

(<https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#cs-230---deep-learning>)CS 230 - Apprentissage profond (l/fr/teaching/cs-230)

Français



Réseaux convolutionnels

Réseaux récurrents

Petites astuces

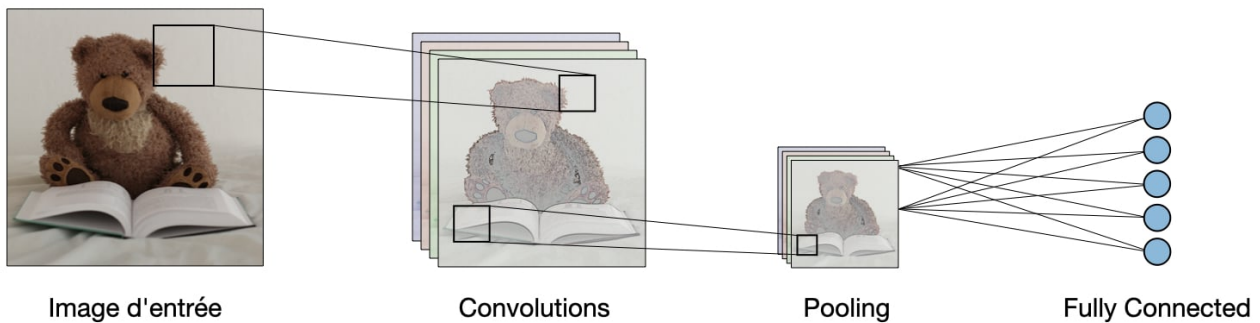
(<https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#cheatsheet>)Pense-bête de réseaux de neurones convolutionnels

☆ Star 5,173

Par Afshine Amidi (<https://twitter.com/afshinea>) et Shervine Amidi (<https://twitter.com/shervinea>)

(<https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#overview>) Vue d'ensemble

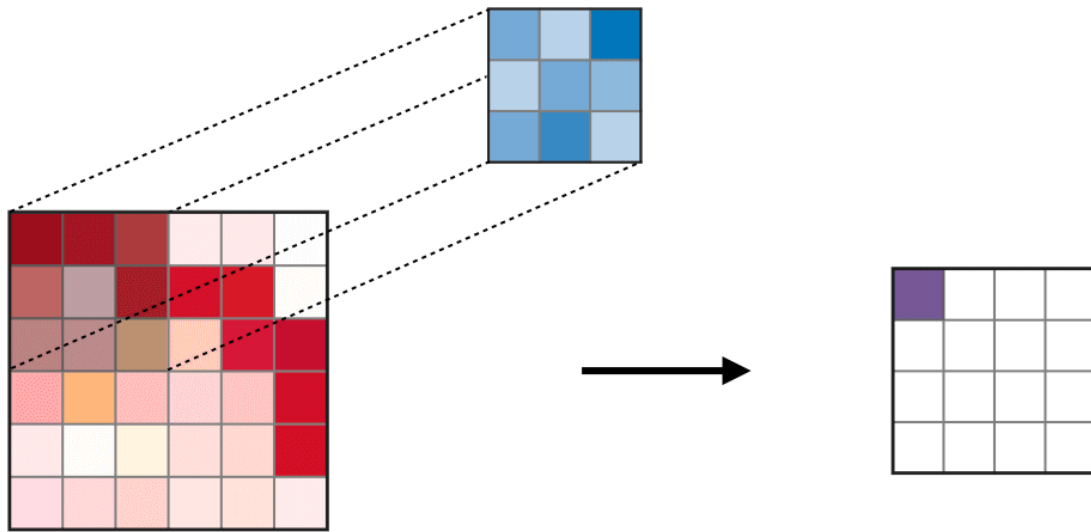
□ **Architecture d'un CNN traditionnel** — Les réseaux de neurones convolutionnels (en anglais *Convolutional neural networks*), aussi connus sous le nom de CNNs, sont un type spécifique de réseaux de neurones qui sont généralement composés des couches suivantes :



La couche convolutionnelle et la couche de pooling peuvent être ajustées en utilisant des paramètres qui sont décrites dans les sections suivantes.

(<https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#layer>) Types de couche

□ **Couche convolutionnelle (CONV)** — La couche convolutionnelle (en anglais *convolution layer*) (CONV) utilise des filtres qui scannent l'entrée I suivant ses dimensions en effectuant des opérations de convolution. Elle peut être réglée en ajustant la taille du filtre F et le stride S . La sortie O de cette opération est appelée *feature map* ou aussi *activation map*.

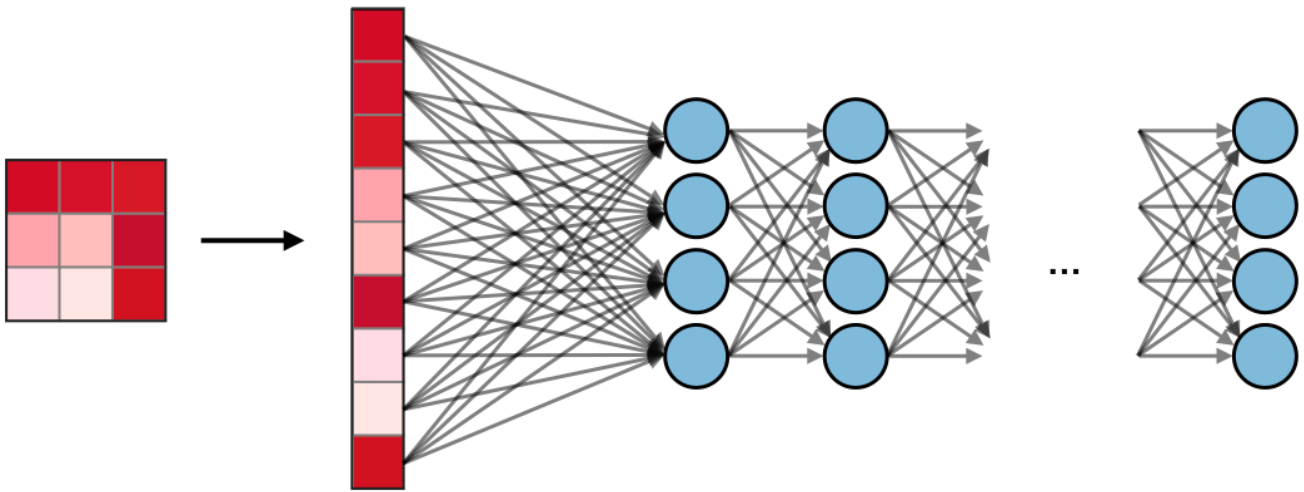


Remarque : l'étape de convolution peut aussi être généralisée dans les cas 1D et 3D.

□ **Pooling (POOL)** — La couche de pooling (en anglais *pooling layer*) (POOL) est une opération de sous-échantillonnage typiquement appliquée après une couche convolutionnelle. En particulier, les types de pooling les plus populaires sont le max et l'average pooling, où les valeurs maximales et moyennes sont prises, respectivement.

Type	Max pooling	Average pooling
But	Chaque opération de pooling sélectionne la valeur maximale de la surface	Chaque opération de pooling sélectionne la valeur moyenne de la surface
Illustration		
Commentaires	<ul style="list-style-type: none"> • Garde les caractéristiques détectées • Plus communément utilisé 	<ul style="list-style-type: none"> • Sous-échantillonne la <i>feature map</i> • Utilisé dans LeNet

□ **Fully Connected (FC)** — La couche de fully connected (en anglais *fully connected layer*) (FC) s'applique sur une entrée préalablement aplatie où chaque entrée est connectée à tous les neurones. Les couches de fully connected sont typiquement présentes à la fin des architectures de CNN et peuvent être utilisées pour optimiser des objectifs tels que les scores de classe.

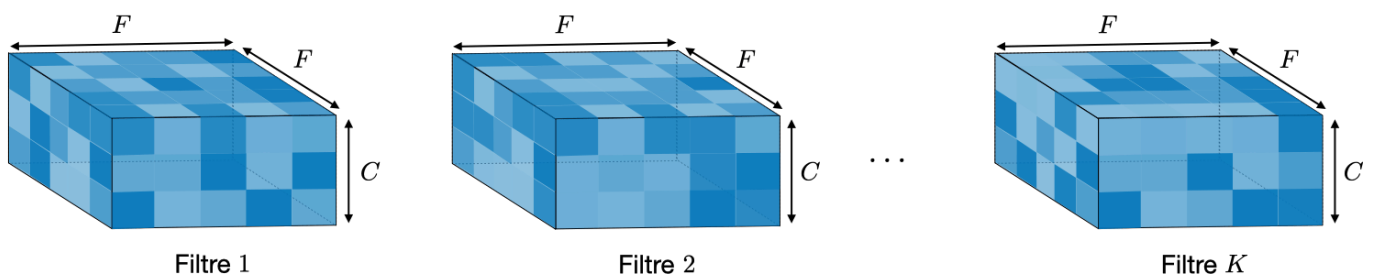


<https://stanford.edu/~shervine/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#filter>

Paramètres du filtre

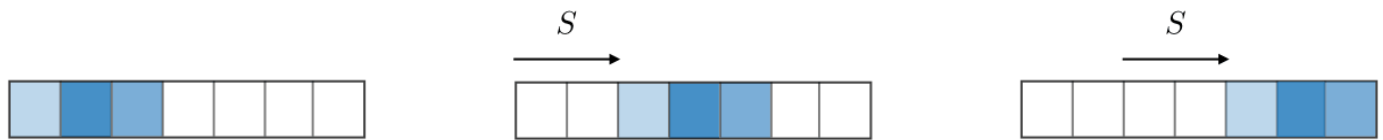
La couche convolutionnelle contient des filtres pour lesquels il est important de savoir comment ajuster ses paramètres.

□ **Dimensions d'un filtre** — Un filtre de taille $F \times F$ appliqué à une entrée contenant C canaux est un volume de taille $F \times F \times C$ qui effectue des convolutions sur une entrée de taille $I \times I \times C$ et qui produit un *feature map* de sortie (aussi appelé *activation map*) de taille $O \times O \times 1$.



Remarque : appliquer K filtres de taille $F \times F$ engendre un feature map de sortie de taille $O \times O \times K$.

□ **Stride** — Dans le contexte d'une opération de convolution ou de pooling, la stride S est un paramètre qui dénote le nombre de pixels par lesquels la fenêtre se déplace après chaque opération.



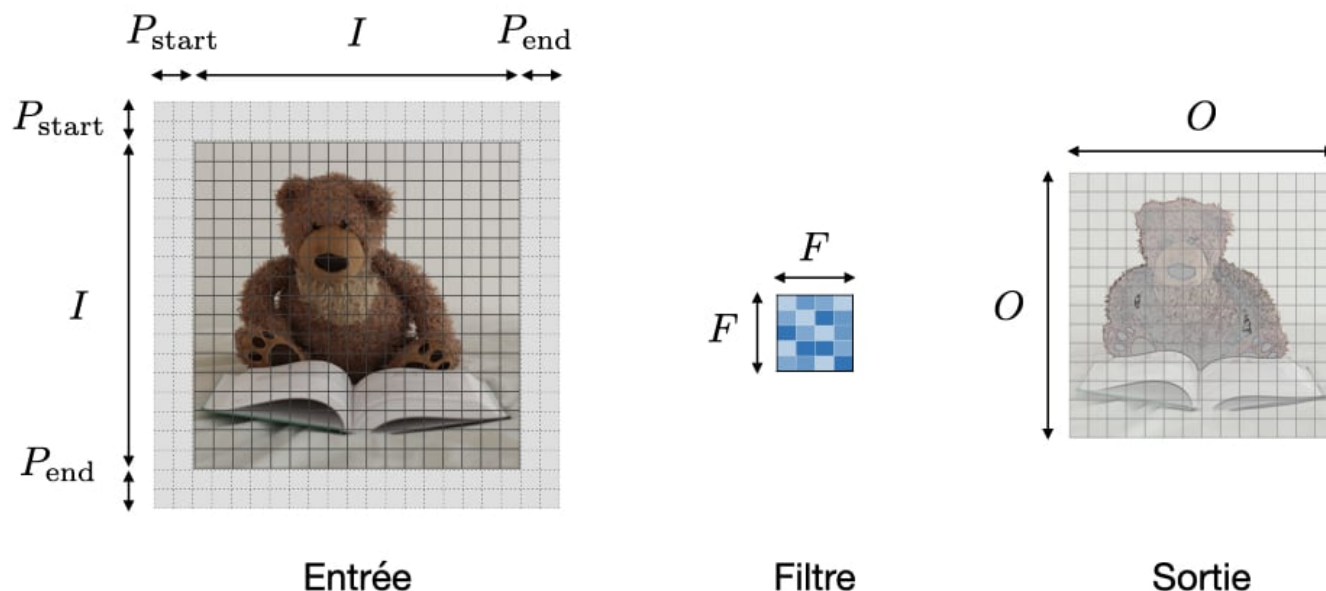
□ **Zero-padding** — Le zero-padding est une technique consistant à ajouter P zeros à chaque côté des frontières de l'entrée. Cette valeur peut être spécifiée soit manuellement, soit automatiquement par le biais d'une des configurations détaillées ci-dessous :

Configuration	Valide	Pareil	Total
Valeur	$P = 0$	$P_{\text{start}} = \left\lfloor \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor$ $P_{\text{end}} = \left\lceil \frac{S \lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil$	$P_{\text{start}} \in \llbracket 0, F - 1 \rrbracket$ $P_{\text{end}} = F - 1$
Illustration			
But	<ul style="list-style-type: none"> • Pas de padding • Enlève la dernière opération de convolution si les dimensions ne collent pas 	<ul style="list-style-type: none"> • Le padding tel que la feature map est de taille $\left\lceil \frac{I}{S} \right\rceil$ • La taille de sortie est mathématiquement satisfaisante • Aussi appelé 'demi' padding 	<ul style="list-style-type: none"> • Padding maximum tel que les dernières convolutions sont appliquées sur les bords de l'entrée • Le filtre 'voit' l'entrée du début à la fin

(<https://stanford.edu/~shervine/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#hyperparameters>)
Réglage des paramètres

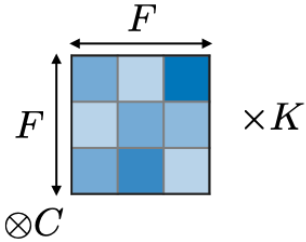
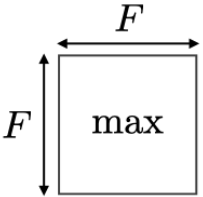
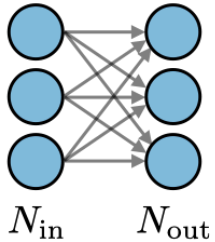
□ **Compatibilité des paramètres dans la couche convolutionnelle** — En notant I le côté du volume d'entrée, F la taille du filtre, P la quantité de zero-padding, S la stride, la taille O de la feature map de sortie suivant cette dimension est telle que :

$$O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1$$



Remarque : on a souvent $P_{\text{start}} = P_{\text{end}} \triangleq P$, auquel cas on remplace $P_{\text{start}} + P_{\text{end}}$ par $2P$ dans la formule au-dessus.

□ **Comprendre la complexité du modèle** — Pour évaluer la complexité d'un modèle, il est souvent utile de déterminer le nombre de paramètres que l'architecture va avoir. Dans une couche donnée d'un réseau de neurones convolutionnels, on a :

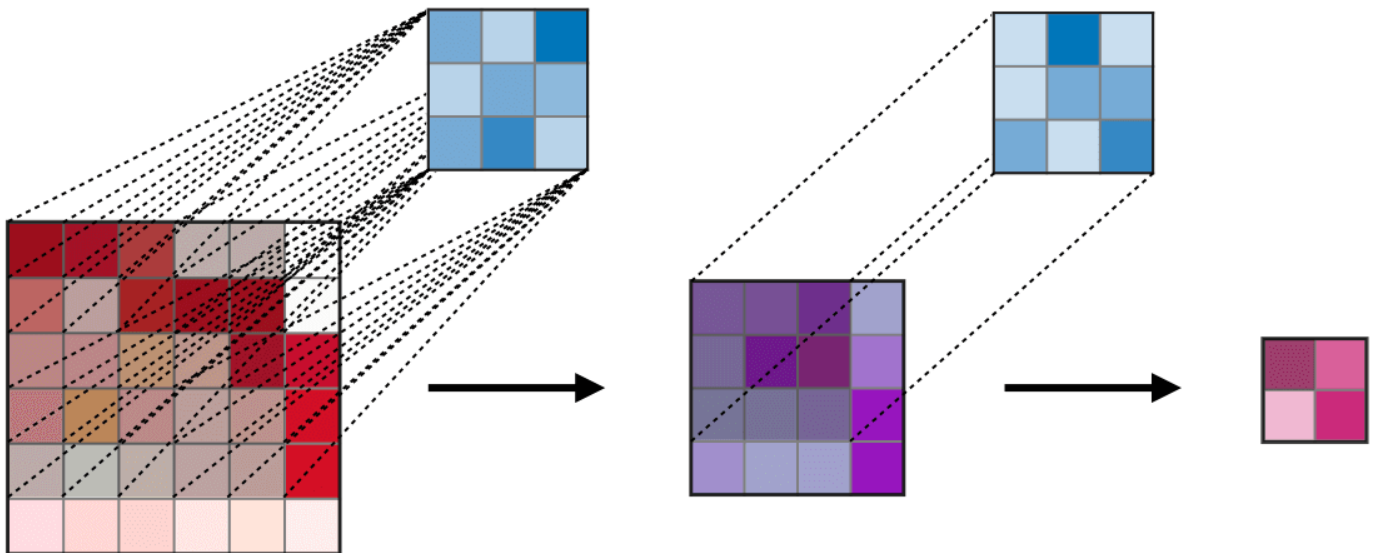
	CONV	POOL	FC
Illustration			
Taille d'entrée	$I \times I \times C$	$I \times I \times C$	N_{in}
Taille de sortie	$O \times O \times K$	$O \times O \times C$	N_{out}

Nombre de paramètres	$(F \times F \times C + 1) \cdot K$	0	$(N_{\text{in}} + 1) \times N_{\text{out}}$
Remarques	<ul style="list-style-type: none"> • Un paramètre de biais par filtre • Dans la plupart des cas, $S < F$ • $2C$ est un choix commun pour K 	<ul style="list-style-type: none"> • L'opération de pooling est effectuée pour chaque canal • Dans la plupart des cas, $S = F$ 	<ul style="list-style-type: none"> • L'entrée est aplatie • Un paramètre de biais par neurone • Le choix du nombre de neurones de FC est libre

□ **Champ récepteur** — Le champ récepteur à la couche k est la surface notée $R_k \times R_k$ de l'entrée que chaque pixel de la k -ième *activation map* peut 'voir'. En notant F_j la taille du filtre de la couche j et S_i la valeur de stride de la couche i et avec la convention $S_0 = 1$, le champ récepteur à la couche k peut être calculé de la manière suivante :

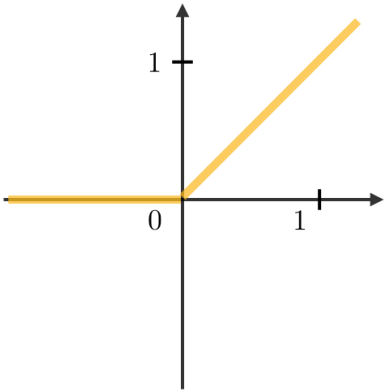
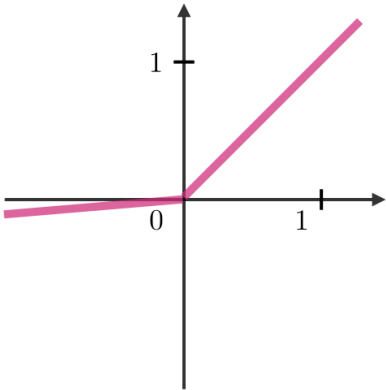
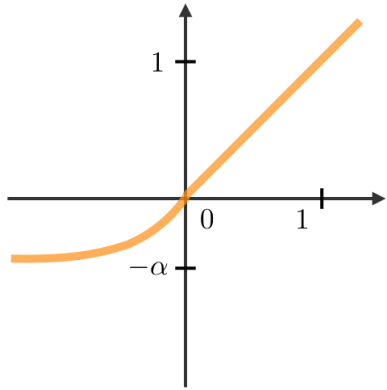
$$R_k = 1 + \sum_{j=1}^k (F_j - 1) \prod_{i=0}^{j-1} S_i$$

Dans l'exemple ci-dessous, on a $F_1 = F_2 = 3$ et $S_1 = S_2 = 1$, ce qui donne $R_2 = 1 + 2 \cdot 1 + 2 \cdot 1 = 5$.



<https://stanford.edu/~shervine/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#activation-function>
Fonctions d'activation communément utilisées

□ **Unité linéaire rectifiée** — La couche d'unité linéaire rectifiée (en anglais *rectified linear unit layer*) (ReLU) est une fonction d'activation g qui est utilisée sur tous les éléments du volume. Elle a pour but d'introduire des complexités non-linéaires au réseau. Ses variantes sont récapitulées dans le tableau suivant :




ReLU	Leaky ReLU	ELU
$g(z) = \max(0, z)$	$g(z) = \max(\epsilon z, z)$ with $\epsilon \ll 1$	$g(z) = \max(\alpha(e^z - 1), z)$ with $\alpha \ll 1$
		
<ul style="list-style-type: none"> • Complexités non-linéaires interprétables d'un point de vue biologique 	<ul style="list-style-type: none"> • Répond au problème de <i>dying ReLU</i> 	<ul style="list-style-type: none"> • Dérivable partout

□ **Softmax** — L'étape softmax peut être vue comme une généralisation de la fonction logistique qui prend comme argument un vecteur de scores $x \in \mathbb{R}^n$ et qui renvoie un vecteur de probabilités $p \in \mathbb{R}^n$ à travers une fonction softmax à la fin de l'architecture. Elle est définie de la manière suivante :

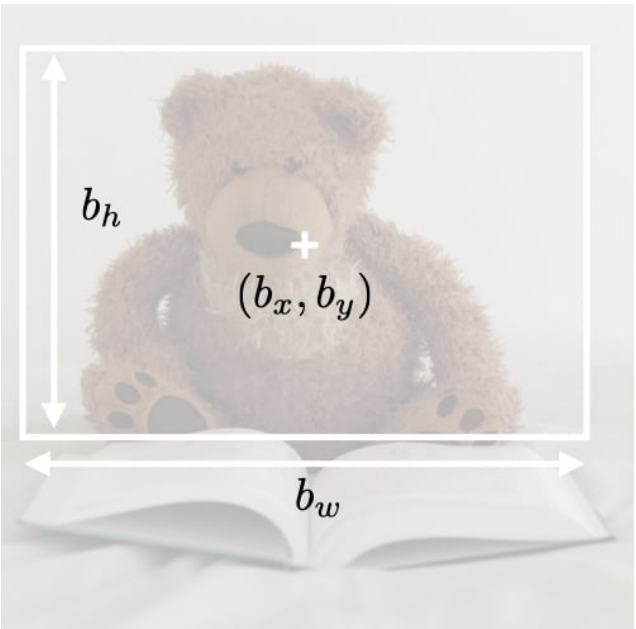
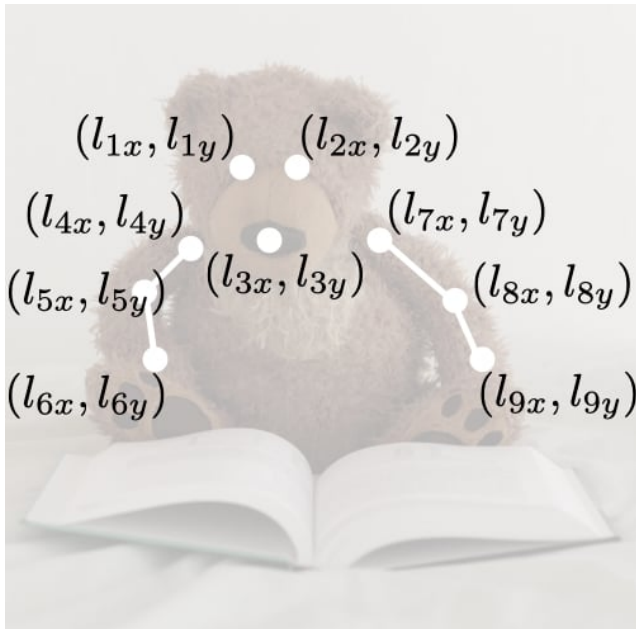
$$p = \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \quad \text{où} \quad p_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

(<https://stanford.edu/~shervine/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#object-detection>)
Détection d'objet

□ **Types de modèles** — Il y a 3 principaux types d'algorithme de reconnaissance d'objet, pour lesquels la nature de ce qui est prédit est différent. Ils sont décrits dans la table ci-dessous :

Classification d'image	Classification avec localisation	Détection
<div>Ours en peluche</div> 	<div>Ours en peluche</div> 	<div>Ours en peluche</div> 
<ul style="list-style-type: none">• Classifie une image• Prédit la probabilité d'un objet	<ul style="list-style-type: none">• Détecte un objet dans une image• Prédit la probabilité de présence d'un objet et où il est situé	<ul style="list-style-type: none">• Peut détecter plusieurs objets dans une image• Prédit les probabilités de présence des objets et où ils sont situés
CNN traditionnel	YOLO simplifié, R-CNN	YOLO, R-CNN

☐ **Détection** — Dans le contexte de la détection d'objet, des méthodes différentes sont utilisées selon si l'on veut juste localiser l'objet ou alors détecter une forme plus complexe dans l'image. Les deux méthodes principales sont résumées dans le tableau ci-dessous :

Détection de zone délimitante	Détection de forme complexe
<ul style="list-style-type: none">• Détecte la partie de l'image où l'objet est situé	<ul style="list-style-type: none">• Détecte la forme ou les caractéristiques d'un objet (e.g. yeux)• Plus granulaire
	

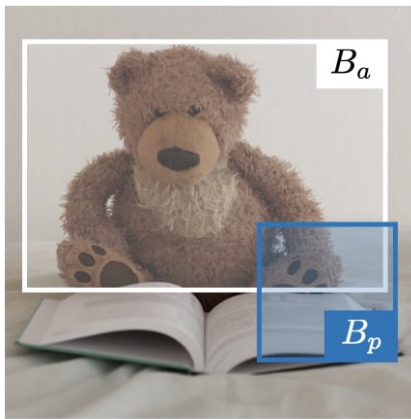
Zone de centre (b_x, b_y) , hauteur b_h et largeur b_w

Points de référence $(l_{1x}, l_{1y}), \dots, (l_{nx}, l_{ny})$

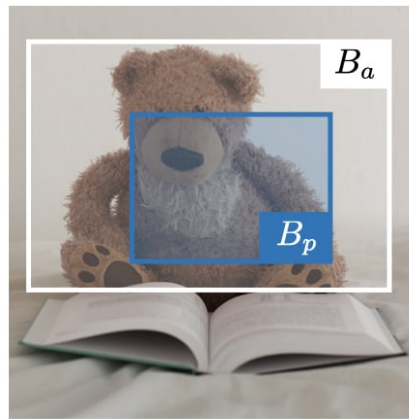
□ **Intersection sur Union** — Intersection sur Union (en anglais *Intersection over Union*), aussi appelé IoU, est une fonction qui quantifie à quel point la zone délimitante prédite B_p est correctement positionnée par rapport à la zone délimitante vraie B_a . Elle est définie de la manière suivante :

$$\text{IoU}(B_p, B_a) = \frac{B_p \cap B_a}{B_p \cup B_a}$$

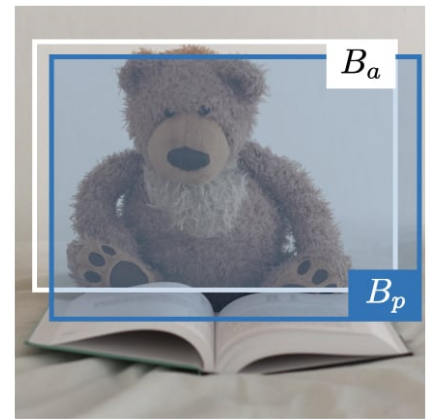
Remarque : on a toujours $\text{IoU} \in [0, 1]$. Par convention, la prédiction B_p d'une zone délimitante est considérée comme étant satisfaisante si l'on a $\text{IoU}(B_p, B_a) \geq 0.5$.



$$\text{IoU}(B_p, B_a) = 0.1$$



$$\text{IoU}(B_p, B_a) = 0.5$$



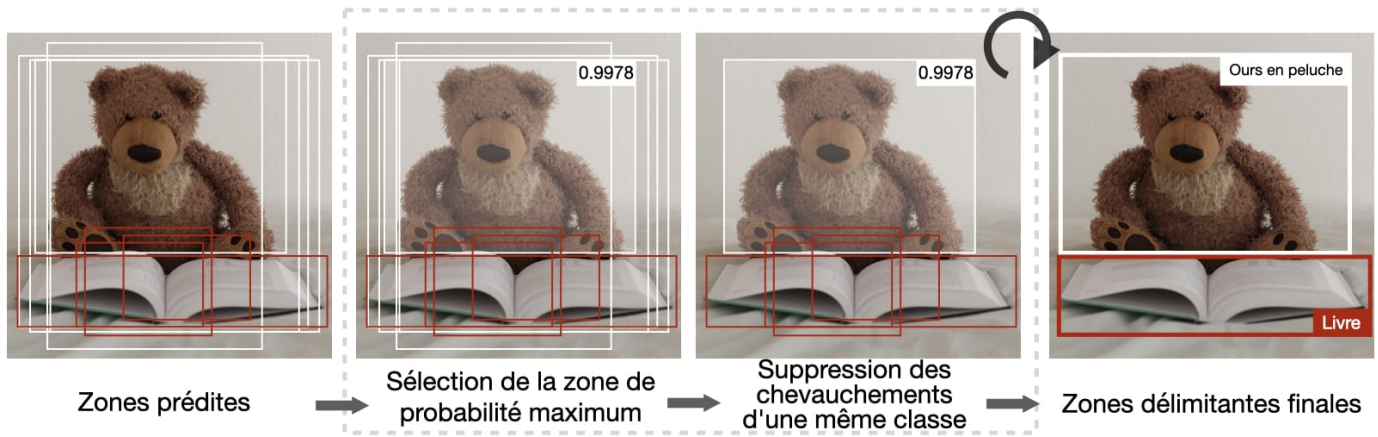
$$\text{IoU}(B_p, B_a) = 0.9$$

□ **Zone d'accroche** — La technique des zones d'accroche (en anglais *anchor boxing*) sert à prédire des zones délimitantes qui se chevauchent. En pratique, on permet au réseau de prédire plus d'une zone délimitante simultanément, où chaque zone prédite doit respecter une forme géométrique particulière. Par exemple, la première prédiction peut potentiellement être une zone rectangulaire d'une forme donnée, tandis qu'une seconde prédiction doit être une zone rectangulaire d'une autre forme.

□ **Suppression non-max** — La technique de suppression non-max (en anglais *non-max suppression*) a pour but d'enlever des zones délimitantes qui se chevauchent et qui prédisent un seul et même objet, en sélectionnant les zones les plus représentatives. Après avoir enlevé toutes les zones ayant une probabilité prédite de moins de 0.6, les étapes suivantes sont répétées pour éliminer les zones redondantes :

Pour une classe donnée,

- Étape 1 : Choisir la zone ayant la plus grande probabilité de prédiction.
- Étape 2 : Enlever toute zone ayant $\text{IoU} \geq 0.5$ avec la zone choisie précédemment.



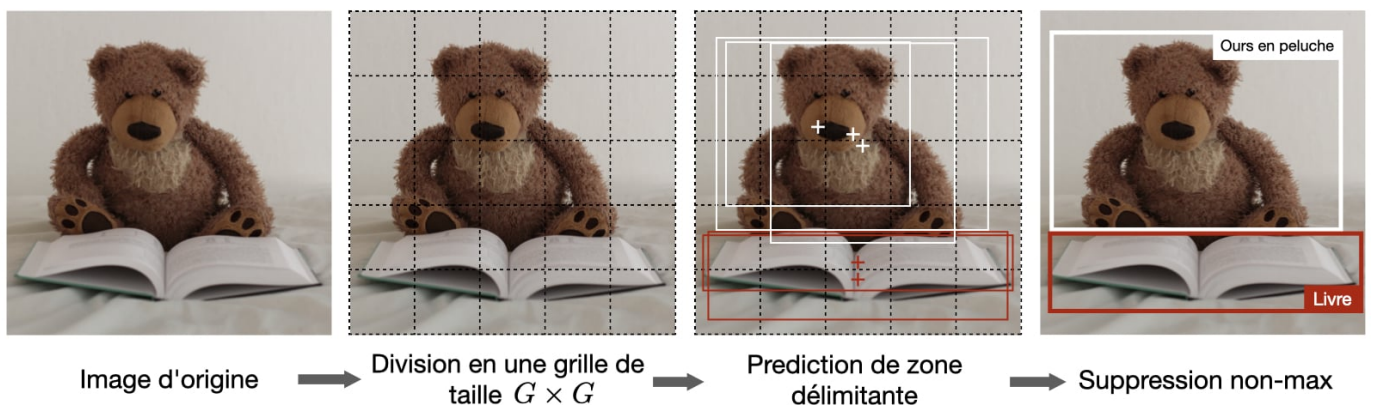
□ **YOLO** — L'algorithme You Only Look Once (YOLO) est un algorithme de détection d'objet qui fonctionne de la manière suivante :

- Étape 1 : Diviser l'image d'entrée en une grille de taille $G \times G$.
- Étape 2 : Pour chaque cellule, faire tourner un CNN qui prédit y de la forme suivante :

$$y = \underbrace{[p_c, b_x, b_y, b_h, b_w, c_1, c_2, \dots, c_p, \dots]}_{\text{répété } k \text{ fois}}^T \in \mathbb{R}^{G \times G \times k \times (5+p)}$$

où p_c est la probabilité de détecter un objet, b_x, b_y, b_h, b_w sont les propriétés de la zone délimitante détectée, c_1, \dots, c_p est une représentation binaire (en anglais *one-hot representation*) de l'une des p classes détectée, et k est le nombre de zones d'accroche.

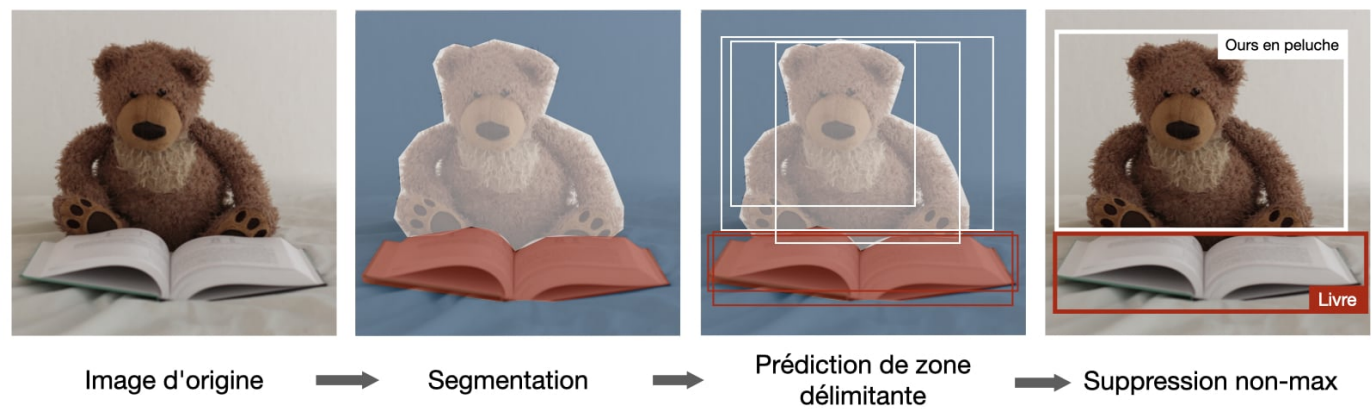
- Étape 3 : Faire tourner l'algorithme de suppression non-max pour enlever des doublons potentiels qui chevauchent des zones délimitantes.



Remarque : lorsque $p_c = 0$, le réseau ne détecte plus d'objet. Dans ce cas, les prédictions correspondantes b_x, \dots, c_p doivent être ignorées.

□ **R-CNN** — L'algorithme de région avec des réseaux de neurones convolutionnels (en anglais *Region with Convolutional Neural Networks*) (R-CNN) est un algorithme de détection d'objet qui segmente l'image d'entrée pour trouver des zones délimitantes pertinentes, puis fait tourner un

algorithme de détection pour trouver les objets les plus probables d'apparaître dans ces zones délimitantes.



Remarque : bien que l'algorithme original soit lent et coûteux en temps de calcul, de nouvelles architectures ont permis de faire tourner l'algorithme plus rapidement, tels que le Fast R-CNN et le Faster R-CNN.

<https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#face>

Vérification et reconnaissance de visage

❑ **Types de modèles** — Deux principaux types de modèle sont récapitulés dans le tableau ci-dessous :

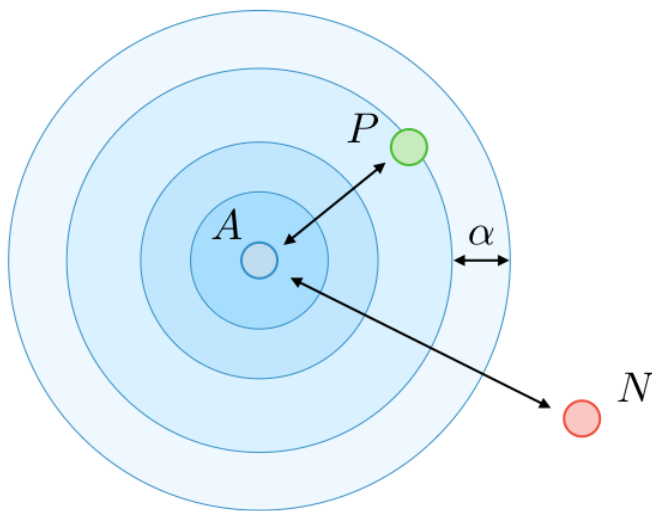
Vérification de visage	Reconnaissance de visage
<ul style="list-style-type: none">• Est-ce la bonne personne ?• Un à un	<ul style="list-style-type: none">• Est-ce une des K personnes dans la base de données ?• Un à plusieurs
<div>Requête<div></div><div></div></div> <div>✓</div> <div>✗</div> <div>Référence<div></div><div></div></div>	<div>Requête<div></div></div> <div>Base de données<div> </div></div>

□ **Apprentissage par coup** — L'apprentissage par coup (en anglais *One Shot Learning*) est un algorithme de vérification de visage qui utilise un training set de petite taille pour apprendre une fonction de similarité qui quantifie à quel point deux images données sont différentes. La fonction de similarité appliquée à deux images est souvent notée $d(\text{image 1}, \text{image 2})$.

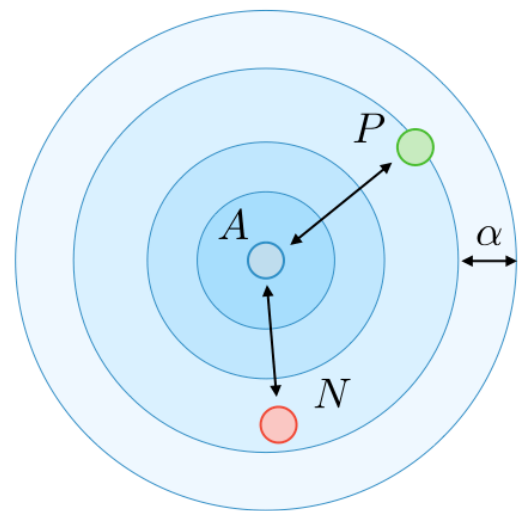
□ **Réseaux siamois** — Les réseaux siamois (en anglais *Siamese Networks*) ont pour but d'apprendre comment encoder des images pour quantifier le degré de différence de deux images données. Pour une image d'entrée donnée $x^{(i)}$, l'encodage de sortie est souvent notée $f(x^{(i)})$.

□ **Loss triple** — Le loss triple (en anglais *triplet loss*) ℓ est une fonction de loss calculée sur une représentation encodée d'un triplet d'images A (accroche), P (positif), et N (négatif). L'exemple d'accroche et l'exemple positif appartiennent à la même classe, tandis que l'exemple négatif appartient à une autre. En notant $\alpha \in \mathbb{R}^+$ le paramètre de marge, le loss est défini de la manière suivante :

$$\ell(A, P, N) = \max(d(A, P) - d(A, N) + \alpha, 0)$$



$$\ell(A, P, N) = 0$$



$$\ell(A, P, N) > 0$$

<https://stanford.edu/~shervine/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#style-transfer>

Transfert de style neuronal

□ **Motivation** — Le but du transfert de style neuronal (en anglais *neural style transfer*) est de générer une image G à partir d'un contenu C et d'un style S .

Contenu C

+

Style S

=

Image générée G

□ **Activation** — Dans une couche l donnée, l'activation est notée $a^{[l]}$ et est de dimensions $n_H \times n_w \times n_c$.

□ **Fonction de coût de contenu** — La fonction de coût de contenu (en anglais *content cost function*), notée $J_{\text{content}}(C, G)$, est utilisée pour quantifier à quel point l'image générée G diffère de l'image de contenu original C . Elle est définie de la manière suivante :

$$J_{\text{content}}(C, G) = \frac{1}{2} \|a^{[l](C)} - a^{[l](G)}\|^2$$

□ **Matrice de style** — La matrice de style (en anglais *style matrix*) $G^{[l]}$ d'une couche l donnée est une matrice de Gram dans laquelle chacun des éléments $G_{kk'}^{[l]}$ quantifie le degré de corrélation des canaux k and k' . Elle est définie en fonction des activations $a^{[l]}$ de la manière suivante :

$$G_{kk'}^{[l]} = \sum_{i=1}^{n_H} \sum_{j=1}^{n_w} a_{ijk}^{[l]} a_{ijk'}^{[l]}$$

Remarque : les matrices de style de l'image de style et de l'image générée sont notées $G^{[l](S)}$ and $G^{[l](G)}$ respectivement.

□ **Fonction de coût de style** — La fonction de coût de style (en anglais *style cost function*), notée $J_{\text{style}}(S, G)$, est utilisée pour quantifier à quel point l'image générée G diffère de l'image de style S . Elle est définie de la manière suivante :

$$J_{\text{style}}^{[l]}(S, G) = \frac{1}{(2n_H n_w n_c)^2} \|G^{[l](S)} - G^{[l](G)}\|_F^2 = \frac{1}{(2n_H n_w n_c)^2} \sum_{k,k'=1}^{n_c} \left(G_{kk'}^{[l](S)} - G_{kk'}^{[l](G)} \right)^2$$

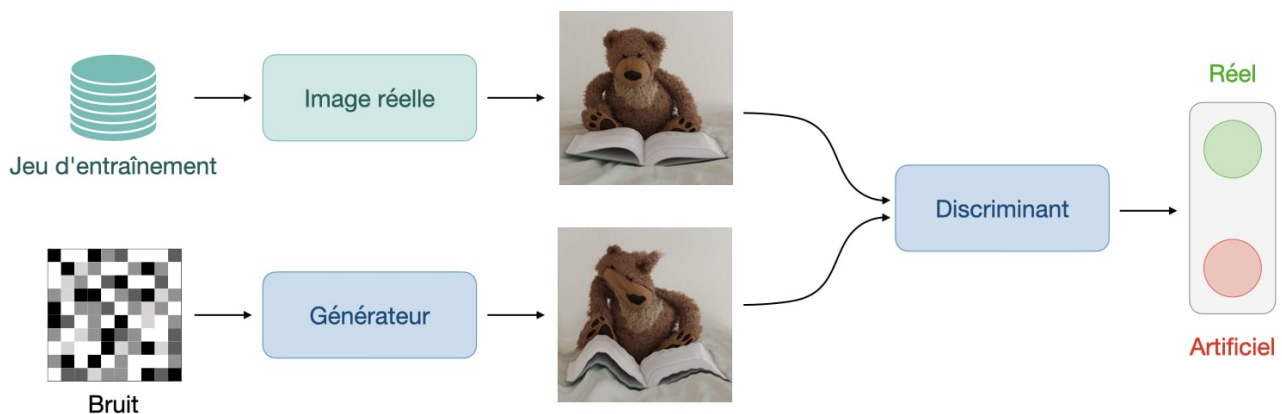
□ **Fonction de coût total** — La fonction de coût total (en anglais *overall cost function*) est définie comme étant une combinaison linéaire des fonctions de coût de contenu et de style, pondérées par les paramètres α, β , de la manière suivante :

$$J(G) = \alpha J_{\text{contenu}}(C, G) + \beta J_{\text{style}}(S, G)$$

Remarque : plus α est grand, plus le modèle privilégiera le contenu et plus β est grand, plus le modèle sera fidèle au style.

<https://stanford.edu/~shervine/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels#ct-architectures> Architectures utilisant des astuces de calcul

□ **Réseau antagoniste génératif** — Les réseaux antagonistes génératifs (en anglais *generative adversarial networks*), aussi connus sous le nom de GANs, sont composés d'un modèle génératif et d'un modèle discriminant, où le modèle génératif a pour but de générer des prédictions aussi réalistes que possibles, qui seront ensuite envoyées dans un modèle discriminant qui aura pour but de différencier une image générée d'une image réelle.



Remarque : les GANs sont utilisées dans des applications pouvant aller de la génération de musique au traitement de texte vers image.

□ **ResNet** — L'architecture du réseau résiduel (en anglais *Residual Network*), aussi appelé ResNet, utilise des blocs résiduels avec un nombre élevé de couches et a pour but de réduire l'erreur de training. Le bloc résiduel est caractérisé par l'équation suivante :

$$a^{[l+2]} = g(a^{[l]} + z^{[l+2]})$$

□ **Inception Network** — Cette architecture utilise des modules d'*inception* et a pour but de tester toute sorte de configuration de convolution pour améliorer sa performance en diversifiant ses attributs. En particulier, elle utilise l'astuce de la convolution 1×1 pour limiter sa complexité de calcul.



(<https://twitter.com/shervinea>)



(<https://linkedin.com/in/shervineamidi>)



(<https://github.com/shervinea>)



(<https://scholar.google.com/citations?user=nMnMTm8AAAAJ>)

