Deep Learning et application à l'anonymisation vidéo

EFREI 2025

Lusso Christelle - Renault Digital

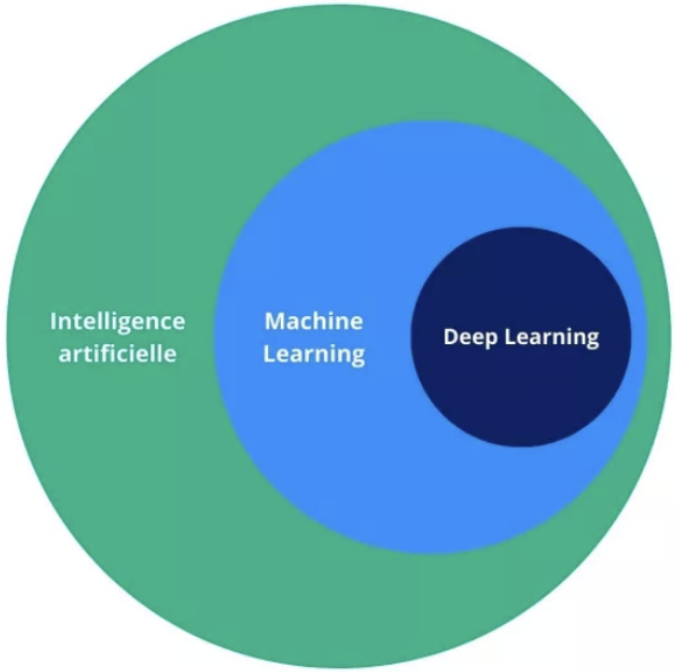
2025-04-18

Sommaire I

Introduction

Introduction

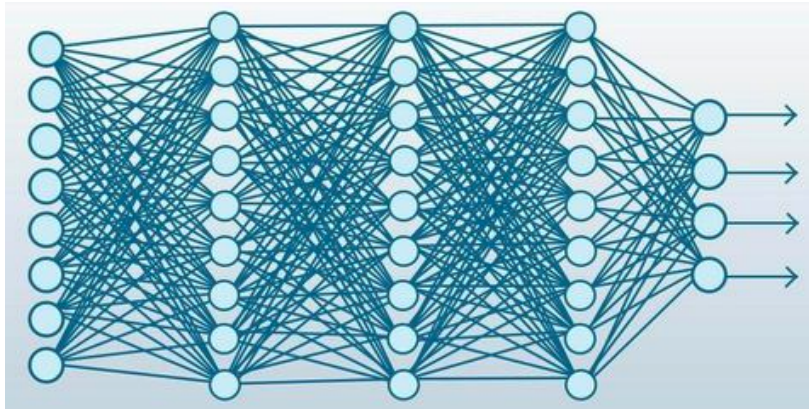
Le **Deep Learning** (apprentissage profond) est une branche du machine learning qui utilise des réseaux de neurones pour apprendre automatiquement des représentations complexes à partir de grandes quantités de données.



🤖 Cette technologie révolutionne des domaines comme la vision par ordinateur (Computer Vision) ou le traitement du langage naturel (NLP), en permettant aux machines d'extraire et d'interpréter des informations.

Réseaux de neurones

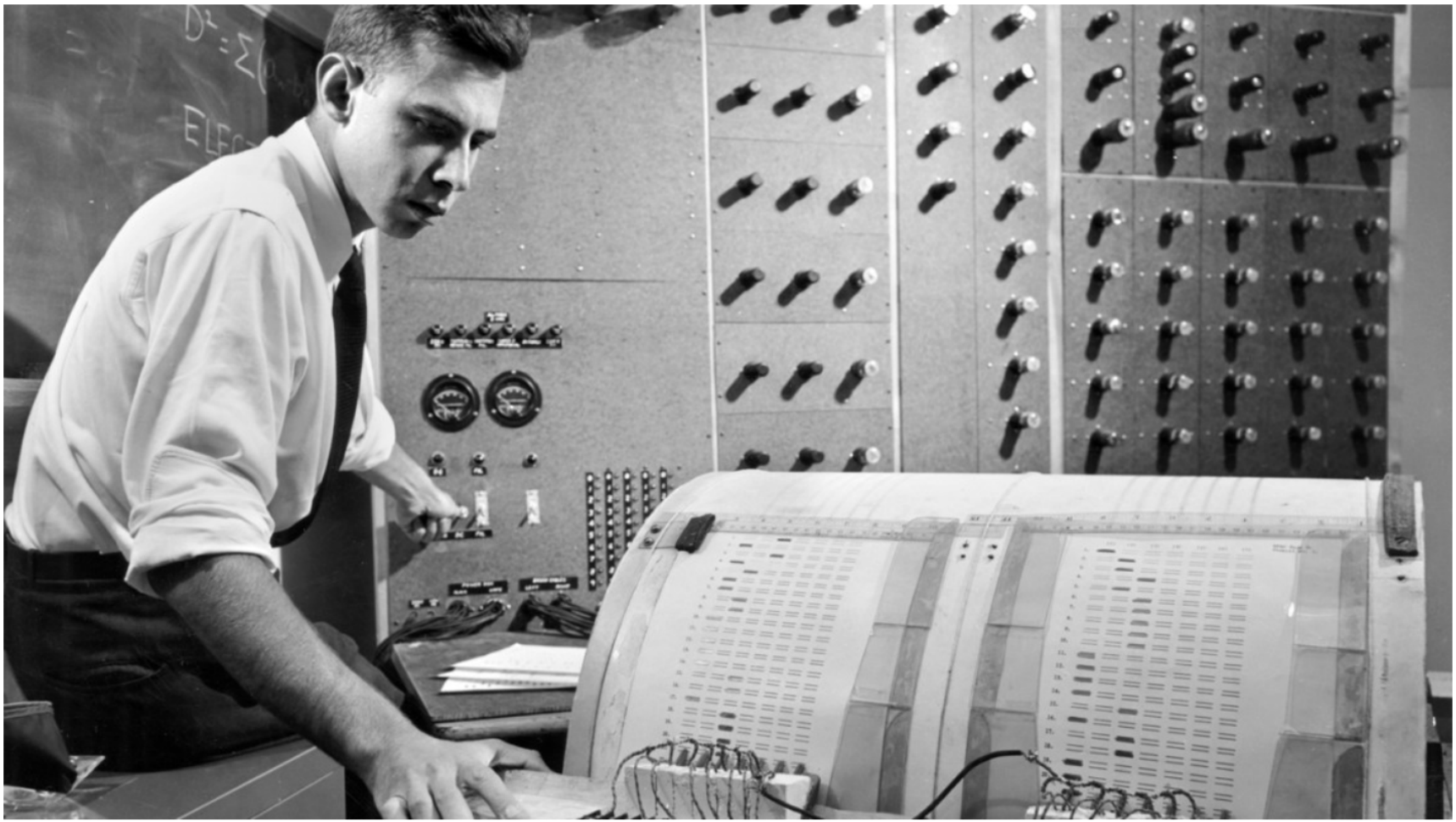
- Un **réseau de neurones** artificiel est un système d'apprentissage automatique imitant le cerveau humain et la manière dont les neurones biologiques s'envoient des signaux.
- Il se compose de neurones artificiels organisés en couches.



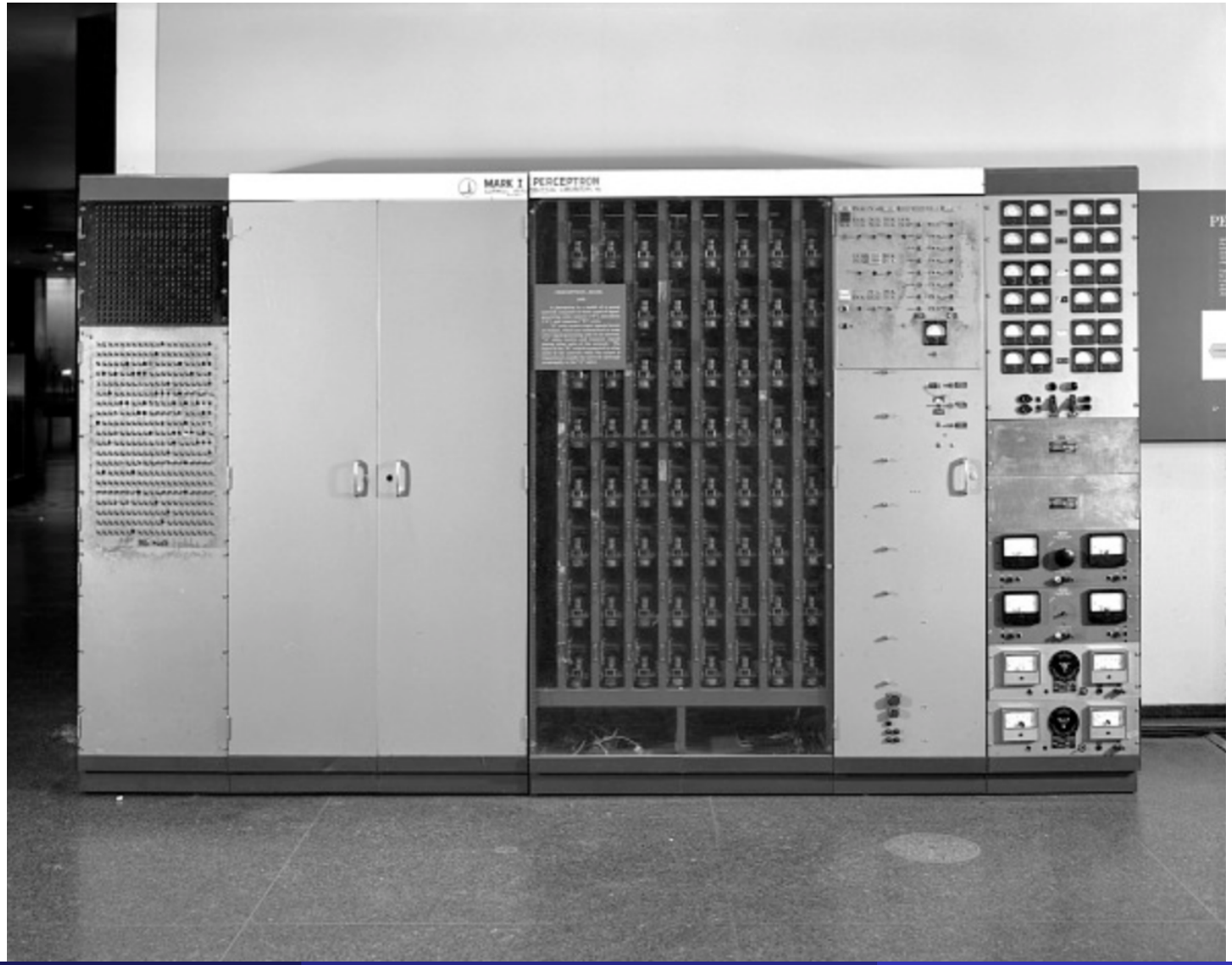
Perceptron

Introduction

- Le **Perceptron** a été crée en 1957 par le psychologue Frank Rosenblatt.
- **But:** modéliser les mécanismes d'apprentissage du cerveau humain.

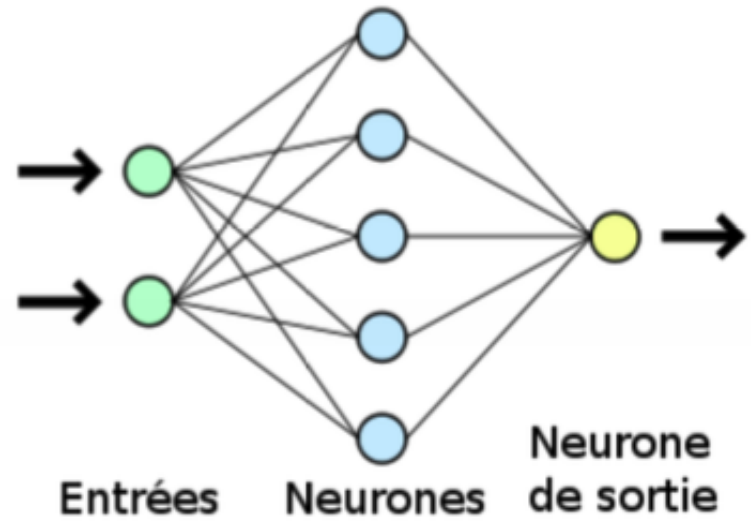


- **Mark 1 Perceptron** a été crée au Cornell Aeronautical Laboratory à Buffalo, New York 🔥
- Caméra, circuits analogiques, capteurs 👉 reconnaitre des formes élémentaires.



Définition

- Réseau de neurones le plus simple !
- Algorithme de classification binaire:
  - Oui ou Non.
  - 0 ou 1.
- Constitué d'**une seule couche** de neurones.



Structure du Perceptron

- Entrées $(x_1, ..., x_n)$ qui sont les variables du problème.
- Poids $(w_1, ..., w_n)$ associés aux entrées.
- Une couche de neurones, avec pour chaque neurone:
  1) Fonction de somme $z = \sum_{i=1}^{n} w_i x_i + b$ où b est un biais.
  2) **Fonction d'activation**:
     - Applique une règle pour produire la sortie (activer un neurone ou non).
     - Exemple: fonction seuil

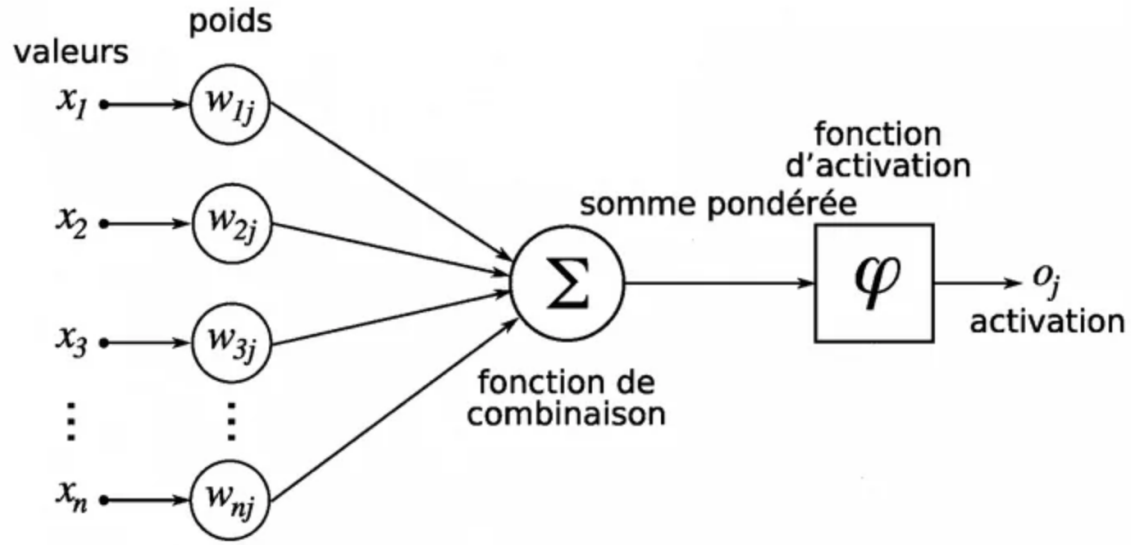
$$sortie = \begin{cases} 1 & \text{si } z > 0 \\ 0 & \text{sinon} \end{cases}$$

Structure du Perceptron

- Entrées $(x_1, ..., x_n)$ qui sont les variables du problème.
- Poids $(w_1, ..., w_n)$ associés aux entrées.
- Une couche de neurones, avec pour chaque neurone:
  1) Fonction de somme $z = \sum_{i=1}^{n} w_i x_i + b$ où b est un biais.
  2) **Fonction d'activation**:
     - Applique une règle pour produire la sortie (activer un neurone ou non).
     - Exemple: fonction seuil

$$sortie = \begin{cases} 1 & \text{si } z > 0 \\ 0 & \text{sinon} \end{cases}$$

- Représentation avec un seul neurone:



Apprentissage

1) Initialisation des poids et biais (aléatoirement).
2) **Propagation avant**:
   - Somme pondérée poids/entrées + biais: $z = \sum_{i=1}^{n} w_i x_i + b$
   - Fonction d'activation: $\phi(z) = \phi(\sum_{i=1}^{n} w_i x_i + b) \rightarrow$ sortie.

Apprentissage

1) Initialisation des poids et biais (aléatoirement).
2) **Propagation avant**:
   - Somme pondérée poids/entrées + biais: $z = \sum_{i=1}^{n} w_i x_i + b$
   - Fonction d'activation: $\phi(z) = \phi(\sum_{i=1}^{n} w_i x_i + b) \rightarrow$ sortie.
2) Calcul de l'erreur $y - \hat{y}$, où
   - y est la valeur réelle,
   - $\hat{y}$ est la sortie prédite.

Apprentissage

1) Initialisation des poids et biais (aléatoirement).
2) **Propagation avant**:
   - Somme pondérée poids/entrées + biais: $z = \sum_{i=1}^{n} w_i x_i + b$
   - Fonction d'activation: $\phi(z) = \phi(\sum_{i=1}^{n} w_i x_i + b) \rightarrow$ sortie.
2) Calcul de l'erreur $y - \hat{y}$, où
   - y est la valeur réelle,
   - $\hat{y}$ est la sortie prédite.
2) **Mise à jour** des poids/biais $\rightarrow$ ajustés selon la **règle d'apprentissage**:

$$w_i = w_i + \eta(y - \hat{y})x_i$$
$$b = b + \eta(y - \hat{y})$$

   - η est le taux d'apprentissage (learning rate).

Apprentissage

1) Initialisation des poids et biais (aléatoirement).
2) **Propagation avant**:
   - Somme pondérée poids/entrées + biais: $z = \sum_{i=1}^{n} w_i x_i + b$
   - Fonction d'activation: $\phi(z) = \phi(\sum_{i=1}^{n} w_i x_i + b) \rightarrow$ sortie.
2) Calcul de l'erreur $y - \hat{y}$, où
   - y est la valeur réelle,
   - $\hat{y}$ est la sortie prédite.
2) **Mise à jour** des poids/biais $\rightarrow$ ajustés selon la **règle d'apprentissage**:

$$w_i = w_i + \eta(y - \hat{y})x_i$$
$$b = b + \eta(y - \hat{y})$$

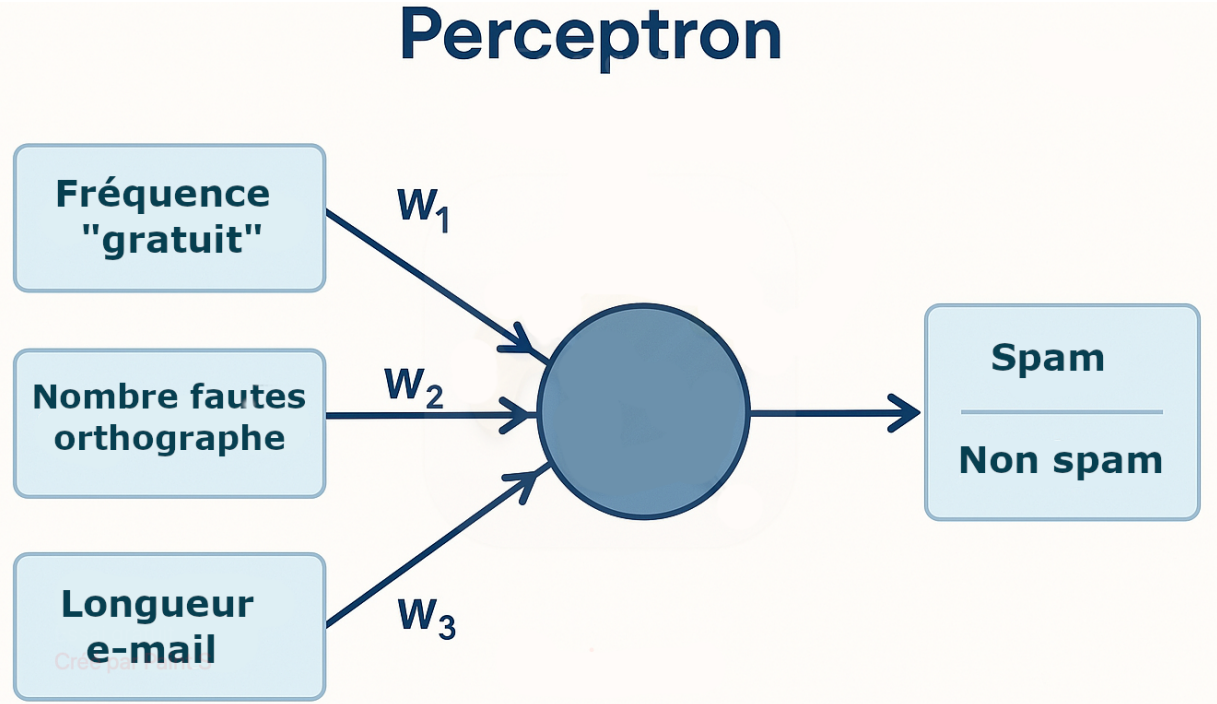
   - η est le taux d'apprentissage (learning rate).

Limitation: le perceptron ne peut résoudre que des problèmes où les classes sont linéairement séparables. Pour résoudre des problèmes plus complexes, il faut un perceptron multicouche.

Application - Classification binaire linéaire

Détection de spam dans les e-mails:

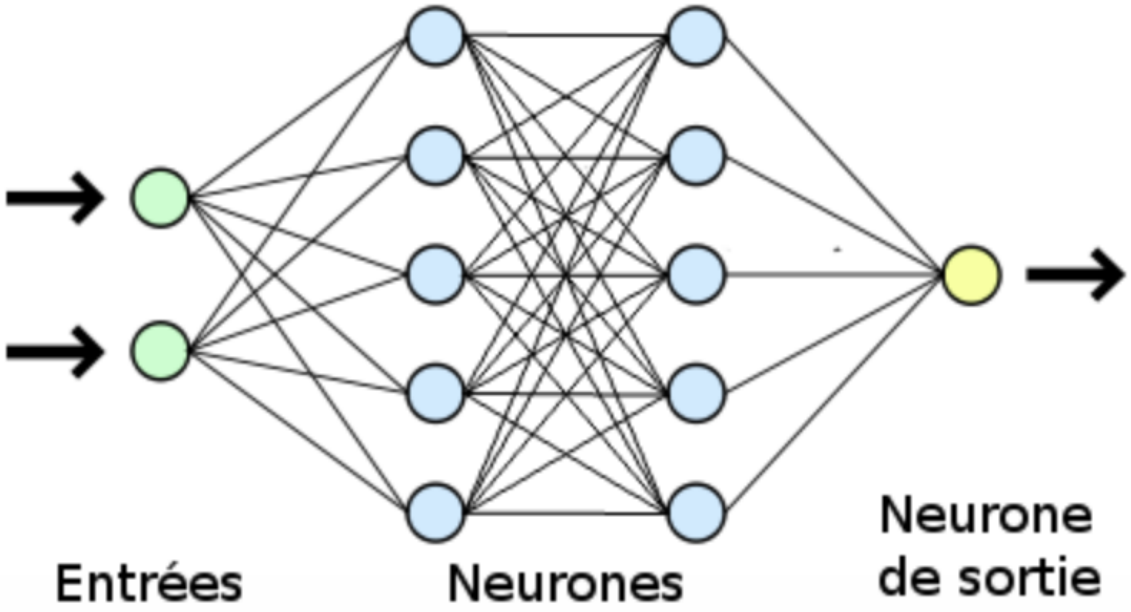
- **Entrée:** e-mail (représenté par des caractéristiques numériques).
- **Caractéristiques:**
  - x_1: fréquence du mot “gratuit”.
  - x_2: nombre de fautes d’orthographe.
  - x_3: longueur du message.
- **Sortie:** 0 ou 1 (spam ou non-spam)



Perceptron Multicouche (MLP)

Structure du multicouche

- Le **Perceptron Multicouche** (MLP) est un réseau de neurones avec une succession de couches:
  - une couche d'entrée $(x_1, ..., x_n)$,
  - plusieurs couches, dites cachées,
  - une couche de sortie (résultat final).
- Tous les neurones d'une couche sont connectés aux neurones des couches adjacentes (**fully-connected**).



Fonctions d'activation

👉 La fonction d'activation dépend du problème considéré.

Perceptron:

- Fonction seuil:
$$f(z) = \begin{cases} 1 & \text{si } z > 0 \\ 0 & \text{sinon} \end{cases}$$
- Fonction sigmoid: $f(z) = \dfrac{1}{1 + exp(-z)}$.

Perceptron Multi-couches:

- Fonction ReLU (Rectified Linear Unit): $f(z) = \max(0, z)$
- Fonction Softmax: $f(z_i) = \dfrac{e^{z_i}}{\sum_j e^{z_j}}$
- Fonction tanh (Tangente hyperbolique): $f(z) = \dfrac{2}{1 + exp(-2z)} - 1$

Entrainement

- L'apprentissage se fait à partir d'un ensemble de données d'entrainement $(\vec{x}, y) \in \mathbb{R}^n \times \mathbb{R}$.
  - $\vec{x}$ sont les données du problème.
  - y est la valeur associée.

🎯 **Pour chaque entrée $\vec{x}$:**

- **Propagation avant** de tous les neurones du réseau.
- $\Rightarrow$ prédiction finale en sortie $\hat{y}$.
- Calcul de l'erreur entre la valeur réelle et la valeur prédite par le réseau.
  - Fonction de perte (Loss function): $L(y) = (y - \hat{y})^2$.
- Pour entrainer le modèle, on remonte l'erreur vers l'entrée: c'est la **rétropropagation**.
- On ajuste les poids de facon à minimiser l'erreur: c'est la **mise à jour**.

Rétropropagation ~1980

Intuition:

- Chaque neurone du réseau est responsable d'une partie de l'erreur.
- Les neurones proches de la sortie influencent directement l'erreur, tandis que ceux des couches précédentes contribuent indirectement.

💡 **Idée:**

- Répartir l'erreur du neurone de sortie en remontant sur tous les neurones connectés.

Rétropropagation ~1980

Intuition:

- Chaque neurone du réseau est responsable d'une partie de l'erreur.
- Les neurones proches de la sortie influencent directement l'erreur, tandis que ceux des couches précédentes contribuent indirectement.

💡 **Idée:**

- Répartir l'erreur du neurone de sortie en remontant sur tous les neurones connectés.

Comment?

- **Objectif:** minimisation de la fonction de perte L (ou fonction coût C) par rapport aux poids ω_{ij}.
- **Solution:** optimisation par descente de gradient.

Rétropropagation et mise à jour

Rétropropagation:

- Calcul de toutes les dérivées partielles $\frac{\partial L}{\partial \omega_{ij}}$.
- Règle de la chaine pour introduire les fonctions d'activation σ_i des couches précédentes: $\frac{\partial L}{\partial \omega_{ij}} = \frac{\partial L}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial z_j} \frac{\partial z_j}{\partial \omega_{ij}}$
- Se faisant, on remonte la contribution de chaque neurone sur l'erreur.

Rétropropagation et mise à jour

Rétropropagation:

- Calcul de toutes les dérivées partielles $\frac{\partial L}{\partial \omega_{ij}}$.
- Règle de la chaine pour introduire les fonctions d'activation σ_i des couches précédentes: $\frac{\partial L}{\partial \omega_{ij}} = \frac{\partial L}{\partial \sigma_i} \frac{\partial \sigma_i}{\partial z_j} \frac{\partial z_j}{\partial \omega_{ij}}$
- Se faisant, on remonte la contribution de chaque neurone sur l'erreur.

Mise à jour des poids:

- Dérivées partielles de L par rapport aux poids $\rightarrow$ direction dans laquelle ajuster les poids.
- A chaque itération i de l'algorithme:

$$\omega_{ij} \leftarrow \omega_{ij} - \eta \frac{\partial L}{\partial \omega_{ij}}$$

où η est le taux d'apprentissage.

Fonction de coût

Perceptron - classification binaire:
Hinge Loss: $L = \max(0, 1 - y \cdot \hat{y})$.

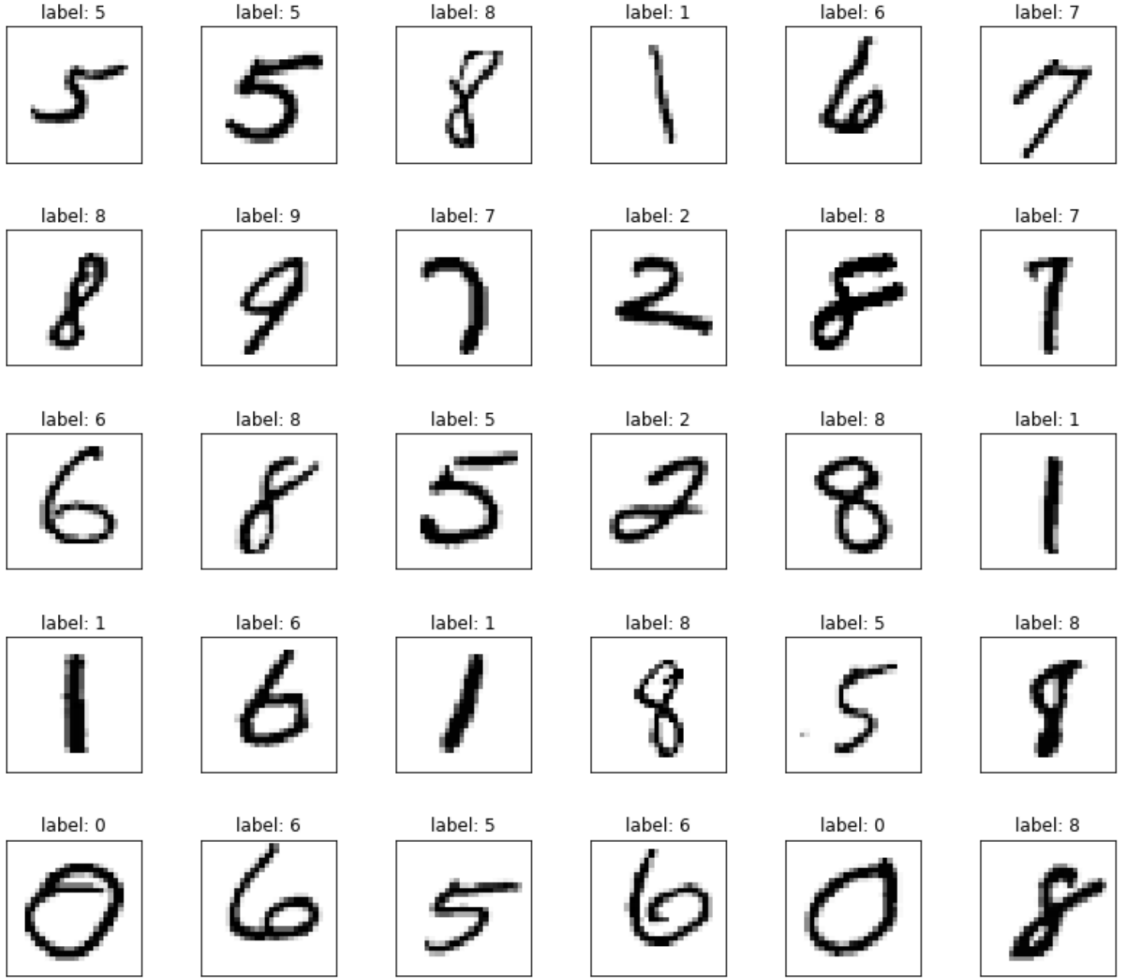
MLP - classification binaire:
Entropie croisée (Binary Cross-Entropy):
$L = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$ pénalise fortement les prédictions incorrectes.

MLP - classification multiclasse:
Entropie croisée catégorique (Categorical Cross-Entropy):
$L = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$ pénalise fortement si la probabilité de la vraie classe est faible.

MLP - régression:
Erreur quadratique moyenne (Mean Squared Error- MSE):
$L = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$ où y_i est la valeur pour la i-em observation, mesure l'écart entre les valeurs prédites et les valeurs réelles.

Application - Classification MNIST

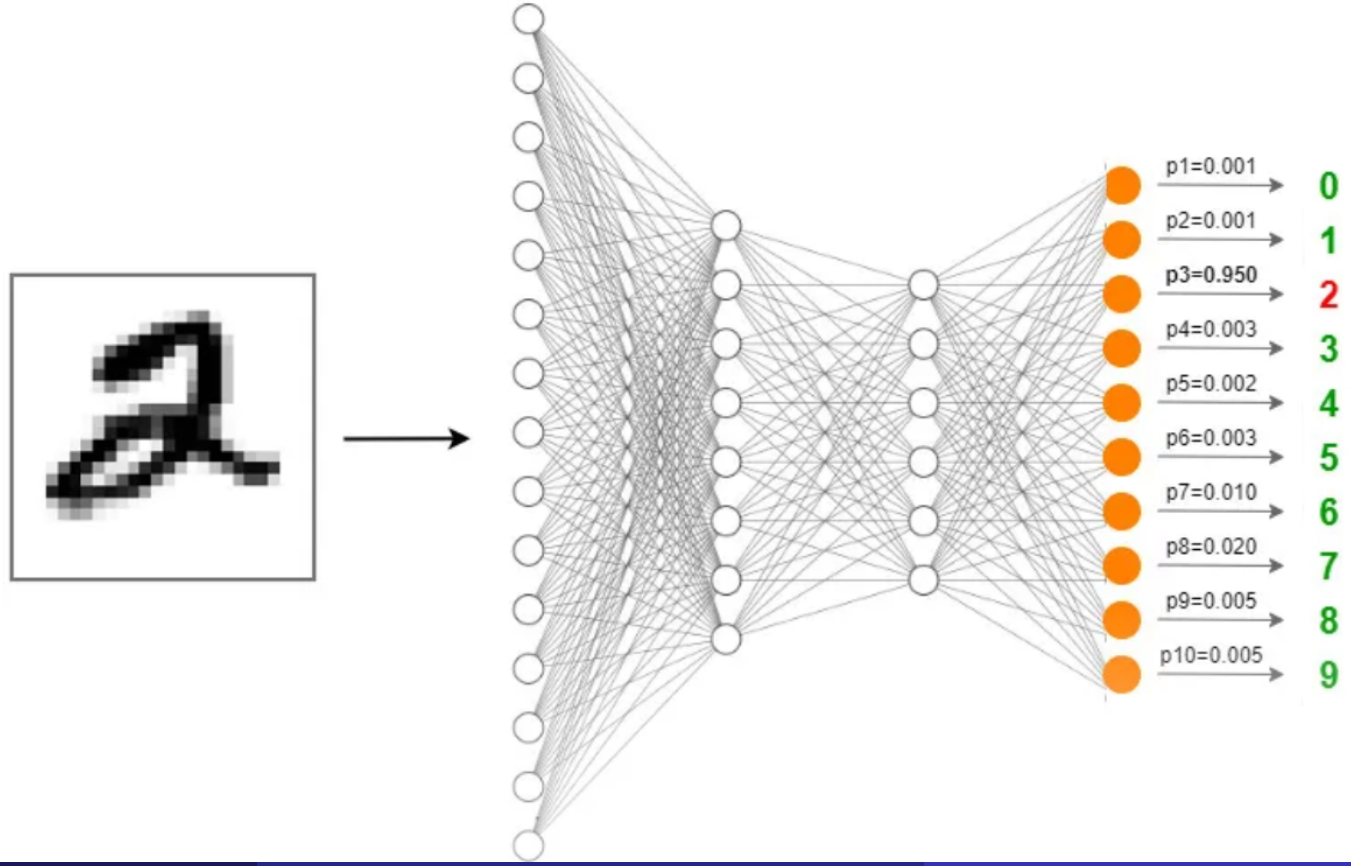
Jeu de données MNIST - chiffres manuscrits:



Application - Classification MNIST

Classification des chiffres manuscrits:

- **Entrée:** petite image 28x28 pixels.
- **Caractéristiques:** 784 valeurs (pixels).
- **Nombre de classes:** 10 (1 classe par chiffre).
- **Sorties:** 10 probabilités d'appartenance à une classe.



Réseaux de neurones convolutifs (CNN)

Introduction

- 🚀 Les CNN datent des années 1980 !
- Yann LeCun a commencé à développer les CNN dès 1989.
  - Reconnaissance de chiffres manuscrits.
  - 1998: réseau LeNet-5 🔥

🥶 1990-2000: l'hiver de l'IA

- Manque de puissance de calcul.
- Manque de données.

🔥 2012: retour en force

- Réseau AlexNet - créé par Alex Krizhevsky.
- Entrainé par GPU.
- 🏆 Coucours Computer Vision - ImageNet.

Introduction

Réseaux de neurones convolutifs:

Les **réseaux de neurones convolutifs** (CNN) ont une architecture spécialement conçue pour traiter les données ayant une structure spatiale. Ces réseaux ont révolutionné des domaines comme la vision par ordinateur (Computer Vision) et le traitement d'image en permettant de classer une image (chat, chien, voiture...).

- 🎯 Années 1980-1990 - Yann LeCun.

Introduction

Réseaux de neurones convolutifs:

Les **réseaux de neurones convolutifs** (CNN) ont une architecture spécialement conçue pour traiter les données ayant une structure spatiale. Ces réseaux ont révolutionné des domaines comme la vision par ordinateur (Computer Vision) et le traitement d'image en permettant de classer une image (chat, chien, voiture...).

- 🎯 Années 1980-1990 - Yann LeCun.

Pourquoi ?

- Les images sont de très grande taille ➔ millions de pixels.
- Avec un réseau fully-connected (MLP):
  - Chaque pixel est connecté à tous les neurones.
  - ✖ milliards de paramètres !

Introduction

Comment ? Inspirés par la structure du cortex visuel 🧠

- Le cortex visuel est constitué de plusieurs couches.
- Chaque couche correspond à une représentation de plus en plus abstraite de la vision.

Introduction

Comment ? Inspirés par la structure du cortex visuel 🧠

- Le cortex visuel est constitué de plusieurs couches.
- Chaque couche correspond à une représentation de plus en plus abstraite de la vision.

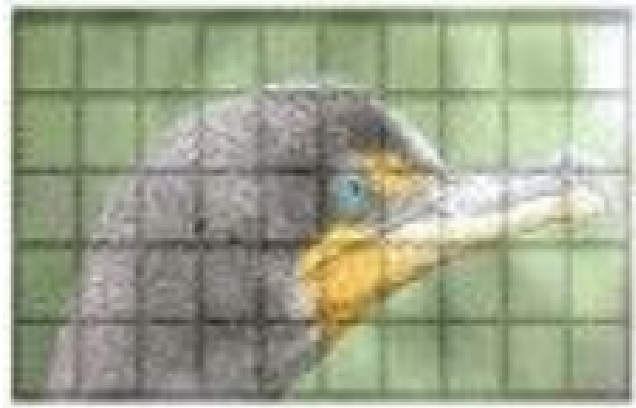
Un réseau **CNN** utilise des couches de convolution pour extraire des caractéristiques: contours, textures ou motifs locaux appelés **features**.

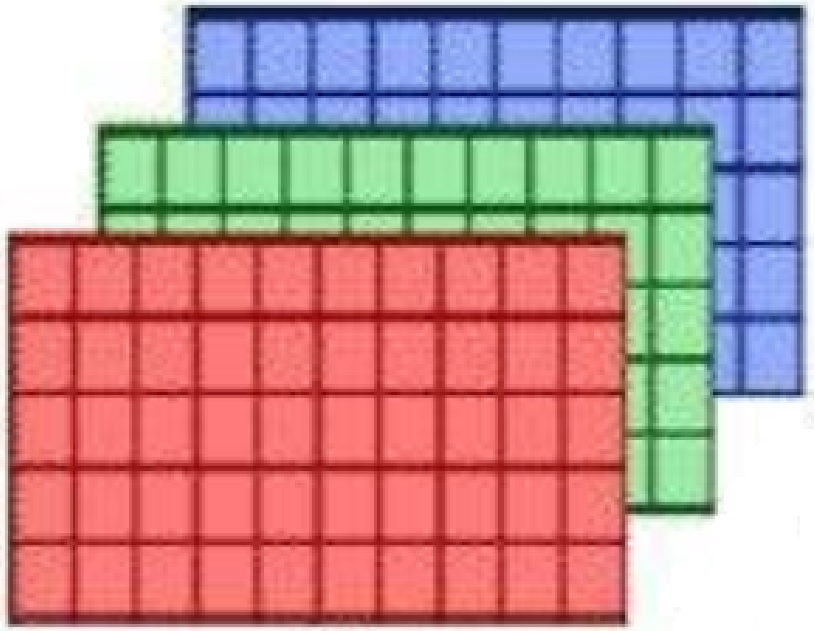
- 1ère couche: mesure la luminosité et la couleur des pixels.
- 2em couche: étudie les corrélations entre les pixels voisins.
- 3em couche: identifie des lignes directrices.
- ...
- Identification de composants, comme les yeux et les oreilles.
- Combinaison des yeux et des oreilles pour détecter la présence d'un visage.

Traitement d'images

- Une image est constituée de pixels.
- Noir et blanc: chaque pixel a un niveau de gris (entre 1 et 256).
- Couleurs: chaque pixel a 3 niveaux de couleur (Rouge, Vert et Bleu).

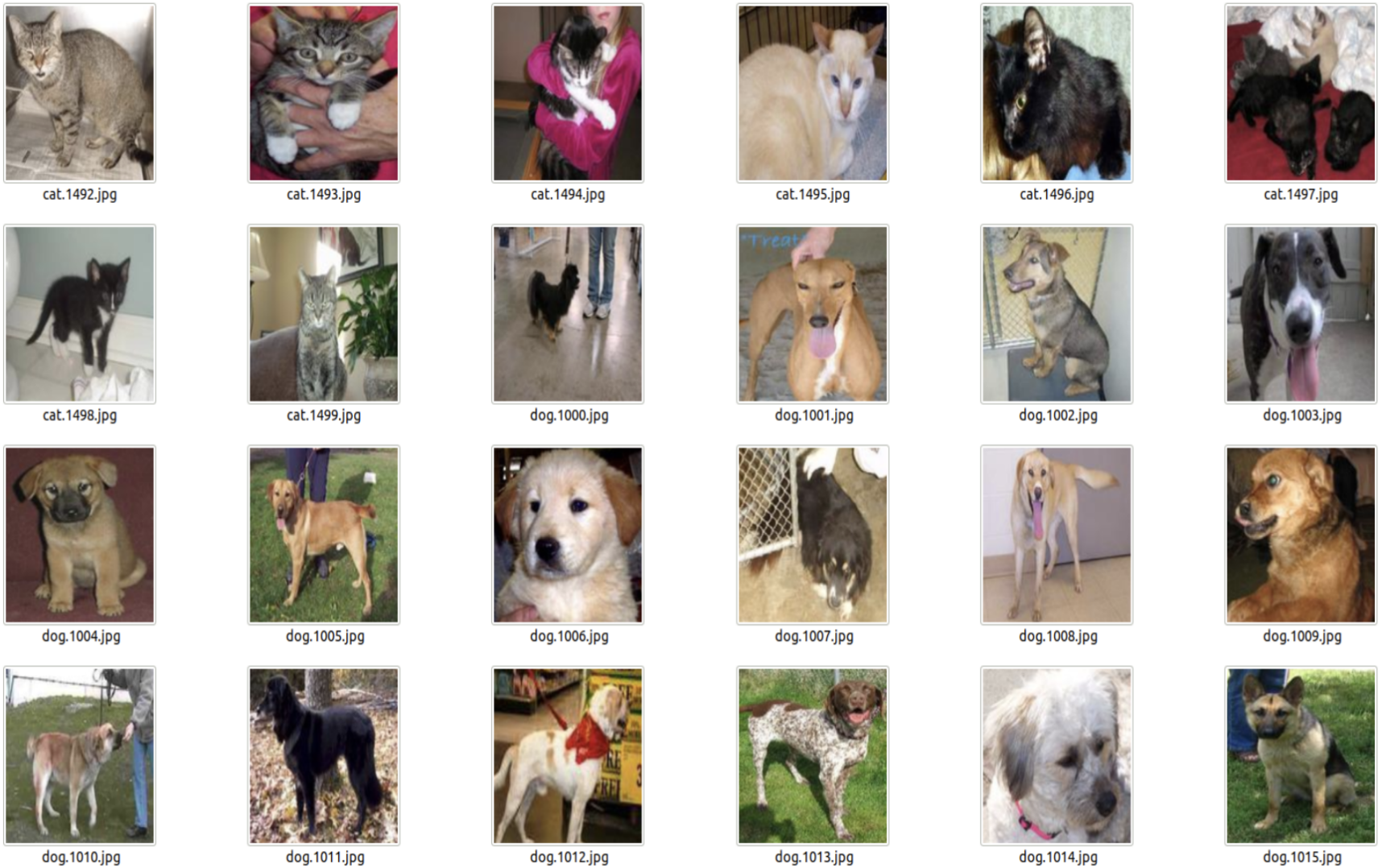
- Image de $N \times N$ pixels $=$ matrice de taille $N \times N$.
- Image couleur de $N \times N$ pixels $=$ 3 matrices de taille $N \times N$.





Données d'entrainement

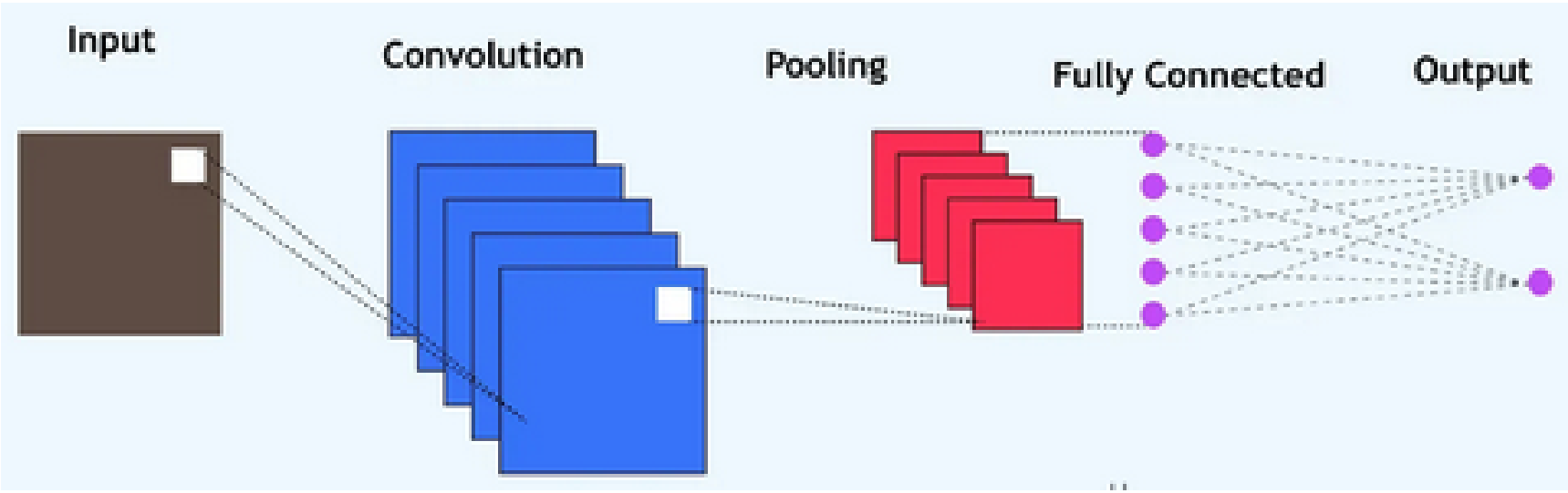
Exemple jeu de données (reconnaissance d'images):



Architecture d'un CNN

Un CNN est constitué de plusieurs partie:

- **Partie convolutive:**
  - couches convolutives → extraction des caractéristiques locales de l'image, comme les contours ou motifs ⇒ **feature maps**.
- **Partie de Pooling:**
  - couches de pooling → réduction de la dimension des **features maps**, tout en préservant les caractéristiques essentielles de l'image.
- **Partie de classification:**
  - couches fully-connected → classification.



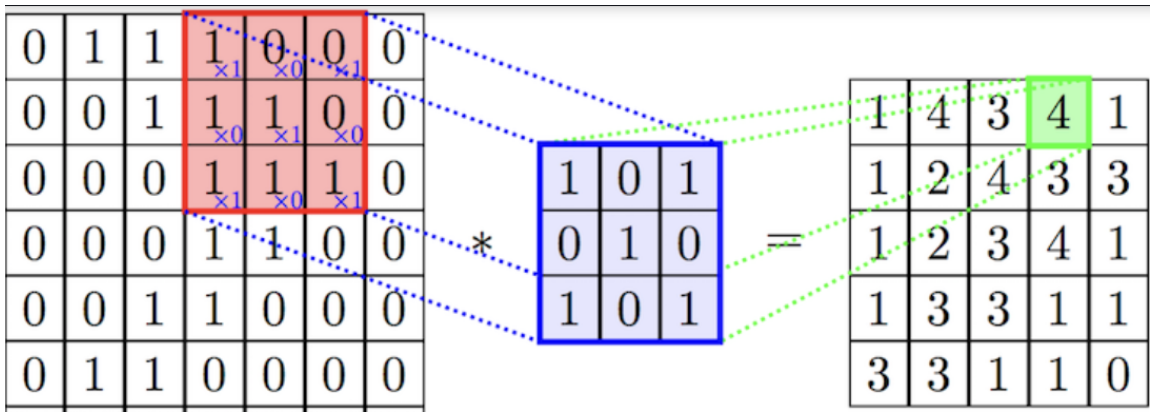
Partie convolutive

- Application d'un **filtre** sur l'image pour détecter des motifs locaux.
- 📌 Un **filtre** est une matrice de taille 3×3.
- Il existe différents types de filtres:
  - Filtre de détection de contours.
  - Filtre Gaussien: appliquer un flou.

Partie convolutive

- Application d'un **filtre** sur l'image pour détecter des motifs locaux.
- 📌 Un **filtre** est une matrice de taille 3×3.
- Il existe différents types de filtres:
  - Filtre de détection de contours.
  - Filtre Gaussien: appliquer un flou.

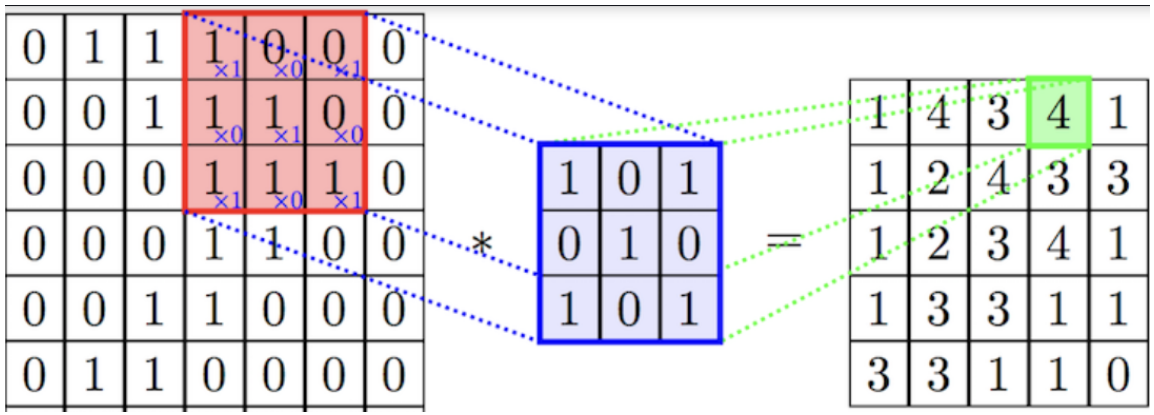
La convolution d'une image s'effectue en déplacant un **filtre** le long de l'image, et en faisant les produits de convolution.



Partie convolutive

- Application d'un **filtre** sur l'image pour détecter des motifs locaux.
- 📌 Un **filtre** est une matrice de taille 3×3.
- Il existe différents types de filtres:
  - Filtre de détection de contours.
  - Filtre Gaussien: appliquer un flou.

La convolution d'une image s'effectue en déplacant un **filtre** le long de l'image, et en faisant les produits de convolution.



La **convolution** peut etre apprise par une couche de neurones de convolution.

Apprentissage d'un filtre

Apprentissage d'un filtre:

- 🧠 Les **coefficients** d'un filtre convolutif sont **appris** par un réseau de neurones:
  - Les poids des neurones correspondent aux coefficients du filtre.
  - Au début ils sont aléatoires 👀
  - Ils se précisent par rétropropagation.
  - Certains deviennent sensibles aux diagonales, d'autres aux textures...

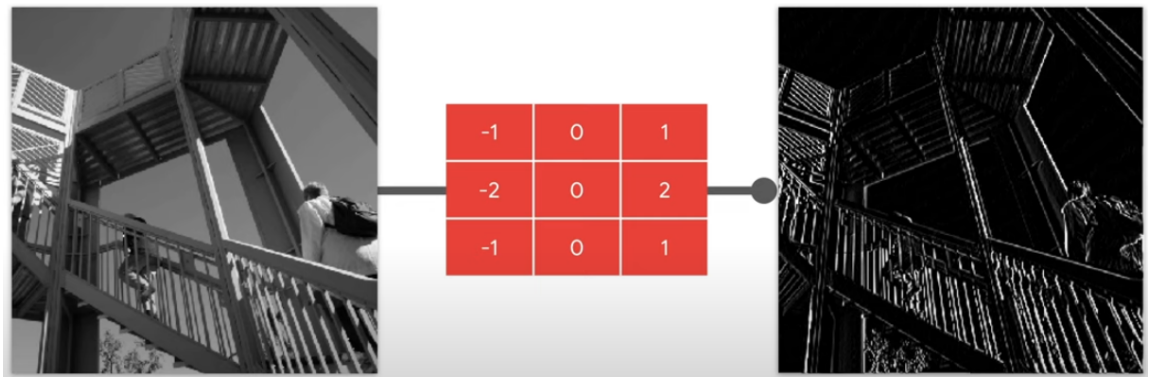
Apprentissage d'un filtre

Apprentissage d'un filtre:

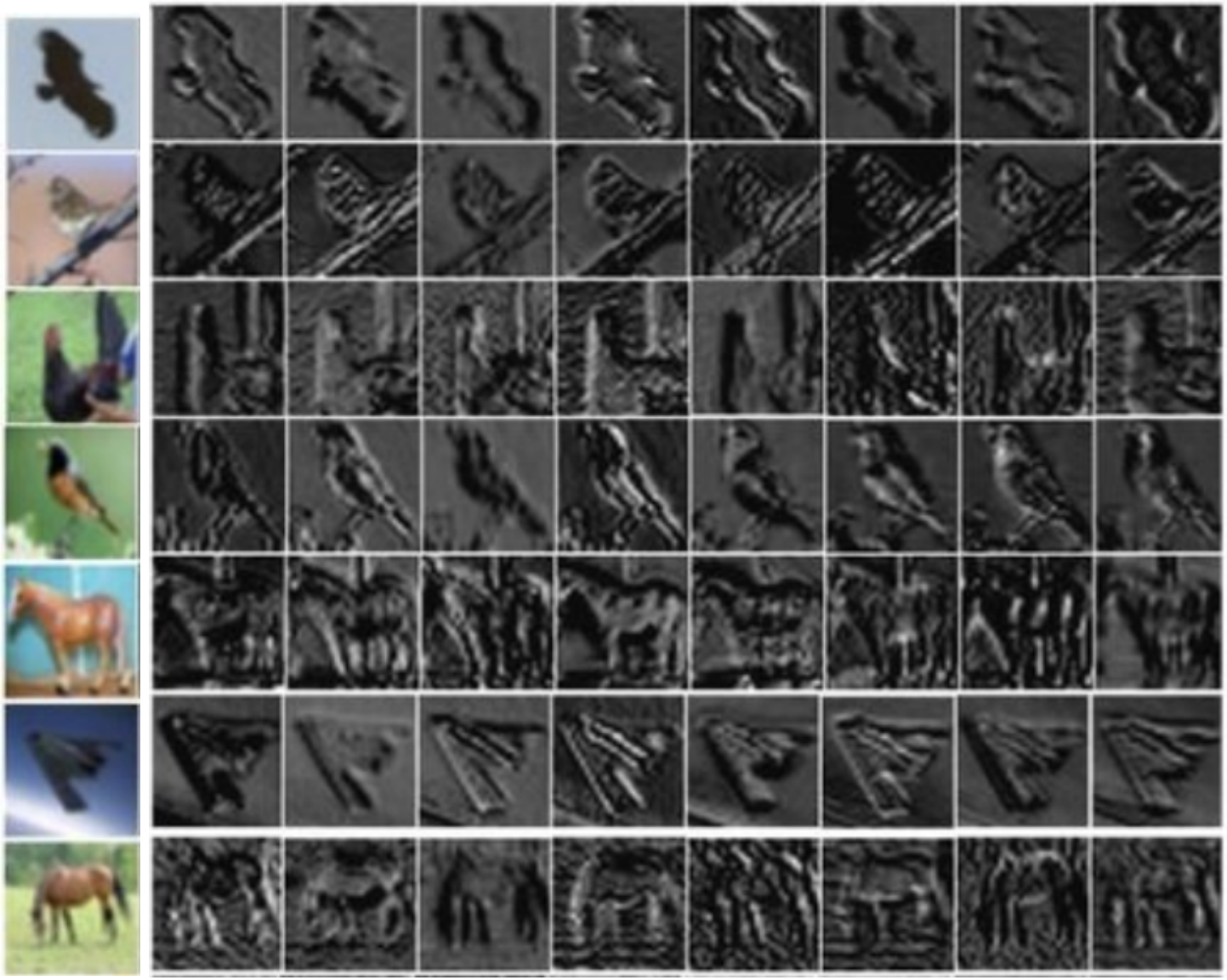
- 🧠 Les **coefficients** d'un filtre convolutif sont **appris** par un réseau de neurones:
  - Les poids des neurones correspondent aux coefficients du filtre.
  - Au début ils sont aléatoires 👀
  - Ils se précisent par rétropropagation.
  - Certains deviennent sensibles aux diagonales, d'autres aux textures...

Feature map:

- En sortie de l'application d'un filtre on obtient une **feature map**.
- La **feature map** met en évidence une caractéristique spécifique détectée sur l'image:



Feature maps



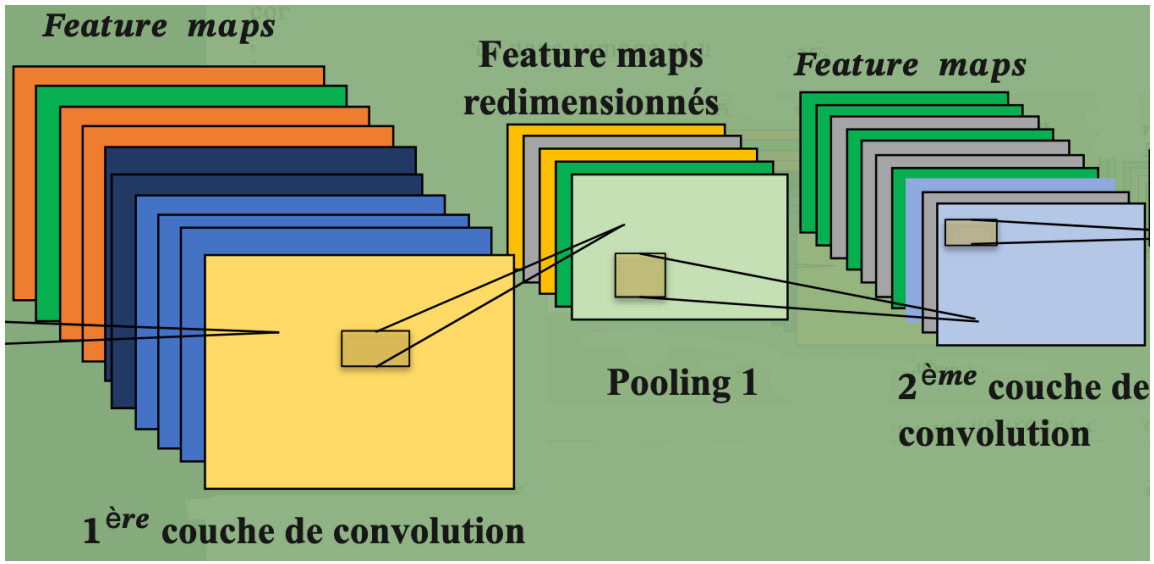
Couches convolutives

- A chaque couche de convolution, plusieurs **filtres** sont appliqués en parallèle $\Rightarrow$ plusieurs **feature maps** en sortie.
- L'ensemple des **feature maps** en sortie d'une couche convolutive forme l'entrée de la couche convolutive suivante, où à nouveau, plusieurs filtres sont appliqués en parallèle, et ainsi de suite.

- Les premières couches détectent des motifs simples (bords, textures).
- Les couches profondes combinent ces motifs pour reconnaître des structures complexes (yeux, visages, etc.).
- Cela permet au réseau d'extraire simultanément différents types de caractéristiques de plus en plus complexes.

Couches convolutives

- A chaque couche de convolution, plusieurs **filtres** sont appliqués en parallèle $\Rightarrow$ plusieurs **feature maps** en sortie.
- L'ensemple des **feature maps** en sortie d'une couche convolutive forme l'entrée de la couche convolutive suivante, où à nouveau, plusieurs filtres sont appliqués en parallèle, et ainsi de suite.



Entre chaque couche de convolution le réseau CNN applique une **couche de Pooling**.

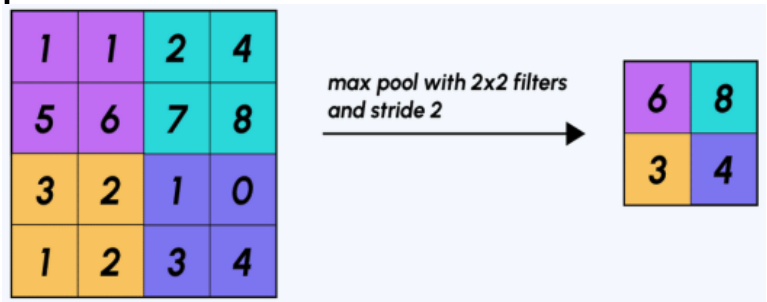
Partie de Pooling

Méthode de sous-échantillonage:

- **Max-pooling**: prendre la valeur maximale des valeurs de la fenetre.
- **Average-pooling**: prendre la valeur moyenne des valeurs de la fenetre.

Couche de Pooling:

- Appliquée entre 2 couches de convolution $\rightarrow$ pour réduire la dimension.
- Recoit en entrée les **feature maps** de la couche convolutive.
- Pour chaque fenetre balayée, avec un pas **stride**, garder la valeur du pooling, par exemple le max:



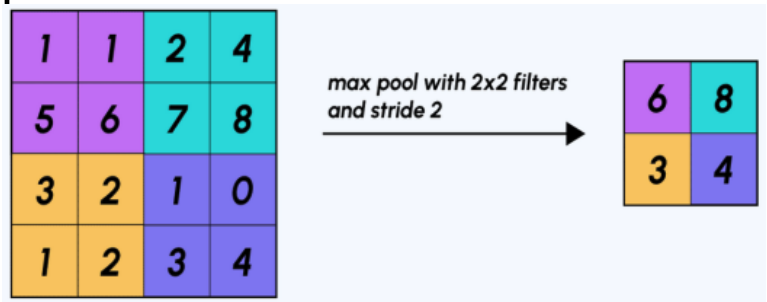
Partie de Pooling

Méthode de sous-échantillonage:

- **Max-pooling**: prendre la valeur maximale des valeurs de la fenetre.
- **Average-pooling**: prendre la valeur moyenne des valeurs de la fenetre.

Couche de Pooling:

- Appliquée entre 2 couches de convolution $\rightarrow$ pour réduire la dimension.
- Recoit en entrée les **feature maps** de la couche convolutive.
- Pour chaque fenetre balayée, avec un pas **stride**, garder la valeur du pooling, par exemple le max:



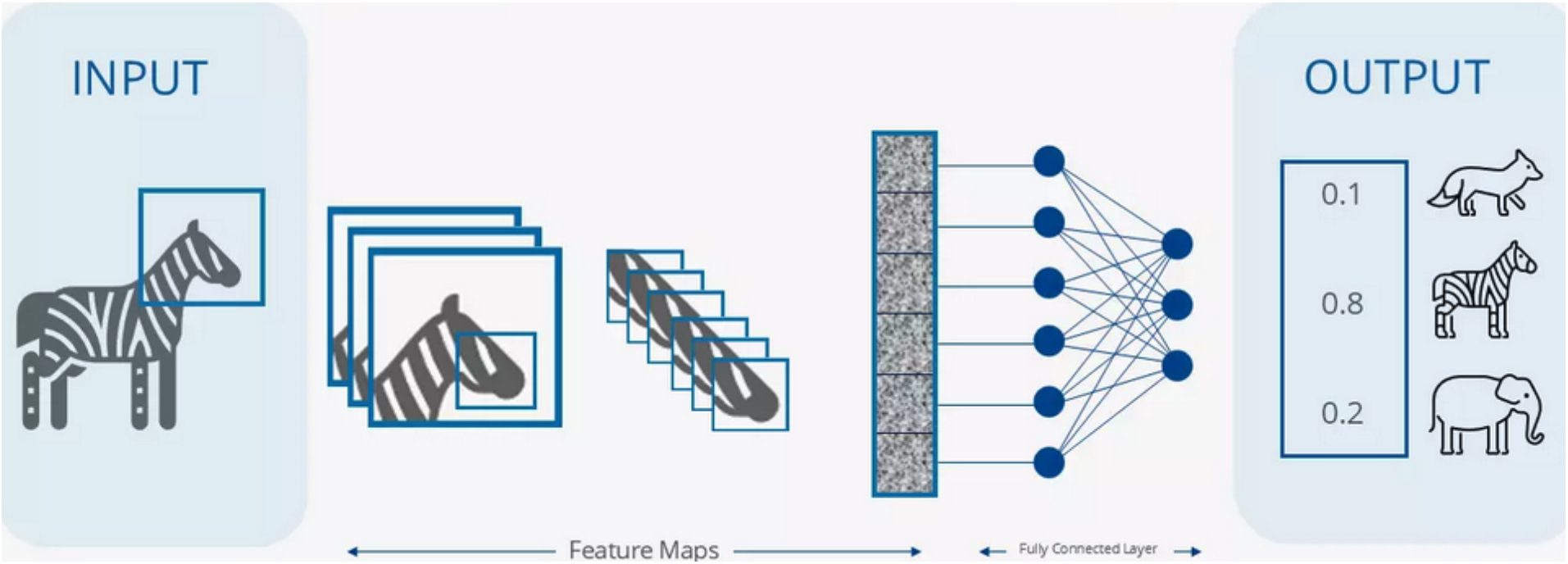
On repète convolution-pooling jusqu'à ce que la dimension soit suffisament petite.

Partie de classification

- On concatène toutes les **feature maps** pour former un vecteur.
- Ce vecteur est donné en entrée à un réseau fully-connected pour classification.
- Le réseau fully-connected renvoie une vecteur de taille C, où C est le nombre de classes, dans laquelle chaque composante est la probabilité pour l'image d'appartenir à une classe.

Partie de classification

- On concatène toutes les **feature maps** pour former un vecteur.
- Ce vecteur est donné en entrée à un réseau fully-connected pour classification.
- Le réseau fully-connected renvoie une vecteur de taille C, où C est le nombre de classes, dans laquelle chaque composante est la probabilité pour l'image d'appartenir à une classe.
- Au final, on sélectionne la classe ayant la plus grande probabilité.



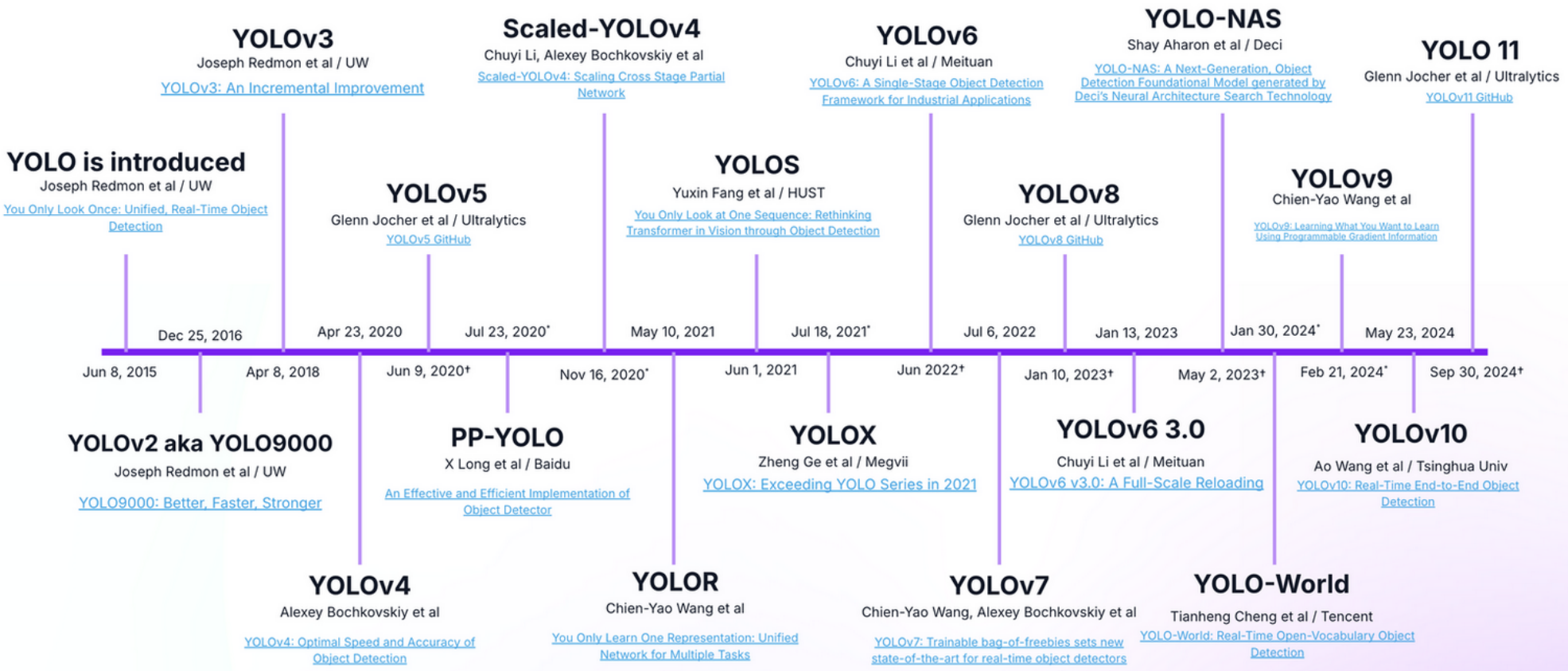
YOLO

Introduction

- 🚀 **YOLO** a été crée en 2016 par Joseph Redmon.
- En 2020 - **YOLOv3**, Jospeh Redmon annonce qu'il **arrête la recherche en IA** pour des raisons éthiques:
  - surveillance militaire,
  - atteinte à la vie privée.
- 🔥 2020 Ultralytics → **YOLOv5**
  - Code open-source 👀
  - Code en Python (vs Darknet - framework en C)
  - Sans publication de recherche.
- 📌 2023 → **YOLOv8**.

Versions YOLO

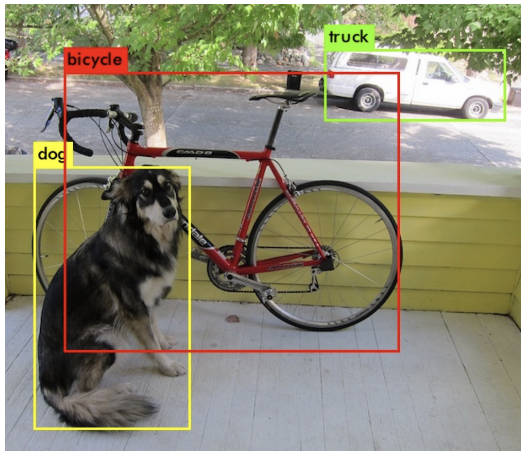
Versions de YOLO - de 2015 à 2025:



Détection d'objets

Pour une image donnée, l'objectif est de :

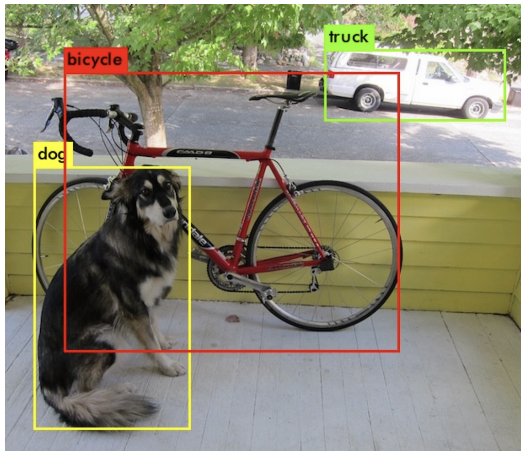
- Identifier où se trouvent les objets en générant des **boîtes englobantes** (bounding boxes).
- Prédire la classe de chaque objet détecté dans ces boîtes.



Détection d'objets

Pour une image donnée, l'objectif est de :

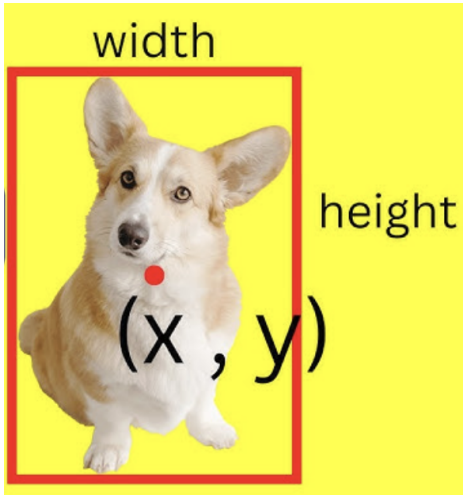
- Identifier où se trouvent les objets en générant des **boîtes englobantes** (bounding boxes).
- Prédire la classe de chaque objet détecté dans ces boîtes.



YOLO (You Only Look Once)

🎯 2016 - Joseph Redmon.

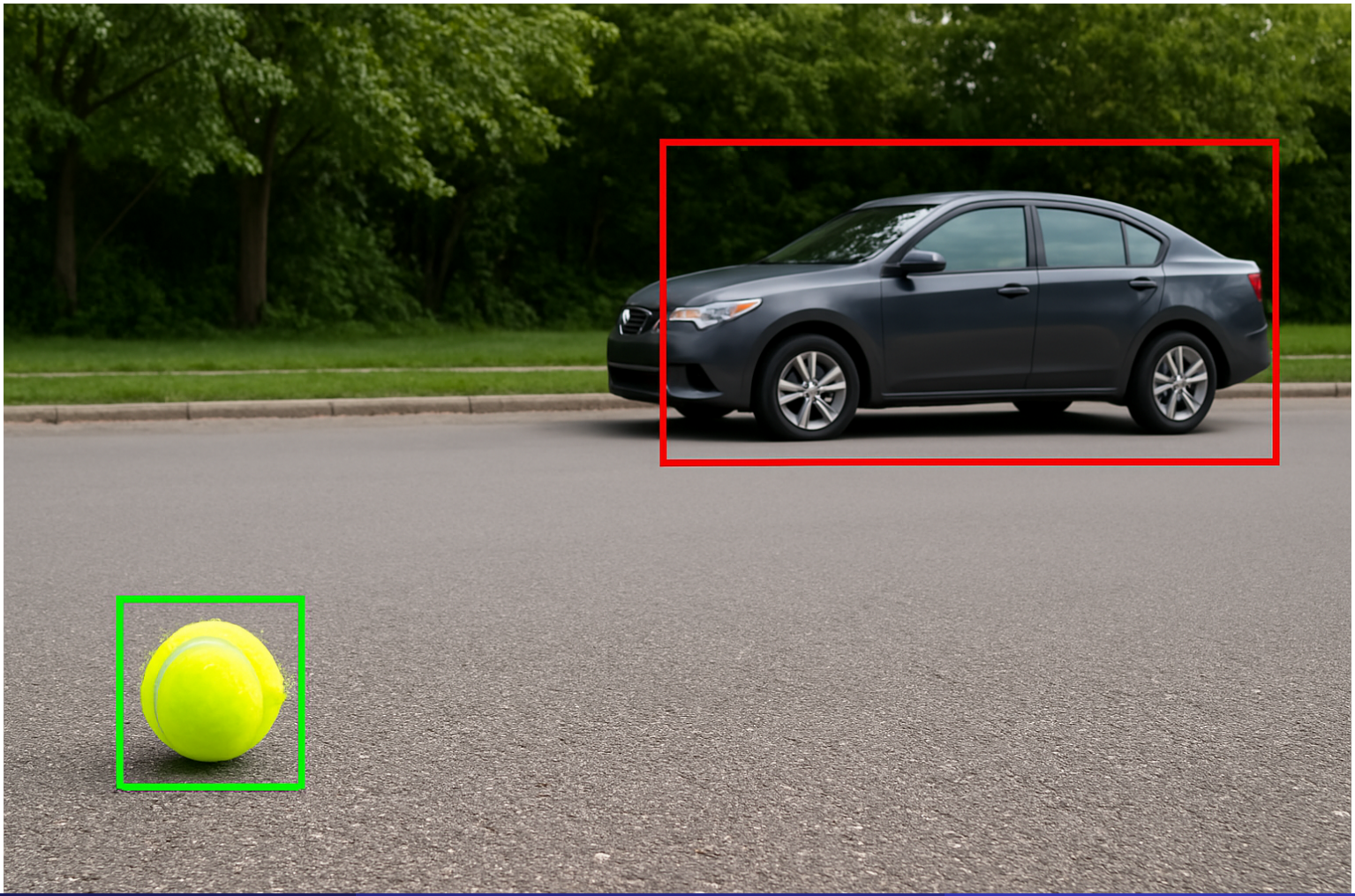
- YOLO est un algorithme de détection d'objets.
- YOLO combine localisation d'objets et classification d'objets en une seule étape.
- La localisation correspond aux coordonnées des **boîtes englobantes** $\Longrightarrow$



Données d'entrainement

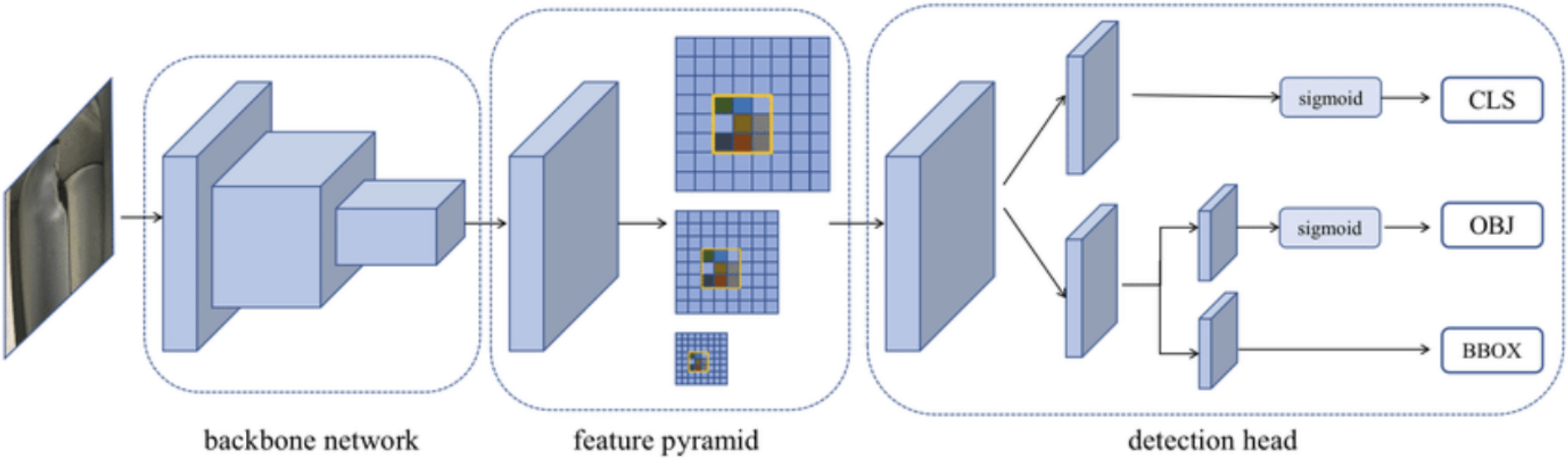
📌 Contrairement aux données d'entrainement du CNN, ici les images contiennent plusieurs objets, et de différentes tailles.

$\Rightarrow$ On ne peut pas appliquer une classification directement.



Architecture YOLO (v8)

- **Backbone:** réseau de convolution profond qui extrait des caractéristiques visuelles importantes comme les textures, formes et couleurs.
- **Neck:** réseau qui fusionne des caractéristiques de différentes échelles pour améliorer la détection des objets de tailles variées.
- **Head:** réseau qui prédit les boîtes englobantes et leurs classes.



Entrainement

Données d'entrainement:

- Les images d'entrainement contiennent plusieurs objets.
- Chaque image est annotée de la facon suivante, pour chaque objet:
  - (x, y) coordonnée du centre de la boite englobante,
  - (w, h) largeur et hauteur de la boite englobante,
  - c indice de confiance,
  - $p_1, p_2, ..., p_N$ probabilités des N classes d'objets (chien, chat...).

Entrainement

Données d'entrainement:

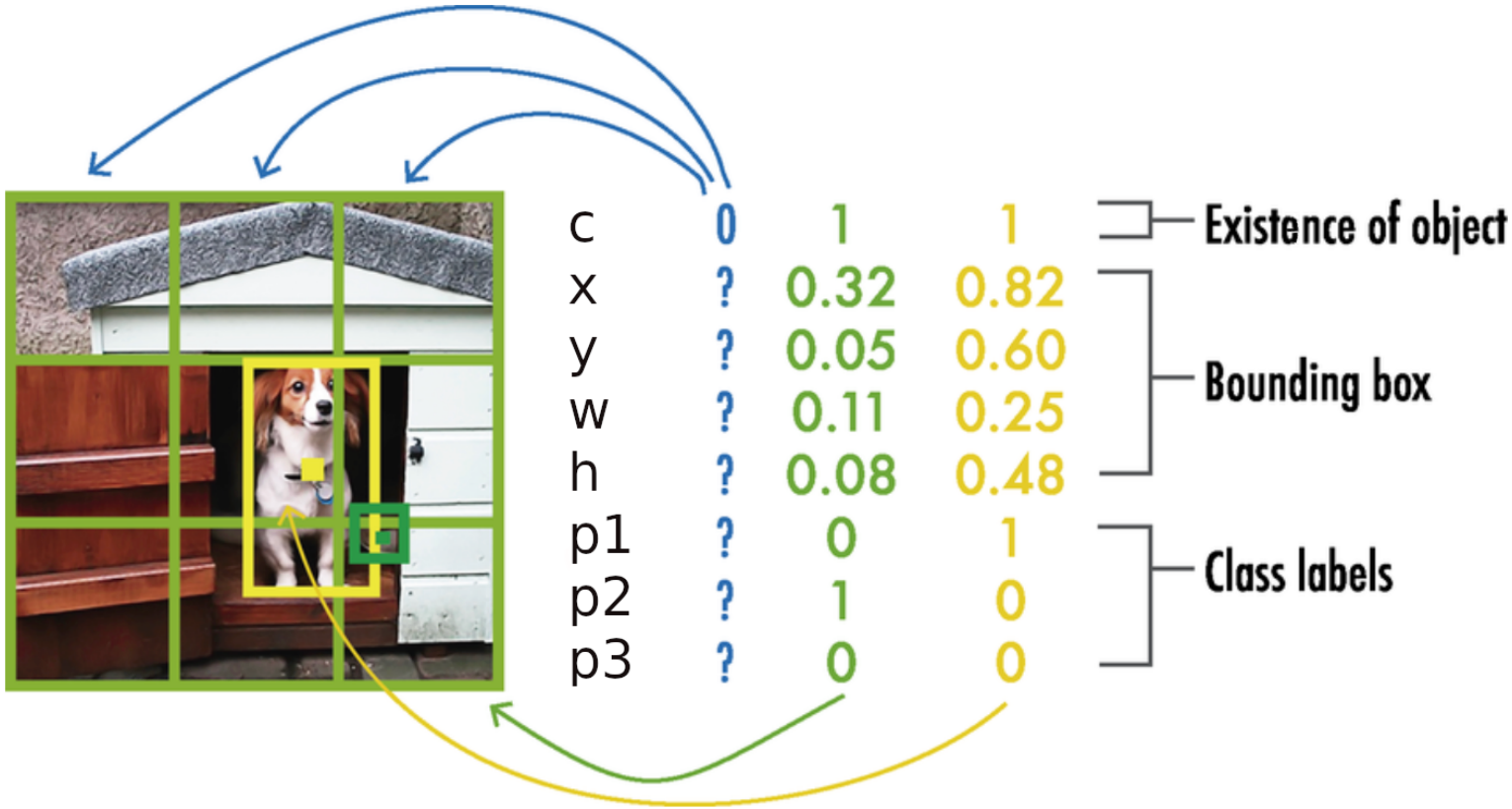
- Les images d'entrainement contiennent plusieurs objets.
- Chaque image est annotée de la facon suivante, pour chaque objet:
  - (x, y) coordonnée du centre de la boite englobante,
  - (w, h) largeur et hauteur de la boite englobante,
  - c indice de confiance,
  - $p_1, p_2, ..., p_N$ probabilités des N classes d'objets (chien, chat...).

Réseau CNN:

- L'image passe dans un backbone CNN qui extrait les caractéristiques visuelles.
- $\Rightarrow$ 3 feature maps:
  - 20×20 $\rightarrow$ détecte objets **petits**
  - 40×40 $\rightarrow$ détecte objets **moyens**
  - 80×80 $\rightarrow$ détecte objets **grands**

Entrainement

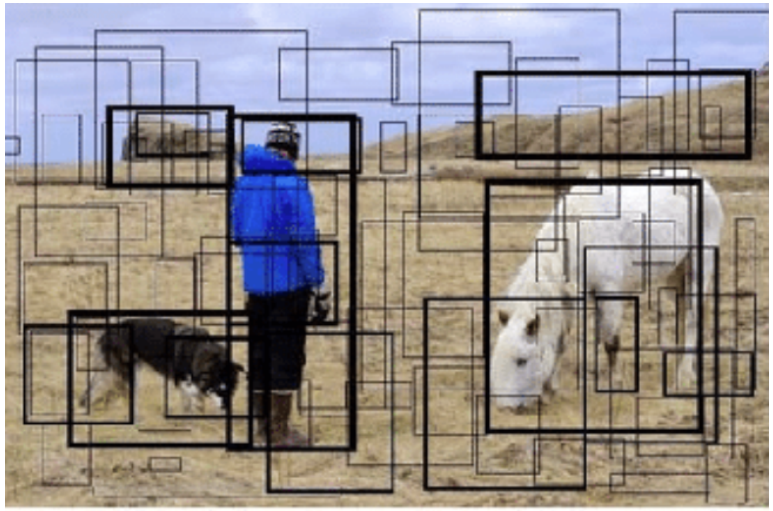
- Chaque feature map est divisée en une grille de taille $S \times S$.
- Chaque cellule de la grille prédit les paramètres:
  - (x, y) coordonnée du centre de la boite englobante relatives à la cellule,
  - (w, h) largeur et hauteur normalisées de la boite englobante,
  - c indice de confiance (probabilité qu'il y ait un objet),
  - $p_1, p_2, ..., p_N$ probabilités des N classes.



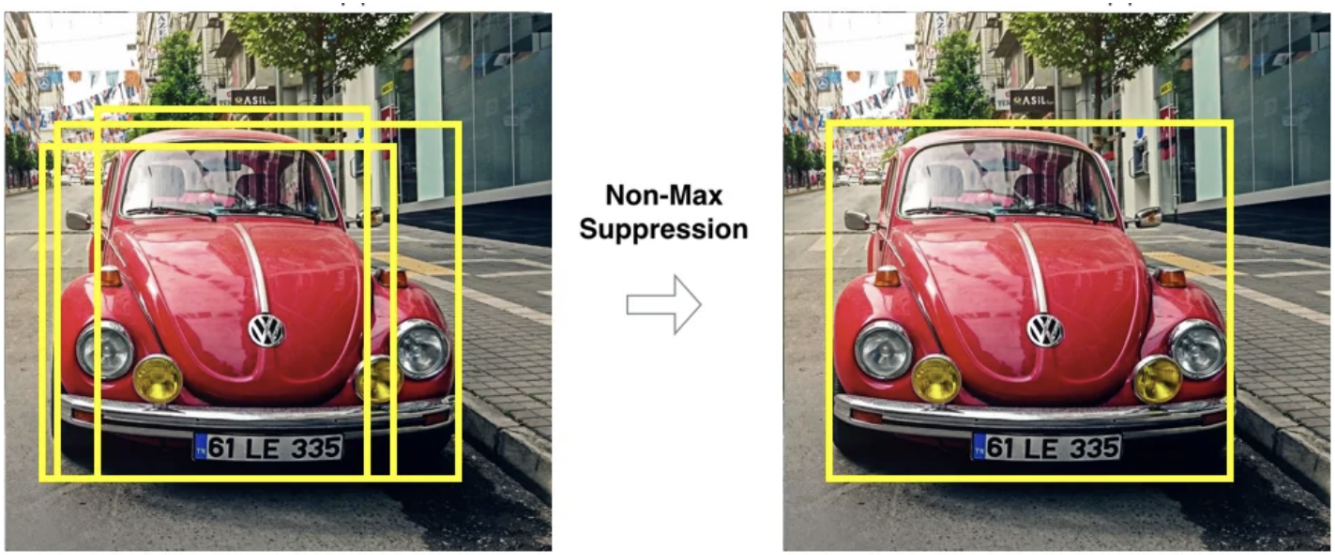
Inférence

Pour une image en entrée on a:

- pour chaque objet
- 3 prédictions (3 feature maps correspondant aux 3 tailles petit, moyen, grand)



- Filtrage: on sélectionne les boites avec $c > 0.5$
- On élimine les doublons par Non-maximum Suppression (NMS).



Non-maximal suppression

But: ne garder que la boite englobante la plus adaptée.

- L'idée est de supprimer les boîtes qui se chevauchent trop et qui représentent probablement le même objet.
- On calcul l'**Intersection over Union** (IoU):
  $IoU(A, B) = \frac{Aire(A \cap B)}{Aire(A \cup B)}$ mesure le degré de recouvrement.
- On élimine toutes les boîtes dont l'IoU avec la boîte sélectionnée dépasse un seuil prédéfini.

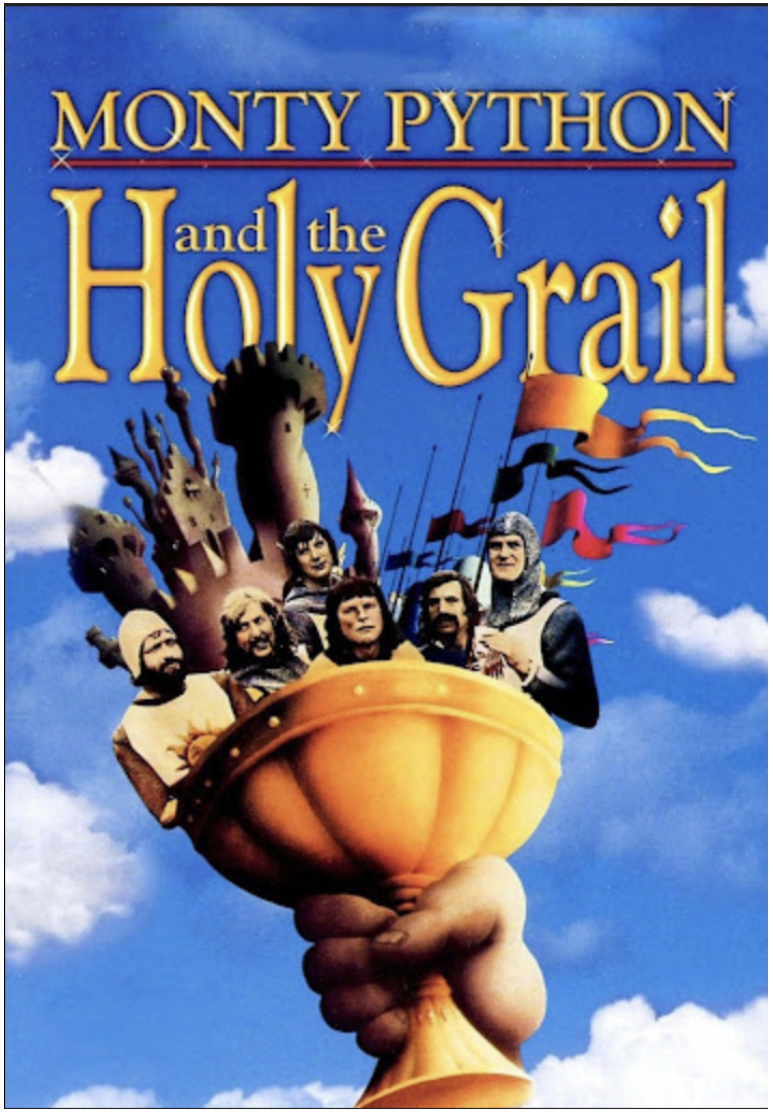
IoU = 0.922

IoU = 0.258

Introduction à Python

Python

- **Python** a été crée en 1991 par le développeur néerlandais Guido van Rossum.
- ✗ Le nom Python ne vient pas du serpent 🐍 ! Guido est un fan des **Monty Python** 😂



Monty Python

- Les **Monty Python** est un groupe de comédiens britanniques, humoristes.
- 🏆 ils ont révolutionné la comédie absurde dans les années 1960-1970.

Python

- Langage de programmation:
  - Interprété.
  - Open-source.

- Syntaxe claire et lisible 👀
  - Indentation obligatoire pour structurer le code.
  - Typage dynamique.
- Modules:
  - 👍 $\sim$ 300 modules inclus nativement.
  - +500 000 packages répertoriés sur PyPI 🔥
- Multi-paradigme : procédural, orienté objet, fonctionnel.
- Immense écosystème : science des données, web, cyber sécurité, IA...

Python en IA

🎯 Simple à lire, facile à maintenir.

- Calculs accélérés: GPU / TPU via CUDA (PyTorch, TensorFlow).
- Large communauté scientifique: publications, notebooks, frameworks.
- Support: StackOverflow, GitHub...

Librairies:

- NumPy, Pandas → manipulation de données.
- Scipy → calcul scientifique.
- Matplotlib, Seaborn, Plotly → visualisation.
- Scikit-learn → Machine Learning traditionnel.
- TensorFlow, PyTorch, Keras → Deep Learning.
- NLTK, spaCy, Transformers → NLP.
- OpenCV, Pillow, scikit-image → Computer Vision.

Exemple de code

Création d'un MLP - classification multi-classes:

```
import tensorflow as tf
from tensorflow.keras import layers

model = tf.keras.Sequential([
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

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

Exemple d'application

Anonymisation vidéo

Contexte:

- 🚗 Tests de roulage.
- 📹 Vidéos GoPro embarquées.
- Utilisée en interne pour état de la route etc...
- ⛔ Pour des raisons de **RGPD** il faut anonymiser ces vidéos.

Anonymisation vidéo

Contexte:

- 🚗 Tests de roulage.
- 📹 Vidéos GoPro embarquées.
- Utilisée en interne pour état de la route etc...
- ⛔ Pour des raisons de **RGPD** il faut anonymiser ces vidéos.

RGPD

Le RGPD (Règlement Général sur la Protection des Données) est un texte de loi européen entré en vigueur en mai 2018. Il encadre la collecte, le traitement, et la conservation des données personnelles des citoyens de l'Union Européenne.

Objectif: protéger la vie privée des personnes en leur donnant plus de contrôle sur leurs données personnelles.

Anonymisation

Anonymisation:

- 🚀 Détection des personnes et des voitures avec **YOLO**.
- Sélection des visages et des plaques d'immatriculation.
  - 🎯 Autres méthodes:
    - Modèle pré-entrainé de détection des visages.
    - Entrainement d'un modèle de détection des plaques d'immatriculation.

Anonymisation

Anonymisation:

- 🚀 Détection des personnes et des voitures avec **YOLO**.
- Sélection des visages et des plaques d'immatriculation.
  - 🎯 Autres méthodes:
    - Modèle pré-entrainé de détection des visages.
    - Entrainement d'un modèle de détection des plaques d'immatriculation.
- Floutage des visages et des plaques d'immatriculation détectés:

Anonymisation

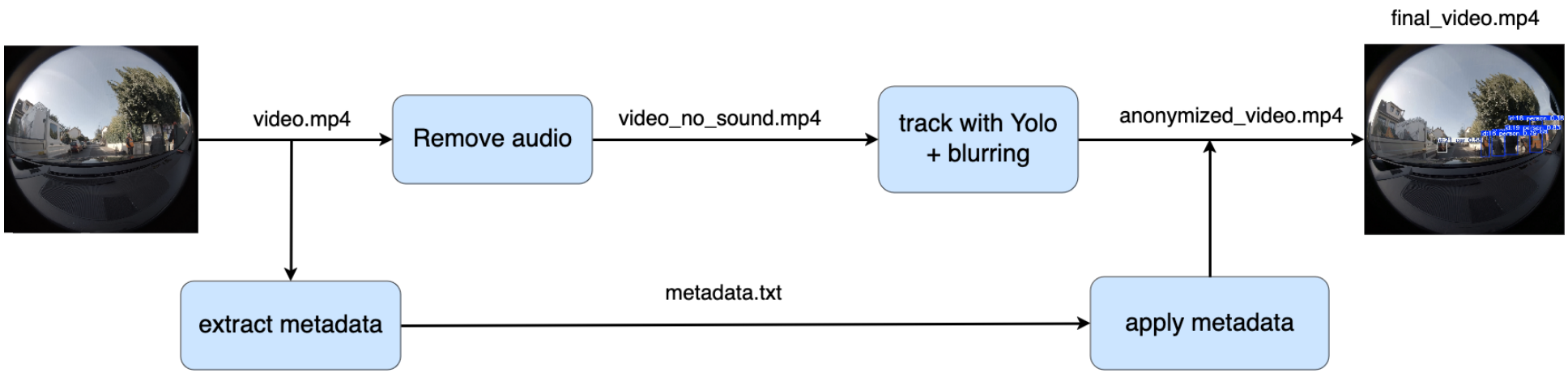
Anonymisation:

- 🚀 Détection des personnes et des voitures avec **YOLO**.
- Sélection des visages et des plaques d'immatriculation.
  - 🎯 Autres méthodes:
    - Modèle pré-entrainé de détection des visages.
    - Entrainement d'un modèle de détection des plaques d'immatriculation.
- Floutage des visages et des plaques d'immatriculation détectés:
  - 👉 application d'un **filtre** Gaussien.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Pipeline de traitement vidéo

- Extraction de la télémétrie et des métadonnées de la vidéo.
  - GPS.
  - Vitesse, etc...
  - Données associées à des timestamps (horodatages).
- ✂️ Découpage de la vidéo en frame (image par image).
- 🕶️ Anonymisation des frames.
- Fusion des frames anonymisés pour reconstruire la vidéo.
- Ré-injection de la télémetrie/métadonnées dans la vidéo anonymisée.



Thanks