Deep Learning

4/ Convolutional Neural Network - advanced

Francesca Galassi, MCF, Esir

francesca.galassi@irisa.fr
Lab Empenn Irisa-Inria

ILSVRC : ImageNet Large Scale Visual Recognition Challenge

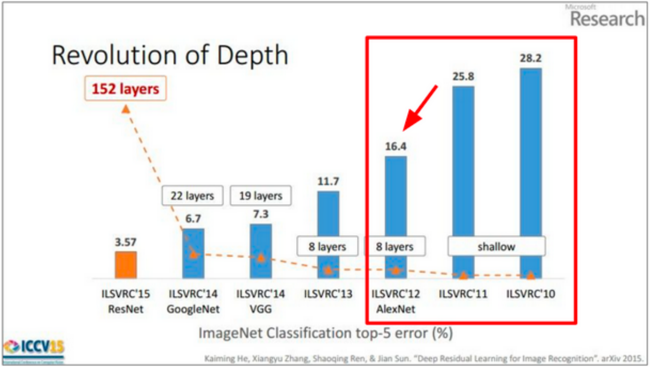
➟ ILSVRC evaluates algorithms for large-scale image classification and object detection since 2010.

- Image classification (1000 categories) and object detection (200 categories), video object detection (30 categories)

ImageNet dataset https://www.image-net.org/

ILSVRC : ImageNet Large Scale Visual Recognition Challenge

- ILSVRC evaluates algorithms for large-scale image classification and object detection since 2010.
  - Image classification (1000 categories) and object detection (200 categories), video object detection (30 categories)
- ILSVRC 2015 : He et al. introduced deep residual learning to train deeper networks effectively.

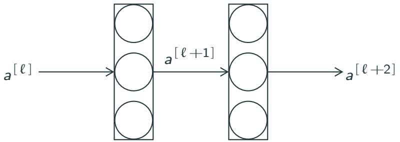


Slide from He K. presentation at ICCV15

Residual Block

Residual Block

- Residual blocks : enabling training of (very) very deep NNs, e.g., over 100 layers
- Plain block :



- The information flows from $a^{[\ell]}$ to $a^{[\ell+2]}$ through all of these steps :

$$a^{[\ell]} \longrightarrow \text{Linear} \longrightarrow \text{ReLU} \xrightarrow{a^{[\ell+1]}} \text{Linear} \longrightarrow \text{ReLU} \longrightarrow a^{[\ell+2]}$$

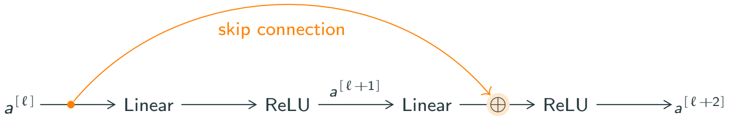
- Corresponding equations :

layer $[\ell+1]$: $\quad z^{[\ell+1]} = W^{[\ell+1]} a^{[\ell]} + b^{[\ell+1]} \quad ; \quad a^{[\ell+1]} = g\left(z^{[\ell+1]}\right)$

layer $[\ell+2]$: $\quad z^{[\ell+2]} = W^{[\ell+2]} a^{[\ell+1]} + b^{[\ell+2]} \quad ; \quad a^{[\ell+2]} = g\left(z^{[\ell+2]}\right)$

Residual Block

➟ Residual block : $a^{[\ell]}$ is injected deeper into the NN via a short cut

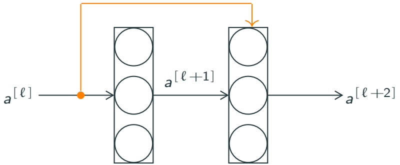


layer $[\ell + 2]$: $z^{[\ell+2]} = W^{[\ell+2]} a^{[\ell+1]} + b^{[\ell+2]}$; $a^{[\ell+2]} = g\left(z^{[\ell+2]}\right)$

➟ $a^{[\ell]}$ is added before applying non-linearity :

$$a^{[\ell+2]} = g\left(z^{[\ell+2]}\right) \longrightarrow \boxed{a^{[\ell+2]} = g\left(z^{[\ell+2]} + a^{[\ell]}\right)}$$

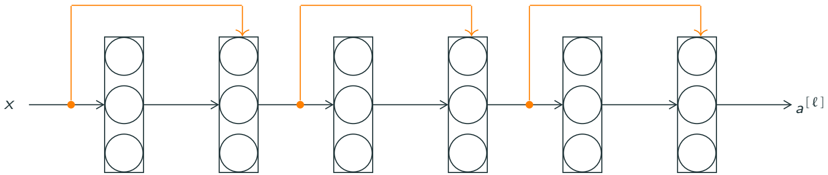
➟ Residual block :



Residual Network

➟ *Plain* very deep network : optimizer has difficulty due to vanishing/exploding gradients

➟ Residual Network [1] : stacking residual blocks allows training of very deep networks

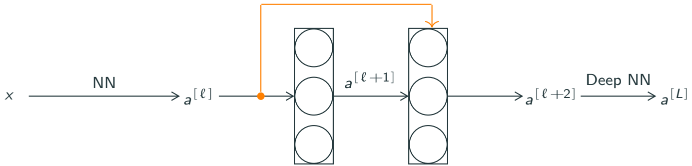


➟ ResNets training error keeps decreasing, where *plain* very deep NN tends to increase

1. [He et al., 2015 : Deep Residual networks for image recognition]

Residual Network

☞ Why do Residual Network work ?



➟ $a^{[\ell+2]} = g\left(z^{[\ell+2]}\right) \quad \longrightarrow \quad a^{[\ell+2]} = g\left(z^{[\ell+2]} + a^{[\ell]}\right)$

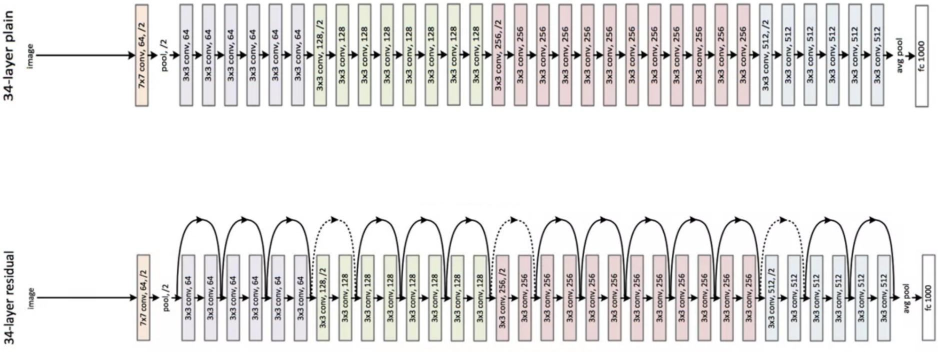
$$\begin{aligned} a^{[\ell+2]} &= g\left(z^{[\ell+2]} + a^{[\ell]}\right) \\ &= g\left(W^{[\ell+2]} a^{[\ell+1]} + b^{[\ell+2]} + a^{[\ell]}\right) \end{aligned}$$

➟ $\underset{\text{weight decay}}{\text{If}} \; \mathrm{W}^{[\ell+2]} = 0,\ b^{[\ell+2]} = 0 \quad \Longrightarrow \quad a^{[\ell+2]} = g\left(a^{[\ell]}\right) \underset{\text{because of ReLU}}{\Longrightarrow} a^{[\ell+2]} = a^{[\ell]}$

Learning the identity function

➟ Remark : in ConvNets *same* convolution often used - or add an extra matrix to resize $a^{[\ell]}$

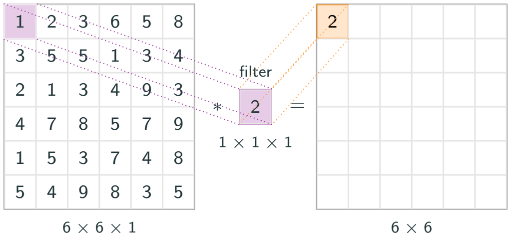
ResNets [2]



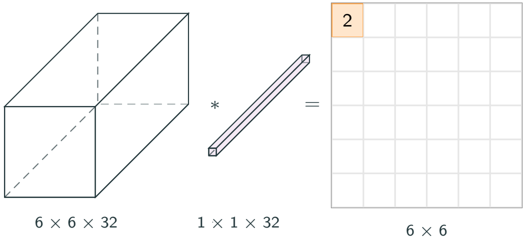
2. [He et al., 2015 : Deep Residual networks for image recognition]

1x1 Convolution

1x1 Convolution [3]



➟ Simple element-wise multiplication if $n_c = 1$.

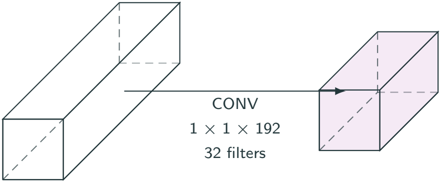


➟ Reducing the number of channels (while introducing non-linearity).

3. [Lin et al. 2013 Network in network]

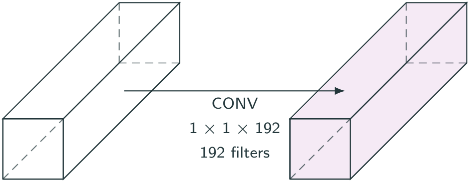
Why Use 1x1 Convolution ?

➟ Decreasing (increasing) the number of channels n_c in the input volume :



Input : $28 \times 28 \times \underbrace{192}_{n_c}$ Output : $28 \times 28 \times \underbrace{32}_{n_c\prime}$

➟ Adding non-linearity, i.e., allowing for more complex function :

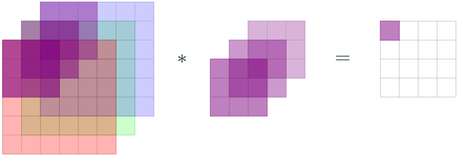


$28 \times 28 \times \underbrace{192}_{n_c}$ $28 \times 28 \times \underbrace{192}_{n_c}$

Depthwise Separable Conv

MobileNets

- **Low computational cost** at deployment
- Ideal for **mobile** and embedded vision applications
- Key idea : depthwise-separable convolutions [4]
- Recall : Standard convolution



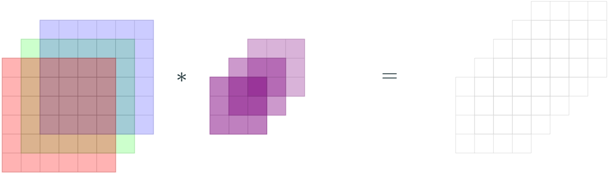
$\underset{n}{6} \times \underset{n}{6} \times \underset{n_c}{3}$ $\quad$ $\underset{f}{3} \times \underset{f}{3} \times \underset{n_c}{3}$ $\quad$ $\underset{n_{out}}{4} \times \underset{n_{out}}{4}$

- Computations :
  (i.) 27 element-wise multiplications ;
  (ii.) Add them up ;
  (iii.) Shift the filter and repeat.

4. [Howard et al. 2017, MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications]

Standard Convolution vs. Depthwise Separable Convolution

➟ Standard convolution operation :



$$\underset{n}{6} \times \underset{n}{6} \times \underset{n_c}{3} \quad * \quad \underset{f}{3} \times \underset{f}{3} \times \underset{n_c}{3} \quad = \quad \underset{n_{out}}{4} \times \underset{n_{out}}{4} \times \underset{n_c\,'}{5}$$

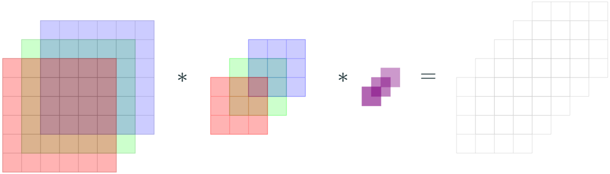
5 filters

➟ Computational cost = # filter params × # filter positions × # of filters

$$= 3 \times 3 \times 3 \times 4 \times 4 \times 5 = 2160 \text{ computations}$$

Depthwise Separable Convolution

➟ Depthwise separable convolution :

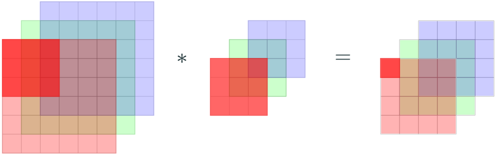


Depthwise conv Pointwise conv

➟ Two steps : **Depthwise** Convolution followed by **Pointwise** Convolution

Depthwise Separable Convolution

1. **Depthwise** convolution :



$$\underset{n}{6} \times \underset{n}{6} \times \underset{n_c}{3} \quad * \quad \underset{f}{3} \times \underset{f}{3} \quad = \quad \underset{n_{out}}{4} \times \underset{n_{out}}{4} \times \underset{n_c}{3}$$

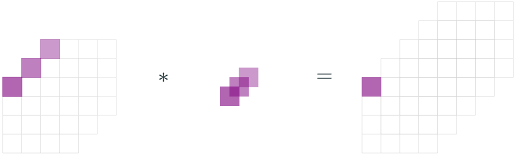
➟ Each filter is applied per channel, resulting in 9 multiplications per filter at 16 locations.

➟ The number of output channels is equal to the number of input channels.

$$\text{Computational cost} = \#\text{filter params} \times \#\text{ filter positions} \times \#\text{ of filters}$$
$$= 3 \times 3 \times 4 \times 4 \times 3 = 432$$

Depthwise Separable Convolution

2. **Pointwise** (1x1) convolution on previous output :



$\underset{n_{out}}{4} \times \underset{n_{out}}{4} \times \underset{n_c}{3}$ $\quad * \quad$ $\underset{f}{1} \times \underset{f}{1} \times \underset{n_c}{3}$ $\quad = \quad$ $\underset{n_{out}}{4} \times \underset{n_{out}}{4} \times \underset{n_{c'}}{5}$

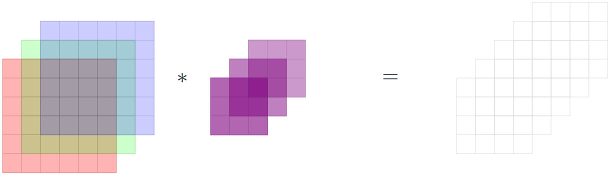
5 filters

$$\text{Computational cost} = \# \text{ filter params} \times \# \text{ filter positions} \times \# \text{ of filters}$$

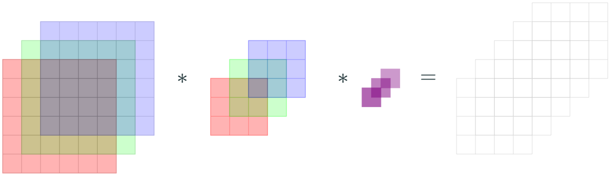
$$= 1 \times 1 \times 3 \times 4 \times 4 \times 5 = 240 \text{ multiplications}$$

Depthwise Separable Convolution

➟ Cost of the standard convolution : 2160 multiplications



➟ Cost of the Depthwise Separable convolution : $432 + 240 = 672$ multiplications



Depthwise Pointwise

Cost Summary

- Cost of standard convolution : 2160
- Cost of depthwise separable convolution : 672

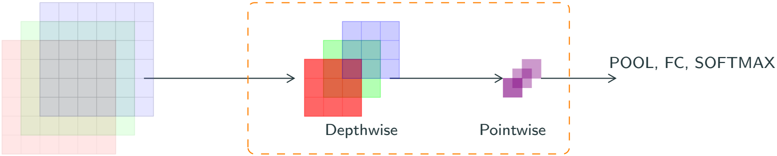
  factor of $\frac{672}{2160} = 0.31 \sim 3$ times more efficient
- In a general case[5] :

$$\frac{\text{depthwise separable convolution}}{\text{standard convolution}} = \frac{1}{n_{c'}} + \frac{1}{f^2}$$
$$\underset{example}{=} \frac{1}{5} + \frac{1}{3^2}$$
$$\underset{commonly}{=} \frac{1}{512} + \frac{1}{3^2} \sim 10 \text{ times cheaper}$$

5. [Howard et al. 2017, MobileNets : Efficient Convolutional Neural Networks for Mobile Vision Applications']

MobileNet

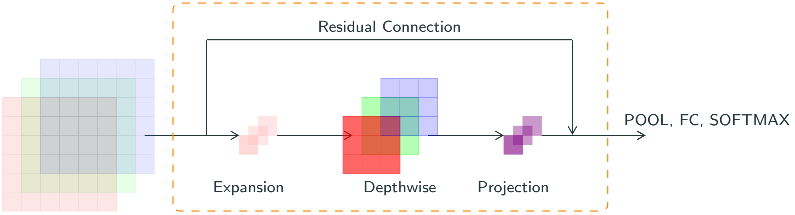
➟ MobileNet v1 : 13 depthwise separable convolutional layers



➟ Good performance, lower computational cost

MobileNets

➟ MobileNet v2 [6] : 17 blocks



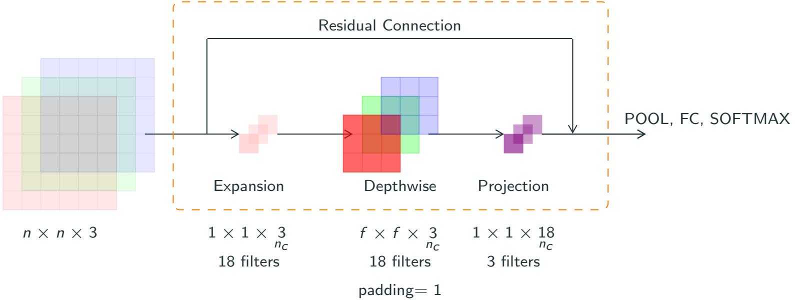
➟ Addition of an expansion layer : lower computational cost

➟ Addition of a residual (or skip) connection : improved backpropagation of gradients

6. [Sandler et al. 2019, MobileNetV2 : Inverted Residuals and Linear Bottlenecks]

MobileNets

➟ MobileNet v2 : 17 bottleneck blocks



➟ Given an input $n \times n \times 3$, the outputs of the operations in the block are :

(i.) Expansion output : $n \times n \times 18$ $\longrightarrow$ expansion by a factor of 6

(ii.) Depthwise output : $n \times n \times 18$

(iii.) Pointwise output : $n \times n \times 3$ $\longrightarrow$ projection

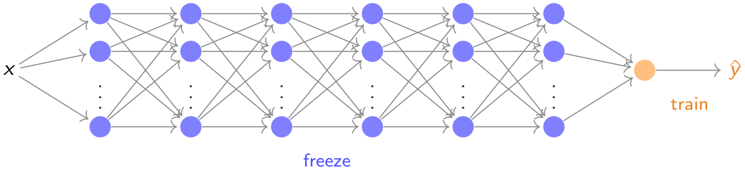
Transfer Learning

Transfer Learning : Fine-tuning

- ➟ Download open-source pre-trained weights on a large dataset
- ➟ Transfer Learning : fine-tune the model or adjust the architecture to your data

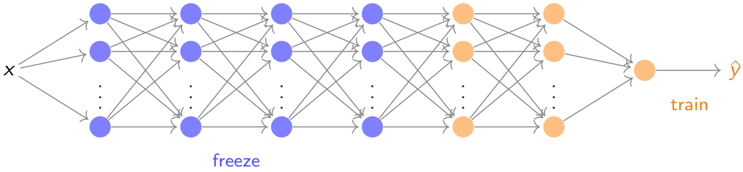
☞ Example : classification problem with small training set

- ➟ Download a network for classification, e.g., on ImageNet
- ➟ Replace the softmax layer and train only parameters of this layer



Fine-tuning

- Example : classification problem with a larger training set.
  - ➟ Train the later layers (or replace with your own layers).



- Example : classification problem with a large training set.
  - ➟ Train all network, i.e., use pre-trained weights as initialization.