Réseaux Convolutifs

Julien Romero

La dernière fois

- Procédure d'entraînement
- Overfit, underfit
- Découpage du dataset et cas particuliers
- Métriques de classification
- Régularisation et weight decay
- Le moment
- Optimiseur : **ADAM**, AdaGrad, RMSProp

Le problème avec le MLP

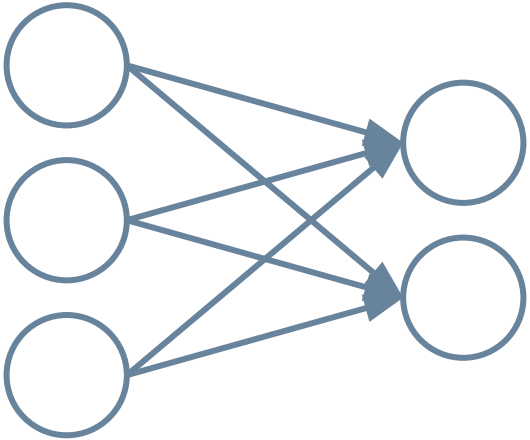
Le MLP peut en théorie résoudre tous les problèmes mais il détruit la structure de l'entrée, ce qui complique la tâche.

Dans une image, l'aspect spacial est très important, on ne veut pas avoir à le réapprendre.

Solution : Créer une nouvelle opération préservant la spatialité de l'image

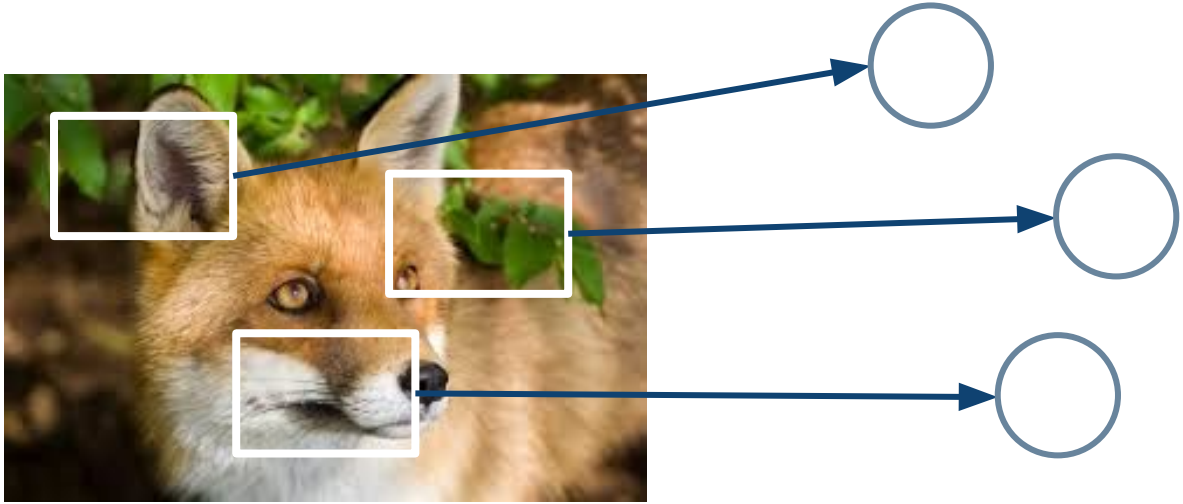
Le perceptron voit tout

- Chaque élément de la sortie du perceptron dépend de toutes les entrées.



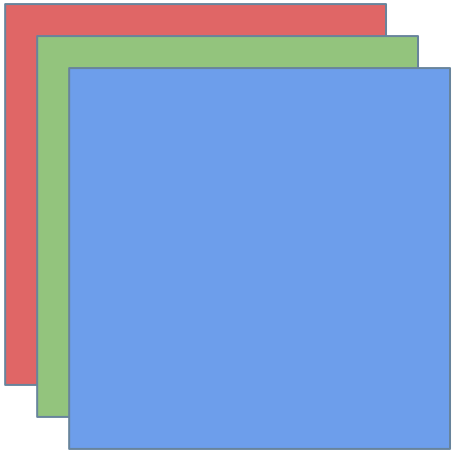
Idée des réseaux convolutifs

- Chaque élément de la sortie de dépend que d'un sous ensemble restreint de l'entrée
  - Ce sous-ensemble est composé de points à côté dans une image.

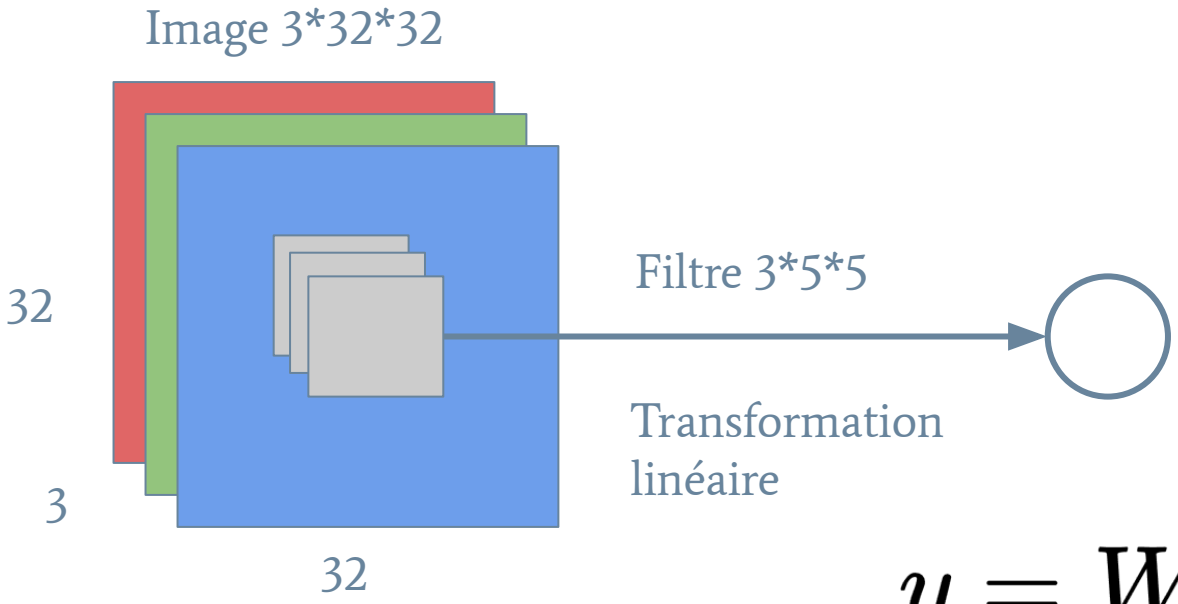


La structure d'une image

- Une image contient un ensemble de pixel représentés en 2D
- Suivant l'image, un pixel est représenté différemment
  - Noir/blanc, niveaux de gris : une seule dimension
  - Couleurs : trois dimensions (Rouge, Vert, Bleu) (+ 1 de transparence)



La couche de convolution

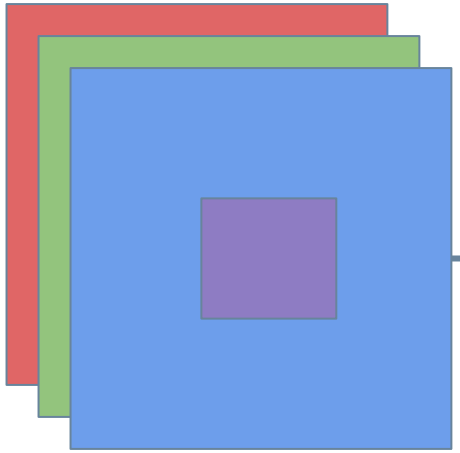


$$y = W_{filtre} * x + b$$

La couche de convolution

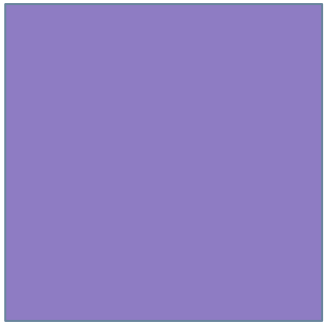
On perd deux pixels sur chaque bord.

Image 3*32*32



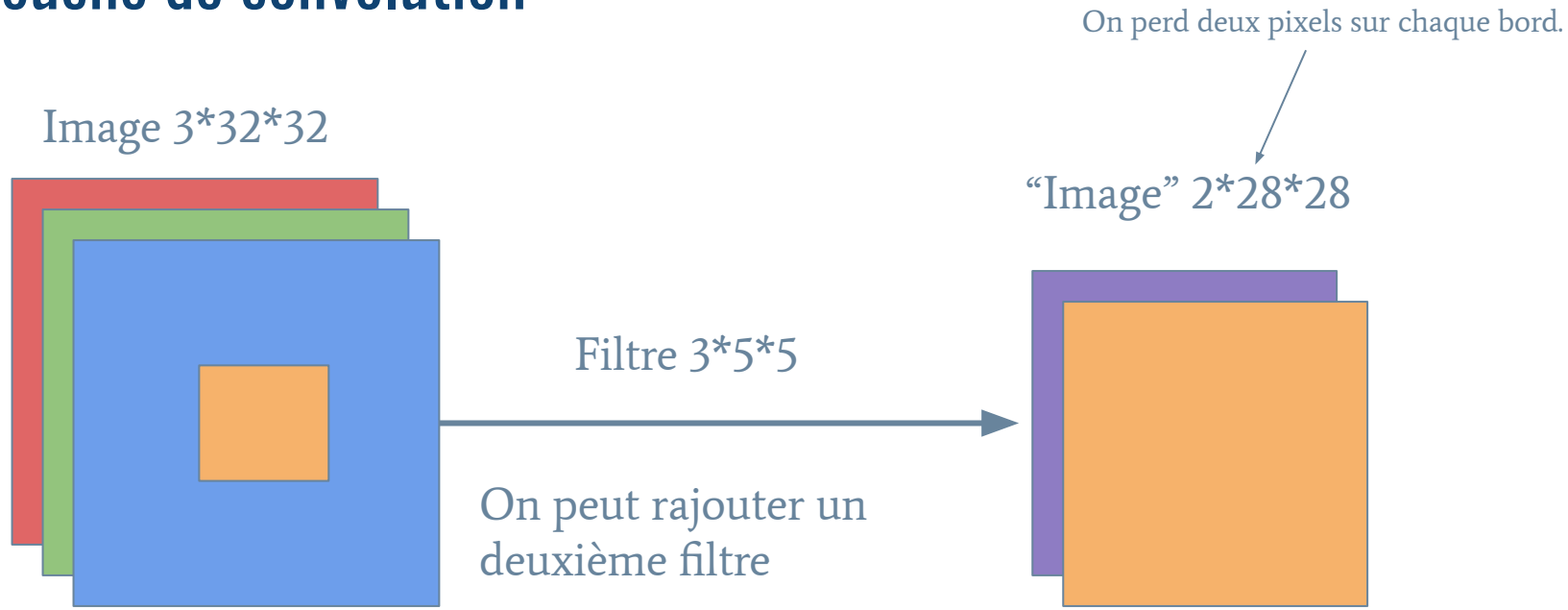
Filtre 3*5*5

“Image” 1*28*28

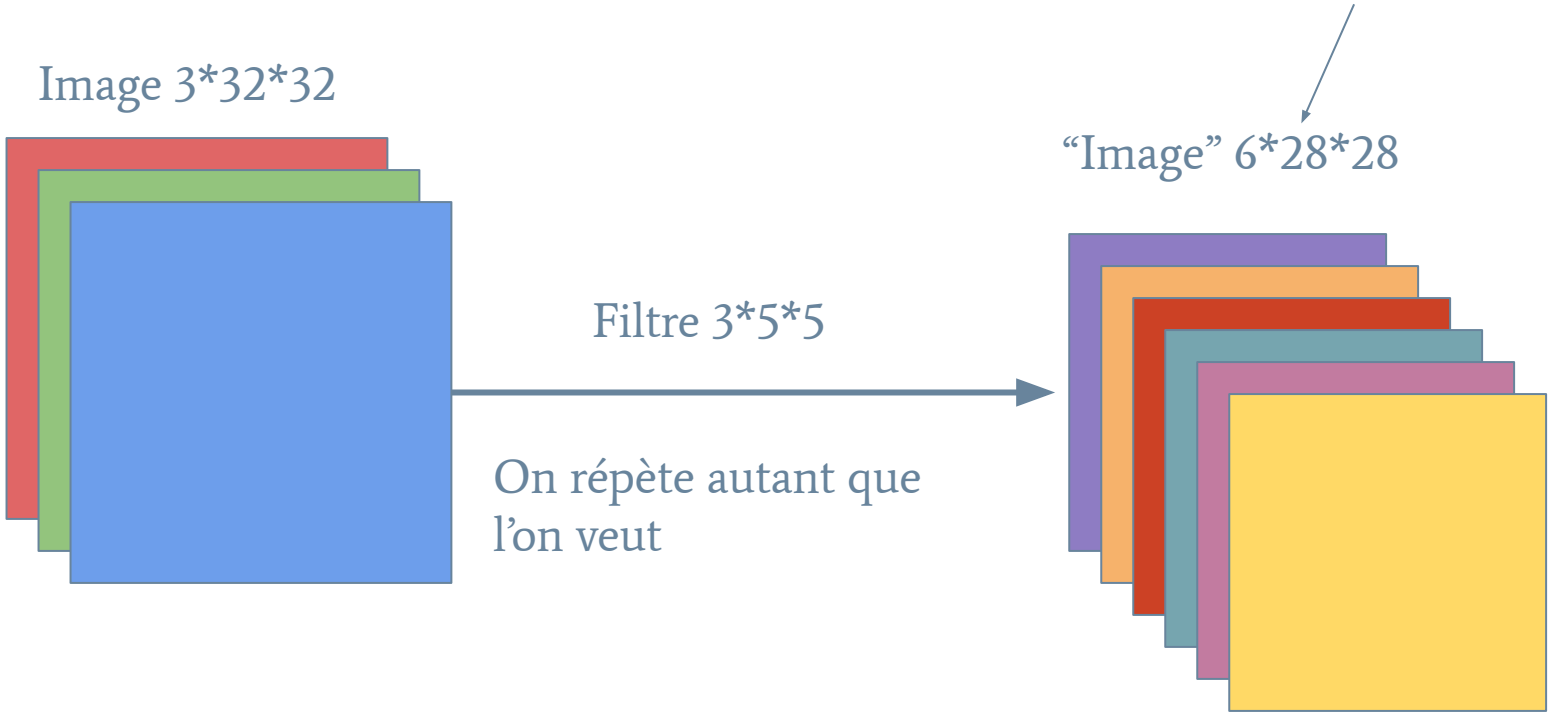


On applique le **même filtre** sur tous les points possibles et on peut reconstituer un plan 2D

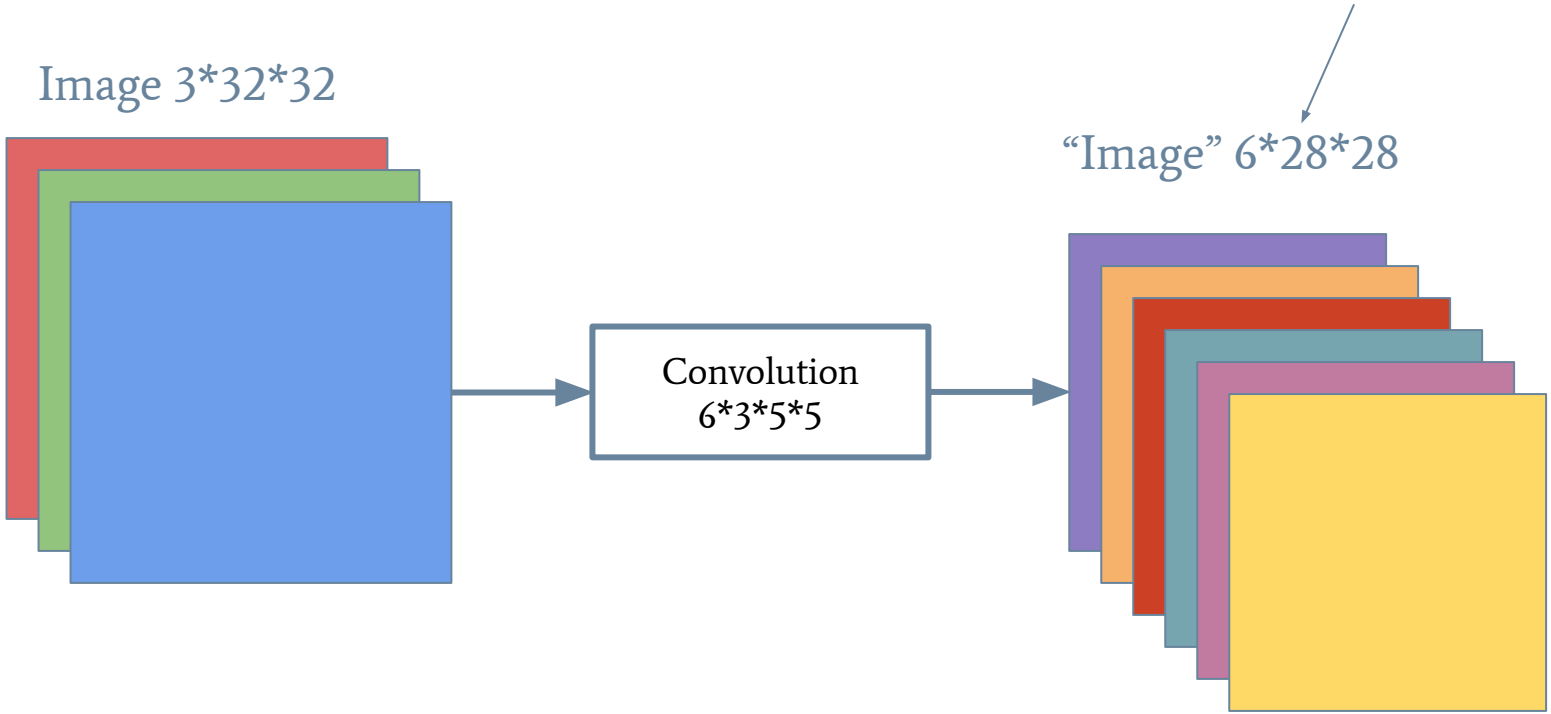
La couche de convolution



La couche de convolution

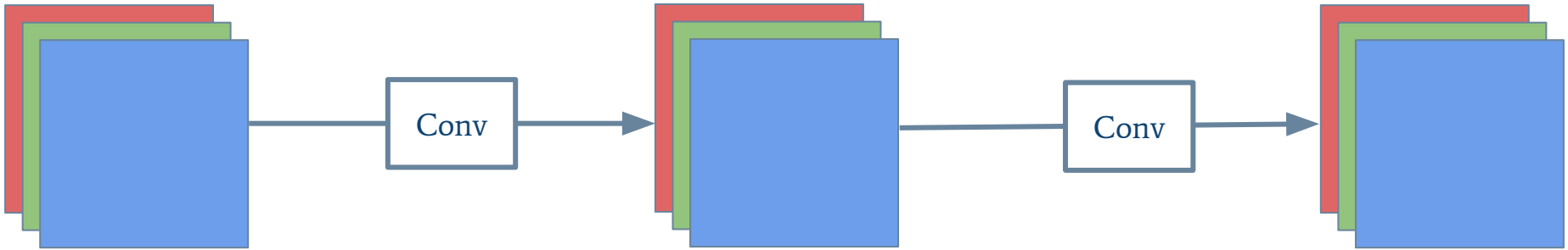


La couche de convolution



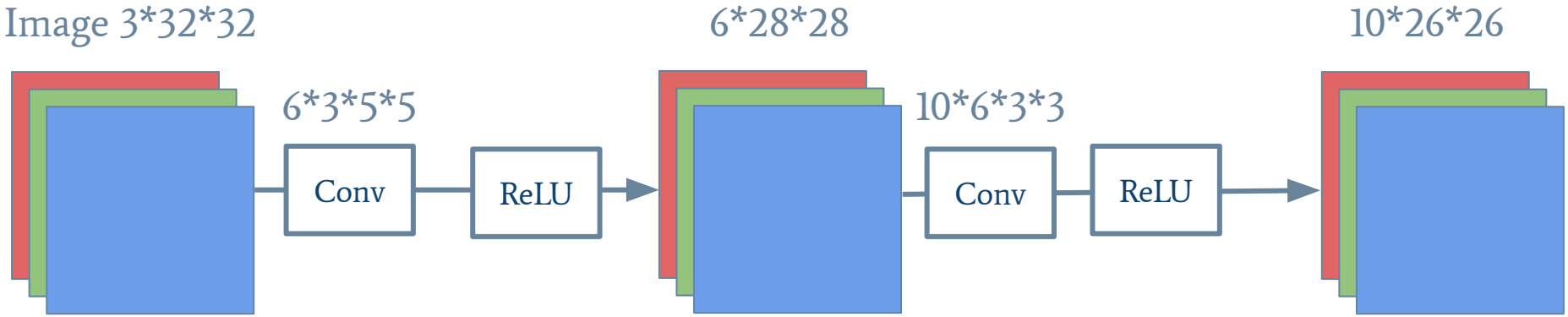
La convolution multicouche

On peut chaîner les convolutions avec des couches cachées mais...

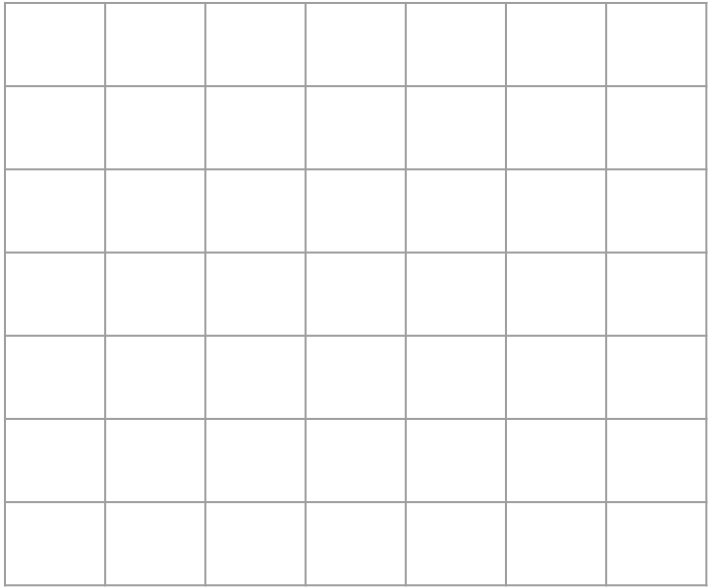


La convolution multicouche

On peut chaîner les convolutions avec des couches cachées mais...
Comme les MLP, sans non-linéarité, on ne gagne rien

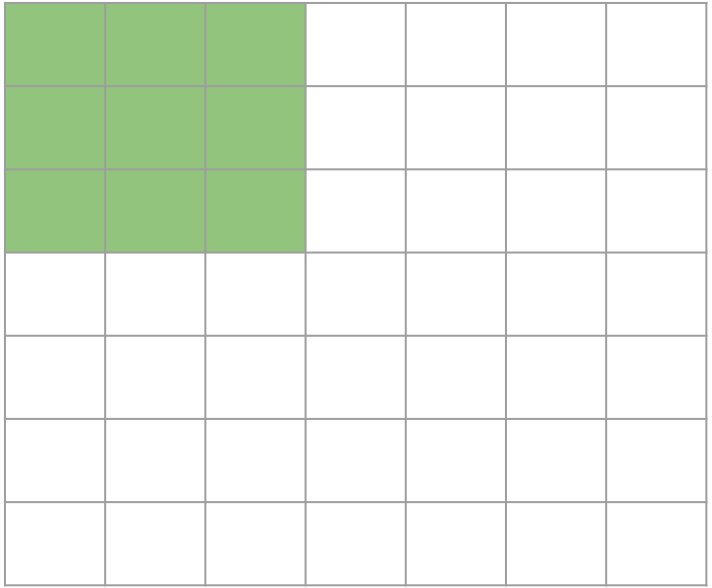


Zoom sur les dimensions spaciales



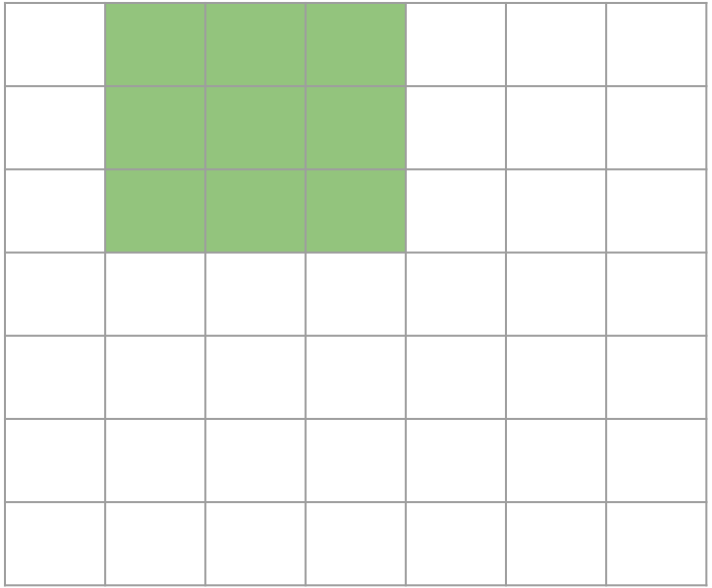
Entrée : 7 * 7
Filtre : 3 * 3

Zoom sur les dimensions spaciales



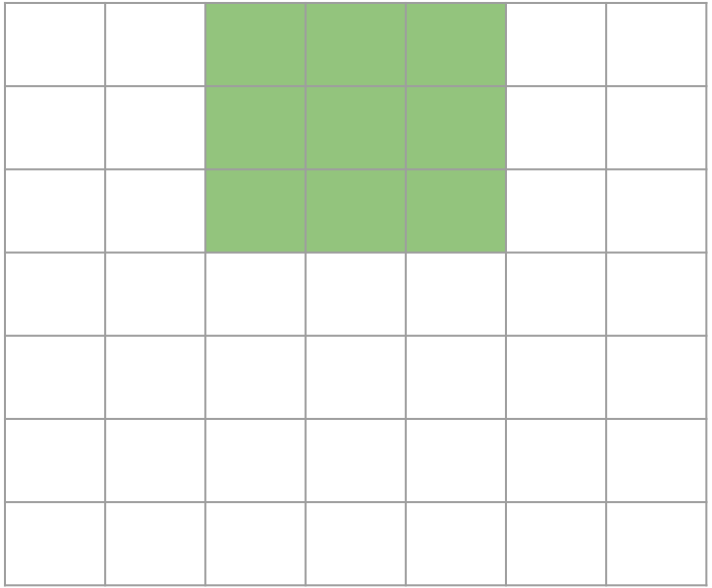
Entrée : 7 * 7
Filtre : 3 * 3

Zoom sur les dimensions spaciales



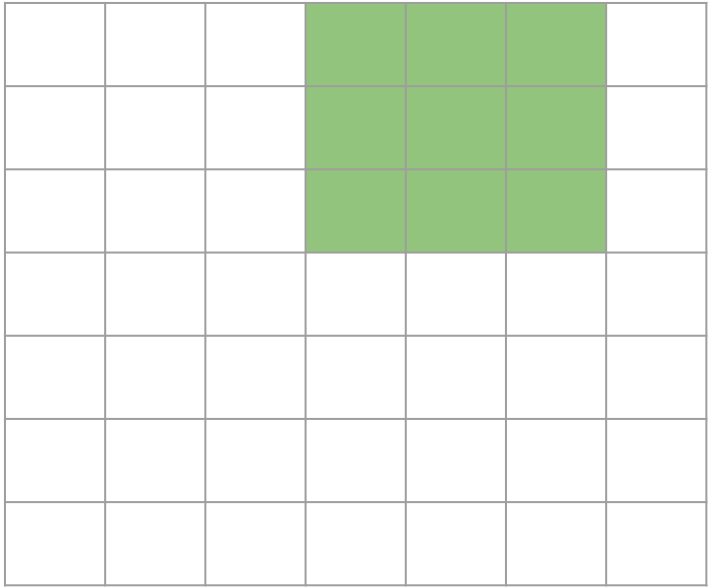
Entrée : 7 * 7
Filtre : 3 * 3

Zoom sur les dimensions spaciales



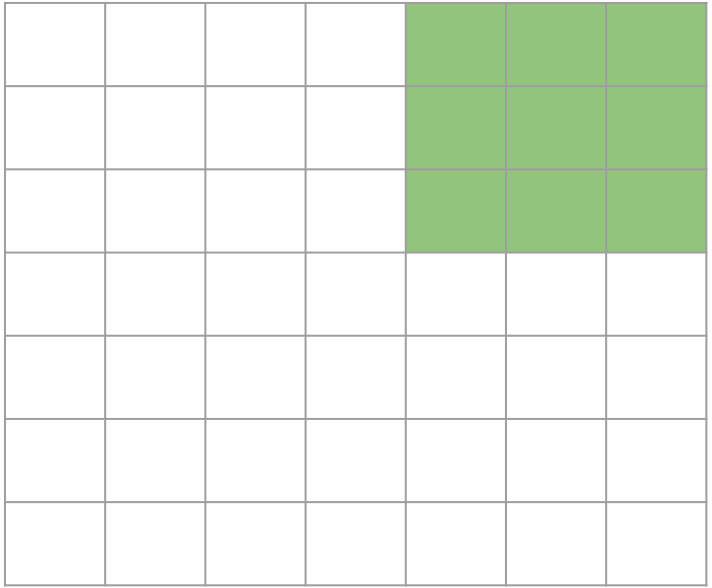
Entrée : 7 * 7
Filtre : 3 * 3

Zoom sur les dimensions spaciales



Entrée : 7 * 7
Filtre : 3 * 3

Zoom sur les dimensions spaciales



Entrée : 7 * 7
Filtre : 3 * 3
Sortie : 5 * 5

En général :
Entrée : N
Filtre : K
Sortie : N - K + 1

On perd en dimension !

La padding

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

Entrée : 7 * 7
Filtre : 3 * 3
Sortie : 5 * 5

En général :
Entrée : N
Filtre : K
Sortie : N - K + 1

On perd en dimension !
Solution : on met des faux bords

La padding

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0

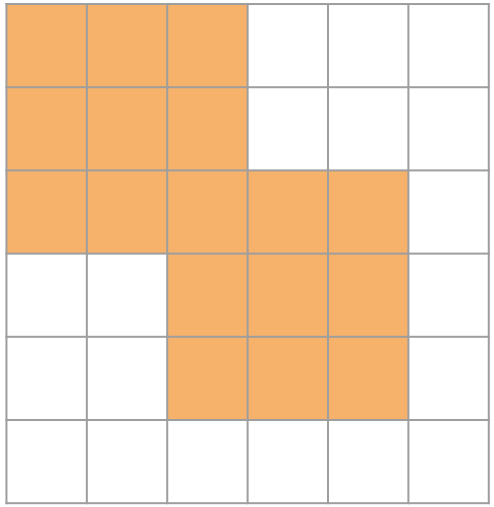
Entrée : 7 * 7
Filtre : 3 * 3
Sortie : 5 * 5

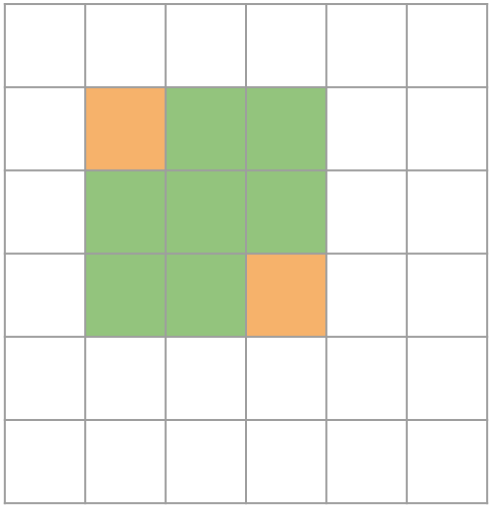
En général :
Entrée : N
Filtre : K
Padding : P (=(K-1)/2 en général)
Sortie : N - K + 1 + 2P

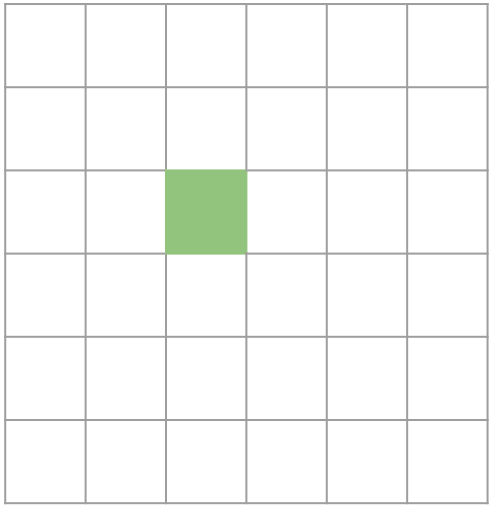
Comment “voir” toute l’image ?

- Un kernel ne voit qu’une partie restreinte de l’image
- Mais on a besoin de tout l’image pour faire notre prédiction !
- Plus on a de couche, plus on voit grand.
  - **Receptive Fields**

Receptive Fields





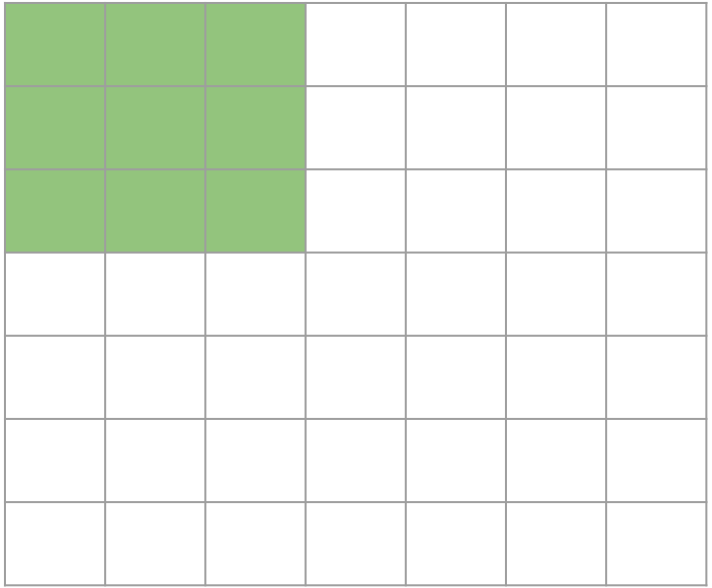


Problème des grandes images : il faut beaucoup de couche pour qu'une sortie "voit" toute l'entrée.
Avec des filtres de taille K, après L couches, une sortie "voit" 1 + L * (K-1) pixels.

Receptive Fields

Problème des grandes images : il faut beaucoup de couche pour qu'une sortie "voit" toute l'entrée.
Avec des filtres de taille K, après L couches, une sortie "voit" 1 + L * (K-1) pixels.

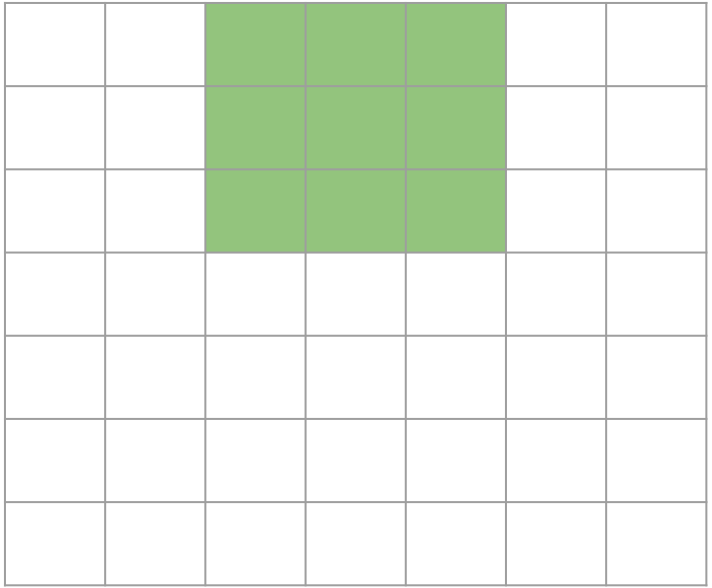
Strided Convolution



Idée : On fait des sauts

Entrée : 7 * 7
Filtre : 3 * 3
Stride : 2

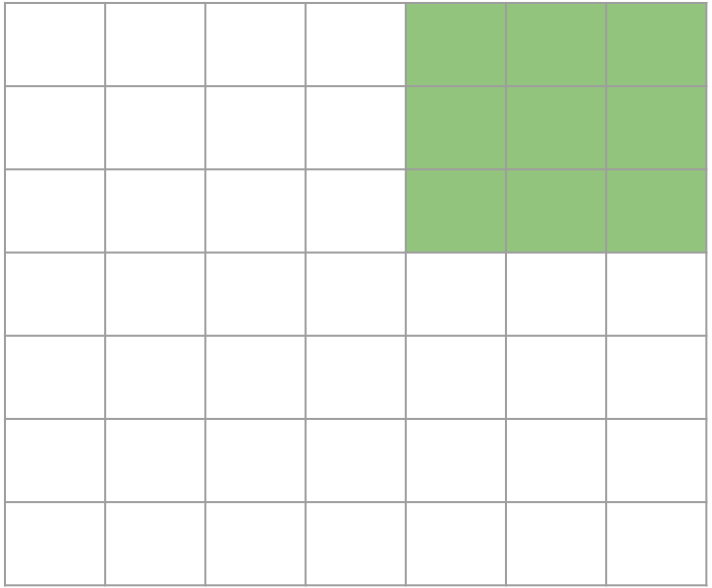
Strided Convolution



Idée : On fait des sauts

Entrée : 7 * 7
Filtre : 3 * 3
Stride : 2

Strided Convolution



Idée : On fait des sauts

Entrée : 7 * 7
Filtre : 3 * 3
Stride : 2
Output : 3 * 3

En général :
Entrée : N
Filtre : K
Padding : P
Stride : S
Sortie : (N - K + 2P) / S + 1

Exemple

J'ai :

- Entrée : image de 3x32x32
- 10 filtres 5x5 avec une stride de 1 et un padding de 2

Quelle est la taille de la sortie ?

Exemple

J'ai :

- Entrée : image de 3x32x32
- 10 filtres 5x5 avec une stride de 1 et un padding de 2

Quelle est la taille de la sortie ?

- (32+2*2-5)/1+1 = 32
- Donc **10 * 32 * 32**

Exemple

J'ai :

- Entrée : image de 3x32x32
- 10 filtres 5x5 avec une stride de 1 et un padding de 2

Comment j'ai de paramètres ?

Exemple

J'ai :

- Entrée : image de 3x32x32
- 10 filtres 5x5 avec une stride de 1 et un padding de 2

Comment j'ai de paramètres ?

- Chaque filtre a 3 * 5 * 5 + 1 (biais) = 76
- On a 10 filtres, donc 760 paramètres

La convolution en bref

- **Entrée** : Cin x H x W
- **Hyperparamètres** :
  - Taille du filtre/kernel : Kh x Kw
  - Nombre de filtres : Cout
  - Padding : P
  - Stride : S
- **Taille du tenseur** : Cout x Cin x Kh x Kw
- **Biais** : Cout
- **Taille de la sortie** : Cout x H' x W', avec
  - $H' = (H - K + 2P) / S + 1$
  - $W' = (W - K + 2P) / S + 1$

La convolution en bref

- **Entrée** : Cin x H x W
- **Hyperparamètres** :
  - Taille du filtre/kernel : Kh x Kw
  - Nombre de filtres : Cout
  - Padding : P
  - Stride : S
- **Taille du tenseur** : Cout x Cin x Kh x Kw
- **Biais** : Cout
- **Taille de la sortie** : Cout x H' x W', avec
  - $H' = (H - K + 2P) / S + 1$
  - $W' = (W - K + 2P) / S + 1$

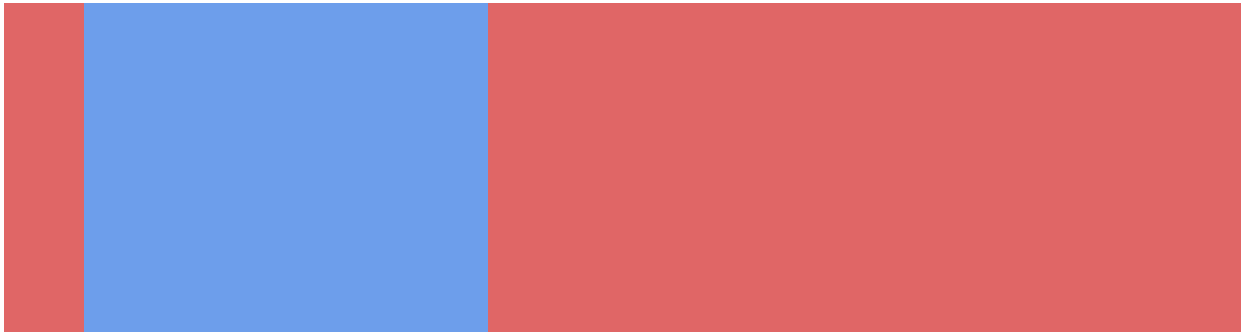
En général :

- Kernel carré : Kh = Kw
- P = (K - 1) / 2
- Cin et Cout des puissances de 2

Autres types de convolutions - 1D

- Nous venons de voir la convolution en 2D, mais elle s'adapte à toutes les dimensions.
  - Exemple 1D : pour du texte

Entrée : Cin x W
Poids : Cout x Cin x K



Autres types de convolutions - 3D

- Nous venons de voir la convolution en 2D, mais elle s'adapte à toutes les dimensions.
  - Exemple 3D : pour de la vidéo

Entrée : Cin x H x W x D
Poids : Cout x Cin x K x K x K

En Pytorch

Conv2d

CLASS torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

En Pytorch

Conv1d

CLASS torch.nn.Conv1d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

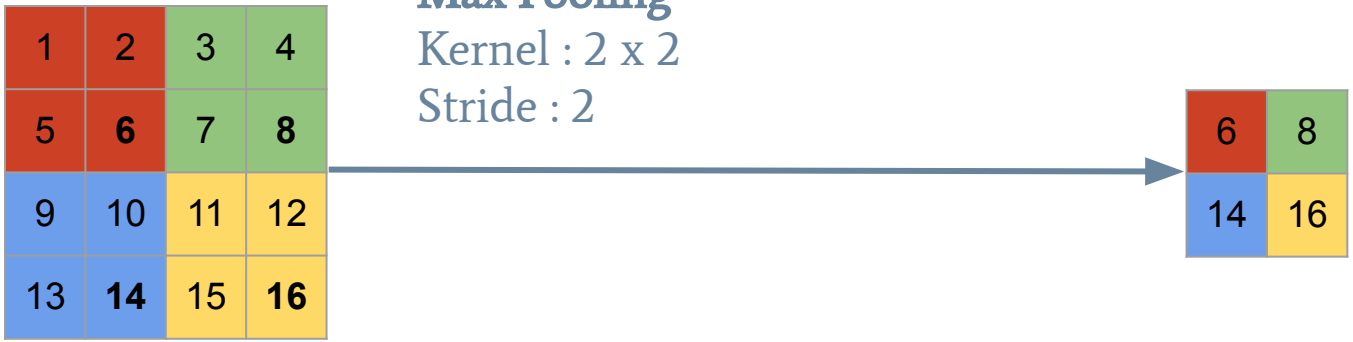
Conv3d

CLASS torch.nn.Conv3d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*) [SOURCE]

Pooling : Une autre manière de faire du downsampling

- **Idée** : On applique un kernel sans paramètre pour réduire la taille de l'entrée
  - Exemples:
    - Max Pooling : prend le maximum sur le kernel
    - Average Pooling : prend la moyenne sur le kernel
- **Hyperparamètres** :
  - Taille du kernel
  - Stride
  - Le type de pooling
- Pas de paramètres à apprendre !

Max Pooling

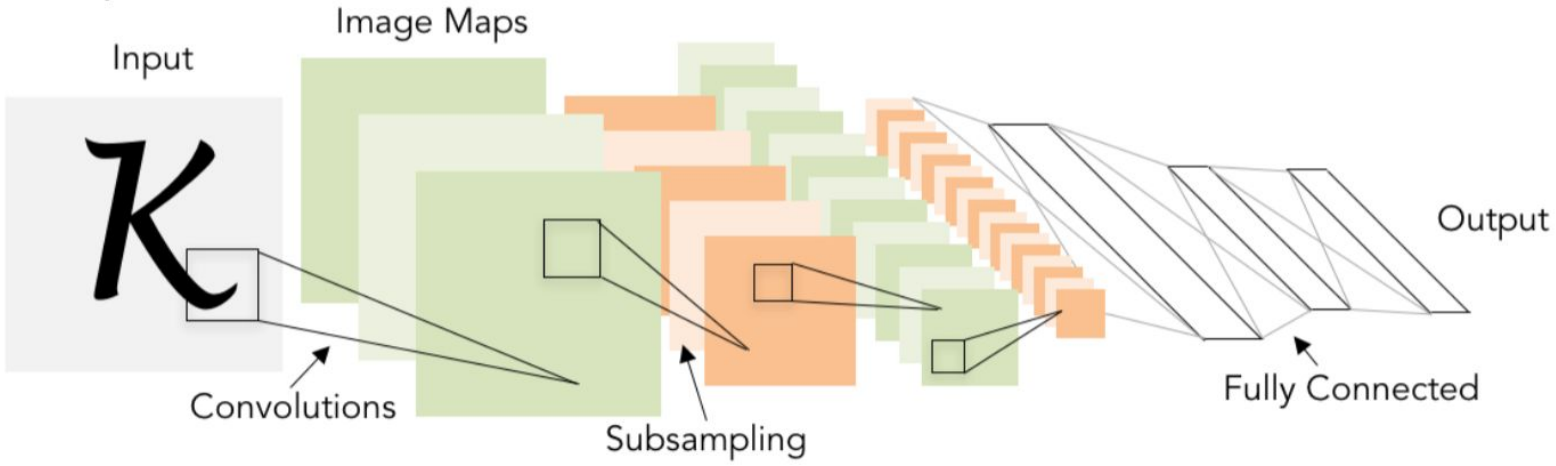


Introduction **d'invariants** de translation dans le réseau

Architecture avec des réseaux convolutifs

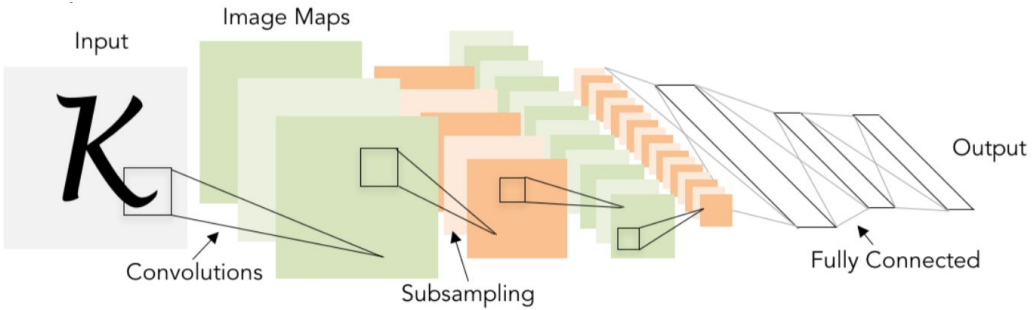
En général : [Conv, ReLU, Pool] x N, flatten, [MLP, ReLU] x N, MLP

Exemple : LeNet-5 (Lecun et al, “Gradient-based learning applied to document recognition”, 1998)



LeNet5

Couche	Taille Sortie	Taille Poids
Entrée	1 x 28 x 28	
Conv (Cout=20, K=5, P=2, S=1)	20 x 28 x 28	20 x 1 x 5 x 5
ReLU	20 x 28 x 28	
MaxPool(K=2, S=2)	20 x 14 x 14	
Conv (Cout=50, K=5, P=2, S=1)	50 x 14 x 14	50 x 20 x 5 x 5
ReLU	50 x 14 x 14	
MaxPool(K=2, S=2)	50 x 7 x 7	
Flatten	2450	
Linear (2450 -> 500)	500	2450 x 500
ReLU	500	
Linear (500 -> 10)	10	500 x 10



En avançant dans le réseau :

- La taille des “images” diminue
- Le nombre de channels augmente (pour préserver le “volume”)

Résumé

- La couche convolutive
  - Kernel/filtre
  - Padding
  - Stride
  - Non-linéarité
  - Pooling
- Un exemple de réseau complet