ConvNet : les réseaux convolutionnels / convolutifs

Hélène Paugam-Moisy

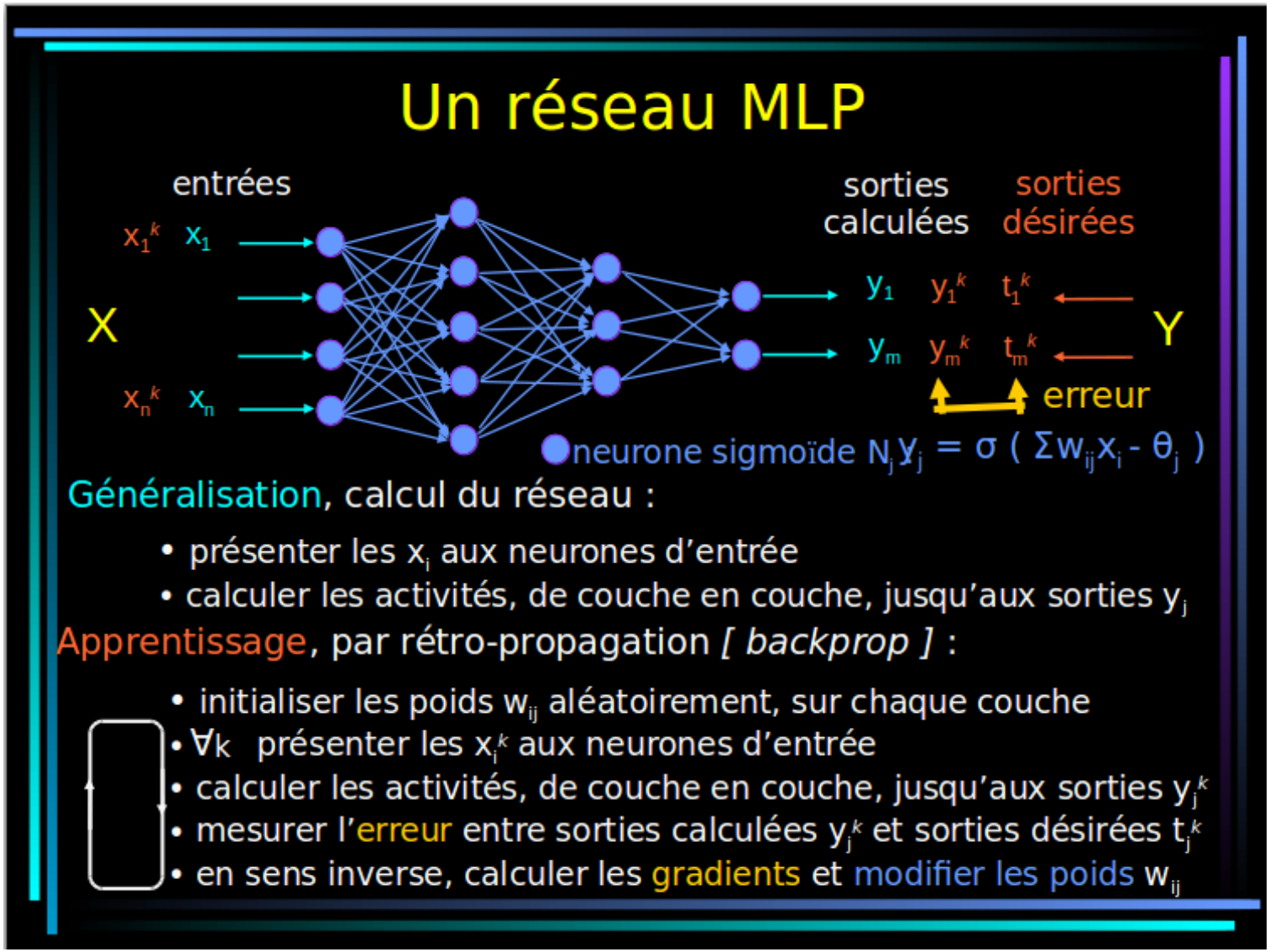
Université des Antilles - DMI - LAMIA

Séminaire LAMIA - 22 novembre 2018

Bref historique

- En 1960, le Perceptron voulait imiter le système visuel mais une seule couche de neurones à seuil ne peut faire que des séparations linéaires.
- D'une part, le système visuel est multicouche (LGN, aires V1, V2, aire MT...) et d'autre part un réseau multicouche peut faire des séparations non-linéaires.
- en 1985, on a utilisé des neurones sigmoïdes pour apprendre, par rétro-propagation du gradient d'erreur, les poids d'un réseau multicouche : le réseau MLP (pour *Multi-Layer Perceptron*).

Le réseau multicouche MLP



Des réseaux multicouches aux *Deep Networks*

Les *Deep Networks* sont des réseaux multicouches, qui sont... profonds.

shallow vs. *deep* architecture pour les réseaux *feedforward*

On savait depuis longtemps qu'il était plus facile de capter certaines non-linéarités avec plusieurs couches de petites tailles plutôt qu'avec une seule très grande couche (ex. parity).

mais.... comment adapter l'apprentissage ? (pb des minima locaux)

Il s'est avéré que la rétropropagation marchait mal dans les réseaux à 3, 4 ou plus... couches cachées, sauf si on savait "mettre de la connaissance dans le réseau" (ex. LeNet de LeCun).

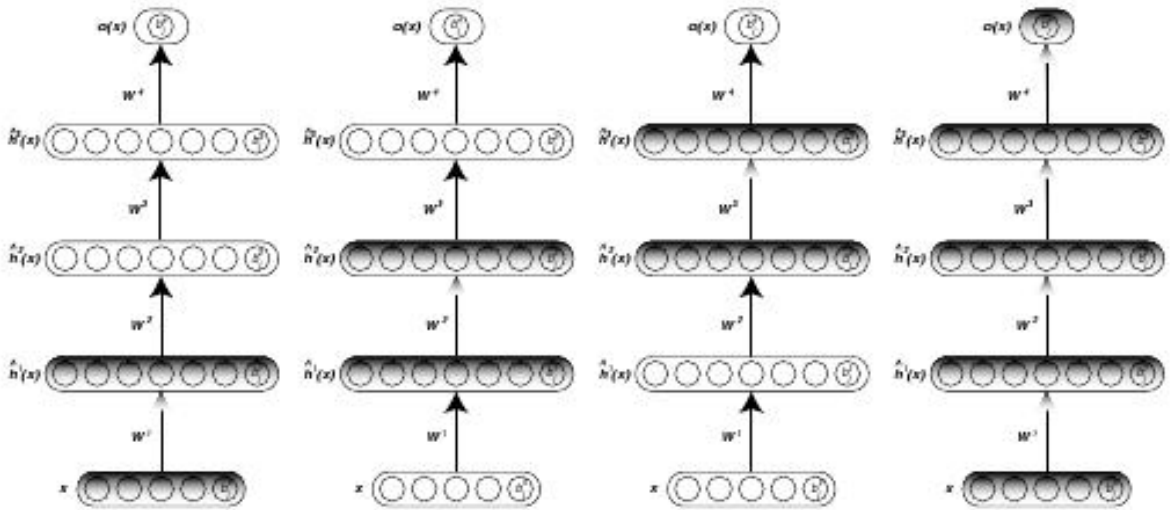
Deep networks : motivations

Pourquoi des "réseaux profonds" ?

- Capturer des non-linéarités d'ordre supérieur.
- Transformer les données d'entrée en une succession de *features* de plus en plus abstraites.
- Obtenir des réseaux ayant des architectures plus compactes.
- Pouvoir apprendre à partir d'un plus petit nombre d'exemples.
- Pouvoir inclure dans l'apprentissage des exemples sans étiquette.

Deep Learning : apprentissage dans un réseau profond

On apprend à reconstruire chaque couche successivement, avec des RBM ou des auto-associateurs, puis on opère une étape de *fine-tuning*, i.e. une rétro-propagation globale et supervisée sur l'ensemble des couches ainsi empilées.



(a) First hidden layer pre-training (b) Second hidden layer pre-training (c) Third hidden layer pre-training (d) Fine-tuning of whole network

Figure 2: Unsupervised greedy layer-wise training procedure.

Historique des "Deep Convolutional Networks"

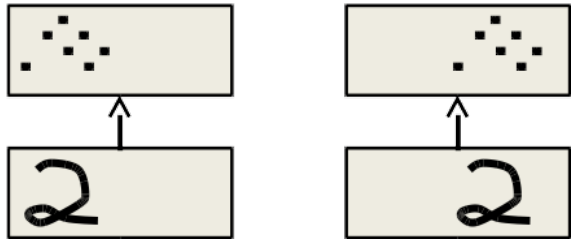
Motivations principales : reconnaissance de formes, fouille d'images réelles.

(inspired by Derek Hoiem, Lana Lazebnik or Jia-bin Huang)

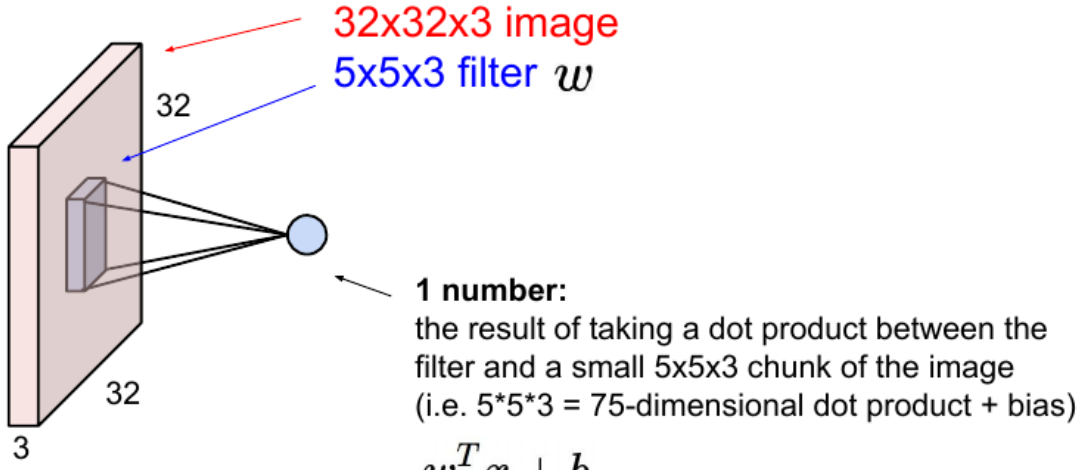
- 1950's : The first neural net (Perceptron) invented by Rosenblatt
- 1980's/1990's : MLP and backprop learning $\Rightarrow$ neural nets are popularized and then left behind as being interesting idea but too hard to optimize
- 1990's : LeCun achieves state-of-art performance on character recognition with convolutional network (main ideas of today's networks)
- 2000's : Hinton, Bottou, Bengio, LeCun, Ng, and others keep trying stuff with deep networks but without much traction/acclaim in vision
- 2010-2011 : Substantial progress in some areas, but vision community still unconvinced *[Some neural net researchers get angry at being ignored/rejected]*
- 2012 : "Shock" at ECCV 2012 with **ImageNet** challenge

Problème de l'invariance par translation

Les réseaux de neurones classiques activent des neurones différents si deux images représentent la même forme mais translatée dans son cadre :

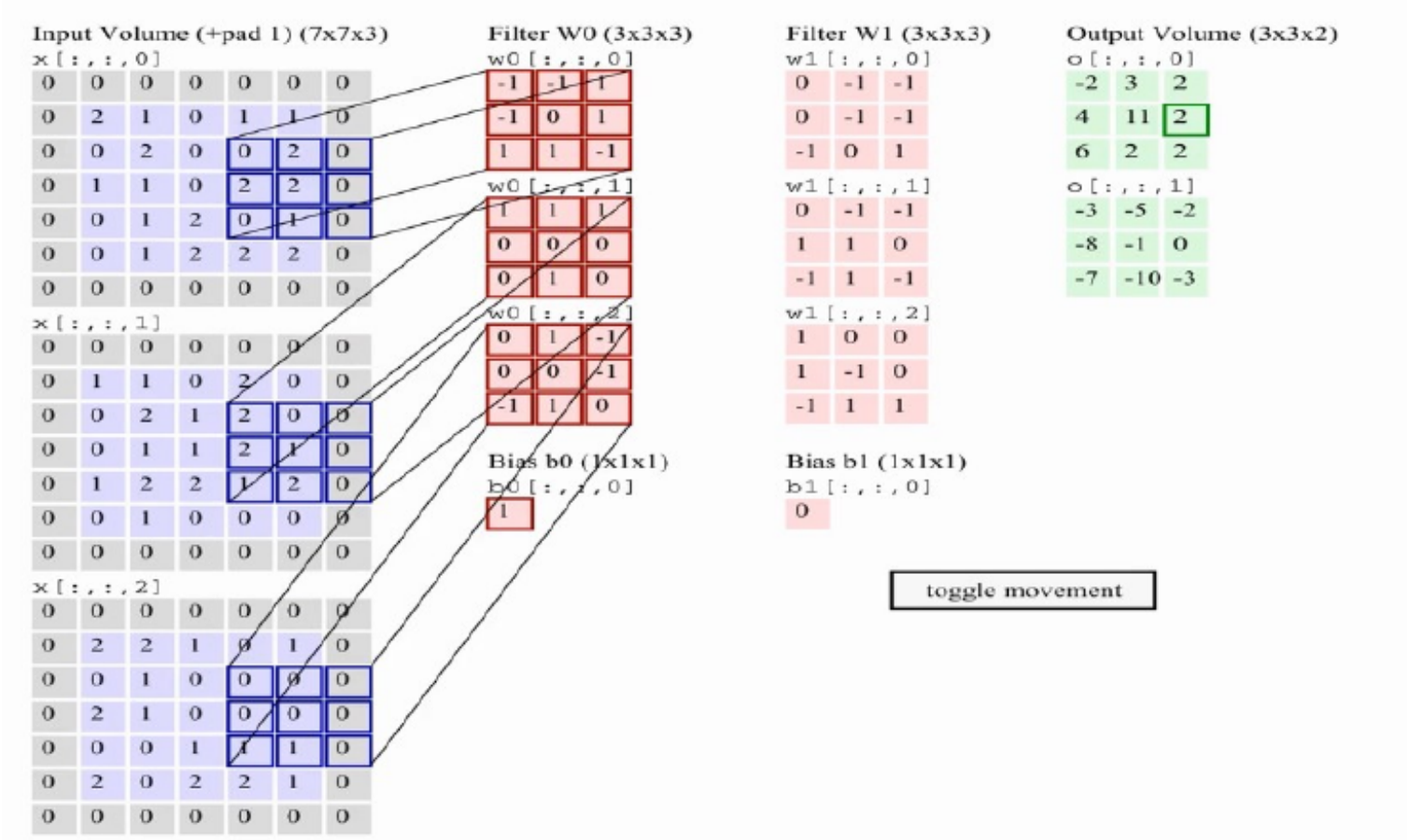


En s'inspirant des techniques du traitement d'image, on va balayer celle-ci avec un filtre convolutif, agissant comme un champ récepteur :



Convolutions multiples

On peut faire passer plusieurs filtres sur une même image : exemple d'une image couleur 5x5 pixels en représentation RVB (donc une épaisseur de 3).



[d'après LeCun, 2016]

Caractéristiques d'une couche de convolution

La couche de convolution est le bloc de construction de base d'un CNN.

Sa dimension dépend de trois paramètres :

- L'épaisseur de la couche : c'est le nombre de filtres
- La marge ou *zero padding* : le nombre de cadres emplis de 0 que l'on ajoute autour de l'image.
- Le pas ou *stride* : le décalage d'un filtre au suivant lors du balayage de l'image (marge et pas s'ajustent entre eux)

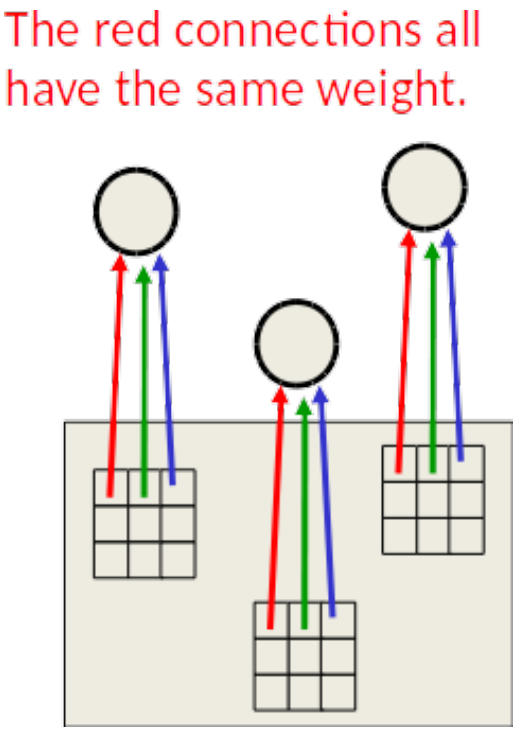
Dans l'exemple précédent, l'épaisseur vaut 2 (les deux filtres W_0 et W_1), la marge vaut 1 et, pour un pas de 2, on aura trois passages de filtre par ligne et par colonne.
Il en résulte une couche de convolution de $3x3x2$ valeurs (en vert, à droite).

Rôle de la couche de convolution

À l'instar du système visuel, une couche de convolution "capte" l'image au moyen de champs récepteurs, chacun ne prenant en compte qu'une petite vignette de l'image.

Du point de vue du réseau de neurones qui en résulte, celui-ci est à poids partagés puisque les poids proviennent des filtres : on retrouve les mêmes dans tous les calculs pour chaque neurone de la couche de convolution.

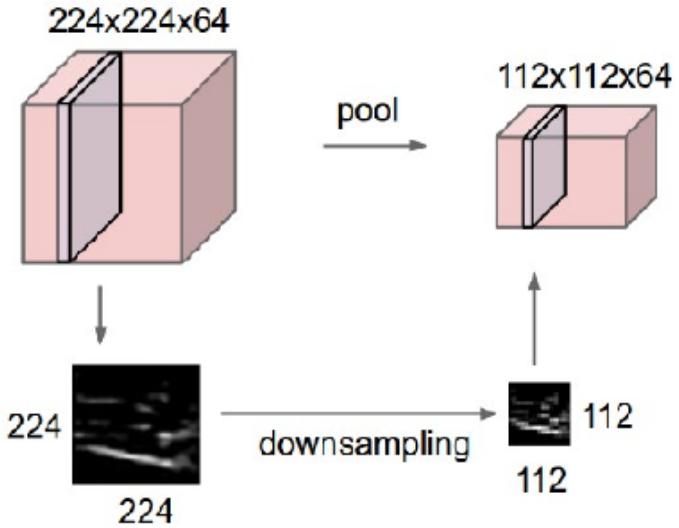
[Figure empruntée à Hinton]



Réduction de dimension

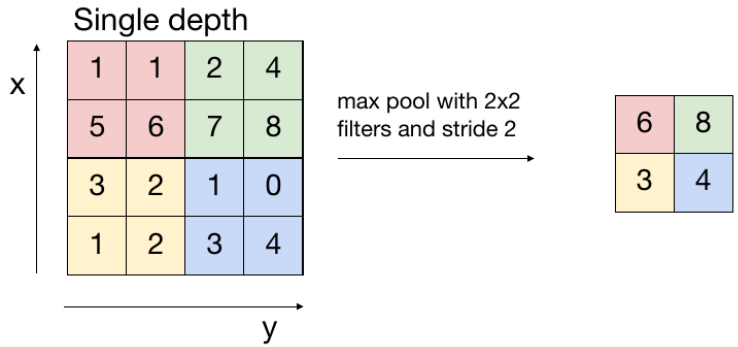
Un autre concept important des CNNs est le pooling qui est une forme de sous-échantillonnage par agrégation. L'objectif est de réduire le nombre de paramètres, et donc de calculs, dans le réseau. On peut, par exemple, l'insérer entre deux couches de convolution pour assurer la robustesse au bruit et aux distortions.

Le pooling fonctionne indépendamment sur chaque tranche et la redimensionne en surface.

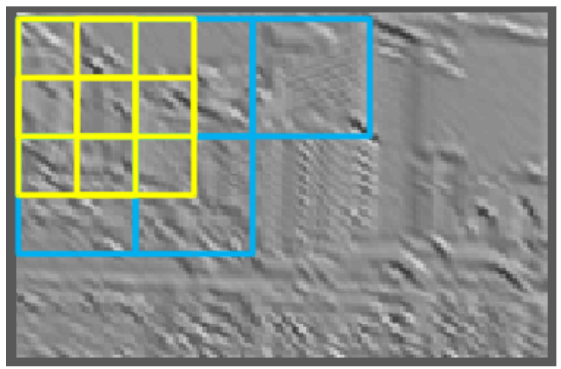


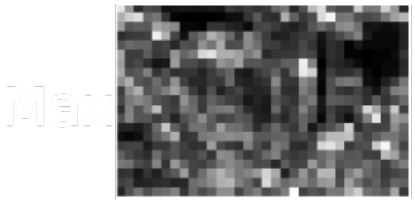
Différents opérateurs de pooling (1)

À partir d'une "tuile" de quelques neurones adjacents, on en crée un seul, à l'aide d'un opérateur, par exemple, le Max :

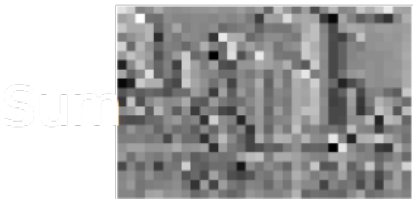


Les opérateurs Max (en haut) et Somme (en bas) ont des effets différents :





MAX



SUM

Différents opérateurs de pooling (2)

De nombreux autres opérateurs peuvent être mis en œuvre : par exemple un average pooling, un L2-norm pooling ou même un pooling stochastique.

Néanmoins, celui qui est le plus utilisé est le max pooling car il augmente plus significativement l'importance des activations fortes. On l'emploie généralement sur des tuiles de quatre neurones $\Rightarrow$ on parle alors de Max-Pool 2x2.

- **Pooling: aggregation over space or feature type**
  - Max, Lp norm, log prob.

$$MAX : Max_i(X_i); \quad L_p : \sqrt[p]{X_i^p}; \quad PROB : \frac{1}{b}\log\left(\sum_i e^{bX_i}\right)$$

[Extrait de "The Future of IA", LeCun, LORIA, Nancy, 2016]

Réseau CNN basique

Les deux ingrédients de base, la couche de convolution et la couche de pooling, peuvent être assemblés de la manière suivante pour constituer un Convolutional Neural Network (CNN) :

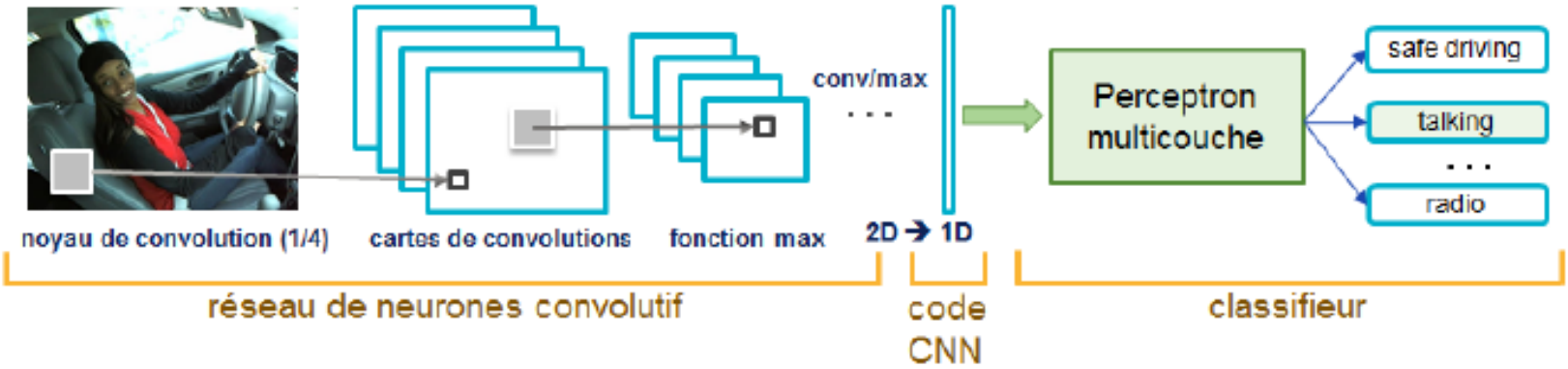


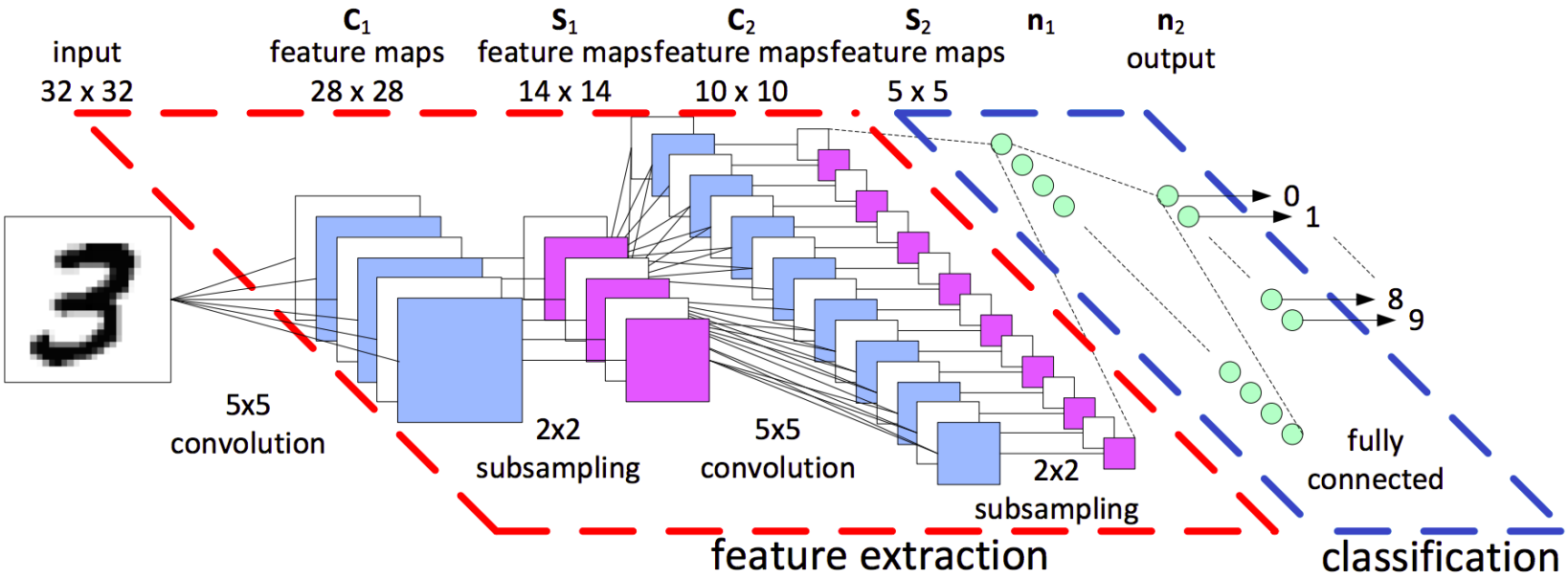
Figure II.5 Architecture standard d'un réseau de neurone convolutionnel

[d'après M.M. Zakaria, 2017]

Le réseau obtenu ("code CNN" sur la figure) est ensuite *mis à plat* en un seul vecteur qui peut servir d'entrée à n'importe quel classifieur, un MLP par exemple.

Architecture de CNN pour l'OCR

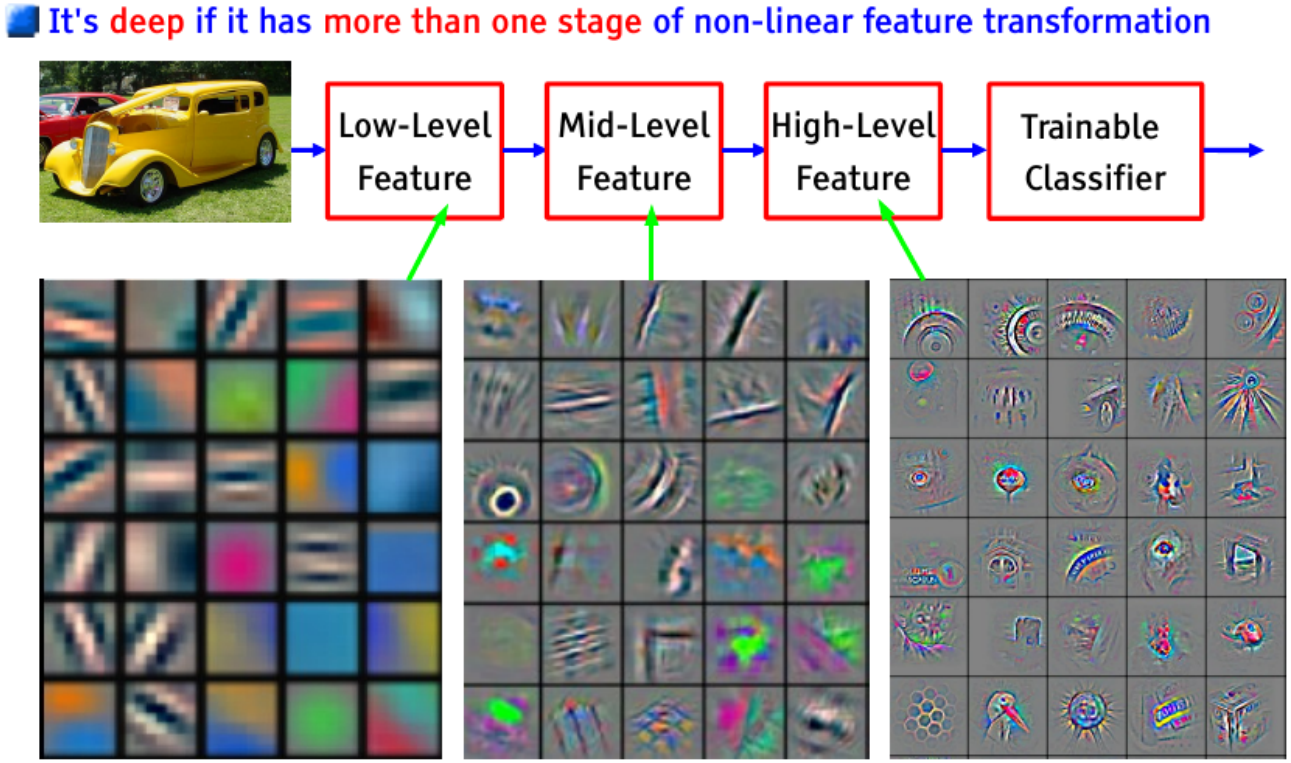
Pour reconnaître des chiffres manuscrits (10 classes en sortie), on peut opérer des convolutions successives.



Les couches appellées "subsampling" (sous-échantillonnage) sont des couches de pooling.

Deep CNN

Néanmoins, on peut empiler plusieurs couches convolutionnelles, en y intercalant (ou pas...) des couches de pooling, et varier à l'infini la composition d'un CNN. On peut alors parler de Deep CNN :



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[Extrait de "The Future of IA", LeCun, LORIA, Nancy, 2016]

Couche de correction : ReLU

Pour améliorer l'efficacité du traitement, on intercale souvent, entre les couches de traitement, une couche qui opère une fonction d'activation sur les signaux issus des filtres.

En effet, les filtres calculent des sommes pondérées $w^T x + b$ (cf. p. 6) mais les neurones classiques appliquent ensuite une fonction d'activation : $s = f(w^T x + b)$ où f est une fonction signe ou de Heaviside (neurone à seuil), ou sigmoïde, ou autre.

La fonction ReLU (abréviation de **Rectified Linear Units**) est définie par : $f(u) = max(0, u)$ $\Rightarrow$ elle force les neurones à retourner des valeurs positives. Elle est couramment utilisée comme fonction d'activation.

On parle alors de couche ReLU.

Couche entièrement connectée : FC

Après plusieurs couches de convolution, de pooling et / ou de ReLU, on construit une couche complètement connectée. C'est une couche "vectorielle" (i.e. sans épaisseur) qui :

1. est construite à partir de tous les étages de la couche précédente,
2. est complètement connectée à la suivante.

Cette couche peut donc être constituée de neurones classiques, sigmoïdes par exemple.

On parle alors de couche FC, pour **Full Connection**.

Couche de perte : LOSS

Comme la couche de sortie d'un réseau MLP, cette dernière couche estime l'erreur entre sortie calculée et sortie désirée. On y définit une fonction de coût ou fonction de perte, la *"LOSS function"*.

Cette fonction peut être l'**erreur quadratique moyenne**, calculée pour n exemples et k neurones de sortie :

$$LOSS_{\mathcal{L}^2} = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}(s_{ij} - sd_{ij})^2$$

où s_{ij} (resp. sd_{ij}) est la $j^{\text{ème}}$ sortie calculée (resp. désirée) pour un exemple X_i.

Exemple : $sd_{ih} = 1$ si l'image X_i est de classe C_h et $sd_{ij} = 0$ pour $j \neq h$

La fonction de coût permet d'estimer une performance et peut aussi servir à modifier les poids des couches FC qui précèdent (ex. rétro-propagation).

LOSS avec fonction *softmax*

Le plus souvent, la fonction de coût est calculée à partir de la fonction **softmax** qui transforme la valeur de chaque neurone de sortie en une probabilité d'appartenance à chacune des k classes :

$$\forall j \in \{1 \dots k\} \quad s_{ij}^{new} = P(class(X_i) = j | X_i) = \frac{e^{s_{ij}}}{\sum_{c=1}^{k} e^{s_{ic}}}$$

où s_{ij} est la $j^{\text{ème}}$ sortie initialement calculée par le réseau pour l'exemple X_i.
ce qui donne l'expression suivante pour la fonction de coût :

$$LOSS_{soft} = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbb{1}_{class(X_i)=j} \log \left(\frac{e^{s_{ij}}}{\sum_{c=1}^{k} e^{s_{ic}}} \right)$$

La couche de sortie est parfois appelée couche LOSS.

Synthèse

On peut donc "empiler" tous les ingrédients permettant de construire des Convolutional Neural Networks, en un ou plusieurs exemplaires chacun, mais . . . pas tout à fait dans n'importe quel ordre !

Exemples d'architectures de CNN ayant un sens pour traiter des données :
(. . . avec une couche LOSS, à chaque fois, à la fin)

- INPUT → CONV → ReLU → FC
- INPUT → [CONV → ReLU → POOL] * 2 → FC → ReLU → FC
  Ici, il y a une seule couche CONV entre chaque couche POOL.
- INPUT → [CONV → ReLU → CONV → ReLU → POOL] * 3 → [FC → ReLU] * 2 → FC
  Là, deux couches CONV sont empilées avant chaque couche POOL.

Le problème d'*overfitting*

Dans un réseau CNN, les *features*, et donc les paramètres du réseau, sont extrêmement nombreux $\Rightarrow$ il y a un fort risque d'*overfitting*, i.e. de **surapprentissage**, ce qui nuira aux qualités prédictives du modèle.

Les techniques de régularisation permettent de pallier cette difficulté.

On peut utiliser les techniques classiques telles que la régularisation $\mathcal{L}^1$ ou la régularisation $\mathcal{L}^2$, au niveau de la fonction de coût.

Plus souvent, dans les CNN, on utilise la méthode du dropout proposée par Hinton en 2012.

Régularisation $\mathcal{L}^2$: ridge regression

La régularisation $\mathcal{L}^2$ appliquée aux poids du réseau était déjà connue sous le nom de weight decay.

Il s'agit d'ajouter à la fonction de coût un terme de pénalisation afin d'éviter aux poids de prendre de trop grandes valeurs.
On cherchera donc à minimiser :

$$LOSS + \lambda \sum_{j=1}^{m} w_j^2$$

où tous les poids du réseau sont réunis dans un vecteur $W = (w_j)_{1 \leqslant j \leqslant m}$ avec m le nombre total de poids, quelles que soient les couches.

λ est un paramètre qui permet de doser l'importance relative que l'on accorde à la fonction de perte vs le contrôle des valeurs des poids.

Régularisation $\mathcal{L}^1$: LASSO

LASSO = "Least Absolute Shrinkage and Selection Operator".

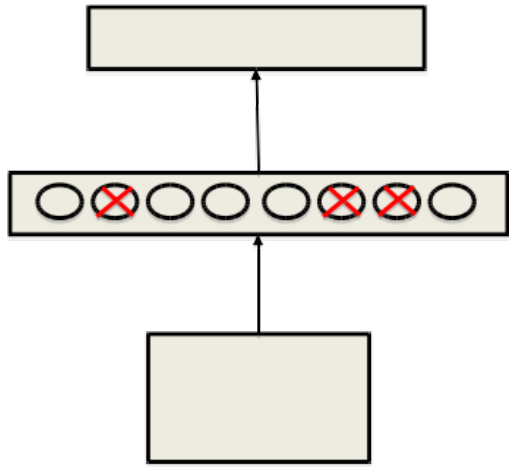
Le principe est le même mais le terme de pénalisation portera sur les valeurs absolues des poids.
On cherchera donc à minimiser :

$$LOSS + \lambda \sum_{j=1}^{m} |w_j|$$

La régularisation $\mathcal{L}^1$ tend à rendre le réseau épars, *i.e.* à imposer à un maximum de poids de s'annuler.
L'ajustement de λ est crucial : une valeur trop grande mettra trop de poids à zéro et conduira à de l'*underfitting*.

Dropout ("abandon", en Français)

Définition de la technique de dropout : en phase d'apprentissage, à chaque présentation d'un exemple, chaque neurone caché est éteint avec une probabilité de $0,5$.



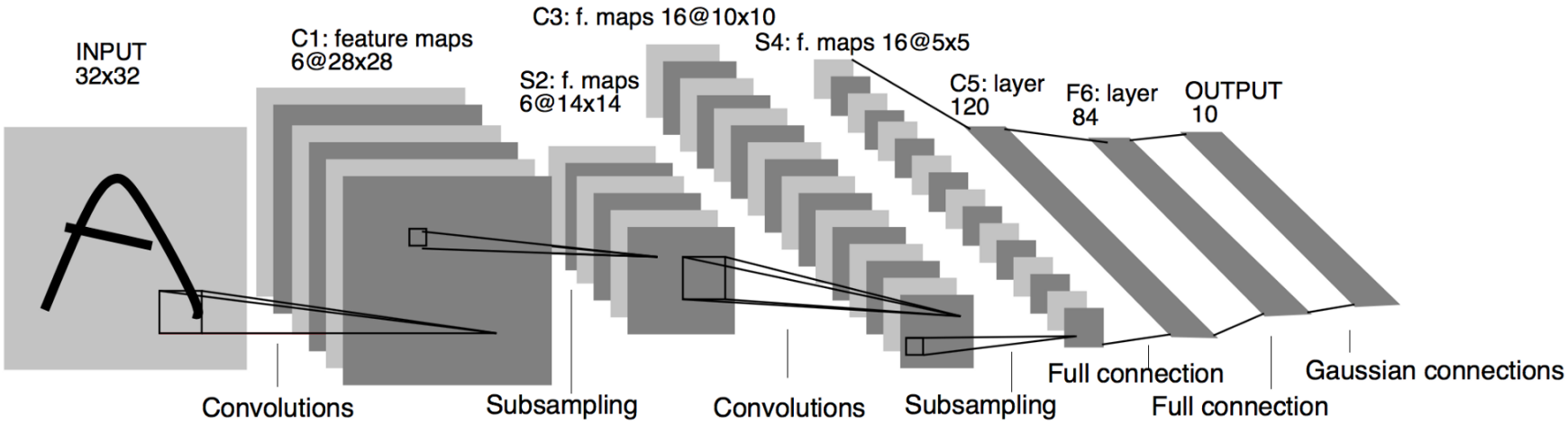
[d'après Hinton, 2012]

Ce sont donc des réseaux différents qui apprennent les différents exemples ! Mais ils ont tout de même des liens entre eux . . . (cf. stratégie de *bagging*).

En phase de généralisation, tous les neurones sont allumés mais on divise par deux tous les poids de connexion (opérant ainsi une forme de moyennage).

LeNet 5 *[http ://yann.lecun.com/exdb/lenet/ – oct. 2018 –]*

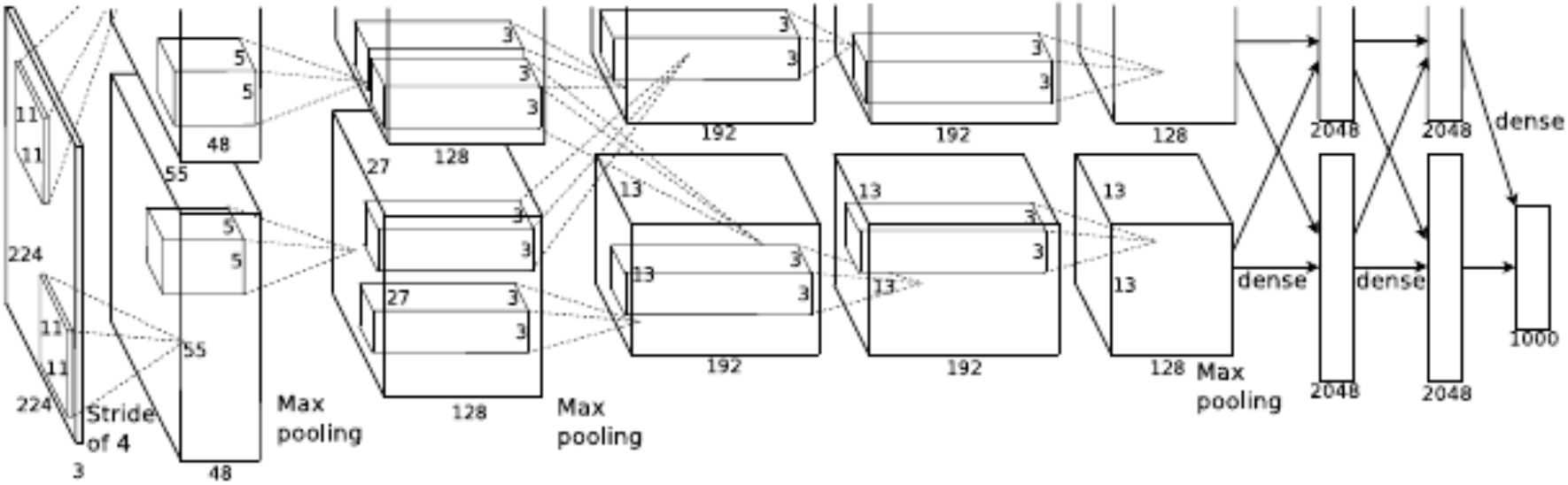
Un des premiers modèles de réseaux convolutifs est LeNet5, réseau proposé par Yann Le Cun, Bottou, Bengio, Haffner, en 1998, et qui est une amélioration du premier réseau "LeNet" que LeCun avait déjà construit en . . . 1990 !



Ce type de réseau (cf. l'URL) permet de reconnaître des caractères manuscrits (OCR) à la volée.

AlexNet

Architecture du réseau CNN vainqueur du challenge **ImageNet 2012** :

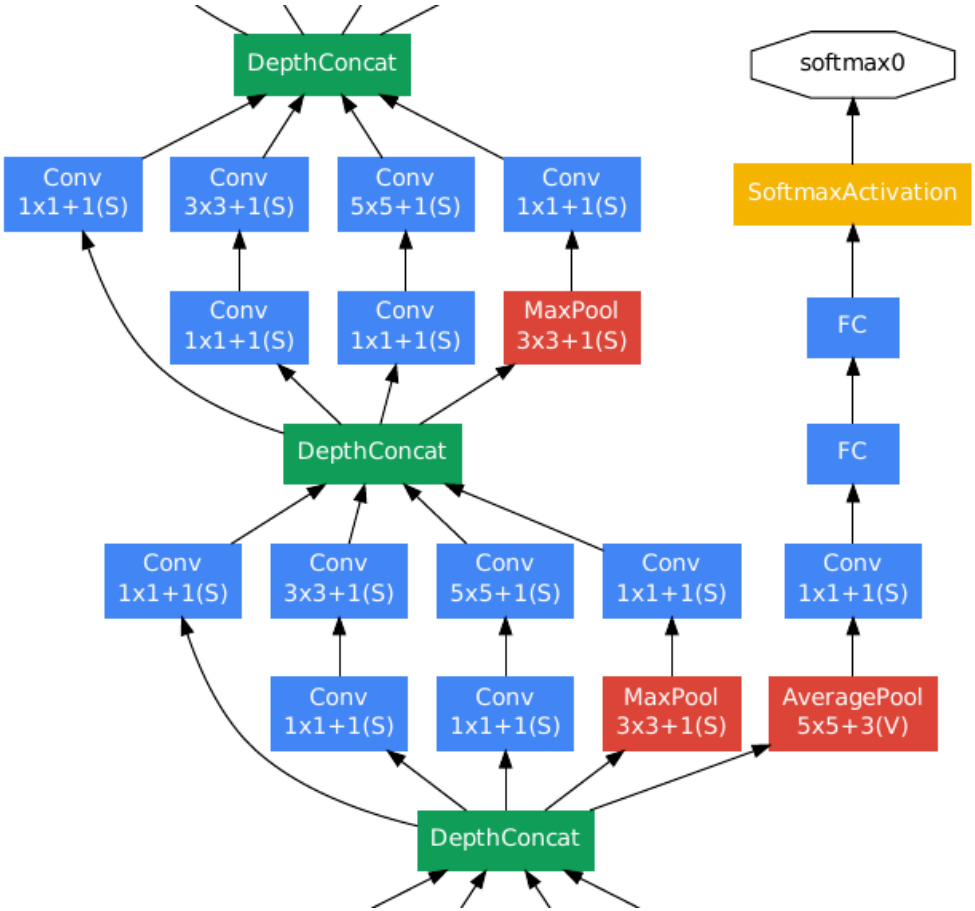


"ImageNet Classification with Deep Convolutional Neural Networks",
Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton – NIPS'2012

(cet article a été cité 31 432 fois..... à ce jour !)

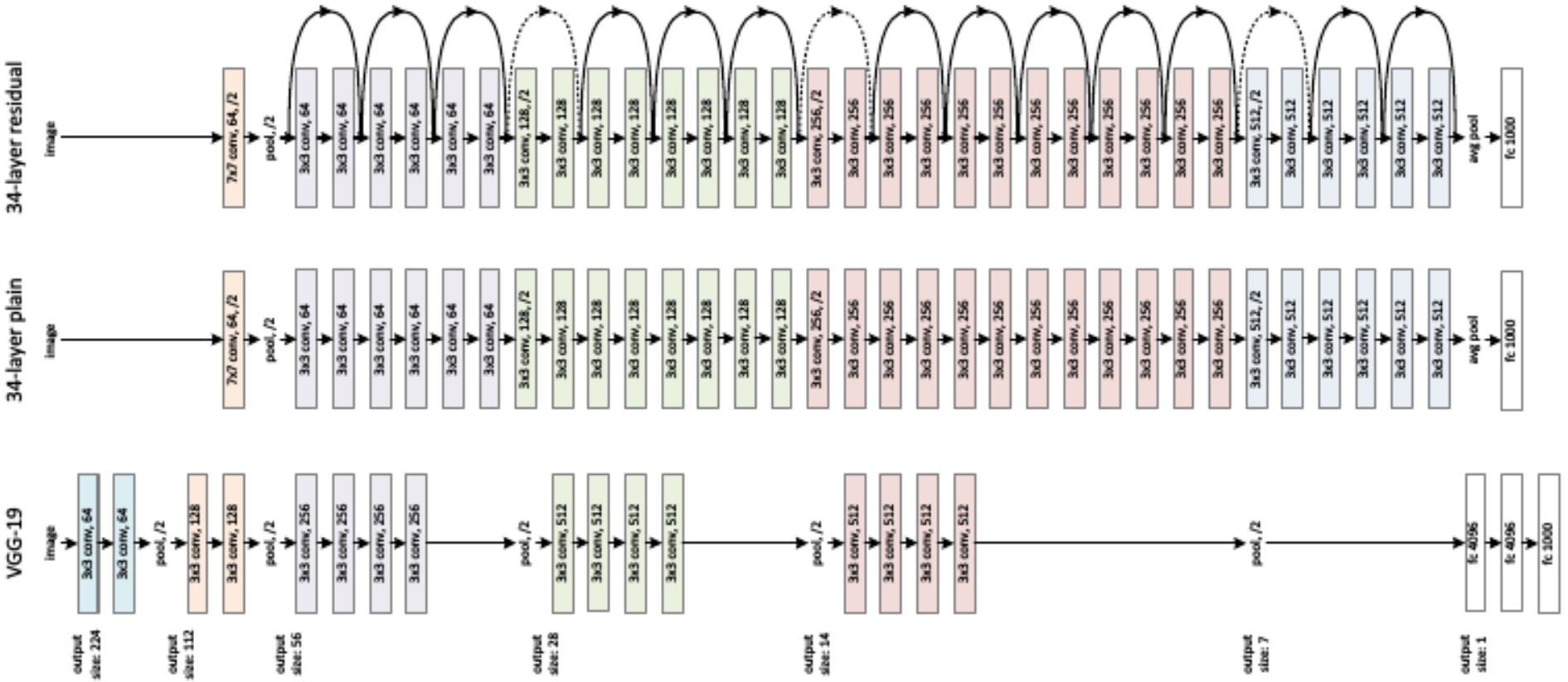
GoogLeNet / Inception V1 : juste un détail...

Seulement un module détaillé de ce réseau, vainqueur de **ILSVRC 2014** :



ResNet = VGG + Residual Nets

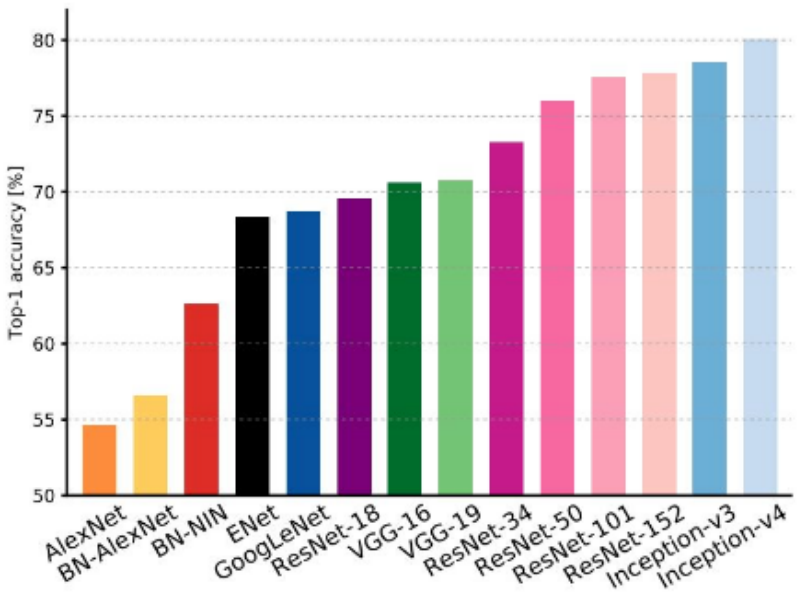
Vainqueur de **ILSVRC 2015** : Image Classification, Localization, Detection

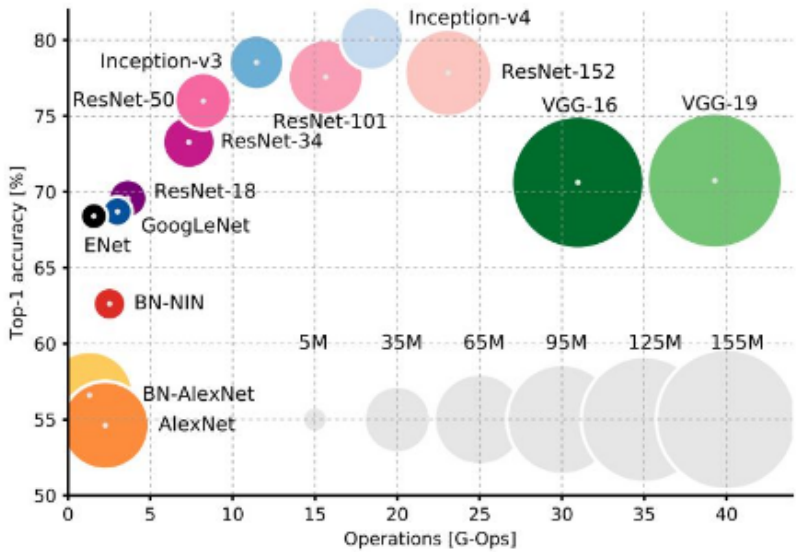


Ce réseau est composé de 152 couches !

Que penser de ces architectures ?

En novembre 2017, Siddharth Das a établi quelques statistiques :





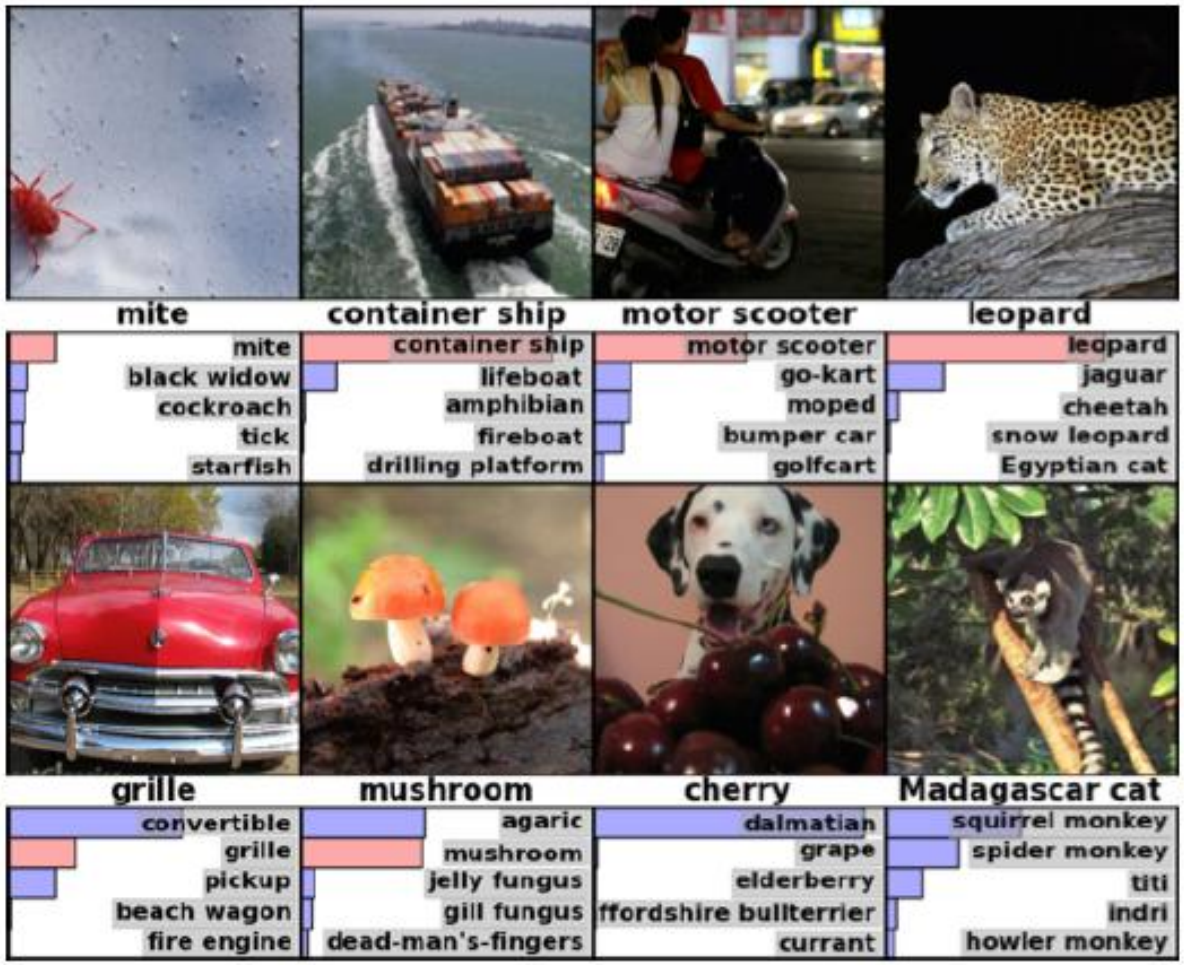
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Pour conclure sur les architectures :

c'est l'inflation en ce qui concerne tant le **nombre** que la **taille** des couches.

Les apprentissages nécessitent une énergie et des temps de calcul gigantesques !

Classification

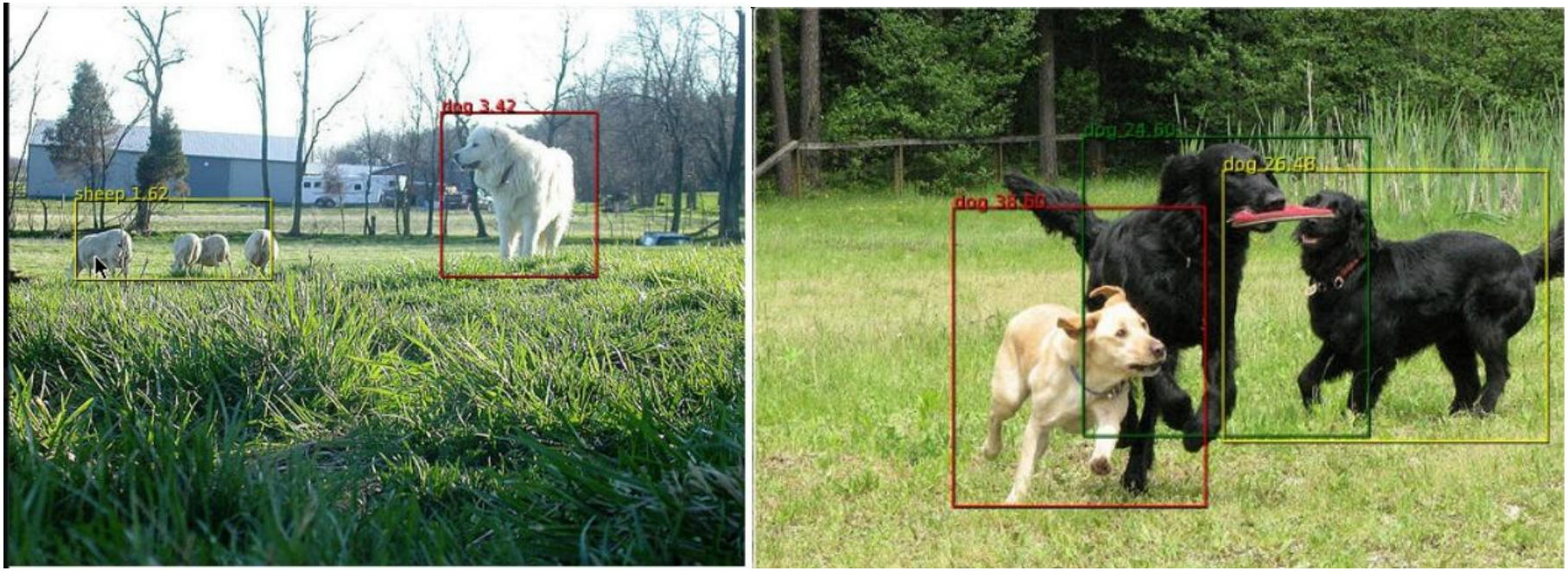


Les 5 classes les plus probables et, en rose, la probabilité attribuée par le réseau AlexNet à la classe désirée (même si objet décentré ou ambigu).

Reconnaissance

Images de test dans la 1ère colonne ; les 5 autres colonnes montrent les 6 images ayant servi à l'apprentissage des *features* par AlexNet.

Détection d'objets



A gauche : on distingue un chien d'un mouton (meme s'il leur ressemble).
A droite : on retrouve les trois chiens, même si les images présentent des recouvrements et des occultations.

[d'après LeCun, 2016]

Annotation de scènes

[from Farabet et al., PAMI 2013]

Conclusion

En résumé

- Les réseaux convolutifs sont une famille de réseaux de neurones à architecture modulable, à partir de différents types de couches.
- Les architectures efficaces sont profondes, avec des extractions de *features* de plus en plus abstraites : *deep learning*.
- Les réseaux convolutifs sont très puissants, notamment pour le traitement d'images, mais aussi langage, parole, diagnostic médical...
- mais... Les réseaux convolutifs sont des "ogres" : énormes et voraces !

Perspectives

On commence à voir apparaître, dans la littérature, des réseaux convolutifs profonds à base de spiking neurons, ce qui les rend :
* beaucoup moins consommateurs d'énergie et de temps de calcul ;
* propices à une implémentation efficace sur des nanocomposants.

Bibliographie

LeNet Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. *Backpropagation applied to handwritten zip code recognition.* Neural Computation, 1(4) :541–551, 1989.

LeNet 5 Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 86(11) :2278–2324, 1998.

AlexNet A. Krizhevsky, I. Sutskever, G. E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks.* In Proc. of NIPS, 25 :1097–1105, 2012.

Image C. Farabet, C. Couprie, L. Najman, Y. LeCun. *Learning Hierarchical Features for Scene Labeling*, IEEE Trans PAMI, 35(8) :1915-29, 2013.

Dropout N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. *Dropout : A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15 :1929-1958, 2014.

Deep I. J. Goodfellow, Y. Bengio, A. Courville. **Deep Learning,** MIT Press, 2016.

Général A. Cornuéjols, L. Miclet, V. Barra. **Apprentissage artificiel - Deep learning, concepts et algorithmes,** 3ème édition, Eyrolles, 2018.

Webographie

- démo "live" de LeNet5 pour la reconnaissance de caractères
  http ://yann.lecun.com/exdb/lenet/ (consulté le 27 nov. 2018)
- différentes architectures de ConvNets
  Convolutional Neural Network Architectures : from LeNet to ResNet
  slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf (27 nov. 2018)
- un cours très complet
  Introduction to Deep Learning – Deep Convolutional Neural Networks
  https ://web.cs.hacettepe.edu.tr/.../l13-deep_learning-convnets.pdf
- survey des méthodes et des applications, par Y. LeCun, 2016
  Deep Learning and The Future of IA – Exposé au LORIA (Nancy)
  deeploria.gforge.inria.fr/lecun-20160412-nancy-loria.pdf (27 nov. 2018)

"Photo de famille"

Les principaux acteurs du courant Deep Learning – ConvNets :

de gauche à droite

Yann LeCun – Geoffrey Hinton – Yoshua Bengio – Andrew Ng