Lecture 6

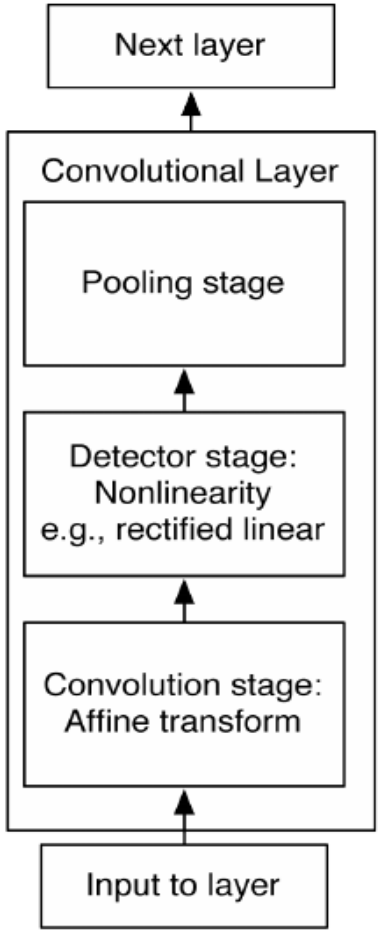
Convolutional Neural Network (CNNs)

Slides are partially based on Book, Deep Learning

by Bengio, Goodfellow, and Aaron Courville, 2015

Convolutional Networks

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.



Convolution

This operation is called convolution.

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

The convolution operation is typically denoted with an asterisk:

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

Discrete convolution

If we now assume that *x* and *w* are defined only on integer *t*, we can define the discrete convolution:

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

In practice

we often use convolutions over more than one axis at a time.

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

The input is usually a multidimensional array of data.

The kernel is usually a multidimensional array of parameters that should be learned.

we assume that these functions are zero everywhere but the finite set of points for which we store the values.

we can implement the infinite summation as a summation over a finite number of array elements.

convolution and cross-correlation

convolution is commutative

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

Cross-correlation,

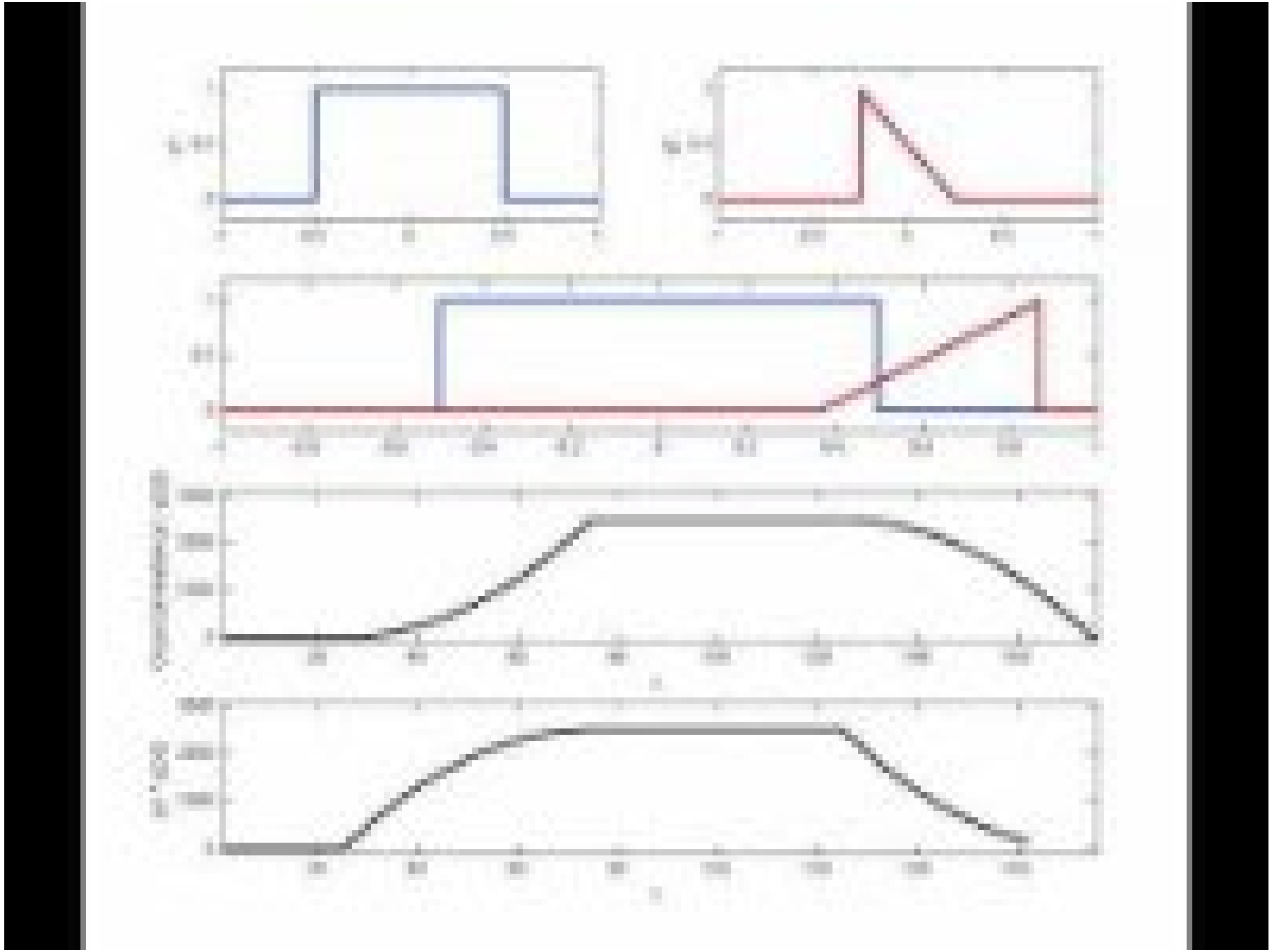
$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m, j+n] K[m,n]$$

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

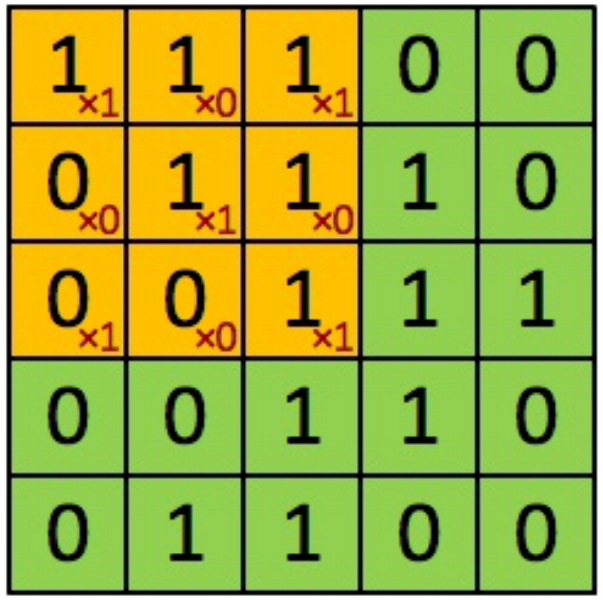
https://www.youtube.com/watch?v=Ma0YONjMZLI

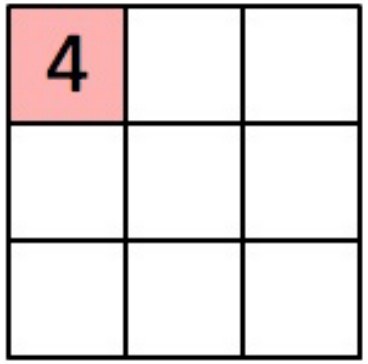
Fig 9.1

Discrete convolution can be viewed as multiplication by a matrix.

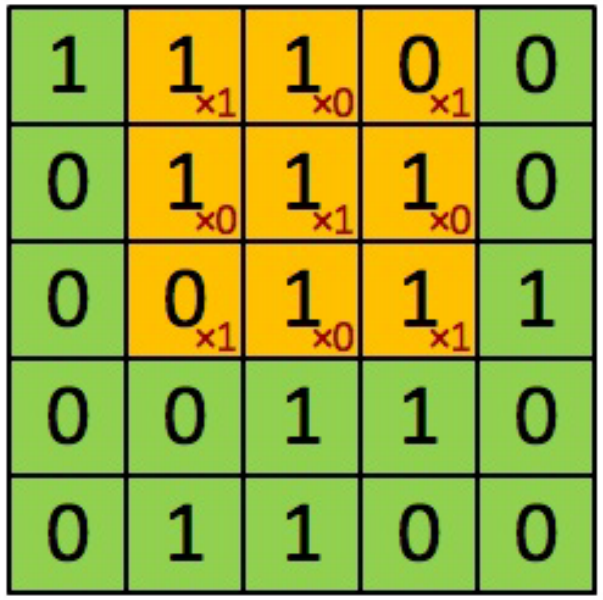


Convolutions

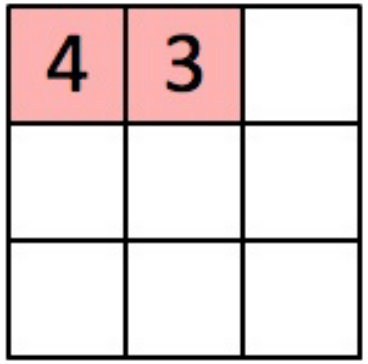




Convolutions

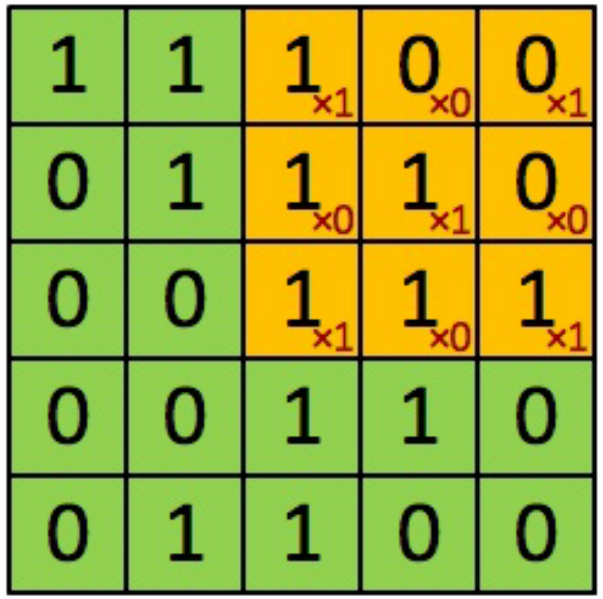


Image

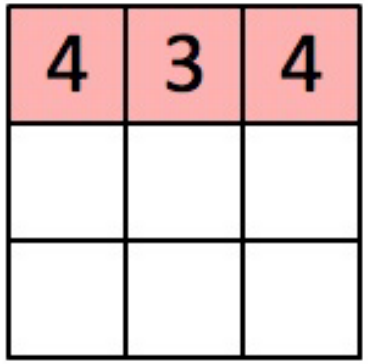


Convolved Feature

Convolutions

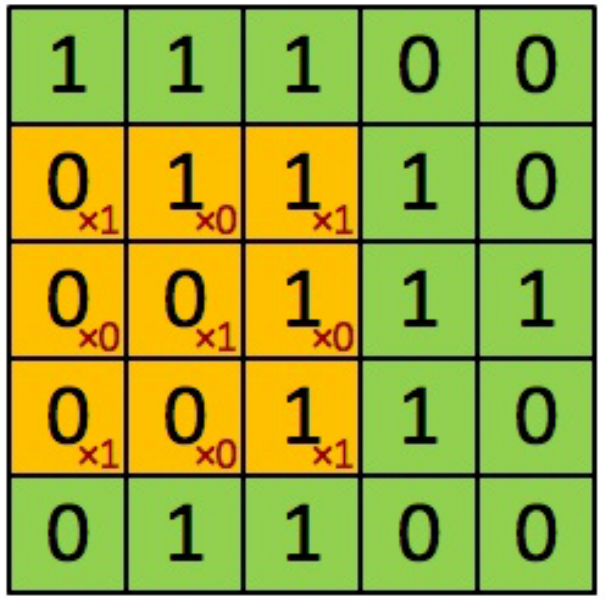


Image

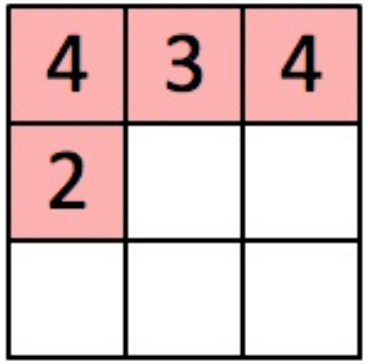


Convolved Feature

Convolutions

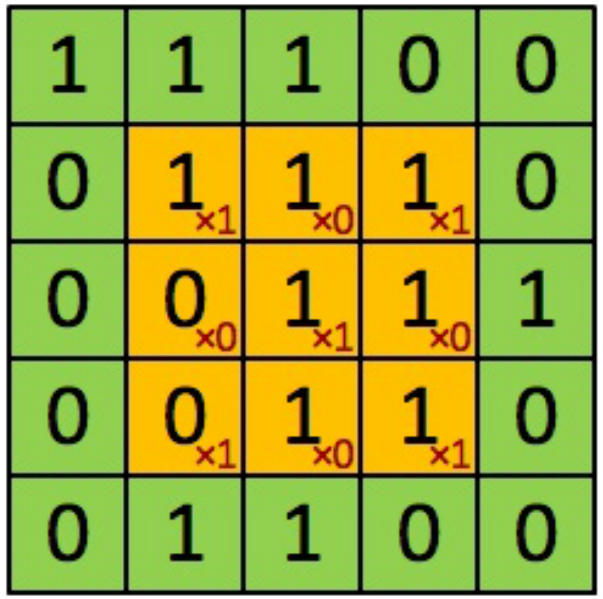


Image



Convolved Feature

Convolutions

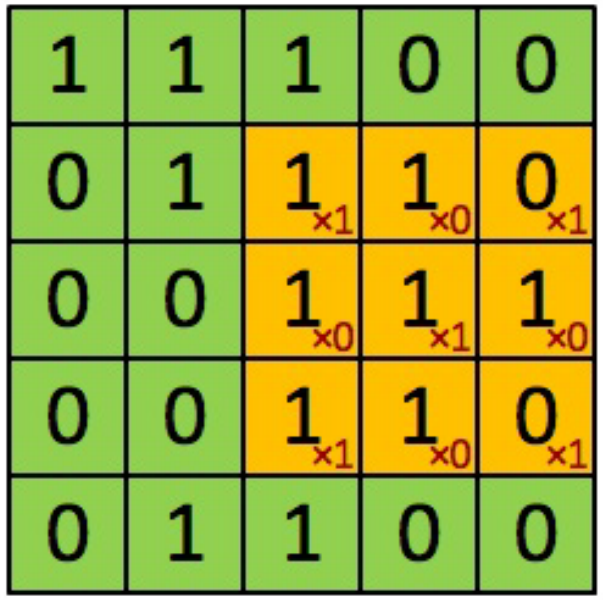


Image

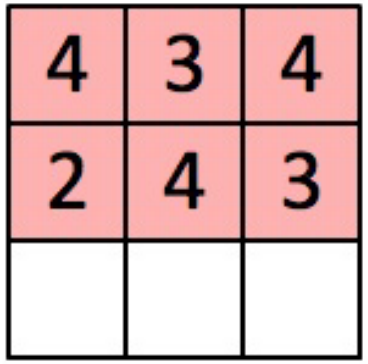
4	3	4
2	4	

Convolved Feature

Convolutions

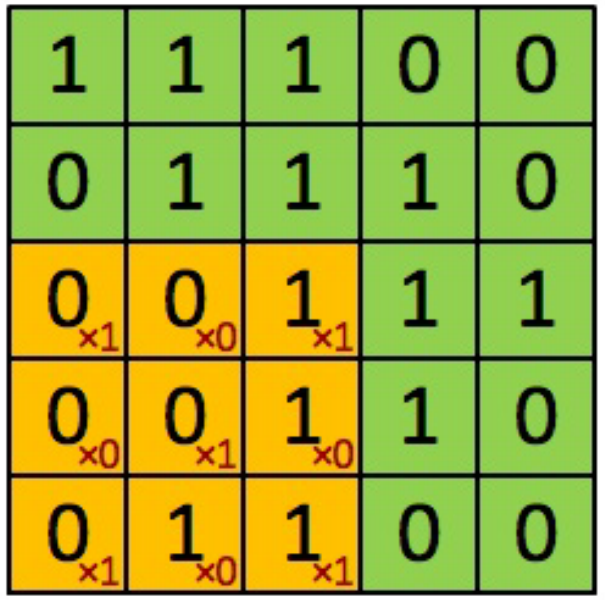


Image



Convolved Feature

Convolutions

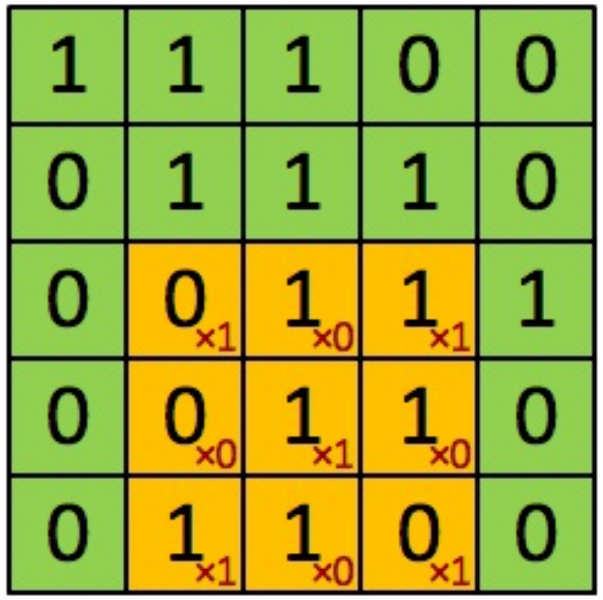


Image

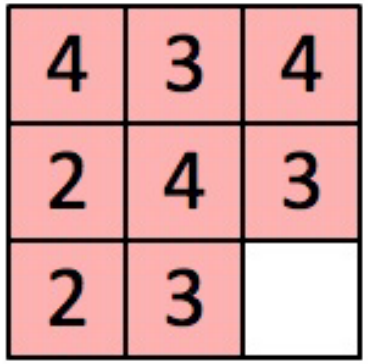
4	3	4
2	4	3
2		

Convolved Feature

Convolutions

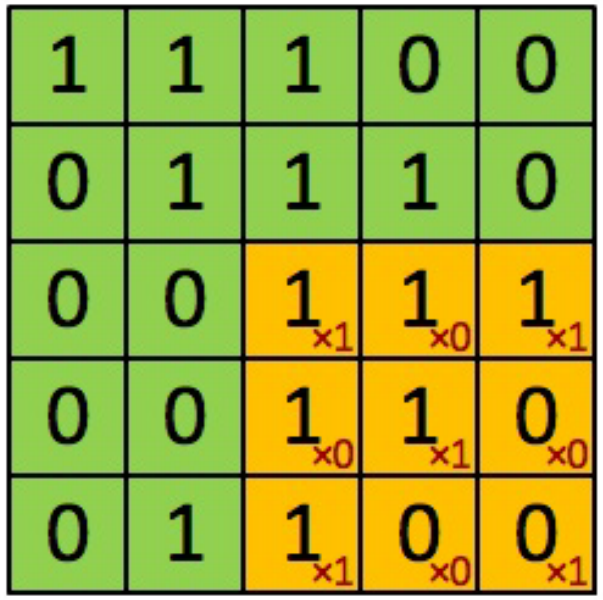


Image



Convolved Feature

Convolutions



Image

4	3	4
2	4	3
2	3	4

Convolved Feature

Sparse interactions

In feed forward neural network every output unit interacts with every input unit.

Convolutional networks, typically have sparse connectivity (sparse weights)

This is accomplished by making the kernel smaller than the input

Sparse interactions

When we have 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).

limit the number of connections each output may have to k, then requires only $k \times n$ parameters and $O(k \times n)$ runtime.

Parameter sharing

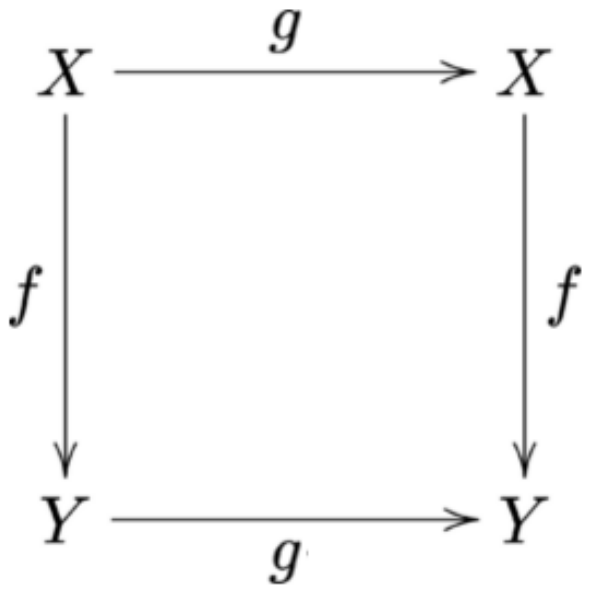
In a traditional neural net, each element of the weight matrix is multiplied by one element of the input. i.e. It is used once when computing the output of a layer.

In CNNs each member of the kernel is used at every position of the input

Instead of learning a separate set of parameters for every location, we learn only one set.

Equivariance

A function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$.



Equivariance

A convolutional layer have equivariance to translation.
For example

$$g(x)[i] = x[i-1]$$

If we apply this transformation to *x*, then apply convolution, the result will be the same as if we applied convolution to *x*, then applied the transformation to the output.

Equivariance

For images, convolution creates a 2-D map of where certain features appear in the input.

Note that convolution is not equivariant to some other transformations, such as changes in the scale or rotation of an image.

Convolutional Networks

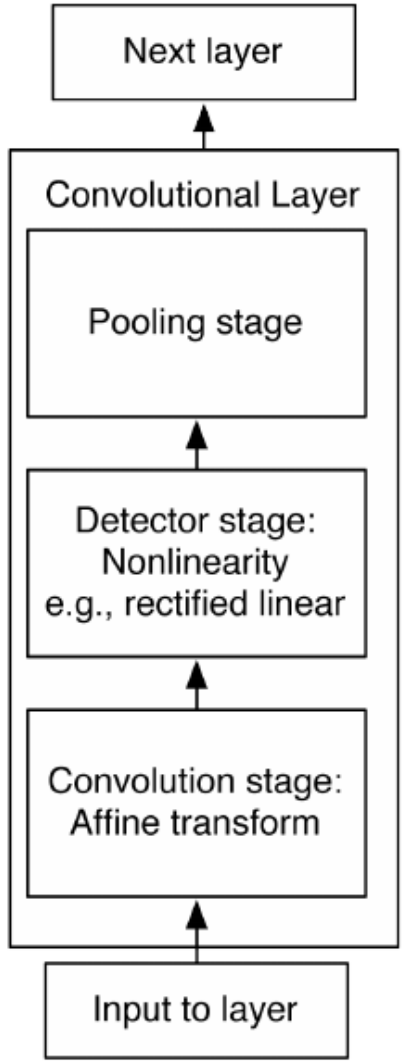
The first stage (Convolution):

The layer performs several convolutions in parallel to produce a set of preactivations.

The second stage (Detector):

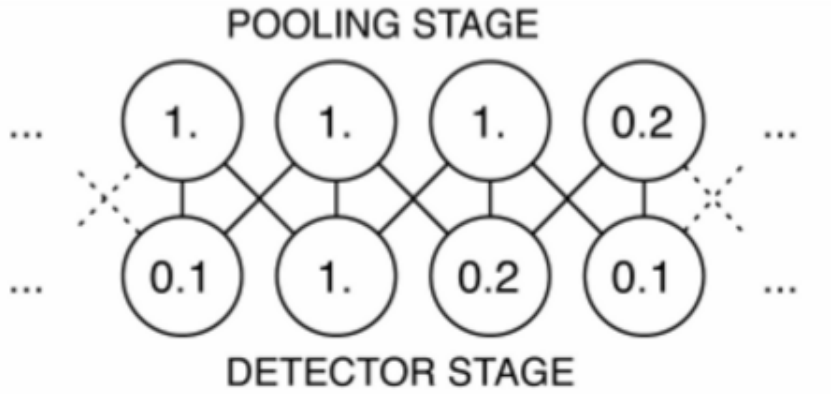
Each preactivation is run through a nonlinear activation function (e.g. rectified linear).

The third stage (Pooling)



Popular Pooling functions

The maximum of a rectangular neighborhood (Max pooling operation)

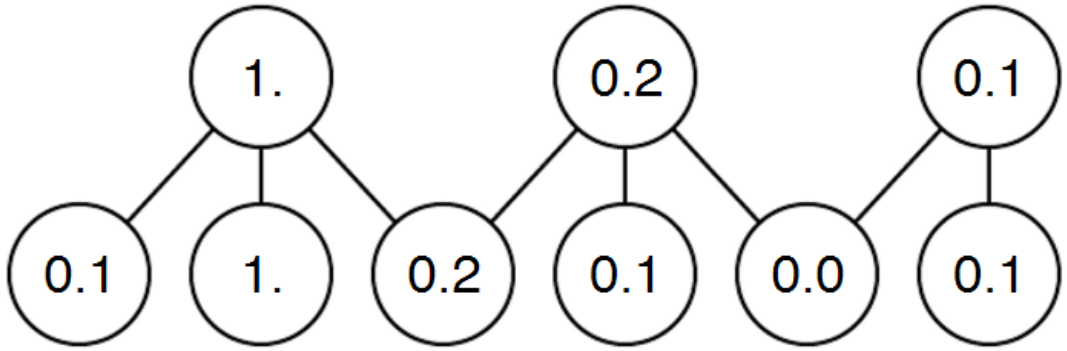


The average of a rectangular neighborhood.

The L2 norm of a rectangular neighborhood.

A weighted average based on the distance from the central pixel.

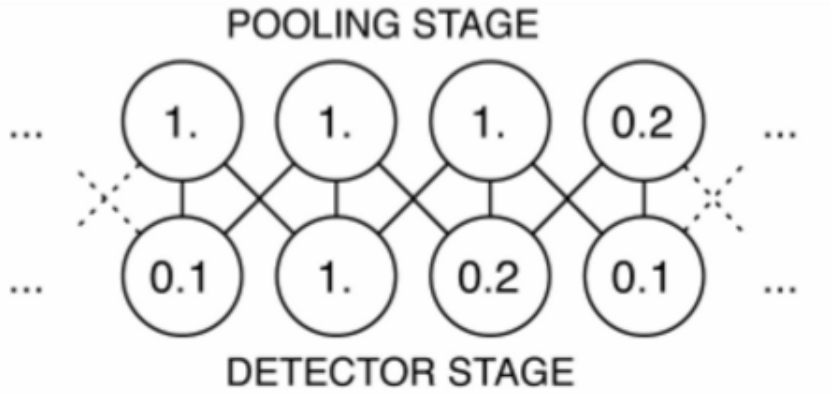
Pooling with downsampling



Max-pooling with a pool width of 3 and a stride between pools of 2. This reduces the representation size by a factor of 2,which reduces the computational and statistical burden on the next layer.

Popular Pooling functions

The maximum of a rectangular neighborhood (Max pooling operation)

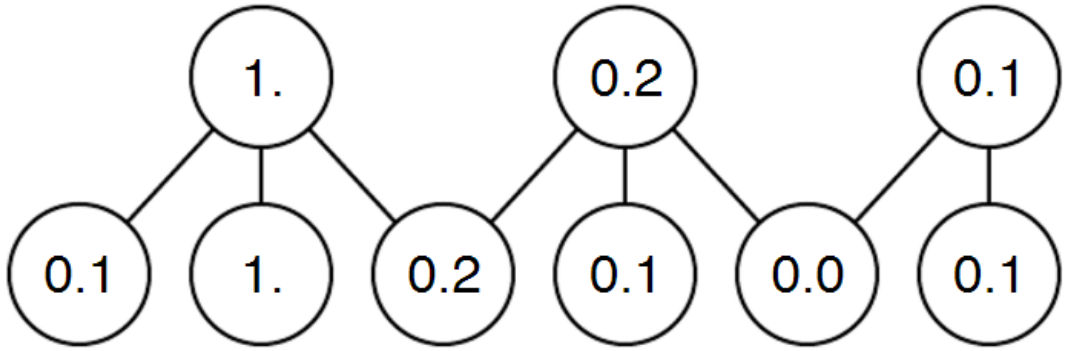


The average of a rectangular neighborhood.

The L2 norm of a rectangular neighborhood.

A weighted average based on the distance from the central pixel.

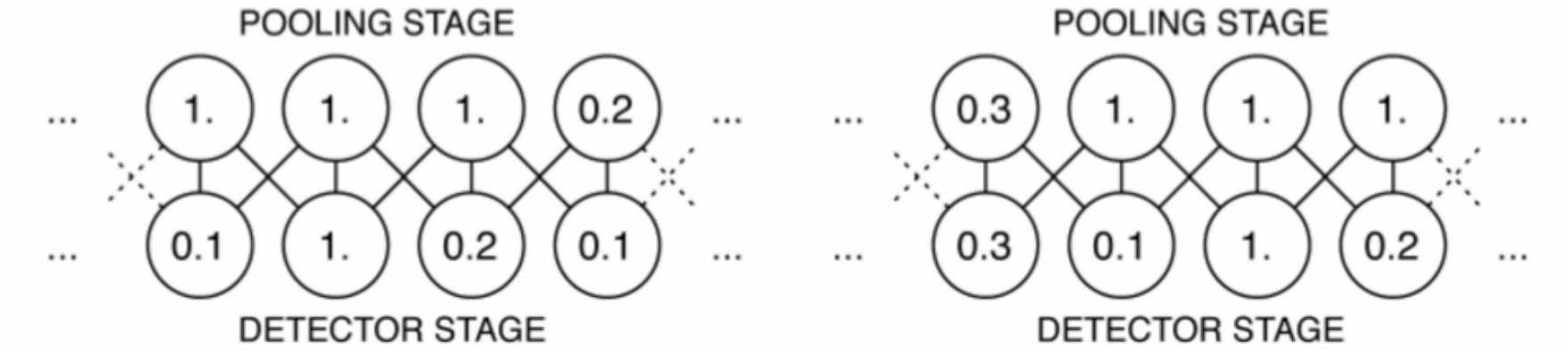
Pooling with downsampling



Max-pooling with a pool width of 3 and a stride between pools of 2. This reduces the representation size by a factor of 2,which reduces the computational and statistical burden on the next layer.

Pooling and translations

pooling helps to make the representation become invariant to small translations of the input.



In Right panel, the input has been shifted to the right by 1 pixel. Every value in the bottom row has changed, but only half of the values in the top row have changed.

Pooling and translations

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: In a face, we need not know the exact location of the eyes.

Pooling: inputs of varying size

Example: we want to classify images of variable size.

The input to the classification layer must have a fixed size.

In the final pooling output (for example) four sets of summary statistics, one for each quadrant of an image, regardless of the image size.

Pooling: inputs of varying size

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).

i.e. a different set of pooling regions for each image.

Learn a single pooling structure that is then applied to all images (Jia et al., 2012).

Convolution and Pooling as an Infinitely Strong Prior

A weak prior is a prior distribution with high entropy, such a Gaussian distribution with high variance

A strong prior has very low entropy, such as a Gaussian distribution with low variance.

An infinitely strong prior places zero probability on some parameters and says

a convolutional net is similar to a fully connected net with an infinitely strong prior over its weights.

Convolution and Pooling as an Infinitely Strong Prior

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

use of convolution as 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.

Convolution and Pooling as an Infinitely Strong Prior

The use of pooling is in infinitely strong prior that each unit should be invariant to small translations.

convolution and pooling can cause underfitting

Practical issues

The input is usually not just a grid of real values.

It is a grid of vector-valued observations.

For example, a color image has a red, green, and blue intensity at each pixel.

Practical issues

When working with images, we usually think of the input and output of the convolution as 3-D tensors.

One index into the different channels and two indices into the 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.

Training

Suppose we want to train a convolutional network that incorporates convolution of kernel stack K applied to multi-channel image V with stride s: $c(K; V; s)$

Suppose we want to minimize some loss function $J(V; K)$.
During forward propagation, we will need to use c itself to output Z,

Z is propagated through the rest of the network and used to compute J.

Training

During backpropagation, we will receive a tensor *G* such that

$$G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(V, K)$$

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

$$g(G, V, s)_{i,j,k,l} = \frac{\partial}{\partial Z_{i,j,k}} J(V, K) = \sum_{m,n} G_{i,m,n} V_{j,ms+k,ns+l}.$$

If this layer is not the bottom layer of the network, we?ll need to compute the gradient with respect to *V* in order to backpropagate the error farther down. To do so, we can use a function

$h(K, G, s)i, j, k =$

$$\frac{\partial}{\partial V_{i,j,k}} J(V, K) = \sum_{l,m|sl+m=j} \sum_{n,p|sn+p=k} \sum_{q} K_{q,i,m,p} G_{i,l,n}.$$

Random or Unsupervised Features

the most expensive part of convolutional network training is learning the features.

When performing supervised training with gradient descent, every gradient step requires a complete run of forward propagation and backward propagation through the entire network.

use features that are not trained in a supervised fashion.

Random or Unsupervised Features

There are two basic strategies for obtaining convolution kernels without supervised training.

One is to simply initialize them randomly.

learn them with an unsupervised criterion.

Random or Unsupervised Features

Random filters often work surprisingly well in convolutional networks.

layers consisting of convolution following by pooling naturally become frequency selective and translation invariant when assigned random weights.

that this provides an inexpensive way to choose the architecture of a convolutional network: first evaluate the performance of several convolutional network architectures by training only the last layer, then take the best of these architectures and train the entire architecture using a more expensive approach.

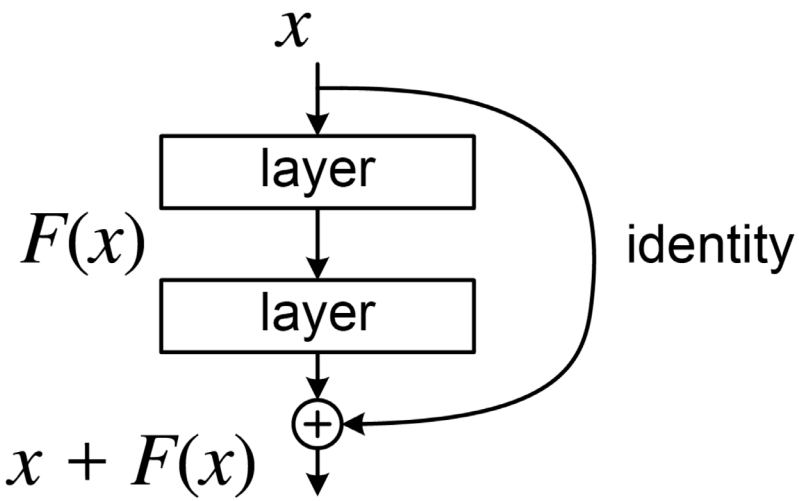
Residual Networks (ResNet)

- ResNet, short for Residual Networks, was introduced by Kaiming He et al. from Microsoft Research in 2015.

- It brought a significant breakthrough in deep learning by enabling the training of much deeper networks, addressing the vanishing gradient problem.

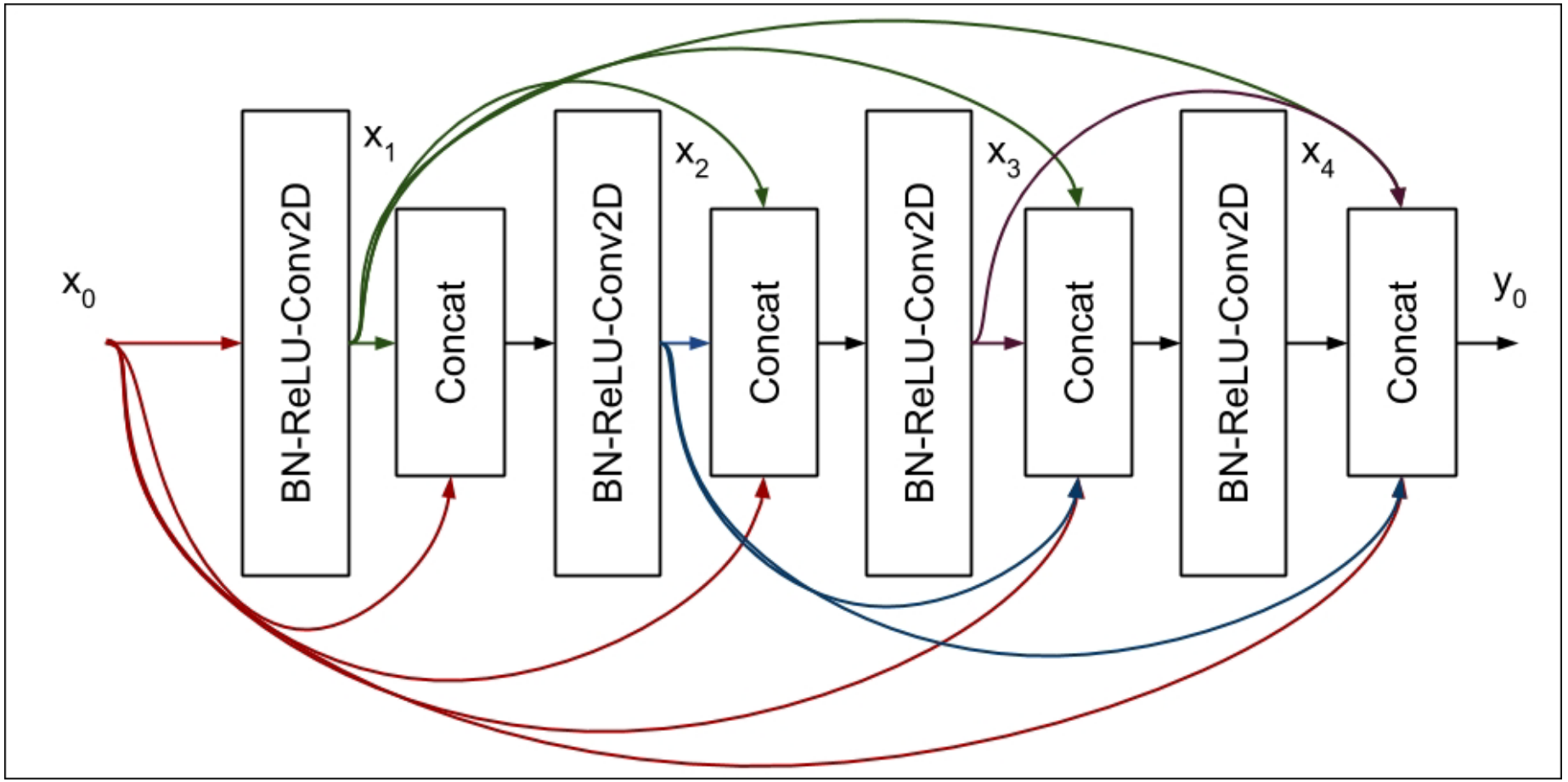
Residual Networks (ResNet)

- ResNet introduces the concept of skip connections (or residual connections) that allow the gradient to be directly backpropagated to earlier layers.
- Skip connections help in overcoming the degradation problem, where the accuracy saturates and then degrades rapidly as the network depth increases.



- .
- **Variants and Applications:**
  - Several variants of ResNet have been developed, including ResNet-50, ResNet-101, and ResNet-152, differing in the number of layers.
  - ResNet has been widely adopted for various computer vision tasks, including image classification, object detection, and facial recognition.

- DenseNet, short for Densely Connected Networks, was introduced by Gao Huang et al. in 2017.
- It is known for its efficient connectivity between layers, which enhances feature propagation and reduces the number of parameters.


x_0
BN-ReLU-Conv2D
x_1
Concat
BN-ReLU-Conv2D
x_2
Concat
BN-ReLU-Conv2D
x_3
Concat
BN-ReLU-Conv2D
x_4
Concat
y_0

- **Key Feature – Dense Connectivity:**
- In DenseNet, each layer receives feature maps from all preceding layers and passes its own feature maps to all subsequent layers.
- This dense connectivity improves the flow of information and gradients throughout the network, mitigating the vanishing gradient problem.