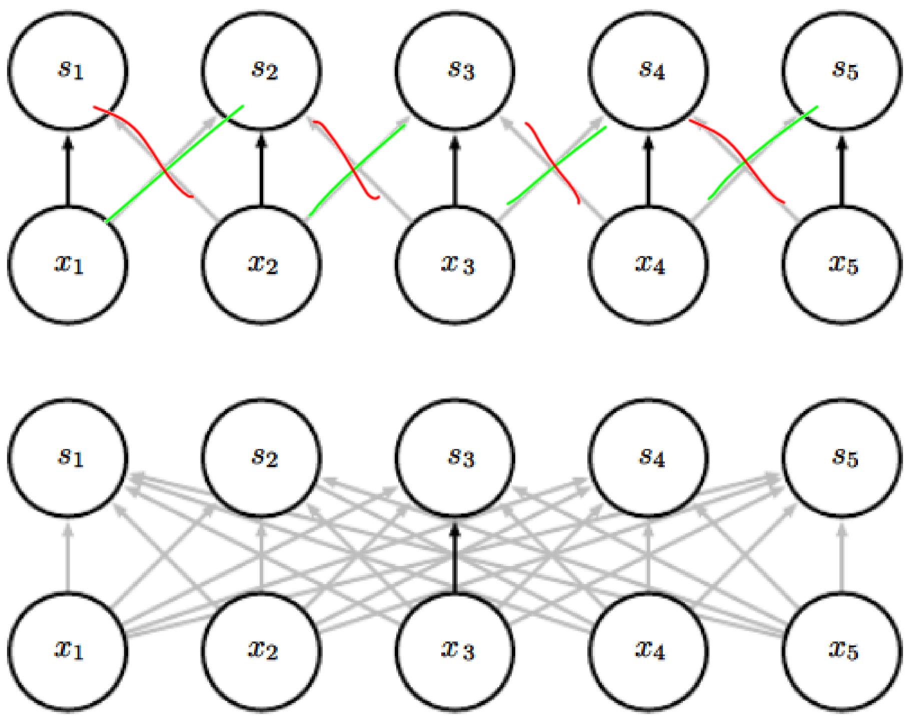
- The parameter sharing used by the convolution operation means that rather than learning a separate set of parameters for every location, we learn only one set.
- This does not affect the runtime of forward propagation - it is still $O(k×n)$ - but it does further reduce the storage requirements of the model to k parameters.
- Recall that k is usually several orders of magnitude less than m. Since m and n are usually roughly the same size, k is practically insignificant compared to $m×n$.
- Convolution is thus dramatically more efficient than dense matrix multiplication in terms of the memory requirements and statistical efficiency.

↳ how many parameters

Parameter Sharing of Convolution

Figure 9.5: Parameter sharing: Black arrows indicate connections that use a particular parameter in two different models.
*(Top)*The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Due to parameter sharing, this single parameter is used at all input locations.
*(Bottom)*The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing so the parameter is used only once.

colours same weights



/ invariance

Equivariance to Translation

- The particular form of parameter sharing of convolution causes the layer to have 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 (shifts) the input, then the convolution function is equivariant to g.
- 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.

Equivariance to Translation

- 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.
- 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.

Equivariance to Translation

- When processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear nearly everywhere in the image, so it is practical to share parameters across the entire image.
- Sharing parameters across the entire image is not always desired. If we are processing images that are centered on an individual’s face, we might want to extract different features at different locations (e.g. eyes and eyebrows at the top, chin at the bottom)
- Convolution is not naturally equivariant to some other transformations, such as changes in scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations.

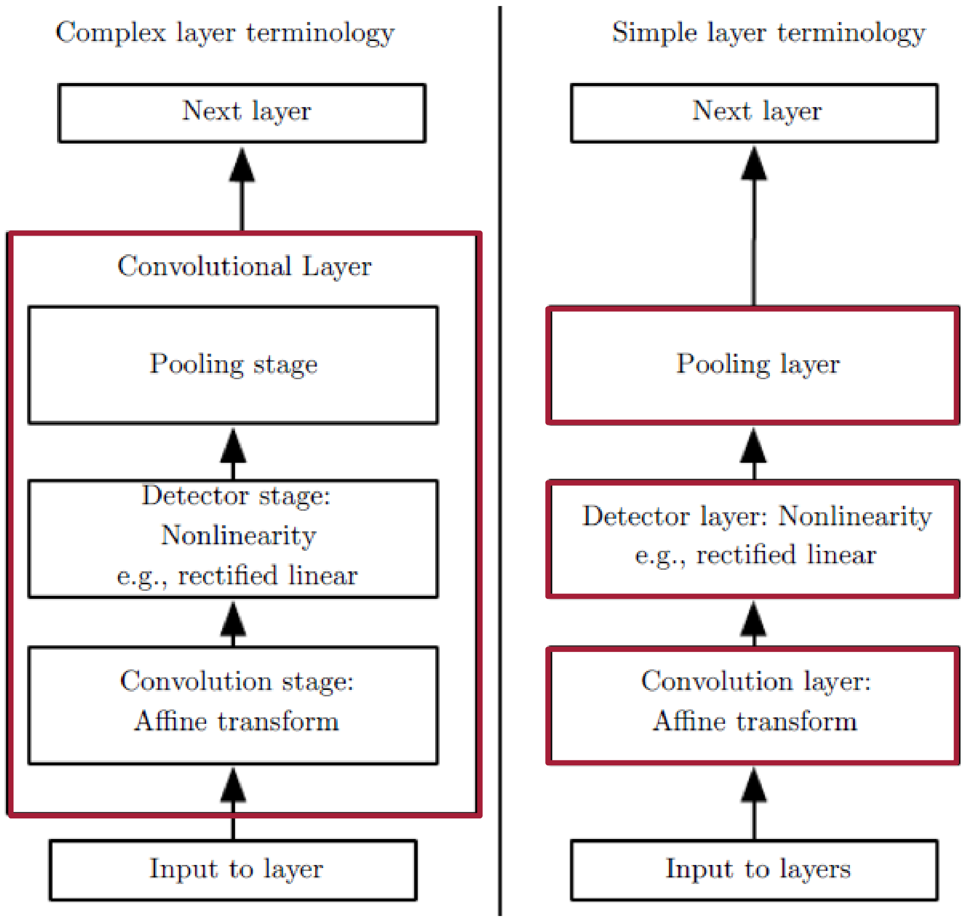
9.3 Pooling

A typical layer of a convolutional network consists of three stages:

1. First stage: the layer performs several convolutions in parallel to produce a set of linear activations.
2. Second stage: each linear activation is run through a nonlinear activation function, such as the ReLU.
   This stage is sometimes called the detector stage.
3. Third stage: we use a pooling function to modify the output of the layer further.

CNN Network Terminologies

Fig 9.7: The components of a typical CNN network layer. Two different terminologies: *(Left)* Here, the CNN is viewed as a small number of relatively complex layers, with each layer having several "stages." In this lecture we generally use this terminology. *(Right)*In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every "layer" has parameters.


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

- 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 L2 norm of a rectangular neighborhood, or
  - a weighted average based on the distance from the central pixel.
- In all cases, pooling helps to make the representation become invariant to small translations of the input.

Translation Invariance of Max Pooling

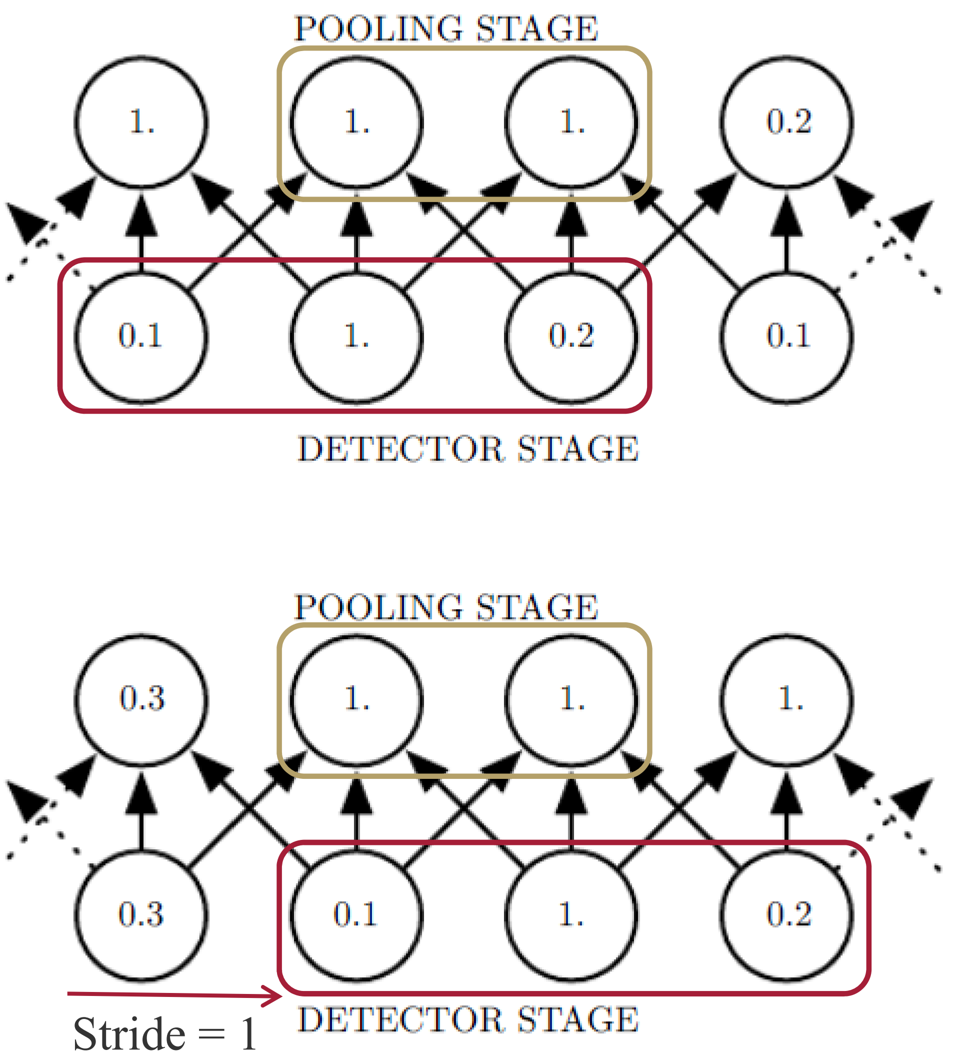


Fig 9.8: Translation invariance of Max pooling:
*(Top)*A view of the middle of the output of a convolutional layer. The bottom row shows outputs of the nonlinearity. The top row shows the outputs of max pooling, with a pooling region width of three pixels. *(Bottom)*A view of the same network, after 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 only sensitive to the maximum value in the neighborhood, not its exact location.

better seen if you downsample

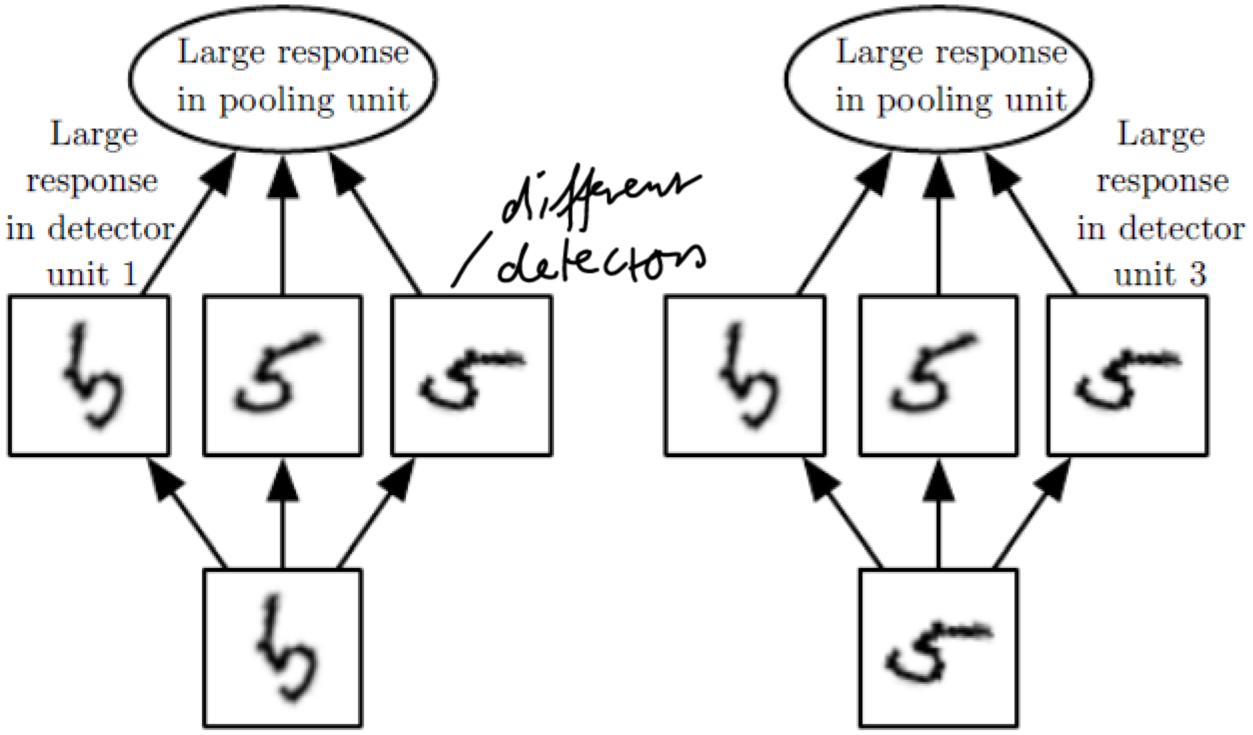
Max Pooling

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

Learned Invariances of Pooling

- By pooling over the outputs of separately parametrized convolutions, the features can learn invariance to other transformations, like rotation.

Fig 9.9: *Example of learned invariances:* A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant rotation.


Large response in pooling unit
Large response in pooling unit
Large response in detector unit 1
different / detectors
Large response in detector unit 3

Pooling with Downsampling

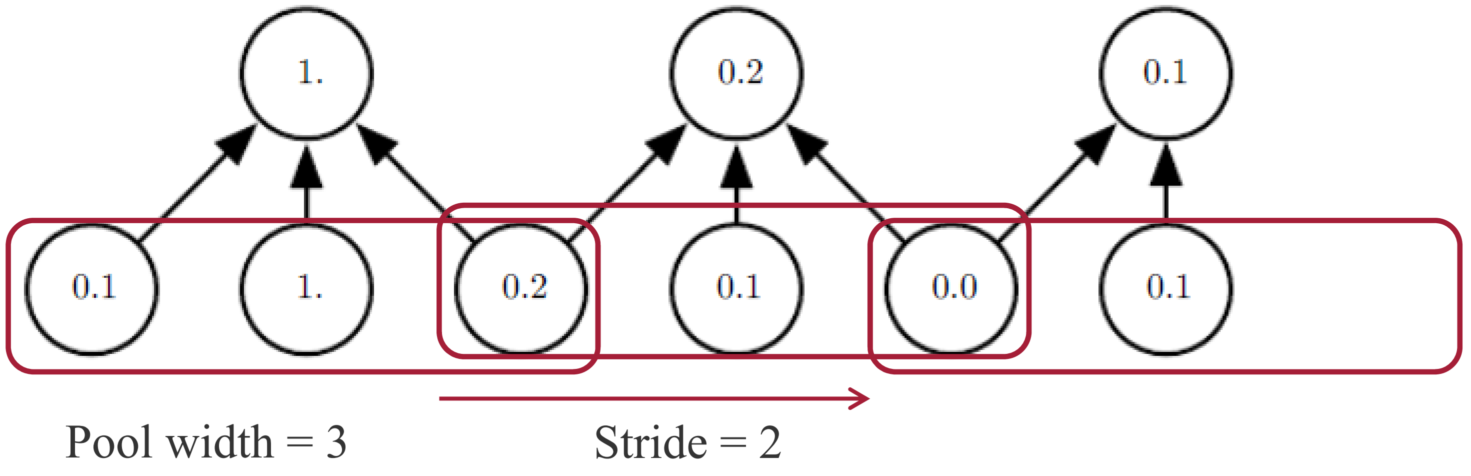


Figure 9.10: *Pooling with downsampling*. Here we use max-pooling with a pool width of 3 and a stride between pools of 2. This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer. Note that the rightmost pooling region has a smaller size, but must be included if we do not want to ignore some of the detector units.

9.4 Convolution and Pooling as an Infinitely Strong Prior

- Recall the concept of a prior probability distribution from sect. 5.2. This encodes our beliefs about what models are reasonable, before we have seen any data.
- Priors can be considered weak or strong depending on how concentrated the probability density in the prior is.
- A weak prior is a prior distribution with high entropy, such as a Gaussian distribution with high variance. This allows the data to move the parameters rather freely.
- A strong prior has very low entropy, such as a Gaussian distribution with low variance. Such a prior plays a more active role in determining where the parameters end up.
- An infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden.

vgl. Kernel

Convolution as Infinitely strong Prior

- We can imagine a convolutional net as 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 receptive field of that hidden unit.
- 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.

Pooling as Infinitely strong Prior

- Likewise, the use of pooling is an infinitely strong prior that each unit should be invariant to small translations.
- One key insight is that convolution and pooling can cause underfitting. Like any prior, convolution and pooling are only useful when the assumptions made by the prior are reasonably accurate. If a task relies on preserving precise spatial information, then using pooling on all features can increase the training error.
- Another key insight from this view is that we should only compare convolutional models to other convolutional models in benchmarks of statistical learning.
- Models that do not use convolution would be able to learn even if we permuted all of the pixels in the image.

9.5 Variants of the Basic Convolution

- When we refer to convolution in the context of neural networks, we usually mean an operation that consists of many applications of convolution in parallel.
- Convolution with a single kernel can only extract 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 but a grid of vector-valued observations.
- E.g., a color image has a R, G, B intensity at each pixel.
- In a multilayer conv. network, the input to the second layer is the output of the first layer, which usually has the output of many different convolutions at each position.

Further Remarks on Convolution

- 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 conv. networks usually use multi-channel convolution, the linear operations they are based on are not guaranteed to be commutative, even if kernel-flipping is used. They are only commutative if each operation has the same # of output channels as input channels.

Computing Convolution Indices

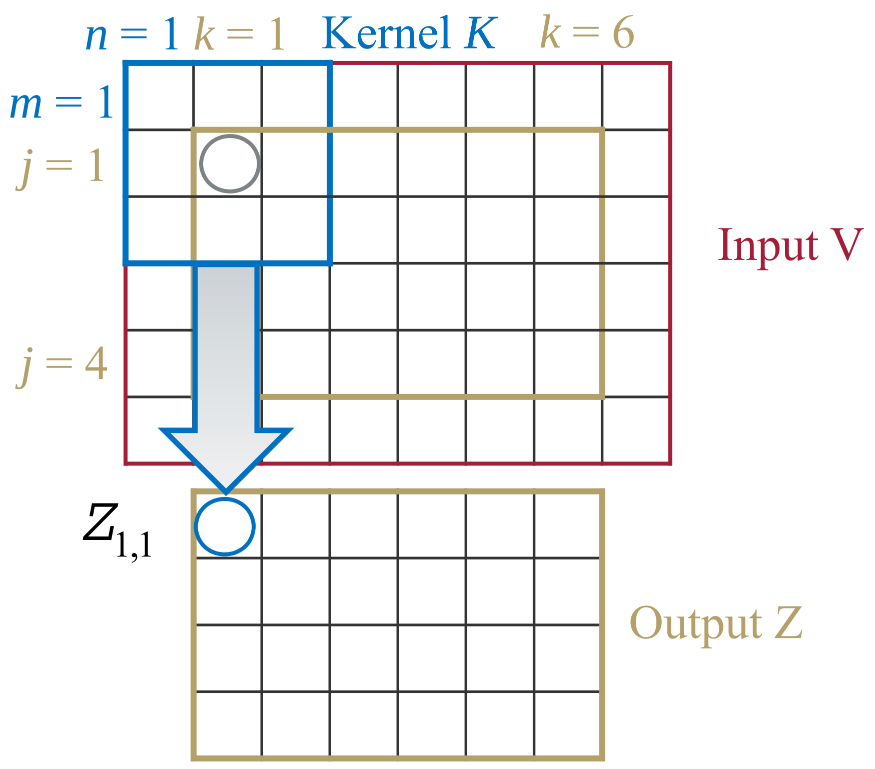
- Assume we have a 4-D kernel tensor **K** with element $K_{i,j,k,l}$ giving the connection strength between a unit in channel i of the output and a unit in channel j of the input, with an offset of k rows and l columns between the output unit and the input unit. Assume our input consists of observed data **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 **Z** with the same format as **V**. If **Z** is produced by convolving **K** across **V** without flipping **K**, then

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

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

Computing Convolution Indices

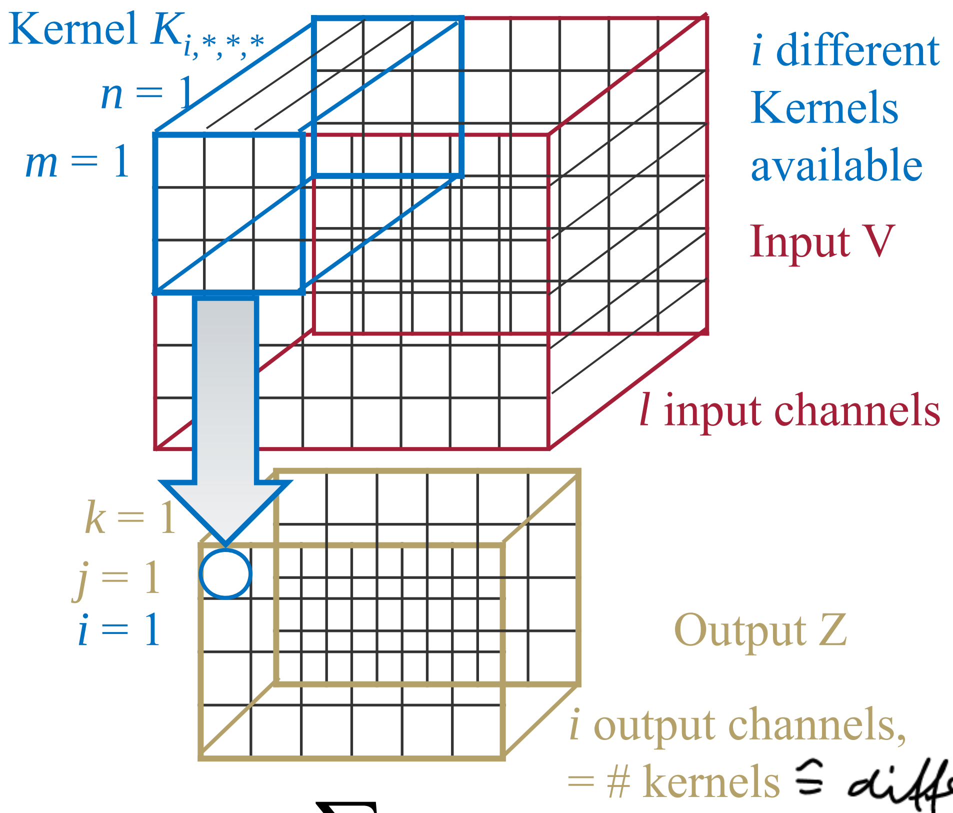
- 2-dim. case



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

$m = 1..3$ $n = 1..3$

- 3-dimensional case



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

$m = 1..3$ $n = 1..3$

Convolution with a Stride

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

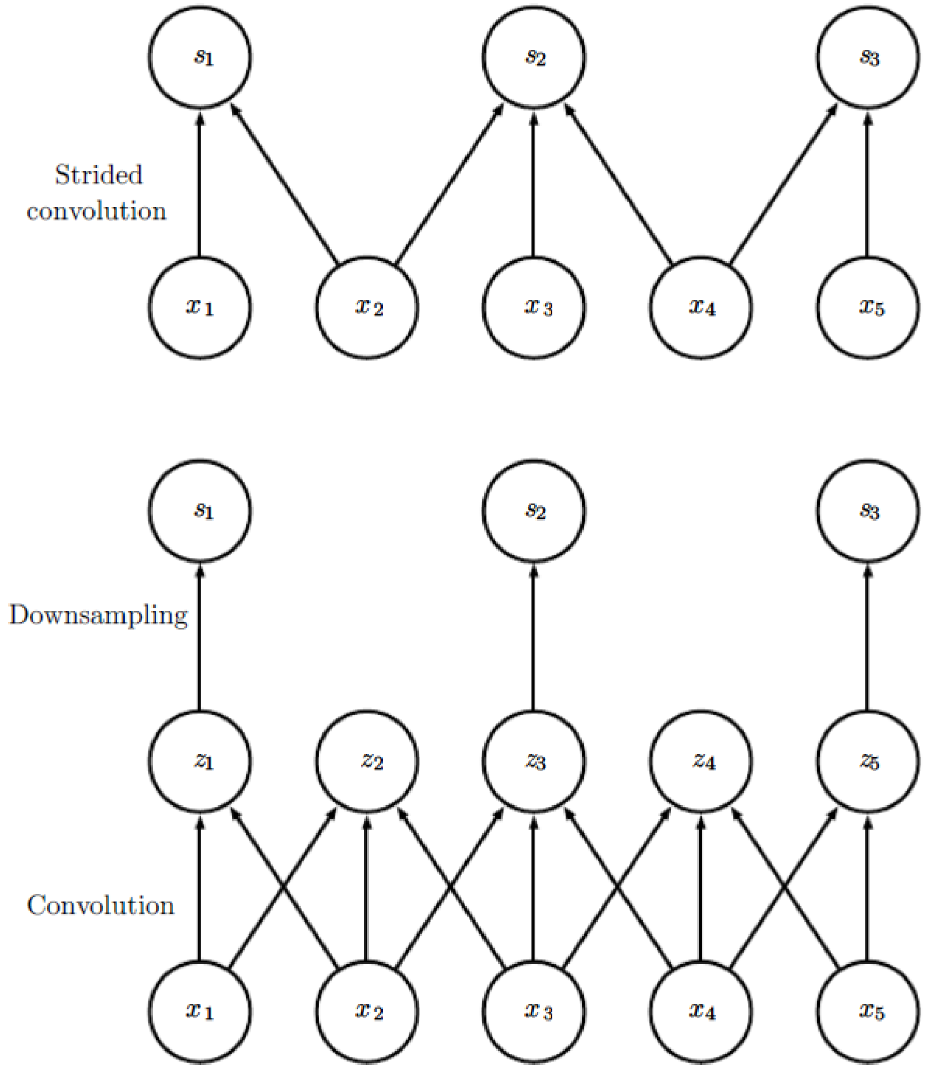
- We may want to skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting features as finely). We can think of this as downsampling the output of the full convol. function.
- If we want to sample only every s pixels in each direction in the output, then we can define a downsampled convolution function c such that

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

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. (not very practical)

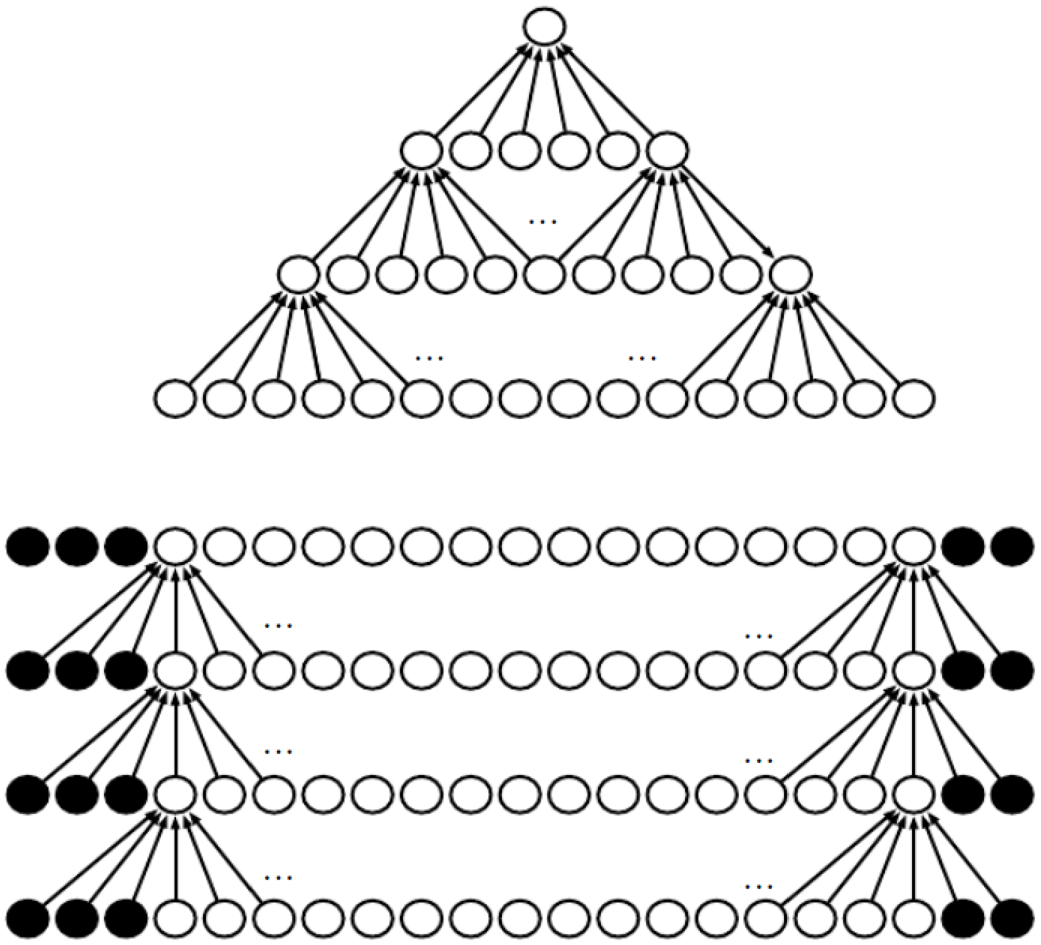
Convolution with a Stride

Figure 9.12: Convolution with a stride. In this example, we use a stride of two.
*(Top)*Convolution with a stride length of two implemented in a single operation.
(Bottom) Convolution with a stride greater than one pixel is equivalent to convolution with unit stride followed by downsampling. Obviously, the two-step approach involving downsampling is computationally wasteful, because it computes many values that are then discarded.



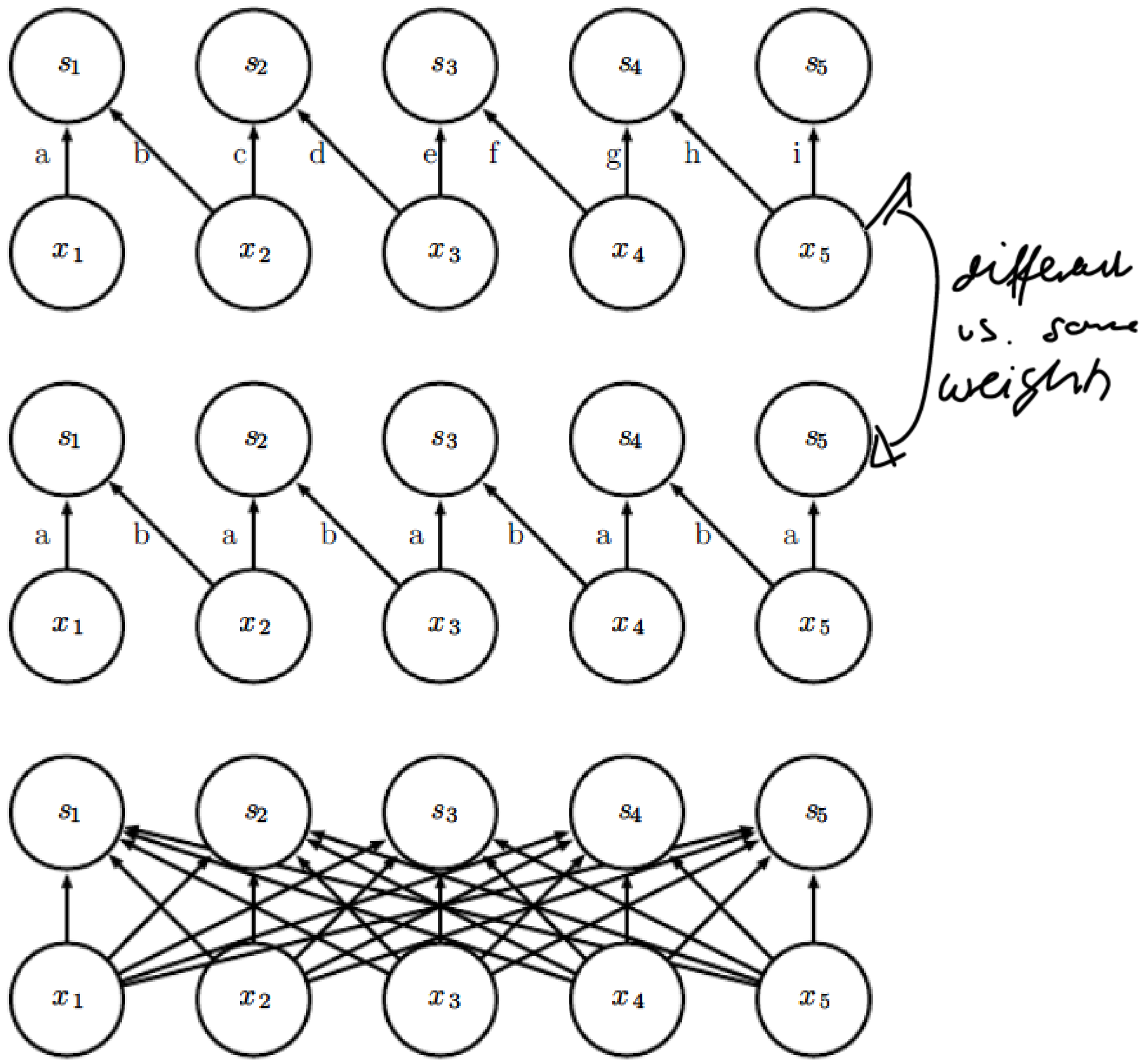
Effect of Zero Padding

Fig. 9.13: *The effect of zero padding on network size*: Consider a convolutional network with a kernel of width six at every layer and no pooling.
(Top) In this network, we do not use any zero padding. This causes the network to shrink by five pixels at each layer. With an input of sixteen pixels, we are only able to have three convolutional layers,
(Bottom) By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.



Local Connections and Convolution

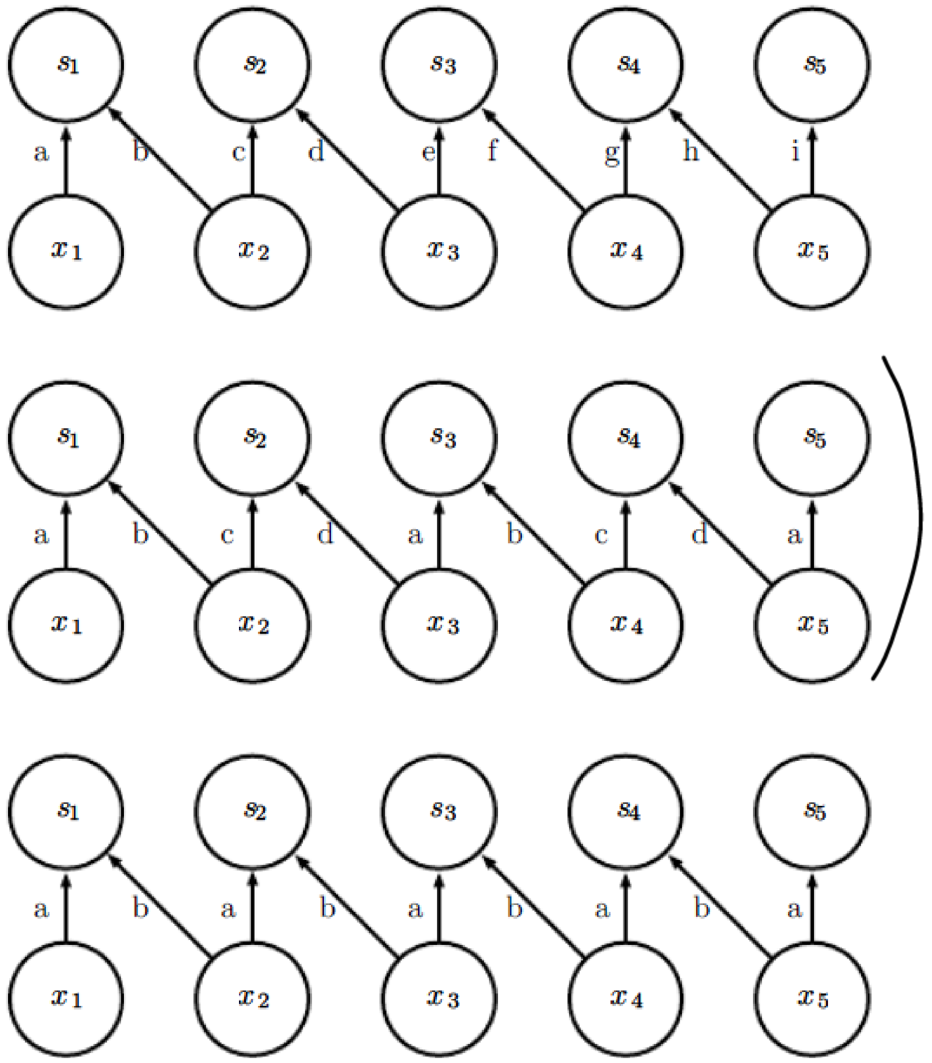
Fig. 9.14: Comparison of local connections, convolution, and full connections.
(Top) A locally connected layer with a patch size of two pixels. Each edge is associated with its own weight parameter.
(Center) A convolutional layer with a kernel width of two pixels. This model has exactly the same connectivity as the locally connected layer. The convol. layer uses the same two weights a, b repeatedly across the entire input.
*(Bottom)*A fully connected layer resembles a locally connected layer in that each edge has its own weight, but it does not have the restricted connectivity of the locally connected layer.



s_1 s_2 s_3 s_4 s_5
a b c d e f g h i
x_1 x_2 x_3 x_4 x_5
s_1 s_2 s_3 s_4 s_5
a b a b a b a b a
x_1 x_2 x_3 x_4 x_5
s_1 s_2 s_3 s_4 s_5
x_1 x_2 x_3 x_4 x_5
differen
vs. sam
weights

Tiled Convolution

Fig. 9.16: A comparison of locally connected layers, tiled convolution, and standard convolution. Here the kernel is two pixels wide.
(Top) A locally connected layer has no sharing at all. We indicate that each connection has its own weight by labeling each connection with a unique letter.
(Center) Tiled convolution has a set of t different kernels. We show the case of t = 2. One kernel has edges labeled a and b, the other has edges labeled c and d. Each time we move one pixel to the right in the output, we cyclically use a different kernel.
(Bottom) Standard convolution is equivalent to tiled convolution with t = 1. There is only one kernel and it is applied everywhere.



s_1 s_2 s_3 s_4 s_5
a b c d e f g h i
x_1 x_2 x_3 x_4 x_5
s_1 s_2 s_3 s_4 s_5
a b c d a b c d a
x_1 x_2 x_3 x_4 x_5
s_1 s_2 s_3 s_4 s_5
a b a b a b a b a
x_1 x_2 x_3 x_4 x_5

9.6 Structured Outputs

- We skip this

9.7 Data Types

- We skip this

9.8 Efficient Convolution Algorithms

- Convolution is equivalent to converting both the input and the kernel to the frequency domain using a Fourier transform, performing point-wise multiplication of the two signals, and converting back to the time domain using an inverse Fourier transform. For some problem sizes, this can be faster than the naive implementation of discrete convolution.
- 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 w elements wide in each dimension, then naive multidimensional convolution requires $O(w^d)$ runtime and parameter storage space, while separable convolution requires $O(wd)$ runtime and parameter storage space. Of course, not every convolution can be represented in this way.

9.9 Random or Unsupervised Features

- We skip this

9.10 The Neuroscientific Basis for Convolutional Networks

- We skip this

Convolutional Networks and the History of Deep Learning

- We skip this