



Computer Vision and Machine Learning (CVML)

Département Télécommunications Services & Usages



Département Télécommunications (TC)

A. Baskurt



Objectifs

- Analyser quantitativement et qualitativement les données
- Représenter les données dans un espace à N dimensions de façon à voir des tendances et à découvrir des structures
- Choisir la méthode de classification appropriée aux données et à la problématique
- Comprendre comment marche un réseau de neurones convolutif (CNN)
- Paramétrier un CNN afin d'optimiser ces performances en terme de classification

Sommaire

- Sommaire :
 - Données & Représentation des données
 - Analyse discriminante de données : classification
 - Classification non supervisée : indexation des images (K. Idrissi)
 - Classification supervisée : reconnaissance de visages (C. Garcia)
 - Réseaux de neurones (Neural Networks : NN)
 - Réseaux de neurones convolutifs (Convolutional NN : CNN)



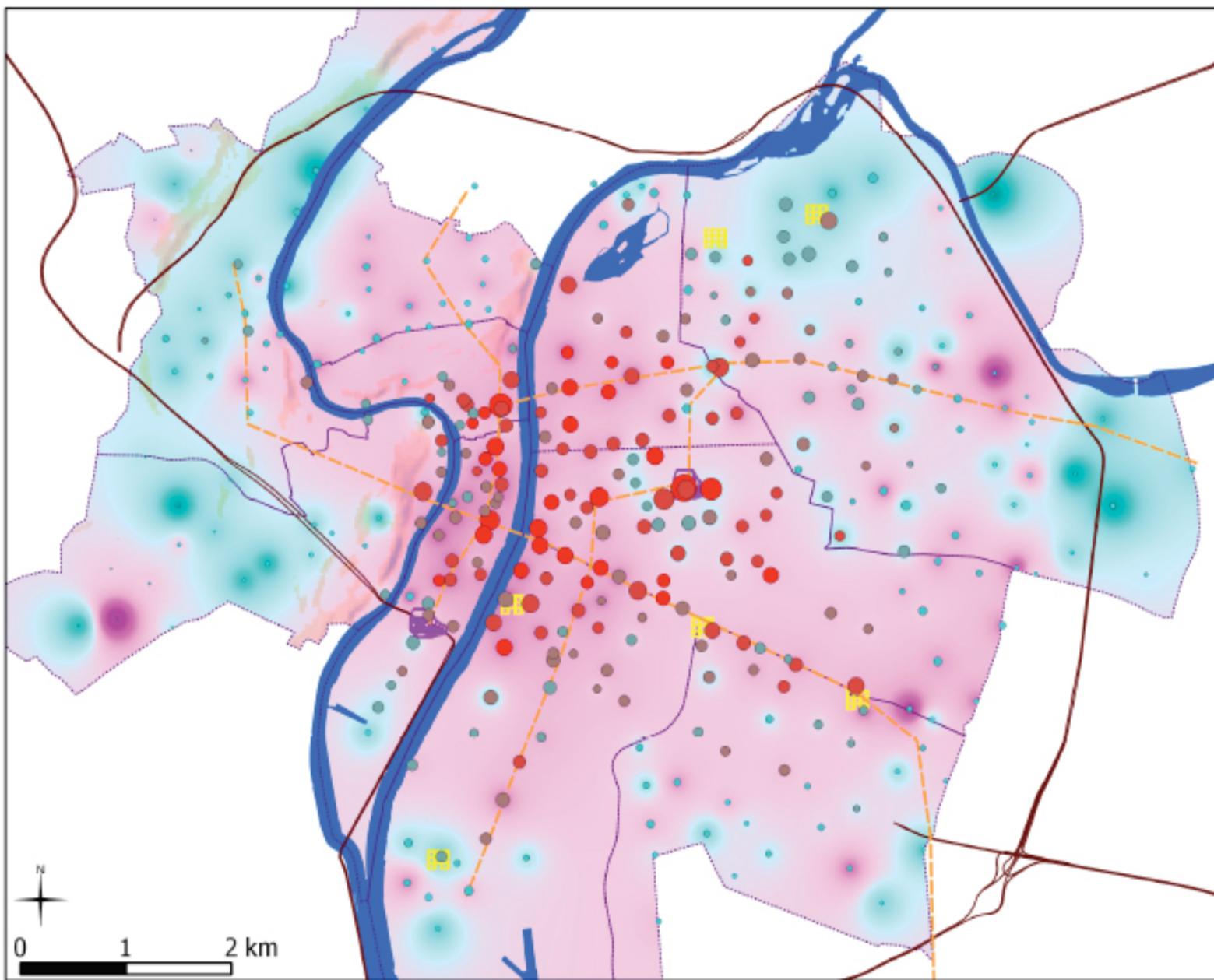
Données & Représentation des données

Données : matière première très convoitée

- Monde en réseau qui génère un déluge de données de plus en plus difficile à appréhender par les humains
- Collecte et préparation des données : débruitage, filtrage, enrichissement des données, réduction de dimension, intégration de données hétérogènes,...
- Analyse discriminante (classification) des données, visualisation, fouille de données (data mining), extraction de la connaissance
- Aide à la décision, détection, reconnaissance, suivi,...

Exemple : données urbaines

- Projet Vélover la ville (sociologues, informaticiens, géographes)
 - ☞ données de mouvements Vélo'V sur plusieurs années : spatialement et chronologiquement indexés et renseignés selon les caractéristiques de sexe et classe d'âge des usagers
 - ☞ données complétées par des enquêtes auprès des usagers (notion de données hétérogènes)
 - ☞ production de connaissance sur la pratique urbaine du VLS (vélo libre service) et ses usagers



Nombre moyen
de mouvements quotidiens

- 800
- 200
- 50
- 5

fréquence de rotation
(par borne et par jour)

- 0.08 - 1
- 1 - 2
- 2 - 3.5
- 3.5 - 5
- 5 - 7
- 7 - 14

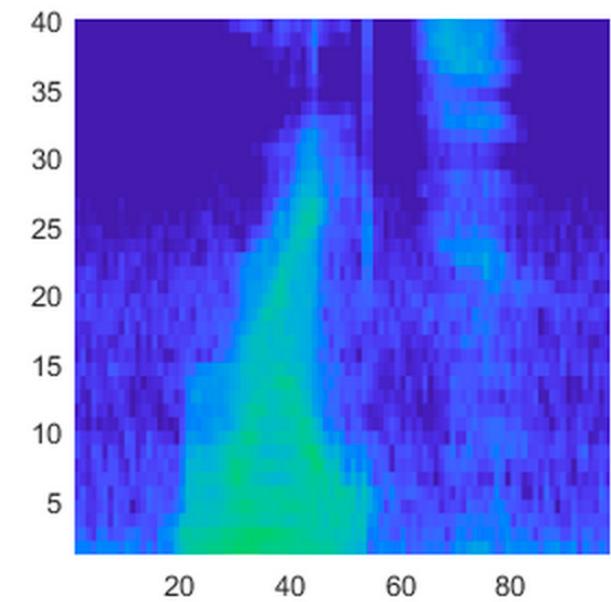
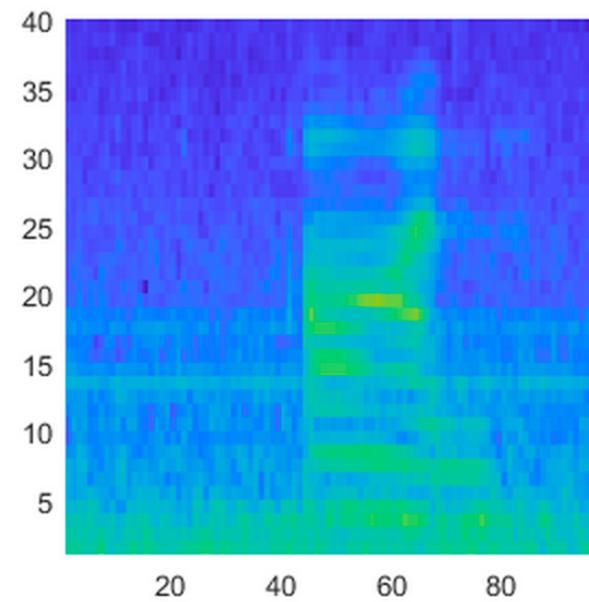
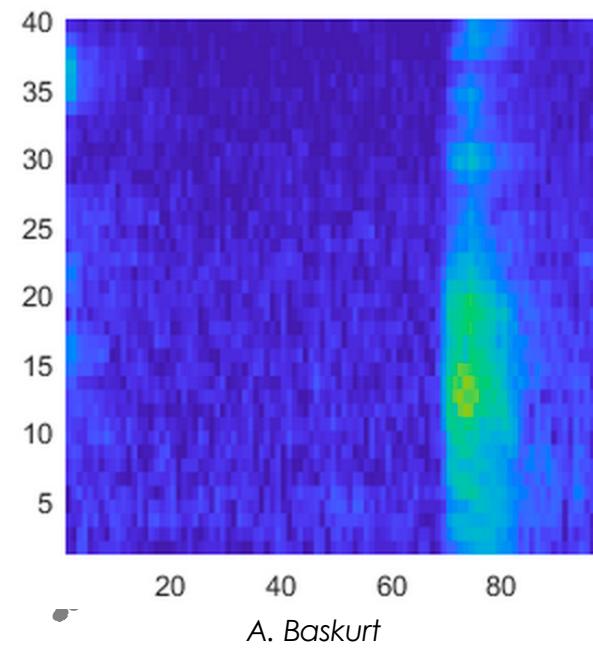
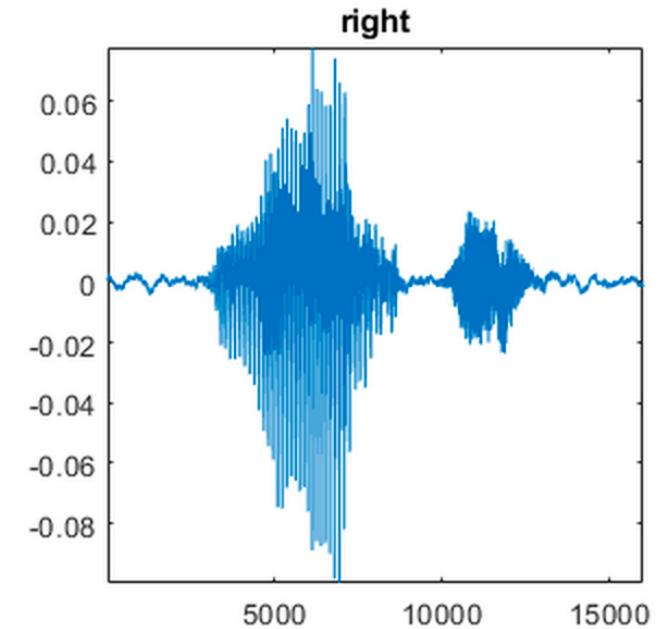
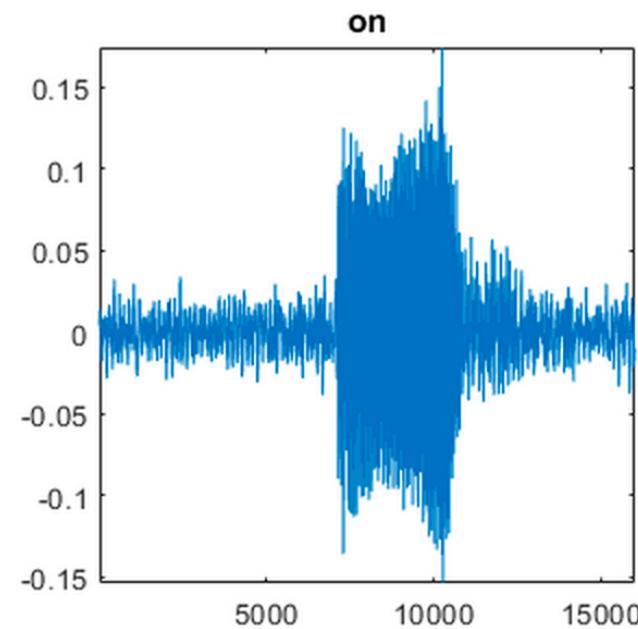
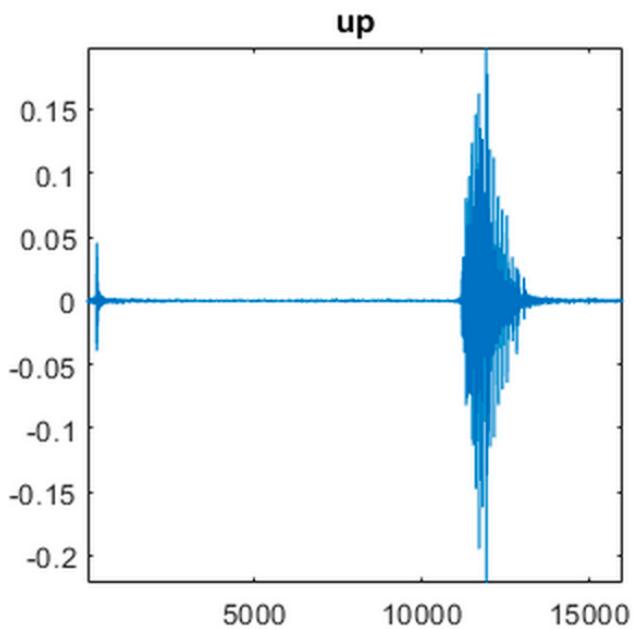
Part des hommes
parmi les usagers

- 0.45 - 0.50
- 0.50 - 0.54
- 0.54 - 0.58
- 0.58 - 0.62
- 0.62 - 0.66
- 0.66 - 0.70
- 0.70 - 0.74
- 0.74 - 0.78
- 0.78 - 0.83
- 0.83 - 0.91

- lignes de métro
- voies autoroutières
- gares principales
- poles universitaires
- limites communales et d'arrondissements
- fortes denivellations

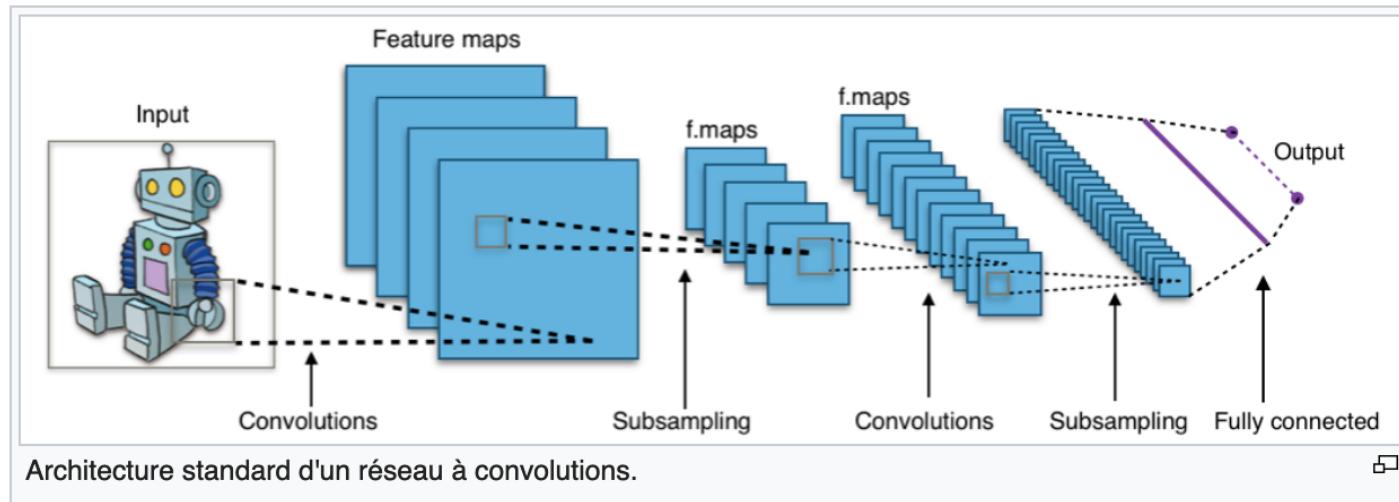
Exemple : données spectrogramme

- Reconnaître des mots à partir de l'analyse des spectrogrammes
- 10 classes : yes, no, up, down, left, right, on, off, stop, go
- construire une base de données avec des milliers d'exemples pour chaque mot
- calculer les spectrogrammes de tous les mots de la base



Exemple : données spectrogramme

- Utiliser un réseaux de neurones profond pour faire l'apprentissage sur une partie de la base



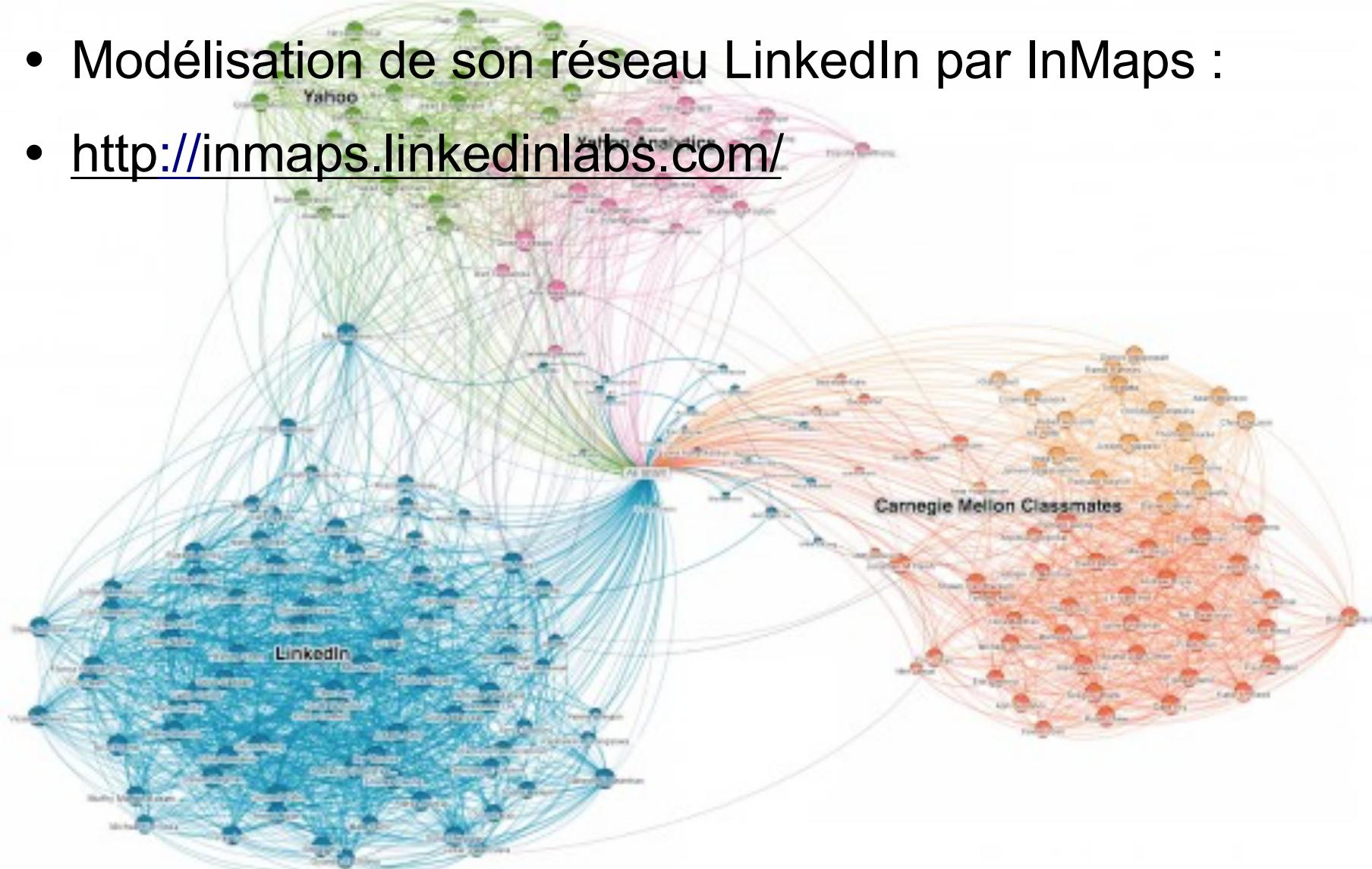
- Tester le réseau avec l'autre partie de la base
- Mesurer l'efficacité du réseau en % de mots bien classés

Confusion Matrix for Validation Data

True class	yes	252	1	2	1	3		1					1		96.6%	3.4%
	no		257		1			1			5	6			95.2%	4.8%
	up			247		1			4		2	4	2		95.0%	5.0%
	down		9		247			1	1		4	2			93.6%	6.4%
	left	1	1			243						1	1		98.4%	1.6%
	right		1	2		3	246	1				3			96.1%	3.9%
	on			4				243	3		1	4	2		94.6%	5.4%
	off			8				2	244			2			95.3%	4.7%
	stop			5	1	2				235		1	2		95.5%	4.5%
	go		3	3				2	3		241	8			92.7%	7.3%
	unknown	3	7	6	4	7	7	6	4	6	5	799	4		93.1%	6.9%
	background												400		100.0%	

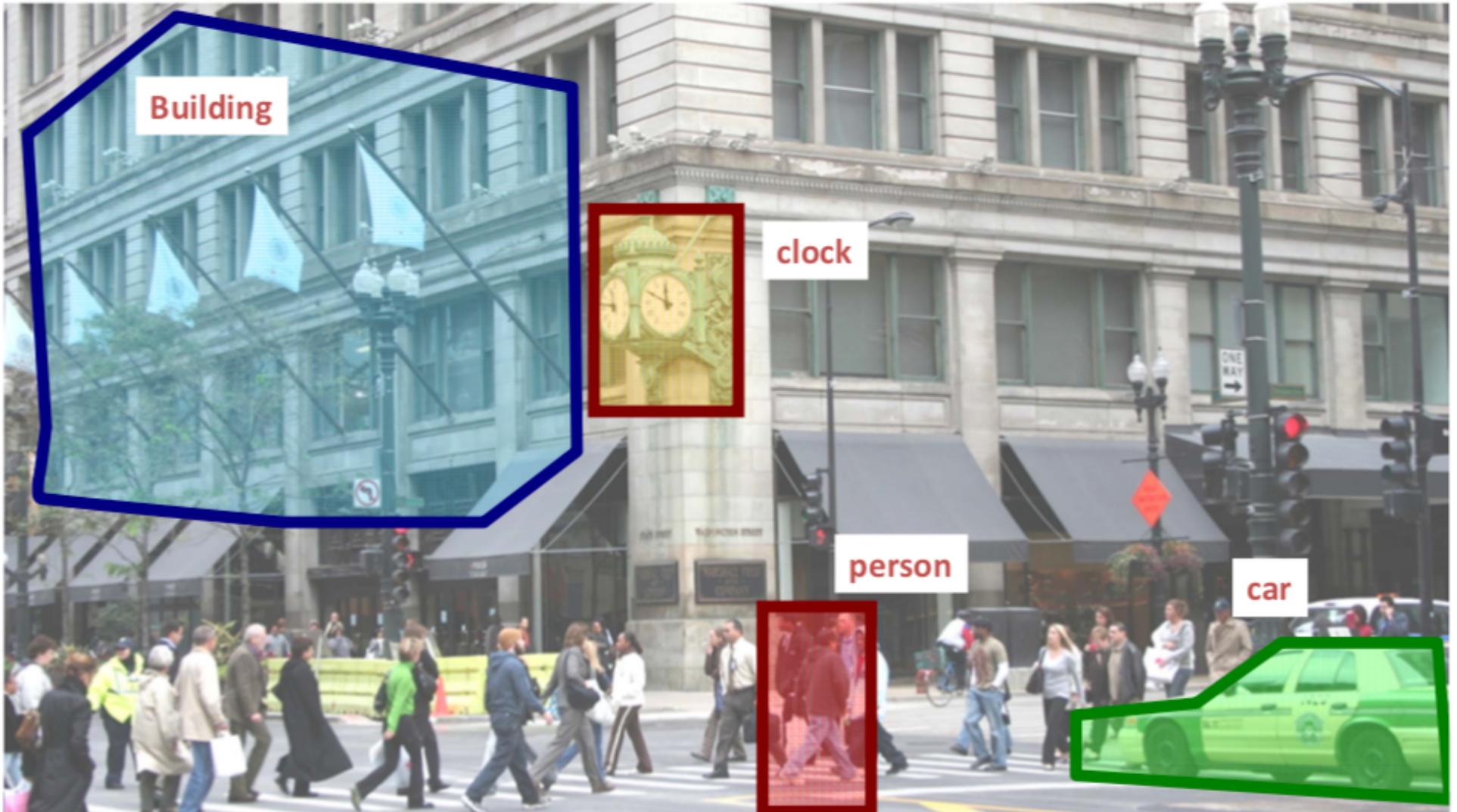
Exemple : données LinkedIn

- Modélisation de son réseau LinkedIn par InMaps :
- <http://inmaps.linkedinlabs.com/>



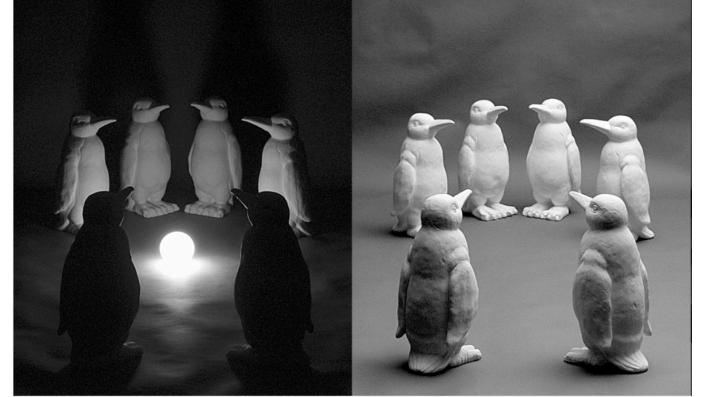
Exemple : données visuelles

Which object does this image contain? [where?]



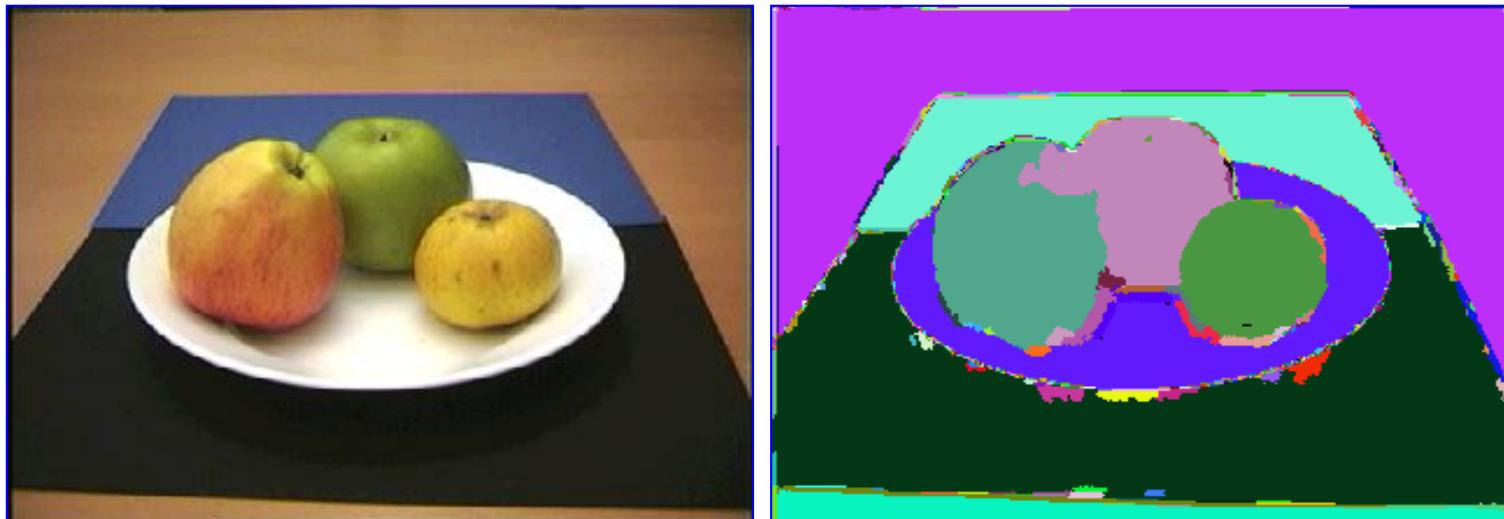
Reconnaissance visuelle

- Développer des méthodes capables de
 - ☞ Classifier des images ou vidéos
 - ☞ Déetecter et localiser des objets visuels
 - ☞ Estimer la sémantique
 - ☞ Classifier de activités humaines, actions
- C'est un challenge, car
 - ☞ Diversité des images, objets (30 000), actions
 - ☞ Variation de points de vue, de l'illumination, de l'échelle
 - ☞ Déformation, occlusion
 - ☞ Variabilité intra classe



Représentation des données visuelles

- Segmenter une image en région : la partitionner en zones homogènes en couleur, texture, forme, mouvement,...

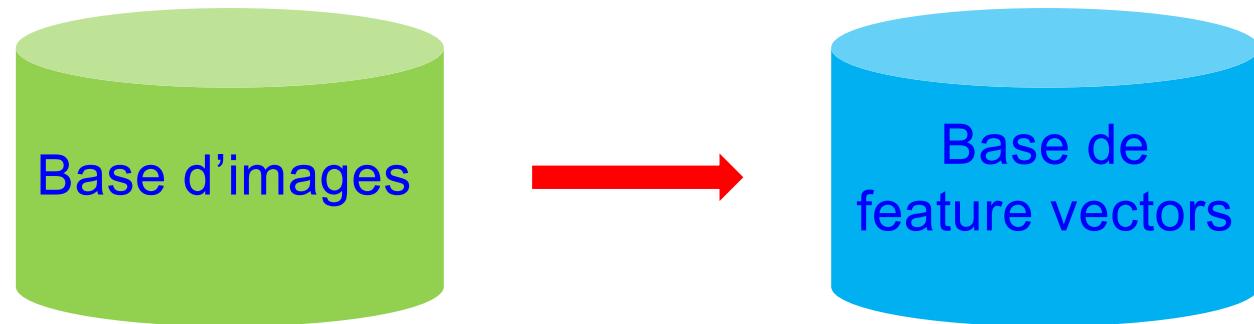


Représentation des données visuelles

- Image segmentée en régions
- Les régions peuvent être sans sémantique
- Les régions peuvent être des visages détectés
 - ☞ Reconnaissance de visages
- Les régions peuvent être des humains en mouvement
 - ☞ Détection et reconnaissance d'actions humaines

Représentation des données visuelles

- Calculer des caractéristiques mathématiques (features) dans chaque région afin de modéliser la couleur, la texture, la forme, le mouvement d'une région
 - ☞ texture rayée, jaune, circulaire, sans mouvement
- Construire un vecteur caractéristique (**feature vector**) : une sorte de signature de la région : **indexer la région**
- Crédit d'une base de vecteurs caractéristiques



Représentation des données visuelles

- Une fois les vecteurs calculés, on est dans un espace à N dimensions
 - ☞ Un vecteur : un point dans l'espace N dimensions (espace de Hilbert muni du produit scalaire, de la norme et de la distance)
 - ☞ Cet espace est appelé « **feature space** »
 - ☞ Une base de données : des millions de vecteurs : un nuage de points dans l'espace
- Rechercher des régions similaires :
 - ☞ Problème d'analyse et de classification de données
 - Utiliser les outils comme la distance ou la corrélation ou le produit scalaire entre deux feature vectors pour mesurer la ressemblance entre deux régions
 - Classer le nuage de points en K classes, puis classer la région inconnue dans une des classes pour reconnaître la région en tant qu'objet

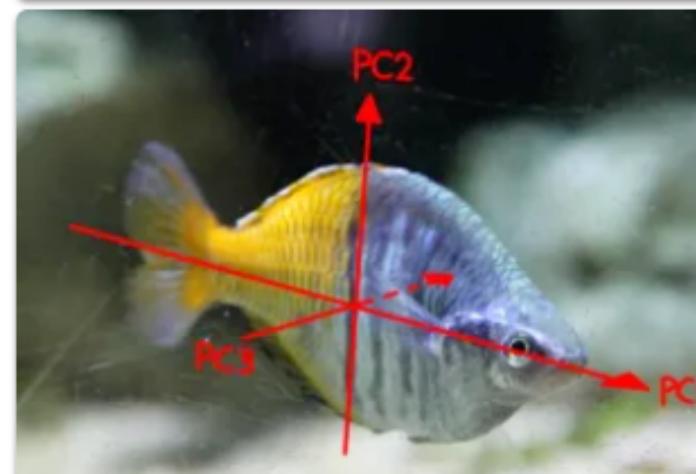
Représentation des données visuelles

- Diminution de la dimension de l'espace de représentation
 - ☞ Analyse en Composante Principale (ACP) :
 - mettre en évidence les Q axes principaux du nuage de points
 - Extraire ainsi l'information pertinente en passant de N dimensions à Q dimensions
 - Passer de N caractéristiques corrélées à Q caractéristiques décorrélées
 - Calcul de la matrice de corrélation M de N variables centrées réduites
 - Diagonalisation de M
 - Axes principaux : vecteurs propres associés aux Q les plus grandes valeurs propres ordonnées

Exemples ACP

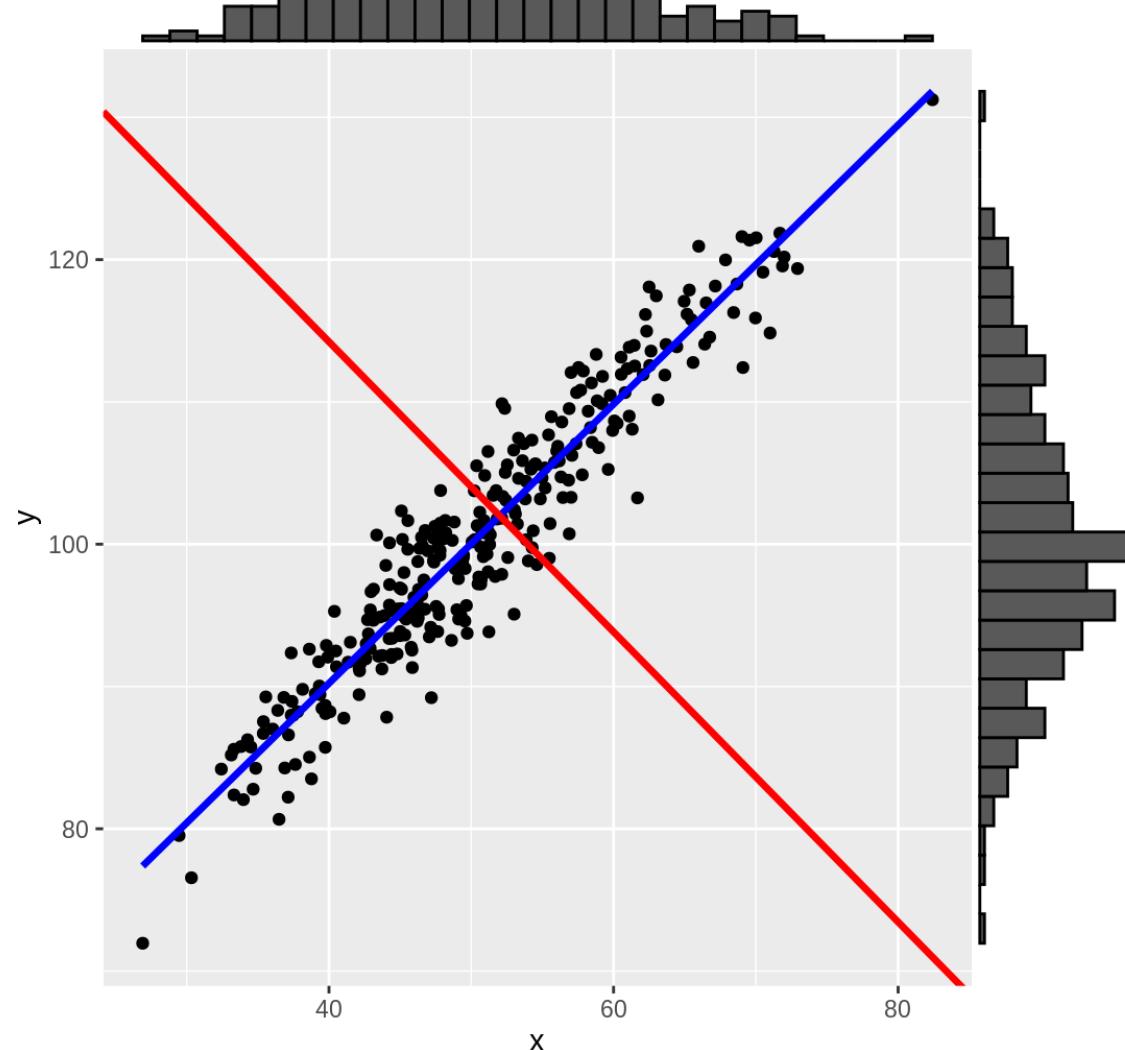
Si le poisson était un nuage de points et si on cherchait des axes permettant de mieux discriminer ce nuage ==> ACP permet de trouver les axes d'inertie du nuage

(*PC : principal component*)

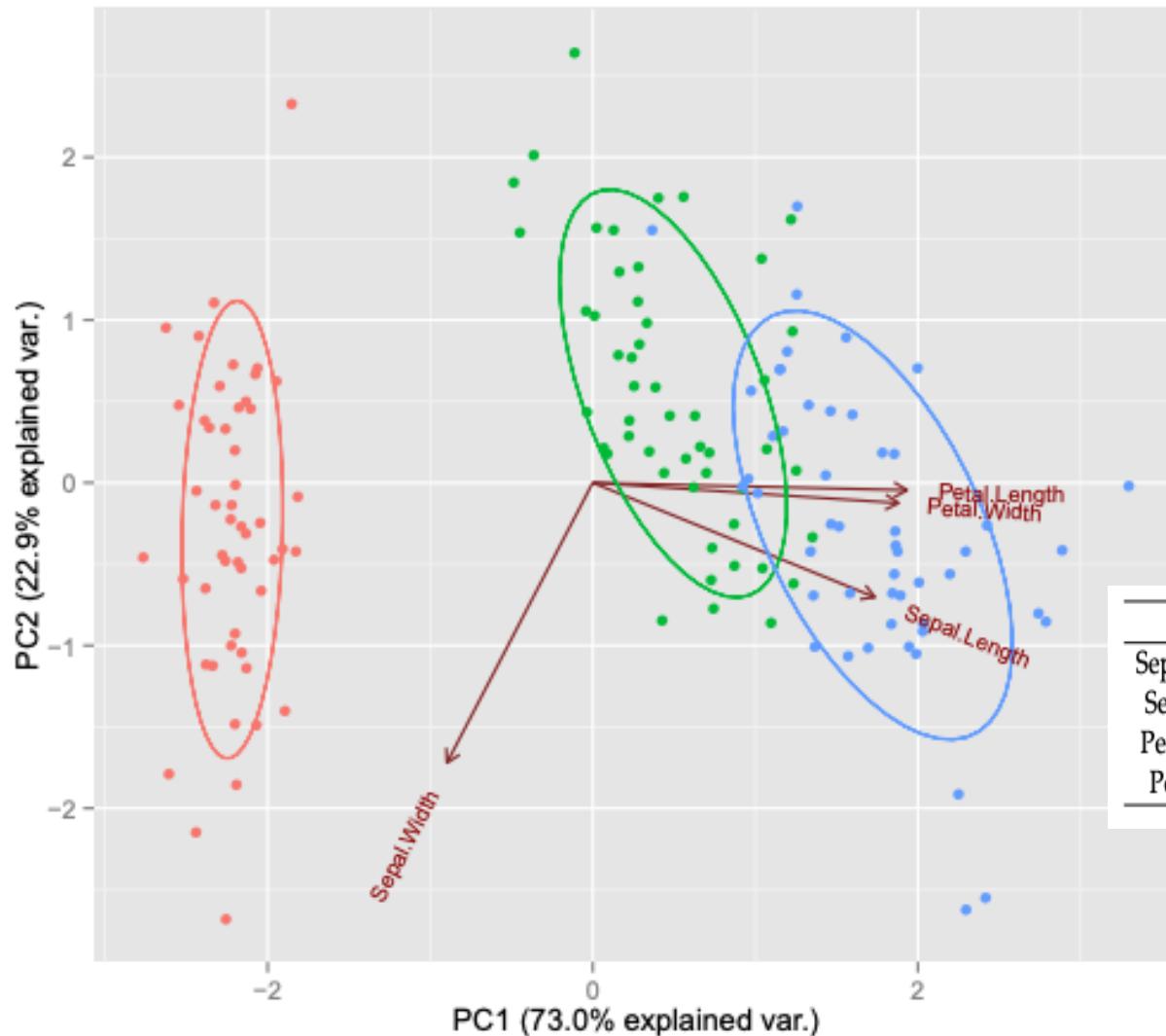


Exemples ACP

La variance en x et en y sont équivalentes, alors que l'axe principal bleu porte l'essentiel de la variance.



Exemples ACP



Données IRIS :
150 individus
4 dimensions
3 espèces : setosa,
versicolor et virginica

groups
setosa
versicolor
virginica

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
Sepal.Length	1.00	-0.12	0.87	0.82
Sepal.Width	-0.12	1.00	-0.43	-0.37
Petal.Length	0.87	-0.43	1.00	0.96
Petal.Width	0.82	-0.37	0.96	1.00

Matrice de corrélation à diagonaliser

Représentation des données visuelles

- Augmentation de la dimension de l'espace de représentation
- Pourquoi : si les caractéristiques mesurées sur les individus ne sont pas suffisantes (si la dimension N est petite)
- Comment passer à un espace plus riche (**feature space**) afin de mieux caractériser chaque individu et donc mieux réussir à le distinguer des autres individus ?



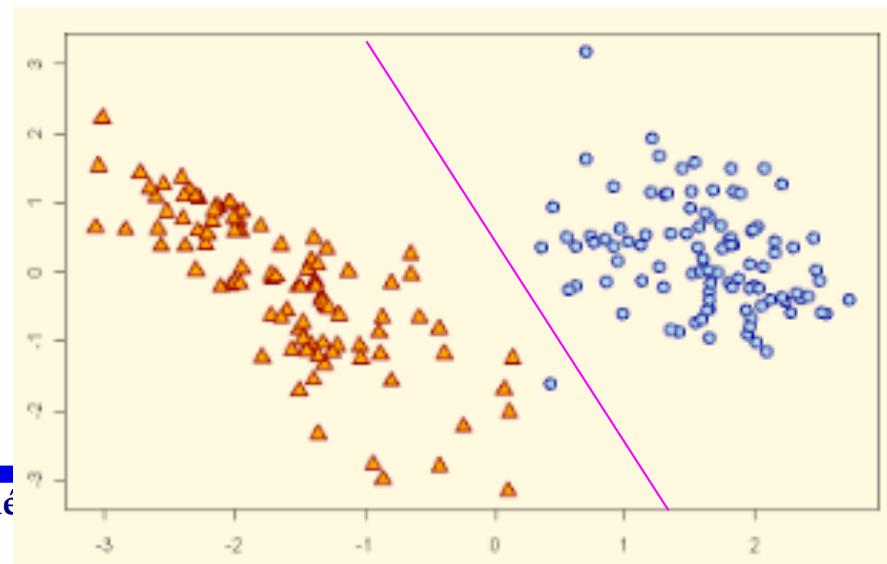
Figure 1: Example positive images for each attribute in our ontology. From left to right: *shorts, sandals, backpack, open-outerwear, sunglasses, skirt, carrying-object, v-neck, stripes, gender, headphones, short-hair, long-hair, logo, jeans*.



Analyse discriminante de données : classification

Analyse discriminante : classification

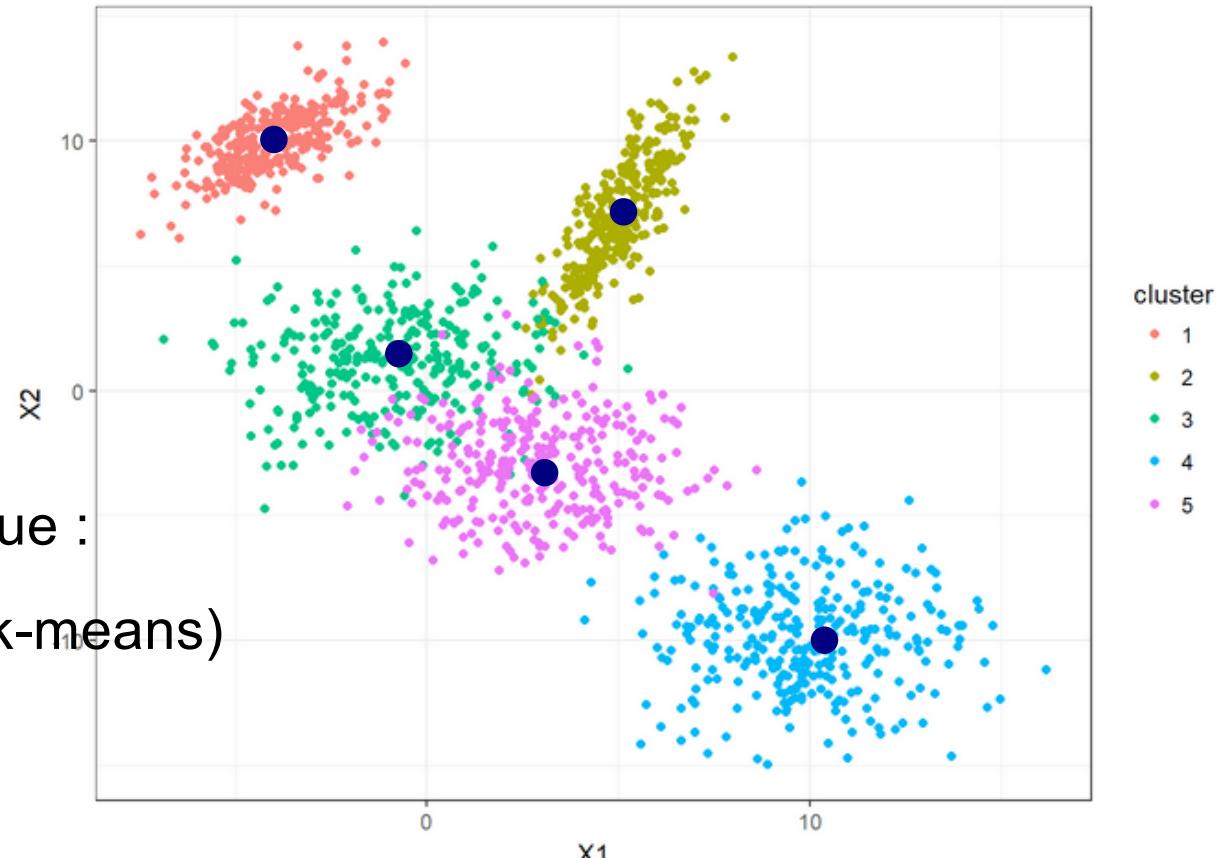
- Problème de classification : partition de l'espace
- Etant donné M individus connus par N caractéristiques, il s'agit d'extraire K classes ou groupes d'individus présentant des caractéristiques communes
- Comment partitionner ce nuage de points en K classes ?
- Méthodes supervisées et non supervisées



Classification : méthodes non supervisées

(présentée par K. Idrissi)

- Méthodes sans supervision
- Les individus ne sont pas labellisés :
 - ☞ on ne connaît pas la classe des individus
 - ☞ on ne connaît pas les classes et donc leur nombre K



Exemple de méthode très connue :

méthode des centres mobiles (k-means)

Classification : méthodes supervisées

(présentée par C. Garcia)

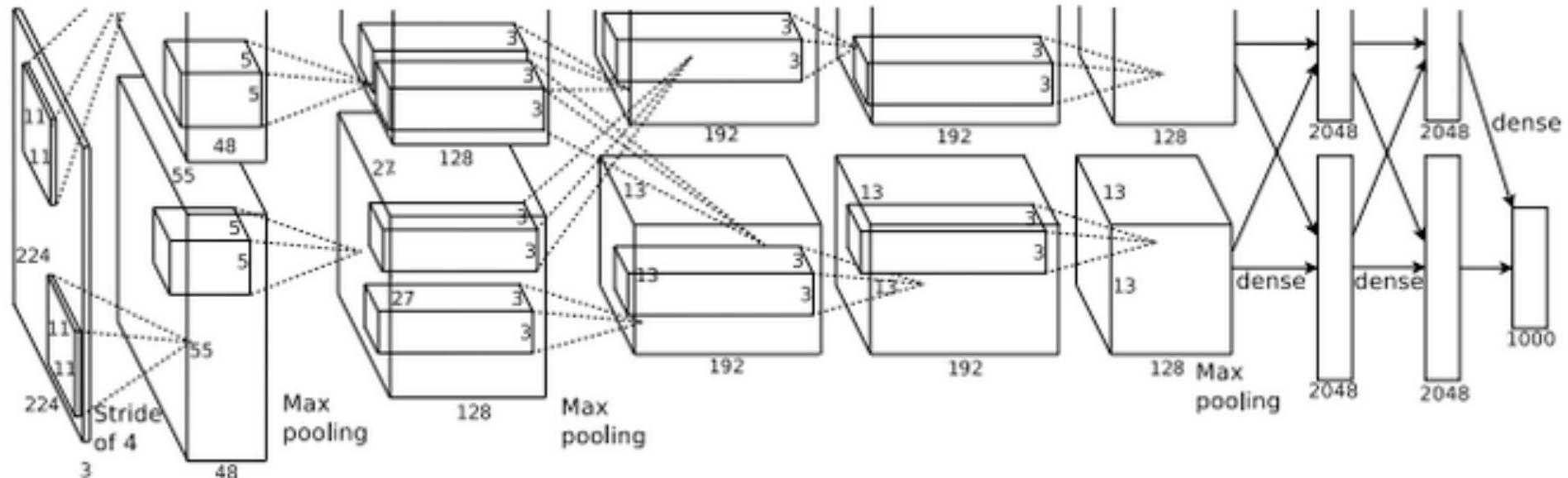
- On connaît les classes a priori
- On connaît la classe de chaque individu
 - ☞ individu étiqueté ou labellisé, par exemple par un expert
- Il s'agit d'apprendre à classer un nouvel individu parmi un ensemble de classes prédéfinies
- Première phase (hors ligne, dite d'apprentissage) : apprendre un modèle à partir des données étiquetées
- Seconde phase (en ligne, dite de test): prédire l'étiquette d'une nouvelle donnée, connaissant le modèle appris

Classification : méthodes supervisées

- Exemple : réseau de neurones profond
- Première phase : apprentissage

☞ Réseau AlexNet 2012 :

☞ Réseau appris sur la base ImageNet (> 1M Image, 1000 classes)



“ImageNet Classification with Deep Convolutional Neural Networks” by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, NIPS 2012, cité plus de 47000 fois depuis...

Classification : méthodes supervisées

- Seconde phase : classification d'un nouvel individu

			
mite mite black widow cockroach tick starfish	container ship container ship lifeboat amphibian fireboat drilling platform	motor scooter motor scooter go-kart moped bumper car golfcart	leopard leopard jaguar cheetah snow leopard Egyptian cat
			
grille convertible grille pickup beach wagon fire engine	mushroom agaric mushroom jelly fungus gill fungus dead-man's-fingers	cherry dalmatian grape elderberry ffordshire bullterrier currant	Madagascar cat squirrel monkey spider monkey titi indri howler monkey
			29

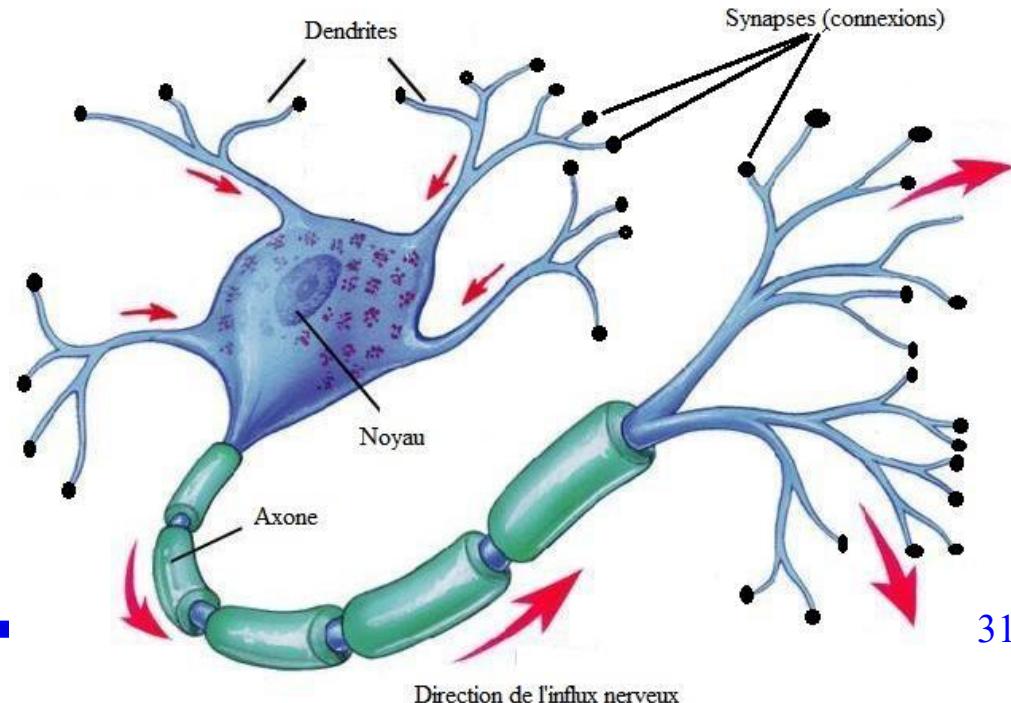


Réseaux de neurones

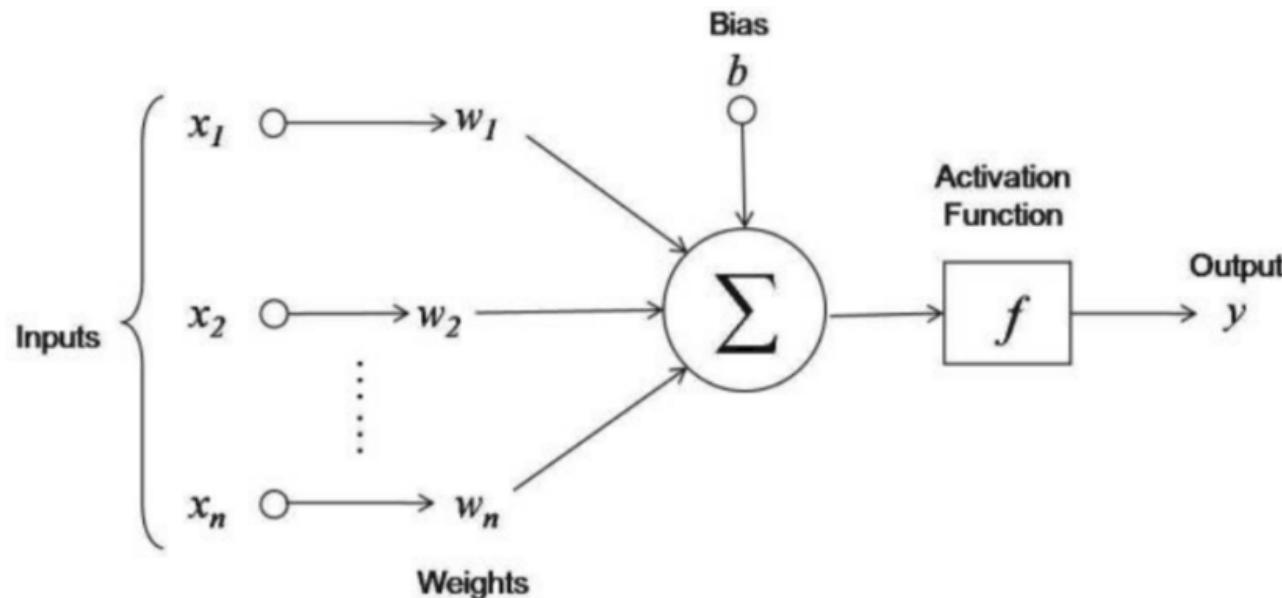
(Neural Networks : NN)

Historique des réseaux de neurones

- Le premier modèle mathématique et informatique du neurone biologique proposé par W. McCulloch et W. Pitts en 1943
 - ☞ Plusieurs entrées qui stimulent le neurone (dendrites précédés de synapses)
 - ☞ Synapses : points de contact avec plus ou moins de sensibilité
 - ☞ Une sortie via l'axone et synapses pour attaquer le neurone suivant
 - ☞ Un neurone réagit si la somme pondérée des stimulations dépasse un certain seuil

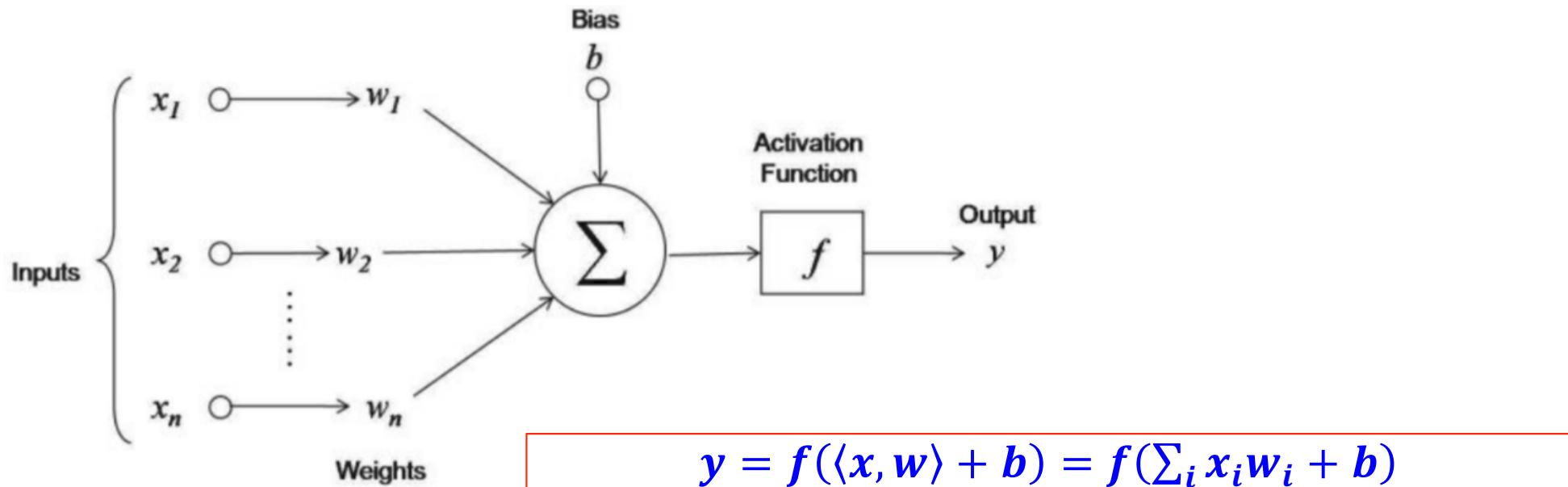


Modélisation mathématique d'un neurone



- x_i : les données (signaux arrivant sur les dendrites via les synapses)
- w_i : les poids associés à chaque entrée (sensibilité de chaque synapse)
- b : biais (seuil d'activation du neurone)
- f : fonction d'activation du neurone
- y : la sortie (via l'axone)

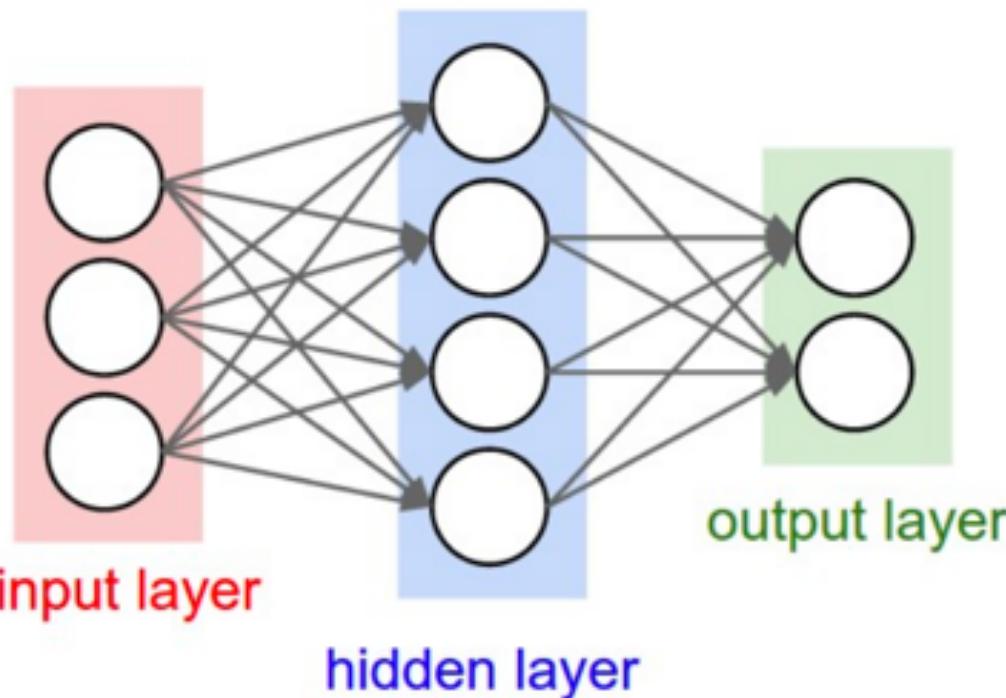
Modélisation mathématique d'un neurone



- Un produit scalaire entre les entrées x_i et les poids w_i ;
- Si le résultat du produit scalaire est plus grand que le seuil d'activation, alors on aura une sortie activée avec la valeur y
- Tout l'enjeu des NN : apprendre les poids w_i

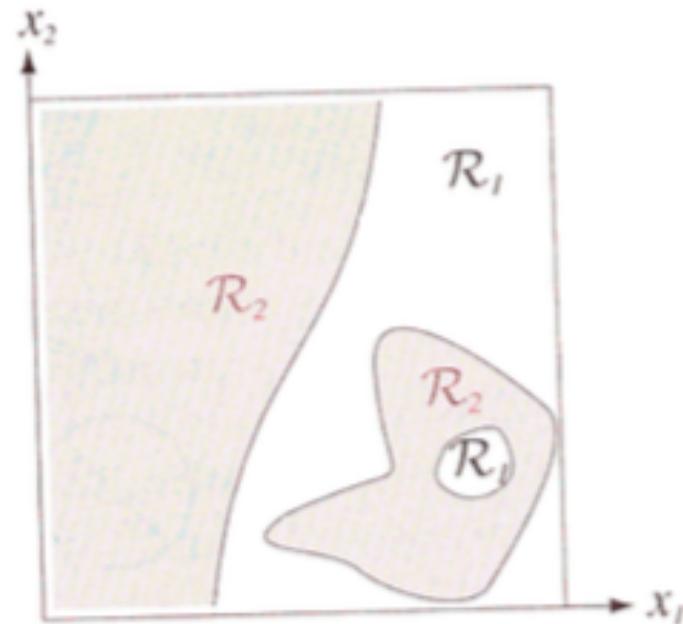
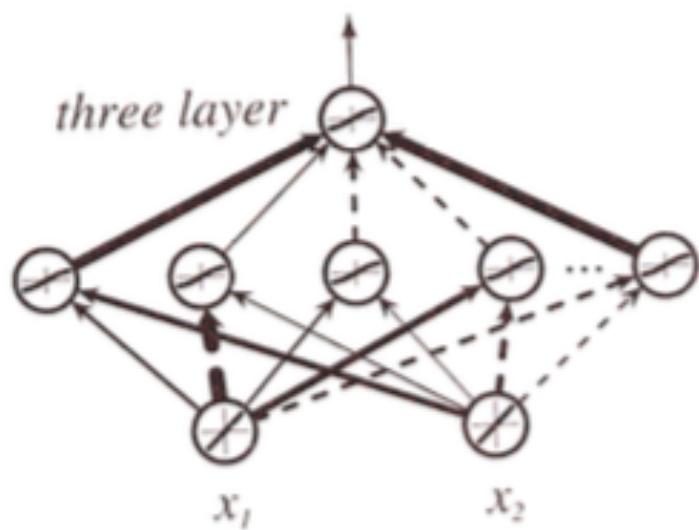
Perceptron Multi-Couches (D. Rumelhart, 1986)

- Entrée : les données
- Une couche de sortie : les classes : poids à apprendre
- Une couche fully connected (FC) : poids à apprendre
 - ☞ les entrées sont connectées à tous les neurones de la couche cachée
 - ☞ les sorties sont connectées à tous les neurones de la couche cachée



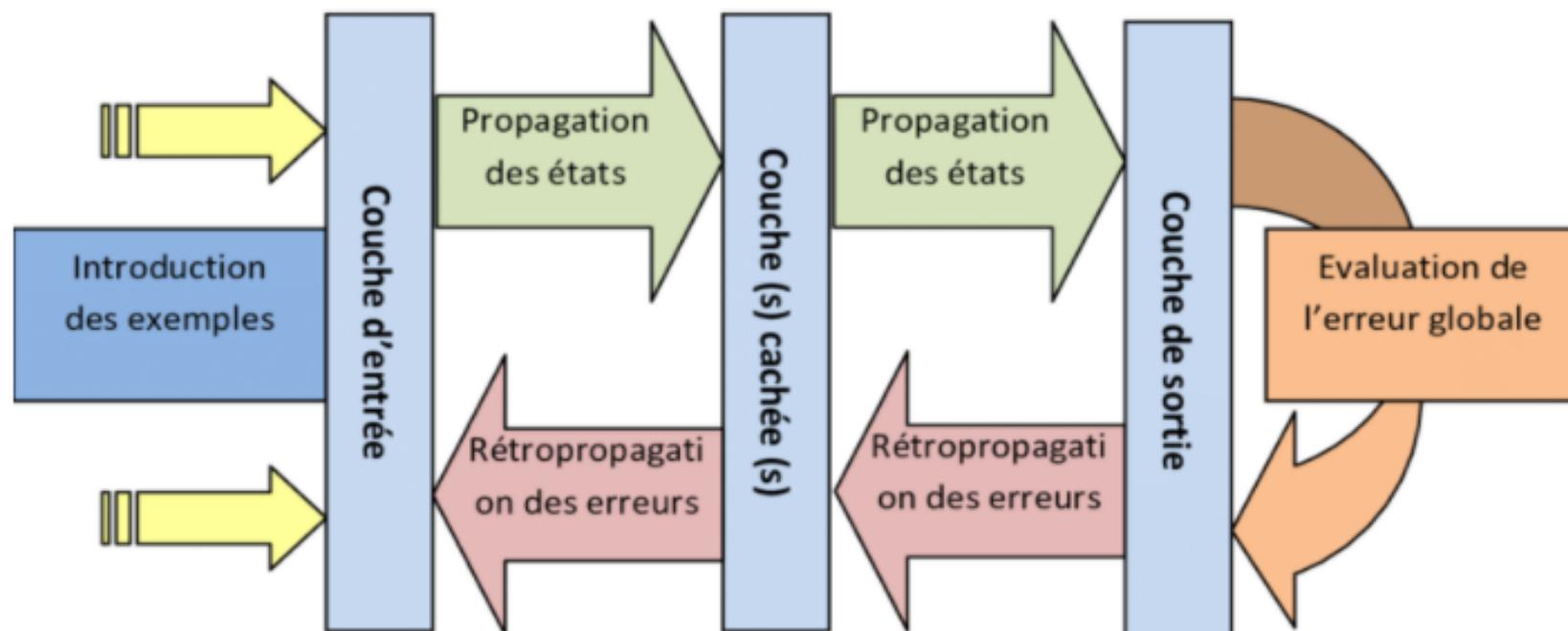
Perceptron Multi-Couches (D. Rumelhart, 1986)

- Utilisation de fonctions d'activation non linéaires
 - ☞ construction de frontières de décision complexes et non linéaires



Perceptron Multi-Couches (D. Rumelhart, 1986)

- Pour chaque entrée x , on connaît la ou les sorties y (apprentissage supervisé)
- Minimisation de l'erreur quadratique $d_2(y, y_{\text{estimée}})$
- Rétro propagation de l'erreur pour mettre à jour les poids w_i

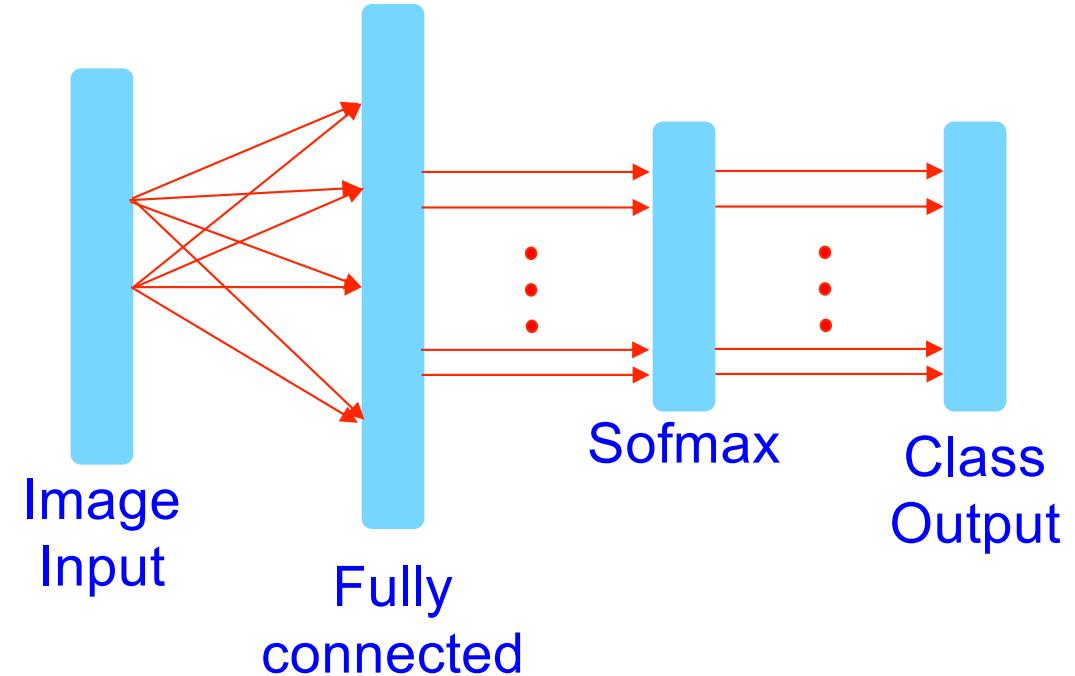


Comment on va travailler ?



- On part d'un exemple de réseau sur Matlab
- On explique chacune des couches utilisées en détail
- On fait l'apprentissage du réseau
- On analyse les performances du réseau
- Puis on joue sur les paramètres pour améliorer le réseau
- Commençons par un simple NN

Exercice 1 : NN à 1 couche

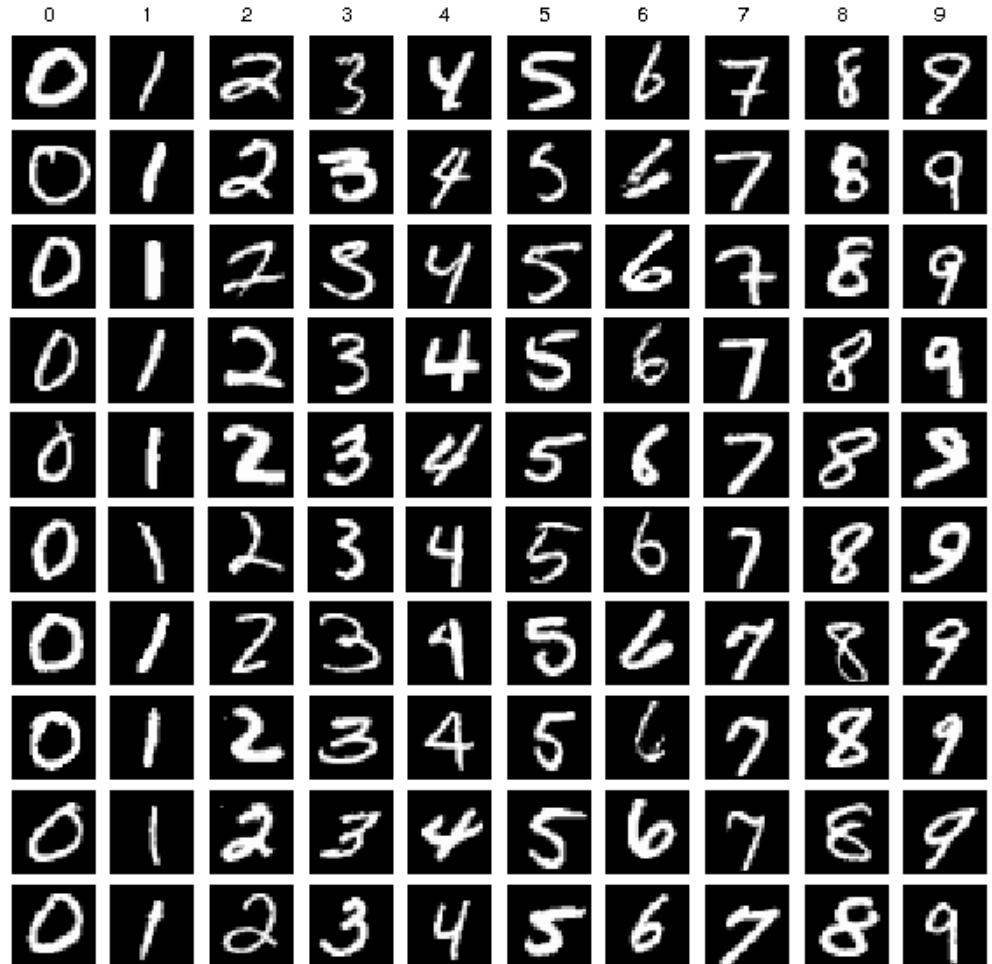


Chaque pixel est un neurone
de la couche d'entrée

- Fichiers matlab à utiliser :
 - ☞ MNIST_database.m ; fonction `prepareData` ; Simple_NN_pour_MNIST.m
 - ☞ Attention : pour matlab, tout est appelé couche : ici, 4 touches
 - ☞ Une vraie couche : c'est lorsqu'on a des poids associés qu'il faut apprendre

Base de données

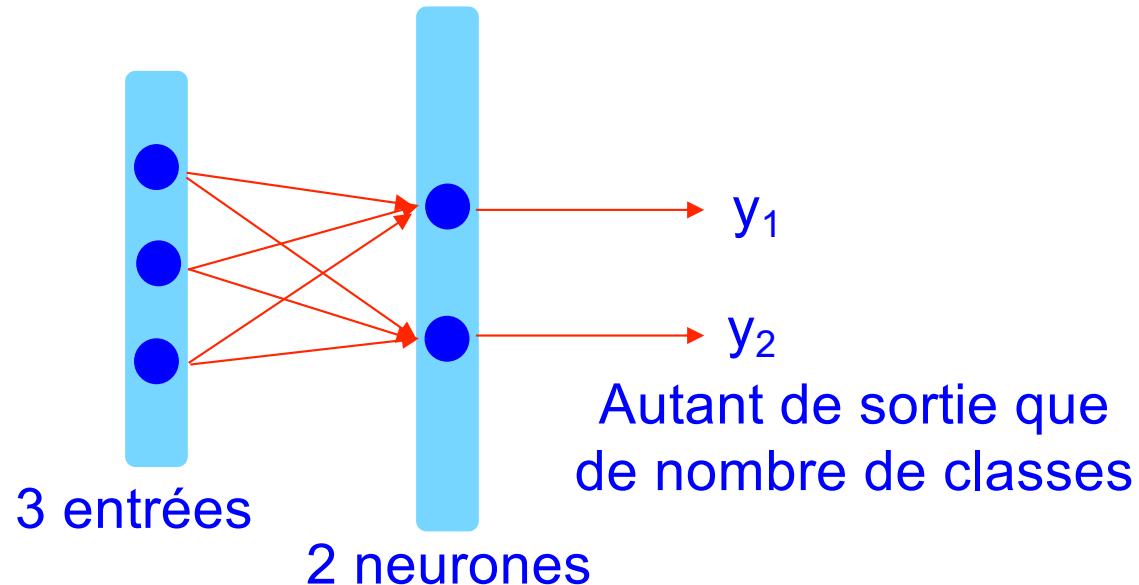
- Dataset MNIST :
 - ☞ Une base de données de chiffres écrits à la main
 - ☞ Image niveaux de gris
 - ☞ Base d'apprentissage : 60000 images 28x28 pixels + labels
 - ☞ Base de test : 10000 images 28x28 + labels



Input Layer : comment on formate des données « image » ?

- Matrice 4D pour stocker les images :
 - ☞ ligne image, colonne image, no canal ou composante, no de l'image
- Vecteur pour mettre les labels «classe de l'image»
- Images normalisées avant l'apprentissage (comme pour la reconnaissance de visages) :
 - ☞ Calculer la moyenne des images de la base d'apprentissage : $E(X)$
 - ☞ Centrer les images : $X - E(X)$ (zerocenter normalization sur matlab)
 - ☞ Normaliser en divisant par l'écart-type : $\frac{X-E(X)}{\sigma(X)}$
 - ☞ On homogénéise ainsi les données, car il ne faut pas envoyer des activités hétérogènes au réseau ; il faut que chaque image contribue de manière homogène à la variation des poids w_i à apprendre

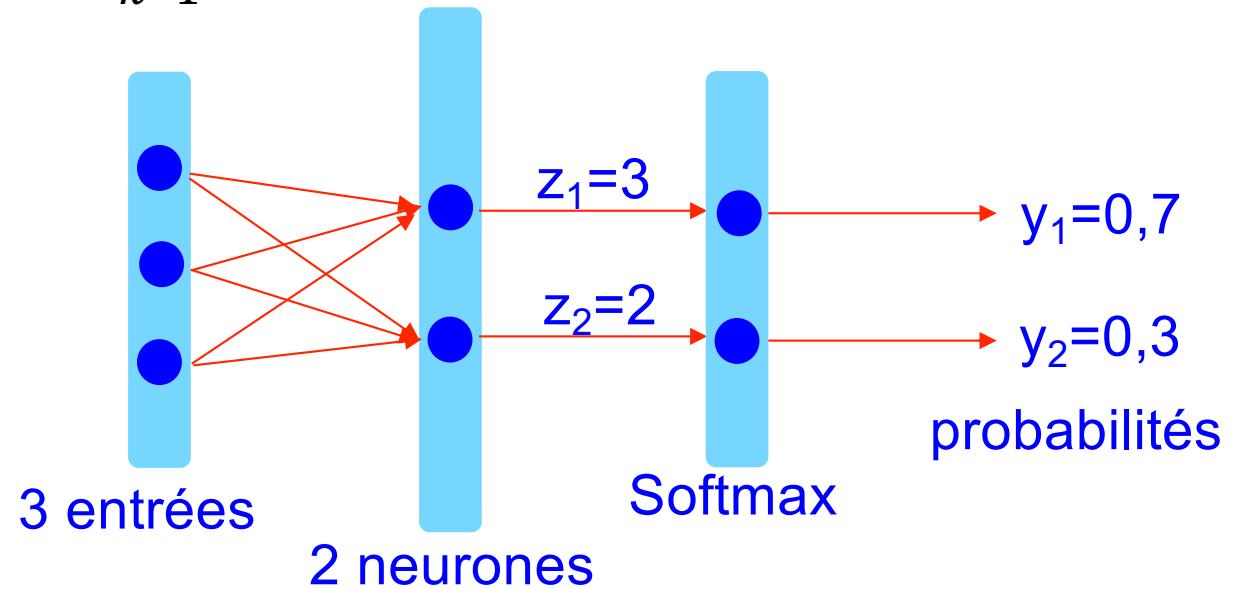
Fully Connected Layer



- 3 activations x_i pondérées par les poids $w_{i,k} \Rightarrow 2$ scores y_k
- avec des poids appris (3×2), pour une entrée inconnue, ce layer combine les entrées et génère des valeurs réelles y_1 et y_2 qui estiment l'appartenance à la classe 1 ou à la classe 2

Fonction softmax

- Fonction softmax (fonction exponentielle normalisée)
- Pour traduire les valeurs réelles en probabilité d'appartenance à chacune des classes possibles
- Pour K classes : $y_k = \frac{e^{x_k}}{\sum_{k=1}^K e^{x_k}}$



Classification Layer : rétropropagation de l'erreur

- Comment faire évoluer les poids lors de l'apprentissage ?
- On les initialise par une distribution gaussienne de moyenne nulle et d'écart-type 0,01 sur matlab
- On commence à présenter les images une par une devant le réseau en précisant : voici la classe de cette image
- Avec les poids initialisés aléatoirement, cela génère des probabilités d'appartenance à chaque classe potentielle en sortie : des y_k

Classification Layer : rétropropagation de l'erreur

- On calcule une fonction coût (**loss function**) (par exemple l'erreur quadratique) entre la sortie y exacte attendue (vérité terrain) et celle estimée par le réseau $y_{\text{estimée}}$
- Dans notre exemple, imaginons que la vérité terrain est la classe 1 :

☞ $d_2(y, y_{\text{estimée}}) = \frac{1}{2} \sum_{k=1}^2 e_k^2 = \frac{1}{2} [(1 - 0,7)^2 + (0 - 0,3)^2]^{1/2} = J$

- ☞ On veut trouver les poids w qui minimise ce coût J
- ☞ Algorithme de descente de gradient : rétropropagation de l'erreur dans le réseau pour modifier les poids

Rétropropagation de l'erreur

Pour chaque vecteur d'entrée X , \rightarrow un vecteur de sortie désiré: Y^{des}

Fonction de coût : **Minimisation de l'erreur quadratique en sortie**

- Parmi les N exemples de la base d'apprentissage, on présente un exemple $X=[x_1 \dots x_{n0}]$
- **Propagation:** on calcule la sortie correspondante $Y=[y_1 \dots y_{n2}]$

• **erreur :** $e_k = y_k^{des} - y_k$

• **coût associé à l'exemple :** $J_{(exemple)} = \frac{1}{2} \sum_{k=1}^{n2} e_k^2$

• **coût global à minimiser :** $J(w) = \sum_{l=1}^N J_{(exemple \ l)}$

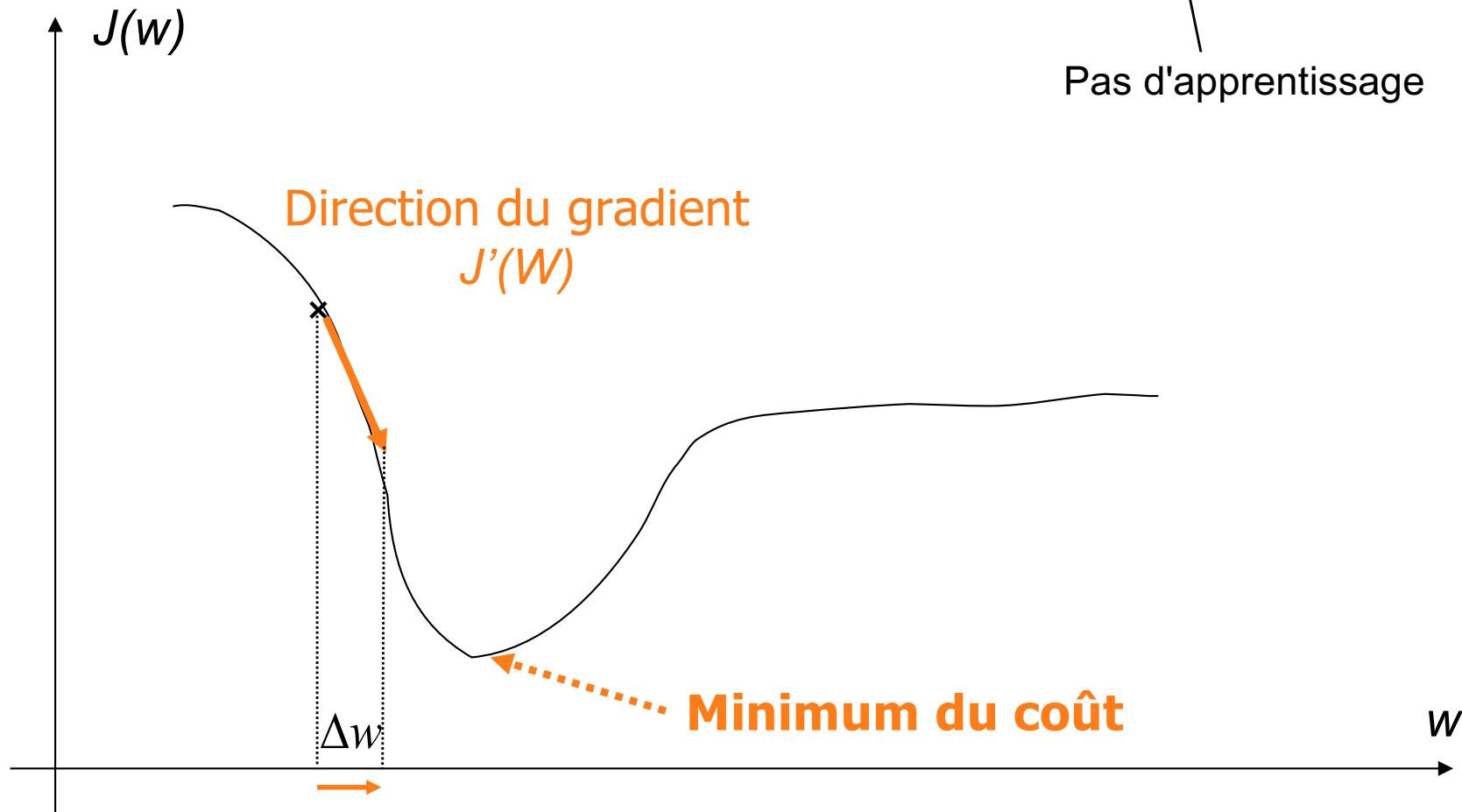
Algorithme de descente du gradient

□ Solution itérative :

Illustration dans le plan $(J(w), w)$

$$\Delta w = -\eta \frac{\partial J}{\partial w}, \quad \eta > 0$$

Pas d'apprentissage

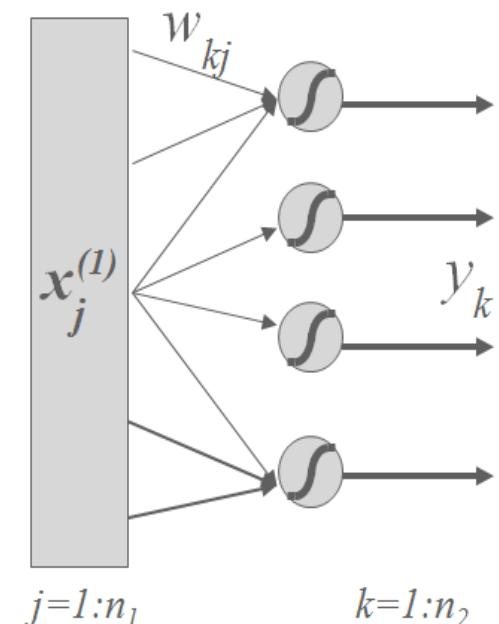


Rétropropagation du gradient : couche de sortie

- Calcul de $\frac{\partial J}{\partial w_{kj}}$ pour un exemple fixé

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial y_k} \frac{\partial y_k}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial w_{kj}}$$

$J = \frac{1}{2} \sum_{k=1}^{n_2} (y_k^{des} - y_k)^2$ $y_k = \Phi(a_k^{(2)})$ $a_k^{(2)} = \sum_{j=0}^{n_1} w_{kj} x_j^{(1)}$
 $- (y_k^{des} - y_k)$ $\Phi'(a_k^{(2)})$ $x_j^{(1)}$
 $\underbrace{\hspace{10em}}_{Err_k = -e_k \Phi'(a_k^{(2)})}$

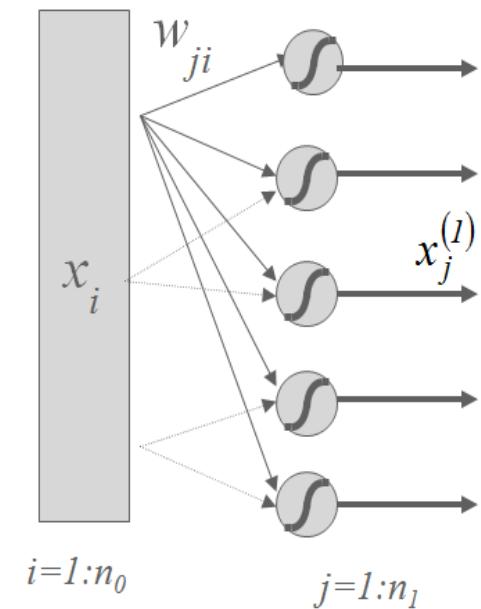


$$\frac{\partial J}{\partial w_{kj}} = Err_k \cdot x_j^{(1)}$$

Rétropropagation du gradient : couche cachée

➤ Calcul de $\frac{\partial J}{\partial w_{ji}}$ pour un exemple fixé

$$\begin{aligned} \frac{\partial J}{\partial w_{ji}} &= \frac{\partial J}{\partial x_j^{(1)}} \frac{\partial x_j^{(1)}}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial w_{ji}} \\ \frac{\partial J}{\partial x_j^{(1)}} &= \sum_{k=0}^{n_2} \frac{\partial J}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial x_j^{(1)}} \\ \frac{\partial J}{\partial x_j^{(1)}} &= \sum_{k=0}^{n_2} Err_k w_{kj} \\ Err_j &\equiv \frac{\partial J}{\partial a_j^{(1)}} = \left(\sum_{k=0}^{n_2} Err_k w_{kj} \right) \Phi'(a_j^{(1)}) \end{aligned}$$



$$\frac{\partial J}{\partial w_{ji}} = Err_j \cdot x_i$$

Rétropropagation de l'erreur dans le NN

- **Cross Entropy Loss** (loss function utilisée par défaut par matlab) :

$$J_{CE} = - \sum_{k=1}^C y_k^{des} \log y_i$$

- Pour la classification (**multi-classes**)
- Le plus souvent utilisée avec la fonction d'activation **softmax** en sortie du réseau de neurones :

$$\varphi(Y) = \frac{e^{y_i}}{\sum_{k=1}^C e^{y_k}}$$

Phase Apprentissage

- Maintenant que l'architecture du réseau est choisi, regardons les paramètres à fixer pour cette phase d'apprentissage (training options sur matlab)
- % Set training options and train the network
- miniBatchSize = 500;
- options = trainingOptions('sgdm', ...
• 'InitialLearnRate', 0.01, ...
• 'MaxEpochs', 4, ...
• 'Shuffle', 'every-epoch', ...
• 'MiniBatchSize', miniBatchSize, ...
• 'ValidationData', {imgDataTest, labelsTest}, ...
• 'ValidationFrequency', 40, ...
• 'Verbose', false, ...
• 'Plots', 'training-progress');

Phase Apprentissage

- `miniBatchSize = 500;`
- La rétropropagation ne se fait pas à chaque exemple (on ne met pas à jour les poids avec chaque image ou observation présentée au réseau)
 - ☞ Bcp de temps de calcul
 - ☞ Ce serait très bruité : chaque exemple en entrée aurait la même influence dans les variations des poids, alors qu'il y a forcément des exemples à la marge
 - ☞ On risque de bloquer le réseau dans un minimum local (cf. ce qui a été expliqué sur la rétropropagation)

Phase Apprentissage

- `miniBatchSize = 500;`
- La rétropropagation se fait tous les 500 exemples
- On fait la moyenne de l'erreur sur le minibatch de 500 exemples
- Ce qui revient à estimer une erreur statistique
- On fait la rétropropagation : Stochastic Gradient Descent (**SGD**)
(cf. ce qui a été vu pour la rétropropagation)
- Exemple : 60 000 images dans la base ; minibatch de 500 → on fera la rétropropagation 120 fois (120 itérations)
- Une itération :
 - ☞ Un minibatch qui est utilisé pour l'apprentissage
 - ☞ Une rétropropagation : une mise à jour des poids du réseau

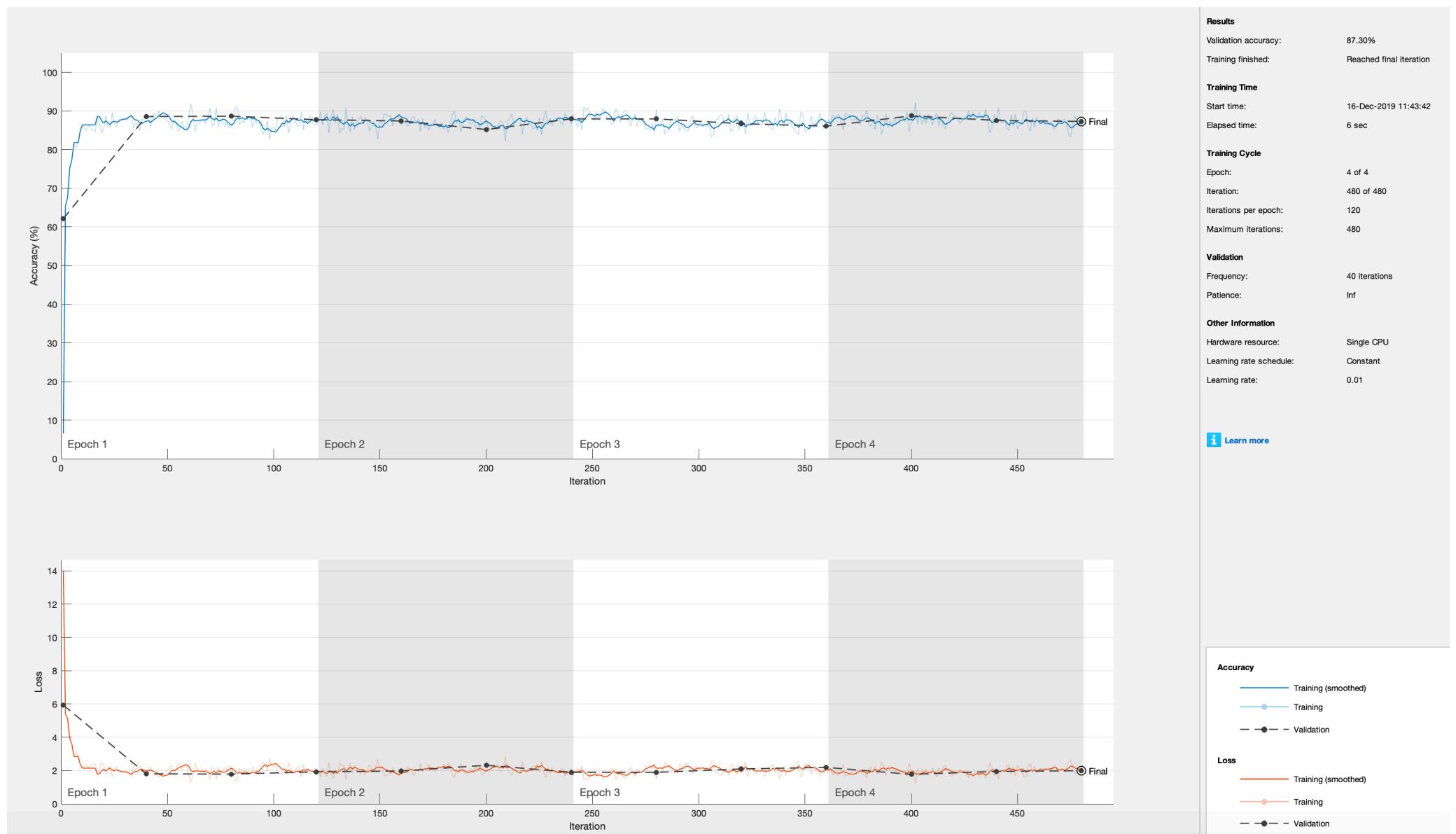
Phase Apprentissage

- `options = trainingOptions('sgdm', ...)`
- `'InitialLearnRate', 0.01, ...`
 - ☞ Fréquence des mises à jour :
 - un coefficient multiplicatif pondérant la fonction coût
 - Pour éviter de grosses variations dans les poids du réseau
 - Ce coefficient peut varier((diminuer) au fur et à mesure des itérations (voir Learning Rate Schedule sur matlab) (par défaut : constant)
- `'MaxEpochs', 4, ...`
 - ☞ epoch : cycle complet d'apprentissage : toute la base a été utilisée
 - ☞ MaxEpochs=4 : on fera 4 passages complets de la base
- `'Shuffle', 'every-epoch', ...`
 - ☞ À chaque epoch, on brasse aléatoirement la base ; ainsi les images ne sont pas présentées dans le même ordre au réseau

Phase Apprentissage

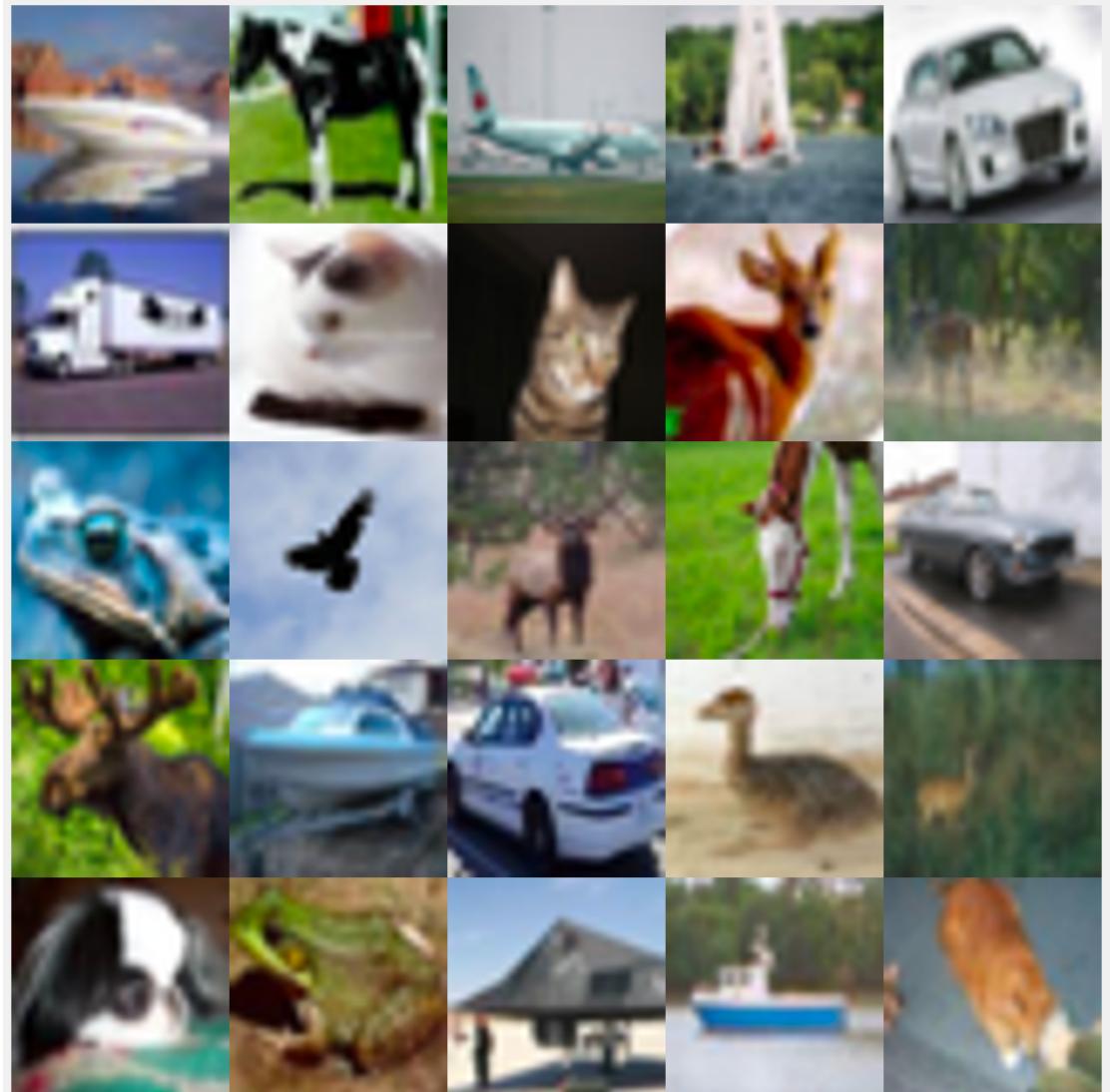
- `'ValidationData', {imgDataTest, labelsTest}`
}, ...
 - ☞ Une base d'apprentissage, une base de validation
- `'ValidationFrequency', 40, ...`
 - ☞ C'est bien de tester le réseau au fur et à mesure qu'il apprend
 - ☞ Toutes les 40 itérations, on teste le réseau en lui proposant en entrée la base de validation ➔ accuracy (pourcentage d'images bien classées)

Training Process



Base de données

- Dataset cifar10 :
 - ☞ Une base d'images couleur
 - ☞ Base d'apprentissage : 50000 images 32x32 pixels + labels
 - ☞ Base de test : 10000 images 32x32 + labels

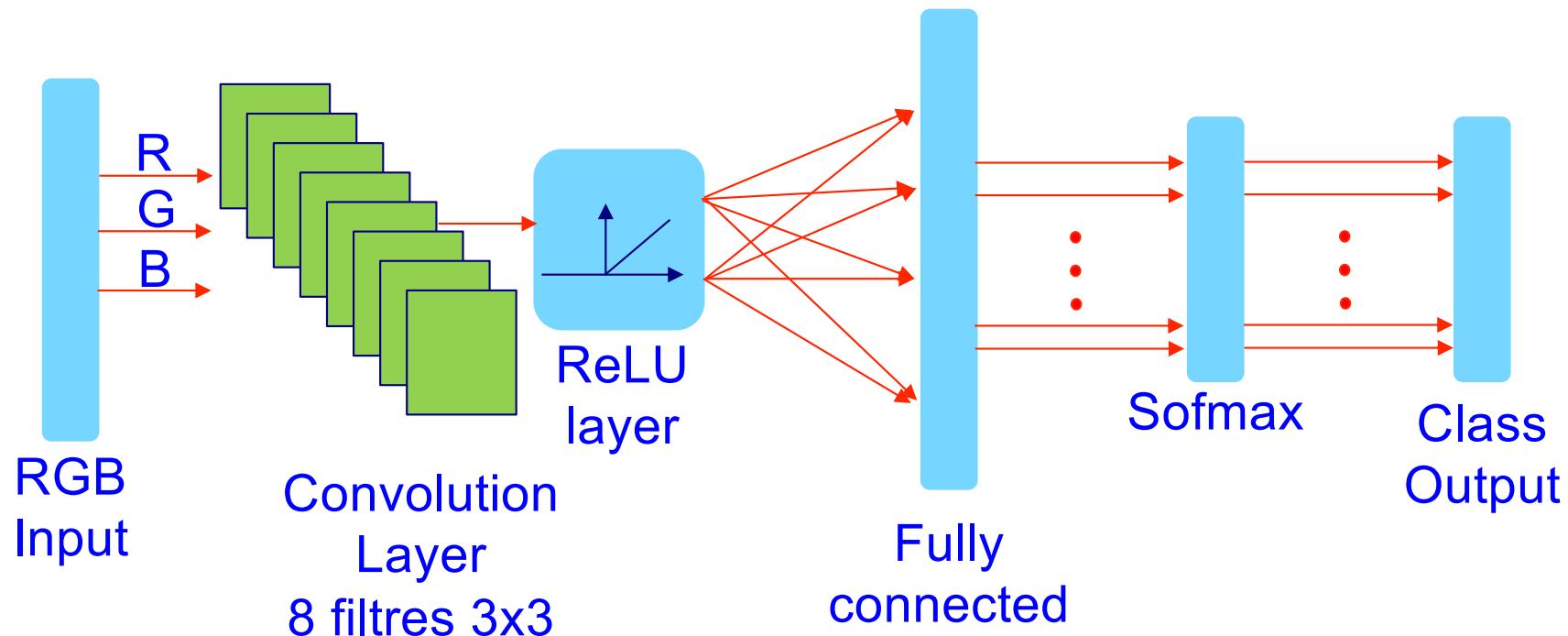




Réseaux de neurones convolutifs

(Convolutional NN : CNN)

Exercice 2 : CNN avec une couche convulsive



- Fichiers matlab à utiliser :
 - ☞ `MNIST_database.m` ; fonction `prepareData` ;
 - `CNN_1coucheC_pour_MNIST.m`

Convolution Layer

- Pourquoi ajouter la couche convulsive ?
- Idée partie d'une expérimentation faite par Hubel and Wiesel in 1962 :
 - ☞ Seuls certains neurones répondent (s'activent) lorsque l'individu voit un contour ou une forme dans une certaine direction ; pour une autre direction, ce sont d'autres neurones qui s'activent ==> il y a donc une spécialisation dans la perception visuelle
 - ☞ L'opérateur de convolution est un opérateur de filtrage très bien adapté pour séparer des structures, orientations dans les images

Convolution Layer

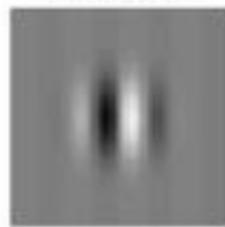
- Exemple : filtres FIR ondelettes qui détectent des variations horizontales, verticales et diagonales :



Convolution Layer

- Exemple : filtres gaussiens orientés qui détectent des variations dans plusieurs directions : ici on visualise les filtres :

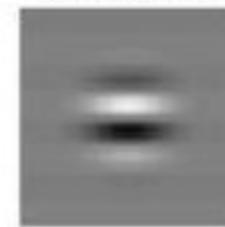
0 degree



45 degree



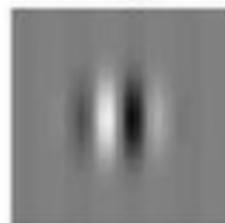
90 degree



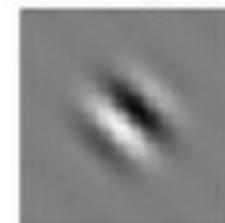
135 degree



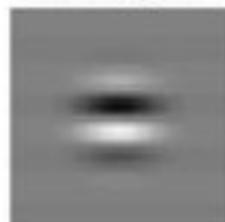
180 degree



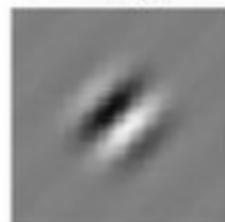
225 degree



270 degree



315 degree



Convolution Layer

- La couche de convolution est composée d'un ensemble de filtres FIR 2D
- **C'est un ensemble de détecteurs spécialisés**
- **Ou c'est un ensemble d'identifieurs de caractéristiques (feature identifier)**
- Un filtre donné va détecter l'existence ou pas d'une structure ou orientation ou forme particulière dans les images qu'on présente au CNN

Convolution Layer

- Fonction matlab : convolution2dLayer
- `convolution2dLayer(5, 4, 'Stride', [1,1], 'Padding', '[1,1,1,1]');`
 - ☞ On définit 4 filtres FIR de taille 5x5
 - ☞ Le masque des filtres est initialisé aléatoirement
 - ☞ Le réseau va apprendre les coefficients des 4 filtres au fur et à mesure des itérations
 - ☞ Stride : choix du pas de glissement : si Stride= [1,1], la fenêtre se déplace d'un pixel en ligne et d'un pixel en colonne
 - ☞ Pour les filtres de taille plus grande, on peut prendre un Stride plus grand pour faire moins de convolution

Convolution Layer

- `convolution2dLayer(5, 4, 'Stride', [1,1], 'Padding', '[1,1,1,1]);`
 - ☞ Padding : mettre des zéros autour de l'image I avant le filtrage : on obtient une image plus grande J : le filtre sera appliqué seulement sur les pixels de cette image J (on attend que le filtre rentre complètement dans J pour commencer la convolution)
 - ☞ L'objectif est de maintenir la même taille de l'image après le filtrage en éliminant les effets de bord dus à la convolution
 - ☞ Dans l'exemple du haut, on ajoute une ligne en haut, une ligne en bas, une colonne à gauche et une colonne à droite
 - ☞ Exercice : imaginons des images 32x32 à l'entrée d'un filtre 5x5. Qu'est-ce qu'il faut choisir comme Padding pour avoir une sortie 32x32 ?

Convolution Layer

- Regardons l'animation ici :

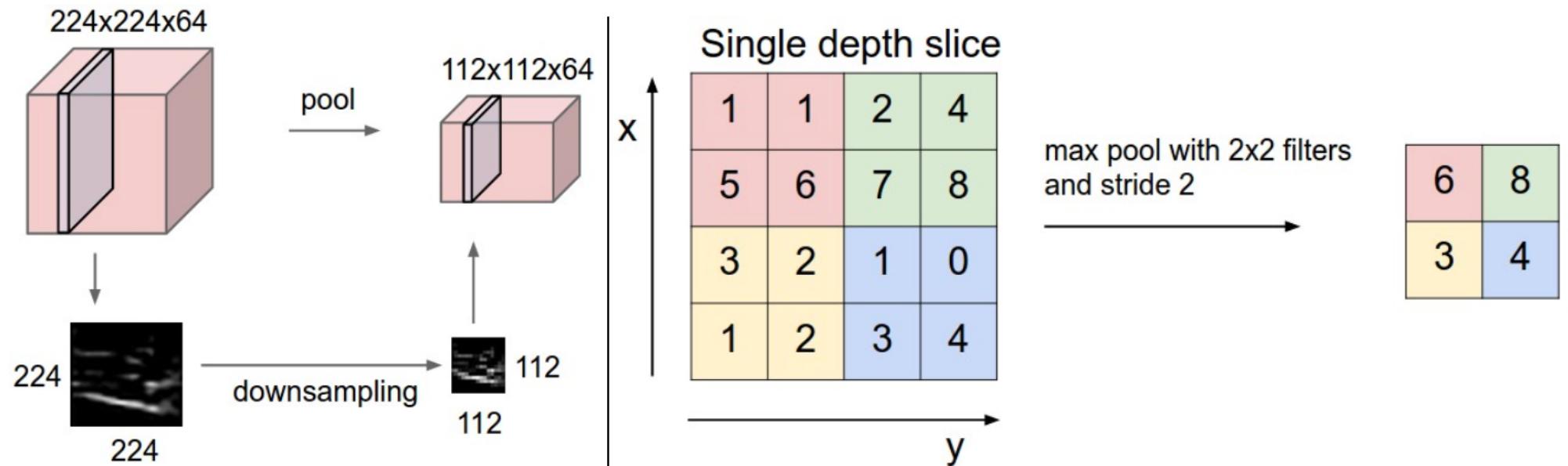
☞ <https://fr.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html>

Pooling Layer

- Pooling Layer : sa fonction est de réduire progressivement (au fur et à mesure qu'on traverse des couches de convolution) la taille des images filtrées par cette succession de filtre FIR et ainsi :
 - ☞ Réduire le nombre de paramètres
 - ☞ Réduire le temps d'exécution
 - ☞ S'éloigner un peu des pixels originaux pour que le réseau ne sur-apprend pas
- C'est un sous-échantillonnage (down sampling)
- Exemple : Average Pooling (2) : sous-échantillonnage par 2, en prenant la moyenne de 4 pixels
- Max Pooling Layer

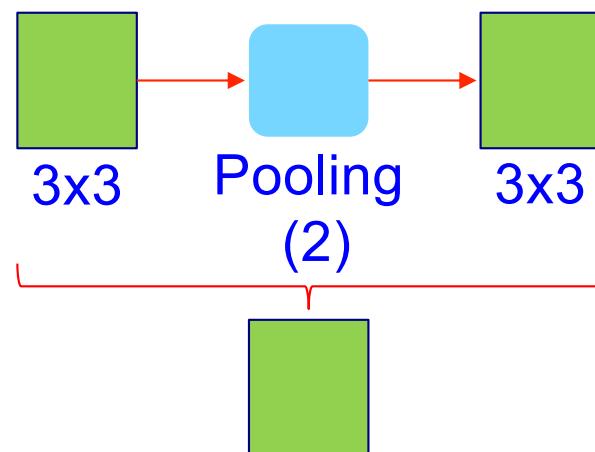
Pooling Layer

- Max Pooling Layer(2) : on sous-échantillonne en prenant le max des 4 pixels



Convolution Layer

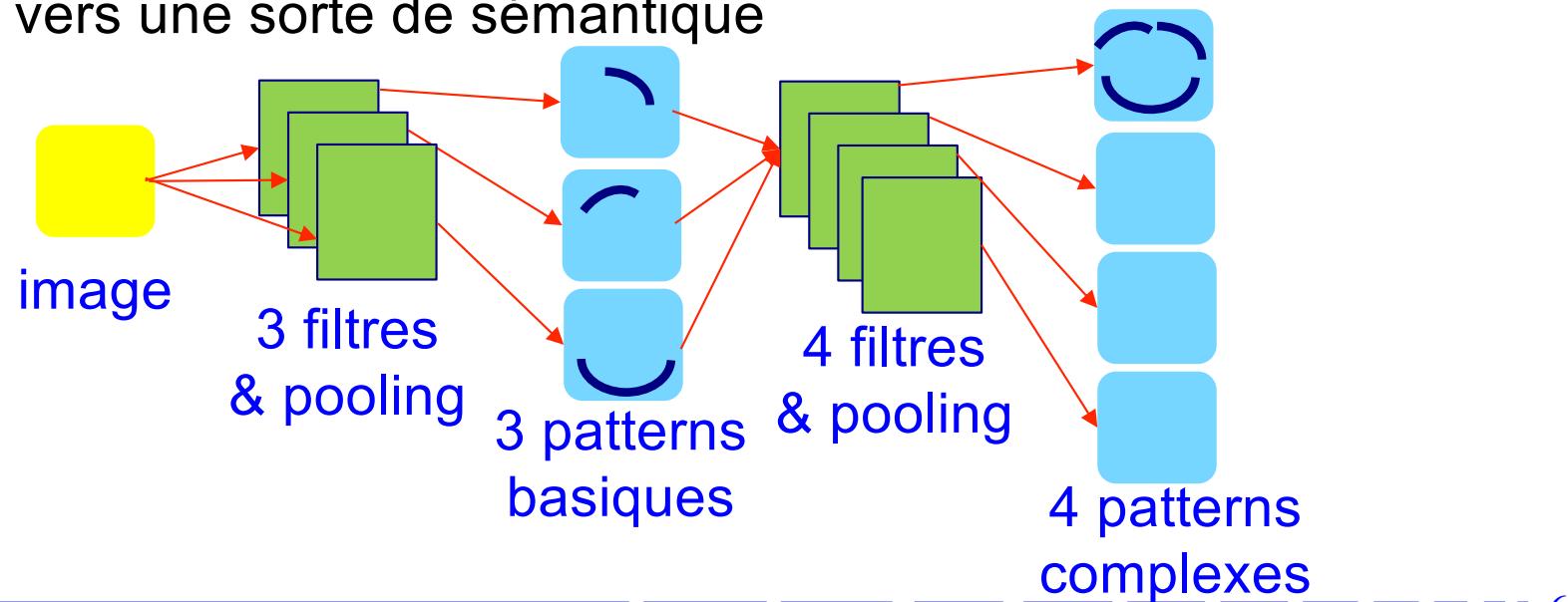
- Pourquoi mettre plusieurs couches en cascade ? (faire du deep learning ?)
 - ☞ 2 filtres FIR 3×3 en cascade est équivalent à 1 filtre FIR 5×5
 - ☞ $N+L-1 = 3+3-1 = 5$
 - ☞ C'est donc équivalent à filtrer une fenêtre plus grande
 - ☞ De plus, avec un pooling au milieu (exemple par 2)



C'est comme si le filtre analyse une fenêtre 8×8 : $3 \times 2 + 2$

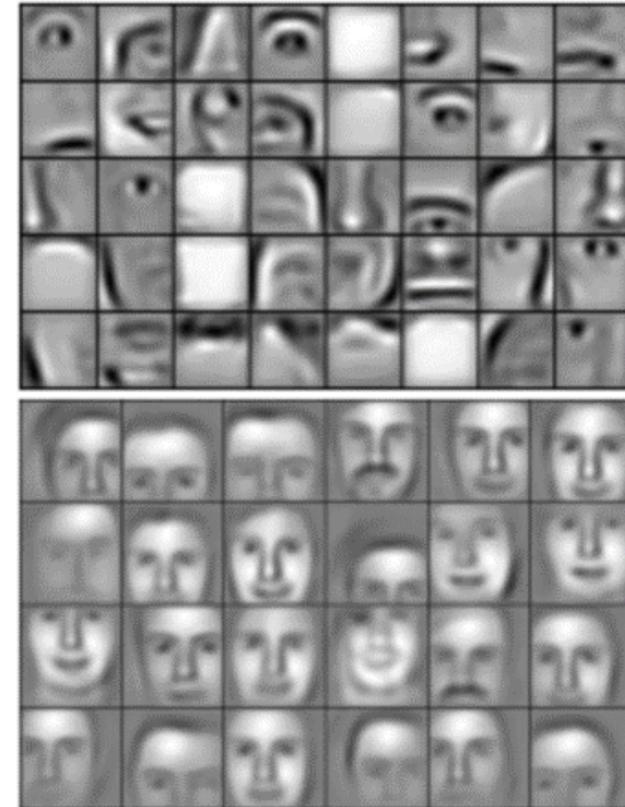
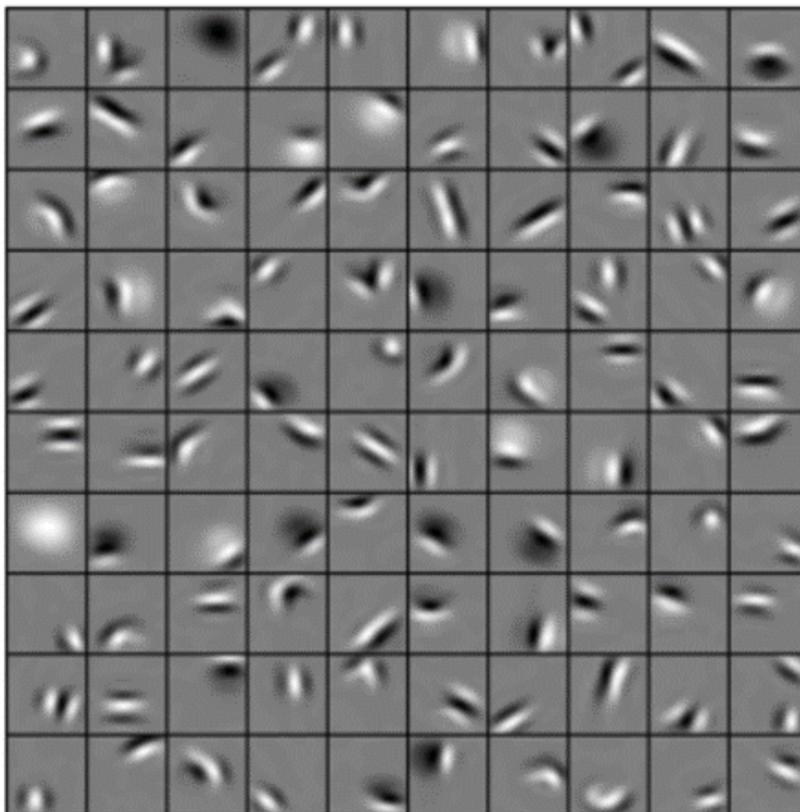
Convolution Layer

- Pourquoi mettre plusieurs couches en cascade ? (faire du deep learning ?)
 - Avec une 2e couche, on analyse et détecte des formes plus grandes
 - De plus, comme on combine les formes basiques obtenues par la première couche, on analyse des formes de plus en plus complexes et grandes ➔ plus on avance dans les couches de convolution, plus on va vers une sorte de sémantique



Convolution Layer

- Pourquoi mettre plusieurs couches en cascade ? (faire du deep learning ?)
 - ☞ Exemple de patterns obtenus avec une base d'images visages

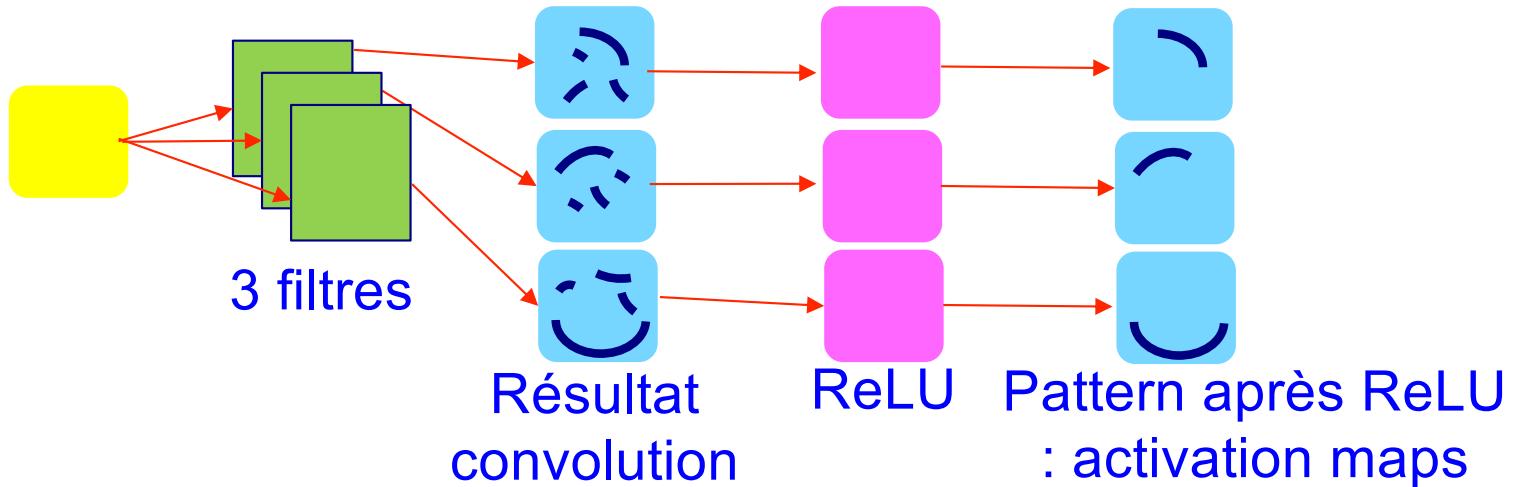


batchNormalization Layer

- Fonction matlab : batchNormalizationLayer
- Avant d'attaquer une couche de convolution, l'idée est d'assurer que les patterns issus de la couche précédente soient normalisés : moyenne nulle et écart-type=1 ; il ne faudrait pas que certaines données soient plus influentes que d'autres, lors de l'apprentissage du réseau
- Cette normalisation se fait pour chaque mini batch.

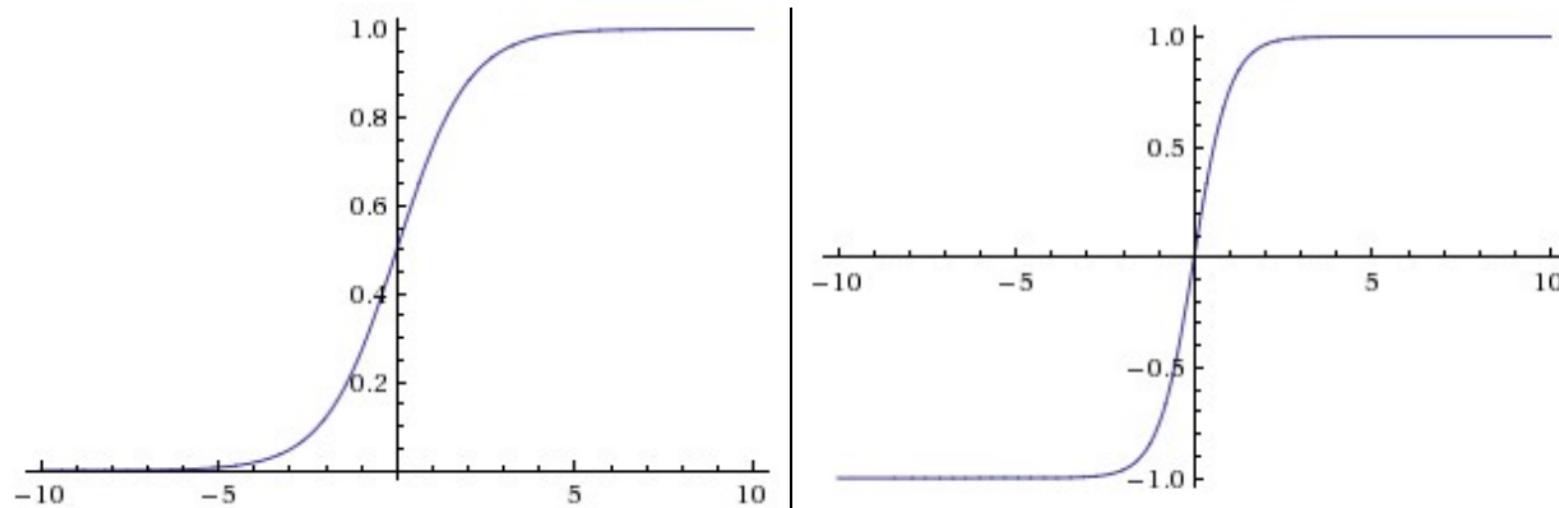
Fonction d'activation après la convolution

- Est-ce qu'on prend toutes les valeurs issues de la convolution ?
 - ☞ Fonction qui décide si le neurone (ici un pixel des patterns qui sortent des filtres) est activé ou non
 - ☞ Plusieurs fonctions non linéaires possibles : sigmoïd, tanh, ReLU
 - ☞ C'est souvent un seuillage qui garde les valeurs plus grandes qu'un seuil



Exemples de fonctions d'activation

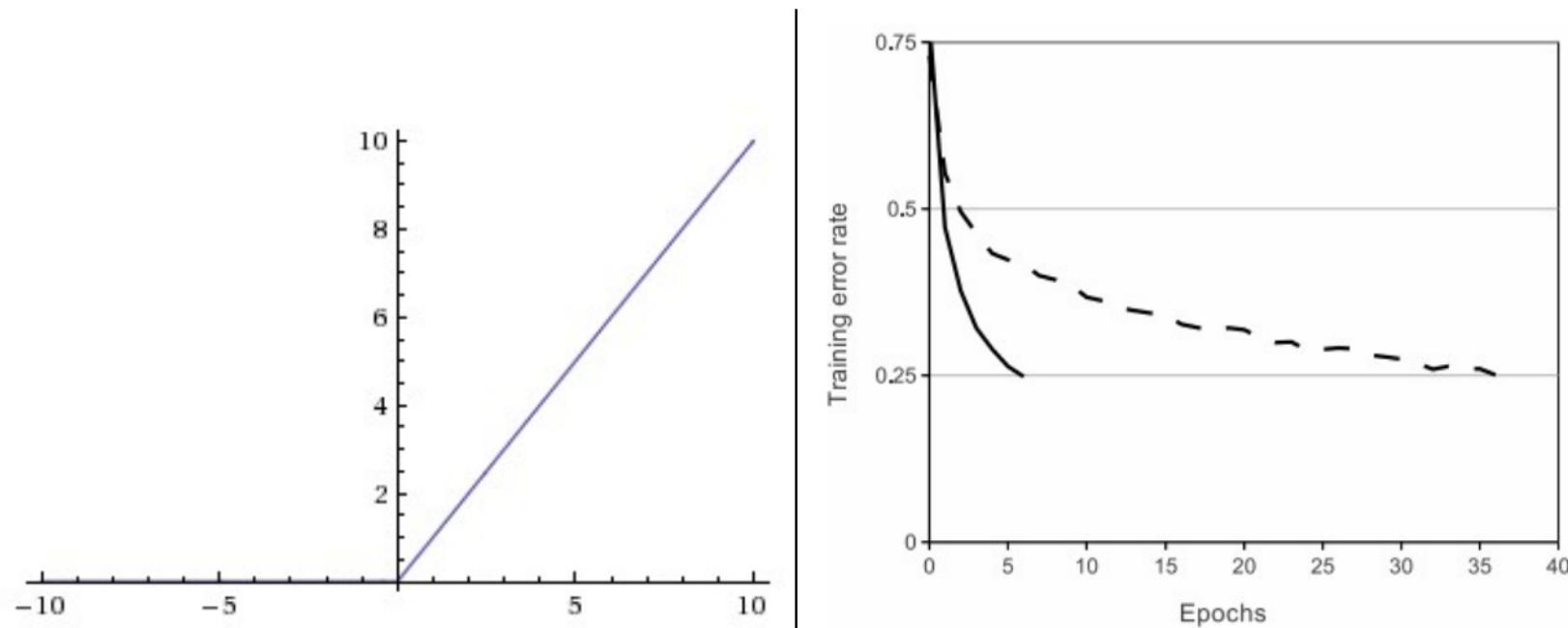
- Fonctions non linéaires et bornées



Left: Sigmoid non-linearity squashes real numbers to range between [0,1] Right: The tanh non-linearity squashes real numbers to range between [-1,1].

Fonction ReLU : fonction rampe...

- Fonction non linéaire (une non linéarité plus franche)



Left: Rectified Linear Unit (ReLU) activation function, which is zero when $x < 0$ and then linear with slope 1 when $x > 0$. Right: A plot from Krizhevsky et al. (pdf) paper indicating the 6x improvement in convergence with the ReLU unit compared to the tanh unit.



Apprentissage par transfert (Transfer Learning)

Transfer Learning

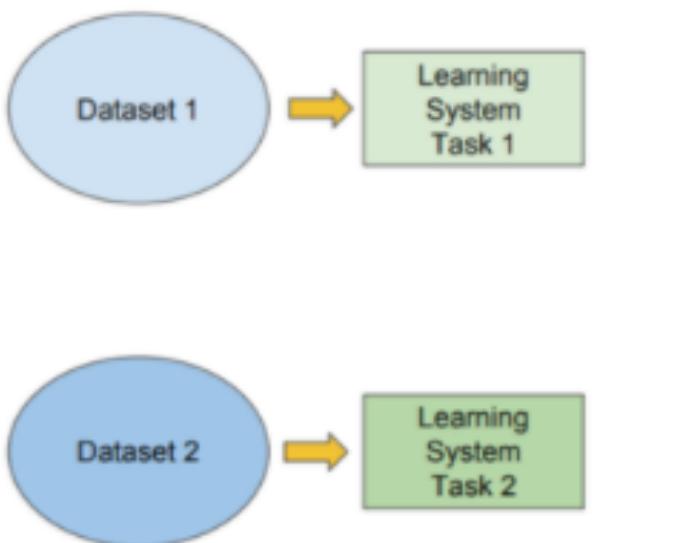
- Apprendre un réseau avec une base généraliste très grande
 - ☞ Utiliser un réseau déjà appris (réseau pré-entraîné) par d'autres qui disposent de gros moyens de calcul par exemple
- Puis spécialiser le réseau à une base spécifique (visages ou détection de piétons ou empreintes digitales)
- Cette base spécifique est probablement de plus petite taille car on a peut-être peu d'images spécifiques

Transfer Learning

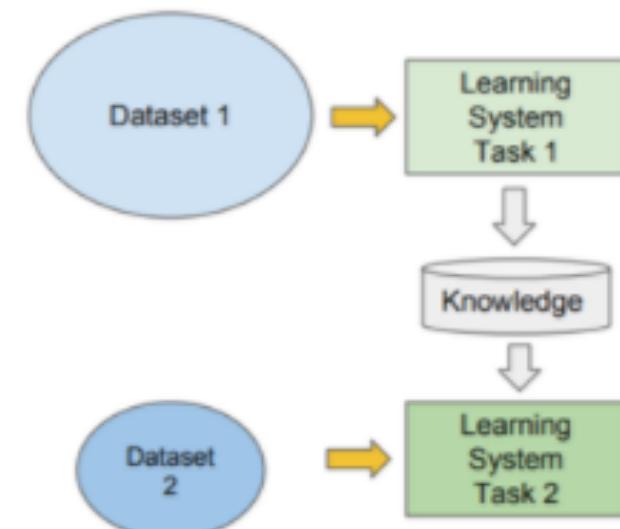
Utiliser la connaissance apprise par le réseau initial

Traditional ML vs Transfer Learning

- Isolated, single task learning:
 - Knowledge is not retained or accumulated. Learning is performed w.o. considering past learned knowledge in other tasks



- Learning of a new tasks relies on the previous learned tasks:
 - Learning process can be faster, more accurate and/or need less training data



Transfer Learning

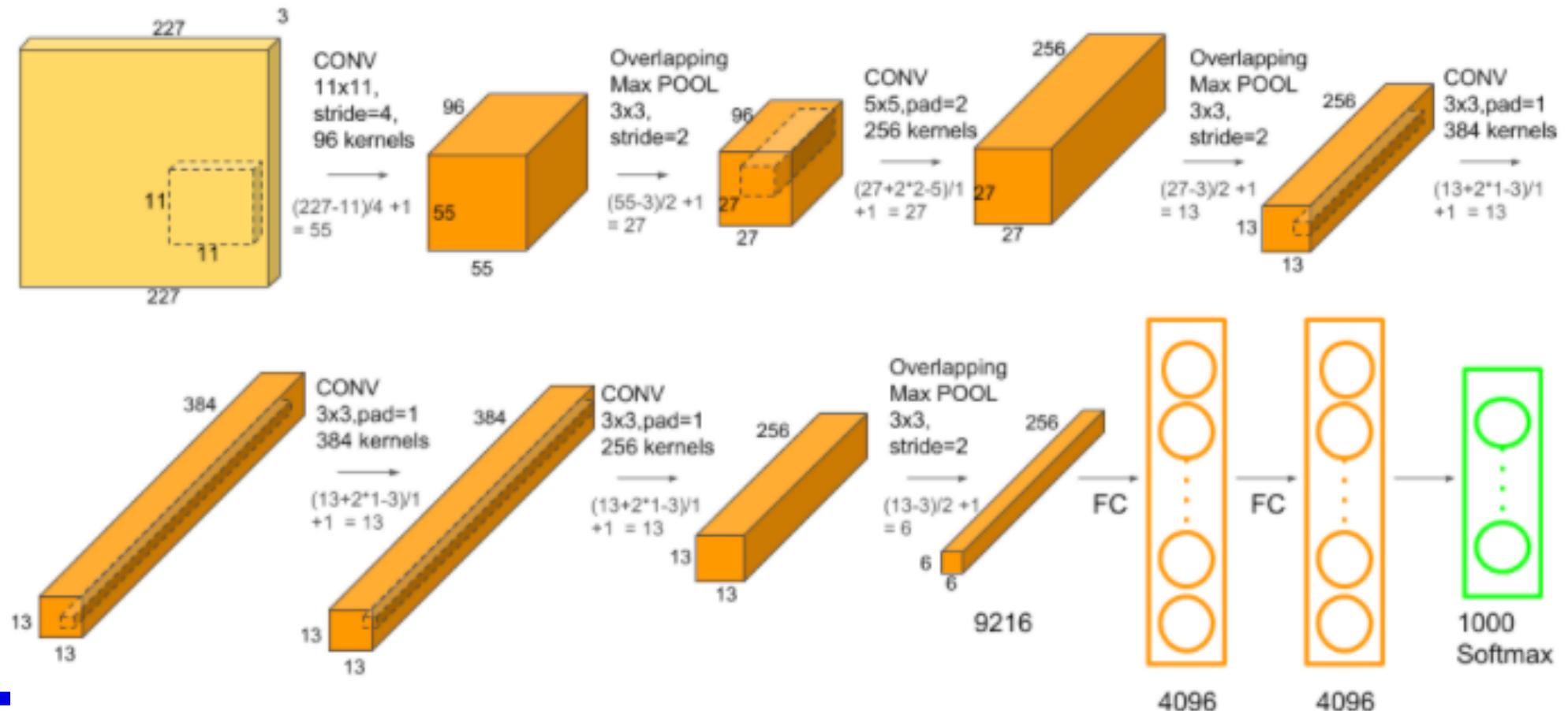
- Comment modifier le réseau initial ?
 - ☞ Surtout ne pas tout changer
 - ☞ Garder les premières couches qui détectent les features simples qui finalement serviront pour n'importe quelle base : *transférer les poids associés au 2^e réseau*
 - ☞ Ne travailler qu'avec certaines couches de sortie (donc sur les détecteurs (filtres) de « semantic features »)
 - ☞ « Fine-tune » les dernières couches de sortie

AlexNet : réseau pré-appris

- Réseau AlexNet 2012 :

- ☞ Réseau appris sur la base ImageNet (> 1 M Image, 1000 classes)

- ☞ 60 million paramètres and 650,000 neurones



Transfer Learning

- Les 3 dernières couches sont configurées pour 1000 classes
- On va les enlever
- A la place, on va mettre :
 - ☞ Une couche Fully Connected (FC)
 - ☞ une fonction softmax
 - ☞ Une couche de sortie
 - ☞ Si la base spécifique est constituée de 10 classes, par exemple MNIST ou CIFAR10, ces 3 nouvelles couches seront configurées avec 10 entrées et sorties



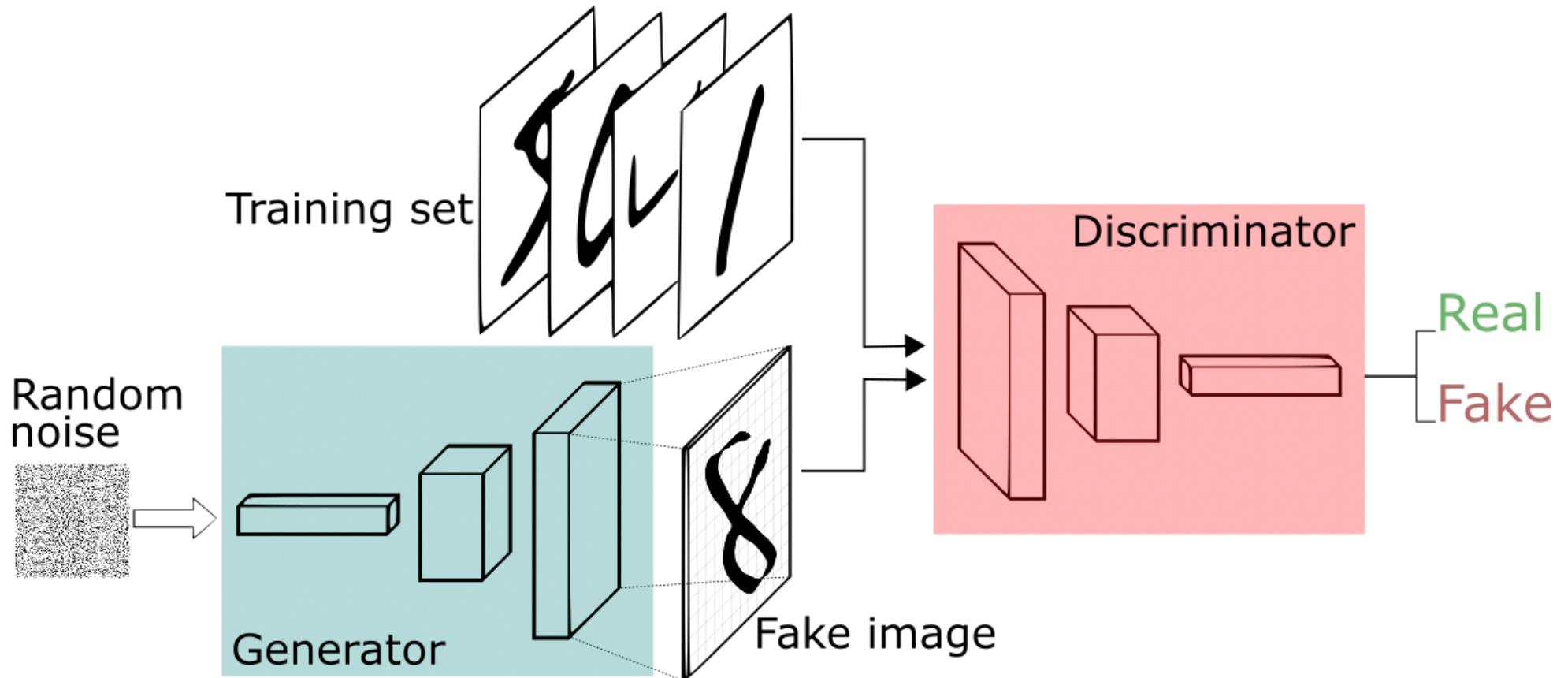
GAN

(Generative Adversarial Networks)

GAN

- Objectif : générer un objet (images, vidéo, son, document, autre) *plus vrai que nature*
- Deux CNN à apprendre :
 - ☞ Un CNN générateur d'objet
 - ☞ Un CNN discriminant qui répond : vrai ou faux

GAN



- Article : *Ian Goodfellow et al., NIPS, 2014*
- Une fois le générateur appris, on laisse tomber le discriminateur

GAN

Exemple de génération d'images MNIST et TFD



Références

- Ouvrage « Analyse des données », Collection Traitement du signal et de l'image, Hermès, Lavoisier
- Ouvrage « Apprentissage artificiel : Deep Learning, concepts et algorithmes », Eyrolles
- <https://adeshpande3.github.io/adeshpande3.github.io/>
 - ☞ A Beginner's Guide To Understanding Convolutional Neural Networks (Parts 1, 2, 3)
- <http://cs231n.github.io/> (notes from Stanford CS class: Convolutional Neural Networks for Visual Recognition)
- <https://fr.mathworks.com/help/deeplearning/ug/layers-of-a-convolutional-neural-network.html>
- https://fr.mathworks.com/campaigns/offers/deep-learning-examples-with-matlab.html?s_v1=25919&elqem=2513773_EM_FR_DIR_18-12_MOE-EDU&elqTrackId=6edbf4ac19a24493bb0efaca842119f0&elq=9922b86fa2cd474f8bd4737d330f13ad&elqaid=25919&elqat=1&elqCampaignId=8903