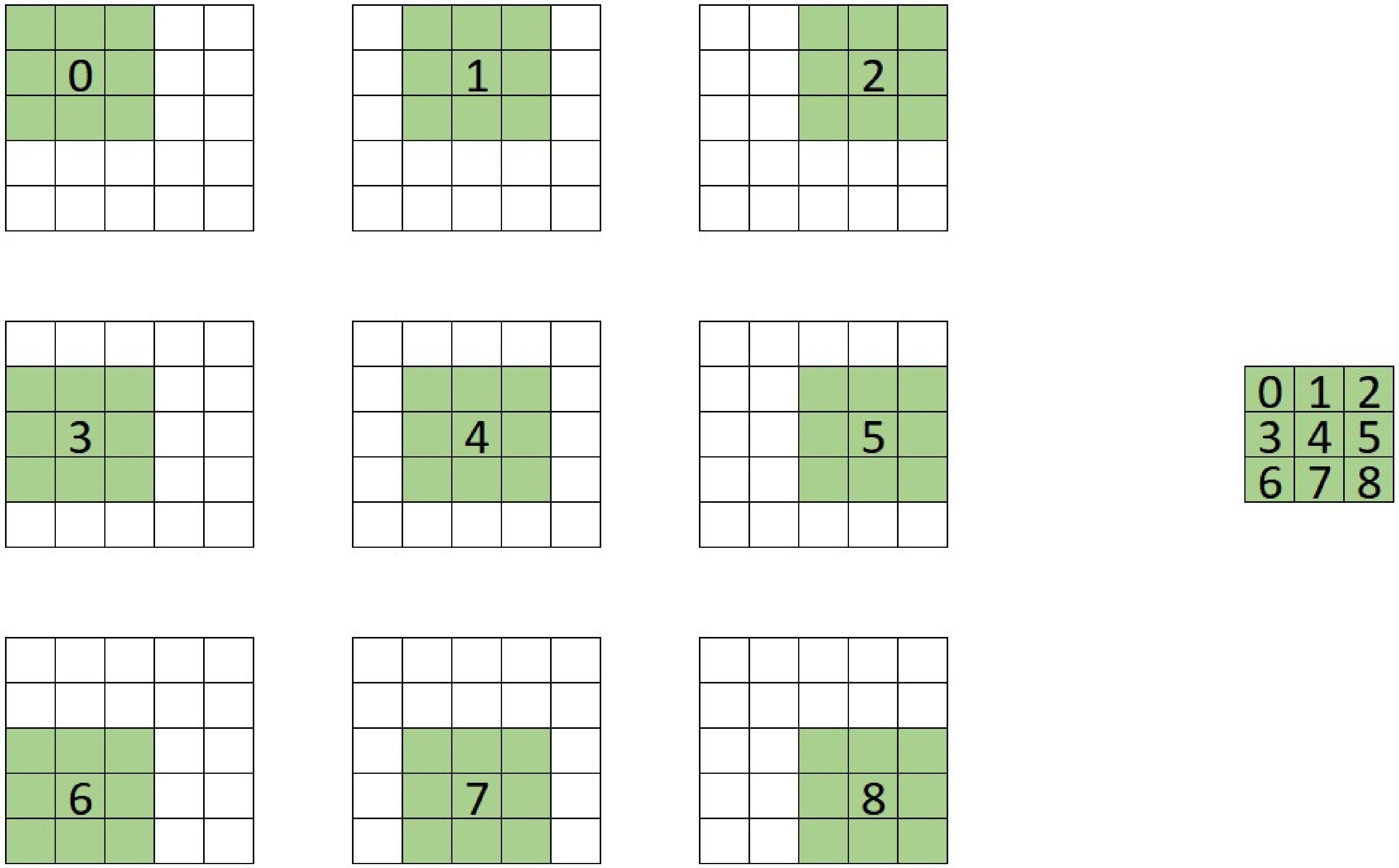
Igor Vustianiuk, 2018

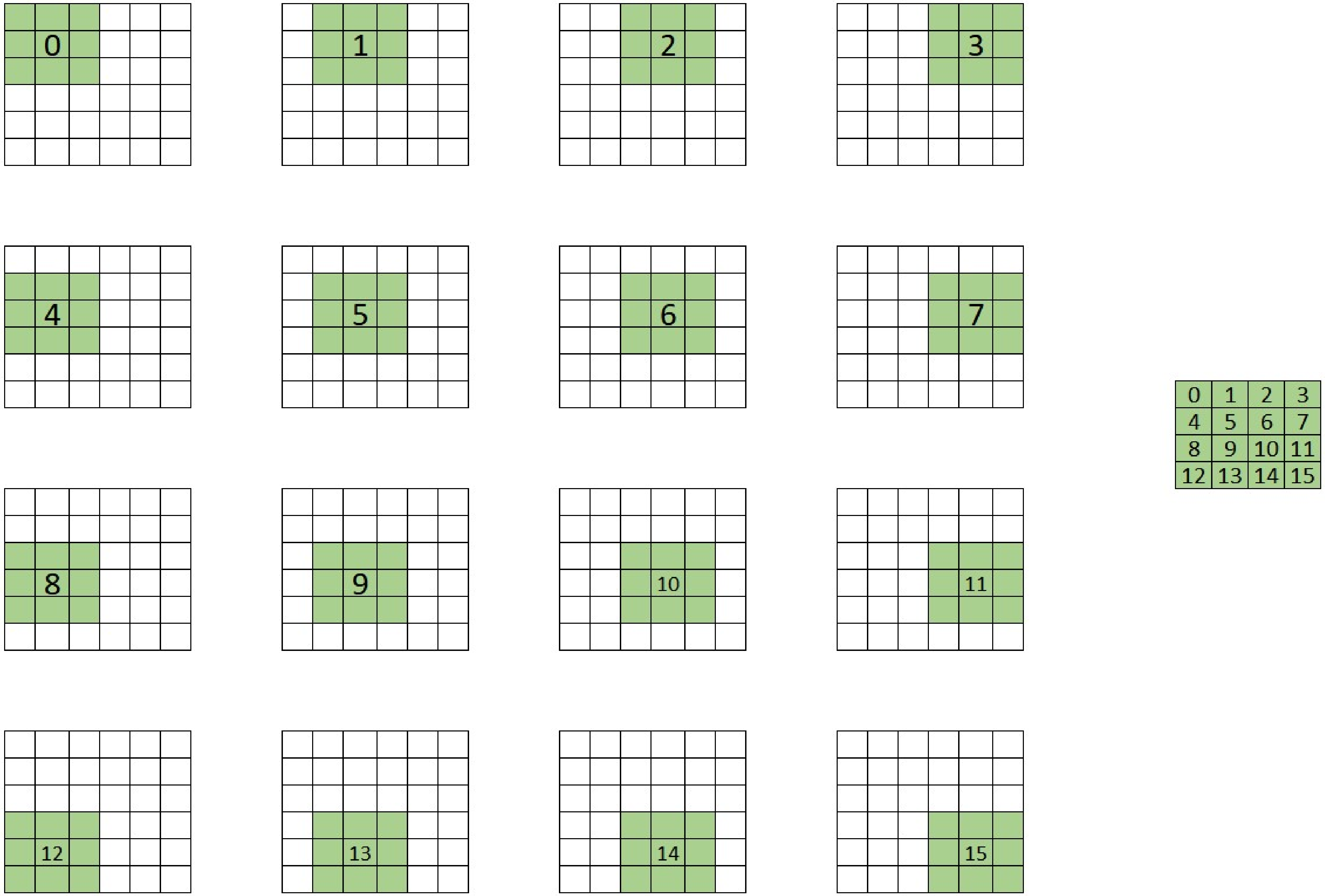
Anatomy and Arithmetic of Convolutional Neural Networks Part II

Odessa, 2018

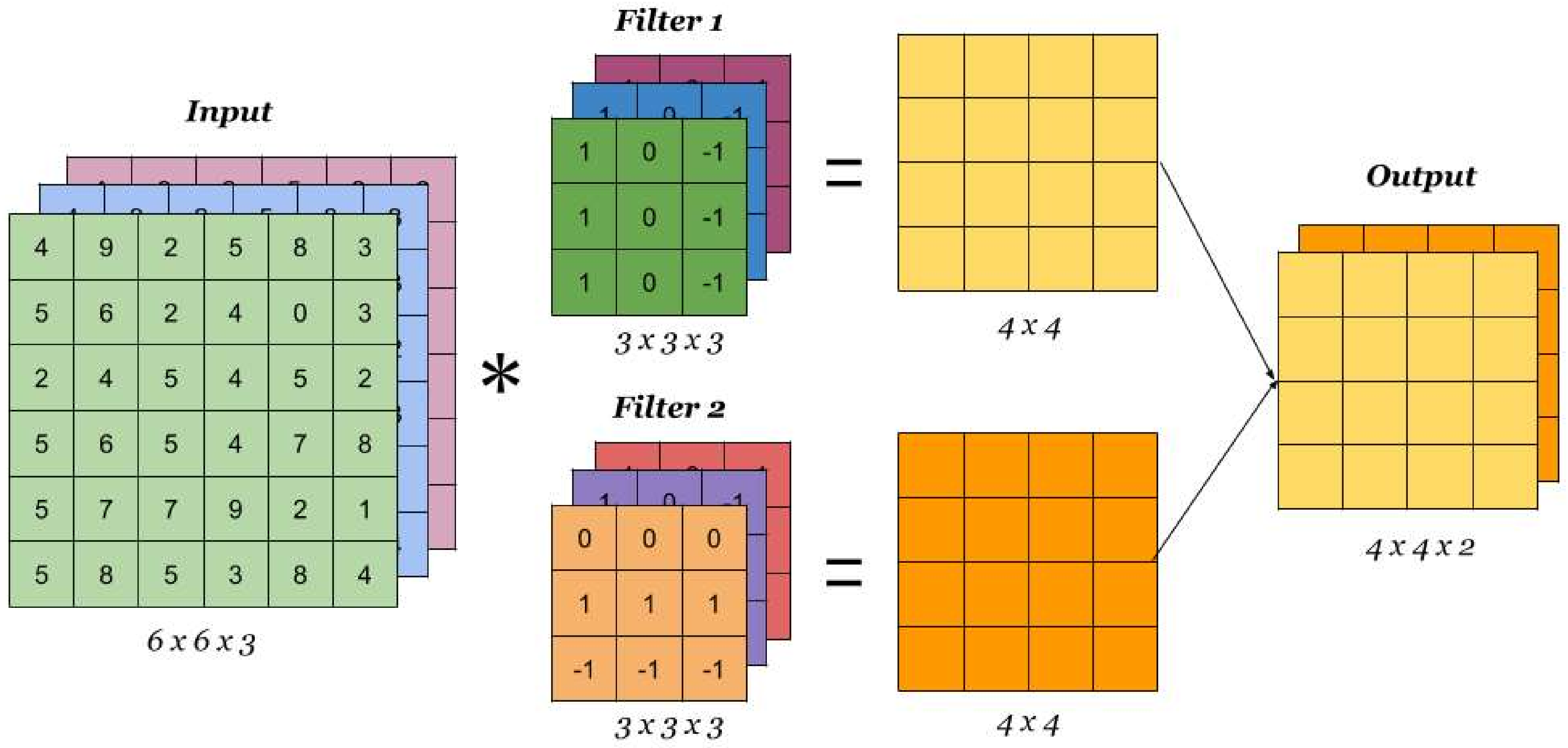
Igor Vustianiuk, 2018



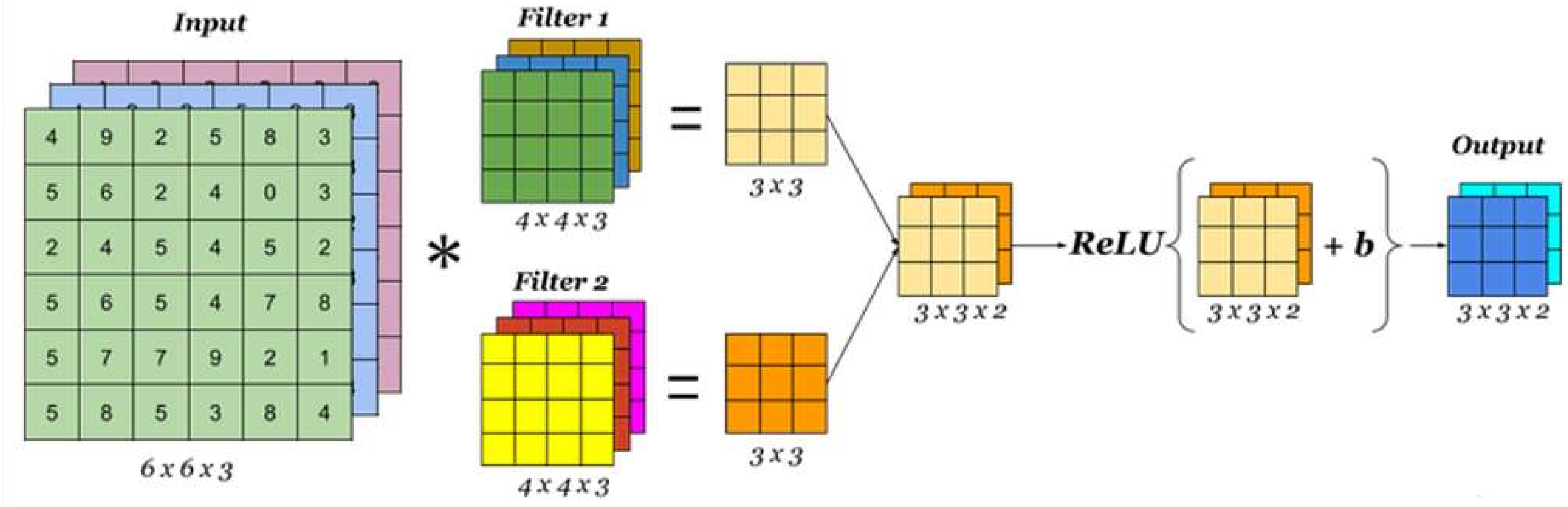
Igor Vustianiuk, 2018



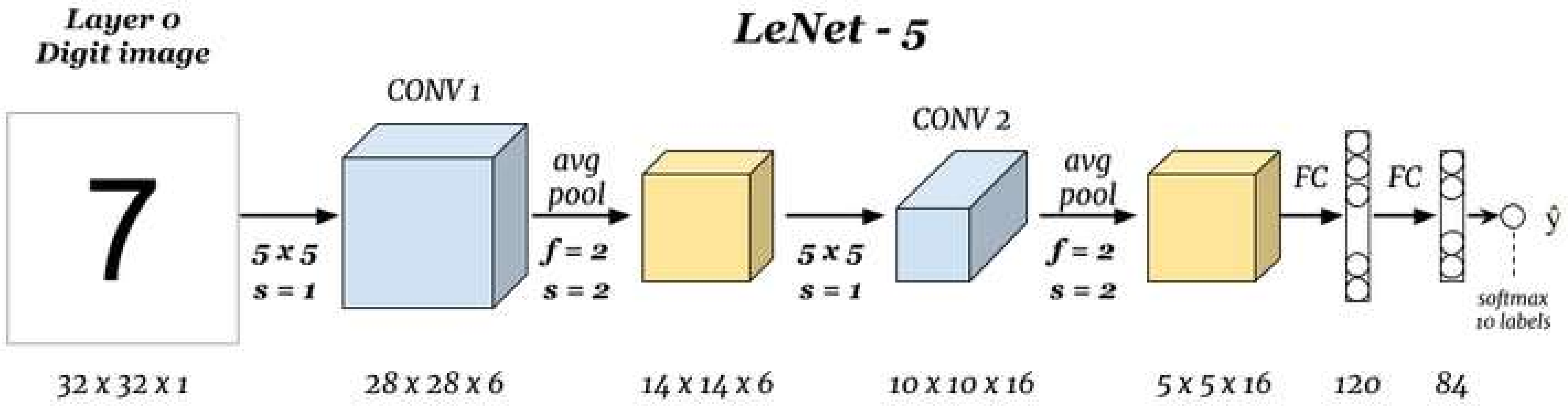
Igor Vustianiuk, 2018



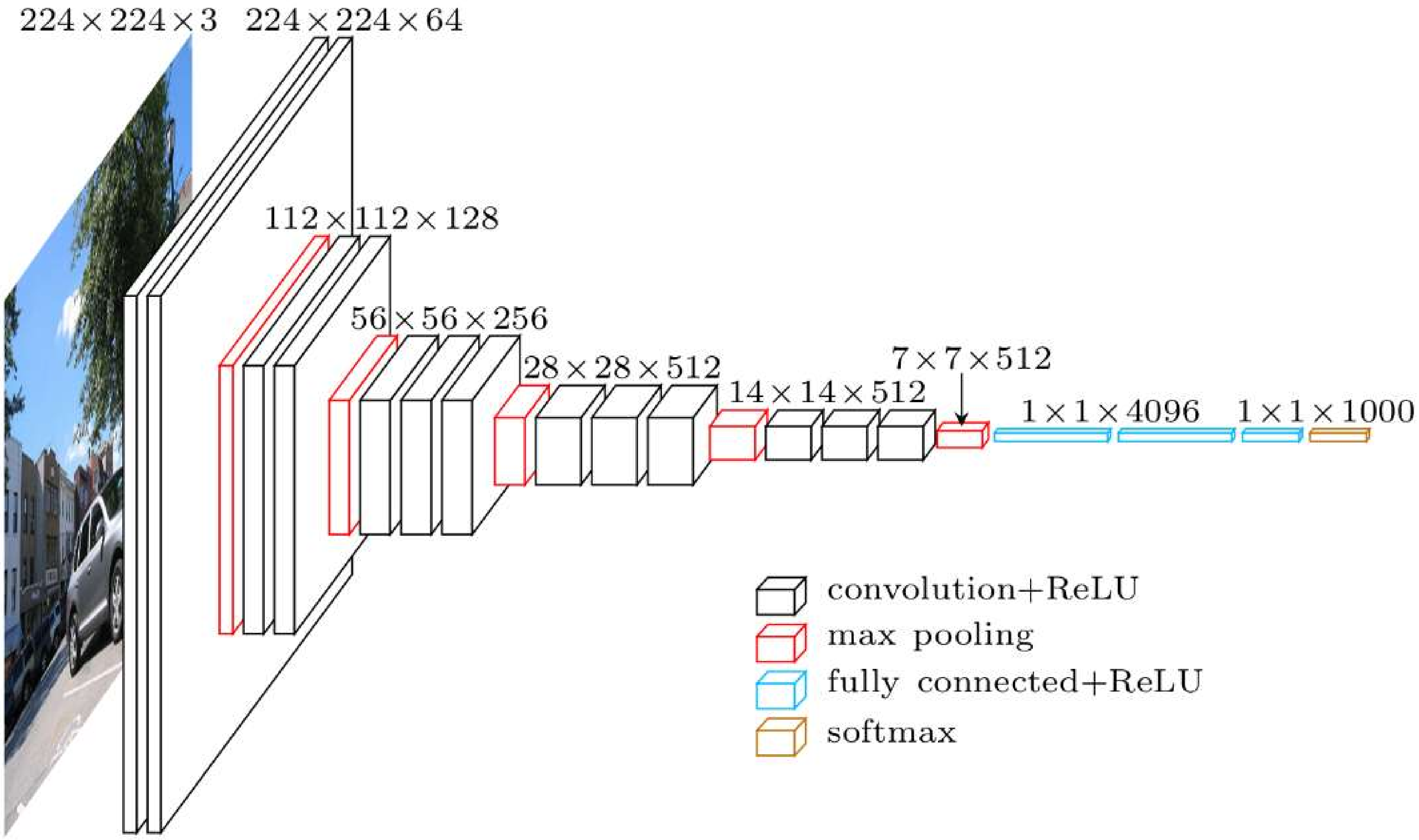
(indoml.com)



(indoml.com)



(indoml.com)

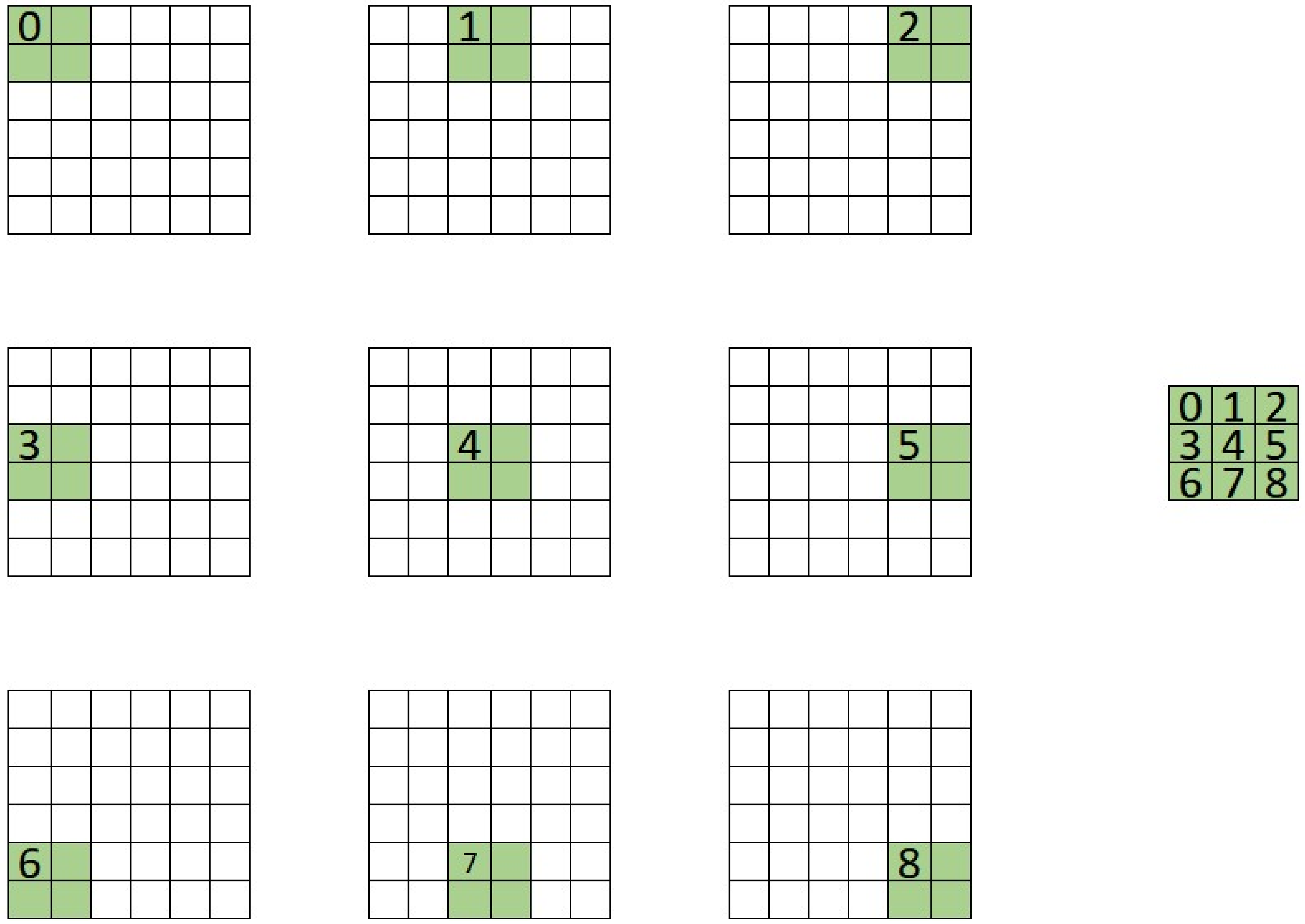


(heuritech.files.wordpress.com)

Igor Vustianiuk, 2018

Strides – 1

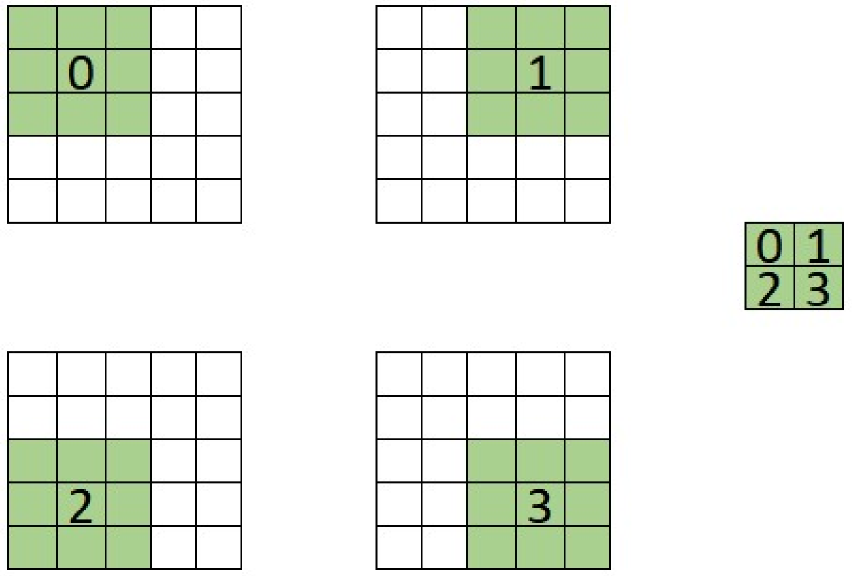
$$n = 6, \qquad f = 2, \qquad s = 2$$



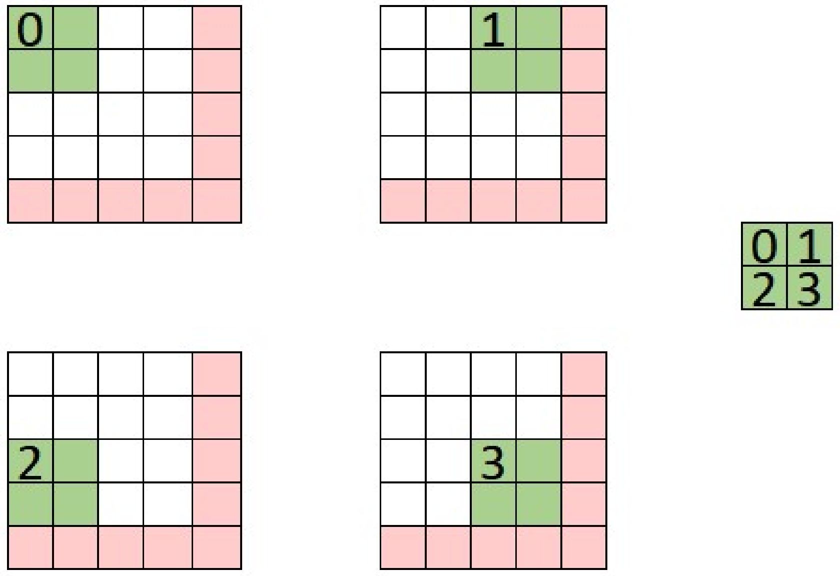
Igor Vustianiuk, 2018

Strides – 2

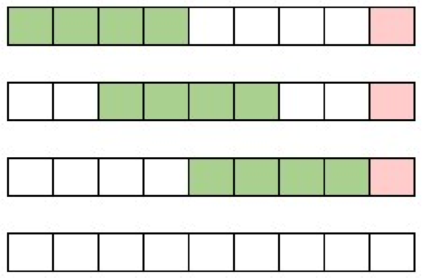
$n = 5, \quad f = 3, \quad s = 2$



$n = 5, \quad f = 2, \quad s = 2$



$n = 9, \quad f = 4, \quad s = 2$



0	$\cdots$	$\cdots$	$n-1$

Possible starting positions: $0,\ s,\ \ldots,\ is,\ \ldots$
For any i kernel “lies” over the following indexes: $is, \ldots, is+f-1$

Minimum available index is 0 so $0 \le is \quad \Longrightarrow \quad 0 \le i$

Maximum available index is $n-1$ so

$$is+f-1 \le n-1 \quad \Longrightarrow \quad is+f \le n \quad \Longrightarrow \quad is \le n-f \quad \Longrightarrow$$

$$i \le \frac{n-f}{s} \quad \Longrightarrow \quad i \le \left\lfloor \frac{n-f}{s} \right\rfloor$$

Number of “filterings” equals number of possible values of i:

$$0 \le i \le \left\lfloor \frac{n-f}{s} \right\rfloor$$

$$CNT = \left\lfloor \frac{n-f}{s} \right\rfloor + 1$$

Padding – 1

$$CNT = \left\lfloor \frac{n-f}{s} \right\rfloor + 1$$

- Size might decrease too fast:

$$n = 32, \qquad f = 5, \qquad s = 2$$

$$32 \longmapsto \left\lfloor \frac{32-5}{2} \right\rfloor + 1 = 14 \longmapsto \left\lfloor \frac{14-5}{2} \right\rfloor + 1 = 5 \longmapsto \left\lfloor \frac{5-5}{2} \right\rfloor + 1 = 1$$

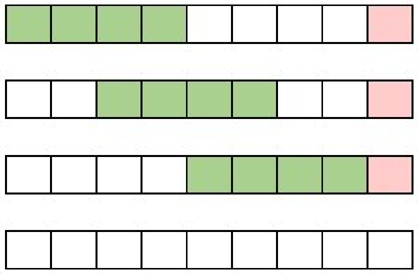
only 3 steps ☹

- Border information might be lost

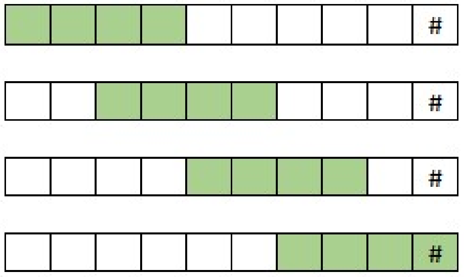
Igor Vustianiuk, 2018

Padding – 2

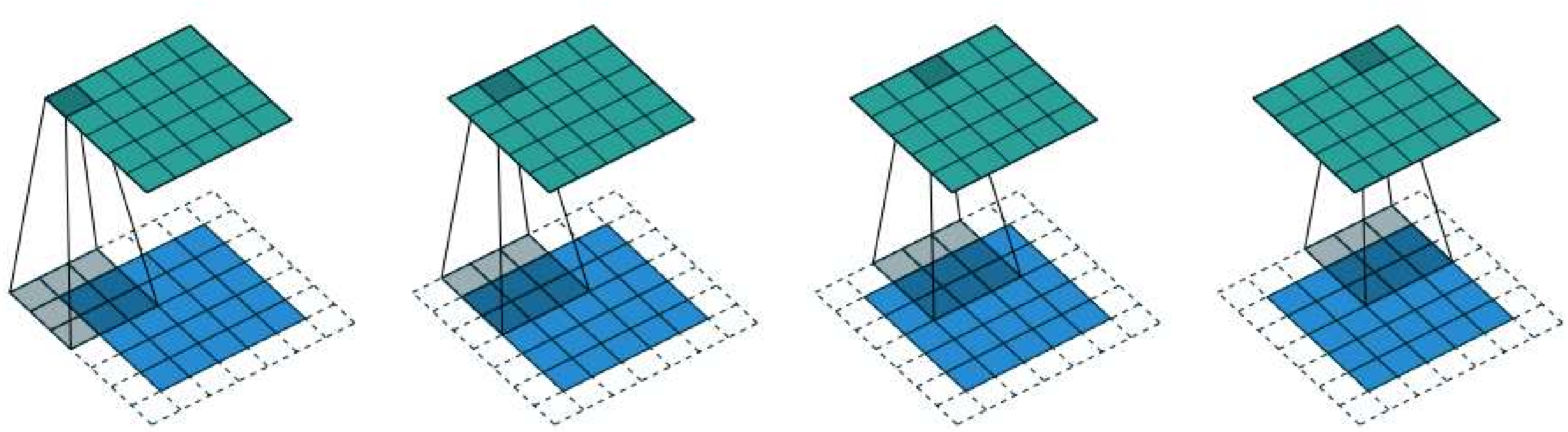
$n = 9, \quad f = 4, \quad s = 2, \quad p_1 = 0, \quad p_2 = 0$



$n = 12, \quad f = 4, \quad s = 2, \quad p_1 = 0, \quad p_2 = 1$



Padding – 3



(https://arxiv.org/abs/1603.07285)

$$n^* = n + p_1 + p_2$$

$$n \quad \longmapsto \quad \left\lfloor \frac{n^* - f}{s} \right\rfloor + 1 = \left\lfloor \frac{n + p_1 + p_2 - f}{s} \right\rfloor + 1$$

Padding – 4

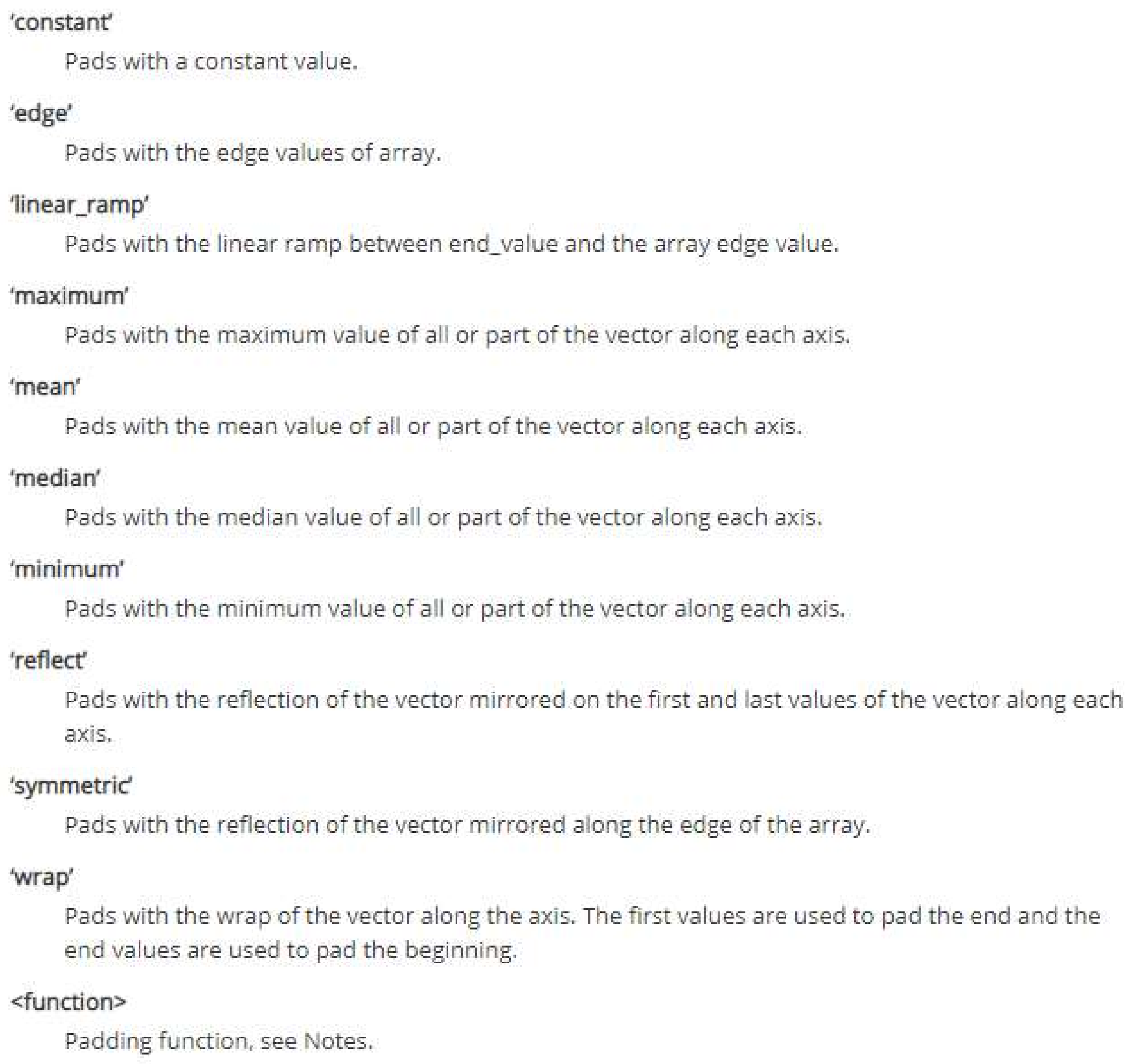
- ‘same’ padding in basic case ($s = 1,\ f \,\%\, 2 == 1$):

$$\left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1 = n$$
$$n + 2p - f = n - 1$$
$$2p = f - 1$$
$$p = \frac{f - 1}{2}$$

- ‘same’ padding in general case:

$$\left\lfloor \frac{n + p_1 + p_2 - f}{s} \right\rfloor + 1 = n$$
$$n + p_1 + p_2 - f = (n - 1)s$$
$$A \coloneqq (n - 1)s - n + f$$
$$A \,\%\, 2 == 0 \quad \longrightarrow \quad p_1 = p_2 = \frac{A}{2}$$
$$A \,\%\, 2 == 1 \quad \longrightarrow \quad p_1 = \lfloor A/2 \rfloor, \qquad p_2 = \lfloor A/2 \rfloor + 1$$

Padding – 5



'constant'

Pads with a constant value.

'edge'

Pads with the edge values of array.

'linear_ramp'

Pads with the linear ramp between end_value and the array edge value.

'maximum'

Pads with the maximum value of all or part of the vector along each axis.

'mean'

Pads with the mean value of all or part of the vector along each axis.

'median'

Pads with the median value of all or part of the vector along each axis.

'minimum'

Pads with the minimum value of all or part of the vector along each axis.

'reflect'

Pads with the reflection of the vector mirrored on the first and last values of the vector along each axis.

'symmetric'

Pads with the reflection of the vector mirrored along the edge of the array.

'wrap'

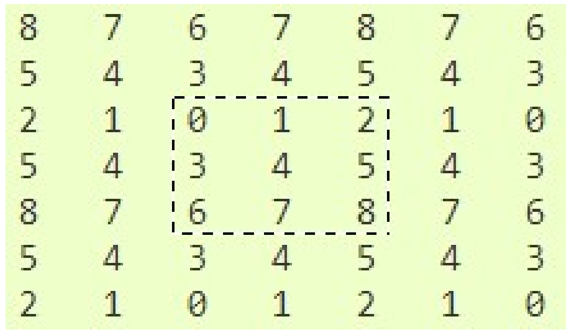
Pads with the wrap of the vector along the axis. The first values are used to pad the end and the end values are used to pad the beginning.

<function>

Padding function, see Notes.

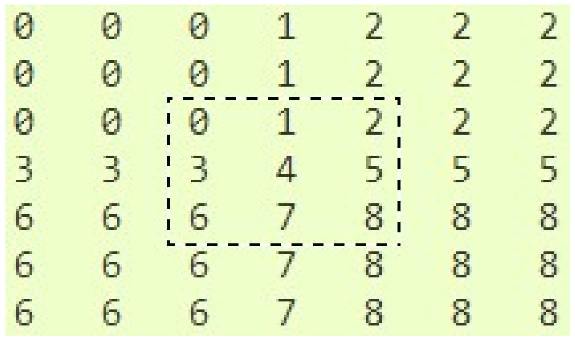
Padding – 6

- Constant: obvious ☺
- Reflection:



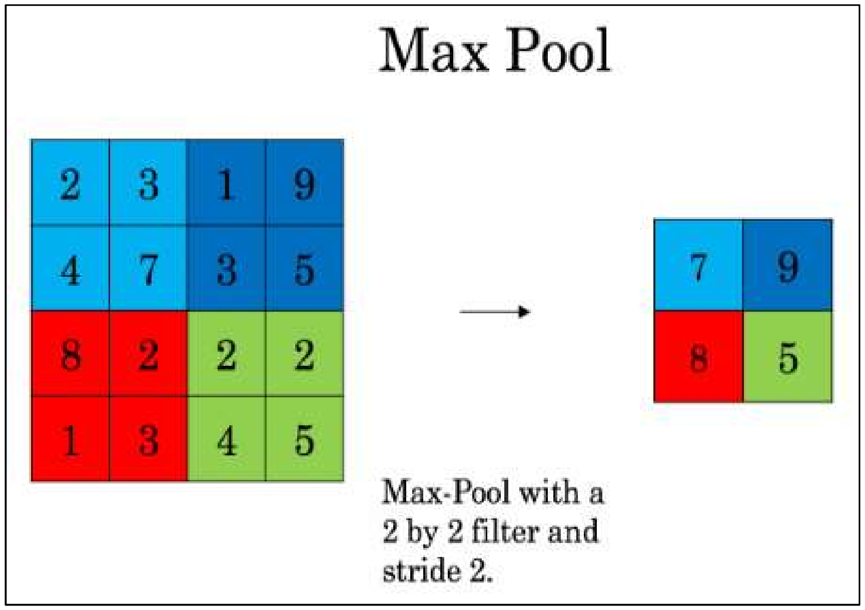
8	7	6	7	8	7	6
5	4	3	4	5	4	3
2	1	0	1	2	1	0
5	4	3	4	5	4	3
8	7	6	7	8	7	6
5	4	3	4	5	4	3
2	1	0	1	2	1	0

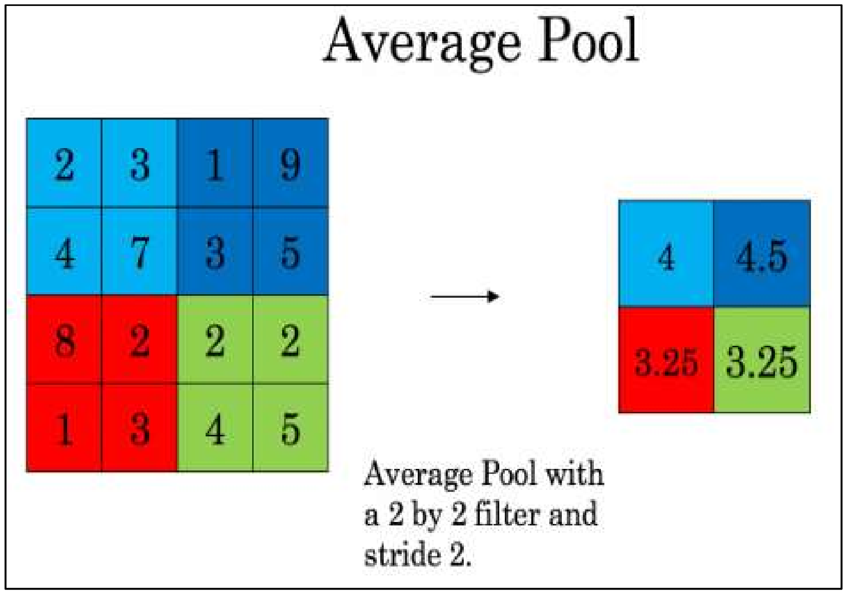
- Replication:



0	0	0	1	2	2	2
0	0	0	1	2	2	2
0	0	0	1	2	2	2
3	3	3	4	5	5	5
6	6	6	7	8	8	8
6	6	6	7	8	8	8
6	6	6	7	8	8	8

Pooling – 1





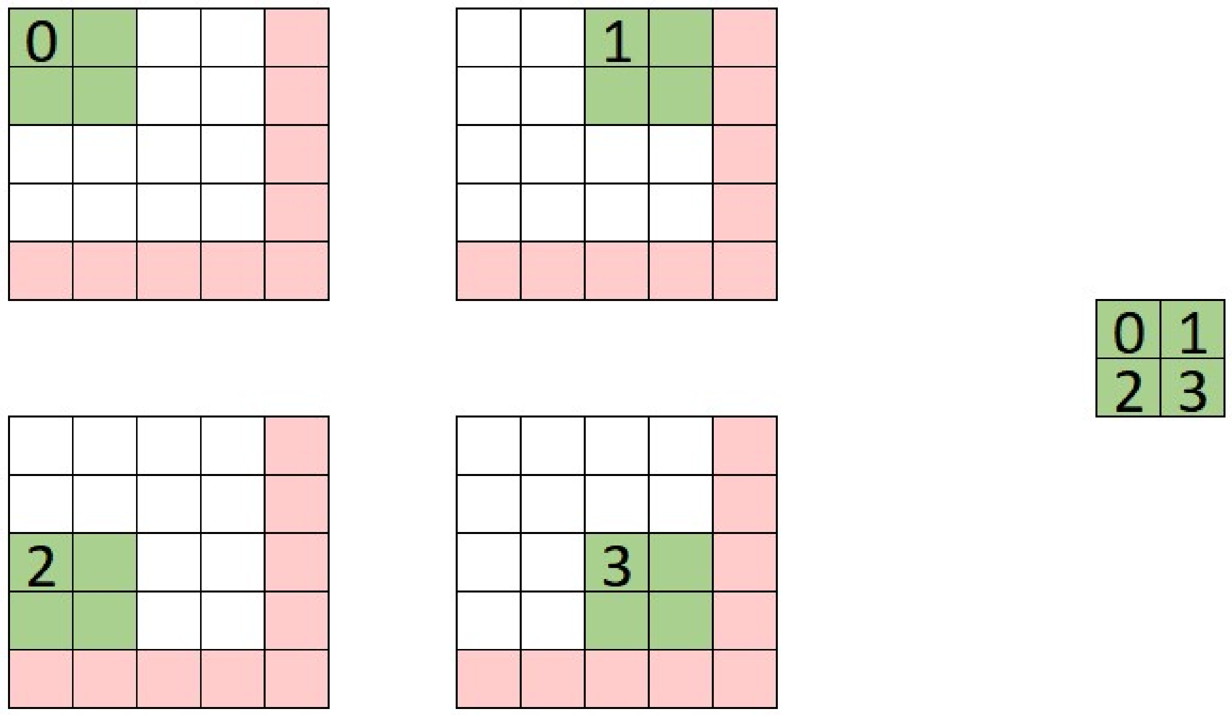
(Coursera)

Usually no padding: $p = 0$

$$n \longrightarrow \left\lfloor \frac{n-f}{s} \right\rfloor + 1$$

Igor Vustianiuk, 2018

Pooling – 2

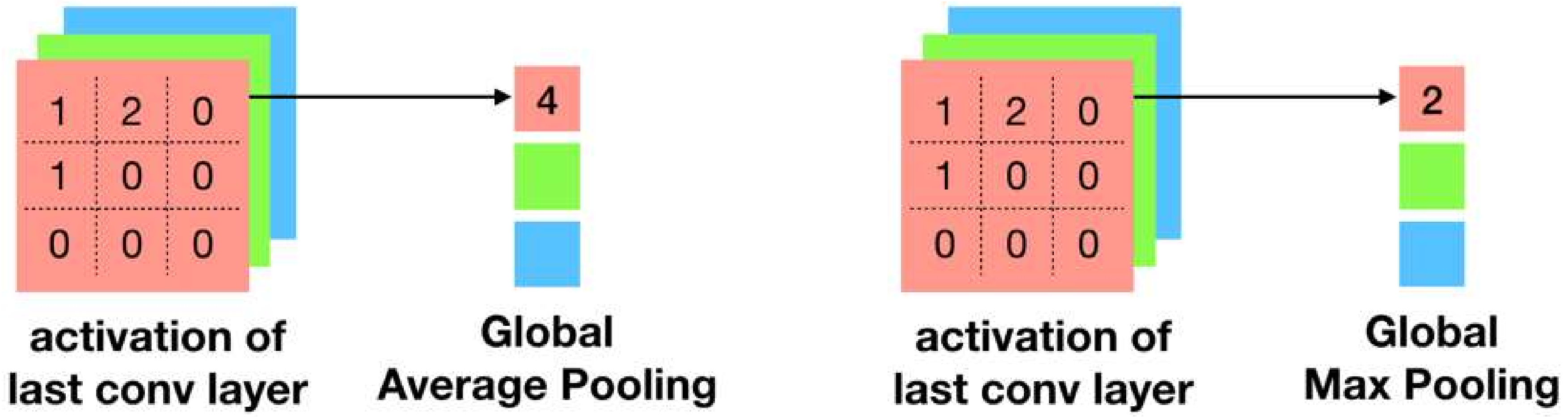


Pooling – 3

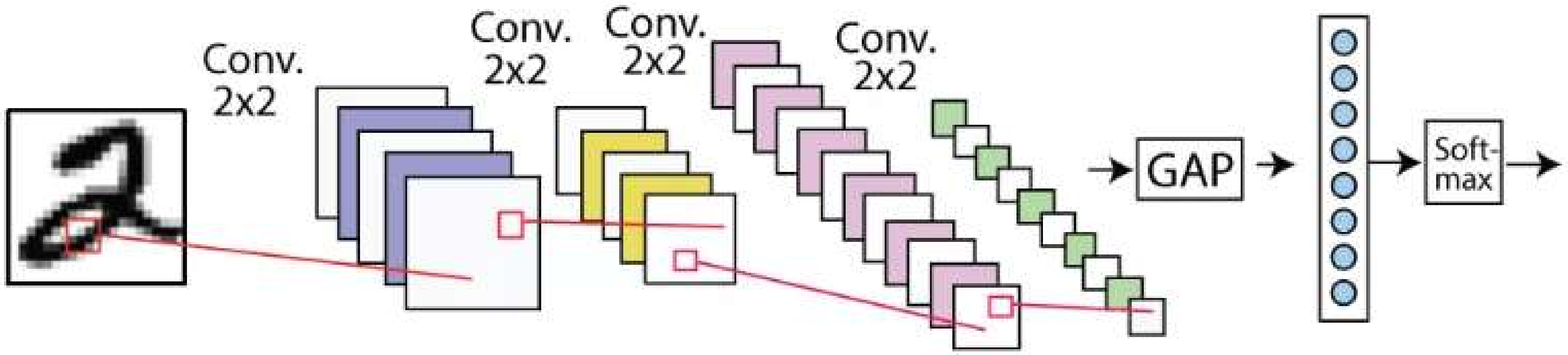
Stanford recommendations (2016):

- max-pool works better than avg-pool for general convolutional layers
- common parameters:
  - square filter of size $(2, 2)$ or $(3, 3)$
  - strides: $s = 2$
  - no padding
- image size should be divisible by multiple times

Pooling – 4



(jsideas.net)



(principlesofdeeplearning.com)

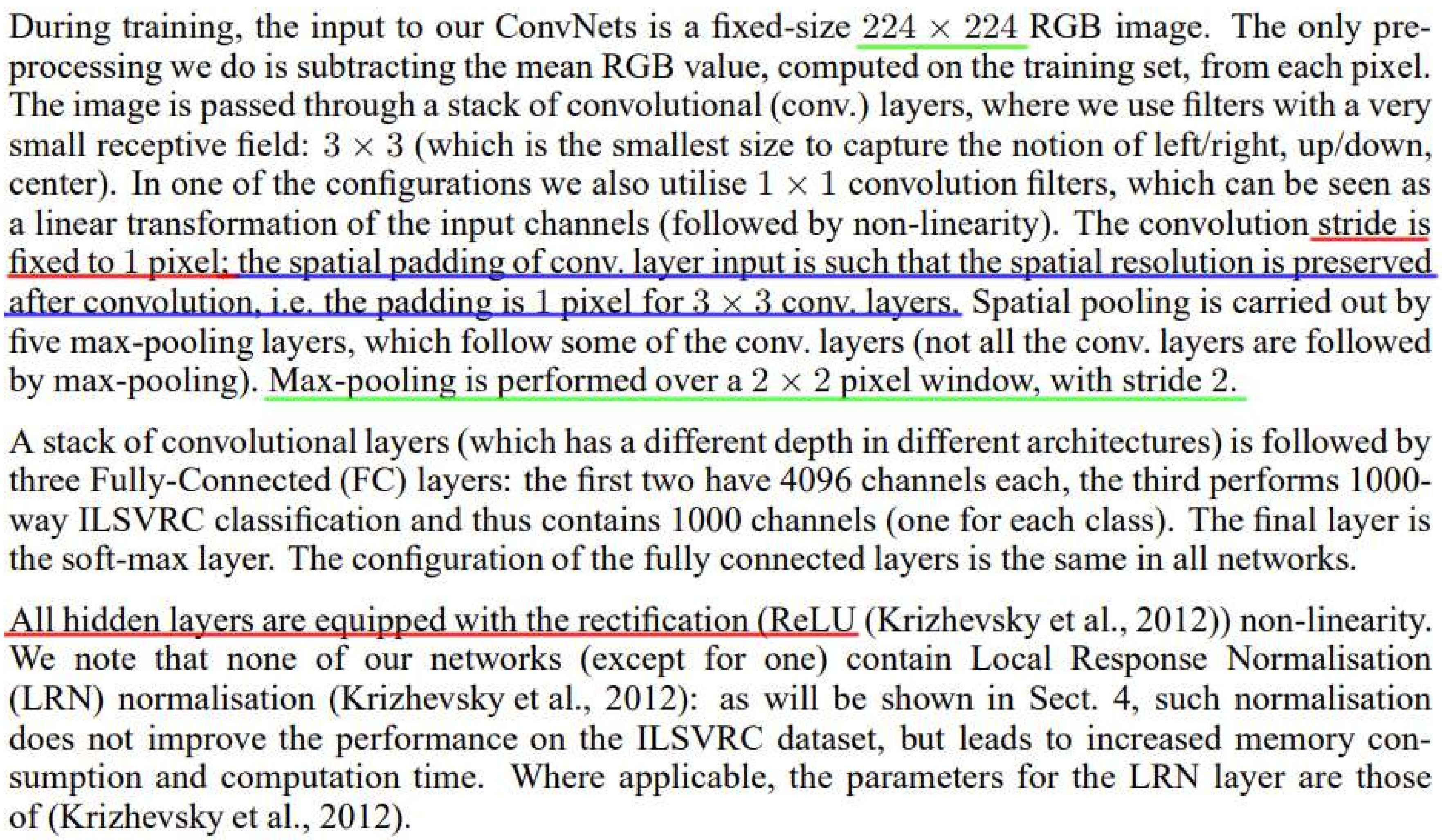
Pooling – 5

Other forms of pooling:

- p-norm Pooling: $X \to \sqrt[p]{\sum_{x \in X} x^p}$
- Fractional Pooling: https://arxiv.org/abs/1412.6071
- Spatial Pyramid Pooling: https://arxiv.org/abs/1406.4729
- Region of Interest (ROI) Pooling: https://arxiv.org/abs/1506.01497

Igor Vustianiuk, 2018

VGG-16 – 1



During training, the input to our ConvNets is a fixed-size 224×224 RGB image. The only pre-processing we do is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilise 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

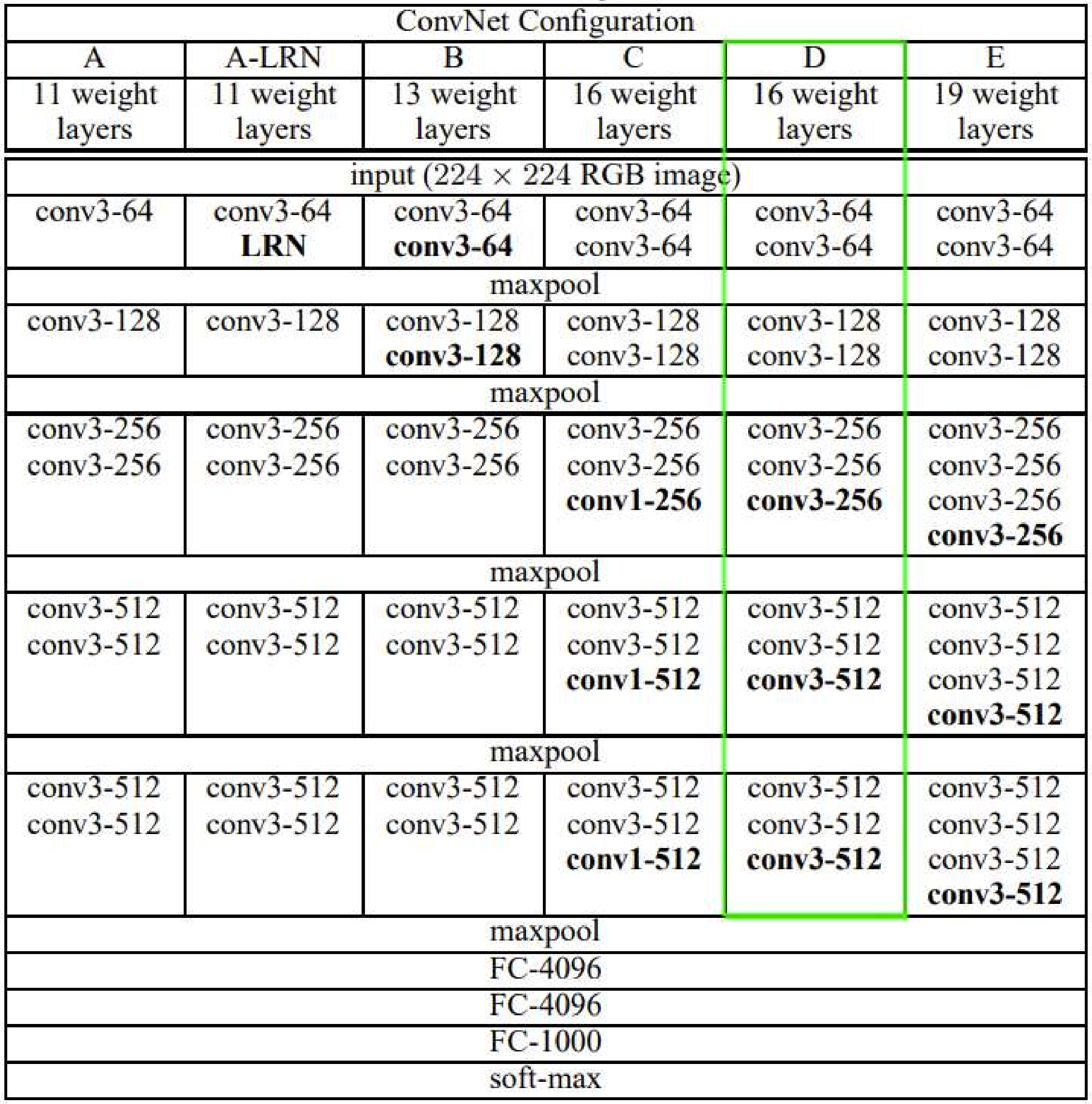
A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU (Krizhevsky et al., 2012)) non-linearity. We note that none of our networks (except for one) contain Local Response Normalisation (LRN) normalisation (Krizhevsky et al., 2012): as will be shown in Sect. 4, such normalisation does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. Where applicable, the parameters for the LRN layer are those of (Krizhevsky et al., 2012).

(https://arxiv.org/abs/1409.1556)

Igor Vustianiuk, 2018

VGG-16 – 2



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Igor Vustianiuk, 2018

VGG-16 – 3

#	# (Trainable)	Type	n (output)	n_C (output)	f	s	p	# of params
1		input	224	3				0
2	1	conv + relu	224	64	3	1	1	1 792
3	2		224					36 928
4		max-pool	112	64	2	2	0	0
5	3	conv + relu	112	128	3	1	1	73 856
6	4		112					147 584
7		max-pool	56	128	2	2	0	0
8	5	conv + relu	56	256	3	1	1	295 168
9	6							590 080
10	7							590 080
11		max-pool	28	256	2	2	0	0
12	8	conv + relu	28	512	3	1	1	1 180 160
13	9							2 359 808
14	10							2 359 808
15		max-pool	14	512	2	2	0	0
16	11	conv + relu	14	512	3	1	1	2 359 808
17	12							2 359 808
18	13							2 359 808
19		max-pool	7	512	2	2	0	0
20		flatten	$(25\,088,)$					0
21	14	fc + relu	$(4\,096,)$					102 764 544
22	15	fc + relu	$(4\,096,)$					16 781 312
23	16	fc + softmax	$(1000,)$					4 097 000
								138 357 544

VGG-16 – 4

Basic case: n_C^{out} square filters of shape $f \times f \times n_C$

$$(n_H,\ n_W,\ n_C^{in}) \quad \longmapsto \quad (n'_H,\ n'_W,\ n_C^{out})$$

$$\#\text{weights} + \#\text{biases} = n_C^{out} \cdot \left(f \cdot f \cdot n_C^{in}\right) + n_C^{out} = n_C^{out} \cdot \left(f \cdot f \cdot n_C^{in} + 1\right)$$

Some calculations:

- number of parameters for conv1: $n_C^{out} = 64,\ n_C^{in} = 3,\ f = 3$

$$64 \cdot (3 \cdot 3 \cdot 3 + 1) = 64 \cdot 28 = 1792$$

- number of parameters for conv1: $n_C^{out} = 64,\ n_C^{in} = 64,\ f = 3$

$$64 \cdot (3 \cdot 3 \cdot 64 + 1) = 64 \cdot 577 = 36\,928$$

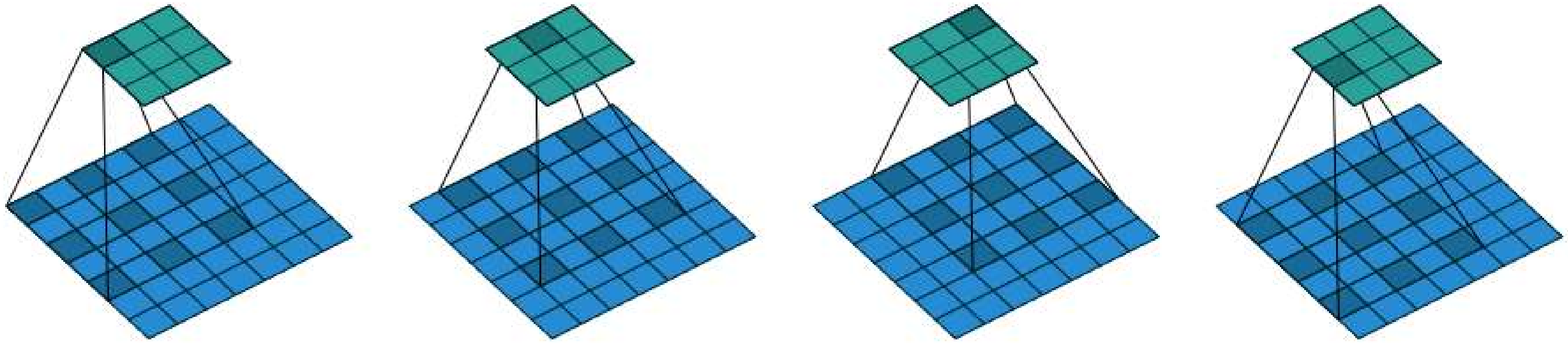
- output shape for first max-pool layer:

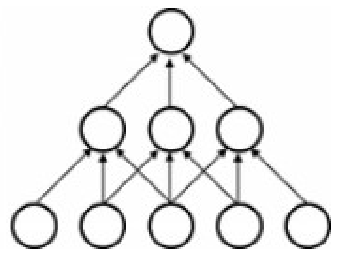
$$n = 224,\ \ f = 2,\ \ s = 2,\ \ p = 0$$

$$n' = \left\lfloor \frac{n + 2p - f}{s} \right\rfloor + 1 = \left\lfloor \frac{224 - 2}{2} \right\rfloor + 1 = 112$$

- flatten layer size: $N = 7 \cdot 7 \cdot 512 = 25\,088$
- number of parameters for fc1: $(25\,088 + 1) \cdot 4096 = 102\,764\,544$

Dilated convolutions – 1

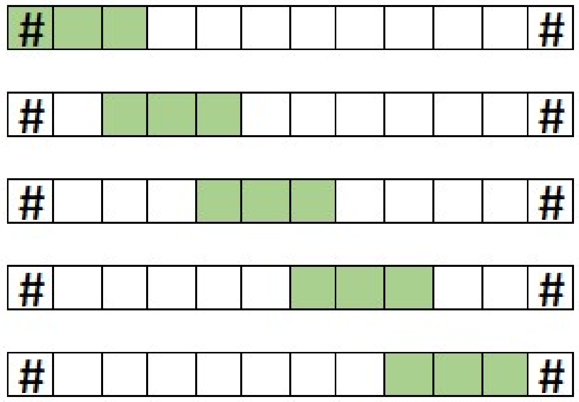
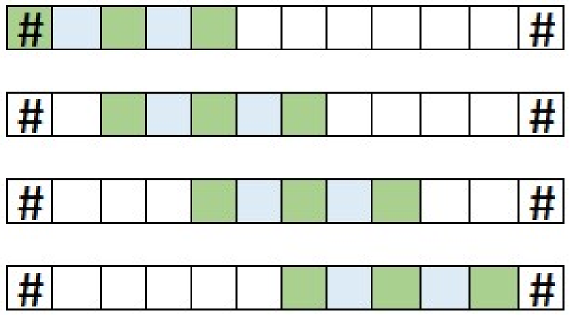

(https://arxiv.org/abs/1603.07285)



Igor Vustianiuk, 2018

Dilated convolutions – 2

New filter parameter: dilation rate d ($d = 1$ for simple convolutions).

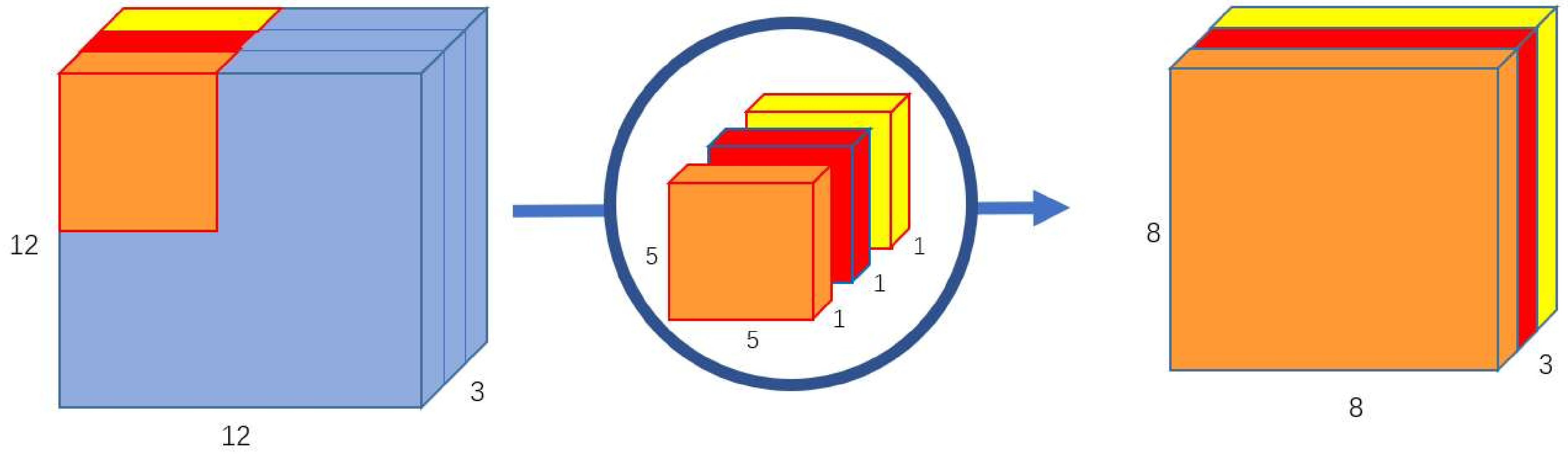
$n = 10, \qquad f = 3, \qquad s = 2, \qquad p_1 = p_2 = 1$	
$d = 1$	$d = 2$
	

$$f^* = f + (f-1)(d-1)$$

$$n \longmapsto \left\lfloor \frac{n + p_1 + p_2 - f^*}{s} \right\rfloor + 1 = \left\lfloor \frac{n + p_1 + p_2 - f - (f-1)(d-1)}{s} \right\rfloor + 1$$

Igor Vustianiuk, 2018

Depthwise convolutions



(Chi-Feng Wang on Medium)

1 x 1 Convolutions

Usual convolutions:

$$(n_H,\ \ n_W,\ \ n_C)\ \longmapsto\ (n'_H,\ \ n'_W,\ \ n'_C)$$

Typical 1 x 1 convolution:

- $s = 1$
- $p =$ 'valid': $p = \frac{f-1}{2} = 0$ (same as 'same'!)
- n'_C – the most important parameter

Applying 1 x 1 convolution:

- $(n_H,\ \ n_W,\ \ n_C)\ \longmapsto\ (n_H,\ \ n_W,\ \ n'_C)$
- Same spatial dimensions, NEW number of channels
- Don't forget about bias and nonlinearity!

Changing shape

- general convolutions: $(n_H, n_W, n_C^{in}) \longmapsto (\boldsymbol{n'_H}, \boldsymbol{n'_W}, \boldsymbol{n_C^{out}})$
- 1 x 1 convolutions with $s = 1$: $(n_H, n_W, n_C^{in}) \longmapsto (n_H, n_W, \boldsymbol{n_C^{out}})$
- depth-wise convolutions: $(n_H, n_W, n_C^{in}) \longmapsto (\boldsymbol{n'_H}, \boldsymbol{n'_W}, n_C^{in})$
- non-global pooling: $(n_H, n_W, n_C^{in}) \longmapsto (\boldsymbol{n'_H}, \boldsymbol{n'_W}, n_C^{in})$
- global pooling: $(n_H, n_W, n_C^{in}) \longmapsto (n_C^{in},)$
- flattening: $(n_H, n_W, n_C^{in}) \longmapsto (n_H \cdot n_W \cdot n_C^{in},)$

Stanford Recommendations (2017)

- Convolutional layer:
  - n_C = power of 2
  - $f = 3, \ s = 1, \ p = 1$
  - $f = 5, \ s = 1, \ p = 2$
  - $f = 5, \ s = 2, \ p =$ whatever fits
  - $f = 1, \ s = 1, \ p = 0$
- Pooling layer:
  - max-pooling performs better in inner layers
  - $f = 2, \ s = 2, \ p = 0$
  - $f = 3, \ s = 2, \ p = 0$
- "Classical" architecture (before ResNet, Inception and co):
  - [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K-(FC-SOFTMAX)
  - $N \leq 5, \ M$ is large, $0 \leq K \leq 2$
- Better fine-tune pretrained successful models than invent new from scratch