agaetis

Big Data & Data Science

Détection d'objets en temps réel

Application d'un algorithme de *Deep Learning*

Léo Beaucourt

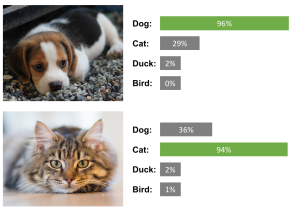
Contexte

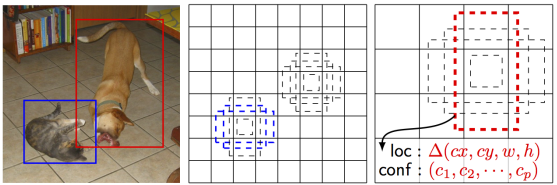
- **Construire une application de détection d'objet en temps réel**
- Computer vision:
  - Véhicule autonomes
  - Modération de contenu (facebook, youtube, ...)
  - Reconnaissance faciale
  - Projets artistiques
- Réseaux de Neurones convolutifs
- *Tensorflow*, *PyTorch*, ...

De la théorie ...

Vision par ordinateur

- Reconnaissance d'images:
  "J'envoie une image à mon classificateur. Il me dit si c'est un chat ou un chien!"
- Détection d'objets:
  "J'envoie une image à mon détecteur. Il me dit où sont les chats et les chiens!"



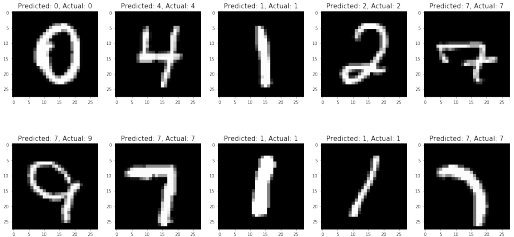


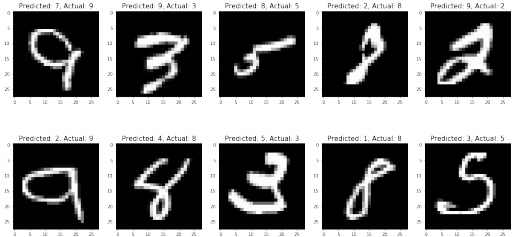
(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

- Une image == matrice (tenseur) ($x_i \in [0,255]$)

Reconnaissance d'images avec une régression logistique

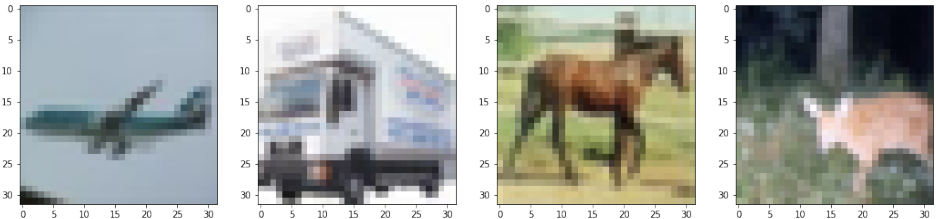
- Classification (multiclasse)
- Objets *"simples"*: chiffres manuscrits
- Régression logistique multiclasse sur MNIST:
  - Durée d'apprentissage $\sim 20sec$ / Précision $\approx 91\%$





Reconnaissance d'objets plus complexes

- Algos ML/DL $\rightarrow$ détectent des structures (*patterns*): lignes, courbes
- Objects == patterns complexes



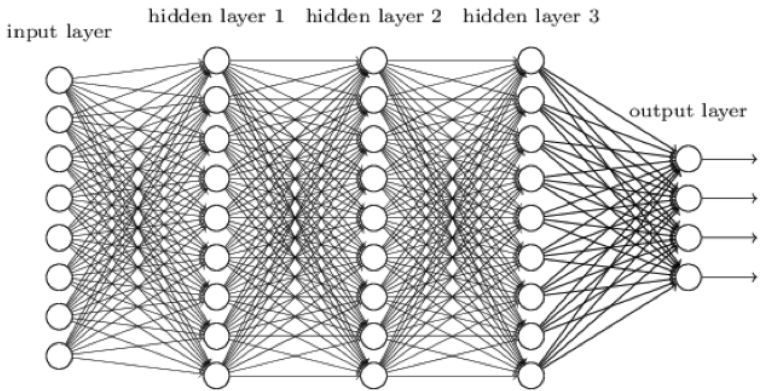
- Regression logistique limitée $\Rightarrow$ Réseaux de neurones !

Deep learning et Réseaux de Neurones

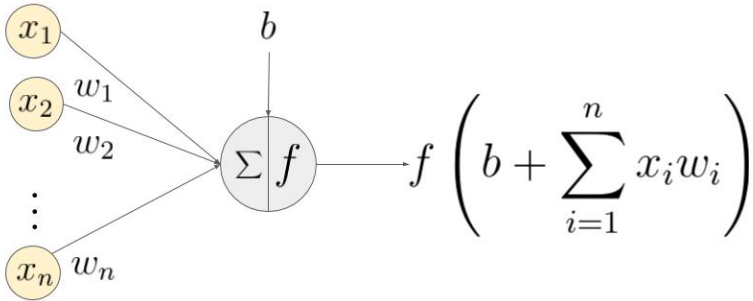
- Buzzword IA == Deep learning (DL) == Réseaux de Neurones (NN)
- Grande quantité de données (**labellisées!**)
- Algorithme d'optimisation: **Forward/Backward propagation**
- Temps/puissance de calcul: GPU

Réseaux de Neurones (fully connected NN): L-layers

- Couche l: $n^{[l]}$ neurones ($n^{[0]} = m$) / $W^{[l]} \in \mathbb{R}^{(n^{[l]} \times n^{[l-1]})}$, $b^{[l]} \in \mathbb{R}^{(n^{[l]} \times 1)}$



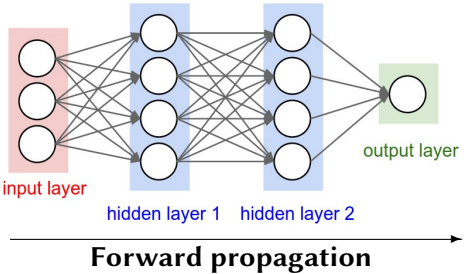
Neurone



- $z_i^{[l]} = \sum_{j=0}^{n^{[l-1]}} w_{ij}^{[l]} \times a_j^{[l-1]} + b_i^{[l]}$ Où, $a^{[0]} = x$
- $a_i^{[l]} = f^{[l]}(z_i^{[l]})$ Où, $f^{[l]}(z)$ **fonction d'activation**

Forward propagation

- Input $\boldsymbol{X}$: $n^{[0]} = 3$
- 2 couches cachées ($n^{[1]} = n^{[2]} = 4$): **ReLU**
- Output: $n^{[3]} = 1$: **Logistique**

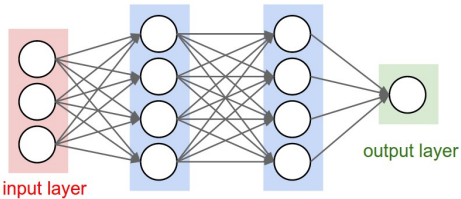


$$
\begin{array}{lcllcl}
X & \in & \mathbb{R}^{3\times m} & Y & \in & \mathbb{R}^{1\times m} \\
W^{[1]} & \in & \mathbb{R}^{4\times 3} & b^{[1]} & \in & \mathbb{R}^{4\times 1} \\
Z^{[1]} & \in & \mathbb{R}^{4\times m} & A^{[1]} & \in & \mathbb{R}^{4\times m} \\
W^{[2]} & \in & \mathbb{R}^{4\times 4} & b^{[2]} & \in & \mathbb{R}^{4\times 1} \\
Z^{[2]} & \in & \mathbb{R}^{4\times m} & A^{[2]} & \in & \mathbb{R}^{4\times m} \\
W^{[3]} & \in & \mathbb{R}^{1\times 4} & b^{[3]} & \in & \mathbb{R}^{1\times 1} \\
Z^{[3]} & \in & \mathbb{R}^{1\times m} & A^{[3]} & \in & \mathbb{R}^{1\times m}
\end{array}
$$

$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= f^{[1]}(Z^{[1]}) = relu(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= f^{[2]}(Z^{[2]}) = relu(Z^{[2]}) \\
Z^{[3]} &= W^{[3]}A^{[2]} + b^{[3]} \\
\hat{Y} &= A^{[3]} = f^{[3]}(Z^{[3]}) = \sigma(Z^{[3]})
\end{aligned}
$$

Forward propagation

- **descente de gradient**
- Fonction de coût: $\mathcal{J}(W^{[l]}, b^{[l]})$
- Propager l'erreur $\hat{Y} - Y$, modifie $W^{[l]}$ et $b^{[l]}$



Répéter: {

Calculez: $\hat{Y}$

$dW^{[L]} := \frac{d\mathcal{J}}{dW^{[L]}}, \quad db^{[L]} := \frac{d\mathcal{J}}{db^{[L]}}$

$W^{[L]} := W^{[L]} - \alpha dW^{[L]}$

$b^{[L]} := b^{[L]} - \alpha db^{[L]}$

$\vdots$

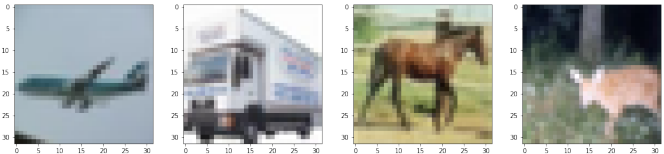
}

Backward propagation

$$
\begin{aligned}
dZ^{[3]} &= A^{[3]} - Y \\
dW^{[3]} &= \tfrac{1}{m} dZ^{[3]} A^{[2]\,T} \\
db^{[3]} &= \tfrac{1}{m} \sum dZ^{[3]} \\
dZ^{[2]} &= W^{[3]\,T} dZ^{[3]} * relu'(Z^{[2]}) \\
dW^{[2]} &= \tfrac{1}{m} dZ^{[2]} A^{[1]\,T} \\
db^{[2]} &= \tfrac{1}{m} \sum dZ^{[2]} \\
dZ^{[1]} &= W^{[2]\,T} dZ^{[2]} * relu'(Z^{[1]}) \\
dW^{[1]} &= \tfrac{1}{m} dZ^{[1]} X^{T} \\
db^{[1]} &= \tfrac{1}{m} \sum dZ^{[1]}
\end{aligned}
$$

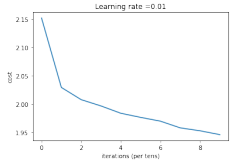
Reconnaissance d'images avec un *fully connected NN*

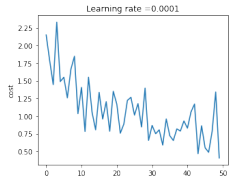
- **CIFAR-10:** 10 classes d'objets, 30K images (train set)

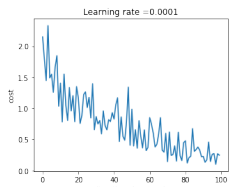


- Architecture: L = 3, $n^{[1]} = 2000$, $n^{[2]} = 1000$
- Mini-batch, ADAM initializer, regularisation, 4 CPU, 8 Go

itérations	**durée**	**précision**
10	5 min	0.18
50	7.6h	0.83
100	14.6h	0.96







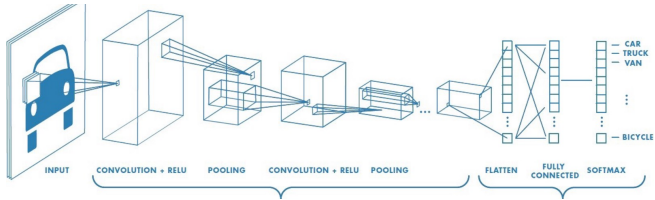
Détection d'objets et grandes images

- Détection d'objets: Classification + Localisation (classe + coordonnées)
- Grandes images (1000 x 1000):
  - 3×10^6 features !
  - $n^{[1]} = 1000$: $W^{[1]}$ == 3×10^9 valeurs
  - Overfitting, mémoire saturée

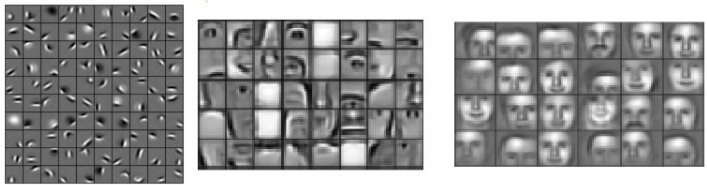
- Solution: Réseau de Neurones Convolutifs (CNN)

Réseaux de neurones convolutifs (CNN): Y. LeCun (1989)

- Couches CNN: Filtres (détecteurs)

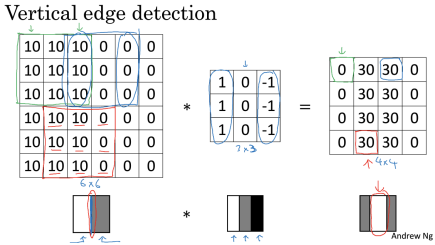


- Reconnaissance de formes (*patterns*) de plus en plus complexes



CNN: Edge detection

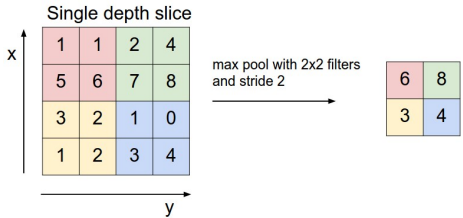
- Une forme a détecter == un filtre convolué à l'image (convolution: $*$)



- Apprentissage: paramètres du filtre ($\sim W^{[i]}$ d'un NN)
- Plusieurs filtres $\Rightarrow$ *Nouvelle image*, volume $(X \times X \times n_{filters})$
- CONV == $[ReLu + biais]$ en sortie de filtre

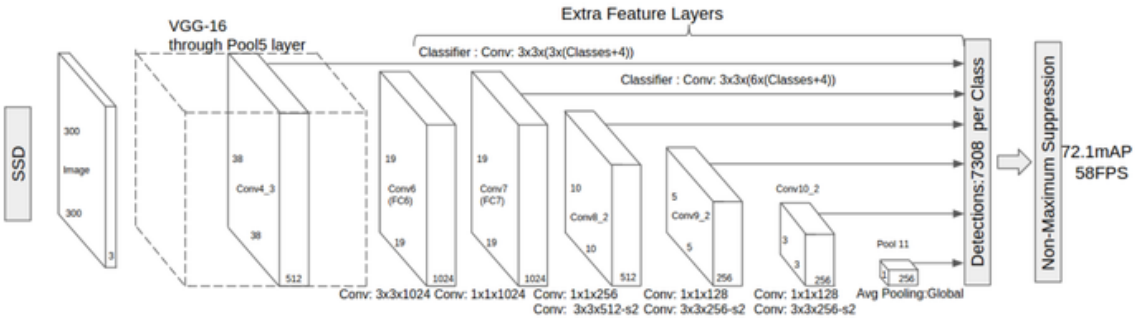
CNN: POOL layer

- Réduit la taille des images intermédiaire & le temps de calcul
- Formes détectées plus robustes



- Pas de paramètres à apprendre
- Un CNN == CONV + POOL + CONV + POOL + CONV + ...
- Exemple: VGG-16: 16 couches, 138M parameters

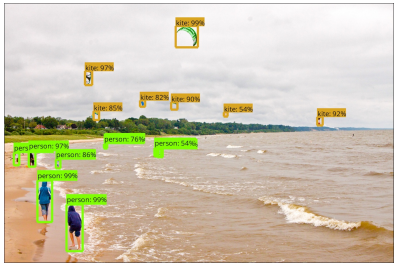
Single Shot (MultiBox) Detector



... à la pratique.

Tensorflow

- Bibliothèque open-source de DL, Google (2011)
- Interface python, Java-Script, …
- API détection d'objet: API
- Modèles pré-entrainés: models

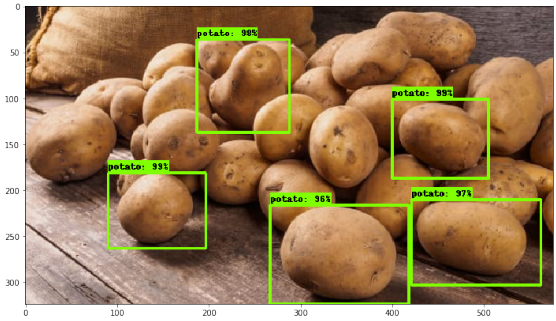


- Surcouche tensorflow: **Keras**

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes
rfcn_resnet101_coco	92	30	Boxes
faster_rcnn_resnet101_coco	106	32	Boxes
faster_rcnn_resnet101_lowproposals_coco	82		Boxes
faster_rcnn_inception_resnet_v2_atrous_coco	620	37	Boxes
faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco	241		Boxes
faster_rcnn_nas	1833	43	Boxes
faster_rcnn_nas_lowproposals_coco	540		Boxes
mask_rcnn_inception_resnet_v2_atrous_coco	771	36	Masks
mask_rcnn_inception_v2_coco	79	25	Masks
mask_rcnn_resnet101_atrous_coco	470	33	Masks
mask_rcnn_resnet50_atrous_coco	343	29	Masks

Tensorflow: utiliser le module de détection d'objet

Load Frozen model in tf.Graph();
Create category index from labels;
Load images into numpy array;
Initialize tf.Session();
for *image* **do**
    Define tensors object;
    Perform detection;
    if *visualization* **then**
        Print image with detections;
    end
end



OpenCV

- Bibliothèque open-source,traitement d'images temps réel (Intel)
- inputs vidéo $\xrightarrow{\text{OpenCV}}$ frames $\xrightarrow{\text{Tensorflow}}$ détection $\xrightarrow{\text{OpenCV}}$ output vidéo

```
cv2.VideoCapture() Get stream;
cv2.VideoWriter() Create output stream;
stream.read() Get frame & next frame status;
while next frame exist: do
    Send frame to tensorflow detector;
    Add detection boxes to frames;
    Display output frame;
    Write output frame;
    stream.read() Get frame & next frame status;
end
Destroy all cv2 objects;
```

Performances: Threading & Multiprocessing

- **Threading:** Séparer la lecture de la vidéo du process python
- **Multiproccesing:** Pool de *workers* effectuant la détection. Input/output queues

```
Thread(target=self.update, args=()).start();
def update(self):
while True: do
    if self.stopped: then
        return
    else
        (self.grabbed, self.frame) = self.stream.read();
    end
end
```

```
Create input/output queues;
Define workers pool;
while frame to manage do
    Read frame and put in input queue;
    Workers take frame to perform detection;
    Frame + detection put in output queue;
    Output stream from output queue;
end
```

Docker

- ∼VM: Portabilité, stabilité
- Image: `docker build Dockerfile -t <ImageName>`
- Conteneur: `docker run <ImageName>`
- Difficulté: envoyer et récupérer le flux vidéo
- Linux: webcam `/dev/video0`
- partager un device avec image docker:
  `docker run –device=/dev/video0`

Docker

```
xhost +local:docker

XSOCK=/tmp/.X11-unix

XAUTH=/tmp/.docker.xauth

xauth nlist $DISPLAY | sed -e 's ..../ffff/' | xauth -f $XAUTH nmerge -

docker run -m 8GB -it –rm –device=/dev/video0 -e DISPLAY=$DISPLAY -v $XSOCK:$XSOCK -v
$XAUTH:$XAUTH -e XAUTHORITY=$XAUTH -v $PWD:/lab realtime-objectdetection

xhost -local:docker
```

Ressources

Blog/tutoriels:

- Building a Real-Time Object Recognition App with Tensorflow and OpenCV (Dat Tran)
- Increasing webcam FPS with Python and OpenCV (Adrian Rosebrock)
- Real-time and video processing object detection using Tensorflow, OpenCV and Docker.

Repository git du projet: (Contributions bienvenues!)

- Object-detection

agaetis

Big Data & Data Science

Merci !

Des questions?

Léo Beaucourt