

# **Cours-TD d'introduction à l'Intelligence Artificielle**

## **Partie V**

# **Les réseaux Profonds**

Simon Gay

# Introduction à l'Intelligence Artificielle

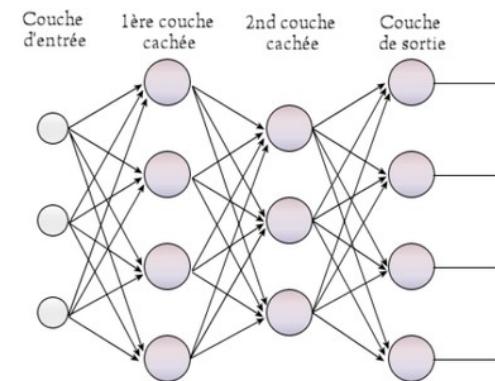
- **Menu :**
  - Théorie :
    - Emergence de l'apprentissage profond
    - Le neurone convolutif
    - Les structures d'un réseau DL
    - Quelques architectures remarquables
    - Biais statistiques et sécurité
    - Les GANs
    - TensorFlow, une librairie pour le DL

# Introduction à l'Intelligence Artificielle

- **La reconnaissance d'images**

- Approche 'classique' :

- Détections de features (cercles, droites, rectangles...)



- On définit un vecteur de valeurs à partir de ces features, leur nombre, leur position...
    - Un réseau de neurone détermine ensuite le contenu de l'image
      - Pas très fiable, >25 % d'erreurs à l'ILSVRC 2011

# Introduction à l'Intelligence Artificielle

- **L'apprentissage profond**

- Les réseaux de neurones peuvent-ils définir ces features par apprentissage ?
- Problèmes :
  - Plus il y a de couches, plus il y a de neurones et de connexions
  - Plus il y a de neurones, plus il faut d'exemples pour les entraîner
    - si le nombre d'exemples est trop faible, il y a risque de **sur-apprentissage** : le réseau devient 'expert' sur les modèles d'entraînement...
    - ...mais devient incapable de généraliser sur d'autres cas
  - Plus de neurones implique aussi plus de ressources
- Pendant plusieurs décennies, les réseaux ont été limités à la fois par la puissance de calcul disponible mais aussi par la quantité de données utilisables

# Introduction à l'Intelligence Artificielle

- **L'apprentissage profond**

- Depuis le milieu des années 2000 :
  - Augmentation de la puissance de calcul des ordinateurs
    - Parallélisation sur GPU (e.g. CUDA, 2007)
    - Architectures dédiées (Tensor Processing Unit, 2016)
  - Augmentation des données récoltées
    - GAFA et la vie privée
    - Big data
    - Captcha
    - Volontarisme
  - Dataset libre COCO (Common Objects in COntext) :  
>200k images labellisées, 80+ classes

# Introduction à l'Intelligence Artificielle

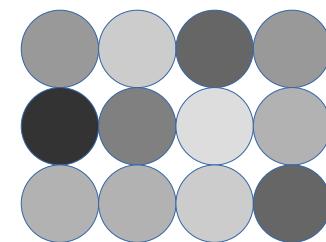
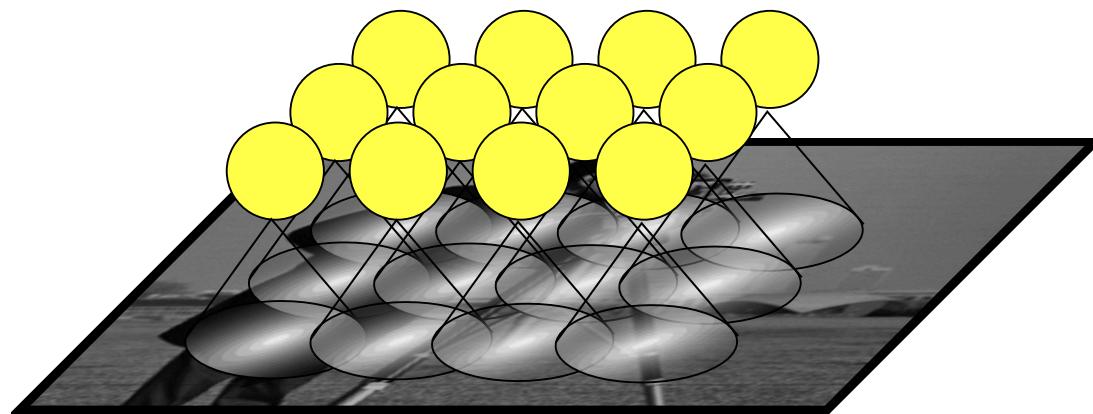
- **L'apprentissage profond**

- Mais pas toujours suffisant !
  - Exemple :  
soit une image de 100x100 pixels : chaque neurone de la première couche contiendra 10 000 connexions !
  - Inspiration biologique :
    - 1968 : des analyses montrent que chez l'animal, les neurones visuels des premières couches de traitement sont connectés à une petite région de la rétine

# Introduction à l'Intelligence Artificielle

- **Le neurone convolutif**

- Les neurones de la rétine et des premières couches de traitement visuels (aire V1) ne traitent qu'une petite partie de la rétine

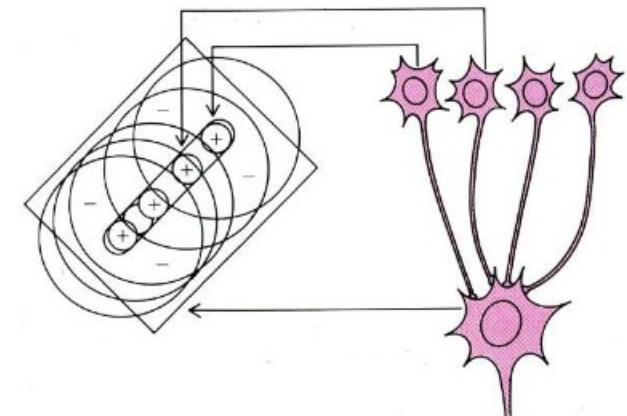
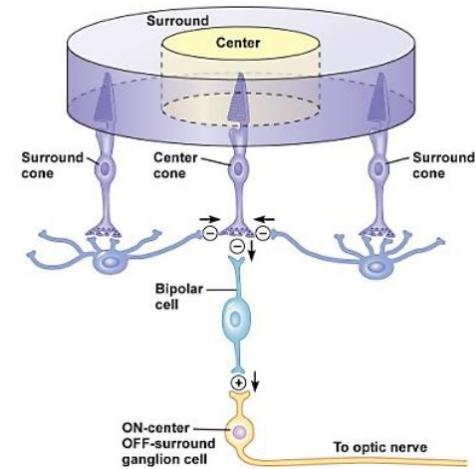


- Ces neurones détectent des features de très bas niveau : contour avec une certaine orientation, points...
- De façon surprenante, beaucoup de neurones partagent un même motif de détection
  - détection d'une même feature sur l'ensemble de la rétine

# Introduction à l'Intelligence Artificielle

- **Le neurone convolutif**

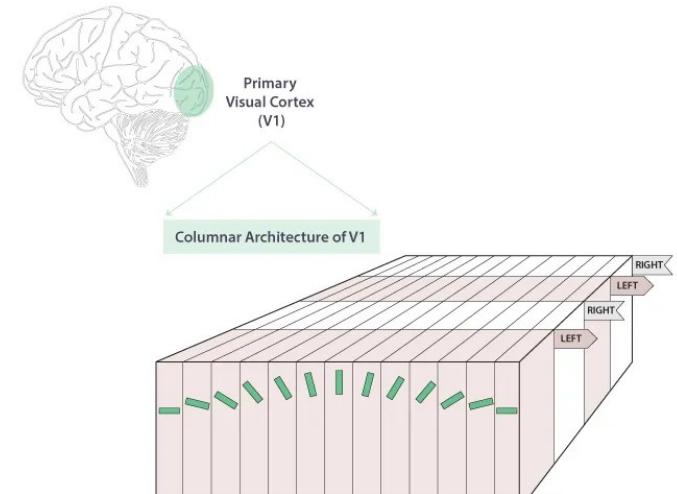
- Dans la rétine :
  - Certains neurones détectent des maximums ou minimum de luminosité (neurones ON-OFF et OFF-ON)
- Dans le cortex visuel (région V1) :
  - Ces neurones permettent de détecter des lignes dans une direction particulière



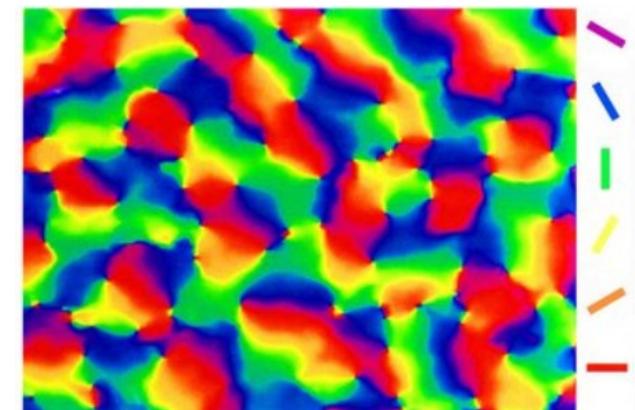
# Introduction à l'Intelligence Artificielle

## • Le neurone convolutif

- Dans le cortex visuel :
  - La région V1 contient des ‘colonnes’ de neurones qui détectent des contours dans une orientation prédéfinie
  - Deux colonnes adjacentes ont une orientation proche
  - Ces colonnes détectent l’ensemble des orientations sur la surface de la rétine de façon presque uniforme (à l’échelle du millimètre)



© Knowing Neurons <http://knowingneurons.com>

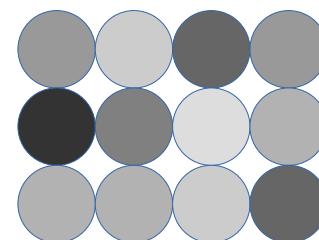
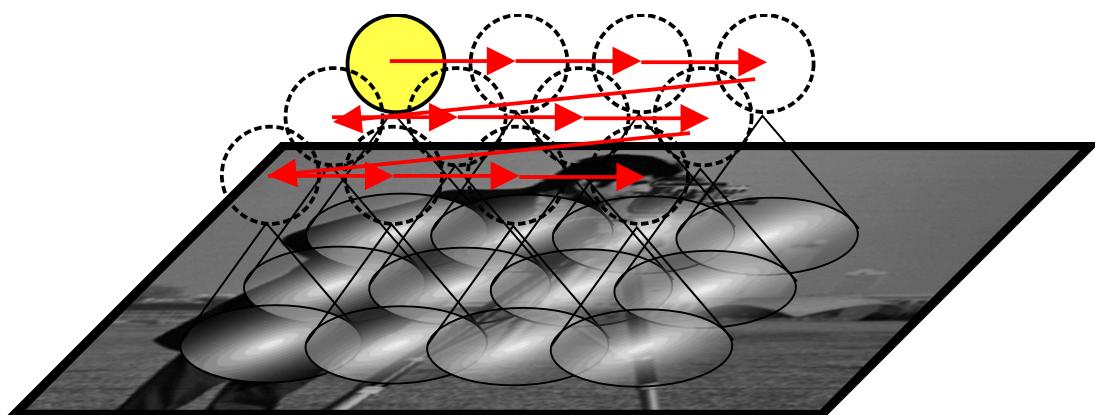


1mm  
Aire V1 du macaque

# Introduction à l'Intelligence Artificielle

- **Le neurone convolutif**

- Yann Le Cunn (milieu années 2000) :
  - Idée : chaque neurone définit une feature à détecter, chaque neurone est appliqué en toute position de la rétine



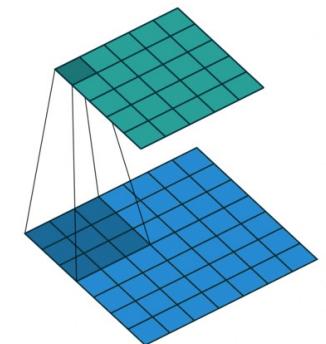
- Même résultat mais avec beaucoup moins de poids à optimiser !
- Fonctionnement proche d'une convolution → *neurones convolutifs*

# Introduction à l'Intelligence Artificielle

- **Les convolutions**

- Un noyau de convolution (kernel)

$$\begin{bmatrix} 0 & 0,25 & 0,5 & 0,25 & 0 \\ 0,25 & 0,5 & 0,75 & 0,5 & 0,25 \\ 0,5 & 0,75 & 1 & 0,75 & 0,5 \\ 0,25 & 0,5 & 0,75 & 0,5 & 0,25 \\ 0 & 0,25 & 0,5 & 0,25 & 0 \end{bmatrix}$$



<https://github.com/vdumoulin>

- On fait le produit scalaire de ce kernel à chaque position de l'image
- Le résultat obtenu en chaque position permet de générer une image donnant la similitude de l'image avec le kernel en chaque point de l'image initiale



$$\otimes \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} =$$



# Introduction à l'Intelligence Artificielle

- **Les convolutions**

- Détection de contours, bordures verticales et bordures horizontales

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

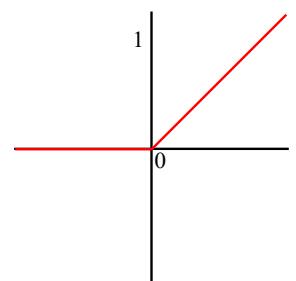
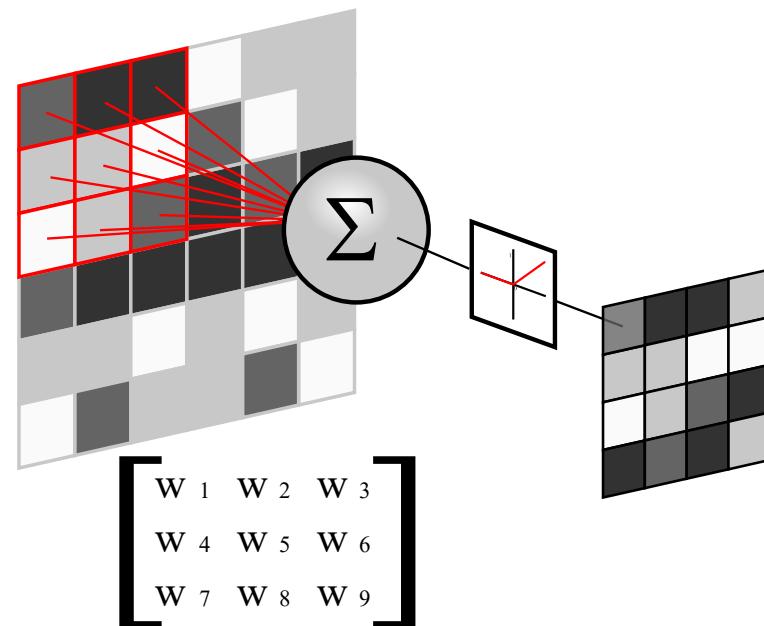
$$\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$



# Introduction à l'Intelligence Artificielle

- **Les neurones convolutifs**

- Similaire à un neurone formel
  - Vecteur d'entrées  $X$
  - Un ensemble de poids  $W$   
→ le kernel
  - Une fonction d'activation
- Fourni une 'image' en sortie
  - Chaque pixel  $p_{xy}$  est la sortie du neurone en  $(x,y)$  sur l'image d'entrée.
- La fonction RELU est utilisée de façon très majoritaire
  - On recherche la présence et non l'absence d'une feature



# Introduction à l'Intelligence Artificielle

- Les neurones convolutifs

image

0,88	0,18	0,89	0,05	0,78	0,63	0,87	0,68
0,94	0,74	0,13	0,95	0,05	0,92	0,03	0,6
0,22	0,34	0,65	0,82	0,08	0,53	0,9	0,75
0,31	0,33	0,18	0,21	0,44	0,29	0,12	0,87
0,99	0,25	0,85	0,88	0,02	0,25	0,19	0,05
0,4	0,44	0,86	0,42	0,75	0,99	0,48	0
0,49	0,59	0,07	0,34	0,04	0,64	0,99	0,3
0,66	0,4	0,12	0	0,06	0	0,2	0,99

kernel du neurone

0,5	0	-1
0	0,5	0
-1	0	0,5



ReLU

0,03	0,18	-0,5	-1,1	0,35	-0,5
0,29	-0,5	0,47	-0,5	-0,1	0,46
-0,9	-0,4	-0,5	-0,7	-0,6	-0,7
0,13	0,15	-0,4	-0,1	-0,3	-1,6
-0,6	-0,7	0,57	0,55	0,77	-0,2
-1	-0,6	-0,2	-0,8	0,26	1,49

0,03	0,18	0	0	0,35	0
0,29	0	0,47	0	0	0,46
0	0	0	0	0	0
0,13	0,15	0	0	0	0
0	0	0,57	0,55	0,77	0
0	0	0	0	0,26	1,49

- On remarque que l'image de convolution est plus petite

# Introduction à l'Intelligence Artificielle

- L'apprentissage profond

#### - Le padding :

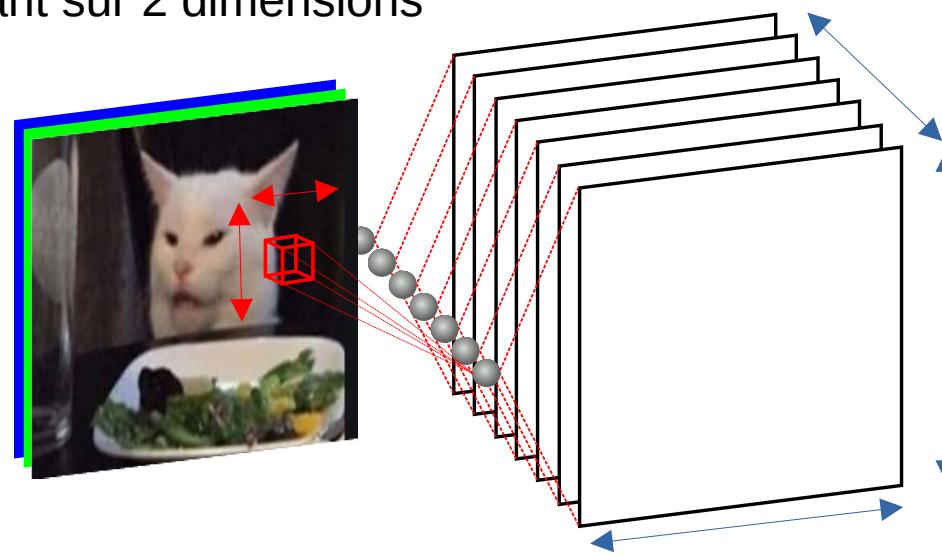
- Pour avoir une image de convolution de la même taille que l'image d'origine, il faut ajouter des pixels autour de l'image
  - Le padding consiste à ajouter des pixels à 0 autour de l'image
    - Pour un kernel de taille  $n$ , il faut ajouter  $\text{int}(n/2)$  pixels autour de l'image

0,88	0,18	0,89	0,05	0,78	0,63	0,87	0,68
0,94	0,74	0,13	0,95	0,05	0,92	0,03	0,6
0,22	0,34	0,65	0,82	0,08	0,53	0,9	0,75
0,31	0,33	0,18	0,21	0,44	0,29	0,12	0,87
0,99	0,25	0,85	0,88	0,02	0,25	0,19	0,05
0,4	0,44	0,86	0,42	0,75	0,99	0,48	0
0,49	0,59	0,07	0,34	0,04	0,64	0,99	0,3
0,66	0,4	0,12	0	0,06	0	0,2	0,99

# Introduction à l'Intelligence Artificielle

- **L'apprentissage profond**

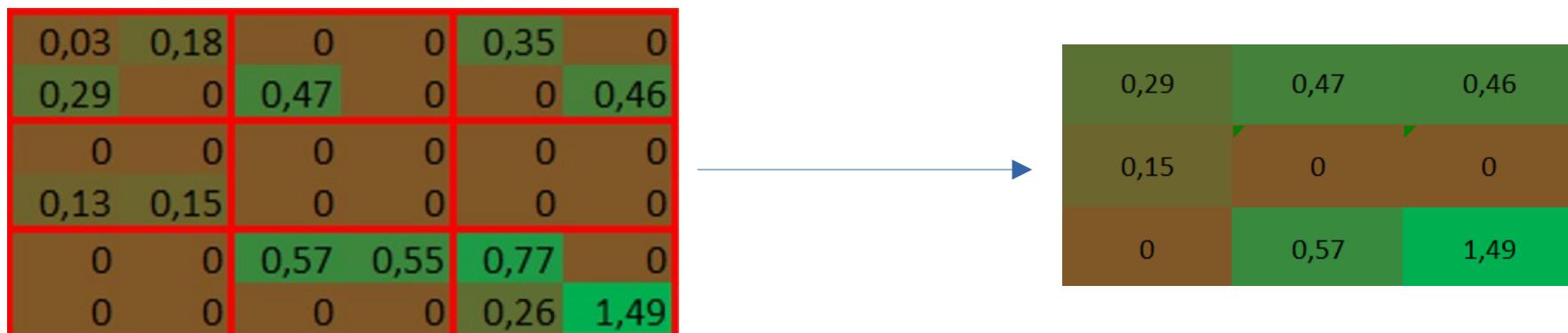
- Chaque neurone convolutif génère une convolution différente
- Un ensemble de neurones convolutif génère un ensemble d'images de convolution
  - Ces images sont rassemblées en une seule, avec une profondeur
  - La profondeur est le nombre de 'layers' de l'image
    - À noter, une image RVB a une profondeur de 3
  - Le kernel d'un neurone a 3 dimensions (largeur, hauteur et profondeur) et se déplaçant sur 2 dimensions



# Introduction à l'Intelligence Artificielle

- **Le pooling**

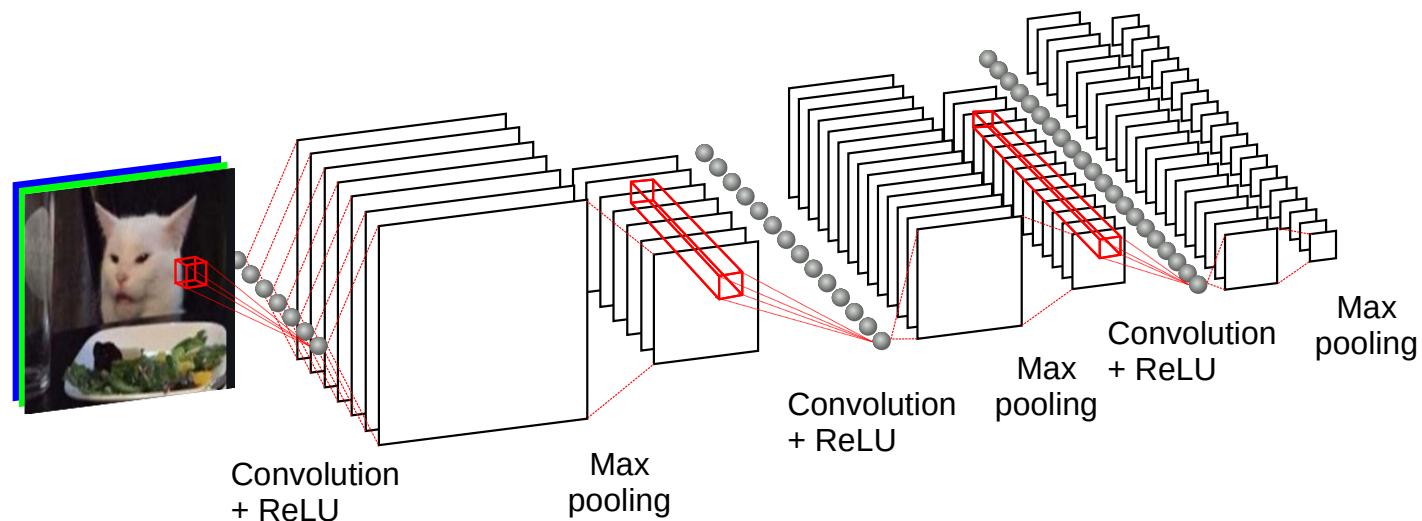
- Sur l'image de convolution, il n'est pas nécessaire de conserver deux pixels contigus de valeur élevée → on cherche seulement la présence de features
- Étape de Pooling
  - On réduit le nombre de pixels à traiter par la couche suivante
  - Les pixels sont regroupés par paquets
    - Average pooling → on fait la moyenne des pixels du groupe
    - Max pooling → on prend le maximum (méthode la plus répandue)



# Introduction à l'Intelligence Artificielle

- **Neurones convolutifs**

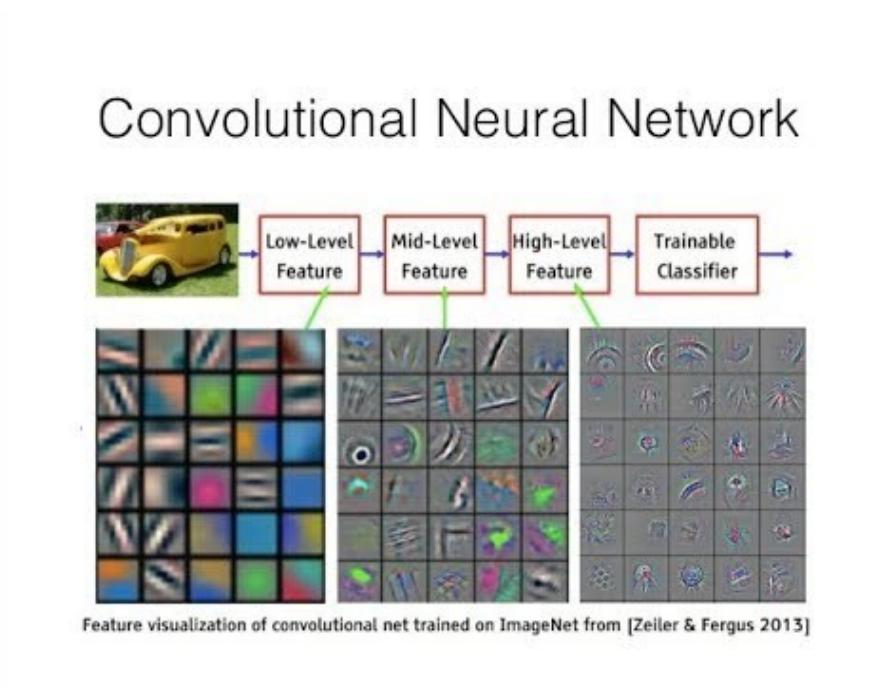
- On peut enchaîner les étapes de convolution et pooling
- De couche en couche, l'image perd en taille mais gagne en profondeur
  - On perd la spatialisation des éléments mais on gagne en information sur la présence de features.



# Introduction à l'Intelligence Artificielle

- **Neurones convolutifs**

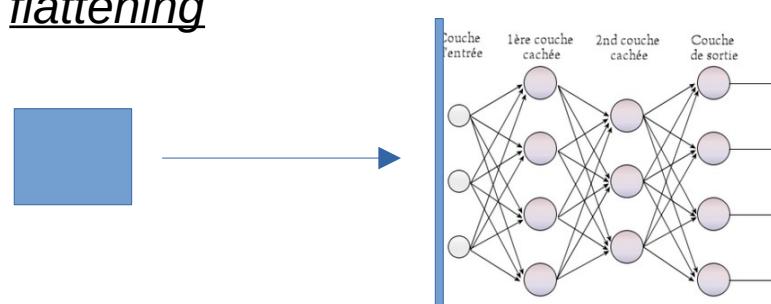
- Les couches les plus basses réagissent à des features très simples (comme la région V1 du cortex visuel!)
- Chaque couche supplémentaire combine des features de la couche précédente pour former des features plus complexes



# Introduction à l'Intelligence Artificielle

- **Couches fully-connected**

- Avec chaque étape de convolution-pooling, l'image ressemble de plus en plus à un vecteur de features !
  - on va pouvoir l'utiliser comme vecteur d'entrés d'un réseau 'normal'
  - On converti l'image de la dernière couche de convolution en un vecteur à une dimension (si l'image a une taille >1 pixel)
    - étape de flattening

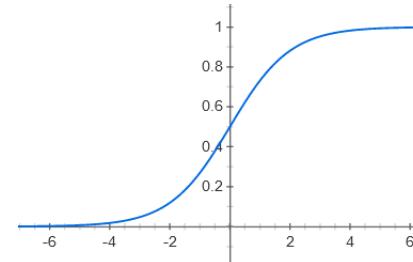
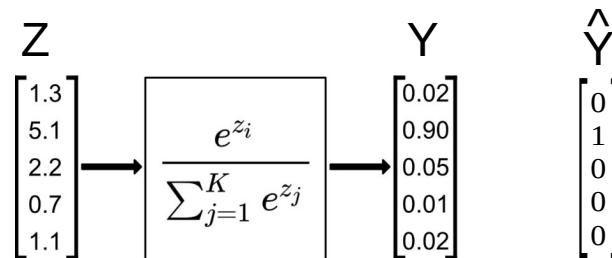


- On utilise ce vecteur comme entrée pour un réseau de neurone qui servira à interpréter ces features
  - Ces couches de neurones sont appelées 'fully connected' (par opposition aux couches convolutives)

# Introduction à l'Intelligence Artificielle

- **Fonction de coût (loss) :**

- On définit une unique fonction de coût pour l'ensemble des sorties
  - Quand on a plusieurs classes, on utilise en général sur la couche de sortie la fonction softmax comme fonction d'activation
    - Aussi appelée *fonction exponentielle normalisée*



- Converti le vecteur de sorties en vecteur de probabilité :

La somme des valeurs est toujours égale à 1

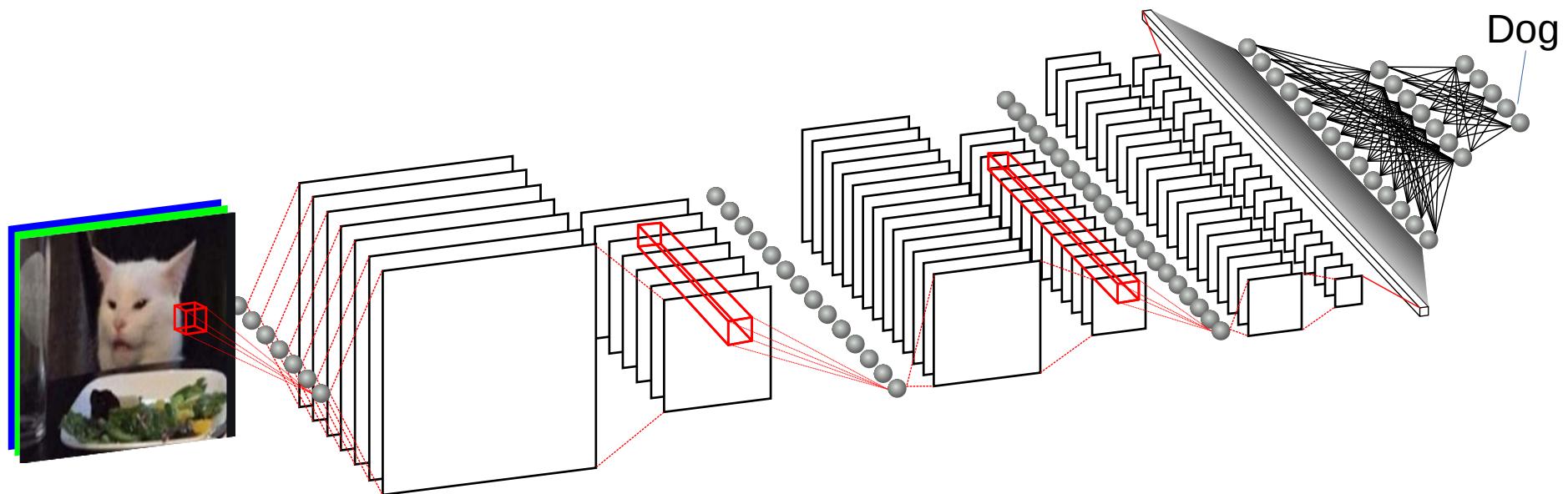
Si une sortie augmente, les autres baissent

- On définit la fonction de coût :  $E(Y) = -\sum_{i=1}^n \hat{y}_i \times \ln(y_i)$
- Et le delta :  $\delta_i = \hat{y}_i - y_i$

entropie croisée

# Introduction à l'Intelligence Artificielle

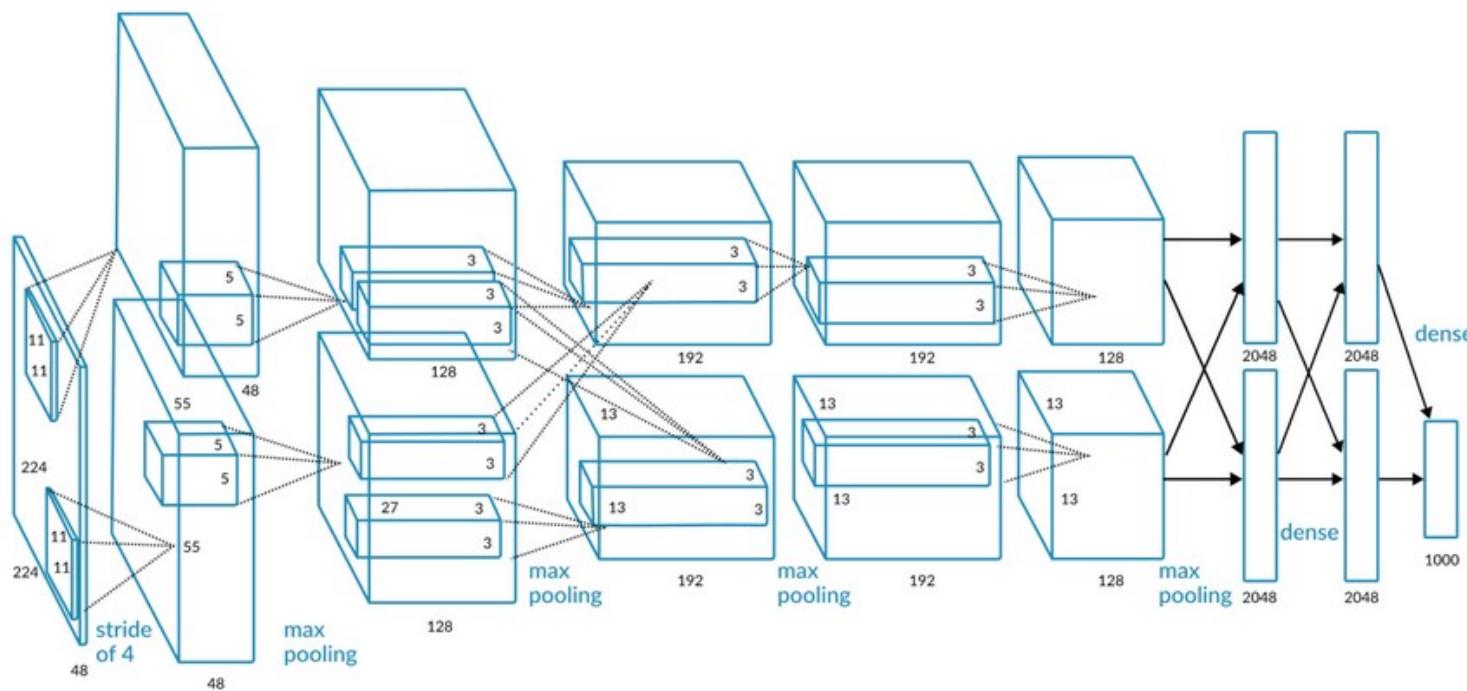
- Architecture 'classique' d'un réseau profond convolutif



- Neurones de convolution + ReLU
- Pooling
- Neurones de convolution + ReLU
- Pooling
- ...
- Flattening
- Couches fully connected
- Couche de sortie

# Introduction à l'Intelligence Artificielle

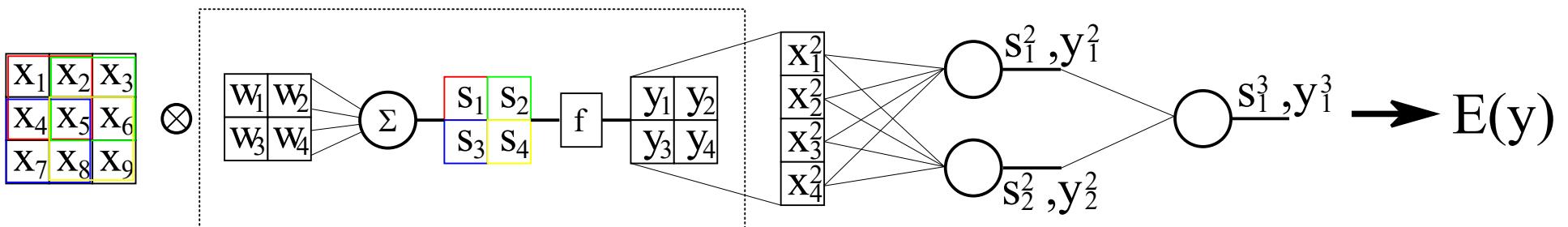
- Mais l'architecture peut être plus complexe :  
exemple : AlphaNet (2012)



# Apprentissage des neurones convolutifs

# Introduction à l'Intelligence Artificielle

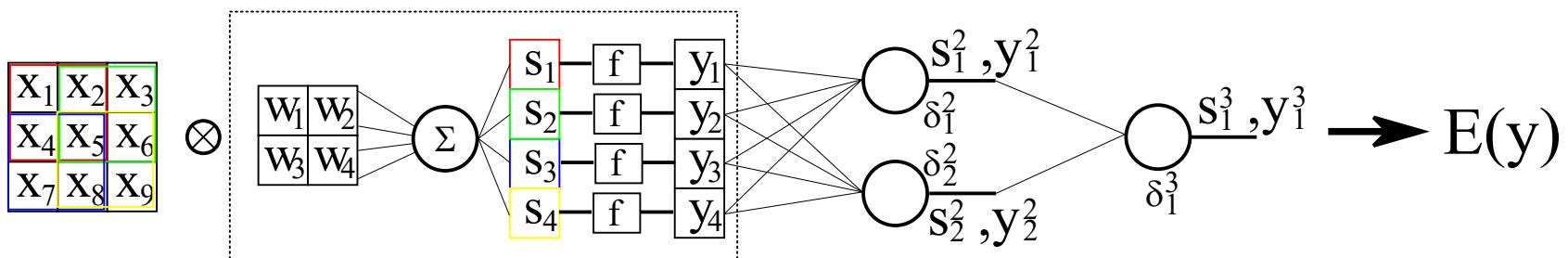
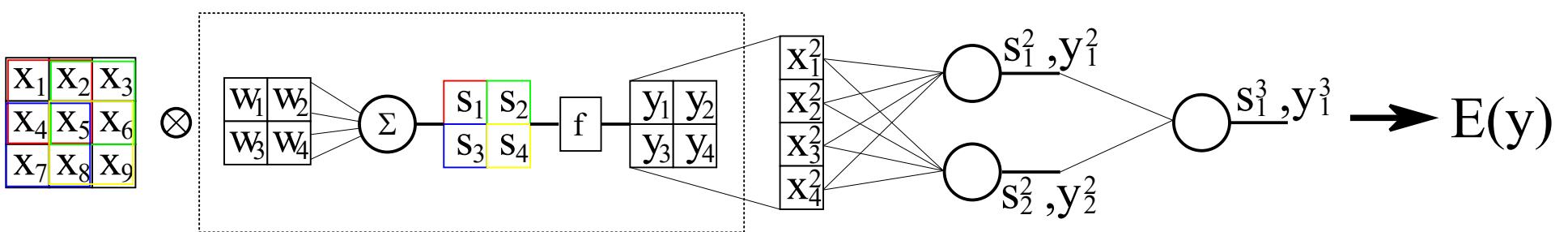
- **Descente de gradient du gradient du neurone convolutif**
  - Prenons un cas très simplifié
    - Un seul neurone convolutif avec 4 poids (kernel de 2x2)
    - Une image de 3x3 pixels, sans padding
      - l'image de convolution fait 2x2 pixels
      - la première couche cachée a donc pour entrée un vecteur de 4 éléments
    - Un réseau fully connected avec deux couches



# Introduction à l'Intelligence Artificielle

- **Descente de gradient du gradient du neurone convolutif**

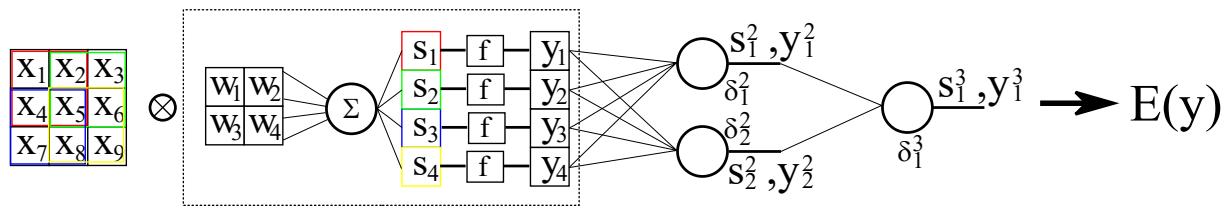
- Développons les « chemins »



- On cherche toujours à calculer  $\frac{\partial E(y)}{\partial w_i}$
  - On connaît les  $\delta$  des neurones fully connected

# Introduction à l'Intelligence Artificielle

- Descente de gradient du gradient du neurone convolutif



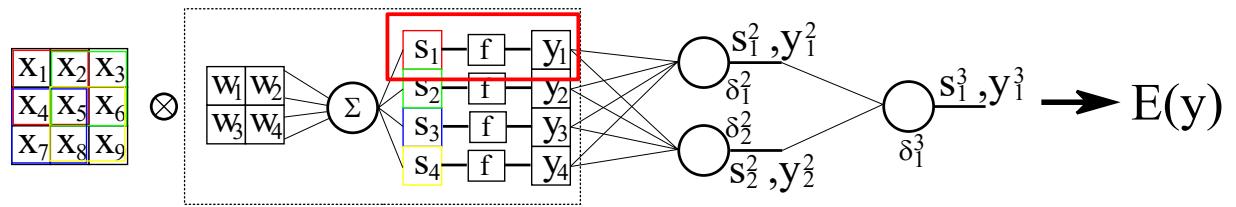
- Théorème de dérivation des fonctions composées (via  $y_1, y_2, y_3$  et  $y_4$ )

$$\begin{aligned} \frac{\partial E(y_1^3)}{\partial w_i} &= \underbrace{\frac{\partial E(y_1^3)}{\partial y_1} \cdot \frac{\partial y_1}{\partial s_1} \cdot \frac{\partial s_1}{\partial w_i}}_{\text{red line}} + \underbrace{\frac{\partial E(y_1^3)}{\partial y_2} \cdot \frac{\partial y_2}{\partial s_2} \cdot \frac{\partial s_2}{\partial w_i}}_{\text{green line}} \\ &+ \underbrace{\frac{\partial E(y_1^3)}{\partial y_3} \cdot \frac{\partial y_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial w_i}}_{\text{blue line}} + \underbrace{\frac{\partial E(y_1^3)}{\partial y_4} \cdot \frac{\partial y_4}{\partial s_4} \cdot \frac{\partial s_4}{\partial w_i}}_{\text{yellow line}} \end{aligned}$$

# Introduction à l'Intelligence Artificielle

- Descente de gradient du gradient du neurone convolutif

X <sub>1</sub>	X <sub>2</sub>	0
W <sub>1</sub>	W <sub>2</sub>	
X <sub>4</sub>	X <sub>5</sub>	0
W <sub>3</sub>	W <sub>4</sub>	
0	0	0



- Prenons-en un seul :

$$\frac{\partial E(y_1^3)}{\partial y_1} \cdot \frac{\partial y_1}{\partial s_1} \cdot \frac{\partial s_1}{\partial w_i}$$

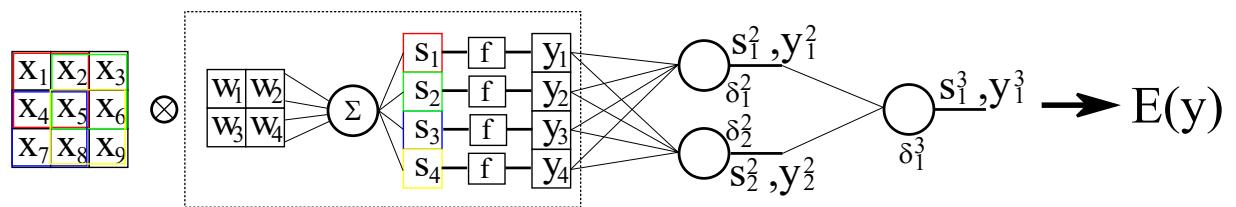
$$\sum_{k \in [1, n]} -\delta_k^2 \cdot w_{k1}^2 \cdot f'(s_1) \cdot \frac{\partial s_1}{\partial w_i}$$

$$\frac{\partial s_1}{\partial w_i} = \frac{\partial x_1 \cdot w_1 + x_2 \cdot w_2 + x_4 \cdot w_3 + x_5 \cdot w_4}{\partial w_i}$$

$$\begin{aligned} \frac{\partial s_1}{\partial w_1} &= x_1 & \frac{\partial s_1}{\partial w_2} &= x_2 \\ \frac{\partial s_1}{\partial w_3} &= x_4 & \frac{\partial s_1}{\partial w_4} &= x_5 \end{aligned}$$

# Introduction à l'Intelligence Artificielle

- Descente de gradient du gradient du neurone convolutif**



- À noter : il faut définir les liens entre poids w, position s et entrée x

x <sub>1</sub> w <sub>1</sub>	x <sub>2</sub> w <sub>2</sub>	0
x <sub>4</sub> w <sub>3</sub>	x <sub>5</sub> w <sub>4</sub>	0
0	0	0

0	0	0
x <sub>4</sub> w <sub>1</sub>	x <sub>5</sub> w <sub>2</sub>	0
x <sub>7</sub> w <sub>3</sub>	x <sub>8</sub> w <sub>4</sub>	0

$$\frac{\partial s_1}{\partial w_1} = x_1 \quad \frac{\partial s_1}{\partial w_2} = x_2 \quad \frac{\partial s_1}{\partial w_3} = x_4 \quad \frac{\partial s_1}{\partial w_4} = x_5$$

$$\frac{\partial s_2}{\partial w_1} = x_2 \quad \frac{\partial s_2}{\partial w_2} = x_3 \quad \frac{\partial s_2}{\partial w_3} = x_5 \quad \frac{\partial s_2}{\partial w_4} = x_6$$

$$\frac{\partial s_3}{\partial w_1} = x_4 \quad \frac{\partial s_3}{\partial w_2} = x_5 \quad \frac{\partial s_3}{\partial w_3} = x_7 \quad \frac{\partial s_3}{\partial w_4} = x_8$$

$$\frac{\partial s_4}{\partial w_1} = x_5 \quad \frac{\partial s_4}{\partial w_2} = x_6 \quad \frac{\partial s_4}{\partial w_3} = x_8 \quad \frac{\partial s_4}{\partial w_4} = x_9$$

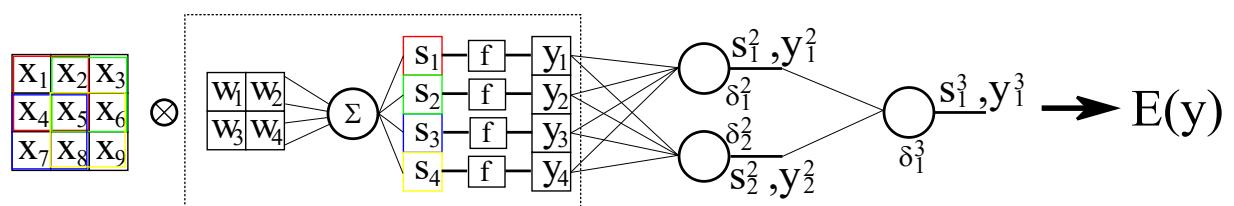
0	x <sub>2</sub> w <sub>1</sub>	x <sub>3</sub> w <sub>2</sub>
0	x <sub>5</sub> w <sub>3</sub>	x <sub>6</sub> w <sub>4</sub>
0	0	0

0	0	0
0	x <sub>5</sub> w <sub>1</sub>	x <sub>6</sub> w <sub>2</sub>
0	x <sub>8</sub> w <sub>3</sub>	x <sub>9</sub> w <sub>4</sub>

# Introduction à l'Intelligence Artificielle

- Descente de gradient du gradient du neurone convolutif**

$x_1$	$x_2$	$x_3$
$w_1$	$w_1$	
$x_4$	$x_5$	$x_6$
$w_1$	$w_1$	$x_6$
$x_7$	$x_8$	$x_9$



- On remplace dans la formule :

$$\frac{\partial E(y_1^3)}{\partial w_1} = - \underbrace{x_1 \cdot f'(s_1) \cdot \sum_{k \in [1, n]} \delta_k^2 \cdot w_{k1}^2}_{\text{Red line}} - \underbrace{x_2 \cdot f'(s_2) \cdot \sum_{k \in [1, n]} \delta_k^2 \cdot w_{k2}^2}_{\text{Green line}}$$

$$- \underbrace{x_4 \cdot f'(s_3) \cdot \sum_{k \in [1, n]} \delta_k^2 \cdot w_{k3}^2}_{\text{Blue line}} - \underbrace{x_5 \cdot f'(s_4) \cdot \sum_{k \in [1, n]} \delta_k^2 \cdot w_{k4}^2}_{\text{Yellow line}}$$

$$\frac{\partial E(y_1^3)}{\partial w_1} = -(\underbrace{x_1 \cdot \delta_1^1}_{\text{Red line}} + \underbrace{x_2 \cdot \delta_2^1}_{\text{Green line}} + \underbrace{x_4 \cdot \delta_3^1}_{\text{Blue line}} + \underbrace{x_5 \cdot \delta_4^1}_{\text{Yellow line}})$$

Autant de termes que de pixels dans l'image de sortie

# Introduction à l'Intelligence Artificielle

- Descente de gradient du gradient du neurone convolutif**

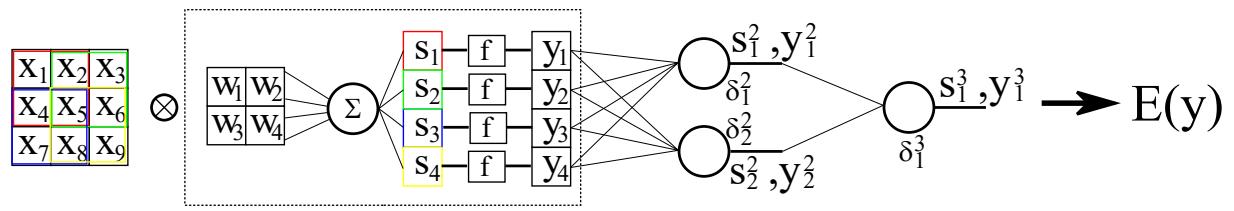
$$\begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline x_1 & x_2 \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline x_2 & x_3 \\ \hline \end{array}$$
  

$$\begin{array}{|c|c|} \hline w_3 & w_4 \\ \hline x_4 & x_5 \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline w_3 & w_4 \\ \hline x_5 & x_6 \\ \hline \end{array}$$
  

$$\begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline x_4 & x_5 \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline x_5 & x_6 \\ \hline \end{array}$$
  

$$\begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline x_4 & x_5 \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline w_1 & w_2 \\ \hline x_5 & x_6 \\ \hline \end{array}$$
  

$$\begin{array}{|c|c|} \hline w_3 & w_4 \\ \hline x_7 & x_8 \\ \hline \end{array} \quad \begin{array}{|c|c|} \hline w_3 & w_4 \\ \hline x_8 & x_9 \\ \hline \end{array}$$



- Dernière étape pour le backpropagation de la couche précédente :

$$\text{Calculer } \frac{\partial E(y_1^3)}{\partial x_i}$$

$$\frac{\partial x_1 \cdot w_1 + x_2 \cdot w_2 + x_4 \cdot w_3 + x_5 \cdot w_4}{\partial x_5}$$

$$\frac{\partial E(y_1^3)}{\partial x_5} = \frac{\partial E(y_1^3)}{\partial s_1} \times \frac{\partial s_1}{\partial x_5} + \frac{\partial E(y_1^3)}{\partial s_2} \times \frac{\partial s_2}{\partial x_5} + \frac{\partial E(y_1^3)}{\partial s_3} \times \frac{\partial s_3}{\partial x_5} + \frac{\partial E(y_1^3)}{\partial s_4} \times \frac{\partial s_4}{\partial x_5}$$

$$\frac{\partial E(y_1^3)}{\partial x_5} = - ( w_4 \cdot \delta_1^1 + w_3 \cdot \delta_2^1 + w_2 \cdot \delta_3^1 + w_1 \cdot \delta_4^1 )$$

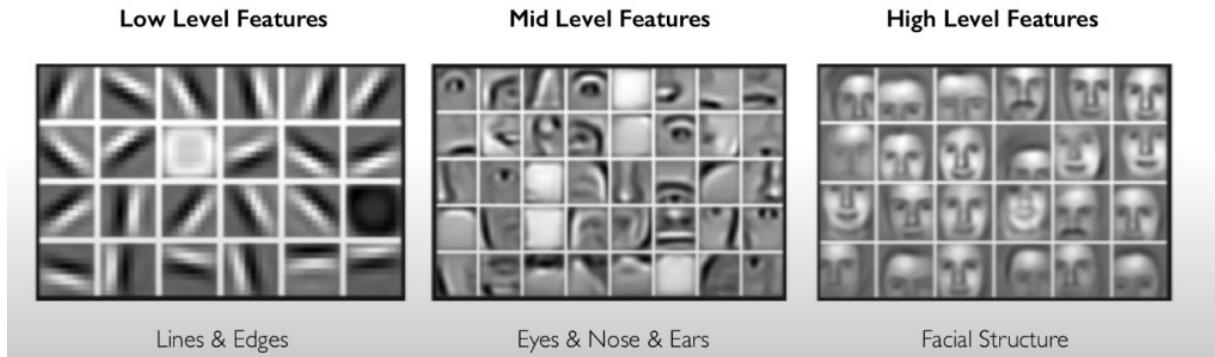
Autant de termes que de pixels dans le noyau qui passeront sur  $x_i$

# Apprentissage des réseaux profonds : la question du dataset

# Introduction à l'Intelligence Artificielle

- **Apprentissage des réseaux profonds**

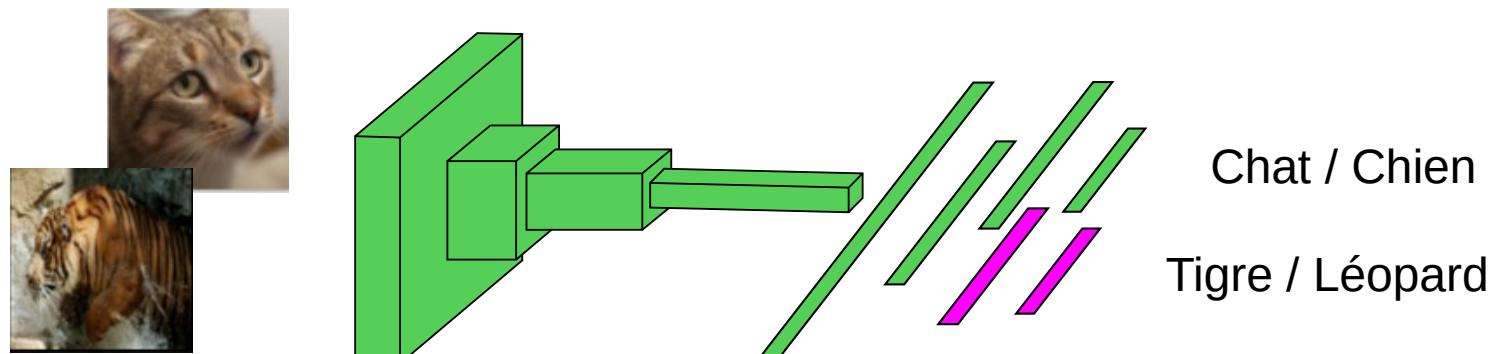
- Plus il y a de couches de neurones, plus le dataset d'apprentissage doit être important.
  - Pour un réseau deep learning, il est parfois nécessaire d'avoir un dataset de plusieurs dizaines de millier, voire centaines de millier d'exemples.
- Doit-on avoir une grande base d'images pour chaque problème ?
- Pour deux problèmes similaires, les couches de convolution génèrent des features très similaires
  - Et d'autant plus dans les couches les plus basses !



# Introduction à l'Intelligence Artificielle

- **Apprentissage des réseaux profonds**

- Sur des problèmes similaires, on peut réutiliser les premières couches d'un réseau pour reconnaître des éléments similaires
  - c'est le **transfer learning**
- On utilise le réseau entraîné sur un problème similaire (ou sur un grand nombre de classes)
- On remplace seulement les dernières couches du réseau

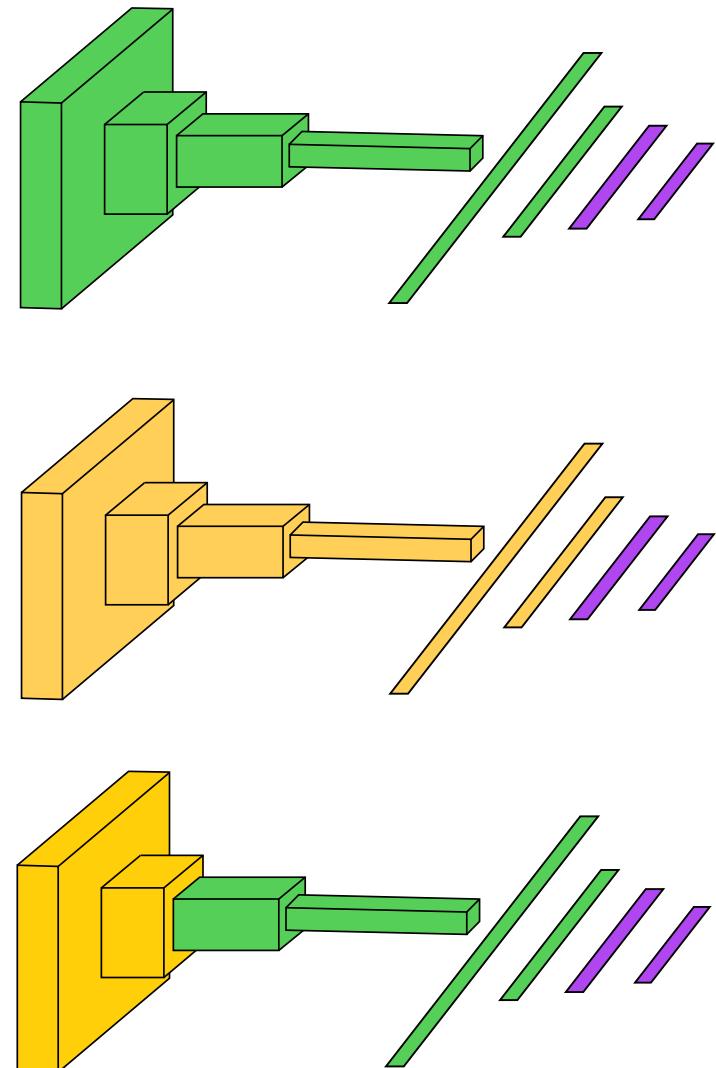


- Plusieurs possibilités qui dépendent du nombre d'exemples disponibles

# Introduction à l'Intelligence Artificielle

## • Le transfer Learning

- Fine tuning total : on remplace la ou les dernières couches de la partie fully-connected par un nouveau réseau initialisé aléatoirement.
  - Apprentissage plus rapide et nécessitant moins d'exemples
  - Les features de bas niveau peuvent s'adapter au nouveau problème
  - Nécessite toujours un dataset important
- Extraction de features : on remplace la ou les dernières couches et on fixe les poids des couches précédentes
  - Apprentissage avec peu d'exemples
  - Résultat pas forcément optimal
  - Peut se contenter d'un dataset réduit
- Fine tuning partiel : on remplace la ou les dernières couches, et on fixe certaines couches du réseau
  - Bon compromis entre les deux précédentes méthodes
  - Features de bas niveau plus 'universelles'



# Introduction à l'Intelligence Artificielle

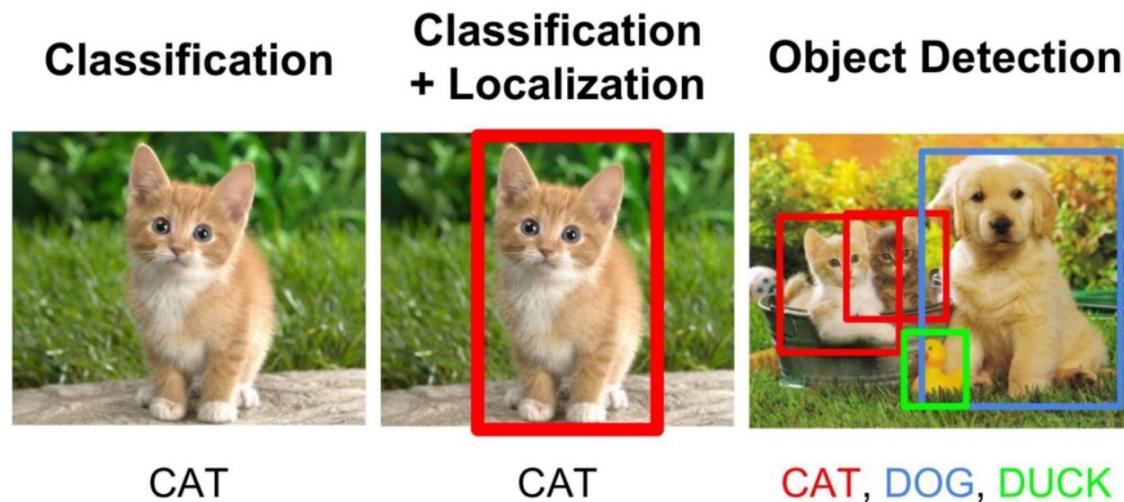
- **Le transfert Learning**

- Principales architectures permettant le transfert learning
  - AlexNet : architecture utilisée lors du ILSVRC 2012
  - VGGnet : une des architectures les plus versatiles. Utilise 16 couches de convolution
  - GoogleNet : vainqueur du ILSVRC 2014, utilise 22 couches de convolution
  - ResNet : vainqueur du ILSVRC 2015, avec un score d'erreur de 3,57 % (plus performant que des participants humains), utilise 152 couches de convolution

# Introduction à l'Intelligence Artificielle

- **Applications**

- Reconnaissance d'images, de son, de texte



- Classification, traduction de texte

→ dans tous les cas, on cherche à générer un vecteur de features

# Introduction à l'Intelligence Artificielle

- **Applications**

- Question : peut-on reconstruire l'image d'origine avec le vecteur de feature ?
  - Les convolutions successives génèrent une perte d'information
  - Le vecteur de features contient très peu d'informations par rapport à l'image d'origine
  - Couches pour renverser le processus d'encodage de l'image
    - Couches fully-connected (petites images)
    - Couches de De-pooling
    - Couches de convolution transposée

# Introduction à l'Intelligence Artificielle

- **Reconstruction de l'image**

- De-Pooling :

- Nearest neighbor

0,29	0,47	0,46
0,15	0	0
0	0,57	1,49

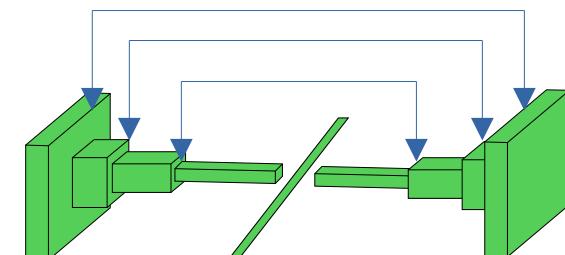
0,29	0,29	0,47	0,47	0,46	0,46
0,29	0,29	0,47	0,47	0,46	0,46
0,15	0,15	0	0	0	0
0,15	0,15	0	0	0	0
0	0	0,57	0,57	1,49	1,49
0	0	0,57	0,57	1,49	1,49

- Bed of nails

0,29	0	0,47	0	0,46	0
0	0	0	0	0	0
0,15	0	0	0	0	0
0	0	0	0	0	0
0	0	0,57	0	1,49	0
0	0	0	0	0	0

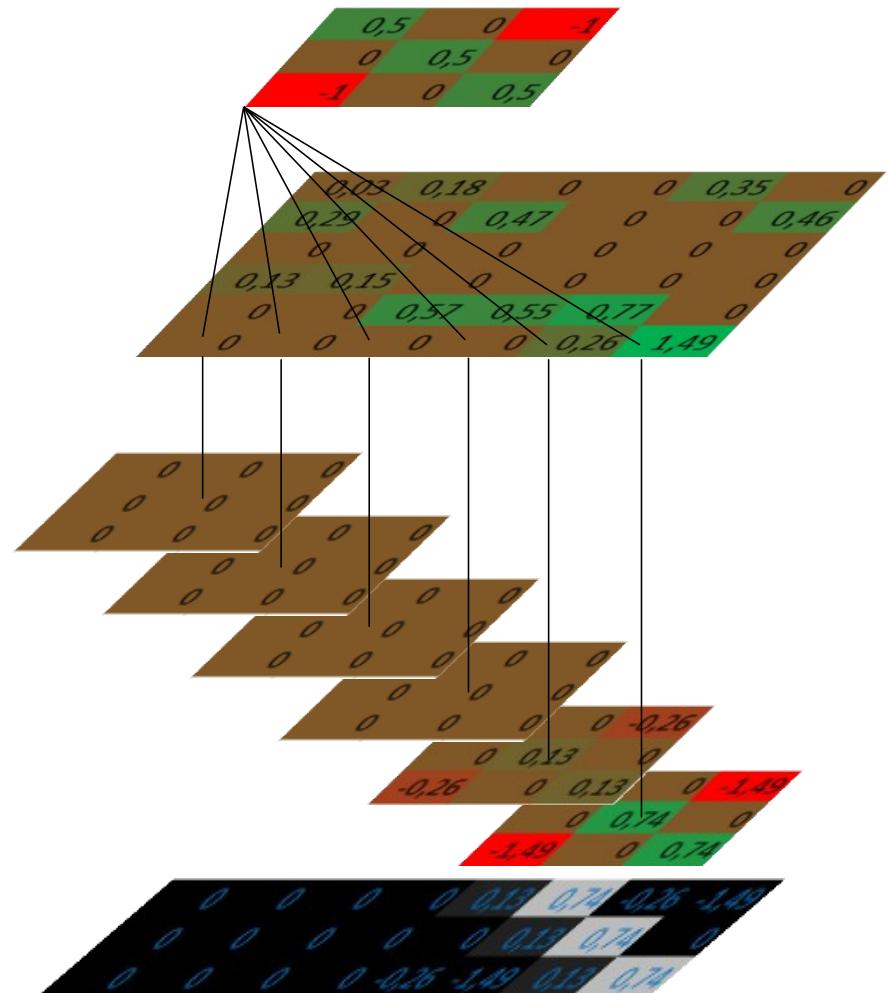
- Max UnPooling

→ on se sert de la couche de pooling 'correspondante' pour savoir quel pixel prend la valeur max



# Introduction à l'Intelligence Artificielle

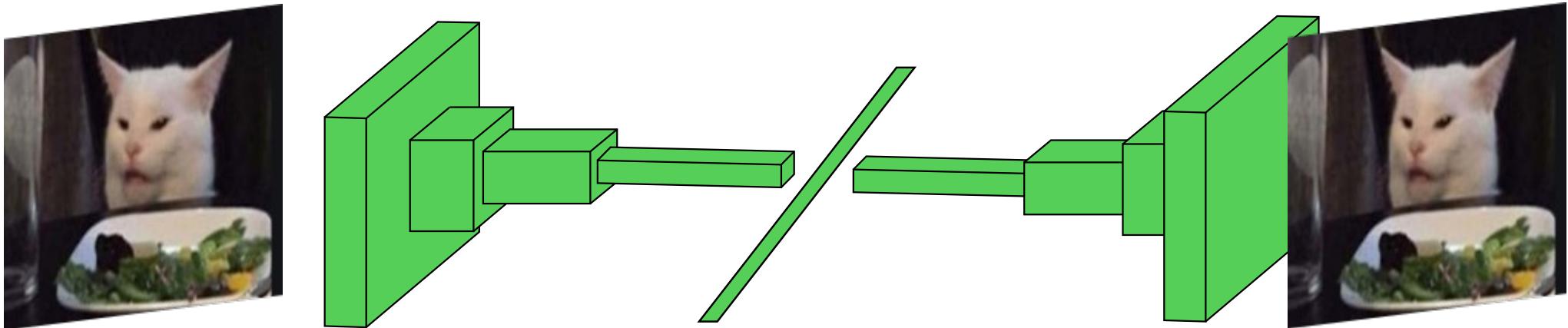
- **Reconstruction de l'image**
  - Convolution transposée :
    - On applique le principe ‘inverse’ de la convolution
    - Sur l'image de sortie, on ajoute les valeurs du kernel pondérées par la valeur du pixel de l'image d'entrée
      - Chaque pixel de l'image de sortie est incrémenté plusieurs fois



# Introduction à l'Intelligence Artificielle

- **Reconstruction de l'image**

- On remplace la partie fully-connected par une version « miroir » du réseau

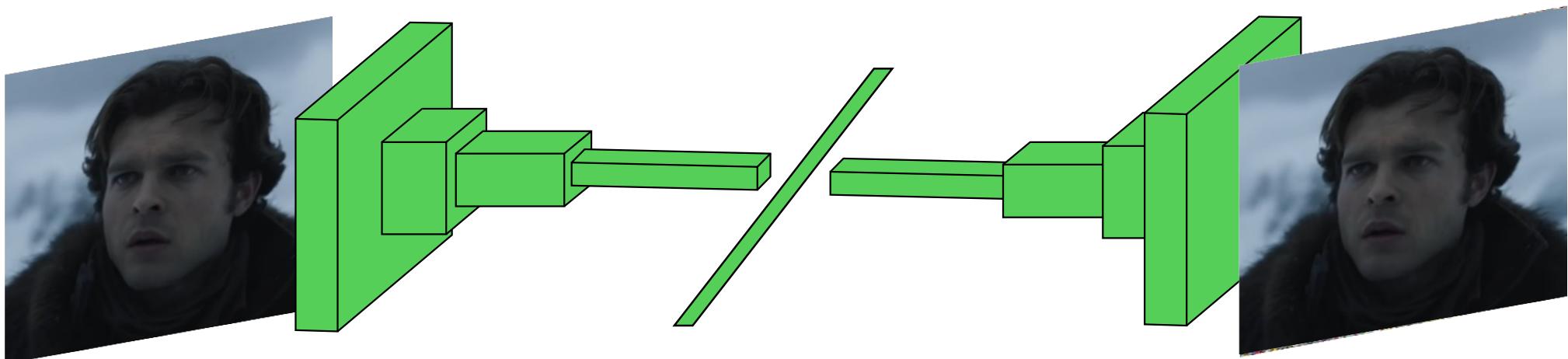


- L'image d'entrée sert comme résultat attendu ( $E = | \text{Img}_{\text{ref}} - \text{Img}_{\text{gen}} |^2$ )
  - La première partie encode l'information : réseau 'encodeur'
  - La seconde partie 'décode' le vecteur pour reconstituer l'image : réseau 'décodeur'
    - Génération des textures, contours...

# Introduction à l'Intelligence Artificielle

- **Reconstruction de l'image**

- Le réseau 'apprend' à définir les informations minimales
- Entraînons le réseau sur un visage particulier

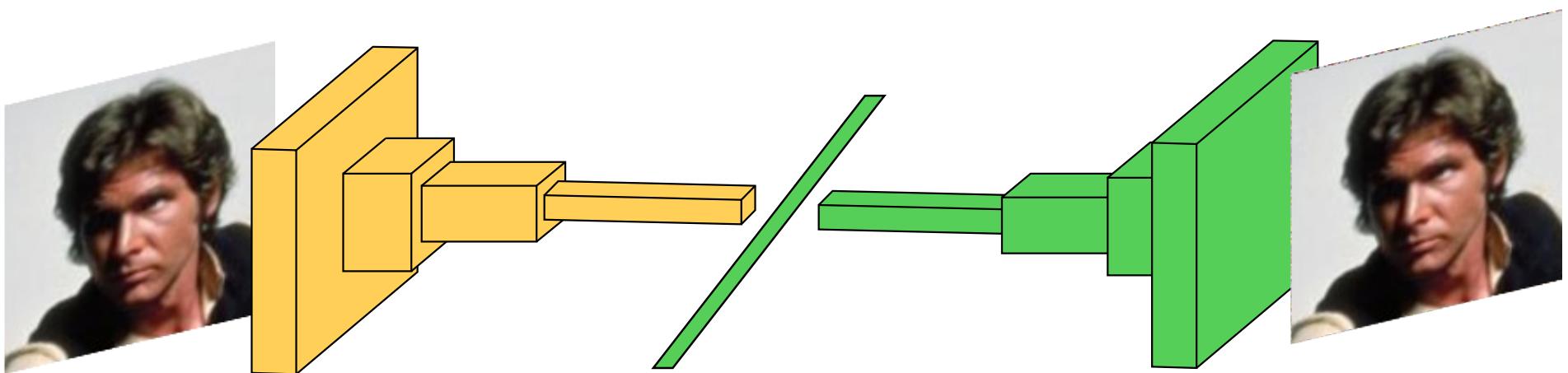


- Le réseau va encoder les informations minimales pour reconstruire l'image
  - Orientation et taille de la tête, expression faciale... (de façon implicite)
  - Le réseau décodeur va recréer le visage à partir de ces informations

# Introduction à l'Intelligence Artificielle

- **Reconstruction de l'image**

- On fige les poids du réseau encodeur
- On réinitialise le réseau décodeur et on l'entraîne avec un autre visage

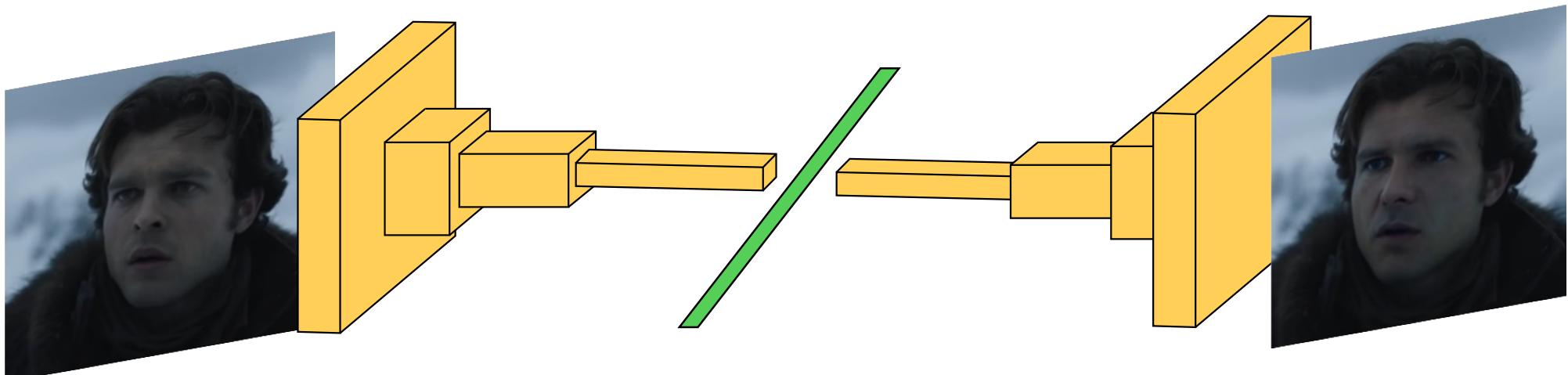


- Le réseau décodeur va fournir un vecteur de feature de façon très similaire
- Le réseau décodeur va devoir apprendre à générer ce nouveau visage

# Introduction à l'Intelligence Artificielle

- **Reconstruction de l'image**

- On fige les poids du réseau décodeur
- On présente des images du premier visage
  - Le réseau encodeur extrait les informations nécessaires à la reconstruction
  - Le réseau décodeur va reconstruire une version du second visage à partir de ces informations



- C'est le principe du deep fake

# Introduction à l'Intelligence Artificielle

- Reconstruction de l'image

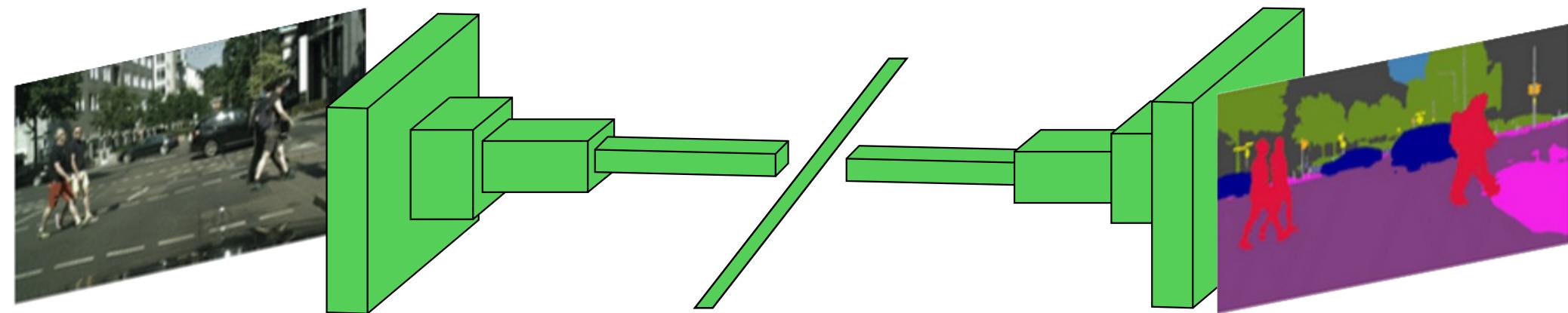


Harrison Ford in Solo: A Star Wars Story (Shamook)

# Introduction à l'Intelligence Artificielle

- **Réseau fully-convolutional**

- On peut générer des images d'un type différent de l'image d'entrée
- Exemple : Mask-RCNN
  - On présente en entrée des images réelles, et en sortie, des images avec des masques définis à la main

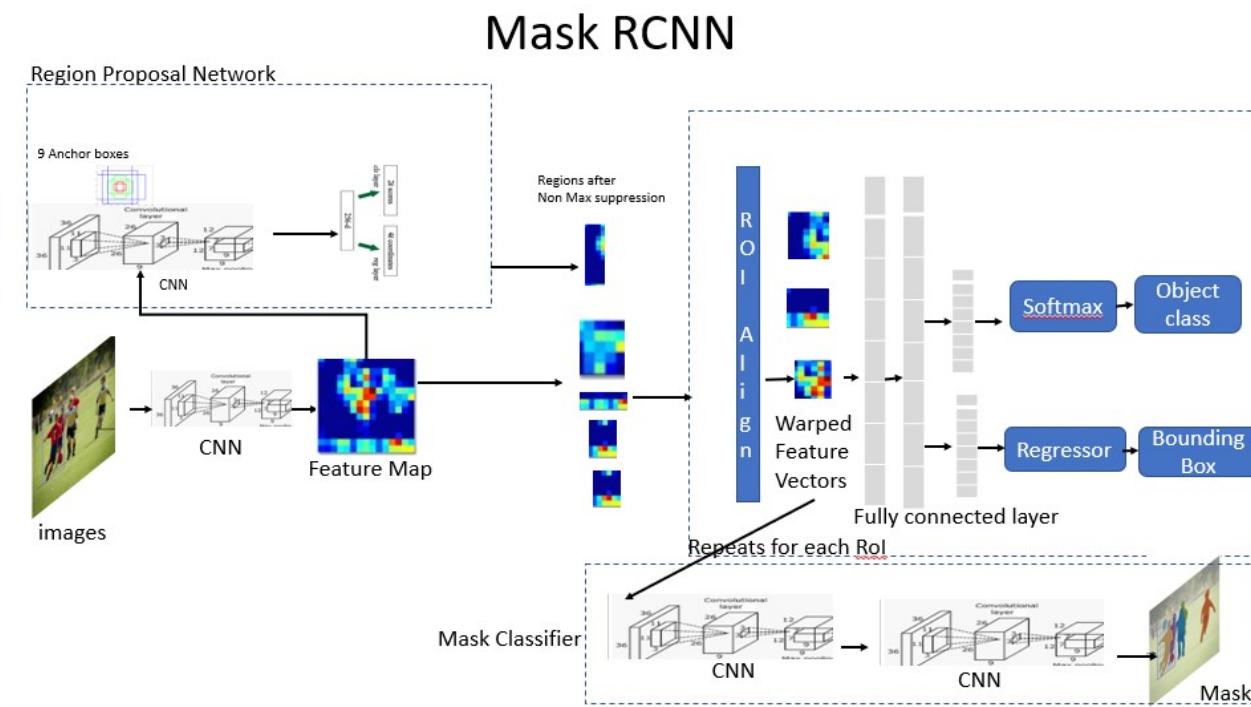


# Introduction à l'Intelligence Artificielle

- **Réseau fully-convolutional**

- Mask-RCNN

- Détection et localisation des éléments dans l'image
      - extraction des 'vignettes' contenant les éléments
    - Détection pixel par pixel pour former un masque de l'objet



# Introduction à l'Intelligence Artificielle

- **Réseau fully-convolutional**
  - Mask-RCNN, dataset COCO (Common Objects in COnText)



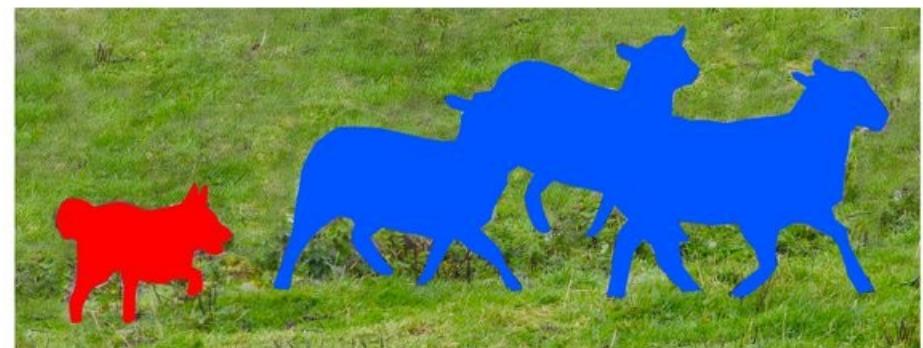
# Introduction à l'Intelligence Artificielle

- Réseau fully-convolutional

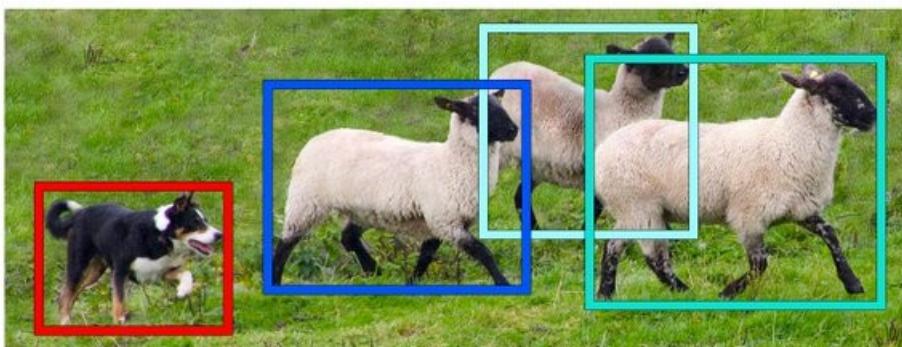
- Segmentation de la scène



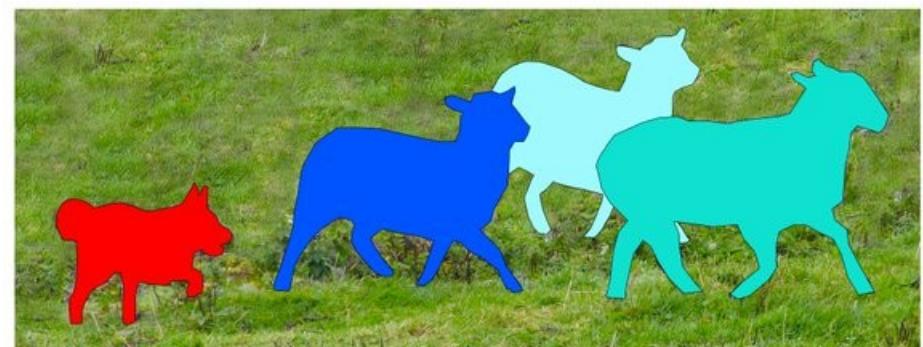
Image Recognition



Semantic Segmentation



Object Detection



Instance Segmentation

# Les données d'apprentissage : biais et sécurité

# Introduction à l'Intelligence Artificielle

- **Les données d'apprentissage**

- Pour entraîner des réseaux aussi immense, il faut de très grandes quantités d'exemples
  - Difficulté de constituer le dataset d'apprentissage
  - Des biais statistiques peuvent émerger dans les données collectées
    - Ces biais peuvent avoir des incidences (parfois grave) sur les résultats
  - Les réseaux de neurones sont tous très sensibles aux biais statistiques, car utilisent les régularités qui peuvent apparaître des les datasets
    - Une IA qui détermine l'age d'une personne va se focaliser sur la taille de ses oreilles

# Introduction à l'Intelligence Artificielle

- **Les données d'apprentissage**

- Exemple de biais :
  - Réseau pour différencier un loup et un husky
  - Très grand taux de réussite
  - Mais quelques erreurs
- Après analyse des features, le réseau se concentre sur la neige !
  - Dans le dataset, une large majorité des photos de loup sont dans un paysage enneigé



Predicted: **wolf**  
True: **wolf**



Predicted: **husky**  
True: **husky**



Predicted: **wolf**  
True: **wolf**



Predicted: **husky**  
True: **husky**



Predicted: **wolf**  
True: **wolf**



Predicted: **wolf**  
True: **husky**

# Introduction à l'Intelligence Artificielle

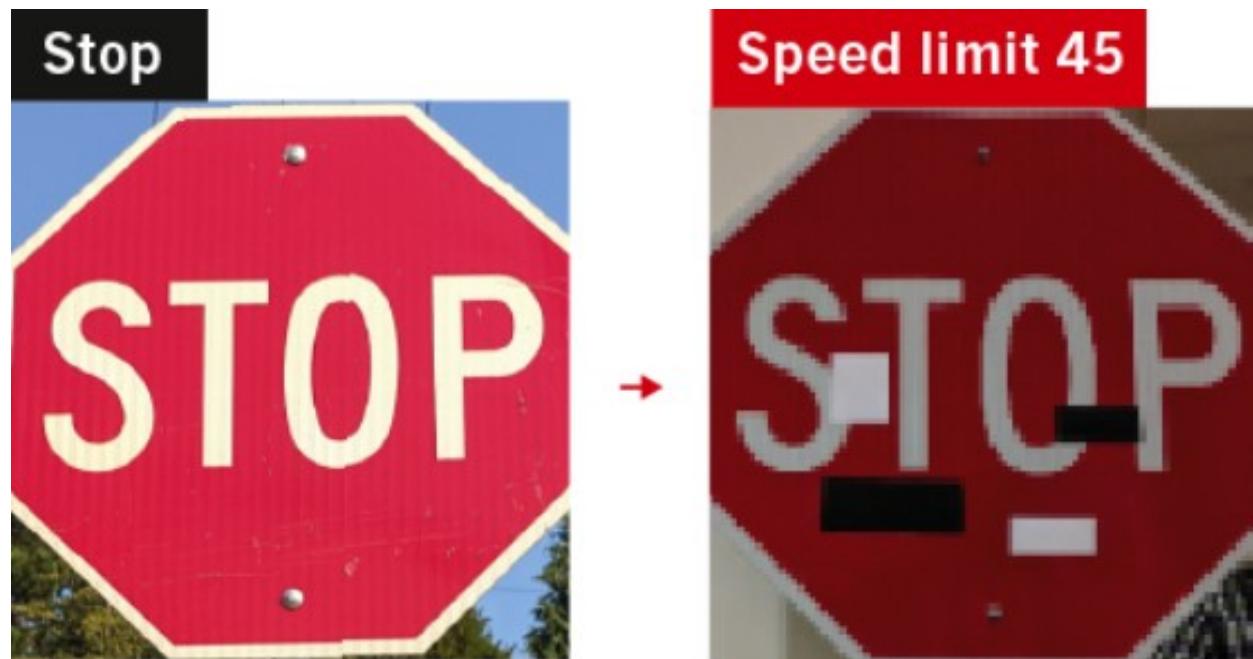
- **Les données d'apprentissage**

- De nombreux types de biais peuvent apparaître :
  - Le biais des données : biais dans la distribution des données en fonction de la catégorie
    - le réseau peut utiliser ces biais pour augmenter son taux de réussite en augmentant la tolérance des classes les moins représentées
  - biais des stéréotypes : biais dans la constitution du dataset liés à sa culture, son milieu social...
  - Biais des variables omises : biais statistiques sur des paramètres auxquels on n'a pas pensé
  - Biais de sélection : biais (plus ou moins volontaire) dans la sélection des données pour montrer un résultat
  - Biais économique et technologique : les choix d'architectures, les heuristiques destinées à simplifier ou réduire les coûts

# Introduction à l'Intelligence Artificielle

- Sécurité

- Entraîner un réseau sur un dataset biaisé ou insuffisant ne réduit pas seulement la fiabilité des résultats, mais rend aussi le réseau vulnérable aux attaques !



# Introduction à l'Intelligence Artificielle

- **Les données d'apprentissage**

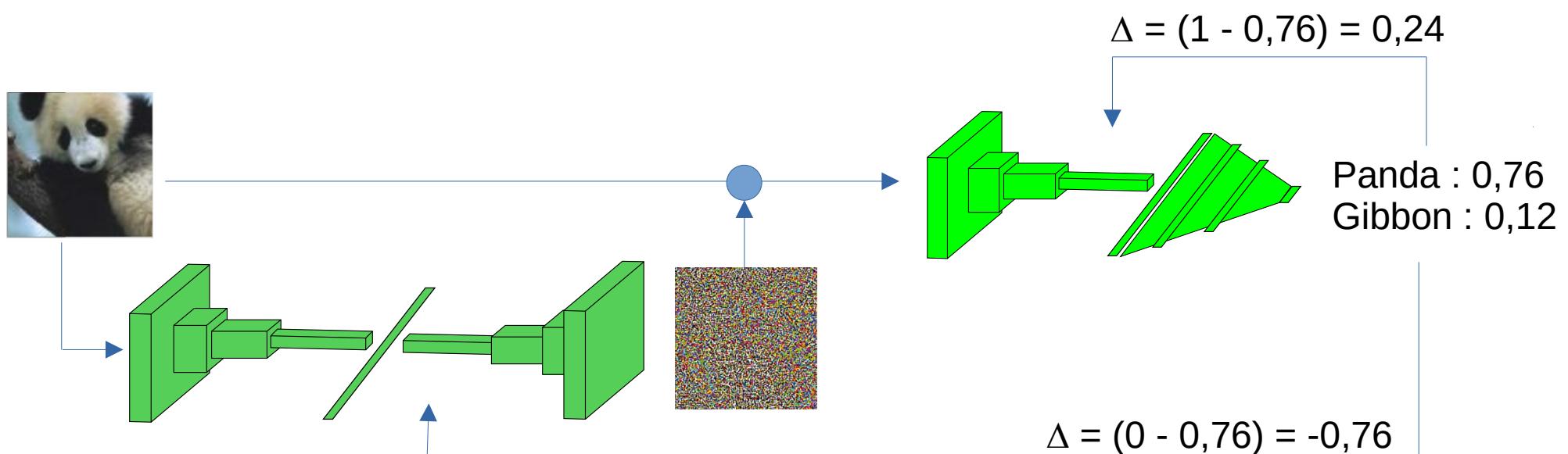


- L'ajout du bruit perturbe la détection des features de bas niveau, qui vont perturber en cascade la détection de features de plus haut niveau
- Comment ces perturbations ont été générées ?

# Introduction à l'Intelligence Artificielle

- **Les données d'apprentissage**

- Un réseau profond ‘normal’ détecte et reconnaît les images
- On ajoute un réseau encodeur-décodeur pour générer du bruit qui s’ajoute à l'image
- Le succès du second réseau est lié à l’erreur du second réseau



- Les deux réseaux sont en concurrence !

# Les GANS

# Introduction à l'Intelligence Artificielle

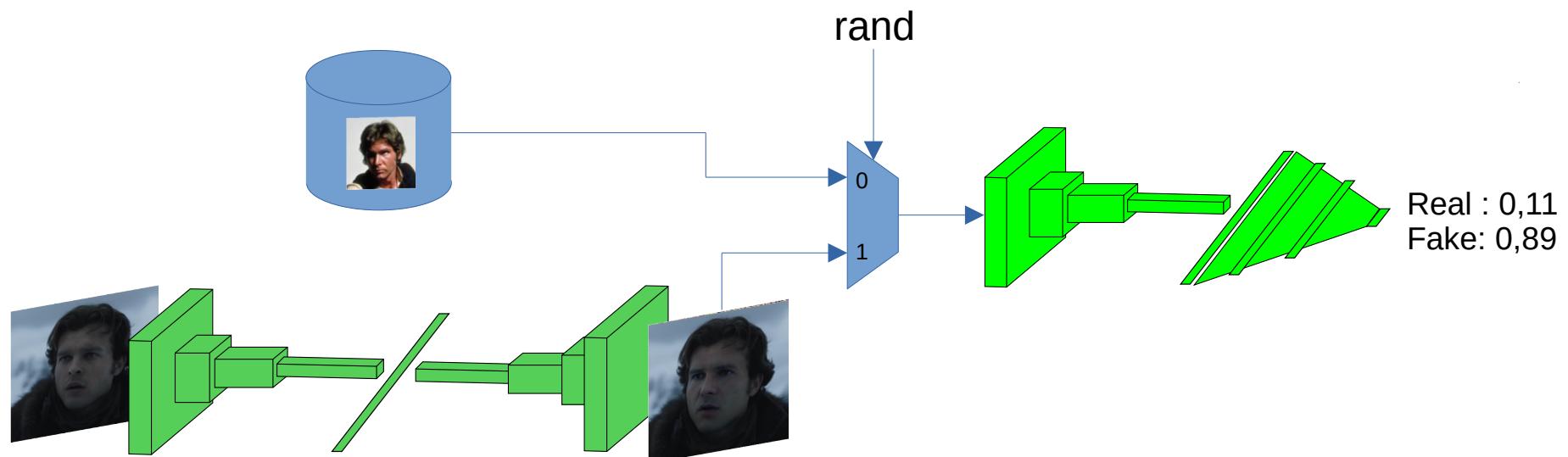
- **Les GANs**

- Réseaux antagonistes génératifs (Generative Adversarial Networks)
- Réseaux fonctionnant par paire
  - Le premier réseau cherche à tromper le second
  - Le second cherche à déterminer si son image d'entrée est réelle ou non
- Les fonctions de coût sont opposées : le succès d'un réseau implique l'échec de l'autre : amélioration mutuelle
  - Le premier réseau s'améliore pour générer des images ‘crédibles’
  - Le second s'améliore pour détecter les fausses images
- Plusieurs architectures pour différents usages

# Introduction à l'Intelligence Artificielle

- **Les GANs**

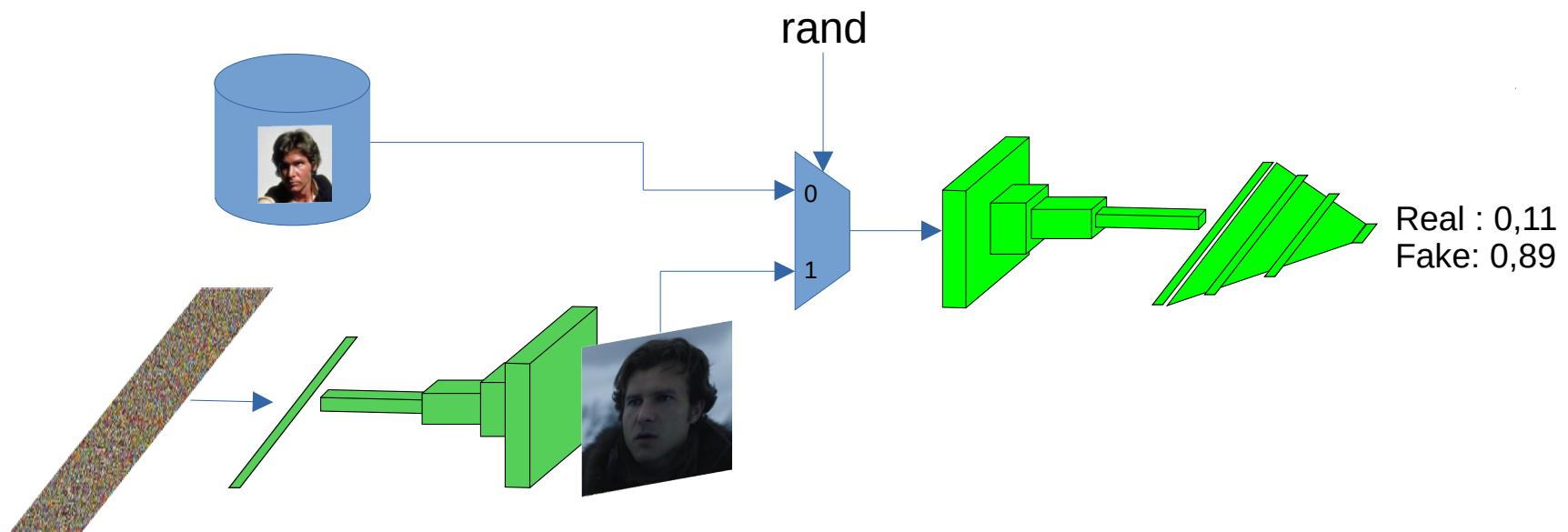
- Des GANS pour améliorer les résultats d'un réseau
  - Le réseau générateur que l'on veut améliorer (pas forcément convolutif)
  - Une base de donnée d'images réelles (ou du type que l'on veut obtenir)
  - Un réseau discriminateur qui apprend à distinguer vraies et fausses images



# Introduction à l'Intelligence Artificielle

- **Les GANs**

- Des GANS peuvent générer des images (ou du son, du texte...) inédites
  - Un réseau uniquement décodeur (pas forcément convolutif)
  - Un générateur de bruit aléatoire
  - Une base de donnée d'images réelles (ou du type que l'on veut obtenir)
  - Un réseau discriminateur qui apprend à distinguer vraies et fausses images



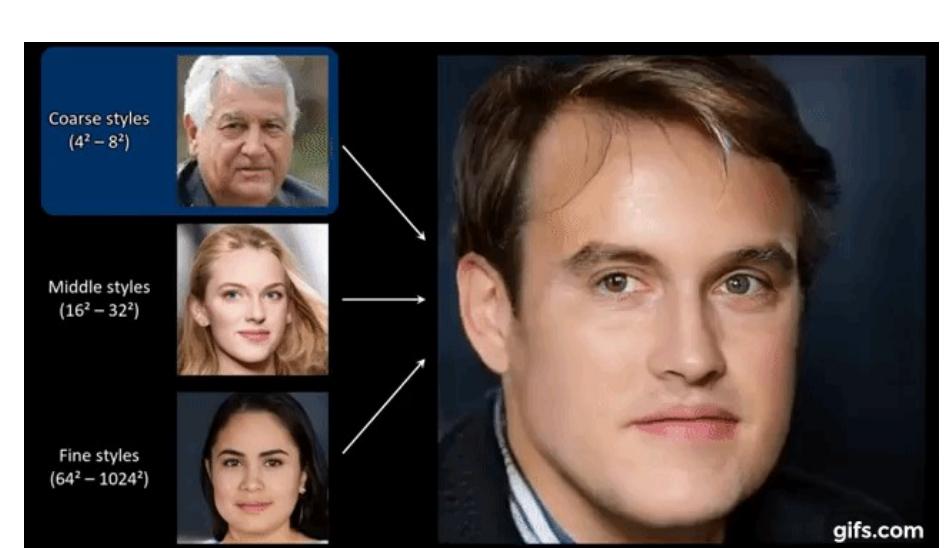
# Introduction à l'Intelligence Artificielle

- **Les GANs**

- GANs génératifs : l'espace latent (latent space)
  - Une fois entraîné, on peut récupérer des visages représentatifs, avec leur vecteur de feature, ou générer ces vecteurs avec un réseau encodeur-décodeur



Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks



gifs.com

- On peut ensuite faire varier ce vecteur d'un visage à un autre pour interpoler
- Ou ajouter et soustraire ces vecteurs pour modifier certains paramètres

# Introduction à l'Intelligence Artificielle

- **Les GANs**

- GANs génératifs : une évolution rapide

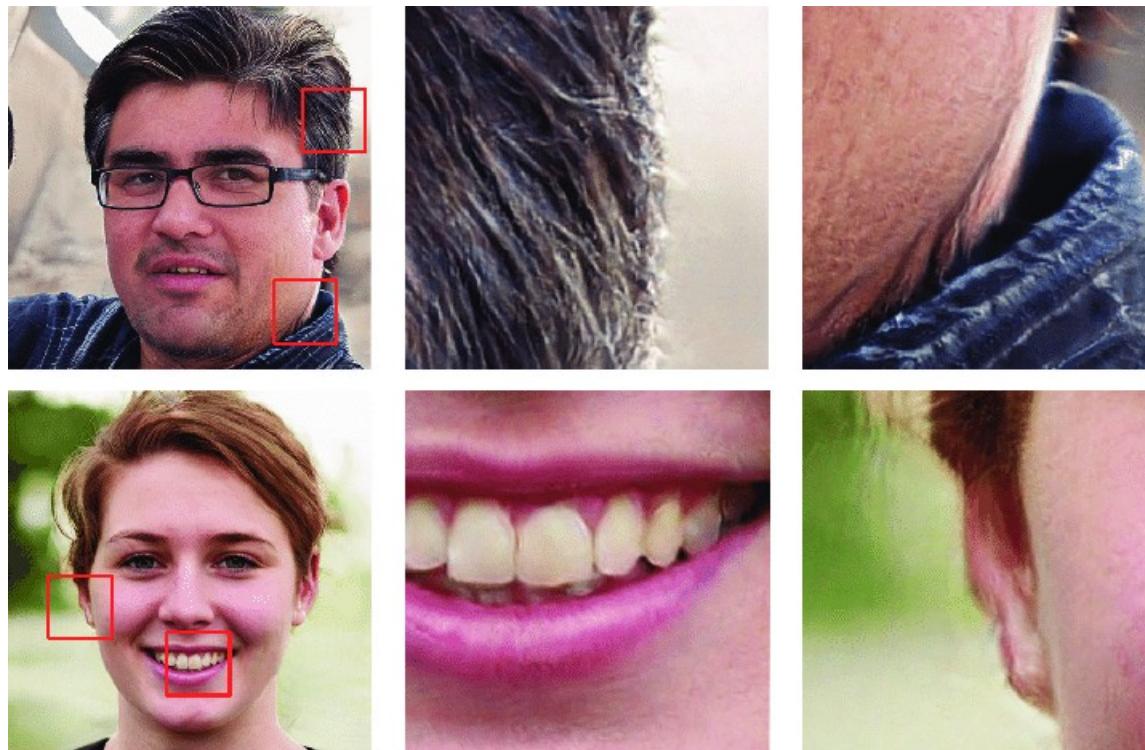


Generative Adversarial Networks (GANs): An Overview of Theoretical Model, Evaluation Metrics, and Recent Developments Preprint, Pegah et al., 2020

# Introduction à l'Intelligence Artificielle

- **Les GANs**

- GANs génératifs : Pas toujours parfait...

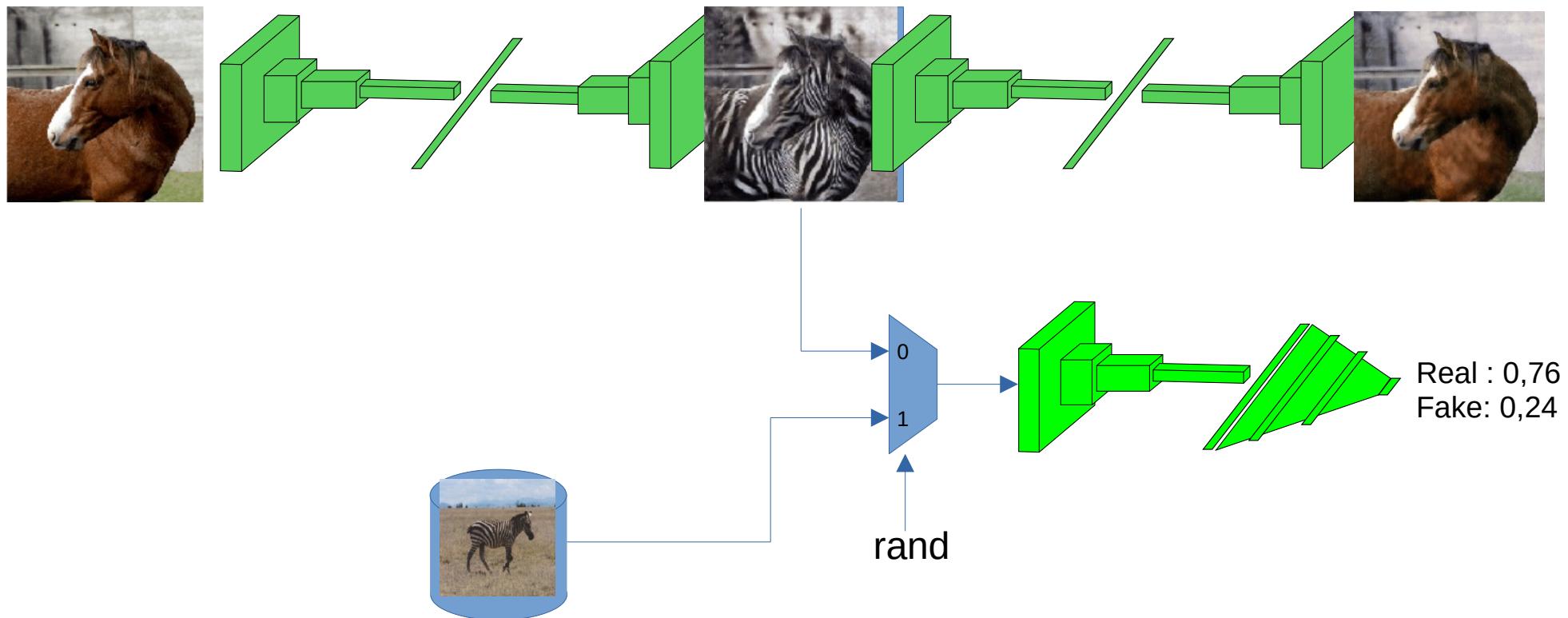


No One Can Escape: A General Approach to Detect Tampered and Generated Image,  
Zhang et al., 2019

# Introduction à l'Intelligence Artificielle

- Les GANs ‘améliorés’

- Cycle GANs :



# Introduction à l'Intelligence Artificielle

- **Les GANs ‘améliorés’**
  - Cycle GANs :

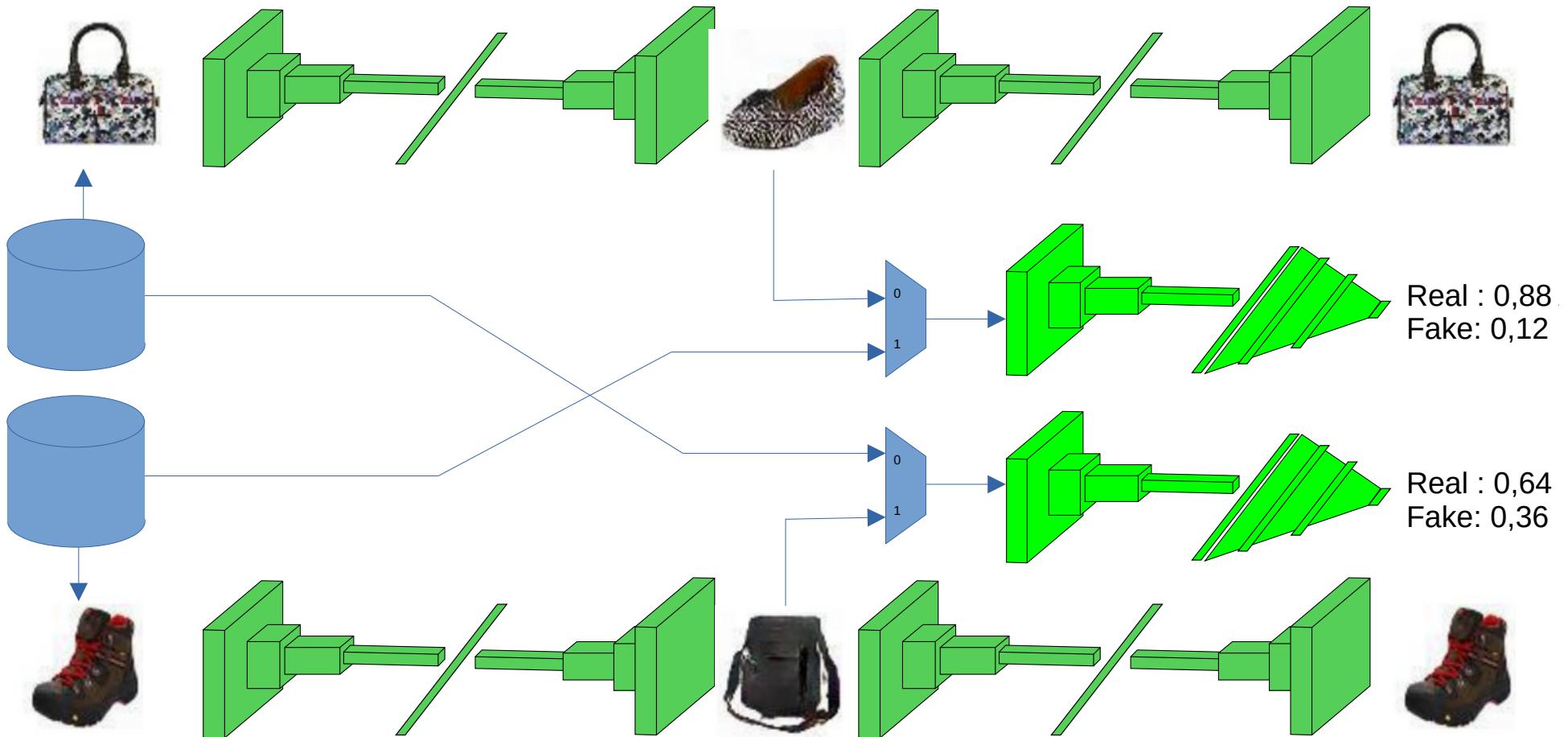


<https://github.com/junyanz/CycleGAN>

# Introduction à l'Intelligence Artificielle

- **Les GANs ‘améliorés’**

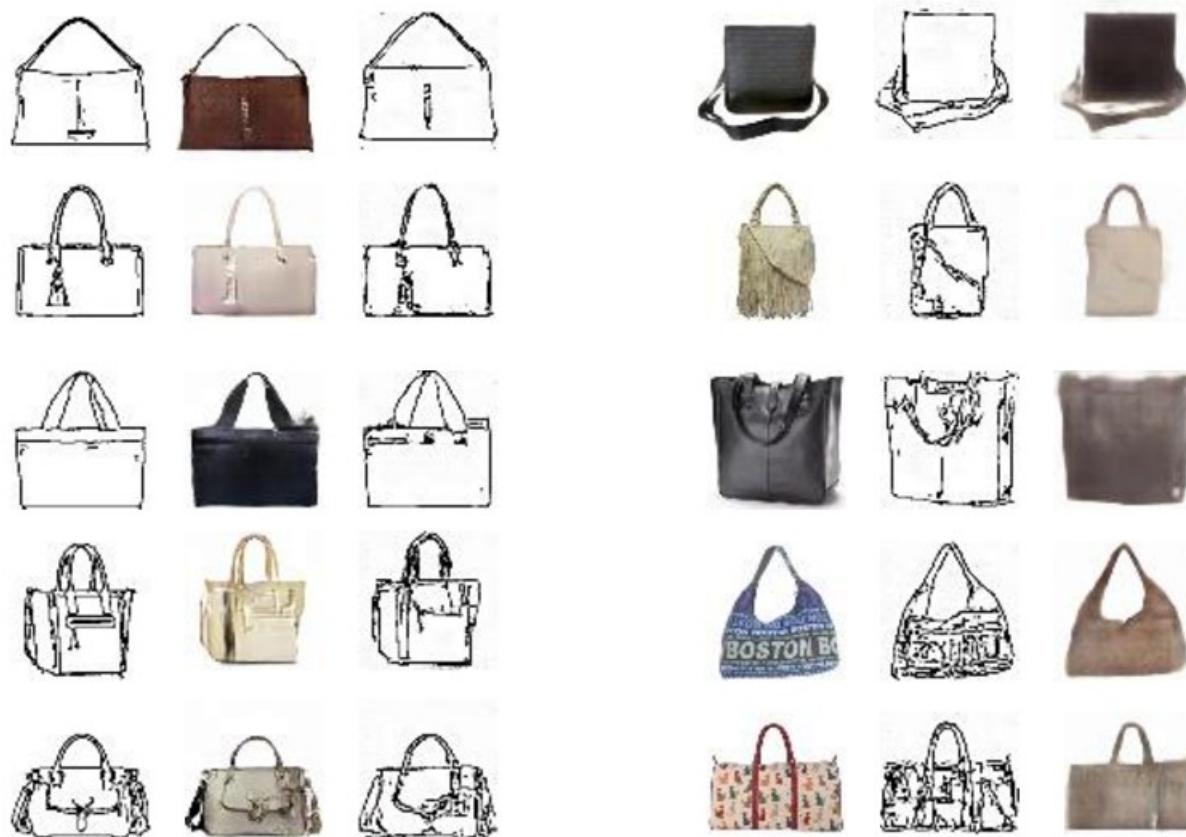
- Disco GANs (Discover Cross-Domain Relations GAN) :



# Introduction à l'Intelligence Artificielle

- **Les GANs ‘améliorés’**

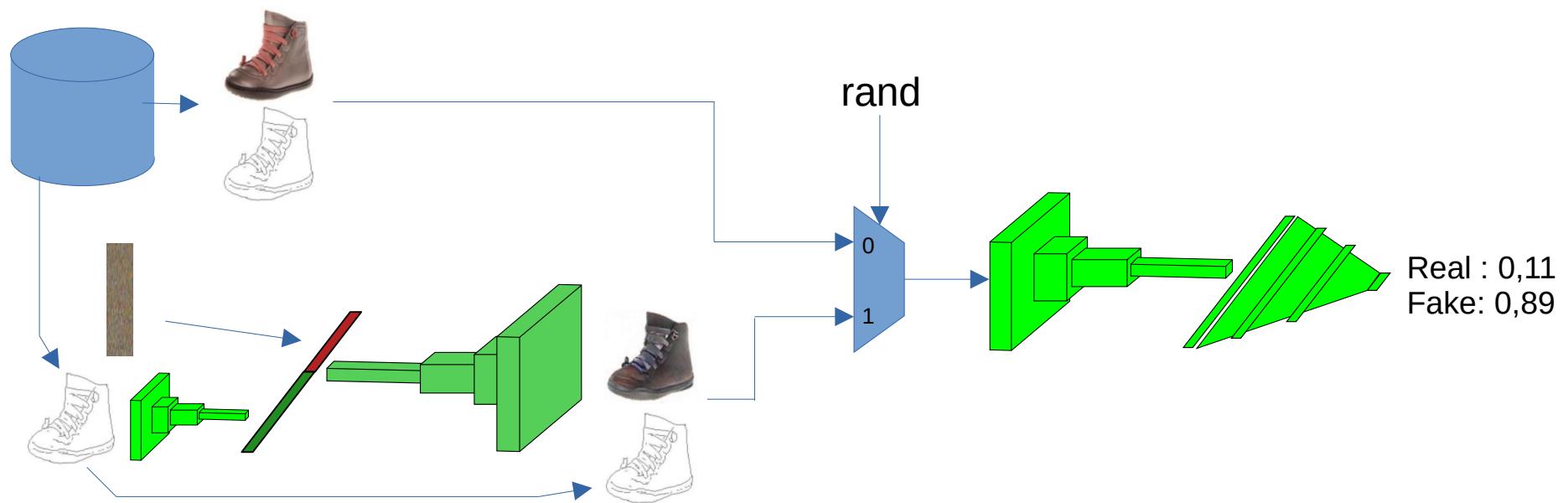
- Disco GANs (Discover Cross-Domain Relations GAN) :



# Introduction à l'Intelligence Artificielle

- **Les GANs ‘améliorés’**

- Conditional GANs : ajout de contraintes sur l'image à générer
  - On ajoute aux images d'entraînement des informations supplémentaires (ex. dessin d'une photo, valeurs numériques...)
  - Le discriminateur intègre la différence image/information
  - Le GAN doit tenir compte de ces informations quand il génère l'image

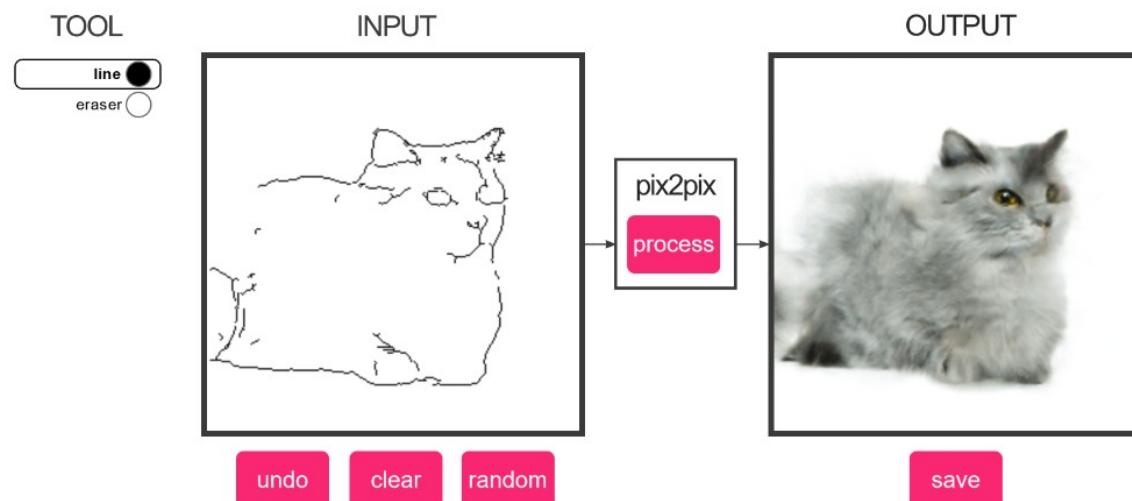
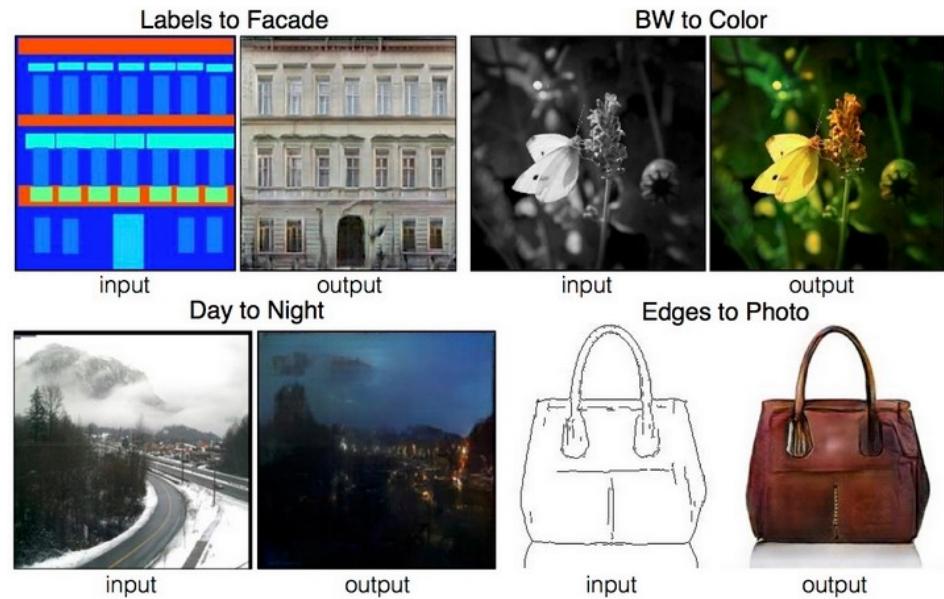


# Introduction à l'Intelligence Artificielle

- **Les GANs ‘améliorés’**

- Conditional GANs :
  - Pix2pix
  - Edges2cats
  - Edges2shoes
  - Edges2handbag

edges2cats

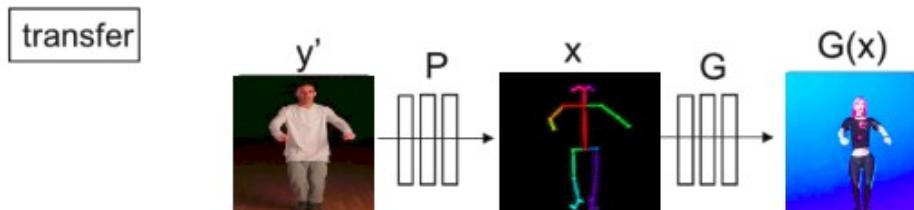
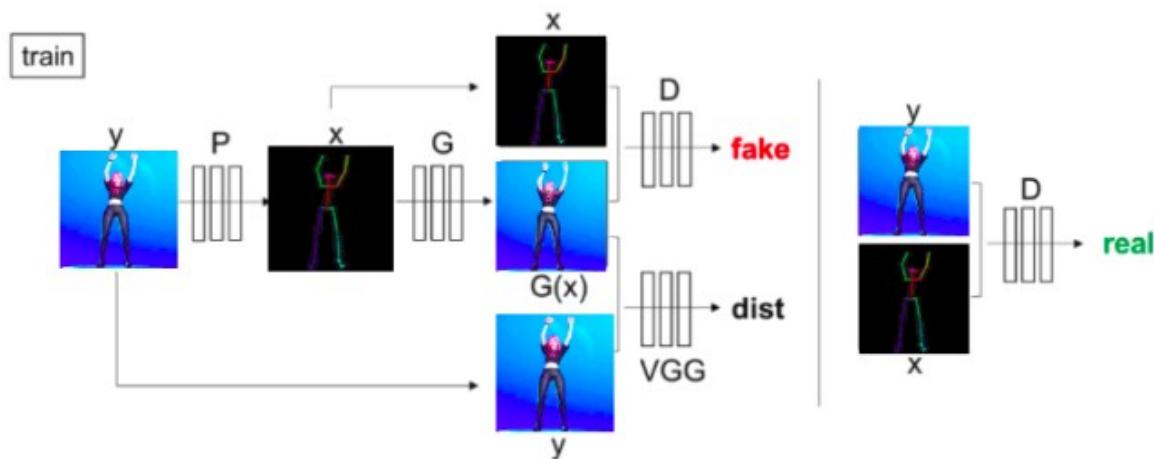


# Introduction à l'Intelligence Artificielle

- Les GANs ‘améliorés’

- Conditional GANs :

- Everybody Dance Now



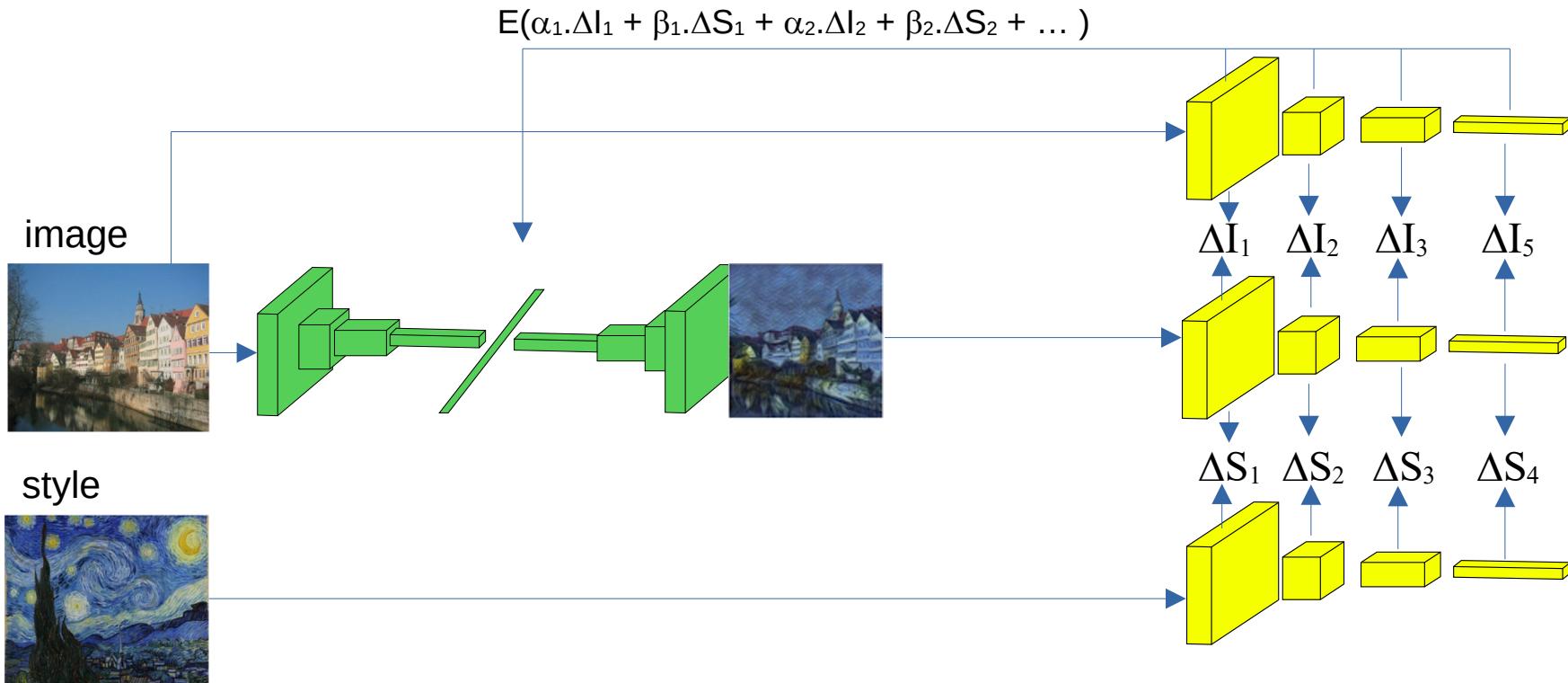
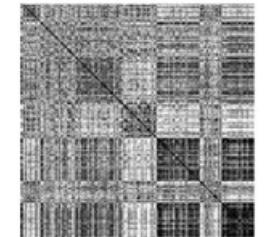
# Introduction à l'Intelligence Artificielle

## • Les GANs ‘améliorés’

### – Transfert style GAN :

- Plus une convergence qu'un apprentissage
- Réseaux encodeurs pré-entraînés (sans fully-connected)
- Différences d'images et différence de style (matrices de Gram)

$$g_{k_1, k_2} = Y_{k_1} \times Y_{k_2}$$



# Introduction à l'Intelligence Artificielle

- Les GANs ‘améliorés’
  - Transfert style GAN



[https://www.tensorflow.org/lite/examples/style\\_transfer/overview](https://www.tensorflow.org/lite/examples/style_transfer/overview)

# Introduction à l'Intelligence Artificielle

- **Text to Image**
  - DALLE-2 (OpenAI)
  - Imagen (Google)



Sprouts in the shape of text 'Imagen' coming out of a fairytale book.



A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat.



A high contrast portrait of a very happy fuzzy panda dressed as a chef in a high end kitchen making dough. There is a painting of flowers on the wall behind him.



Teddy bears swimming at the Olympics 400m Butterfly event.



A cute corgi lives in a house made out of sushi.

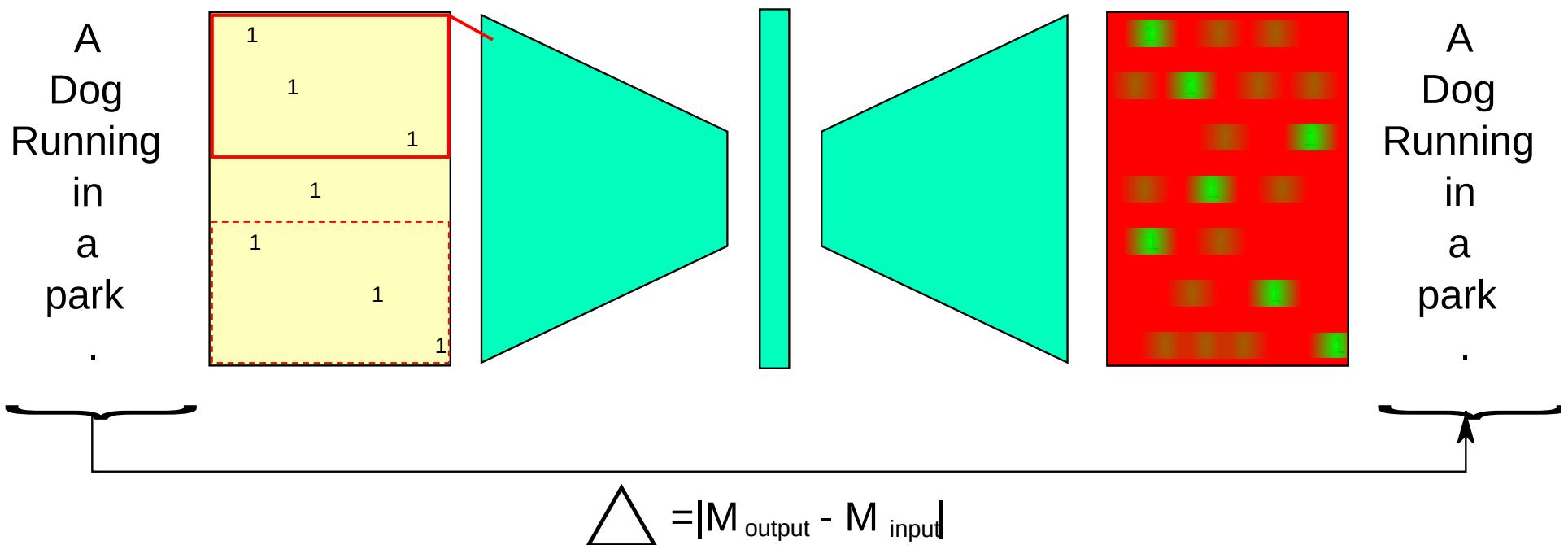


A cute sloth holding a small treasure chest. A bright golden glow is coming from the chest.

# Introduction à l'Intelligence Artificielle

- **Text2text**

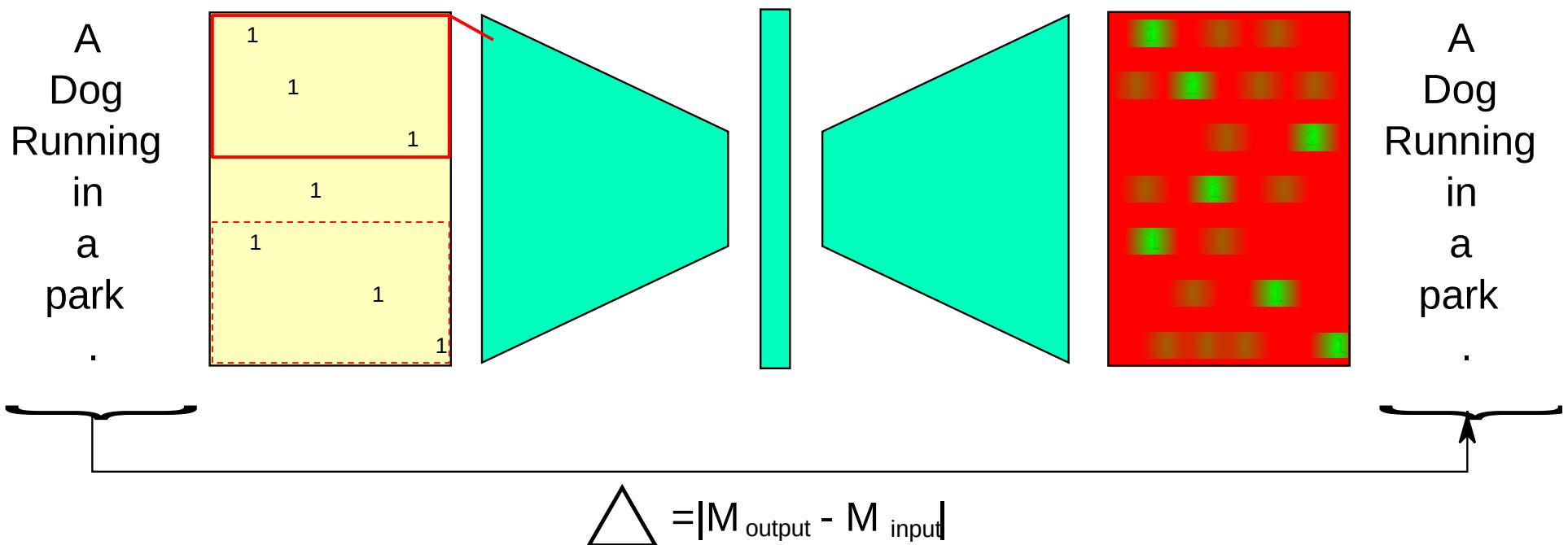
- Un mot est représenté par un vecteur de la taille du dictionnaire ('chien' = [ 0,0,0,0,...,0,0,1,0,0,...,0,0])
- Une phrase est représentée par une matrice de taille  $n_{\text{mots}} \times n_{\text{dictionnaire}}$
- Le réseau utilise sur ses premières couches des neurones convolutifs 1D, (souvent très nombreux) couvrant ainsi un mot et ses plus proches voisins.



# Introduction à l'Intelligence Artificielle

- **Text2text**

- Conséquence directe : un mot est indissociable de son contexte, et donc sa signification
  - Deux mots avec un sens proche seront proches dans l'espace latent
  - Un mot avec des définitions différentes sera représenté avec des vecteurs éloignés dans l'espace latent (ex : 'temps' formera des vecteurs différents si il est entouré de 'heure' ou de 'pluie')

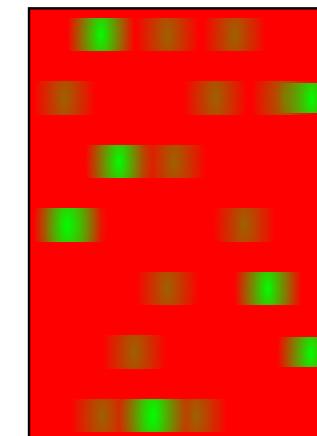
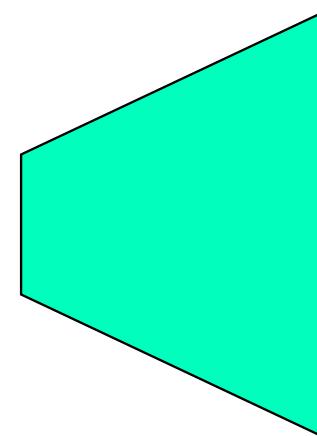
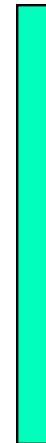
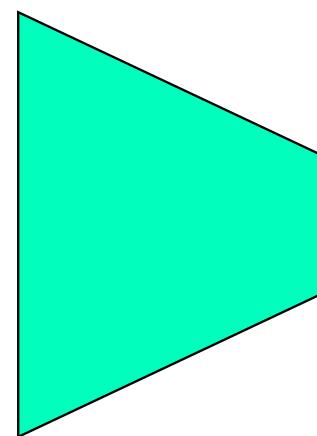
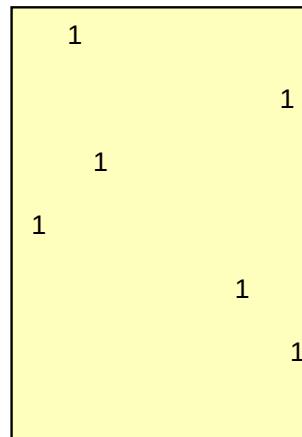


# Introduction à l'Intelligence Artificielle

- **Text2text**

- Pour l'exploitation : un mot est masqué, le réseau doit apprendre à retrouver le mot à partir de son contexte (BERT, GPT)
- ChatGPT 3 est une version très élaborée de ce principe
  - Calcul du mot suivant le plus probable

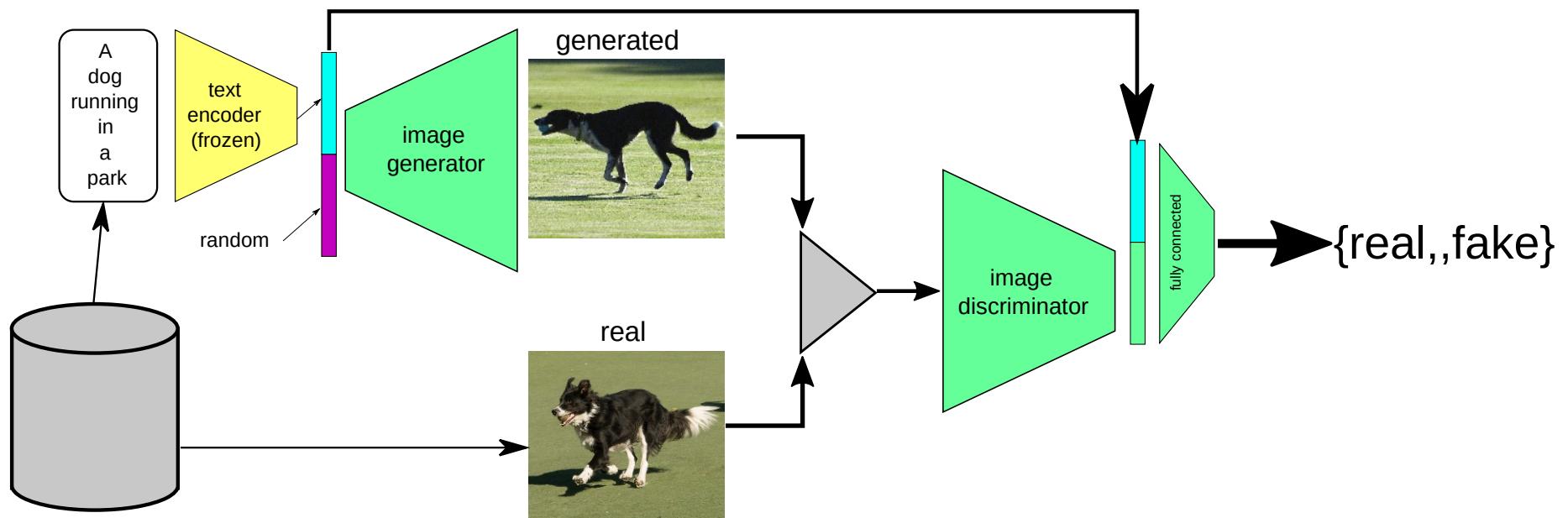
Hello  
,  
how  
are  
you  
?  
**EMPTY**



Hello  
,  
how  
are  
you  
?  
**Fine**

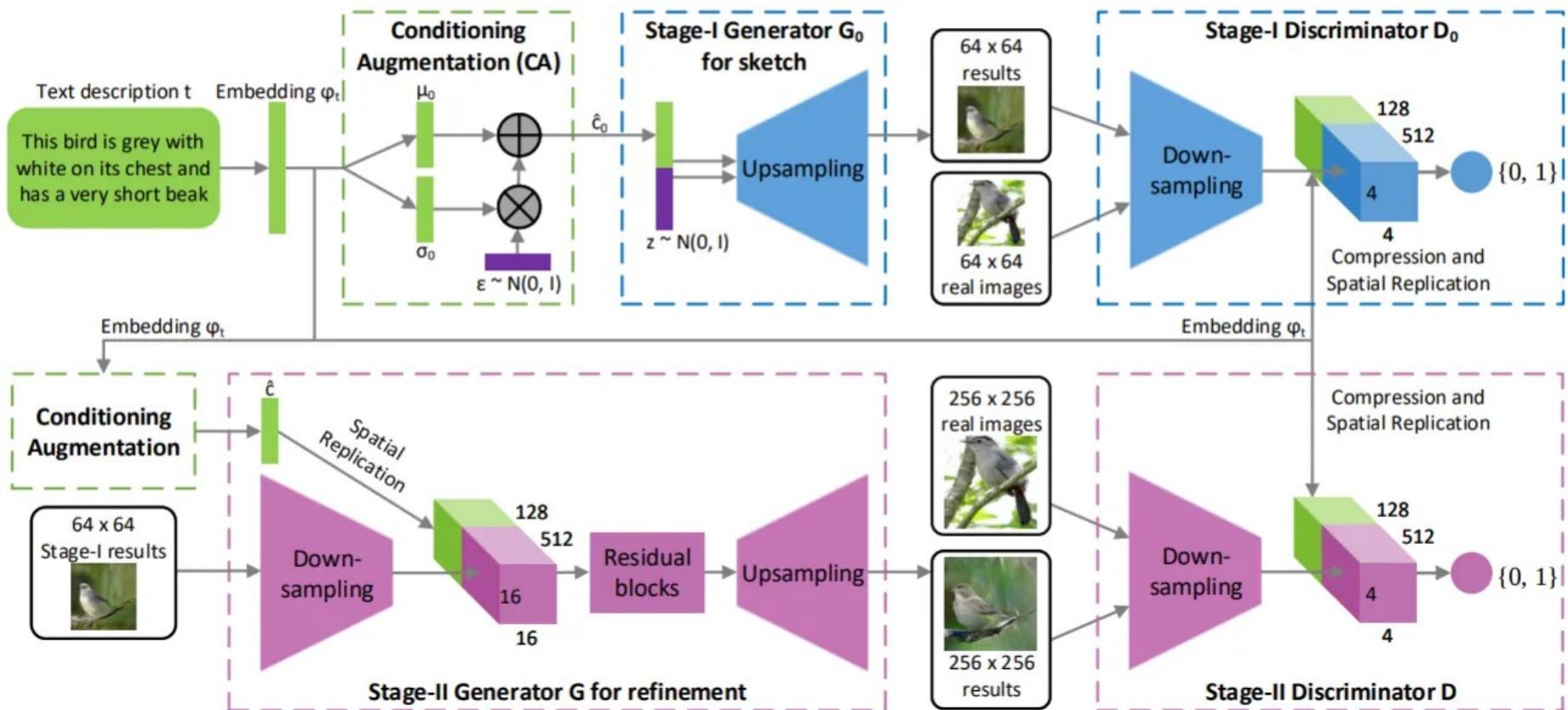
# Introduction à l'Intelligence Artificielle

- **Text2image : approche Conditional GAN**
  - Réseau encodeur (pré-entraîné) pour le texte : vecteur de condition
  - Conditional GAN utilisant en entrée un vecteur aléatoire ('seed') et le vecteur de condition
  - Comparaison de paires images-vecteur condition



# Introduction à l'Intelligence Artificielle

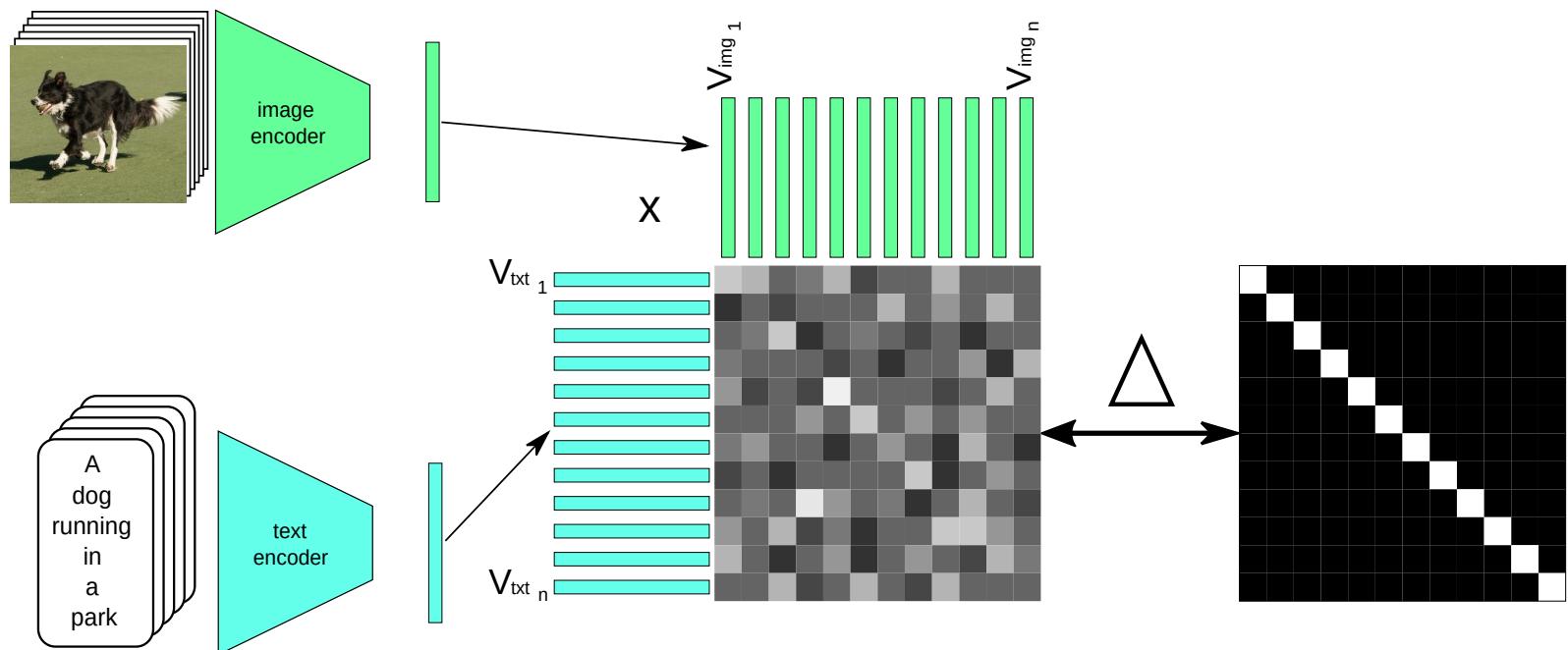
- **Text2image : approche Conditional GAN**
  - StackGAN<sup>1</sup> utilise un second ‘étage’ constitué d’un autoencodeur avec condition pour augmenter la résolution des images générées



<sup>1</sup><https://arxiv.org/abs/1612.03242>

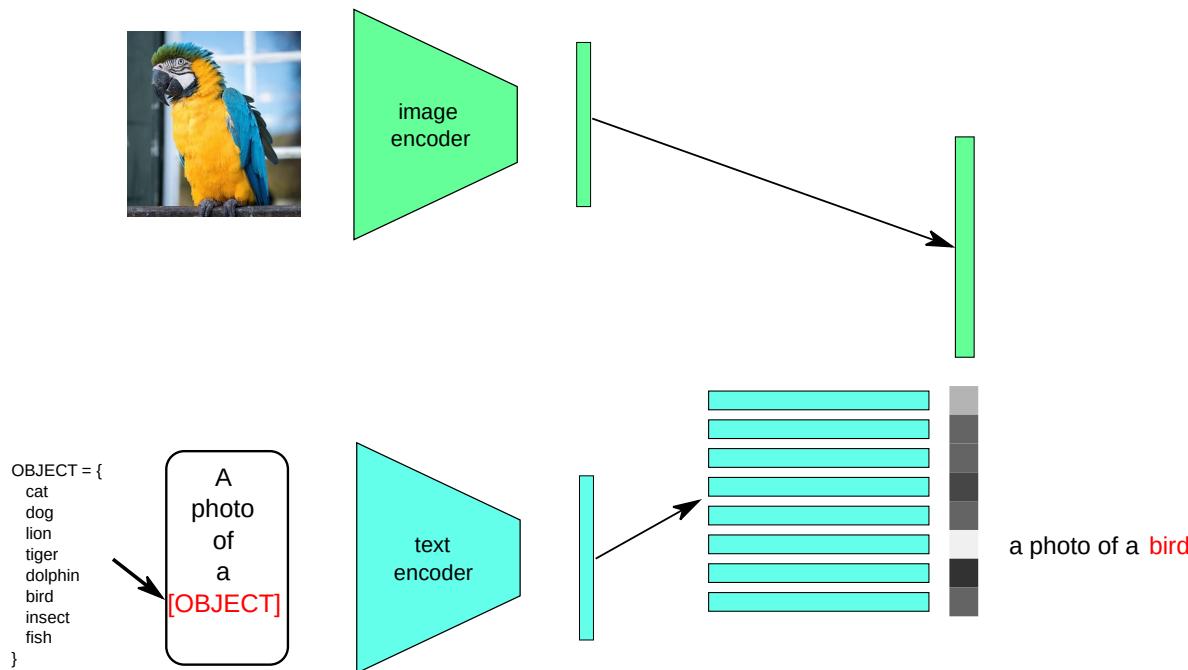
# Introduction à l'Intelligence Artificielle

- **Lier des modèles : CLIP (Contrastive Language–Image Pre-training)**
  - Deux réseaux encodeurs : un pour des images, un pour du texte
  - Matrice de corrélation : on maximise la diagonale et on minimise les autres
  - Idée : forcer les espaces latents des deux réseaux à converger vers une même répartition : la phrase ‘un chien qui cours dans un parc’ sera représenté par le même vecteur d’une image représentant cette scène.



# Introduction à l'Intelligence Artificielle

- **Lier des modèles : CLIP (Contrastive Language–Image Pre-training)**
  - On crée un ensemble de catégories et de phrases-type
  - On présente une image au réseau
  - Test des catégories : estimation de la catégorie la plus probable
  - On peut catégoriser *a posteriori* ! (Zero-Shot prediction)



# Introduction à l'Intelligence Artificielle

- **Diffusion**

- Principe : on ajoute du bruit à l'image, le modèle apprend à dé-bruiter l'image
- Différent niveaux de bruits sont ajoutés aux images d'entraînement

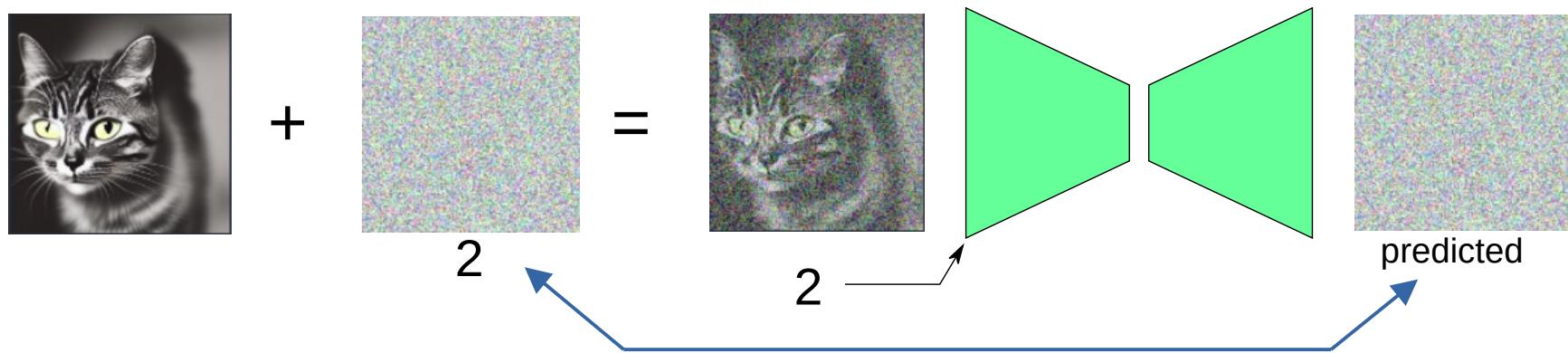


<https://stable-diffusion-art.com/how-stable-diffusion-work/>

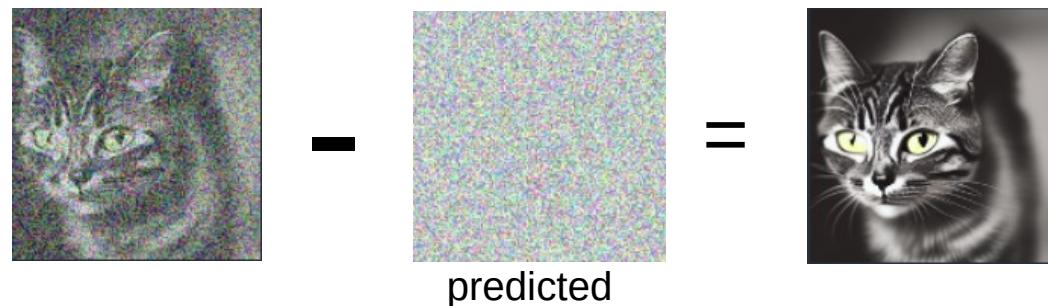
# Introduction à l'Intelligence Artificielle

- **Diffusion**

- Un réseau apprend à prédire le masque du bruit ajouté à l'image



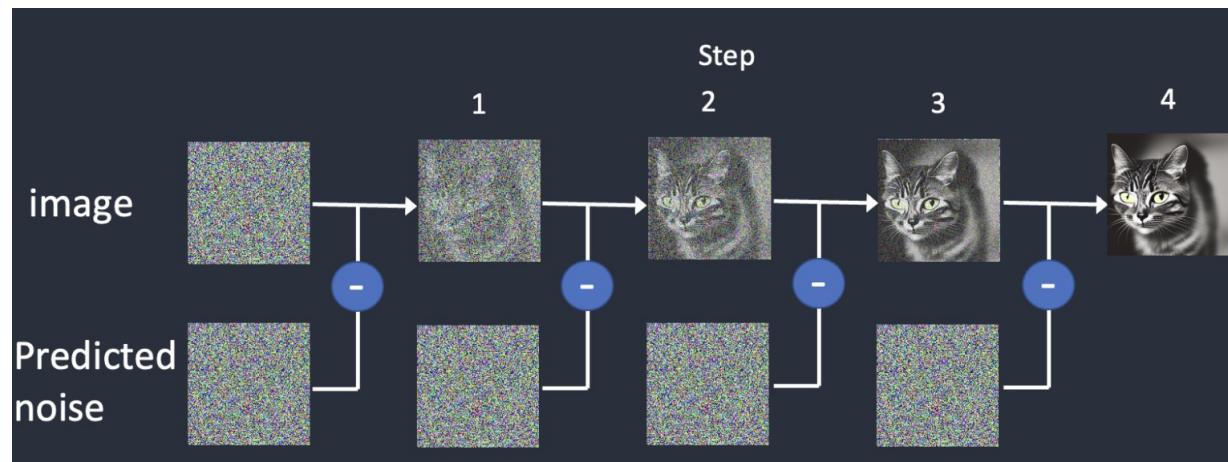
- En théorie, si on soustrait le bruit prédict à l'image bruitée, on doit obtenir l'image initiale



# Introduction à l'Intelligence Artificielle

- **Diffusion**

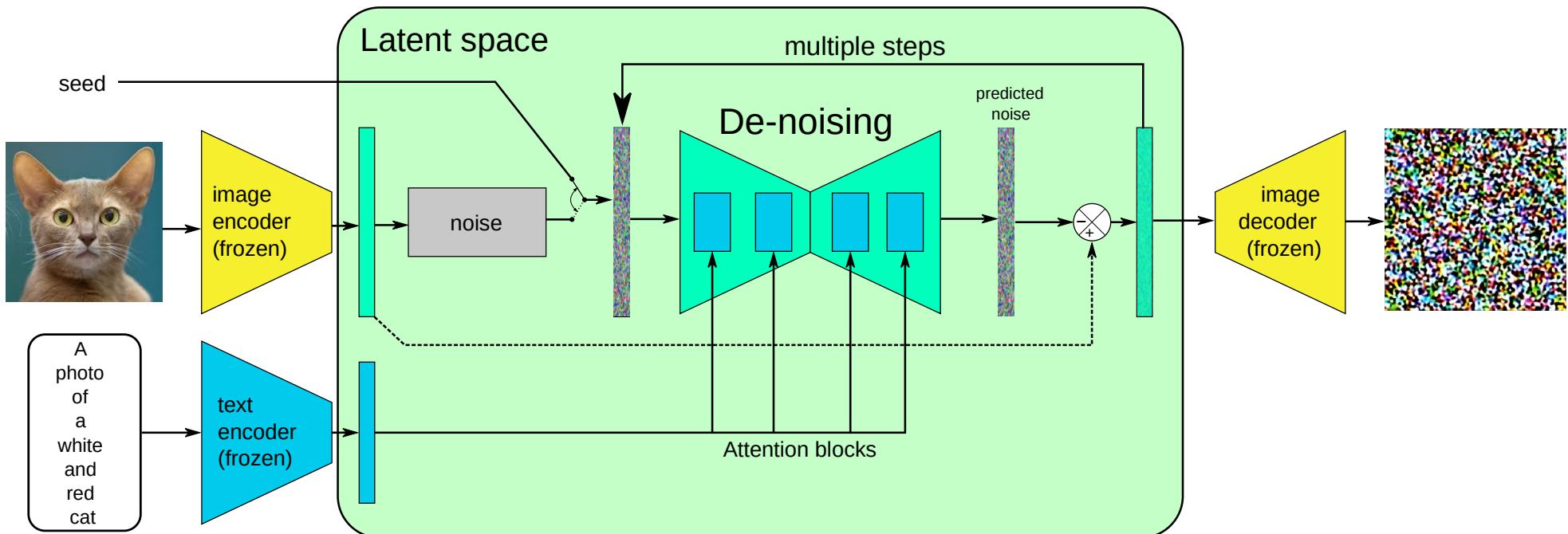
- Si on applique ce principe à une image de bruit aléatoire, le réseau va prédire un masque de bruit que l'on peut soustraire à l'image



- On peut répéter le processus plusieurs fois ('diffusion') Pour obtenir une image convenable
  - Forme de paréidolie artificielle
  - Les images générées dépendent de la base d'apprentissage !

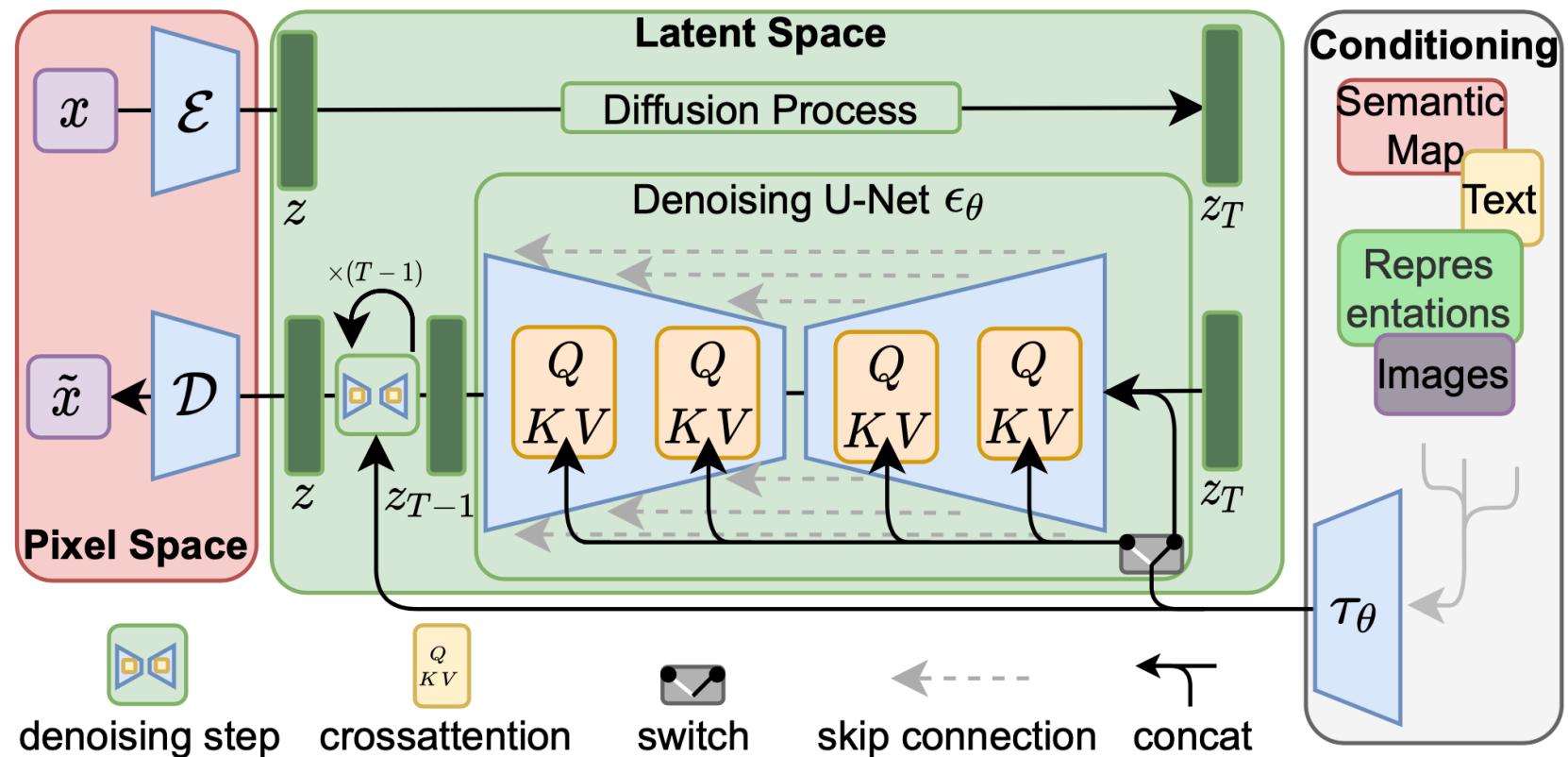
# Introduction à l'Intelligence Artificielle

- **Latent diffusion**
  - Le bruit est ajouté sur le vecteur de l'espace latent et non sur l'image → bruit sur des caractéristiques de haut niveau
- **Conditions : attention blocks**
  - Il est possible de guider le modèle à l'aide de conditions (par exemple du texte) → relations entre texte et images (CLIP), augmente ou réduit les valeurs de certains éléments du vecteur de l'espace latent
  - modèle dit de cross-attention (relations entre mots d'une phrase)



# Introduction à l'Intelligence Artificielle

- Stable diffusion



Rombach & Blattmann, et al. 2022

# Librairies pour le Deep Learning

# Introduction à l'Intelligence Artificielle

- **Plusieurs librairies pour faciliter le développement :**

- TensorFlow : développé par Google, rendu libre en 2015, issue du projet Disbelief (2011)
  - Depuis 2017, il existe une version Lite pour les systèmes embarqués



- MXNet : développé par Apache, utilisable dans de nombreux langages



- Caffe



- Théanos : depuis 2008



- Microsoft Cognitive Toolkit : depuis 2016



- PyTorch : développée par Facebook, s'appuie sur Torch



- Keras : bibliothèque proposant des fonction pour utiliser de façon simple les librairies Tensorflow et Théanos

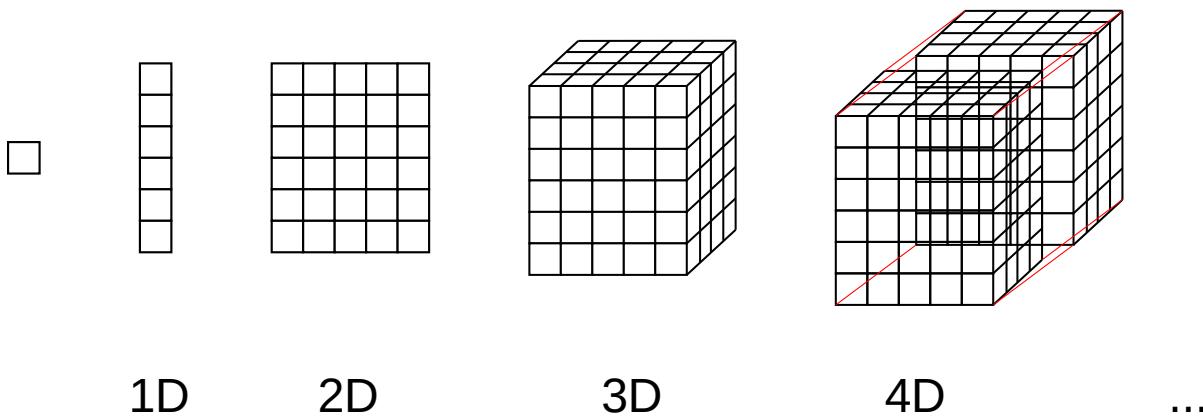


# Introduction à l'Intelligence Artificielle



- **TensorFlow**

- Librairie destinée à manipuler des *tenseurs*
  - Un tenseur est une matrice avec un nombre quelconque de dimensions



- Nombreuses fonctions optimisées
- Possibilité de parallélisation sur GPU !

# Introduction à l'Intelligence Artificielle



- **TensorFlow**

- Import : `import tensorflow as tf`

```
A = tf.constant([[1,2,3],[4,5,6]])
A = tf.Variable([[1,2,3],[4,5,6]])
```

- Constructeurs :

```
A = tf.ones((3,3))
A = tf.zeros((3,3))
A = tf.eye(3)
```

```
A = tf.random.normal((3,3), mean=0, stddev=1)
A = tf.range(5)
```

```
tf.Tensor(
[[1 2 3]
 [4 5 6]], shape=(2, 3), dtype=int32)           <tf.Variable 'Variable:0' shape=(2, 3) dtype=int32, numpy=
                                                    array([[1, 2, 3],
                                                       [4, 5, 6]], dtype=int32)>
```

```
tf.Tensor(
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]], shape=(3, 3), dtype=float32)        tf.Tensor(
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]], shape=(3, 3), dtype=float32)        tf.Tensor(
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]], shape=(3, 3), dtype=float32)
```

```
tf.Tensor(
[[ 1.3420206  1.7878479 -1.0414563]
 [-1.5547394  0.5375859 -1.1487345]
 [ 1.2995738  1.029388   0.9286815]], shape=(3, 3), dtype=float32)
```

```
tf.Tensor([0 1 2 3 4], shape=(5,), dtype=int32)
```

# Introduction à l'Intelligence Artificielle



- **TensorFlow**

- Quelques fonctions pour manipuler les tenseurs :

```
C = tf.add(A,B)          # somme elt par elt
C = tf.subtract(A,B)     # soustraction elt par elt
C = tf.multiply(A,B)    # multiplication elt par elt
C = tf.divide(A,B)      # division elt par elt

s = tf.tensordot(A,B, axes=1) # produit scalaire
c = tf.matmul(A,B)           # produit matriciel

N = C.numpy()              # conversion tenseur->numpy
C = tf.convert_to_tensor(N) # conversion numpy-> tenseur

v = tf.Variable(A)          # conversion constant->variable
v.assign([5,3,9])           # mise à jour variables
```

# Introduction à l'Intelligence Artificielle



- **Keras**

- Bibliothèque de fonctions exploitant Tensorflow ou Theanos

```
import tensorflow as tf
from tensorflow import keras
```

- Permet d'effectuer très simplement les étapes suivantes :

- Charger un dataset ‘courant’ (ie souvent utilisé pour les benchmarks)
    - Définir l’architecture de son réseau
    - Définir les paramètres d’apprentissage
    - Entraîner le réseau
    - Évaluer le réseau
    - Utiliser le réseau pour faire des prédictions

# Introduction à l'Intelligence Artificielle



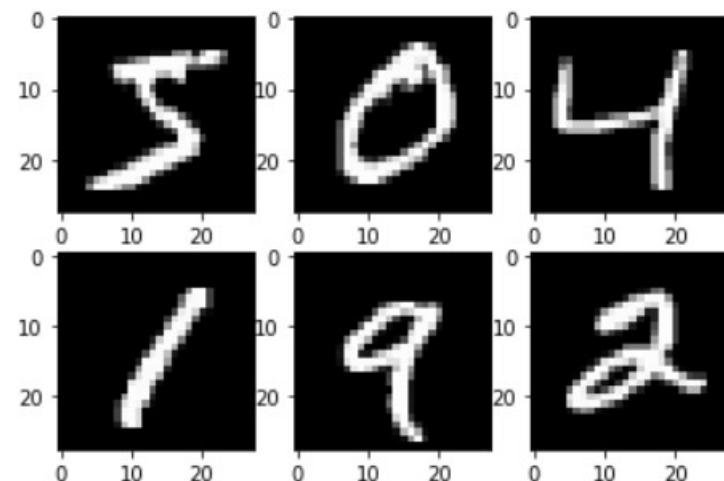
- **Keras**

- Chargement d'un dataset :

```
dataset = keras.datasets.mnist  
  
(img_train, output_train), (img_test, output_test) = dataset.load_data()  
  
img_train=img_train/255.0  
img_test=img_test/255.0
```

```
print(img_train.shape)  
print(output_train.shape)
```

```
(60000, 28, 28)  
(60000,)
```

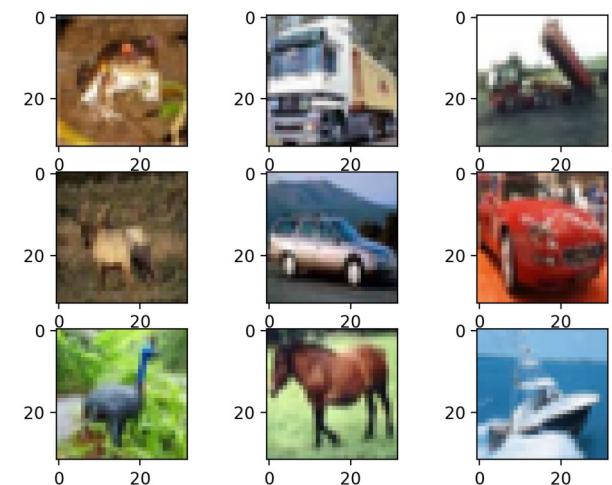


# Introduction à l'Intelligence Artificielle



- **Keras**

- Datasets disponibles :
  - Boston housing prices (valeurs)  
→ *boston\_housing*
  - CIFAR-10 (images couleurs de 32x32)  
→ *cifar10*
  - CIFAR-100 (idem avec 100 classes)  
→ *cifar100*
  - Fashion MNIST (images N&B 28x28)  
→ *fashion\_mnist*
  - IMDB movie reviews (textes) → *imdb*
  - MNIST (images N&B 28x28) → *mnist*
  - Articles Reuters (textes) → *reuters*



CIFAR-10  
(50 000 images)

# Introduction à l'Intelligence Artificielle



- **Keras**

- Définir l'architecture du réseau :
  - Modèle défini couche par couche (comme notre réseau multi-couche)
  - Initialisation du modèle (méthode séquentielle) :

```
network = keras.Sequential()
```

- Puis on ajoute les couches successives (de différents types) du modèle :

```
network.add(keras.layers.Flatten(input_shape=(28, 28)))
network.add(keras.layers.Dense(40, activation='sigmoid'))
network.add(keras.layers.Dense(10, activation='sigmoid'))
```

# Introduction à l'Intelligence Artificielle



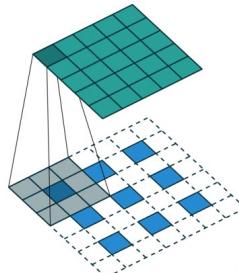
- **Keras** : Quelques types de couches importantes (avec quelques paramètres importants) :
  - **tf.keras.layers.Dense(**  
*units,*  
*activation=None,*  
*use\_bias=True,*  
    **)**
    - nombre de neurones de la couche
    - fonction d'activation
    - utilisation d'un biais
  - **tf.keras.layers.Conv2D(**  
*filters,*  
*kernel\_size,*  
*strides=(1, 1),*  
*padding="valid",*  
*activation=None,*  
*use\_bias=True,*  
    **)**
    - note : existe en version 1D et 3D
    - nombre de neurones
    - taille du kernel
    - déplacements (pas) du kernel
    - présence d'un padding si "same"
    - fonction d'activation
    - utilisation d'un biais
- Quelques fonctions d'activation : relu, sigmoid, softmax, tanh, ...

# Introduction à l'Intelligence Artificielle



- **Keras** : Quelques types de couches importantes :

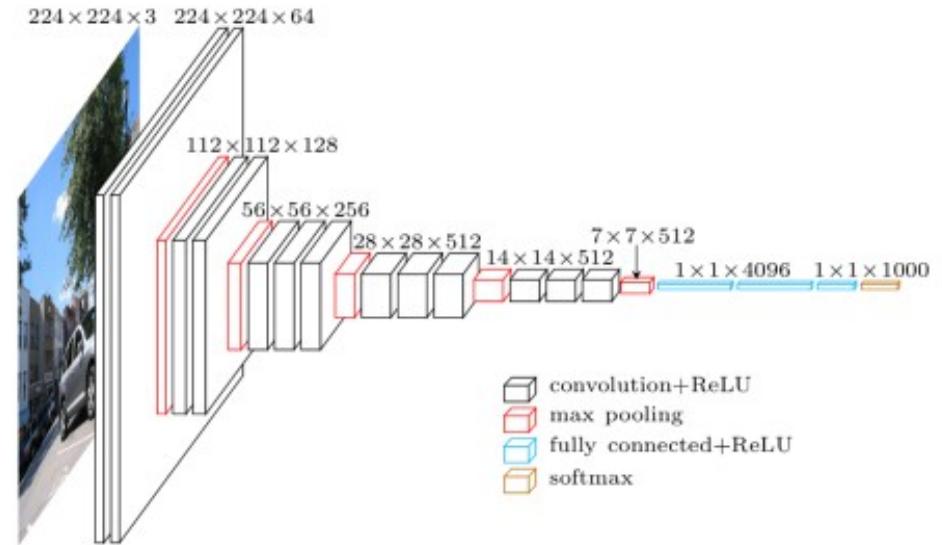
- **tf.keras.layers.MaxPooling2D(**  
    `pool_size=(2, 2),`  
    `strides=None,`  
    `padding="valid"`  
    `)`  
    - réduction du pooling
    - pas du déplacement
  - **tf.keras.layers.UpSampling2D(**  
    `size=(2, 2),`  
    `interpolation="nearest"`  
    `)`  
    - augmentation du unpooling
    - "nearest" ou "bilinear"
  - **tf.keras.layers.Conv2DTranspose(**  
    `filters,`  
    `kernel_size,`  
    `strides=(1, 1),`  
    `padding="valid",`  
    `activation=None,`  
    `use_bias=True,`  
    `)`  
    - nombre d'images en sortie
    - taille du noyau
    - ajoute des 0 entre les pixels
    - permet un 'bed of nails'
    - avec stride=(2,2)
- Liste complète des 100+ types de layers sur : <https://keras.io/api/layers/>



# Introduction à l'Intelligence Artificielle

- **Keras**

- Implémentons l'architecture VGG16 :



```
model = Sequential( )
```

```
model.add(Conv2D(input_shape=(224,224,3), filters=64, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
```

```
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

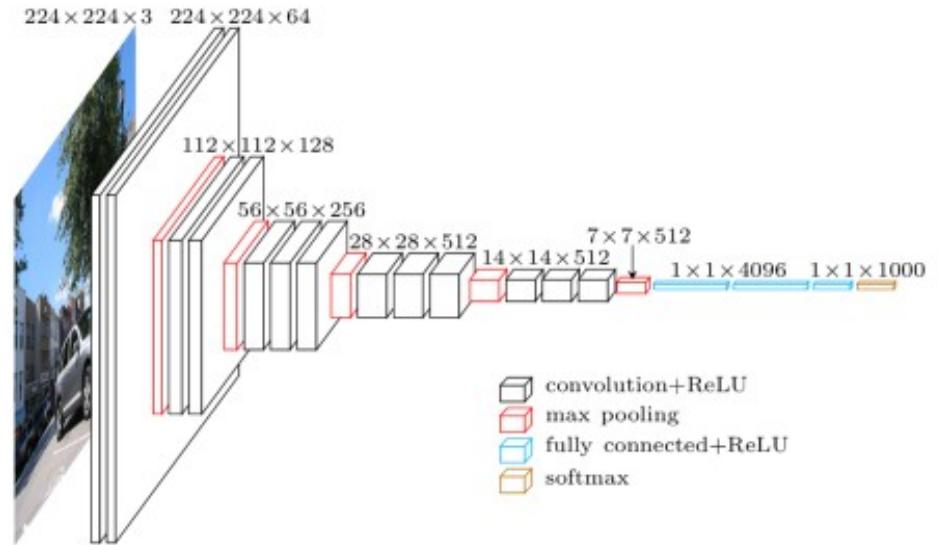
```
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu")) X 3
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu")) X 3
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

# Introduction à l'Intelligence Artificielle

- **Keras**

- Implémentons l'architecture VGG16 :



...

```
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu"))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
model.add(Flatten( ))
```

```
model.add(Dense(units=4096,activation="relu"))
model.add(Dense(units=4096,activation="relu"))
```

```
model.add(Dense(units=2, activation="softmax"))
```

# Introduction à l'Intelligence Artificielle



- **Keras**

- On peut aussi implémenter un séquentiel avec un vecteur de couches :

```
model = Sequential( [  
    Conv2D(input_shape=(224,224,3), filters=64, kernel_size=(3,3), padding="same", activation="relu"),  
    Conv2D(filters=64,kernel_size=(3,3), padding="same", activation="relu"),  
    MaxPool2D(pool_size=(2,2), strides=(2,2)), ...  
])
```

- Autre méthode : le modèle fonctionnel

- On crée les couches séparément et on connecte les entrées et les sorties

```
input_layer = Input(shape=(28,28,1))  
conv1 = Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32,32,1))(input_layer)  
conv2 = Conv2D(64, (3, 3), activation='relu', input_shape=(32,32,1))(input_layer)  
merge = concatenate([conv1, conv2])  
flatten = Flatten()(merge)  
output_layer = Dense(num_classes, activation='softmax')(flatten)  
model = Model(inputs=input_layer, outputs=output_layer)
```

- Permet de définir des réseaux non séquentiels (fusion, séparation...)

# Introduction à l'Intelligence Artificielle



- **Keras : Paramètres d'apprentissage**

- On sélectionne la fonction de coût (loss) :

```
loss=keras.losses.CategoricalCrossentropy()
```

on a aussi :

- *MeanSquaredError* :  $E = (\hat{y} - y)^2$
  - ... et quelques dizaines d'autres sur <https://keras.io/api/losses>
- Si le vecteur de résultats est sous la forme de labels ([ 1, 5, 3, 0, 6, ... ]), il faut les convertir en vecteurs pour les comparer avec la couche de sortie
  - On utilise la fonction *to\_categorical* :

```
output_train=keras.utils.to_categorical(output_train, num_classes=10)
```

→ [ [0,1,0,0,0,0,0,0,0],[0,0,0,0,1,0,0,0,0,0],[0,0,1,0,0,0,0,0,0,0],... ]

- À noter : la fonction de coût *SparseCategoricalCrossentropy* est une fonction CategoricalCrossentropy qui se sert d'un vecteur de labels

# Introduction à l'Intelligence Artificielle



- **Keras : Paramètres d'apprentissage**

- On sélectionne la fonction d'optimisation :

```
optim = keras.optimizers.SGD(learning_rate=0.01)
```

on a aussi quelques fonctions plus optimisées : Adam, Adadelta, Nadam, Ftrl...  
(voir <https://keras.io/api/optimizers/>)

- Puis compile le réseau :

```
model.compile(loss=loss, optimizer=optim, metrics=["accuracy"])
```

# Introduction à l'Intelligence Artificielle



- **Keras : entraînement du réseau**

- On utilise la fonction *fit* :

```
model.fit(img_train, output_train, batch_size=64, epochs=10, shuffle=True, verbose=2)
```

- img\_train et output\_train : le dataset et les réponses attendues
    - batch\_size : le batch est le nombre d'exemple testés simultanément
    - Epochs : nombre d'epochs
    - Shuffle : pour mélanger ou non le batch
    - Verbose : gérer les données affichées pendant l'apprentissage

```
Epoch 6/10
938/938 - 1s - loss: 0.0361 - accuracy: 0.9889
Epoch 7/10
938/938 - 1s - loss: 0.0354 - accuracy: 0.9892
Epoch 8/10
938/938 - 1s - loss: 0.0348 - accuracy: 0.9894
Epoch 9/10
938/938 - 1s - loss: 0.0343 - accuracy: 0.9896
Epoch 10/10
938/938 - 1s - loss: 0.0338 - accuracy: 0.9898
```

# Introduction à l'Intelligence Artificielle



- **Keras : évaluation et exploitation**

- On évalue le réseau avec *evaluate* sur le dataset d'évaluation

```
model.evaluate(img_test, output_test, batch_size=16, verbose=2)
```

```
10/10 - 0s - loss: 0.1384 - accuracy: 0.9654  
[0.1383775770664215, 0.965399980545044]
```

- On peut ensuite utiliser le réseau pour faire des prédictions :

```
model.predict(img_test[0:1,:,:])
```

```
array([[1.0455251e-03, 1.4841557e-04, 5.5478072e-01, 9.7418678e-01,  
       7.2960603e-07, 2.4808049e-03, 8.8775799e-11, 9.9999410e-01,  
       1.3713300e-02, 2.5117993e-02]], dtype=float32)
```

On peut aussi récupérer un tenseur avec :

```
model(img_test[0:1,:,:])
```

# Introduction à l'Intelligence Artificielle

- **Conclusion :**

- Le domaine du deep learning est un domaine assez récent
- Mais qui donne des résultats déjà impressionnantes
- Des progrès restent à faire sur de nombreux aspects, et la marge de progression est énorme
- Un réseau neuronal reste cependant un classifieur, incapable de comprendre ou d'interpréter les résultats qu'il génère, et ne connaît rien de plus que ce qu'on lui a présenté en exemple
- Une IA doit, pour se développer, pouvoir explorer par elle-même son environnement
  - en route pour l'apprentissage par renforcement ?

# Introduction à l'Intelligence Artificielle

- State-of-art :

