

# RÉSEAUX CONVOLUTIFS

## SOMMAIRE

### 1- Introduction

2

- 1.1- Inspiration biologique
- 1.2- Convolution discrète

### 2- Définition des couches

3

- 2.1- Couche de convolution
- 2.2- Couche non linéaire
- 2.3- Couches de normalisation
- 2.4- Couche d'agrégation et de sous-échantillonnage
- 2.5- Couche complètement connectée

### 3- Régularisation

9

- 3.1- Régularisation de la fonction de coût
- 3.2- Dropout
- 3.3- Partage de paramètres

### 4- Initialisation

10

- 4.1- Initialisation prenant en compte les variations des neurones
- 4.2- Initialisation de He

### 5- Apprentissage

11

- 5.1- Le problème de l'entraînement
- 5.2- Apprentissage glouton par couche
- 5.3- Visualisation du mécanisme des réseaux convolutifs
- 5.4- Méthodes de visualisation de base
- 5.5- Méthodes fondées sur les activations
- 5.6- Méthodes fondées sur le gradient

# 1- INTRODUCTION

## 1.1- Inspiration biologique

Un réseau de neurones convolutif (CNN, *Convolutional Neural Network* ou ConvNet) est un type de réseau de neurones artificiels acyclique à propagation avant, dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Les neurones de cette région du cerveau sont arrangés de sorte à ce qu'ils correspondent à des régions (appelés champs réceptifs) qui se chevauchent lors du pavage du champ visuel. Ils sont de plus organisés de manière hiérarchique, en couches (aire visuelle primaire V1, secondaire V2, puis aires V3, V4, V5 et V6, gyrus temporal inférieur), chacune des couches étant spécialisée dans une tâche, de plus en plus abstraite en allant de l'entrée vers la sortie. En simplifiant à l'extrême, une fois que les signaux lumineux sont reçus par la rétine et convertis en potentiels d'action :

- L'aire primaire V1 s'intéresse principalement à la détection de contours, ces contours étant définis comme des zones de fort contraste de signaux visuels reçus.
- L'aire V2 reçoit les informations de V1 et extrait des informations telles que la fréquence spatiale, l'orientation, ou encore la couleur.
- L'aire V4, qui reçoit des informations de V2, mais aussi de V1 directement, détecte des caractéristiques plus complexes et abstraites liées par exemple à la forme.
- Le gyrus temporal inférieur est chargé de la partie sémantique (reconnaissance des objets), à partir des informations reçues des aires précédentes et d'une mémoire des informations stockées sur des objets.

L'architecture et le fonctionnement des réseaux convolutifs sont inspirés par ces processus biologiques. Ces réseaux consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations. Les réseaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel (voir section 6- pour quelques exemples)

Un réseau convolutif se compose de deux types de neurones, agencés en couches traitant successivement l'information. Dans le cas du traitement de données de type images, on a ainsi :

- des *neurones de traitement*, qui traitent une portion limitée de l'image (le champ réceptif) au travers d'une fonction de convolution ;
- des *neurones* de mise en commun des sorties dits *d'agrégation totale ou partielle (pooling)*.

Un traitement correctif non linéaire est appliqué entre chaque couche pour améliorer la pertinence du résultat. L'ensemble des sorties d'une couche de traitement permet de reconstituer une image intermédiaire, dite carte de caractéristiques (feature map), qui sert de base à la couche suivante. Les couches et leurs connexions apprennent des niveaux d'abstraction croissants et extraient des caractéristiques de plus en plus haut niveau des données d'entrée.

Dans la suite, le propos sera illustré sur des images 2D en niveaux de gris, de taille  $n_1 \times n_2$  :

$$\begin{aligned} \mathbf{I}: [[1 \cdots n_1]] \times [[1 \cdots n_2]] &\rightarrow \mathbb{R} \\ (i, j) &\mapsto I_{i,j} \end{aligned}$$

**I** sera indifféremment vue comme une fonction ou une matrice.

## 1.2- Convolution discrète

Pour reproduire la notion de champ réceptif, et ainsi permettre aux neurones de détecter des caractéristiques de petite taille mais porteurs d'information, l'idée est de laisser un neurone caché voir et traiter seulement une petite portion de l'image qu'il prend en entrée. L'outil retenu dans les réseaux convolutifs est la convolution discrète.

**Définition 1-1 (Convolution discrète)** Soient  $h_1, h_2 \in \mathbb{N}$ ,  $\mathbf{K} \in \mathbb{R}^{(2h_1+1) \times (2h_2+1)}$ . La convolution discrète de  $\mathbf{I}$  par le filtre  $\mathbf{K}$  est donnée par :

$$(\mathbf{K} * \mathbf{I})_{r,s} = \sum_{u=-h_1}^{h_1} \sum_{v=-h_2}^{h_2} K_{u,v} I_{r+u,s+v} \quad (1)$$

où  $\mathbf{K}$  est donné par :

$$\mathbf{K} = \begin{pmatrix} K_{-h_1,-h_2} & \cdots & K_{-h_1,h_2} \\ \vdots & K_{0,0} & \vdots \\ K_{h_1,-h_2} & \cdots & K_{h_1,h_2} \end{pmatrix}. \quad (2)$$

La taille du filtre  $(2h_1 + 1) \times (2h_2 + 1)$  précise le champ visuel capturé et traité par  $\mathbf{K}$ .

Lorsque  $\mathbf{K}$  parcourt  $\mathbf{I}$ , le déplacement du filtre est réglé par deux paramètres de *stride* (horizontal et vertical). Un stride de 1 horizontal (respectivement vertical) signifie que  $\mathbf{K}$  se déplace d'une position horizontale (resp. verticale) à chaque application de .1. Les valeurs de stride peuvent également être supérieures et ainsi sous-échantillonner  $\mathbf{I}$ .

Le comportement du filtre sur les bords de  $\mathbf{I}$  doit également être précisé, par l'intermédiaire d'un paramètre de *padding*. Si l'image convoluée  $(\mathbf{K} * \mathbf{I})$  doit posséder la même taille que  $\mathbf{I}$ , alors  $2h_1$  lignes de 0 ( $h_1$  en haut et  $h_1$  en bas) et  $2h_2$  colonnes de 0 ( $h_2$  à gauche et  $h_2$  à droite) doivent être ajoutées. Dans le cas où la convolution est réalisée sans padding, l'image convoluée est de taille  $(n_1 - 2h_1) \times (n_2 - 2h_2)$ .

## 2- DÉFINITION DES COUCHES

Nous introduisons ici les différents types de couches utilisées dans les réseaux convolutifs. L'assemblage de ces couches permet de construire des architectures complexes pour la classification ou la régression, dont certaines seront précisées dans un prochain cours.

### 2.1- Couche de convolution

Soit  $l \in \mathbb{N}$  une couche de convolution. L'entrée de la couche  $l$  est composée de  $n^{(l-1)}$  cartes provenant de la couche précédente, de taille  $n_1^{(l-1)} \times n_2^{(l-1)}$ . Dans le cas de la couche d'entrée du réseau ( $l = 1$ ), l'entrée est l'image  $\mathbf{I}$ . La sortie de la couche  $l$  est formée de  $n^{(l)}$  cartes de taille  $n_1^{(l)} \times n_2^{(l)}$ . La  $i^e$  carte de la couche  $l$ , notée  $\mathbf{Y}_i^{(l)}$ , se calcule comme :

$$\mathbf{Y}_i^{(l)} = \mathbf{B}_i^{(l)} + \sum_{j=1}^{n^{(l-1)}} \mathbf{K}_{i,j}^{(l)} * \mathbf{Y}_j^{(l-1)} \quad (3)$$

où  $\mathbf{B}_i^{(l)}$  est une matrice de biais et  $\mathbf{K}_{i,j}^{(l)}$  est le filtre de taille  $(2h_1^{(l)} + 1) \times (2h_2^{(l)} + 1)$  connectant la  $j^e$  carte de la couche  $(l - 1)$  à la  $i^e$  carte de la couche  $l$  (voir la figure 2-2).

$n_1^{(l)}$  et  $n_2^{(l)}$  doivent prendre en compte les effets de bords : lors du calcul de la convolution, seuls les pixels dont la somme est définie avec des indices positifs doivent être traités. Dans le cas où le padding n'est pas utilisé, les cartes de sortie ont donc une taille de  $n_1^{(l)} = n_1^{(l-1)} - 2h_1^{(l)}$  et  $n_2^{(l)} = n_2^{(l-1)} - 2h_2^{(l)}$ .

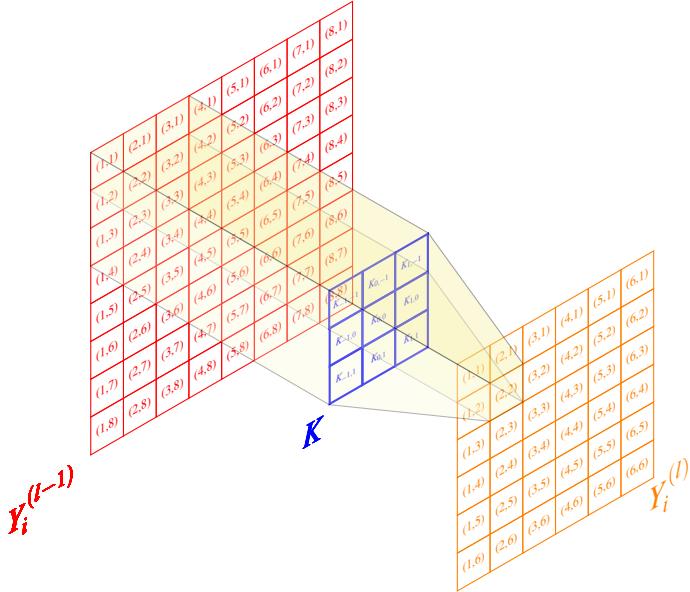


FIGURE 2-1 – Illustration des calculs effectués dans une opération de convolution discrète. Le pixel (2,2) de l'image  $\mathbf{Y}_i^{(l)}$  est une combinaison linéaire des pixels  $(i, j), i, j \in \llbracket 1, 3 \rrbracket$  de  $\mathbf{Y}_i^{(l-1)}$ , les coefficients de la combinaison étant portés par le filtre  $\mathbf{K}$  (équation .3).

Souvent, les filtres utilisés pour calculer  $\mathbf{Y}_i^{(l)}$  sont les mêmes, i.e.  $\mathbf{K}_{i,j}^{(l)} = \mathbf{K}_{i,k}^{(l)}$  pour  $j \neq k$ . De plus, la somme dans l'équation (.3) peut être conduite sur un sous ensemble des cartes d'entrée.

Il est possible de mettre en correspondance la couche de convolution et l'opération (.3) qu'elle effectue, avec un perceptron multicouche. Pour cela, il suffit de réécrire l'équation (.3) : chaque carte  $\mathbf{Y}_i^{(l)}$  de la couche  $l$  est formée de  $n_1^{(l)} \cdot n_2^{(l)}$  neurones organisés dans un tableau à deux dimensions. Le neurone en position  $(r, s)$  calcule :

$$(\mathbf{Y}_i^{(l)})_{r,s} = (\mathbf{B}_i^{(l)})_{r,s} + \sum_{j=1}^{n^{(l-1)}} (\mathbf{K}_{i,j}^{(l)} * \mathbf{Y}_j^{(l-1)})_{r,s} \quad (4)$$

$$= (\mathbf{B}_i^{(l)})_{r,s} + \sum_{j=1}^{n^{(l-1)}} \sum_{u=-h_1^{(l)}}^{h_1^{(l)}} \sum_{v=-h_2^{(l)}}^{h_2^{(l)}} (\mathbf{K}_{i,j}^{(l)})_{u,v} (\mathbf{Y}_j^{(l-1)})_{r+u, s+v} \quad (5)$$

Les paramètres du réseau à entraîner (poids) peuvent alors être trouvés dans les filtres  $\mathbf{K}_{i,j}^{(l)}$  et les matrices de biais  $\mathbf{B}_i^{(l)}$ .

Comme nous le verrons dans la section 2.4-, un sous-échantillonnage est utilisé pour diminuer l'influence du bruit et des distorsions dans les images. Le sous-échantillonnage peut être également réalisé simplement avec des paramètres de stride, en sautant un nombre fixe de pixels dans les dimensions horizontale (saut  $s_1^{(l)}$ ) et verticale (saut  $s_2^{(l)}$ ) avant d'appliquer de nouveau le filtre. La taille des images de sortie est alors :

$$n_1^{(l)} = \frac{n_1^{(l-1)} - 2h_1^{(l)}}{s_1^{(l)} + 1} \quad \text{et} \quad n_2^{(l)} = \frac{n_2^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)} + 1}. \quad (6)$$

Un point clé des réseaux convolutifs est d'exploiter la corrélation spatiale des données. L'utilisation des noyaux permet d'alléger le modèle, plutôt que d'utiliser des couches complètement connectées.

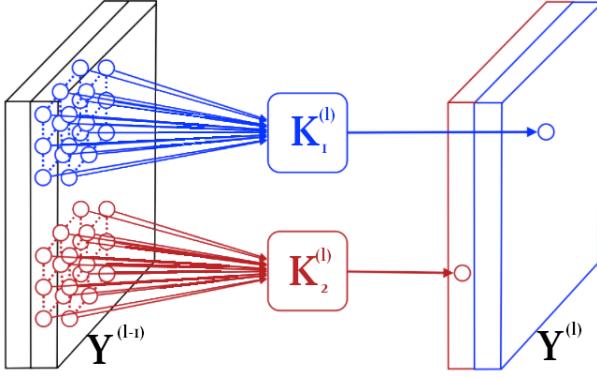


FIGURE 2-2 – Illustration d'une couche de convolution  $l$ . L'image d'entrée ( $l = 1$ ) ou une carte de caractéristiques de la couche ( $l - 1$ ) est convoluée par différents filtres pour donner les cartes de sortie de la couche  $l$ .

## 2.2- Couche non linéaire

Pour augmenter le pouvoir d'expression des réseaux profonds, on utilise des couches non linéaires. Les entrées d'une couche non linéaire sont  $n^{(l-1)}$  cartes et ses sorties  $n^{(l)} = n^{(l-1)}$  cartes  $\mathbf{Y}_i^{(l)}$ , de taille  $n_1^{(l-1)} \times n_2^{(l-1)}$  telles que  $n_1^{(l)} = n_1^{(l-1)}$  et  $n_2^{(l)} = n_2^{(l-1)}$ , données par  $\mathbf{Y}_i^{(l)} = f(\mathbf{Y}_i^{(l-1)})$ , où  $f$  est la fonction d'activation utilisée dans la couche  $l$ . Le tableau .1 propose quelques fonctions d'activation usuelles.

En apprentissage profond, il a été reporté que la sigmoïde et la tangente hyperbolique avaient des performances moindres que la fonction d'activation *softsign* :

$$\mathbf{Y}_i^{(l)} = \frac{1}{1 + |\mathbf{Y}_i^{(l-1)}|}. \quad (7)$$

En effet, les valeurs des pixels des cartes  $\mathbf{Y}_i^{(l-1)}$  arrivant près des paliers de saturation de ces fonctions donnent des gradients faibles, qui ont tendance à s'annuler (problème du *gradient évanescant* ou *vanishing gradient*) lors de la phase d'apprentissage par rétropropagation du gradient. Une autre fonction, non saturante elle, est très largement utilisée. Il s'agit de la fonction ReLU (Rectified Linear Unit) [9] :

$$\mathbf{Y}_i^{(l)} = \max(0, \mathbf{Y}_i^{(l-1)}). \quad (8)$$

Les neurones utilisant la fonction décrite dans l'équation (8) sont appelés neurones linéaires rectifiés. Glorot et Bengio [5] ont montré que l'utilisation d'une couche ReLU en tant que couche non linéaire permettait un entraînement efficace de réseaux profonds sans pré-entraînement non supervisé. Plusieurs variantes de cette fonction existent, par exemple pour assurer une différentiabilité en 0 ou pour proposer des valeurs non nulles pour des valeurs négatives de l'argument. La figure 2-3. illustre quelques unes de ces fonctions d'activation.

## 2.3- Couches de normalisation

La normalisation prend aujourd'hui une place de plus en plus importante, notamment depuis les travaux de Ioffe et Szegedy [8]. Les auteurs suggèrent qu'un changement dans la distribution des activations d'un réseau profond, résultant de la présentation d'un nouveau mini batch d'exemples, ralentit le processus d'apprentissage. Pour pallier ce problème, chaque activation du mini batch est centrée et normée (variance unité), la moyenne et la variance étant calculées sur le mini batch entier, indépendamment pour chaque activation. Des paramètres d'offset  $\beta$  et multiplicatif  $\gamma$  sont alors appliqués pour normaliser les données d'entrée (algorithme 1).

Lorsque la descente de gradient est achevée, un post apprentissage est appliqué dans lequel la moyenne et la variance sont calculées sur l'ensemble d'entraînement et remplacent  $\mu_B$  et  $\sigma_B^2$  (algorithme 2).

Nom	Graphe	$f$	$f'$
Rampe		$f(x) = x$	$f'(x) = 1$
Heaviside		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{si } x \neq 0 \\ ? & \text{si } x = 0 \end{cases}$
Logistique ou sigmoïde		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tangente hyperbolique		$f(x) = \tanh(x) \\ = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f^2(x)$
Arc Tangente		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
ReLU		$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{si } x \neq 0 \\ 1 & \text{si } x = 0 \end{cases}$
Exponentielle Linéaire		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$

TABLE .1 – Quelques fonctions d’activation

## 2.4- Couche d’agrégation et de sous-échantillonnage

Le sous-échantillonnage (pooling) des cartes obtenues par les couches précédentes a pour objectif d’assurer une robustesse au bruit et aux distorsions.

La sortie d’une couche d’agrégation  $l$  (figure 2-4) est composée de  $n^{(l)} = n^{(l-1)}$  cartes de taille réduite. En général, l’agrégation est effectuée en déplaçant dans les cartes d’entrée une fenêtre de taille  $2p \times 2p$  toutes les  $q$  positions (il y a recouvrement si  $q < p$  et non recouvrement sinon), et en calculant, pour chaque position de la fenêtre, une seule valeur, affectée à la position centrale dans la carte de sortie. On distingue généralement deux types d’agrégation :

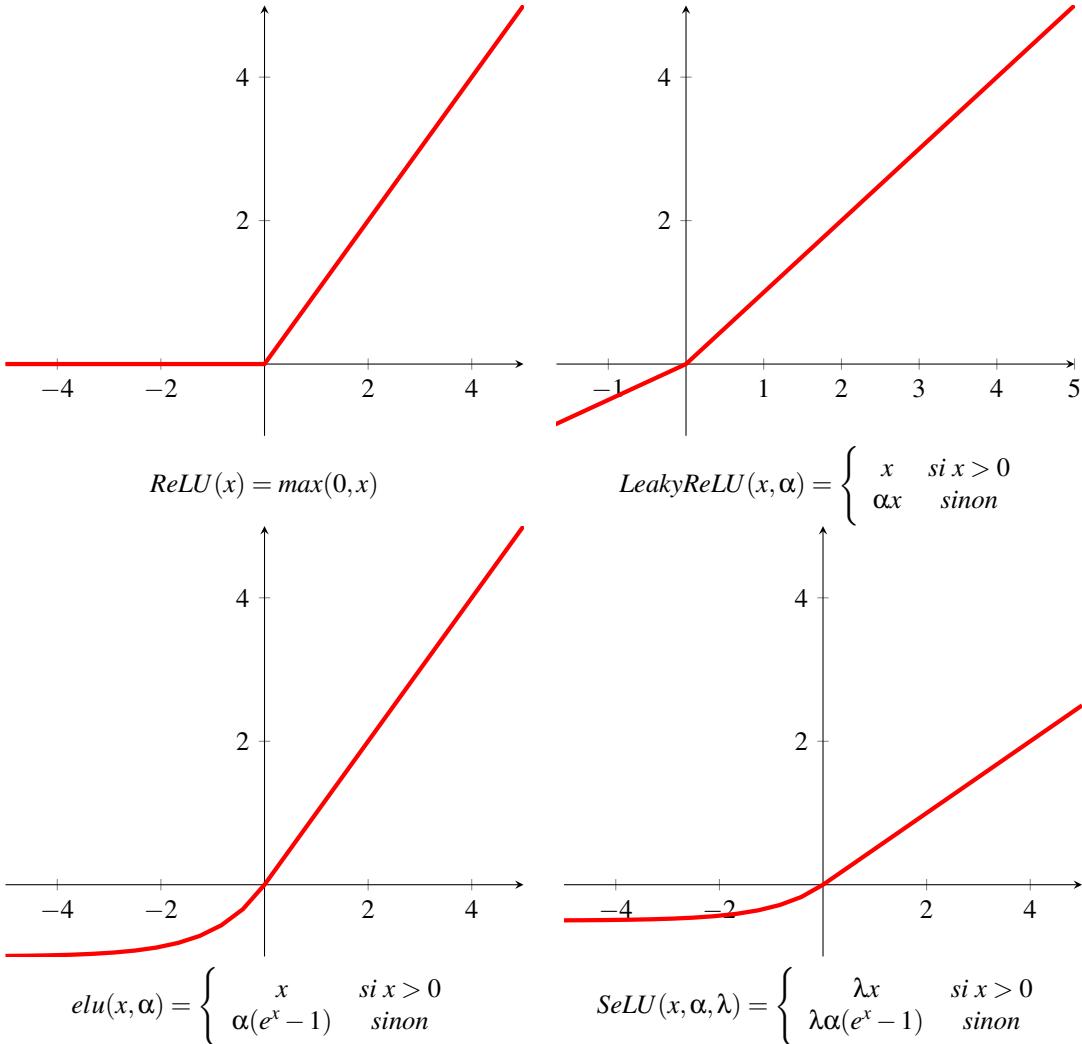


FIGURE 2-3 – Quelques fonctions d’activation

---

**Algorithme 1 :** Normalisation par batch sur la présentation d’un mini batch  $\mathcal{B}$

---

**Données :** valeurs de l’activation  $x$  sur un mini batch  $\mathcal{B} = \{x_1 \dots x_m\}$

Paramètres  $\beta, \gamma$  à apprendre

**Résultat :** Données normalisées  $\{y_1 \dots y_m\} = BN_{\gamma, \beta}(x_1 \dots x_m)$

**début**

$$\left| \begin{array}{l} \mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i \\ \sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \\ \text{pour } i=1 \text{ à } m \text{ faire} \\ \quad \left| \begin{array}{l} y_i = \gamma \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \beta \end{array} \right. \end{array} \right.$$


---

**La moyenne** : on utilise un filtre  $\mathbf{K}_{\mathbf{B}}$  de taille  $(2h_1 + 1) \times (2h_2 + 1)$  défini par :

$$(\mathbf{K}_{\mathbf{B}})_{r,s} = \frac{1}{(2h_1 + 1)(2h_2 + 1)}$$

---

**Algorithme 2 : Normalisation par batch d'un réseau**


---

**Données :** un réseau  $N$   
 un ensemble d'activations  $\{x^1 \dots x^K\}$

**début**

$N_n = N$

**pour  $i=1$  à  $K$  faire**

- Calculer  $y^i = BN_{\gamma, \beta}(x^i)$  à l'aide de l'algorithme 1
- Modifier chaque couche de  $N_n$  : l'entrée  $y^i$  remplace l'entrée  $x^i$

Entraîner  $N_n$  pour optimiser les paramètres de  $N$  et  $(\gamma^i, \beta^i)_{1 \leq i \leq K}$

$N^f = N_n$

**pour  $i=1$  à  $K$  faire**

- Utiliser  $N^f$  sur des batchs  $\mathcal{B}$  de taille  $m$
- Calculer la moyenne des moyennes  $\bar{x}^i$  et des variances  $Var(x^i)$
- Remplacer dans  $N^f$  la transformation  $y^i = BN_{\gamma, \beta}(x^i)$  par

$$y^i = \frac{\gamma^i}{\sqrt{Var(x^i) + \epsilon}} x^i + \left( \beta^i - \frac{\gamma^i \bar{x}^i}{\sqrt{Var(x^i) + \epsilon}} \right)$$


---

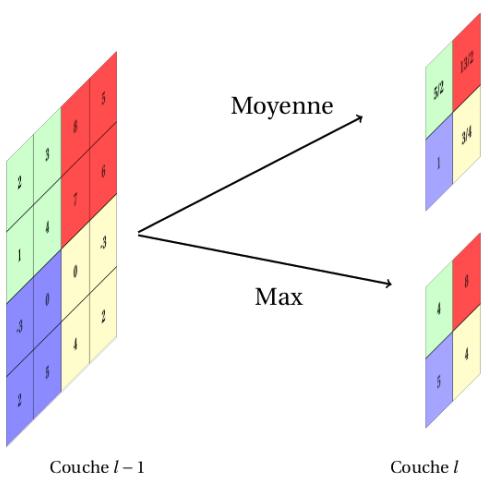


FIGURE 2-4 – Couche d'agrégation et de sous-échantillonnage  $l$ . Chacune des  $n^{(l-1)}$  cartes de la couche  $l-1$  est traitée individuellement. Chaque neurone des  $n^{(l)} = n^{(l-1)}$  cartes de sortie est la moyenne (ou le maximum) des valeurs contenues dans une fenêtre de taille donnée dans la carte correspondante de la couche  $(l-1)$ .

**Le maximum** : la valeur maximum dans la fenêtre est retenue.

Le maximum est souvent utilisé pour assurer une convergence rapide durant la phase d'entraînement. L'agrégation avec recouvrement, elle, semble assurer une réduction du phénomène de surapprentissage

## 2.5- Couche complètement connectée

---

Si  $l$  et  $(l-1)$  sont des couches complètement connectées, l'équation :

$$z_i^{(l)} = \sum_{k=0}^{m^{(l-1)}} w_{i,k}^{(l)} y_k^{(l-1)} \quad \text{ou} \quad \mathbf{Z}^{(l)} = \mathbf{W}^{(l)} \mathbf{Y}^{(l-1)} \quad (9)$$

avec  $\mathbf{Z}^{(l)}$ ,  $\mathbf{W}^{(l)}$  et  $\mathbf{Y}^{(l-1)}$  les représentations vectorielle et matricielle des entrées  $z_i^{(l)}$ , des poids  $w_{i,k}^{(l)}$  et des sorties  $y_k^{(l-1)}$ , permet de relier ces deux couches.

Dans le cas contraire, la couche  $l$  attend  $n^{(l-1)}$  entrées de taille  $n_1^{(l-1)} \times n_2^{(l-1)}$  et le  $i^e$  neurone de la couche

$l$  calcule :

$$y_i^{(l)} = f(z_i^{(l)}) \quad \text{avec} \quad z_i^{(l)} = \sum_{j=1}^{n^{(l-1)}} \sum_{r=1}^{n_1^{(l-1)}} \sum_{s=1}^{n_2^{(l-1)}} w_{i,j,r,s}^{(l)} (\mathbf{Y}_j^{(l-1)})_{r,s} \quad (10)$$

où  $w_{i,j,r,s}^{(l)}$  est le poids connectant le neurone en position  $(r,s)$  de la  $j^e$  carte de la couche  $(l-1)$  au  $i^e$  neurone de la couche  $l$ .

En pratique, les réseaux convolutifs sont utilisés pour apprendre une hiérarchie dans les données et la (ou les) couche(s) complètement connectée(s) est(sont) utilisée(s) en bout de réseau pour des tâches de classification ou de régression.

Une couche de classification classiquement mise en œuvre utilise le classifieur softmax, qui généralise la régression logistique au cas multiclass ( $k$  classes). L'ensemble d'apprentissage  $\mathcal{E}_a = \{(\mathbf{x}^{(i)}, y^{(i)}), i \in [1 \dots m]\}$  est donc tel que  $y^{(i)} \in [1 \dots k]$  et le classifieur estime la probabilité  $P(y^{(i)} = j | \mathbf{x}^{(i)})$  pour chaque classe  $1 \leq j \leq k$ . Le classifieur softmax calcule cette probabilité selon :

$$\forall j \in [1 \dots k] \quad P(y^{(i)} = j | \mathbf{x}^{(i)}, \mathbf{W}) = \frac{e^{\mathbf{W}_j^\top \mathbf{x}^{(i)}}}{\sum_{l=1}^k e^{\mathbf{W}_l^\top \mathbf{x}^{(i)}}} \quad (11)$$

où  $\mathbf{W}$  est la matrice des paramètres du modèle (les poids). Ces paramètres sont obtenus en minimisant une fonction de coût, qui peut par exemple s'écrire :

$$J(\mathbf{W}) = -\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^k \mathbb{I}_{y^{(i)}=j} \log \left( \frac{e^{\mathbf{W}_j^\top \mathbf{x}^{(i)}}}{\sum_{l=1}^k e^{\mathbf{W}_l^\top \mathbf{x}^{(i)}}} \right) + \frac{\lambda}{2} \sum_{i=1}^n \sum_{j=1}^k W_{ji}^2 \quad (12)$$

où  $\lambda$  est un paramètre de régularisation contrôlant le second terme du coût qui pénalise les grandes valeurs des poids (régularisation  $\ell_2$ ).

## 3- RÉGULARISATION

Un des enjeux principaux en apprentissage automatique est de construire des algorithmes ayant une bonne capacité de généralisation. Les stratégies mises en œuvre pour arriver à cette fin rentrent dans la catégorie générale de la régularisation et de nombreuses méthodes sont aujourd'hui proposées en ce sens. Nous faisons ici un focus sur trois stratégies largement utilisées en apprentissage profond.

### 3.1- Régularisation de la fonction de coût

L'équation .12 est un exemple de régularisation de la fonction de coût, utilisée lors de la phase d'entraînement. À la fonction d'erreur est ajoutée une fonction des poids du réseau, qui peut prendre de multiples formes. Les deux principales stratégies sont :

- La régularisation  $\ell_2$  (ou ridge regression), qui force les poids à avoir une faible valeur absolue : un terme de régularisation fonction de la norme  $\ell_2$  de la matrice des poids est ajouté (à la manière de l'équation .12). On parle souvent de weight decay.
- La régularisation  $\ell_1$ , qui tend à rendre épars le réseau profond, *i.e.* à imposer à un maximum de poids de s'annuler. Un terme de régularisation, somme pondérée des valeurs absolues des poids, est ajouté à la fonction objectif.

## 3.2- Dropout

Les techniques de dropout se rapprochent des stratégies classiques de bagging en apprentissage automatique. L'objectif est d'entraîner un ensemble constitué de tous les sous-réseaux qui peuvent être construits en supprimant des neurones (hors neurones d'entrée et de sortie) du réseau initial. Si le réseau comporte  $|W|$  neurones cachés, il existe ainsi  $2^{|W|}$  modèles possibles. En pratique, les neurones cachés se voient perturbés par un bruit binomial, qui a pour effet de les empêcher de fonctionner en groupe et de les rendre, au contraire, plus indépendants. Le phénomène de surapprentissage est ainsi fortement réduit sur le réseau, qui doit décomposer les entrées en caractéristiques pertinentes, indépendamment les unes des autres. Les réseaux construits par dropout partagent partiellement leurs paramètres, ce qui diminue l'empreinte mémoire de la méthode.

Lors de la phase de prédiction, le réseau complet est utilisé, mais les neurones cachés sont pondérés par la fraction de bruit utilisé pendant l'apprentissage (*i.e.* pour chaque neurone le nombre de fois où il a été supprimé d'un sous-réseau, rapporté au nombre total de réseaux), afin de conserver la valeur moyenne des activations des neurones identiques à celles durant l'apprentissage.

Notons qu'il est également possible d'éteindre non pas un neurone, mais un poids. La stratégie correspondante est appelée DropConnect.

## 3.3- Partage de paramètres

La régularisation de la fonction de coût permet d'imposer aux poids certaines contraintes (par exemple de rester faibles en amplitude pour la régularisation  $\ell_2$ , ou de s'annuler pour la régularisation  $\ell_1$ ). Il peut également être intéressant d'imposer certains a priori sur les poids, par exemple une dépendance entre les valeurs des paramètres.

Une dépendance classique consiste à imposer que les valeurs de certains poids soient proches les unes des autres (dans le cas par exemple où deux modèles de classification  $M_1$  et  $M_2$ , de paramètres  $W_1$  et  $W_2$ , opèrent sur des données similaires et sur des classes identiques) et, là encore, une stratégie de pénalisation de la fonction objectif peut être utilisée. Cependant, il est plus courant dans ce cas d'imposer que les paramètres soient égaux (dans l'exemple précédent imposer  $W_1 = W_2$ ) et d'arriver à une stratégie dite de partage des paramètres. Dans le cas des réseaux convolutifs utilisés en vision, cette régularisation est assez intuitive puisque les entrées (images) possèdent de nombreuses propriétés invariantes par transformations affines (une image de voiture reste une image de voiture, même si l'image est translatée ou mise à l'échelle, cf. figure 3-5). Le réseau exploite alors ce partage de paramètres, en calculant une même caractéristique (un neurone et son poids) à différentes positions dans l'image. De ce fait, le nombre de paramètres est drastiquement réduit, ainsi que l'empreinte mémoire du réseau appris.

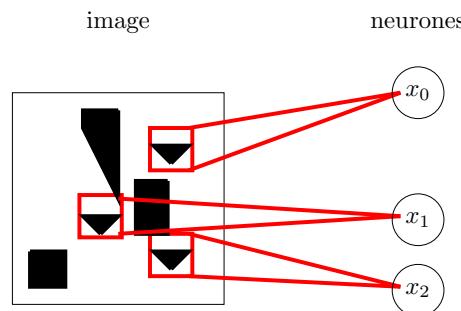


FIGURE 3-5 – Partage de paramètres : les neurones voient des champs réceptifs distincts, mais partagent les mêmes paramètres (poids). Leur capacité de détection d'un triangle restera la même, quelle que soit la position de l'objet dans l'image.

## 4- INITIALISATION

Une initialisation convenable des poids est essentielle pour assurer une convergence de la phase d'entraînement. Un choix arbitraire des poids (à zéro, à de petites ou grandes valeurs aléatoires) peut ralentir, voire causer de la redondance dans le réseau (problème de la symétrie).

Plusieurs schémas d'initialisation ont été proposés et nous donnons dans les deux paragraphes qui suivent d'eux d'entre eux.

## 4.1- Initialisation prenant en compte les variations des neurones

Pour illustrer le propos, on suppose que l'entrée du réseau de neurones est composée de  $n^{(1)}$  entrées  $\mathbf{x} = (x_1 \dots x_{n^{(1)}})^\top$ , les  $x_i$  étant i.i.d, normalisés selon une loi  $\mathcal{N}(0, \sigma_x)$ . Pour simplifier la démonstration, on suppose que la couche suivante calcule un simple potentiel post synaptique : pour un neurone de cette couche, ce potentiel est  $\mathbf{w}^\top \mathbf{x}$ , avec  $\mathbf{w} \in \mathbb{R}^{n^{(1)}}$  i.i.d  $\mathcal{N}(0, \sigma_w)$ . La variance de ce potentiel est alors

$$Var(\mathbf{w}^\top \mathbf{x}) = \sum_{i=1}^{n^{(1)}} Var(w_i x_i) = \sum_{i=1}^{n^{(1)}} (\mathbb{E}(w_i)^2 Var(x_i) + \mathbb{E}(x_i)^2 Var(w_i) + Var(w_i) Var(x_i))$$

Les entrées et les poids sont de moyenne nulle, donc

$$Var(\mathbf{w}^\top \mathbf{x}) = \sum_{i=1}^{n^{(1)}} \sigma_x \sigma_w$$

et puisque les  $w_i x_i$  sont i.i.d.

$$Var(\mathbf{w}^\top \mathbf{x}) = n^{(1)} \sigma_x \sigma_w$$

On montre plus généralement que sur la  $l^e$  couche cachée, la variance de  $\mathbf{Y}^{(l)}$  est

$$Var(\mathbf{Y}^{(l)}) = (n^{(l)} Var(w_i))^l Var(x_i)$$

Chaque neurone peut donc varier dans un rapport de  $n^{(l)}$  fois la variation de son entrée (qui est elle même  $n^{(l-1)}$  fois la variance de son entrée...)

On a alors les cas de figure suivants :

- si  $n^{(l)} Var(w_i) > 1$  le gradient va tendre vers de grandes valeurs à mesure que l'on s'enfonce dans le réseau (que  $l$  croît)
- si  $n^{(l)} Var(w_i) < 1$  le gradient disparaît à mesure que l'on s'enfonce dans le réseau

Pour éviter ces deux problèmes, la solution est de forcer  $n^{(l)} Var(w_i) = 1$ , soit  $Var(w_i) = 1/n^{(l)}$ . On initialise donc les poids se la couche  $l$  selon une loi

$$\frac{1}{\sqrt{n^{(l-1)}}} \mathcal{N}(0, 1)$$

Cette procédure est la méthode d'initialisation de Xavier, ou de Glorot [4].

## 4.2- Initialisation de He

He a montré dans [7] que la méthode de Xavier pouvait ne pas fonctionner correctement lorsque les traitements non linéaires étaient effectués par la fonction ReLU. Les auteurs proposent alors de plutôt multiplier par  $\frac{\sqrt{2}}{\sqrt{n^{(l-1)}}}$  pour prendre en compte la partie négative qui ne participe pas au calcul de la variance.

## 5- APPRENTISSAGE

La présence de nombreuses couches cachées va permettre de calculer des caractéristiques beaucoup plus complexes et informatives des entrées. Chaque couche calculant une transformation non linéaire de la couche précédente, le pouvoir de représentation de ces réseaux s'en trouve amélioré. On peut par exemple montrer qu'il existe des fonctions qu'un réseau à  $k$  couches peut représenter de manière compacte (avec un nombre de neurones cachés qui est polynomial en le nombre des entrées), alors qu'un réseau à  $k-1$  couches ne peut pas le faire, à moins d'avoir une combinatoire exponentielle sur le nombre de neurones cachés.

## 5.1- Le problème de l'entraînement

---

Si l'intérêt de ces réseaux est manifeste, la complexité de leur utilisation vient de l'étape d'apprentissage. Jusqu'à récemment, l'algorithme utilisé était classique et consistait en une initialisation aléatoire des poids du réseau, suivie d'un entraînement sur un ensemble d'apprentissage, en minimisant une fonction objectif. Cependant, dans le cas des réseaux profonds, cette approche peut ne pas être adaptée :

- Les données étiquetées doivent être en nombre suffisant pour permettre un entraînement efficace, d'autant plus que le réseau est complexe. Dans le cas contraire, un surapprentissage peut notamment être induit.
- Sur un tel réseau, l'apprentissage se résume à l'optimisation d'une fonction fortement non convexe, qui amène presque sûrement à des minima locaux lorsque des algorithmes classiques sont utilisés.
- Dans l'étape de rétropropagation, les gradients diminuent rapidement à mesure que le nombre de couches cachées augmente. La dérivée de la fonction objectif par rapport à  $\mathbf{W}$  devient alors très faible à mesure que le calcul se rétropropage vers la couche d'entrée. Les poids des premières couches changent donc très lentement et le réseau n'est plus en capacité d'apprendre. Ce problème est connu sous le nom de problème de gradient évanescence (vanishing gradient).

L'algorithme principalement utilisé pour l'apprentissage des réseaux convolutifs reste la rétropropagation du gradient. Le choix de la fonction objectif, de sa régularisation, de la méthode d'optimisation (descente de gradient, méthodes à taux d'apprentissage adaptatifs telles qu'AdaGrad, RMSProp ou Adam) et des paramètres associés, ou des techniques de présentation des exemples (batchs, minibatchs) sont autant de facteurs importants permettant aux modèles non seulement de converger vers un optimum local satisfaisant, mais également de proposer un modèle final ayant une bonne capacité de généralisation.

Aujourd'hui, de nombreux réseaux, déjà entraînés, sont mis à disposition. En effet, ces entraînements nécessitent de grandes bases d'apprentissage (type ImageNet) et une puissance de calcul assez élevée (GPU obligatoire(s)). Pour le traitement de problèmes précis, des méthodes existent, qui partent de ces réseaux préentraînés et les modifient localement pour, par exemple, apprendre de nouvelles classes d'images non encore vues par le réseau. L'idée sous-jacente est que les premières couches capturent des caractéristiques bas niveau et que la sémantique vient avec les couches profondes. Ainsi, dans un problème de classification, où les classes n'ont pas été apprises, on peut supposer qu'en conservant les premières couches on extraite des caractéristiques communes des images (bords, colorimétrie,...) et qu'en changeant les dernières couches (information sémantique et haut niveau et étage de classification), c'est-à-dire en réapprenant les connexions, on spécifiera le nouveau réseau pour la nouvelle tâche de classification. Cette approche rentre dans le cadre des méthodes de *transfer learning* [10] et de *fine tuning*, cas particulier d'adaptation de domaine :

- Les méthodes de transfert prennent un réseau déjà entraîné, enlèvent la dernière couche complètement connectée et traitent le réseau restant comme un extracteur de caractéristiques. Un nouveau classifieur, la dernière couche, est alors entraîné sur le nouveau problème.
- Les méthodes de fine tuning ré-entraînent le classifieur du réseau et remettent à jour les poids du réseau pré-entraîné par rétropropagation.

Plusieurs facteurs influent sur le choix de la méthode à utiliser : la taille des données d'apprentissage du nouveau problème et la ressemblance du nouveau jeu de données avec celui qui a servi à entraîner le réseau initial :

- Pour un jeu de données similaire de petite taille, on utilise du transfer learning, avec un classifieur utilisé sur les caractéristiques calculées sur les dernières couches du réseau initial.
- Pour un jeu de données de petite taille et un problème différent, on utilise du transfer learning, avec un classifieur utilisé sur les caractéristiques calculées sur les premières couches du réseau initial.
- Pour un jeu de données, similaire ou non, de grande taille, on utilise le fine tuning.

Notons qu'il est toujours possible d'augmenter la taille du jeu de données par une technique d'augmentation de données.

Dans le cas où un réseau ad hoc doit être construit et où une base d'apprentissage suffisante est disponible, l'entraînement par optimisation reste possible. Il existe en particulier des techniques d'apprentissage couche à couche, lorsque les couches successives calculent des fonctions d'activation des couches précédentes (empilement d'autoencodeurs par exemple).

## 5.2- Apprentissage glouton par couche

L'idée est d'entraîner les couches une à une, d'abord dans un réseau à une couche cachée, puis à deux couches cachées... À chaque étape  $k$ , la couche  $k$  est ajoutée et a pour entrée la couche  $k - 1$  précédemment entraînée. L'entraînement peut être supervisé, mais le plus souvent il est non supervisé. Les poids issus de cet entraînement servent d'initialisation pour le réseau final.

En comparaison des points précédents, cette approche est bien plus pertinente :

- Les données non étiquetées sont très faciles à obtenir.
- L'initialisation des poids sur des données non étiquetées est plus performante qu'une initialisation aléatoire. Empiriquement, une méthode type descente de gradient permet d'aboutir à un meilleur minimum local (les données non étiquetées fournissent en effet des informations a priori déjà importantes sur les données).

## 5.3- Visualisation du mécanisme des réseaux convolutifs

Le mécanisme interne des réseaux convolutifs est mal compris et l'analyse des raisons qui font que leur puissance de prédiction est importante n'est pas aisée. S'il est toujours possible de rétroprojeter les activations depuis la première couche de convolution, les couches d'agrégation et de rectification empêchent de comprendre le fonctionnement des couches suivantes, ce qui peut être gênant dans la construction et l'amélioration de ces réseaux.

Les méthodes de visualisation du fonctionnement des réseaux convolutifs peuvent être rangées en trois catégories, décrites dans les paragraphes suivants.

## 5.4- Méthodes de visualisation de base

Les méthodes les plus simples consistent à visualiser les activations lors du passage d'une image dans le réseau. Pour des activations type ReLU, ces activations sont ininterprétables au début de l'entraînement, mais à mesure que ce dernier progresse, les cartes d'activation  $Y_i^{(l)}$  deviennent localisées et éparses.

Il est également possible de visualiser les filtres des différentes couches de convolution (figure 5-6). Les filtres des premières couches agissent comme des détecteurs de bords et coins et, à mesure que l'on s'enfonce dans le réseau, les filtres capturent des concepts haut niveau comme des objets ou encore des visages. Citons encore d'autres méthodes qui proposent de visualiser les dernières couches (les couches complètement connectées) de grande dimension (par exemple 4096 pour AlexNet) via une méthode de réduction de dimension.

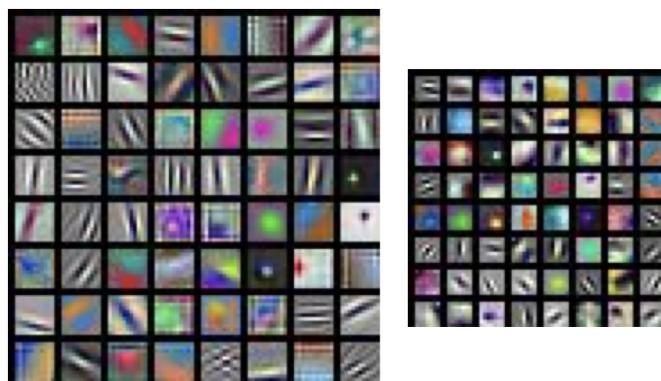


FIGURE 5-6 – Visualisation des filtres de la première couche d'AlexNet (à gauche, 64 filtres  $11 \times 11$ ) et de ResNet-18 (à droite, 64 filtres  $7 \times 7$ ).

## 5.5- Méthodes fondées sur les activations

Plusieurs stratégies peuvent être adoptées pour sonder le fonctionnement d'un réseau convolutif, en utilisant les informations portées par les cartes  $Y_i^{(l)}$ , parmi lesquelles :

- Utiliser des couches de convolution transposée (improprement appelées parfois couches de déconvolution), ajoutées à chaque couche de convolution du réseau. Étant données les cartes d'entrée de la couche  $l$ , les cartes de sortie  $Y_i^{(l)}$  sont envoyées dans la couche de convolution transposée correspondante au niveau  $l$ . Cette dernière reconstruit les  $Y_i^{(l-1)}$  qui ont permis le calcul des activations de la couche  $l$ . Le processus est alors itéré jusqu'à atteindre la couche d'entrée  $l = 1$ , les activations de la couche  $l$  étant alors rétroprojectées dans le plan image [13]. La présence de couches d'agrégation et de rectification rend ce processus non inversible (par exemple, une couche d'agrégation maximum nécessite de connaître à quelles positions de l'image  $Y_i^{(l)}$  sont situés les maxima retenus).
- Faire passer un grand nombre d'images dans le réseau et, pour un neurone particulier, conserver celle qui a le plus activé ce neurone. Il est alors possible de visualiser les images pour comprendre ce à quoi le neurone s'intéresse dans son champ réceptif (figure 5-7).



FIGURE 5-7 – Champ réceptif de quelques neurones de la dernière couche d'agrégation du réseau AlexNet, superposées aux images ayant le plus fortement activé ces neurones. Le champ est encadré en blanc, et la valeur d'activation correspondante est reportée en haut. On voit par exemple que certains neurones sont très sensibles aux textes, d'autres aux réflexions spéculaires, ou encore aux hauts du corps (source : [3]).

- Cacher (par un rectangle noir par exemple) différentes parties de l'image d'entrée qui est d'une certaine classe (disons un chien) et observer la sortie du réseau (la probabilité de la classe de l'image d'entrée). En représentant les valeurs de probabilité de la classe d'intérêt comme une fonction de la position du rectangle occultant, il est possible de voir si le réseau s'intéresse effectivement aux parties de l'image spécifiques de la classe, ou à des autres zones (le fond par exemple) (figure 5-8).

## 5.6- Méthodes fondées sur le gradient

Pour comprendre quelle(s) partie(s) de l'image est (sont) utilisée(s) par le réseau pour effectuer une prédiction, il est possible de calculer des cartes de saillance (saliency maps). L'idée est relativement simple : calculer le gradient de la classe de sortie par rapport à l'image d'entrée. Cela indique à quel point une petite variation dans l'image induit un changement de prédiction. En visualisant les gradients, on observe alors par exemple leurs fortes valeurs, indiquant qu'une petite variation du pixel correspondant augmente la valeur de sortie. Il est également possible d'utiliser le gradient par rapport à la dernière couche de convolution (approche Grad-CAM), ce qui permet de récupérer des informations de localisation spatiale des régions importantes pour la prédiction (figure 5-9, droite).

Plus généralement, en choisissant un neurone intermédiaire du réseau (d'une couche de convolution), la méthode de rétropropagation guidée calcule le gradient de sa valeur par rapport aux pixels de l'image

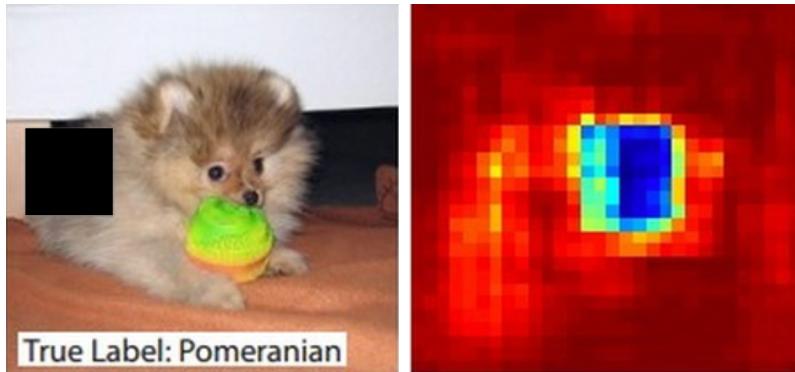


FIGURE 5-8 – Occlusion d’une image (à gauche). Le rectangle noir est déplacé dans l’image et pour chaque position la probabilité de la classe de l’image (ici un loulou de Poméranie) est enregistrée. Ces probabilités sont ensuite représentées sous forme d’une carte 2D (à droite). La probabilité de la classe s’effondre lorsque le rectangle couvre une partie de la face du chien. Cela suggère que cette face est grandement responsable de la forte probabilité de classement de l’image comme un loulou. A l’inverse, l’occlusion du fond n’altère pas la forte valeur de probabilité de la classe (source : [13]).

d’entrée, ce qui permet de souligner les parties de l’image auxquelles ce neurone répond (figure 5-9, milieu).

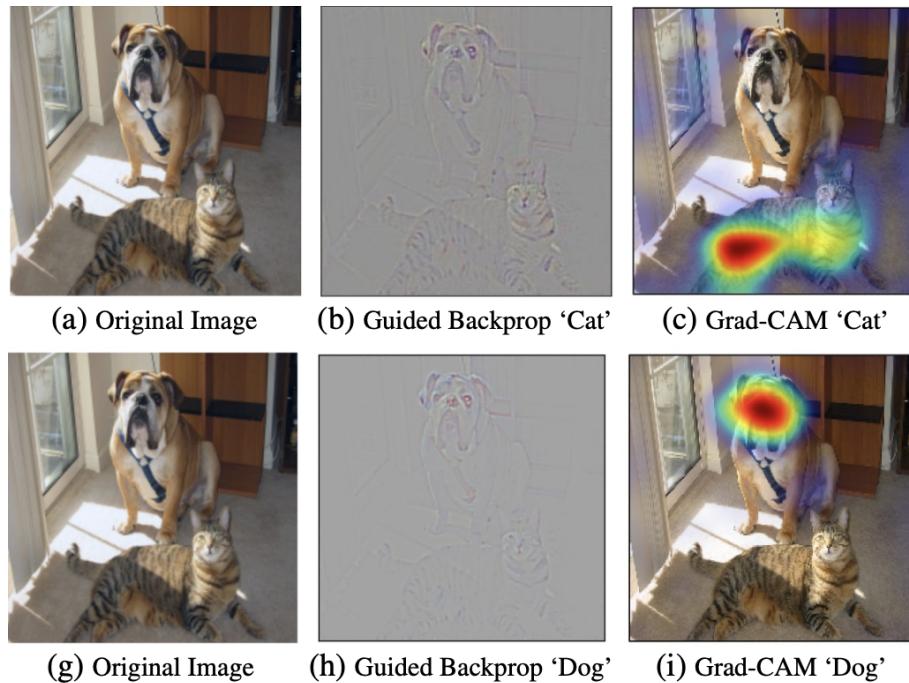


FIGURE 5-9 – Approches par gradient de visualisation du fonctionnement d’un réseau convolutif. Comparaison de la méthode de rétropropagation guidée et de Grad-CAM.(source : [11]).

Ces gradients peuvent également être utilisés dans la méthode de montée de gradient (gradient Ascent), dont l’objectif est de générer une image qui active de manière maximale un neurone donné du réseau. Le principe est d’itérativement passer l’image d’entrée  $\mathbf{I}$  dans le réseau pour obtenir les valeurs des cartes  $\mathbf{Y}_i^{(l)}$ , de rétropropager pour obtenir le gradient d’un neurone par rapport aux pixels de  $\mathbf{I}$  et d’opérer une petite modification de ces pixels. Outre son aspect informatif sur la structure interne du réseau étudié (visualisation

des cartes  $\mathbf{Y}_i^{(l)}$  intermédiaires), cette méthode produit des images parfois très artistiques (figure 5-10).

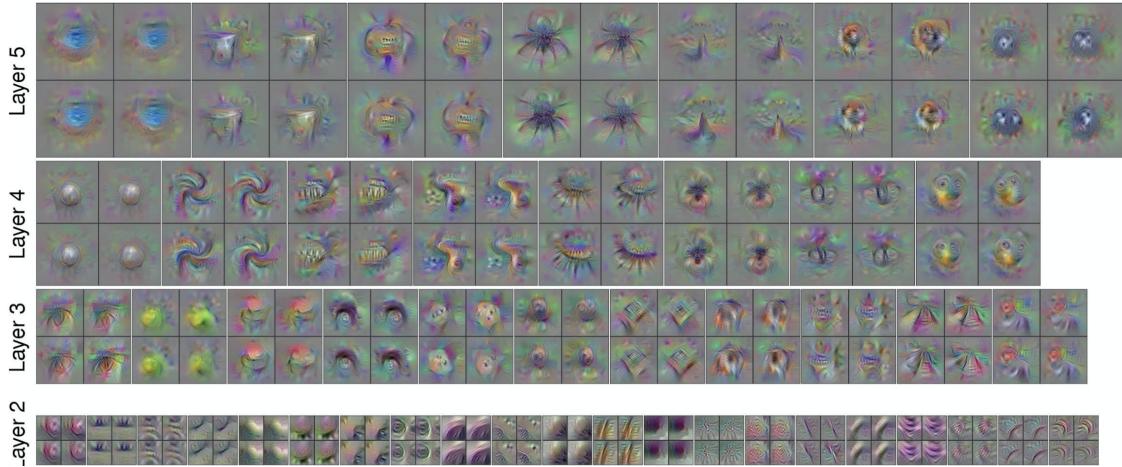


FIGURE 5-10 – Visualisation des 4 premières couches de convolution d'un réseau convolutif par montée de gradient (source : [12]).

## 6- QUELQUES APPLICATIONS

Depuis leur introduction en reconnaissance de caractères manuscrits [1], les réseaux convolutifs n'ont cessé de trouver des champs applicatifs, commerciaux et parfois ludiques. Parmi ces applications, nous en citons ici quelques-unes.

### 6.0.0-1 La classification d'images

Premier domaine d'application des réseaux convolutifs, la classification d'images consiste à affecter une image à une classe, apprise par le réseau sur un grand nombre d'exemples. Depuis l'avènement d'ImageNet, et la mise en place de la compétition ILSVRC, les résultats obtenus ne cessent de s'améliorer et sont depuis quelques années la référence dans ce domaine (dépassant même les performances humaines en 2015).

### 6.0.0-2 L'annotation de scènes

Des réseaux convolutifs ont été utilisés pour annoter des scènes 2D ou 2D+t, *i.e.* assigner à chaque pixel un label identifiant l'objet auquel il appartient. De nombreux réseaux ont été développés à cet effet (R-CNN, Fast R-CNN, Mask R-CNN par exemple) (figure 6-11(a)).

### 6.0.0-3 La reconnaissance d'actions

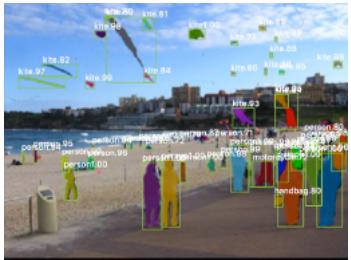
Le développement de champs réceptifs 3D dans les réseaux convolutifs a permis d'extraire dans ces réseaux des caractéristiques invariantes à la translation. L'intégration de techniques de régularisation adaptée (type partage de paramètres) a rendu possible l'optimisation de ces réseaux pour la reconnaissance d'actions dans des scènes dynamiques.

### 6.0.0-4 L'analyse de documents

L'analyse de documents à des fins de reconnaissance de caractères, de classification de documents ou encore d'annotation sémantique a largement bénéficié de l'apport des réseaux convolutifs.

### 6.0.0-5 L'augmentation de données

De nombreux réseaux ont été proposés pour ajouter à des données nD des informations manquantes (colorisation d'images, figure 6-11(b), inpainting, restauration d'images, superrésolution), ou pour proposer des



(a) Annotation sémantique (source [6]).



(b) Colorisation d'images (source [14])



(c) transfert de style (source [2])

FIGURE 6-11 – Quelques applications des réseaux convolutifs.

versions différentes des données initiales en fonction d'une contrainte de style extérieure (transfert de style, figure 6-11(c)).

Les réseaux convolutifs sont des réseaux spécialisés pour traiter des données dont la topologie se conforme à une structure de grille n-dimensionnelle. Dans le cas de données 1D séquentielles, d'autres réseaux performants ont été développés : les réseaux récurrents.

## 7- PARTIE PRATIQUE

A l'aide du notebook fourni, réalisez le réseau convolutif de la figure 7-12.

La fonction de perte utilisée est l'entropie croisée et l'algorithme d'optimisation ADAM.

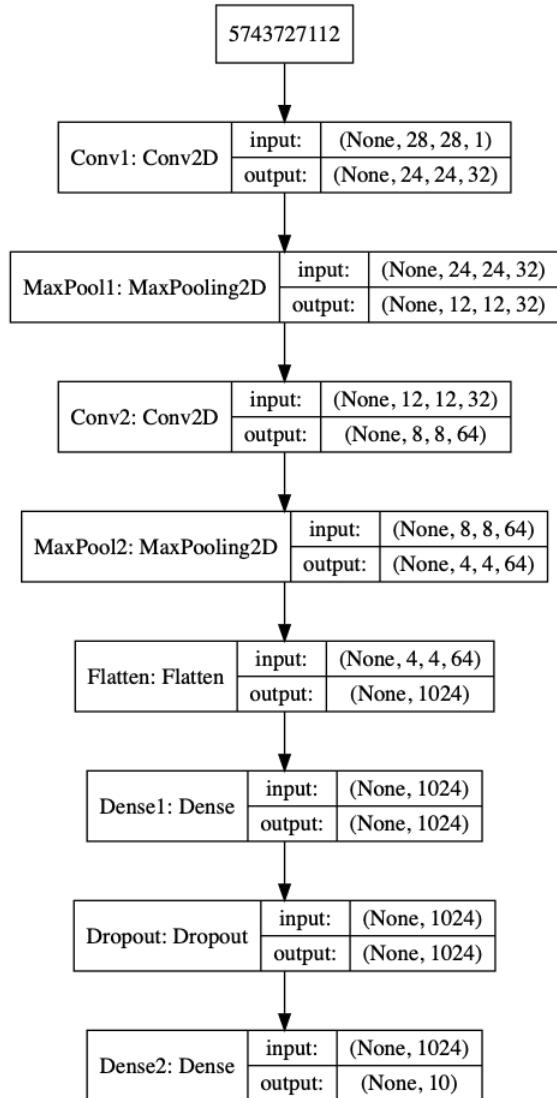


FIGURE 7-12 – Architecture du CNN à construire

# BIBLIOGRAPHIE

- [1] Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Hubbard, L. D. Jackel, and D. Henderson. Advances in neural information processing systems 2. chapter Handwritten Digit Recognition with a Back-propagation Network, pages 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.
- [2] L.A. Gatys, A.S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.
- [3] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 580–587, June 2014.
- [4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS’10)*. Society for Artificial Intelligence and Statistics, 2010.
- [5] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudák, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011.
- [6] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.
- [7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [8] Sergey Ioffe and Christian Szegedy. Batch normalization : Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015.
- [9] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In Johannes Finkenkranz and Thorsten Joachims, editors, *ICML*, pages 807–814. Omnipress, 2010.
- [10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10) :1345–1359, October 2010.
- [11] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam : Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626. IEEE Computer Society, 2017.
- [12] Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *CoRR*, abs/1506.06579, 2015.
- [13] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.
- [14] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris N. Metaxas. Stackgan : Text to photo-realistic image synthesis with stacked generative adversarial networks. *CoRR*, abs/1612.03242, 2016.