Architectures de CNN

Classification d’images

Sommaire

Introduction

- In deep learning, tranfer learning means reusing the weights in one or more layers from a pre-trained network model and either keeping the weights fixed, fine tuning them, or adapting the weights entirely when training the mode.

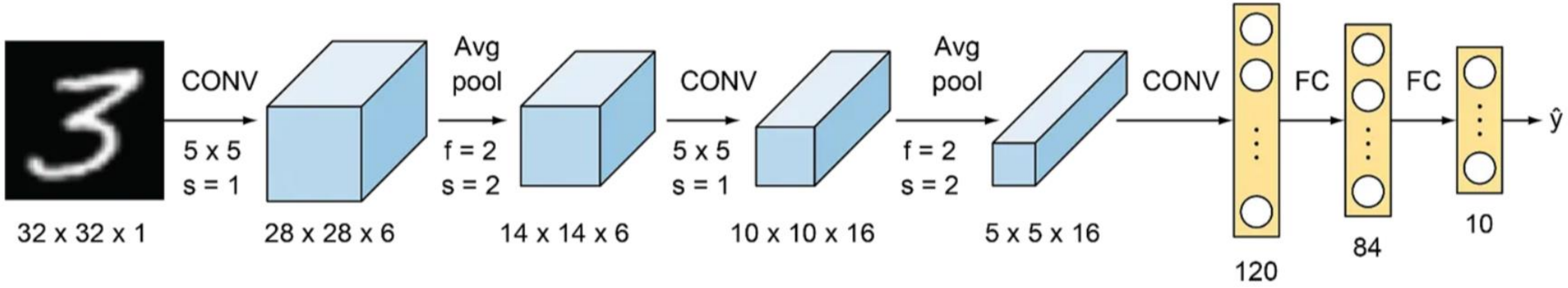
- For some architectures, such as CNN, it is common to keep the earlier layers (those closest to the input layer) frozen because they capture lower-level features, while later layers often discern high-level features that can be more related to the task that the model is trained on.

- Models that are pre-trained on large and general corpora are usually fine-tuned by reusing the model's parameters as a starting point and adding a task-specific layer trained from scratch.

- Fine-tuning the full model is common as well and often yields better results, but it is more computationally expensive.

1. Architectures simples
LeNet-5 (1998)

- The LeNet-5 architecture is perhaps the most widely known CNN architecture. It was created by Yann LeCun in 1998 and has been widely used for handwritten digit recognition (MNIST).



$\simeq$ 60k paramètres

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

LeNet-5 (1998)

- Classic : a couple of Conv/Pooling are followed by fully connected layers.
- Tanh is used was an activation function except for the output layer.
- Max pooling was not famous at that time, this is why average pooling is used for LeNet-5.

Architecture

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1×1	5×5	1	tanh
S4	Avg pooling	16	5×5	2×2	2	tanh
C3	Convolution	16	10×10	5×5	1	tanh
S2	Avg pooling	6	14×14	2×2	2	tanh
C1	Convolution	6	28×28	5×5	1	tanh
In	Input	1	32×32	–	–	–

LeNet - 5 : Architecture

- The first convolution layer with 6 filters (5x5) and a stride of one

  Input : 32 x 32 images | Output : 28x28x6
- Average pooling with a filter size 2x2 and astride of 2

  Input 28x28x6 | Output : 14x14x6
- A second convolution layer with 16 filters (5x5) and a stride of one

  Input 14x14x6 | Output : 10x10x16
- Average pooling with a filter size 2x2 and astride of 2

  Input 10x10x16 | Output : 5x5x16
- A fully connected layer with 120 units

  A fully connected layer with 84 units
- Output layer : Softmax fully connected with 10 units

ImageNet Large Scale Visual Recognition (INLSVR)

ImageNet challenge

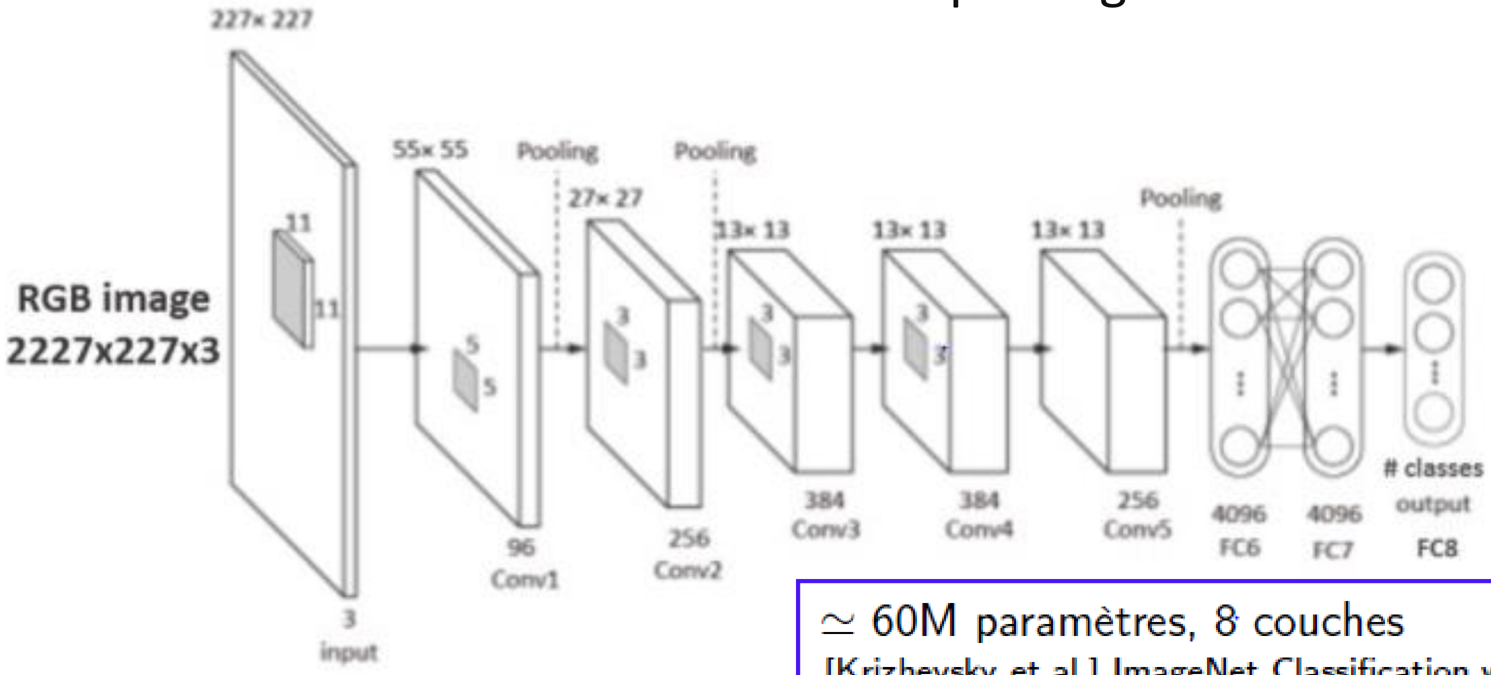
- A good measure of the progress on CNN architectures is the error rate in competitions such as the ILSVRC ImageNet challenge.
- The Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) is a subset of the large hand-labeled ImageNet dataset (10,000,000 labeled images depicting 10,000+ object categories). The images are large (256 pixels high). The training data is a subset of ImageNet containing the 1000 categories and 1.2 million images.

- In this competition the top-five error rate for image classification fell from over 26% to less than 2.3% in just six years.

- The top-five error rate is the number of test images for which the system's top five predictions did not include the correct answer.

AlexNet (2012)

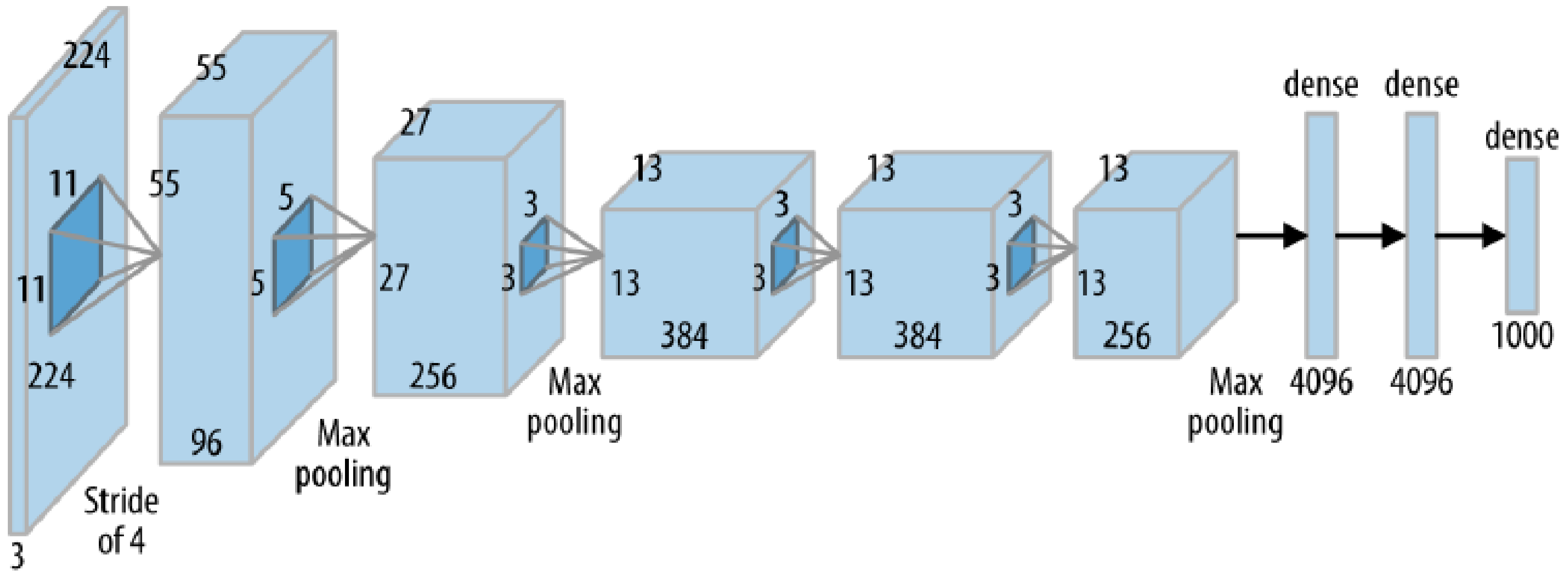
- Proposée en 2012 par Alex Krizhevky et al.
- AlexNet competed in the ImageNet Challenge on September 30, 2012. The network achieved a top-5 error of 15.3%
- L'architecture AlexNet inclue 5 couches de convolution, 3 couches de pooling et 3 couches FC.
- Elle ressemble à LeNet-5 en étant plus profonde.
- Elle a été la première à empiler des couches de convolution directement les unes au dessus des autres sans intercaler des couches de pooling.



$\simeq$ 60M paramètres, 8 couches

[Krizhevsky et al.] ImageNet Classification with Deep Convolutional Neural Networks.

AlexNet (2012)



Observations :

- Diminution progressive de la taille des filtres ($11 \to 5 \to 3$)
- Diminution progressive de la taille de l'image ($224 \to 55 \to 27 \to 13$)
- Augmentation progressive du nombre de filtres ($96 \to 256 \to 384$)
- *Stride* puis *Max Pooling*

AlexNet (2012)

Architecture

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6×6	3×3	2	valid	–
C7	Convolution	256	13×13	3×3	1	same	ReLU
C6	Convolution	384	13×13	3×3	1	same	ReLU
C5	Convolution	384	13×13	3×3	1	same	ReLU
S4	Max pooling	256	13×13	3×3	2	valid	–
C3	Convolution	256	27×27	5×5	1	same	ReLU
S2	Max pooling	96	27×27	3×3	2	valid	–
C1	Convolution	96	55×55	11×11	4	valid	ReLU
In	Input	3 (RGB)	227×227	–	–	–	–

VGG16 (2014)

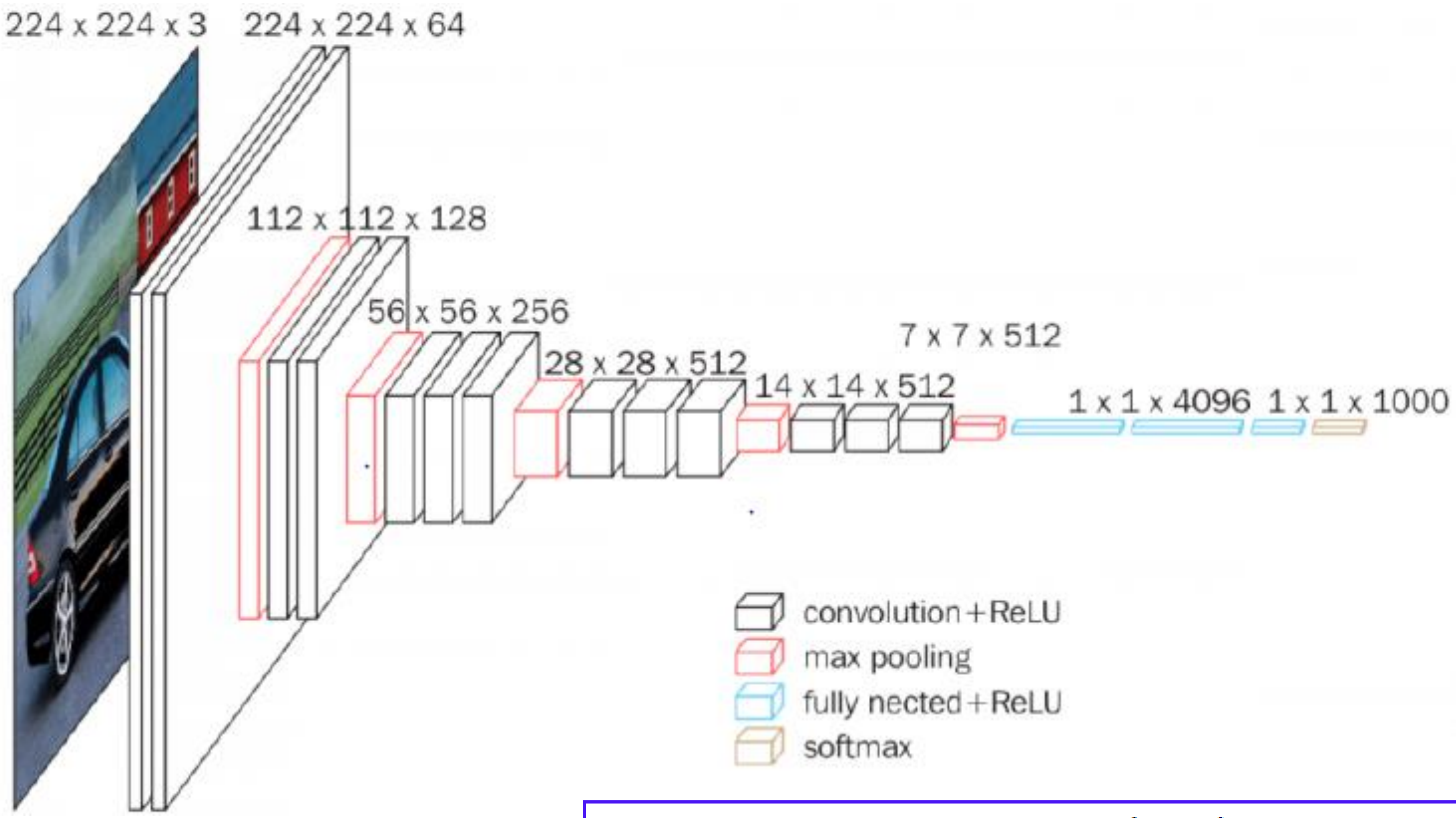
- It was submitted to ILSVR Challenge 2014 and the model achieves 92.7% top-5 test accuracy in Imagenet.

- The main concept is stacking of convolutional layers to create a DNN.

- VGG shows that the depth of the network has an important role. DNN give better results.

- One drawback of VGGNet is that this network is that it contains around 160M parameters, most of them are consumed in the fully connected layers.

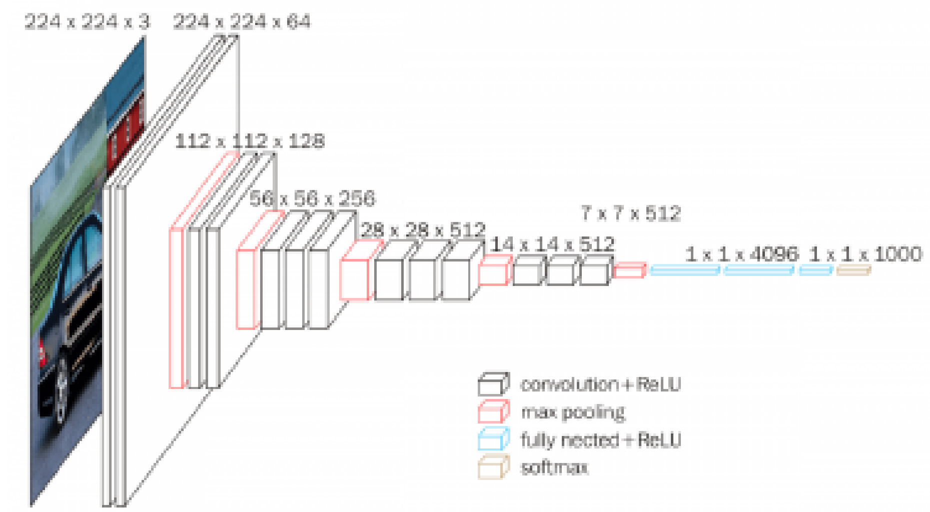
VGG16 (2014)



$\simeq$ 138M paramètres, 16 couches dans sa version standard.

[Simonyan et Zisserman] Very Deep Convolutional Networks for Large-Scale Image Recognition

VGG16 (2014)



Objectif : étudier l'impact de la profondeur sur les performances du réseau.
→ Pour cela, les auteurs ont rendu l'architecture du réseau très régulière :

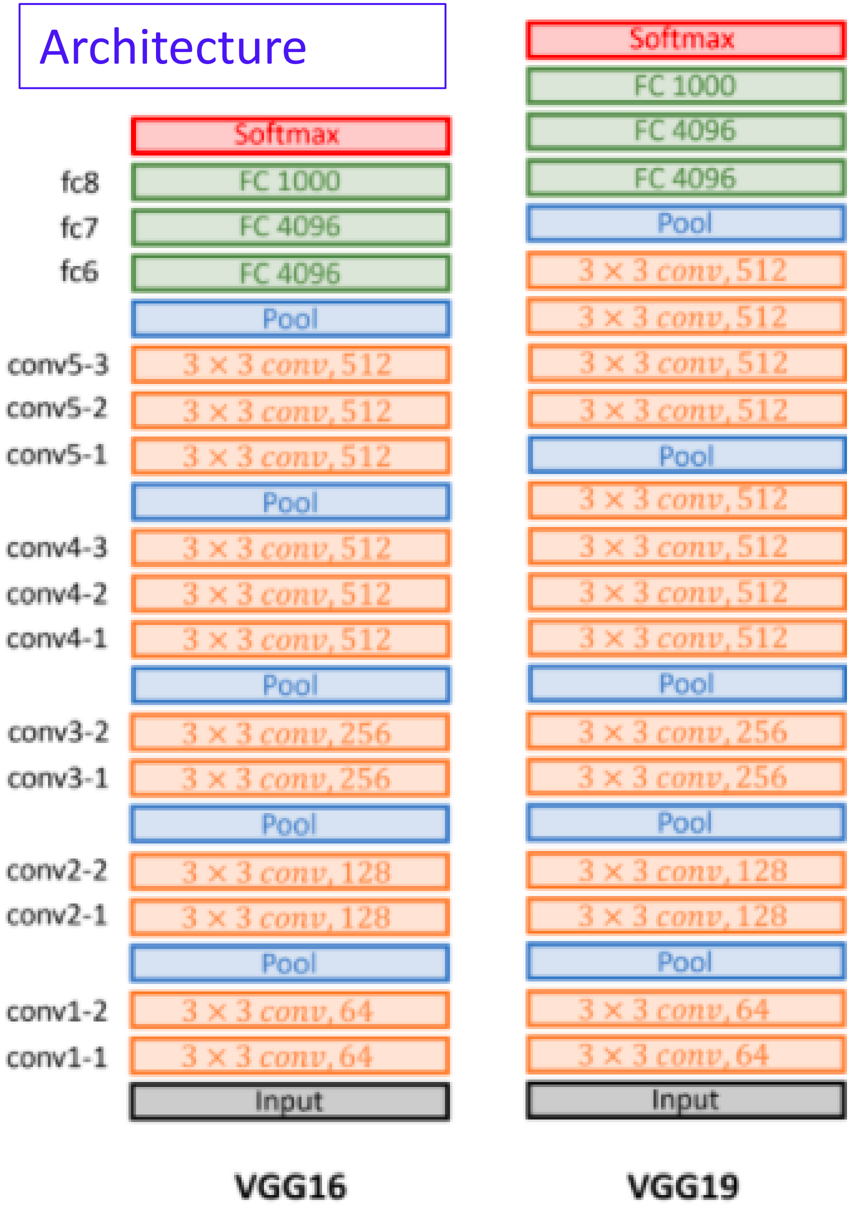
- Utilisation systématique de convolutions 3×3
- Reprise des grandes caractéristiques d'`AlexNet`, en les régularisant :
  - Diminution progressive de la taille de l'image ($224 \to 112 \to 56$ ...)
  - Augmentation progressive du nombre de filtres ($64 \to 128 \to 256$...)

VGG16 (2014)

- Number 16 in VGG−16 refers to the fact that this has 16 layers that have some weights. This is a pretty large network, and has a total of about 138 million parameters.

- Because VGG−16 does almost as well as the VGG−19 a lot of people will use VGG−16.

Architecture



Layers of $VGG-16$ and $VGG-19$

On retiendra…

- Une erreur courante consiste à utiliser des noyaux de convolution trop volumineux… Plutôt qu'une couche de convolution avec un noyau 5x5, vaut mieux empiler deux couches avec des noyaux 3×3 , la charge de calcul et le nombre de paramètres seront bien inférieures, avec de meilleures performances…

- L'exception concerne la première couche de convolution, elle a typiquement un grand noyau, par exemple (5x5), avec un pas de 2 ou plus. Cela permet de diminuer la dimension spatiale de l'image sans perdre trop d'information.

- Une pratique courante consiste à doubler le nombre de filtre après chaque couche de pooling : puisqu'une couche de pooling divise chaque dimension par un facteur 2, nous pouvons doubler ce nombre sans craindre de voir exploser le nombre de paramètres.

Application

Formasys

- Comprendre le transfert learning et réaliser une application de classification des photos
  https://www.youtube.com/watch?v=jRaPzSR98uk&t=372s&ab_channel=FORMASYS
- Code
  https://github.com/formasys/classification_pneumonia/blob/master/pneumonia_1.ipynb

- Connecter Google Colab à Kaggle pour importer les bases de données
  https://www.youtube.com/watch?v=up8R4jT49Ak&t=27s&ab_channel=FORMASYS

Noob Data Analyst

- Easiest Way to Download Kaggle Datasets using Opendataset in Jupyter Notebook
  https://www.youtube.com/watch?v=yYvSscuM8so&ab_channel=NoobDataAnalyst

```python
from glob import glob

from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dense, Flatten
from keras.models import Model
```

[4]
```python
# importer la base de données
data_path='/content/drive/My Drive/Kaggle/pneumonia_data/chest_xray'
```

[5]
```python
#spécifier le chemin des bases de données train et test
train_dir=os.path.join(data_path,'train')
test_dir=os.path.join(data_path,'test')
```

```python
# Préparer le modèle de base
IMG_SHAPE=(224,224,3)
base_model=VGG16(input_shape=IMG_SHAPE,include_top=False,weights='imagenet')
base_model.summary()
```

```
block5_pool (MaxPooling2D)   (None, 7, 7, 512)        0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

[7]
```python
# bloquer le modèle de base
base_model.trainable=False
```

+ Code + Texte

[9]
```python
base_model.output
```

```
<tf.Tensor 'block5_pool/MaxPool:0' shape=(None, 7, 7, 512) dtype=float32>
```

```
#Ajouter les couches de sorties
x=Flatten()(base_model.output)
prediction=Dense(2,activation='softmax')(x)
```

```
[ ]
```

```
[20] #création du modèle global
model=Model(inputs=base_model.input,outputs=prediction)
model.summary()
```

```
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 14,764,866
Non-trainable params: 0
_________________________________________________________________
```

```
[24] model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```python
# Creer les photo augmentées
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory(train_dir,
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_dir,
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```python
# Appliquer le modèle
model.fit_generator(training_set, epochs=5, validation_data=test_set)
```

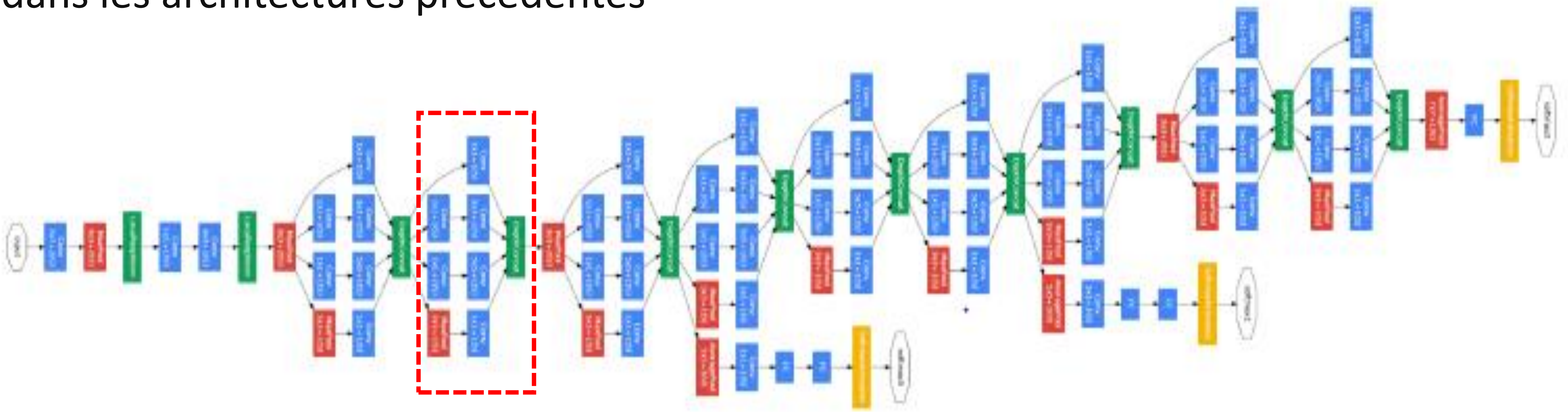
```
valid_loss, valid_accuracy = model.evaluate_generator(test_set)
```

```
print("Accuracy after transfer learning: {}".format(valid_accuracy))
```

2. Architectures avancées : Inception, ResNet

GoogLeNet (ou Inception, 2014)

- Cette architecture est caractérisée par la présence de sous-réseaux nommée modules *Inception*
- Ces modules permettent à GoogLeNet d'utiliser les paramètres beaucoup plus efficacement que dans les architectures précédentes



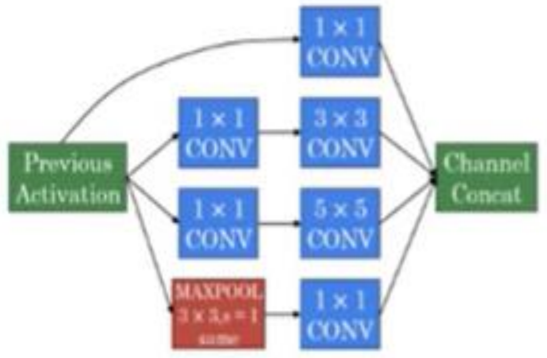
Module Inception

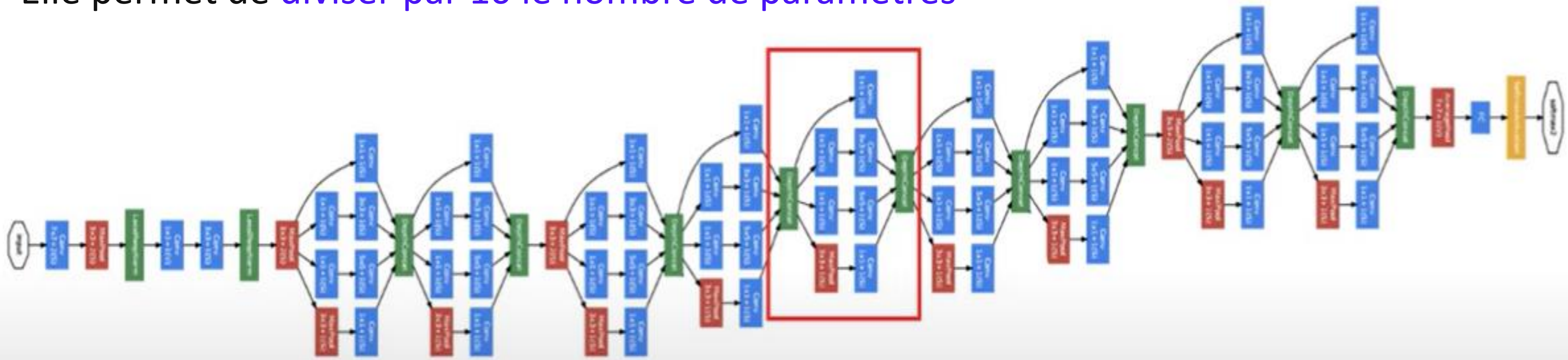
$\simeq$ 7M paramètres, 22 couches

[Szegedy et al.] Going deeper with convolutions.

GoogLeNet (ou Inception, 2014)

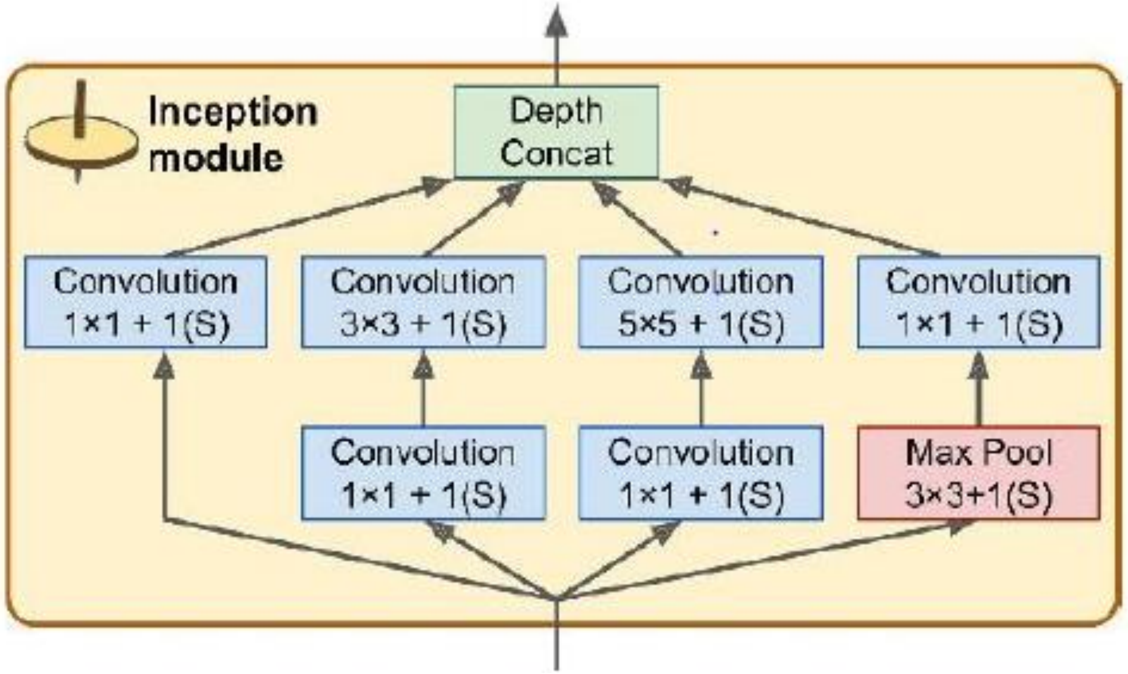
- L'architecture GoogLeNet a été développée par Christian Szegedy et *al.* de Google Research, et il a remporté le défi ILSVRC 2014 en faisant passer le top-5 error rate en dessous de 7 %.
- Cette excellente performance vient en grande partie du fait que le réseau était beaucoup plus profond que les précédents CNN.
- Elle permet de diviser par 10 le nombre de paramètres



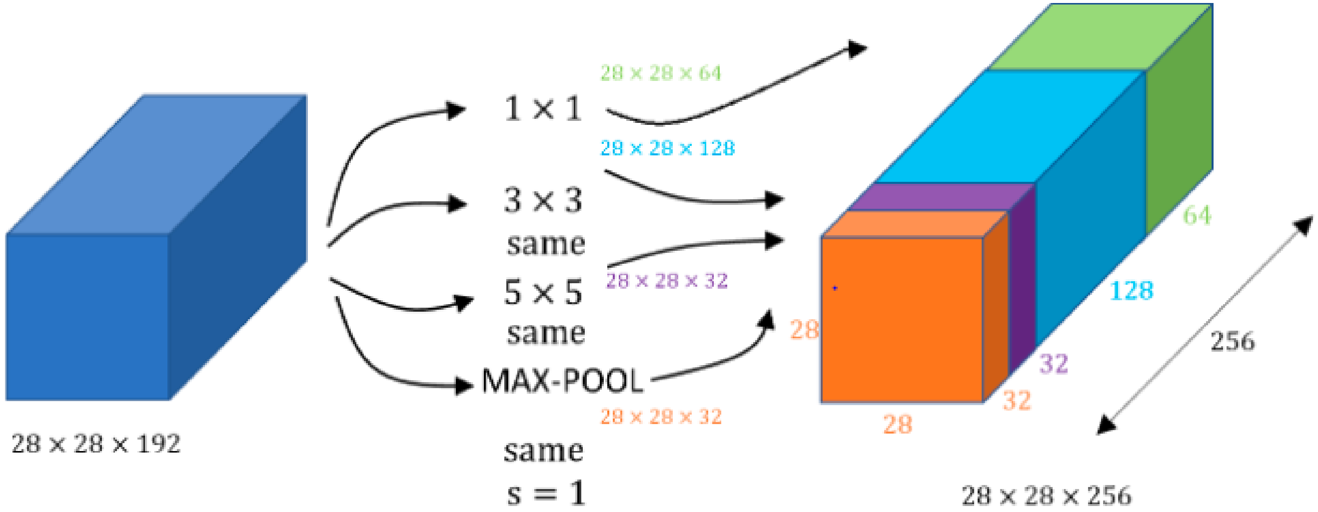


Inception module

- When designing a layer for a convnet we might have to choose between a 1×1 filter, a 3×3 or 5×5 or maybe a pooling layer.
- An Inception network solves this by saying:“ Why shouldn’t we apply them all ? » ”.
- The second set of convolutional layers uses different kernel sizes (1 × 1, 3 × 3, and 5 × 5), allowing them to capture patterns at different scales.

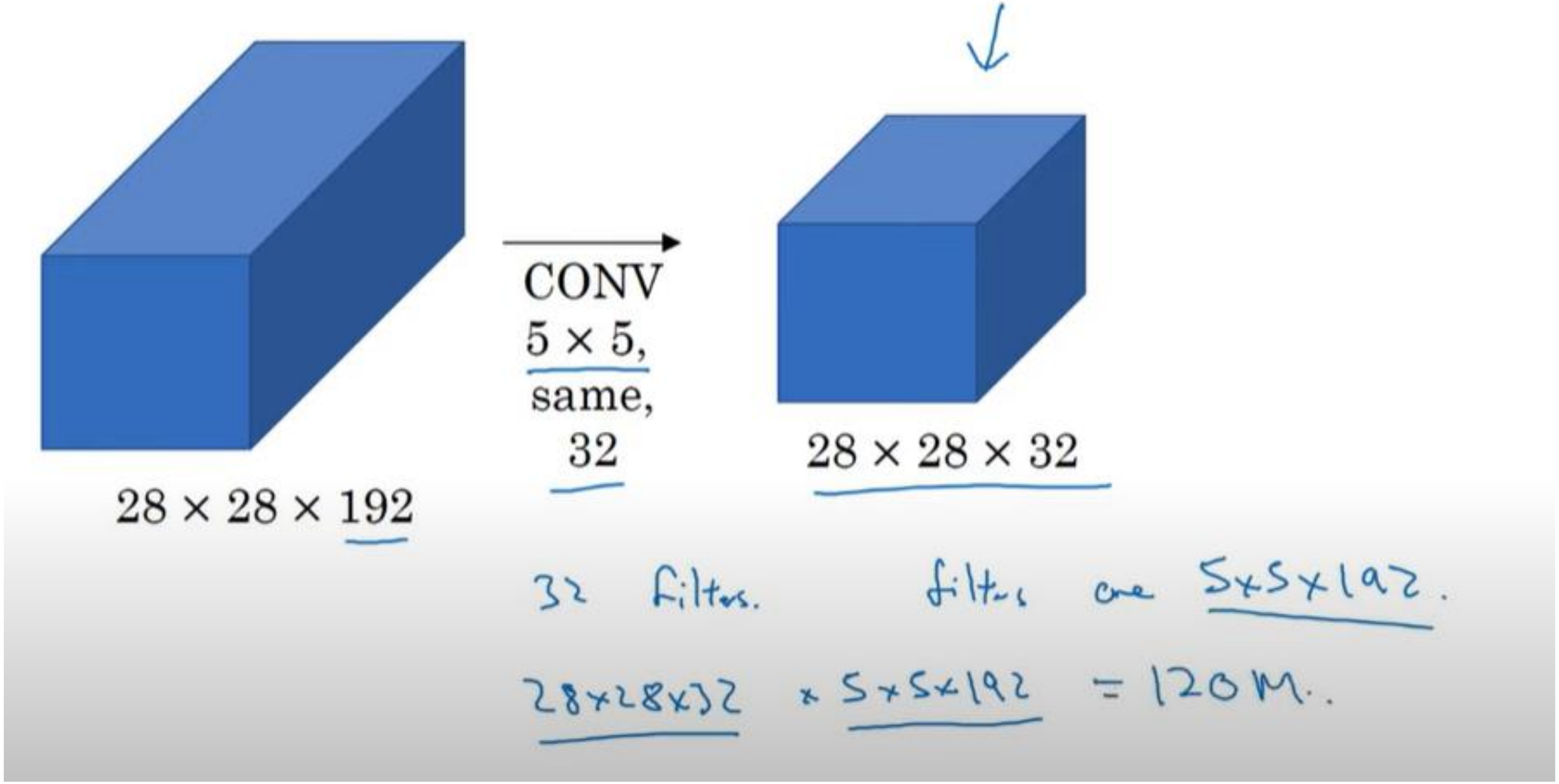


- Every single layer uses a stride of 1 and "same" padding (even the max pooling layer), so their outputs all have the same height and width as their inputs. This makes it possible to concatenate all the outputs along the depth dimension in the final depth concatenation layer.



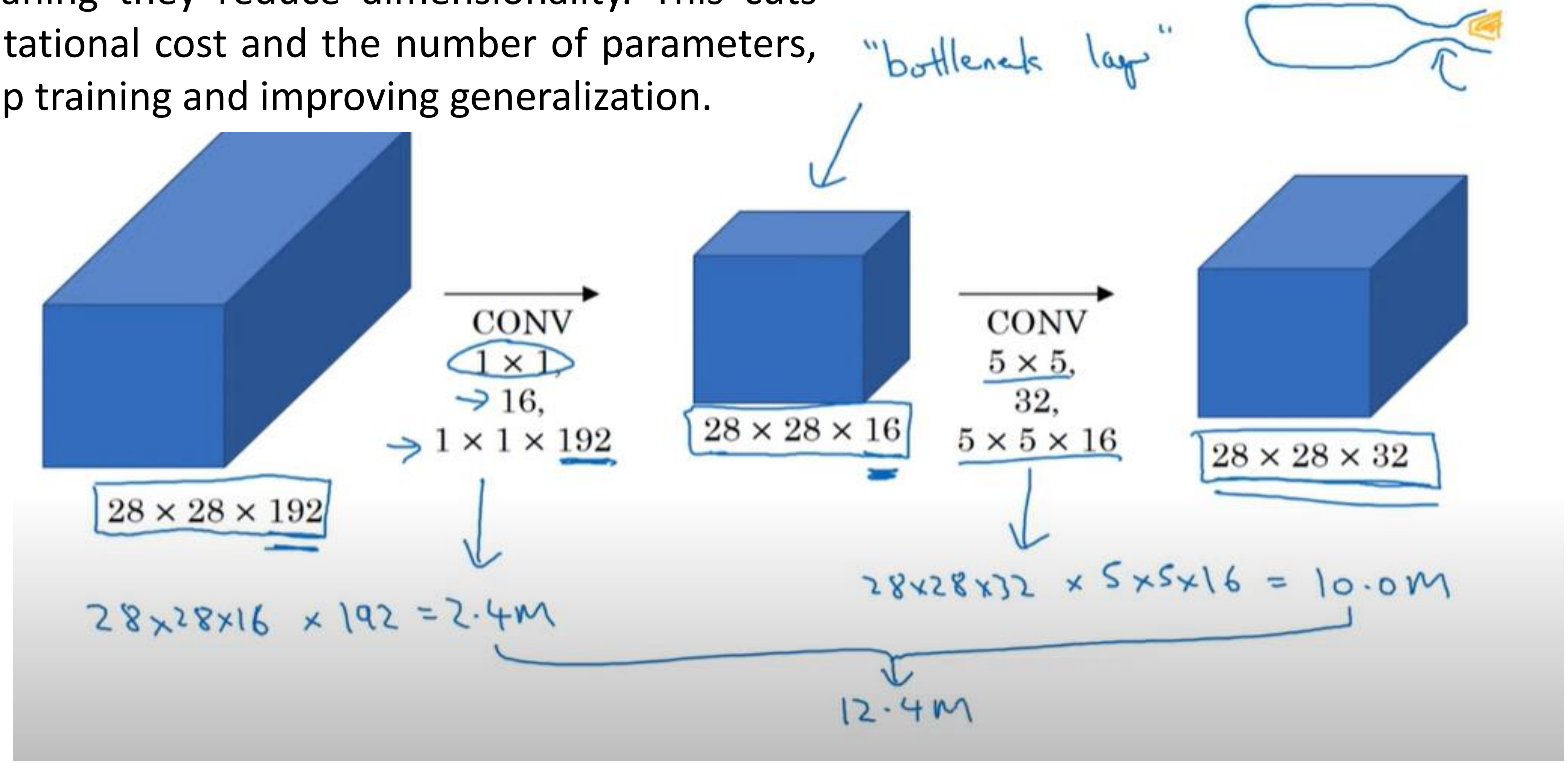
The problem of computation cost

- En considérant un kernel 5x5, le nombre d’opérations avoisine les 120M…



Using 1x1 convolution

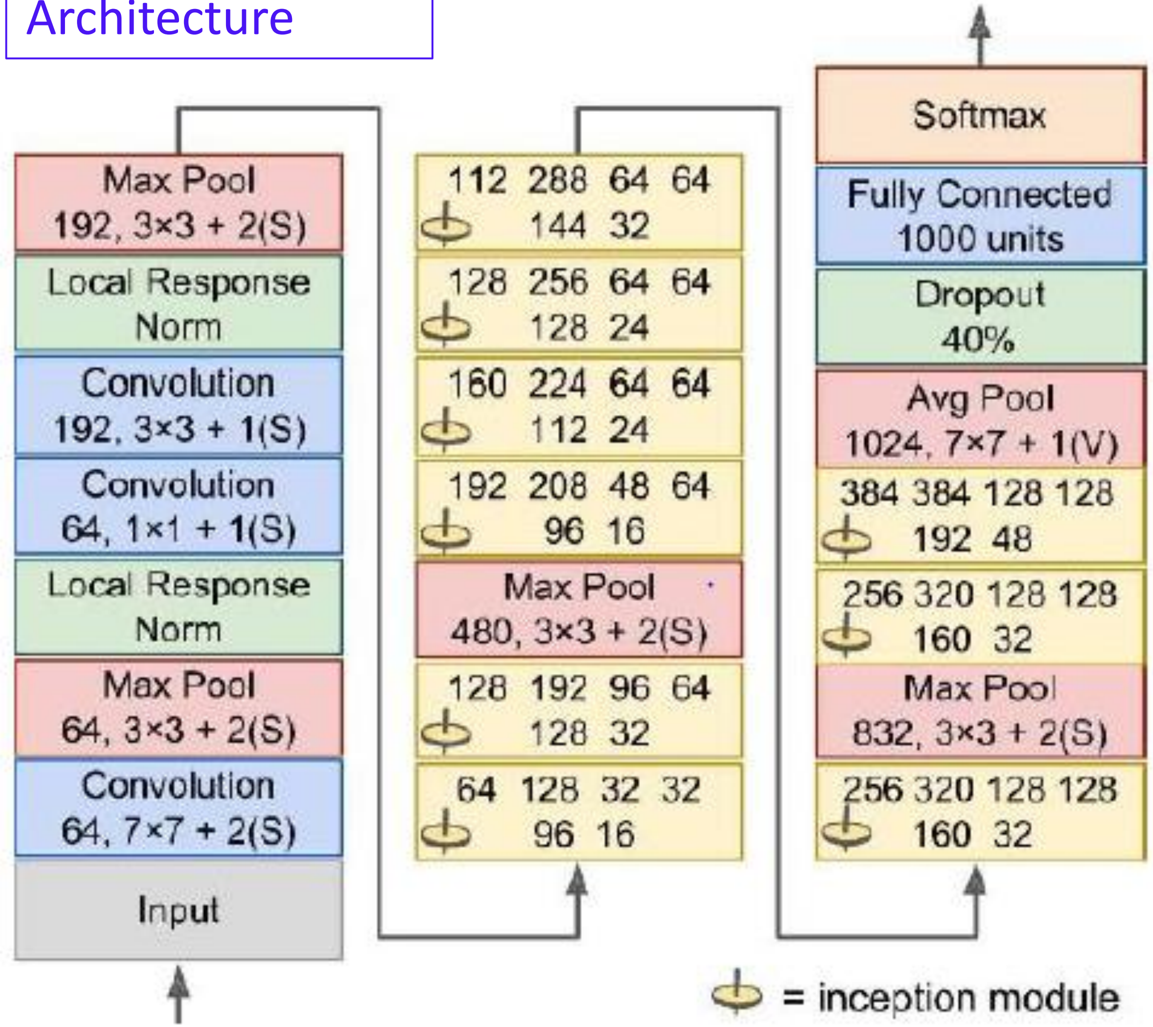
- 1x1 kernels are configured to output fewer feature maps than their inputs, so they serve as *bottleneck layers*, meaning they reduce dimensionality. This cuts the computational cost and the number of parameters, speeding up training and improving generalization.



- Les bottlenk layers font passer le nombre d'opérations de 120M à 12.4M !

GoogLeNet (ou Inception, 2014)

Architecture



- Des variantes de l’architecture GoogLeNet, notamment *Inception-v3* et *Inception-v4.*

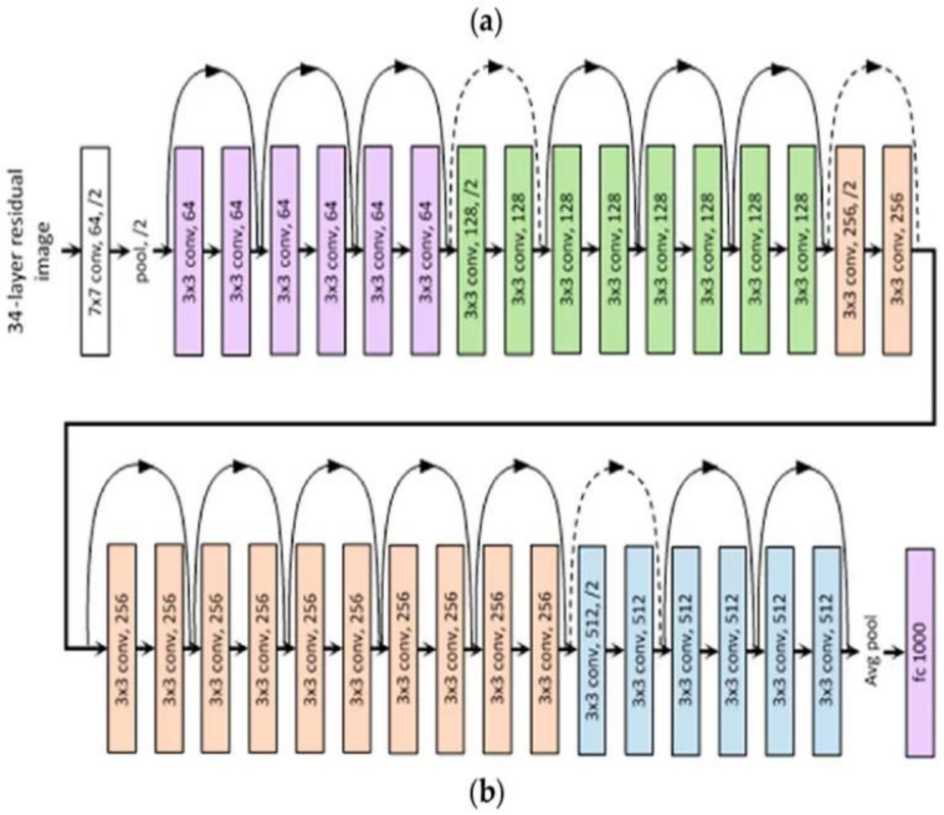
ResNet (2015)

- He et al. won the ILSVRC 2015 challenge using a *Residual Network* (or*ResNet*), that delivered an astounding top-five error rate under 3.6%.

- The winning variant used an extremely deep CNN composed of 152 layers (other variants had 34, 50, and 101 layers).

- It confirmed the general trend: models are getting deeper and deeper, with fewer and fewer parameters.

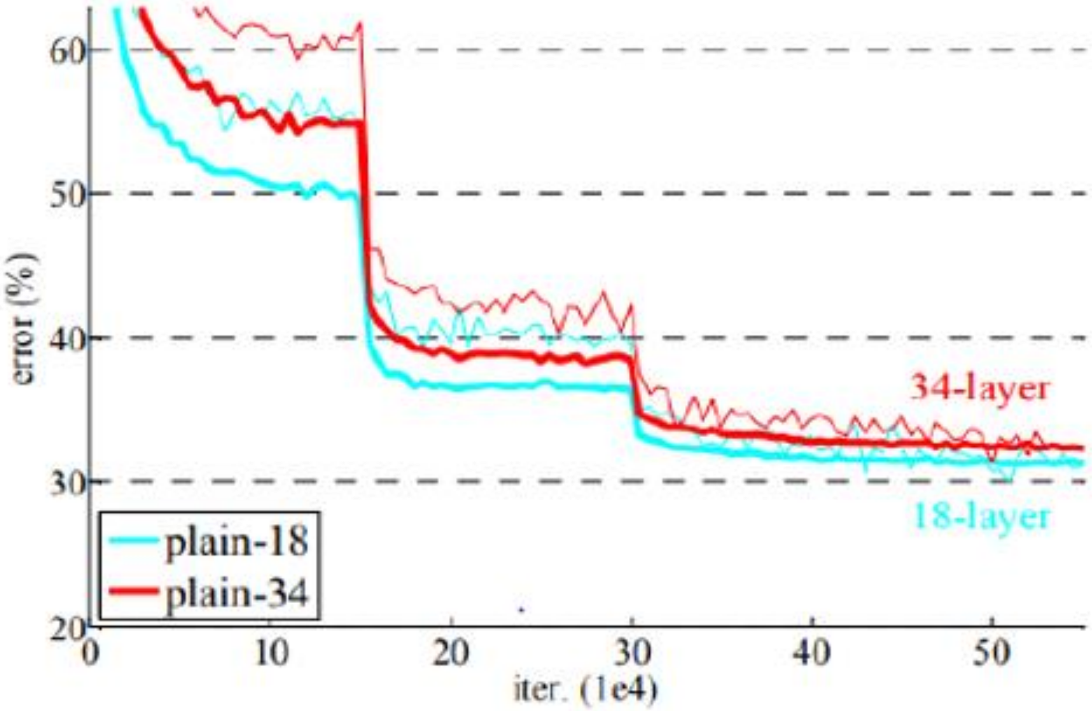
- The key to being able to train such a deep network is to use *skip connections* (also called *shortcut connections*): the signal feeding into a layer is also added to the output of a layer located a bit higher up the stack.



ResNet-34 Layered architecture

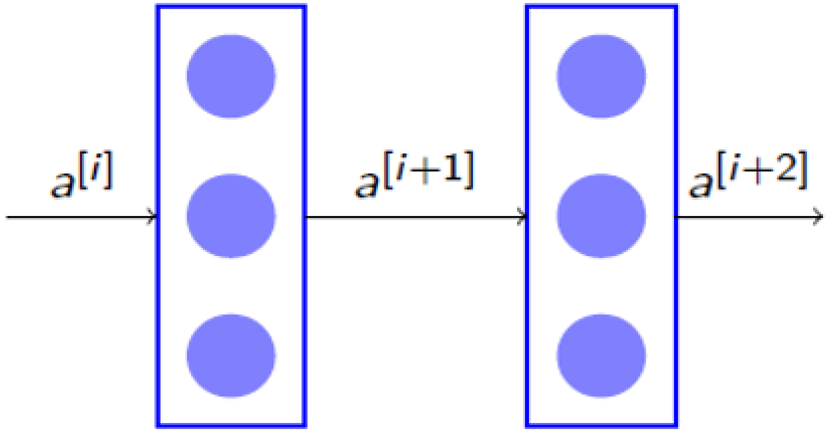
ResNet (2015)

- Les réseaux les plus profonds devraient en théorie permettre d'approximer de plus en plus efficacement l'ensemble d'apprentissage. En pratique, ils posent de nombreux problèmes d'optimisation, comme celui de l'evanescence des gradients (mais aussi celui de l'explosion des gradients).



Training on ImageNet. This curves denote training error, and bold curves denote validation error: plain networks of 18 and 34 layers. (He et *al.* 2015).

Blocs résiduels

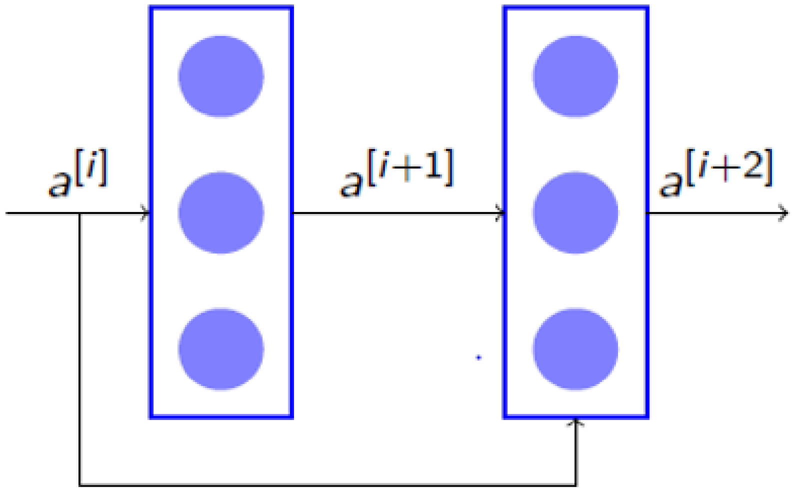


- Cette vue abstrait les liaisons synaptiques entre les 2 couches i+1 et i+2 :

$$a^{[i+1]} = \mathrm{ReLU}(W^{[i+1]}a^{[i]} + b^{[i+1]})$$

et

$$a^{[i+2]} = \mathrm{ReLU}(W^{[i+2]}a^{[i+1]} + b^{[i+2]})$$

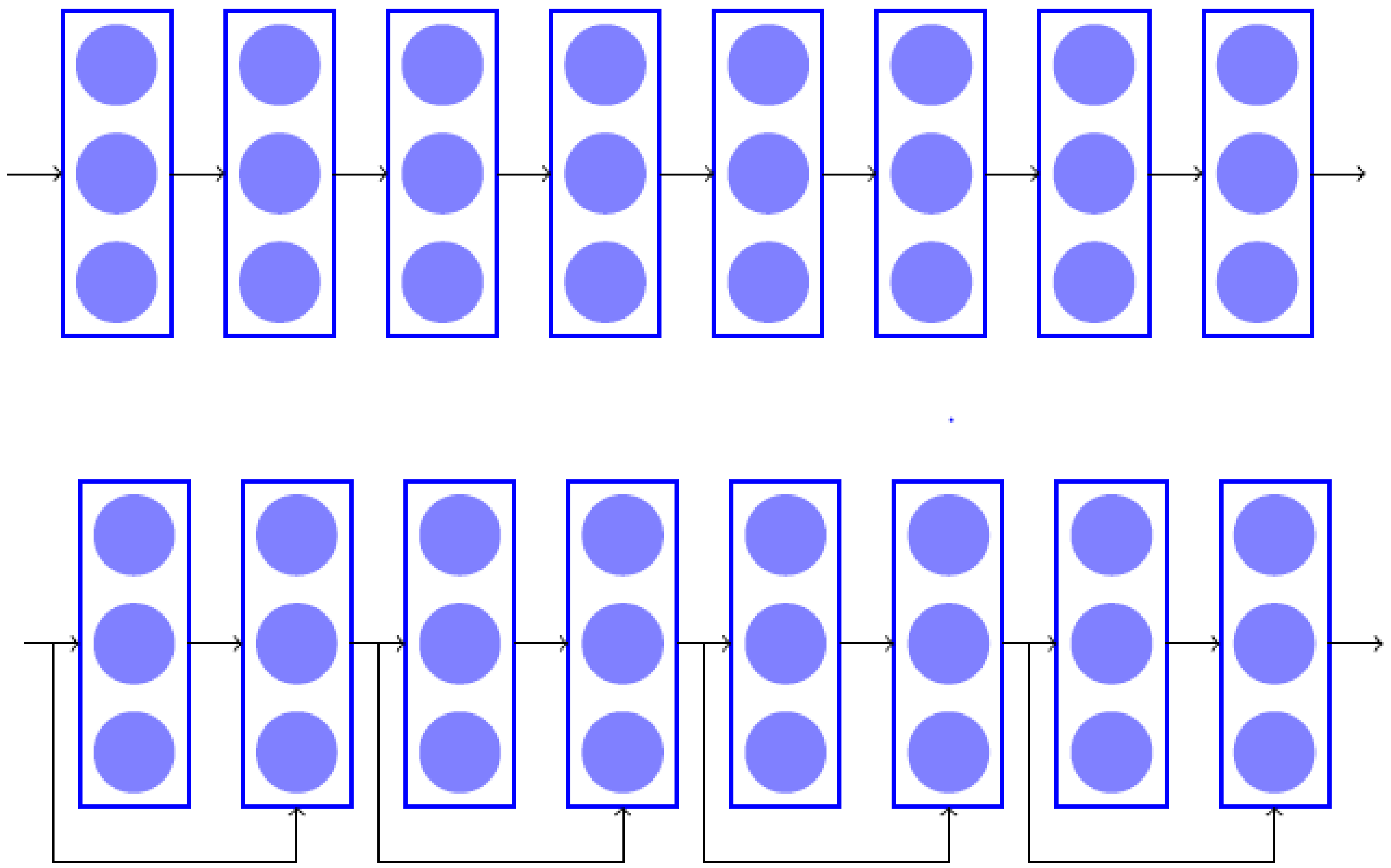


- On peut ajouter une connexion comme présenté ci-dessous (skip-connexion) pour former un **bloc résiduel** :

$$a^{[i+2]} = \mathrm{ReLU}(W^{[i+2]}a^{[i+1]} + b^{[i+2]} + a^{[i]})$$

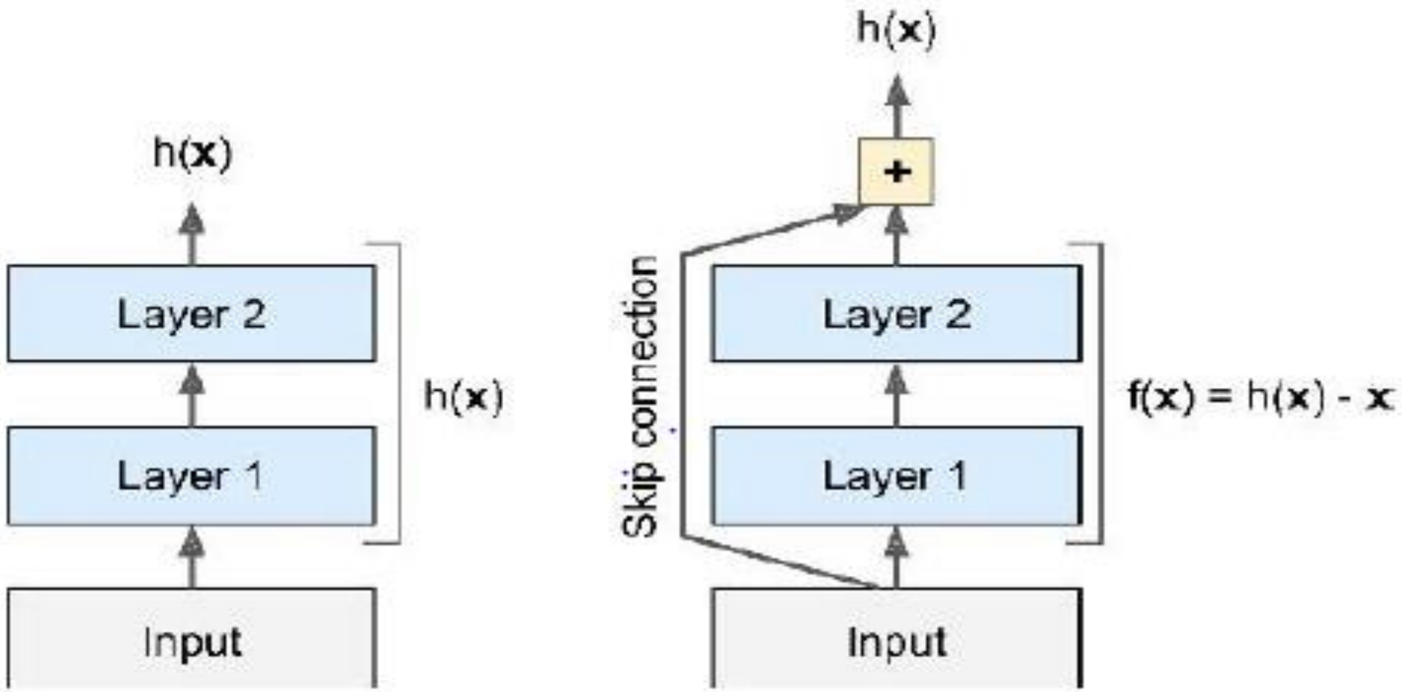
Blocs résiduels

- Rendre résiduel un réseau de neurones :



Blocs résiduels

- When training a neural network, the goal is to make it model a target function $h(\mathbf{x})$. If you add the input $\mathbf{x}$ to the output of the network (i.e., you add a skip connection), then the network will be forced to model $f(\mathbf{x}) = h(\mathbf{x}) - \mathbf{x}$ rather than $h(\mathbf{x})$. This is called *residual learning* :

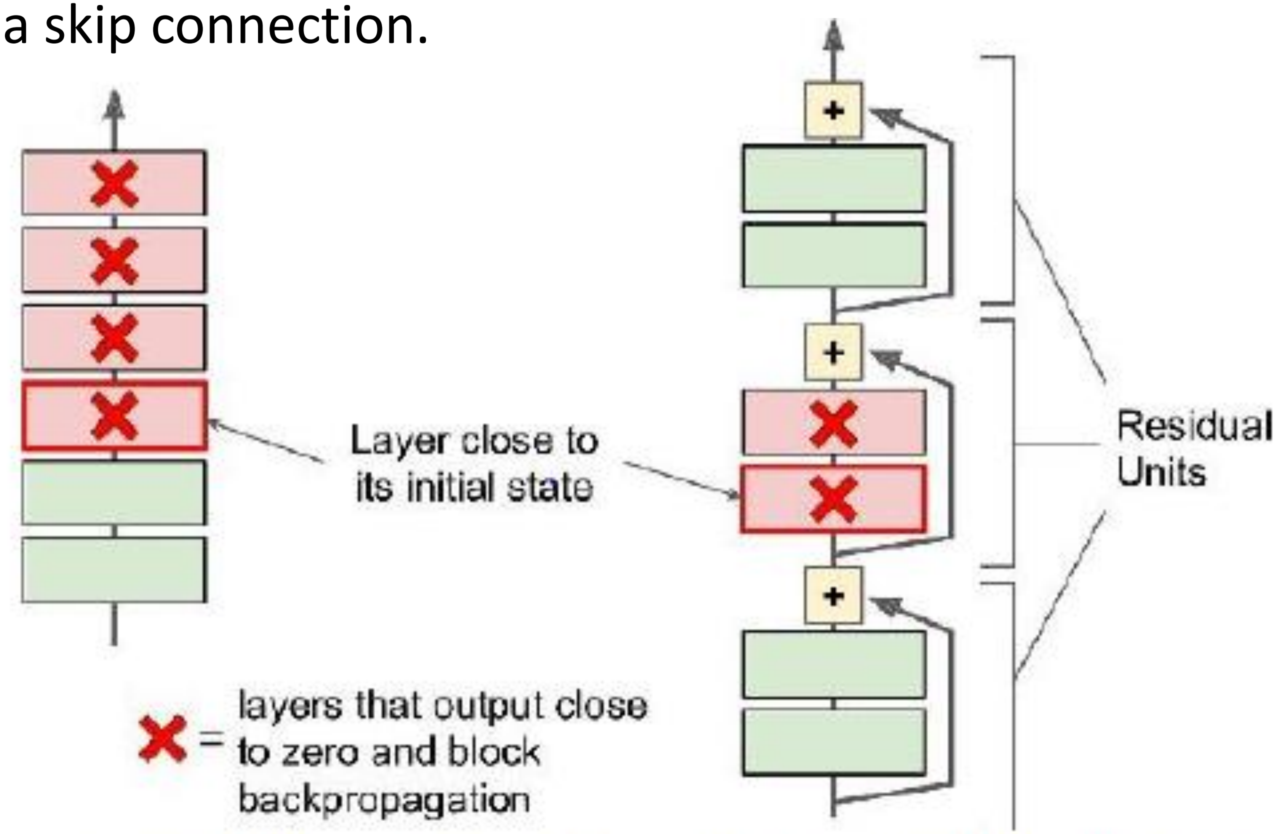


Residual learning

- When you initialize a regular neural network, its weights are close to zero, so the network just outputs values close to zero. If you add a skip connection, the resulting network just outputs a copy of its inputs; in other words, it initially models the identity function. If the target function is fairly close to the identity function (which is often the case), this will speed up training considerably.

Blocs résiduels

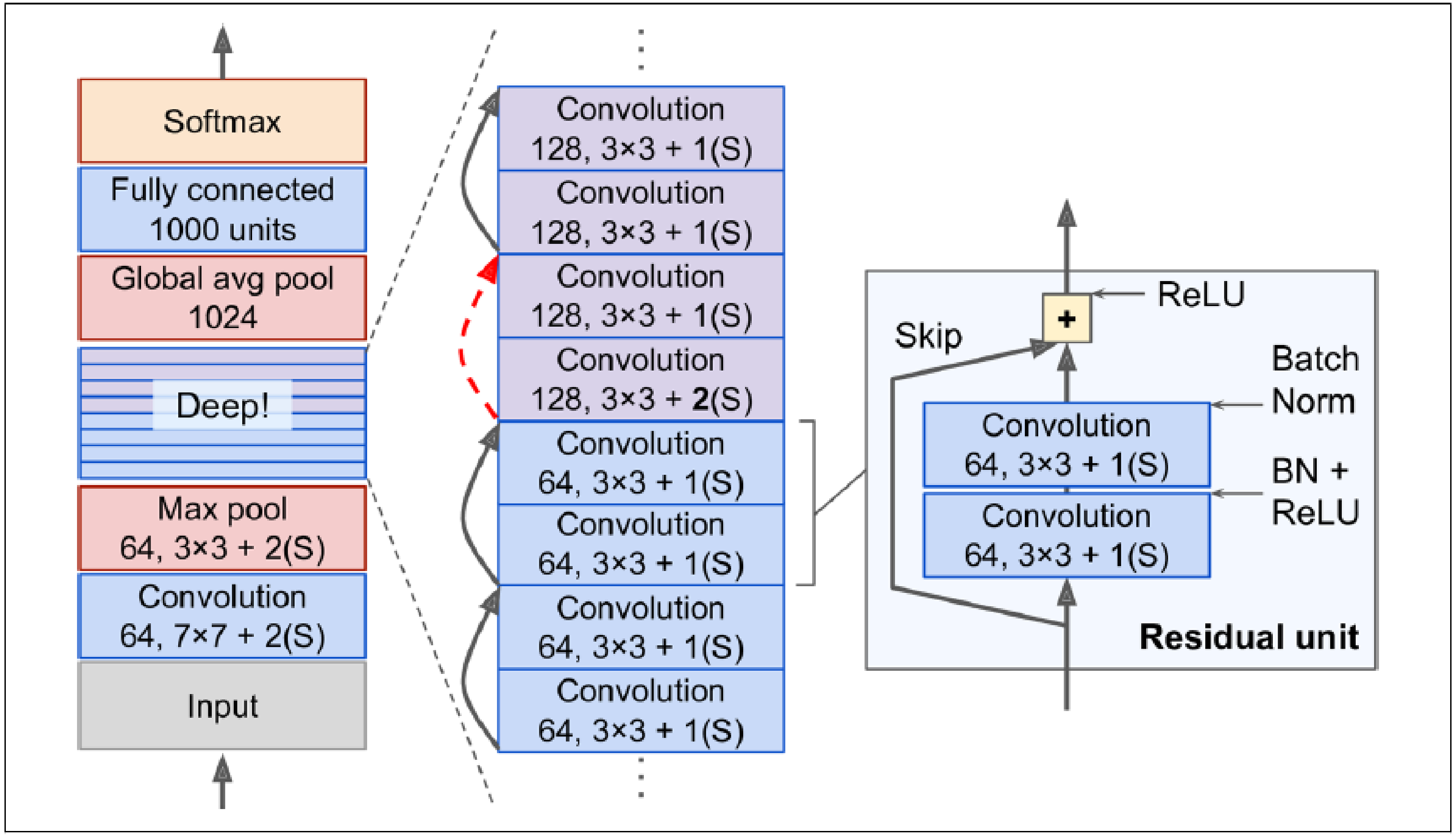
- Moreover, if you add many skip connections, the networks cab start making progress even if several layers have not started learning yet.
- Thanks to skip connections, the signal can easily make its way across the whole network.
- The deep residual network can be seen as a stack of residual units, where each residual unit is a small network whith a skip connection.



Regular deep neural network (left) and deep residual network (right)

Architecture

- It is surprisingly simple. It starts and ends exactly like GoogLeNet (except without a dropout layer), and in between is just a very deep stack of simple residual units.
- Each residual unit is composed of two convolutional layers (and no pooling layer!), with Batch Normalization (BN) and ReLU activation, using 3 × 3 kernels and preserving spatial dimensions (stride 1, "same" padding).

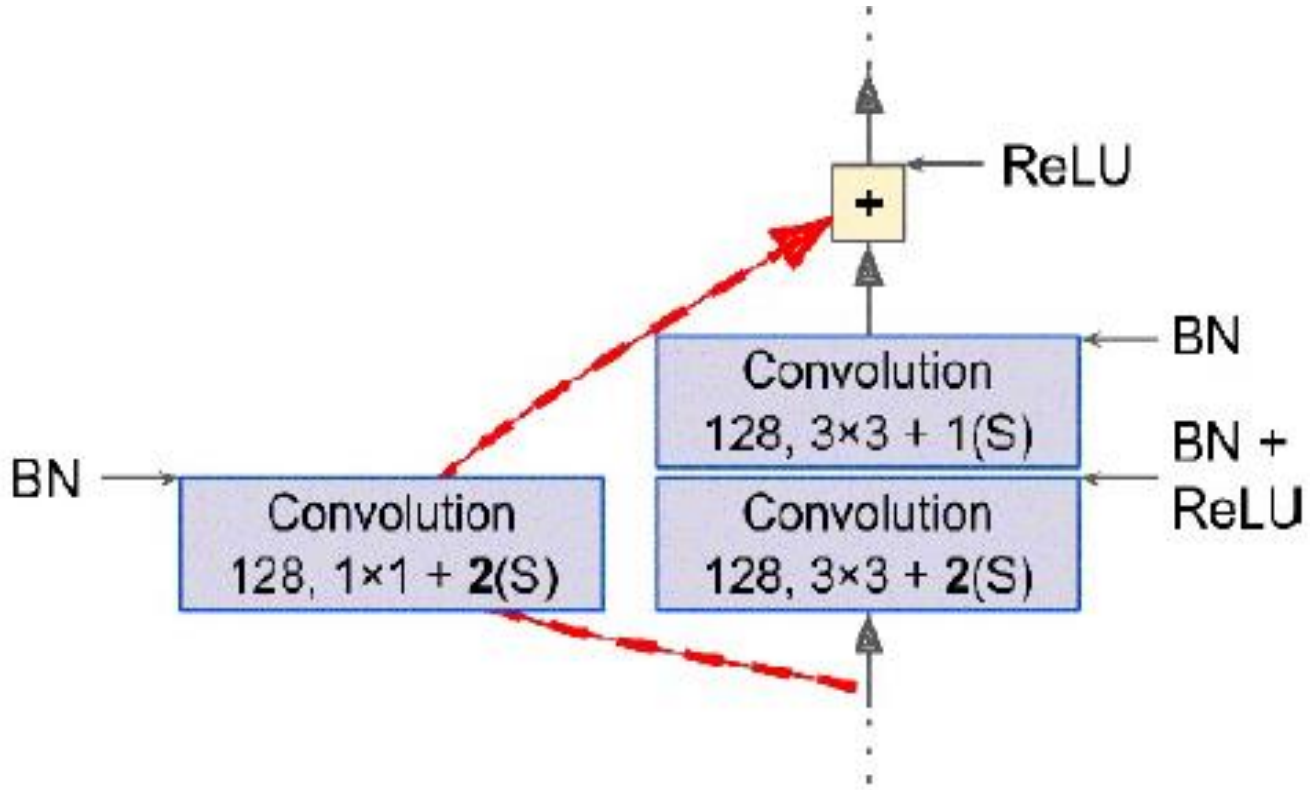


Architecture

- Note that the number of feature maps is doubled every few residual units, at the same time as their height and width are halved (using a convolutional layer with stride 2).

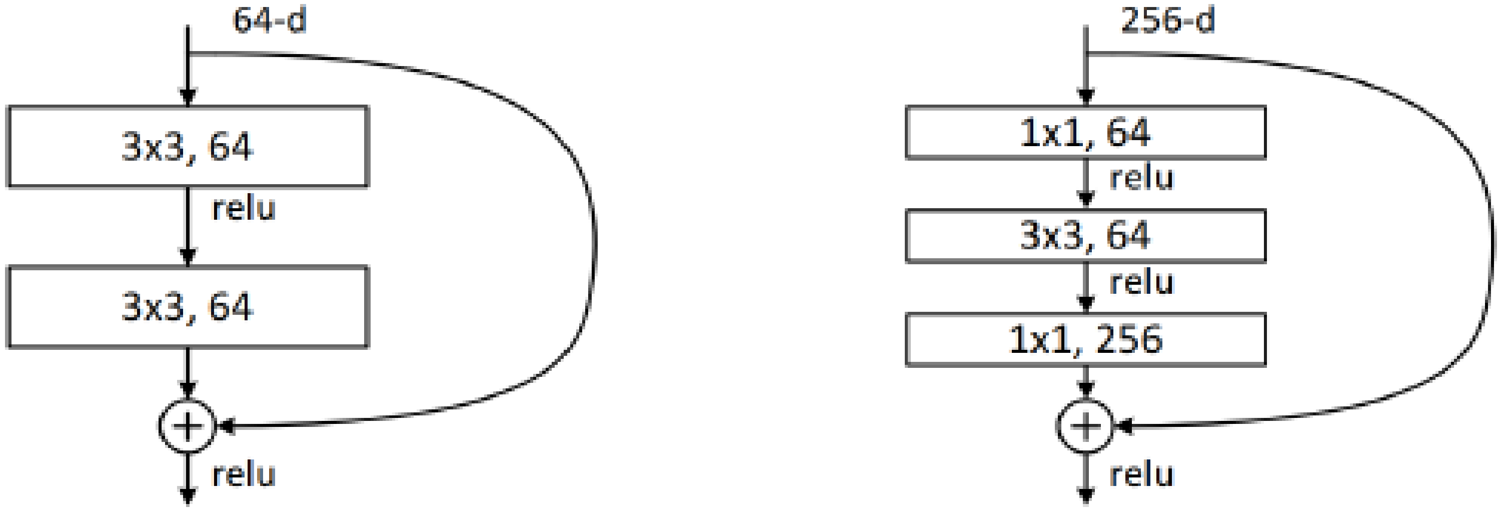
- When this happens the inputs cannot be added directly to the outputs of the residual unit since they don’t have the same shape

- To solve this problem, the inputs are passed through a 1 × 1 convolutional layer with stride 2 and the right number of output feature maps

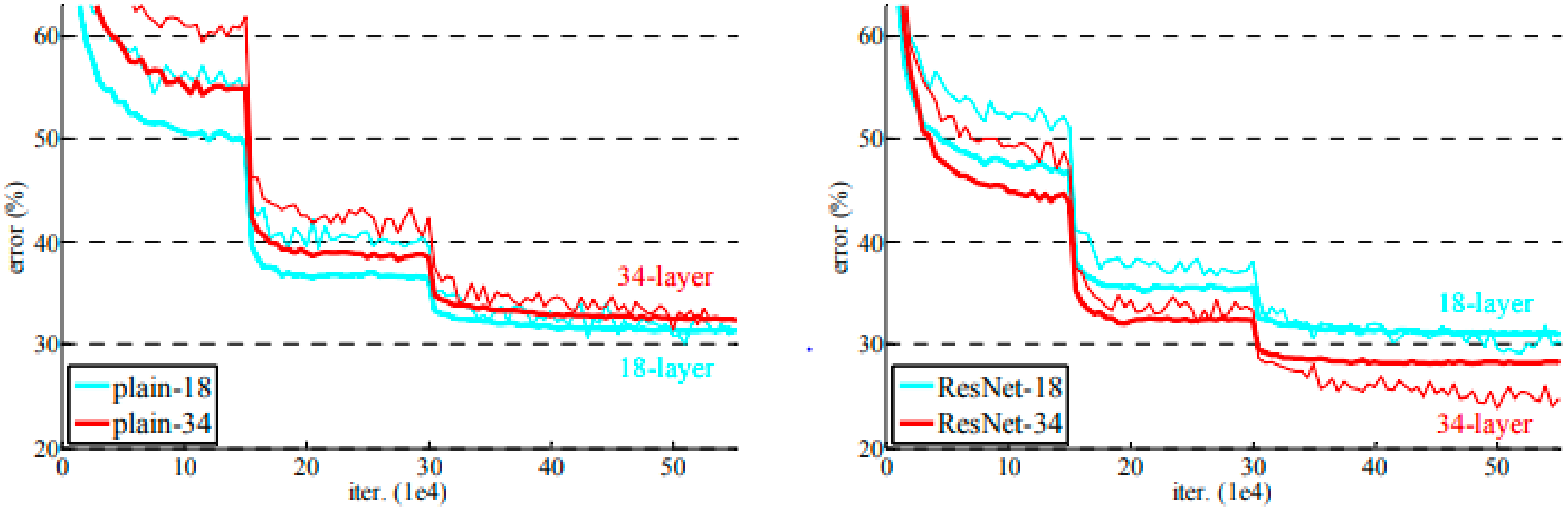


Architecture

- Les blocs résiduels ont permis aux auteurs d'entraîner des réseaux de plusieurs centaines de couches. Le réseau qui a remporté le *challenge* `ImageNet` en 2015 comptait d'ailleurs 152 couches.
- Les problèmes de ces architectures très profondes ne sont plus liés à l'optimisation mais aux performances, ce qui a motivé les auteurs à introduire des blocs résiduels de format différent :



Performance de ResNet en fonction du nombre de couches



Training on ImageNet. This curves denote training error, and bold curves denote validation error:

- (a) plain networks of 18 and 34 layers.
- (b) ResNets of 18 and 34 layers

[He et al.] Deep Residual Learning for Image Recognition

Keras applications : Performance des réseaux pré-entraînés

keras.io/api/applications/

convention, "Height-Width-Depth".

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9

Liens

- DeepLearningAI : Andrex Ng
  Inception I
  https://www.youtube.com/watch?v=C86ZXvgpejM&ab_channel=DeepLearningAI

  Inception II
  https://www.youtube.com/watch?v=KfV8CJh7hE0&ab_channel=DeepLearningAI

- #017 CNN Inception Network
  https://datahacker.rs/deep-learning-inception-network/

- Deep Learning in the Trenches: Understanding Inception Network from Scratch
  https://www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/

Liens

Utiliser des modèles pré-entraînés pour un transfert d'apprentissage

- Tutorial 28- Create CNN Model Using Transfer Learning using Vgg 16, Resnet ...
  https://www.youtube.com/watch?v=zBOavqh3kWU&ab_channel=KrishNaik
  https://github.com/krishnaik06/Transfer-Learning/blob/master/face_Recognition.py

- The Keras blog : Building powerful image classification models using very little data By Francois Chollet
  The Cats and Dogs data.
  https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

Mini-Projet : Cats Vs Dogs

Data Cats Vs Dogs

https://www.kaggle.com/datasets/arpitjain007/dog-vs-cat-fastai

Sentdex

Chargement de vos propres données - Les bases du Deep Learning avec Python, TensorFlow et Keras p.2

https://www.youtube.com/watch?v=j-3vuBynnOE&ab_channel=sentdex

How to use your trained model - Deep Learning basics with Python, TensorFlow and Keras p.6

https://www.youtube.com/watch?v=A4K6D_gx2Iw&ab_channel=sentdex

Balaji Srinivasan

Préprocessing de la base

Image Classification with Keras, Tensorflow | Cat Vs Dog Prediction | Convolution Neural Networks P1

https://www.youtube.com/watch?v=FLf5qmSOkwU&ab_channel=BalajiSrinivasan

Application d'un CNN basique

Image Classification with Keras, Tensorflow | Cat Vs Dog Prediction | Convolution Neural Networks P2

https://www.youtube.com/watch?v=4ae13fiKDqo&ab_channel=BalajiSrinivasan

Nicholas

Modification du VGG16 pour l'appliquer à Cats Vs Dogs

ECC4306 - Dog-Cat Image Classifier Model Using VGG16

https://www.youtube.com/watch?v=F4z3vnX4RDk&t=502s&ab_channel=Nicholas