

# Convolutional Neural Networks

Thomas Ranvier

Ynov

# Sommaire

① Introduction

② Convolution

③ Architecture d'un CNN

④ Applications

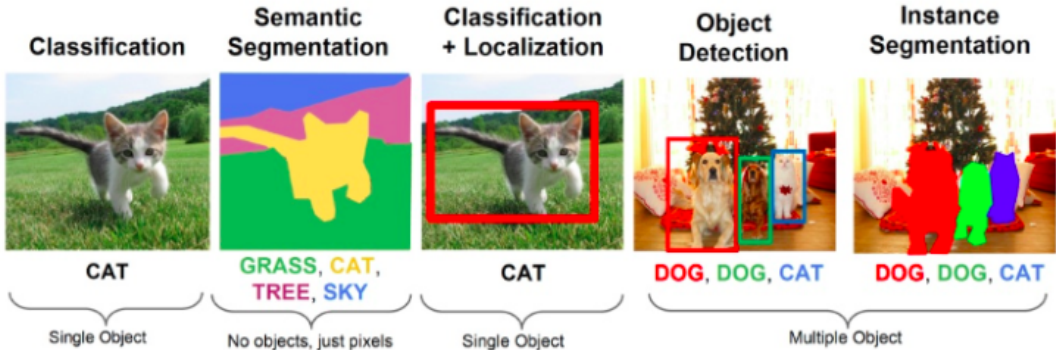
# Introduction

## Convolutional Neural Networks

# La vision par ordinateur

## Pourquoi ?

On souhaite permettre à un ordinateur de voir et comprendre des images pour être ensuite capable de réaliser différentes tâches

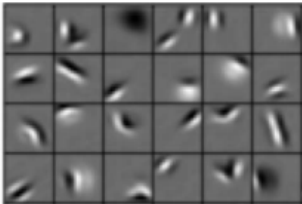


# Apprendre automatiquement les features à partir d'images

## Objectif

On souhaite que le modèle soit capable d'apprendre les features définissant des objets de lui-même, sans que l'on ait besoin de les définir manuellement

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

High level features



Facial structure

# Représentation numérique d'une image

## Représentation numérique

- Les images sont constitués d'un ensemble de pixel organisés sur une grille 2D : (1920, 1080)





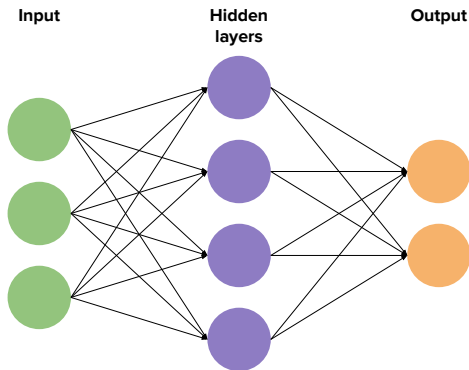




# Vision par ordinateur avec un MLP

## Désavantages d'un MLP pour le traitement d'images

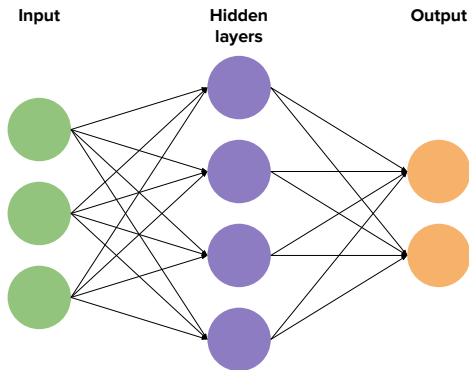
- Perte de l'information spatiale 2D



# Vision par ordinateur avec un MLP

## Désavantages d'un MLP pour le traitement d'images

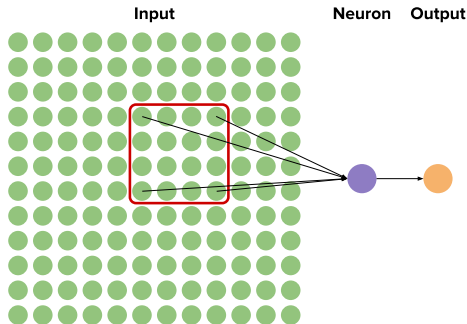
- Perte de l'information spatiale 2D
- Quantité énorme de paramètres, chaque neurone de la première couche est relié à chaque pixel



# Vision par ordinateur avec un modèle spécialisé pour le traitement d'images

## Intuition d'une convolution

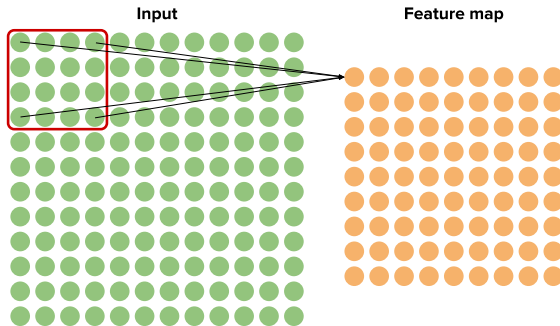
- Définir un neurone comme une fenêtre opérant sur une partie de l'image



# Vision par ordinateur avec un modèle spécialisé pour le traitement d'images

## Intuition d'une convolution

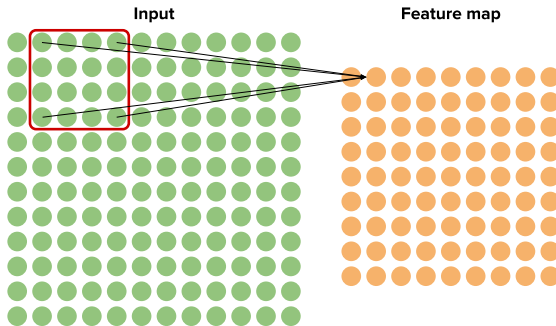
- Définir un neurone comme une fenêtre opérant sur une partie de l'image
- Faire parcourir la fenêtre glissante sur l'ensemble de l'image pour générer une feature map



# Vision par ordinateur avec un modèle spécialisé pour le traitement d'images

## Intuition d'une convolution

- Définir un neurone comme une fenêtre opérant sur une partie de l'image
- Faire parcourir la fenêtre glissante sur l'ensemble de l'image pour générer une feature map
- Utiliser  $n$  neurones pour générer une feature map de profondeur  $n$  (on parlera de  $n$  "channels")



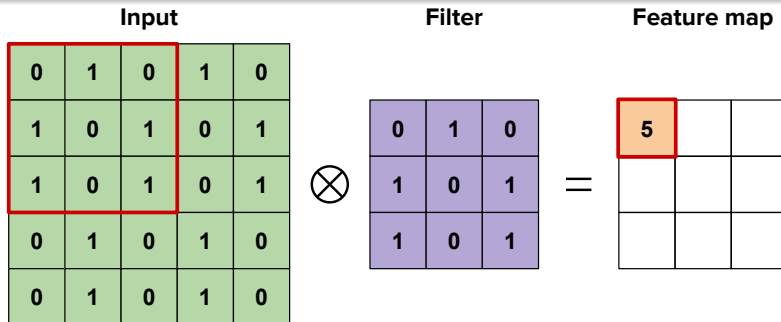
# Convolution

L'opération de convolution

# Intuition derrière une convolution

## Intuition derrière une convolution

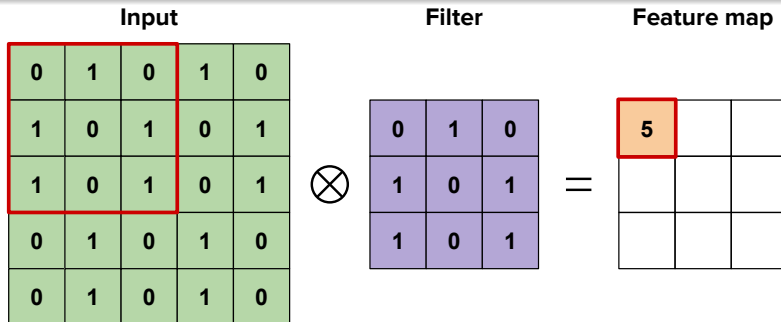
- On appelle les neurones d'une couche de convolution des "filtres"



# Intuition derrière une convolution

## Intuition derrière une convolution

- On appelle les neurones d'une couche de convolution des “filtres”
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle “strides”

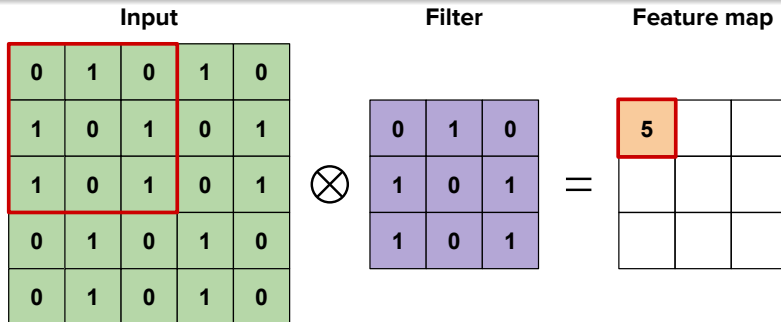




# Intuition derrière une convolution

## Intuition derrière une convolution

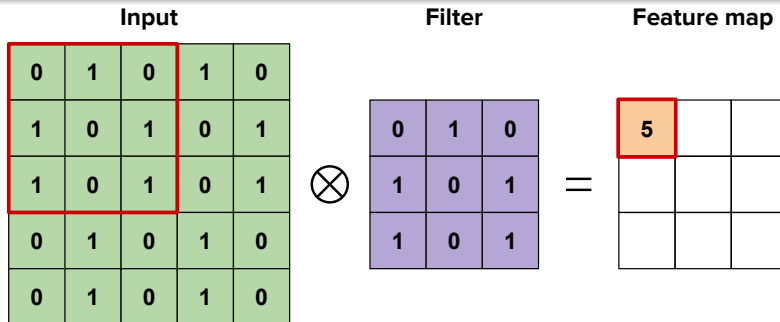
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre



# Intuition derrière une convolution

## Intuition derrière une convolution

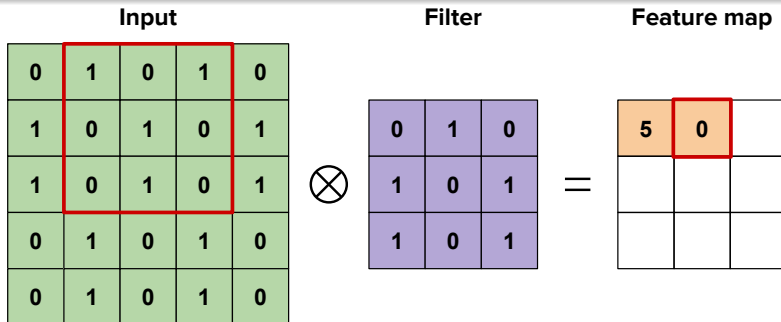
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Intuition derrière une convolution

## Intuition derrière une convolution

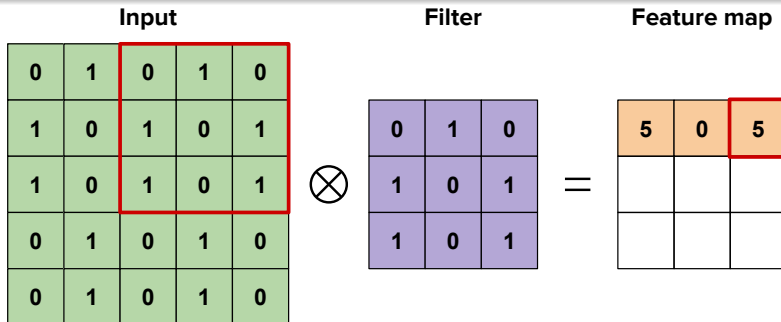
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Intuition derrière une convolution

## Intuition derrière une convolution

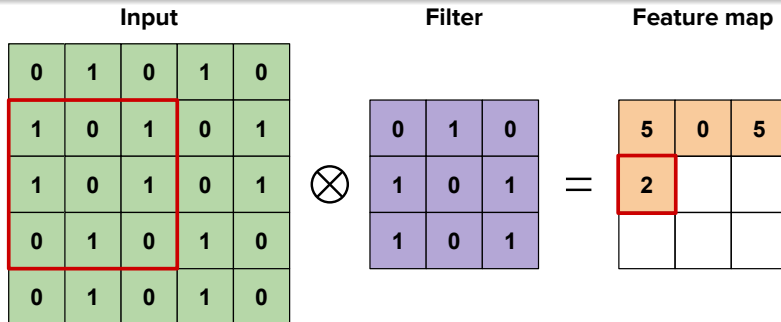
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Intuition derrière une convolution

## Intuition derrière une convolution

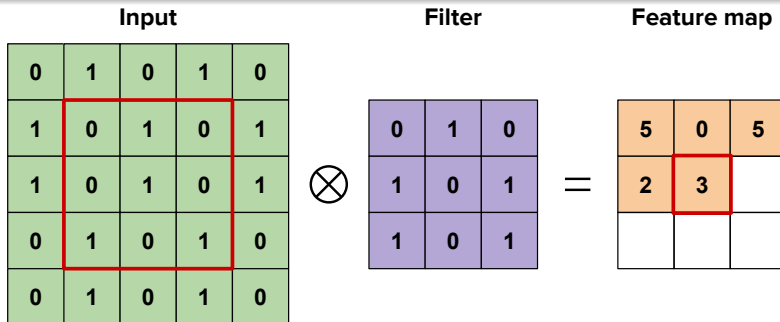
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Intuition derrière une convolution

## Intuition derrière une convolution

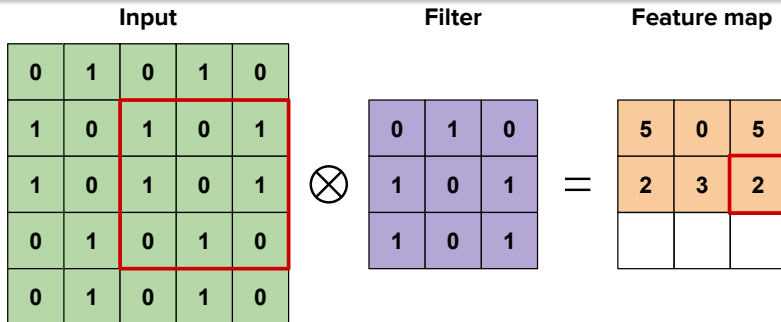
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Intuition derrière une convolution

## Intuition derrière une convolution

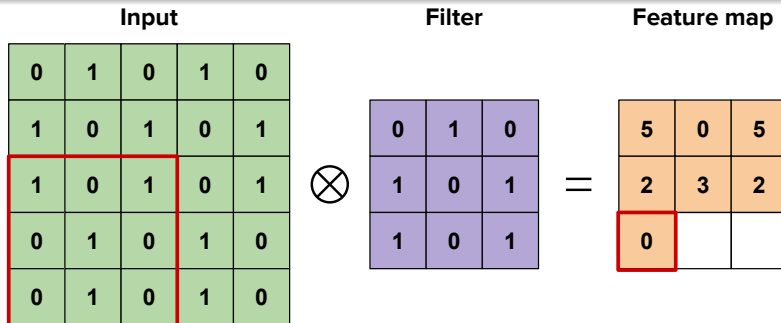
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Intuition derrière une convolution

## Intuition derrière une convolution

- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"

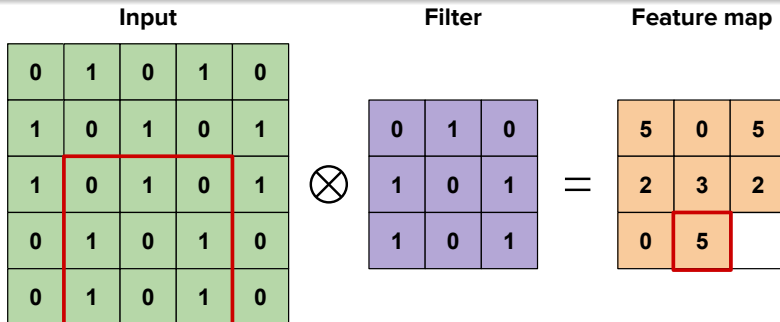




# Intuition derrière une convolution

## Intuition derrière une convolution

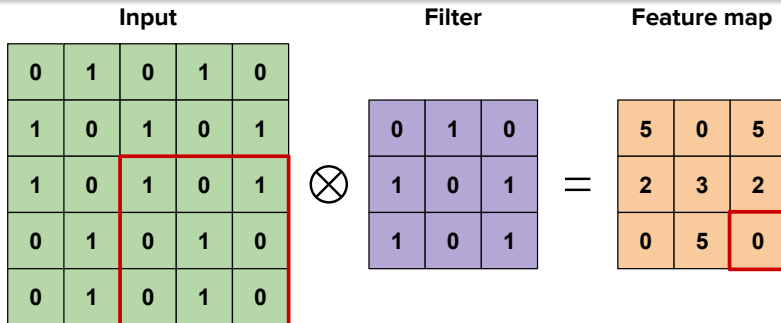
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Intuition derrière une convolution

## Intuition derrière une convolution

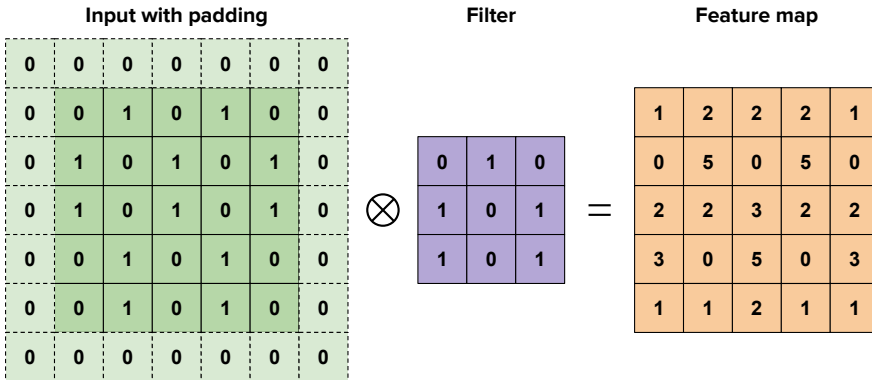
- On appelle les neurones d'une couche de convolution des "filtres"
- Les filtres se déplacent de gauche à droite et haut en bas selon un pas que l'on appelle "strides"
- Un filtre fait la somme du produit Hadamard entre les valeurs d'input et les valeurs du filtre
- On appelle l'output d'une couche à convolutions une "feature map"



# Le padding

## L'intérêt du padding

On ajoute des “pixels” à 0 sur les contours de l'image, cela permet de ne pas perdre l'information sur les bords



# Le padding

## Calculer la taille de l'output

La taille de la feature map obtenue en sortie peut être calculée avec la formule suivante :

$$O = \frac{I - K + 2P}{S} + 1$$

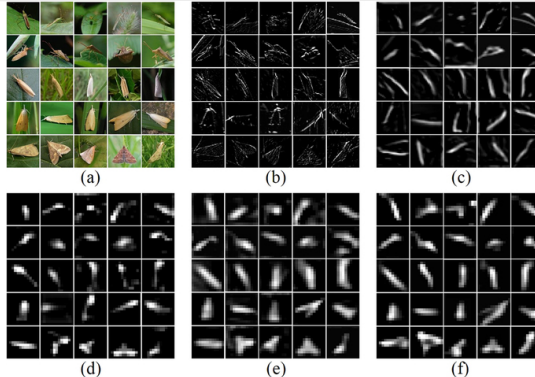
Avec :

- $I$  = Taille de l'input
- $K$  = Taille du filtre (aussi appelé kernel)
- $P$  = Taille du padding
- $S$  = Strides

# Intérêt des convolutions

## Intérêt des convolutions

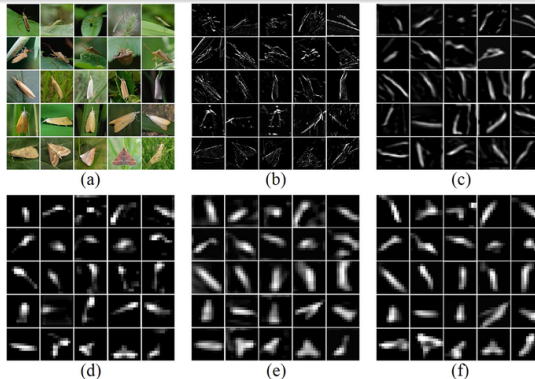
- Le but de chaque filtre de convolution est d'extraire une feature spécifique de l'image



# Intérêt des convolutions

## Intérêt des convolutions

- Le but de chaque filtre de convolution est d'extraire une feature spécifique de l'image
- Un modèle à convolutions enchaîne les couches à convolutions pour extraire des features à différents niveaux d'abstractions



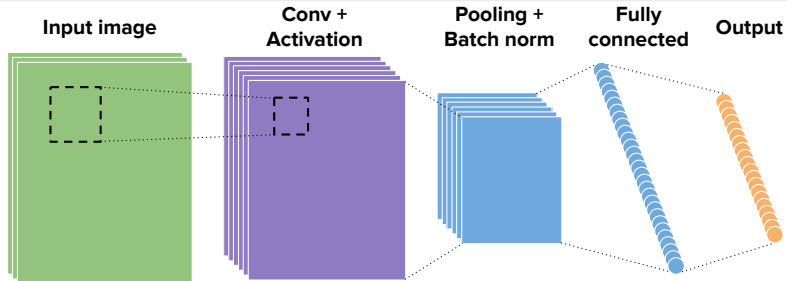
# Architecture d'un CNN

Blocs utilisables pour construire un modèle à convolutions

# Architecture type d'un CNN pour une tâche de classification

## Couches

- 1 Convolution : Extrait des features à partir de l'input donné

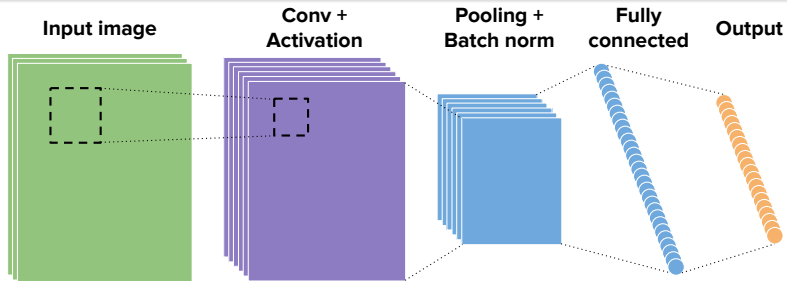




# Architecture type d'un CNN pour une tâche de classification

## Couches

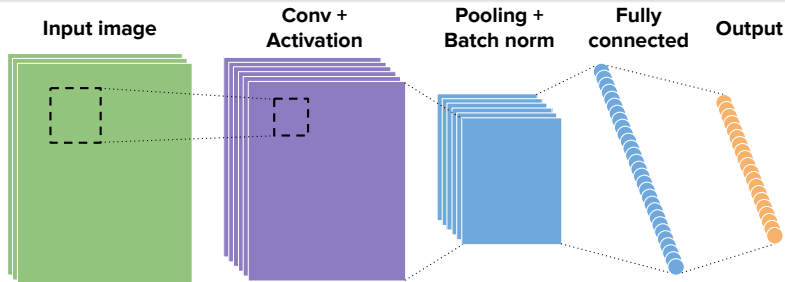
- 1 Convolution : Extrait des features à partir de l'input donné
- 2 Activation : Utilisation d'une non-linéarité après chaque couche de convolutions



# Architecture type d'un CNN pour une tâche de classification

## Couches

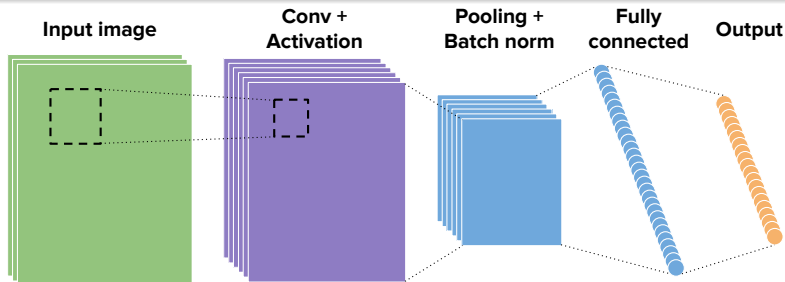
- 1 Convolution : Extrait des features à partir de l'input donné
- 2 Activation : Utilisation d'une non-linéarité après chaque couche de convolutions
- 3 Pooling : Réduit la taille de l'input



# Architecture type d'un CNN pour une tâche de classification

## Couches

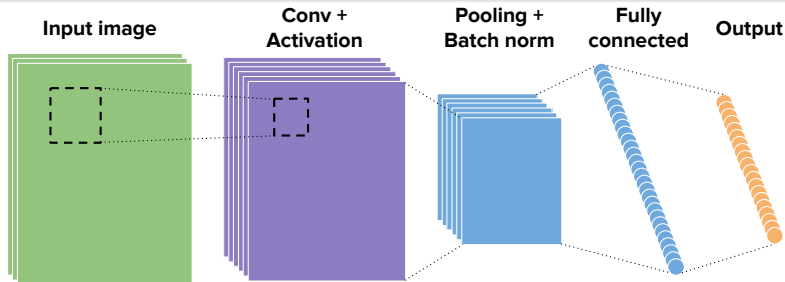
- 1 Convolution : Extrait des features à partir de l'input donné
- 2 Activation : Utilisation d'une non-linéarité après chaque couche de convolutions
- 3 Pooling : Réduit la taille de l'input
- 4 Batch norm : Normalise les données, généralement entre la partie conv et celle fully-connected



# Architecture type d'un CNN pour une tâche de classification

## Couches

- 1 Convolution : Extrait des features à partir de l'input donné
- 2 Activation : Utilisation d'une non-linéarité après chaque couche de convolutions
- 3 Pooling : Réduit la taille de l'input
- 4 Batch norm : Normalise les données, généralement entre la partie conv et celle fully-connected
- 5 Fully-connected : Utilise les features extraites par les convolutions pour classifier les données



# Convolutions avec Keras

## Conv2D

```
tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding="valid", activation=None)
```

- filters : Nombre de filtres à créer, correspondra à la profondeur de la feature map générée

# Convolutions avec Keras

## Conv2D

```
tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding="valid", activation=None)
```

- filters : Nombre de filtres à créer, correspondra à la profondeur de la feature map générée
- kernel\_size : Dimensions des filtres

# Convolutions avec Keras

## Conv2D

```
tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding="valid", activation=None)
```

- filters : Nombre de filtres à créer, correspondra à la profondeur de la feature map générée
- kernel\_size : Dimensions des filtres
- strides : Pas de déplacement des filtres sur l'input

# Convolutions avec Keras

## Conv2D

```
tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding="valid", activation=None)
```

- filters : Nombre de filtres à créer, correspondra à la profondeur de la feature map générée
- kernel\_size : Dimensions des filtres
- strides : Pas de déplacement des filtres sur l'input
- padding : "valid" n'applique pas de padding, "same" génère un padding tel que l'output ait les mêmes dimensions que l'input



# Convolutions avec Keras

## Conv2D

```
tf.keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding="valid", activation=None)
```

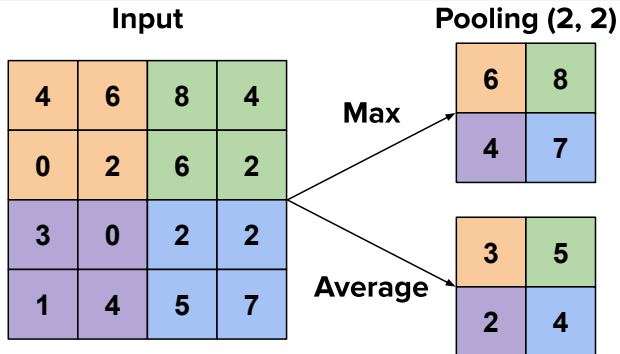
- filters : Nombre de filtres à créer, correspondra à la profondeur de la feature map générée
- kernel\_size : Dimensions des filtres
- strides : Pas de déplacement des filtres sur l'input
- padding : "valid" n'applique pas de padding, "same" génère un padding tel que l'output ait les mêmes dimensions que l'input
- activation : La non-linéarité à appliquer après la couche de convolutions

# Pooling

## Pooling

Les 2 principaux types de pooling utilisés sont les Max et Average poolings :

① `tf.keras.layers.MaxPooling2D(pool_size=(2, 2))`

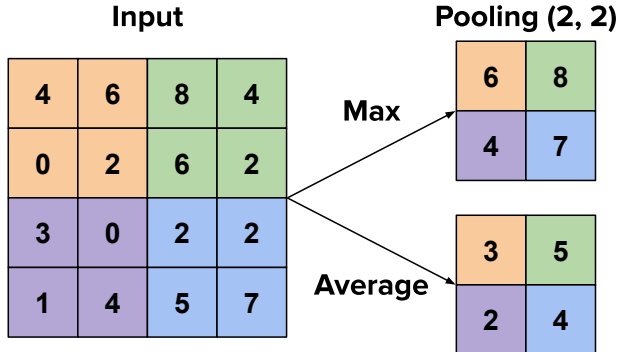


# Pooling

## Pooling

Les 2 principaux types de pooling utilisés sont les Max et Average poolings :

- 1 `tf.keras.layers.MaxPooling2D(pool_size=(2, 2))`
- 2 `tf.keras.layers.AveragePooling2D(pool_size=(2, 2))`



# Exemple d'architecture en pratique

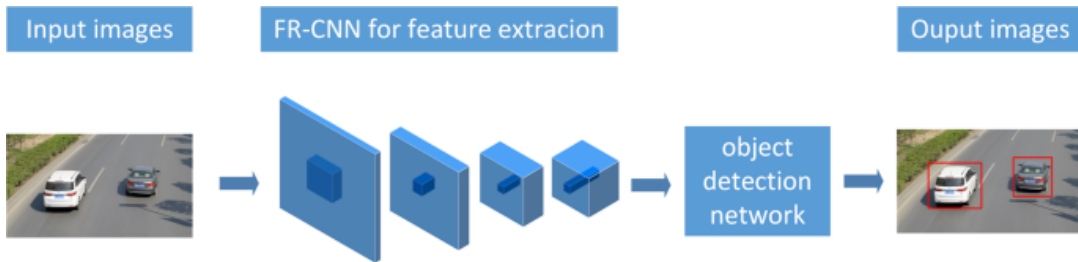
## Architecture de modèle à convolutions sous Keras

```
model = tf.keras.Sequential([  
    # First conv layer  
    tf.keras.layers.Conv2D(32, 5, strides=1, padding='same', activation='relu'),  
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),  
  
    # Second conv layer  
    tf.keras.layers.Conv2D(64, 3, strides=1, padding='same', activation='relu'),  
    tf.keras.layers.MaxPool2D(pool_size=(2,2)),  
  
    # Batch norm for stabilization  
    tf.keras.layers.BatchNormalization(),  
  
    # Flatten data into a vector for the fully-connected part  
    tf.keras.layers.Flatten(),  
  
    tf.keras.layers.Dense(units=128, activation='relu'),  
    tf.keras.layers.Dense(units=10, activation='softmax')  
])
```

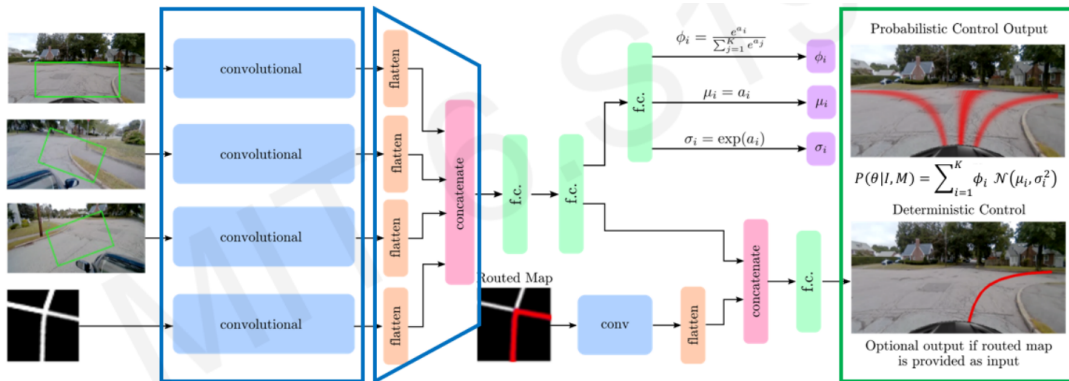
# Applications

Applications réelles de CNN

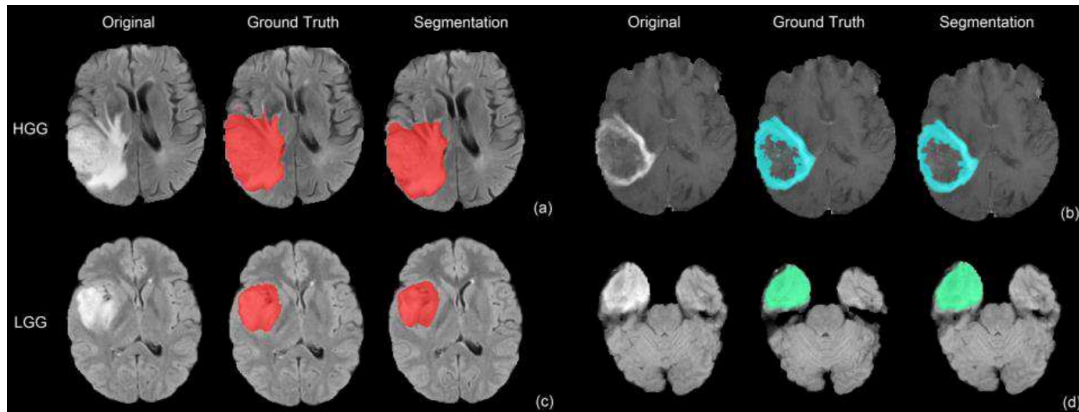
# Détection d'objets



# Voiture autonome

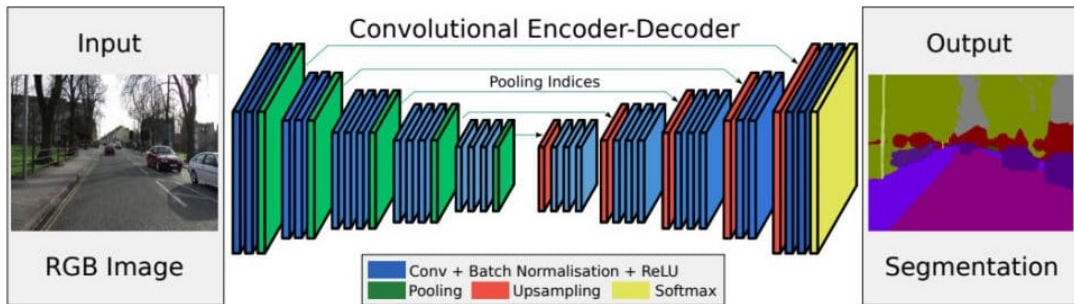


# Imagerie médicale





# Segmentation



# Transfert de style

