Building Deep Learning applications with Keras: Deep Feedforward Networks and Convolutional Neural Networks

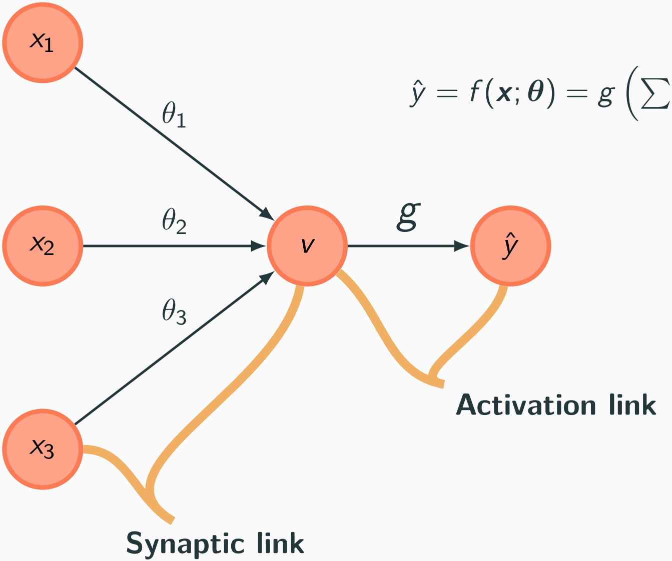
Felipe Salvatore
https://felipessalvatore.github.io/

Lucas Moura
http://lmoura.me/

June 15, 2018

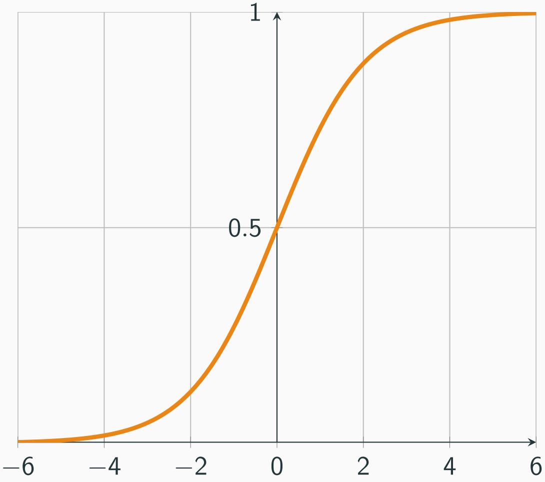
Neural Networks

Perceptron



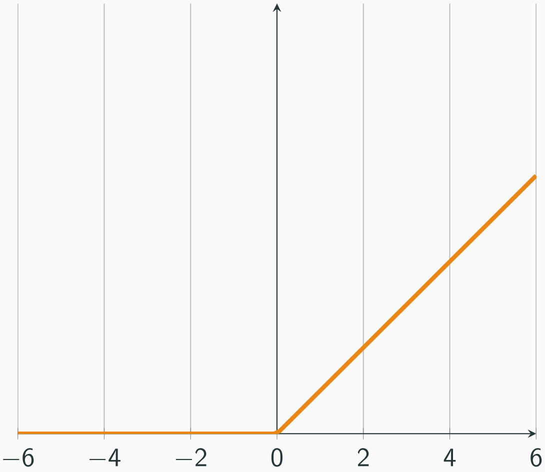
$$\hat{y} = f(\boldsymbol{x};\boldsymbol{\theta}) = g\left(\sum_{i=1}^{3} \theta_i x_i\right)$$

Sigmoid function



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

ReLU: Rectified Linear Units



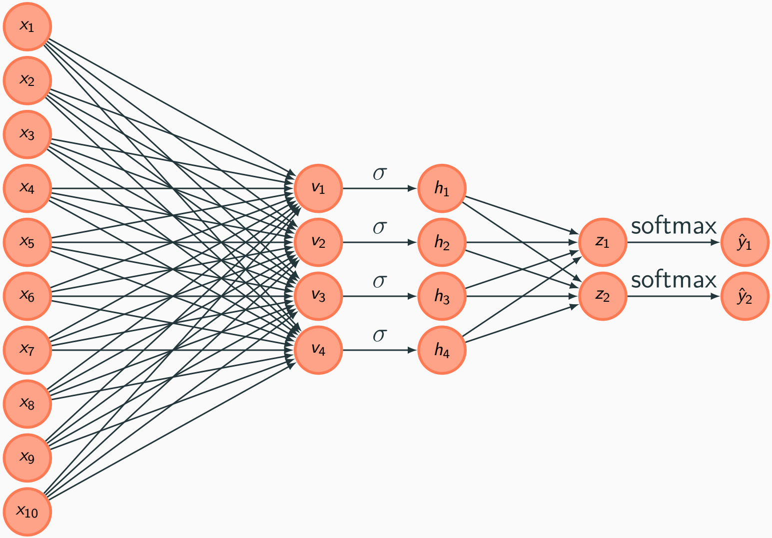
$$g(x) = max\{0, x\}$$

Softmax function

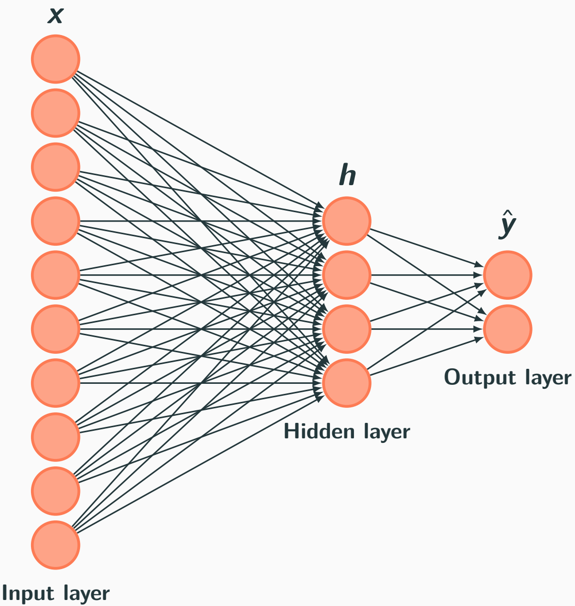
$$\begin{bmatrix} 3.82 \\ 5.35 \\ 1.44 \\ -1.26 \\ 2.71 \\ 1.98 \end{bmatrix} \xrightarrow{\text{softmax}} \begin{bmatrix} 0.16115195 \\ 0.74422819 \\ 0.01491471 \\ 0.00100235 \\ 0.05310907 \\ 0.02559374 \end{bmatrix}$$

$$softmax(x) = \frac{e^x}{\sum e^x}$$

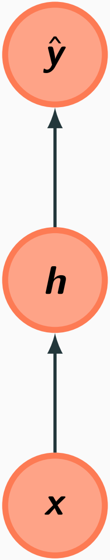
Neural network, classical representation



Neural network, classical representation



Neural network, computational graph



Deep Feedforward Networks

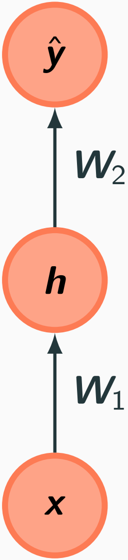
Deep Feedforward Networks

Deep Feedforward Networks (also called feedforward neural networks, multilayer perceptrons or just neural networks) is a family of parametric models $\hat{\boldsymbol{y}} = f(\boldsymbol{x}; \boldsymbol{\theta})$.

These models can be described as a function composition, for example:

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = f^{(2)}(f^{(1)}(\boldsymbol{x}; \boldsymbol{\theta}); \boldsymbol{\theta})$$

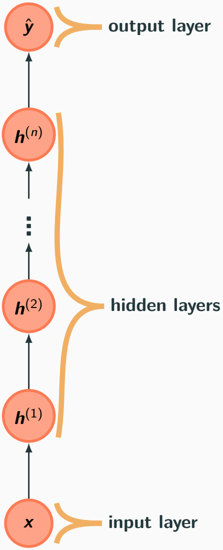
Neural network, example



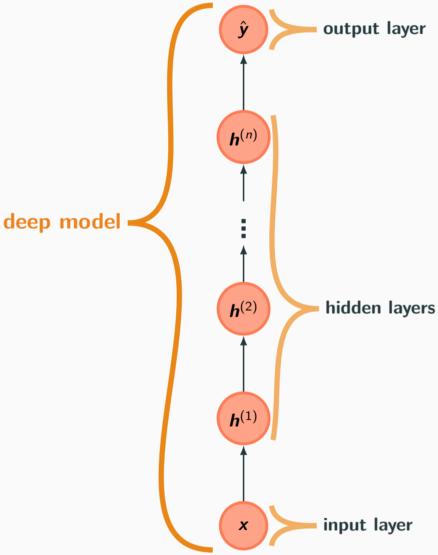
$$\hat{y} = f^{(2)}(f^{(1)}(\boldsymbol{x}; \boldsymbol{W}_1); \boldsymbol{W}_2)$$

$$\hat{y} = softmax(\boldsymbol{W}_2(\sigma(\boldsymbol{W}_1 \boldsymbol{x})))$$

Neural network, deep network



Neural network, deep network

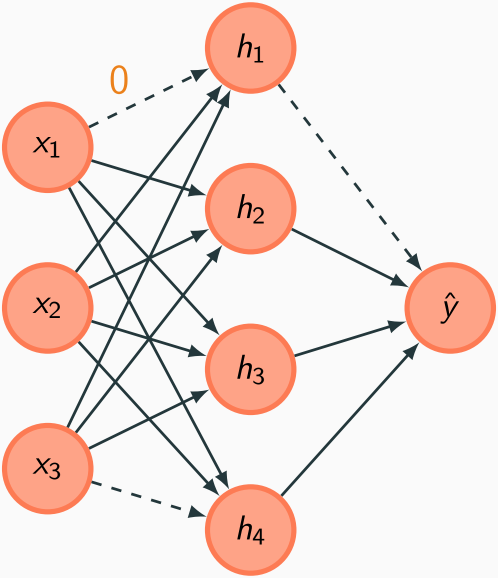


Regularization

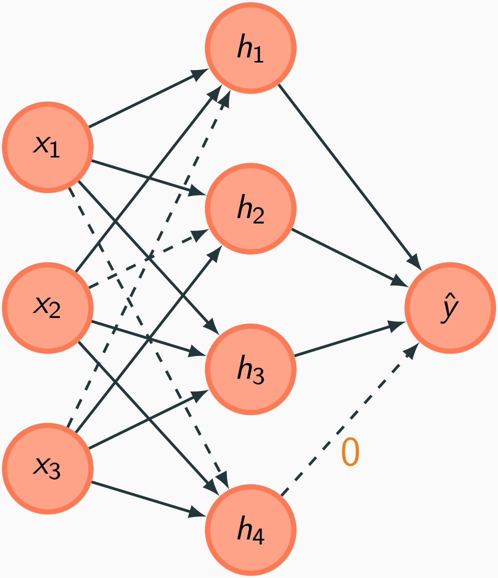
The strategies designed to reduce the model's generalization error (but not its training error) are called Regularization. Some popular procedures inside the deep learning community are:

- L^2 Parameter Regularization
- Early Stopping
- Dropout

Dropout

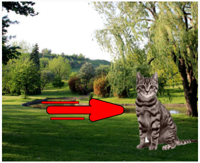


Dropout



Convolutional Neural Networks (CNN)

Intuition



- Regarding image classification, the human eye is translational invariant.
- In computer vision, convolution is the process of applying a filter (kernel) to an image.
- This operation is easy to implement: by using a Toeplitz matrix, convolution can be view as matrix multiplication.

Applying a filter (from[1])

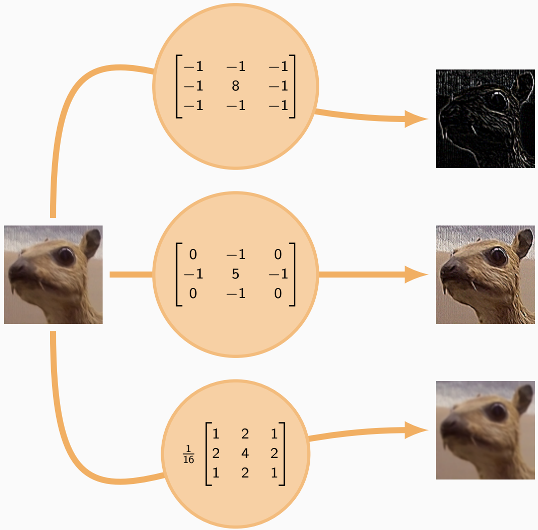


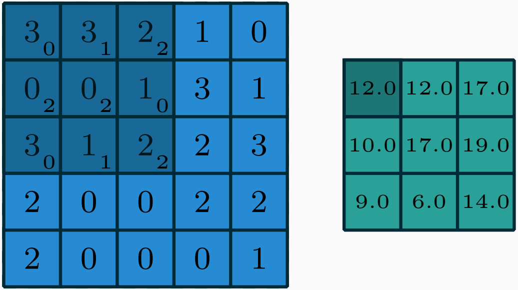
Image example (from [2])

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

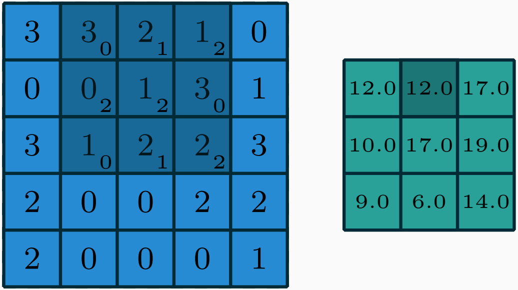
Filter example (from [2])

0	1	2
2	2	0
0	1	2

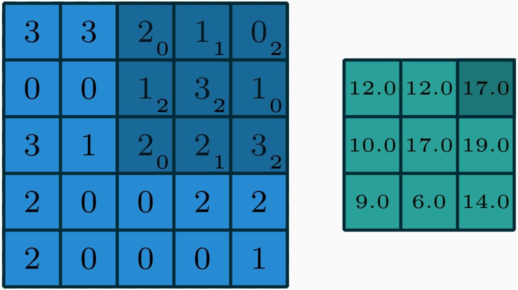
Convolution (from [2])

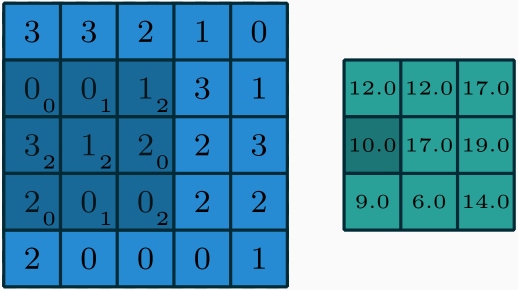


Convolution (from [2])



Convolution (from [2])




3 3 2 1 0
0 0 0 1 1 2 3 1
3 2 1 2 2 0 2 3
2 0 0 1 0 2 2 2
2 0 0 0 1
12.0 12.0 17.0
10.0 17.0 19.0
9.0 6.0 14.0

Convolution (from [2])

3	3	2	1	0
0	0_0	1_1	3_2	1
3	1_2	2_2	2_0	3
2	0_0	0_1	2_2	2
2	0	0	0	1

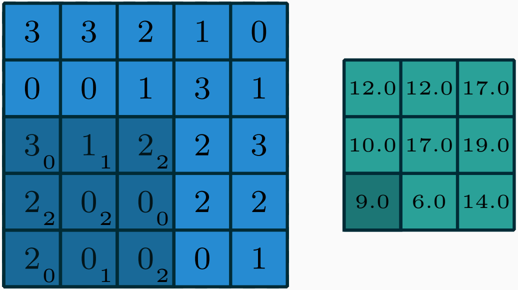
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Convolution (from [2])

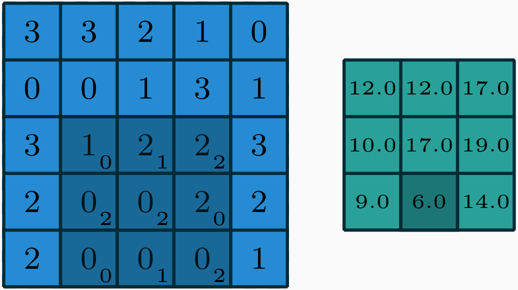
3	3	2	1	0
0	0	1_0	3_1	1_2
3	1	2_2	2_2	3_0
2	0	0_0	2_1	2_2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

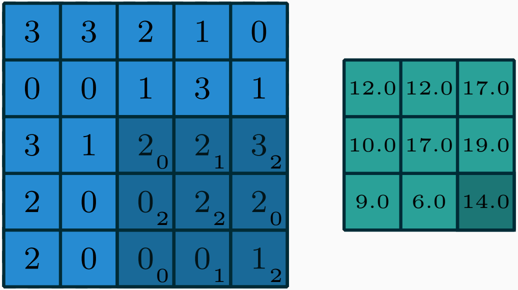
Convolution (from [2])



Convolution (from [2])



Convolution (from [2])



Feature map (from [2])

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

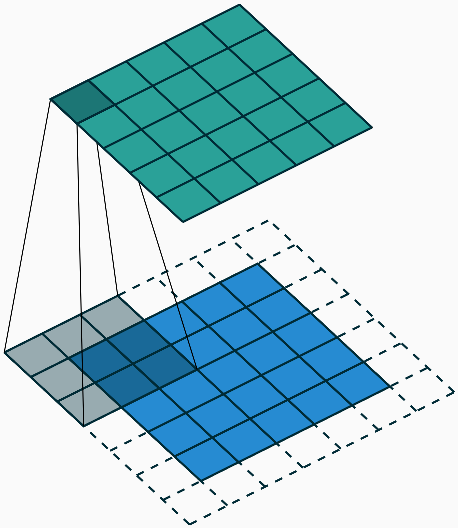
Feature map

The size of the feature map is given by the equation:

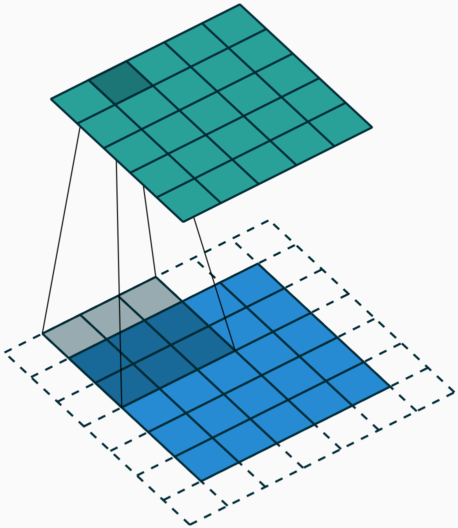
$$o = (i - k) + 1$$

where $o \times o$ is the feature map size, $i \times i$ is the input image size,$k \times k$ is the input kernel size, **stride** $= 1$ and **padding** was not used.

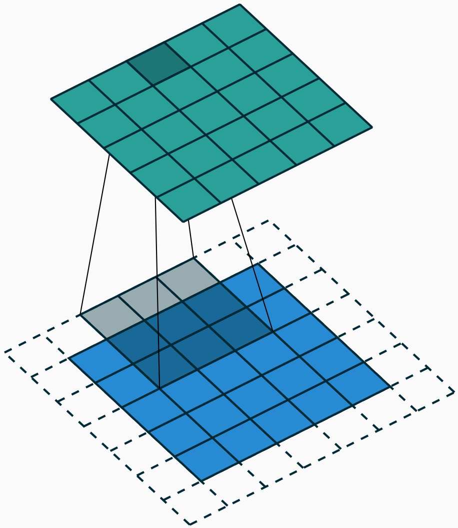
Same Padding (from [2])



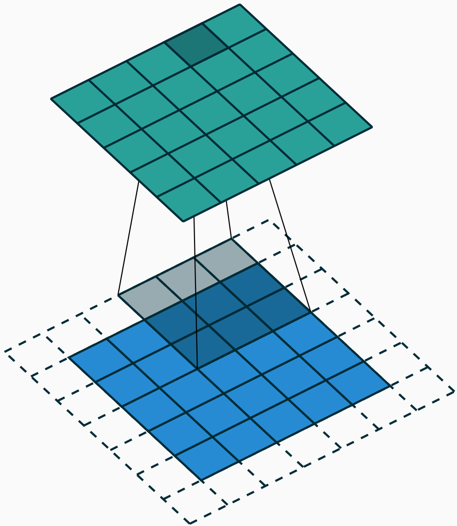
Same Padding (from [2])



Same Padding (from [2])



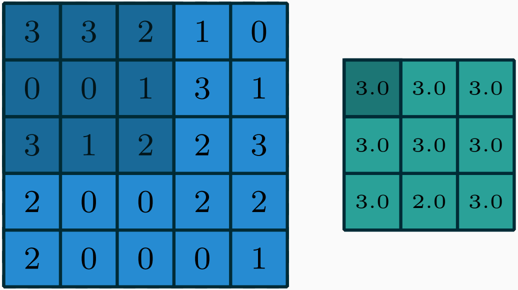
Same Padding (from [2])



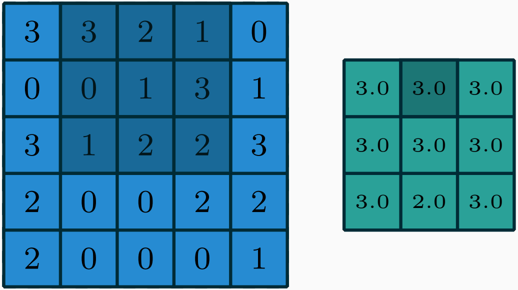
Pooling

- We add one **pooling** layer between convolution layers.
- Using pooling layers we can progressively decrease the image size.

Max Pooling (from [2])



Max Pooling (from [2])



Max Pooling (from [2])

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

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Max Pooling (from [2])

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

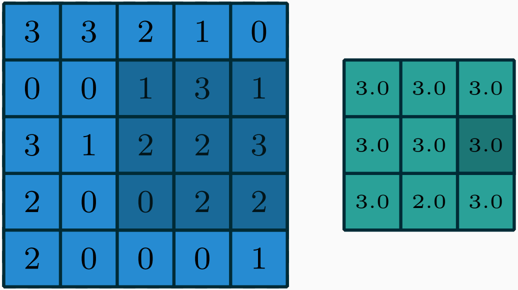
3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Max Pooling (from [2])

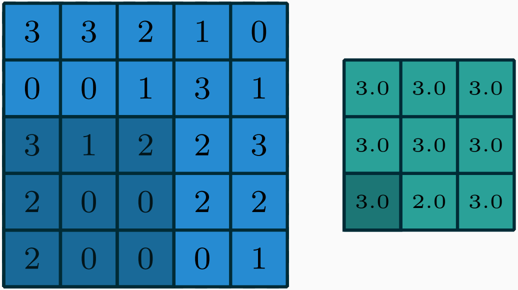
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Max Pooling (from [2])



Max Pooling (from [2])

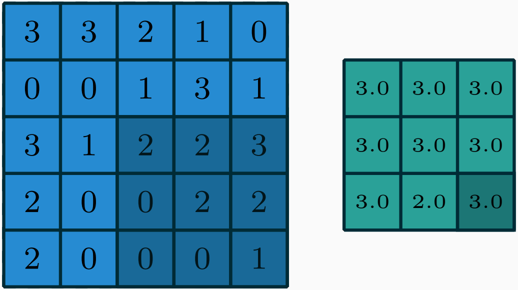


Max Pooling (from [2])

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

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

Max Pooling (from [2])



CNN architecture

$$\text{CONV} \rightarrow \text{POOL} \longrightarrow \text{ReLU} \longrightarrow (...) \longrightarrow \text{FC}$$

- Convolucional layers: learnable kernels
- Pooling: reduces image size
- Activation function: similar to DFN (ReLU, sigmoid, etc.)
- Fully-Connected: DFN

References I

Kernel (image processing).
https://en.wikipedia.org/wiki/Kernel_(image_processing).

V. Dumoulin and F. Visin.
A guide to convolution arithmetic for deep learning, 2016.

I. Goodfellow, Y. Bengio, and A. Courville.
***Deep Learning*.**
MIT Press, 2017.