

COMPUTATIONAL INTELLIGENCE
DEEP LEARNING

Convolutional Neural Networks – CNN





Adrian Horzyk

horzyk@agh.edu.pl

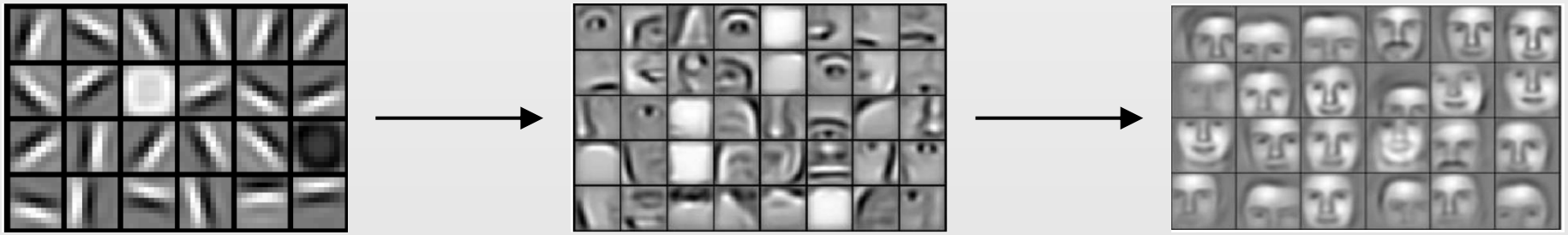


AGH University of Science and Technology
Krakow, Poland

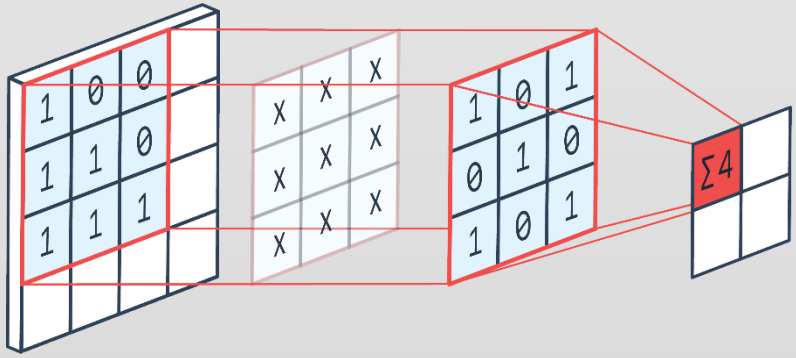
Computer Vision

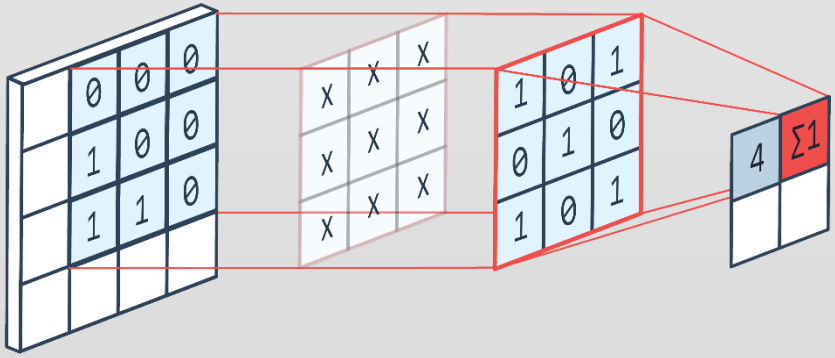


Computer vision is a group of tasks that play a very important role today and can be supported by convolutional neural networks (CNN) due to their unique ability to recognize objects whenever their location in the image:



Convolutional filters allow us to filter out and detect basic and secondary features gradually in the subsequent layers of the network using adaptive filtering (dot product) and weights of the filters trained during the CNN training process:





Filters allow the network to adjust them to recognize particular shapes and colors.

Filters and Convolutions



Filters are commonly used in computer graphics, and allow us to find edges and convolve images:

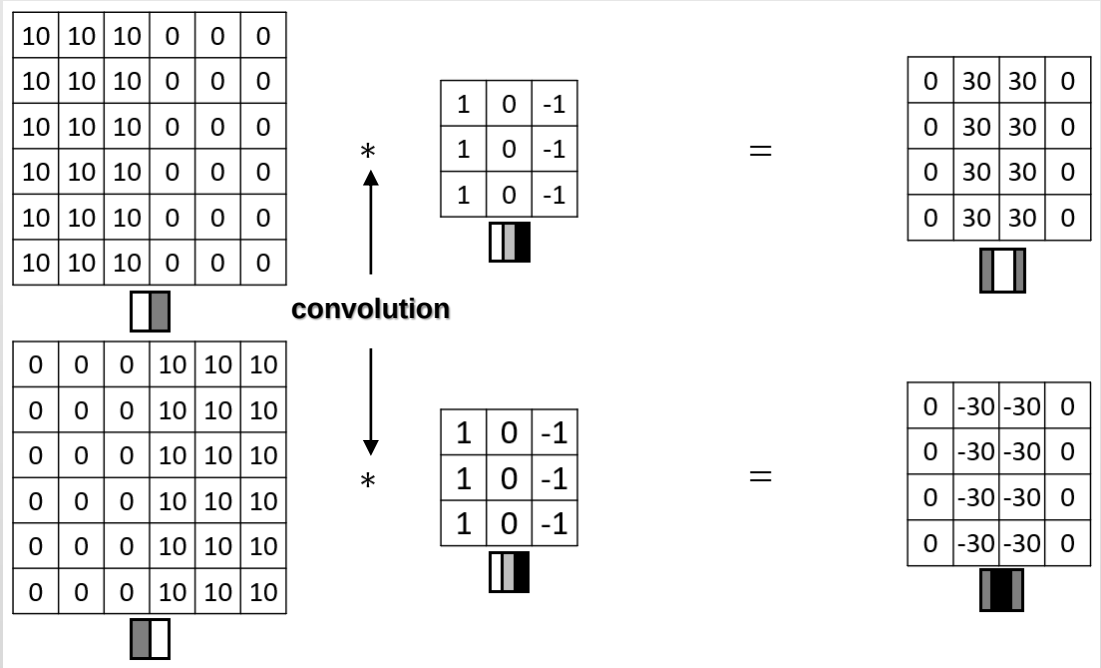
1	0	-1
1	0	-1
1	0	-1

Vertical

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

Horizontal

- **Example result of applying the vertical-line filter:**



Adaptive Filtering



In convolutional layers, we use adaptive filters, which have no constant filters but weights w_i that are adapted during the training process to represent frequent patterns of the filter size in the input images:

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

*

convolution

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

=

The output value is computed as a dot product of the input area where the filter is adapted and the filter (matrix of the adaptable weights).

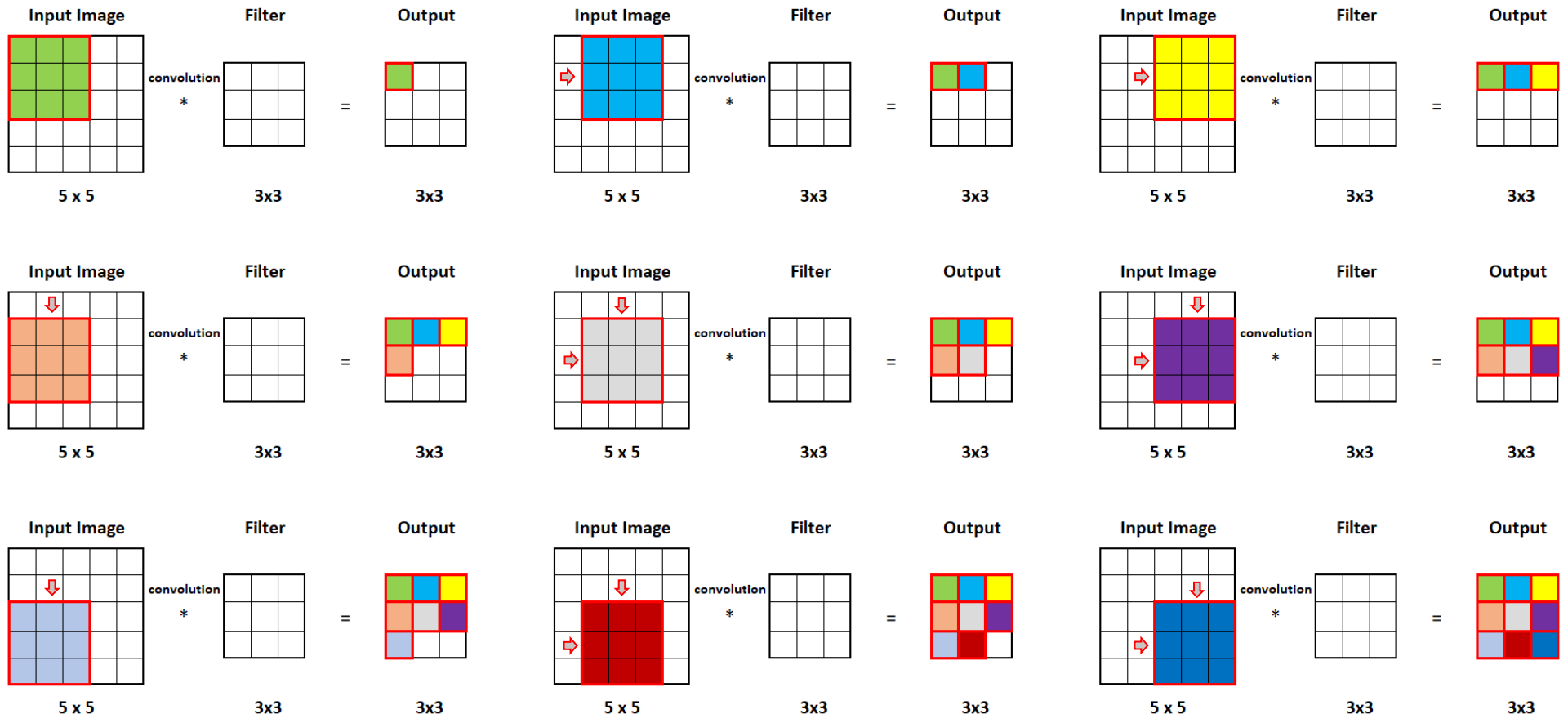
Weights are parameters of the model, so they are updated in the training process.

Stride 1



To adapt the filter to the whole image we must move the filter over the image with a given stride s that defines the number of fields (pixels) we move in vertical and/or horizontal directions (it is a hyperparameter of the model):

- **For stride 1 we jump over two pixels as presented in the figure below:**

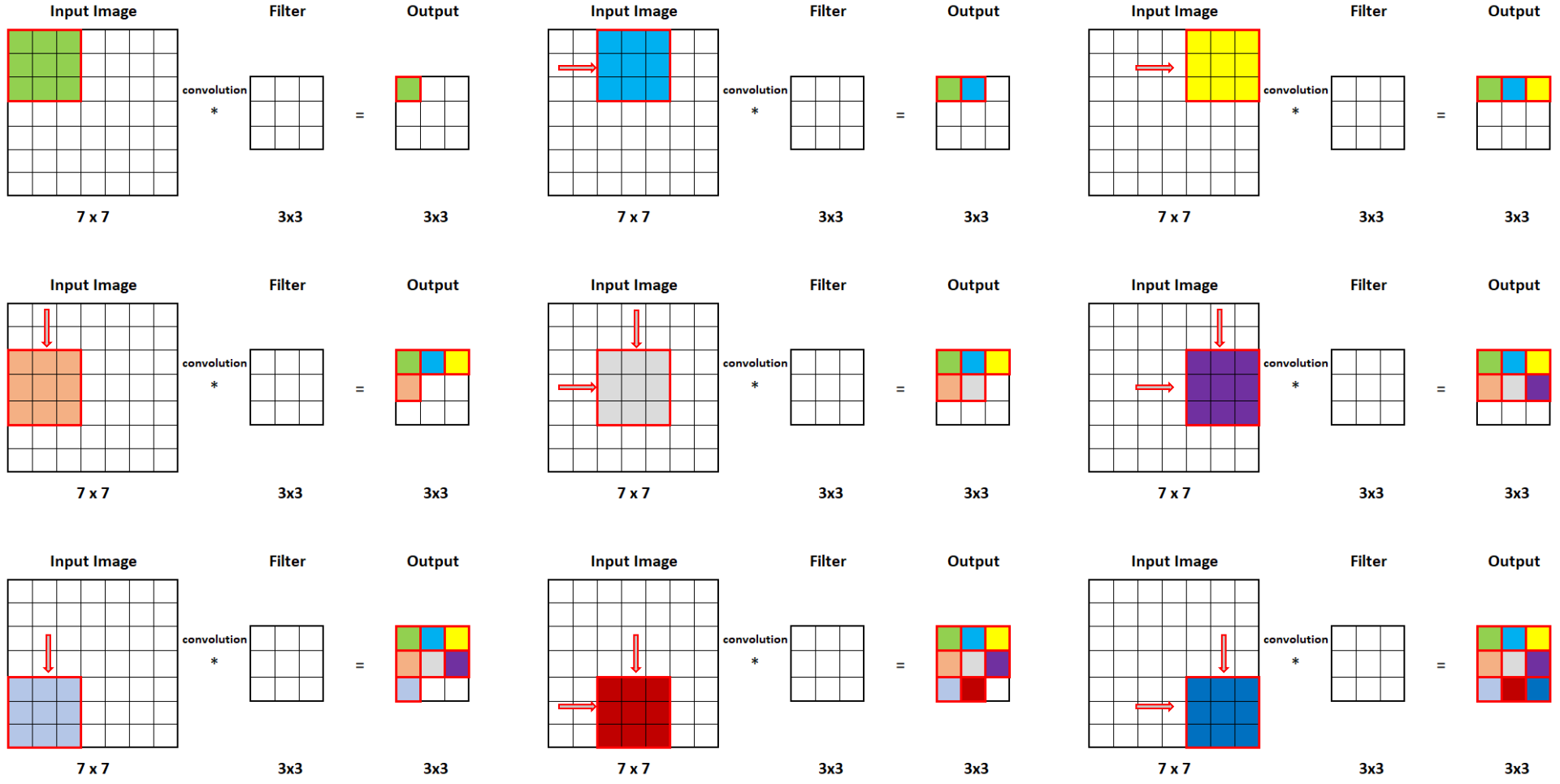


Stride 2



To adapt the filter to the whole image we must move the filter over the image with a given stride s that defines the number of fields (pixels) we move in vertical and/or horizontal directions (it is a hyperparameter of the model):

- **For stride 2 we jump over two pixels as presented in the figure below:**

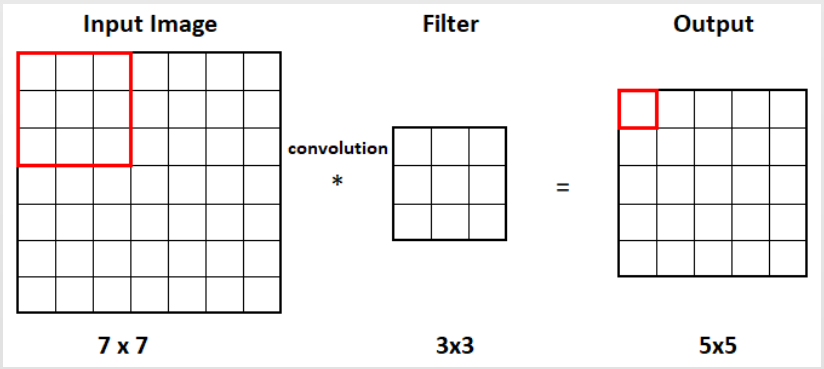


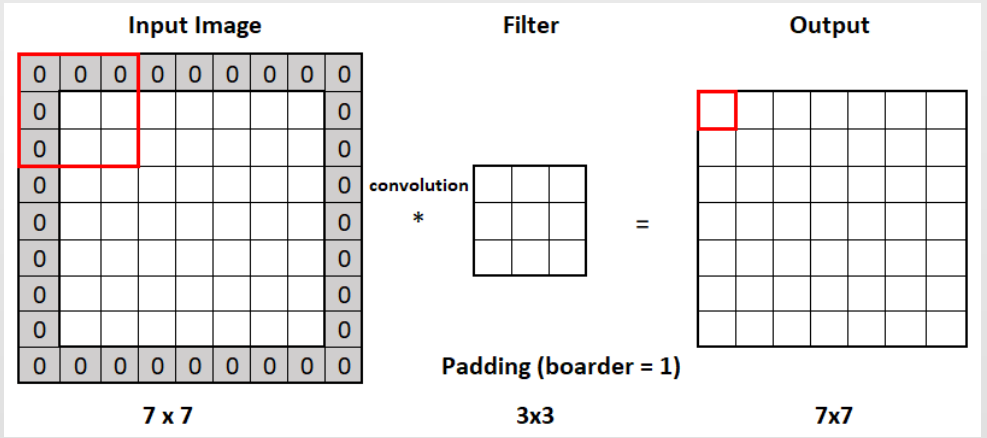
Padding



When moving the filter (f x f) over the image (n x n) with a given stride, we cannot move over the edges/boarder of the image, so we are forced to treat the pixels on boarders in the different way ("Valid") or add 0-value boarder outside the image to adapt filters on the boarders ("Same"):

- **Valid Convolution (no padding): Output size is n x n * f x f = (n – f + 1) x (n – f + 1)**





- **Same Convolution (padding is balances the filter size p = (f – 1)/2, then the output size is the same as the one of the input image.**
- **The chosen way of convolution ("same" or "valid") is one the hyperparameters of the model!**

Output volume size for stride and padding

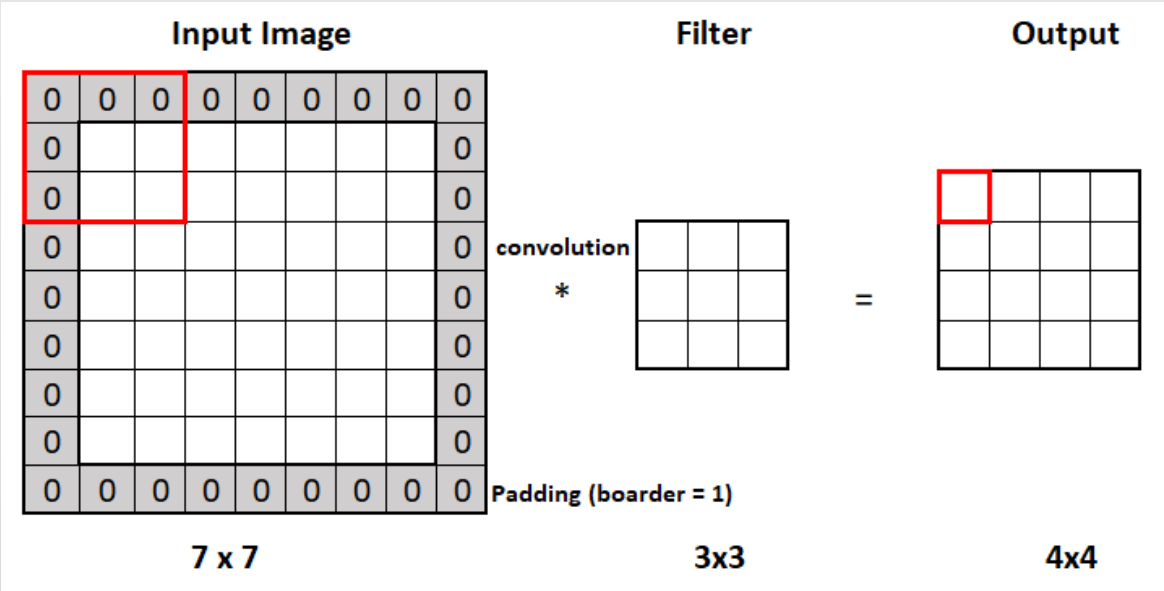


The output matrix size can be computed for given:

- **Input matrix (image) dimension *n x n***
- **Filter size *f x f***
- **Stride *s***
- **Padding *p***

in the following way: $\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right]$

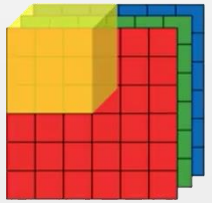
Example for n = 7, f = 3, s = 2, p = 1: $\left[\frac{7+2\cdot1-3}{2}+1\right] \times \left[\frac{7+2\cdot1-3}{2}+1\right] = 4 \times 4$

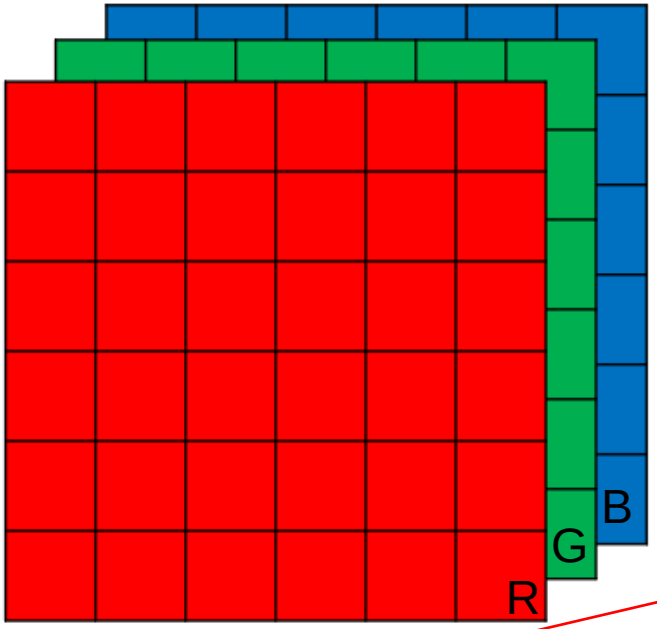


Multiple Adaptive Filters on RGB Images



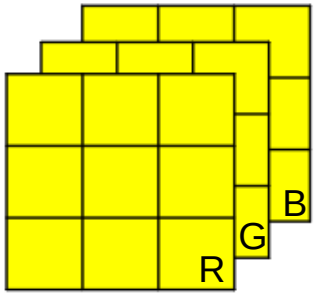
If the input image has 3 color channels then the filters must also have the depth equal to 3, so we always convolve over the volume:





6 x 6 x 3

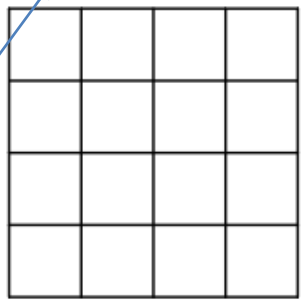
*



3 x 3 x 3

Filter 1:

=



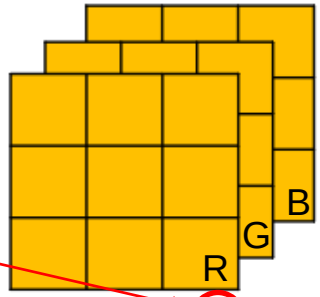
4 x 4

Number of channels (filters or depth of the conv. layer):

$n_c = 2$

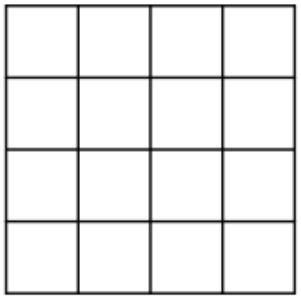
Filter 2:

*



3 x 3 x 3

=



4 x 4

Output Volume Size =

$$\left[\frac{n+2p-f}{s}+1\right] \times \left[\frac{n+2p-f}{s}+1\right] \times n_c$$

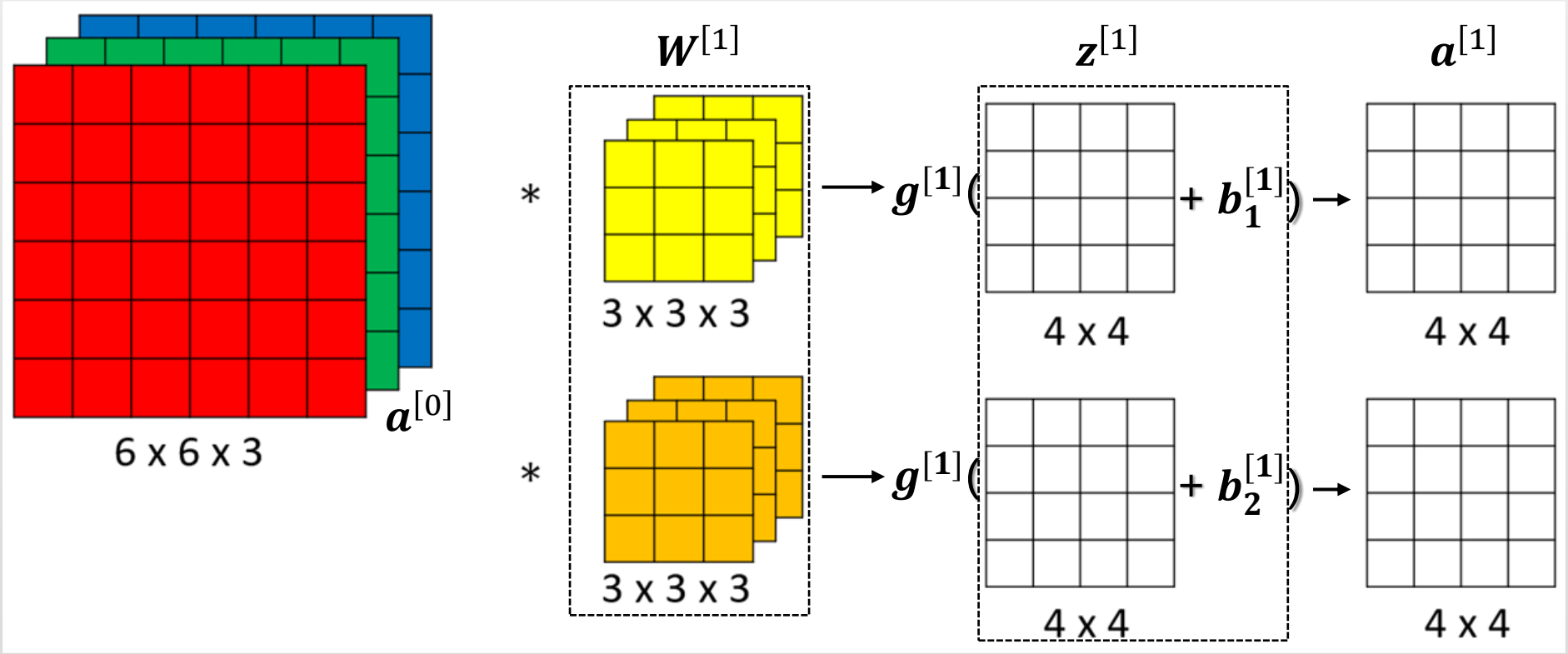
Convolution and Convolutional Layer



What happens in the convolutional layer?

The input $a^{[0]}$ is convolved by the convolutional filters $W^{[1]}$ and using the bias $b^{[1]}$ the output $a^{[1]}$ is computed (here two filters are used):

$$z^{[1]} = W^{[1]} \cdot a^{[0]} + b^{[1]} \qquad a^{[1]} = g^{[1]}\left(z^{[1]}\right)$$



Number of parameters = (number of weights + bias) * number of filters = (3x3x3 + 1) * 2 = 28 * 2 = 56

Convolutional Layer Notation



For a convolutional layer l, we will use the following notations:

$f^{[l]}$ - filter size

$p^{[l]}$ - padding

$s^{[l]}$ - stride

$n_H^{[l]}$ - height (vertical dimension)

$n_W^{[l]}$ - width (horizontal dimension)

$n_c^{[l]}$ - number of channels or filters (depth of the layer)

For a given input:

$$n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$$

we get the following filter size:

$$f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$$

and weight size:

$$f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$$

and the output:

$$n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]} = \left[\frac{n_H^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1\right] \times \left[\frac{n_W^{[l-1]} + 2 \cdot p^{[l]} - f^{[l]}}{s^{[l]}} + 1\right] \times n_c^{[l]}$$

$$A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$$

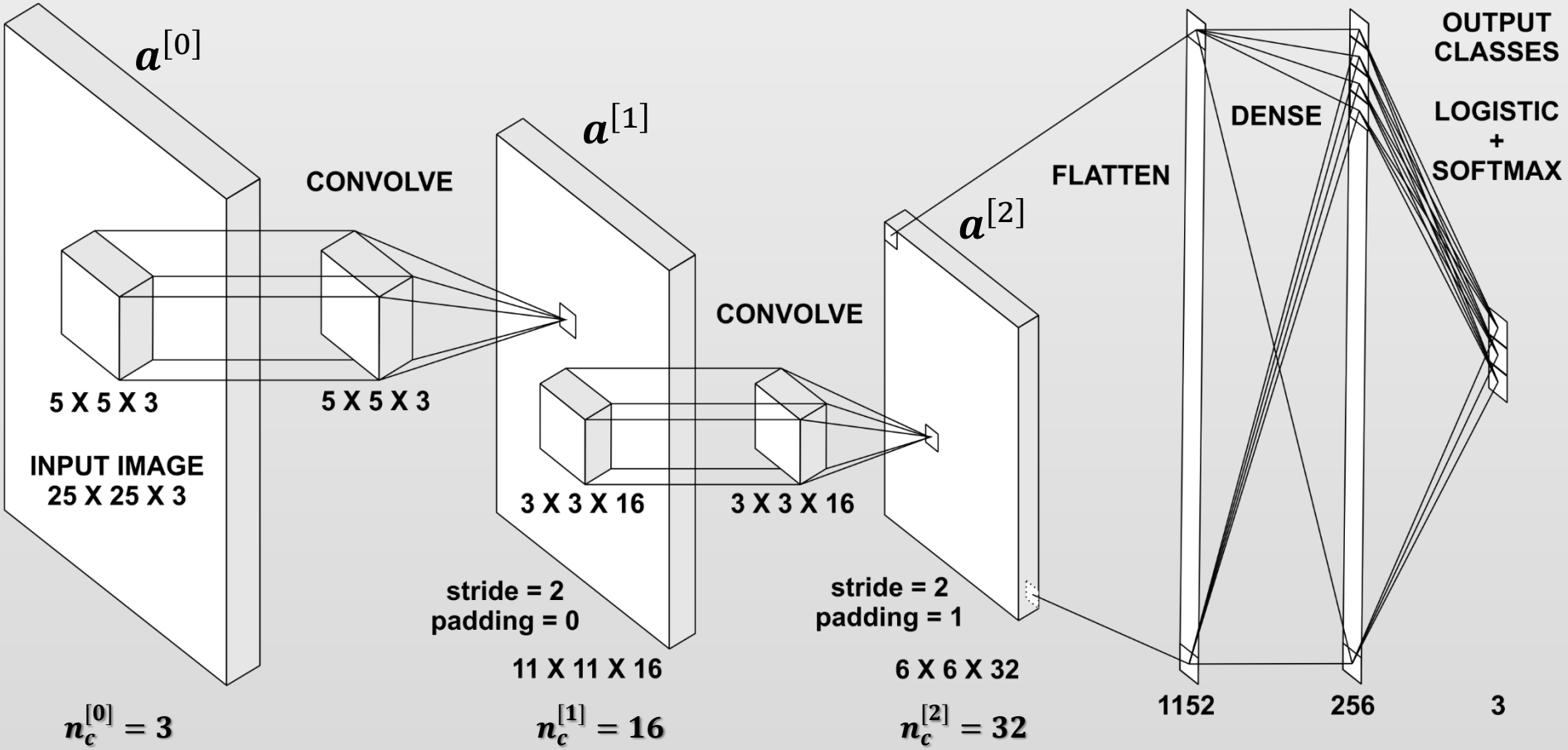
Example of Simple Convolutional Network



Let's compute the sizes for this exemplar convolutional network:

$$n_H^{[1]} \times n_W^{[1]} \times n_c^{[1]} = \left[\frac{25 + 2 \cdot 0 - 5}{2} + 1\right] \times \left[\frac{25 + 2 \cdot 0 - 5}{2} + 1\right] \times 16 = 11 \times 11 \times 16$$

$$n_H^{[2]} \times n_W^{[2]} \times n_c^{[2]} = \left[\frac{11 + 2 \cdot 1 - 3}{2} + 1\right] \times \left[\frac{11 + 2 \cdot 1 - 3}{2} + 1\right] \times 32 = 6 \times 6 \times 32 = 1152 = n_H^{[3]}$$

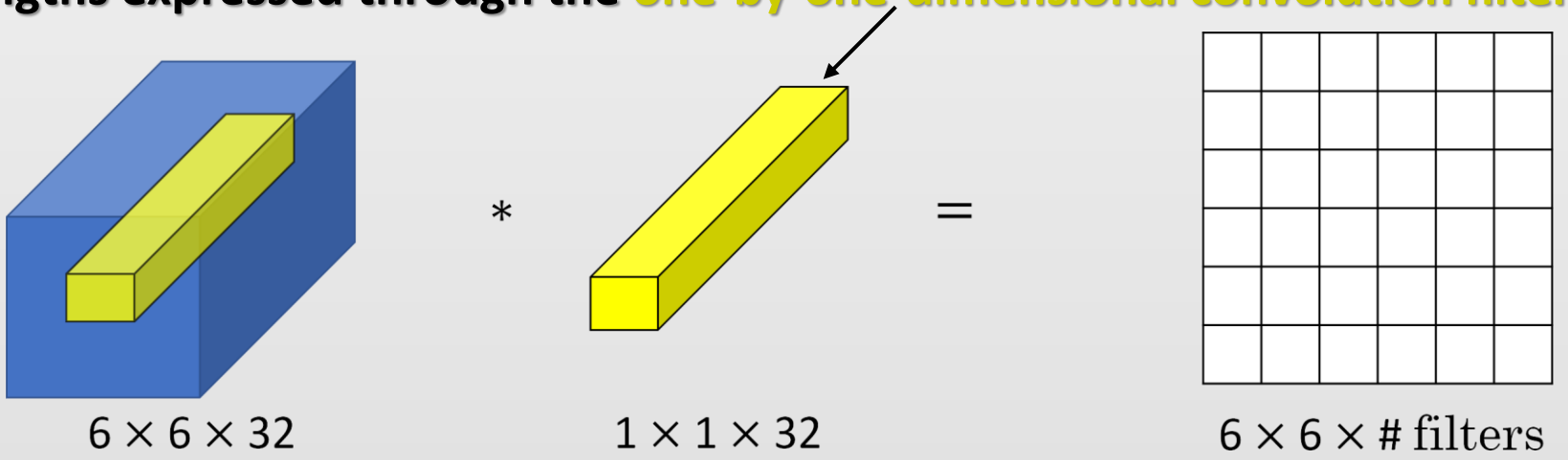


1 x 1 Convolutions

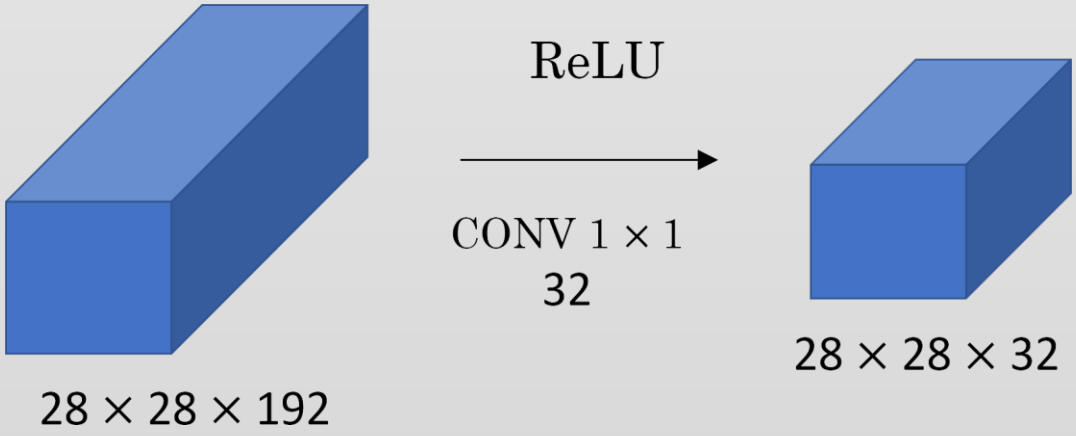


[Paper: Network In Network, Authors: Min Lin, Qiang Chen, Shuicheng Yan. National University of Singapore, arXiv preprint, 2013]:

One-by-one convolutions (called also as network in network) can use various features represented by the various convolutional filters with different strengths expressed through the one-by-one-dimensional convolution filter:



This kind of convolution can be used to shrink the filter volume (depth):

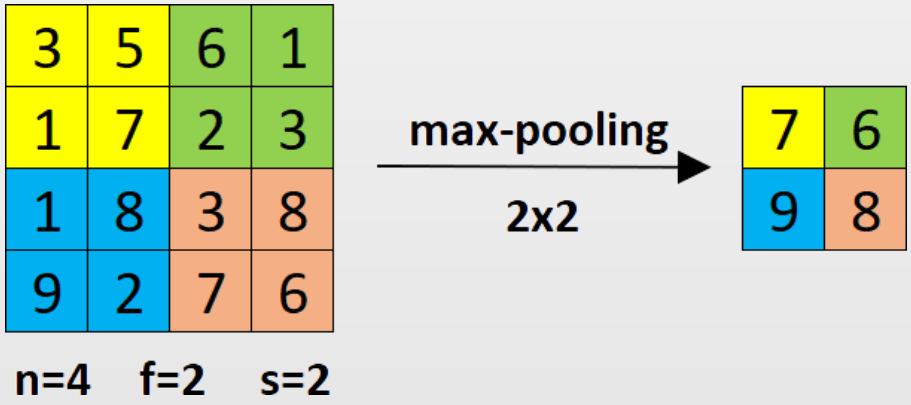


Pooling Layer

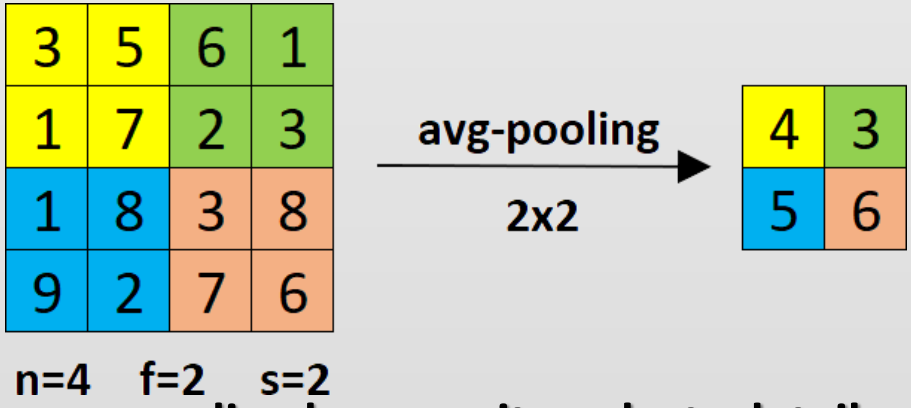


To downsample the image, we often use pooling layers:

- **Max-pooling chooses the maximum value from the selected region (stride = 2):**



- **Avg-pooling chooses the average value from the selected region (stride = 2):**



Be careful about using max-pooling because it neglects details.

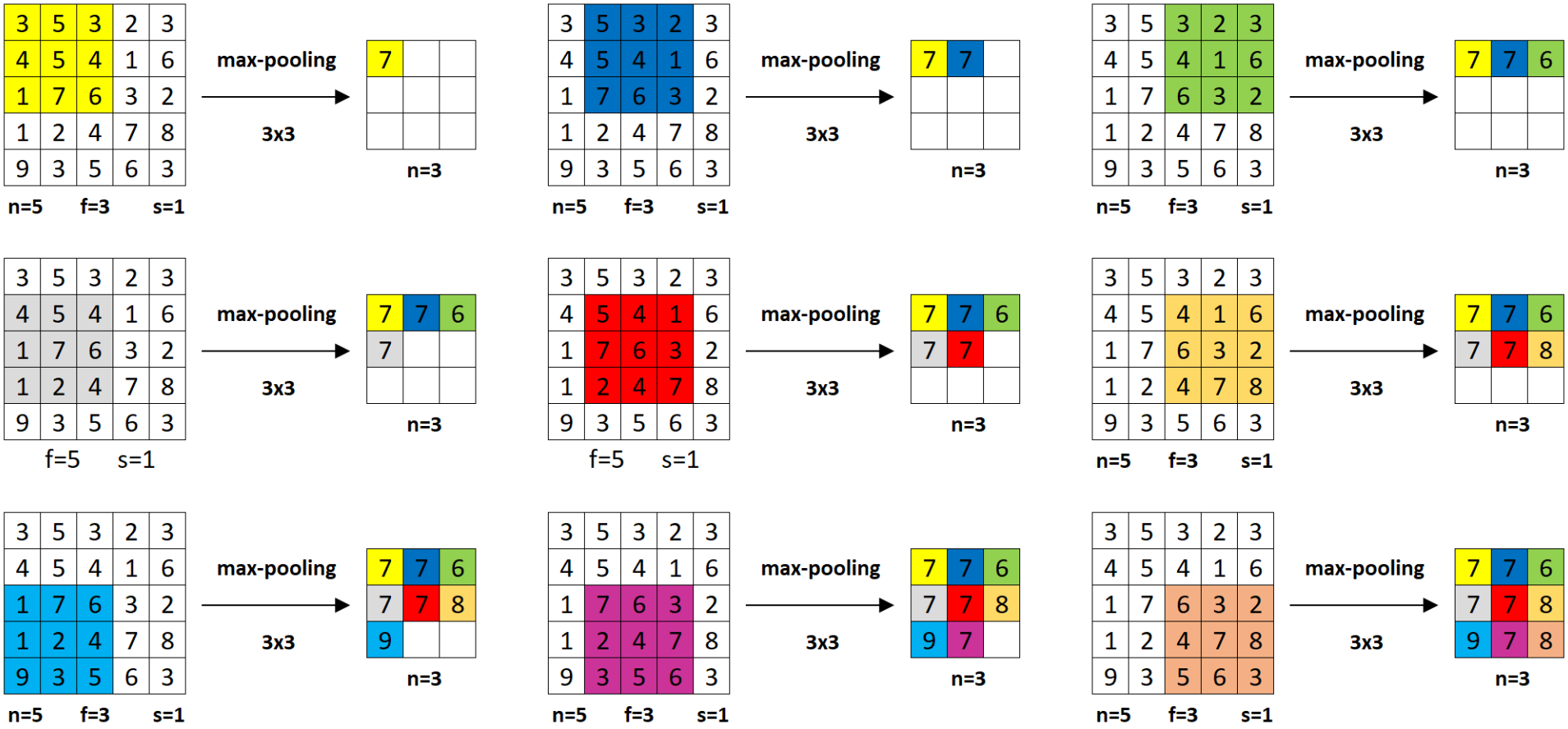
Max-pooling is more often used in the convolutional networks.

We usually do not use padding (padding = 0) for the pooling operations.

Pooling Layer



Max-pooling layer for stride = 1, filter size = 3x3:



Notice that there are no parameters that can be adapted during the training process!

Max-pooling and avg-pooling are computed separately for each channel.

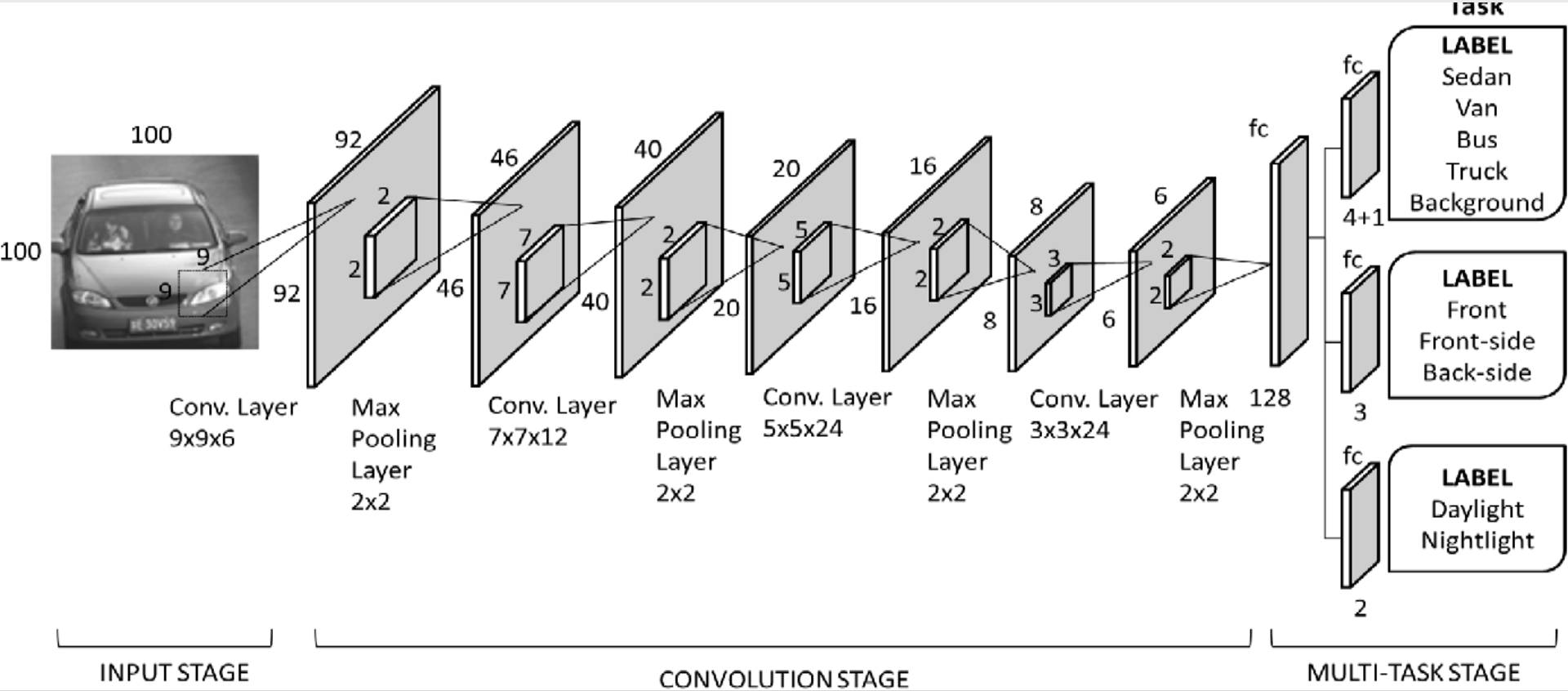
It is often used to downsample the high-dimensional images.

Pooling layers



Pooling layers are usually counted together with convolutional layers, however sometimes are computed separately.

An example convolutional network with pooling layers:



Presentation of CNN Structure



We usually present the structure of convolutional networks in the following way:

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

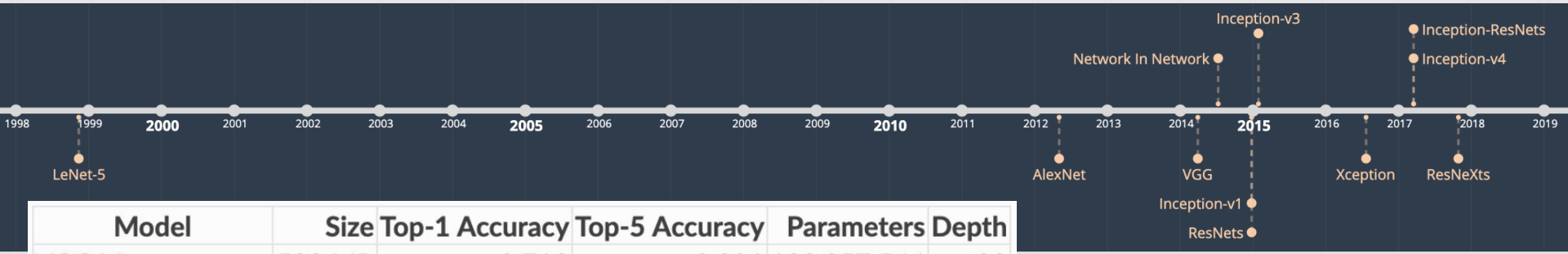
Benefits of using CNN



Convolutional Neural Network:

- **Share parameters**, so the same feature may be recognized in any part of the image
- **Use sparse connections**, so the convolutional layers are not connected all-to-all (dense/fully-connected), which saves a lot of parameters, and allows to train the network faster.
- Outputs depend directly only on some **selected areas** of the input images, so the neurons can specialize in recognizing, but their position in the convolutional layer defines the location where the features have been found.

Timeline of the development of Convolutional Neural Networks:



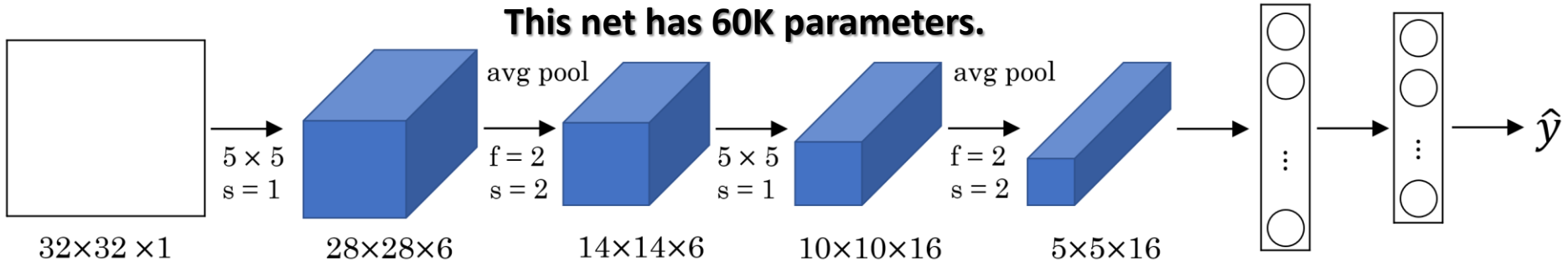
Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
VGG16	528 MB	0.713	0.901	138,357,544	23
InceptionV3	92 MB	0.779	0.937	23,851,784	159
ResNet50	98 MB	0.749	0.921	25,636,712	-
Xception	88 MB	0.790	0.945	22,910,480	126
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
ResNeXt50	96 MB	0.777	0.938	25,097,128	-

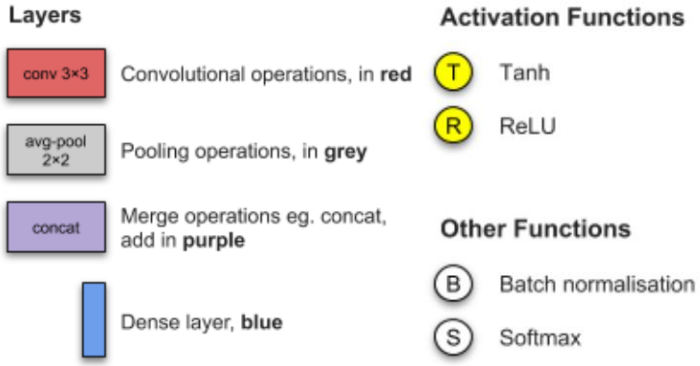
LeNet-5 (1998)



[LeCun et al., 1998. Gradient-based learning applied to document recognition]:

This net has 60K parameters.

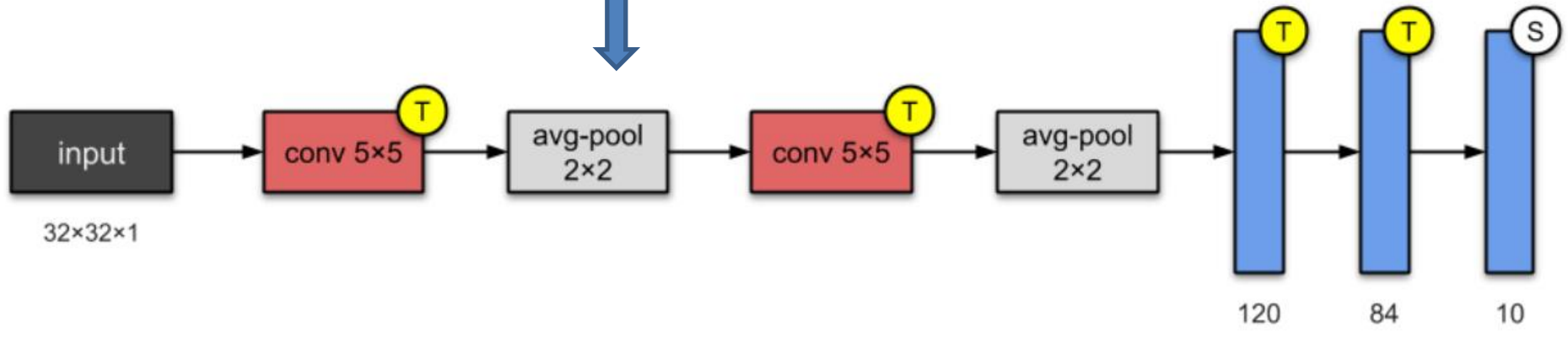




LeNet-5 is one of the simplest architectures.

The average-pooling layer as we know it now was called a sub-sampling layer and it had trainable weights which isn't the current practice of designing CNNs nowadays.

The modern version of LeNet-5 uses SoftMax in the output layer.



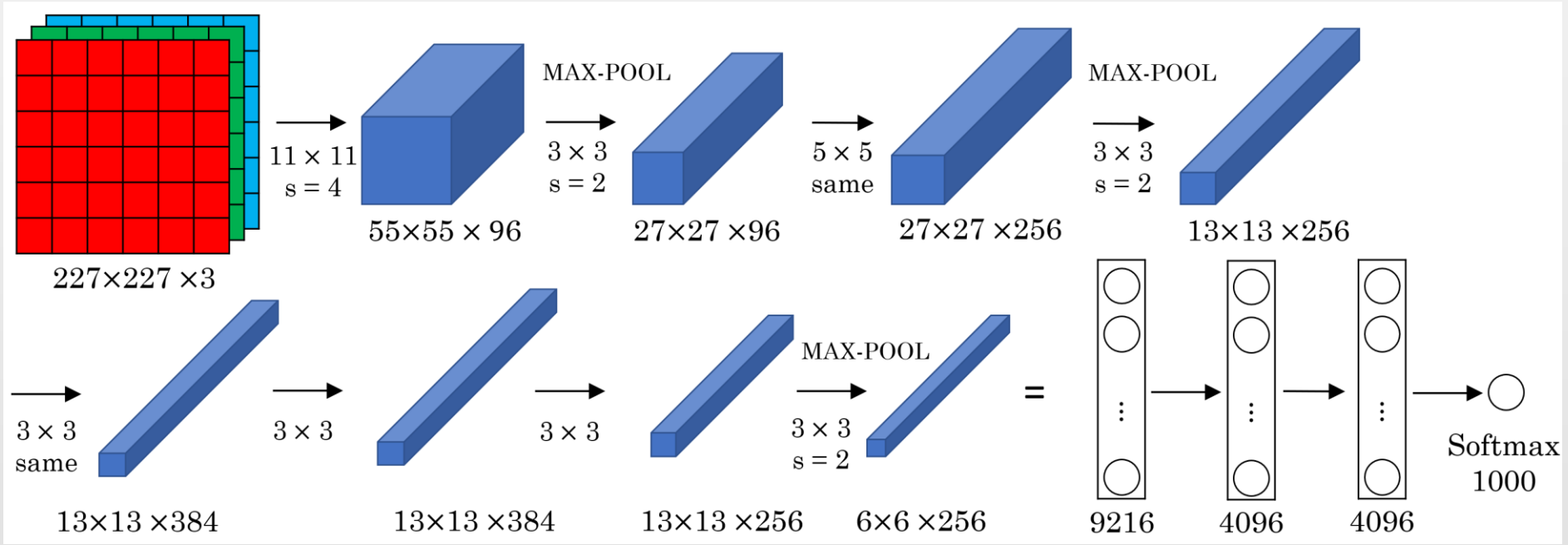
[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

AlexNet (2012)

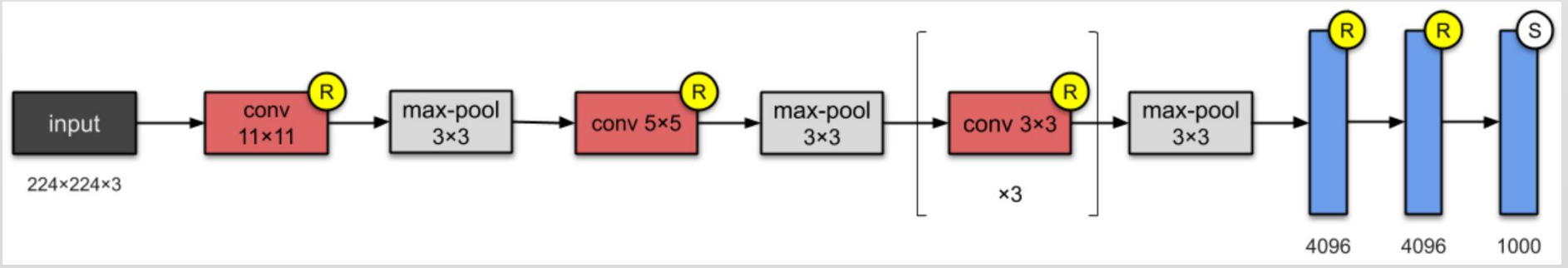


[Krizhevsky et al., 2012. ImageNet classification with deep convolutional neural networks]:

It was the first to implement Rectified Linear Units (ReLUs) as activation functions.



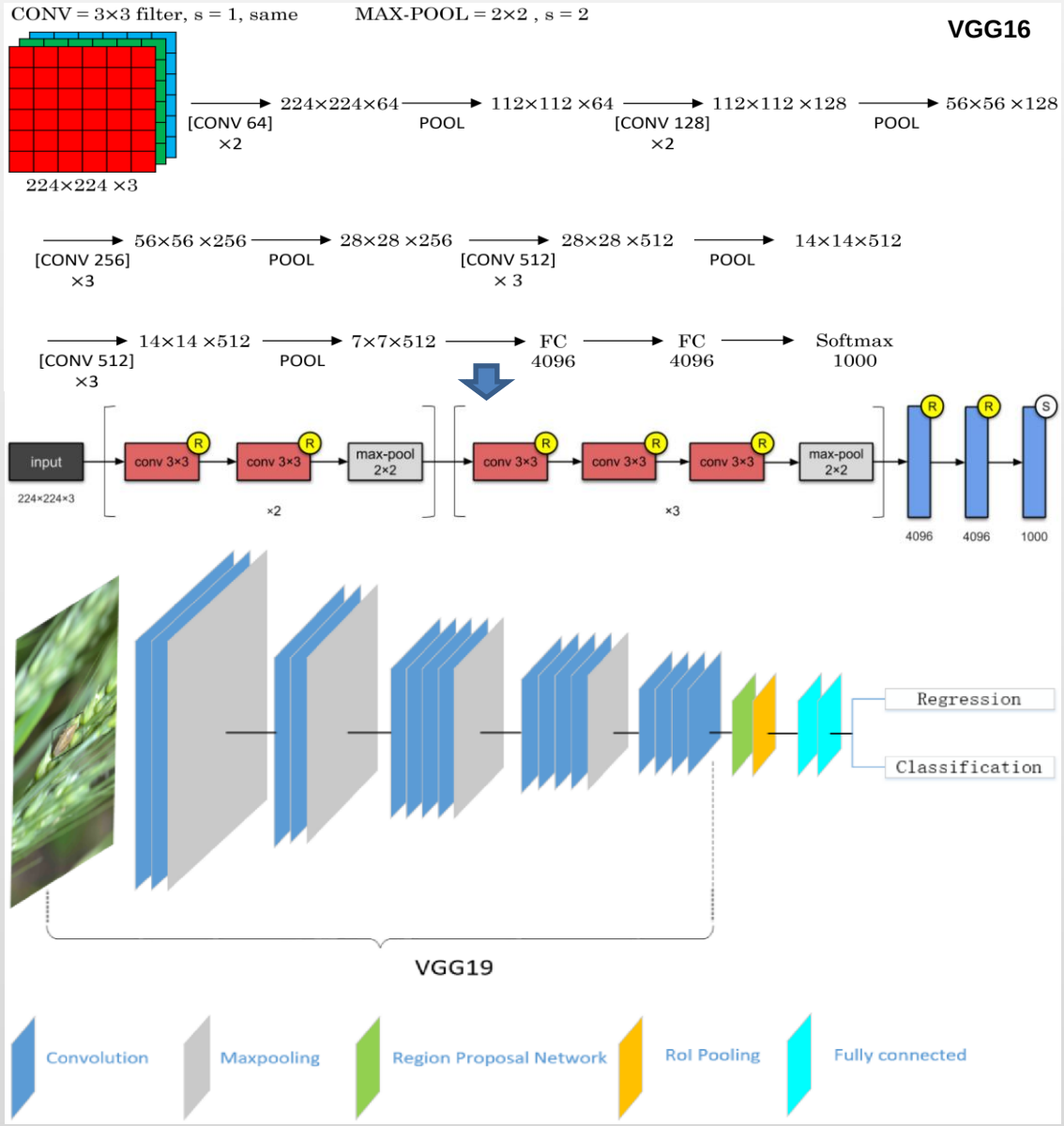
This net has 60M parameters.



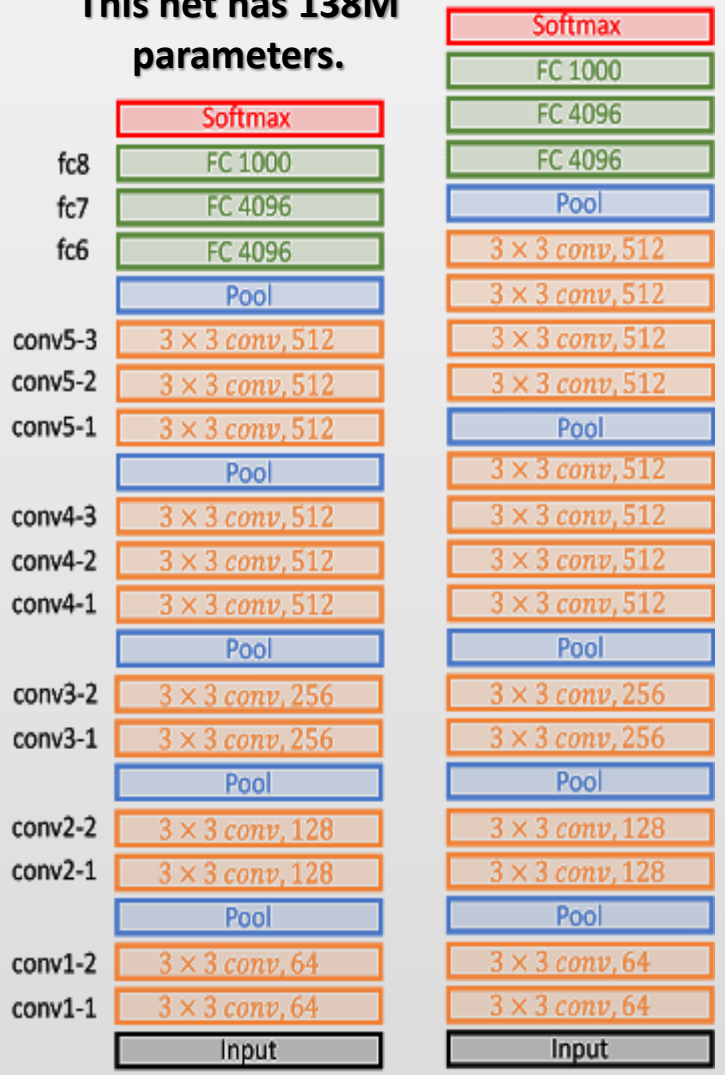
VGG-16 and VGG-19 (2014)



[Simonyan & Zisserman 2015. Very deep convolutional networks for large-scale image recognition]:



This net has 138M parameters.



ResNets

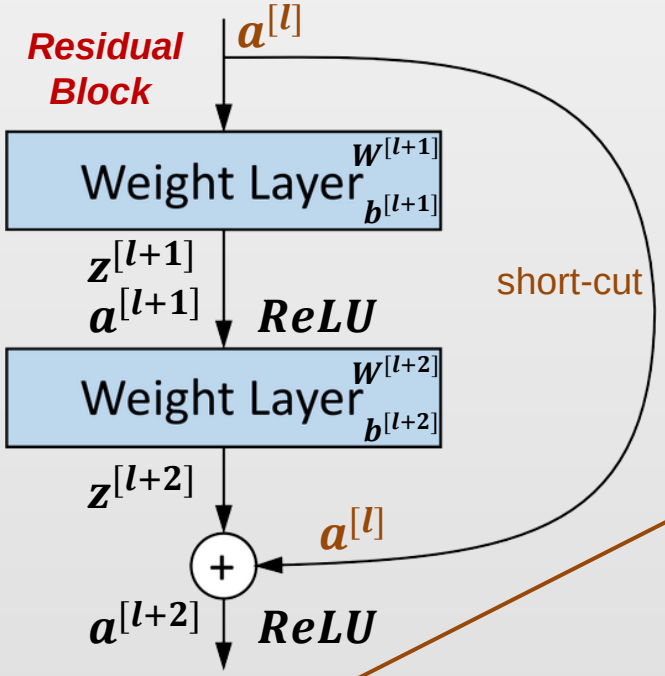


[He at al., 2015, Deep residual networks for image recognition]:

ResNets are constructed from the stacked residual blocks that regularize the non-linear processing using short-cut (identity, skip connection) connections:

$$z^{[l+1]} = W^{[l+1]} \cdot a^{[l]} + b^{[l+1]}$$
$$a^{[l+1]} = ReLU\left(z^{[l+1]}\right)$$

$$z^{[l+2]} = W^{[l+2]} \cdot a^{[l+1]} + b^{[l+2]}$$
$$a^{[l+2]} = ReLU\left(z^{[l+2]} + a^{[l]}\right)$$

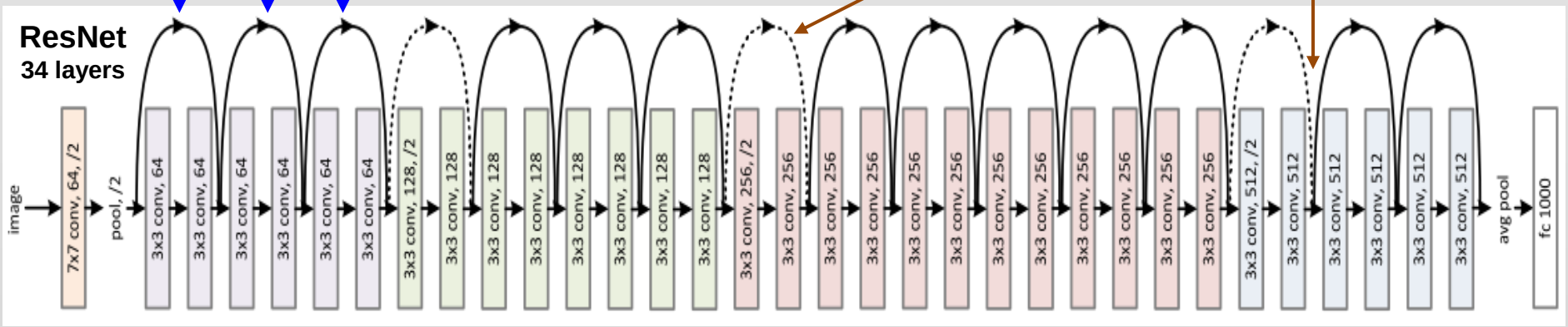


ResNets are allow us to construct much deeper architectures because residual blocks avoid overfitting.

$a^{[l]}$ and $z^{[l+2]}$ must have the same dimensions, so in ResNets, we use the "same convolutions":

If we want to use different dimensions of $a^{[l]}$ and $z^{[l+2]}$, we must use an extra weight matrix W_s to transform:

$$a^{[l+2]} = ReLU\left(z^{[l+2]} + W_s^{[l+2]} \cdot a^{[l]}\right)$$

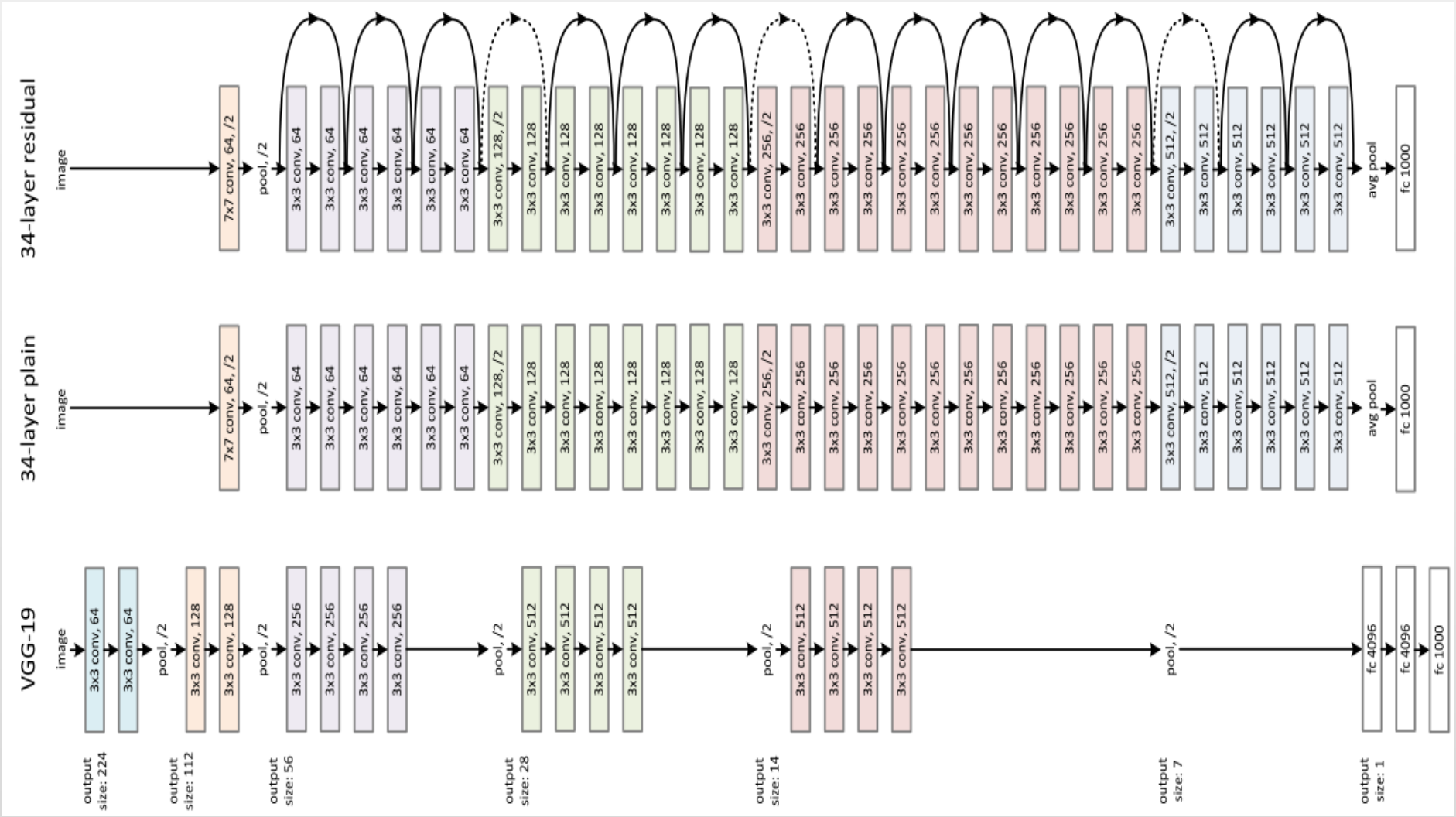


Comparison of ResNet to PlainNet and VGG-19

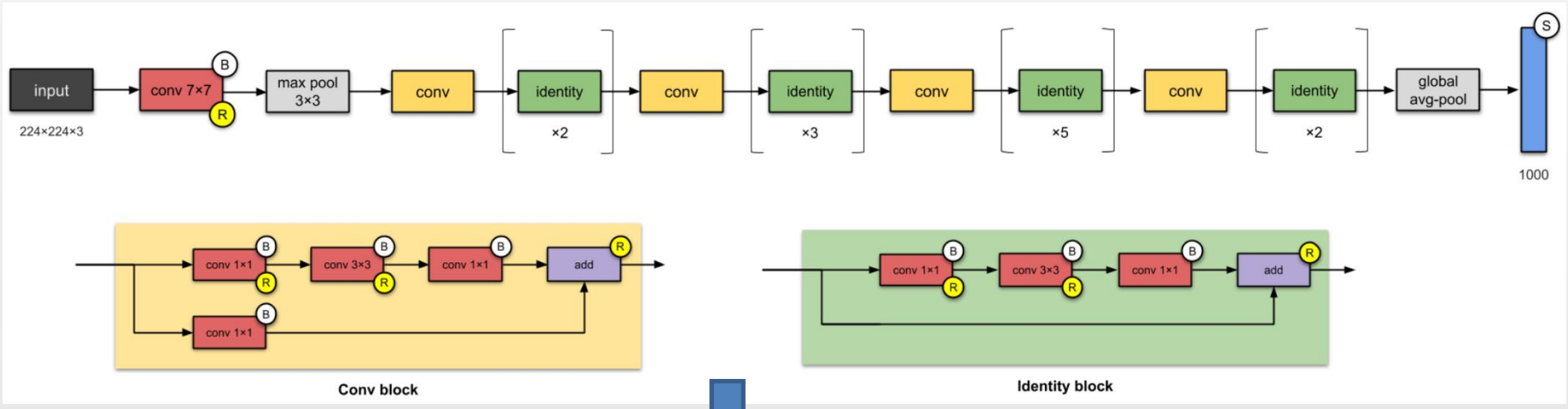


[He at al., 2015, Deep residual networks for image recognition]:

ResNets are constructed from the stacked residual blocks that regularize the non-linear processing using short-cut (identity, skip connection) connections:



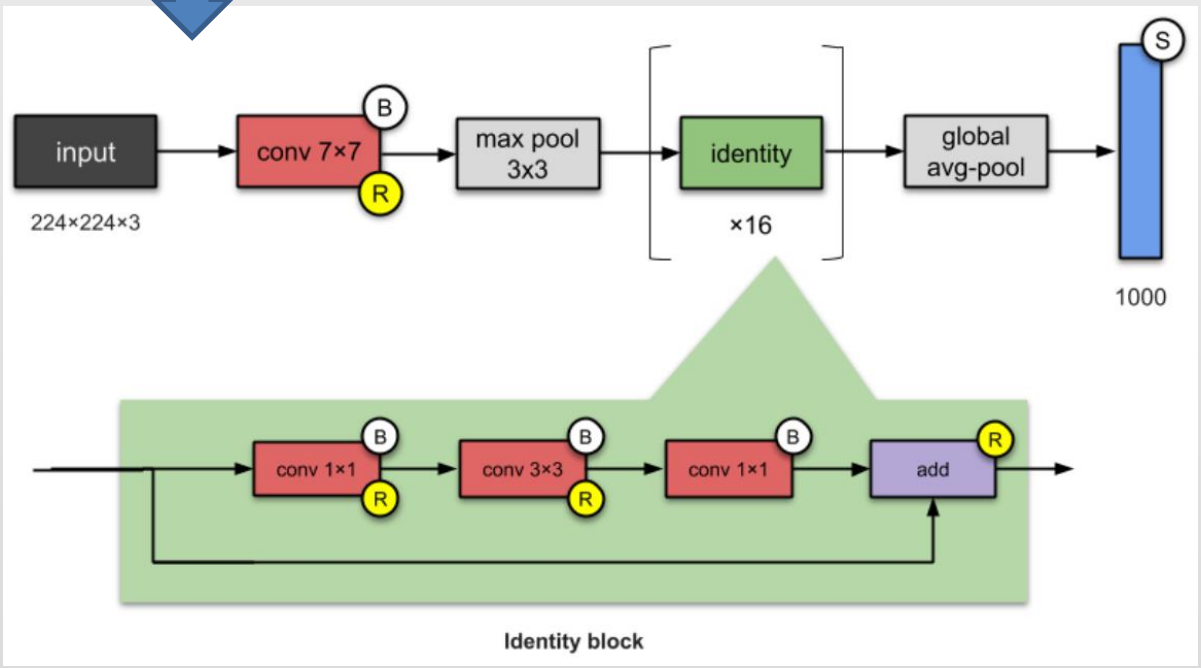
ResNet-50 (2015)



This net has 26M parameters!

It used skip connections the first time, designed much deeper CNNs (up to 152 layers) without compromise with generalization, and was among the first to use batch normalisation.

Paper: [Deep Residual Learning for Image Recognition](), Authors: Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Microsoft Published in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

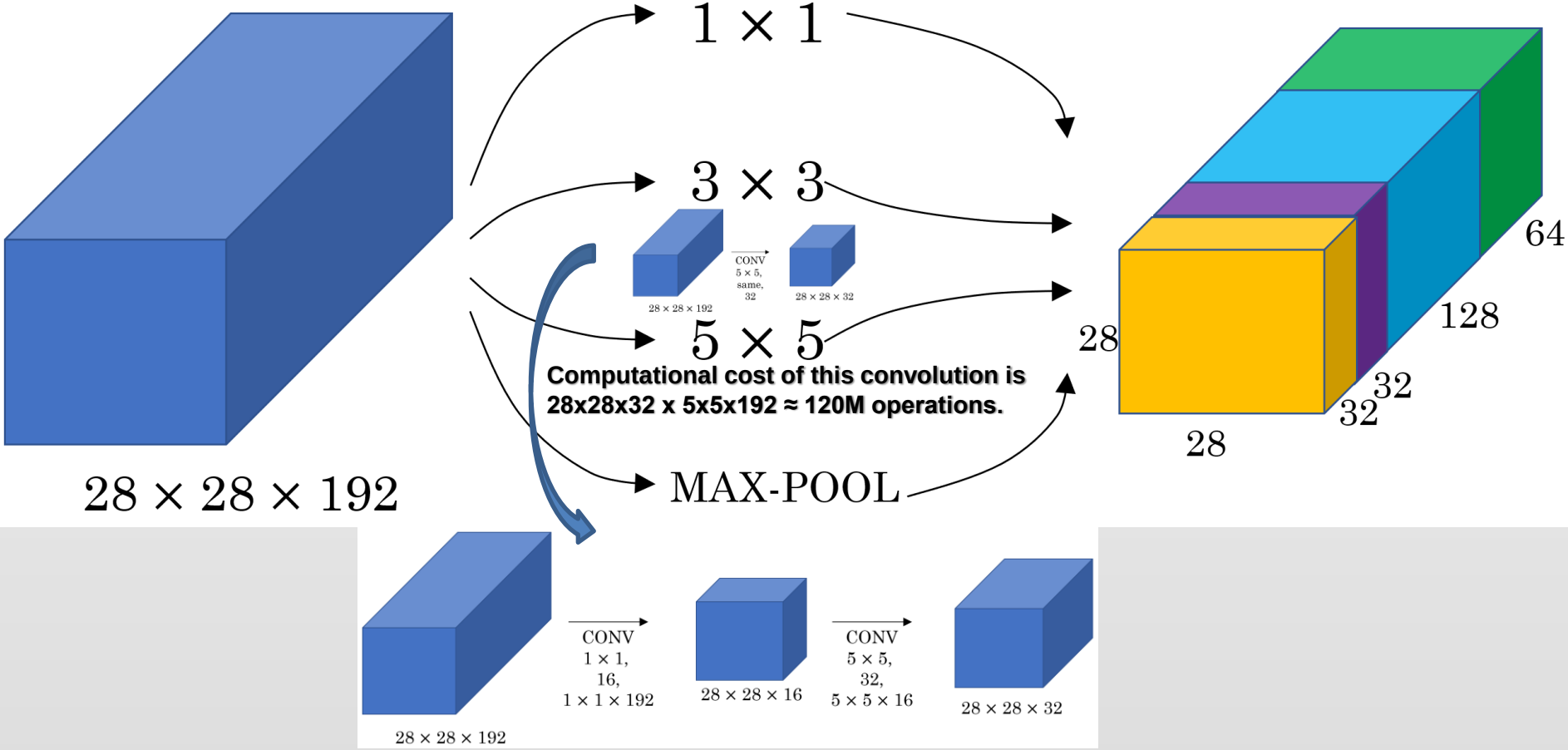


[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

Inception Module



Inception modules allow to use various convolutions (filters) at the same time:



Using 1x1 convolutions we can reduce the number or multiplications 10 times:

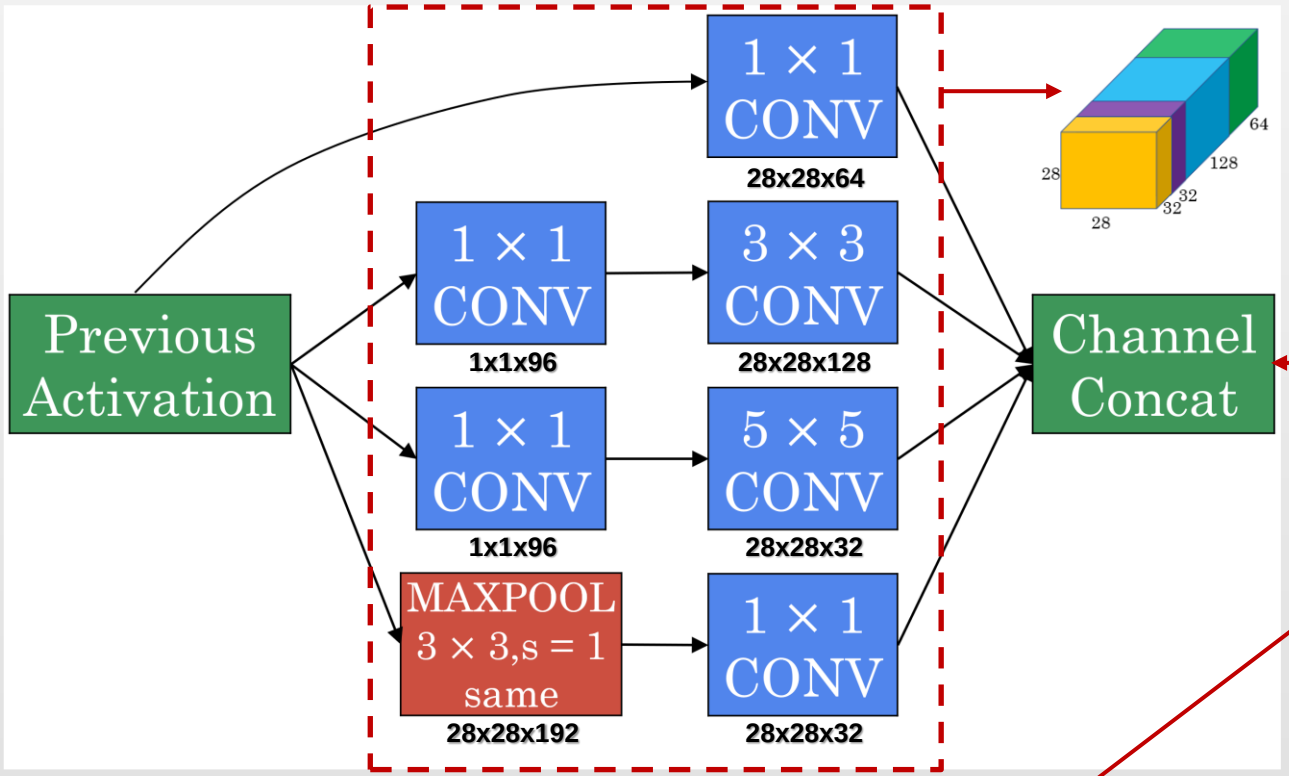
(28x28x16 x 1x1x192) + (28x28x16 x 1x1x192) ≈ 12.4M operations

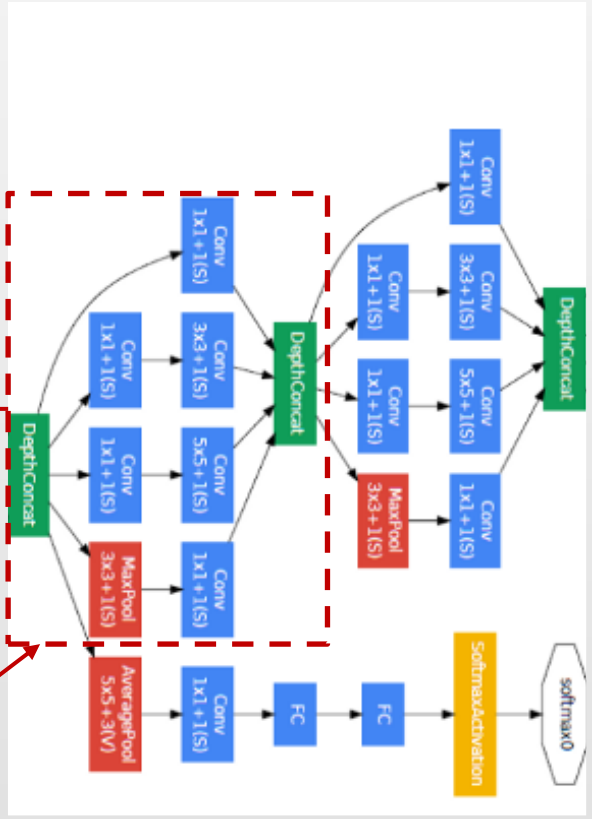
[Szegedy et al. 2014. Going deeper with convolutions]

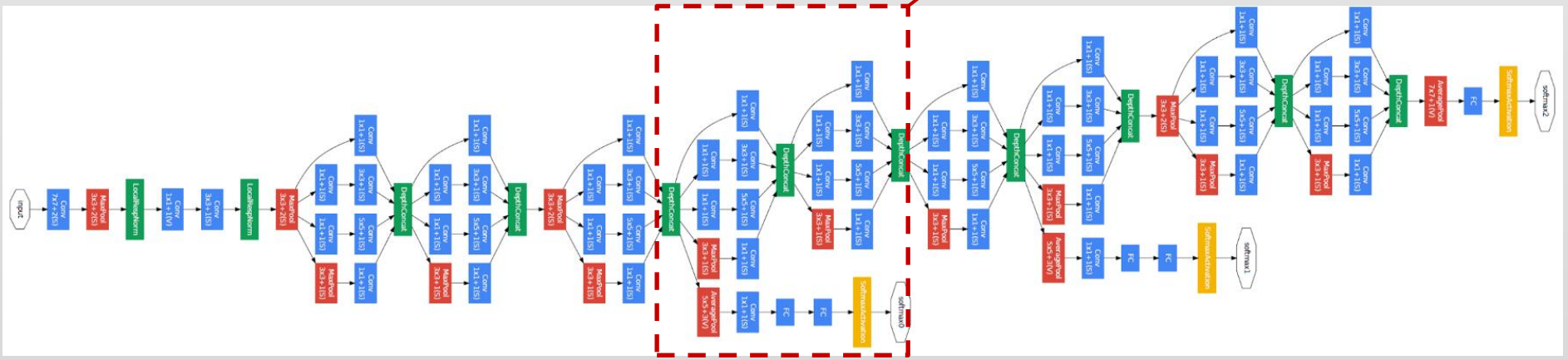
Inception Networks (2014)



Building an inception network from inception modules:



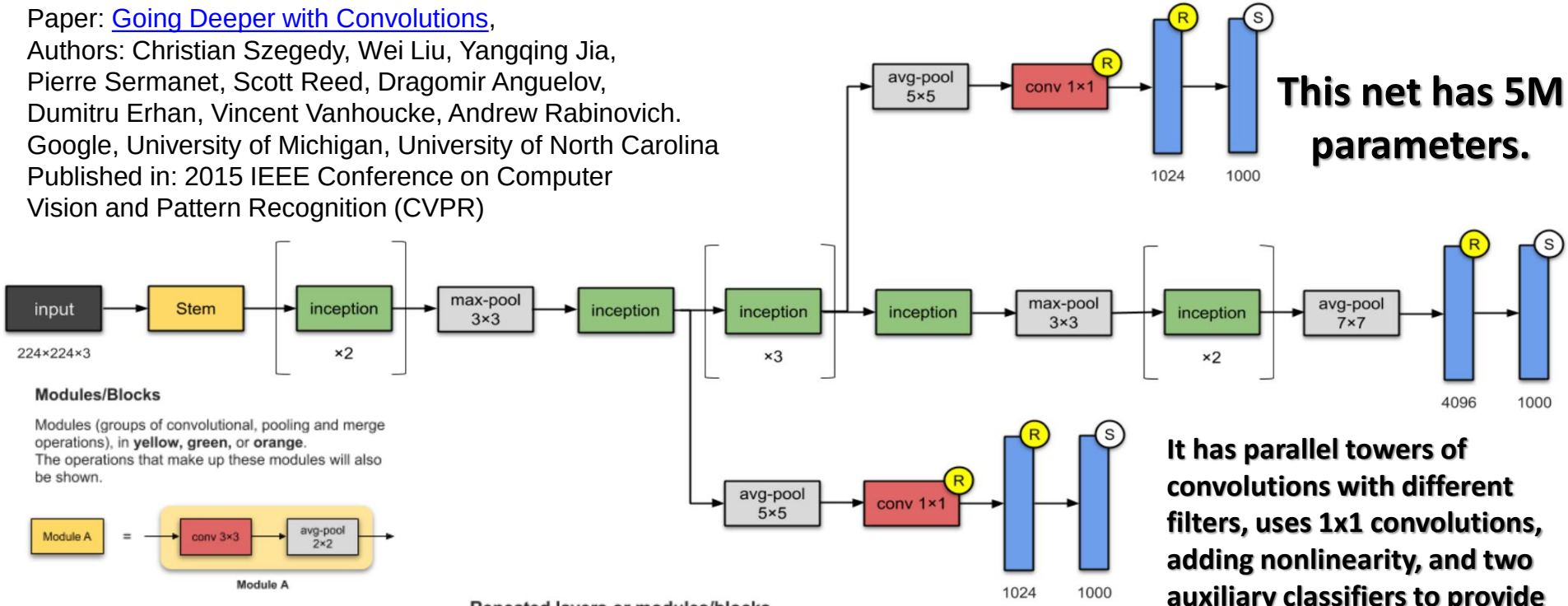




Inception-v1 (2014)



Paper: [Going Deeper with Convolutions](),
Authors: Christian Szegedy, Wei Liu, Yangqing Jia,
Pierre Sermanet, Scott Reed, Dragomir Anguelov,
Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich.
Google, University of Michigan, University of North Carolina
Published in: 2015 IEEE Conference on Computer
Vision and Pattern Recognition (CVPR)



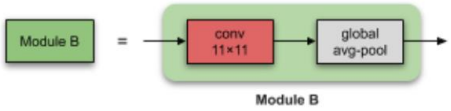
This net has 5M parameters.

It has parallel towers of convolutions with different filters, uses 1x1 convolutions, adding nonlinearity, and two auxiliary classifiers to provide additional regularization.

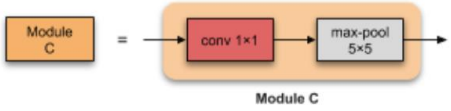
Modules/Blocks

Modules (groups of convolutional, pooling and merge operations), in **yellow, green,** or **orange**.
The operations that make up these modules will also be shown.

Module A

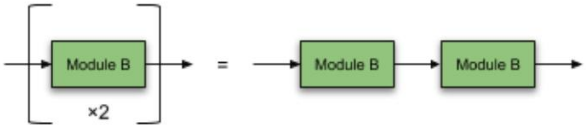


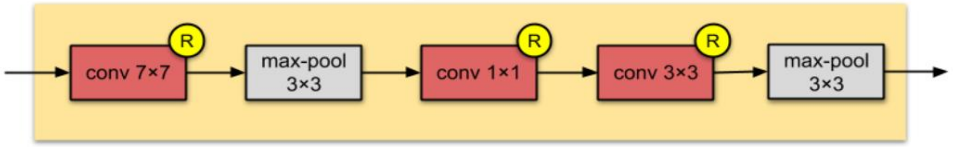
Module B



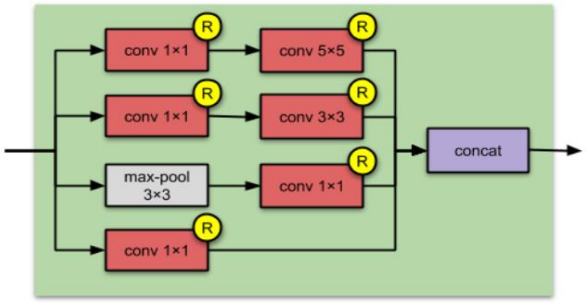
Module C

Repeated layers or modules/blocks





Stem



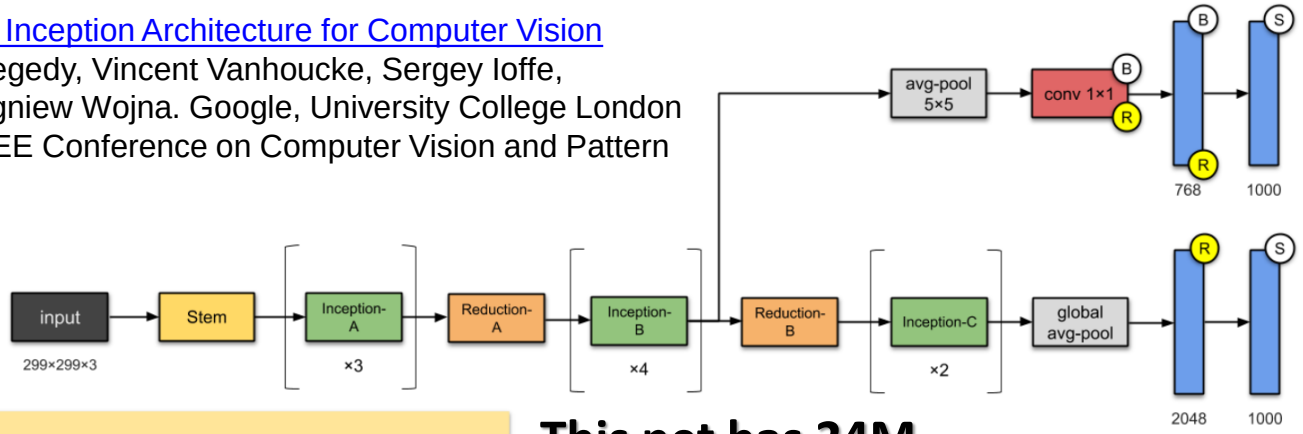
Inception module

[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

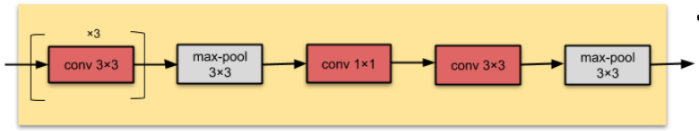
Inception-v3 (2015)



Paper: [Rethinking the Inception Architecture for Computer Vision]
Authors: Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Google, University College London
Published in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

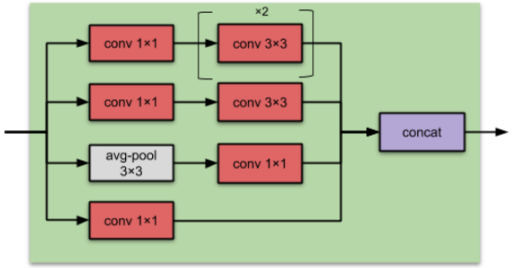


It factorizes $n \times n$ convolutions into asymmetric convolutions: $1 \times n$ and $n \times 1$ convolutions, 5×5 convolution to two 3×3 convolutions, and replaces 7×7 to a series of 3×3 convolutions

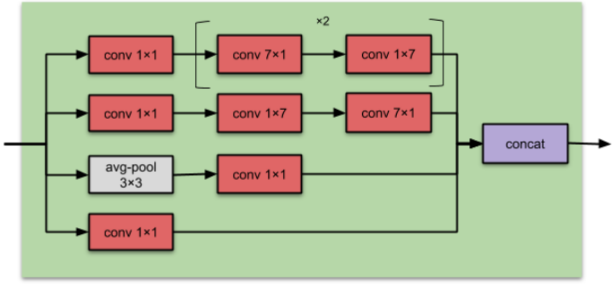


Stem

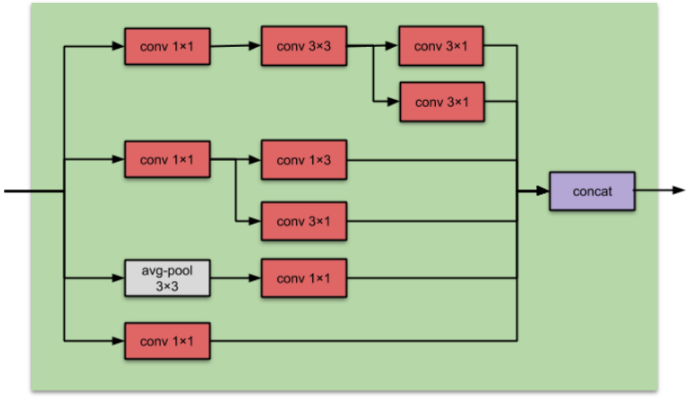
This net has 24M parameters.



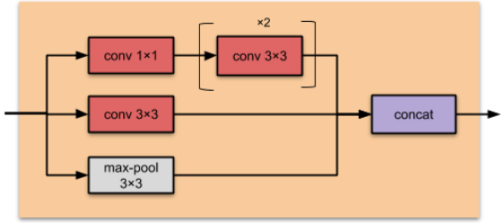
Inception-A



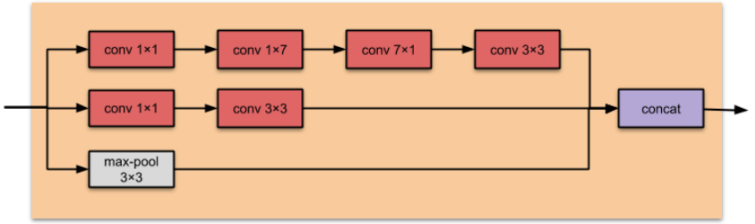
Inception-B



Inception-C



Reduction-A



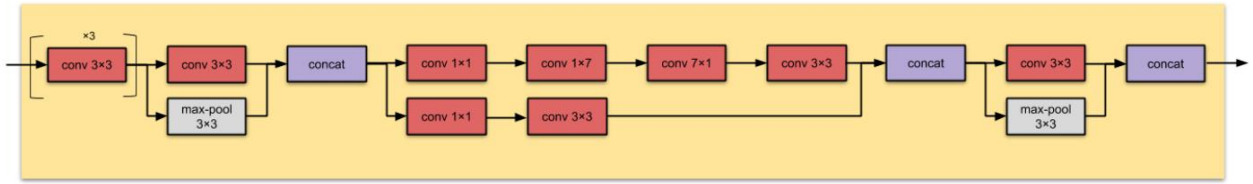
Reduction-B

[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

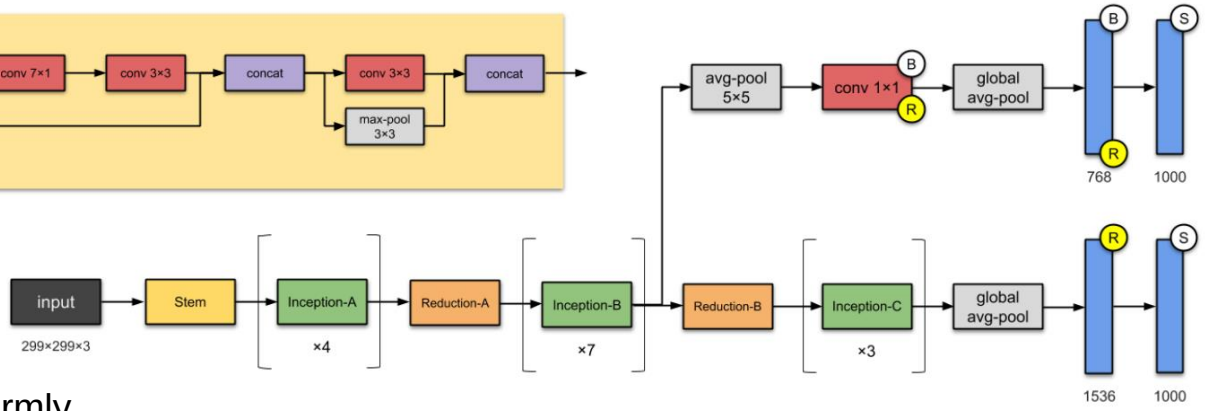
Inception-v4 (2016)



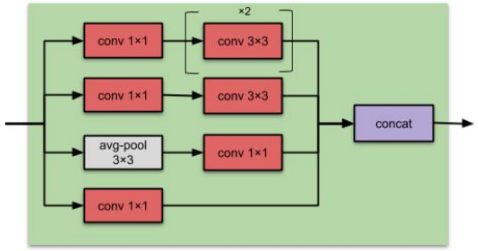
Paper: [Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning]
Authors: Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi. Google.
Published in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence



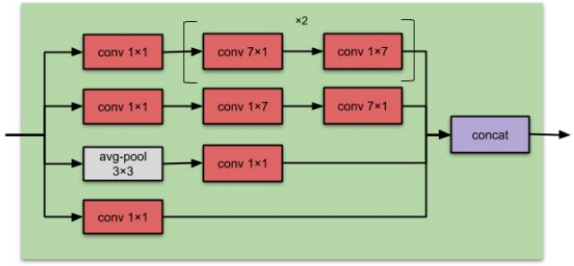
Stem



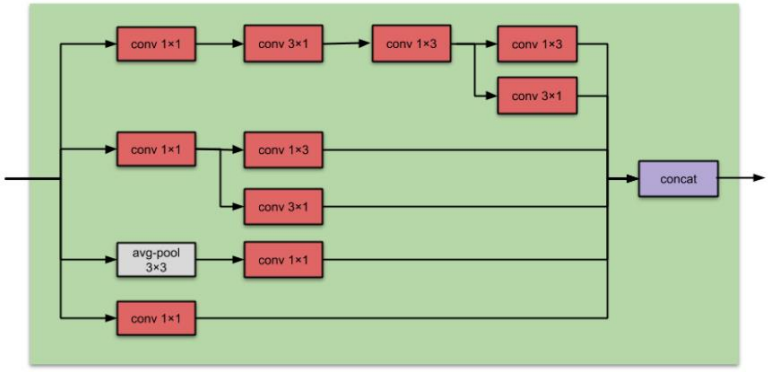
It changed in Stem module,
added more Inception modules,
and chose Inception-v3 modules uniformly,
i.e. used the same number of filters for every module.



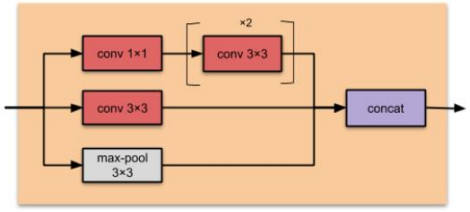
Inception-A



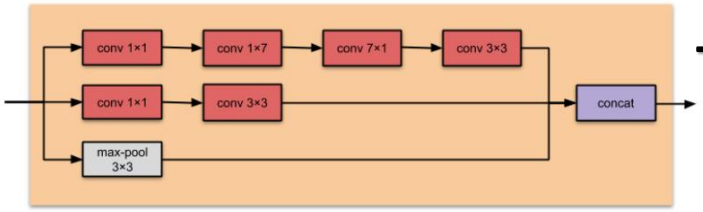
Inception-B



Inception-C



Reduction-A



Reduction-B

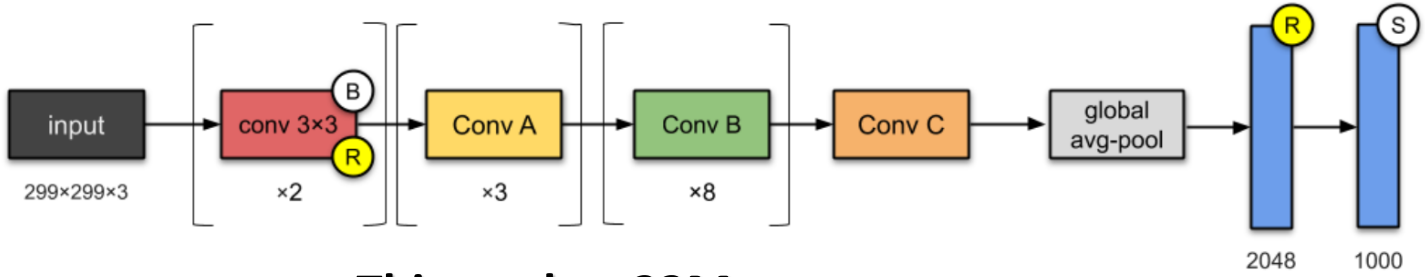
This net has 43M parameters.

[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

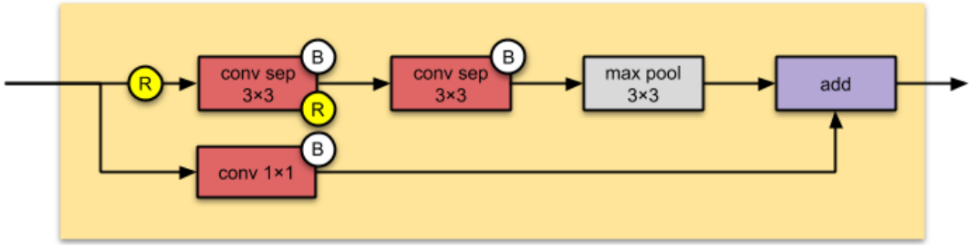
Xception (2016)



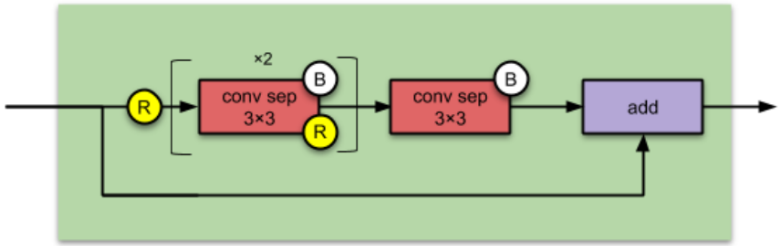
Xception is an adaptation from Inception, where the Inception modules have been replaced with depth-wise separable convolutions.



This net has 23M parameters.

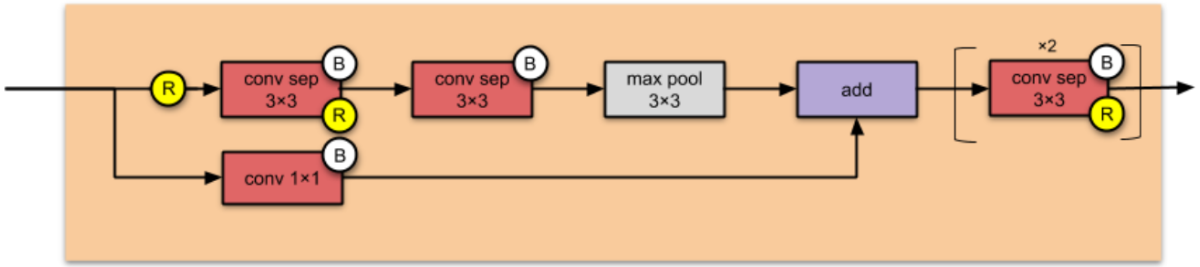


Conv A



Conv B

Paper: [Xception: Deep Learning with Depthwise Separable Convolutions]
Authors: François Chollet. Google.
Published in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)



Conv C

Cross-channel correlations were captured by 1×1 convolutions, and spatial correlations within each channel were captured via the regular 3×3 or 5×5 convolutions.

[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

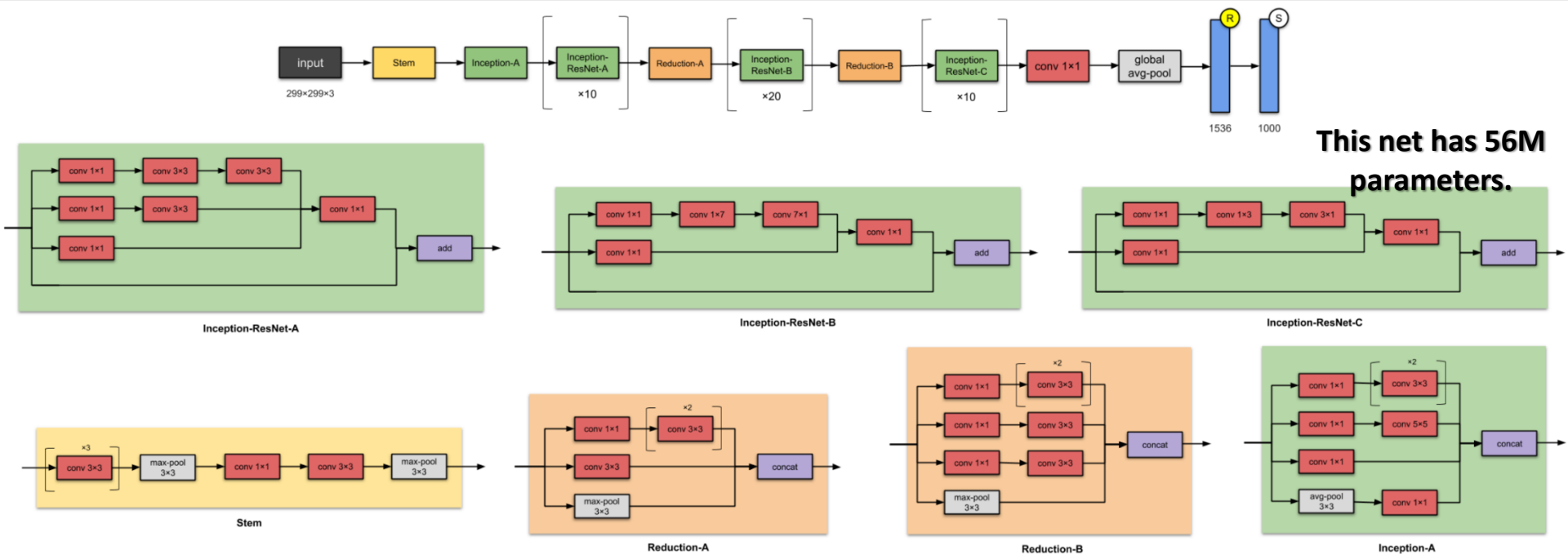
Inception ResNet-v2 (2016)



Paper: [Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning], Authors: Christian Szegedy, Sergey Loffe, Vincent Vanhoucke, Alex Alemi. Google. Published in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence

This solution:

- converts Inception modules to Residual Inception blocks.
- adds more Inception modules.
- adds a new type of Inception module (Inception-A) after the Stem module.



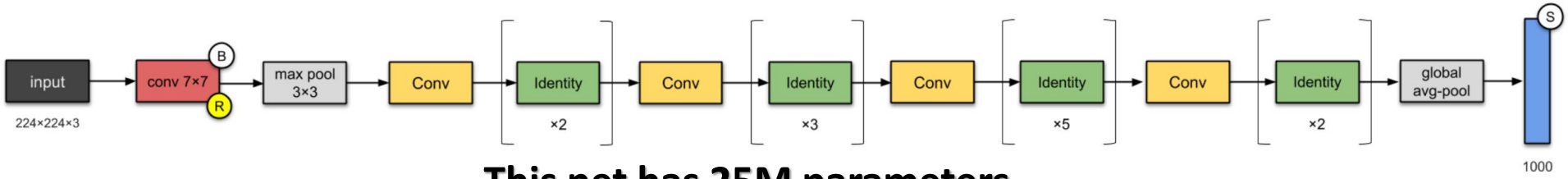
[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

ResNeXt-50 (2017)

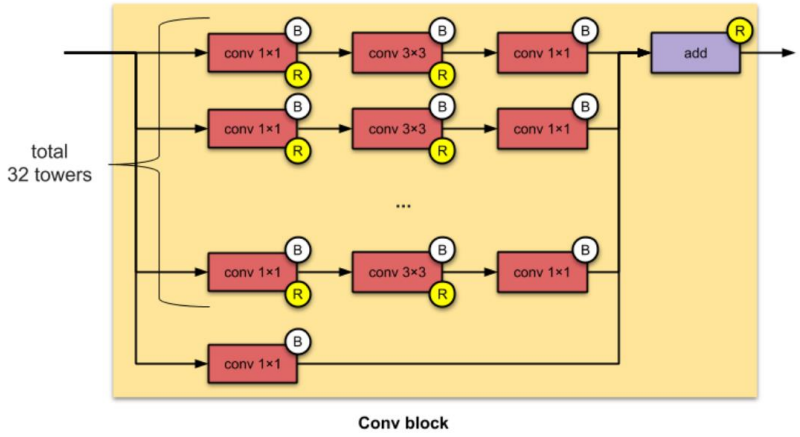


Paper: [Aggregated Residual Transformations for Deep Neural Networks]
Authors: Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He. University of California San Diego, Facebook Research
Published in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

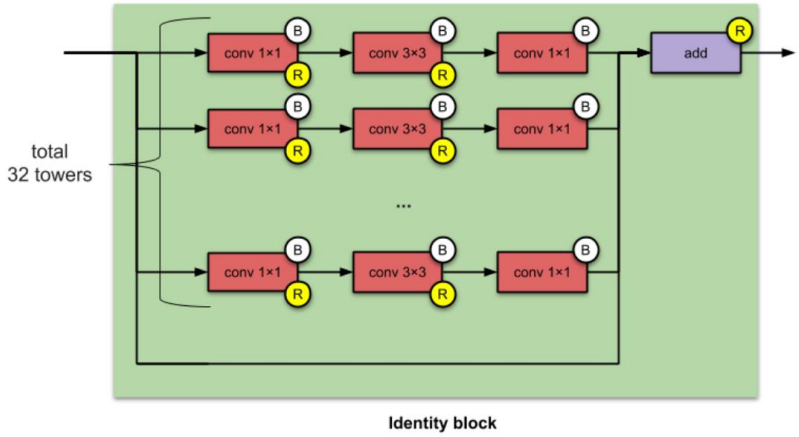
It scales up the number of parallel towers (“cardinality”) within a module:



This net has 25M parameters.



Conv block



Identity block

[https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d]

GitHub sources



It is not necessary to implement all these networks from scratch, but use the original sources available on GitHub repositories:

1. Find the source at GitHub.
2. Copy the source at GitHub repository.
3. Clone it in your computer:
   > git clone https://github.com/...
4. Go to the repository, e.g.: cd deep-residual-networks
5. Go to the prototxt/more and look at the structure of the chosen network.

When implementing a selected type of the network, we usually use open-source implementations available on GitHub and adapt it to our tasks.

In the same way, we copy implementations with trained parameters when we want to use transfer learning.

Transfer Learning and some Tricks



Training of big deep learning architectures can take weeks on many GPU units, so it is wise to use some open-source networks that are already trained on big datasets, and next, retrain the network adapting it to the given task.

Such weights might be a very good initialization of the weights of your network.

This way is usually faster than training the network from scratch.

This is called a transfer learning.

When transferring the network with weights, we can freeze some number of the first layers (not changing their weights during the following training), and just train only parameters of e.g. soft-max layer or a few last layers more.

We can also unfreeze some layers during the training when not achieving good-enough results. We can also freeze the fewer number of first layers.

To make training faster, we can compute output values of the last frozen layer, save them to the disk, and use them instead of original inputs to train the last unfrozen layers only.



Let's start using convolutional networks!









- ✓ Questions?
- ✓ Remarks?
- ✓ Suggestions?
- ✓ Wishes?



Bibliography and Literature

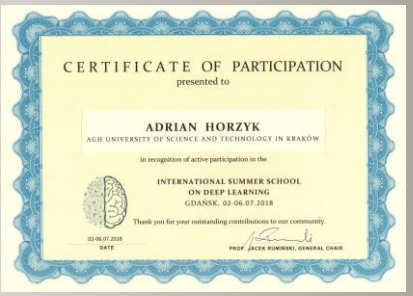
1. https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d
2. https://arxiv.org/abs/1602.07261 - Inception v4
3. Nikola K. Kasabov, *Time-Space, Spiking Neural Networks and Brain-Inspired Artificial Intelligence*, In Springer Series on Bio- and Neurosystems, Vol 7., Springer, 2019.
4. Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016, ISBN 978-1-59327-741-3 or PWN 2018.
5. Holk Cruse, *Neural Networks as Cybernetic Systems*, 2nd and revised edition
6. R. Rojas, *Neural Networks*, Springer-Verlag, Berlin, 1996.
7. *Convolutional Neural Network* (Stanford)
8. *Visualizing and Understanding Convolutional Networks*, Zeiler, Fergus, ECCV 2014
9. IBM: https://www.ibm.com/developerworks/library/ba-data-becomes-knowledge-1/index.html
10. NVIDIA: https://developer.nvidia.com/discover/convolutional-neural-network
11. JUPYTER: https://jupyter.org/



Adrian Horzyk

horzyk@agh.edu.pl

Google: Horzyk



University of Science and Technology in Krakow, Poland