CNN Basics

Shubhra Aich
s.aich.72@gmail.com

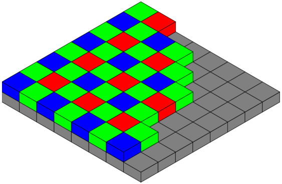
January 17, 2018

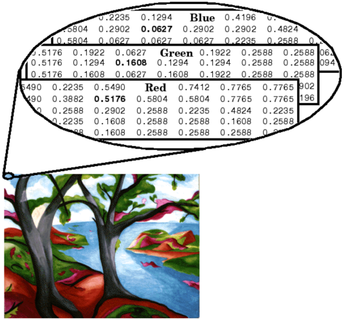
Outline

- RGB image basics
- ANN vs CNN
- Operations in CNN
    - Convolution
    - Pooling
    - Rectification
    - Dropout

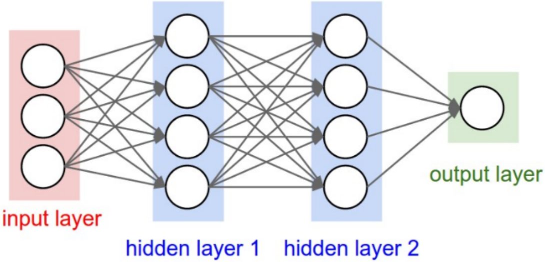
RGB image basics

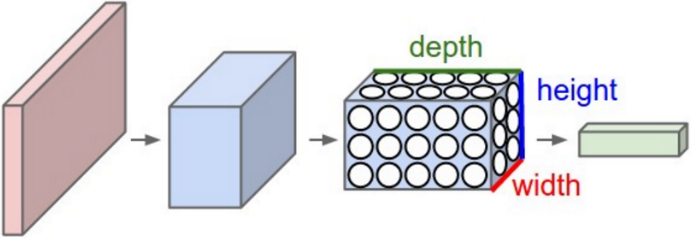
RGB image consists of 3 channels or planes, namely Red, Green and Blue.
We see the combined effect of these channels.





Artifical *vs* Convolutional NN





Operations in CNN

- Convolution
- Rectification
- Pooling
- Dropout

Convolution ...(1)

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	2	2	0	0	0
0	1	0	2	0	1	0
0	1	2	1	1	2	0
0	0	1	1	1	1	0
0	2	2	1	0	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	2	0	1	1	0
0	0	1	2	1	0	0
0	0	0	0	0	2	0
0	0	2	0	1	2	0
0	0	1	2	0	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	0	2	2	0
0	2	1	1	1	2	0
0	2	0	1	0	0	0
0	0	1	2	0	2	0
0	1	2	2	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	-1	-1
-1	1	1
1	1	-1

w0[:,:,1]

-1	-1	0
1	-1	1
1	1	0

w0[:,:,2]

1	-1	1
0	1	0
-1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	0
0	1	-1
-1	1	0

w1[:,:,1]

0	0	-1
-1	0	-1
1	-1	0

w1[:,:,2]

1	1	0
1	-1	-1
1	-1	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	6	2
2	-2	0
7	-5	-3

o[:,:,1]

-9	0	0
-1	3	3
-6	1	6

Convolution ...(2)

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	2	2	0	0	0
0	1	0	2	0	1	0
0	1	2	1	1	2	0
0	0	1	1	1	1	0
0	2	2	1	0	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	2	0	1	1	0
0	0	1	2	1	0	0
0	0	0	0	0	2	0
0	0	2	0	1	2	0
0	0	1	2	0	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	0	2	2	0
0	2	1	1	1	2	0
0	2	0	1	0	0	0
0	0	1	2	0	2	0
0	1	2	2	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	-1	-1
-1	1	1
1	1	-1

w0[:,:,1]

-1	-1	0
1	-1	1
1	1	0

w0[:,:,2]

1	-1	1
0	1	0
-1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	0
0	1	-1
-1	1	0

w1[:,:,1]

0	0	-1
-1	0	-1
1	-1	0

w1[:,:,2]

1	1	0
1	-1	-1
1	-1	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

3	6	2
2	-2	0
7	-5	-3

o[:,:,1]

-9	0	0
-1	3	3
-6	1	6

Convolution ...(3)

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	2	2	0	0	0
0	1	0	2	0	1	0
0	1	2	1	1	2	0
0	0	1	1	1	1	0
0	2	2	1	0	2	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	2	2	0	1	1	0
0	0	1	2	1	0	0
0	0	0	0	0	2	0
0	0	2	0	1	2	0
0	0	1	2	0	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	0	2	2	0
0	2	1	1	1	2	0
0	2	0	1	0	0	0
0	0	1	2	0	2	0
0	1	2	2	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

-1	-1	-1
-1	1	1
1	1	-1

w0[:,:,1]

-1	-1	0
1	-1	1
1	1	0

w0[:,:,2]

1	-1	1
0	1	0
-1	-1	-1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	1	0
0	1	-1
-1	1	0

w1[:,:,1]

0	0	-1
-1	0	-1
1	-1	0

w1[:,:,2]

1	1	0
1	-1	-1
1	-1	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

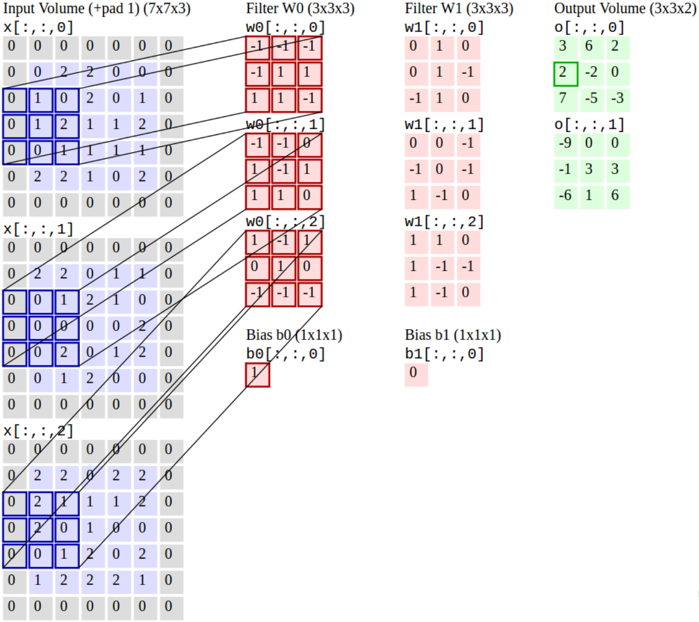
o[:,:,0]

3	6	2
2	-2	0
7	-5	-3

o[:,:,1]

-9	0	0
-1	3	3
-6	1	6

Convolution ...(4)



Convolution ...(5)

Input Volume (+pad 1) (7x7x3)

`x[:,:,0]`

0	0	0	0	0	0	0
0	0	2	2	0	0	0
0	1	0	2	0	1	0
0	1	2	1	1	2	0
0	0	1	1	1	1	0
0	2	2	1	0	2	0
0	0	0	0	0	0	0

`x[:,:,1]`

0	0	0	0	0	0	0
0	2	2	0	1	1	0
0	0	1	2	1	0	0
0	0	0	0	0	2	0
0	0	2	0	1	2	0
0	0	1	2	0	0	0
0	0	0	0	0	0	0

`x[:,:,2]`

0	0	0	0	0	0	0
0	2	2	0	2	2	0
0	2	1	1	1	2	0
0	2	0	1	0	0	0
0	0	1	2	0	2	0
0	1	2	2	2	1	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

`w0[:,:,0]`

-1	-1	-1
-1	1	1
1	1	-1

`w0[:,:,1]`

-1	-1	0
1	-1	1
1	1	0

`w0[:,:,2]`

1	-1	1
0	1	0
-1	-1	-1

Bias b0 (1x1x1)

`b0[:,:,0]`

1

Filter W1 (3x3x3)

`w1[:,:,0]`

0	1	0
0	1	-1
-1	1	0

`w1[:,:,1]`

0	0	-1
-1	0	-1
1	-1	0

`w1[:,:,2]`

1	1	0
1	-1	-1
1	-1	0

Bias b1 (1x1x1)

`b1[:,:,0]`

0

Output Volume (3x3x2)

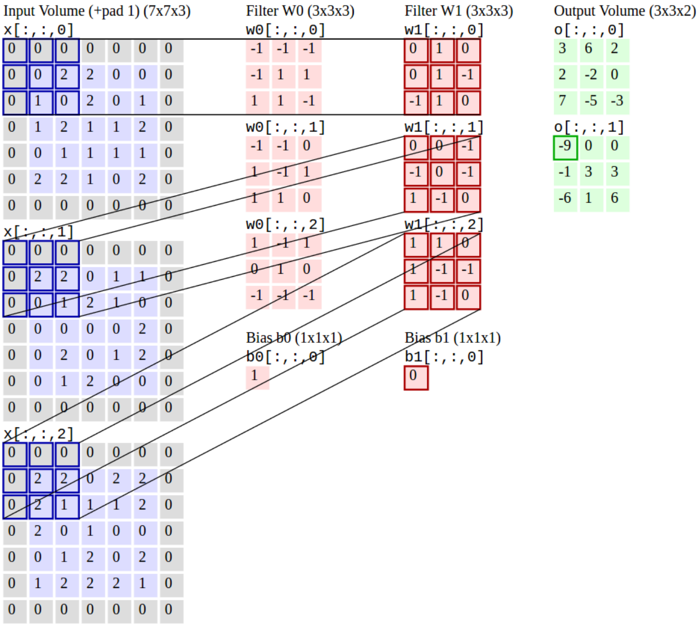
`o[:,:,0]`

3	6	2
2	-2	0
7	-5	-3

`o[:,:,1]`

-9	0	0
-1	3	3
-6	1	6

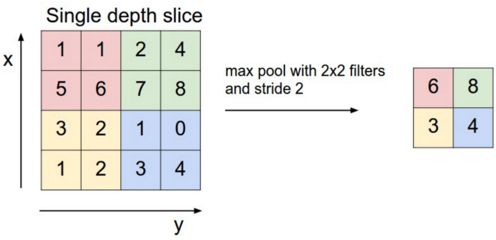
Convolution ...(6)



Equations of Convolution

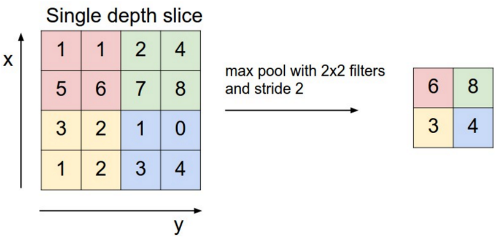
w_i = weights of the filter
x_i = outputs of the previous layer
$\mathrm{z} = \sum_i w_i x_i + b = w.x + b \equiv w * x + b$
$\mathrm{a} = \mathrm{f(z)}$
f = activation/ rectification / squashing function

Pooling



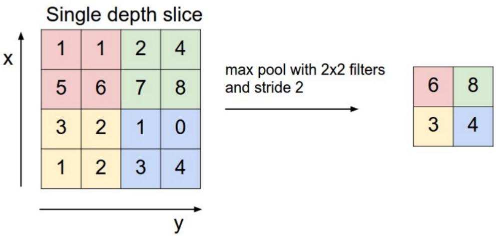
- Max-pooling.
- Average pooling.

Pooling



- Max-pooling.
- Average pooling.
- Min-pooling

Pooling



- Max-pooling.
- Average pooling.
- Min-pooling

Why do we use Convolution and Pooling? ...(1)

CNN combines 3 architectural ideas to deal with shift, scale and distortion.

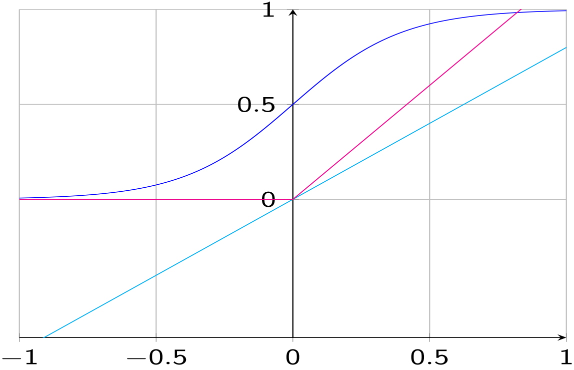
- Local receptive fields/filters
  - The idea of local filters dates back to Hubel-Wiesel's discovery of locally sensitive, orientation selective neurons in the cat's visual cortex.
  - It helps to extract elementary visual features, such as edges, corners, end-points etc.
- Shared weights
  - Distortions or shifts of the input causes the position of salient features to vary.
  - Elementary feature detectors that are useful on one region of the image are likely to be useful across the entire image.
  - This knowledge can be applied by forcing a set of units, whose receptive fields are located at different places on the to be identical.
  - These features are combined by subsequent layers to detect more meaningful features.

Why do we use Convolution and Pooling? ...(2)

- Sub-sampling/Pooling
  - Once a feature is detected, its exact location becomes less important, only approximate position relative to other features is relevant.
  - This precise position is even harmful they vary from image to image.
  - The simplest way to reduce this precision is sub-sampling or pooling.

Rectification

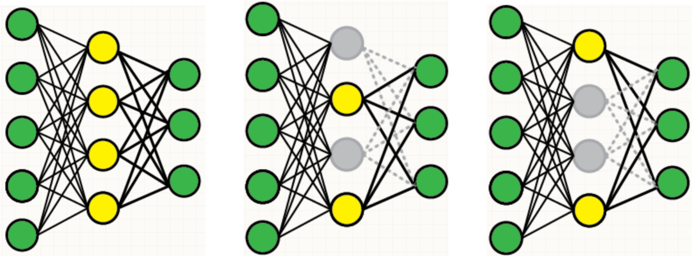
- Linear neuron, $y = wx + b$
- Rectified linear neuron (ReLU), $y = \begin{cases} wx + b, & \text{if} x > 0 \\ 0, & \text{otherwise} \end{cases}$
- Sigmoid/logistic neuron, $y = \frac{1}{1+exp(-\alpha(wx+b))}$



Choice of Recitification Functions

- Backpropagation is the most popular learning algorithm for a network comprising real-valued units.
- This learning algorithm adjusts the weights in each iteration in proportion to the derivative of the final error function w.r.t. the weights.
- To facilitate learning, rectification functions are expected to have more or less smooth derivatives.
- Linear neurons can map only linear input-output relationships.
- Linear neurons are unbounded both below and above.
- Sigmoid activation works well for simple ANN. However, in deeper models like CNN, tiny error-gradients of sigmoid units make learning horrendously slow.

Dropout



- In each iteration of learning, we randomly omit each hidden unit with a prespecified probability.
- Using dropout, for a network comprising single hidden layer with H hidden units, in each iteration, we randomly sample from 2^H different architectures.
- All architectures share weights.
- Dropout prevents overfitting.
- In test time, the full network is used with parameter values halved.