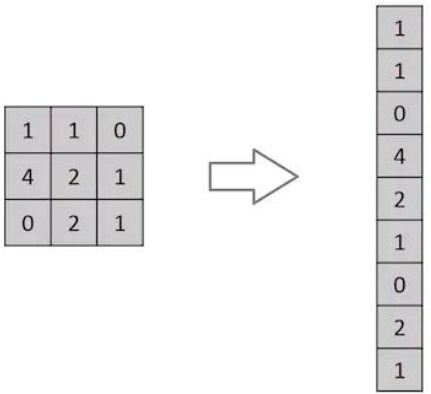
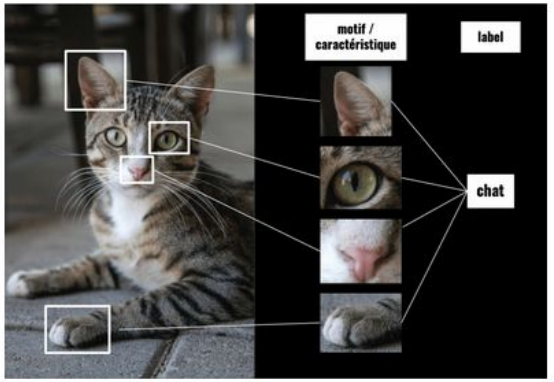
Convolutional Neural Network

Motivation

- La performance de réseau de neurone (RN) dépend fortement de la **qualité des *features*** préalablement trouvées.
- En pratique, l'erreur de classification n'est jamais nulle.
- Les résultats peuvent alors être améliorés en créant de nouvelles méthodes d'extraction de *features*.
- Particulièrement pour les images c’est encore plus compliqué d’utiliser le RN classiques



Les réseaux de neurones convolutifs (CNN)



Extraction de Caractéristiques

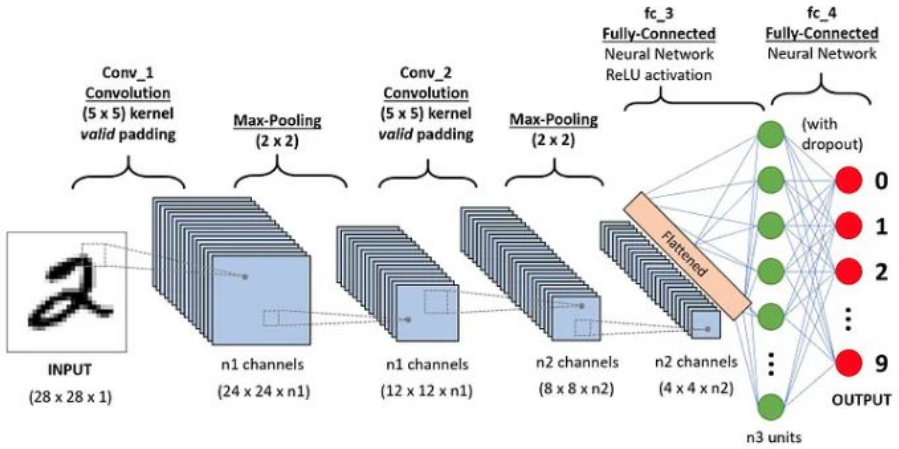
- Les CNNs sont une méthodologie similaire à

RN classique, mais plus adapté aux images

- les *features* sont apprises automatiquement
- Les CNN réalisent eux-mêmes tout le boulot fastidieux d'extraction et description de *features*
- l'architecture spécifique du réseau permet d'extraire des *features* de différentes complexités, des plus simples au plus sophistiquée

Les différentes couches d'un CNN

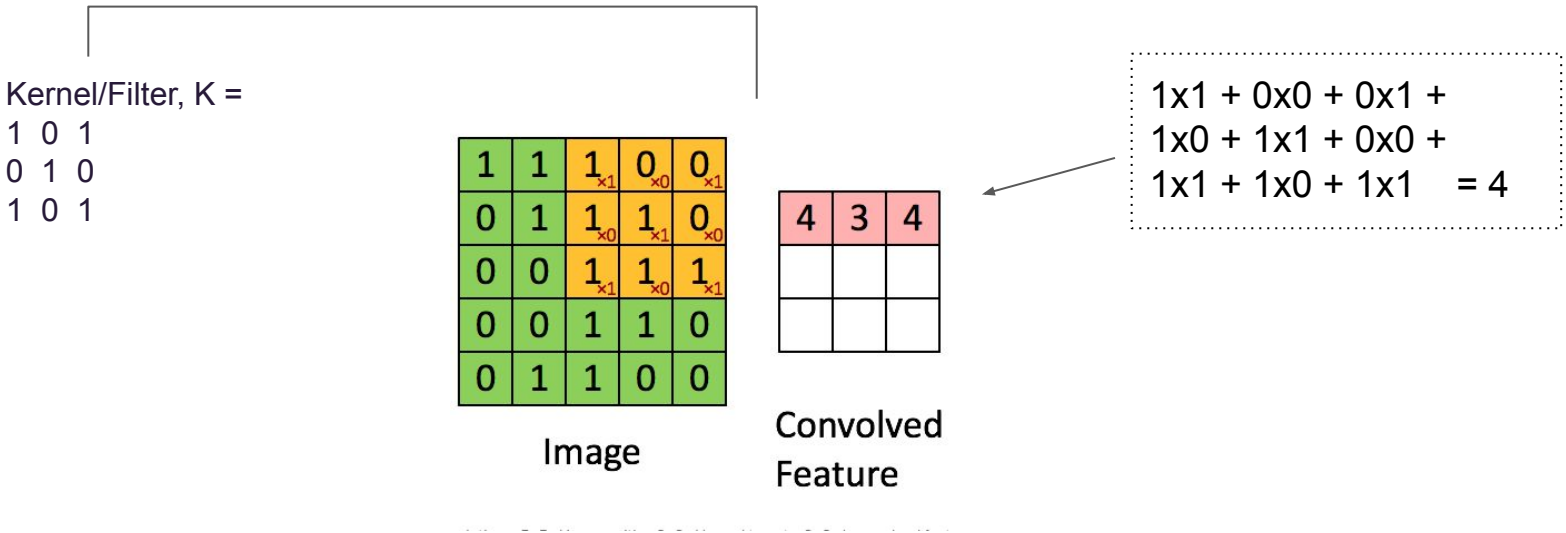
- Il existe quatre types de couches pour un réseau de neurones convolutif :
  - **convolution**,
  - ***pooling,***
  - **flatten**
  - ***fully-connected***.



La couche de convolution

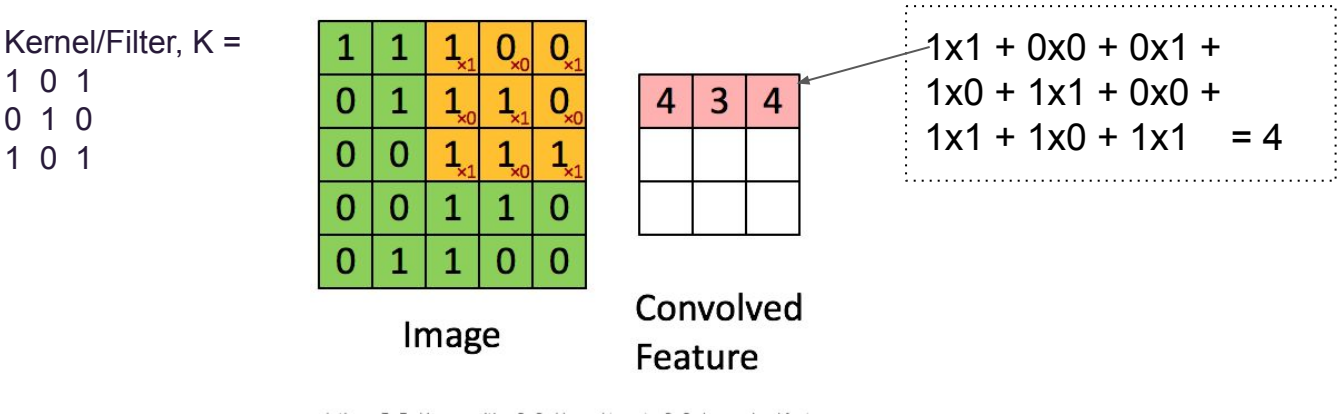
- C’est la composante clé des réseaux de neurones convolutifs, et constitue toujours au moins leur première couche.
- Son but est de repérer la présence d'un ensemble de *features* dans les entrées (imagens, etc)
- On réalise un filtrage par convolution :
  - le principe est de faire "glisser" une fenêtre représentant la *feature* sur l'image,
  - puis de calculer le produit de convolution entre la *feature* et chaque portion de l'image balayée.
  - Une *feature* est alors vue comme un filtre/kernel

La couche de convolution



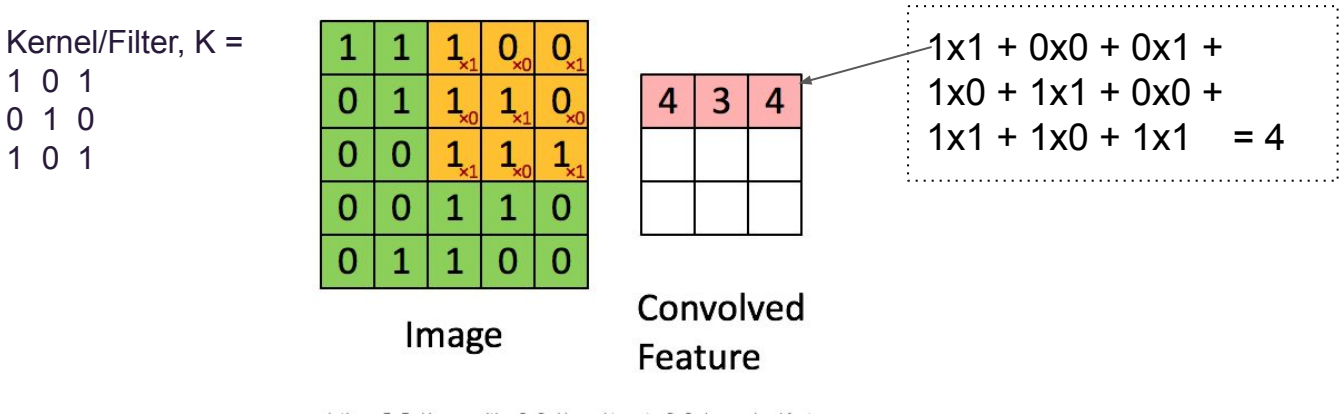
La couche de convolution

- La couche de convolution reçoit donc en entrée plusieurs images,
- et calcule la convolution de chacune d'entre elles avec chaque filtre.
- Les filtres correspondent exactement aux *features* que l'on souhaite retrouver dans les images.



La couche de convolution

- On obtient pour chaque paire (image, filtre) un ***feature map***, qui nous indique où se situent les *features* dans l'image : plus la valeur est élevée, plus l'endroit correspondant dans l'image ressemble à la *feature*.



Type de Filtre

1d

2d

3d

Comment choisir les filters ?

- les *filters* **ne sont pas** pré-définies mais apprises par le réseau lors la phase d'entraînement !
- Ils sont initialisés puis mis à jour par **rétropropagation du gradient**.

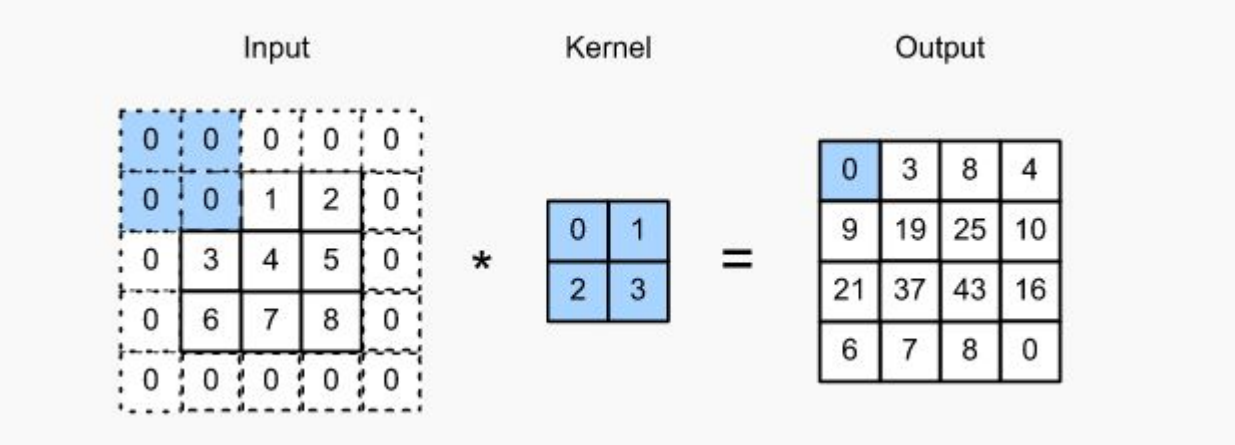
Kernel/Filter, K1=
1 0 1
0 1 0
1 0 1

Kernel/Filter, K2=
0 0 0
1 0 1
0 1 0

Paramètres de couche convolution - Padding

- Pour une entrée de dimension (n x n) et un kernel (f x f), les dimensions de la sortie après une opération de convolution est (n - f + 1) x (n - f + 1).

n=5; f= 2



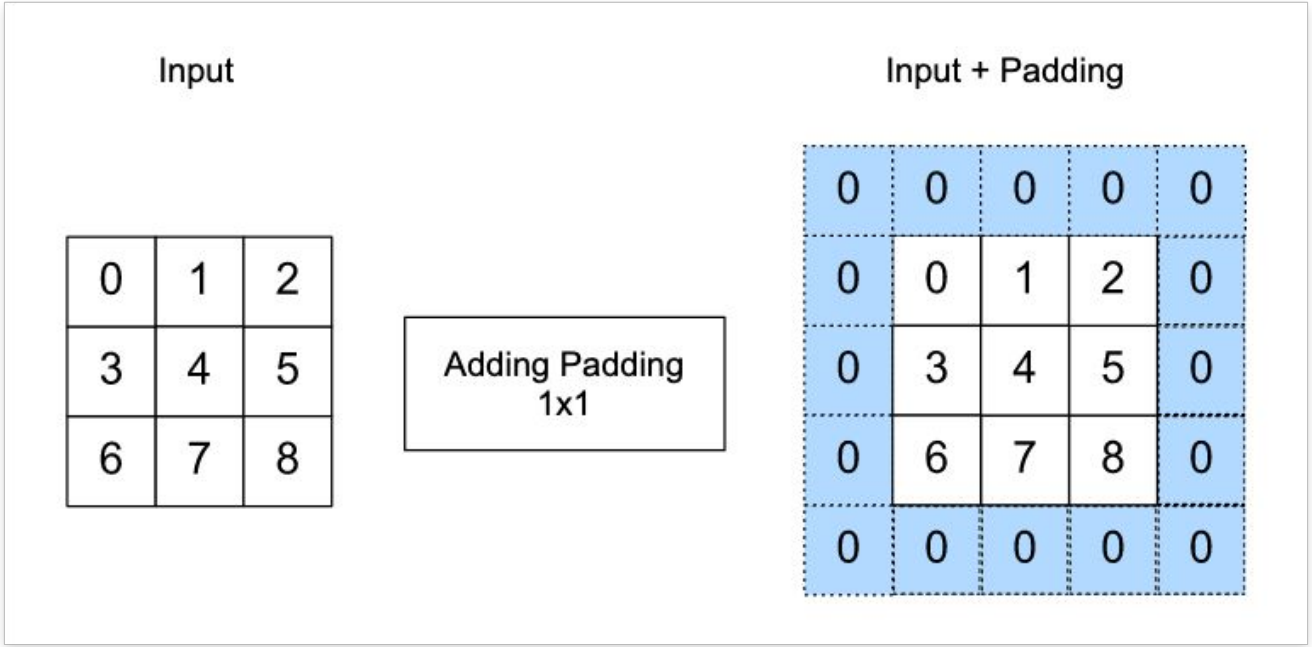
(5 x 5) (2 x 2) (5 - 2 + 1 x 5 - 2 + 1)
(4 x 4)

Paramètres de couche convolution - Padding

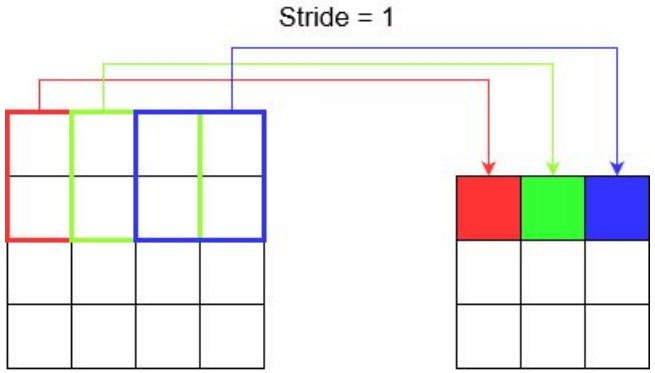
- Ainsi, les données **rétrécissent** à chaque fois qu'une opération de convolution est effectuée.
- Cela impose une limite supérieure au nombre de fois qu'une telle opération peut être effectuée avant que les données ne se réduisent à néant, ce qui nous empêche de construire des réseaux plus profonds.

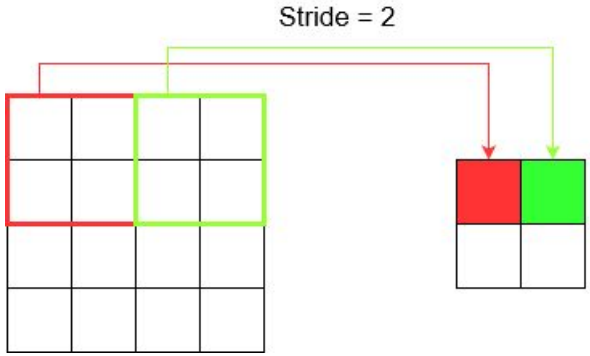
Paramètres de couche convolution - Padding

- Padding consiste simplement à ajouter des couches de zéros à nos données d'entrée afin d'éviter ces problèmes.



Paramètres de couche convolution - Stride

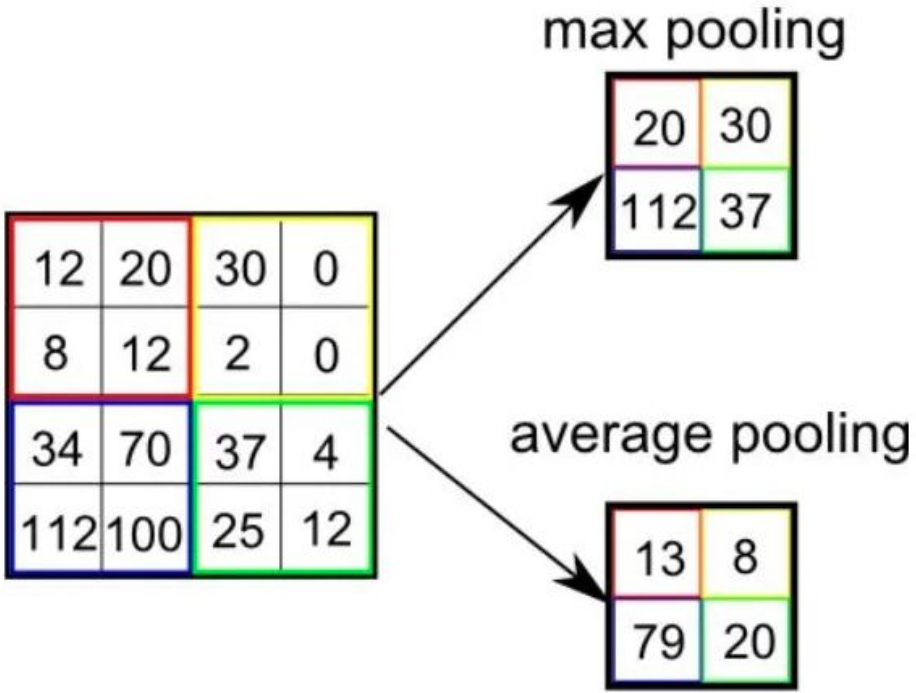




La couche de pooling

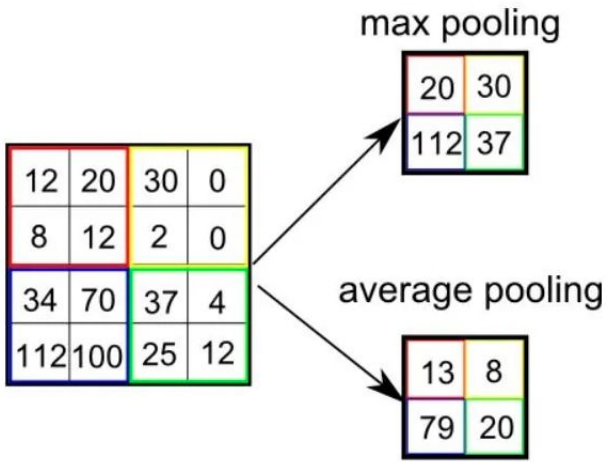
- L'opération de *pooling* consiste à **réduire la taille des output de la couche de convolution**, tout en préservant leurs caractéristiques importantes.
- Elles sont souvent placé entre deux couches de convolution : elle reçoit en entrée plusieurs *feature maps*,
- Pour cela, on découpe l'image en cellules régulières, puis on garde au sein de chaque cellule la valeur maximale (max pooling) ou la moyenne (average pooling)

La couche de pooling



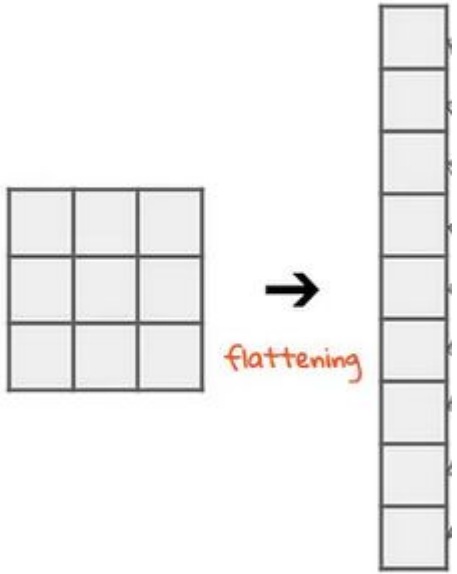
La couche de pooling

- En pratique, on utilise souvent des **cellules carrées de petite taille** pour ne pas perdre trop d'informations.
- La couche de pooling permet de réduire le nombre de paramètres et de calculs dans le réseau.
- On améliore ainsi l'efficacité du réseau et on évite le sur-apprentissage



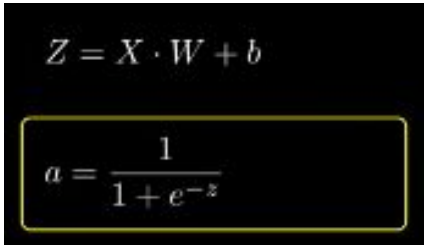
La couche Flatten

- Flatten permet d’aplatir l’entré et réduire sa dimension

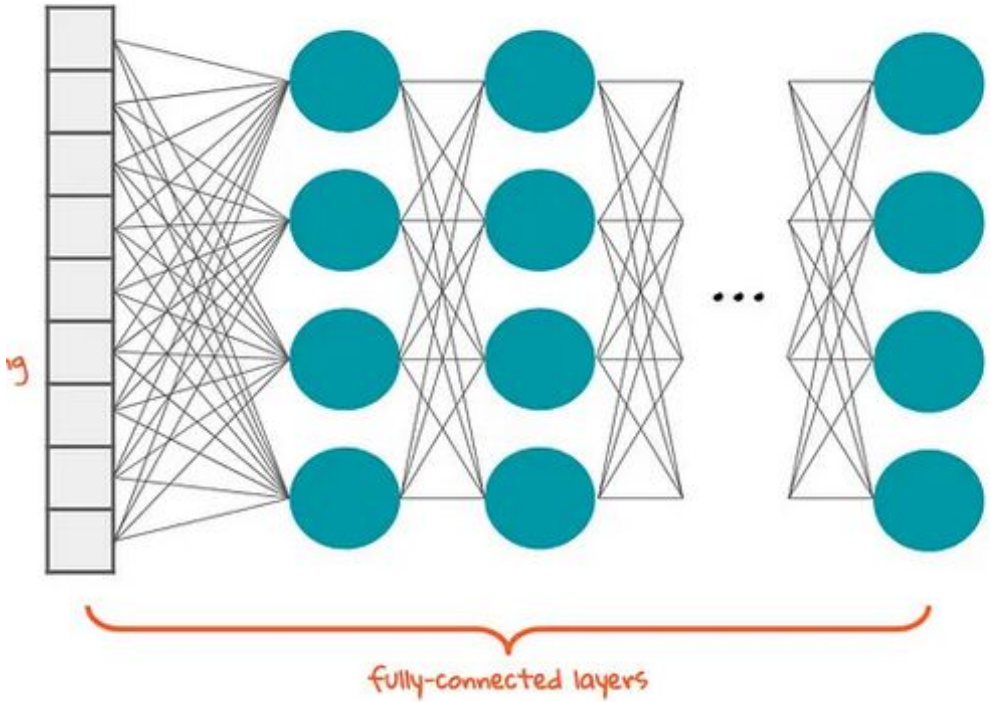


La couche fully-connected

- C’est toujours la dernière couche d'un CNN ou non – elle n'est donc pas caractéristique d'un CNN.
- Elle reçoit un vecteur en entrée et produit un nouveau vecteur en sortie.
- Pour cela, elle applique une **combinaison linéaire** puis éventuellement une **fonction d'activation** aux valeurs reçues en entrée (les neurones classiques).

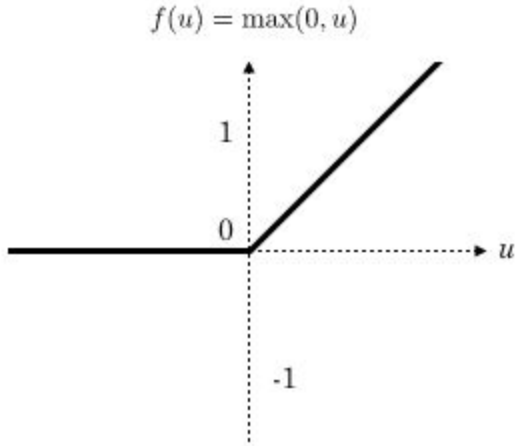


La couche fully-connected

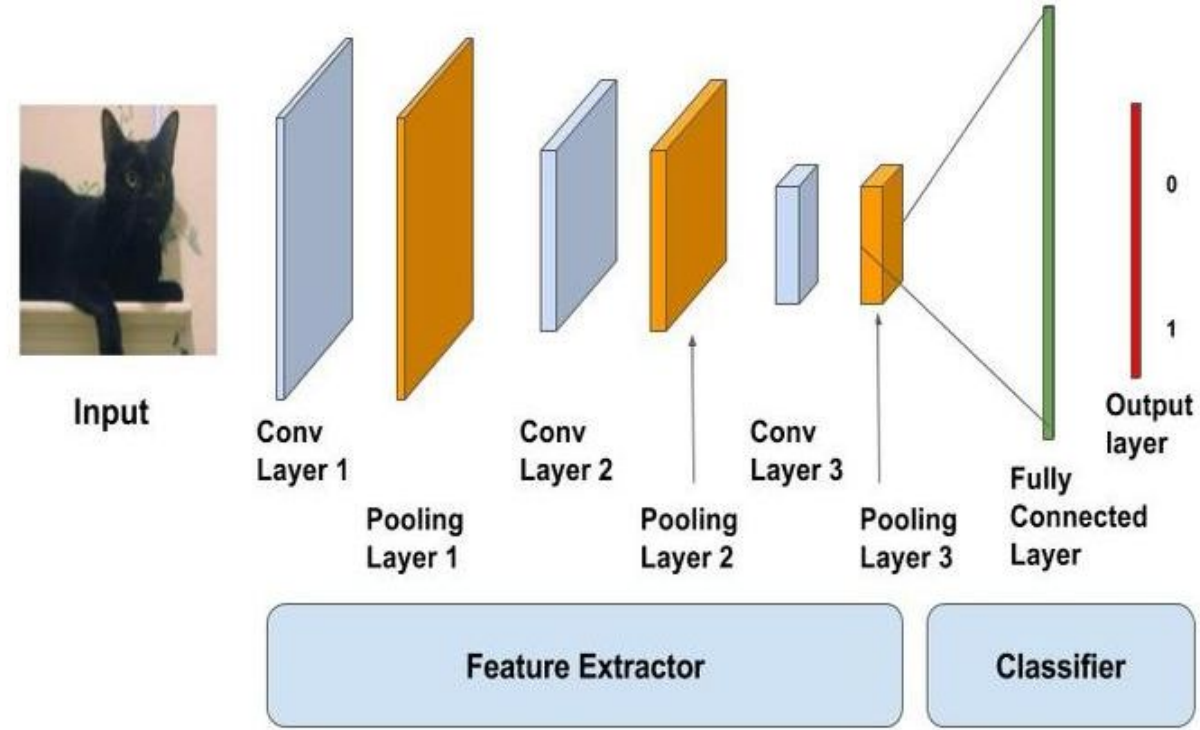


Fonction d’activation ReLU

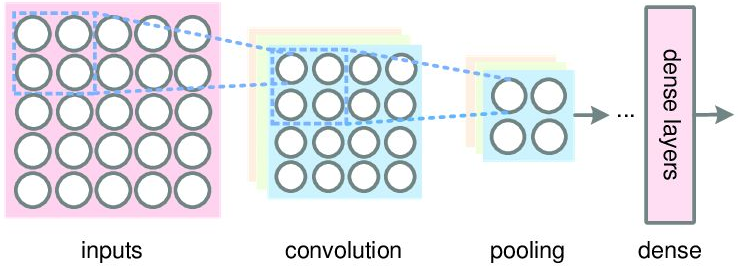
- ReLU (Rectified Linear Units) désigne la fonction réelle non-linéaire définie par ReLU(x)=max(0,x)
- La couche de correction ReLU remplace donc toutes les valeurs négatives reçues en entrées par des zéros. Elle joue le rôle de fonction d'activation



Implementation CNN

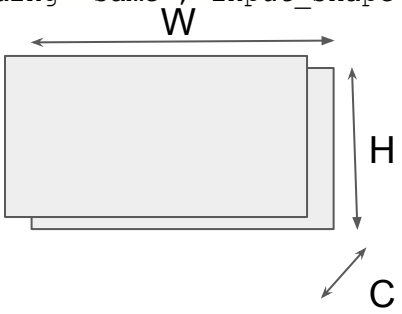


Implementation CNN




```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.models import Sequential
```

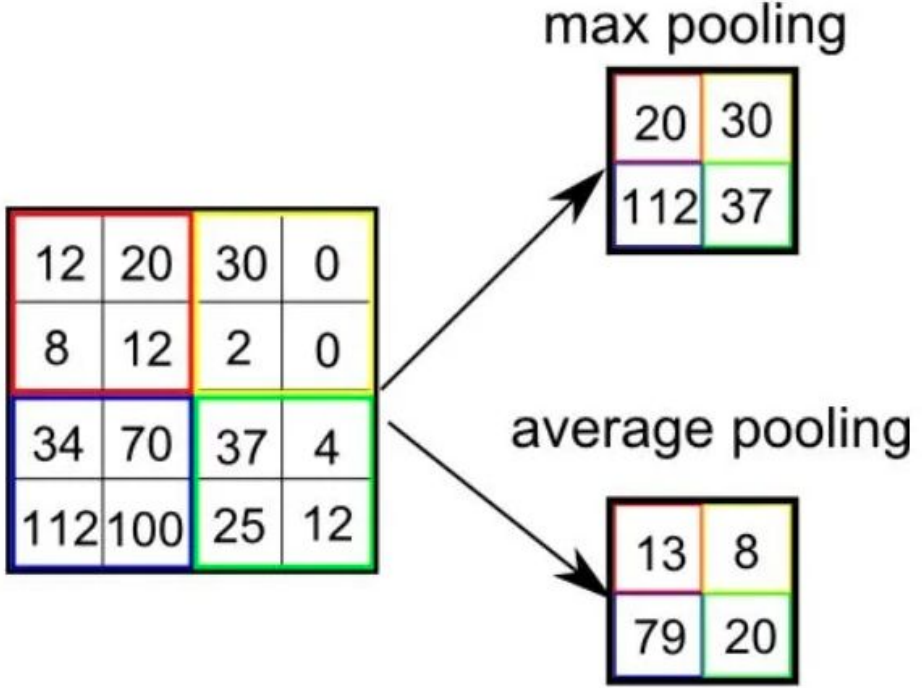
```
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1, 1), padding="same", input_shape=(W, H, C)
activation='relu'))
```



- **64 filtres**
- **un kernel de taille (3, 3)**
- **un pas de (1,1)**, aussi appelé **strides** (par défaut il est égal à (1,1)),
- **une fonction d'activation**

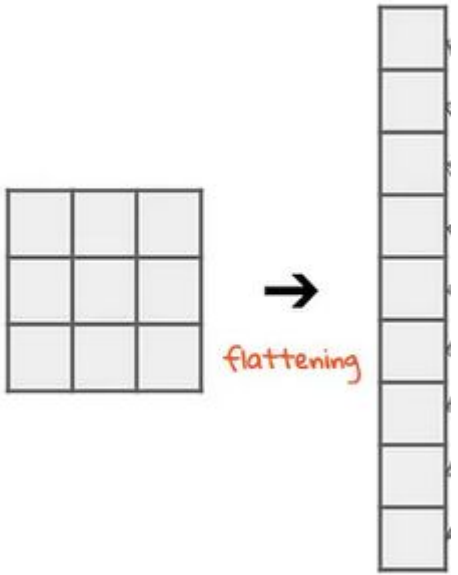
Implementation CNN

```
model.add(MaxPool2D(2,2))
```



Implementation CNN

```
model.add(Flatten())
```



Implementation CNN

```
model.add(Dense(4,activation='relu'))
```

Implementation CNN

```
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Implementation CNN

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
from tensorflow.keras.models import Sequential
```

```
model = Sequential()
model.add(Conv2D(filters=64, kernel_size=(3,3), strides=(1, 1), padding="same",
input shape=(W, H, C) activation='relu'))
model.add(MaxPool2D(2,2))
model.add(Flatten())
model.add(Dense(4,activation='relu'))
```

Implementation CNN

```
from sklearn.model_selection import train_test_split



X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)


#fitting the model
model.fit(X_train,y_train,epochs=50)

predicted_labels = model.predict(np.stack(X_test))
```