Neural Networks for Computing

Convolution Networks

Hikmat Farhat

October 30, 2017

Introduction

- Convolution networks or Convolution neural networks as a special kind of NN
- Usually used for data that are know to have a grid-like topology
- Examples are time-series data that can be thought of as a 1 dimensional grid of samples taken at regular intervals
- Or image data with can be considered as a 2-dimensional grid of pixels (or 3-dimensional volumes for color images)
- CNN have been very successful in practical applications
- The name "convolution" refers to an operation that is **similar** to the convolution operation used in physics and engineering.

Overview

- A convolution network usually has at least one convolution layer
- A convolution operation (slightly different from the usual definition) is done by multiplying
- Weights element-wise with a portion of the input
- The same operation with the same weights is repeated over all the input
- The set of weights are usually referred to as the **kernel**
- The result of the convolution is referred to as the **feature map**

Convolution operation

- The convolution of two functions f and g as used in physics and engineering is defined as

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

- The discrete version is written as

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

- For finite domains

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

Convolution in NN

- In neural networks we use a related operation, cross-correlation is an operation involving the input I and a kernel K is defined as

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

- The range of the indices m and n depend on the kernel size.
- The equation above assumes a **stride** of size 1. Later we will deal with strides of different sizes.

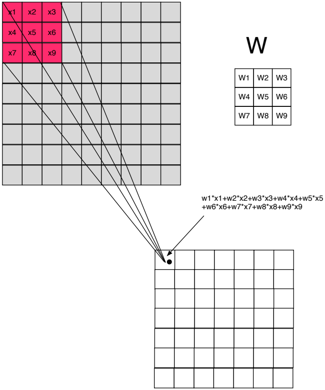
Motivation

- There three basic ideas behind convolution
  1. Sparse interaction
  2. Parameters sharing
  3. Equivariant representation
- The NN that we have dealt with so far use matrix multiplication of the input with the weights
- Each input unit interacts with each output unit.
- In convolution each output unit interacts with a portion(typically small) of the input
- This is done by making the kernel small compared with the input.
- This also leads to a smaller number of parameters.
- Parameters sharing: the same kernel (weights) are used for all the locations in the input.

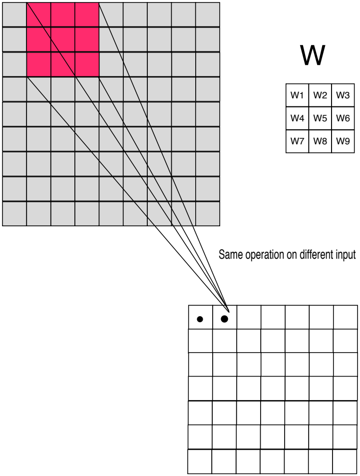
Translation equivariance

- First equivariance is NOT invariance. Let T be the translation operation and f the convolution operation.
- Let X be an input.
- Translation invariance means $f(T(X)) = f(X)$ a property which convolution does NOT possess.
- Translation equivariance means $f(T(X)) = T(f(X))$ a property which convolution does possess.
- What equivariance means is that if a feature is detected by the convolution at position x and reported at output y then when the input is shifted, say by d, then the feature will be detect in the output at $y + d$

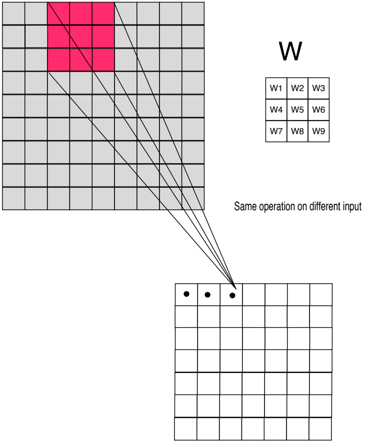
How convolution works for grayscale images



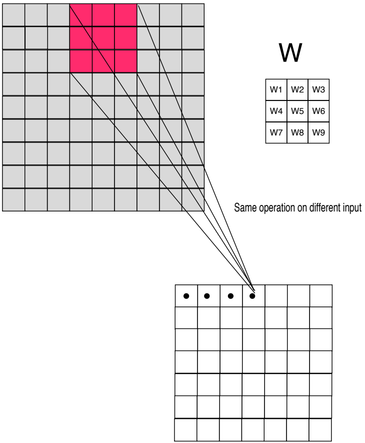
How convolution works for grayscale images



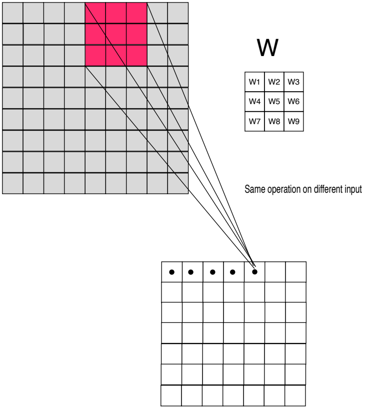
How convolution works for grayscale images



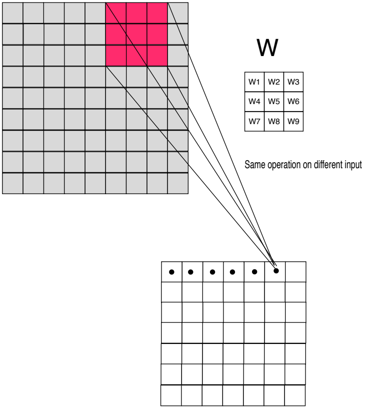
How convolution works for grayscale images



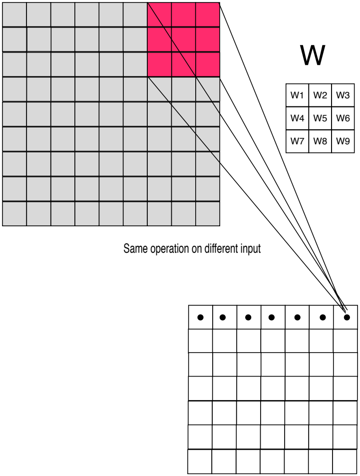
How convolution works for grayscale images



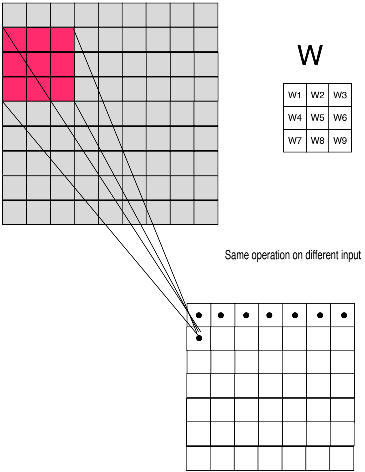
How convolution works for grayscale images



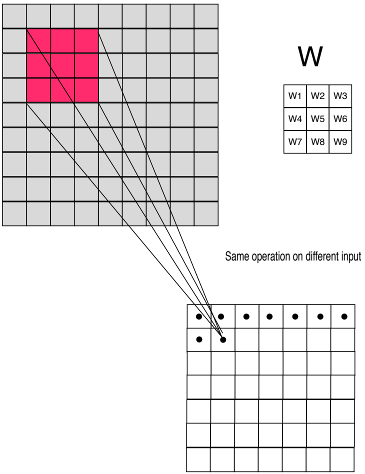
How convolution works for grayscale images



How convolution works for grayscale images



How convolution works for grayscale images



Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

=

6		

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

$=$

6	8	

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

$=$

6	8	2

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

$=$

6	8	2
6		

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

=

6	8	2
6	8	

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

=

6	8	2
6	8	2

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

$=$

6	8	2
6	8	2
6		

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

$=$

6	8	2
6	8	2
6	8	

Example

Input

1	2	4	3
5	6	8	7
9	10	12	11
13	14	16	2

$\odot$

Kernel

1	-2
-3	4

$=$

6	8	2
6	8	2
6	8	-50

What about color images?

- Typically each pixel in a color image is specified as RGB
- So in addition to the space position each pixel has a depth
- In this case the kernel will have depth 3 as well
- The convolution is done as before with the color an extra dimension
- Note that convolution networks are mostly used for images so basically we always have either 2 and 3 dimensions for grayscale and color images
- In our first example we had an input of 9x9 (grayscale image), a kernel with a receptive field of size 3x3
- The output was 3x3

Parameters

- In the second example we had an input of 4x4 (also grayscale), a kernel with a receptive field of size 2x2
- The output was also 3x3. Is the output always 3x3?
- No. If
  1. $W_1 \times H_1 \times D_1$ is the size of the input,
  2. $F \times F$ is the size of the receptive field,
  3. K the number of filters
- Then the output has size $W_2 \times H_2 \times K$ where

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

- Note that the depth of the output is equal to the number of filters.

numpy example

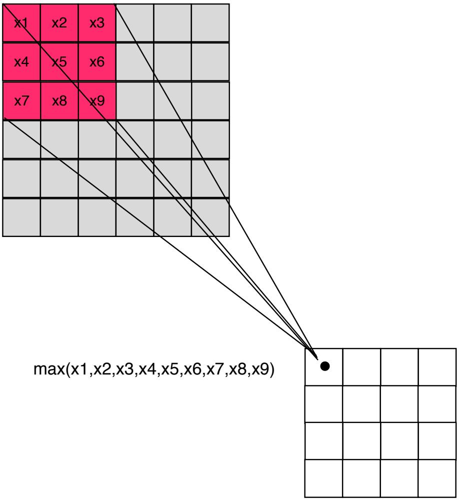
- Suppose the input X has size 4, we use 2 filters (W_0 and W_1) with size 2 and the stride is 1 (no padding). Then the output can be written (not all the values are shown)

```
Z[0,0,0]=np.sum(X[0:2,0:2,0]*W_0)
Z[0,1,0]=np.sum(X[0:2,1:3,0]*W_0)
Z[0,2,0]=np.sum(X[0:2,2:4,0]*W_0)
Z[1,0,0]=np.sum(X[1:3,0:2,0]*W_0)
...
Z[0,0,1]=np.sum(X[0:2,0:2,0]*W_1)
```

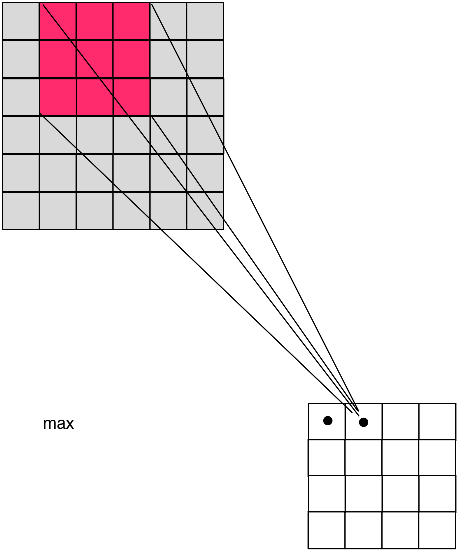
Pooling

- It is common to insert a pooling layer between successive convolution layers.
- The main functions of the pooling layer
  1. reduce the spatial size of the input
  2. reduce the amount of parameters
- Pooling operates independently on every depth-slice of the input and resizes it spatially.
- Usually, the pooling layer uses filters of size 2×2 applied with a stride of 2.
- The pooling is done by using the max operation on a receptive field.

Example pooling



Example pooling



Parameters

- As in the case of convolution if the input is of size $W_1 \times H_1 \times D$ and we choose the receptive field size F and a stride S then the output has dimensions $W_2 \times H_2 \times D_2$ where
  1. $D_2 = D_1$ since the pooling is done per depth slice
  2. $W_2 = \frac{W_1 - F}{S} + 1$
  3. $H_2 = \frac{H_1 - F}{S} + 1$
- It is NOT common to use padding in the pooling layer.
- The most common receptive field size and stride are $F = 2$ and $S = 2$.

Why pooling

- Pooling replace a region in the input with a summary statistic usually the max value when using max pooling.
- This makes the computation almost invariant to translation as can be seen in the example below

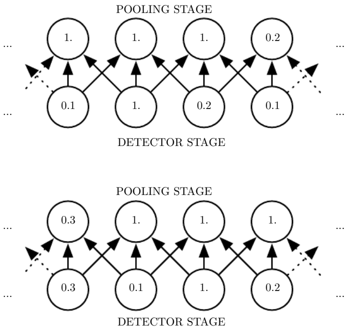


Figure : From Goodfellow et. al.

Architecture

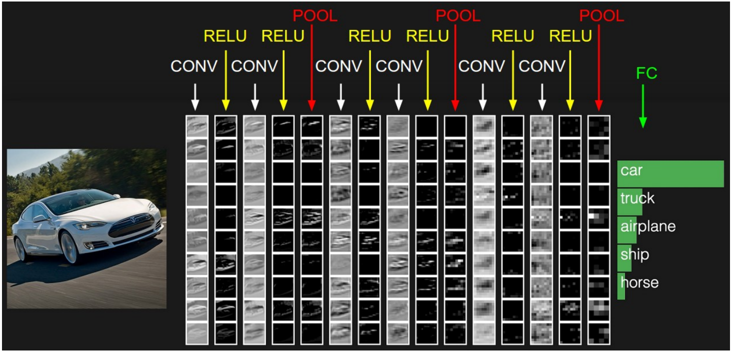
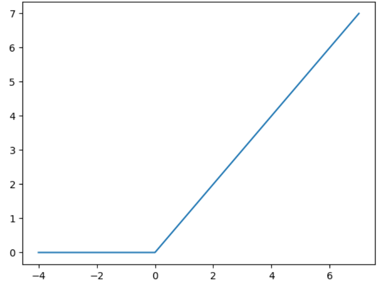


Figure : Andrej Karpathy

ReLU

- The ReLU (Rectified Linear Unit) function is used instead of the sigmoid or the softmax is defined by

$$ReLU(x) = \max(0, x)$$



Why ReLU

- The most important advantage of ReLU is the reduced likelihood of vanishing gradient
- For large values of the input the sigmoid and its derivative go to zero very quickly which makes learning hard
- In the case of the ReLU this does not happen, the derivative is constant
- As we discussed before this is very important especially when the architecture has many layers which makes learning with sigmoid almost impossible.
- Recall that $\sigma' = \sigma(1 - \sigma) \leq 0.25$ for all values