

# APPRENTISSAGE PROFOND EN IMAGES ET VISION PAR ORDINATEUR

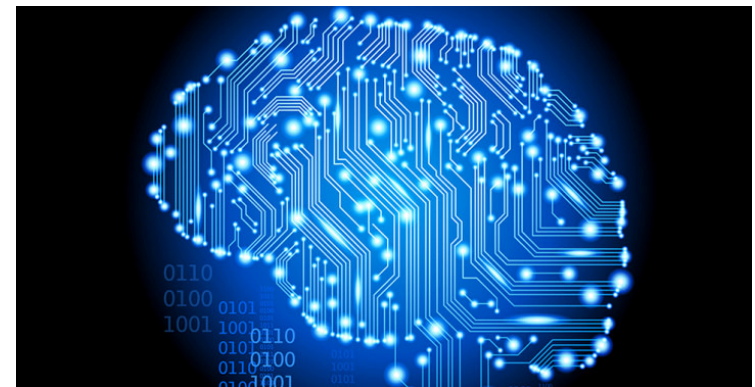
---

Alexandre Meyer<sup>1</sup>

<sup>1</sup>Equipe SAARA, laboratoire LIRIS



Master ID3D



# Papiers à lire

Soccer on Your Tabletop

Konstantinos Rematas, Ira Kemelmacher-Shlizerman, Brian Curless, Steve Seitz

CVPR 2018

<https://arxiv.org/abs/1806.00890>

GAN Dissection: Visualizing and Understanding Generative Adversarial Networks

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B. Tenenbaum<sup>1</sup>, William T. Freeman, Antonio Torralba

<https://gandisection.csail.mit.edu/> (ESSAYER LA DEMO)

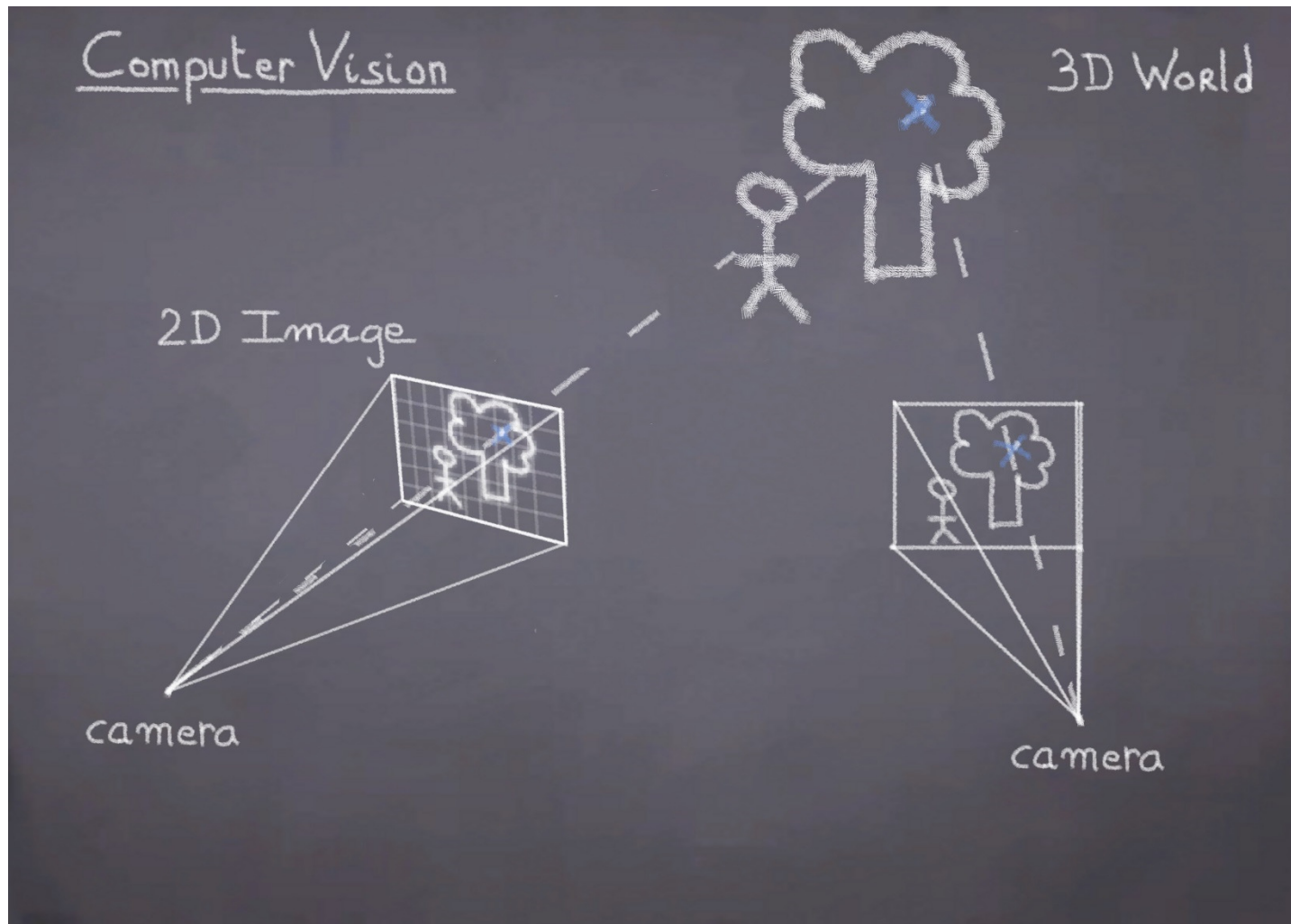
FaceNet: A Unified Embedding for Face Recognition and Clustering

Florian Schroff, Dmitry Kalenichenko, James Philbin

CVPR 2015

<https://arxiv.org/abs/1503.03832>

# Vision par ordinateur



# Vision par ordinateur et vision humaine



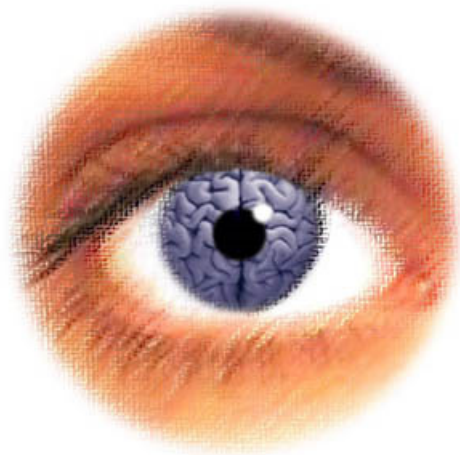
Nature

≠

inspire l'



Artificiel



≠





# Vision par ordinateur et vision humaine

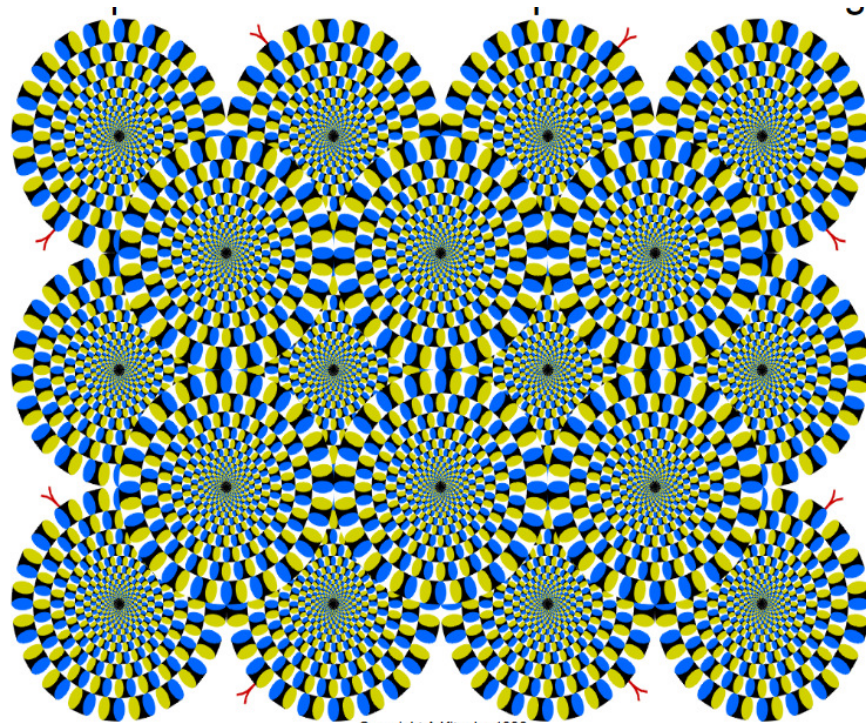
- Vision par ordinateur reste limitée même si d'énormes progrès ont été réalisés ...



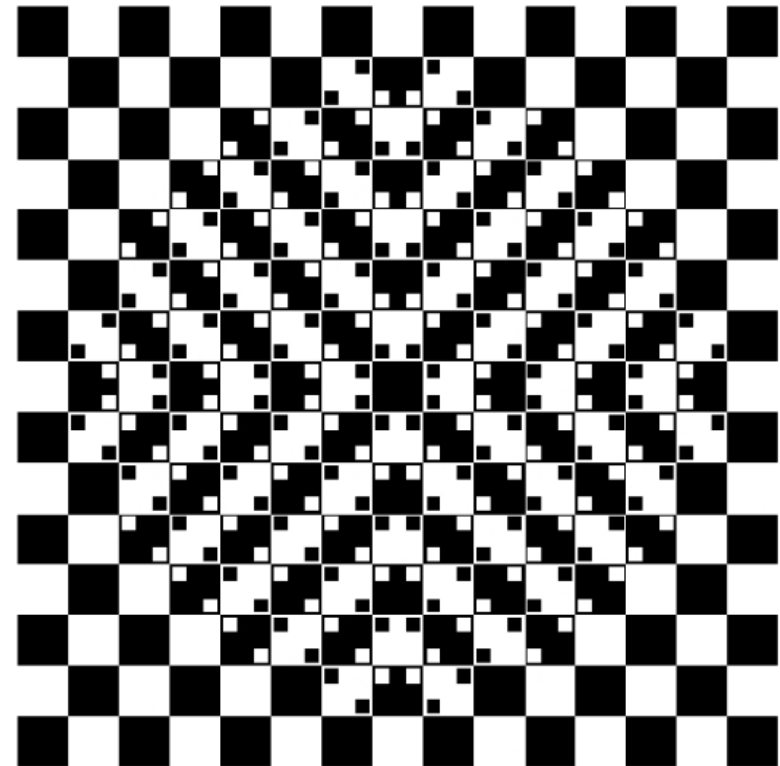
26x27 pixels

# Vision par ordinateur et vision humaine

- Vision par ordinateur reste limitée
  - Pour l'instant

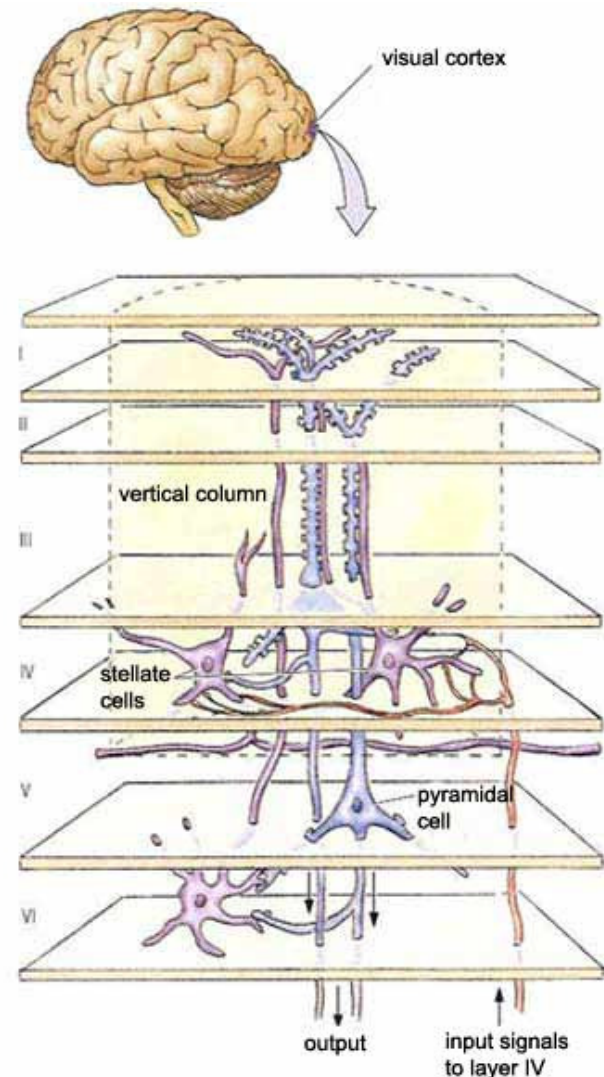


Copyright A.Kitaoka 1998

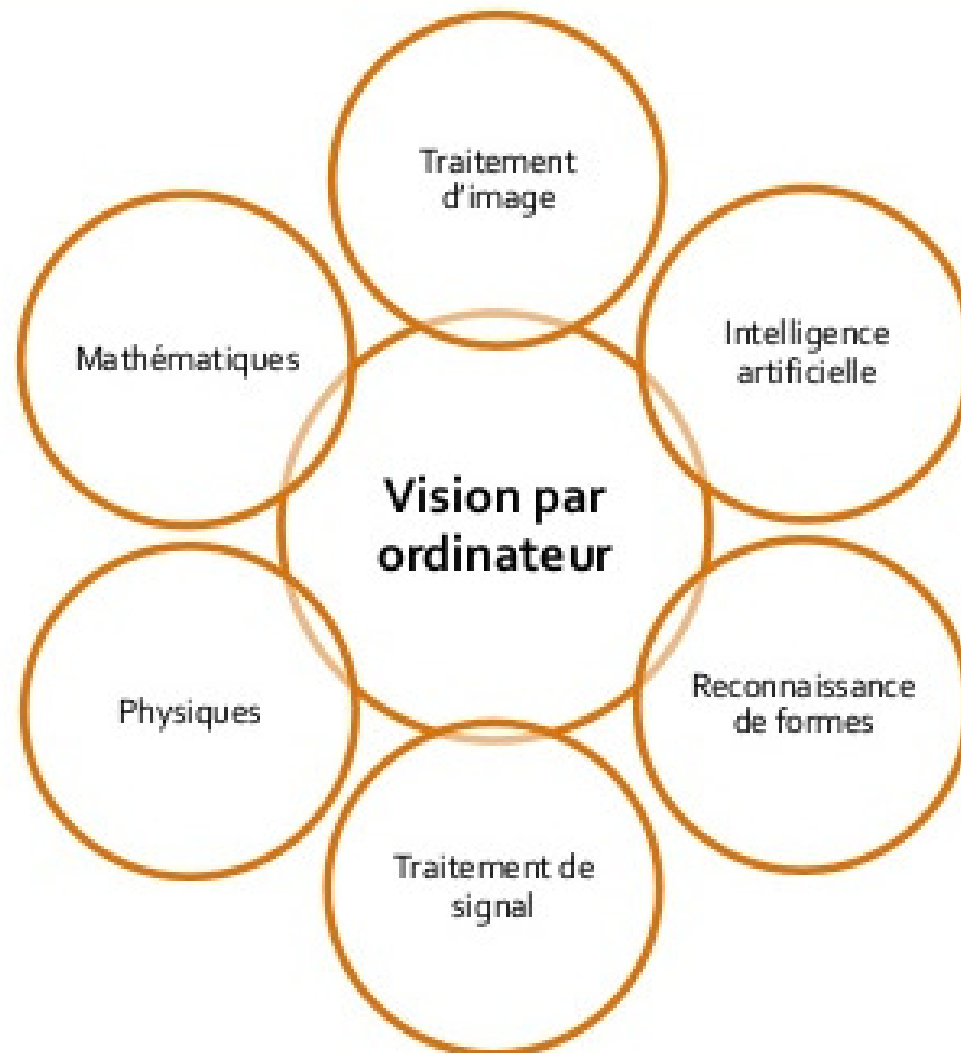


# Vision par ordinateur et vision humaine

- Vision par ordinateur
  - Traitement d'images
    - Changer la luminosité
    - Mettre en évidence certains aspects
    - ...
  - Reconnaissance des formes
    - Retrouver les lignes, les cercles, etc.
    - ...
    - Retrouver des visages
- Vision par ordinateur
  - Identifier les motifs
    - « On voit un visage humain »
    - « Il s'agit de Paul »
  - Identifier des actions
    - « La personne vis un boulon »
  - Analyser une action



# Vision par ordinateur



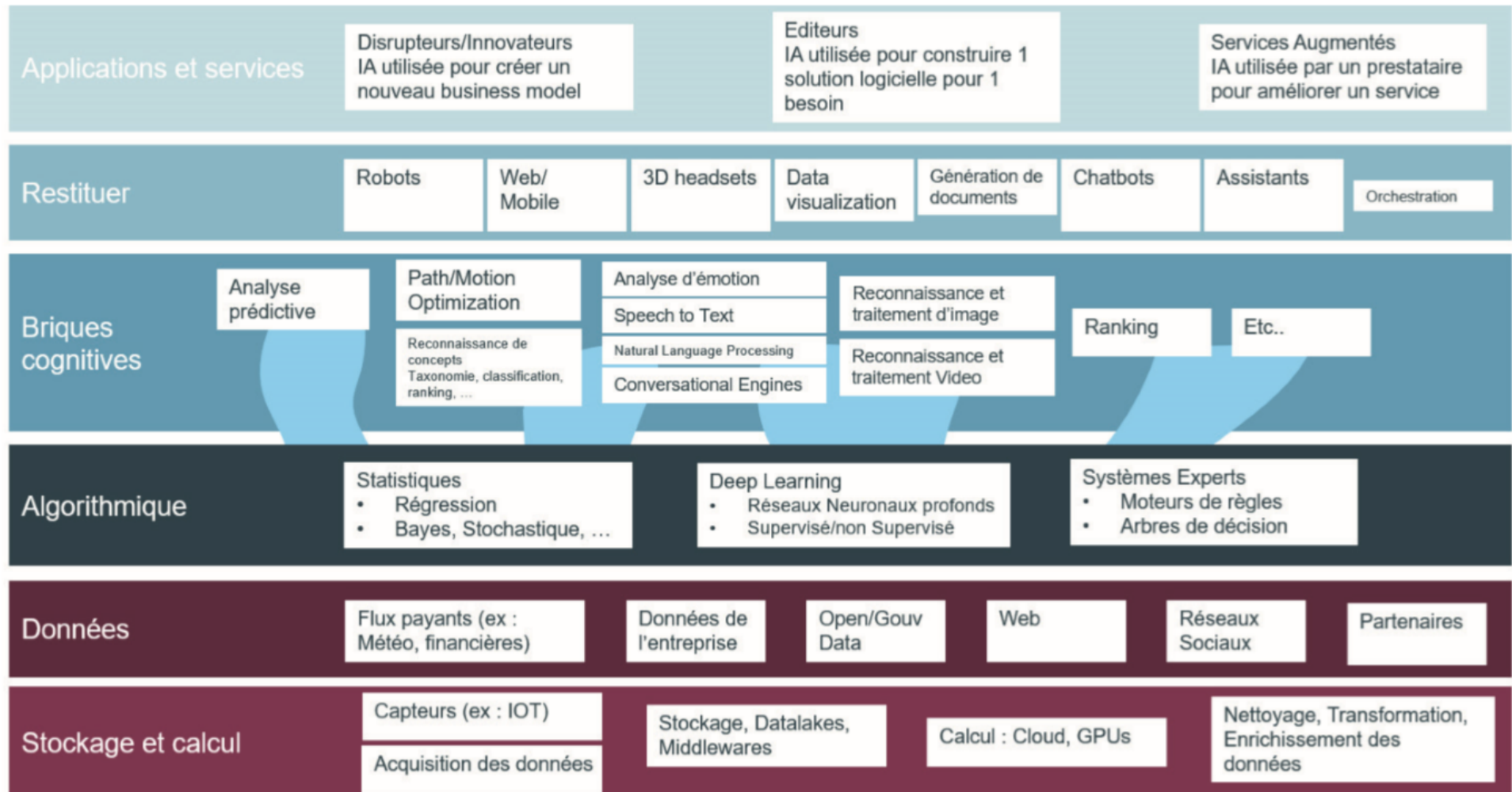
# RESEAUX DE NEURONES (PROFONDS)

---

AI

- ...
- Machine Learning (apprentissage machine)
  - Random Forest
  - SVM
  - Baysien
  - ...
  - Neural Network
  - Deep Learning (dans ce cours voir ca comme un « outils »)
  - Apprentissage par renforcement

# IA / ML / DL

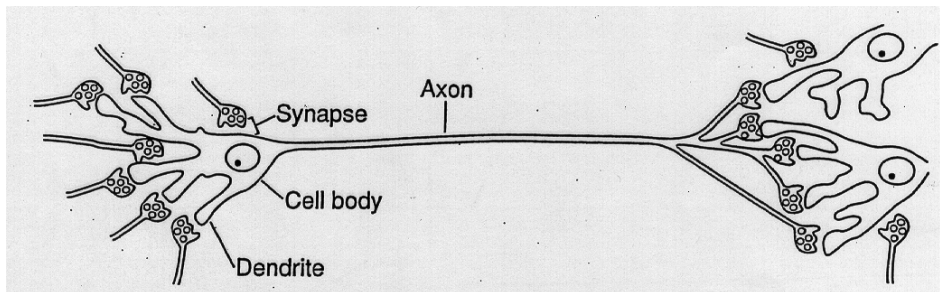




# Réseau de neurones artificiel (RNA)

## un modèle de calcul inspiré du cerveau humain

- Cerveau humain :
  - 10 milliards de neurones
  - 60 milliards de connexions (synapses)
  - Un synapse peut être inhibant ou excitant.
- RNA :
  - Un nombre fini de processeurs élémentaires (neurones).
  - Liens pondérés passant un signal d'un neurone vers d'autres.
  - Plusieurs signaux d'entrée par neurone

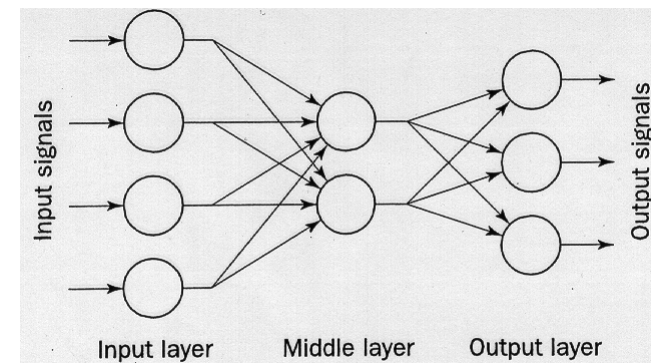


### Cerveau

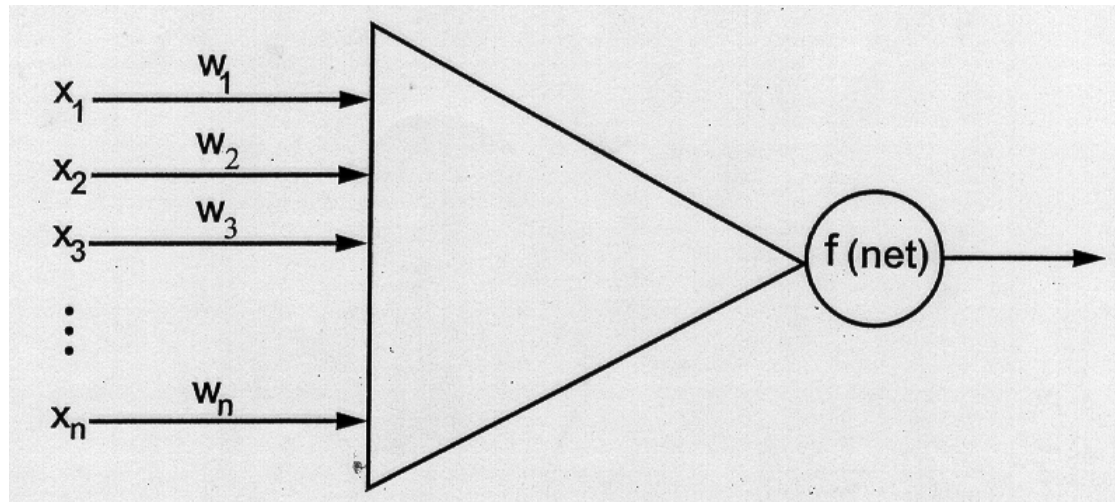
cellule (soma)  
dendrites  
synapses  
axon

### RNA

neurone  
entrées  
poids  
sortie



# Introduction générale aux réseaux de neurones

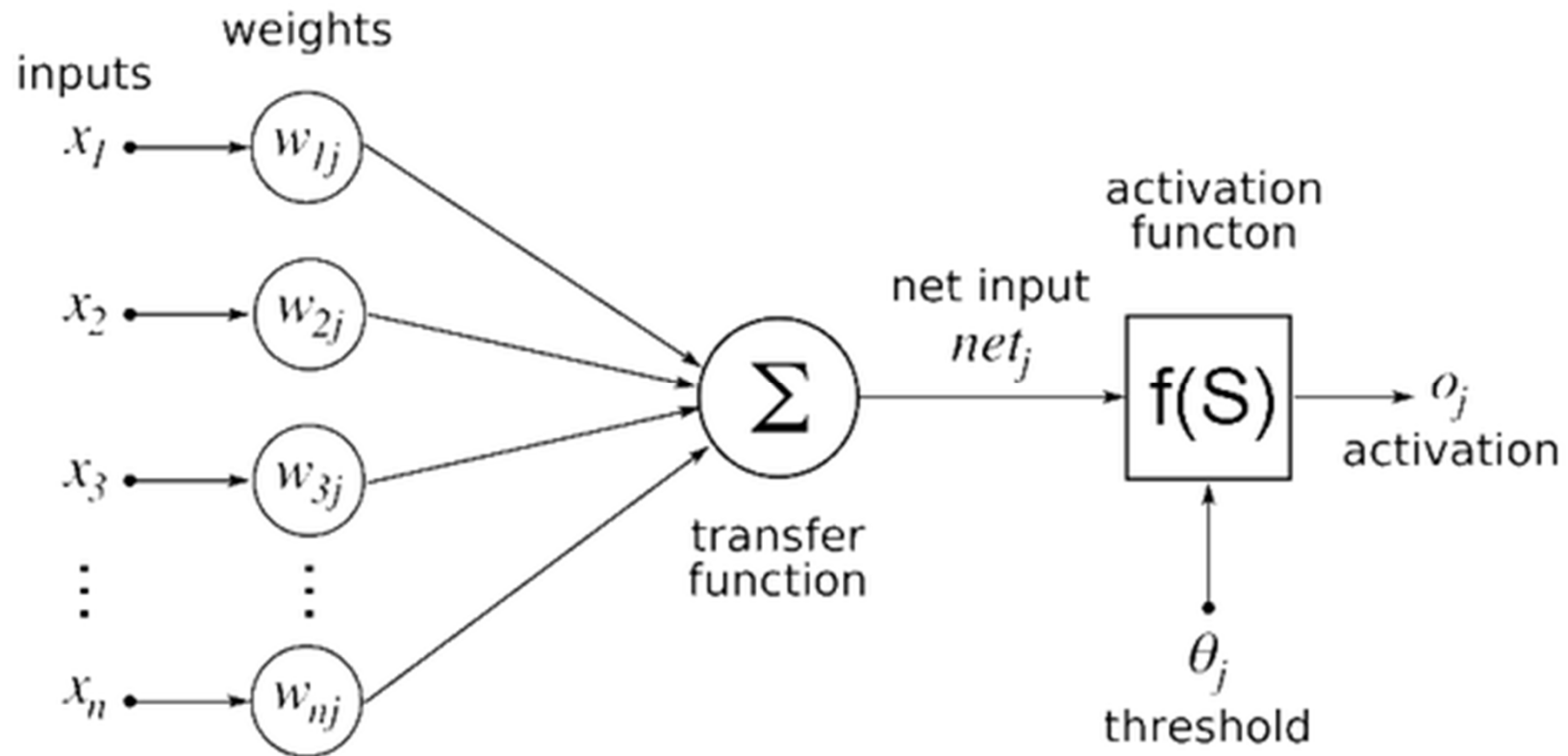


[McCulloch-Pitts, 1943]

- Un type de neurone simple :  $nD \rightarrow 1D$  avec  $f=\text{sign}$

- $\text{net} = \sum w_i x_i$   $x$  = données d'entrée,  $w$ =les poids
- $f(\text{net}) = +1$  si  $\text{net} \geq 0$ ,  $-1$  sinon.  $f$  = Fonction d'activation du neurone
- C.à-d. ici : 1 neurone =  $\text{sign}(\sum w_i x_i)$

# Introduction générale aux réseaux de neurones

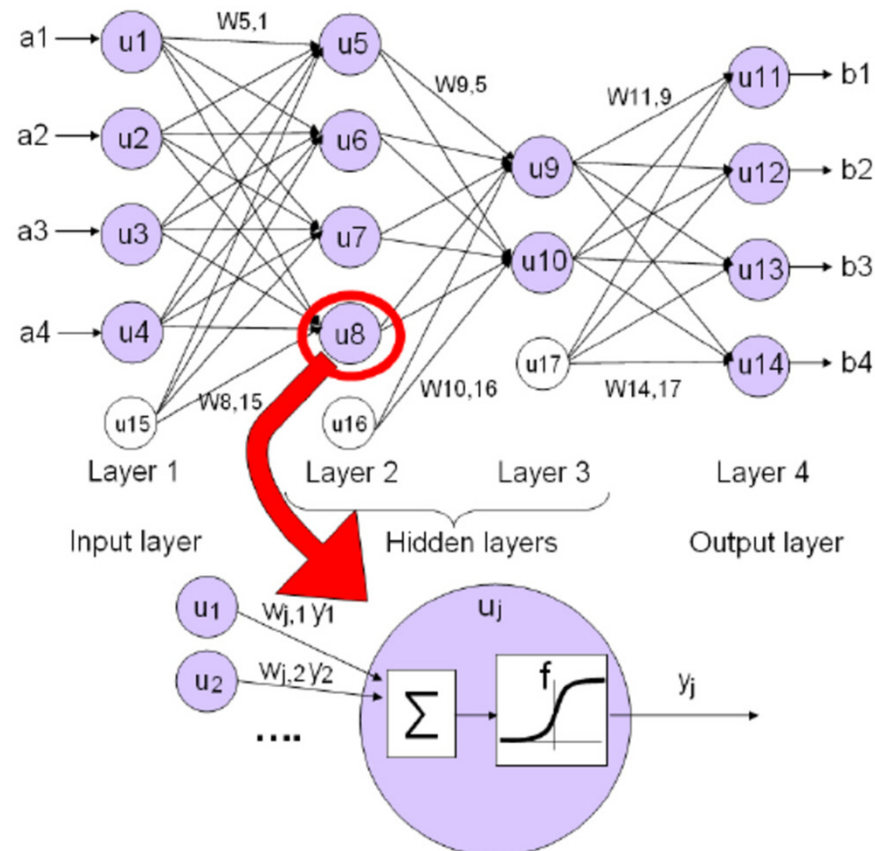


Apprentissage consiste à trouver les poids  $w_{ij}$  par optimisation

# Introduction générale aux réseaux de neurones

nD en entrée  $\rightarrow$  mD en sortie

Les poids  $w_{ij}$  forment une matrice A : sortie = A.entrée + B



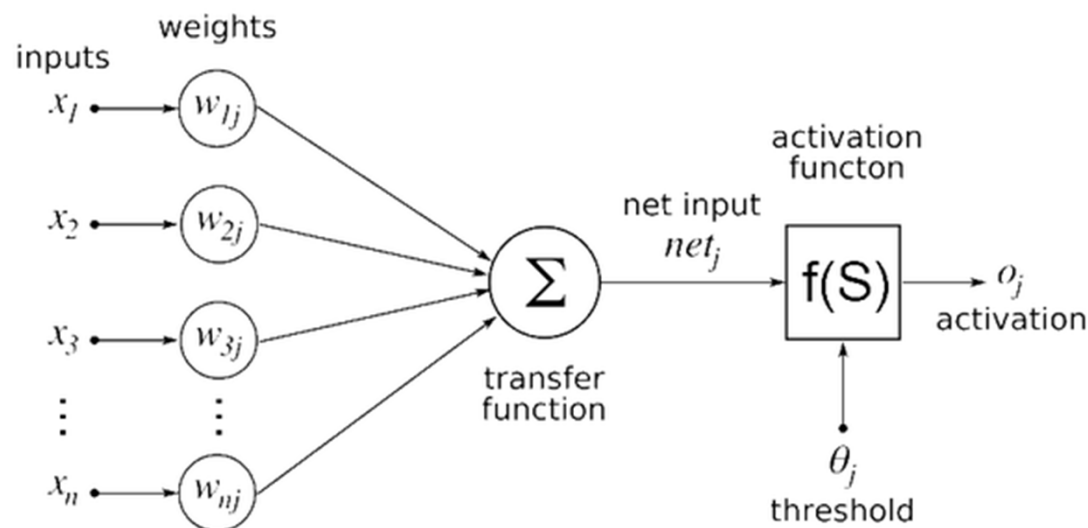
# Introduction générale aux réseaux de neurones

Apprentissage consiste à trouver les poids  $w_{ij}$  par optimisation

- Base de connaissance, un jeu d'apprentissage
- une série de couple (entrée, sortie) donc de

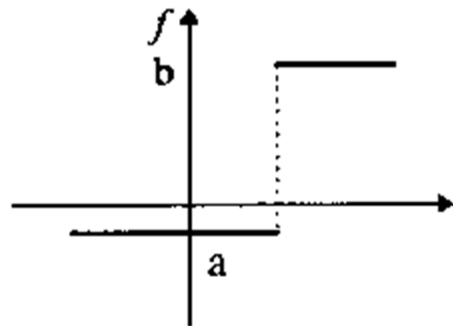
Entrée= $(x_0, \dots, x_i, \dots, x_j)$ , sortie= $(y_0, \dots, y_i, \dots, y_j)$

- Intuition : initialiser  $w_{ij}$  au hasard, puis descente de gradient ( $\sim$ )

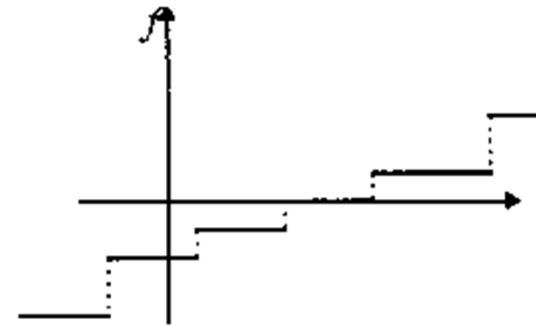


# Introduction générale aux réseaux de neurones

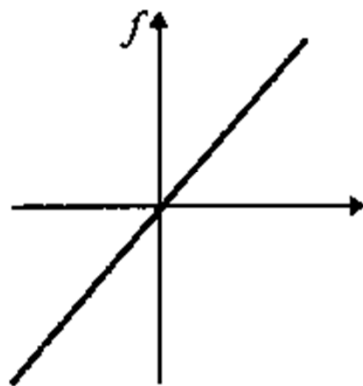
Les fonctions d'activation  $f$



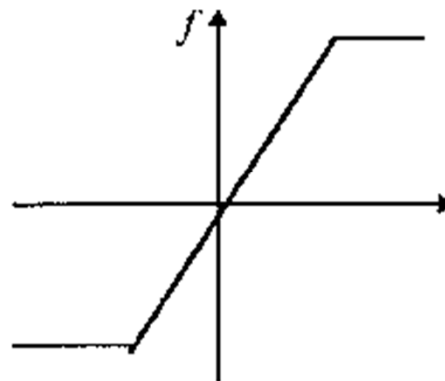
Fonction à seuil



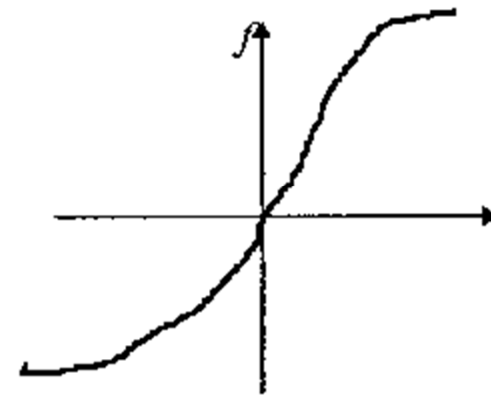
Fonction à valeurs discrètes



Linéaire



Saturation



Sigmoïde

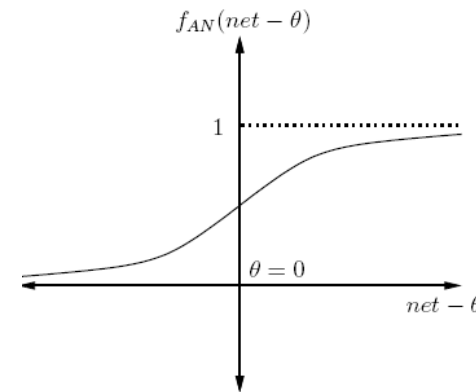


# Introduction générale aux réseaux de neurones

La fonction sigmoïde

$$f_{AN}(\tilde{net}) \in (0, 1).$$

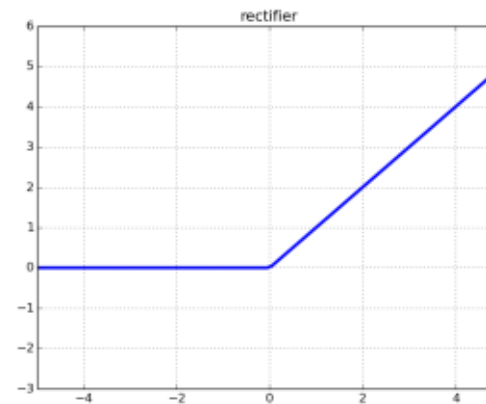
$$f_{AN}(net - \theta) = \frac{1}{1 + e^{-\lambda(net - \theta)}}$$



(d) Sigmoid function

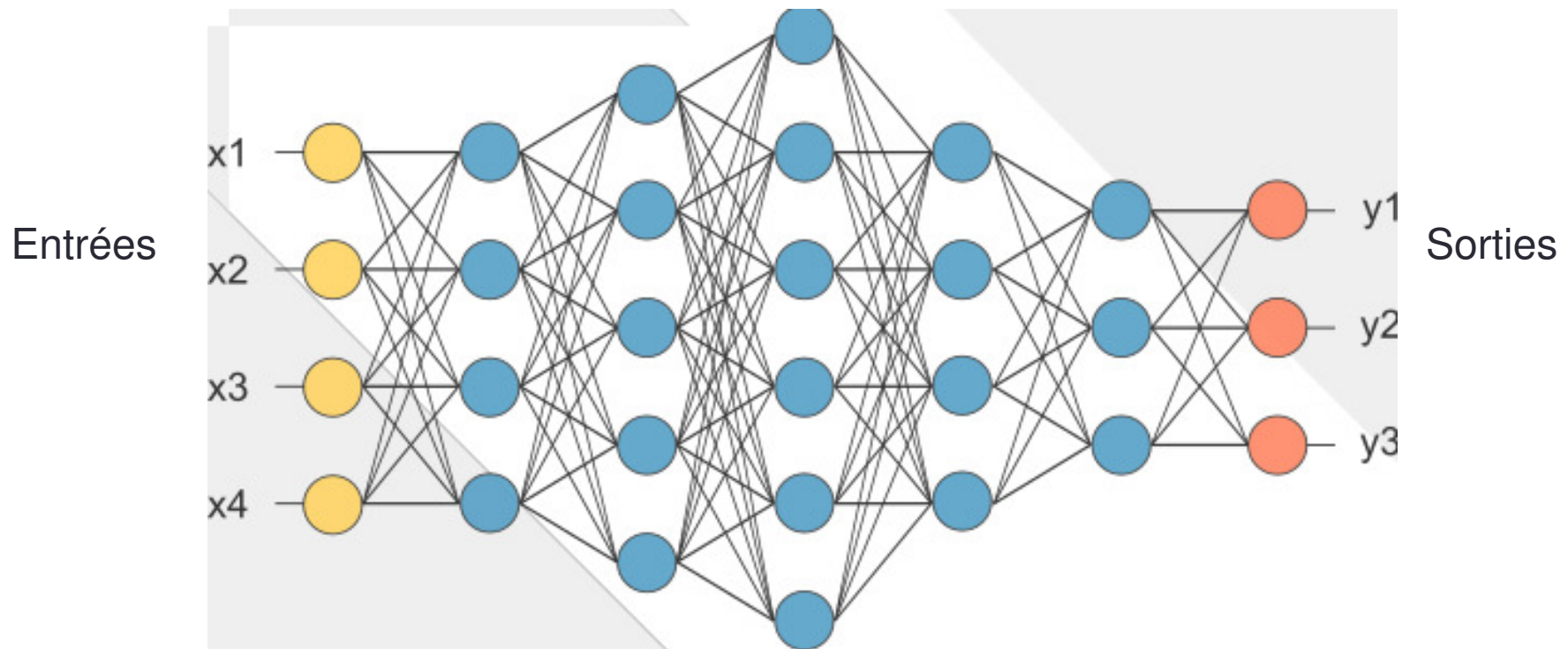
RELU (Rectified Linear)

if  $x < 0$  then return 0  
else return  $x$



# Réseau de neurones profonds

- Chaque cercle est un neurone  $f(A.X+B)$ 
  - En bleu les couches cachées



# La fonction d'erreur (LOSS)

- Entraînement ou apprentissage
  - = une optimisation pour trouver les bons poids  $W$  qui maximisent les résultats sur une base de connaissance (input  $X$ , output  $Y$ )
- La fonction d'erreur que l'on cherche à minimiser
  - Plusieurs type possible suivant le problème

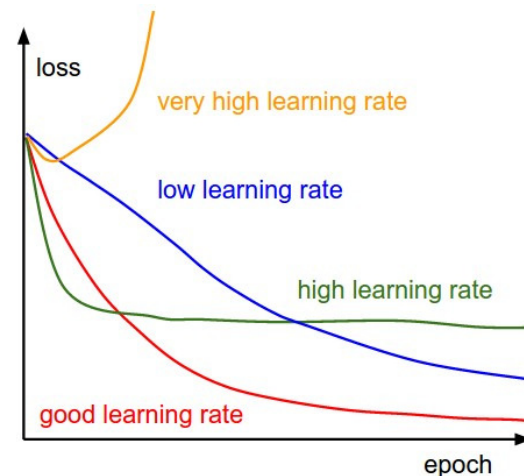
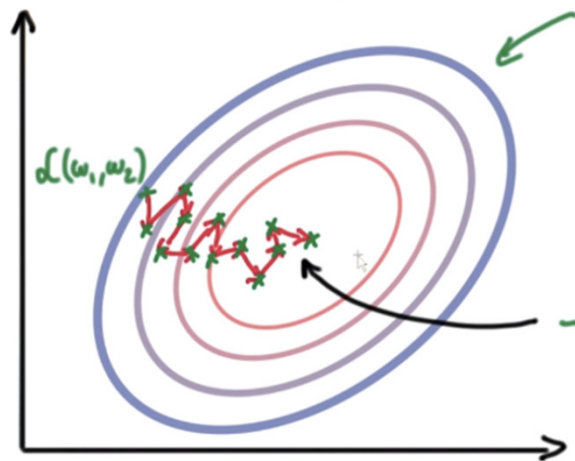
$$L1LossFunction = \sum_{i=1}^n |y_{true} - y_{predicted}|$$

$$L2LossFunction = \sum_{i=1}^n (y_{true} - y_{predicted})^2$$

- Généralement L2-Loss est préférée, mais si vous avez des *outliers* (valeurs aberrantes) dans vos données la L1 peut être meilleurs

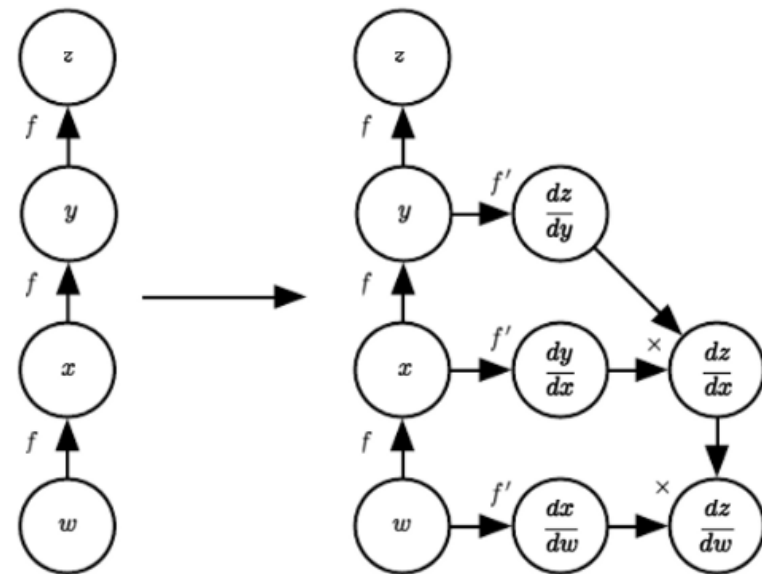
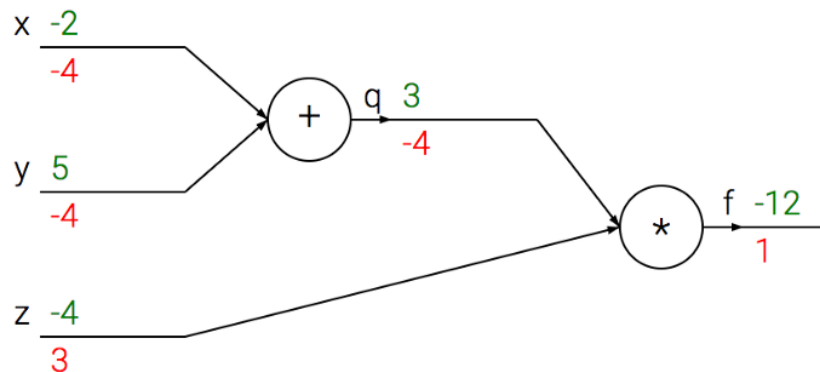
# Réseau de neurones : entraînement

- Stochastic Gradient Descent (SGD)
  - Stochastic car la descente de gradient est calculée sur un sous-échantillonnage des données (epoch)
  - Learning rate = progression ou pas d'apprentissage
  - Moment = comme stochastic le changement de direction se fait avec une inertie
- Besoin des GPU pour la puissance de calcul
  - Par exemple, ImageNET 15 millions d'images avec 20 000 labels



# Entraînement : backpropagation

- Dans un réseau le calcul « classique » du gradient peut être long, très long à calculer et demander
- Back propagation permet de faire le calcul avec une unique passe d'aller retour



# Réseau de neurones : du code

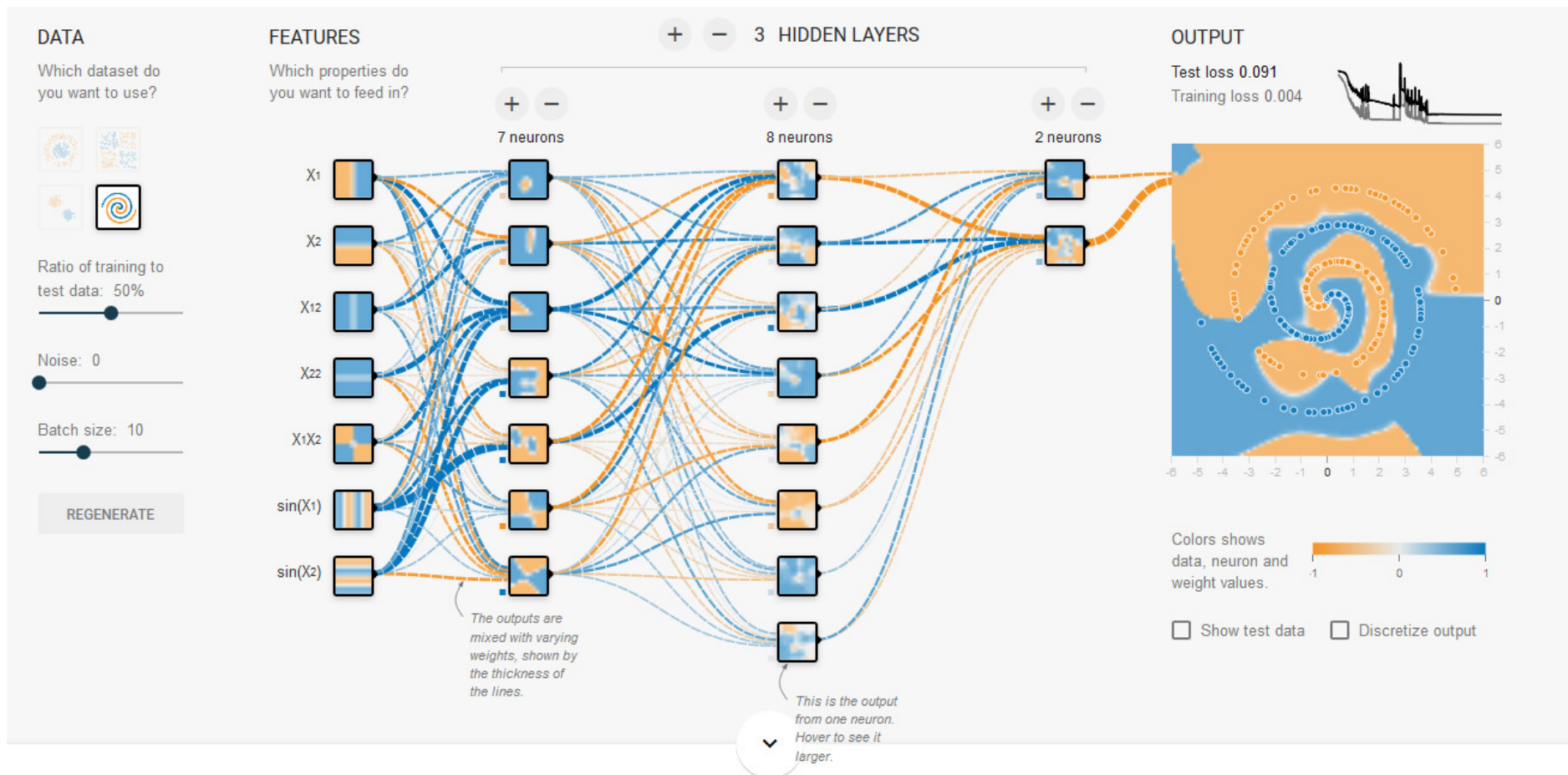
- Plateformes en python
  - TensorFlow (Google)
  - Theano
  - PyTorch (Yann Lecun, maintenant Facebook)
  - CNTK Microsoft Cognitive Toolkit
  - ...
- Accélération GPU
- Des exemples de code
  - Keras = une sur-couche facilitant le codage des réseaux
    - Cible : TensorFlow, Theano, CNTK
  - PyTorch
  - TensorFlow 2.0 comportera des fonctions de haut niveau comme Keras/PyTorch



# Réseau de neurones

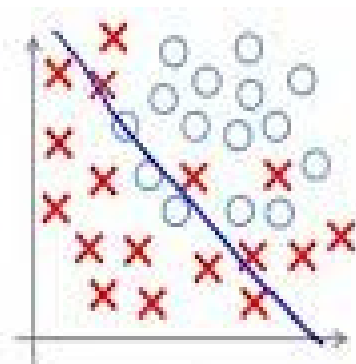
- Exemple de classification d'un nuage de points

<http://playground.tensorflow.org>



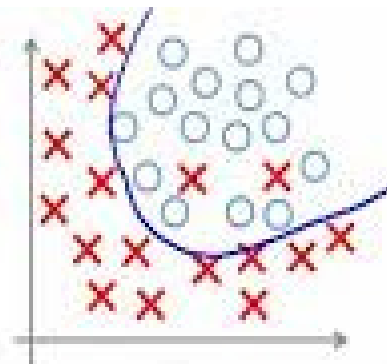
# Surapprentissage / Overfitting

- Surapprentissage / Overfitting : c'est quoi ?
  - Quand un modèle apprend trop de détail/bruit sur les données avec pour conséquence d'être mauvais sur la généralisation (ie. Quand il verra des nouvelles données jamais vu avant)
  - Comme un étudiant qui « apprend par cœur » sans « comprendre »

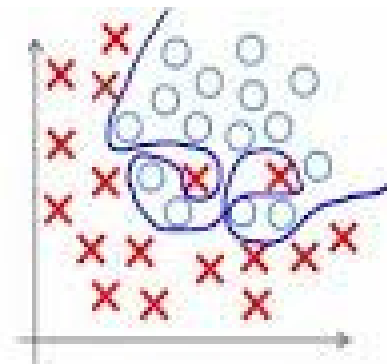


Under-fitting

(too simple to explain the variance)



Appropriate-fitting



Over-fitting

(overfitting – too good to be true)

# Régularisation

Les techniques de **Regularisation** sont les techniques utilisées pour résoudre le surapprentissage (overfitting) en apprentissage machine

Par exemple

- L1 Regularization or Lasso Regularization

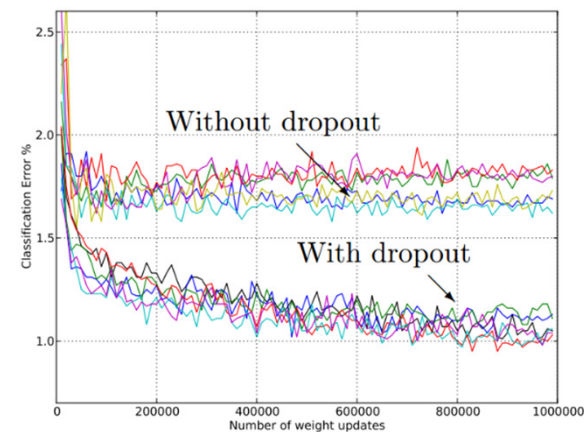
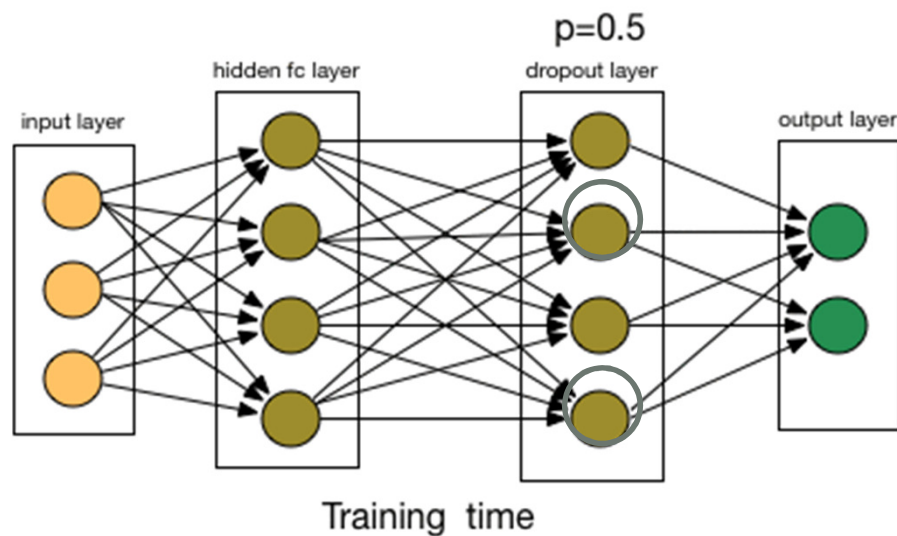
$$\text{Min} \left( \sum_{i=1}^n (y_i - w_i x_i)^2 + p \sum_{i=1}^n |w_i| \right)$$

- L2 Regularization or Ridge Regularization

$$\text{Min} \left( \sum_{i=1}^n (y_i - w_i x_i)^2 + p \sum_{i=1}^n (w_i)^2 \right)$$

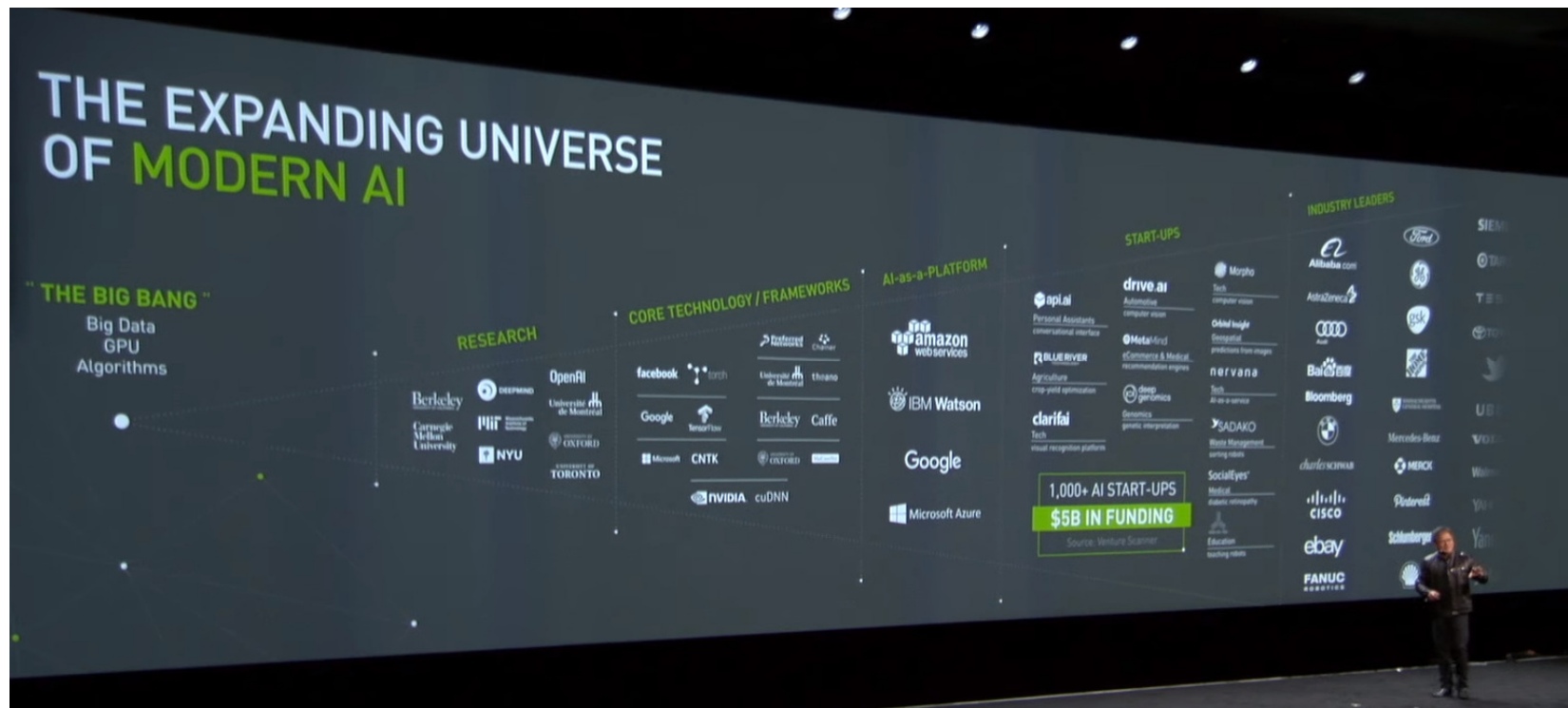
# Drop Out

- Dropout aide pour éviter le surapprentissage (overfitting)
  - force le réseau à apprendre de manière plus robuste
  - technique de [régularisation](#)
- Dropout annule temporairement aléatoirement certain neurone avec une proba  $p$



# Explosion du deep learning

- Des données
- Du GPU
- Un algo de back-propagation rapide et facilement codable sur GPU
- Des algo, des algo, des algo ...



# DEEP LEARNING POUR LES BASQUES DE L'IMAGES ET DE LA VISION

## ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.





# Deep Learning a permis des avancées fortes en vision par ordinateur

Image Classification

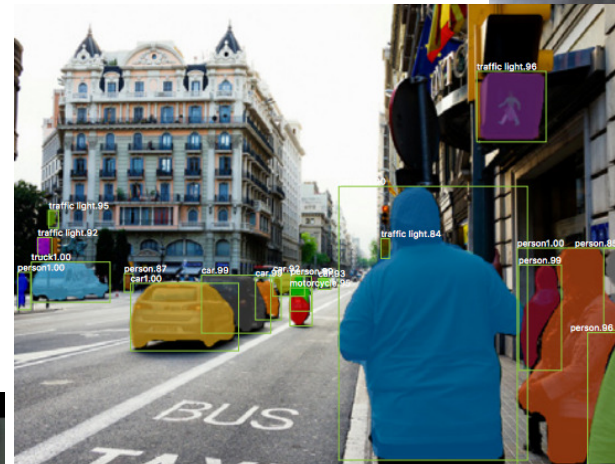
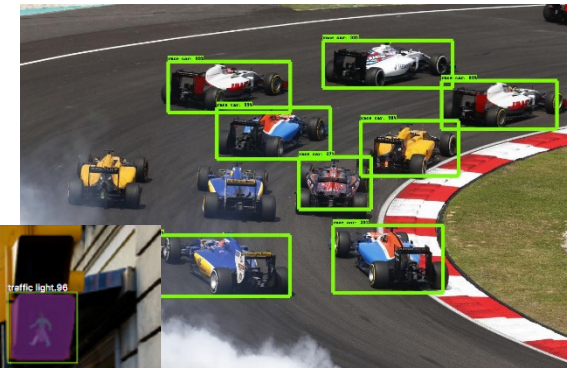
Object Detection

Object Tracking

Semantic Segmentation

Instance Segmentation

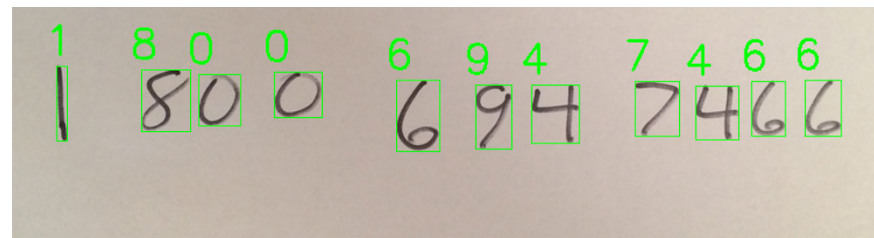
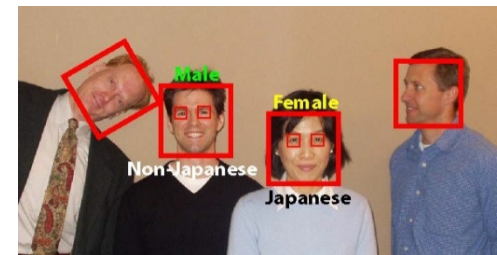
...



# Deep Learning et Reconnaissance

## Reconnaissance

- d'une famille d'objets : « c'est un chat »
- d'un objet précis : « c'est Paul », « c'est un 8 »
- d'une expression/style : « le visage sourit »
- ...



# Vision de haut niveau : reconnaissance



Qu'est-ce que vous voyez dans cette image ?

Tâche extrêmement difficile :  
le tigre doit être reconnu sous tous les angles, parfois caché, avec des éclairages différents sur chaque photo.

→ Test de Turing sur l'« intelligence artificielle »

# Filtres

- Détecteur de courbes



Original image



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

\*

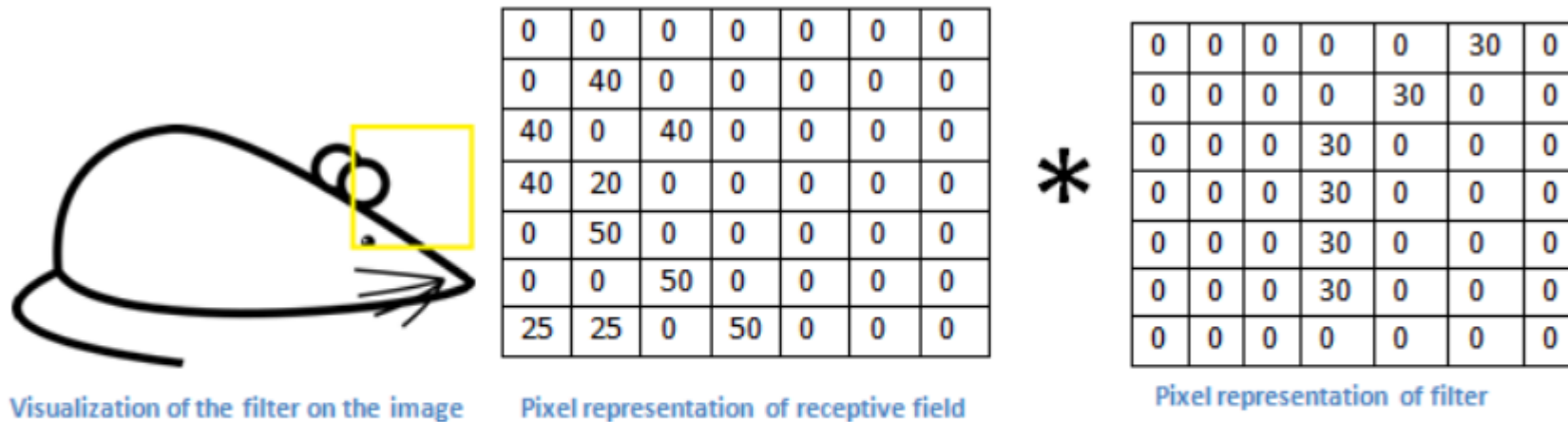
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation =  $(50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600$  (A large number!)

# Filtre : détecteur de courbes

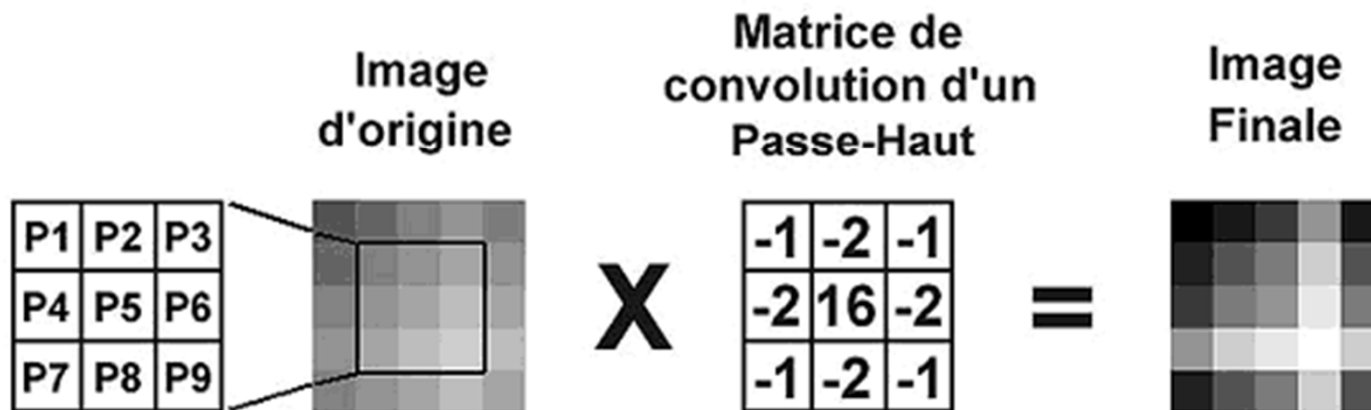
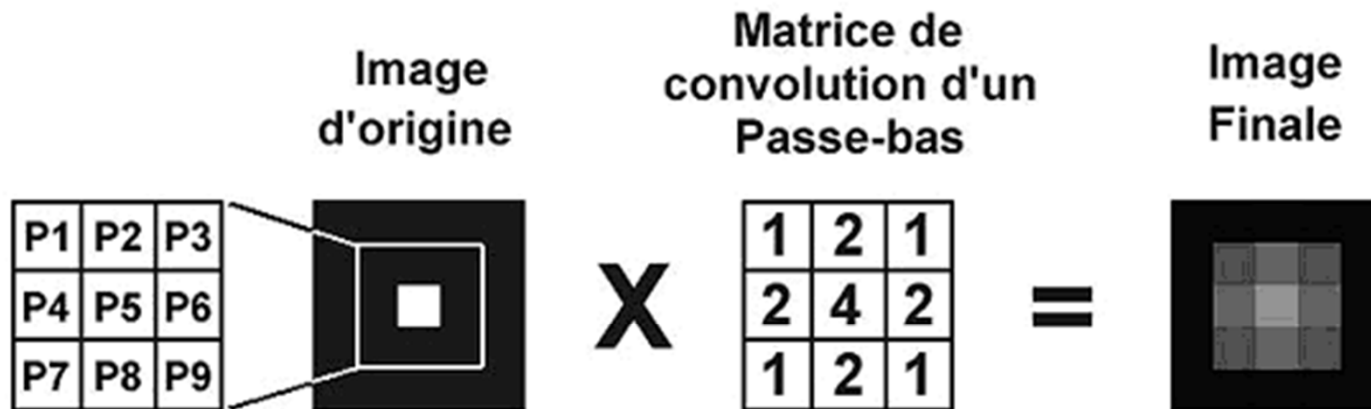
- Le même filtre ailleurs



Multiplication and Summation = 0

- On obtient donc une carte d'activation de ce filtre
  - Feature map

# Traitement d'images : filtres



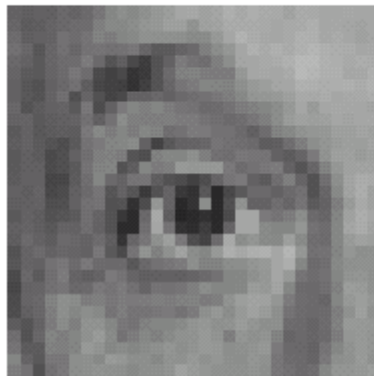


# Filtre Gaussien : Blurring

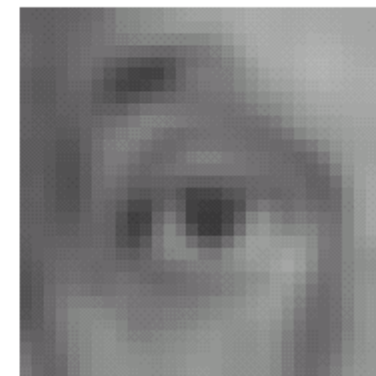
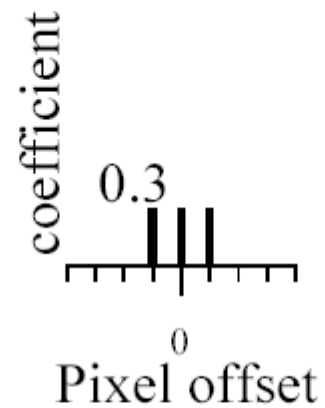
F

$1/9$

1	1	1
1	1	1
1	1	1



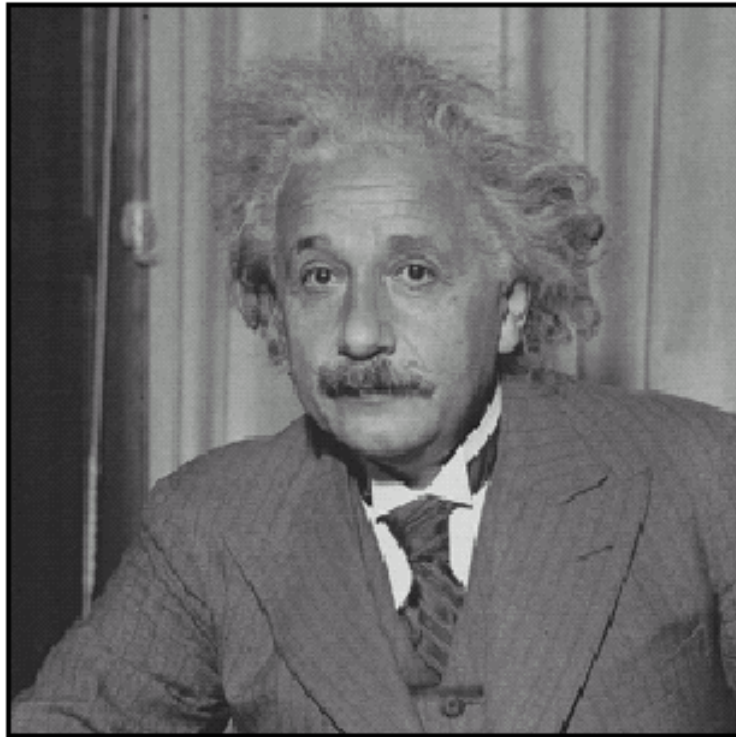
original



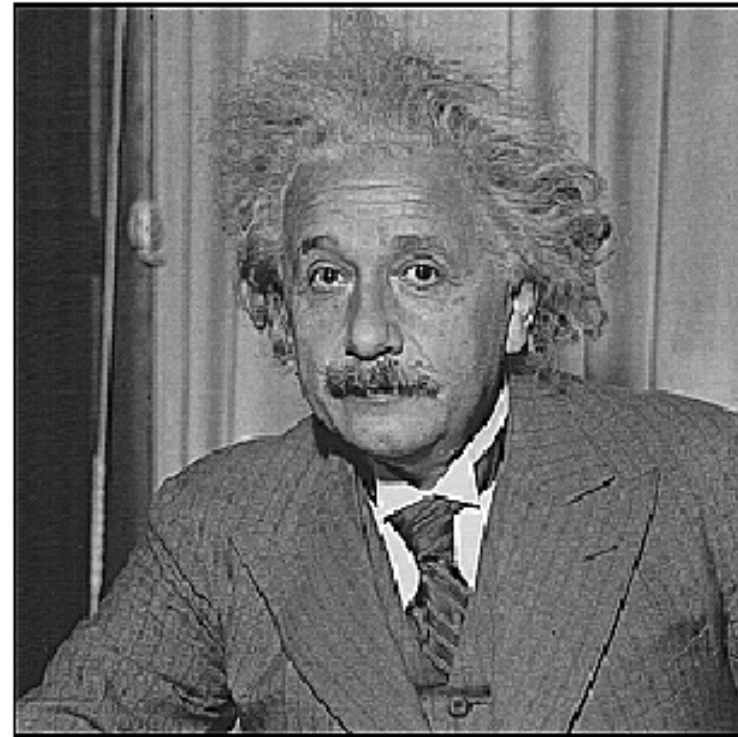
Blurred (filter applied in both dimensions).



# Sharpening



**before**



**after**

# Filtre : détecteur d'arêtes

Filter Builder - Alwaysbusy's Corner

Kernel dimensions

	X	Y
Load Filter	1	1
Save Filter	2	2
	3	3
	4	4
	5	5

X Kernel

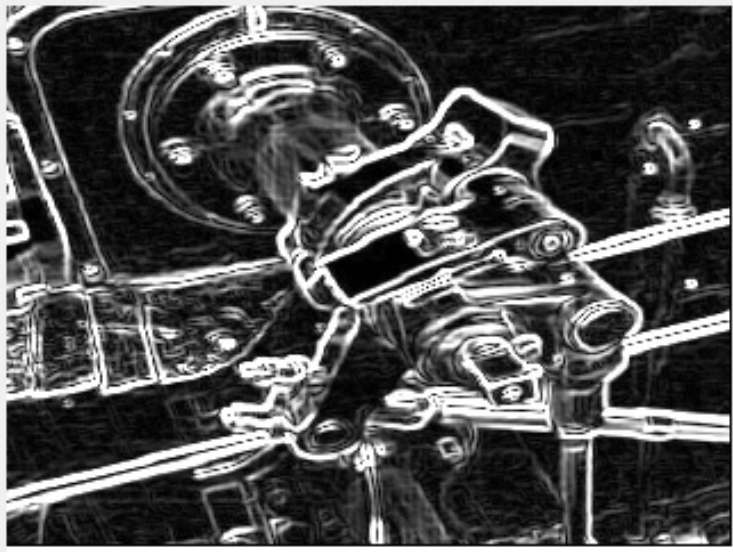
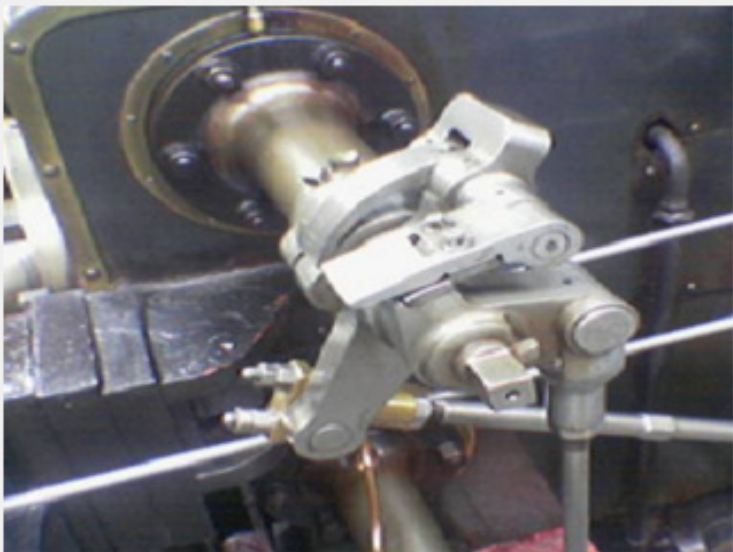
-1	-1	1
-2	0	2
-1	0	1
-2	1	2

Y Kernel

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

Load picture

Apply Filter



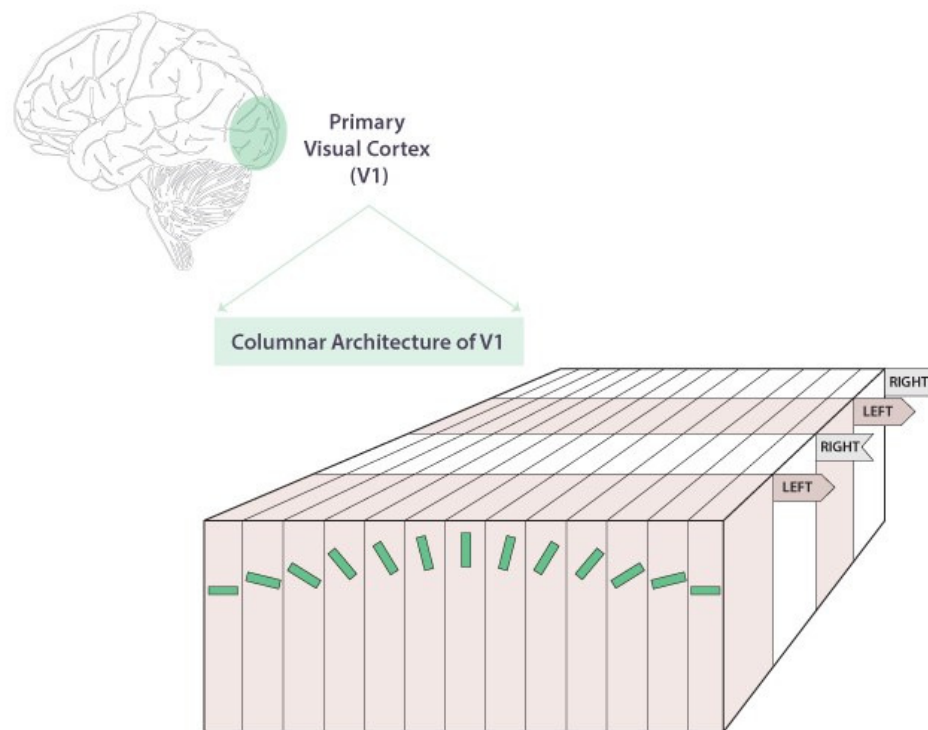
# Les Difficultés

Quel enchainement/combinaison de filtres permettront de bien classifier une image ?

- Avant (le deep learning)
  - un humain (ingénieur/chercheur) proposait des filtres selon son savoir-faire pour produire des descripteurs (quelques centaines de valeurs)
  - Puis classification (SVM, Random Forest, etc.)
- Maintenant, réseau de convolution **ConvNeuralNET (CNN)**
  - Optimise des poids dans plusieurs filtres pour produire les descripteurs (*feature maps*)
  - Puis les dernières couches complètement connectées font la classification

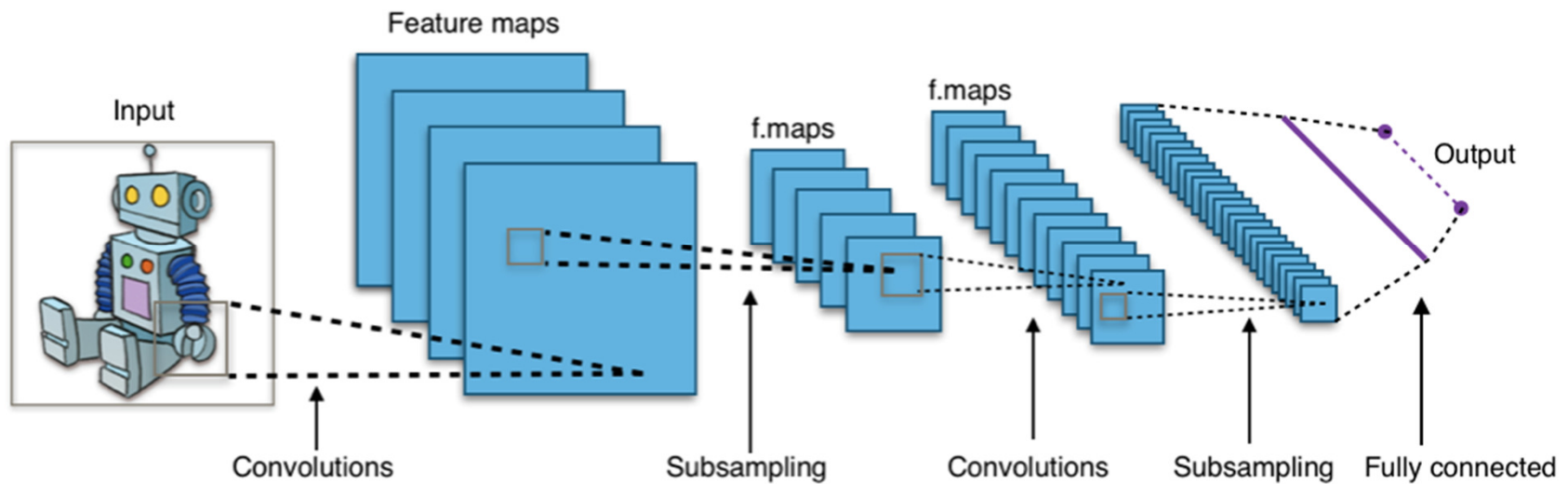
# Expérience de Huble

- Huble and Wiesel en 1962 montre que dans le cortex visuel les neurones sont organisés pour répondre à des patrons précis : lignes avec différentes inclinaisons, etc.



# Réseau de convolution ConvNET

- Reconnaître un objet avec un réseau de convolution

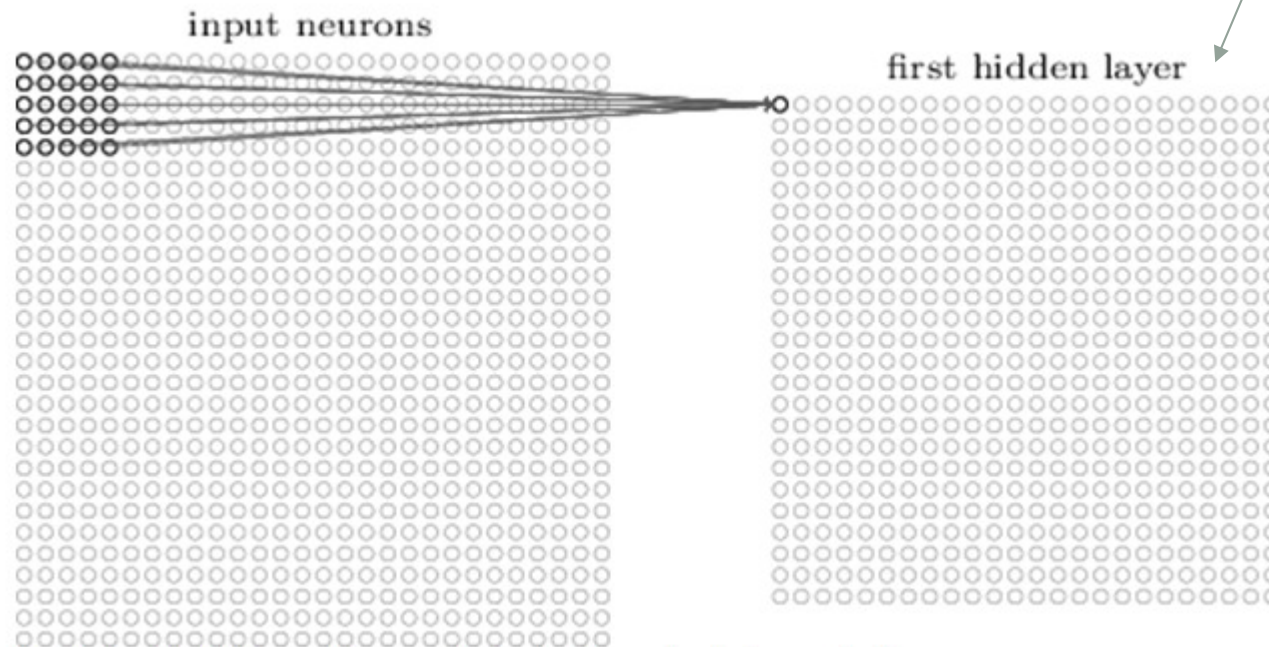




# Une convolution

Activation map  
Feature map  
Image de caractéristiques

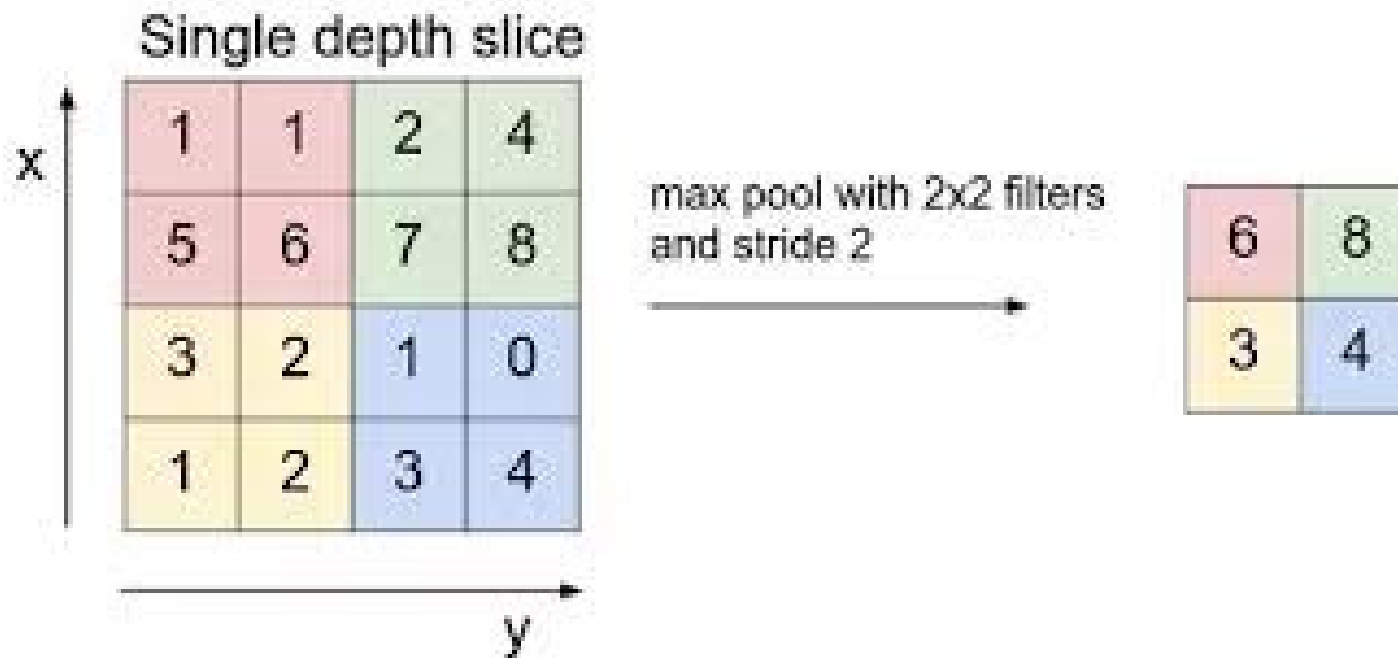
- Input :  $32 \times 32 \times 1 \rightarrow \text{Conv}(5,5) \rightarrow 28 \times 28 \times 1$
- Input :  $32 \times 32 \times 3 \rightarrow \text{Conv}(5,5,3) \rightarrow 28 \times 28 \times 1$ 
  - La convolution se fait sur les 3 channels



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

# Max Pooling

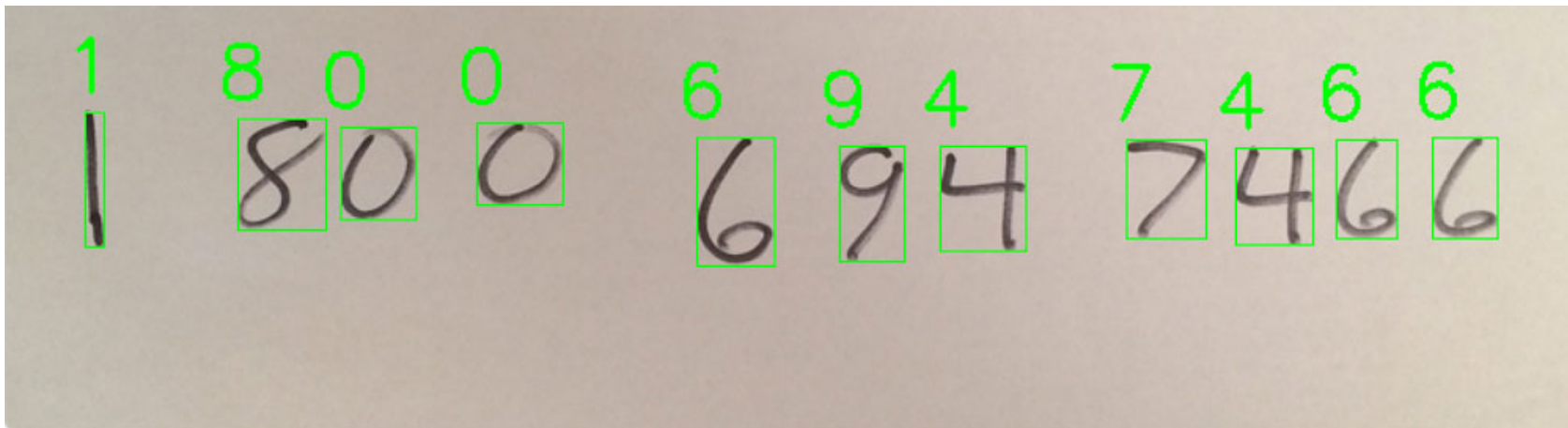
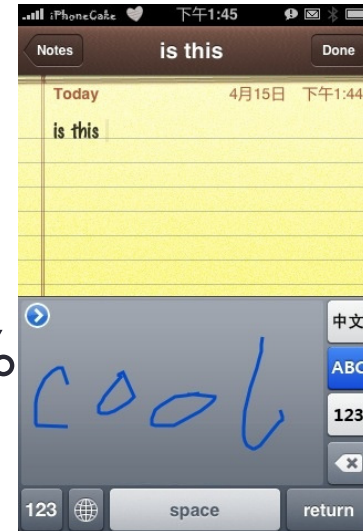
- Reduction de dimensions





# Reconnaissance d'écriture

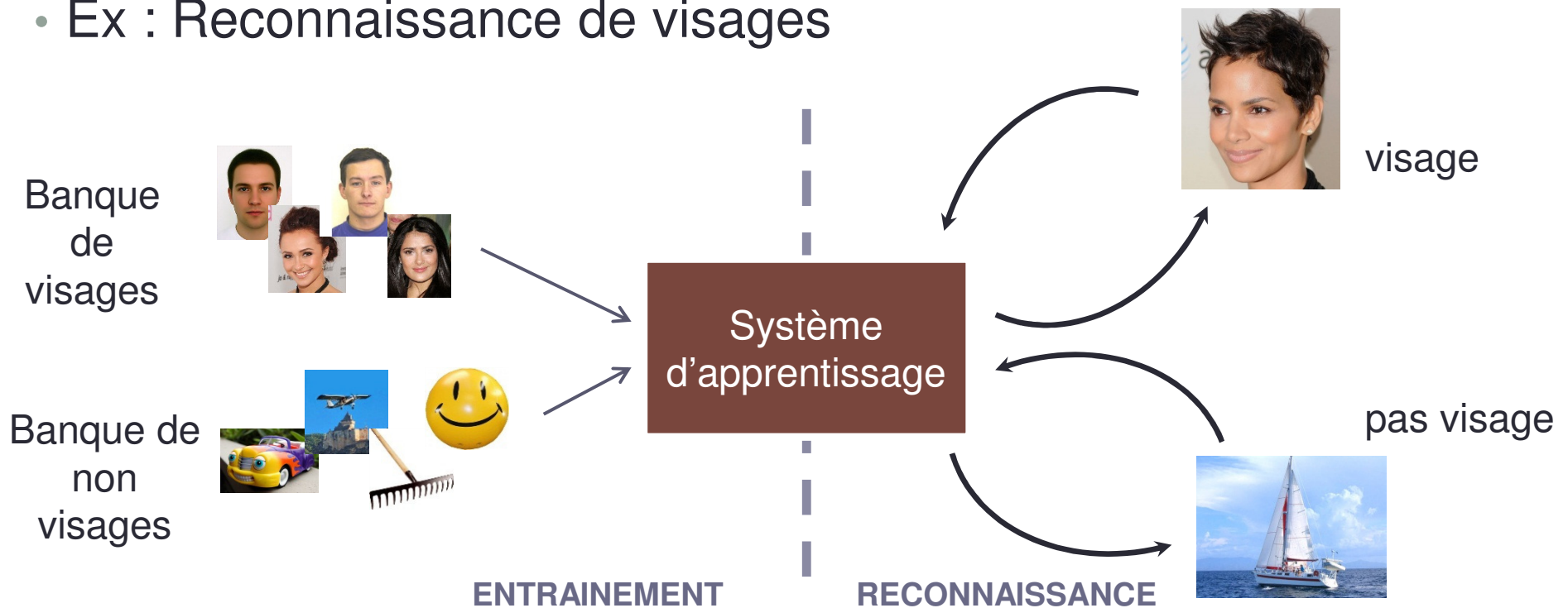
- Reconnaissance d'écriture
  - La Poste : codes postaux sur enveloppe
  - Puis écriture sur tablette
- Avec un ConvNET, taux de bonne reco > 99%



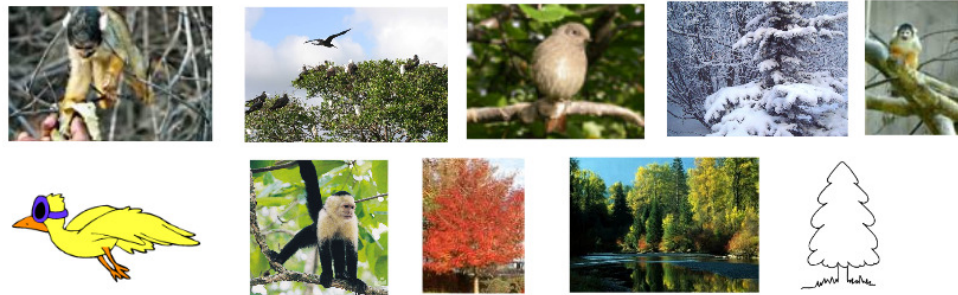
# Reconnaissance de visages

- L'apprentissage automatique
  - A partir d'une banque d'exemple, l'ordinateur apprend à classer différents éléments.

- Ex : Reconnaissance de visages

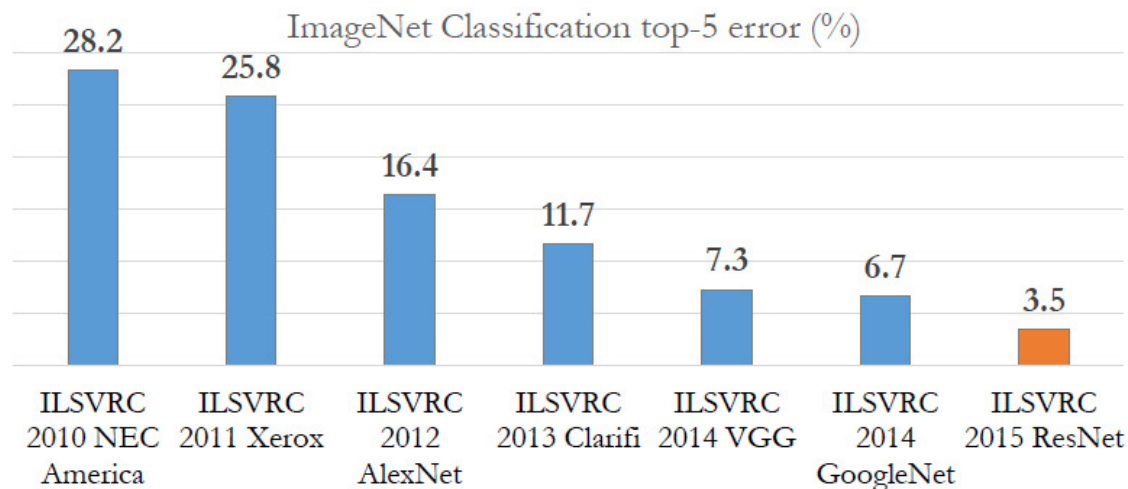


# IMAGENET=corpus d'images



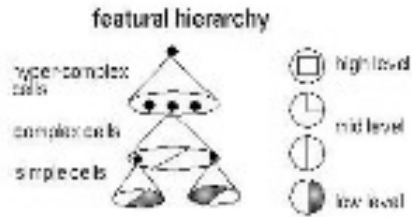
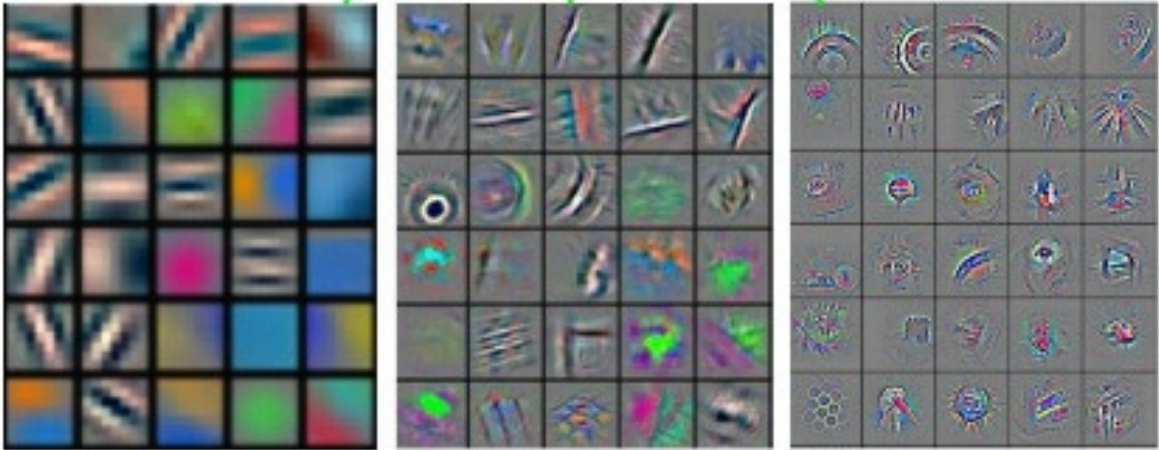
- Classification

- Historique vision humaine : reperer un predateur ou un membre de sa famille rapidement
- Concours IMAGENET → mettre un label sur une image
  - 14 millions d'images avec 20000 labels



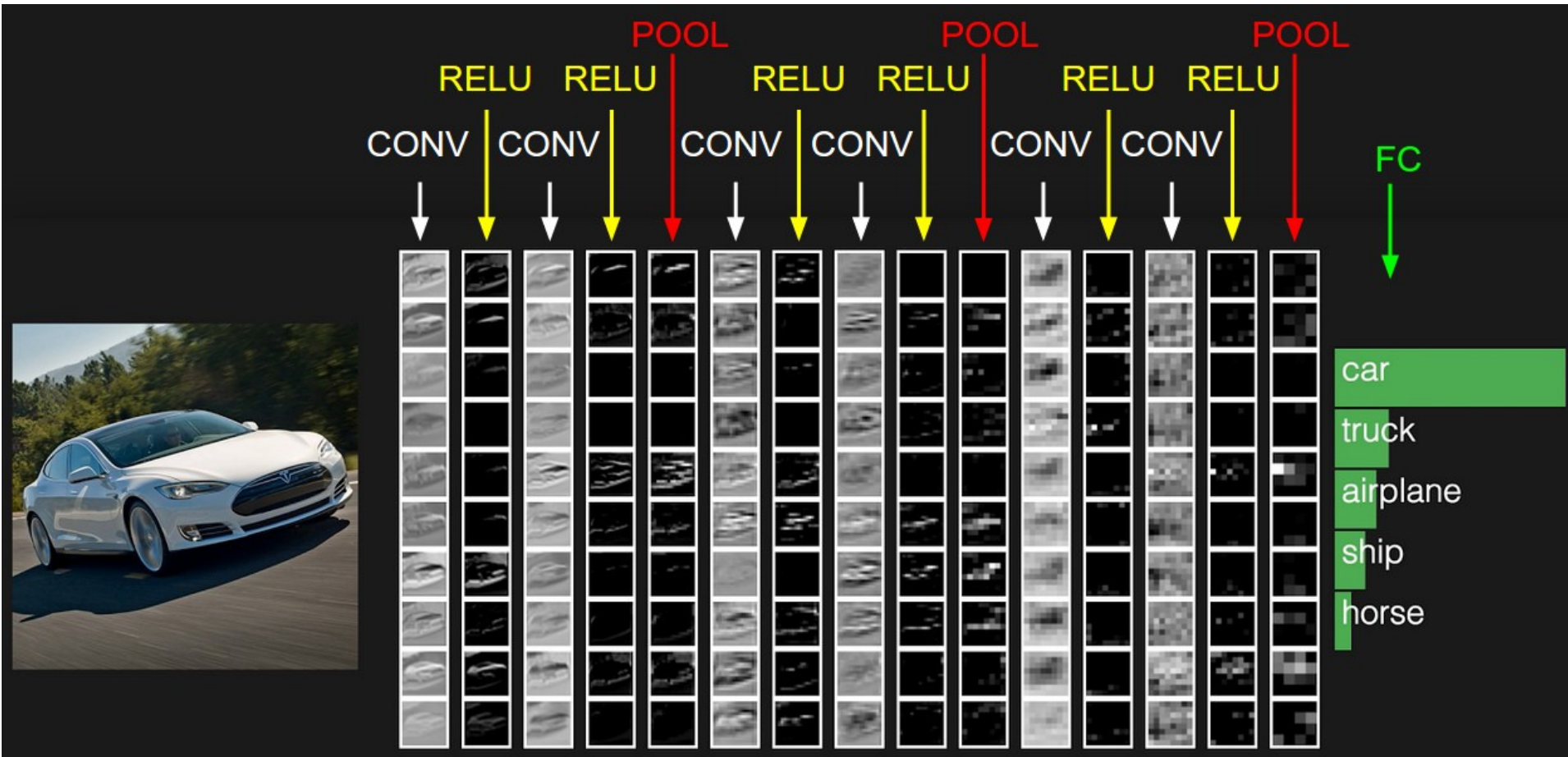
# Réseau de convolution ConvNET

## ConvNet : Interpretation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

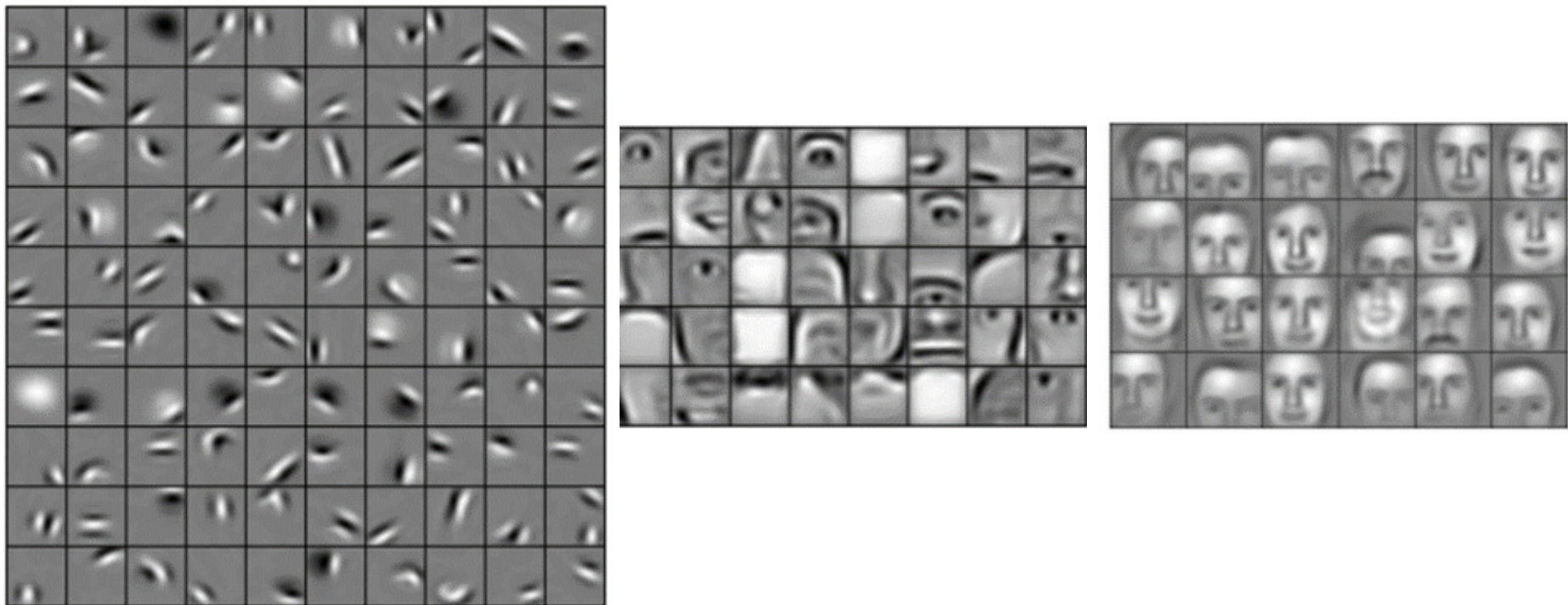
# Réseau de convolution ConvNET





# ConvNET : visage

- Exemple de features (filtres) produit par le réseau dans les couches cachées



Couches du début

Couches profondes



K. Simonyan, A. Zisserman

[Very Deep Convolutional Networks for Large-Scale Image Recognition](#)

arXiv technical report, 2014

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

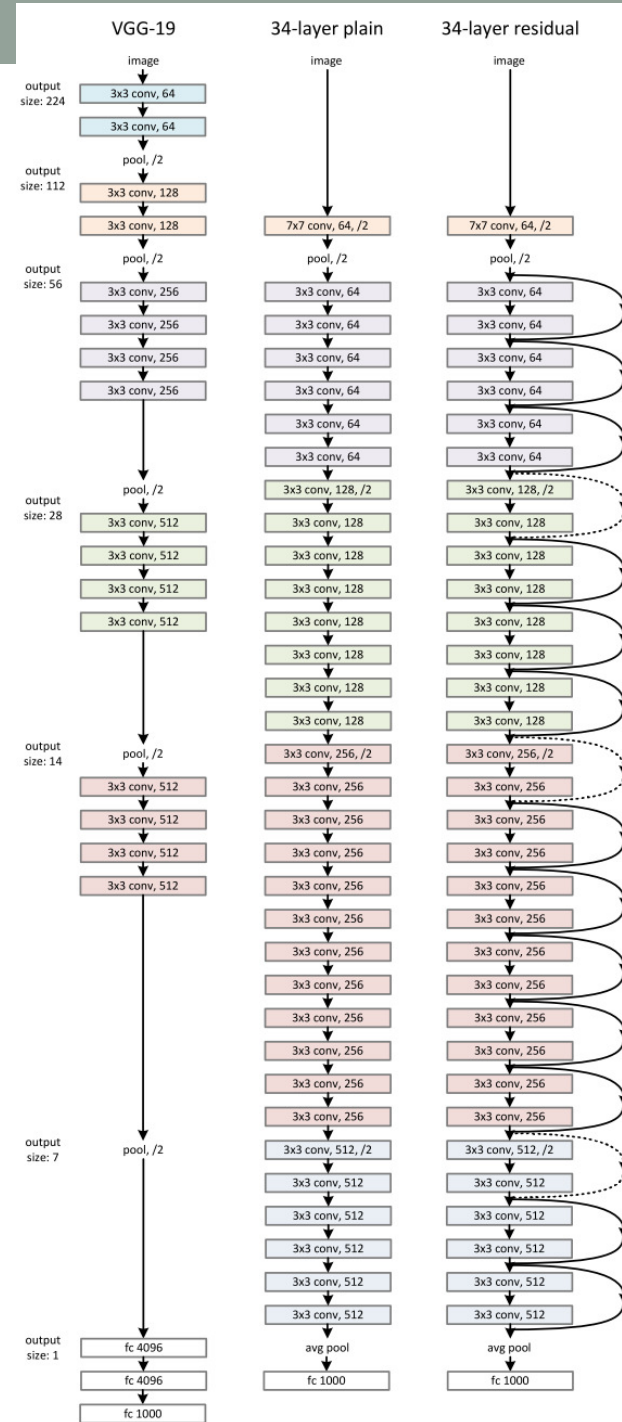


# VGG19 : voir TP pour afficher les couches

```
0 Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
1 ReLU(inplace)
2 Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
3 ReLU(inplace)
4 MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
5 Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
6 ReLU(inplace)
7 Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
8 ReLU(inplace)
9 MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
10 Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
11 ReLU(inplace)
12 Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
13 ReLU(inplace)
14 Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
15 ReLU(inplace)
16 Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
17 ReLU(inplace)
18 MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
19 Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
20 ReLU(inplace)
21 Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
22 ReLU(inplace)
```

# ResNET-50

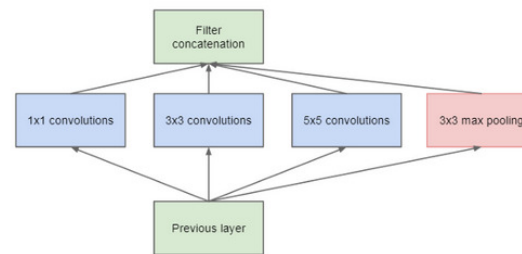
Deep Residual Learning for Image Recognition  
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun  
<https://arxiv.org/abs/1512.03385>



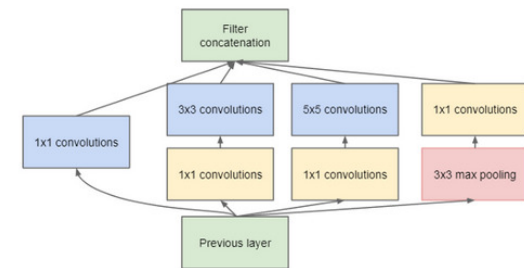
# Réseaux de reconnaissance

Souvent utilisés en pré-traitement comme « traitement d'images »

- VGG
  - VGG16, VGG19
  - Série de Conv, Max-pooling, and activation, puis fully-connected (FC)
- ResNet50
  - Grande profondeur
- Inception v3
  - Large
- Xception
  - extreme inception
- MobileNet
  - Optimisé pour mobile






(a) Inception module, naïve version



(b) Inception module with dimension reductions

- Dans KERAS / PyTorch ces modèles sont pré-entraînés (ouf)

# Les données

	VGGNet	DeepVideo	GNMT
Used For	Identifying Image Category	Identifying Video Category	Translation
Input	Image 	Video 	English Text 
Output	1000 Categories	47 Categories	French Text
Parameters	140M	~100M	380M
Data Size	1.2M Images with assigned Category	1.1M Videos with assigned Category	6M Sentence Pairs, 340M Words
Dataset	ILSVRC-2012	Sports-1M	WMT'14

Number of parameters (in millions), for popular neural networks.

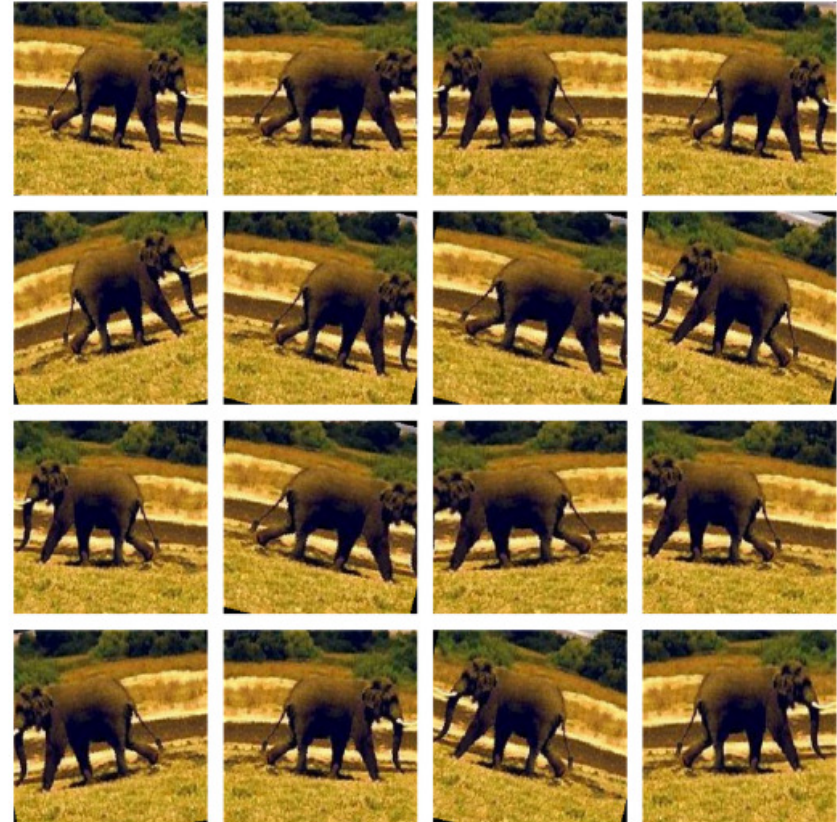
# Problème : manque de données pour DL

- Taille de certaines base de données
  - MNIST : nombres manuscrits, 70000 images
  - EMNIST : lettres manuscrites
    - Équilibré 134000 images
    - Sinon 814000 images
  - ImageNET : 14 millions d'images avec 20000 labels
  - CelebFaces (CelebA) : 202,599 images de visages de 10,177 célébrités
- Pour de nombreux autres problèmes
  - Cohn Kanade : 486 vidéo de 97 personnes exprimant une expression
  - ...
- Branches du machine learning qui cherchent à fonctionner avec moins de données
  - DL plus efficaces : GAN, etc.
  - One-shot learning / Few-shot learning

# Augmentation de données

La base

- Flip
- Rotation
- Scale
- Crop
- Translation
- Gaussien Noise

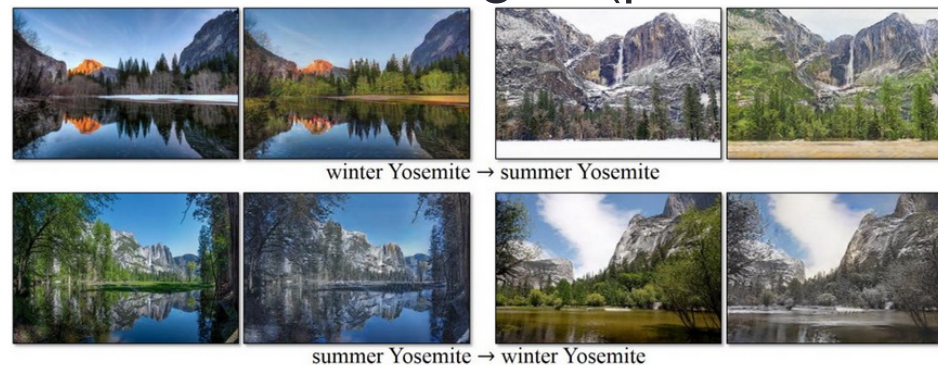




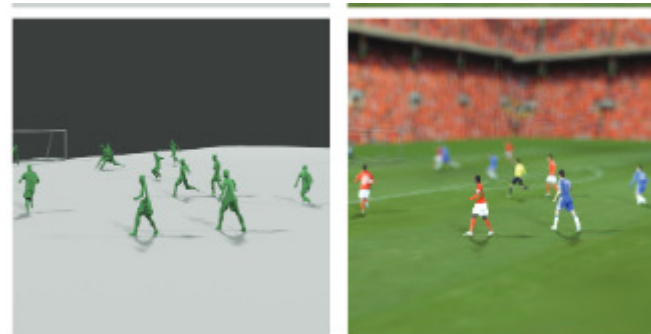
# Augmentation de données

Un peu plus loin

- GAN pour transformer des images (palette de couleur, style, etc.)



- Données issus d'images de synthèses



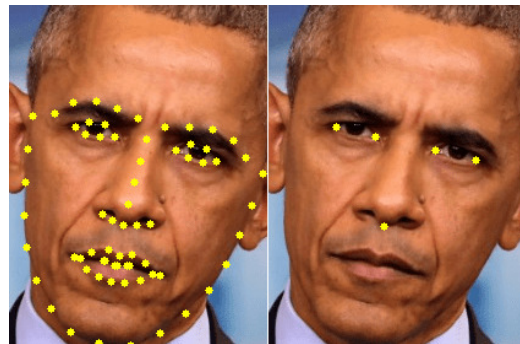
Do We Really Need to Collect Millions of Faces for Effective Face Recognition?  
<https://arxiv.org/pdf/1603.07057.pdf>



# Augmentation de données : aider le réseau avec des données annexes

Par exemple : corpus d'images/vidéo de visages

- DL sur uniquement les images, possible mais ...
  - Possibilité d'en extraire des points du visage par détection : coins de la bouche, nez, yeux
    - DL sur les points caractéristiques + images
- Bien plus efficaces !!!



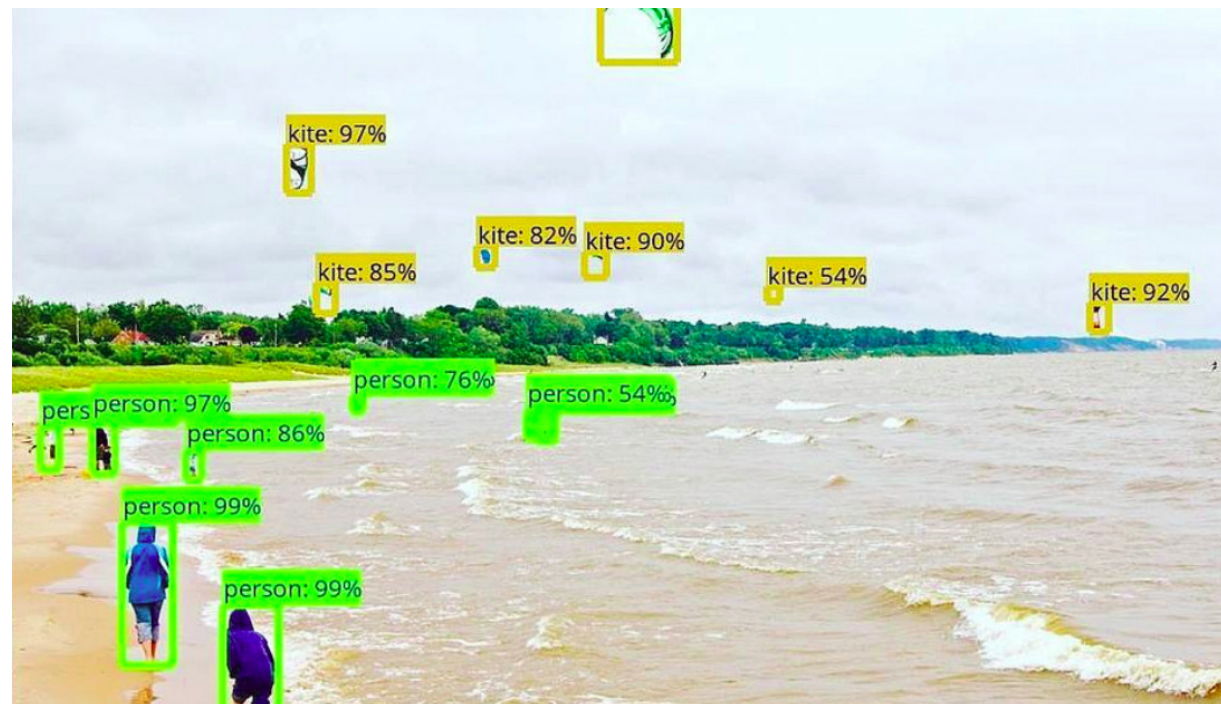
reconnaissance



Expressions  
Nom de la personne  
Etc.

# Détection d'objets

- Plus dur que la reconnaissance car les objets peuvent varier
  - Taille, rotation, etc. comme en reconnaissance
  - Position dan l'image : le nombre de rectangle à tester peu être grand



# Détection d'objets

## Region-Based Convolutional Neural Network (R-CNN )

- Segmentation (quelconque)
- Fusion de région : similarité de couleur, texture, forme, ...
- Chaque région produit une région d'intérêt : ROI



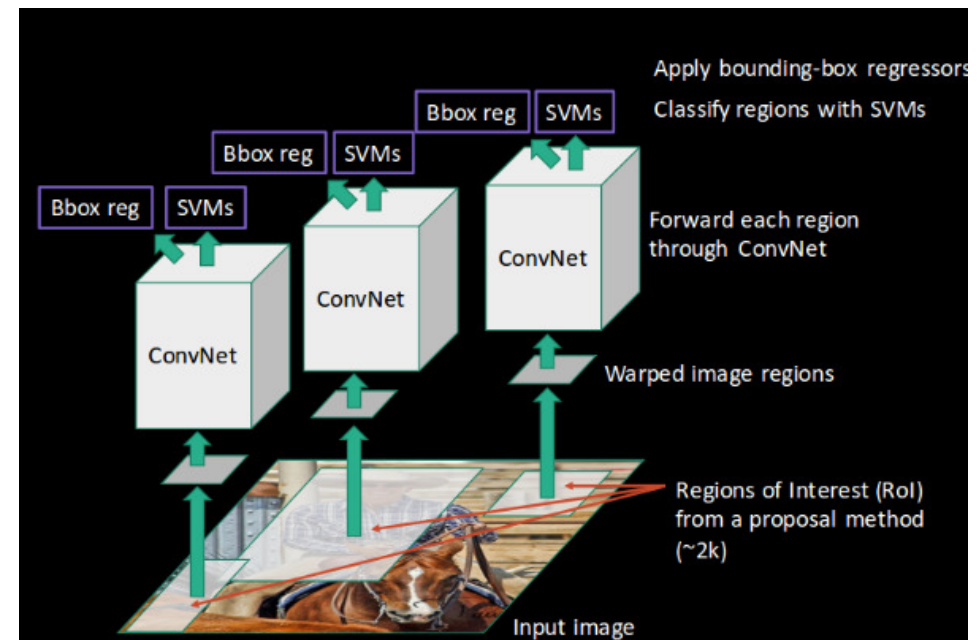
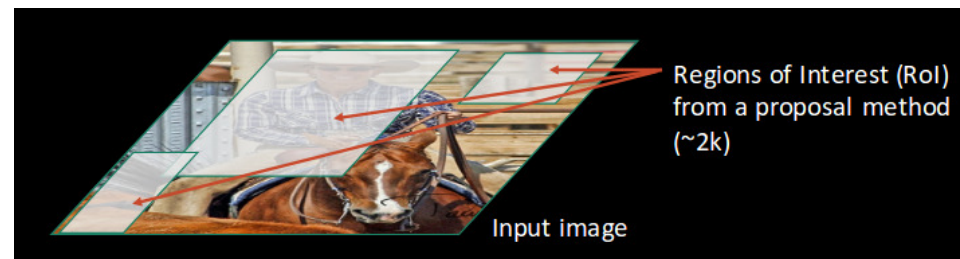
# Détection d'objets

## Region-Based Convolutional Neural Network (R-CNN)

- ROI
  - CNN → descripteurs
  - SVM classifie
  - BoundingBox regression pour ajuster la Bbox

### Problème avec R-CNN

- 2000 régions
  - 2000xNB descripteurs
- 1 minute par image



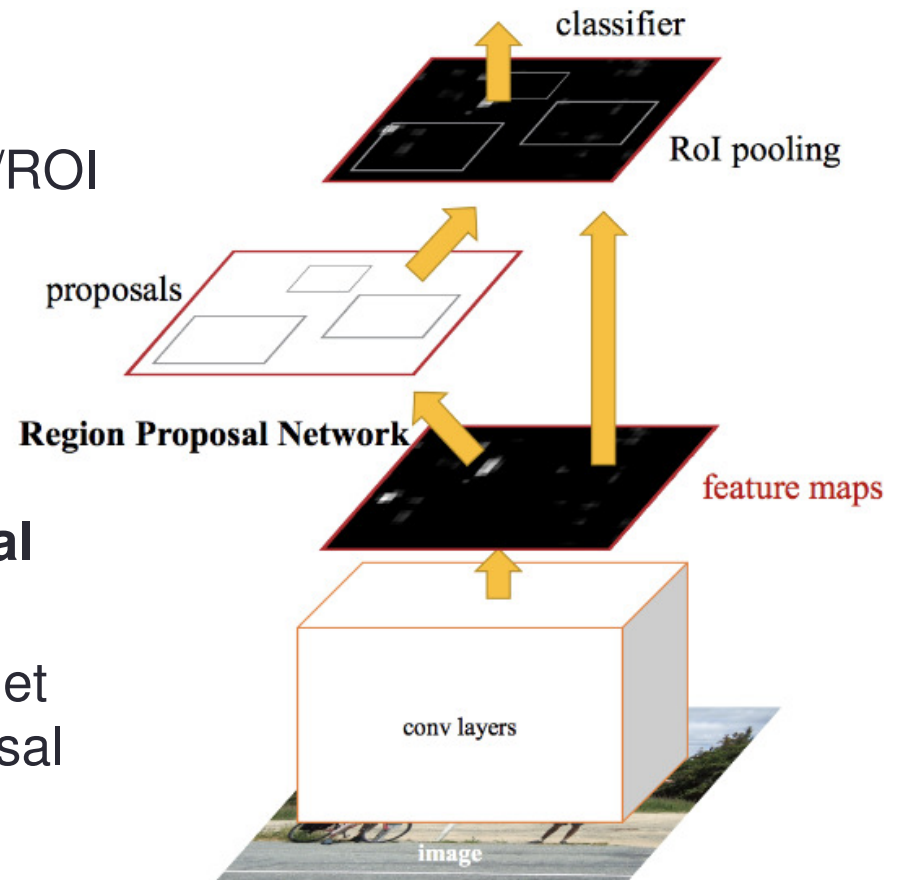
# Détection d'objets

## Fast Region-Based Convolutional Neural Network

- Replace la phase de segmentation/ROI par un CNN
- 2 secondes par images

## Faster Region-Based Convolutional Neural Network

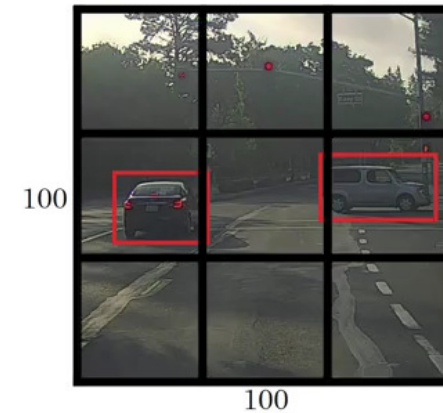
- Replace la phase de segmentation et sélection de ROI par Region Proposal Network (RPN)
- Temps 0.2 seconde par image



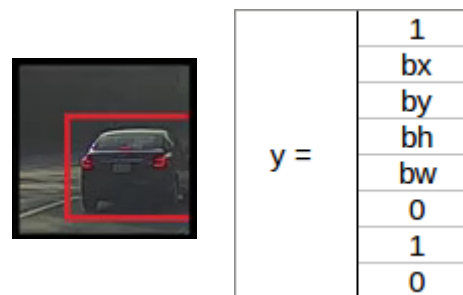
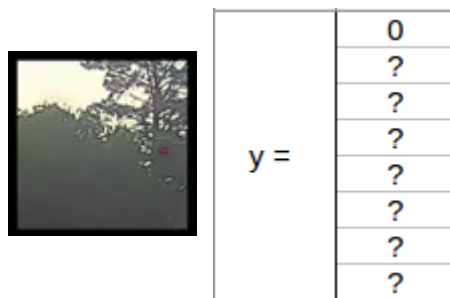


# Détection d'objets

- R-CNN; Fast R-CNN; Faster R-CNN
- YOLO : rapide, 45 images/seconde
  - Par exemple, détection de 3 classes
    - Pc : objet présent dans la fenêtre
    - bx,by,bh,bw : bounding box
    - c1, c2, c3 : présences des 3 classes

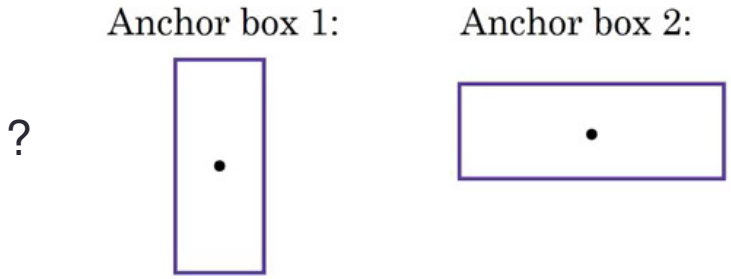
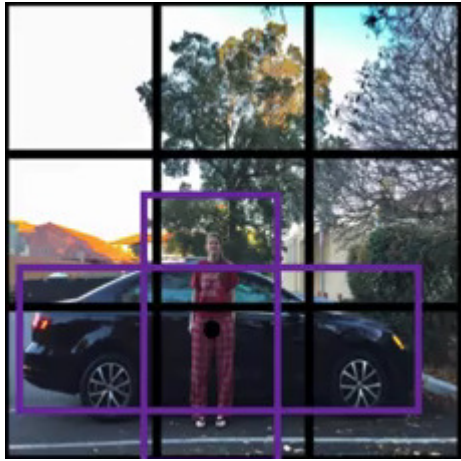
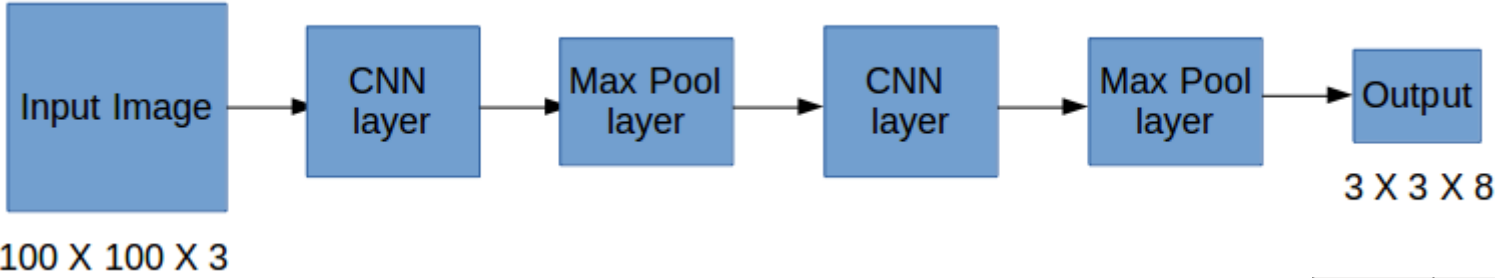


y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3



# Détection d'objets

- R-CNN; Fast R-CNN; Faster R-CNN
- YOLO : rapide, 45 images/seconde



y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3
	pc
	bx
	by
	bh
bw	
c1	
c2	
c3	



# Segmentation d'images

- Le zèbre 😊
  - La nature l'a fait évoluer pour se dissimuler
  - Le plus dur pour les algo de vision



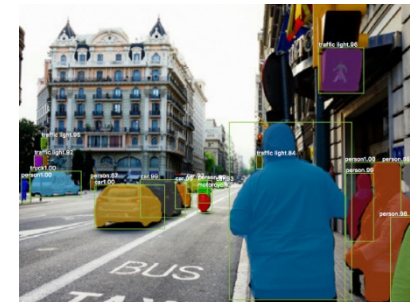
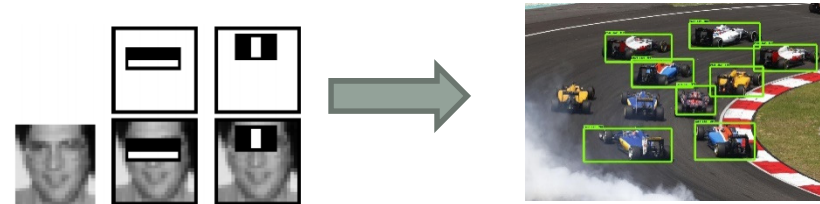
# Sémantique et image

- Entraîne un réseau à regrouper des régions **+ VIDEO** pour segmenter et mettre des labels sur une image



# Deep Learning a permis des avancées fortes en vision par ordinateur

- Souvent des idées simples mais malignes marchent le mieux
  - Viola–Jones object detection 2004
  - ...
  - ...
  - Yolo 2018
  - Dans les basiques reste à voir : Object Tracking, Semantic Segmentation, Instance Segmentation
- Ces outils de visions peuvent être maintenant
  - Utilisées dans des applications de tous les jours (mobile, jeux vidéo, ...)
  - Utilisées en recherche pour résoudre des problèmes d'une autre nature : Reconstruction 3D, MoCap, Extraction de textures, illumination, etc.



# POUR ALLER PLUS LOIN AVEC LES RÉSEAUX ET L'IMAGES

---

- Différents types de réseaux
  - RNN, LSTM, autoencoder, clustering, ...
  - vers de l'apprentissage semi ou non supervisée
- Apprentissage par transfert
- Un exemple d'application : transfert de style (+TP)
- ...



# Différents problèmes / différents réseaux

Images : classification et « traitement d'images »

- CNN à 2 niveaux de convolution (cf. TP)
- CNN à 19..50 niveaux → VGG, ResNET, AlexNet, GoogleNET, etc.

Données temporelles

- RNN
- LSTM

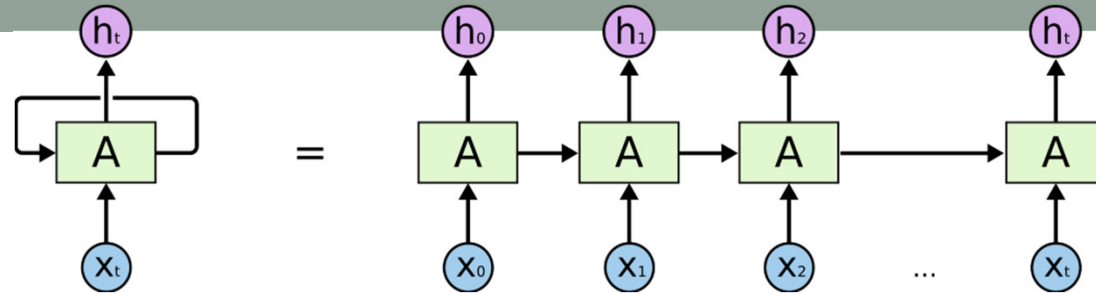
Semi supervisé : par exemple des données mais pas de labels

- Auto-encoder
- Construction de cluster : par exemple FaceNET

Divers problèmes utilisant fortement les réseaux

- Transfert de style sur des images [Gatys]
- Super résolution
- Segmentation
- Générer des données : GAN
- Apprentissage par renforcement : Deep Q-Learning, etc.

# RNN

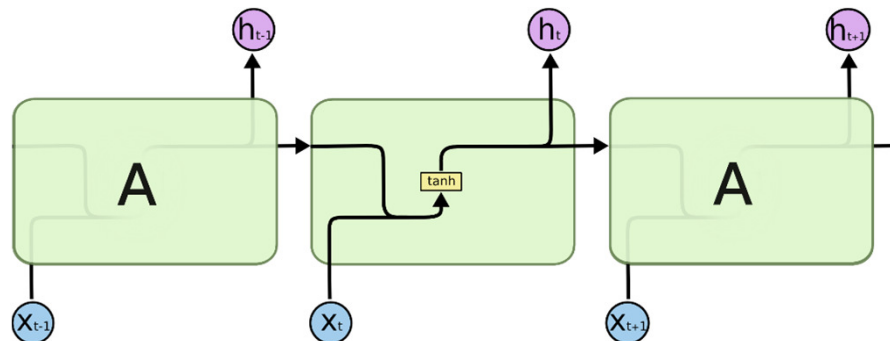


- Pour des données temporelles
  - Prédire le prochain mot
  - Composer de la musique
  - Reconnaître le langage parlé
  - Détection d'erreur dans une série d'événements
  - Prédiction de la bourses

→ Recurrent Neural Network, LSTM, Gated Recurrent Unit,

...

RNN simple



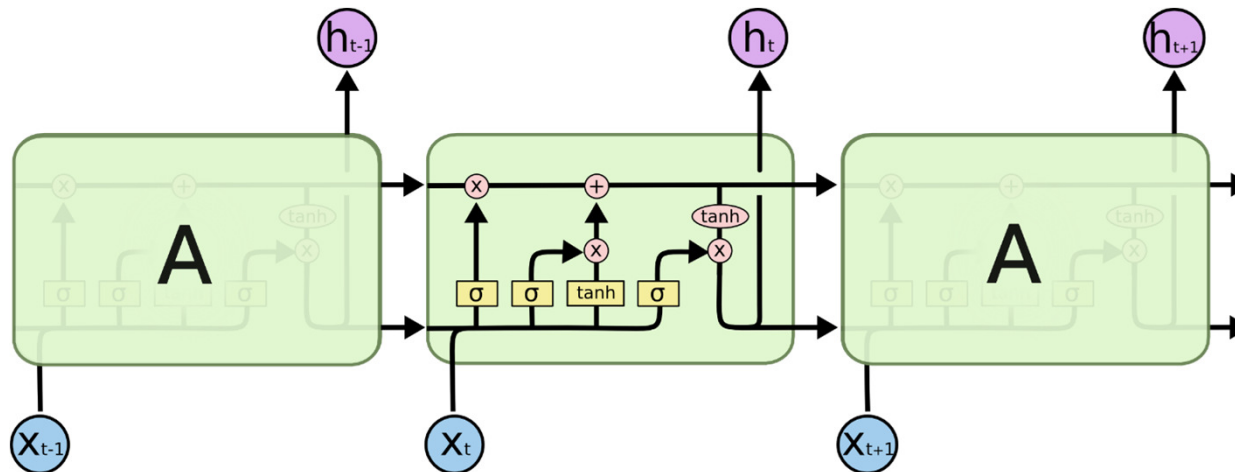
# LSTM

- Pour des données temporelles

→ Long Short-Term Memory

Par exemple

- 6 -> 7 -> 8 -> ?      On voudrait 9
- 2 -> 4 -> 8 -> ?      On voudrait 16
- Se baser sur 8 ne suffit pas
- A une mémoire courte et long terme
- **Apprend quand se souvenir et quand oublier**

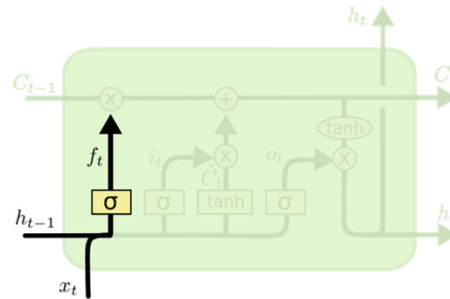




# LSTM

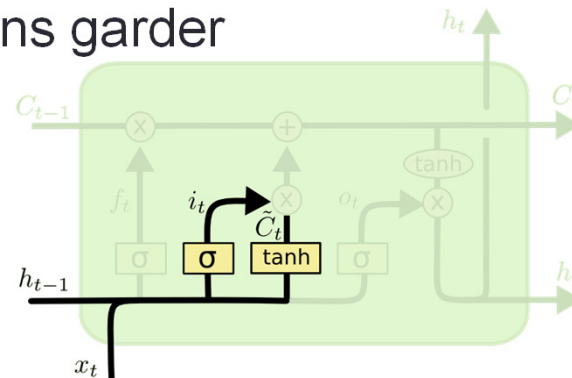
## Les couches

- Oublier ou garder: 0..1



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

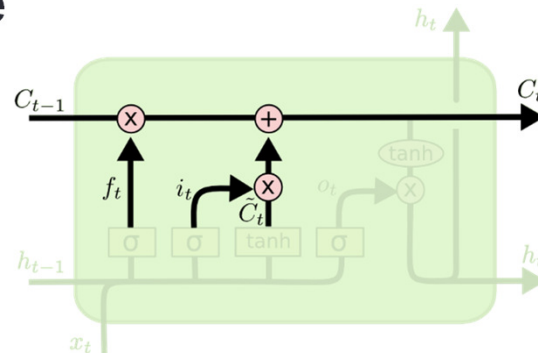
- Quelles informations garder



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

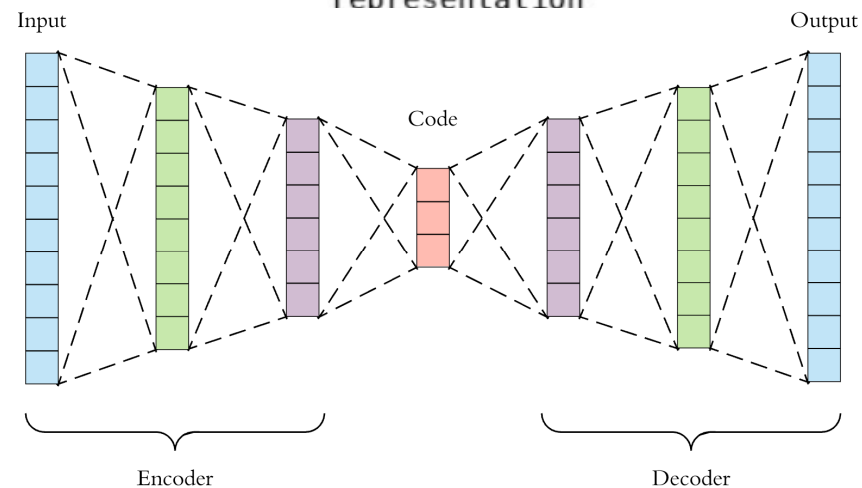
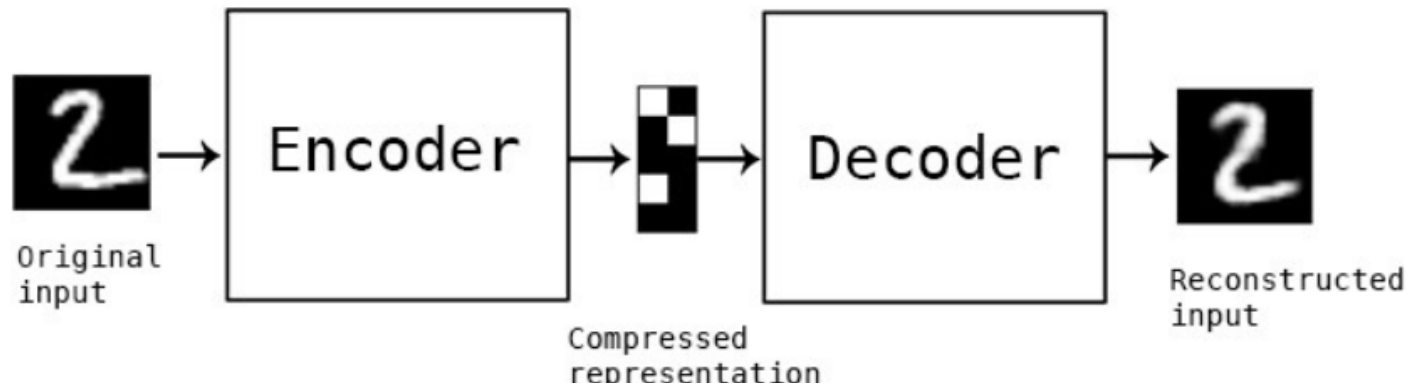
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- Produire la sortie



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

# Auto Encoder (AE) : principe

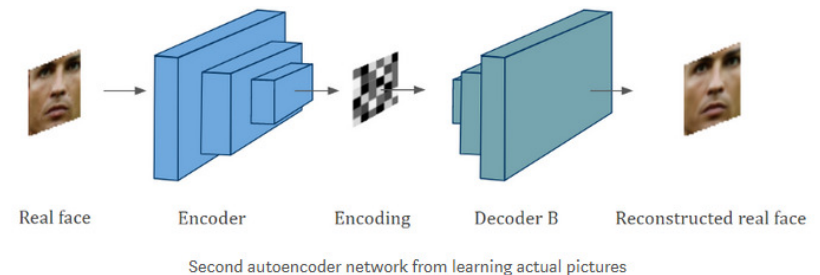
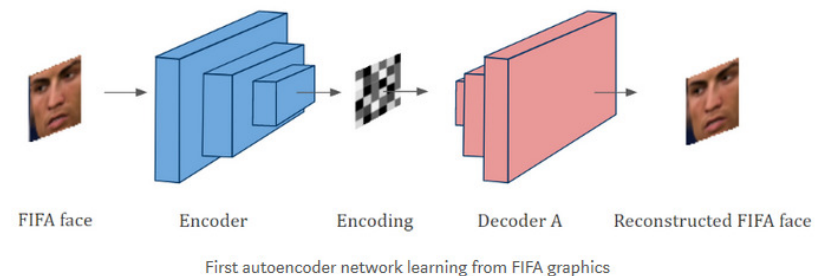
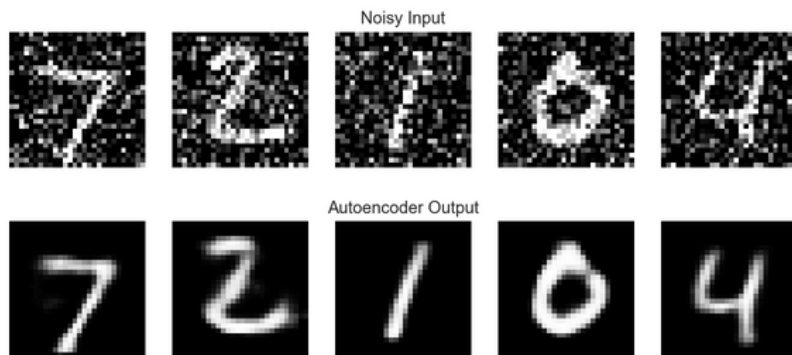
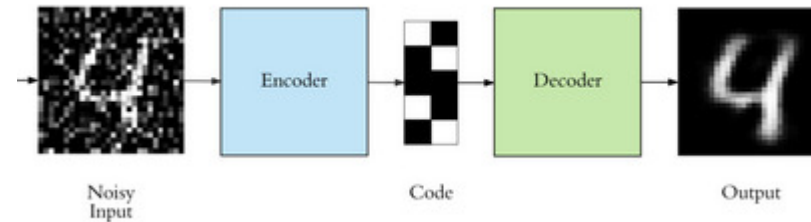


$$Obj = L(x, \hat{x})$$

- Il existe de nombreux types d'autoencoder

# Auto Encoder (AE) : principe

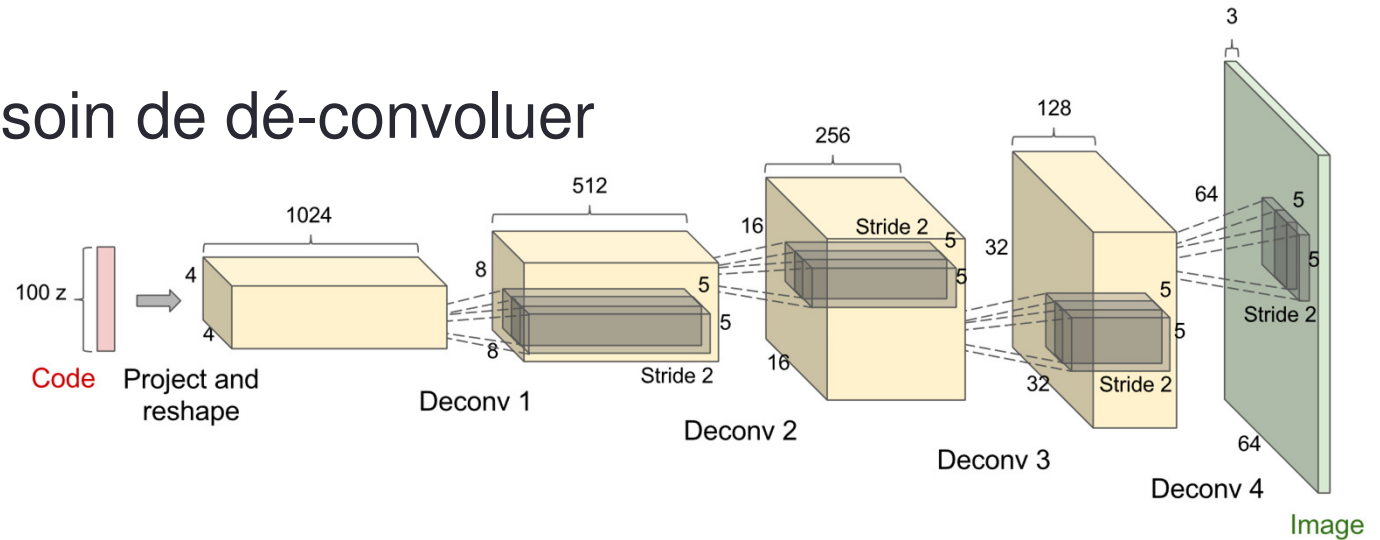
- A quoi ca sert ?
  - Débruitage
  - Compression
  - Code compact
    - Possibilité d'appliqué des traitements sur le « code »
    - Un peu le même esprit que la PCA
  - Prémisse du Génératif



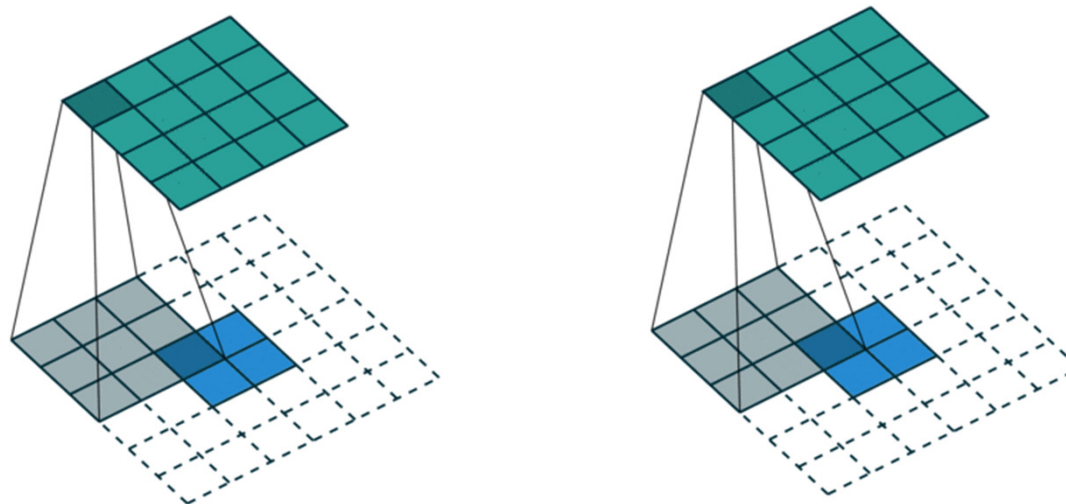
Amélioration visage de Ronaldo dans FIFA

# Decode

Si image : besoin de dé-convoluer

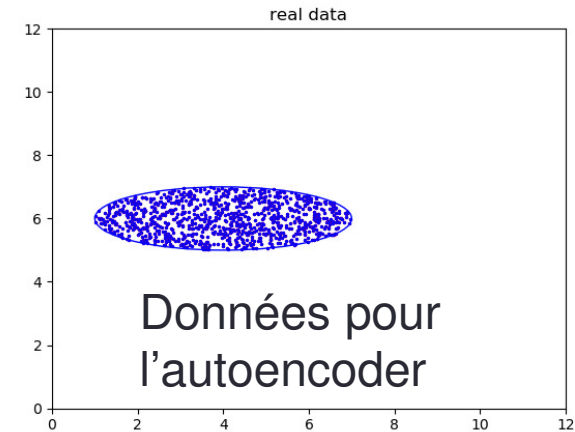


Devrait s'appeler plutôt transposed convolutional layer

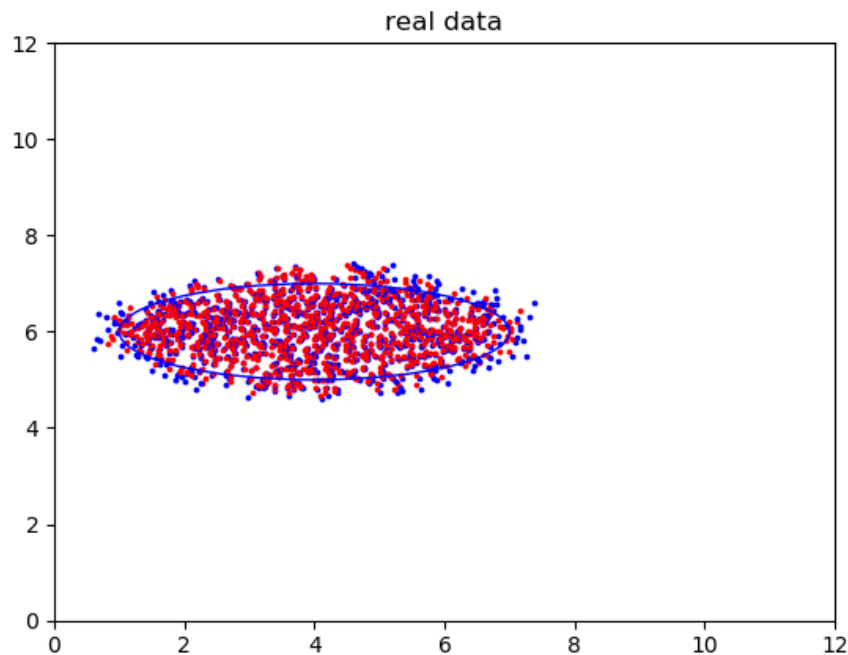


# Auto-encoder

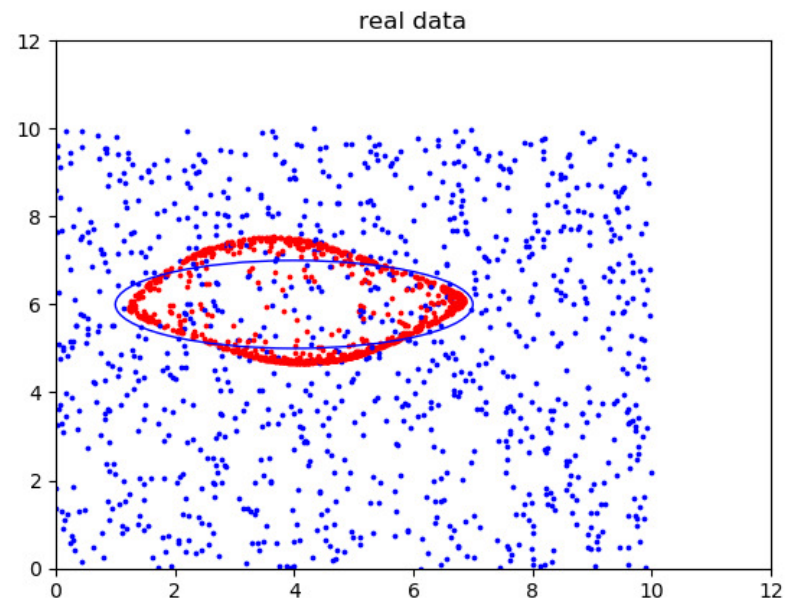
- Démo nuage de points
  - activate p37-keras
  - points bleus donnés à l'autoencoder qui donne les points rouges



Après 1 passe



après plusieurs passes

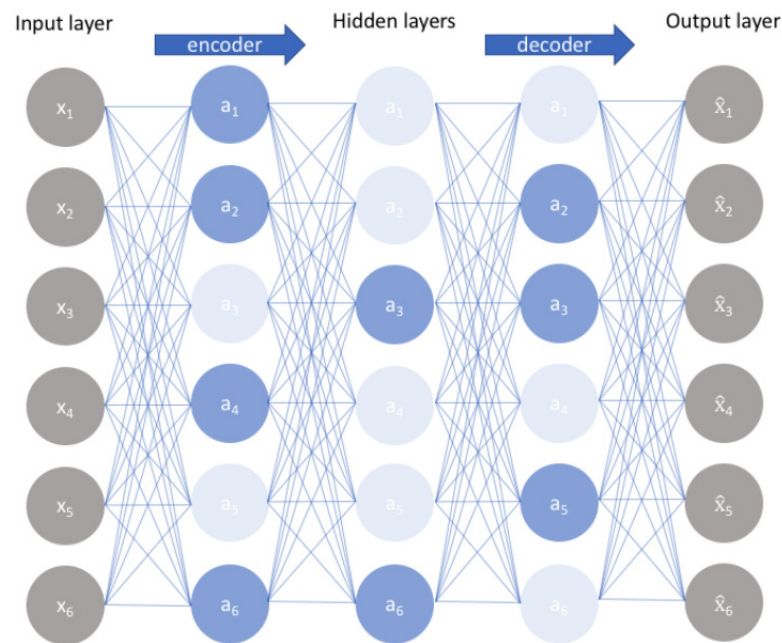


# Sparse Auto Encoder

Dans la fonction de coût, favorise « sparsity » (clairsemé)

→ Améliore les performances

- Intuition : avoir essentiellement des neurones utiles (avec un poids fort) pour avoir une représentation « intelligente »



Sparse Autoencoder

Distance entre input et output

$$Obj = L(x, \hat{x}) + \lambda \sum_i |a_i^{(h)}|$$

Favorise la sparsity



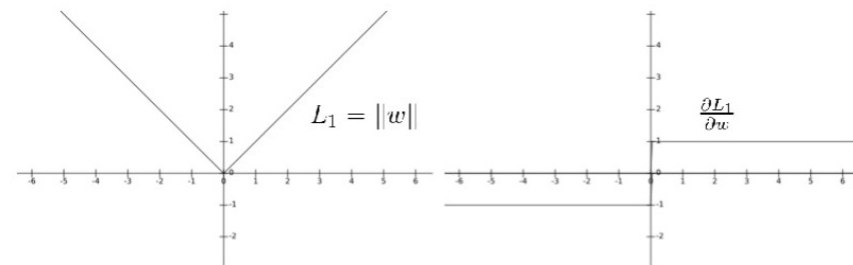
# Sparse Auto Encoder : norme L1 / L2

- Utiliser la norme L1 dans la fonction de coût

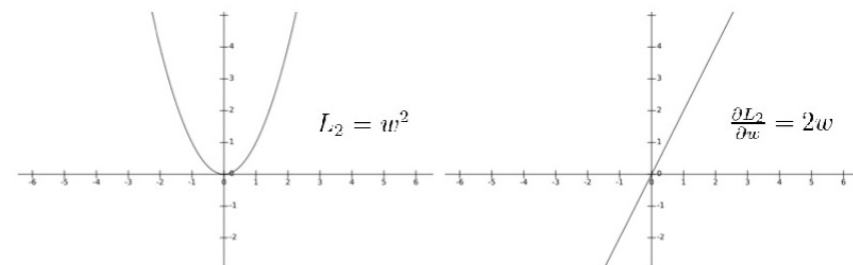
$$Obj = L(x, \hat{x}) + \lambda \sum_i |a_i^{(h)}|$$

Intuition : gradient vaut -1 ou +1 donc un pas plus grand à chaque itération que avec la norme L2 qui va faire des petits pas

$$L_1 = \|w\|, L_2 = w^2$$



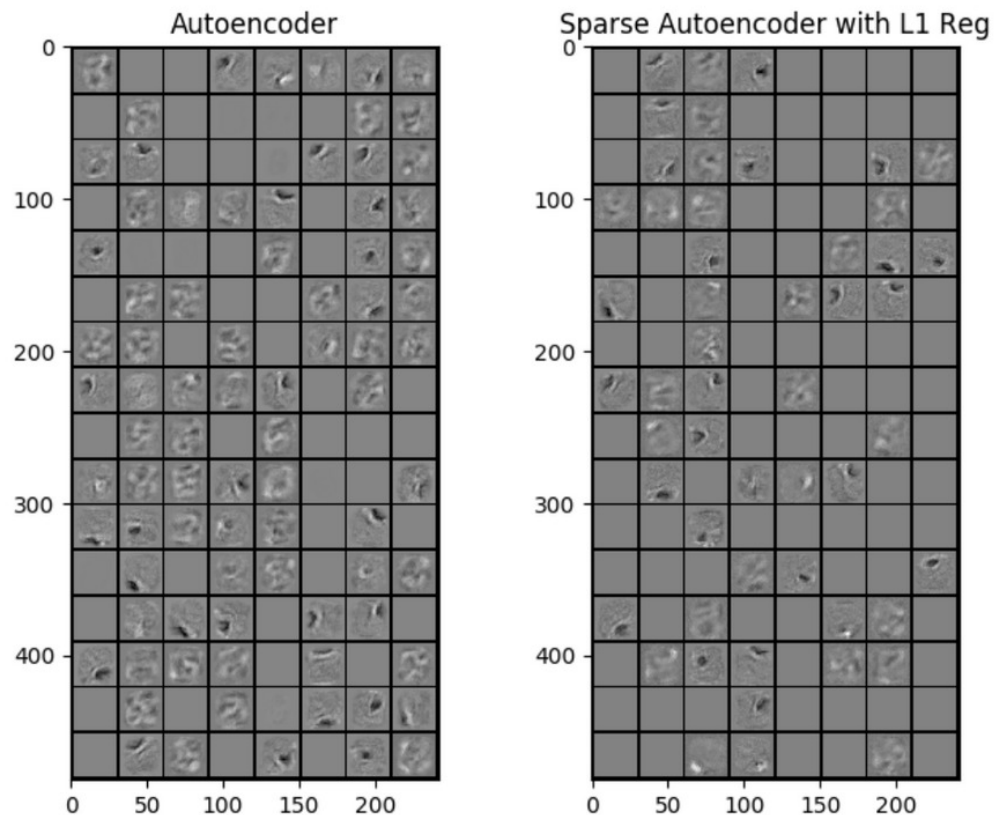
L1 regularization and its derivative



L2 regularization and its derivative

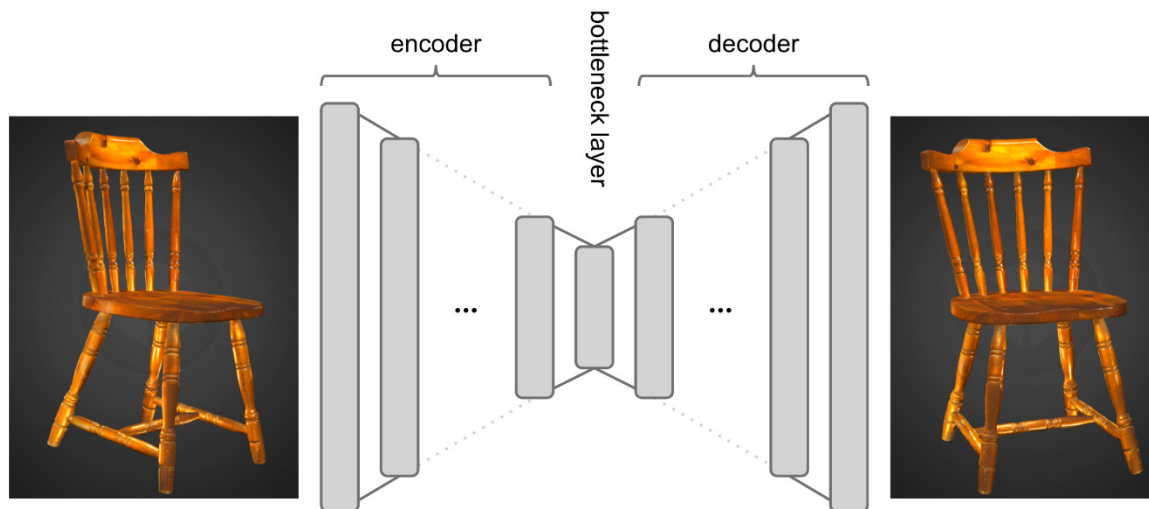
# Sparse Auto Encoder : norme L1

- Moins de neurones activés → les neurones utiles plus efficaces → représentation plus « intelligente »



# Auto-encoder ...

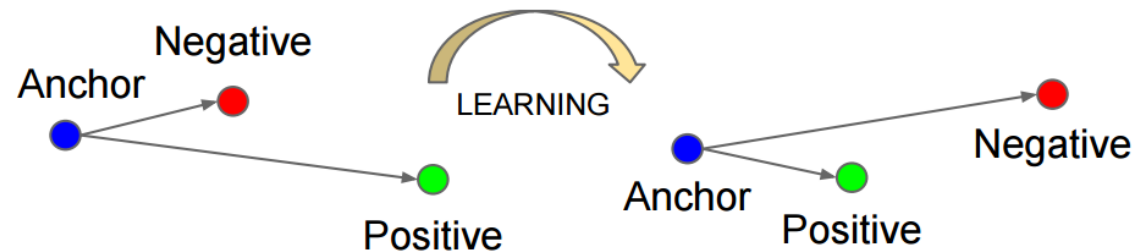
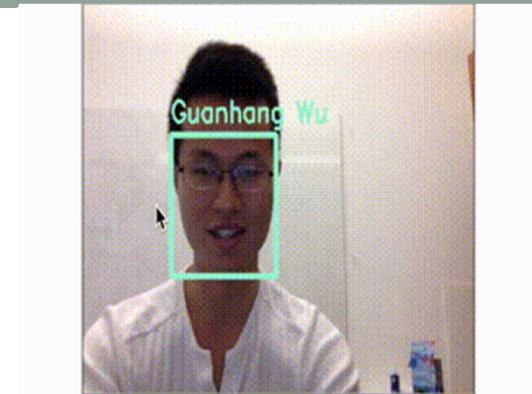
- De nombreux auto encodeur existent
  - Variational AE, ...
  - Directement inclus dans un GAN
  - Possibilité de les entrainer sur autres choses que des images
    - Maillage 3D
    - Animation (cf. cours d'animation)
  - Encore un fois un problème vaste ... mais un outils puissant



# FaceNet

Reconnaitre le nom de la personne d'une image de son visage

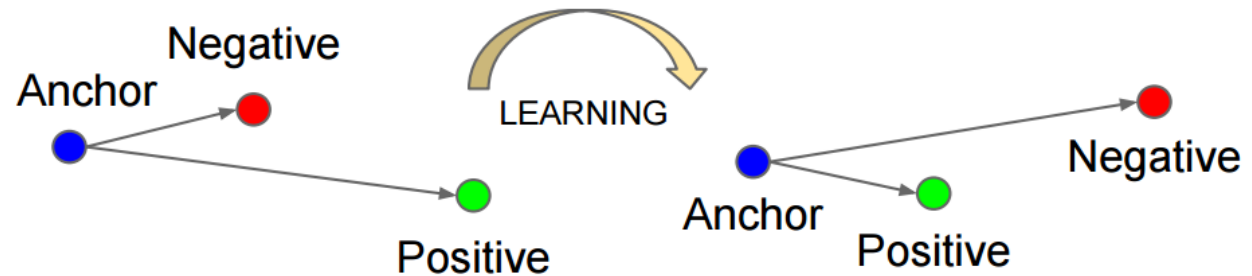
- Plus efficacement que CNN de base
- Apprend une représentation des données pour rapprocher les éléments ayant un rapport entre eux et éloigner les autres → **construire des clusters**



FaceNet: A Unified Embedding for Face Recognition and Clustering, CVPR 2015

<http://cmusatyalab.github.io/openface/>

# FaceNet

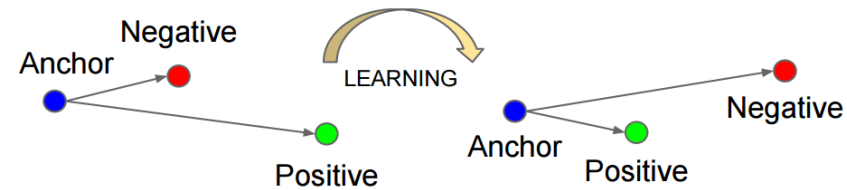


- Utilise un triplet
  - une instance “anchor”  $x$  : le visage d’une personne A
  - une instance positive  $x_+$  : le visage de la même personne A
  - une instance negative  $x_-$  : le visage d’une autre personne B
- $f(x)$  la representation de  $x$ , la fonction de coût

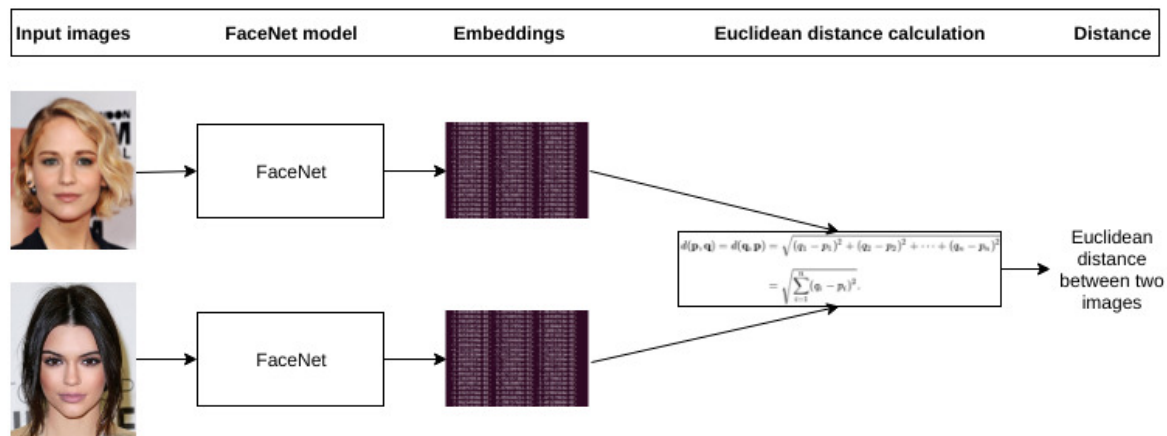
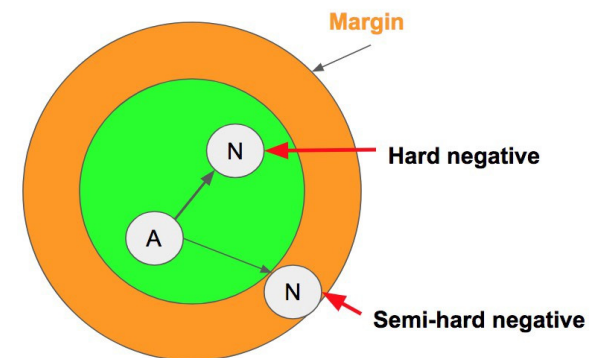
$$\max(0, \|f(x) - f(x_+)\|^2 - \|f(x) - f(x_-)\|^2 + \alpha)$$

- Ignore le triplet quand  $x_+$  est déjà plus proche que  $x_-$
- Apprendre  $f(x)$  en utilisant la backpropagation

# FaceNet



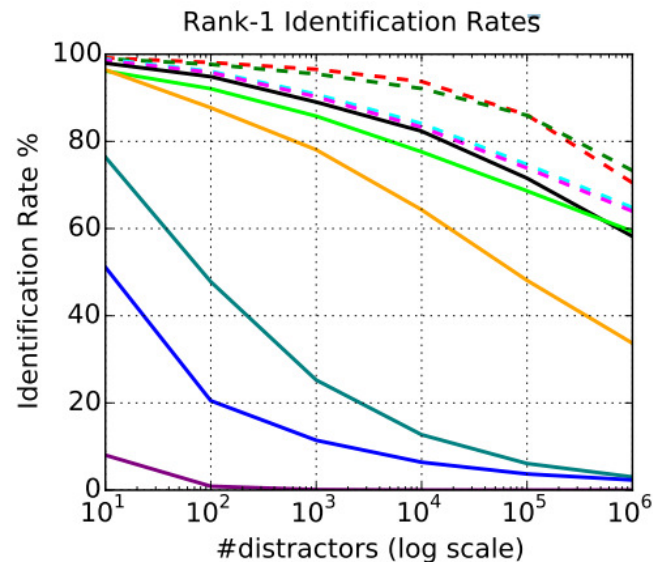
- Le nombre de triplet est gigantesque
  - $10^6$  de visage  $\Rightarrow 10^{18}$  triplets
  - $\rightarrow$  progression
  - Choisir des triplets semi difficiles
  - Choisir des triplets difficiles
- Produit une description d'un visage en 128 dimensions



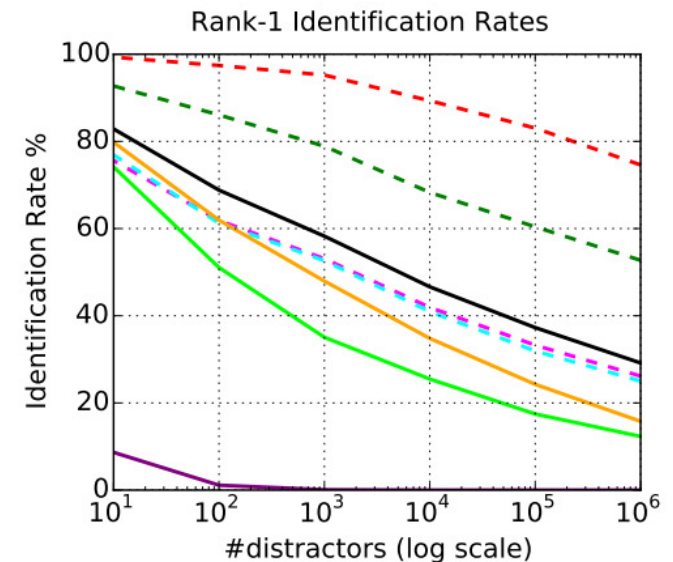


# MegaFace Benchmark

- Google – FaceNet v8
- NTechLAB - FaceNLarge
- Beijing Faceall 1600Norm
- Beijing Faceall 1600
- NTechLAB – FaceNSmall
- Barebones\_FR
- 3DiVi – tdvm6
- LBP
- Joint Bayes
- Random



(a) FaceScrub + MegaFace



(b) FGNET + MegaFace

# FaceNet : identifier une personne

- FaceNET les erreurs par paires →
  - Parmi ces erreurs 13 ont été mal classées dans la base de données
- Un exemple de cluster pour une personne

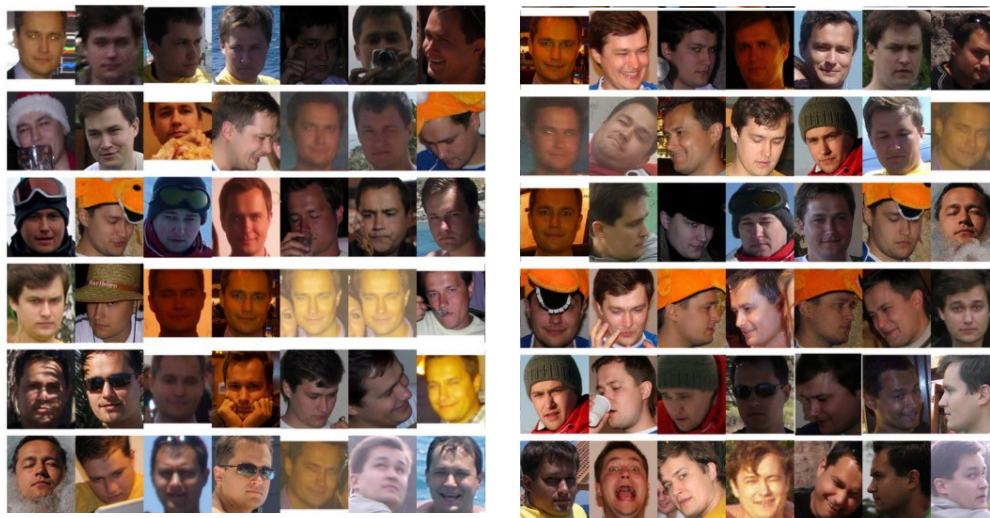


Figure 7. **Face Clustering.** Shown is an exemplar cluster for one user. All these images in the users personal photo collection were clustered together.



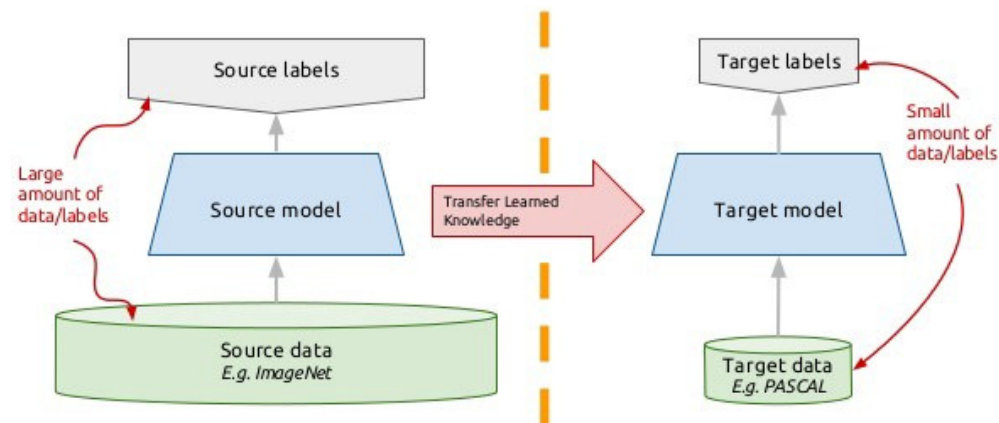
Figure 6. **LFW errors.** This shows all pairs of images that were incorrectly classified on LFW. Only eight of the 13 false rejects shown here are actual errors the other five are mislabeled in LFW.

# Transfert d'apprentissage avec CNN

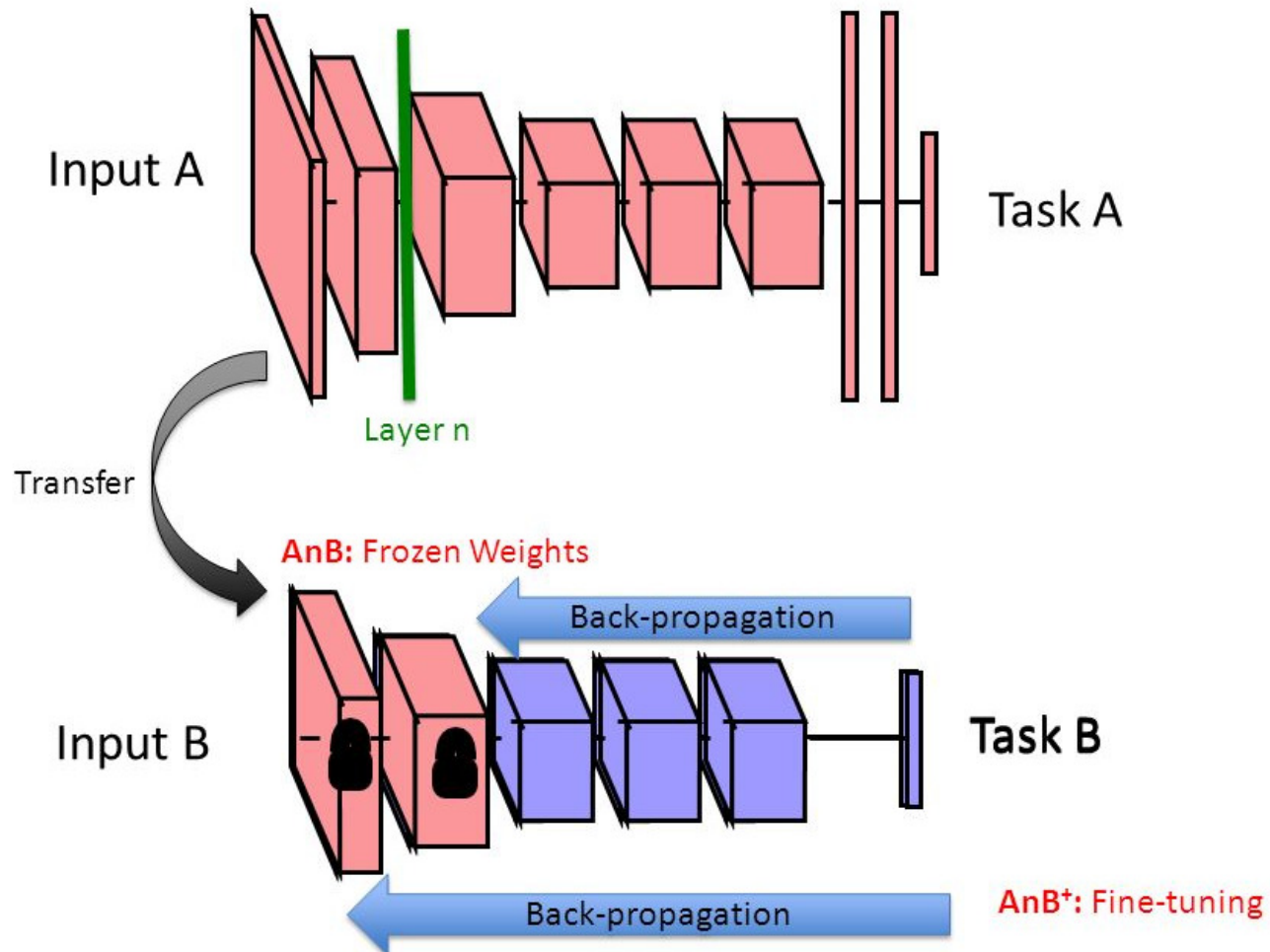
Approche simple pour se spécialiser

- Modèle pré-entraîné sur une grande base de données assez générale
- Gèle certains paramètres (weights) : couches basses de convolution
- Ajoute des couches « classifier » avec ses paramètres à entraîner sur les données spécifiques
- Entraîne ce réseau sur les données spécifiques
- Eventuellement dégèle tous les paramètres à la fin

## Transfer learning: idea



# Transfert d'apprentissage avec CNN



# Transfert d'apprentissage avec CNN

- Un exemple avec PyTorch

```
from torchvision import models
model = models.vgg16(pretrained=True)
```

```
# Freeze model weights
for param in model.parameters():
    param.requires_grad = False
```

```
import torch.nn as nn

# Add on classifier
model.classifier[6] = nn.Sequential(
    nn.Linear(n_inputs, 256),
    nn.ReLU(),
    nn.Dropout(0.4),
    nn.Linear(256, n_classes),
    nn.LogSoftmax(dim=1))
```

# Transfert d'apprentissage avec CNN

## Transfert learning

- Un domaine vaste
- Adapter un apprentissage est un des prochains verrous
- ...

**How transferable are features in deep neural networks?**

[Jason Yosinski](#), [Jeff Clune](#), [Yoshua Bengio](#), [Hod Lipson](#)

NIPS2014



# POUR ALLER PLUS LOIN AVEC LES IMAGES

---

Les réseaux sont un outils pour de nombreux problèmes, on profite

- Framework
  - GPU, Optimiseur, etc.
  - Learning en python mais après utilisation en C++, C#, Java, etc.
  - Format standard de fichiers : Open Neural Network Exchange
- Communauté grande
- Nombreux tutoriaux et explications
  - [medium.com](https://medium.com)
  - [letslearnai.com](https://letslearnai.com)
  - Etc.
- Recherche reproductible (Github)

# Transfert de style entre images

## 1 Upload photo

The first picture defines the scene you would like to have painted.



## 2 Choose style

Choose among predefined styles or upload your own style image.



## 3 Submit

Our servers paint the image for you. You get an email when it's done.

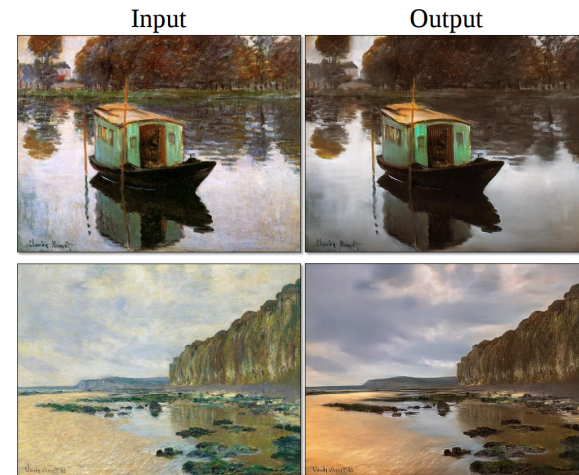


Image Style Transfer Using Convolutional Neural Networks  
Gatys et al. CVPR 2016

# Transfert de style entre images

- Différentier
  - Contenu de l'image : objets et leurs places/positions/orientations
  - Style : couleur et textures
- VGG19 pour extraire les caractéristiques
  - Chaque couche de convolutions va produire une carte de caractéristiques (features)
  - Optimisation avec deux termes :  $\text{Coût\_Contenu} + \text{Coût\_Style}$



Caractéristiques à différentes échelles



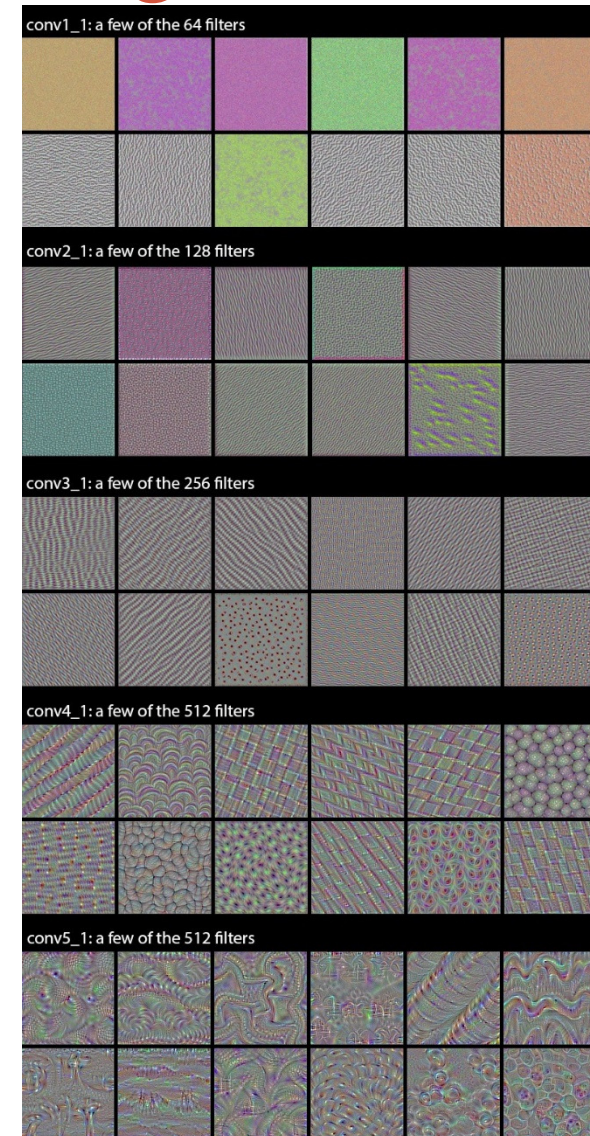
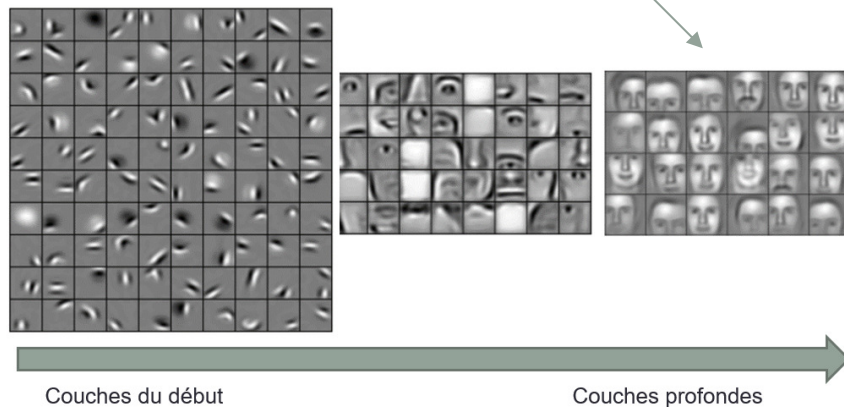
# Transfert de style entre images

VGG19 pour extraire les caractéristiques

- Chaque couche de convolutions va produire un vecteur de caractéristiques de dimension

Batch\_size x Nfeatures x Height x Weight

- Certaines couches codent plutôt le contenu (vers le fond du réseau), d'autres plutôt le style (vers le début)



Features visualization of VGG network

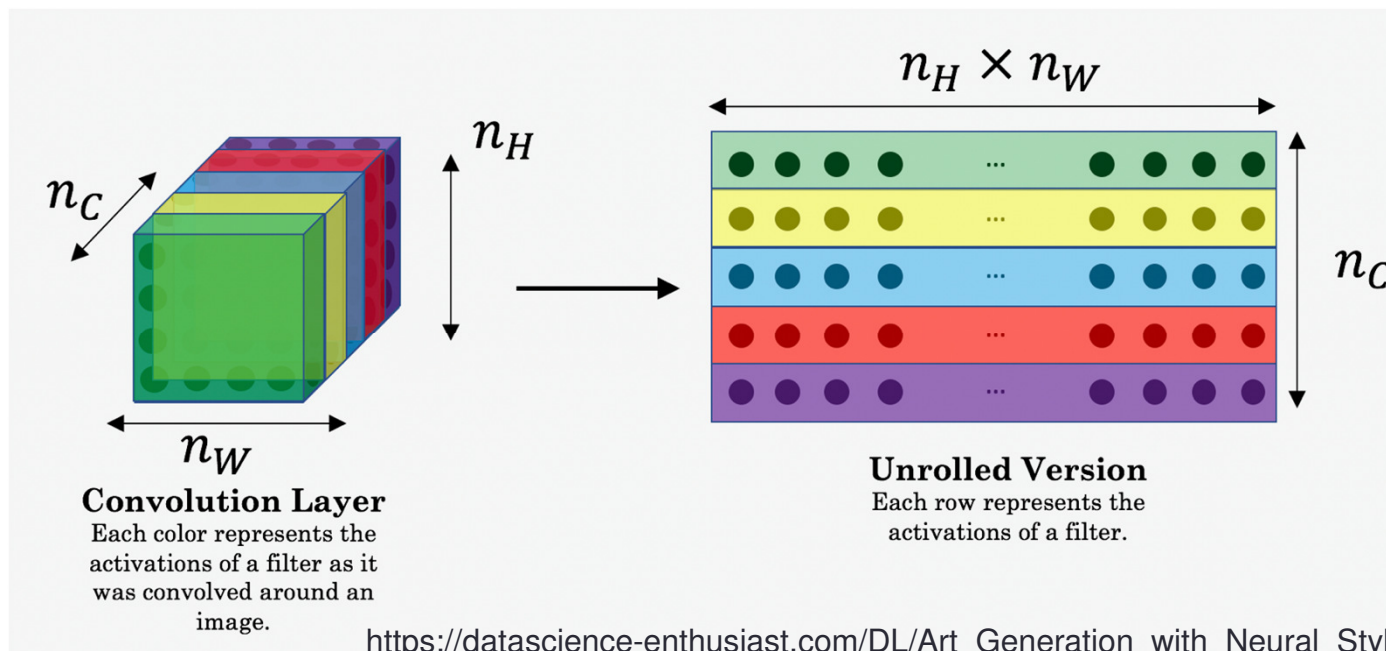
# Transfert de style entre images

VGG19 pour extraire les caractéristiques

- Chaque couche de convolutions va produire un vecteur de caractéristiques de dimension

$N_{\text{features}}(N_c \text{ sur la figure}) \times \text{Height} \times \text{Weight}$

→ à aplatir en  $N_{\text{features}} \times N_{\text{pixels}}$  avec  $N_{\text{pixels}} = \text{Height} \times \text{Weight}$



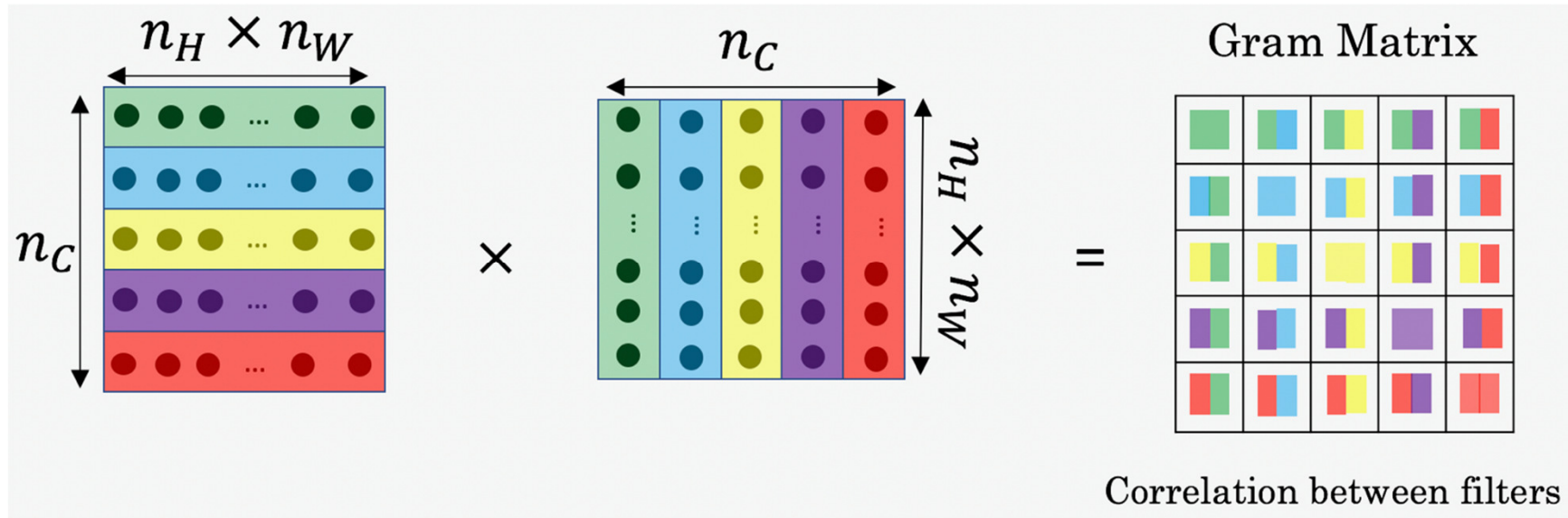
# Transfert de style : matrice de Gram

- **Matrice de Gram :  $M \times M^t$** 
  - Produit scalaire entre toutes les caractéristiques
  - Corrélation entre les features
- Avec une matrice F de caractéristique  $\mathbf{F}$ , une entrée de la matrice de Gram G est

$$G_{ij} = \sum_k F_{ik} F_{jk}$$



# Transfert de style : matrice de Gram



Si une entrée dans la matrice de Gram a une valeur proche de 0, cela signifie que les 2 *features* ne s'activent pas simultanément (non corrélation). Et vice versa, si une entrée a une grande valeur, cela signifie que les 2 *features* s'activent simultanément (corrélation).

Nous allons chercher à créer une image qui réplique un même schéma d'activations des *features* de style.

# Transfert de style : coût de contenu

- Si on peut construire une image qui a une carte de caractéristiques équivalentes pour un niveau de convolution donné à une autre image. Ces deux images auront le même contenu (surtout pour les couches profondes) — mais pas nécessairement la même texture ou style.
- Soit une couche de convolution  $l$  dans VGG, la fonction de coût de contenu est défini comme la moyenne au carré de l'erreur entre la carte de *features*  $\mathbf{F}$  de l'image de contenu  $\mathbf{C}$  et la carte de *features* de l'image générée  $\mathbf{Y}$ .

$$\mathcal{L}_{content} = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

# Transfert de style : coût de style

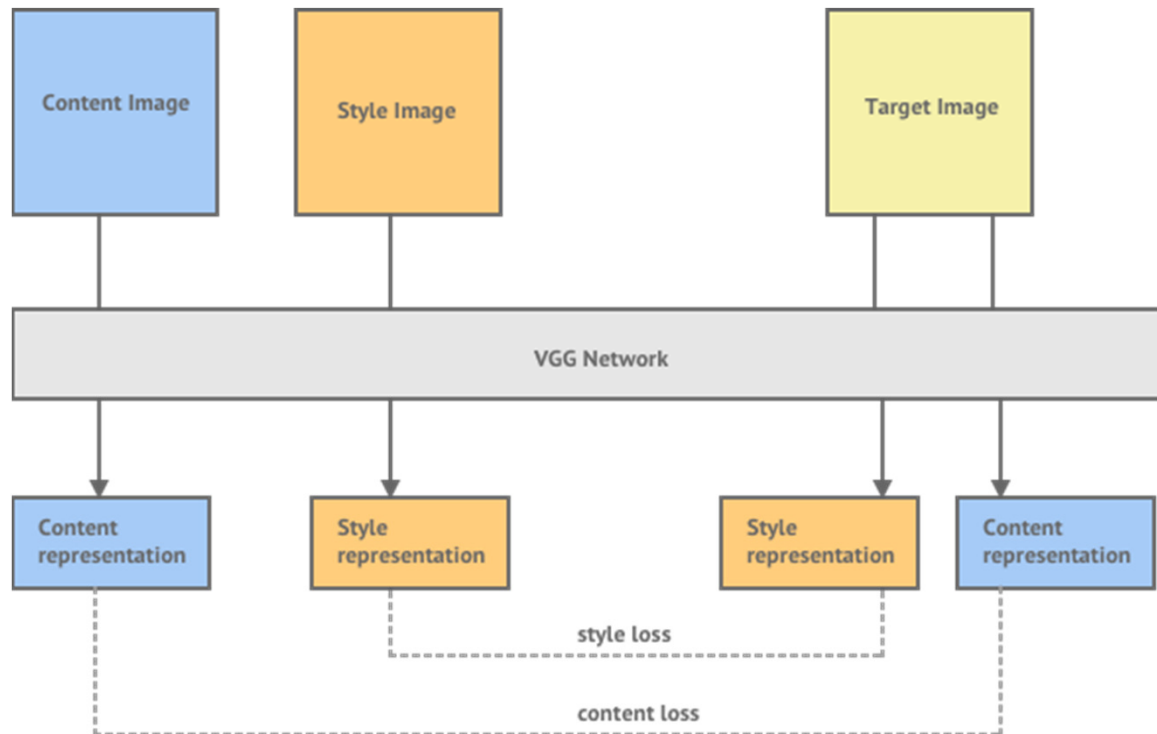
- Le calcul de coût du style est similaire au calcul de coût de contenu, mais on le calcule à partir de la matrice de Gram à la place de directement utiliser les features

$$\mathcal{L}_{style} = \frac{1}{2} \sum_{l=0}^L (G_{ij}^l - A_{ij}^l)^2$$

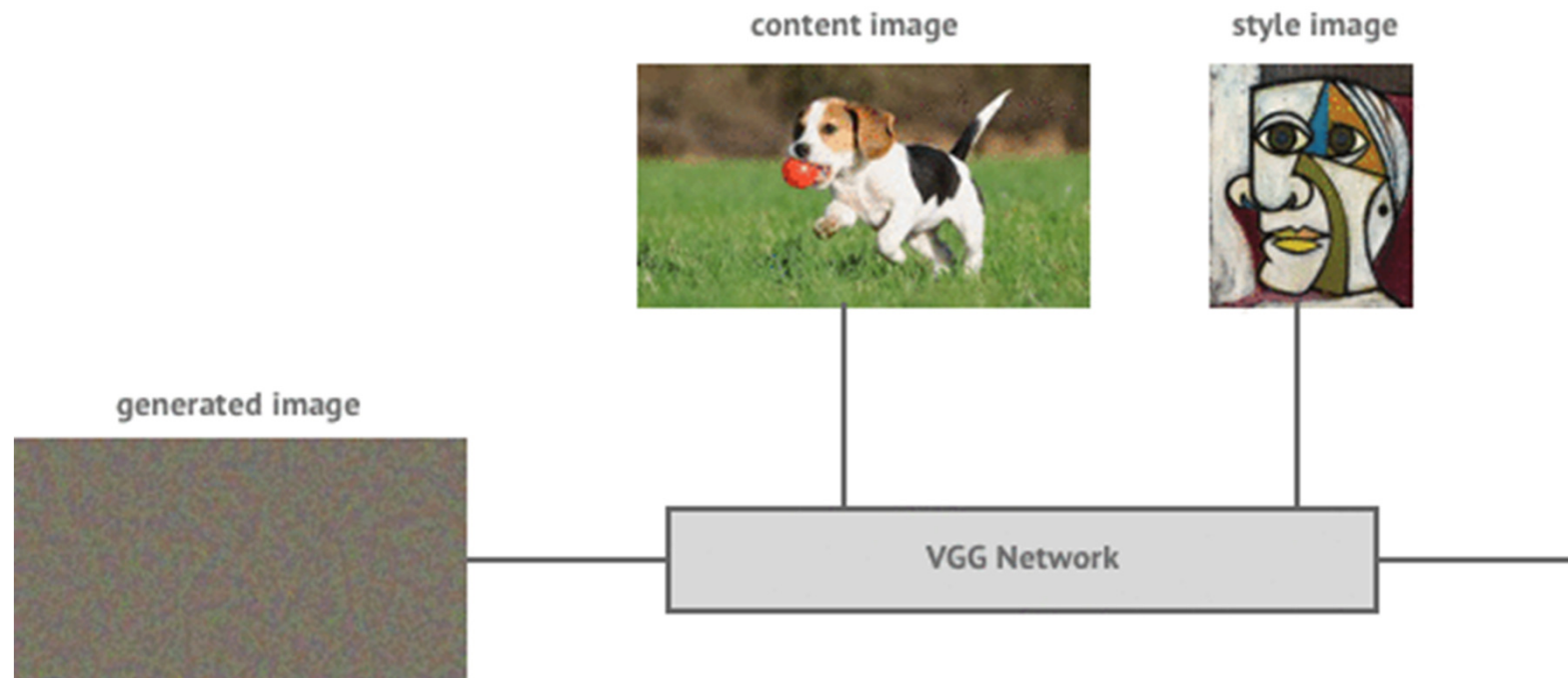
# Transfert de style : optimisation

- La fonction de coût à optimiser

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$

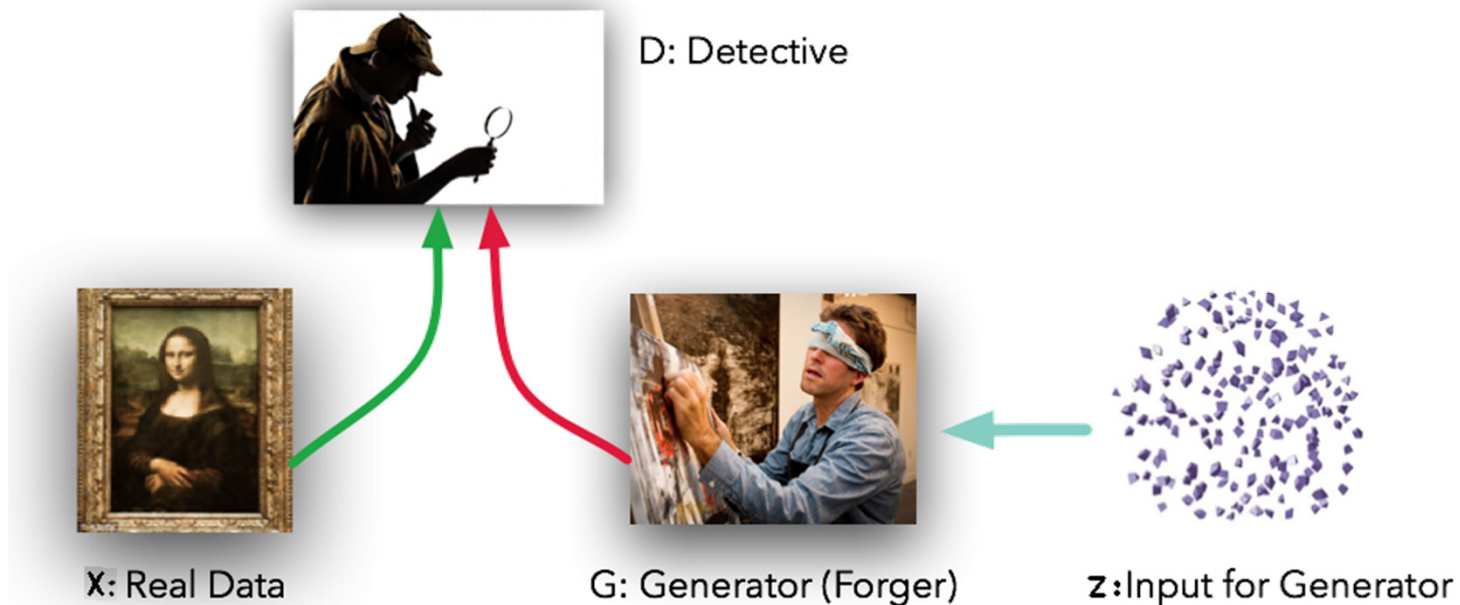


# Transfert de style



# Réseau de neurones et génération : GAN

- GAN = Generative Adversarial Networks
  - Proposé en 2016, déjà plusieurs centaines de papiers/variantes
  - + Fonctionne avec une quantité de données moindre car en génère
  - Instable





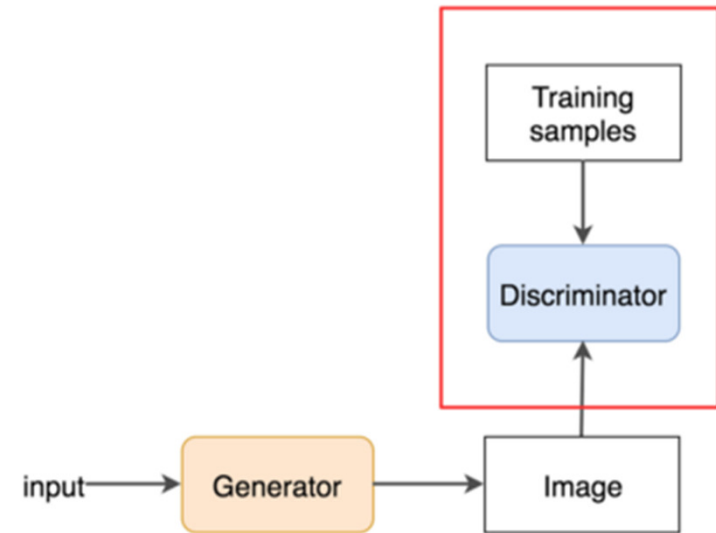
# Réseau de neurones et génération : GAN

- Generative Adversarial Networks (2014), Goodfellow et al.
  - Cité 2000 fois en 3 ans
- Plusieurs centaines de variantes
- Les applications « cools » des GAN
  - Génération de visages
  - Personne avec poses différentes
  - Transfert de texture
  - Super résolution
  - Texte vers images
  - ...

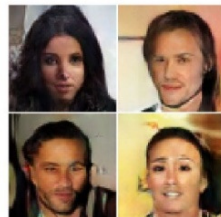
- A General Retrieval Network for Scalable Adversarial Classification (Paper)
- Attention Unification Generative Adversarial Nets (arXiv)
- LossGAN: Boosting Generative Models (arXiv)
- Adversarial Autoencoders (arXiv)
- Adversarial Discriminative Domain Adaptation (arXiv)
- Generative Generator-Encoder Networks (arXiv)
- Adversarial Natural Language (arXiv) [Code]
- Adversarially Learned Inference (arXiv) [Code]
- GCGAN: adversarial GANs with GAN (arXiv)
- An Adversarial Regularization for Semi-Supervised Training of Structured Output Neural Networks (arXiv)
- WGAN-GP: Wasserstein GAN with Gradient Penalty (arXiv)
- Wasserstein Adversarial Networks (arXiv)
- Wasserstein GAN (arXiv)
- Wasserstein GAN with Gradient Penalty (arXiv) [Code]
- Boundary Equilibrium Generative Adversarial Networks (Paper) (arXiv) [Code]
- Binary Generative Adversarial Networks for Image Retrieval (arXiv)
- Boundary-Equilibrium Generative Adversarial Networks (arXiv) [Code]
- CausalGAN: Learning Causal Implicit Generative Models with Adversarial Training (arXiv)
- Class-Specific Generative Adversarial Networks (arXiv)
- Comparison of Variational Autoencoders and GAN-based Training of Real NVPs (arXiv)
- Conditional GANs for Arbitrary-Dimensional Image Generation (arXiv)
- Conditional Generative Adversarial Nets (arXiv) [Code]
- Connecting Generative Adversarial Networks and Self-Organizing Maps (Paper)
- Continuous Learning in Generative Adversarial Nets (arXiv)
- CGAN-GAN: Continuous recurrent neural networks with adversarial training (arXiv)
- CM-GAN: Cross-modal Generative Adversarial Networks for Common Representation Learning (arXiv)
- Cooperative Training of Descriptor and Generator Networks (arXiv)
- CoopGAN: Cooperative Generative Adversarial Networks (arXiv) [Code]
- Duality GAN (arXiv)
- Deep and Hierarchical Implicit Models (arXiv)
- Energy-based Generative Adversarial Networks (arXiv) [Code]
- Exploring and Harnessing Adversarial Examples (arXiv)
- Real-GAN: Bridging Implicit and explicit learning in generative models (arXiv)
- F-GAN: Training Generative Neural Networks using Variational Discrepancy Minimization (arXiv) [Code]
- GAN of GAN: Generative Adversarial Networks with Variational Inference Learning (arXiv)
- Generative Station and Equilibrium in Generative Adversarial Nets (GAN) (arXiv)
- Generating Images with recurrent adversarial networks (arXiv)
- Generative Adversarial Networks (arXiv) [Code] [Code]
- Generative Adversarial Networks as Variational Training of Energy-Based Models (arXiv)
- Generative Adversarial Networks with Inverse Transformation Unit (arXiv)
- Generative Adversarial Parallelization (arXiv) [Code]
- Generative Adversarial Radial Basis Function Networks for One Shot Learning (arXiv)
- Generative Adversarial Emotional Networks (Paper)
- Generative Cooperative Net for Image Generation and Data Augmentation (arXiv)
- Generative Moment Matching Networks (arXiv) [Code]
- Generative Semantic Disambiguation with Generative GAN (arXiv)
- Generative GAN (arXiv)
- Good Self-Organized Learning that Regulars a Real GAN (arXiv)
- Gradient-based GAN optimization is locally stable (arXiv)
- How to Train Your GAN (arXiv)
- Image Quality Assessment Techniques Show Improved Training and Evaluation of Wasserstein Generative Adversarial Networks (arXiv)
- Improved Self-Organized Learning with GANs using Variational Inference (arXiv)
- Improved Techniques for Training GANs (arXiv) [Code]
- Improved Training of Wasserstein GANs (arXiv) [Code]
- InGAN: Interpretable Representation Learning by Information Unleashing Generative Adversarial Nets (arXiv) [Code]
- Inventing The Generator Of A Generative Adversarial Network (Paper)
- In The Net (DNL) The Adversarial Generator-Encoder Networks (arXiv)
- InGAN: Invariant to Real-Time Inference in GAN (arXiv)
- Learning in Implicit Generative Models (Paper)
- Learning Loss for Knowledge Distillation with Conditional Adversarial Networks (arXiv)
- Learning to Discriminate Cross-Domain Variations with Generative Adversarial Networks (arXiv) [Code]
- Learning Text-to-Image with the Parallelized GAN (arXiv) [Code]
- Least Squares Generative Adversarial Networks (arXiv) [Code]
- Linking Generative Adversarial Learning and Binary Classification (arXiv)
- Loss-GANs: Generative Adversarial Networks on Implicit Discrepancy (arXiv)
- LSGAN: Layer-wise Relaxed Generative Adversarial Networks for Image Generation (arXiv)
- MGGAN: Marginal Adaptation for Generative Adversarial Networks (arXiv) [Code]
- Variational-DeepFlow: Supervised Dynamic Generative Adversarial Networks (arXiv)
- Multi-Stage and Generative Recurrent Variational GAN (arXiv)
- NavaGAN: Patching Multi-Agent GANs (arXiv)
- NVD-GAN: Towards Deep Generative Modeling of Moment Matching Networks (arXiv)
- Noise Regularized Generative Adversarial Networks (arXiv) [Code]
- Multi-Agent Dynamic Generative Adversarial Networks (arXiv)
- Multi-Generator Generative Adversarial Nets (arXiv)
- Objective-Relaxed Generative Adversarial Networks (ORGAN) for Sequence Generation Models (arXiv)
- On Convergence and Stability of GANs (arXiv)
- On the effect of Batch Normalization and Weight Normalization in Generative Adversarial Networks (arXiv)
- On the Quantitative Analysis of Gradient-based Generative Models (arXiv)
- Splitting the Latent Space of Generative Networks (arXiv)
- Parameterizing Wasserstein GANs with a GAN (arXiv)
- PixelGAN: Wasserstein GANs (arXiv)
- SegGAN: Generative Adversarial Networks with Multi-scale U-Net for Medical Image Segmentation (arXiv)
- SelfGAN: Self-supervised Generative Adversarial Networks with Policy Gradient (arXiv)
- Single-to-Multi-Modal Adversarial Perturbations for Deep Networks (Paper)
- SelfGAN-GAN (arXiv)
- SelfGAN: Self-supervised Generative Adversarial Networks through Regularization (arXiv)
- Stacked Generative Adversarial Networks (arXiv)
- Statistics of Deep Generated Images (arXiv)
- Structured Generative Adversarial Networks (arXiv)
- Training Generative Adversarial Nets (arXiv)
- The Cramer Distance as a Solution to Wasserstein GAN's Problems (arXiv)

# GAN

- Problème compliqué
  - Nombreuse version avant d'obtenir de bons résultats
- Mais souvent impressionnant



DCGAN  
11/2015



EBGAN-PT  
9/2016



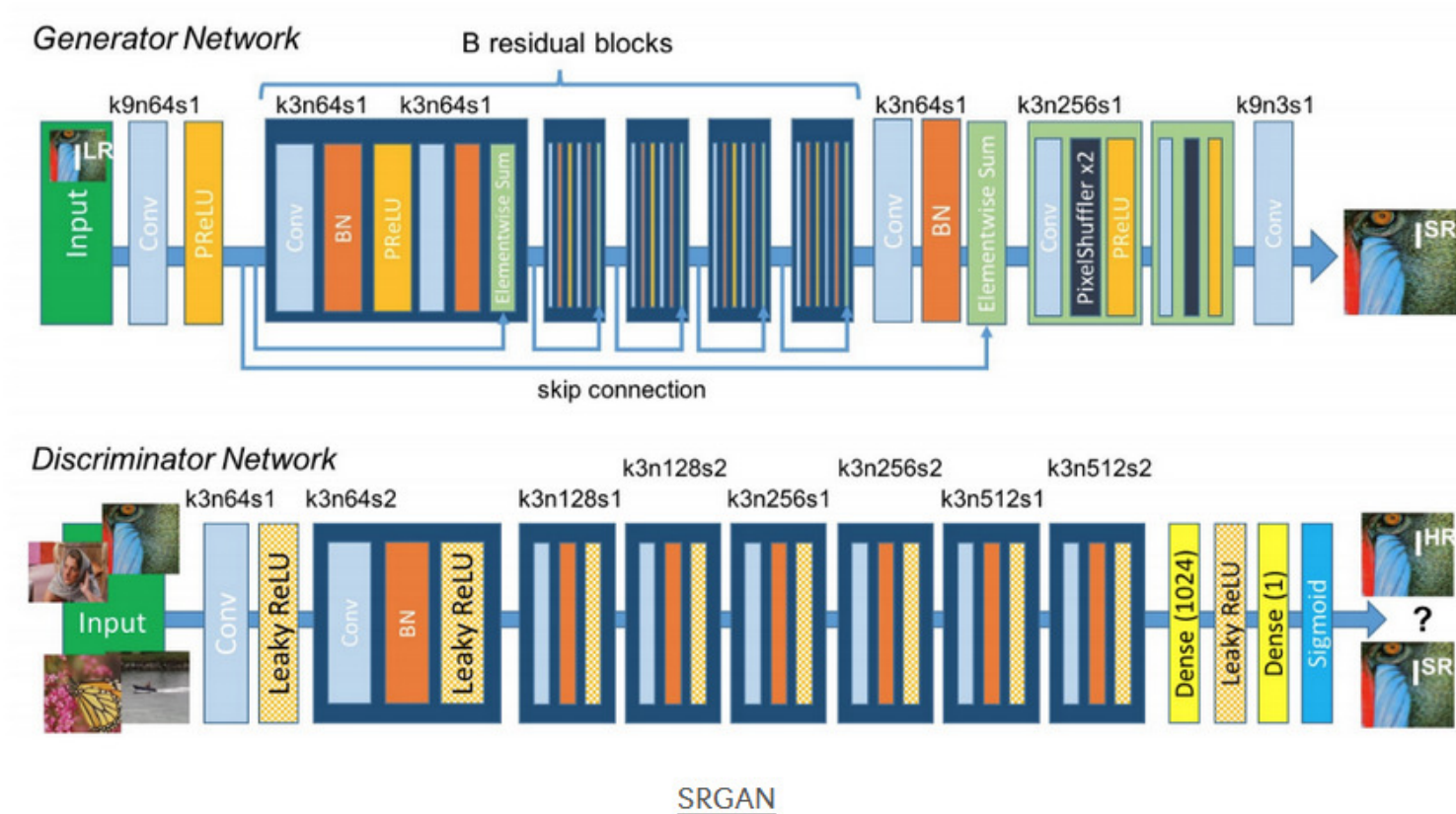
BEGAN  
3/2017  
128 x 128



Progressive GAN  
10/2017  
1024 x 1024

# Un exemple de GAN : SRGAN

- Super résolution



# Un exemple de GAN : SRGAN

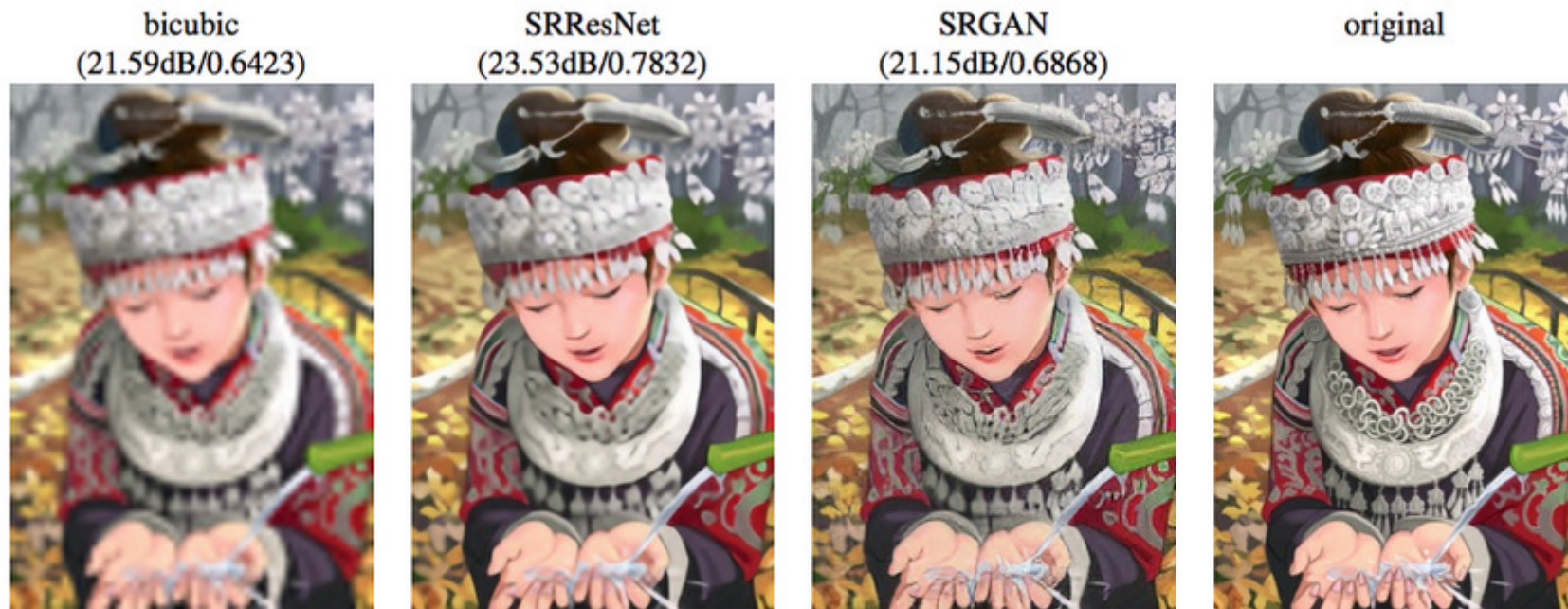
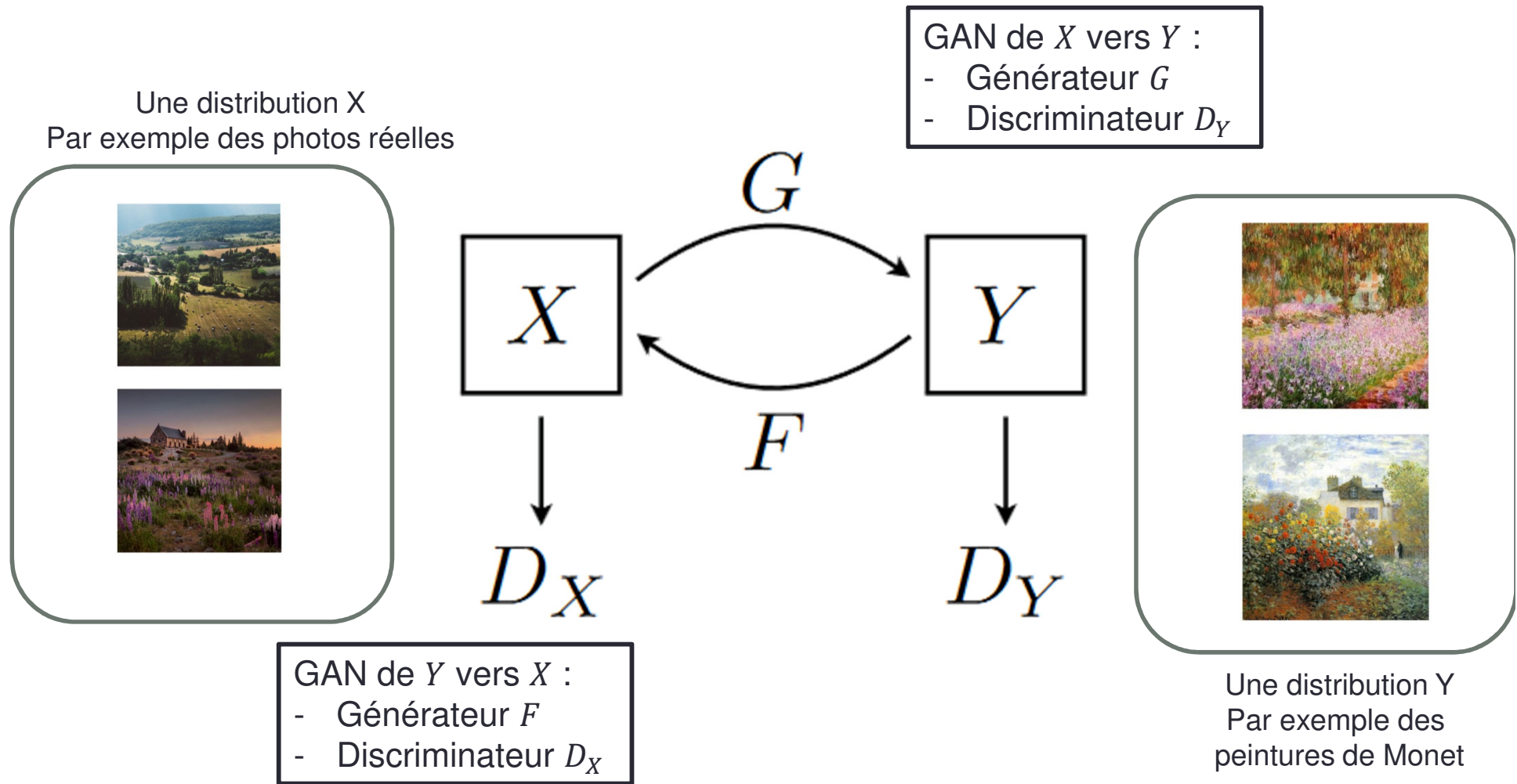


Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

SRGAN

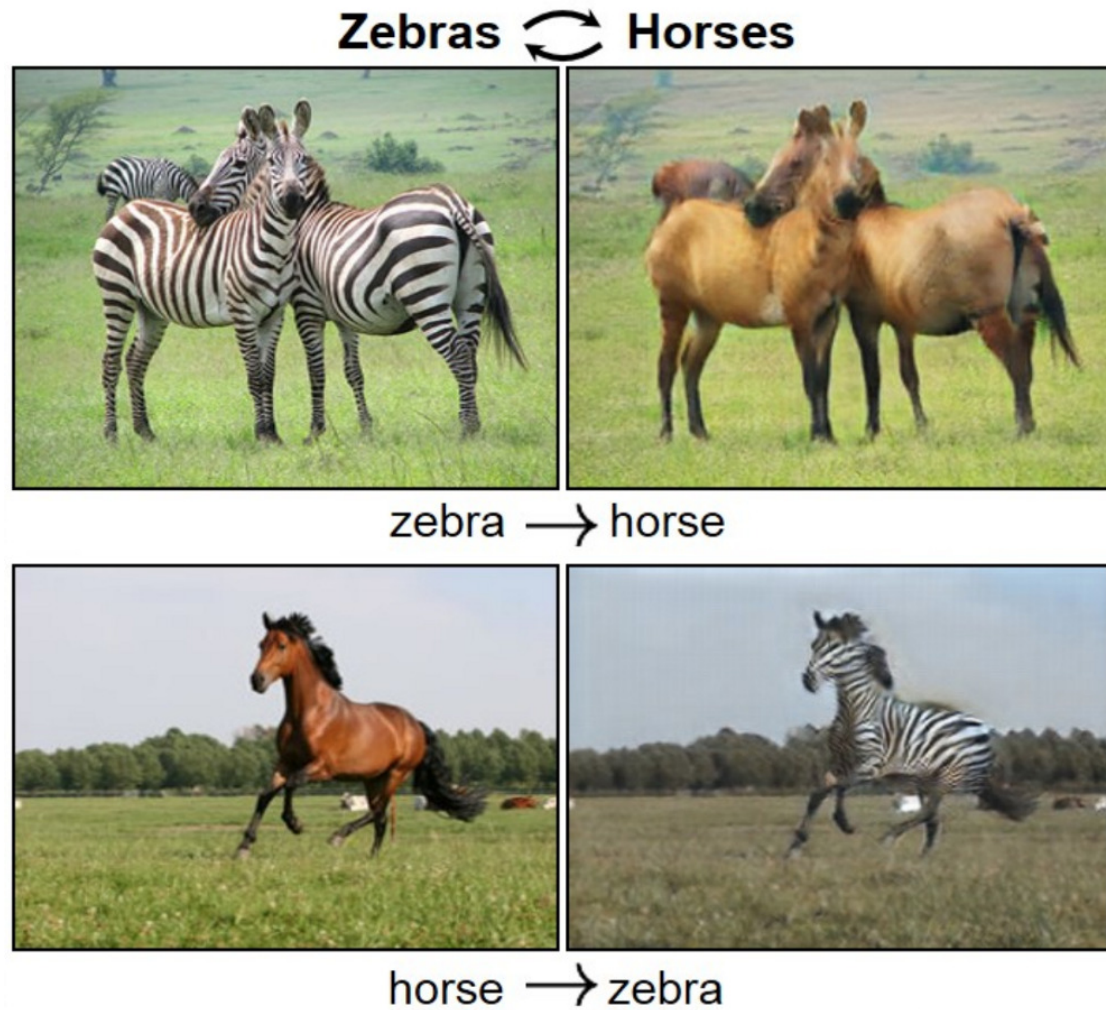


# Transfert entre distributions : CycleGAN



# Cycle GAN

- Distribution
  - Cheval
  - Zèbre





# GAN : un article à lire

GAN Dissection: Visualizing and Understanding Generative Adversarial Networks

David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B. Tenenbaum<sup>1</sup>, William T. Freeman, Antonio Torralba

<https://gandisection.csail.mit.edu/> (ESSAYER LA DEMO)



# GAN : les problèmes

Entraîner un GAN n'est pas simple

- **Mode collapse** : le générateur produit un nombre limité d'exemple
- **Diminished gradient** : le discriminateur deviant trop bon, trop rapidement et donc le generator n'apprend rien
- **Non-convergence** : les paramètres du modèle oscillent et ne converge jamais
- Plus généralement, un déséquilibre entre le generateur et le discriminateur provoque du sur apprentissage, hyper sensibilité au paramètres

# GAN : modification d'un visage

- StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation (2017)

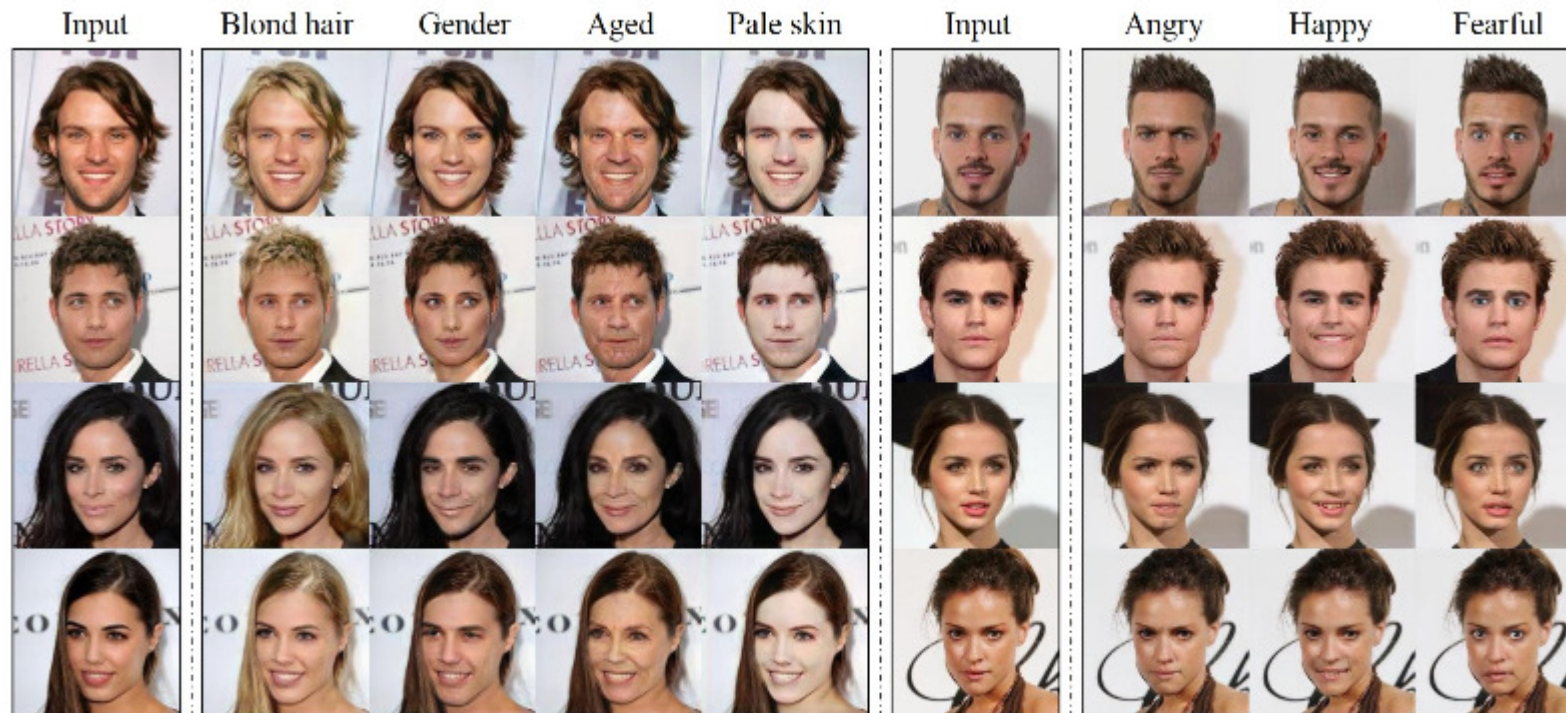


Figure 1. Multi-domain image-to-image translation results on the CelebA dataset via transferring knowledge learned from the RaFD dataset. The first and sixth columns show input images while the remaining columns are images generated by StarGAN. Note that the images are generated by a single generator network, and facial expression labels such as angry, happy, and fearful are from RaFD, not CelebA.

# GAN : modification d'un visage

- StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation (2017)

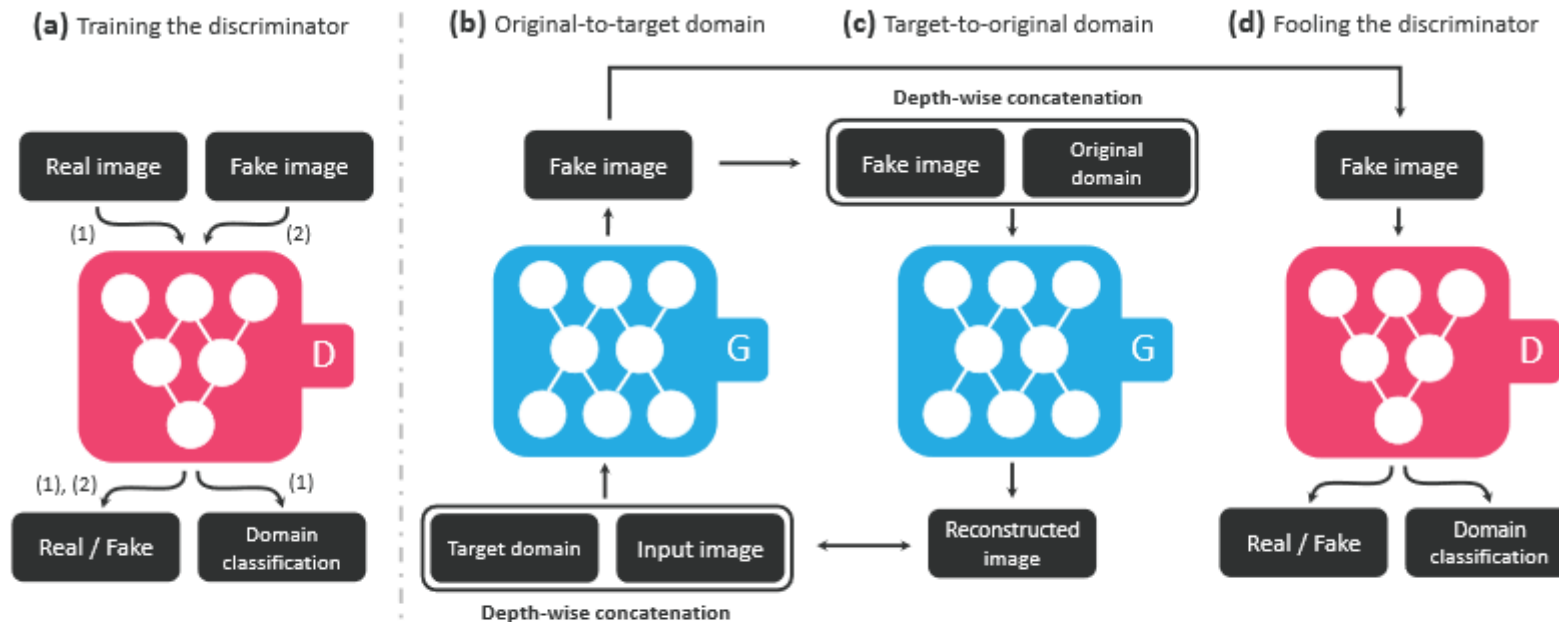


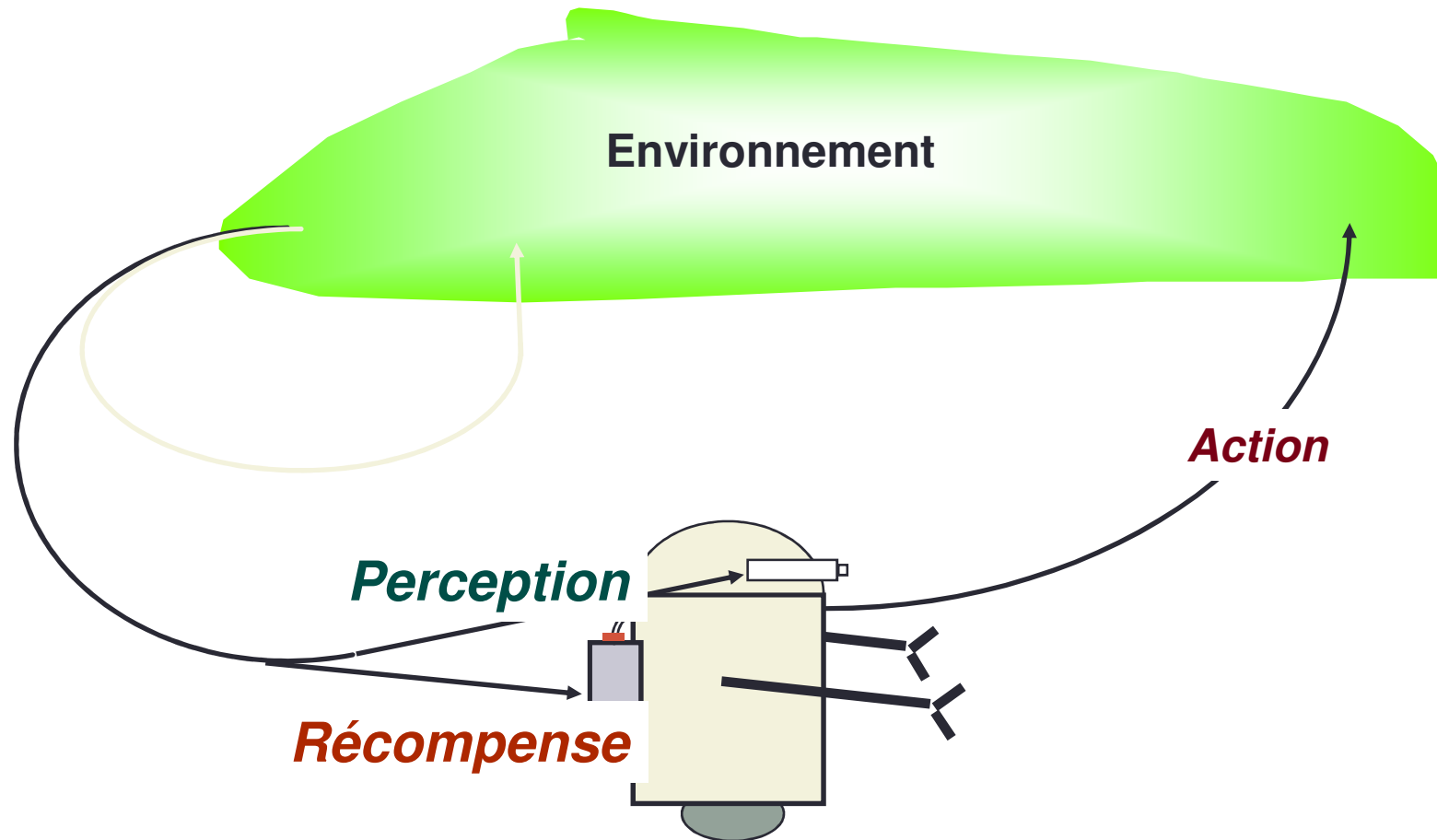
Figure 3. Overview of StarGAN, consisting of two modules, a discriminator  $D$  and a generator  $G$ . (a)  $D$  learns to distinguish between real and fake images and classify the real images to its corresponding domain. (b)  $G$  takes in as input both the image and target domain label and generates an fake image. The target domain label is spatially replicated and concatenated with the input image. (c)  $G$  tries to reconstruct the original image from the fake image given the original domain label. (d)  $G$  tries to generate images indistinguishable from real images and classifiable as target domain by  $D$ .



# APRENTISSAGE PAR RENFORCEMENT

---

# Apprentissage par renforcement



Problème non uniquement lié à la production d'un geste



# Apprentissage par renforcement

- Temps discret:  $t$
- États :  $s_t \in S$
- Actions :  $a_t \in A(s_t)$
- Récompenses :  $r_t \in R(s_t)$
- L'agent :  $s_t \rightarrow a_t$
- L'environnement :  $(s_t, a_t) \rightarrow s_{t+1}, r_{t+1}$
- Politique :  $\pi_t : S \rightarrow A$ 
  - Avec  $\pi_t(s, a) = \text{Prob que } a_t = a \text{ si } s_t = s$
- Les transitions et récompenses ne dépendent que de l'état et de l'action précédents : processus Markovien  $T, R$

# Apprentissage par renforcement

- *Politique* :

ensemble d'associations *situation* → *action* (une application)

Une simple table ... un algorithme de recherche intensive

Eventuellement stochastique

- *Fonction de renforcement* :

- Définit implicitement le but poursuivi

- Une fonction :  $(\text{état}, \text{action}) \rightarrow \text{récompense} \in \mathcal{R}$

- *Fonction d'évaluation*  $V(s)$  ou  $Q(s,a)$  :

- Récompense accumulée sur le long-terme

- *Modèle de l'environnement* :

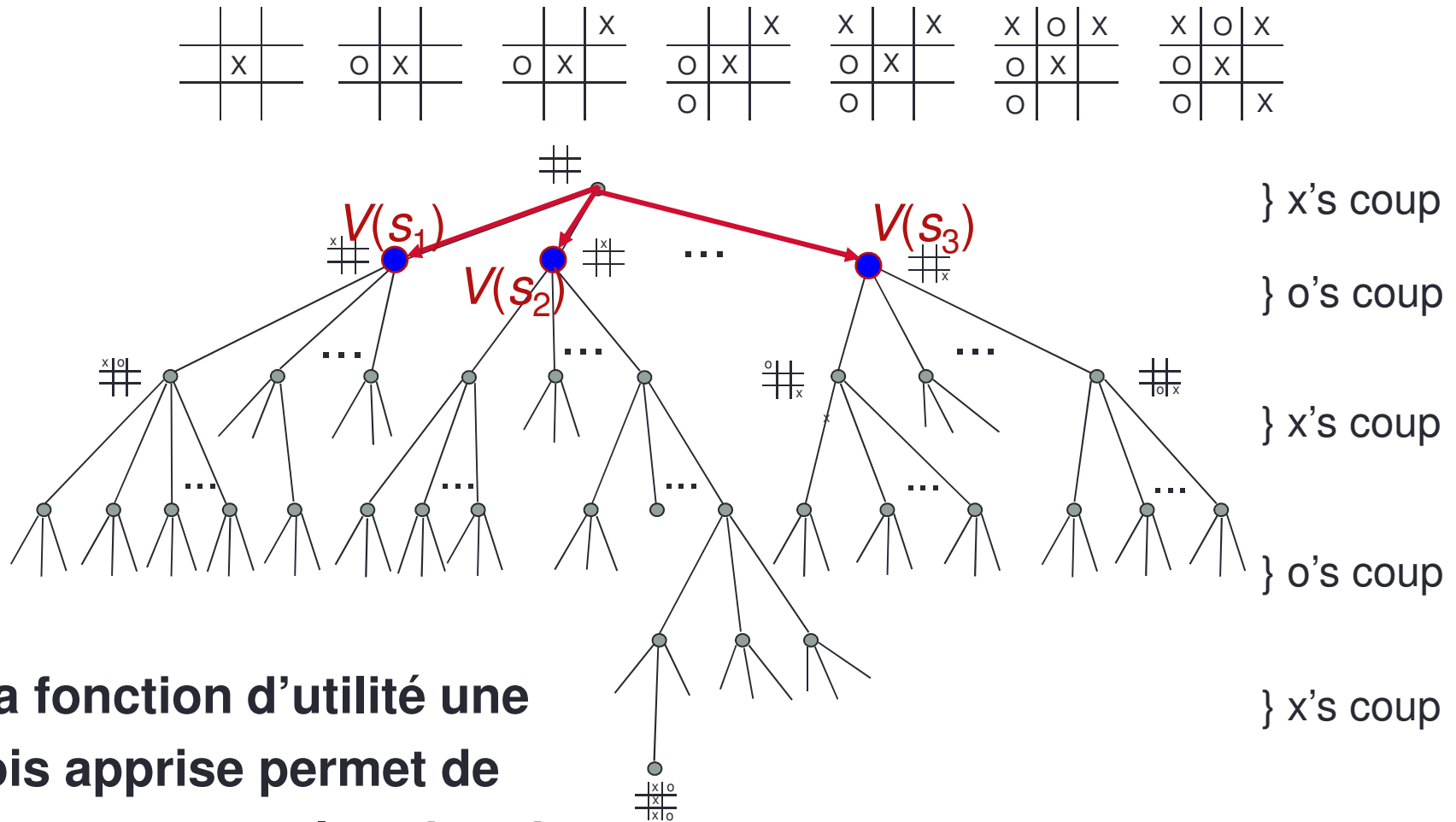
- Fonctions  $T$  et  $R$  :  $(\text{état}(t), \text{action}) \rightarrow (\text{état}(t+1), \text{récompense})$

# Apprentissage par renforcement

Principe :

- Choisir une action sans avoir besoin de faire une exploration (simulée) en avant
- Il faut donc disposer d'une fonction d'évaluation locale résumant une espérance de gain si l'on choisit cette action : *fonction d'utilité*
- Il faut apprendre cette fonction d'utilité : *apprentissage par renforcement*

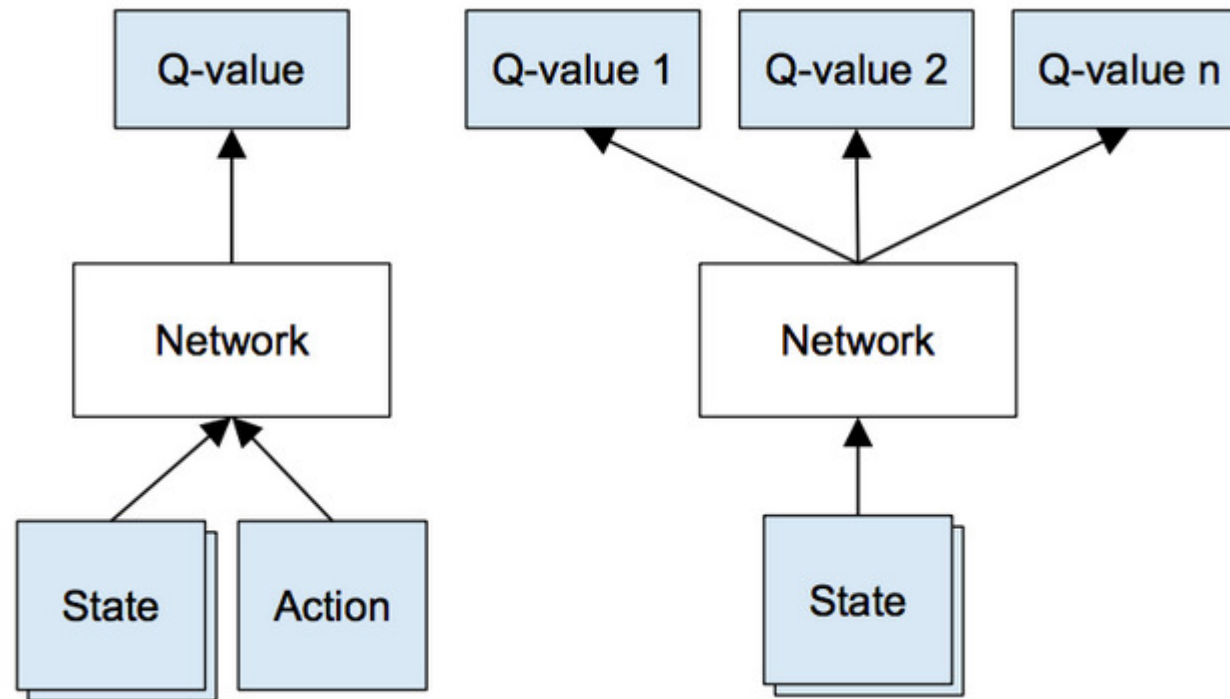
# Apprentissage par renforcement



**La fonction d'utilité une fois apprise permet de jouer sans exploration de l'arbre de jeu**

# Deep Q Learning

- Réseau de neurones bien adaptés au problème



# Learning to move

+vidéos

