EPSI l'École d'ingénierie informatique

Réseaux Convolutifs

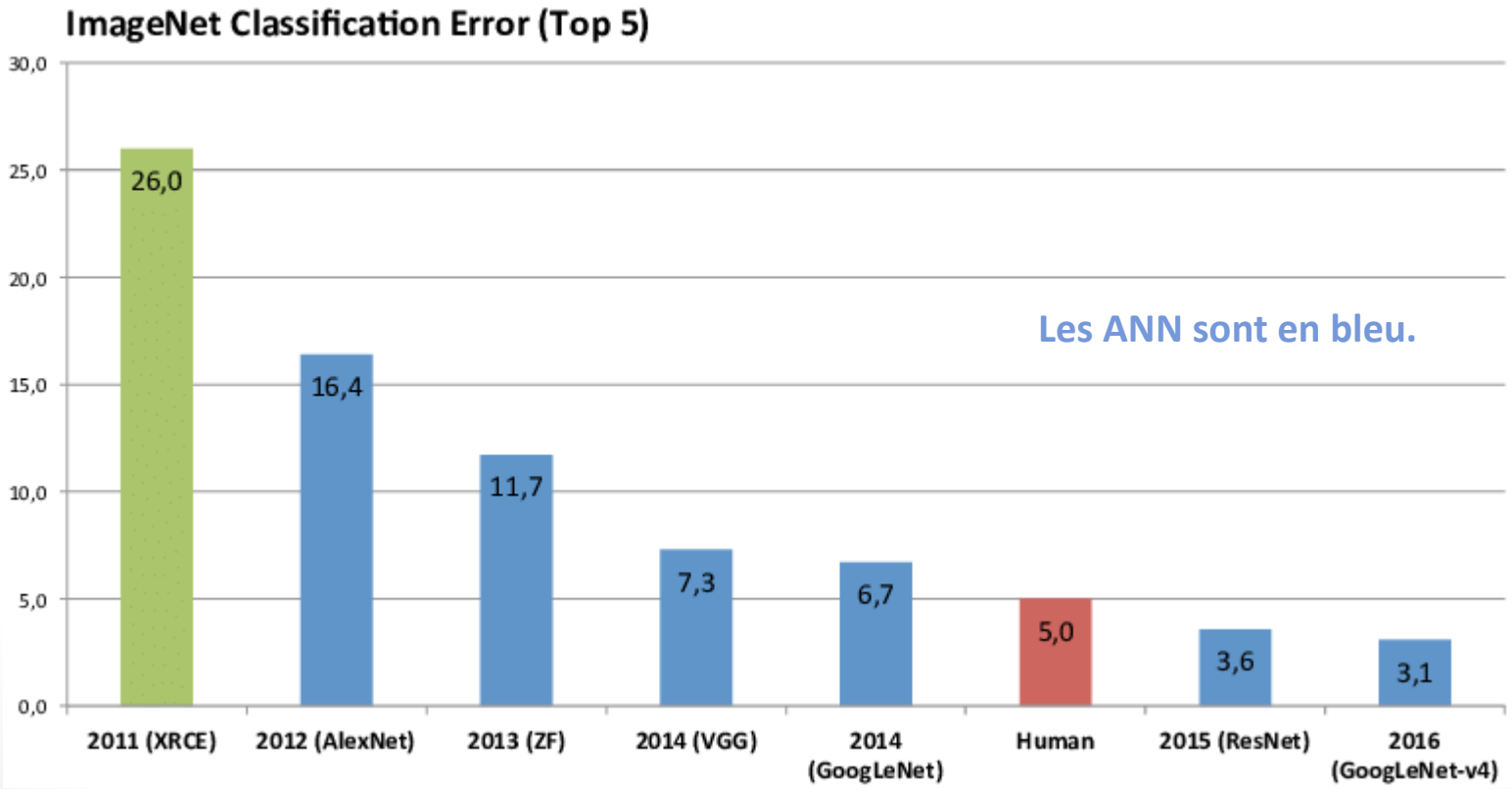
ConvNet ou CNN : Convolutional Networks

Jean-Claude Houbart

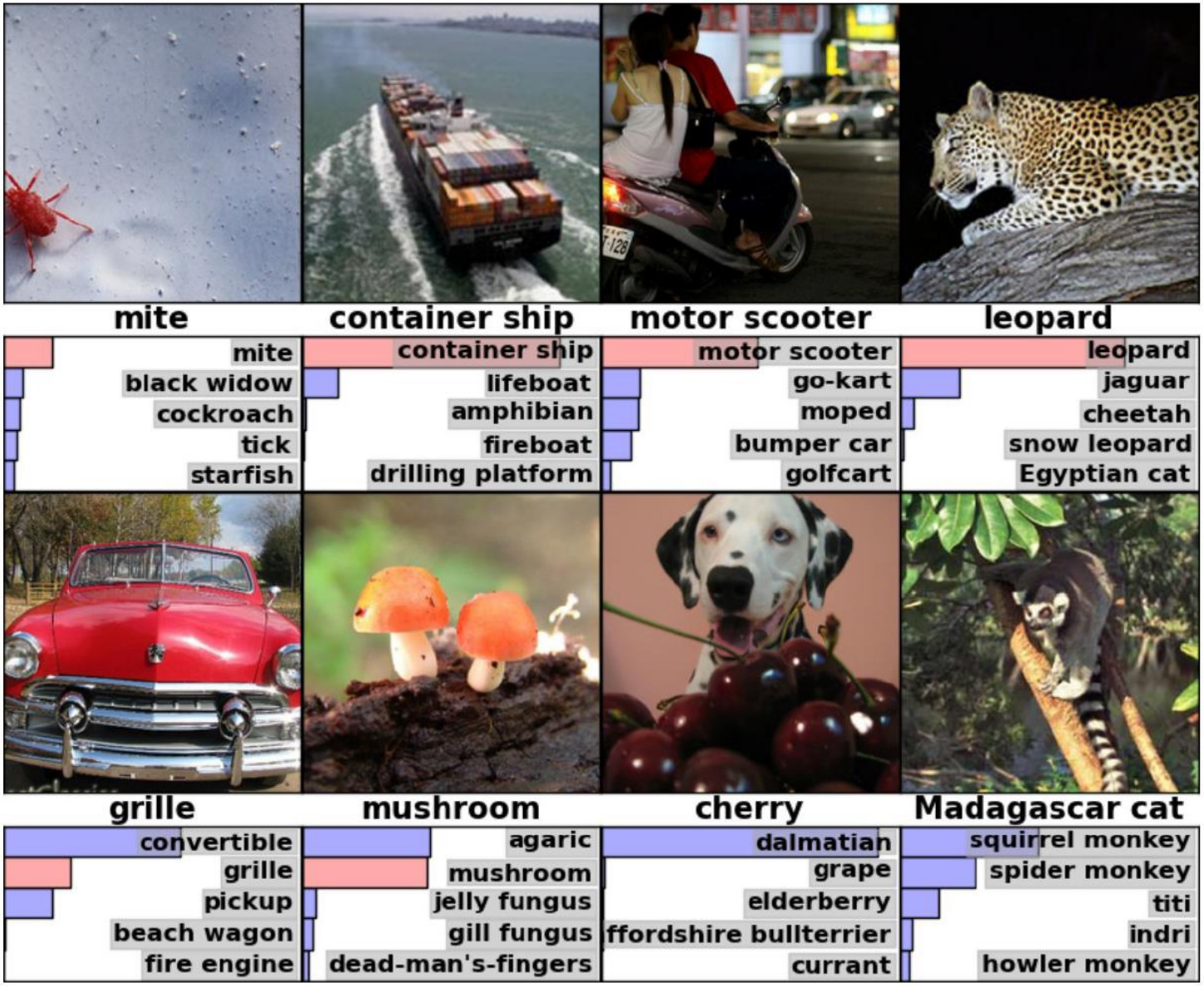
RAPPELS

2012 : Le Choc

- ImageNet Large Scale Visual Recognition Challenge est une compétition annuelle de reconnaissance d'image basée sur 10 millions d'images annotées et 1000 labels.
- En 2017 repris pas Kaggle.

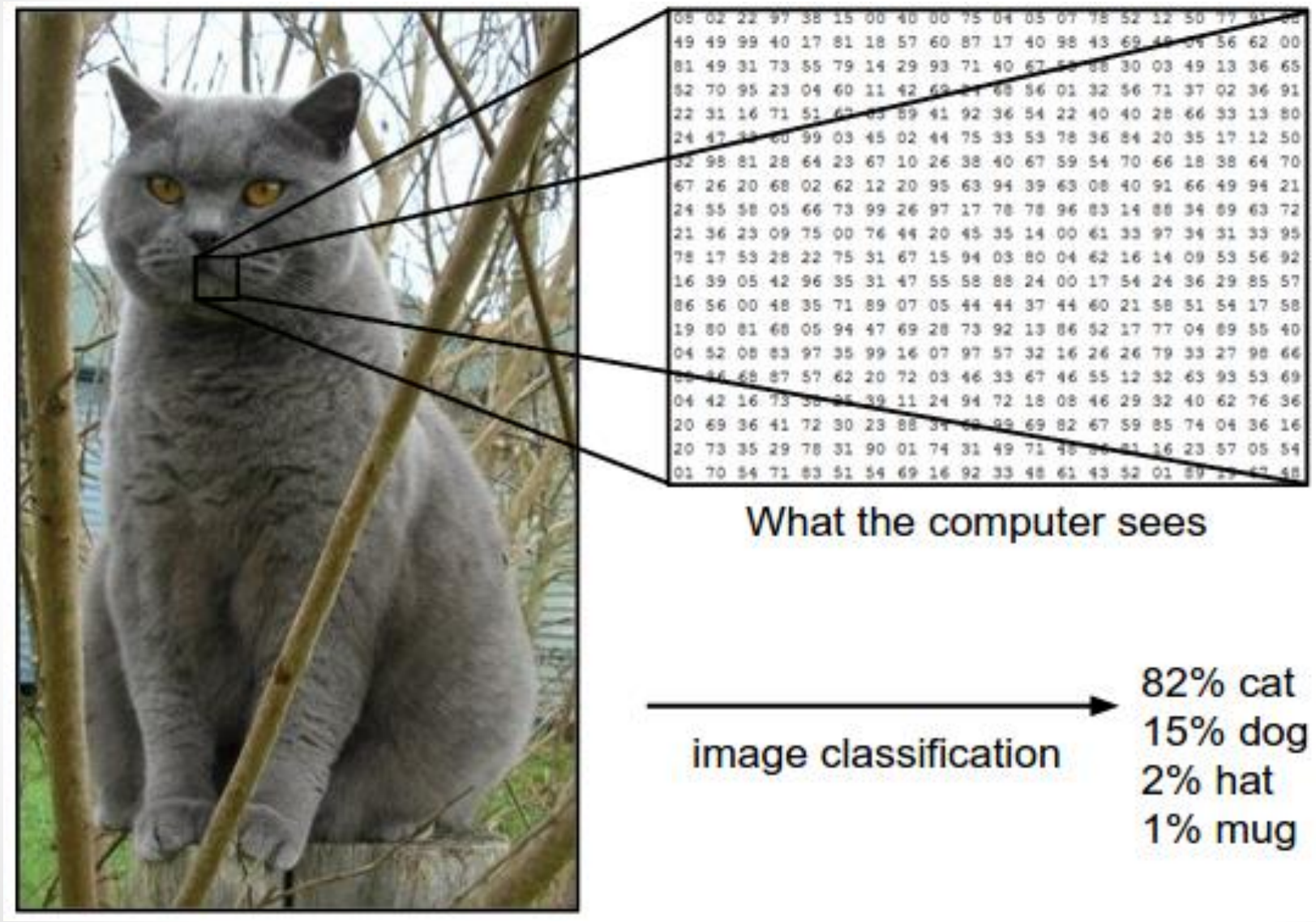


ImageNet


mite
mite
black widow
cockroach
tick
starfish
container ship
container ship
lifeboat
amphibian
fireboat
drilling platform
motor scooter
motor scooter
go-kart
moped
bumper car
golfcart
leopard
leopard
jaguar
cheetah
snow leopard
Egyptian cat
grille
convertible
grille
pickup
beach wagon
fire engine
mushroom
agaric
mushroom
jelly fungus
gill fungus
dead-man's-fingers
cherry
dalmatian
grape
elderberry
ffordshire bullterrier
currant
Madagascar cat
squirrel monkey
spider monkey
titi
indri
howler monkey

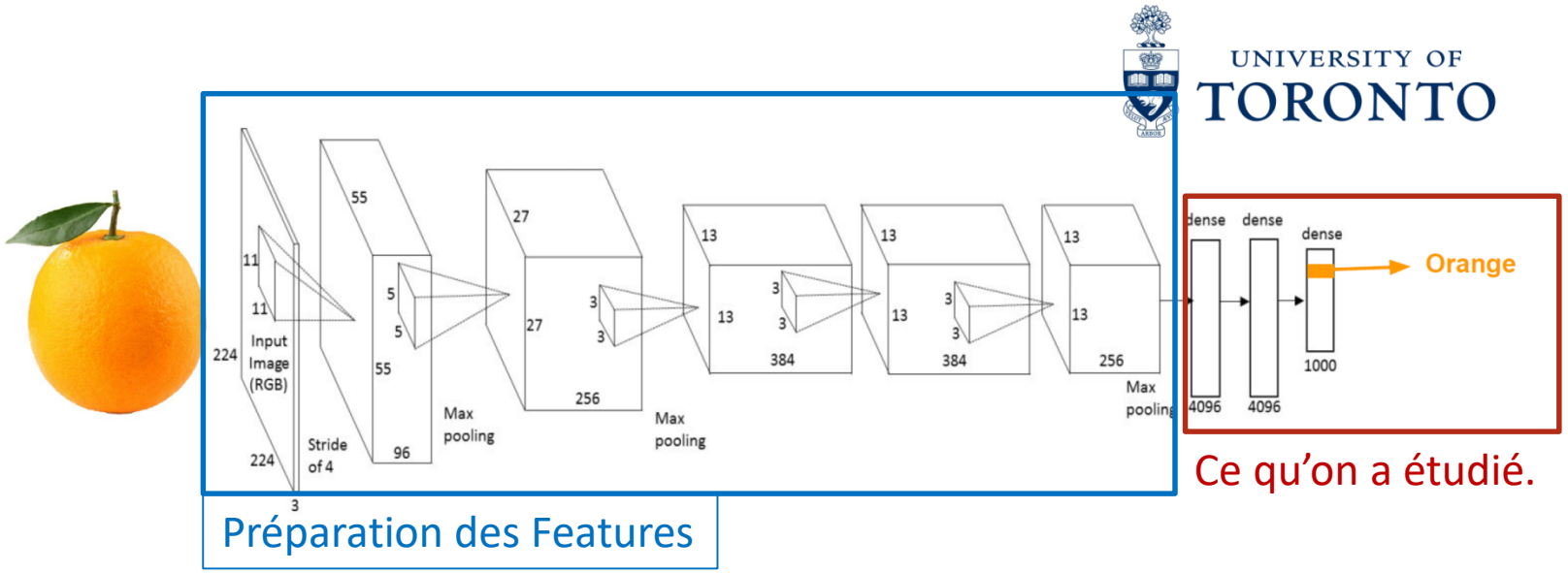
I. RÉSEAUX DE NEURONES CONVOLUTIFS

Une image vue par un ANN



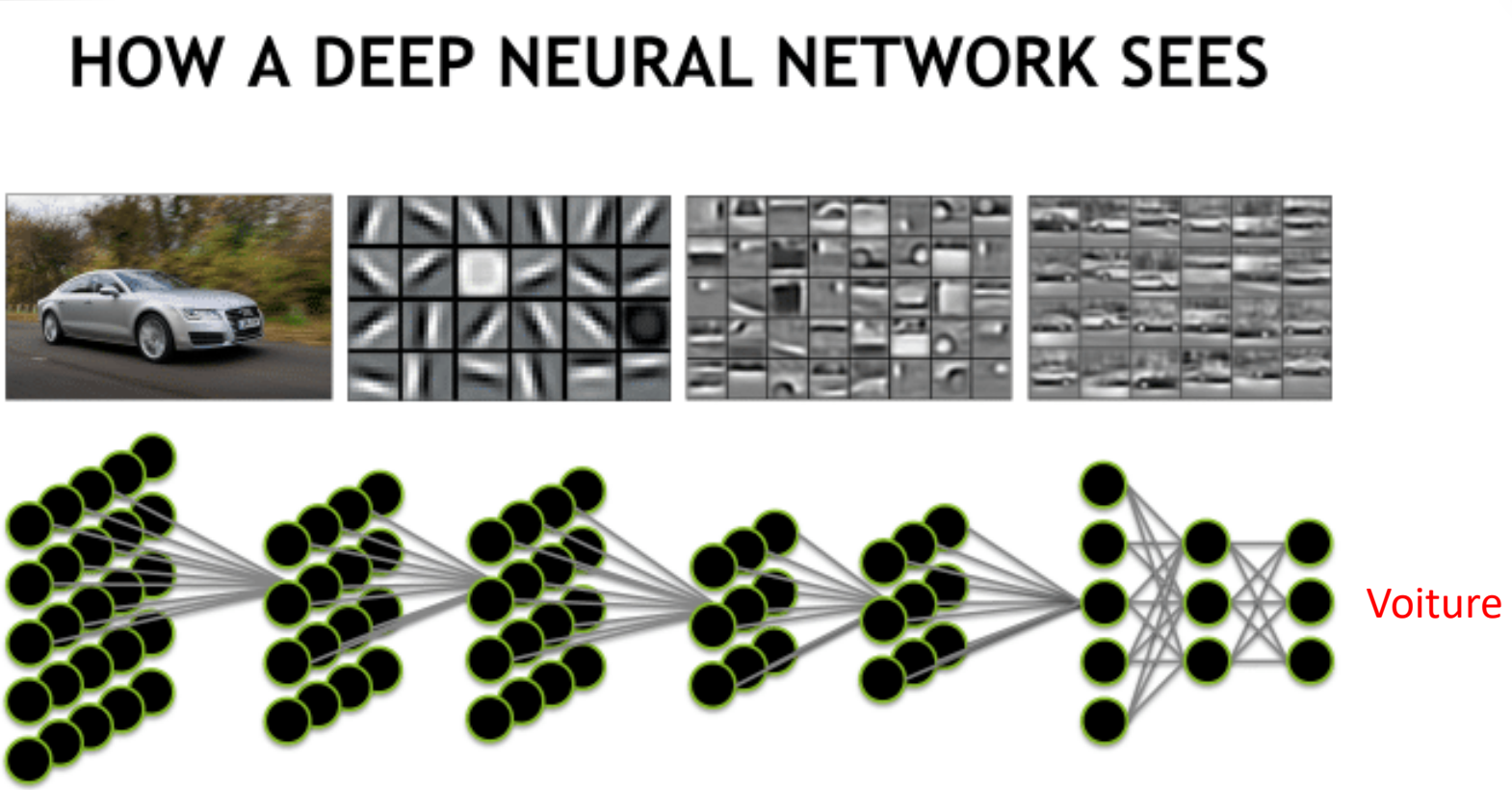
Couches de convolution et de Pooling

- Dans AlexNet, le réseau ANN proprement dit est précédé de deux autres types de couches : Convolution et Pooling.
- Idée : Apprendre les features.



A Krizhevsky, I Sutskever, GE Hinton "Imagenet classification with deep convolutional neural networks" NIPS 2012

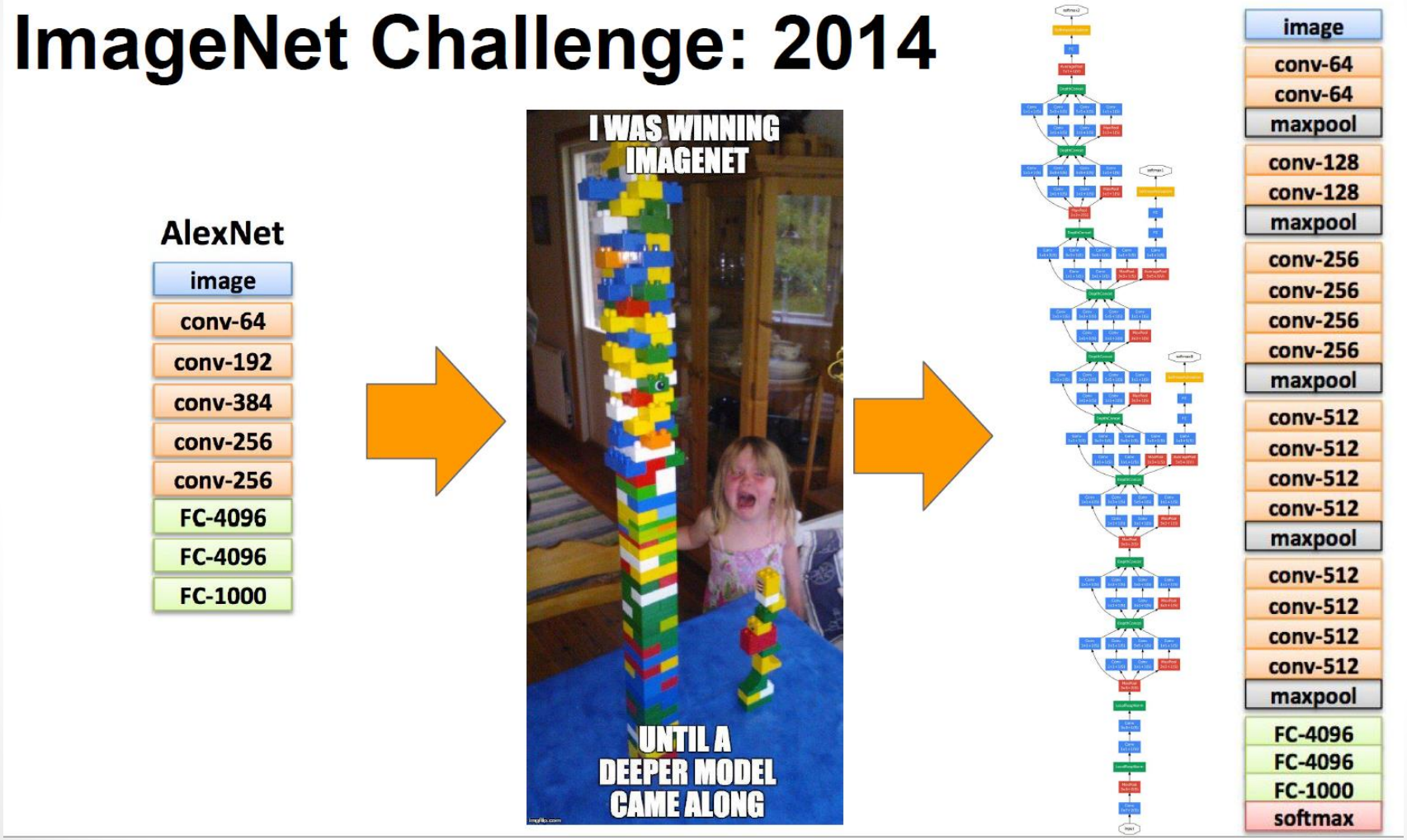
Préparation des Features



- La premier partie de ce type de réseaux de neurones fonctionne comme un **extracteur de features**.
- Elle applique plusieurs opérations de filtrage par **convolution**.
- Les paramètres de filtrages sont appris pendant la backpropagation.

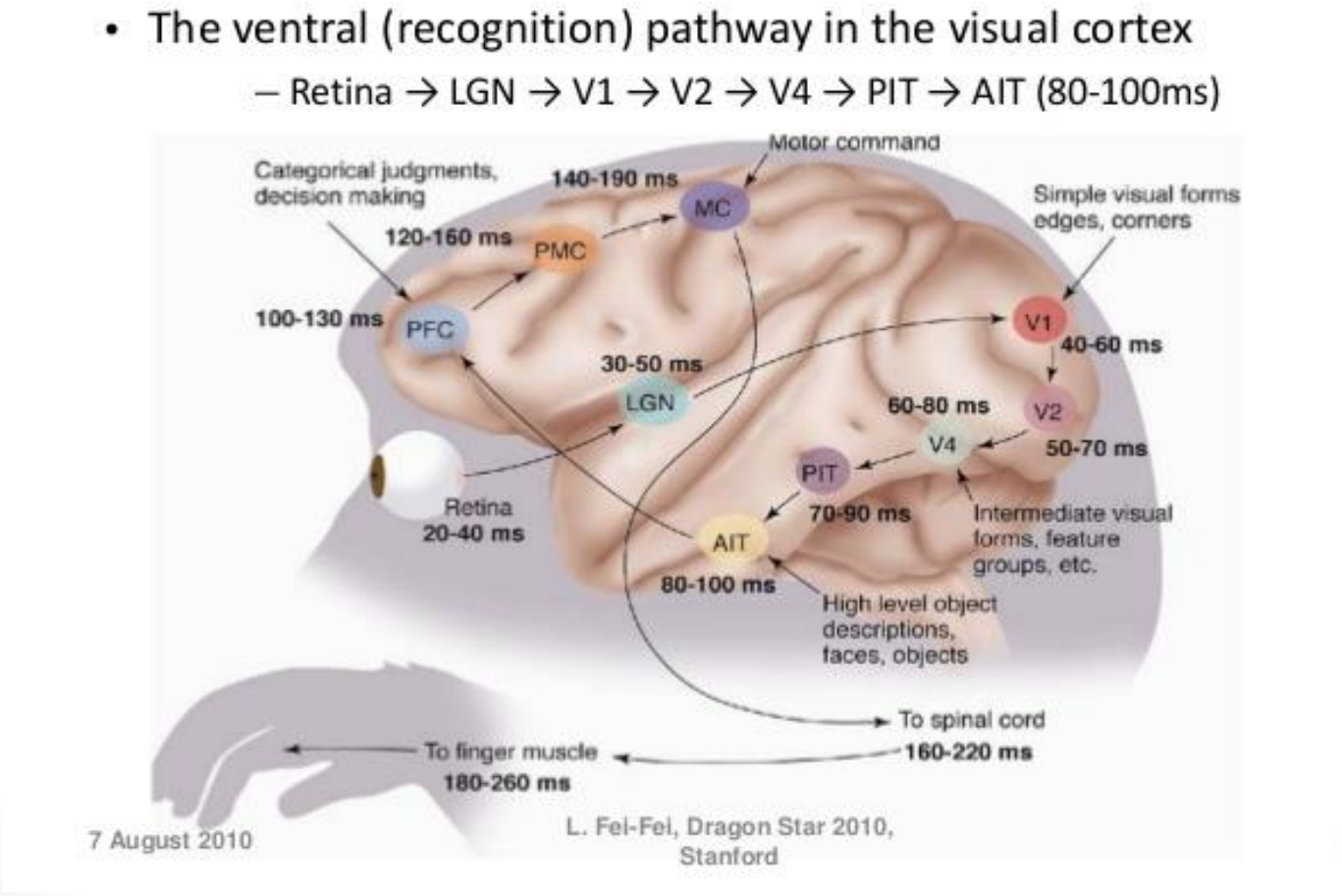
Augmentation du nombre de couches

- On assiste à une surenchère dans le nombre de couches.



Encore la métaphore biologique

- Les réseaux convolutifs imitent le fonctionnement de la vision dans le cerveau.



Apprentissage des Features

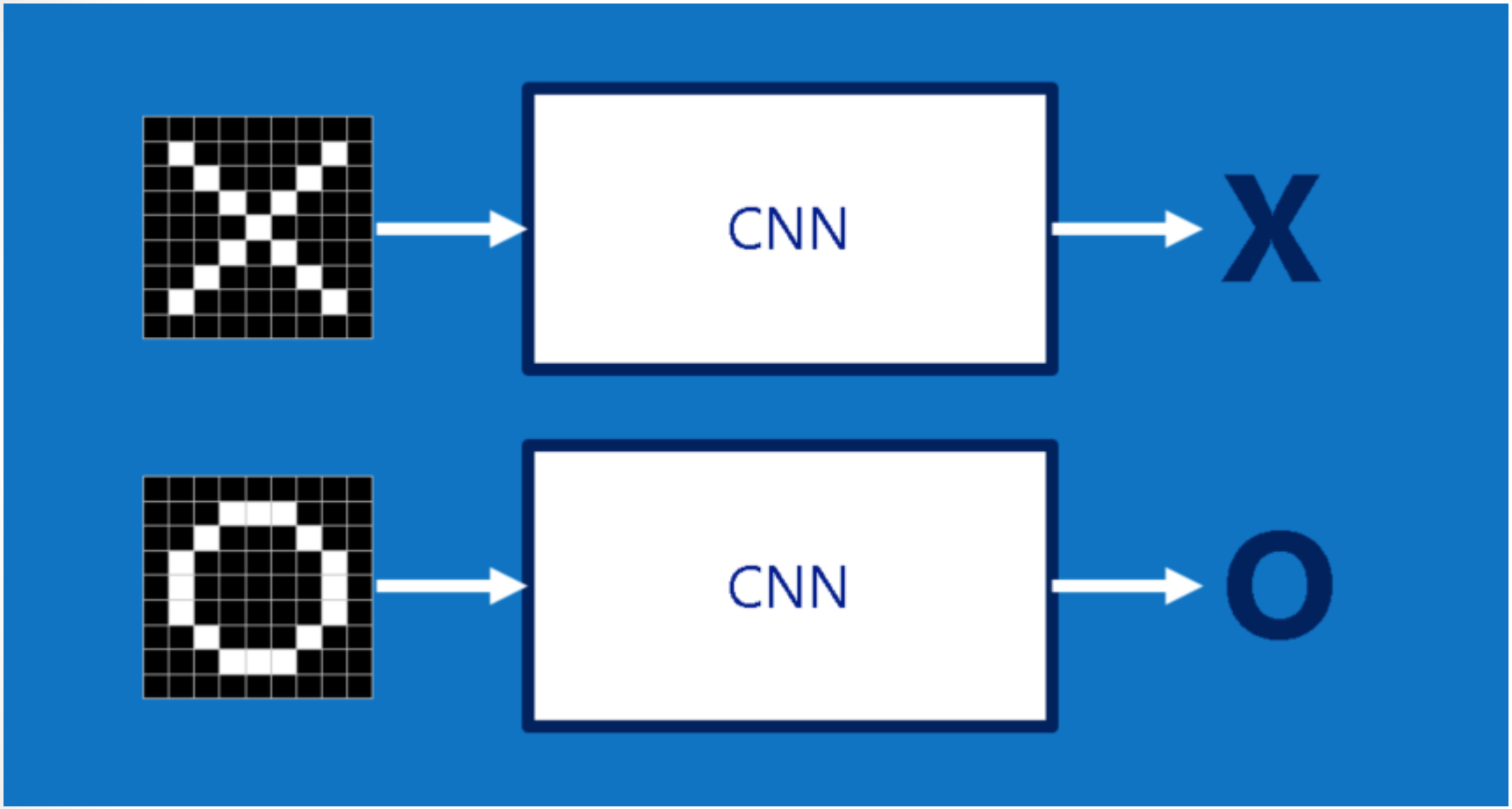
- Les features ne sont pas pré-définies, mais **apprises par le modèle** lors la phase d'entraînement

- Les **noyaux de filtre** désignent les poids de la **couche de convolution**. Ils sont initialisés puis mis à jour par rétropropagation du gradient.

- Les CNN sont capables de déterminer tout seul les éléments discriminants d'une image, en s'adaptant au problème posé.

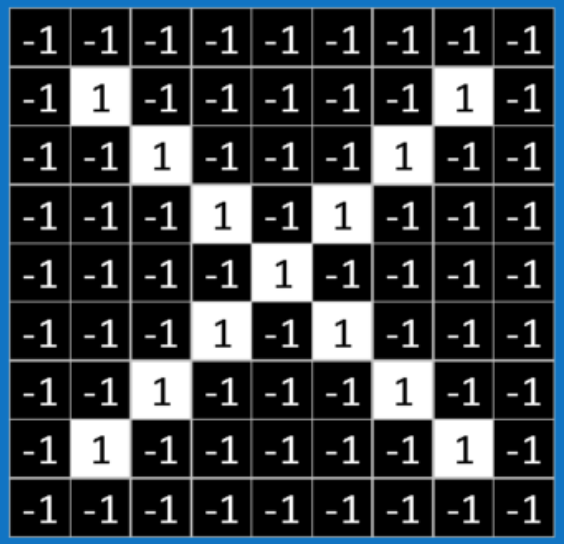
- Par exemple, si la question est de distinguer les chats des chiens, les features appris automatiquement peuvent décrire la forme des oreilles ou des pattes. …

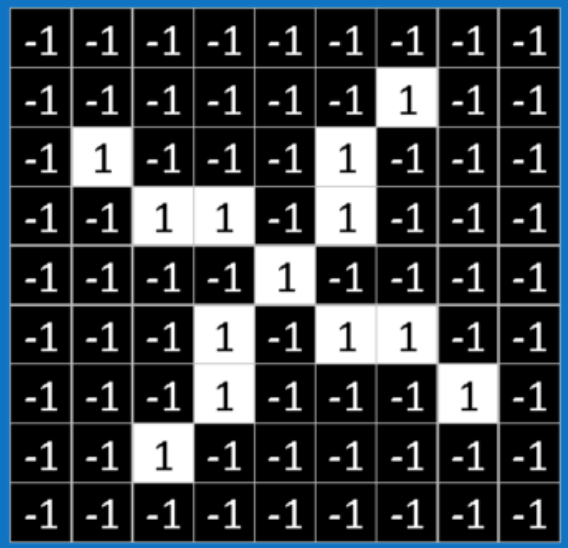
Exemple : Distinguer les ‘X’ et les ‘O’



- Source : “How Convolutional Neural Networks Work”. **Brandon Rohrer** (Senior Data Scientist à Facebook)

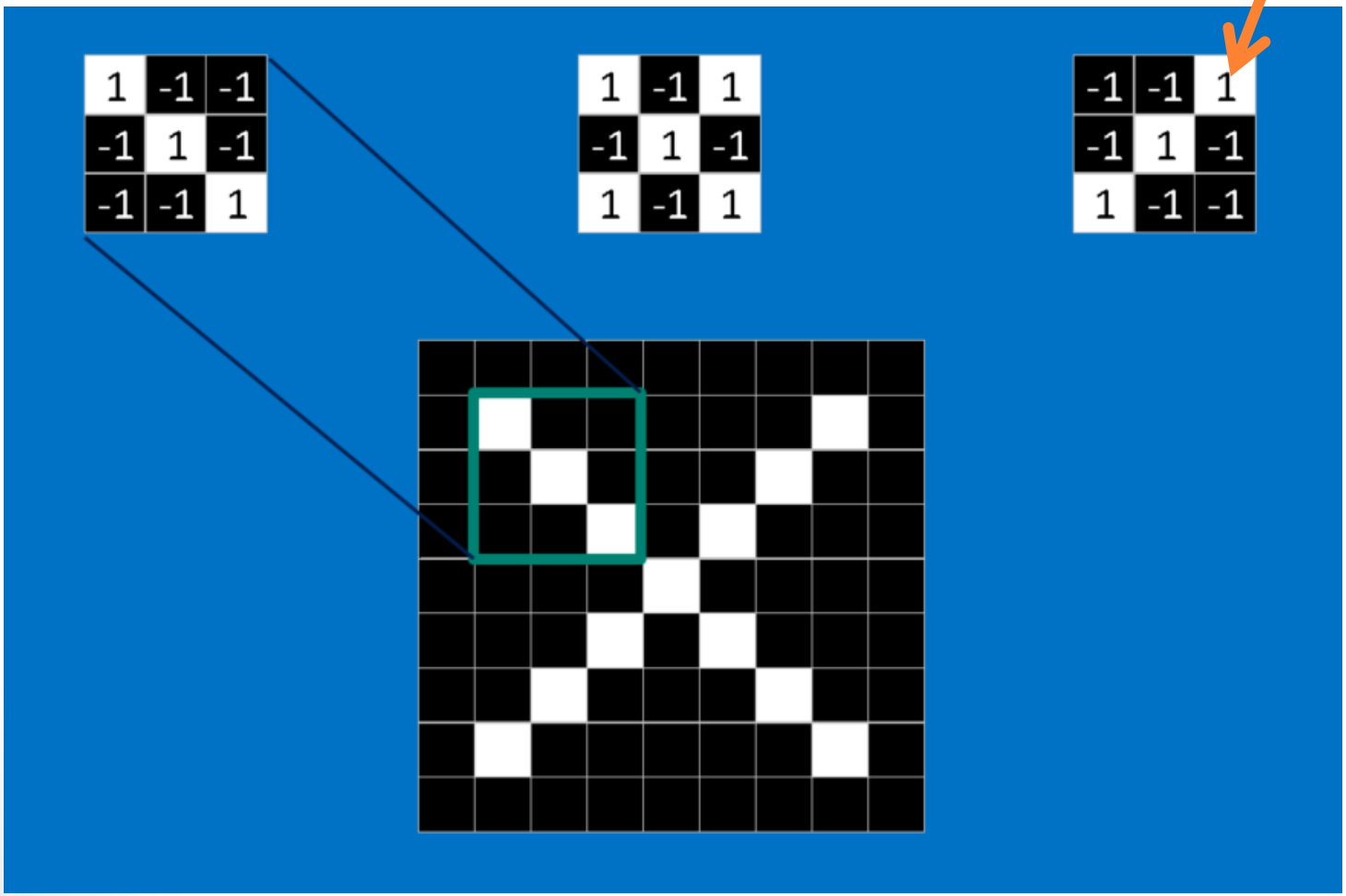
- Problème : Comment reconnaître les images déformées, pivotées, redimensionnées….
- Dans notre exemple, chaque pixel vaut 1 ou -1.





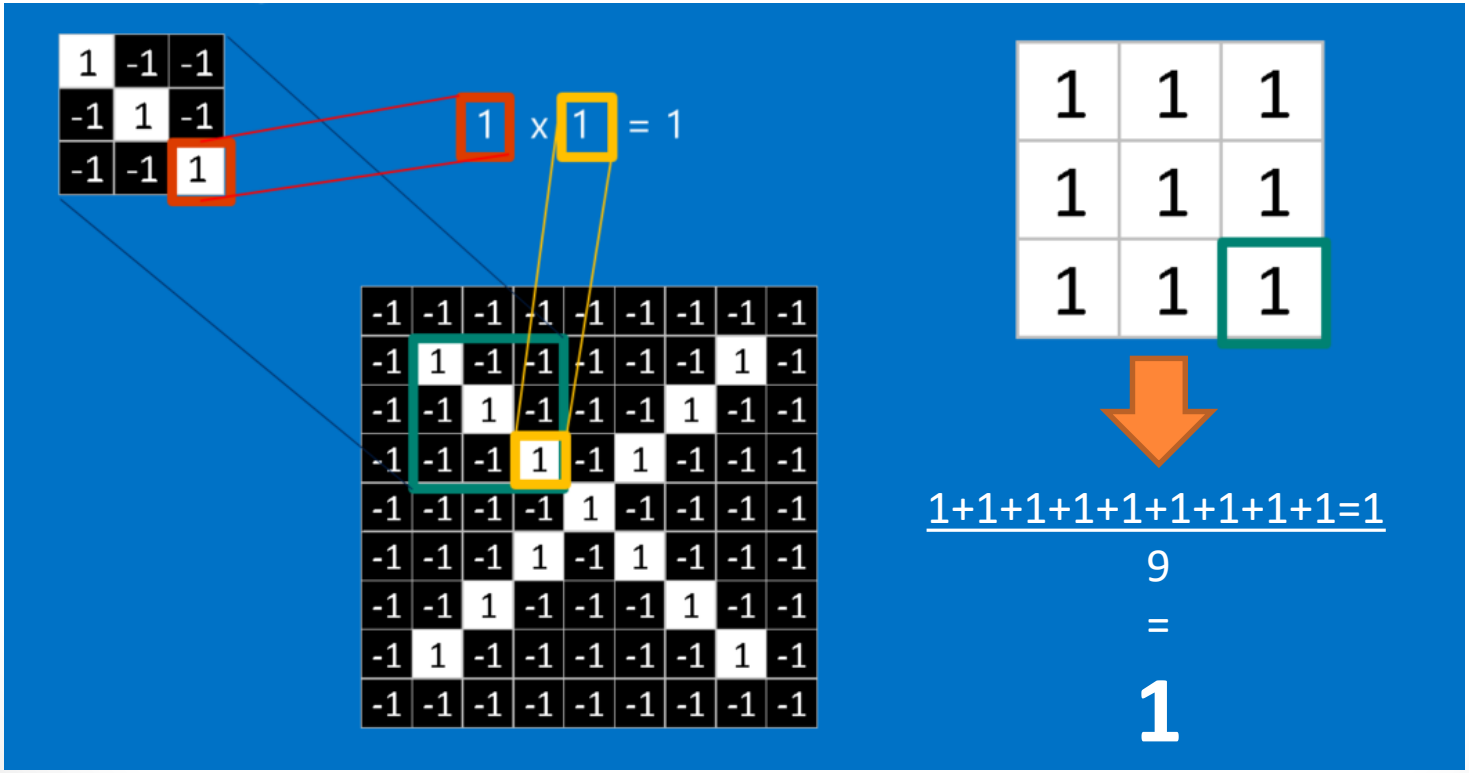
Fragments intéressants

- Le modèle va **apprendre** des fragments utiles pour reconnaitre des « X ».

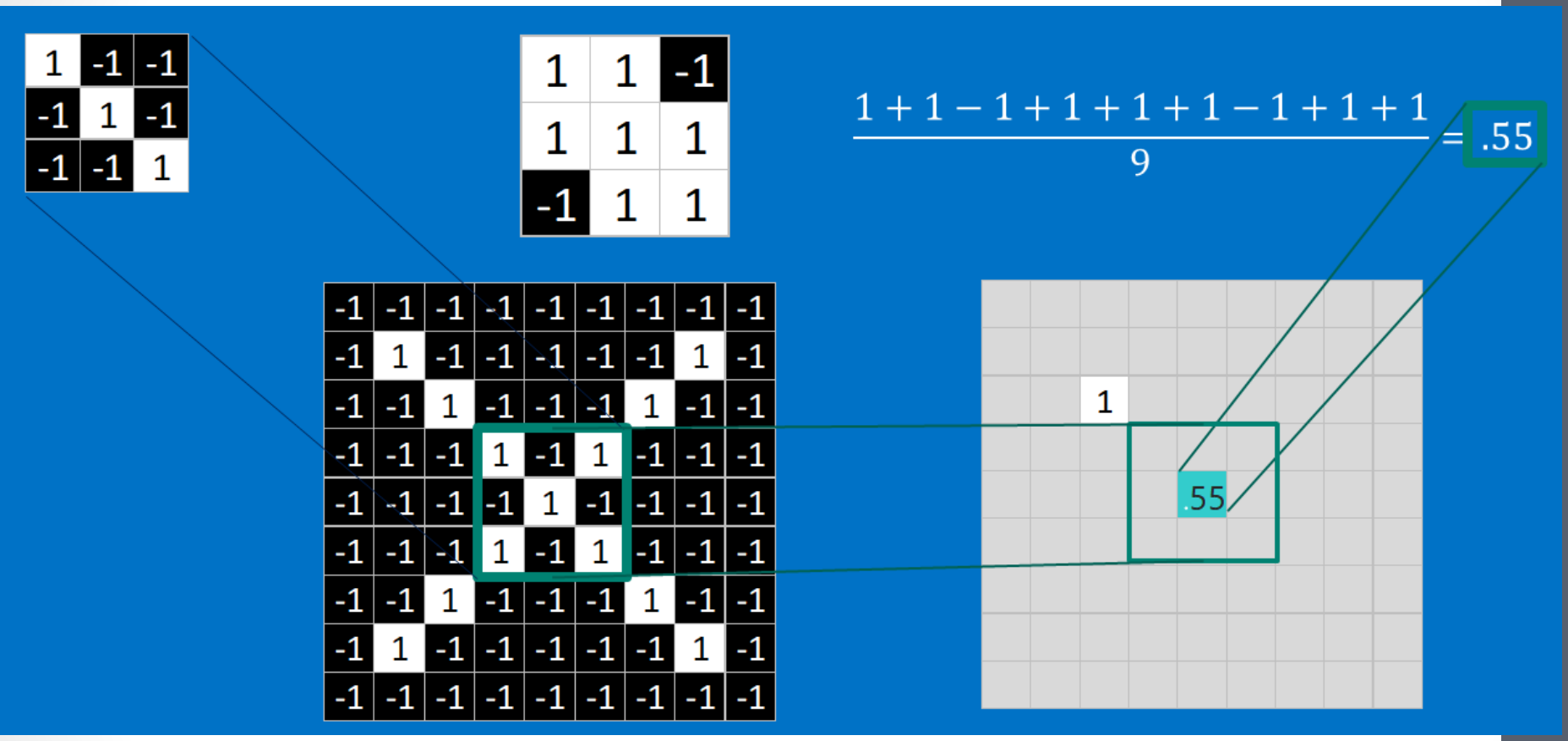


Opération de Convolution

- On multiplie pixel par pixel le filtre et une zone de l'image.
- On obtient, une matrice de la taille de l'image. On en prend la moyenne.

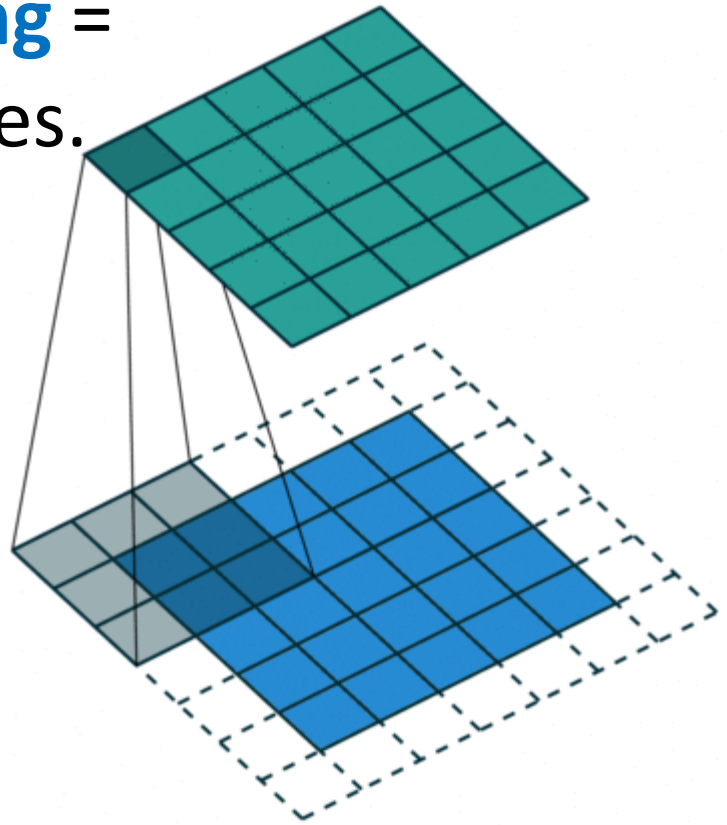


Opération de Convolution



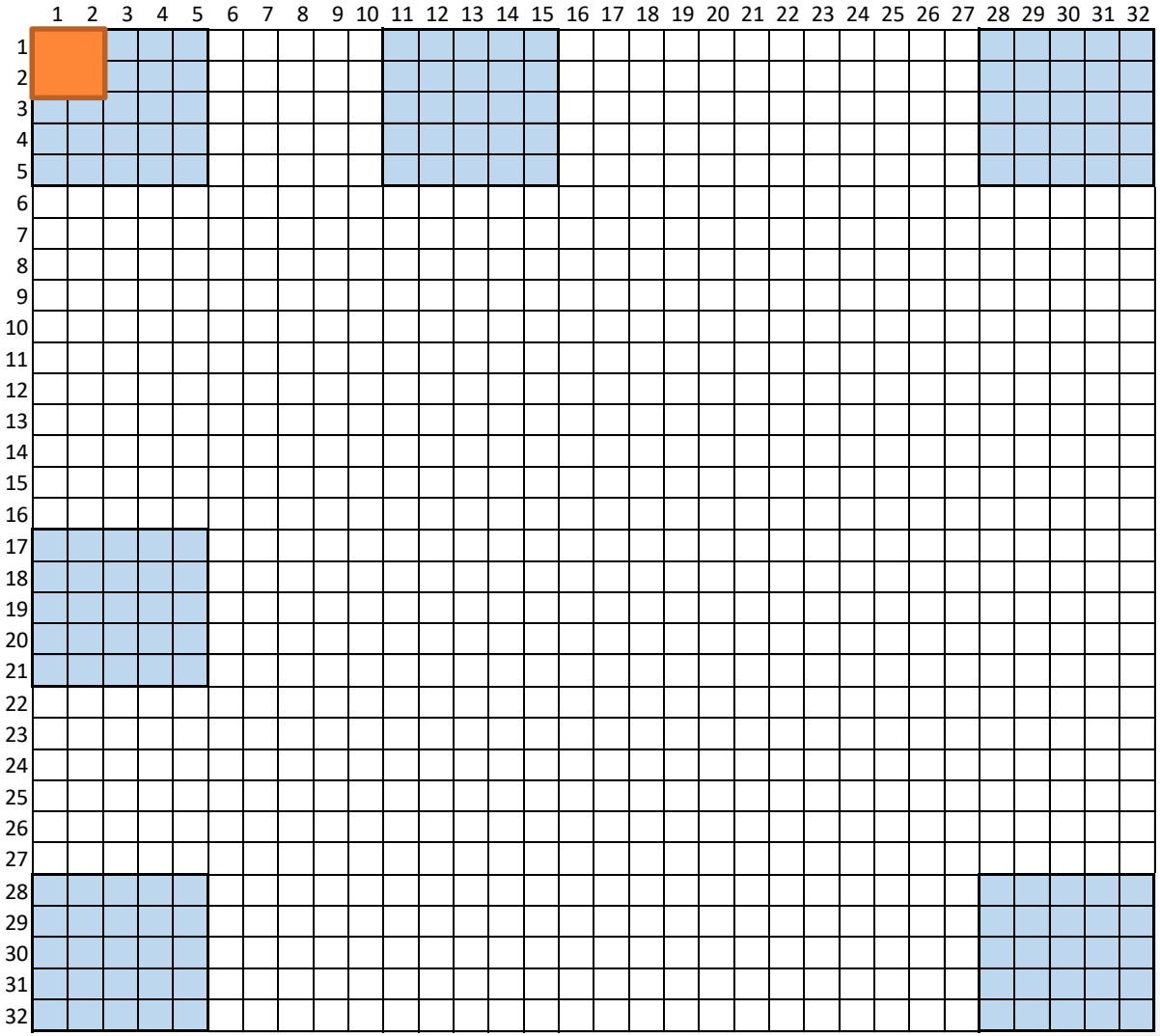
Convolution – Padding - Stride

- On parcourt l'image d'origine (en bleu) et on crée une « carte » appelé « **feature map** » (vert)
- La taille du filtre (3) et le pas entre chaque étape (**padding** = 1) sont des hyper-paramètres.
- Le **stride** est le pas entre chaque projection (1 en général)
- Les deux grilles n'ont pas forcément la même taille.



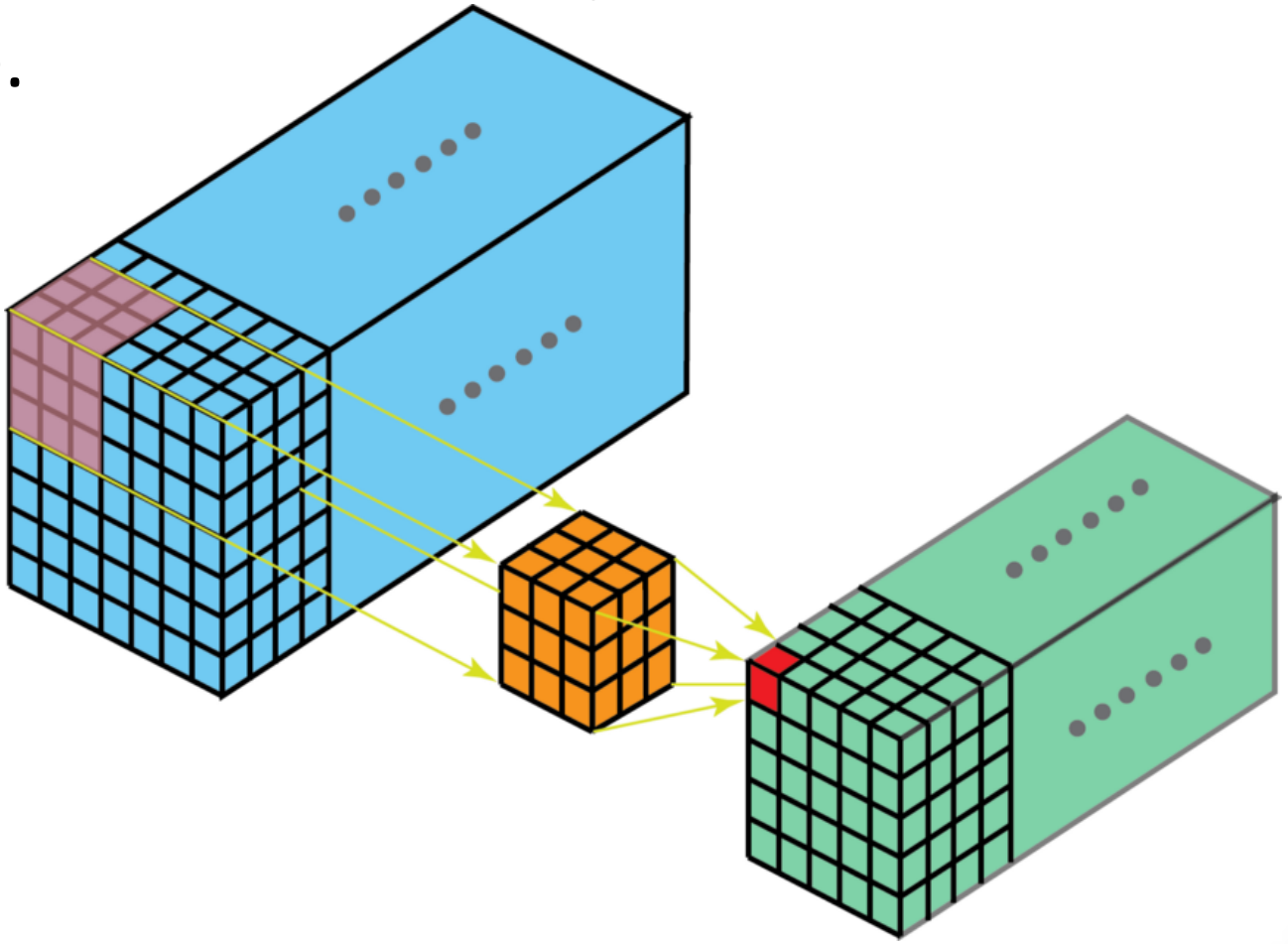
Dimension de sortie

- Sans padding, la grille de résultat est plus petite que celle d'origine.
- Ici on voit que la grille 32 x 32 avec un filtre 5 x 5 donne une grille 28 x 28.

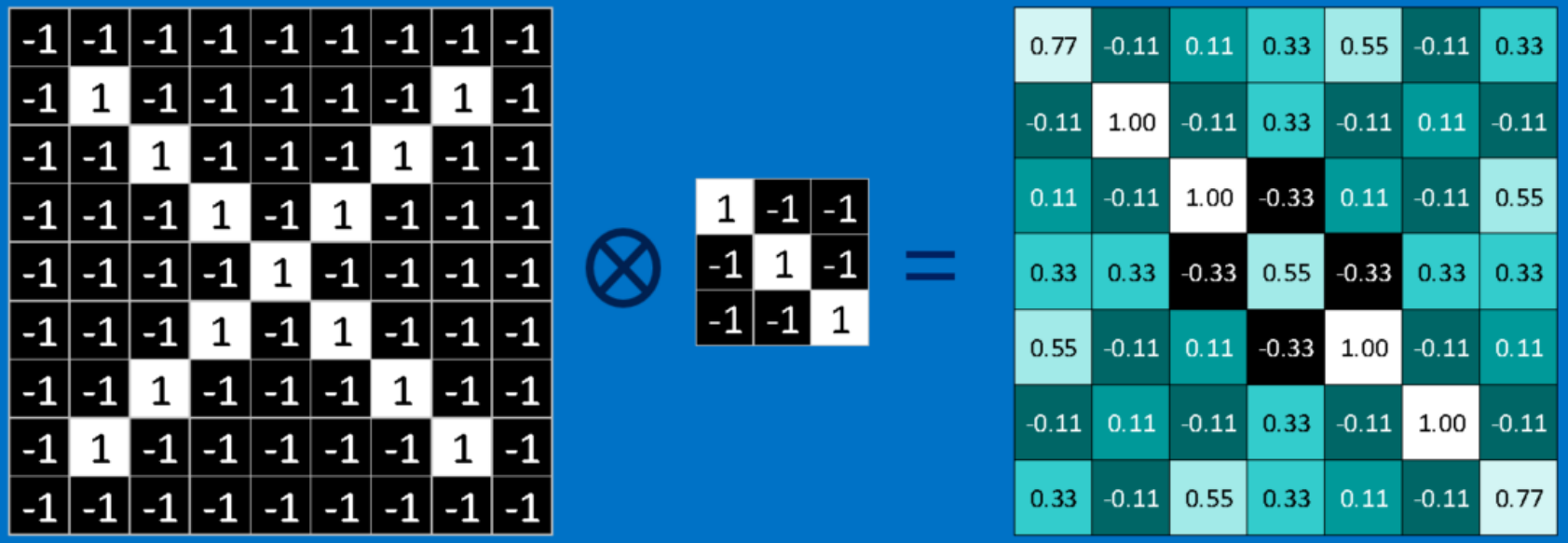


Convolution 3D

- Le noyau de filtre peut être en 3D. On change alors éventuellement la profondeur de la couche.

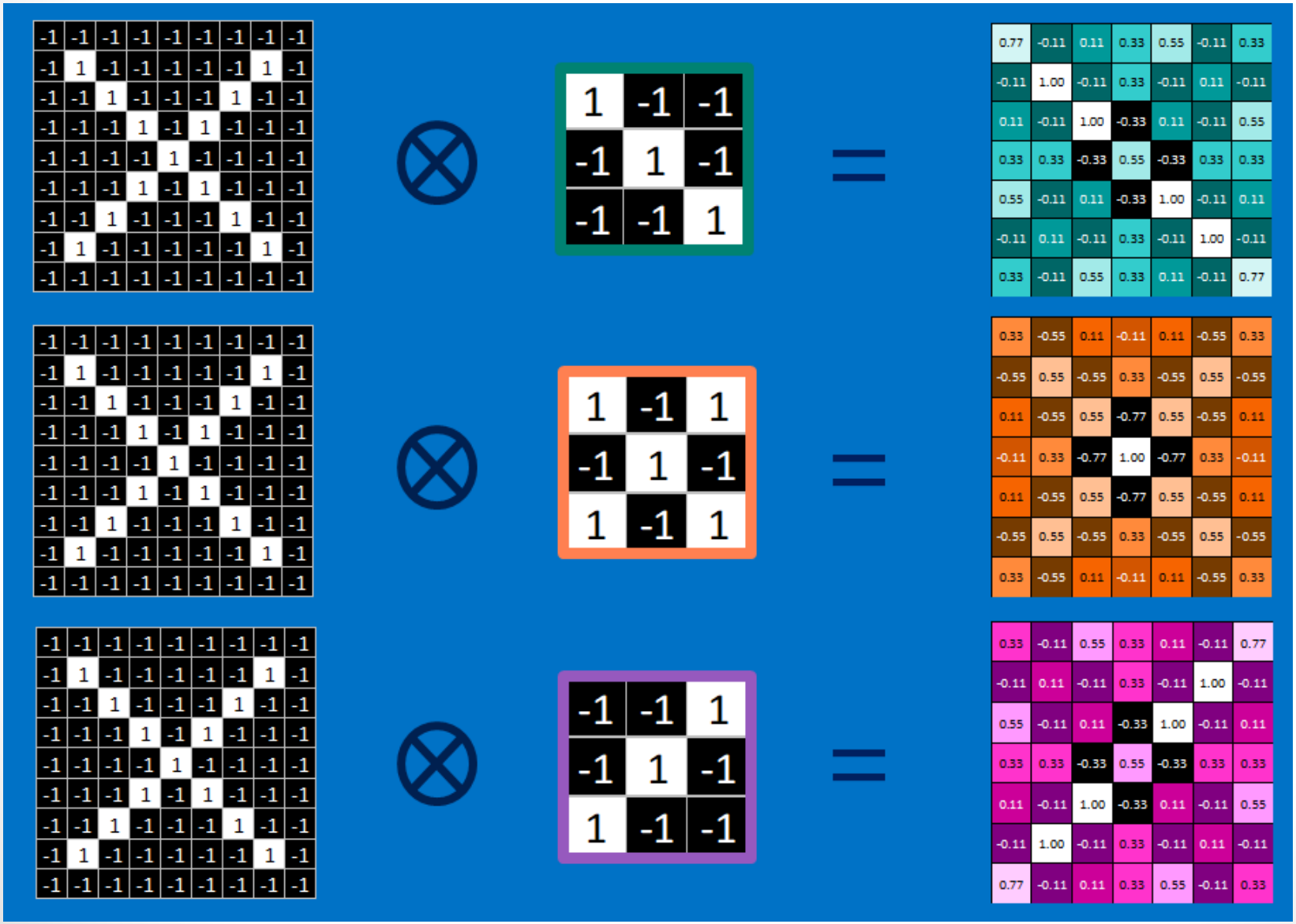


Convolution pour un filtre



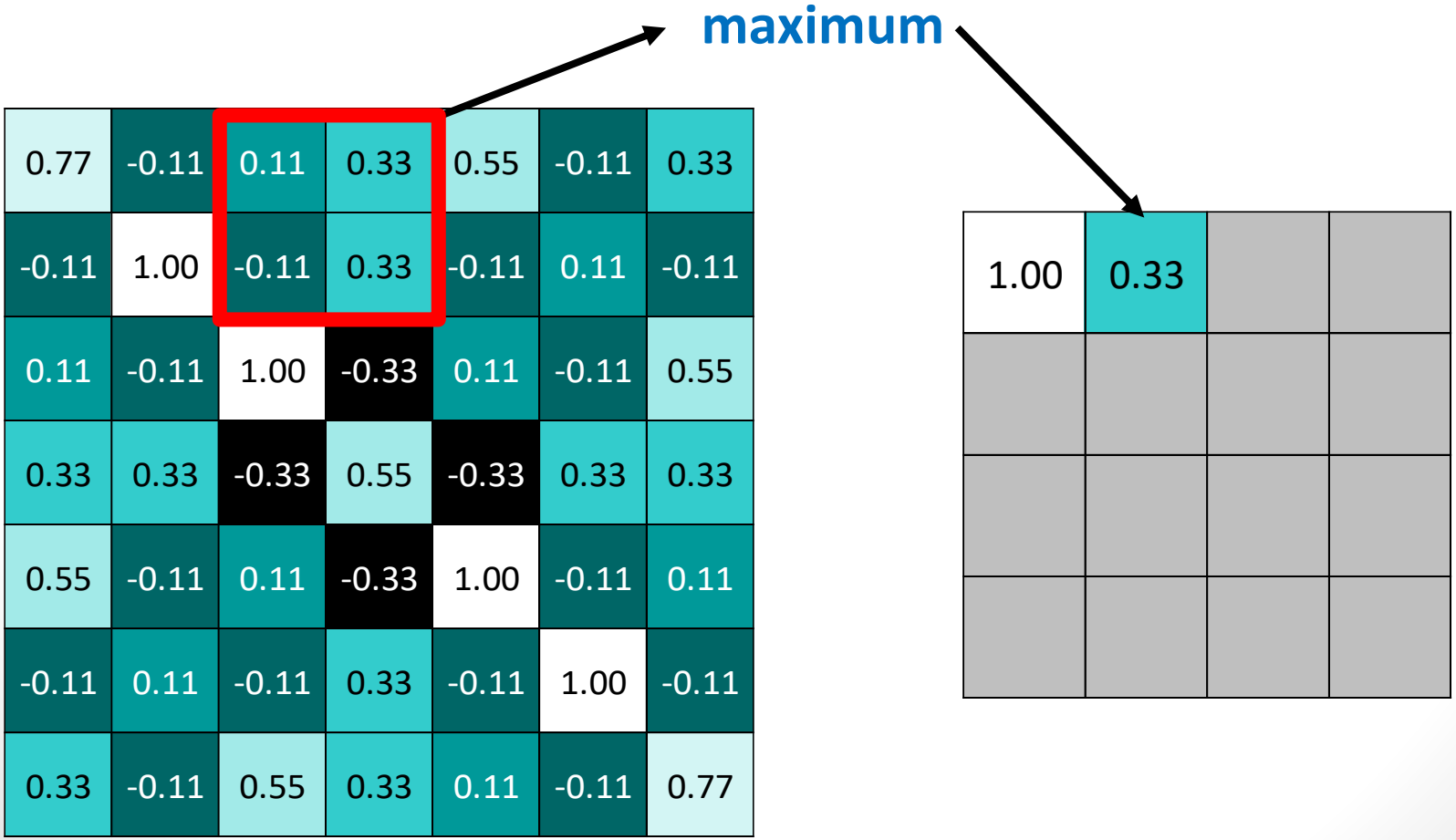
Couche de Convolution

- On applique tous les filtres :



Pooling

- Pour réduire les features map, on va appliquer une opération de **pooling**.
- On utilise généralement la valeur max. La somme est possible.



Pooling d'une feature map

- on utilise souvent une fenêtre de 2 ou 3 pixels de côté et une valeur de 2 pixels le pas.

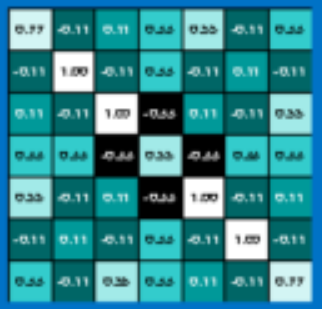
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

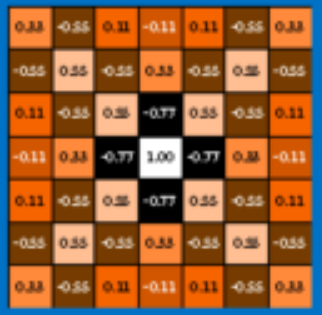
max pooling

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

Couche de Pooling

- Le Pooling est appliqué à chaque « feature map »:





1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

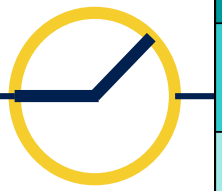
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33

0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

Couche RELU

- Les valeurs négatives sont passées à 0

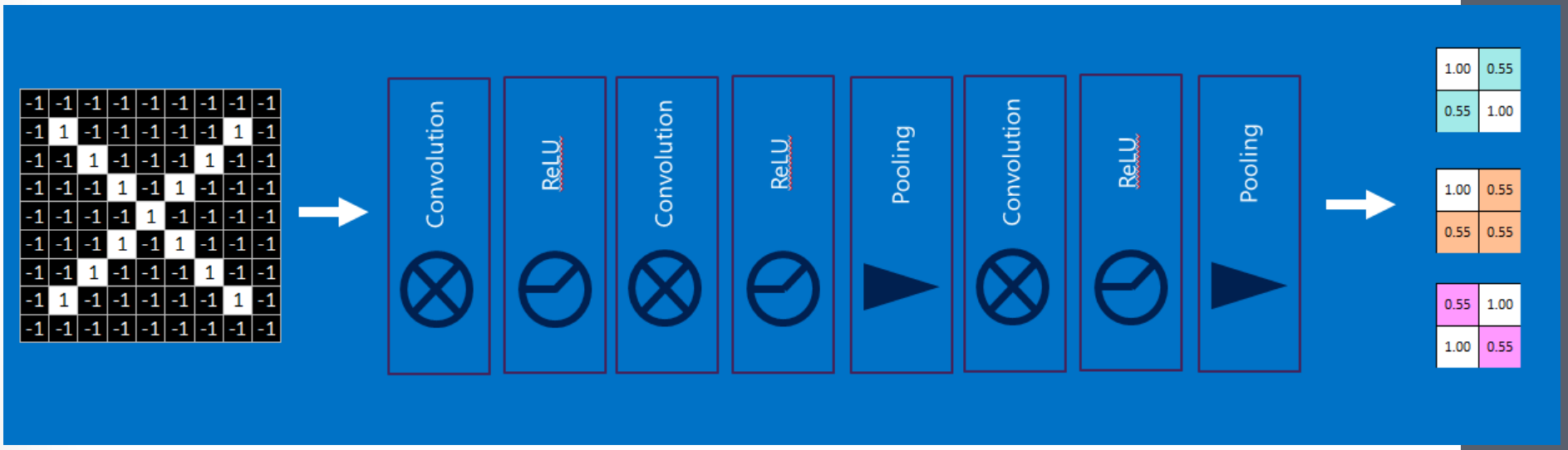
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77



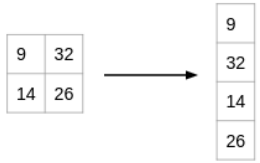
0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	0.77

Combinaison des couches

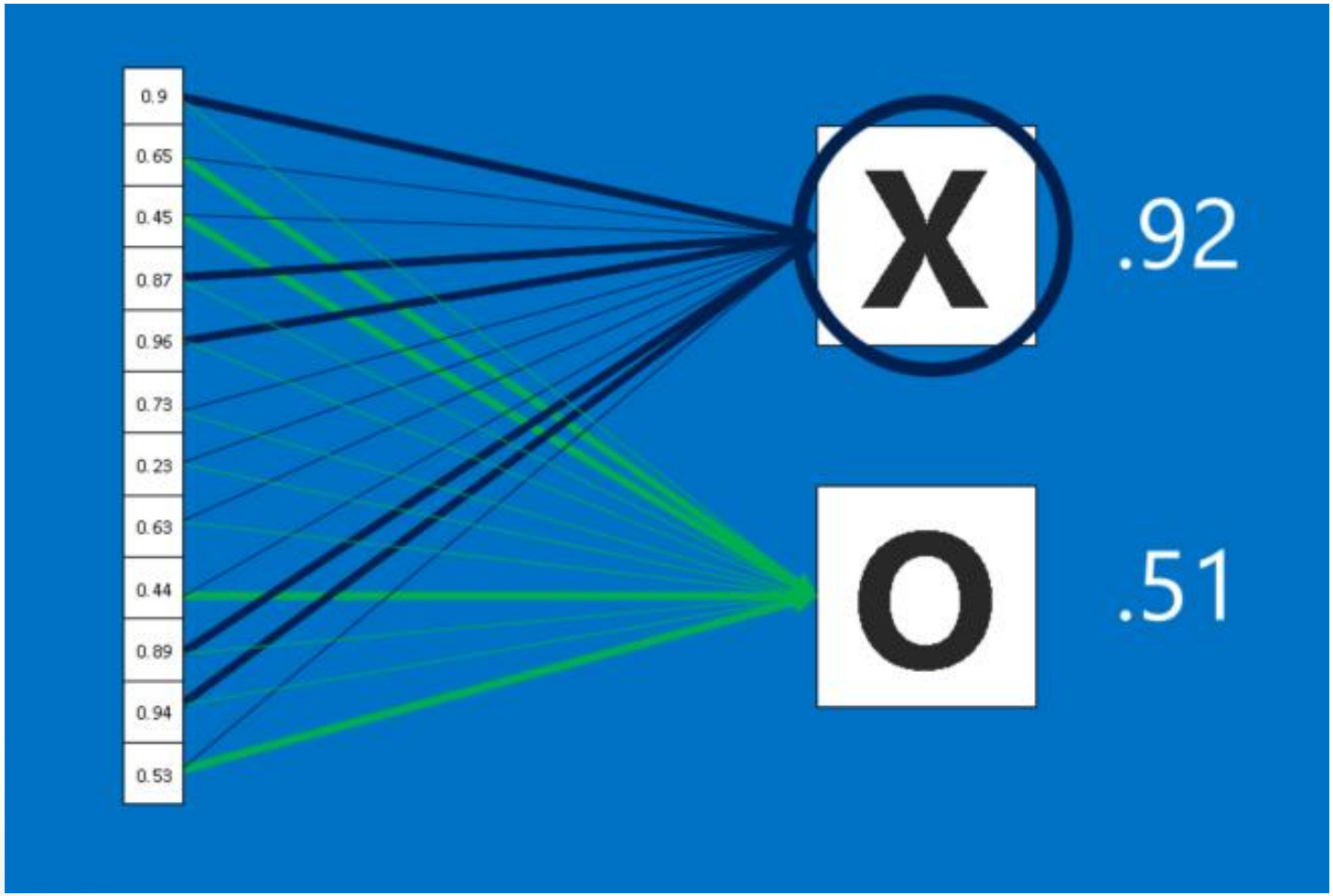
- On peut combiner à l'infini les 3 couches.



Dépliement à la fin :

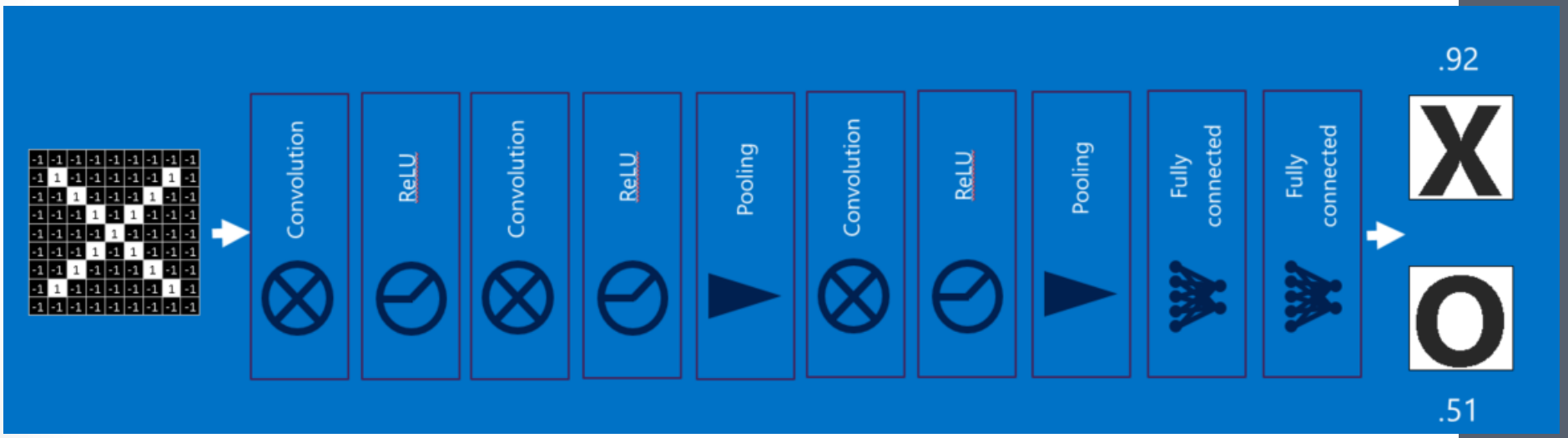


Dépliement et couche dense



Résultat Final

- Les couches de convolutions sont complétées par des couches ANN « **fully connected** »:



https://www.youtube.com/watch?v=f0t-OCG79-U

```python
import torch
import torch.nn as nn

class BasicCNNModel(nn.Module):
    def __init__(self, inp_dim=(10, 10), outp_dim=(10, 10)):
        super(BasicCNNModel, self).__init__()

        CONV_IN = 3
        KERNEL_SIZE = 3
        CONV_OUT_1 = 50
        CONV_OUT_2 = 100
        DENSE_IN = CONV_OUT_2

        self.relu = nn.ReLU()
        self.conv2d_1 = nn.Conv2d(CONV_IN, CONV_OUT_1, kernel_size=KERNEL_SIZE)
        self.conv2d_2 = nn.Conv2d(CONV_OUT_1, CONV_OUT_2, kernel_size=KERNEL_SIZE)
        self.dense_1 = nn.Linear(DENSE_IN, outp_dim[0]*outp_dim[1]*10)
        self.softmax = nn.Softmax(dim=1)


    def forward(self, x, outp_dim):
        # ... preparation de x

        # Convolution 1
        self.conv2d_1.in_features = x.shape[1]
        conv_1_out = self.relu(self.conv2d_1(x))

        # Convolution 2
        self.conv2d_2.in_features = conv_1_out.shape[1]
        conv_2_out = self.relu(self.conv2d_2(conv_1_out))

        # max Pooling
        self.dense_1.out_features = outp_dim
        feature_vector, _ = torch.max(conv_2_out, 2)
        feature_vector, _ = torch.max(feature_vector, 2)

        # Couche finale et softmax
        logit_outputs = self.dense_1(feature_vector)
        return self.softmax(logit_outputs)
```