Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Week 11/12: Deep Learning

CS7CS4/CS4404 Machine Learning
2018-11-21

Dr Joeran Beel
Assistant Professor in Intelligent Systems
Department of Computer Science and Statistics
Trinity College Dublin, Ireland

Dr Douglas Leith
Professor in Computer Systems
Department of Computer Science and Statistics
Trinity College Dublin, Ireland

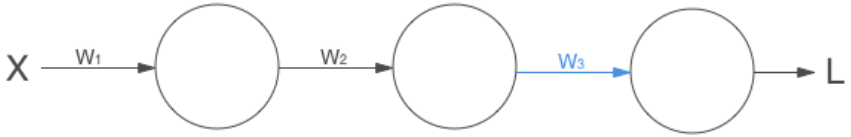
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Challenges in Multi-Layer Networks

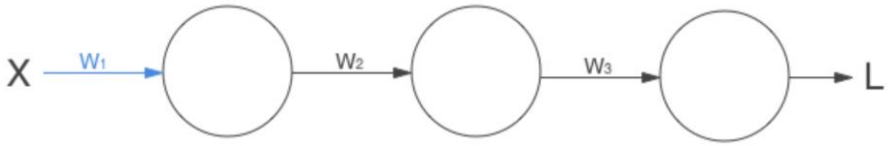
Vanishing/Exploding Gradient Descent

- **The more layers, the more multiplications in backpropagation, the smaller the gradient, the smaller the weight adjustments**
- **Or, e.g. in recurrent NN, the opposite: gradient "explodes"**
- **In general: Unstable gradient, i.e. different layers learn at different speeds**

→ Finding ideal weights for each layer (especially the early ones) takes very long



$$\frac{\partial Loss}{\partial W_3} = \frac{\partial Loss}{\partial f(z_3)} \cdot \frac{\partial f(z_3)}{\partial W_3} = \frac{\partial Loss}{\partial f(z_3)} \cdot f'(z_3) \cdot W_3$$



$$\frac{\partial Loss}{\partial W_1} = \frac{\partial Loss}{\partial f(z_3)} \cdot \frac{\partial f(z_3)}{\partial f(z_2)} \cdot \frac{\partial f(z_2)}{\partial f(z1)} \cdot \frac{\partial f(z_1)}{\partial W_1}$$

$$= \frac{\partial Loss}{\partial f(z_3)} \cdot f'(z_3) \cdot W_3 \cdot f'(z_2) \cdot W_2 \cdot f'(z_1) \cdot W_1$$

http://harinisuresh.com/2016/10/09/lstms/

Example (1)

Slides from one of the previous lectures

Adjust Weights in Output Layer

Click to add text

$$\frac{\partial E_{total}}{\partial w_5} = \left[\frac{\partial E_{total}}{\partial o_{on1}}\right]\frac{\partial o_{on1}}{\partial z_{on1}}\frac{\partial z_{on1}}{\partial w_5} = \left[\frac{\partial E_1}{\partial o_{on1}} + \frac{\partial E_2}{\partial o_{on1}}\right]\frac{\partial o_{on1}}{\partial z_{on1}}\frac{\partial z_{on1}}{\partial w_5}$$

$$E_{total} = E_1 + E_2 = \left[\frac{1}{2}(y_1 - o_{on1})^2\right] + \left[\frac{1}{2}(y_2 - o_{on2})^2\right]$$

$$\frac{\partial E_{total}}{\partial o_{on1}} = -(y_1 - o_{on1}) + 0 = 0.741$$

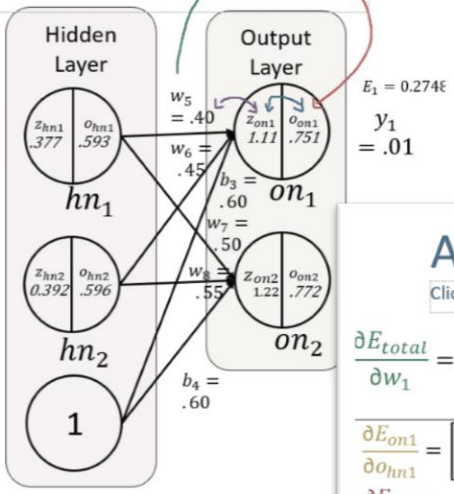
$$o_{on1} = \sigma(z_{on1}) = \frac{1}{1 + e^{-z_{on1}}}$$

$$\frac{\partial o_{on1}}{\partial z_{on1}} = o_{on1}(1 - o_{on1}) = 0.186$$

$$z_{on1} = [w_5 * o_{hn1}] + [w_6 * o_{hn2}] + [b_3]$$

$$\frac{\partial z_{on1}}{\partial w_5} = \frac{\partial (w_5 * o_{hn1})}{\partial w_5} + [0] + [0] = o_{hn1} = 0.593$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.741 * 0.186 * 0.593 = 0.082$$



Adjust Weights in Hidden Layer

Click to add text

$$\frac{\partial E_{total}}{\partial w_1} = \left[\frac{\partial E_{total}}{\partial o_{hn1}}\right]\frac{\partial o_{hn1}}{\partial z_{hn1}}\frac{\partial z_{hn1}}{\partial w_1} = \left[\frac{\partial E_{on1}}{\partial o_{hn1}} + \frac{\partial E_{on2}}{\partial o_{hn1}}\right]\frac{\partial o_{hn1}}{\partial z_{hn1}}\frac{\partial z_{hn1}}{\partial w_1}$$

$$\frac{\partial E_{on1}}{\partial o_{hn1}} = \left[\frac{\partial E_{on1}}{\partial z_{hn1}}\right] * \left[\frac{\partial z_{on1}}{\partial o_{hn1}}\right] = \left[\frac{\partial E_{on1}}{\partial o_{on1}} * \frac{\partial o_{on1}}{\partial z_{on1}}\right] * [w_5] = [0.741 * 0.186] * [0.4] = 0.055$$

$$\frac{\partial E_{on2}}{\partial o_{hn1}} = -0.019$$

$$\frac{\partial E_{total}}{\partial o_{on1}} = 0.055 + (-0.019) = 0.036$$

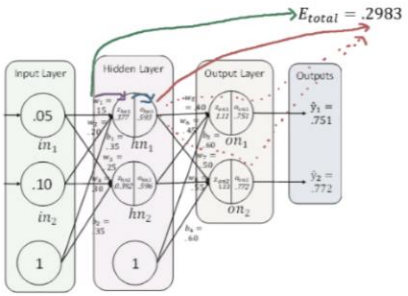
$$o_{on1} = \sigma(z_{on1}) = \frac{1}{1 + e^{-z_{on1}}}$$

$$\frac{\partial o_{on1}}{\partial z_{on1}} = o_{hn1}(1 - o_{hn1}) = 0.241$$

$$z_{on1} = [w_1 * o_{in1}] + [w_2 * o_{in2}] + [b_1]$$

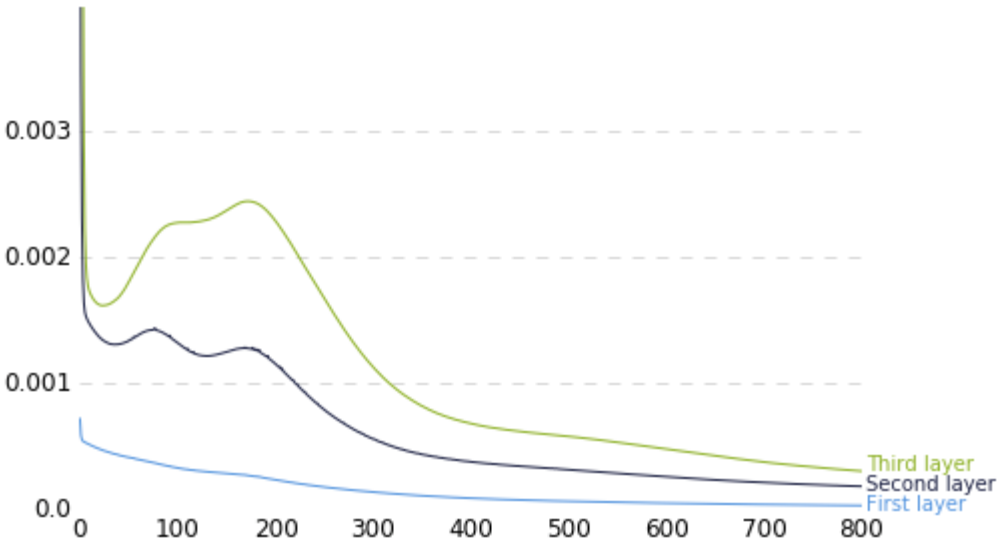
$$\frac{\partial z_{on1}}{\partial w_1} = [1 * o_{in1}] + [0] + [0] = 0.05$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036 * 0.241 * 0.05 = 0.00043$$



Example (2)

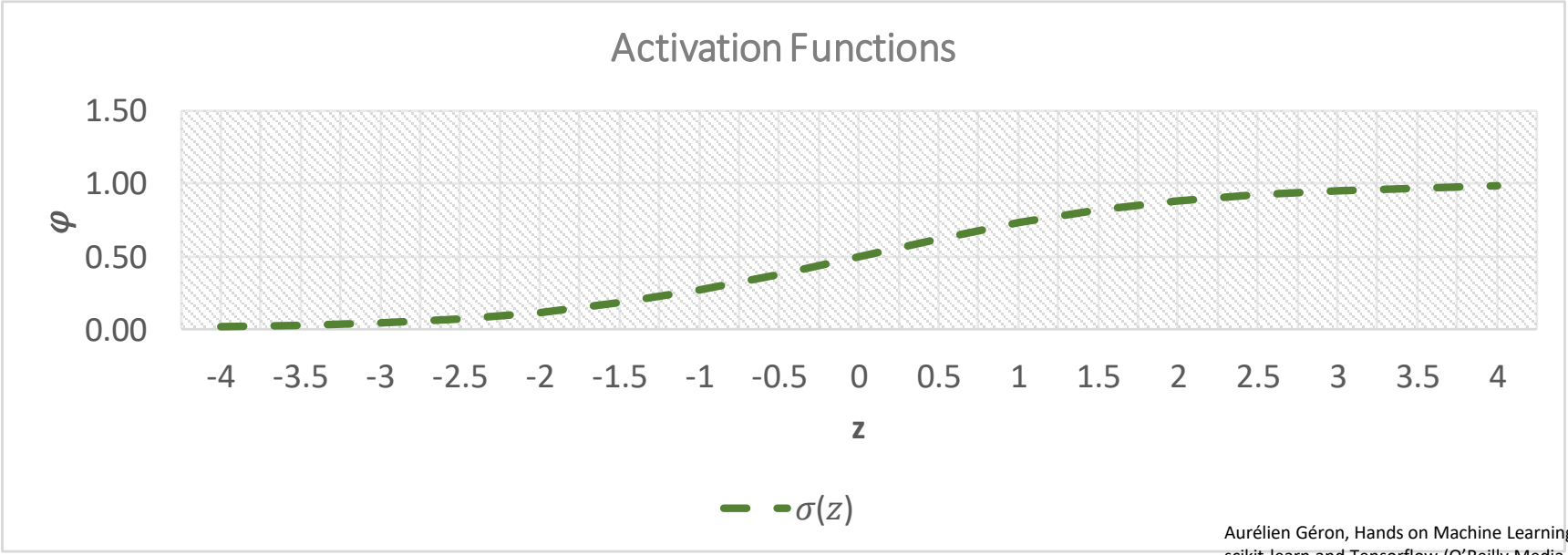
- **MNIST dataset**
- **Three layers**
- **Change of gradient by number of iterations**
- **Much higher gradient for third layer than for first layer → much more changes in third layer's weights**



http://harinisuresh.com/2016/10/09/lstms/

Reasons for Vanishing/Exploding Gradient

- **Suboptimal Activation Functions**
  - Sigmoid activation function saturates quickly, and gradient is close to 0; mean is 0.5
- **Suboptimal Weight Initialisation**
  - Random weight initialisation between 0 and 1 with normal distribution → Variance of a layer's Outputs > Variance of its Inputs → variances increases with each layer.
  - Better: Equal variance for Input and Output; Equal variance of gradients



Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Xavier / He Weight initialization

- **You will find many variations of these formulas …**

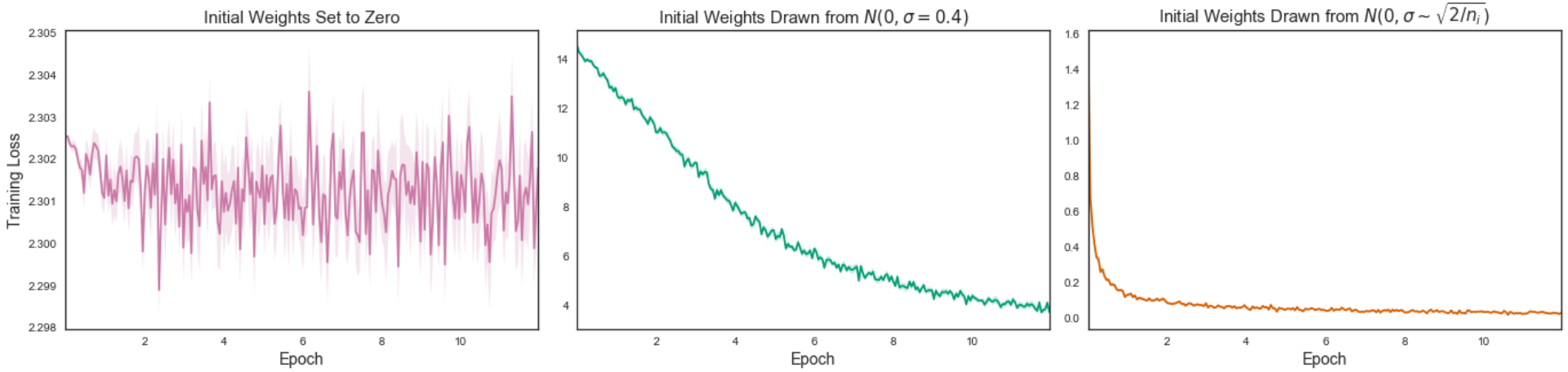
φ	**Uniform distribution [-r, r]**	**Normal Distribution** ($\mu = 0$)
Sigmoid	$r = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = \sqrt{\frac{2}{n_{inputs} + n_{outputs}}}$
tanh	$r = \sqrt[4]{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = \sqrt[4]{\frac{2}{n_{inputs} + n_{outputs}}}$
ReLU (and variants)	$r = \sqrt[\sqrt{2}]{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = \sqrt[\sqrt{2}]{\frac{2}{n_{inputs} + n_{outputs}}}$

… and little empirical guidance about which one to choose

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Example

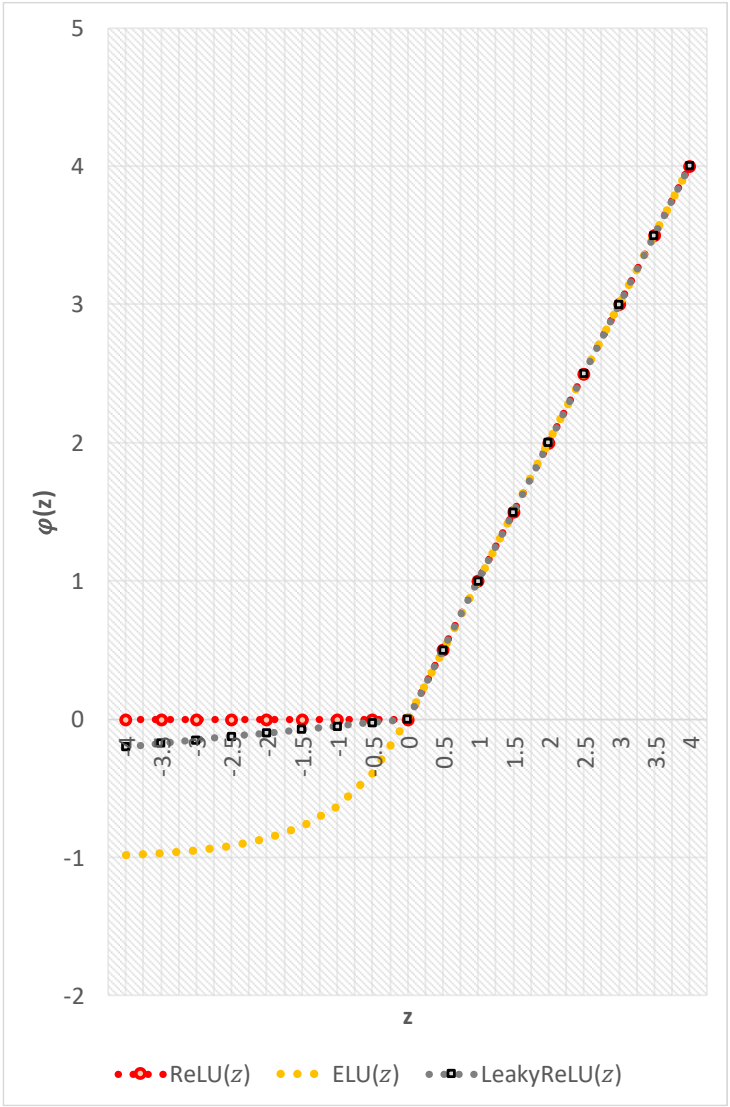
- **MNIST dataset**
- **12 epochs**
- **128 images per epoch**
- **More details and experiments: https://intoli.com/blog/neural-network-initialization/**



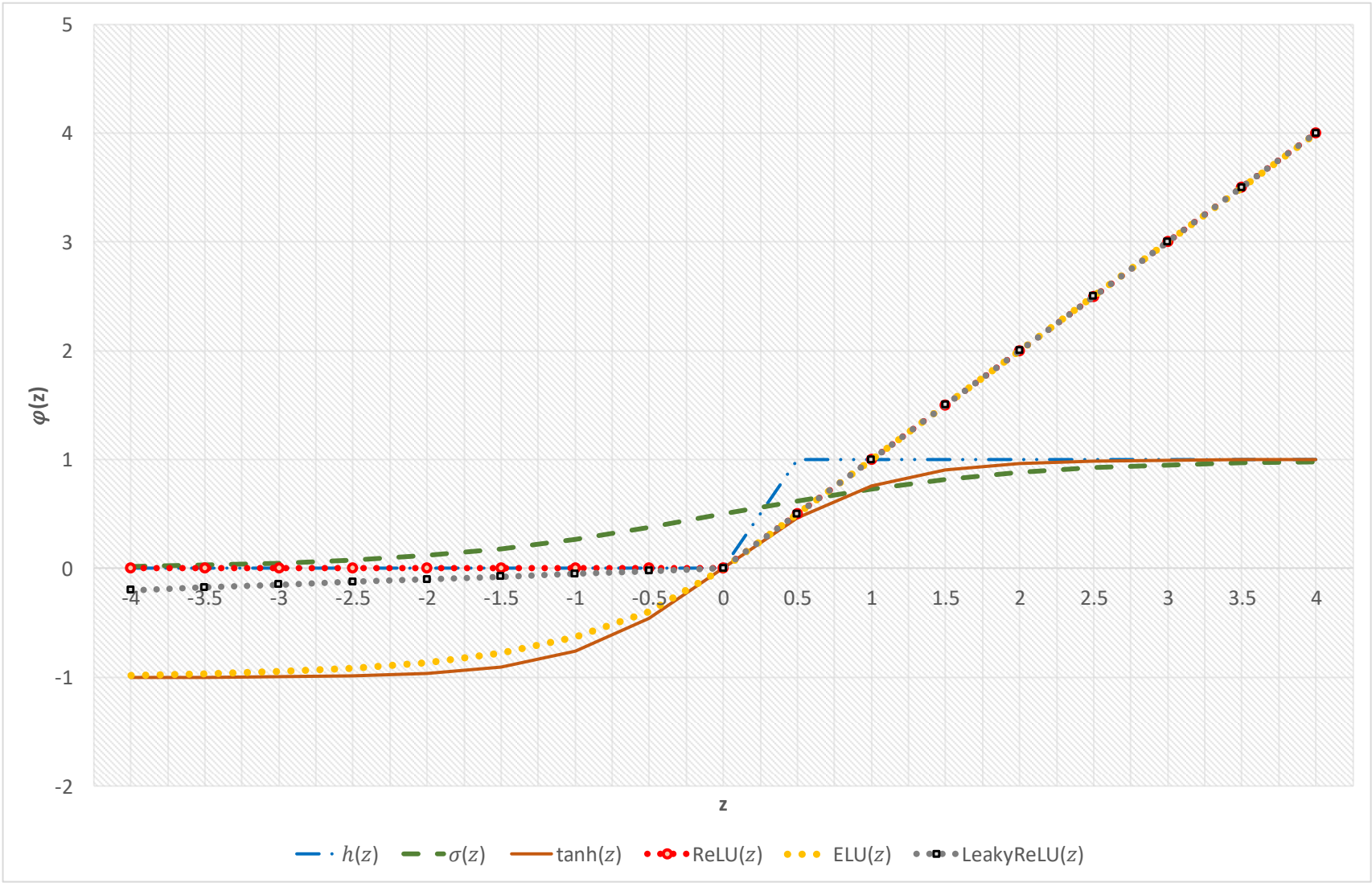
https://intoli.com/blog/neural-network-initialization/

Variations of the Rectified Linear Unit Function

- **The "Dying Neuron" Problem**
  - Once $z < 0$, neuron outputs 0
  - Unlikely to ever „get back to life" (gradient is 0)
- **Solution**
  - $\text{LeakyReLU}(z) = \max(\alpha z, z)$ $\alpha = 0.01$
  - Exponential Linear Unit $\text{ELU}(z) = \begin{cases} \alpha(e^z - 1) & z < 0 \\ z & z \geq 0 \end{cases}$ $\alpha = 1$



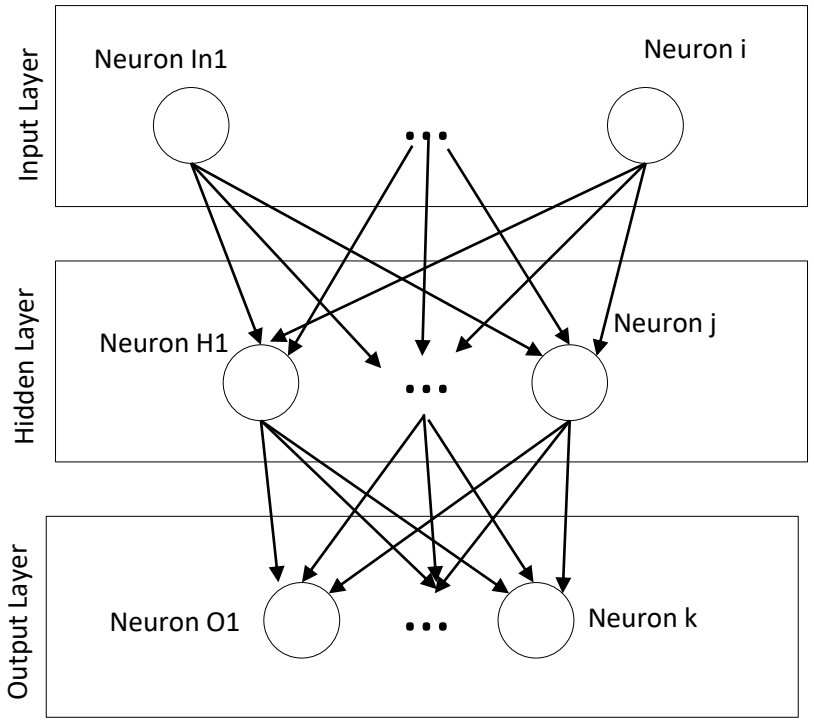
All Activation Functions



Too many Inputs, Weights, …

- **Example**
  - Input: 1 MegaPixel Image (1,000 x 1,000 Pixels)
  - 1 Hidden Layer with 1,000 Neurons
  - 1 Output Layer with 10 Neurons

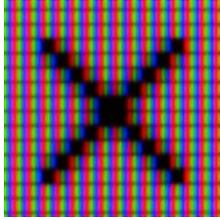
→ How many weights would this Neural Network have?



https://www.tcd.ie/business/assets/img/video-slider/QS-rankings.jpg

https://farm6.static.flickr.com/5634/2834820049_2997e73794_b.jpg

RGB Image



Too many Inputs, Weights, …

- **Fully Connected (Deep) Neural Networks would require too many connections**
- **Example**

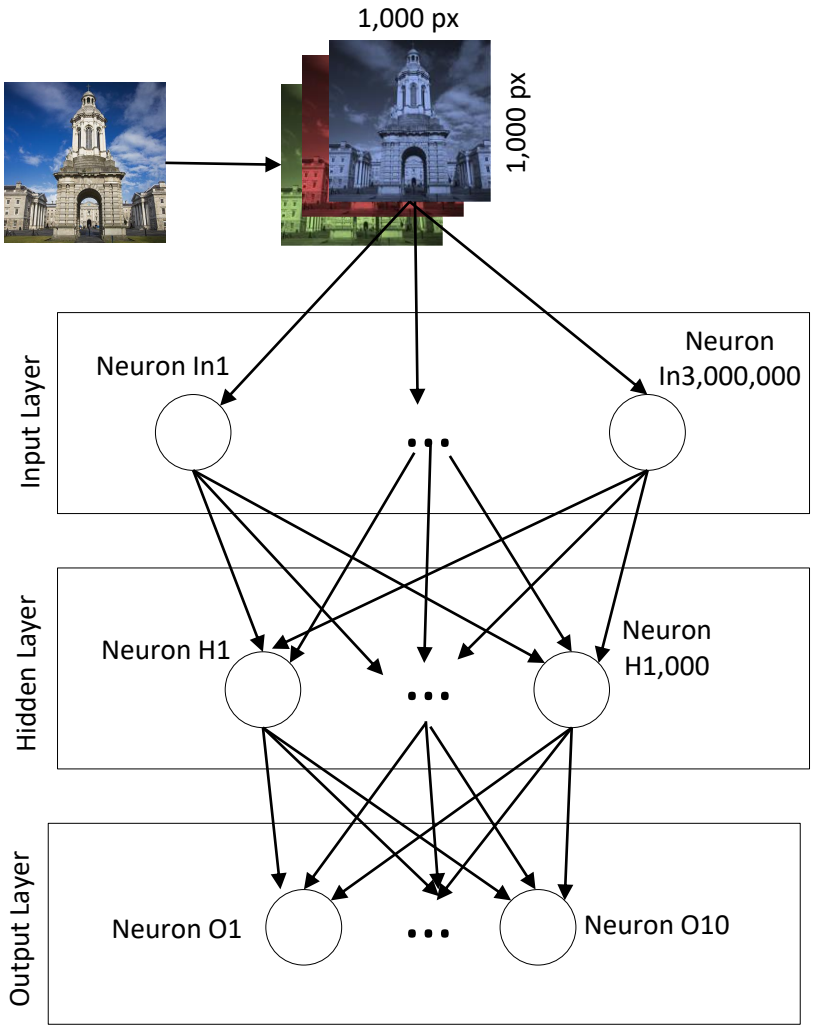
– 1 MegaPixel Image (1,000 x 1,000 Pixels)
→ 1,000,000 Pixels, each with 3 colours
→ 3,000,000 Input Neurons

– 1 Hidden Layer with 1,000 Neurons

– 1 Output Layer with 10 Neurons

→ (3,000,000 * 1,000) + (1,000 * 10) ≈ 3 Billion connections/weights with just a small image and one hidden layer

- **Solution: New architectures that are not fully connected (e.g. Convolutional NN)**



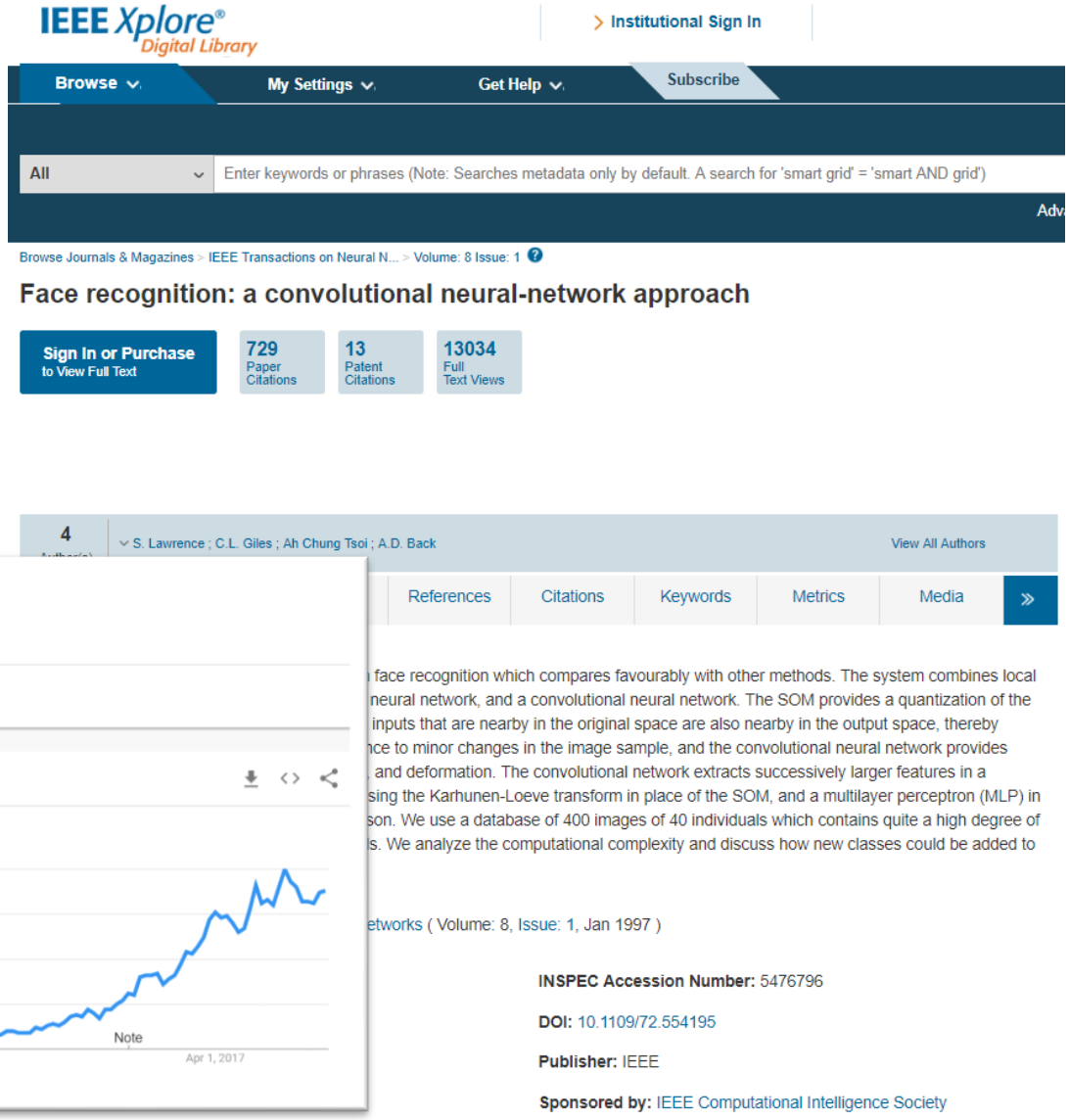
https://www.tcd.ie/business/assets/img/video-slider/QS-rankings.jpg

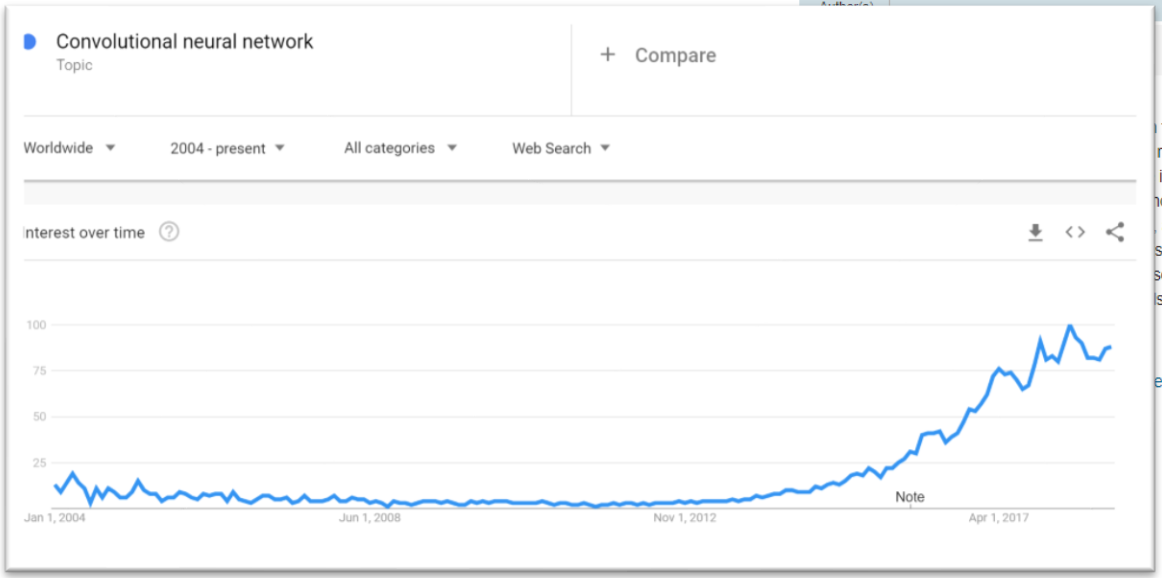
Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Convolutional Neural Networks (CNN)

Deep Learning is not new (nor is Face Recognition)

- **Just one example https://ieeexplore.ieee.org/abstract/document/554195/**

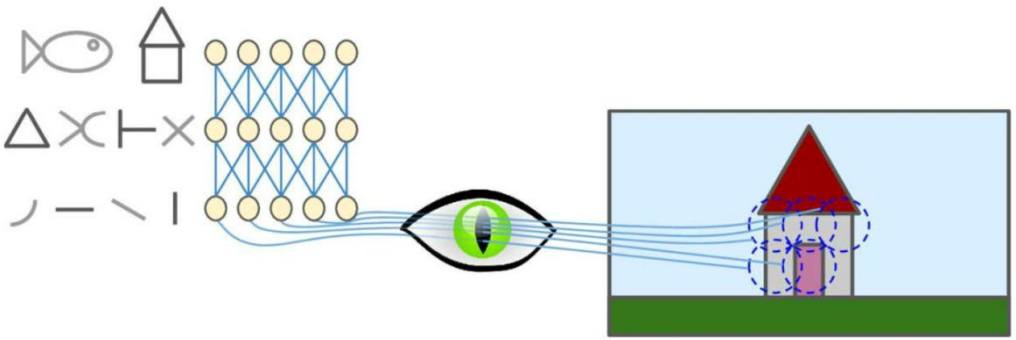


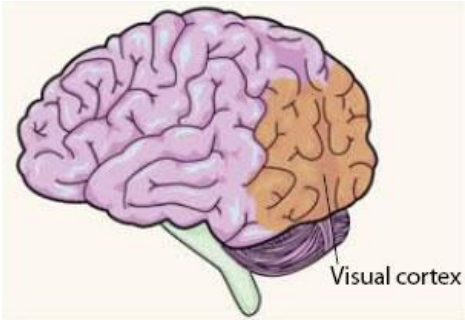


https://trends.google.de/trends/explore?date=all&q=%2Fm%2F0x2dbhq

History & Intuition

- **Introduced in 1989 (LeCun et al., Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1(4):541-551)**
- **Excellent performance for image recognition (and similar tasks such as voice recognition and natural language processing)**
- **Inspired by the human brain's visual cortex**
  - Each neuron reacts only to a stimuli in a limited region
  - Regions may overlap





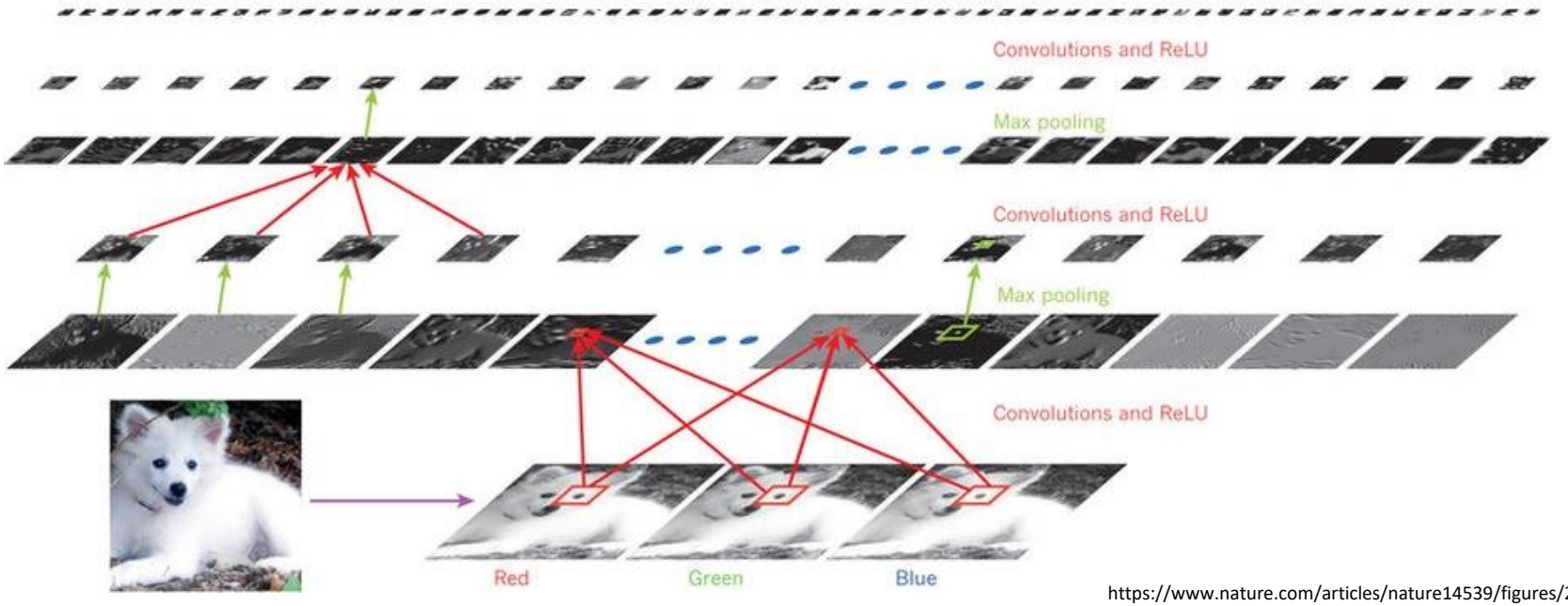
Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep learning (MIT press, 2016).

http://ilearn.careerforce.org.nz/pluginfile.php/452/mod_book/chapter/142/visual_cortex.jpg

CNN Architecture (1)

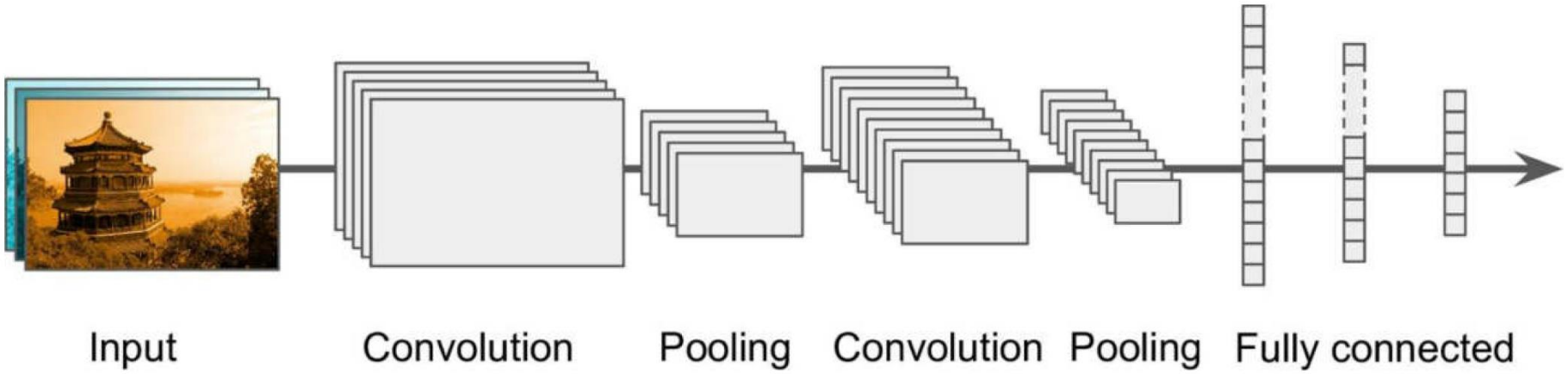
- **Input Data: Multiple Arrays**
  - 1D for sequences
  - 2D images (+1D if colour)
  - 3D videos (+1D if colour)
- **Intuition**
  - Features→Edges→Motives→Parts →Objects
  - Features→Sounds→Phones→Pho nemes→Syllables→Words→Sente nces



https://www.nature.com/articles/nature14539/figures/2

CNN Architecture (2)

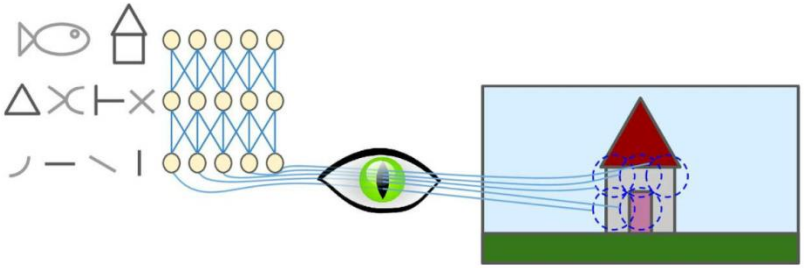
- **Key Concepts**
  - Local Connections / Sparse Interactions
  - Shared Parameters
  - Pooling



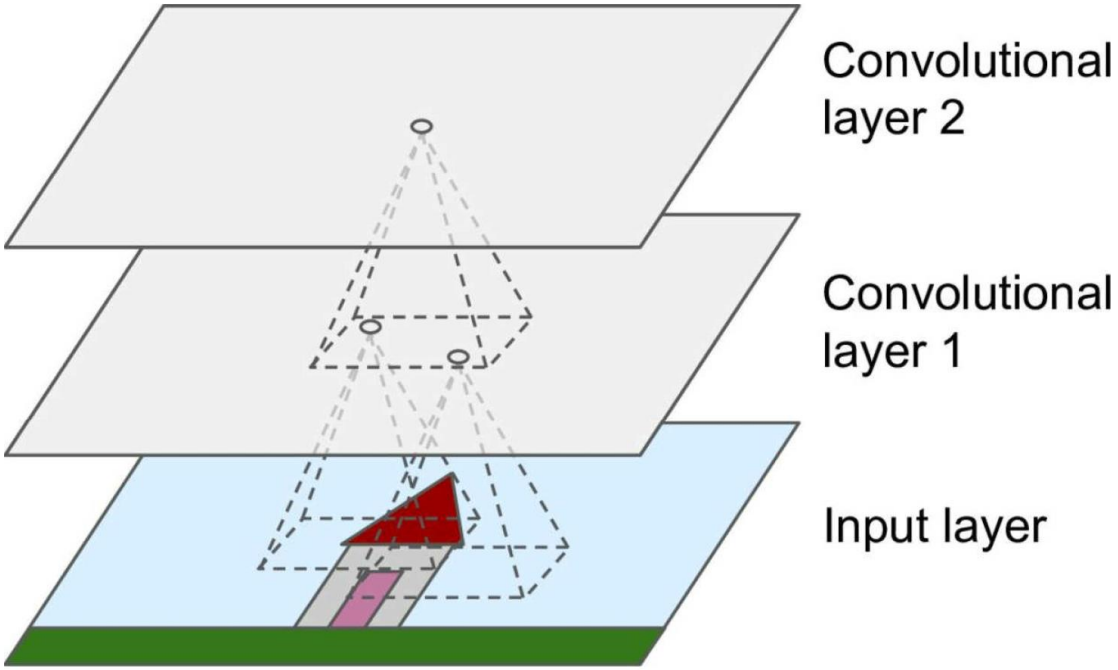
https://www.nature.com/articles/nature14539/figures/3

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Convolutional Layer (1)



- **Each neuron is connected only to a few other neurons that are within a small region.**

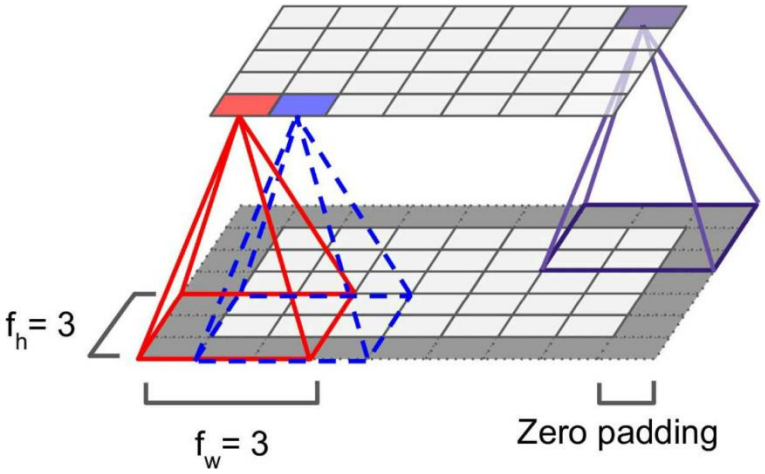


Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

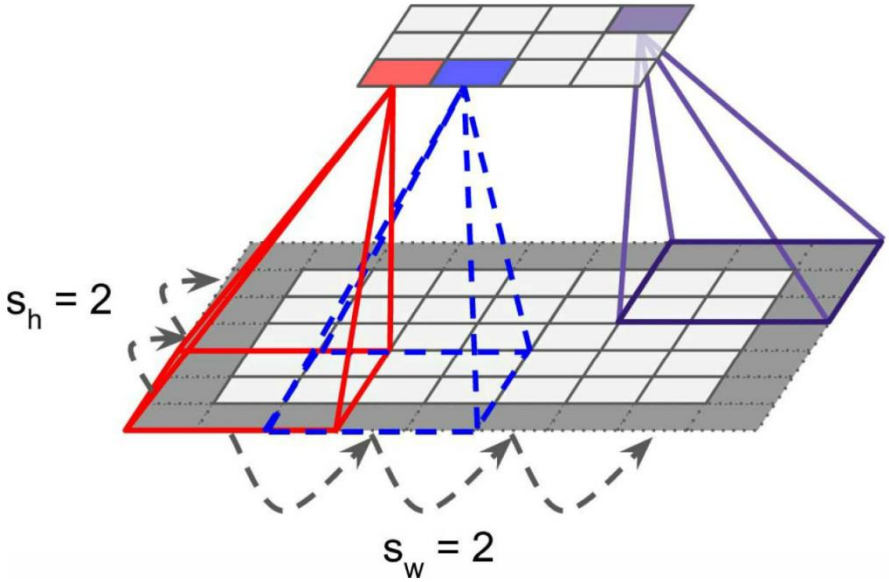
Convolutional Layer (2)

$f_h = Field\ Height$
$f_w = Field\ Width$

- **Same layer size**
  - Each neuron in row *i* and column *j* receives inputs from neurons located in previous layer at row *i* to $i + f_h - 1$, and column *j* to j $+ f_w - 1$
  - Zero padding ensures same size of layers



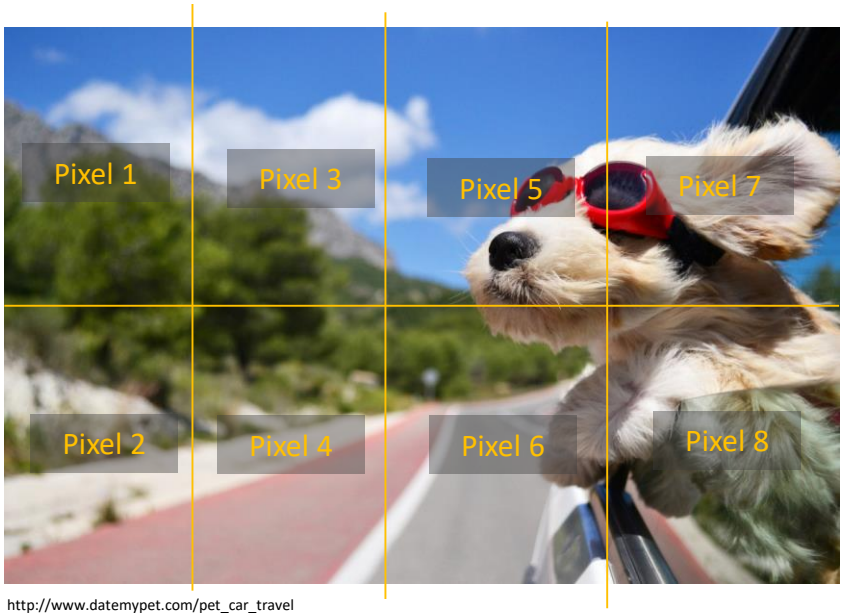
- **Smaller Layer size (reduce feature space)**
  - „Stride“ = Distance between two consecutive fields



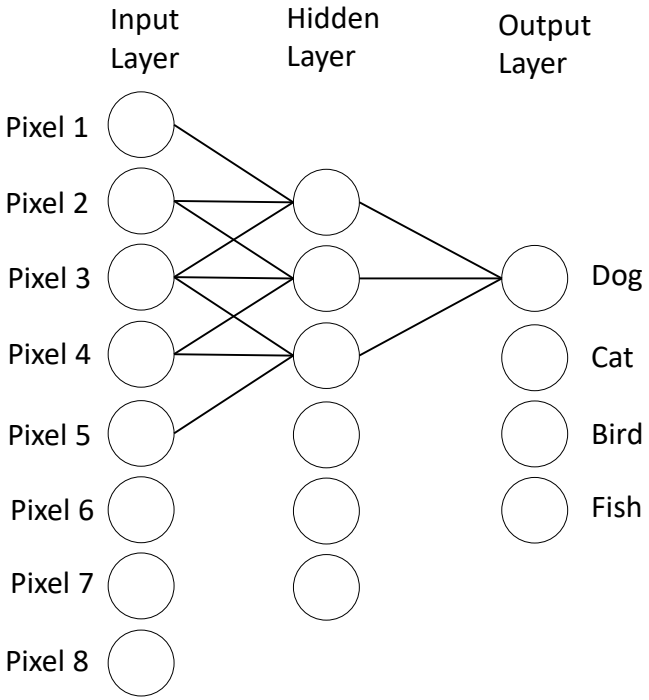
Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Question

- **Have Pixels 6-8 no impact on predicting whether the image shows a dog?**



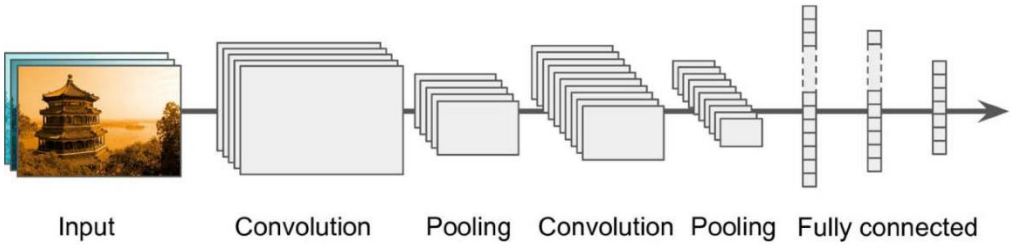
http://www.datemypet.com/pet_car_travel

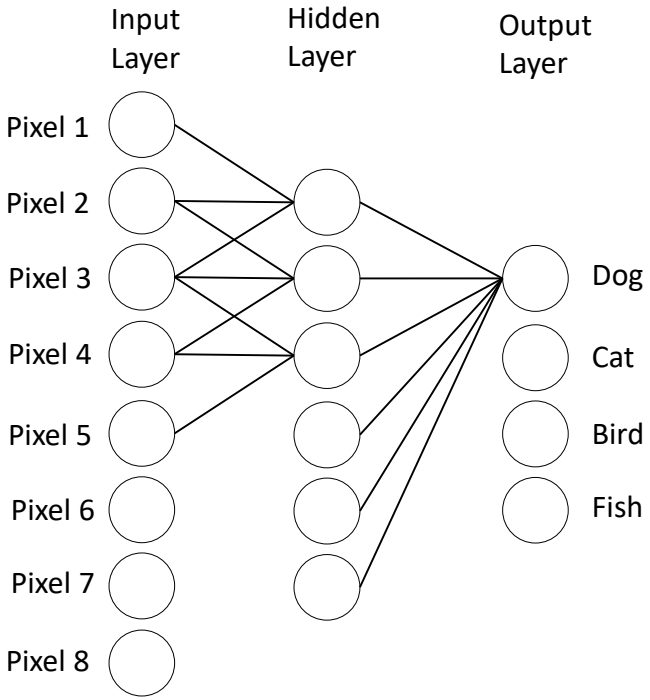


Answer

„Not really“

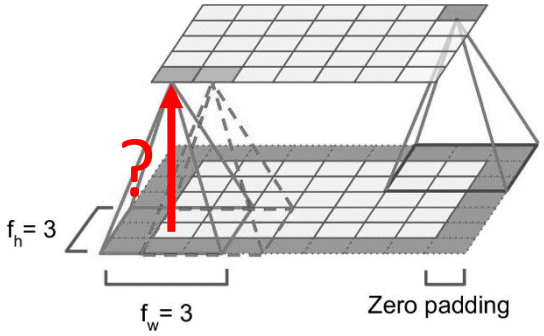
- **Even though a CNN is not fully connected, outputs are still effected by all (or most) inputs if there are enough layers.**
- **Fully-connected layers ensure this**

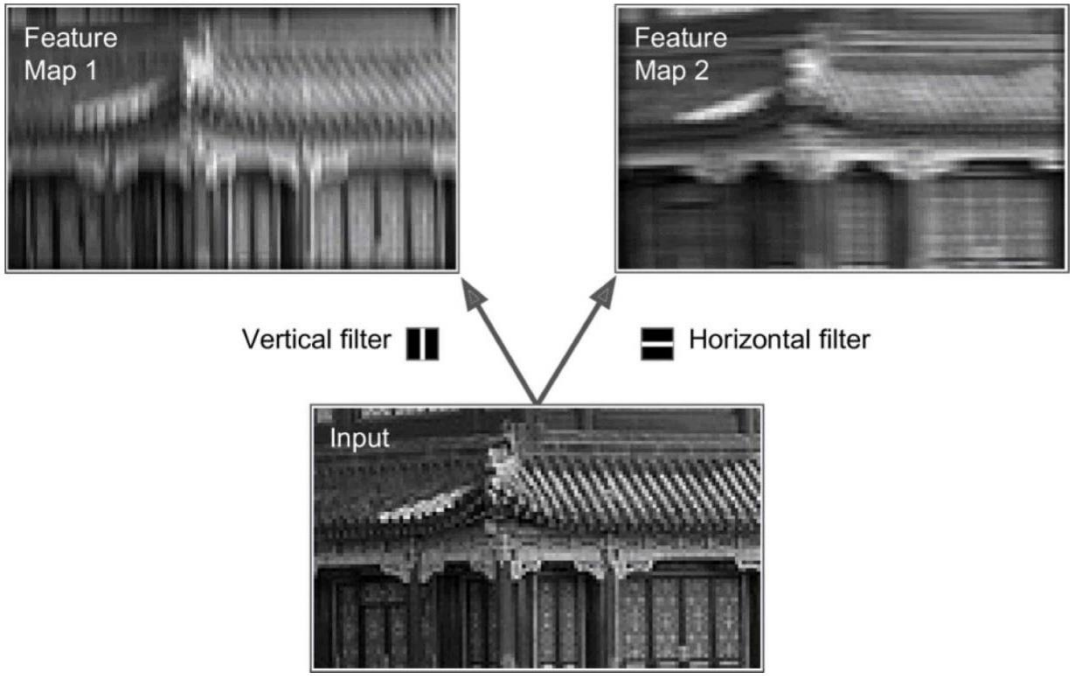




Convolutional Kernels / Filters / Weights

- **Weights in CNN are represented as filters**
- **Intuition: Filters are like „templates“ (e.g. for edges, circles, …)**
- **Filter Size = Size of Receptive Field**
- **Same filter used for all neurons in one layer**

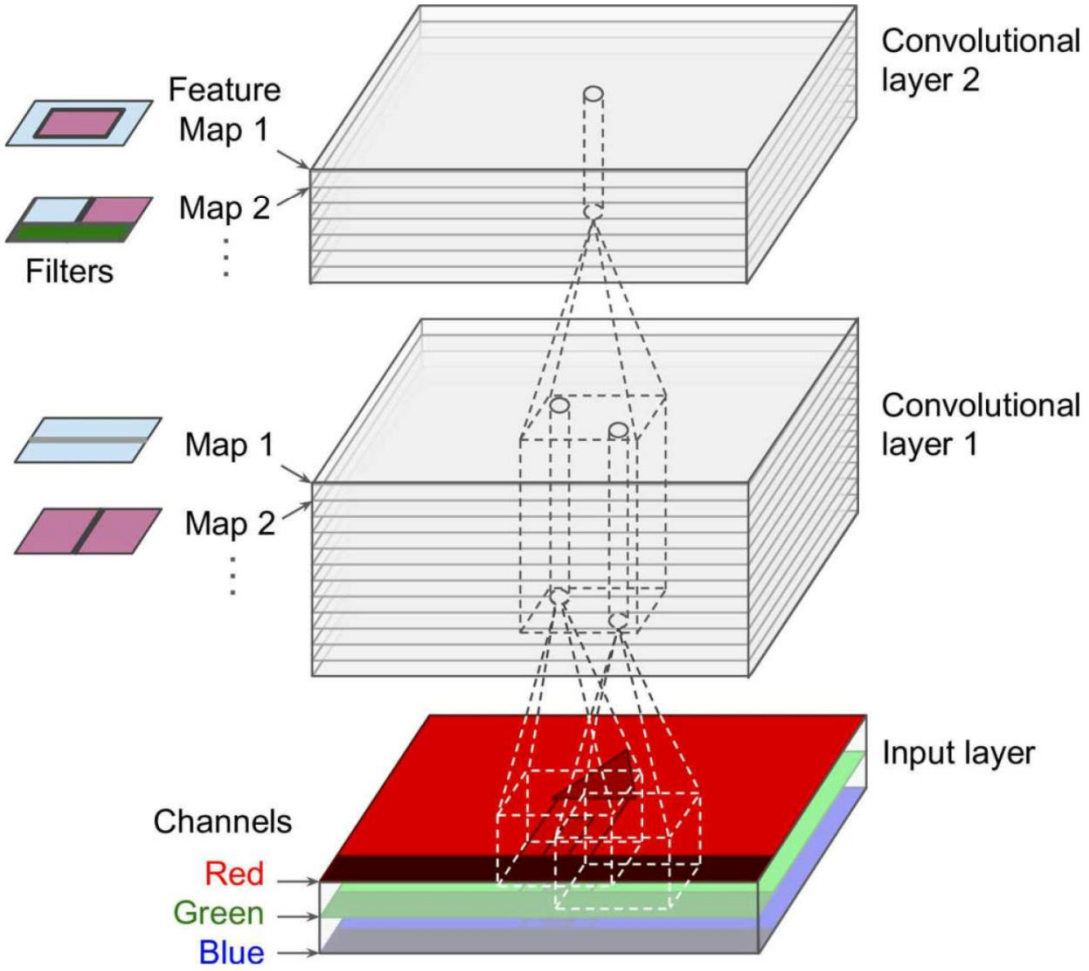




Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Stacking Feature Maps

- **Convolutional Layers are 3D Layers**
- **Stack of multiple feature maps of equal size**
- **All neurons in a feature map share same parameters (filters/weights and bias)**
- **Different feature maps may have different parameters**



Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Output of a Neuron

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k}$$

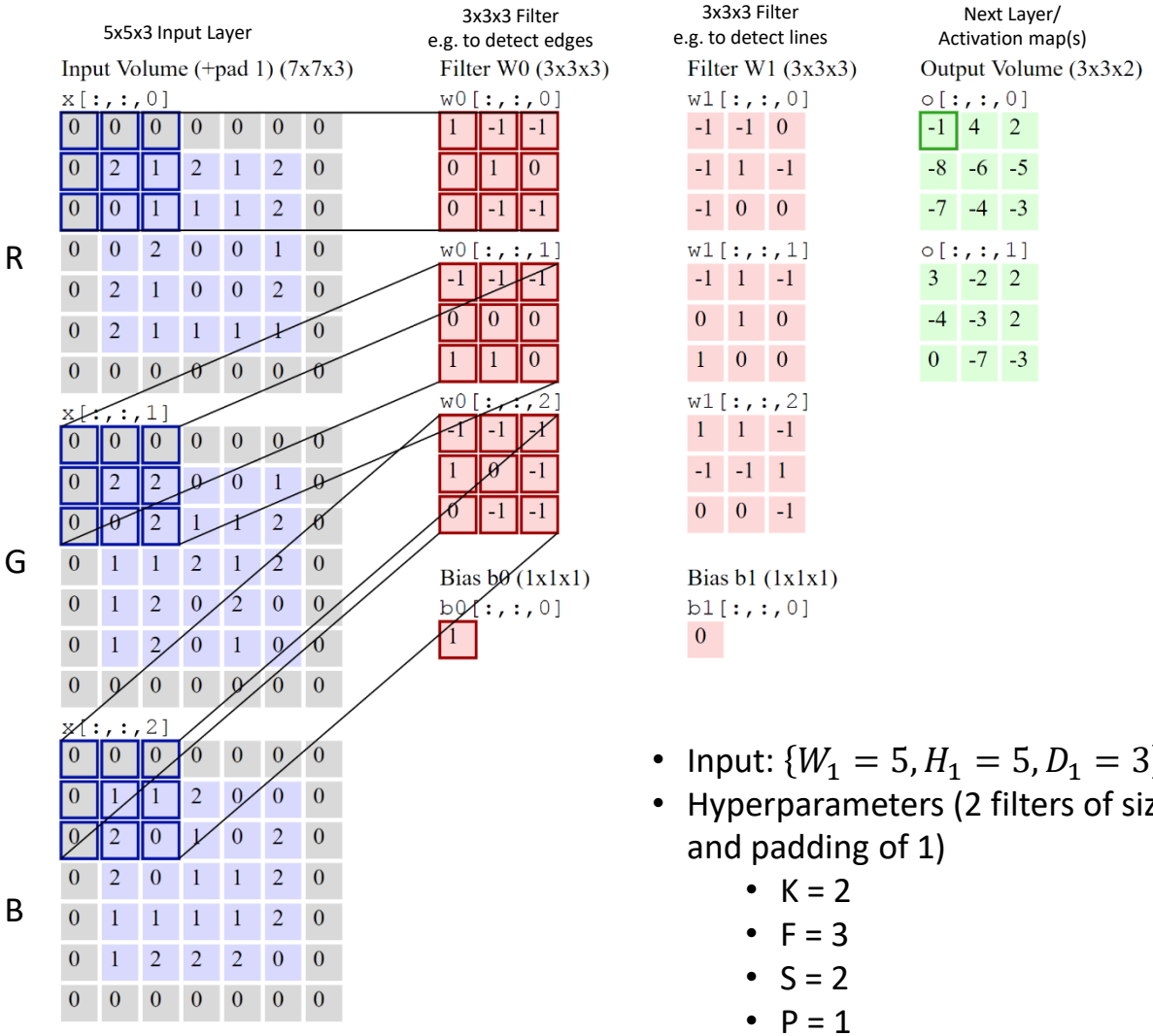
$$i' = i * s_h + u$$
$$j' = j * s_w + v$$

$z_{i,j,k}$	**Output of neuron in row *i*, column *j*, feature map *k* (in layer *l*)**
$s_{h/w}$	**Vertical/Horizontal Strides**
$f_{h/w}$	**Height/Width of receptive field**
$f_{n'}$	**Number of feature maps in previous layer (*l-1*)**
$x_{i',j',k'}$	**Output of neuron in layer *l-1*, row *i'*, column *j'*, feature map *k'***
b_k	**Bias term for feature map *k* (in layer *l*)**
$w_{u,v,k',k}$	**Connection weight between any neuron in feature map *k* of layer *l* and its input at row *u*, column *v*, and feature map *k'***

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Demo

http://cs231n.github.io/convolutional-networks/



- Input: $\{W_1 = 5, H_1 = 5, D_1 = 3\}$
- Hyperparameters (2 filters of size 3x3 with stride of 2 and padding of 1)
  - K = 2
  - F = 3
  - S = 2
  - P = 1
- Output: $\{W_2 = 3, H_2 = 3, D_2 = 2\}$

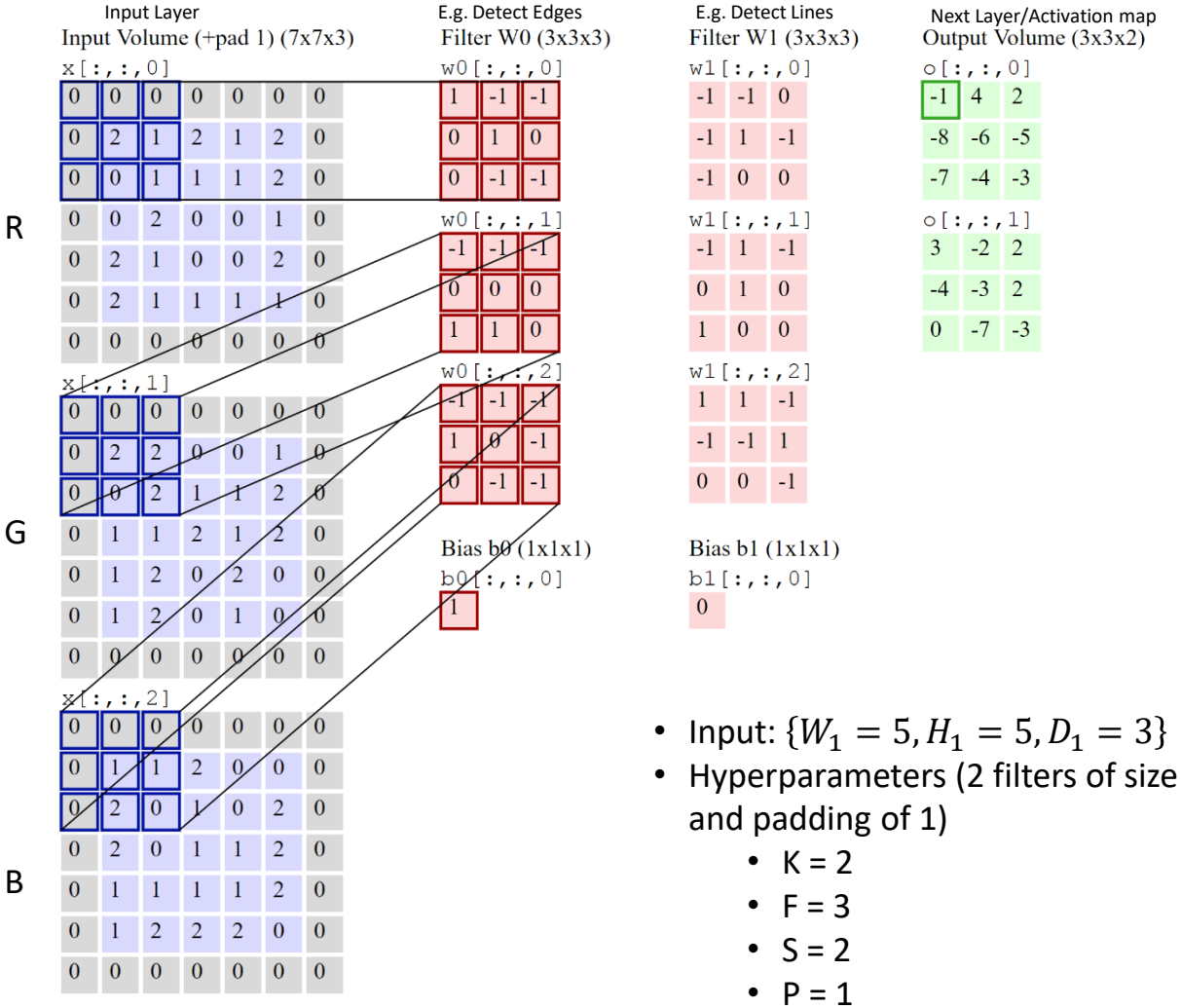
Demo

http://cs231n.github.io/convolutional-networks/

$$\begin{bmatrix}0&0&0\\0&2&1\\0&0&1\end{bmatrix}\cdot\begin{bmatrix}1&-1&-1\\0&1&0\\0&-1&-1\end{bmatrix}=\begin{bmatrix}0&0&0\\0&2&0\\0&0&-1\end{bmatrix}=1$$

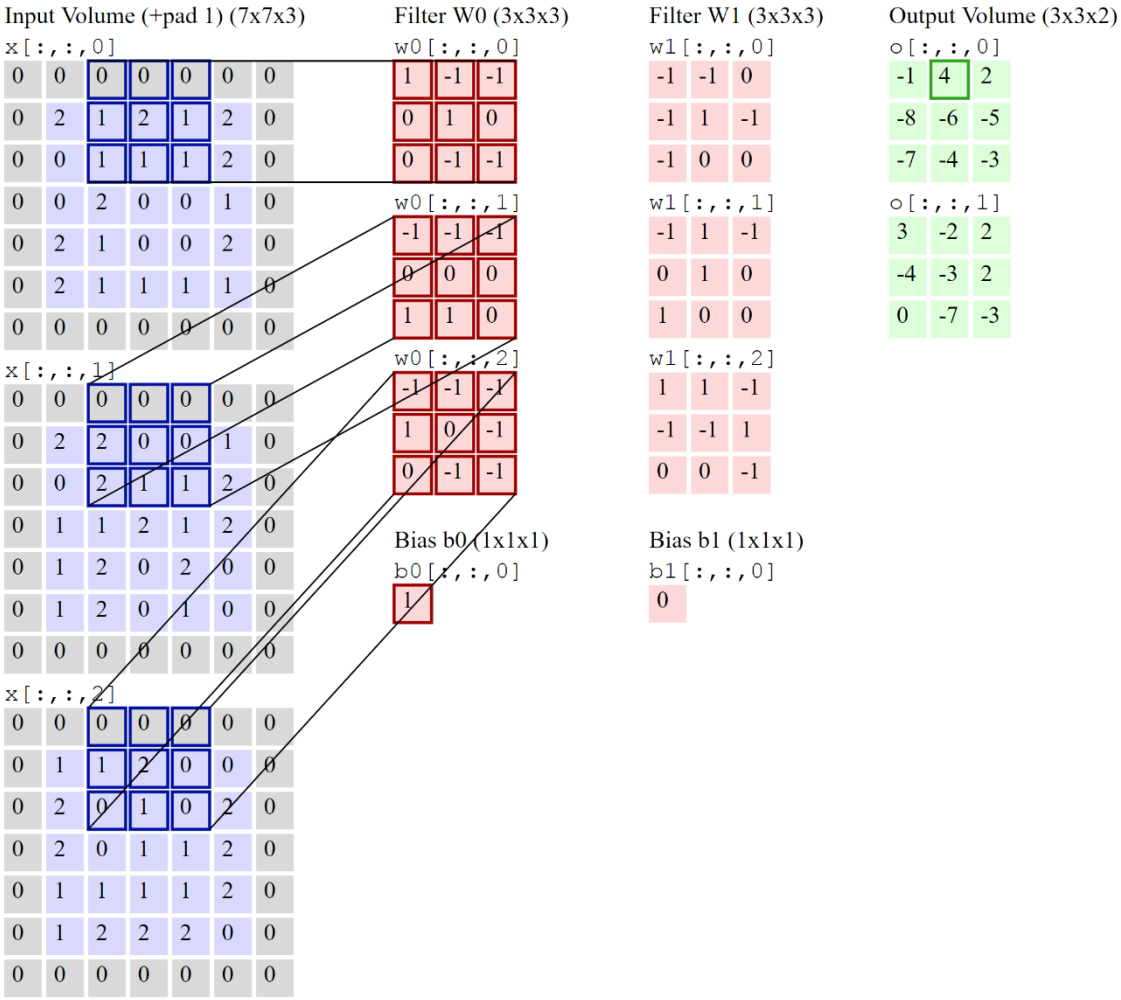
$$\begin{bmatrix}0&0&0\\0&2&2\\0&0&2\end{bmatrix}\cdot\begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&0\end{bmatrix}=\begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix}=0$$

$$\begin{bmatrix}0&0&0\\0&1&1\\0&2&0\end{bmatrix}\cdot\begin{bmatrix}-1&-1&-1\\1&0&-1\\0&-1&-1\end{bmatrix}=\begin{bmatrix}0&0&0\\0&0&-1\\0&-2&0\end{bmatrix}=-3$$

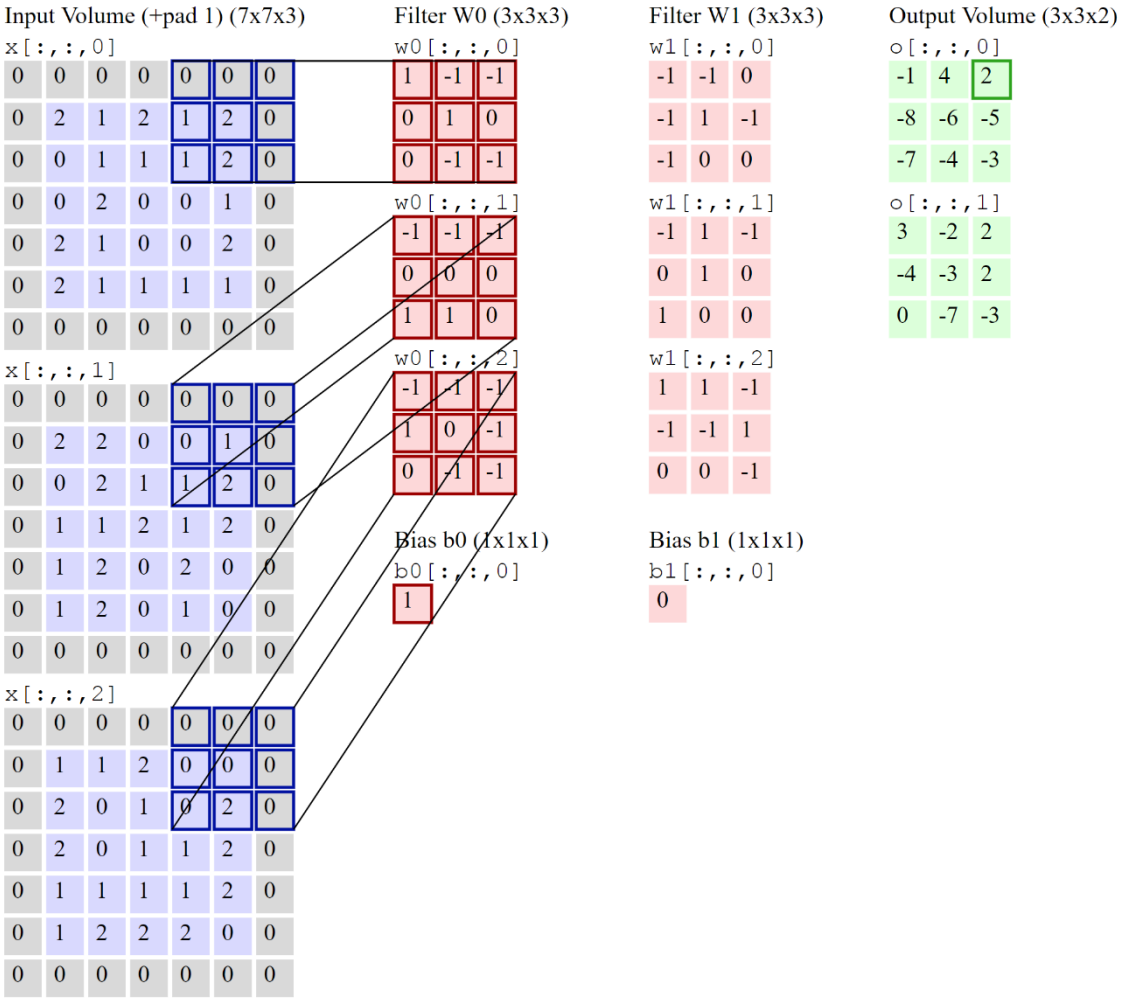


- Input: $\{W_1 = 5, H_1 = 5, D_1 = 3\}$
- Hyperparameters (2 filters of size 3x3 with stride of 2 and padding of 1)
  - K = 2
  - F = 3
  - S = 2
  - P = 1
- Output: $\{W_2 = 3, H_2 = 3, D_2 = 2\}$

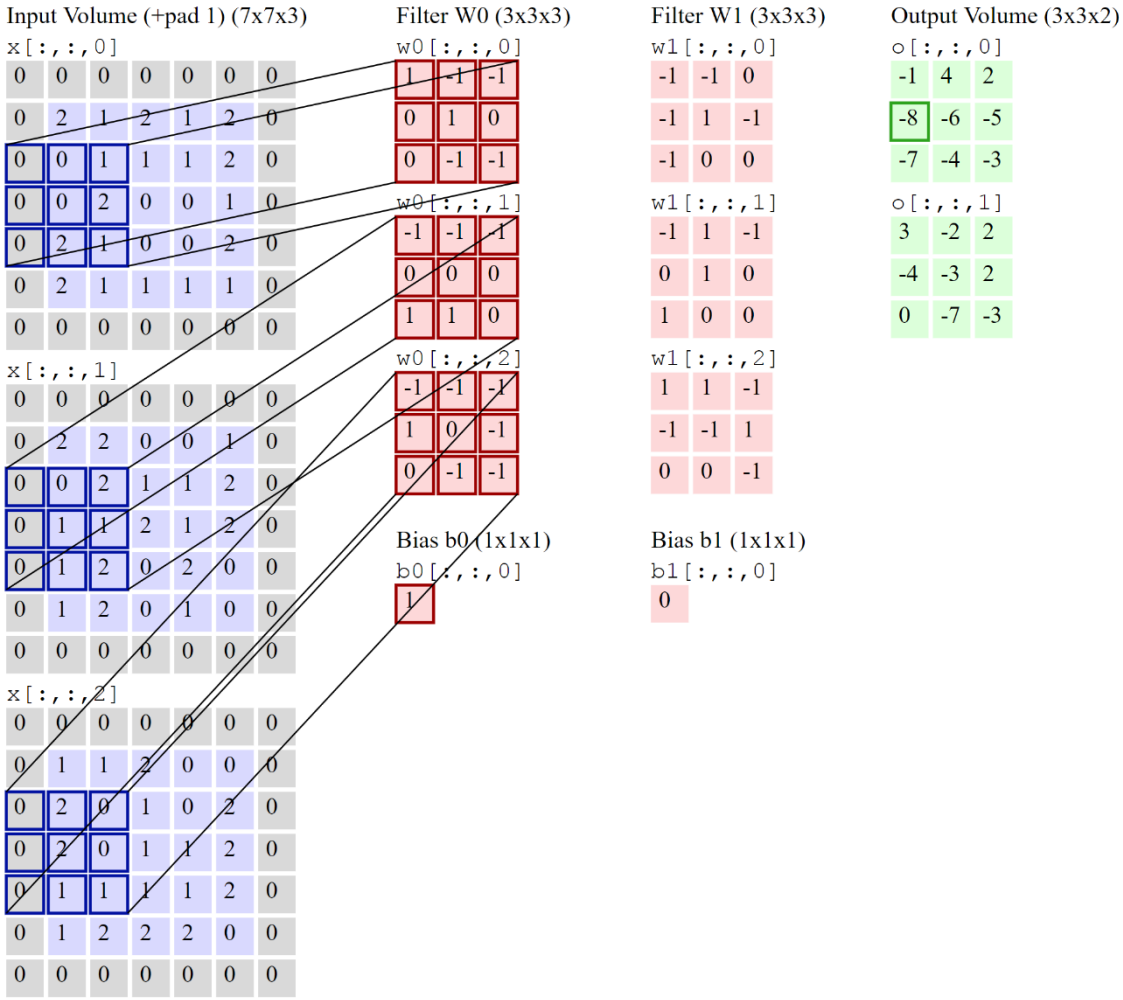
Demo


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 -1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

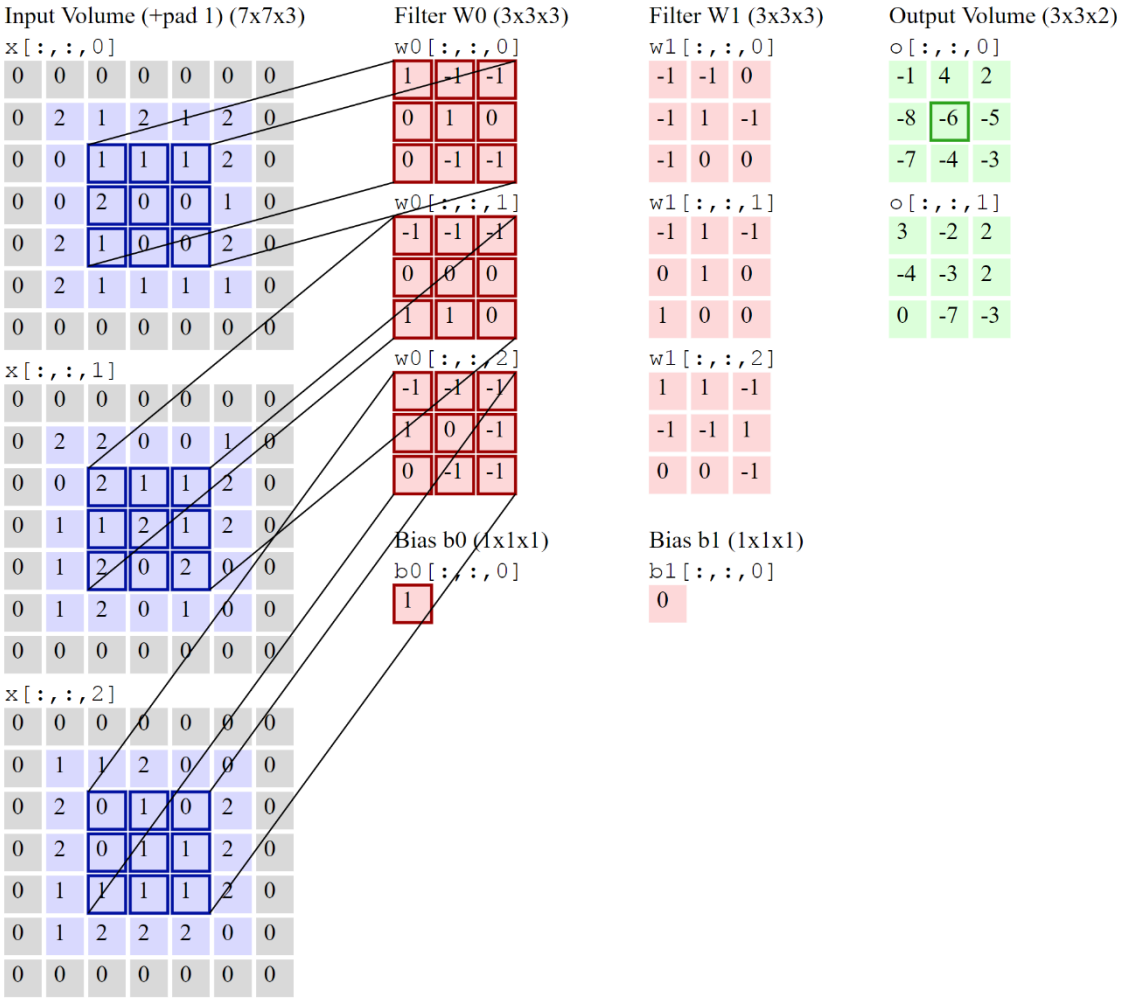
Demo


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 -1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

Demo

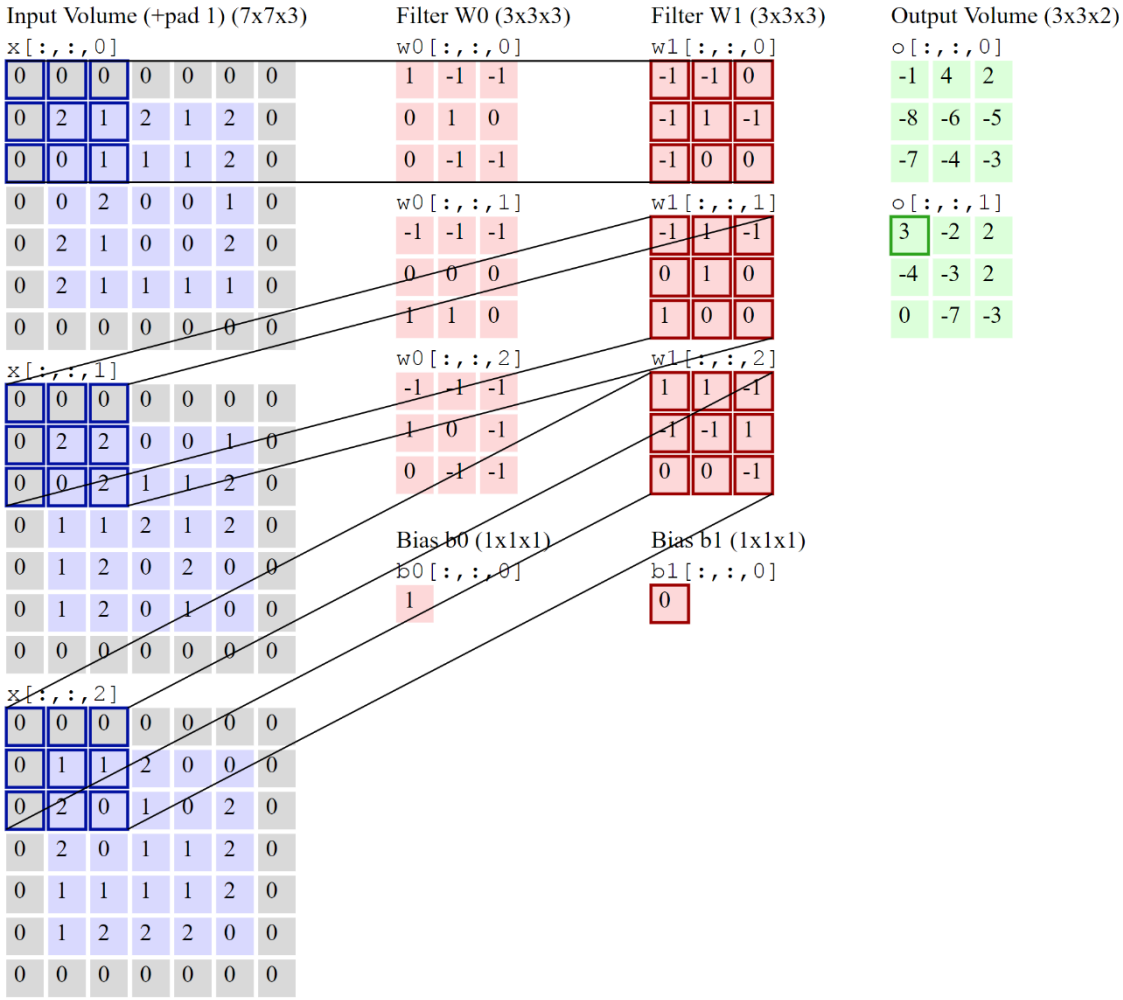

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 -1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

Demo

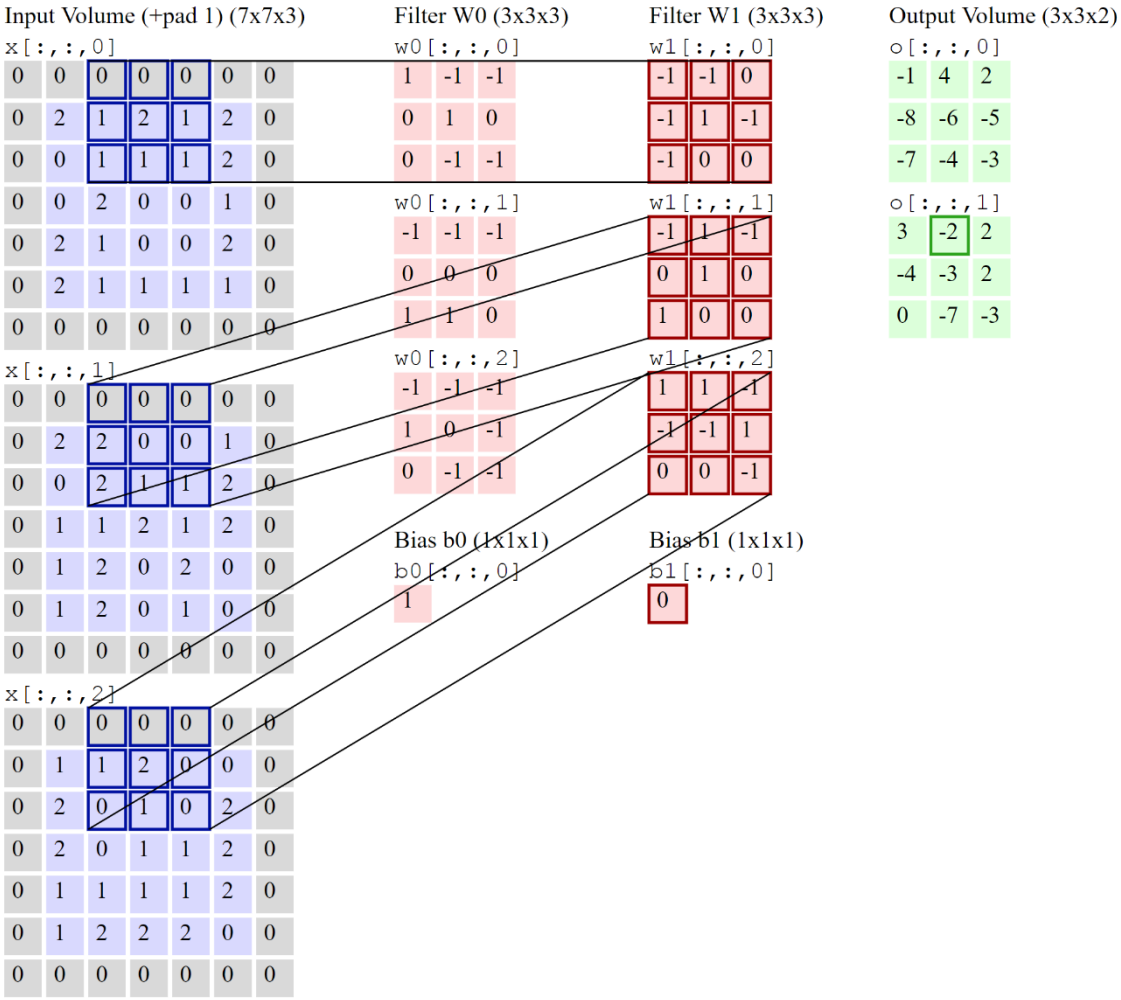

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 -1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

Demo

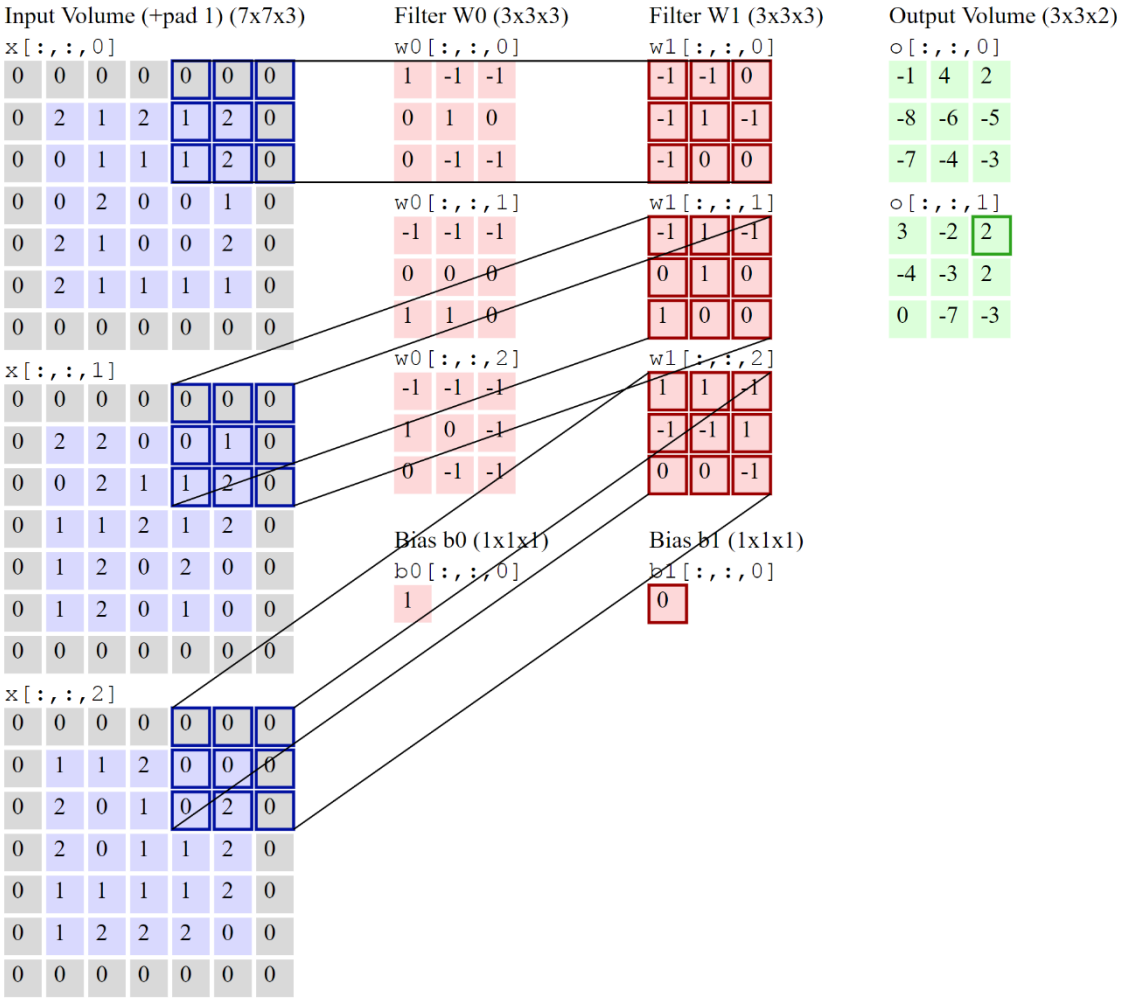
Demo


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 -1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

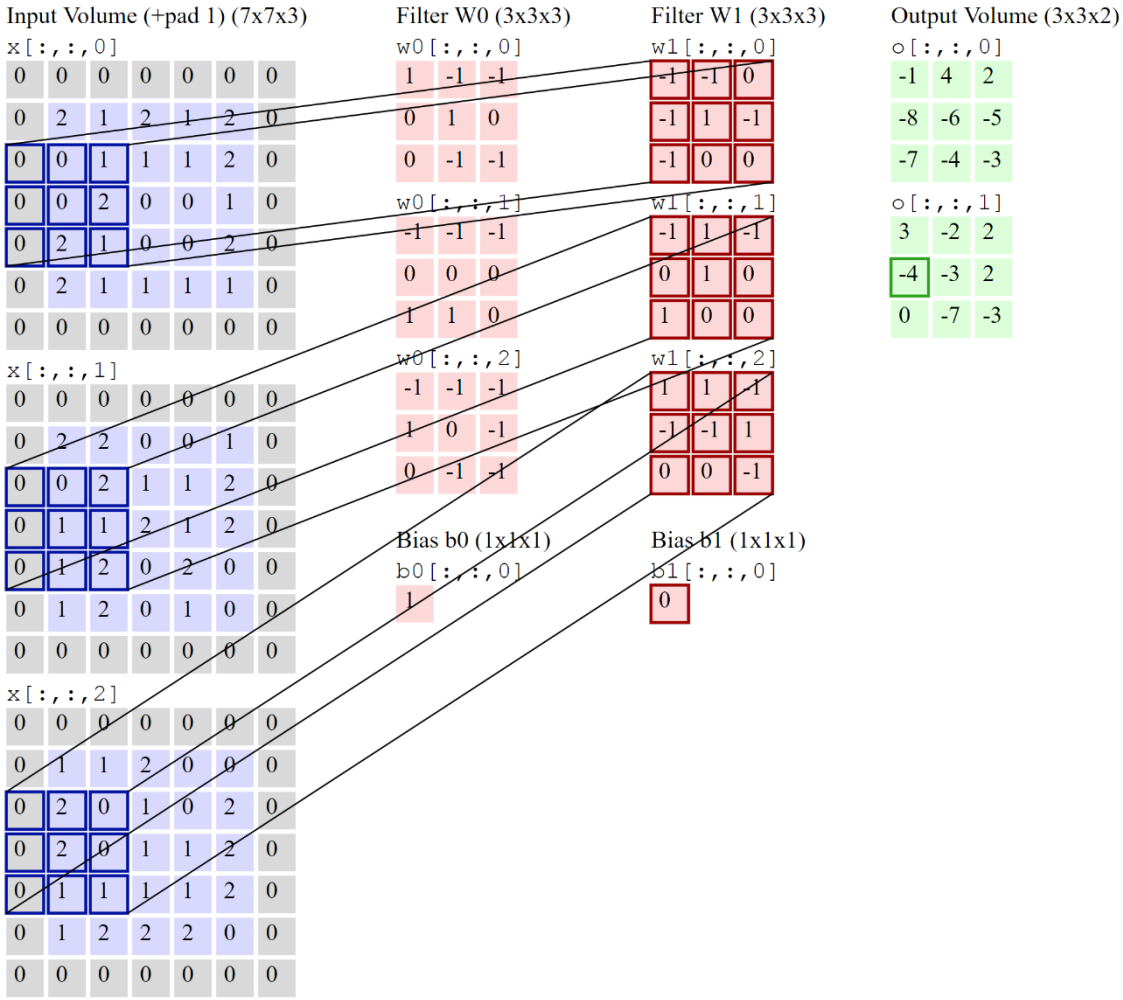
Demo


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 -1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

Demo

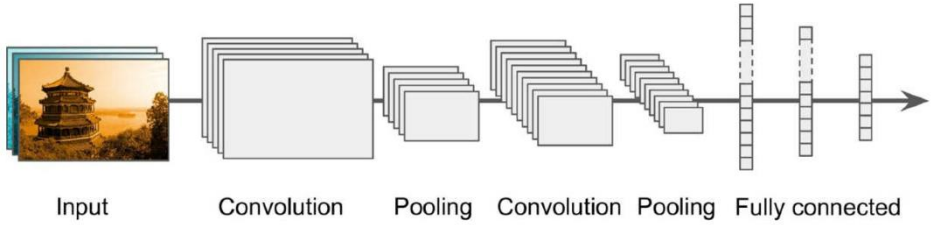

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

Demo


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 2 1 2 1 2 0
0 0 1 1 1 2 0
0 0 2 0 0 1 0
0 2 1 0 0 2 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 2 2 0 0 1 0
0 0 2 1 1 2 0
0 1 1 2 1 2 0
0 1 2 0 2 0 0
0 1 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 1 2 0 0 0
0 2 0 1 0 2 0
0 2 0 1 1 2 0
0 1 1 1 1 2 0
0 1 2 2 2 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 -1 -1
0 1 0
0 -1 -1
w0[:,:,1]
-1 -1 -1
0 0 0
1 1 0
w0[:,:,2]
-1 -1 -1
1 0 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 -1
-1 0 0
w1[:,:,1]
-1 1 -1
0 1 0
1 0 0
w1[:,:,2]
1 1 1
-1 -1 1
0 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-1 4 2
-8 -6 -5
-7 -4 -3
o[:,:,1]
3 -2 2
-4 -3 2
0 -7 -3

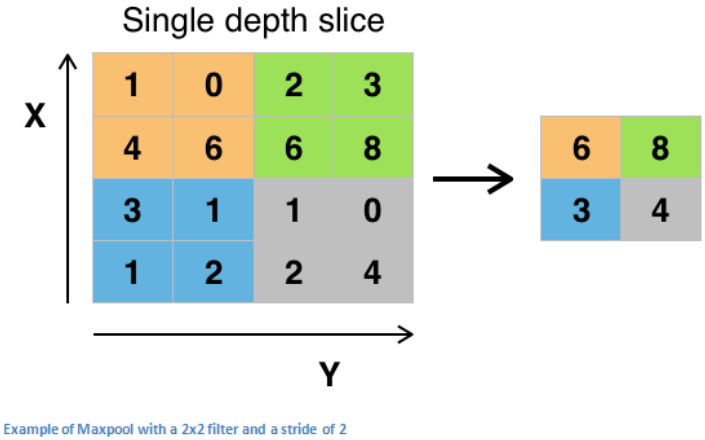
Demo

Pooling Layer

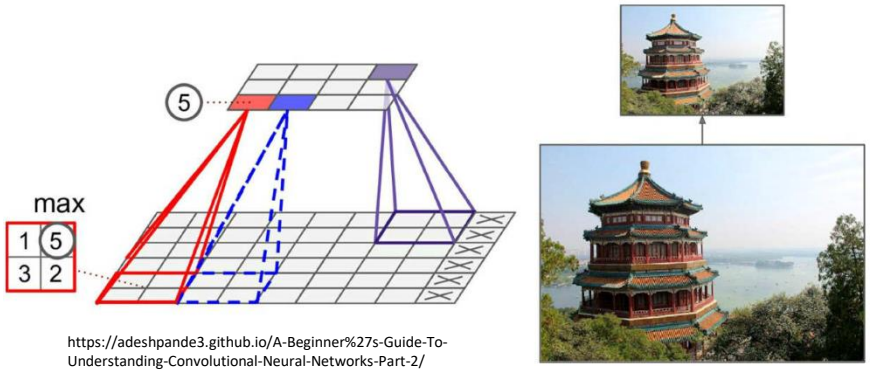


- **Purpose**
  - Subsample input to reduce computational load, memory, and parameters
  - make robust against variations
- **Same architecture as a convolutional layer (perceptive field, stride, padding), but no weights**
- **Aggregating function: max or mean**
- **Example below**
  - 2x2 pooling kernel
  - Stride = 2
  - No padding

→ Drops 75% of the input values



Example of Maxpool with a 2x2 filter and a stride of 2



https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Dimension Example (1)

- **Input: 128x128 image**

  → [128x128x3] array

- **Filter: Size: 5x5, Padding=1, Stride=1**

  → [5x5x3] array

- **1st convolutional layer e.g. with 12 filters**

  → [128x128x12] array

- **Apply ReLU activation**

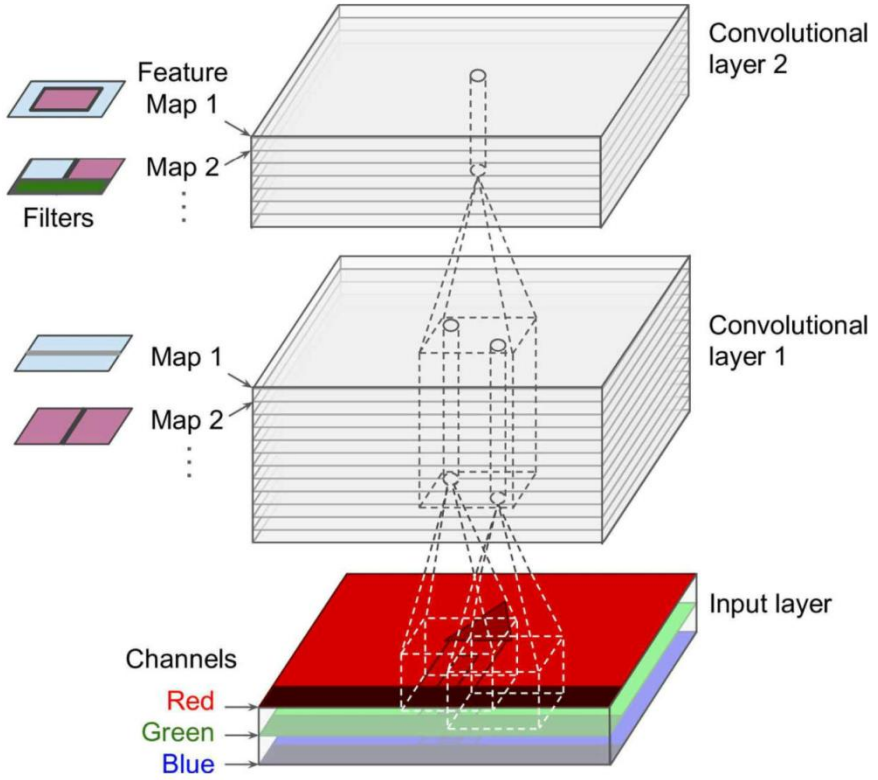
  → [128x128x12] array

- **Pool layers (down sample image)**

  → [32x32x12] array

- **2nd convolutional layer e.g. with 64 filters**

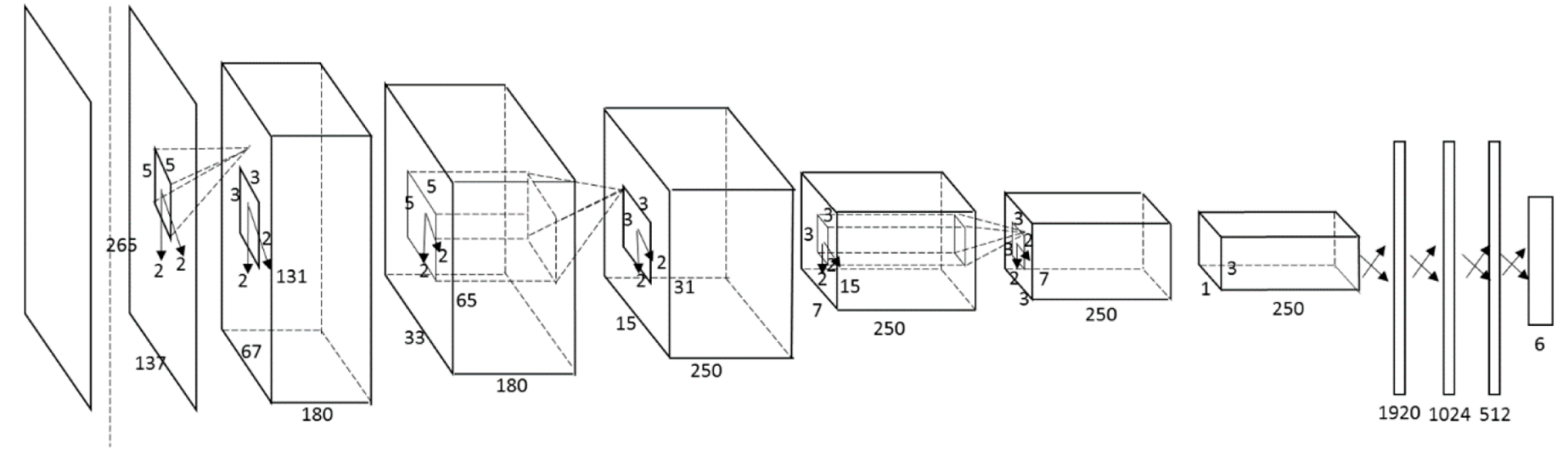
  → [32x32x64] array



http://cs231n.github.io/convolutional-networks/

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Dimension Example (2)



Input image
[265 137]

Conv. layer
Filter size [5 5]
Filter num. 180
Stride [2 2]
+ReLU

Max pooling
Filter size [3 3]
Stride [2 2]

Conv. layer
Filter size [5 5]
Filter num. 250
Stride [2 2]
+ReLU

Max pooling
Filter size [3 3]
Stride [2 2]

Conv. layer
Filter size [3 3]
Filter num. 250
Stride [2 2]
+ReLU

Max pooling
Filter size [3 3]
Stride [2 2]

Full connection Fully con.
+ReLU/dropout +soft Max

http://www.mdpi.com/sensors/sensors-16-02160/article_deploy/html/images/sensors-16-02160-g002.png

Architecture Variations

- **LeNet-5 (1998)**
- **AlexNet (2012)**
- **GoogleLeNet (2014)**
- **ResNet (2015)**

Table 13-1. LeNet-5 architecture

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully Connected	–	10	–	–	RBF
F6	Fully Connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg Pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg Pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

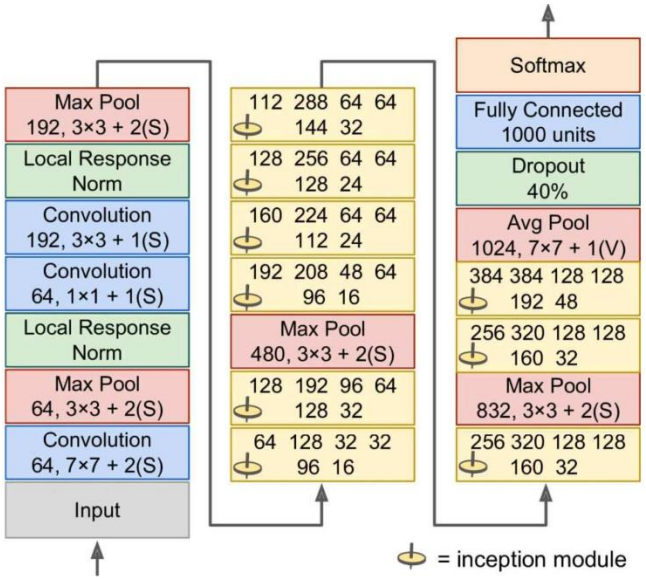


Figure 13-11. GoogLeNet architecture

Table 13-2. AlexNet architecture

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Training CNNs

- **Via Backpropagation**
- **Read, e.g.**
  - http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Q in the summation above represents the output region bounded by dashed lines and is composed of pixels in the output that are affected by the single pixel $x_{i',j'}$ in the input feature map. A more formal way of representing Eq. 16 is:

$$\begin{aligned}\frac{\partial E}{\partial x^l_{i',j'}} &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1}\frac{\partial E}{\partial x^{l+1}_{i'-m,j'-n}}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} \\ &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1}\delta^{l+1}_{i'-m,j'-n}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} \end{aligned} \tag{17}$$

In the region Q, the height ranges from $i'-0$ to $i'-(k_1-1)$ and width $j'-0$ to $j'-(k_2-1)$. These two can simply be represented by $i'-m$ and $j'-n$ in the summation since the iterators m and n exists in the following similar ranges from $0 \leq m \leq k_1-1$ and $0 \leq n \leq k_2-1$.

In Eq. 17 , $x^{l+1}_{i'-m,j'-n}$ is equivalent to $w^{l+1}_{m',n'}o^l_{i'-m+m',j'-n+n'}+b^{l+1}$ and expanding this part of the equation gives us:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} &= \frac{\partial}{\partial x^l_{i',j'}}\left(\sum_{m'}\sum_{n'}w^{l+1}_{m',n'}o^l_{i'-m+m',j'-n+n'}+b^{l+1}\right) \\ &= \frac{\partial}{\partial x^l_{i',j'}}\left(\sum_{m'}\sum_{n'}w^{l+1}_{m',n'}f\left(x^l_{i'-m+m',j'-n+n'}\right)+b^{l+1}\right)\end{aligned} \tag{18}$$

Further expanding the summation in Eq. 17 and taking the partial derivatives for all the components results in zero values for all except the components where $m'=m$ and $n'=n$ so that $f\left(x^l_{i'-m+m',j'-n+n'}\right)$ becomes $f\left(x^l_{i',j'}\right)$ and $w^{l+1}_{m',n'}$ becomes $w^{l+1}_{m,n}$ in the relevant part of the expanded summation as follows:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} &= \frac{\partial}{\partial x^l_{i',j'}}\left(w^{l+1}_{m',n'}f\left(x^l_{0-m+m',0-n+n'}\right)+\cdots+w^{l+1}_{m,n}f\left(x^l_{i',j'}\right)+\cdots+b^{l+1}\right) \\ &= \frac{\partial}{\partial x^l_{i',j'}}\left(w^{l+1}_{m,n}f\left(x^l_{i',j'}\right)\right) \\ &= w^{l+1}_{m,n}\frac{\partial}{\partial x^l_{i',j'}}\left(f\left(x^l_{i',j'}\right)\right) \\ &= w^{l+1}_{m,n}f'\left(x^l_{i',j'}\right)\end{aligned} \tag{19}$$

Substituting Eq. 19 in Eq. 17 gives us the following results:

$$\frac{\partial E}{\partial x^l_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1}\delta^{l+1}_{i'-m,j'-n}w^{l+1}_{m,n}f'\left(x^l_{i',j'}\right) \tag{20}$$

For backpropagation, we make use of the flipped kernel and as a result we will now have a convolution that is expressed as a cross-correlation with a flipped kernel:

$$\begin{aligned}\frac{\partial E}{\partial x^l_{i',j'}} &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1}\delta^{l+1}_{i'-m,j'-n}w^{l+1}_{m,n}f'\left(x^l_{i',j'}\right) \\ &= \mathrm{rot}_{180^\circ}\left\{\sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1}\delta^{l+1}_{i'+m,j'+n}w^{l+1}_{m,n}\right\}f'\left(x^l_{i',j'}\right) \qquad (21) \\ &= \delta^{l+1}_{i',j'} * \mathrm{rot}_{180^\circ}\left\{w^{l+1}_{m,n}\right\}f'\left(x^l_{i',j'}\right) \qquad (22)\end{aligned}$$

http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

Video Illustration

https://www.youtube.com/watch?v=AgkfIQ4IGaM

More Reading

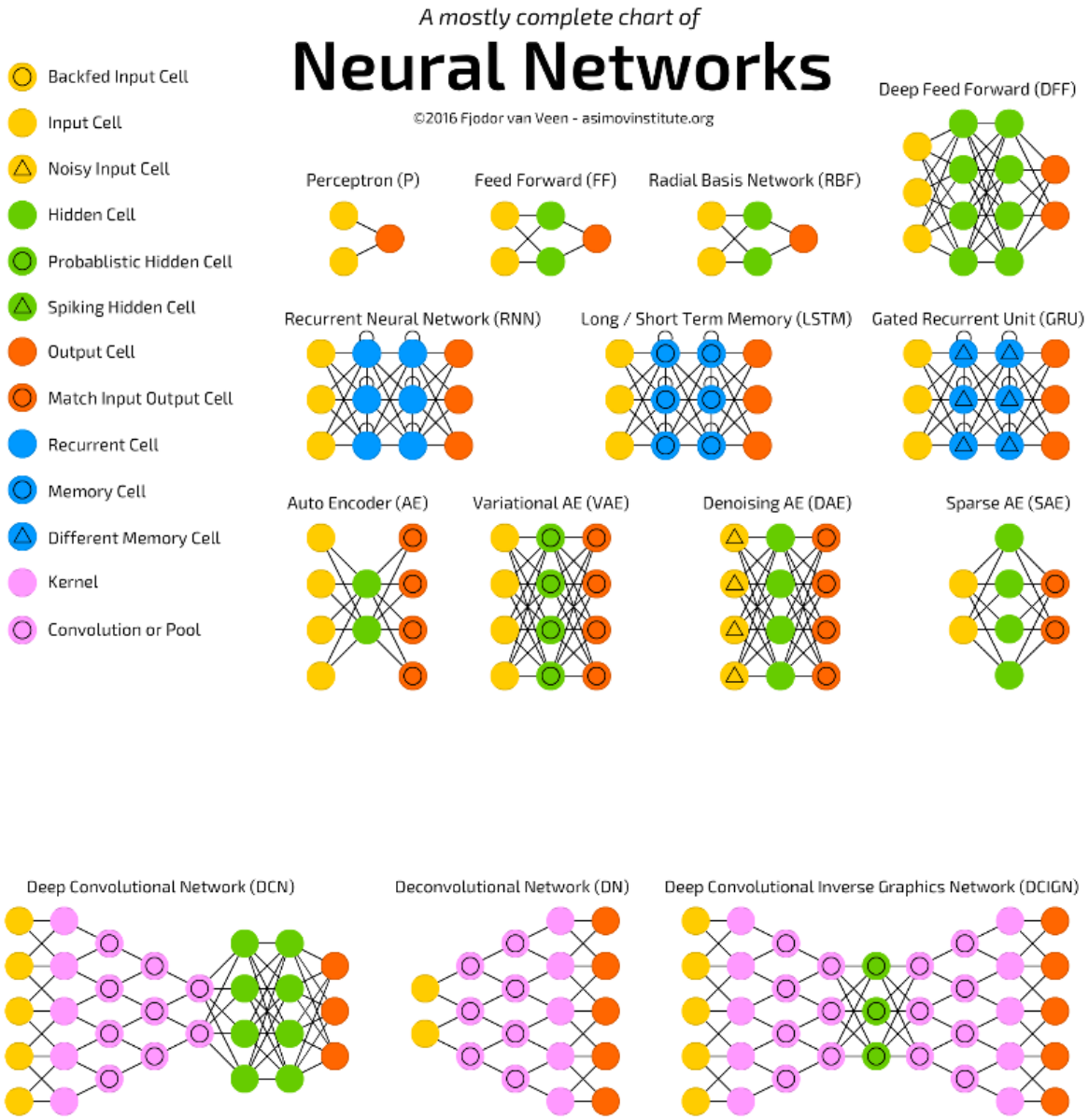
- **Practical**
  - Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).
  - https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
- **More theoretical**
  - Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep learning (MIT press, 2016).
  - http://cs231n.github.io/convolutional-networks/

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Recurrent Neural Networks

So Far…

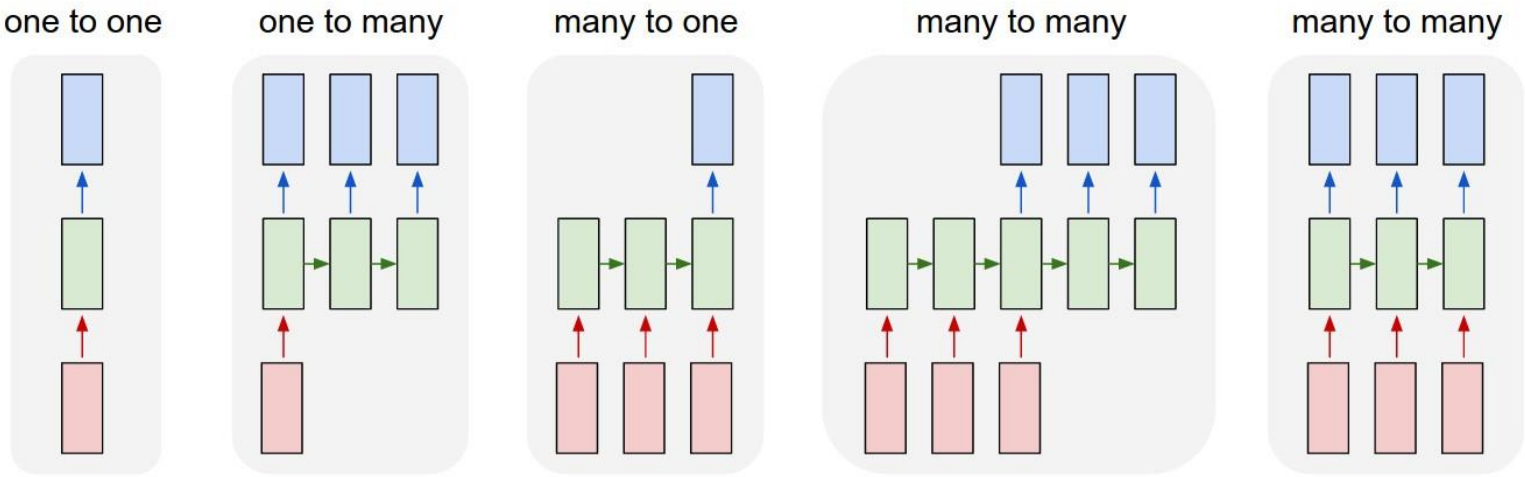
- **Feed-Forward Networks: From Input Layer to Output Layer**
- **Now: Recurrent**



http://www.asimovinstitute.org/wp-content/uploads/2016/09/neuralnetworks.png

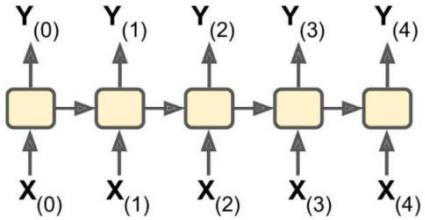
Applications

- **Particularly suited for sequence inputs of arbitrary length**
- Predict next character or word in a sequence
- „Predict“ next note in a melody → create new songs
- Translation: English Sentence→ Output the main idea behind the sentence → translate to e.g. German
- Stock Market
- Autonomous Driving

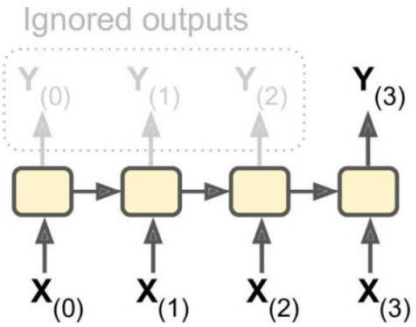


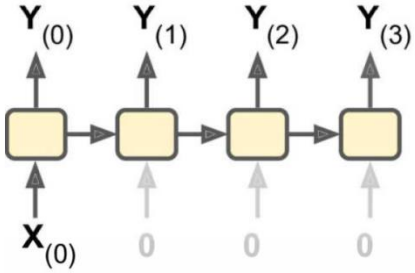
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

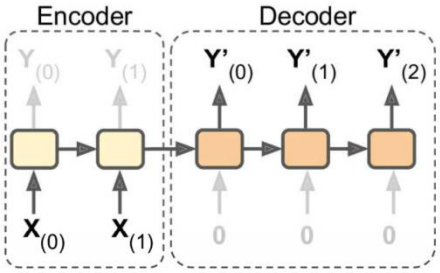
Input and Output Sequences



- **Sequence-to-Sequence**
  - Predict outputs in the future
  - E.g. predict stock prices
- **Sequence-to-Vector**
  - Output a single vector
  - E.g. sentiment score
- **Vector-to-Sequence**
  - Feed single input (and 0s for next time steps); let it output a sequence
  - E.g. Image caption
- **Delayed Sequence-to-Sequence**
  - Feed sequence; output vector; use that vector as input; output sequence
  - E.g. machine translation



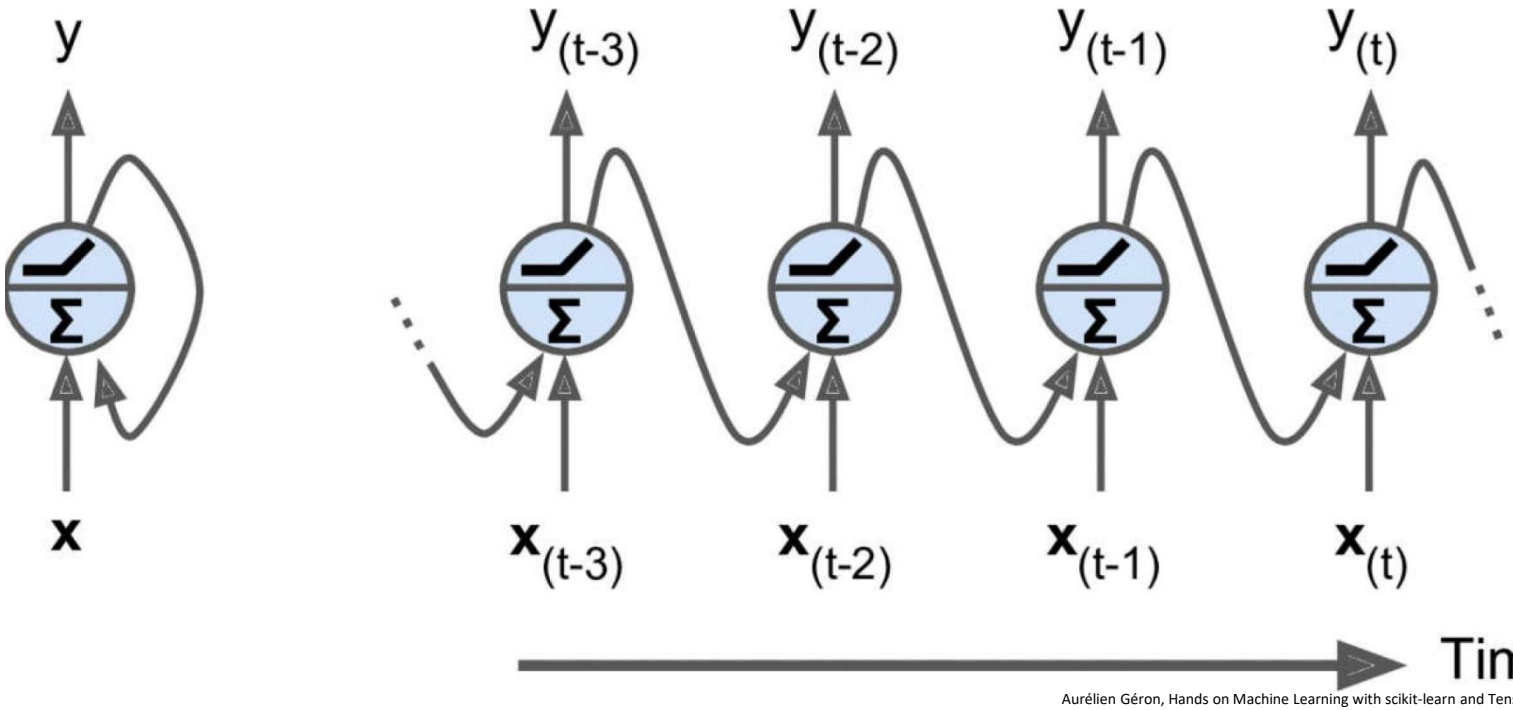




Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

A Recurrent Neuron

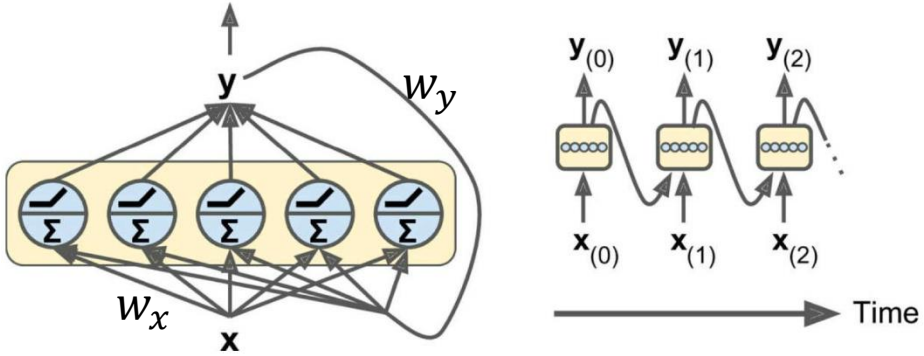
- **Single input *x* to one Neuron**
- **Output *z* being sent back to the same Neuron**
- **At each time step („frame“), the neuron receives a new *x* and its own output from the previous time step *t-1***



Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Multiple Neurons (in one layer)

- **At each time step, every neuron receives input vector and output vector** $y_{(t-1)}$
- **Each neuron has set of weight vectors for inputs and outputs of the previous timestep**
- **Output of a single recurrent neuron for a single instance:** $y = \varphi(z)$ **with** $z = b + x_{(t)} * w_x + y_{(t-1)} * w_y$



- **More precisely, since inputs, outputs and weights are vectors:**

$$z = b + x_{(t)}^T \cdot w_x + y_{(t-1)}^T \cdot w_y$$

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

For Mini-Batches (Multiple Instances)

$$Y_{(t)} = \varphi(z) = \boldsymbol{\varphi}\left(\boldsymbol{b} + \boldsymbol{X_{(t)}} \cdot \boldsymbol{W_x} + \boldsymbol{Y_{(t-1)}} \cdot \boldsymbol{W_y}\right) = \varphi\left(\left[X_{(t)} \cdot Y_{(t-1)}\right]\right) \cdot W + b)$$

$Y_{(t)}$ — $\mathrm{m} * n_{neurons}$ **matrix containing a layer's *n* neuron's outputs at timestep t, for each of the *m* instances**

$X_{(t)}$ — $\mathrm{m} * n_{inputs}$ **matrix containing the instances' *n* input features**

W_x — $n_{inputs} * n_{neurons}$ **matrix with connections weights for inputs of current time step**

W_y — $n_{neurons} * n_{neurons}$ **matrix with connections weights for outputs of previous time step**

b — **Vector of size $n_{neurons}$ containing each neuron's bias term**

W — **Concatenated matrix** $\begin{bmatrix} W_x \\ W_y \end{bmatrix}$ **of shape** $\left(n_{inputs} + n_{neurons}\right) * n_{neurons})$

Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Memory Cells

$Y_{(t)}$ is a function of $X_{(t)}$ and $Y_{(t-1)}$, which is a function of $X_{(t-1)}$ and $Y_{(t-2)}$ … .

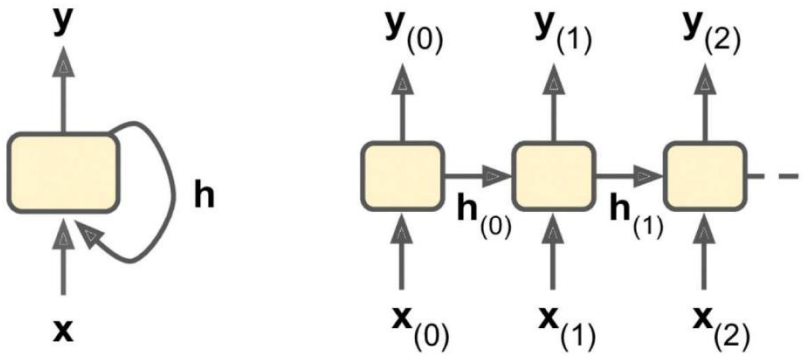
→ **$Y_{(t)}$ is a function of all inputs since $t=0$, i.e $X_{(0)}$, $X_{(1)}$,…, $X_{(t)}$**

→ **$Y_{(t)}$ has kind of a memory**

→ **A part of a neural network that preserves some state across time is called a "memory cell" (or "cell"). There are more complex memory cells than the one we cover in this lecture.**

- **A cell's state at time t is denoted $h_{(t)}$**

$h_{(t)} = f(h_{(t-1)}, x_{(t)})$



Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

More Details on RNN

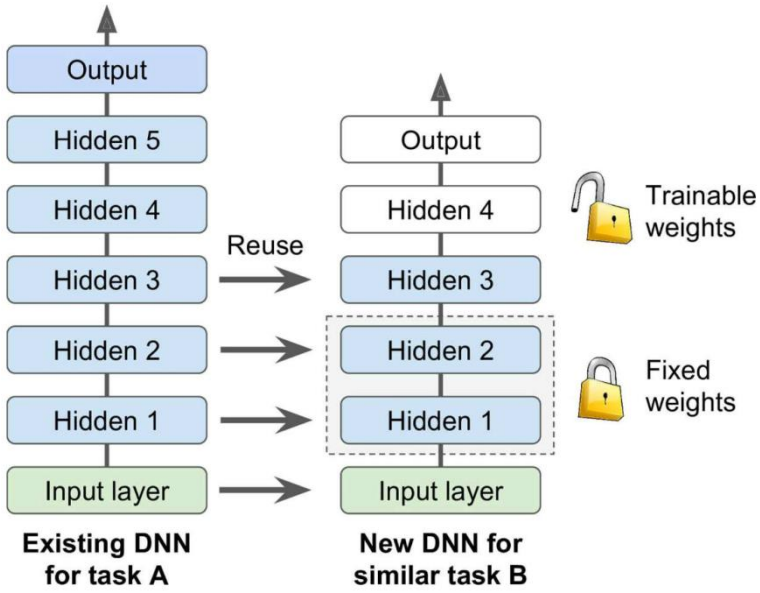
- Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep learning (MIT press, 2016).
- http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Reusing Pretrained Layers

Transfer Learning

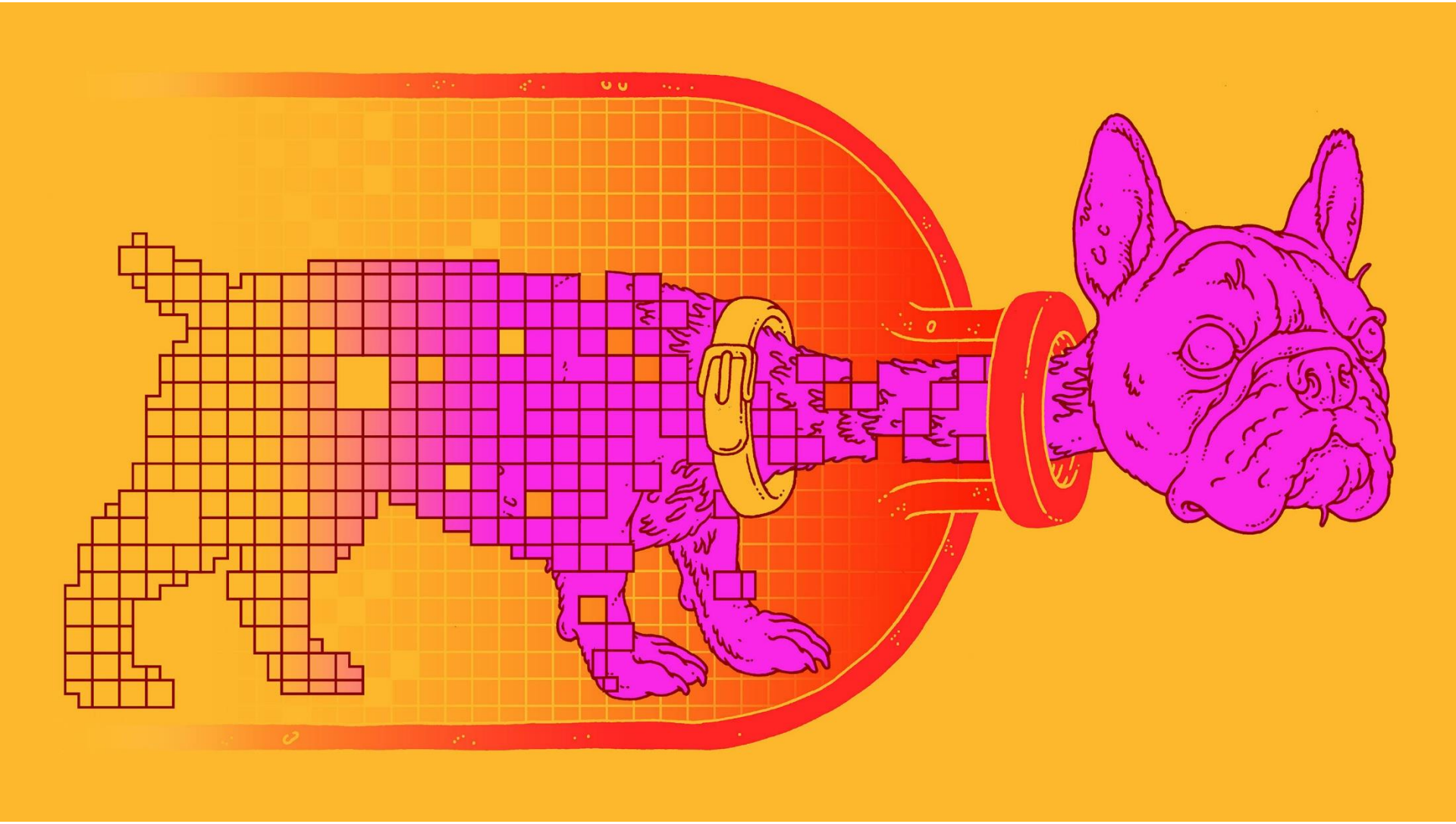
- **Training entire Deep NN takes lots of time and data**
- **It is possible to retrain only some layers → faster and less data required**
  - Re-use lower layers
  - Retrain upper layers
- **Input features must be the same (e.g. same size of images)**
- **Example**
  - Goal: Deep NN for vehicle classification
  - Available: Fully trained Deep NN for animal classification
  - Solution: Use lower layers for animal classification, retrain upper layers for vehicle classification. Resize input images if necessary.



Aurélien Géron, Hands on Machine Learning with scikit-learn and Tensorflow (O'Reilly Media, 2017).

Bottleneck

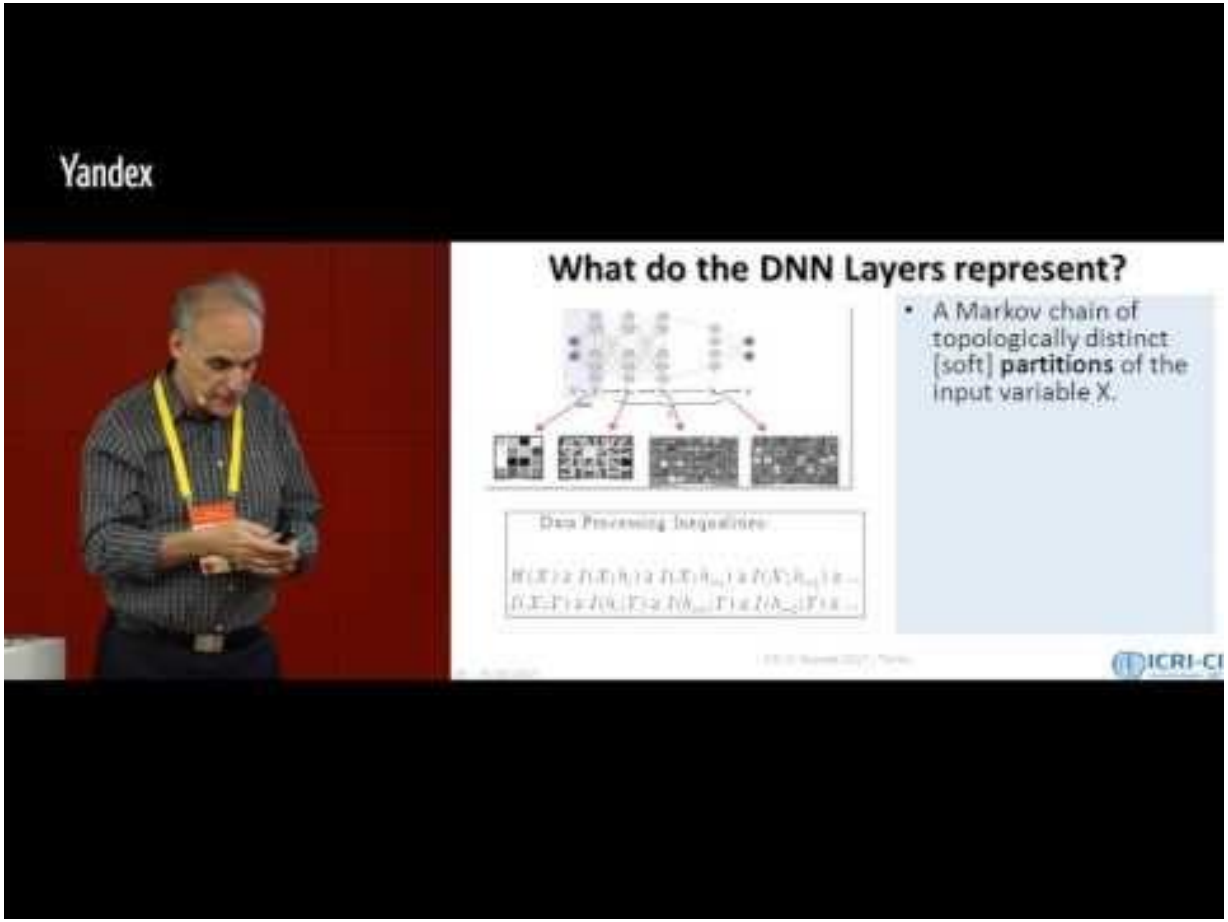
Information Theory of Deep Learning. Naftali Tishby



https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921

Information Theory of Deep Learning. Naftali Tishby

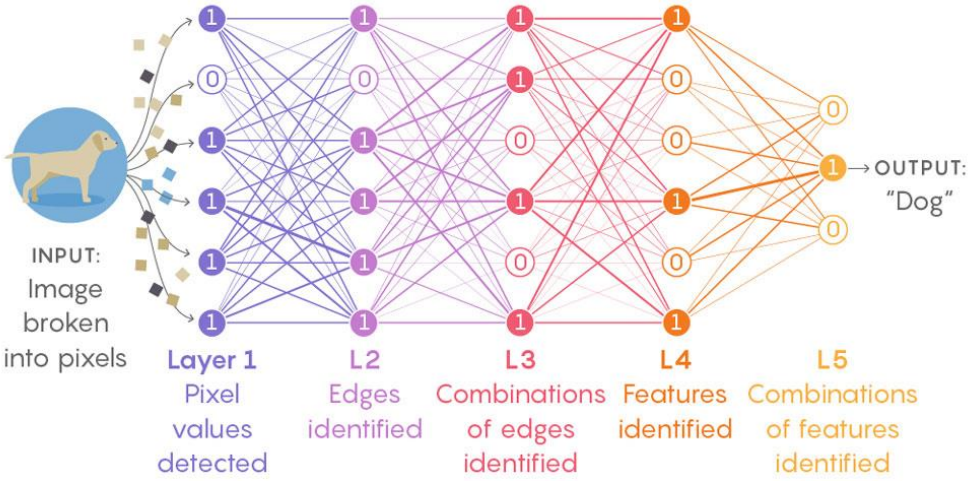
- **https://www.youtube.com/watch?v=bLqJHjXihK8**



Information Theory of Deep Learning. Naftali Tishby

Learning From Experience

Deep neural networks learn by adjusting the strengths of their connections to better convey input signals through multiple layers to neurons associated with the right general concepts.

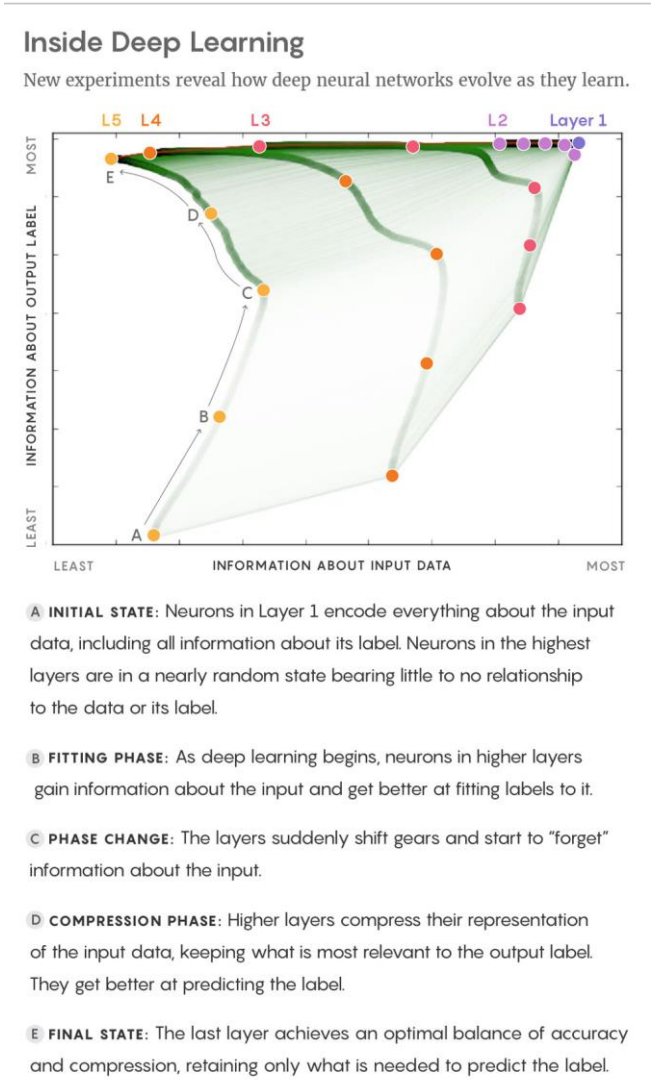


When data is fed into a network, each artificial neuron that fires (labeled "1") transmits signals to certain neurons in the next layer, which are likely to fire if multiple signals are received. The process filters out noise and retains only the most relevant features.

https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921

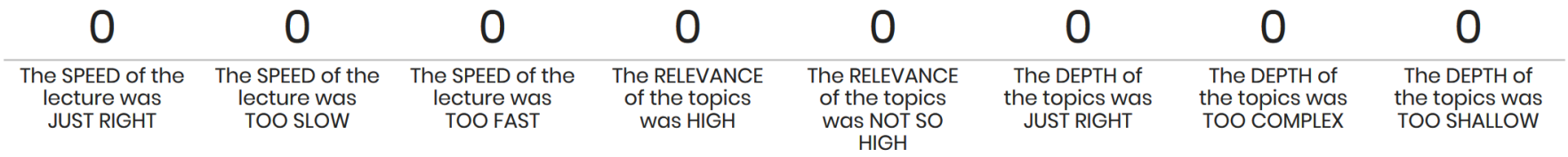
Information Theory of Deep Learning. Naftali Tishby

"In their experiments, Tishby and Shwartz-Ziv tracked how much information each layer of a deep neural network retained about the input data and how much information each one retained about the output label. The scientists found that, layer by layer, the networks converged to the information bottleneck theoretical bound: a theoretical limit derived in Tishby, Pereira and Bialek's original paper that represents the absolute best the system can do at extracting relevant information. At the bound, the network has compressed the input as much as possible without sacrificing the ability to accurately predict its label. […] Tishby and Shwartz-Ziv also made the intriguing discovery that deep learning proceeds in two phases: a short "fitting" phase, during which the network learns to label its training data, and a much longer "compression" phase, during which it becomes good at generalization, as measured by its performance at labeling new test data."



https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921

Lecture Evaluation



Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Thank you