

Réseaux de neurones profonds

Catherine ACHARD

Institut des Systèmes Intelligents et de
Robotique

catherine.achard@sorbonne-universite.fr

*

Références

<http://cs231n.github.io/>

<http://www.yaronhadad.com/deep-learning-most-amazing-applications>

<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

<http://yann.lecun.com/exdb/lenet/index.html>

<https://blog.octo.com/classification-dimages-les-reseaux-de-neurones-convolutifs-en-toute-simplicité/>

<http://www.deeplearningbook.org/>

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

<https://deeplearning4j.org/>

Les applications

* Applications

<http://www.yaronhadad.com/deep-learning-most-amazing-applications/>

Classification

- But : L'algorithme doit dire à quelle catégorie appartient une entrée
- Exemple : L'image d'une personne → son identité

Régression

- But : L'algorithme doit associer une valeur numérique à une entrée
- Exemple : Paramètres météorologique → température à 24 heures

Retranscription

- But : A partir de l'observation d'une entrée, la transcrire sous une forme textuelle
- Exemple : A partir d'images de google street, donner les numéros de rue

Traduction

- But : convertir une séquence de symbole en une autre séquence de symbole dans un autre langage
- Exemple : traduction du français à l'anglais

Détection d'anomalie

- But : Trouver si une entrée est atypique ou non
- Exemple : détecter des voitures en sens contraire sur des images routières

Synthèse

- But : apprendre à générer de nouveaux exemples proches de ceux d'une base de données
- Application : Synthétiser des visages sous un autre point de vue

Débruitage

- But : Apprendre à générer des données non bruitées à partir de données bruitées

* Applications

<http://www.yaronhadad.com/deep-learning-most-amazing-applications/>

Re-colorier des images ou vidéos noir et blanc



Iizuka, S., Simo-Serra, E., & Ishikawa, H. (2016). Let there be color!: joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics*

* Applications

<http://www.yaronhadad.com/deep-learning-most-amazing-applications/>

Super-résolution : augmentation d'un facteur 3



Image originale



Interpolation bi-cubique

Dong, C., Loy, C. C., He, K., & Tang, X. (2014, September). Learning a deep convolutional network for image super-resolution. In *European Conference on Computer Vision* (pp. 184-199). Springer, Cham.



Apprentissage (deep)

* Applications

<http://www.yaronhadad.com/deep-learning-most-amazing-applications/>

Génération automatique d'écriture manuscrite

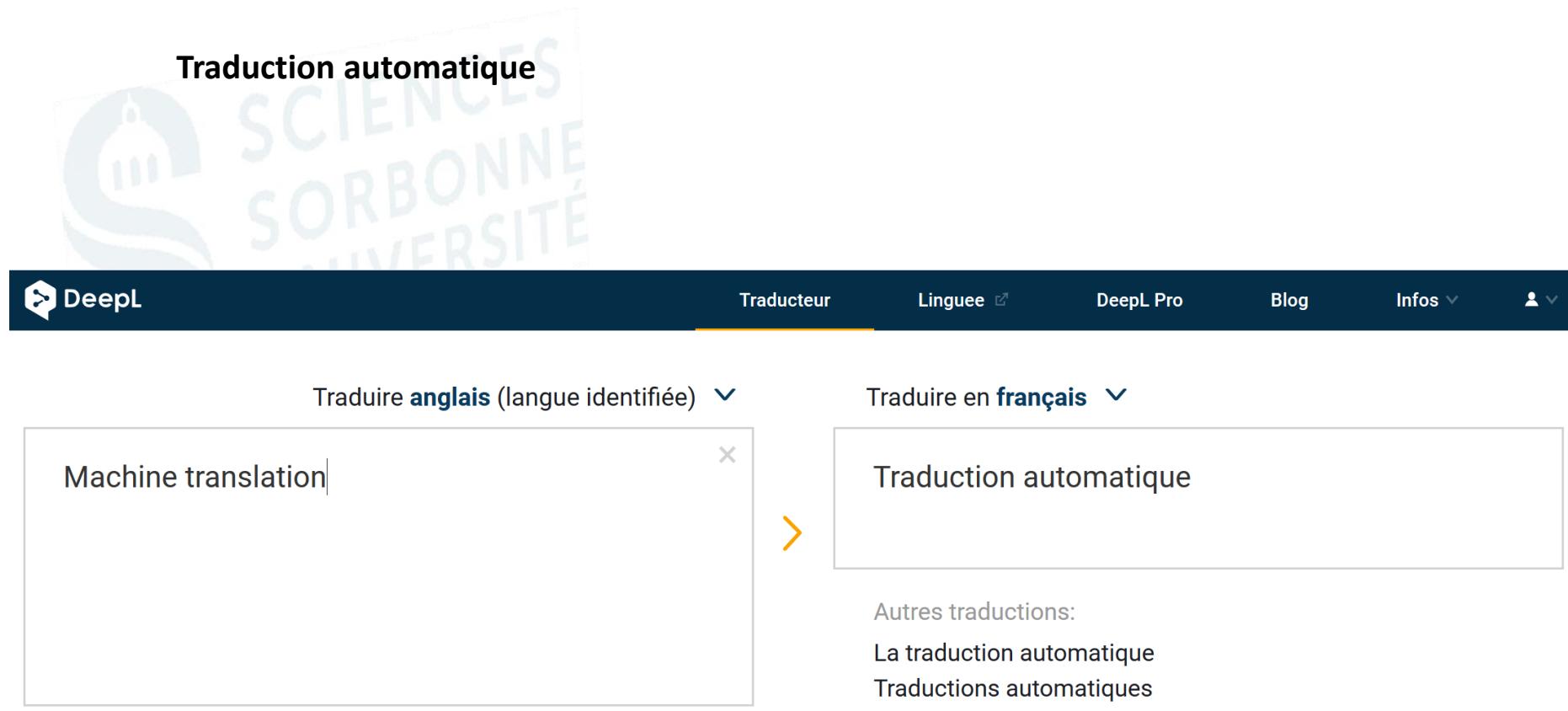
more of national temperament
more of national temperament

Figure 15: **Real and generated handwriting.** The top line in each block is real, the rest are unbiased samples from the synthesis network. The two texts are from the validation set and were not seen during training.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

* Applications

<http://www.yaronhadad.com/deep-learning-most-amazing-applications/>



The screenshot shows the DeepL translation interface. At the top, there's a navigation bar with the DeepL logo, language selection dropdowns ('Traduire anglais' and 'Traduire en français'), and links for 'Traducteur', 'Linguee', 'DeepL Pro', 'Blog', 'Infos', and user profile. Below the bar, two main tabs are visible: 'Machine translation' (selected) and 'Traduction automatique'. Each tab has a dropdown menu showing its name and other related terms like 'Autres traductions'.

Selected Tab	Target Language	Other Options
Machine translation	anglais	Autres traductions: La traduction automatique Traductions automatiques
Traduction automatique	français	Autres traductions: Machine translation Traductions automatiques

* Applications

<http://www.yaronhadad.com/deep-learning-most-amazing-applications/>

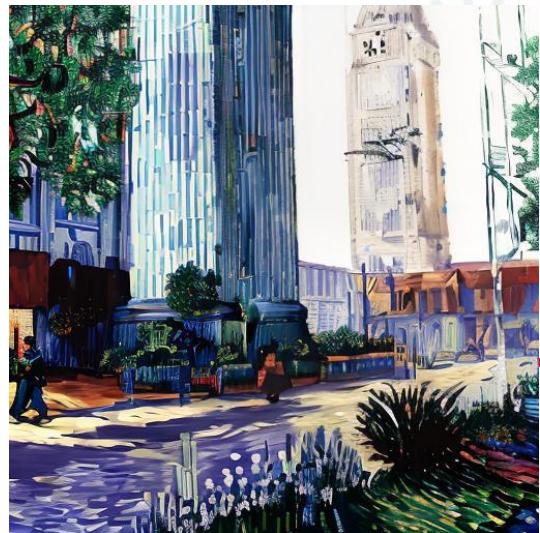
La publicité

- accroître la pertinence des publicités
- créer de la publicité prédictive axée sur les données
- faire des enchères en temps réel pour les publicités
- publicité d'affichage ciblée

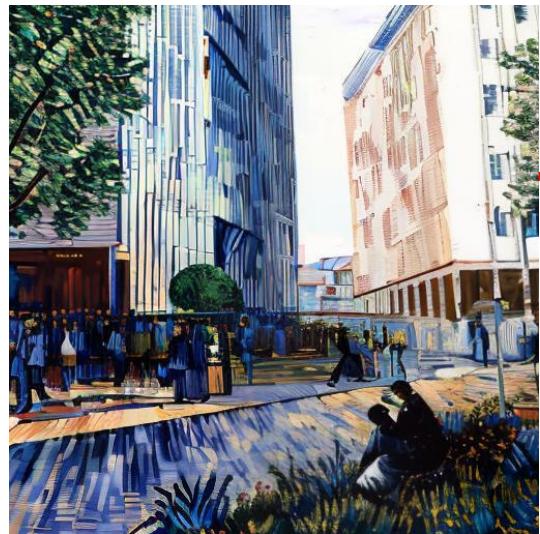
* Applications

<http://www.yaronhadad.com/deep-learning-most-amazing-applications/>

La génération d'images



Style Van Googh



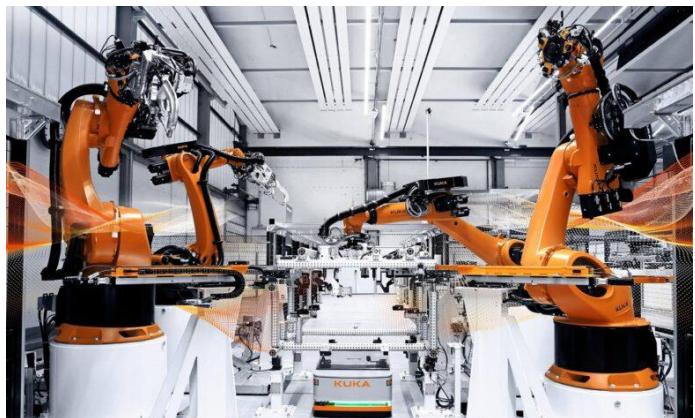


Et pour la robotique ?



Robotique traditionnelle

- Peu adaptative
- Simple automate qui reproduit des tâches



<https://www.robot-magazine.fr/le-marche-en-pleine-croissance-des-robots-industriels-et-de-lautomatisation>

Robotique collaborative ou cobot

- Réduit les risques de TMS et éliminer la manipulation de charges lourdes
- Améliore l'ergonomie d'un poste de travail
- Rend du personnel disponible pour des opérations à plus forte valeur ajoutée
- Favorise l'expertise et le savoir-faire humain
- **Adaptative et interactive**
- **Réagit à l'environnement**



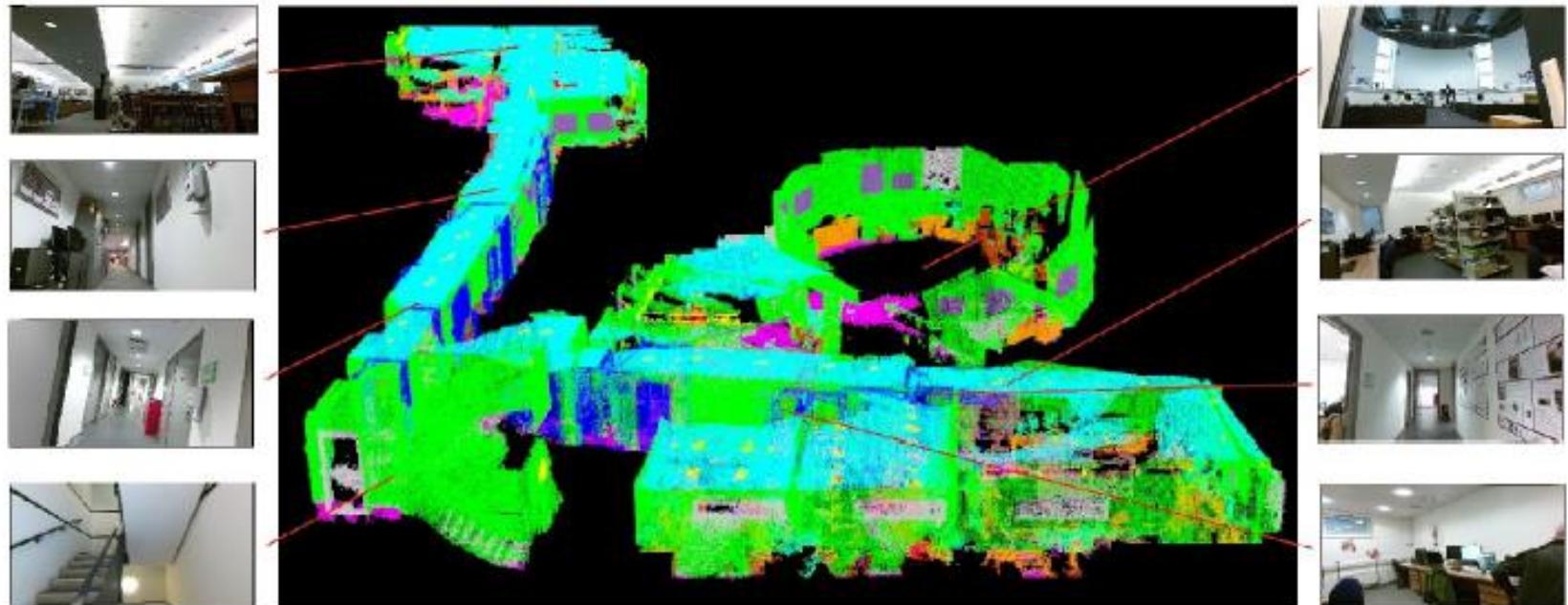
<https://www.humarobotics.com/industries-faites-le-choix-de-la-robotique-collaborative/>

Et en robotique, pourquoi de l'intelligence ?

- **Pour percevoir l'environnement**
 - détecter des objets ou des humains
 - se localiser, estimer la pose
 - estimer la profondeur
 - suivre des mouvement
 - Segmenter la scène (robotique chirurgicale)
- **Pour prédire le futur et donc anticiper**
 - Quelle trajectoire va prendre la voiture devant moi, le piéton sur le trottoir, quel geste va réaliser un humain en collaboration avec un robot
- **Pour planifier des actions**
 - Comment réagir suite à une situation
- **Pour interpréter des comportements humains, de voiture, ...**
 - Reconnaissance d'actions pour la collaboration avec un robot, de situation normale ou anormale, d'émotion, du langage
- **Pour générer des mouvements (avatar, véhicule, bras articulé)**
 - Avoir un comportement interactif plutôt que de suivre un ensemble de règle, modéliser les interactions sociales
- **Apprendre des dynamiques complexes de grande dimension**
 - s'adapter rapidement à de nouvelles dynamiques pour de la saisie de nouveaux objets, le déplacement sur des surfaces inconnues, la gestion des interactions, l'adaptation à la défaillance
- **La localisation d'objet**
 - Pick and place

Utilisation du deep learning en robotique

SLAM: Simultaneous Localization and Mapping

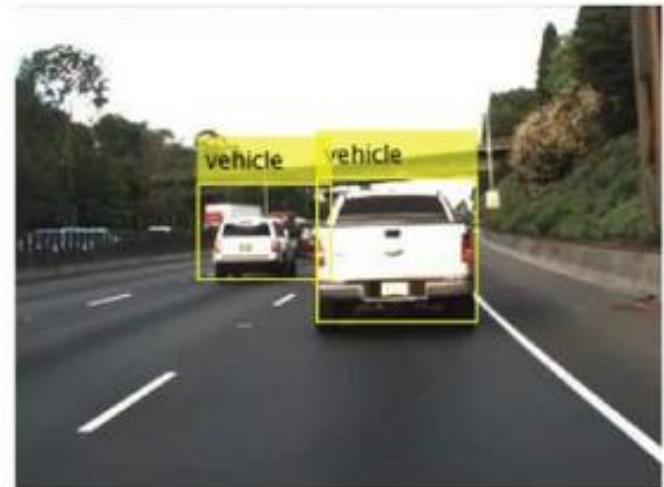


Utilisation du deep learning en robotique

Détection d'objets



OBJECT DETECTION
ALGORITHM

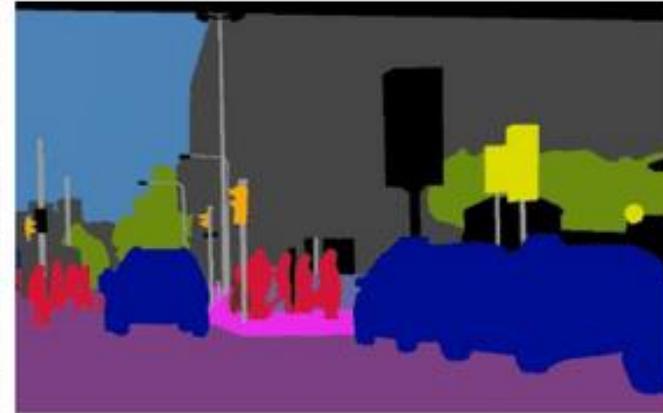


<https://www.superannotate.com/blog/object-detection-with-deep-learning>

Segmentation d'images



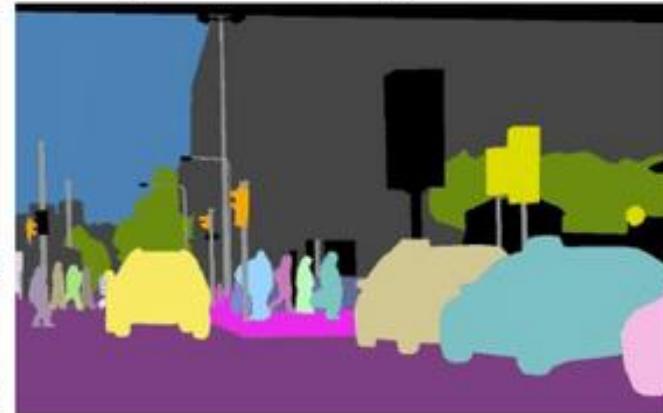
(a) Image



(b) Semantic Segmentation

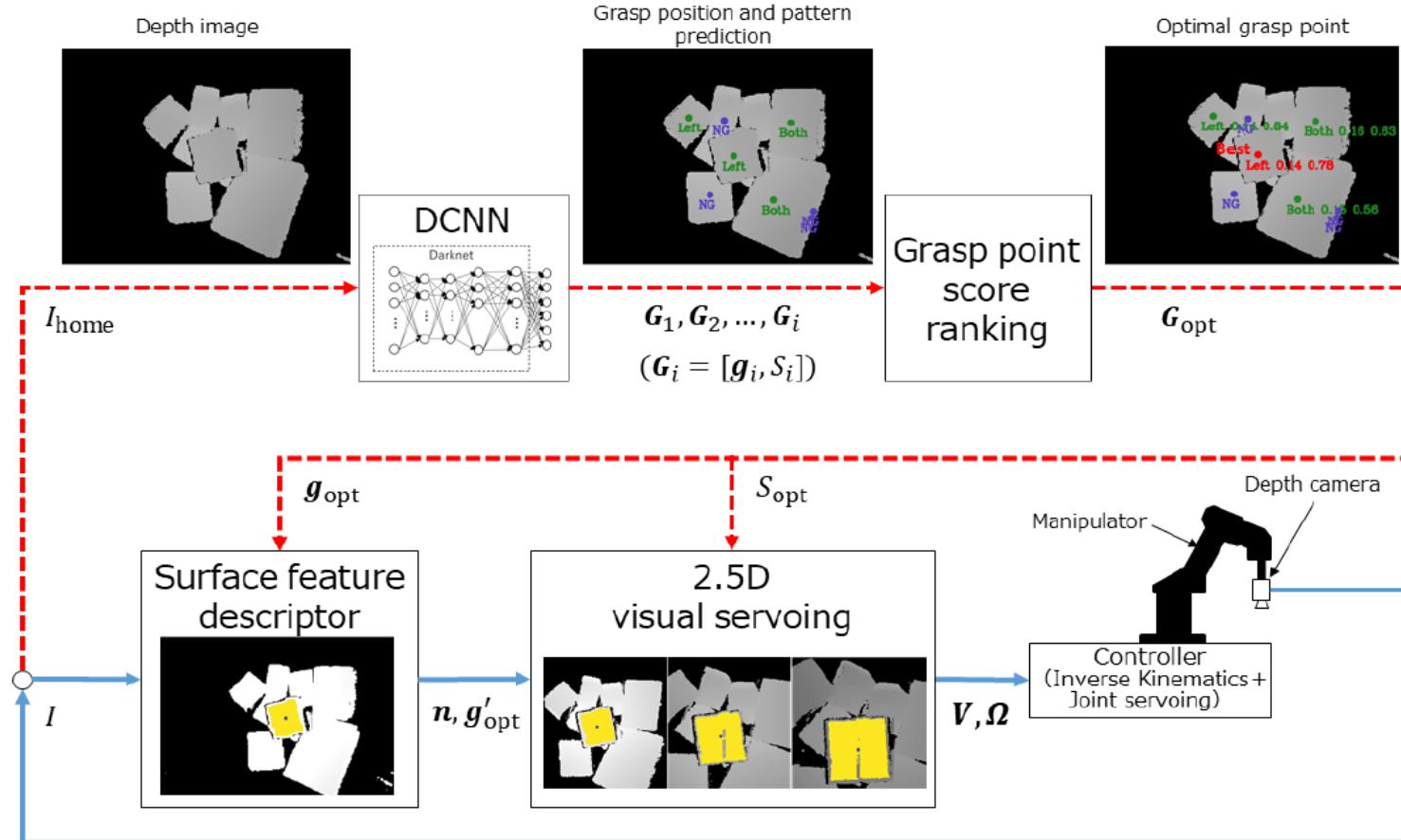


(c) Instance Segmentation



(d) Panoptic Segmentation

Robotic Bin-Picking



Jiang, P., Ishihara, Y., Sugiyama, N., Oaki, J., Tokura, S., Sugahara, A., & Ogawa, A. (2020). Depth image–based deep learning of grasp planning for textureless planar-faced objects in vision-guided robotic bin-picking. *Sensors*, 20(3), 706.

Utilisation du deep learning en robotique

Estimation de la profondeur



Input



Monodepth2 (M)

Godard, C., Mac Aodha, O., Firman, M., & Brostow, G. J. (2019). Digging into self-supervised monocular depth estimation. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 3828-3838).

6 DOF Camera relocalization (PosNet)



Figure 1: **PoseNet: Convolutional neural network monocular camera relocalization.** Relocalization results for an input image (top), the predicted camera pose of a visual reconstruction (middle), shown again overlaid in red on the original image (bottom). Our system relocalizes to within approximately $2m$ and 6° for large outdoor scenes spanning $50,000m^2$. For an online demonstration, please see our project webpage: mi.eng.cam.ac.uk/projects/relocalisation/

Kendall, A., Grimes, M., & Cipolla, R. (2015). Posenet: A convolutional network for real-time 6-dof camera relocalization. In Proceedings of the IEEE international conference on computer vision (pp. 2938-2946).

*Utilisation du deep learning en robotique

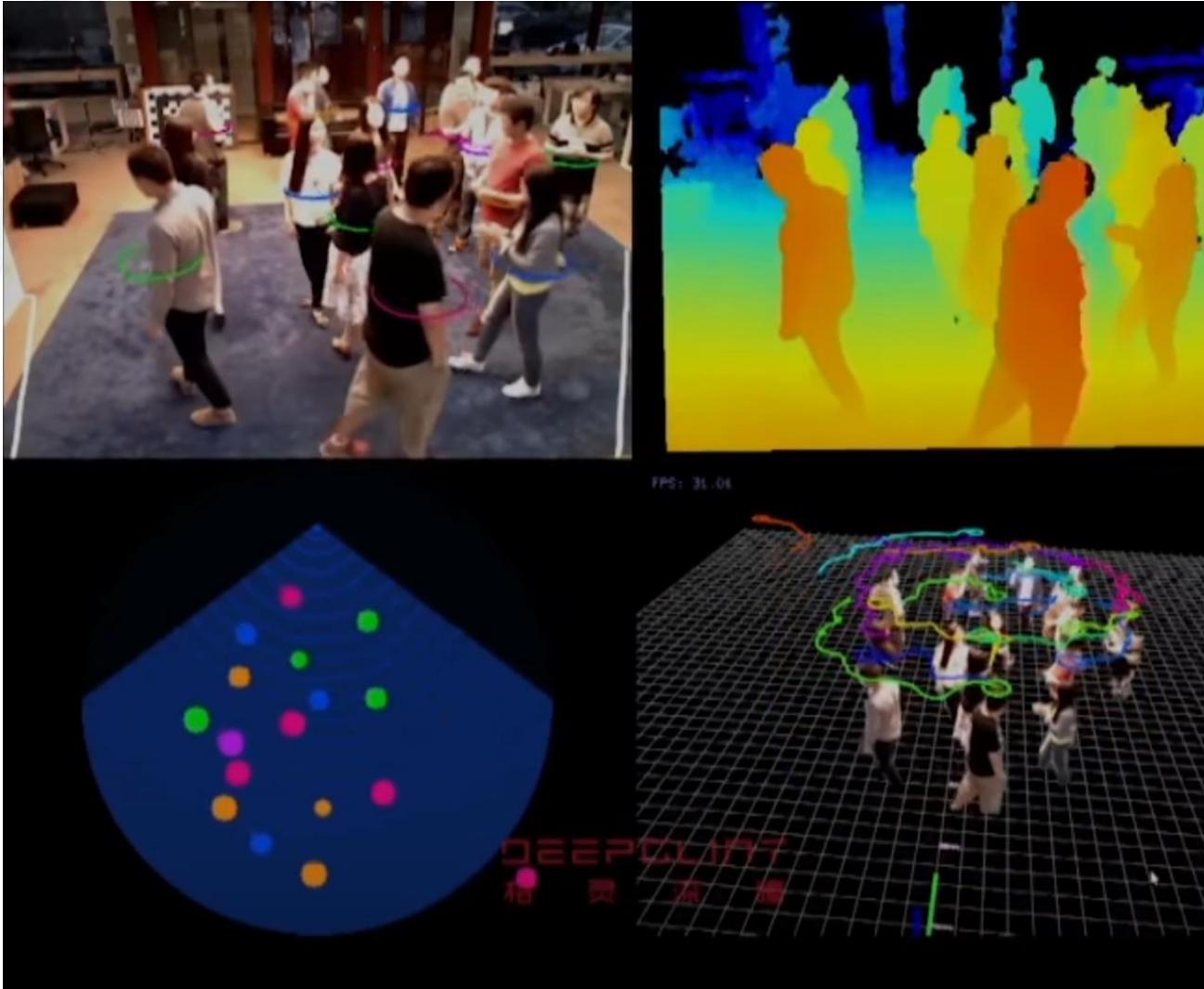
Estimation de la pose multi-personnes temps réel



Cao, Z., Simon, T., Wei, S. E., & Sheikh, Y. (2017, July). Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR* (Vol. 1, No. 2, p. 7).

Même en 3D maintenant, à partir d'images RGB !

Analyse temps réel du comportement



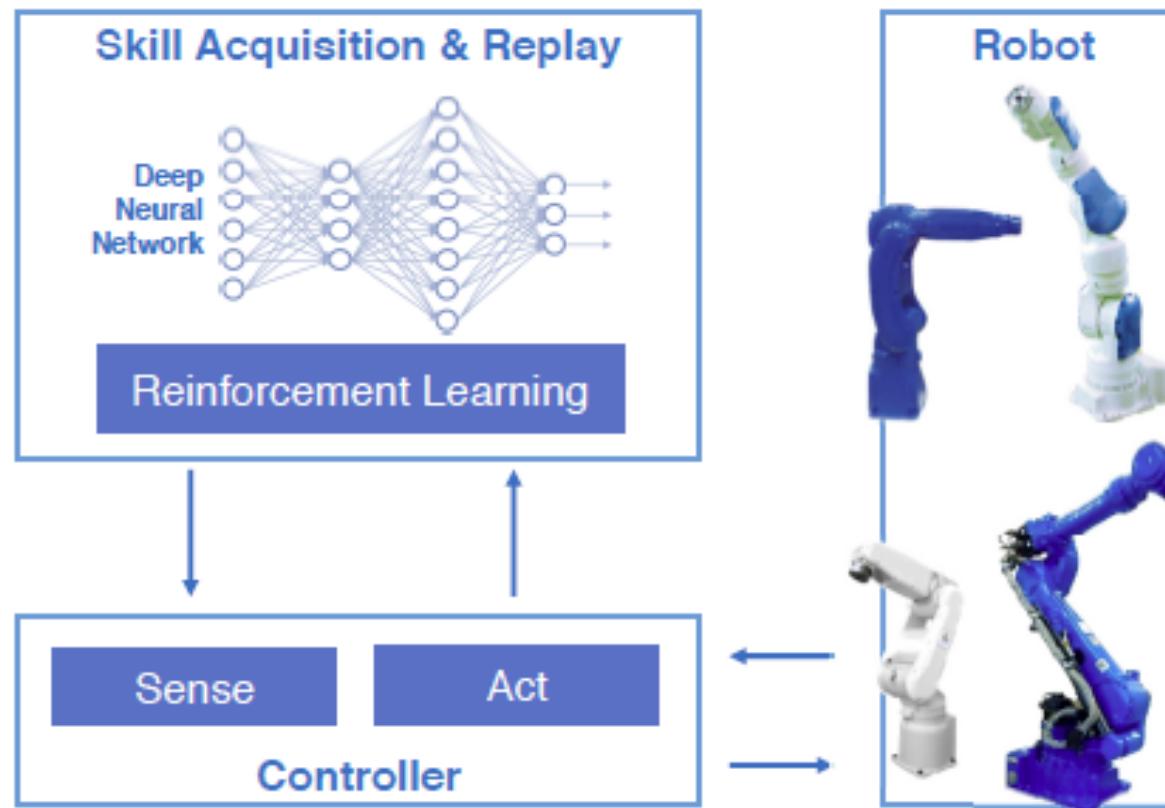
Robotique sociale



<https://www.evenement.com/animations/animation-evenementielle-rencontre-robotique-sociale/>

*Utilisation du deep learning en robotique

Contrôle



Inoue, T., De Magistris, G., Munawar, A., Yokoya, T., & Tachibana, R. (2017, September). Deep reinforcement learning for high precision assembly tasks. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 819-825). IEEE.



Le perceptron



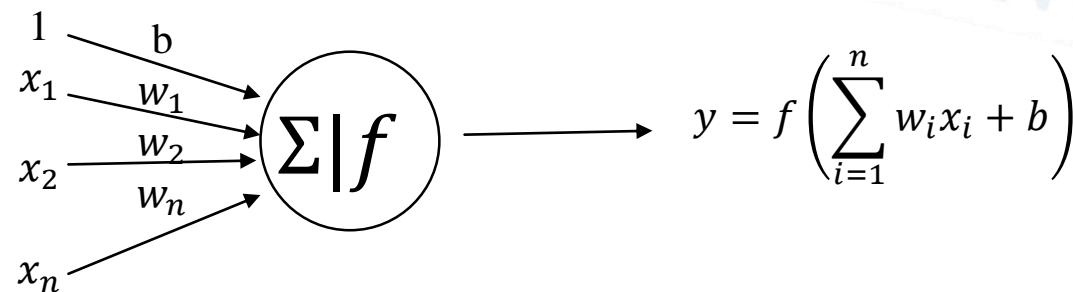
$x = [x_1, \dots, x_n]^T$: signal d'entrée

Un neurone est défini par :

$$y = f\left(\sum_{i=1}^n w_i x_i + b\right)$$

Avec

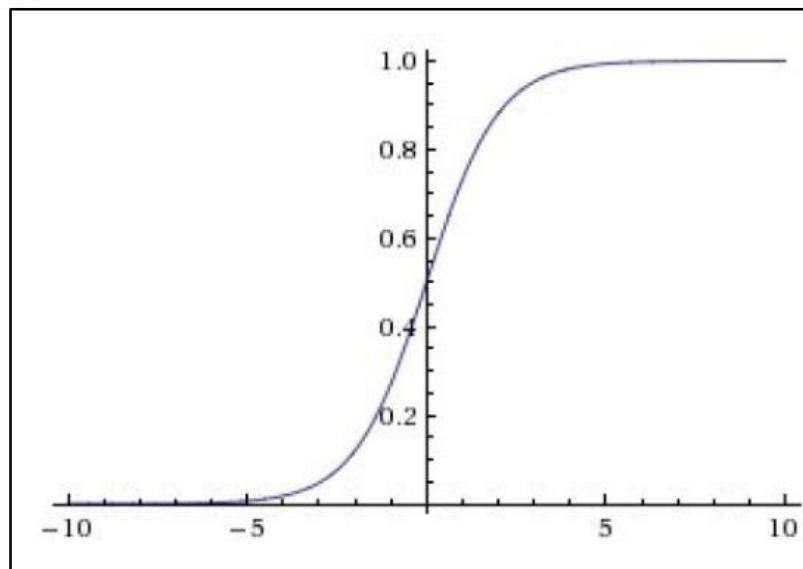
- w_i les poids
- b le biais
- $a = \sum_{i=1}^n w_i x_i + b$ activation
- f la fonction d'activation, fonction sigmoïde par exemple



Différentes fonctions d'activations possibles

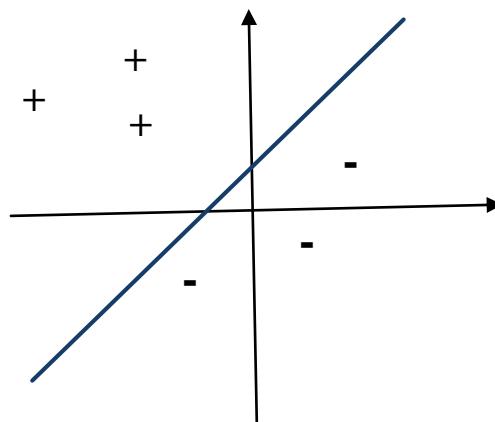
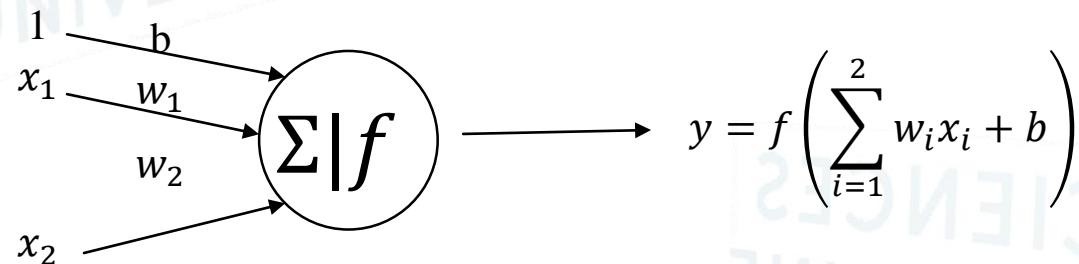
Sigmoïde

- $f(x) = \frac{1}{1+e^{-\lambda x}}$



Apprentissage

En choisissant judicieusement les poids w_i et b , le neurone peut séparer les données linéairement séparables (2 classes)

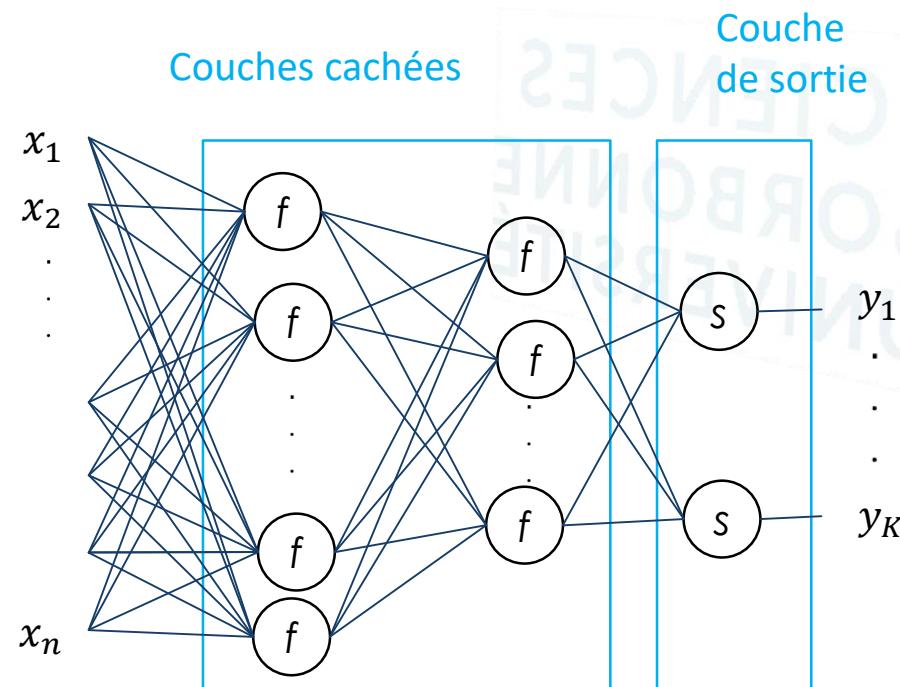


Pour un problème non linéaire, on utilise des couches cachées

Il s'agit du **perceptron multicouches (multilayer perceptron MLP)**

Deux types de couches :

- Une ou plusieurs **couches cachées** composées de plusieurs neurones
- Une **couche de sortie** composée de K neurones $\mathbf{y} = [y_1, \dots, y_K]^T$



Le perceptron multi-couches

*Les réseaux de neurones

En régression

- sortie continue
- la dernière couche a une fonction d'activation linéaire

En classification à K classes

One Hot Encoding : la dernière couche a K neurones.

Pour un exemple t de la classe k ,

- $y_{t,k} = 1$
- $y_{t,k' \neq k} = 0$
- $y_{t,k}$: valeur du k ème neurone de sortie pour l'exemple t

La dernière couche a une fonction d'activation **softmax** qui transforme la sortie du réseau en probabilités d'appartenance aux classes :

$$y_{t,k} = \frac{\exp(y_{t,k})}{\sum_{k'} \exp(y_{t,k'})}$$

*Les réseaux de neurones

Apprentissage : trouver les poids du réseau pour que les sorties soient le plus proches possible des sorties désirées.

Minimisation d'une fonction appelée **fonction perte** ou **loss function** :

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} \sum_{t=1}^N l_{\mathbf{w}}(\mathbf{y}_t, \hat{\mathbf{y}}_t)$$

Où \mathbf{w} : vecteur contenant tous les poids et biais du réseau,

\mathbf{y}_t : sortie désirée pour l'exemple x_t

$\hat{\mathbf{y}}_t$: sortie prédite par le réseau

N : nombre d'exemples d'apprentissage

En **régression, erreur quadratique** (MSE : Mean Square Error) :

$$l_{\mathbf{w}}(\mathbf{y}_t, \hat{\mathbf{y}}_t) = \|\mathbf{y}_t - \hat{\mathbf{y}}_t\|^2$$

En **classification à K classes, entropie croisée** (après le softmax) :

$$l_{\mathbf{w}}(\mathbf{y}_t, \hat{\mathbf{y}}_t) = - \sum_{k=1}^K y_{t,k} \log \hat{y}_{t,k}$$

Exercice

Considérons un problème de classification à 3 classes ($K=3$).

On considère 3 exemples appartenant à la classe 1 dont la sortie du réseau,

avant le softmax, vaut : $\begin{pmatrix} 0.9 \\ 0.2 \\ 0.2 \end{pmatrix}$ $\begin{pmatrix} 0.2 \\ 0.3 \\ 0.2 \end{pmatrix}$ $\begin{pmatrix} 0.2 \\ 0.9 \\ 0.2 \end{pmatrix}$

Déterminer la valeur de l'entropie croisée pour ces 3 exemples.

*Les réseaux de neurones

Exercice

Considérons un problème de classification à 3 classes ($K=3$).

On considère 3 exemples appartenant à la classe 1 dont la sortie du réseau,

avant le softmax, vaut : $\begin{pmatrix} 0.9 \\ 0.2 \\ 0.2 \end{pmatrix} \quad \begin{pmatrix} 0.2 \\ 0.3 \\ 0.2 \end{pmatrix} \quad \begin{pmatrix} 0.2 \\ 0.9 \\ 0.2 \end{pmatrix}$

Déterminer la valeur de l'entropie croisée pour ces 3 exemples.

Il faut d'abord faire le softmax. Les 3 vecteurs deviennent:

$$\begin{pmatrix} 0.5017 \\ 0.2491 \\ 0.2491 \end{pmatrix} \quad \begin{pmatrix} 0.3220 \\ 0.3559 \\ 0.3220 \end{pmatrix} \quad \begin{pmatrix} 0.2491 \\ 0.5017 \\ 0.2491 \end{pmatrix}$$

Puis $l_w(\mathbf{y}_t, \widehat{\mathbf{y}}_t) = -\sum_{k=1}^K y_{t,k} \log \widehat{y}_{t,k}$

$$l_{w1} = -1 \times \log(0,5017) - 0 \times \log(0,2491) - 0 \times \log(0,2491) = 0,30$$

$$l_{w2} = -1 \times \log(0,3220) - 0 \times \log(0,3559) - 0 \times \log(0,3220) = 0,49$$

$$l_{w3} = -1 \times \log(0,2491) - 0 \times \log(0,5017) - 0 \times \log(0,2491) = 0,60$$

- ➔ Le dernier exemple amène à une erreur plus forte.
- ➔ La fonction perte que l'on cherche à minimiser est la somme des entropies croisées : on veut que tous les exemples soient bien classés

*Les réseaux de neurones

Minimisation d'une fonction par descente de gradient

→ il faut que la fonction perte soit dérivable, dépendant des poids

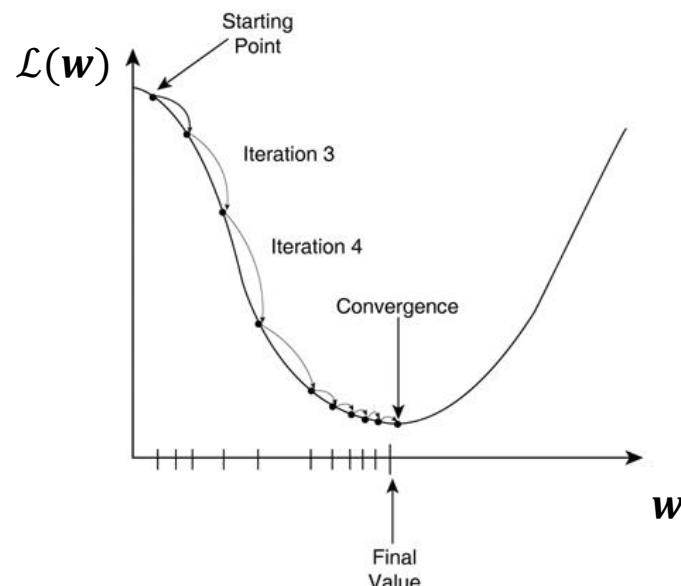
Raisonnons sur une fonction générale $\mathcal{L}(\mathbf{w})$

Partant de poids initiaux w_0 , le minimum de $\mathcal{L}(W)$ s'obtient par itération avec :

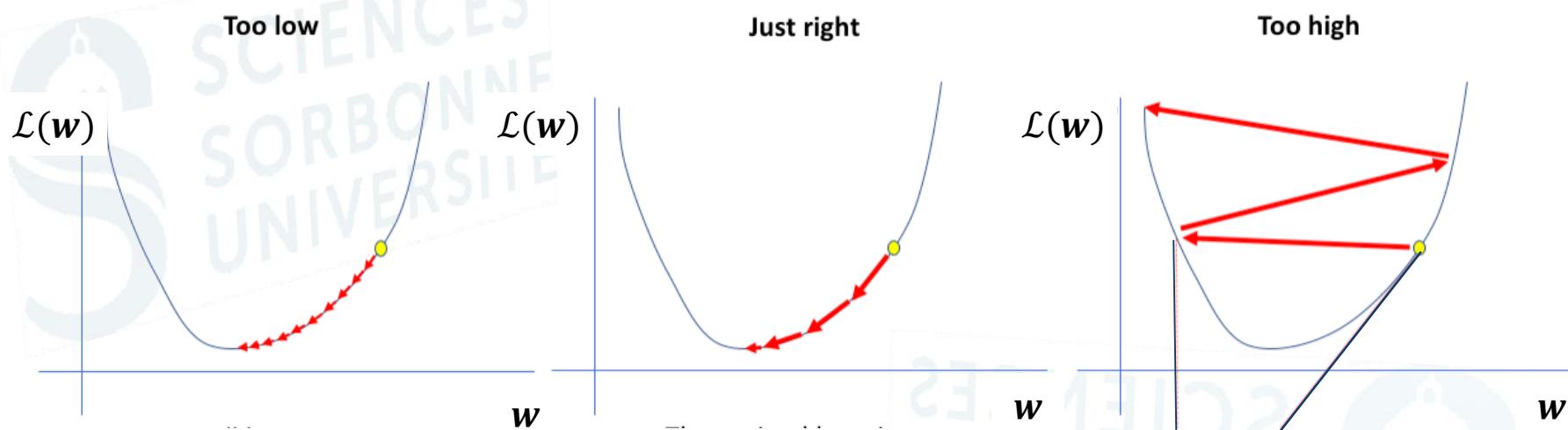
$$\mathbf{w}_{it} = \mathbf{w}_{it-1} - \lambda \nabla \mathcal{L}(\mathbf{w}_{it-1})$$

Où

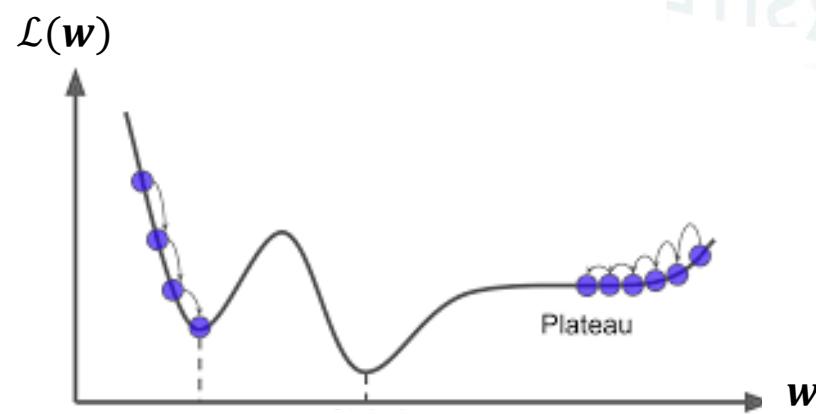
- $\nabla \mathcal{L}(\mathbf{w})$ est le gradient de \mathcal{L} par rapport aux poids \mathbf{w}
- λ est le pas d'apprentissage.



La convergence dépend du pas d'apprentissage



Et du point initial

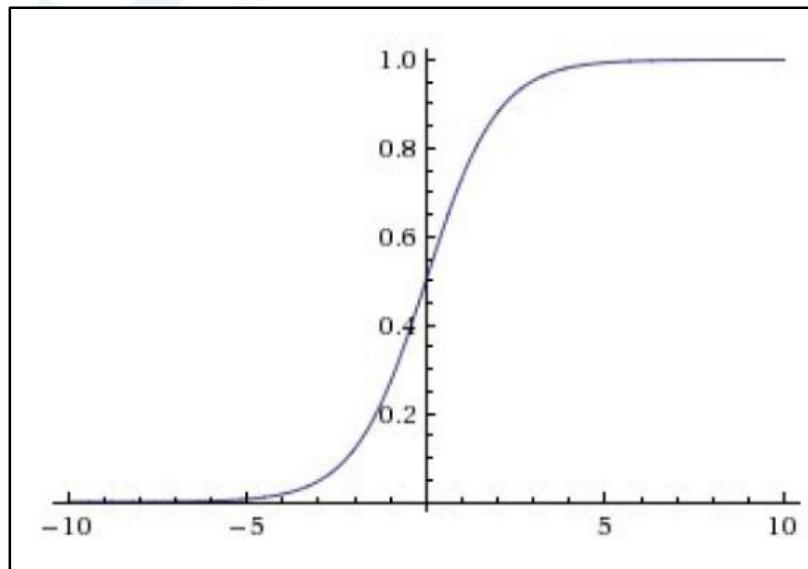


*Les réseaux de neurones

Différentes fonctions d'activations possibles

Sigmoïde

- $f(x) = \frac{1}{1+e^{-\lambda x}}$
- $\frac{\partial f(x)}{\partial x} = \frac{\lambda e^{-\lambda x}}{(1+e^{-\lambda x})^2} = f(x)(1 - f(x))$

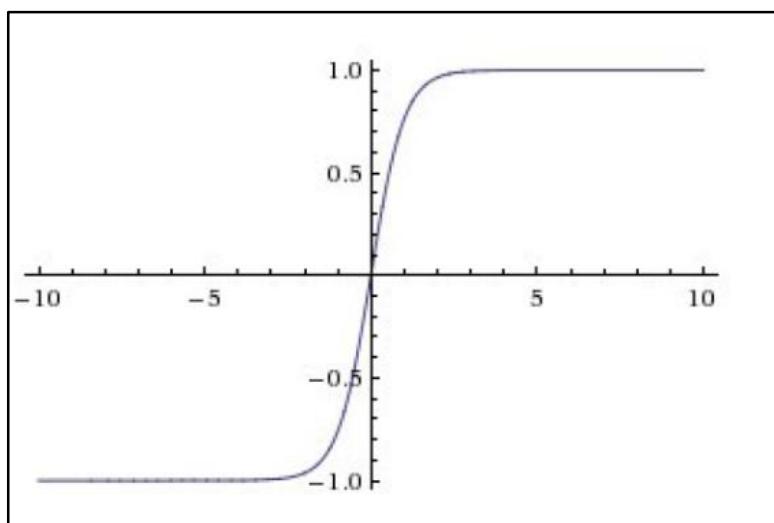


Deux inconvénients

- Sature et tue les gradients pour de fortes activations du neurones
- La sortie n'est pas centrée

Tangente hyperbolique

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$\frac{\partial f(x)}{\partial x} = 1 - f^2(x)$$



Inconvénient

- Sature et tue les gradients pour de fortes activations du neurones

Avantage

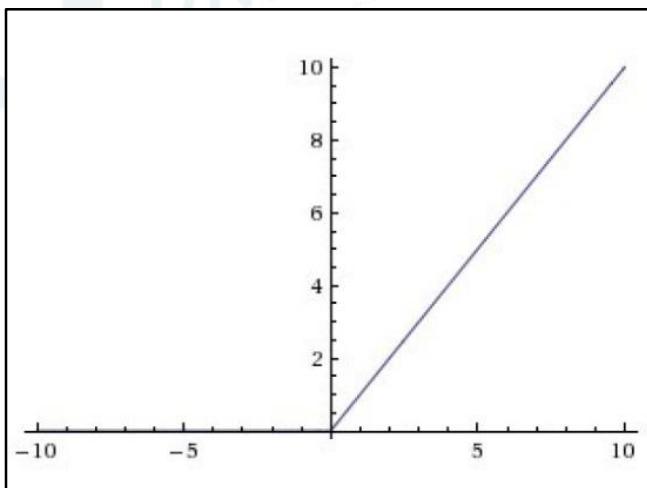
- La sortie est centrée

*Les réseaux de neurones

RELU (REctified Linear Unit)

$$f(x) = \max(0, x)$$

$$\frac{\partial f(x)}{\partial x} = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 & \text{si } x > 0 \end{cases}$$



Avantages

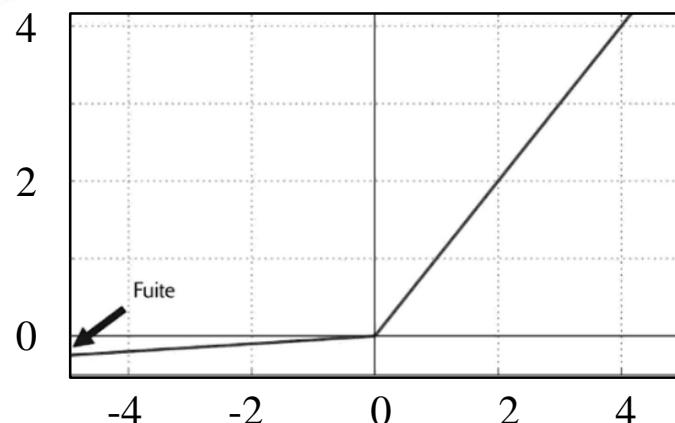
- Très rapide à calculer (pas d'exponentielle)
- Permet au réseau de converger beaucoup plus vite (six fois)
- Forts gradients (0 ou 1)

Inconvénient

- Certains neurones « meurent » en ne produisant que des zéros (les poids sont tels que la sortie est nulle pour toutes les entrées). Comme le gradient devient nul, ils ne se réactivent jamais

LeakyRELU

$$f(x) = \max(\alpha x, x) \text{ avec } \alpha = 0,01$$



Avantages

- Très rapide à calculer (pas d'exponentielle)
- Permet au réseau de converger beaucoup plus vite (six fois)

*Les réseaux de neurones

Pour mettre en place un MLP, l'utilisateur doit donc déterminer

1. les variables d'entrée et la variable de sortie
2. Le nombre de couches cachées et le nombre de neurones de chaque couche (dilemme biais/variance)
3. Le nombre maximum d'itérations
4. Le taux d'apprentissage λ (ou sa stratégie d'évolution)

L'initialisation des poids est aléatoire

*Les réseaux de neurones

Les variables d'entrée et la variable de sortie

Les variables d'entrées doivent être **centrées et normalisées** de manière à bien activer les fonctions d'activation

→ Pour chaque dimension, on retire la moyenne et divise par l'écart-type

ATTENTION : moyenne et écart-type doivent être estimés sur les données d'apprentissage et appliqués ensuite à toute la base (apprentissage et test)

L'initialisation des poids

Pour ne pas saturer les fonctions d'activation, les poids doivent être initialisés à de petites valeurs.

Une solution est de les tirer selon une loi normale centrée, réduite

Le nombre de couches cachées et le nombre de neurones de chaque couche (dilemme biais/variance)

Apprentissage avec 1 couche cachée composée de

3 neurones

6 neurones

20 neurones



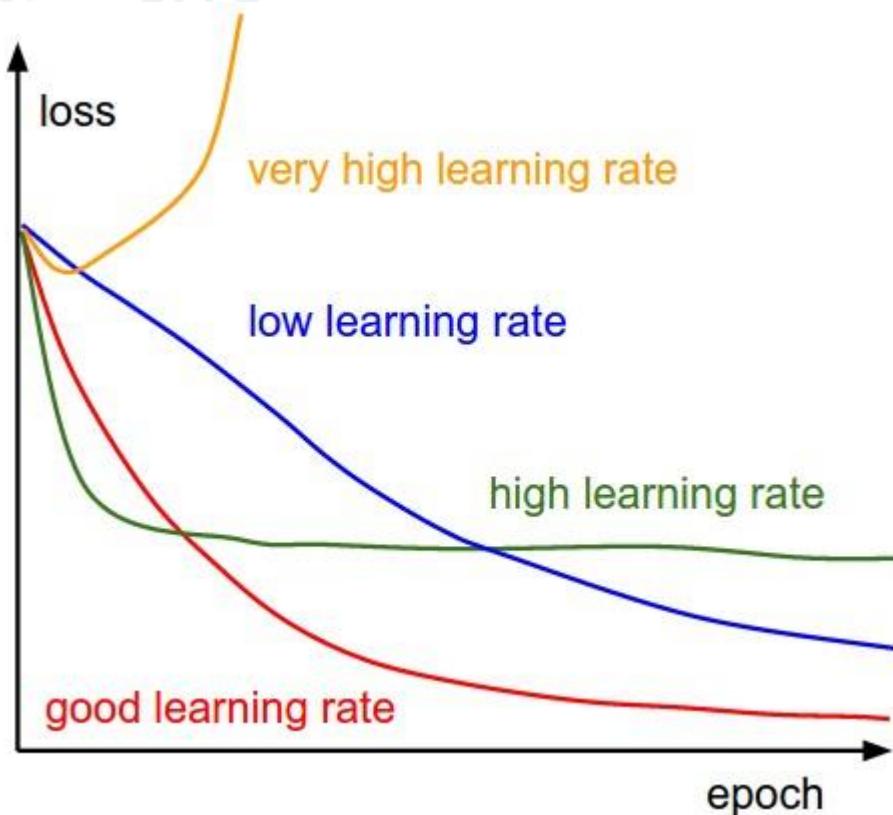
<https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html>

- ➔ Avec plus de neurones, on peut approximer des fonctions plus complexes mais **attention à l'overfitting**
- ➔ Il vaut mieux utiliser des modèles plus complexes avec une régularisation pour éviter l'overfitting (voir plus loin)

Pas d'apprentissage

Le réglage du pas d'apprentissage est important :

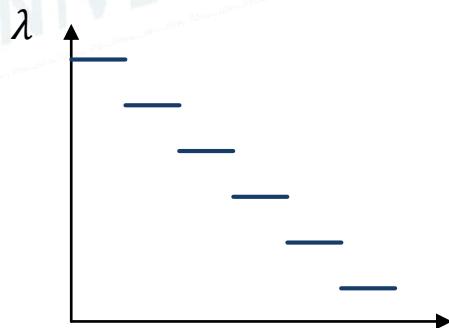
- s'il est trop fort, il peut amener à une divergence
- S'il est trop faible, l'apprentissage aura du mal à converger



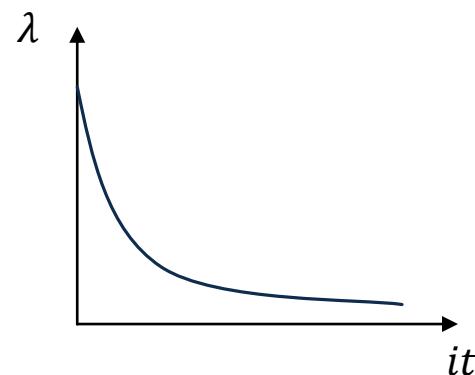
Pas d'apprentissage

Plusieurs solutions

- Pas d'apprentissage constant (**step decay**)
- Diminution par paliers : il diminue d'un pourcentage toutes les T itérations

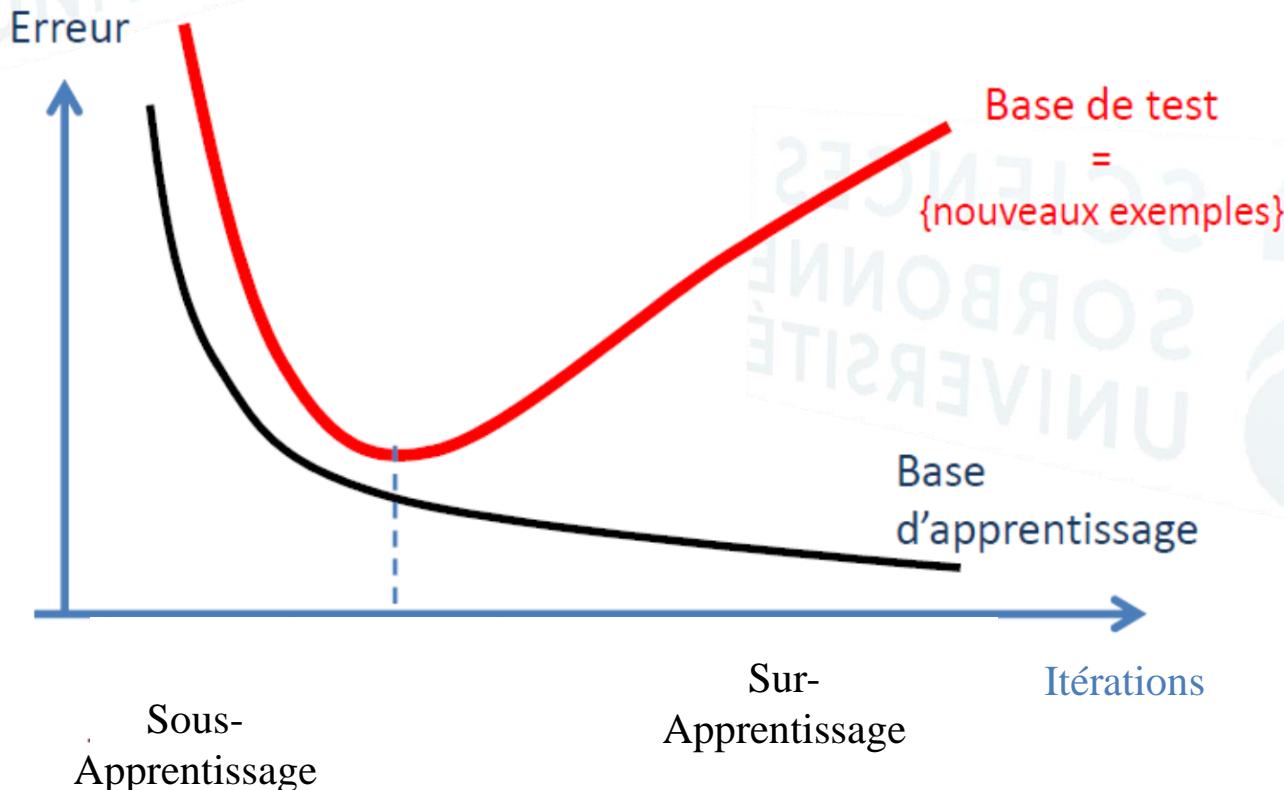


- Diminution exponentielle (**Exponential decay**): $\lambda_{it} = \lambda_0 e^{-\beta it}$



Nombre d'itérations : attention à l'over-fitting !

Solution : base de validation pour arrêter l'apprentissage



*Les réseaux de neurones

Descente de gradient sur la base d'apprentissage

On a vu que la mise à jour des poids est réalisée avec :

$$\mathbf{w}_{it+1} = \mathbf{w}_{it} - \lambda \nabla \mathcal{L}(\mathbf{w}_{it})$$

$$\mathbf{w}_{it+1} = \mathbf{w}_{it} - \lambda \frac{1}{N} \sum_{t=1}^N \nabla l_{\mathbf{w}}(y_t, \hat{y}_t)$$

- Le gradient est calculé pour chaque exemple x_t de la base d'apprentissage mais le modèle est mis à jour que quand tous les exemples sont passés

Avantage

- Erreur et convergence stables

Inconvénient

- La convergence peut être faite dans un minima local
- L'ensemble de la base d'apprentissage doit pouvoir être chargée en mémoire

Descente de gradient stochastique, Stochastic gradient descent (SGD)

Le modèle est mis à jour après chaque exemple d'apprentissage (tiré aléatoirement selon une distribution uniforme)

Avantage :

- Pas besoin d'avoir tous les exemples en mémoire
- Moins de risque de rester dans un minima local

Inconvénient :

- Plus couteux en temps de calcul car il faut faire la mise à jour pour chaque exemple
- La descente de gradient est très bruitée

*Les réseaux de neurones

Descente de gradient par mini-batch

C'est une solution intermédiaire où la base est divisée en mini-batches de taille fixe.
 La mise à jour se fait à la fin de chaque mini-batch:

$$\mathbf{w}_{it+1} = \mathbf{w}_{it} - \lambda \nabla \mathcal{L}(\mathbf{w}_{it})$$

$$\mathbf{w}_{t+1} = \mathbf{w}_{it} - \lambda \frac{1}{N} \sum_{batch} \nabla l_{\mathbf{w}}(\mathbf{y}_t, \widehat{\mathbf{y}_t})$$

En général, entre 32 et 256 exemples par mini-batch (dépend de la dimension des exemples)

Avantage

- Minimise l'erreur de généralisation (les gradients ‘bruités’ agissent comme une régularisation)
- Convergence plus rapide

*Les réseaux de neurones

Terminologie

- **Mini batch ou batch** : sous ensemble de la base d'apprentissage
- **Epoch** : nombre de fois que tous les exemples sont vus en apprentissage
- **Itération** : nombre de batchs vus en apprentissage

Exercice

Considérons une base d'apprentissage composée de 200 exemples,
Nous prenons des batchs de taille 5 et 1000 epochs

- Combien y a-t-il de batchs ?
- Combien t a-t-il d'itérations ?

*Les réseaux de neurones

Terminologie

- **Mini batch ou batch** : sous ensemble de la base d'apprentissage
- **Epoch** : nombre de fois que tous les exemples sont vus en apprentissage
- **Itération** : nombre de batchs vus en apprentissage

Exercice

Considérons une base d'apprentissage composée de 200 exemples,
Nous prenons des batchs de taille 5 et 1000 epochs

- Combien y a-t-il de batchs ?
- Combien t a-t-il d'itérations ?
- 40 batchs
- 40.000 itérations

La batch normalisation

Idée : Stabiliser l'apprentissage des batchs

Pendant l'apprentissage, les batchs changent à chaque itération et peuvent ne pas avoir la même distribution. Les poids du CNN s'adaptent en permanence.

→ On normalise les entrées $x^t = [x_1^t, \dots, x_k^t, \dots, x_n^t]$

$$x_k^t = \frac{x_k^t - \mu_k}{\sigma_k}$$

Avec $\mu_k = \text{mean}_{t=1,\dots,N} x_k^t$ et $\sigma_k = \text{std}_{t=1,\dots,N} x_k^t$

Batch 1



Rose
(y=1)



Not Rose
(y=0)

Batch 2



Rose
(y=1)



Not Rose
(y=0)

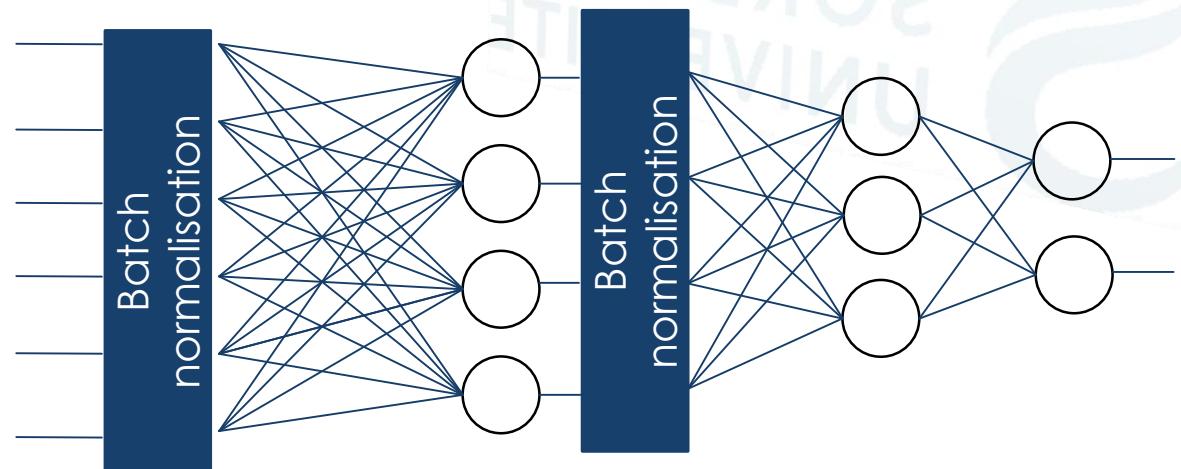
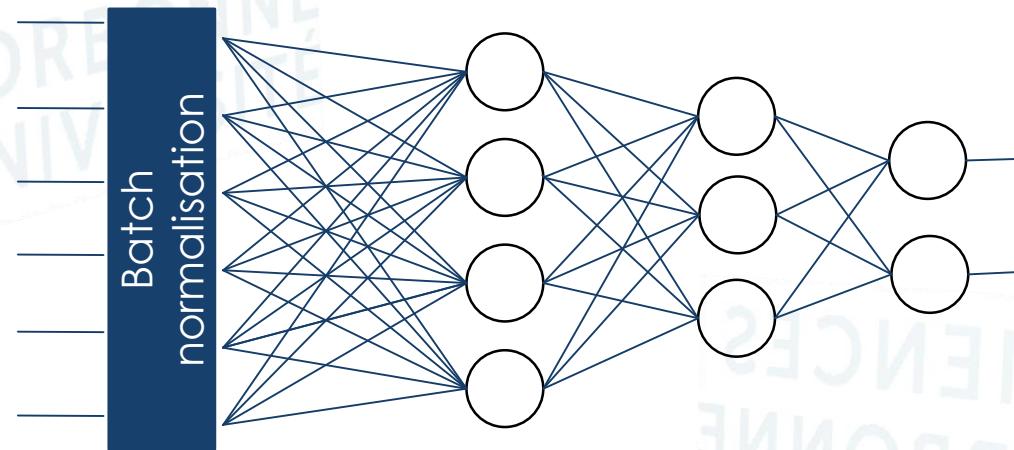
La batch normalisation durant le test

Après l'apprentissage, on utilise toute la base pour estimer des statistiques fiables μ et σ .

On utilise ces statistiques durant l'étape de prédiction

La batch normalisation

Cette normalisation peut être faite sur les entrées de toutes les couches



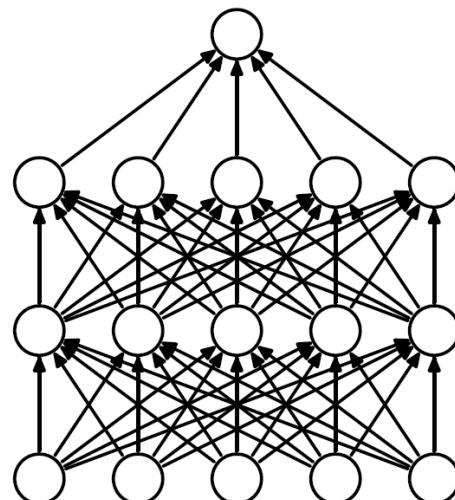
Le dropout

But : éviter le sur-apprentissage

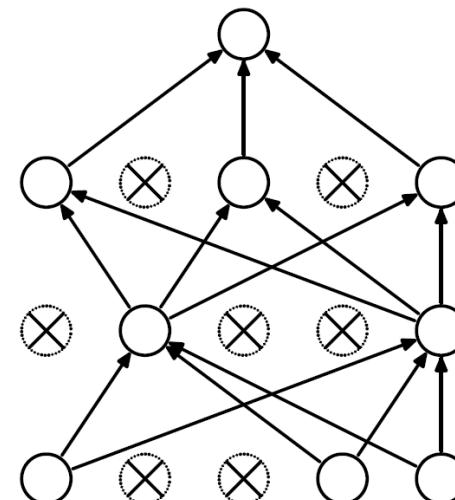
Les réseaux profonds ont un grand nombre de paramètres → sujet au sur apprentissage et à la perte de gradient

Idée du dropout : pour chaque epoch, enlever aléatoirement des neurones lors de l'apprentissage

En test, tous les neurones sont utilisés



(a) Standard Neural Net



(b) After applying dropout.

[Dropout: A Simple Way to Prevent Neural Networks from Overfitting \(2014, N. Srivastava et. al\)](#)

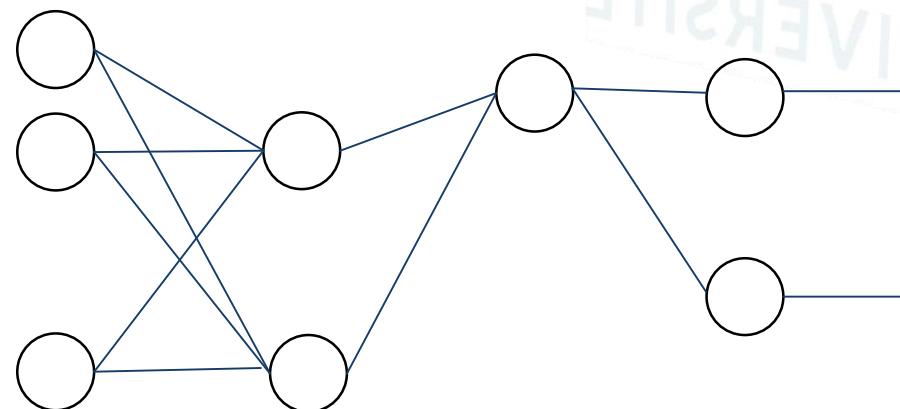
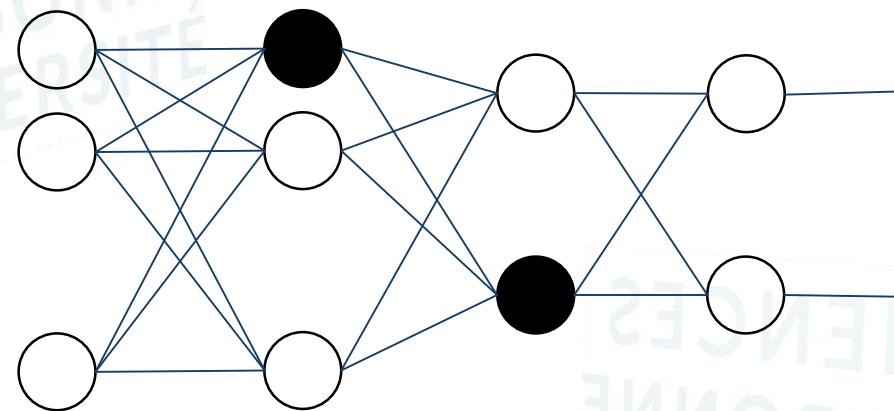
Le dropout

Avantages

- ➔ Les sous-réseaux se focalisent moins sur des détails
- ➔ Les sous-réseaux ont moins de paramètres ➔ évite l'over-fitting
- ➔ Améliore la vitesse d'apprentissage (moins de poids à mettre à jour)

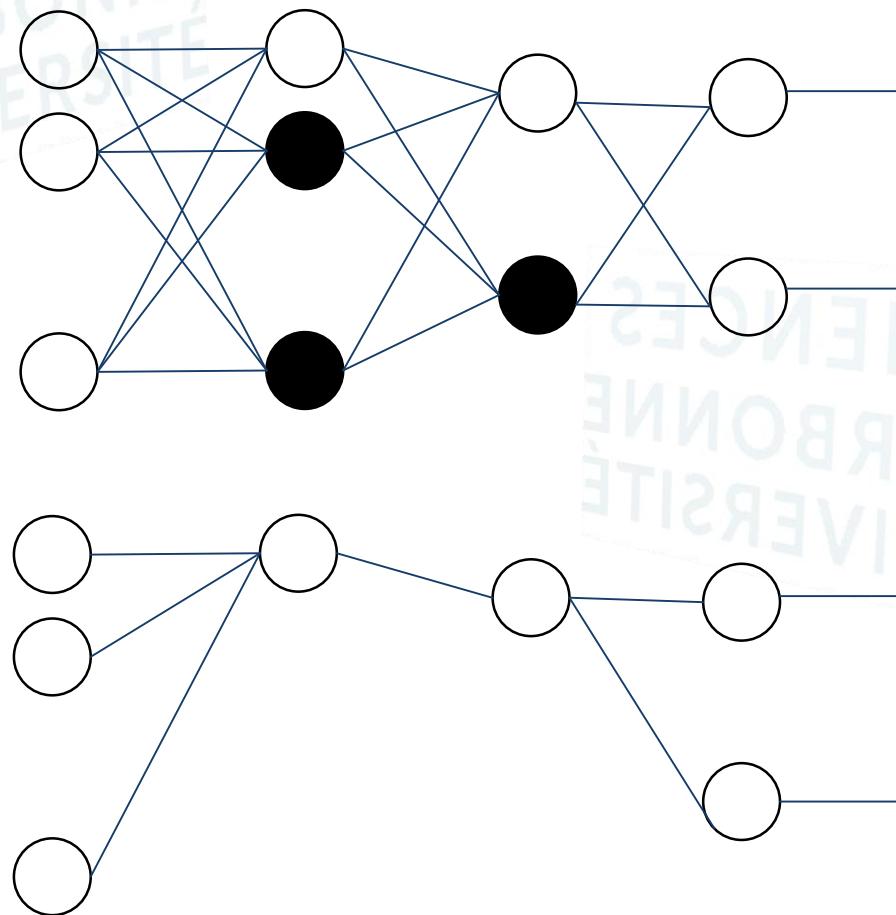
Le dropout

Epoch 1



Le dropout

Epoch 2



*Les réseaux de neurones

La régularisation

But: éviter le sur-apprentissage

Idée : un réseau avec des petits poids est sans doute plus simple qu'un réseau avec des poids importants

- **Régularisation L2**

on pénalise la fonction perte avec la norme L2

$$\mathcal{L} + \frac{1}{2} \alpha \|\mathbf{w}\|^2$$

- **Régularisation 1**

on pénalise la fonction objective avec la norme L1

$$\mathcal{L} + \frac{1}{2} \alpha |\mathbf{w}|$$

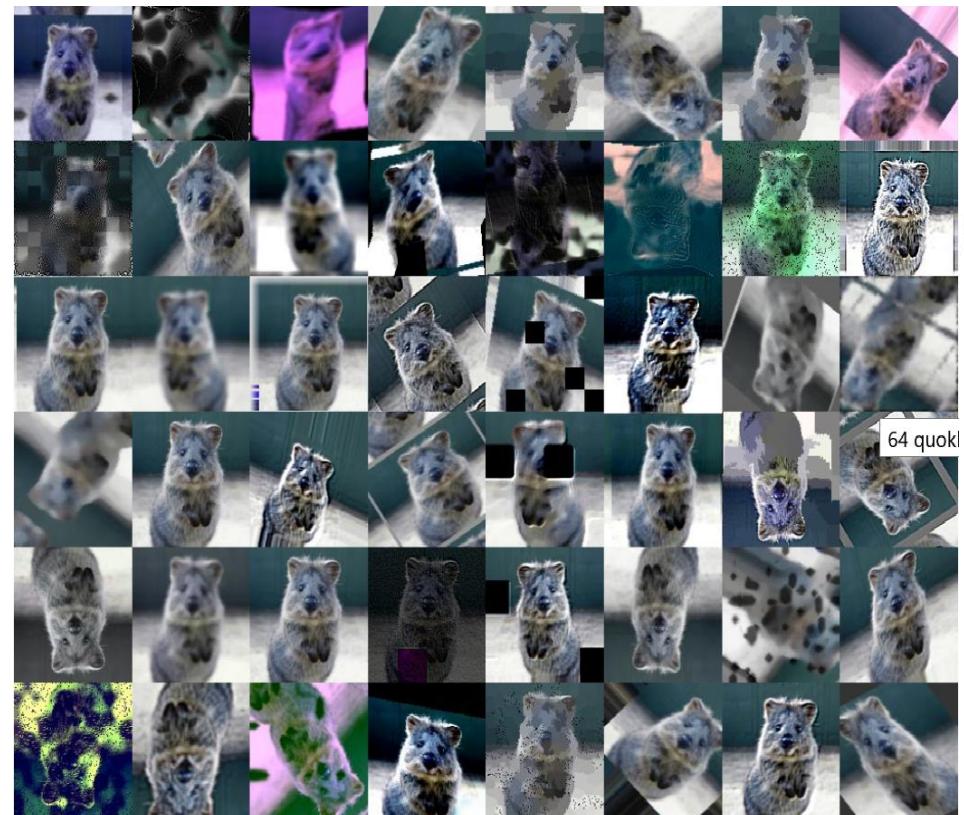
➔ Tend à avoir des poids nuls (modèle sparse) et d'autres très grands

Augmentation des données (data augmentation)

But: éviter le sur-apprentissage

Idée : créer de nouvelles données légèrement différentes

- Translation aléatoire
- Rotation aléatoire
- Crop aléatoire
- Ajout aléatoire dans les composante RVB



<https://github.com/aleju/imgaug>

Application : reconnaissance de chiffres manuscrit

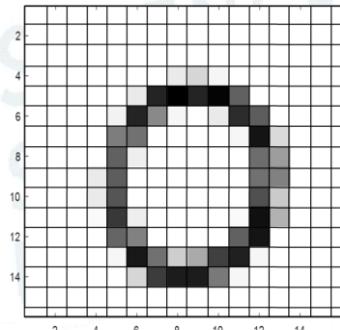
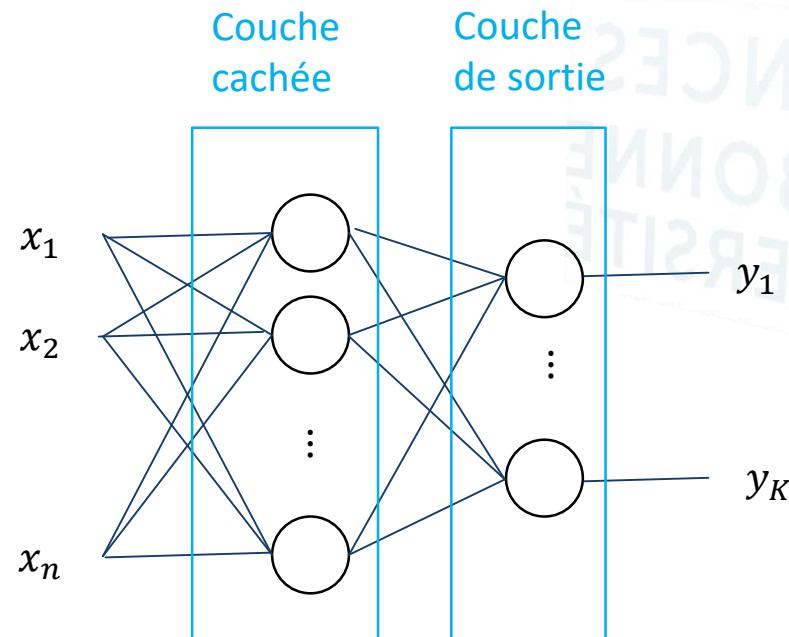


Image 16x16 → vecteur de dimension 256
10 chiffres

Exercice :

- Combien y a-t-il d'entrée ?
- Avec 100 neurones sur la couche cachée, combien y a-t-il de poids et de biais à estimer ?
- Et si l'image était de taille 100x100 ?



Application : reconnaissance de chiffres manuscrit

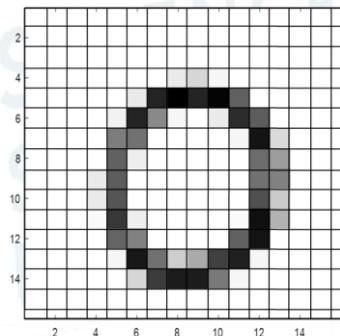
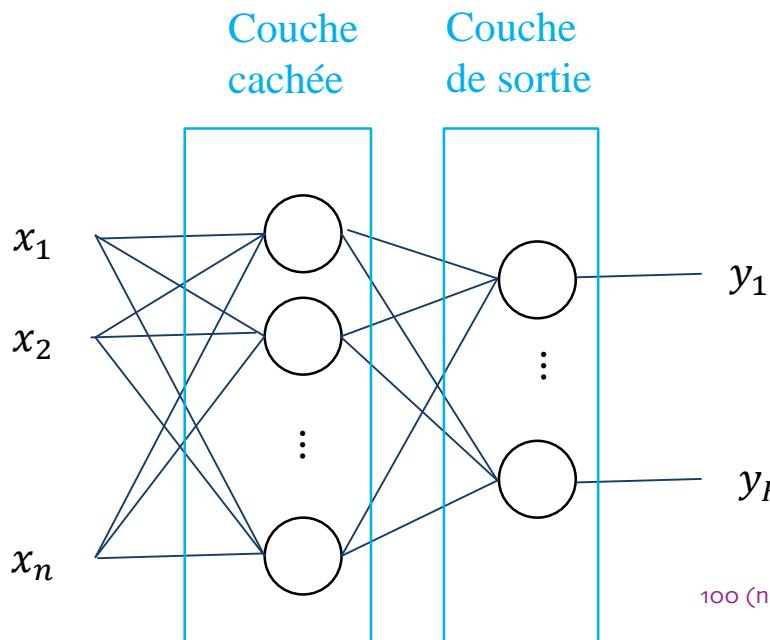


Image 16x16 → vecteur de dimension 256
10 chiffres

Exercice :

- Combien y a-t-il de neurones en entrée et en sortie ?
- Avec 100 neurones sur la couche cachée, combien y a-t-il de poids et de biais à estimer ?
- Et si l'image était de taille 100x100 ?



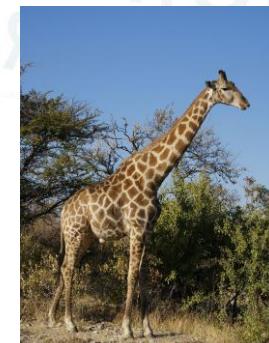
- 256 neurones en entrée
 - 10 neurones en sortie
- Couche cachée : $256 \times 100 + 100$
 Couche de sortie : $10 \times 100 + 10$
 Soit : **26.710** paramètres
 Pour une image de 100x100
1.011.110 paramètres

100 (neurones qu'on connecte à 10 sorties) => $100 \times 10 + 10$ le biais de la sortie

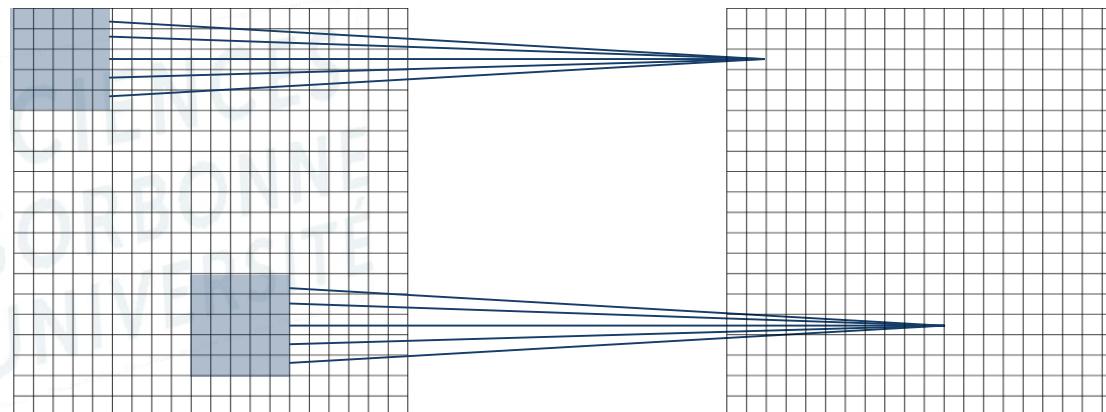
Les réseaux convolutifs profonds

Pourquoi aller plus loin ?

- Le perceptron multicouches ne passe pas à l'échelle (grande image)
- Une légère translation de la forme (1 pixel) change fortement les données d'entrée puisque chaque neurone de la couche d'entrée représente un pixel précis
- De la même façon, le réseau est très sensible aux changements d'échelle



Rappel sur la convolution



$$IC(x, y) = \sum_i \sum_j G(i, j) I(x - i, y - j)$$

L'idée des réseaux convolutifs est d'appliquer des convolutions dans les premières couches du réseau

Les coefficients du filtre (et un biais supplémentaire) sont les poids du réseau

Un neurone par pixel MAIS

- Chaque neurone n'est connecté qu'à un petit nombre de neurones de la couche précédente
- Les poids sont partagés par tous les neurones
- Les résultats de convolution sont envoyés à une fonction d'activation

Exercice :

Donner le résultat de convolution de l'image suivante par le filtre suivant (on ne gérera pas les bords)

Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

Image convoluée

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

Exercice :

Donner le résultat de convolution de l'image suivante par le filtre suivant (on ne générera pas les bords)

Image

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filtre

1	0	1
0	1	0
1	0	1

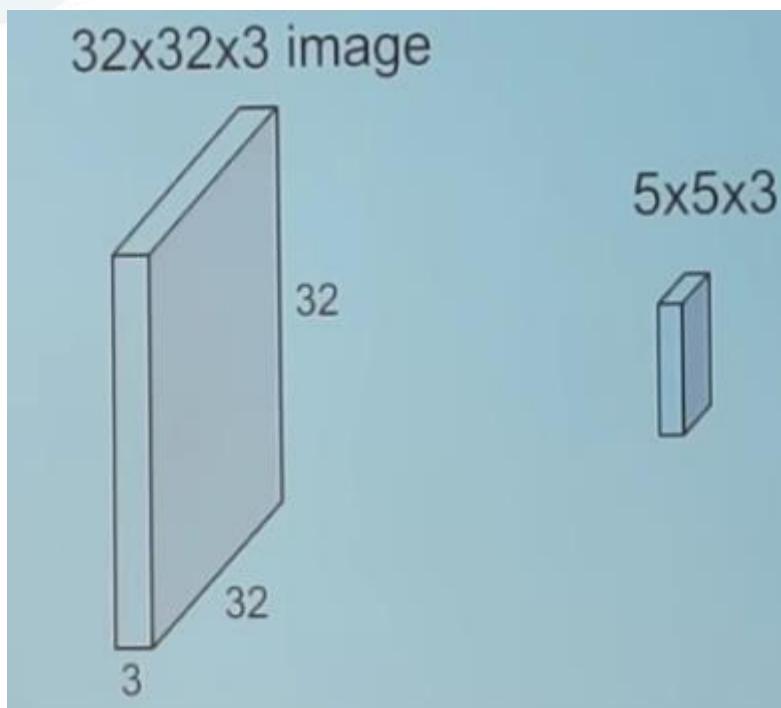
Image convoluée

4	3	4
2	4	3
2	3	4

Explication de :

[https://medium.com/technologymadeeasy/
the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5f8](https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5f8)

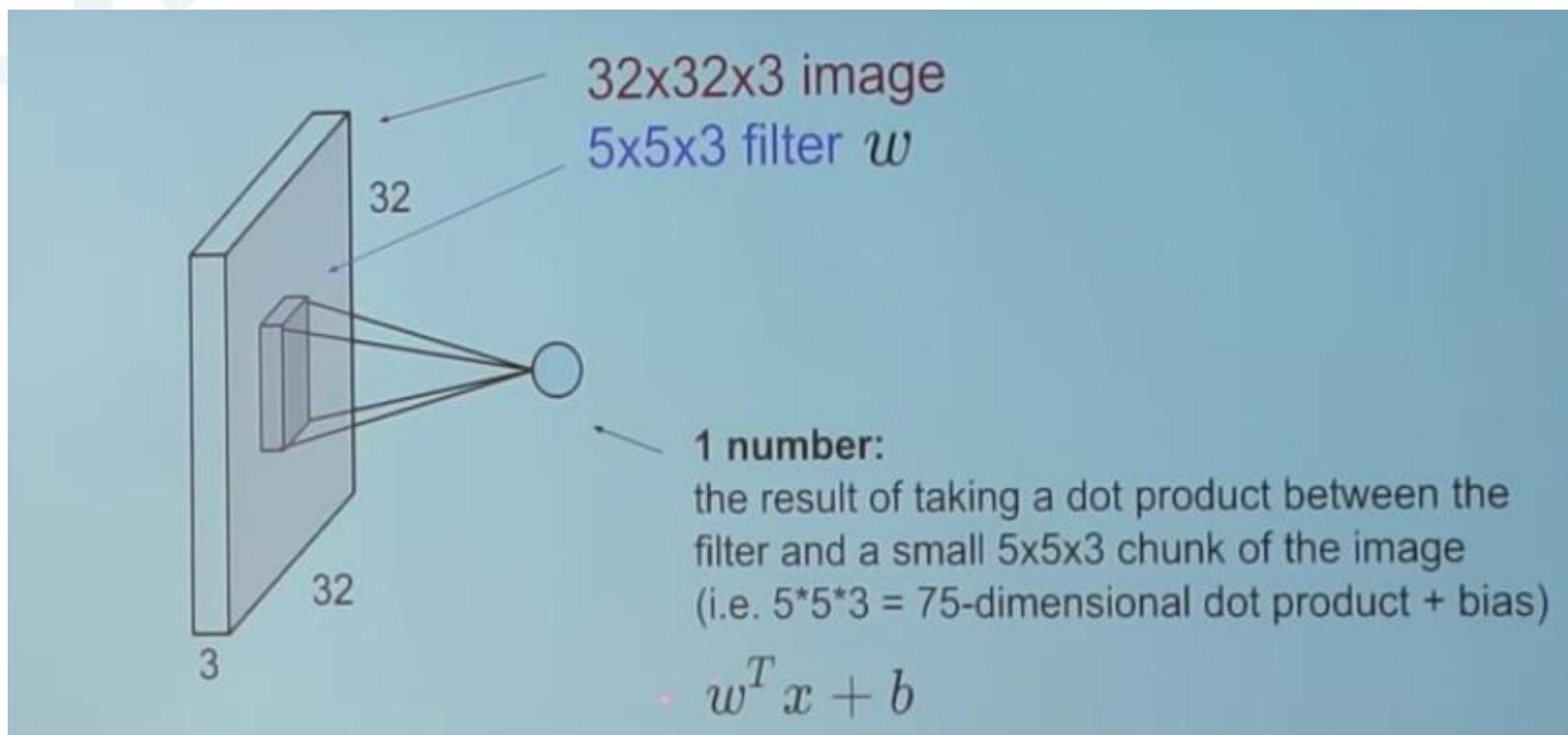
Les CNN travaillent sur des volumes :



Explication de :

[https://medium.com/technologymadeeasy/
the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5f8](https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5f8)

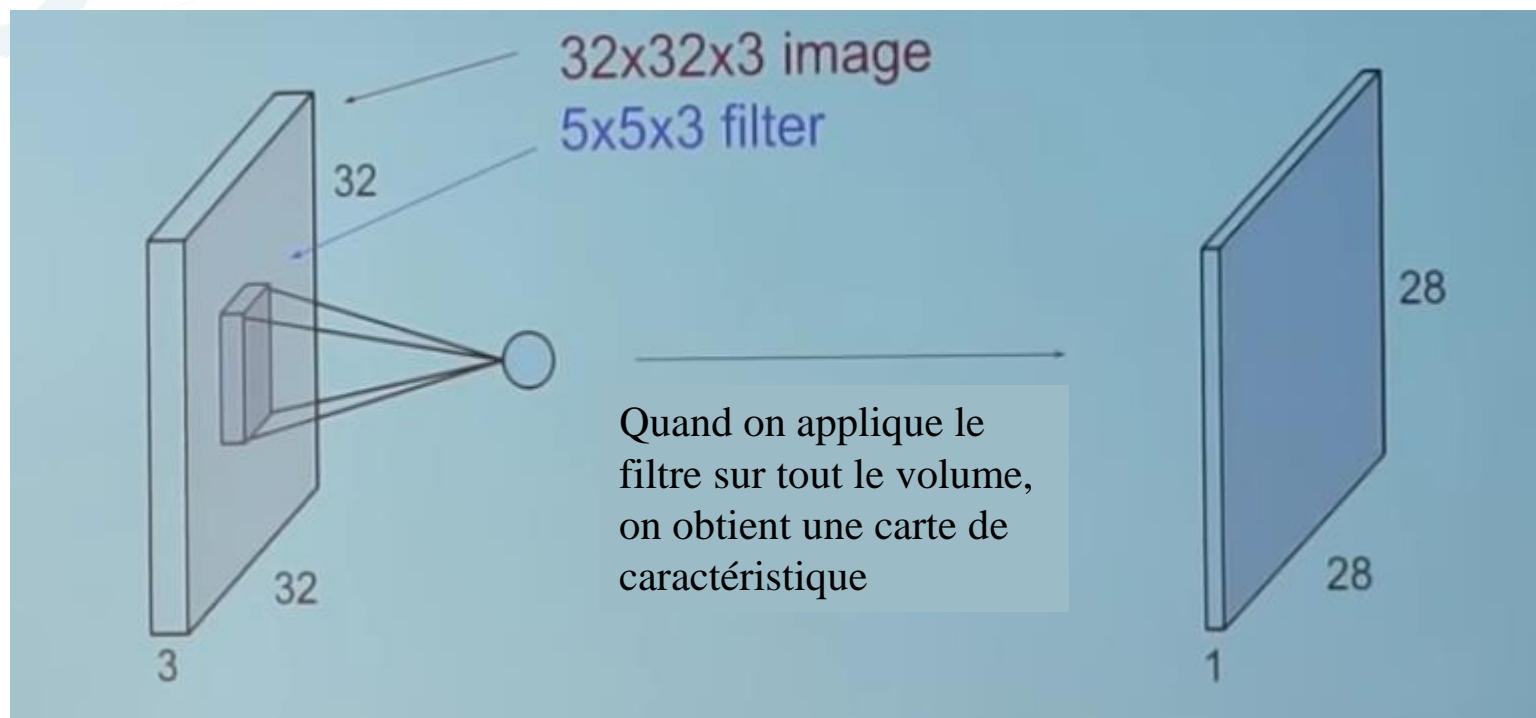
Les CNN travaillent sur des volumes :



Explication de :

[https://medium.com/technologymadeeasy/
the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8](https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8)

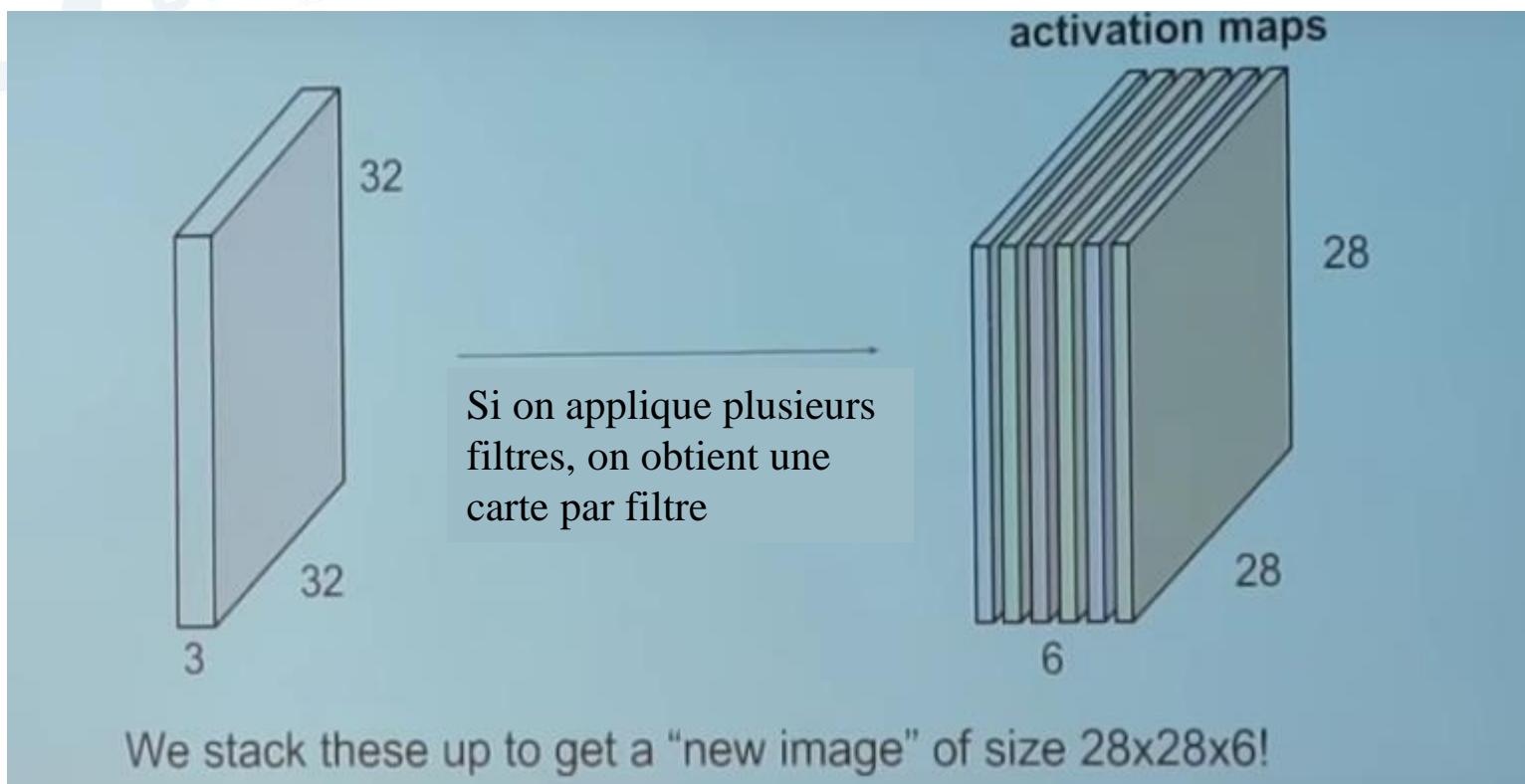
Les CNN travaillent sur des volumes :

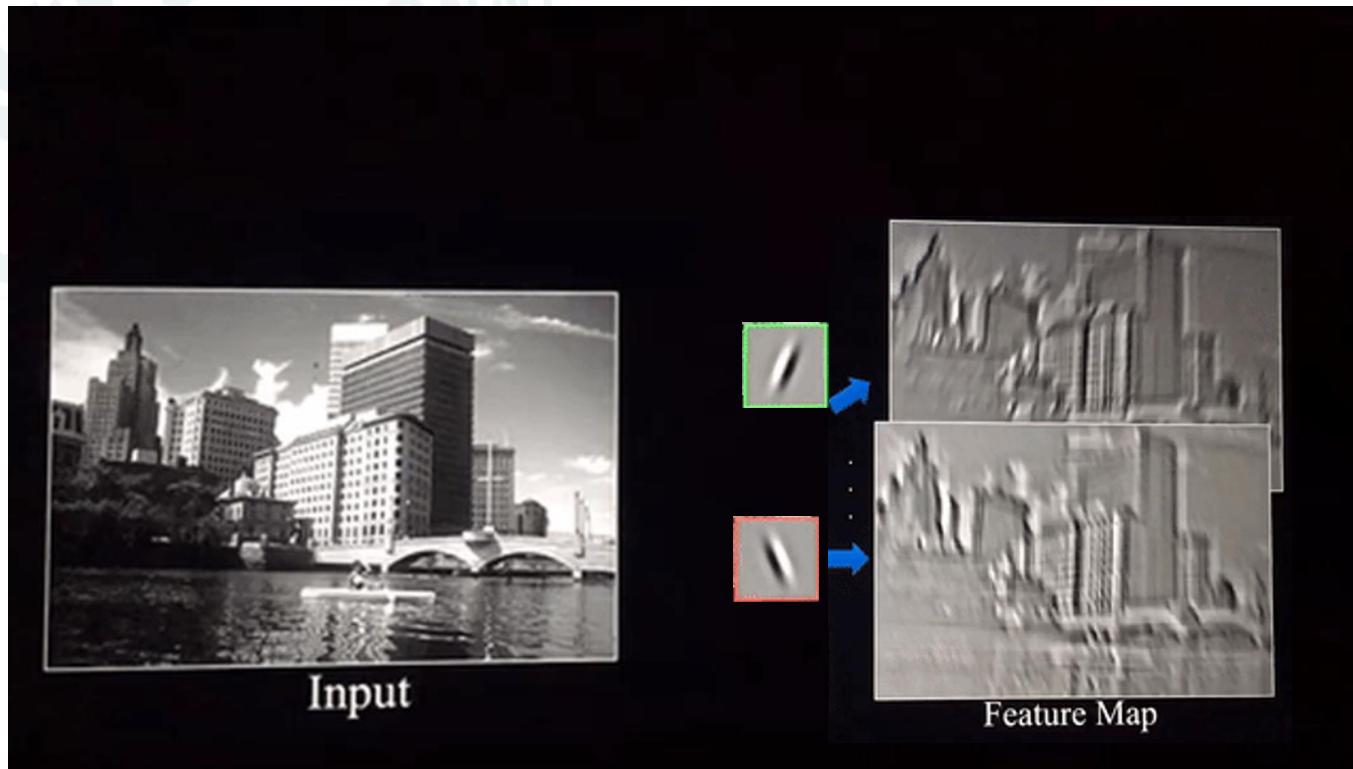


Explication de :

[https://medium.com/technologymadeeasy/
the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8](https://medium.com/technologymadeeasy/the-best-explanation-of-convolutional-neural-networks-on-the-internet-fbb8b1ad5df8)

Les CNN travaillent sur des volumes :





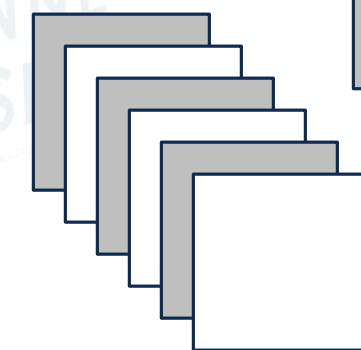
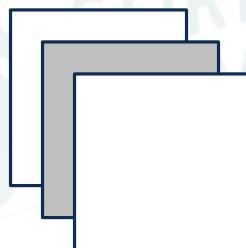
<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>

Dans les réseaux convolutifs, plusieurs convolutions sont estimées pour chaque couche, par exemple

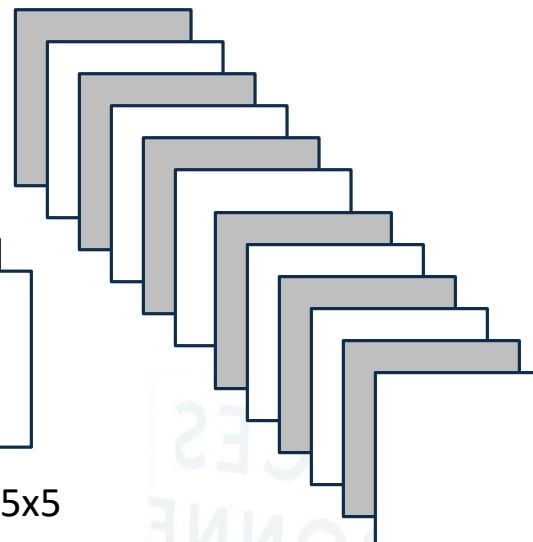
Couche 3

Couche 1

Couche 2



6 filtres 5x5



12 filtres 5x5

Pour un filtre donné, tous les neurones extraient exactement la même caractéristique (un contour vertical par exemple) à des positions différentes

→ Les CNN sont bien adaptés pour être invariants en translation. Une translation sur l'entrée va produire une translation sur la sortie

Stride et padding

Ces deux valeurs sont utiles pour définir une couche de convolution.

- Le **stride** indique le pas avec lequel on déplace le filtre. Un stride de 1 amène à une convolution classique. Un stride de 2 amène à une image 2 fois plus petite.
- Le **padding** sert à ajouter des colonnes et des lignes de zéros sur les bords de l'image pour que la convolution puisse être calculée sur toute l'image. Un padding de 1 ajoute 2 lignes et 2 colonnes (1 au début et 1 à la fin)

Exercice

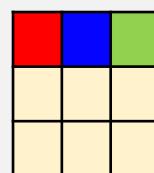
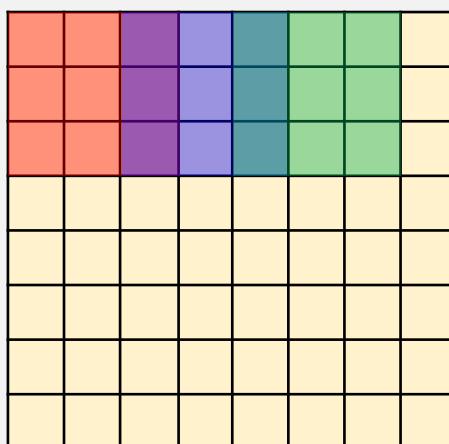
Considérons une image de taille 8x8 à laquelle on applique une couche convolutionnelle composée de 6 filtres 3x3, avec un stride de 2 et un padding de 0. Comment sera la sortie ?

Même question avec un padding de 1.

Exercice

Considérons une image de taille 8x8 à laquelle on applique une couche convolutionnelle composée de 6 filtres 3x3, avec un stride de 2 et un padding de 0. Comment sera la sortie ?

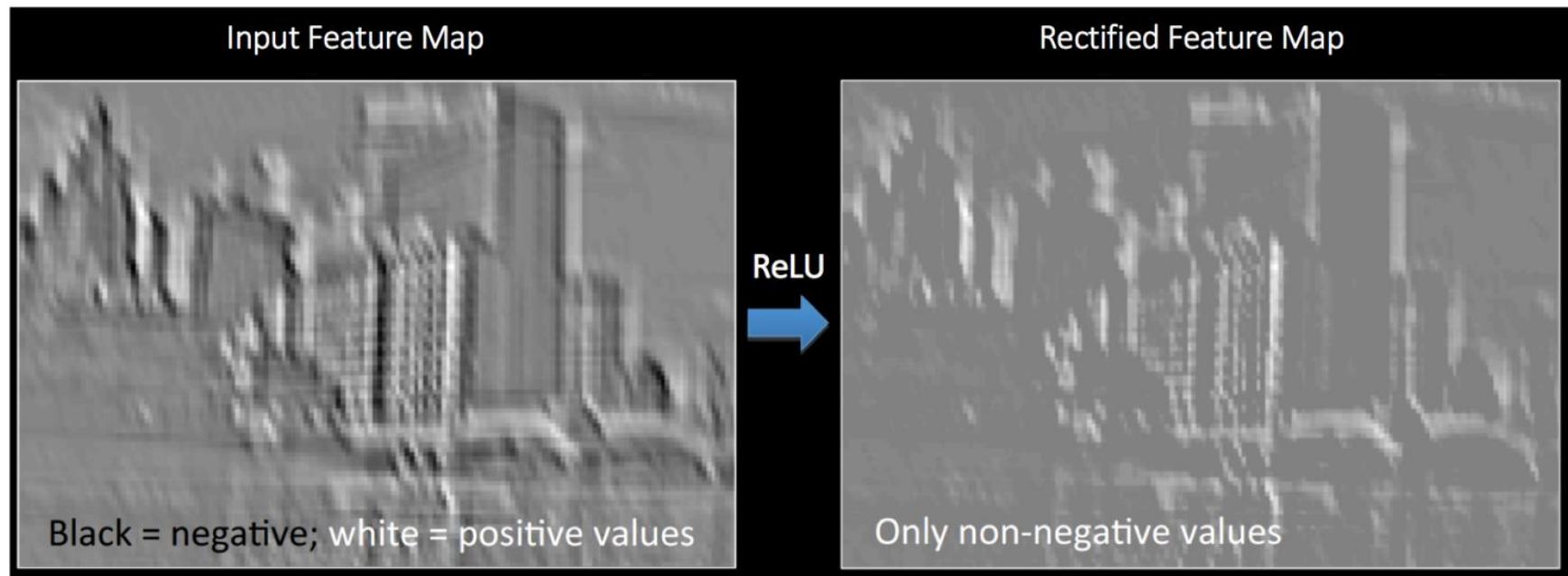
Même question avec un padding de 1.



Avec un stride de 2 et un padding de 0, on arrive à 6 cartes (6 filtres) de taille 3x3

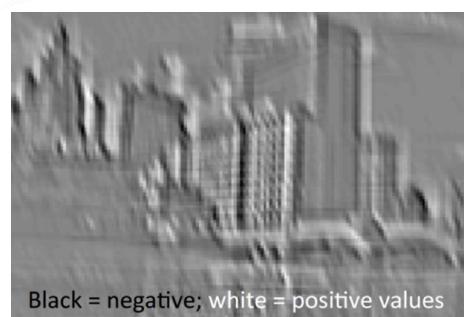
Avec un padding de 1, on a 6 cartes de taille 4x4

Chaque neurone des couches de convolution est soumis à une **fonction d'activation non linéaire**



http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

* Les réseaux convolutionnels



Convolution



Activation non linéaire

*Les réseaux convolutionnels

Les CNN comportent aussi des couches de sous-échantillonnage appelées pooling,

Le pooling

- réduit la dimension spatiale → le nombre de paramètres → contrôle le sur-apprentissage
- ajoute de l'invariance spatiale en translation
- ajoute de l'invariance spatiale en échelle
- Ils dépendent également du stride et du padding

Deux principaux types de pooling : max pooling ou average pooling

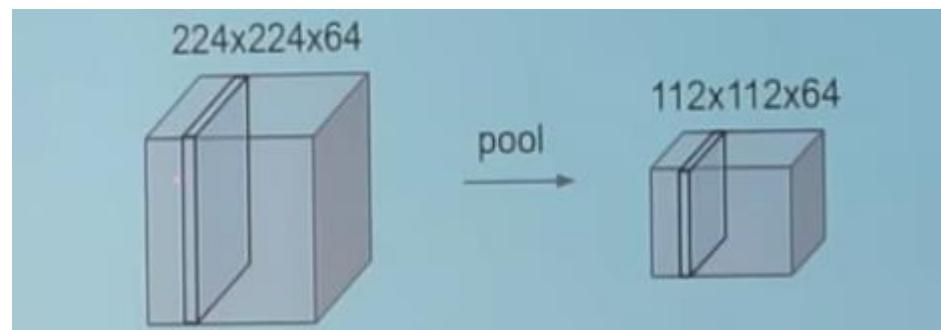
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

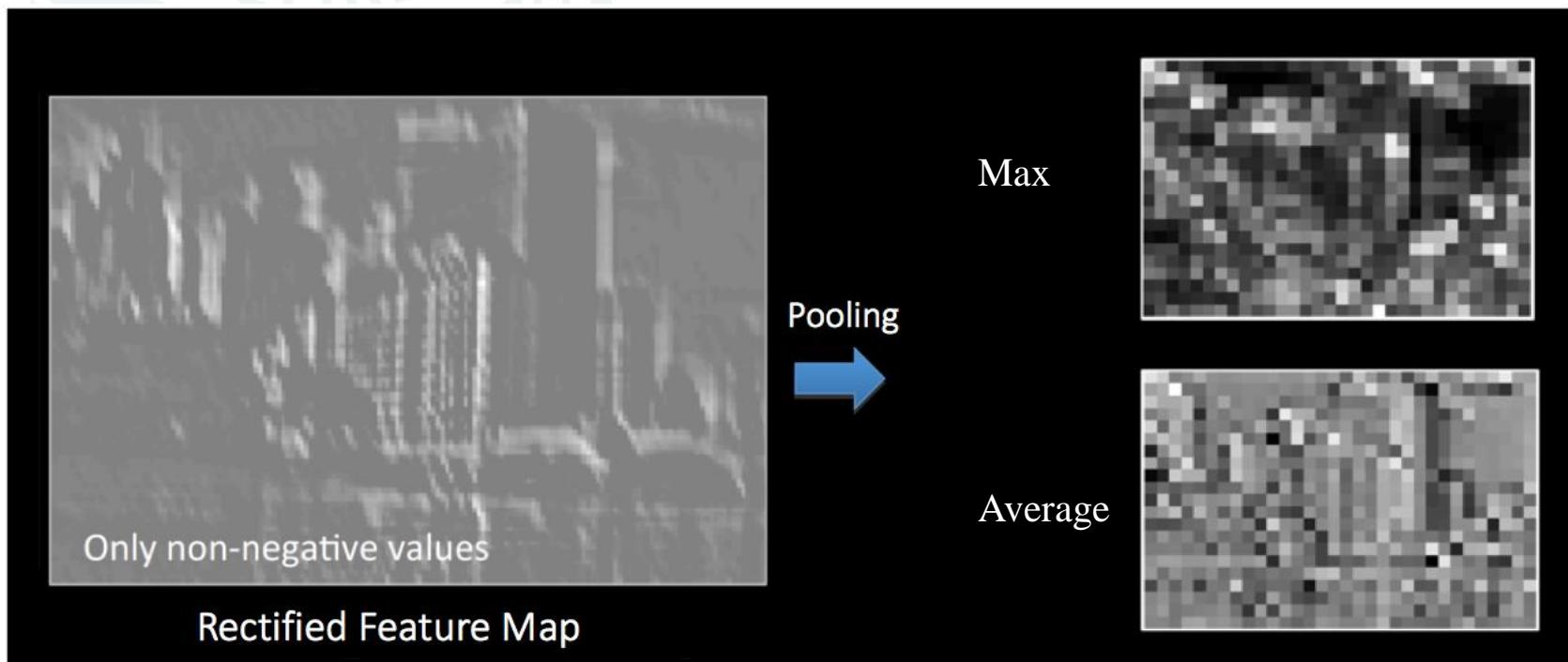
6	8
3	4

Max pooling

3	5
2	2

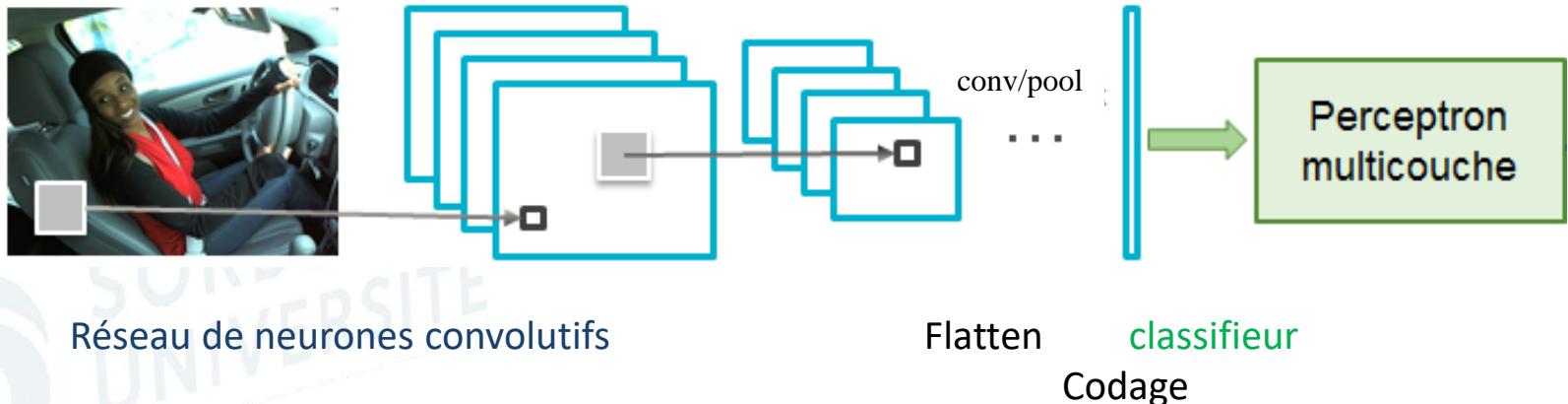
Average pooling





http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

*Les réseaux convolutionnels

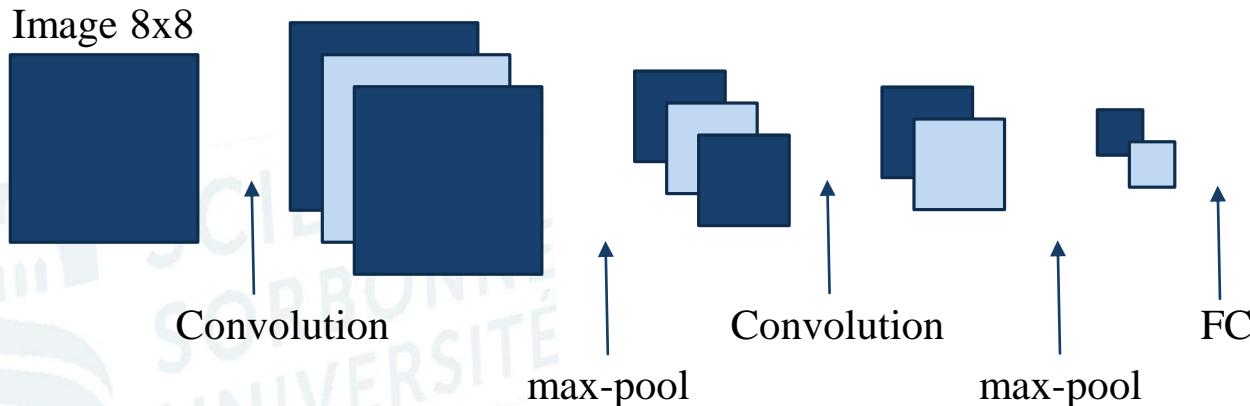


La première partie, composée de couches de convolution et de pooling, extrait les caractéristiques

La seconde partie, avec des neurones entièrement connectés (comme un MLP), réalise la classification

<https://blog.octo.com/classification-dimages-les-reseaux-de-neurones-convolutifs-en-toute-simplicite/>

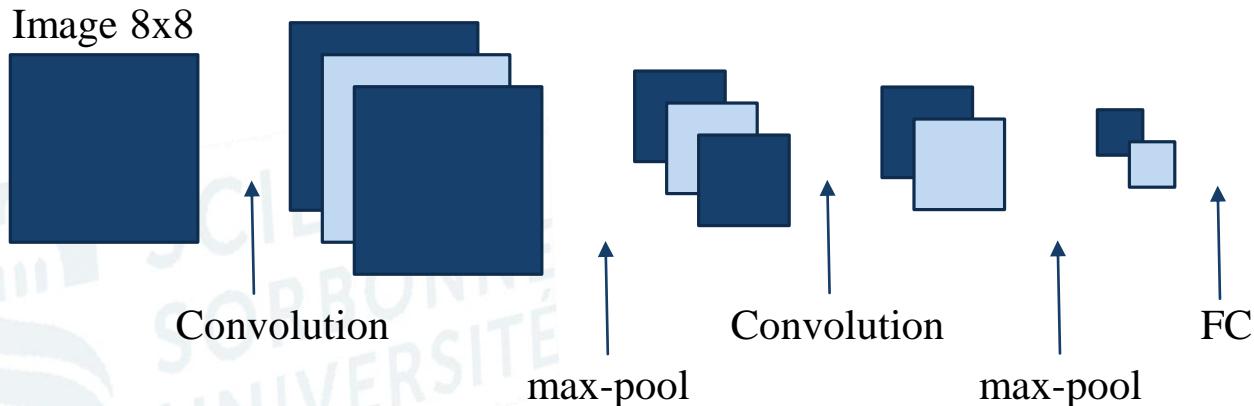
*Les réseaux convolutionnels



Exercice

- On considère le réseau ci-dessus pour lequel, toutes les opérations de convolution et de pooling sont faites avec un padding ‘same’.
- Les poolings (max pooling) se font avec une fenêtre de taille 3×3 et un stride de 2
- Les convolutions se font avec des fenêtres 3×3 et un stride de 1
- La couche « fully connected » FC a 3 neurones
- Toutes les fonctions d’activation sont des RELU, sauf sur la dernière couche où on a du soft-max

*Les réseaux convolutionnels



Exercice

- On met en entrée l'image ci-dessous: Quelles seront les 3 images en sortie de la première couche sachant que les 3 filtres utilisés sont (stride=1, padding=1, pas de biais, RELU):

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

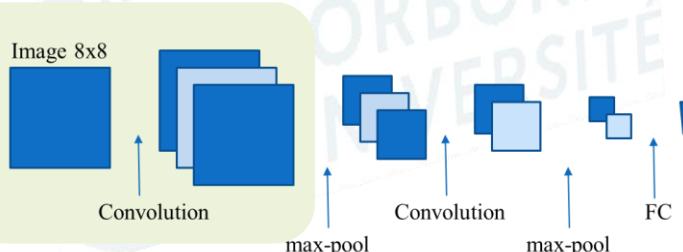
0	0	0	1	1	0	0	0
0	0	1	1	1	0	0	0
0	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0

*Les réseaux convolutionnels

Exercice

- On met en entrée l'image ci-dessous: Quelles seront les 3 images en sortie de la première couche sachant que les 3 filtres utilisés sont : (stride=1, padding=1, pas de biais, RELU):

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



0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	0
0	0	1	1	1	1	0	0
0	0	0	1	1	0	0	0

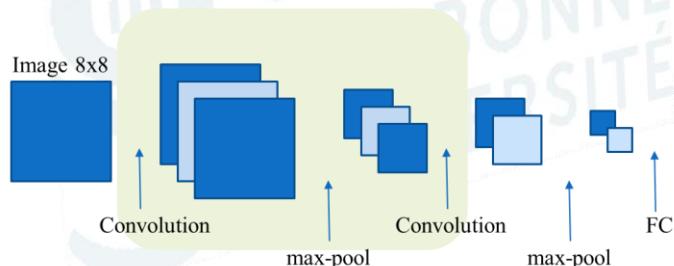


0	1	3	5	5	3	1	0
1	3	6	8	8	6	3	1
3	6	8	9	9	8	6	3
5	8	9	9	9	9	8	5
5	8	9	9	9	9	8	5
3	6	8	9	9	8	6	3
1	3	6	8	8	6	3	1
0	1	3	5	5	3	1	0
0	0	0	0	1	2	1	0
0	0	0	0	1	2	2	1
0	0	0	0	0	1	2	2
0	0	0	0	0	0	1	3
0	0	0	0	0	0	1	3
0	0	0	0	0	1	2	2
0	0	0	0	1	2	2	1
0	0	0	0	1	2	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1
2	2	1	0	0	1	2	2
1	2	2	1	1	2	2	1
0	1	2	3	3	2	1	0

* Les réseaux convolutionnels

Exercice

- Quelles seront les sorties de la première couche de pooling 3x3 (stride=2, padding=1, max pooling)?



3	8	8	6
8	9	9	9
8	9	9	9
6	9	9	8

0	0	2	2
0	0	2	3
0	0	1	3
0	0	2	2

0	0	0	0
0	0	0	0
2	2	1	2
2	3	3	2

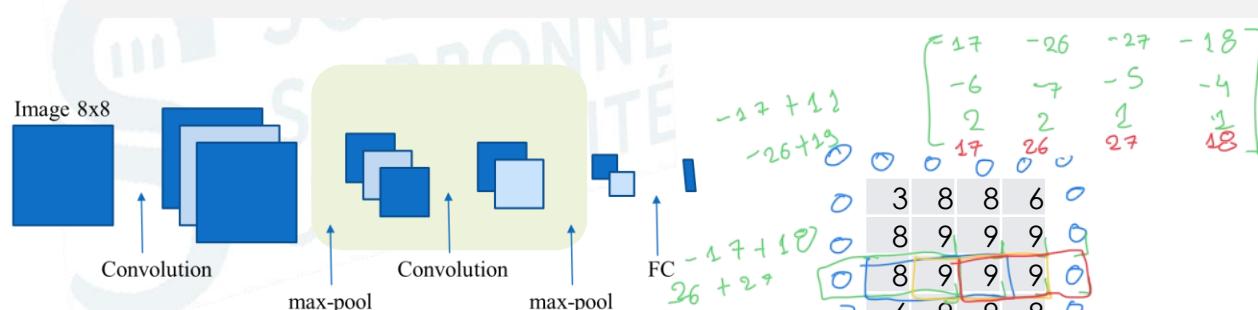
* Les réseaux convolutionnels

Suit une étape de convolution avec les filtres (mêmes valeurs sur les 3 cartes):

$$\begin{array}{cccc|c} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & -1 \\ 1 & 1 & 0 & -1 \\ \hline 1 & 0 & -1 & 0 \end{array}$$

1	1	1
1	1	1
0	1	1
-1	0	0
-1	-1	-1

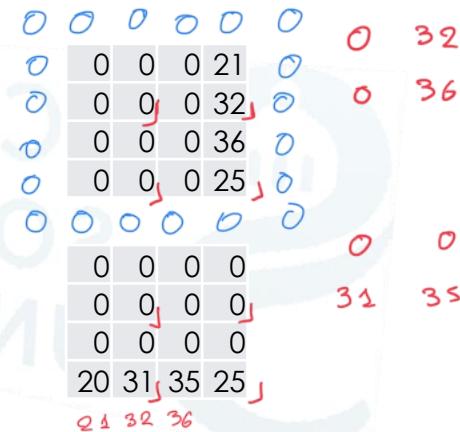
stride=1
padding=1
pas de biais
RELU



$$\begin{array}{l} \text{L1L} \\ \text{000} \\ -1-1-1 \end{array} \left[\begin{array}{cccc} 0 & -2 & -5 & -5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 9 & 9 \end{array} \right] \begin{array}{l} 0 \\ 0 \\ 0 \\ 0 \\ + \end{array} \left[\begin{array}{ccccc} 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & 2 & 3 & 0 \\ 0 & 0 & 1 & 3 & 0 \\ 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right]$$

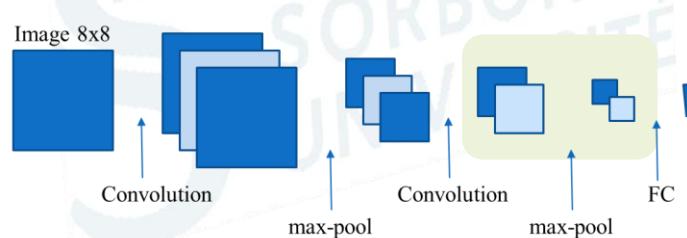
$$\left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ -9 & -5 & -5 & -3 \\ -5 & -8 & -8 & -5 \\ 9 & 5 & 5 & 3 \end{array} \right] \quad \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 1 & 2 \\ 0 & 2 & 3 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right]$$

$$\left[\begin{array}{cccc} 17 & -26 & -27 & -18 \\ -6 & 7 & -5 & -4 \\ 2 & 2 & 1 & 1 \\ 17 & 26 & 8 & 18 \end{array} \right] + \left[\begin{array}{cccc} 0 & -2 & -5 & -5 \\ -9 & -9 & -5 & -3 \\ -5 & -8 & -7 & -4 \\ - & - & -7 & 7 \end{array} \right] = \left[\begin{array}{cccc} 0 & 0 & 0 & 0 \\ -9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array} \right]$$



*Les réseaux convolutionnels

Puis de nouveau un max-pooling (stride=2, padding=1)

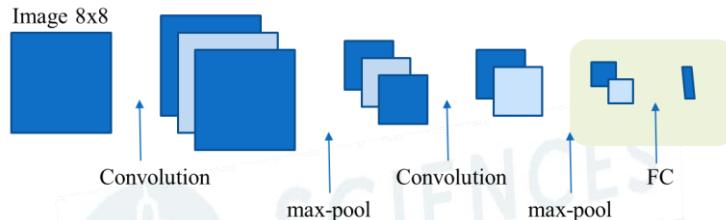


$$\begin{matrix}
 & 0 & 0 & 0 & 21 \\
 & 0 & 0 & 0 & 32 \\
 & 0 & 0 & 0 & 36 \\
 & 0 & 0 & 0 & 25 \\
 \\
 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 0 \\
 20 & 31 & 35 & 25
 \end{matrix}$$

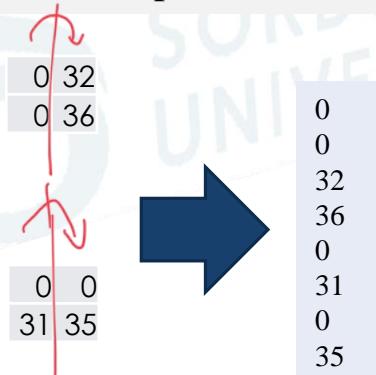


$$\begin{matrix}
 & 0 & 32 \\
 & 0 & 36 \\
 \\
 & 0 & 0 \\
 31 & 35 \\
 32 & 36
 \end{matrix}$$

*Les réseaux convolutionnels



Puis une mise à plat :



final couche

La couche FC est composée de 3 neurones dont les poids et biais valent:

1	1	-1	-1	0	0	0	0	0	0
0	0	1	1	1	1	0	0	0	5
0	0	0	0	-1	-1	1	1	1	3

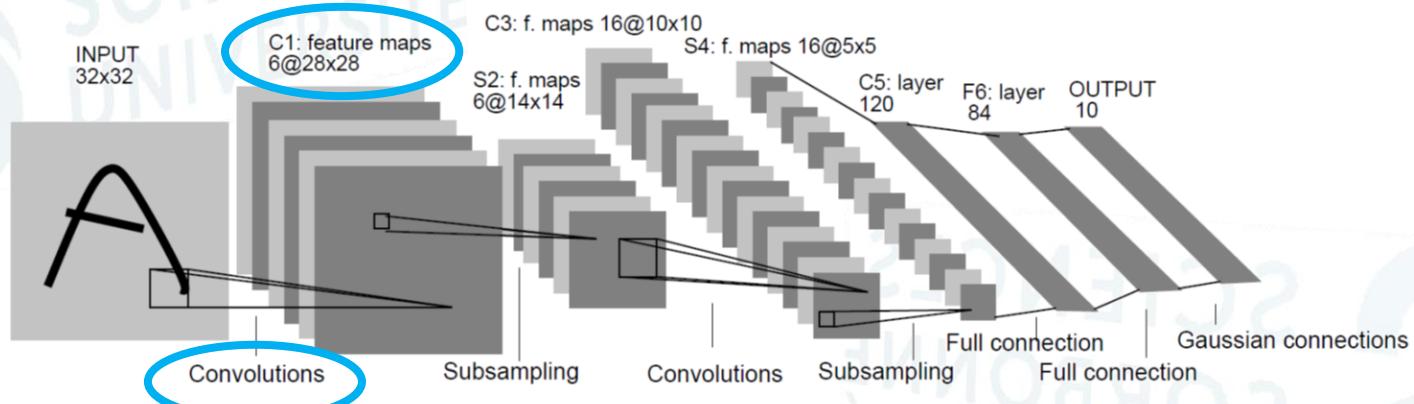
Que vaudra la sortie avant le soft-max ? -68, 104 et 7

Après le soft-max ? 0, 1, 0

Un des premiers CNN (un petit CNN) a été proposé par Yan Lecun en 1998 : **LENET5**

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, **Gradient-Based Learning Applied to Document Recognition**,
Proceedings of the IEEE, 86(11):2278-2324, November 1998

Les couches de convolution



Exercice

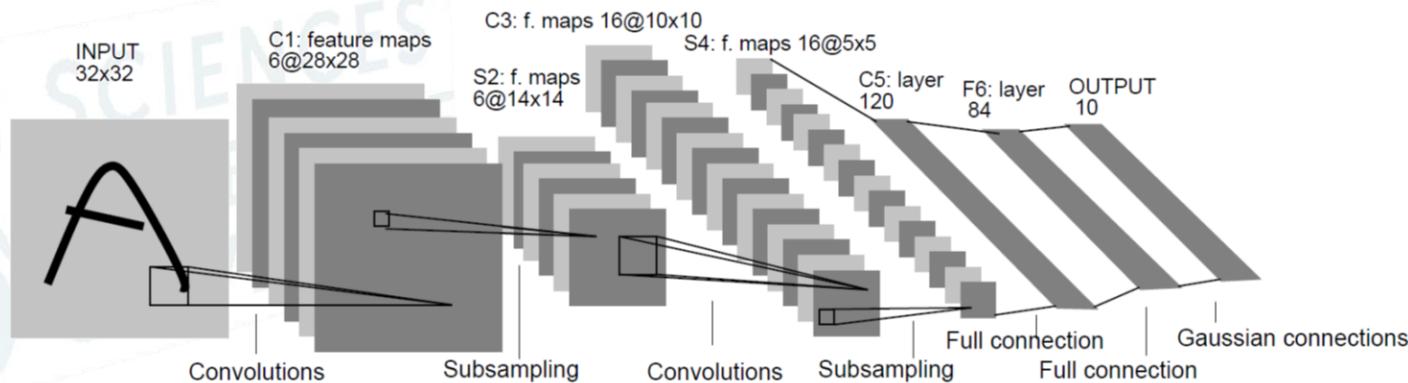
Couche C1 : 6 filtres de taille 5x5

Pourquoi les features map sont de taille 28x28 *parce qu'on a fait padding = 0. Donc il était diminué*

Combien de paramètres y a-t-il à estimer sur la première couche ?

$$6 \times (5 \times 5 + 1)$$

Les couches de convolution



Exercice

Couche C1 : 6 filtres de taille 5x5

Pourquoi les features map sont de taille 28x28

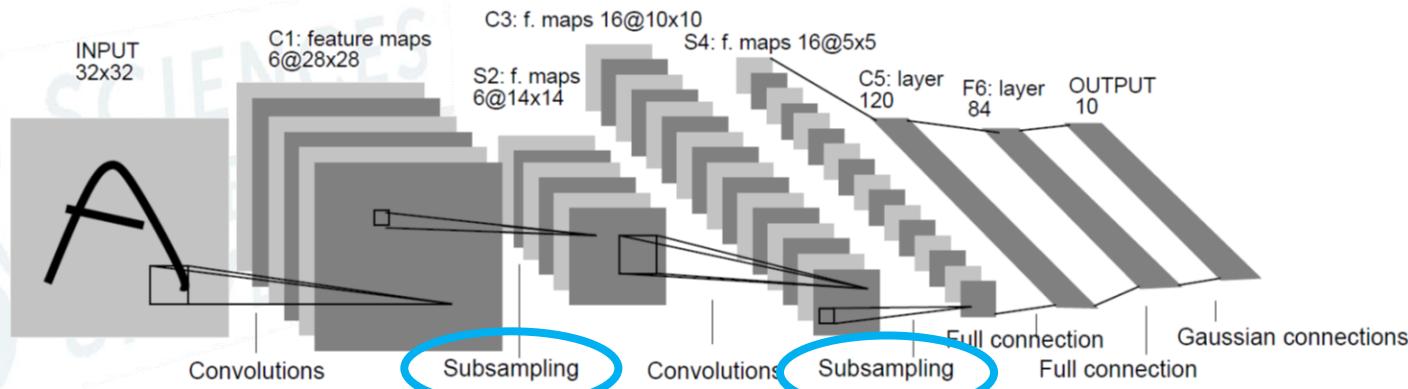
Combien de paramètres y a-t-il à estimer sur la première couche ?

Effet de bord → 2 pixels de chaque côté → stride=1, padding=0

Pour chaque filtre, il faut estimer les coefficients du filtre 5x5 et le biais

$$\rightarrow 6 * (5 \times 5 + 1) = 156$$

Le pooling (ou agglomération - subsampling)

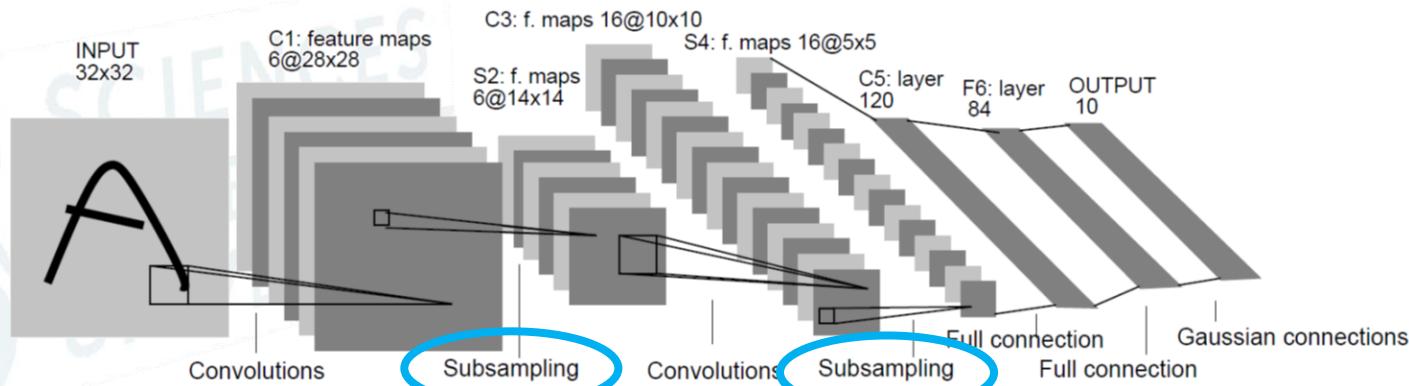


Le pooling

Quel stride ?

c'est de stride 2 car c'est la moitié chaque fois

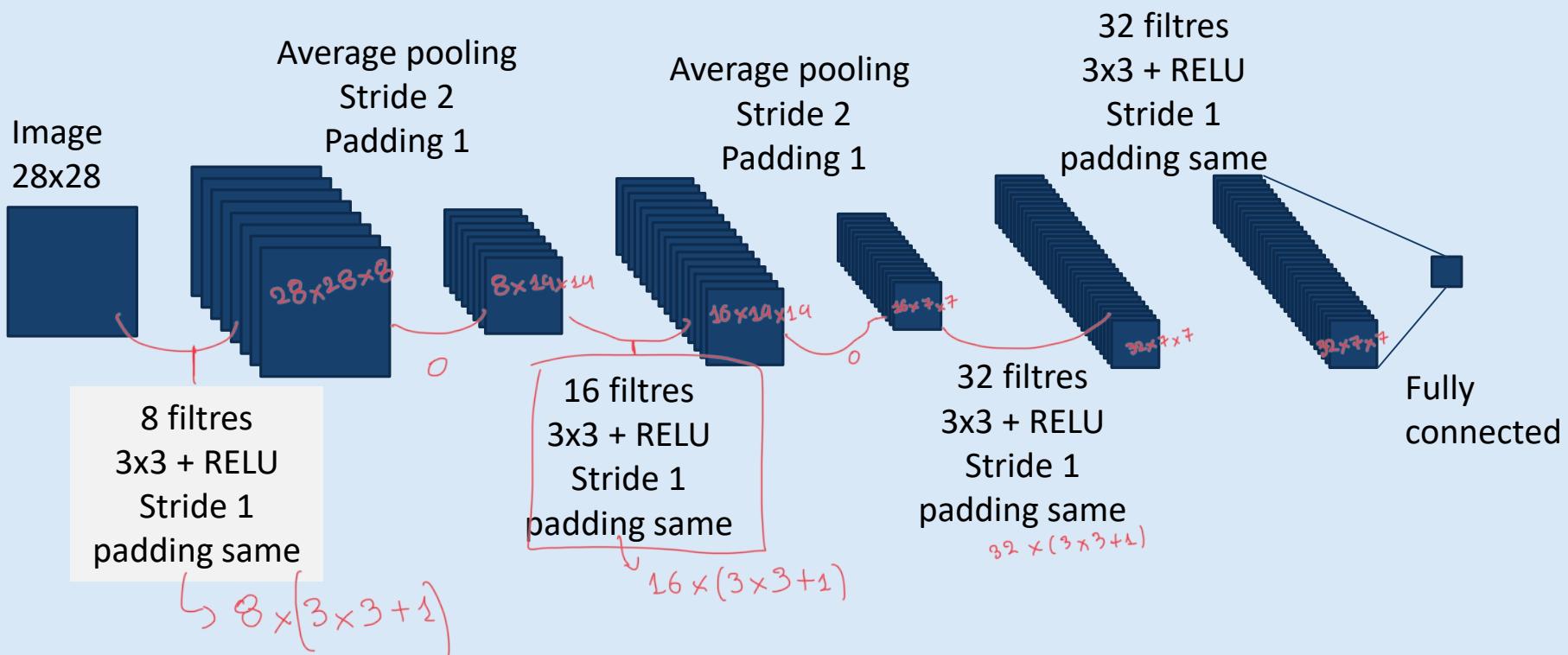
Le pooling (ou agglomération - subsampling)



Le pooling
Quel stride ?

Stride = 2

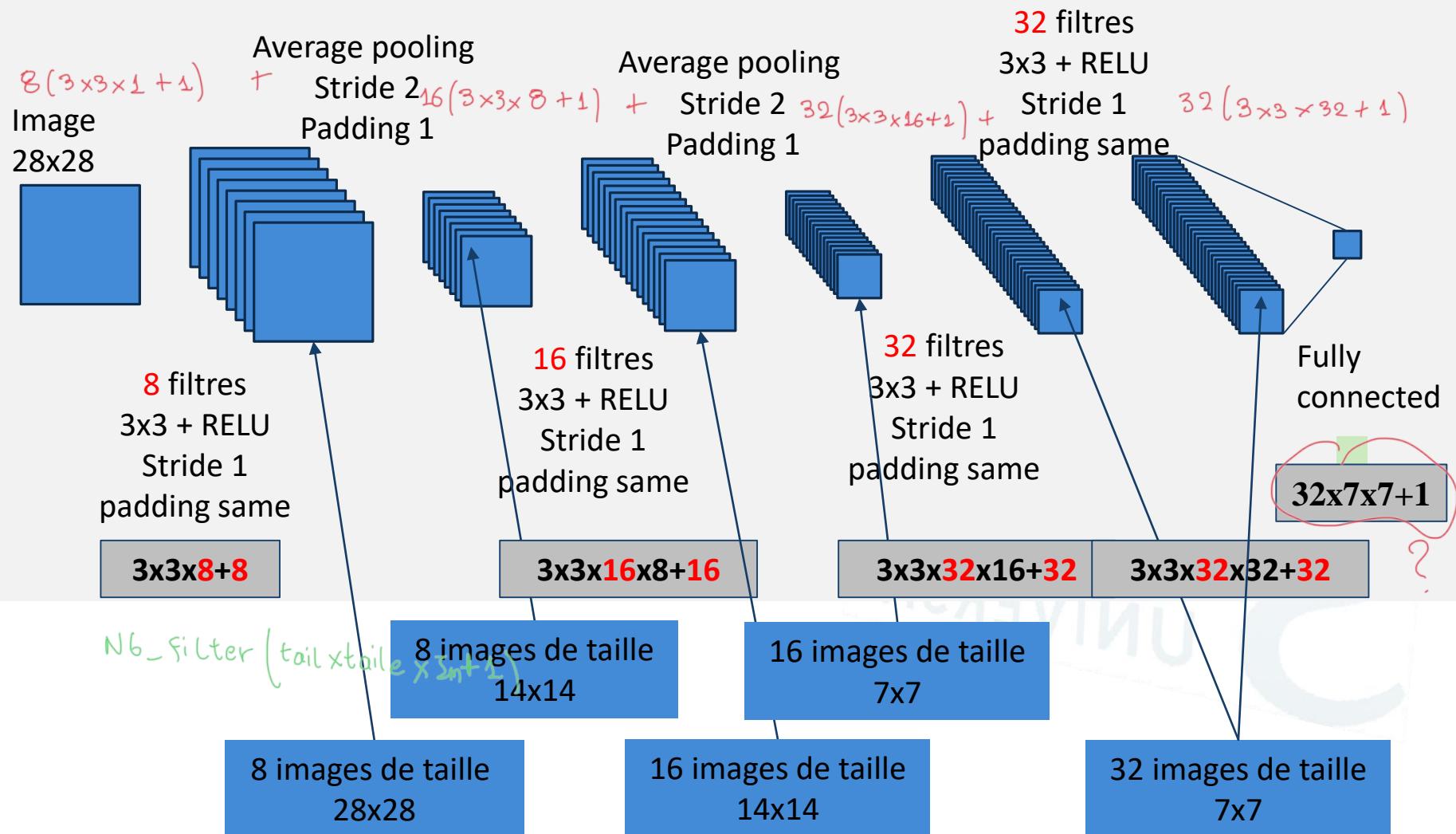
*Les réseaux convolutionnels



Exercice

On considère le réseau ci-dessus utilisé pour régresser une variable. Donner le nombre de paramètres à estimer

*Les réseaux convolutionnels

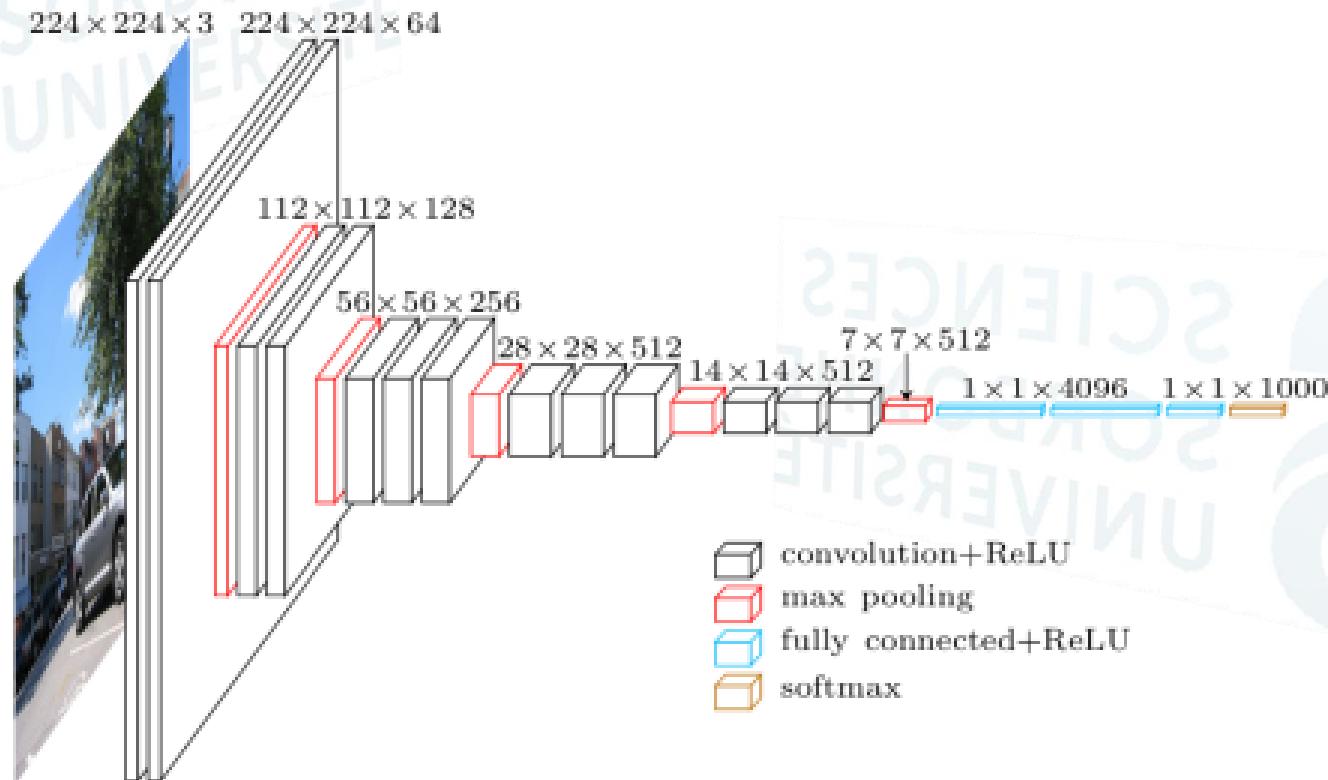


Nombre total de paramètres : $(3 \times 3 \times 8+8) + (3 \times 3 \times 16 \times 8+16) + (3 \times 3 \times 32 \times 16+32) + (3 \times 3 \times 32 \times 32+32) = 16705$

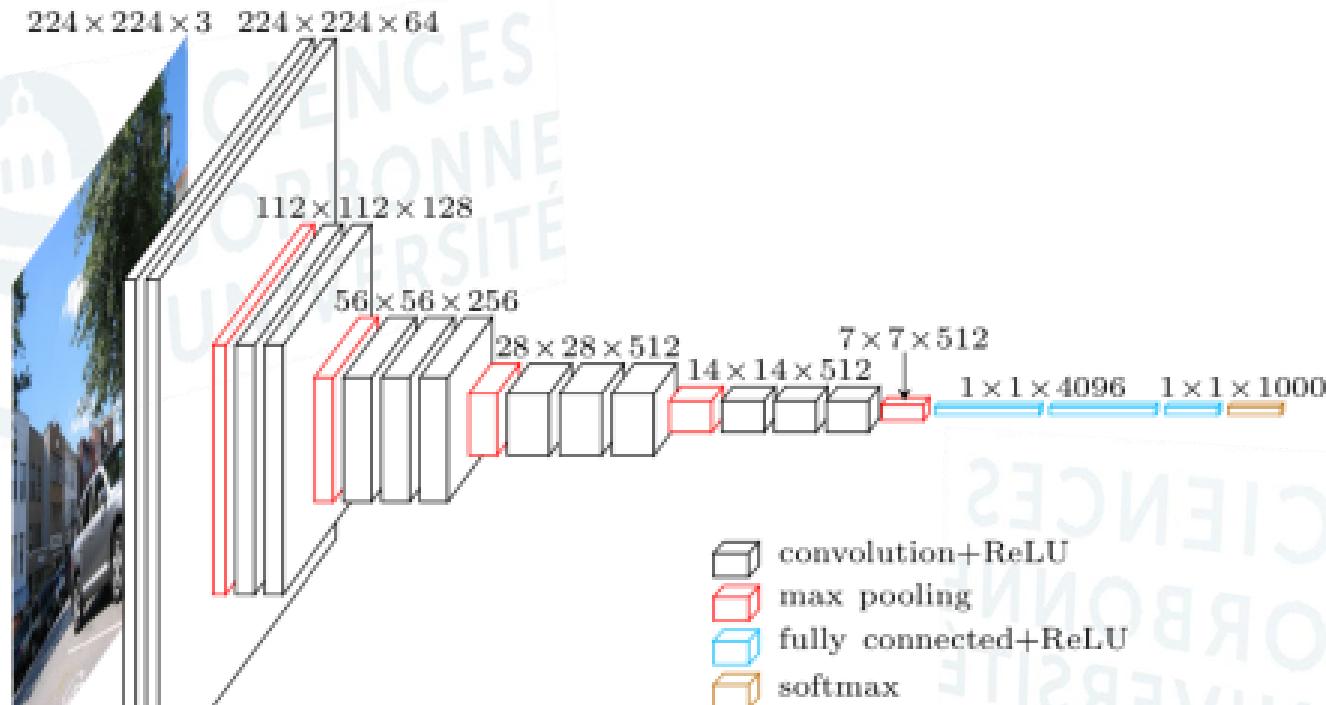
Architecture VGG

* L'architecture VGG16

SIMONYAN, Karen et ZISSERMAN, Andrew. Very deep convolutional networks for large-scale image recognition.
arXiv preprint arXiv:1409.1556, 2014.
http://www.robots.ox.ac.uk/~vgg/research/very_deep/



* L'architecture VGG16



Toutes les convolutions sont faites avec des filtres 3×3 , pourquoi ?

Toutes les convolutions sont faites avec des filtres 3x3, pourquoi ?

Deux convolutions 3x3 reviennent à une convolution 5x5

Trois convolutions 3x3 reviennent à une convolutions ??

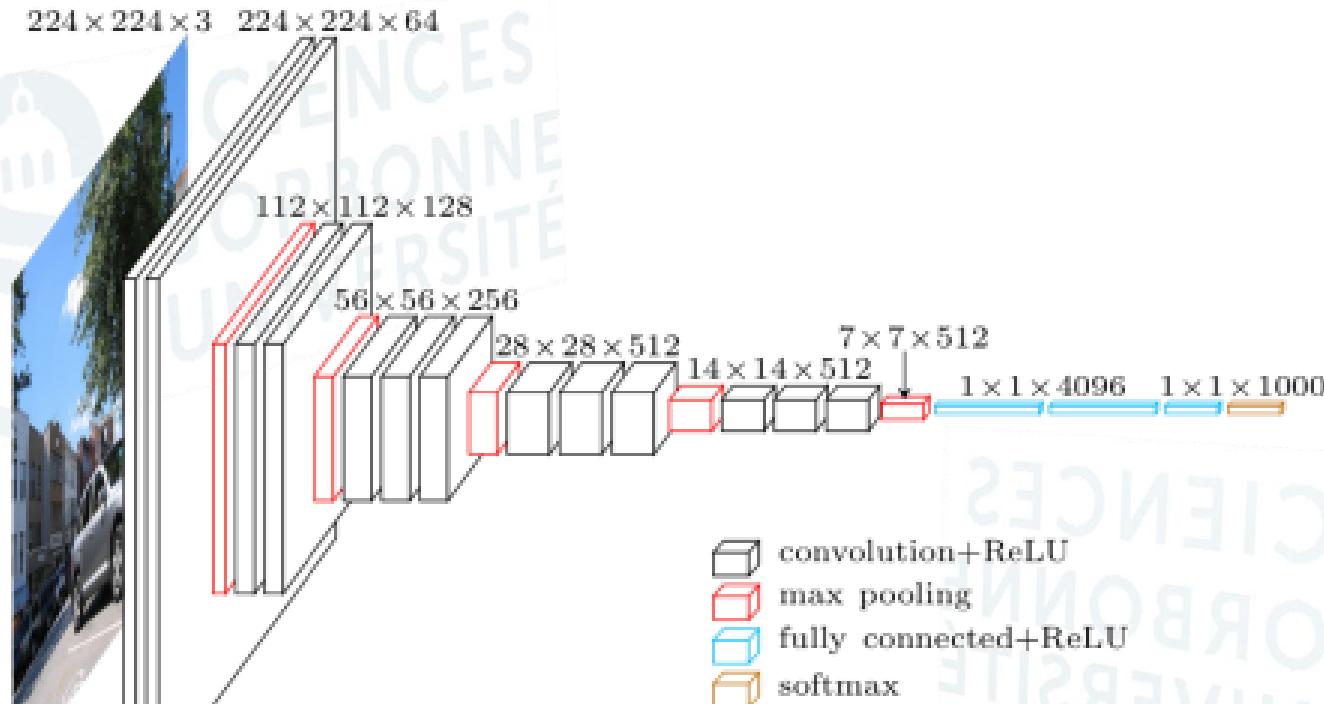
Un filtre large (pas tous) peut être remplacé par une pile de filtres successifs plus petits

→ Moins de paramètres à estimer

→ Plus de non linéarités

* L'architecture VGG16

Exemple : VGG16



Combien y a-t-il de paramètres à estimer ?

* L'architecture VGG16

Exemple : VGG16

INPUT: [224x224x3] poids:

CONV3-64: [224x224x64] poids :

CONV3-64: [224x224x64] poids :

POOL2: [112x112x64] poids :

CONV3-128: [112x112x128] poids :

CONV3-128: [112x112x128] poids :

POOL2: [56x56x128] poids :

CONV3-256: [56x56x256] poids :

CONV3-256: [56x56x256] poids :

CONV3-256: [56x56x256] poids :

POOL2: [28x28x256] poids :

CONV3-512: [28x28x512] poids :

CONV3-512: [28x28x512] poids :

CONV3-512: [28x28x512] poids :

POOL2: [14x14x512] poids :

CONV3-512: [14x14x512] poids :

CONV3-512: [14x14x512] poids :

CONV3-512: [14x14x512] poids :

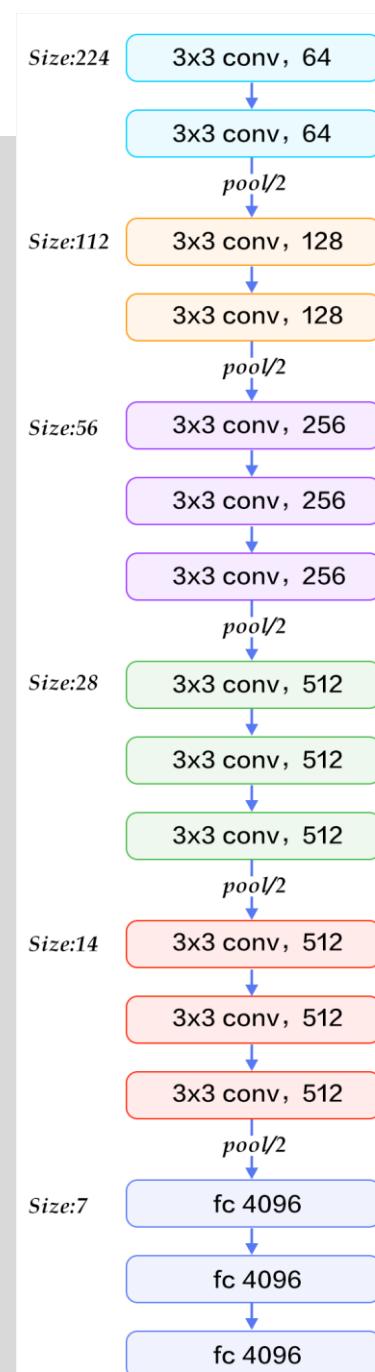
POOL2: [7x7x512] poids :

FC: [1x1x4096] poids :

FC: [1x1x4096] poids :

FC: [1x1x1000] poids :

TOTAL :



* L'architecture VGG16

Exemple : VGG16

INPUT: [224x224x3] poids: 0

CONV3-64: [224x224x64] poids : $(3*3*3)*64 + 64 = 1792$

CONV3-64: [224x224x64] poids : $(3*3*64)*64+64 = 36928$

POOL2: [112x112x64] poids : 0

CONV3-128: [112x112x128] poids : $(3*3*64)*128+128 = 73856$

CONV3-128: [112x112x128] poids : $(3*3*128)*128+128 = 147584$

POOL2: [56x56x128] poids : 0

CONV3-256: [56x56x256] poids : $(3*3*128)*256+256 = 295168$

CONV3-256: [56x56x256] poids : $(3*3*256)*256+256 = 590080$

CONV3-256: [56x56x256] poids : $(3*3*256)*256+256 = 590080$

POOL2: [28x28x256] poids : 0

CONV3-512: [28x28x512] poids : $(3*3*256)*512+512 = 1180160$

CONV3-512: [28x28x512] poids : $(3*3*512)*512+512 = 2359808$

CONV3-512: [28x28x512] poids : $(3*3*512)*512+512 = 2359808$

POOL2: [14x14x512] poids : 0

CONV3-512: [14x14x512] poids : $(3*3*512)*512+512 = 2359808$

CONV3-512: [14x14x512] poids : $(3*3*512)*512+512 = 2359808$

CONV3-512: [14x14x512] poids : $(3*3*512)*512+512 = 2359808$

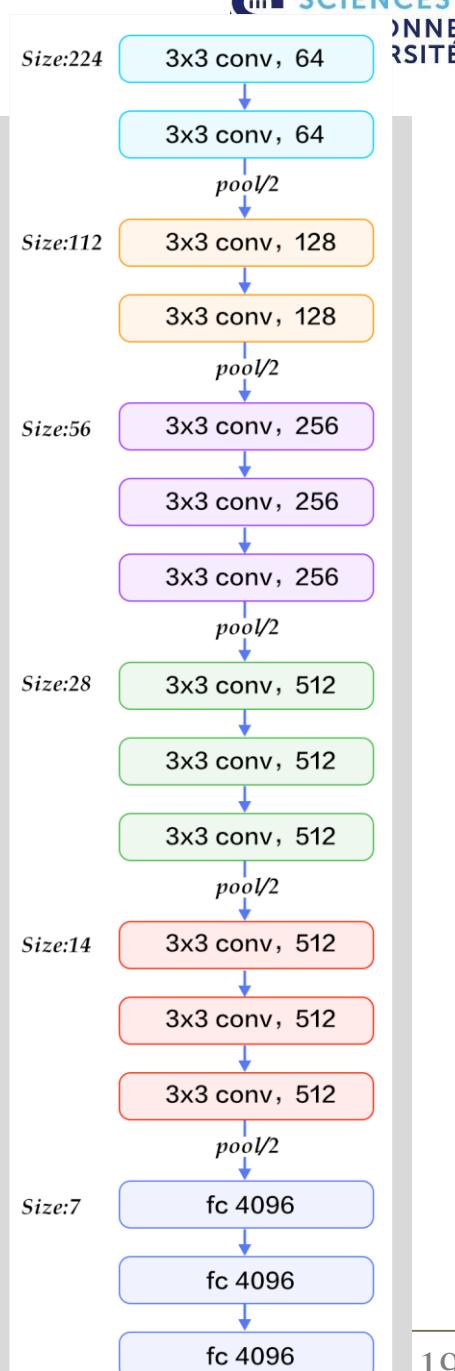
POOL2: [7x7x512] poids : 0

FC: [1x1x4096] poids : $7*7*512*4096+4096 = 102764544$

FC: [1x1x4096] poids : $4096*4096+4096 = 16781312$

FC: [1x1x1000] poids : $4096*1000+1000 = 4097000$

TOTAL : ~138.000.000 paramètres



VGG16

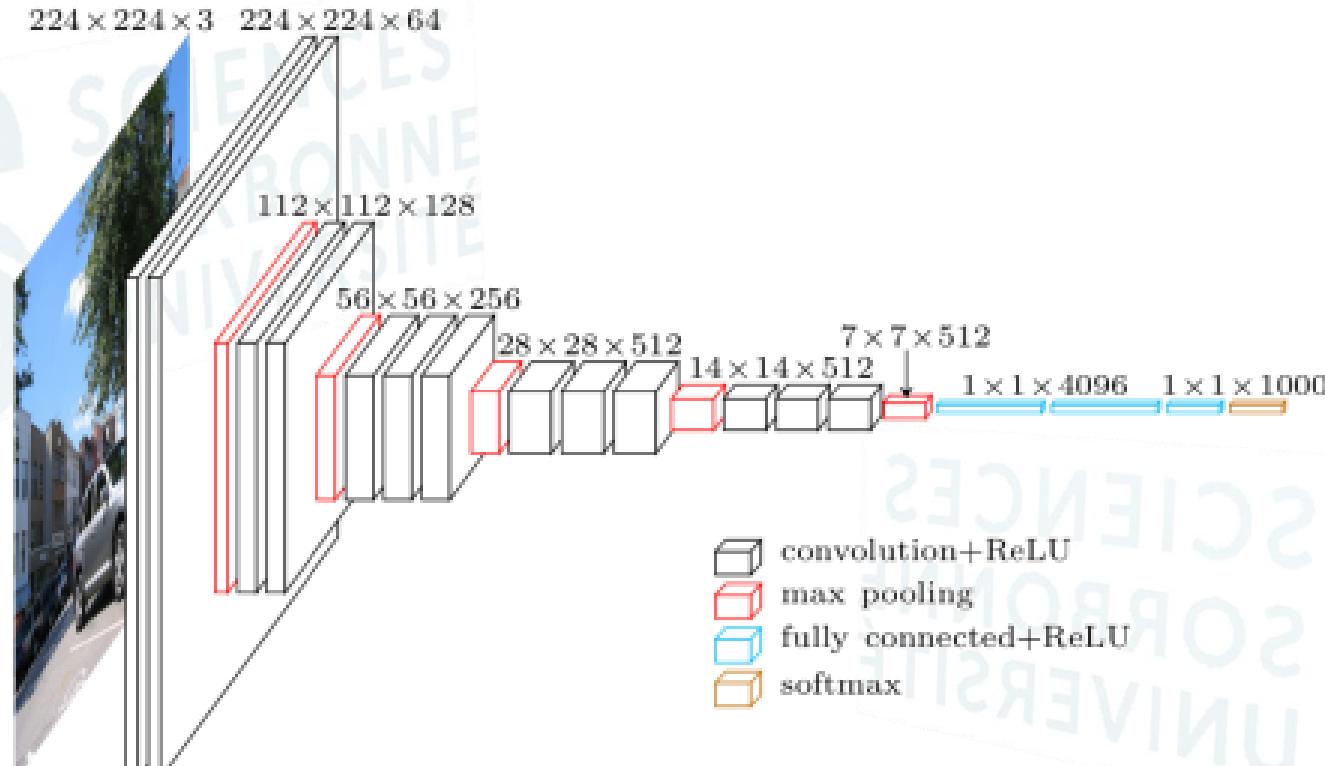
Appris sur **ImageNet** :

138.000.000 de paramètres à estimer
1.200.000 images en apprentissage
1000 classes d'images à reconnaître

- Taille du batch=256
- Tous les neurones sont équipés de la fonction d'activation **RELU**
- Descente de gradient avec **momentum** ($m=0.9$)
- **Dropout** lors de l'apprentissage des deux premières couches entièrement connectés
- Pas d'apprentissage initial = 0,01. Il décroît d'un facteur 10 trois fois pour arriver à 0,00001 à la fin
- Appris sur imagenet (3 semaines sur 4 GPU)

* L'architecture VGG16

Visualiser les filtres de convolution



On ne visualise pas les filtres eux mêmes mais les images qui maximisent la réponse de chaque filtre :

- On choisit une fonction perte qui maximise la valeur d'un filtre
- On part d'une image blanche
- On fait une descente de gradient pour modifier l'image d'entrée

* L'architecture VGG16

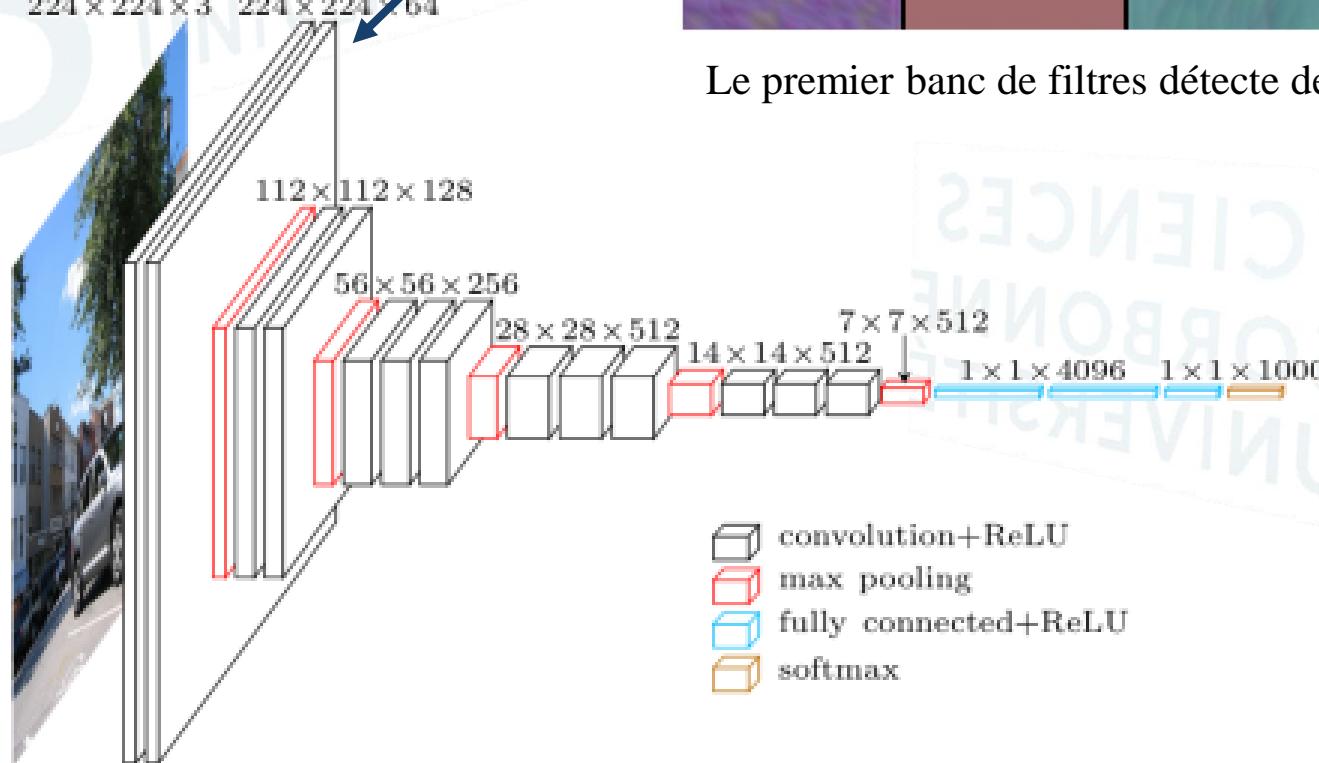
Exemple : VGG16

8 filtres (parmi 64) de la première couche de convolution

$224 \times 224 \times 3$ $224 \times 224 \times 64$



Le premier banc de filtres détecte des couleurs



<https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>

* L'architecture VGG16

Exemple : VGG16

8 filtres (parmi 128) de la seconde couche de convolution

$224 \times 224 \times 3$ $224 \times 224 \times 64$

$112 \times 112 \times 128$

$56 \times 56 \times 256$

$28 \times 28 \times 512$

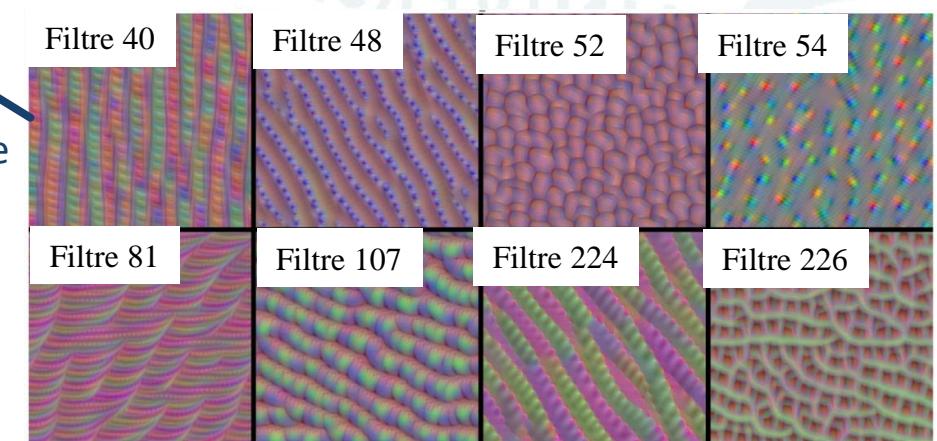
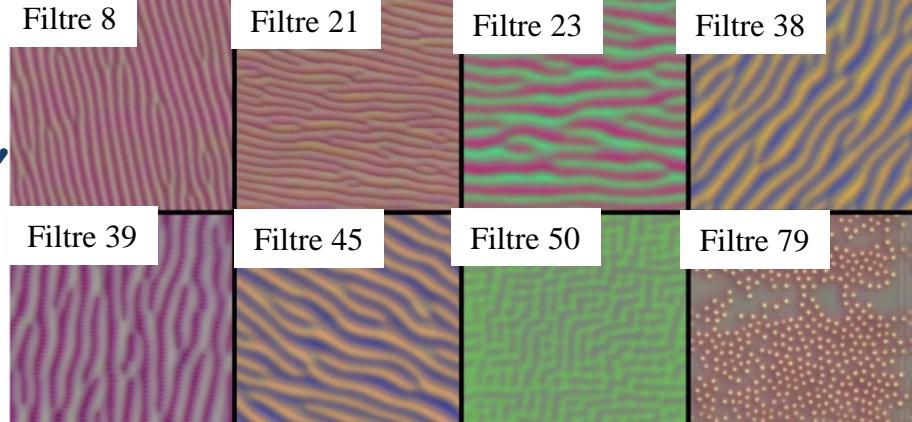
$14 \times 14 \times 512$

$7 \times 7 \times 512$

$1 \times 1 \times 4096$

$1 \times 1 \times 1000$

8 filtres (parmi 256) de la 3ième couche de convolution



* L'architecture VGG16

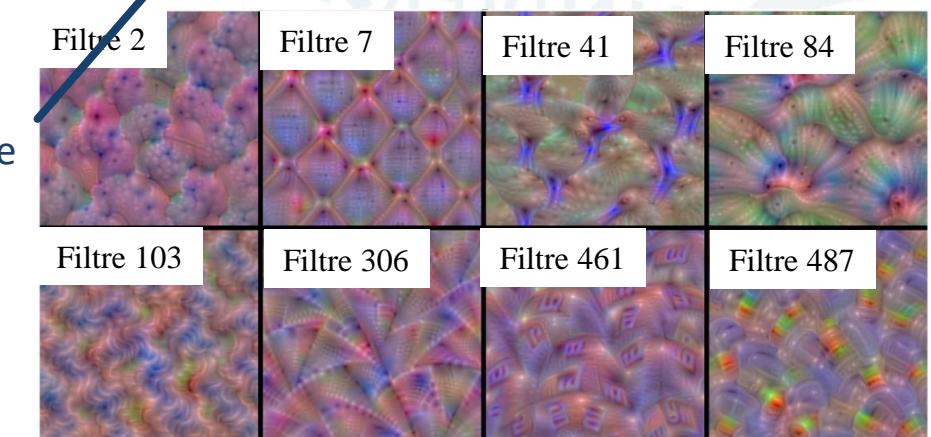
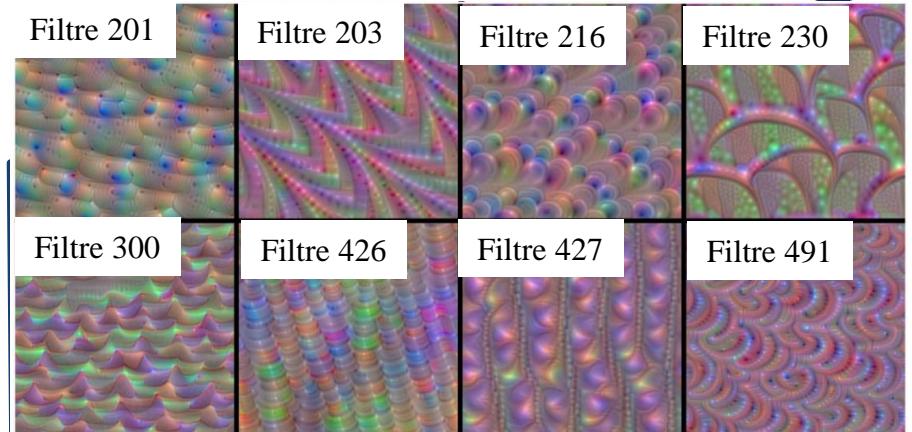
Exemple : VGG16

8 filtres (parmi 512) de la 4^{ème} couche de convolution

$224 \times 224 \times 3$ $224 \times 224 \times 64$

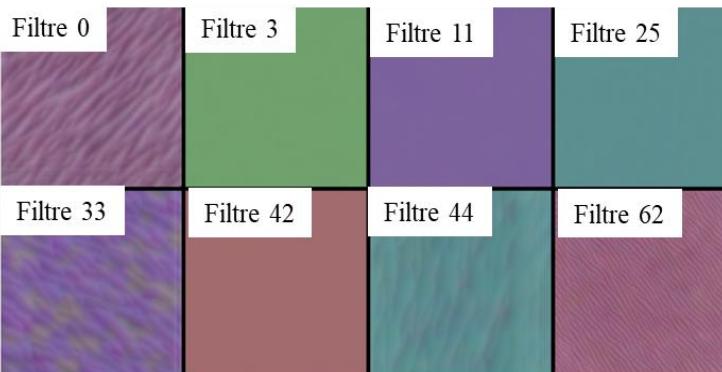


8 filtres (parmi 512) de la 5^{ème} couche de convolution

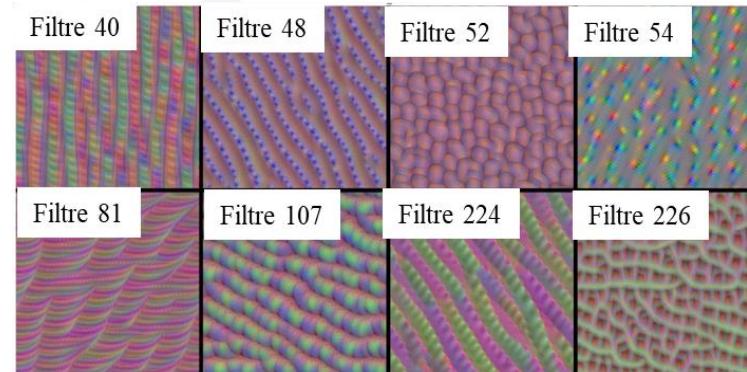


* L'architecture VGG16

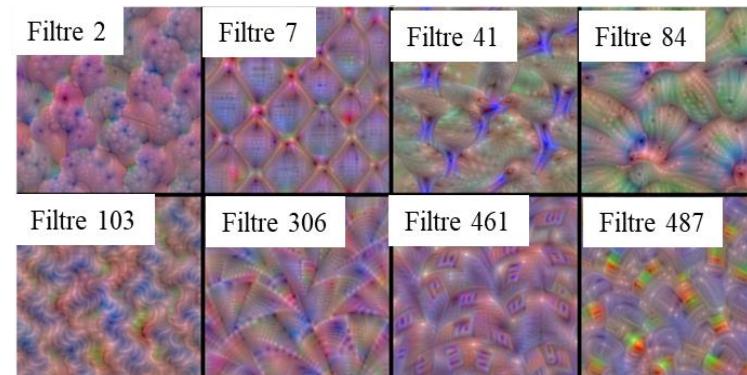
Couche 1



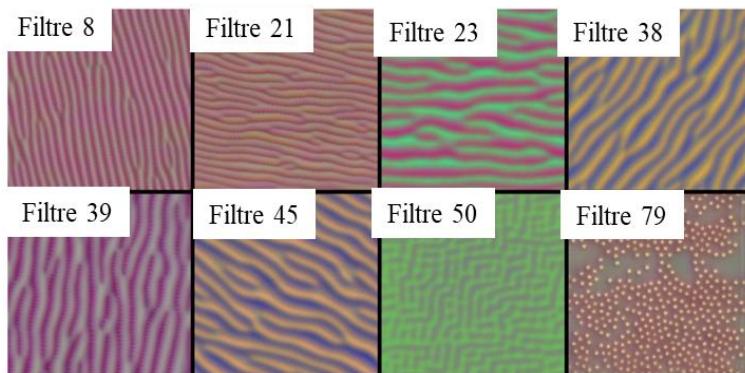
Couche 3



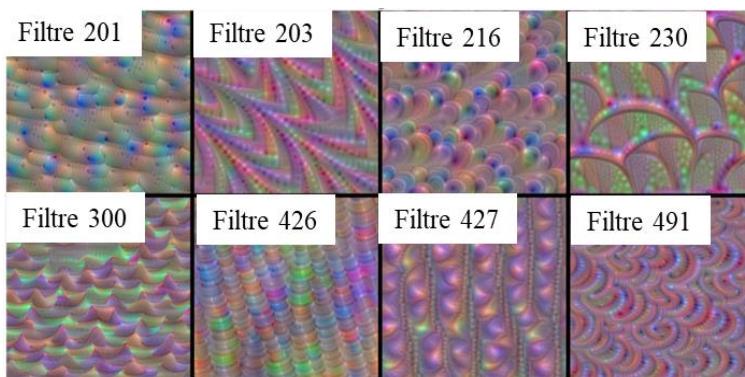
Couche 5



Couche 2



Couche 4



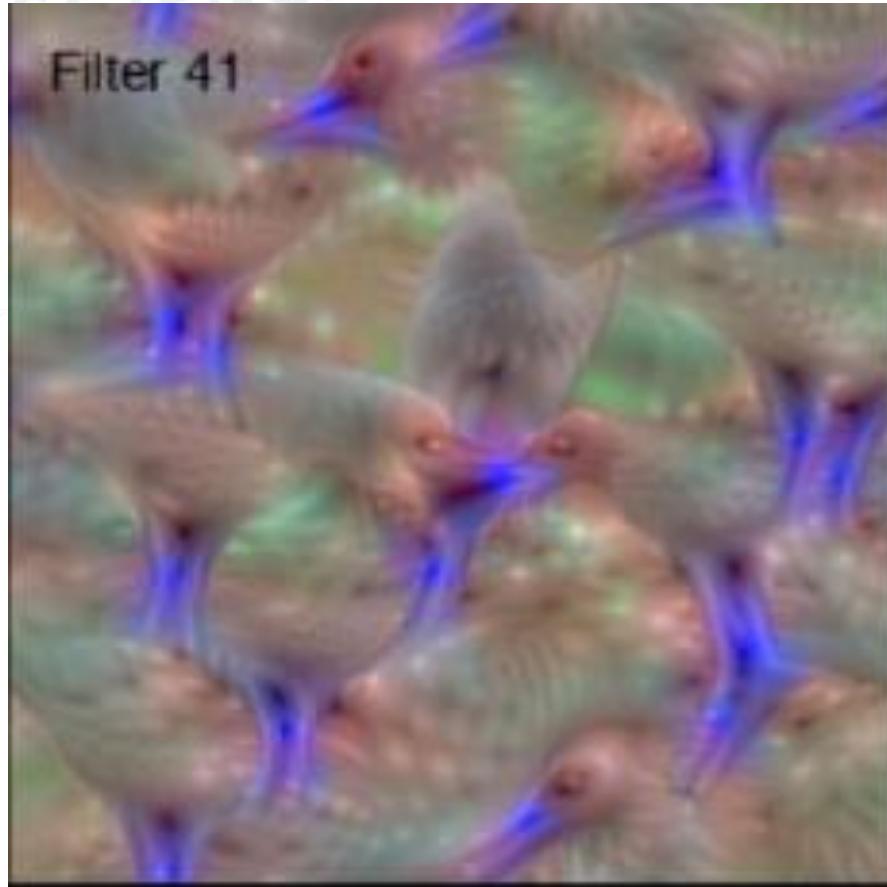
Les couches 1 et 2 détectent principalement des couleurs, contours, formes simples

Qui on s'enfonce dans le réseau, on voit des formes de plus en plus complexes

Par exemple, le filtre 41 du bloc 5 semble encoder des têtes d'oiseaux

<https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>

* L'architecture VGG16



Transfert learning

Fine tuning ou transfer learning

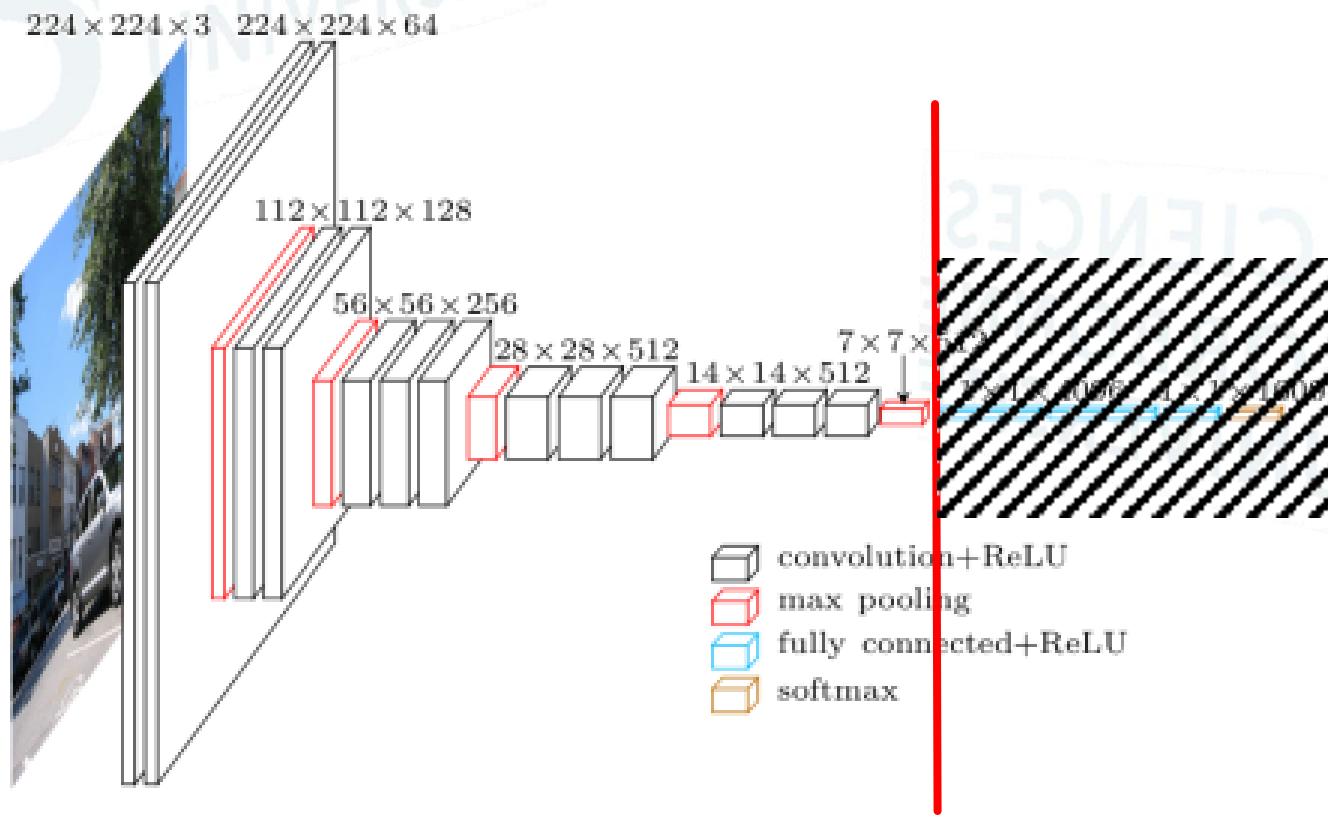
But: Profiter des réseaux déjà existants (ils mettent 2 à 3 semaines à être appris sur de multiple GPU)

Le Fine Tuning consiste à utiliser un réseau déjà entraîné comme initialisation pour un nouveau problème.

- ➔ On profite d'un réseau déjà appris sur des millions d'images comme extracteur de caractéristiques
- ➔ Il suffit de changer les dernières couches entièrement connectées pour les réapprendre sur le nouveau problème
- ➔ Lors de l'entraînement du nouveau réseau, on peut
 - Geler les couches initiales et apprendre seulement les nouvelles couches
 - Utiliser un faible taux d'apprentissage pour les couches déjà apprises

Exemple avec l'architecture VGG :

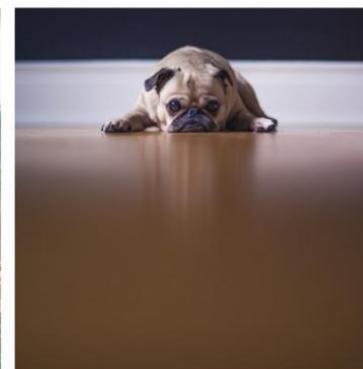
On récupère toute la première partie du réseau et on apprend juste les couches fully connected



Architecture GoogleNet

Inception : autre architecture de réseau

- Les parties intéressantes de l'image peuvent beaucoup varier en taille
- Choisir la taille des filtres ou leur nombre est difficile
- Des réseaux très profonds sont sujets à l'overfitting



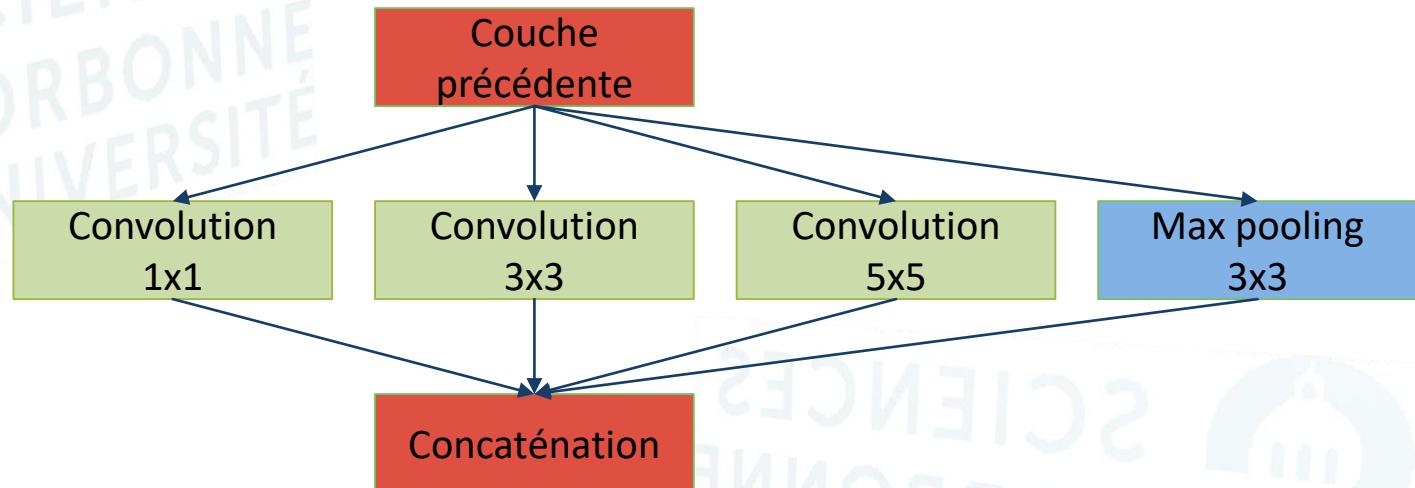
<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

Pourquoi pas des filtres de taille différente au même niveau ?

Le réseau devient plus large mais pas plus profond (problème du vanishing gradient)

* L'architecture GoogleNet - Inception

D'où la première idée naïve

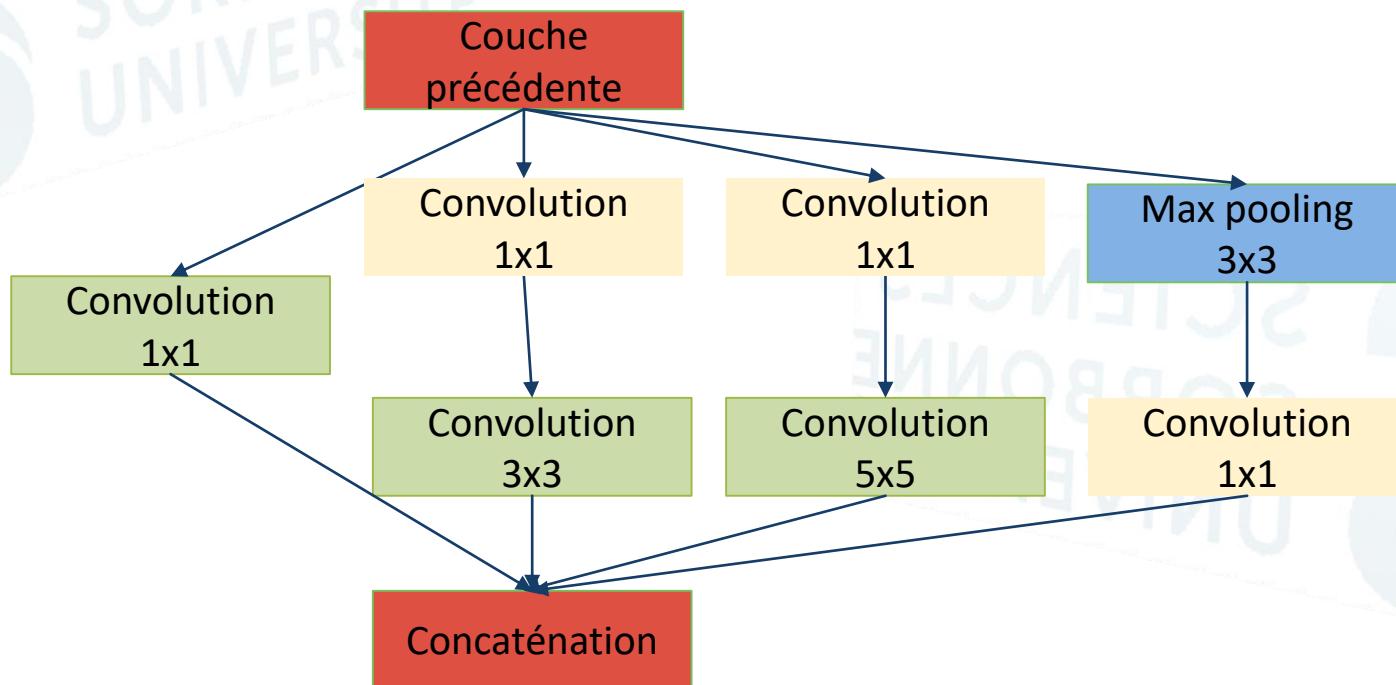


Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*

➔ Beaucoup de paramètres à estimer

Pour limiter le nombre de map d'entrée (et donc le nombre de coefficients à estimer) on ajoute une **convolution 1x1**

D'où le module **Inception**



Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*

* L'architecture GoogleNet - Inception

Inception : autre architecture de réseau

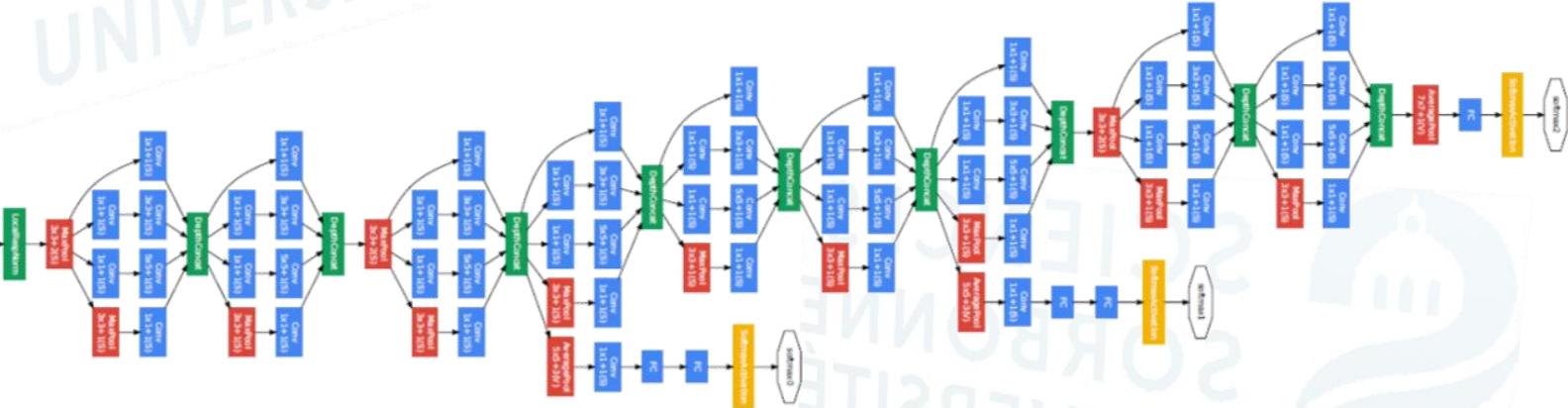
D'où le réseau GoogleNet

Convolution

Concatenation

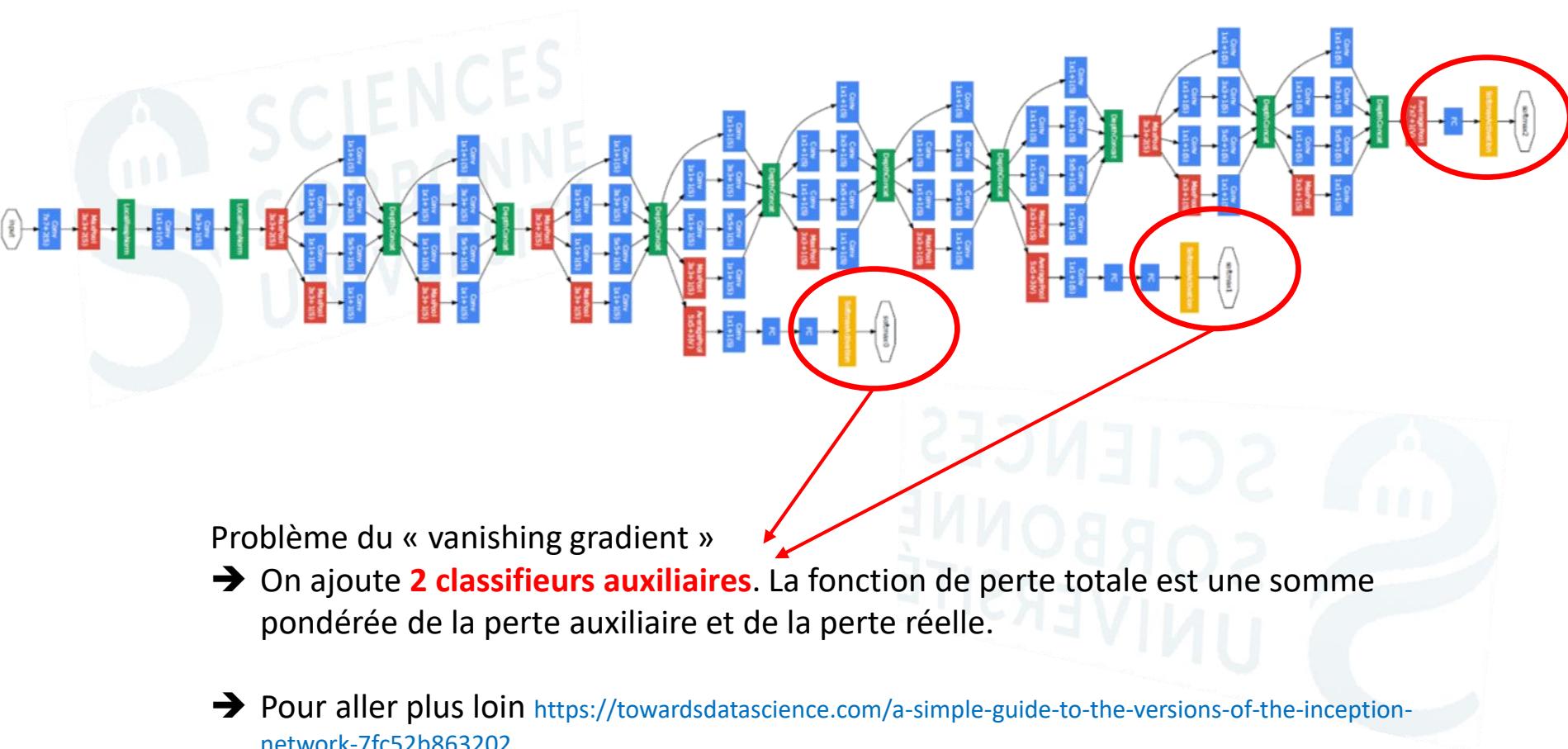
Pooling

Soft-max



Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*

* L'architecture GoogleNet - Inception



* L'architecture GoogleNet - Inception

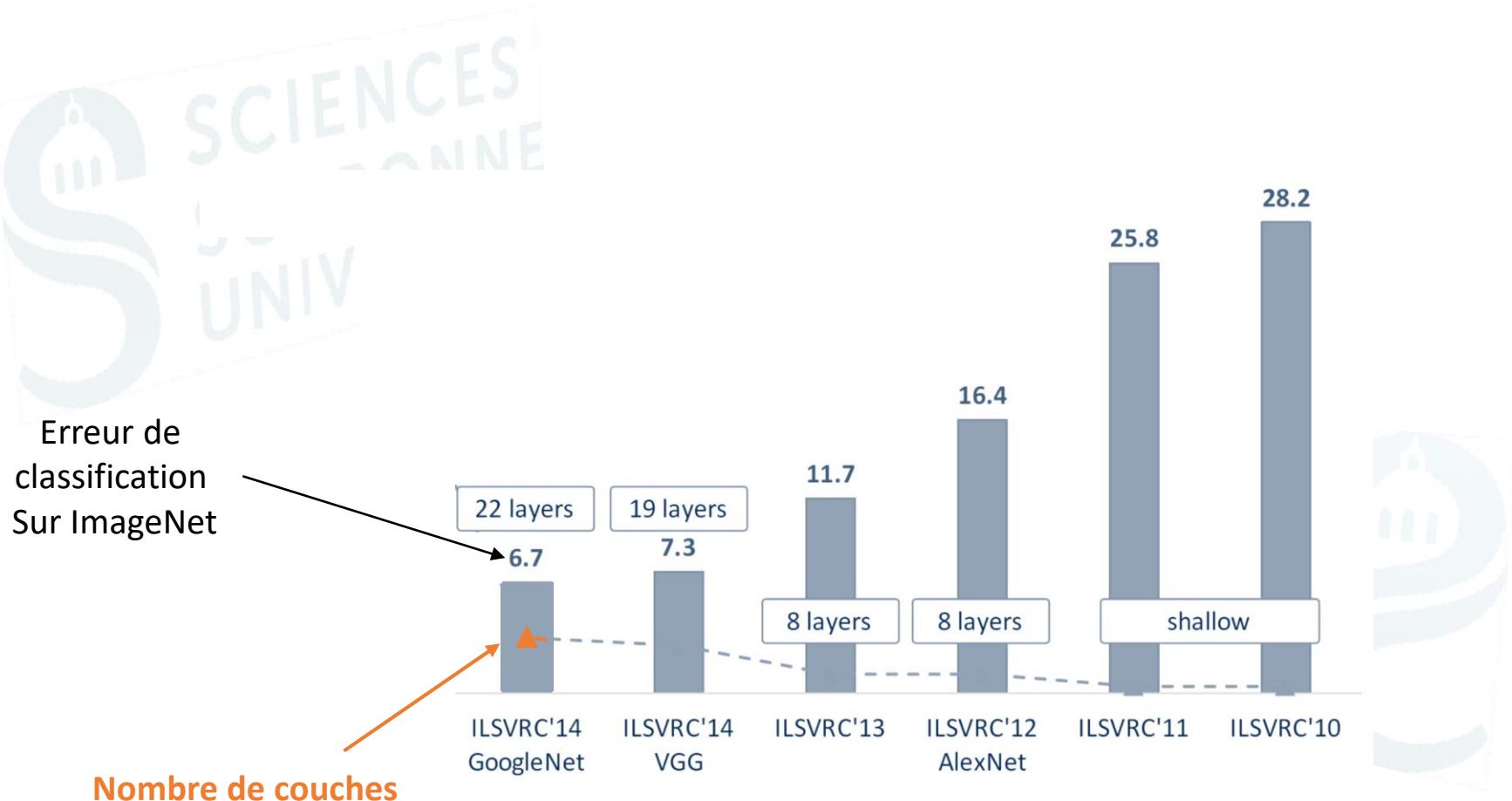


Image issue de : a tutorial on Deep Residual Networks - Kaiming He - ICML 2016.

Architecture Resnet

Autre architecture de réseau Resnet

- Quand la profondeur des réseaux augmente, la précision sature puis diminue rapidement
- Ce n'est **pas dû au sur-apprentissage** quand on travaille sur de très grandes bases de données : l'erreur augmente aussi sur la base d'apprentissage
- Probablement dû au **problème de la disparition du gradient** (vanishing gradient). Solution par RELU, DROPOUT, et pour aller plus loin Resnet
- En partant d'un réseau peu profond aux bons résultats, on peut construire un réseau plus profond en gardant les couches du premier réseau et en ajoutant des couches identités ➔ on ne devrait pas augmenter l'erreur
- Mais, on ne sait pas faire de couches identités avec des fonctions non linéaires

* L'architecture Resnet

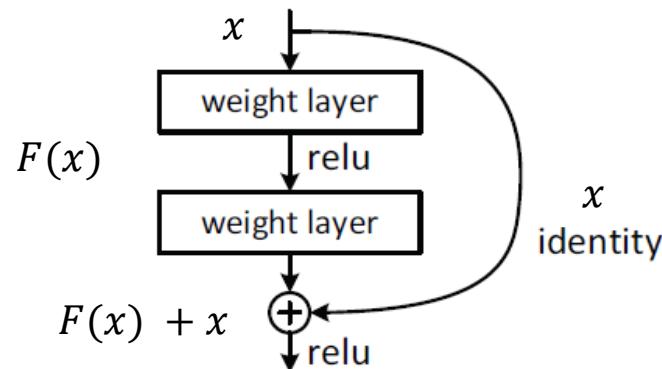
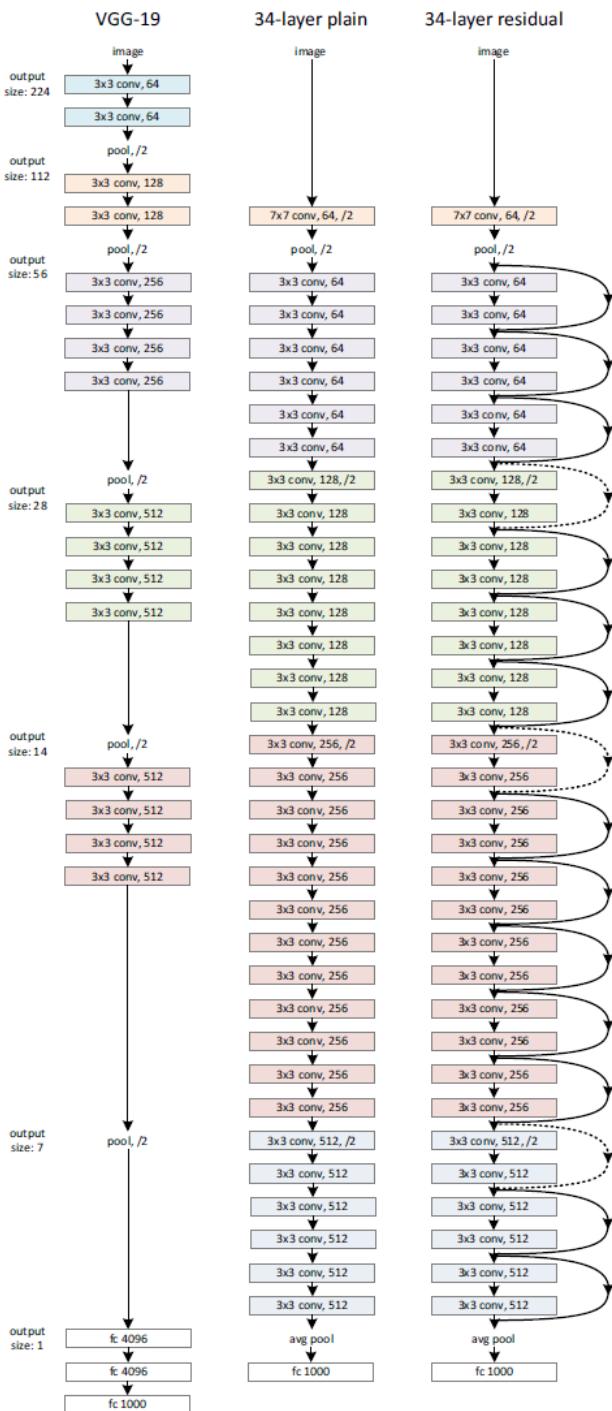


Image issue de : He, K., Zhang, X., Ren, S., & Sun, J. Deep residual learning for image recognition. CVPR2016

- Les couches apprennent une fonction résiduelle

$$H(x) = F(x) + x$$
 où $H(x)$ est la fonction désirée
- On suppose que cet apprentissage est plus simple : à l'extrême, on pourra forcer $F(x)$ à zéro pour prendre la connexion directe
- En ajoutant la connexion directe (**skip connexion**), on n'ajoute pas de paramètres à apprendre

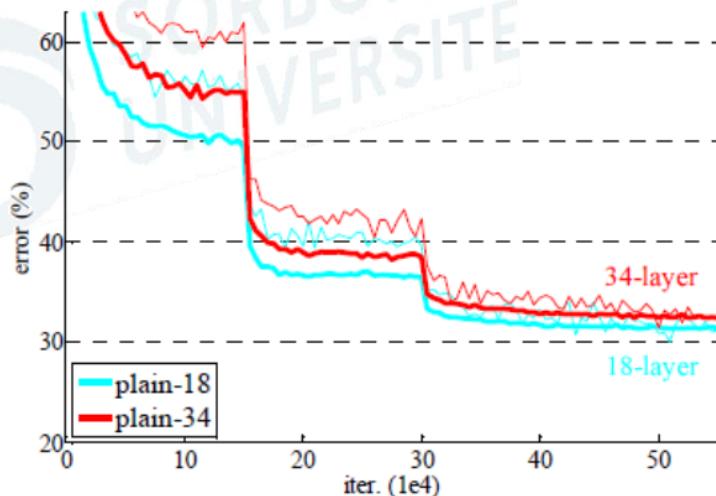


Comparaison de 3 architectures sur ImageNet:

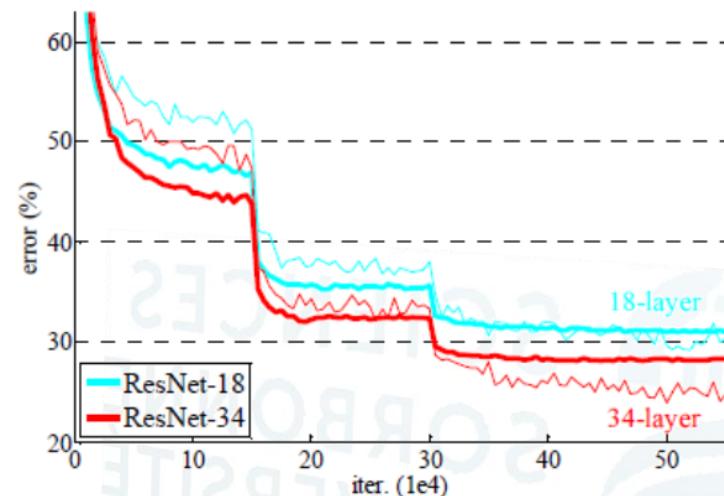
- VGG 19 couches
- 34 couches pleines
- 34 couches résiduelles

* L'architecture Resnet

Trait fin : erreur en apprentissage
 Trait gras : erreur en test



Couches pleines
 18 (VGG) ou 34 couches



Couches résiduelles
 18 ou 34 couches

Image issue de : He, K., Zhang, X., Ren, S., & Sun, J. Deep residual learning for image recognition. CVPR2016

→ Resnet : la révolution de la profondeur

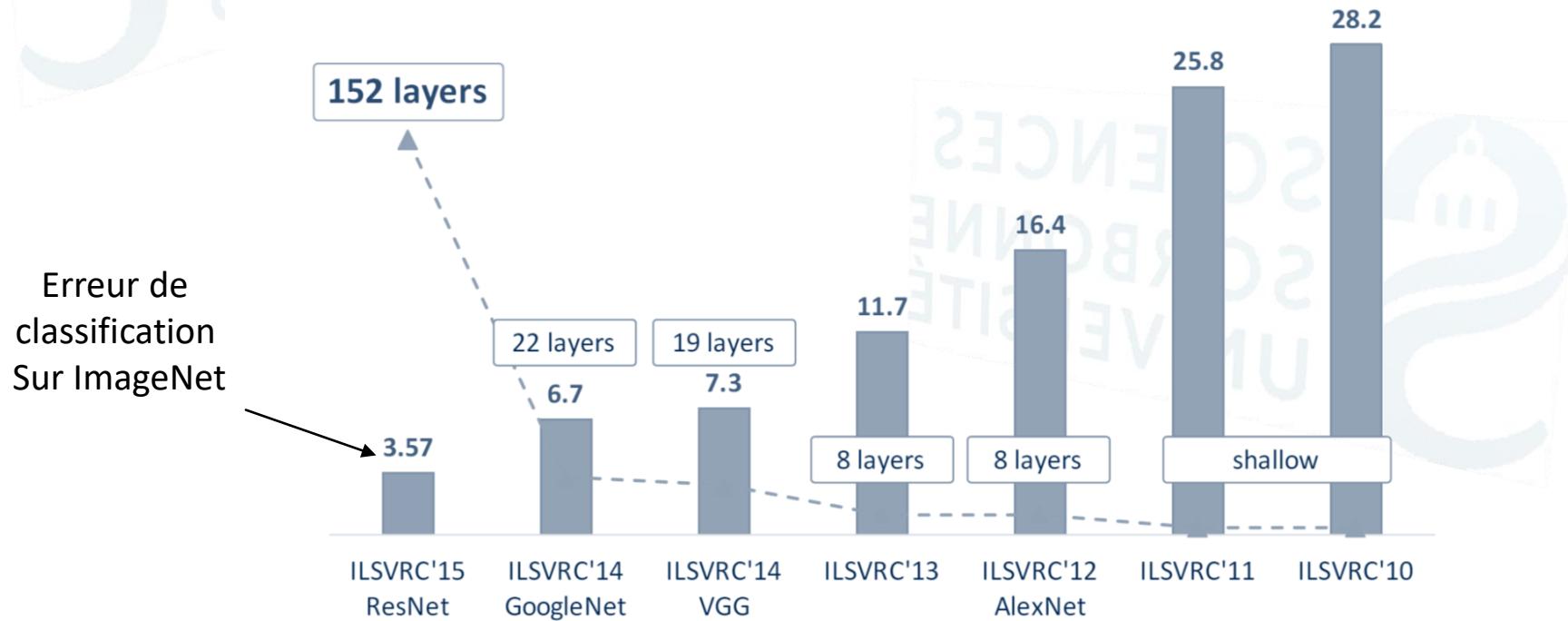


Image issue de : a tutorial on Deep Residual Networks - Kaiming He - ICML 2016.

Les grands jeux de données

*Les grands jeux de données

ImageNet

<http://www.image-net.org/>

La base est organisée en « synonym set » or « **synset** » : ensemble de mots ou phrases décrivant un concept compréhensible ou catégorie. Ils peuvent être décomposés en sous-catégories

Statistiques d'avril 2010

- Nombre de synsets: **21,841**
- Nombre total d'images : **14,197,122**
- Nombre de d'images avec boite englobante annotée: **1,034,908**

- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. *CVPR 2009*.
- Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015

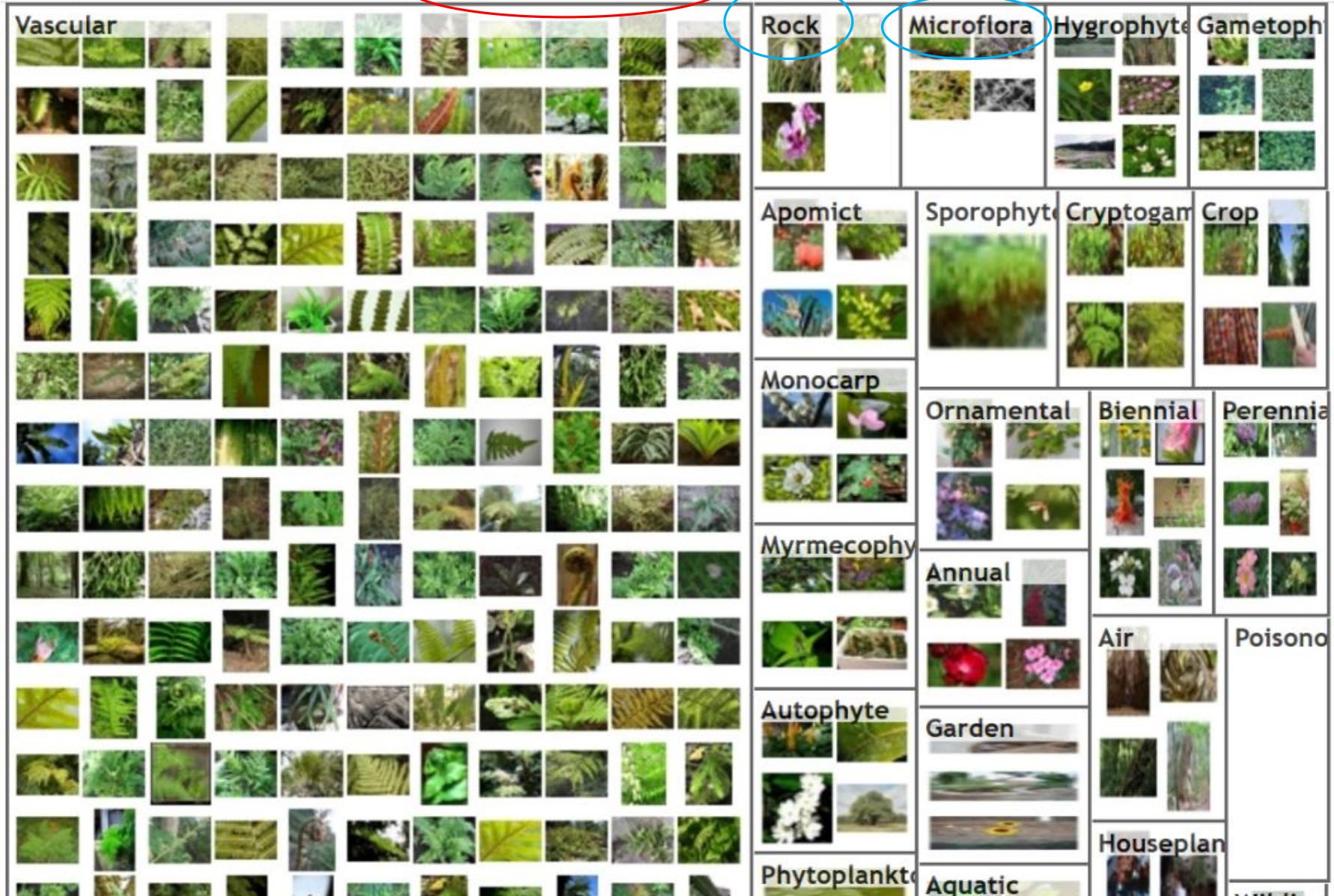
Les grands jeux de données



ImageNet 2011 Fall Release

Plant, flora, plant life

Catégorie et sous-catégories



Challenges

- **Localisation d'objet** : L'algorithme produit 5 classes d'objets et 5 boîtes englobantes. La qualité de la localisation est évaluée sur l'objet qui « matche » le mieux à la vérité
- **Détection d'objets** : L'algorithme produit un ensemble de classes d'objets et leur boîte englobante. L'ensemble doit contenir toutes les instances des objets présents dans l'image. Pénalisation si oubli ou détection multiple
- **Détection d'objets à partir de vidéo** : même chose sur des vidéos
- **Classification de scène** : L'algorithme doit retourner les 5 catégories de scènes qui correspondent le plus à une image donnée

- J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database. *CVPR 2009*.
- Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 2015

MS coco

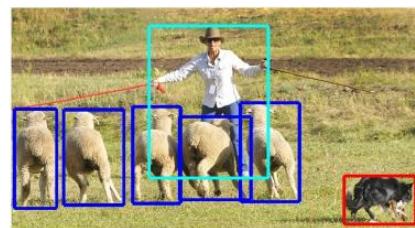
<http://cocodataset.org>

Statistiques

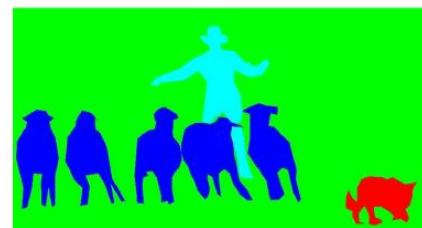
- 91 type d'objets
- Chaque objet est étiqueté avec sa segmentation pour une localisation précise
- 2,500,00 millions d'instances d'objet
- 328,000 images



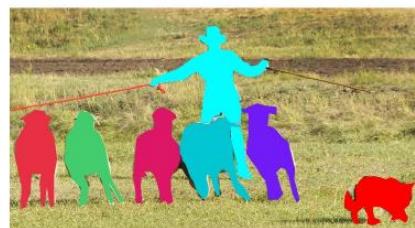
(a) Image classification



(b) Object localization



(c) Semantic segmentation



(d) This work

- Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740-755). Springer, Cham.

Coco challenges

- Détection d'objet (boîte englobante ou région)



- Détection de points d'intérêt



- Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014, September). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740-755). Springer, Cham.

La segmentation d'images

Segmentation d'images

Segmentation sémantique



predict →

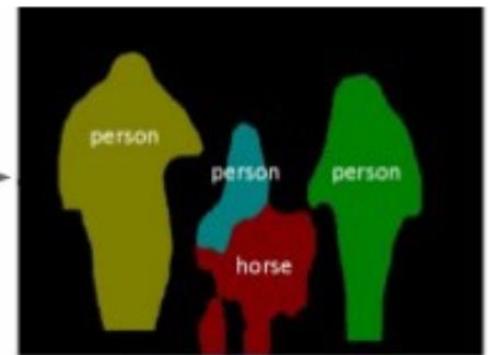


Person
Bicycle
Background

Segmentation d'instance



→



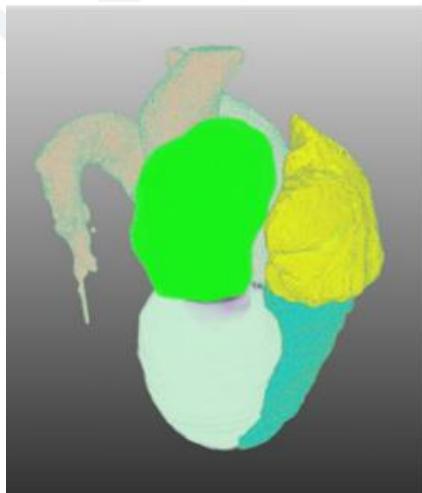
<https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>

U-net

Convolutional networks for biomedical image segmentation

Ronneberger, O., Fischer, P., & Brox, T. (2015, October).. In *International Conference on Medical image computing and computer-assisted intervention*

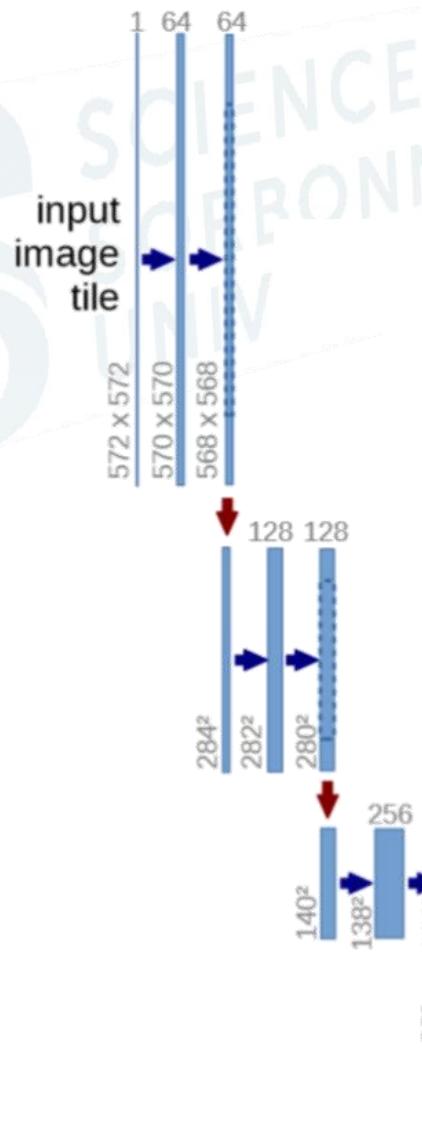
Idée :



Pour segmenter une image, on a besoin
d'informations locales, au niveau du pixel mais aussi
d'un **contexte global**

→ **Solution**

Approche multi-résolution



Branche descendante

- 2 convolutions 3x3 (sans padding) + RELU :
 - on double le nombre de filtres à chaque étage
- max pooling 2x2 avec un stride de 2
 - On divise la résolution par 2 à chaque étage

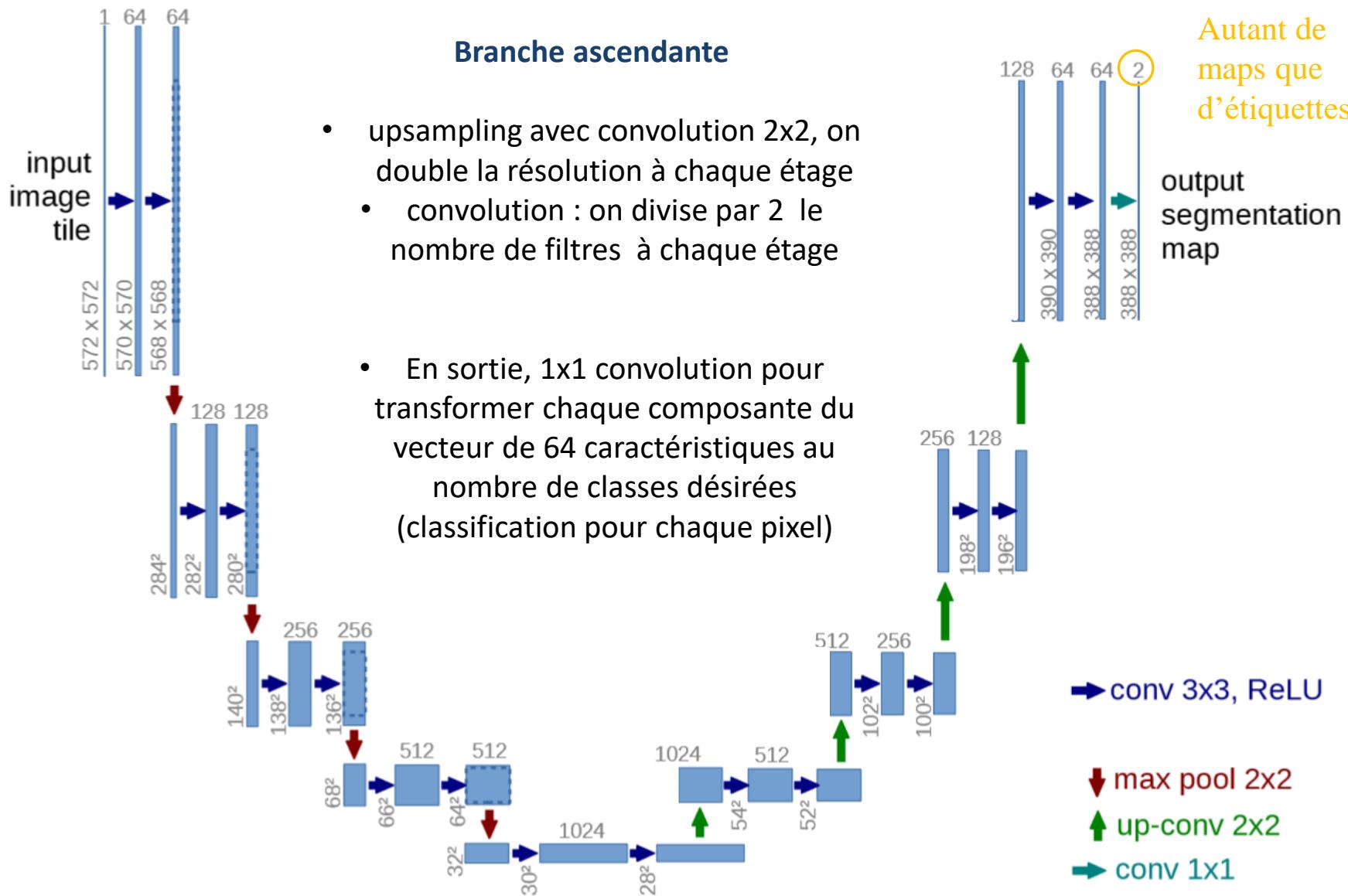
On arrive à une carte de $32 \times 32 \times 512$.

Chaque pixel de la carte 32×32 est encodé par 512 caractéristiques qui décrivent le contexte global

Problème

La segmentation va avoir une très mauvaise résolution

Segmentation d'images



Segmentation d'images

Sortie : Autant de maps que d'étiquettes
Quelle fonction perte ?

Segmentation d'images

Sortie : Autant de maps que d'étiquettes
Quelle fonction perte ?

Pour chaque pixel, on a un problème de classification → softmax puis cross-entropy

Rq : Beaucoup d'autres loss existent

Segmentation d'images

Branche ascendante - upsampling

Mais comment fonctionne l'upsampling ?

La **convolution transposée** par un filtre 2x2 consiste à calculer, pour chaque position de l'image d'entrée, une matrice de taille 2x2 en multipliant l'entrée par le filtre

Exemple :

0	1
2	3

transposed convolution
sans padding avec, stride = 1 :

1	1
1	1

$$= \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array} + \begin{array}{|c|c|} \hline & 1 & 1 \\ \hline & 1 & 1 \\ \hline \end{array} + \begin{array}{|c|c|} \hline & & \\ \hline 2 & 2 \\ \hline 2 & 2 \\ \hline \end{array} + \begin{array}{|c|c|} \hline & & \\ \hline 3 & 3 \\ \hline 3 & 3 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 0 & 1 & 1 \\ \hline 2 & 6 & 4 \\ \hline 2 & 5 & 3 \\ \hline \end{array}$$

Segmentation d'images

Branche ascendante - upsampling

En faisant la même chose avec un stride de 2 ,

0	1
2	3

transposed convolution
sans padding, stride = 2 :

1	1
1	1

$$= \begin{array}{|c|c|c|c|} \hline 0 & 0 & & \\ \hline 0 & 0 & & \\ \hline & & 1 & 1 \\ \hline & & 1 & 1 \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & 2 & 2 \\ \hline & & 2 & 2 \\ \hline \end{array} + \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & & & \\ \hline & & 3 & 3 \\ \hline & & 3 & 3 \\ \hline \end{array}$$

$$= \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 1 & 1 \\ \hline 2 & 2 & 3 & 3 \\ \hline 2 & 2 & 3 & 3 \\ \hline \end{array}$$

On a bien transformé la matrice d'entrée de taille 2x2 en une matrice de sortie de taille 4x4

Segmentation d'images

Branche ascendante - upsampling

Exercice

Réaliser

2	1
4	4

transposed convolution
sans padding, stride =1 :

1	4	1
1	4	3
3	3	1

Segmentation d'images

Branche ascendante - upsampling

Exercice

Réaliser

2	1
4	4

transposed convolution
sans padding, stride =1 :

1	4	1
1	4	3
3	3	1

2	8	2	0
2	8	6	0
6	6	2	0
0	0	0	0

0	1	4	1
0	1	4	3
0	3	3	1
0	0	0	0

+

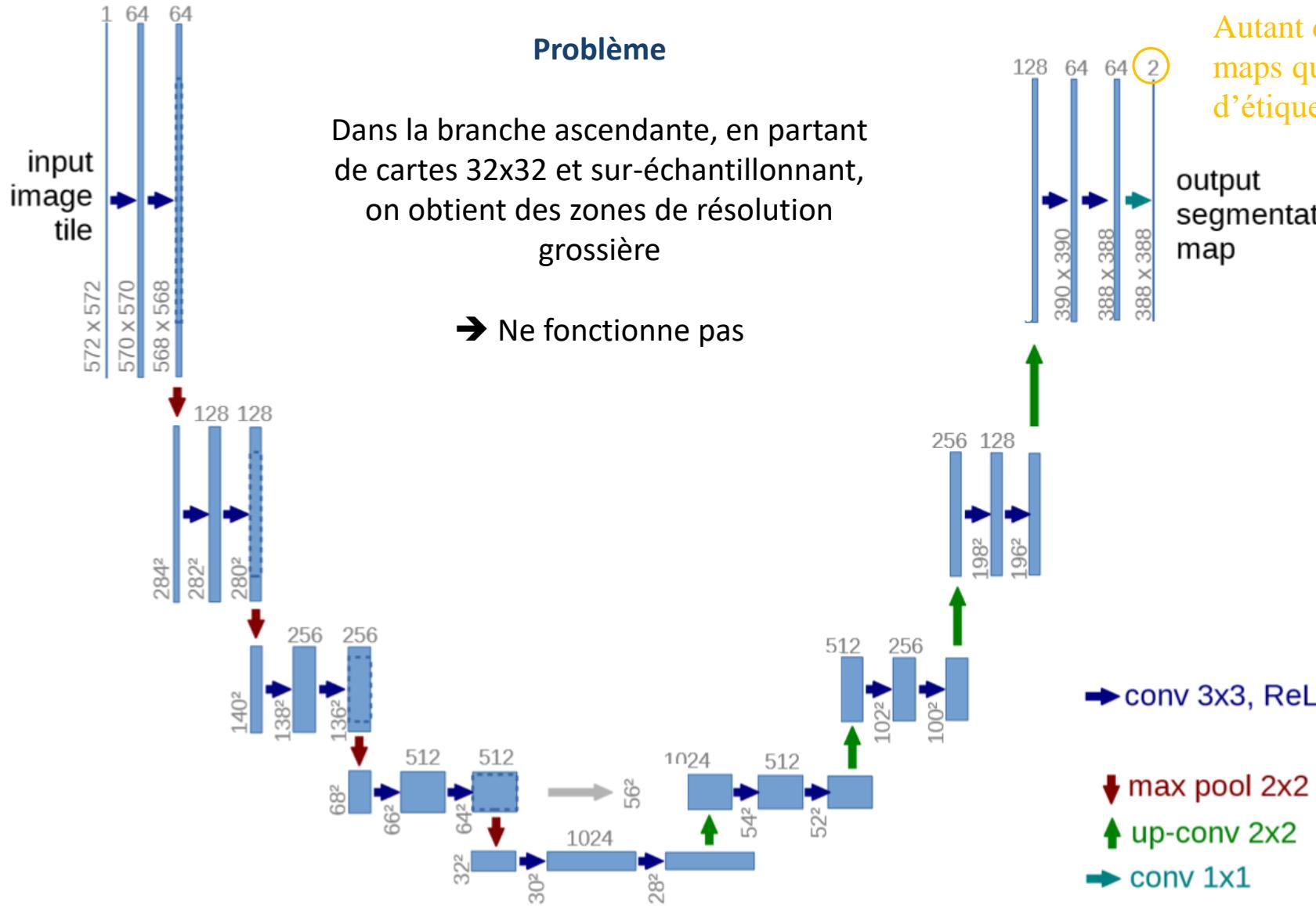
0	0	0	0
4	16	4	0
4	16	12	0
12	12	4	0

0	0	0	0
0	4	16	4
0	4	16	12
0	12	12	4

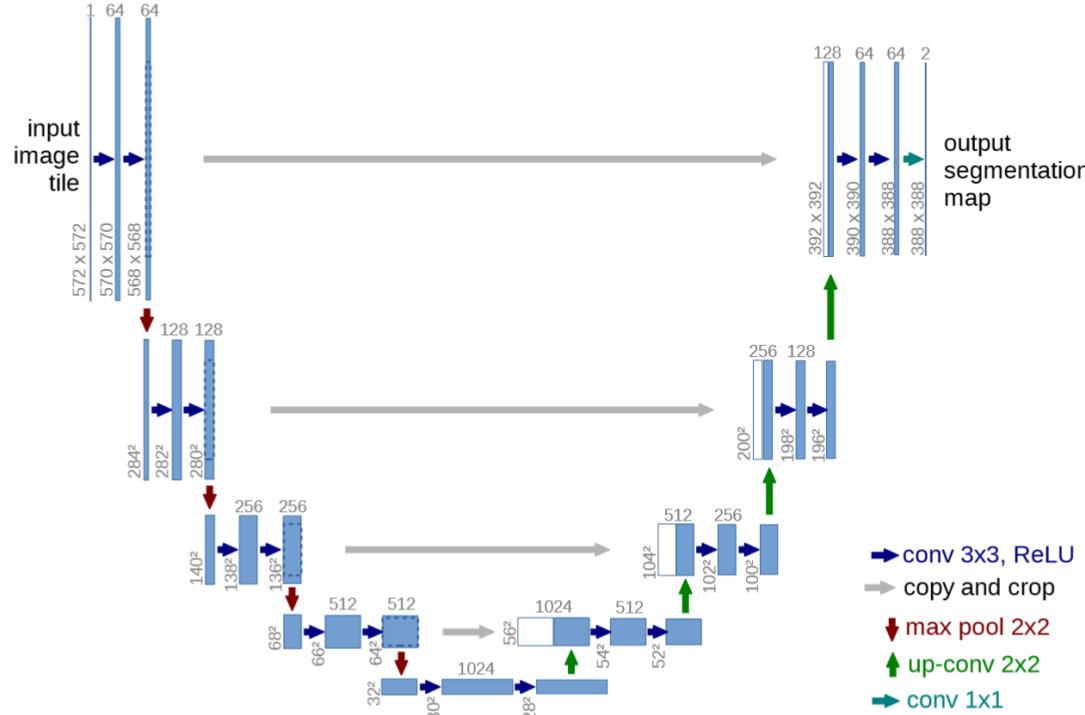


2	9	6	1
6	29	30	7
10	29	33	13
12	24	16	4

Segmentation d'images



Segmentation d'images



Branche ascendante

- concaténation avec la carte cropée correspondante de la branche descendante
 - On garde les informations sémantiques de la branche montante
 - On garde la précision au niveau des contours de la branche descendante

Segmentation d'images

Fin segmentation