



Data NoBlabla

**Introduction au
Deep Learning
avec Keras**

Julien Guillaumin

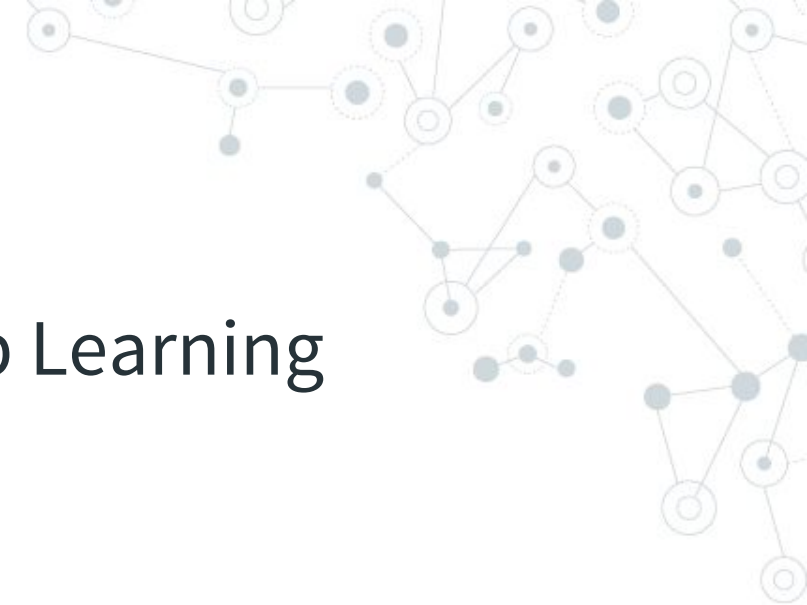

- ◎ Etudiant à Télécom Bretagne
- ◎ Stage chez Continental
 - Deep Learning
- ◎ Kaggle-addict & MOOC-friendly



julien.guillaumin@telecom-bretagne.eu

Objectifs de l'atelier

- ◎ Découvrir le Deep Learning
 - Théorie et Pratique
- ◎ Prendre en main Keras
 - Créer un modèle
 - Contrôler l'entraînement
 - L'utiliser et le sauvegarder
- ◎ Application aux images
 - Classification

- 
- 
1. Machine Learning & Deep Learning
 2. Prendre en main Keras
 3. Réseaux de neurones profonds (DNN)
 4. Réseaux de neurones convolutifs (CNN)

1. Machine Learning et Deep Learning

Rappels :

- ◎ Champ d'étude de l'IA
- ◎ Concevoir des algorithmes qui apprennent
 - À partir de données
 - Résoudre des tâches complexes



Apprentissage :

- **Données d'entraînement**
- **Ajuster les paramètres de l'algorithme**
- **Généralisation**



Inférence

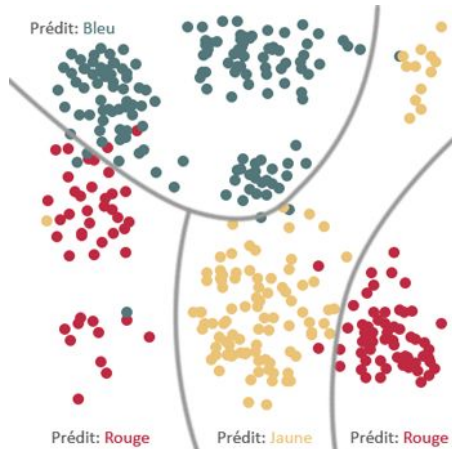
- **Nouvelles données**
- **Evaluer le modèle**

Apprentissage supervisé

- ◎ On connaît la bonne réponse !

Exemple:

- Classification



Apprentissage non-supervisé

- ◎ Trouver une structure dans les données

Exemple:

- Clustering



Classification d'images

Données labellisées:

airplane



automobile



bird



cat



deer



dog



frog



horse



ship



truck



Données non-labellisées:

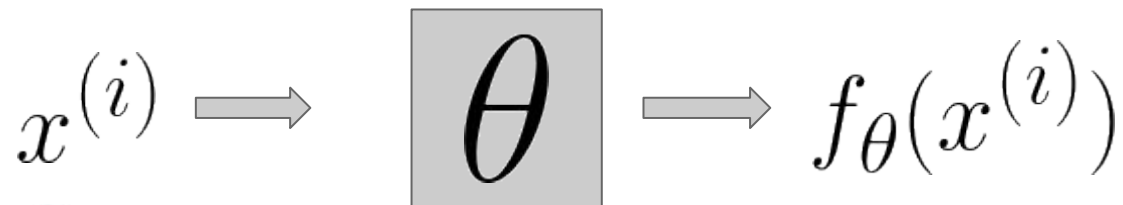


Un peu de notation

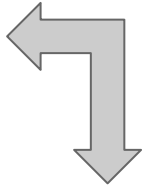
Données : $(x^{(i)}, y^{(i)})$

Paramètres entraînables : θ

Fonction de prédiction : $f_{\theta}(x^{(i)})$



Cas de la classification d'images



$f_{\theta}(x^{(i)}) \Rightarrow$

$$\begin{cases} \mathbb{P}(Y = truck \mid x^{(i)}, \theta) \\ \mathbb{P}(Y = bird \mid x^{(i)}, \theta) \\ \vdots \\ \mathbb{P}(Y = plane \mid x^{(i)}, \theta) \end{cases}$$

0.12

0.81

·

·

·

·

0.05

$y^{(i)} \Rightarrow bird$

$$E = -\log (P(Y = y^{(i)} \mid x^{(i)}, \theta))$$

negative log-likelihood

Fonction de coût

Erreur pour un exemple :

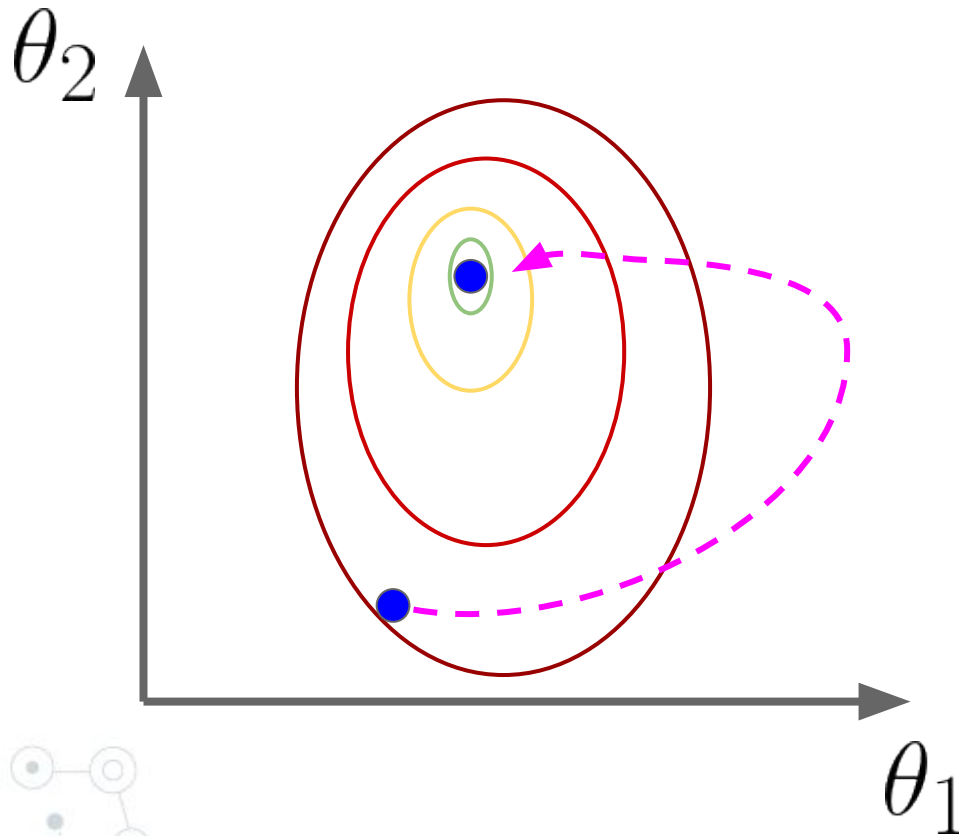
$$E(x^{(i)}, y^{(i)}, \theta) = L(f_{\theta}(x^{(i)}), y^{(i)})$$

Erreur pour l'ensemble des données:

$$E(X, Y, \theta) = \frac{1}{N} \sum_{n=1}^N E(x^{(i)}, y^{(i)}, \theta)$$

Problème d'optimisation:

© Trouver θ qui minimise $E(X, Y, \theta)$



Descente du gradient

- © Méthode itérative
- © λ , pas d'apprentissage (hyper-paramètre)
- © Mise à jour de θ :

$$\theta_{t+1} = \theta_t - \lambda \frac{\partial E(X, Y, \theta)}{\partial \theta}$$

t : itération

Du SGD au minibatch SGD

SGD = Stochastic Gradient Descent

SGD classique :

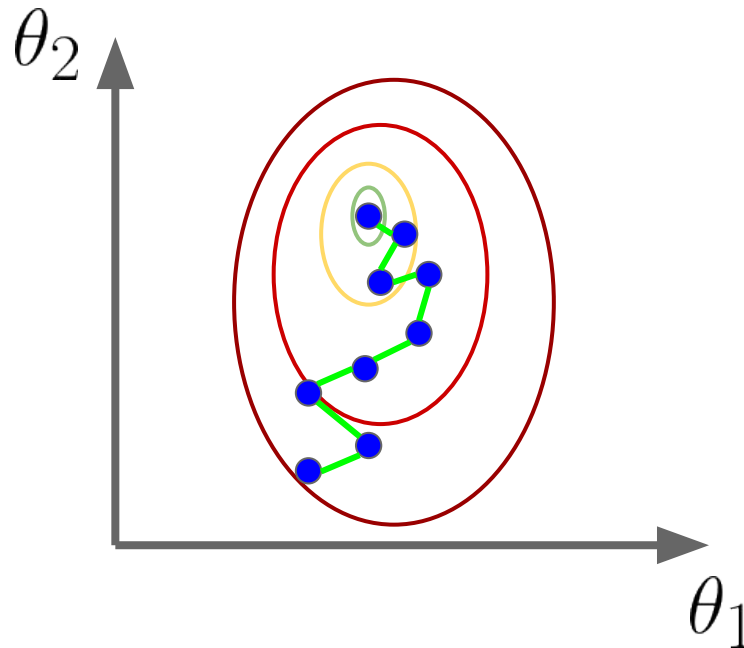
$$\theta_{t+1} = \theta_t - \lambda \frac{\partial E(X^{(i)}, Y^{(i)}, \theta)}{\partial \theta}$$

Minibatch SGD :

$$\theta_{t+1} = \theta_t - \lambda \frac{\partial E(X^{[batch]}, Y^{[batch]}, \theta)}{\partial \theta}$$

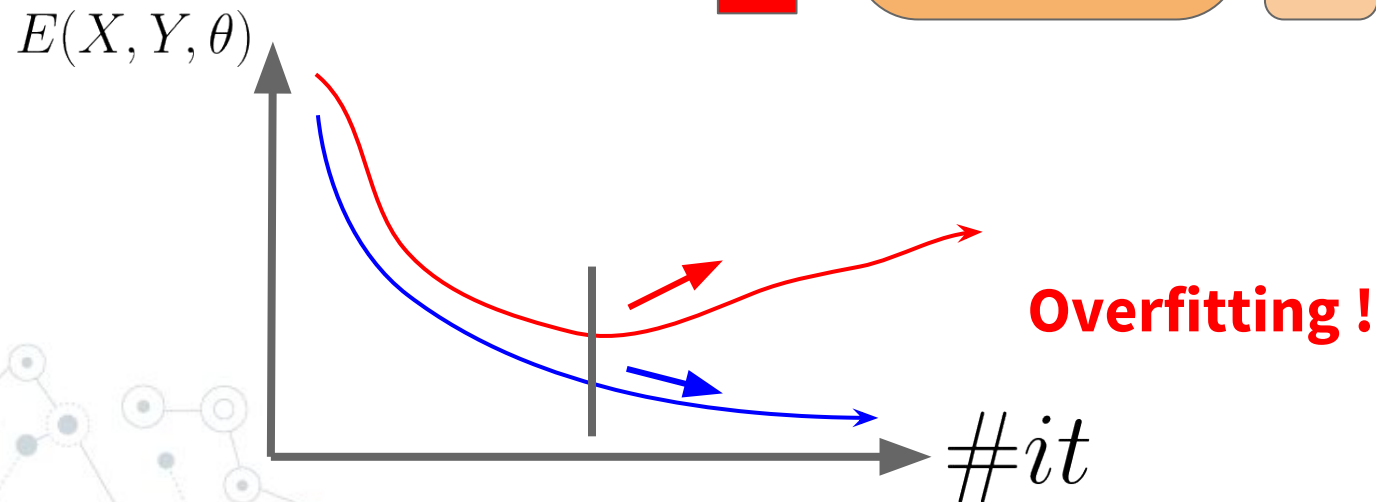
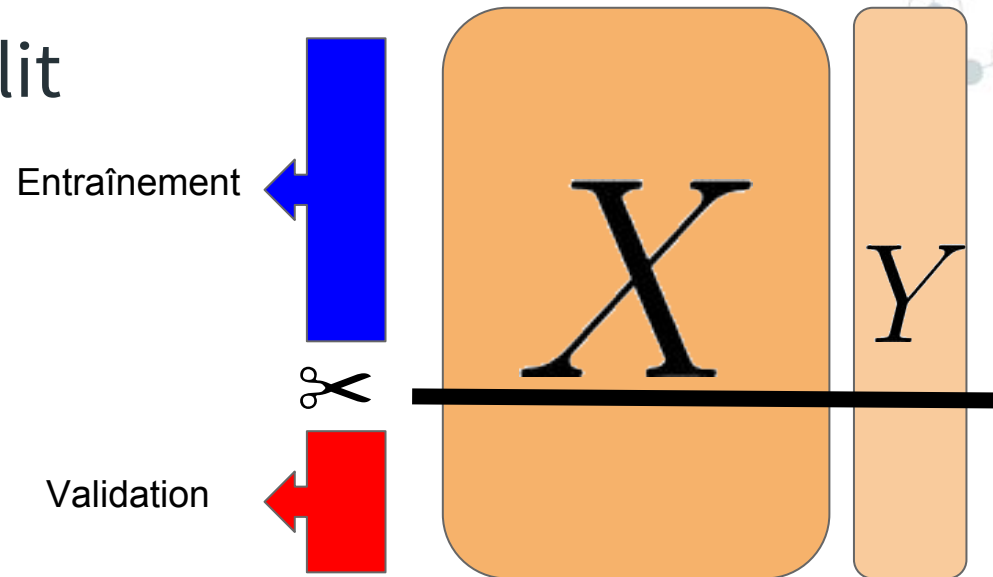
Pseudo-code pour le minibatch SGD

```
for epoch in range(nb_epoch):  
    for batch in range(nb_batch):  
        train_model(model, X = X[batch*batch_size : (batch+1)*batch_size],  
                      y = y[batch*batch_size : (batch+1)*batch_size],  
                      learning_rate = lr)
```



Mesurer les performances

Train/Test split



Combattre l'overfitting : régularisation

- ◎ Appliquer des contraintes au modèle pour réduire le nombre de paramètres libres et leurs amplitudes.

Exemple : régularisation “L2”

$$E = L(f_{\theta}(x^{(i)}), y^{(i)}) + \mu ||\theta||^2$$

Premier modèle : regression logistique



32

32

x

x_j



$$y = \text{softmax}(Wx + b)$$

Wx

b

s

W

+



$$\theta = \{W, b\}$$

$$\hookrightarrow \text{softmax}(h_i) = \frac{e^{h_i}}{\sum_{j=1}^{10} e^{h_j}}$$

Premier modèle : regression logistique



32

32

x

x_j

$$y = \text{softmax}(Wx)$$

s

$$\mathbb{P}(Y = \text{truck} \mid x^{(i)}, \theta)$$

$$\mathbb{P}(Y = \text{bird} \mid x^{(i)}, \theta)$$

\vdots

$$\mathbb{P}(Y = \text{plane} \mid x^{(i)}, \theta)$$

W

1

$$s_i = \text{softmax}\left(\sum_{j=1}^{1024+1} w_{i,j} x_j\right)$$

Et le Deep Learning ?

Là où le Deep Learning joue un rôle:

- construction/représentation de $f_{\theta}(x^{(i)})$
- ◎ Succession de fonctions/transformations
- ◎ Découpage en couches
- ◎ Profondeur : # de non-linéarité (ou couches cachées)

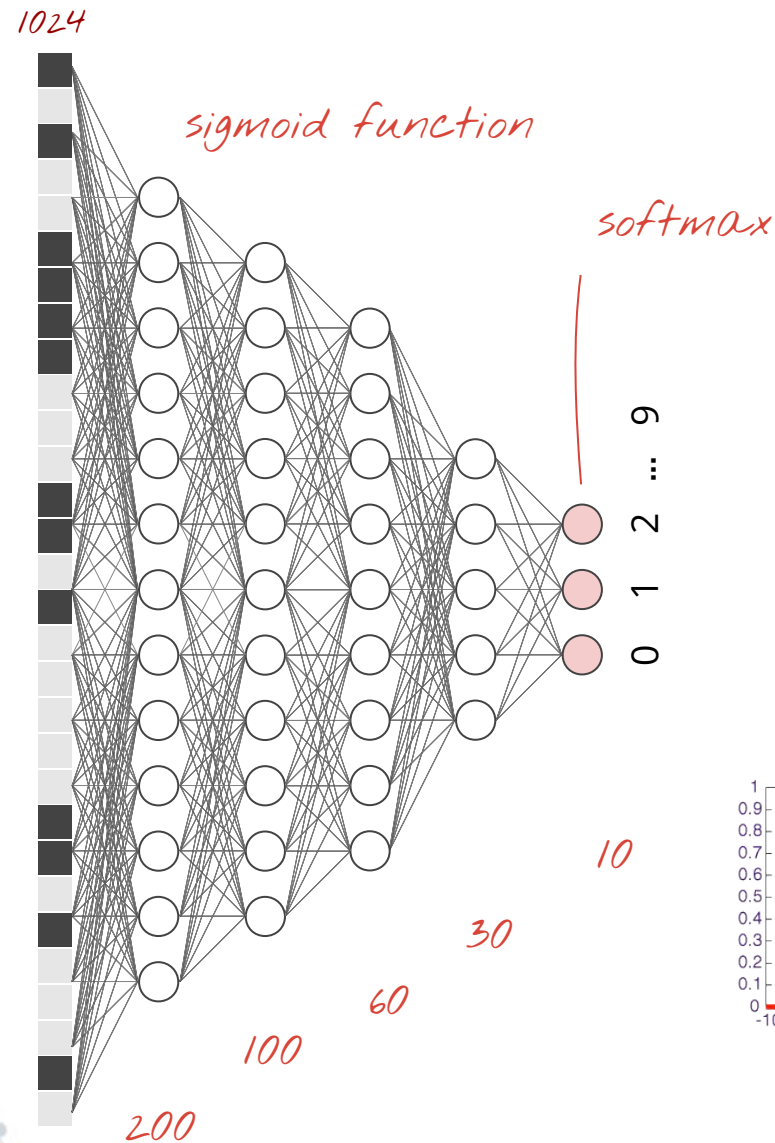
Ce soir ?

- ◎ Réseaux de neurones profonds (DNN)
- ◎ Réseaux de neurones convolutifs (CNN)

Mais beaucoup d'autres :

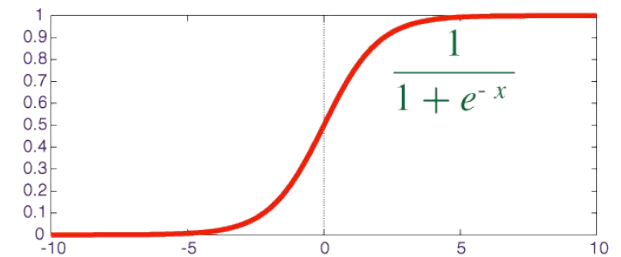
- ◎ LSTM (pour le texte)
- ◎ Autoencoders (et ses variantes)
- ◎ Deep Belief Networks (DBN)
- ◎ Generative Adversarial Networks (GAN) !

Réseaux de neurones profonds (DNN)

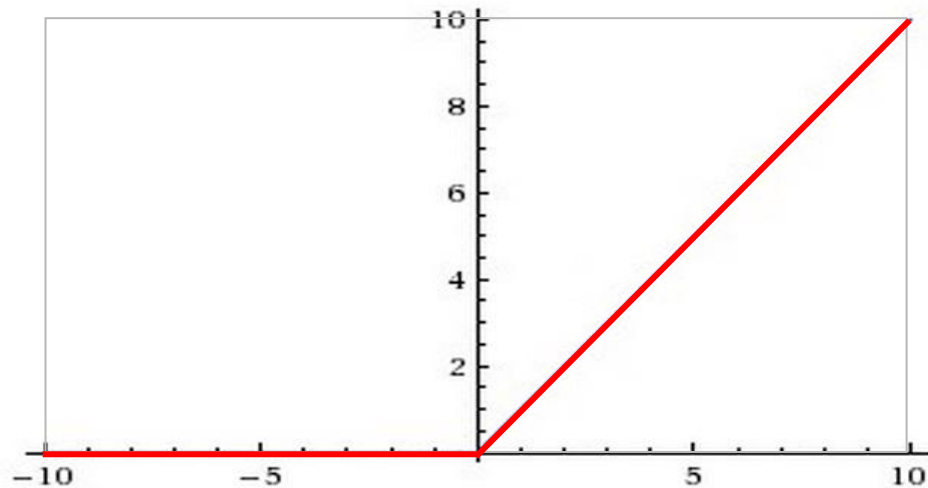


Fonction d'activation :

- Sigmoid
- Tanh
- ReLU !

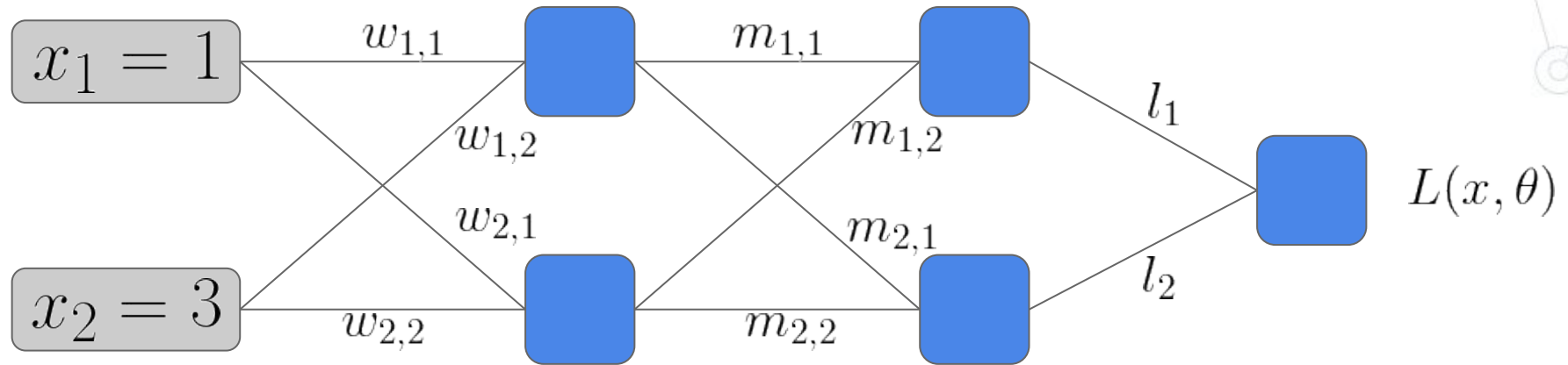


- ◎ Inspiration biologique
- ◎ Transformation d'un vecteur (features) en un autre vecteur !
- ◎ Empiler les couches "full-connected"
- ◎ Fonctions d'activation:
 - Historiquement : sigmoid, tanh
 - Très bon résultats avec ReLU (Rectified Linear Unit)

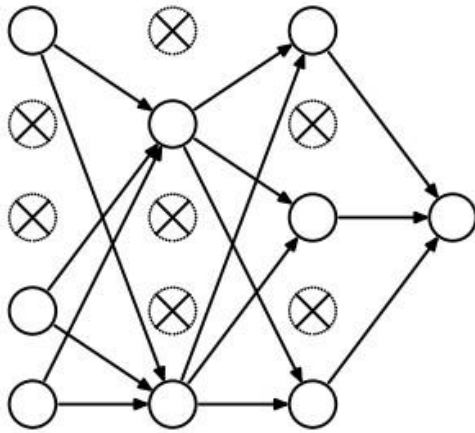


Rétro-propagation du gradient

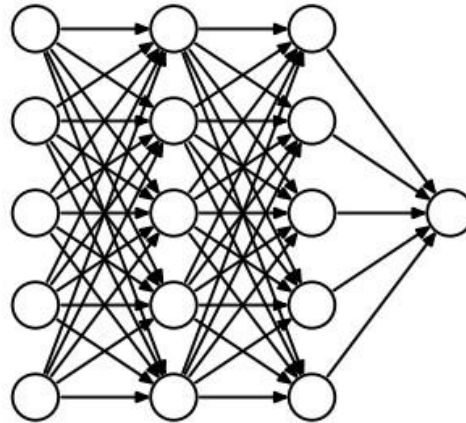
- ⊙ Calcul du gradient par rétro-propagation du gradient
- ⊙ Application de la règle de dérivation des fonctions composées



- Facilement beaucoup de paramètres : overfitting !
- Technique de régularisation : Dropout



Entraînement :
 $\text{prob_keep} = 0.6$

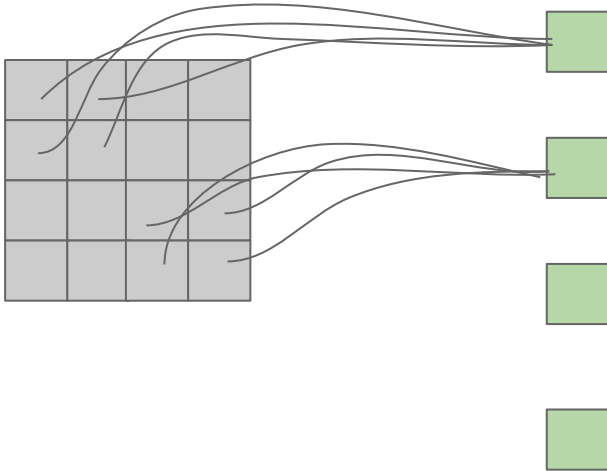


Inférence :
 $\text{prob_keep} = 0.6$

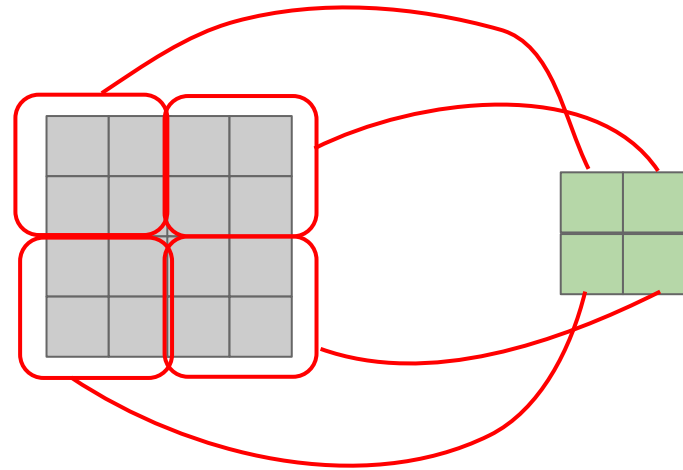
Réseaux de neurones convolutifs (CNNs)

DNNs profonds : beaucoup trop de paramètres !

Tirer profit de la forte corrélation entre pixels voisins



Partager les poids !

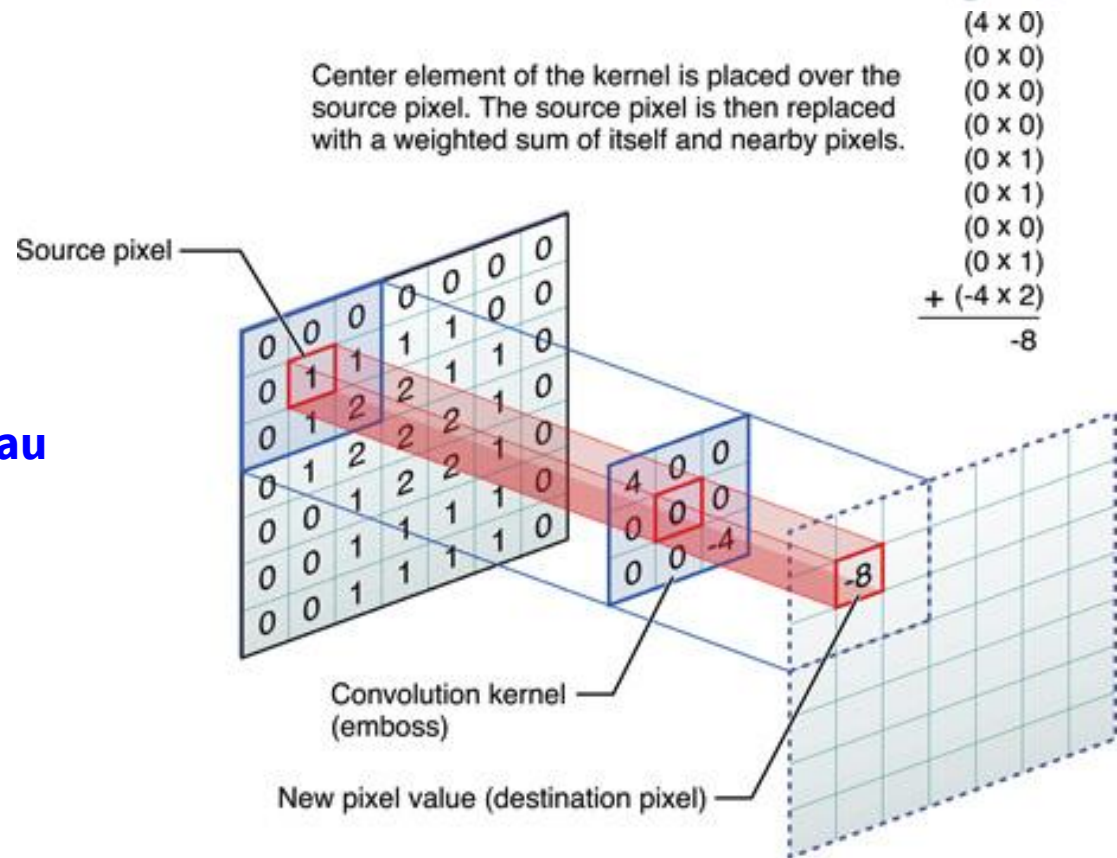


On retrouve tout ça avec des **convolutions** !

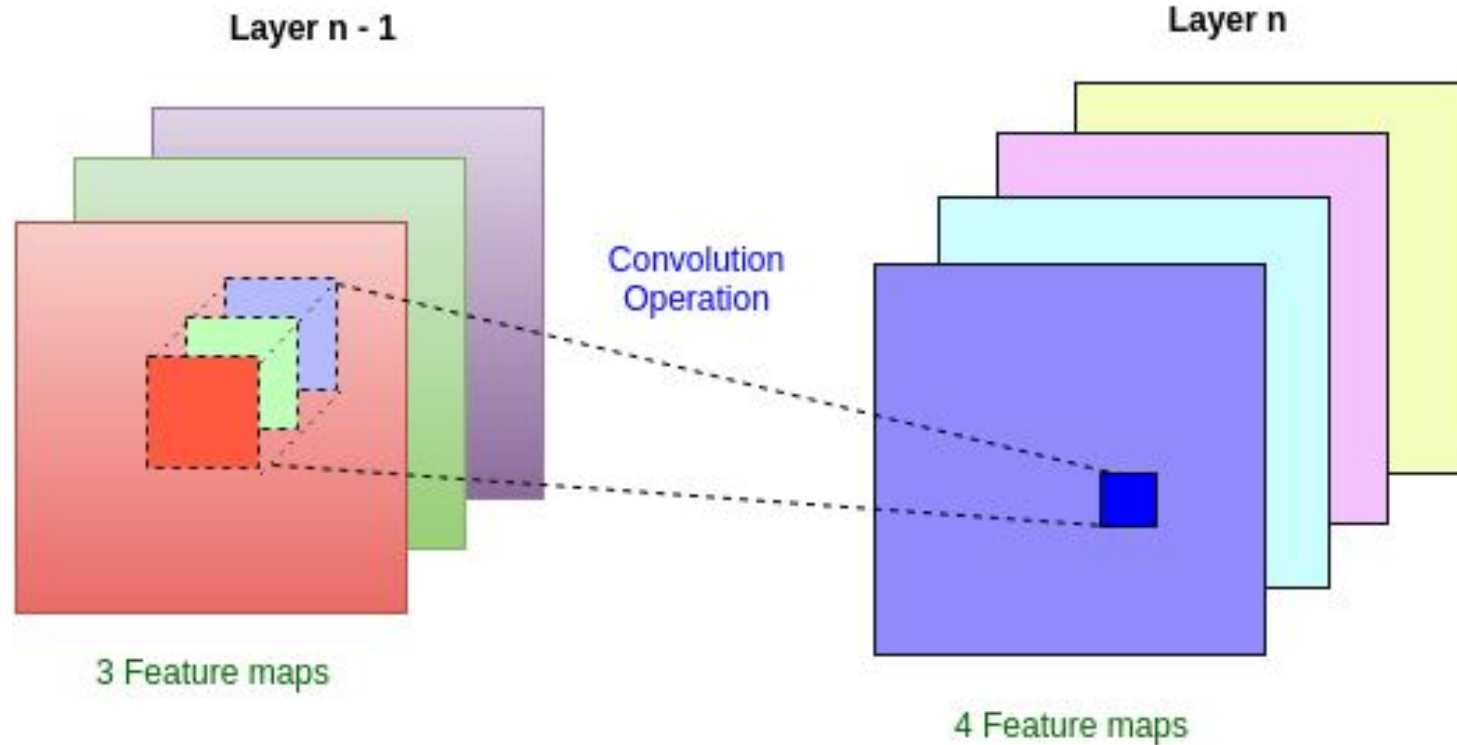
On déplace sur l'image un noyau de convolution dont les poids sont les paramètres entraînables !

‘Stride’ : façon dont le noyau se déplace sur l'image !

Ex : (1,1) ou (2,2)



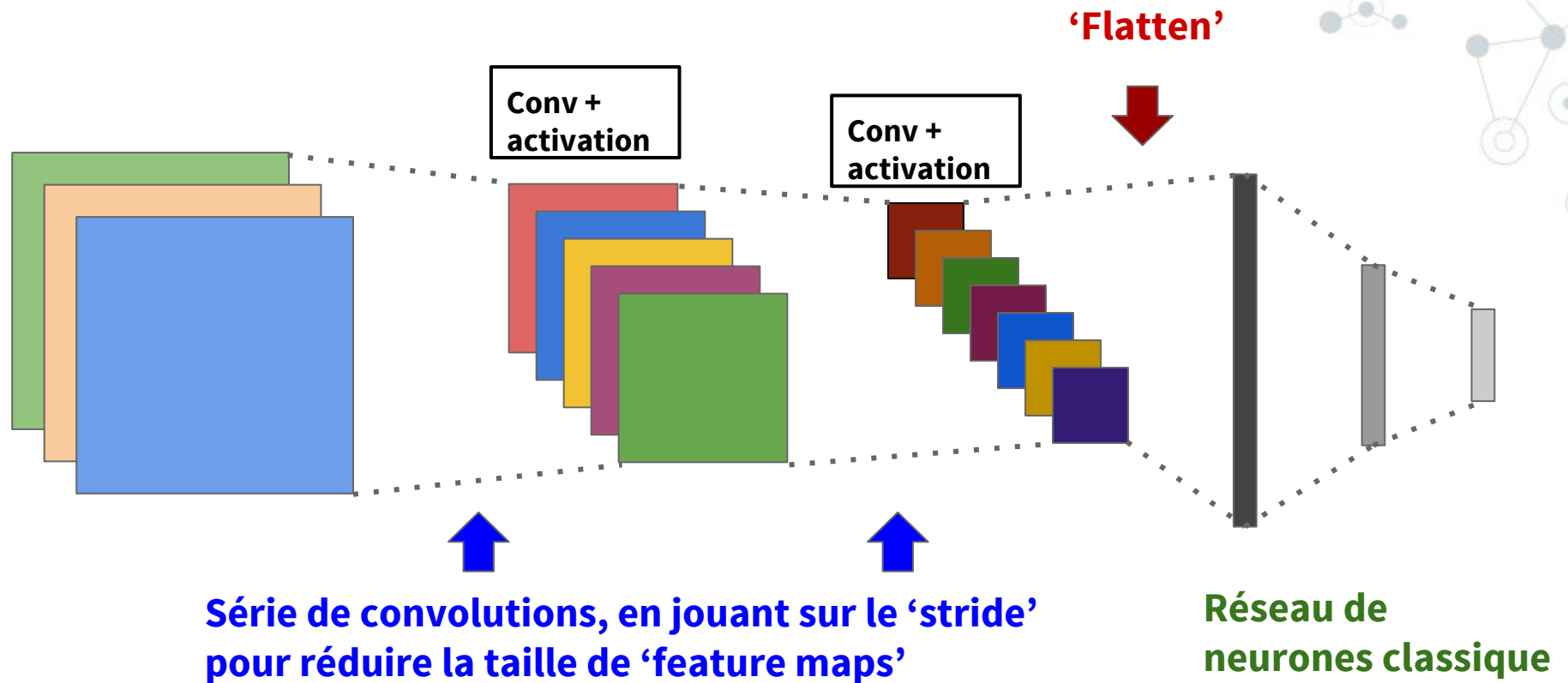
On crée des “Features Maps”



Ici 4 noyaux 3D, pour passer d'une couche avec 3 feature maps (RGB par exemple) à une couche avec 4 feature maps !

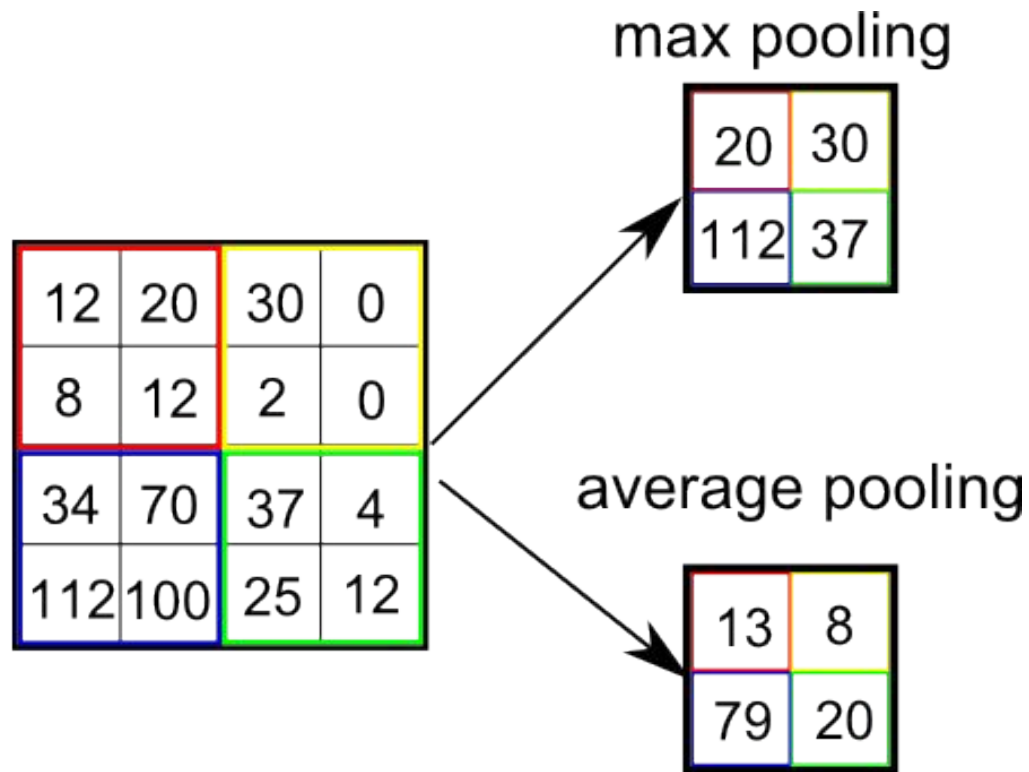
$(3 \times 3 \times 3) \times 4$: 108 paramètres !

Modèle simple : séries de convolutions + réseau classique



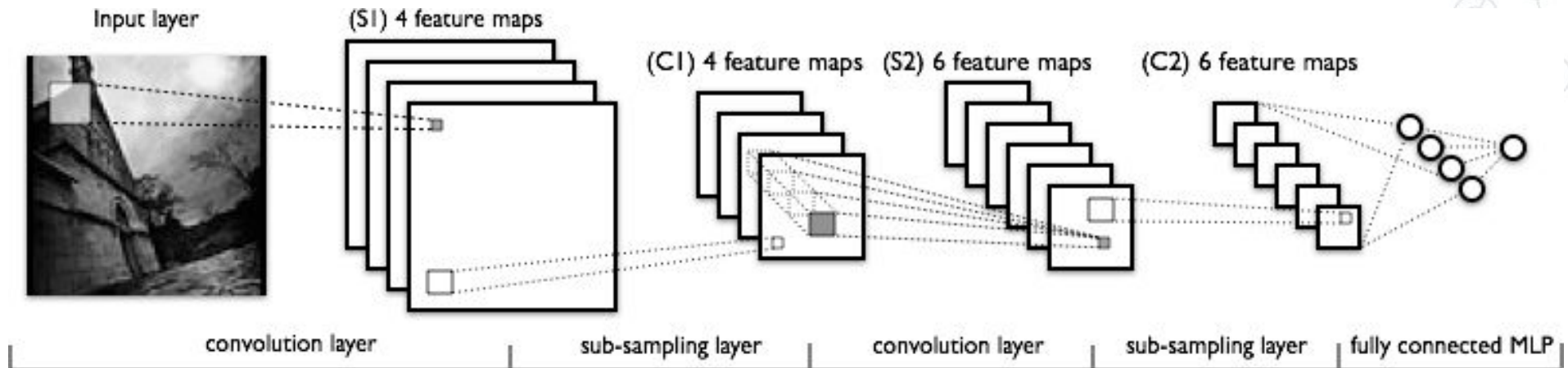
Pooling sur les Feature Maps

(après la fonction d'activation)



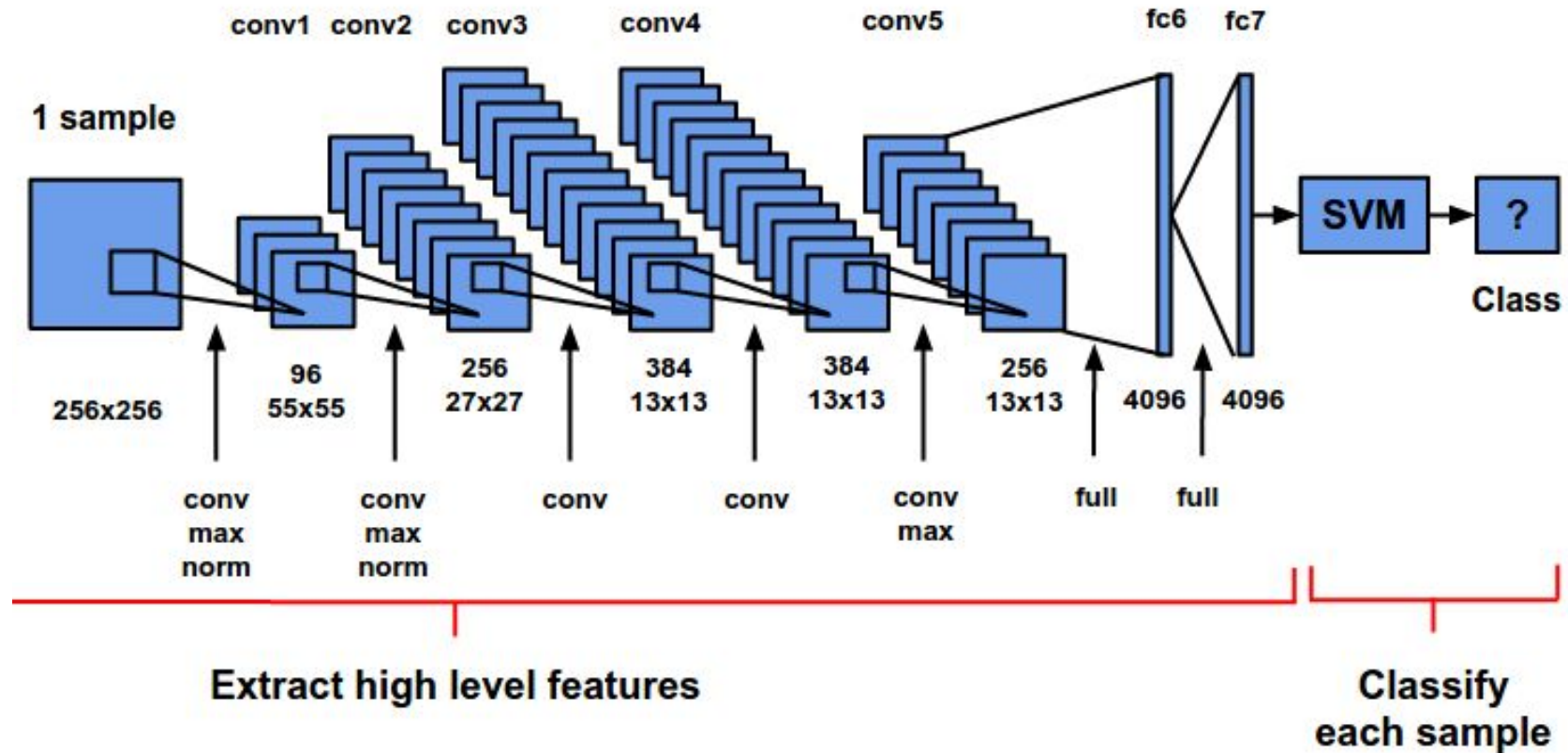
Permet de réduire la dimension des Feature Maps

Premier CNN : LeNet-5 (1989)



Application à la reconnaissance de caractères manuscrits sur les chèques !

AlexNet - 2012



Vainqueur de la compétition ImageNet (1000 classes)

Machine Learning vs Deep Learning



Traditional Machine Learning Flow



Deep Learning Flow

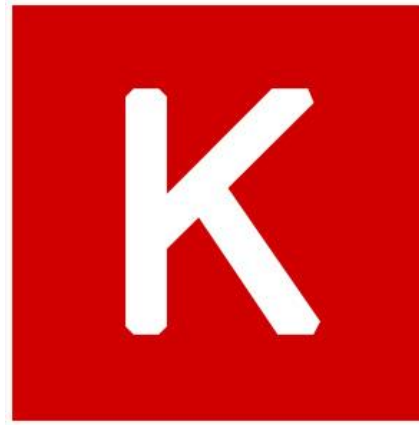
D'un point de vue pratique ?

- ◎ Beaucoup de mathématiques (dérivation, convolutions, des produits matriciels)
- ◎ Fonctionnement par 'batch' : possibilité d'effectuer des calculs en parallèle
- ◎ Algorithme itératif

 Choisir une bonne librairie/framework

- Matériel ?
- Environnement ?
- Embarqué ?
- Temps réel ?

Keras.io



Librairie Python pour le Deep Learning

- Utilise Theano ou Tensorflow, de façon indifférente
- Haut niveau ! (au dessus de Theano ou Tensorflow)
- Rapidité de développement: on ajoute les couches les unes aux autres !
- Très flexible et modulaire ! (multiple input / output)
- CPU ou GPU de façon transparente !
- Problèmes de mémoire
- messages d'erreur difficilement utilisables

Theano (ou TensorFlow)

- définir et évaluer des expressions mathématiques
- représentation de l'expression avec un graphe de variables symboliques (tensors)
- Générer et compiler du code optimisé pour le CPU (C++) ou GPU (CUDA)
- CPU/GPU transparent (drivers + lib CUDA + lib NVIDIA + GPU activé)
- dérivation automatique (très utile pour le gradient)
- syntaxe proche de Numpy

Theano et TensorFlow sont très similaires !

TensorFlow permet plus facilement de faire du Deep Learning

Theano plus rapide (toujours le cas ?)

Aujourd'hui : Backend = THEANO

Keras.io

Sans Keras : Des problèmes de tuyauterie entre chaque couche !

Keras :

- des routines d'entrainements (Minibatch SGD, RMSprop, Adagrad, ...)
- des outils pour gérer les dataset de grande taille (> 5Go)
- rapidité pour créer la fonction
- un large catalogue de couches déjà codées
- possibilité d'en rajouter facilement (pour les deux backends ou uniquement un)

Avantages :

- Documentation de très bonne qualité
- Communauté
- Largement utilisée sur Kaggle

Et les autres ?

- Caffe (+ Nvidia DIGIT) (C++, API pour Matlab et Python)
- Torch (Facebook) (Lua)
- CNTK (Microsoft)
- deeplearning4j (Java)

D'autres libraries similaires à Keras : Pylearn2, Blocks, Fuel, Lasagne

Performances ? Très difficile de comparer correctement !!

Les versions des drivers et des librairies Cuda (comme cuDNN) comptent beaucoup !

Neon : la plus rapide ?

Environnement de travail

- Documentation de Keras
- Répertoires avec les données déjà préparées (souvent une réduction !) + scripts pour la préparation
- Les notebooks Jupyter
- Les slides
- Les Dockerfiles (CPU ou GPU)

Aller plus loin ?

- Bien comprendre le fonctionnement de Theano (ou de TensorFlow)
- D'autres algorithmes d'optimisation (RMSprop, Adagrad, ...)
- D'autres modèles profonds : VGG16, GoogleNet, ResNet-152, GANs
- D'autres applications avec les images : segmentation, détection, génération
- D'autres données : vidéo, son, texte, multi-modale (image + texte), optimisation combinatoire
- Vers l'embarqué ?