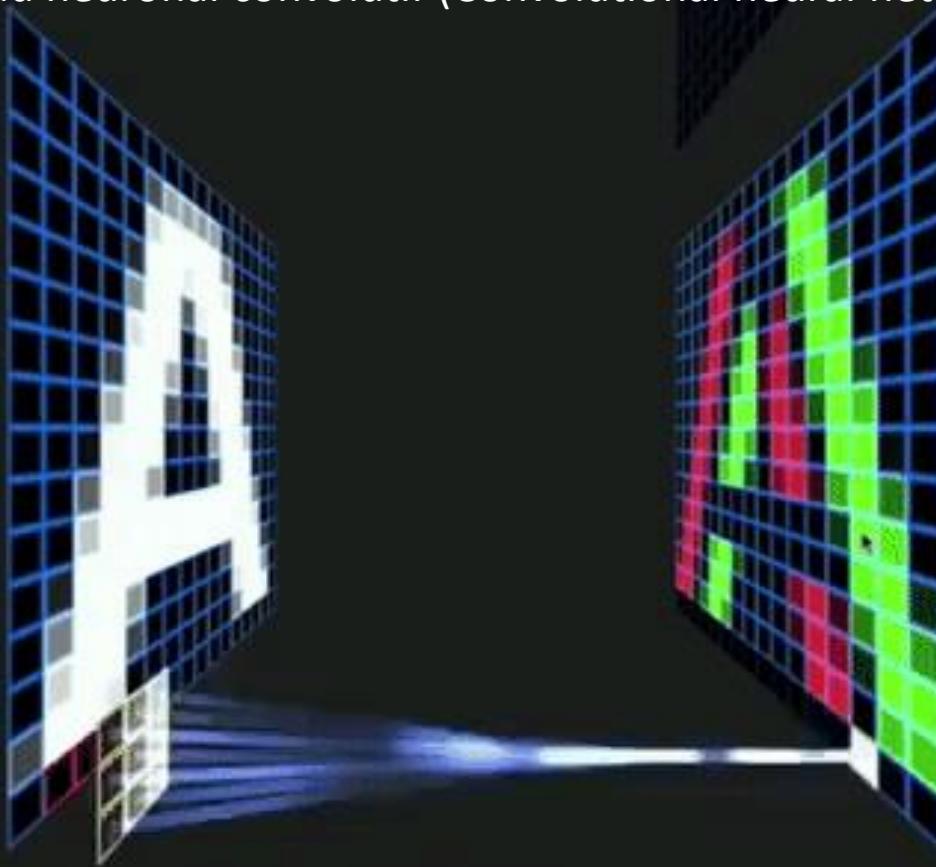
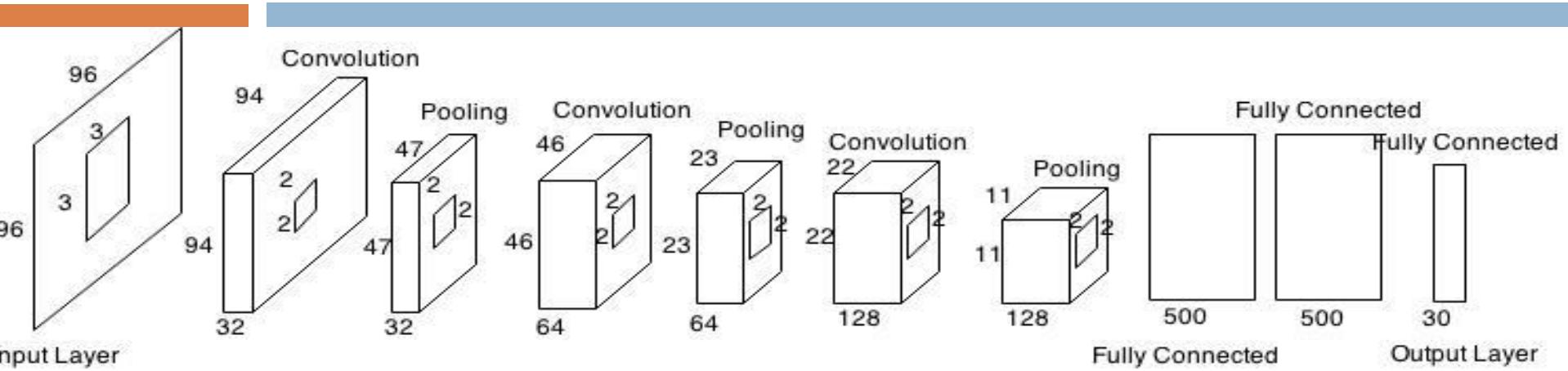


Pr. BEN LAHMAR EL Habib

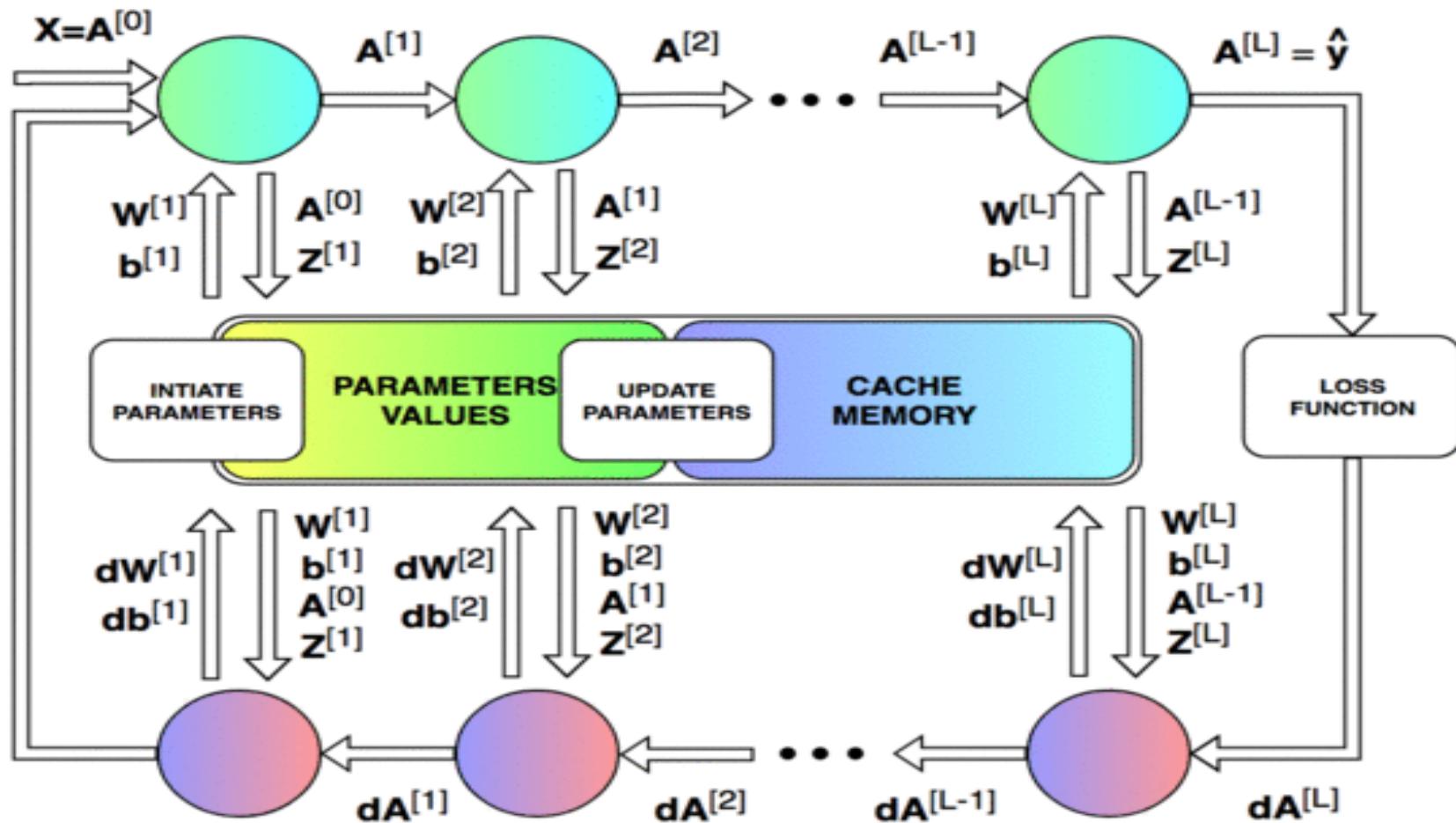
Réseau neuronal convolutif (Convolutional neural network)



gifs.com



FORWARD PROPAGATION



BACKWARD PROPAGATION

Convolutional

- Le nom "réseau neuronal convolutif" indique que le réseau utilise une opération mathématique appelée convolution
- La convolution est un type spécialisé d'opération linéaire.
- Les réseaux convolutionnels sont simplement des réseaux de neurones qui utilisent la convolution au lieu de la multiplication matricielle générale dans au moins une de leurs couches.

Digital Photo Data Structure

9

0	0	0	0	0	0	0	0	0
0	0	0	255	255	0	0	0	0
0	0	255	0	0	255	0	0	0
0	0	255	0	0	255	0	0	0
0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0
0	0	255	255	255	0	0	0	0
0	0	0	0	0	0	0	0	0

Blue

Green

Red

<https://towardsdatascience.com/gentle-dive-into-math-behind-convolutional-neural-networks-79a07dd44cf9>

$$s(t) = \int x(a)w(t-a)da. \quad (9.1)$$

This operation is called **convolution**. The convolution operation is typically denoted with an asterisk:

$$s(t) = (x * w)(t). \quad (9.2)$$

$$(f * g)(t) \triangleq \int_{-\infty}^{+\infty} f(\tau)g(t - \tau) d\tau.$$

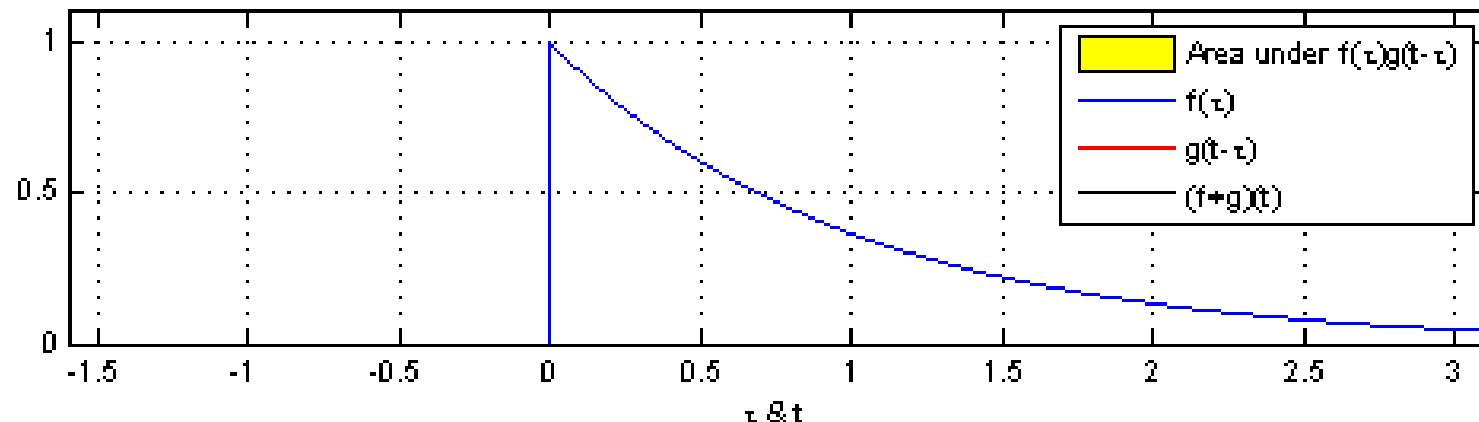
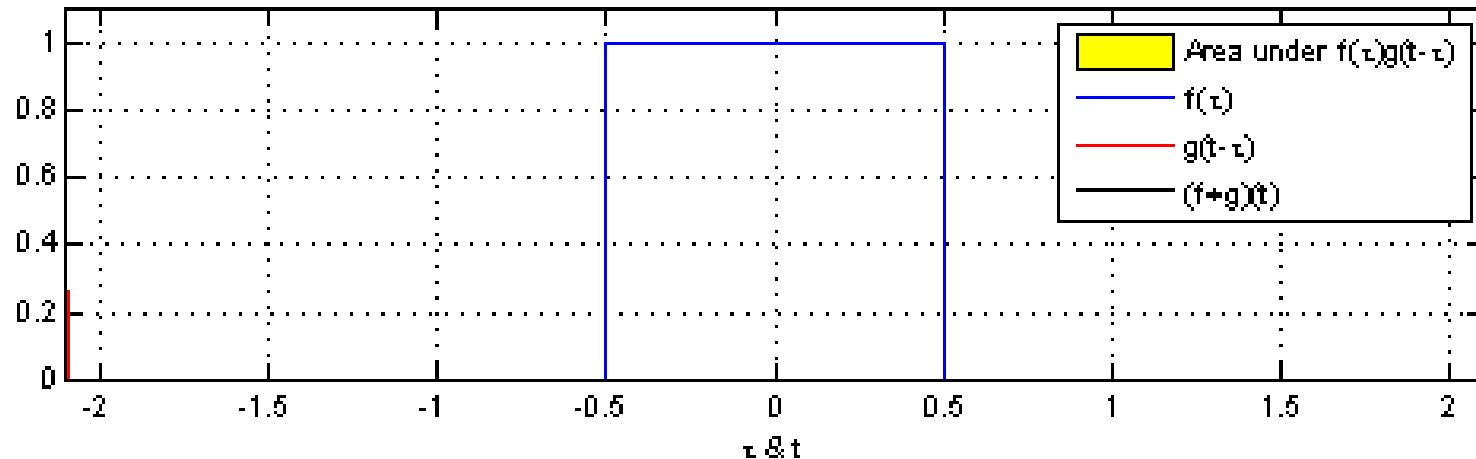
Dans la terminologie des réseaux convolutionnels, l'argument first (dans cet exemple, la fonction x) de la convolution est souvent appelé l'entrée, et le second argument (dans cet exemple, la fonction w) le noyau. La sortie est parfois appelée carte de caractéristiques (feature map)

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a).$$

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

Dans les applications d'apprentissage machine, l'entrée est généralement un tableau multidimensionnel de données, et le noyau est généralement un tableau multidimensionnel de paramètres qui sont adaptés par l'algorithme d'apprentissage.

Exemples



Exemple

Par exemple, si nous utilisons une image bidimensionnelle I comme entrée, nous voulons probablement aussi utiliser un noyau bidimensionnel K :

$$S(i, j) = (I * K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n).$$

Convolution is commutative, meaning we can equivalently write

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i - m, j - n)K(m, n).$$

5	8	5	9
4	5	0	-9
5	88	0	1
1	2	5	7

1	0
0	1

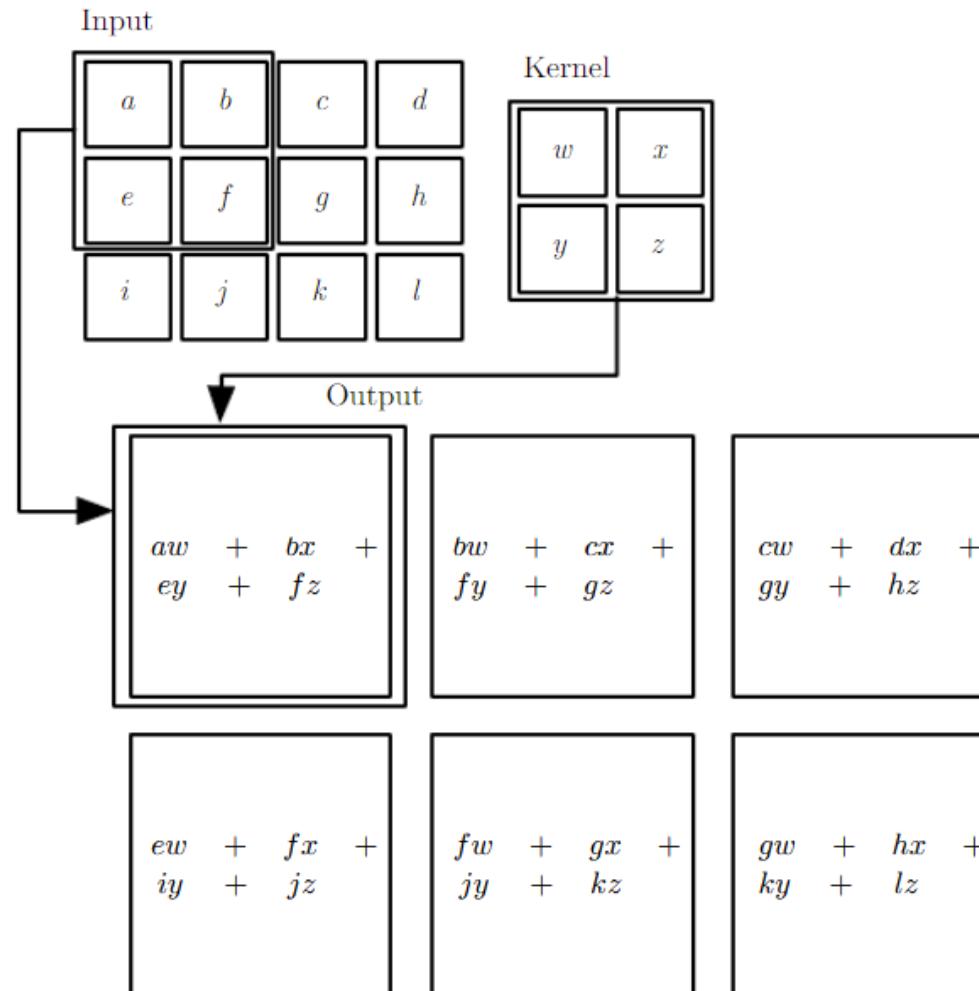
$5*1+8*0+$ $4*0+1*5$		

kernel

- Au lieu de cela, de nombreuses bibliothèques de réseaux de neurones mettent en œuvre une fonction connexe appelée **corrélation croisée**, qui est la même que la convolution mais sans flipping le noyau

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n).$$

An example of 2-D convolution without kernel flipping



Exemple

Kernel Convolution Example

Input Image

10	10	10	10	10	10
10	10	10	10	10	10
10	10	10	10	10	10
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

*

Kernel

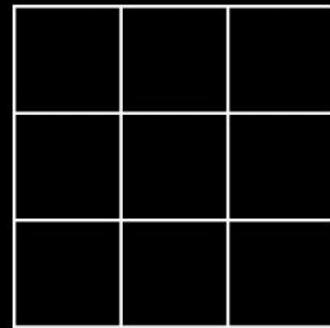
1	2	1
0	0	0
-1	-2	-1

=

Feature Map

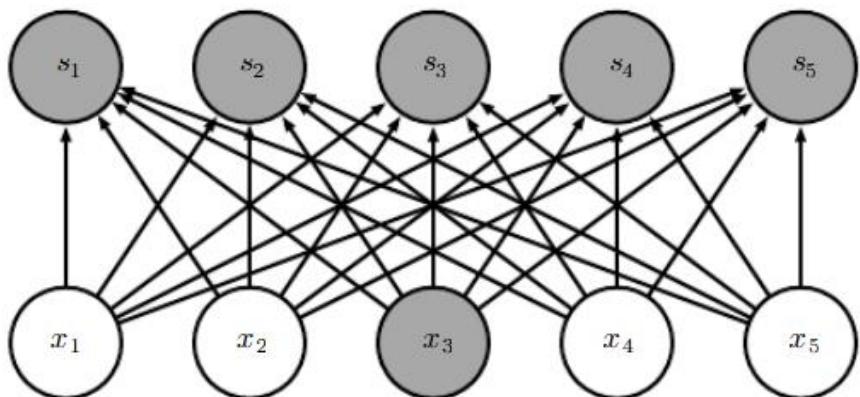
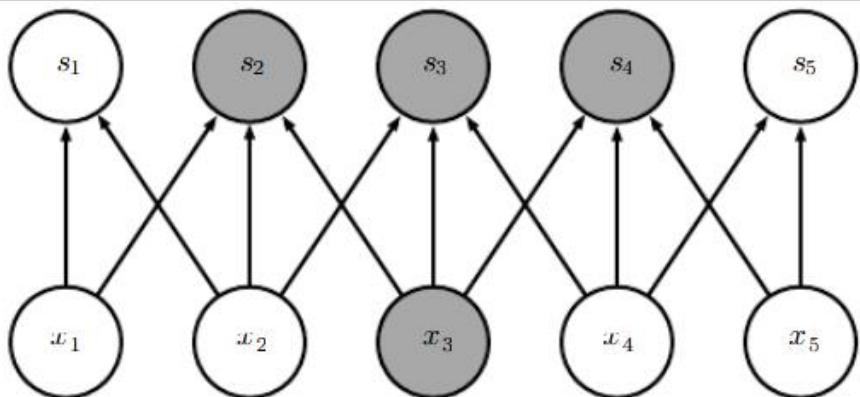
Edge Detection

Using Kernel Convolution



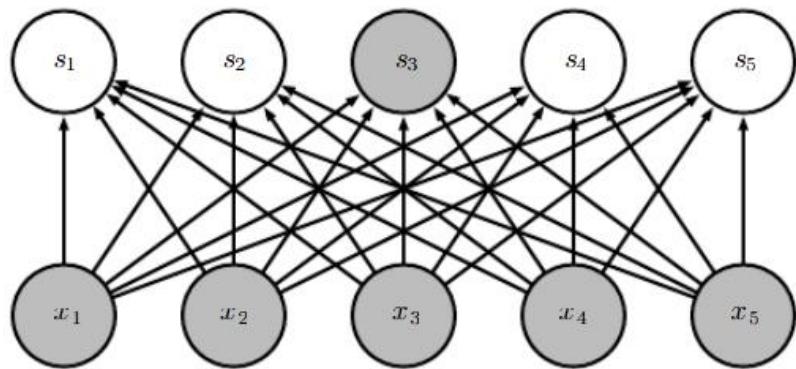
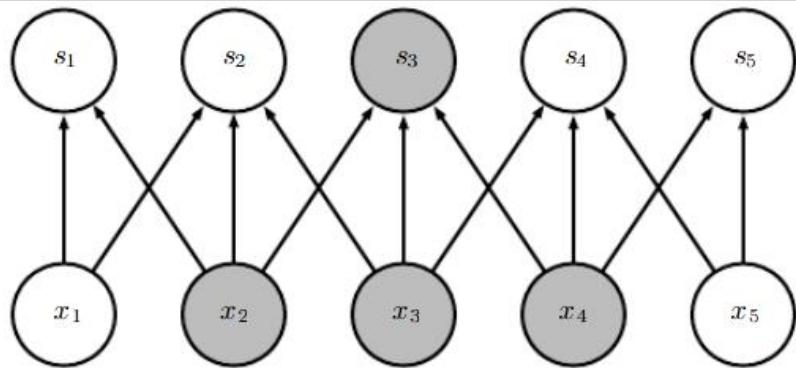
sparse connectivity

strided convolution



- Connectivité réduite, vue d'en bas. Nous mettons en évidence une unité d'entrée, x_3 , et nous mettons en évidence les unités de sortie en s qui sont affected par cette unité.
- (Haut) Lorsque s est formé par convolution avec un noyau de largeur 3, seules trois sorties sont affected par x_3 .
- (En bas) Lorsque s est formé par multiplication matricielle, la connectivité n'est plus rare, de sorte que toutes les sorties sont affected par x_3

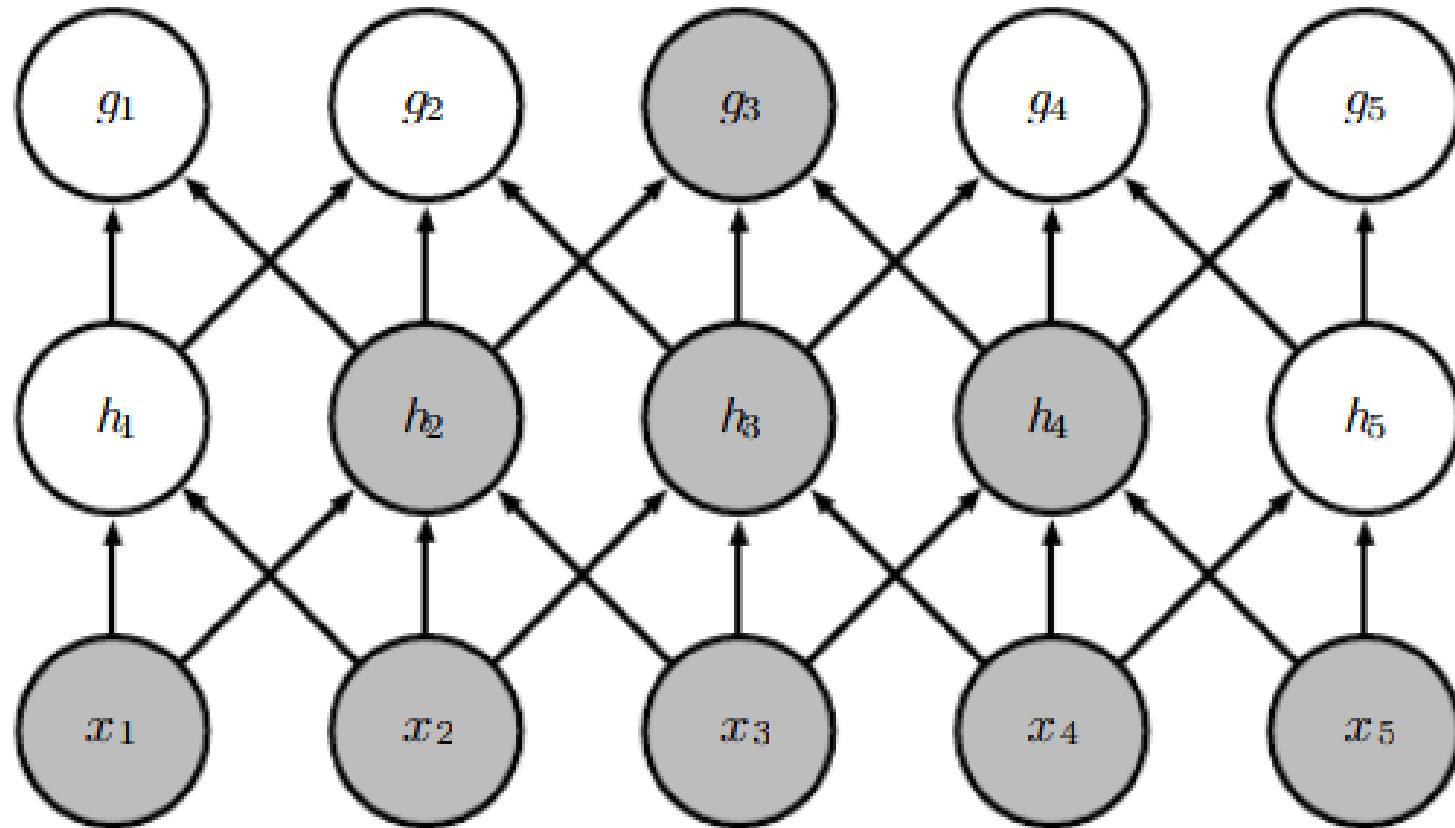
Sparse connectivity



- Connectivité réduite, vue de dessus. Nous mettons en évidence une unité de sortie, s_3 , et nous mettons en évidence les unités d'entrée en x qui affect cette unité. Ces unités sont connues sous le nom de field réceptif de s_3 .
- (Haut) Lorsque s est formé par convolution avec un noyau de largeur 3, seulement trois entrées affect s_3 .
- (En bas) Lorsque s est formé par multiplication matricielle, la connectivité n'est plus rare, de sorte que toutes les entrées affect s_3 .

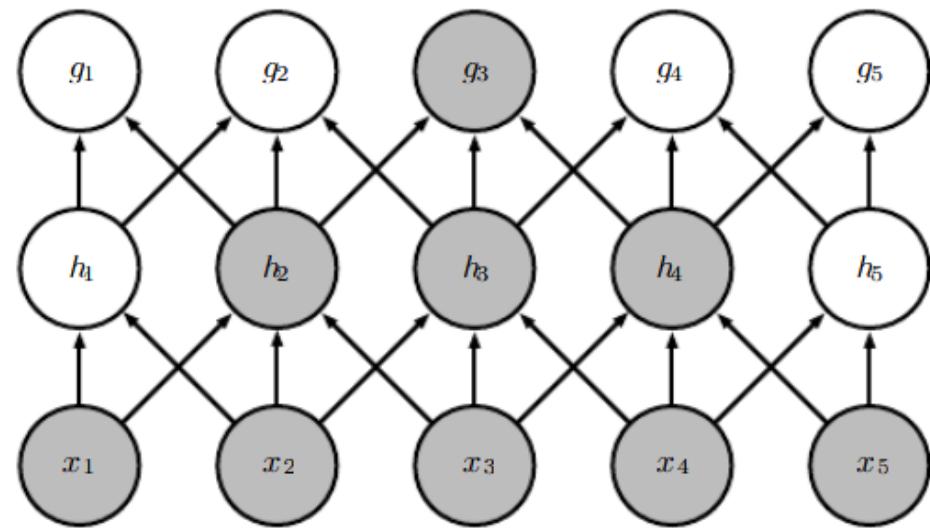
Connections

- Dans un réseau convolutif profond, les unités des couches les plus profondes peuvent interagir indirectement avec une plus grande partie de l'entrée



Connections

- Le field réceptif des unités situées dans les couches profondes d'un réseau convolutif est plus grand que le field réceptif des unités situées dans les couches peu profondes.
- Cela signifie que, même si les connexions directes dans un réseau convolutif sont très rares, les unités des couches profondes peuvent être indirectement connectées à la totalité ou à la majeure partie de l'image d'entrée





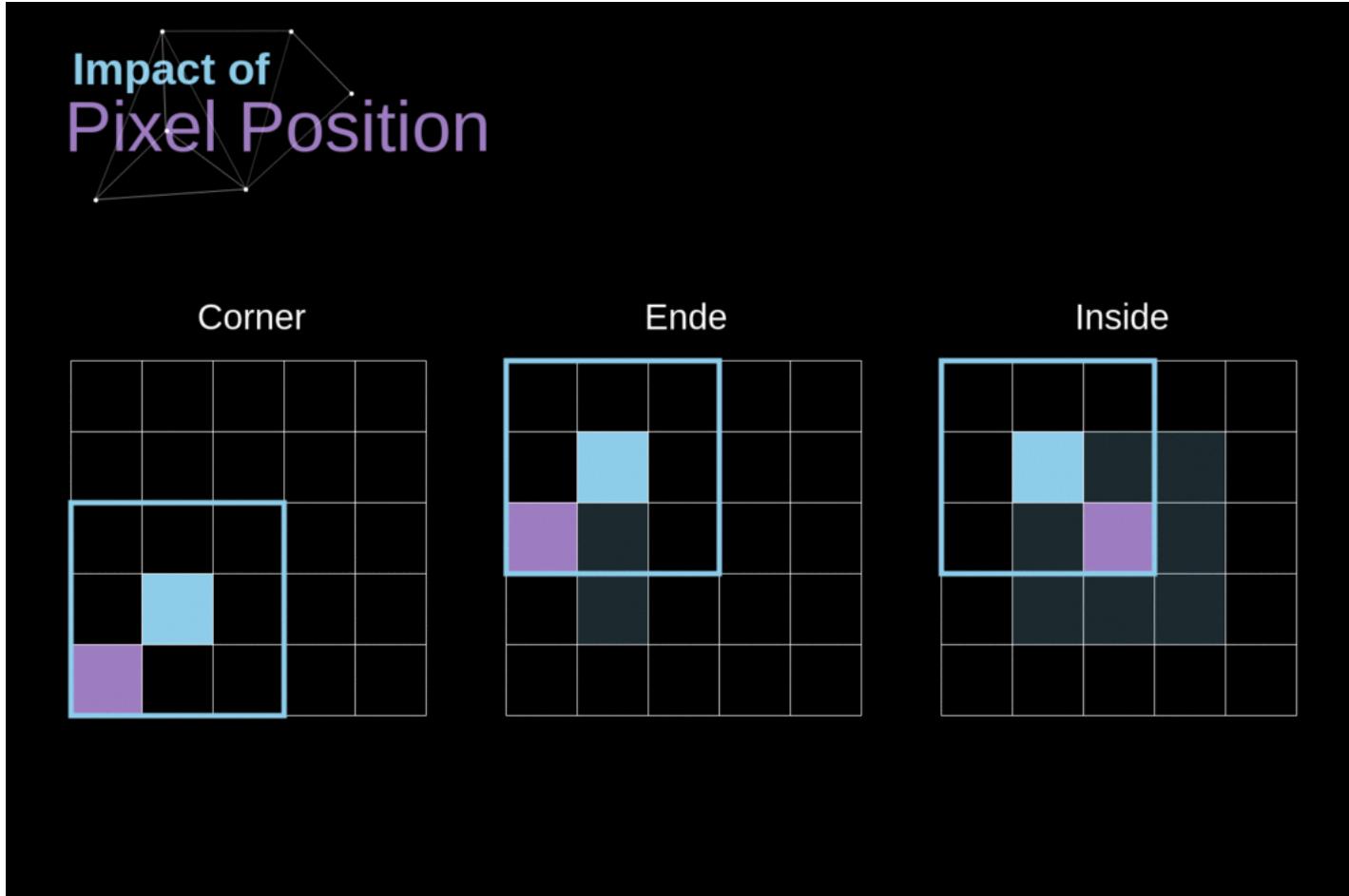
how to handle the convolution in the other two directions
(height & width)

Comment gérer la convolution dans les deux autres directions (hauteur et largeur)

Terminologies

- **Kernel size:** (Taille du noyau) le noyau est abordé dans la section précédente. La taille du noyau définit le champ de vision de la convolution.
- **Stride :** il définit la taille du pas du noyau lorsqu'il glisse dans l'image. Stride de 1 signifie que le noyau glisse à travers l'image pixel par pixel. Le pas de 2 signifie que le noyau glisse à travers l'image en déplaçant 2 pixels par pas (c'est-à-dire en sautant 1 pixel). Nous pouvons utiliser la foulée (≥ 2) pour le sous-échantillonnage d'une image.
- **Padding:** the padding définit le traitement de la bordure d'une image. Une convolution complétée(padded) ((‘same’ padding in Tensorflow dans Tensorflow) maintiendra les dimensions de sortie spatiale égales à l'image d'entrée, en complétant le cas échéant les 0 autour des limites d'entrée. Par ailleurs, la convolution unpadded(remplissage ‘valide’ dans Tensorflow) n'effectue la convolution que sur les pixels de l'image en entrée, sans ajouter de 0 autour des limites de l'entrée. La taille de sortie est inférieure à la taille d'entrée.

Exemple



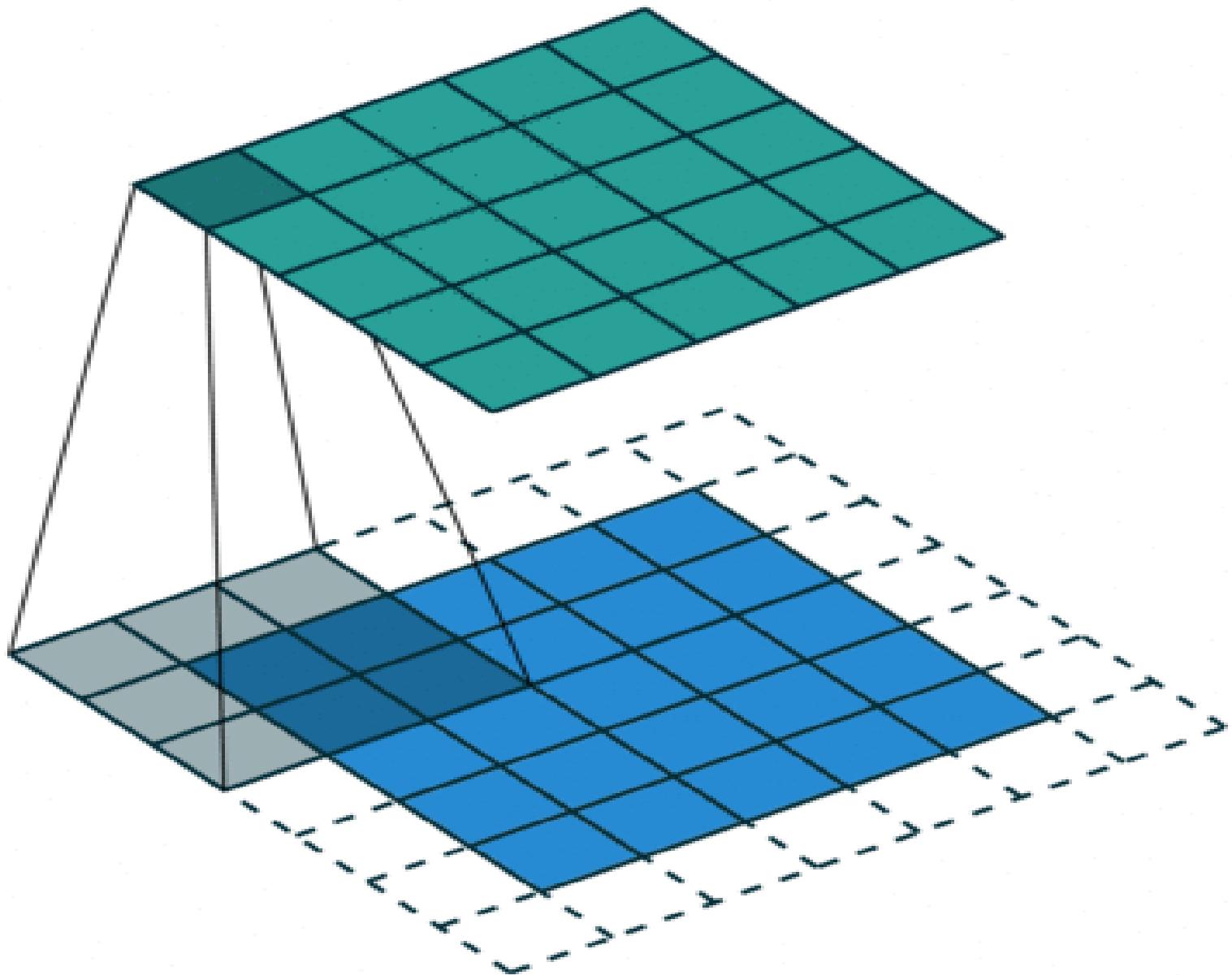


Illustration describes a 2D convolution using a kernel size of 3, stride of 1 and padding of 1.

Padding (Valid and Same)

- “**same**” padding, l'entrée et la sortie ont les mêmes dimensions.
- Le fait de ne pas utiliser un padding, est parfois appelé “**valid**” padding.

ref

- <https://arxiv.org/pdf/1603.07285.pdf>

step

- Lors de la conception de notre architecture CNN, nous pouvons décider d'augmenter le pas si nous voulons que les champs réceptifs se chevauchent moins ou si nous voulons des dimensions spatiales plus petites de notre carte de caractéristiques (feature map). Les dimensions de la matrice de sortie - en tenant compte du padding et du pas - peuvent être calculées à l'aide de la formule suivante.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - f}{s} + 1 \right\rfloor$$

where p is padding and f is the filter dimension (usually **odd**) and s **stride**

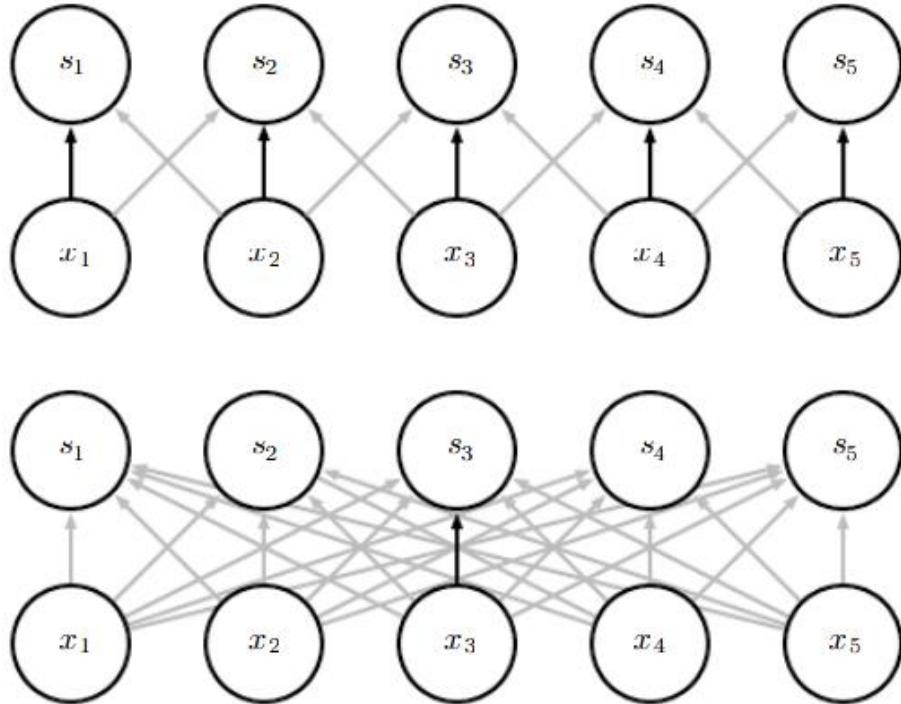
Parameter sharing

- désigne l'utilisation du même paramètre pour plus d'une fonction dans un modèle. Dans un réseau neuronal traditionnel, chaque élément de la matrice de poids est utilisé exactement une fois lors du calcul de la sortie d'une couche. Il est multiplié par un élément de l'entrée, puis n'est jamais revu.
- En tant que synonyme de partage des paramètres, on peut dire qu'un réseau a des poids liés, car la valeur du poids appliqué à une entrée est liée à la valeur d'un poids appliqué ailleurs.

Parameter sharing

- Dans un réseau neuronal convolutif, chaque membre du noyau est utilisé à chaque position de l'entrée (à l'exception peut-être de certains des pixels de limite, selon les décisions de conception concernant la limite). Le partage des paramètres utilisé par l'opération de convolution signifie qu'au lieu d'apprendre un ensemble distinct de paramètres pour chaque emplacement, nous n'en apprenons qu'un seul.

Parameter sharing.



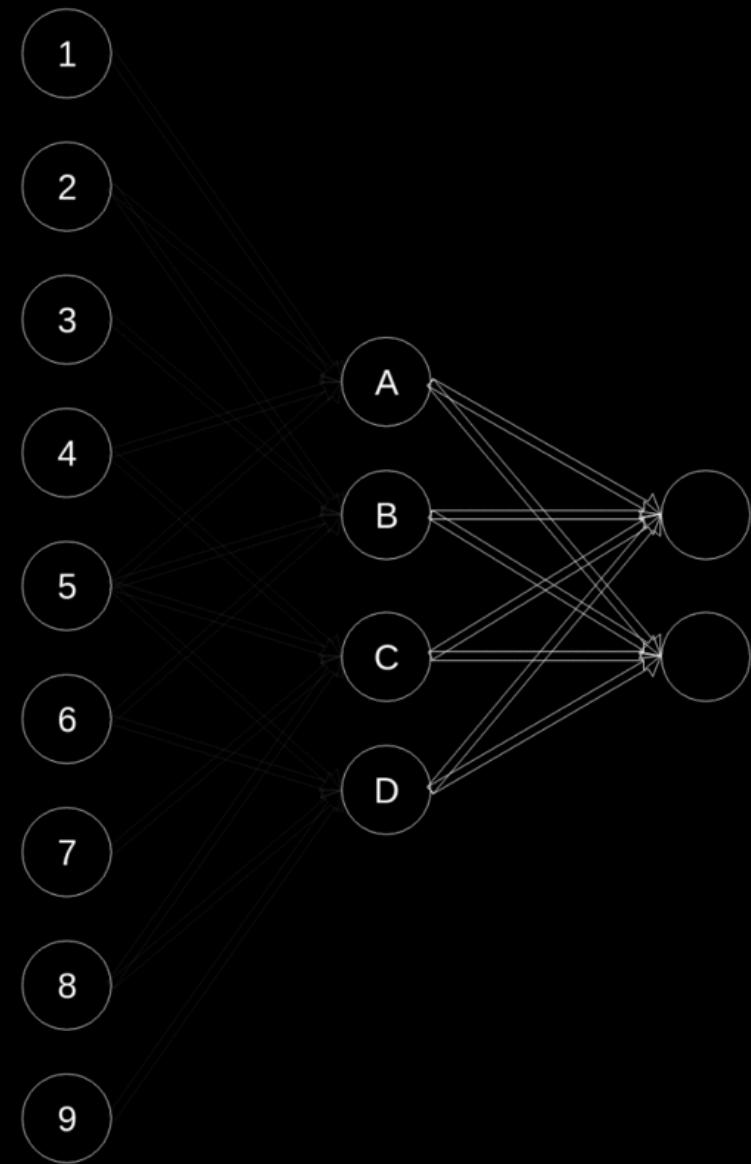
- Les flèches noires indiquent les connexions qui utilisent un paramètre particulier dans deux modèles de différents.
- (Haut) Les flèches noires indiquent les utilisations de l'élément central d'un noyau à 3 éléments dans un modèle convolutif. En raison du partage des paramètres, ce paramètre unique est utilisé dans tous les lieux d'entrée.

(En bas) L'unique flèche noire indique l'utilisation de l'élément central de la matrice de poids dans un modèle entièrement connecté. Ce modèle n'a pas de partage de paramètres, de sorte que le paramètre n'est utilisé qu'une seule fois.

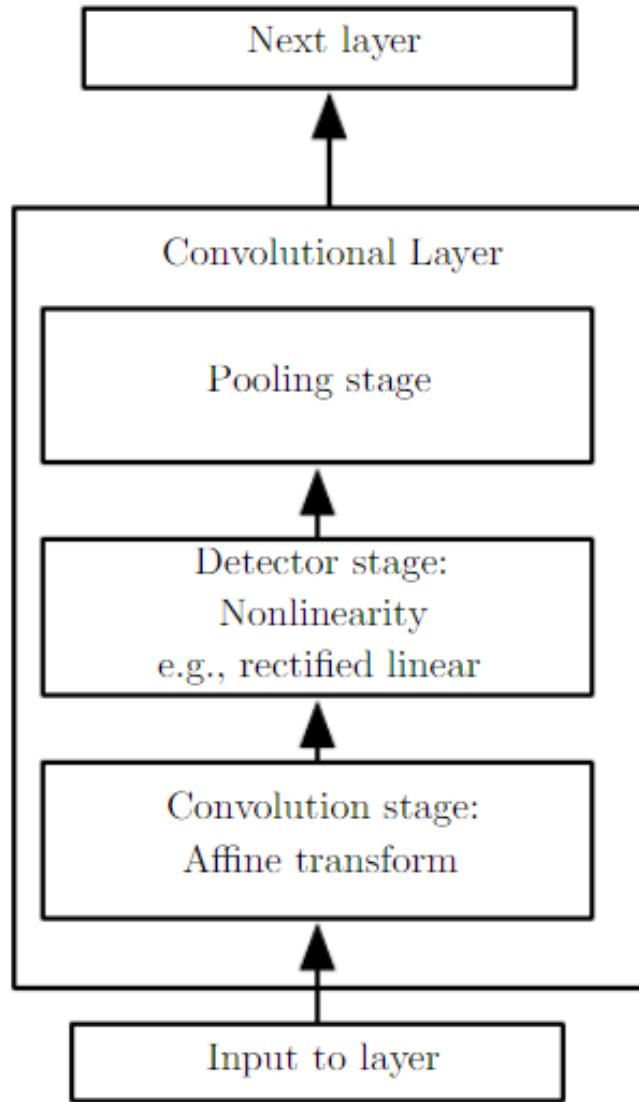
Connection Cutting Parameters Shering

Input Image Kernel Feature Map

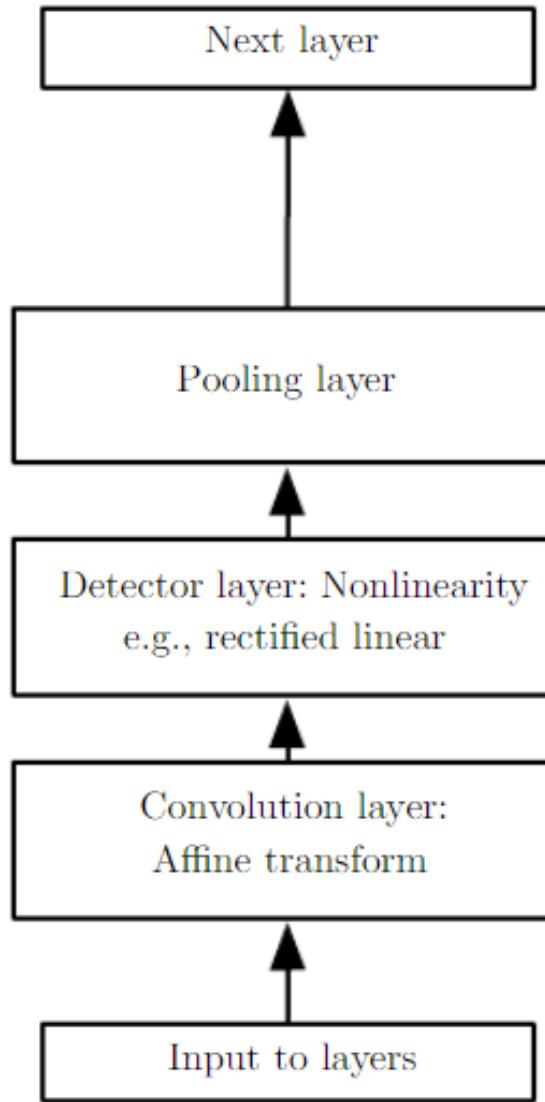
$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix} * \begin{matrix} I & II \\ III & IV \end{matrix} = \begin{matrix} A & B \\ C & D \end{matrix}$$



Complex layer terminology



Simple layer terminology



Pooling function

- Une fonction de mise en commun (Pooling) remplace la sortie du réseau à un certain endroit par une statistique sommaire des sorties à proximité. Par exemple, l'opération de mise en commun max (Zhouand Chellappa, 1988) indique la sortie maximale dans un voisinage rectangulaire.
- D'autres fonctions de mise en commun populaires comprennent la moyenne d'un voisinage rectangulaire, la norme L2 d'un voisinage rectangulaire, ou une moyenne pondérée basée sur la distance du pixel central.

Pooling Layers

- Ils sont utilisés principalement pour réduire la taille du tenseur et accélérer les calculs. Ces couches sont simples: nous devons diviser notre image en différentes régions, puis effectuer une opération pour chacune de ces parties. Par exemple, pour la couche Max Pool, nous sélectionnons une valeur maximale de chaque région et la plaçons à l'endroit correspondant dans la sortie.
- La mise en commun (**Pooling**) divise la largeur et la hauteur de l'entrée par la taille du pool.

Max Pooling Example

Input

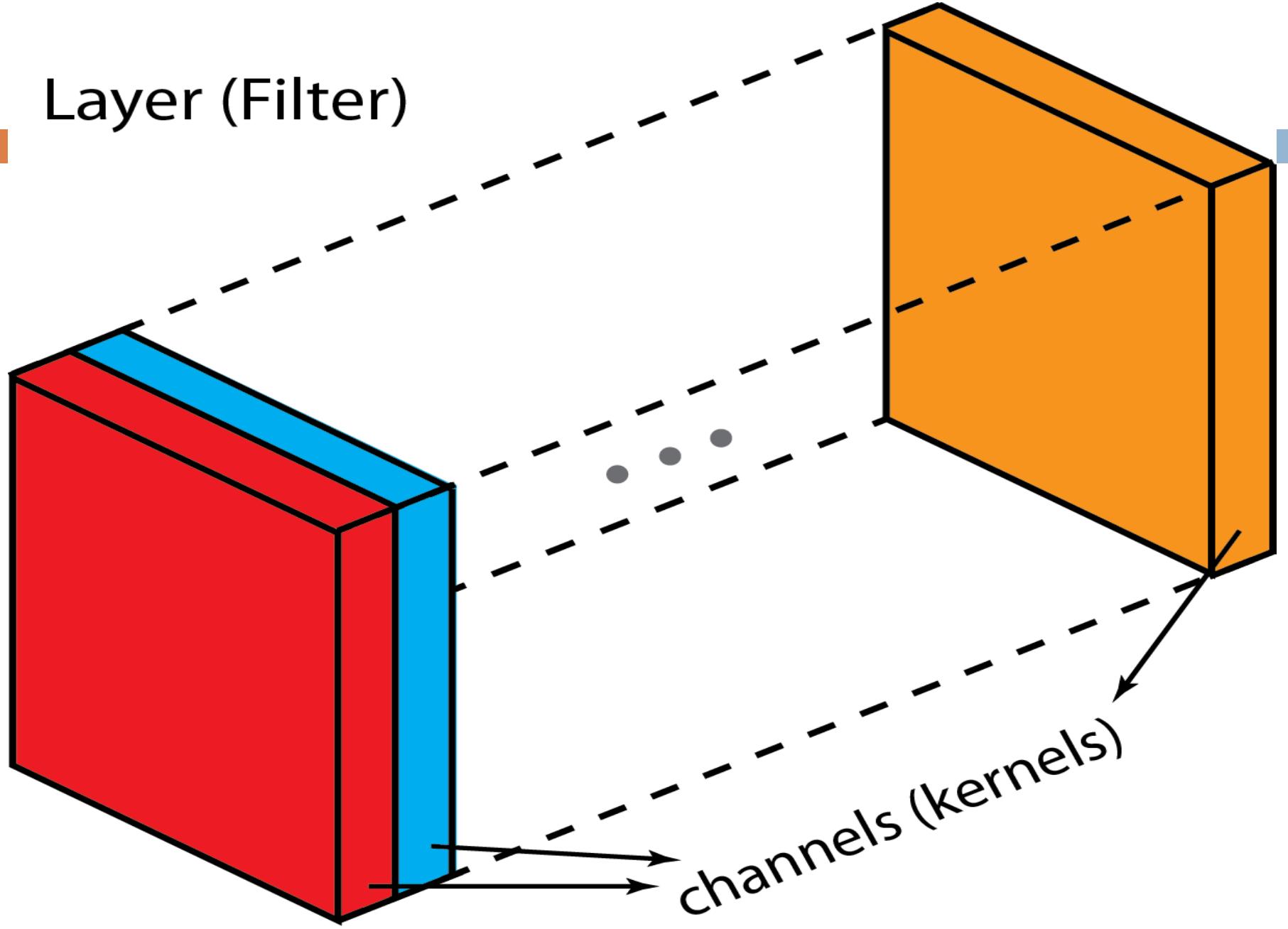
3	0	1	5	1	3
5	7	3	4	4	6
7	7	1	8	3	5
6	1	7	0	0	5
0	4	5	5	7	2
3	2	0	2	0	2

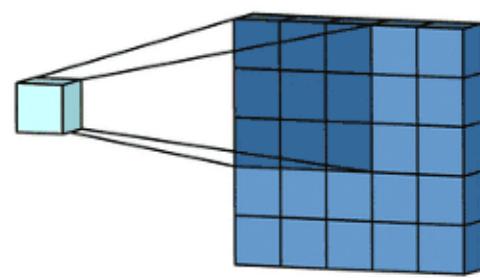
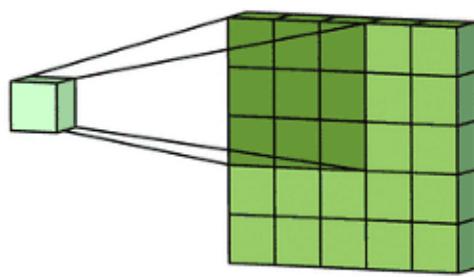
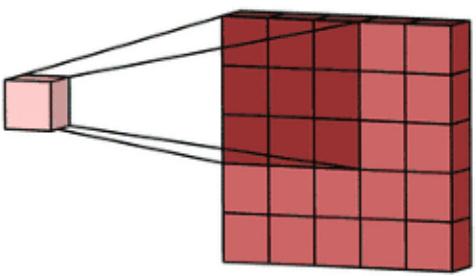
Output

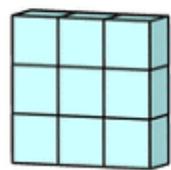
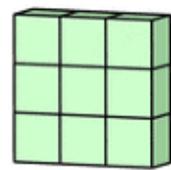
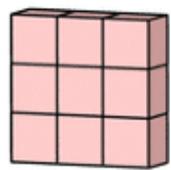
The transition to the third dimension

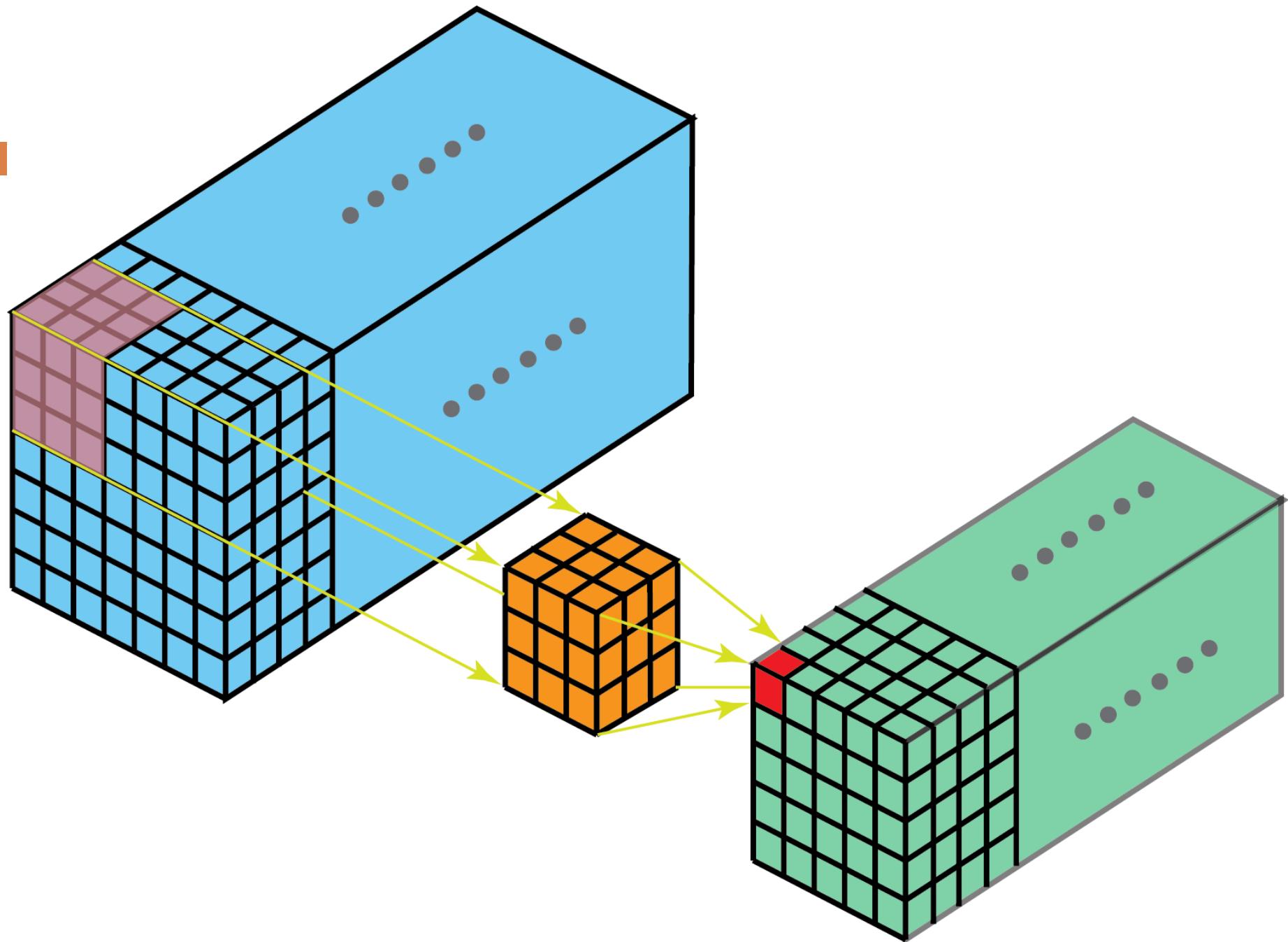
- La différence entre le filtre et le noyau est un peu délicate.
- Parfois, ils sont utilisés de manière interchangeable, ce qui pourrait créer des confusions. En fait, ces deux termes présentent une différence subtile.
 - ▣ Un "noyau" désigne un ensemble de poids en 2D.
 - ▣ Le terme "filtre" désigne les structures 3D de plusieurs noyaux empilés ensemble.

Layer (Filter)

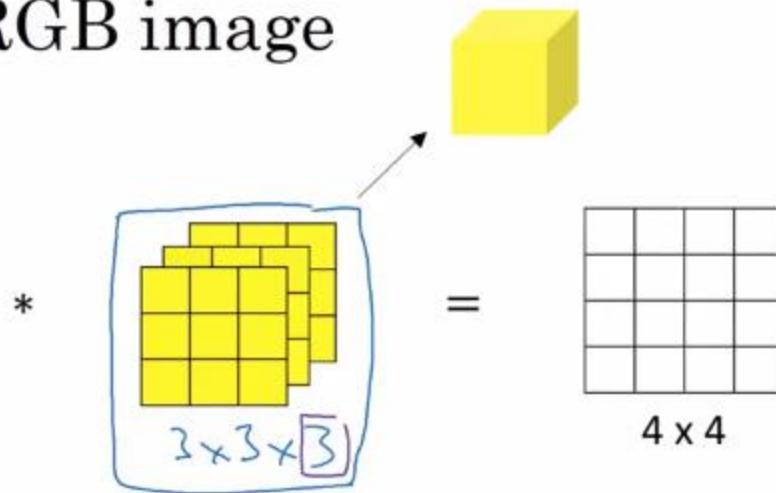
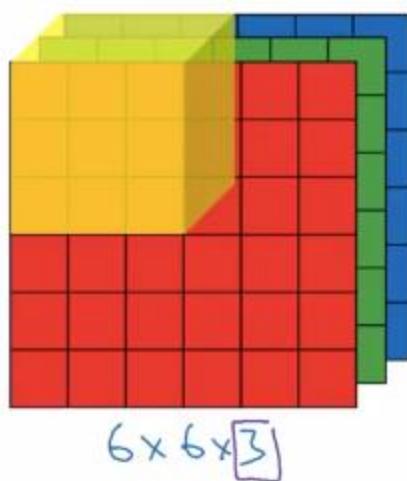








Convolutions on RGB image



Softmax

La **fonction softmax**, ou **fonction exponentielle normalisée**, est une généralisation de la fonction logistique qui prend en entrée un vecteur X de n nombres réels et qui en sort un vecteur $S(X)$ de K nombres réels strictement positifs et de somme 1.

La fonction est définie par :

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

Softmax est mis en œuvre via une couche de réseau de neurones juste avant la couche du résultat. La couche Softmax doit comporter le même nombre de nœuds que la couche du résultat.

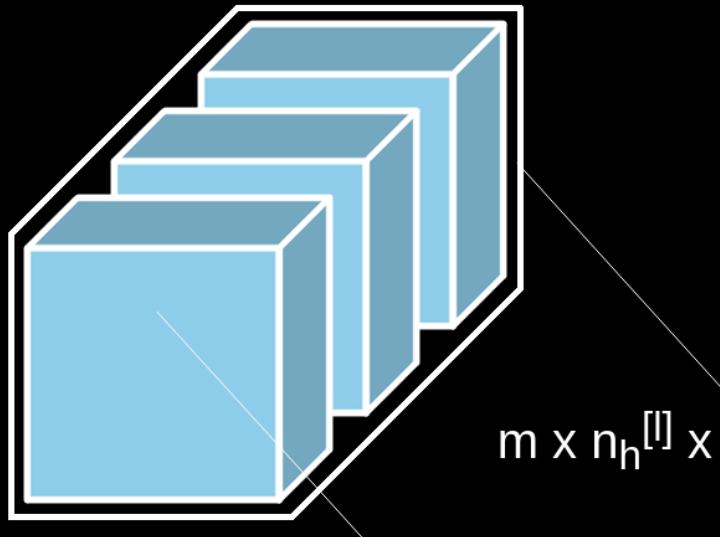
Softmax

- Le dénominateur $\sum_{j=1}^n e^{x_j}$ agit comme un régularisateur pour s'assurer que $\sum_{c=1}^C y_c = 1$
- En tant que couche de sortie d'un réseau neuronal, la fonction softmax peut être représentée graphiquement comme une couche avec C neurones.

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

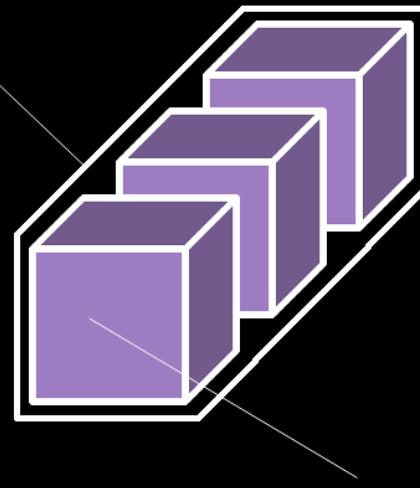
Tensors Dimensions

Activations $\mathbf{A}[l]$



Weights $\mathbf{W}[l]$

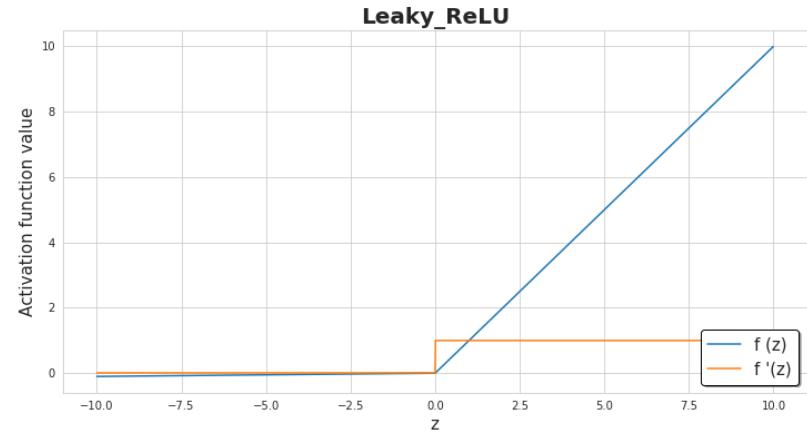
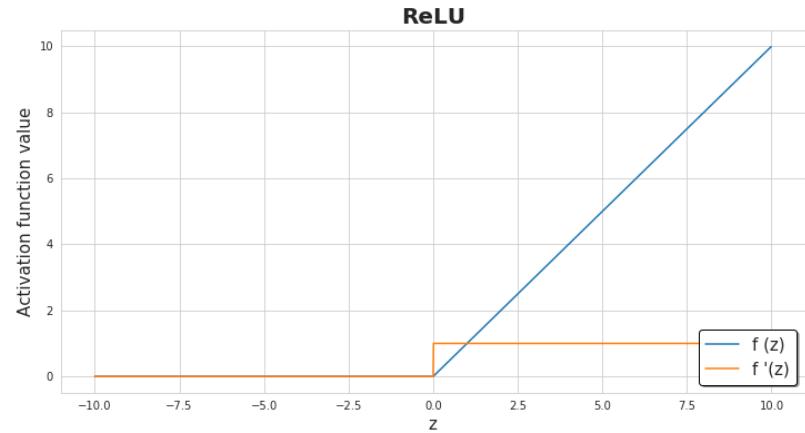
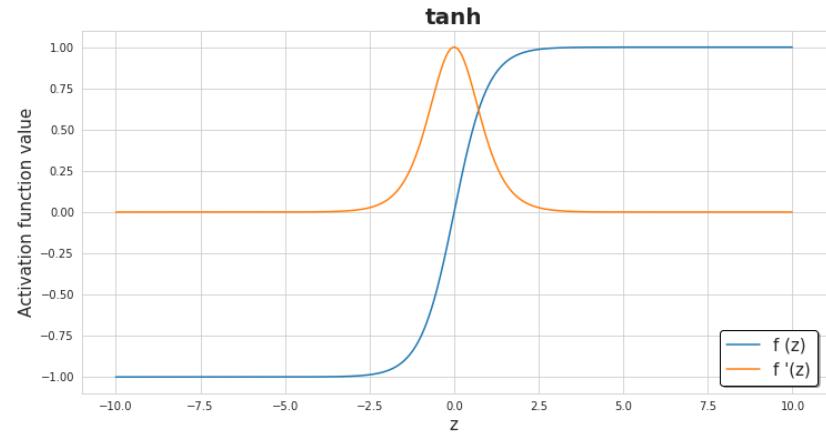
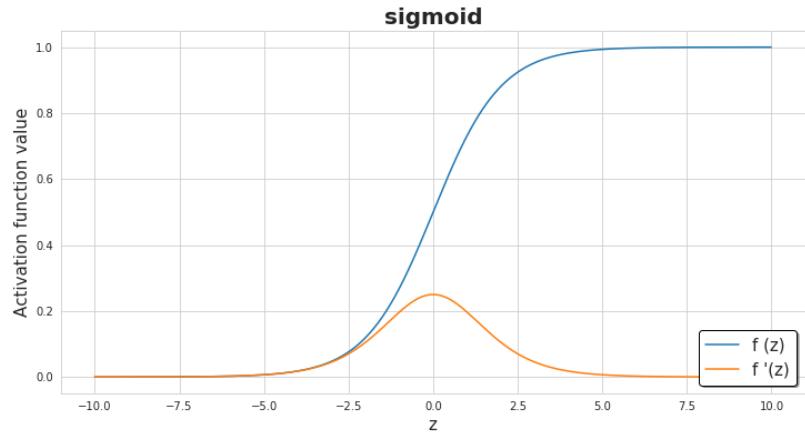
$f[l] \times f[l] \times n_c^{[l-1]} \times n_c^{[l]}$



Single example
 $n_h[l] \times n_w[l] \times n_c[l]$

Single filter
 $f[l] \times f[l] \times n_c^{[l-1]}$

Activation functions



Back Propagation

Training

- La formation d'un réseau de neurones comprend généralement deux phases:
 - Une phase d'avancement, où l'entrée passe entièrement à travers le réseau.
 - Une phase arrière, où les gradients sont rétropropagés (backprop) et les poids sont mis à jour.
- Nous suivrons ce schéma pour former notre CNN. Nous utiliserons également deux grandes idées spécifiques à la mise en œuvre :
- **During the forward** phase, each layer will **cache** any data (like inputs, intermediate values, etc) it'll need for the backward phase. This means that any backward phase must be preceded by a corresponding forward phase.
- **During the backward** phase, each layer will **receive a gradient** and also **return a gradient**. It will receive the gradient of loss with respect to its **outputs** $\partial L / \partial \text{out}$ and return the gradient of loss with respect to its **inputs** $\partial L / \partial \text{in}$.

Forward propagation

- La propagation vers l'avant se fait en deux étapes. La première consiste à calculer la valeur intermédiaire Z , qui est obtenue par la convolution des données d'entrée de la couche précédente avec le tenseur W (contenant des filtres), puis à ajouter le biais b .
- La seconde est l'application d'une fonction d'activation non linéaire à notre valeur intermédiaire (notre activation est désignée par g).

$$\mathbf{Z}^{[l]} = \mathbf{W}^{[l]} \cdot \mathbf{A}^{[l-1]} + \mathbf{b}^{[l]} \quad \mathbf{A}^{[l]} = g^{[l]}(\mathbf{Z}^{[l]})$$

Perte d'entropie croisée

- L'entropie croisée indique la distance entre ce que le modèle estime que la distribution de sortie devrait être et ce qu'est réellement la distribution d'origine.
- $$H(y,p) = -\sum_i y_i \log(p_i)$$
- La mesure d'entropie croisée est une alternative largement utilisée de l'erreur quadratique.
- Il est utilisé lorsque les activations de nœuds peuvent être comprises comme représentant la probabilité que chaque hypothèse soit vraie, c'est-à-dire lorsque la sortie est une distribution de probabilité. Ainsi, il est utilisé comme fonction de perte dans les réseaux de neurones qui ont des activations softmax dans la couche de sortie.

Cost function

- **cross-entropy loss** $H(y, p) = -\sum_i y_i \log(p_i)$
- où p_i est la probabilité prédite pour la classe i correcte

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

$$H(p, q) = - \sum_x p(x) \log q(x)$$

Les deux formules sont fondamentalement équivalentes l'une à l'autre

Backprop: Softmax

- la première chose que nous devons calculer est l'entrée de la phase de retour de la couche Softmax, $\partial L / \partial \text{out}_s$, où out_s est la sortie de la couche Softmax

$$\frac{\partial L}{\partial \text{out}_s(i)} = \begin{cases} 0 & \text{if } i \neq c \\ -\frac{1}{p_i} & \text{if } i = c \end{cases}$$

Backprop

- let's calculate the gradient of $out_s(c)$ with respect to the totals (the values passed in to the softmax activation).
- Let t_i be the total for class i . Then we can write $out_s(c)$ as:

$$out_s(c) = \frac{e^{t_c}}{\sum_i e^{t_i}} = \frac{e^{t_c}}{S}$$

where $S = \sum_i e^{t_i}$.

we can see about $\partial L / \partial out_s$ is that it's only nonzero for c , the correct class.

$$S = \sum \Theta^i$$

$$g_{tt}(c) = \frac{e^{t_c}}{S} = e^{t_c S - 1}$$

$$\begin{aligned}\frac{\partial g_{tt}(c)}{\partial t_k} &= \frac{\partial g_{tt}(c)}{\partial S} * \frac{\partial S}{\partial t_k} - \underbrace{\frac{\partial e^{t_c S - 1}}{\partial S}}_{= -e^{t_c S - 1} \cdot e^{t_k}} * \frac{\partial S}{\partial t_k} \\ &= -e^{t_c S - 1} \cdot \frac{\partial S}{\partial t_k} \\ &= -\frac{e^{t_c S} * e^{t_k}}{S^2}\end{aligned}$$

$$K = G$$

$$f \ast g$$

$$S = \sum f_i x_i$$

$$\frac{\partial \ln S(c)}{\partial t_c}$$

$$= \frac{S \cdot e^{t_c} - e^{t_c} \cancel{\downarrow} \cancel{\partial t_c}}{S e}$$

$$\begin{aligned} \frac{\partial x}{\partial t} &= \left(\frac{\partial x}{\partial t_c} \right) \cancel{e^{t_c}} \cancel{S^2} + \cancel{x} \\ &= S e^{t_c} - e^{t_c} \cdot e^{t_c} \\ &= \frac{e^{t_c} (S - e^{t_c})}{S^2} \end{aligned}$$

Demonstration

- Now, consider some class k such that $k \neq c$. We can rewrite $\text{out}_s(c)$ as:
- $\text{out}_s(c) = e^{t_c} S^{-1}$ $\text{out}_s(c) = e^{t_c} S^{-1}$

$$\begin{aligned}\frac{\partial \text{out}_s(c)}{\partial t_k} &= \frac{\partial \text{out}_s(c)}{\partial S} \left(\frac{\partial S}{\partial t_k} \right) \\ &= -e^{t_c} S^{-2} \left(\frac{\partial S}{\partial t_k} \right) \\ &= -e^{t_c} S^{-2} (e^{t_k}) \\ &= \boxed{-\frac{e^{t_c} e^{t_k}}{S^2}}\end{aligned}$$

Demonstration

□ K=c

$$\begin{aligned}\frac{\partial \text{out}_s(c)}{\partial t_c} &= \frac{Se^{t_c} - e^{t_c} \frac{\partial S}{\partial t_c}}{S^2} \\ &= \frac{Se^{t_c} - e^{t_c} e^{t_c}}{S^2} \\ &= \boxed{\frac{e^{t_c} (S - e^{t_c})}{S^2}}\end{aligned}$$

Demonstration

$$\frac{\partial \text{out}_s(k)}{\partial t} = \begin{cases} \frac{-e^{t_c} e^{t_k}}{S^2} & \text{if } k \neq c \\ \frac{e^{t_c} (S - e^{t_c})}{S^2} & \text{if } k = c \end{cases}$$

- We ultimately want the gradients of loss against weights, biases, and input:

$$t = w * \text{input} + b$$

$$\frac{\partial t}{\partial w} = \text{input}$$

$$\frac{\partial t}{\partial b} = 1$$

$$\frac{\partial t}{\partial \text{input}} = w$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \text{out}} * \frac{\partial \text{out}}{\partial t} * \frac{\partial t}{\partial w}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \text{out}} * \frac{\partial \text{out}}{\partial t} * \frac{\partial t}{\partial b}$$

$$\frac{\partial L}{\partial \text{input}} = \frac{\partial L}{\partial \text{out}} * \frac{\partial \text{out}}{\partial t} * \frac{\partial t}{\partial \text{input}}$$

Pooling Layers Backpropagation

- As we remember, in the forward propagation for max pooling, we select the maximum value from each region and transfer them to the next layer. It is therefore clear that during back propagation, the gradient should not affect elements of the matrix that were not included in the forward pass.
- In practice, this is achieved by creating a mask that remembers the position of the values used in the first phase, which we can later utilize to transfer the gradients

Max Pooling Backpropagation

Input

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

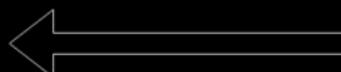
Output

Forward
pass



Mask

Backward
pass

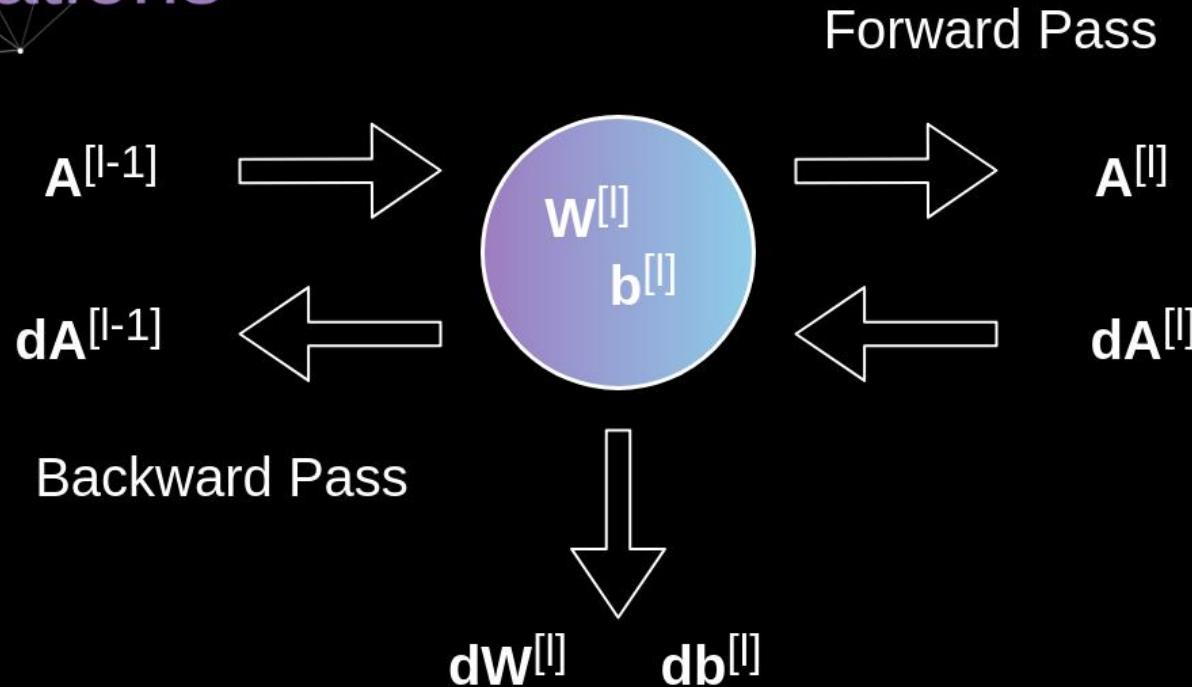


Pooling Layers Backpropagation

- $\partial L / \partial \text{input} = 0$ for non-max pixels
- $\partial \text{output} / \partial \text{input} = 1$ for max pixel
- $\partial L / \partial \text{input} = \partial L / \partial \text{output}$.

Convolutional Layer Backpropagation

Single Layer
Operations



$$\mathbf{dA}^{[l]} = \frac{\partial L}{\partial \mathbf{A}^{[l]}} \quad \mathbf{dZ}^{[l]} = \frac{\partial L}{\partial \mathbf{Z}^{[l]}} \quad \mathbf{dW}^{[l]} = \frac{\partial L}{\partial \mathbf{W}^{[l]}} \quad \mathbf{db}^{[l]} = \frac{\partial L}{\partial \mathbf{b}^{[l]}}$$

Formula

$$\text{dZ}^{[l]} = \underbrace{\text{dA}^{[l]}}_{=} * \underbrace{g'(\mathbf{z}^{[l]})}_{=}$$

$$\text{dA}^+ = \sum_{m=0}^{n_h} \sum_{n=0}^{n_w} \mathbf{w} \cdot \underbrace{\text{dZ}[m, n]}_{=}$$

Convolutional layer Backpropagation

Rotated Kernel

f_{22}	f_{21}
f_{12}	f_{11}

*

Previous layer
derivative

Diagram illustrating the convolution operation:

The diagram shows a 2x2 rotated kernel (f₂₂, f₂₁; f₁₂, f₁₁) multiplied by a 2x2 previous layer derivative (dZ₁₁, dZ₁₂; dZ₂₁, dZ₂₂). The result is a 2x2 convolutional layer derivative.

dZ ₁₁	dZ ₁₂
dZ ₂₁	dZ ₂₂

=

Convolutional layer
derivative

Back example

$$\text{out}(i, j) = \text{convolve}(\text{image}, \text{filter})$$

$$= \sum_{x=0}^3 \sum_{y=0}^3 \text{image}(i + x, j + y) * \text{filter}(x, y)$$

$$\frac{\partial \text{out}(i, j)}{\partial \text{filter}(x, y)} = \text{image}(i + x, j + y)$$

$$\frac{\partial L}{\partial \text{filter}(x, y)} = \sum_i \sum_j \frac{\partial L}{\partial \text{out}(i, j)} * \frac{\partial \text{out}(i, j)}{\partial \text{filter}(x, y)}$$

Back

- In the standard MLP, we can define an error of neuron j as:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

where z_j^l is just:

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

- and for clarity, $a_j^l = \sigma(z_j^l)$ where σ is an activation function such as sigmoid, hyperbolic tangent or relu.

Demonstration

- In CNN matrix multiplications are replaced by convolutions. And we have $\tilde{z}_{x,y}$

$$z_{x,y}^{l+1} = w^{l+1} * \sigma(z_{x,y}^l) + b_{x,y}^{l+1} = \sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x-a,y-b}^l) + b_{x,y}^{l+1}$$

- We start from statement:

$$\delta_{x,y}^l = \overline{\frac{\partial C}{\partial z_{x,y}^l}} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z_{x',y'}^{l+1}} \frac{\partial z_{x',y'}^{l+1}}{\partial z_{x,y}^l}$$

$$\frac{\partial C}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \frac{\partial C}{\partial z_{x',y'}^{l+1}} \frac{\partial z_{x',y'}^{l+1}}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} \frac{\partial (\sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{x'-a,y'-b}^l) + b_{x',y'}^{l+1})}{\partial z_{x,y}^l}$$

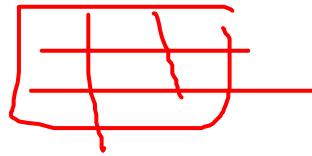
- First term is replaced by definition of error, while second has become large because we put it here expression on $\tilde{z}_{x',y'}^{l+1}$

- But all components of sums equal 0, except these ones that are indexed: $x=x'-a$ and $y=y'-b$. So:

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} \frac{\partial \left(\sum_a \sum_b w_{a,b}^{l+1} \sigma(z_{\underbrace{x'-a}_{\text{red}}, \underbrace{y'-b}_{\text{red}})^l + b_{x',y'}^{l+1} \right)}{\partial z_{x,y}^l} = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{a,b}^{l+1} \sigma'(z_{x,y}^l)$$

- If $x=x'-a$ and $y=y'-b$ then $a=\underline{x'-x}$ and $b=\underline{y'-y}$ so we can reformulate above equation to:

$$\sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{a,b}^{l+1} \sigma'(z_{x,y}^l) = \sum_{x'} \sum_{y'} \delta_{x',y'}^{l+1} w_{\underline{x'-x}, \underline{y'-y}}^{l+1} \sigma'(z_{x,y}^l)$$



$$ROT180(w_{x,y}^{l+1}) = w_{-x,-y}^{l+1}$$

$$\frac{\partial C}{\partial w_{a,b}^l} = \sum_x \sum_y \frac{\partial C}{\partial z_{x,y}^l} \frac{\partial z_{x,y}^l}{\partial w_{a,b}^l} = \sum_x \sum_y \delta_{x,y}^l \frac{\partial (\sum_{a'} \sum_{b'} w_{a',b'}^l \sigma(z_{x-a',y-b'}^l) + b_{x,y}^l)}{\partial w_{a,b}^l} =$$

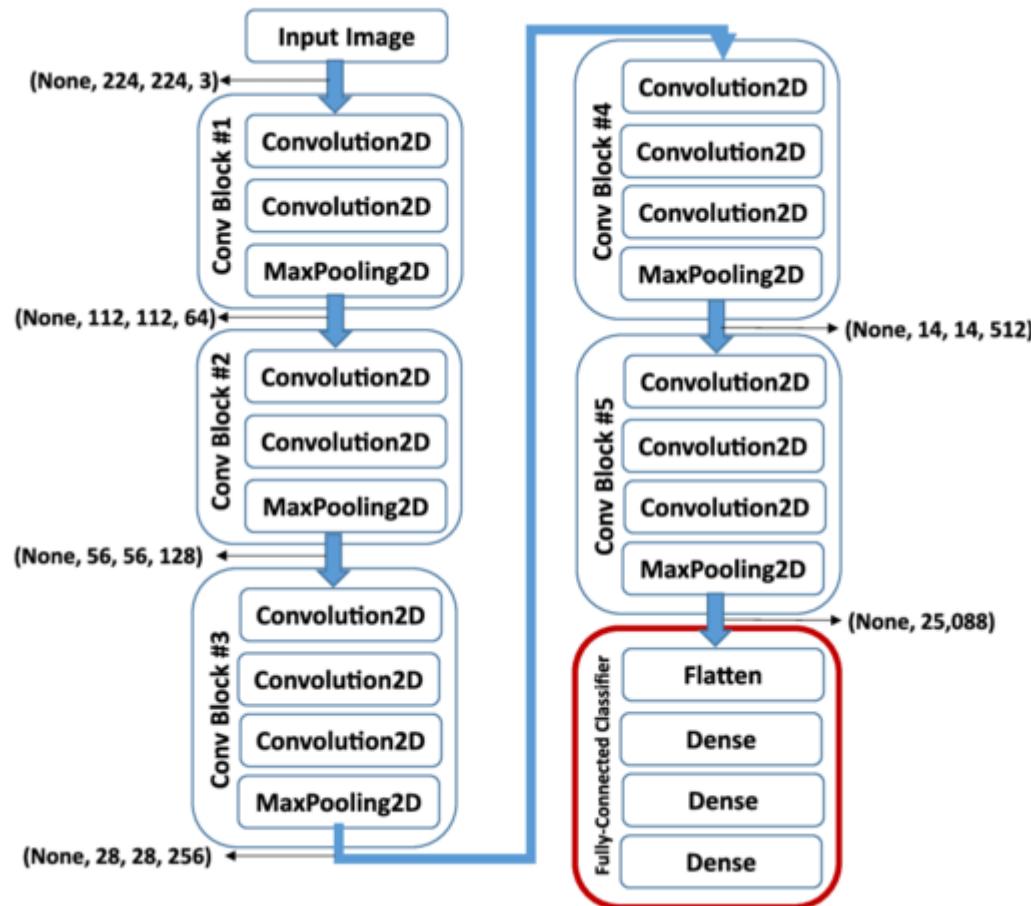
$$\sum_x \sum_y \delta_{x,y}^l \sigma(z_{x-a,y-b}^{l-1}) = \delta_{a,b}^l * \sigma(z_{-a,-b}^{l-1}) = \delta_{a,b}^l * \sigma(ROT180(z_{a,b}^{l-1}))$$



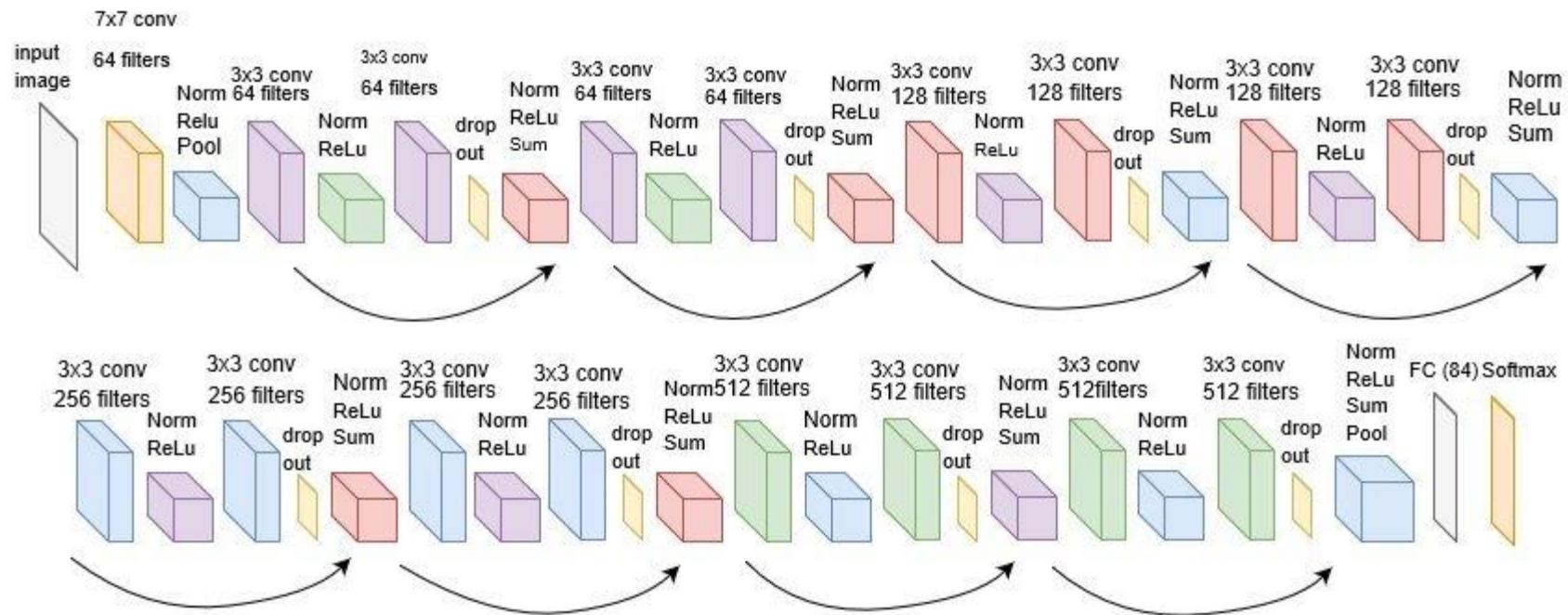
EXAMPLE

- LeNet (le plus simple, pour découvrir les CNN),
- AlexNet (lancé en 2012),
- ZFNet (amélioration d'AlexNet tombée dans l'oubli),
- GoogLeNet (parent du célèbre Inception, un algorithme de reconnaissance d'image avec d'autres subtilités, au même titre que YOLO, MobileNet, etc...),
- VGGNet (encore très utilisé et puissant),
- ResNet (pareil, même s'il s'éloigne d'un CNN traditionnel)

Architecture du CNN : VGG 16



Architecture du CNN : ResNet 18



MobileNet

- MobileNet utilise des convolutions séparables en profondeur .
- Il réduit considérablement le nombre de paramètres par rapport au réseau avec des convolutions régulières avec la même profondeur dans les filets. Il en résulte des réseaux de neurones profonds légers.
- Une convolution séparable en profondeur (**depthwise separable convolution**) est réalisée à partir de deux opérations.
 - **Convolution en profondeur (Depthwise convolution)**
 - **Convolution ponctuelle (Pointwise convolution)**

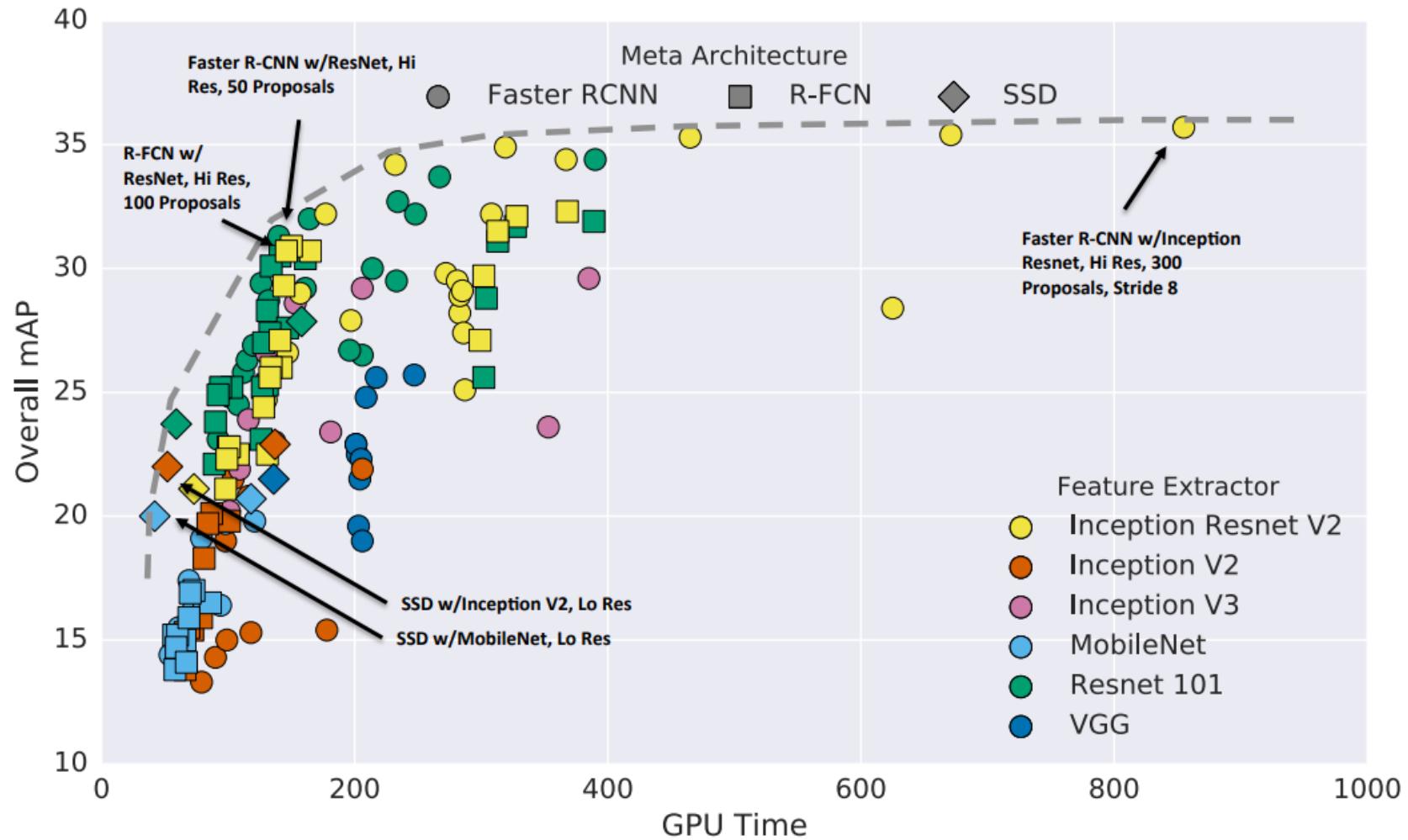
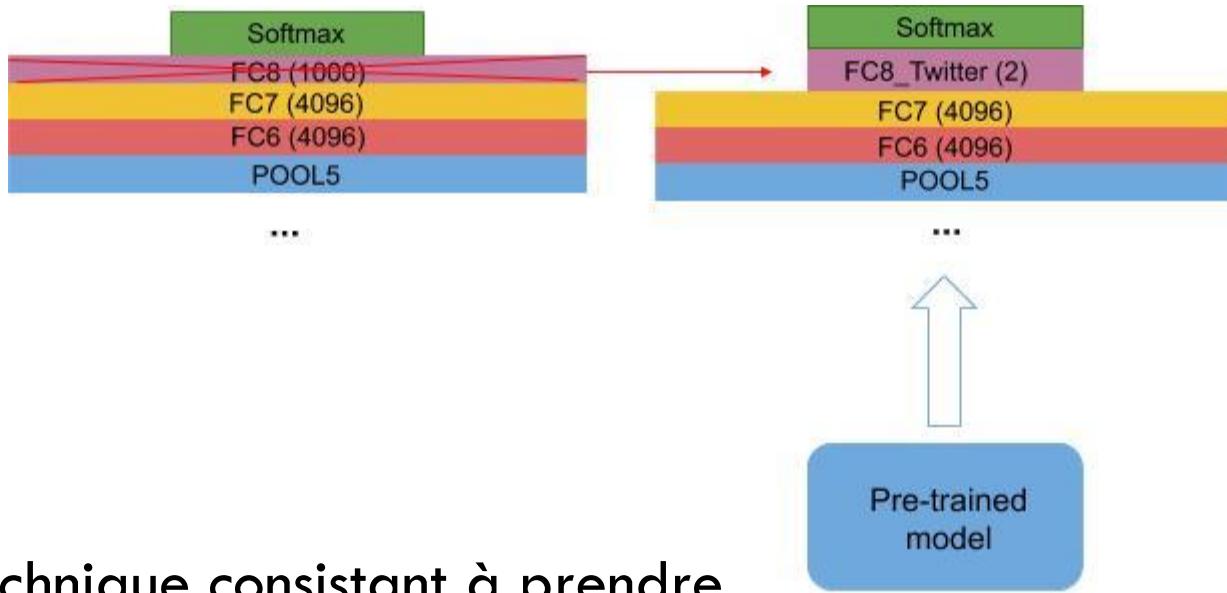


Figure 2: Accuracy vs time, with marker shapes indicating meta-architecture and colors indicating feature extractor. Each (meta-architecture, feature extractor) pair can correspond to multiple points on this plot due to changing input sizes, stride, etc.

le fine-tuning (transfer learning)

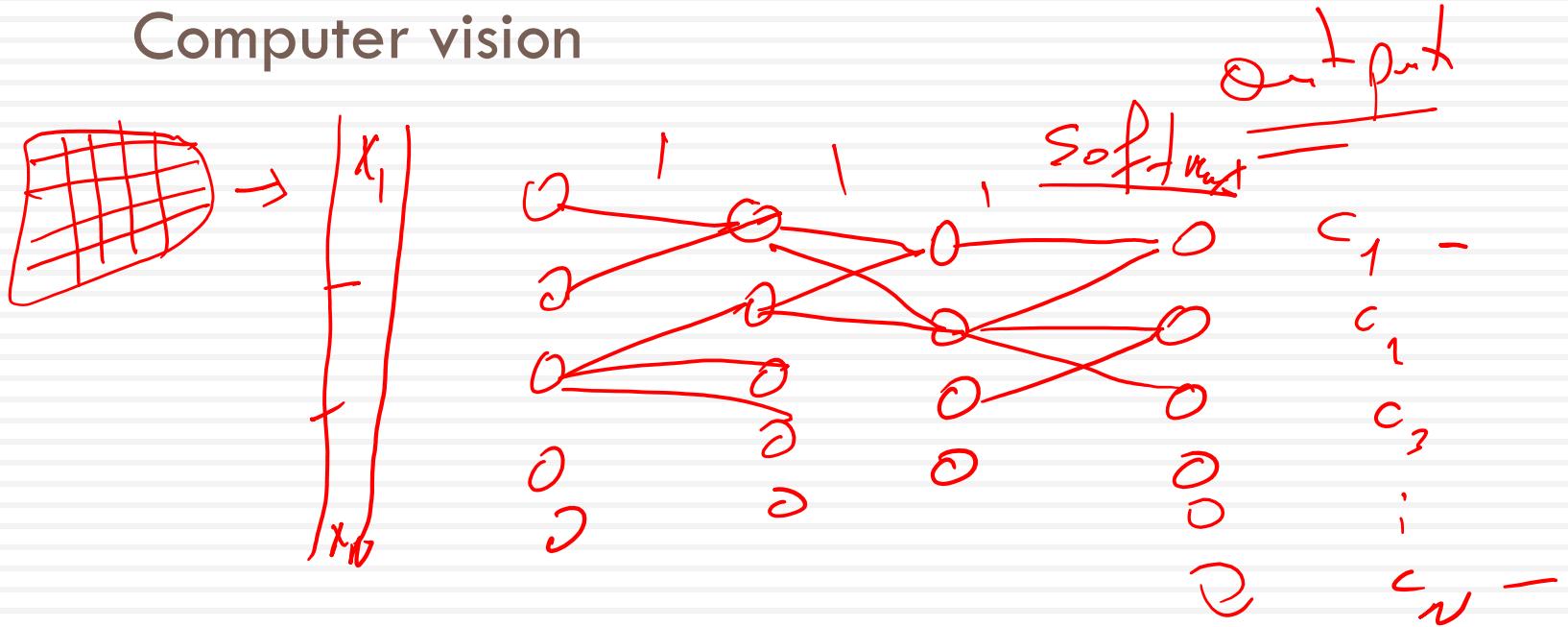
Adapter un autre CNN à vos besoins

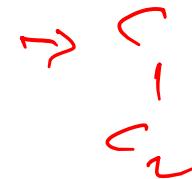
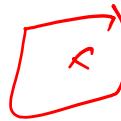


Il s'agit d'une technique consistant à prendre un CNN entraîné pour résoudre un problème spécifique, et de l'adapter à un autre problème assez proche.

Applications de CNN

Computer vision





- une branche de l'intelligence artificielle dont l'objectif est d'analyser des informations contenues dans des images ou des vidéos. Nous pouvons distinguer plusieurs tâches de vision par ordinateur:
- **Classification d'image** : prédire le type ou la classe d'un objet dans une image.
 - Entrée : une image avec un seul objet, comme une photographie.
 - Sortie : une étiquette de classe (par exemple un ou plusieurs entiers mappés sur des étiquettes de classe).
- ✓ □ **Localisation d'objets** : localisez la présence d'objets dans une image et indiquez leur emplacement avec un cadre de sélection.
 - Entrée : une image avec un ou plusieurs objets, comme une photographie.
 - Sortie : une ou plusieurs boîtes englobantes (définies par exemple par un point, une largeur et une hauteur).
- **Détection d'objets** : localisez la présence d'objets avec un cadre de sélection et les types ou classes des objets localisés dans une image.
 - Entrée : une image avec un ou plusieurs objets, comme une photographie.
 - Sortie : une ou plusieurs boîtes englobantes (définies par exemple par un point, une largeur et une hauteur) et une étiquette de classe pour chaque boîte englobante.
- **Segmentation d'objet** ou «segmentation sémantique», où les instances d'objets reconnus sont indiquées en mettant en évidence les pixels spécifiques de l'objet au lieu d'un cadre de délimitation grossier

Vision : tâches principales

Classification



CAT


Détection



DOG, DOG, CAT

Segmentation



GRASS, CAT,
TREE, SKY

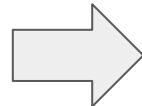
Segmentation

d'instances



DOG, DOG, CAT

Classification

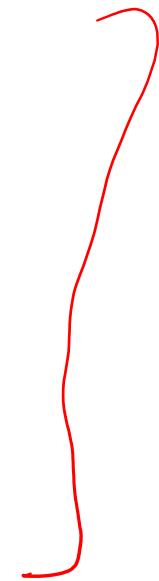


Cochon

Chat

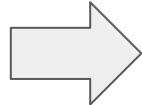
Chien

Cheval



- La sortie est une valeur discrète qui représente une catégorie

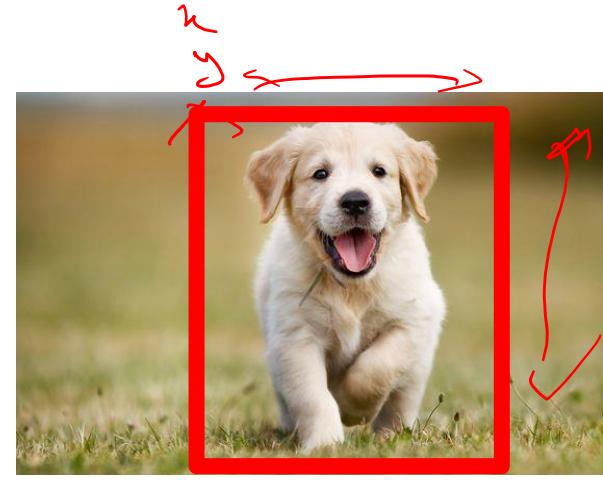
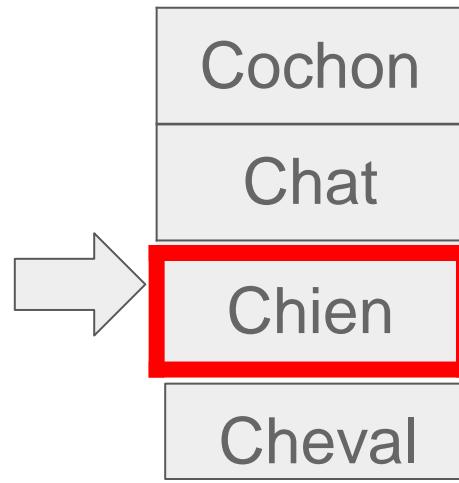
Classification Multi-étiquette



Cochon	Cochon
Chat	Chat
Chien	Chien
Cheval	Cheval

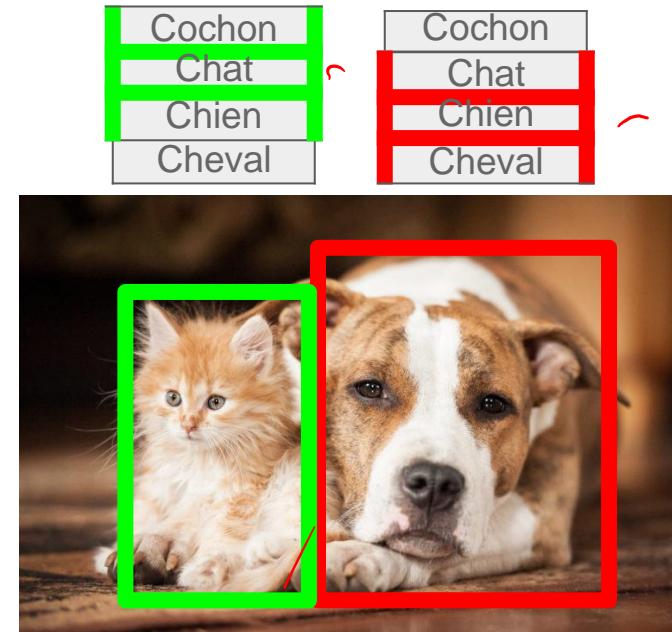
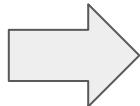
- La sortie sont plusieurs valeurs discrètes qui représentent toutes les catégories dans l'image

Localisation ✓



- La sortie est un valeur discret qui représente une catégorie + une boîte de délimitation ("bounding box")

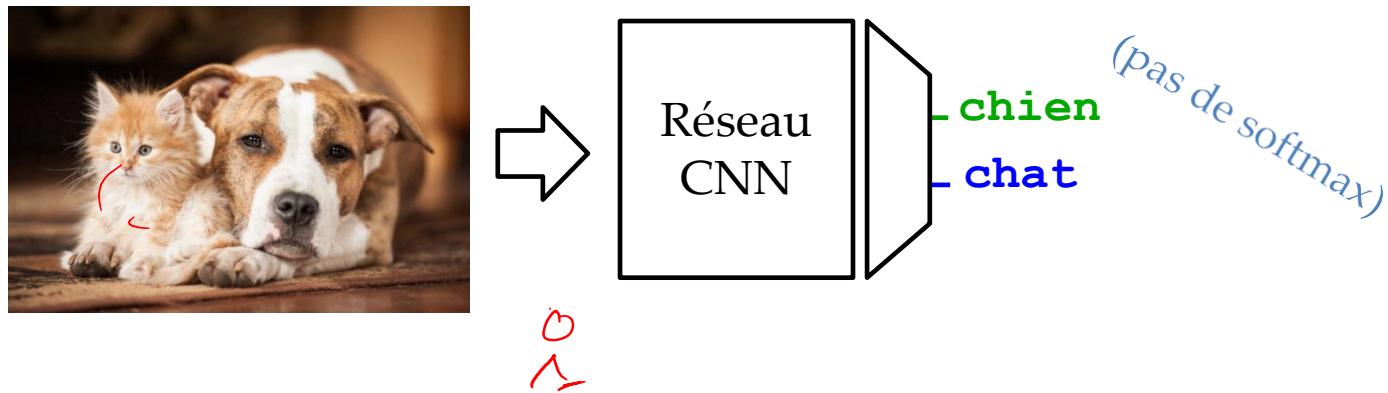
Détection



- La sortie sont des boîtes de délimitation associées à des valeurs discrets qui représentent la catégorie de chaque boîte

Détection

- Une des difficultés est que l'on ne connaît pas le **nombre exact d'instances** dans l'image
 - si on savait d'avance que les images ne contiennent au maximum qu'un chat et un chien :



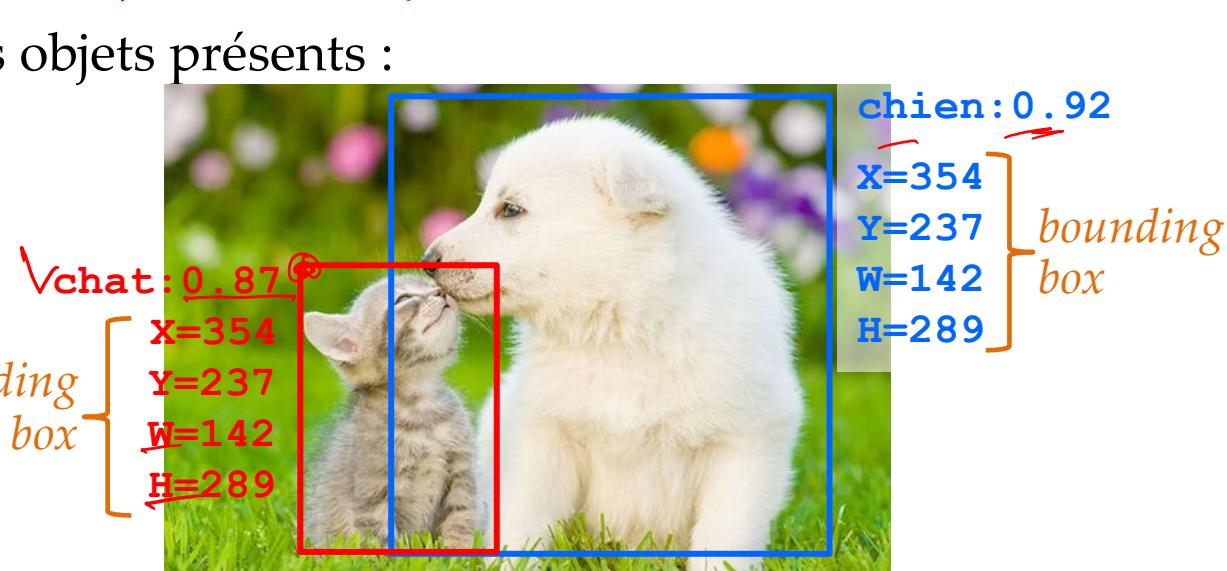
- Sujet de recherche très fertile en soit

Détection : définition

100%

- Pour :
 - une image d'entrée ✓
 - une liste prédéterminée de classes ✓
- Trouver, pour tous les objets présents :

- la position (bounding box) ✓
- la classe ✓

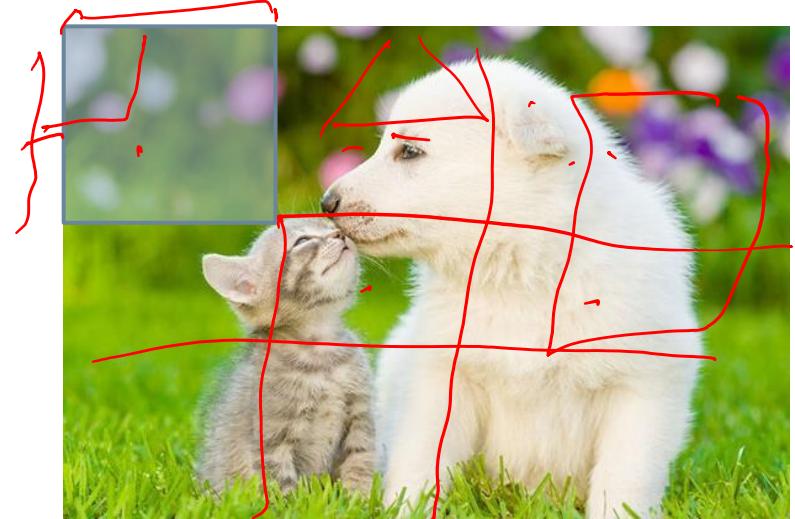


- Nombre de prédictions va varier d'une image à l'autre
 - Architecture doit en tenir compte

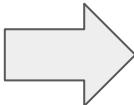
Détection via classification

~  10.4 | 1024

- Possible de faire de la détection par un réseau de classification
- Approche par **fenêtre coulissante (crops)**
- Passe chaque crop dans un réseau classificateur
- Conserve n prédictions les plus confiantes ↗
- Choix de la géométrie de la fenêtre :
 - taille, aspect ratio
- **Fastidieux**, car des milliers de passes dans le réseau classificateur : dizaines de secondes

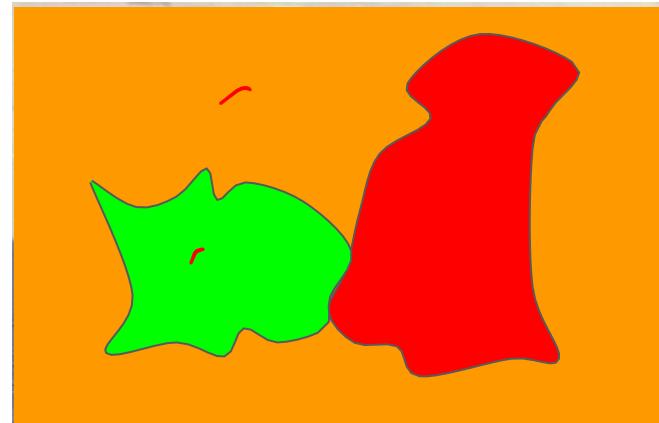


Segmentation



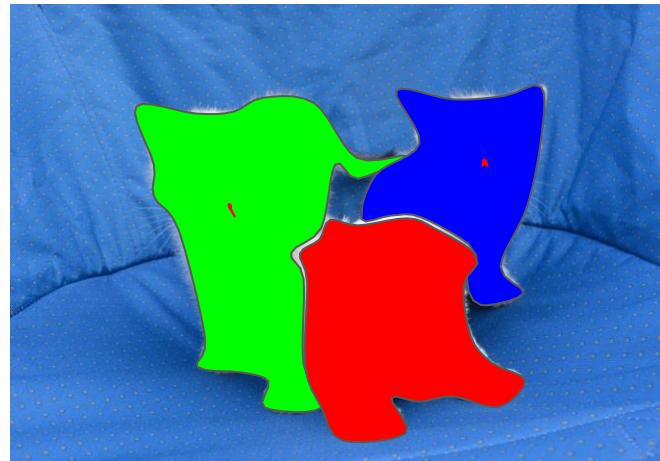
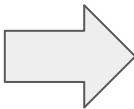
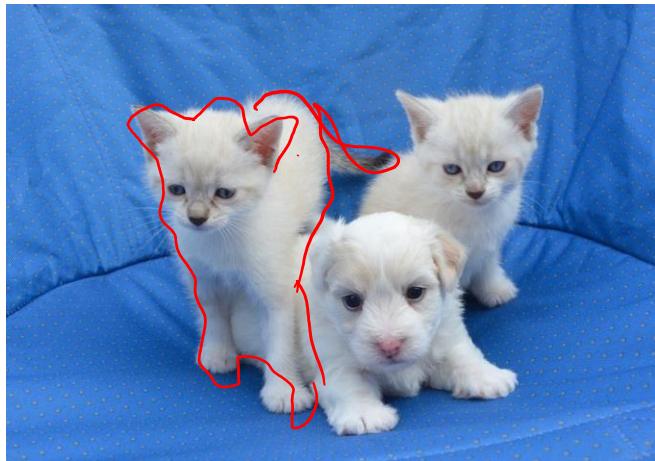
Cochon
Chat
Chien
Cheval

Cochon
Chat
Chien
Cheval



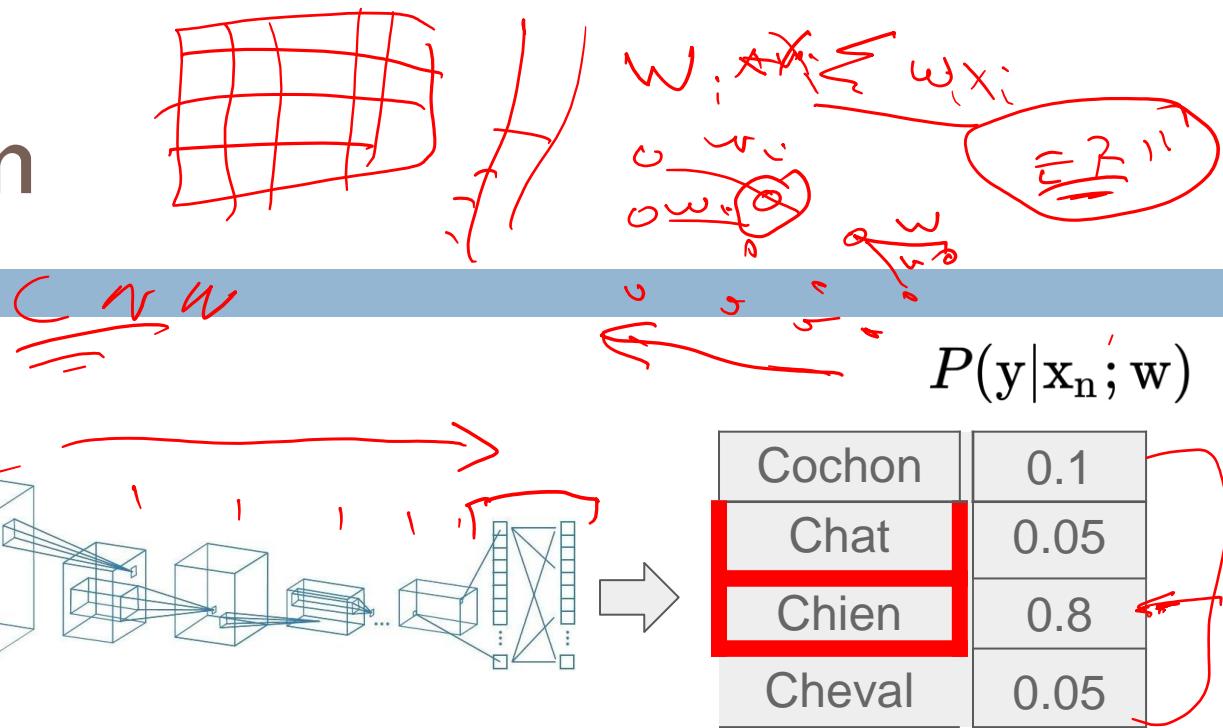
- Chaque pixel de l'image est associé à une catégorie

Segmentation d'instances



- Chaque instance d'objet est segmentée indépendamment d'autres

Classification



- Fonction objectif: ✓
 - log vraisemblance(softmax) = entropie-croisée

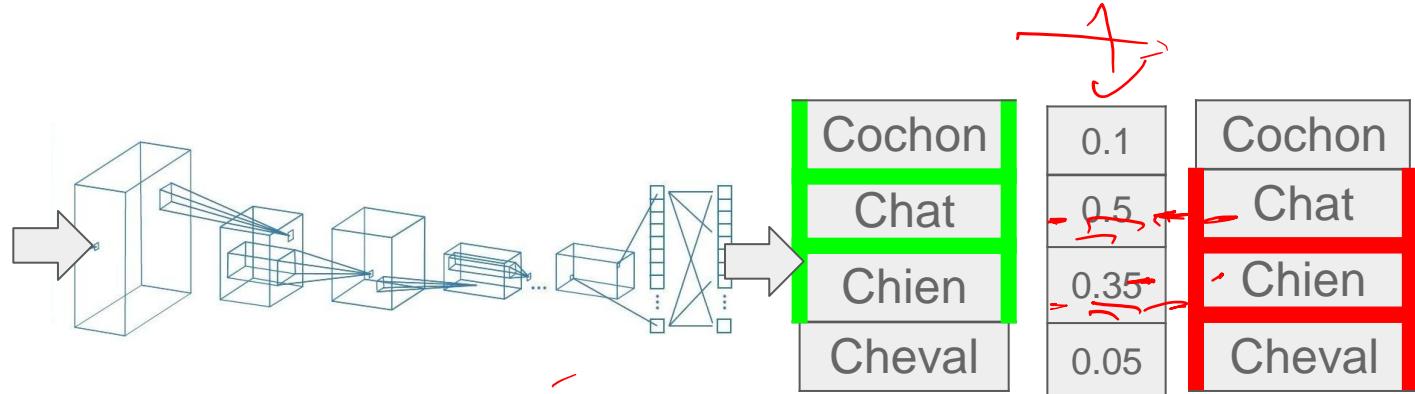
$$E_{cls} = - \sum_{n=1}^N \log(P(y = d_n | x_n; w))$$

- Base de données:
 - ImageNet, CIFAR10, CIFAR100, etc.

$$\text{softmax}(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^D \exp(y_j)}$$

10
10
chat

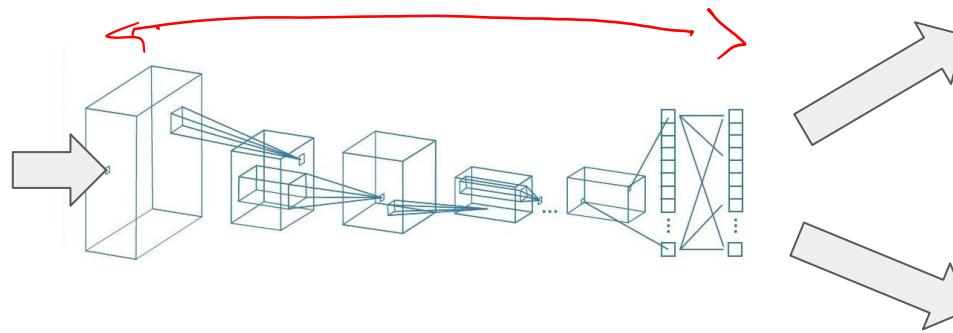
Classification Multi-étiquette



- Entraînement:
 - Comme dans la classification normale (entropie-croisée), mais la probabilité est distribuée entre les catégories présentes dans l'image
- Évaluation:
 - soit on connaît le nombre d'objets dans l'image
 - soit on décide une seuil sur le “softmax” (ex: $P>0.3$)

3
7
P>0.3

Localisation

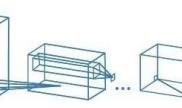
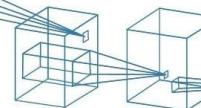
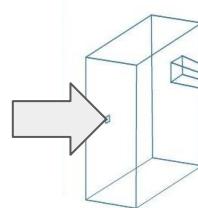


Cochon
Chat
Chien
Cheval

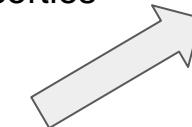


- **Apprentissage multitâche:**
 - classification pour choisir la catégorie
 - approximation pour estimer la boîte de délimitation

Localisation



couche
totalement
connectée
 K sorties

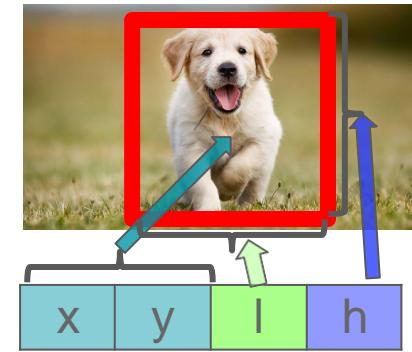


E_{cls}	
0.1	Cochon
0.05	Chat
0.8	Chien
0.05	Cheval

couche
totalement
connectée
4 sorties



- Apprentissage multitâche:



E_{loc}

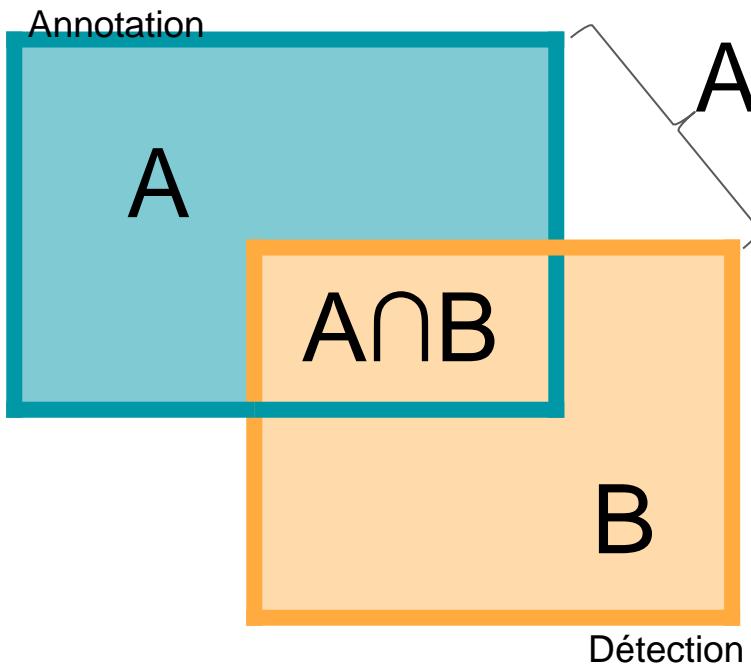
- $E = E_{cls} + E_{loc}$

- $E_{cls} = - \sum_{n=1}^N \log(P(y = d_n | \mathbf{x}_n; \mathbf{w}))$

- $E_{loc} = \sum_{n=1}^N \|f(\mathbf{x}_n; \mathbf{w}) - d_n\|_2^2$

Evaluation

IoU: Intersection sur union



$$1 < IoU < 0$$

si $IoU > 0.5$ vrai positif

si $IoU < 0.5$ faux positif

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

La reconnaissance d'objets

Classification



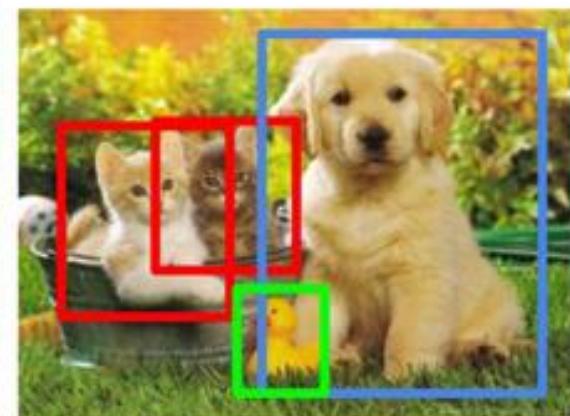
CAT

Classification + Localization



CAT

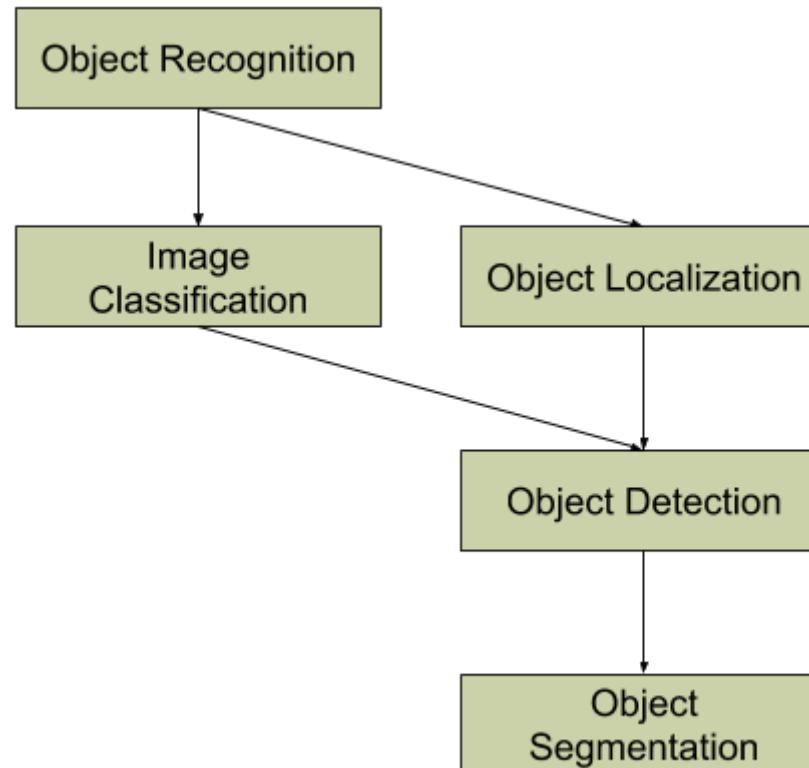
Object Detection



CAT, DOG, DUCK

la reconnaissance d'objets

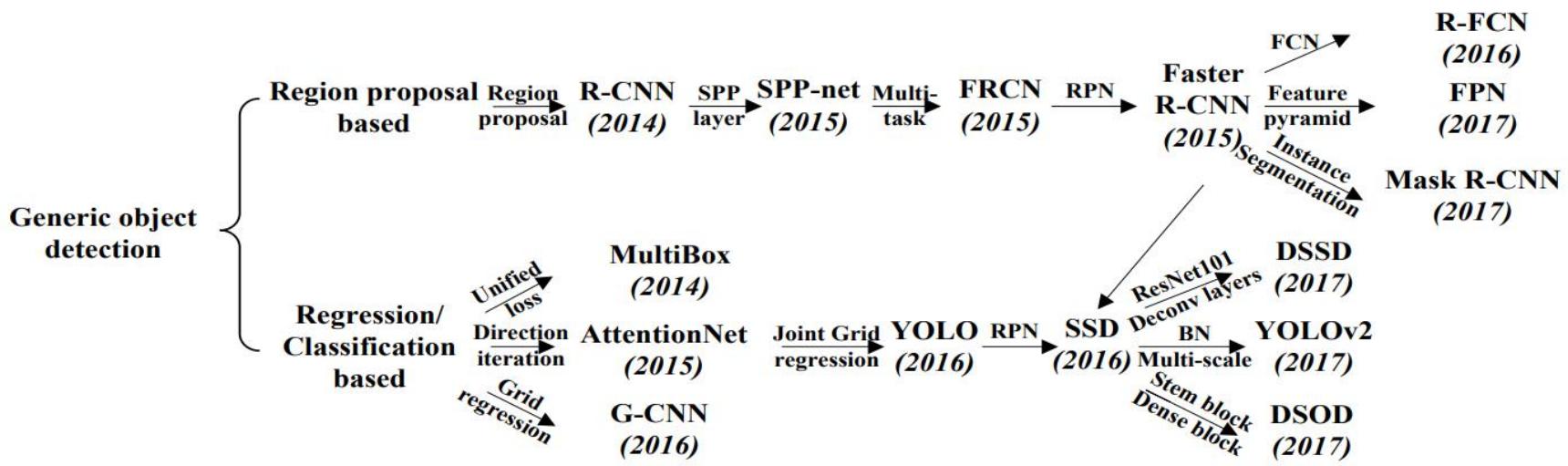
- la reconnaissance d'objets fait référence à une suite de tâches de vision par ordinateur.



Performances

- Les performances d'un modèle pour la classification d'images sont évaluées à l'aide de **l'erreur de classification moyenne** sur les étiquettes de classe prédites.
- Les performances d'un modèle pour la localisation d'un seul objet sont évaluées à l'aide de **la distance entre la boîte englobante attendue et prévue** pour la classe attendue.
- Les performances d'un modèle pour la reconnaissance d'objets sont évaluées à l'aide de **la précision et du rappel** à travers chacune des meilleures boîtes englobantes correspondantes pour les objets connus de l'image.

Two types of frameworks: region proposal based and regression/classification based.



Catégories d'algorithmes

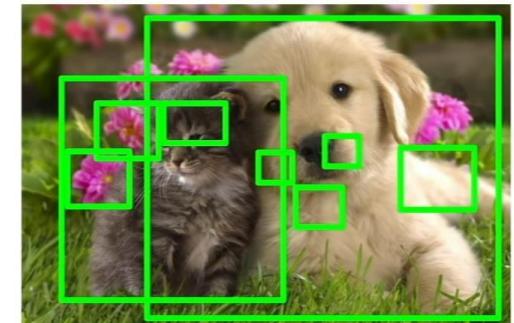
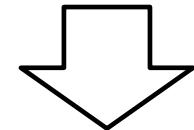
- **Basé sur des régions proposées** (*region proposal*)
 - R-CNN
 - Fast-RCNN
 - Faster RCNN
- **Grille fixe** (*grid-based*)
 - YOLO (v1, v2, v3)
 - SSD (single-shot detection)

Famille de modèles R-CNN



Region proposal (classique)

- Algorithmes qui proposent des **régions prometteuses** en terme de présence d'objets
- Basés parfois sur des heuristiques (les premiers algo)
 - Recherche de *blobs*
 - Distributions particulières de contours (*edge*)
- Selective Search propose 1000 régions en quelques seconds sur CPU (*pas temps-réel*)
- Voir aussi Edge Boxes



Region proposal

- Au lieu d'utiliser une fenêtre coulissante on utilise de proposition d'objets, ex: **selective search**
- On pré-sélectionne seulement les fenêtres qui représentent un objet!
- Basé sur segmentation mais aussi d'autres méthodes...
- De 10k fenêtres (ou propositions) à <1000, avec une performance similaire!

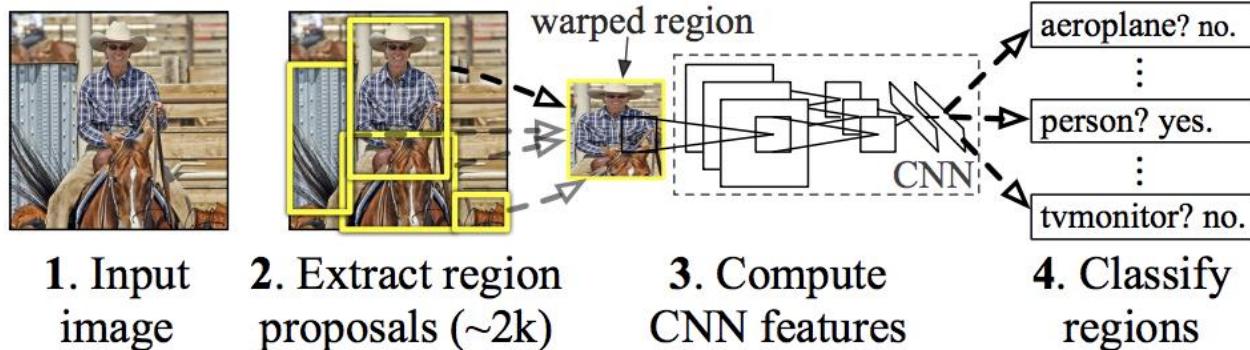
[“Selective Search”, K. van de Sande et al, ICCV’11]

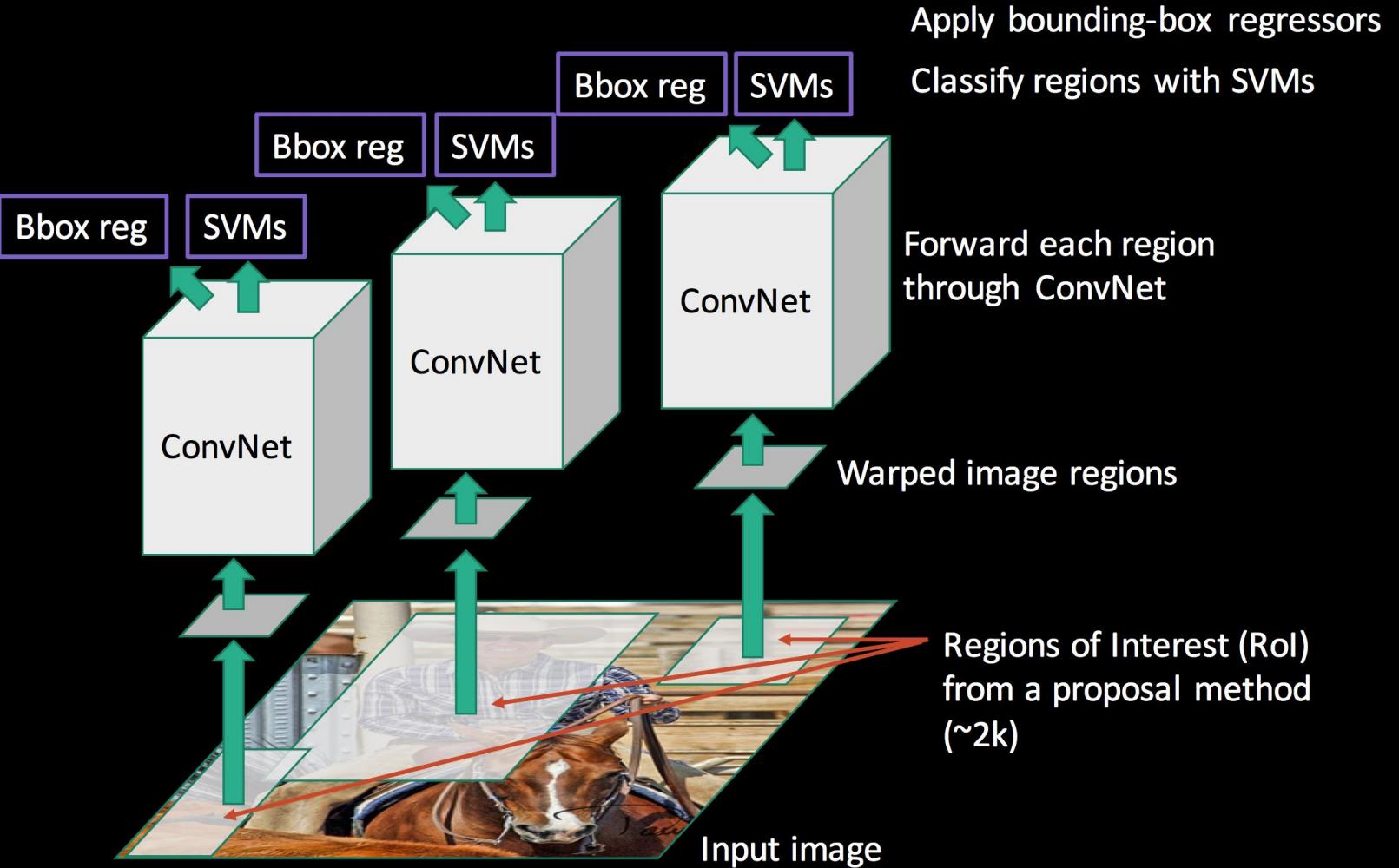


R-CNN

- Le modèle R-CNN est composé de trois modules:
 - Module 1: Proposition de région . Génère et extraire des propositions de régions indépendantes de la catégorie (des boîtes englobantes candidates)
 - Module 2: Extracteur de fonctionnalités . Extraire la caractéristique de chaque région candidate, par exemple en utilisant un réseau neuronal à convolution profonde. (CNN profond d'AlexNet)
 - Module 3: Classificateur . Classifier les entités comme l'une des classes connues. modèle SVM linéaire.

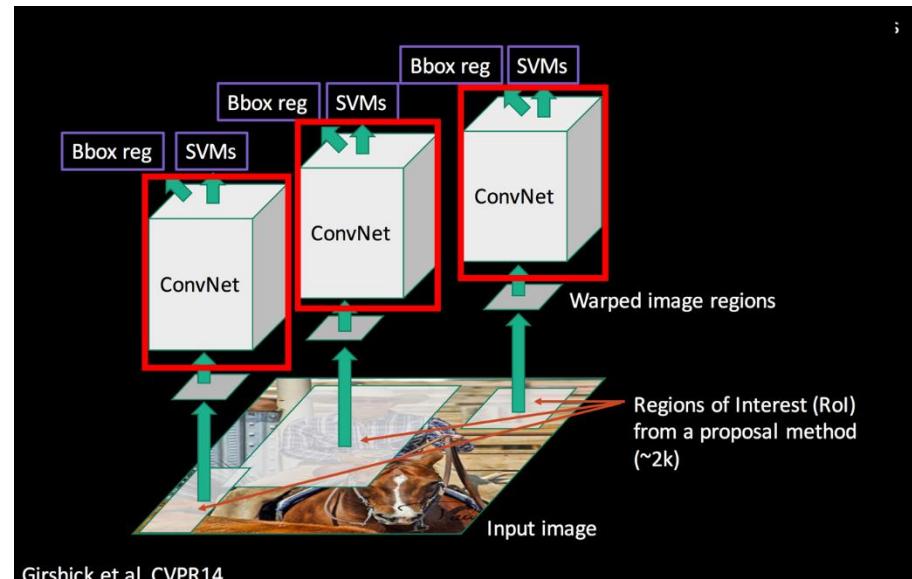
R-CNN: *Regions with CNN features*





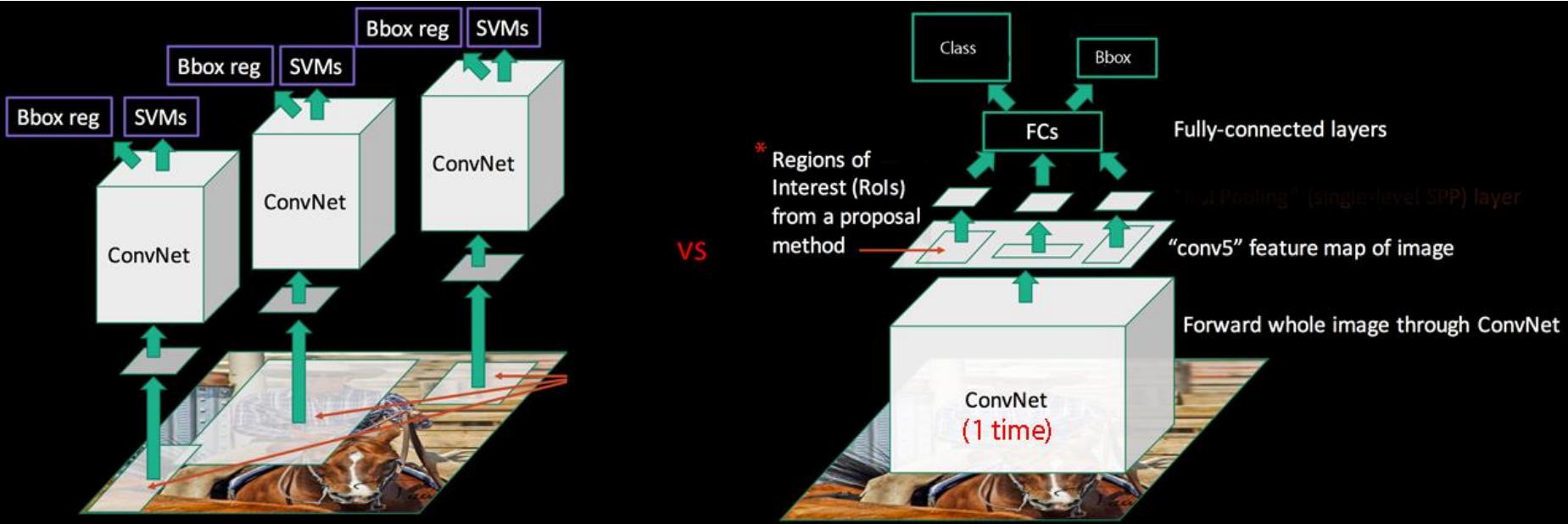
Inconvénients

- R-CNN est lent dans la formation et l'inférence.
Nous avons 2000 propositions dont chacune devait être traitée par un CNN pour extraire des fonctionnalités, et par conséquent, R-CNN répétera le ConvNet 2000 fois pour extraire des fonctionnalités.



Solution

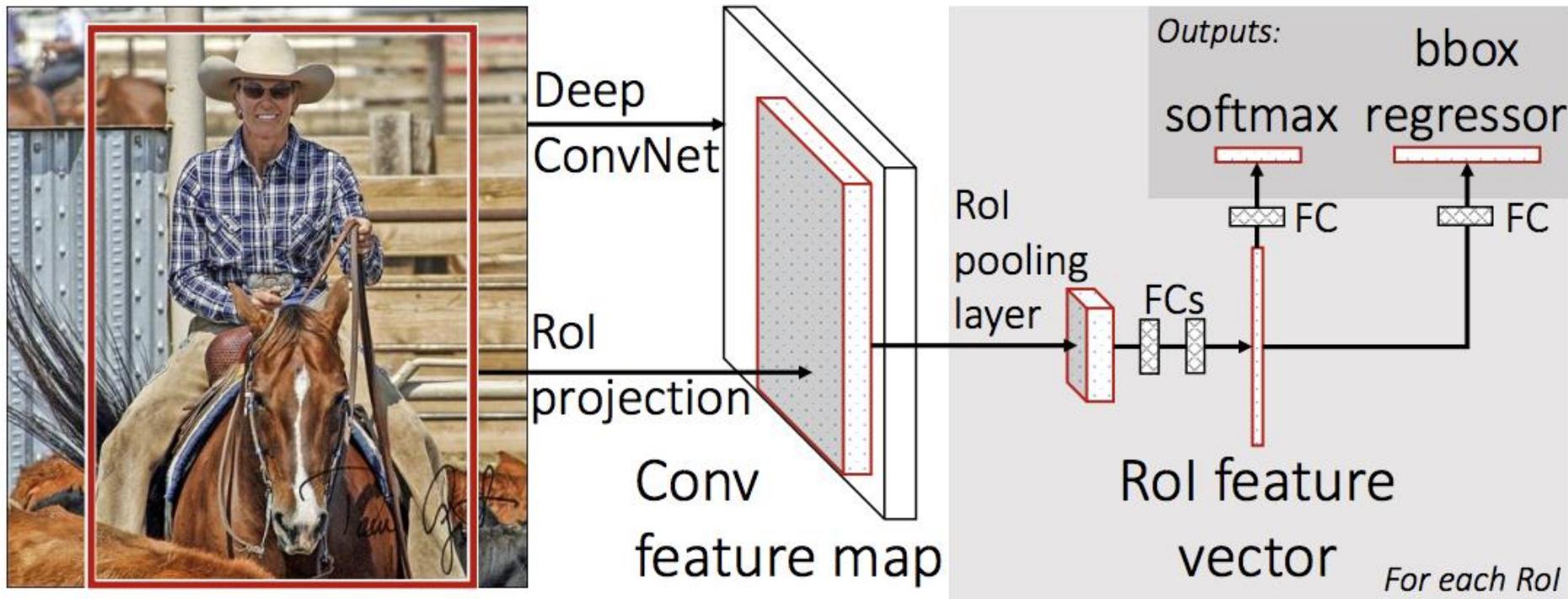
- Par conséquent, au lieu de convertir 2000 régions en cartes de caractéristiques correspondantes, nous convertissons l'image entière **une fois**.



Fast R-CNN

https://openaccess.thecvf.com/content_iccv_2015/papers/Girshick_Fast_R-CNN_ICCV_2015_paper.pdf

- Compte tenu du grand succès de R-CNN, Ross Girshick (Microsoft Research), a proposé une extension pour résoudre les problèmes de vitesse de R-CNN dans un article de 2015 intitulé « Fast R-CNN ».

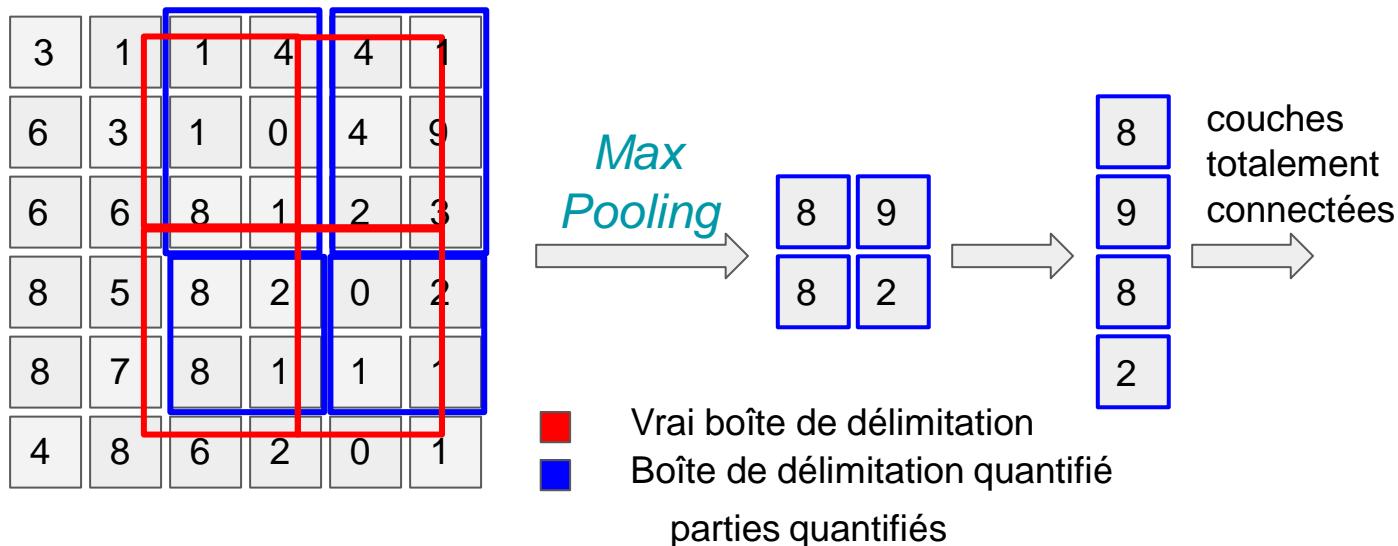


Fast R-CNN

- Fast R-CNN est proposé comme un modèle unique au lieu d'un pipeline pour apprendre et produire directement des régions et des classifications.
- L'architecture du modèle prend la photographie comme un ensemble de propositions de régions qui sont transmises à travers un réseau neuronal convolutif profond.
- Un CNN pré-entraîné, tel qu'un VGG-16, est utilisé pour l'extraction de caractéristiques.
- La fin du CNN profond est une couche personnalisée appelée couche de **regroupement de régions d'intérêt (ROI)**, ou **regroupement de RoI**, qui extrait des fonctionnalités spécifiques pour une région candidate d'entrée donnée.

Pooling de la région d'intérêt

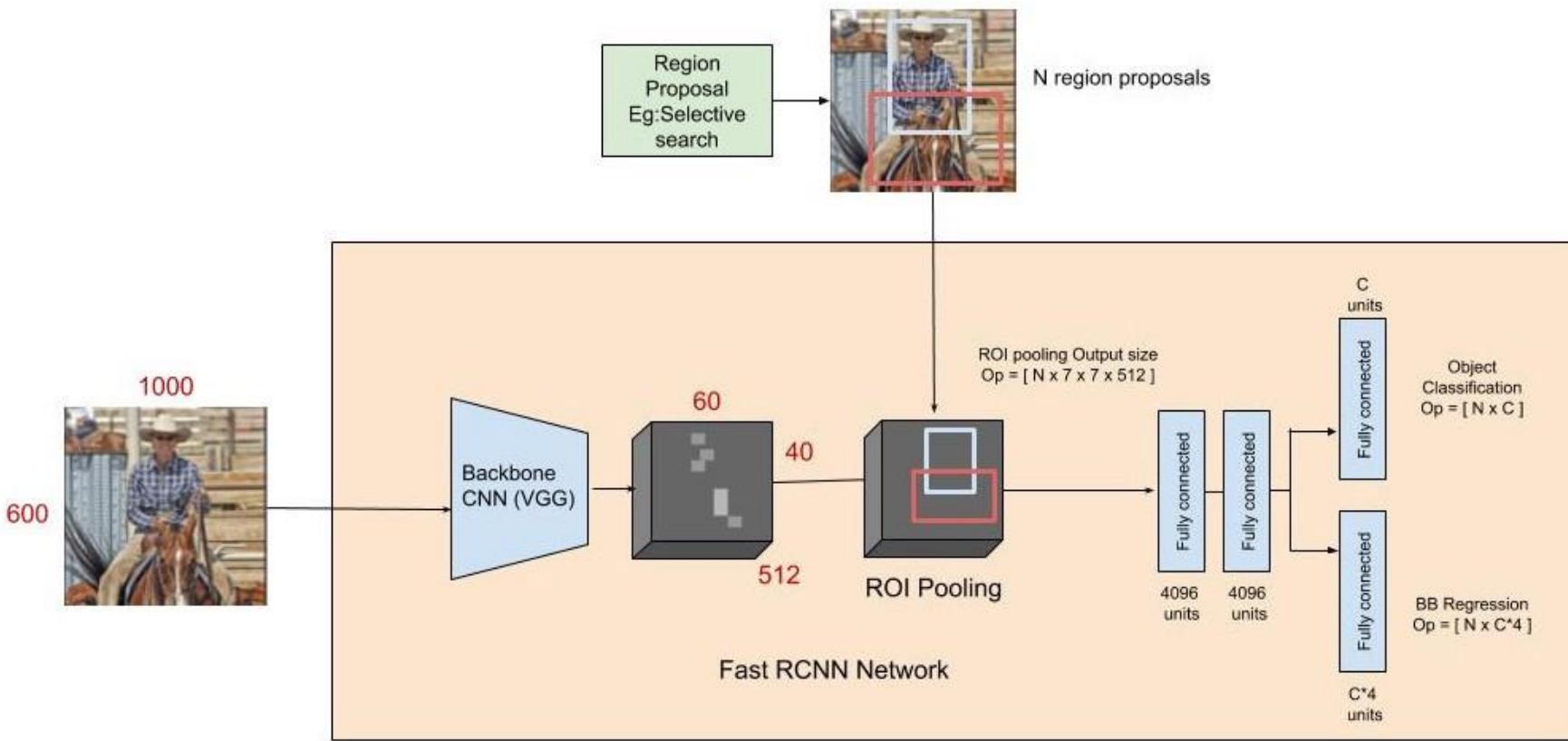
- ❑ Régions définies par des propositions d'objets
Régions divisés en 3x3 ou 7x7 parties
- ❑ (*dans l'exemple 2x2 pour simplicité*)



Fast R-CNN

- Le Fast R-CNN consiste en un CNN (généralement pré-formé sur la tâche de classification ImageNet) avec sa couche de pooling finale remplacée par une couche de «pooling ROI» et sa couche FC finale est remplacée par deux branches - a $(K + 1)$ une branche de couche softmax de catégorie et une branche de régression de boîte englobante spécifique à la catégorie.

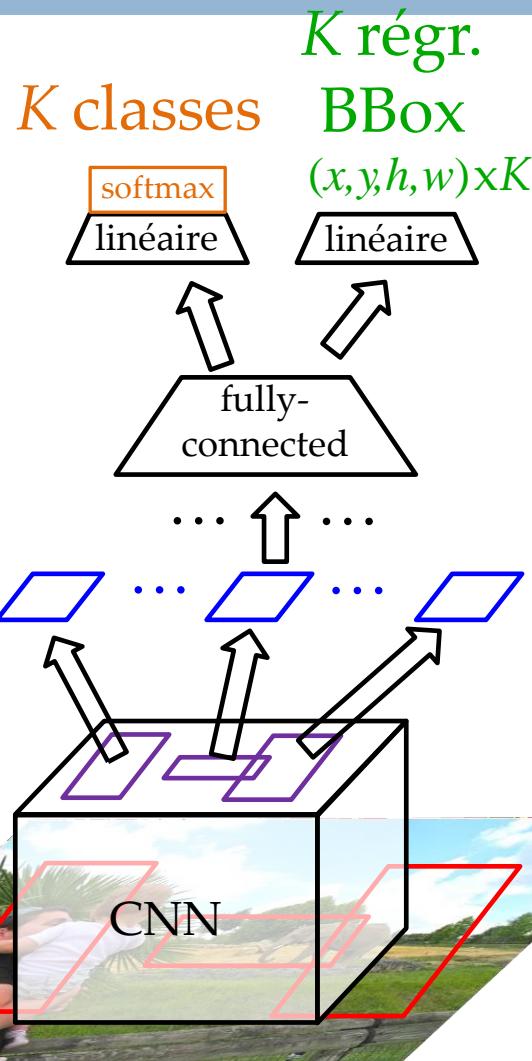
Fast R-CNN



Temps de calcul

- **2 sec Prop RoI**
- 0.3 sec partie réseau

Fast R-CNN



Perte régression

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Ajustement des la *bounding* box via tête de régression

Classification avec réseau linéaire + softmax

Déforme vers taille unique d'image d'entrée

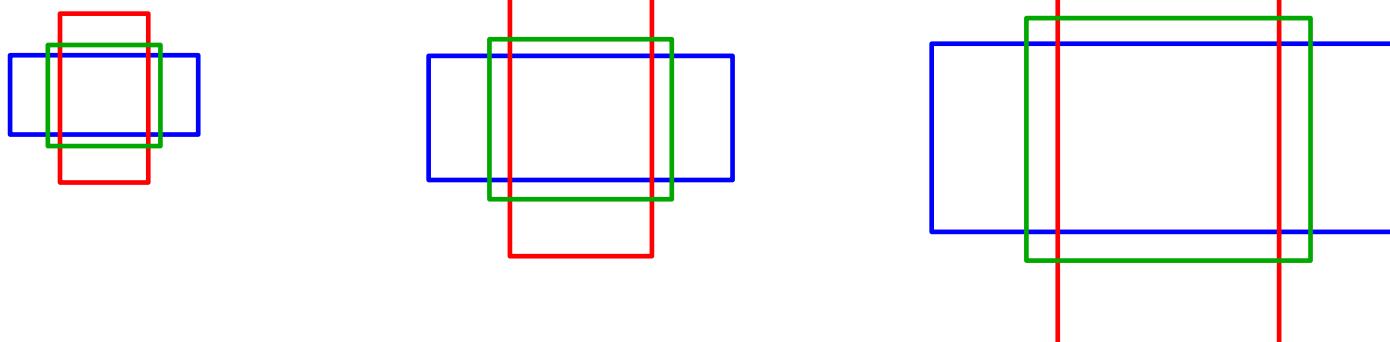
Projette les RoI dans le feature map

Passe dans un backbone CNN

Applique un algorithme de proposition de régions (RoI) (Selective Search)

Faster R-CNN

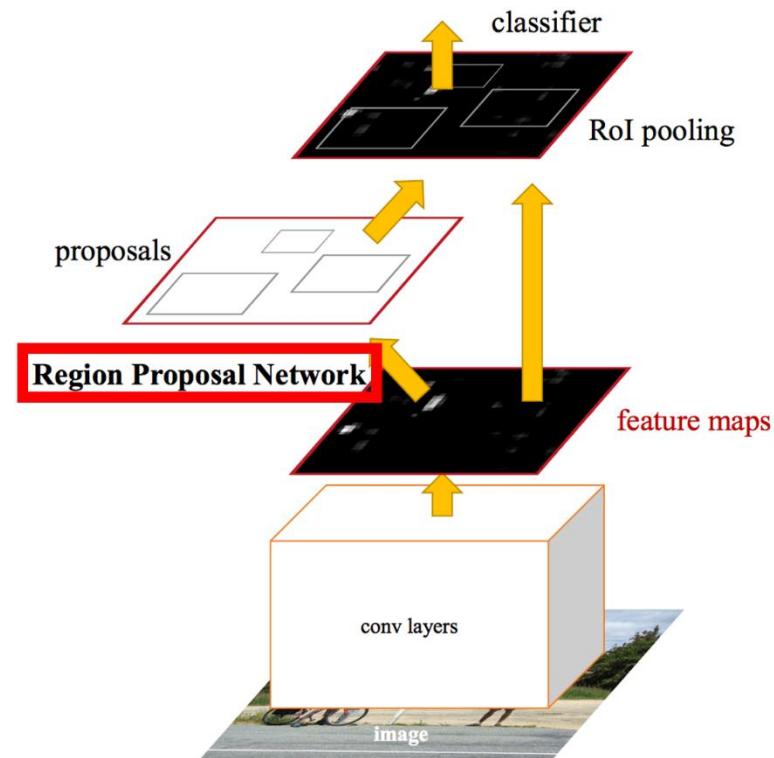
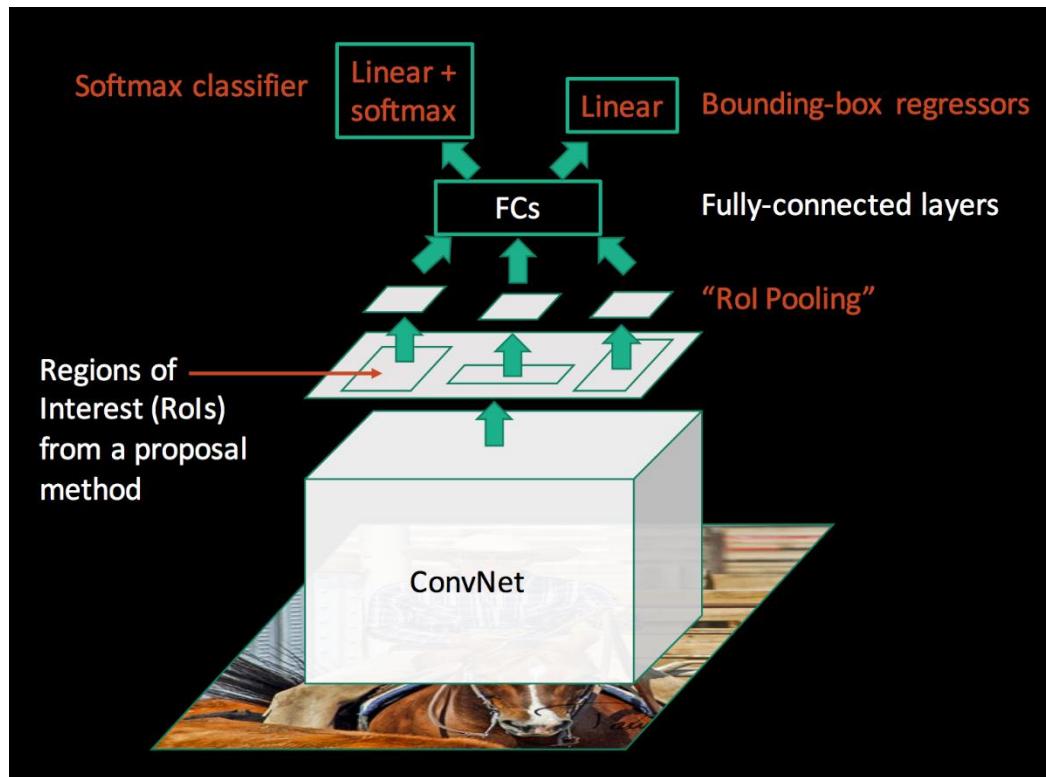
- Laisser le réseau faire les propositions :
RPN (Region Proposal Network)
- Accélère grandement le processus
- RPN s'améliore via backprop des pertes
- Introduction de 9 prototypes de bounding box (**anchor box**) :



Fast R-CNN vs Faster R-CNN

- La différence entre Fast R-CNN et Faster R-CNN est que nous n'utilisons pas de méthode de proposition de région spéciale pour créer des propositions de région.
- Au lieu de cela, nous formons un réseau de proposition de région qui prend les cartes de caractéristiques comme propositions de région d'entrée et de sortie. Ces propositions sont ensuite introduites dans la couche de regroupement RoI dans le Fast R-CNN.

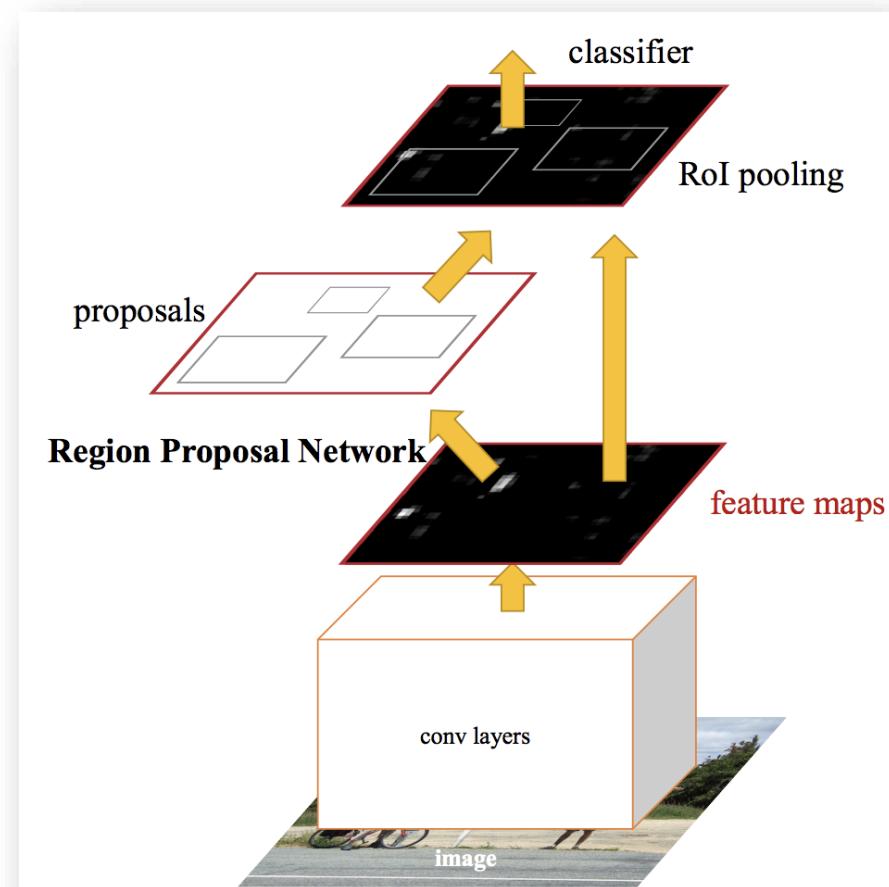
Fast R-CNN vs Faster R-CNN



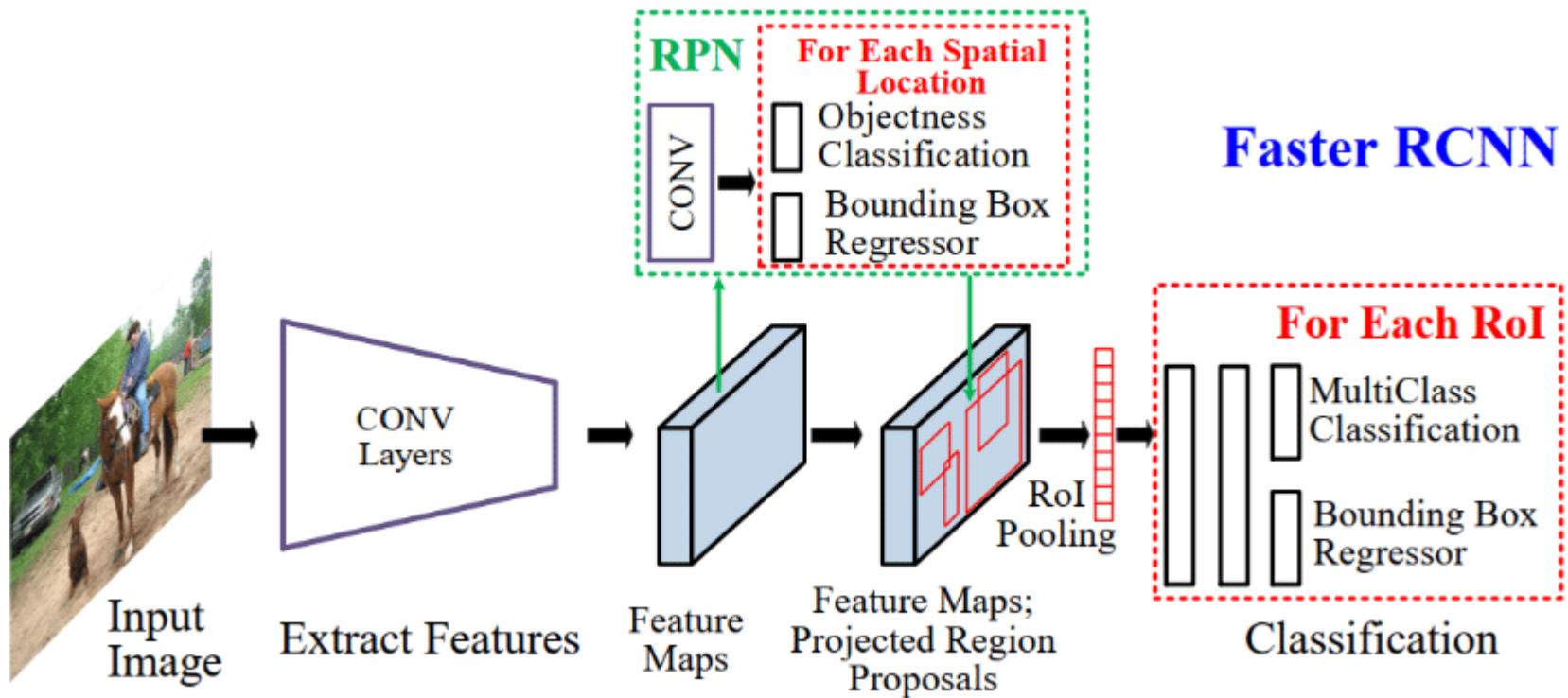
Faster R-CNN

<https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf>

- L'architecture du modèle a été encore améliorée tant pour la vitesse d'entraînement que pour la détection par Shaoqing Ren, et al. à Microsoft Research dans l'article de 2016 intitulé « Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks ».
- **Module 1:** Réseau de proposition de région . Réseau de neurones convolutifs pour proposer des régions et le type d'objet à considérer dans la région.
- **Module 2:** Fast R -CNN . Réseau de neurones convolutifs pour extraire des caractéristiques des régions proposées et générer la boîte englobante et les étiquettes de classe



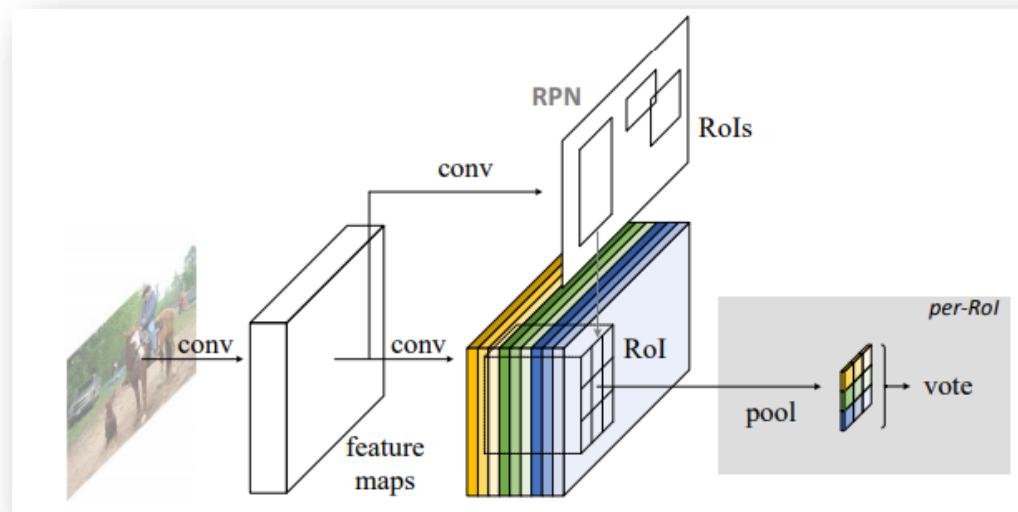
Faster R-CNN



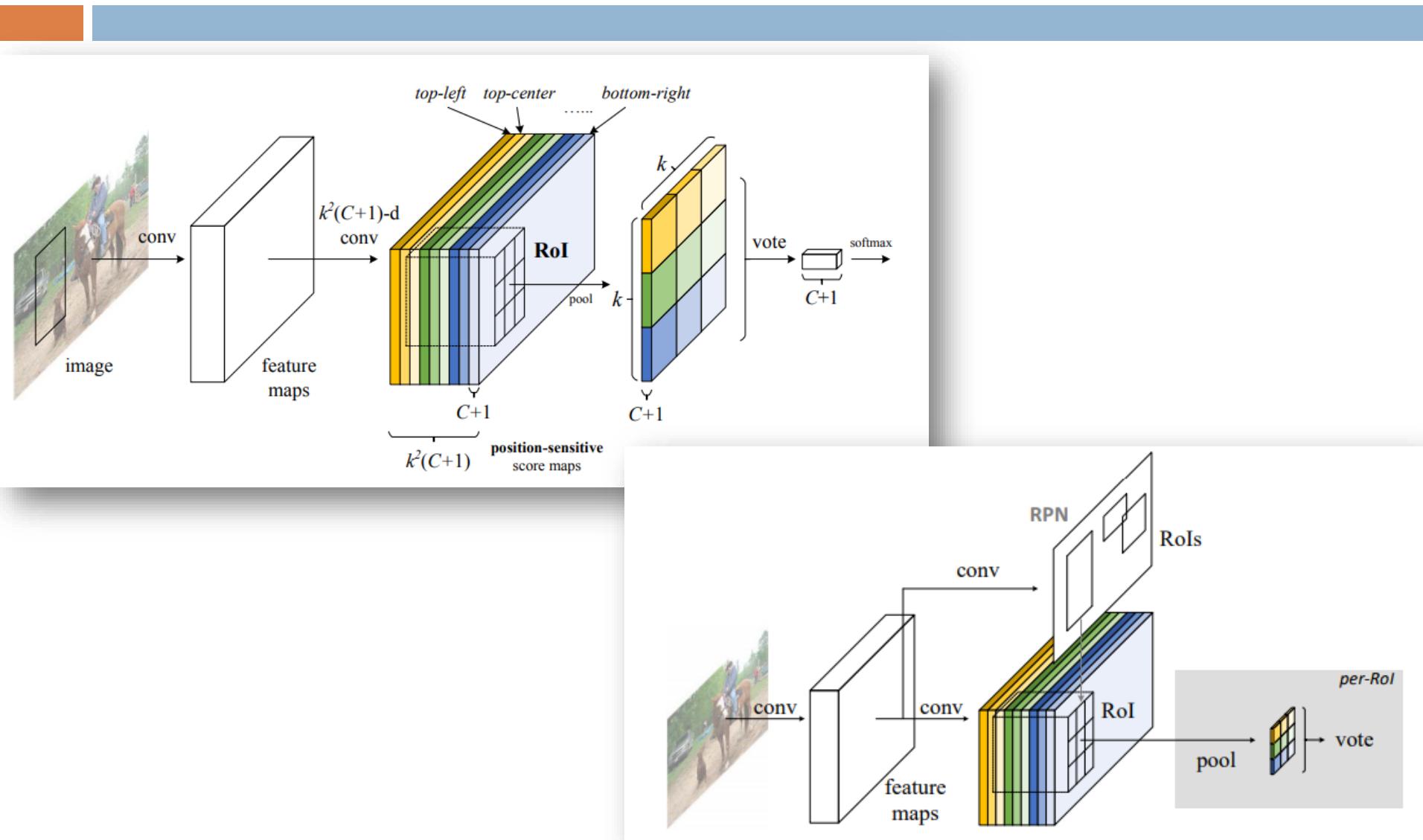
Réseau entièrement convolutif basé sur la région (R-FCN)

<https://arxiv.org/pdf/1605.06409.pdf>

- Les réseaux entièrement convolutifs basés sur la région ou R-FCN est un détecteur basé sur la région pour la détection d'objets.
- Contrairement à d'autres détecteurs basés sur des régions qui appliquent un sous-réseau coûteux par région tel que Fast R-CNN ou Faster R-CNN, ce détecteur basé sur des régions est entièrement convolutif avec presque tous les calculs partagés sur toute l'image.



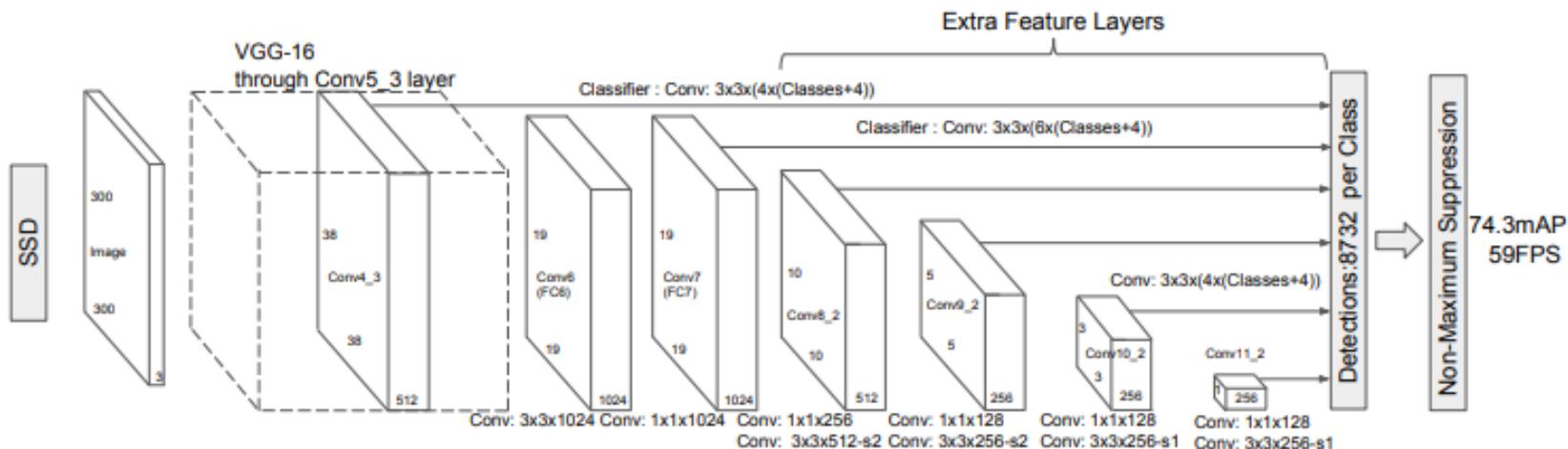
R-FCN



Détecteur à un coup (SSD)

<https://arxiv.org/pdf/1512.02325.pdf>

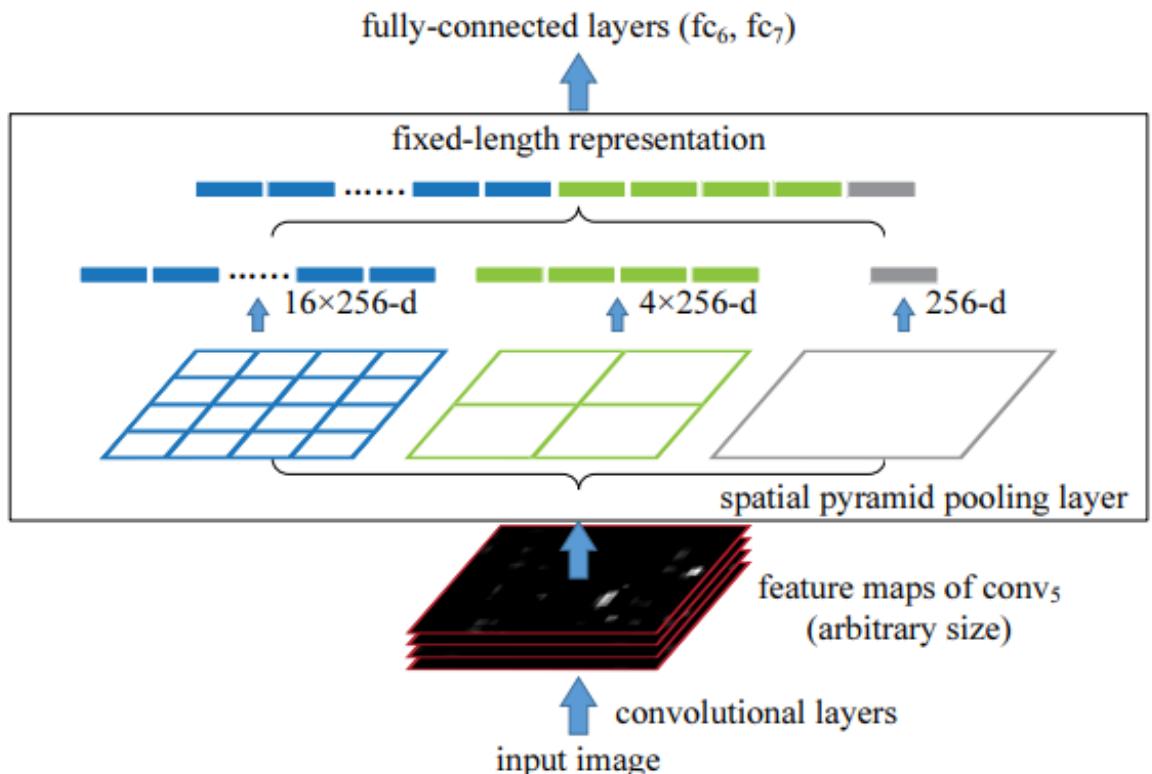
- Single Shot Detector (SSD) est une méthode de détection d'objets dans des images à l'aide d'un seul réseau neuronal profond



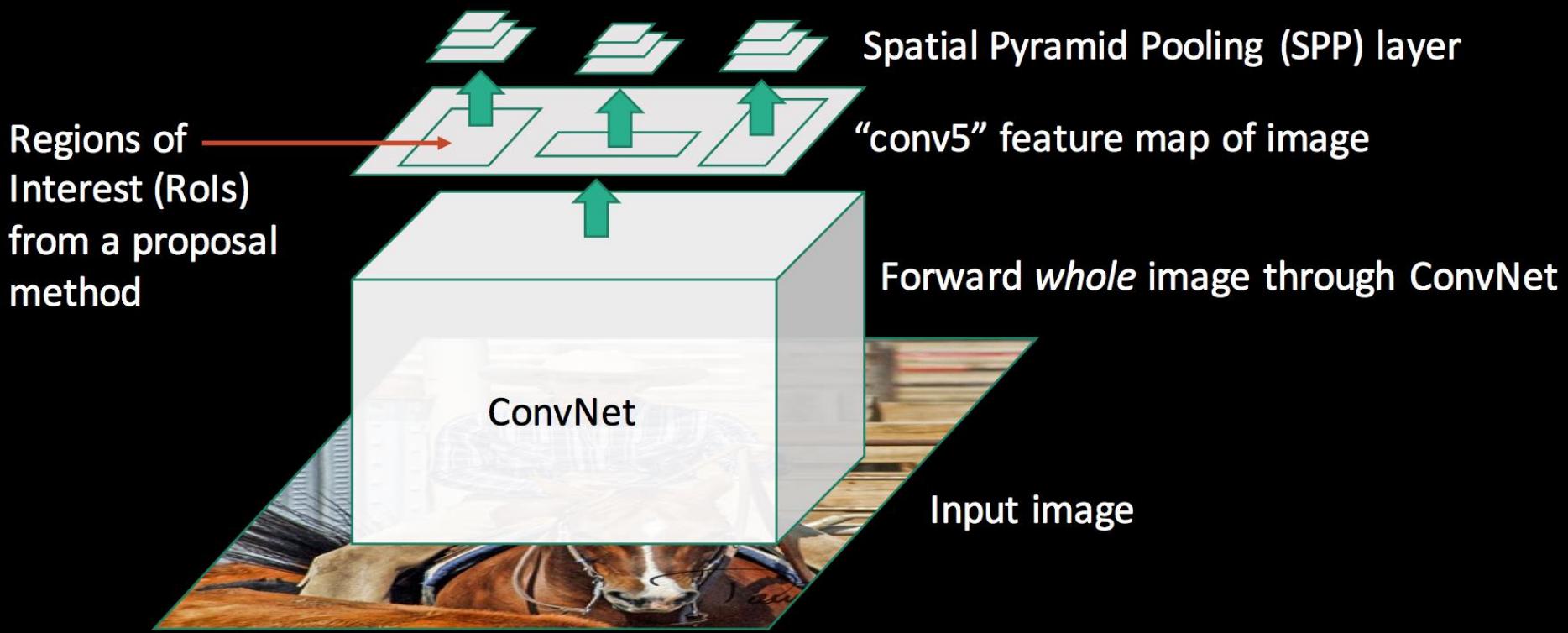
Regroupement de pyramides spatiales (SPP-net)

<https://arxiv.org/pdf/1406.4729.pdf>

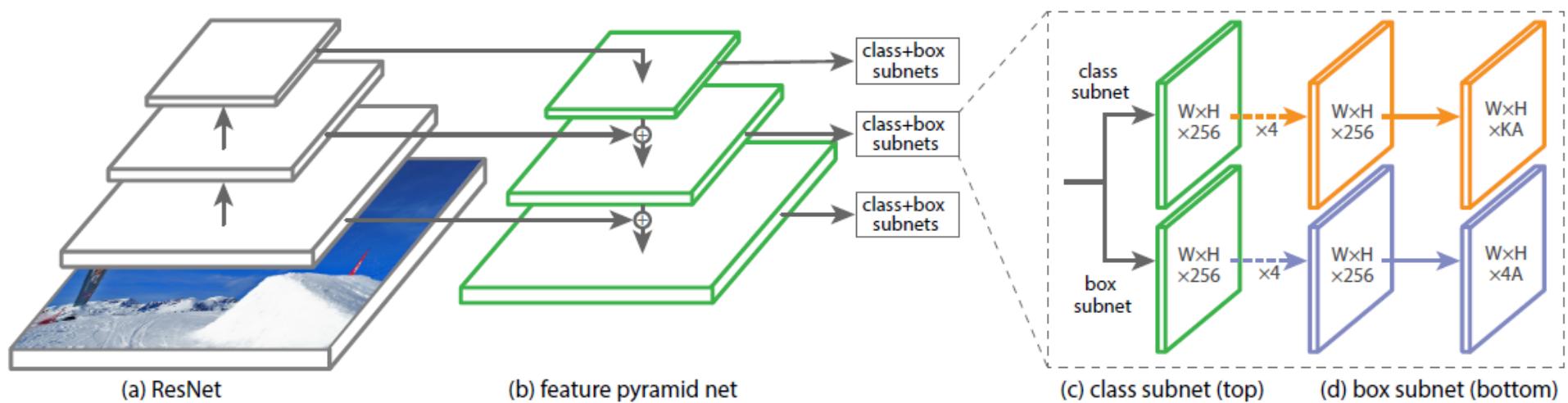
- Le regroupement de pyramides spatiales (SPP-net) est une structure de réseau qui peut générer une représentation de longueur fixe quelle que soit la taille / l'échelle de l'image.



SPP-net



Détecteur RetinaNet



Famille de modèles YOLO

You Only Look Once

- Le modèle YOLO a été décrit pour la première fois par Joseph Redmon et al. dans l'article de 2015 intitulé « *Vous ne regardez qu'une fois: détection unifiée d'objets en temps réel* ».

<https://arxiv.org/abs/1506.02640>

- L'approche implique *un seul réseau neuronal* formé de bout en bout qui prend une photo en entrée et prédit directement les boîtes englobantes et les étiquettes de classe pour chaque boîte englobante

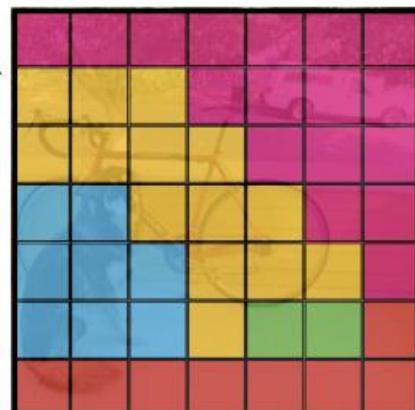
YOLO



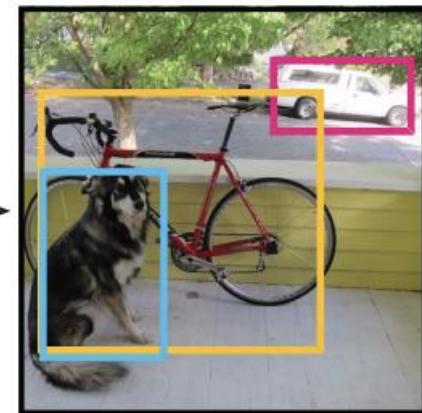
$S \times S$ grid on input



Bounding boxes + confidence

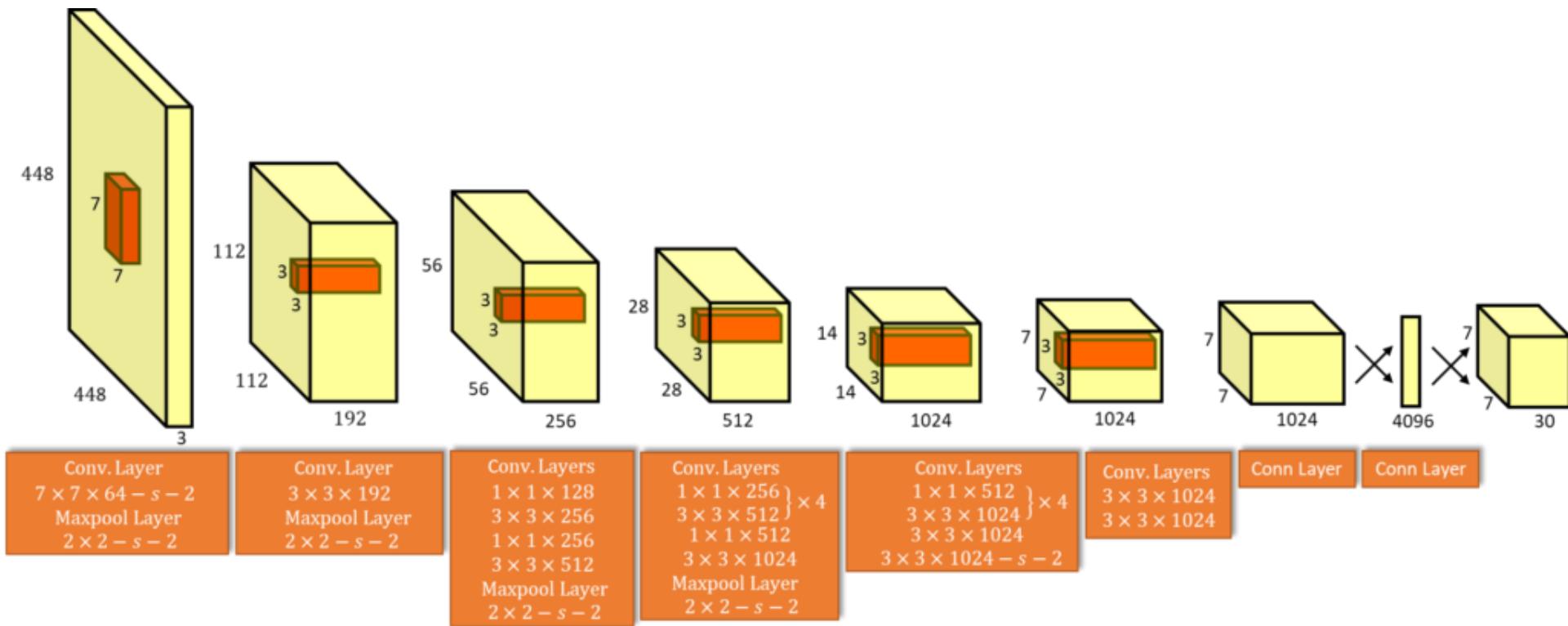


Class probability map



Final detections

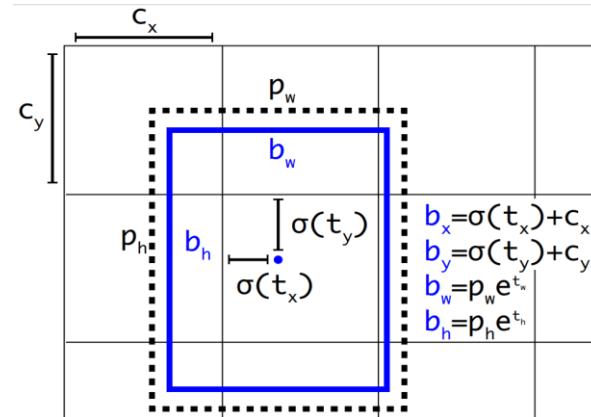
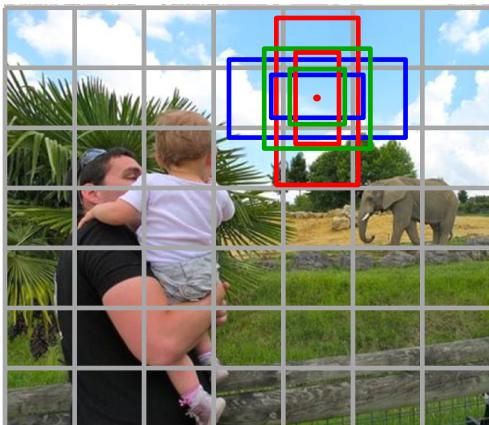
Yolo v1 ARCHITECTURE



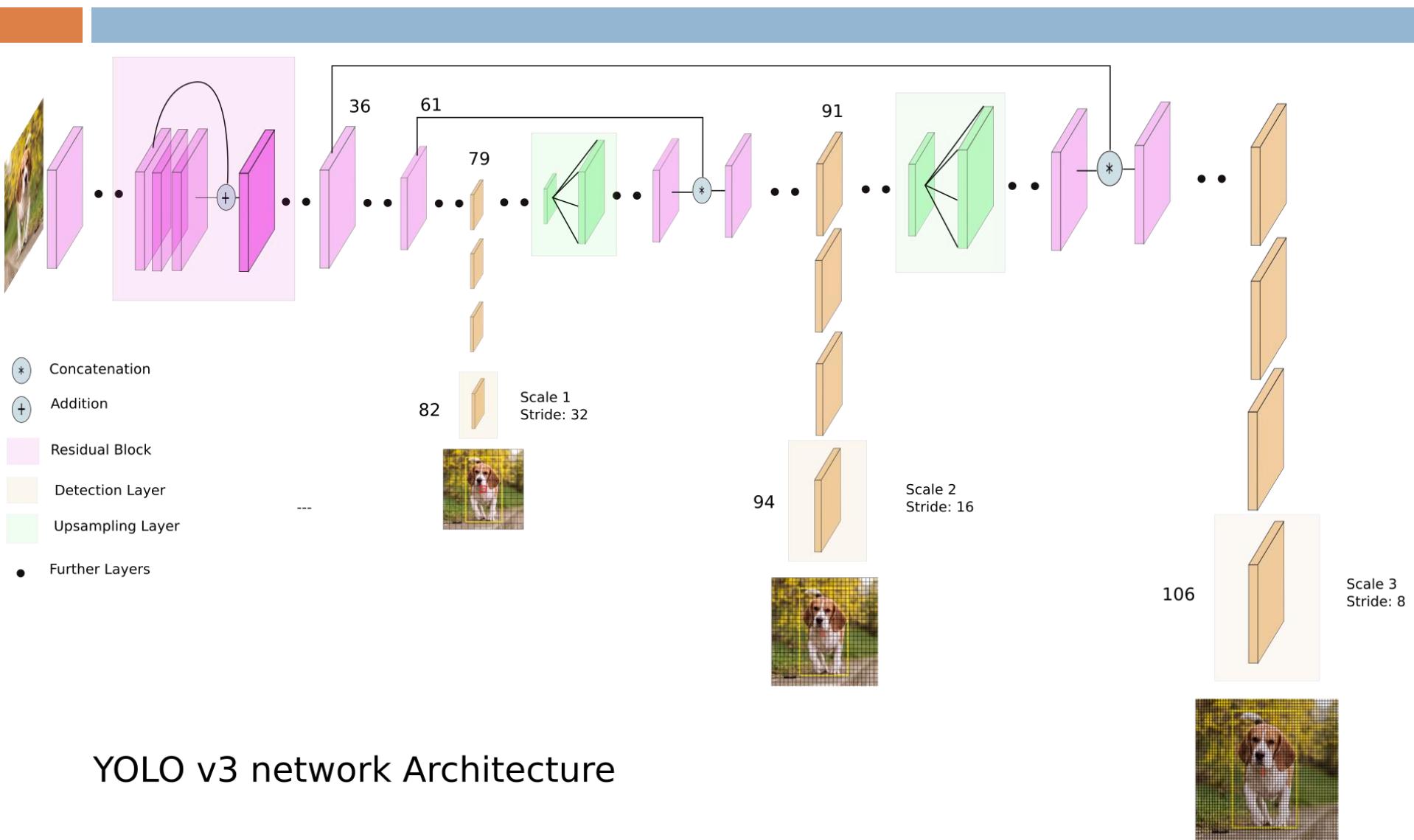
- **YOLOv2**
- **YOLOv3**
- **YOLOv4**

Approche sans proposal (Yolo v3)

- Grille régulière ($7 \times 7 \rightarrow$ feature map)
- Pour chaque position **centrale de la grille**
 - pour chaque anchor box, prédire :
 - classe (incluant la classe background)
 - régression (b_x , b_y , b_w , b_h) sur les paramètres de l'anchor box + confiance (objectness)
- Prédiction obtenues via Fully-convolutional (FCN) d'une 1×1 : très rapide! (20 ms)



Yolo 3



L'architecture de yolov4

