Introduction au Deep Learning

CONSULTING BSTORM

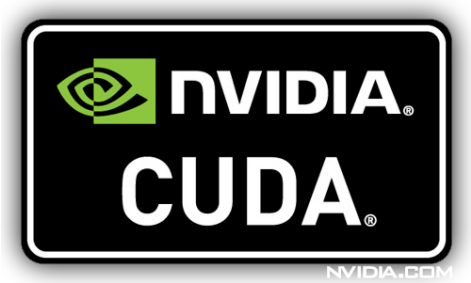
1. Introduction

1957 → Naissance du Perceptron

Depuis 2015 → Explosion des applications de Deep Learning :

- Vision par ordinateur
- Médecine
- Finance
- …

1. Introduction



PyTorch

1. Introduction

Puissance de calcul :

CPU *vs* **GPU**

→ **CPUs** sont très efficaces pour exécuter une succession d'opérations

→ **GPUs** sont très efficaces pour exécuter des opérations **simultanément**

Le Deep Learning est **hautement parallélisable**

1. Introduction

Les données :

→ Internet a rendu possible l'**échange** de données

→ Intérêt croissant pour les données et leur collecte

→ Nouvelles technologies (e.g. NoSQL)

Le Deep Learning nécessite beaucoup de données pour être efficace **><** Machine Learning classique

1. Introduction

Les librairies **Open Source** :

TensorFlow (Google) :

- Première version en 2015
- Seconde version (TF 2.0) en septembre 2019

PyTorch (Facebook) :

- Première version en 2016

→ Outils très jeunes !

1. Introduction

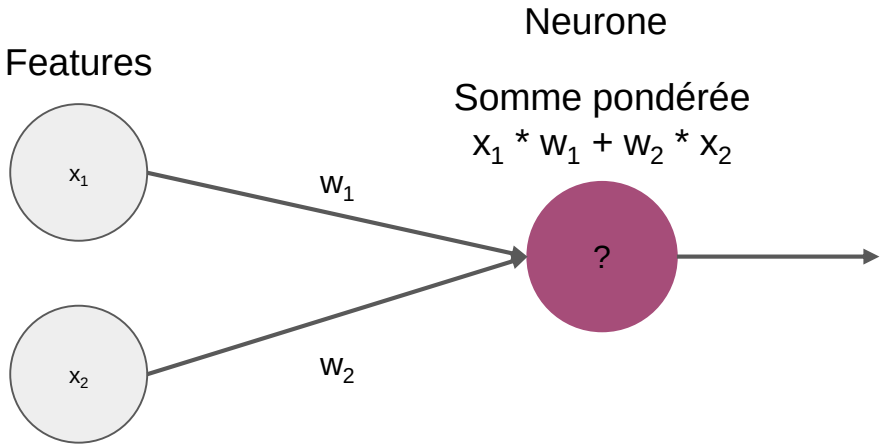
Le Deep Learning pour :

→ De gros volume de données

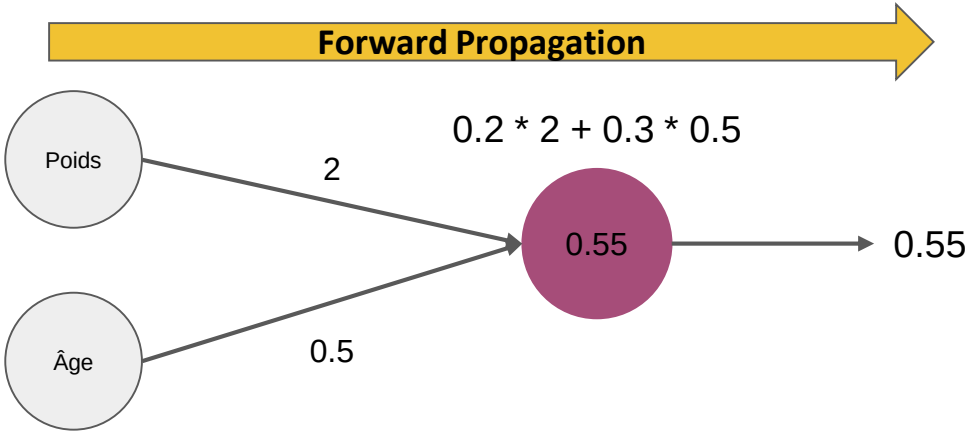
→ Des données peu ou pas structurées (image, son, texte)

→ Pour générer de nouvelles données

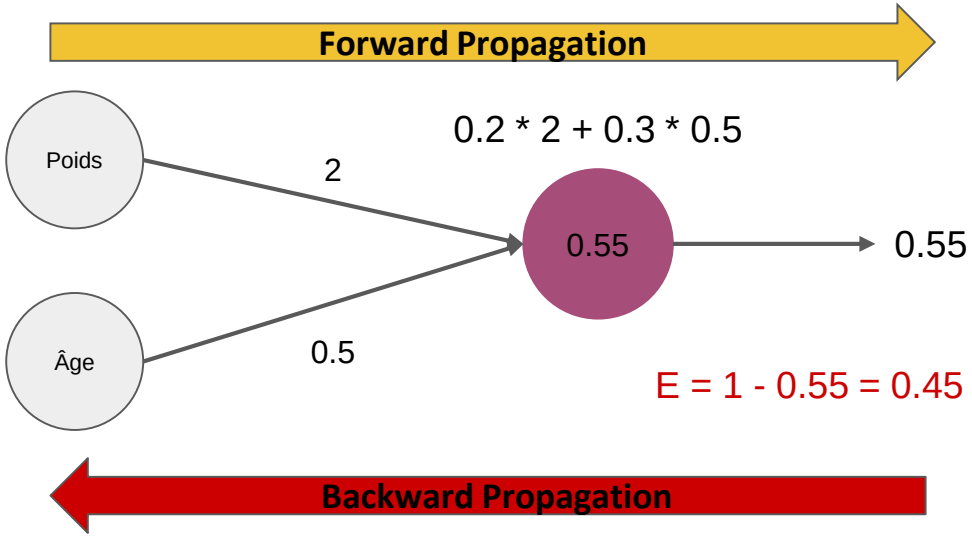
2. Le neurone



2. Le neurone



2. Le neurone

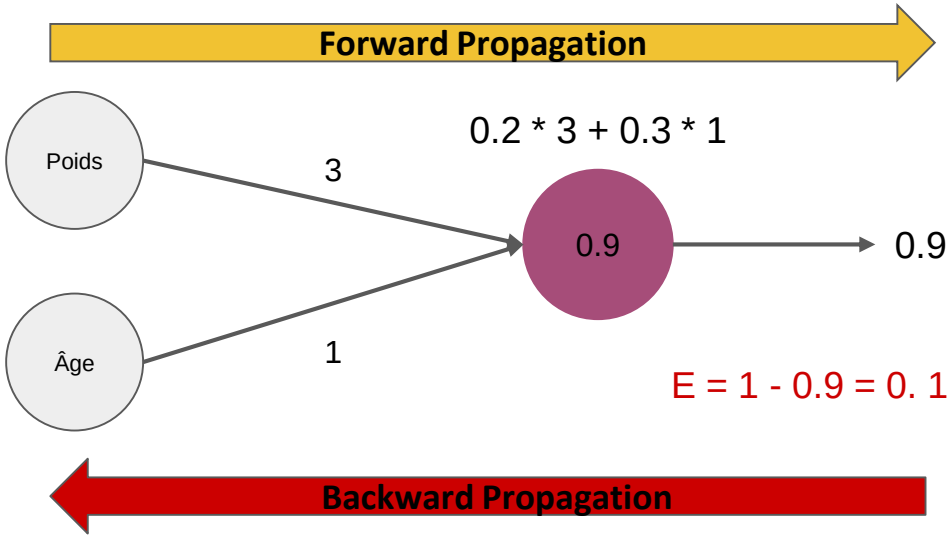


On calcule une erreur de manière à modifier nos poids

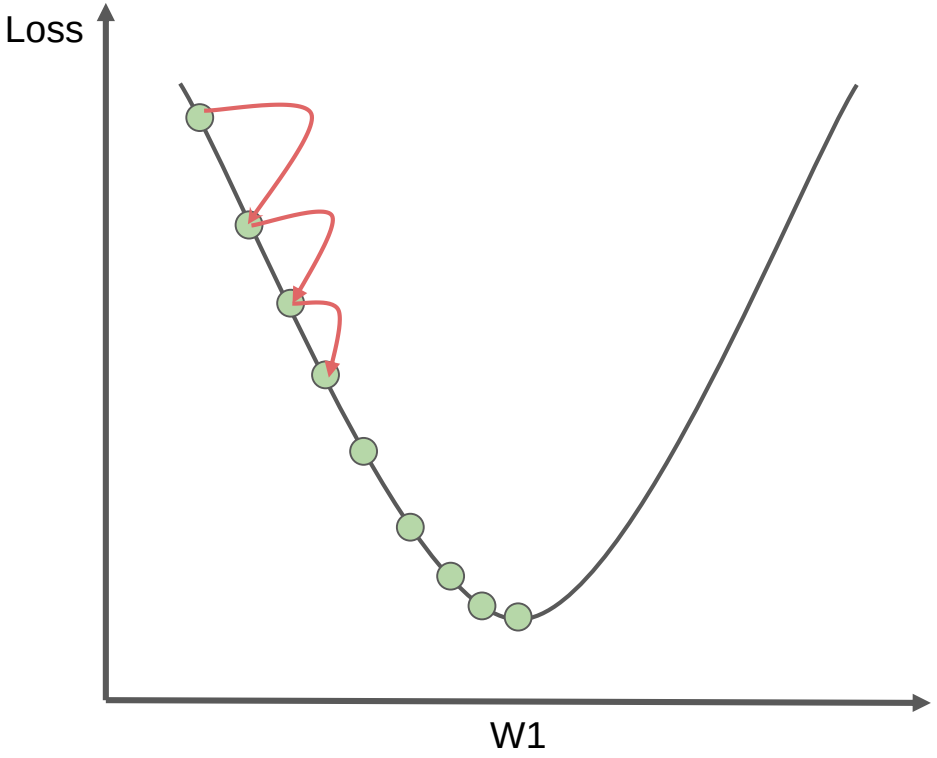
On répète cette opération jusqu'à obtenir de petites erreurs

→ Descente de gradient

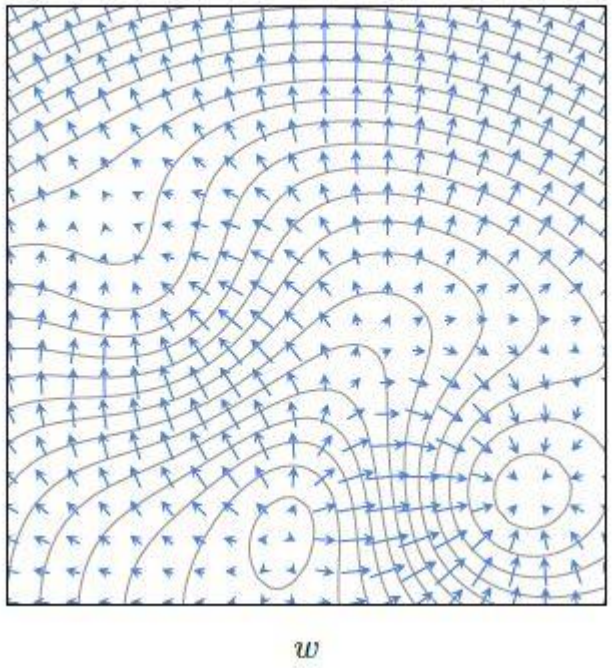
2. Le neurone

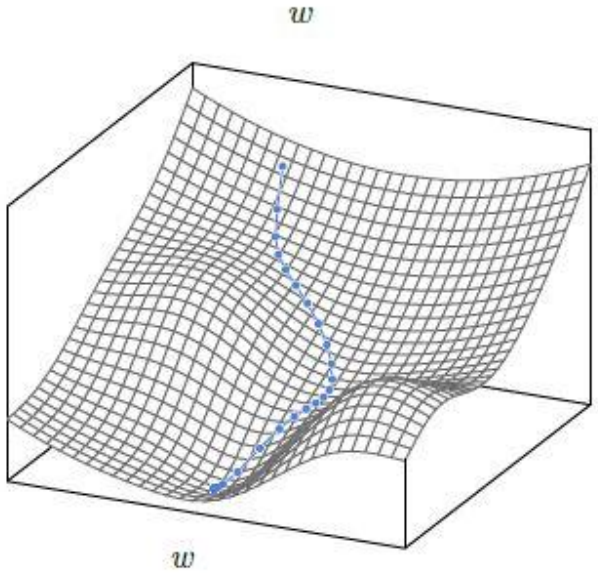

Forward Propagation
0.2 * 3 + 0.3 * 1
Poids
3
0.9
0.9
Âge
1
E = 1 - 0.9 = 0. 1
Backward Propagation

2. Le neurone



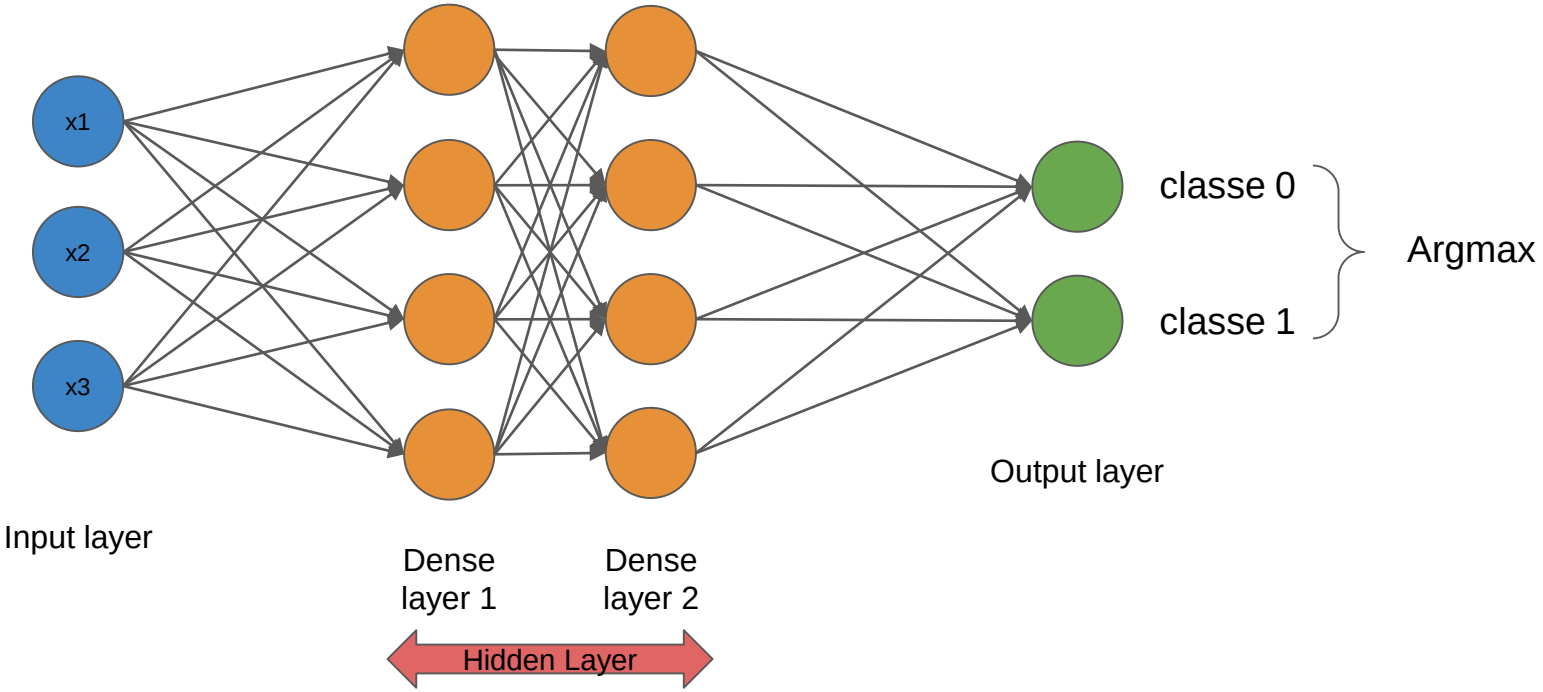
2. Le neurone





"The Little Book of Deep Learning", François Fleuret, v1.0 - 27/06/2023, p. 35

3. Les réseaux de neurones



3. Les réseaux de neurones

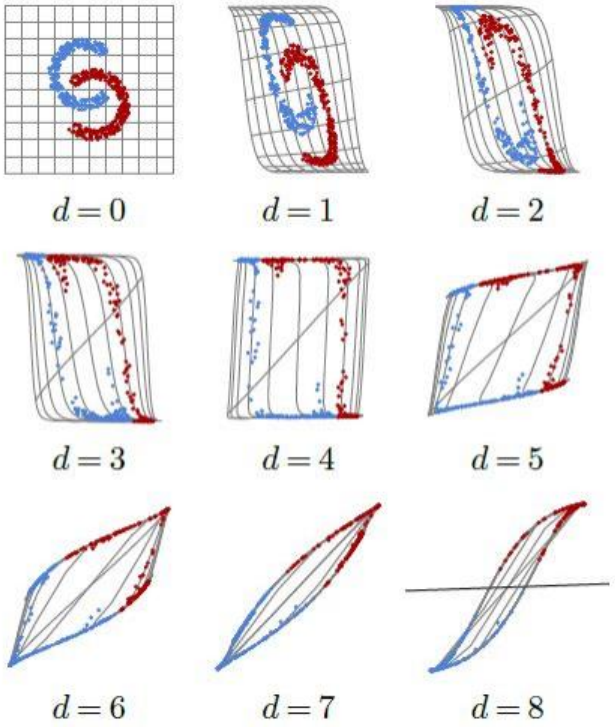


Figure 3.4: *Each plot shows the deformation of the space and the resulting positioning of the training points in $\mathbb{R}^2$ after d layers of processing, starting with the input to the model itself (top-left). The oblique line in the last plot (bottom-right) shows the final affine decision.*

"The Little Book of Deep Learning", François Fleuret, v1.0 - 27/06/2023, p. 44

4. Batch, batch size, epoch

En pratique, on ne calcule pas une erreur après chaque sample, on préfère travailler par batch de samples avant de calculer une erreur et de corriger les poids.

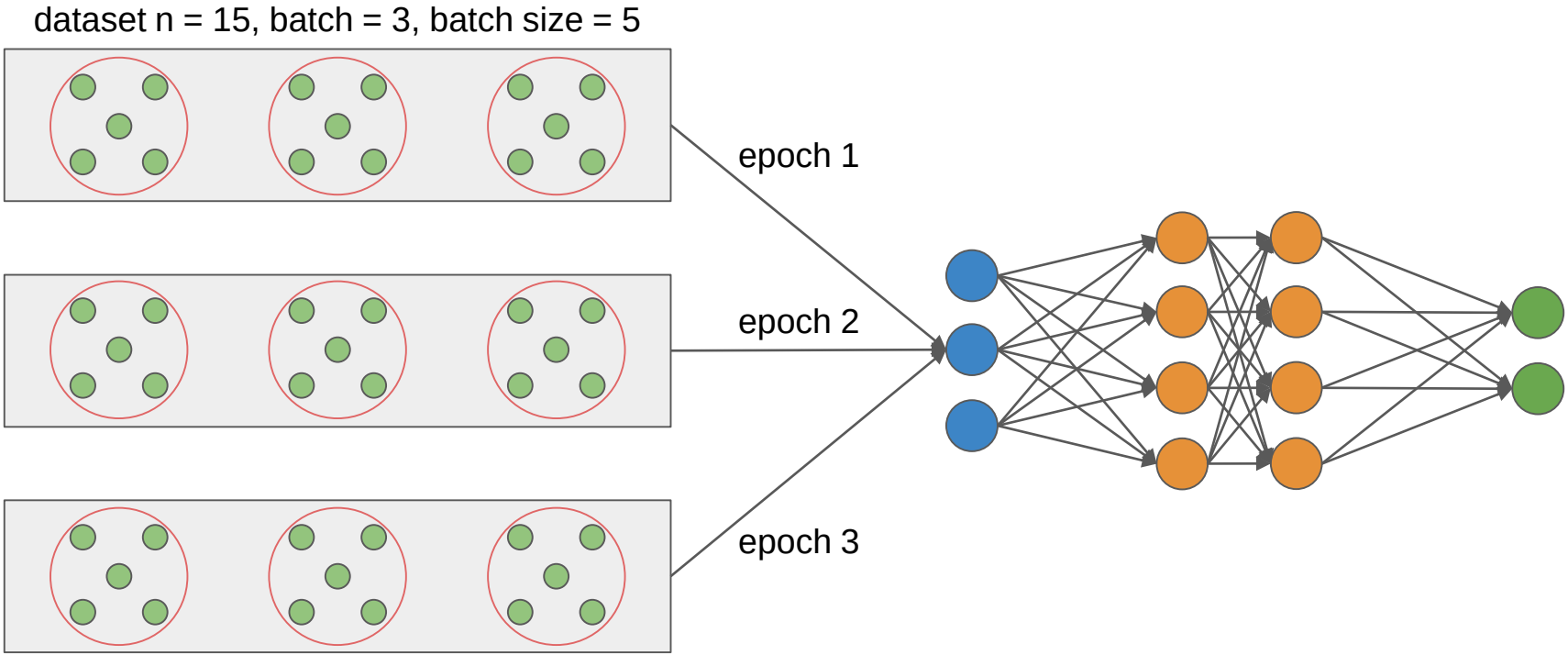
batch = un sous ensemble du dataset

batch size = le nombre de samples par batch

On passe aussi plusieurs fois notre dataset dans le réseau.

epochs = le nombre de fois que le réseau ‘voit’ les données

4. Batch, batch size, epoch



5. Les fonctions d'activation

Nous avons vu comment calculer la somme pondérée

Il est possible de moduler cette somme avec des fonctions d'activations :

- Linéaire
- Sigmoïde
- Softmax
- Tangente hyperbolique
- ReLU (Rectified Linear Unit)
- Leaky ReLU

Pourquoi ? → Non linéarité, normalisation des sommes pondérées

5.1. Sigmoïde et Softmax

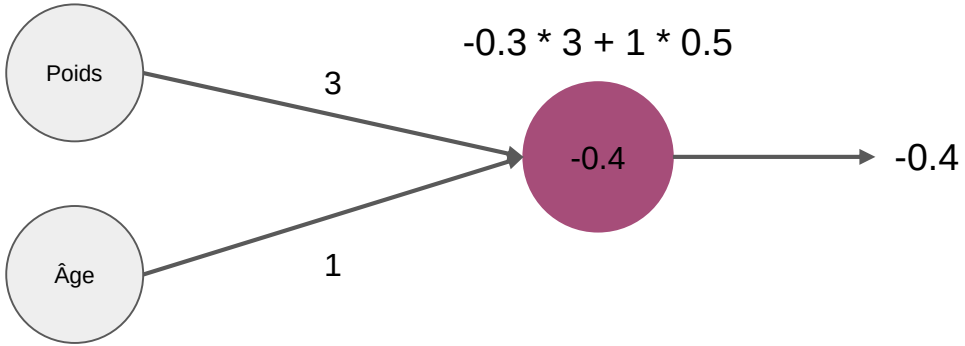
Dans notre exemple du Perceptron, nous partions du principe que la valeur est sortie était comprise entre 0 et 1

En réalité, la sortie d'un neurone n'est pas par défaut comprise entre 0 et 1

Une autre question se pose également, que faire si nous avons plus que deux classes ?

→ Il faut pouvoir normaliser la sortie à l'aide d'une fonction d'activation

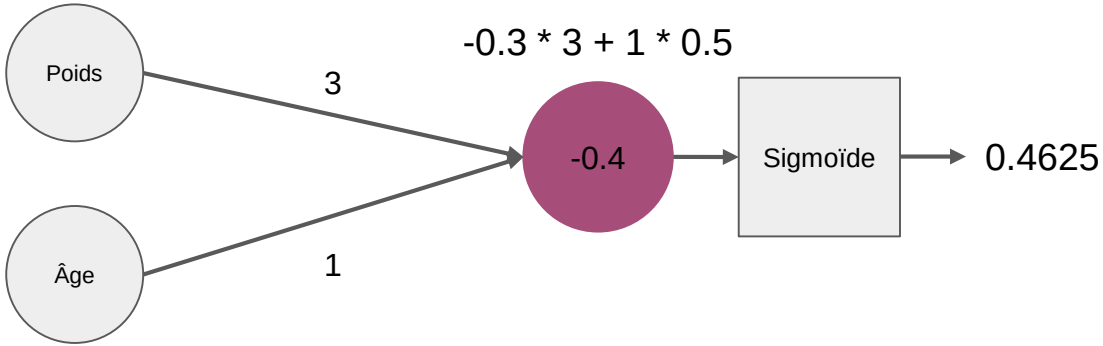
5.1. Sigmoïde et Softmax



Si on change la valeur de nos entrées on obtient -0.4

→ Difficile à interpréter

5.1. Sigmoïde et Softmax

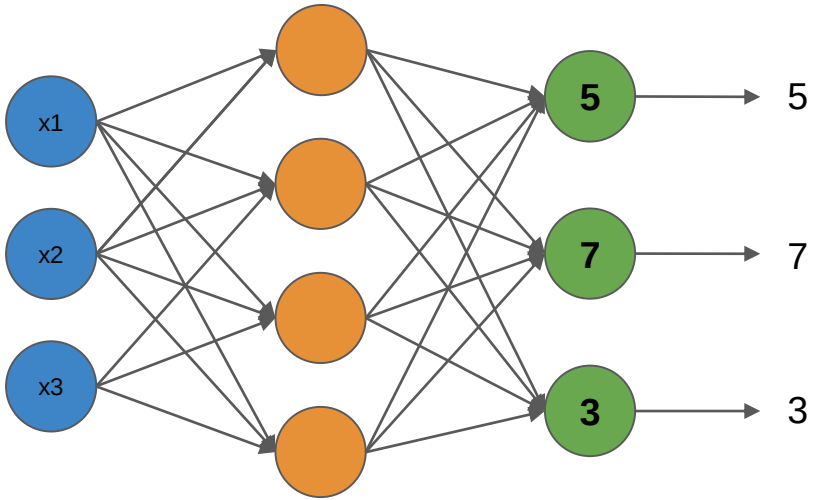


La fonction sigmoïde permet de remettre les valeurs sur une échelle comprise entre 0 et 1

→ Interprétable

→ Classification binaire (soit 0, soit 1)

5.1. Sigmoïde et Softmax

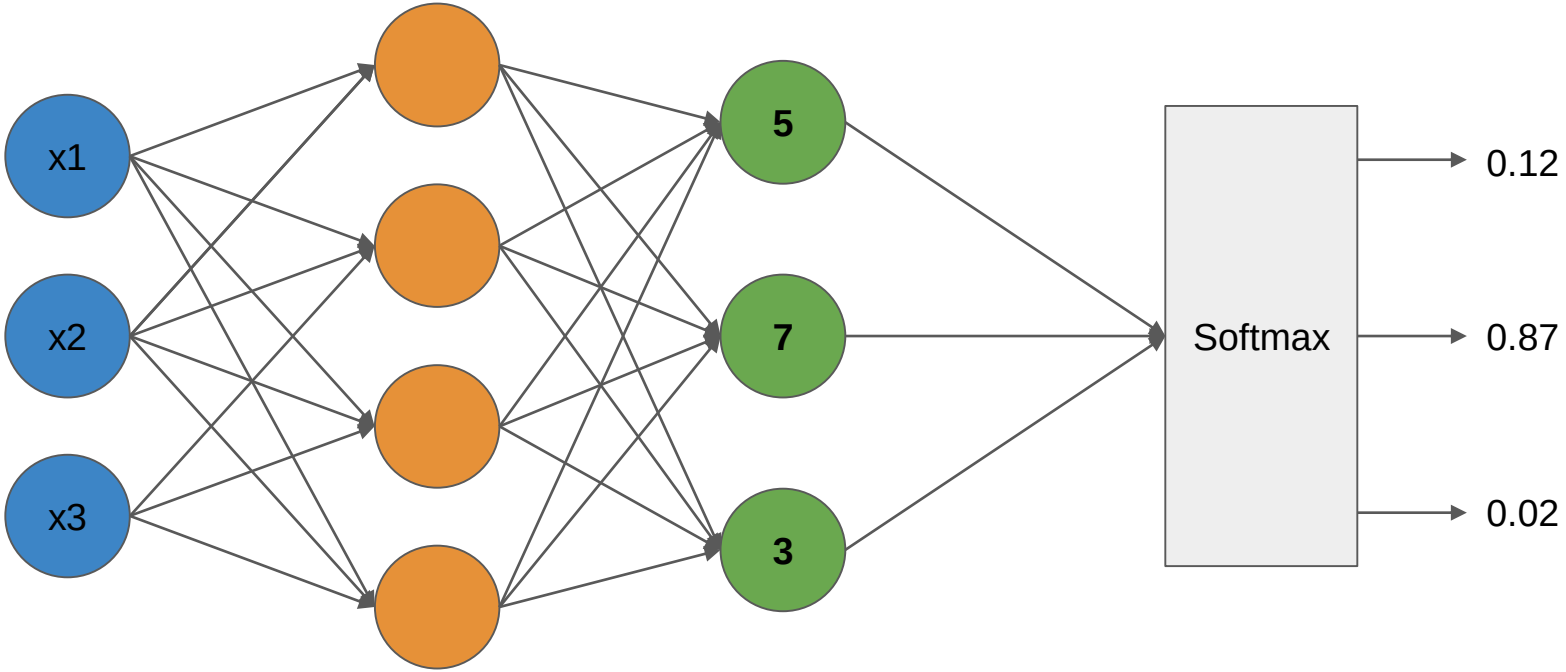


On ne peut plus utiliser la fonction sigmoïde au delà de 2 classes.

Il faut trouver une autre solution pour exprimer ceci sous forme de probabilité

→ Softmax

5.1. Sigmoïde et Softmax

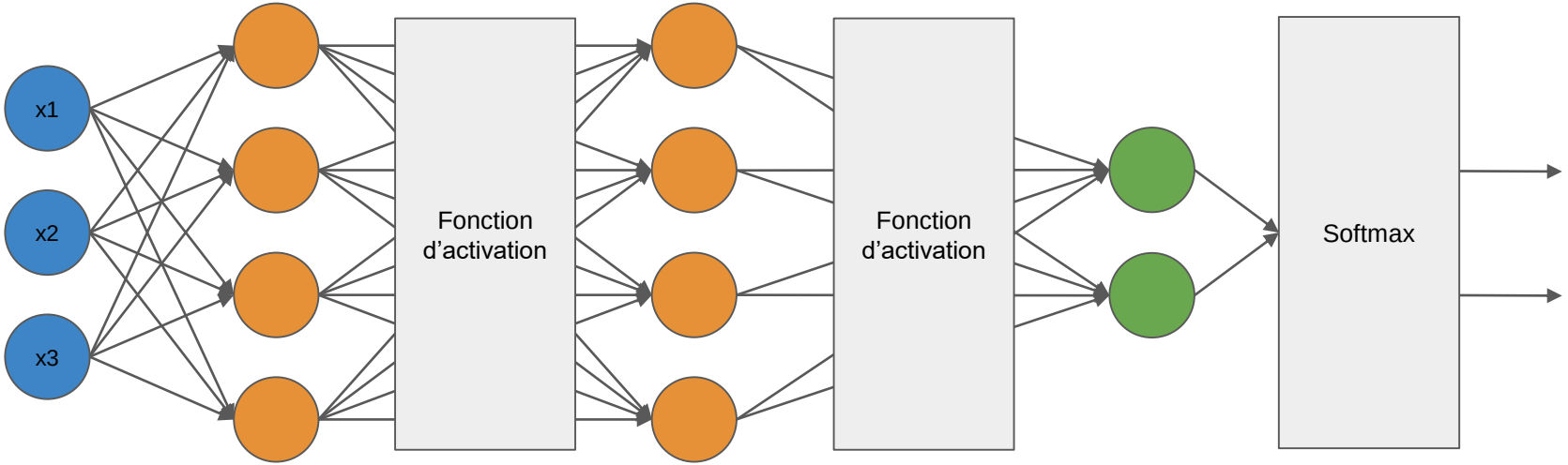


5.2. Non linéarité

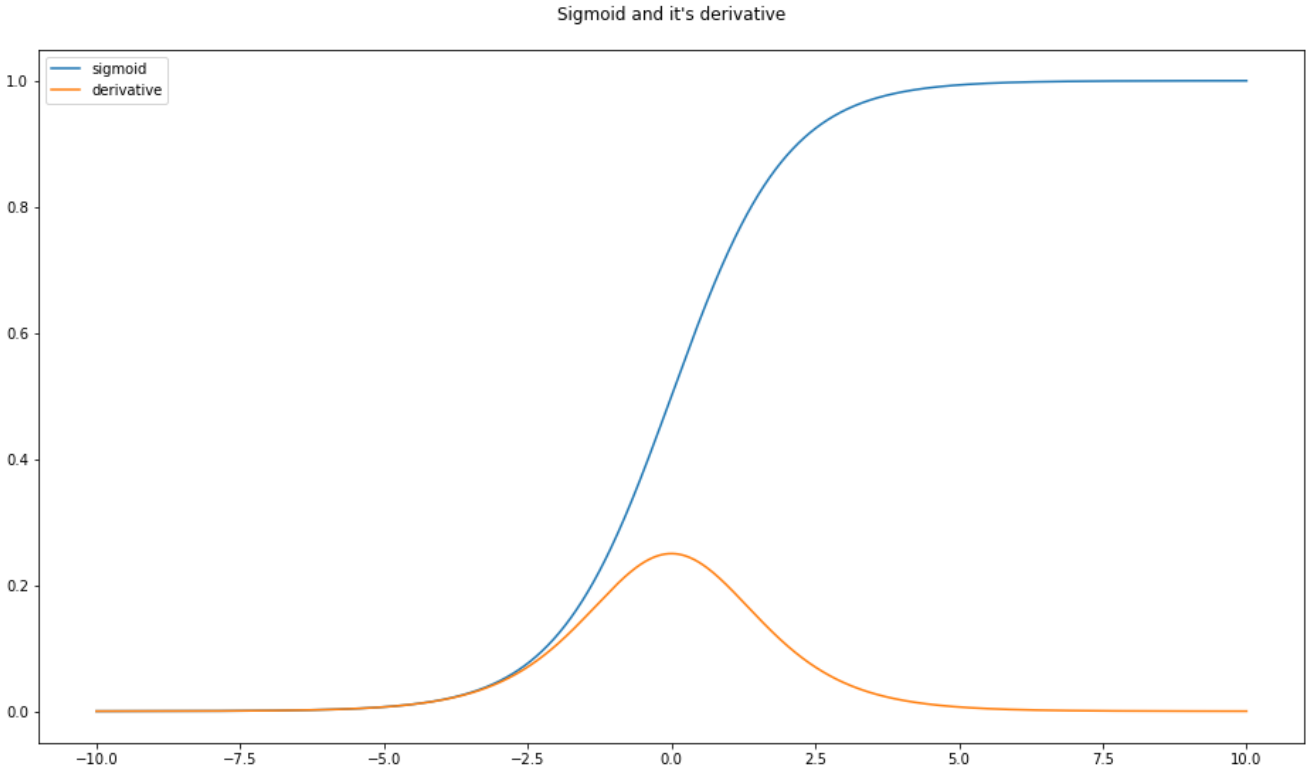
Jusqu'ici on utilisait les fonctions uniquement pour moduler la couche de sortie, mais on peut également les utiliser pour moduler les sorties des couches cachées

→ Pour apporter de la non linéarité

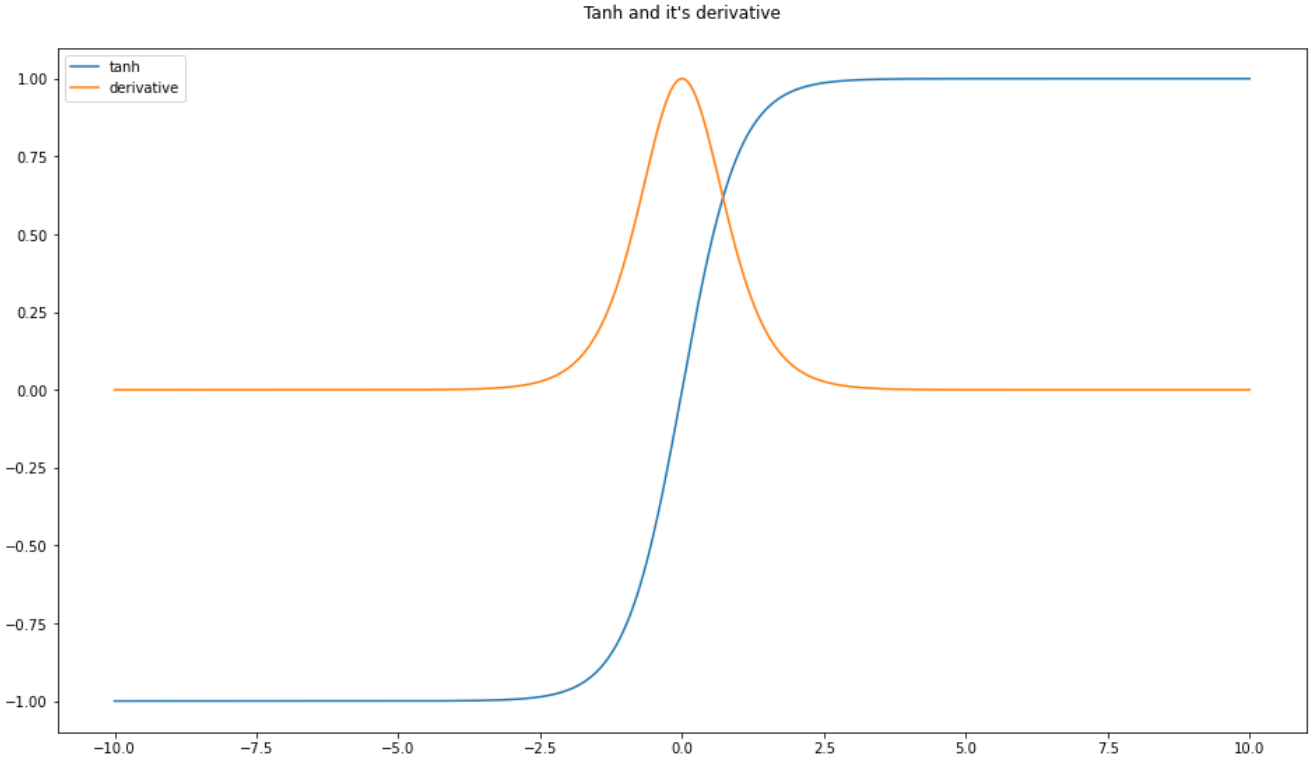
5.2. Non linéarité



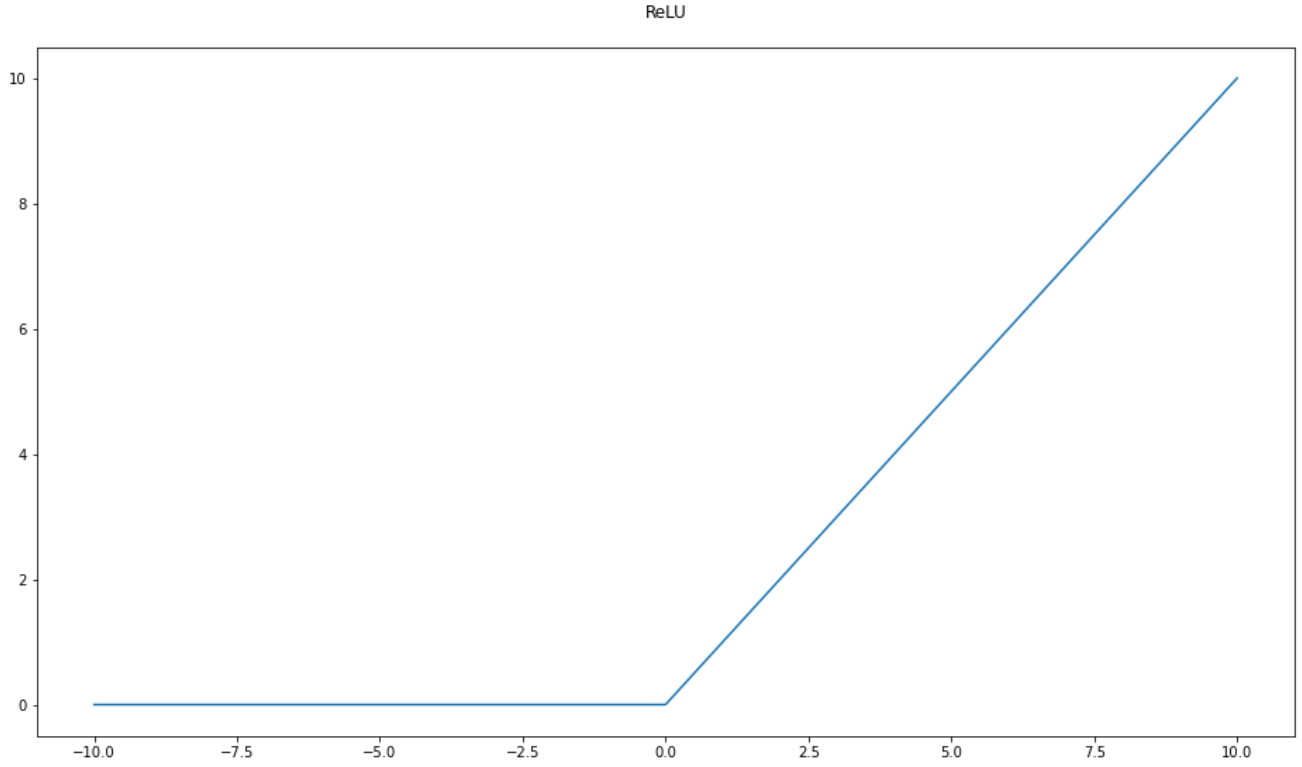
5.2. Non linéarité



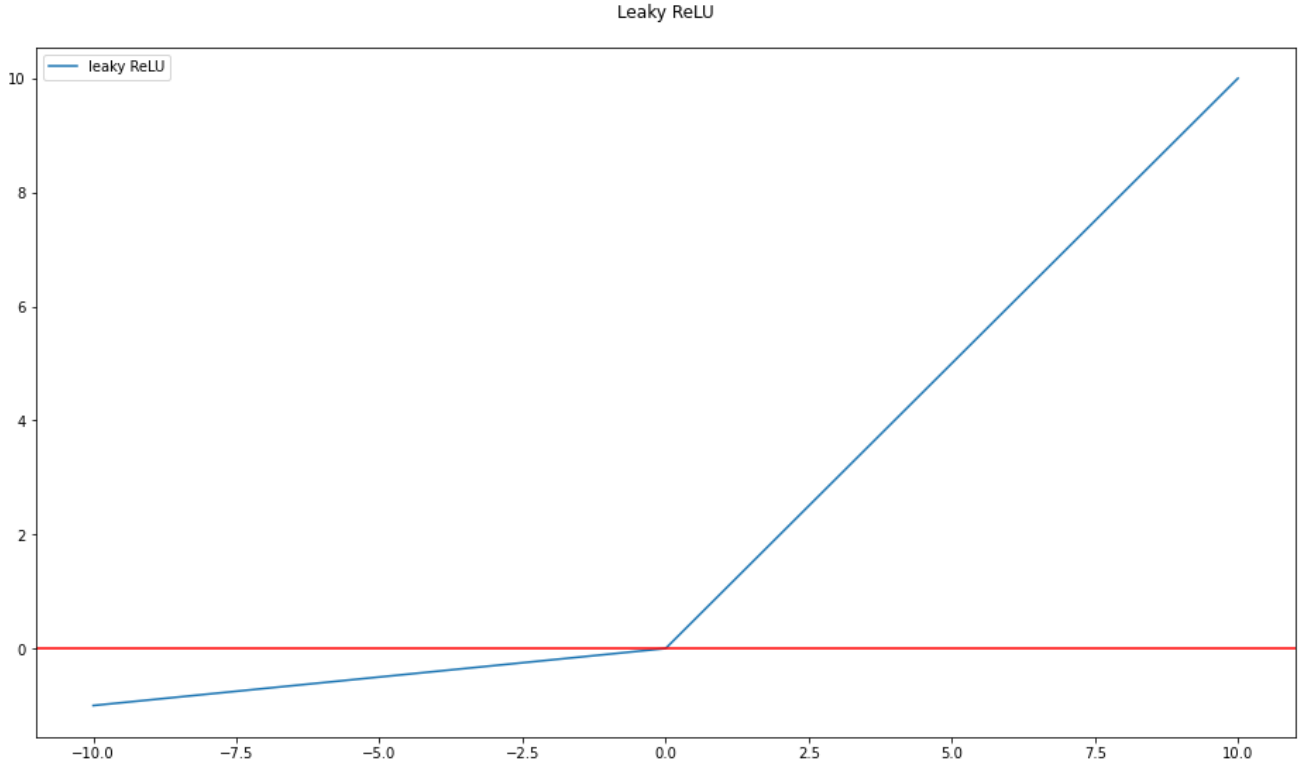
5.2. Non linéarité



5.2. Non linéarité



5.2. Non linéarité



6. Les architectures

Il existe des architectures de réseaux de neurones particulières :

→ Convolution Neural Network (**CNN**)

→ Recurrent Neural Network (**RNN**)

→ Long Short Term Memory (**LSTM**)

→ Generative Adversarial Network (**GAN**)

→ ...

CNN

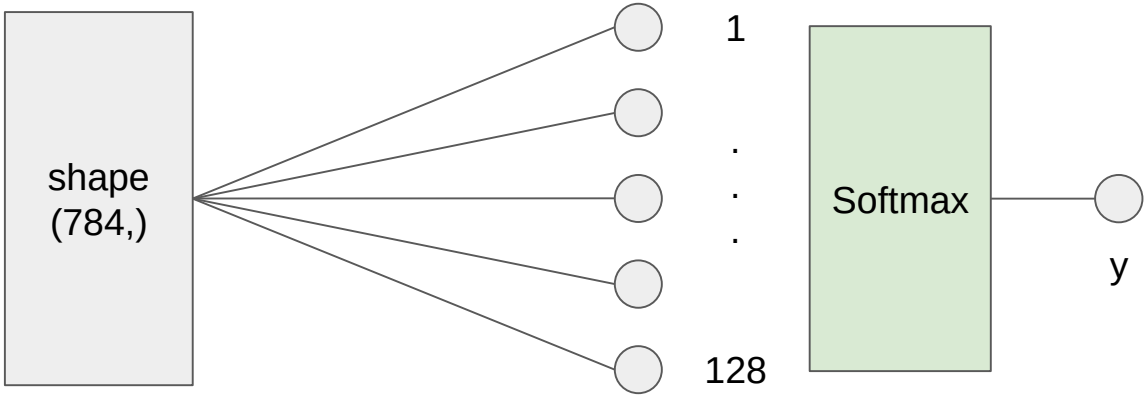
Convolutional Neural Networks

CONSULTING BSTORM

1. Introduction

Fashion mnist noir et blanc :

- shape = (28,28,1)



1. Introduction

L'approche avec de simples couches denses pose un problème. En effet, le nombre de paramètres à calculer explose très vite :

- Des images avec une shape (28,28,1) connectées à un couche dense de 128 unités : 28 * 28 * 1 * 128 + 128 = 100 480 paramètres

- Image shape (28,28,3) : 28 * 28 * 3 * 128 + 128 = 301 184 paramètres

- Image shape = (224,224,3) : 224 * 224 * 3 * 128 + 128 = 19 267 712

1. Introduction

Les CNN permettent d'extraire l'information utile tout en réduisant la quantité d'information à traiter. Un CNN est principalement composé de deux opérations :

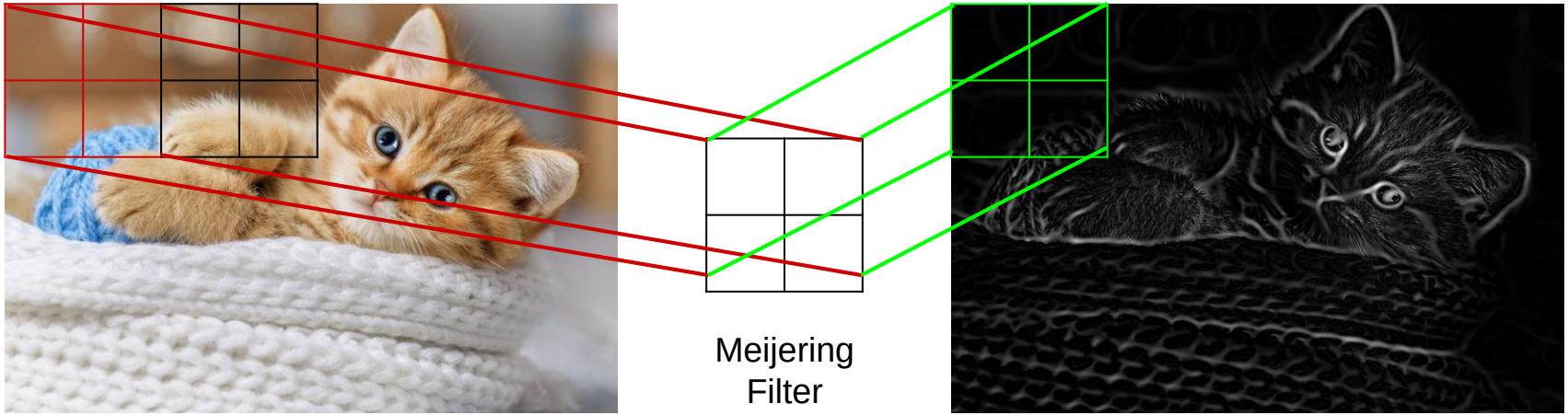
1. Une opération avec les couches de convolutions qui doit permettre d'extraire l'information utile. Distinguer le bruit de l'information.

2. Une opération de pooling qui permet de drastiquement réduire la quantité d'information.

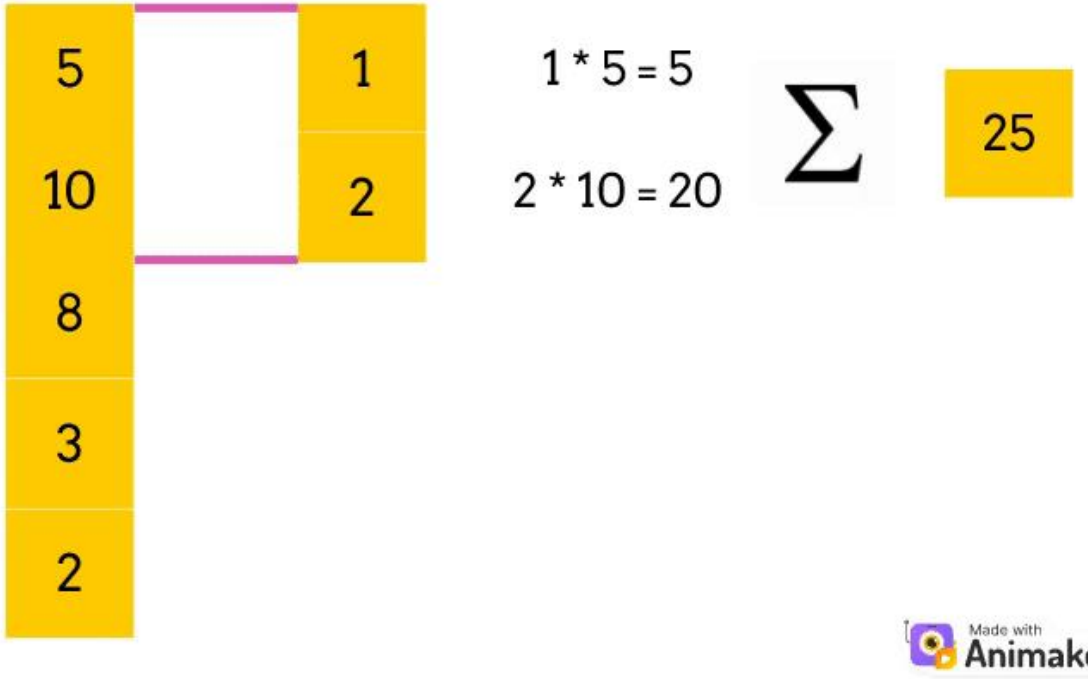
2. La convolution

Une convolution désigne le fait de balayer une image avec un filtre.

Ce filtre doit permettre de faire ressortir certains éléments.



2. La convolution



2. La convolution

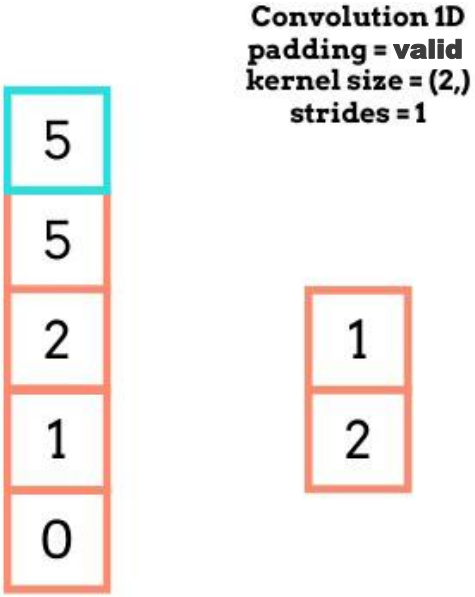
Quelques notions importantes pour une opération de convolution :

- kernel_size : au plus cette valeur est petite, au plus on capture une information locale

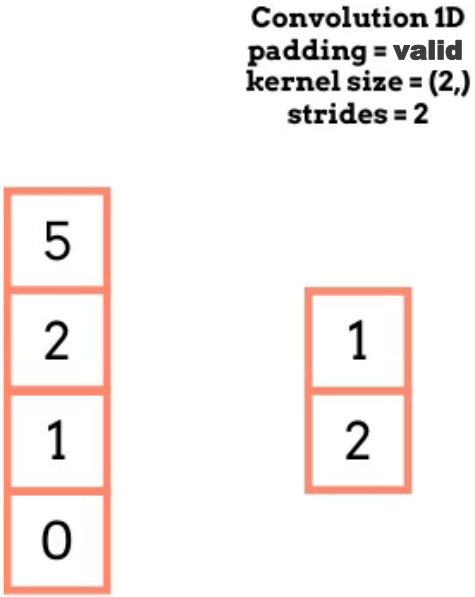
- Stride : désigne le pas. Un stride plus élevé diminue la quantité d'informations en sortie et peut jouer le rôle de pooling (*cfr.* infra)

- Padding : permet de conserver un taille d'entrée et de sortie constante pour un stride de 1

2. La convolution

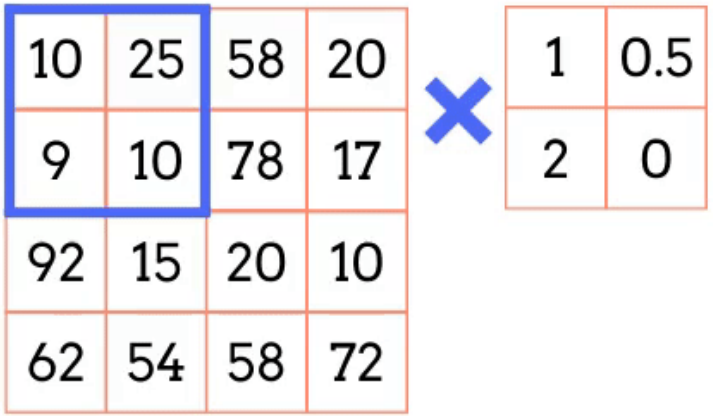


2. La convolution



2. La convolution

Convolution 1D
padding = valid
kernel size = (2, 2)
strides = 1



2. La convolution

Convolution 2D
padding = same
kernel size = (2, 2)
strides = 1

0	0	0	0	0
0	10	25	58	20
0	9	10	78	17
0	92	15	20	10
0	62	54	58	72

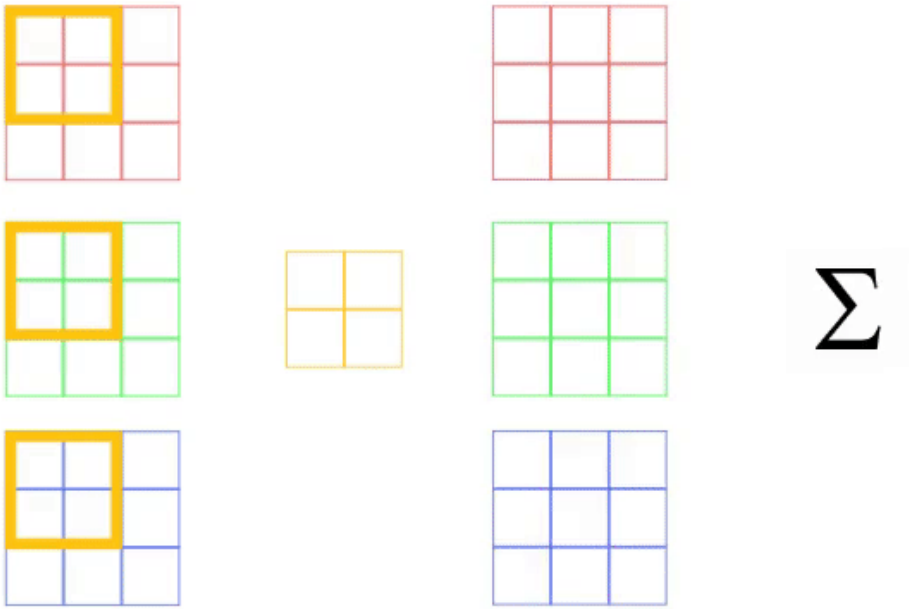
1	0.5
2	0

2. La convolution

Dans le cas des images, il s'agit bien de convolutions 2D même avec des images sur plusieurs canaux (avec couleurs par exemple)

Chacun des canaux est scanné indépendamment des autres avec le même kernel et ensuite les canaux sont sommés.

2. La convolution



3. Le pooling

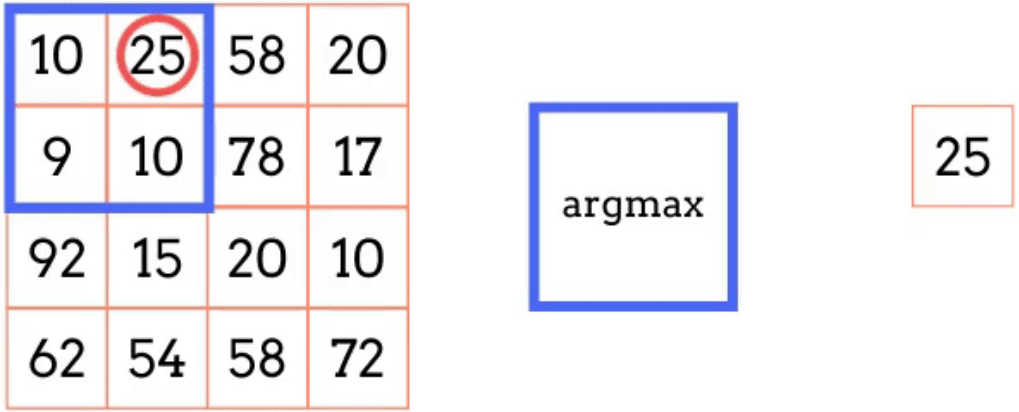
L’opération de pooling est en réalité également une opération de convolution mais avec deux différences majeures :

1. La fonction utilisée n’est pas un produit scalaire et n’est pas une couche d’apprentissage

2. Contrairement aux convolutions que nous venons de voir dont le rôle est d’extraire de l’information, l’opération de pooling doit condenser l’information

3. Le pooling

MaxPooling2D
padding = valid
kernel size = (2, 2)
strides = kernel size



4. Architecture

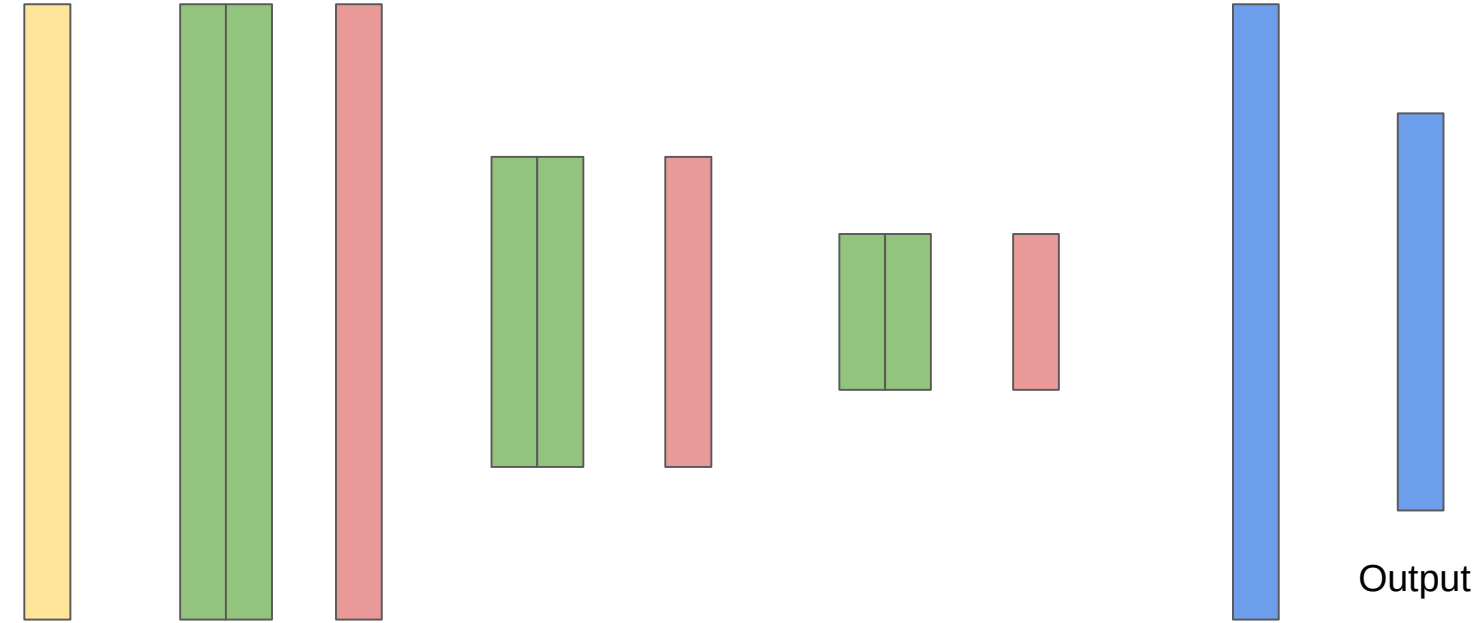
En pratique :

- On constitue plusieurs filtres
- On peut enchaîner les convolutions
- Le pooling permet de réduire la quantité de pixels
- On relie toutes nos convolutions à une simple couche dense

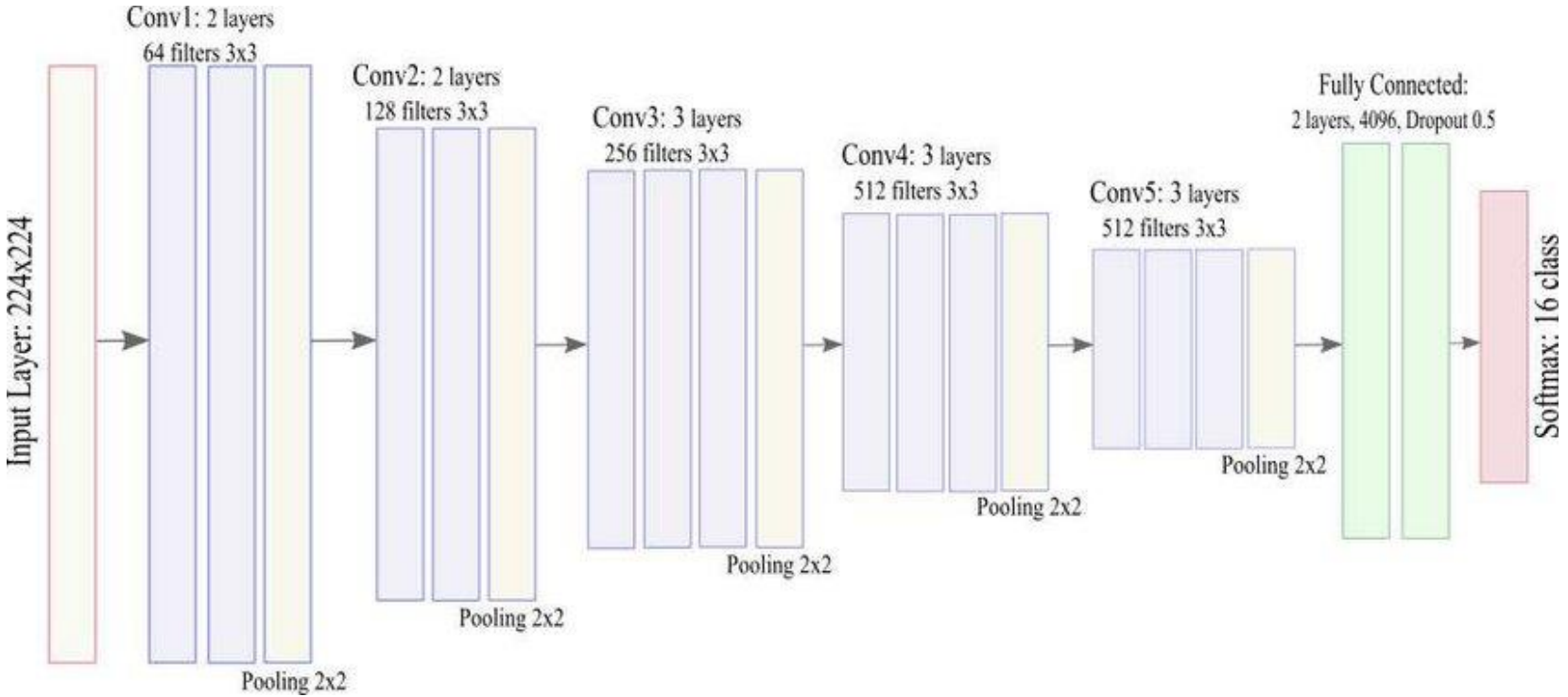
→ Permet d'extraire les données importantes des informations.

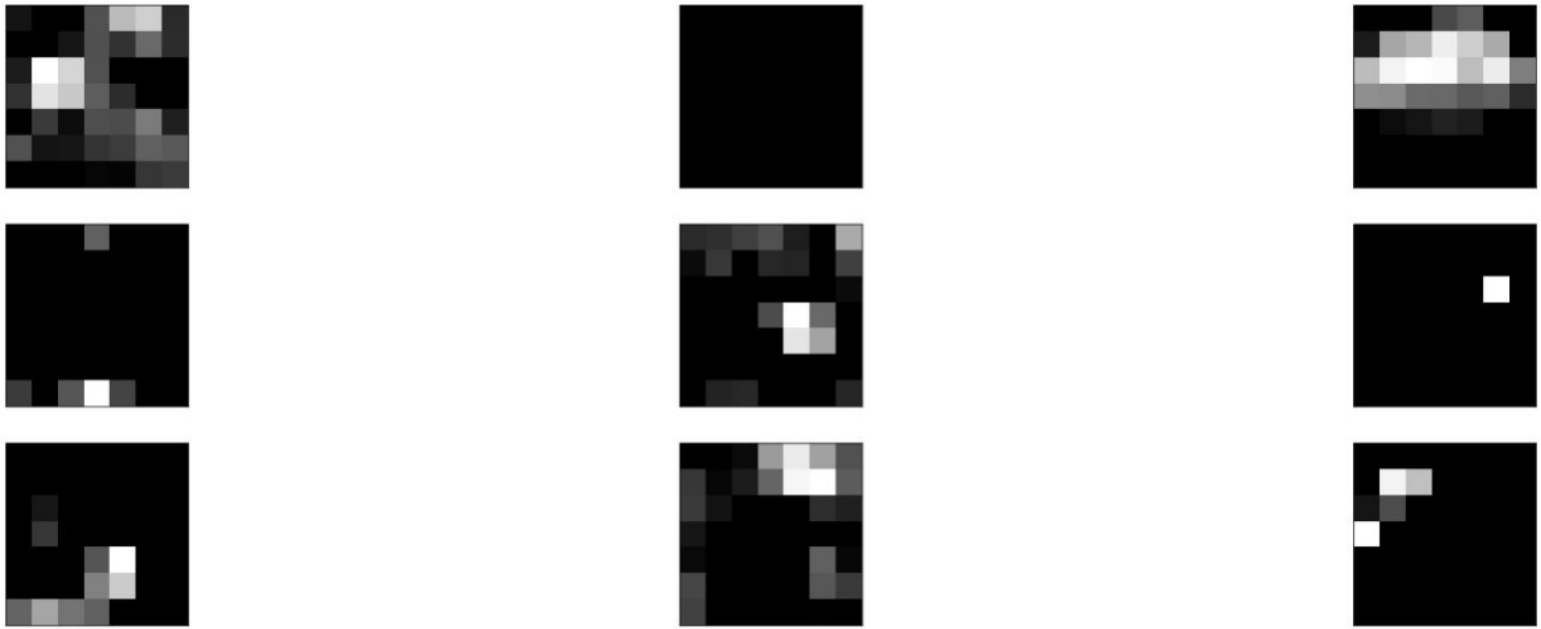
Ex : **VGG16**

4. Architecture



4. Architecture





5. Data Augmentation

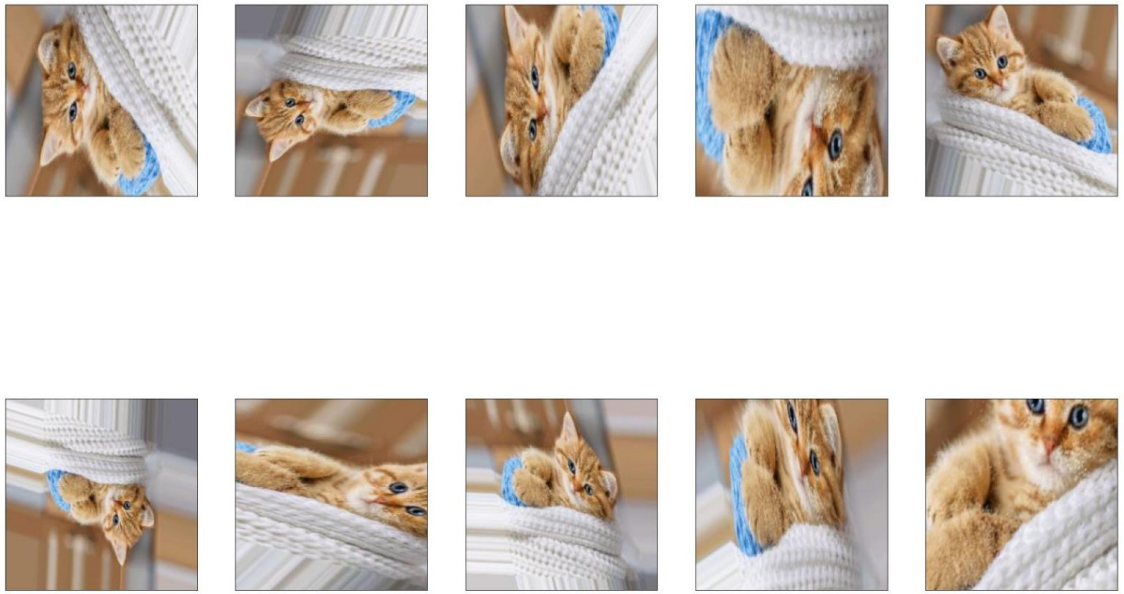
La Data Augmentation permet de créer des copies des images originales tout en les modifiant légèrement. Ces modifications permettent d'obtenir artificiellement plus de diversité dans le jeu de données ce qui, en théorie, doit permettre de contrer l'overfitting.

Comment modifier des images :

- Rotation
- Inverser haut/bas, droite/gauche
- Distorsion
- Inverser les canaux de couleurs

5. Data Augmentation

À chaque epoch, le modèle est entraîné avec des images qui diffèrent légèrement.



6. Autres architectures

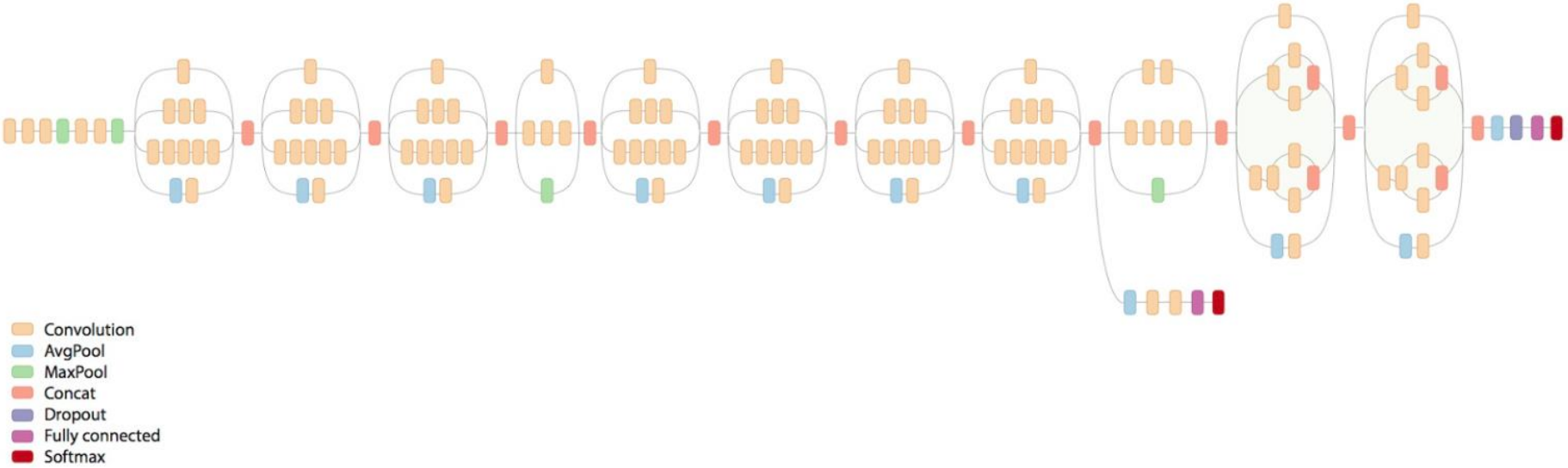
Jusqu'ici, nous avons utilisé des modèles séquentiels relativement simples mais il existe d'autres architectures plus complexes.

Ces architectures ne peuvent se construire qu'en passant par l'API fonctionnelle de Keras.

- Inception
- Resnet
- Mobilenet
- ConvNeXt

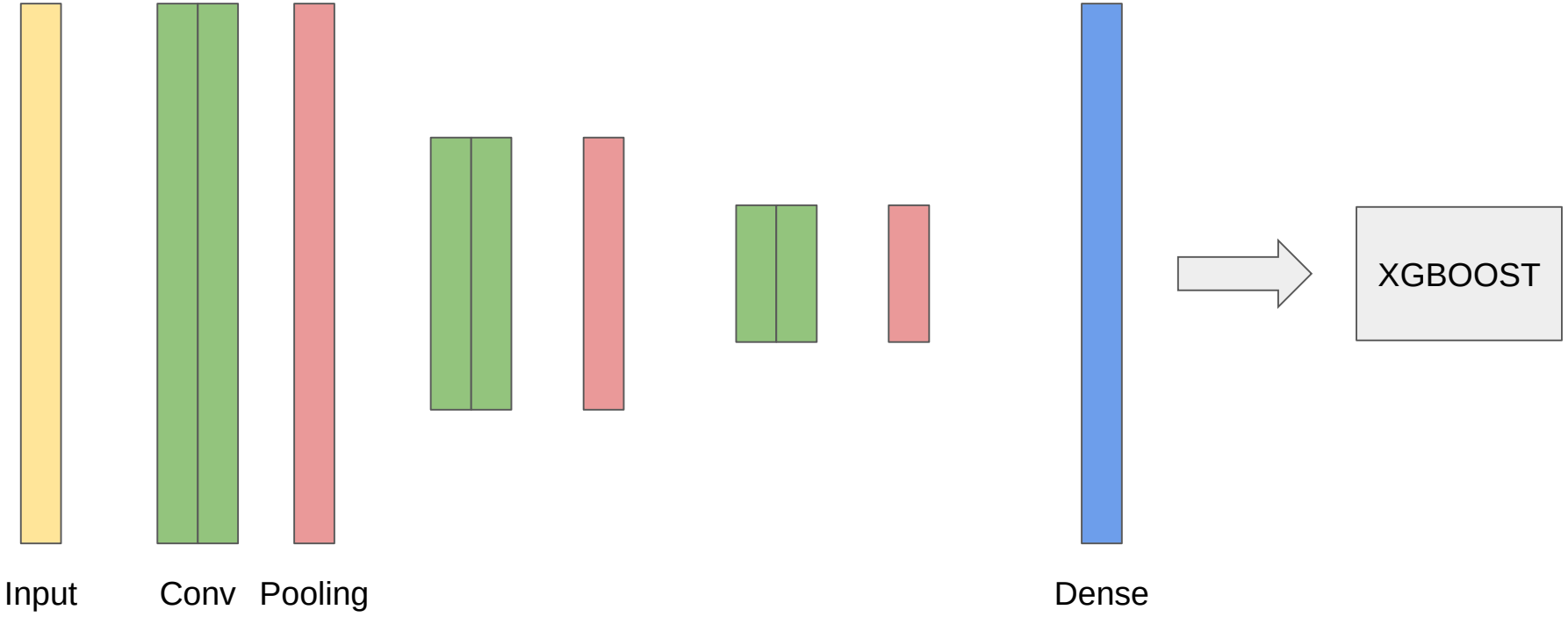
6. Autres architectures

InceptionV3



6. Autres architectures

CNN + XGBOOST



AE

Autoencoders

CONSULTING BSTORM

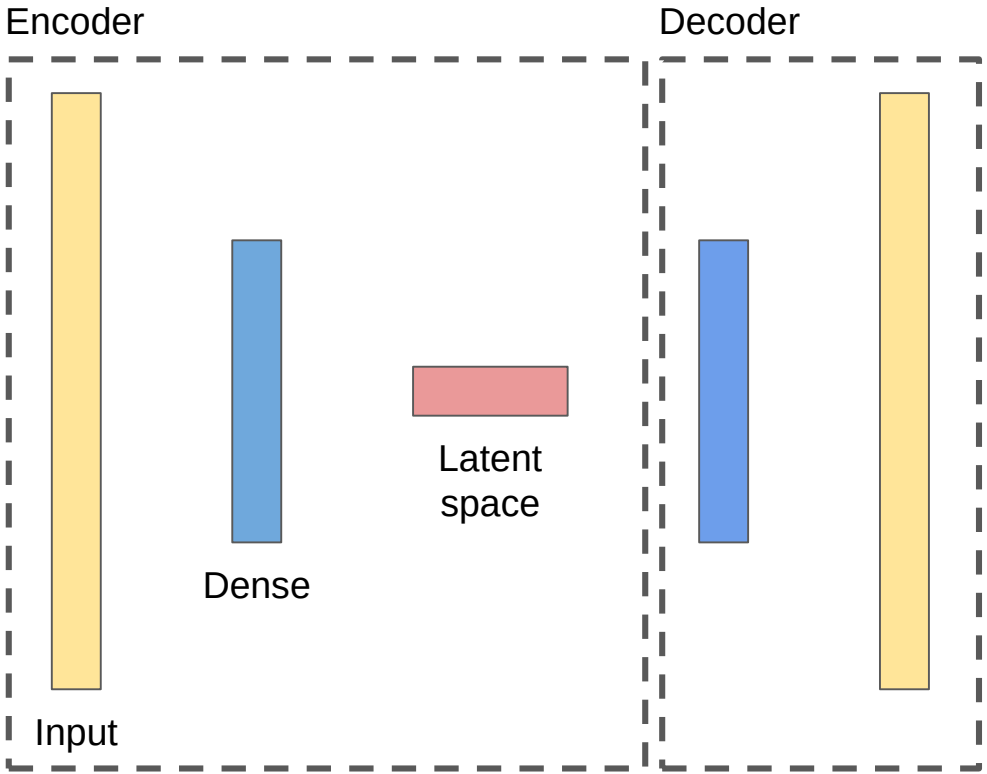
1. Introduction

Les autoencodeurs correspondent à une architecture particulière de réseau de neurones qui possède deux entités :

- L'encodeur qui a pour vocation de projeter l'information dans un nouvel espace réduit (espace latent). Il s'agit donc d'une réduction de la dimensionnalité
- Le décodeur reconstruit l'information sur base de l'espace latent

→ Il s'agit d'un apprentissage non supervisé

1. Introduction



Représentation classique d'un AE avec une couche d'entrée projetée dans un espace latent à l'aide d'une simple couche dense et ensuite reconstruite avec une autre couche dense.

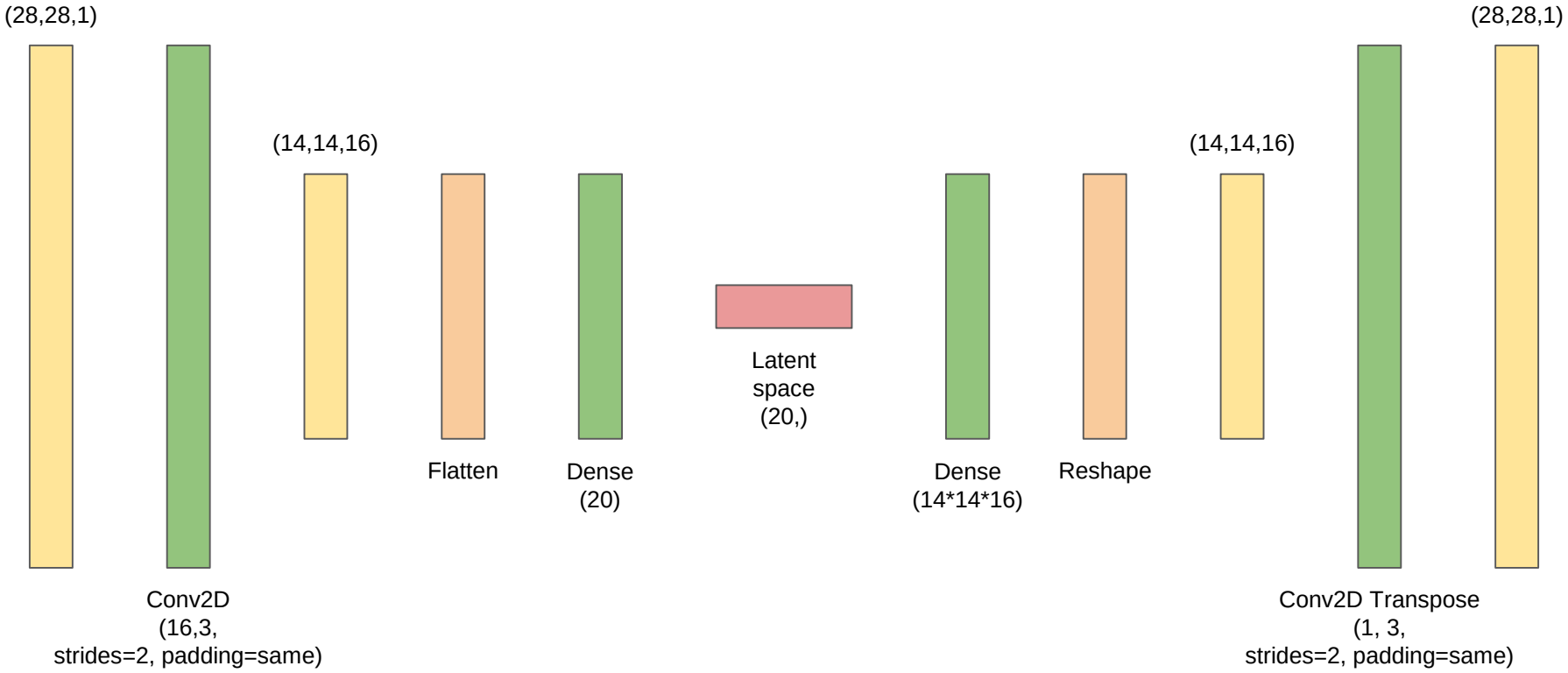
2. Application aux images

Dans le cadre des images, les autoencoders peuvent servir de débruiteur.

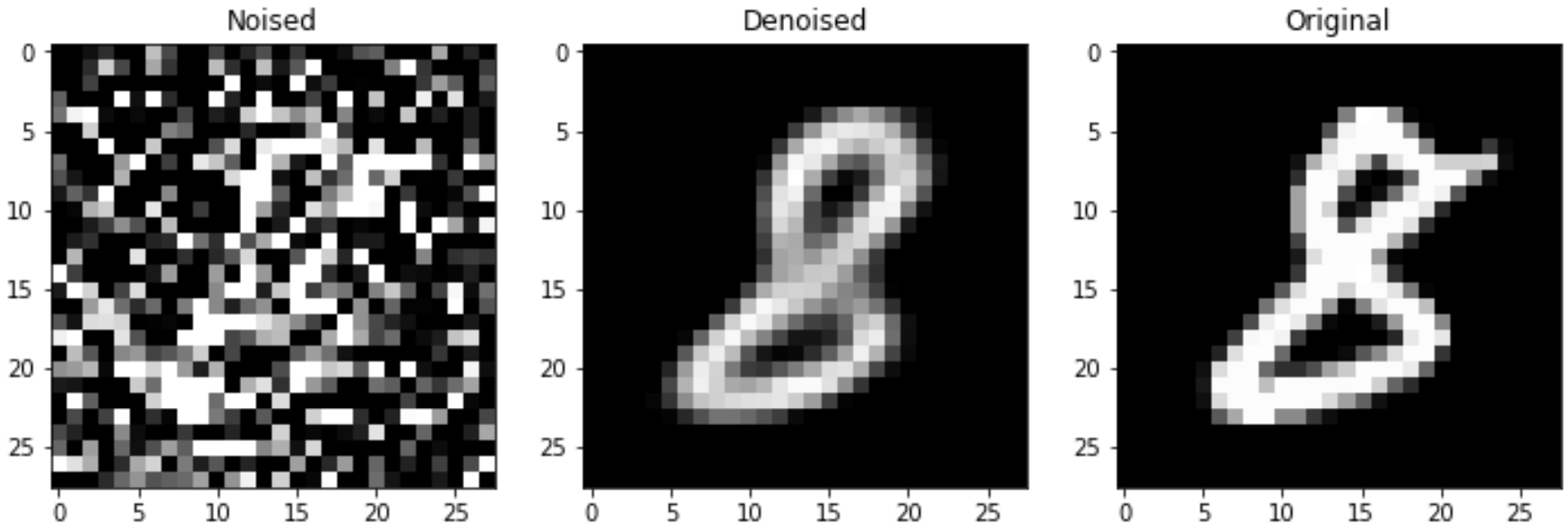
L’encoder sera constitué de couches de convolution avec comme objectif de projeter notre image dans un espace latent.

Le décodeur lui reconstruira une image débruitée.

2. Application aux images



2. Application aux images



Autres architectures

Intuition, exemples

CONSULTING BSTORM

1. RNN

Capacité à gérer des séquences :

- Des lettres, des mots
- Des images
- Du son, des paroles
- …

1. RNN

Capacité à gérer des séquences :

- Des lettres, des mots
- Des images
- Du son, des paroles
- …

1. RNN

Capacité à gérer des séquences :

- Des lettres, des mots
- Des images
- Du son, des paroles
- …

1. RNN

Capacité à gérer des séquences :

- Des lettres, des mots
- Des images
- Du son, des paroles
- …

1. RNN

→ Il faut avoir une mémoire pour prédire le déplacement d'un objet

Où les utilise-t-on ?

- Reconnaissance vocale
- Voiture autonome
- Génération de texte
- Génération de musique

LSTM, BI-LSTM sont des **RNN** améliorés

2. GAN

Architecture récente (2014)

L'idée est simple, on met deux réseaux en compétition :

- Un générateur
- Un discriminateur

L'objectif du générateur est de produire de l'information que le discriminateur ne pourra pas identifier comme fausse

2. GAN

Où les utilise-t-on ?

- Deepfake :
  - Vidéo 1
  - Vidéo 2
- Faux visages
- Génération de collections de mode
- Modélisation 3D (Architecture, chimie, pharmacie)
- ...

3. Reinforcement Learning

Cas particulier puisque le réseau de neurone doit se débrouiller :

- Sans données
- Sans règles

Les données sont inhérentes à l'environnement

On ne fournit qu'une seule chose : une **récompense**

3. Reinforcement Learning

Difficile à mettre en oeuvre dans la réalité :

- Simulation
- Réalité

Très souvent appliqué aux jeux :

e.g. société DeepMind :

- AlphaGo
- Starcraft 2