PART 2.2: CONVOLUTION NEURAL NETWORK - THEORY

Figures and content retrieved from Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

CONVOLUTION NEURAL NETWORK (CNN)

- First proposed by LeCun in 1989
- *"Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers."* *[*Goodfellow et al. 2016*]*
- Devised for processing data with grid-like topology.
  - EX> Time series data (1D), image data (2D)

- The main difference between a CNN and regular NN is that it uses convolution operation instead of matrix multiplication as in NN.

- Operations in CNN: Convolution and Pooling

COMPONENT 1: CONVOLUTION

Working Example: Tracking location of spaceship with a laser sensor

- Sensor output: $x(t)$, position of spaceship at time t
- Assume noisy sensor:
  - We want to take the weighted avg. of measurements
  - Less aged, a, measurement should have higher weights, w(a).

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

One example of convolution

Rewritten with convolution operation , *.

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

  - In a discrete time:

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

TERMINOLOGY

Input

Feature map

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

Kernel

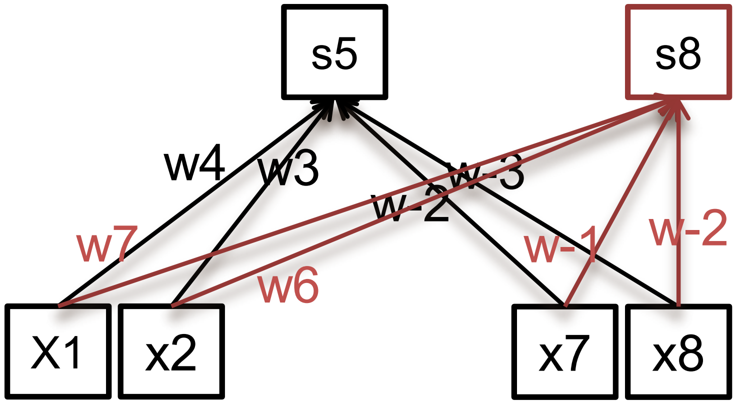
- Input: usually a tensor of data
- Kernel: usually a tensor of parameters that are adapted by the learning algorithm.

1D DATA &

- Point-wise convolution output for 1D data

$$s(t) = (x * w)(t) = \sum_a x(a)w(t-a)$$

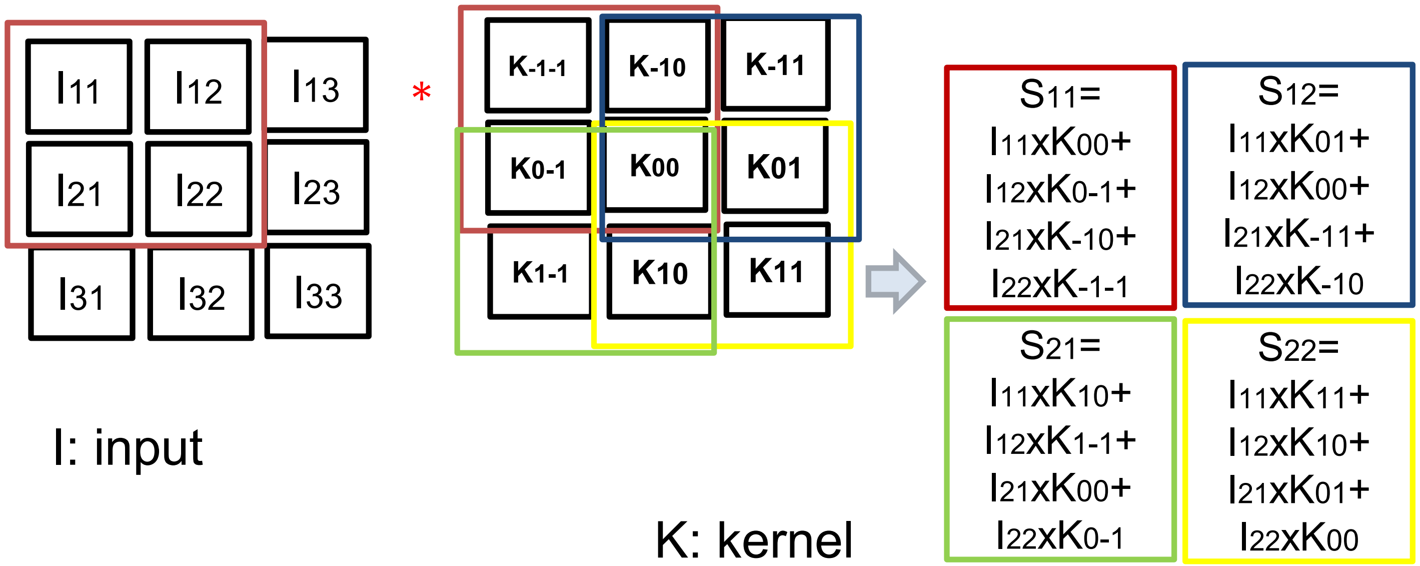
Weight based on distance to t from a



2D DATA

- Point-wise convolution output for 2D data

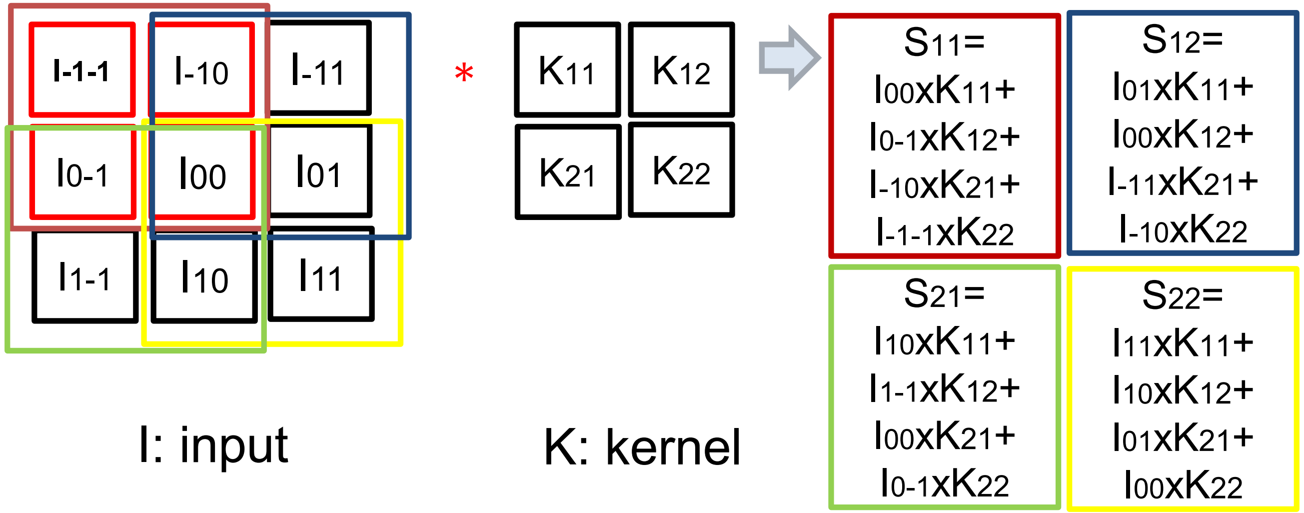
$$s(t) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m,j-n)$$



2D DATA

- Convolution is commutative

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



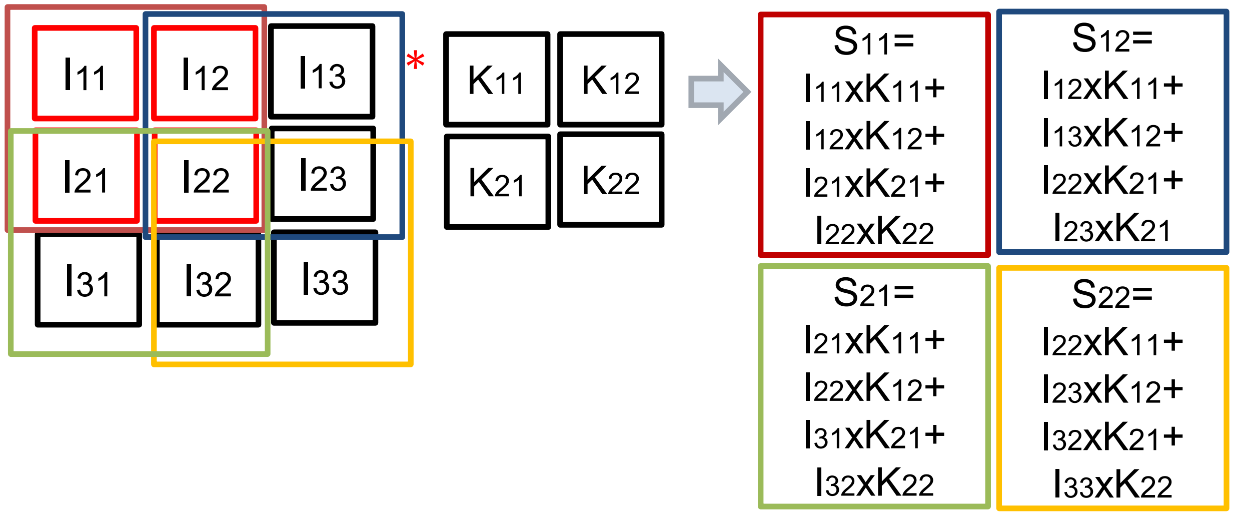
EXTENDED DEFINITION OF CONVOLUTION

- Cross correlation function (without flipping)

2D Kernel

2D Input

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



- Note: Many machine learning libraries implement cross-correlation but call it convolution.

Cross correlation

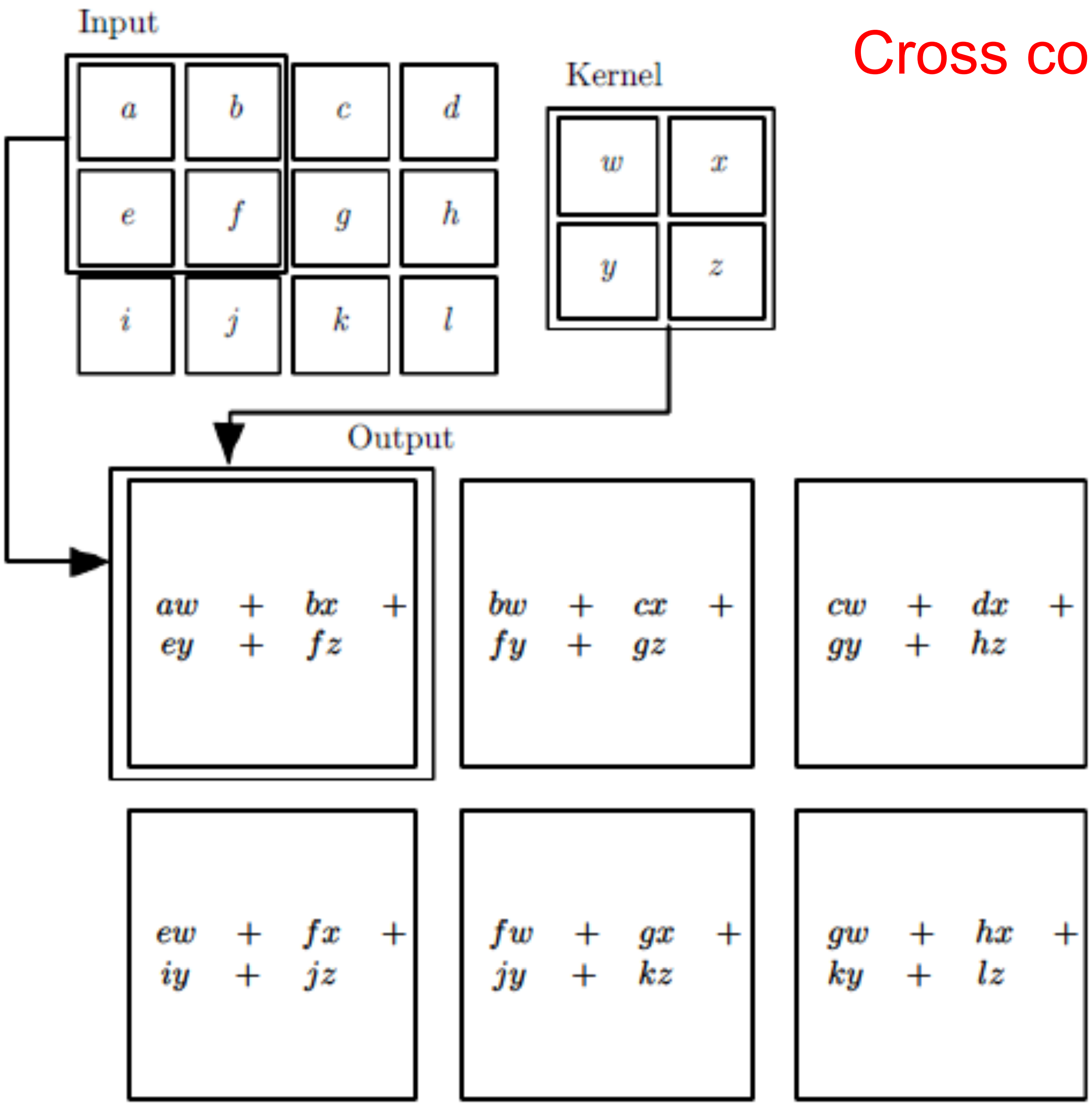


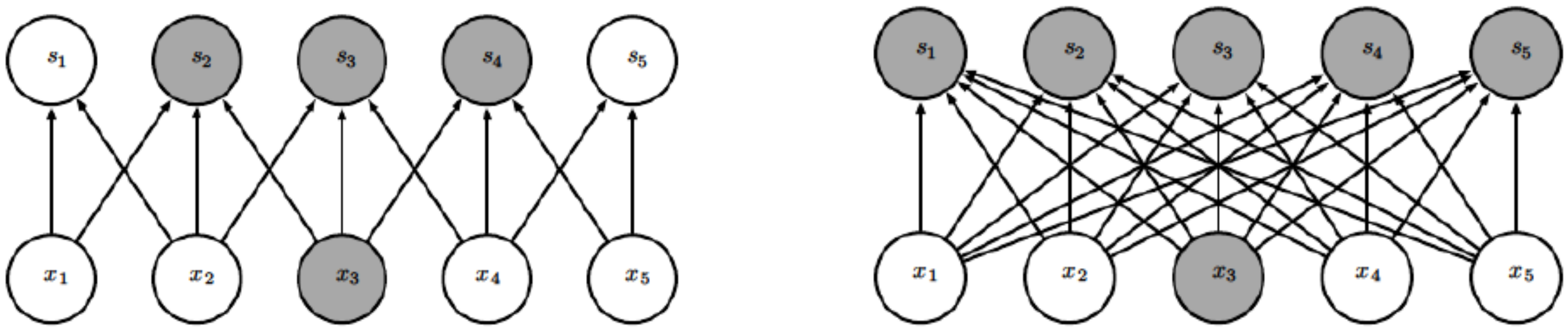
Figure 9.1 of Goodfellow et al. 2016:

MOTIVATION

- Convolution enable the following:
  - Sparse interactions,
  - Parameter sharing
  - Equivariant representations.
- Provides a means for working with inputs of variable size.

SPARSE INTERACTIONS

- Convolutional networks typically have sparse interactions
- Q: How can it be done?
- A: Making the kernel smaller than the input.



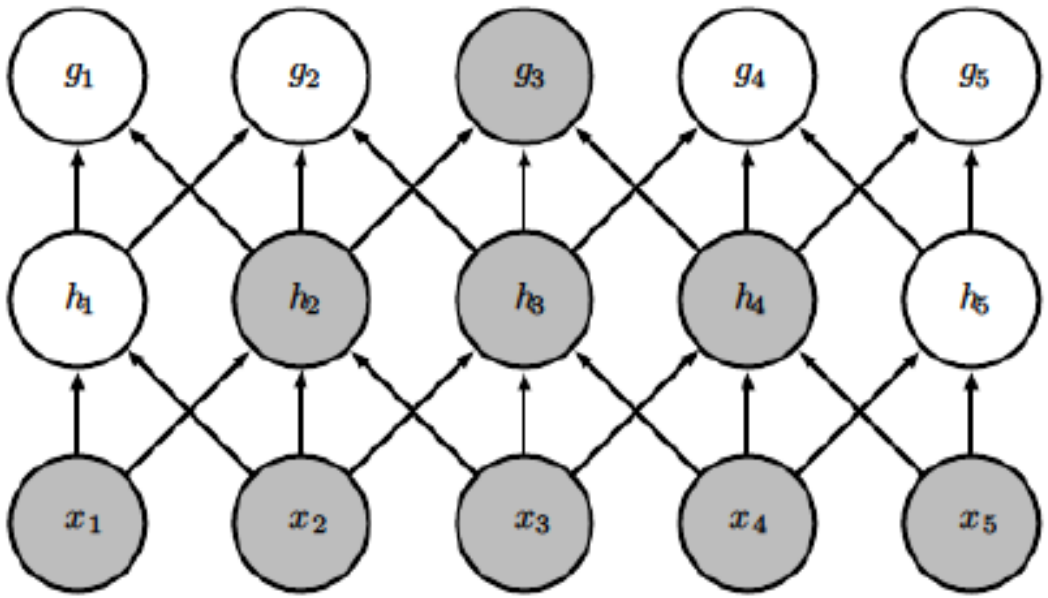
CNN traditional NN

- Q: Why is it beneficial?
- A:1) fewer parameters: reduces the memory requirements and improves its statistical efficiency.

  2) computing the output requires fewer operations.

SPARSE INTERACTIONS

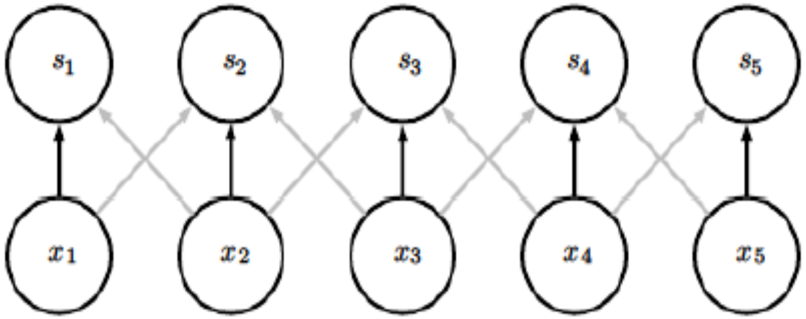
- Q: Would sparsity cause reduction on performance?
- A: Not really, since we have deep layers. Even though *direct* connections in a convolutional net are very sparse, units in the deeper layers can be *indirectly* connected to all or most of the input image.



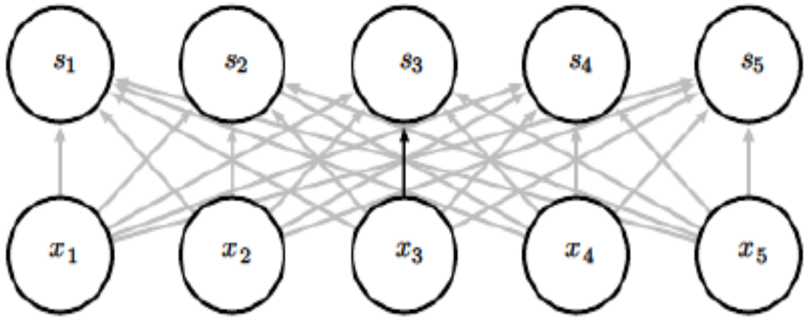
PARAMETER SHARING

- **Parameter sharing** is the uses of same parameter for more than one function in a model.
  - Note: In a traditional NN, each element of the weight matrix is used exactly once when computing the output of a layer.
- AKA **tied weights:**
  - Value of the weight applied to one input is tied to the value of a weight applied in other location in the CNN

PARAMETER SHARING



CNN



traditional NN

- Q: How is it beneficial?
- A: Reduce the storage requirements of the model to *k* parameters.

- Q: Does it decrease runtime of forward propagation?
- A: No, sill O(k x n)

EQUIVARIANT REPRESENTATIONS

- In case of convolution, the particular form of parameter sharing causes the layer to have a property called **equivariance to translation**
  - EX> shifting righ/left or up/down the input 2D image does not change the output of CNN

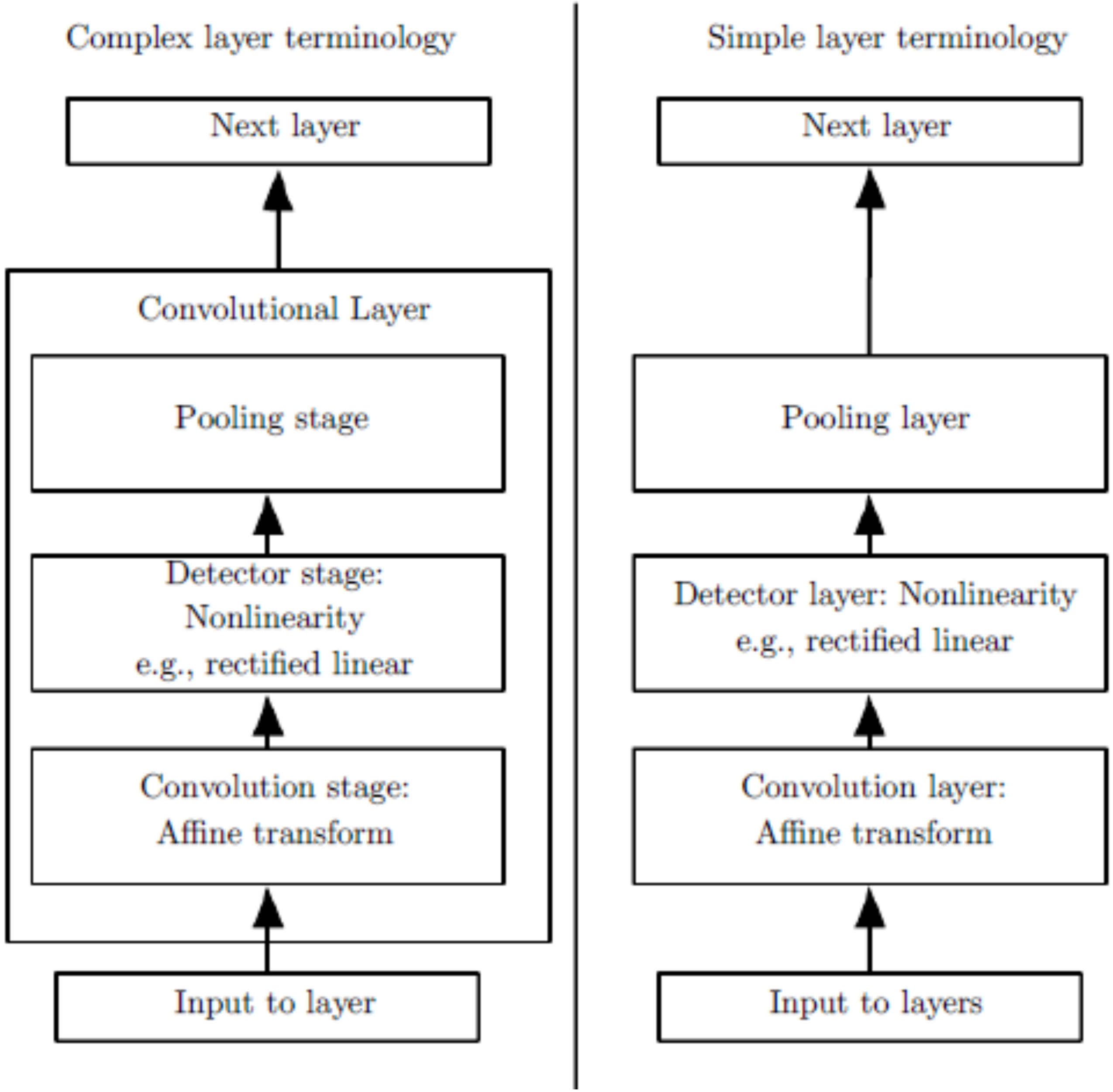
EQUIVARIANT REPRESENTATIONS

- Q: Do we always want equivariance to translation?
- A: No, we may want to learn location specific patterns.

- Q: Can convolution also allow equivalence to other types of transformations like scale and rotation?
- A: No, but **pooling** can help.

COMPONENT 2: POOLING

components of a typical convolutional neural network layer.



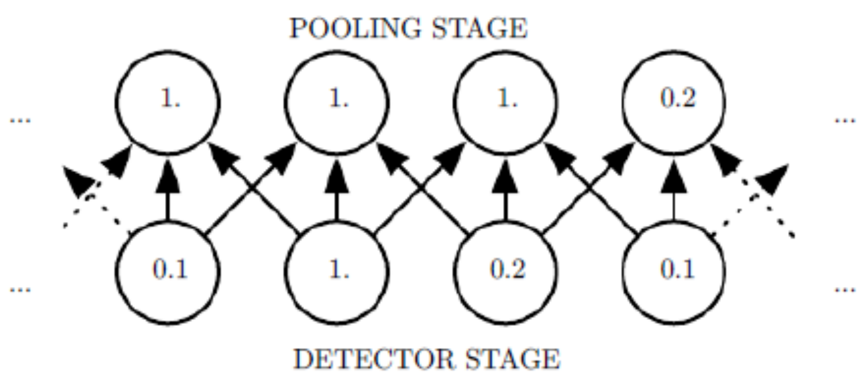
POOLING

- **Pooling function**: replaces the output of the network at a certain location with a summary statistic of the nearby outputs
- Types of pooling
  - Max pooling
  - Average pooling
  - L2 norm
  - Weighted average

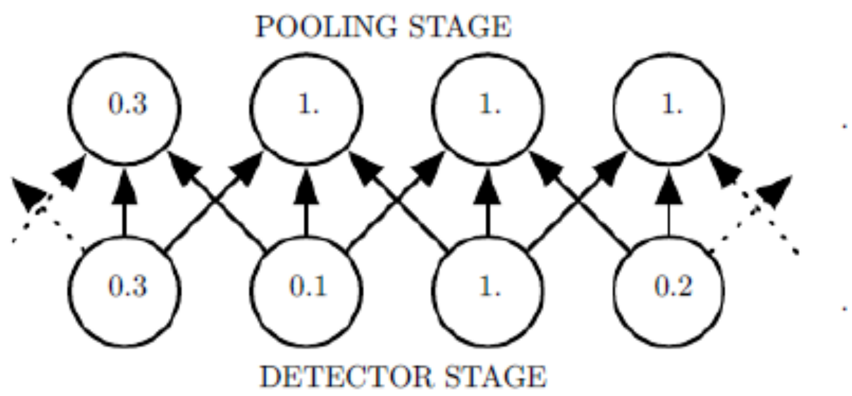
POOLING BENEFITS

- Representation becomes approximately invariant to small translations of the input.
- *"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."*

max pooling example:



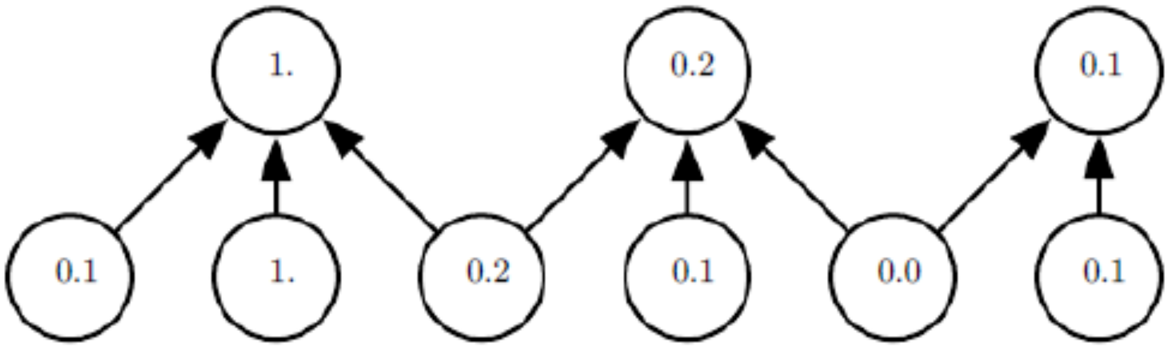
Original input



Input shifted to the right by one pixel

POOLING WITH DOWN-SAMPLING

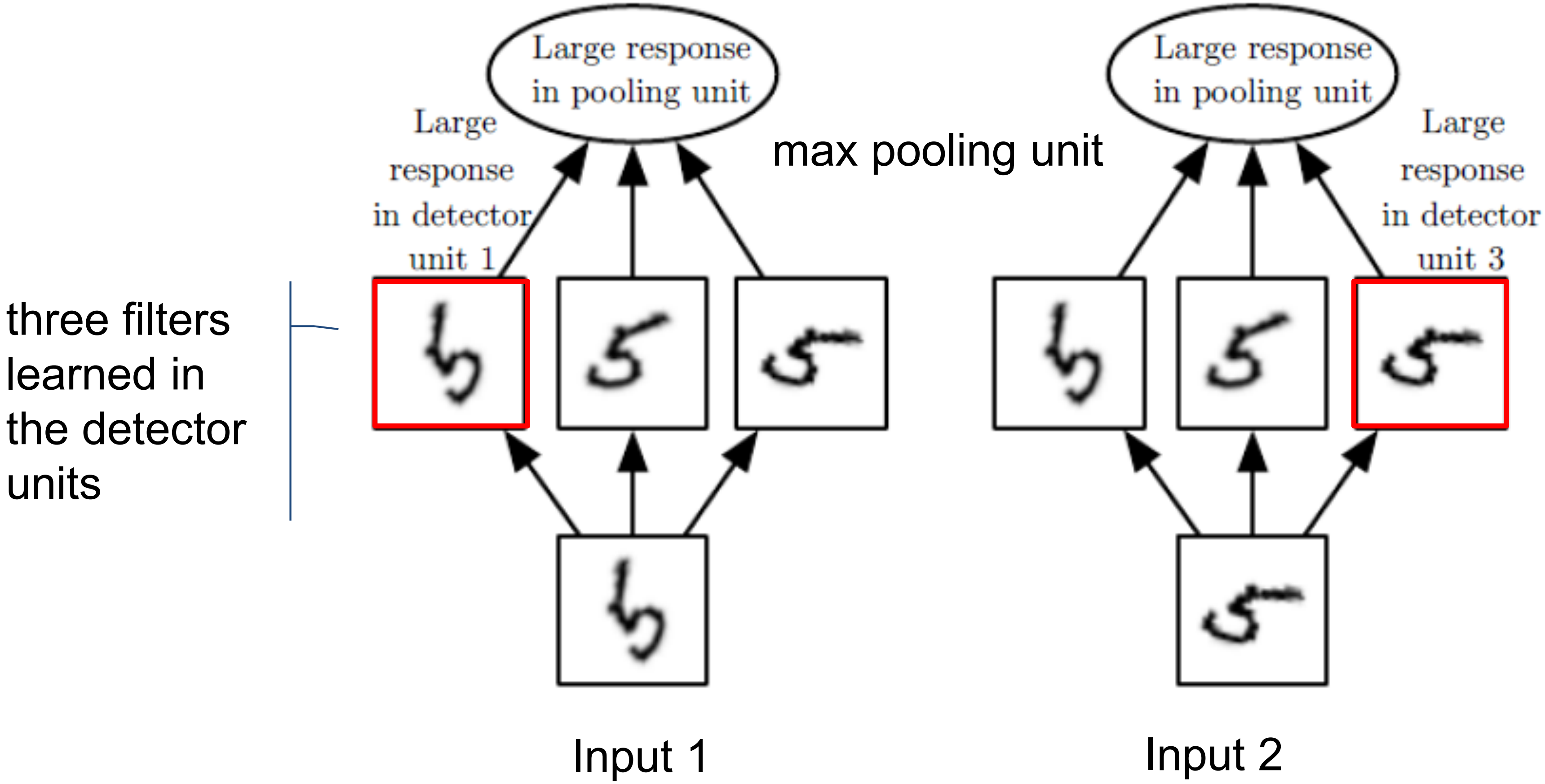
- Pooling is needed for down-sampling



Example of max-pooling with a pool width of three and a stride between pools of two.

EXAMPLE OF LEARNED INVARIANCES

CNN with three filters are intended to detect a hand-written 5.



INFINITELY STRONG PRIOR

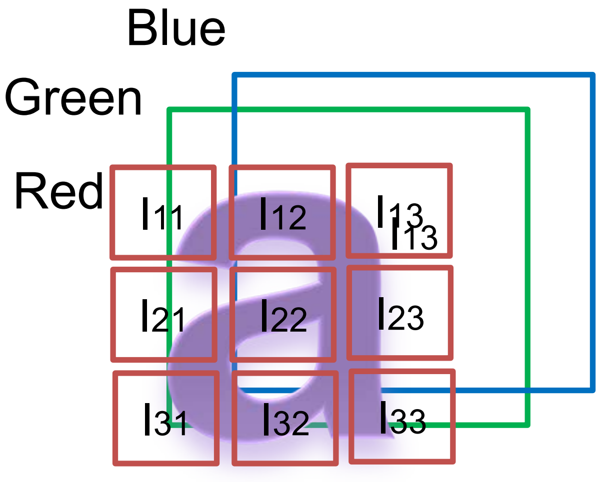
- Convolution and Pooling acts as a **infinitely strong prior**
- Infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden
  - In another word, these parameters don't need to be learned.
- Like any strong priors Convolution and pooling can cause underfitting.
- We should only compare convolutional models to other convolutional models in benchmarks of statistical learning performance.

PART 2.2: CONVOLUTION NEURAL NETWORK - PRACTICE

VARIANTS OF THE BASIC CONVOLUTION FUNCTION

- Convolution functions used in practice differ slightly compared to convolution operation as it is usually understood in the mathematical literature.

- 1) The input is usually not just a grid of real values but grid of vector-valued observations.
  - Ex> Color image has red, green and blue intensity at each pixel (3-D tensors)



VARIANTS OF THE BASIC CONVOLUTION FUNCTION

- Working example: colored 2D image
  - Assume 4-D kernel tensor K (4D)
    - Element $K_{i,l,m,n}$ giving the connection strength between a unit in channel *i* of the output and a unit in channel *l* of the input, with an offset of *m* rows and *n* columns between the output unit and the input unit.
  - Assume input consists of observed data V (3D)
    - Element $V_{i,j,k}$ giving the value of the input unit within channel *i* at row *j* and column *k* .
  - Assume output consists of Z with the same format as V (3D)
  - If Z is produced by convolving K across V without flipping K, then

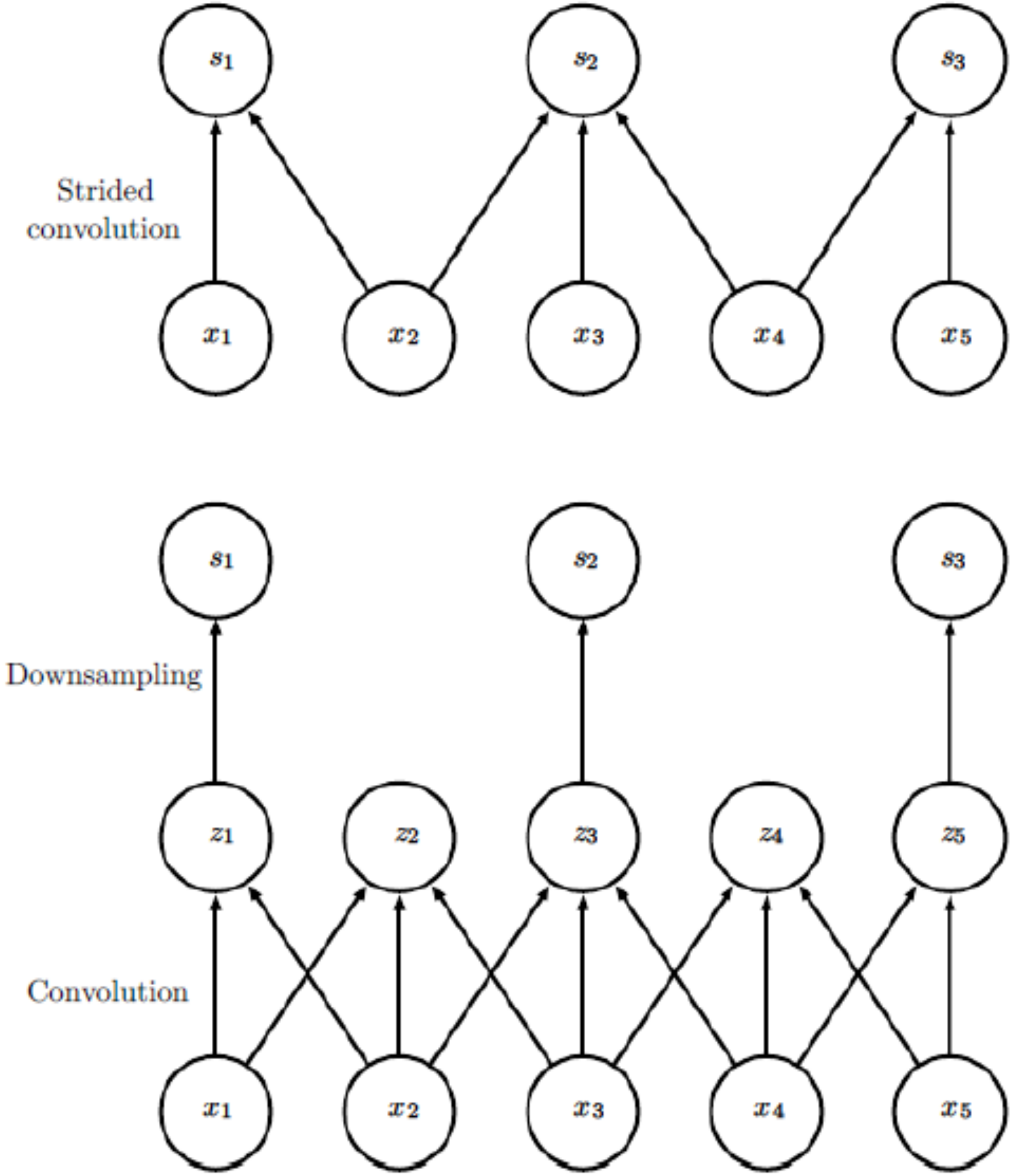
$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

DOWNSAMPLING – STRIDE

- We may want to skip over some positions of the kernel in order 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
- We refer to *s* as the **stride** of this downsampled convolution

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

Stride example
S=2

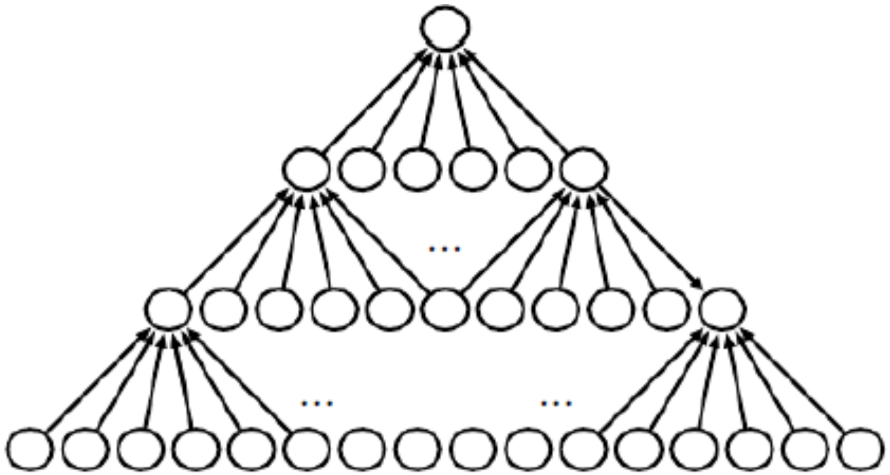


ZERO-PADDING

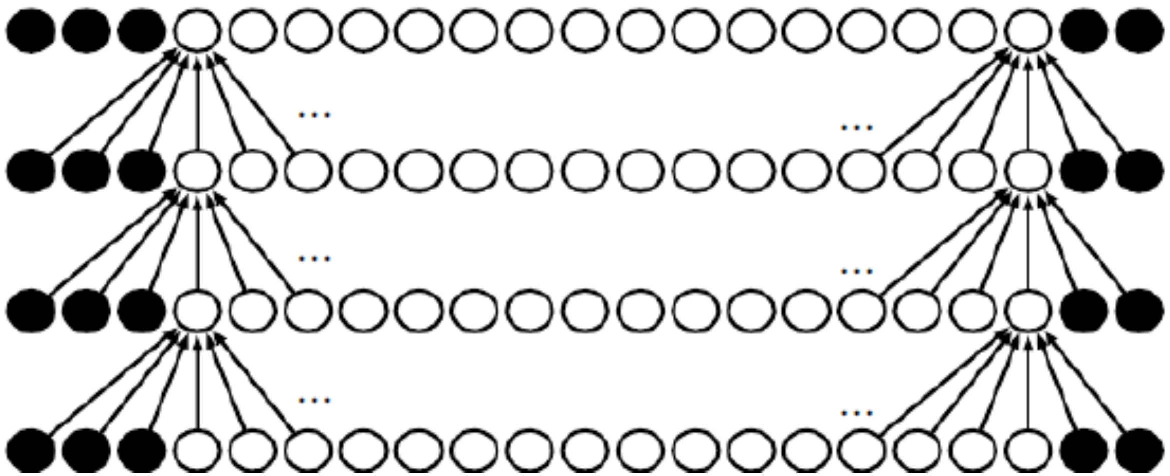
- Valid convolution:
  - Extreme case in which no zero-padding is used whatsoever, and the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the input
- Same convolution:
  - Just enough zero-padding is added to keep the size of the output equal to the size of the input
- Full convolution
  - Other extreme case where enough zeroes are added for every pixel to be visited k times in each direction, resulting an output image of width $m + k - 1$.

ZERO PADDING

Valid convolution



Same convolution



WHAT IF WE DON'T WANT TO CONVOLUTE?

- In some application, it's more appropriate to not use convolution, but rather locally connected layers
- Q: How do we modify the model?
- A: **Unshared convolution** approach

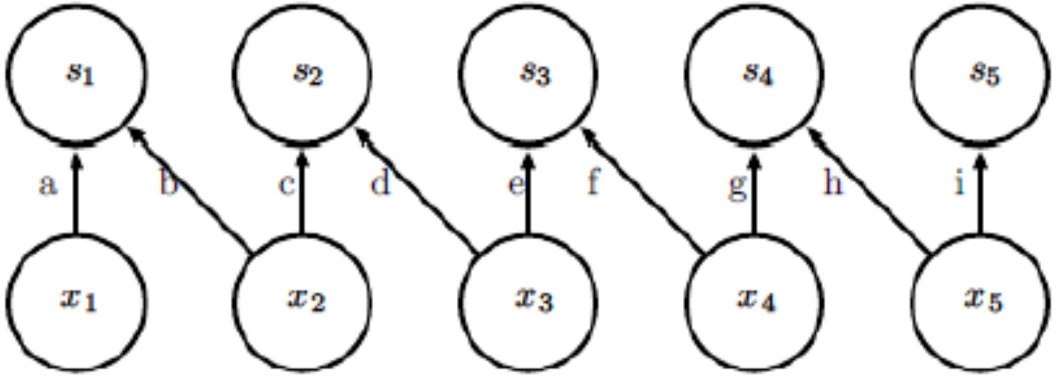
UNSHARED CONVOLUTION

- Assume weight are in a 6-D tensor W
  - $w_{i,j,k,l,m,n}$: *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.
- Pro: Able to learn location sensitive filters.
- Con: Memory requirements increase only by a factor of the size of the entire output feature map.

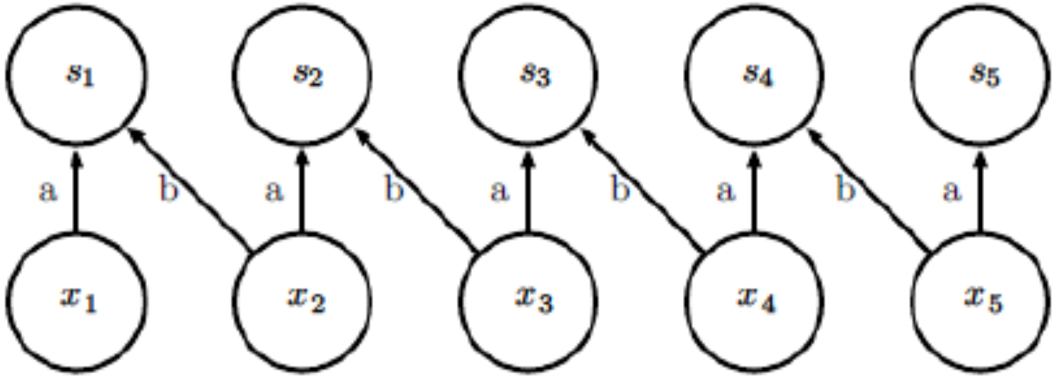
$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} w_{i,j,k,l,m,n}$$

Unshared convolution, aka locally connected layer, since it is similar operation to discrete convolution with a small kernel, but without sharing parameters across locations

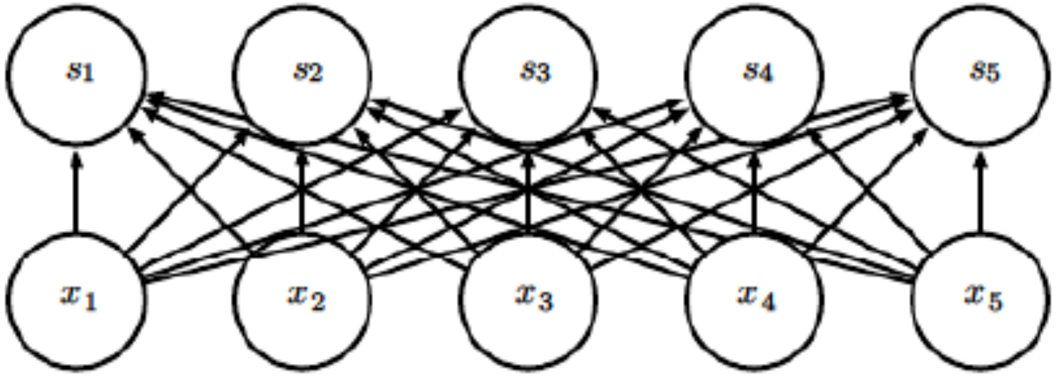
Unshared convolution
(locally connected layer)



Convolution
(parameter sharing)



Fully connected
(no parameter sharing)



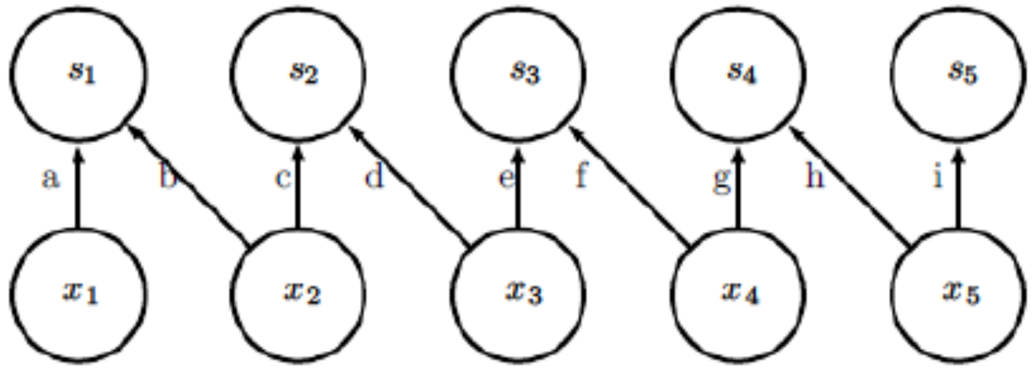
TILED CONVOLUTION

- **Tiled convolution** learn a set of kernels that is rotated through as we move through space, rather than learning a separate set of weights at *every* spatial location as in locally connected layer.

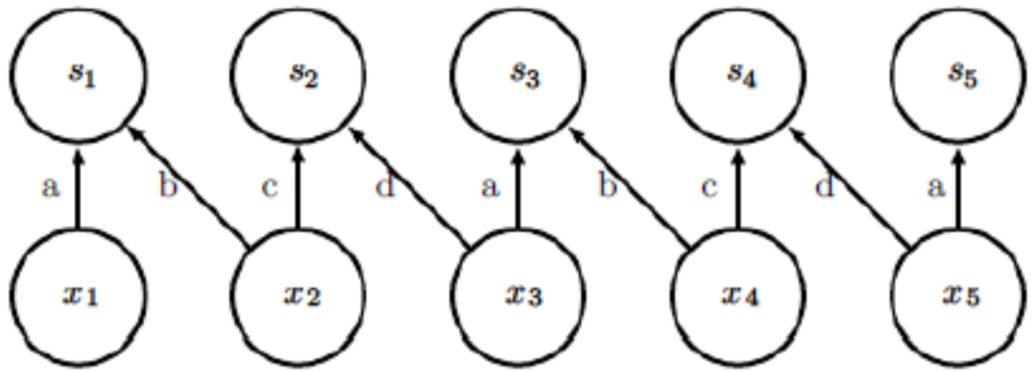
$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n,j\%t+1,k\%t+1}$$

- Pro:
  - It Offers a compromise between a convolutional layer and a locally connected layer.
  - Memory requirements for storing the parameters will incre ase only by a factor of the size of this set of kernels

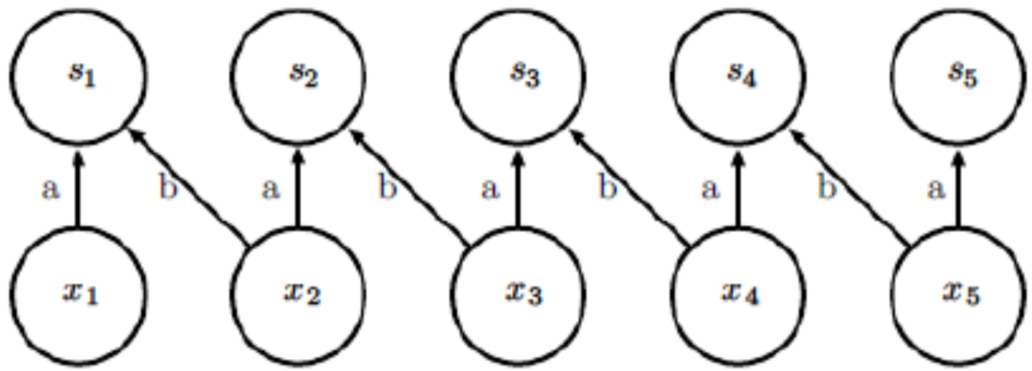
Locally connected layers



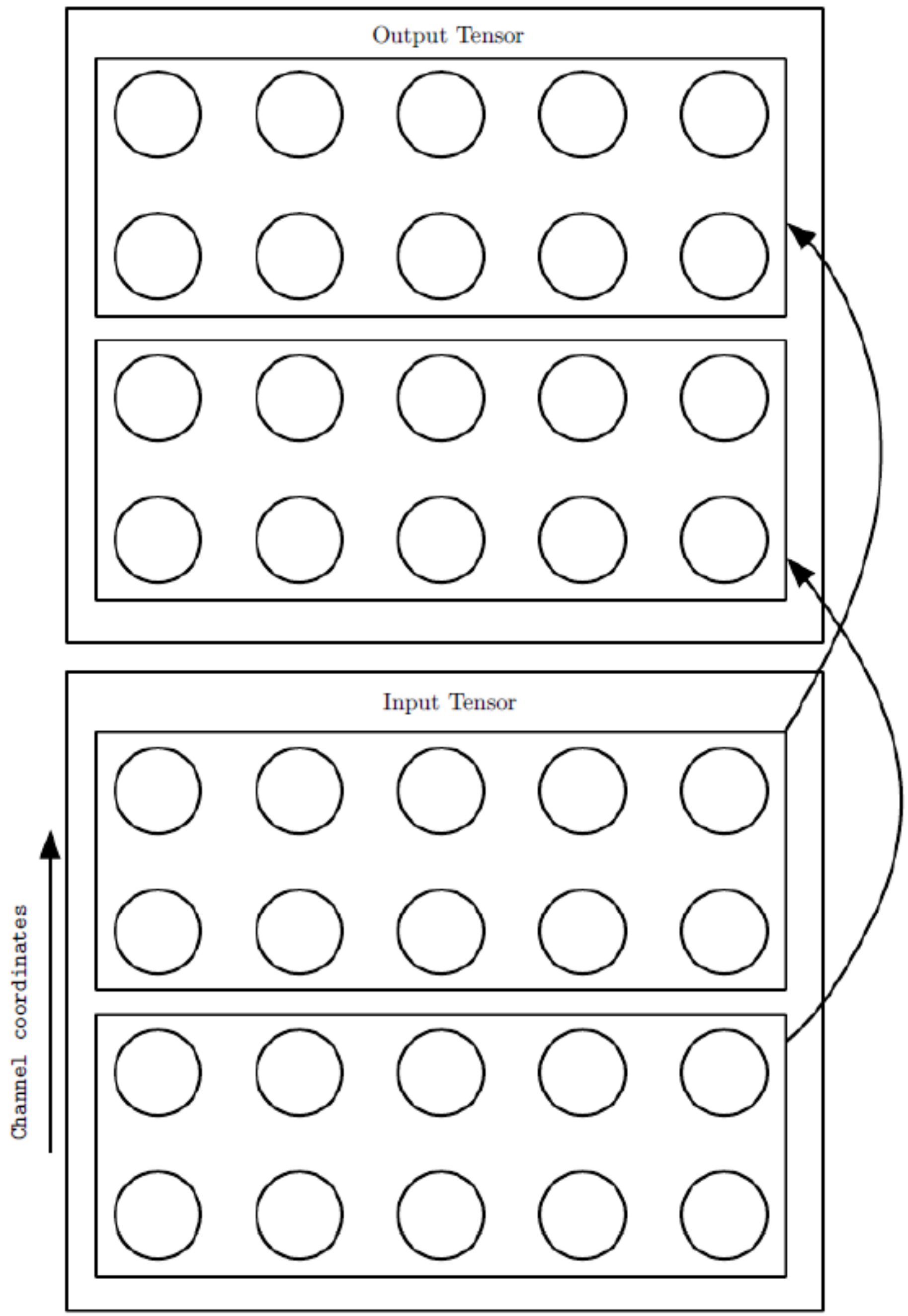
Tiled convolution



Standard convolution



convolutional network
with the first two output
channels connected



COMPUTE THE GRADIENTS IN CNN

- 3 operations needed to compute the gradients in CNN
  - Convolution,
  - Backprop from output to weights, and
  - Backprop from output to inputs

* Details omitted

CHOOSING POOLING METHODS

- **No best answer here** but:
- Some theoretical work gives guidance as to which kinds of pooling one should use in various situations (Boureau *et al.* 2010).
- It is also possible to dynamically pool features together, for example, by running a clustering algorithm on the locations of interesting features (Boureau *et al*., 2011). 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 (Jia *et al*., 2012).

RANDOM OR UNSUPERVISED FEATURES

- Feature learning in CNN is very expensive
  - Every gradient step requires complete run of forward propagation and backward propagation

- Three ways to obtaining convolution kernels without supervised training.
  - Initialize them randomly
  - Design them by hand
  - Learn the kernels with an unsupervised criterion
    - *Apply k*-means clustering to small image patches, then use each learned centroid as a convolution kernel.
    - Greedy layer-wise pretraining (convolutional deep belief network)

GATHER MORE DATA OR RETUNE THE MODEL?

- It is often much better to gather more data than to improve the learning algorithm. But data can be expensive.
- Measure the training set performance.
  - Poor training set performance: the learning algorithm is not using the training data properly.
    - Try increasing the size of the model - more layers or more hidden units
    - Try improving the learning algorithm - tune the hyperparameters
    - If the two does not work, quality of the training data may be poor.

GATHER MORE DATA OR RETUNE THE MODEL?

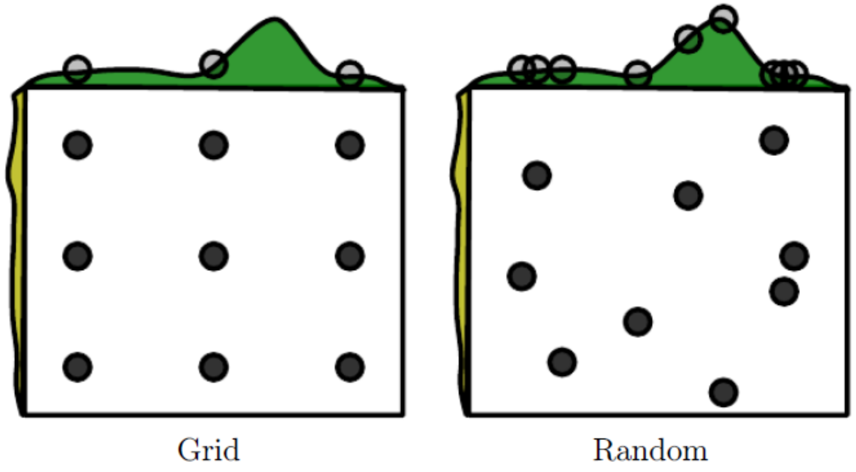
- Acceptable training set performance, then measure the performance of test set.
  - If test set performance is good enough – no more work to do.
  - If test set performance is bad (big gap between training and testing),
    - Gathering more data - most effective solutions.
    - Reduce the size of the model by adjusting hyperparameters, e.g., weight decay coefficients,
    - Adding regularization strategies such as dropout.

SELECTING HYPERPARAMETERS

- Choosing hyperparameters manually
  - Requires understanding what the hyperparameters do and how machine learning models achieve good generalization
  - Requires understanding of how the data behaves
- Choosing hyperparameters automatically
  - Computationally costly

CHOOSING HYPERPARAMETERS AUTOMATICALLY

- Grid search.
- Random Search
  - Random search finds good solutions faster than grid search
- Combination approach
  - Gird search then random search on selected range of values
- Model-Based optimization
  - Difficult
  -



Grid search vs random search
- two hyperparameter case

PARAMETER INITIALIZATION

- Important to initialize all weights to small random values.
- Bias terms can be initialized to zero or to small positive values.