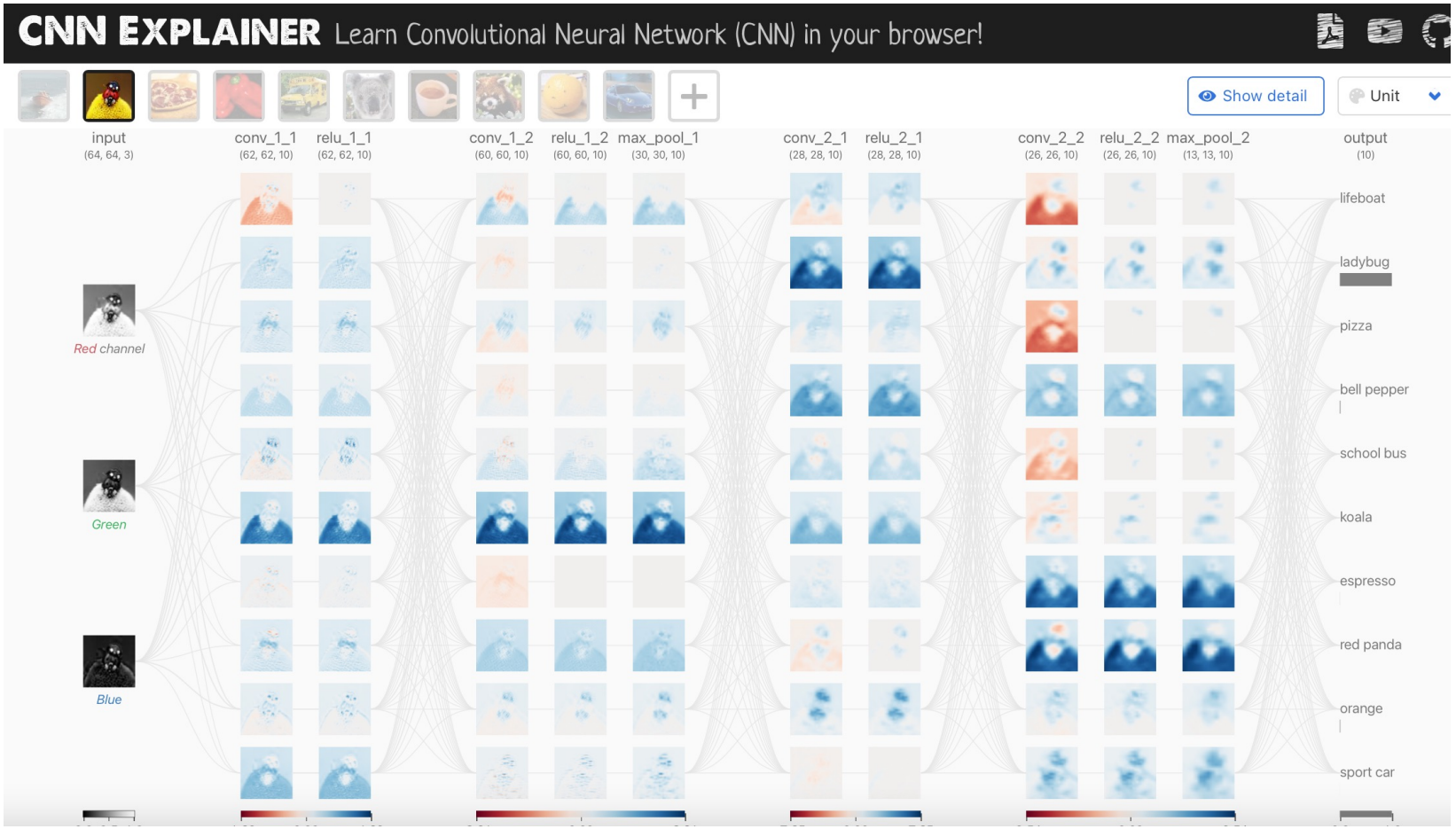
Les réseaux de neurones convolutifs (CNN)

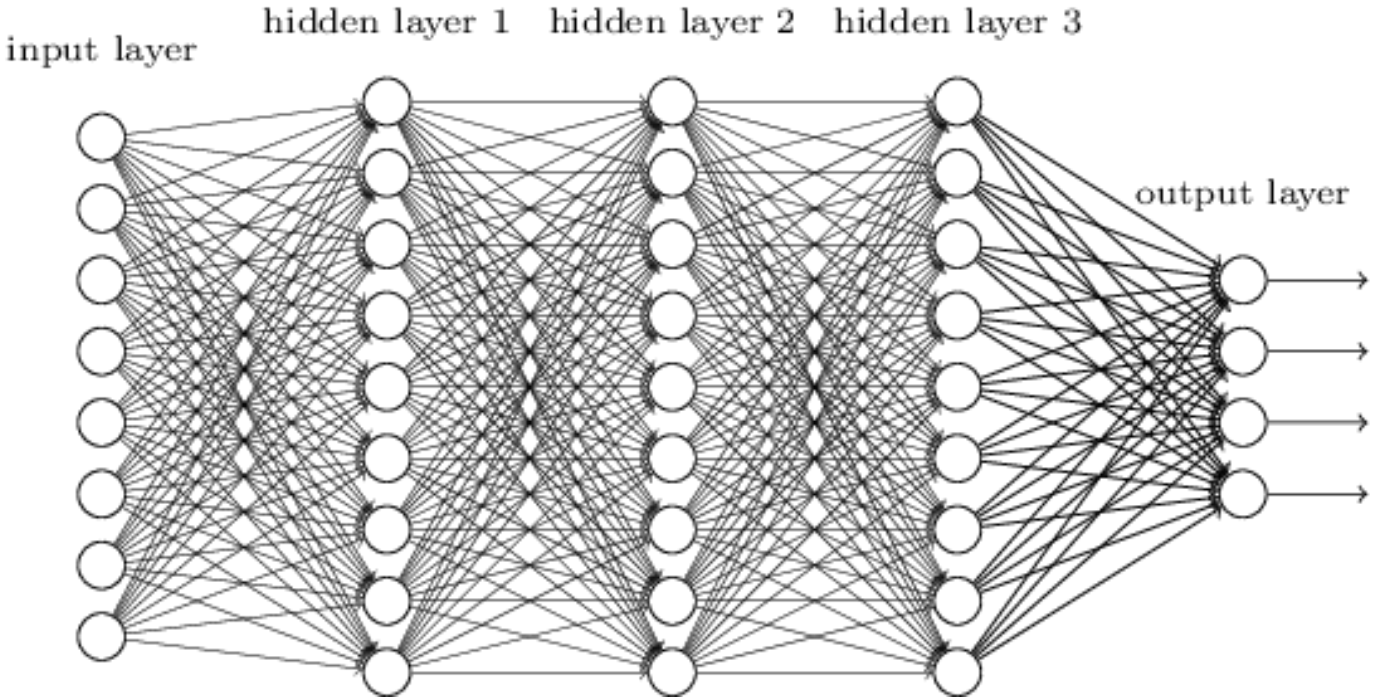
Pascal Poncelet
LIRMM
Pascal.Poncelet@lirmm.fr
http://www.lirmm.fr/~poncelet

Comprendre ce qui se passe :

- https://poloclub.github.io/cnn-explainer/

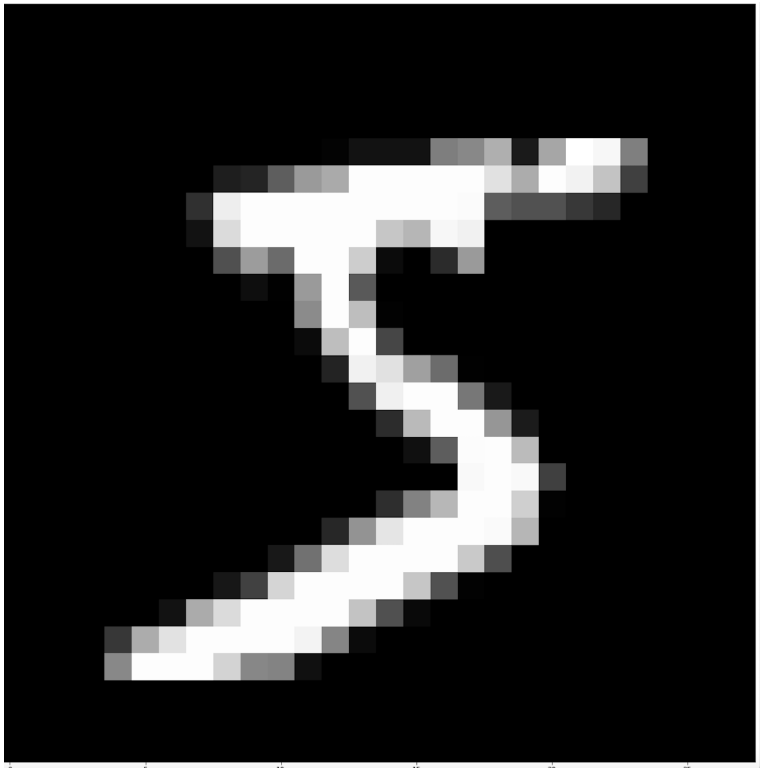


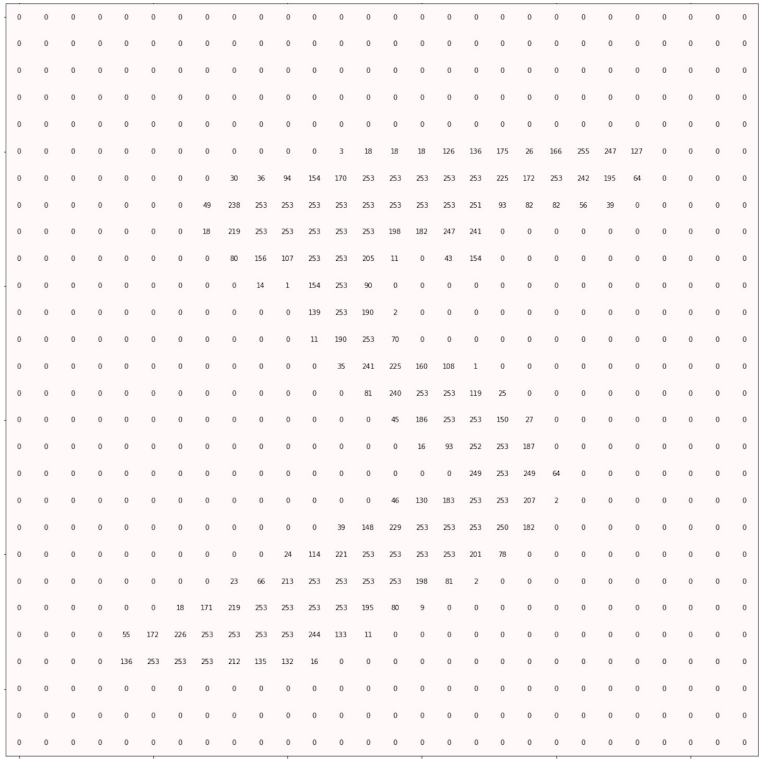
Multi-layer perceptron



- Le modèle peut être trop grand : beaucoup de paramètres à apprendre
- Input layer : « feature selection » ?
- Est-ce qu'on a besoin de tous les arcs ?

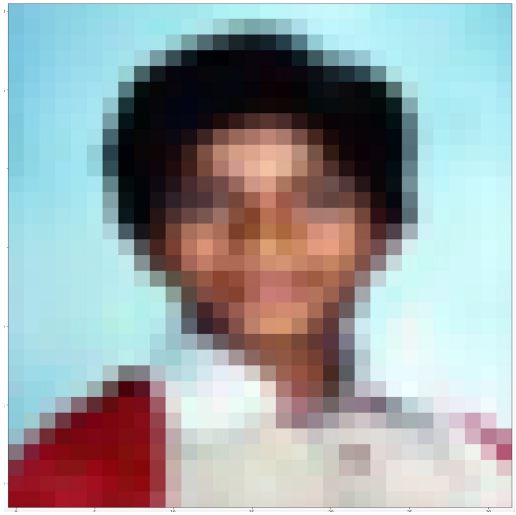
Une image

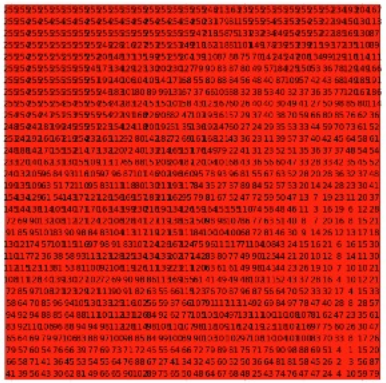


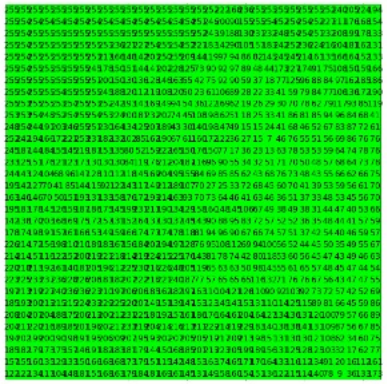


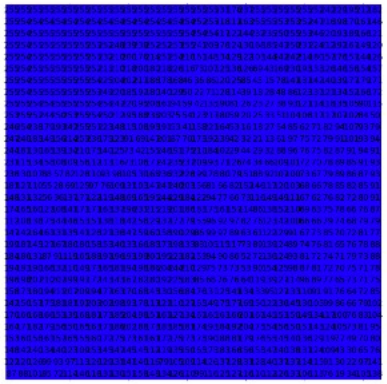
- Un « 5 » - jeu de données MNIST - en gris (valeurs comprises entre 0 et 255)
- Une image = 28×28 pixels

Une image RGB



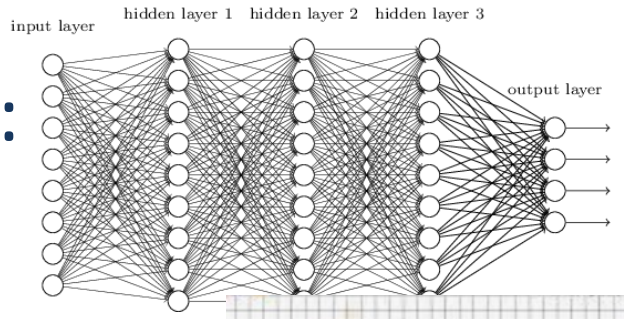






- Une personne du jeu de données CIFAR 10 en RGB
- Une image = 32×32 pixels

Avec un MLP



- Conversion pour être compatible :

- 1 image : 32 X 32 = 1024
  - Il faut un vecteur de 1024 pour la couche d'entrée
  - Si la première couche cachée possède 100 neurones :
    - 100 x 1024 = 102400 paramètres !!!!!

Avec un MLP

- Sensibilité aux variations

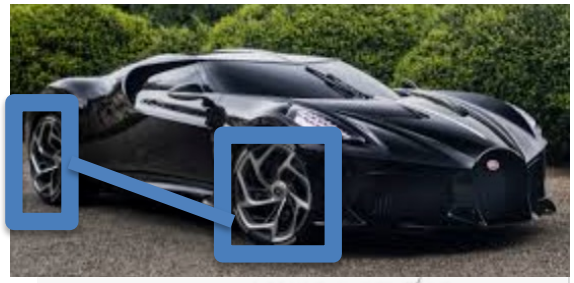
Même image ???

Avec un MLP

- Approche « sac de mots » : pas de corrélation entre les features

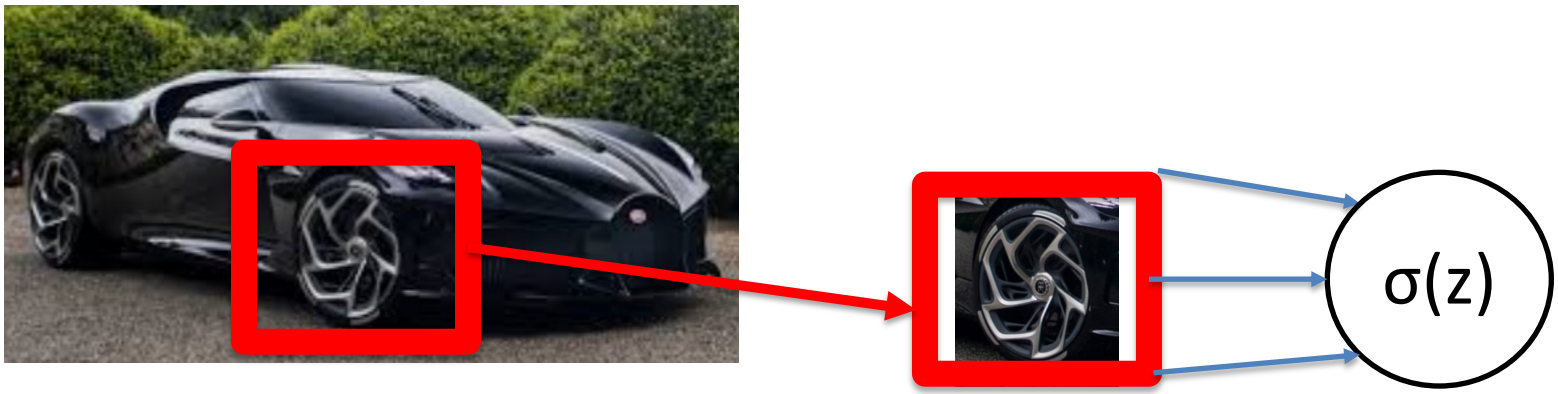
Avec un MLP

- Approche « sac de mots » : pas de corrélation entre les features



Apprendre une image ?

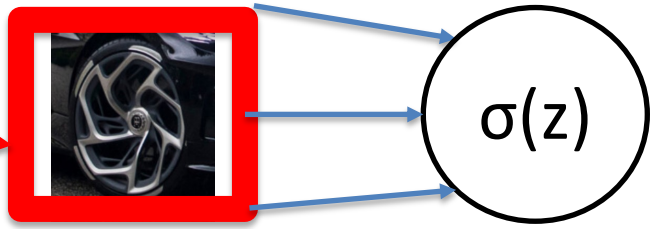
- Des « motifs/features» peuvent servir à déterminer une classe

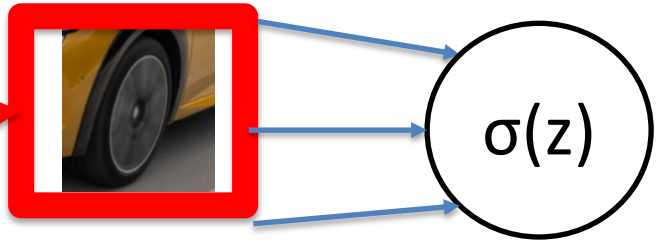


Apprendre une image ?

- Des « motifs» existent à différents endroits !
- En fait c'est la même chose : des filtres !

Roue droite



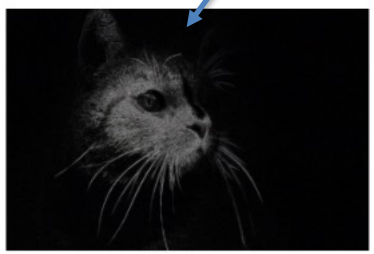


Roue gauche

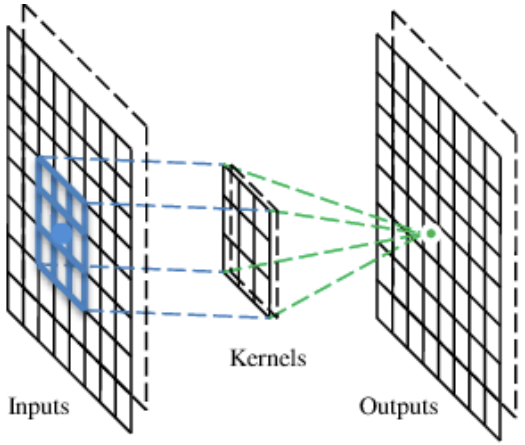
Une couche de convolution

- La couche de convolution utilise des filtres
- Filtre (Kernel) : matrice qui sert à faire la convolution
- Un filtre est une matrice que l'on applique sur l'image :

Sharpen



Fushia



Exemple d'application de filtres sur Lena

Original | Sharpen | Edge Detect | "Strong" Edge Detect

- Lena est l'une des photos les plus utilisée depuis 1973 en traitement d'image.
- La photo originale vient de la page centrale du numéro de novembre 1972 de Playboy

Une démonstration de filtres : https://setosa.io/ev/image-kernels/

Principe d'une convolution

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

Image : matrice 6 x 6

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

Filtre 1

Principe d'une convolution

$$(I * F)(i,j) = (F * I)(i,j) = \sum_m \sum_n F(m,n) I(i-m, j-n)$$

commutative

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

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

F=Filtre 1

*(I * F) (x, y)*

3			

(**1**x1 + 0x-1 + 0x-1 + 0x-1 + **1**x1
+ 0x-1 + 0x-1 + 0x-1 +**1**x1) = **3**

Principe d'une convolution

Stride = 1

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

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

Filtre 1

3	**-1**		

Principe d'une convolution

Stride = 1

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

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

Filtre 1

3	**-1**	**-3**	

Principe d'une convolution

Stride = 1

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

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

3	**-1**	**-3**	-1
-3			

Principe d'une convolution

Stride = 1

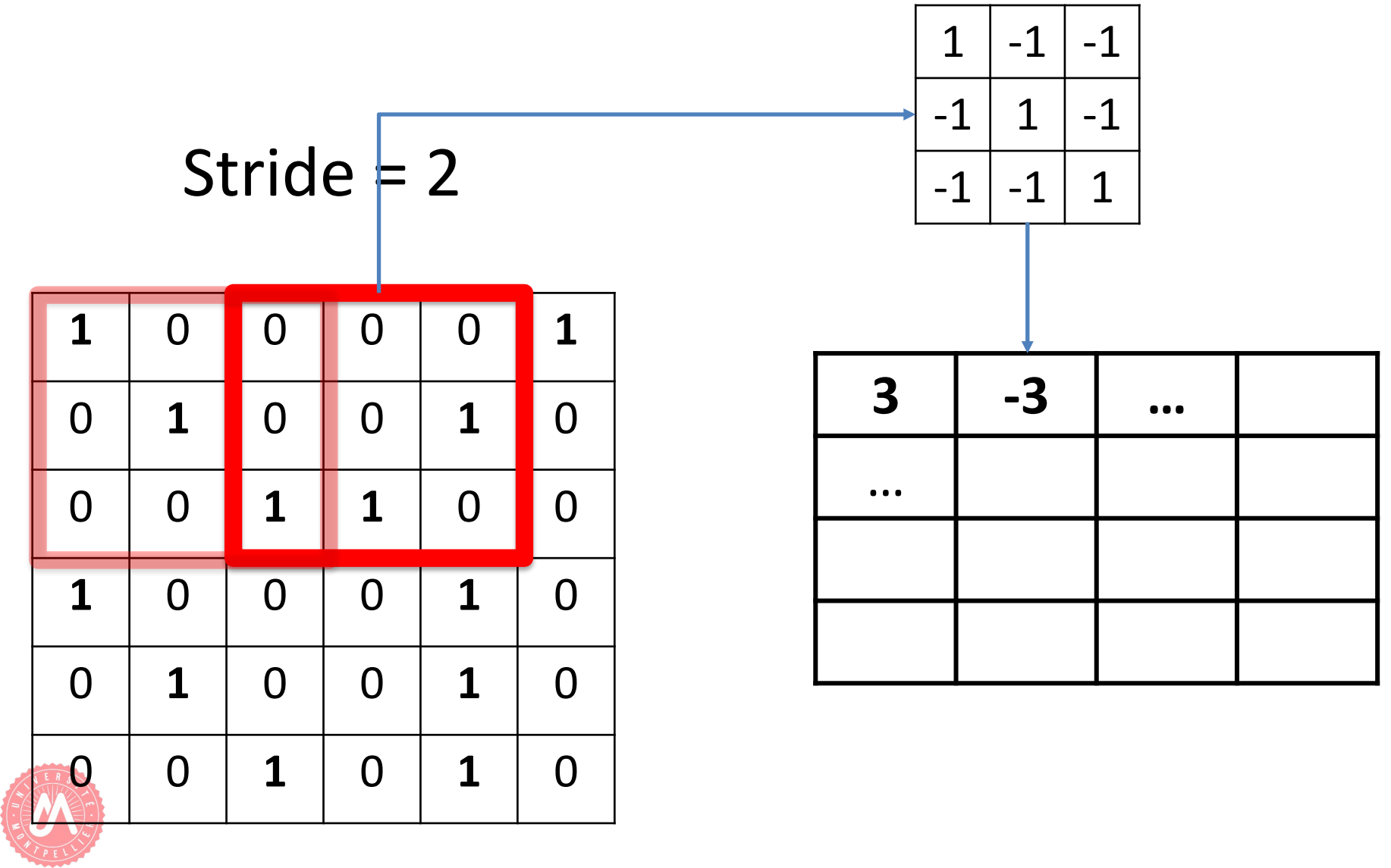
1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

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

Filtre 1

3	**-1**	**-3**	**-1**
-3	**1**	**0**	**-3**
-3	**-3**	**0**	**1**
3	**-2**	**-2**	**-1**

Principe d'une convolution



Plusieurs filtres : répéter l'opération

Stride = 1

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

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

Filtre 1

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

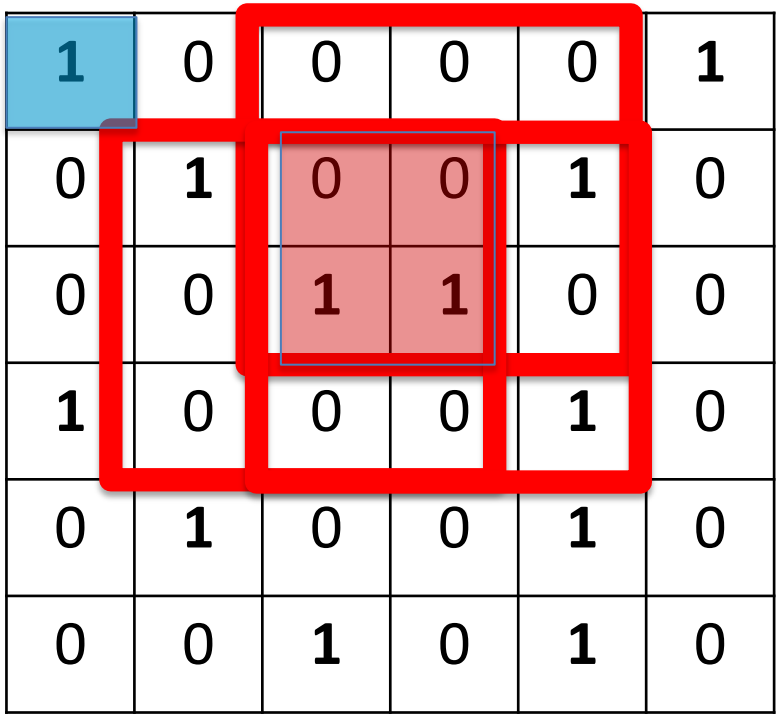
Filtre 2

-1	**-1**	**-1**	**-1**
-1	**..**	**..**	**1**
-1	**..**	**..**	**1**
-1	**0**	**-4**	**3**

2 images (4 x 4) formant une matrice de 2 x 4 x 4

Ici 2 feature maps

Padding



- Le pixel (bleu) dans le coin ne sera vu qu'une seule fois
- Les pixels du milieu (rouge) seront vus plusieurs fois :
  - plus d'informations sur les pixels du milieu !
  - Diminution des sorties
  - Perte d'informations sur les coins de l'image

Padding

- Couche supplémentaire ajoutée à la bordure de l'image

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

Image : matrice 6 x 6

0	0	0	0	0	0	0	0
0	**1**	0	0	0	0	**1**	0
0	0	**1**	0	0	**1**	0	0
0	0	0	**1**	**1**	0	0	0
0	**1**	0	0	0	**1**	0	0
0	0	**1**	0	0	**1**	0	0
0	0	0	**1**	0	**1**	0	0
0	0	0	0	0	0	0	0

Image : matrice 8 x 8

Padding

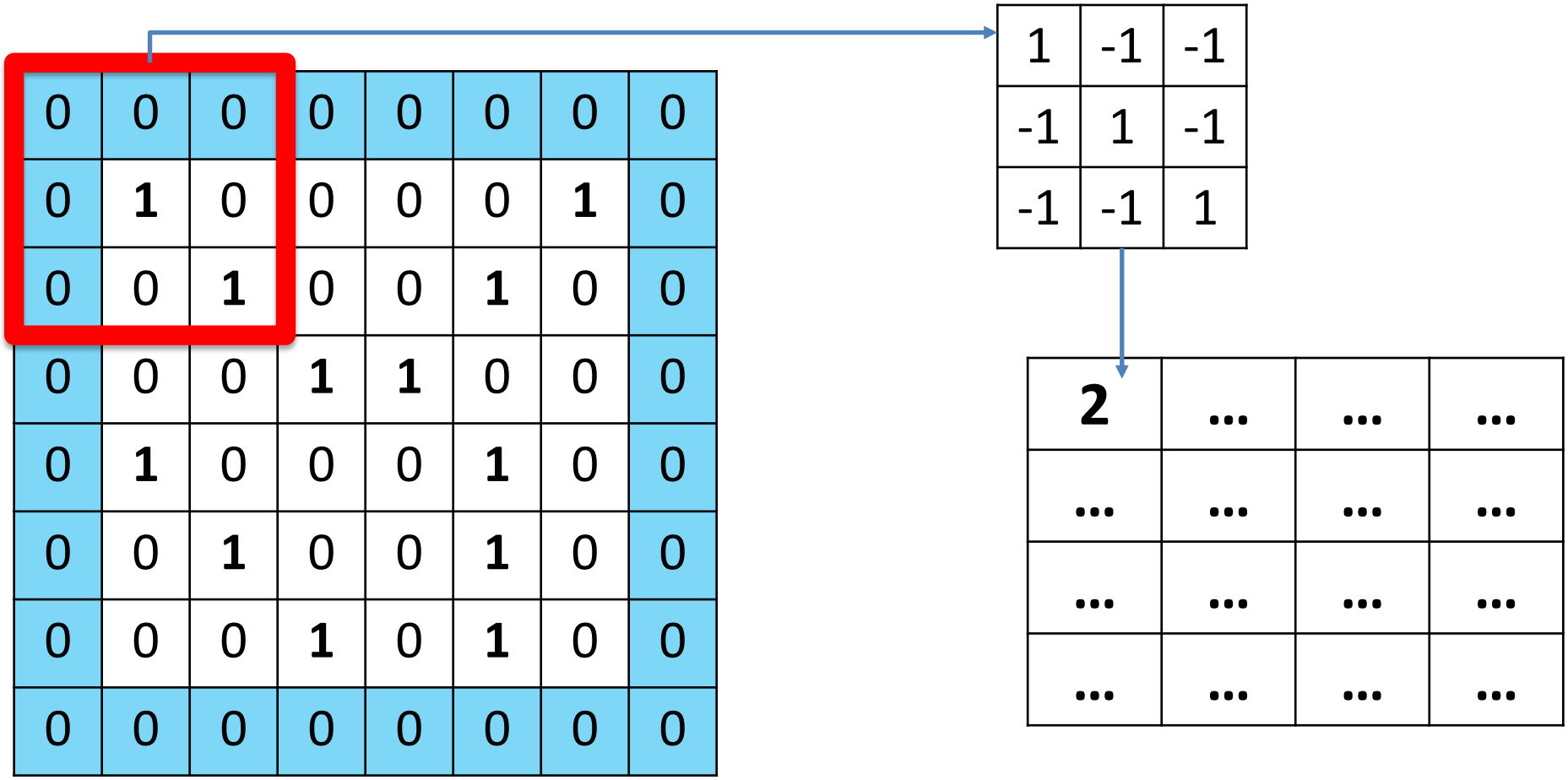
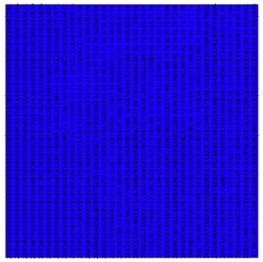
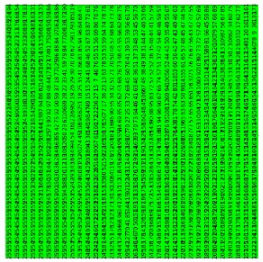
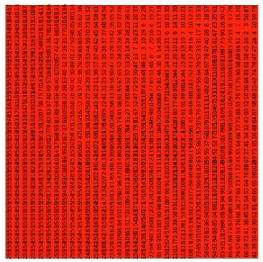

0 0 0 0 0 0 0 0
0 1 0 0 0 1 0
0 0 1 0 0 1 0 0
0 0 0 1 1 0 0 0
0 1 0 0 0 1 0 0
0 0 1 0 0 1 0 0
0 0 0 1 0 1 0 0
0 0 0 0 0 0 0 0
1 -1 -1
-1 1 -1
-1 -1 1
2 ... ...
... ...
... ...
... ...

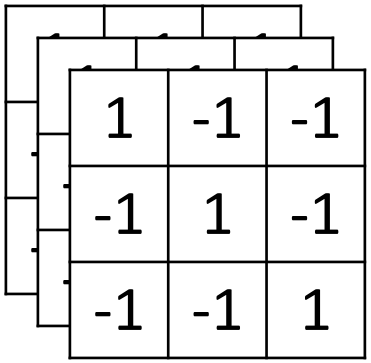
Image couleur : même principe

- 3 canaux :

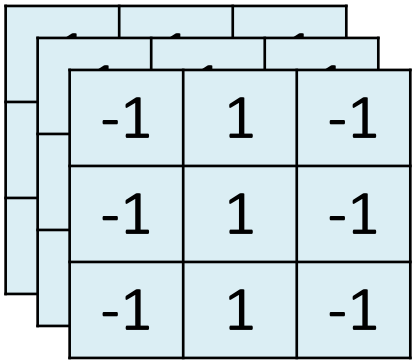




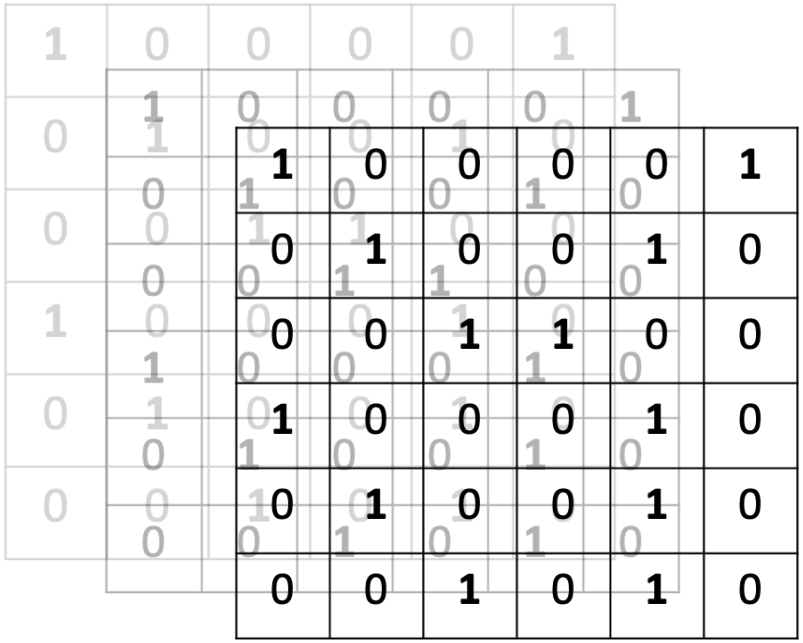




Filtre 1



Filtre 2



Convolution : pour résumer

Chaque filtre permet de détecter un motif.

1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

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

Filtre 1

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

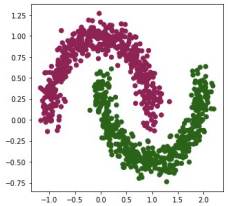
Filtre 2

.

.

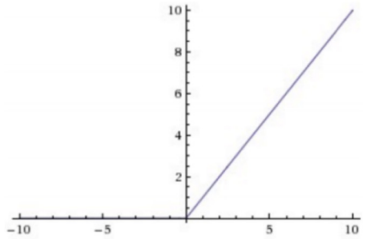
Les filtres sont dérivables, leur valeur correspondent aux paramètres à apprendre

Une couche de fonctions d'activation

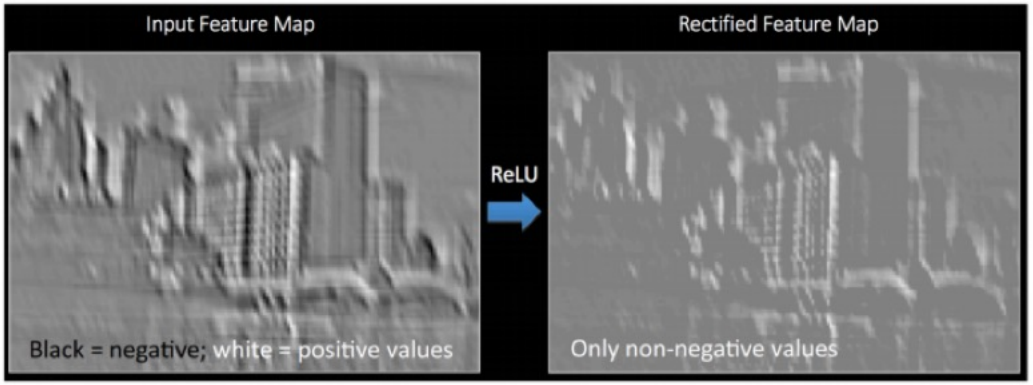


- Introduction de non linéarité
- Techniquement pas une couche car il n'y a pas de paramètres à apprendre
- Généralement : Relu ou Leeky Relu

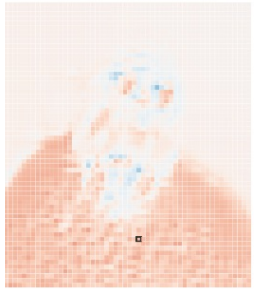
Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

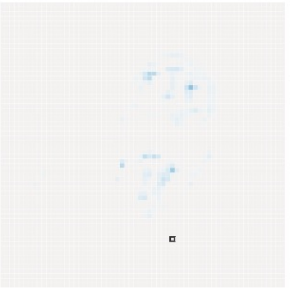


Input (62, 62)



max(0 , −0.7) = 0

Output (62, 62)



Exemple

- Sur la sortie précédente

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Feature map

RELU

3	0	0	0
0	1	0	0
0	0	0	1
3	0	0	0

Activation map

Convolution vs fully connected

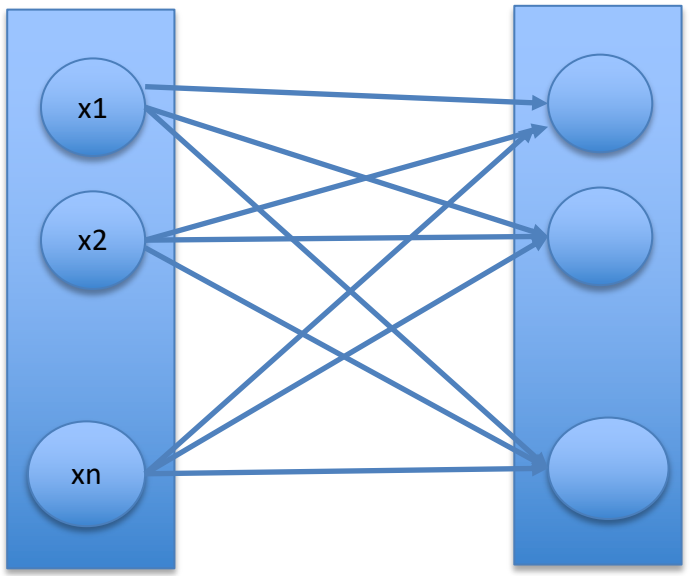
1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0

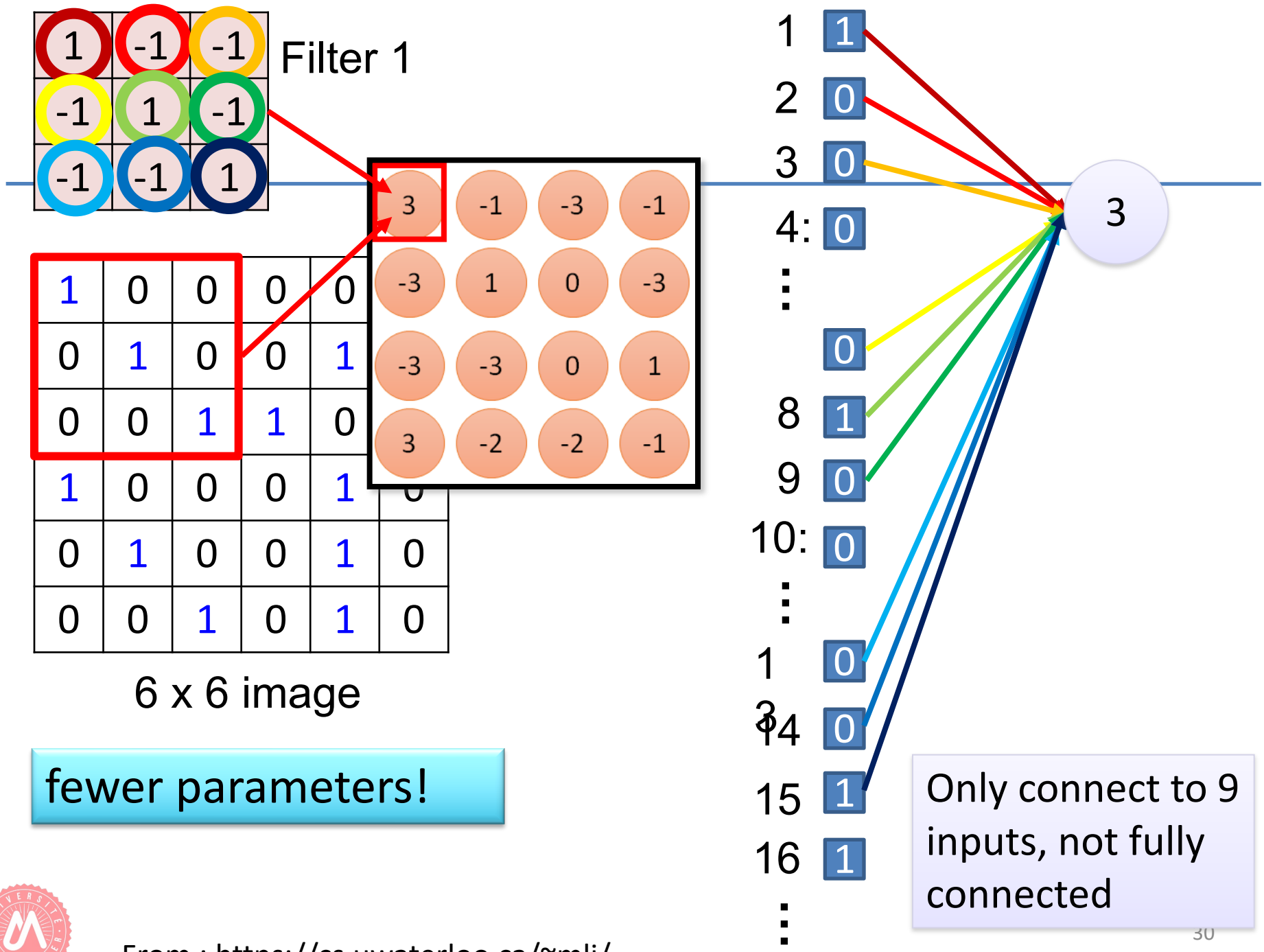
1	-1	-1
-1	1	-1
-1	-1	1

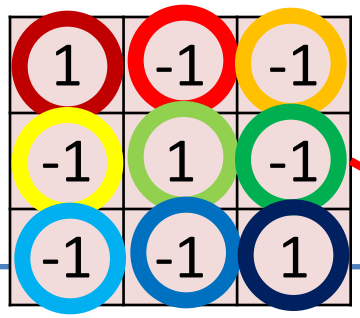
Filtre 1

3	**-1**	**-3**	**-1**
-3	**1**	**0**	**-3**
-3	**-3**	**0**	**1**
3	**-2**	**-2**	**-1**

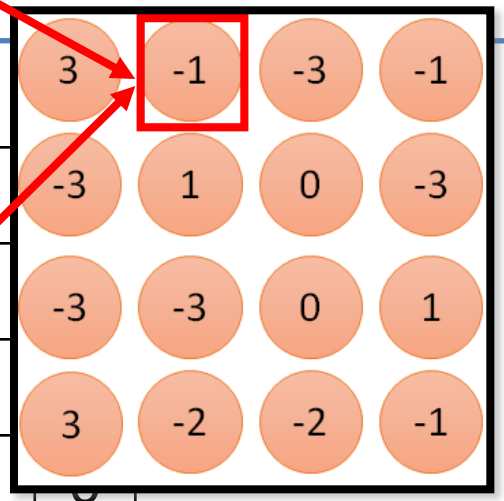
1	0	0	0	0	**1**
0	**1**	0	0	**1**	0
0	0	**1**	**1**	0	0
1	0	0	0	**1**	0
0	**1**	0	0	**1**	0
0	0	**1**	0	**1**	0




1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
1
2
3
4:
8
9
10:
1
3
14
15
16
0
0
0
1
0
0
0
1
1
1
3
Only connect to 9 inputs, not fully connected
fewer parameters!
From : https://cs.uwaterloo.ca/~mli/

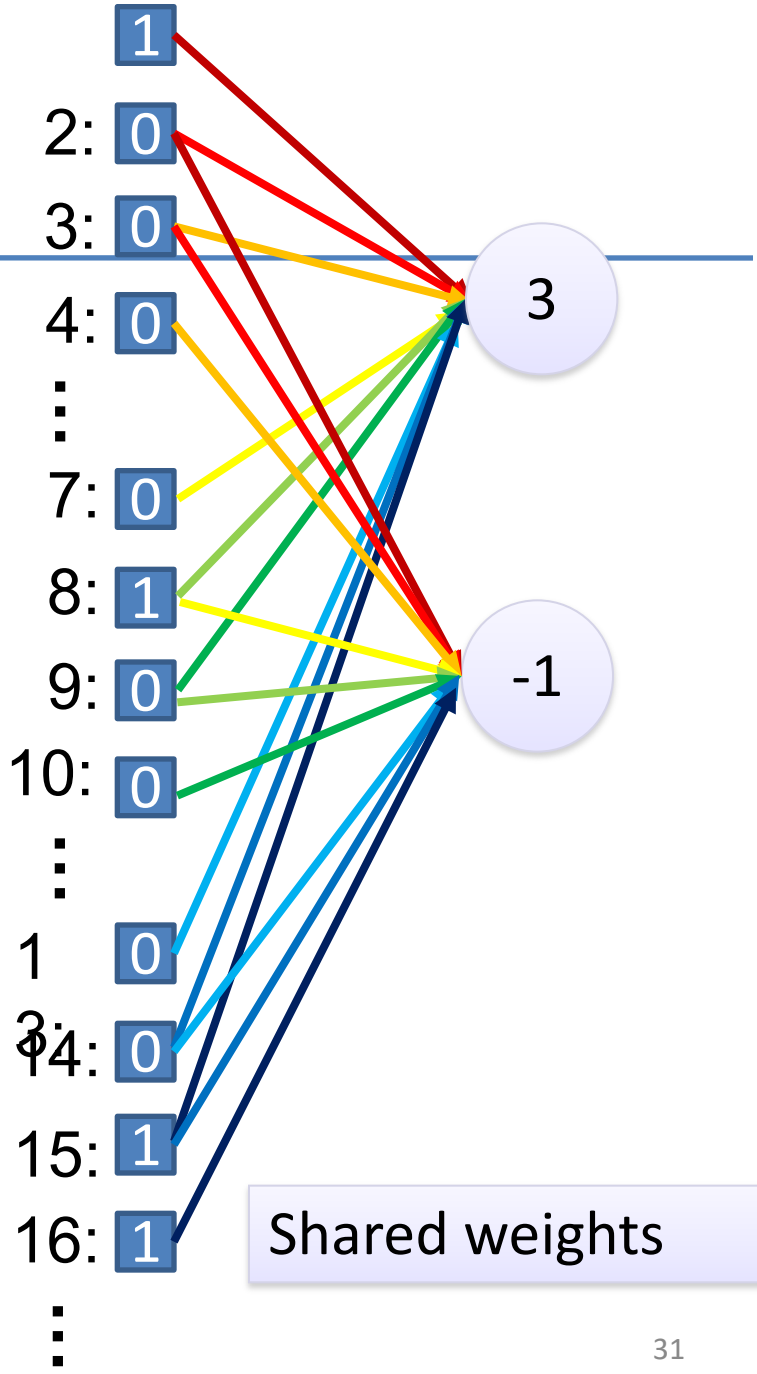


Filter 1



1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



Fewer parameters

Even fewer parameters

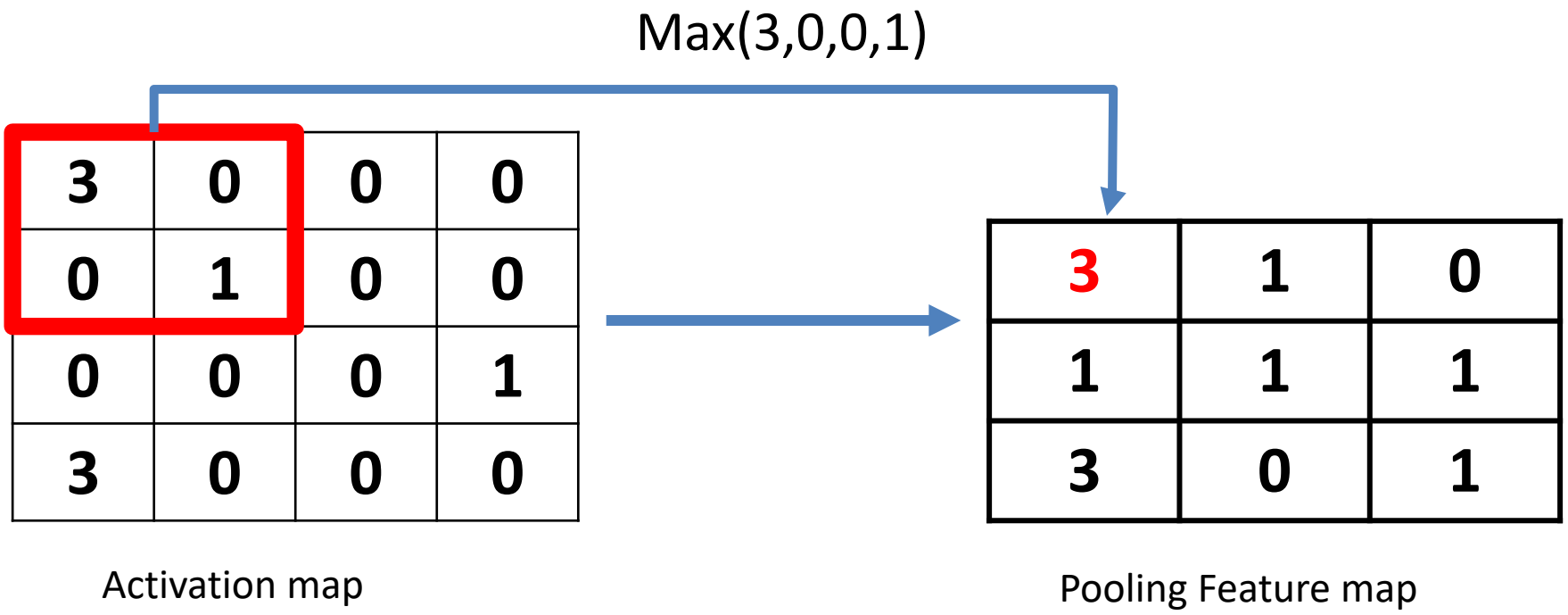
Shared weights

From : https://cs.uwaterloo.ca/~mli/

Réduire la taille : pooling

- Objectifs :
  - Obtenir un représentation plus petite (empêche le surapprentissage en éliminant les données inutiles et gain en vitesse de calcul)
  - Réduire encore la taille de l'image !
  - Voir dilatation …

Réduire la taille : Pooling

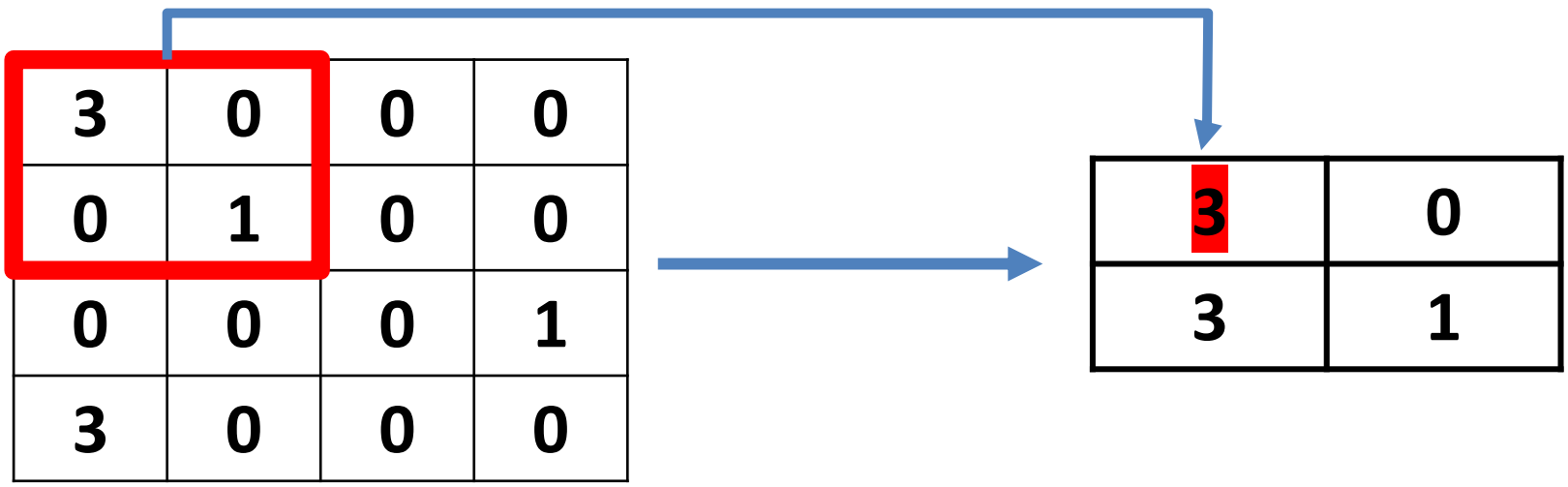


Max pooling avec un filtre 2x2 et un stride de 1

Réduire la taille : pooling

Encore plus petit : 75% de réduction

Max(3,0,0,1)



Activation map

Pooling Feature map

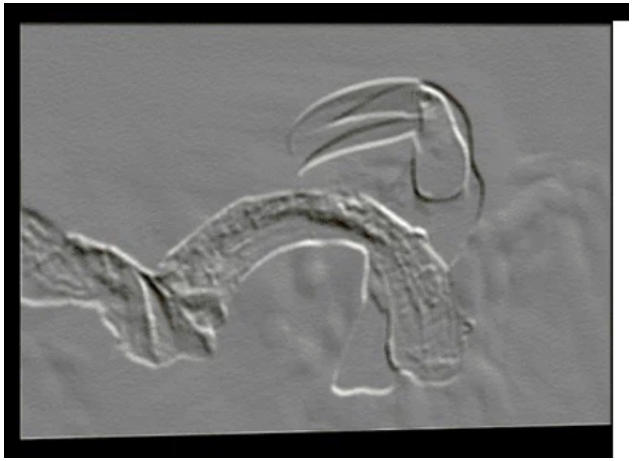
Max pooling avec un filtre 2x2 et un stride de 2

Généralement utilisation d'un stride de 2 pour le pooling.

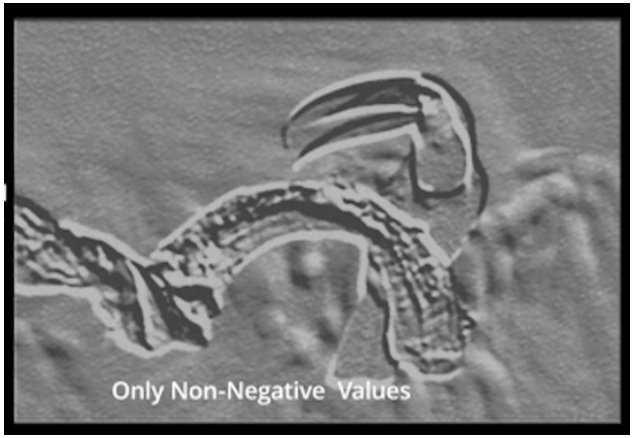
3 pour les images > 200 pixels

Les étapes en résumé

Image originale



Feature map

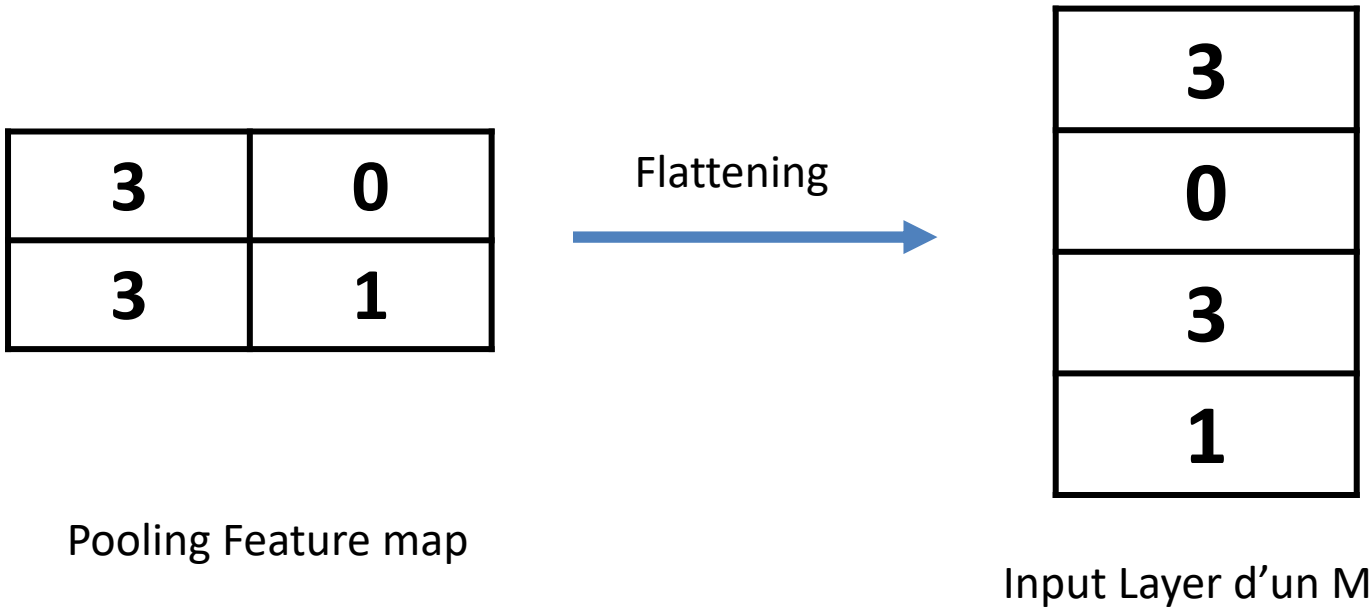


Activation map

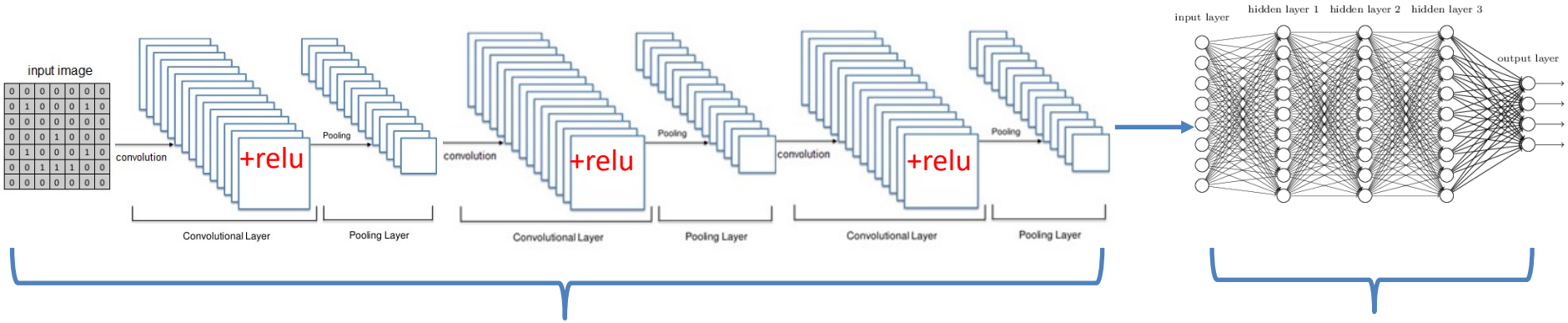
Pooling feature map

Couche Flatten

- Objectif : transformer la matrice de sortie en vecteur pour servir d'entrée à un MLP



Architecture d'un CNN

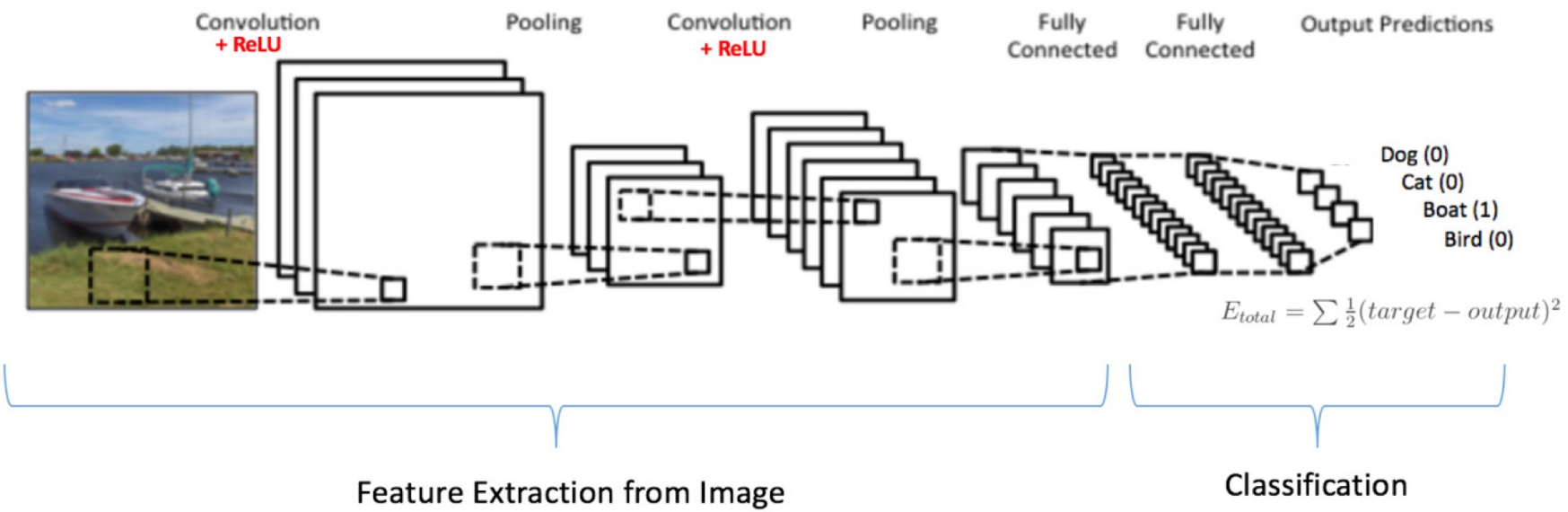


Extraction des features de l'image

Classification

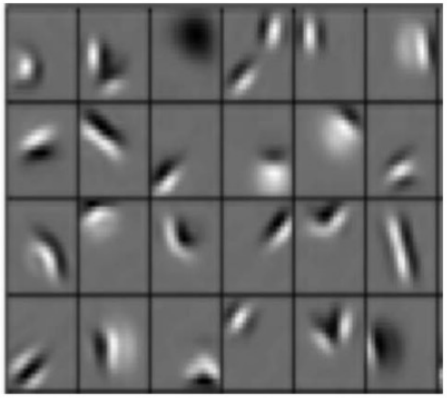
INPUT => CONV => RELU => POOL => CONV => RELU => POOL => CONV => RELU => FC => SOFTMAX

Architecture d'un CNN



Rôle des différentes convolution

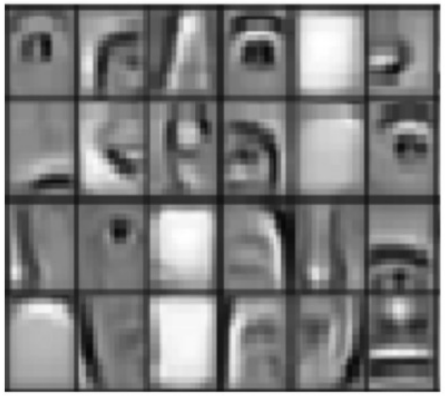
Low level features



Edges, dark spots

Conv Layer 1

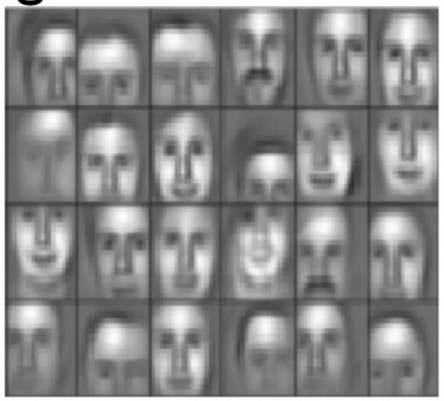
Mid level features



Eyes, ears,nose

Conv Layer 2

High level features

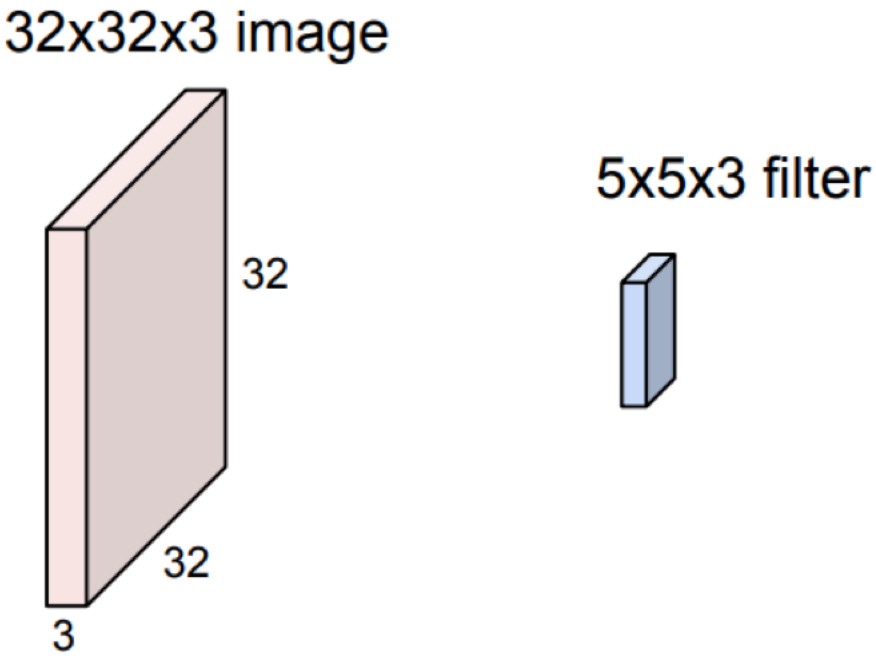


Facial structure

Conv Layer 3

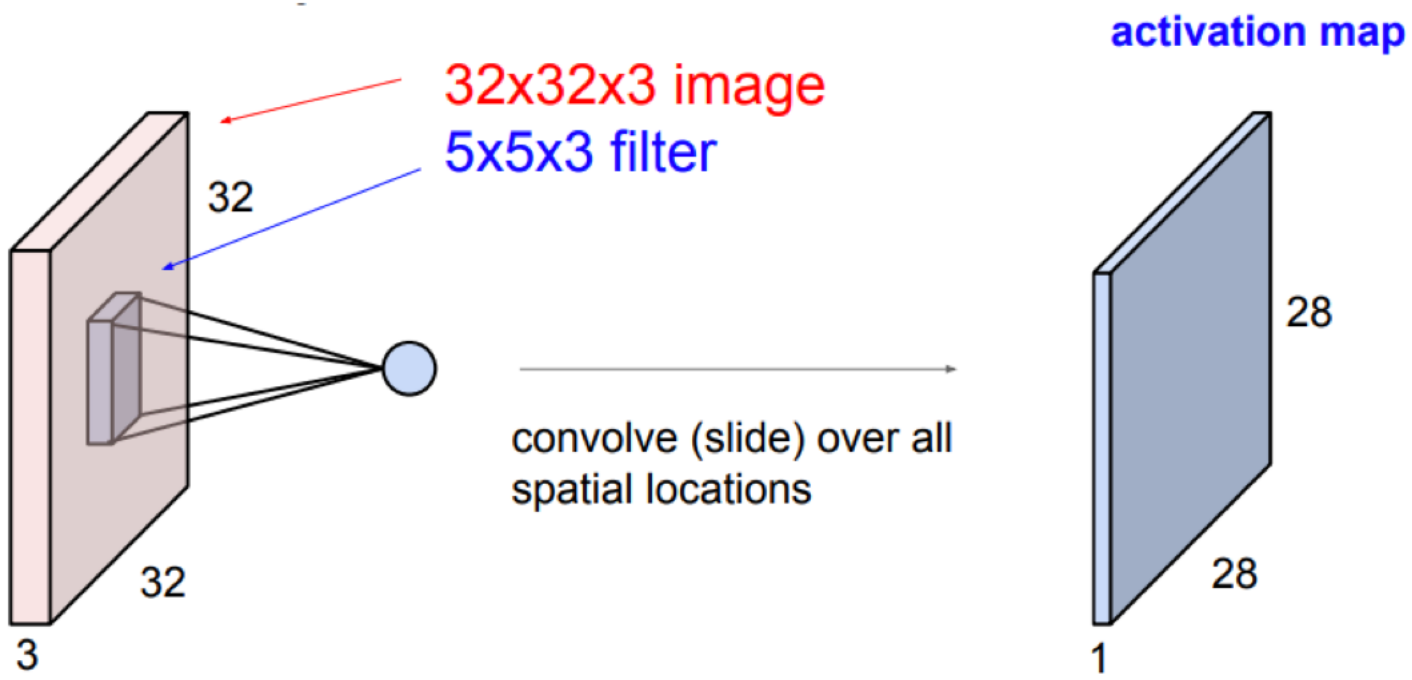
Source: Lee+ ICML2009

Taille des layers



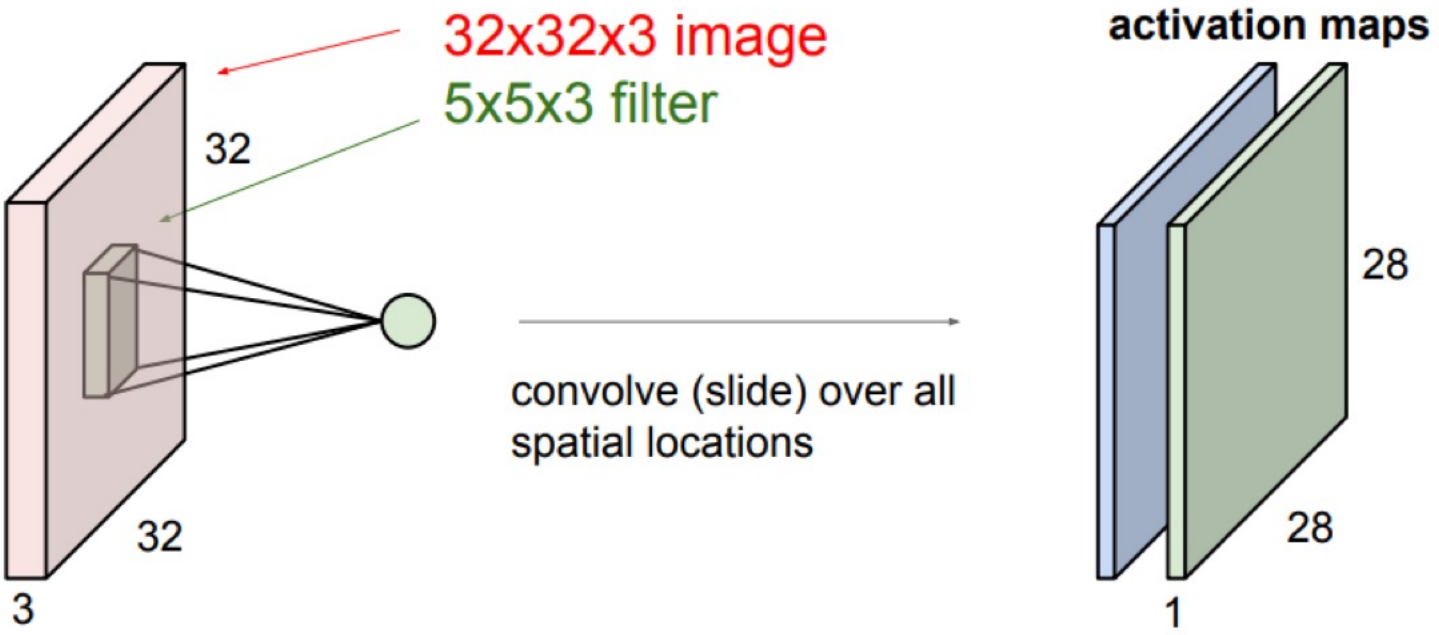
Faire une convolution du filtre sur l'image

Taille des layers



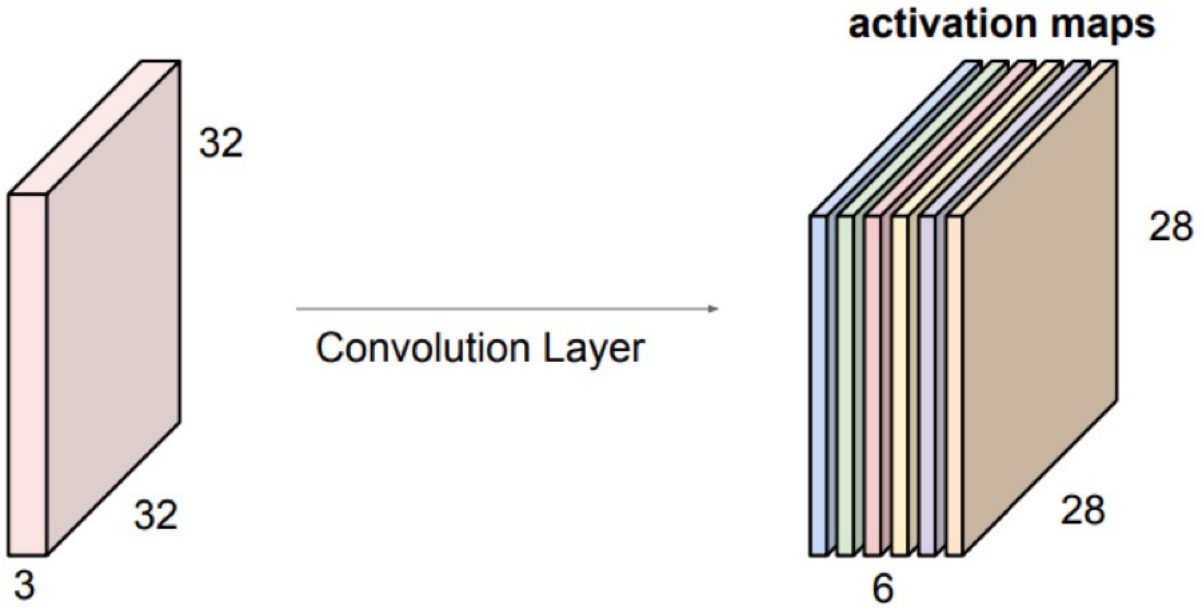
Activation map pour le filtre 1

Taille des layers



Activation map pour le filtre 2

Taille des layers



Activation map pour ….. 6 filres

Les activations maps sont empilées pour créer une nouvelle image de taille 28x28x6

Taille de sortie de la convolution

Avec une image de :

W1 x H1 x D1 (D1 = nombre de canaux),

K, le nombre de filtres,

F, la taille du filtre,

S, le stride et P, le nombre de zéro du padding

La sortie de la convolution est :

$W2 = (W1-F+2xP)/S +1$

$H2 = (H1 – F + 2xP)/S +1$

D2 : K

Pour une image 255x255 avec 64 canaux, avec 40 filtres de taille 5x5, sans padding et stride=1 (défaut) :

$W2 = (128 - 5 + 2x0)/1 +1 = 124$

$H2 = (128 – 5 + 2x0)/1 +1 = 124$

D2 = 40

Outputshape (124, 124,40)

Filtre, stride et padding : carrés ?

- Un filtre (resp. padding) n'est pas nécessairement une matrice carrée … pour obtenir la taille il faut redéfinir :

  1 image = W1 x H1 x D1

  Nombre de filtres : K

  Taille du filtre : KW, KH

  Taille du stride : SW, SH

  Taille du padding :

  Padding hauteur haut (P_H1)

  Padding hauteur bas (P_H2)

  Padding largeur à gauche (P_W1)

  Padding margeur à droite (P_W2)

Filtre, stride et padding : carrés ?

$$W2 = (W1+P\_W1+PW2-KW)/SW +1$$

$$H2 = (H1 + P\_H1 +PH2 – KH)/SH +1$$

$$D2 = K$$

- Pour une image couleur de 255x255 avec 30 filtres de taille 7, 128, un stride de 3x4 et et un padding de 2 (e.g. [2, 2, 2,2] :

  W2 =(255 + 2 + 2 - 7) / 4 + 1 = 64

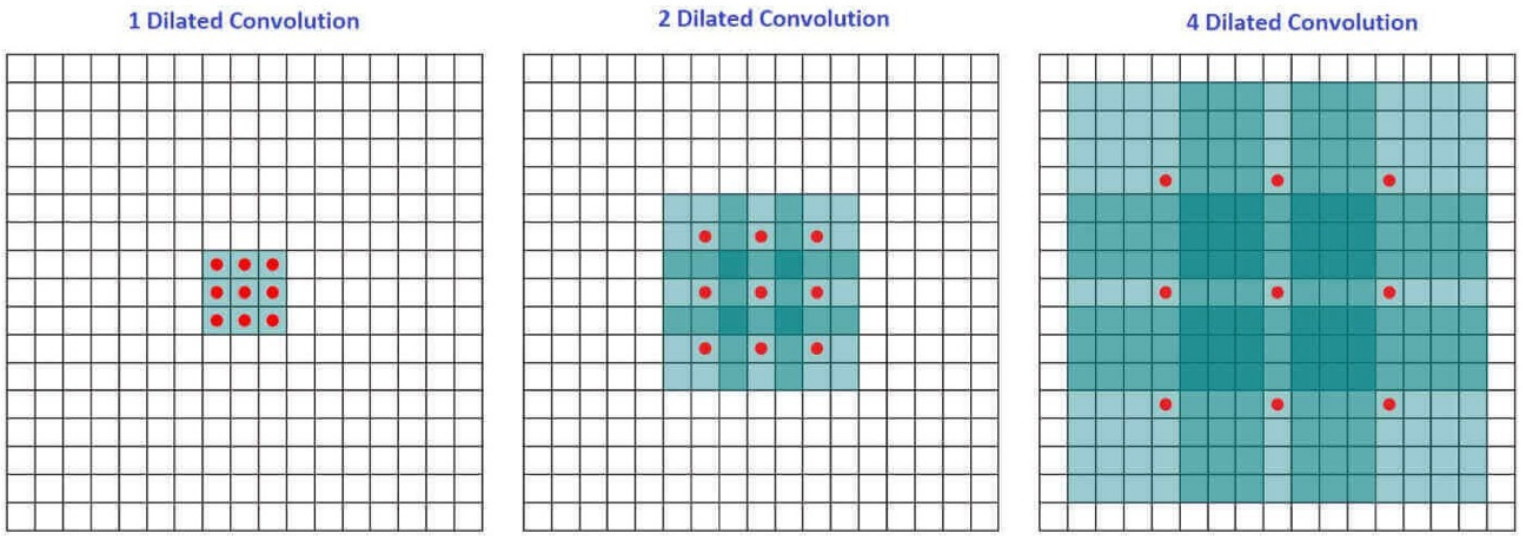
  H2 = (255 + 2 + 2 - 7) / 3 + 1 = 85

  D2 = 30

Outputshape (85, 64, 30)

Dilatation

- La dilatation permet de sauter certains points lors de la convolution
- Utilisation d'un paramètre (l = dilatation rate) qui permet de dire combien on saute de pixels



Dilatation

- Elle permet d’étendre la zone de l'image d'entrée couverte sans pooling.
- Avoir un champ de vision plus large au moindre coût
- Pas de perte de résolution de l’image de sortie : il s’agit d’une extension

Un mot sur la backpropagation

$$\begin{bmatrix} O_{11} & O_{12} \\ O_{21} & O_{22} \end{bmatrix} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{bmatrix} \right)$$

$$O_{11} = F_{11}X_{11} + F_{12}X_{12} + F_{21}X_{21} + F_{22}X_{22}$$

$$O_{12} = F_{11}X_{12} + F_{12}X_{13} + F_{21}X_{22} + F_{22}X_{23}$$

$$O_{21} = F_{11}X_{21} + F_{12}X_{22} + F_{21}X_{31} + F_{22}X_{32}$$

$$O_{22} = F_{11}X_{22} + F_{12}X_{23} + F_{21}X_{32} + F_{22}X_{33}$$

$$\begin{bmatrix} \partial E/\partial F_{11} & \partial E/\partial F_{12} \\ \partial E/\partial F_{21} & \partial E/\partial F_{22} \end{bmatrix} = \text{Convolution}\left(\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \end{bmatrix}, \begin{bmatrix} \partial E/\partial O_{11} & \partial E/\partial O_{12} \\ \partial E/\partial O_{21} & \partial E/\partial O_{22} \end{bmatrix} \right)$$

https://medium.com/@2017csm1006/forward-and-backpropagation-in-convolutional-neural-network-4dfa96d7b37e

En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Partie Convolution et pooling – Extraction des feature maps

Conversion en vecteur pour entrer dans la partie MLP (input layer)

partie MLP classique pour La classification

En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

S'il n'y a pas d'ambiguïté le nom des variables peut être enlevé : filters, kernel_size et pool_size sont optionnels

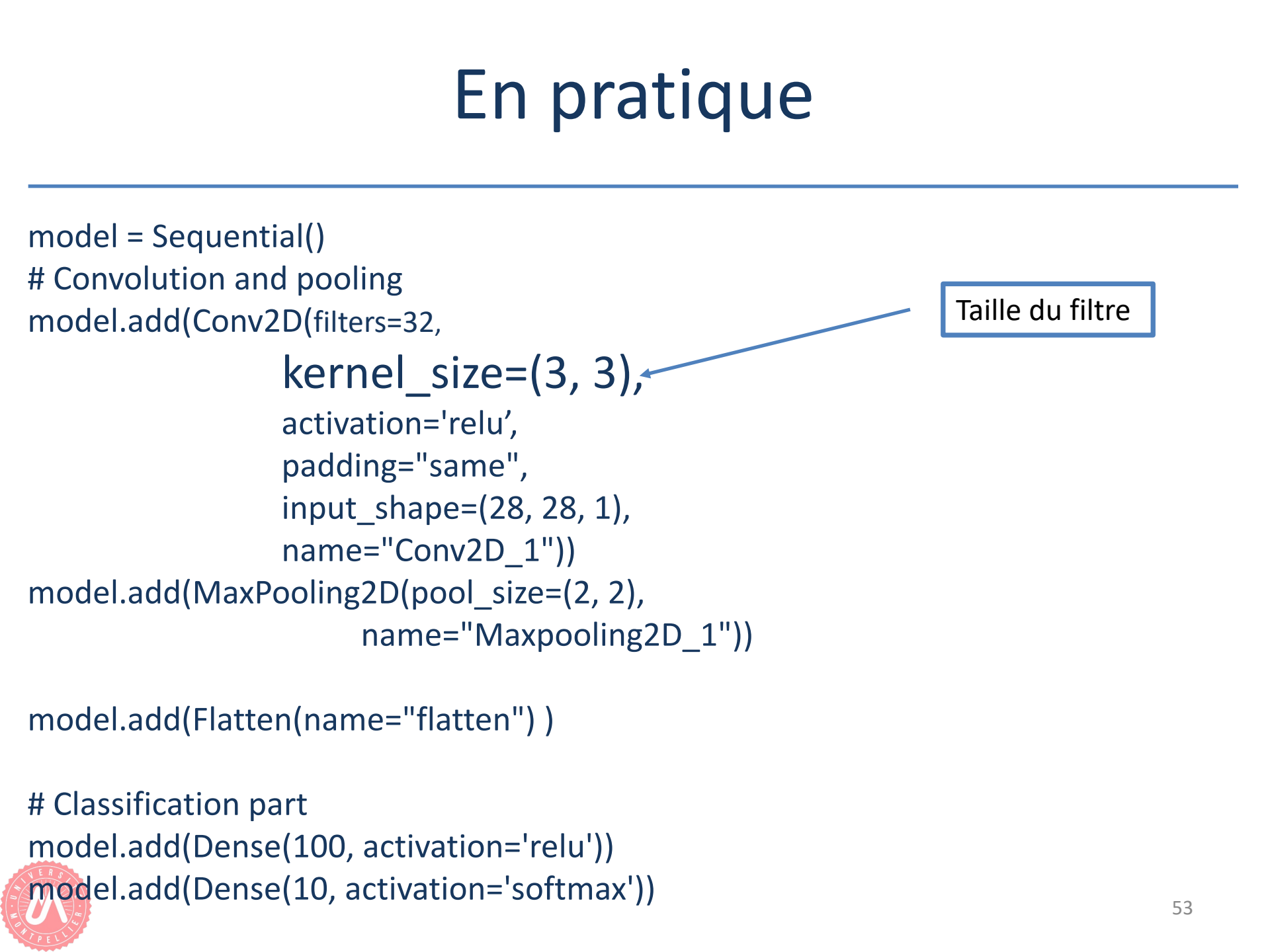
En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Nombre de filtres.
Augmentation du nombre en fonction des niveaux car le motif devient plus complexe. Souvent des puissances de 2. (16, 32, 64, …)

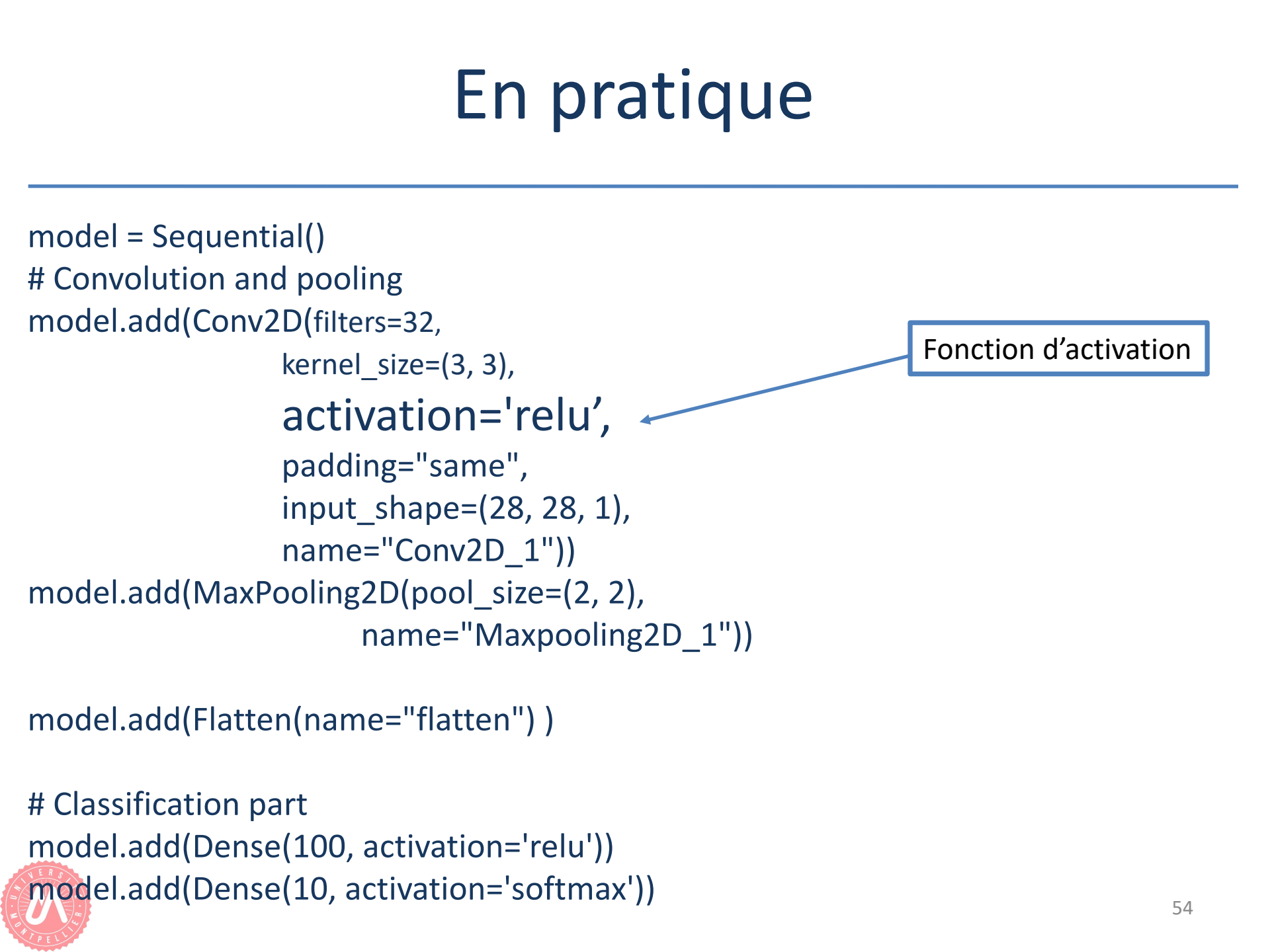
En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```



En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```



En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Keras offre deux possibilités : "*valid*" ou "*same* ».

"*valid*" signifie pas de remplissage, i.e. c'est la valeur par défaut.
"*same*" met des zéros uniformément à gauche/droite ou haut/bas de l'entrée.

Lorsque padding="same" et strides=1, la sortie a la même taille que l'entrée.

En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Taille de l'image d'entrée : Ici 28 x 28 et il n'y a qu'un canal (1). C'est du noir et blanc. Pour couleur RGB, la valeur est 3.

En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Pour nommer le layer.
pas du tout obligatoire
mais peut être pratique
si l'on veut accéder à une
couche par son nom plutôt
que son numéro

En pratique

```
model = Sequential()
# Convolution and pooling
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                 activation='relu',
                 padding="same",
                 input_shape=(28, 28, 1),
                 name="Conv2D_1"))
model.add(MaxPooling2D(pool_size=(2, 2),
                       name="Maxpooling2D_1"))

model.add(Flatten(name="flatten") )

# Classification part
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Taille du filtre pour le pooling
Attention si aucun padding n'est précisé (par défaut), la valeur du padding prend celle de pool_size.

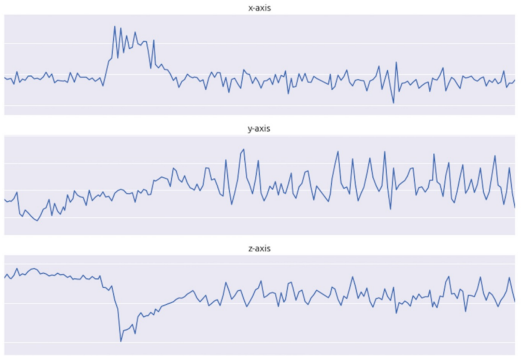
En pratique

```
model.add(Conv2D(filters=32,
                 kernel_size=(3, 3),
                  activation='relu',
                 padding="same",
                 dilation_rate=2,
                  input_shape=(28, 28, 1),
                  name="Conv2D_1"))
```

Taux de dilatation.
Si dilatation = 1, on prend chaque 1er élément (pas de trous).
Si dilatation = 2, on prend tous les 2 (écarts de taille 1). Etc.

Les convolutions 1D

- Séries temporelles, textes




```
from keras.layers import Conv1D

model = Sequential()

model.add(Conv1D(1, kernel_size=5, input_shape = (120, 3)))
```

Le noyau est un vecteur

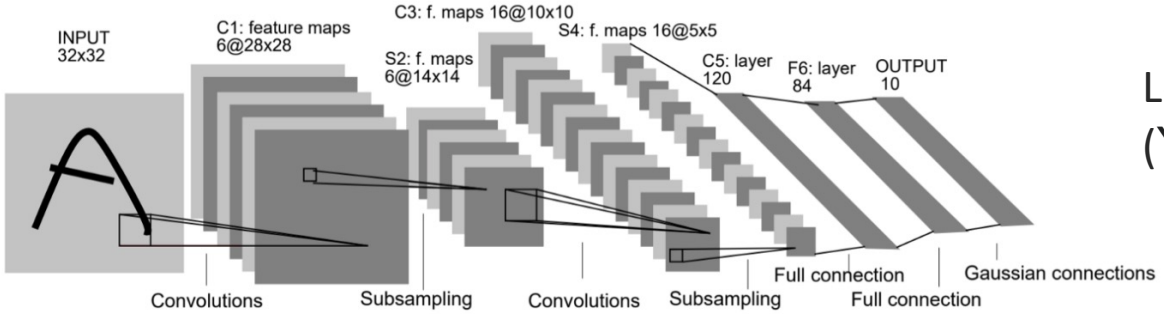
Les convolutions 3D

- Images 3D (IRM), Vidéos, images satellites

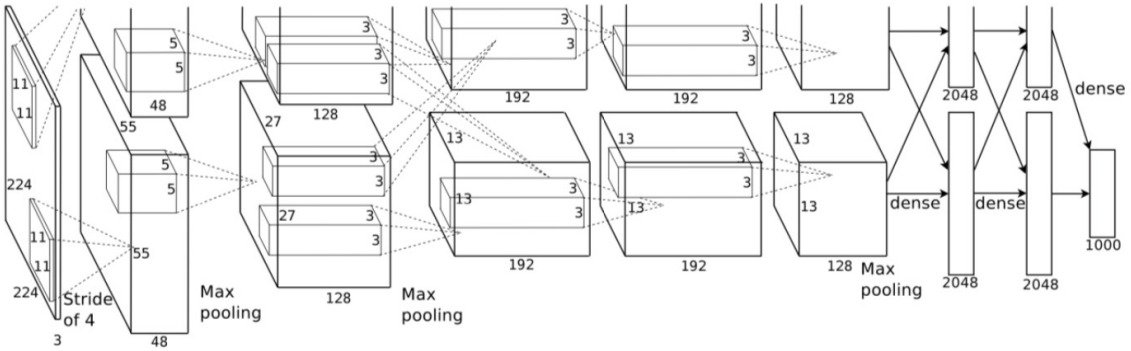
```
from keras.layers import Conv3D

model = Sequential()

model.add(Conv3D(1, kernel_size=(3,3,3), input_shape = (128, 128, 128, 3)))
```

Le noyau est en 3D vecteur

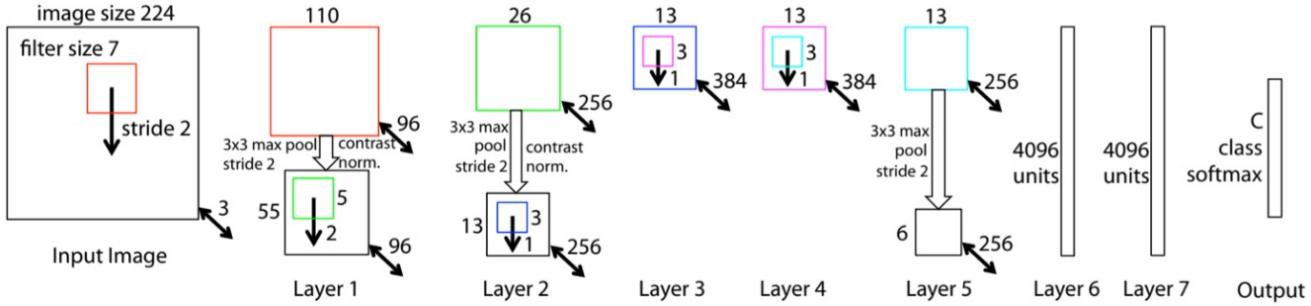
Des architectures en constante évolution



LeNet-5 - 1998
(Yann Le Cun Facebook)

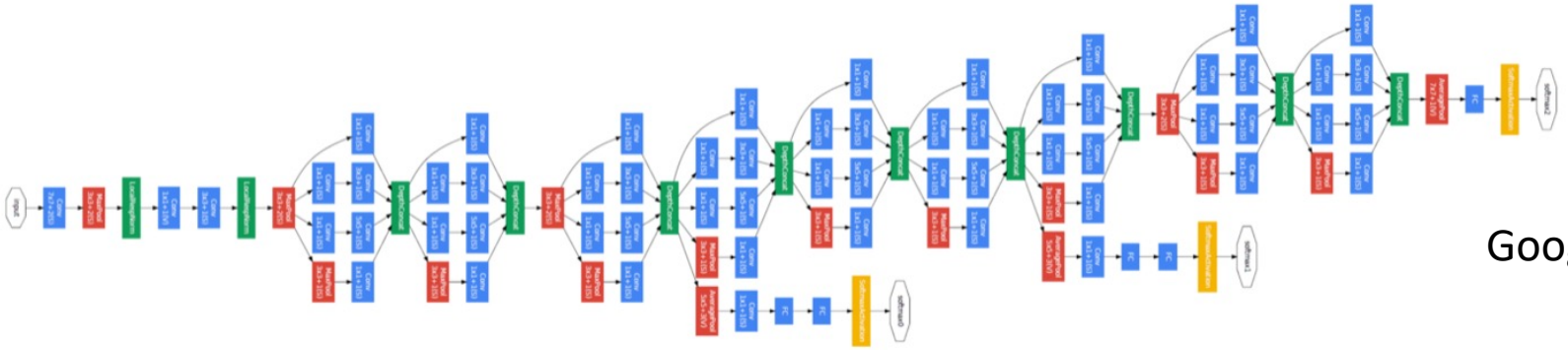


AlexNet - 2012

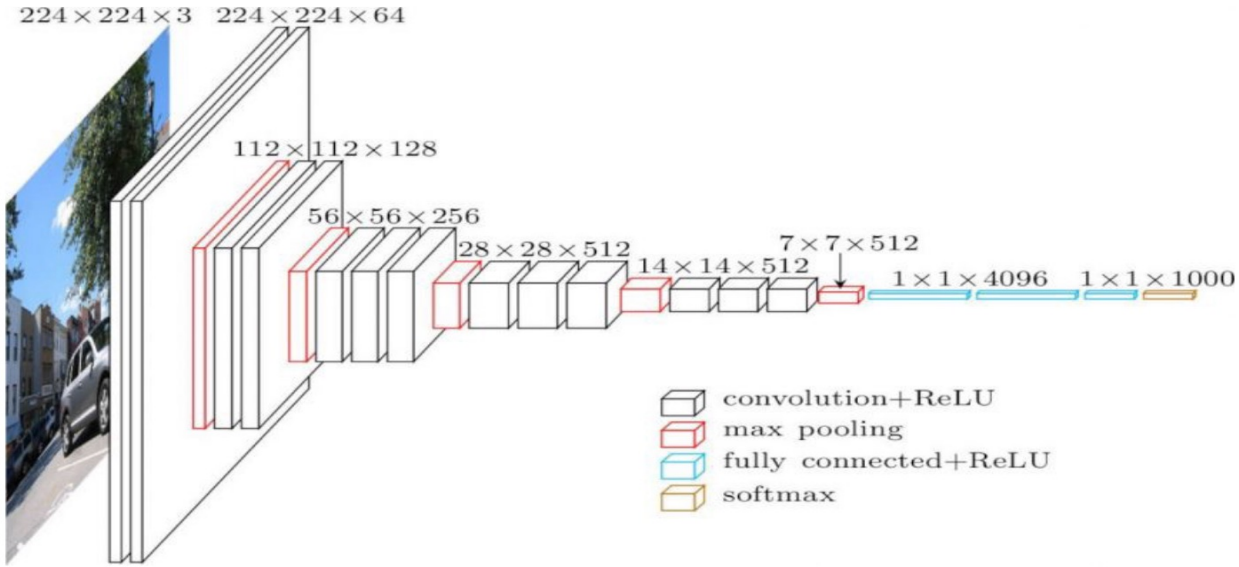


Zfnet - 2013

Des architectures en constante évolution

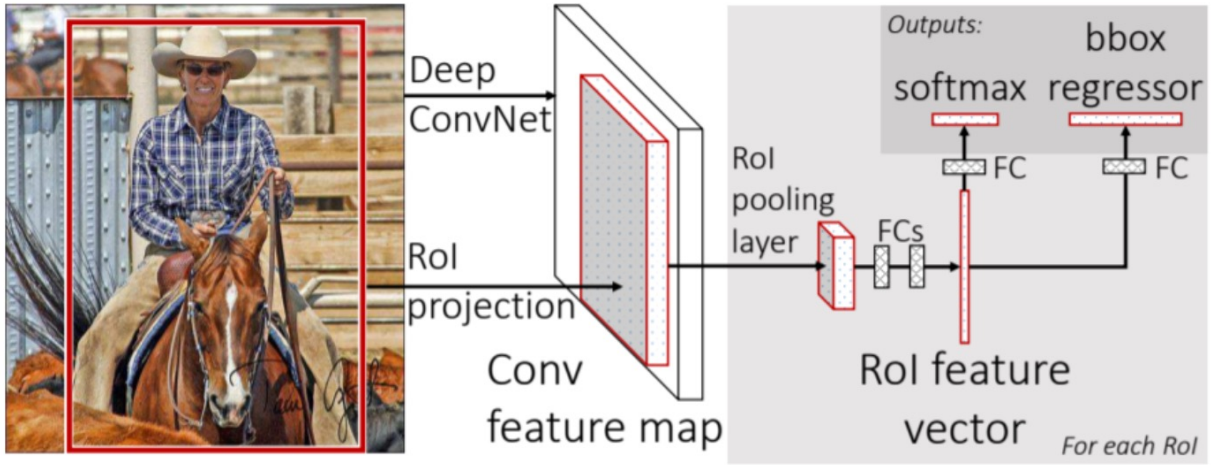


GoogleNet - 2015

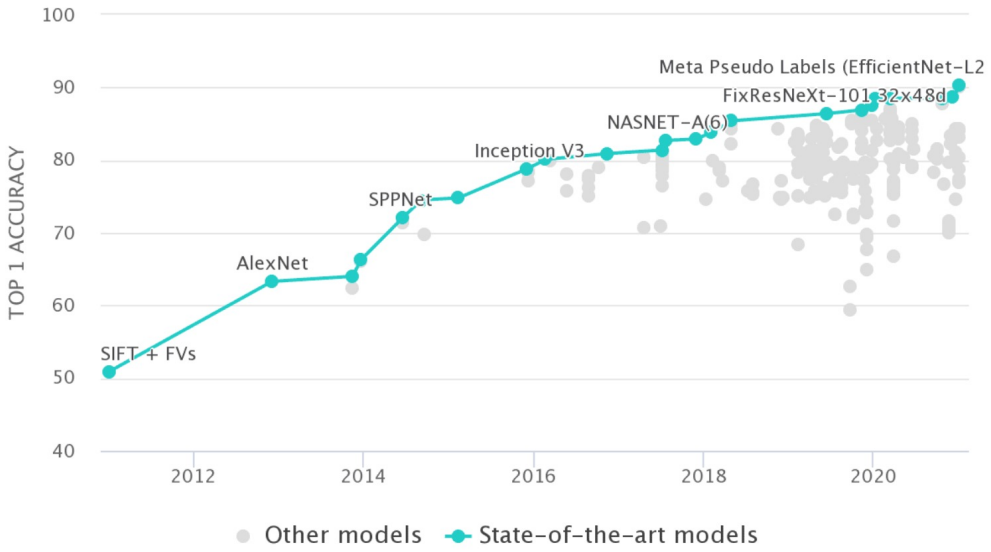


VGG16 - 2015

Des architectures en constante évolution



Fast R-CNN - 2015

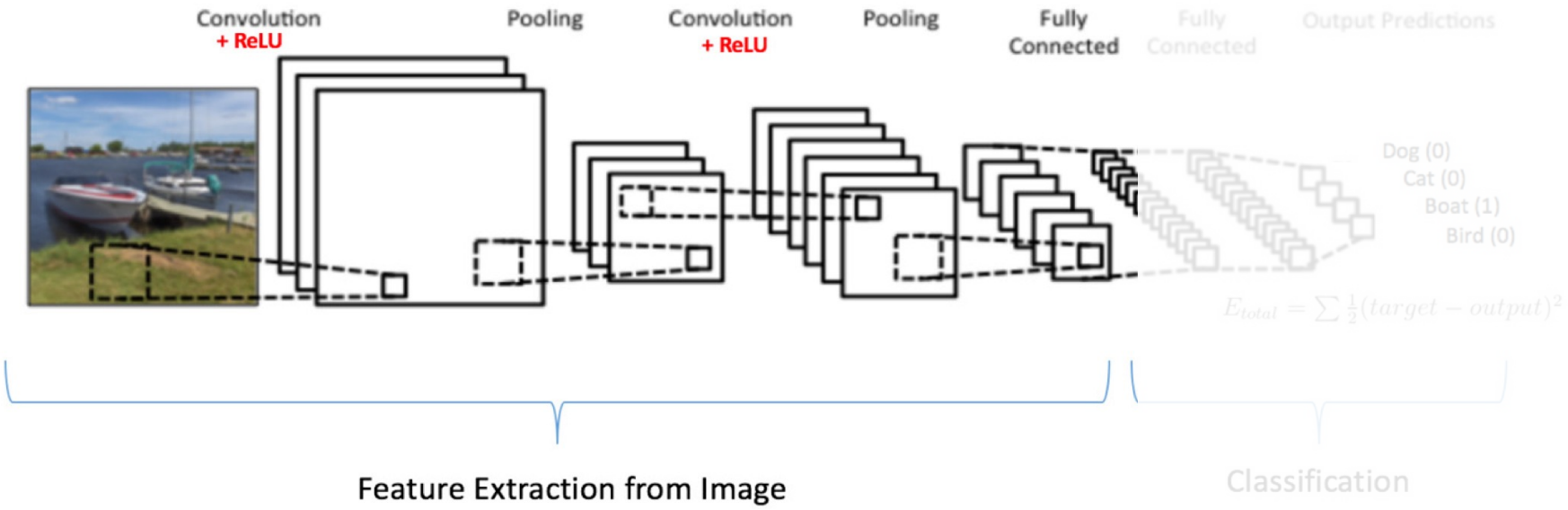


Source : AI summer

Transfer learning

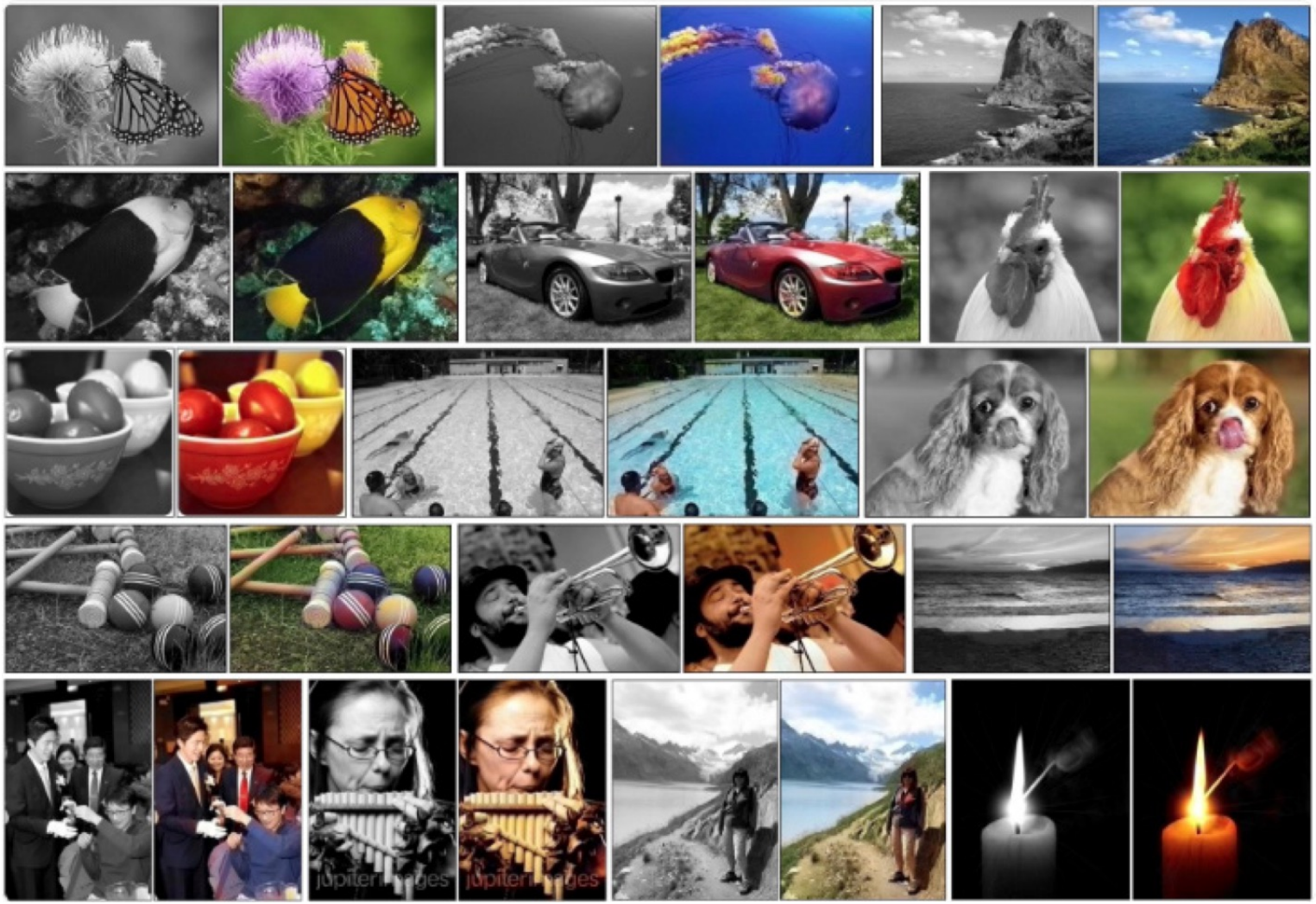
- Profiter des modèles déjà appris sur des milliards de données (le poids/les valeurs des filtres ont été appris) pour améliorer le modèle
  - Il faut faire attention à la taille des images !
  - Il faut rendre les du modèle transféré comme « non apprenable »
  - Il faut enlever partie classification du modèle transféré et la remplacer par sa propre partie (les sorties attendues ne sont généralement pas les mêmes – softmax vs sigmoid – nombre de classes

Une base pour pleins d'applications



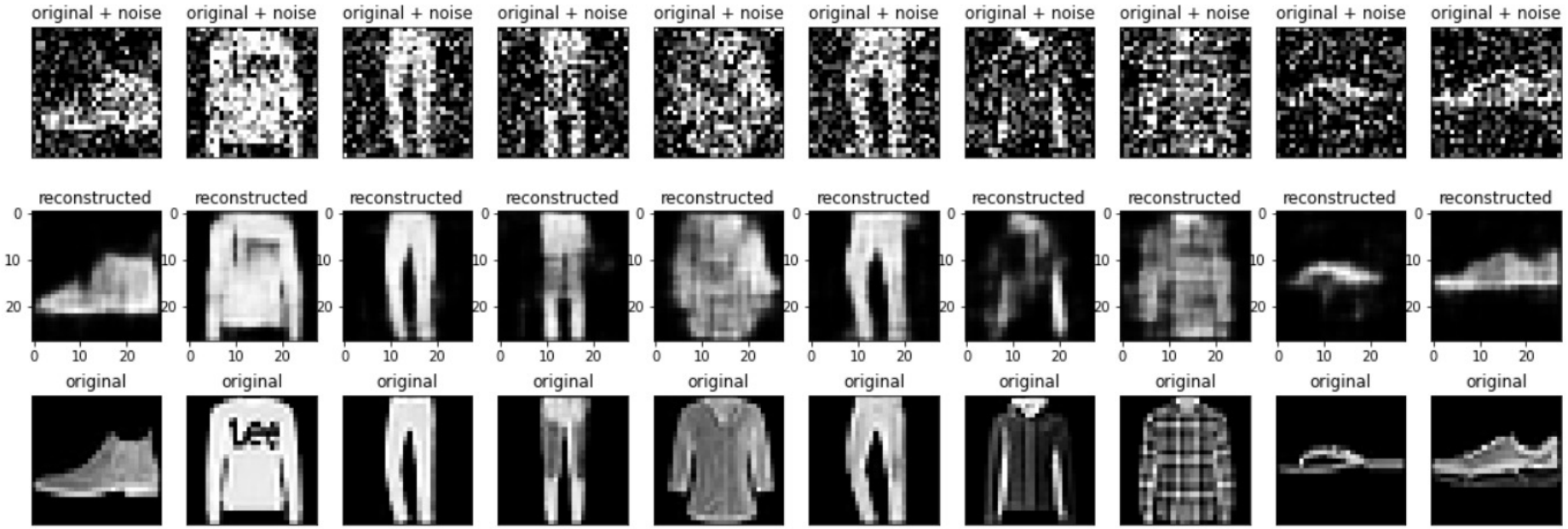
Segmentation
Détection
Correction ….

Colorisation

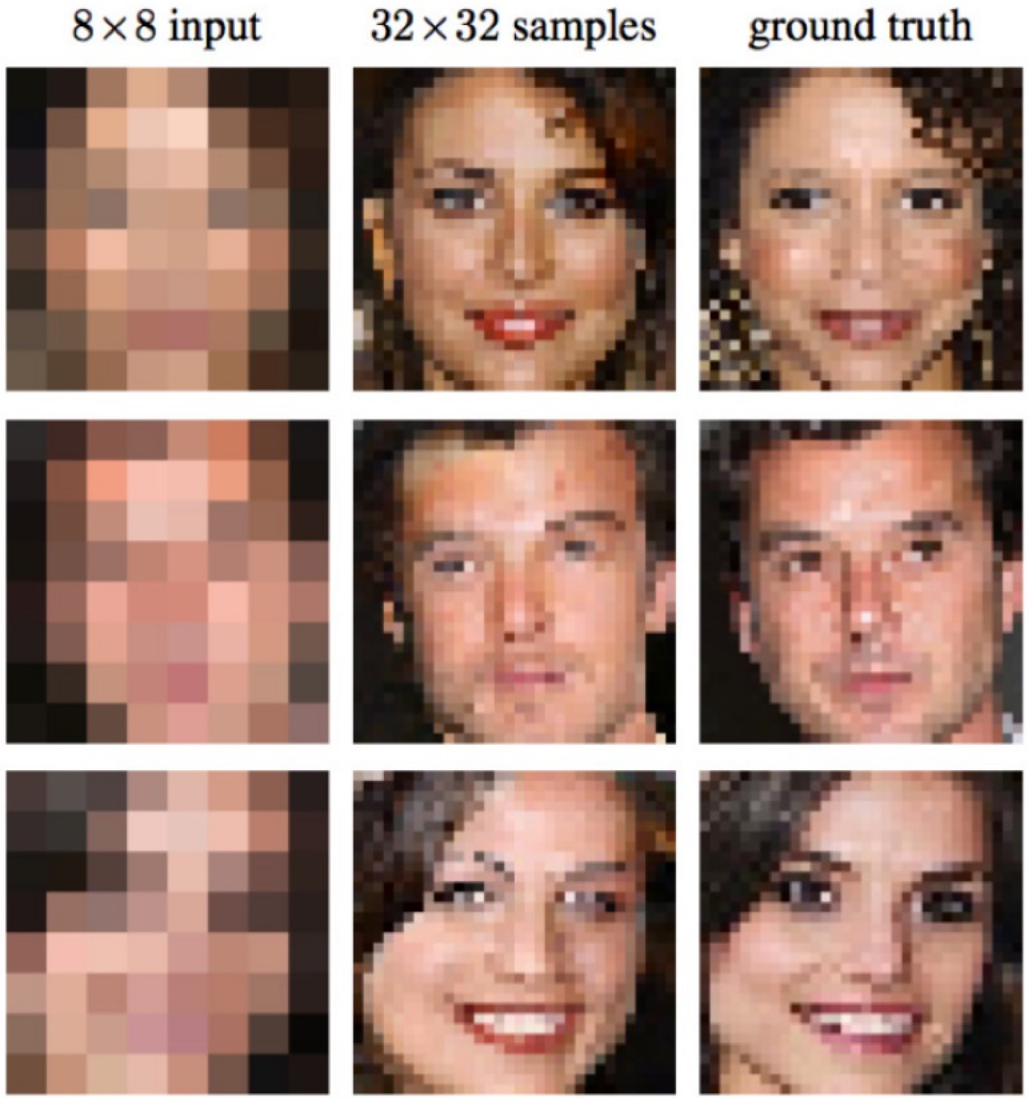


Source : MIT

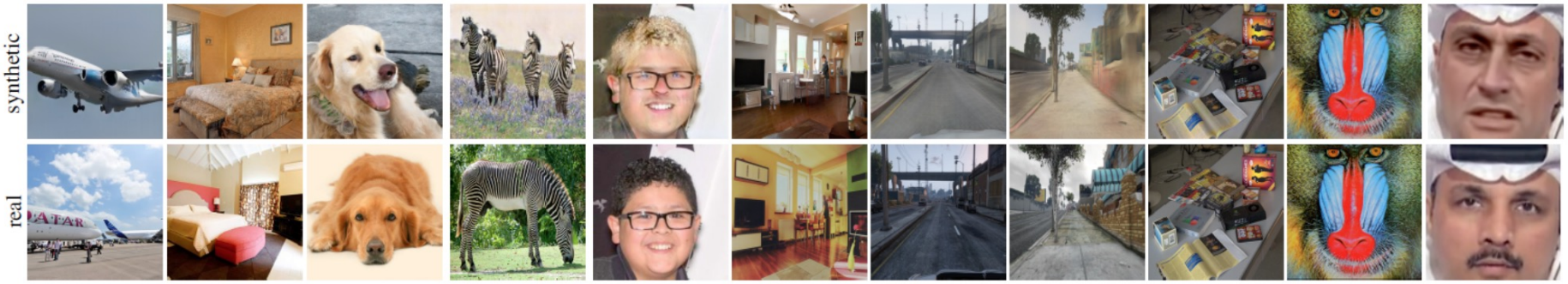
Enlever le bruit



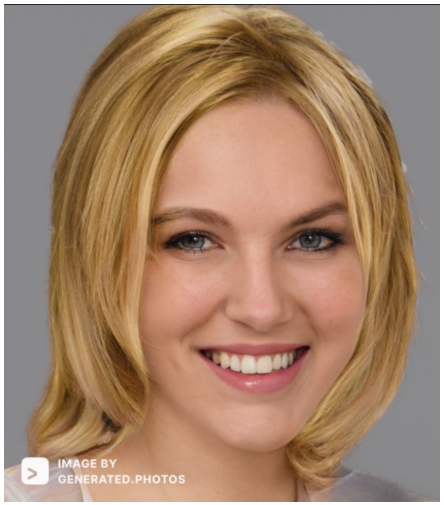
Augmenter la qualité de l'image



Générer des images

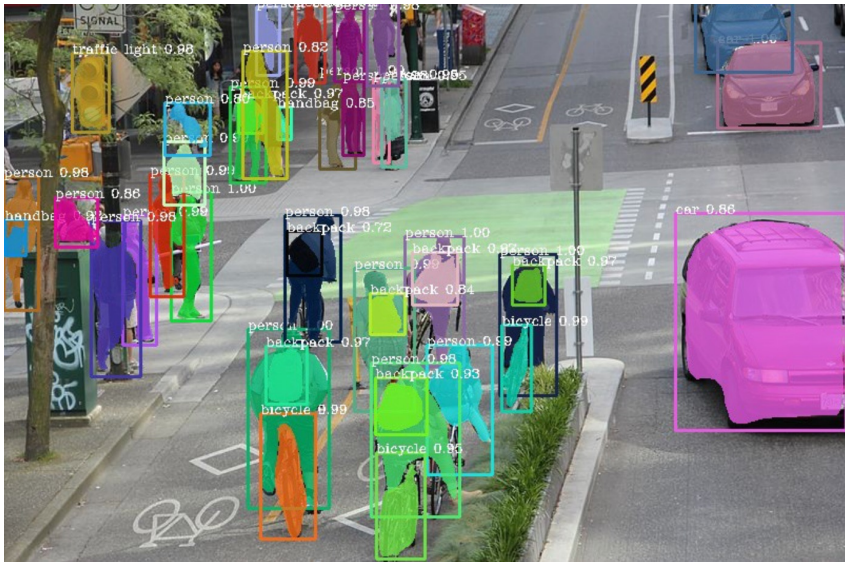


CNN-generated images are surprisingly easy to spot...for now
https://peterwang512.github.io/CNNDetection/

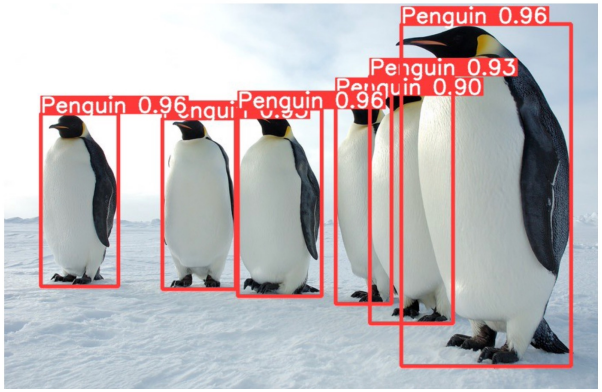


https://generated.photos/face-generator/new

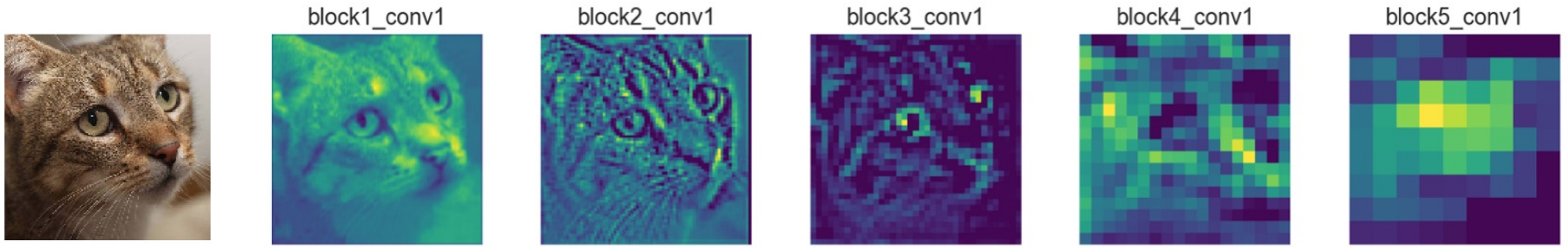
Segmentation d'images



PIXELLIB - https://towardsdatascience.com/image-segmentation-with-six-lines-0f-code-acb870a462e8



Yolo – https://towardsdatascience.com/the-practical-guide-for-object-detection-with-yolov5-algorithm-74c04aac4843



- Des questions ?