BASICS OF NEURAL NETWORKS

Session: CNNs fundamentals

credits

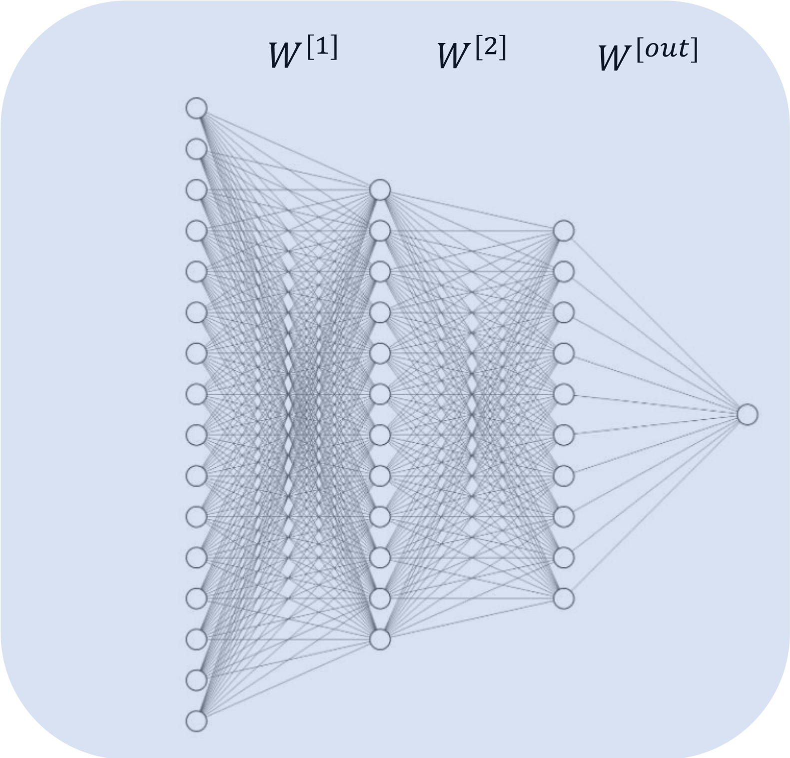
2025 SCHOOL AT THE IAA-CSIC

EDUARDO SÁNCHEZ KARHUNEN

DEPT. ARTIFICIAL INTELLIGENCE. UNIV. SEVILLE. SPAIN

The Problem with MLPs for Images

- Their strength becomes a weakness:
  - MLPs works well for very small images (e. g. MNIST).
- Fail in modern images:
  - They are 1D: requite a flat vector as input.
  - Parameter explosion:
    - For a image resolution of 12 Mpixels (e. g. $12 \cdot 1024 \times 1024$)
    - Input layer: $n_{input} \approx 12 \cdot 10^6$ neurons.
    - hidden_1: $n_1 = 10^4$ neurons.
    - $\#\mathbf{W}^{[1]} \approx 12 \cdot 10^6 \cdot 10^4 \approx 10^{10}$

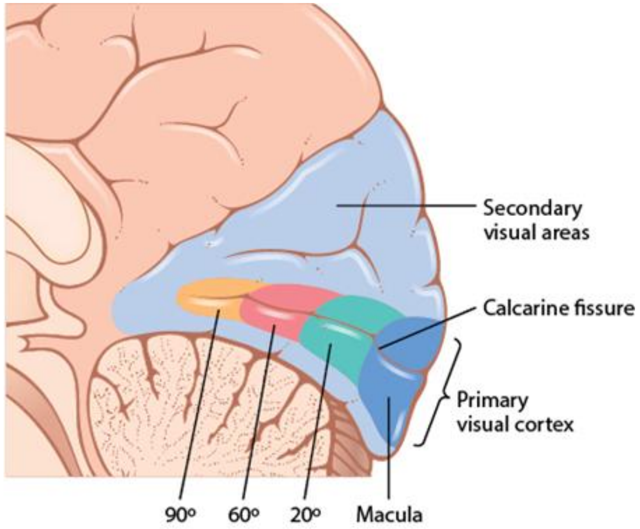


credits

A new architecture is needed: respect 2D images and is far more parameter-efficient

A New Inspiration: The Visual Cortex

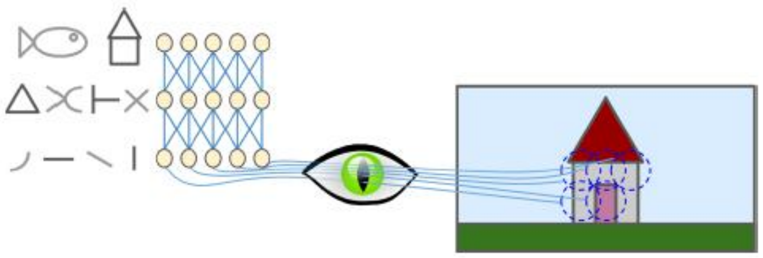
- Once again: look to the brain
  - Perception is processed in zones non-related to conscious activity.
    - We “sense” unconsciously.
    - We cannot express it using rules. e. g. find a cat in an image.
  - Specialized areas in the brain for each sense:
    - Visual cortex, Auditive cortex, …
  - Successive brain areas form a preprocessing pipeline:
    - Before reaching brain areas of conscious activity.
  - There are brain zones that combine information from different senses.



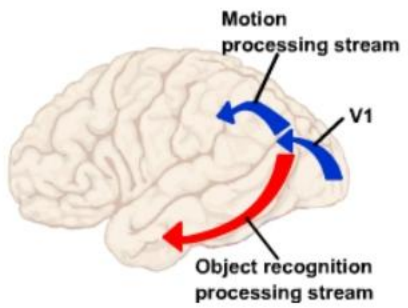
credits

Visual Cortex Neurophysiology

- Hubel & Torsten (Nobel Prize ‘81)
  - Local receptive fields:
    - Mapping retina -> cortex: small neuron groups “see” specific regions of the visual field.
    - Overlapping fields cover the entire visual field.
  - Simple feature detection: some neurons specialized to detect very simple patterns.
    - E. g. vertical lines, horizontal lines, orientations.
  - Hierarchical composition: Other groups of neurons react to combinations of simple patterns.
    - E.g. shapes, textures, objects.

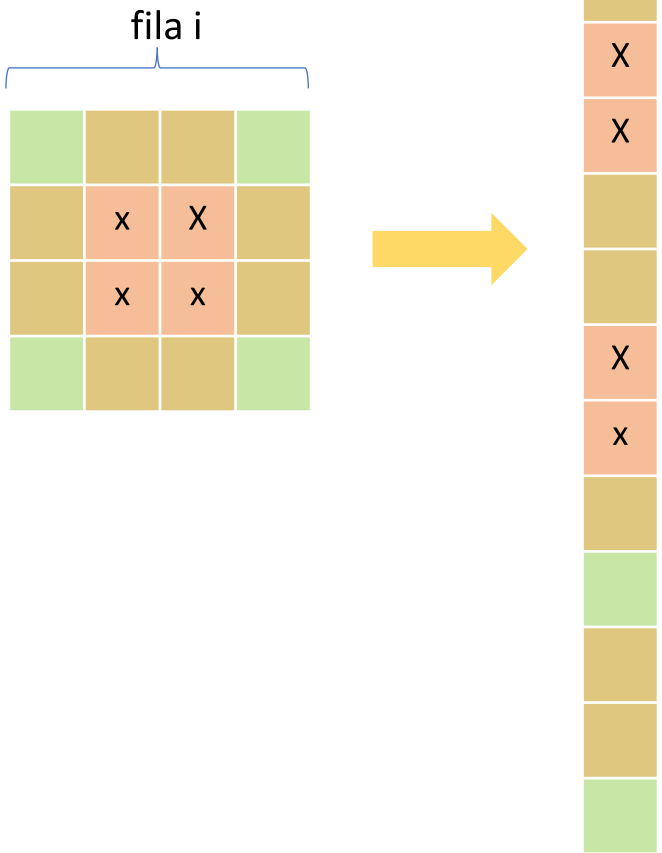


credits



credits

- Constraints on architectures (LeCun, NY, 1998):
  - Idea:
    - We have, a priori knowledge of the task.
    - Constraints on model architecture, based on this knowledge.
  - Expected impact:
    - A significant reduction of parameters.
    - Improve of model generalization capacity.

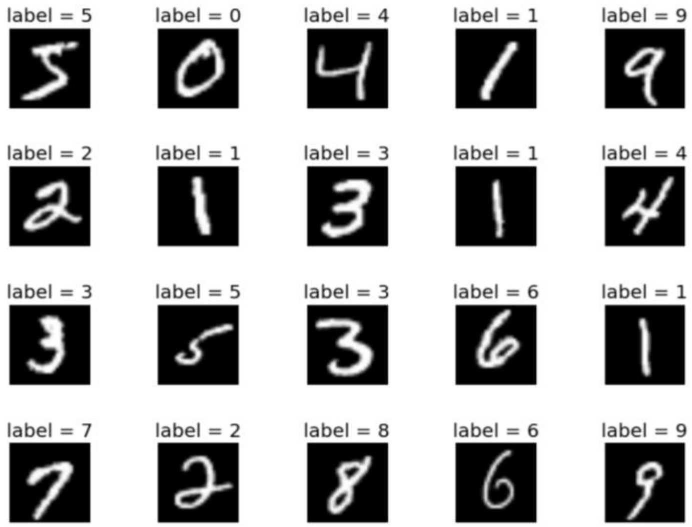


Applicable in images?

- Flattening -> loss of neighborhood info.
- It makes sense to define a grid.
- Good idea to extract local features and combine them.

First CNN: LeNet-5

- LeNet-5 (LeCun, 1989-1998):
  - Task: handwritten digit recognition.
  - Released a reference dataset: MNIST.
  - Applied the novel backpropagation technique to images problems.
  - Foundation of current CNNs.



credits

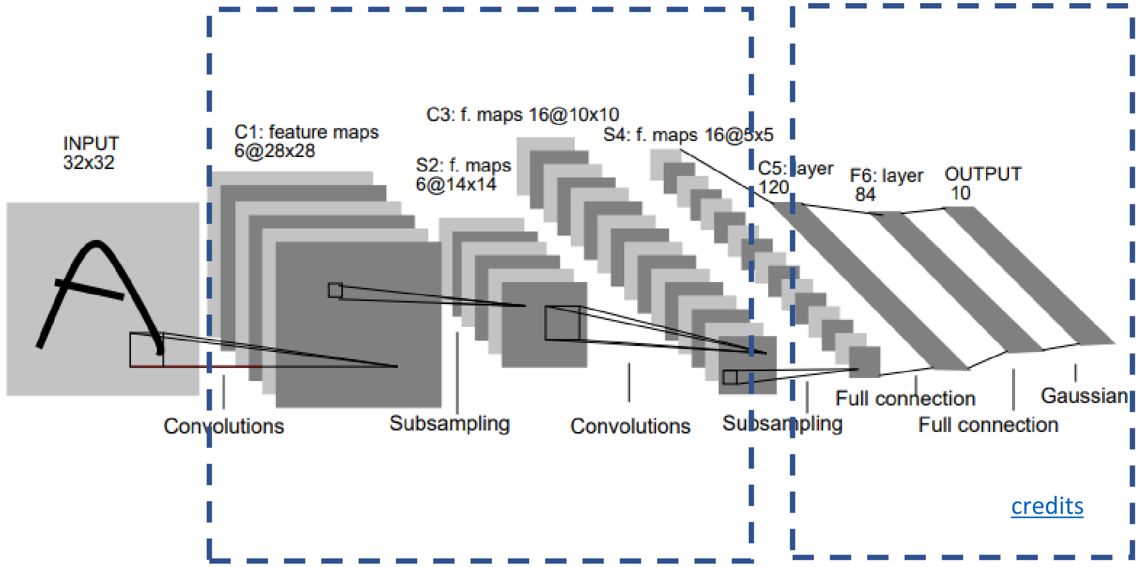
A model ahead of its time:

- Computational power (GPUs) needed wasn't widely available yet.
- When GPUs appeared: explosion of interest in CNNs:
  - In AlexNet (2012): variation of LeNet
  - Won the ImageNet: image recognition challenge.
  - Since then, CNNs the de facto standard for image classification.

The Basic Blocks of a CNN

- Two main parts:
  - A classification block:
    - Standard MLP predicting the class.
    - Based on the flattened 1D-array it receives.
    - Idea: inject in this 1D-array as much info as possible.
  - A new "preprocessing block" before the MLP:
    - Idea: perform a feature extraction in 2D.
    - Inspired in the visual cortex ideas.

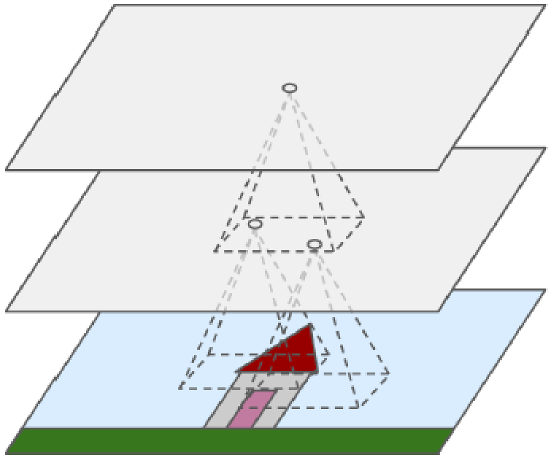
Two new types of layers: **Convolutional layer**
Pooling layer



Preprocessing MLP

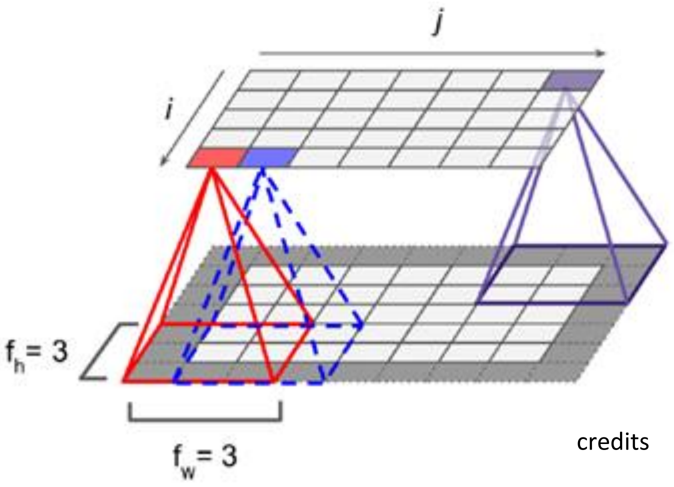
Emulate local receptive fields:

- Goal: avoid losing pixel neighborhood info.
- Perform operations in 2D: no flattening.
- Each neuron of the (conv) layer:
  - Not connected to all pixels (neurons) of the input.
  - Only looks a small local patch (f_W, f_H).



Convolution: Human Interpretation

- All pixels (i, j) of the convolutional layer are traversed.
- The patch is applied and shifted.
- The global resulting effect is a convolution.



credits

Convolution Operation with Kernels

Renaming the 2D operation: convolution

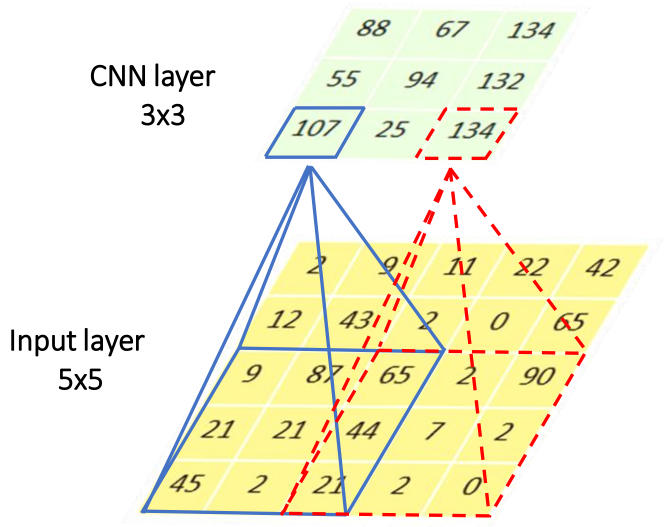
- Input image= matrix of values in range 0-255 (grayscale)
- Convolutional layer: 2D-dense layer with all weights zero except for the concrete patch.
- Kernel (or filter): 3x3, 5x5, 7x7. *Dim(Kernel) << Dim(Imagen)*
- Convolution: $H = I \otimes K$, element-wise product of both matrices and a final sum of all its elements.
  - Output layer: Feature Map.

$$\begin{bmatrix} 2 & 9 & 11 & 22 & 42 \\ 12 & 43 & 2 & 0 & 65 \\ 9 & 87 & 65 & 2 & 90 \\ 21 & 21 & 44 & 7 & 2 \\ 45 & 2 & 21 & 2 & 0 \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 88 & 67 & 134 \\ 55 & 94 & 132 \\ 107 & 25 & 134 \end{bmatrix}$$

Image: convolutional layer input 5x5

Kernel 3x3

Feature map convolutional layer output 3x3



Convolution Step by Step

Kernel

0	0	1
1	0	0
0	0	1

2_0	9_0	11_1	22	42
12_1	43_0	2_0	0	65
9_0	87_0	65_1	2	90
21	21	44	7	2
45	2	21	2	0

=

convolution

88		

2	9	11	22	42
12_0	43_0	2_1	0	65
9_1	87_0	65_0	2	90
21_0	21_0	44_1	7	2
45	2	21	2	0

=

convolution

88		
55		

2	9	11	22	42
12	43	2	0	65
9_0	87_0	65_1	2	90
21_1	21_0	44_0	7	2
45_0	2_0	21_1	2	0

=

convolution

88		
55		
107		

...

2	9_0	11_0	22_1	42
12	43_1	2_0	0_0	65
9	87_0	65_0	2_1	90
21	21	44	7	2
45	2	21	2	0

=

convolution

88	67	
55		
107		

...

The Purpose: Feature Extraction

⊗

Identity kernel

0	0	0
0	1	0
0	0	0

=

Feature map 1

⊗

Outline kernel

-1	-1	-1
-1	8	-1
-1	-1	-1

=

Feature map 2

⊗

Sobel left

1	0	-1
2	0	-2
1	0	-1

=

Feature map 3

⊗

Sobel bottom

-1	-2	-1
0	0	0
1	2	1

=

Feature map 4

⊗

Lighten kernel

0	0	0
0	2	0
0	0	0

=

Feature map 5

⊗

Darken kernel

0	0	0
0	0,5	0
0	0	0

=

Feature map 6

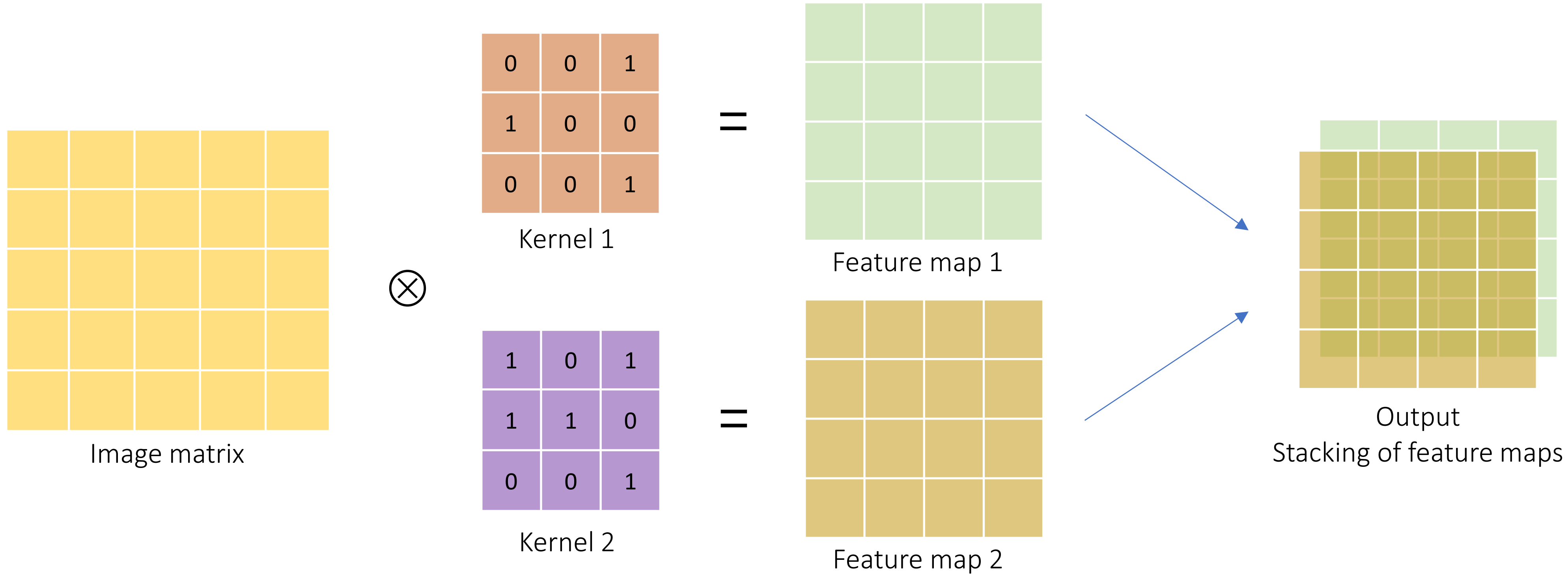
credits

https://setosa.io/ev/image-kernels/

Stacking of Feature Maps

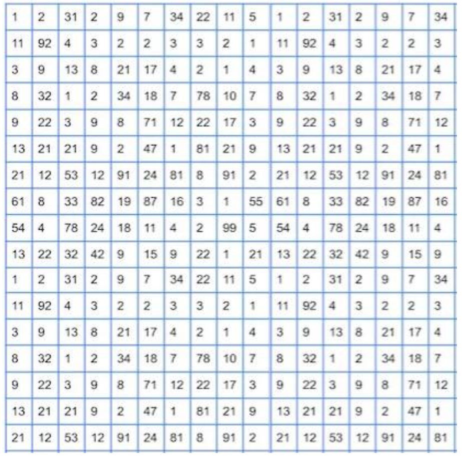
- More than one kernel can be applied:
  - 1 kernel → 1 feature map.
  - Apply as many kernels as needed.
  - Each kernel has its own weights

NN learns the weight during training to extract useful features

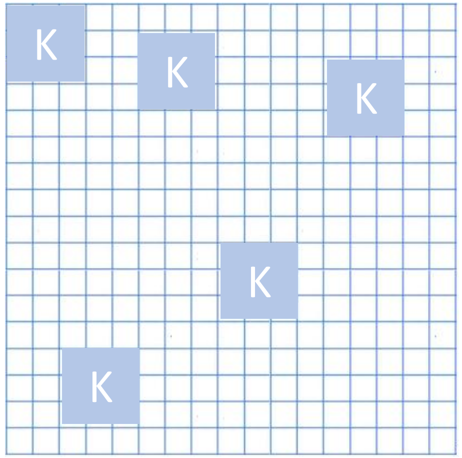


Superpowers of Convolutional Layers

- ## Sparse weights:
  - Dense layers: neuron “has weights with” every neuron in the prev. layer.
  - Conv layers: neuron has only weights to a patch of the previous layer.



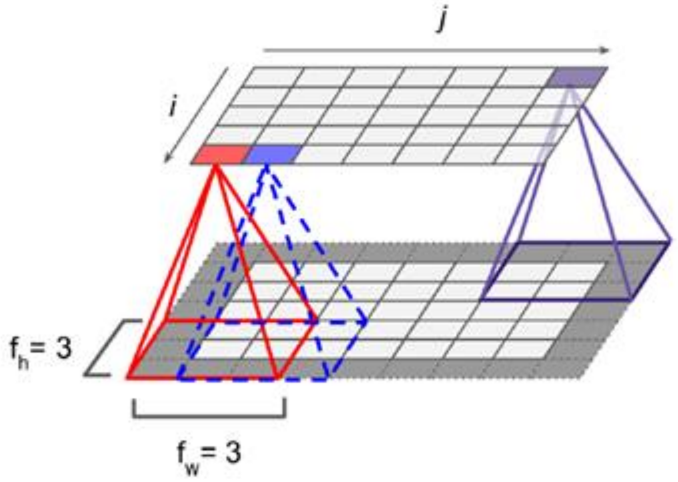
- ## Parameter (weights) sharing:
  - Dense layers: In general, weight values are different.
  - Conv layers: weights are shared by all neurons in the layer.



- ## Equivariance to translation:
  - Kernel can detect the feature in any area of the image (under translation).
  - Non-equivariance under other transformations (e. g. rotation).

Convolutional Layer Parameters: Padding

- Convolution alters image shape:
  - No-padding: loss of $(\boldsymbol{f_H} - \mathbf{1})$ rows and $(f_W-\mathbf{1})$ columns.
  - Zero-padding:
    - Add zeros around the border of the image.
    - Goal: maintain original image size after convolution.



2_0	9_0	11_1	22	42
12_1	43_0	2_0	0	65
9_0	87_0	65_1	2	90
21	21	44	7	2
45	2	21	2	0

5x5 + padding = No
(in TF, *valid*)

=

88		

3x3

0_0	0_0	0_1	0	0	0	0
0_1	2_0	9_0	11	22	42	0
0_0	12_1	43_0	2	0	65	0
0	9	87	65	2	90	0
0	21	21	44	7	2	0
0	45	2	21	2	0	0
0	0	0	0	0	0	0

5x5 + padding = yes
(in TF, *same*)

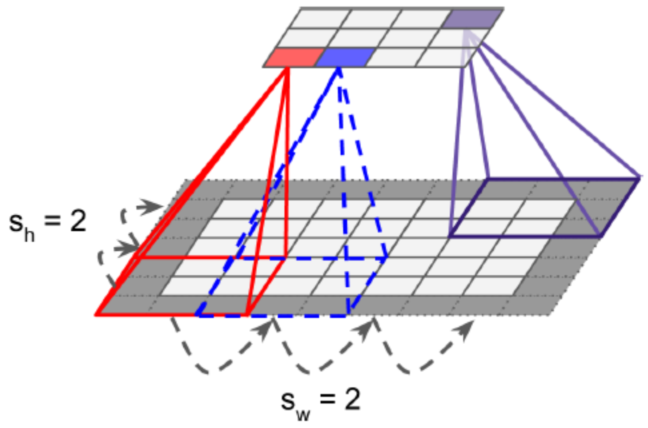
=

12				
	88			

5x5

Convolutional Layer Parameters: Stride

- Control kernel "jumping":
  - Typically, stride = 1.
  - If stride > 1, output size < original image size.
  - If necessary, S_V: vertical stride ≠ S_H: horizontal stride.



0_0	0_0	0_1	0	0	0	0
0_1	2_0	9_0	11	22	42	0
0_0	12_1	43_0	2	0	65	0
0	9	87	65	2	90	0
0	21	21	44	7	2	0
0	45	2	21	2	0	0
0	0	0	0	0	0	0

=

Stride = 2

12		

0	0	0	0	0	0	0
0	2	9	11	22	42	0
0_0	12_0	43_1	2	0	65	0
0_1	9_0	87_0	65	2	90	0
0_0	21_1	21_0	44	7	2	0
0	45	2	21	2	0	0
0	0	0	0	0	0	0

=

Stride = 2

12		
64		

From Matrices to Tensors

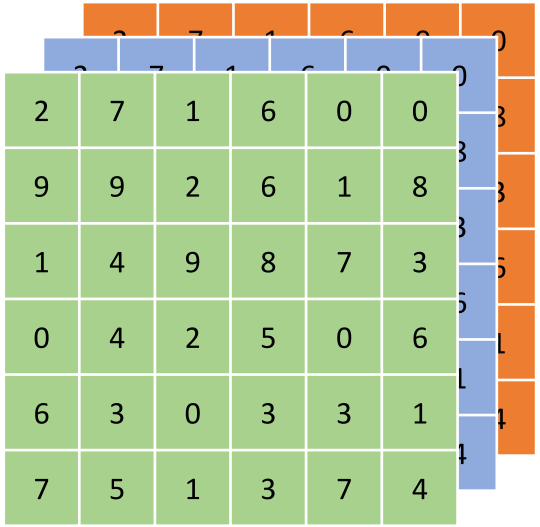
Images representation:

- **Grayscale image:** 1 x 2D-array.
- **Color image:** 3 x 2D-arrays (or channels).
  - 1 channel for each color: Red (R), Green (G), Blue (B).
- Extra channels depending on the problem:
  - E. g. satellite images: multiple additional infrared channels.

Grayscale (2D-Tensor)

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

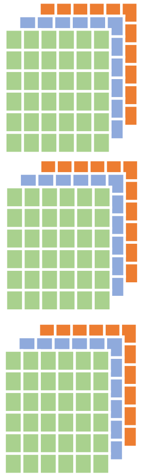
Colour-3 channels (3D-Tensor)



In NN frameworks:

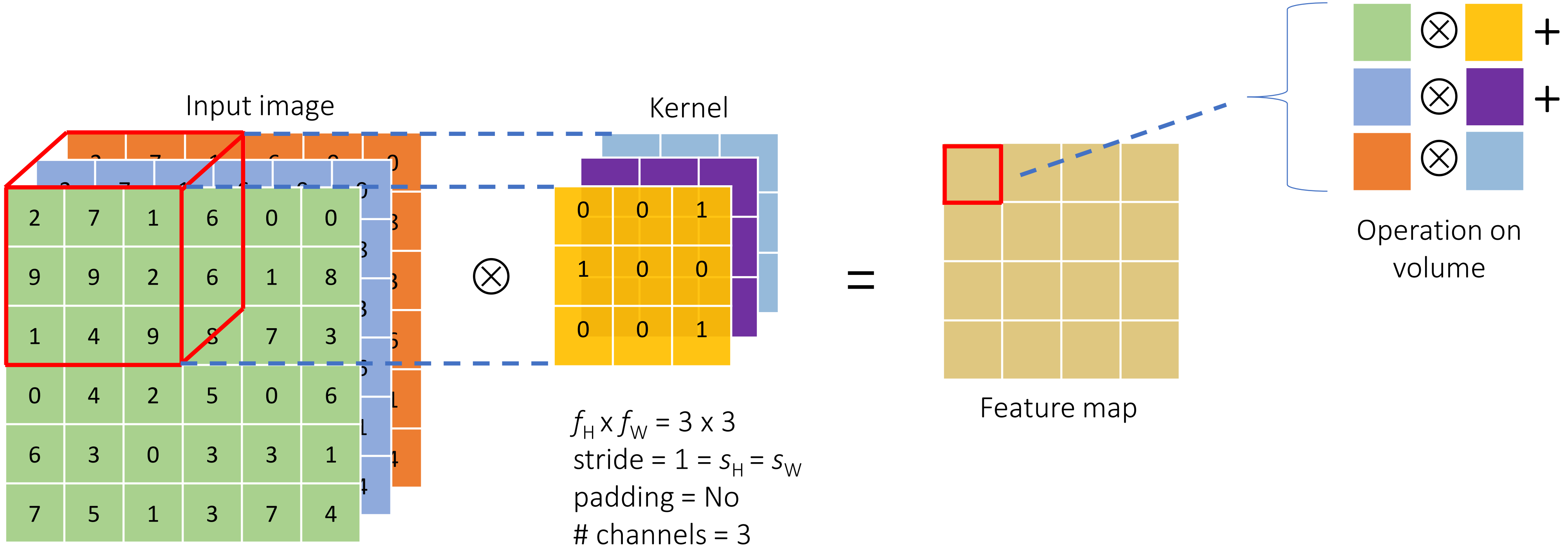
- 3D-tensors: [*height, width, channels*]
- Mini-batch: [*batchsize, height, width, channels*]

Mini-batch

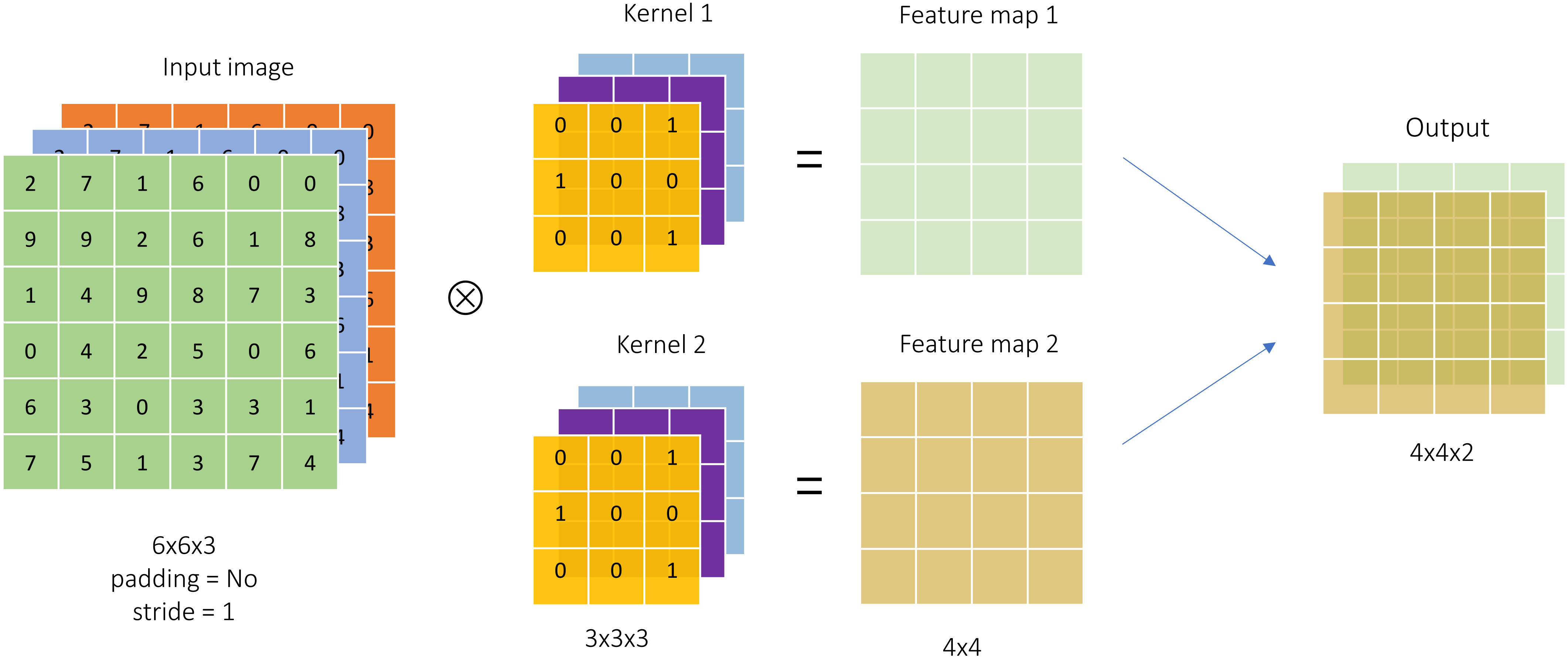


Convolution with Multiple Channels

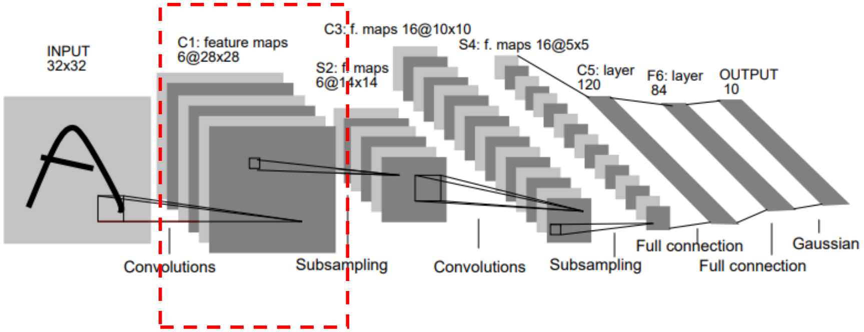
- Convolutions on volume:
  - #channels image = #channels kernels
  - Σ(each kernel "channel" acts on its associated image channel).
  - Output = 1 feature map.



Multiple Channels & Multiple Kernels



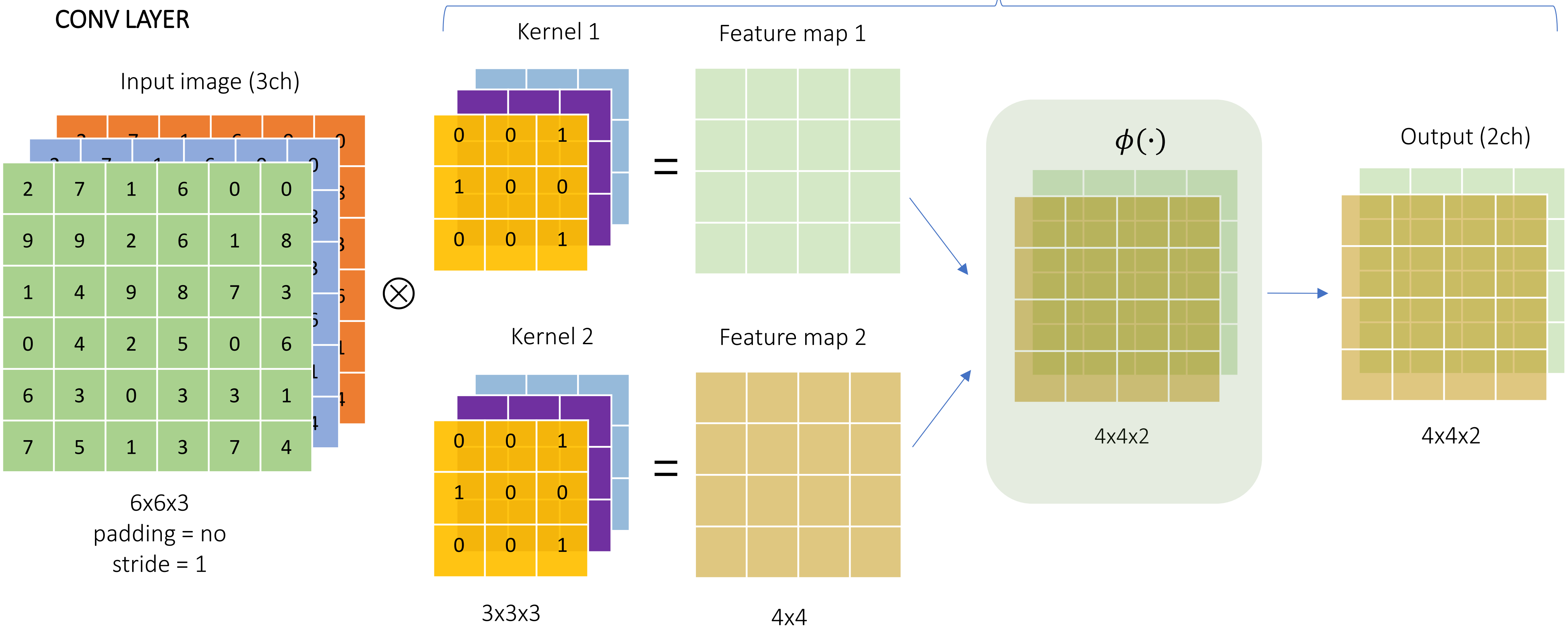
Convolutional Layer: Whole Picture



CONV LAYER

Training: learn kernel weights

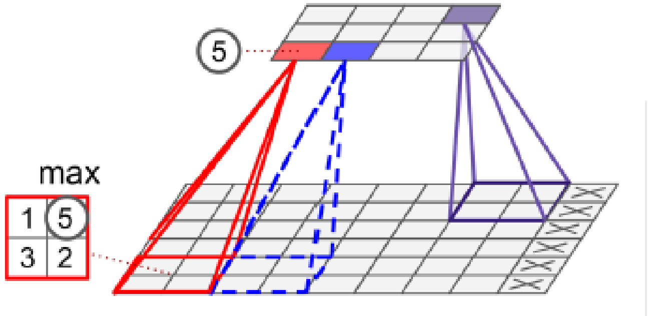
CONVOLUTIONAL LAYER (extracting 2 features)



Pooling Layer

Subsampling:

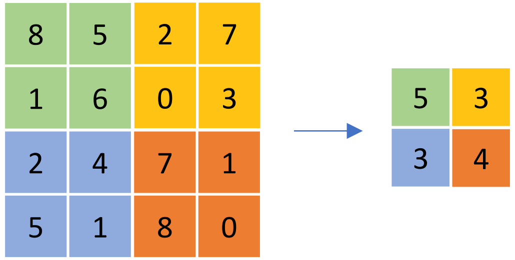
- Goal: reduce spatial dimensions of feature maps.
  - Smaller and more robust.
- Operation: similar to a convolution. Slide an aggregator window.
- Applied separately to each channel of a feature map.
- # input channels = # output channels.



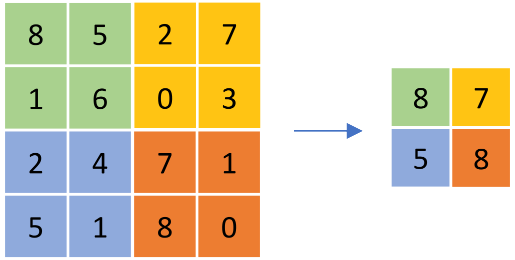
Layer parameters:

- No-trainable weights.
- Stride > 1, to reduce output size. Typ. stride = 2.

Avg Pooling

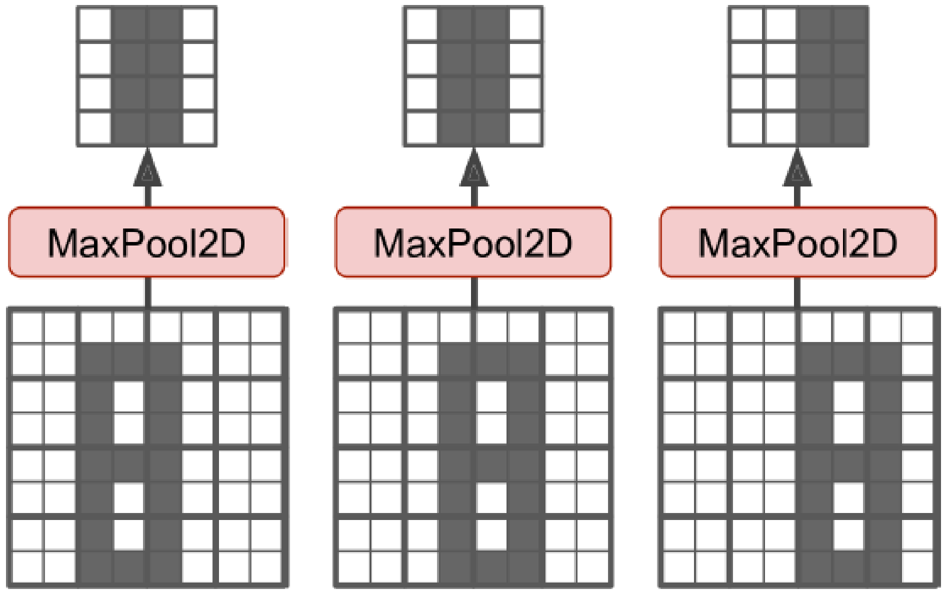


Max Pooling



Types of Pooling

- Common pooling layers:
  - Older: AvgPool2D.
    - Averaging: less info is lost.
  - Most common: MaxPool2D.
    - Preserves stronger features.
    - Sends clearer signals to the next layer.
  - Lately: GlobalAvgPool2D.
    - Highly destructive.
    - Return a unique value. Not a feature map.
    - Useful as an output layer.

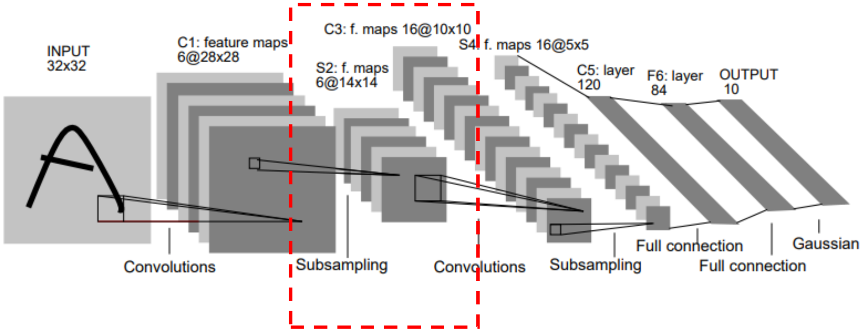


credits

Capture small invariances:

- Translations, rotations and scaling:
  1. Useful in classification problems
  2. Non-useful in segmentation problems

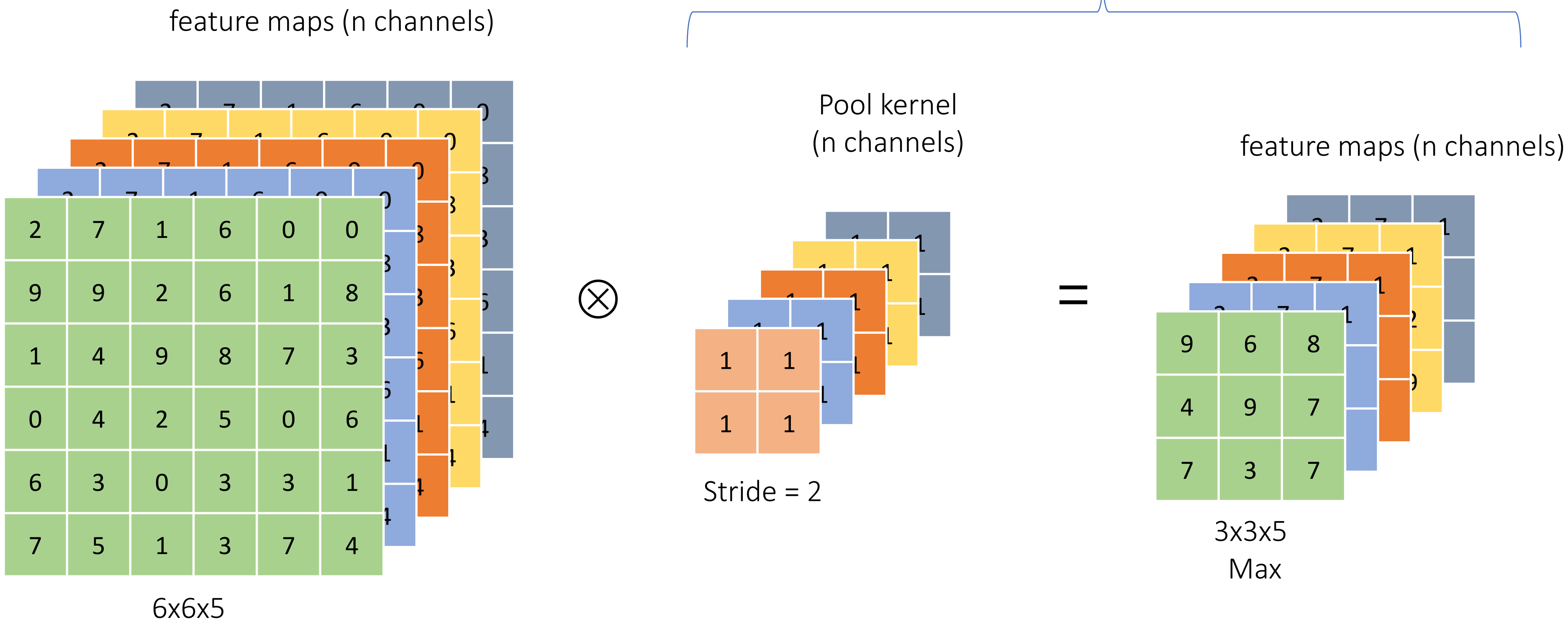
Putting it all Together



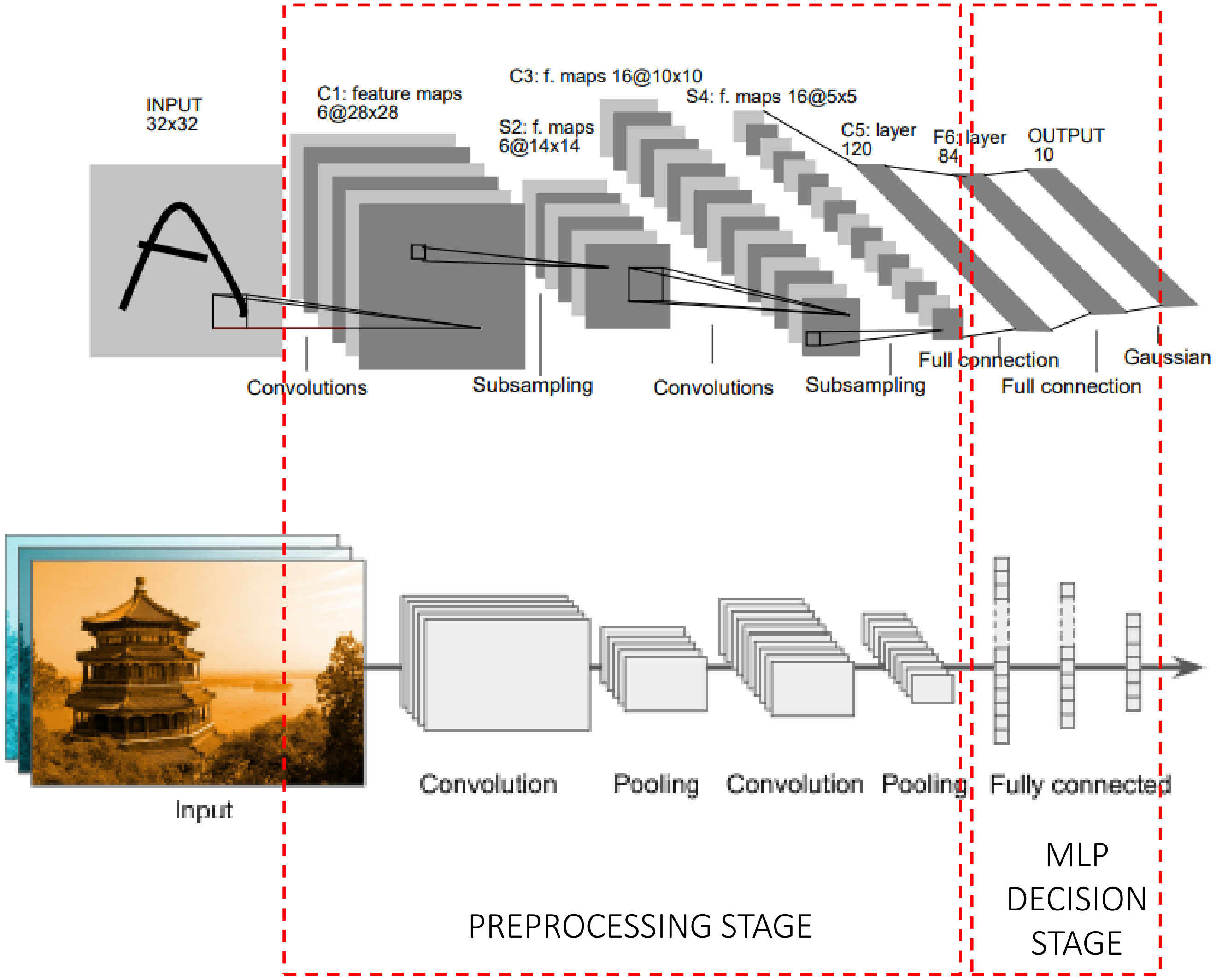
POOLING LAYER

Training: nothing to learn. All weights = 1

POOL LAYER: SUBSAMPLES 2:1



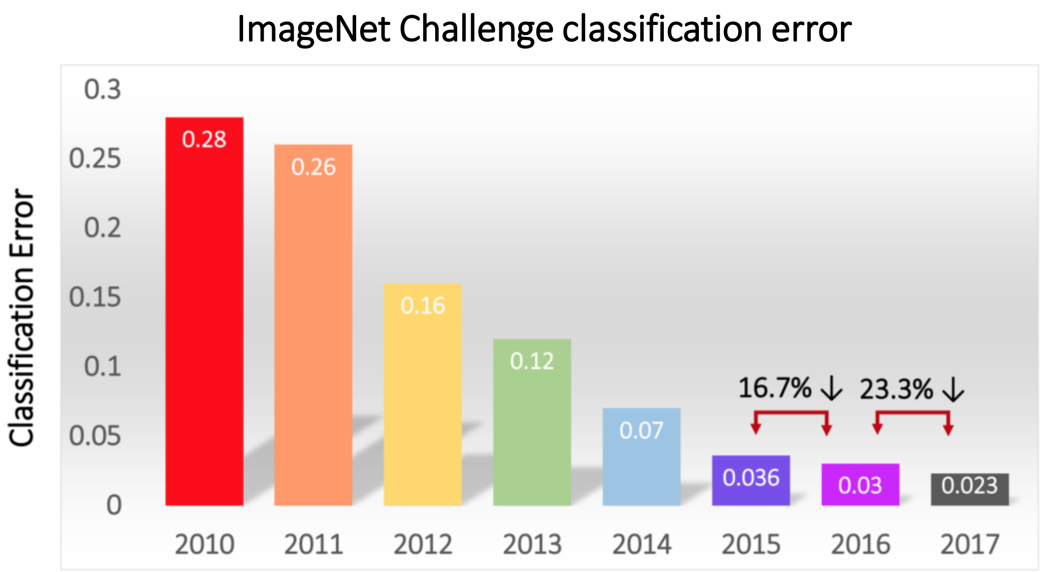
A Typical CNN Architecture



- **Stacking layers:**
  - Combining conv & pool layers
  - Typ: $(\text{Conv} + \text{pool})_1$ ..... $(\text{Conv} + \text{pool})_2$
- **As you progress:**
  - Spatial dimensions get smaller.
  - Number of kernels gets larger.

Reference Challenge

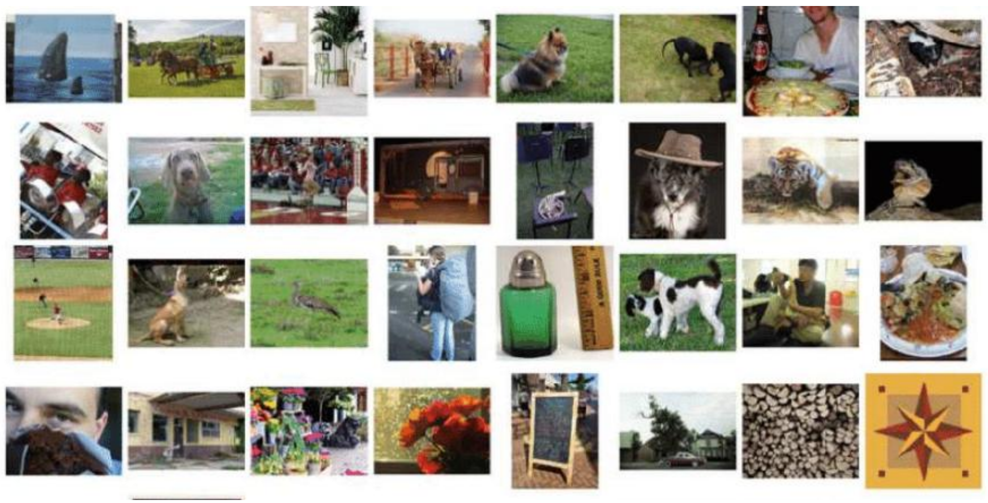
- ImageNET challenge (2010):
  - ILSVRC (ImageNet Large Scale Visual Recognition Challenge)
  - 1.2M images (up to 256 pixels).
  - Classification problem.
  - #clases = 10.000



credits

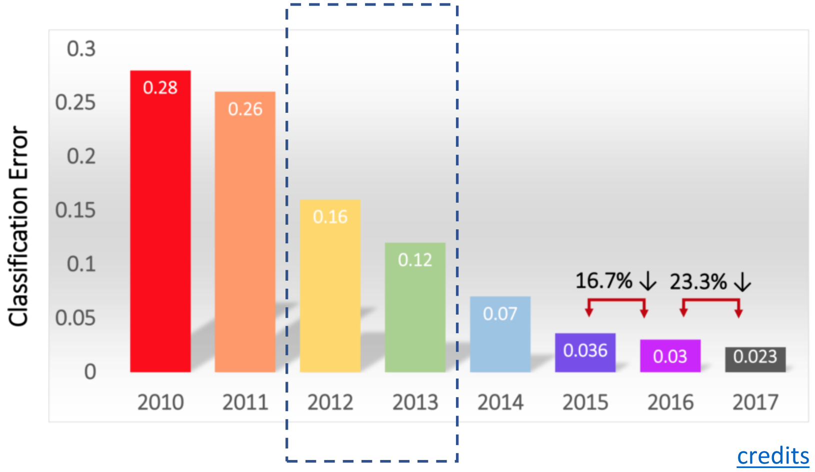
Winner model: top-5 error rate

- Classifier outputs probability of each class.
- Classes with top-5 probabilities are selected.
- % times correct class is not among them.
- Also exists top-1 rate.



AlexNet (2012, error 16%):

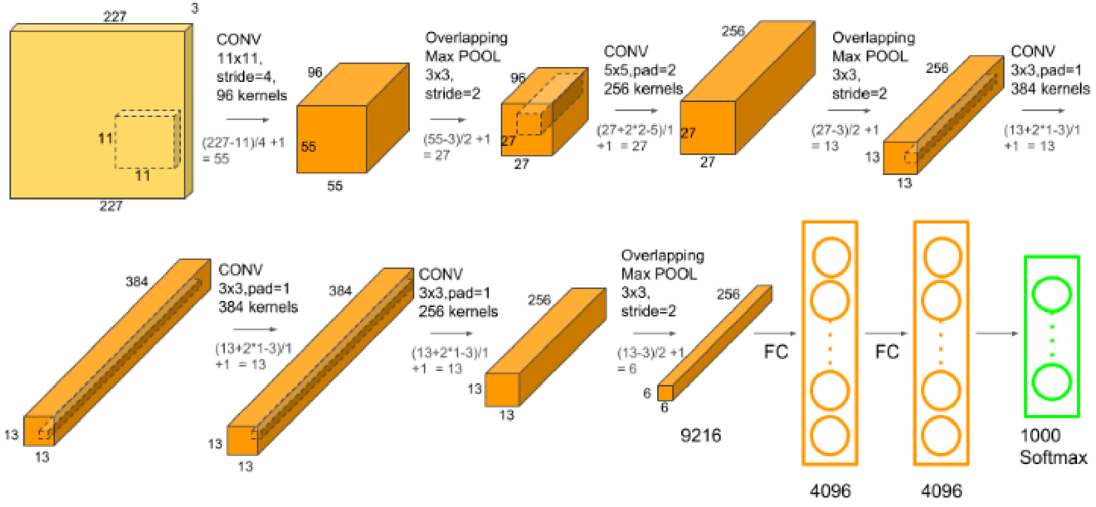
- Based on LeNet-5.
- First conv stacking without pooling.
- # layers = 5 conv + 3 dense
- # params = 60M
- Dropout 50%.
- Data augm.: brightness, shifting and flipping.



credits

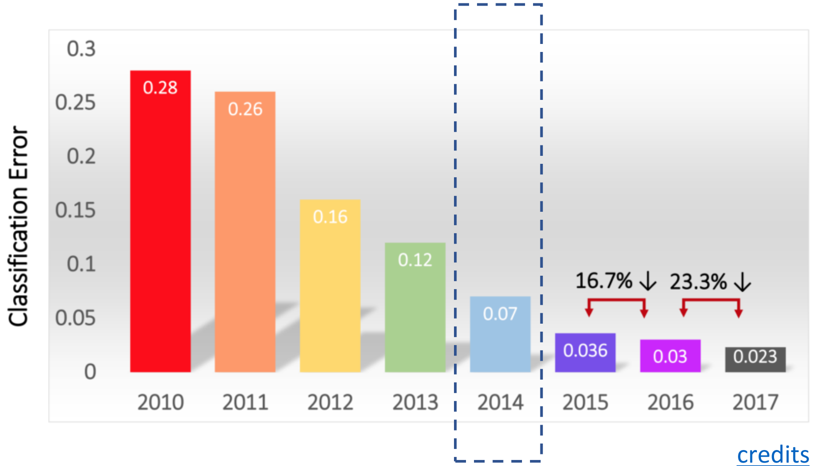
ZF Net (2013, error 12%)

- Hyperparameters tuning:
  - #feature maps
  - kernel size, stride.



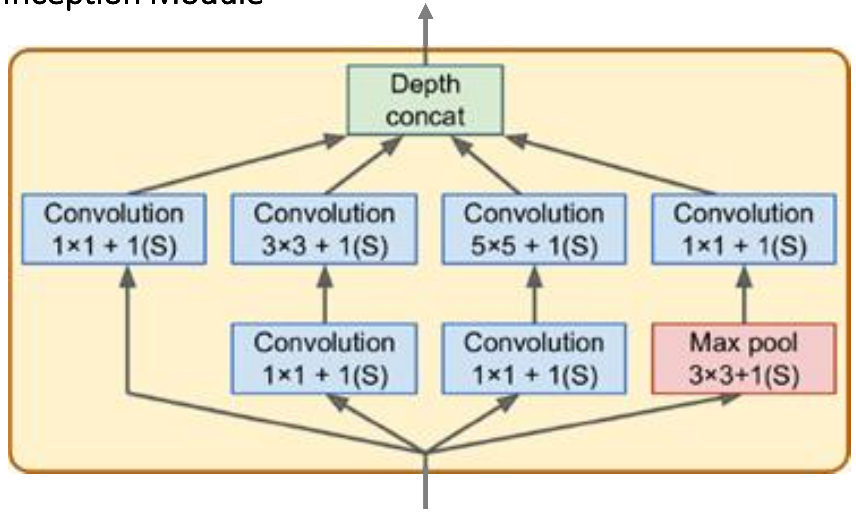
credits

- GoogleNet (2014, error 7%):
    - Innovation: Inception module.
    - # layers = 22 (much deeper)
    - # params = 6M



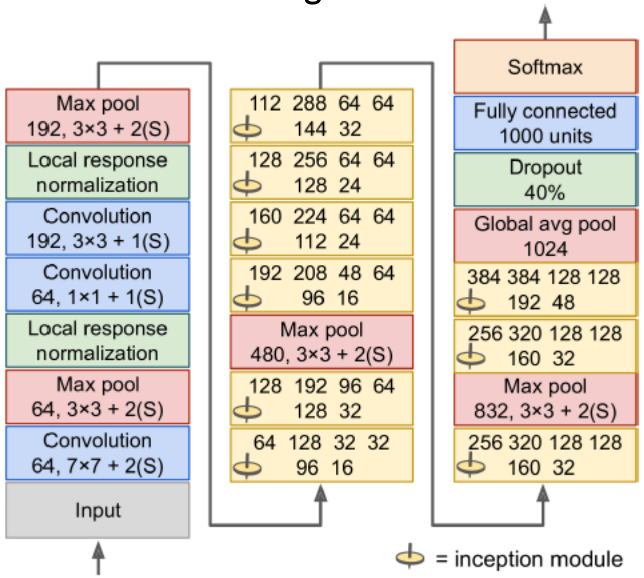
credits

Inception Module



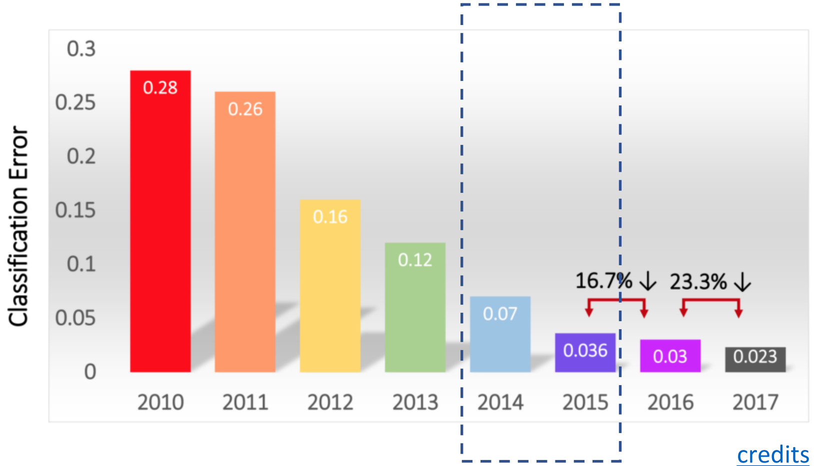
3x3 + 1(S) => Kernel 3x3 + Stride=1 + "Same"

GoogleNet



VGGNet (2014, error 7.3%):

- A very simple classical.
- Stacking de 2 conv + pooling.
- # layers = 16, 19 conv + 2 dense.
- VGG16, VGG19, ...
- # params = 140M.

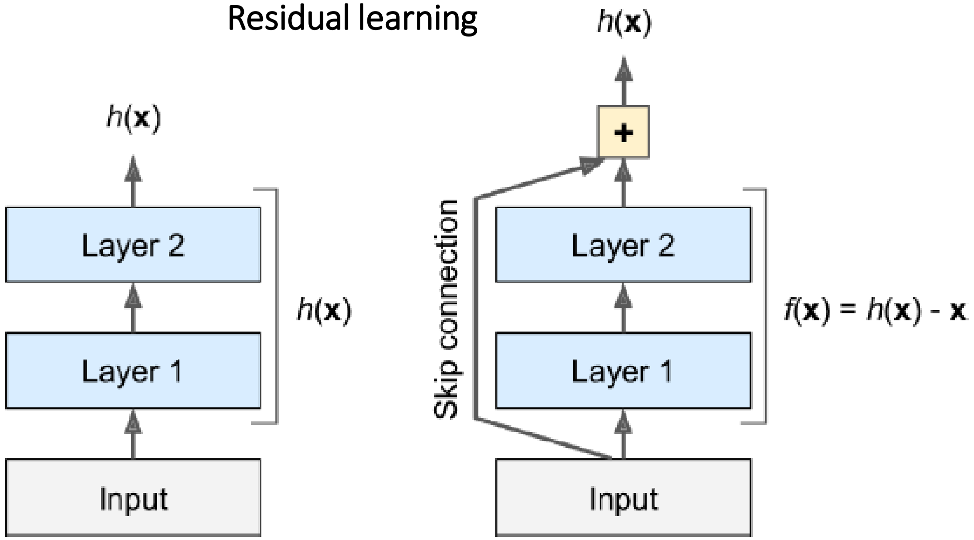


credits

ResNet (2015, error 3.6%):

- Innovation: residual nets.
- # layers = 152.
- # params = 11M .

Residual learning



credits

28