

Réseaux de convolution

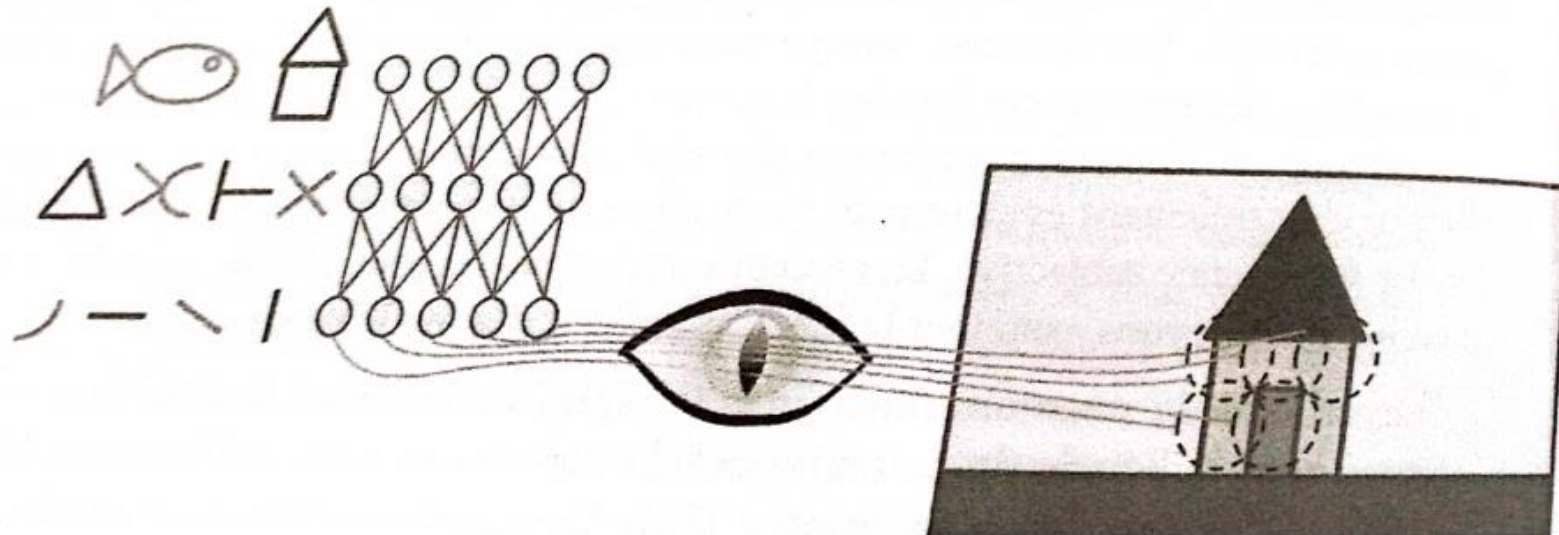
Convolution Neural Network (CNN)

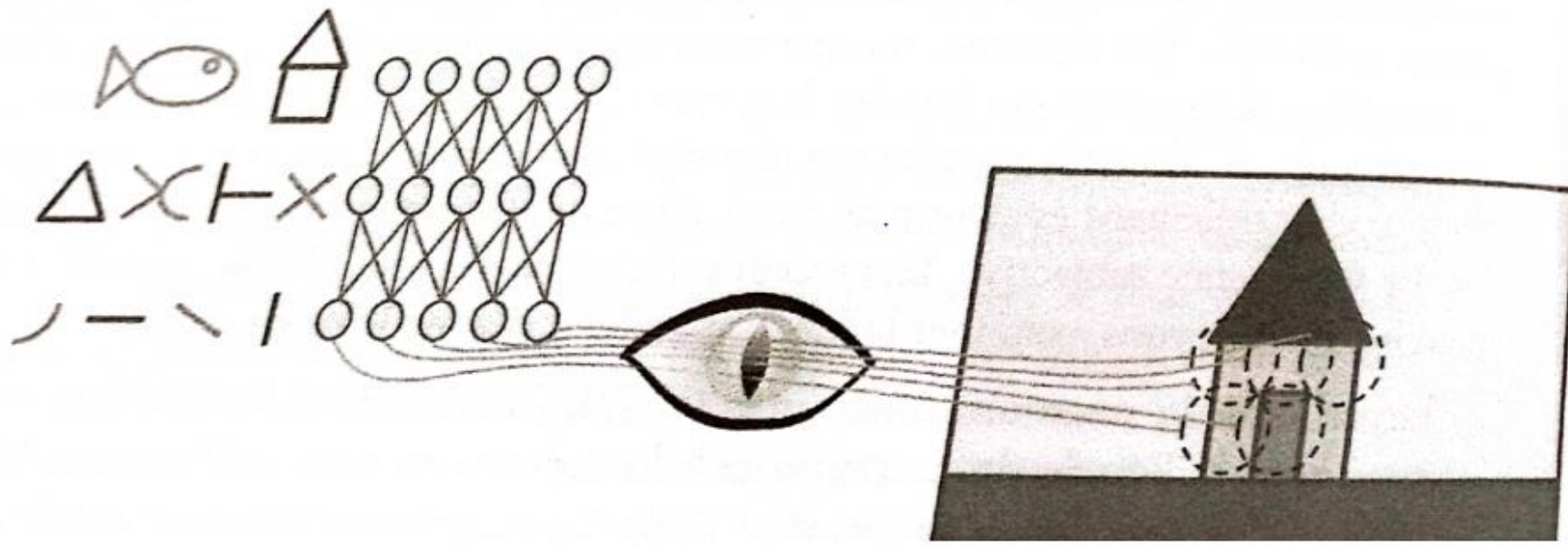
Sommaire

1. Introduction : un peu d'histoire ...
2. Couche de convolution
3. Filtre (kernel ou matrice de convolution)
4. Stride
5. Padding
6. Pooling
7. Classification-Fully Connected Layer
8. Des liens ...

1. Introduction : un peu d'histoire

En 1958⁷⁶ et en 1959⁷⁷, David H. Hubel et Torsten Wiesel ont mené une série d'expériences sur des chats (et, quelques années plus tard, sur des singes⁷⁸), apportant des informations essentielles sur la structure du cortex visuel (en 1981, ils ont reçu le prix Nobel de physiologie ou médecine pour leurs travaux). Ils ont notamment montré que de nombreux neurones du cortex visuel ont un petit *champ récepteur local* et qu'ils réagissent donc uniquement à un stimulus visuel qui se trouve dans une région limitée du champ visuel (voir la figure sur laquelle les champs récepteurs locaux de cinq neurones sont représentés par les cercles en pointillés).

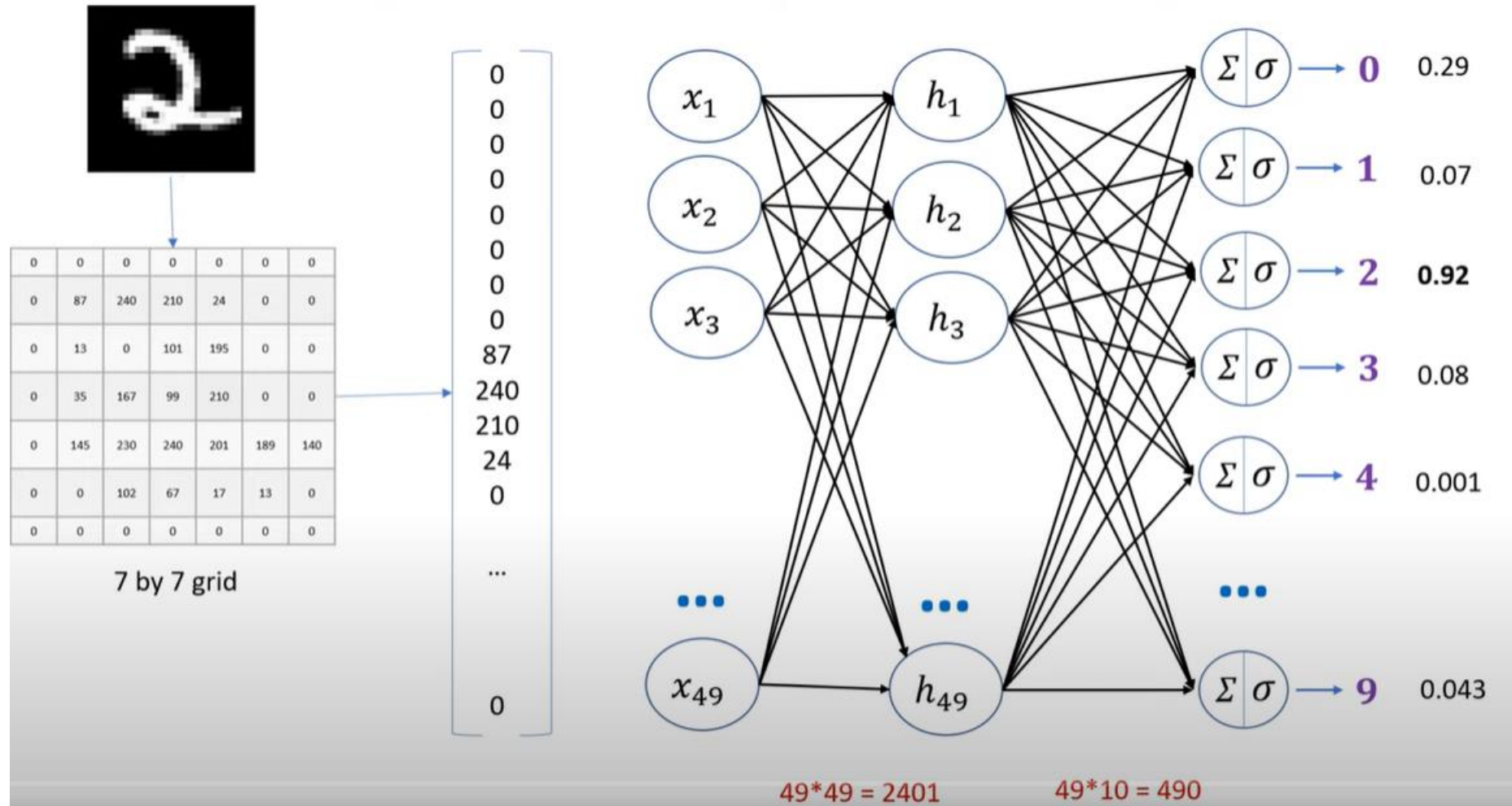




Les champs récepteurs des différents neurones peuvent se chevaucher et ils couvrent ensemble l'intégralité du champ visuel. Ils ont également montré que certains neurones réagissent uniquement aux images de lignes horizontales, tandis que d'autres réagissent uniquement aux lignes ayant d'autres orientations (deux neurones peuvent avoir le même champ récepteur mais réagir à des orientations de lignes différentes). Ils ont remarqué que certains neurones ont des champs récepteurs plus larges et qu'ils réagissent à des motifs plus complexes, correspondant à des combinaisons de motifs de plus bas niveau. Ces observations ont conduit à l'idée que les neurones de plus haut niveau se fondent sur la sortie des neurones voisins de plus bas niveau (sur la figure , chaque neurone est connecté uniquement à quelques neurones de la couche précédente). Cette architecture puissante est capable de détecter toutes sortes de motifs complexes dans n'importe quelle zone du champ visuel.

Limites de réseaux de neurones simples

- Dans le cas d'image à faible résolution, nous avons vu qu'il est parfaitement possible d'utiliser des ANN simple : cas des images MNIST...



Limites de réseaux de neurones simples



Image size = 1920 x 1080 X 3

First layer neurons = 1920 x 1080 X 3 ~ 6 million

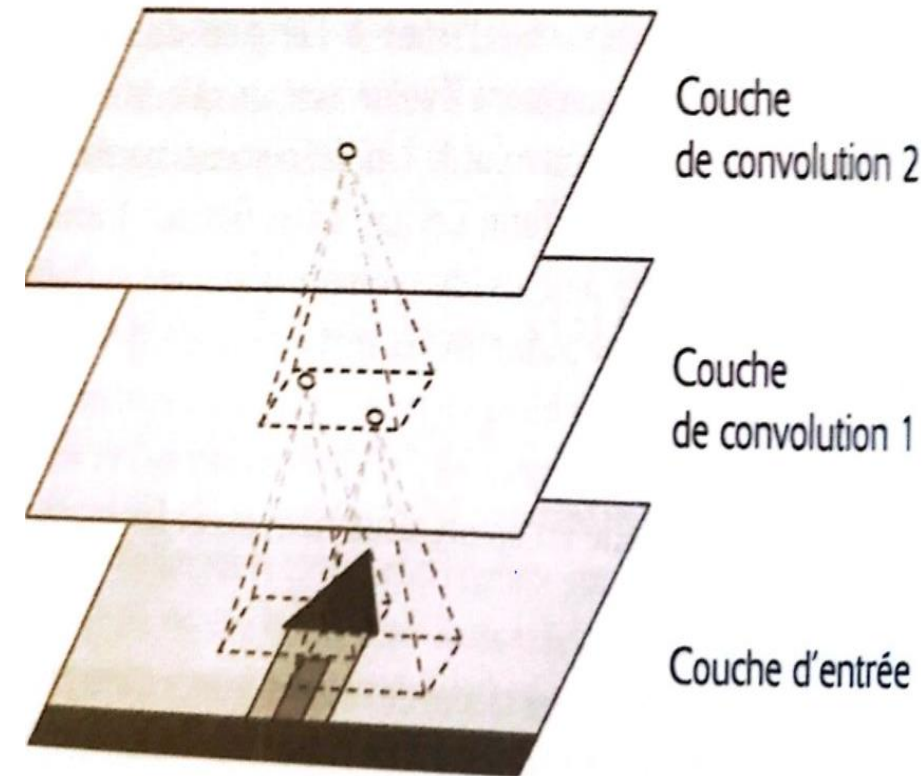
Hidden layer neurons = Let's say you keep it ~ 4 million

Weights between input and hidden layer = 6 mil * 4 mil
= 24 million

- Trop de calculs !
- En plus, lors qu'un réseau classique a appris à reconnaître un motif en un endroit, il ne peut le reconnaître qu'en cet endroit précis.
- Les CNN résolvent ce problème en utilisant des couches partiellement connectées.

2. Couches de convolution

- La couche de convolution est le bloc de construction le plus important d'un CNN. Dans la première couche de convolution, les neurones ne sont pas connectés à chaque pixel de l'image d'entrée mais uniquement aux pixels dans leurs champs récepteur.
- A leur tour, les neurones de la deuxième couche de convolution sont chacun connectés uniquement aux neurones situés à l'intérieur d'un petit rectangle de la première couche.
- Cette architecture permet au réseau de se focaliser sur des caractéristiques de bas niveau dans la première couche cachée, puis de les assembler en caractéristiques de plus haut niveau dans la couche suivante.



Comment reconnait-on une image ?



Koala's **eye**? = Y



Koala's **nose**? = Y



Koala's **ears**? = Y



Koala's **hands**? = Y



Koala's **legs**? = Y



Koala's **head**? = Y

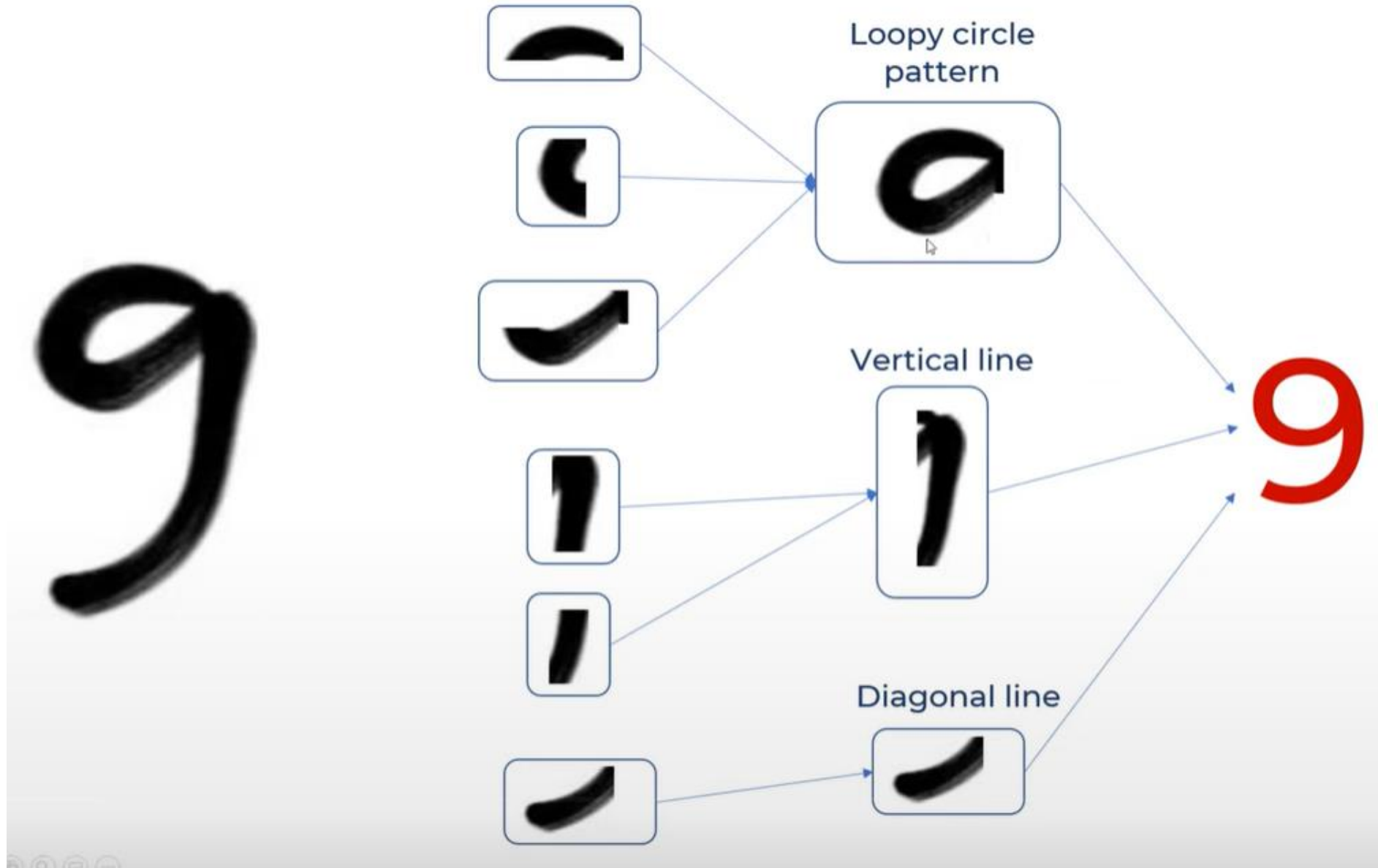


Koala's **body**? = Y



Is it **Koala**? = Y

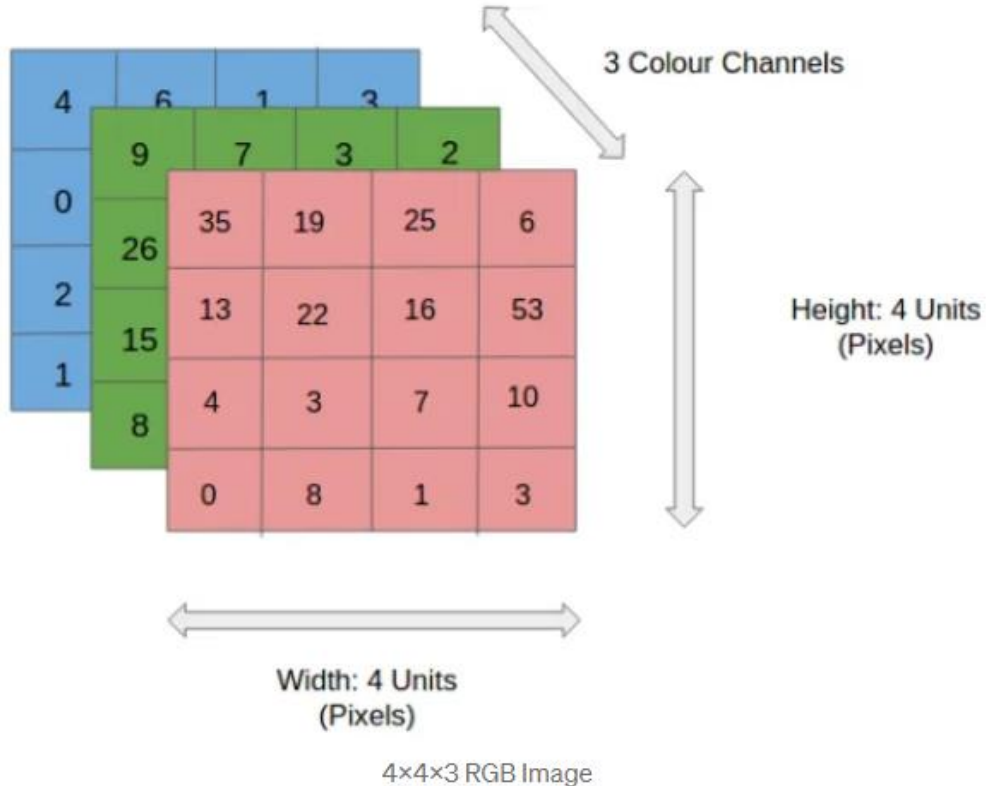
Comment reconnait-on une image ?



La convolution

- An RGB image that has been separated by its three color planes — Red, Green, and Blue.

Input Image



- In the above demonstration, the green section resembles our **5x5x1 input image, I**.
- The element involved in the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in color yellow.

Convolution Layer — The Kernel

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved Feature

La convolution

- In the above demonstration, the green section resembles our **5x5x1 input image**.
- The element involved in the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in color yellow.

- We have selected K as a 3x3x1 matrix:

Convolution Layer — The Kernel

Kernel/Filter, K =		
1	0	1
0	1	0
1	0	1

- Convolution operation means, for a given input we re-estimate it as the weighted average of all inputs around it.

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

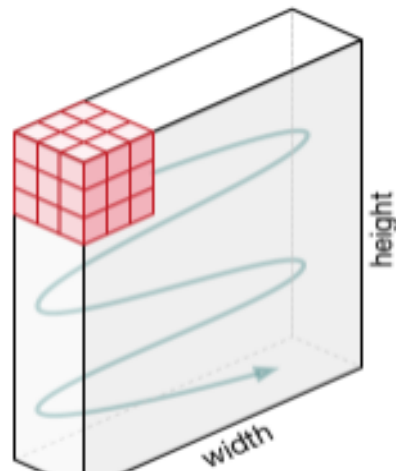
Convolved Feature

3. Filtre (kernel ou matrice de convolution)

```
Kernel/Filter, K =
```

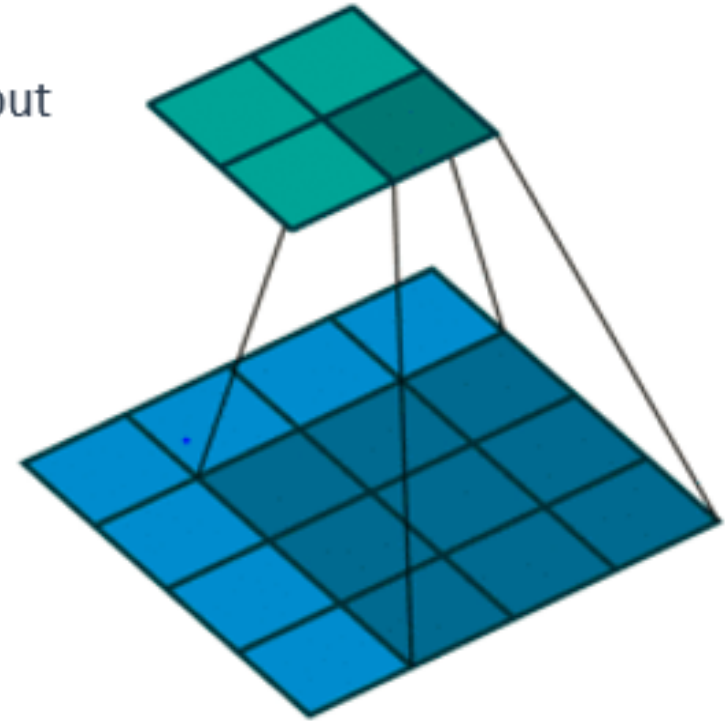
```
1  0  1  
0  1  0  
1  0  1
```

- The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.



Visualisation de la convolution

- Having a 4x4 image and a 3x3 filter, hence we are getting output after convolution 2x2



- If we have $N \times N$ image size and $F \times F$ filter size then after convolution result will be $(N-F+1) \times (N-F+1)$

Des filtres, pourquoi ?



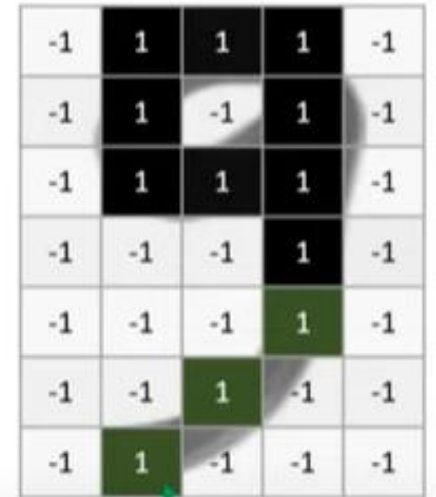
Loopy pattern
filter



Vertical line
filter



Diagonal line
filter



Comment fonctionnent les filtres?

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

*

1	1	1
1	-1	1
1	1	1

-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33

Feature Map

Application de la ReLu

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

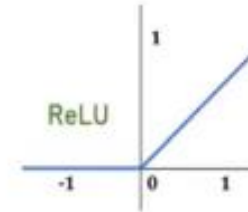
*

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1




-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Convolution + ReLu

Loopy pattern detector

 *

1	1	1
1	-1	1
1	1	1

=

	1	

Loopy pattern detector

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{matrix} = \begin{matrix} & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{matrix}$$

Loopy pattern detector

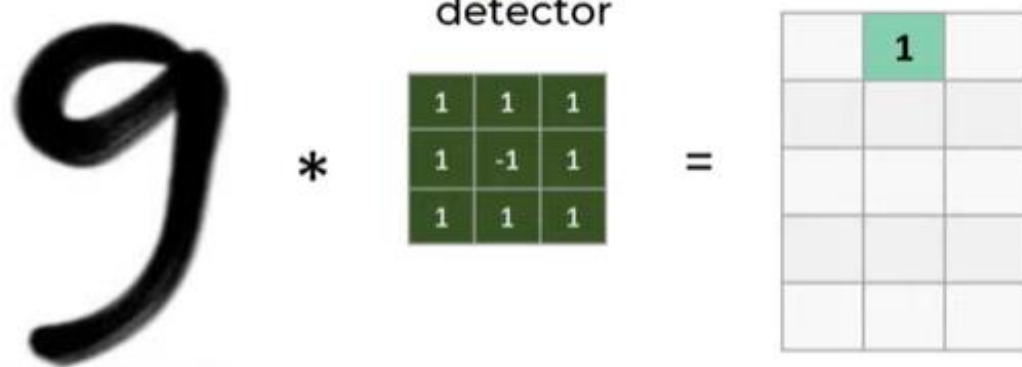
$8 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} & 1 & \\ & & \\ & & \\ & & \\ & & \\ & 1 & \end{bmatrix}$

[illegible]

Filters are nothing but feature detectors...

Motifs

Loopy pattern detector

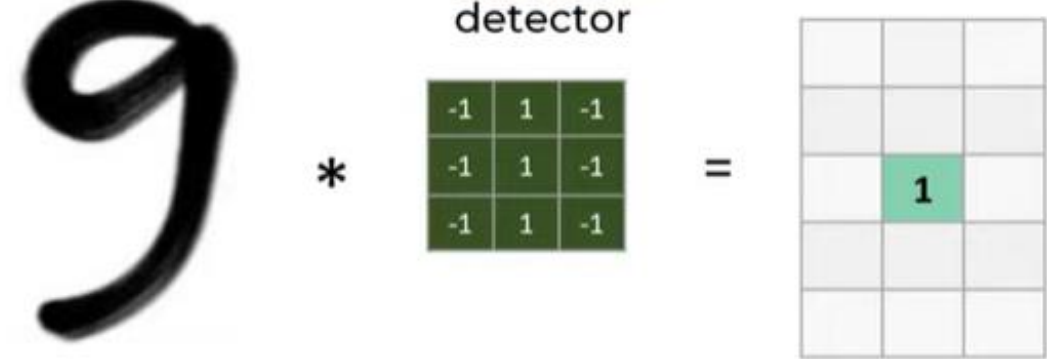


The diagram shows a handwritten digit '9' on the left. To its right is a multiplication symbol followed by a 3x3 green kernel matrix:

1	1	1
1	-1	1
1	1	1

Following this is an equals sign and a 5x5 output grid. The cell at row 2, column 2 (0-indexed) is highlighted in green and contains the value 1.

Vertical line detector

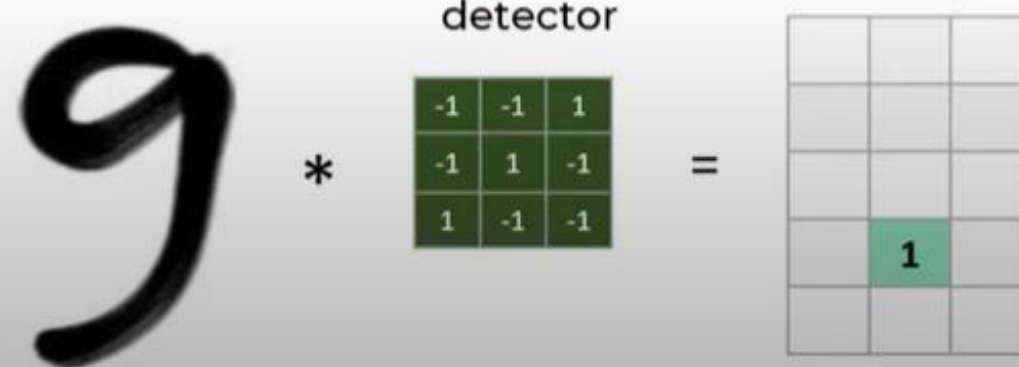


The diagram shows a handwritten digit '9' on the left. To its right is a multiplication symbol followed by a 3x3 green kernel matrix:

-1	1	-1
-1	1	-1
-1	1	-1

Following this is an equals sign and a 5x5 output grid. The cell at row 2, column 2 (0-indexed) is highlighted in green and contains the value 1.

Diagonal line detector



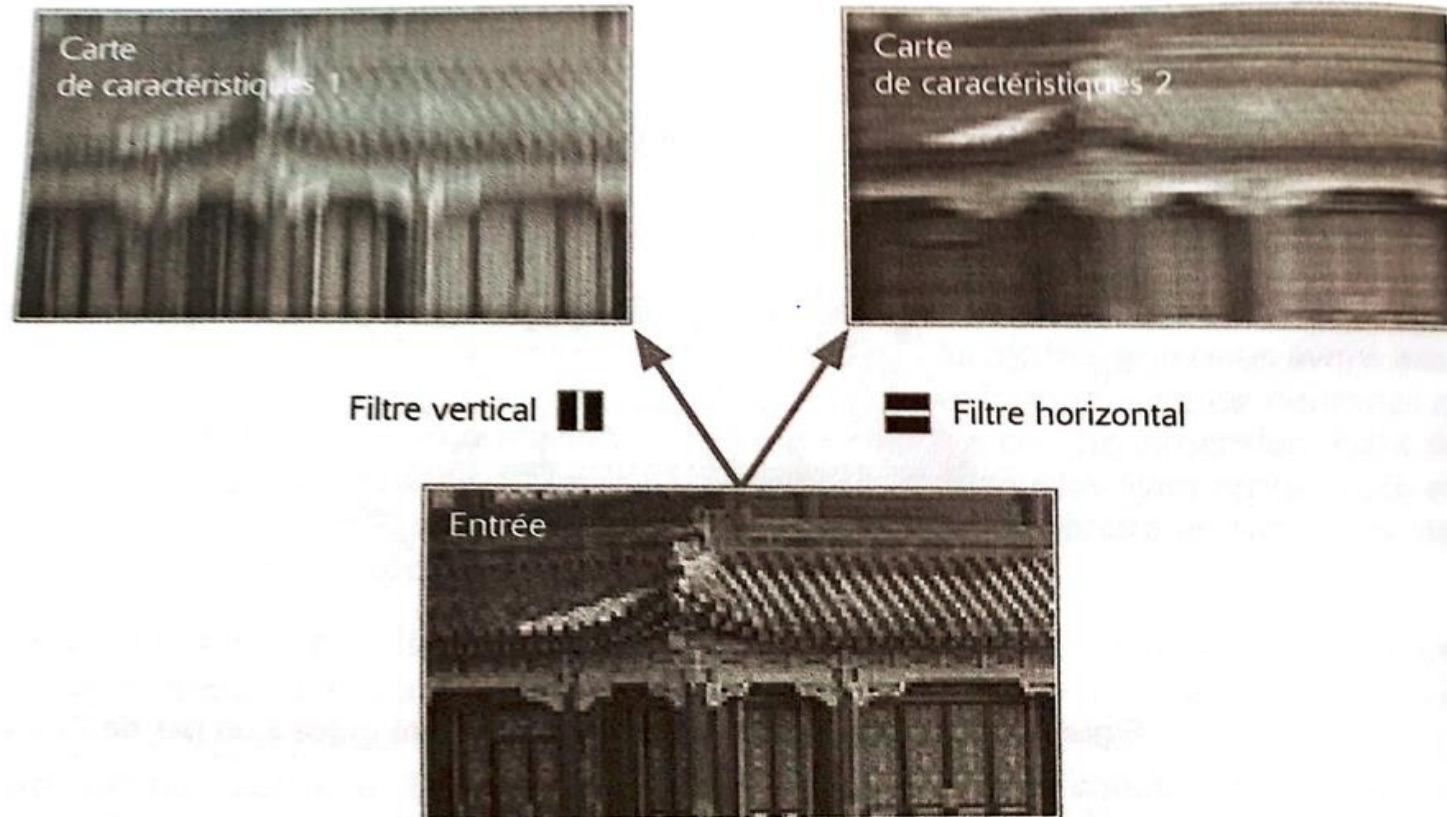
The diagram shows a handwritten digit '9' on the left. To its right is a multiplication symbol followed by a 3x3 green kernel matrix:

-1	-1	1
-1	1	-1
1	-1	-1

Following this is an equals sign and a 5x5 output grid. The cell at row 2, column 2 (0-indexed) is highlighted in green and contains the value 1.

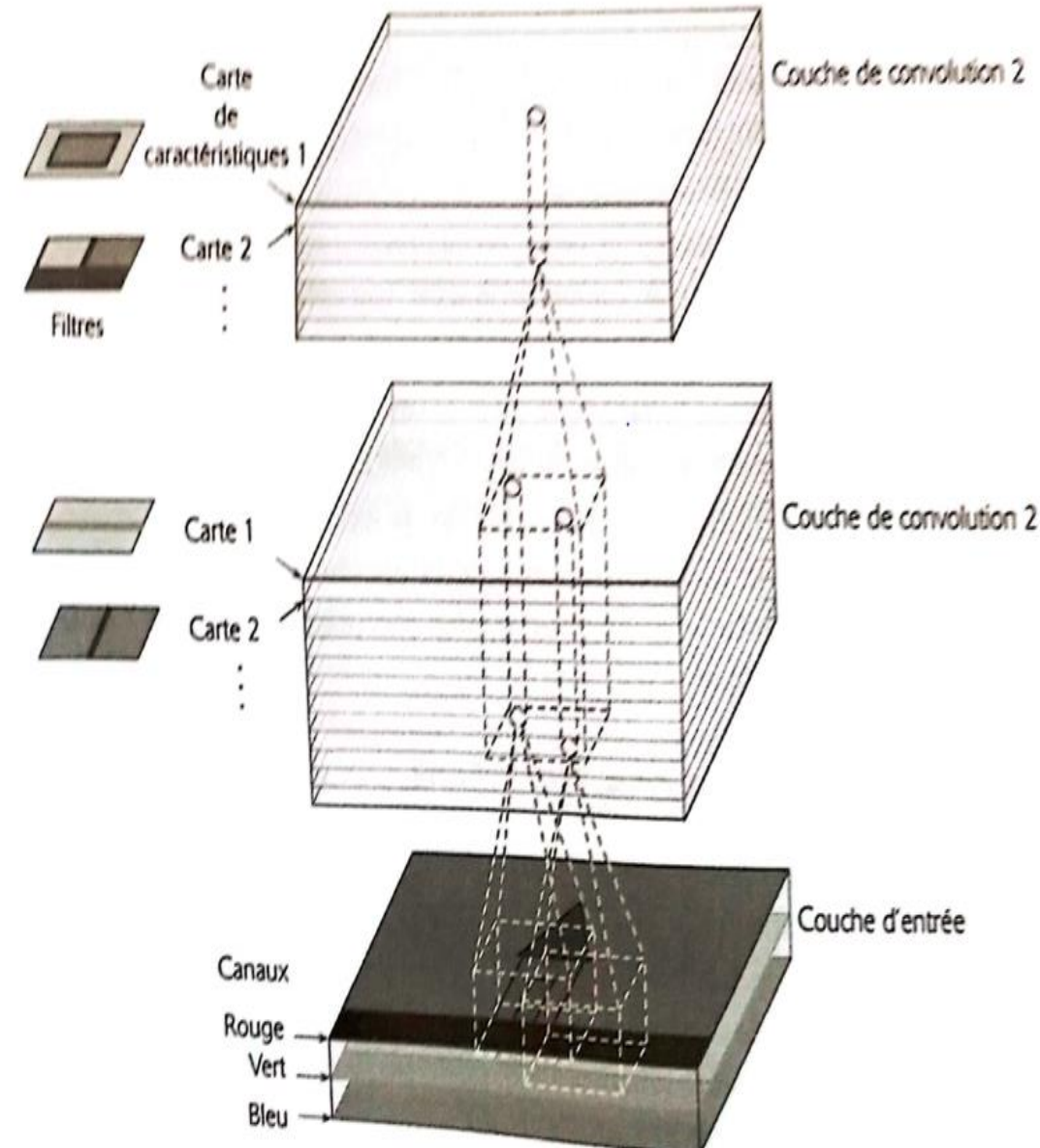
Exemples de filtres

- Les neurones qui utilisent le premier filtre (carré noir avec ligne blanche verticale) ignorant tout ce qui trouve dans leur champ récepteur , à l'exception de la ligne verticale centrale
- Pour le second filtre, les neurones utilisant ses poids, ignoreront tout hormis la ligne horizontale.
- Evidemment on ne va pas définir les filtres manuellement ! **Au cours de l'entraînement, la couche de convolution apprend les filtres qui seront les plus utiles à sa tâche.**



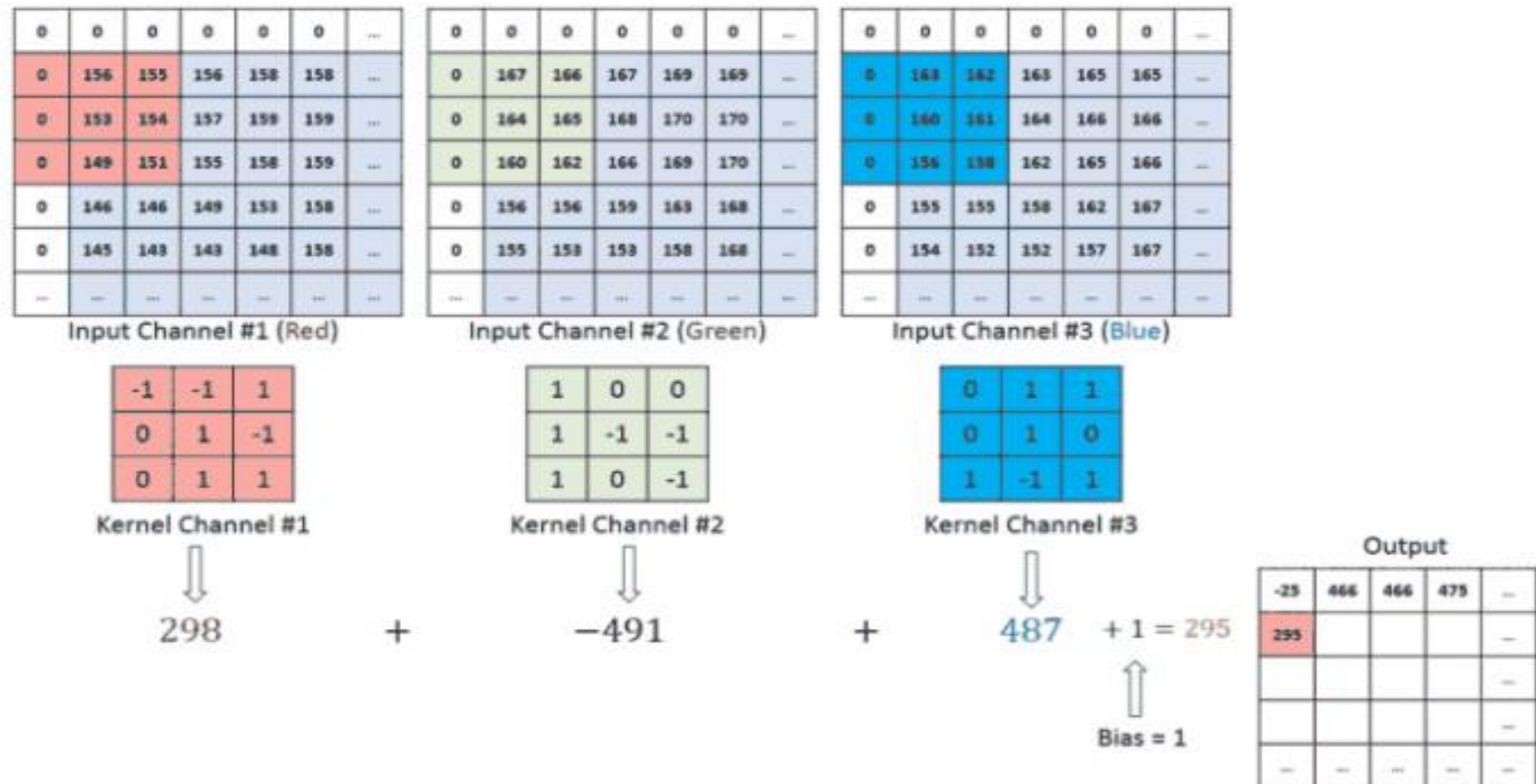
Empiler plusieurs cartes de caractéristiques (Feature maps)

- Jusqu'à présent, nous avons représenté chaque **couche de convolution** comme une fine couche à deux dimensions, mais en réalité elle est constituée de **plusieurs cartes de caractéristiques** de taille égale.
- Il est donc plus correct de la représenter en 3 dimensions.
- A l'intérieur d'une carte de caractéristiques, tous les neurones partagent les mêmes paramètres (poids et terme constant).
- Les différentes cartes de caractéristiques peuvent avoir des paramètres distincts.



Cas des 3 canaux RGB

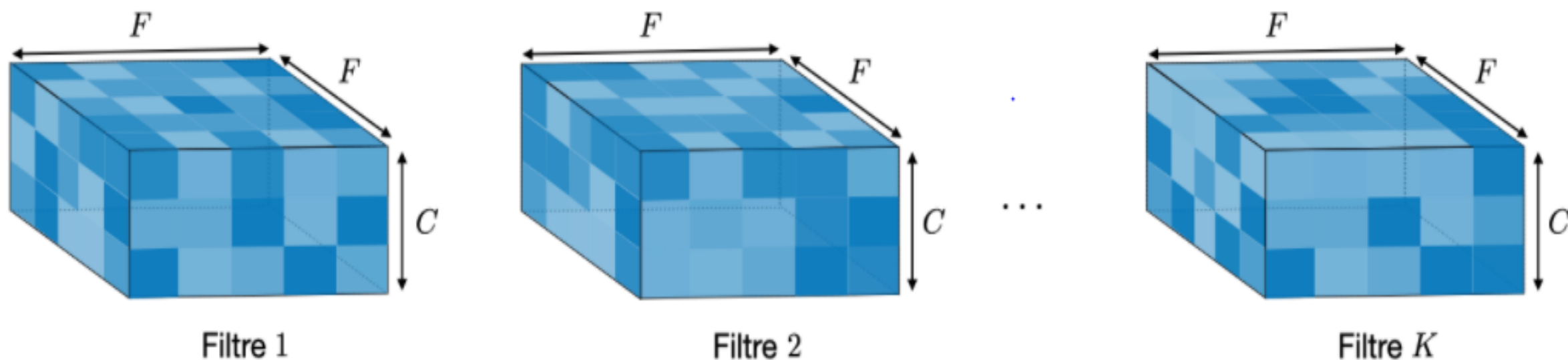
- In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image.
- Matrix Multiplication is performed between K and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.



Convolution operation on a $M \times N \times 3$ image matrix with a $3 \times 3 \times 3$ Kernel

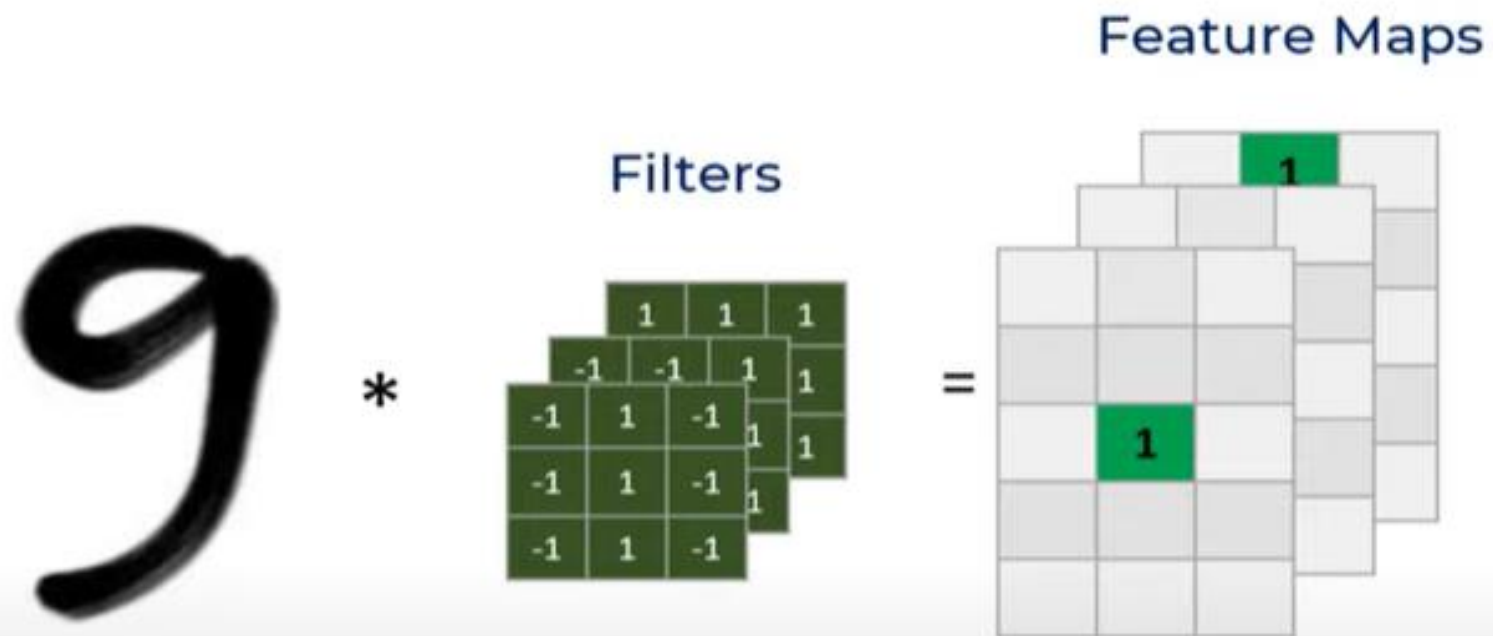
Application de K filtres...

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



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

Feature Maps

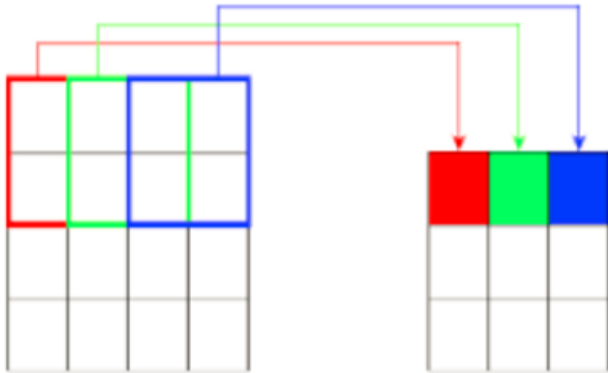


Si on résumait cette première partie ...

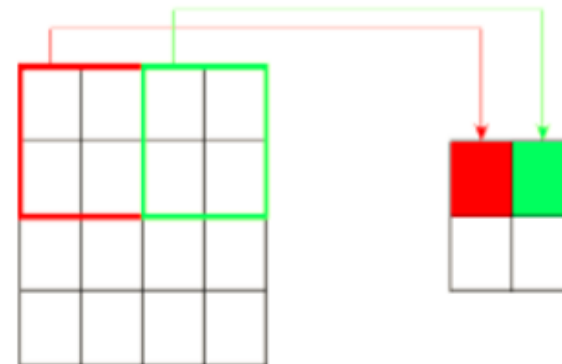
- Une couche de convolution applique simultanément plusieurs filtres à ses entrées, ce qui lui permet de détecter plusieurs caractéristiques n'importe où dans ses entrées.
- Puisque tous les neurones d'une carte de caractéristiques partagent les mêmes paramètres (i.e. le même filtre), le nombre de paramètres du modèle s'en trouve considérablement réduit.
- Plus, important encore, cela signifie également que dès que le CNN a appris un motif en un endroit, il peut le reconnaître partout ailleurs.
- A l'opposé, lorsqu'un réseau classique a appris à reconnaître un motif en un endroit, il ne peut le reconnaître qu'en cet endroit précis.

4. Stride

- The filter moves to the right with a certain Stride Value till it parses the complete width.
- Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.
- Stride is the number of pixels shifts over the input matrix : when the stride is s the we move the filters to s pixels at a time.



Stride = 1



Stride = 2

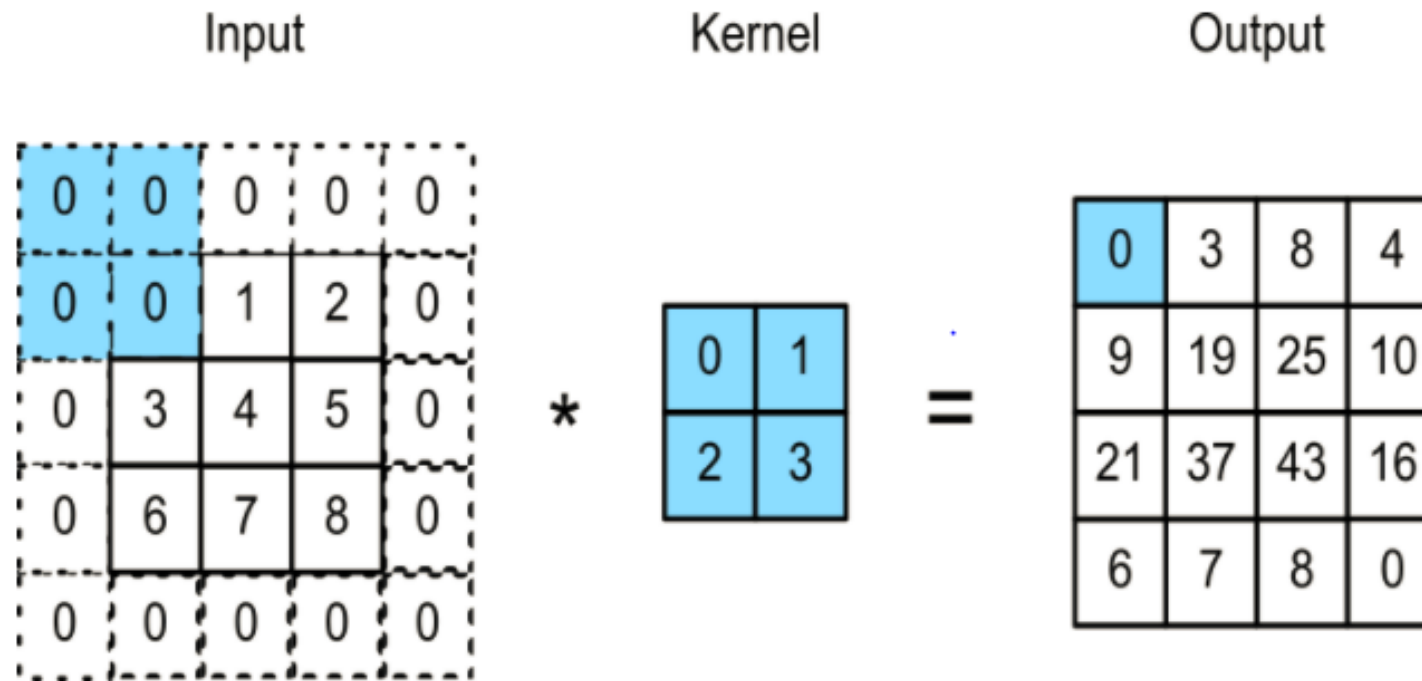
5. Padding

- Without image padding, the result of convolution will be smaller than the original image size. Thus, the image shrinks every time a convolution operation is performed.
- Also, pixels on the corner are much less used than those in the middle of the image.
- To avoid these we add layers of zeros to our input image, we'll call this process padding.

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

Two types of padding : Valid and Same

- **Valid padding** : It implies no padding at all.
- **Same padding** : In this case, we add p layers such that the output image has the same dimensions as the input image.



The result size of a convolution will be $((N - F + 2P) / S) + 1$

Keras :convolution layer

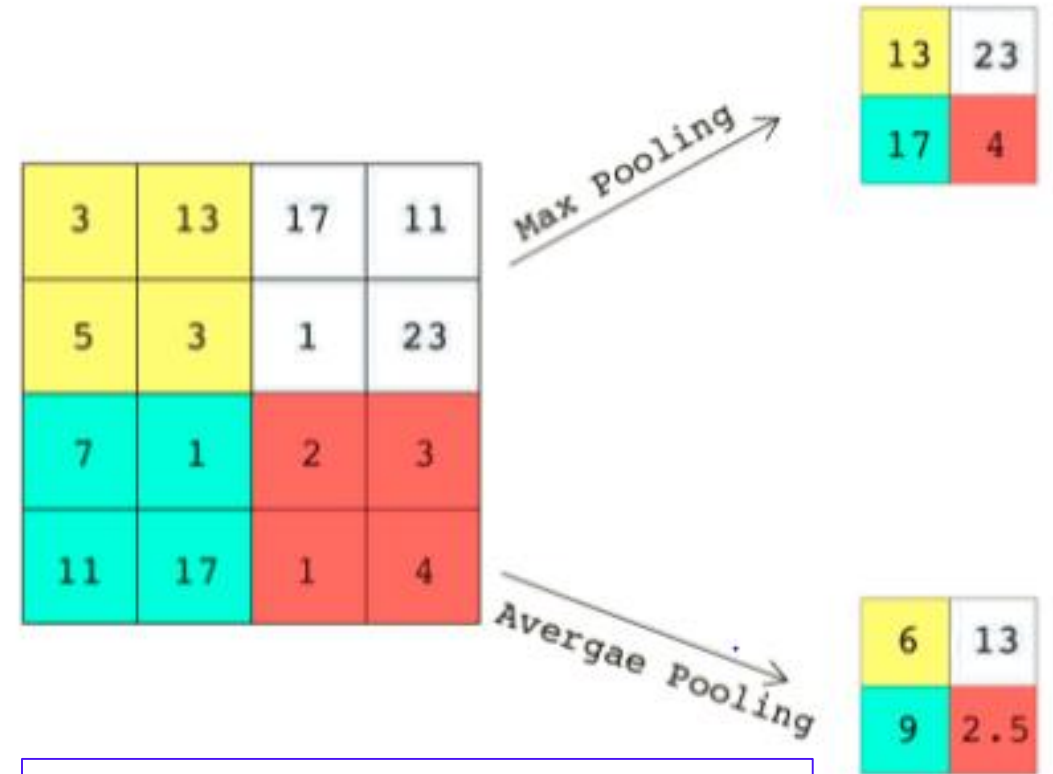
```
tf.keras.layers.Conv2D(  
    filters,  
    kernel_size,  
    strides=(1, 1),  
    padding="valid",  
    data_format=None,  
    dilation_rate=(1, 1),  
    groups=1,  
    activation=None,  
    use_bias=True,  
    kernel_initializer="glorot_uniform",  
    bias_initializer="zeros",  
    kernel_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    bias_constraint=None,  
    **kwargs  
)
```

```
from tensorflow.keras.layers import Conv2D
```

```
Conv2D(32, kernel_size=(5, 5), activation='sigmoid', input_shape=(28, 28, 1), padding='same')
```

6. Pooling

- Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature.
- In the example below we will reduce each spatial dimension in half :
- Furthermore, it is useful for extracting dominant features which are rotational and positional invariant, thus maintaining the process of effectively training of the model.
- This is to decrease the computational power required to process the data through dimensionality reduction.



A 2x2 window (filter) with a stride = 2

Two main types of Pooling : Max and Average

- **Max (resp. Min) Pooling** returns the **maximum (resp. minimum) value** from the portion of the image covered by the Kernel.
- On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.
- Max Pooling also performs as a **Noise Suppressant**. It discards the noisy activations altogether and also performs de-noising along with dimensionality reduction.
- On the other hand, Average Pooling simply performs dimensionality reduction as a noise suppressing mechanism.
- Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. For example: in MNIST dataset, the digits are represented in white color and the background is black. So, max pooling is used. Similarly, min pooling is used in the other way round.

-1	1	1	1	-1
-1	1	-1	1	-1
-1	1	1	1	-1
-1	-1	-1	1	-1
-1	-1	-1	1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1

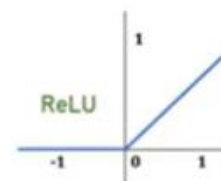
*

Loopy pattern
filter

1	1	1
1	-1	1
1	1	1



-0.11	1	-0.11
-0.55	0.11	-0.33
-0.33	0.33	-0.33
-0.22	-0.11	-0.22
-0.33	-0.33	-0.33



0	1	0
0	0.11	0
0	0.33	0
0	0	0
0	0	0

Max
pooling



1	1
0.33	0.33
0.33	0.33
0	0

Shifted 9 at
different position

1	1	1	-1	-1
1	-1	1	-1	-1
1	1	1	-1	-1
-1	-1	1	-1	-1
-1	-1	1	-1	-1
-1	1	-1	-1	-1
1	-1	-1	-1	-1

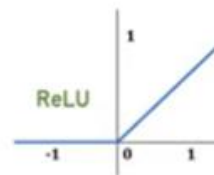
Loopy pattern
filter

*

1	1	1
1	-1	1
1	1	1

→

1	-0.11	-0.11
0.11	-0.33	0.33
0.33	-0.33	-0.33
-0.11	-0.55	-0.33
-0.55	-0.33	-0.55



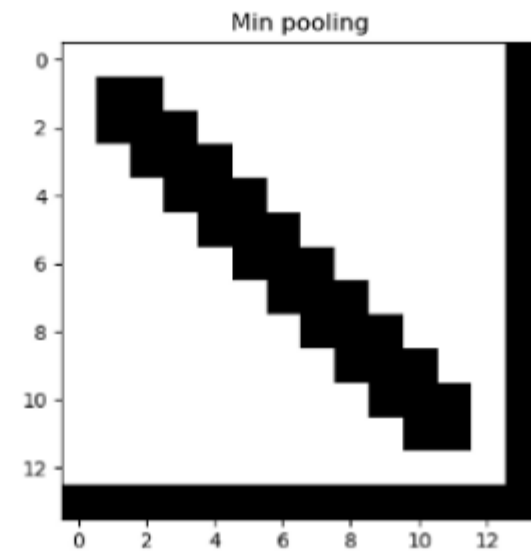
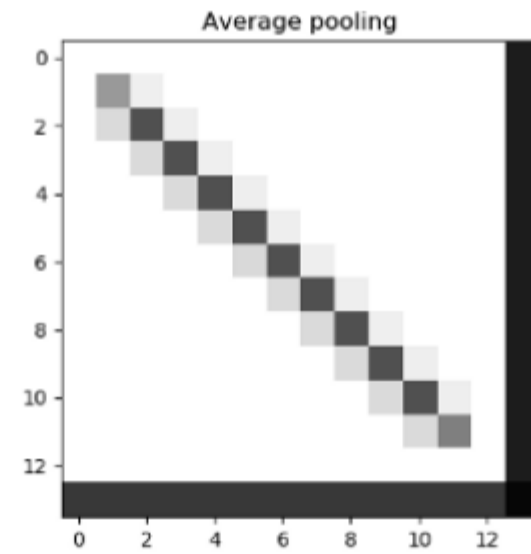
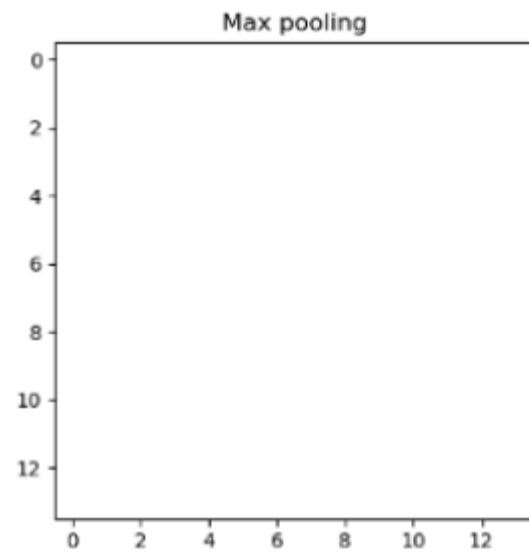
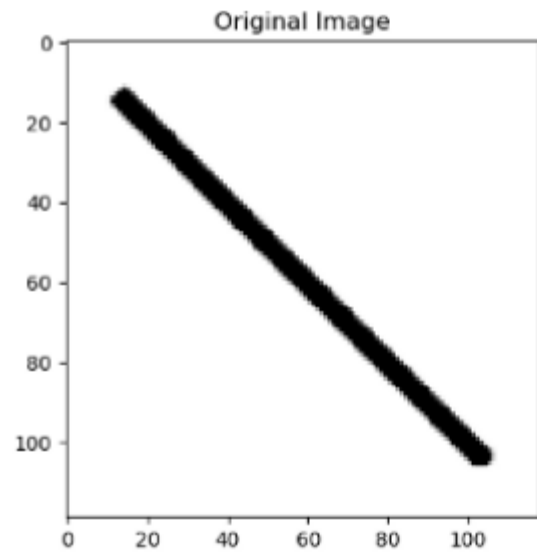
→

1	0	0
0.11	0	0.33
0.33	0	0
0	0	0
0	0	0

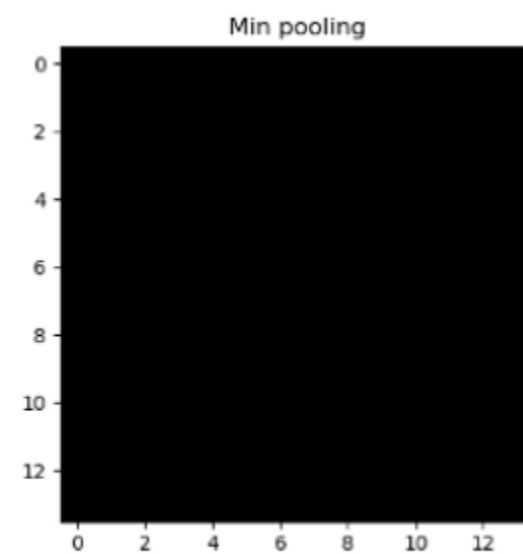
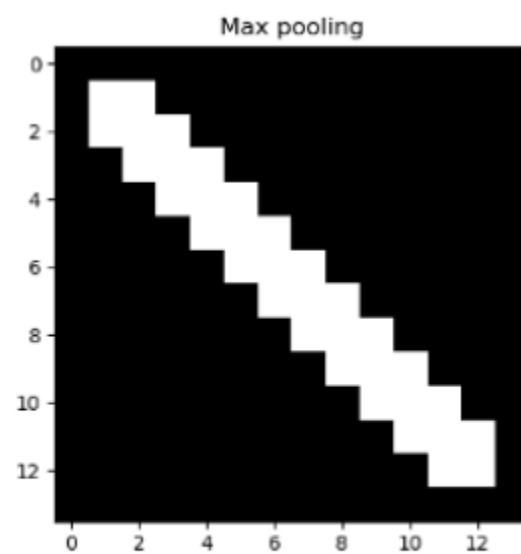
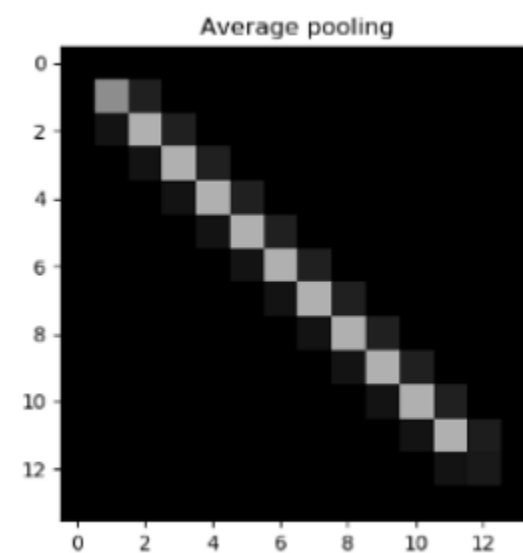
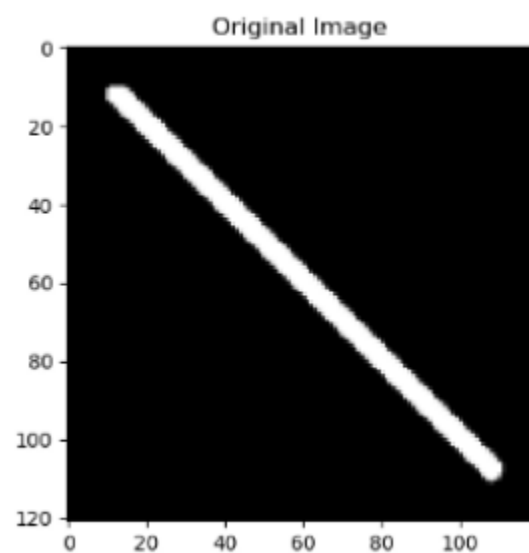
Max
pooling

→

1	0.33
0.33	0.33
0.33	0
0	0



Min pooling gives better result for images with white background and black object



Max pooling gives better result for the images with black background and white object (Ex: MNIST dataset)

Ainsi

- Pooling reduces dimensions and computation
- Reduces overfitting as there are less parameters
- Model is tolerant towards variations and distortions because if there is a distortion, by picking the maximum we filter the noise

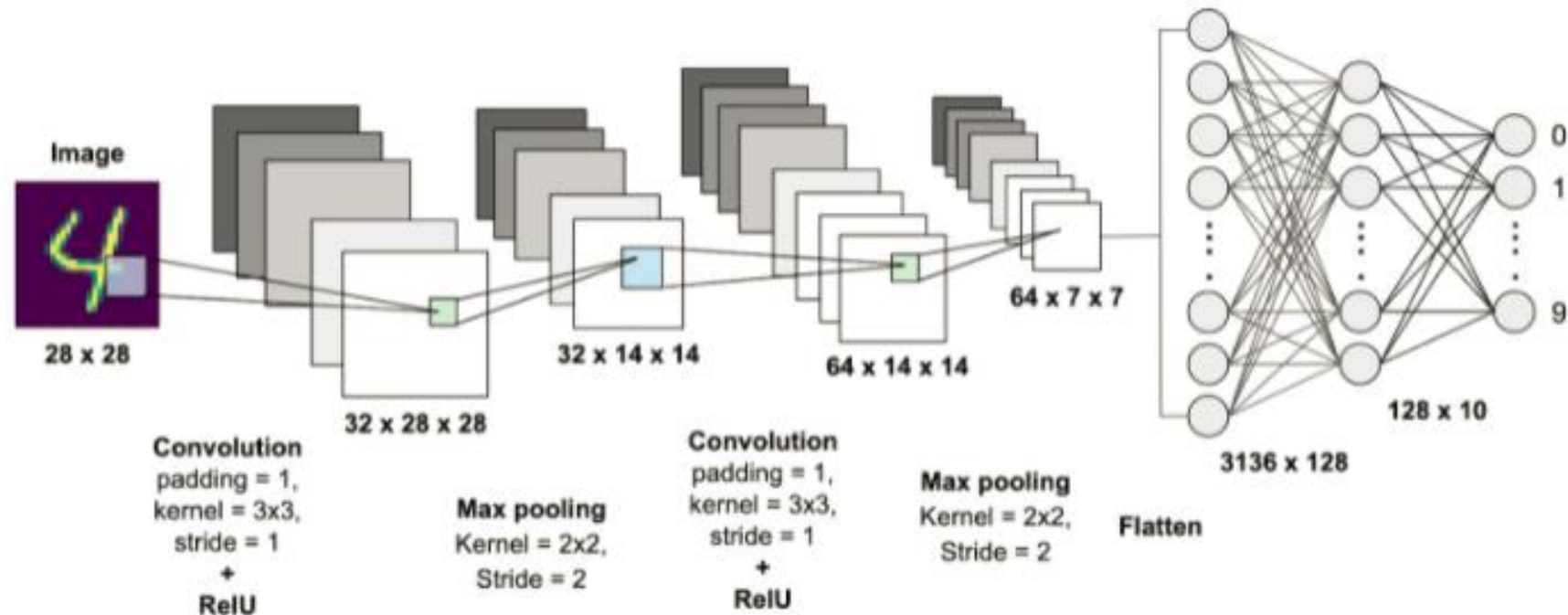
Keras :pooling layer

```
tf.keras.layers.MaxPooling2D(  
    pool_size=(2, 2), strides=None, padding="valid", data_format=None, **kwargs  
)
```

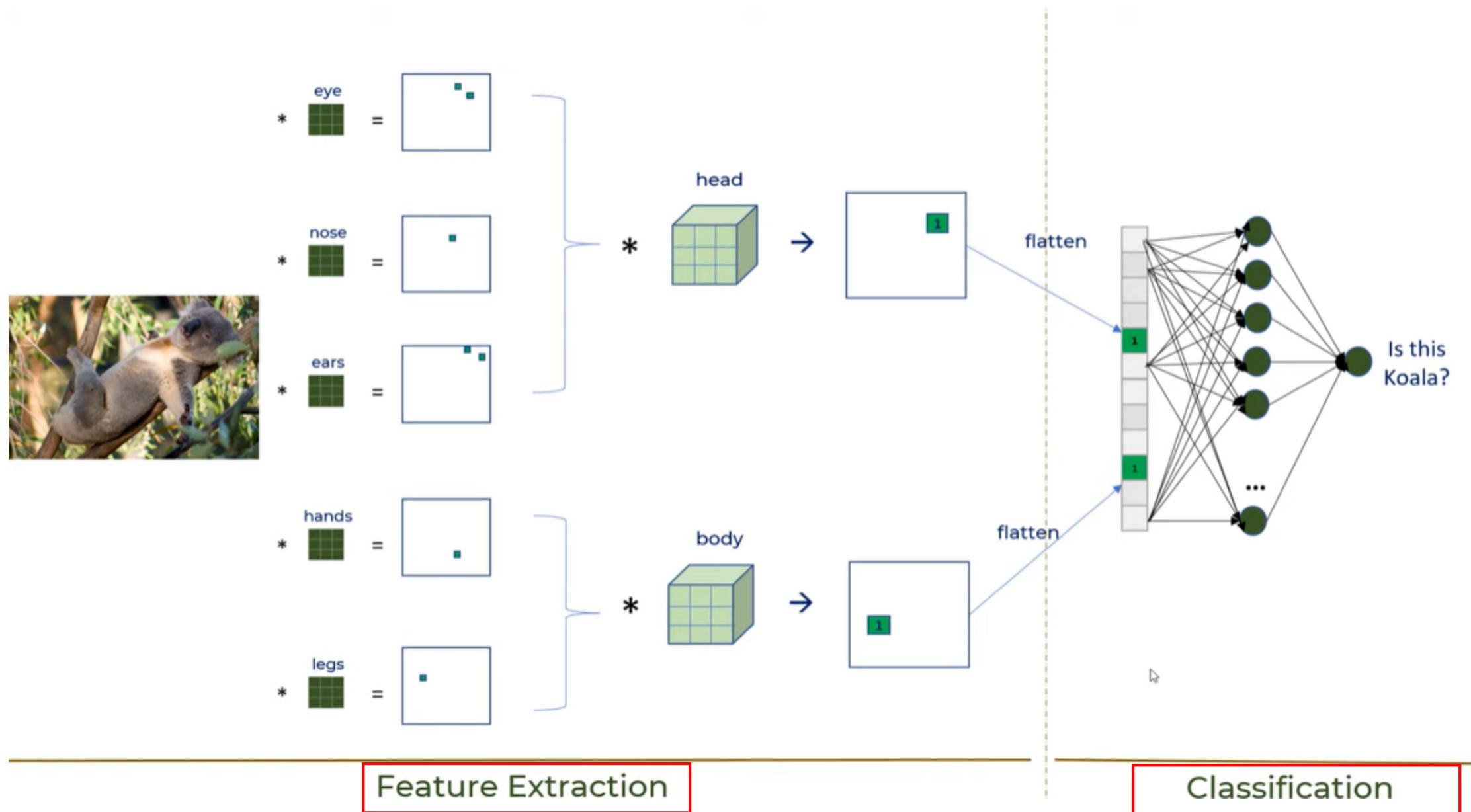
```
from tensorflow.keras.layers import MaxPooling2D  
MaxPooling2D(pool_size=(2, 2))
```

7. Classification-Fully Connected Layer (FC Layer)

- So far we've learned features through convolution, introduced non-linearity through activation functions and reduced dimensionality with pooling but we need to go further in order to use these features for classifying.
- Convolution and Pool layers output high-level features of input that fully connected layers uses for classification ; the second part of the Neural Net will have the same architecture as Deep Feed Forward Neural Net.
- We finally obtain a probability of image belonging to a particular class.



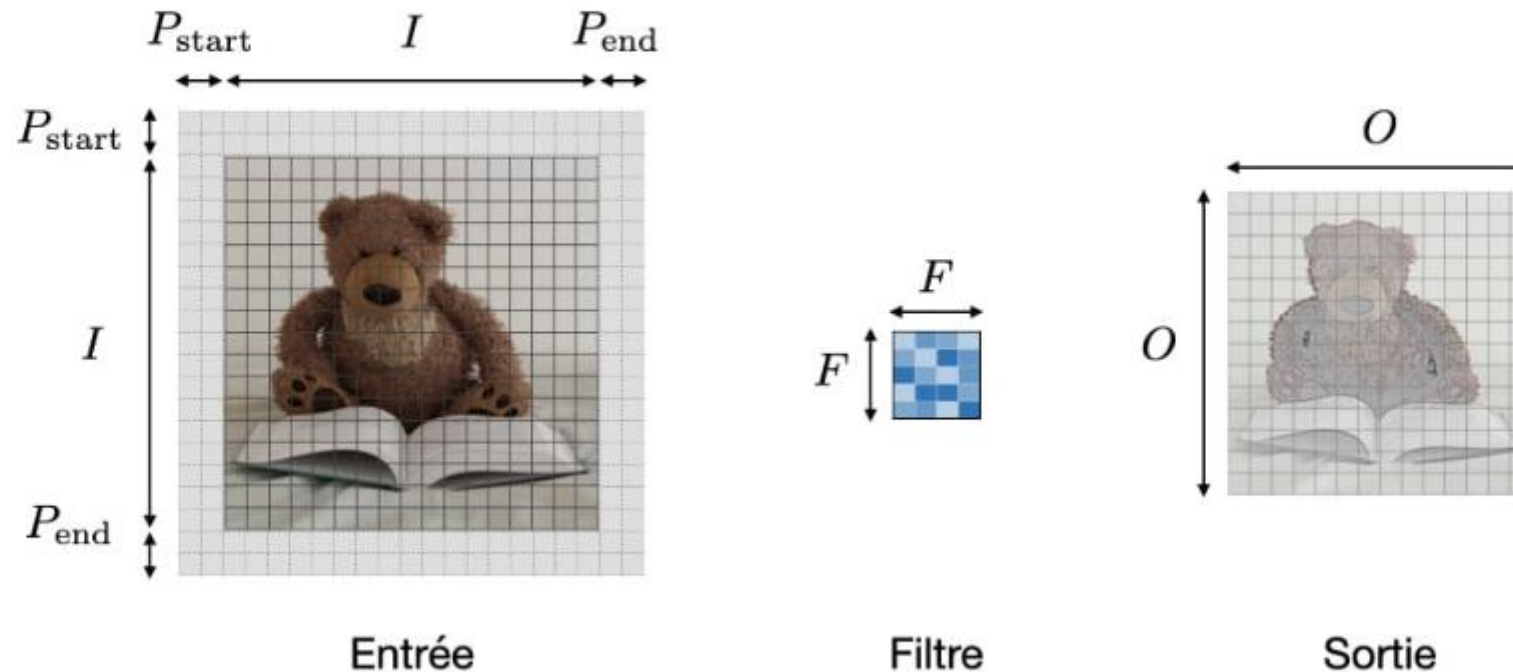
Classification-Fully Connected Layer (FC Layer)



Et les calculs ?

I : taille de l'entrée ; F = taille du filtre ; C = nombre de canaux ;
K : nombre de filtres ; S : stride

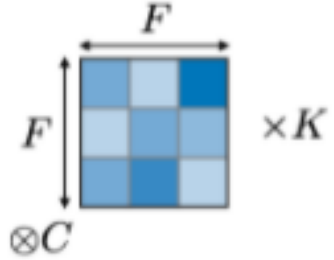
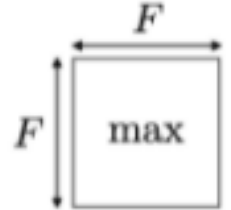
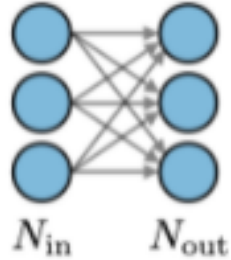
$$O = \frac{I - F + P_{\text{start}} + P_{\text{end}}}{S} + 1$$



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

Et les calculs ?

I : taille de l'entrée ; F = taille du filtre ; C = nombre de canaux ;
K : nombre de filtres ; S : stride

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

8. Des Liens

- <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>
- <https://kharshit.github.io/blog/2018/12/14/filters-in-convolutional-neural-networks>
- <https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels>
- <https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/>
- Convolutional Neural Networks (CNNs) explained
https://www.youtube.com/watch?v=YRhxdVk_sIs&ab_channel=deeplizard
- Simple explanation of convolutional neural network | Deep Learning Tutorial 23
https://www.youtube.com/watch?v=zfiSAzpy9NM&ab_channel=codebasics