Fondamentaux du Deep learning

Louis Thiry (INRIA Rennes)

louis.thiry@outlook.fr

Formation

Take home message:

- Une méthodologie précise pour approximer des fonctions
- Chaque point est crucial: attention requise !
- Optimisation: a base de gradient.
- Autodiff: plus besoin de calculer le gradient.
- Linear regression can work suprisingly good !

Machine Learning

- "*Les algorithmes de Machine Learning apprennent de manière autonome à effectuer une tâche ou à réaliser des prédictions à partir de données et améliorent leurs performances au fil du temps. Une fois entraîné, l'algorithme pourra retrouver les patterns dans de nouvelles données.*" Datascientest.com

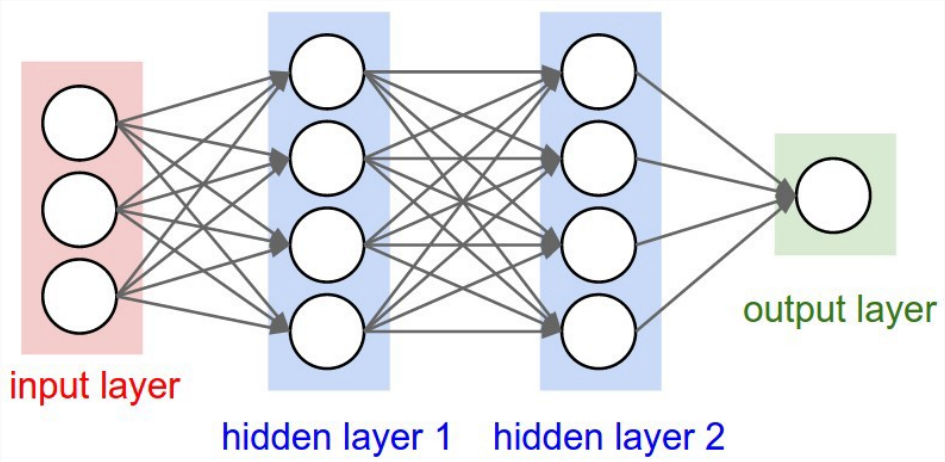
- " Champ d'étude de l'intelligence artificielle qui se fonde sur des approches mathématiques et statistiques pour donner aux ordinateurs la capacité d' "apprendre" à partir de données, c'est-à-dire d'améliorer leurs performances à résoudre des tâches sans être explicitement programmés pour chacune. " Wikipedia.com

$\rightarrow$ Définitions imprécises et souvent anthropomorphiques.

Les réseaux de neurones

Les réseaux de neurones

- Un **réseau de neurone** est une fonction paramétrique. On applique une succession d'opérations linéaire suivi d'une fonction linéaire a l'entrée pour calculer la sortie.
- Une **architecture** de réseau de neurone définit une classe de fonction $\mathcal{F} = \{f_\theta, \theta \in \mathbb{R}^d\}$.
- Habituellement représentée par un graphe, *rappelle* la structure des premières couches du cortex visuel



Les grandes familles de réseaux de neurones.

- Réseaux de neurones fully-connected
- Réseaux de neurones convolutifs pour les images.
- LSTM (Long Time Short Time Memory), (RNN) Recurrent Neural Network, Transformers pour les données sequentielles: suites, series temporelles...
- AutoEncodeur, Reseaux generatifs adversariaux (GANs) pour la génération de données: images, sons, graphes....
- Et bien d'autres...

L'intérêt des réseaux de neurones?

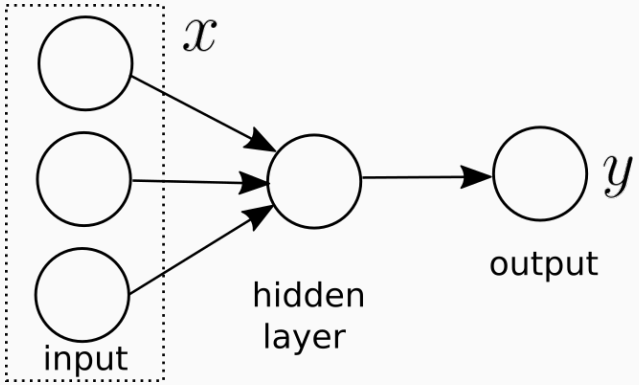
- Extrêmement efficaces en traitement du signal $\rightarrow$ nombreuse applications industrielles
- Prise en main facilitée par des bibliothèques auto-differentiables open-source: PyTorch, Keras, JAX...
- Nécessitent essentiellement des données bien structurées.
- Versatiles et re-utilisables.
- Une communauté habituée a publier et partager des codes de qualité.
- Des avance plus en plus de nouvelles applications: littérature riche et passionnante à suivre.

Les réseaux de neurones fully-connected

Une couche cachée, un neurone.

$$f : \mathbb{R}^d \to \mathbb{R}$$
$$x \mapsto c \times \sigma(\langle w, x \rangle + b) + d$$

- $w \in \mathbb{R}^d, (b, c, d) \in \mathbb{R}$: poids du réseau ($b, d$ biais).
- $\sigma : \mathbb{R} \to \mathbb{R}$ est une non-linéarité/fonction d'activation. ex:
  - ReLU (Rectified Linear Unit): $\sigma(x) = \max(0, x)$.
  - Sigmoid $\sigma(x) = 1/(1 + exp(-x))$.
- $d = 3$, réseau à une couche cachée d'un neurone.

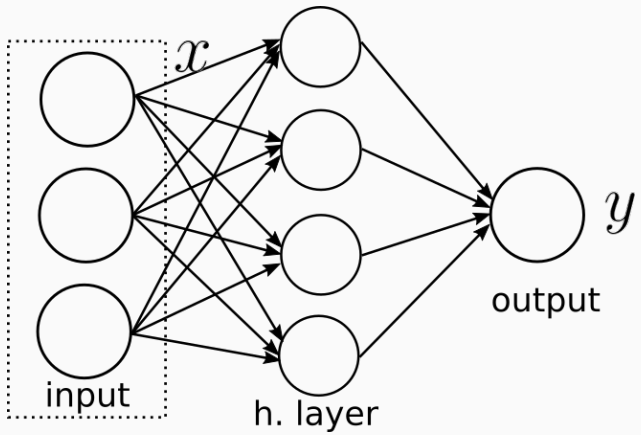


Les réseaux de neurones fully-connected

Une couche cachée, plusieurs neurones.

$$f : \mathbb{R}^d \to \mathbb{R}$$
$$x \mapsto \langle c, \sigma(Mx + b) \rangle + d$$

- $M \in \mathbb{R}^{d \times h}, b, c \in \mathbb{R}^h, d \in \mathbb{R}$: poids du réseau.
- Non-linéarité $\sigma : \mathbb{R} \to \mathbb{R}$ appliquée *element-wise*.
- $d = 3$, réseau à une couche cachée de 4 neurones.

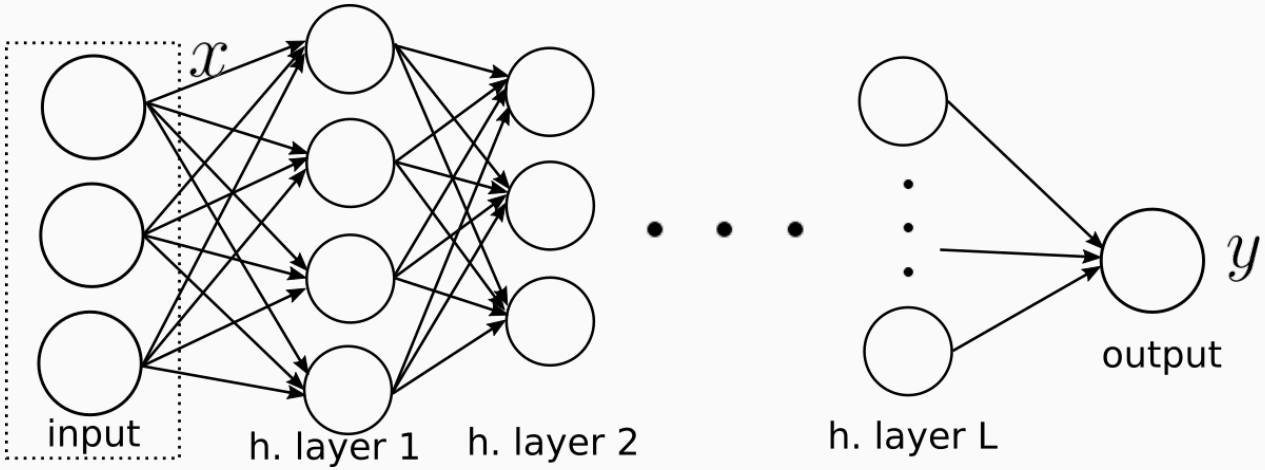


Les réseaux de neurones fully-connected

Plusieurs couches cachées a plusieurs neurones.

$$f : \mathbb{R}^d \to \mathbb{R}$$
$$x \mapsto \langle c, \sigma\left(M_L ... \sigma(M_1 x + b_1) + b_l\right)\rangle + d$$

- Poids: $M_1 \in \mathbb{R}^{d \times h_1}, M_l \in \mathbb{R}^{h_{l-1} \times h_l}\, b_l \in \mathbb{R}^{h_l}, d \in \mathbb{R}$.
- *Deep*: profond au sens nombre de couches.
- $d = 3$, réseau à L couches cachées de h_l neurones.



Les réseaux de neurones

Soit $\Phi_\theta(x)$ la dernière couche cachée du réseau de neurones.

- La prédiction est

$$f(x) = \langle c, \Phi_\theta(x) \rangle + b$$

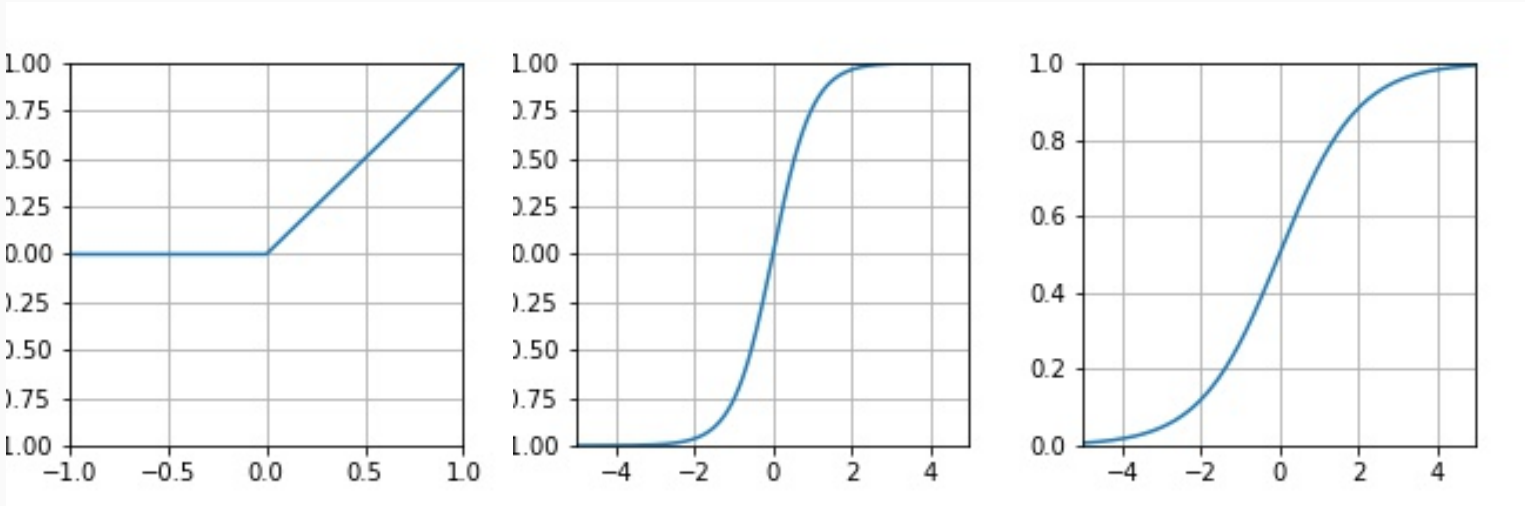
- C'est une régression linéaire en prenant $\Phi_\theta(x)$ en entrée
- Avec les reseau de neurones, on optimise a la fois la regression lineaire **et** la représentation
- **Feature engineering automatique**

Non-linéarités

Une non-linéarité après une linéarité permet

- Soit de normaliser la valeur des features (= feature map), e.g. dans $[0, 1]$ (sigmoid) ou $[-1, 1]$ (tanh).

- Soit de donner la possibilité de discriminer l'information importante. D'où le nom de fonctions d'activation.
  Non-linéarités = éliminer de l'information.



(a) ReLU **(b)** tanh **(c)** sigmoid

Théorie d'approximation

Théorème d'approximation universelle: réseau à une couche cachée contenant un nombre fini de neurones peut approximer des fonctions continues sur des sous-ensembles compacts de $\mathbb{R}^D$.

- Premier papier, non-linéarité sigmoid : Cybenko, 1989.
- Autres non-linearites Hornik, 1991.
- Toute non-linéarité non-polynomiale Leshno et al., 1993.

Théorie d'approximation

Exercice:

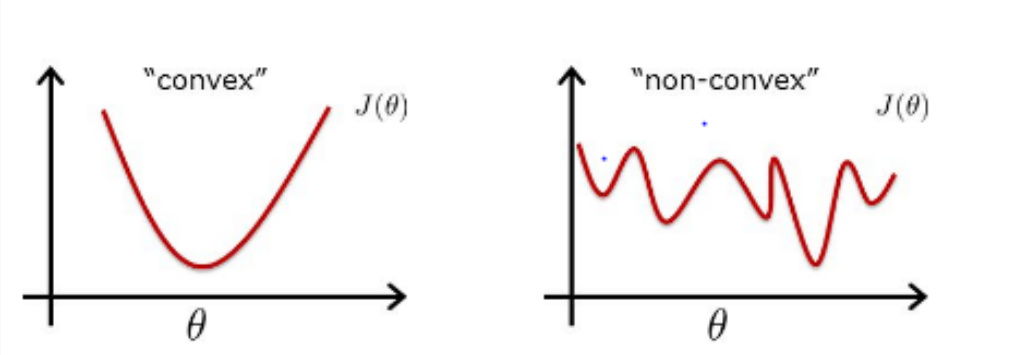
- Soit $f : [0, 1]^d \mapsto \mathbb{R}$ continue.
- Réseau de neurones a une couche cachée sans biais
- Non-linéarité: cos et sin.

Connaissez vous un théorème d'approximation pour un tel cas ? Comment s'appelle-t-il ? Est-ce utile en pratique pour approximer n'importe quelle fonction ?

$\rightarrow$ résultats intéressants mais a l'utilité pratique limitée

Optimisation

- Régression linéaire: forme close $\rightarrow$ optimisation immédiate
- Régression logistique: fonction convexe $\rightarrow$ garantie de convergence de la descente de gradient.
- Réseaux de neurones: ni forme close, ni garantie de convergence: optimisation compliquée !



Optimisation des réseaux de neurones

- Méthode: descente de gradient améliorée: accélération, quasi-Newton, pas adaptatif...
- Il y a des hyperparamètres liés aux méthodes d'optimisation que l'on utilise $\rightarrow$ peuvent être très sensibles.
- Un réseau entraîné: on a trouvé les bons paramètres (sauf éventuellement pour la dernière couche).
- Vocabulaire lié à l'optimisation: *learning rate*, *backward pass*, *epoch*, *batch-size*...
- L'optimisation **convexe en grande dimension** n'est déjà pas facile.
- Réseaux de neurones = **grande dimension** + **non-convexité**.

Optimisation des réseaux de neurones

On optimise un critère $\mathcal{L}$ moyen

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(f_\theta(x^i), y^i). \tag{1}$$

- θ: paramètre a optimiser.
- Résoudre (1) est maintenant un problème difficile. Il faut par exemple choisir le nombre d'itérations dans la descente de gradient.
- N est très grand: $10^4 - 10^9$. Calculer

$$\nabla_\theta \mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta L(f_\theta(x^i), y^i)$$

à chaque itération de la descente de gradient est **prohibitif** !

Descente de Gradient Stochastique

On tire aléatoirement $B << n$ $(i_1, \cdots, i_B)$ entre 0 et n et

$$\theta_{t+1} = \theta_t - \lambda_t \frac{1}{B} \sum_{j=1}^{B} \nabla_\theta L(f_{\theta_t}(x^{i_j}), y^{i_j}) \ . \qquad \text{(Stochastic GD)}$$

- B = batch size.
- **Epoch** = après n/B itérations de SGD.
- Stochastic = $\frac{1}{B} \sum_{j=1}^{B} \nabla L(f_{\theta_t}(x^{i_j}), y^{i_j})$ est une estimation **stochastique** (= aléatoire) de $\nabla F(\theta)$.

Quelques définitions

- **Epoch:** visiter une fois toutes les éléments de la base de donnée.
- **Forward pass:** prendre un data point x^i, un réseau de neurone $f()$ et évaluer $f(x^i)$ (en parallèle).
- **Backward pass:** un réseau de neurone $f()$ est en fait parametrisé par $f_w()$. Une *backward pass* c'est calculer

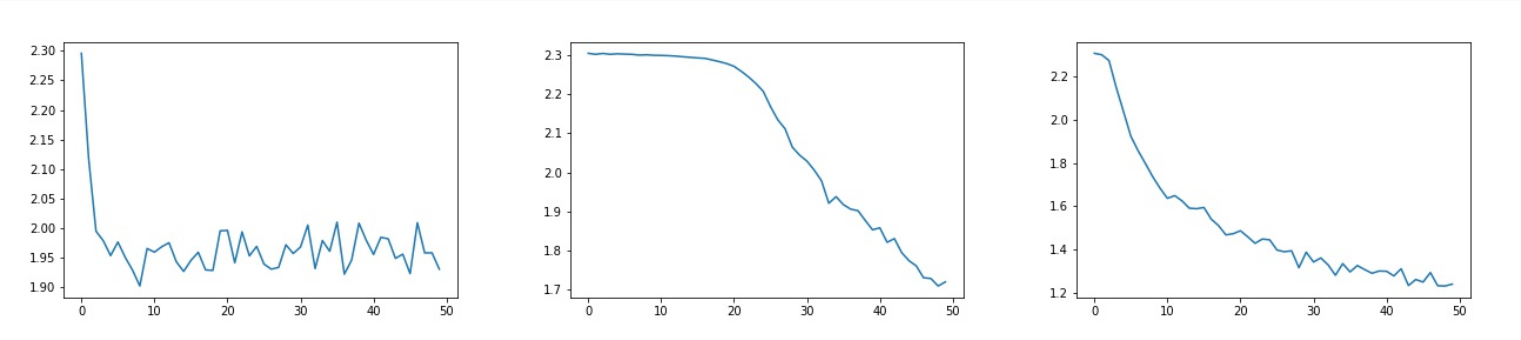
$$\nabla_w L(f_w(x^i), y^i).$$

Avec l'autodiff, c'est magique, il n'y a pas besoin de réfléchir, l'ordinateur le fait pour nous.

Métaphore pour bien se souvenir

- jeu de donnée = livre.
- réseau de neurone = écolier.
- apprentissage = lire le livre plusieurs fois.
- nombre de pages lues à la fois = batch-size.
- nombre de fois le livre a été lu entièrement = nbr epochs.
- learning-rate = attention accordée à chaque page (quitte à oublier ce qui a été appris avant).

Le choix du learning rate?



(d) lr grand **(e)** lr petit **(f)** lr bon

- Décroitre le learning rate au fur et à mesure des *epochs*: *learning rate schedule*.
- Learning rate adaptifs aux données du batch. Si les images sont rares, alors on prend un grand lr.
- Les algo **Adagrad, RMSprop, Adam** proposent des heuristiques utilisées partout.

Comprendre les paramètres

- "batch-size" : le nombre de chapitre que le modèle lit à chaque itération.
- "epoch" : le nombre de fois où l'on relit tous les livres.
- "lr": le learning rate = de combien l'alpiniste suit le gradient.

Mise en pratique: notebook sur les réseaux de neurones.

La Batch Norm

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift Ioffe et al., 2015

- Inspiration: centrer-réduire les données pour accélérer l'optimisation

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}}$$

- Ajout d'un epsilon de sécurité
- Ajout de deux paramètres optimisables γ et β

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

- Accélération peut être spectaculaire + effet de régularisation

Le Dropout

Dropout: a simple way to prevent neural networks from overfitting, Srivastava et al., 2014

- Inspiration: empêcher la "sur-spécialisation" des poids du réseau pour éviter le sur-apprentissage
- Pendant l'entraînement, les poids sont mis a zéro avec une proba p, donc inchangés avec proba $1 - p$.
- En utilisation normale, les poids sont inchangés.
- Effet significatif, assez sensible au paramètre p.

Les Fondamentaux du Deep Learning

- Machine learning:
  - Importance des données
  - Classe de fonction
  - Fonction de coût
  - Optimisation
- **Abracadabra** : Différentiation automatique !
- Optimisation non-convexe en grand dimension: peu de théorie, beaucoup de *bonnes pratiques*.
- Réseaux de neurones: attention au sur-apprentissage.
- Choix délicat de nombreux hyper-paramètres

Deep learning pour la classification d'images

Un application historique

- Back-propagation pour reconnaissance de chiffres LeCun et al., 1989.
- Application industrielle: reconnaissance de code postaux LeCun et al., 1998
- Revolution AlexNet: Krizhevsky et al., 2012.
- Mini-revolution ResNet: He et al., 2016.
- Succès des lauréats du prix Turing Le Cun, Hinton, Bengio pour le *Deep Learning* LeCun et al., 2015.

Bases de données d'images

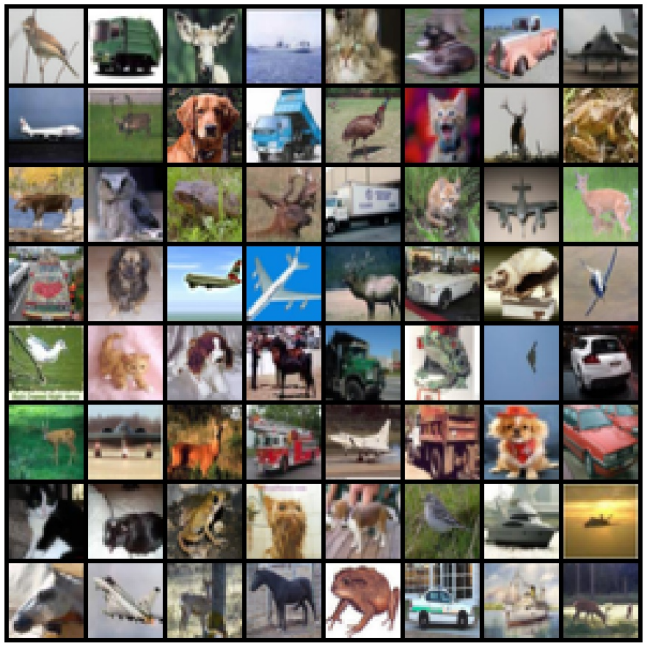
Nom	d	# dataset	nbr classes	year
MNIST	28×28	6×10^4	10	1996
CIFAR10	32×32	6×10^4	10	2009
CIFAR100	32×32	6×10^4	100	2009
ImageNet	256×256	14×10^6	2×10^4	2009

- Publication de bases de données intimement lie au développement du Deep Learning
- Facilement téléchargeables.
- Liste non exhaustive, voir ici.
- Possible biais, par ex. ImageNet wikipedia.

MNIST

- 10 classes de chiffres
- 60,000 images binaires 28×28.

CIFAR10



- 10 classes: chien, chat, voiture, camion...
- 50,000 images couleur 32×32.

ImageNet

Base de données ImageNet

- 1,3 Millions d'images couleurs.
- 1000 classes: 50 types de chiens, 3 de requins...
- Tailles variables, généralement redimensionnées 224×224

Réseaux convolutifs.

Analogie avec notre Vision

"The suggestion is that the function of the brain and nervous system and sense organs is in the main eliminative and not productive."

The Door of Perceptions, Aldous Huxley 1952

"Primate vision is an existence proof for the functionality of systems computer vision seek to develop."

Modeling visual attention via selective tuning, J.K. Tsotsos et al. 1994

Un réseau de convolution

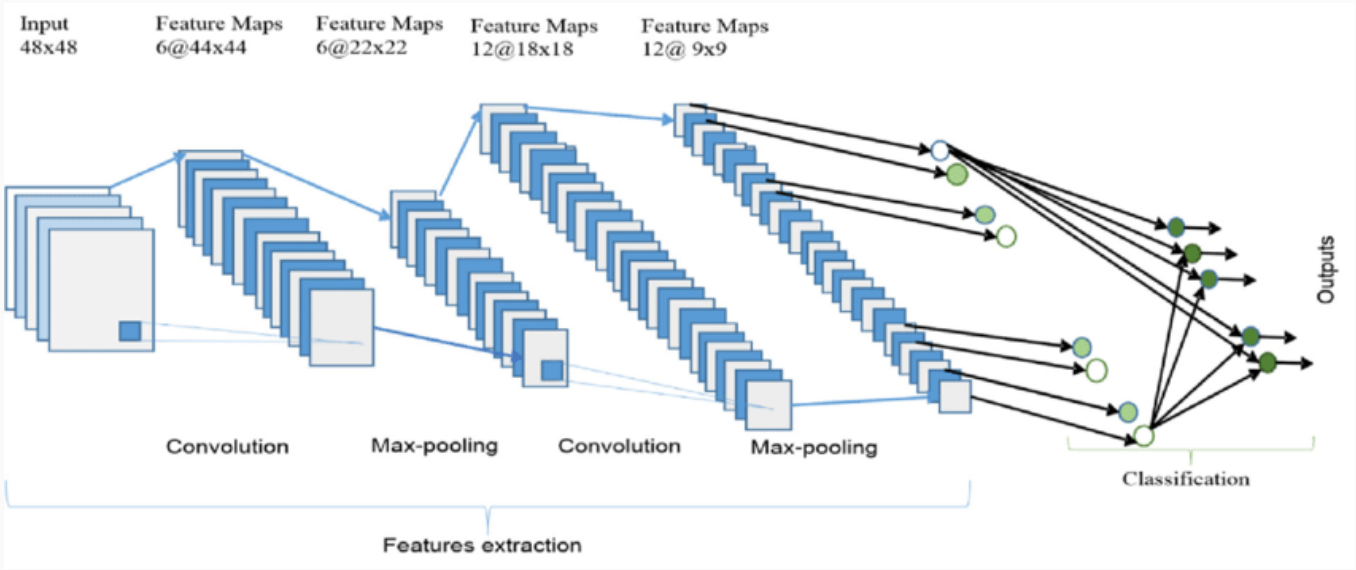


Figure 1: Réseau de neurones convolutionnel.

1. L'opération de convolution d'une **image** avec un **filtre** (appris) (= comme les paramètres d'une régression linéaire).
2. Convolution + non-linéarité.
3. Superposer et paralléliser ces opérations.
4. Ajouter d'autres opérations: pooling, batch-norm, etc..

C'est quoi une image?

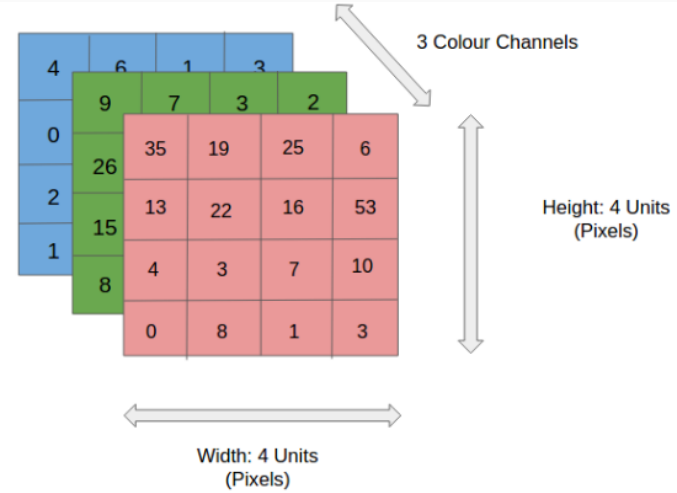


Figure 2: Une image RGB (mais d'autres espaces d'images) = on parle de **channel** pour chaque couleur. Chaque pixel est une valeur d'intensité = un entier dans $\{0, 255\}$ ou un nombre dans $[0, 1]$.

Une image = un tenseur de taille (H, W, C).

Plus tard on donnera au réseaux des tenseurs de taille (B, H, W, C) où B est la *batch-size*, i.e., **le nombre d'exemples en parallèle.**

La convolution.

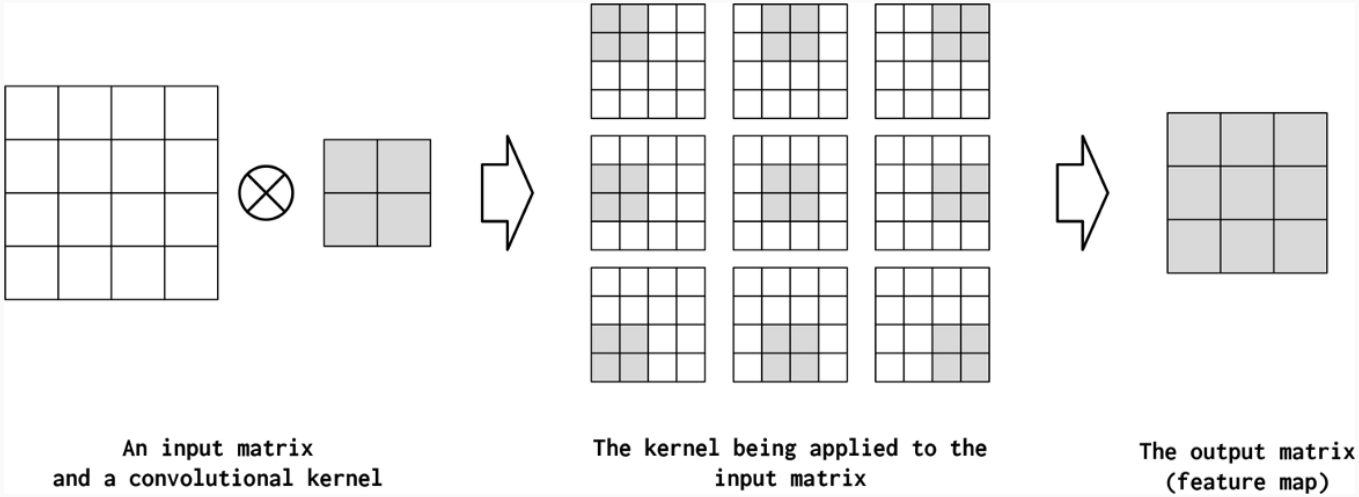


Figure 3: En gris c'est le **filtre (convolution kernel)** qui dépend de 4 paramètres. Ce sont eux que le réseau de neurone va apprendre/optimiser. Pour une image on applique la convolution séparément à chaque *channel* de couleur. Image prise ici

.

Convolution en image.

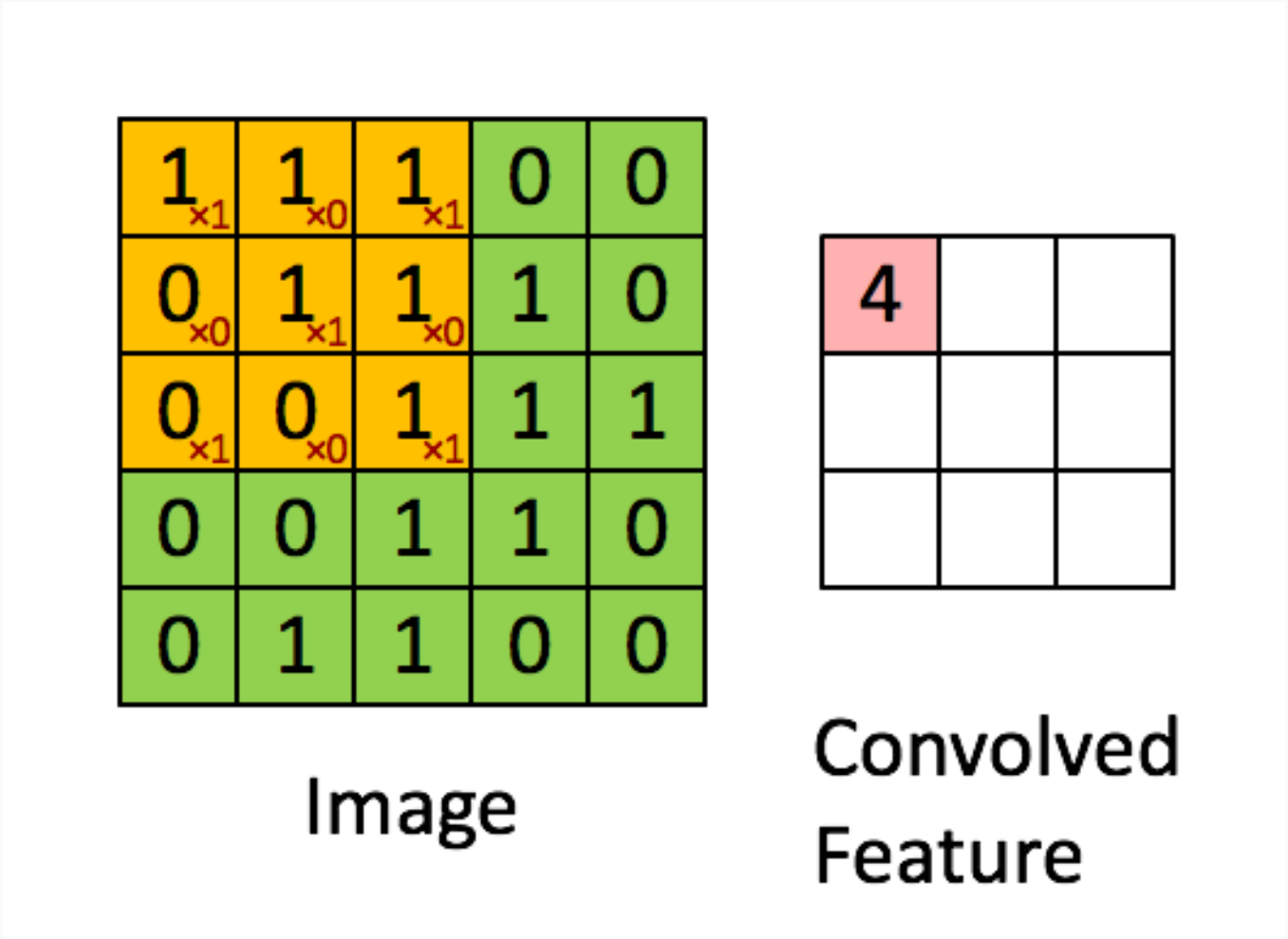


Figure 4: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

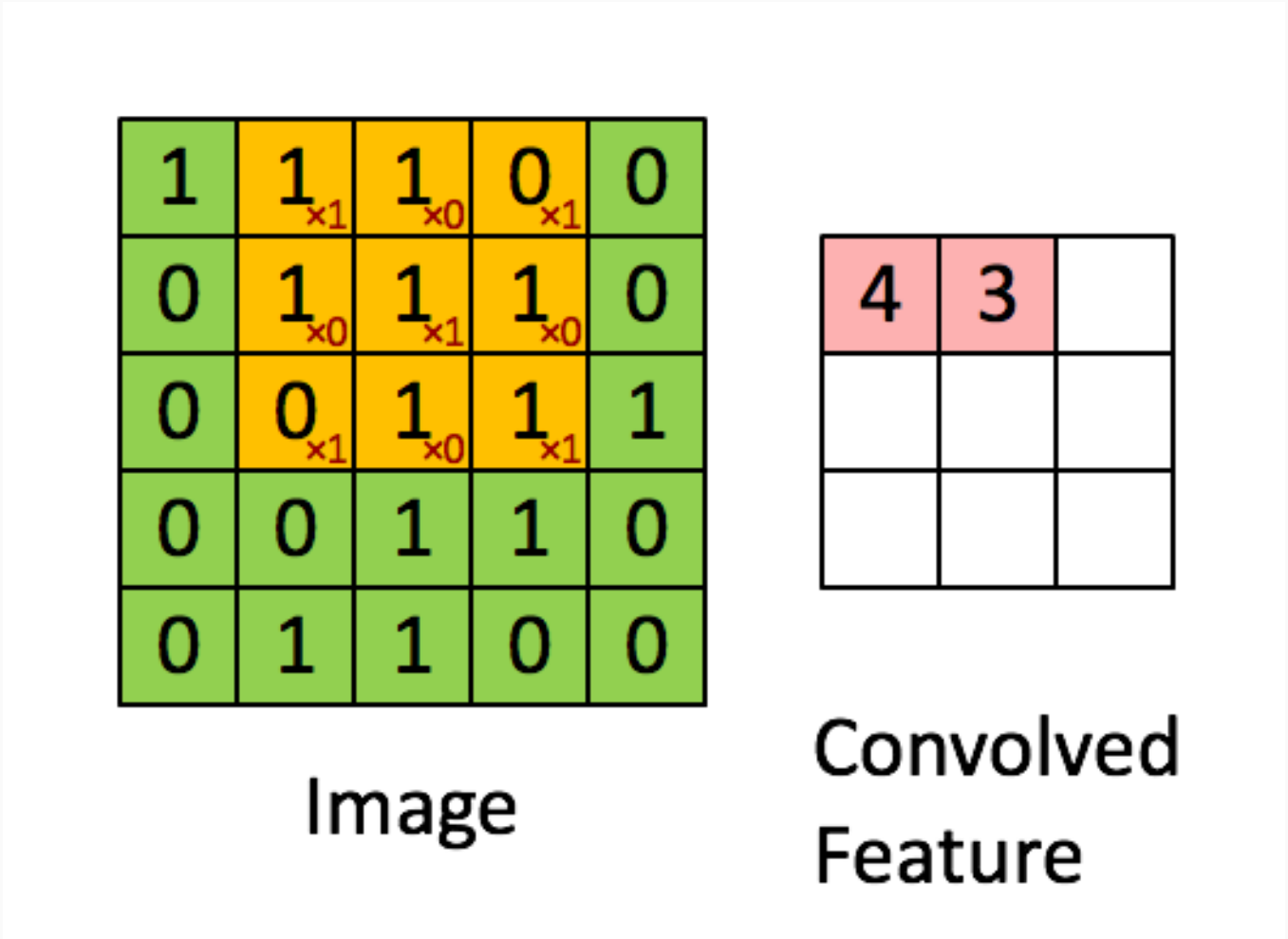


Figure 5: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

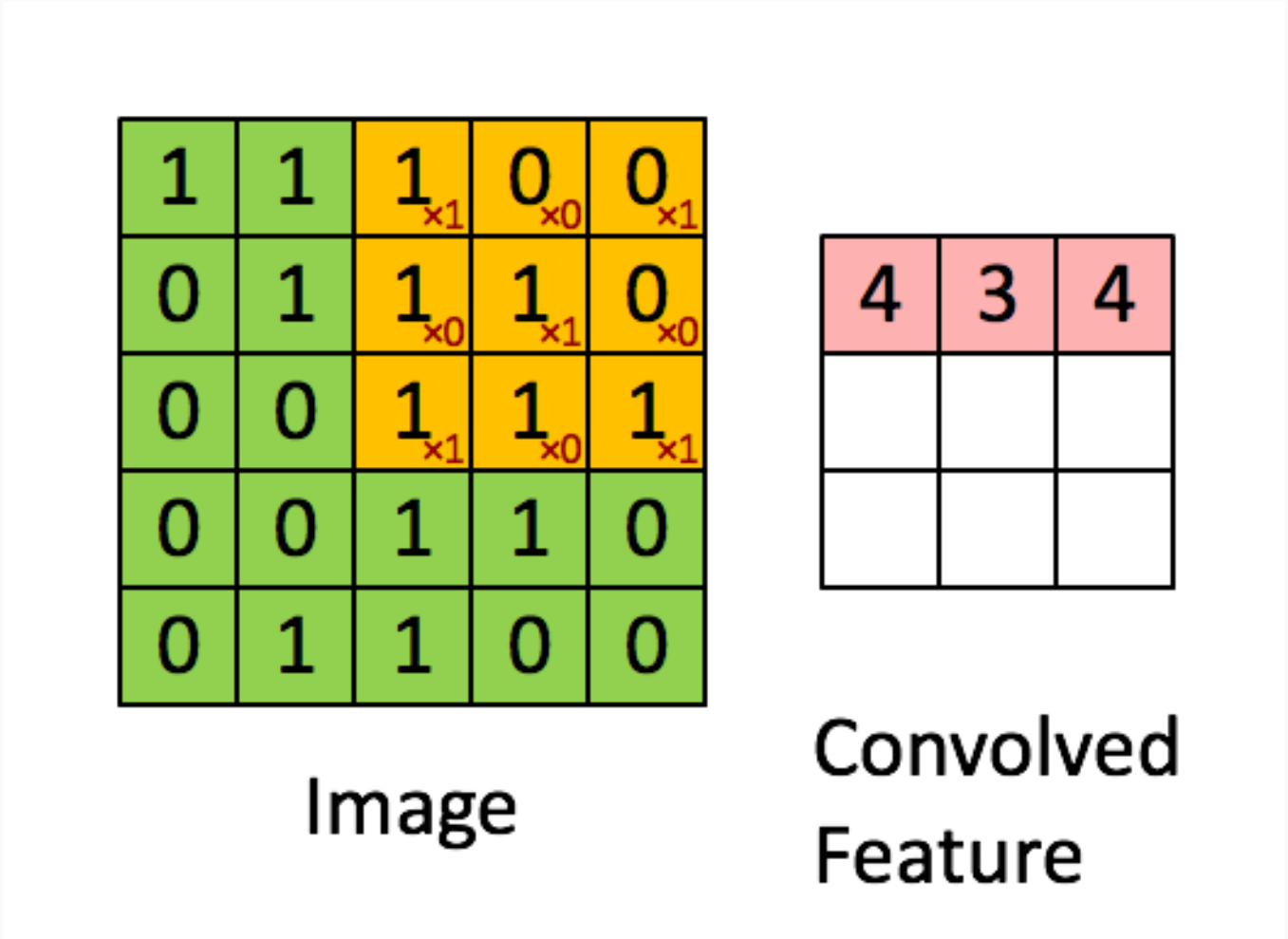


Figure 6: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

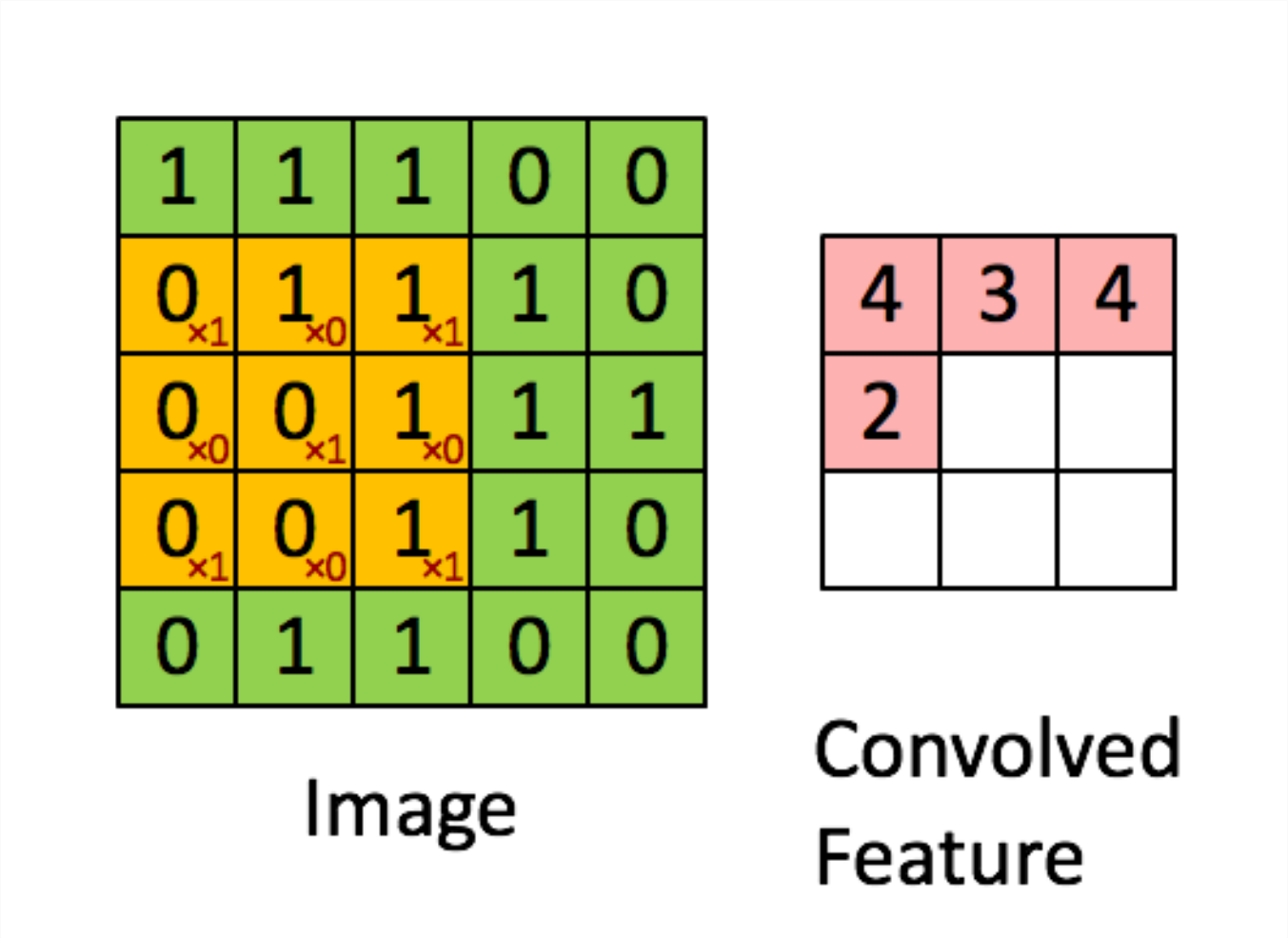


Figure 7: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

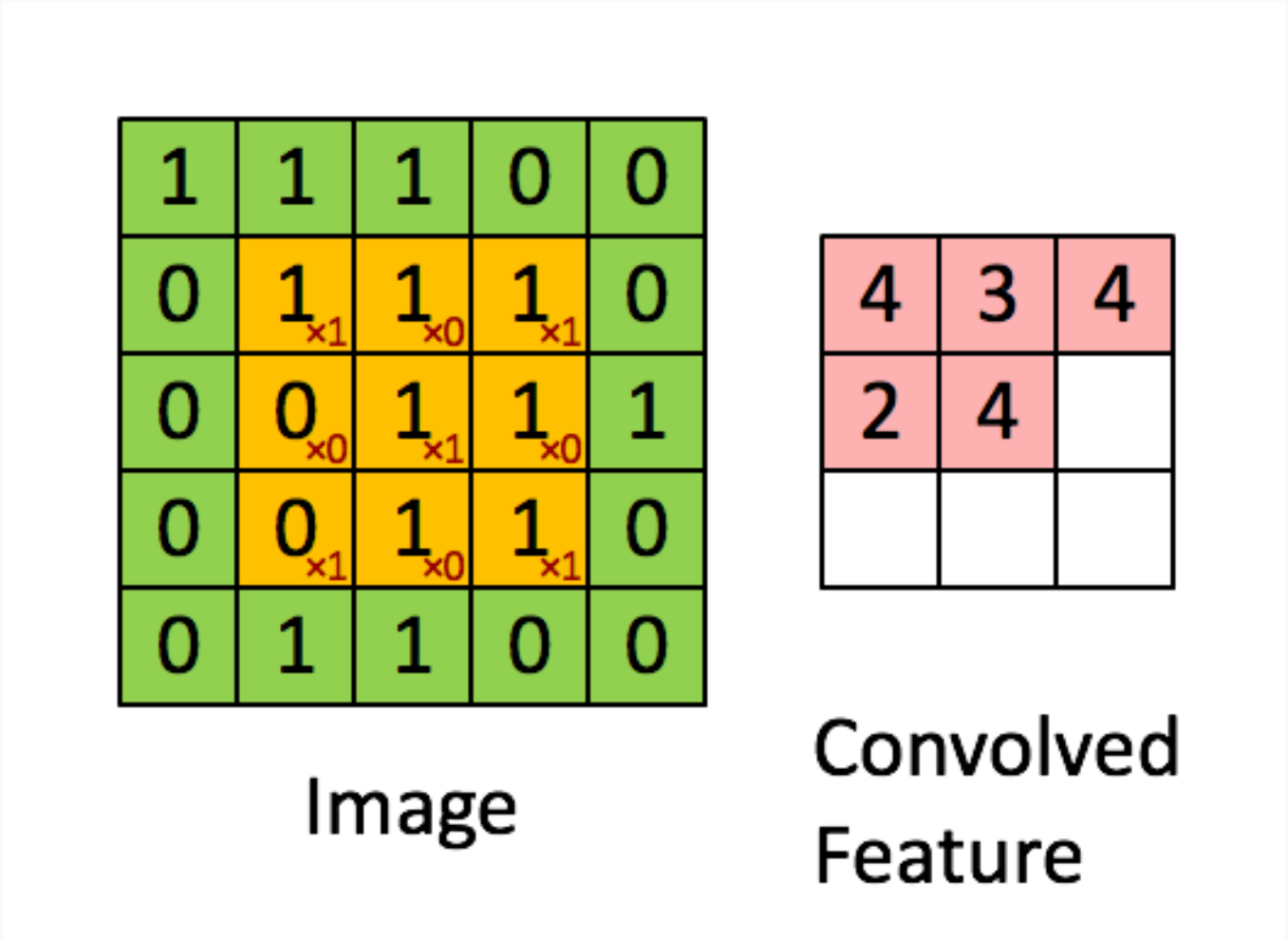


Figure 8: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

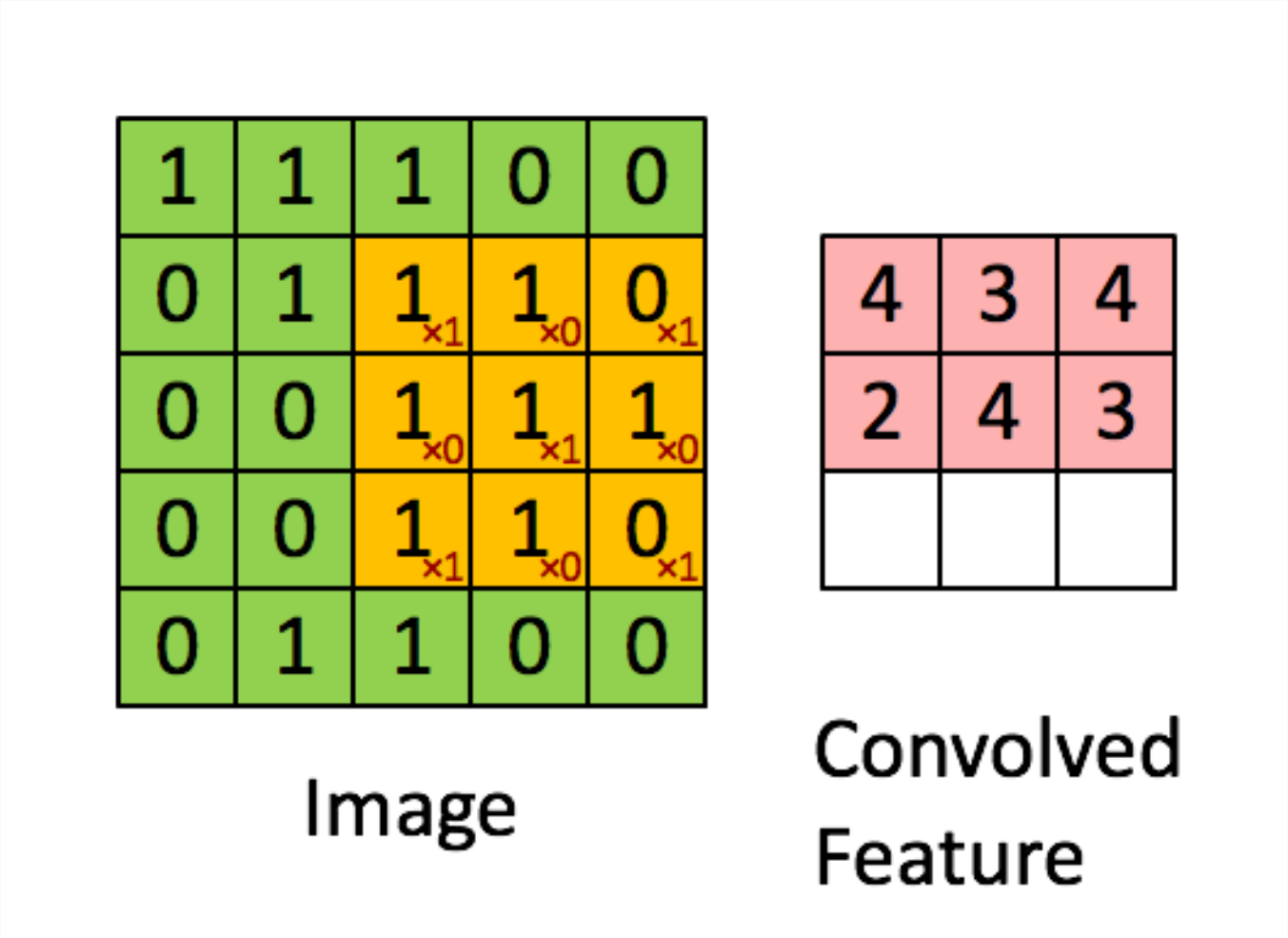


Figure 9: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

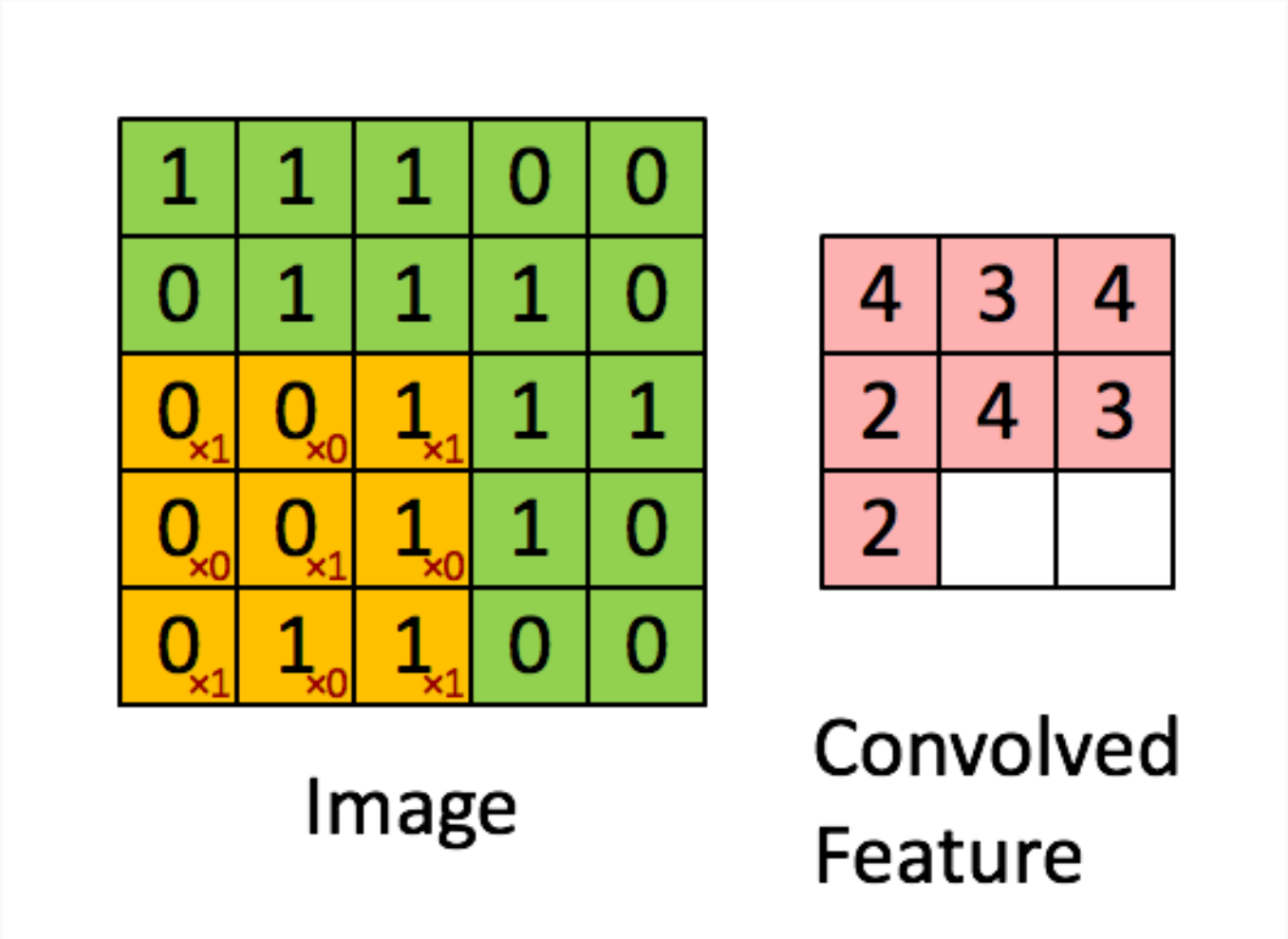


Figure 10: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

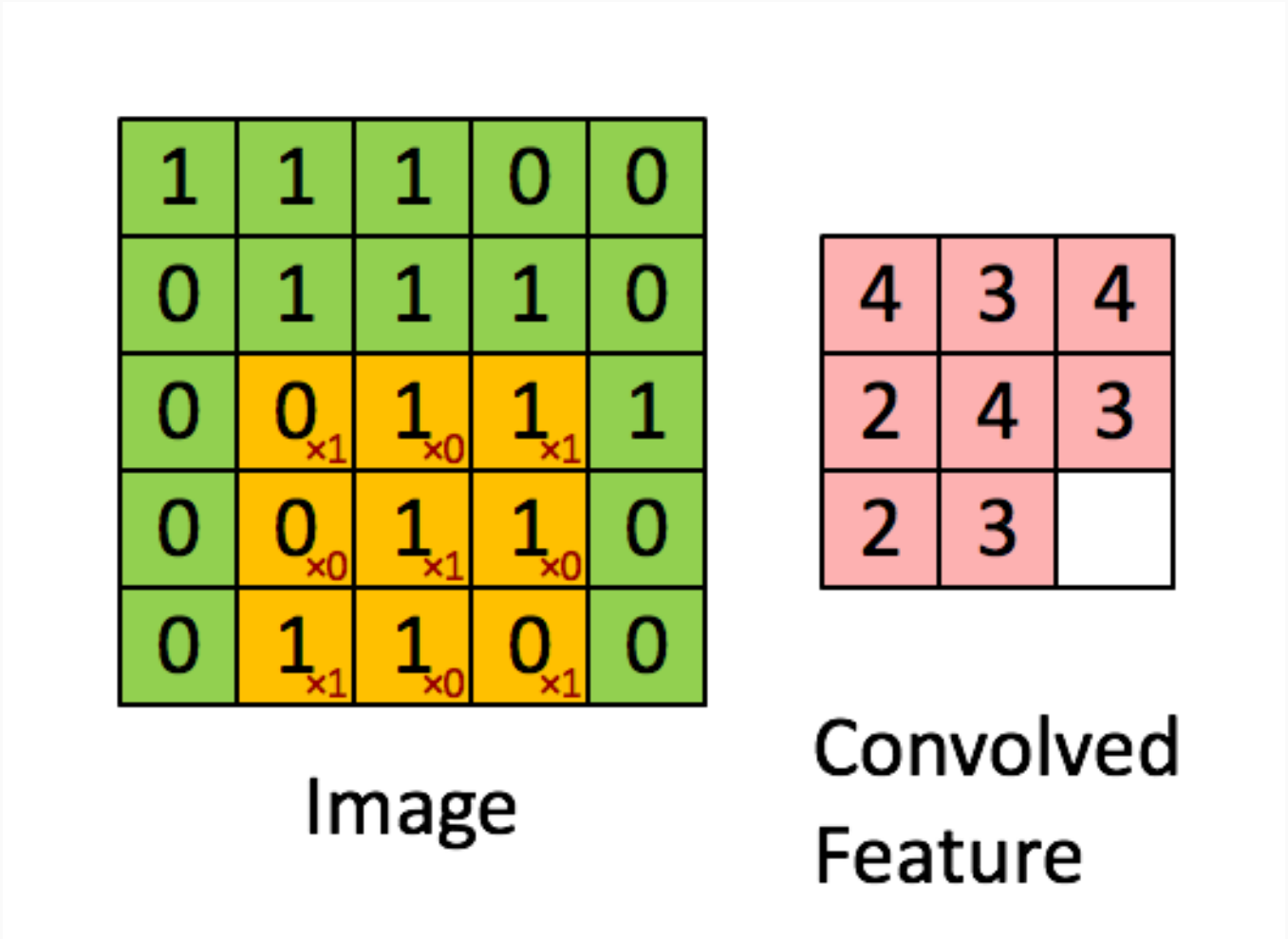


Figure 11: Image prise ici. De gauche à droite: image et **feature map**.

Convolution en image.

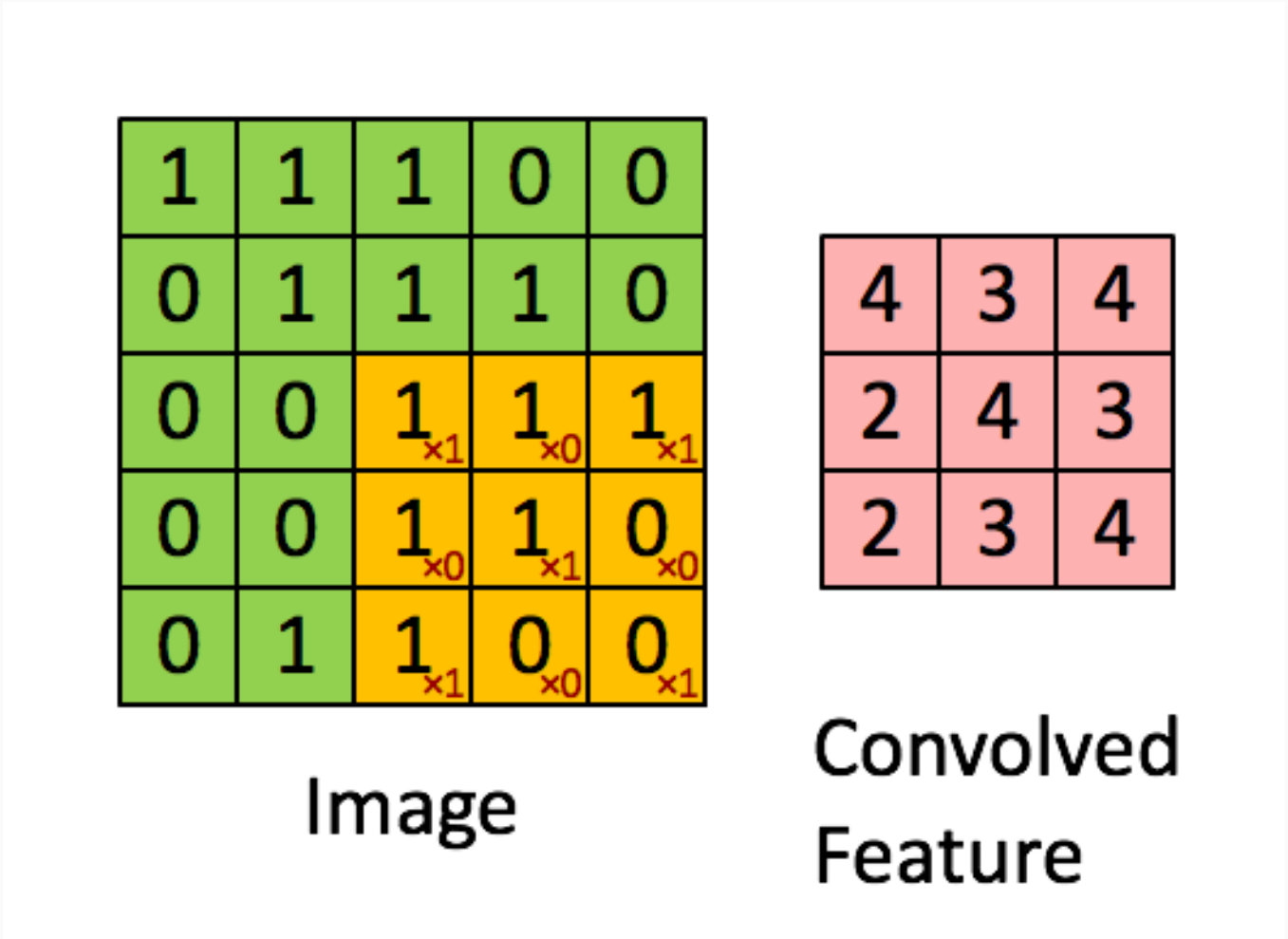


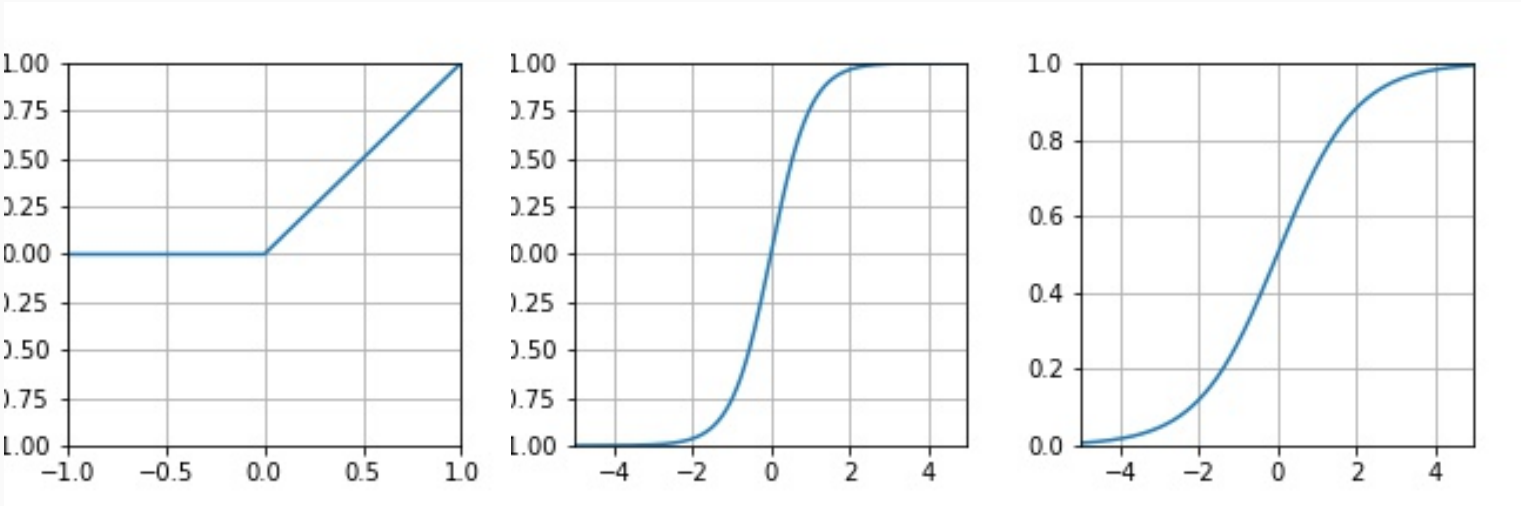
Figure 12: Image prise ici. De gauche à droite: image et **feature map**.

Non-linéarités

Une non-linéarité après une convolution permet

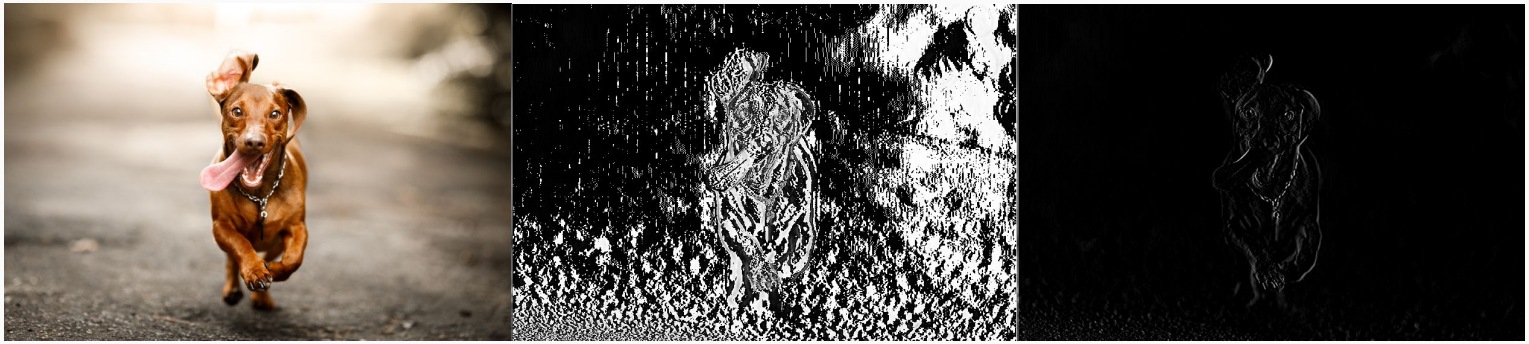
- Soit de normaliser la valeur des pixels de l'image convoluée (= feature map), e.g. dans $[0, 1]$ (sigmoid) ou $[-1, 1]$ (tanh).

- Soit de donner la possibilité de discriminer l'information importante. D'où le nom de fonction d'activation.
  Non-linéarité = éliminer de l'information.



(a) ReLU **(b)** tanh **(c)** sigmoid

Règle pour apprendre: Convolution + non-linéarité



(d) Original **(e)** Convolution **(f)** Conv. + ReLU

Figure 13: Kernel de taille $(1, 2)$ et paramètres $[-1, 1]$

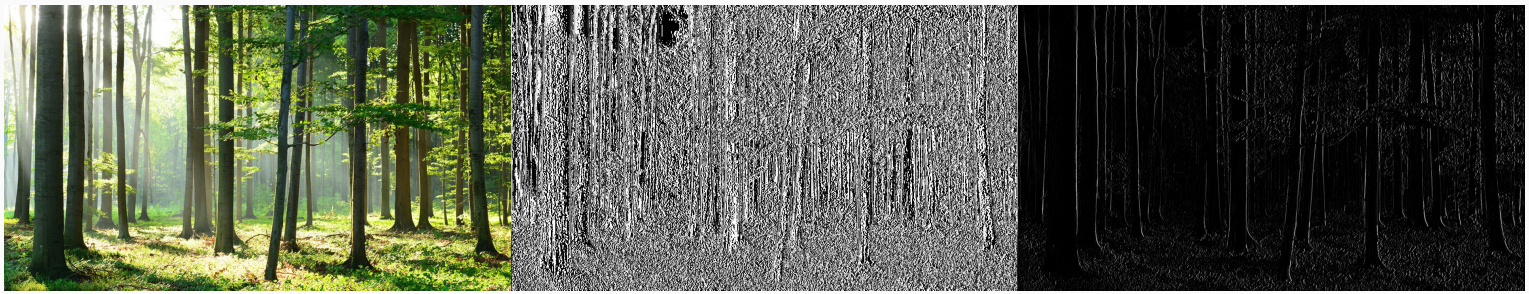
Des règles d'apprentissage.

Règle pour apprendre: Convolution. + non-linéarité

(a) Original **(b)** Convolution **(c)** Conv. + ReLU

Figure 14: Kernel Sobel Horizontal de taille $(3, 3)$ et paramètres $[[1, 0, -1], [2, 0, -2], [1, 0, -1]]$, détecte les contours verticaux. Il y en a peu dans cette image, l'information est éliminée.

Règle pour apprendre: Convolution. + non-linéarité



(a) Original **(b)** Convolution **(c)** Conv. + ReLU

Figure 15: Kernel Sobel Horizontal de taille $(3, 3)$ et paramètres $[[1, 0, -1], [2, 0, -2], [1, 0, -1]]$. Ici on détecte bien les éléments verticaux! La non-linéarité permet de filtrer l'information. **Si le jeu de donnée était constitué seulement de photos de forêts et d'escalier, ce filtre suffirait pour trouver une bonne règle de décision**.

Règle pour apprendre: Convolution. + non-linearité

(a) Original **(b)** Convolution **(c)** Conv. + ReLU

Figure 16: Exemple où l'image initiale n'a pas de texture sauf du bruit.

Convolution Stride

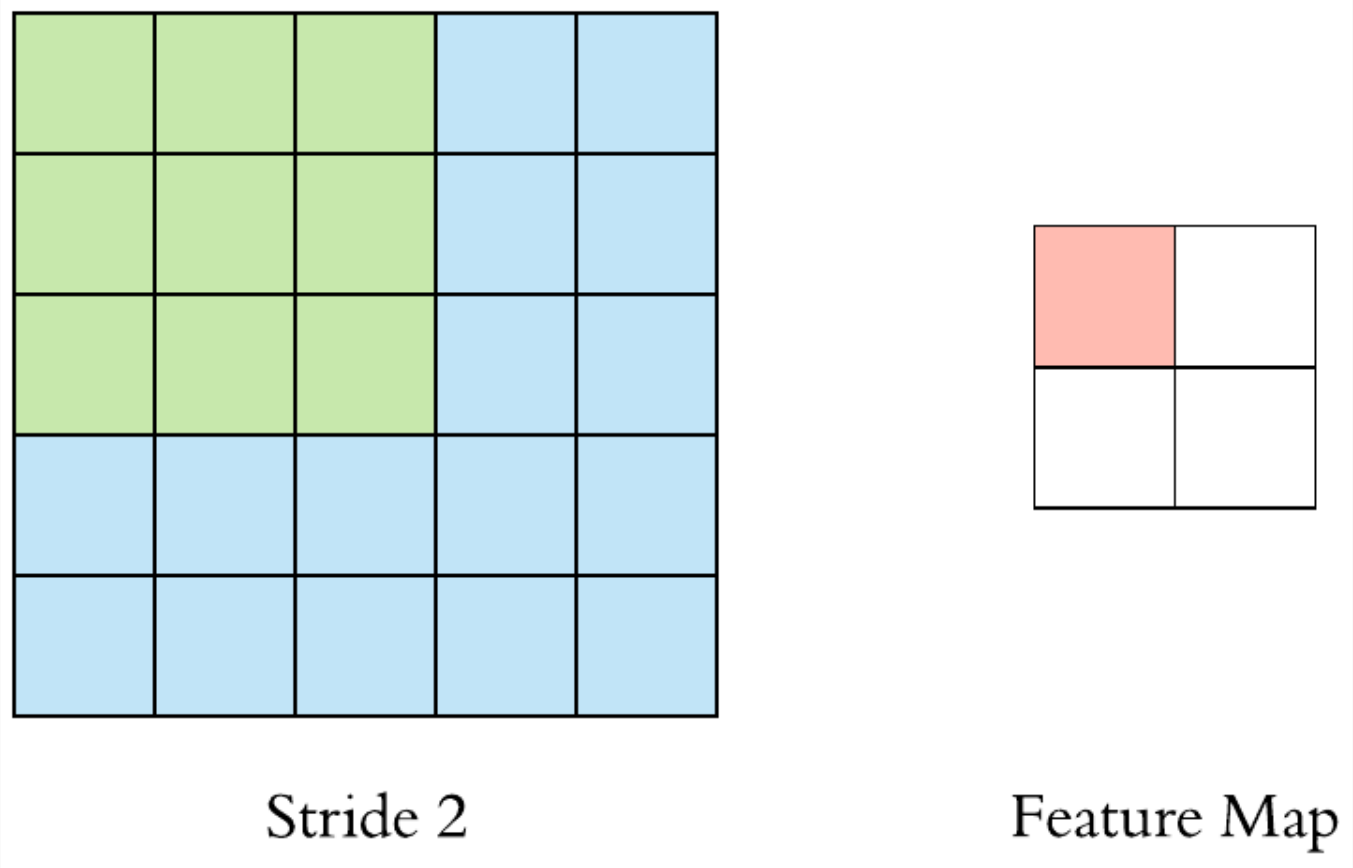


Figure 17: Noter que le résultat d'une convolution est appelée "feature map". Image prise ici.

Convolution Stride

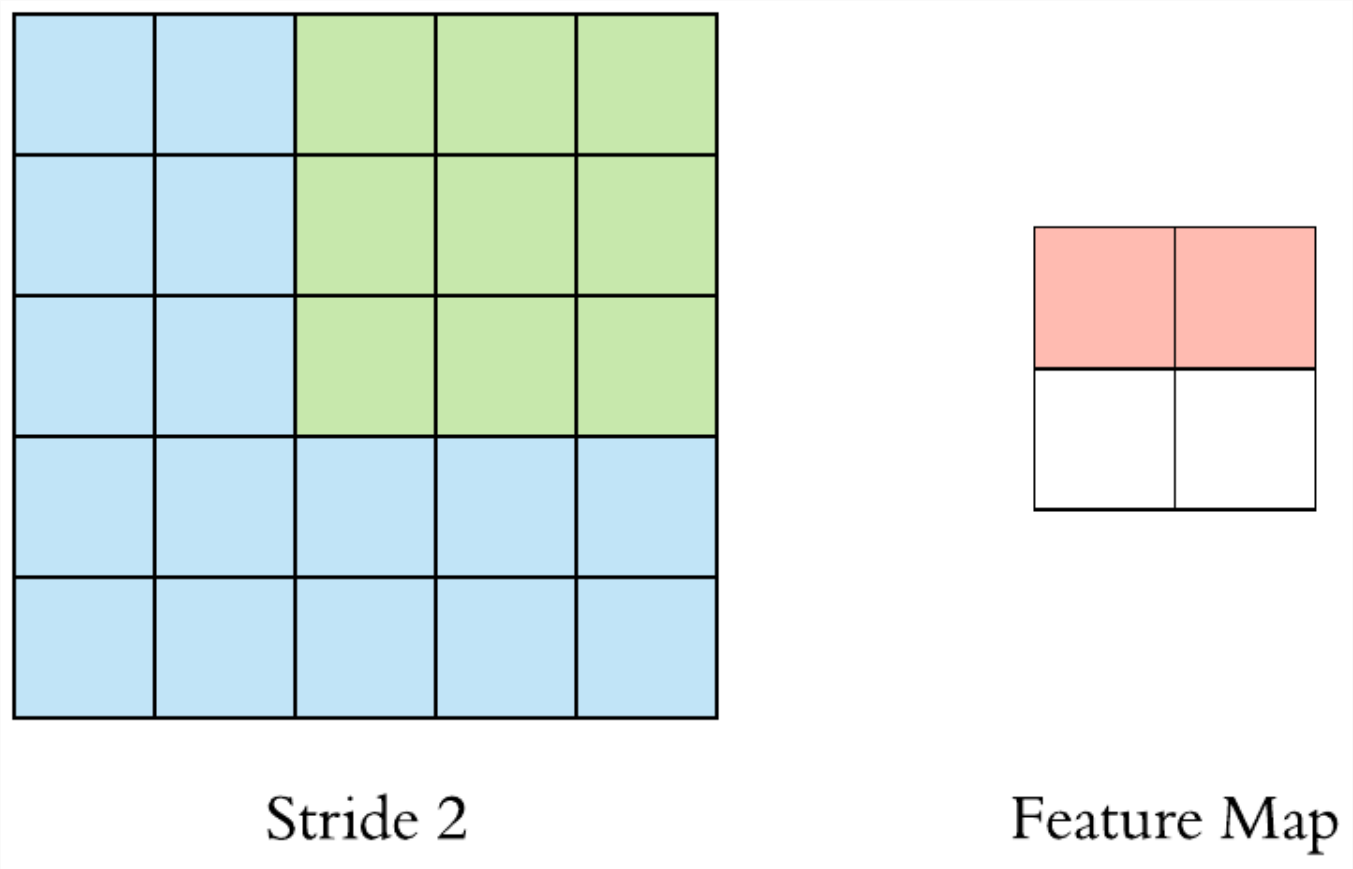


Figure 18: Image prise ici.

Convolution Stride

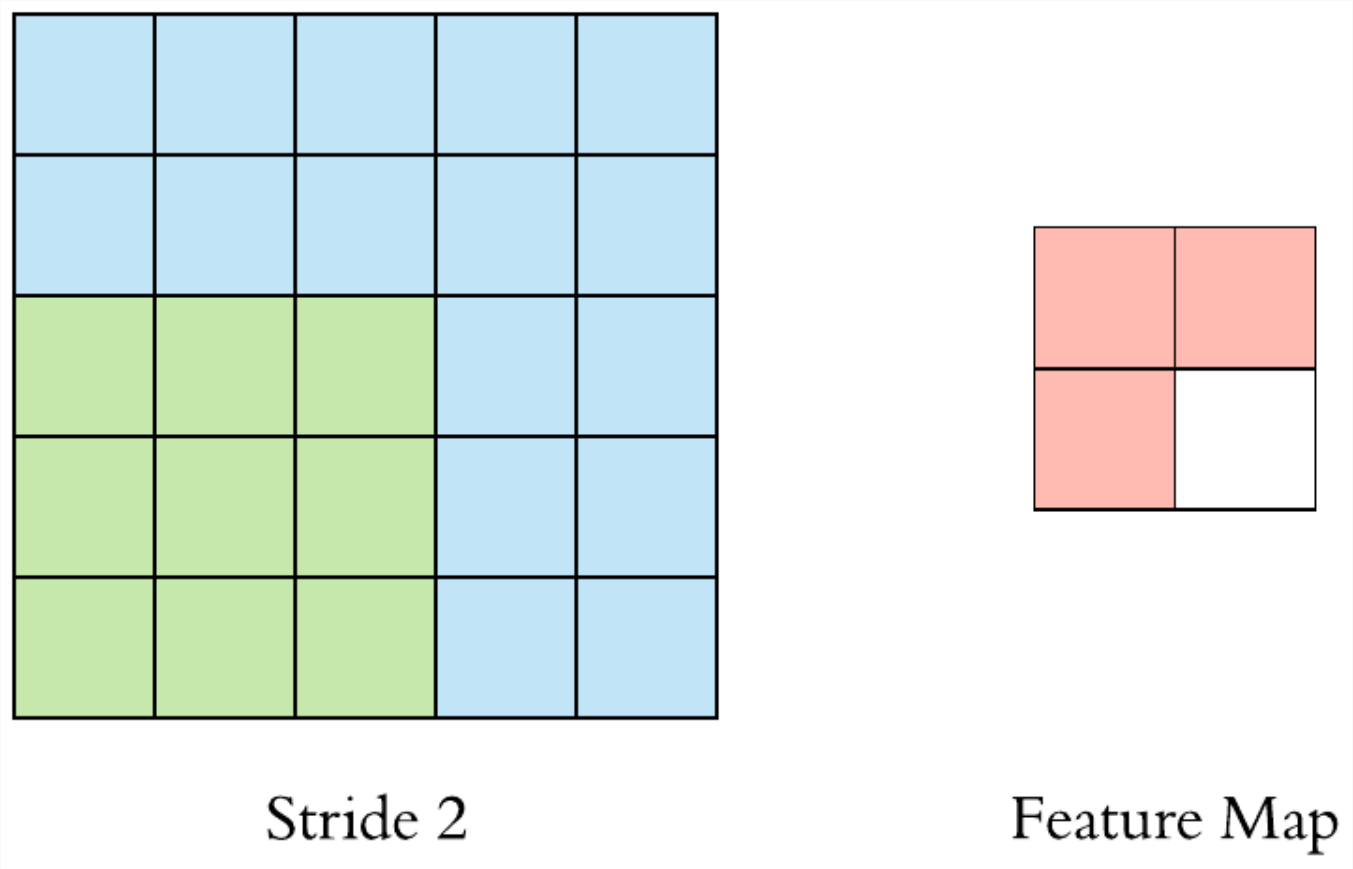


Figure 19: Image prise ici

Convolution Stride

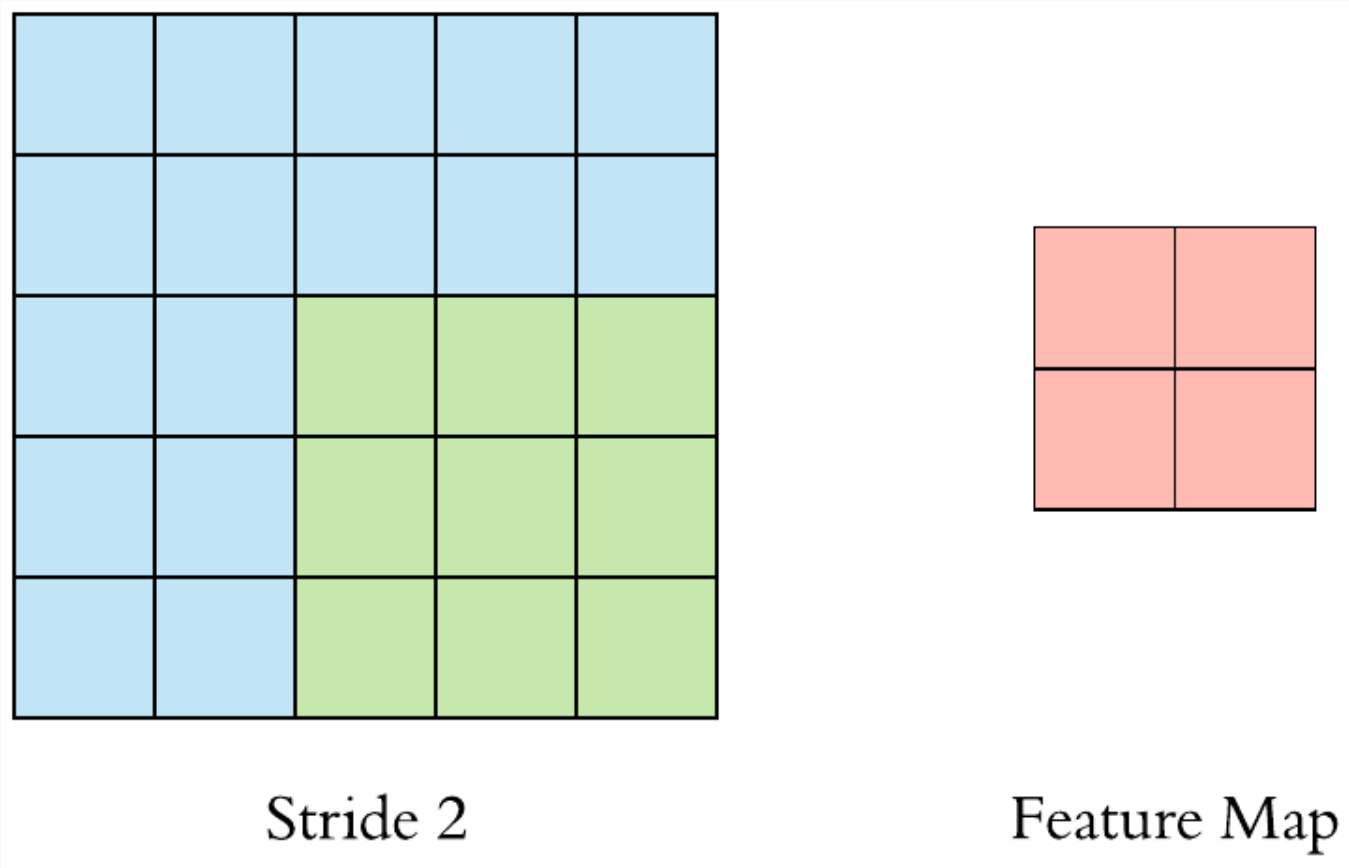


Figure 20: Image prise ici

Convolution Padding

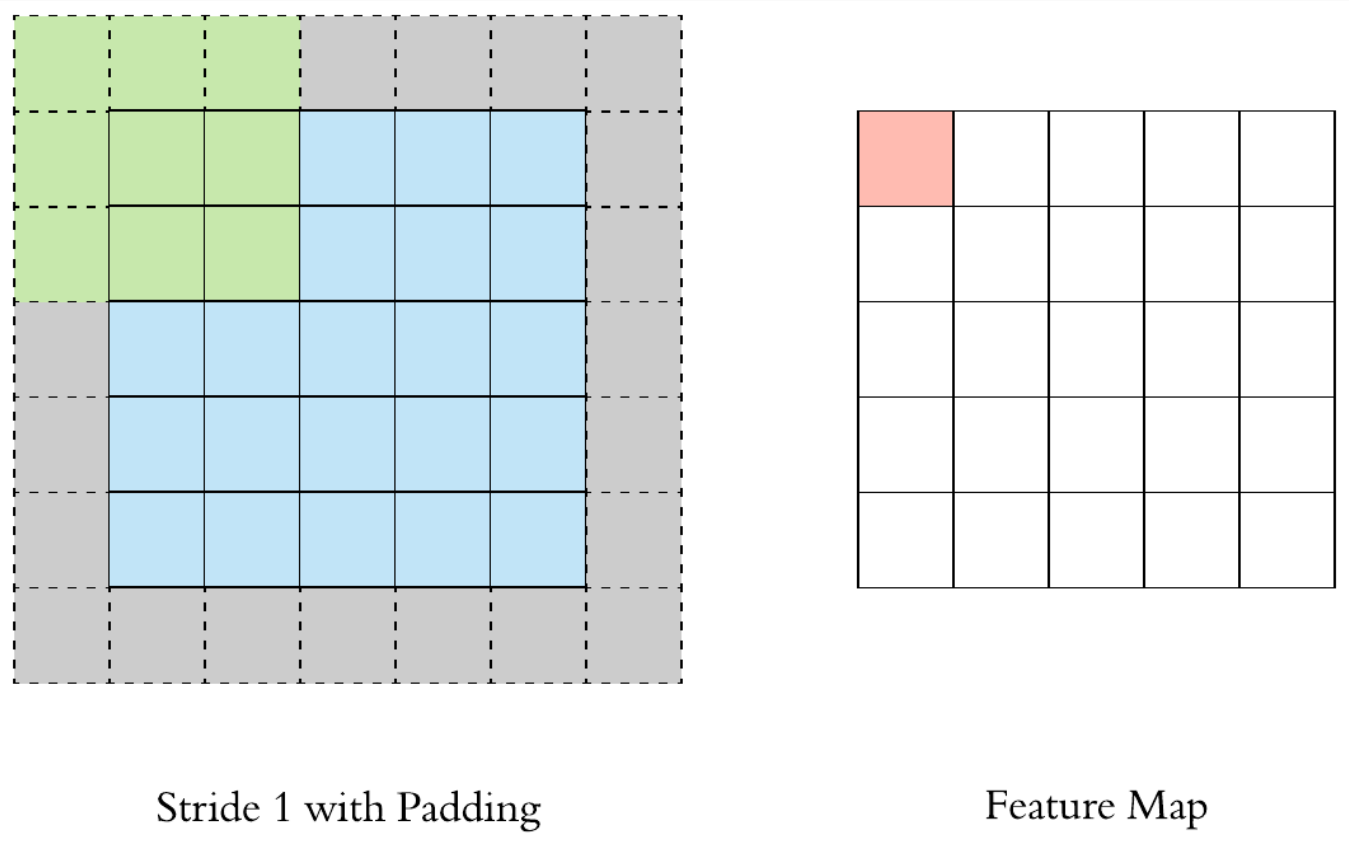


Figure 21: Image prise ici

Convolution Padding

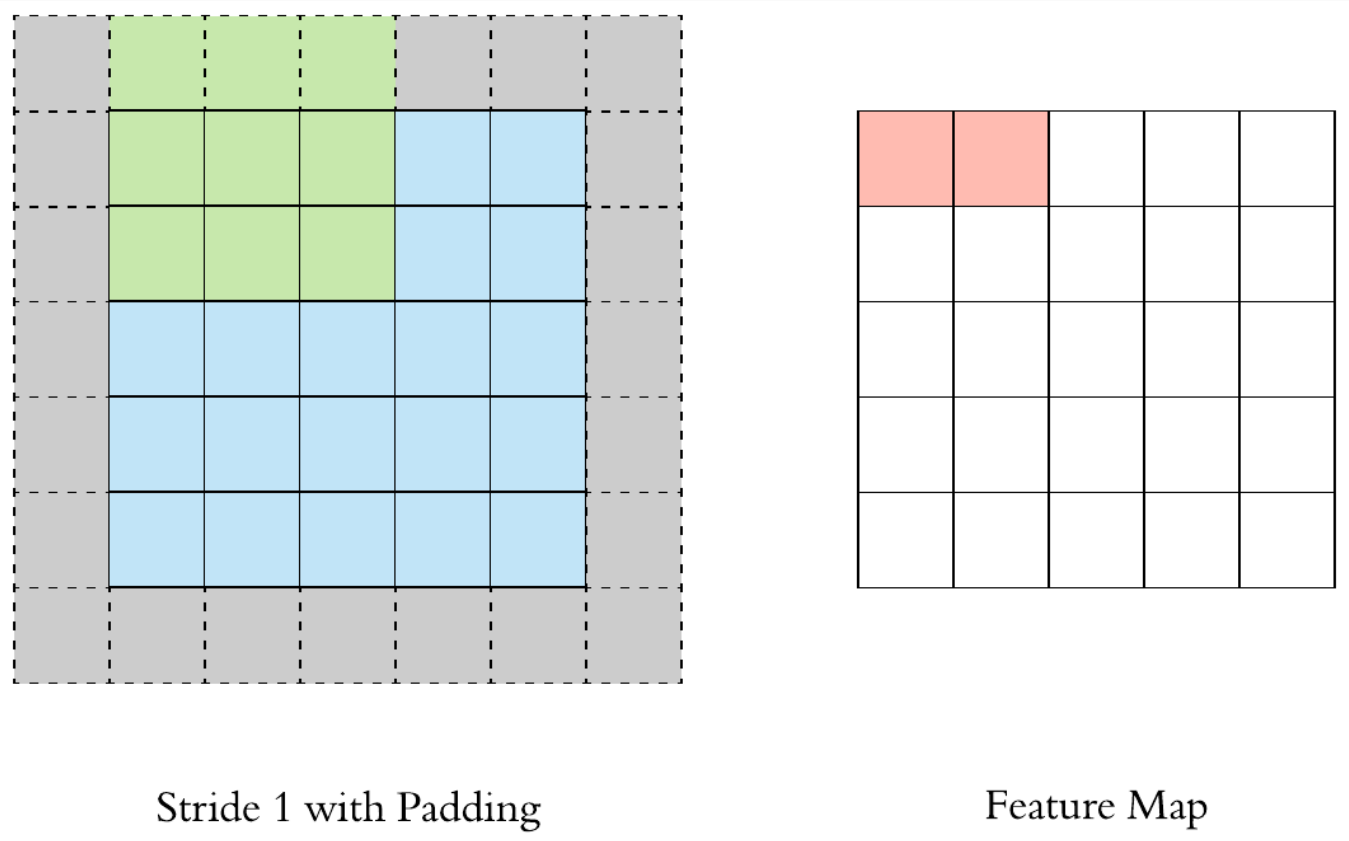


Figure 22: Image prise ici

Convolution Padding

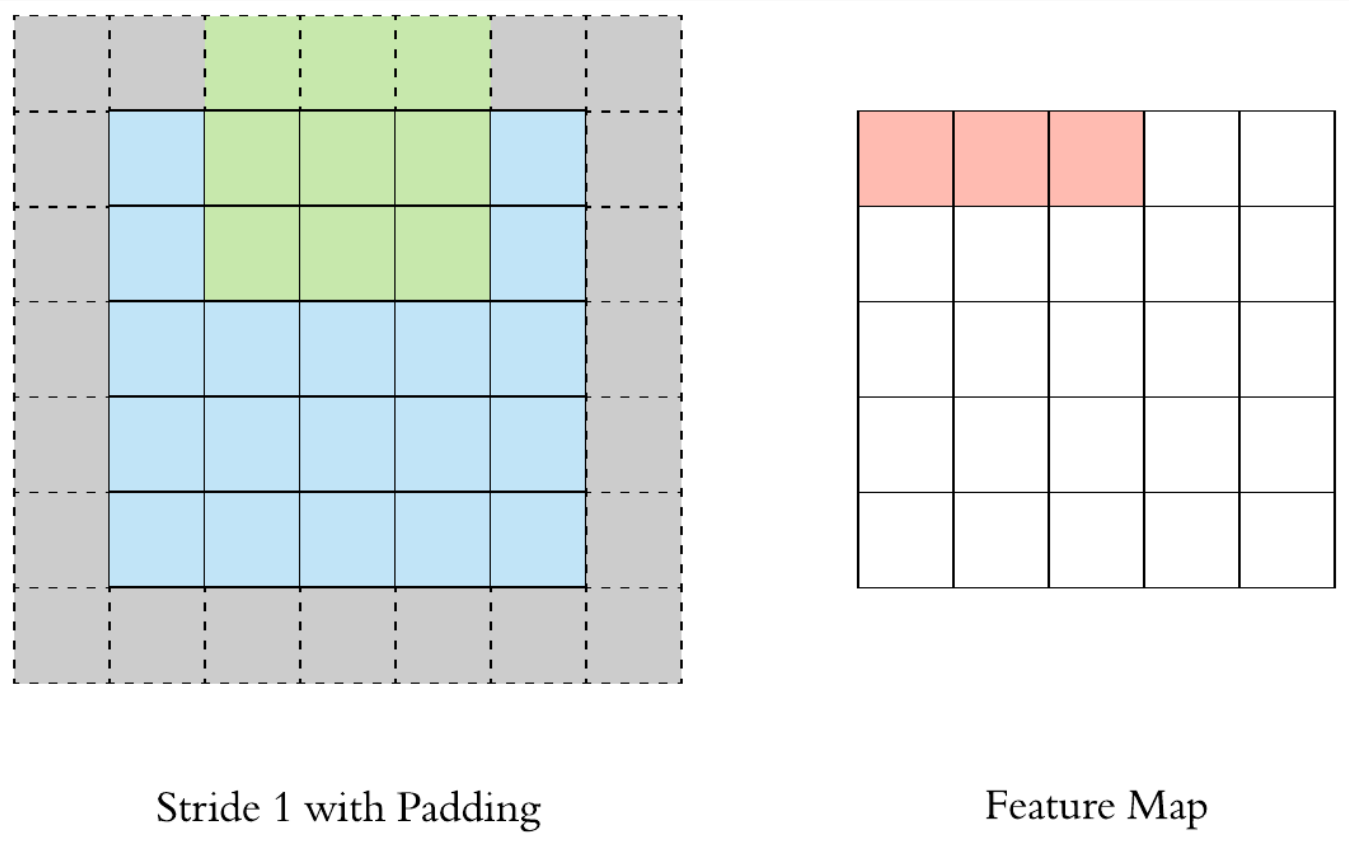


Figure 23: Image prise ici

Convolution Padding

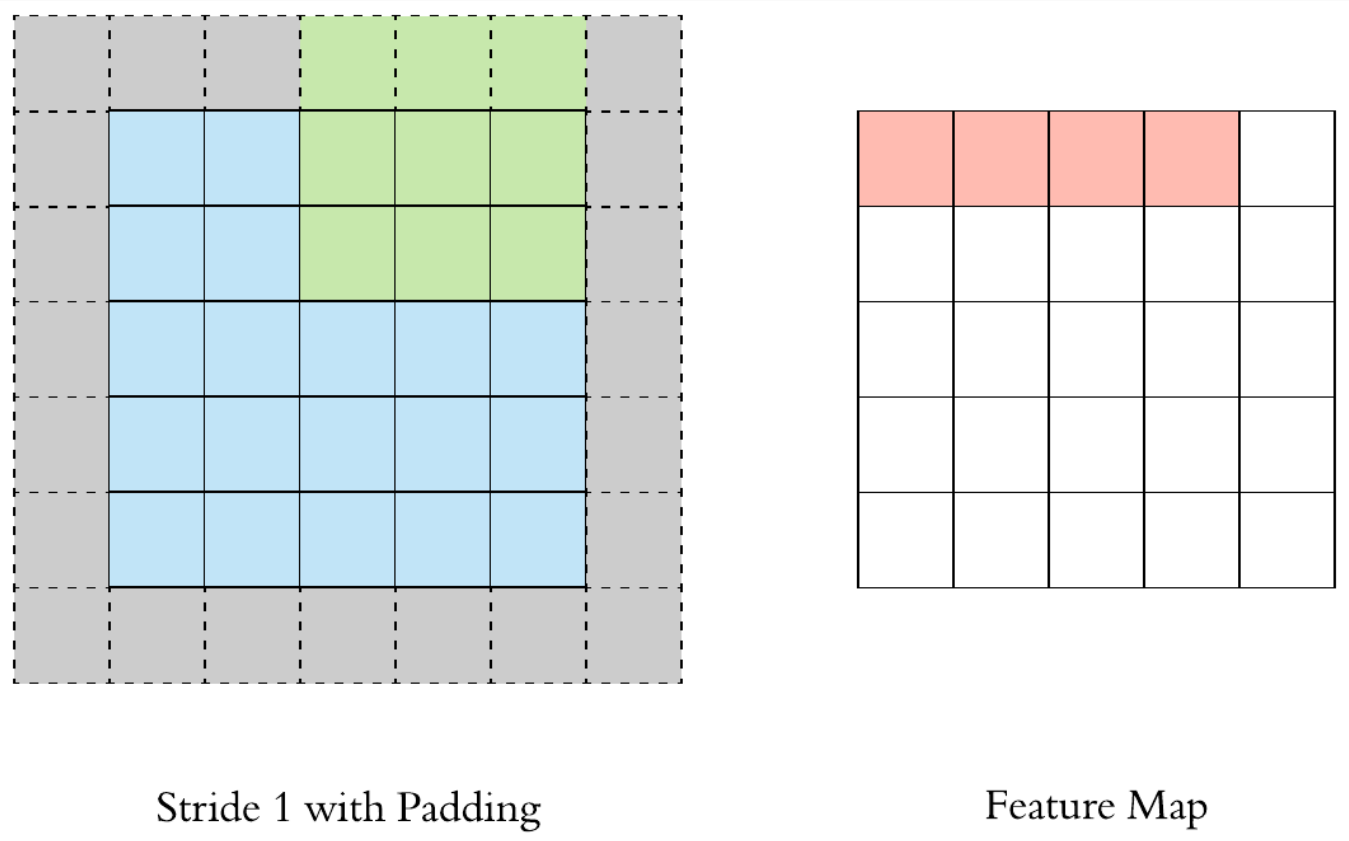


Figure 24: Image prise ici

Les opérations de Pooling

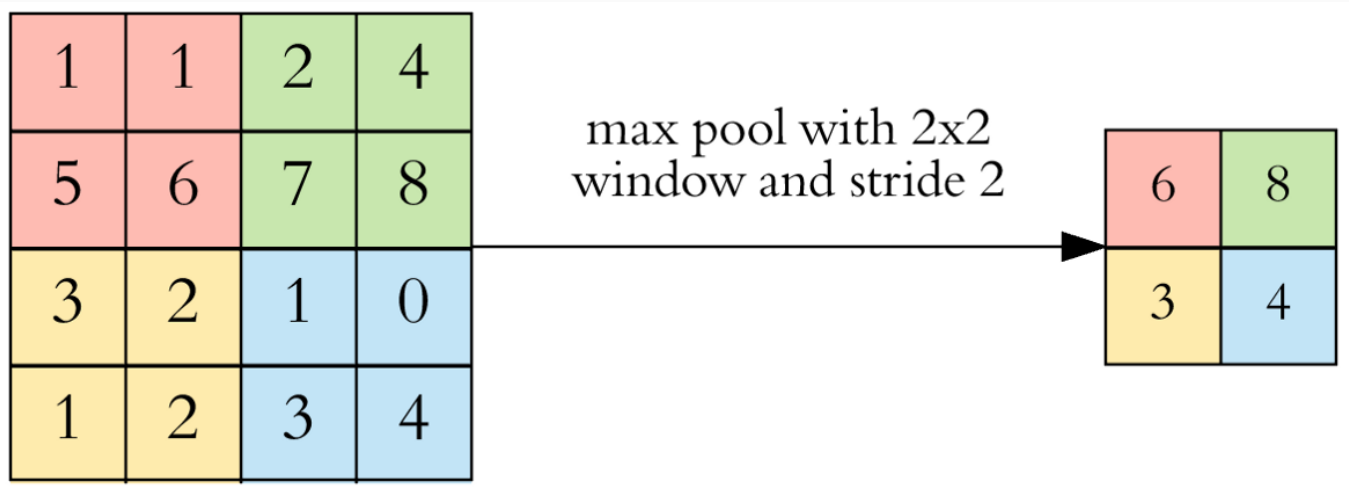


Figure 25: Pooling = aggregation. On passe d'une image (**feature map** de 16 pixels à une de 4 pixels!): 1) réduction de la dimension de l'image 2) garde seulement les éléments importants 3) réduit le coût computationnel du réseau. Image prise ici.

Layers, Channels

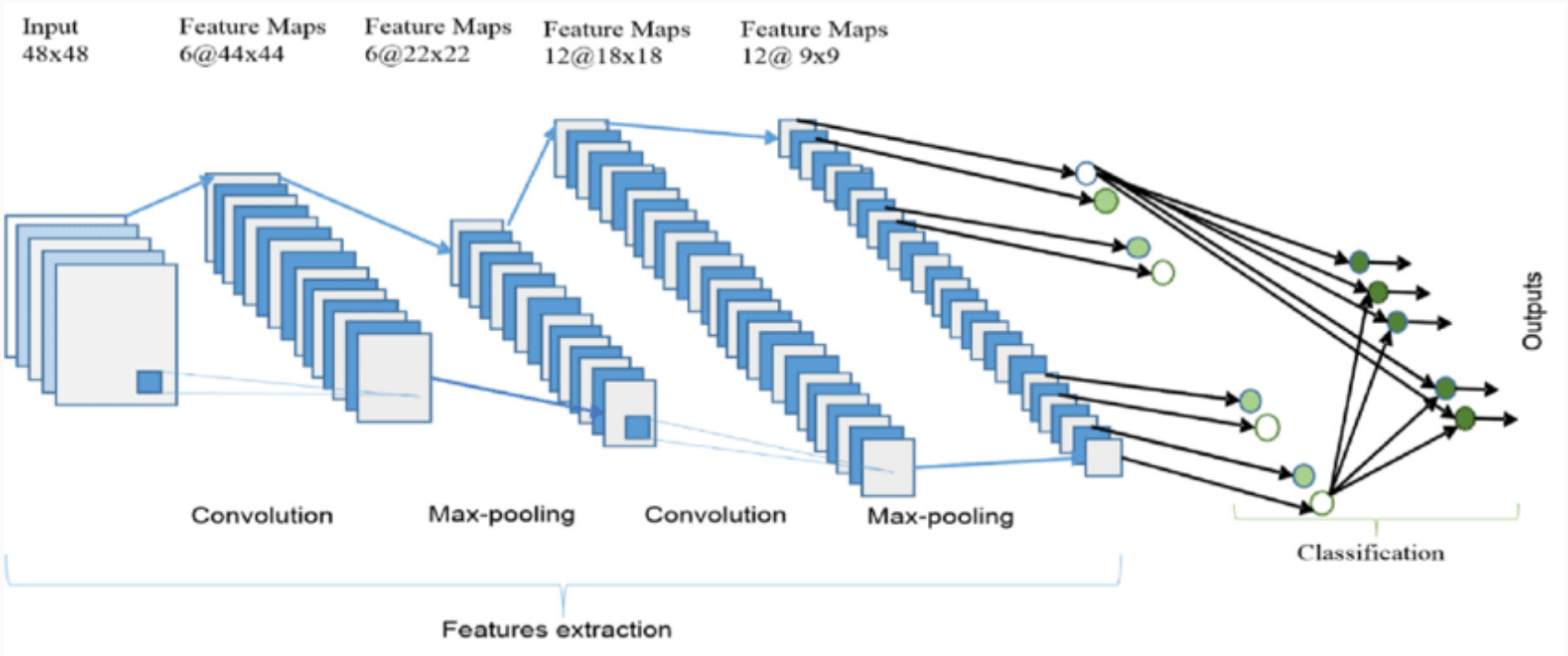


Figure 26: Distinction entre apprendre une représentation (**Features extraction**) et la classification. Image prise dans (Alom et al., 2019). 2 couches (**layers**) de convolution. 5 canaux (**channels**) dans l'image de départ. Chaque couche intermédiaire est une *feature map*. (Penser pour le calcul en parallèle)

Convolution VS Fully-connected

Convolution a très peu de paramètres comparé à une couche linéaire (**linear layer** ou **fully-connected**).

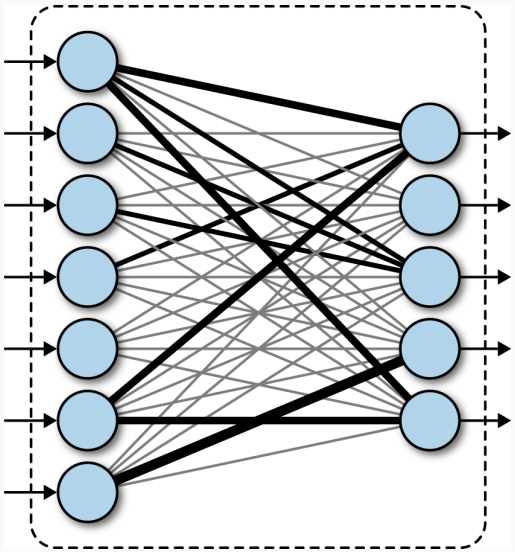


Figure 27: Pour convolution, image = tensor (H,W,C). Pour couche linéaire, image = vecteur.

Classification multi-classe

K classes, sortie binaire $y_k \in \mathbb{R}^K$.

- Les réseau sort des scores $(s_1, \dots s_K) \in \mathbb{R}^K$
- Probabilités calculées par fonction Softmax

$$p_k = \frac{\exp(s_k)}{\sum_{l=1}^{K} \exp(s_l)}.$$

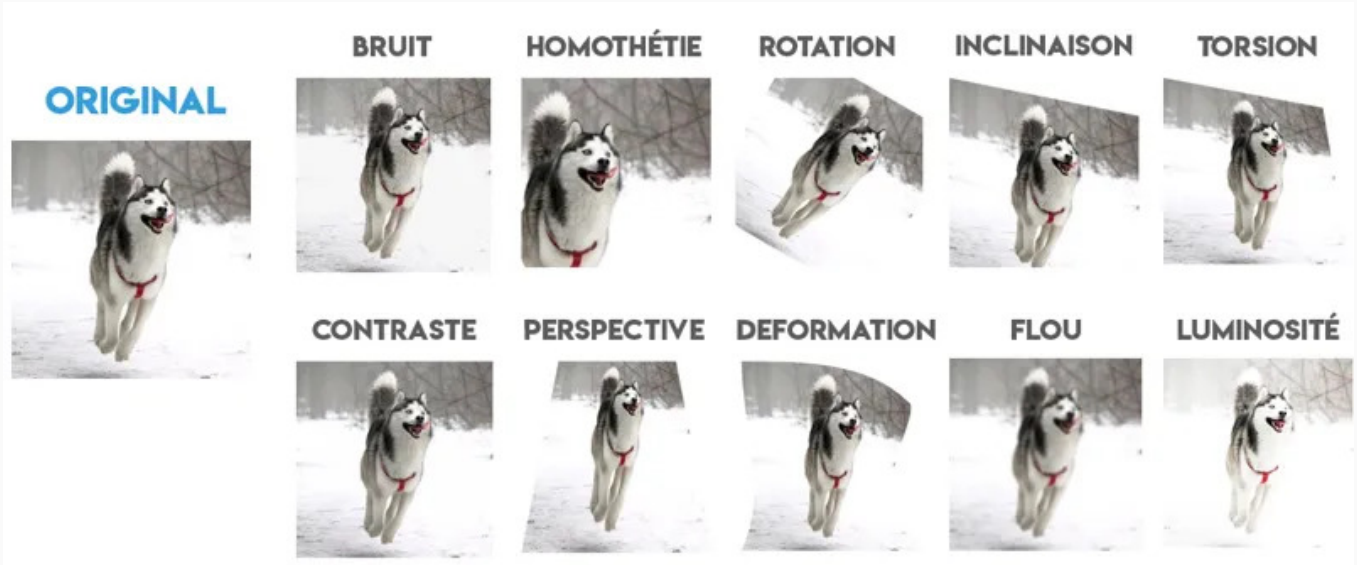
- Fonction de coût

$$-\sum_{k=1}^{K} y_k \log(p_k). \qquad \text{(Cross-Entropy loss)}$$

- Sorties s_k: **logits** (log-probabilités, regression logistique).

Mise en pratique: notebook CNN sur CIFAR-10.

Data Augmentation



Idée: transformations qui **préservent la classe de l'image**.

Simple à implémenter pendant l'entraînement (on ne stocke pas le dataset *augmenté*)!

$\rightarrow$ Réduit le sur-apprentissage.