ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

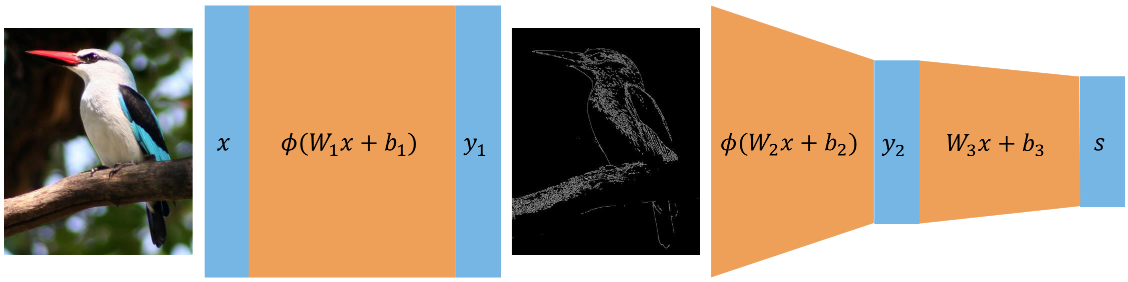
Deep Learning for Computer Vision

Luigi Di Stefano (luigi.distefano@unibo.it)

Department of Computer Science and Engineering (DISI)
University of Bologna, Italy

Limits of FC layers

Let's assume that, to solve the task, the first FC layer would need to detect some kind of local features (e.g. edges, corners, blobs..)



$M=N=224 \;\rightarrow W_1=(3\times M\times N)\times(M\times N)\approx 7.5\times 10^9$

2 Flops (*Multiply&Add*) per param ≈ 15 *Giga Flops*

$M=N=1024 \;\rightarrow W_1 \;\approx 3.2\times 10^{12} \rightarrow \approx$ *6.4 Tera Flops*

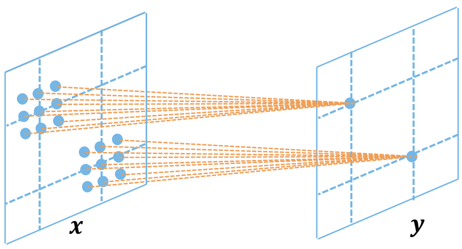
FLOATING POINT OPERATIONS

FC layers require too many parameters and *Flops* to compute simple, local features (unless the image is very small).

Convolution to the rescue

· DON'T FLATTEN IMAGE

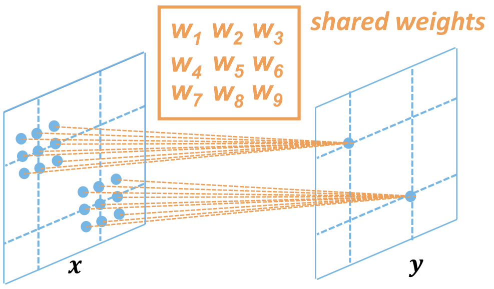
In image processing and classical computer vision we rely on convolution (correlation) to detect local features - as well as larger patterns- in images based on hand-crafted filters (kernels) Similarly, in deep learning we deploy **convolutional layers** to detect features and patters based on **filters learnt by minimizing a loss function**.

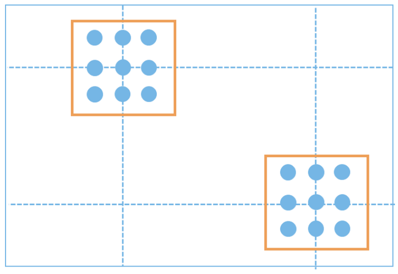


- In a **conv layer** the input and output are not flattened, i.e. they **preserve the spatial (2D) structure of images**.
- Unlike FC layers, in a conv layer each output unit is connected only to a –small- set of neighbouring input units. This realizes a so called **local receptive field**.
- Unlike FC layers, the weights associated with the connections between an output unit and its input neighbours are the same for all output units. Thus, **weights are** said to be **shared**.

- Conv layers embody **inductive biases** dealing with the structure of images: pixels exhibit **informative local patterns** that **may appear everywhere across the image**. → SO WE APPLY THE SAME SET OF WAYS

What a conv layer does compute ?





$$y(i,j) = w_1 x(i-1,j-1) + w_2 x(i-1,j) + w_3 x(i-1,j+1) + w_4 x(i,j-1) + w_5 x(i,j) + w_5 x(i,j+1) + w_7 x(i+1,j-1) + w_8 x(i+1,j) + w_9 x(i+1,j+1)$$

$$w = \begin{bmatrix} w(-1,-1) & w(-1,0) & w(-1,1) \\ w(0,-1) & w(0,0) & w(0,1) \\ w(1,-1) & w(1,0) & w(1,1) \end{bmatrix} \longrightarrow y(i,j) = \sum_{m=-1}^{m=1} \sum_{l=-1}^{l=1} w(m,l) x(i-m, j-l)$$

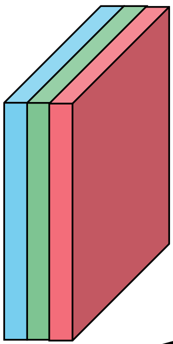
Correlation !

·THERE IS NO FLIPPING OF KERNEL

Multiple input channels

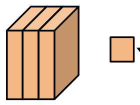
RGB images have 3 channels, so convolution kernels must be 3-dimensional tensors of size $3 \times H_K \times W_K$

Input image



e.g. $\mathbf{3} \times 32 \times 32$

Filter or kernel



e.g. $\mathbf{3} \times 5 \times 5$

2D convolution between **vector-valued functions**.

As usual, we actually compute an affine function, so we also have a ***bias term***

$$[K * I](i,j) = \sum_m \sum_l K\ (m,l) I\ (i-m, j-l)$$

We get one number at each position (i,j) of the input image

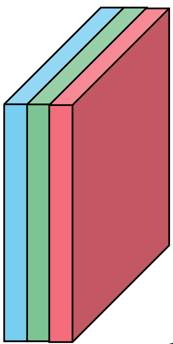
n indexes through the channels of both the image and the filter.

The number of channels (**depth**) of both the input image and the filter must always be the same. Therefore, when specifying the filter, one may omit its depth as it is implicitly defined by that of the input image. For example, a 5×5 filter applied to an RGB image will actually be a $3 \times 5 \times 5$ tensor and will have 76 parameters (75 weights + 1 bias).

Multiple input channels

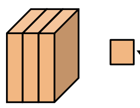
RGB images have 3 channels, so convolution kernels must be 3-dimensional tensors of size $3 \times H_K \times W_K$

Input image



e.g. $\mathbf{3} \times 32 \times 32$

Filter or kernel



e.g. $\mathbf{3} \times 5 \times 5$

2D convolution between **vector-valued functions**.

As usual, we actually compute an affine function, so we also have a ***bias term***

$$[K * I](i,j) = \sum_{n=1}^{3} \sum_m \sum_l K_{\boldsymbol{n}}(m,l) I_{\boldsymbol{n}}(i-m, j-l) + \boldsymbol{b}$$

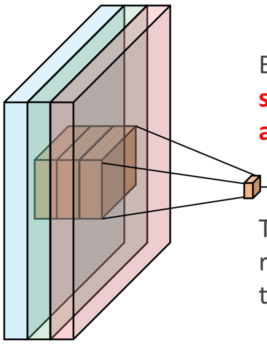
We get one number at each position (i,j) of the input image

n indexes through the channels of both the image and the filter.

The number of channels (**depth**) of both the input image and the filter must always be the same. Therefore, when specifying the filter, one may omit its depth as it is implicitly defined by that of the input image. For example, a 5×5 filter applied to an RGB image will actually be a $3 \times 5 \times 5$ tensor and will have 76 parameters (75 weights + 1 bias).

Output Activations – Feature Maps

$$[K * I](i,j) = \sum_{n=1}^{3} \sum_{m} \sum_{l} K_n(m,l) I_n(i-m, j-l) + b$$

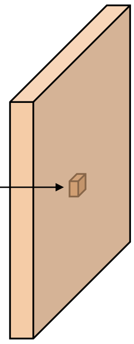


Filter or kernel
e.g. $\mathbf{3} \times \mathbf{5} \times \mathbf{5}$

Input image
e.g. $\mathbf{3} \times \mathbf{32} \times \mathbf{32}$

By sliding the filter over the image, we get a **single-channel** output image, i.e. the **output activation** of the conv layer.

The output activations of conv layers are also referred to as **feature maps** as the learnt filters tend to specialize in detecting specific features/patterns.



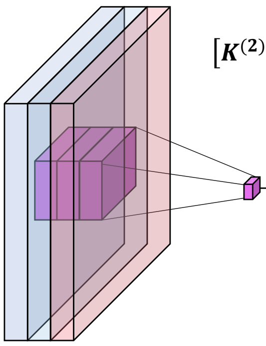
Output activation
e.g. $\mathbf{1} \times \mathbf{28} \times \mathbf{28}$

WE CAN'T COMPUTE CONVOLUTION AT BORDER

Multiple Output Channels (1)

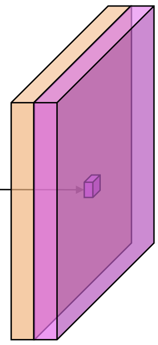
It may be useful to obtain a multi-channel activation by applying different filters with the same size and different weights within the same conv layer. For example, we may deploy two filters such that the conv layer would have the ability to detect two kinds of fatures, e.g. horizontal and vertical edges.

$$\left[\boldsymbol{K}^{(2)} * I\right](j,i) = \sum_{n=1}^{3} \sum_{m} \sum_{l} \boldsymbol{K}_{\boldsymbol{n}}^{(2)}(m,l) I(i-m, j-l) + \boldsymbol{b}^{(2)}$$



Second Filter
e.g. $\mathbf{3} \times \mathbf{5} \times \mathbf{5}$

Input image
e.g. $\mathbf{3} \times \mathbf{32} \times \mathbf{32}$

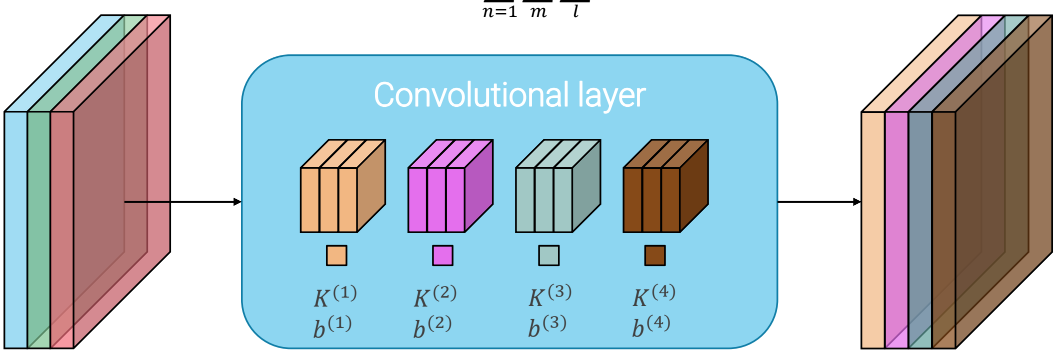


Output activation
e.g. $\mathbf{2} \times \mathbf{28} \times \mathbf{28}$

Multiple Output Channels (2)

If we want an even more powerful conv layer we may apply, e.g. , four filters. The whole operation realized by the layer can be expressed as:

$$[K * I]_{k}(j,i) = \sum_{n=1}^{3} \sum_{m} \sum_{l} \boldsymbol{K}_{\boldsymbol{n}}^{(k)}(m,l) I(i-m, j-l) + b^{(k)} \qquad \boldsymbol{k = 1, \ldots, 4}$$



Input image
e.g. $\mathbf{3} \times \mathbf{32} \times \mathbf{32}$

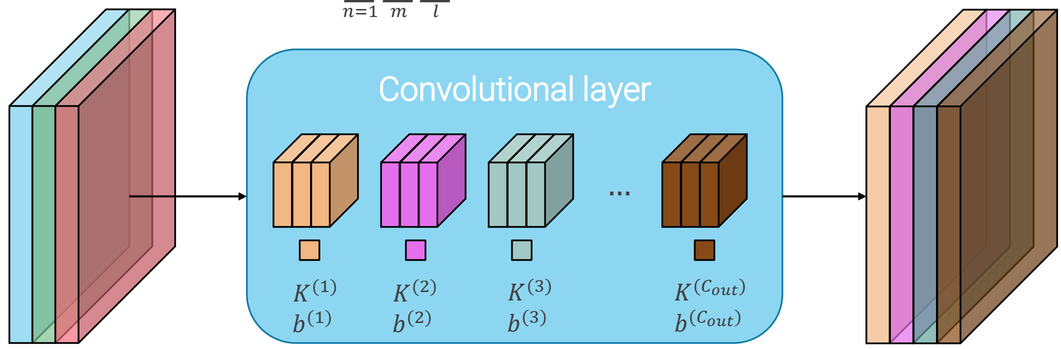
Filters
e.g. $\mathbf{4} \times \mathbf{3} \times \mathbf{5} \times \mathbf{5}$

Output activation
e.g. $\mathbf{4} \times \mathbf{28} \times \mathbf{28}$

General structure of a convolutional layer

A conv layer receives a multi-channel (C_{in}) input activation and produces a multi-channel (C_{out}) output activation by applying as many *vector-valued* filters as the output channels, with the depth of the filters given by the number of input channels.

$$[K * I]_k(j,i) = \sum_{n=1}^{C_{in}} \sum_m \sum_l K_n^{(k)}(m,l) I_n(i-m, j-l) + b^{(k)} \qquad k = 1, \dots, C_{out}$$



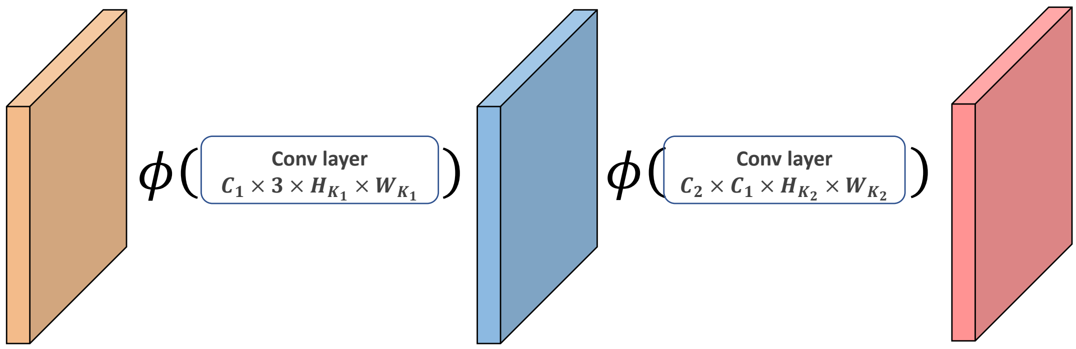
Input activation $C_{in} \times H_{in} \times W_{in}$

Filters $C_{out} \times C_{in} \times H_K \times W_K$

Output activation $C_{out} \times H_{out} \times W_{out}$

Chaining convolutional layers

A convolutional layer is a special form of linear layer (indeed it can be expressed as a matrix multiplication). Thus, to take advantage of depth by chaining multiple layers we need to introduce non-linear activations (typically ***ReLU***). Moreover, to avoid shrinking the activations along thee chain we ***(zero)pad*** the input to each layer.



Input image $3 \times H_{in} \times W_{in}$

First activation $C_1 \times H_1 \times W_1$

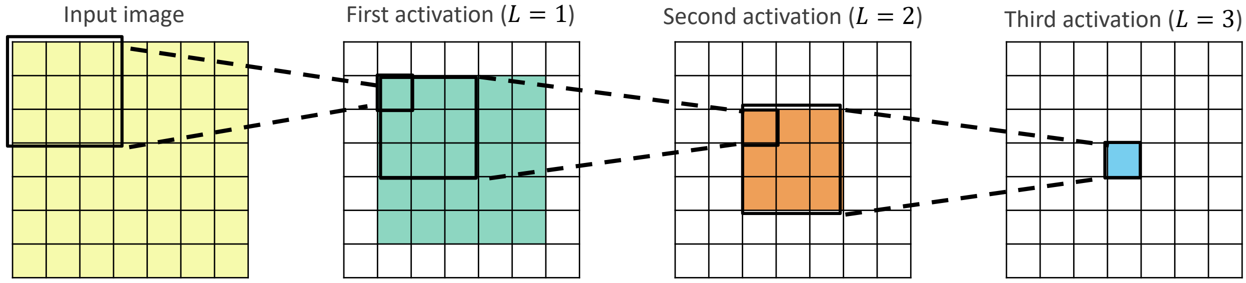
Second activation $C_2 \times H_2 \times W_2$

Receptive Field

The set of input pixels affecting a hidden unit is referred to as the receptive field of the unit. As we traverse a chain of conv layers the receptive field gets larger and larger, so as to compute features dealing with larger and larger image regions (from *local* to *global* features).

E.g., if the kernel size is $H_K \times W_K$, the size of the receptive field at the L-th activation is $[1 + L(H_K - 1)] \times [1 + L(W_K - 1)]$

Input image | First activation ($L = 1$) | Second activation ($L = 2$) | Third activation ($L = 3$)



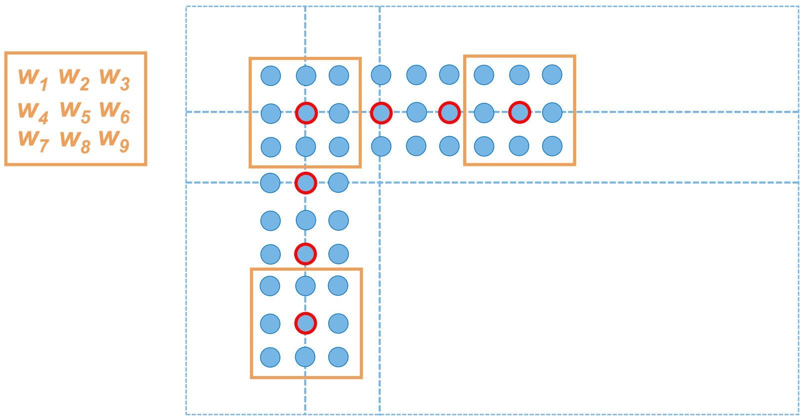
Thus, both the height and width of the receptive field grow linearly with the number of layers. To obtain larger receptive fields with a limited number of layers we down-sample the activations.

Strided Convolution

Rather then densely, convolution may be computed every S (stride) positions in both directions.

-> STRIDE OF 2 (COMMON)

S=2



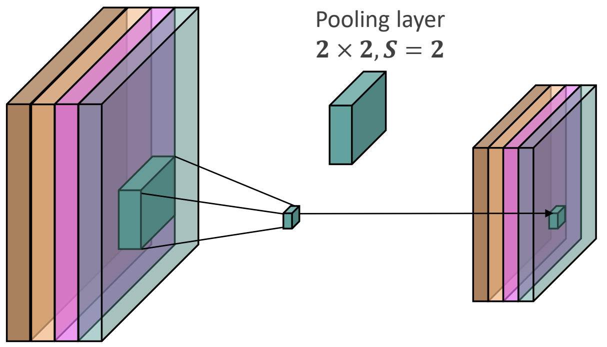
If the input activation is zero-padded according to the size of the filter so to avoid shrinking the output, the actual size of the down-sampled activation computed by a strided convolution is given by:

$$H_{out} = \left\lfloor \frac{H_{in}-1}{S} \right\rfloor + 1$$

$$W_{out} = \left\lfloor \frac{W_{out}-1}{S} \right\rfloor + 1$$

Pooling Layers

Aggregate neighbouring values into a single output by a specific hand-crafted function. The pooling kernel is applied channel-wise and with a stride (**s>1**) to get a down-sampled output.



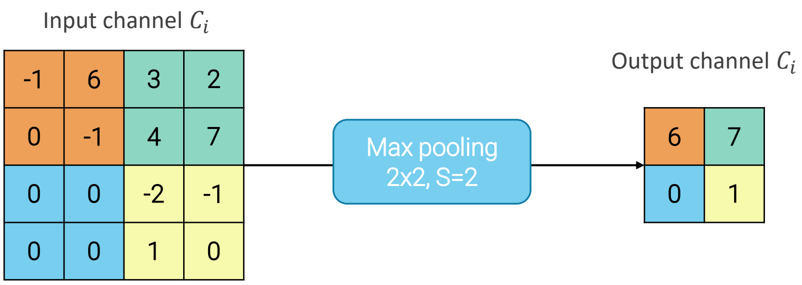
Input activation,
e.g. $\mathbf{4} \times \mathbf{28} \times \mathbf{28}$

Output activation,
e.g. $\mathbf{4} \times \mathbf{14} \times \mathbf{14}$

Hyper-parameters
Kernel width and height: $\boldsymbol{W_K} \times \boldsymbol{H_K}$
Pooling function: max, avg, …
Stride $\boldsymbol{S}$ (>1)

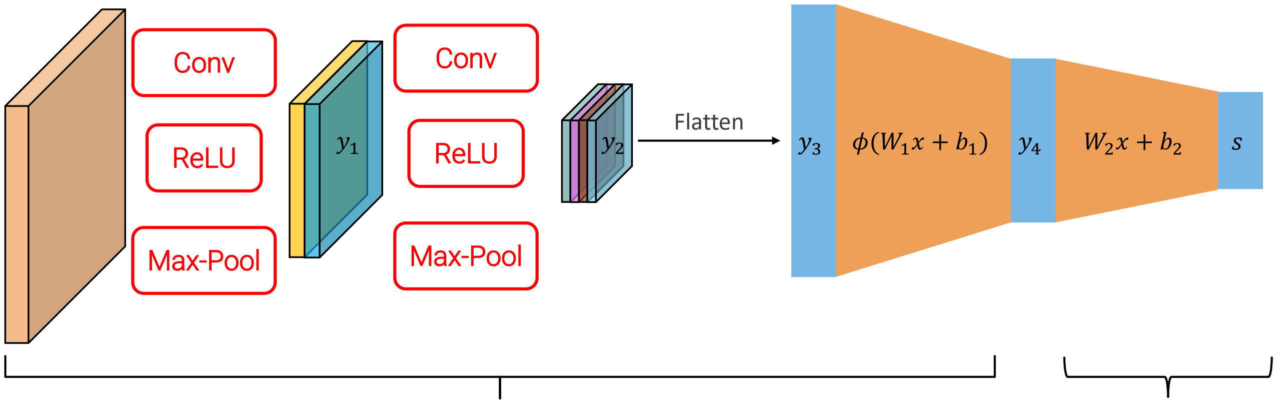
Most common choice:
$2 \times 2,\ S = 2,\ \max$ (Max Pooling)

Max Pooling



Compared to strided convolutions, max pooling
- has no learnable parameters (pro and con)
- provides **invariance** to small spatial shifts.

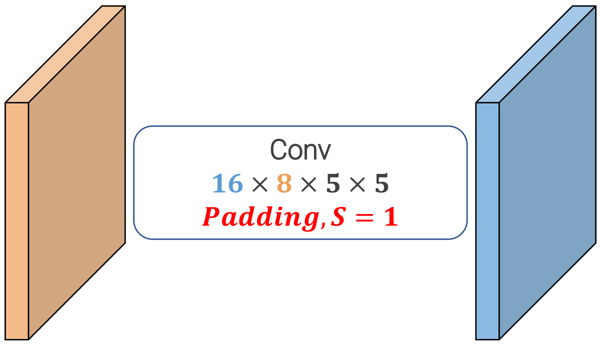
Convolutional Neural Netwoks (CNNs – convnets)



N **Conv+ReLU+Pooling** layers followed by M fully connected layers
This portion of the network is also called **feature extractor.**

The final fully connected layer is also called the **classifier.**

Number of parameters and *Flops* (1)



Input activation
$8\times 32\times 32$

Output activation
$16\times 32\times 32$

- The number of learnable parameters for the Conv layer is
  $16\times(8\times 5\times 5+1)=16\times 201=3{,}216$ *bias*
- Hence, there are

  $$16\times 32\times 32=16{,}384$$

  values in the output activation.
- To compute each output value we take the dot product between *n* weights and the input, which requires *2n* Flops (MAC). Thus, the total number of Flops is:

  $$16{,}384\times 8\times 5\times 5\times 2\cong 6.5\text{M flops}$$

Number of parameters and *Flops* (2)

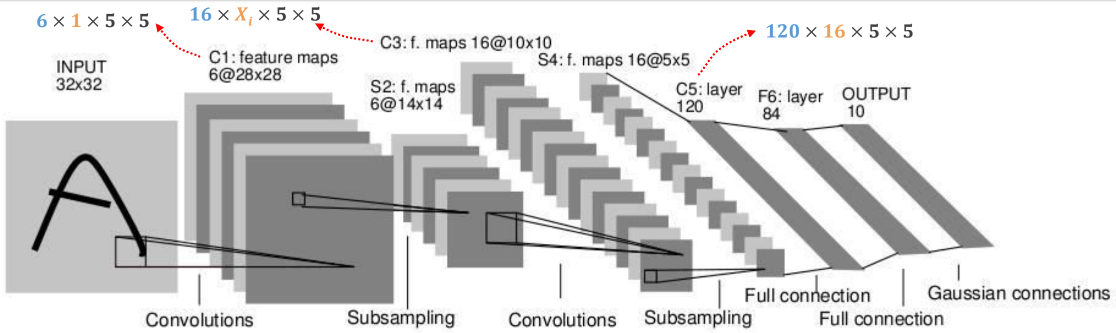
#params = $C_{out}\times(C_{in}\times H_K\times W_K+1)$

Output Channels   Input Channels   Kernel Size

Flops= $2\times(C_{out}\times H\times W)\times(C_{in}\times H_K\times W_K)$

Activation Size

LeNet-5



- Proposed to classify handwritten digits (MNIST dataset) and used in the US to read checks automatically.
- Alongside the layers, the spatial dimension decreases and the number of channels increases. Normalization of inputs (zero mean and unit variance) to accelerate learning.
- 5x5 convolutional kernels, no padding, average pooling (with trainable scale and bias), tanh non-linearities.
- Sparse connection matrix in C3 (convs take input from a subset of input channels as detailed in Tab. 1 of the paper).
- Two fully connected layers: F6, OUTPUT (10 RBF units: each unit compute the distance between its input vector and the corresponding parameter vector).

Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. "Gradient-based learning applied to document recognition", Proceedings of the IEEE. 1998.

AlexNet (1)

- **Won the ILSVRC 2012 bringing the Top-5 error from 25.8 to 16.4.**
- About 60M parameters, trained for 5-6 days on 2 GPUs.
- At training time, random-cropping of 224x224 patches (and their horizontal reflections) from the 256x256 RGB input images and *colour jittering* (**massive data augmentation**).
- At test time, averaging predictions (i.e. *softmax*) across 10 patches (central + 4 corner alongside their horizontal reflections).
- 8 layers with weights (5 Conv + 3 FC): most of the parameters are in the FC layers.
- All layers (Conv and FC) deploy ReLU non-linearities which yield faster training compared to saturating non-linearities (see Fig. 1 in the paper).
- First Conv layer has a stride of 4 (S=4): **stem layer** performing heavy reduction of the spatial size of activations, mainly to reduce memory and computation cost. In all other Conv layers S=1.
- Last FC layer has 1000 units (as many as the ILSVRC classes), the penultimate FC layer is the feature/representation layer and has a cardinality of 4096.

Layer	#Filters/ #Units	Filter Size	S	P	Activation Size
conv1	96	11x11	4	2	55x55
Pool1	1	3x3	2	0	27x27
conv2	256	5x5	1	2	27x27
pool2	1	3x3	2	0	13x13
conv3	384	3x3	1	1	13x13
conv4	384	3x3	1	1	13x13
conv5	256	3x3	1	1	13x13
pool3	1	3x3	2	0	6x6
flatten	0	0	0	0	1x1
fc6	4096	-	-	-	1x1
fc7	4096	-	-	-	1x1
fc8	1000	-	-	-	1x1

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", NeurIPS 2012

AlexNet (2)

Totals: #params ≅ 61M, GFlops ≅ 2.3

overlapping (max) pooling

Layer	#Filters/ #Units	Filter Size	S	P	Activation Size
conv1	96	11x11	4	2	55x55
Pool1	1	3x3	2	0	27x27
conv2	256	5x5	1	2	27x27
pool2	1	3x3	2	0	13x13
conv3	384	3x3	1	1	13x13
conv4	384	3x3	1	1	13x13
conv5	256	3x3	1	1	13x13
pool3	1	3x3	2	0	6x6
flatten	0	0	0	0	1x1
fc6	4096	-	-	-	1x1
fc7	4096	-	-	-	1x1
fc8	1000	-	-	-	1x1

conv1: $96 \times 3 \times 11 \times 11 \rightarrow$ **#params ≅ 35K, MFlops ≅ 211**

conv2: $256 \times 96 \times 5 \times 5 \rightarrow$ **#params ≅ 615K, MFlops ≅ 896**

conv3: $384 \times 256 \times 3 \times 3 \rightarrow$ **#params ≅ 885K, MFlops ≅ 299**

conv4: $384 \times 384 \times 3 \times 3 \rightarrow$ **#params ≅ 1.2M, MFlops ≅ 448**

conv5: $256 \times 384 \times 3 \times 3 \rightarrow$ **#params ≅ 885K, MFlops ≅ 299**

fc6: $4096 \times (6 \times 6 \times 256) \rightarrow$ **#params ≅ 37.5M, MFlops ≅ 75**

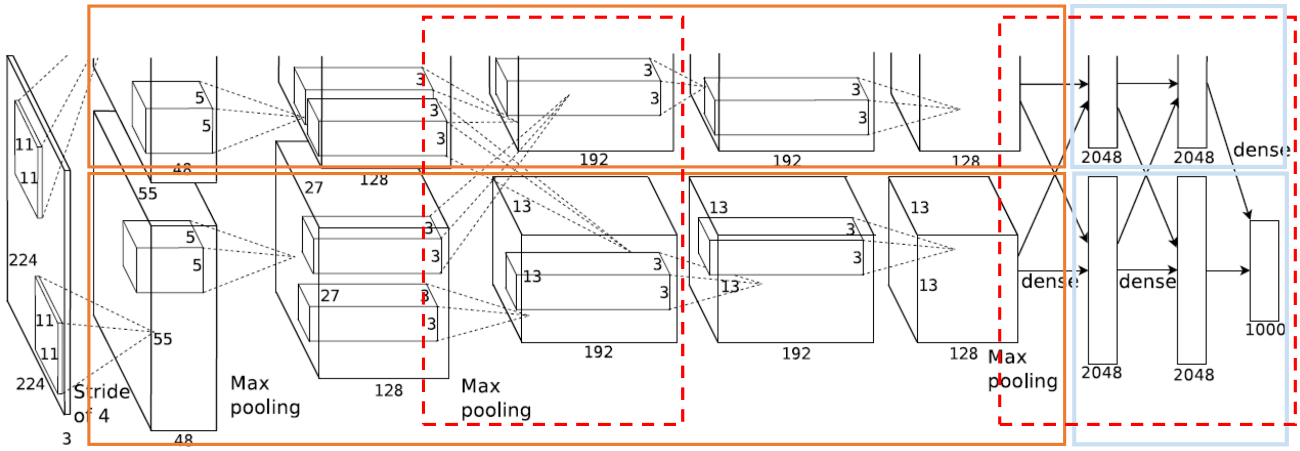
fc7: $4096 \times (4096) \rightarrow$ **#params ≅ 16.7M, MFlops ≅ 33**

fc8: $1000 \times (4096) \rightarrow$ **#params ≅ 4M, MFlops ≅ 8**

- Local Contrast Normalization (after conv1 and conv2): activations are normalized by the sum of those at the same spatial position in a few (*n=5*) *adjacent* channels (mimics *lateral inhibition* in real neurons).
- **Dropout** (fc6,fc7): at training time the output of each unit is set to zero with probability 0.5. This forces units to learn more robust features since none of them can rely on the presence of particular other ones.

AlexNet (3)

In the original implementation the computational load was split between two GPUs



The red boxes with dashed lines highlight layers that take input from both GPUs

VGG

- **Second place in ILSVRC 2014 (Top-5 error: 7.5 %)**
- Explores the benefits of deep and regular architectures based on a few simple design choices:
  - 3x3 conv layers with S=1, P=1
  - 2x2 max-pooling, S=2, P=0
  - #Filters (#channels) double after every pool
- The architecture is designed as a repetition of **stages:** a chain of layers that *process activations at the same spatial resolution* (***conv-conv-pool***, ***conv-conv-conv-pool*** and ***conv-conv-conv-conv-pool***).
  - A **stage** has the ***same receptive field*** as a single larger convolution but, given the same number of input/output channels, introduces ***more non-linearities*** and requires ***less parameters*** and ***less computation***. A stage requires ***more memory*** to store the activations, though.
    - For example, a single $C \times C \times 5 \times 5$ conv layer:
      #params= $C \times (C \times 5 \times 5 + 1)$= $\mathbf{25 \times C^2} + C$
      #Flops=$(C \times W \times H) \times C \times 5 \times 5 \times 2$ = $\mathbf{50 \times C^2 \times W \times H}$
      #activations= $C \times W \times H$
    - while a stage consisting of **2** staked $C \times C \times 3 \times 3$ conv layers (same receptive field):
      #params= $2 \times C \times (C \times 3 \times 3 + 1)$ = $\mathbf{18 \times C^2} + 2C$
      #Flops= $2 \times (C \times W \times H) \times C \times 3 \times 3 \times 2$ = $\mathbf{36 \times C^2 \times W \times H}$
      #activations= $\mathbf{2} \times C \times W \times H$

VGG-16	VGG-19
conv3-64	conv3-64
conv3-64	conv3-64
maxpool	maxpool
conv3-128	conv3-128
conv3-128	conv3-128
maxpool	maxpool
conv3-256	conv3-256
conv3-256	conv3-256
conv3-256	conv3-256
	conv3-256
maxpool	maxpool
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
	conv3-512
maxpool	maxpool
conv3-512	conv3-512
conv3-512	conv3-512
conv3-512	conv3-512
	conv3-512
maxpool	maxpool
FC-4096	FC-4096
FC-4096	FC-4096
FC-1000	FC-1000

Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-scale Image Recognition", ICLR 2015

VGG-16

	Layer	#Filters/ #Units	Filter Size	S	P	Activation Size
S1	Conv1	64	3x3	1	1	224x224
	Conv2	64	3x3	1	1	224x224
	Pool1	1	2x2	2	0	112x112
S2	Conv3	128	3x3	1	1	112x112
	Conv4	128	3x3	1	1	112x112
	Pool2	1	2x2	2	0	56x56
S3	Conv5	256	3x3	1	1	56x56
	Conv6	256	3x3	1	1	56x56
	Conv7	256	3x3	1	1	56x56
	Pool3	1	2x2	2	0	28x28
S4	Conv8	512	3x3	1	1	28x28
	Conv9	512	3x3	1	1	28x28
	Conv10	512	3x3	1	1	28x28
	Pool4	1	2x2	2	0	14x14
S5	Conv11	512	3x3	1	1	14x14
	Conv12	512	3x3	1	1	14x14
	Conv13	512	3x3	1	1	14x14
	Pool5	1	2x2	2	0	7x7
	Flatten	0	0	0	0	1x1
	Fc14	4096	-	-	-	1x1
	Fc15	4096	-	-	-	1x1
	fc16	1000	-	-	-	1x1

Totals: #params ≅ 138M, GFlops ≅ 39

$64 \times 3 \times 3 \times 3 \rightarrow$ **#params ≅ 1.8K, MFlops ≅ 173**

$64 \times 64 \times 3 \times 3 \rightarrow$ **#params ≅ 37K, GFlops ≅ 3.7**

$128 \times 64 \times 3 \times 3 \rightarrow$ **#params ≅ 74K, GFlops ≅ 1.85**

$128 \times 128 \times 3 \times 3 \rightarrow$ **#params ≅ 147.6K, GFlops ≅ 3.7**

$256 \times 128 \times 3 \times 3 \rightarrow$ **#params ≅ 295K, GFlops ≅ 1.85**

$2 \times 256 \times 256 \times 3 \times 3 \rightarrow$ **#params ≅ 1.19M, GFlops ≅ 7.4**

$512 \times 256 \times 3 \times 3 \rightarrow$ **#params ≅ 1.18M, GFlops ≅ 1.85**

$2 \times 512 \times 512 \times 3 \times 3 \rightarrow$ **#params ≅ 4.7M, GFlops ≅ 7.4**

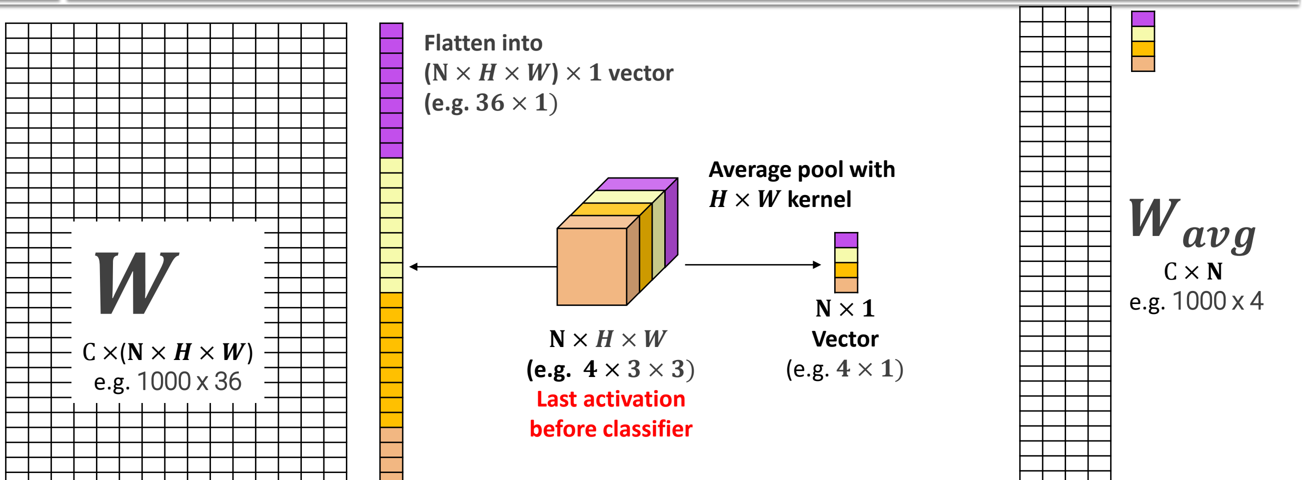
$3 \times 512 \times 512 \times 3 \times 3 \rightarrow$ **#params ≅ 7M, GFlops ≅ 11.1**

$4096 \times (\underbrace{7 \times 7 \times 512}_{25088}) \rightarrow$ **#params ≅ 102.7M, MFlops ≅ 205**

$4096 \times (4096) \rightarrow$ **#params ≅ 16.7M, MFlops ≅ 33**

$1000 \times (4096) \rightarrow$ **#params ≅ 4M, MFlops ≅ 8**

Global Average Pooling

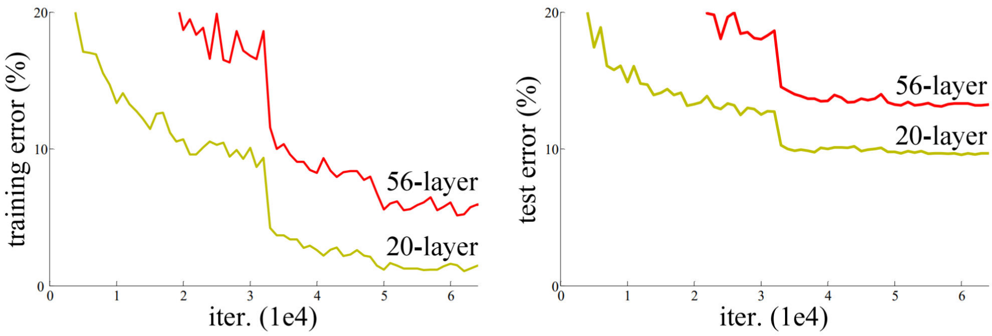


This reduces significantly the number of parameters at the interface between convolutional features and FC layers

Lin et al., "Network in network", ICLR 2014

Residual Networks - Motivation

The success of the VGG design would suggest to increase depth to improve performance, but..

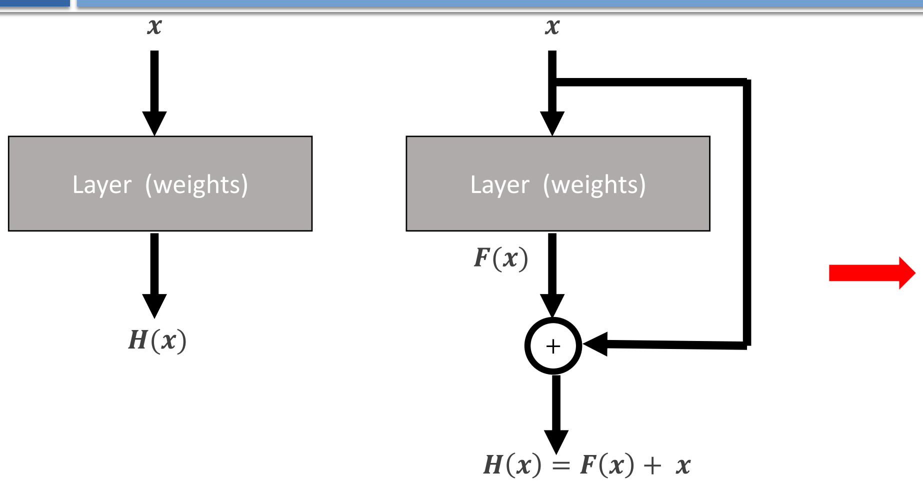


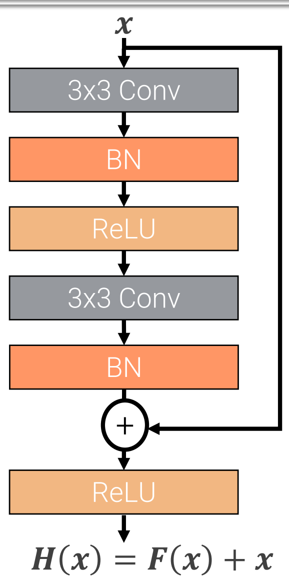
The problem is not (only) overfitting, the training error is larger for the deeper network !

Training very deep networks turns out to be inherently hard !

Kaiming He et al., "Deep Residual learning for image recognition", CVPR 2016

Residual Learning and Residual Blocks

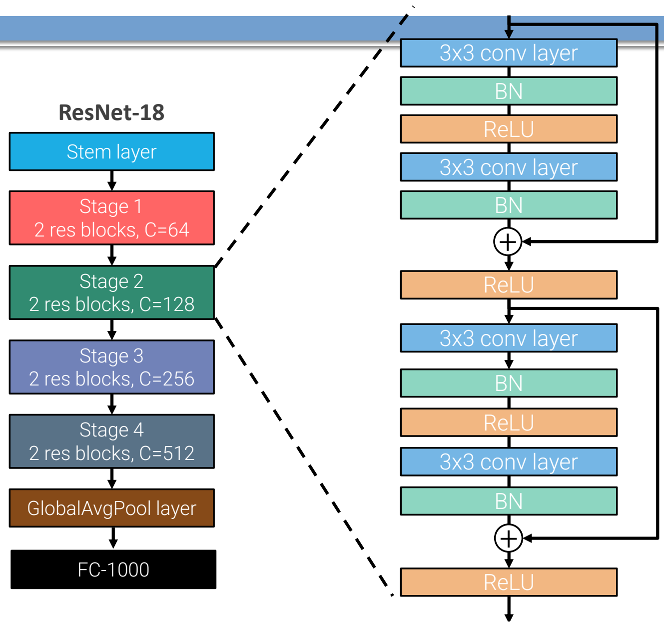




Conversely, residual learning via skip connections (*shortcuts*) enables to train successfully very deep networks (even as deep as ResNet-152)

Residual Networks – Architecture

- Inspired by VGG: a few simple design choices, repetition of stages.
- Stages are stacks of Residual Blocks (RB). Each RB includes two 3x3 conv layers.
- The first RB in *most* stages halves the spatial resolution (S=2) and doubles the number of channels.
- Initial Stem layer (S=2 conv + 2x2 max pool) to quickly down-sample the input image and Global Average Pooling to efficiency interface the final 1000-ways FC layer.
- Naming notation similar to VGG: ResNet-X denotes a ResNet architecture having X layers with learnable parameters.



ResNet18

Totals: #params ≅ 11.5M, GFlops ≅ 3.6

Layer	#Filters/ #Units	Filter Size	S	P	Activation Size
Conv1	64	7x7	2	3	112x112
Pool1	1	2x2	2	0	56x56
Conv2_1	64	3x3	1	1	56x56
Conv2_2	64	3x3	1	1	56x56
Conv2_3	64	3x3	1	1	56x56
Conv2_4	64	3x3	1	1	56x56
Conv3_1	128	3x3	2	1	28x28
Conv3_2	128	3x3	1	1	28x28
Conv3_3	128	3x3	1	1	28x28
Conv3_4	128	3x3	1	1	28x28
Conv4_1	256	3x3	2	1	14x14
Conv4_2	256	3x3	1	1	14x14
Conv4_3	256	3x3	1	1	14x14
Conv4_4	256	3x3	1	1	14x14
Conv5_1	512	3x3	2	1	7x7
Conv5_2	512	3x3	1	1	7x7
Conv5_3	512	3x3	1	1	7x7
Conv5_4	512	3x3	1	1	7x7
AvgPool	1	7x7	1	0	1x1
fc1	1000	-	-	-	1x1

S1: Conv2_1 – Conv2_4; S2: Conv3_1 – Conv3_4; S3: Conv4_1 – Conv4_4; S4: Conv5_1 – Conv5_4

$64 \times 3 \times 7 \times 7 \rightarrow$ #params ≅ 9.5K, MFlops ≅ 236

$4 \times 64 \times 64 \times 3 \times 3 \rightarrow$ #params ≅ 148 K, GFlops ≅ 0.9

$128 \times 64 \times 3 \times 3 \rightarrow$ #params ≅ 74 K, MFlops ≅ 115

$3 \times 128 \times 128 \times 3 \times 3 \rightarrow$ #params ≅ 443 K, GFlops ≅ 0.7

$256 \times 128 \times 3 \times 3 \rightarrow$ #params ≅ 295 K, MFlops ≅ 115

$3 \times 256 \times 256 \times 3 \times 3 \rightarrow$ #params ≅ 1.8 M, GFlops ≅ 0.7

$512 \times 256 \times 3 \times 3 \rightarrow$ #params ≅ 1.18 M, MFlops ≅ 115

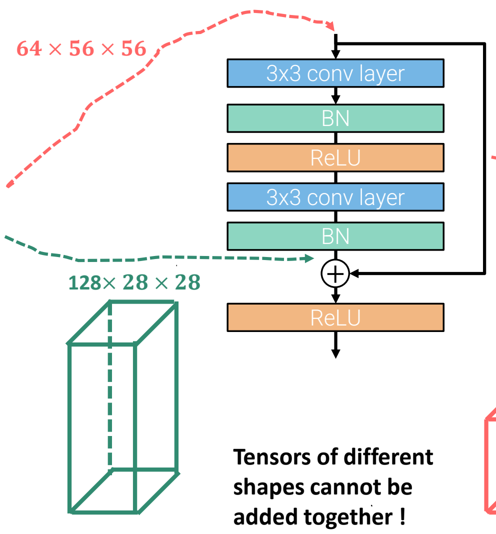
$3 \times 512 \times 512 \times 3 \times 3 \rightarrow$ #params ≅ 7 M, GFlops ≅ 0.7

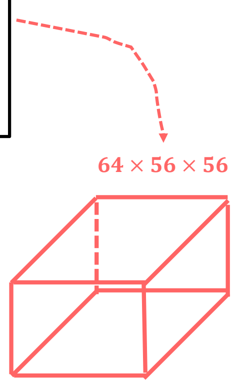
$1000 \times 512 \rightarrow$ #params ≅ 0.5 M, MFlops ≅ 1

First RB in a stage & shape of skip connections

Layer	#Filters/ #Units	Filter Size	S	P	Activation Size
Conv1	64	7x7	2	3	112x112
Pool1	1	2x2	2	0	56x56
Conv2_1	64	3x3	1	1	56x56
Conv2_2	64	3x3	1	1	56x56
Conv2_3	64	3x3	1	1	56x56
Conv2_4	64	3x3	1	1	56x56
Conv3_1	128	3x3	2	1	28x28
Conv3_2	128	3x3	1	1	28x28
Conv3_3	128	3x3	1	1	28x28
Conv3_4	128	3x3	1	1	28x28
Conv4_1	256	3x3	2	1	14x14
Conv4_2	256	3x3	1	1	14x14
Conv4_3	256	3x3	1	1	14x14
Conv4_4	256	3x3	1	1	14x14
Conv5_1	512	3x3	2	1	7x7
Conv5_2	512	3x3	1	1	7x7
Conv5_3	512	3x3	1	1	7x7
Conv5_4	512	3x3	1	1	7x7
AvgPool	1	7x7	1	0	1x1
fc1	1000	-	-	-	1x1

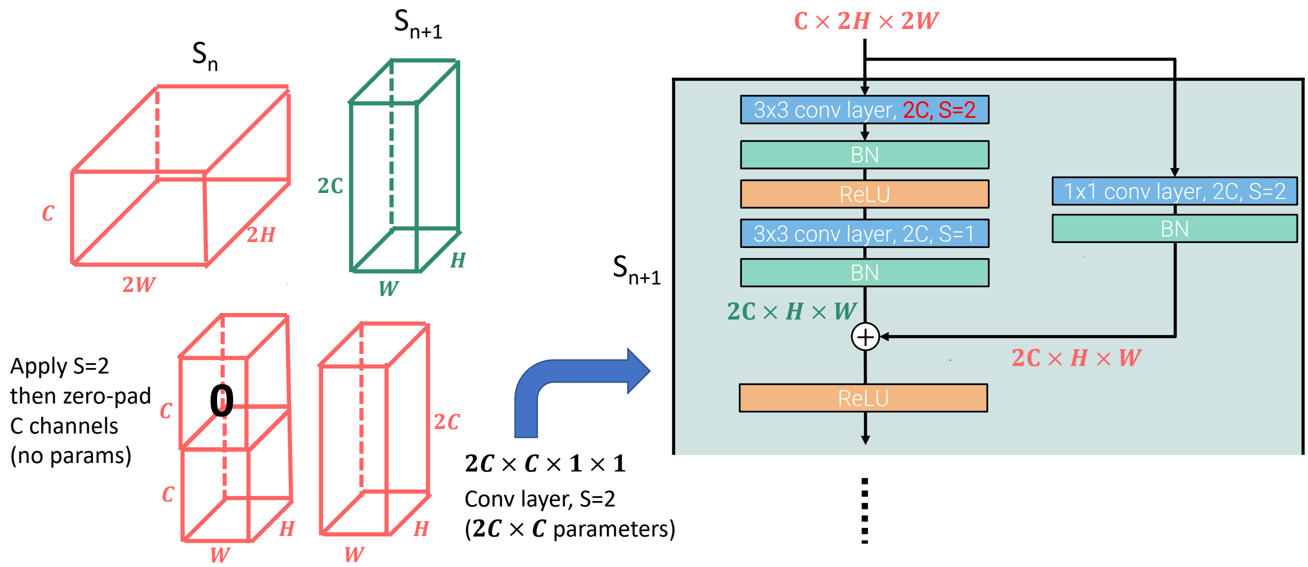
RB: Conv2_3 – Conv2_4; RB: Conv3_1 – Conv3_2



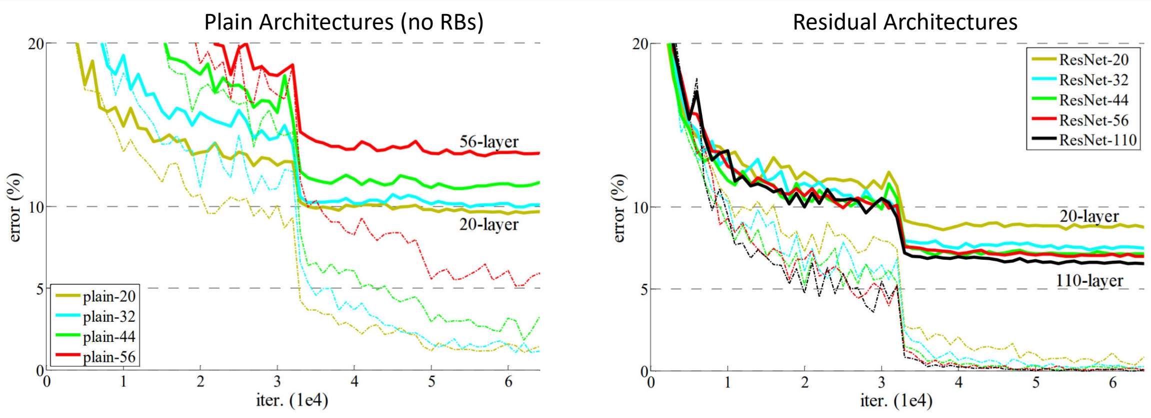


Tensors of different shapes cannot be added together !

Modified first RB in a stage



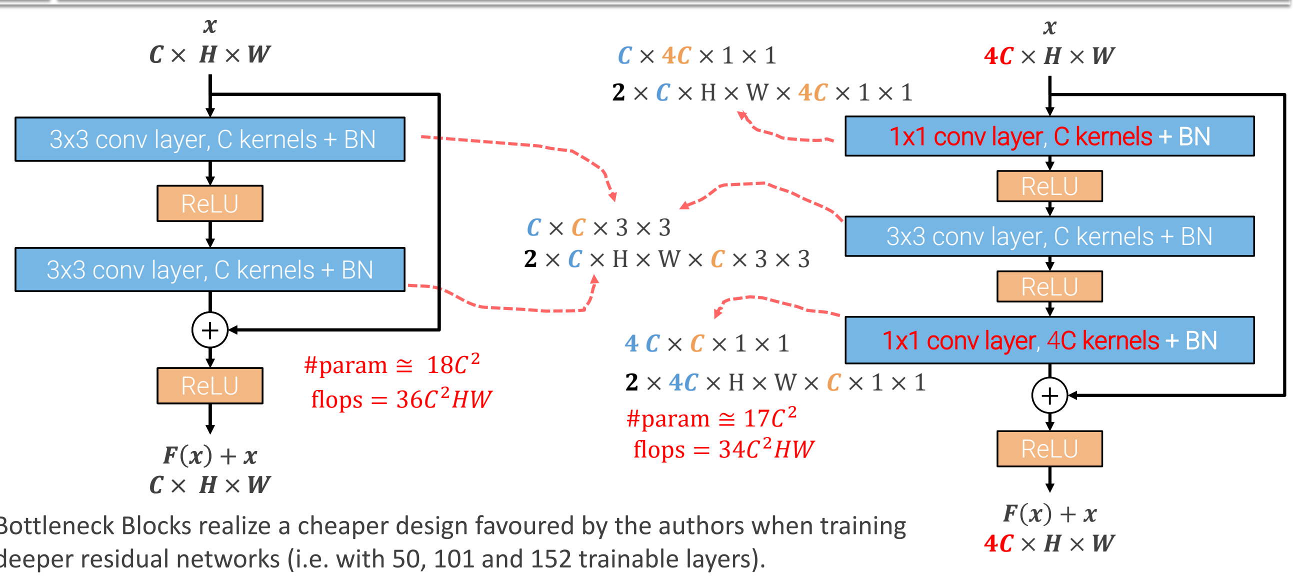
Residual vs Plain Architectures



The shallower the better....... **....the deeper the better !**

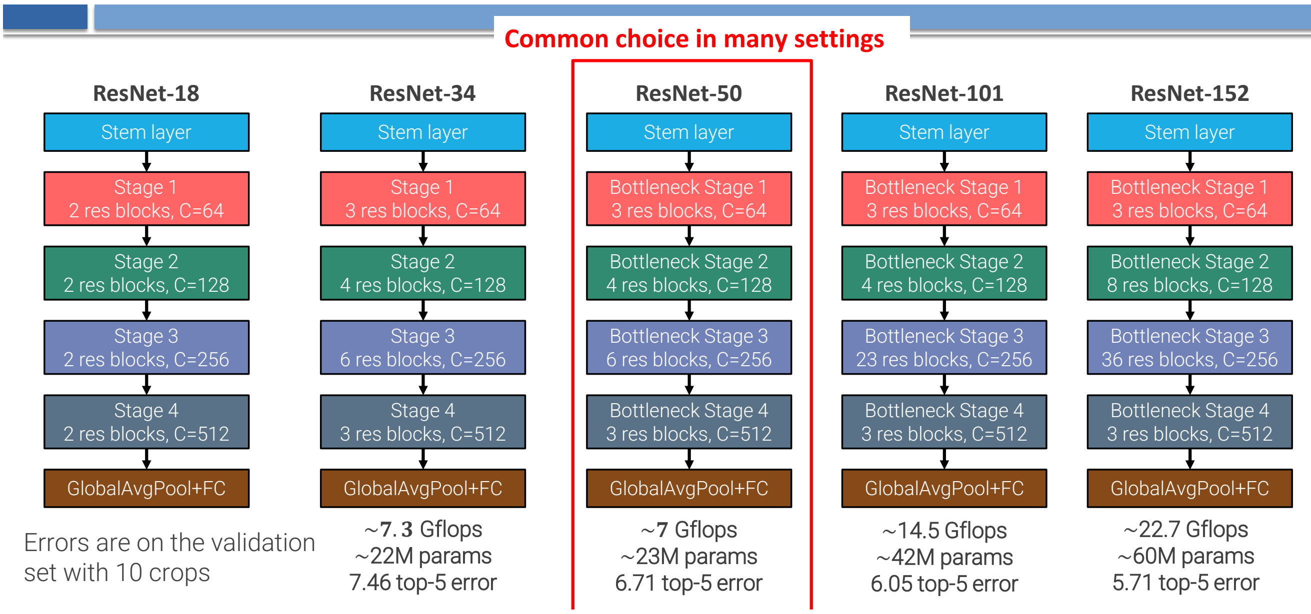
Residual Blocks allow for training deep CNNs. When properly trained, deep architectures outperform shallower ones. In 2015, ResNets won all the main computer vision competitions by large margins. In ILSVRC, an ensemble of ResNets (including two models with 152 layers) brought the Top-5 error from 6.7 to 3.6.

Bottleneck Residual Blocks for deeper ResNets



Bottleneck Blocks realize a cheaper design favoured by the authors when training deeper residual networks (i.e. with 50, 101 and 152 trainable layers).

Main ResNet Architectures



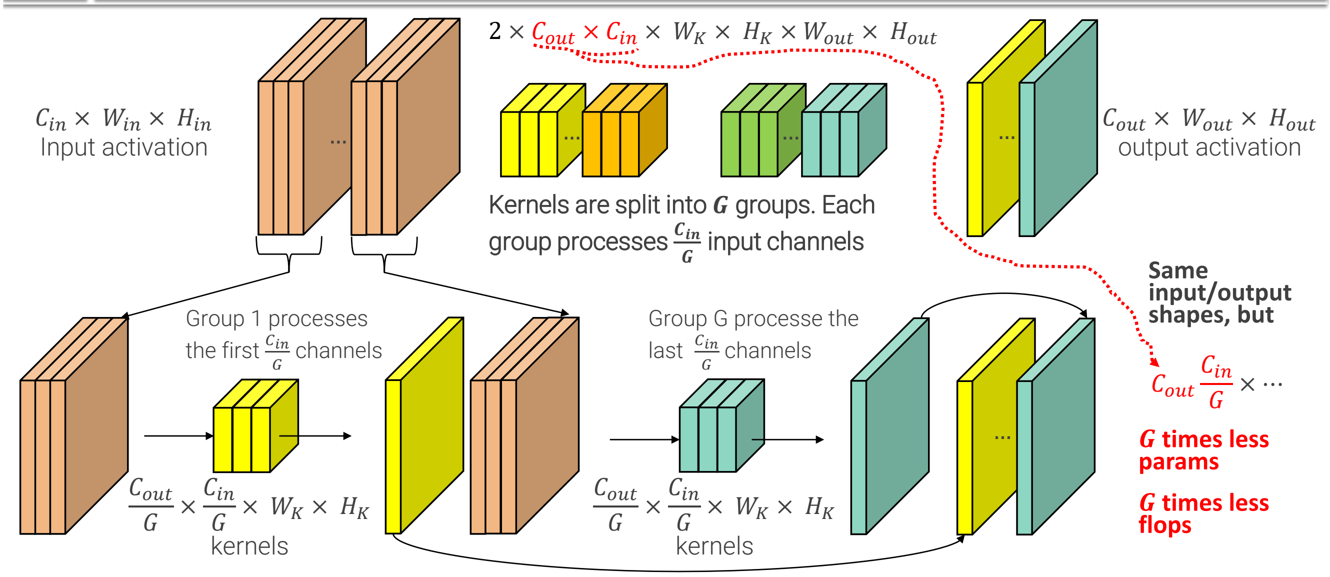
ResNets (from the paper)

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

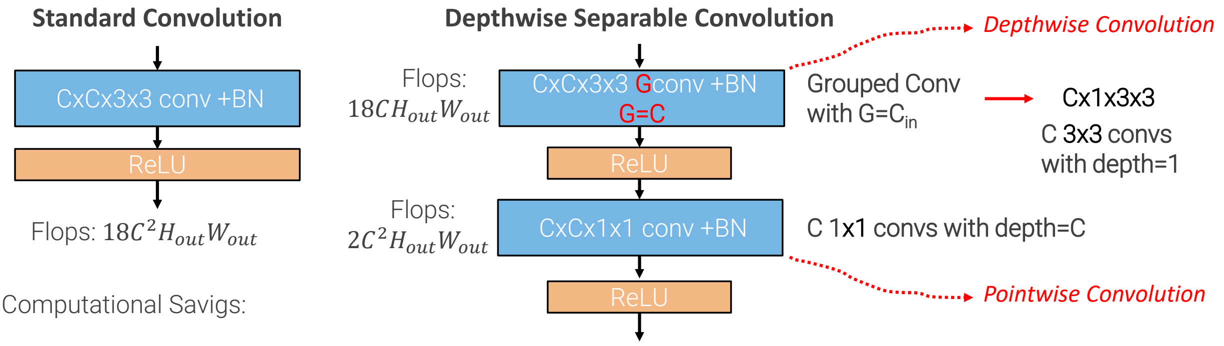
Down-sampling is performed by conv3_1, conv4_1 and con5_1.

MACs

Grouped Convolutions



Depthwise Separable Convolutions



Computational Savigs:

$$\frac{18C^2}{18C+2C^2} = \frac{18C}{18+2C} \cong [8, 8.85]$$

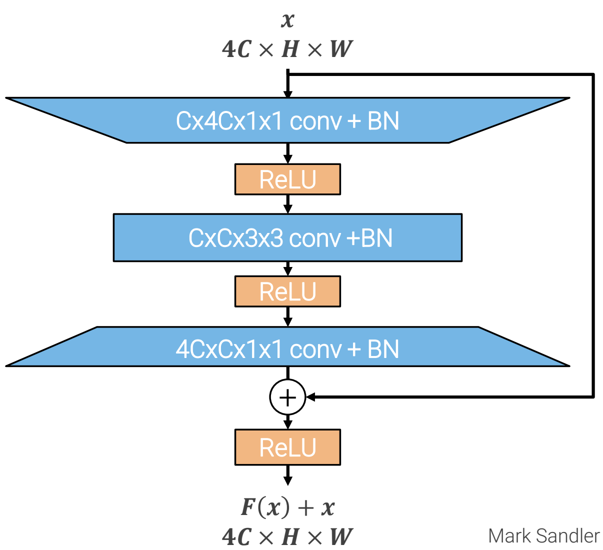
for C in [64, 512]

Standard convolutions used in CNNs *filter* features spatially while *combining* them to produce new representations.

with **Depthwise Separable Convolutions** the two steps are split and carried out sequentially to gain substantial computational savings.

Howard et al, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv 2017
F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions", CVPR 2017

A closer look at Bottleneck Residual Blocks



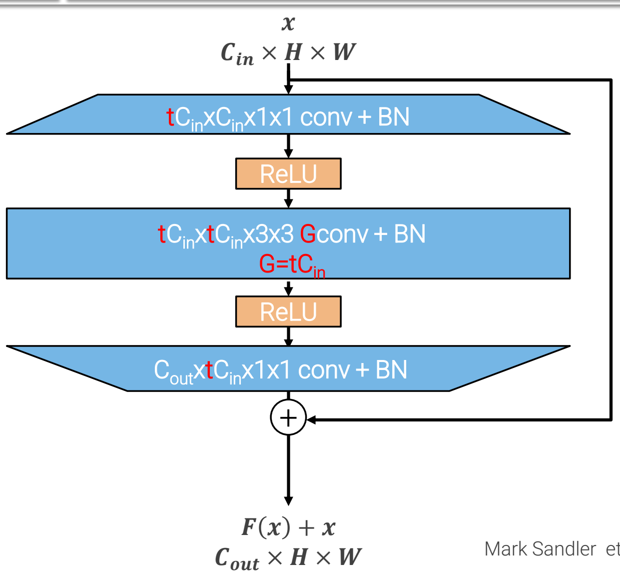
The bottleneck residual block was introduced to scale up the depth of ResNets by increasing significantly the number of blocks per stage without growing too much the computation and number of parameters.

Purposely, it uses a pair of 1x1 convs, where **the first compresses** the number of channel and **the second expands** them.

Hence, the 3x3 convolution that processes spatial information, i.e. the core representation learning function performed by the block, is carried out in a **compressed** domain. This may result in **information loss**.

Mark Sandler et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks", CVPR 2018

Inverted Residual Block



To avoid th potential information loss of standard bottleneck blocks, **MobileNet-v2** proposes to use **inverted residual blocks**.

In such blocks, the first 1x1 conv expands the channels according to a chosen **expansion factor t**, while the second compresses them back (to the same or a different number of channels, i.e. C_{out} may be different than C_{in}) .

To limit the increase in computation, the inner 3x3 convolution is realized as a **depthwise convolution**.

The last difference wrt standard bottleneck blocks is the **removal of non-linearities between residual blocks**: this is motivated by a theoretical study and experimentally verified.

Compared to the standard bottleneck block, the inverted design is **considerably more memory efficient** at inference time.

Mark Sandler et al., "MobileNetV2: Inverted Residuals and Linear Bottlenecks", CVPR 2018

MobileNet-v2

- **MobileNet-v2** is a deep architecture specifically tailored for **mobile and resource constrained platforms**. It is based on a stack of **inverted residual blocks** and features 54 layers with parameters. It deploys only 1x1 and 3x3 convs.
- The **number of channels** grows slowly compared to previous architectures to keep the complexity low. A low number of channels does not require an heavy size reduction in the **stem layer** (**s=2**). Yet, the representation must be expanded by a **pointwise convolution** before feeding it to final **k-way classifier** via **global average pooling**.
- As for the stack of **inverted residual blocks**, each line in the table can be seen as a *stage*. When a stage down-samples the activation, it does so by applying **s=2** in the inner 3x3 conv of the first inverted residual block.
- Whenever spatial dimensions or number of channels do not match between input and output of a block, there are no skip connections.

Data taken fom the paper

Top-1	#params	MACs	CPU
72.0	3.4M	300M	75 ms

Google Pixel 1

C_{out} *stride*

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

e.g. k=1000 (ILSVRC)

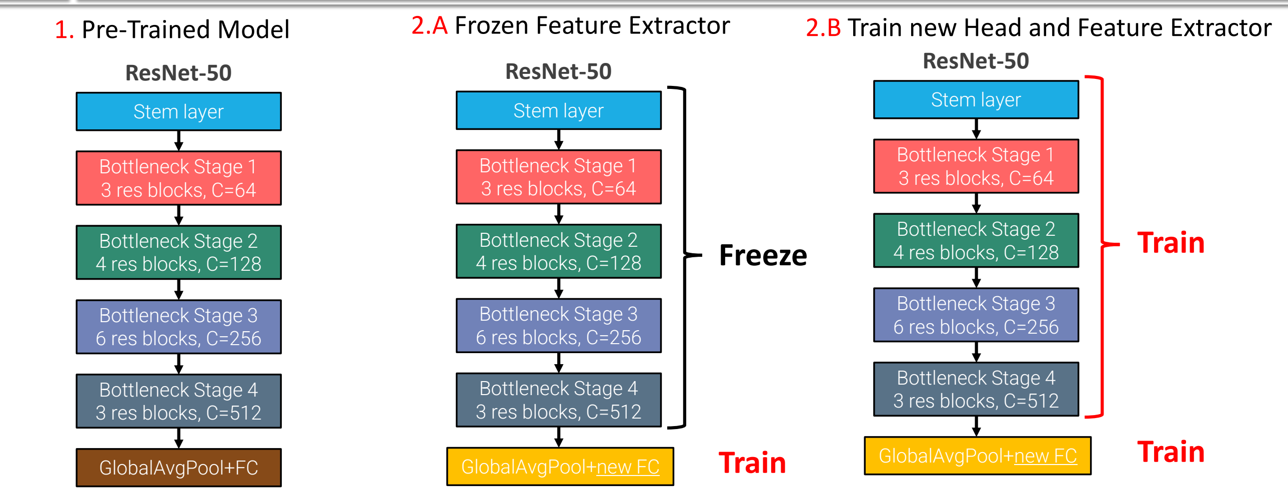
Transfer Learning (1)

- To prevent ***overfitting***, a large/deep (i.e. ***high capacity***) neural network requires a large number of samples to effectively train its many. But annotated (aka ***supervised***) training data are expensive…what if in our scenario we only have a small training set?
- We may deploy a two-steps approach referred to as **Transfer Learning:**

1. **Pre-train** the network on a large dataset (e.g. ImageNet)
2. **Fine-tune** the pre-trained network on the smaller, task-specific dataset.

- Typically, in the first step one relies on **standard architectures** (e.g. ResNet-50) and downloads the pretrained model from a public repository.

Transfer Learning (2)



2.B: *warm-up* with a frozen feature extractor (like in 2.A) then fine-tune the whole model with a very small learning rate. The initial layers may still be kept frozen as they typically learn general, low-level features.