# Réseaux de neurones convolutifs (CNNs)

Jérémie Cabessa

Laboratoire DAVID, UVSQ

# Réseaux de neurones convolutifs (CNNs)

- Les **réseaux de neurones convolutifs (CNNs)** sont un type de réseaux de neurones particulièrement adaptés pour le traitement d'images et de vidéos (computer vision).
- Ils possèdent toutefois d'autres types d'applications: systèmes de recommandation, le traitement du langage naturel, etc.
- Leur fonctionnement est inspiré par les processus biologiques: cortex visuel des animaux.

# Réseaux de neurones convolutifs (CNNs)

- Les **réseaux de neurones convolutifs (CNNs)** sont un type de réseaux de neurones particulièrement adaptés pour le traitement d'images et de vidéos (computer vision).
- Ils possèdent toutefois d'autres types d'applications: systèmes de recommandation, le traitement du langage naturel, etc.
- Leur fonctionnement est inspiré par les processus biologiques: cortex visuel des animaux.

# Réseaux de neurones convolutifs (CNNs)

- Les **réseaux de neurones convolutifs (CNNs)** sont un type de réseaux de neurones particulièrement adaptés pour le traitement d'images et de vidéos (computer vision).
- Ils possèdent toutefois d'autres types d'applications: systèmes de recommandation, le traitement du langage naturel, etc.
- Leur fonctionnement est inspiré par les processus biologiques: cortex visuel des animaux.

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - DeNet: [Ranzato et al., 2013]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - DeNet: [Ranzato et al., 2013]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - U-Net: [Ronneberger et al., 2015]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - U-Net: [Ronneberger et al., 2015]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - U-Net: [Ronneberger et al., 2015]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - U-Net: [Ronneberger et al., 2015]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - U-Net: [Ronneberger et al., 2015]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - U-Net: [Ronneberger et al., 2015]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Réseaux de neurones convolutifs (CNNs)

- Articles fondateurs sur les CNNs:
  - [Fukushima, 1980, Fukushima, 1988]
  - [LeCun et al., 1998]
- Articles qui ont contribués à de grandes avancées dans le domaine des CNNs:
  - AlexNet: [Krizhevsky et al., 2012]
  - U-Net: [Ronneberger et al., 2015]
  - VGGNet: [Simonyan and Zisserman, 2015]
  - ResNet: [He et al., 2016]
- Par la suite, le domaine a explosé...

# Convolutions

- Les signaux de grande dimension (images, vidéos, audios) posent problème lorsqu'ils sont traités comme des vecteurs non structurés (flattened).
- En effet, une couche linéaire prenant en entrée une image RGB de $256 \times 256$ pixels, et produisant en sortie une image de même taille nécessite $(256 \cdot 256 \cdot 3)^2 \simeq 3.87 \times 10^{10}$ paramètres.
- Empreinte mémoire correspondante de $\sim 150$ Go...

# Convolutions

- Les signaux de grande dimension (images, vidéos, audios) posent problème lorsqu'ils sont traités comme des vecteurs non structurés (flattened).
- En effet, une couche linéaire prenant en entrée une image RGB de 256 x 256 pixels, et produisant en sortie une image de même taille nécessite $(256 \cdot 256 \cdot 3)^2 \simeq 3.87 \times 10^{10}$ paramètres.
- Empreinte mémoire correspondante de $\sim 150$ Go...

# Convolutions

- Les signaux de grande dimension (images, vidéos, audios) posent problème lorsqu'ils sont traités comme des vecteurs non structurés (flattened).
- En effet, une couche linéaire prenant en entrée une image RGB de 256 x 256 pixels, et produisant en sortie une image de même taille nécessite $(256 \cdot 256 \cdot 3)^2 \simeq 3.87 \times 10^{10}$ paramètres.
- Empreinte mémoire correspondante de $\sim 150$ Go...

# Résumé : Convolutions et Invariances XXX

- Une telle exigence est incohérente avec l'intuition selon laquelle ces signaux ont une certaine "invariance par translation".
- Une transformation significative en un endroit peut / doit être utilisée partout.
- Une couche de convolution incarne cette idée.
- Elle applique la même transformation linéaire localement, partout.

# Résumé : Convolutions et Invariances XXX

- Une telle exigence est incohérente avec l'intuition selon laquelle ces signaux ont une certaine "invariance par translation".
- Une transformation significative en un endroit peut / doit être utilisée partout.
- Une couche de convolution incarne cette idée.
- Elle applique la même transformation linéaire localement, partout.

# Résumé : Convolutions et Invariances XXX

- Une telle exigence est incohérente avec l'intuition selon laquelle ces signaux ont une certaine "invariance par translation".
- Une transformation significative en un endroit peut / doit être utilisée partout.
- Une couche de convolution incarne cette idée.
- Elle applique la même transformation linéaire localement, partout.

# Résumé : Convolutions et Invariances XXX

- Une telle exigence est incohérente avec l'intuition selon laquelle ces signaux ont une certaine "invariance par translation".
- Une transformation significative en un endroit peut / doit être utilisée partout.
- Une couche de convolution incarne cette idée.
- Elle applique la même transformation linéaire localement, partout.

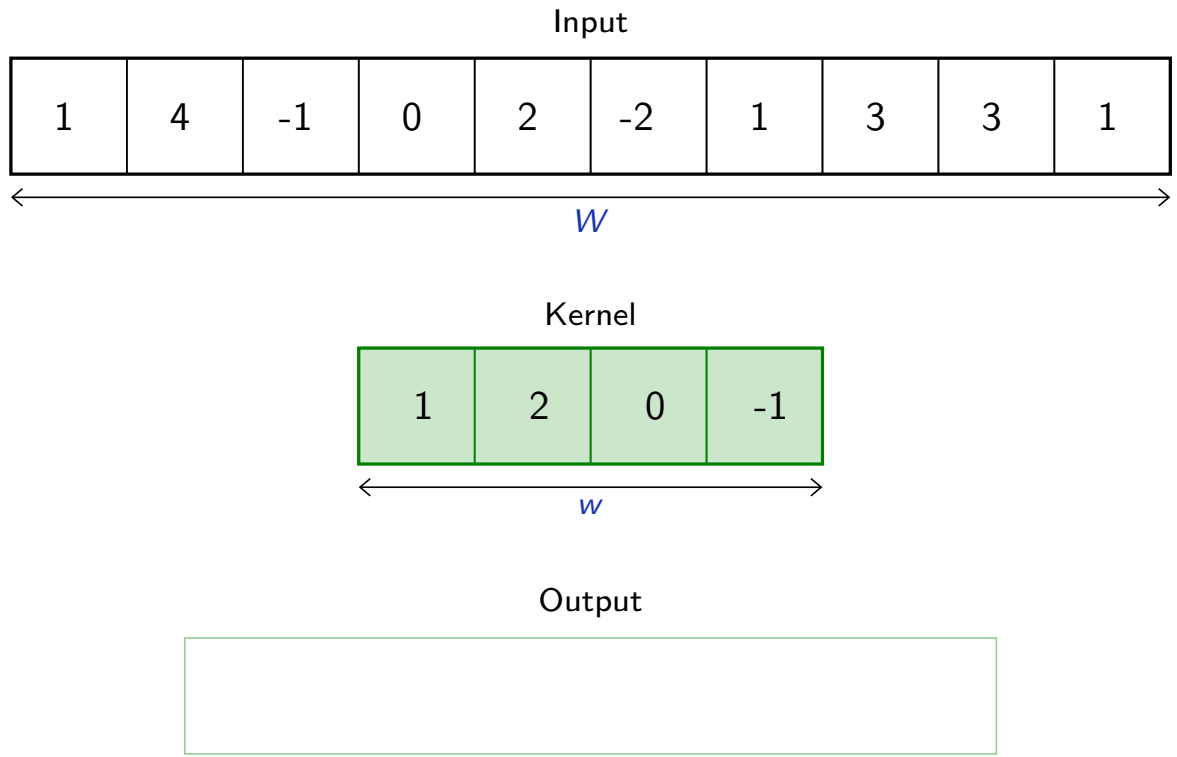
# Convolution: 1D

Input

| 1 | 4 | -1 | 0 | 2 | -2 | 1 | 3 | 3 | 1 |
|---|---|---|---|---|---|---|---|---|---|

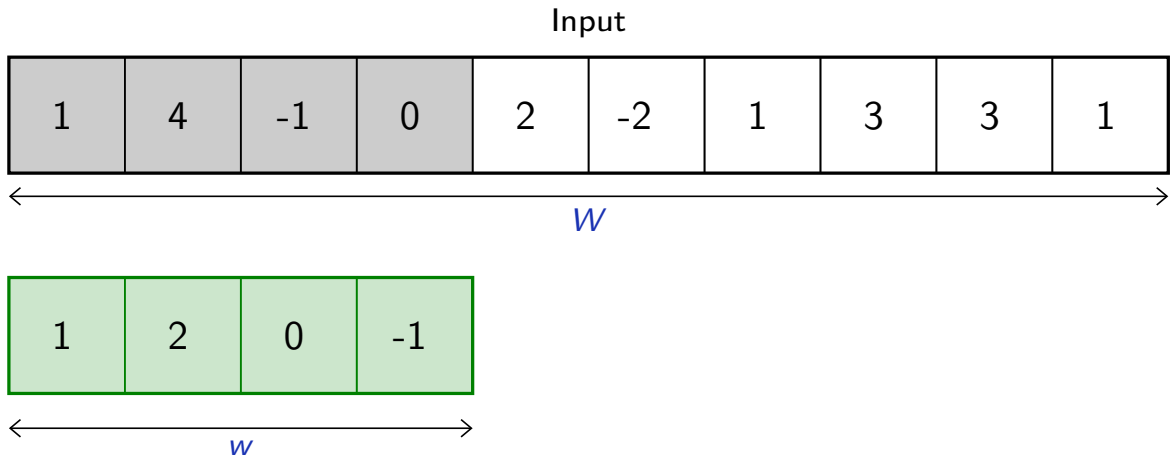
Figures taken from [Fleuret, 2022].

# Convolution: 1D



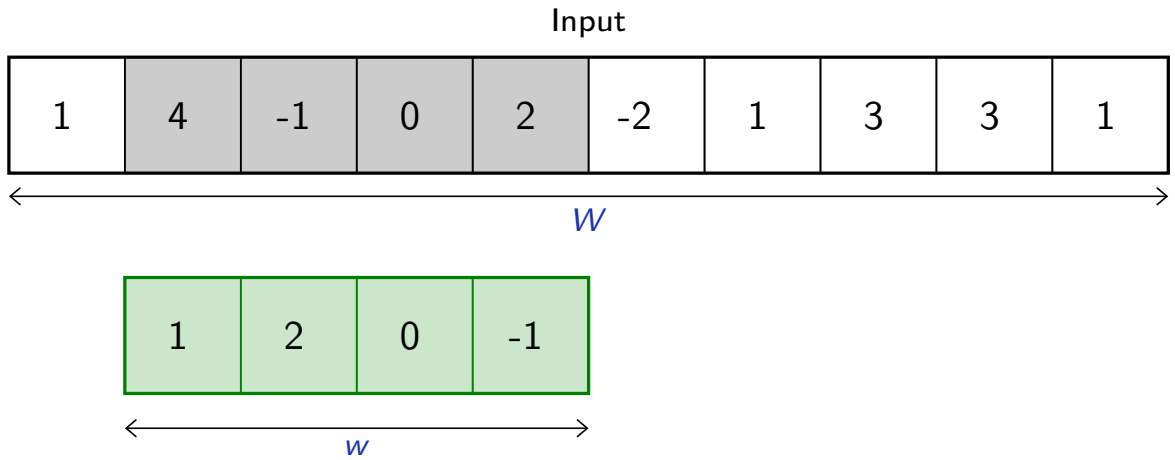

Figures taken from [Fleuret, 2022].

# Convolution: 1D



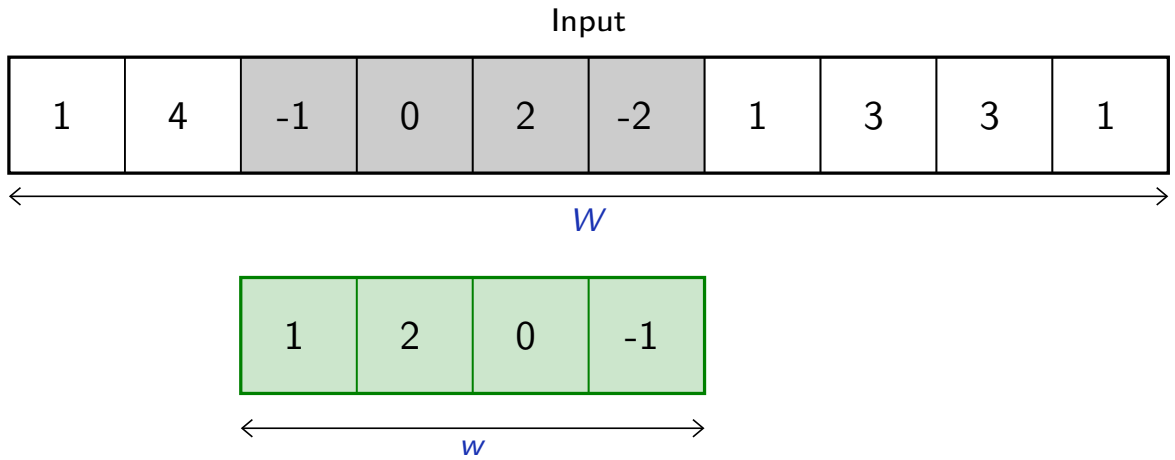

Figures taken from [Fleuret, 2022].

# Convolution: 1D



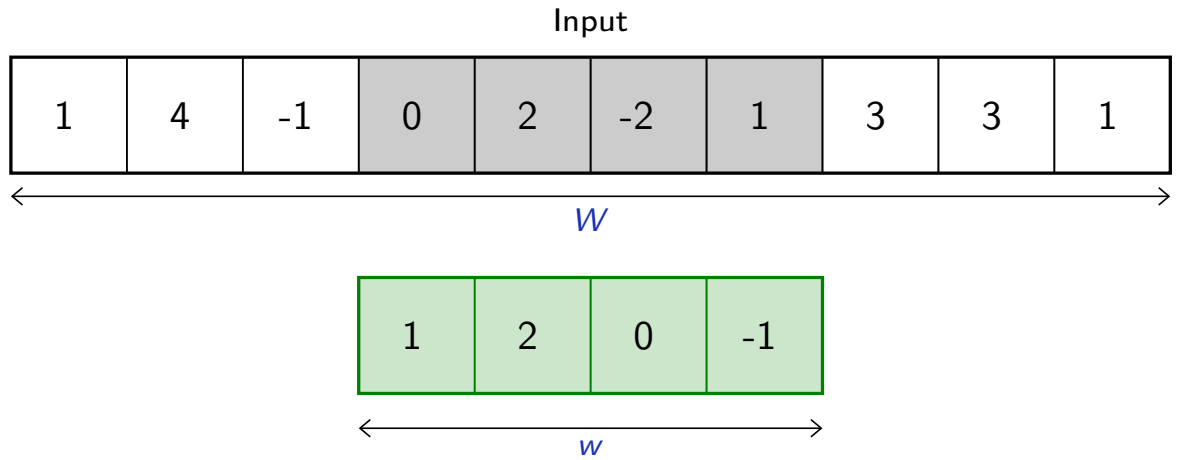

Figures taken from [Fleuret, 2022].

# Convolution: 1D



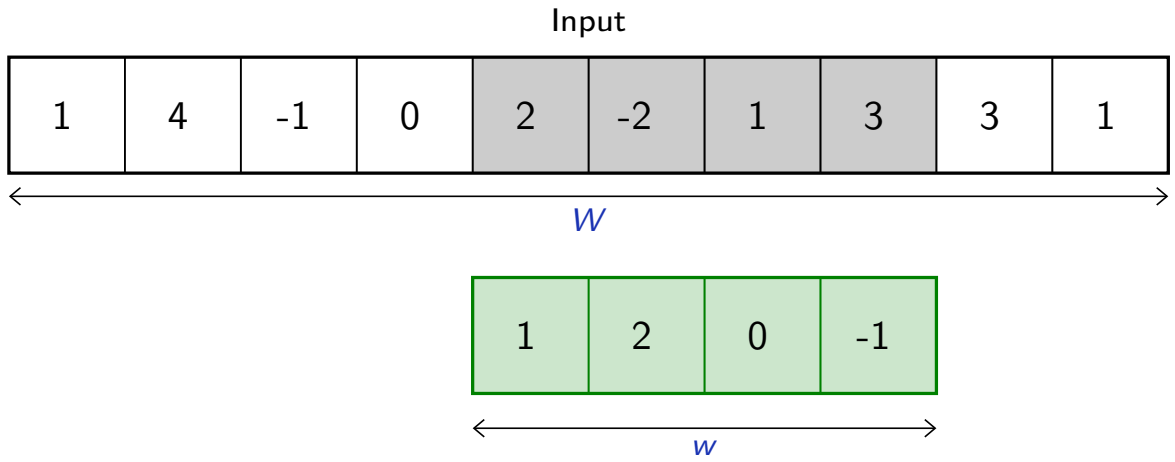

Figures taken from [Fleuret, 2022].

# Convolution: 1D



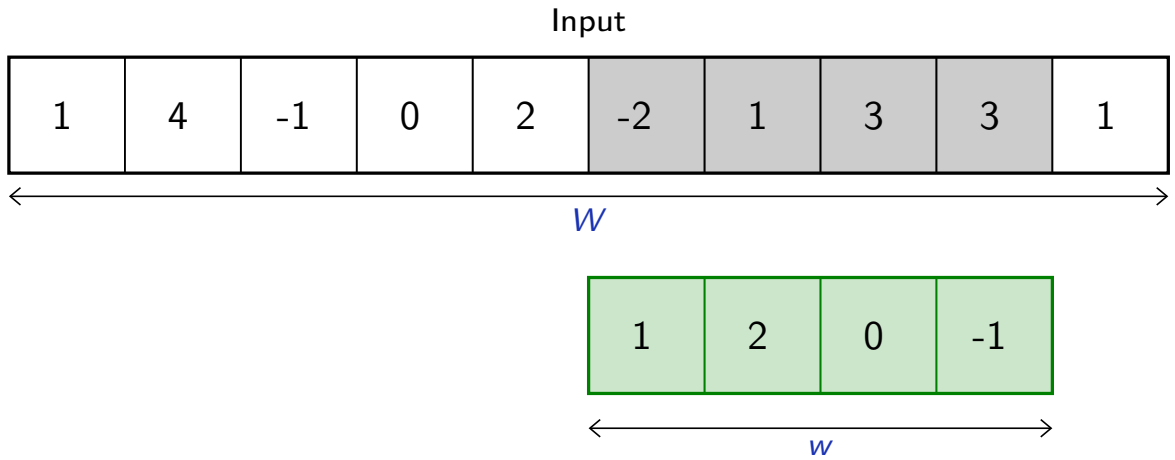

Figures taken from [Fleuret, 2022].

# Convolution: 1D



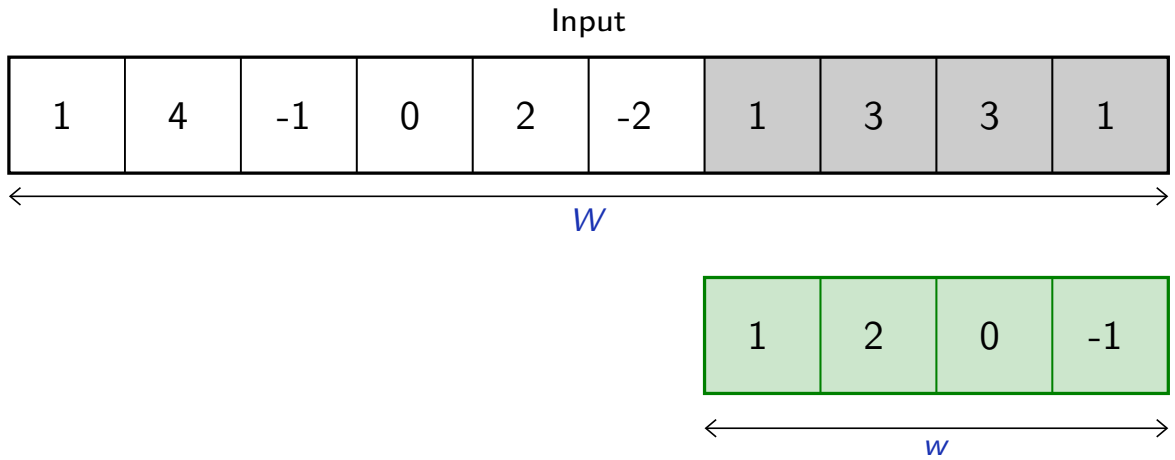

Figures taken from [Fleuret, 2022].

# Convolution: 1D



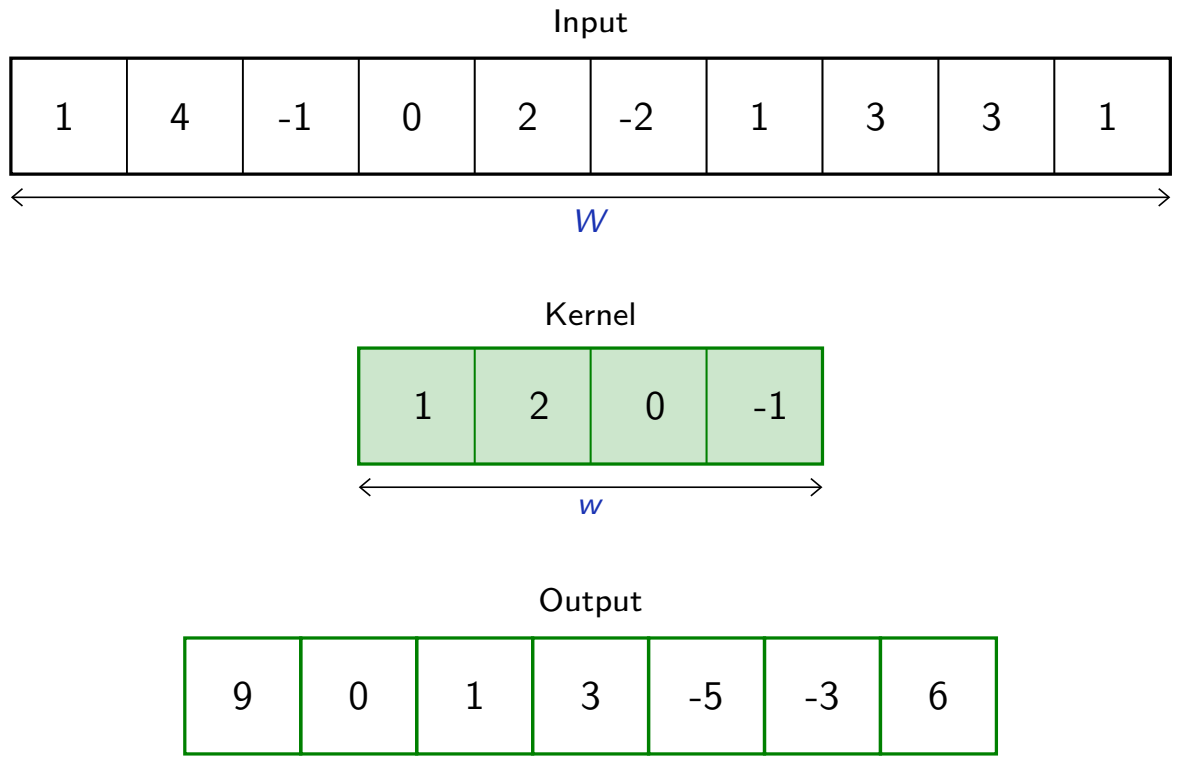

Figures taken from [Fleuret, 2022].

# Convolution: 1D




Figures taken from [Fleuret, 2022].

# Convolution: 1D




Figures taken from [Fleuret, 2022].

# Convolution: 1D

- Étant donné un vecteur d'*input* $\boldsymbol{x}$ de taille $W$

$$\boldsymbol{x} = (x_1, \ldots, x_W)$$

et un *noyau* ou *filtre* (*kernel*, *filter*) $\boldsymbol{k}$ de taille $w$,

$$\boldsymbol{k} = (k_1, \ldots, k_w)$$

la **convolution** $\boldsymbol{x} * \boldsymbol{k}$ est un vecteur de taille $W - w + 1$ dont les composantes sont données par

$$(\boldsymbol{x} * \boldsymbol{k})_i = \boldsymbol{x}_{[i:i+w-1]} \cdot \boldsymbol{k} = \sum_{j=1}^{w} x_{i-1+j} \cdot k_j$$

# Convolution: 1D

- Étant donné un vecteur d'*input* $\boldsymbol{x}$ de taille $W$

$$\boldsymbol{x} = (x_1, \ldots, x_W)$$

et un *noyau* ou *filtre* (*kernel*, *filter*) $\boldsymbol{k}$ de taille $w$,

$$\boldsymbol{k} = (k_1, \ldots, k_w)$$

la **convolution** $\boldsymbol{x} * \boldsymbol{k}$ est un vecteur de taille $W - w + 1$ dont les composantes sont données par

$$(\boldsymbol{x} * \boldsymbol{k})_i = \boldsymbol{x}_{[i:i+w-1]} \cdot \boldsymbol{k} = \sum_{j=1}^{w} x_{i-1+j} \cdot k_j$$

# Convolution: 1D

- Étant donné un vecteur d'*input* $\boldsymbol{x}$ de taille $W$

$$\boldsymbol{x} = (x_1, \ldots, x_W)$$

et un *noyau* ou *filtre* (*kernel*, *filter*) $\boldsymbol{k}$ de taille $w$,

$$\boldsymbol{k} = (k_1, \ldots, k_w)$$

la **convolution** $\boldsymbol{x} * \boldsymbol{k}$ est un vecteur de taille $W - w + 1$ dont les composantes sont données par

$$(\boldsymbol{x} * \boldsymbol{k})_i = \boldsymbol{x}_{[i:i+w-1]} \cdot \boldsymbol{k} = \sum_{j=1}^{w} x_{i-1+j} \cdot k_j$$

# Convolution: 1D

- Étant donné un vecteur d'*input* $\boldsymbol{x}$ de taille $W$

$$\boldsymbol{x} = (x_1, \ldots, x_W)$$

et un *noyau* ou *filtre* (*kernel*, *filter*) $\boldsymbol{k}$ de taille $w$,

$$\boldsymbol{k} = (k_1, \ldots, k_w)$$

la **convolution** $\boldsymbol{x} * \boldsymbol{k}$ est un vecteur de taille $W - w + 1$ dont les composantes sont données par

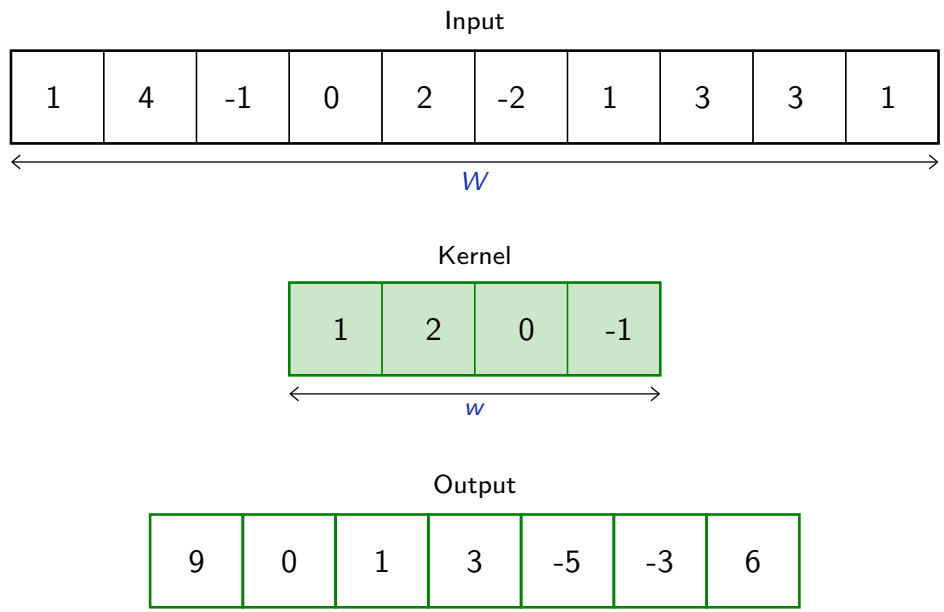
$$(\boldsymbol{x} * \boldsymbol{k})_i = \boldsymbol{x}_{[i:i+w-1]} \cdot \boldsymbol{k} = \sum_{j=1}^{w} x_{i-1+j} \cdot k_j$$

# Convolution: 1D

- Exemple numérique:

$$(1, 2, 3, 4) * (3, 2) = (3 + 4, 6 + 6, 9 + 8) = (7, 12, 17)$$

- Exemple visuel:

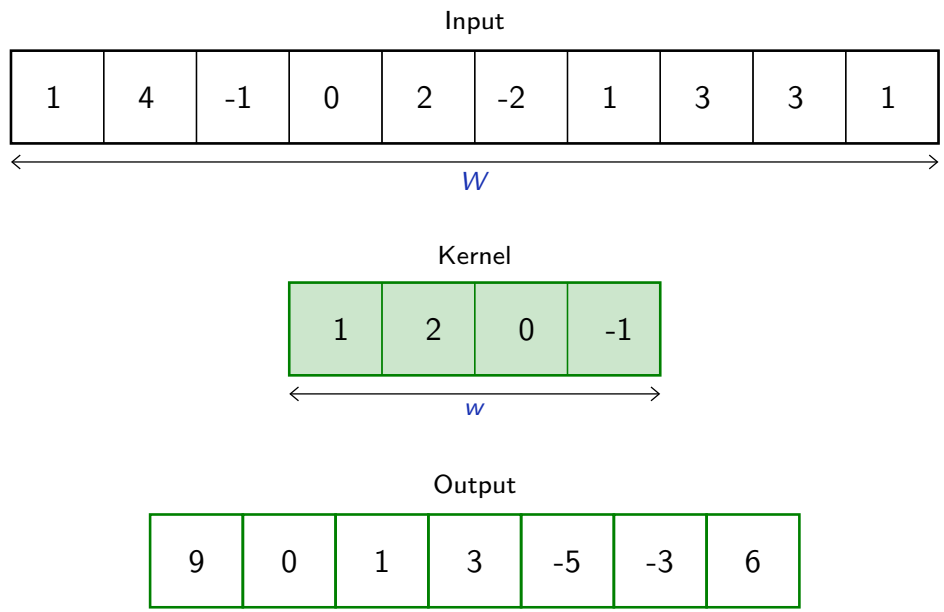


# Convolution: 1D

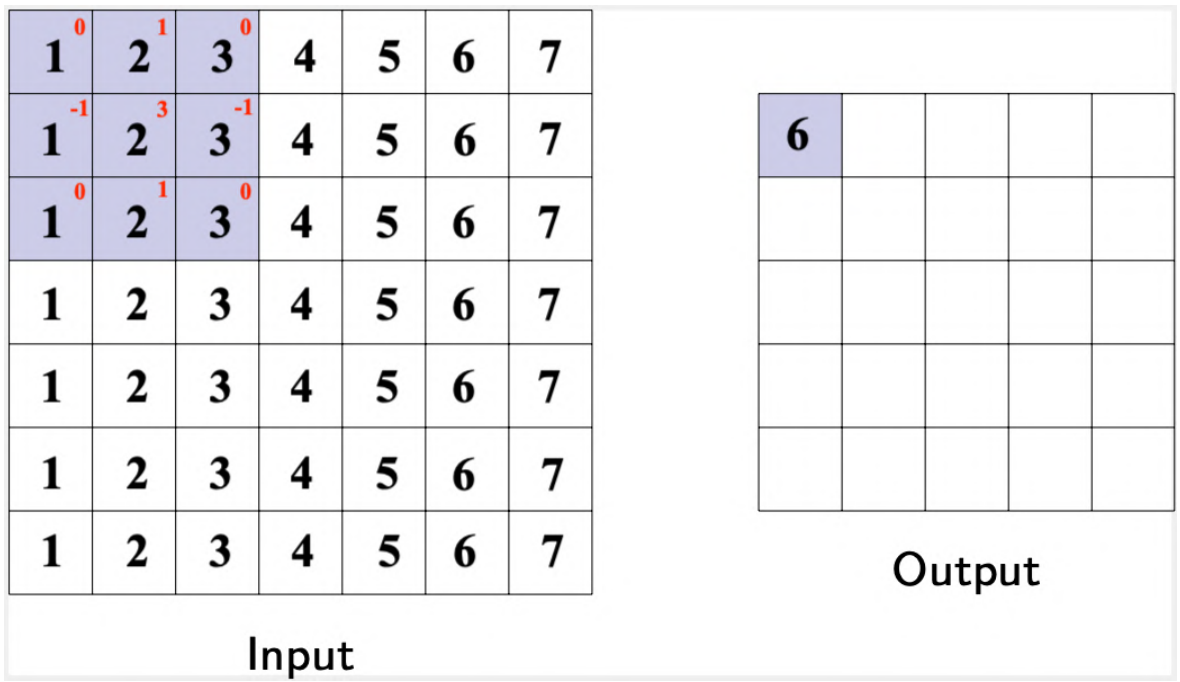
- Exemple numérique:

$$(1, 2, 3, 4) * (3, 2) = (3 + 4, 6 + 6, 9 + 8) = (7, 12, 17)$$

- Exemple visuel:

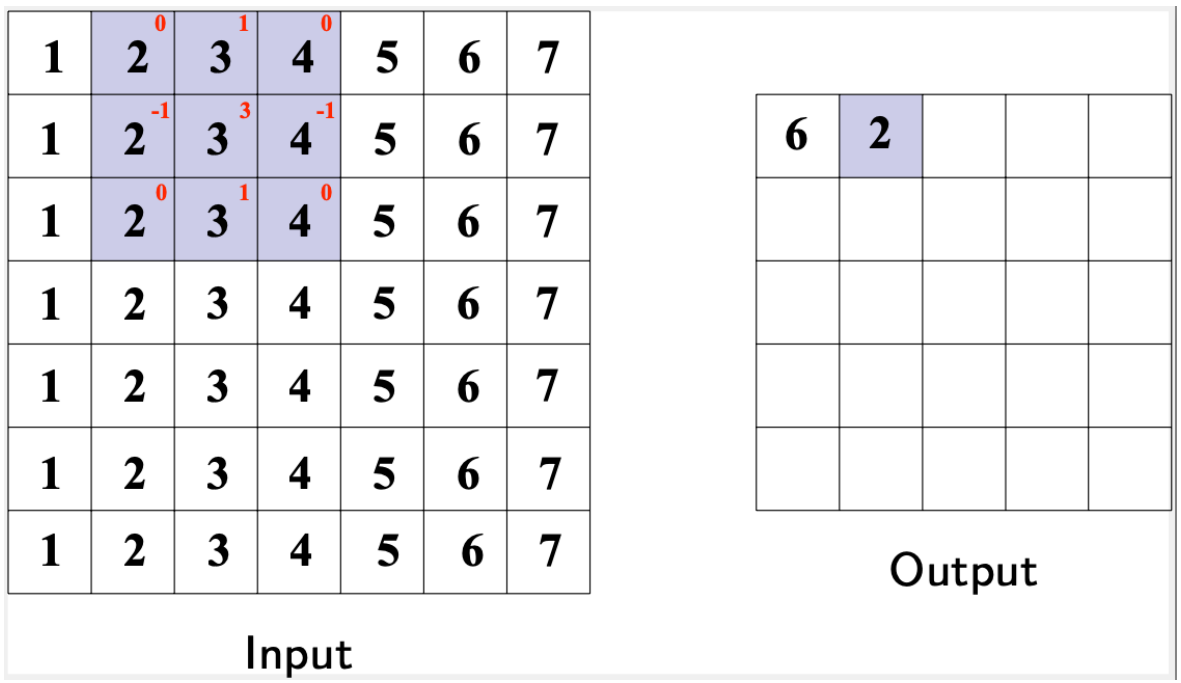


# Convolution: 2D



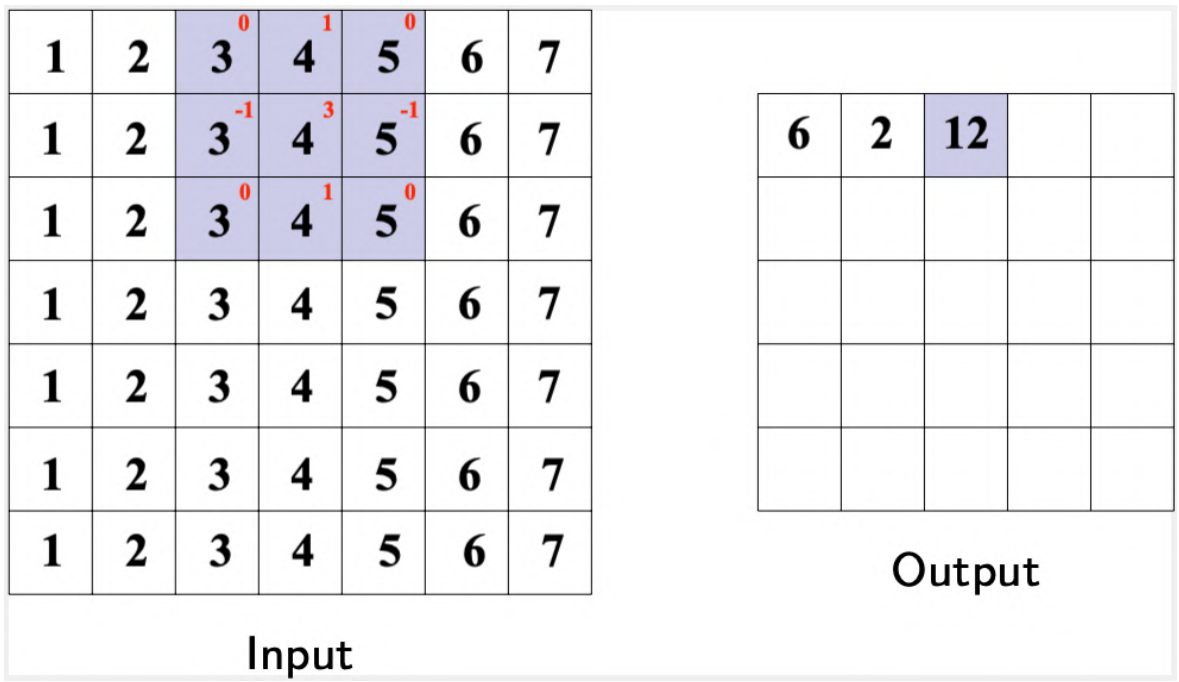

Figures taken from [Fleuret, 2022]

# Convolution: 2D



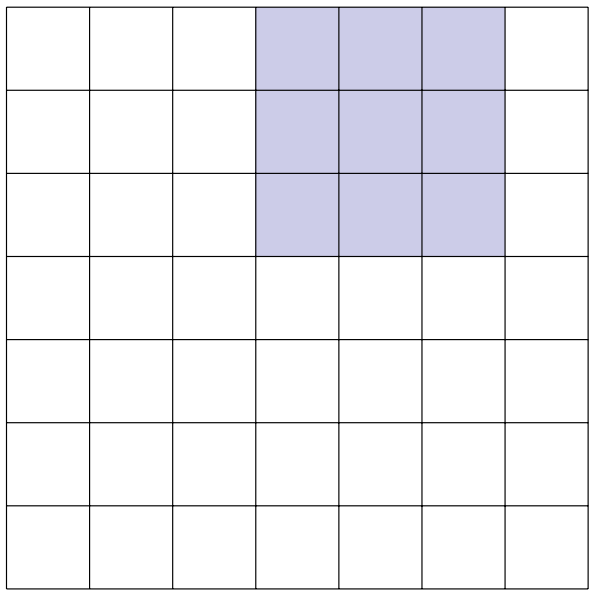

Figures taken from [Fleuret, 2022]

# Convolution: 2D

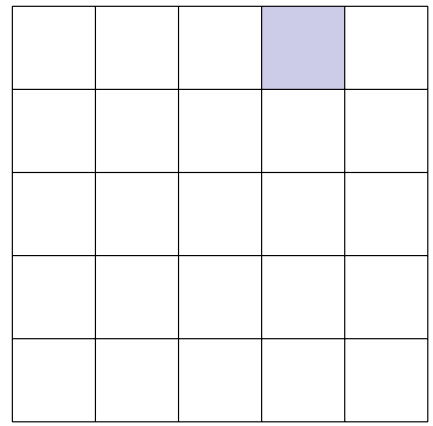



Figures taken from [Fleuret, 2022]

# Convolution: 2D



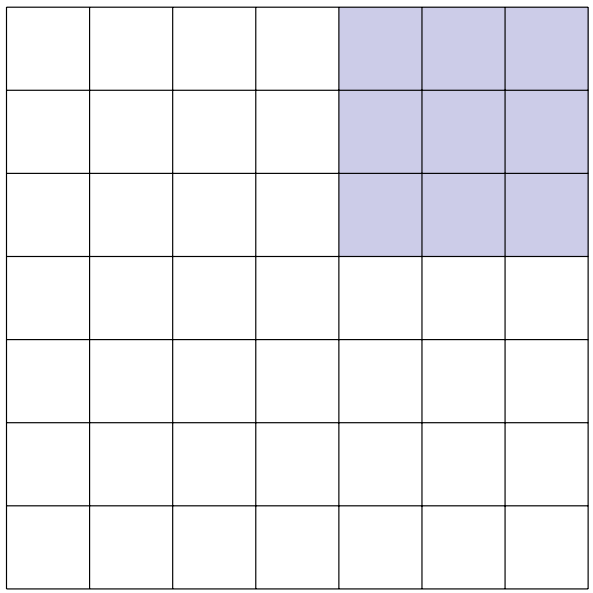
Input



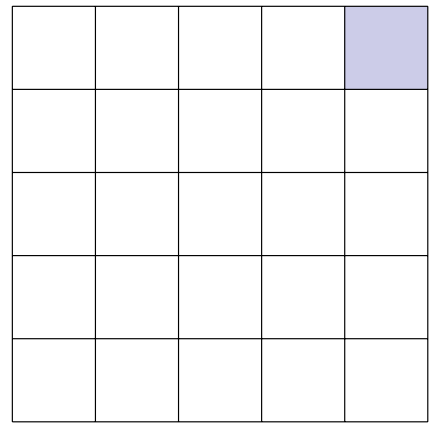
Output

Figures taken from [Fleuret, 2022]

# Convolution: 2D



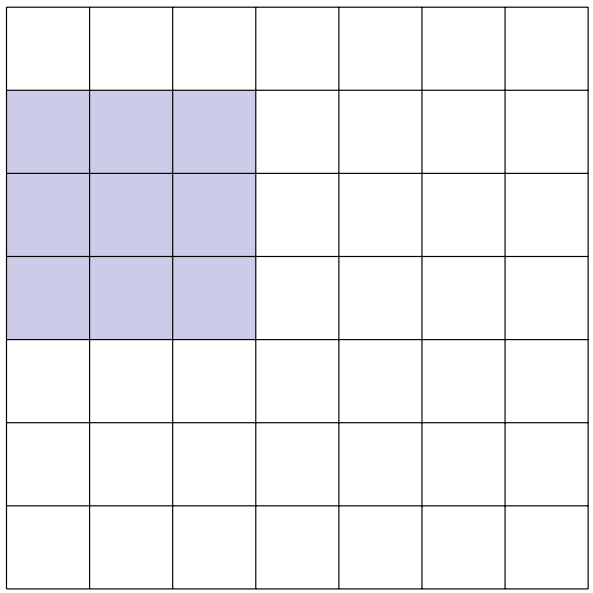
Input



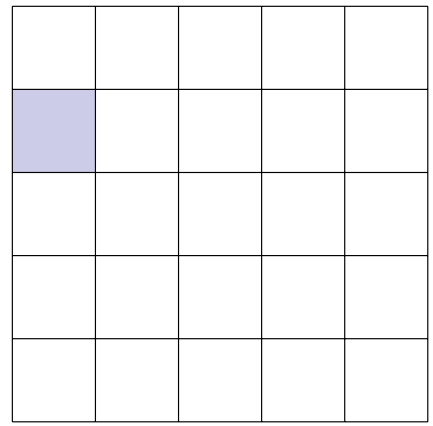
Output

Figures taken from [Fleuret, 2022]

# Convolution: 2D



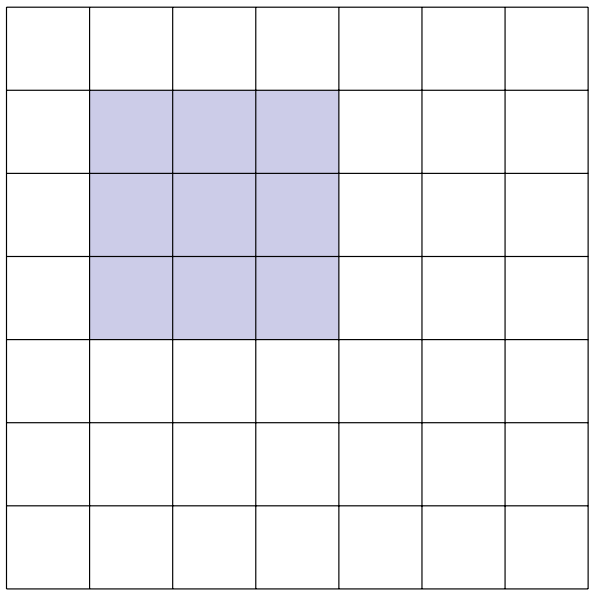
Input



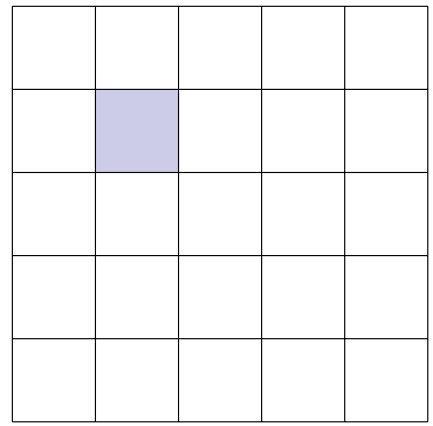
Output

Figures taken from [Fleuret, 2022]

# Convolution: 2D



Input



Output

Figures taken from [Fleuret, 2022]

# Convolution: 2D

- Étant donné une matrice d'*input* $\boldsymbol{X}$ de taille $H \times W$

$$\boldsymbol{X} = (x_{i,j})$$

et un *noyau de convolution* ou *filtre* (*kernel*, *filter*) $\boldsymbol{K}$ de taille $h \times w$,

$$\boldsymbol{K} = (k_{i,j})$$

la **convolution** $\boldsymbol{X} * \boldsymbol{K}$ est une matrice de taille $(H - h + 1) \times (W - w + 1)$ dont les composantes sont données par

$$\begin{aligned}(\boldsymbol{X} * \boldsymbol{K})_{i,j} &= \boldsymbol{X}_{[i:i+h-1,j:j+w-1]} \odot \boldsymbol{K} \\ &= \sum_{k=1}^{h} \sum_{l=1}^{w} x_{i-1+k,j-1+l} \cdot u_{k,l}\end{aligned}$$

# Convolution: 2D

- Étant donné une matrice d'*input* $\boldsymbol{X}$ de taille $H \times W$

$$\boldsymbol{X} = (x_{i,j})$$

et un *noyau de convolution* ou *filtre* (*kernel, filter*) $\boldsymbol{K}$ de taille $h \times w$,

$$\boldsymbol{K} = (k_{i,j})$$

la **convolution** $\boldsymbol{X} * \boldsymbol{K}$ est une matrice de taille $(H - h + 1) \times (W - w + 1)$ dont les composantes sont données par

$$\begin{aligned}(\boldsymbol{X} * \boldsymbol{K})_{i,j} &= \boldsymbol{X}_{[i:i+h-1,j:j+w-1]} \odot \boldsymbol{K} \\ &= \sum_{k=1}^{h} \sum_{l=1}^{w} x_{i-1+k,j-1+l} \cdot u_{k,l}\end{aligned}$$

# Convolution: 2D

- Étant donné une matrice d'*input* $\boldsymbol{X}$ de taille $H \times W$

$$\boldsymbol{X} = (x_{i,j})$$

et un *noyau de convolution* ou *filtre* (*kernel, filter*) $\boldsymbol{K}$ de taille $h \times w$,

$$\boldsymbol{K} = (k_{i,j})$$

la **convolution** $\boldsymbol{X} * \boldsymbol{K}$ est une matrice de taille $(H - h + 1) \times (W - w + 1)$ dont les composantes sont données par

$$\begin{aligned}(\boldsymbol{X} * \boldsymbol{K})_{i,j} &= \boldsymbol{X}_{[i:i+h-1,j:j+w-1]} \odot \boldsymbol{K} \\ &= \sum_{k=1}^{h} \sum_{l=1}^{w} x_{i-1+k,j-1+l} \cdot u_{k,l}\end{aligned}$$

# Convolution: 2D

- Étant donné une matrice d'*input* $\boldsymbol{X}$ de taille $H \times W$

$$\boldsymbol{X} = (x_{i,j})$$

et un *noyau de convolution* ou *filtre* (*kernel, filter*) $\boldsymbol{K}$ de taille $h \times w$,

$$\boldsymbol{K} = (k_{i,j})$$

la **convolution** $\boldsymbol{X} * \boldsymbol{K}$ est une matrice de taille $(H - h + 1) \times (W - w + 1)$ dont les composantes sont données par

$$\begin{aligned}(\boldsymbol{X} * \boldsymbol{K})_{i,j} &= \boldsymbol{X}_{[i:i+h-1,j:j+w-1]} \odot \boldsymbol{K} \\ &= \sum_{k=1}^{h} \sum_{l=1}^{w} x_{i-1+k,j-1+l} \cdot u_{k,l}\end{aligned}$$

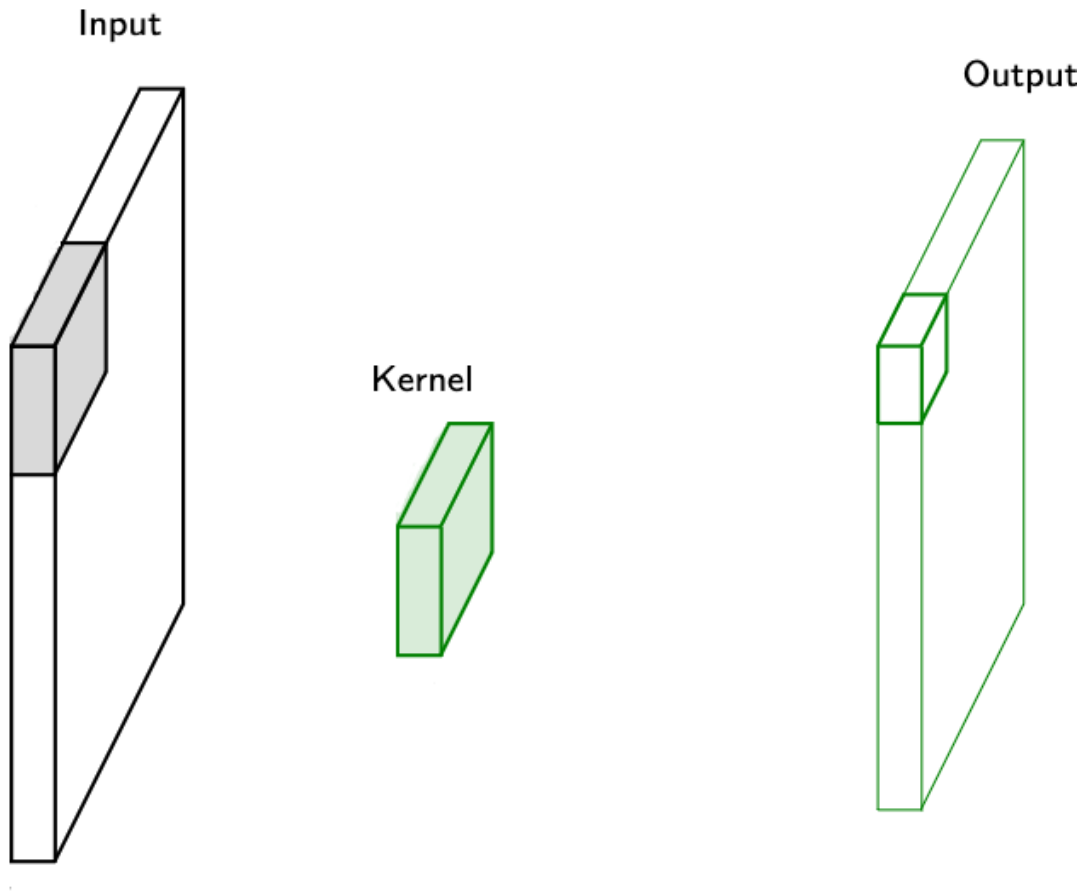
# Convolution: 2D

- Exemples:

  Voir site web: https://deeplizard.com/resource/pavq7noze2

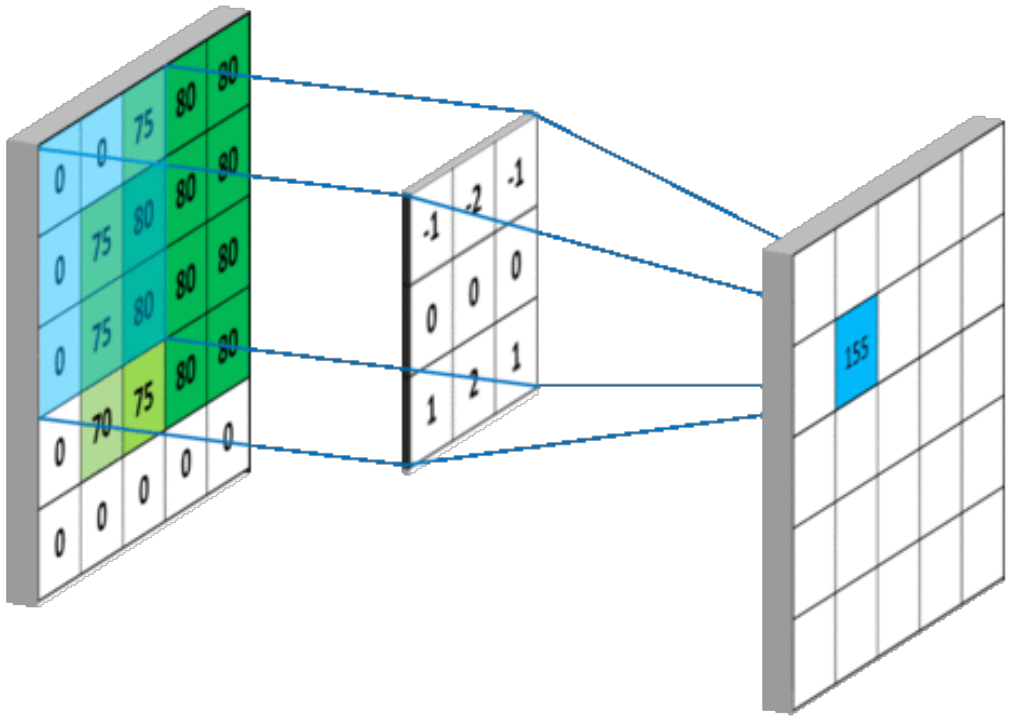
  Tester différents filtres: top/bottom/left/right edge filters...

# Couche convolutive (conv. layer)



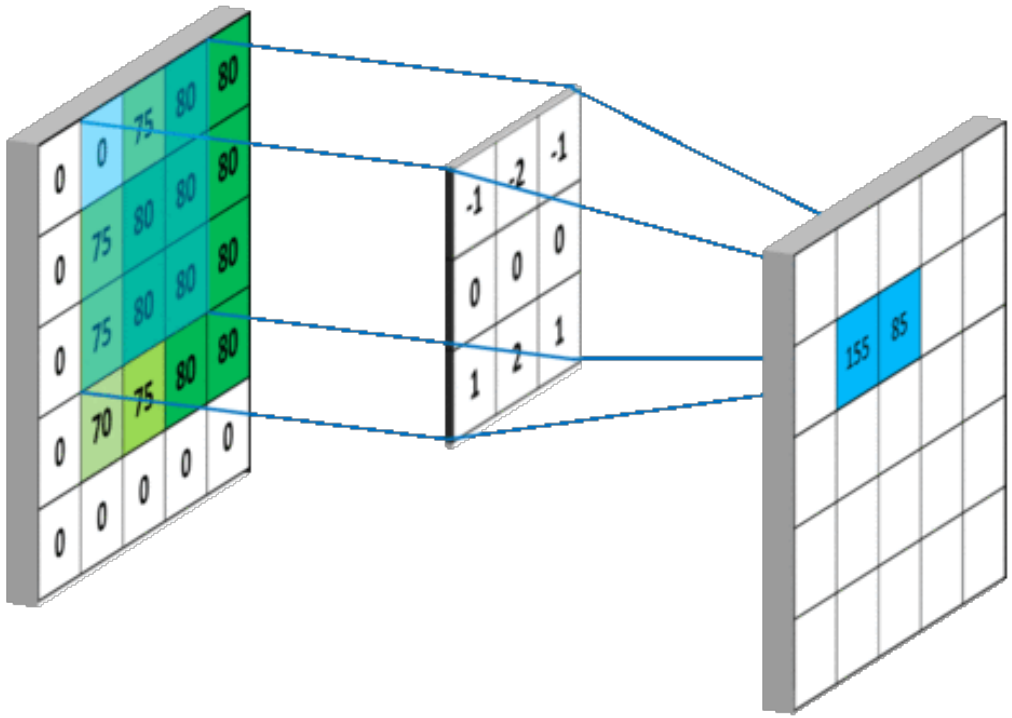

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)



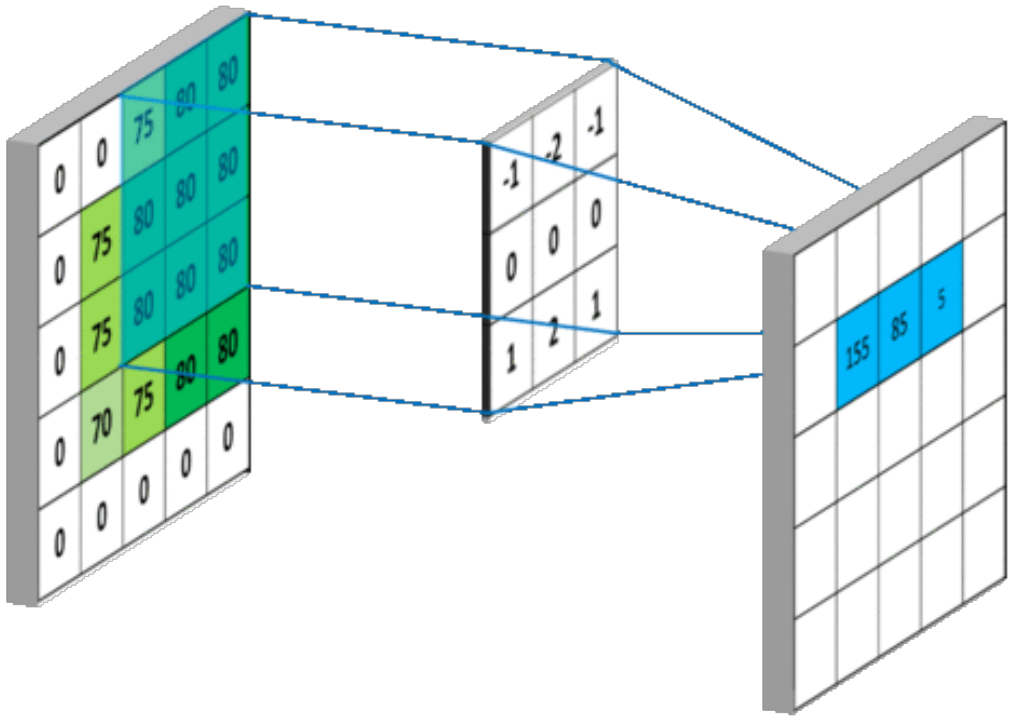

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)



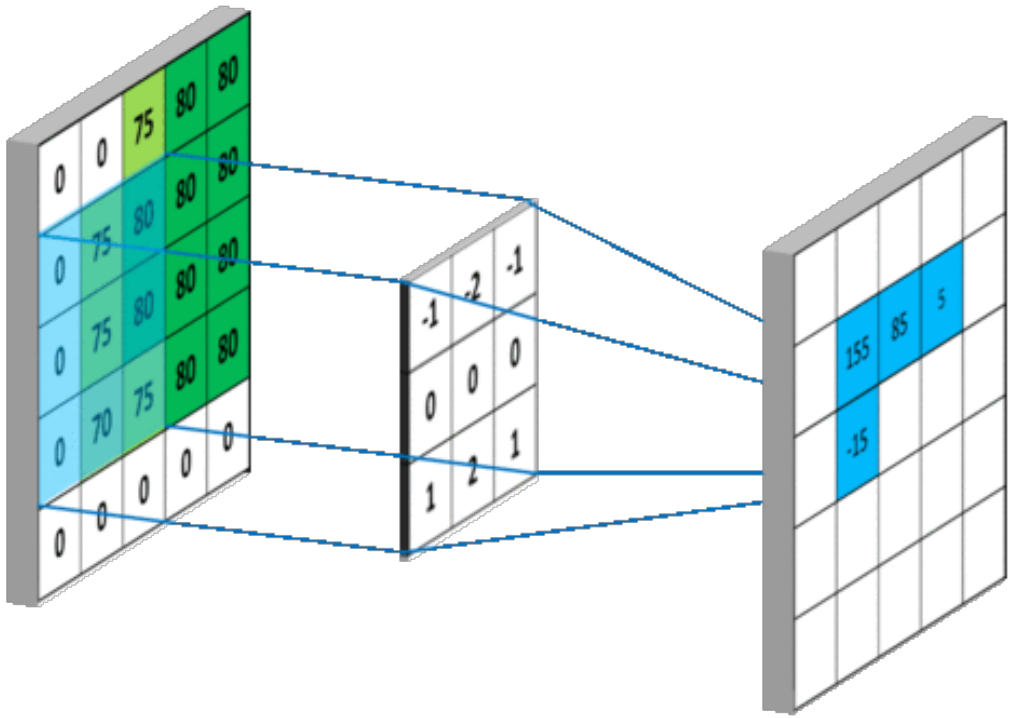

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)



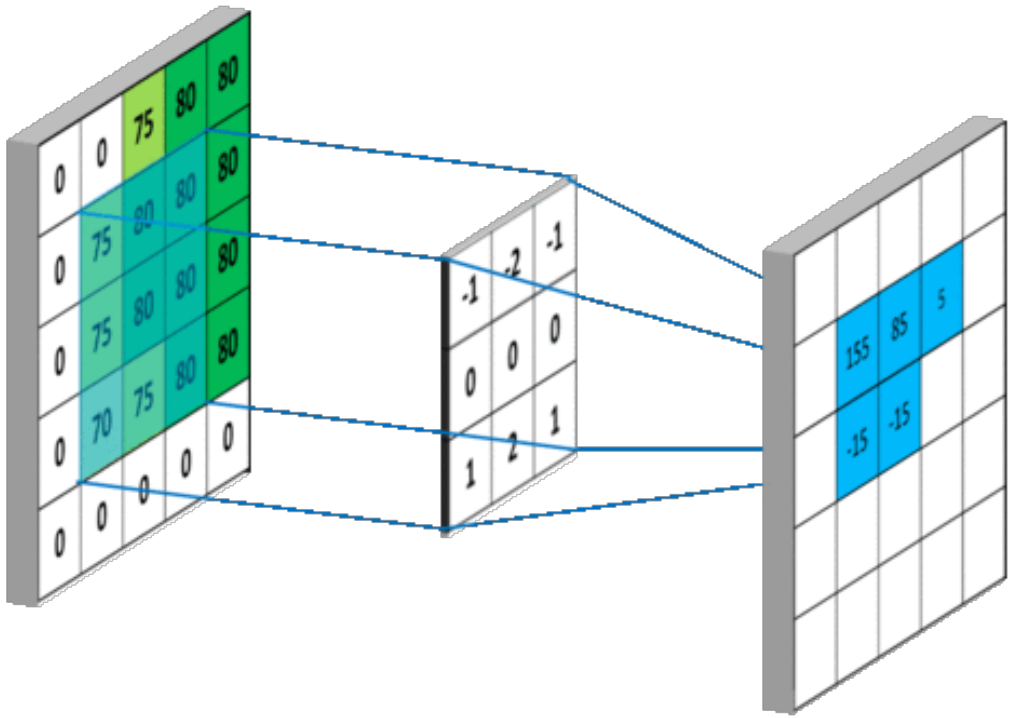

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)



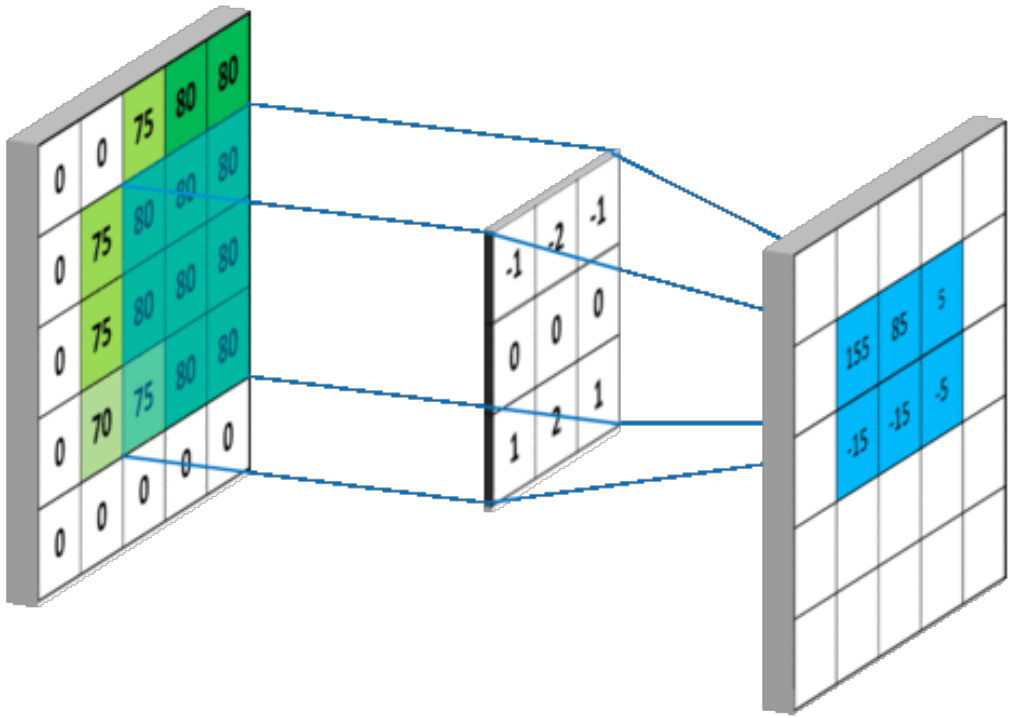

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)



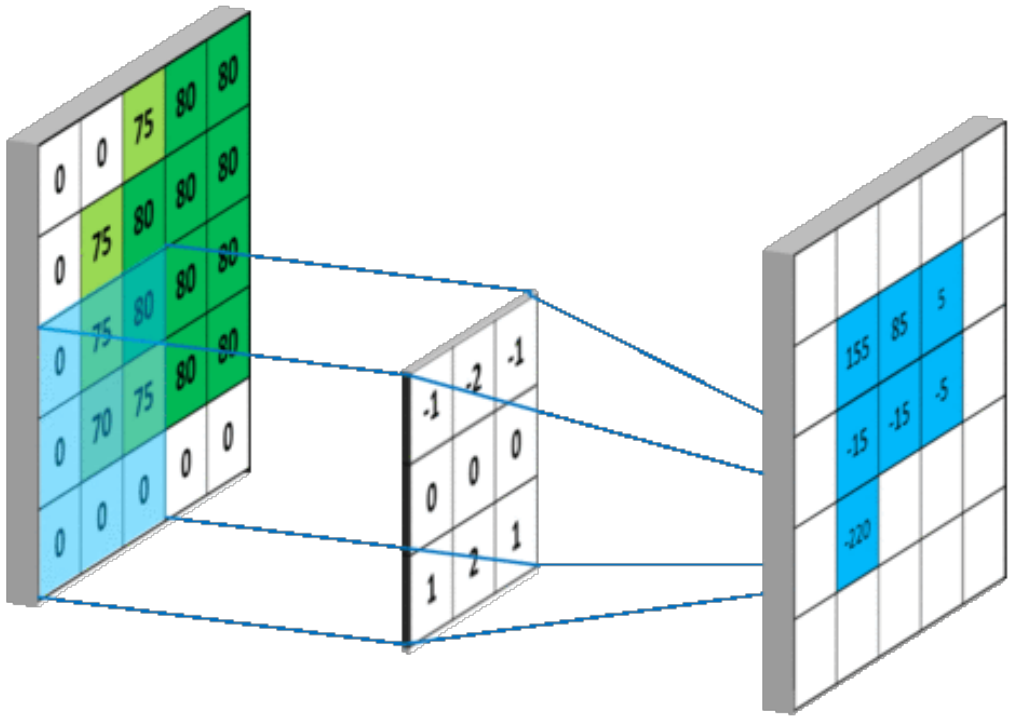

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)



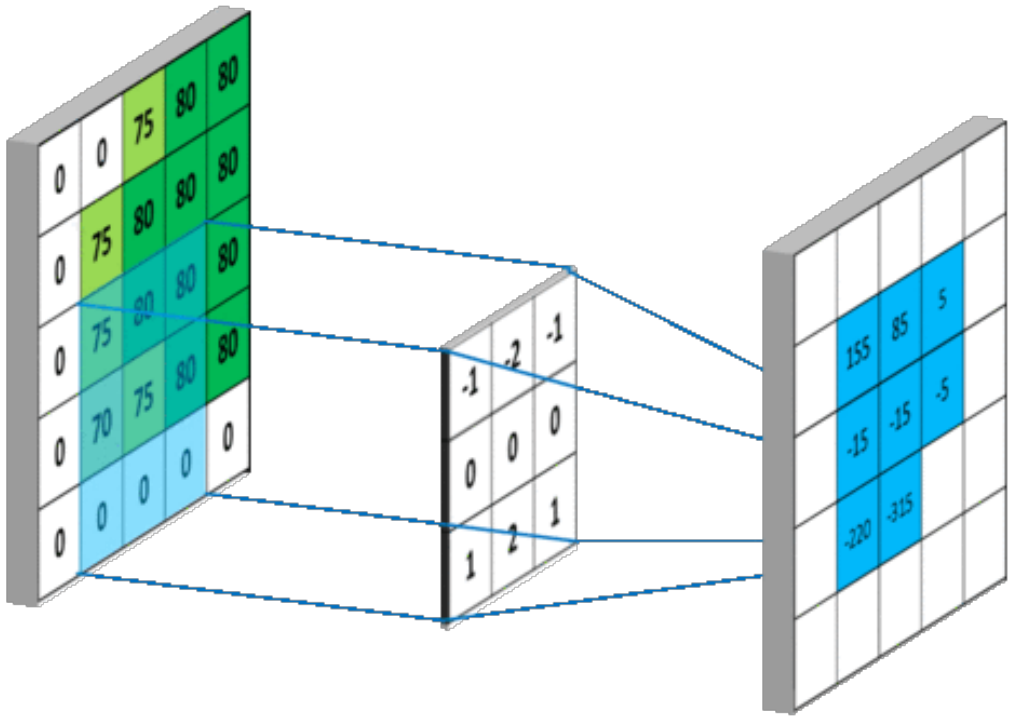

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)



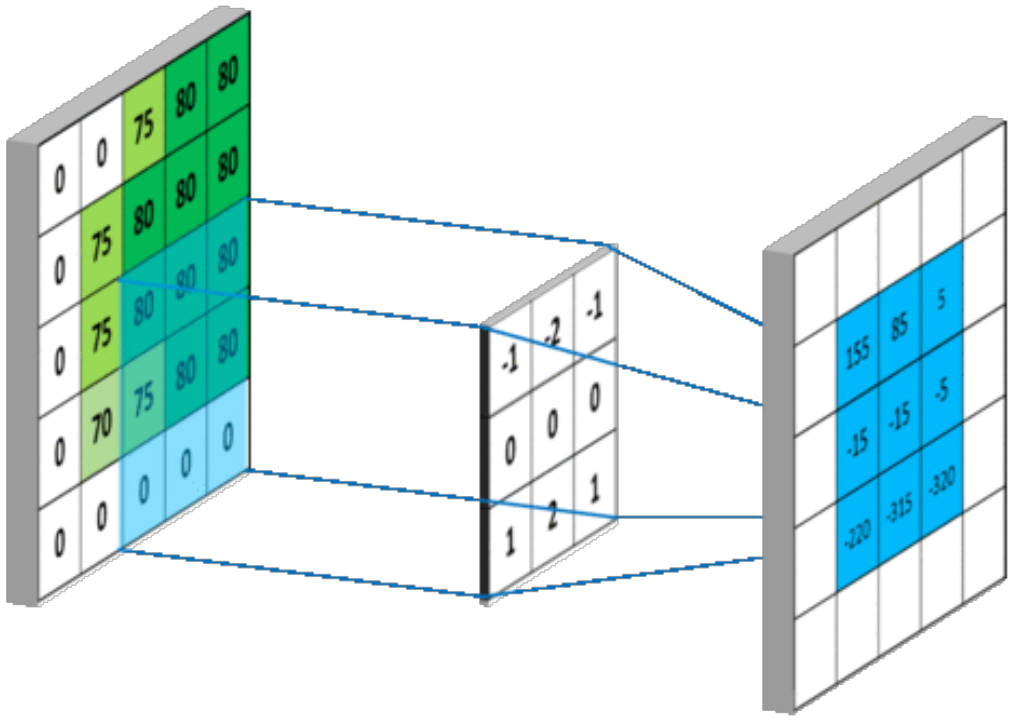

Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

# Couche convolutive (conv. layer)

- L'output de la convolution est appelée *activation map*.
- La partie de l'input qui participe au calcul d'une composante d'output est appelée *receptive field*.
- Une couche convolutive standard est généralement appelée *fully connected layer*.
- Dans une couche convolutive, ce sont les paramètres (valeurs) du noyau $K$ qui sont appris!

# Couche convolutive (conv. layer)

- L'output de la convolution est appelée *activation map*.
- La partie de l'input qui participe au calcul d'une composante d'output est appelée *receptive field*.
- Une couche convolutive standard est généralement appelée *fully connected layer*.
- Dans une couche convolutive, ce sont les paramètres (valeurs) du noyau $K$ qui sont appris!

# Couche convolutive (conv. layer)

- L'output de la convolution est appelée *activation map*.
- La partie de l'input qui participe au calcul d'une composante d'output est appelée *receptive field*.
- Une couche convolutive standard est généralement appelée *fully connected layer*.
- Dans une couche convolutive, ce sont les paramètres (valeurs) du noyau $K$ qui sont appris!

# Couche convolutive (conv. layer)

- L'output de la convolution est appelée *activation map*.
- La partie de l'input qui participe au calcul d'une composante d'output est appelée *receptive field*.
- Une couche convolutive standard est généralement appelée *fully connected layer*.
- **Dans une couche convolutive, ce sont les paramètres (valeurs) du noyau $K$ qui sont appris!**

# Couche convolutive (conv. layer)

- En pratique, les images en couleurs sont représentées par des tenseur RGB de profondeur $C = 3$.
- Chacune des matrices "rouges" (R), "verts" (G), et "bleus" (B) est appelée un *canal (channel)*.
- Ainsi, la convolution sera appliquée entre une input 3D $\boldsymbol{X}$ et un noyau 3D $\boldsymbol{K}$.

# Couche convolutive (conv. layer)

- En pratique, les images en couleurs sont représentées par des tenseur RGB de profondeur $C = 3$.
- Chacune des matrices "rouges" (R), "verts" (G), et "bleus" (B) est appelée un *canal (channel)*.
- Ainsi, la convolution sera appliquée entre une input 3D $\boldsymbol{X}$ et un noyau 3D $\boldsymbol{K}$.

# Couche convolutive (conv. layer)

- En pratique, les images en couleurs sont représentées par des tenseur RGB de profondeur $C = 3$.
- Chacune des matrices "rouges" (R), "verts" (G), et "bleus" (B) est appelée un *canal (channel)*.
- Ainsi, la convolution sera appliquée entre une input 3D $\boldsymbol{X}$ et un noyau 3D $\boldsymbol{K}$.

# Couche convolutive (conv. layer)

- La convolution entre une input $\boldsymbol{X}$ de dimension $H \times W \times C$ et un noyau $\boldsymbol{K}$ de dimension $h \times w \times C$ donne une output

$$\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{K}$$

  de dimension $(H - h + 1) \times (W - w + 1)$.

- L'opération de convolution est généralisée à tous les canaux de manière "naturelle".
- L'output est une matrice de profondeur $1$ et non $C$. Si on veut une output de profondeur $> 1$, on rajoute des noyaux (cf. slide suivant).

# Couche convolutive (conv. layer)

- La convolution entre une input $\boldsymbol{X}$ de dimension $H \times W \times C$ et un noyau $\boldsymbol{K}$ de dimension $h \times w \times C$ donne une output

$$\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{K}$$

  de dimension $(H - h + 1) \times (W - w + 1)$.

- L'opération de convolution est généralisée à tous les canaux de manière "naturelle".
- L'output est une matrice de profondeur $1$ et non $C$. Si on veut une output de profondeur $> 1$, on rajoute des noyaux (cf. slide suivant).

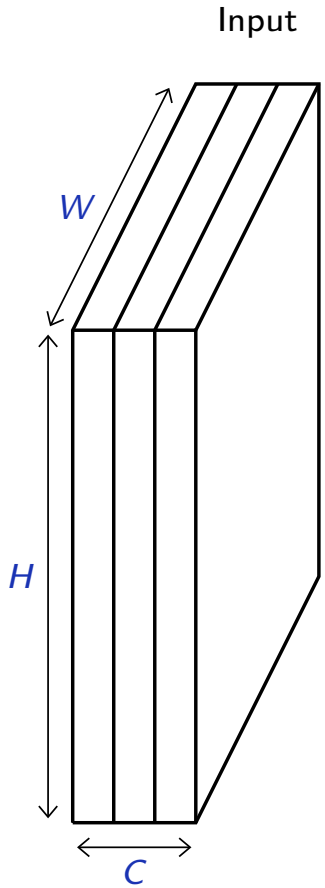
# Couche convolutive (conv. layer)

- La convolution entre une input $\boldsymbol{X}$ de dimension $H \times W \times C$ et un noyau $\boldsymbol{K}$ de dimension $h \times w \times C$ donne une output

$$\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{K}$$

  de dimension $(H - h + 1) \times (W - w + 1)$.
- L'opération de convolution est généralisée à tous les canaux de manière "naturelle".
- L'output est une matrice de profondeur $1$ et non $C$. Si on veut une output de profondeur $> 1$, on rajoute des noyaux (cf. slide suivant).

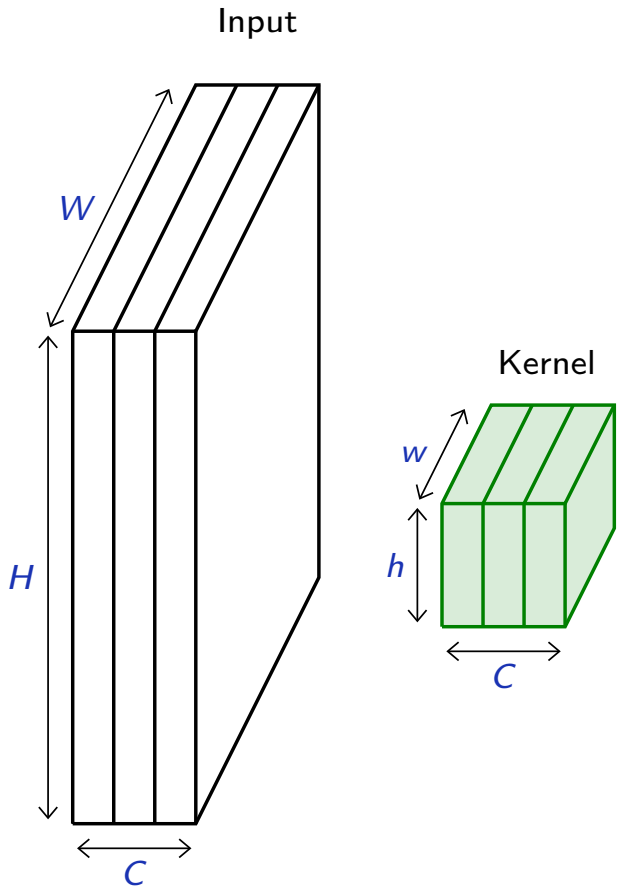
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

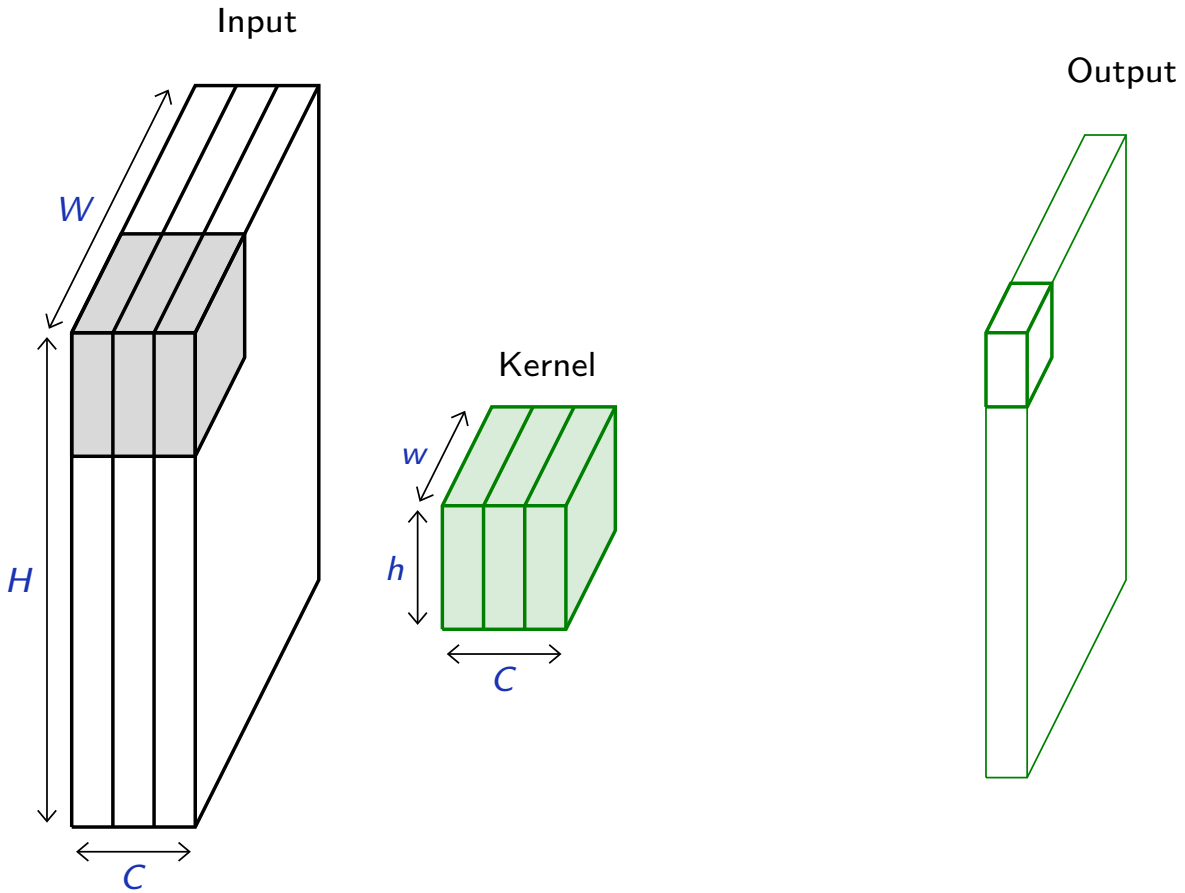
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

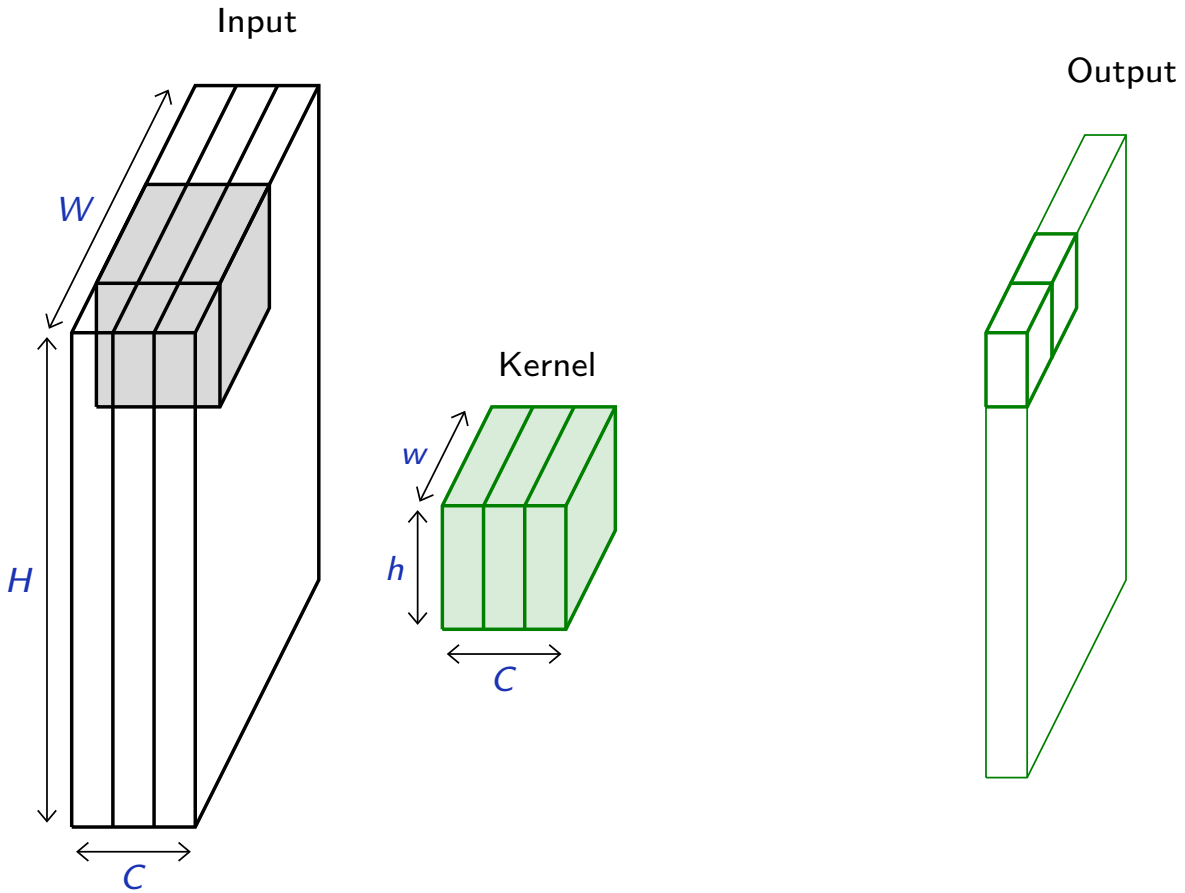
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

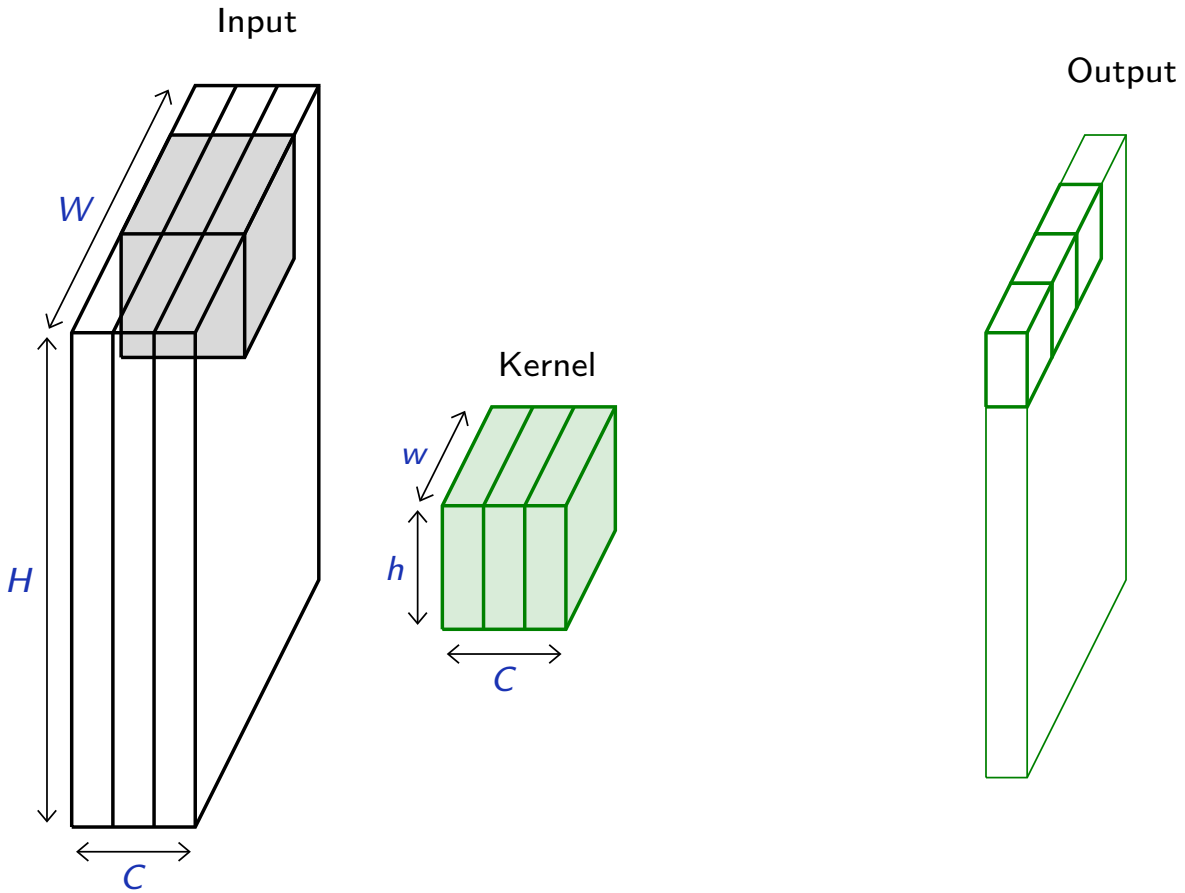
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

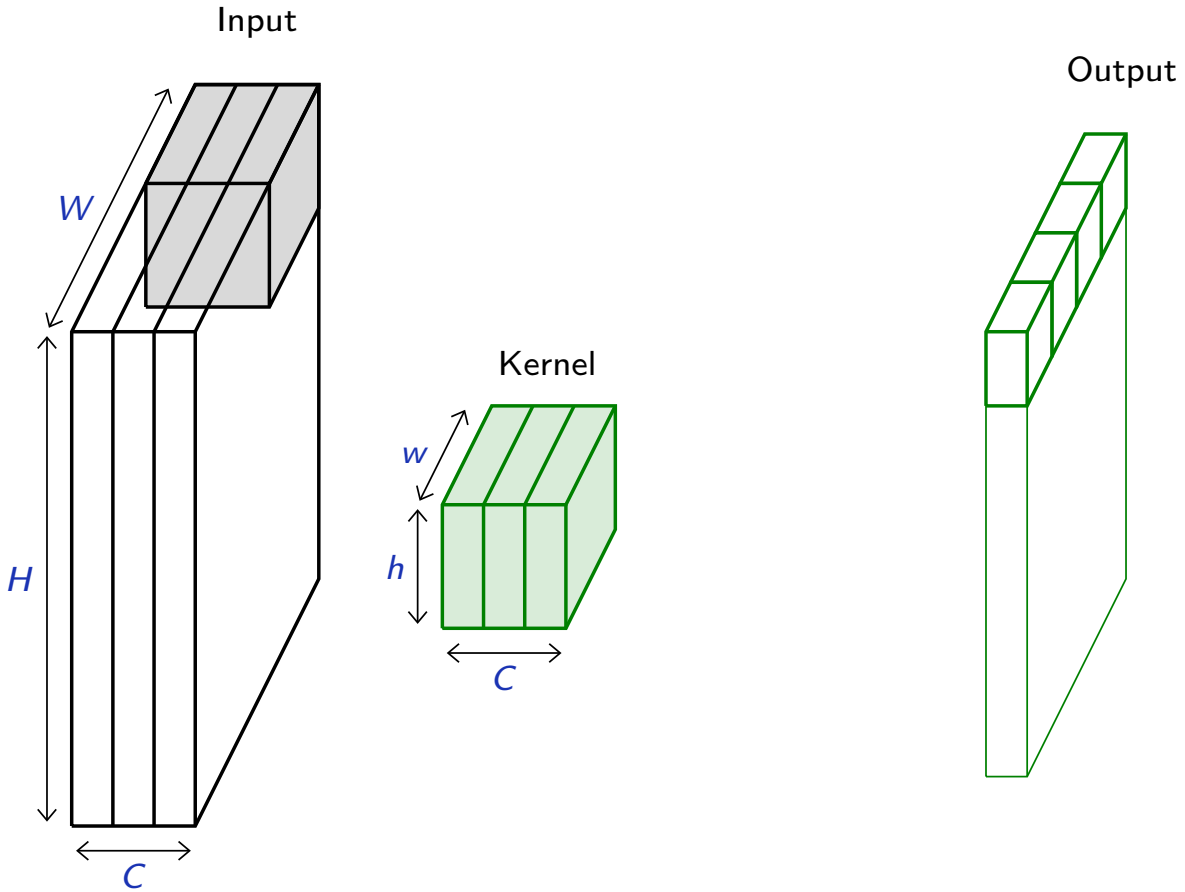
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

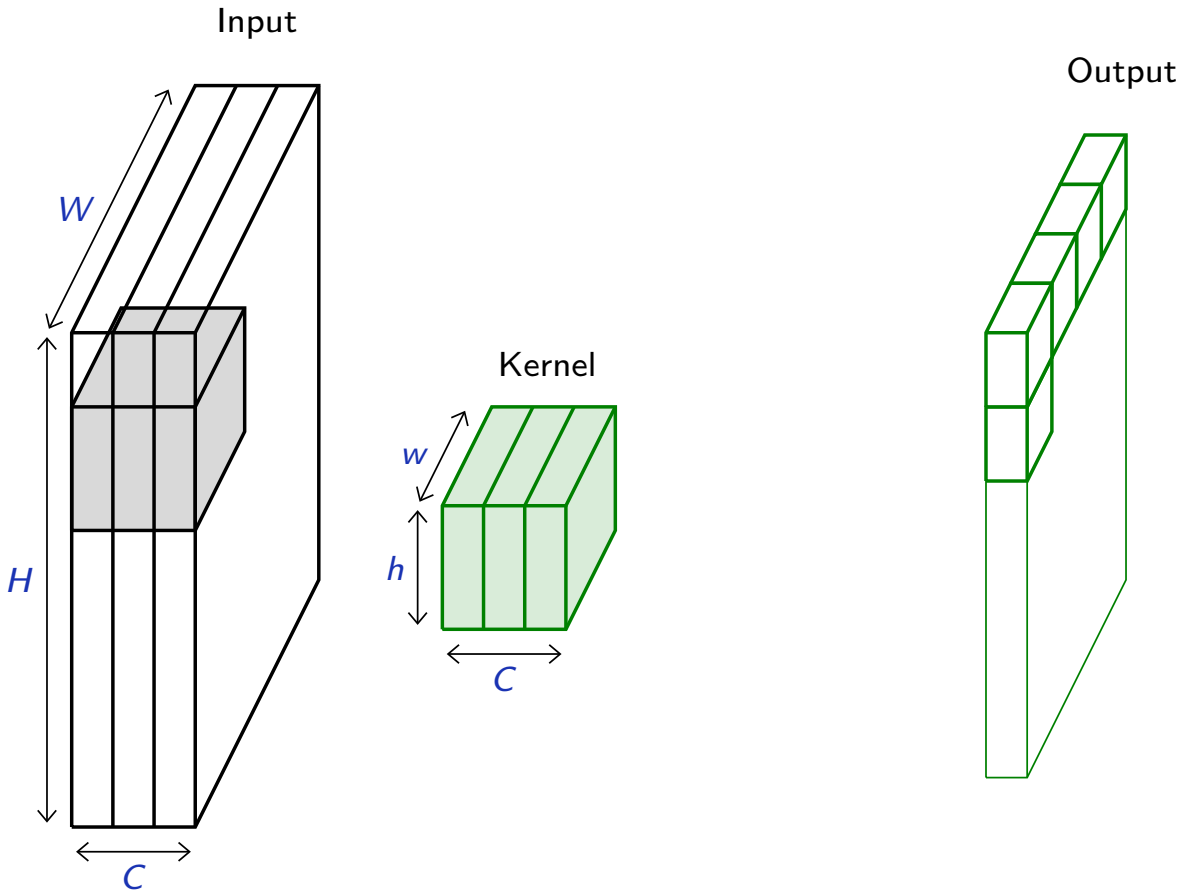
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

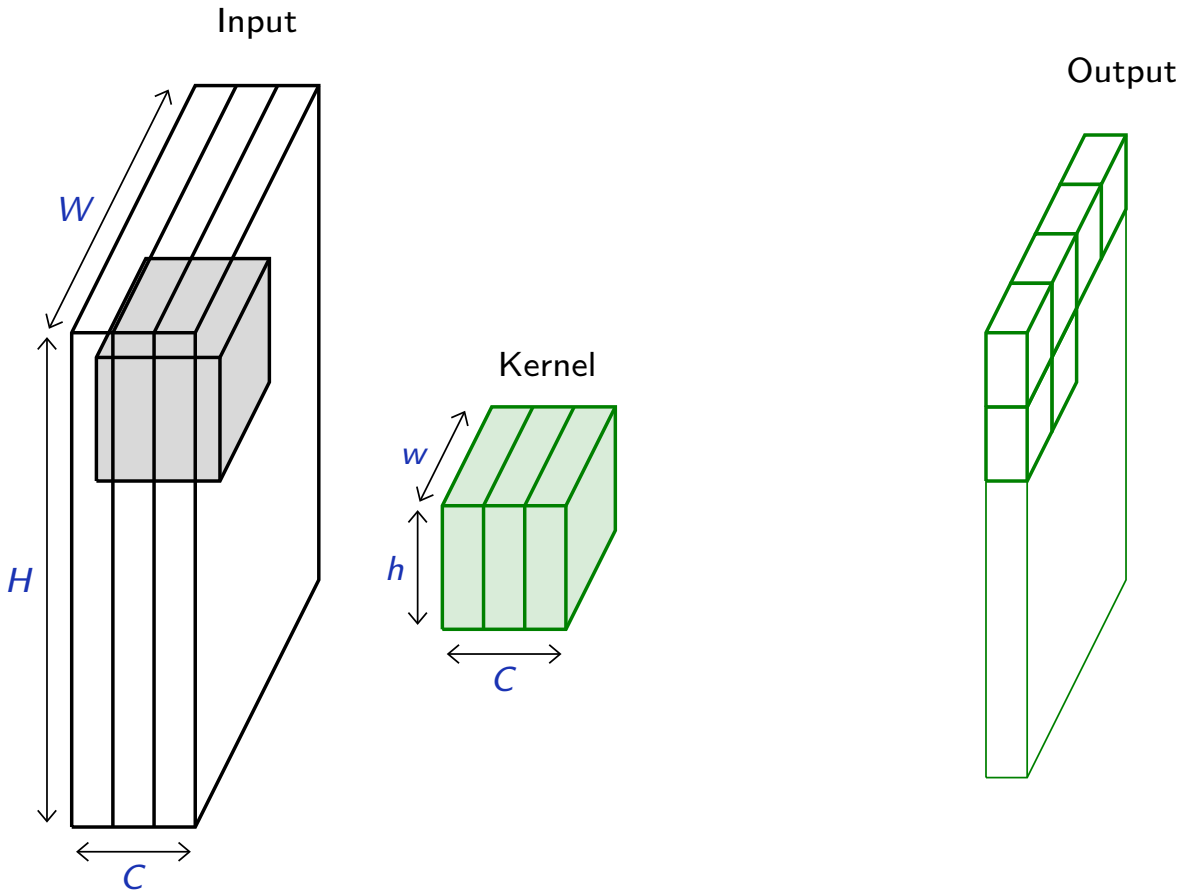
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

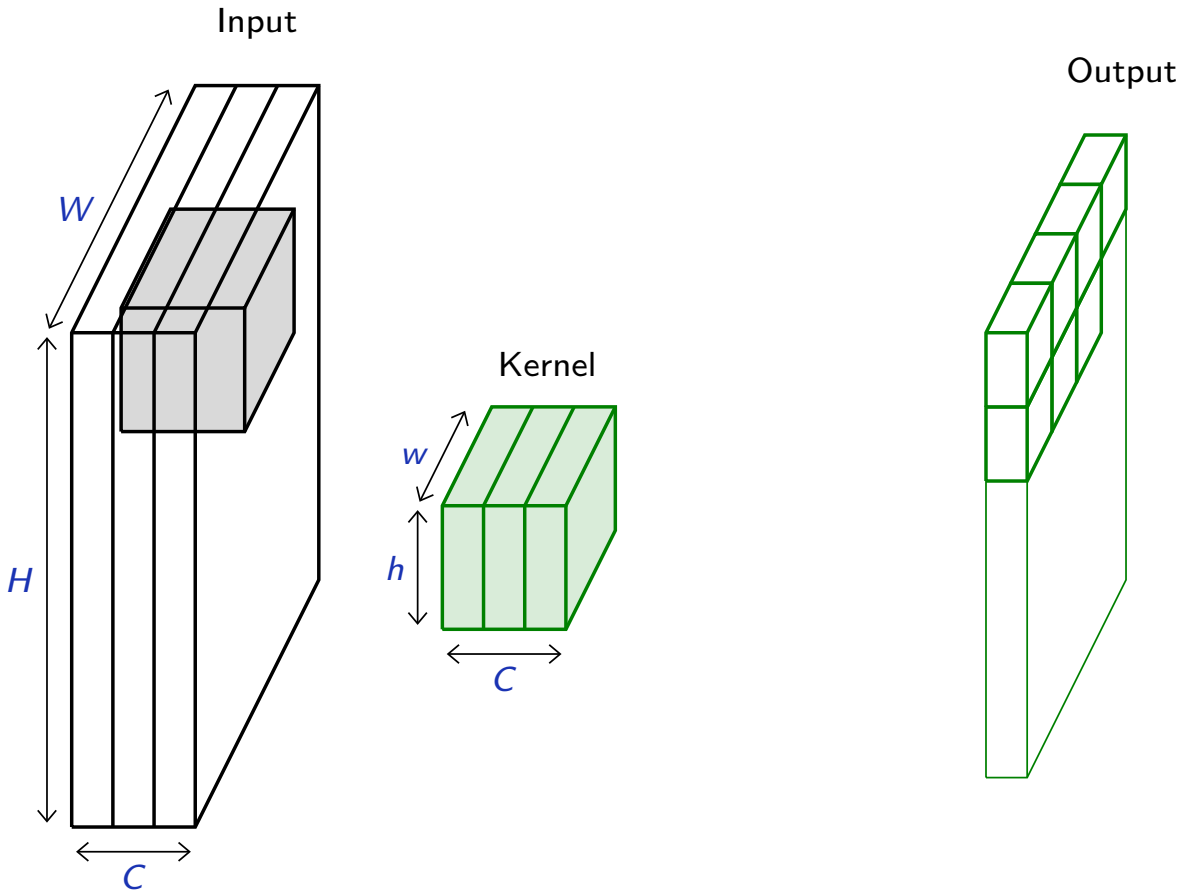
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

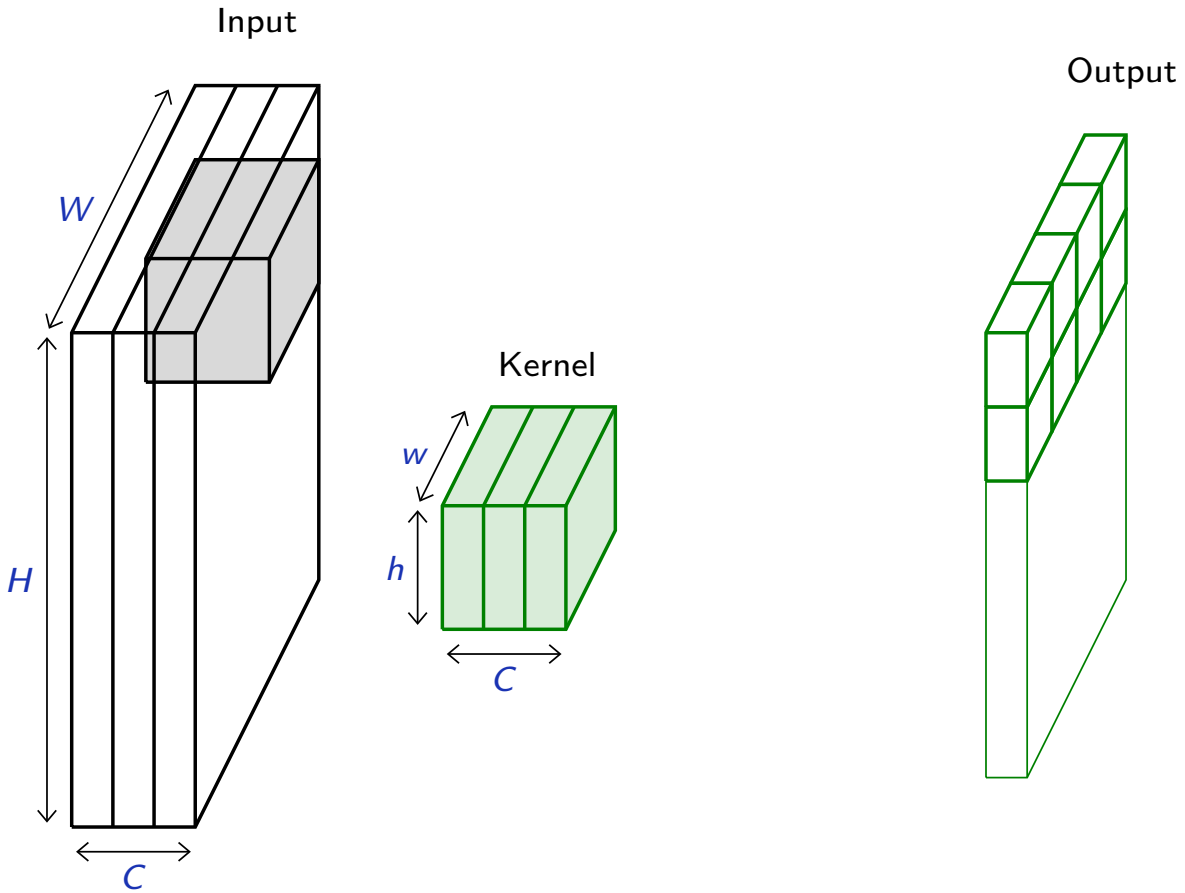
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

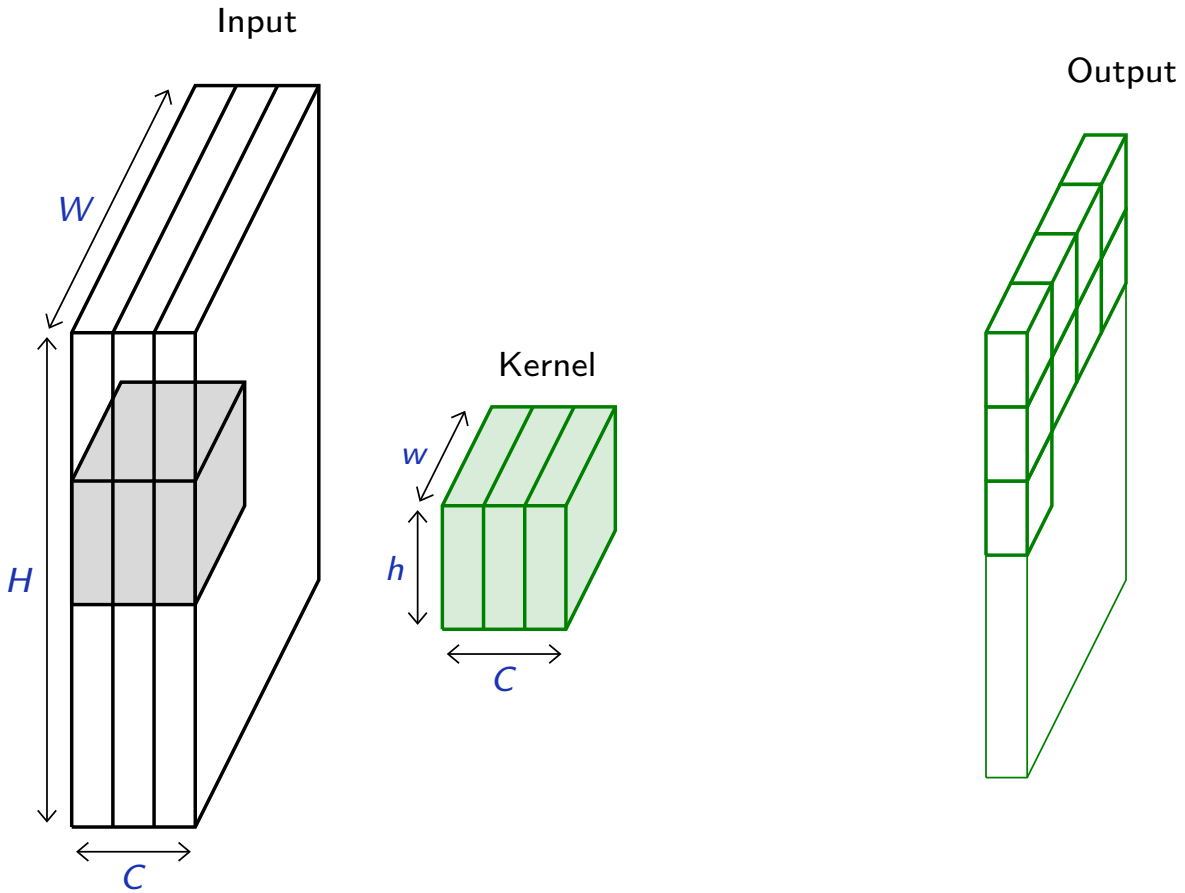
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

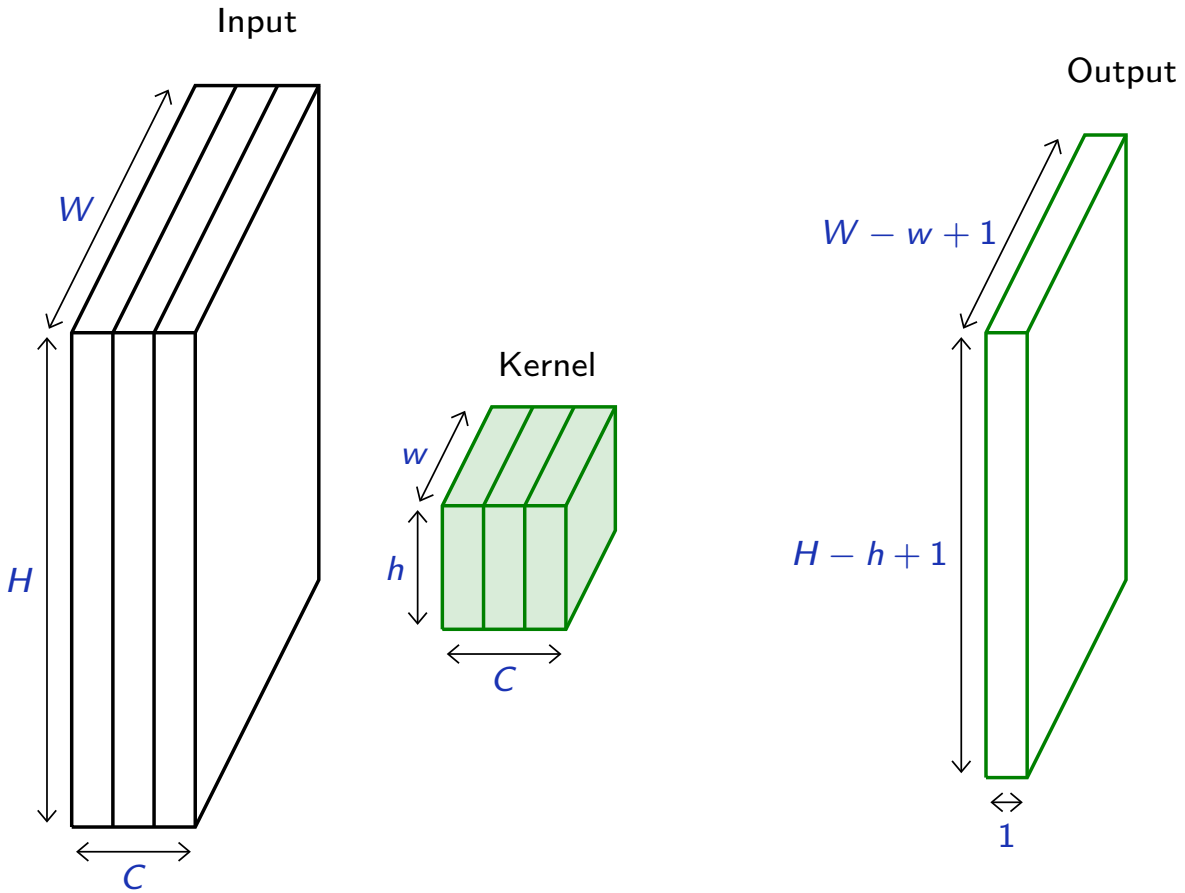
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

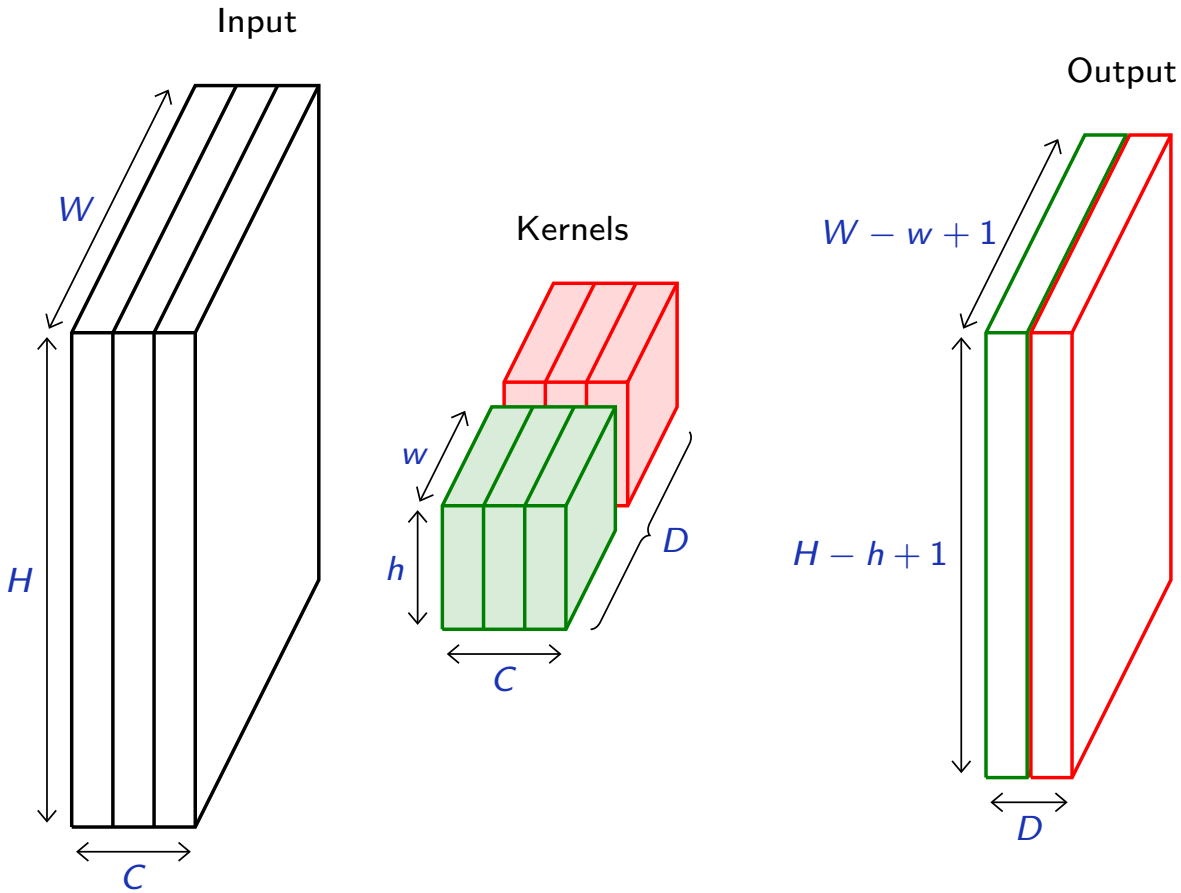
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

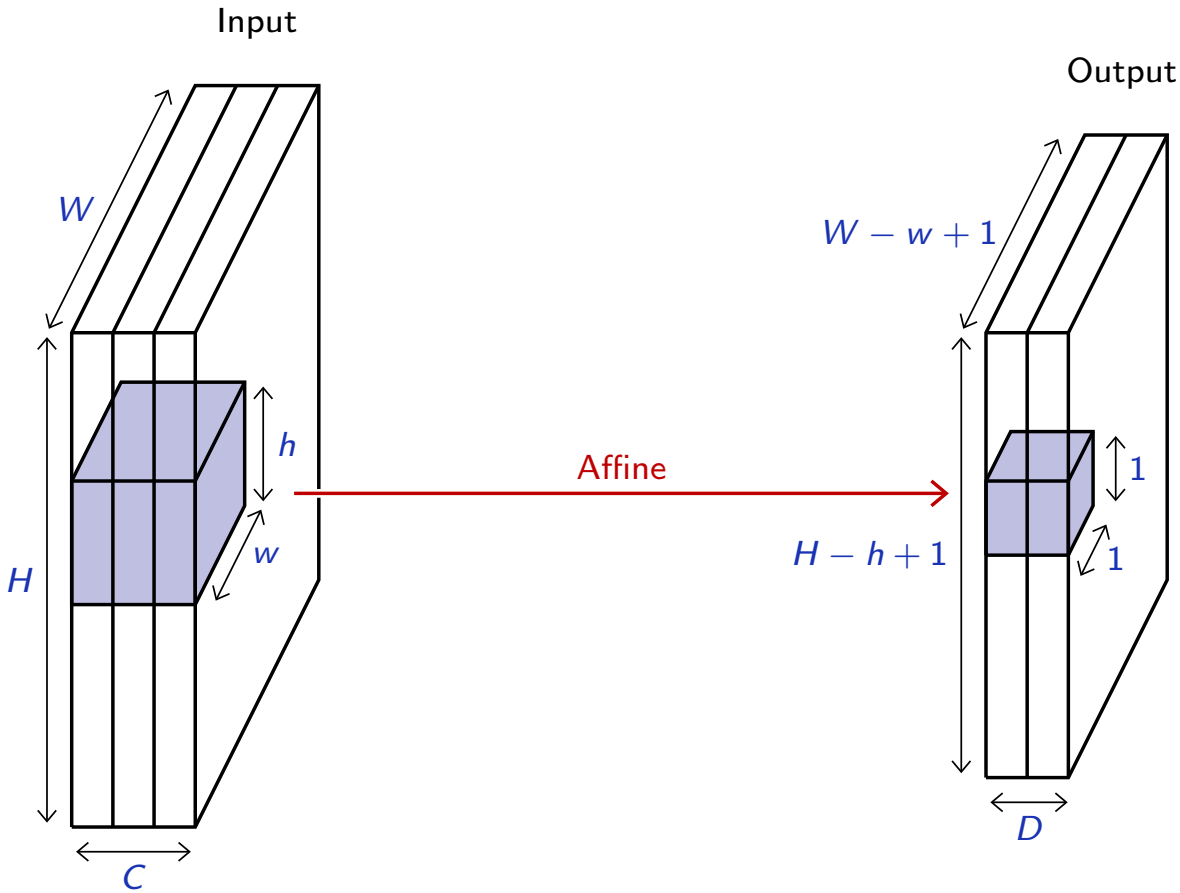
# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

# Couche convolutive (conv. layer)




Figures taken from [Fleuret, 2022].

- La convolution agrège tous les canaux, donnant une output de profondeur 1.

# Couche convolutive (conv. layer)

- La convolution est une opération affine entre l'input $\boldsymbol{X}$ et l'output $\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{K}$.
- Pour introduire de la non-linéarité, on applique généralement une fonction non-linéaire $\sigma$ ($\tanh$, ReLU, etc.) à $\boldsymbol{Z}$.
- La dynamique d'une couche convolutive s'écrit alors:

$$\boldsymbol{A} = \sigma(\boldsymbol{Z}) = \sigma(\boldsymbol{X} * \boldsymbol{K})$$

où $\sigma$ est appliquée composante par composante.

- Les paramètres (valeurs) du noyau $\boldsymbol{K}$ sont appris par backpropagation.

# Couche convolutive (conv. layer)

- La convolution est une opération affine entre l'input $\boldsymbol{X}$ et l'output $\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{K}$.
- Pour introduire de la non-linéarité, on applique généralement une fonction non-linéaire $\sigma$ ($\tanh$, ReLU, etc.) à $\boldsymbol{Z}$.
- La dynamique d'une couche convolutive s'écrit alors:

$$\boldsymbol{A} = \sigma(\boldsymbol{Z}) = \sigma(\boldsymbol{X} * \boldsymbol{K})$$

où $\sigma$ est appliquée composante par composante.

- Les paramètres (valeurs) du noyau $\boldsymbol{K}$ sont appris par backpropagation.

# Couche convolutive (conv. layer)

- La convolution est une opération affine entre l'input $\boldsymbol{X}$ et l'output $\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{K}$.
- Pour introduire de la non-linéarité, on applique généralement une fonction non-linéaire $\sigma$ ($\tanh$, ReLU, etc.) à $\boldsymbol{Z}$.
- La dynamique d'une couche convolutive s'écrit alors:

$$\boldsymbol{A} = \sigma(\boldsymbol{Z}) = \sigma(\boldsymbol{X} * \boldsymbol{K})$$

où $\sigma$ est appliquée composante par composante.

- Les paramètres (valeurs) du noyau $\boldsymbol{K}$ sont appris par backpropagation.

# Couche convolutive (conv. layer)

- La convolution est une opération affine entre l'input $\boldsymbol{X}$ et l'output $\boldsymbol{Z} = \boldsymbol{X} * \boldsymbol{K}$.
- Pour introduire de la non-linéarité, on applique généralement une fonction non-linéaire $\sigma$ ($\tanh$, ReLU, etc.) à $\boldsymbol{Z}$.
- La dynamique d'une couche convolutive s'écrit alors:

$$\boldsymbol{A} = \sigma(\boldsymbol{Z}) = \sigma(\boldsymbol{X} * \boldsymbol{K})$$

où $\sigma$ est appliquée composante par composante.

- Les paramètres (valeurs) du noyau $\boldsymbol{K}$ sont appris par backpropagation.

# Padding, stride et dilatation

Les couches convolutives possèdent également les spécifications suivantes:

- Le **padding**: spécifie les dimension d'un cadre de valeur $0$ ajouté de autour de l'input.
- Le **stride**: spécifie la taille du pas avec lequel le filtre de convolution parcourt l'input.
- La **dilatation**: module l'expansion (ou écartement) du filtre sans ajouter de paramètres.

# Padding, stride et dilatation

Les couches convolutives possèdent également les spécifications suivantes:

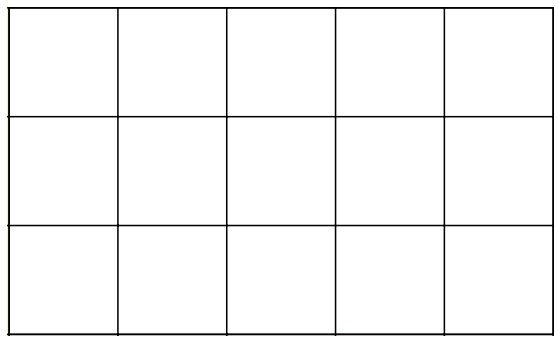
- Le **padding**: spécifie les dimension d'un cadre de valeur $0$ ajouté de autour de l'input.
- Le **stride**: spécifie la taille du pas avec lequel le filtre de convolution parcourt l'input.
- La **dilatation**: module l'expansion (ou écartement) du filtre sans ajouter de paramètres.

# Padding, stride et dilatation

Les couches convolutives possèdent également les spécifications suivantes:

- Le **padding**: spécifie les dimension d'un cadre de valeur $0$ ajouté de autour de l'input.
- Le **stride**: spécifie la taille du pas avec lequel le filtre de convolution parcourt l'input.
- La **dilatation**: module l'expansion (ou écartement) du filtre sans ajouter de paramètres.

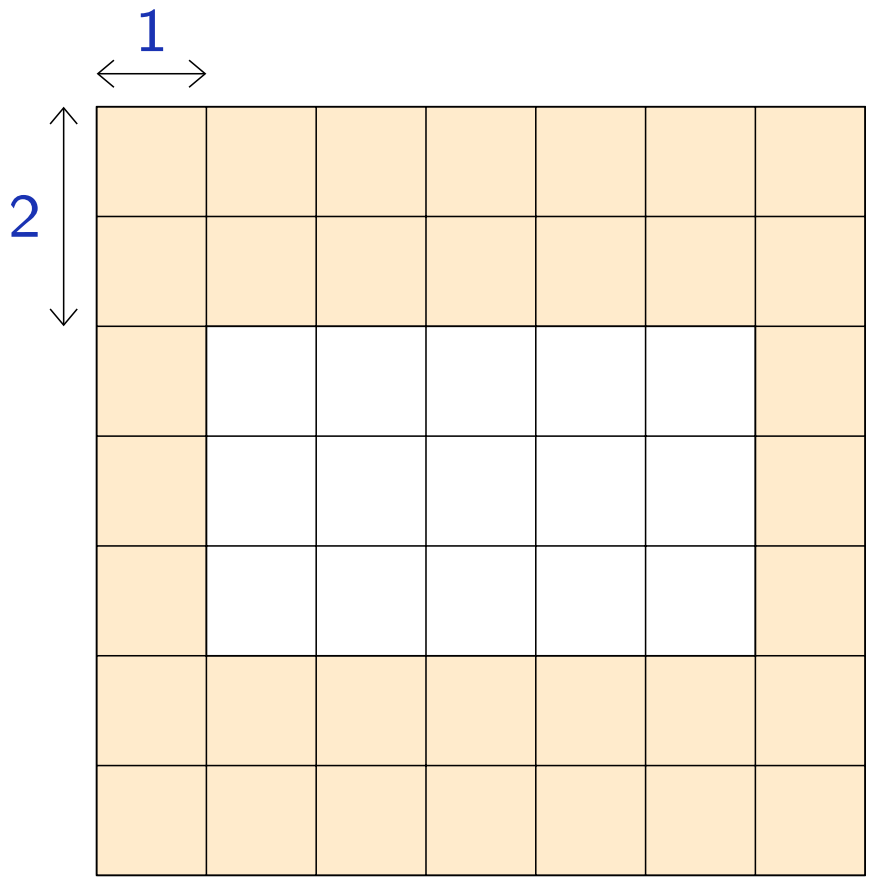
# Padding and stride



Input

Figures taken from [Fleuret, 2022].

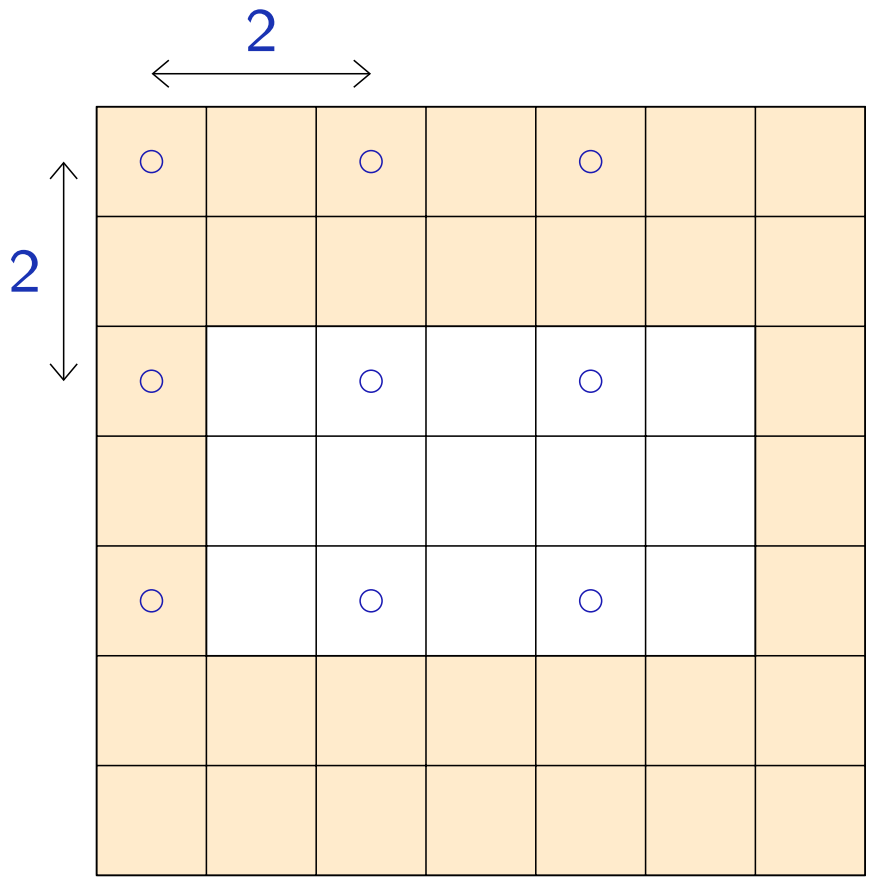
# Padding and stride




Input

Figures taken from [Fleuret, 2022].

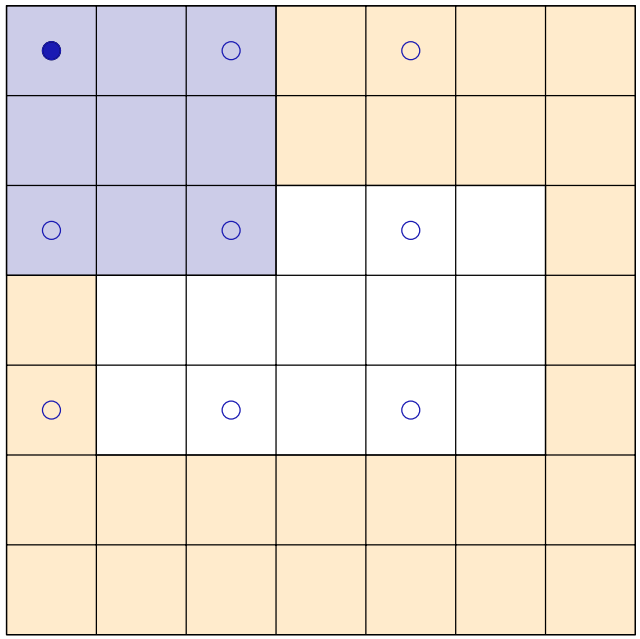
# Padding and stride



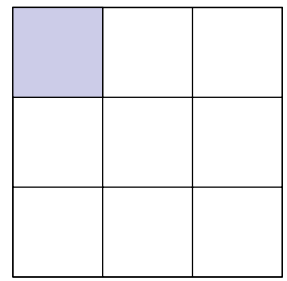

Input

Figures taken from [Fleuret, 2022].

# Padding and stride



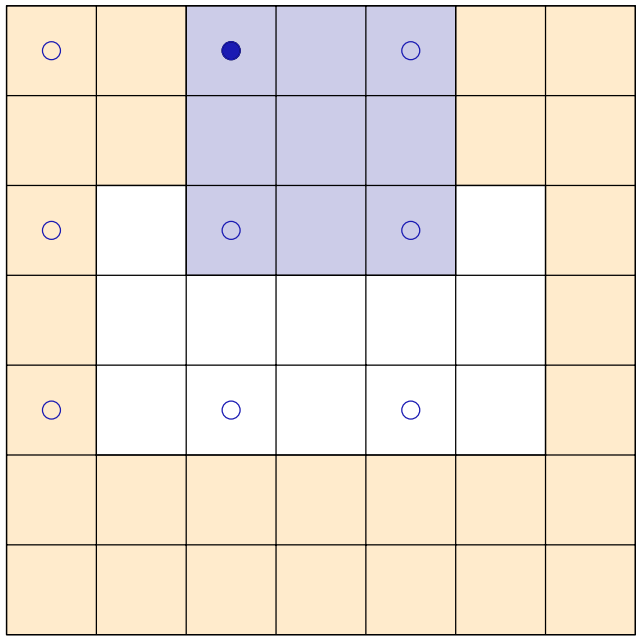
Input



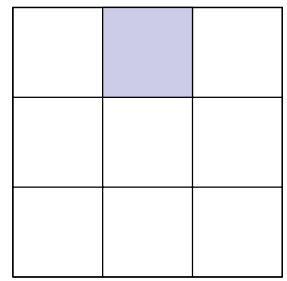
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



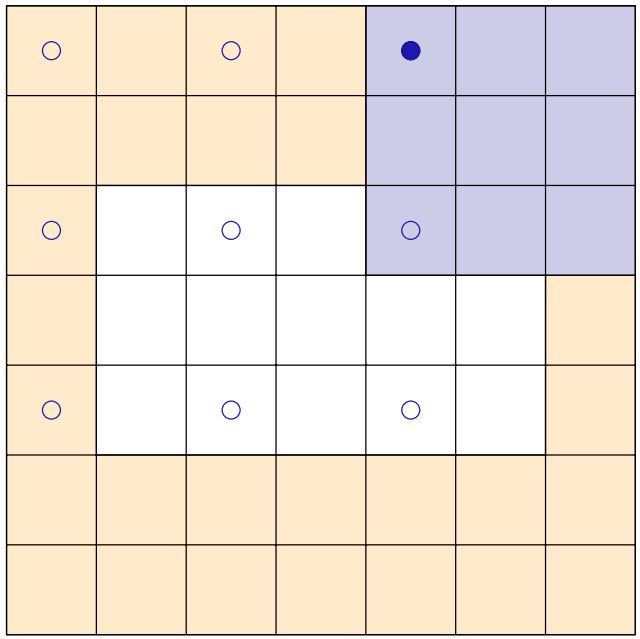
Input



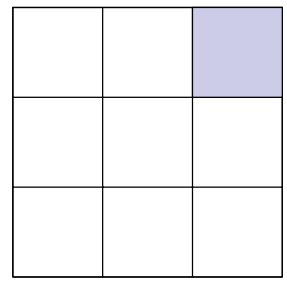
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



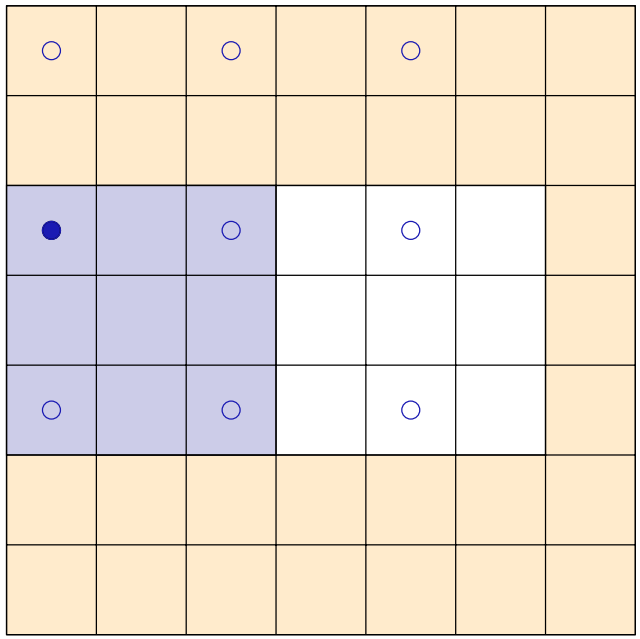
Input



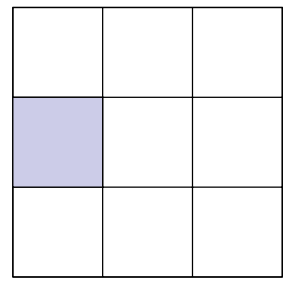
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



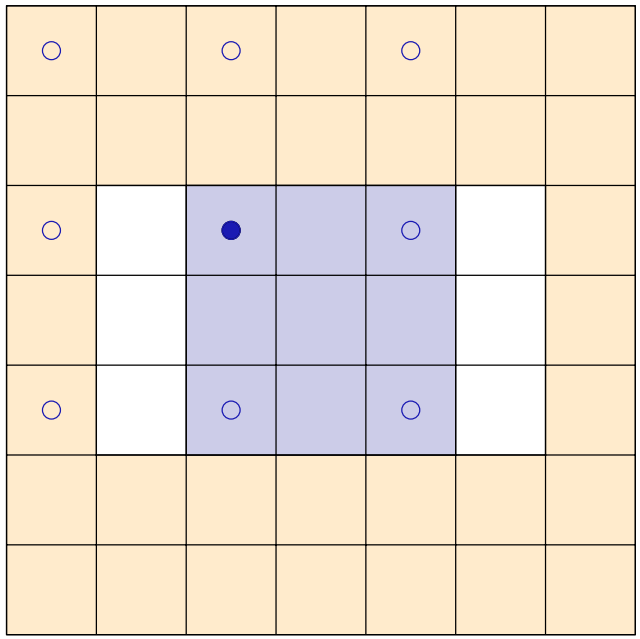
Input



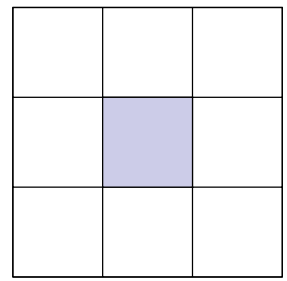
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



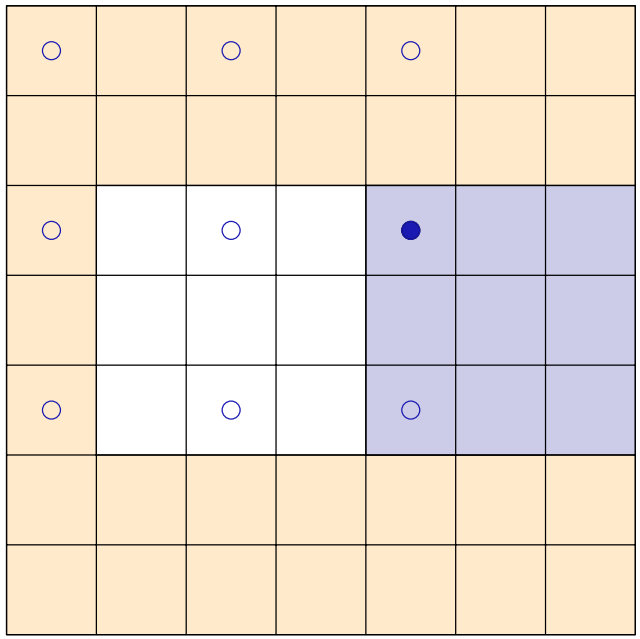
Input



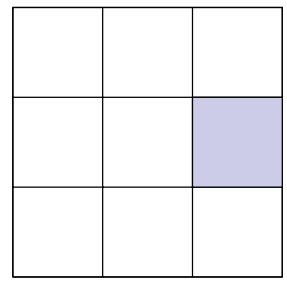
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



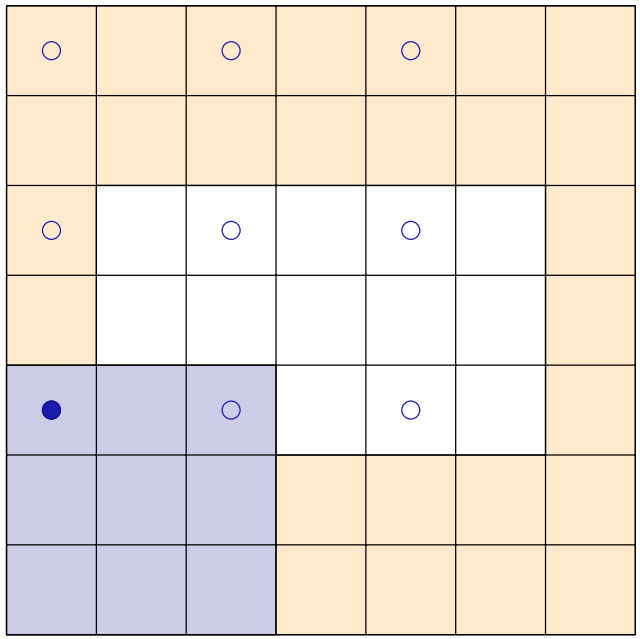
Input



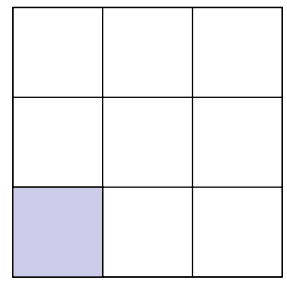
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



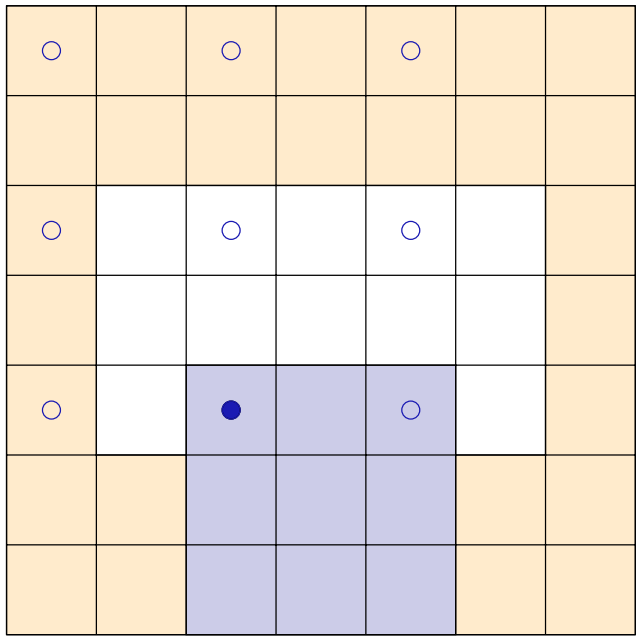
Input



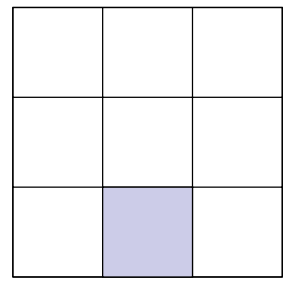
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



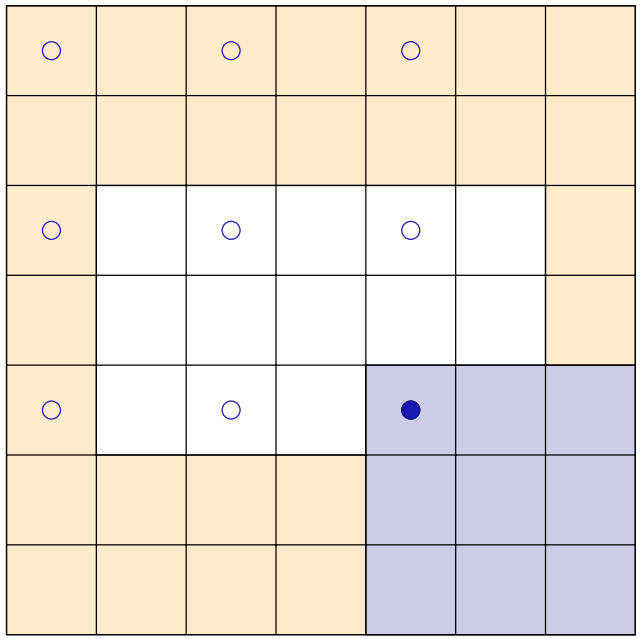
Input



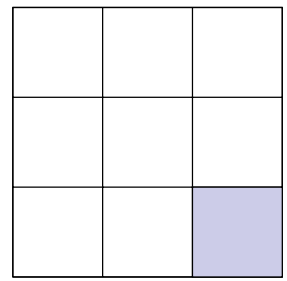
Output

Figures taken from [Fleuret, 2022].

# Padding and stride



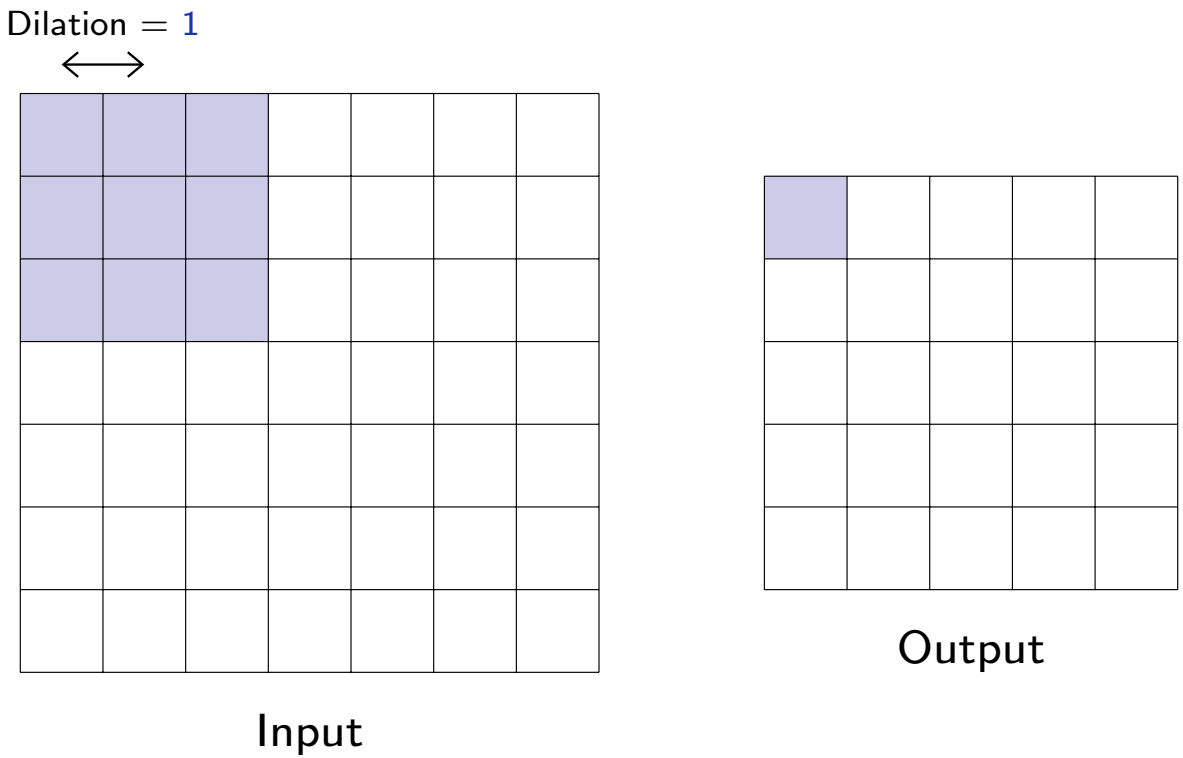
Input



Output

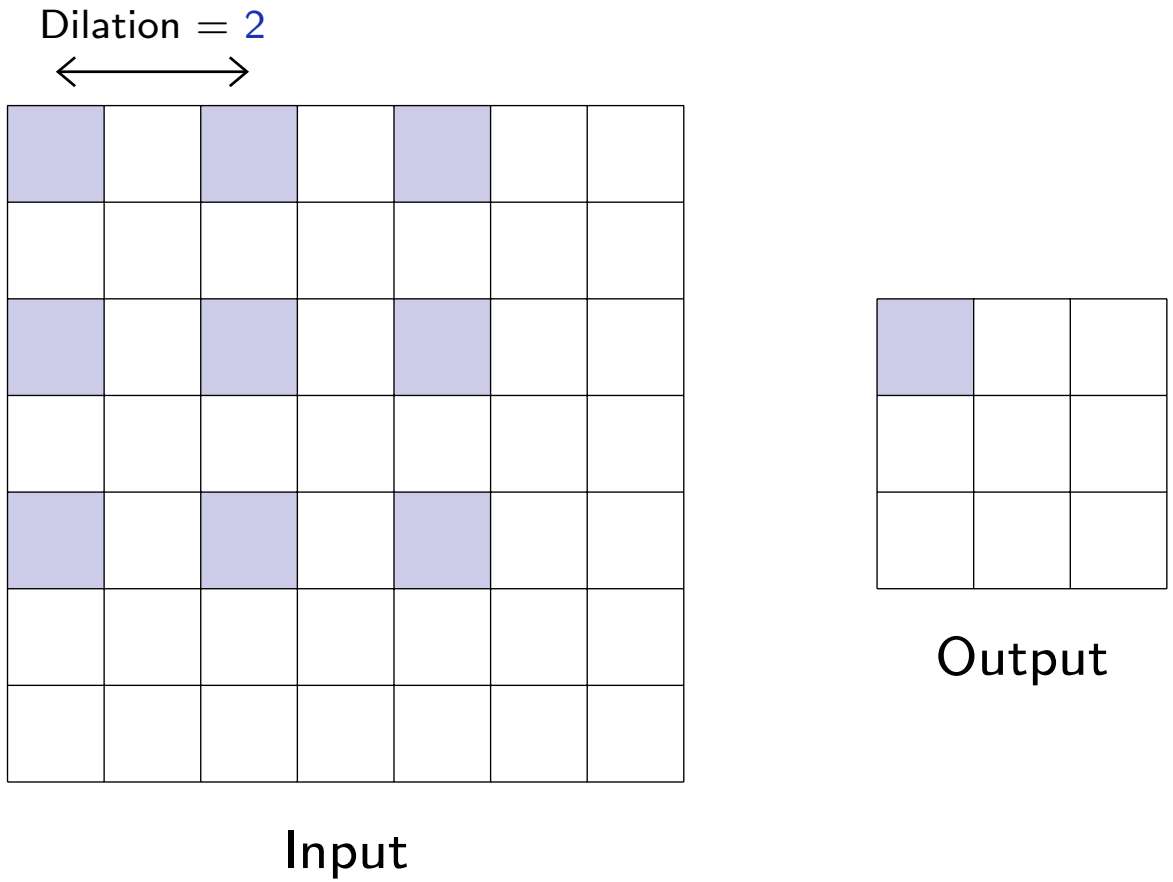
Figures taken from [Fleuret, 2022].

# Dilatation



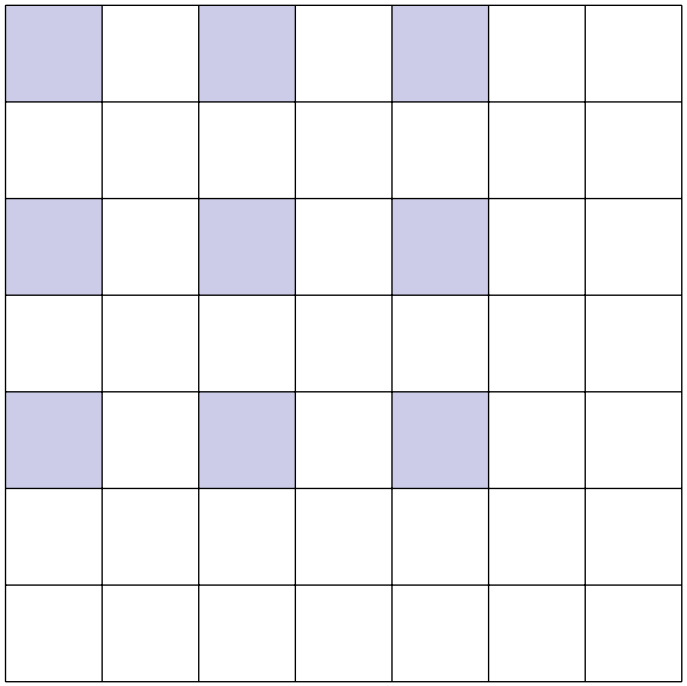

Figures taken from [Fleuret, 2022].

# Dilatation

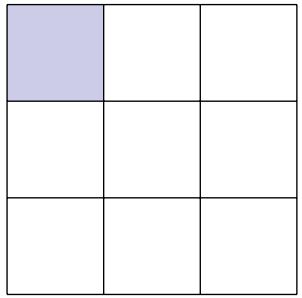



Figures taken from [Fleuret, 2022].

# Dilatation



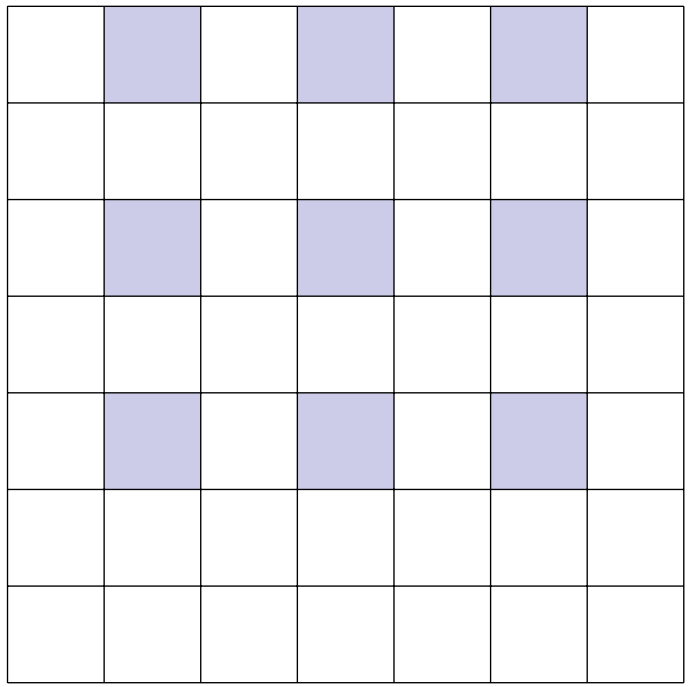
Input



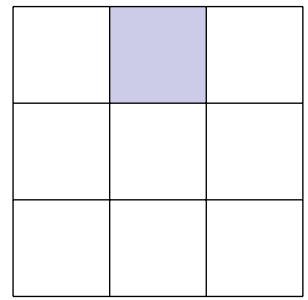
Output

Figures taken from [Fleuret, 2022].

# Dilatation



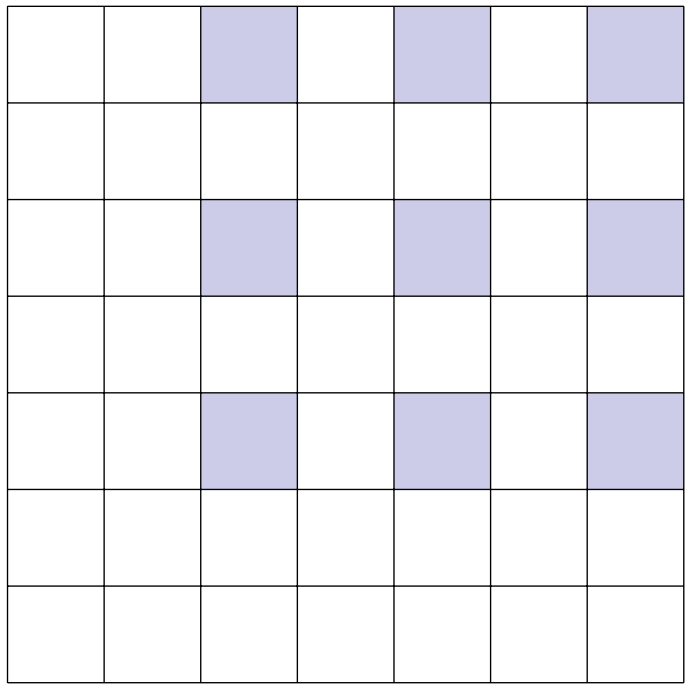
Input



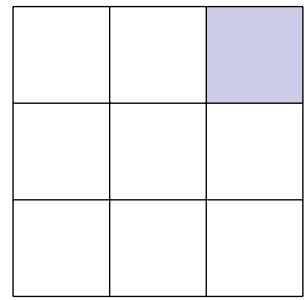
Output

Figures taken from [Fleuret, 2022].

# Dilatation



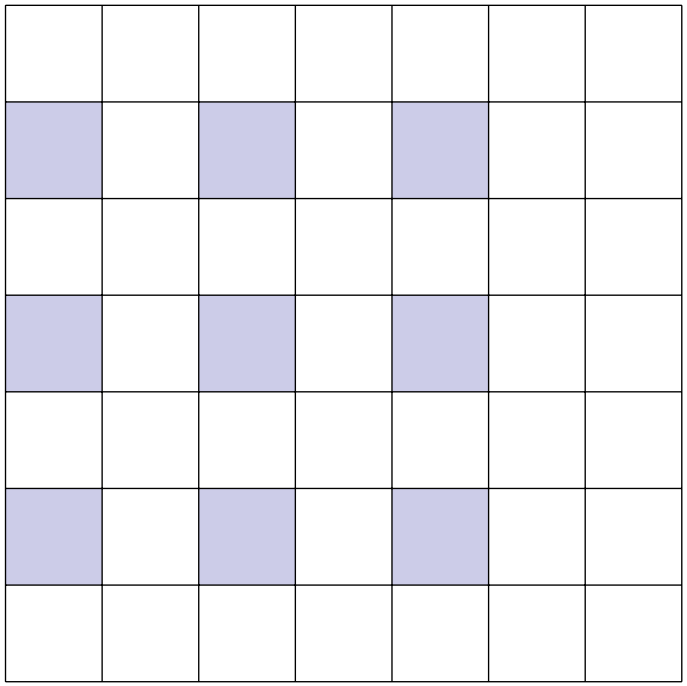
Input



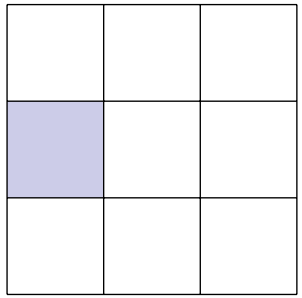
Output

Figures taken from [Fleuret, 2022].

# Dilatation



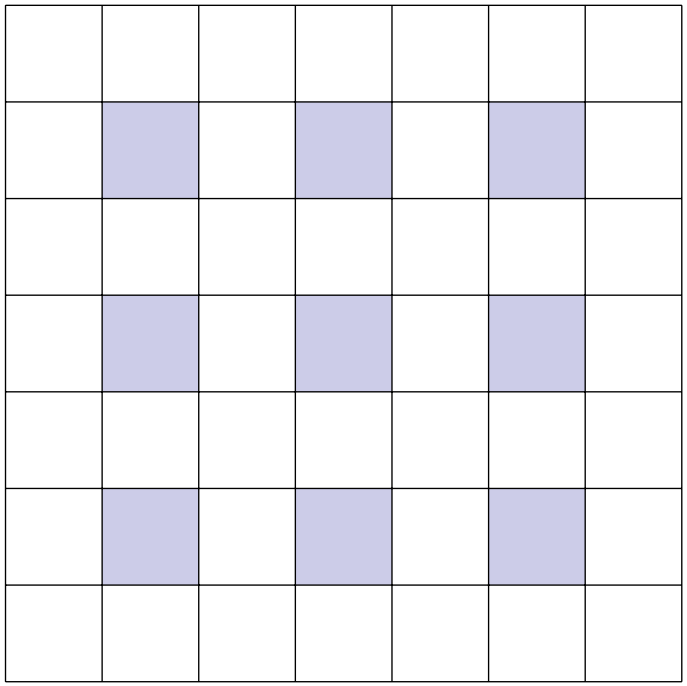
Input



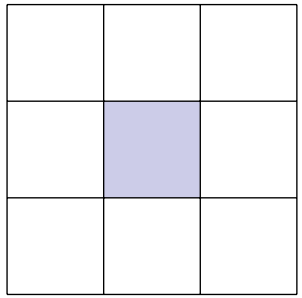
Output

Figures taken from [Fleuret, 2022].

# Dilatation



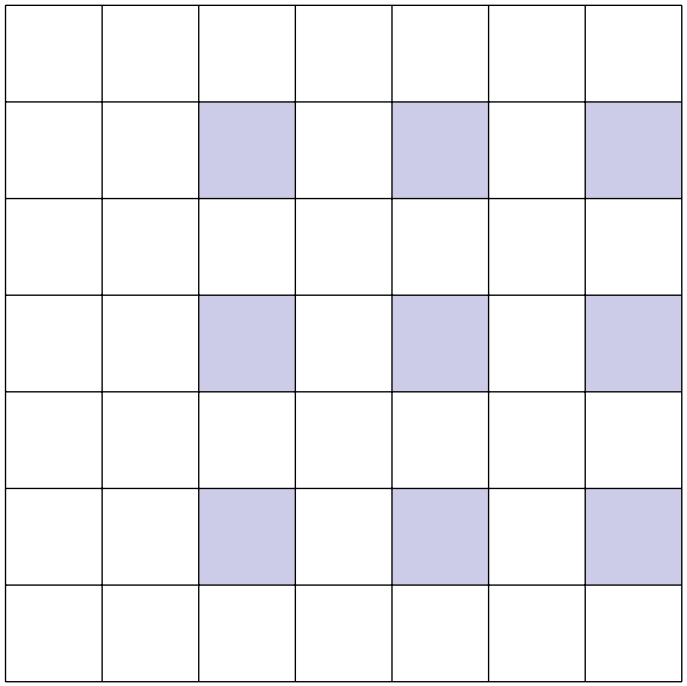
Input



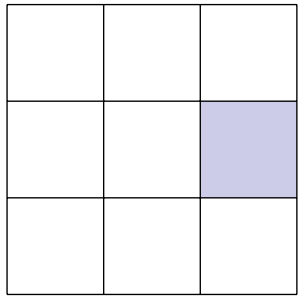
Output

Figures taken from [Fleuret, 2022].

# Dilatation



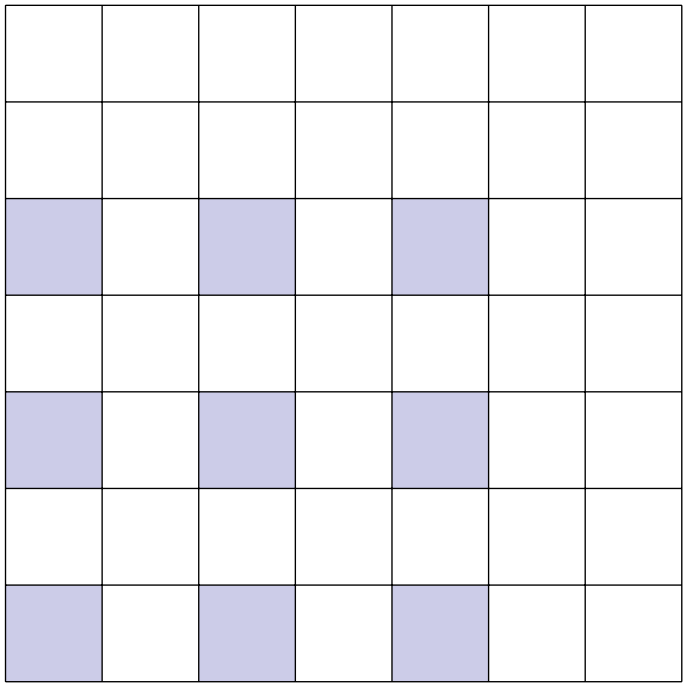
Input



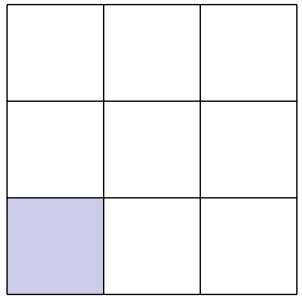
Output

Figures taken from [Fleuret, 2022].

# Dilatation



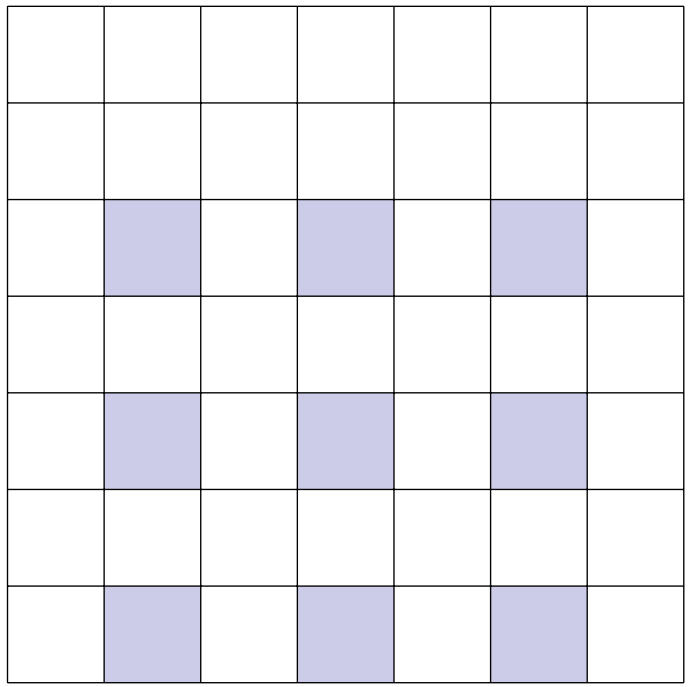
Input



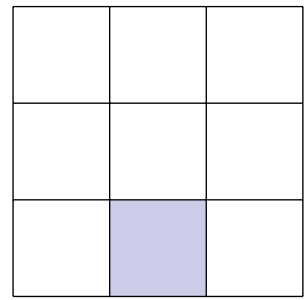
Output

Figures taken from [Fleuret, 2022].

# Dilatation



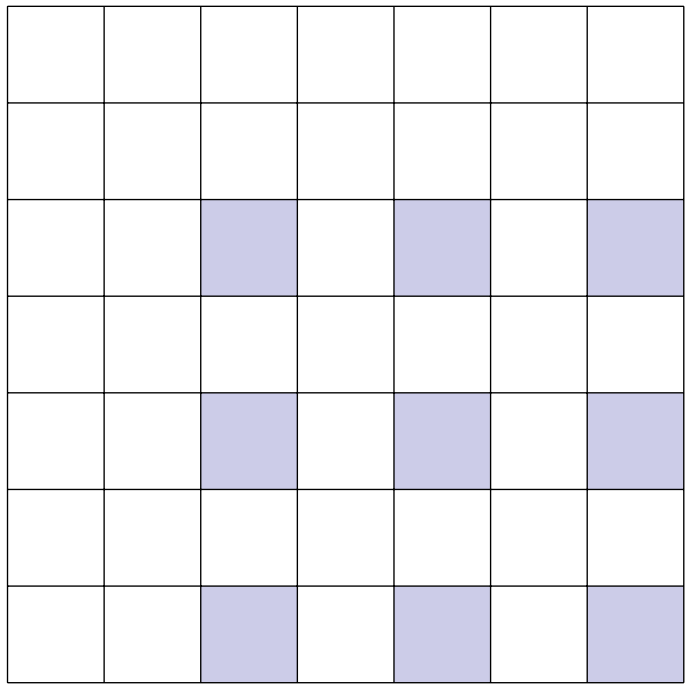
Input



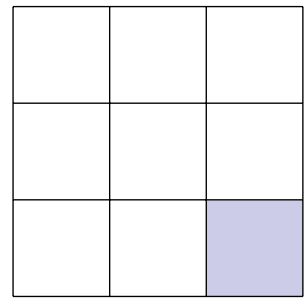
Output

Figures taken from [Fleuret, 2022].

# Dilatation



Input



Output

Figures taken from [Fleuret, 2022].

# Avantages des couches convolutives

Les couches convolutives possèdent les avantages suivants par rapport aux couches denses classiques:

- **Parameter sharing**

  Stockage mémoire:

  $k$ paramètres partagés (conv.) vs $m \times n$ paramètres (dense), avec $k << m, n$.

- **Sparse connectivity**

  Forward pass:

  $O(k \times n)$ (conv.) vs $O(m \times n)$ (dense) pour calcul de l'output, avec $k << m$.

# Avantages des couches convolutives

Les couches convolutives possèdent les avantages suivants par rapport aux couches denses classiques:

- **Parameter sharing**

  Stockage mémoire:

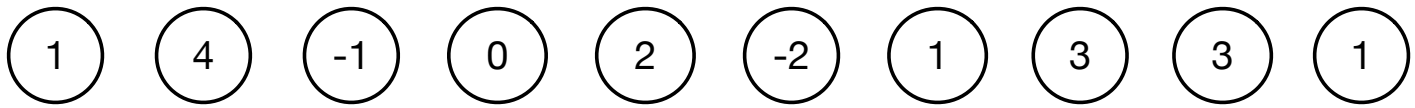
  $k$ paramètres partagés (conv.) vs $m \times n$ paramètres (dense), avec $k << m, n$.

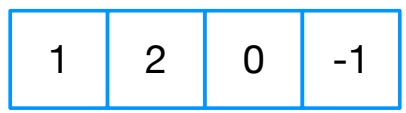
- **Sparse connectivity**

  Forward pass:

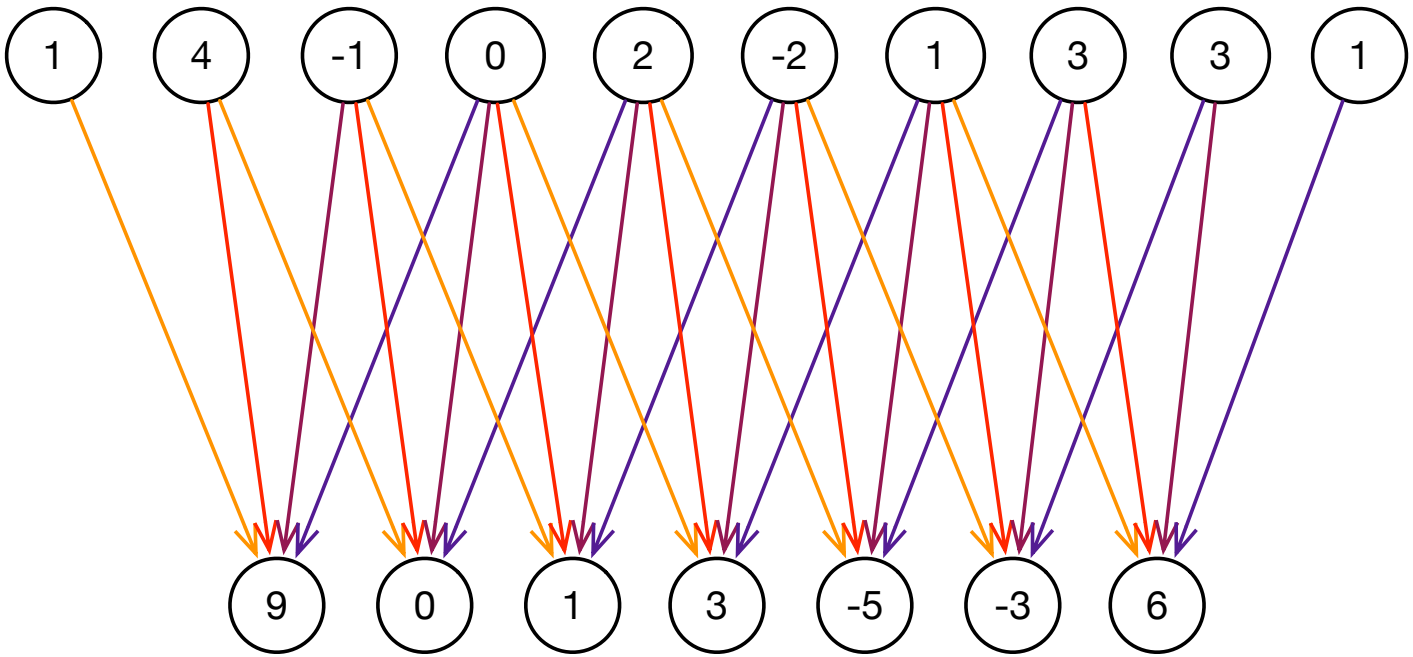
  $O(k \times n)$ (conv.) vs $O(m \times n)$ (dense) pour calcul de l'output, avec $k << m$.

# Avantages des couches convolutives

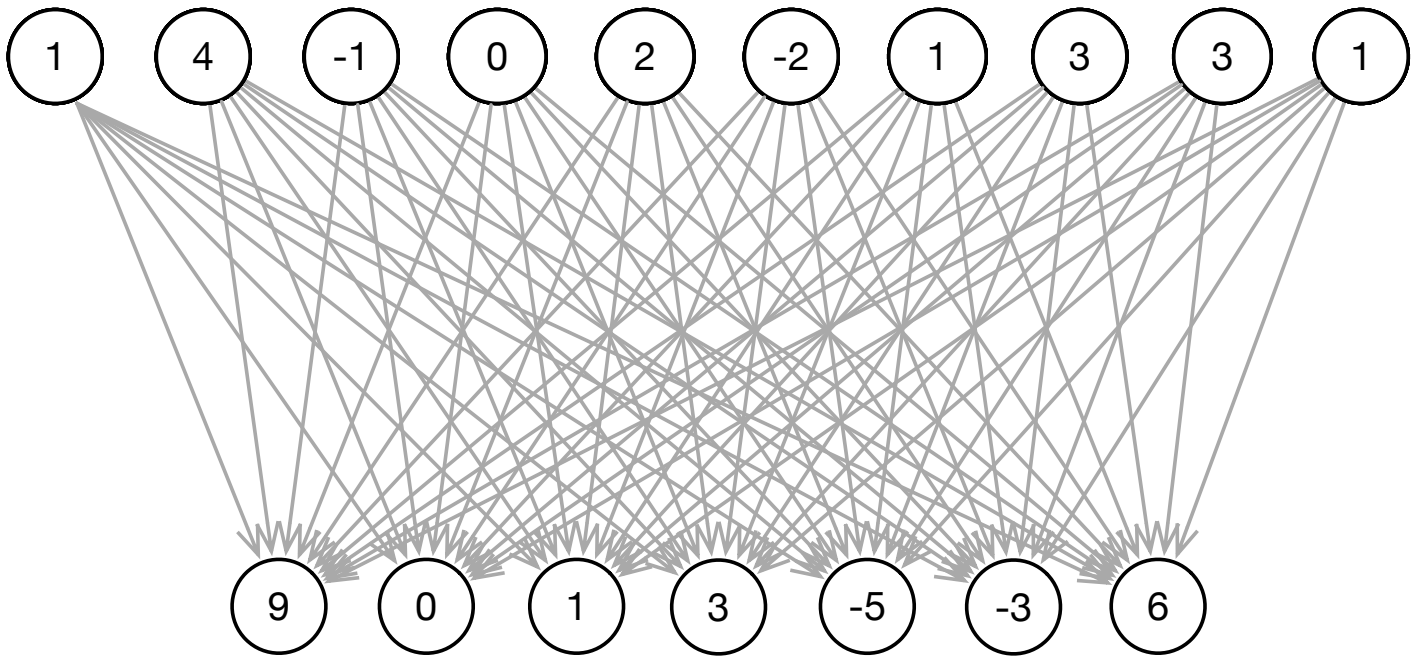


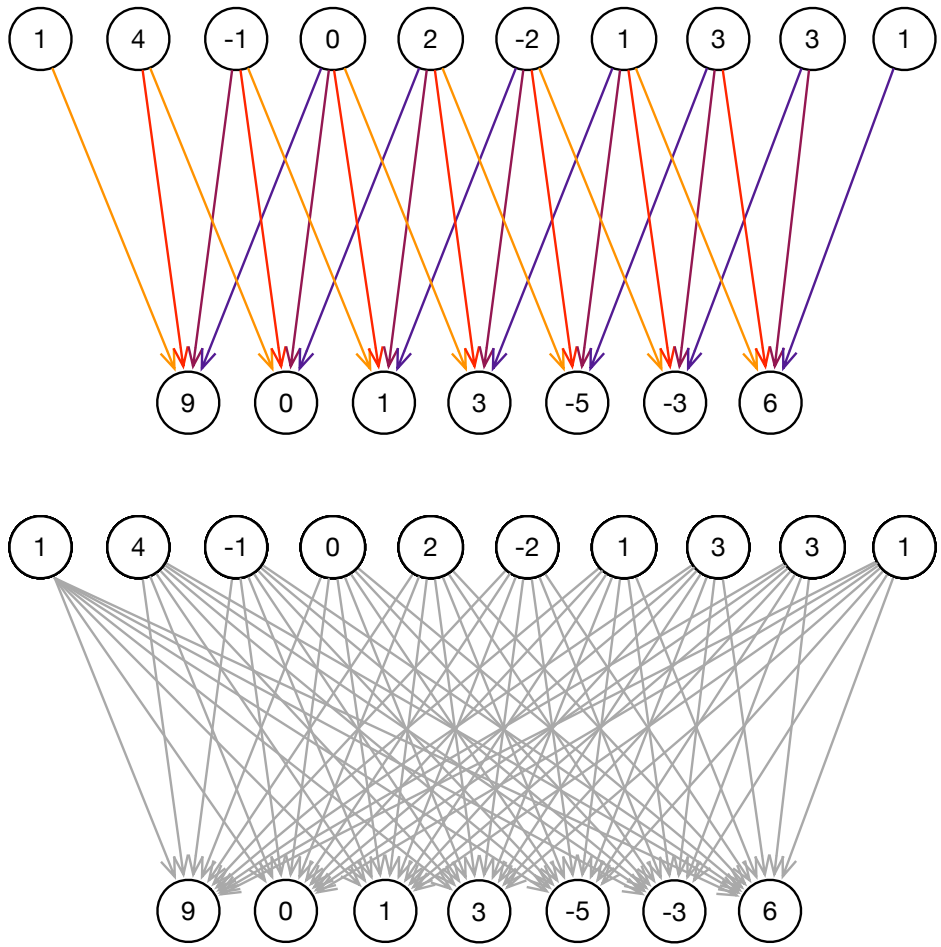

# Avantages des couches convolutives



# Avantages des couches convolutives



# Avantages des couches convolutives



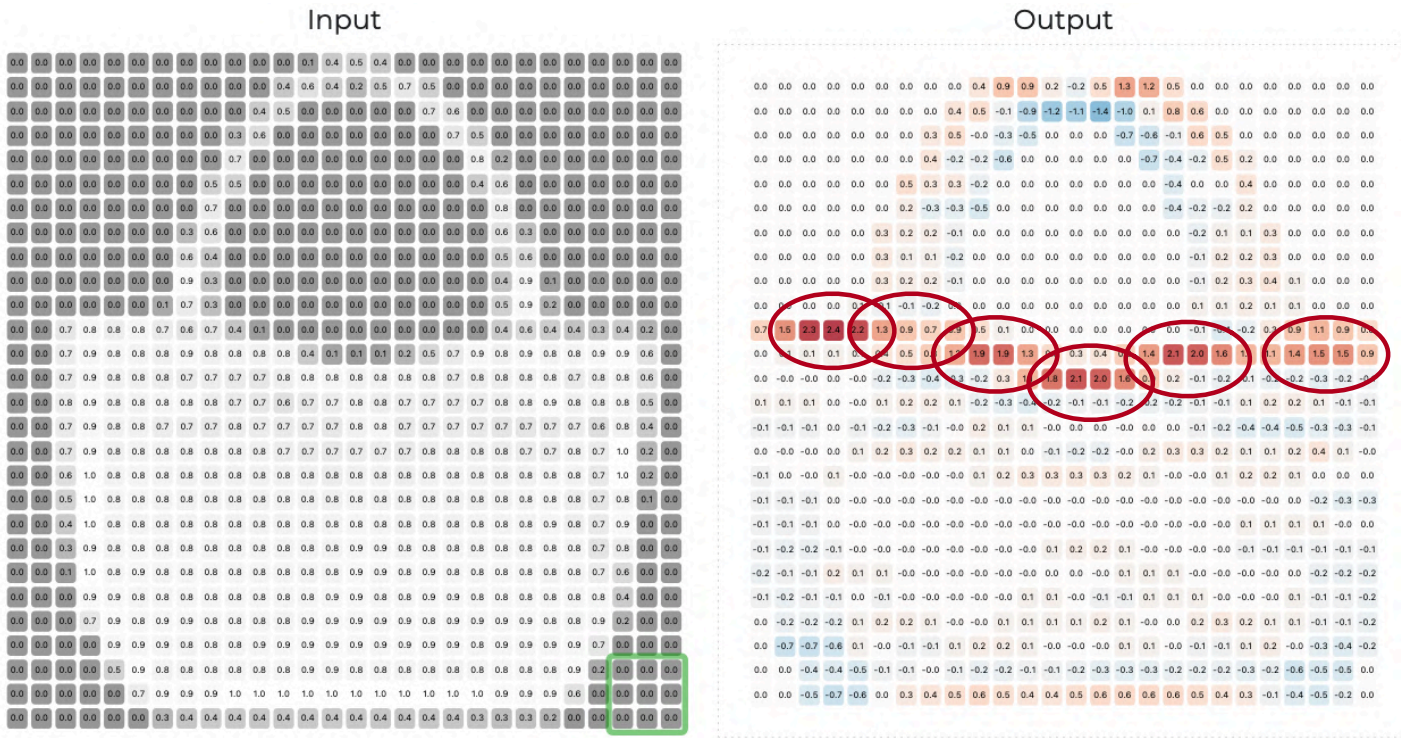
# Avantages des couches convolutives

- **Equivariant representation**

  Des patterns similaires de l'input (e.g., bords, arrondis, etc.), quels que soient leurs emplacements, sont transformés de manières similaires dans l'output.

  Ainsi, les filtres apprennent des représentations de patterns spécifiques, et une image sera caractérisée par l'emplacement de ces patterns.

# Avantages des couches convolutives




- Les bords supérieurs dans l'input, indépendamment de leur emplacement, sont détectés de manières similaires dans l'output (invariance par translation).

# Pooling

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Input

| | | | | |
|---|---|---|---|---|
| 3 | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Output

Figures taken from [Fleuret, 2022]

# Pooling

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Input

| | | | | |
|---|---|---|---|---|
| 3 | 4 | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Output

Figures taken from [Fleuret, 2022]

# Pooling

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

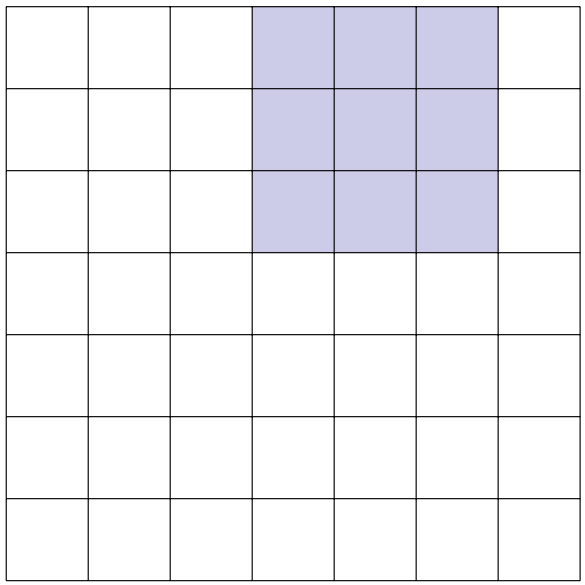
Input

| | | | | |
|---|---|---|---|---|
| 3 | 4 | 5 | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

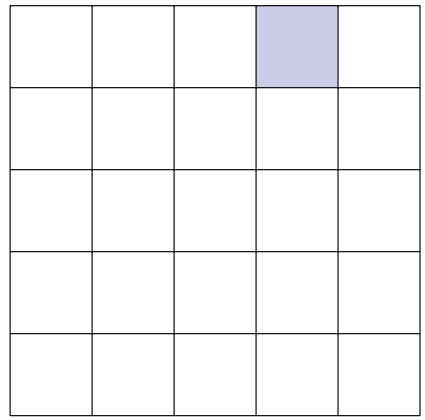
Output

Figures taken from [Fleuret, 2022]

# Pooling



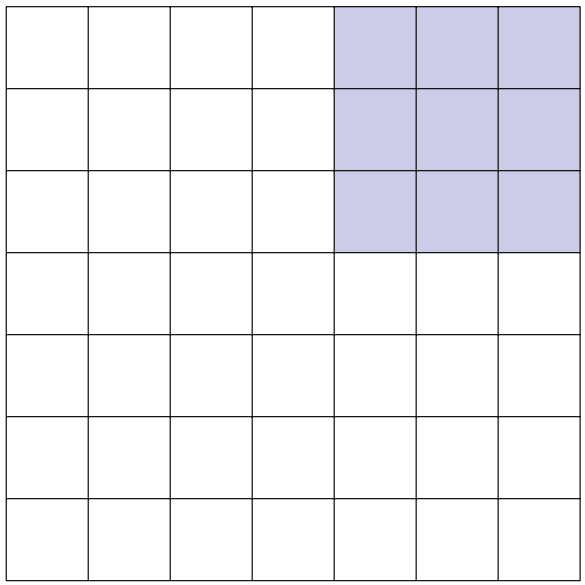
Input



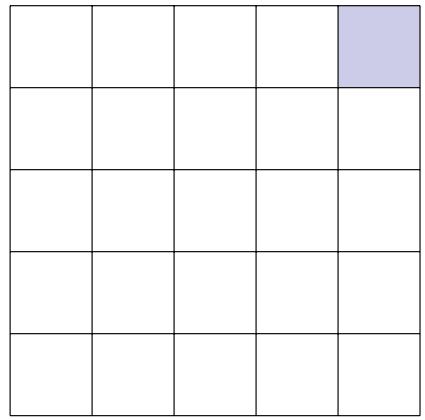
Output

Figures taken from [Fleuret, 2022]

# Pooling



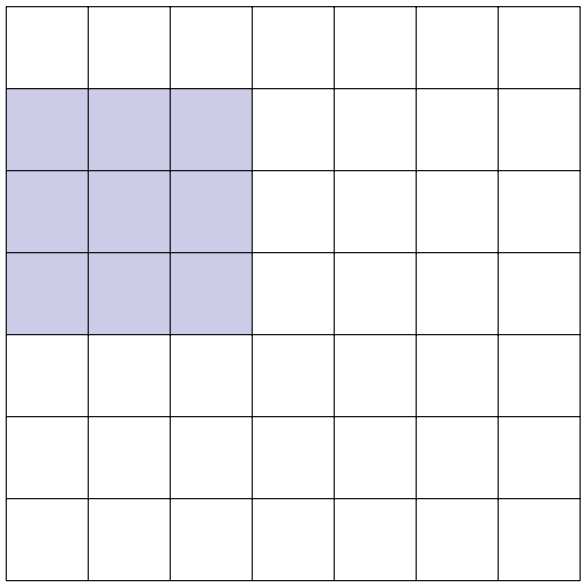
Input



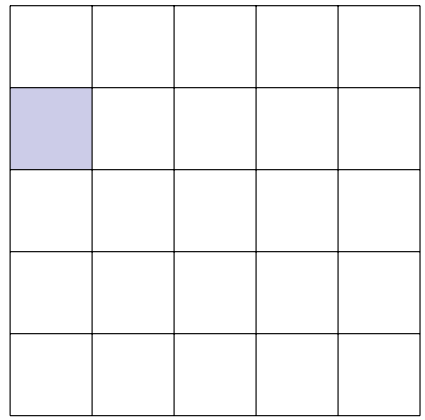
Output

Figures taken from [Fleuret, 2022]

# Pooling



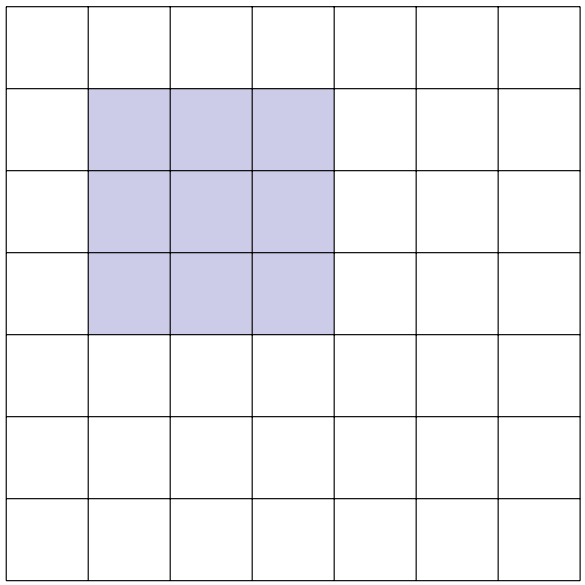
Input



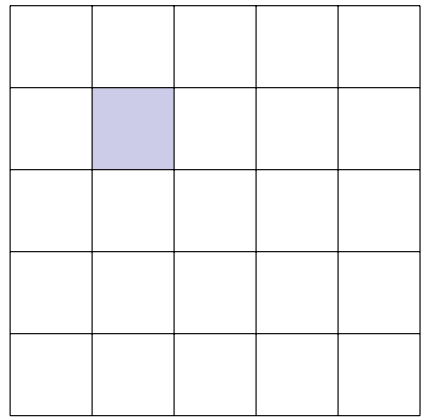
Output

Figures taken from [Fleuret, 2022]

# Pooling



Input



Output

Figures taken from [Fleuret, 2022]

# Pooling

- Étant donné une matrice d'*input* $\boldsymbol{X}$ de taille $H \times W$ et un *filtre de pooling* $\boldsymbol{P}$ de taille $h \times w$, les opérations de **maximum and average pooling**

$$\text{MaxPool}(\boldsymbol{X}) \quad \text{et} \quad \text{AvgPool}(\boldsymbol{X})$$

donnent des matrices de taille $(H-h+1) \times (W-w+1)$ dont les composantes sont

$$\begin{aligned} \text{MaxPool}(\boldsymbol{X})_{i,j} &= \max\left(\boldsymbol{X}_{[i:i+h-1,j:j+w-1]}\right) \\ \text{AvgPool}(\boldsymbol{X})_{i,j} &= \text{avg}\left(\boldsymbol{X}_{[i:i+h-1,j:j+w-1]}\right) \end{aligned}$$

- Contrairement aux convolutions, les opérations de pooling ne font intervenir aucun paramètres!

# Pooling

- Étant donné une matrice d'*input* $\boldsymbol{X}$ de taille $H \times W$ et un *filtre de pooling* $\boldsymbol{P}$ de taille $h \times w$, les opérations de **maximum and average pooling**

$$\text{MaxPool}(\boldsymbol{X}) \quad \text{et} \quad \text{AvgPool}(\boldsymbol{X})$$

donnent des matrices de taille $(H-h+1) \times (W-w+1)$ dont les composantes sont

$$\begin{aligned} \text{MaxPool}(\boldsymbol{X})_{i,j} &= \max\left(\boldsymbol{X}_{[i:i+h-1,j:j+w-1]}\right) \\ \text{AvgPool}(\boldsymbol{X})_{i,j} &= \text{avg}\left(\boldsymbol{X}_{[i:i+h-1,j:j+w-1]}\right) \end{aligned}$$

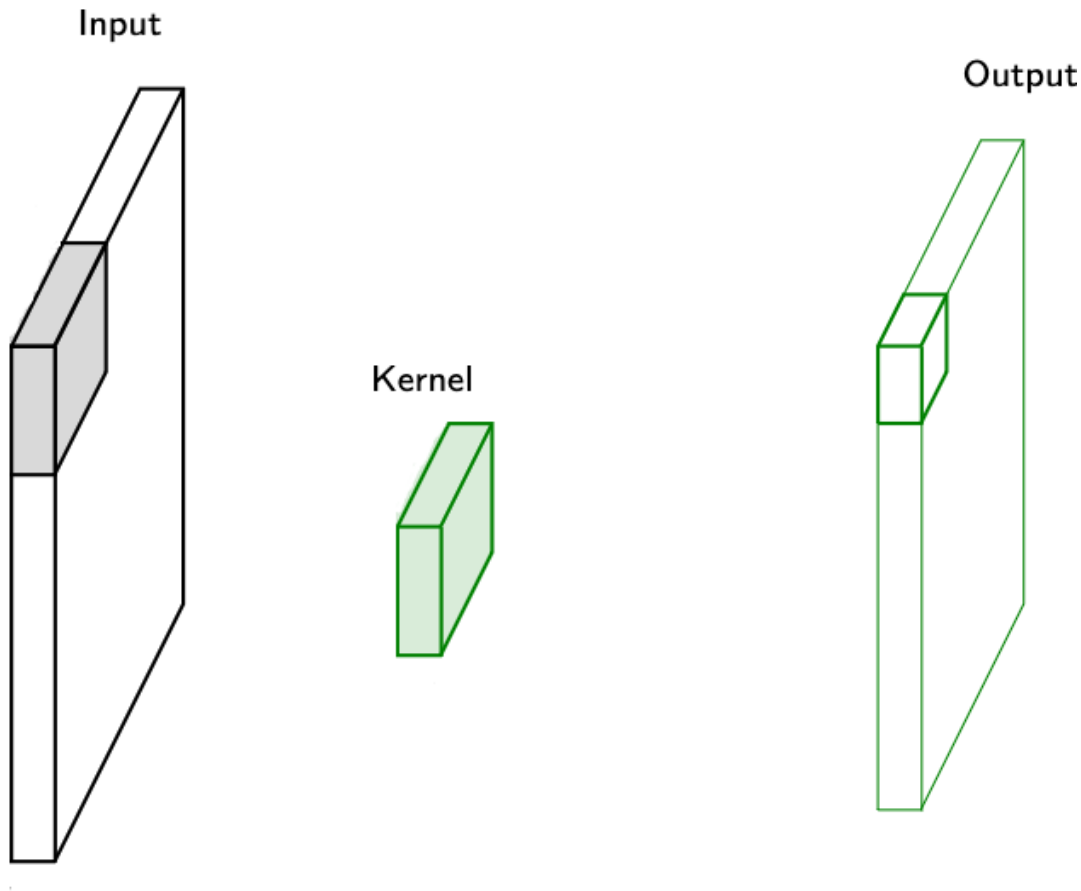
- Contrairement aux convolutions, les opérations de pooling ne font intervenir aucun paramètres!

# Pooling

- Exemples:

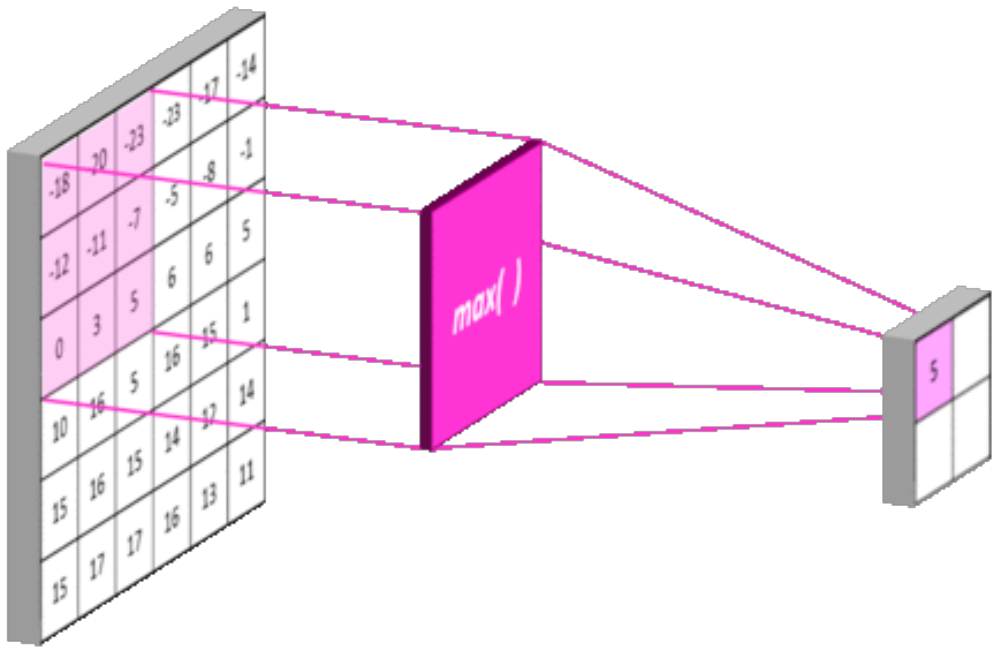
  Voir site web: https://deeplizard.com/resource/pavq7noze3

# Couche de pooling (pooling layer)

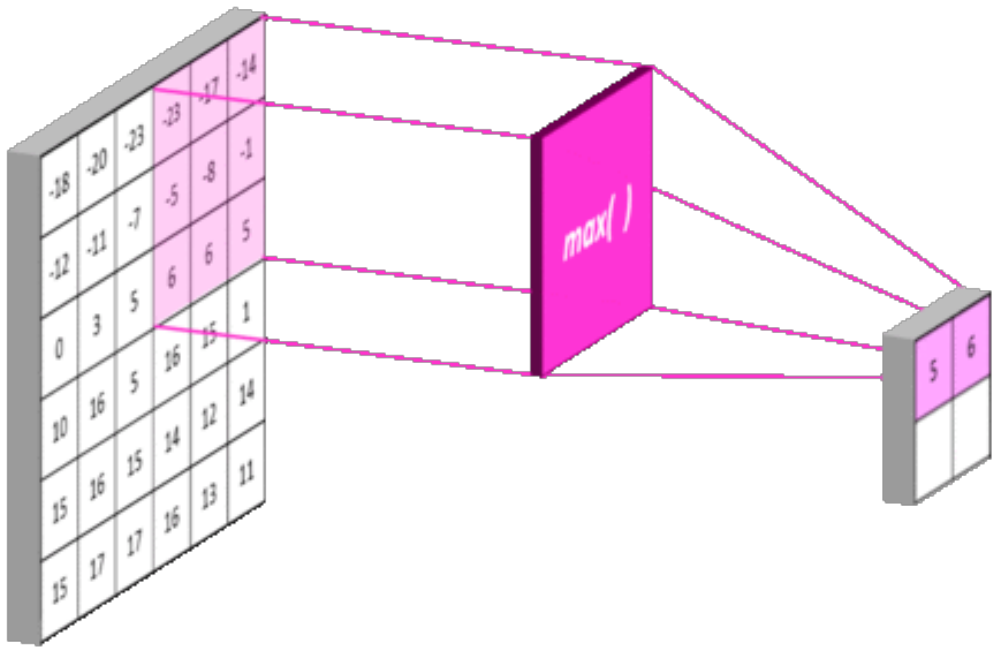



Figures taken from [Fleuret, 2022].

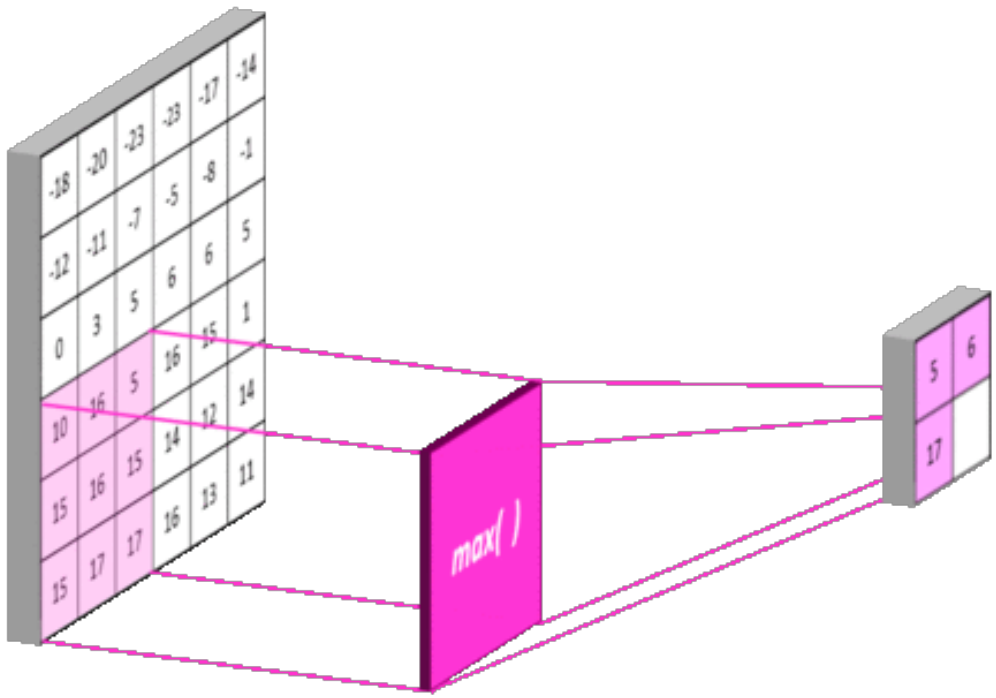
# Couche de pooling (pooling layer)



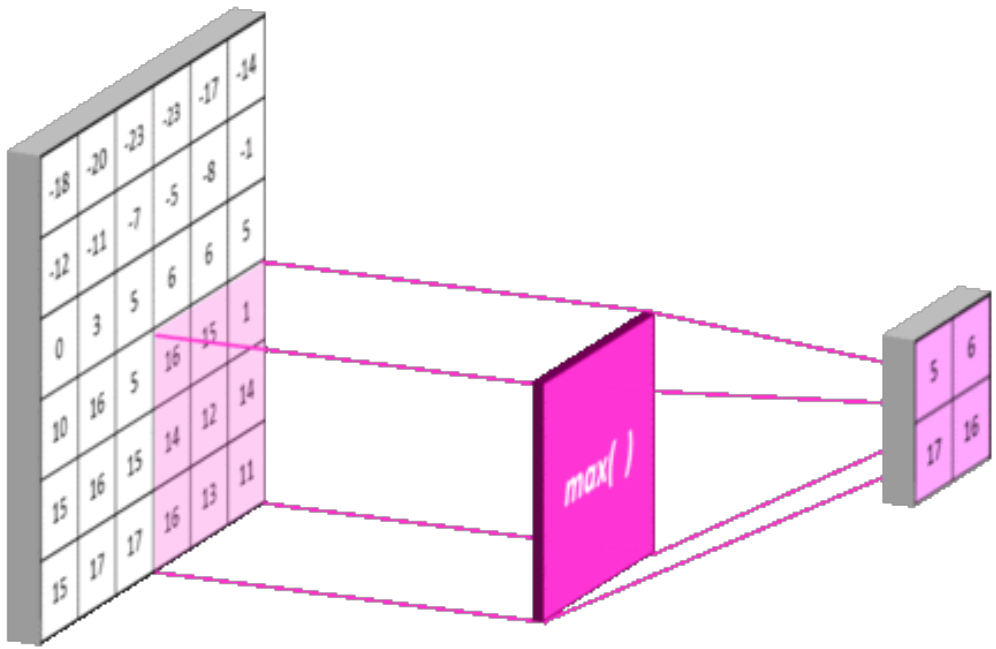
# Couche de pooling (pooling layer)



# Couche de pooling (pooling layer)



# Couche de pooling (pooling layer)



# Couche de pooling (pooling layer)

- Une couche de pooling est appelée *pooling layer*.
- Les deux principaux types de pooling sont le *max pooling* et le *average pooling*, mais il en existe d'autres.
- Une couche de pooling ne fait intervenir aucun paramètre; il n'y a donc aucun paramètre à apprendre!

# Couche de pooling (pooling layer)

- Une couche de pooling est appelée *pooling layer*.
- Les deux principaux types de pooling sont le *max pooling* et le *average pooling*, mais il en existe d'autres.
- Une couche de pooling ne fait intervenir aucun paramètre; il n'y a donc aucun paramètre à apprendre!

# Couche de pooling (pooling layer)

- Une couche de pooling est appelée *pooling layer*.
- Les deux principaux types de pooling sont le *max pooling* et le *average pooling*, mais il en existe d'autres.
- **Une couche de pooling ne fait intervenir aucun paramètre; il n'y a donc aucun paramètre à apprendre!**

# Couche de pooling (pooling layer)

- Le max ou avg pooling appliqué à une input $\boldsymbol{X}$ de dimension $H \times W \times C$ par le biais d'un filtre de dimension $h \times w$ donne une output

$$\boldsymbol{Z} = \text{MaxPool}(\boldsymbol{X}) \quad \text{ou} \quad \boldsymbol{Z} = \text{AvgPool}(\boldsymbol{X})$$

de dimension $(H - h + 1) \times (W - w + 1) \times C$, respectivement.

- Cette fois-ci, l'opération de pooling est appliquée canal par canal. Ainsi, l'output est de même profondeur $C$ que l'input (cf. slide suivant)!

# Couche de pooling (pooling layer)

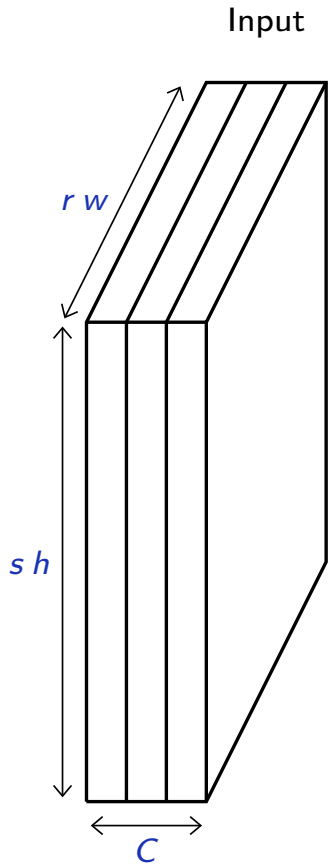
- Le max ou avg pooling appliqué à une input $\boldsymbol{X}$ de dimension $H \times W \times C$ par le biais d'un filtre de dimension $h \times w$ donne une output

$$\boldsymbol{Z} = \mathrm{MaxPool}(\boldsymbol{X}) \quad \text{ou} \quad \boldsymbol{Z} = \mathrm{AvgPool}(\boldsymbol{X})$$

de dimension $(H - h + 1) \times (W - w + 1) \times C$, respectivement.

- Cette fois-ci, l'opération de pooling est appliquée canal par canal. Ainsi, l'output est de même profondeur $C$ que l'input (cf. slide suivant)!

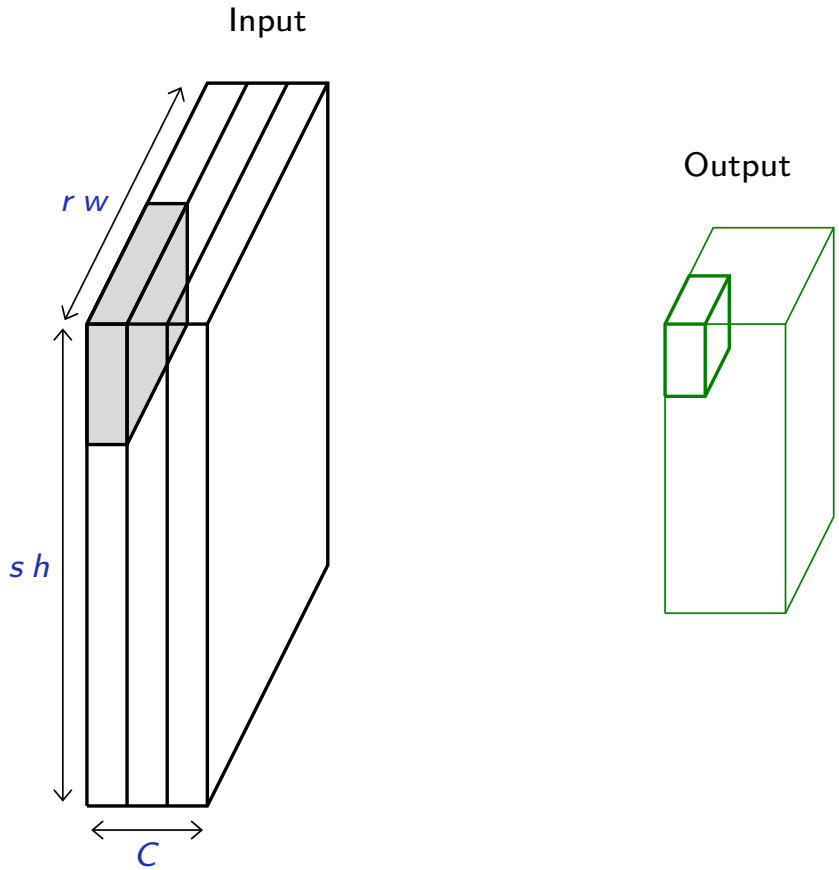
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

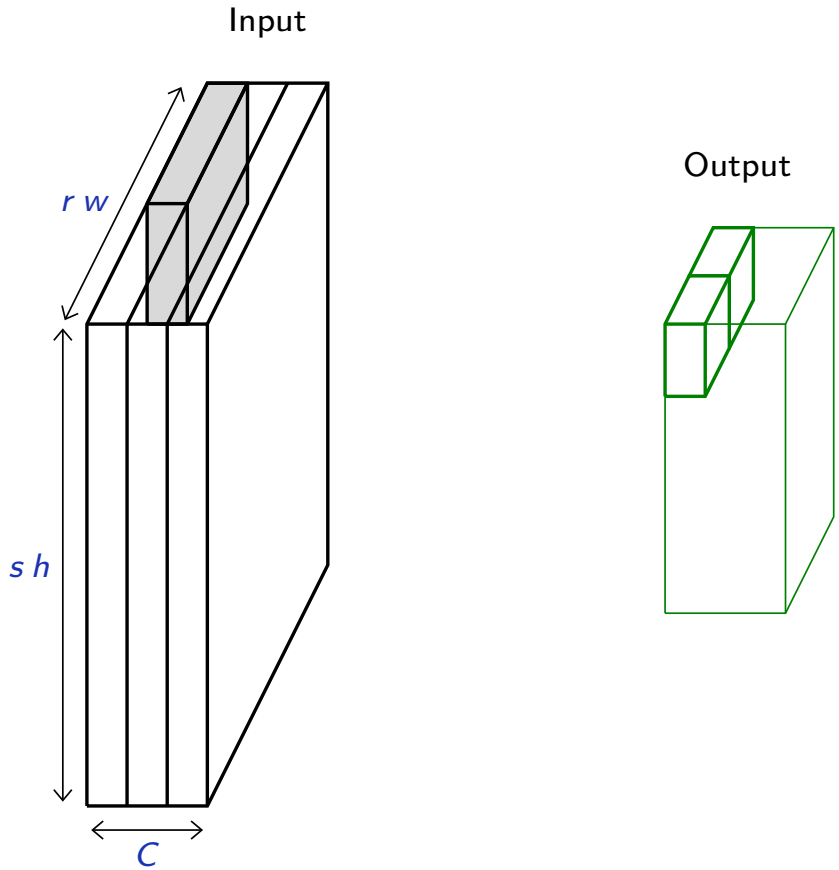
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

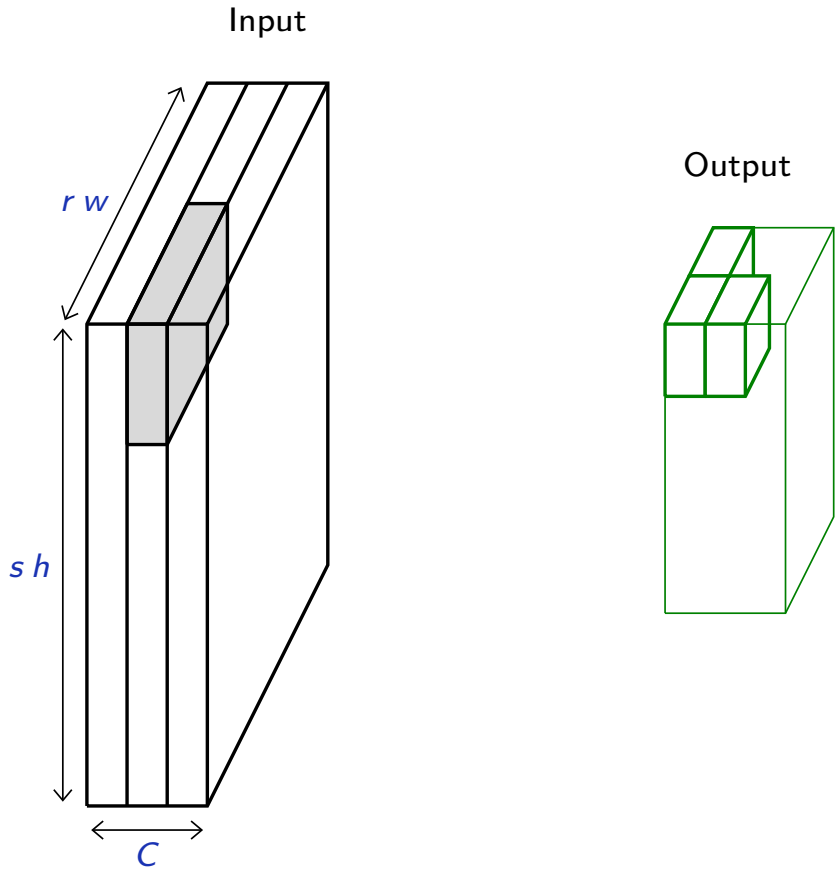
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

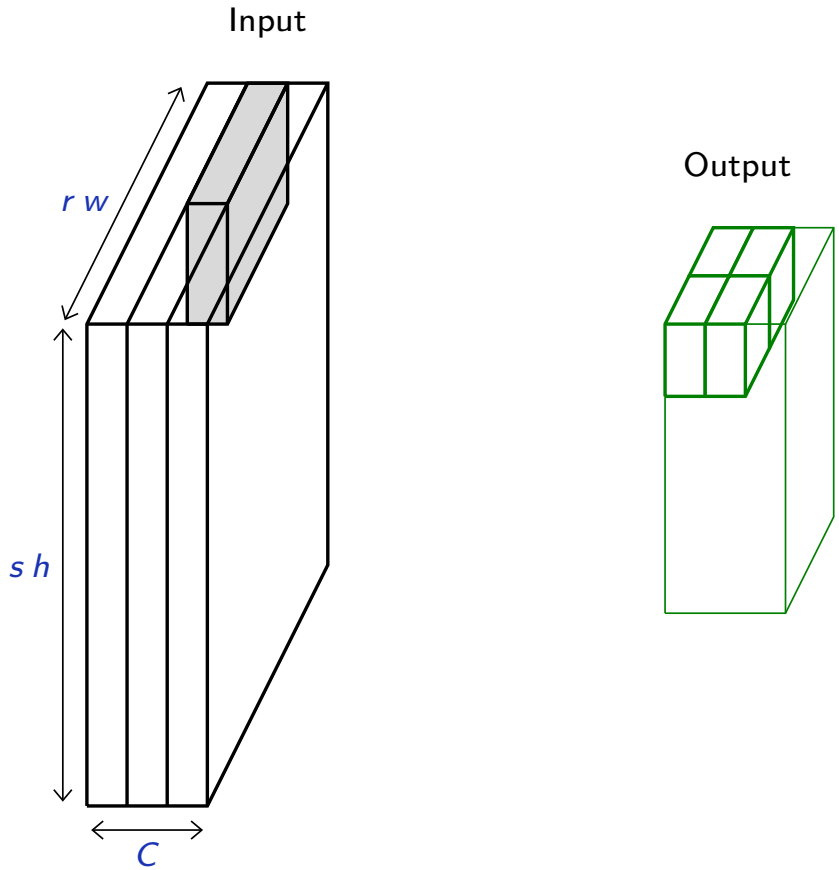
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

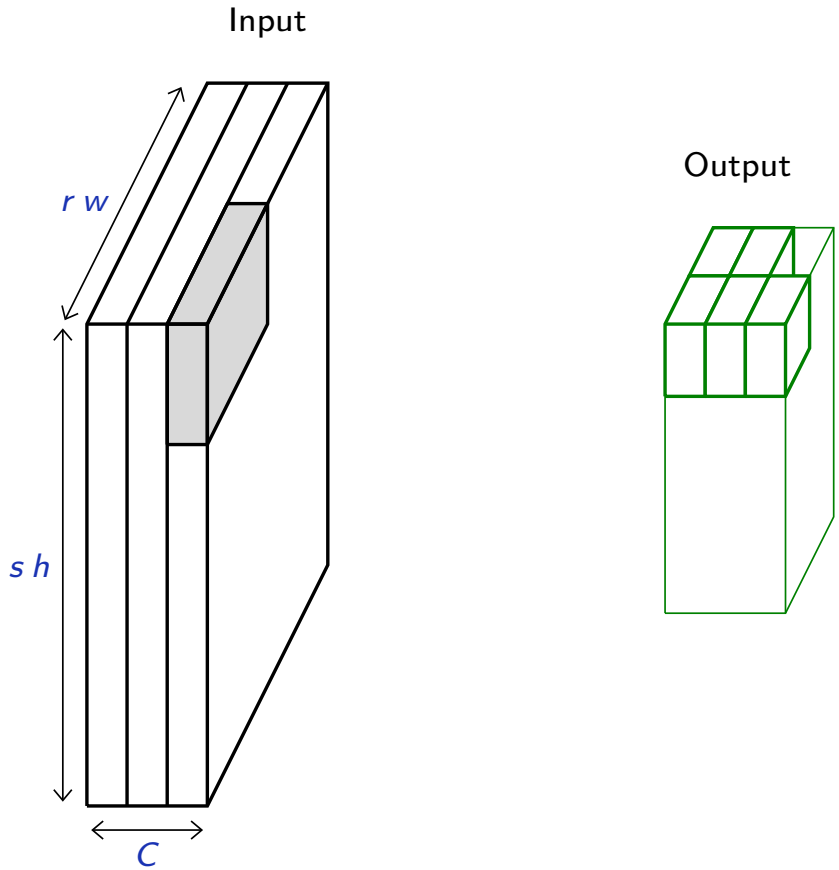
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

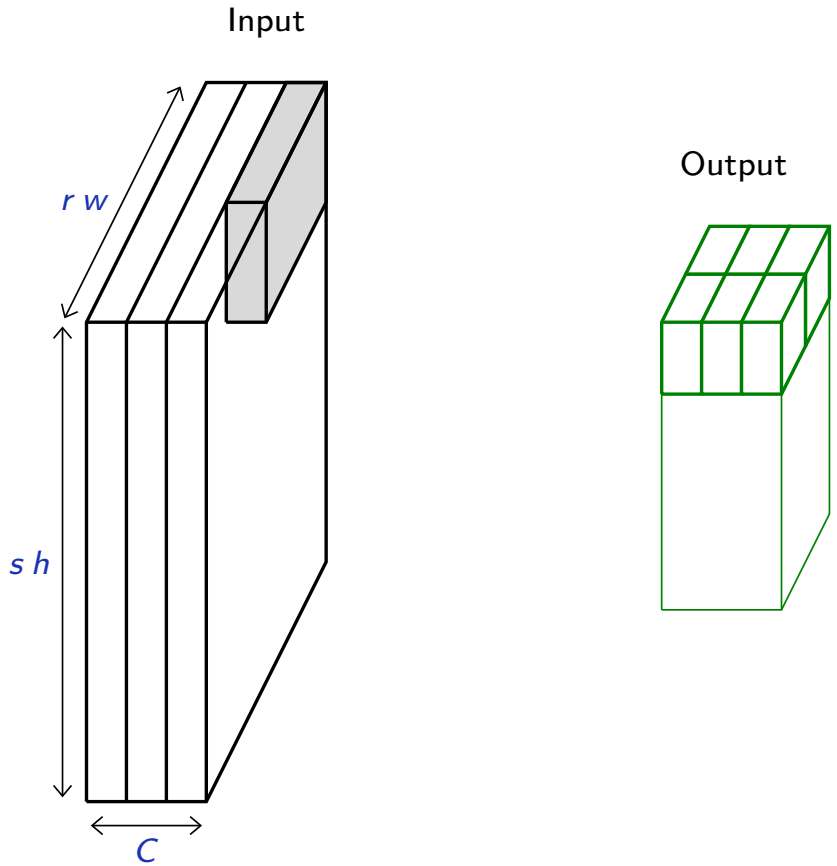
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

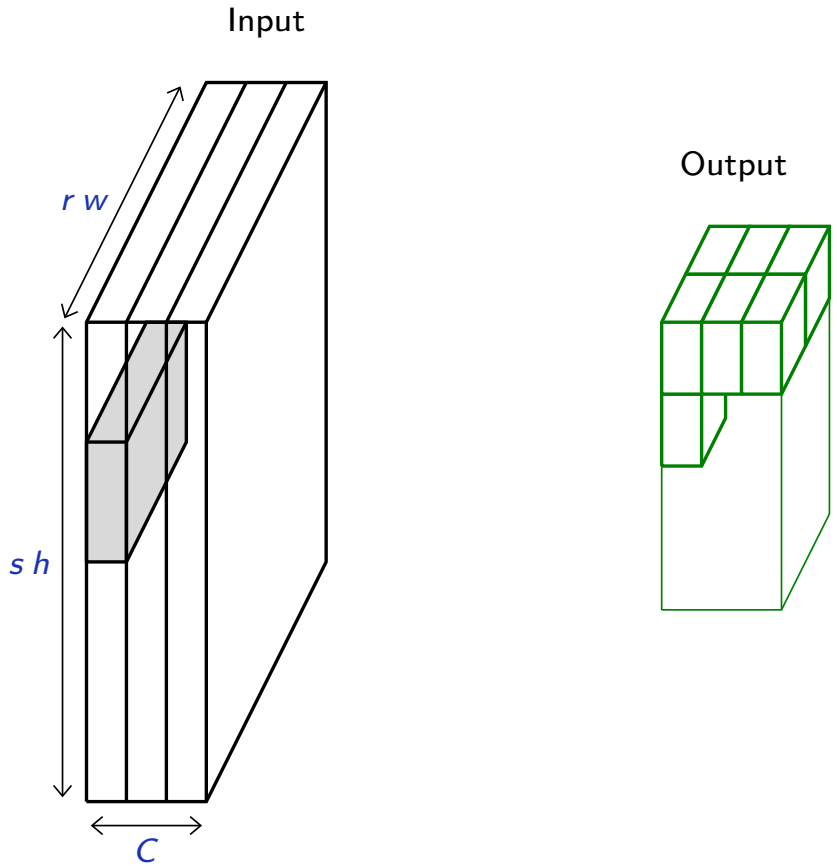
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

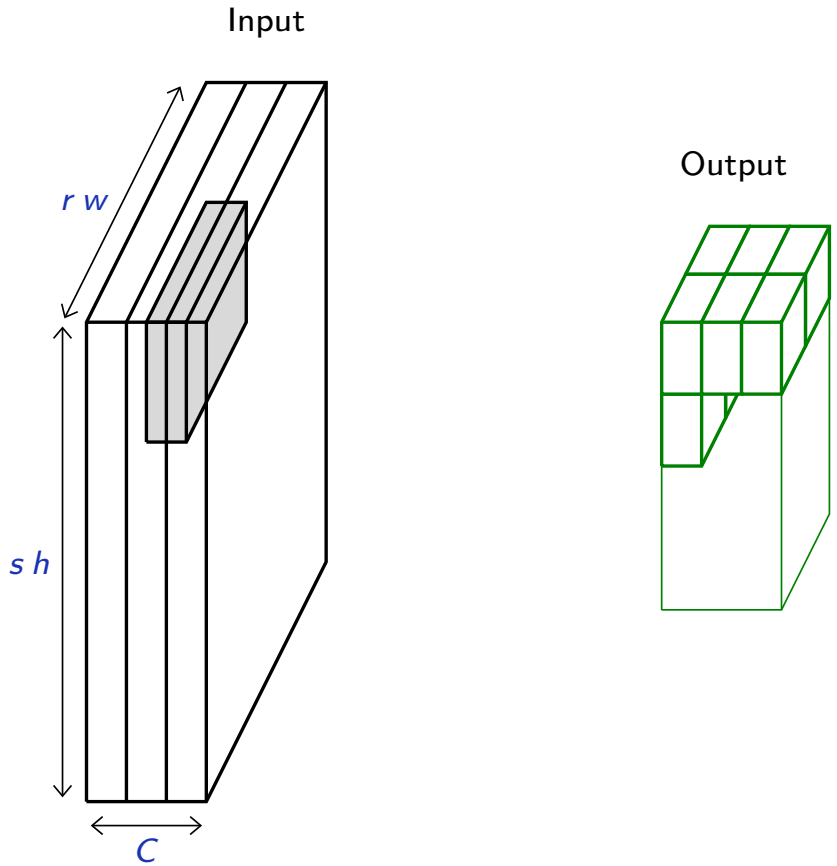
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

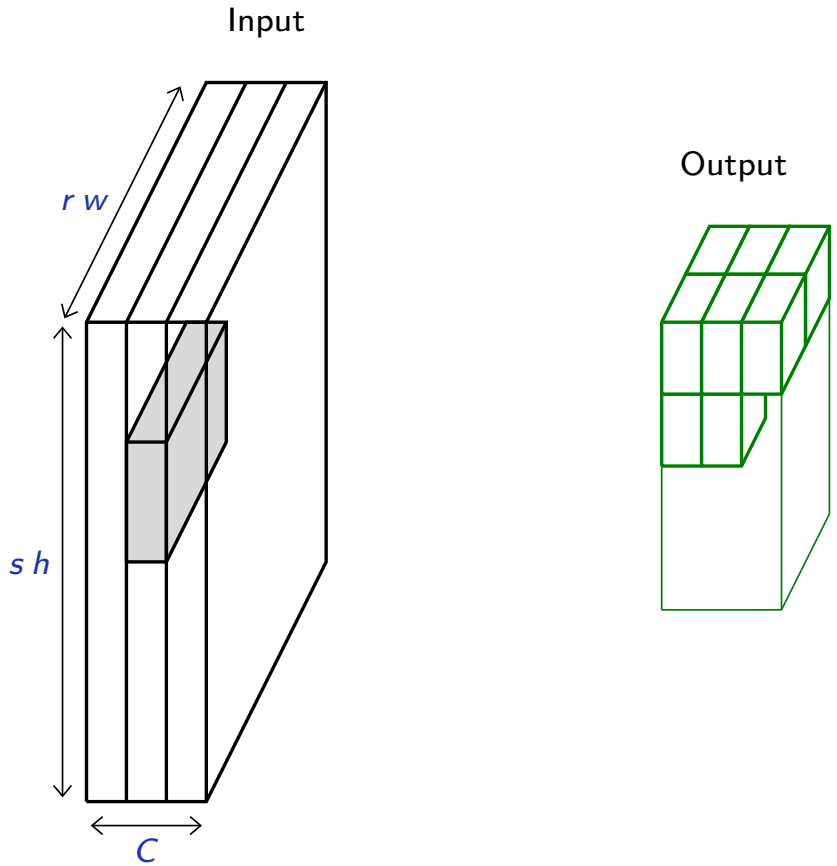
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

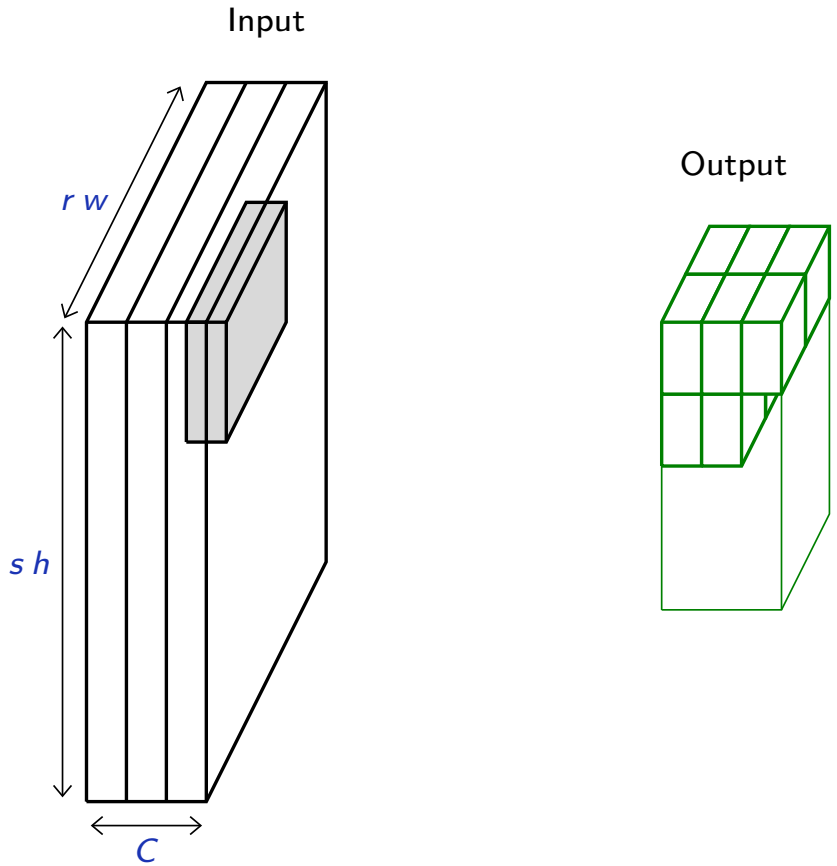
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

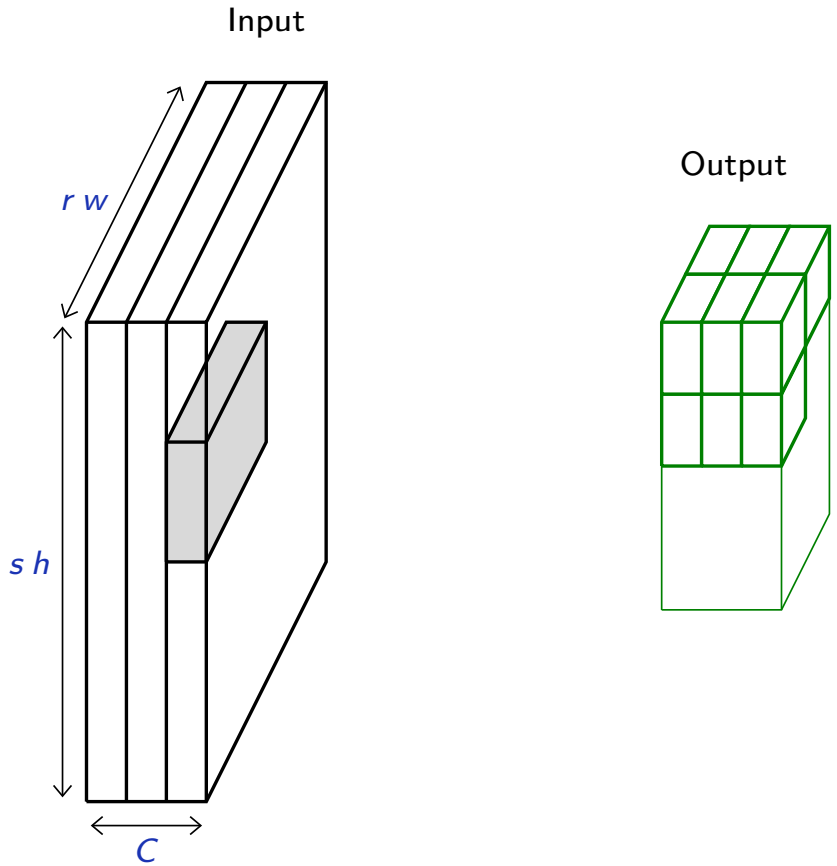
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

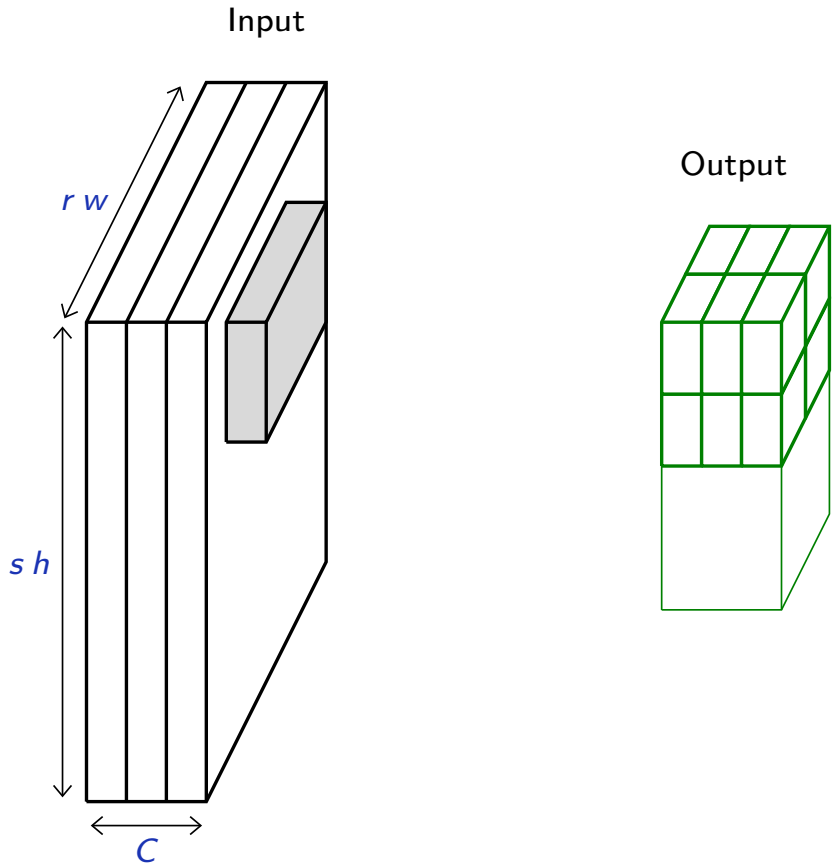
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

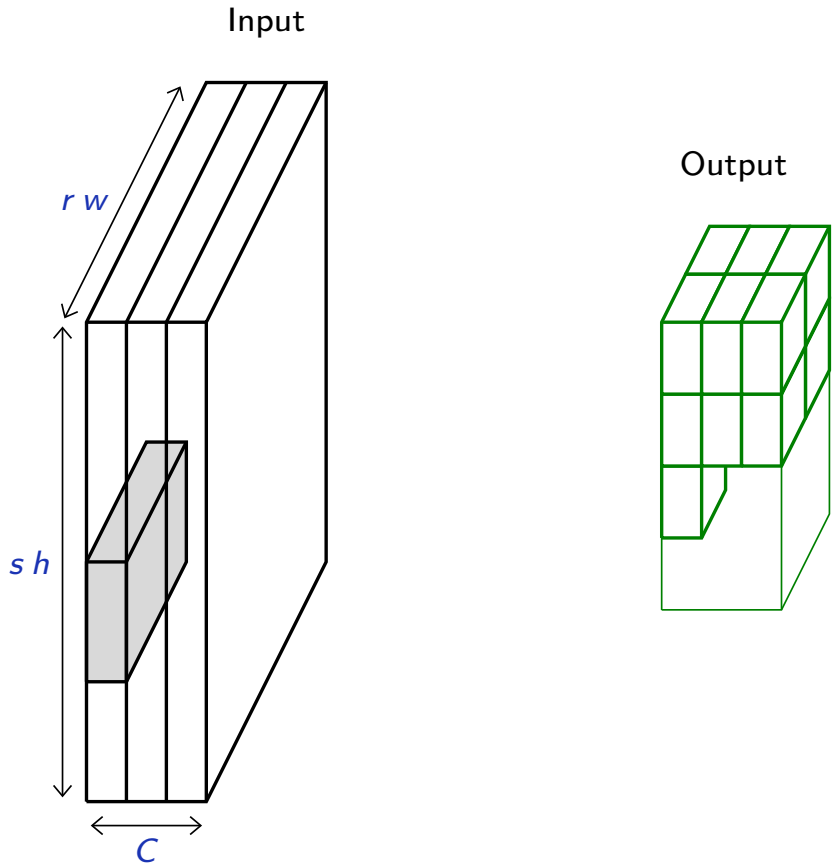
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

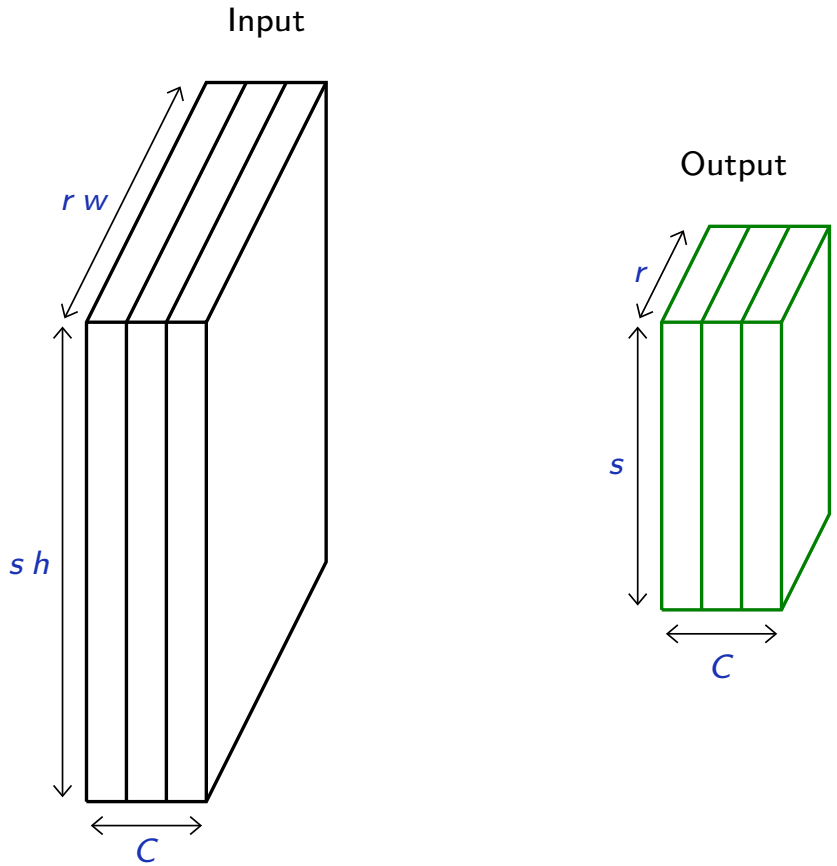
# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

# Couche de pooling (pooling layer)




Figures taken from [Fleuret, 2022]

- Contrairement à la convolution, le pooling est appliqué indépendamment sur chaque canal, donnant une output de profondeur $C$.

# Rôle des couches pooling

Les couches de pooling jouent les rôle suivants:

- **Aggregate information**

  Un groupe de pixels est agrégé en un seul pixel d'information.

- **Dimension reduction**

  Une couche de pooling réduit la dimension de l'input, et donc l'efficacité computationnelle.

- **Procuce translation invariance**

  Si l'input (ou des patterns de l'input) sont légèrement translatés, leur "pooled representation" restera très similaire.

# Rôle des couches pooling

Les couches de pooling jouent les rôle suivants:

- **Aggregate information**

  Un groupe de pixels est agrégé en un seul pixel d'information.

- **Dimension reduction**

  Une couche de pooling réduit la dimension de l'input, et donc l'efficacité computationnelle.

- **Procuce translation invariance**

  Si l'input (ou des patterns de l'input) sont légèrement translatés, leur "pooled representation" restera très similaire.

# Rôle des couches pooling

Les couches de pooling jouent les rôle suivants:

- **Aggregate information**

  Un groupe de pixels est agrégé en un seul pixel d'information.

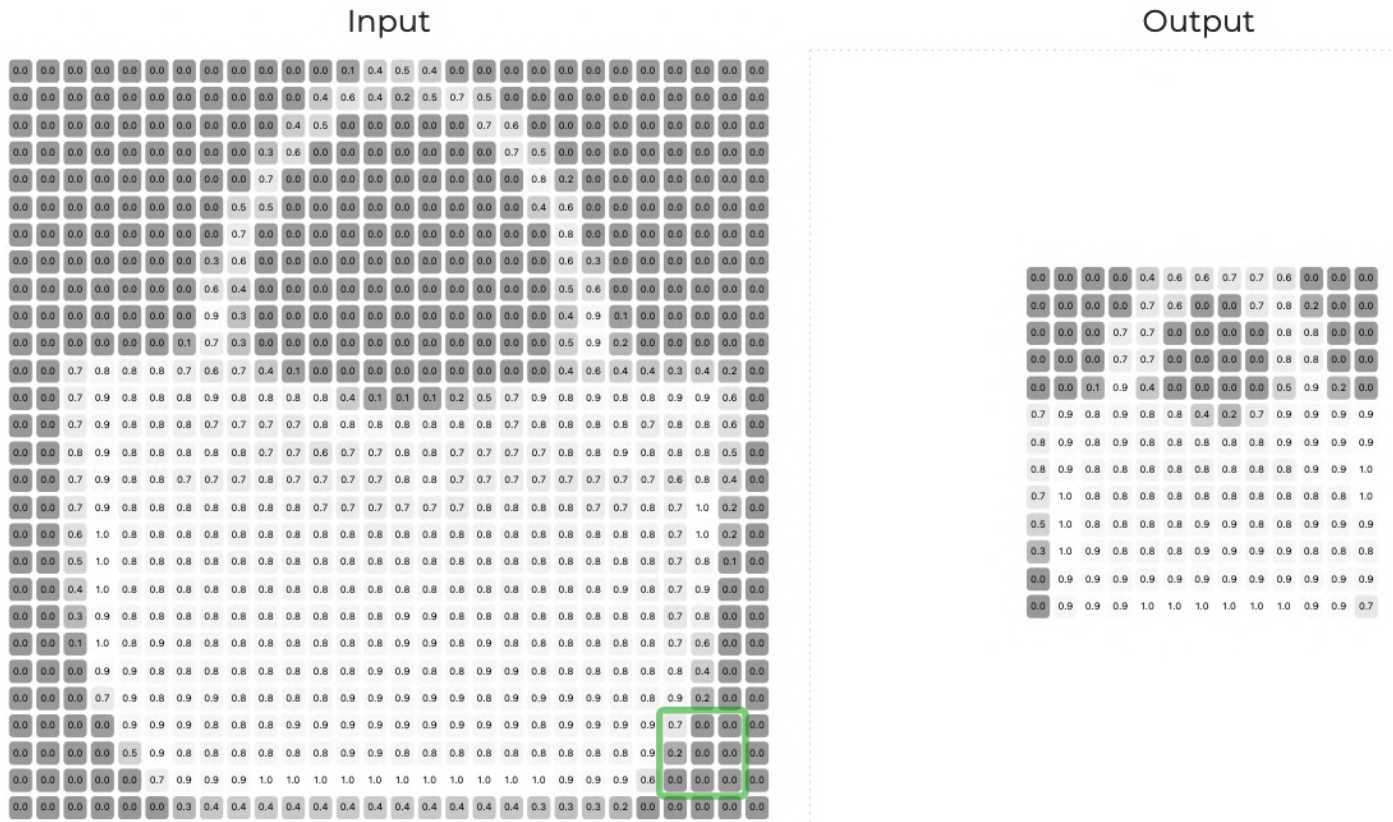
- **Dimension reduction**

  Une couche de pooling réduit la dimension de l'input, et donc l'efficacité computationnelle.

- **Procuce translation invariance**

  Si l'input (ou des patterns de l'input) sont légèrement translatés, leur "pooled representation" restera très similaire.

# Rôle des couches pooling



# Réseau de neurones convolutif (CNNs)

- Un **réseau de neurones convolutif (CNN)** est composé de couches convolutives, de couches pooling et en général de quelques dernières couches denses classiques.
- Les paramètres (poids) des filtres convolutifs (et des couches denses classiques s'il y en a) sont appris par backpropagation!
- Les couches de pooling ne font intervenir aucun paramètre!
- Les filtres sont donc des *features' extractors* et les features qu'ils extraient sont appris par entraînement.
- Un CNN crée une représentation enrichie de l'image de départ (input) qui est en adéquation avec la tâche que l'on souhaite résoudre.

# Réseau de neurones convolutif (CNNs)

- Un **réseau de neurones convolutif (CNN)** est composé de couches convolutives, de couches pooling et en général de quelques dernières couches denses classiques.
- Les paramètres (poids) des filtres convolutifs (et des couches denses classiques s'il y en a) sont appris par backpropagation!
- Les couches de pooling ne font intervenir aucun paramètre!
- Les filtres sont donc des *features' extractors* et les features qu'ils extraient sont appris par entraînement.
- Un CNN crée une représentation enrichie de l'image de départ (input) qui est en adéquation avec la tâche que l'on souhaite résoudre.

# Réseau de neurones convolutif (CNNs)

- Un **réseau de neurones convolutif (CNN)** est composé de couches convolutives, de couches pooling et en général de quelques dernières couches denses classiques.
- Les paramètres (poids) des filtres convolutifs (et des couches denses classiques s'il y en a) sont appris par backpropagation!
- Les couches de pooling ne font intervenir aucun paramètre!
- Les filtres sont donc des *features' extractors* et les features qu'ils extraient sont appris par entraînement.
- Un CNN crée une représentation enrichie de l'image de départ (input) qui est en adéquation avec la tâche que l'on souhaite résoudre.

# Réseau de neurones convolutif (CNNs)

- Un **réseau de neurones convolutif (CNN)** est composé de couches convolutives, de couches pooling et en général de quelques dernières couches denses classiques.
- Les paramètres (poids) des filtres convolutifs (et des couches denses classiques s'il y en a) sont appris par backpropagation!
- Les couches de pooling ne font intervenir aucun paramètre!
- Les filtres sont donc des *features' extractors* et les features qu'ils extraient sont appris par entraînement.
- Un CNN crée une représentation enrichie de l'image de départ (input) qui est en adéquation avec la tâche que l'on souhaite résoudre.

# Réseau de neurones convolutif (CNNs)

- Un **réseau de neurones convolutif (CNN)** est composé de couches convolutives, de couches pooling et en général de quelques dernières couches denses classiques.
- Les paramètres (poids) des filtres convolutifs (et des couches denses classiques s'il y en a) sont appris par backpropagation!
- Les couches de pooling ne font intervenir aucun paramètre!
- Les filtres sont donc des *features' extractors* et les features qu'ils extraient sont appris par entraînement.
- **Un CNN crée une représentation enrichie de l'image de départ (input) qui est en adéquation avec la tâche que l'on souhaite résoudre.**

# AlexNet [Krizhevsky et al., 2012]

- L'architecture AlexNet a représenté une avancée majeure dans le domaine de la computer vision.
- Le modèle se compose 5 couche convolutives, 2 couches de Max Pooling et 3 couche denses.
- Ce réseau a été entraîné pour un problème de classification d'images sur le dataset ImageNet (15M d'images, 22K catégories).
- AlexNet s'entraîne par backpropagation!

# AlexNet [Krizhevsky et al., 2012]

- L'architecture AlexNet a représenté une avancée majeure dans le domaine de la computer vision.
- Le modèle se compose $5$ couche convolutives, $2$ couches de Max Pooling et $3$ couche denses.
- Ce réseau a été entraîné pour un problème de classification d'images sur le dataset ImageNet (15M d'images, 22K catégories).
- AlexNet s'entraîne par backpropagation!

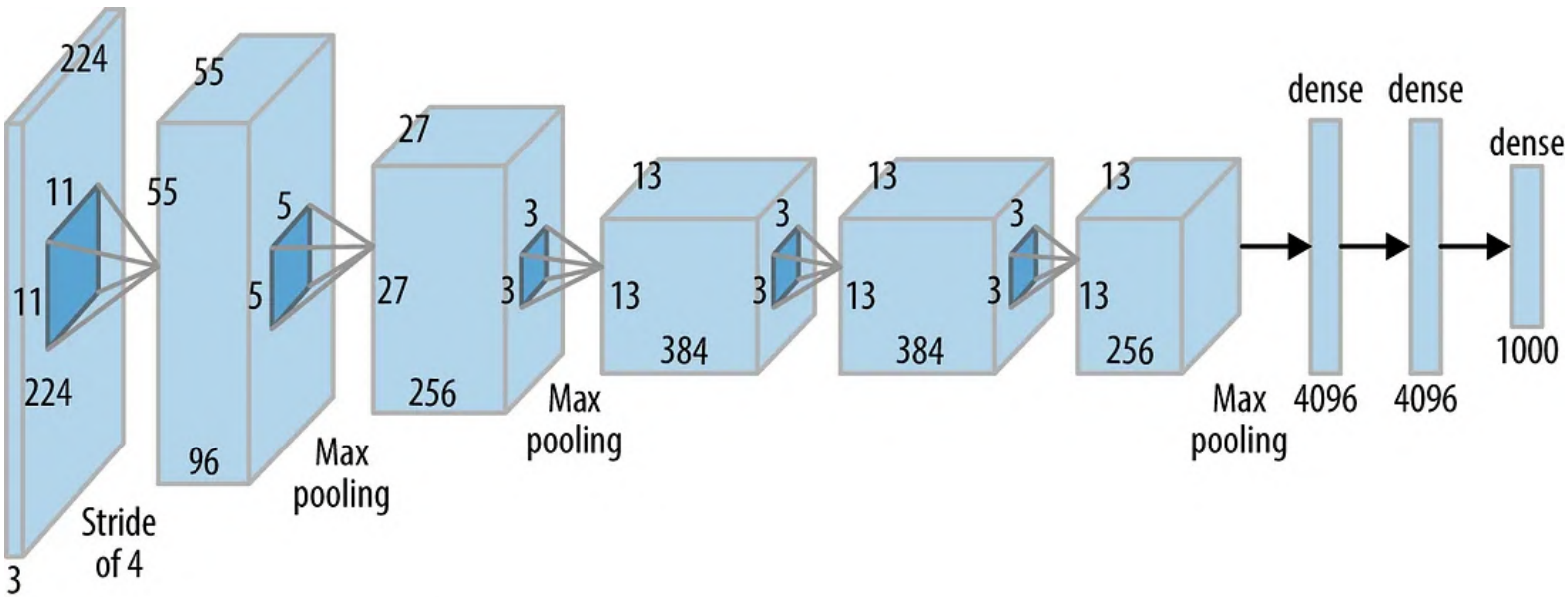
# AlexNet [Krizhevsky et al., 2012]

- L'architecture AlexNet a représenté une avancée majeure dans le domaine de la computer vision.
- Le modèle se compose $5$ couche convolutives, $2$ couches de Max Pooling et $3$ couche denses.
- Ce réseau a été entraîné pour un problème de classification d'images sur le dataset ImageNet (15M d'images, 22K catégories).
- AlexNet s'entraîne par backpropagation!

# AlexNet [Krizhevsky et al., 2012]

- L'architecture AlexNet a représenté une avancée majeure dans le domaine de la computer vision.
- Le modèle se compose $5$ couche convolutives, $2$ couches de Max Pooling et $3$ couche denses.
- Ce réseau a été entraîné pour un problème de classification d'images sur le dataset ImageNet (15M d'images, 22K catégories).
- AlexNet s'entraîne par backpropagation!

# AlexNet




Figures taken from

https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

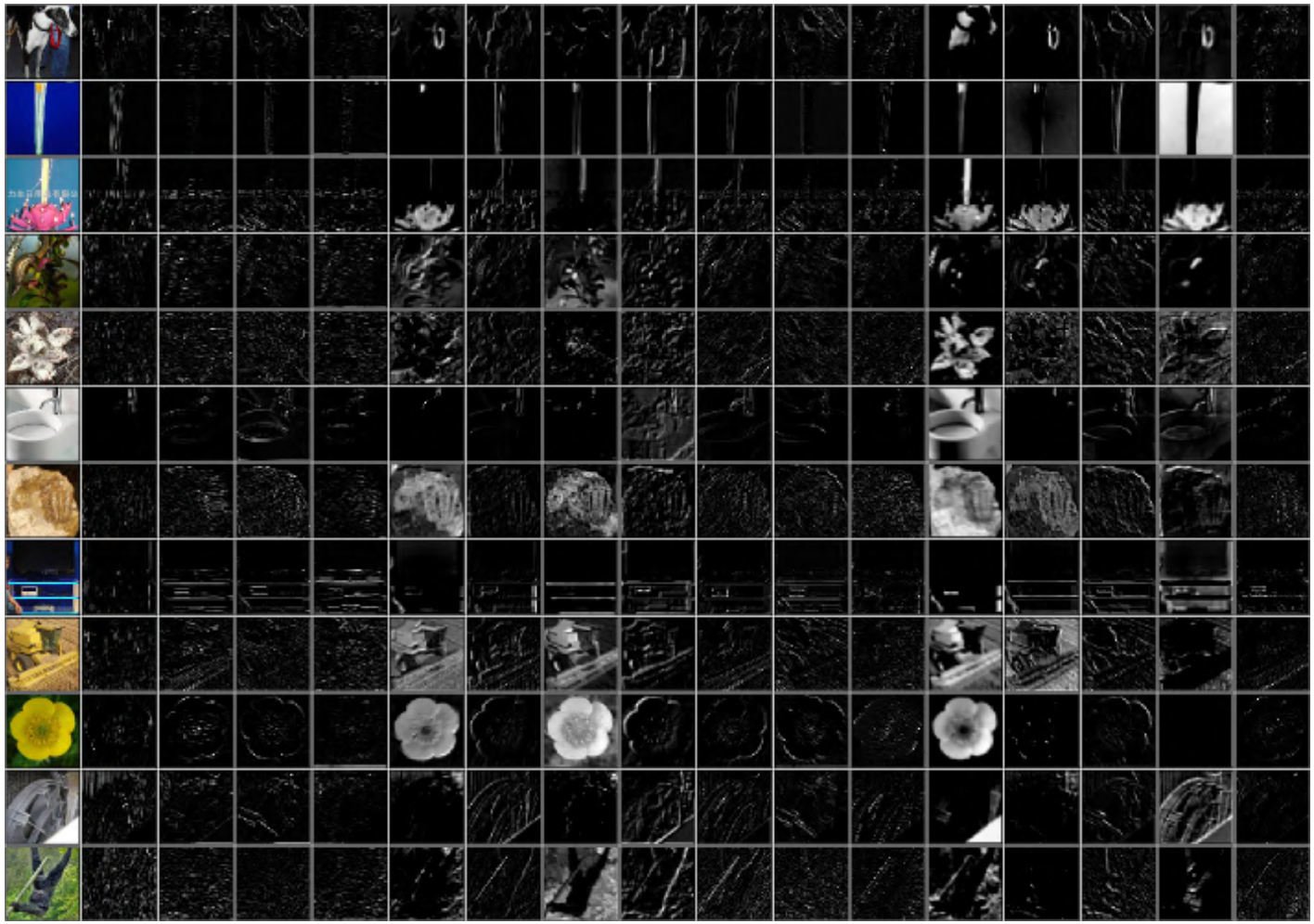
# AlexNet

| AlexNet Network - Structural Details | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Input | | | Output | | | Layer | Stride | Pad | Kernel size | | in | out | # of Param |
| 227 | 227 | 3 | 55 | 55 | 96 | conv1 | 4 | 0 | 11 | 11 | 3 | 96 | 34944 |
| 55 | 55 | 96 | 27 | 27 | 96 | maxpool1 | 2 | 0 | 3 | 3 | 96 | 96 | 0 |
| 27 | 27 | 96 | 27 | 27 | 256 | conv2 | 1 | 2 | 5 | 5 | 96 | 256 | 614656 |
| 27 | 27 | 256 | 13 | 13 | 256 | maxpool2 | 2 | 0 | 3 | 3 | 256 | 256 | 0 |
| 13 | 13 | 256 | 13 | 13 | 384 | conv3 | 1 | 1 | 3 | 3 | 256 | 384 | 885120 |
| 13 | 13 | 384 | 13 | 13 | 384 | conv4 | 1 | 1 | 3 | 3 | 384 | 384 | 1327488 |
| 13 | 13 | 384 | 13 | 13 | 256 | conv5 | 1 | 1 | 3 | 3 | 384 | 256 | 884992 |
| 13 | 13 | 256 | 6 | 6 | 256 | maxpool5 | 2 | 0 | 3 | 3 | 256 | 256 | 0 |
| | | | | | | fc6 | | | 1 | 1 | 9216 | 4096 | 37752832 |
| | | | | | | fc7 | | | 1 | 1 | 4096 | 4096 | 16781312 |
| | | | | | | fc8 | | | 1 | 1 | 4096 | 1000 | 4097000 |
| **Total** | | | | | | | | | | | | | **62,378,344** |

Figures taken from

https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

# AlexNet: convolution and pooling layers

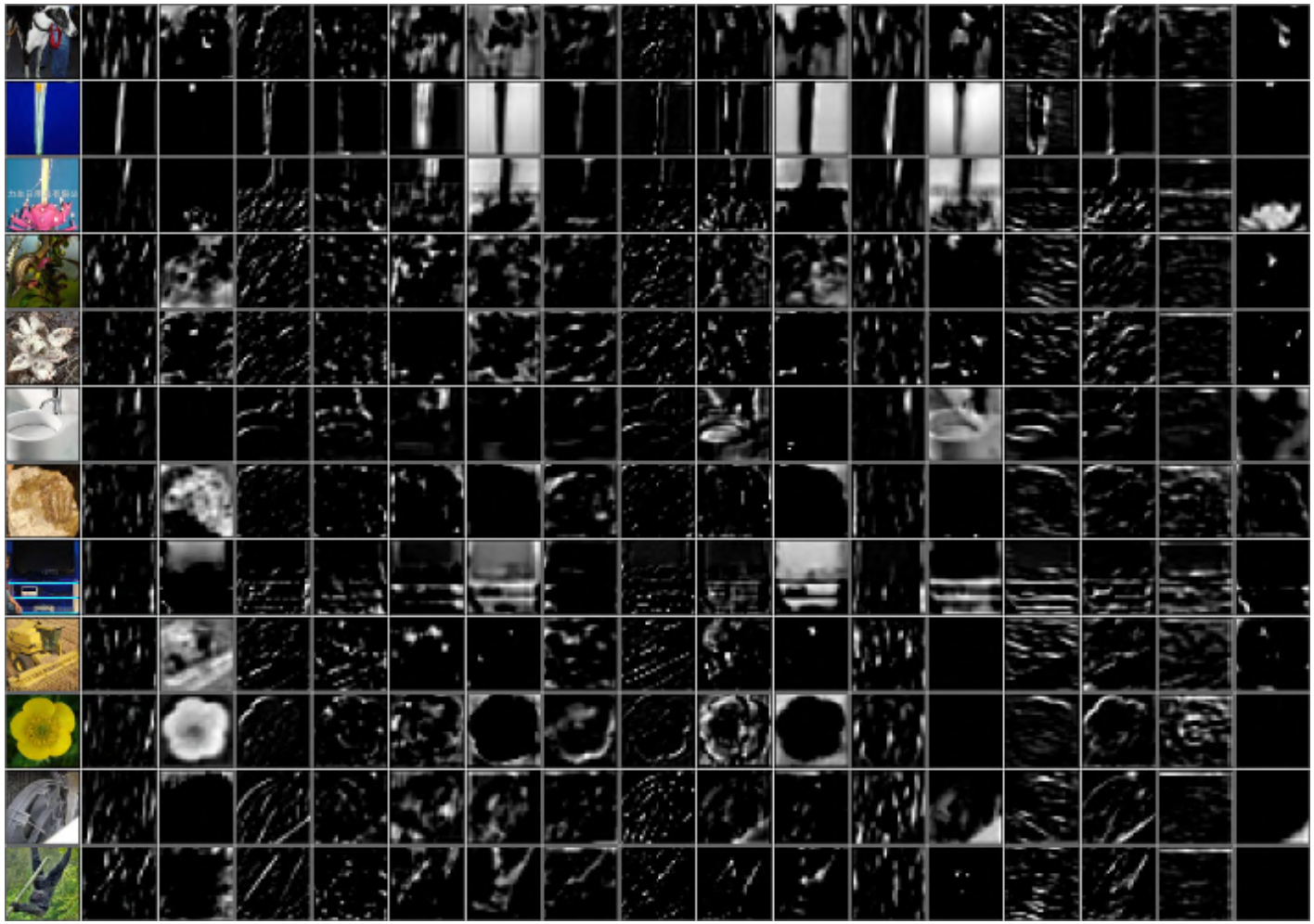


Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# AlexNet: convolution and pooling layers

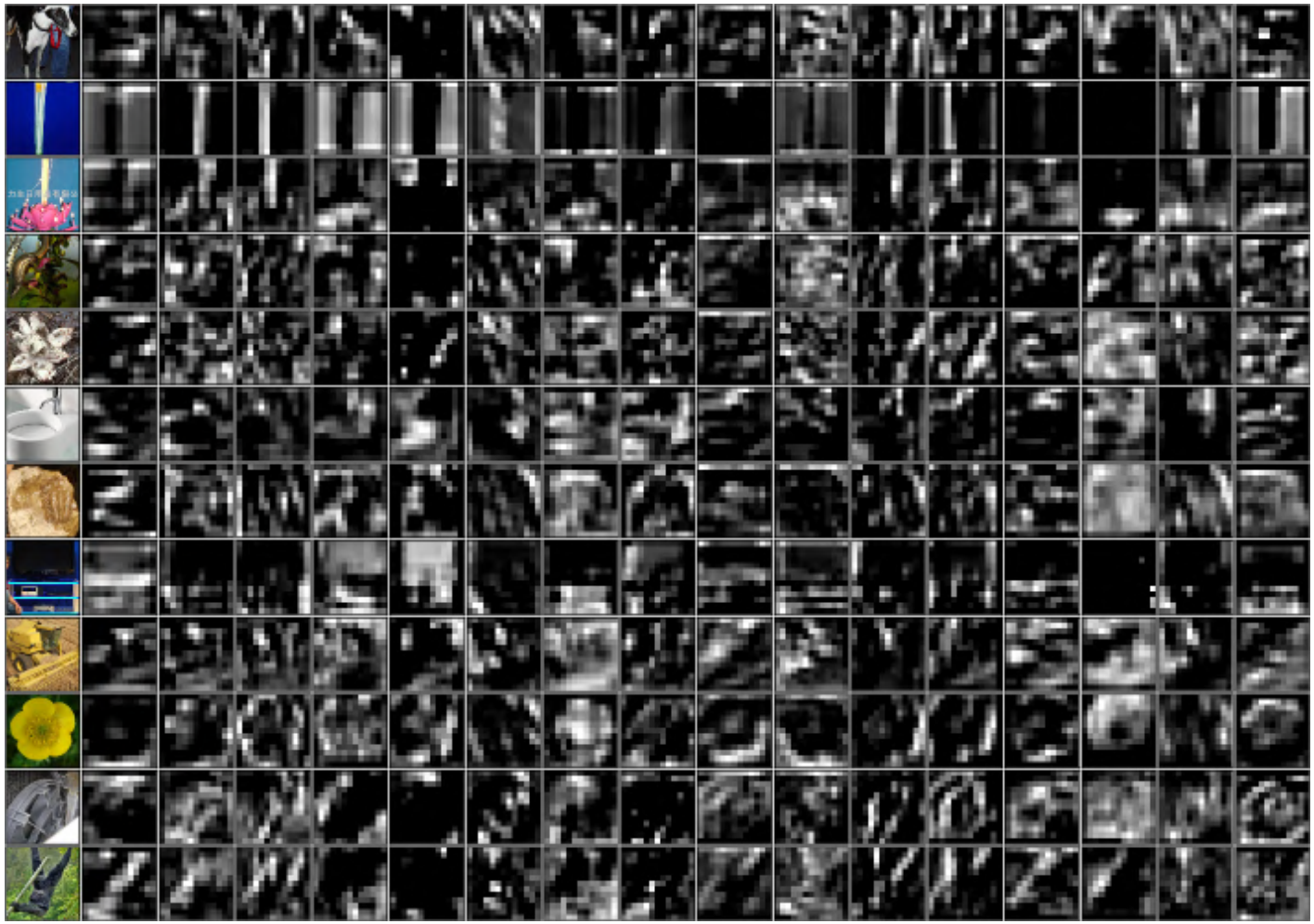


Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# AlexNet: convolution and pooling layers

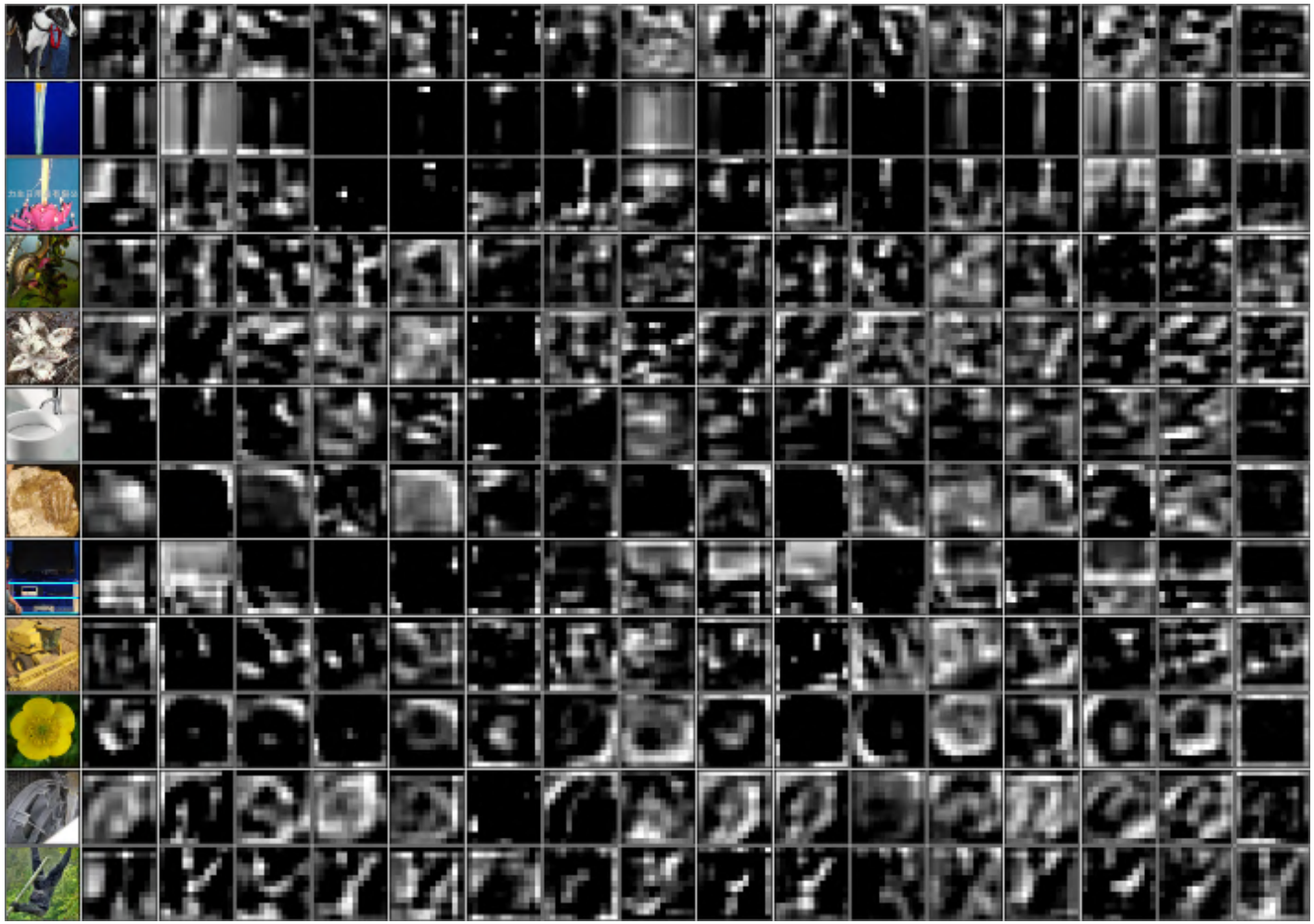


Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# AlexNet: convolution and pooling layers

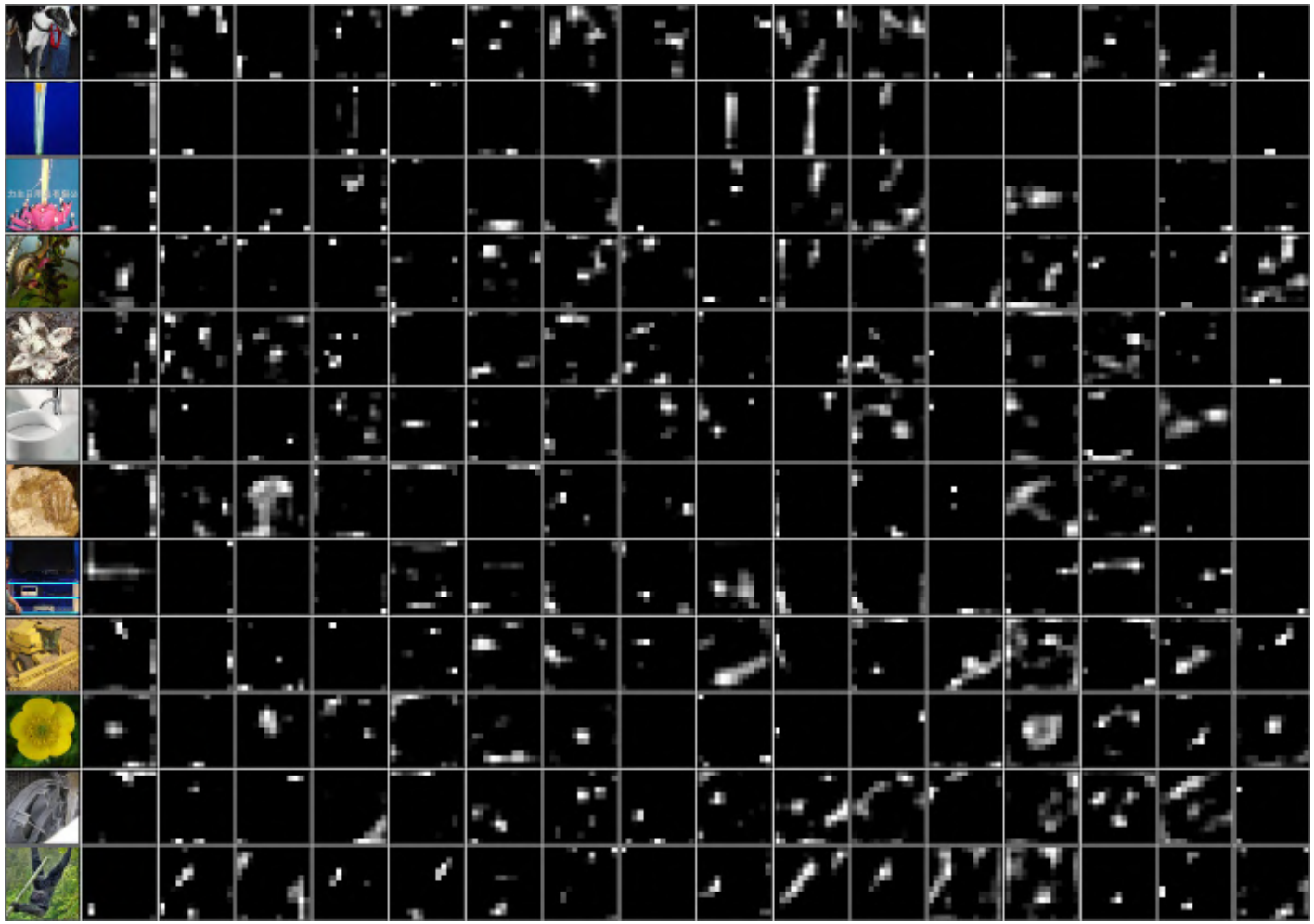


Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# AlexNet: convolution and pooling layers

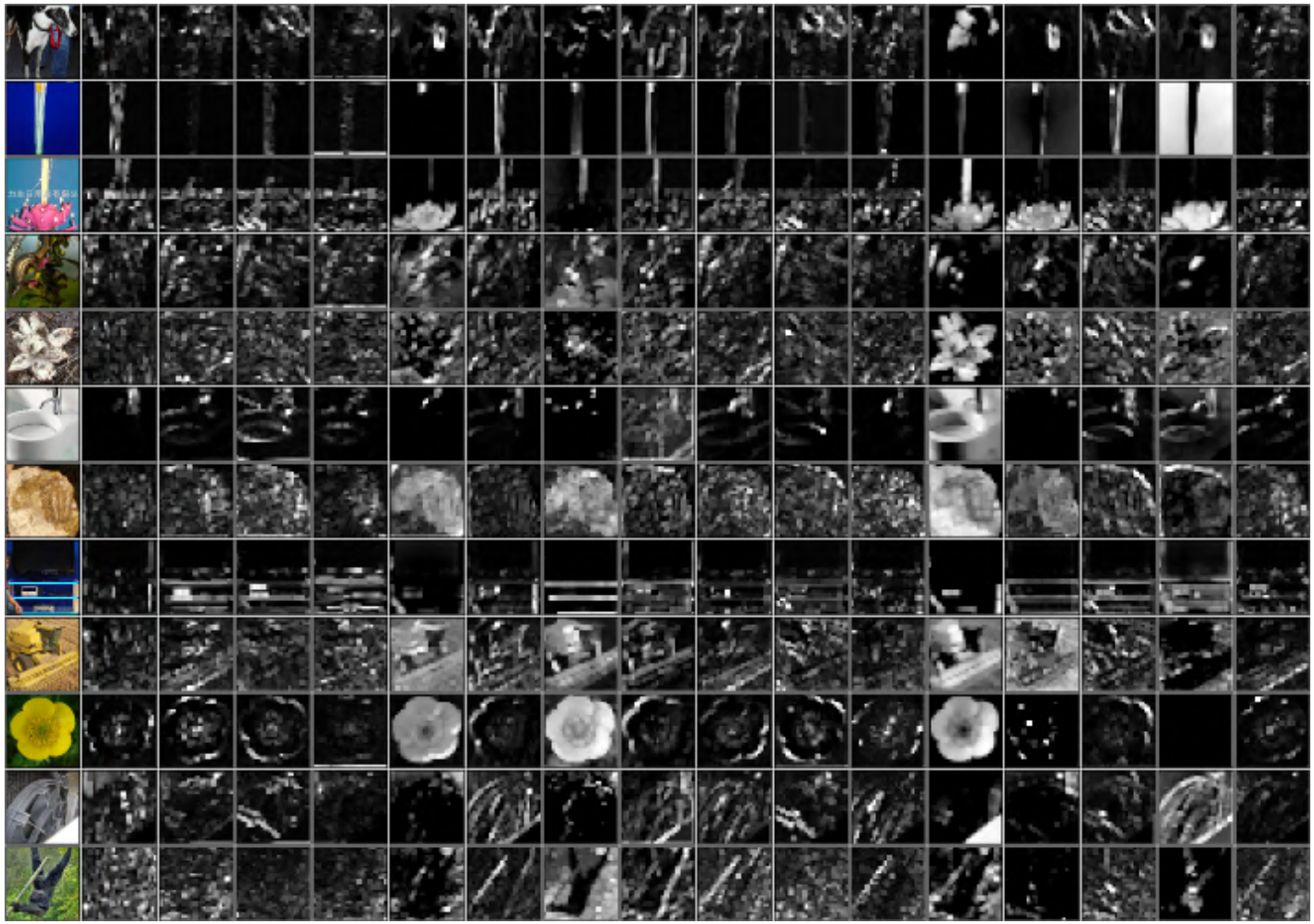


Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# AlexNet: convolution and pooling layers

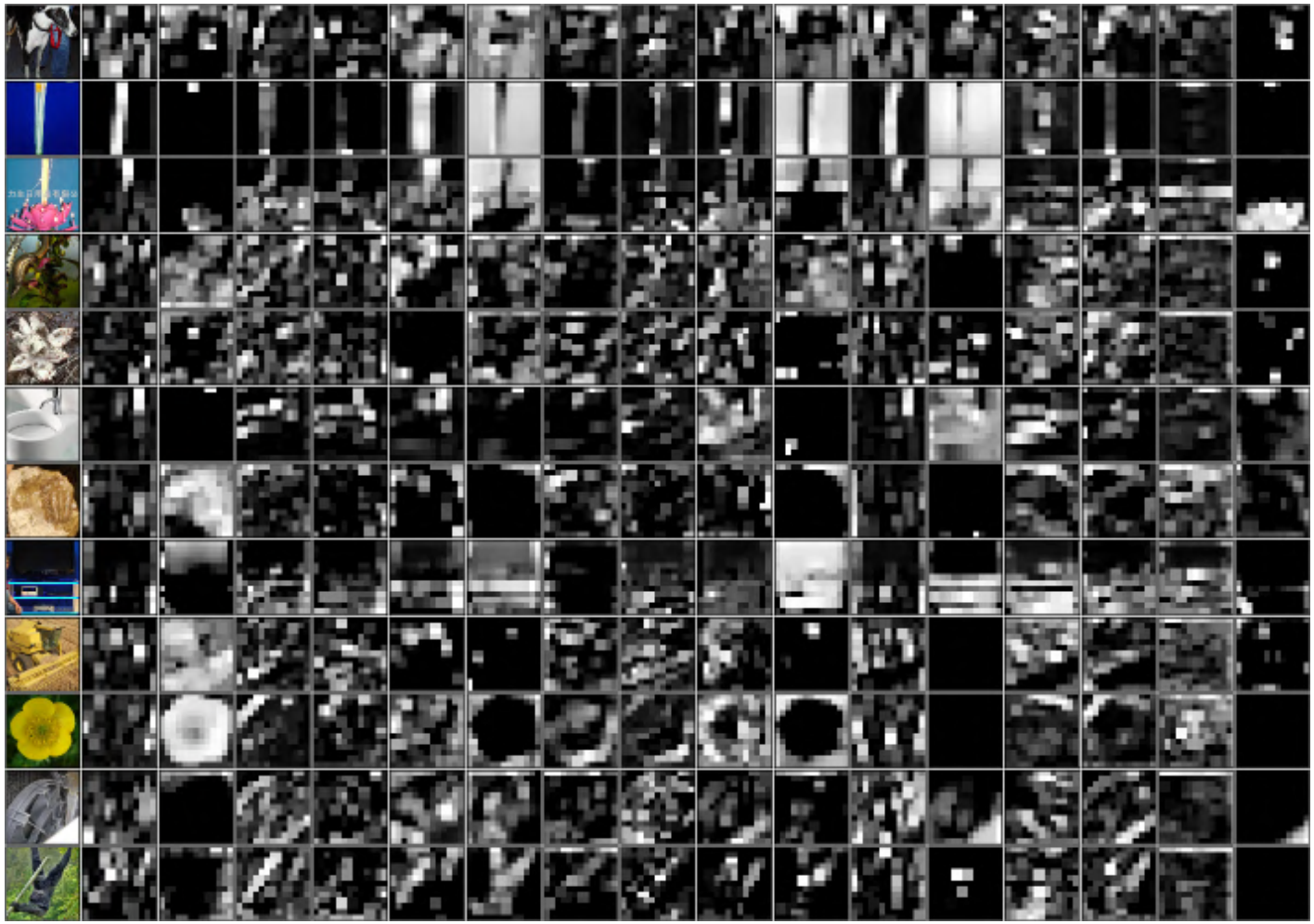


Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# AlexNet: convolution and pooling layers

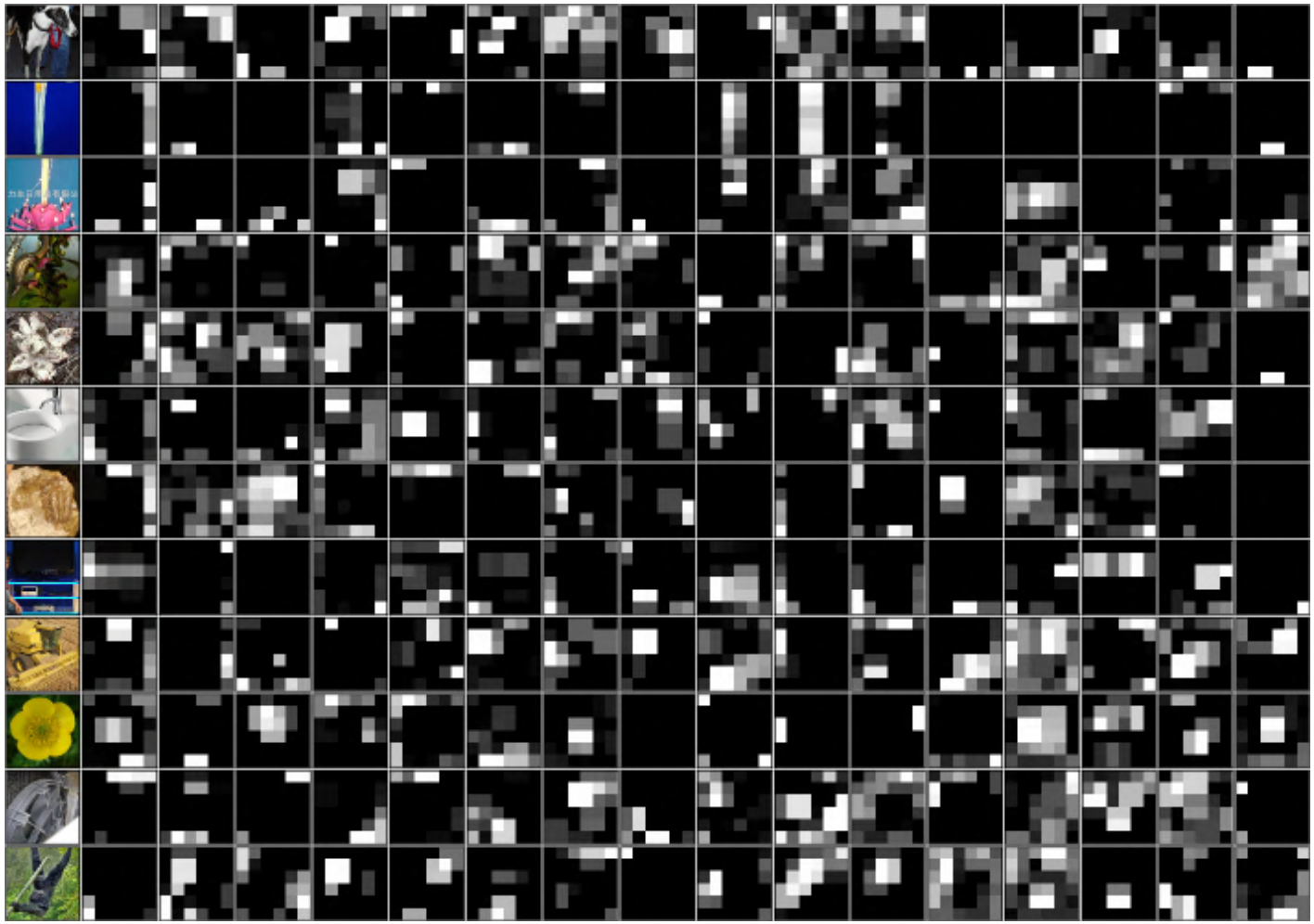


Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# AlexNet: convolution and pooling layers



Activation maps (16 random channels) of convolution layers (1–5).

Activation maps (16 random channels) of max pooling layers (1–3).

Figures taken from [Krizhevsky et al., 2012].

# VGGNet [Simonyan and Zisserman, 2015]

- L'architecture VGGNet a représenté une autre avancée.
- Le réseau se compose de 16 à 19 couches convolutives et de couches Max Pooling.
- Ce réseau a été également entraîné pour un problème de classification d'images sur le dataset ImageNet.
- Augmenter la profondeur du réseau et diminuer la taille des filtres améliore la performance.
- VGGNet s'entraîne par backpropagation!

# VGGNet [Simonyan and Zisserman, 2015]

- L'architecture VGGNet a représenté une autre avancée.
- Le réseau se compose de $16$ à $19$ couches convolutives et de couches Max Pooling.
- Ce réseau a été également entraîné pour un problème de classification d'images sur le dataset ImageNet.
- Augmenter la profondeur du réseau et diminuer la taille des filtres améliore la performance.
- VGGNet s'entraîne par backpropagation!

# VGGNet [Simonyan and Zisserman, 2015]

- L'architecture VGGNet a représenté une autre avancée.
- Le réseau se compose de $16$ à $19$ couches convolutives et de couches Max Pooling.
- Ce réseau a été également entraîné pour un problème de classification d'images sur le dataset ImageNet.
- Augmenter la profondeur du réseau et diminuer la taille des filtres améliore la performance.
- VGGNet s'entraîne par backpropagation!

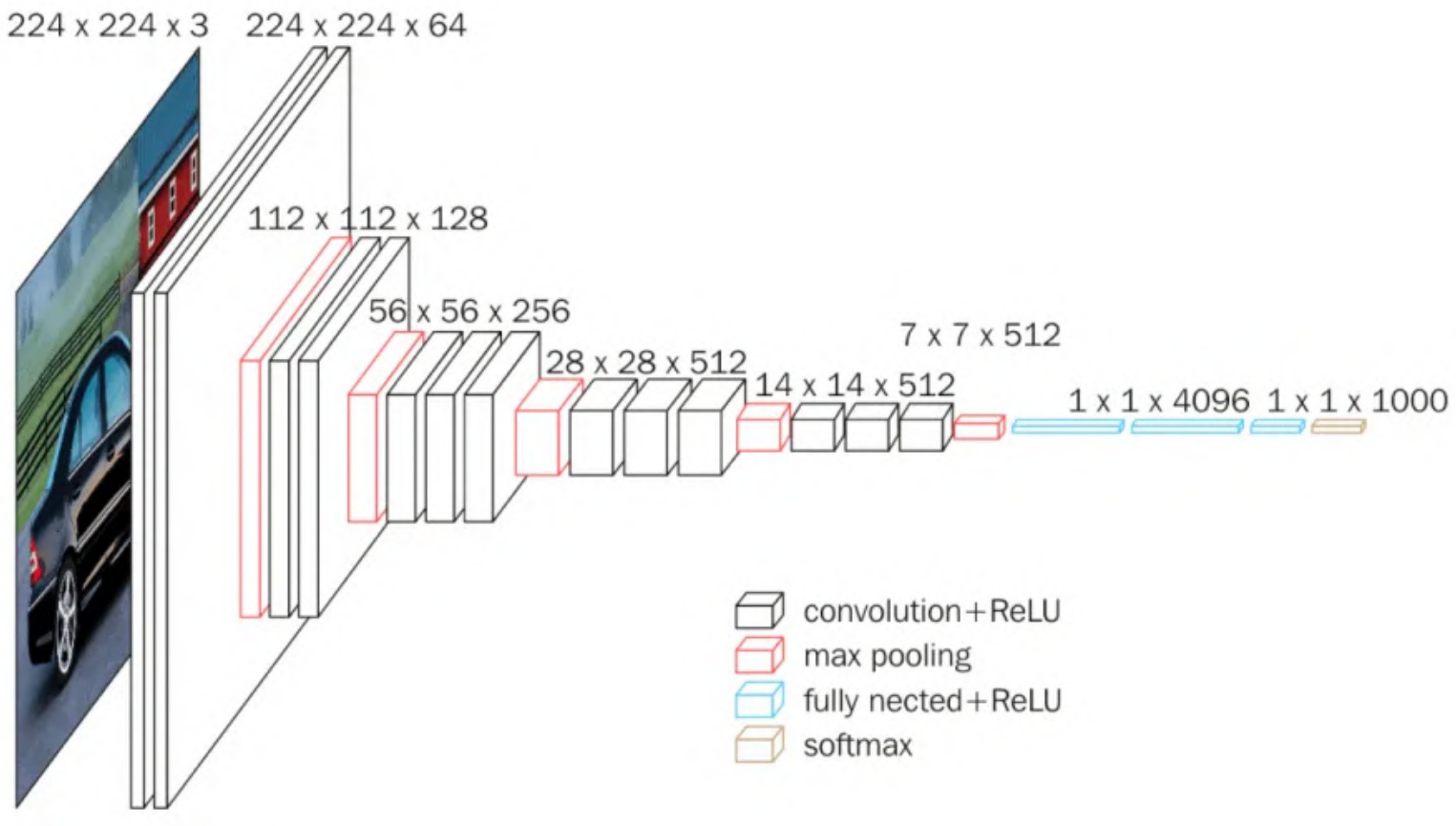
# VGGNet [Simonyan and Zisserman, 2015]

- L'architecture VGGNet a représenté une autre avancée.
- Le réseau se compose de $16$ à $19$ couches convolutives et de couches Max Pooling.
- Ce réseau a été également entraîné pour un problème de classification d'images sur le dataset ImageNet.
- Augmenter la profondeur du réseau et diminuer la taille des filtres améliore la performance.
- VGGNet s'entraîne par backpropagation!

# VGGNet [Simonyan and Zisserman, 2015]

- L'architecture VGGNet a représenté une autre avancée.
- Le réseau se compose de $16$ à $19$ couches convolutives et de couches Max Pooling.
- Ce réseau a été également entraîné pour un problème de classification d'images sur le dataset ImageNet.
- Augmenter la profondeur du réseau et diminuer la taille des filtres améliore la performance.
- VGGNet s'entraîne par backpropagation!

# VGGNet




Figures taken from

https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

# VGGNet

| VGG16 - Structural Details | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # | Input Image | | | output | | | Layer | Stride | Kernel | | in | out | Param |
| 1 | 224 | 224 | 3 | 224 | 224 | 64 | conv3-64 | 1 | 3 | 3 | 3 | 64 | 1792 |
| 2 | 224 | 224 | 64 | 224 | 224 | 64 | conv3064 | 1 | 3 | 3 | 64 | 64 | 36928 |
| | 224 | 224 | 64 | 112 | 112 | 64 | maxpool | 2 | 2 | 2 | 64 | 64 | 0 |
| 3 | 112 | 112 | 64 | 112 | 112 | 128 | conv3-128 | 1 | 3 | 3 | 64 | 128 | 73856 |
| 4 | 112 | 112 | 128 | 112 | 112 | 128 | conv3-128 | 1 | 3 | 3 | 128 | 128 | 147584 |
| | 112 | 112 | 128 | 56 | 56 | 128 | maxpool | 2 | 2 | 2 | 128 | 128 | 65664 |
| 5 | 56 | 56 | 128 | 56 | 56 | 256 | conv3-256 | 1 | 3 | 3 | 128 | 256 | 295168 |
| 6 | 56 | 56 | 256 | 56 | 56 | 256 | conv3-256 | 1 | 3 | 3 | 256 | 256 | 590080 |
| 7 | 56 | 56 | 256 | 56 | 56 | 256 | conv3-256 | 1 | 3 | 3 | 256 | 256 | 590080 |
| | 56 | 56 | 256 | 28 | 28 | 256 | maxpool | 2 | 2 | 2 | 256 | 256 | 0 |
| 8 | 28 | 28 | 256 | 28 | 28 | 512 | conv3-512 | 1 | 3 | 3 | 256 | 512 | 1180160 |
| 9 | 28 | 28 | 512 | 28 | 28 | 512 | conv3-512 | 1 | 3 | 3 | 512 | 512 | 2359808 |
| 10 | 28 | 28 | 512 | 28 | 28 | 512 | conv3-512 | 1 | 3 | 3 | 512 | 512 | 2359808 |
| | 28 | 28 | 512 | 14 | 14 | 512 | maxpool | 2 | 2 | 2 | 512 | 512 | 0 |
| 11 | 14 | 14 | 512 | 14 | 14 | 512 | conv3-512 | 1 | 3 | 3 | 512 | 512 | 2359808 |
| 12 | 14 | 14 | 512 | 14 | 14 | 512 | conv3-512 | 1 | 3 | 3 | 512 | 512 | 2359808 |
| 13 | 14 | 14 | 512 | 14 | 14 | 512 | conv3-512 | 1 | 3 | 3 | 512 | 512 | 2359808 |
| | 14 | 14 | 512 | 7 | 7 | 512 | maxpool | 2 | 2 | 2 | 512 | 512 | 0 |
| 14 | 1 | 1 | 25088 | 1 | 1 | 4096 | fc | | 1 | 1 | 25088 | 4096 | 102764544 |
| 15 | 1 | 1 | 4096 | 1 | 1 | 4096 | fc | | 1 | 1 | 4096 | 4096 | 16781312 |
| 16 | 1 | 1 | 4096 | 1 | 1 | 1000 | fc | | 1 | 1 | 4096 | 1000 | 4097000 |
| Total | | | | | | | | | | | | | 138,423,208 |

Figures taken from

https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

# Bibliographie I



Fleuret, F. (2022).
Deep Learning Course.


Fukushima, K. (1980).
Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.
*Biological Cybernetics*, 36:193–202.


Fukushima, K. (1988).
Neocognitron: A hierarchical neural network capable of visual pattern recognition.

*Neural Networks*, 1(2):119–130.


He, K., Zhang, X., Ren, S., and Sun, J. (2016).
Deep residual learning for image recognition.
In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society.

# Bibliographie II


Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012).
Imagenet classification with deep convolutional neural networks.
In Bartlett, P. L., Pereira, F. C. N., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114.


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


Ronneberger, O., Fischer, P., and Brox, T. (2015).
U-net: Convolutional networks for biomedical image segmentation.
In Navab, N., Hornegger, J., III, W. M. W., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer.

# Bibliographie III

Simonyan, K. and Zisserman, A. (2015).
Very deep convolutional networks for large-scale image recognition.
In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.