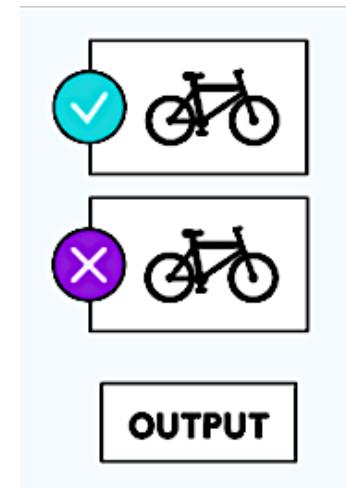
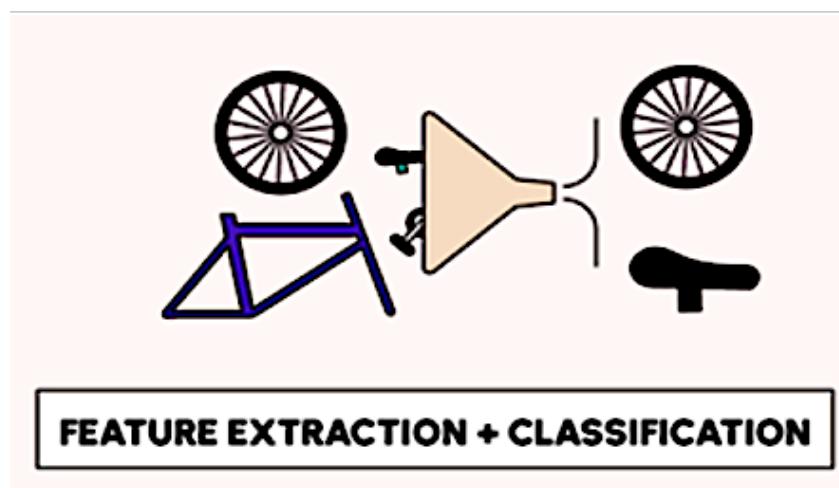
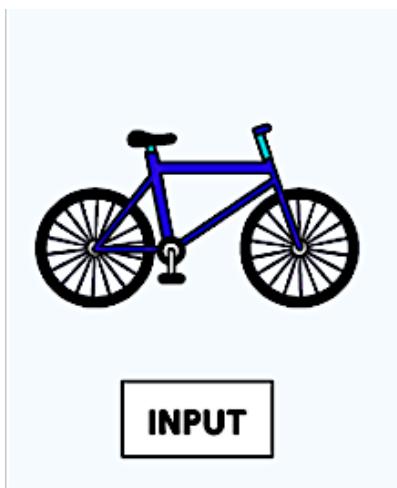
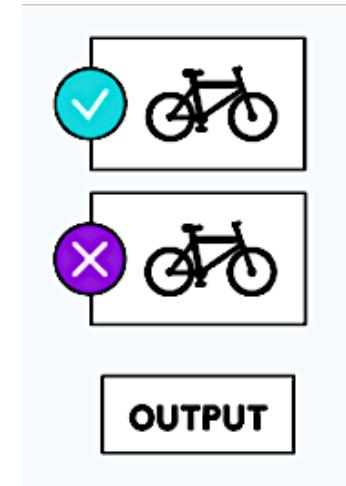
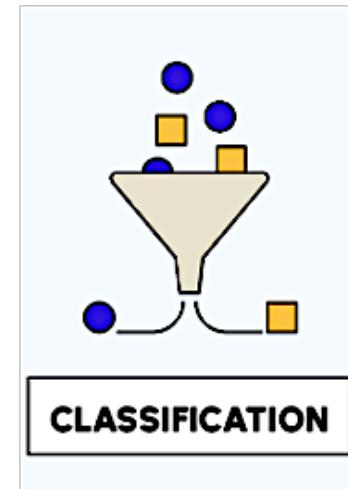
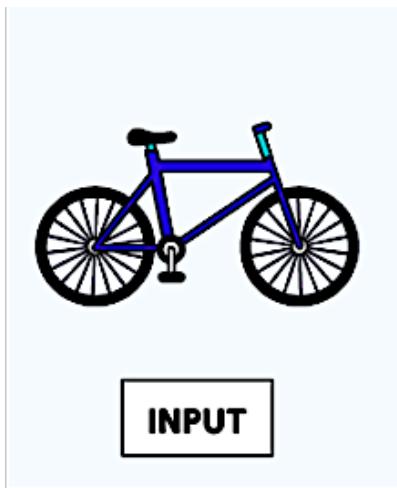


Deep Learning : Les Réseaux de Neurones convolutionnels (CNN)

AL SAIDI IBTISSAM

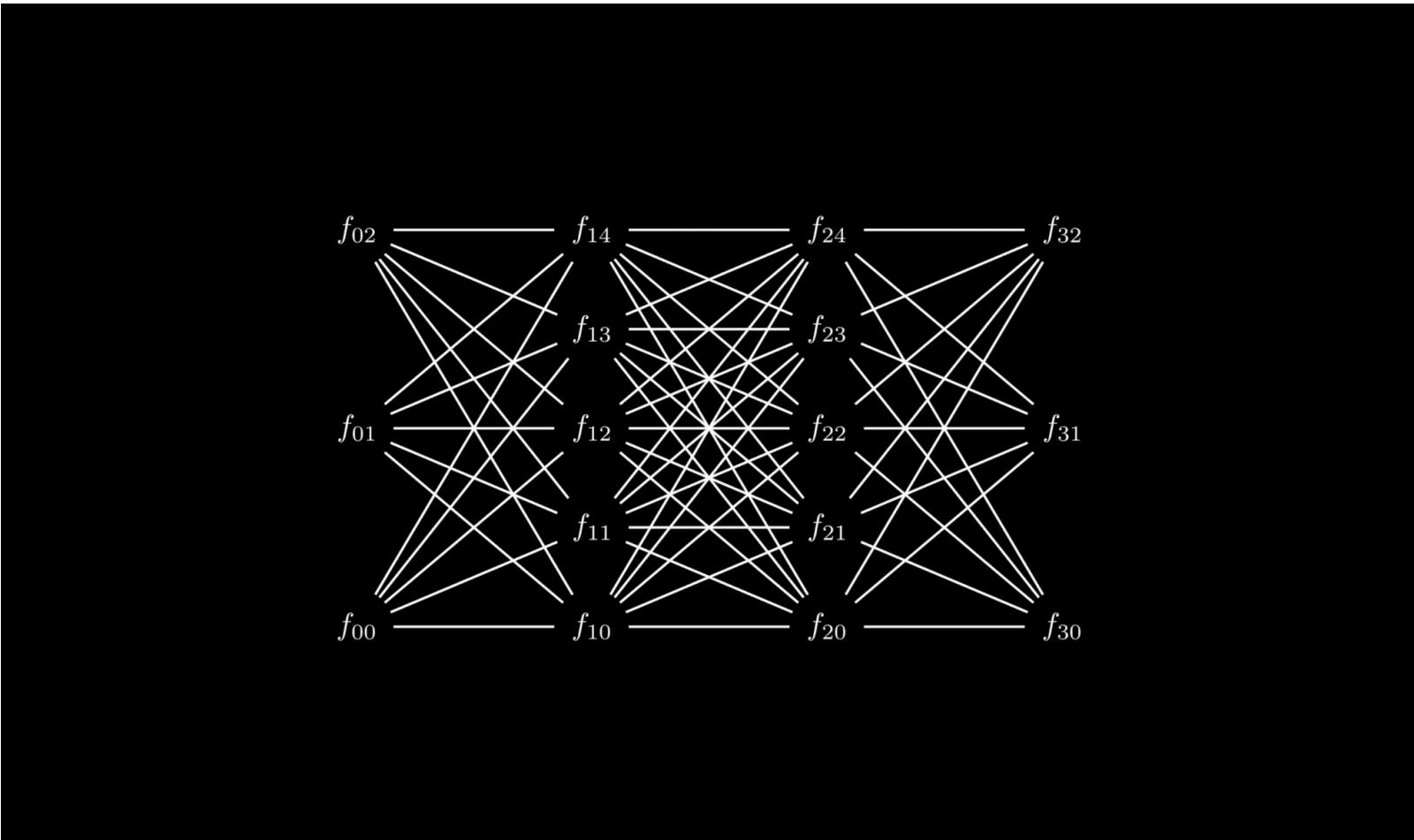
Machine learning VS Deep Learning



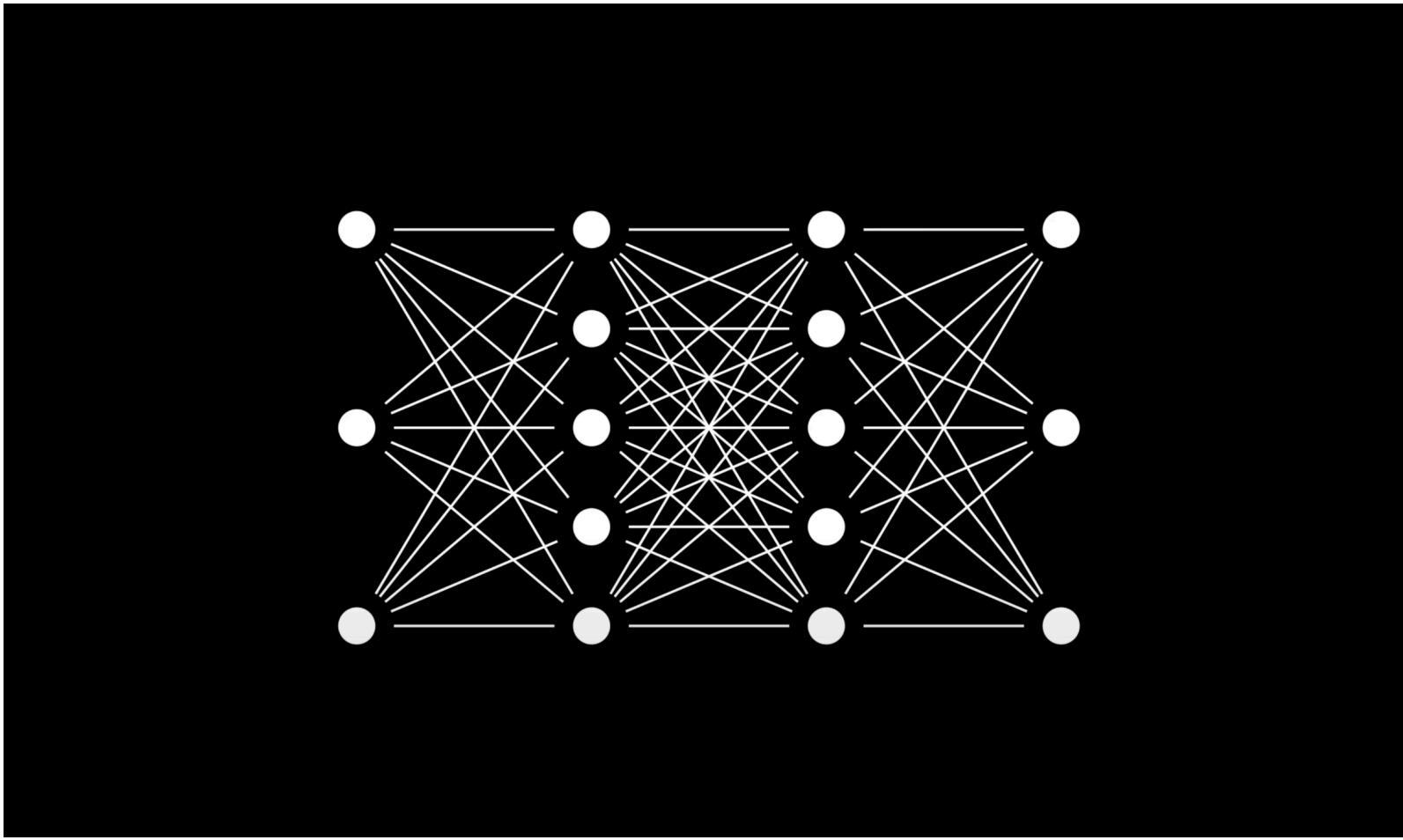
Machine learning VS Deep learning

- MACHINE LEARNING :
 - Développer un **modèle**, en se servant d'un **algorithme d'optimisation** pour **minimiser les erreurs** entre le **modèle** et nos **données**
- qu'en est il du Deep learning ?
 - Deep Learning est un domaine du Machine Learning dans lequel, au lieu de développer un des modèles du machine Learning comme SVM, arbre de décision, etc., on développe à la place ce qu'on appelle des réseaux de neurones artificiels.

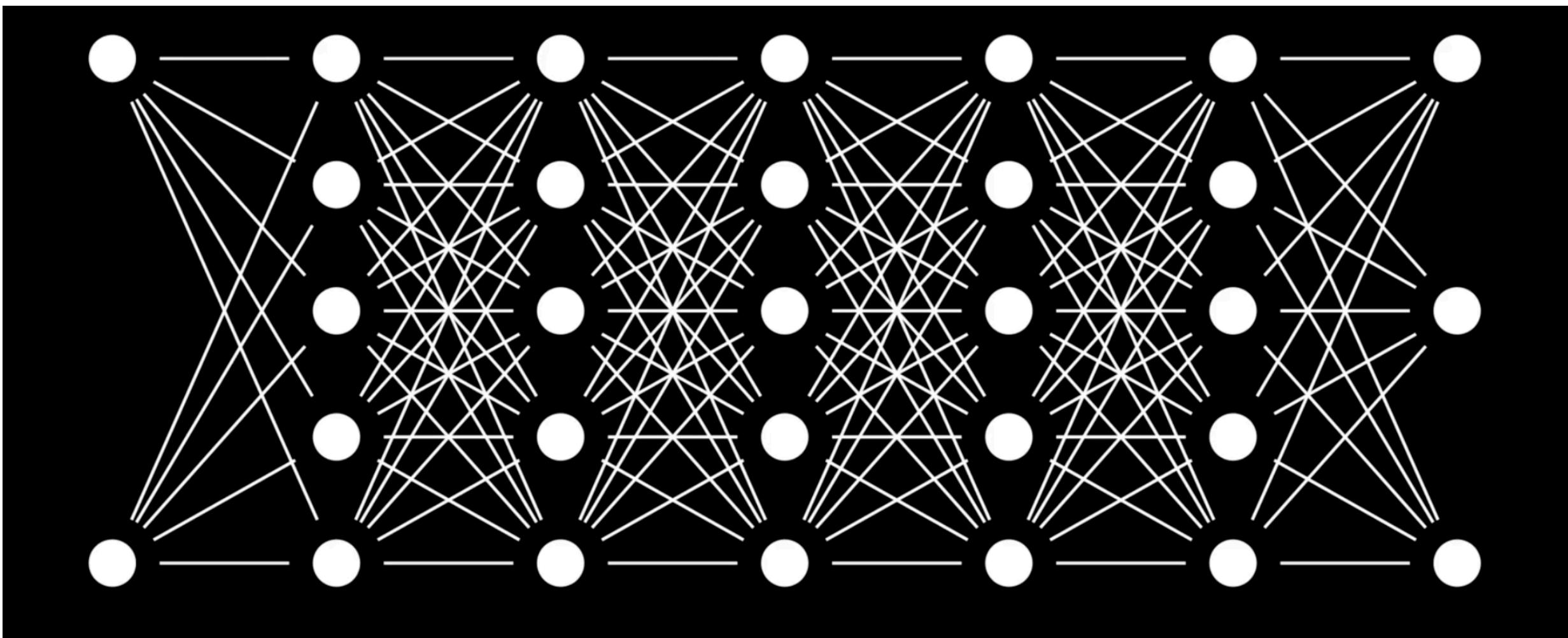
Deep learning



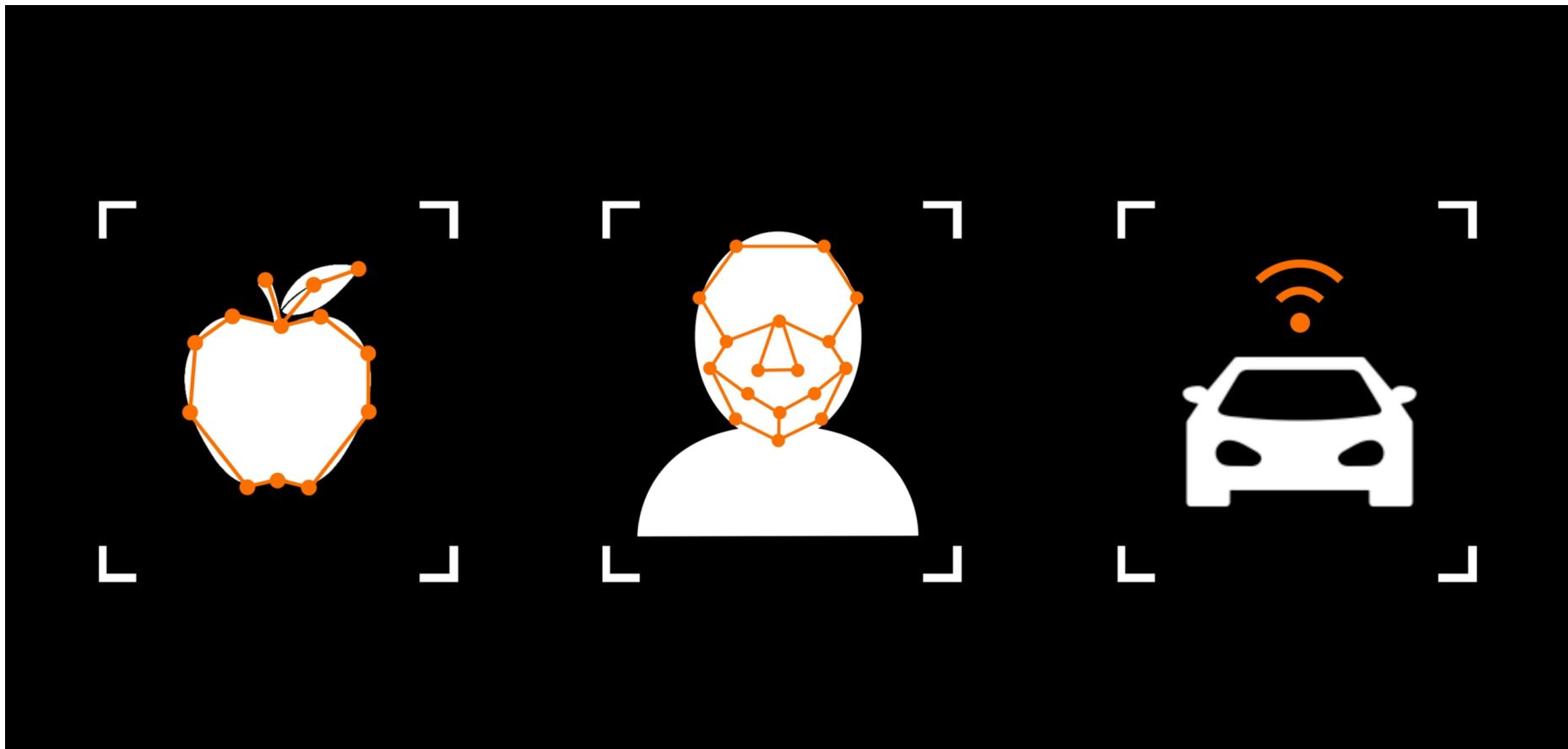
Deep learning



Deep learning



Deep learning



Apprentissage des réseaux de neurones

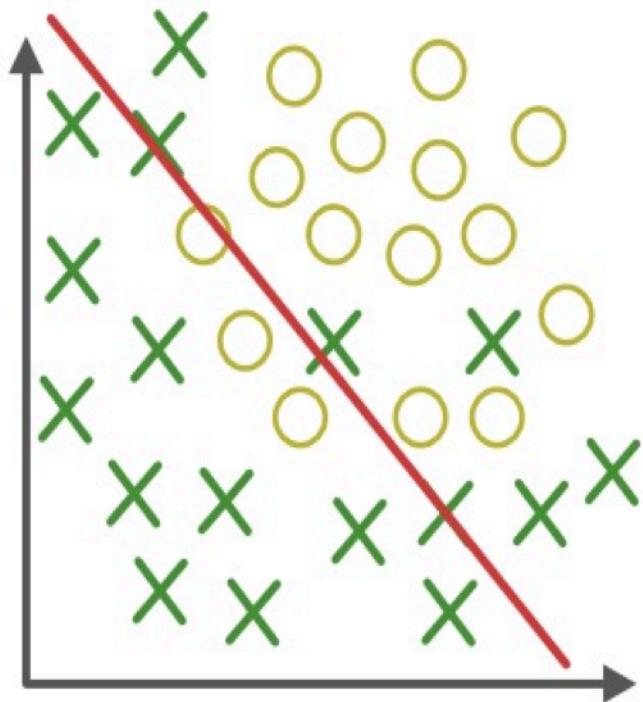
Objectif du Deep Learning

Puisque les réseaux neuronaux peuvent théoriquement représenter N'IMPORTE quelle fonction, comment apprenons-nous à créer des modèles performants pour les données générées dans des problèmes du monde réel...

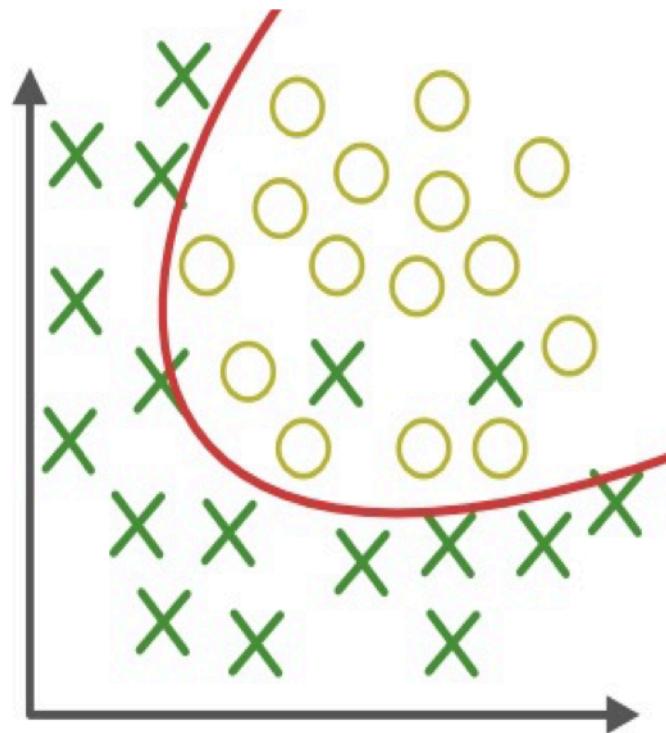
Capacité du modèle

Quel modèle choisiriez-vous pour séparer x de o ?

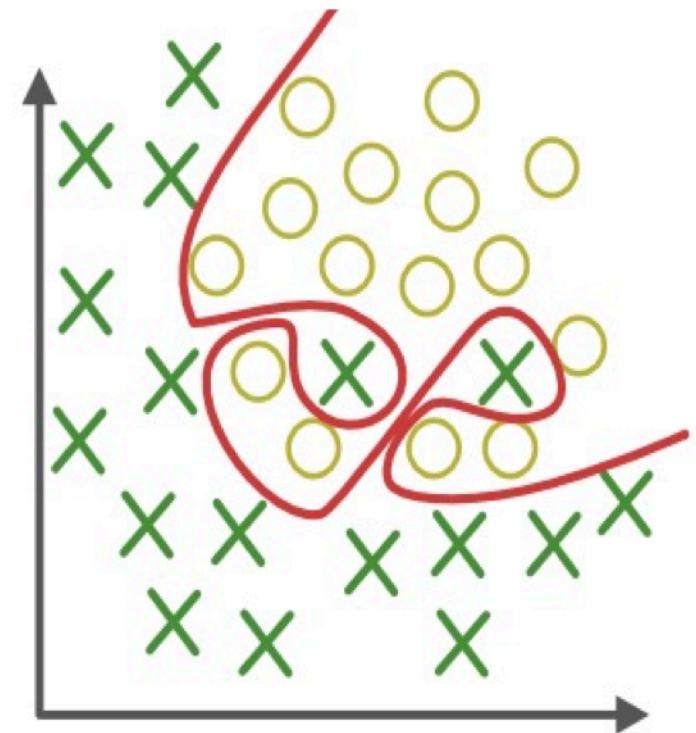
(a)



(b)



(c)

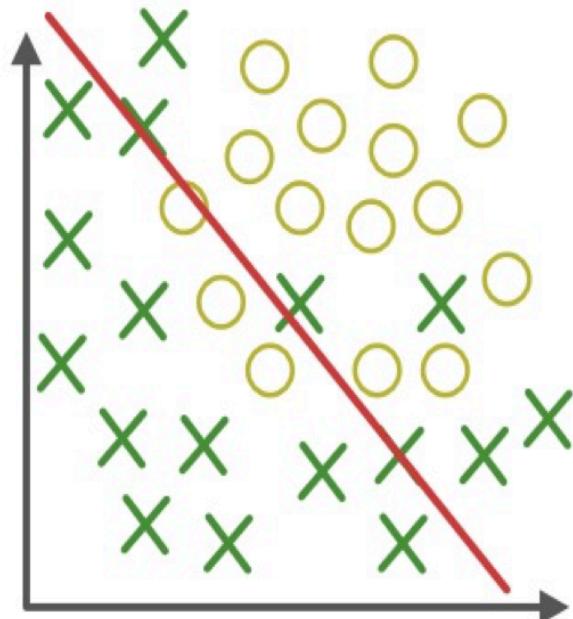


Capacité du modèle

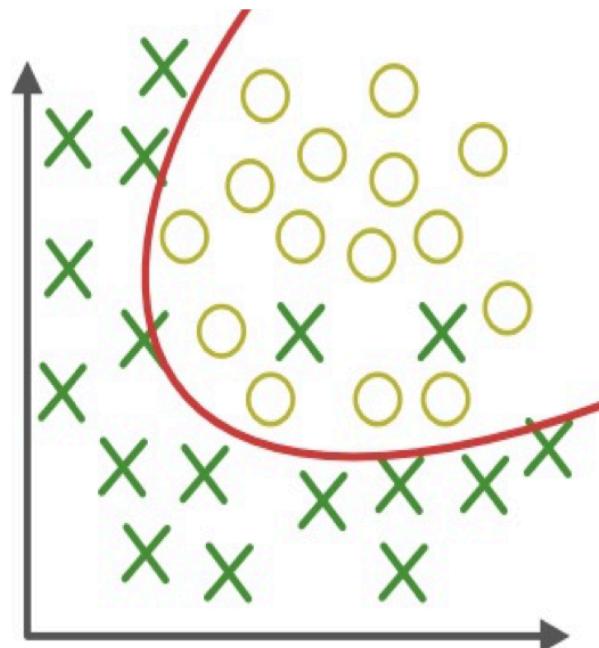
Défi majeur pour les réseaux neuronaux en raison de leur grand nombre de paramètres

Sous-ajustement : trop simple pour expliquer les données

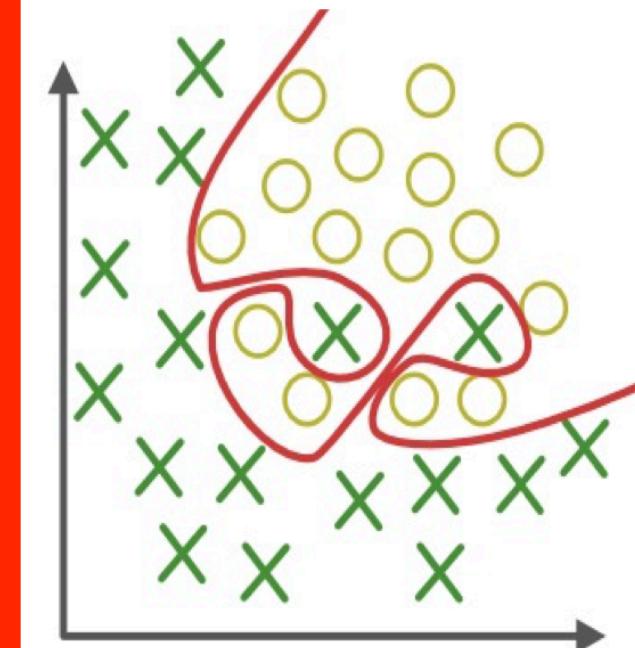
(a)



(b)

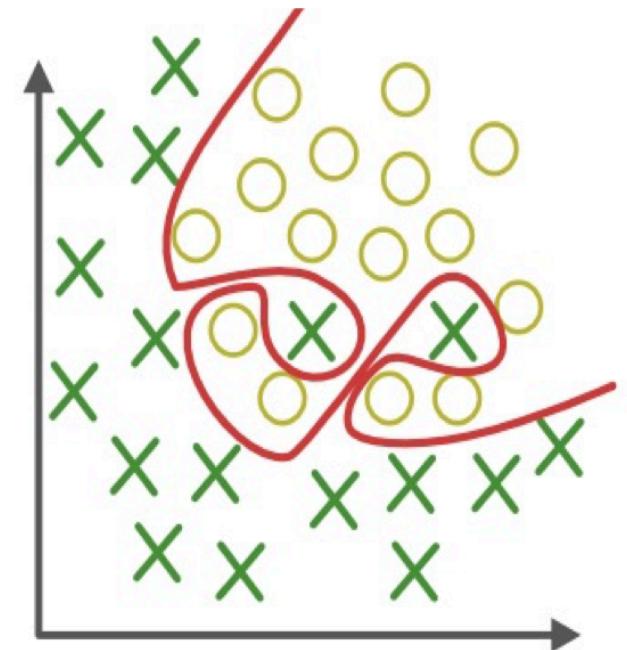


(c)



Capacité du modèle : Sur-ajustement

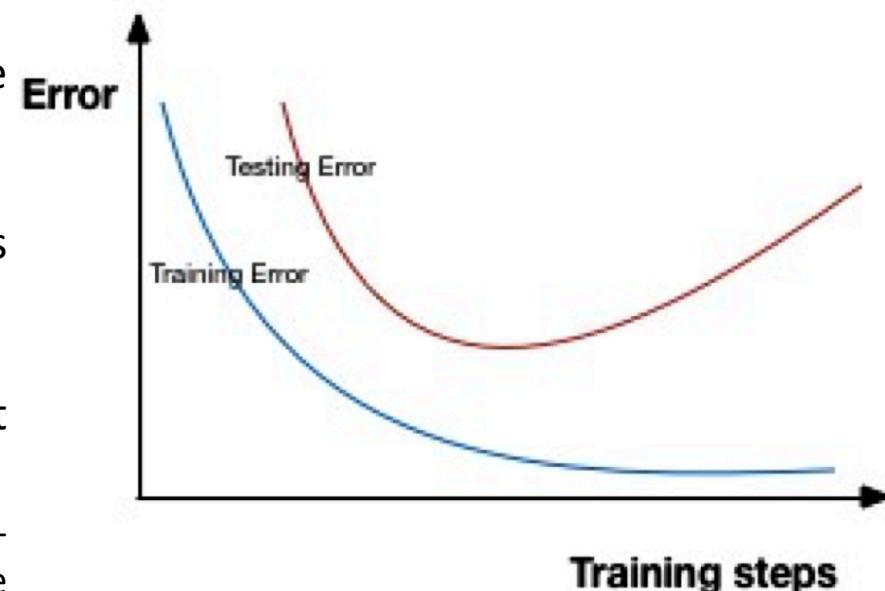
- Qu'apprennent les modèles qui surajustent ?
- Comment modéliser le bruit !
- Quelles pourraient être les causes du bruit dans un ensemble de données ?



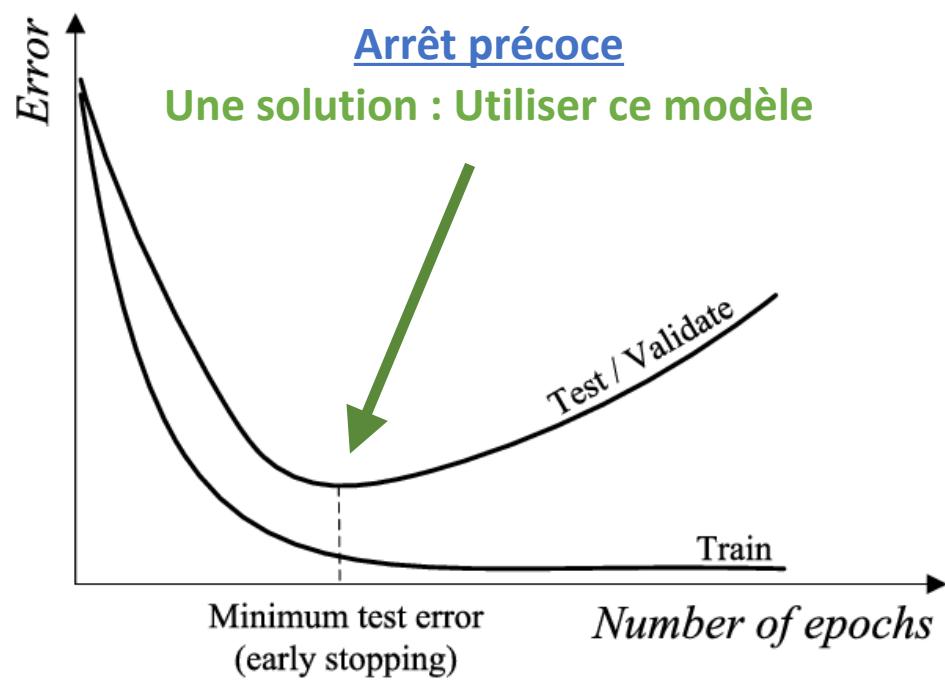
Capacité du modèle : Sur-ajustement

► Pour détecter le sur-ajustement, analysez l'erreur/la perte des modèles testés sur **les données d'entraînement et les données de test.**

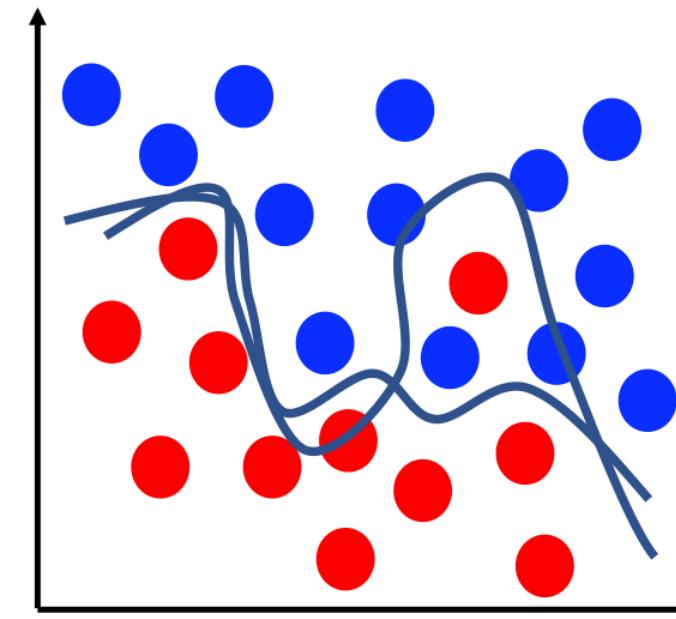
- Qu'arrive-t-il à l'erreur des données d'entraînement lorsque le nombre d'étapes d'entraînement augmente ?
 - ➔ L'erreur diminue.
- Qu'arrive-t-il à l'erreur des données de test lorsque le nombre d'étapes d'entraînement augmente ?
 - ➔ L'erreur diminue puis augmente.
- Pourquoi l'erreur d'entraînement diminue-t-elle et l'erreur de test augmente-t-elle ?
 - ➔ La modélisation du bruit dans les données d'entraînement (c'est-à-dire le « sur-ajustement») réduit l'erreur d'entraînement au détriment de la perte de connaissances qui généraliseraient aux données de test non observées.



Capacité du modèle : Comment éviter le surajustement ?



Ajouter des données d'entraînement

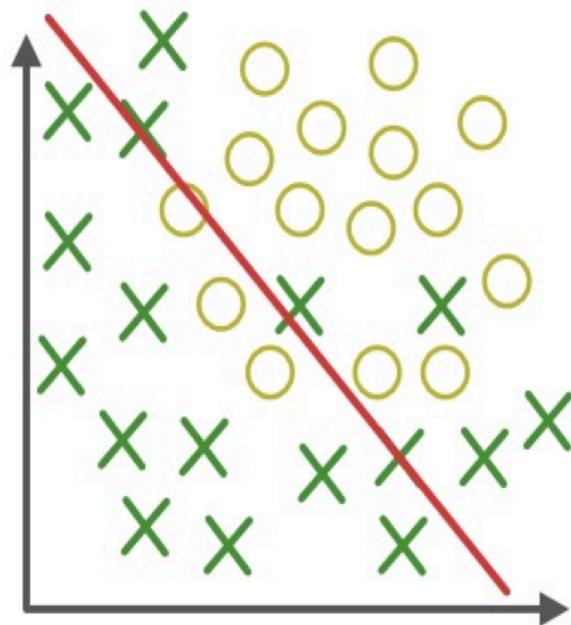


Beaucoup plus de techniques seront discutées dans ce cours...

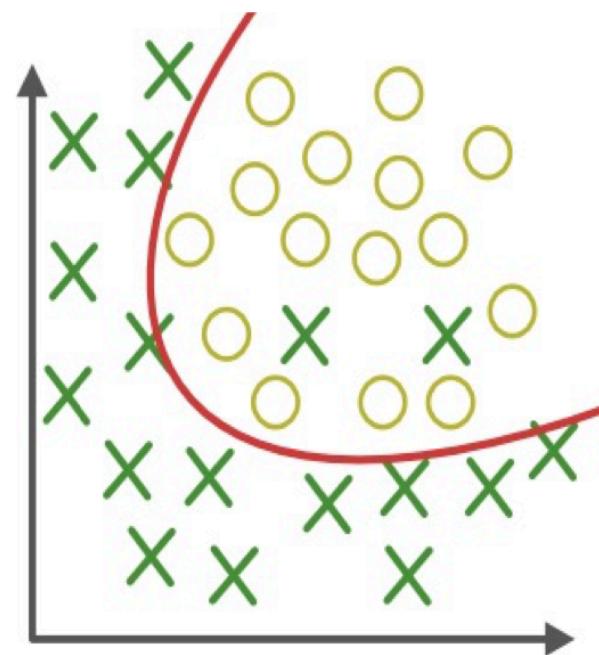
Capacité du modèle

Sous-ajustement : trop simple pour expliquer les données

(a)

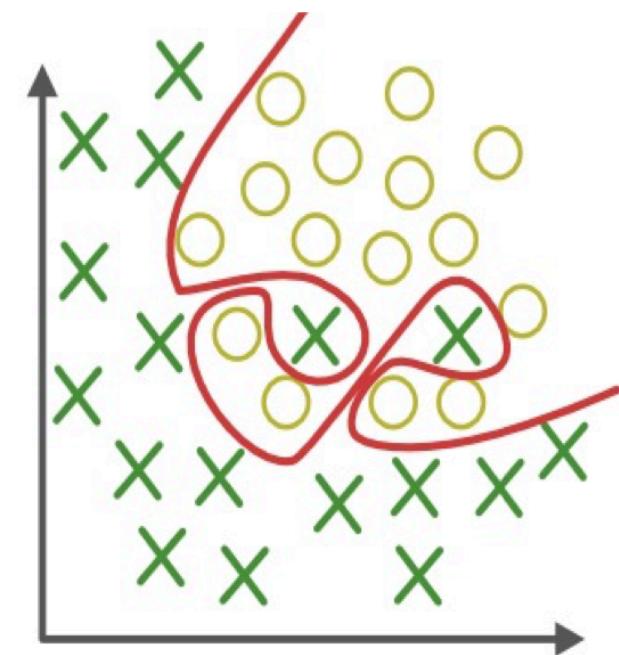


(b)



Sur-ajustement : trop complexe pour généraliser à un ensemble de test

(c)



Capacité du modèle : Sous-ajustement

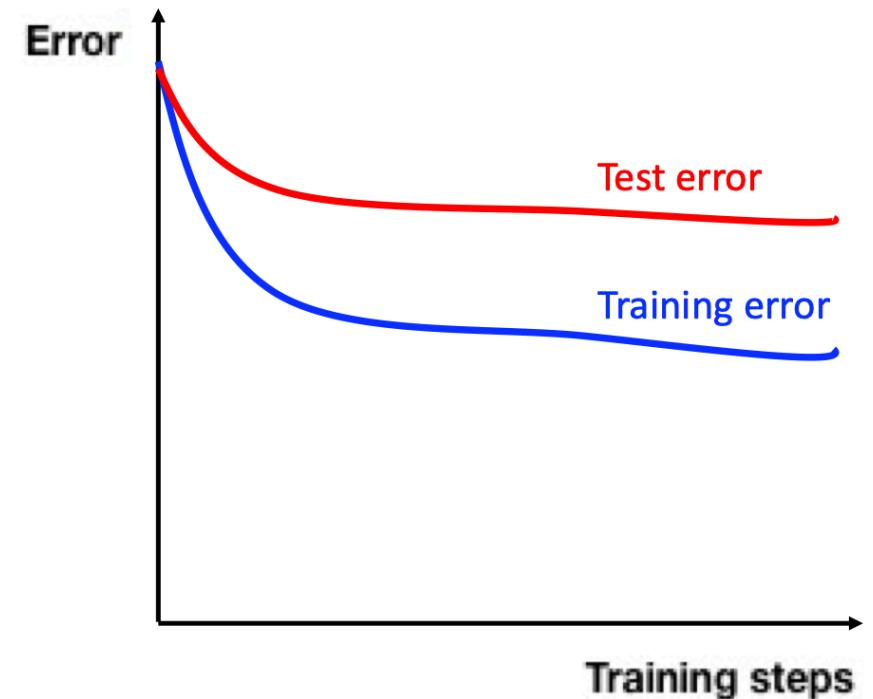
- Pour détecter le sous-ajustement, analysez l'erreur/la perte des modèles testés sur les données d'entraînement (et éventuellement sur les données de test)

- Que se passe-t-il à l'erreur des données d'entraînement à mesure que le nombre d'étapes d'entraînement augmente ?

→ L'erreur reste élevée

- Que se passe-t-il à l'erreur des données de test à mesure que le nombre d'étapes d'entraînement augmente ?

→ L'erreur reste élevée



Capacité du modèle : Comment éviter le sous-ajustement ?

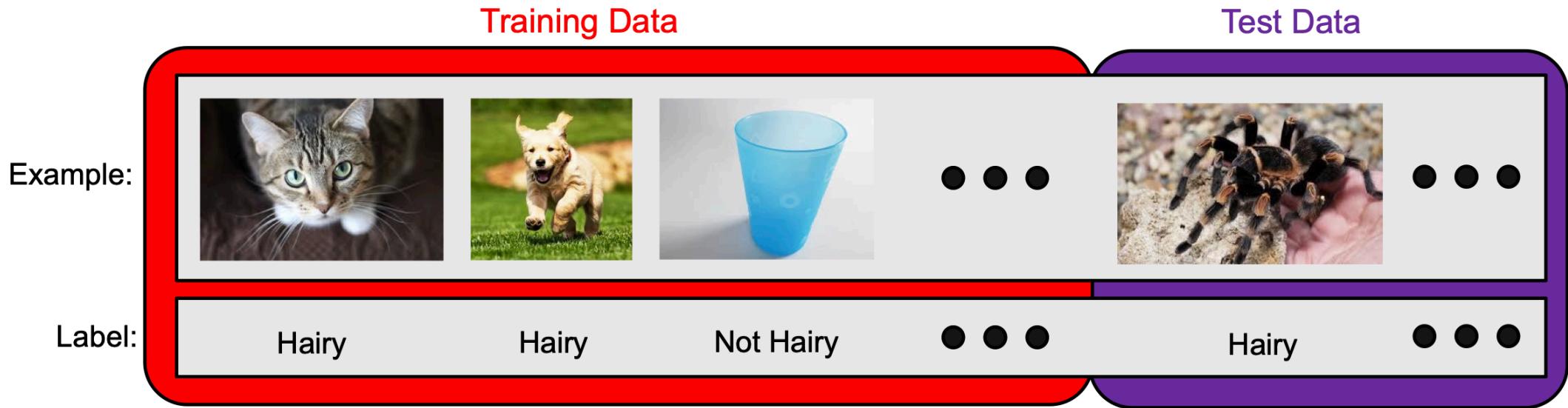
Augmenter la complexité de la représentation, par exemple en ajoutant le nombre de couches et/ou d'unités dans un réseau neuronal.

Capacité du modèle

Objectif : apprendre un modèle avec une capacité ni trop petite ni trop grande afin qu'il puisse bien généraliser lors de la prédiction sur des données de test précédemment non observées.

Sélection des hyperparamètres du modèle

Notre objectif est de concevoir des modèles qui généralisent bien à de nouveaux exemples, préalablement non observés (données de test).



Défi majeur : comment sélectionner un modèle sans observer à plusieurs reprises les données de test (ce qui entraîne un sur-ajustement) ?

Sélection des hyperparamètres du modèle : Décisions de conception du modèle

Hyperparamètres du modèle (sélectionnés) ; par exemple,

- Nombre de couches
- Nombre d'unités dans chaque couche
- Fonction d'activation
- Taille du lot (batch size)
- Taux d'apprentissage (learning rate)
- ...

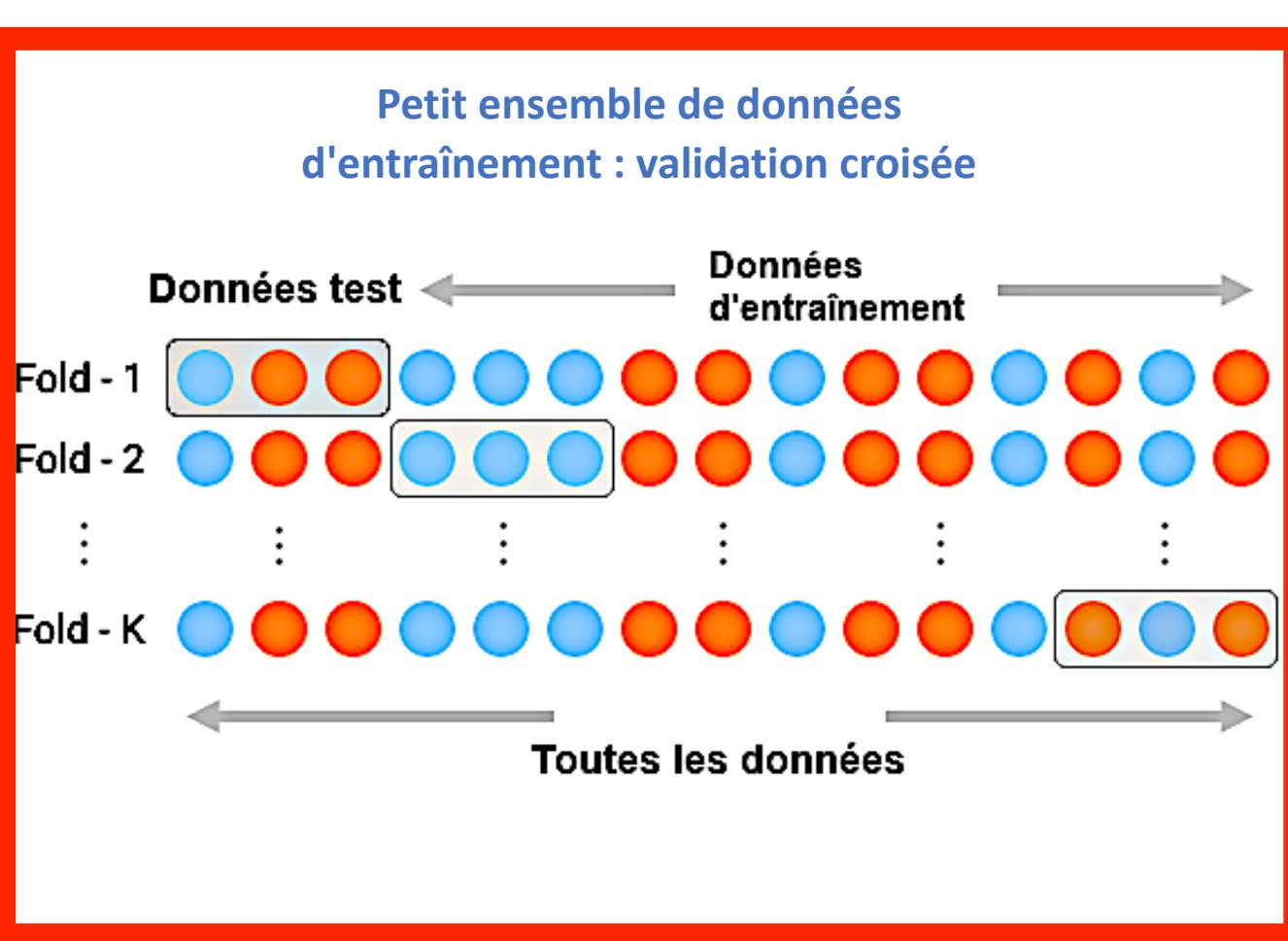
Paramètres du modèle (trainer)

Poids
Biais

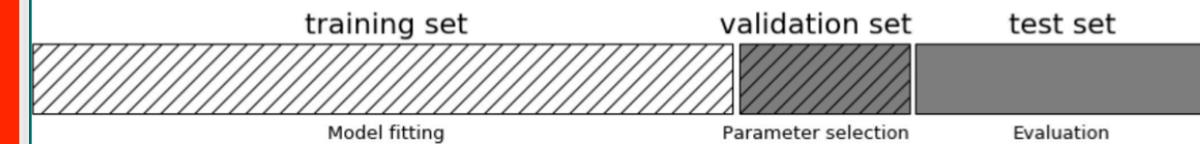
Défi clé : comment sélectionner un modèle sans observer de manière répétée les données de test (ce qui entraîne un surajustement) ?

Sélection des hyperparamètres du modèle : Décisions de conception du modèle

Pour des résultats statistiquement solides :



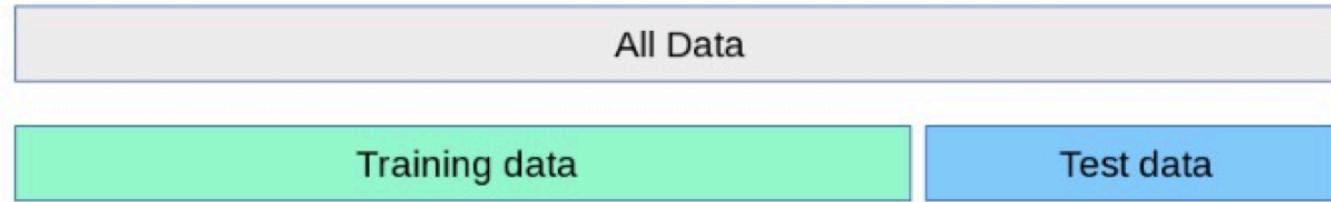
Sinon : division entraînement/validation



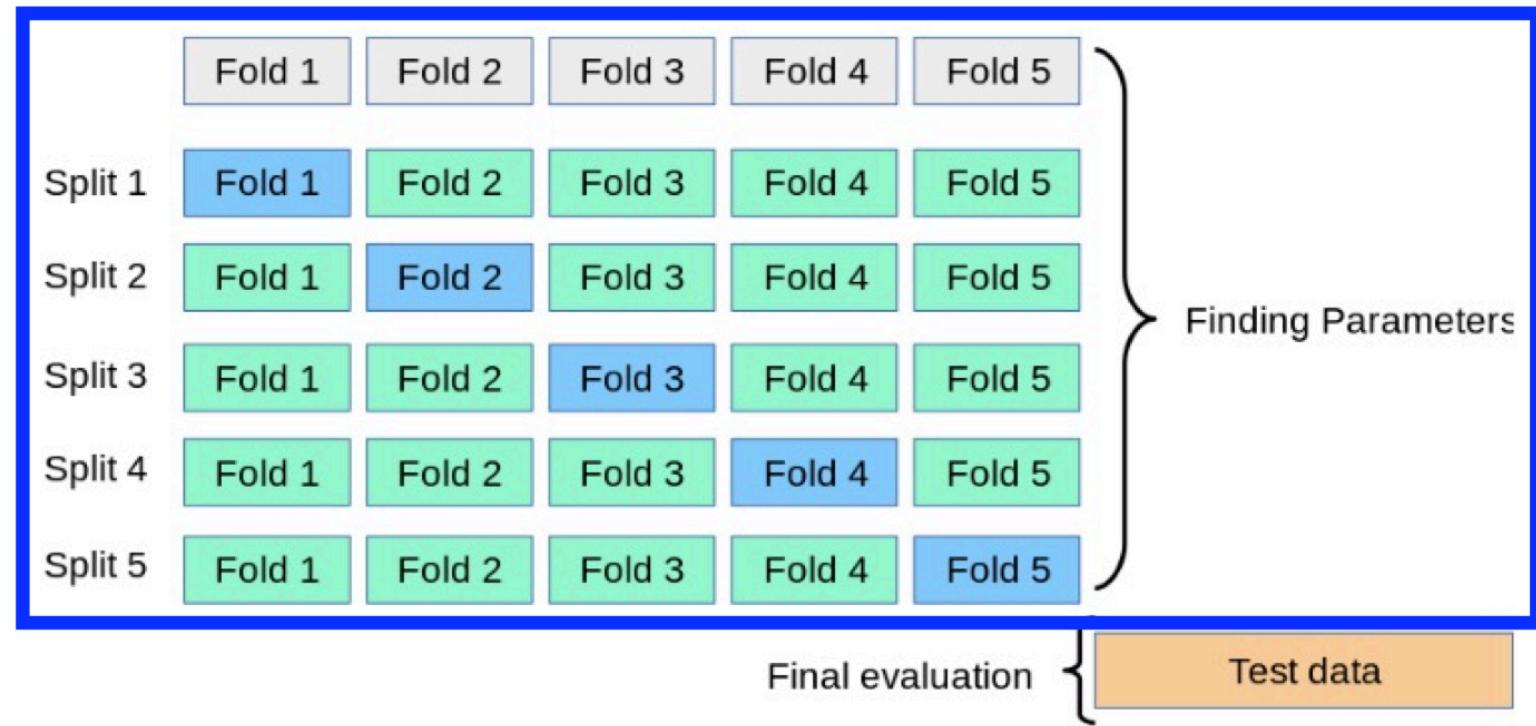
Sélection des hyperparamètres du modèle : Validation croisée

- ▶ **Validation croisée** (k-fold, où k = 10 est le plus populaire)
- ▶ Partitionnez les données de manière aléatoire en k sous-ensembles mutuellement exclusifs, chacun ayant une taille approximativement égale.
- ▶ À la i-ème itération, utilisez Fold_i comme ensemble de test et les autres comme ensemble d'entraînement.

Sélection des hyperparamètres du modèle : Validation croisée (Limitez l'influence de la répartition des ensembles de données)

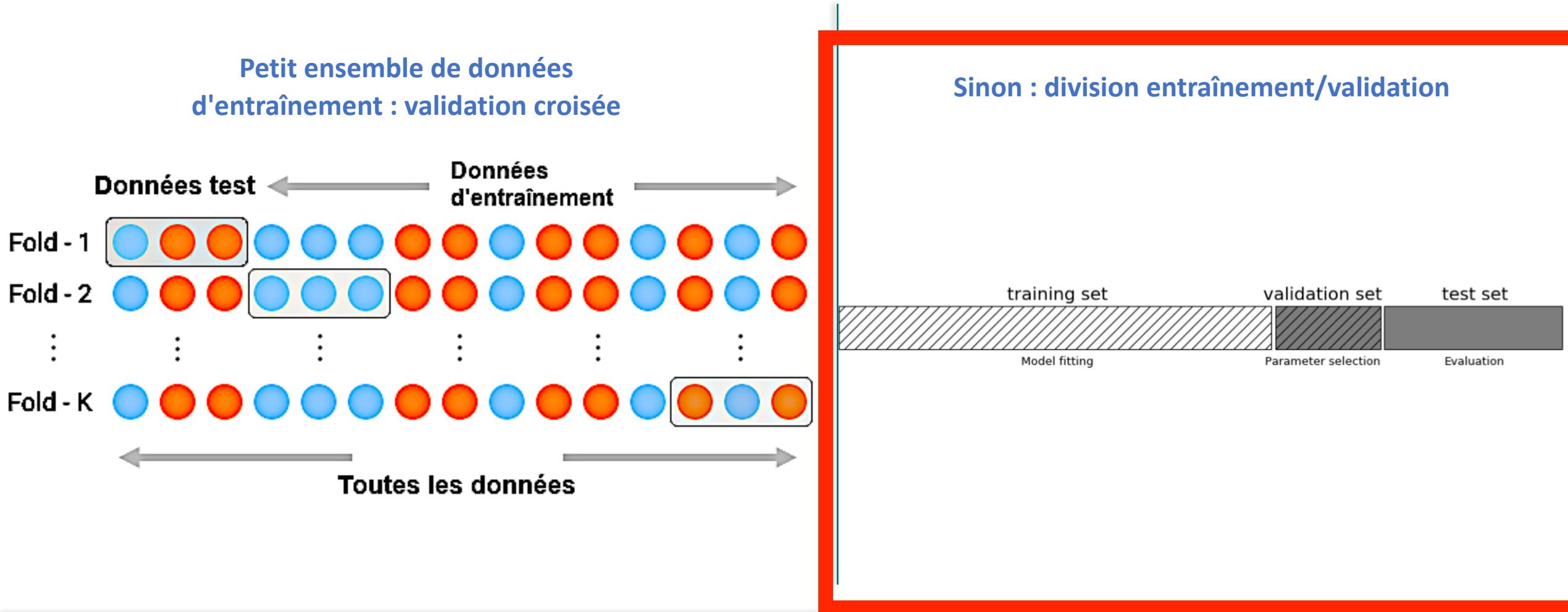


**Généralement,
sélectionnez les
hyperparamètres
qui conduisent aux
meilleurs résultats
globaux à travers
toutes les
partitions.**



Sélection des hyperparamètres du modèle : Décisions de conception du modèle

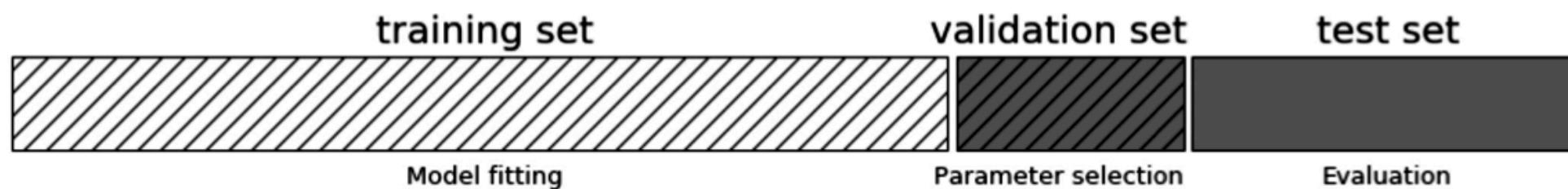
Pour des résultats statistiquement solides :





Sélection des hyperparamètres du modèle : Répartition de la validation

- ▶ Diviser les données d'entraînement en ensembles "entraînement" et « validation»

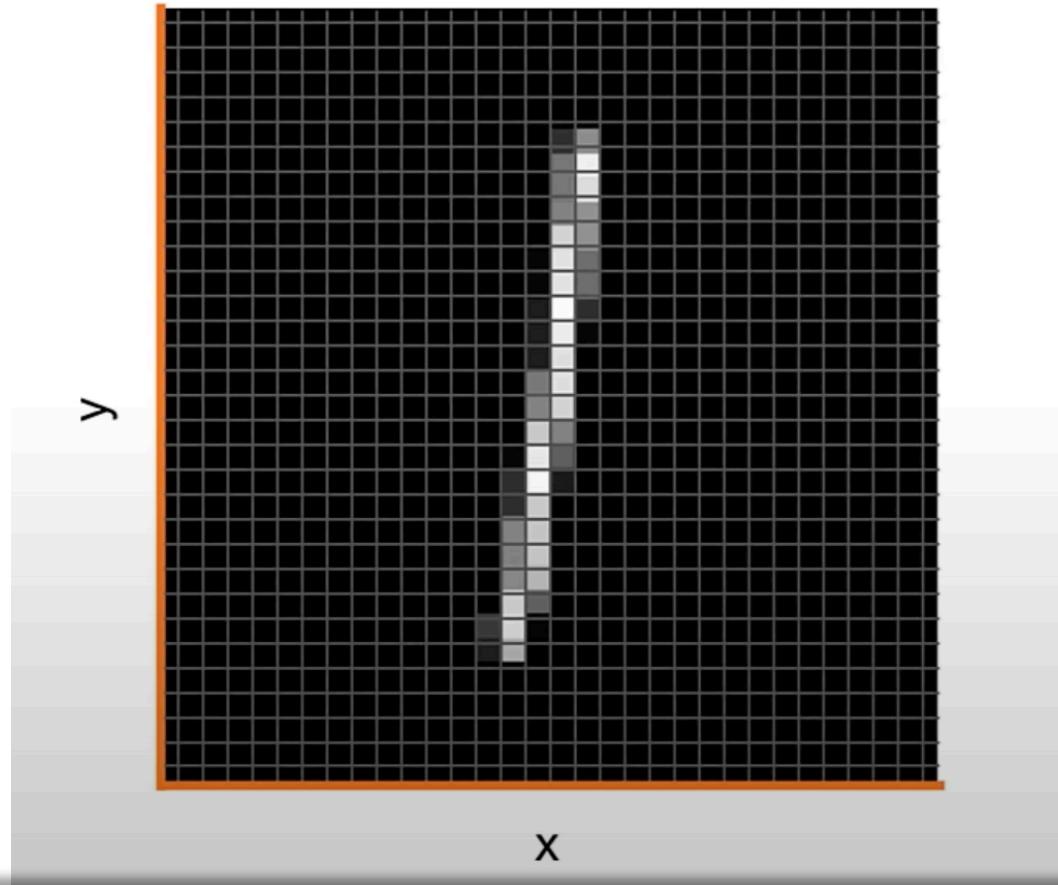


- ▶ Sélection des **hyperparamètres** : tester les modèles entraînés avec différentes valeurs d'hyperparamètres sur l'ensemble de validation pour trouver le meilleur
- ▶ Modèle final : réentraîner en utilisant les hyperparamètres du modèle sélectionnés à partir des tests sur l'ensemble de validation en utilisant les données des divisions d'entraînement ET de validation

Perceptron multi-couches pour la classification

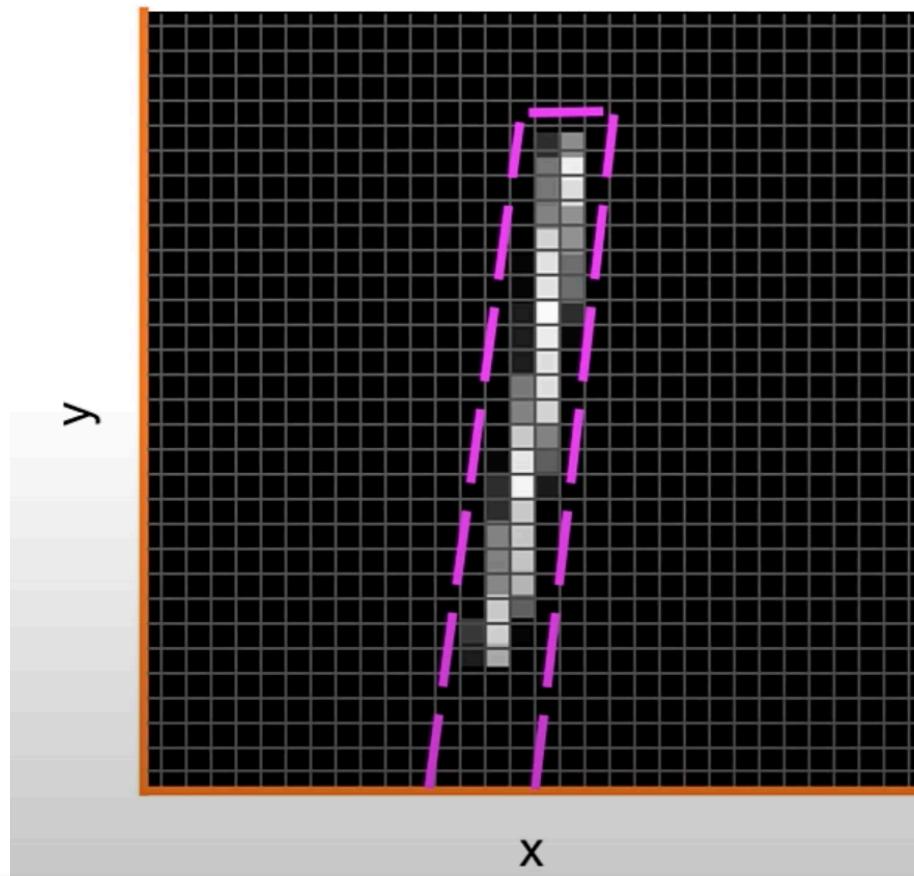
Classification : Images

- Élargir ce que vous connaissez



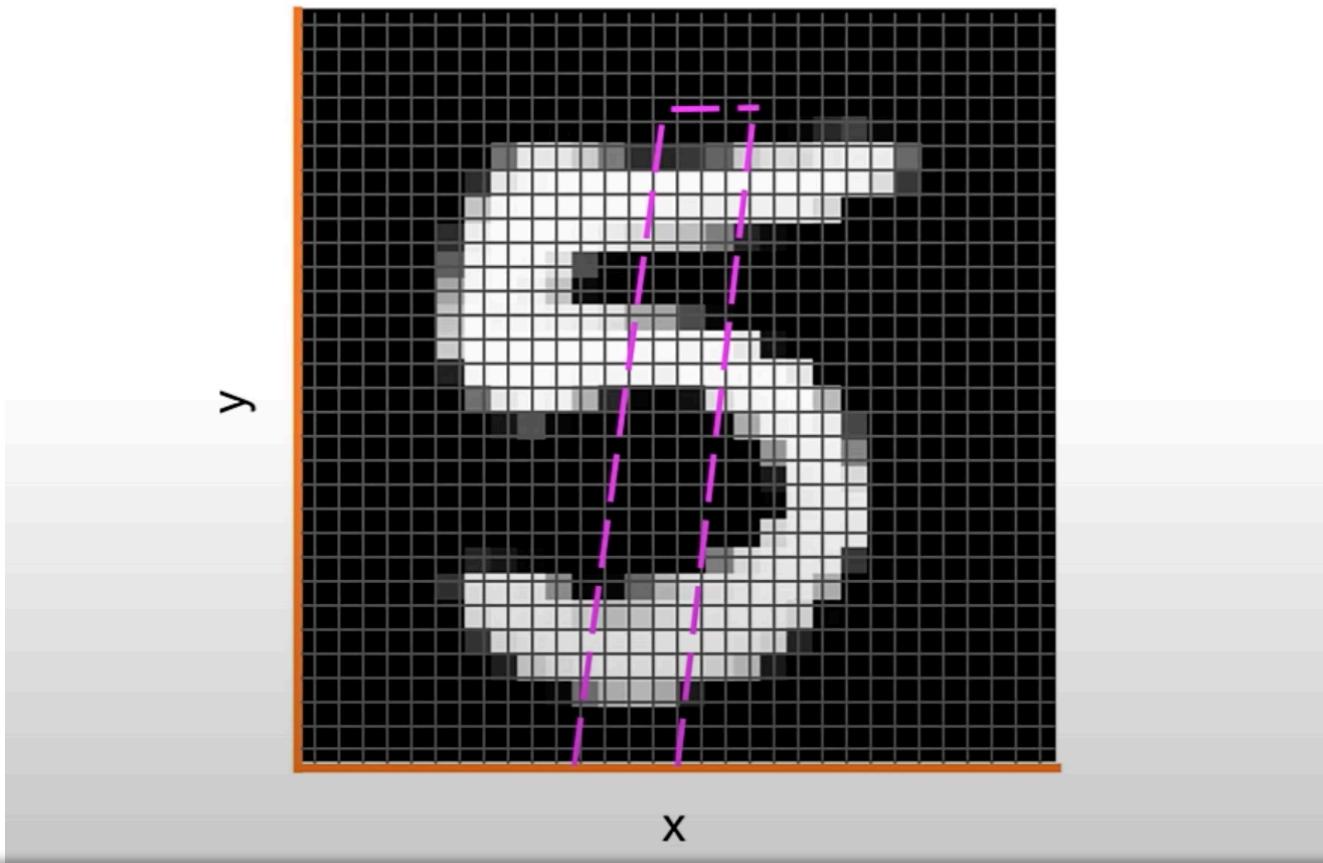
Classification : Images

- Élargir ce que vous connaissez



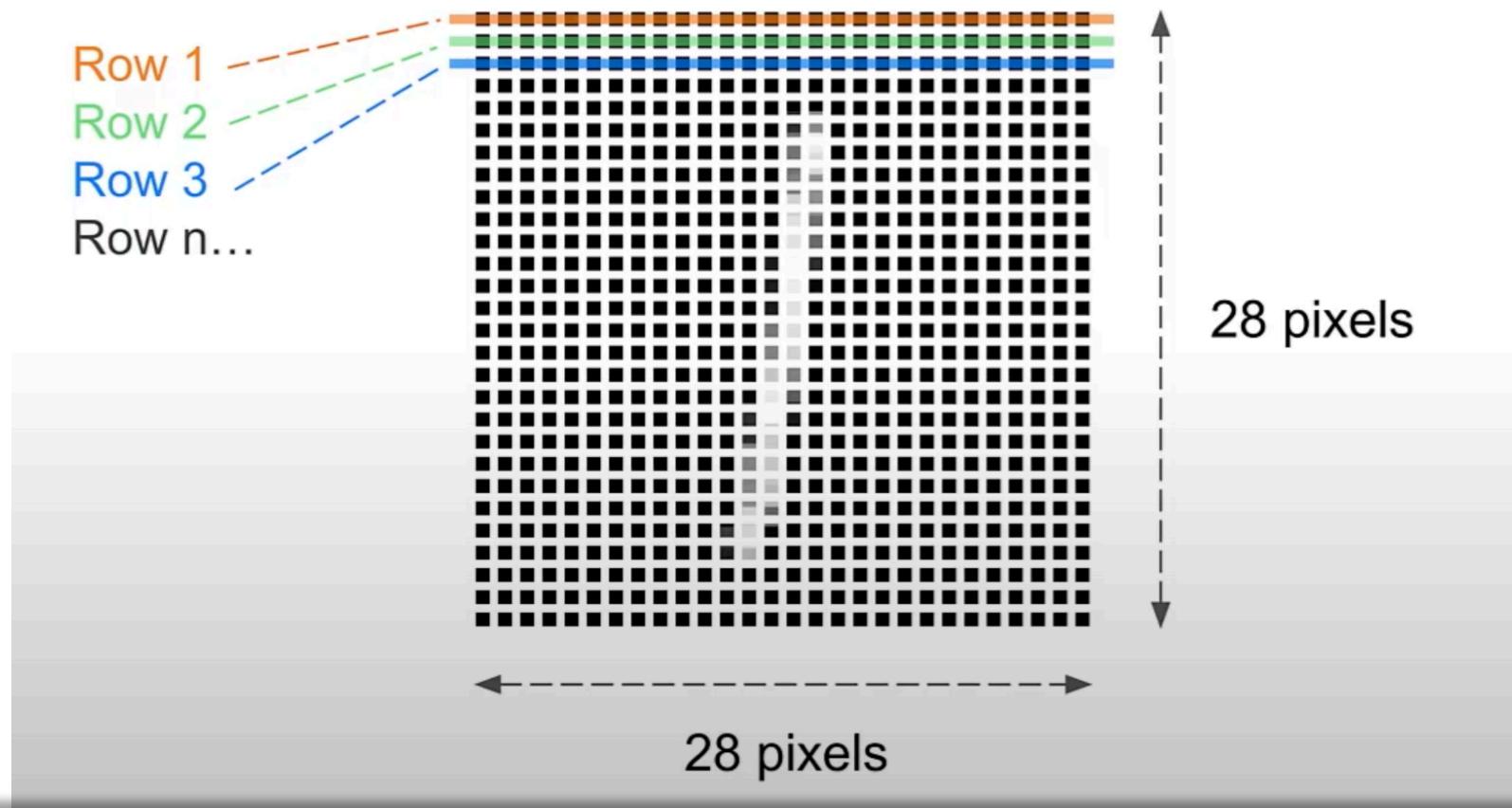
Classification : Images

- Élargir ce que vous connaissez



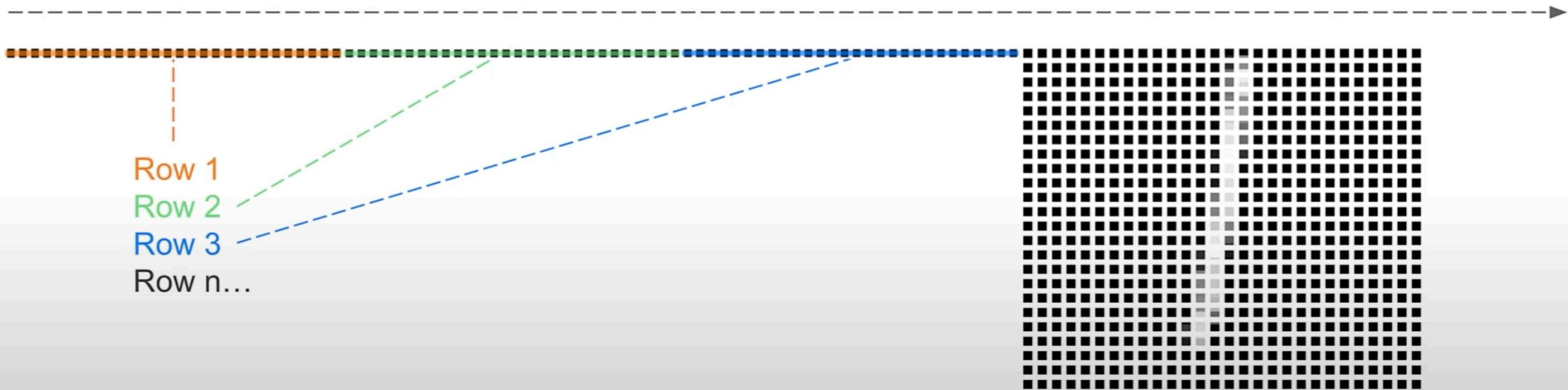
Transformation des images en tenseurs

- Pour une utilisation dans un perceptron multicouche



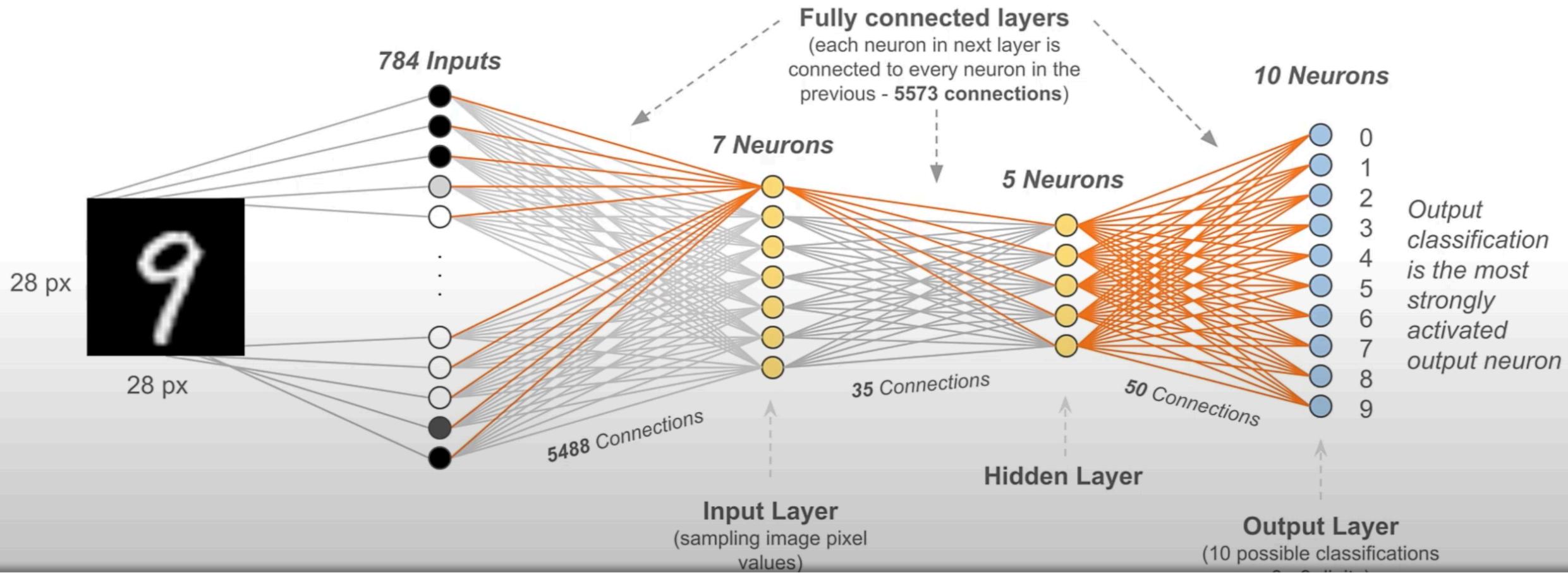
Transformation des images en tenseurs

- Pour une utilisation dans un perceptron multicouche



Perceptron multicouche - réseaux neuronaux profonds

- Chaque couche est entièrement connectée à la suivante. Les données ne circulent que dans le sens forward :



Encodage one-hot : De la régression à la classification

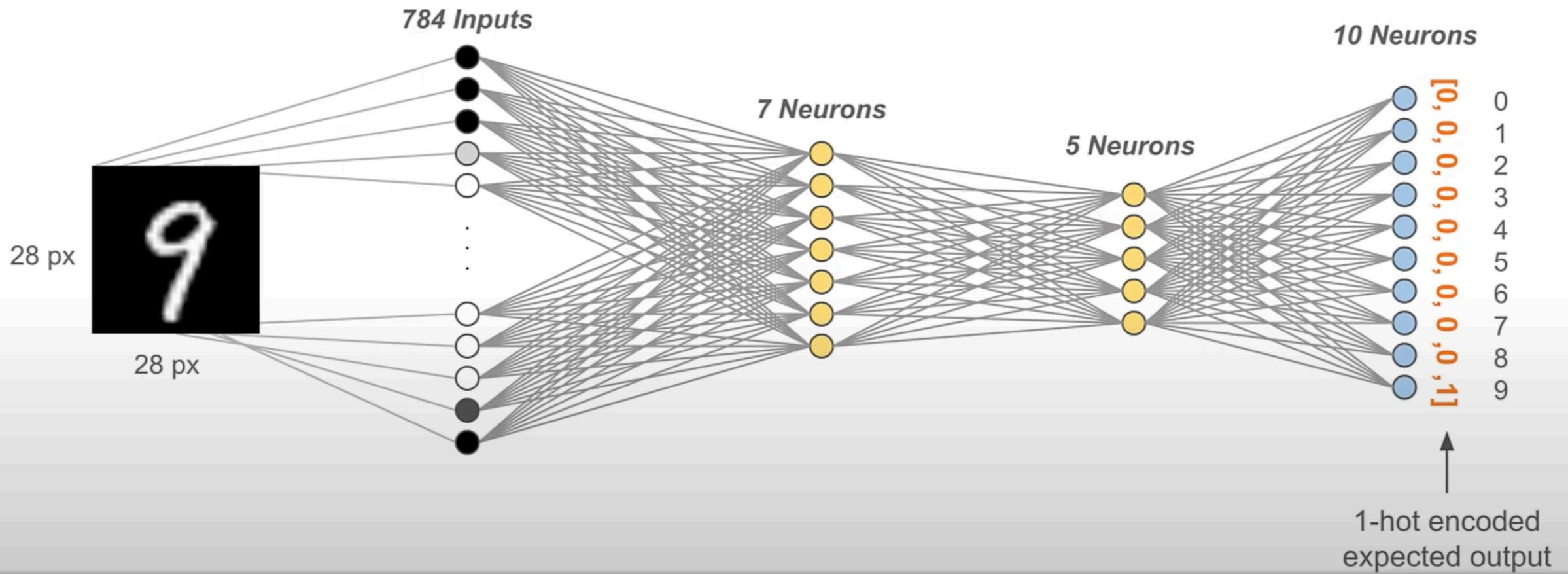
- En introduisant les encodages "1-hot" (encodage one-hot).

$4 == [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$

$0 == [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$
$1 == [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]$
$2 == [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$
$3 == [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$
$4 == [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]$
$5 == [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$
$6 == [0, 0, 0, 0, 0, 0, 1, 0, 0, 0]$
$7 == [0, 0, 0, 0, 0, 0, 0, 1, 0, 0]$
$8 == [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$
$9 == [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$

Encodage one-hot

- En introduisant les encodages "1-hot" (encodage one-hot).



Activation Softmax

- Les sorties qui se totalisent à 1.
- [0,04, 0,01, 0, 0,03, **0,89**, 0, 0,01, 0, 0, 0,02]
- Ces chiffres s'additionnent pour atteindre un total de 1.

Categorical Cross-Entropy

- Une autre fonction de coût sophistiquée

```
model.compile({  
    optimizer: 'sgd',  
    loss: 'categoricalCrossentropy',  
});
```

Mesures de performance pour la classification

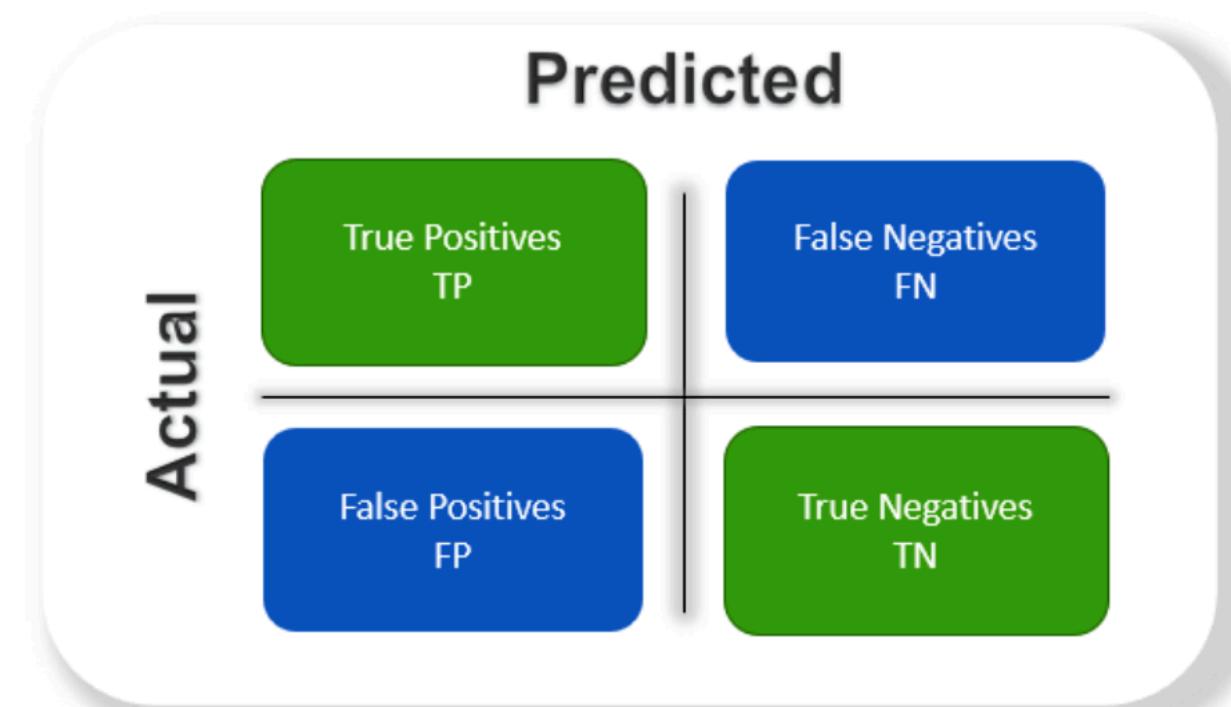
- Confusion Matrix
- Classification Accuracy
- Precision
- Recall
- F1 Score

Mesures de performance pour la classification : Confusion Matrix

- ▶ Pour simplifier, nous discuterons principalement des choses en termes de problème de classification binaire où disons que nous devrons trouver si une image est celle d'un chat ou d'un chien. Ou un patient a un cancer (positif) ou est trouvé en bonne santé (négatif).
- ▶ Certains termes courants pour être clairs sont:
 - ✓ **True positives (TP)**: Predicted positive and are actually positive.
 - ✓ **False positives (FP)**: Predicted positive and are actually negative.
 - ✓ **True negatives (TN)**: Predicted negative and are actually negative.
 - ✓ **False negatives (FN)**: Predicted negative and are actually positive.

Mesures de performance pour la classification : Confusion Matrix

- Il s'agit simplement d'une représentation des paramètres en haut dans un format matriciel. Une meilleure visualisation est toujours bonne :)



Mesures de performance pour la classification : Confusion Matrix (Classification Accuracy)

- ▶ La métrique la plus couramment utilisée pour juger un modèle et n'est en fait pas un indicateur clair de la performance. Le pire se produit lorsque les classes sont déséquilibrées.

$$\frac{TP + TN}{TP + FP + TN + FN}$$

$$Accuracy = \frac{NumberOfCorrectPredictions}{TotalNumberOfPredictionsMade}$$

- ▶ Prenons par exemple un modèle de détection du cancer : Les chances d'avoir un cancer sont très faibles.
 - Disons que sur 100 :
 - ➔ 90 des patients n'ont pas de cancer
 - ➔ 10 autres patients l'ont effectivement.
 - Nous ne voulons pas manquer un patient qui a un cancer mais qui n'est pas détecté (faux négatif). Détecter tout le monde comme n'ayant pas de cancer donne une précision de 90%. Le modèle n'a rien fait ici, mais a simplement donné le cancer gratuitement pour toutes les 100 prédictions.
- ▶ Nous avons certainement besoin de meilleures alternatives.

Mesures de performance pour la classification : Confusion Matrix (Precision)

- ▶ Pourcentage d'instances positives sur le total des instances positives prévues. Ici, le dénominateur est la prédiction du modèle effectuée comme positive à partir de l'ensemble de données donné.
- ▶ Prenez-le comme pour savoir «**à quel point le modèle est juste quand il dit qu'il est juste**».

$$\frac{TP}{TP + FP}$$

Mesures de performance pour la classification : Confusion Matrix (Recall/Sensitivity/True Positive Rate)

- ▶ Pourcentage d'instances positives par rapport au nombre total d'instances positives réelles. Par conséquent, le dénominateur ($TP + FN$) est ici le nombre réel d'instances positives présentes dans l'ensemble de données.
- ▶ Prenez-le comme pour découvrir «**combien de bons supplémentaires, le modèle a manqué quand il a montré les bons**».

$$\frac{TP}{TP + FN}$$

Mesures de performance pour la classification : Confusion Matrix (F1 score)

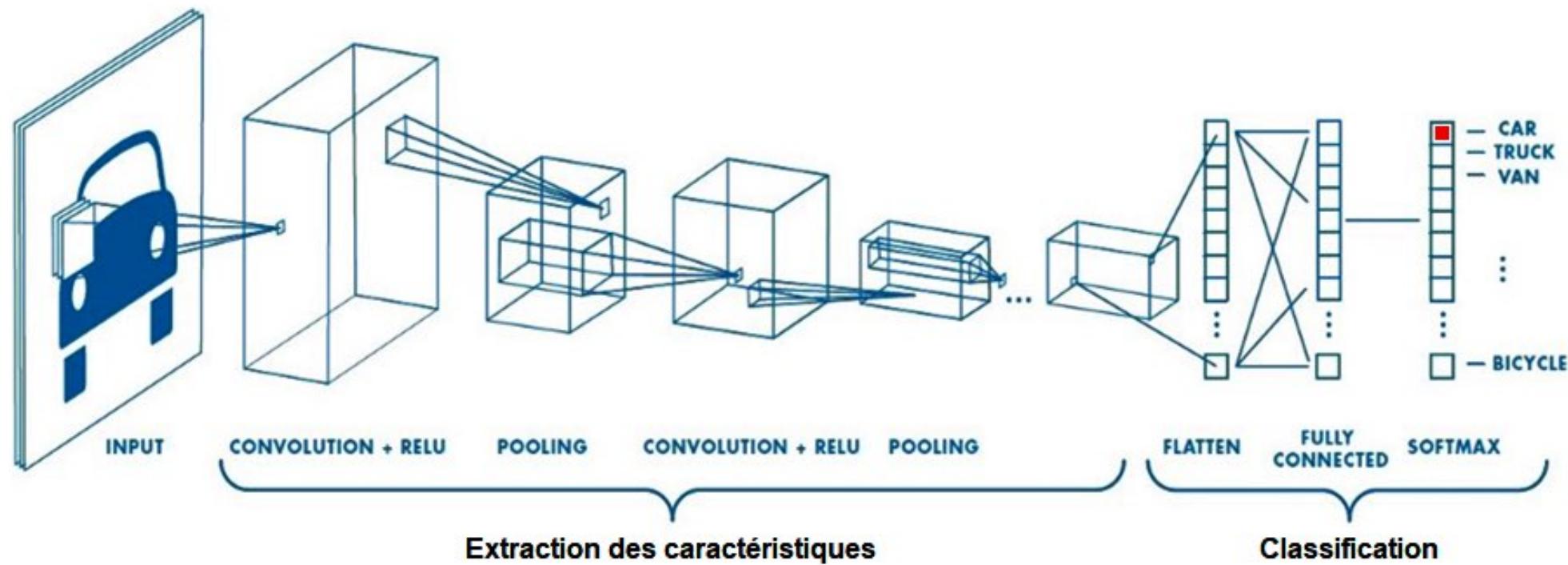
- C'est la moyenne harmonique de précision et de rappel. Cela prend la contribution des deux, donc plus le score F1 est élevé, mieux c'est.

$$\frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 * precision * recall}{precision + recall}$$

- Voyez qu'en raison du produit dans le numérateur si l'on devient bas, le score final F1 diminue considérablement.
- Ainsi, un modèle réussit bien dans le score F1 si les positifs prédis sont en fait positifs (précision) et ne rate pas les positifs et les prédis négatifs (rappel).

Réseaux de Neurones convolutionnels (CNN)

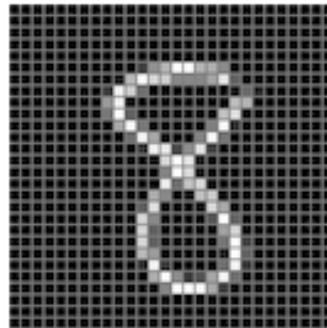
Réseaux de Neurones convolutionnels (CNN)



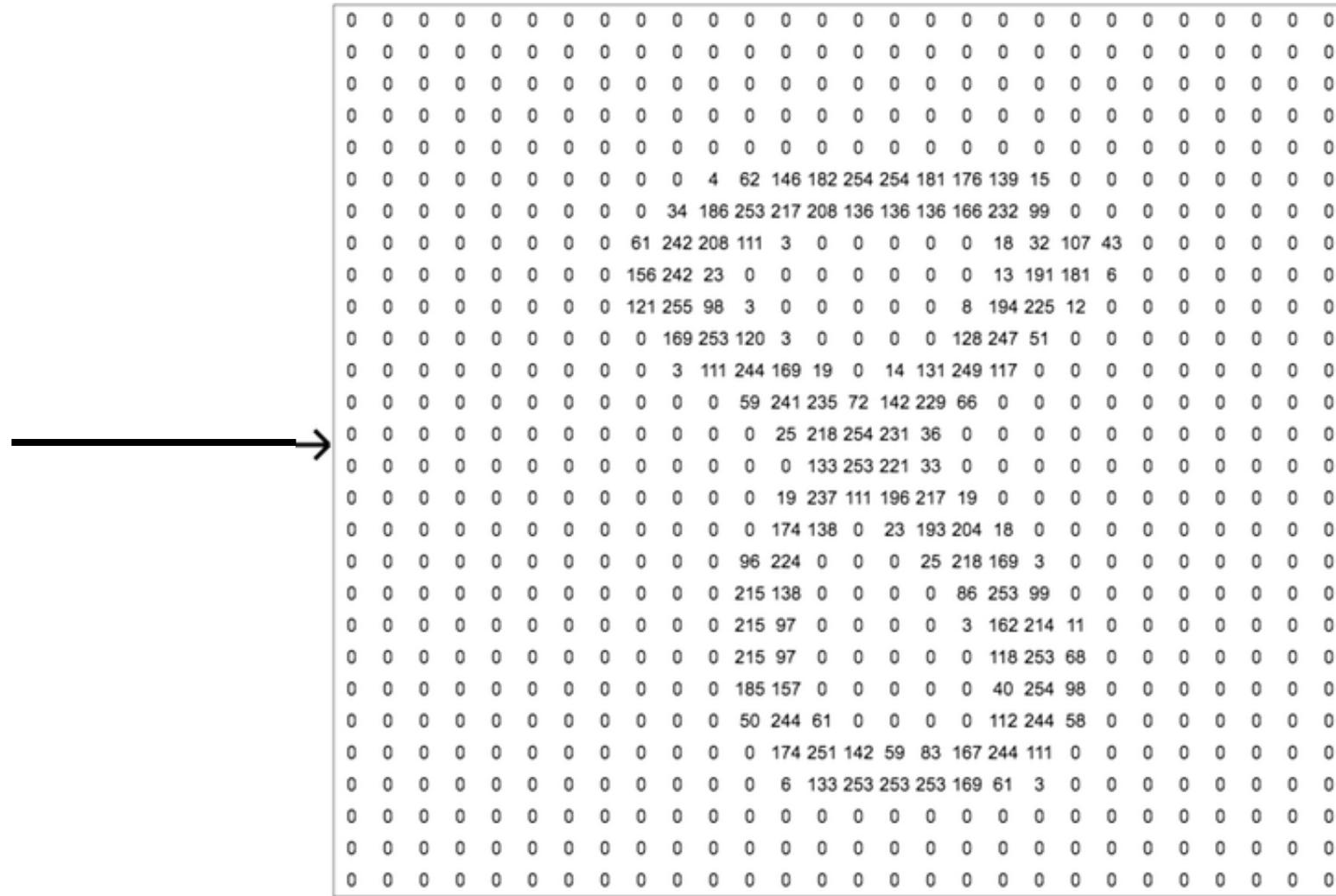
Pourquoi les Réseaux de Neurones Convolutionnels (CNN) ?

- Les bases de données d'analyse comparative de l'apprentissage automatique tels que la base de données MNIST de chiffres manuscrits conviennent à la plupart des formes d'ANN, en raison de sa dimensionnalité d'image relativement petite de seulement 28×28 .
- Avec cette base de données, un seul neurone dans la première couche cachée contiendra 784 poids ($28 \times 28 \times 1$ où 1 sachez que MNIST est normalisé à des valeurs en noir et blanc uniquement), ce qui est gérable pour la plupart des formes d'ANN.

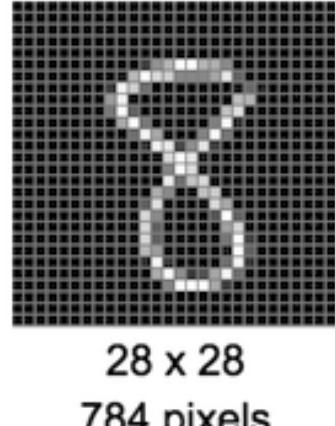
Comment classifier des images?



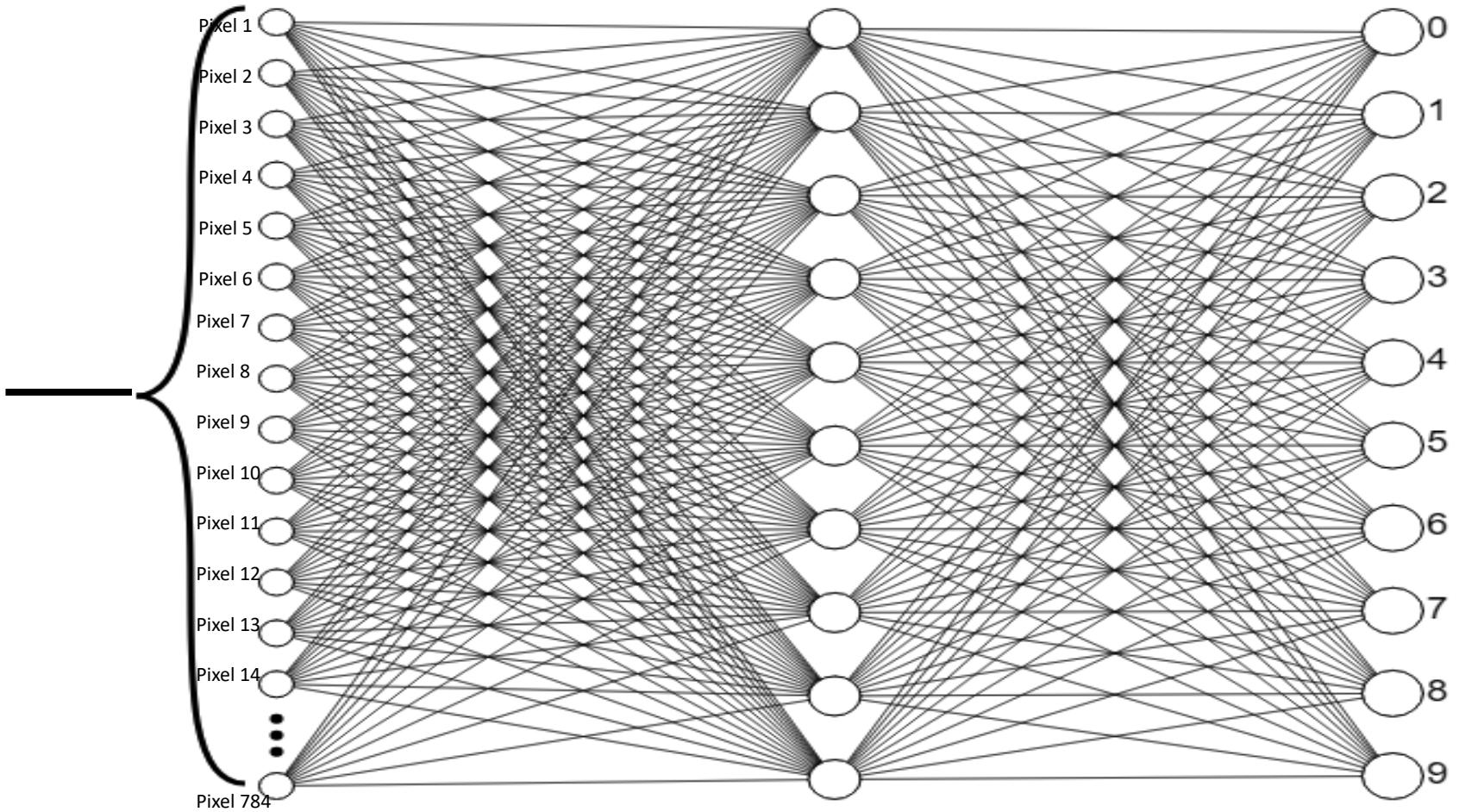
28 x 28
784 pixels



Comment classifier des images?



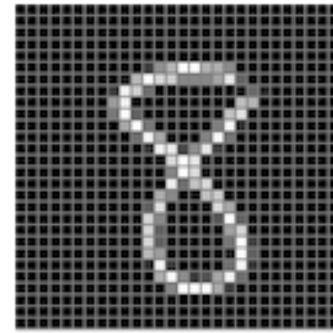
linéarisation



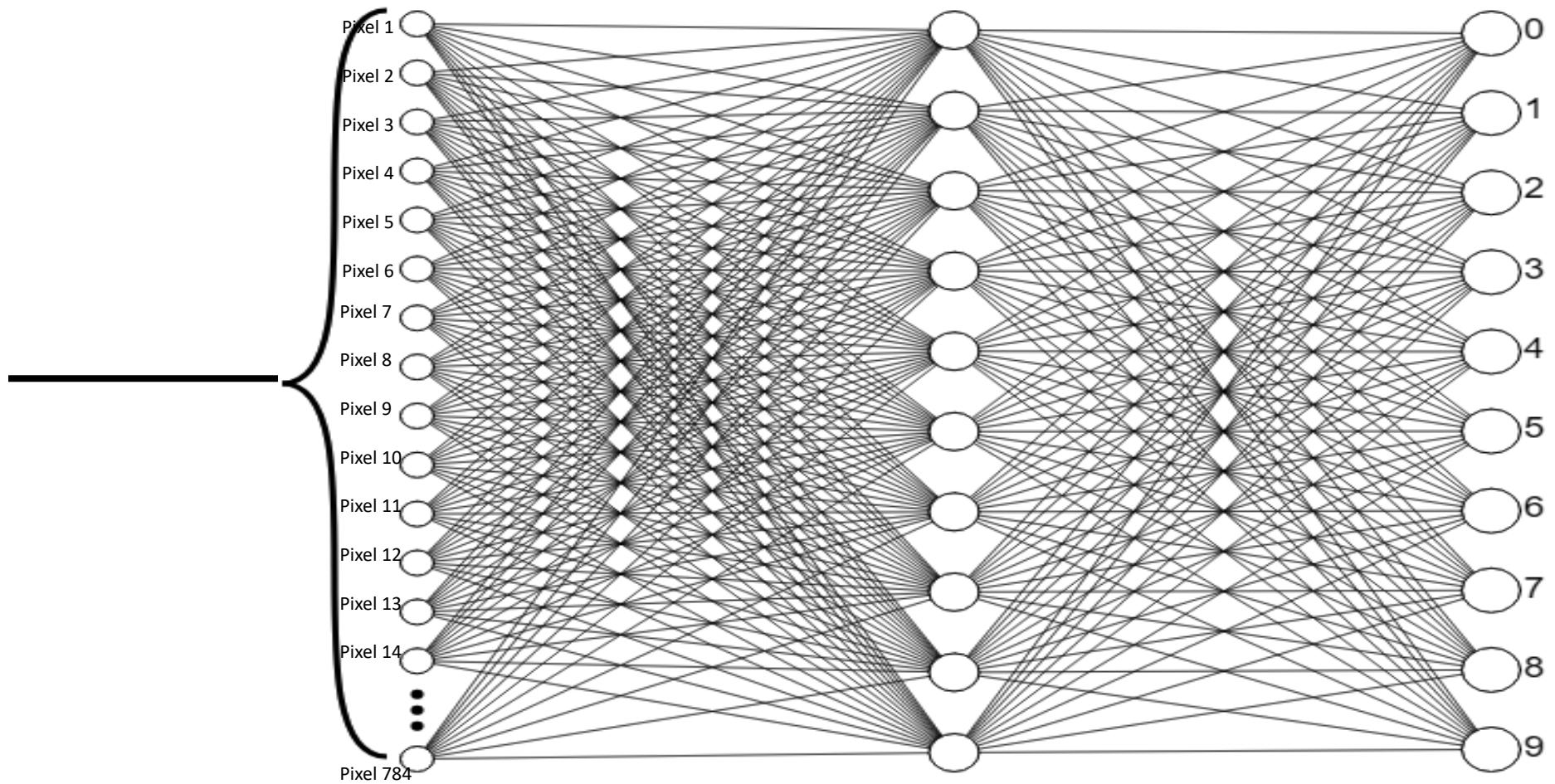
10 neurones

https://ml4a.github.io/ml4a/neural_networks/

Beaucoup de paramètres (7850 paramètres dans la couche 1)



28 x 28
784 pixels



10 neurones

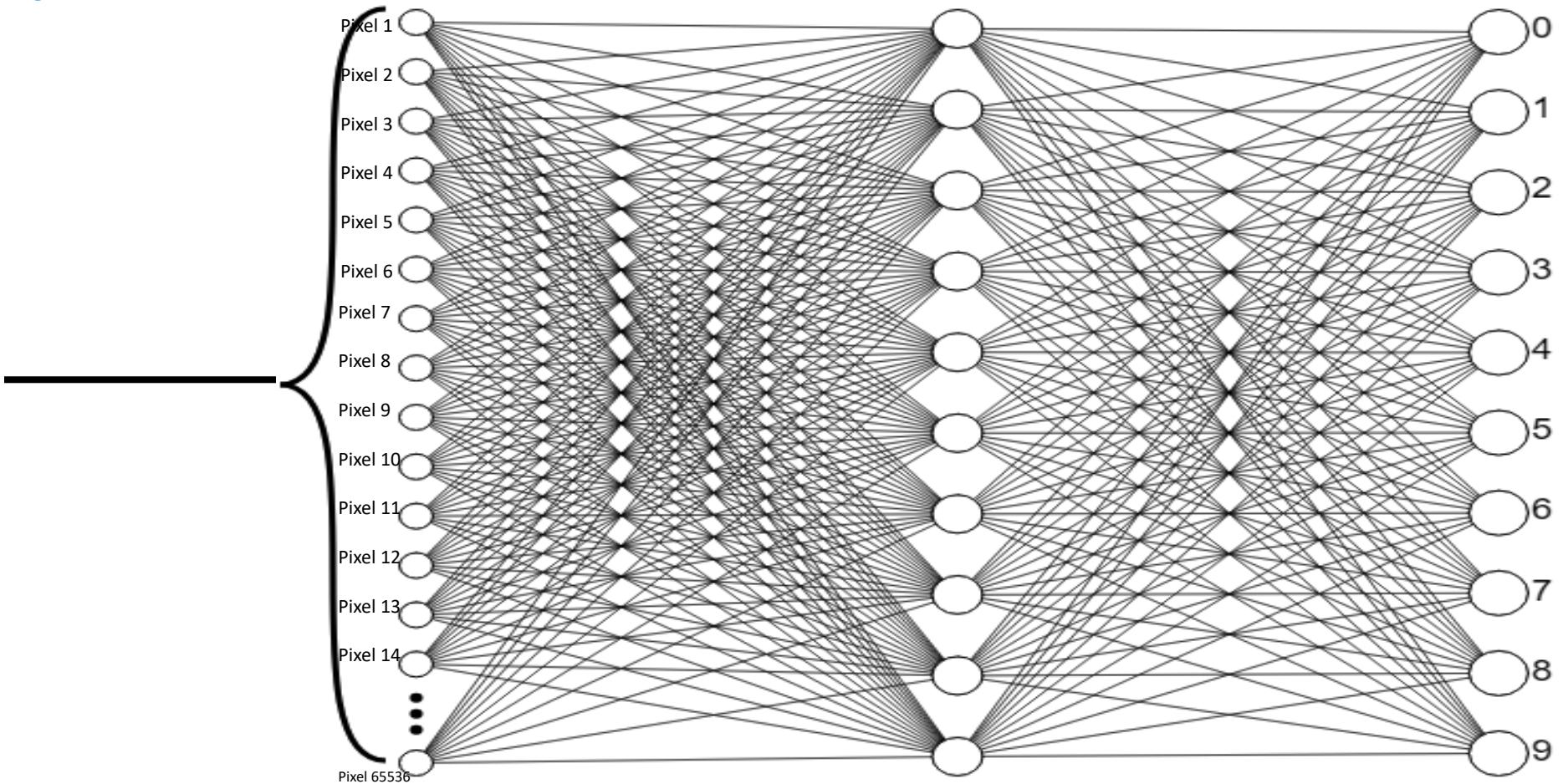
https://ml4a.github.io/ml4a/neural_networks/

Beaucoup trop de paramètres (655,370 paramètres dans la couche 1)



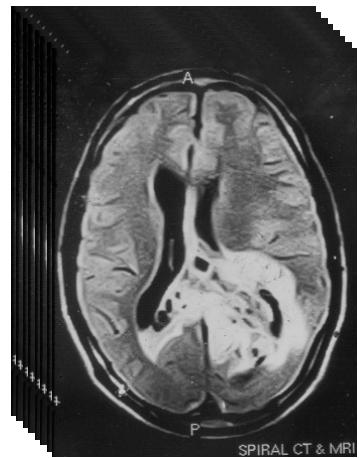
256x256

Image médicale (IRM de cerveau)



https://ml4a.github.io/ml4a/neural_networks/

Beaucoup TROP de paramètres (160M de paramètres dans la couche 1)



256x256x256

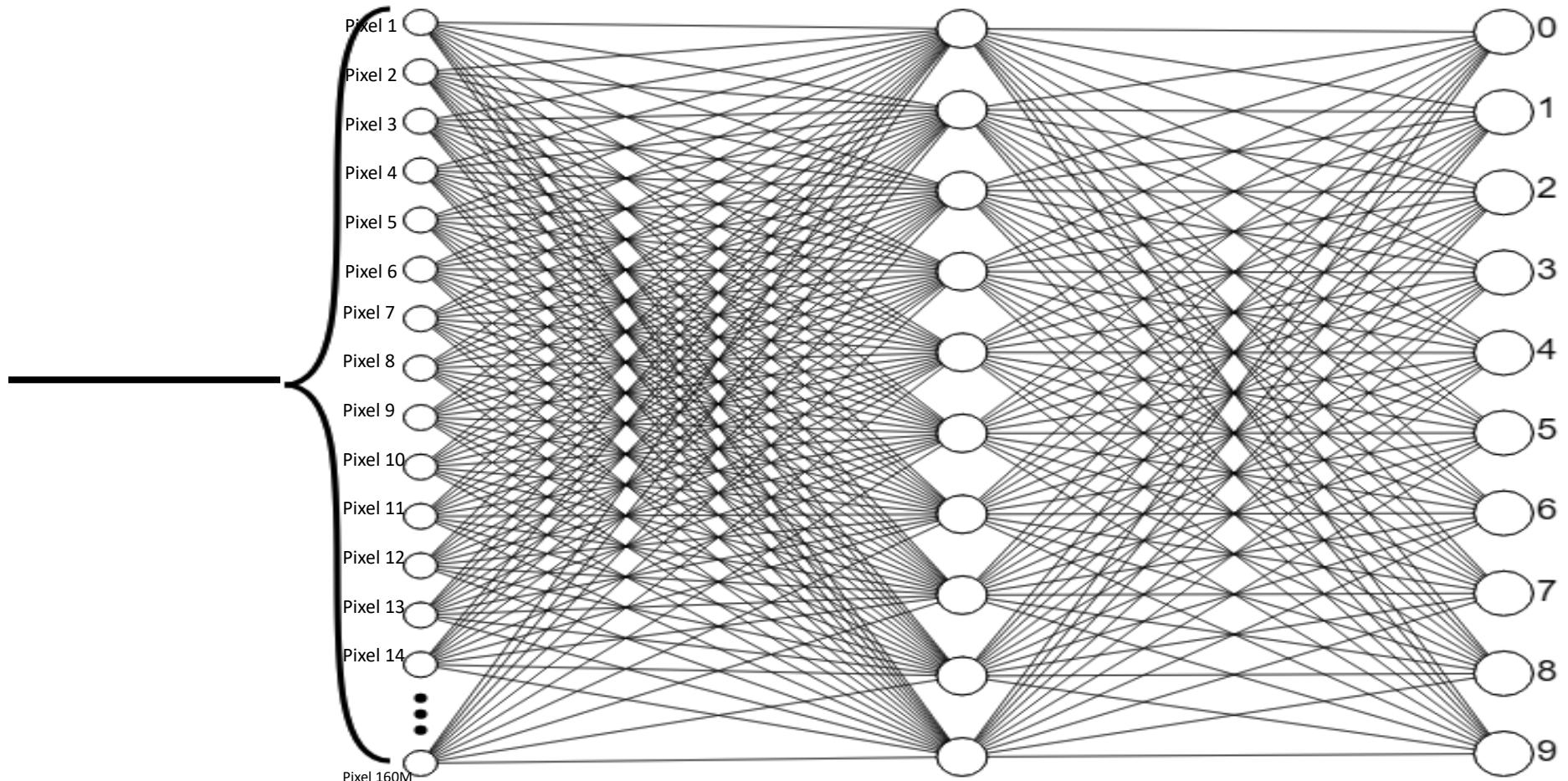
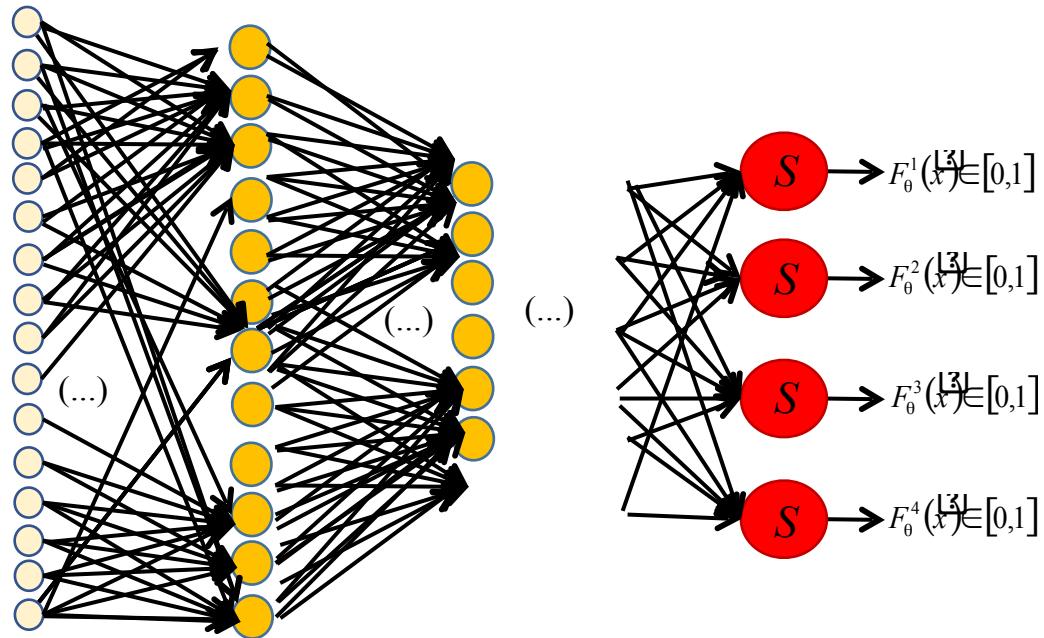


Image médicale 3D (IRM de cerveau)

https://ml4a.github.io/ml4a/neural_networks/

Problème:

Les **couches pleinement connectées (fully-connected layers)**
sont problématiques lorsque le **nombre de neurones** est élevé.

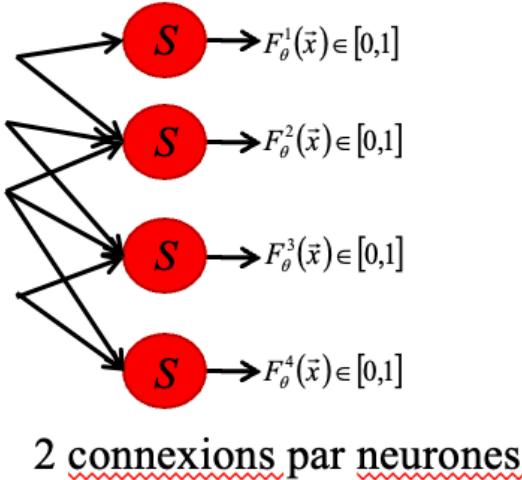
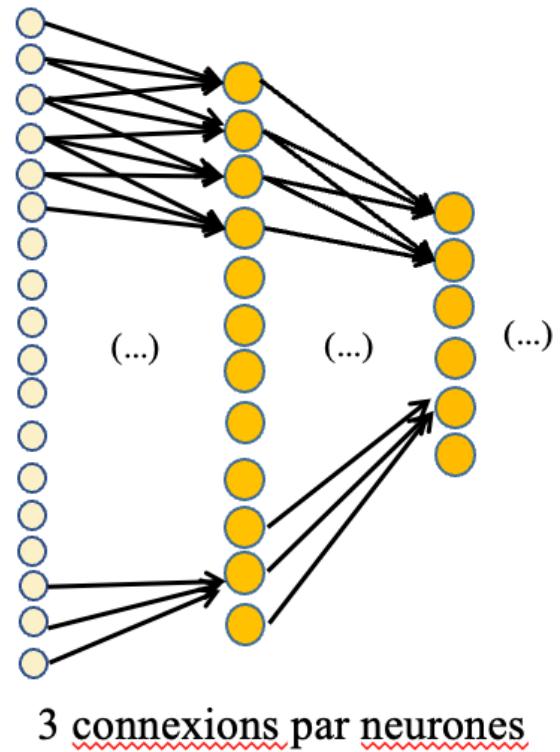


150-D en entrée avec 150 neurones dans la 1ère couche => 22,500 paramètres dans la couche cachée !!

Comment réduire le nombre de connections?

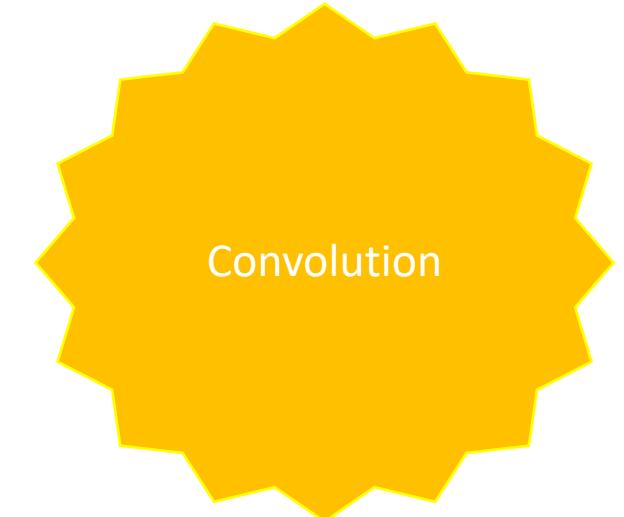
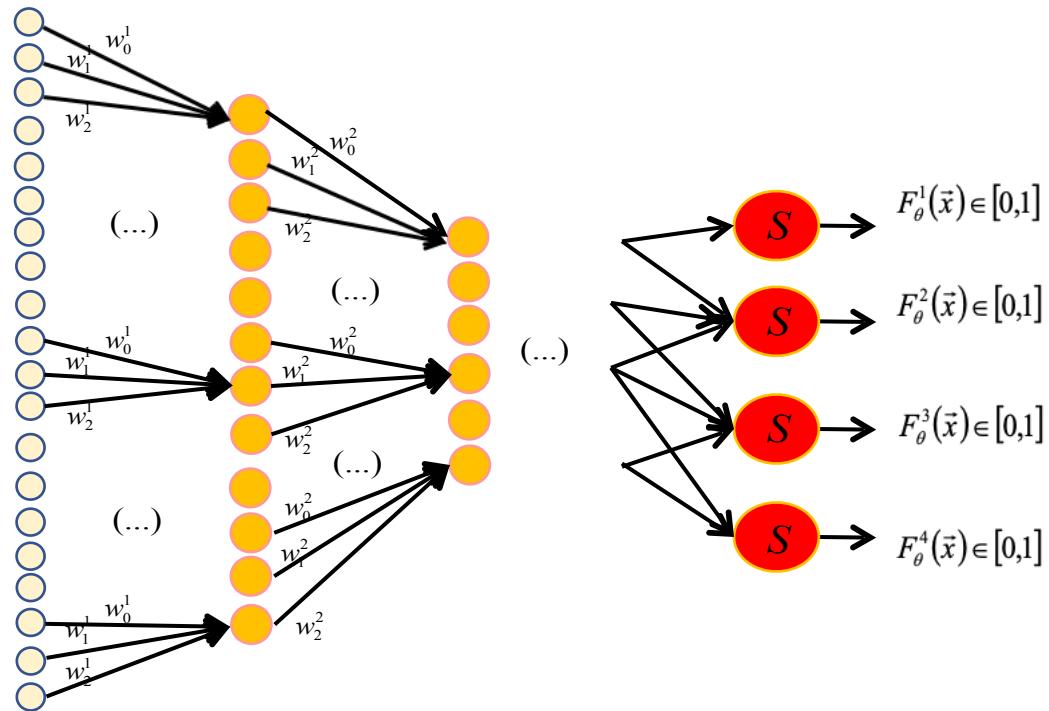


Solution : connexions partielles



150-D en entrée avec 148 neurones dans la 1ère couche => 444 paramètres dans la première couche!!

Paramètres partagés : les neurones de la couche 1 partagent les mêmes poids



150-D en entrée avec 148 neurones dans la 1ère couche => 3 paramètres dans la couche d'entrée!!

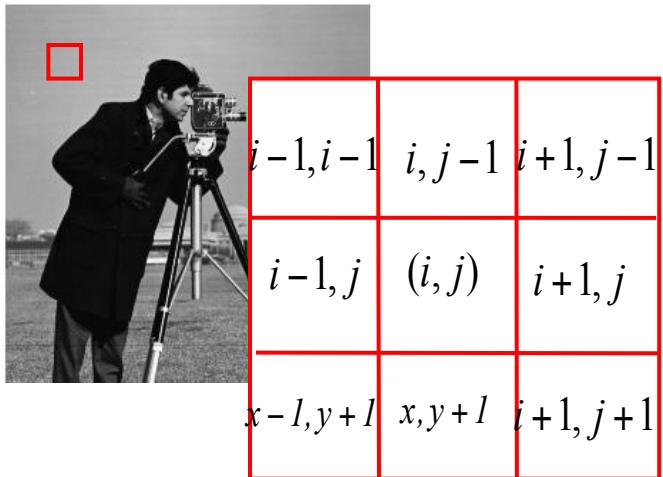
Faible nombre de paramètres = on peut augmenter la profondeur!

Filtage 2D

(sans flip de filtre)

$$(x * W)(i, j) = \sum_u \sum_v f(i + u, j + v)W(u, v)$$

$x(i, j)$



$W(u, v)$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$$\begin{aligned}(x * W)(i, j) = & w_1 x(i - 1, j - 1) + w_2 x(i, j - 1) + w_3 x(i + 1, j - 1) \\& + w_4 x(i - 1, j) + w_5 x(i, j) + w_6 x(i + 1, j) \\& + w_7 x(i - 1, j + 1) + w_8 x(i, j + 1) + w_9 x(i + 1, j + 1)\end{aligned}$$

Couche de convolution : Exemple du produit de convolution

7	2	3	3	8
4	5	3	8	4
3	3	2	8	4
2	8	7	2	7
5	4	4	5	4

*

1	0	-1
1	0	-1
1	0	-1

=

6		

$$7 \times 1 + 4 \times 1 + 3 \times 1 + \\ 2 \times 0 + 5 \times 0 + 3 \times 0 + \\ 3 \times -1 + 3 \times -1 + 2 \times -1 \\ = 6$$

Convolution

Simple convolution d'une matrice (5x5) avec un noyau (3x3)

Couche de convolution : Exemple du produit de convolution dans RGB

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

-25				...
				...
				...
				...
...

Output

Bias = 1

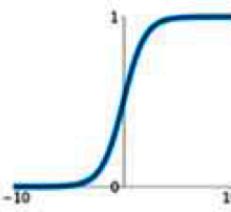
Couche de convolution : Fonction d'activation

Chaque couche convulsive contient une fonction d'activation

S'assure que tous les neurons sont activés

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

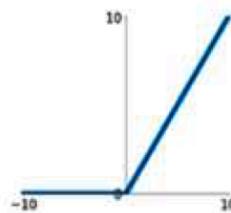


.....
tanh(x)



ReLU

$$\max(0, x)$$



Leaky ReLU
 $\max(0.1x, x)$

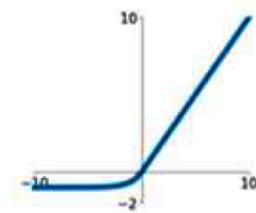


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

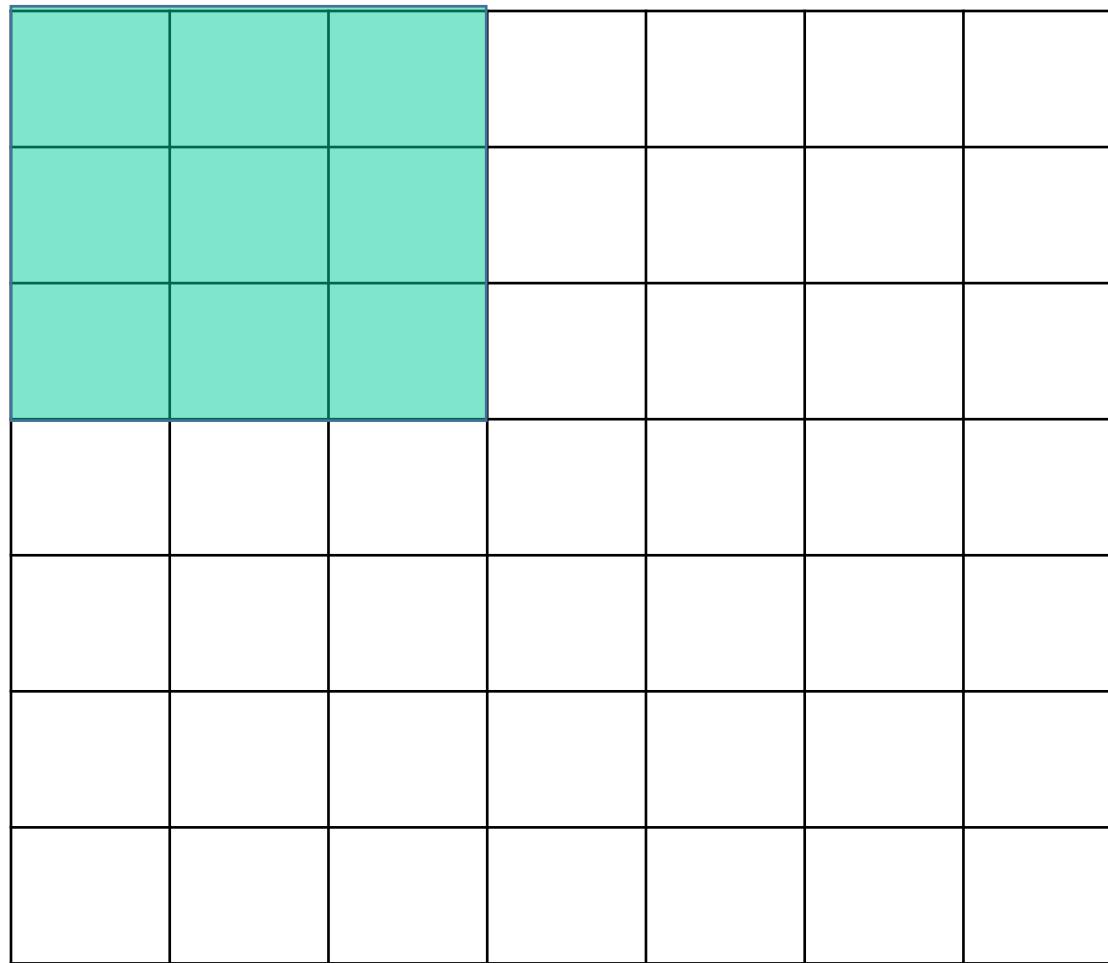
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Réduit le problème de la
disparition du gradient

Convolution 2D : Stride

7

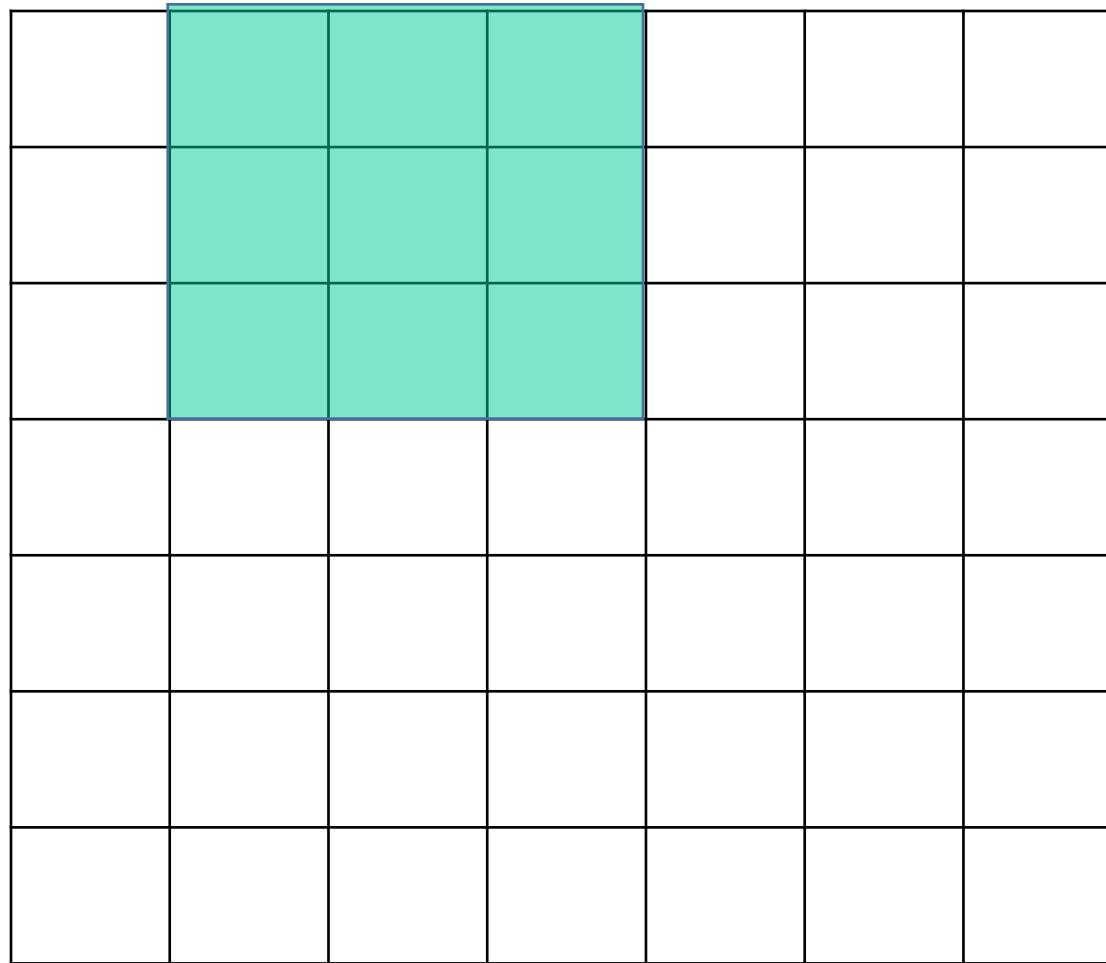


Filtre = 3x3
Stride = 1

7

Convolution 2D : Stride

7

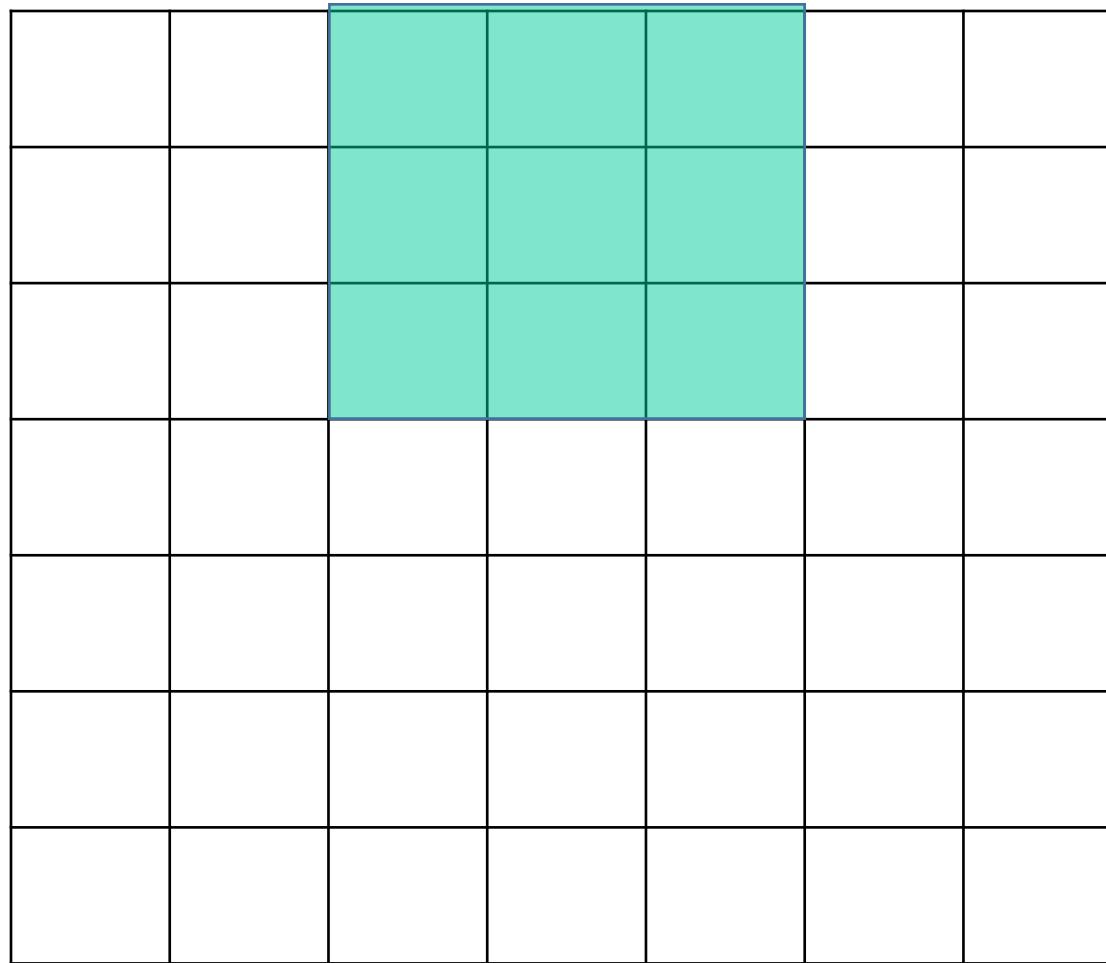


Filtre = 3x3
Stride = 1

7

Convolution 2D : Stride

7

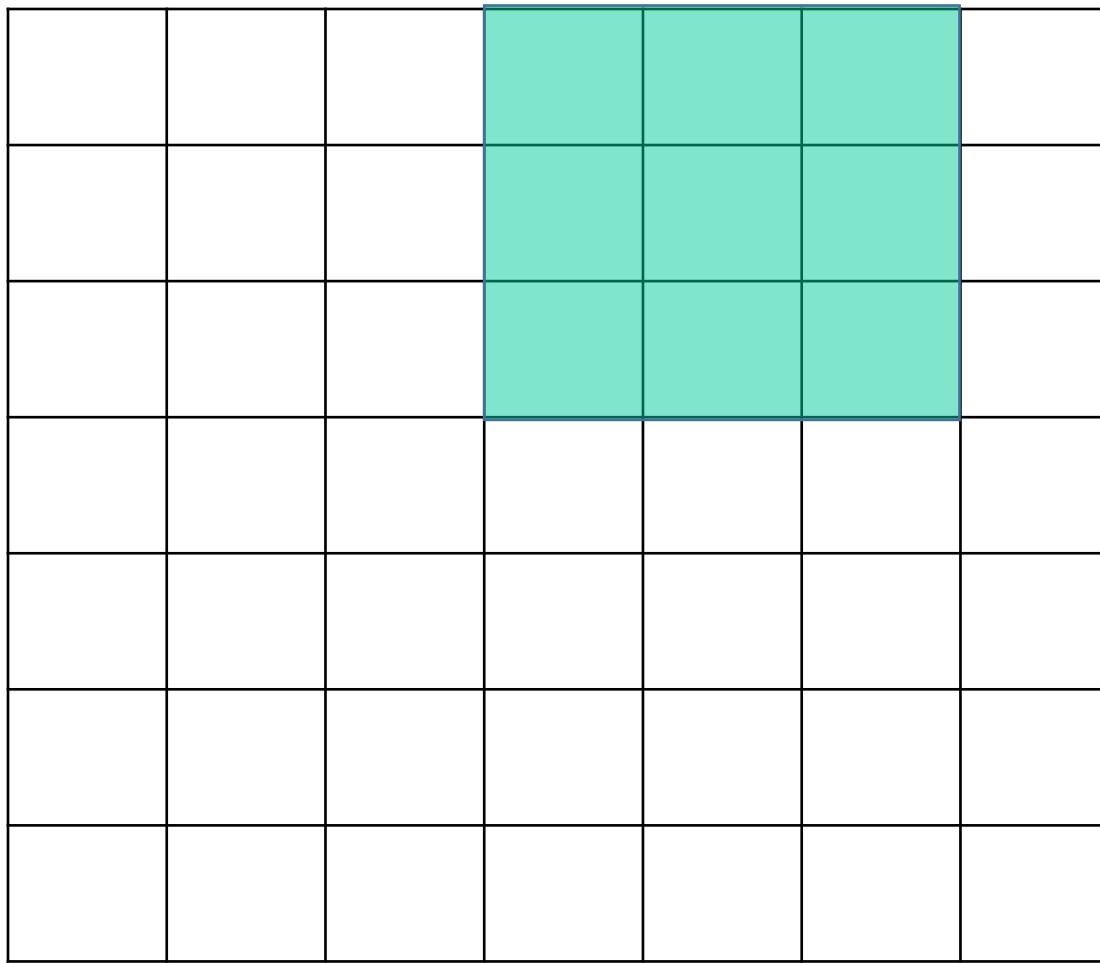


Filtre = 3x3
Stride = 1

7

Convolution 2D : Stride

7

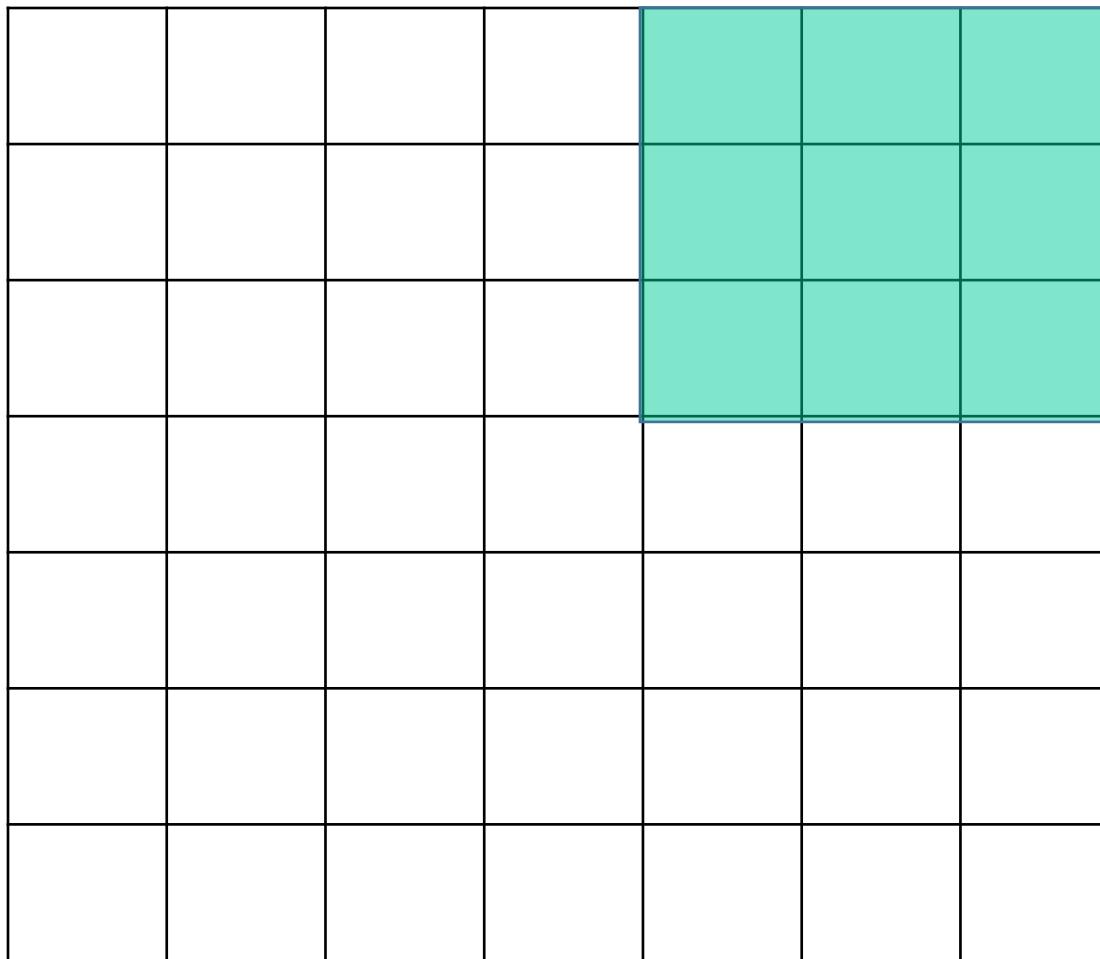


Filtre = 3x3
Stride = 1

7

Convolution 2D

7



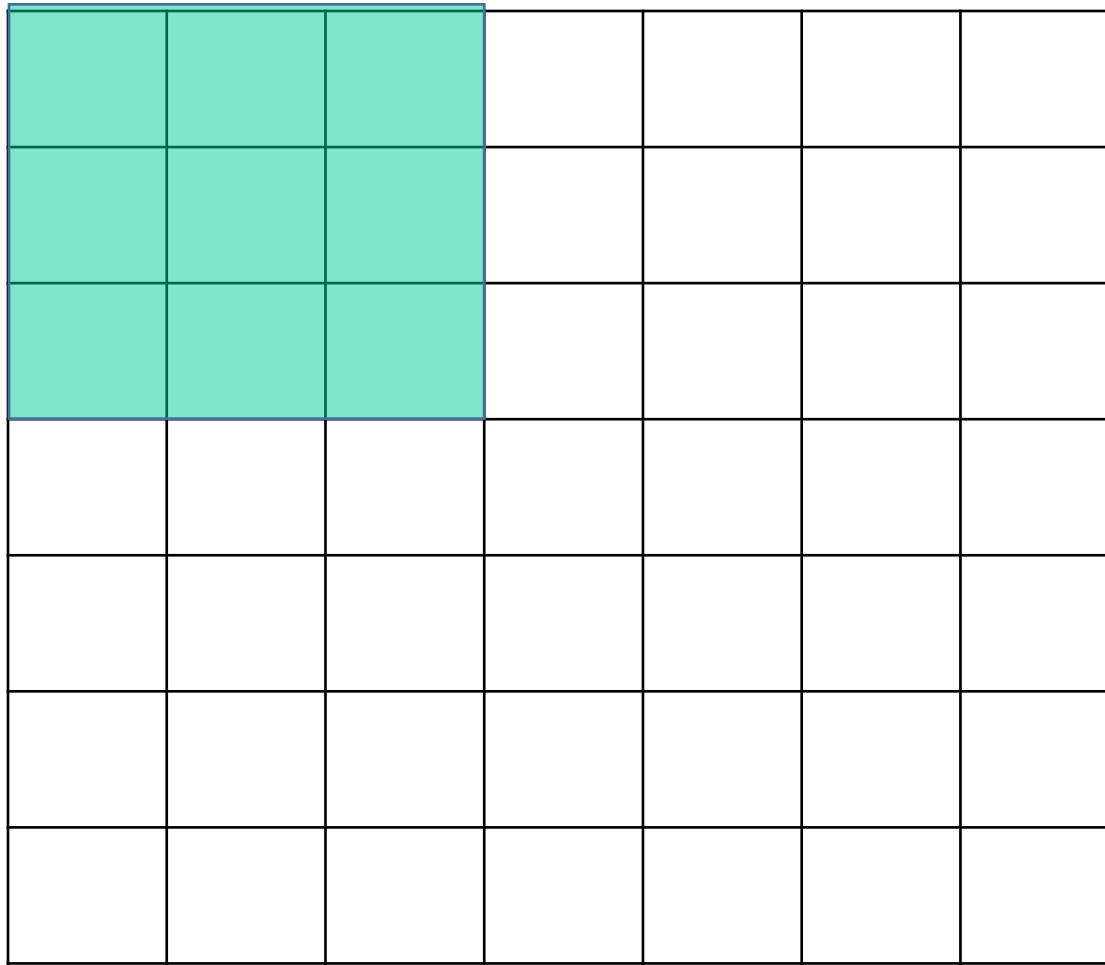
Filtre = 3x3
Stride = 1

7

Taille de la carte d'activation
(pour stride 1) = **5x5**

Convolution 2D

7

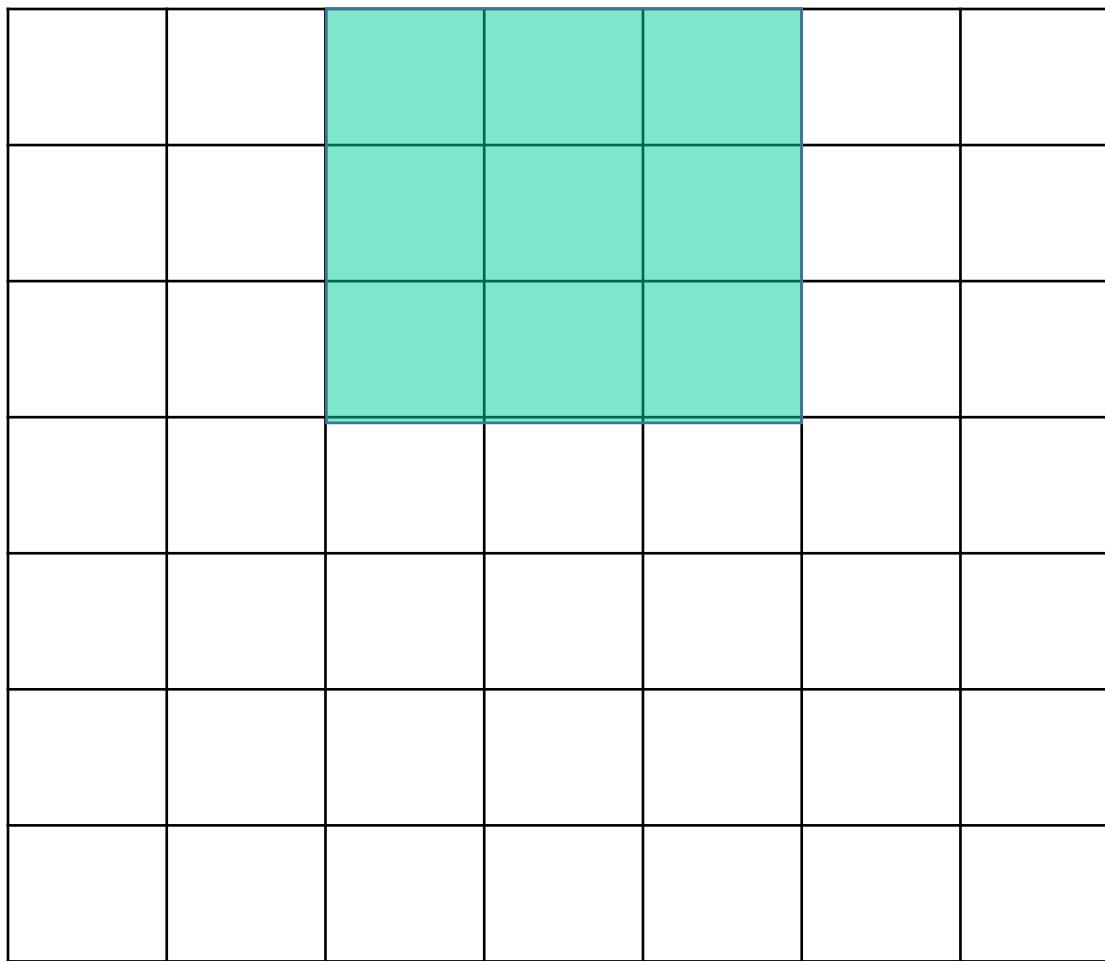


Filtre = 3x3
Stride = 2

7

Convolution 2D

7

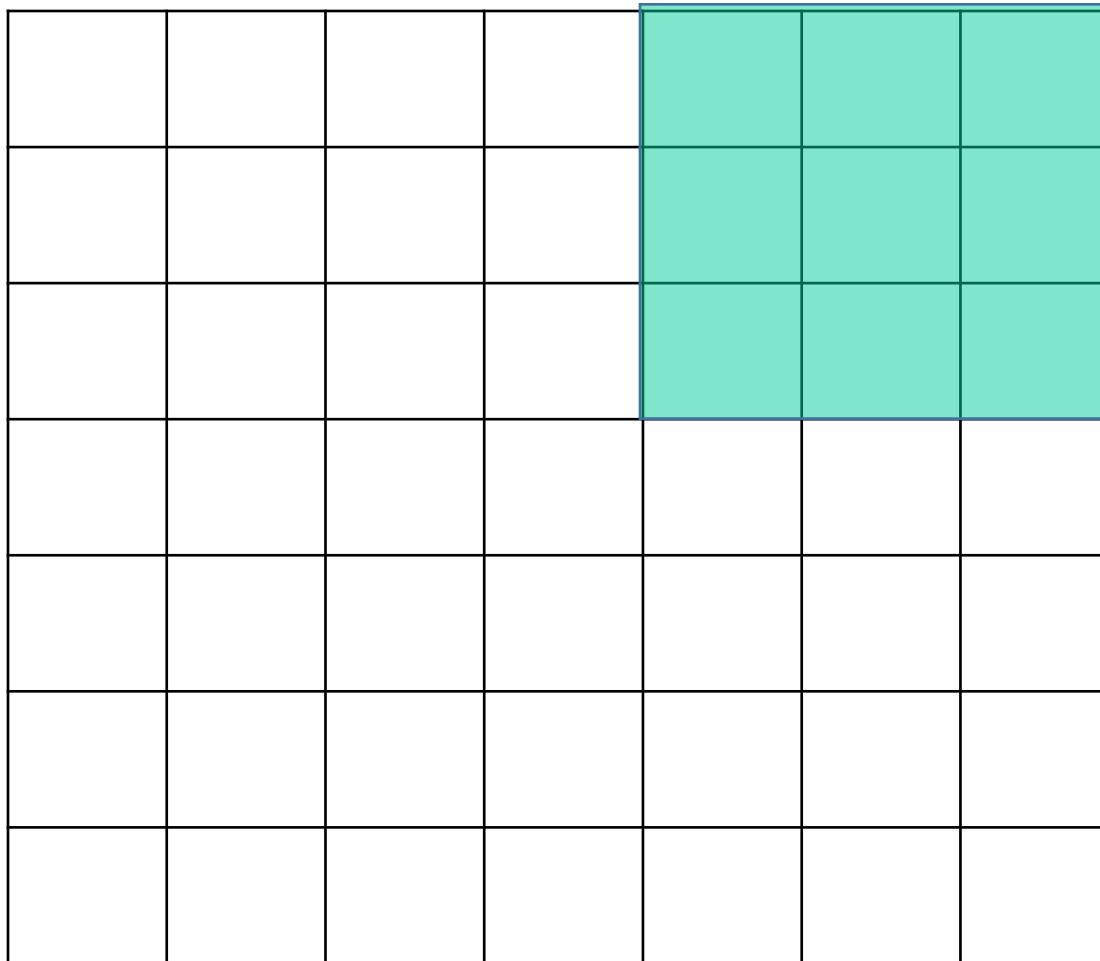


Filtre = 3x3
Stride = 2

7

Convolution 2D

7



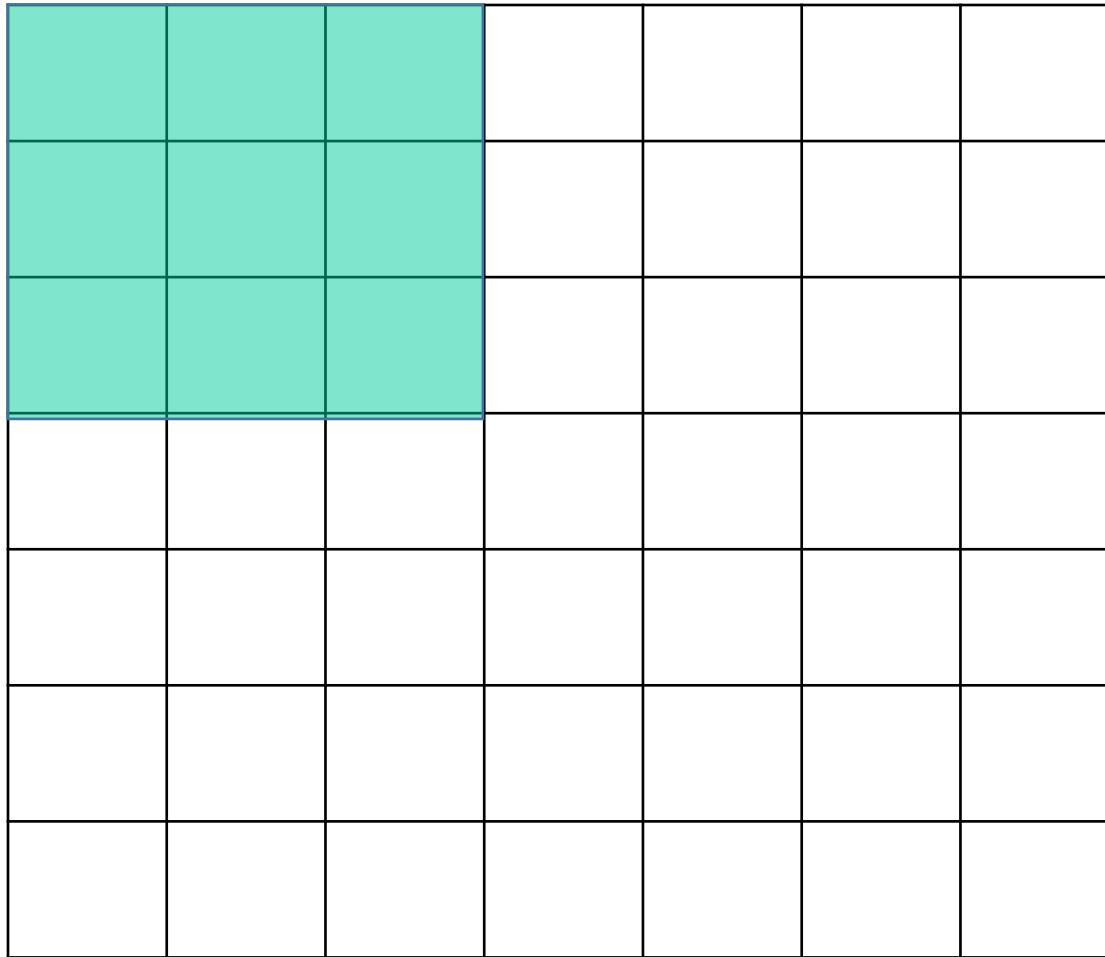
Filtre = 3x3
Stride = 2

7

Taille de la carte d'activation
(pour stride 2) = **3x3**

Convolution 2D

7

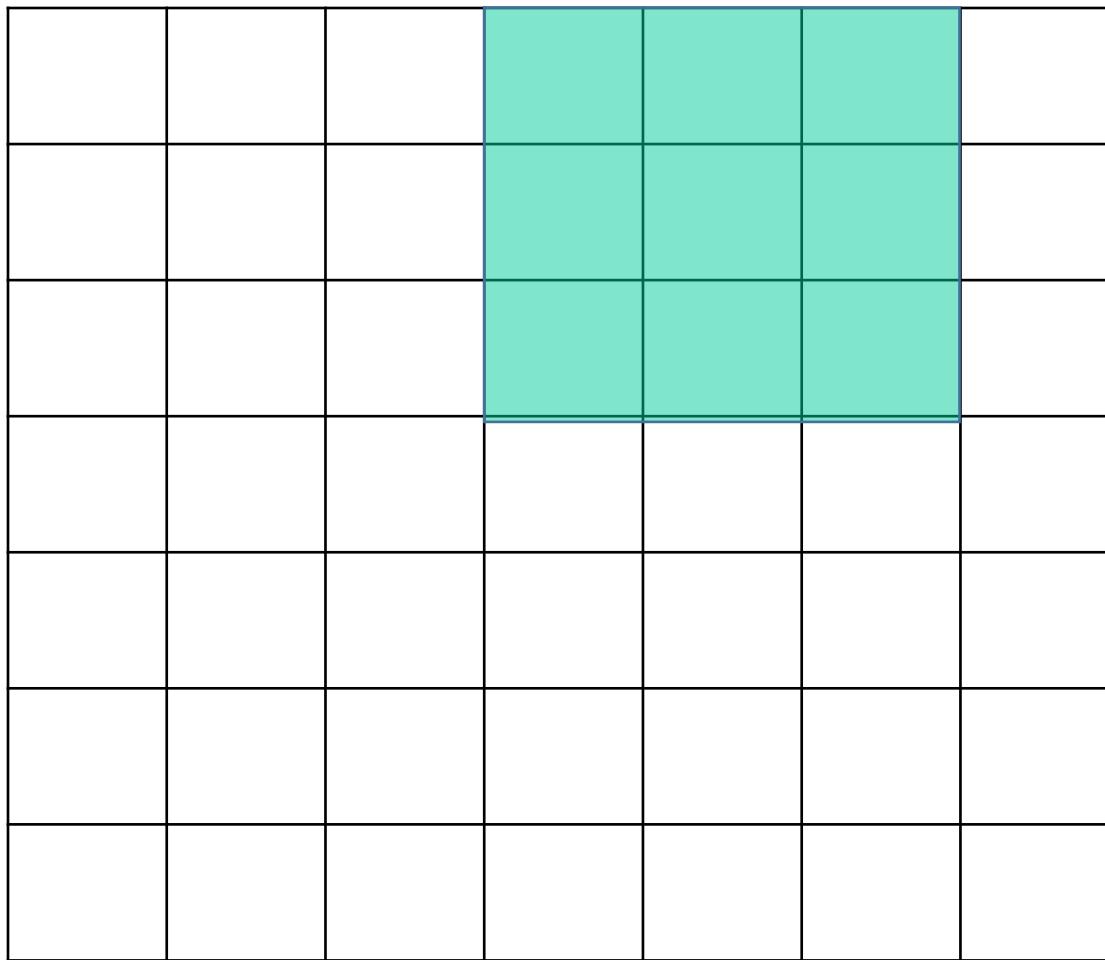


Filtre = 3x3
Stride = 3

7

Convolution 2D

7

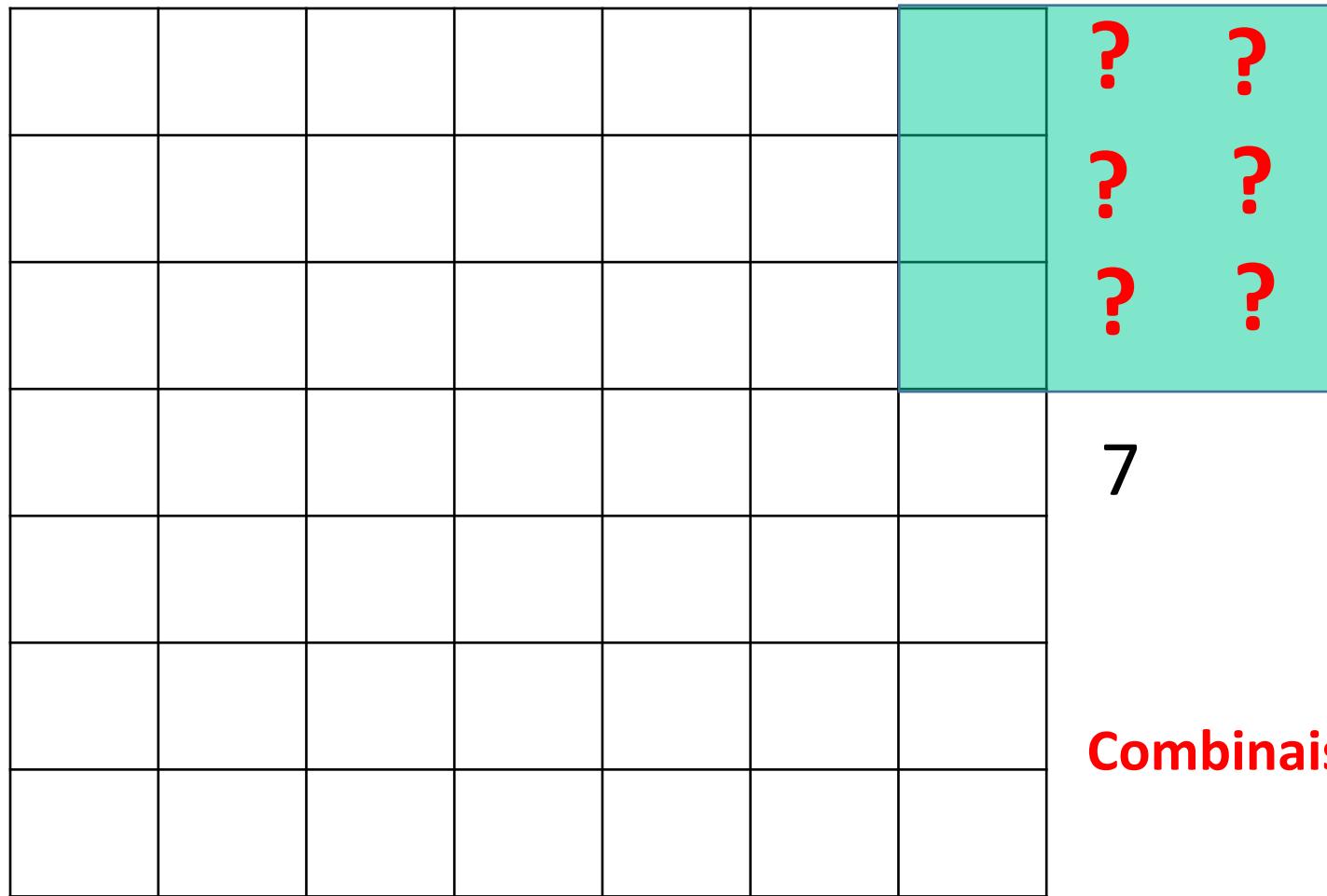


Filtre = 3x3
Stride = 3

7

Convolution 2D

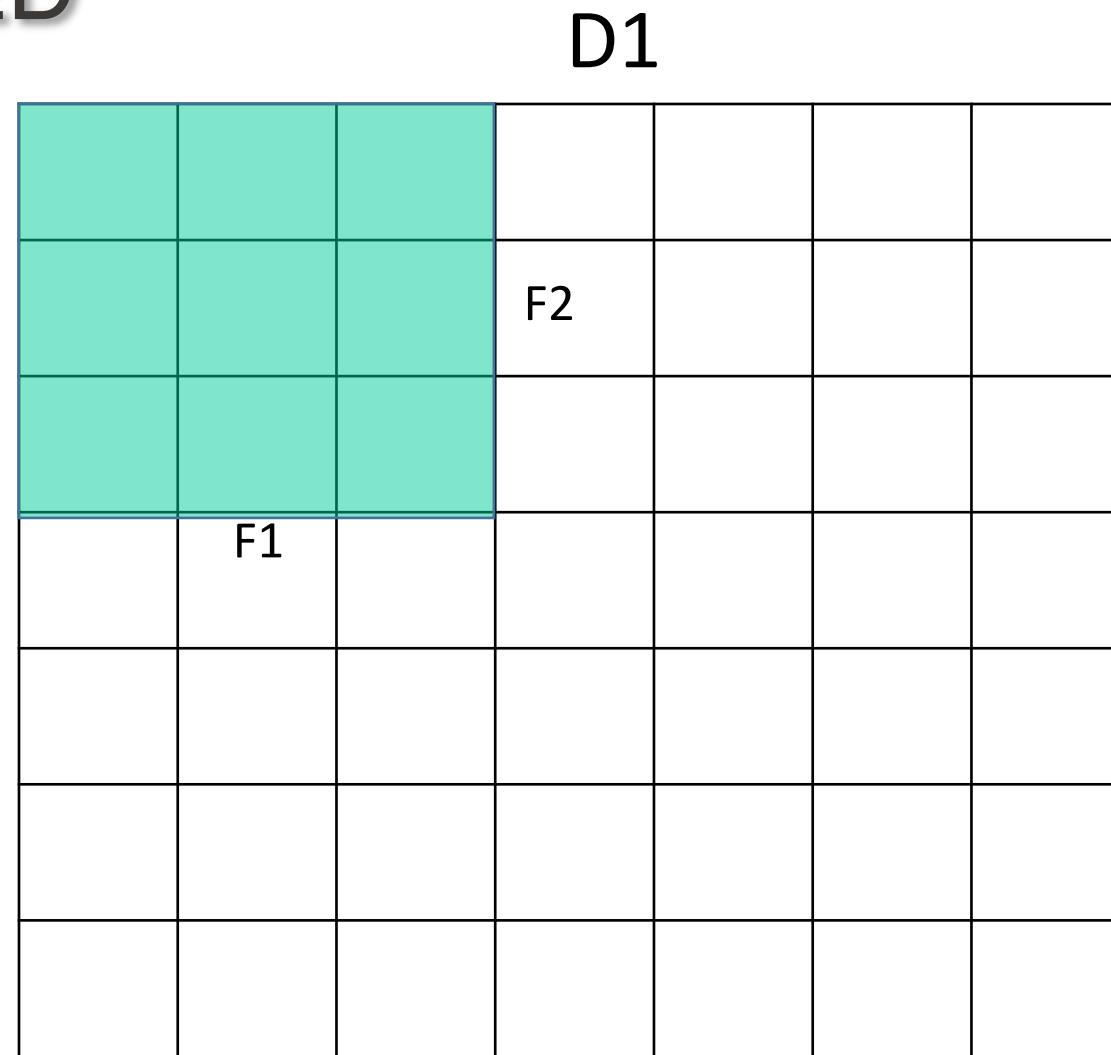
7



Filtre = 3x3
Stride = 3

Combinaison D-F-S invalide!

Convolution 2D



Doit être un entier

D2

Taille de la carte d'activation :

$((D1-F1)/S) + 1 \times$

$((D2-F2)/S) + 1$

Parfois on souhaite que le **nombre de neurones** dans la carte d'activation soit **le même** que la couche précédente

Comment gérer les bords?

?	10	20	-30	40	50
x	x	x			

.1	.2	.3
----	----	----

Option 1 : Ajout de zéros (« *zero padding* » remplacer ? par 0)

f(u)
0 10 20 -30 40 -50 0

(f*W)(u)
8 -4 8 -10 -6

Option 2 : Réflexion (« *reflexion padding* »)

f(u)
20 10 20 -30 40 -50 40

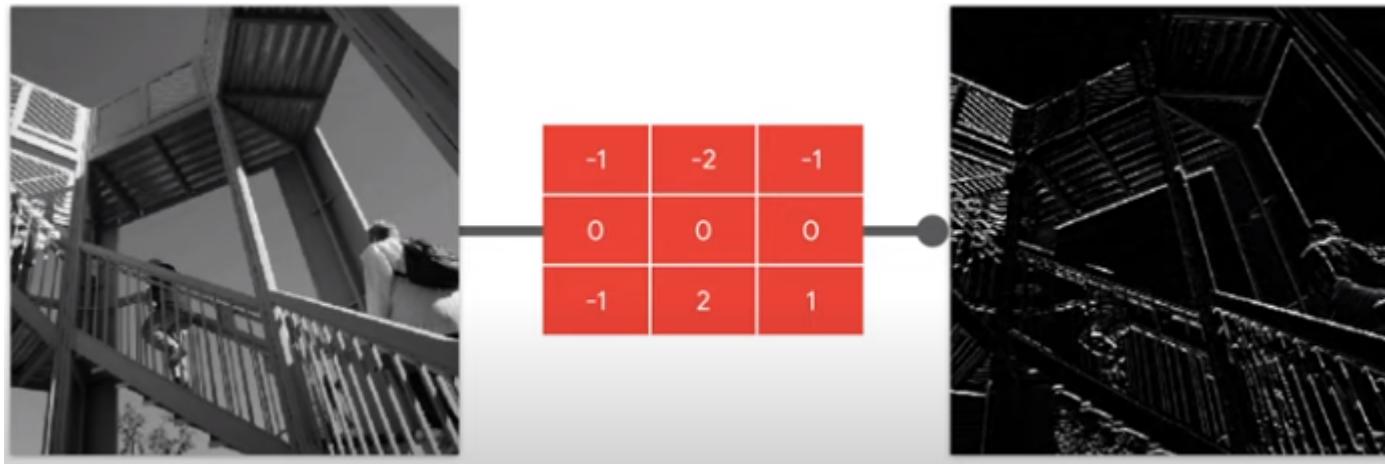
(f*W)(u)
10 -4 8 -10 2

Option 3 : Étirement (« *stretching padding* »)

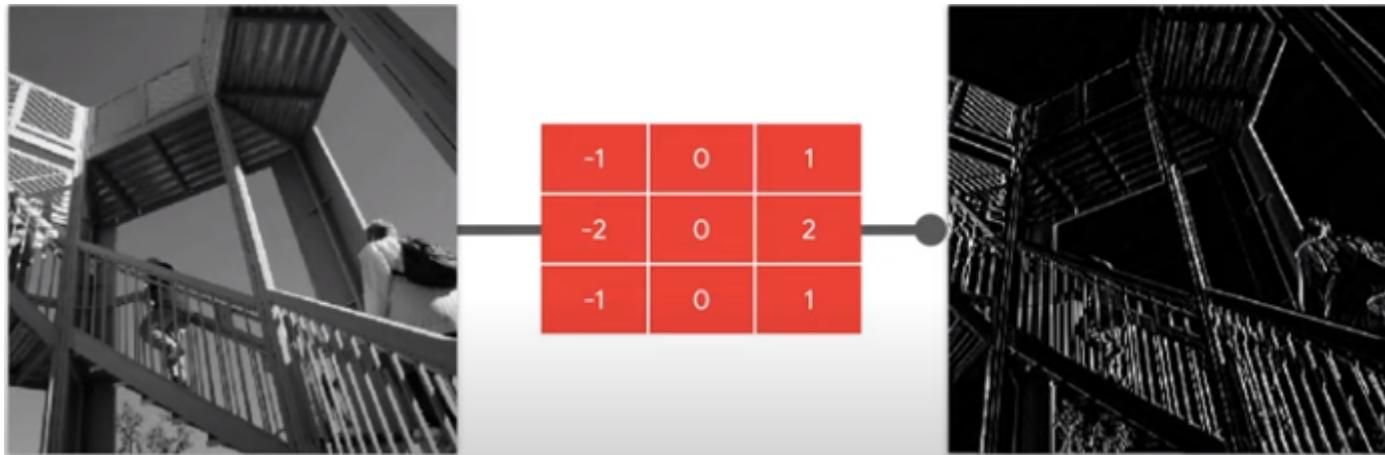
f(u)
10 10 20 -30 40 -50 -50

(f*W)(u)
9 -4 8 -10 -21

Chaque filtre a un travail; certains d'entre eux détectent les bords certains d'entre eux ne détectent que les lignes horizontales. La tâche du filtre devient plus complexe dans les couches profondes du réseau convolutionnel.

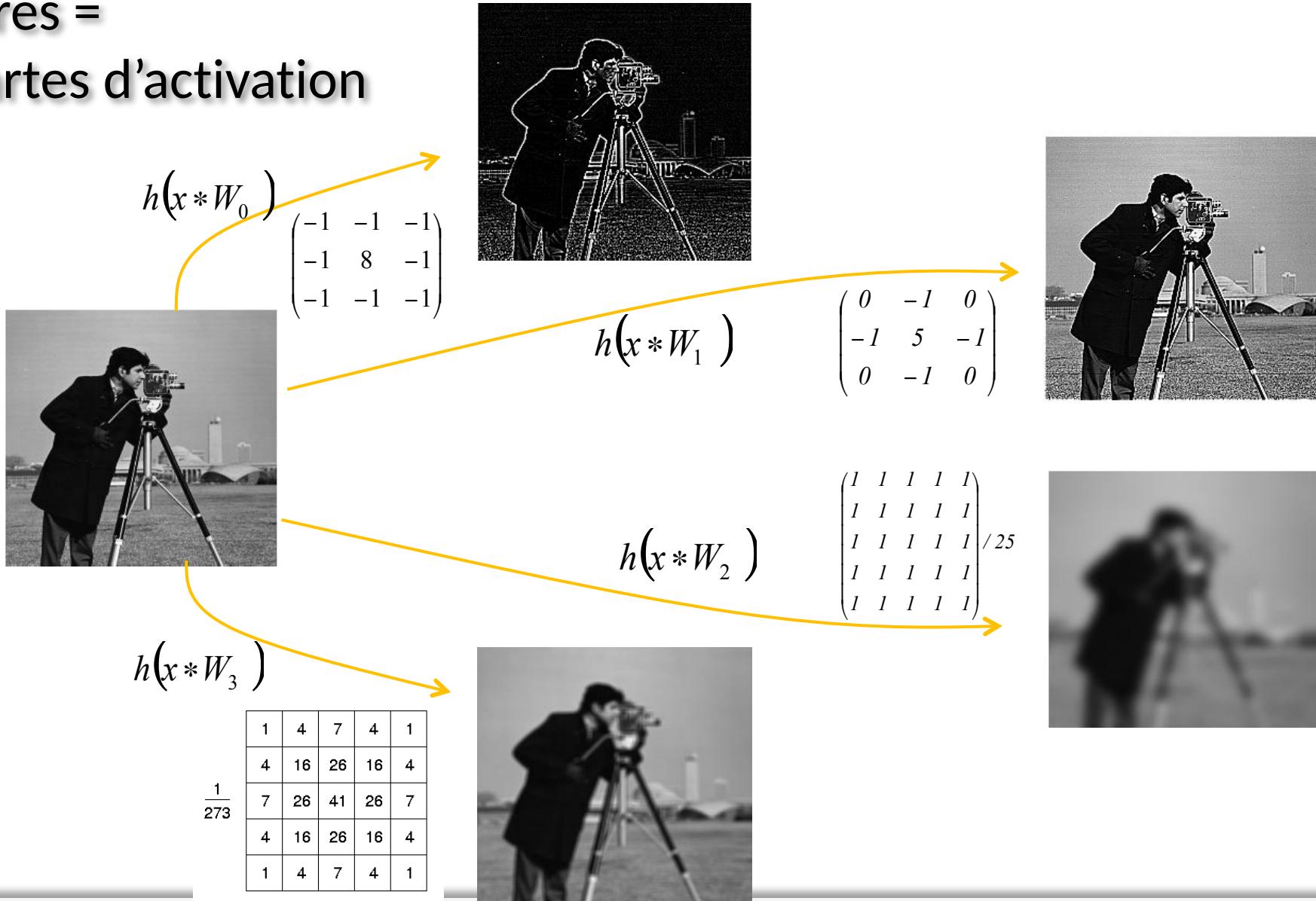


Par exemple, la première couche du réseau peut détecter uniquement les lignes horizontales

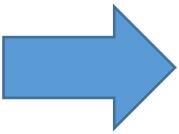


Tandis que ce filtre supprime tous sauf les lignes verticales

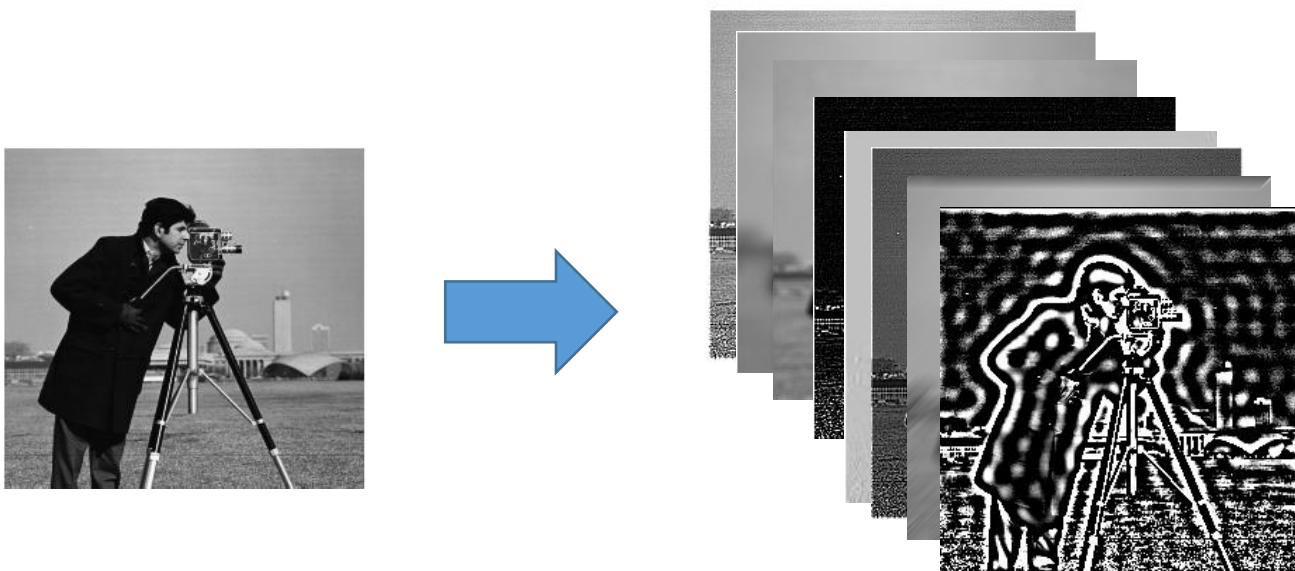
Différents filtres = différentes cartes d'activation



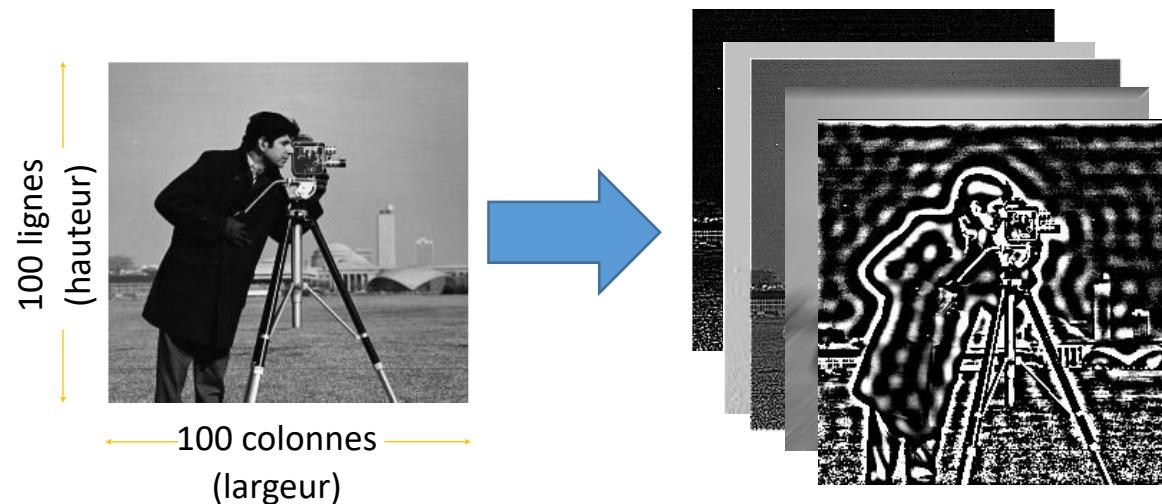
4 filtres = Couche convolutionnelle avec 4 cartes d'activation



K filtres = Couche convulsive avec K cartes d'activation

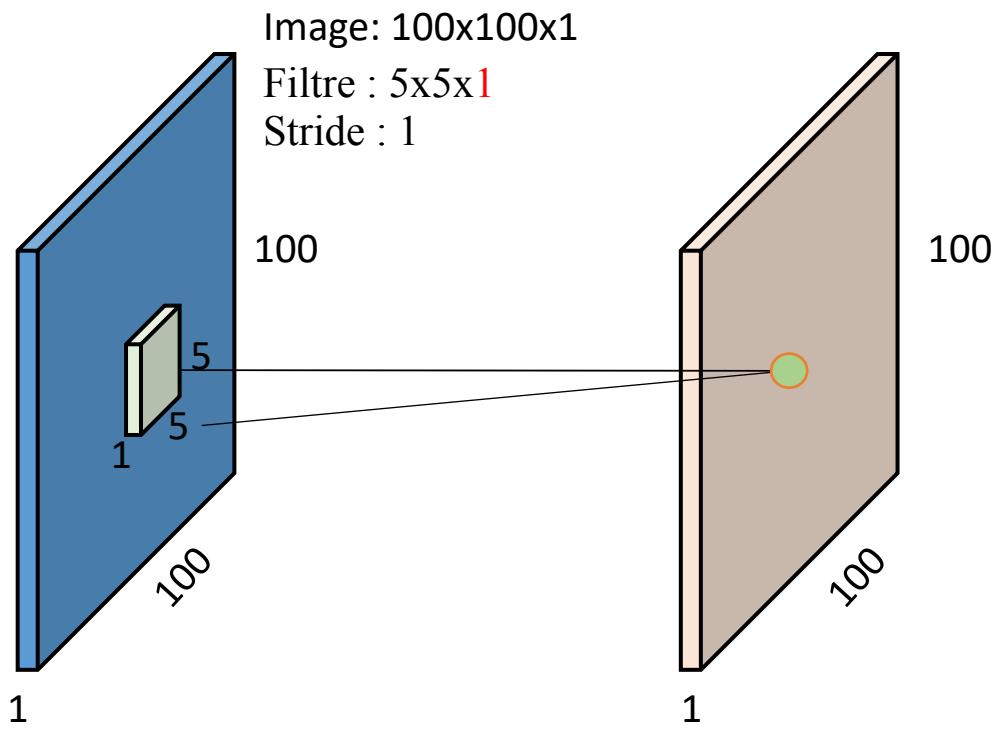


Ex.: taille de filtre : 5x5, 5 cartes d'activation, convolution « same »

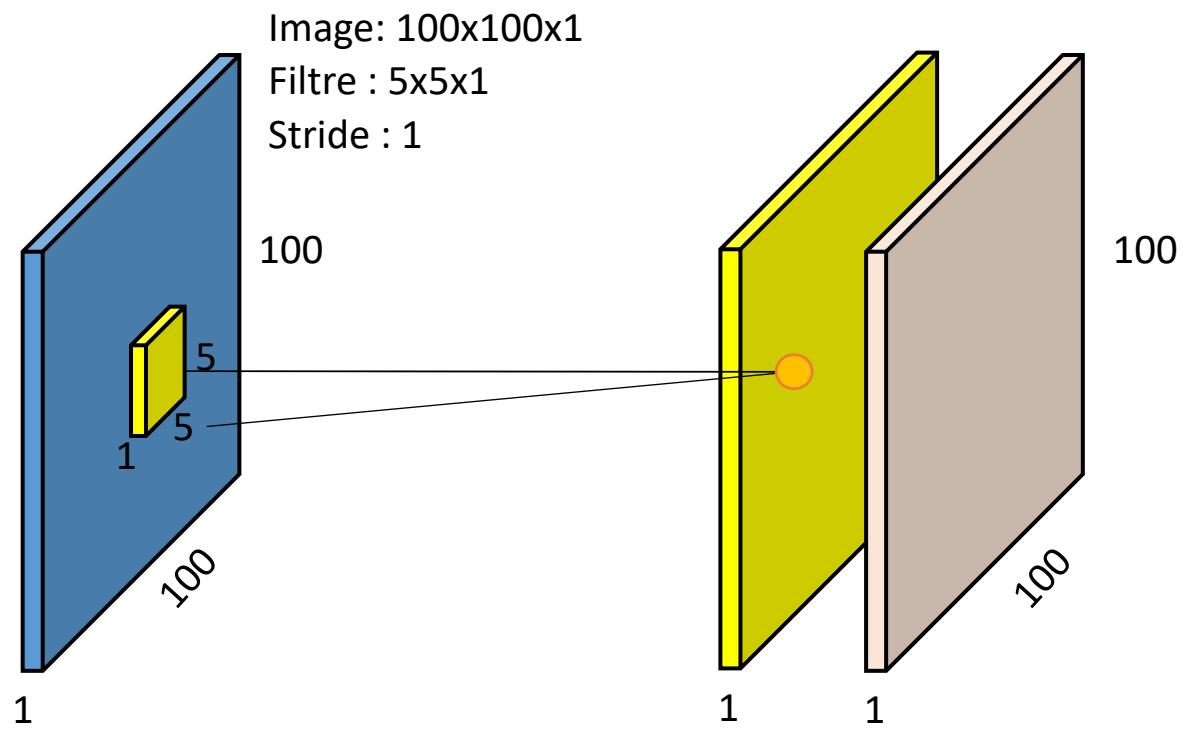


- 10,000 neurones par carte d'activation
- 50,000 neurones au total
- $5 \times 5 \times 5 = 125$ paramètres au total

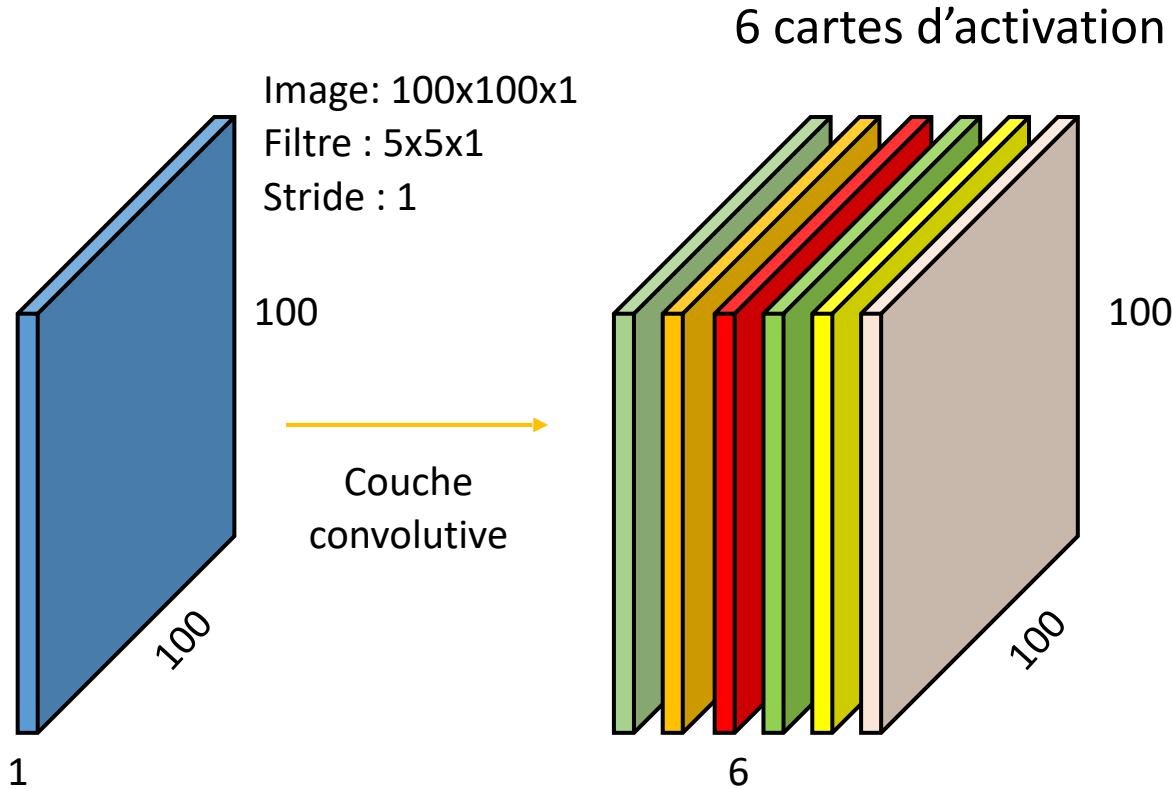
Représentation schématique
(1 filtre et 1 carte d'activation, convolution « same »)



Représentation schématique
(2 filtres et 2 cartes d'activation, convolution « *same* »)

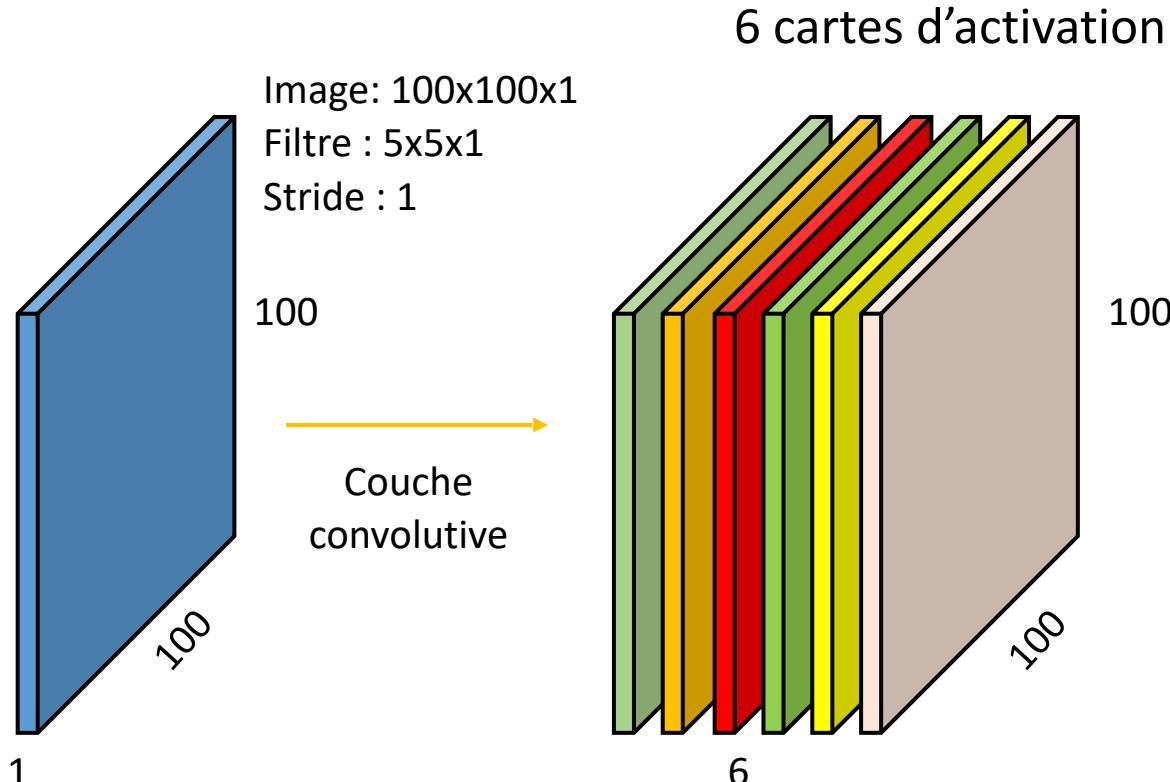


Représentation schématique
(6 filtres et 6 cartes d'activation, convolution « same »)



Combien de neurones
et de paramètres
au total?

Représentation schématique
(6 filtres et 6 cartes d'activation, convolution « same »)

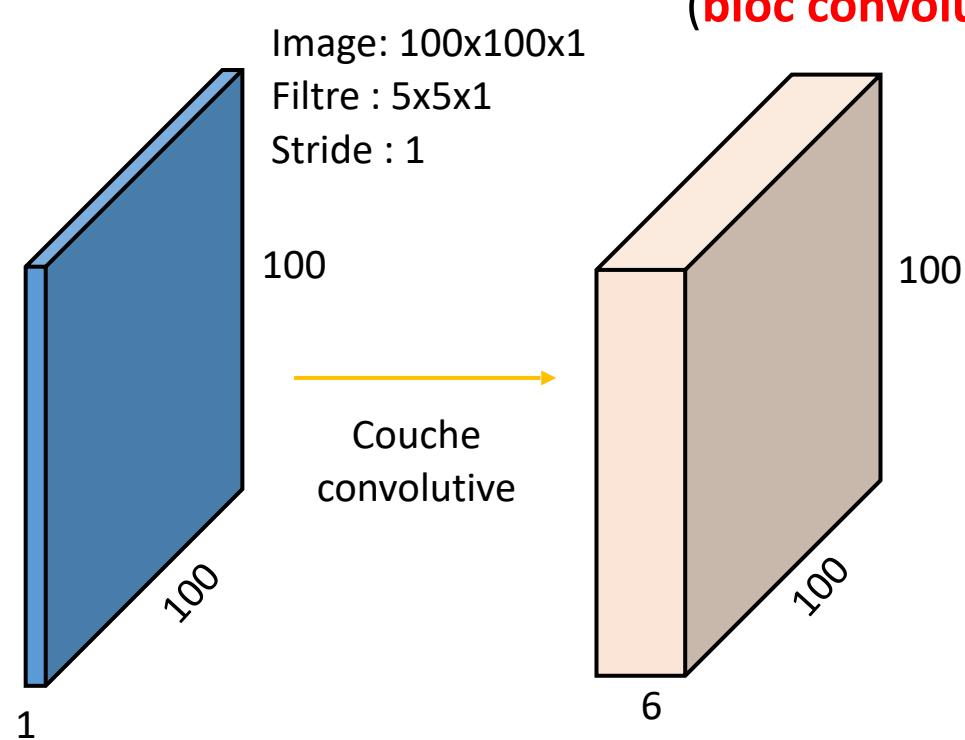


100x100x6=60,000 neurones
6x5x5x1=150 paramètres

A large yellow speech bubble contains the text '100x100x6=60,000 neurones' and '6x5x5x1=150 paramètres', providing specific numerical details about the size of the neural network and the number of parameters involved in the convolutional layer.

Représentation schématique simplifiée

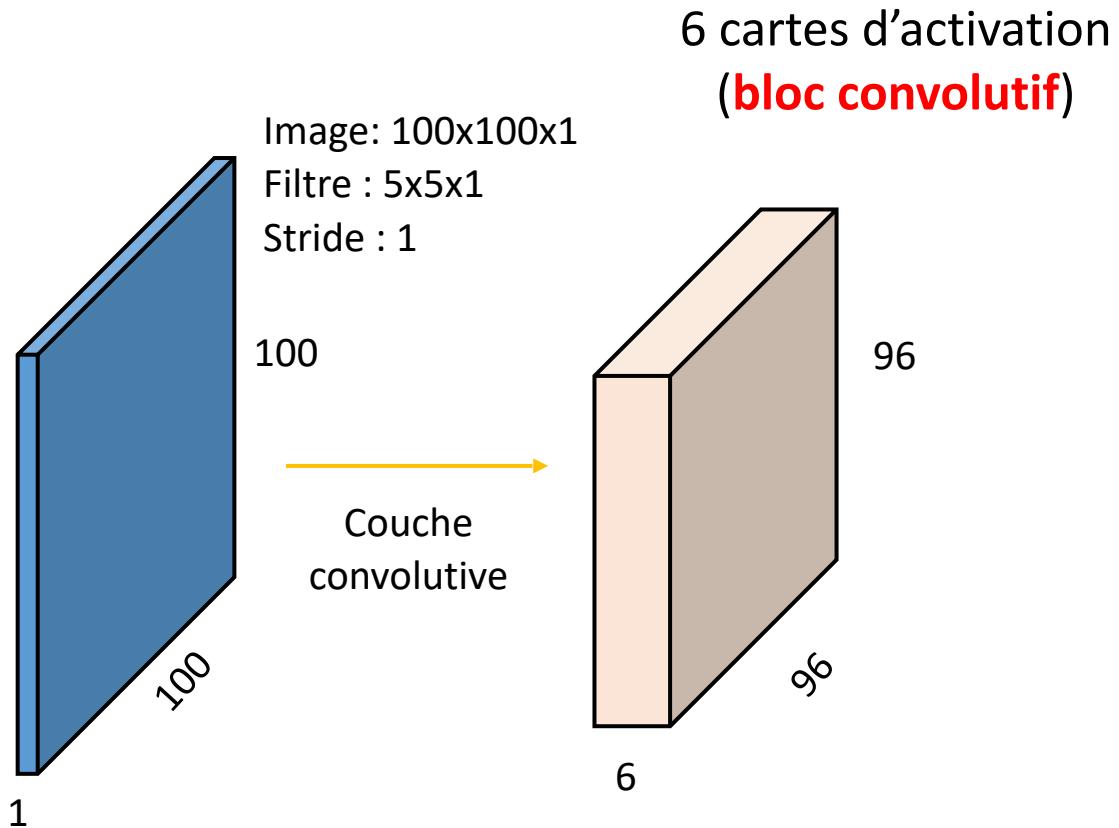
(6 filtres et 6 carte d'activation, convolution « *same* »)



100x100x6=60,000 neurones
6x5x5x1=150 paramètres

A yellow jagged shape containing text about the computational cost of the convolutional layer. It states that there are 60,000 neurons in the 6 activation maps and 150 parameters in the 6 filters.

Représentation schématique simplifiée
(6 filtres et 6 carte d'activation, convolution « *valid* »)



Combien de neurones
et de paramètres
au total?

Représentation schématique simplifiée

(6 filtres et 6 carte d'activation, convolution « *valid* »)

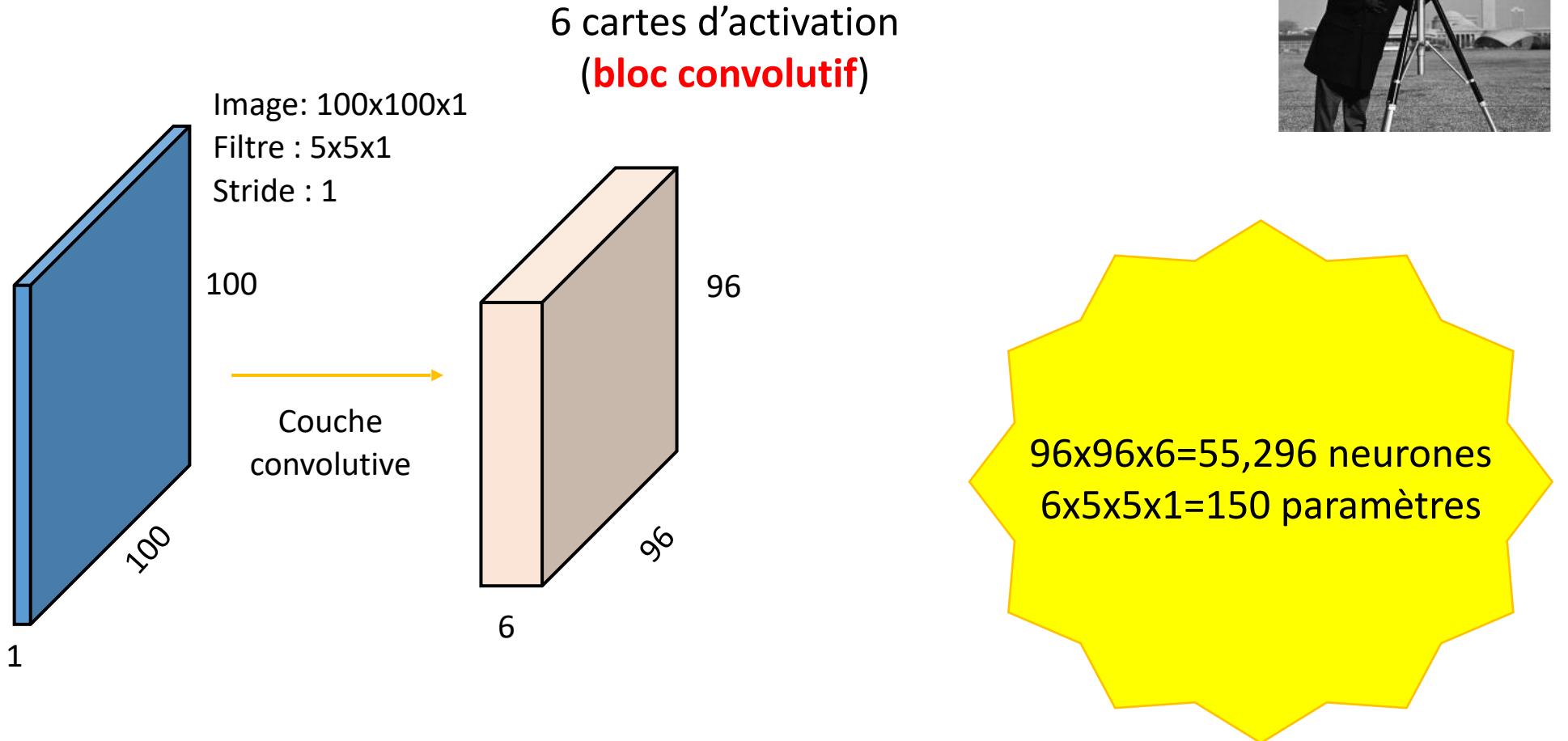
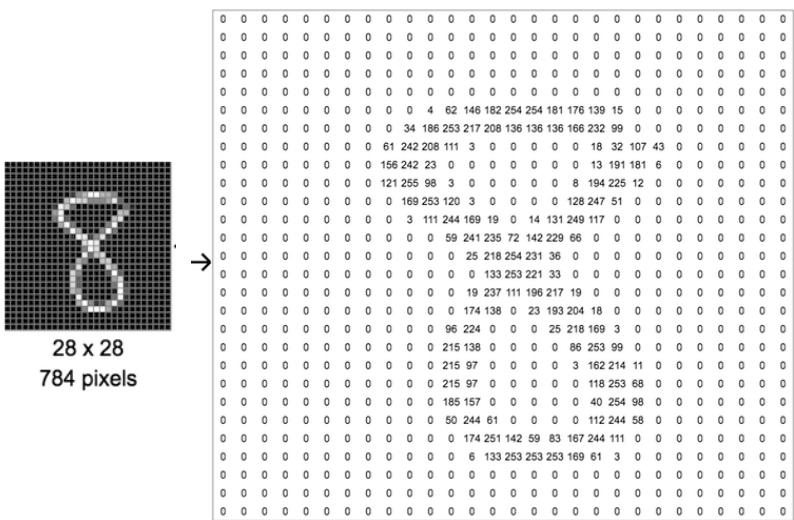


Image noir/blanc vs. couleur



https://ml4a.github.io/ml4a/neural_networks/

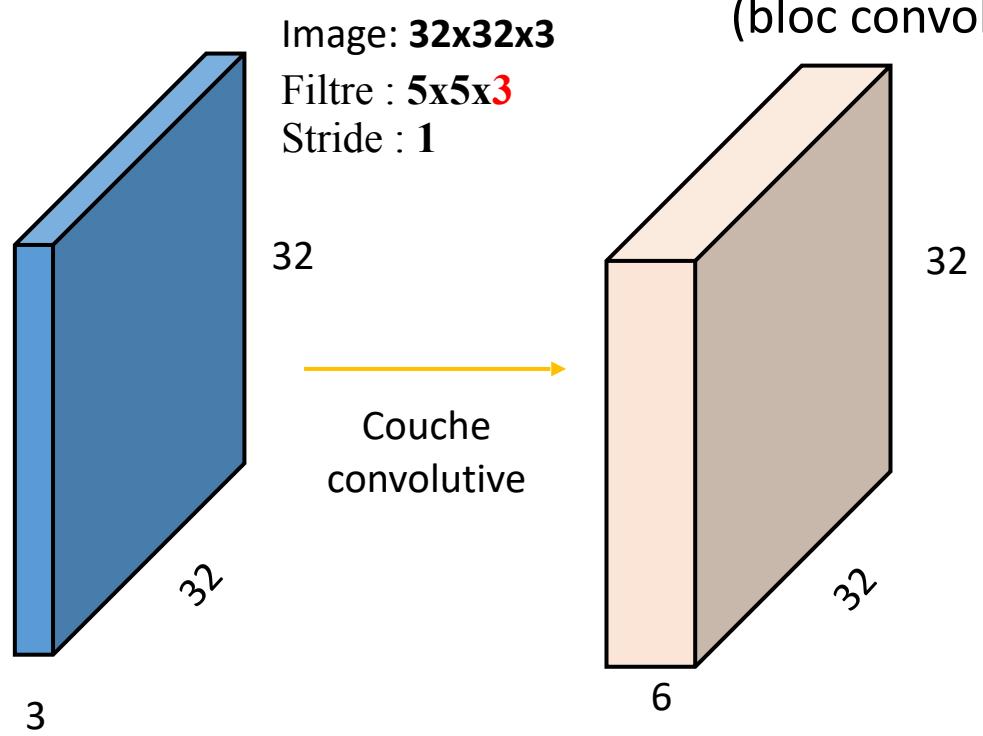
1 "canal" (*channel*)



<https://www.kdnuggets.com/2019/12/convert-rgb-image-grayscale.html>

3 "canaux" (*channels*)

Représentation schématique images couleurs
(ex.: images RGB de CIFAR10
convolution « *same* »)



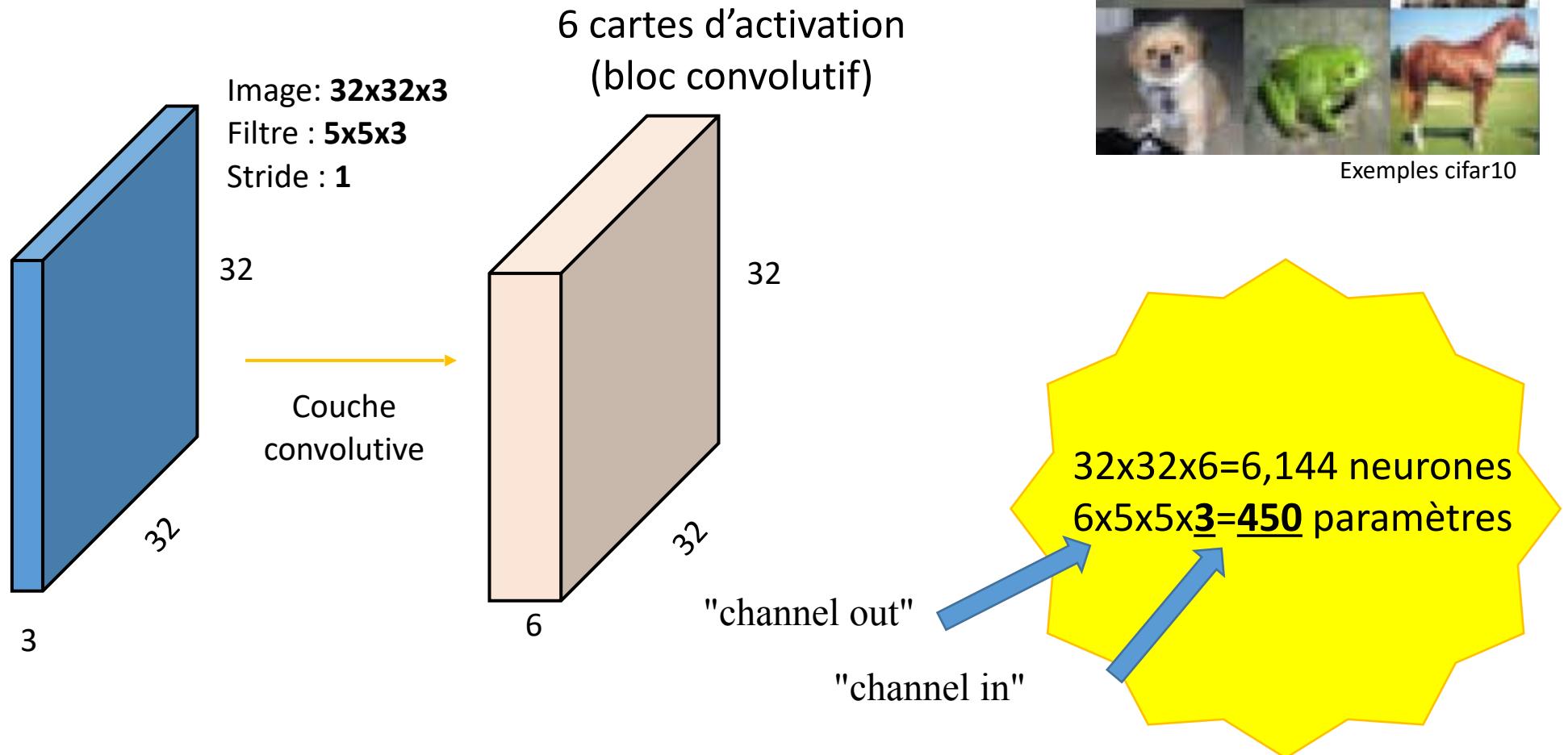
6 cartes d'activation (bloc convolutif)



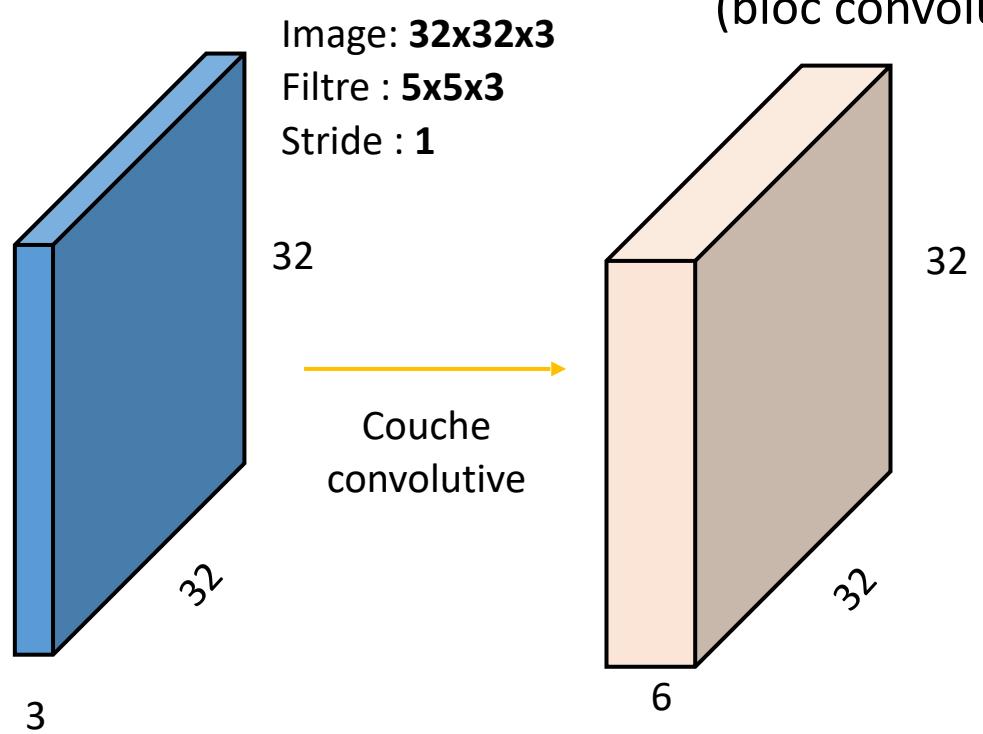
Exemples cifar10

Combien de neurones et de paramètres au total?

Représentation schématique images couleurs
(ex.: images RGB de CIFAR10
convolution « *same* »)



Représentation schématique images couleurs
(ex.: images RGB de CIFAR10
convolution « *same* »)

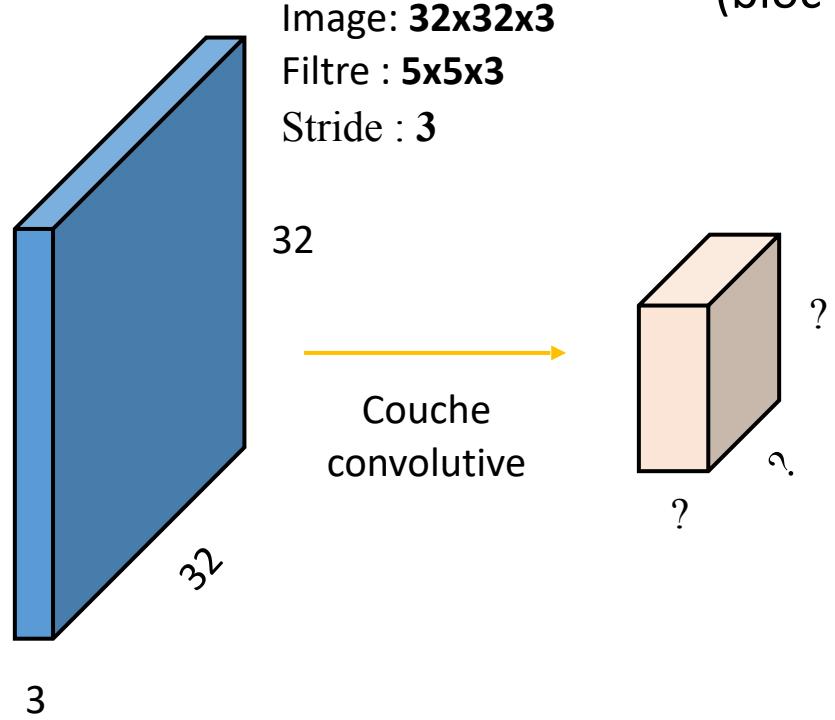


6 cartes d'activation
(bloc convolutif)



Qu'arrivera-t-il si on
utilise une stride de 3?

Représentation schématique images couleurs
(ex.: images RGB de CIFAR10
convolution « *same* »)



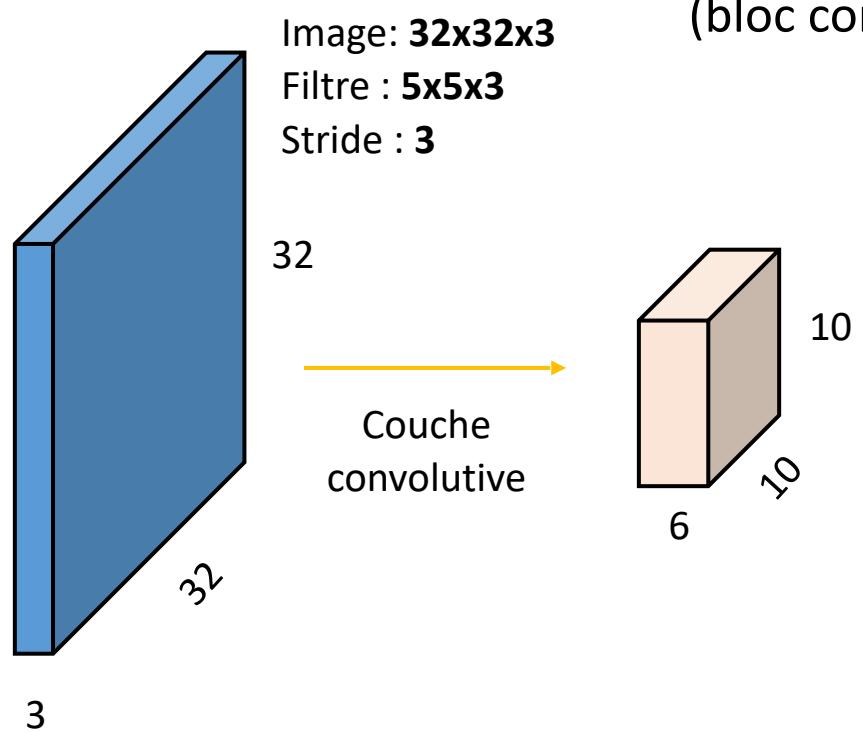
6 cartes d'activation
(bloc convolutif)



Exemples cifar10

$$\begin{aligned} & \textcolor{red}{(D-F)/S+1} \\ & = \\ & \textcolor{red}{(32-5)/3+1=10} \end{aligned}$$

Représentation schématique images couleurs
(ex.: images RGB de CIFAR10
convolution « *same* »)



6 cartes d'activation
(bloc convolutif)

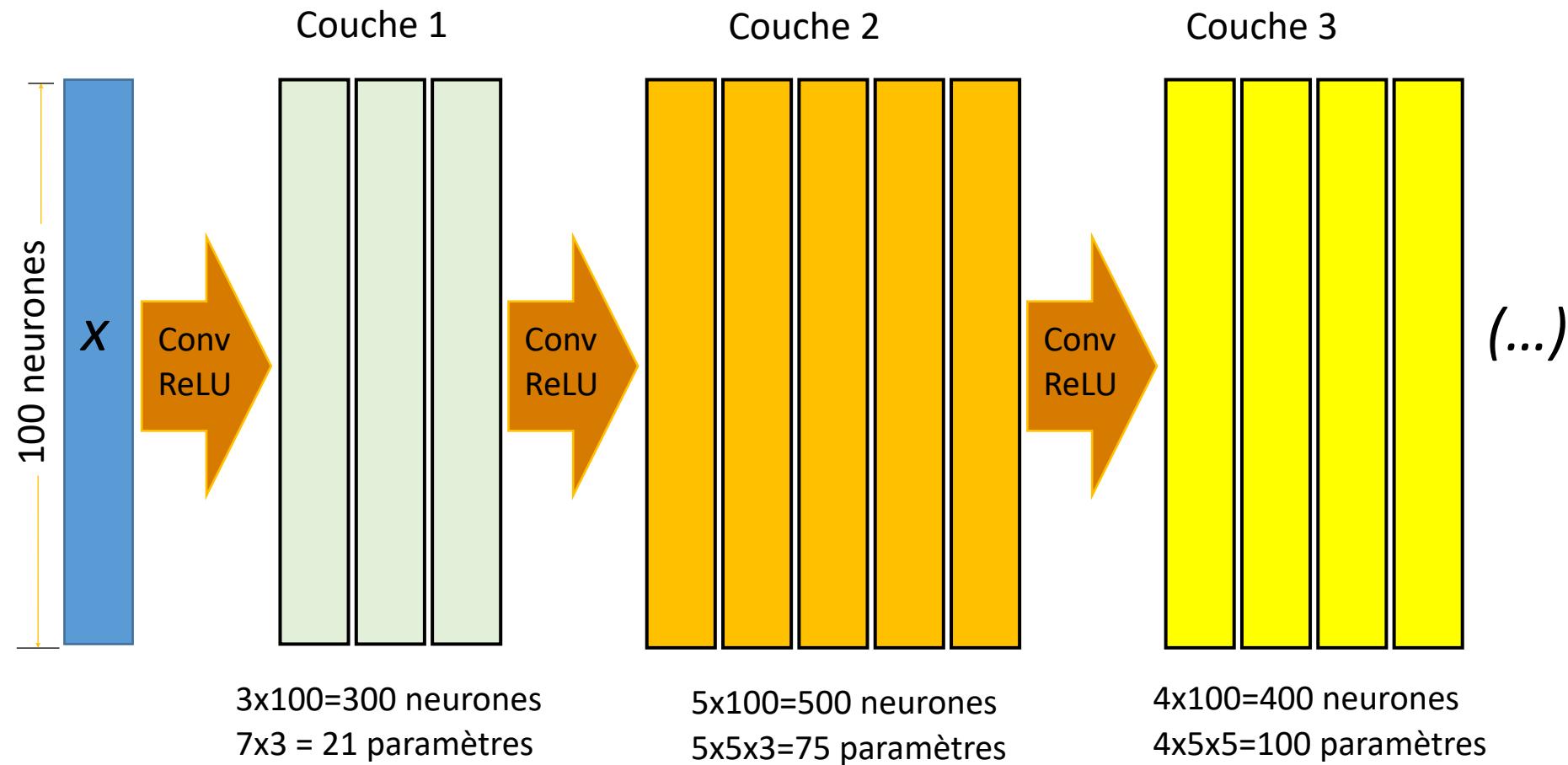


10x10x6=600 neurones
6x5x5x3=450 paramètres

Tout comme un Perceptron multi-couches, un réseau à convolution contient **plusieurs couches consécutives**

Exemple : 3 filtres couche 1 : taille 7
5 filtres couche 2 : taille 5
4 filtres couche 3 : taille 5

Convolution « same »
Stride 1



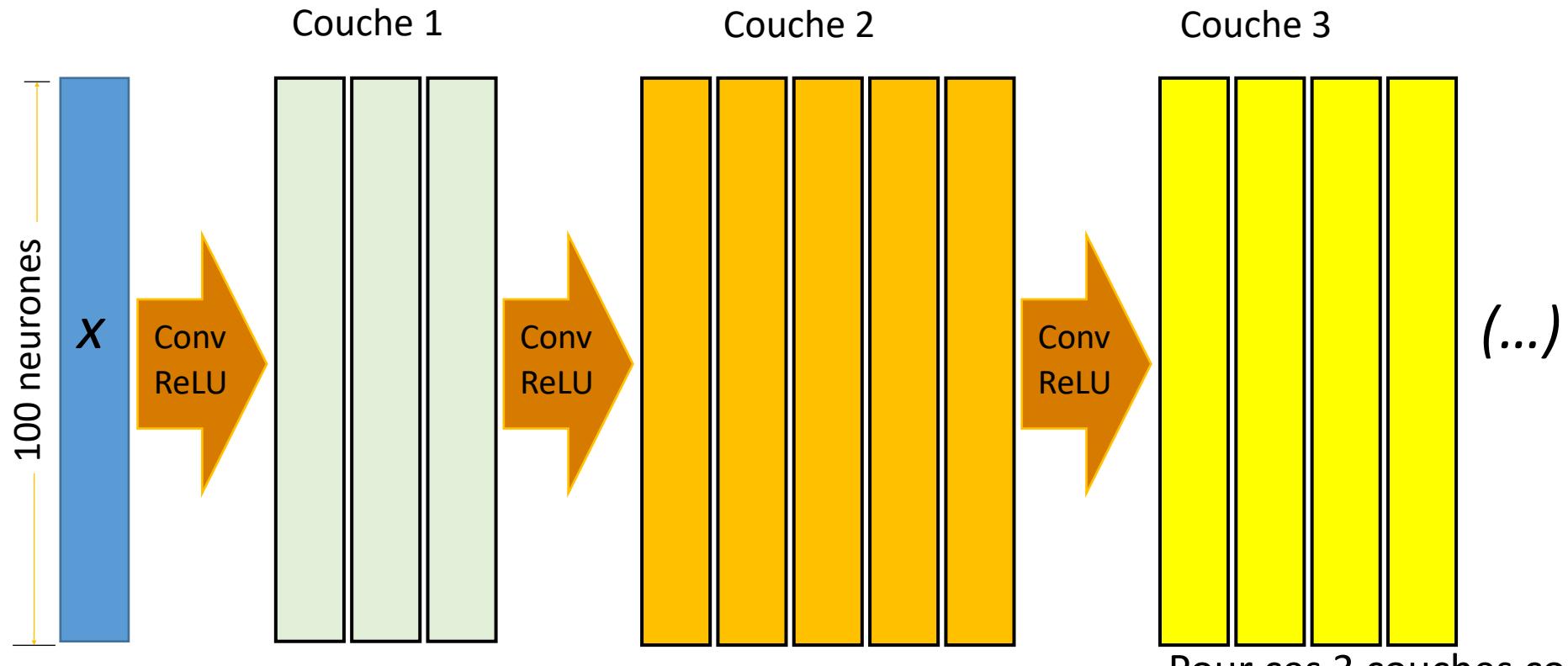
Exemple : 3 filtres couche 1 : taille 7

5 filtres couche 2 : taille 5

4 filtres couche 3 : taille 5

Convolution « *same* »

Stride 1

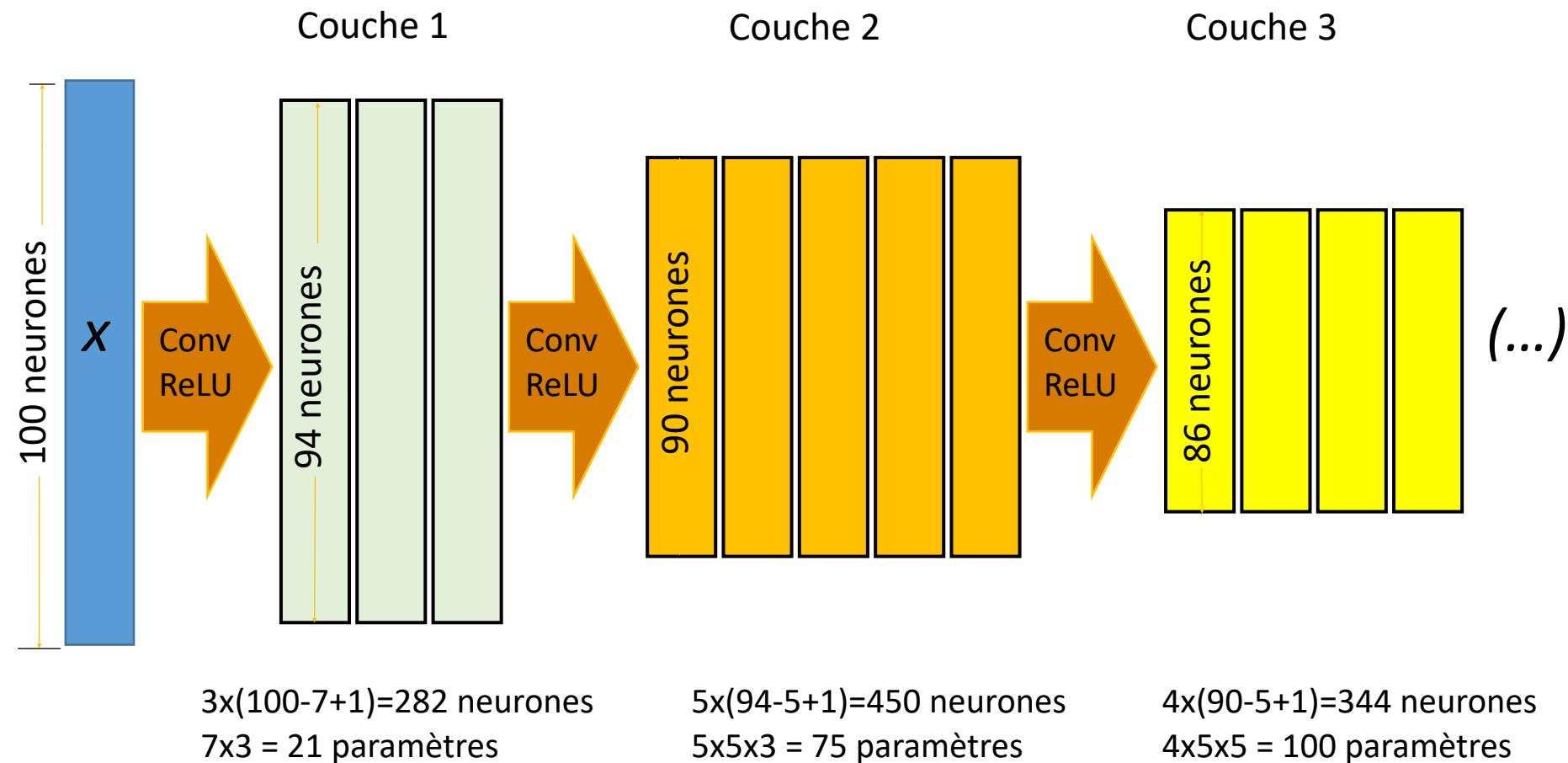


Pour ces 3 couches convolutionnelles

- 1200 neurones au total
- 196 paramètres au total

Exemple : 3 filtres couche 1 : taille 7
5 filtres couche 2 : taille 5
4 filtres couche 3 : taille 5

Convolution « **valid** »
Stride 1

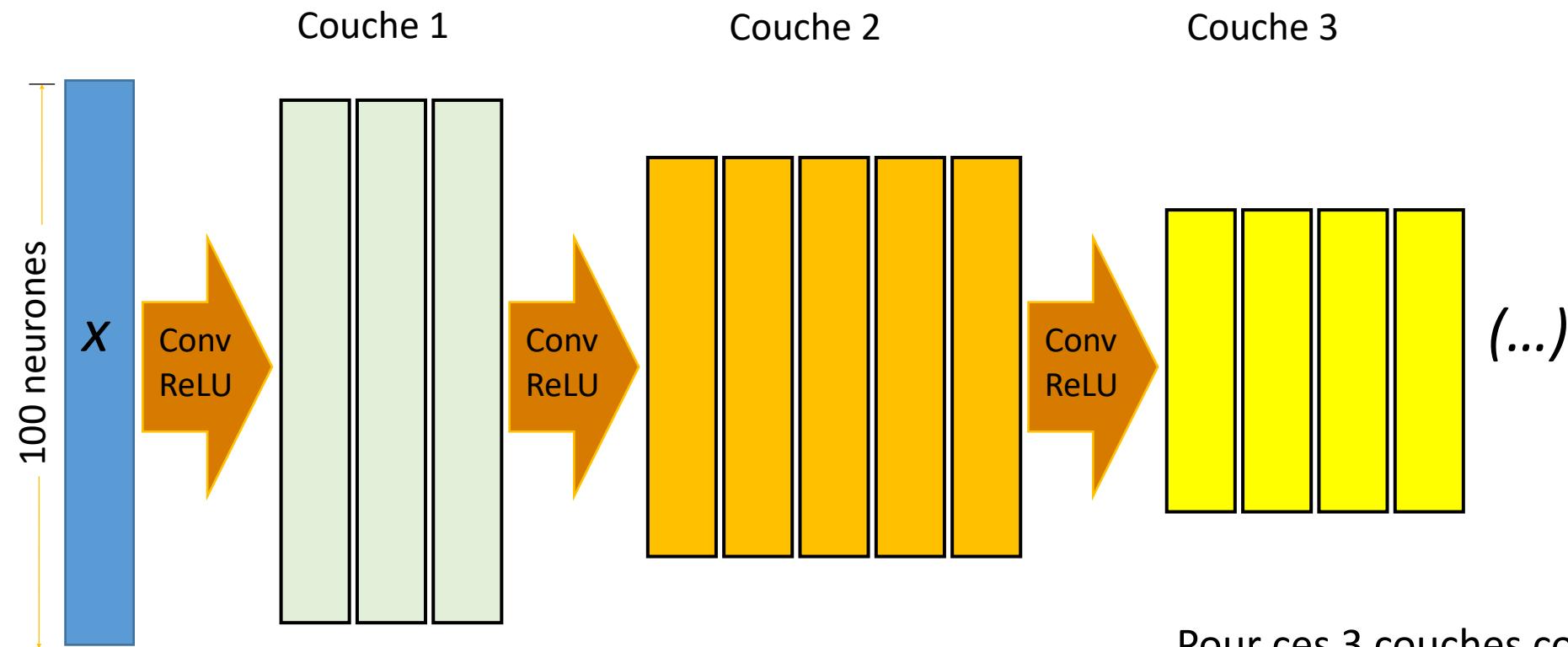


Exemple : 3 filtres couche 1 : taille 7

5 filtres couche 2 : taille 5

4 filtres couche 3 : taille 5

Convolution « **valid** »
Stride 1

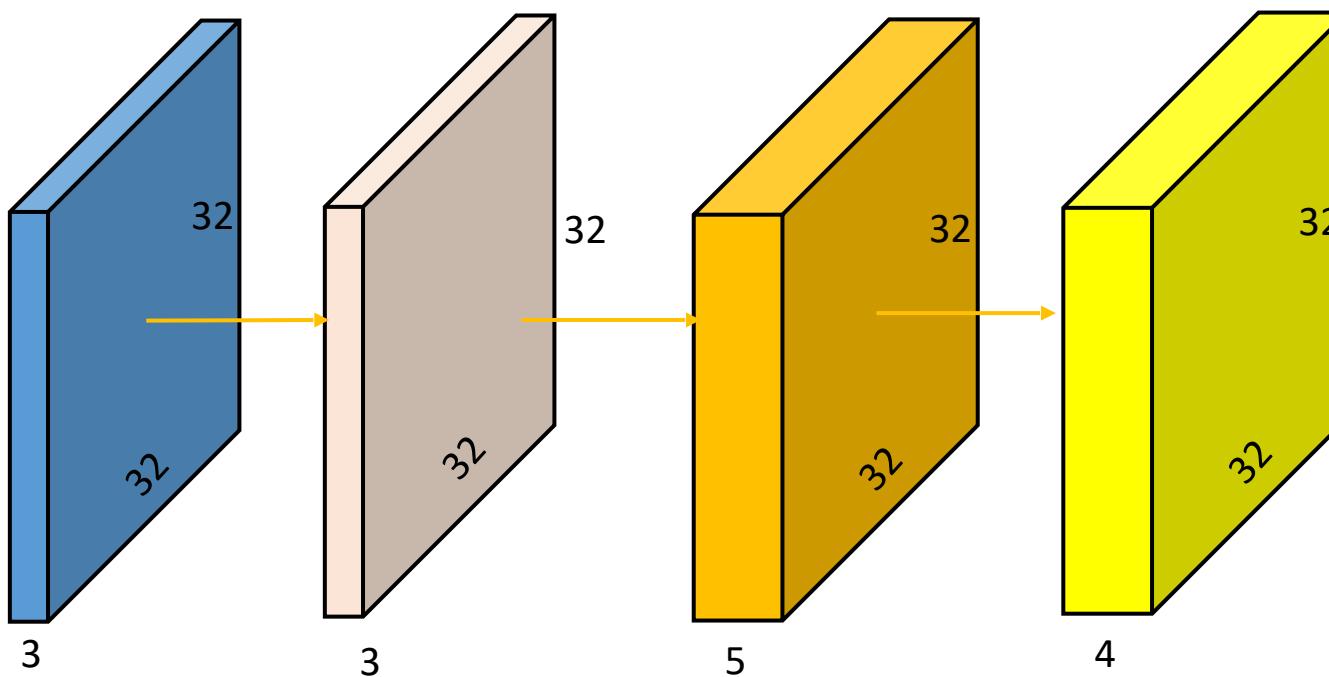


Pour ces 3 couches convolutionnelles

- 1076 neurones au total
- 196 paramètres au total

Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « *same* »

Image: 32x32x3
Stride : 1



$$3 \times (32 \times 32) = 3,072 \text{ neurones}$$
$$3 \times 7 \times 7 \times 3 = 441 \text{ paramètres}$$

$$5 \times (32 \times 32) = 5,120 \text{ neurones}$$
$$5 \times 9 \times 9 \times 3 = 1,215 \text{ paramètres}$$

$$4 \times (32 \times 32) = 4,096 \text{ neurones}$$
$$4 \times 11 \times 11 \times 5 = 2,420 \text{ paramètres}$$

Image: **32x32x3**
Stride : 1

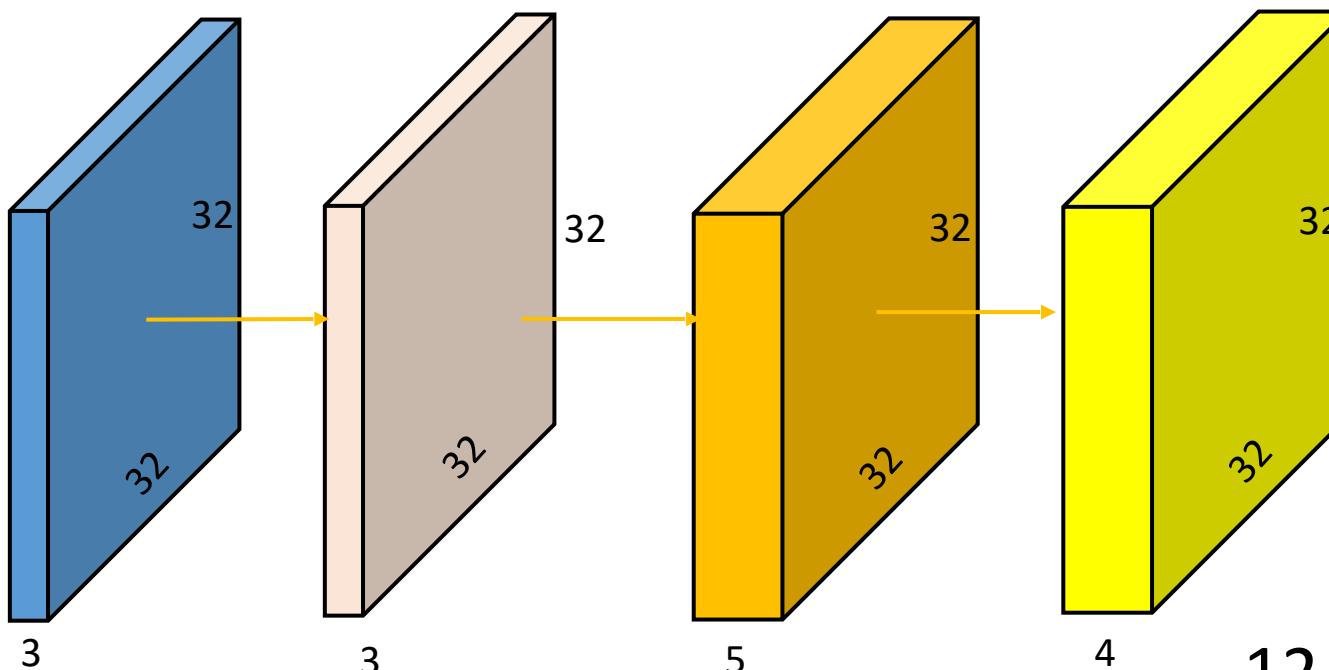
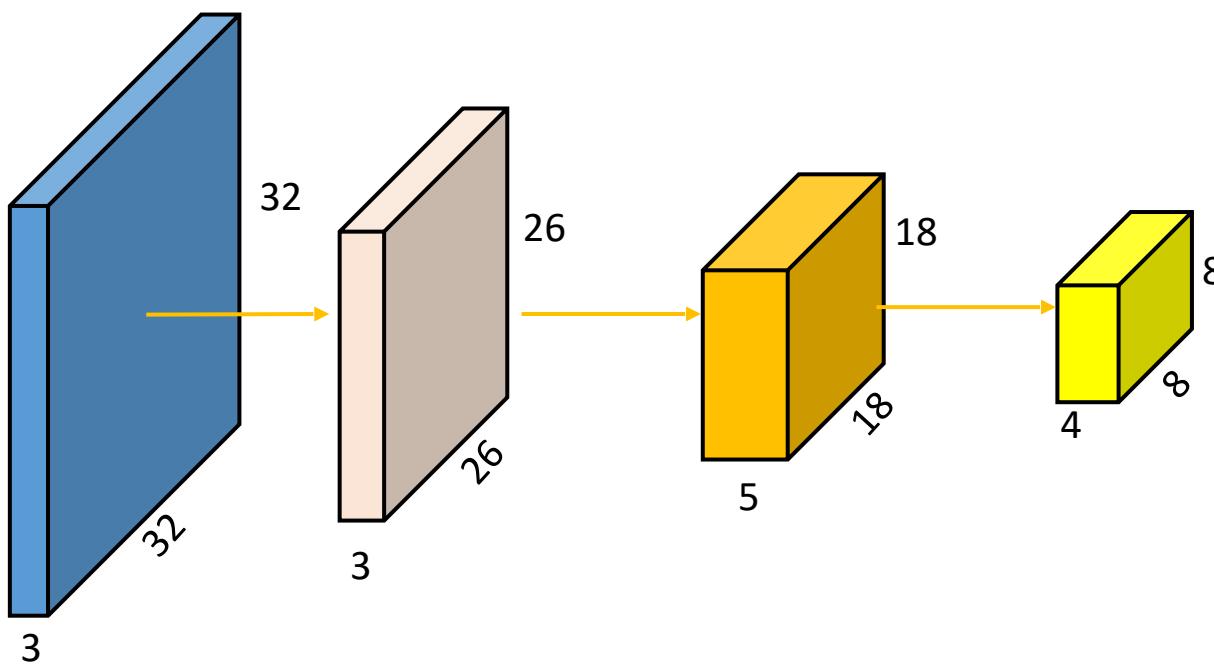


Image RGB : couche 1 : 3 filtres de taille 7×7
couche 2 : 5 filtres de taille 9×9
couche 3 : 4 filtres de taille 11×11
convolution « *same* »

**12,288 neurones au total
4,076 paramètres au total**

Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « valid »

Image: 32x32x3
Stride : 1



$$3 \times (26 \times 26) = 2,028 \text{ neurones}$$
$$3 \times 7 \times 7 \times 3 = 441 \text{ paramètres}$$

$$5 \times (18 \times 18) = 1,620 \text{ neurones}$$
$$5 \times 9 \times 9 \times 3 = 1,215 \text{ paramètres}$$

$$4 \times (8 \times 8) = 256 \text{ neurones}$$
$$4 \times 11 \times 11 \times 5 = 2,420 \text{ paramètres}$$

Image: 32x32x3
Stride : 1

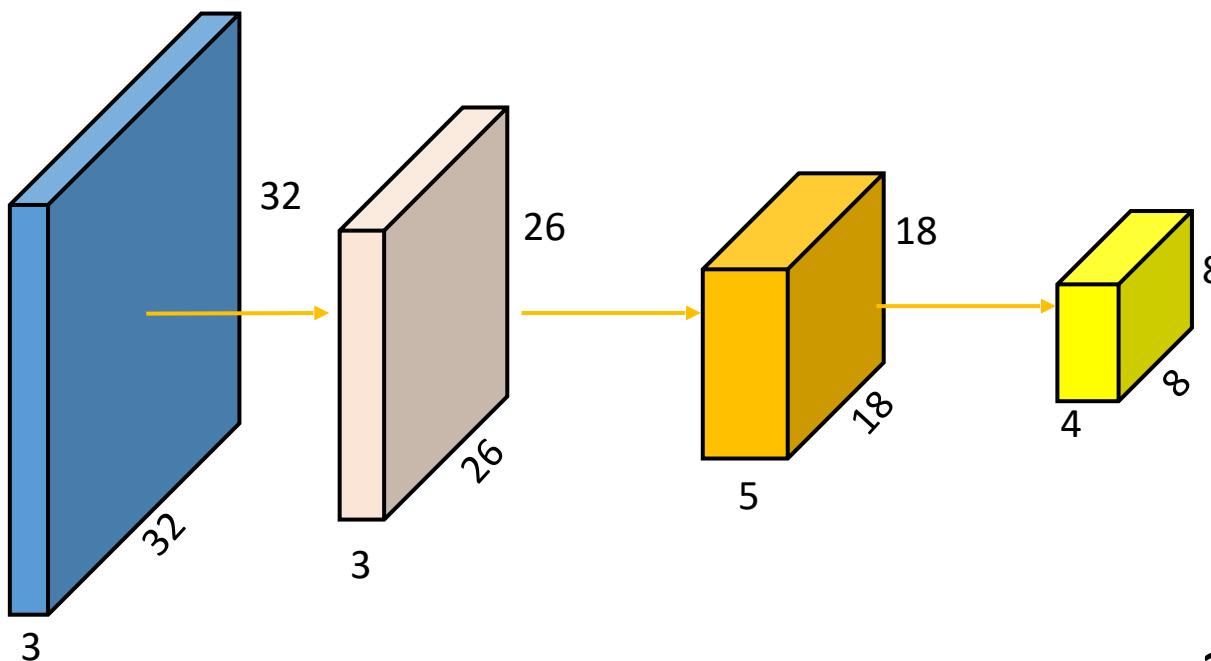


Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « valid »

3,904 neurones au total
4,076 paramètres au total

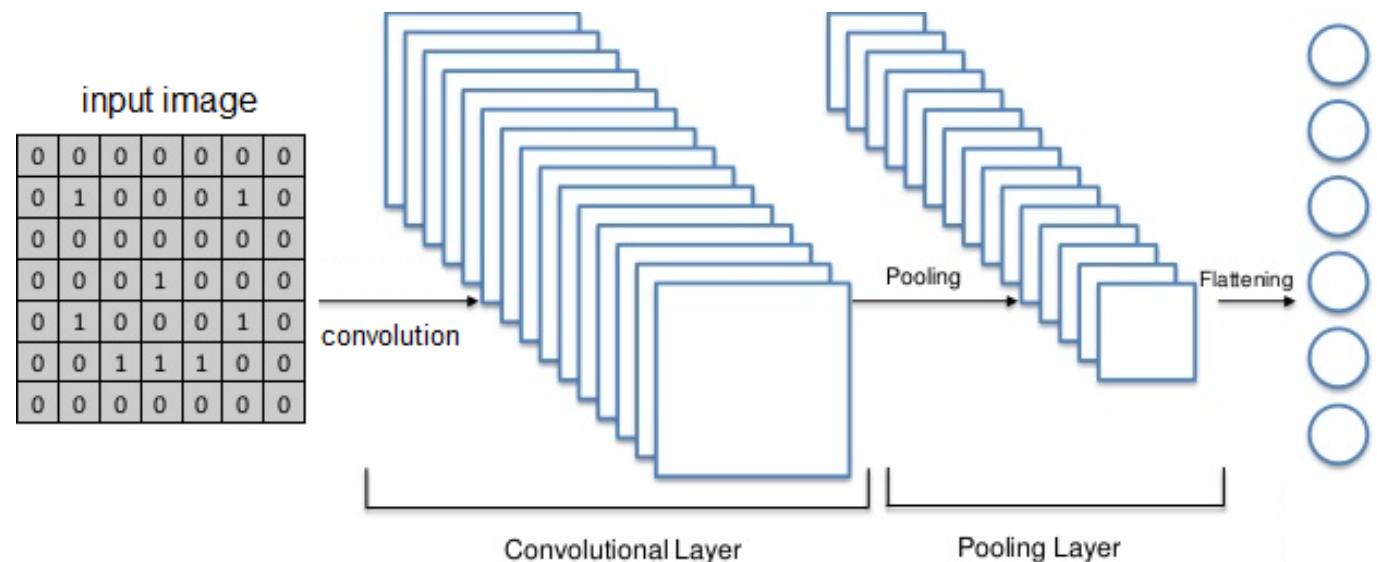
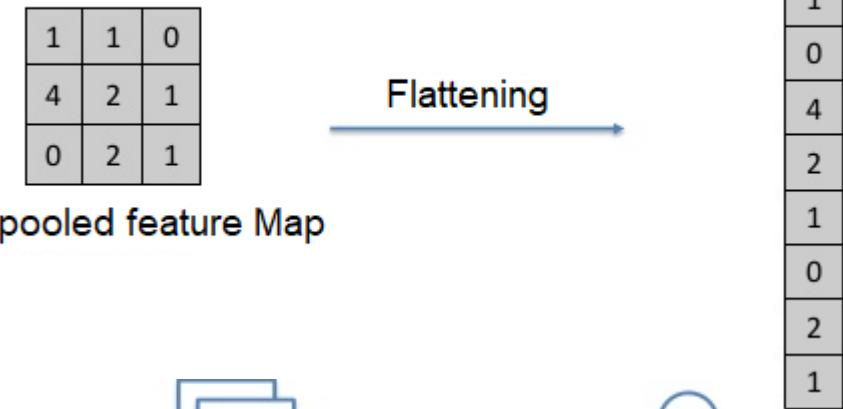
Tout comme un Perceptron multi-couches, un réseau à convolution se termine par une **couche de sortie** avec **1 neurone par variable prédite**

Flattening

Cette étape est simple.

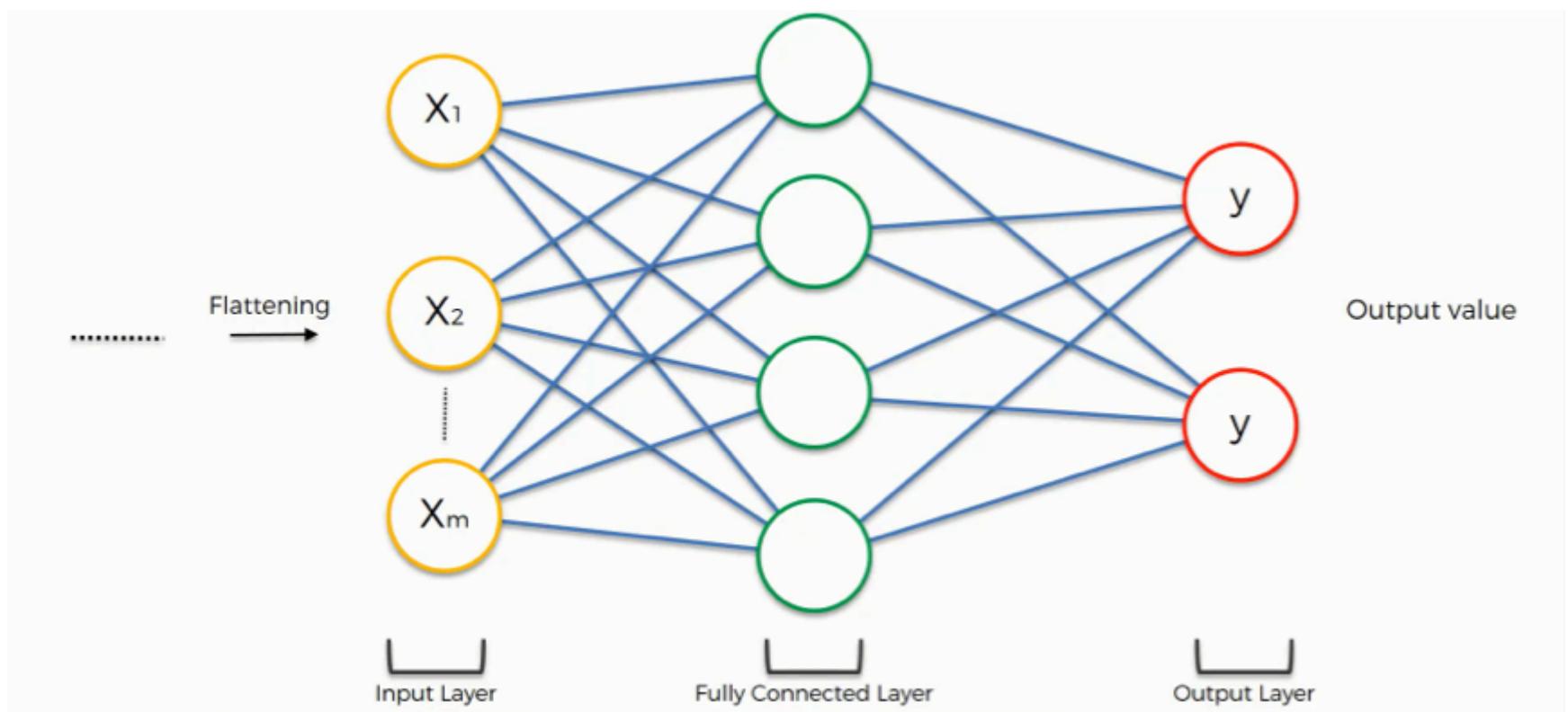
Après avoir terminé les deux étapes précédentes, nous sommes censés avoir une carte des fonctionnalités regroupées maintenant. Aplatir notre carte d'entités regroupées dans une colonne comme dans l'image.

La raison pour laquelle nous faisons cela est que nous allons devoir insérer ces données dans un réseau neuronal artificiel plus tard.

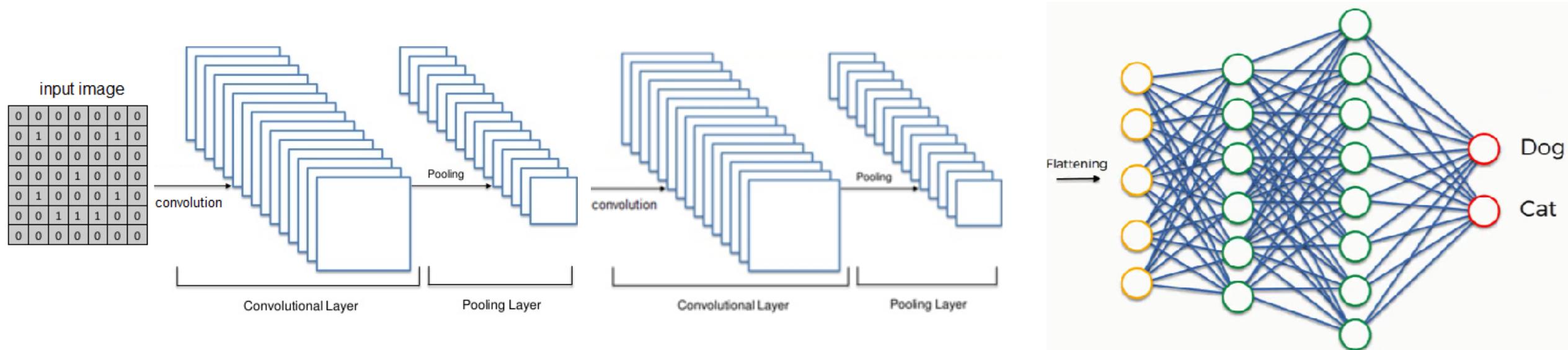


Full Connection

- C'est là que les réseaux de neurones artificiels et les réseaux de neurones convolutifs se heurtent lorsque nous ajoutons les premiers à nos seconds.
- C'est ici que le processus de création d'un réseau neuronal convolutif commence à prendre un tour plus complexe et sophistiqué.



Full Connection

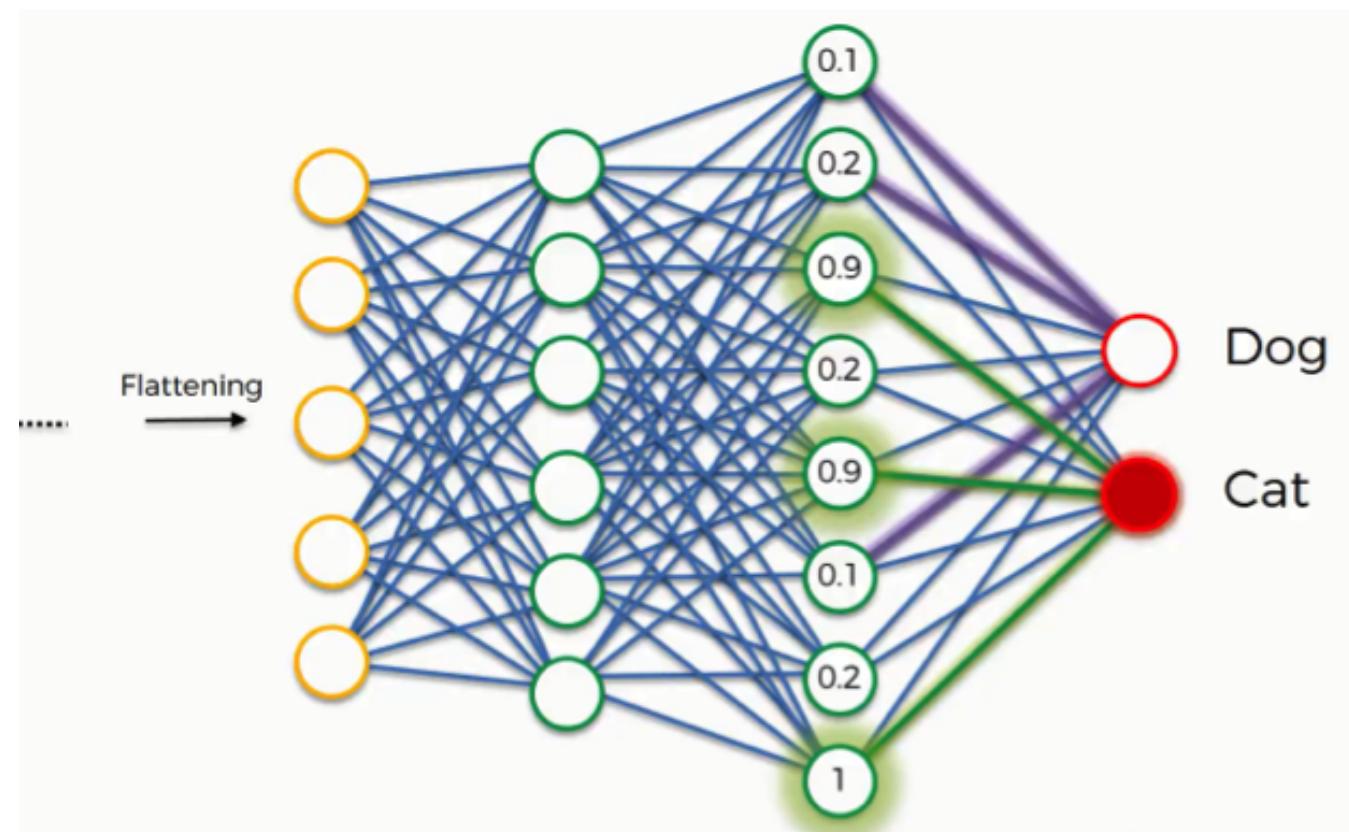


La couche softmax

- z_i sont les éléments du vecteur d'entrée et peuvent prendre n'importe quelle valeur réelle
- K est le nombre de classes dans le classifieur multi-classes.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

Dans ce cas K=2



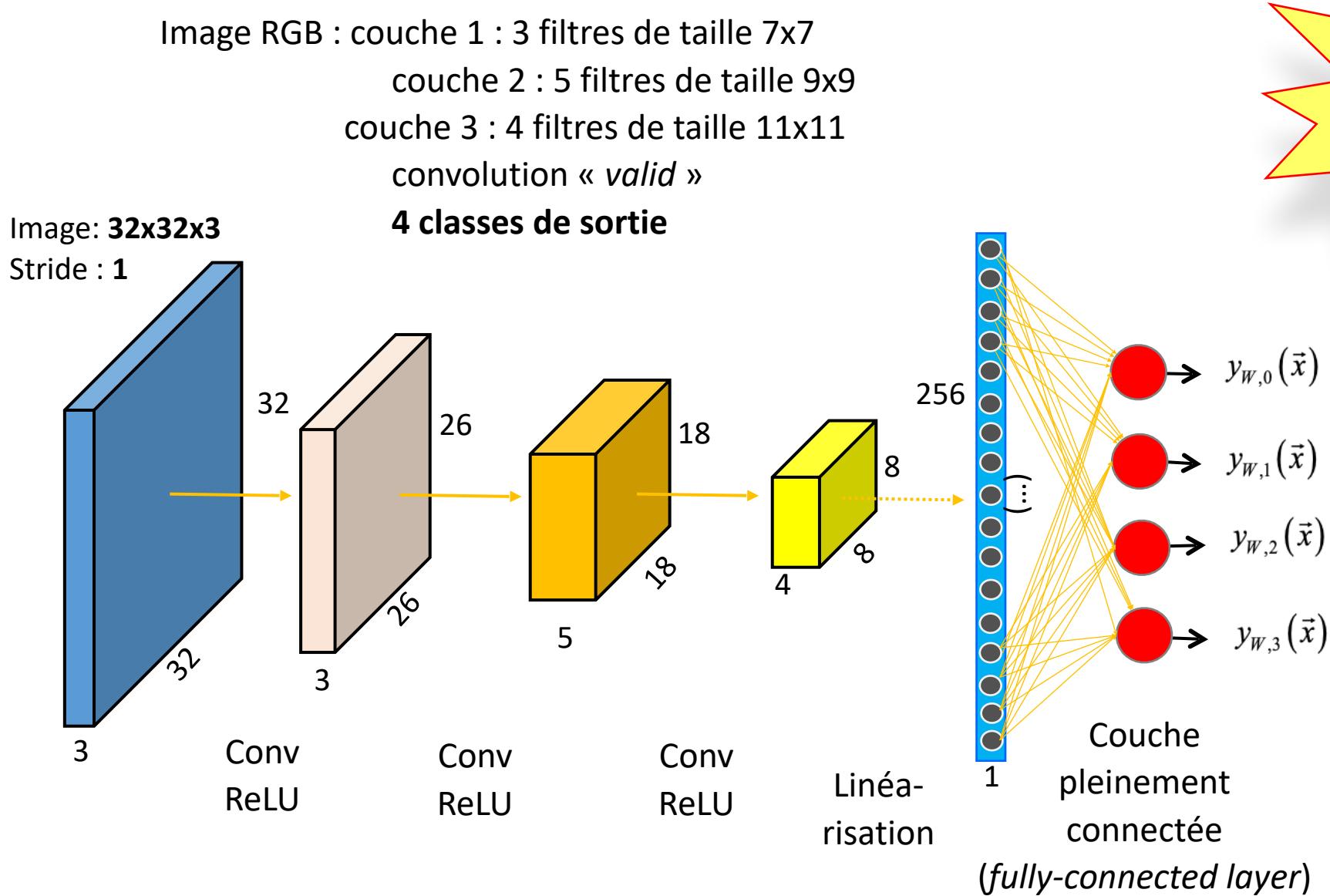
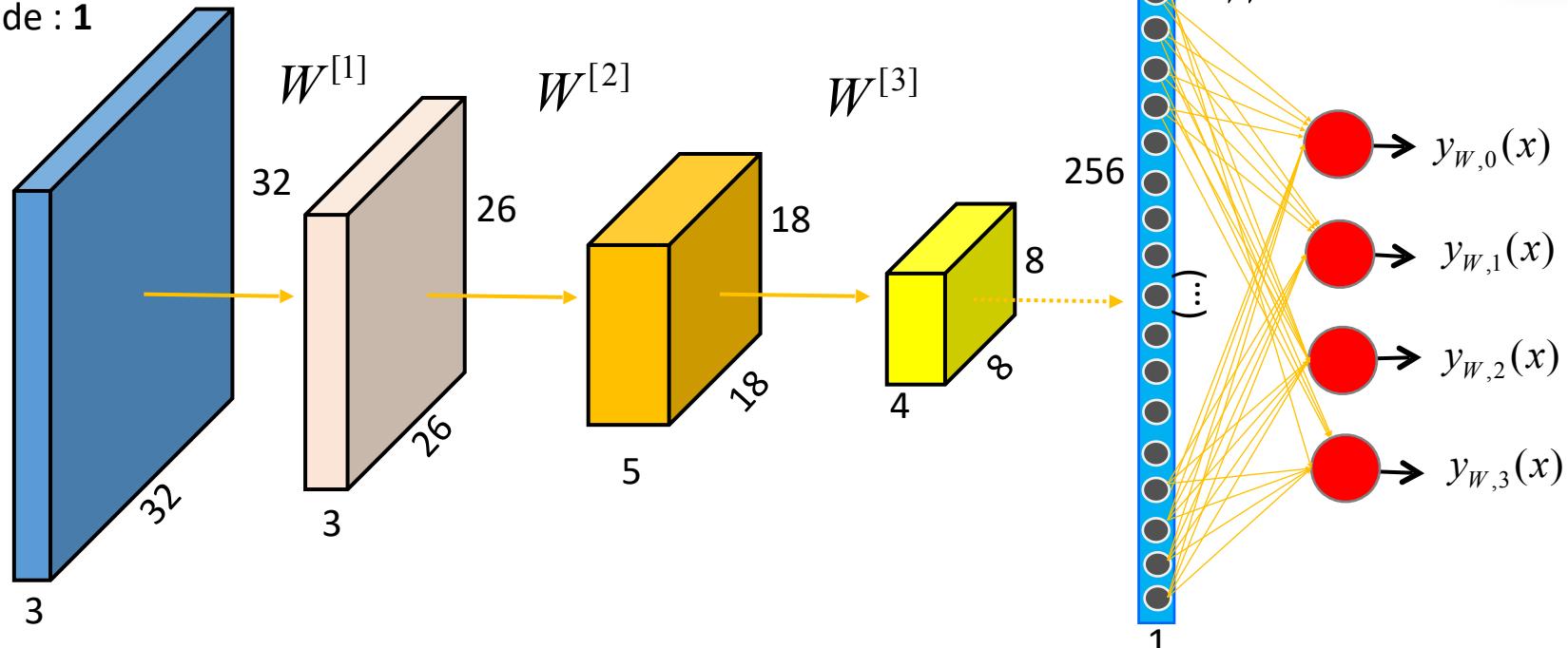


Image RGB : couche 1 : 3 filtres de taille 7x7
 couche 2 : 5 filtres de taille 9x9
 couche 3 : 4 filtres de taille 11x11
 convolution « valid »
4 classes de sortie
 Image: 32x32x3
 Stride : 1



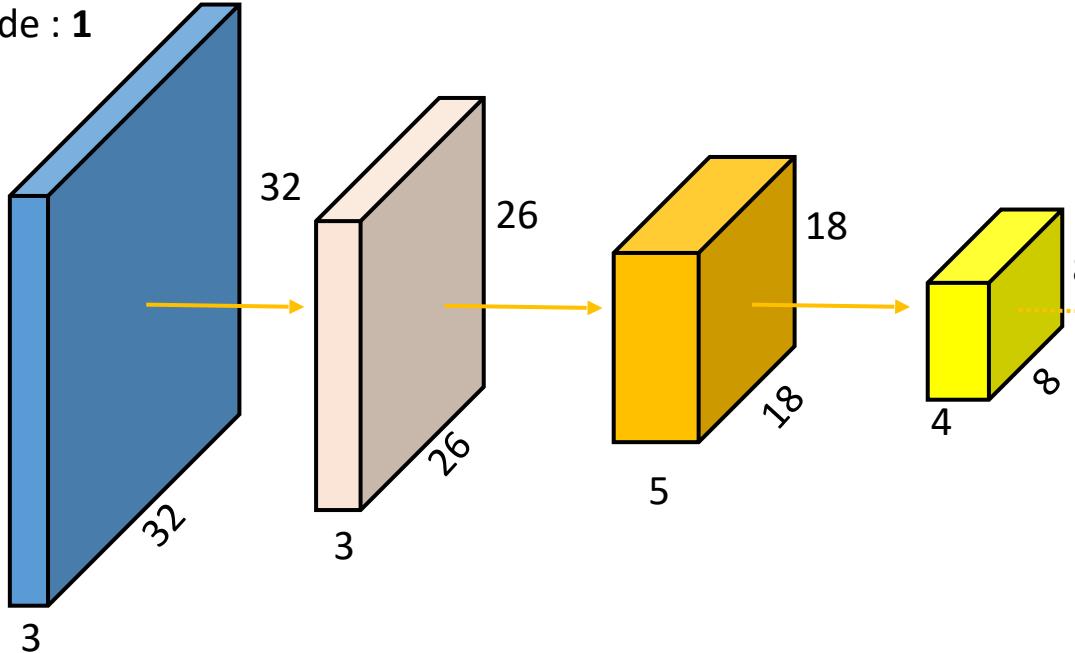
$$y_W(x) = W^{[4]} \left(W^{[3]} * h \left(W^{[2]} * h \left(W^{[1]} * x \right) \right) \right)$$



$$y_W(x) = W^{[4]} \left(W^{[3]} * h \left(W^{[2]} * h \left(W^{[1]} * x \right) \right) \right)$$



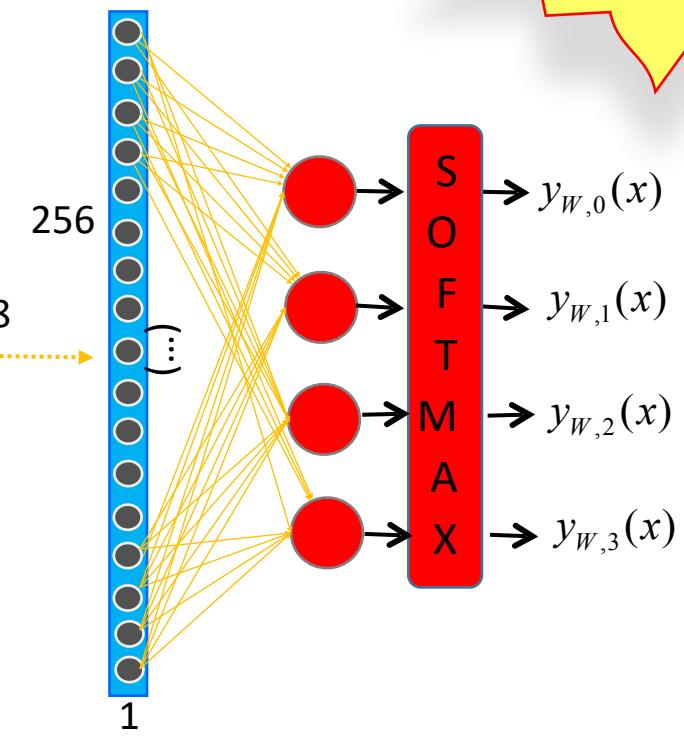
Image: **32x32x3**
Stride : 1

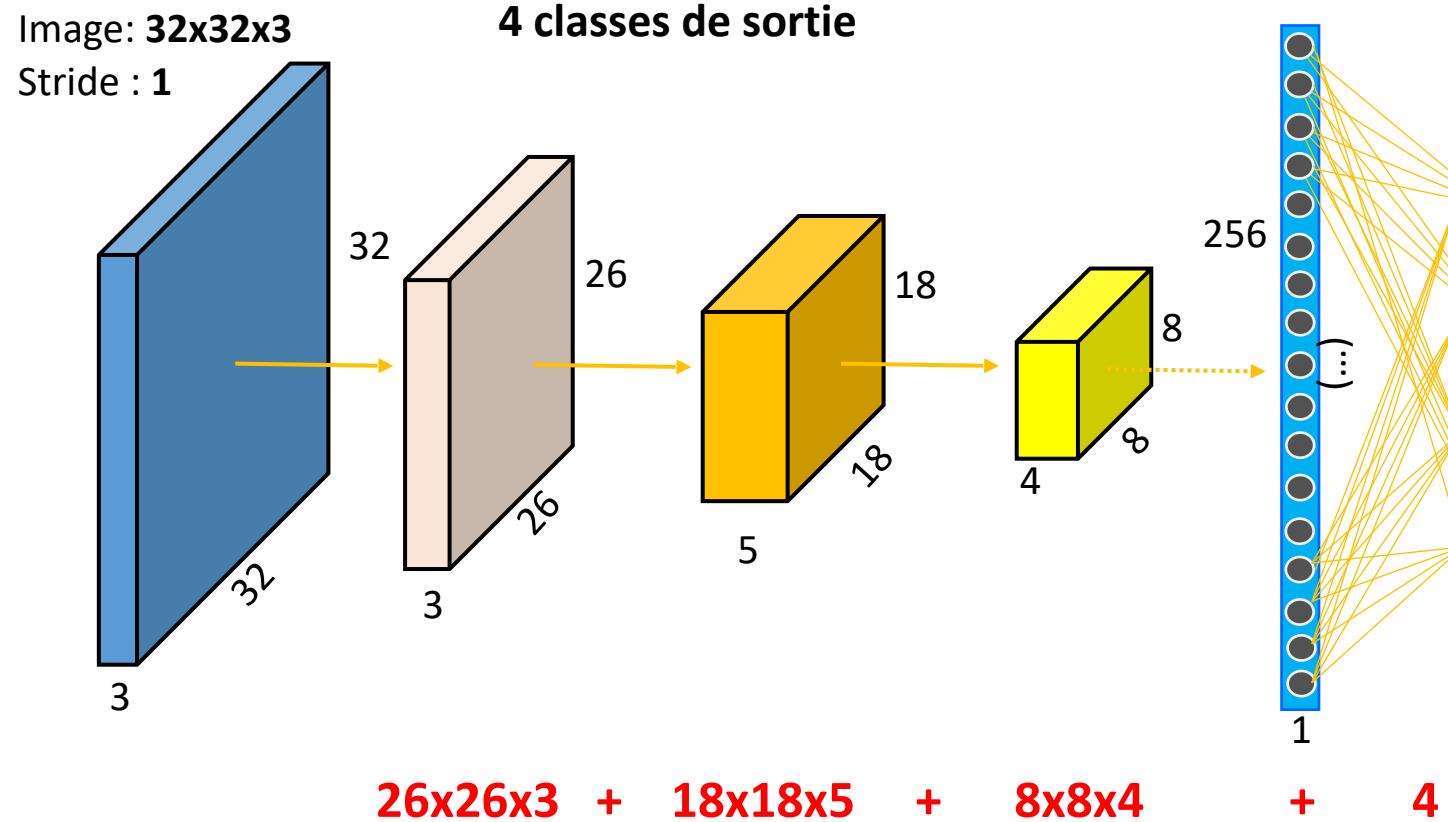


Nombre total de neurones?

Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « *valid* »

4 classes de sortie

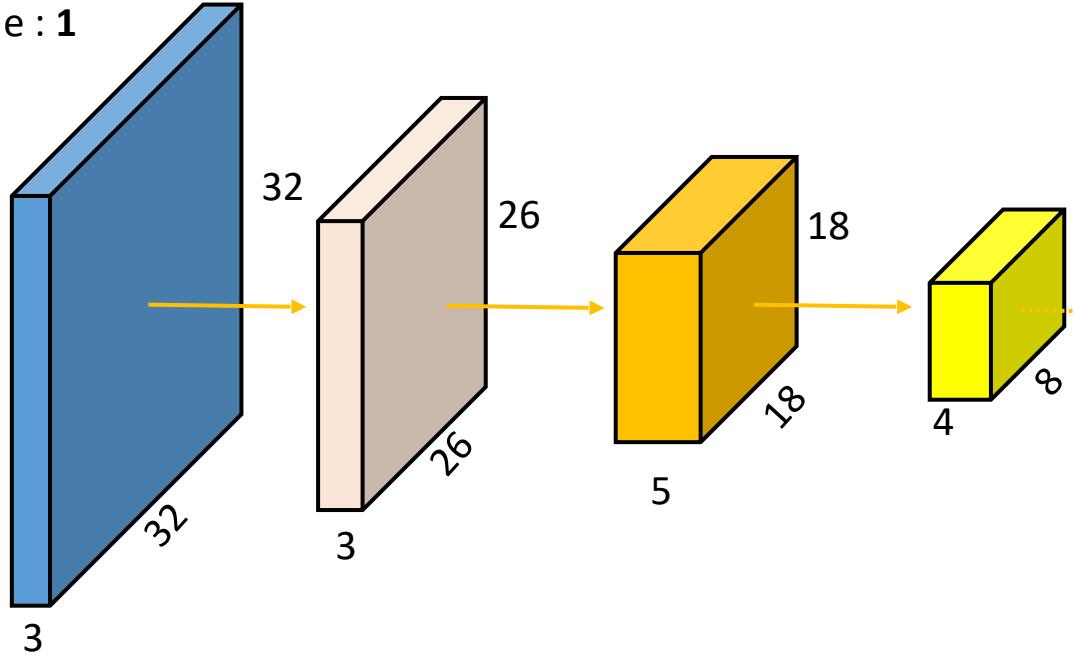




Cross-entropy loss

Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « valid »
4 classes de sortie

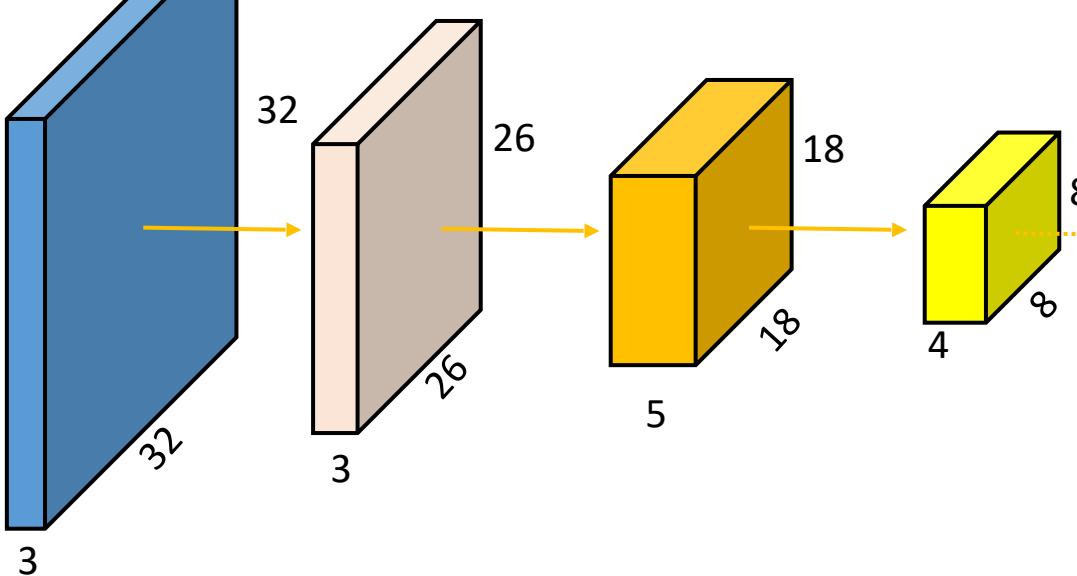
Image: 32x32x3
Stride : 1



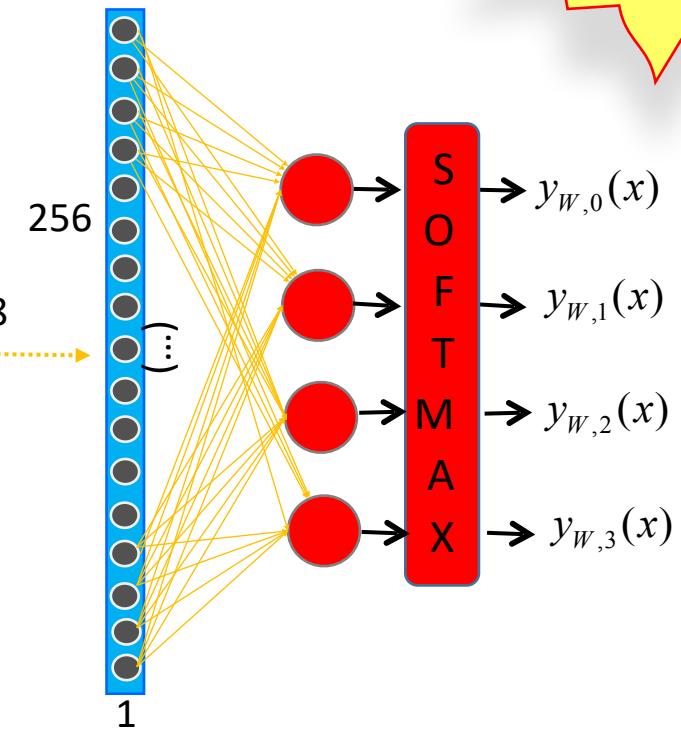
3,908 neurones

Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « valid »
4 classes de sortie

Image: 32x32x3
Stride : 1



Nombre total de paramètres?



**Cross-entropy
loss**

Image: $32 \times 32 \times 3$
Stride : 1

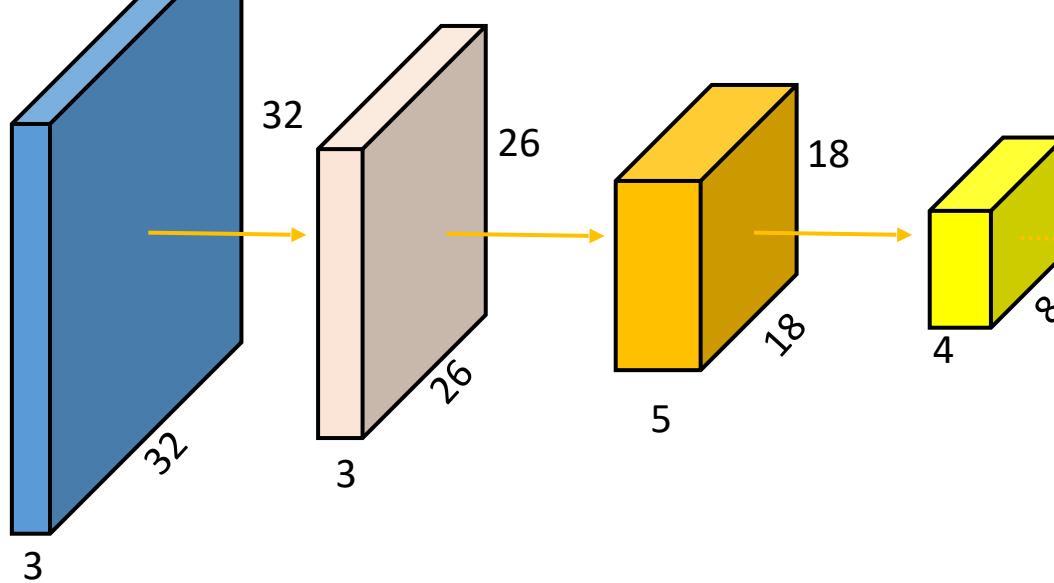
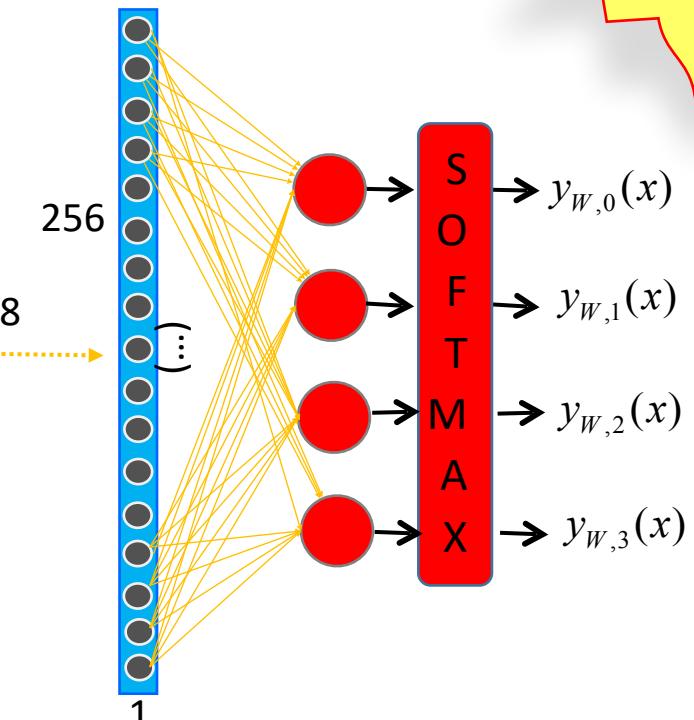


Image RGB : couche 1 : 3 filtres de taille 7×7
couche 2 : 5 filtres de taille 9×9
couche 3 : 4 filtres de taille 11×11
convolution « valid »
4 classes de sortie



$$3 \times 7 \times 7 \times 3 + 5 \times 9 \times 9 \times 3 + 4 \times 11 \times 11 \times 5 + 256 \times 4$$

Image: **32x32x3**
Stride : **1**

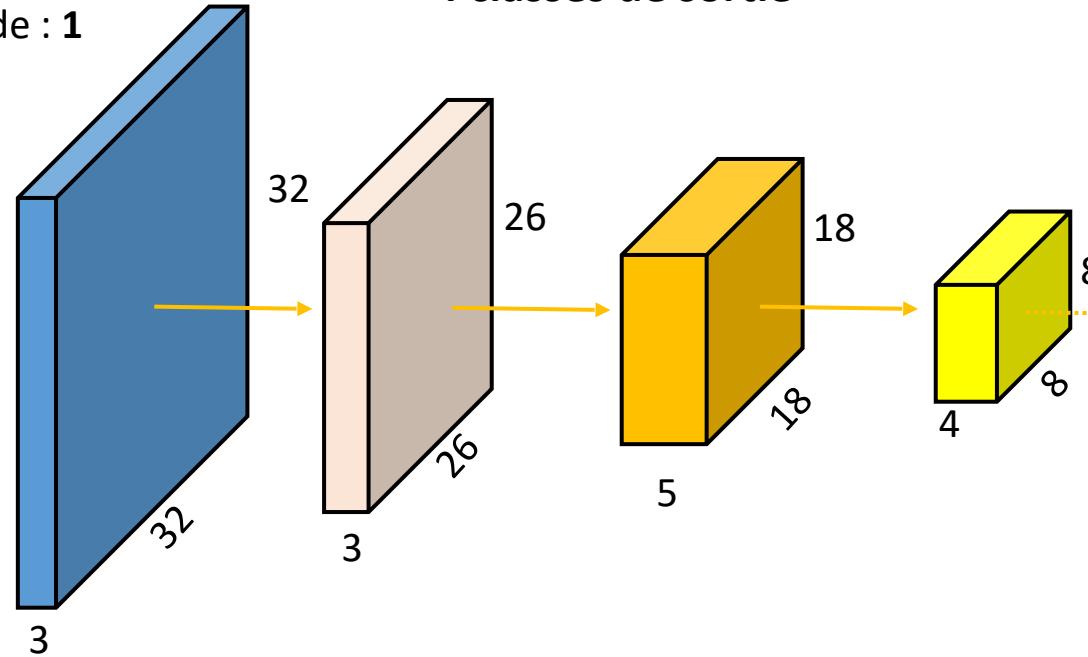
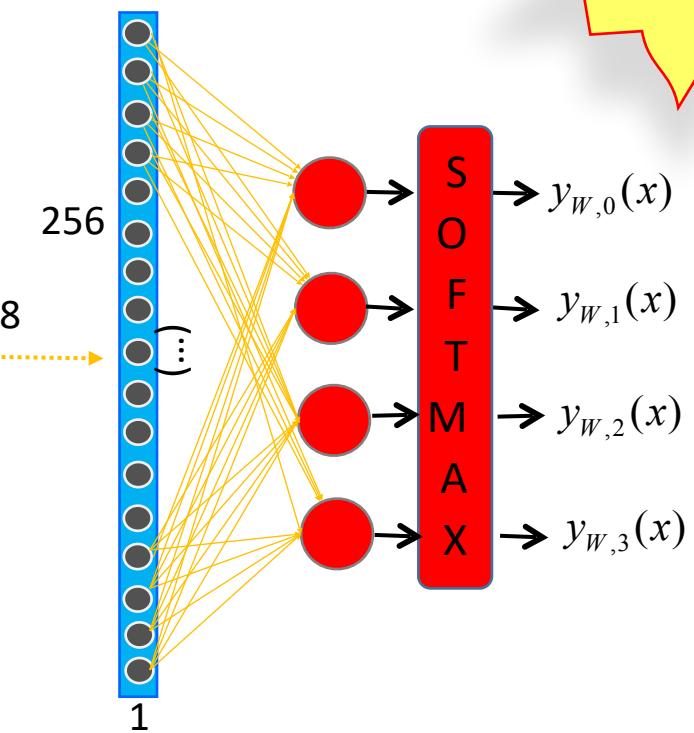


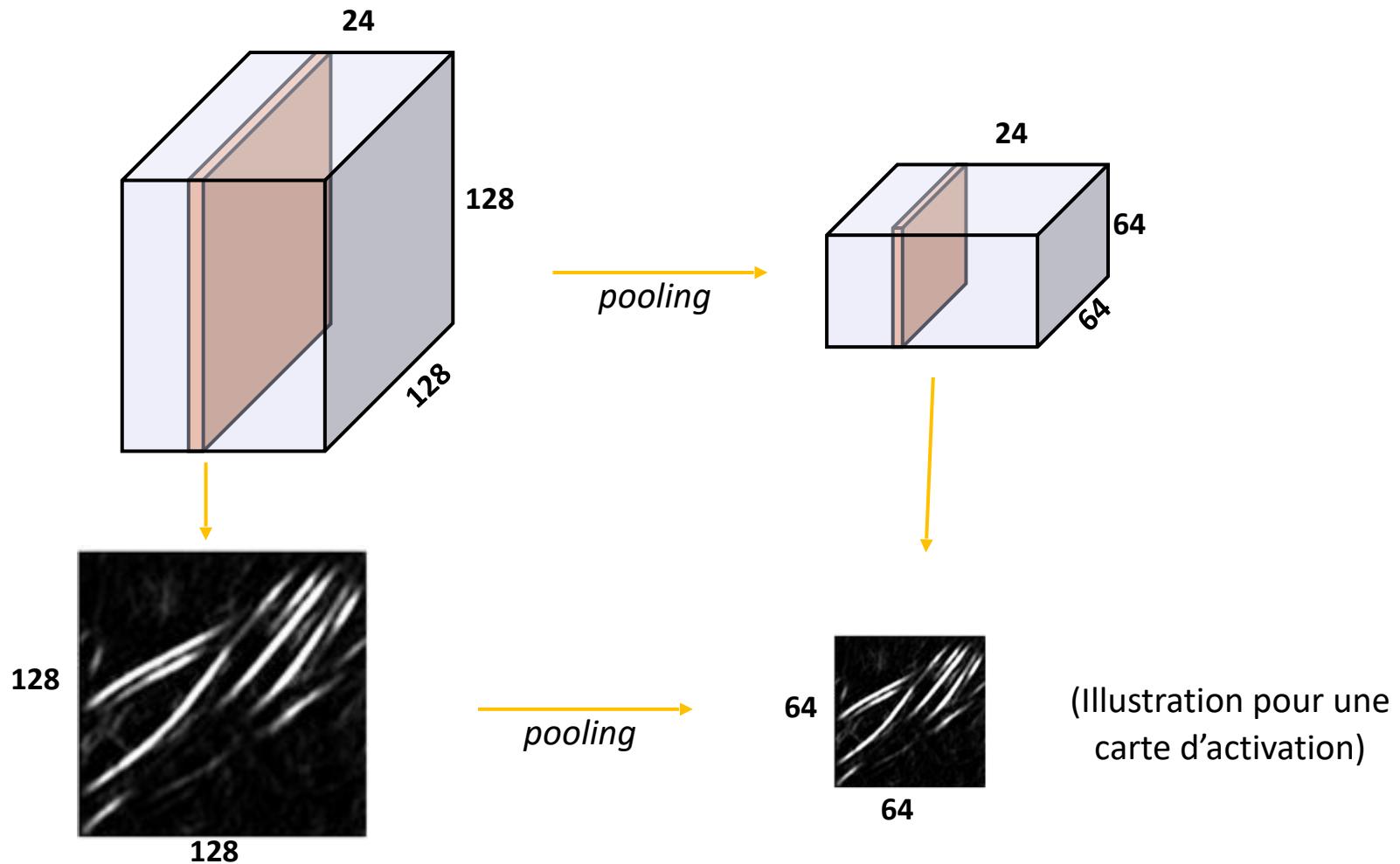
Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « *valid* »
4 classes de sortie



5,100 paramètres

Pooling

Réduction de la taille des cartes d'activation



Max pooling

1	2	4	4	9	3	1	2
6	7	8	4	-3	-3	6	3
9	-9	8	-4	5	5	3	0
8	-8	9	-9	5	5	0	1
0	0	1	2	7	9	7	8
-1	-3	3	6	8	8	7	6
9	9	8	2	1	5	-1	-1
1	1	-2	8	3	7	4	-2

Max pool par filtre
2x2 avec stride =2

7	8	9	6
9	9	5	3
0	6	9	8
9	8	7	4

Mean pooling

1	2	4	4	9	3	1	2
6	7	8	4	-3	-3	6	3
9	-9	8	-4	5	5	3	0
8	-8	9	-9	5	5	0	1
0	0	1	2	7	9	7	8
-1	-3	3	6	8	8	7	6
9	9	8	2	1	5	-1	-1
1	1	-2	8	3	7	4	-2

Moyenne par filtre
2x2 avec stride =2

4	5	3	4
0	1	5	1
-1	8	8	7
5	4	4	1

Global pooling

Max ou Mean pooling « valid » avec un filtre de la taille des canaux

Résultat : un **vecteur** de la taille du nombre de canaux

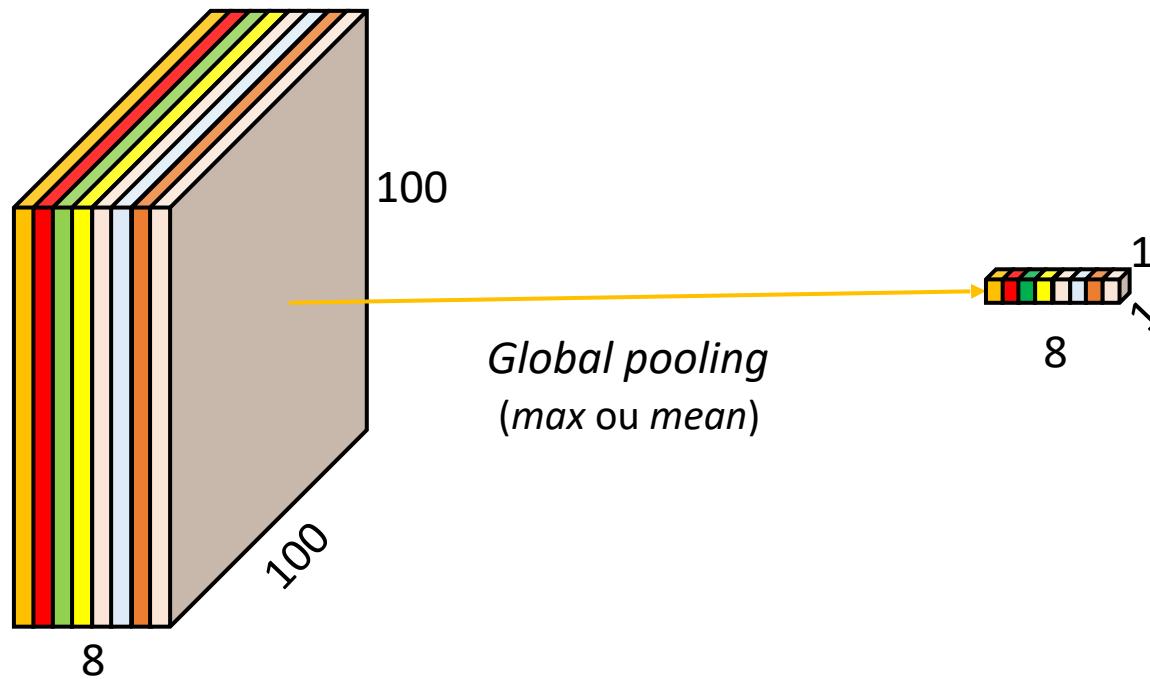
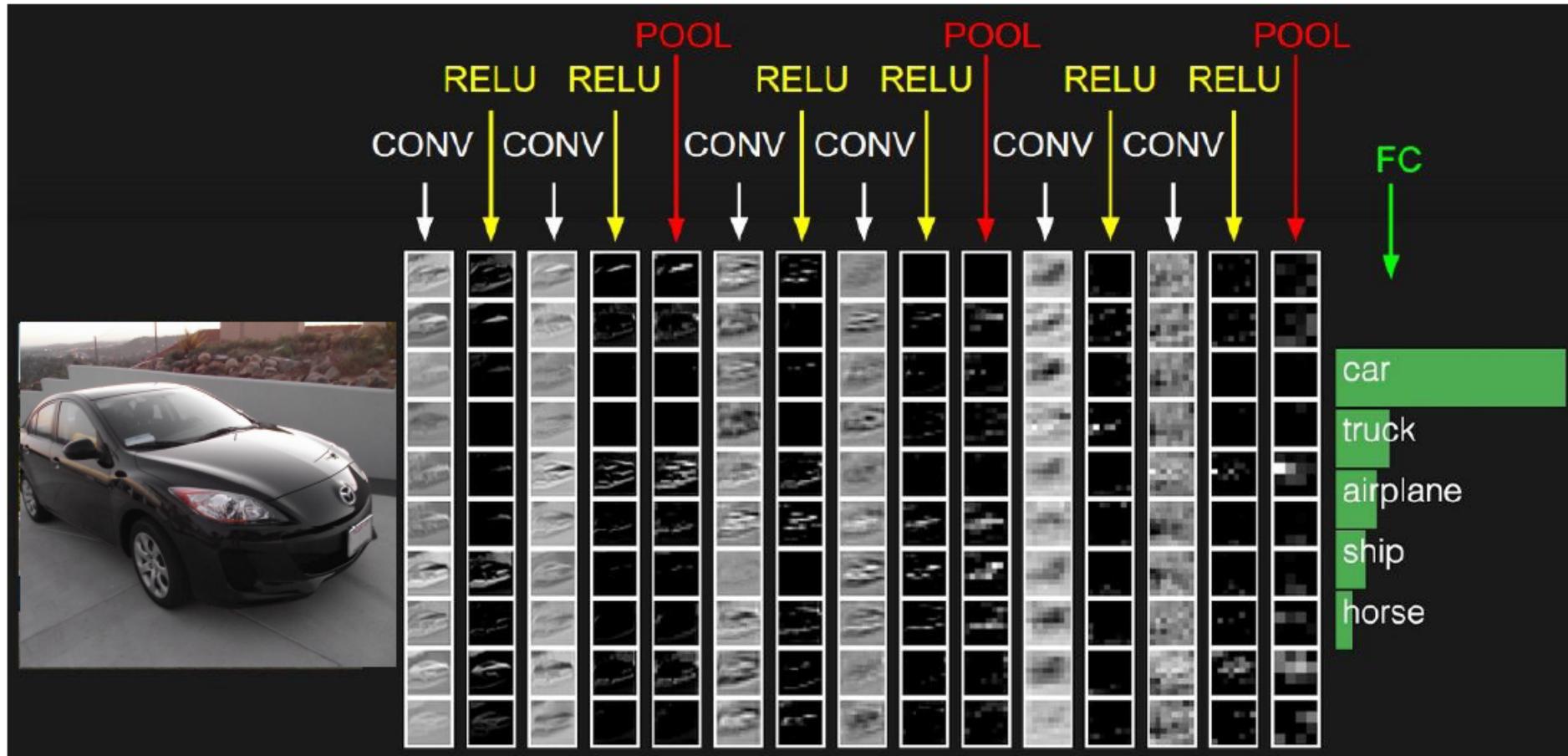


Illustration d'un CNN complet



Crédit : cs231 Stanford