

Réseaux de neurones  
IFT 780

Réseaux à convolution  
Par  
Pierre-Marc Jodoin

1

---

---

---

---

---

---

---

---

kD, 4 Classes, Réseau à 4 couches cachées

Couche d'entrée
Couche cachée 1
Couche cachée 2
Couche cachée 3
Couche cachée 4
Couche de sortie

**Hinge loss**

Couches pleinement connectées (fully-connected layers)

$$y_W(\vec{x}) = W^{[4]} \sigma \left( W^{[3]} \sigma \left( W^{[2]} \sigma \left( W^{[1]} \sigma \left( W^{[0]} \vec{x} \right) \right) \right) \right)$$

2

---

---

---

---

---

---

---

---

kD, 4 Classes, Réseau à 4 couches cachées

Couche d'entrée
Couche cachée 1
Couche cachée 2
Couche cachée 3
Couche cachée 4
Couche de sortie

**Entropie Croisée**  
  
**Softmax**

Couches pleinement connectées (fully-connected layers)

$$y_W(\vec{x}) = \text{softmax} \left( W^{[4]} \sigma \left( W^{[3]} \sigma \left( W^{[2]} \sigma \left( W^{[1]} \sigma \left( W^{[0]} \vec{x} \right) \right) \right) \right) \right)$$

3

---

---

---

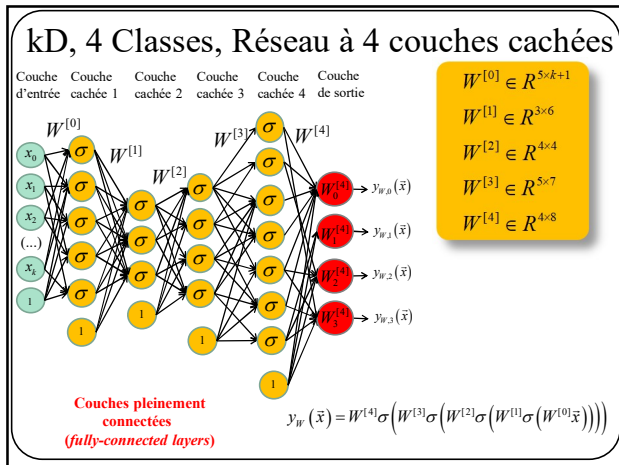
---

---

---

---

---



4

---

---

---

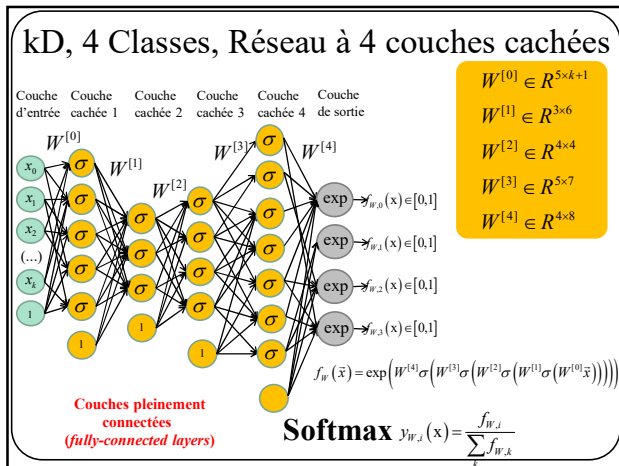
---

---

---

---

---



5

---

---

---

---

---

---

---

---

## Optimisation

Descente de gradient

$$\mathbf{w}^{[k+1]} = \mathbf{w}^{[k]} - \eta^{[k]} \nabla E$$

$\nabla E$  → Gradient de la fonction de coût  
 $\eta$  → Taux d'apprentissage ou "learning rate".

**Descente de gradient stochastique**

Initialiser  $\mathbf{w}$   
 $k=0$   
 FAIRE  $k=k+1$   
 FOR  $n=1$  to  $N$   
 $\mathbf{w} = \mathbf{w} - \eta^{[k]} \nabla E(\vec{x}_n)$

JUSQU'À ce que toutes les données soient bien classées ou  $k = \text{MAX\_ITER}$

**Optimisation par Batch**

Initialiser  $\mathbf{w}$   
 $k=0$   
 FAIRE  $k=k+1$   
 $\mathbf{w} = \mathbf{w} - \eta^{[k]} \sum_i \nabla E(\vec{x}_i)$

JUSQU'À ce que toutes les données soient bien classées ou  $k = \text{MAX\_ITER}$

Parfois  $\eta^{[k]} = \text{cst} / k$

6

---

---

---

---

---

---

---

---

## Les bonnes pratiques

7

---

---

---

---

---

---

---

### Optimisation

#### Descente de gradient

$$\mathbf{w}^{[k+1]} = \mathbf{w}^{[k]} - \eta^{[k]} \nabla E$$

↗ Gradient de la fonction de coût

↘ Taux d'apprentissage ou "learning rate".

Essentiel  
au TP4

#### Optimisation par **mini-batch**

Initialiser  $\mathbf{w}$   
 $k=0$   
FAIRE  $k=k+1$

FAIRE  $n=0$  à  $N$  par sauts de  $MBS / \text{"Mini-batch size"}$

$$\mathbf{w} = \mathbf{w} - \eta^{[k]} \sum_{i=n}^{n+MBS} \nabla E(\vec{x}_i)$$

} Itération

JUSQU'À ce que toutes les données soient bien classées ou  
 $k=\text{MAX\_ITER}$

8

---

---

---

---

---

---

---

### Optimisation

#### Descente de gradient

$$\mathbf{w}^{[k+1]} = \mathbf{w}^{[k]} - \eta^{[k]} \nabla E$$

↗ Gradient de la fonction de coût

↘ Taux d'apprentissage ou "learning rate".

#### Optimisation par **mini-batch**

Initialiser  $\mathbf{w}$   
 $k=0$   
FAIRE  $k=k+1$

FAIRE  $n=0$  à  $N$  par sauts de  $MBS / \text{"Mini-batch size"}$

$$\mathbf{w} = \mathbf{w} - \eta^{[k]} \sum_{i=n}^{n+MBS} \nabla E(\vec{x}_i)$$

} Epoch

JUSQU'À ce que toutes les données sont bien classées ou  
 $k=\text{MAX\_ITER}$

9

---

---

---

---

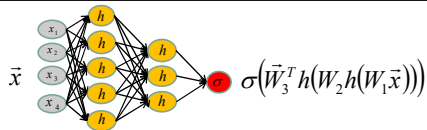
---

---

---

Mini-batch = **vectorisation** de la  
propagation avant et de la rétro-  
propagation

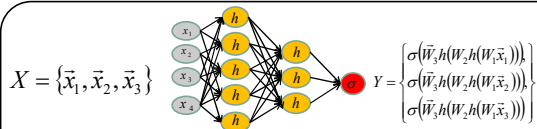
10



Propagation avant pour un réseau à 2 couches cachées (7 étapes)

Étape 2	$\vec{x}$	$\in \mathbb{R}^4$
	$W_1 \vec{x}$	$\in \mathbb{R}^5$
	$h(W_1 \vec{x})$	$\in \mathbb{R}^5$
	$W_2 h(W_1 \vec{x})$	$\in \mathbb{R}^3$
	$h(W_2 h(W_1 \vec{x}))$	$\in \mathbb{R}^3$
Étape 6	$\vec{W}_3^T (h(W_2 h(W_1 \vec{x})))$	$\in \mathbb{R}$
	$\sigma(\vec{W}_3^T (h(W_2 h(W_1 \vec{x}))))$	$\in \mathbb{R}$

11



Propagation avant pour un réseau à 2 couches cachées (7 étapes)

POUR i allant de 0 à 2

$\vec{x} = X[i]$	$\in \mathbb{R}^4$
$W_1 \vec{x}$	$\in \mathbb{R}^5$
$h(W_1 \vec{x})$	$\in \mathbb{R}^5$
$W_2 h(W_1 \vec{x})$	$\in \mathbb{R}^3$
$h(W_2 h(W_1 \vec{x}))$	$\in \mathbb{R}^3$
$\vec{W}_3^T (h(W_2 h(W_1 \vec{x})))$	$\in \mathbb{R}$
$Y[i] = \sigma(\vec{W}_3^T (h(W_2 h(W_1 \vec{x}))))$	$\in \mathbb{R}$

Solution  
naïve et peu  
efficace

TP3

12

### Solution

Il est plus efficace d'effectuer **UNE multiplication matricielle** que **PLUSIEURS multiplications matrice-vecteur** (exemple de la 2<sup>e</sup> étape, batch de 3)

$$\begin{aligned}
 W_1 \vec{x}_1 &= \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{pmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} \\
 W_1 \vec{x}_2 &= \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{pmatrix} \begin{bmatrix} d \\ e \\ f \\ g \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} \\
 W_1 \vec{x}_3 &= \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{pmatrix} \begin{bmatrix} h \\ i \\ j \\ k \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}
 \end{aligned}$$

**TROIS** multi. matrice-vecteur

13

---

---

---

---

---

---

---

---

### Solution

Il est plus efficace d'effectuer **UNE multiplication matricielle** que **PLUSIEURS matrice-vecteur** (exemple de la 2<sup>e</sup> étape, batch de 3)

$$W_1 X = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{pmatrix} \begin{bmatrix} a & d & h \\ b & e & i \\ c & f & j \\ d & g & k \end{bmatrix} = \begin{bmatrix} u_1 & v_1 & z_1 \\ u_2 & v_2 & z_2 \\ u_3 & v_3 & z_3 \\ u_4 & v_4 & z_4 \\ u_5 & v_5 & z_5 \end{bmatrix}$$

**UNE** multiplication matricielle

14

---

---

---

---

---

---

---

---

### Solution

Il est plus efficace d'effectuer **UNE multiplication matricielle** que **PLUSIEURS produits scalaires** (exemple de la 6<sup>e</sup> étape, batch de 3)

$$\begin{aligned}
 \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} &= w_1 a + w_2 b + w_3 c \\
 \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} d \\ e \\ f \end{pmatrix} &= w_1 d + w_2 e + w_3 f \\
 \begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix} \begin{pmatrix} g \\ h \\ i \end{pmatrix} &= w_1 g + w_2 h + w_3 i
 \end{aligned}$$

**TROIS** produits scalaires

$\vec{w}_3^T (h(w_2 h(w_1 \vec{x})))$

15

---

---

---

---

---

---

---

---

## Solution

Il est plus efficace d'effectuer **UNE multiplication matricielle** que PLUSIEURS produits scalaires (**exemple de la 6<sup>e</sup> étape, batch de 3**)

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{pmatrix} a & d & g \\ b & e & h \\ c & f & i \end{pmatrix} = \begin{pmatrix} w_1a + w_2b + w_3c \\ w_1d + w_2e + w_3f \\ w_1g + w_2h + w_3i \end{pmatrix} = Y$$

UNE  
multiplication  
matricielle

16

---

---

---

---

---

---

---

---

## Conclusion

100% du temps, on combine ensemble les données dans des **mini-batch** de 2 à 32 données.

17

---

---

---

---

---

---

---

---

## Les réseaux à convolution

18

---

---

---

---

---

---

---

---

[illegible]

---

---

---

---

---

---

# Comment classifier des images?

The diagram illustrates a neural network architecture for image classification. On the left, a 28 x 28 pixel grayscale image of the digit '8' is shown, with the text '28 x 28' and '784 pixels' below it. This input is connected to a layer of 784 nodes, labeled 'Pixel 1' through 'Pixel 784'. These nodes are fully connected to a hidden layer of 10 nodes, labeled 'Node 1' through 'Node 10'. The hidden layer nodes are then fully connected to an output layer of 10 nodes, labeled '0' through '9', representing the possible digits. The connections between layers are shown as a dense web of lines, indicating a fully connected architecture.

[https://ml4k.github.io/ml4k neural\\_network/](https://ml4k.github.io/ml4k neural_network/)

---

---

---

---

---

---

[illegible]

---

---

---

---

---

---

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

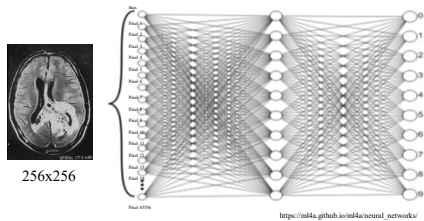


Image médicale (IRM de cerveau)

22

---

---

---

---

---

---

---

---

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

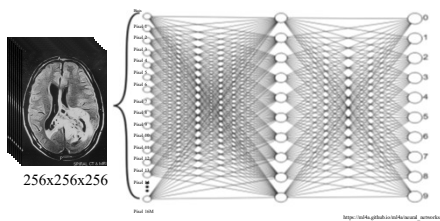


Image médicale 3D (IRM de cerveau)

23

---

---

---

---

---

---

---

---

Comment réduire le nombre de connections?



24

24

---

---

---

---

---

---

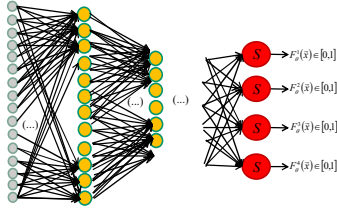
---

---



## Comment réduire le nombre de connections?

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 => **22,200 paramètres** dans la couche d'entrée!!

25

---

---

---

---

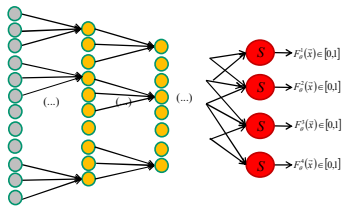
---

---

---

---

## Solution : connexions partielles



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

26

---

---

---

---

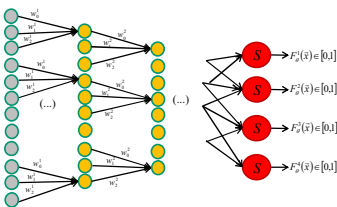
---

---

---

---

## 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!**

27

---

---

---

---

---

---

---

---

# Convolution et couche convolutionnelle **1D**

28

---

---

---

---

---

---

---

---

## Exemple 1D de la convolution

$$(f * W)(v) = \sum_{u=-\infty}^{\infty} f(u)W(v-u)$$

(signal d'entrée)

$f(u)$

$[10 \ 20 \ 30 \ 40 \ 50]$

(filtre)

$W(u)$

$[.1 \ .2 \ .3]$

(filtre)

$W(-6)$

$[.3 \ .2 \ .1]$

$(f * W)(1)$

$[10 \ 20 \ 30 \ 40 \ 50]$   
x x x  
 $[.3 \ .2 \ .1]$

$3+4+3$

$[4 \ \ ] \ \ ]$

$(f * W)(2)$

$[10 \ 20 \ 30 \ 40 \ 50]$   
x x x  
 $[.3 \ .2 \ .1]$

$6-6+4$

$[4 \ 4 \ \ ]$

$(f * W)(3)$

$[10 \ 20 \ 30 \ 40 \ 50]$   
x x x  
 $[.3 \ .2 \ .1]$

$-9+8+5$

$[4 \ 4 \ -6]$

29

29

---

---

---

---

---

---

---

---

# En gros

**convolution** = **produit scalaire** + **translation**

30

---

---

---

---

---

---

---

---

**La convolution des réseaux de neurones = corrélation**

$$(f * W)(v) = \sum_{u=-\infty}^{\infty} f(u)W(v+u)$$

(signal d'entrée)  $f(u)$   
 $\begin{bmatrix} 10 & 20 & -30 & 40 & -50 \end{bmatrix}$

(filtre)  $W(u)$   
 $\begin{bmatrix} .1 & .2 & .3 \end{bmatrix}$

(filtre)  $W(+u)$   
 $\begin{bmatrix} .1 & .2 & .3 \end{bmatrix}$

**(f\*W)(1)**

$$\begin{bmatrix} 10 & 20 & -30 & 40 & -50 \\ \times & \times & \times & & \\ .1 & .2 & .3 & & \end{bmatrix}$$

1+4-9  
 $\downarrow$   
 $\begin{bmatrix} -4 & & \end{bmatrix}$

**(f\*W)(2)**

$$\begin{bmatrix} 10 & 20 & -30 & 40 & -50 \\ & \times & \times & \times & \\ .1 & .2 & .3 & & \end{bmatrix}$$

2-6+12  
 $\downarrow$   
 $\begin{bmatrix} -4 & 8 & \end{bmatrix}$

**(f\*W)(3)**

$$\begin{bmatrix} 10 & 20 & -30 & 40 & -50 \\ & & \times & \times & \times \\ .1 & .2 & .3 & & \end{bmatrix}$$

-3+8-15  
 $\downarrow$   
 $\begin{bmatrix} -4 & 8 & -10 \end{bmatrix}$

31

---

---

---

---

---

---

---

---

31

**L'opération de la page précédente est équivalente à**

32

---

---

---

---

---

---

---

---

32

**L'opération de la page précédente est équivalente à**

Fonction d'activation (ex. ReLU)

33

---

---

---

---

---

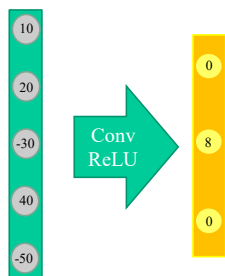
---

---

---

33

### Représentation graphique courante (simple)



34

---

---

---

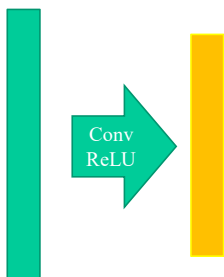
---

---

---

---

### Représentation graphique courante (encore plus simple)



35

---

---

---

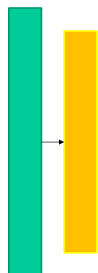
---

---

---

---

### Représentation graphique courante (vraiment ultra simple)



36

---

---

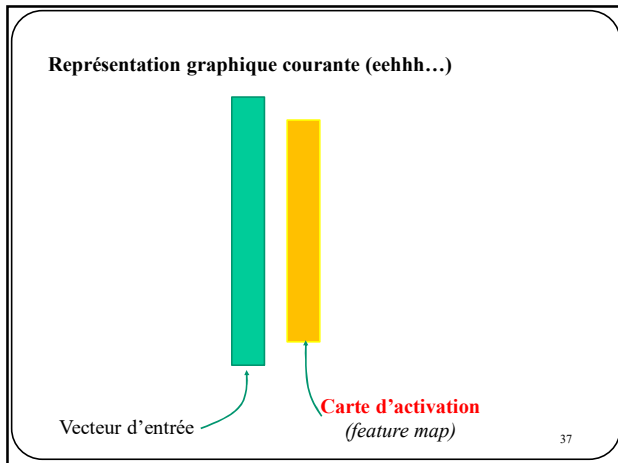
---

---

---

---

---



37

---

---

---

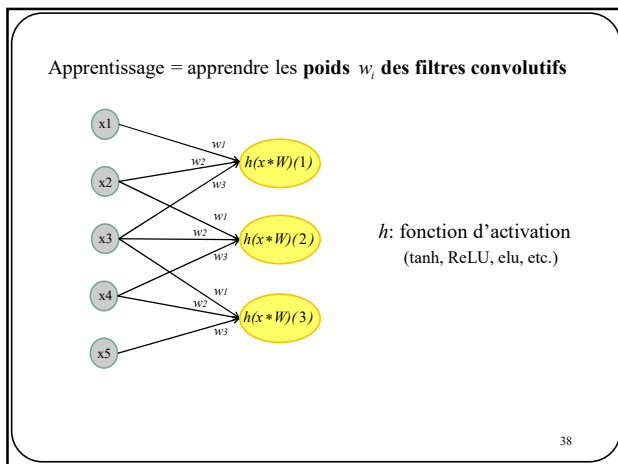
---

---

---

---

---



38

---

---

---

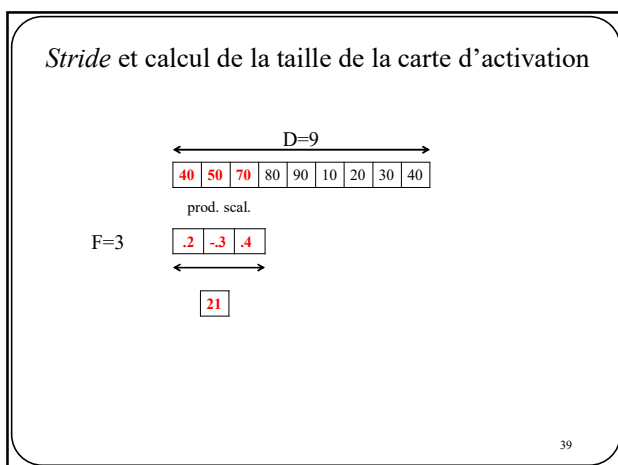
---

---

---

---

---



39

---

---

---

---

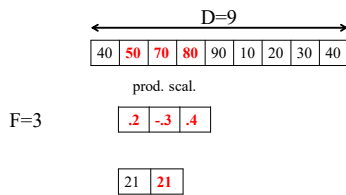
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Stride = 1

40

40

---

---

---

---

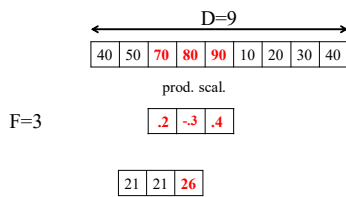
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



41

41

---

---

---

---

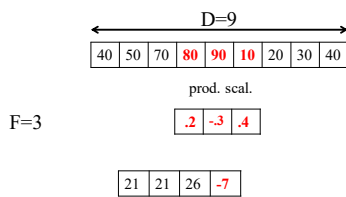
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



42

42

---

---

---

---

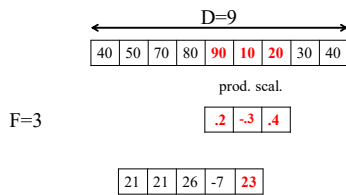
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



43

---

---

---

---

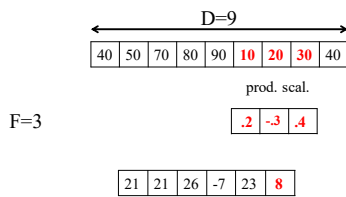
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



44

---

---

---

---

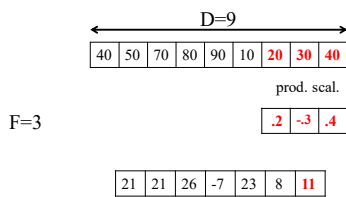
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Taille de la carte d'activation = 7

45

---

---

---

---

---

---

---

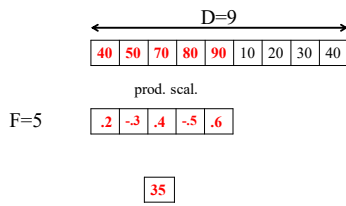
---

43

44

45

### Stride et calcul de la taille de la carte d'activation



46

---

---

---

---

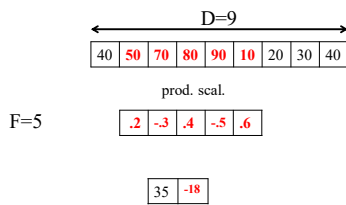
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Stride = 1

47

---

---

---

---

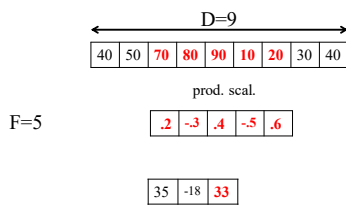
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



48

---

---

---

---

---

---

---

---

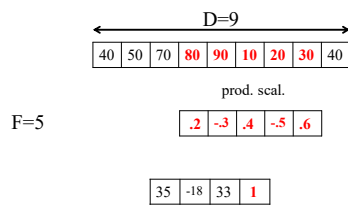
46

47

48



### Stride et calcul de la taille de la carte d'activation



49

---

---

---

---

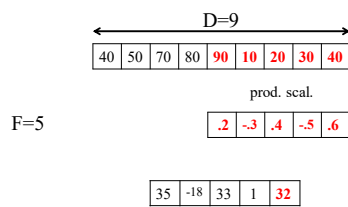
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Taille de la carte d'activation = **5**

50

---

---

---

---

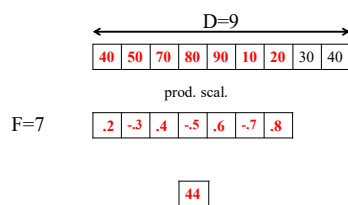
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



51

---

---

---

---

---

---

---

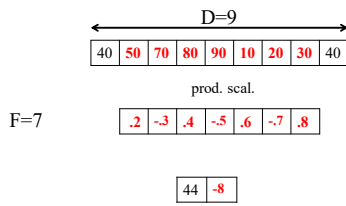
---

49

50

51

### Stride et calcul de la taille de la carte d'activation



Stride = 1

52

---

---

---

---

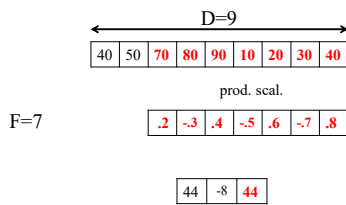
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Taille de la carte d'activation = 3

53

---

---

---

---

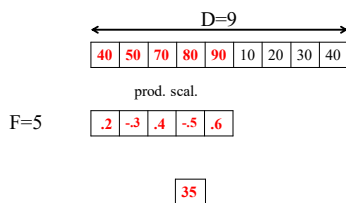
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



54

---

---

---

---

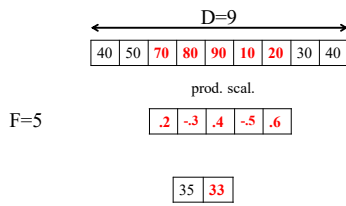
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Stride = 2

55

---

---

---

---

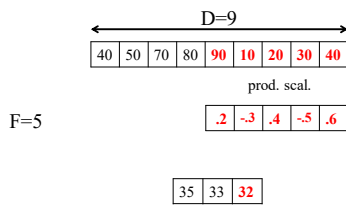
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Taille de la carte d'activation = 3

56

---

---

---

---

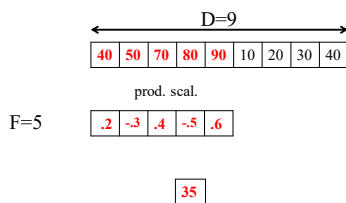
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



57

---

---

---

---

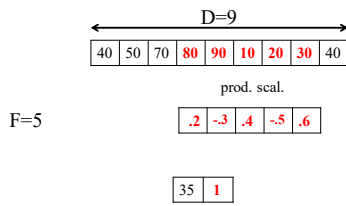
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



Stride = 3

58

---

---

---

---

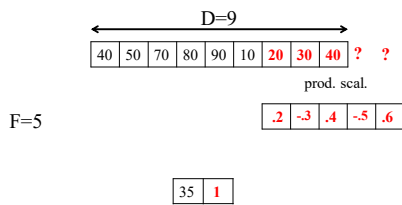
---

---

---

---

### Stride et calcul de la taille de la carte d'activation



**ERREUR! Combinaison D-F-S invalide**

59

---

---

---

---

---

---

---

---

### Stride et calcul de la taille de la carte d'activation

Taille de la carte d'activation =  **$(D-F)/S+1$**



60

---

---

---

---

---

---

---

---

58

59

60

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

Comment gérer les bords?

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

$f(u)$   $(f^*W)(u)$

$\begin{bmatrix} 0 & 10 & 20 & 30 & 40 & 50 & 0 \end{bmatrix}$   $\begin{bmatrix} 8 & -4 & 8 & 10 & -6 \end{bmatrix}$

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

$f(u)$   $(f^*W)(u)$

$\begin{bmatrix} 20 & 10 & 20 & 30 & 40 & 50 & 40 \end{bmatrix}$   $\begin{bmatrix} 10 & -4 & 8 & 10 & 2 \end{bmatrix}$

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

$f(u)$   $(f^*W)(u)$

$\begin{bmatrix} 10 & 10 & 20 & 30 & 40 & 50 & 50 \end{bmatrix}$   $\begin{bmatrix} 9 & -4 & 8 & 10 & -2 \end{bmatrix}$

61

---

---

---

---

---

---

---

---

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

Comment gérer les bords?

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

$f(u)$   $(f^*W)(u)$

$\begin{bmatrix} 0 & 10 & 20 & 30 & 40 & 50 & 0 \end{bmatrix}$   $\begin{bmatrix} 8 & -4 & 8 & 10 & -6 \end{bmatrix}$

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

$f(u)$   $(f^*W)(u)$

$\begin{bmatrix} 20 & 10 & 20 & 30 & 40 & 50 & 40 \end{bmatrix}$   $\begin{bmatrix} 10 & -4 & 8 & 10 & 2 \end{bmatrix}$

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

$f(u)$   $(f^*W)(u)$

$\begin{bmatrix} 10 & 10 & 20 & 30 & 40 & 50 & 50 \end{bmatrix}$   $\begin{bmatrix} 9 & -4 & 8 & 10 & -2 \end{bmatrix}$

De loin l'option la plus utilisée

62

---

---

---

---

---

---

---

---

**Couche convolutionnelle sans « padding »**

signal d'entrée

Carte d'activation (feature map)

**Couche convolutionnelle avec « padding »**

signal d'entrée

Carte d'activation (feature map)

63

---

---

---

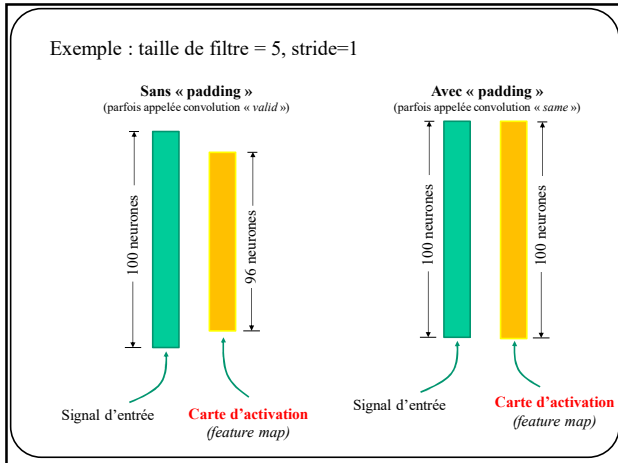
---

---

---

---

---



64

---

---

---

---

---

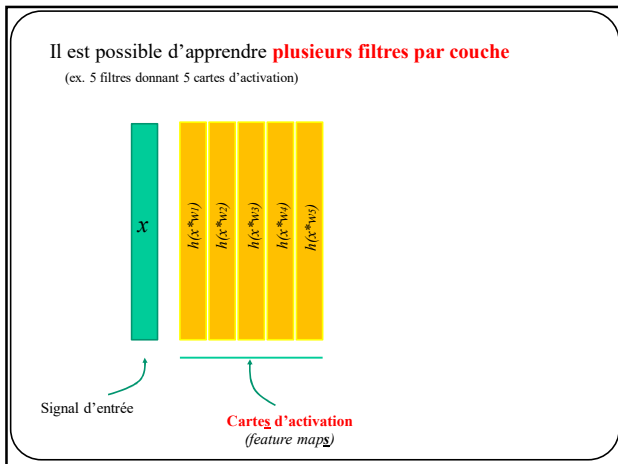
---

---

---

---

---



65

---

---

---

---

---

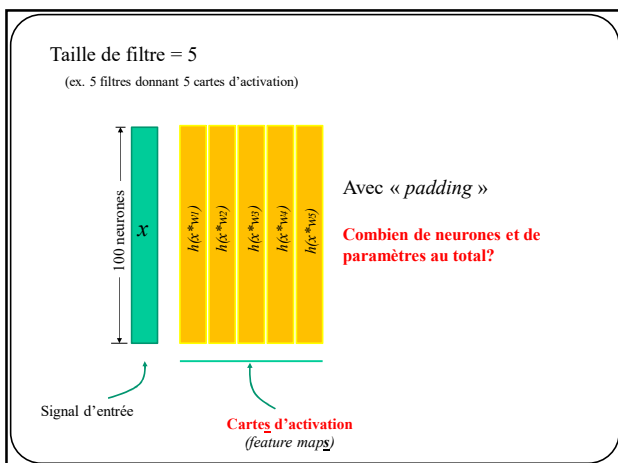
---

---

---

---

---



66

---

---

---

---

---

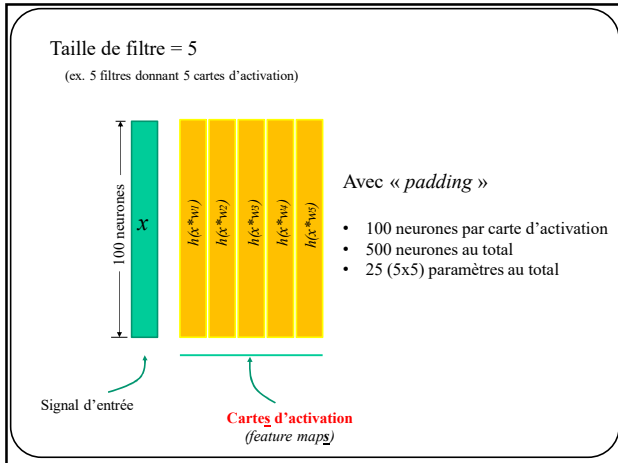
---

---

---

---

---



67

---

---

---

---

---

---

---

---

---

---

# Convolution et couche convolutionnelle **2D**

68

---

---

---

---

---

---

---

---

---

---

## Filtage 2D (sans flip de filtre)

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

$x(i, j)$

$W(u, v)$

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

$$(x * W)(i, j) = w_1 x(i-1, j-1) + w_2 x(i, j-1) + w_3 x(i+1, j-1) + w_4 x(i-1, j) + w_5 x(i, j) + w_6 x(i+1, j) + w_7 x(i-1, j+1) + w_8 x(i, j+1) + w_9 x(i+1, j+1)$$

69

---

---

---

---

---

---

---

---

---

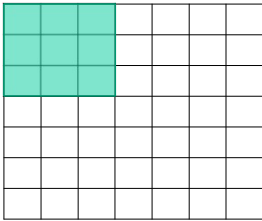
---

Convolution 2D

Filtre = 3x3

Stride = 1

7



7

---

---

---

---

---

---

---

---

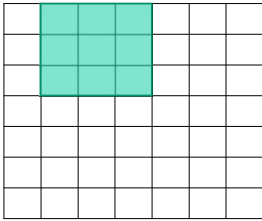
70

Convolution 2D

Filtre = 3x3

Stride = 1

7



7

---

---

---

---

---

---

---

---

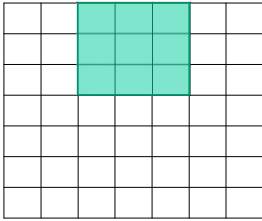
71

Convolution 2D

Filtre = 3x3

Stride = 1

7



7

---

---

---

---

---

---

---

---

72



### Convolution 2D

Filtre = 3x3  
Stride = 1

73

---

---

---

---

---

---

---

---

### Convolution 2D

Filtre = 3x3  
Stride = 1

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

74

---

---

---

---

---

---

---

---

### Convolution 2D

Filtre = 3x3  
Stride = 2

75

---

---

---

---

---

---

---

---

### Convolution 2D

Filtre = 3x3  
Stride = 2

76

---

---

---

---

---

---

---

---

### Convolution 2D

Filtre = 3x3  
Stride = 2

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

77

---

---

---

---

---

---

---

---

### Convolution 2D

Filtre = 3x3  
Stride = 3

78

---

---

---

---

---

---

---

---

### Convolution 2D

Filtre = 3x3  
Stride = 3

79

---

---

---

---

---

---

---

---

### Convolution 2D

Filtre = 3x3  
Stride = 2

**Combinaison D-F-S invalide!**

80

---

---

---

---

---

---

---

---

### Convolution 2D

Taille de la carte d'activation :  
 **$(D1-F1)/S+1 \times (D2-F2)/S+1$**

81

---

---

---

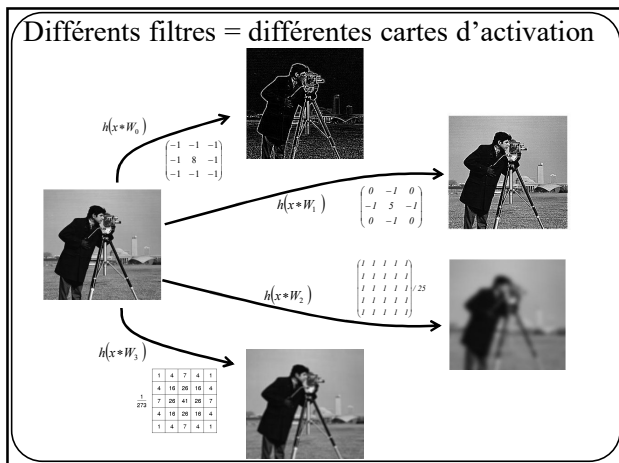
---

---

---

---

---



82

---

---

---

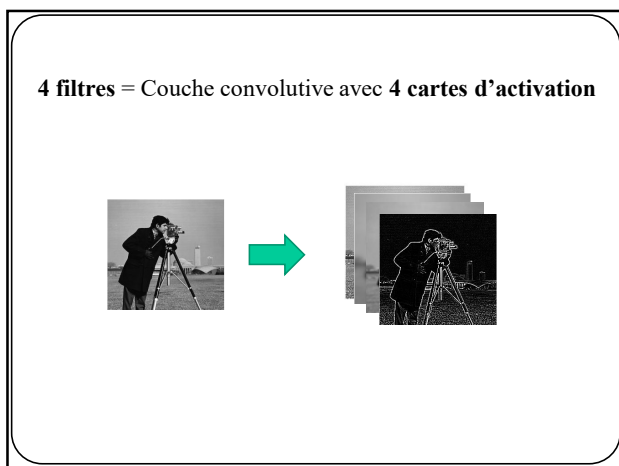
---

---

---

---

---



83

---

---

---

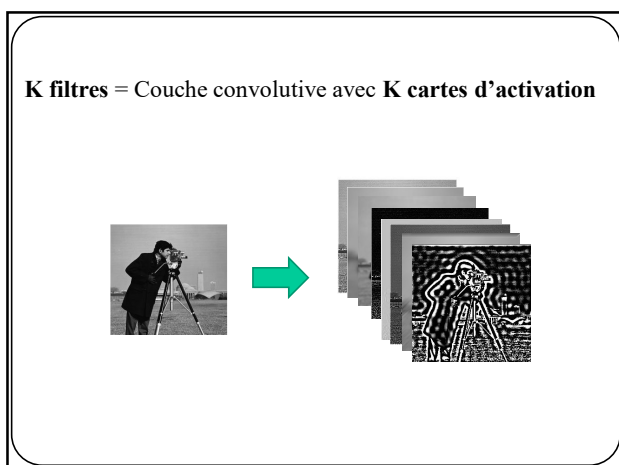
---

---

---

---

---



84

---

---

---

---

---

---

---

---

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

85

---

---

---

---

---

---

---

---

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

Image: 100x100x1  
Filtre : 5x5x1  
Stride : 1

86

---

---

---

---

---

---

---

---

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

Image: 100x100x1  
Filtre : 5x5x1  
Stride : 1

87

---

---

---

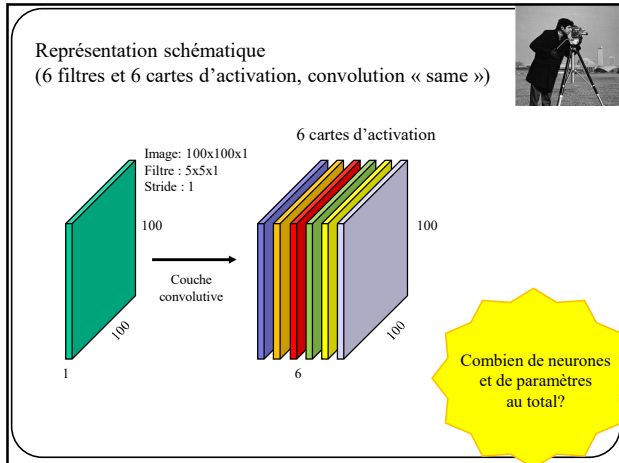
---

---

---

---

---



88

---

---

---

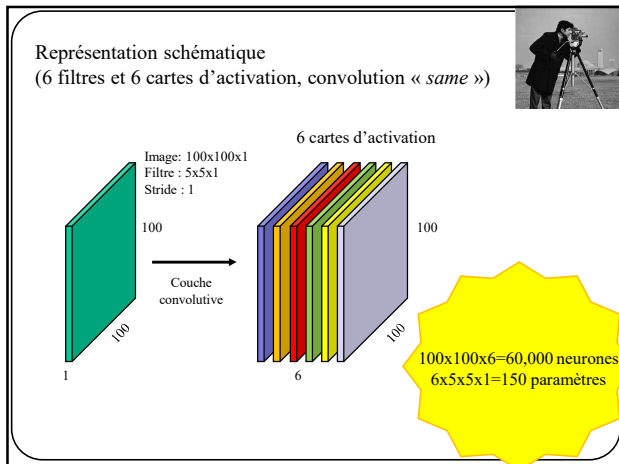
---

---

---

---

---



89

---

---

---

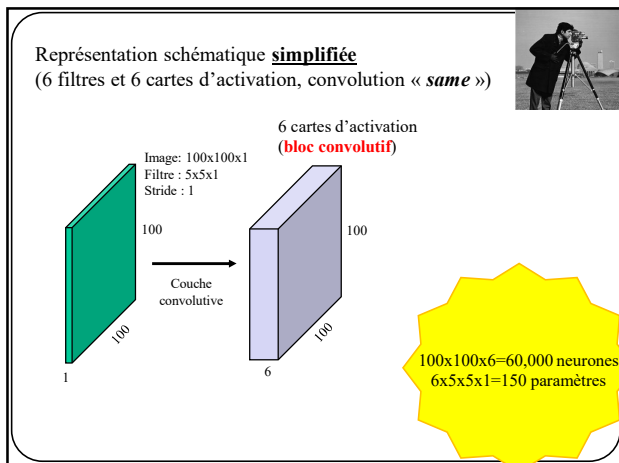
---

---

---

---

---



90

---

---

---

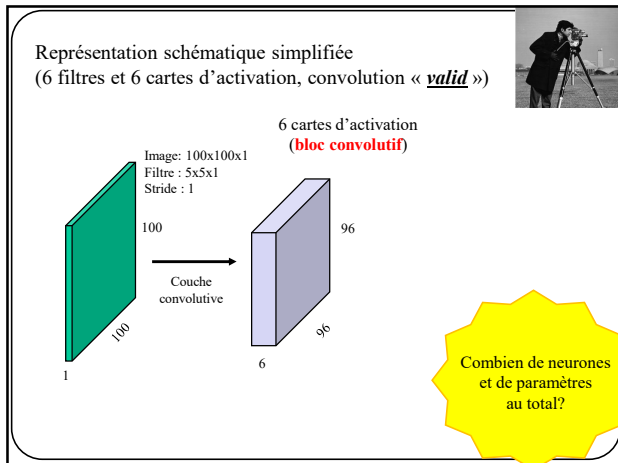
---

---

---

---

---



91

---

---

---

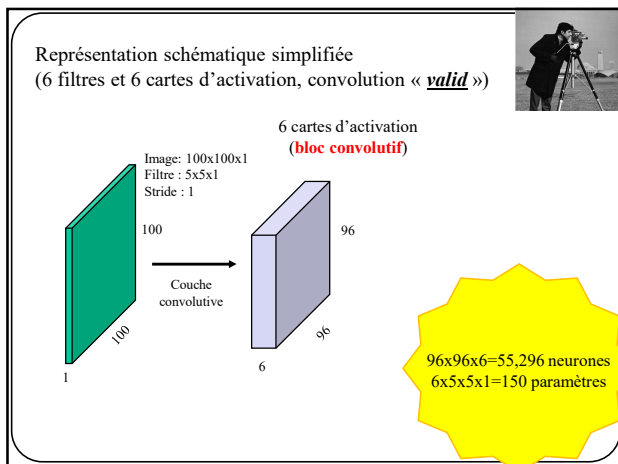
---

---

---

---

---



92

---

---

---

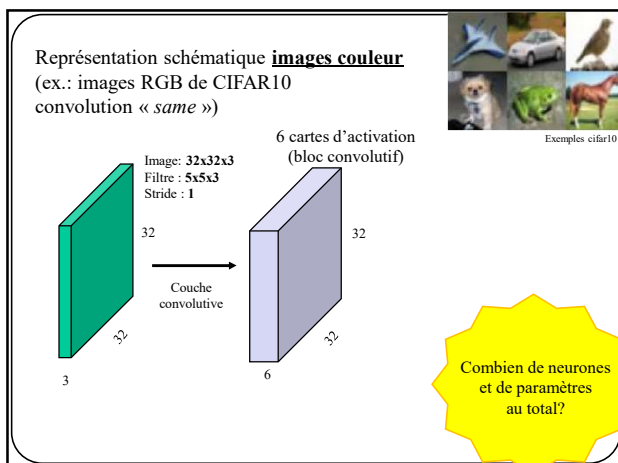
---

---

---

---

---



93

---

---

---

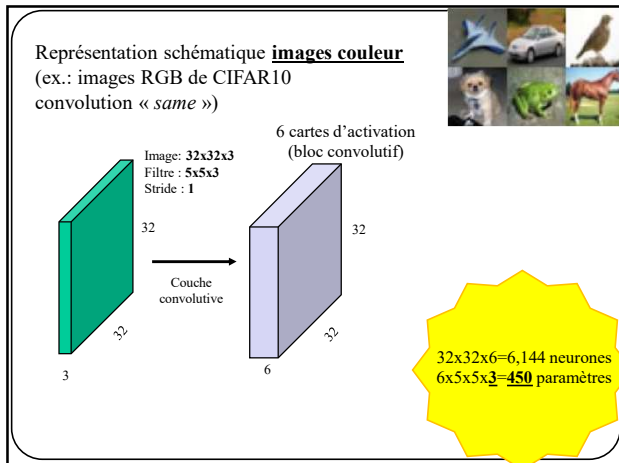
---

---

---

---

---



94

---

---

---

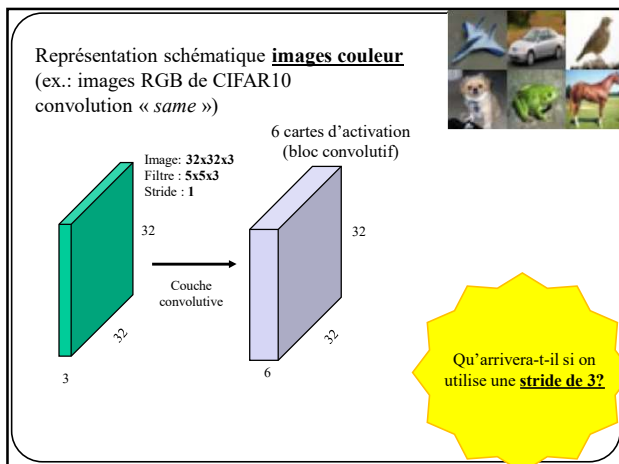
---

---

---

---

---



95

---

---

---

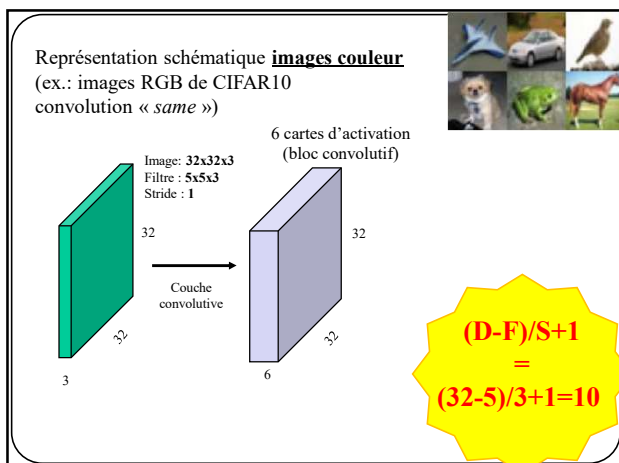
---

---

---

---

---



96

---

---

---

---

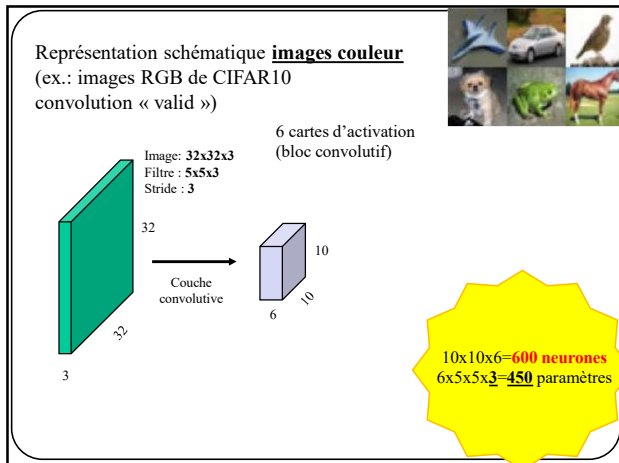
---

---

---

---





97

---

---

---

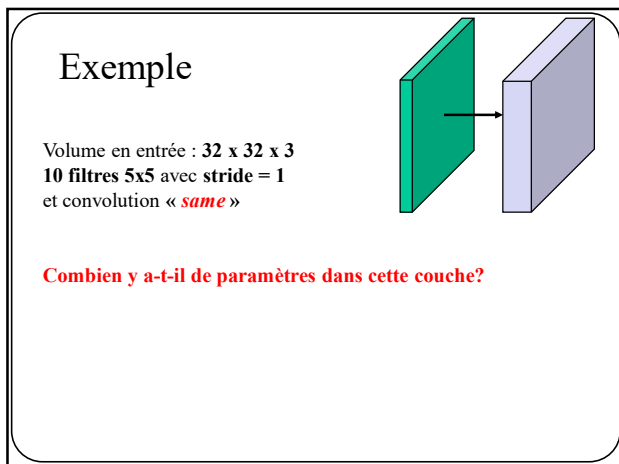
---

---

---

---

---



98

---

---

---

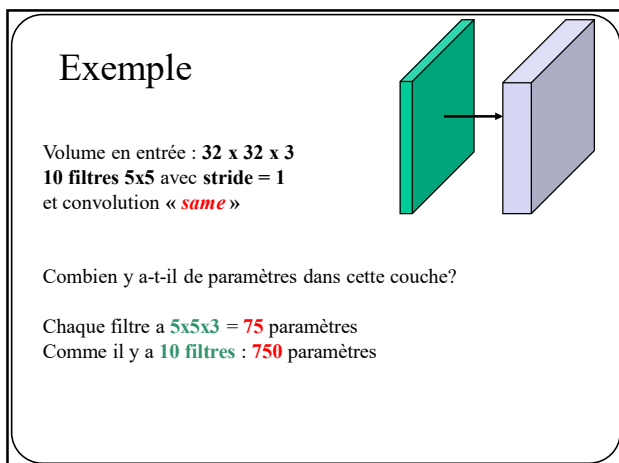
---

---

---

---

---



99

---

---

---

---

---

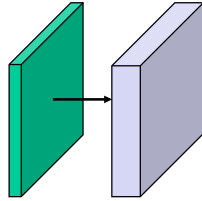
---

---

---

## Exemple

Volume en entrée :  $32 \times 32 \times 3$   
10 filtres  $5 \times 5$  avec  $\text{stride} = 1$   
et convolution « *same* ».



Combien y a-t-il de paramètres dans cette couche si on ajoute un biais?

Chaque filtre a  $5 \times 5 \times 3 + 1 = 76$  paramètres (+1 pour le biais)  
Comme il y a 10 filtres : 760 paramètres

100

---

---

---

---

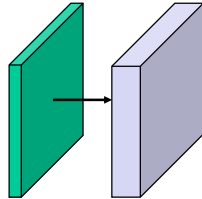
---

---

---

## Exemple

Volume en entrée :  $32 \times 32 \times 3$   
10 filtres  $5 \times 5$  avec  $\text{stride} = 1$   
et convolution « *valid* »



Combien de paramètres dans cette couche?

101

---

---

---

---

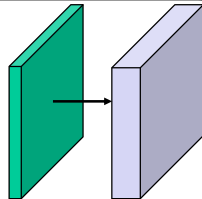
---

---

---

## Exemple

Volume en entrée :  $32 \times 32 \times 3$   
10 filtres  $5 \times 5$  avec  $\text{stride} = 1$   
et convolution « *valid* »



Combien de paramètres dans cette couche?

Même chose, cela ne change pas la conformité des filtres

102

---

---

---

---

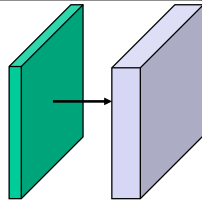
---

---

---

## Exemple

Volume en entrée :  $32 \times 32 \times 3$   
10 filtres  $5 \times 5$  avec  $\text{stride} = 1$   
et convolution « *valid* »



Combien de **neurones** dans les cartes d'activations?

---

---

---

---

---

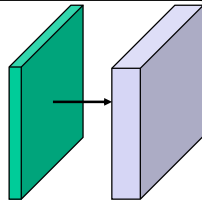
---

---

103

## Exemple

Volume en entrée :  $32 \times 32 \times 3$   
10 filtres  $5 \times 5$  avec  $\text{stride} = 1$   
et convolution « *valid* »



Combien de **neurones** dans les cartes d'activations?

$$(32-5+1) \times (32-5+1) \times 10 = 7,840$$

---

---

---

---

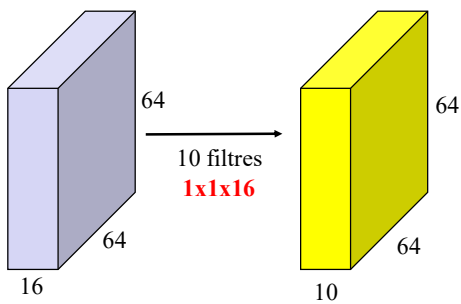
---

---

---

104

## Des filtres $1 \times 1$ ? Oui ça marche



---

---

---

---

---

---

---

105

## Exemple simple d'un filtre 1x1



$$\begin{bmatrix} 1 & 1 & 1 \\ 3 & 3 & 3 \end{bmatrix}$$

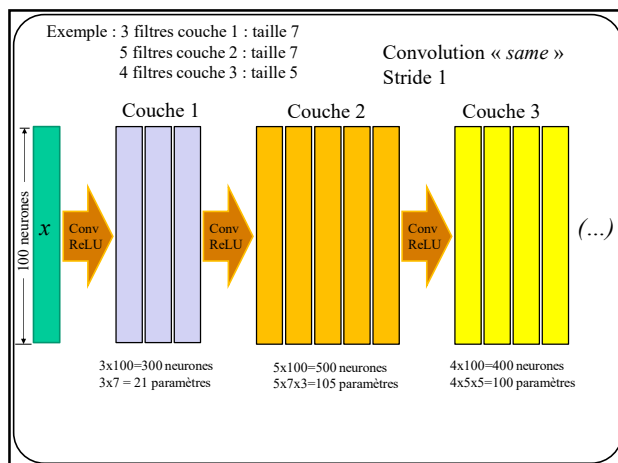


Filtre moyennant les canaux **rouge**, **vert**, **bleu** d'une image couleur.  
Résultat, une image en niveau de gris.

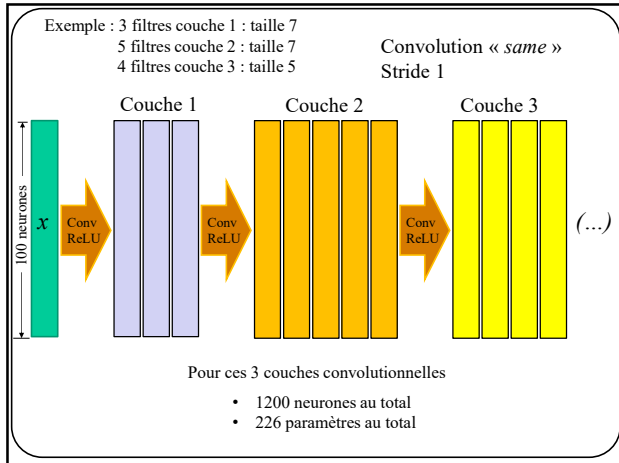
106

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

107



108



109

---

---

---

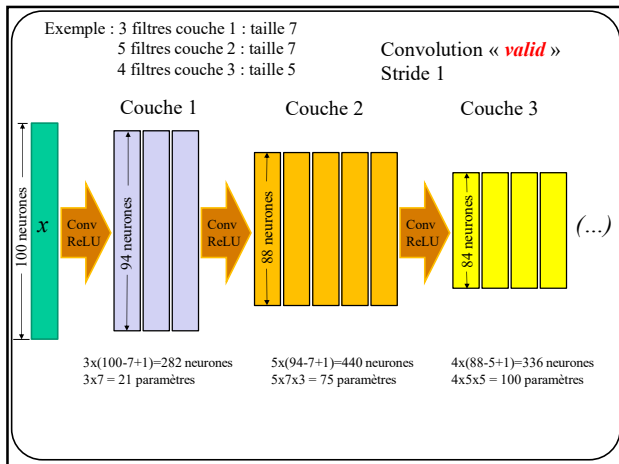
---

---

---

---

---



110

---

---

---

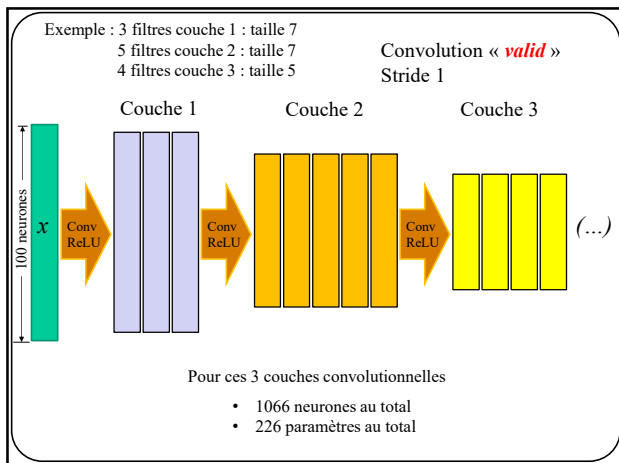
---

---

---

---

---



111

---

---

---

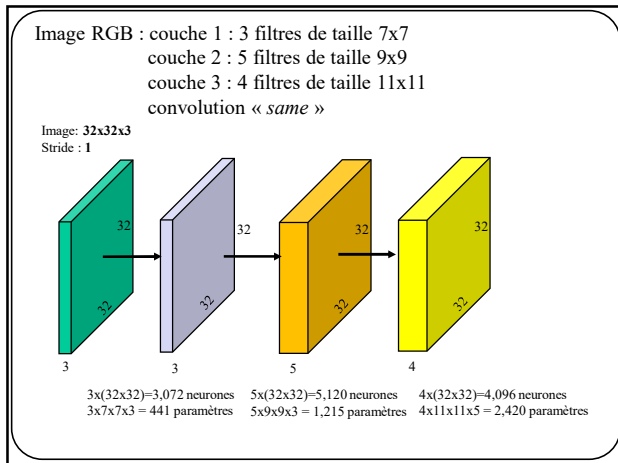
---

---

---

---

---



112

---

---

---

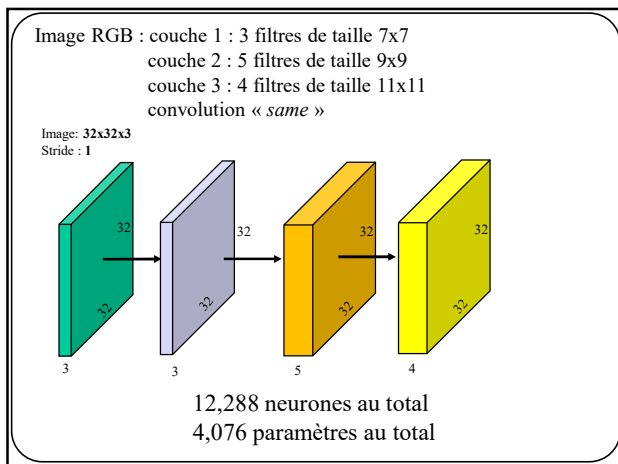
---

---

---

---

---



113

---

---

---

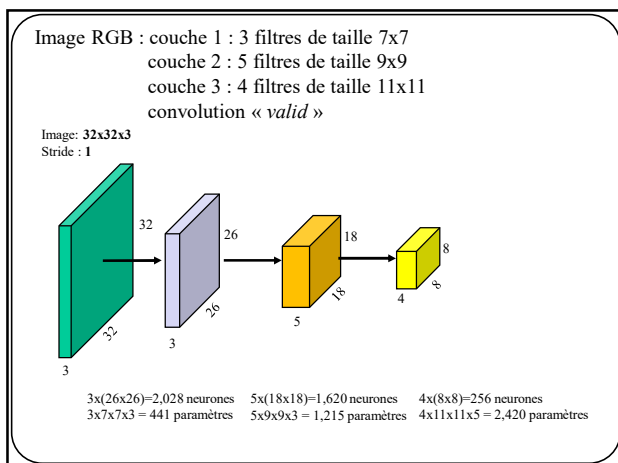
---

---

---

---

---



114

---

---

---

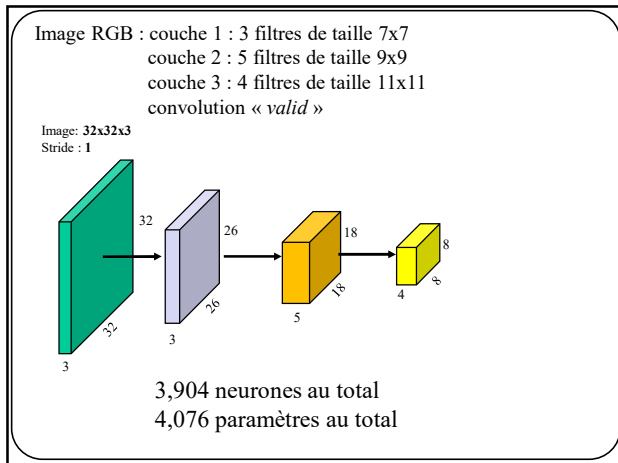
---

---

---

---

---



115

---

---

---

---

---

---

---

---

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

116

---

---

---

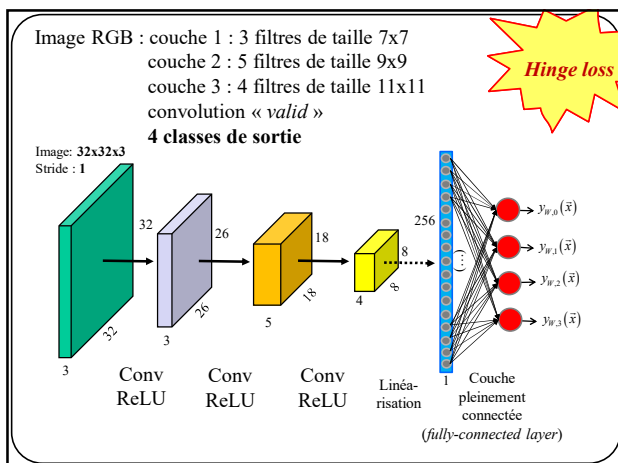
---

---

---

---

---



117

---

---

---

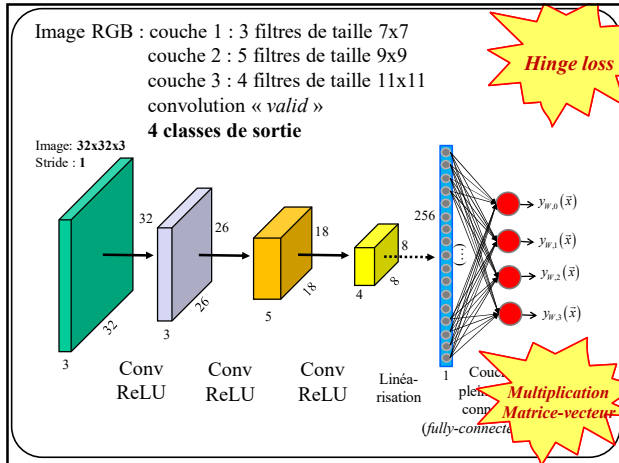
---

---

---

---

---



118

---

---

---

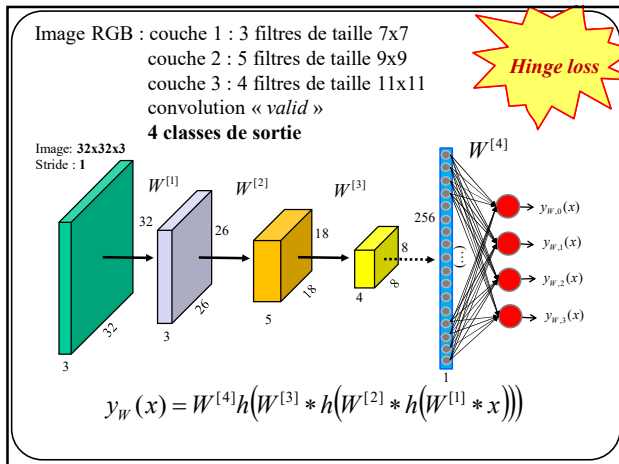
---

---

---

---

---



119

---

---

---

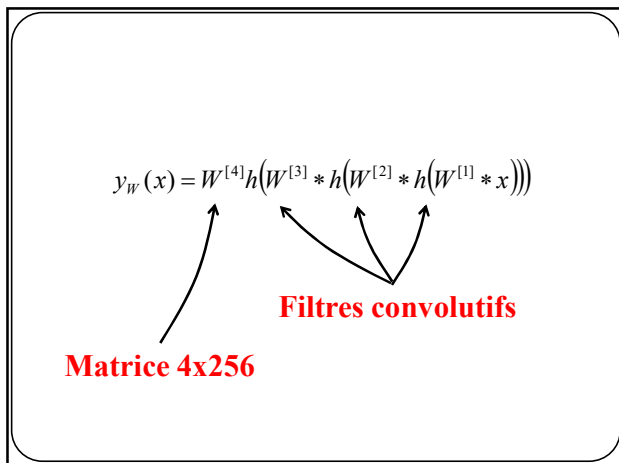
---

---

---

---

---



120

---

---

---

---

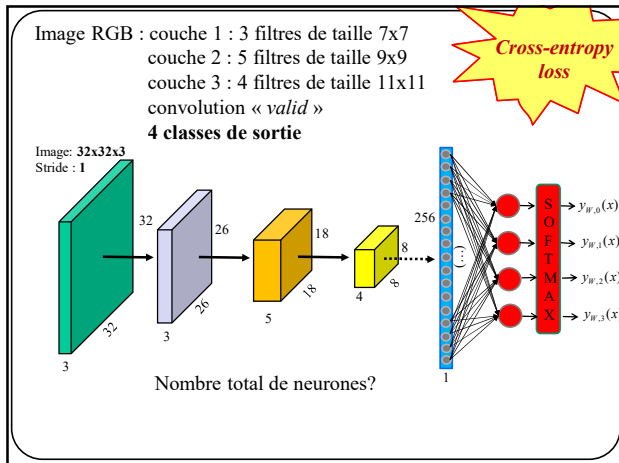
---

---

---

---





121

---

---

---

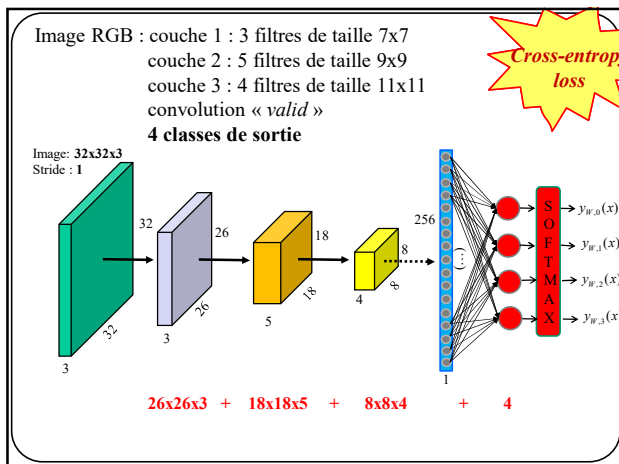
---

---

---

---

---



122

---

---

---

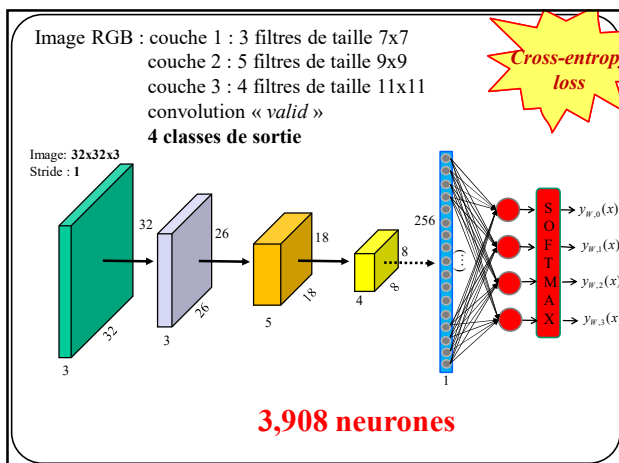
---

---

---

---

---



123

---

---

---

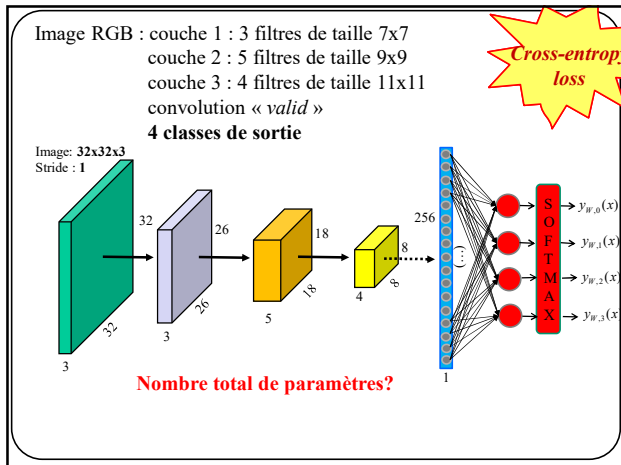
---

---

---

---

---



124

---

---

---

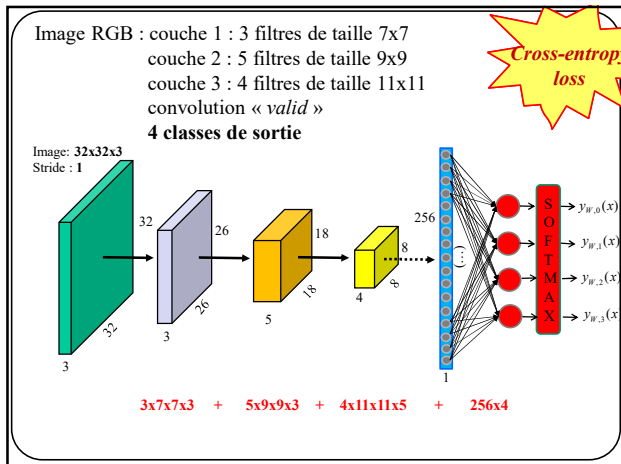
---

---

---

---

---



125

---

---

---

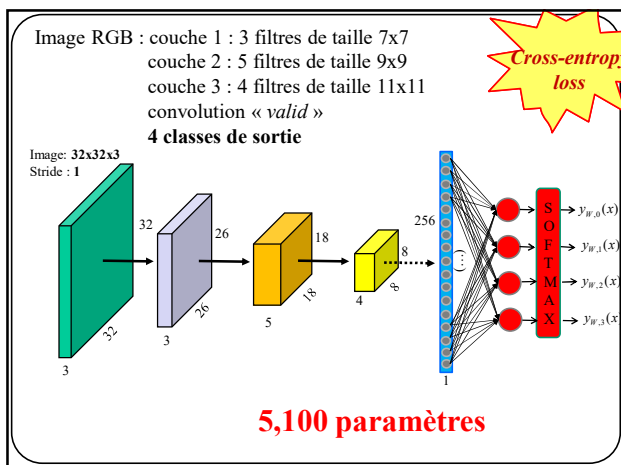
---

---

---

---

---



126

---

---

---

---

---

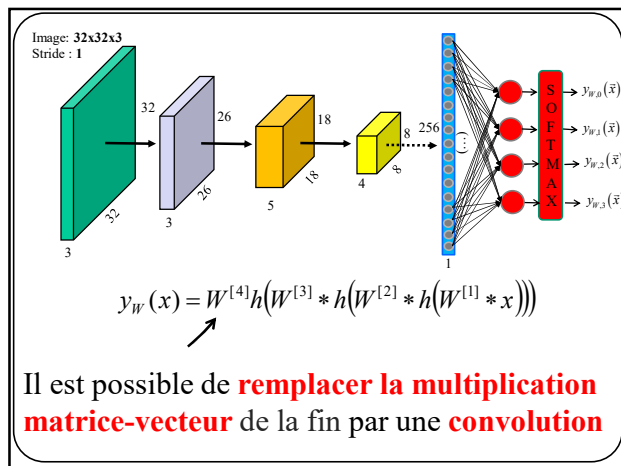
---

---

---

# Réseaux à convolution vs Réseaux **pleinement** convolutifs

127



128

## Exemple 1d

(convolution « valid »)

$$\begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 40 & 50 & 70 & 80 & 90 & 10 & 20 & 30 & 40 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline .2 & -.3 & .4 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|c|} \hline 21 & 21 & 26 & -7 & 23 & 8 & 11 \\ \hline \end{array}$$

129

**Exemple 1d**  
(convolution « valid »)

$\xrightarrow{9}$ 
 $\xleftarrow{5}$ 
 $\xleftarrow{5}$

40	50	70	80	90	10	20	30	40
----	----	----	----	----	----	----	----	----

 $*$ 

.2	-.3	.4	-.5	.6
----	-----	----	-----	----

 $=$ 

35	-18	33	1	32
----	-----	----	---	----

130

---

---

---

---

---

---

---

---

**Exemple 1d**  
(convolution « valid »)

$\xrightarrow{9}$ 
 $\xleftarrow{7}$ 
 $\xleftarrow{3}$

40	50	70	80	90	10	20	30	40
----	----	----	----	----	----	----	----	----

 $*$ 

.2	-.3	.4	-.5	.6	-.7	.8
----	-----	----	-----	----	-----	----

 $=$ 

44	-8	44
----	----	----

131

---

---

---

---

---

---

---

---

Taille filtre = nb de neurones couche précédente

$\xrightarrow{9}$ 
 $\xleftarrow{9}$ 
 $\xleftarrow{1}$

40	50	70	80	90	10	20	30	40
----	----	----	----	----	----	----	----	----

 $*$ 

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

 $=$ 

31
----

132

---

---

---

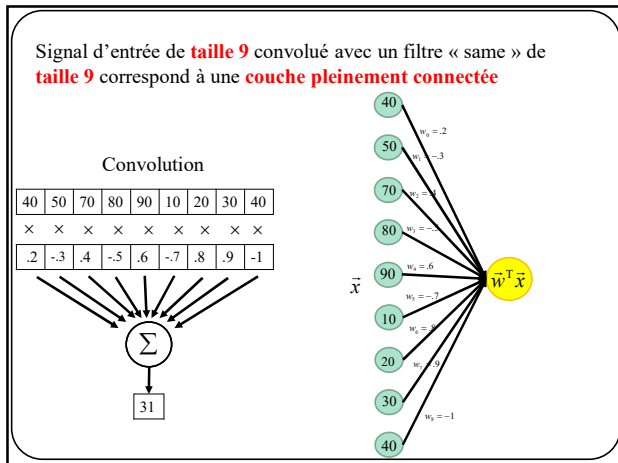
---

---

---

---

---



133

---

---

---

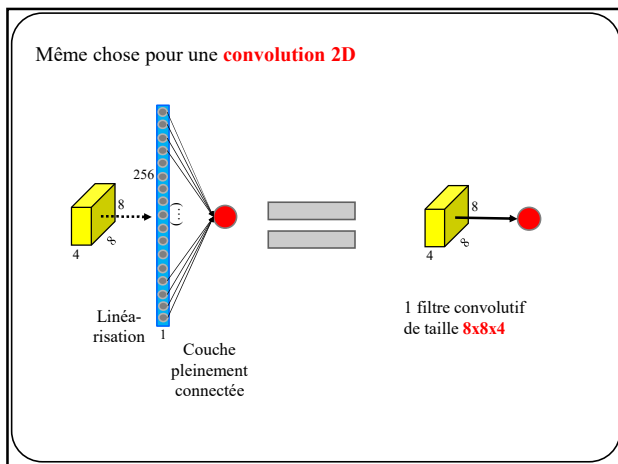
---

---

---

---

---



134

---

---

---

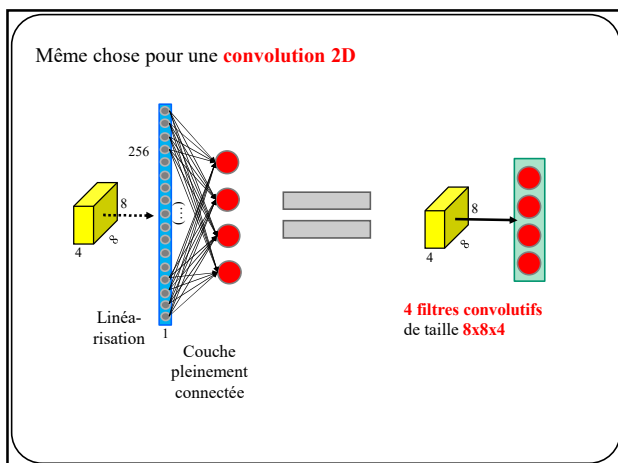
---

---

---

---

---



135

---

---

---

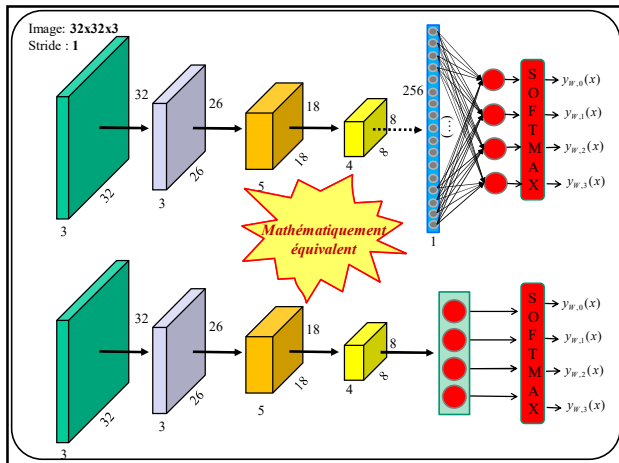
---

---

---

---

---



136

---

---

---

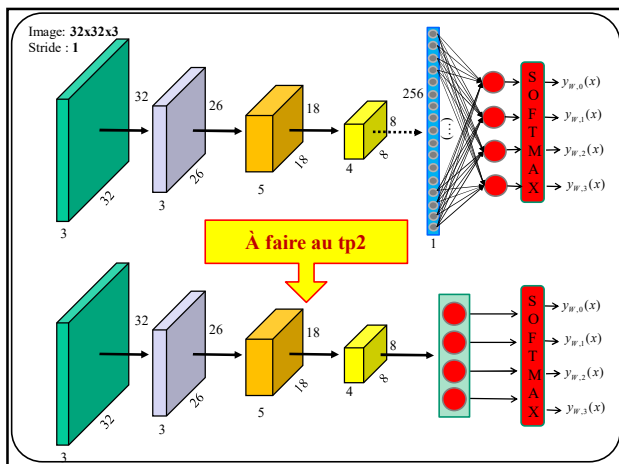
---

---

---

---

---



137

---

---

---

---

---

---

---

---

### Configurations équivalentes

couche 1 : 3 filtres de taille 7x7	couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9	couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11	couche 3 : 4 filtres de taille 11x11
<b>couche 4 : 4 filtres de taille 8x8</b>	<b>couche 4 : 4 filtres de taille 8x8</b>
Softmax	Softmax

En fait, presque équivalent ...

**Question : qu'arrive-t-il si on remplace l'image 32x32x3 par une image 64x64x3?**

138

---

---

---

---

---

---

---

---

# Pooling

139

---

---

---

---

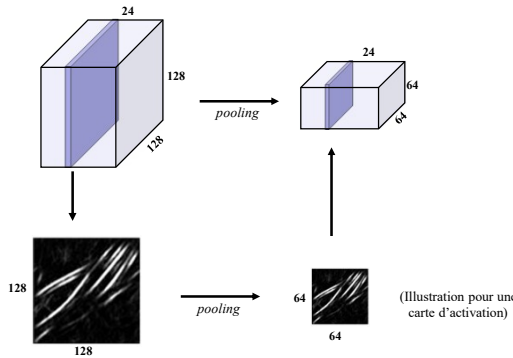
---

---

---

---

## Réduction de la taille des cartes d'activation



140

---

---

---

---

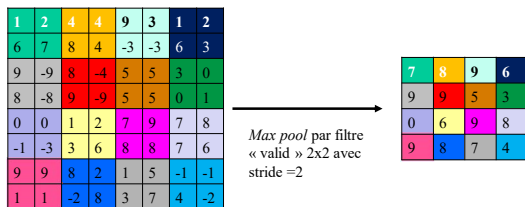
---

---

---

---

## Max pooling



141

---

---

---

---

---

---

---

---

## 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  
« valid » 2x2 avec  
stride=2

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

142

---

---

---

---

---

---

---

---

## 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 pooling 2x2  
avec **stride=1**



143

---

---

---

---

---

---

---

---

## 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 pooling 3x3  
avec **stride=2**



144

---

---

---

---

---

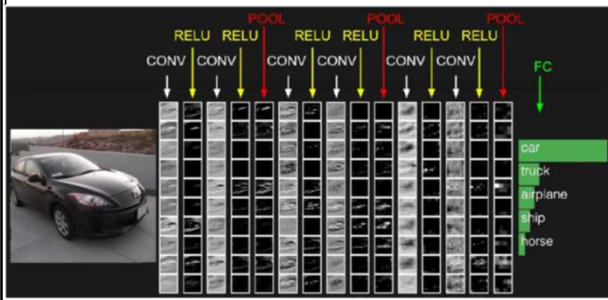
---

---

---



## Illustration d'un CNN complet



Credit : cs231 Stanford

145

## Multiplication matricielle parcimonieuse

<https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215>

147

Il est **plus rapide** de multiplier des matrices que de les convoluer.

Ex.: convolution « valid », un canal d'entrée et une carte d'activation, filtre 3x3

Entrée					Filtre					
X0	X1	X2	X3	*	W0	W1	W2	=	Y1	Y2
X4	X5	X6	X7		W3	W4	W5		Y3	Y4
X8	X9	X10	X11		W6	W7	W8			
X12	X13	X14	X15							

148

Il est **plus rapide** de multiplier des matrices que de les convoluer.

Ex.: convolution « valid », un canal d'entrée et une carte d'activation, filtre 3x3

Entrée					Filtre					
X0	X1	X2	X3	*	W0	W1	W2	=	Y0	Y1
X4	X5	X6	X7		W3	W4	W5		Y2	Y3
X8	X9	X10	X11		W6	W7	W8			
X12	X13	X14	X15							

On peut **remplacer** une **convolution** par une **multiplication matrice-matrice** ou **matrice-vecteur**

en **linéarisant** le filtre et en « **matriciant** » l'entrée

149

---

---

---

---

---

---

---

---

## Rappel

Ex.: convolution « valid », un canal d'entrée et une carte d'activation, filtre 3x3

W0	W1	W2	X3		Y0	Y1
W3	W4	W5	X7		Y2	Y3
W6	W7	W8	X11			
X12	X13	X14	X15			

$$Y0 = W0.X0 + W1.X1 + W2.X2 + W3.X4 + W4.X5 + W5.X6 + W6.X8 + W7.X9 + W8.X10$$

150

---

---

---

---

---

---

---

---

## Rappel

Ex.: convolution « valid », un canal d'entrée et une carte d'activation, filtre 3x3

X0	W0	W1	W2		Y0	Y1
X4	W3	W4	W5		Y2	Y3
X8	W6	W7	W8			
X12	X13	X14	X15			

$$Y1 = W0.X1 + W1.X2 + W2.X3 + W3.X5 + W4.X6 + W5.X7 + W6.X9 + W7.X10 + W8.X11$$

151

---

---

---

---

---

---

---

---

## Rappel

Ex.: convolution « valid », un canal d'entrée et une carte d'activation, filtre 3x3

X0	X1	X2	X3
W0	W1	W2	X7
W3	W4	W5	X11
W6	W7	W8	X15

Y0	Y1
Y2	Y3

$$Y2 = W0.X4 + W1.X5 + W2.X6 + W3.X8 + W4.X9 + W5.X10 + W6.X12 + W7.X13 + W8.X14$$

152

---

---

---

---

---

---

---

---

## Rappel

Ex.: convolution « valid », un canal d'entrée et une carte d'activation, filtre 3x3

X0	X1	X2	X3
X4	W0	W1	W2
X8	W3	W4	W5
X12	W6	W7	W8

Y0	Y1
Y2	Y3

$$Y3 = W0.X5 + W1.X6 + W2.X7 + W3.X9 + W4.X10 + W5.X11 + W6.X13 + W7.X14 + W8.X15$$

153

---

---

---

---

---

---

---

---

## Autrement dit...

W0	W1	W2	X3
W3	W4	W5	X7
W6	W7	W8	X11
X12	X13	X14	X15

X0
X1
X2
X4
X5
X6
X8
X9
X10

Y0
----

154

---

---

---

---

---

---

---

---

Autrement dit...

X0	W0	W1	W2
X4	W3	W4	W5
X8	W6	W7	W8
X12	X13	X14	X15

X0	X1
X1	X2
X2	X3
X4	X5
X5	X6
X6	X7
X8	X9
X9	X10
X10	X11

Y0	Y1
----	----

155

---

---

---

---

---

---

---

---

Autrement dit...

X0	X1	X2	X3
W0	W1	W2	X7
W3	W4	W5	X11
W6	W7	W8	X15

X0	X1	X4
X1	X2	X5
X2	X3	X6
X4	X5	X8
X5	X6	X9
X6	X7	X10
X8	X9	X11
X9	X10	X12
X10	X11	X13

Y0	Y1
Y2	

156

---

---

---

---

---

---

---

---

Autrement dit...

X0	X1	X2	X3
X4	W0	W1	W2
X8	W3	W4	W5
X12	W6	W7	W8

X0	X1	X4	X5
X1	X2	X5	X6
X2	X3	X6	X7
X4	X5	X8	X9
X5	X6	X9	X10
X6	X7	X10	X11
X8	X9	X11	X13
X9	X10	X12	X14
X10	X11	X13	X15

Y0	Y1
Y2	Y3

157

---

---

---

---

---

---

---

---

Convolution « valid » en **linéarisant le filtre** et en « **matriquant** » l'entrée

w0	w1	w2	w3	w4	w5	w6	w7	w8
----	----	----	----	----	----	----	----	----

x

x0	x1	x4	x5
x1	x2	x5	x6
x2	x3	x6	x7
x4	x5	x8	x9
x5	x6	x9	x10
x6	x7	x10	x11
x8	x9	x11	x13
x9	x10	x12	x14
x10	x11	x13	x15

=

y0	y1	y2	y3
----	----	----	----

158

Autre exemple  
conv « valid », mini-batch de 2 entrées

2 données en entrée

x0	x1	x2	x3
x4	x5	x6	x7
x8	x9	x10	x11
x12	x13	x14	x15
x16	x17	x18	x19
x20	x21	x22	x23
x24	x25	x26	x27
x28	x29	x30	x31

\*

Filtre

w0	w1	w2
w3	w4	w5
w6	w7	w8

=

y0	y1
y2	y3
y4	y5
y6	y7

159

Autre exemple  
conv « valid », mini-batch de 2 entrées

w0	w1	w2	w3	w4	w5	w6	w7	w8
----	----	----	----	----	----	----	----	----

x

x16	x17	x20	x21
x0	x1	x4	x5
x1	x2	x5	x6
x2	x3	x6	x7
x4	x5	x8	x9
x5	x6	x9	x10
x6	x7	x10	x11
x8	x9	x11	x13
x9	x10	x12	x14
x10	x11	x13	x15

=

y0	y1	y2	y3
y4	y5	y6	y7

160

### Autre exemple

conv « valid », une entrée, deux filtres

X0	X1	X2	X3
X4	X5	X6	X7
X8	X9	X10	X11
X12	X13	X14	X15

\*

W0	W1	W2
W3	W4	W5
W6	W7	W8
W9	W10	W11
W12	W13	W14
W15	W16	W17

=

Y0	Y1
Y2	Y3
Y4	Y5
Y6	Y7

161

---

---

---

---

---

---

---

---

### Autre exemple

conv « valid », une entrée, deux filtres

W0	W1	W2	W3	W4	W5	W6	W7	W8
W9	W10	W11	W12	W13	W14	W15	W16	W17

x

X0	X1	X4	X5
X1	X2	X5	X6
X2	X3	X6	X7
X4	X5	X8	X9
X5	X6	X9	X10
X6	X7	X10	X11
X8	X9	X11	X13
X9	X10	X12	X14
X10	X11	X13	X15

=

Y0	Y1	Y2	Y3
Y4	Y5	Y6	Y7

162

---

---

---

---

---

---

---

---

### Autre exemple

conv « valid », une entrée avec deux canaux, un filtre

X0	X1	X2	X3
X4	X5	X6	X7
X8	X9	X10	X11
X12	X13	X14	X15
X16	X17	X18	X19
X20	X21	X22	X23
X24	X25	X26	X27
X28	X29	X30	X31

\*

W0	W1	W2
W3	W4	W5
W6	W7	W8
W9	W10	W11
W12	W13	W14
W15	W16	W17

=

Y0	Y1
Y2	Y3

163

---

---

---

---

---

---

---

---

### Autre exemple

conv « valid », une entrée avec deux canaux, un filtre

W0	W1	W2	W3	(...)	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15	W16	W17
----	----	----	----	-------	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----

X

X0	X1	X4	X5
X1	X2	X5	X6
X2	X3	X6	X7
X4	X5	X8	X9
(...)	(...)	(...)	(...)
X22	X23	X26	X27
X24	X25	X27	X29
X25	X26	X28	X30
X26	X27	X29	X31

=

Y0	Y1	Y2	Y3
----	----	----	----

164

---

---

---

---

---

---

---

---

## Exemples d'architectures connues

165

---

---

---

---

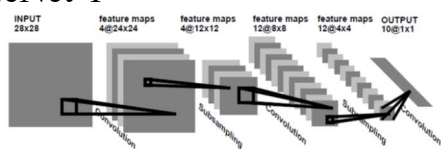
---

---

---

---

### LeNet-1



INPUT 28x28

feature maps 4@24x24

feature maps 4@12x12

feature maps 12@8x8

feature maps 12@4x4

OUTPUT 10@1x1

Une des plus vieilles architectures faites pour la reconnaissance de caractères.

- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 4 et 12 filtres)
- 2 *average pooling*
- 1 couche pleinement connectée
- 10 classes

Input

CONV

POOL

CONV

POOL

FC

SOFTMAX

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4):541-551

166

---

---

---

---

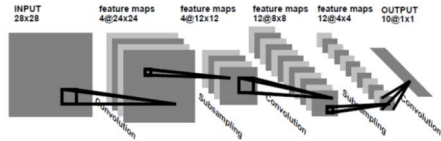
---

---

---

---

## LeNet-1



Une des plus vieilles architectures faites pour la reconnaissance de caractères.

- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 4 et 12 filtres)
- 2 *average pooling*
- 1 couche pleinement connectée
- 10 classes

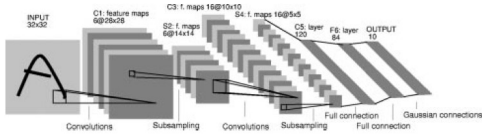
Input  
CONV  
POOL  
CONV  
POOL  
FC  
SOFTMAX

Combien de neurones?  
Combien de paramètres?

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541-551

167

## LeNet-5



Version améliorée:

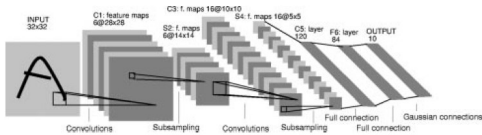
- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 6 et 16 filtres)
- 2 *average pooling*
- 3 couches pleinement connectées (120, 84 et 10 neurones)
- 10 classes

Input  
CONV  
POOL  
CONV  
POOL  
FC  
FC  
FC  
SOFTMAX

LeCun, Y.; Bottou, L.; Bengio, Y. & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278 - 232

168

## LeNet-5



Version améliorée:

- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 6 et 16 filtres)
- 2 *average pooling*
- 3 couches pleinement connectées (120, 84 et 10 neurones)
- 10 classes

Input  
CONV  
POOL  
CONV  
POOL  
FC  
FC  
FC  
SOFTMAX

Combien de neurones?  
Combien de paramètres?

169



## Classification d'images

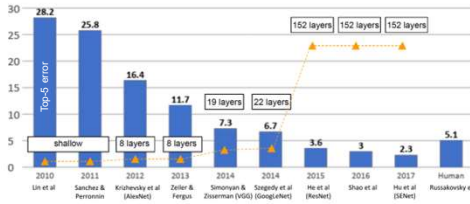
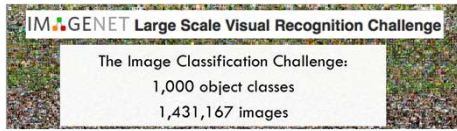


Image: <http://cs231n.stanford.edu/slides/2018>

170

170

## Classification d'images

AlexNet [Krizhevsky et al, 2012]

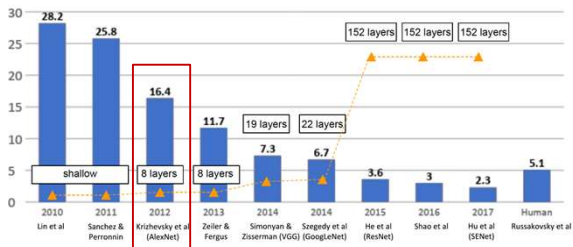
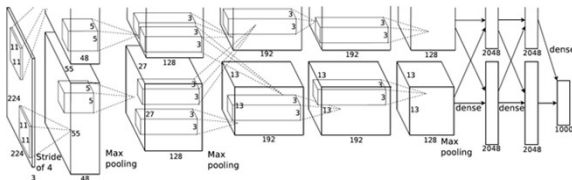


Image: <http://cs231n.stanford.edu/slides/2018>

171

171

## AlexNet



- Premier CNN à bien performer sur ImageNet (amélioration de 10% par rapport aux autres)
- Utilisation de techniques aujourd'hui fréquemment utilisées: **ReLU**, **data augmentation** and **dropout**
- Utilisation de **GPUs** (2 dans leur cas)
- Point de départ de la **révolution du "deep learning"** en vision par ordinateur

Image: Krizhevsky et al. "Imagenet classification with deep convolutional neural networks," NIPS 2012.

172

CONV1  
MAX POOL1  
NORM1  
CONV2  
MAX POOL2  
NORM2  
CONV3  
CONV4  
CONV5  
Max POOL3  
FC6  
FC7  
FC8

Réponse:  $(227-11)/4+1 = 55$

---

---

---

---

---

---

CONV1  
MAX POOL1  
NORM1  
CONV2  
MAX POOL2  
NORM2  
CONV3  
CONV4  
CONV5  
Max POOL3  
FC6  
FC7  
FC8

Q: Quel est le nombre de paramètres?

---

---

---

---

---

---

CONV1  
MAX POOL1  
NORM1  
CONV2  
MAX POOL2  
NORM2  
CONV3  
CONV4  
CONV5  
Max POOL3  
FC6  
FC7  
FC8

Paramètres :  $11 \times 11 \times 96 \times 3 = 34,848$ 

---

---

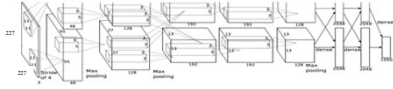
---

---

---

---

## AlexNet



**ENTRÉE** : 227x227x3  
**CONV1**: 96 x 55 x 55

**Couche 2** MaxPool : 3x3 stride stride 2

Quelle est la taille des cartes d'activation?

Réponse:  $(55-3)/2+1 = 27$

176

---

---

---

---

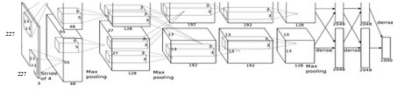
---

---

---

---

## AlexNet



**ENTRÉE** : 227x227x3  
**CONV1**: 96 x 55 x 55

**Couche 2** MaxPool : 3x3 stride stride 2  
 27 x 27 x 96

Combien y a-t-il de paramètres?

Réponse: 0!

177

---

---

---

---

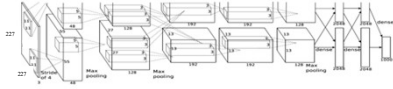
---

---

---

---

## AlexNet



**ENTRÉE** : 227x227x3  
**CONV1**: 55 x 55 x 96  
**MAX POOL1**: 27 x 27 x 96  
 ...

178

---

---

---

---

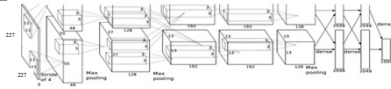
---

---

---

---

## AlexNet



**ENTRÉE** 227 x 227 x 3  
 96 filtres 11x11, stride 4, pad 0, **CONV1** 55 x 55 x 96  
 filtre 3x3, stride 2, **MAX POOL1** 27 x 27 x 96  
 normalisation par couche, **NORM1** 27 x 27 x 96  
 256 filtres 5x5, stride 1, pad 2, **CONV2** 27 x 27 x 256  
 filtre 3x3, stride 2, **MAX POOL2** 13 x 13 x 256  
 normalisation par couche, **NORM2** 13 x 13 x 256  
 384 filtres 3x3, stride 1, pad 1, **CONV3** 13 x 13 x 384  
 384 filtres 3x3, stride 1, pad 1, **CONV4** 13 x 13 x 384  
 256 filtres 3x3, stride 1, pad 1, **CONV5** 13 x 13 x 256  
 filtre 3x3, stride 2, **MAX POOL3** 6 x 6 x 256  
**FC6** 4096  
**FC7** 4096  
**FC8** 1000

179

---

---

---

---

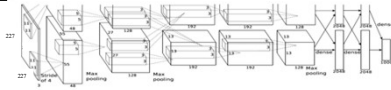
---

---

---

---

## AlexNet



**ENTRÉE** 227 x 227 x 3  
**CONV1** 55 x 55 x 96  
**MAX POOL1** 27 x 27 x 96  
**NORM1** 27 x 27 x 96  
**CONV2** 27 x 27 x 256  
**MAX POOL2** 13 x 13 x 256  
**NORM2** 13 x 13 x 256  
**CONV3** 13 x 13 x 384  
**CONV4** 13 x 13 x 384  
**CONV5** 13 x 13 x 256  
**MAX POOL3** 6 x 6 x 256  
**FC6** 4096  
**FC7** 4096  
**FC8** 1000

### Notes additionnelles:

Fonction d'activation **ReLU**  
**Augmentation de données**  
**LayerNorm** : peu utilisé aujourd'hui  
**Dropout** 0.5  
**Batch\_size** 128  
**SGD + momentum**  
**Taux d'apprentissage** 0.01 avec  
 réduction par plateau d'un facteur 10  
**~68 millions** de paramètres

180

---

---

---

---

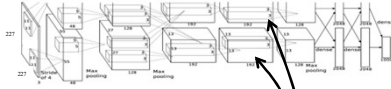
---

---

---

---

## AlexNet



**ENTRÉE** 227 x 227 x 3  
**CONV1** 55 x 55 x 96  
**MAX POOL1** 27 x 27 x 96  
**NORM1** 27 x 27 x 96  
**CONV2** 27 x 27 x 256  
**MAX POOL2** 13 x 13 x 256  
**NORM2** 13 x 13 x 256  
**CONV3** 13 x 13 x 384  
**CONV4** 13 x 13 x 384  
**CONV5** 13 x 13 x 256  
**MAX POOL3** 6 x 6 x 256  
**FC6** 4096  
**FC7** 4096  
**FC8** 1000

### Notes additionnelles:

Utilisation de **2 GPUs**  
**Fait voter un ensemble de 7 CNN:**  
 18.2% -> 15.4

181

---

---

---

---

---

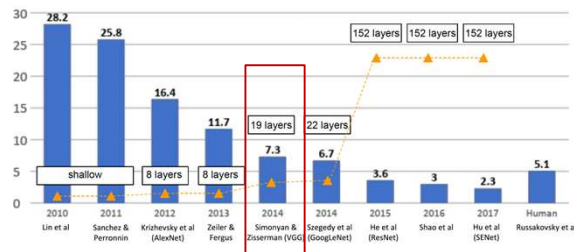
---

---

---

# Classification d'images

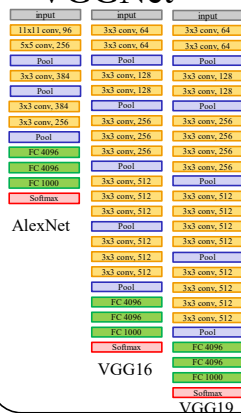
VGGNet [Simonyan and Zisserman, 2014]



Karen Simonyan, Andrew Zisserman "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015  
Image: <http://cs231n.stanford.edu/slides/2018>

182

## VGGNet



Ce qui caractérise VGGNet par rapport à ses prédécesseurs:

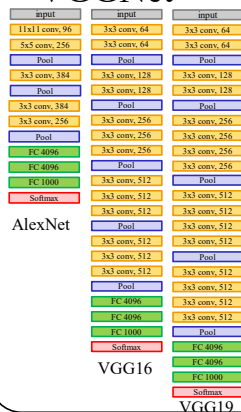
- Uniquement des **filtres 3x3, stride 1, pad 1**
- **Plus profond**

AlexNet : 8 couches  
VGGNet : 16 ou 19 couches

7.3% d'erreur contre 11.7% pour ZFNet

183

## VGGNet

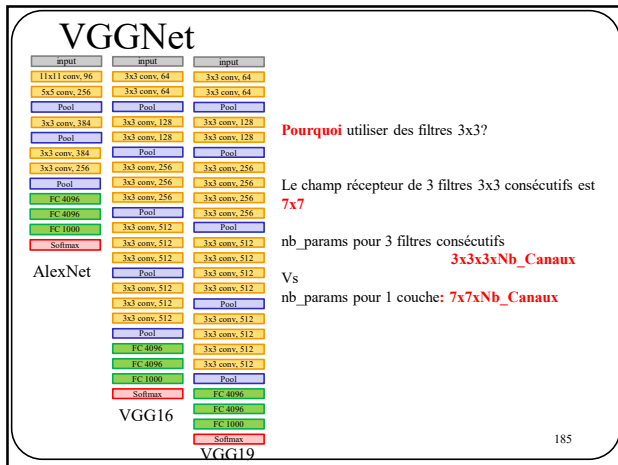


**Pourquoi** utiliser des filtres 3x3?

**Indice1** : quel est le **champ récepteur** (*receptive field*) d'une série de 3 couches 3x3?

**Indice2** : combien de paramètres pour ces 3 couches?

184



185

---

---

---

---

---

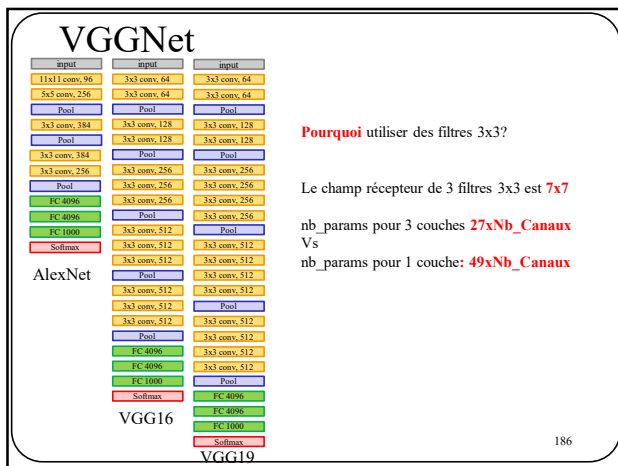
---

---

---

---

---



186

---

---

---

---

---

---

---

---

---

---



187

---

---

---

---

---

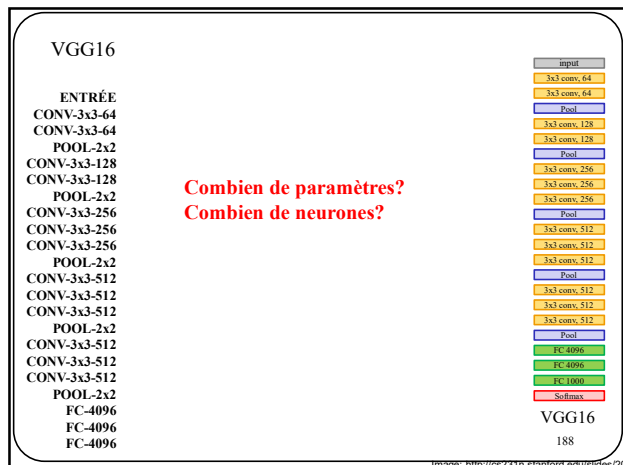
---

---

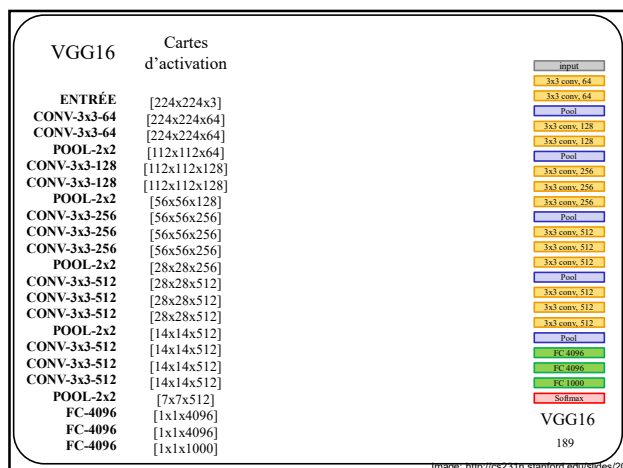
---

---

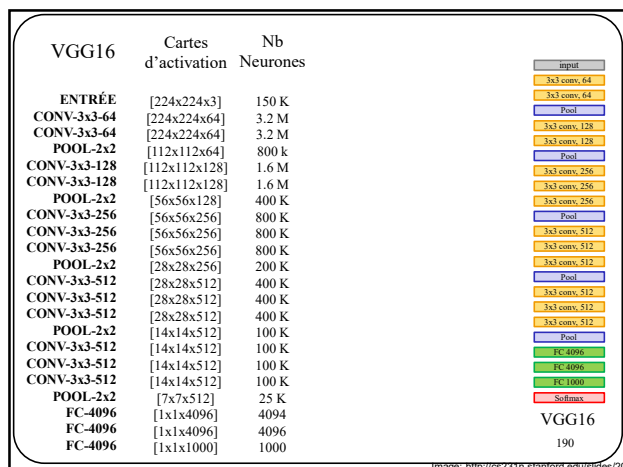
---



188



189



190

VGG16	Cartes d'activation	Nb Neurones	Nb Paramètres	
ENTRÉE	[224x224x3]	150 K	0	input
CONV-3x3-64	[224x224x64]	3.2 M	$(3*3*3)*64 = 1,728$	3x3 conv, 64
CONV-3x3-64	[224x224x64]	3.2 M	$(3*3*64)*64 = 36,864$	3x3 conv, 64
POOL-2x2	[112x112x64]	800 k	0	Pool
CONV-3x3-128	[112x112x128]	1.6 M	$(3*3*64)*128 = 73,728$	3x3 conv, 128
CONV-3x3-128	[112x112x128]	1.6 M	$(3*3*128)*128 = 147,456$	3x3 conv, 128
POOL-2x2	[56x56x128]	400 K	0	Pool
CONV-3x3-256	[56x56x256]	800 K	$(3*3*128)*256 = 294,912$	3x3 conv, 256
CONV-3x3-256	[56x56x256]	800 K	$(3*3*256)*256 = 589,824$	3x3 conv, 256
CONV-3x3-256	[56x56x256]	800 K	$(3*3*256)*256 = 589,824$	3x3 conv, 256
POOL-2x2	[28x28x256]	200 K	0	Pool
CONV-3x3-512	[28x28x512]	400 K	$(3*3*256)*512 = 1,179,648$	3x3 conv, 512
CONV-3x3-512	[28x28x512]	400 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
CONV-3x3-512	[28x28x512]	400 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
POOL-2x2	[14x14x512]	100 K	0	Pool
CONV-3x3-512	[14x14x512]	100 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
CONV-3x3-512	[14x14x512]	100 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
CONV-3x3-512	[14x14x512]	100 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
POOL-2x2	[7x7x512]	25 K	0	Pool
FC-4096	[1x1x4096]	4094	$7*7*512*4096 = 102,760,448$	FC, 4096
FC-4096	[1x1x4096]	4096	$4096*4096 = 16,777,216$	FC, 4096
FC-4096	[1x1x1000]	1000	$4096*1000 = 4,096,000$	FC, 1000
				Softmax
				VGG16
				191

191

VGGNet	
Nb neurones totaux : ~15 M	
Mémoire totale neurones (4 octets par neurones) : ~60 Mo	
Nb paramètres totaux : 138 M	
Mémoire total paramètres (4 octets par paramètres) : 552 Mo	
~ 612 Mo pour la propagation avant d'une image	
	input
	3x3 conv, 64
	3x3 conv, 64
	Pool
	3x3 conv, 128
	3x3 conv, 128
	Pool
	3x3 conv, 256
	3x3 conv, 256
	3x3 conv, 256
	Pool
	3x3 conv, 512
	3x3 conv, 512
	3x3 conv, 512
	Pool
	3x3 conv, 512
	3x3 conv, 512
	3x3 conv, 512
	Pool
	FC, 4096
	FC, 4096
	FC, 1000
	Softmax
	VGG16
	192

192

VGGNet	
Nb neurones totaux : ~15 M	
Mémoire totale neurones (4 octets par neurones) : ~60 Mo	
Nb paramètres totaux : ~138 M	
Mémoire total paramètres (4 octets par paramètres) : ~552 Mo	
~612 Mo pour la propagation avant d'une image	
>1.1 Go si on inclut la rétro-propagation	
	input
	3x3 conv, 64
	3x3 conv, 64
	Pool
	3x3 conv, 128
	3x3 conv, 128
	Pool
	3x3 conv, 256
	3x3 conv, 256
	3x3 conv, 256
	Pool
	3x3 conv, 512
	3x3 conv, 512
	3x3 conv, 512
	Pool
	3x3 conv, 512
	3x3 conv, 512
	3x3 conv, 512
	Pool
	FC, 4096
	FC, 4096
	FC, 1000
	Softmax
	VGG16
	193

193



## Réseaux récurrents

(si le temps le permet)

194

---

---

---

---

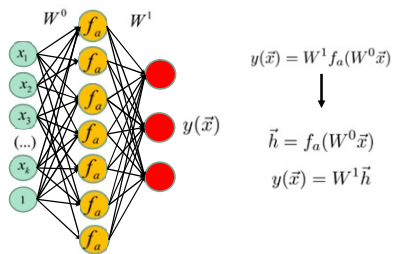
---

---

---

---

### Réseau de neurones de base (régression)



$$y(\vec{x}) = W^1 f_a(W^0 \vec{x})$$

$$\downarrow$$

$$\vec{h} = f_a(W^0 \vec{x})$$

$$y(\vec{x}) = W^1 \vec{h}$$

$f_a$  : fonction d'activation

195

---

---

---

---

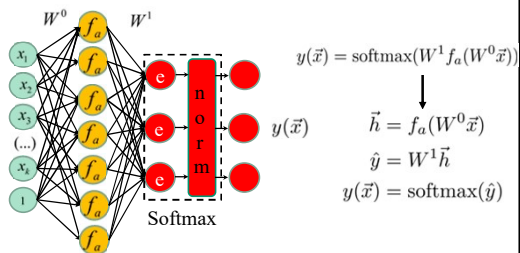
---

---

---

---

### Réseau de neurones de base (classification)



$$y(\vec{x}) = \text{softmax}(W^1 f_a(W^0 \vec{x}))$$

$$\downarrow$$

$$\vec{h} = f_a(W^0 \vec{x})$$

$$\hat{y} = W^1 \vec{h}$$

$$y(\vec{x}) = \text{softmax}(\hat{y})$$

196

---

---

---

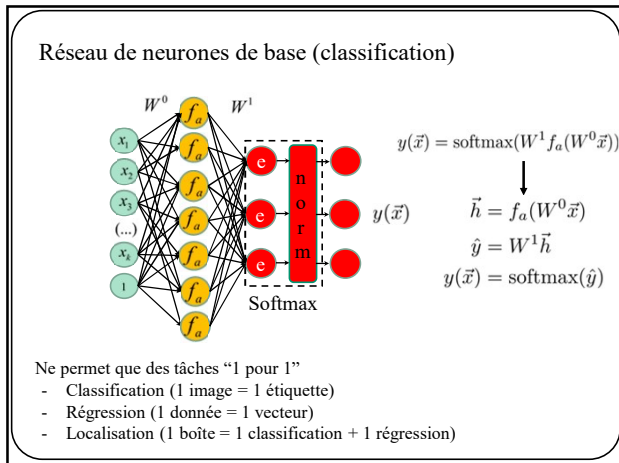
---

---

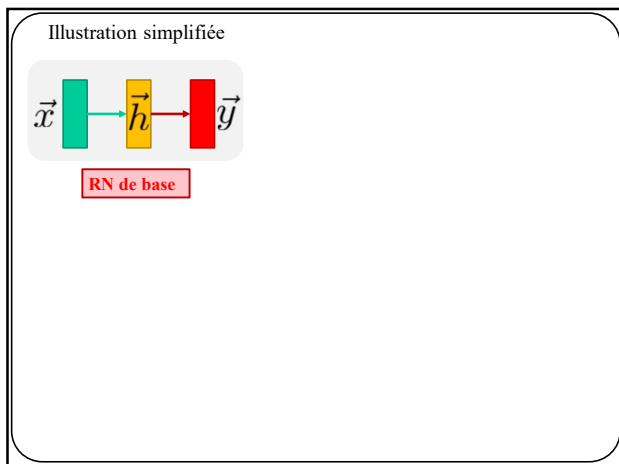
---

---

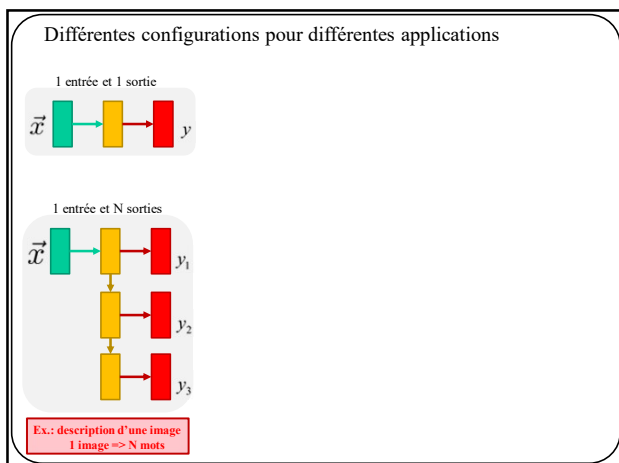
---



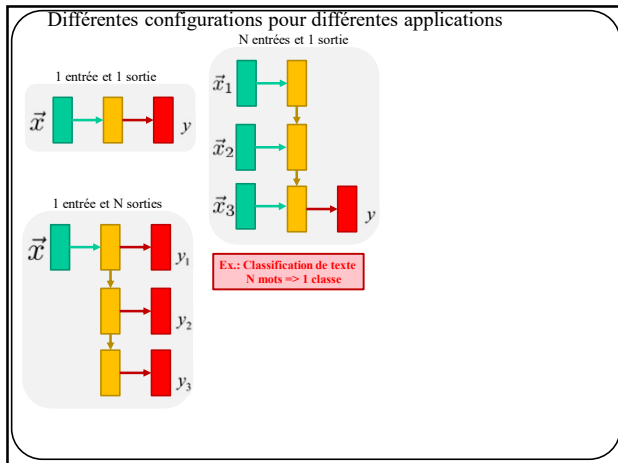
197



198



199



200

---

---

---

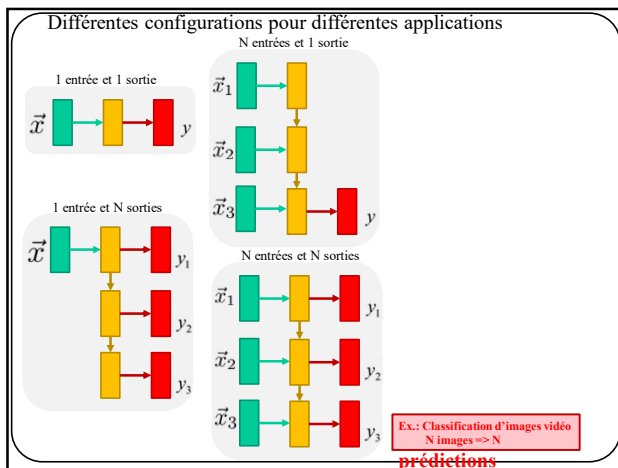
---

---

---

---

---



201

---

---

---

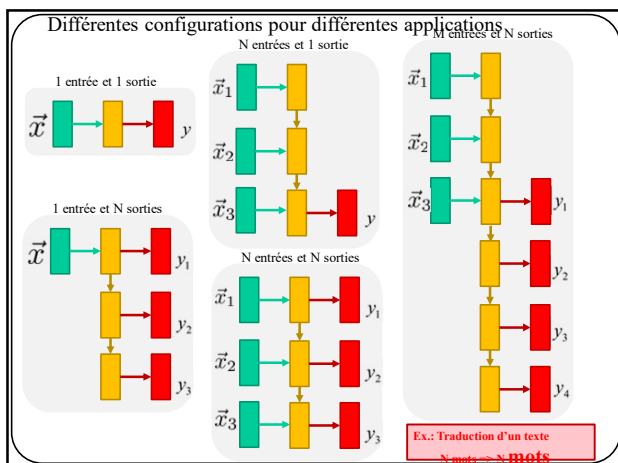
---

---

---

---

---



202

---

---

---

---

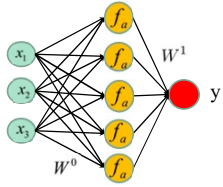
---

---

---

---

**Réseau récurrent** : la sortie des neurones est réinjectée dans leur entrée



203

---

---

---

---

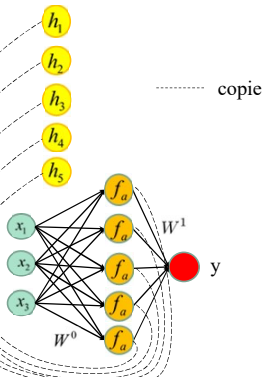
---

---

---

---

**Réseau récurrent** : la sortie des neurones est réinjectée dans leur entrée



204

---

---

---

---

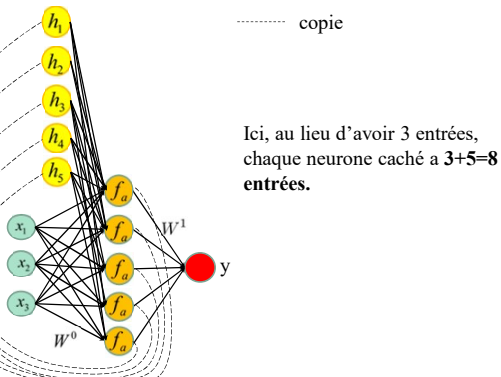
---

---

---

---

**Réseau récurrent** : la sortie des neurones est réinjectée dans leur entrée



205

---

---

---

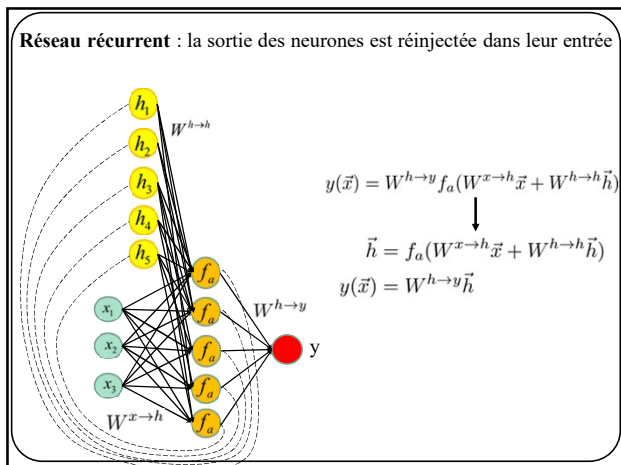
---

---

---

---

---



206

---

---

---

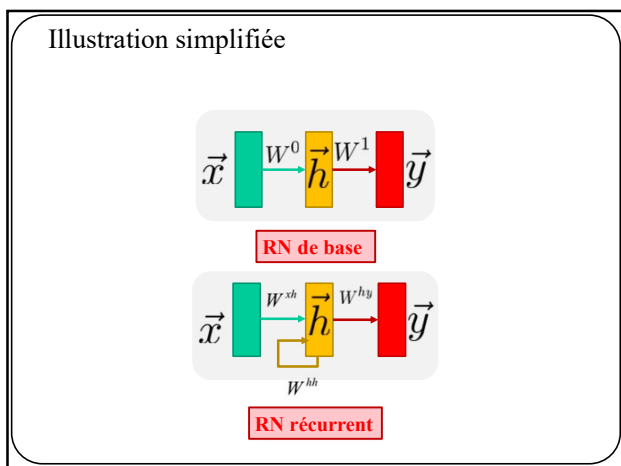
---

---

---

---

---



207

---

---

---

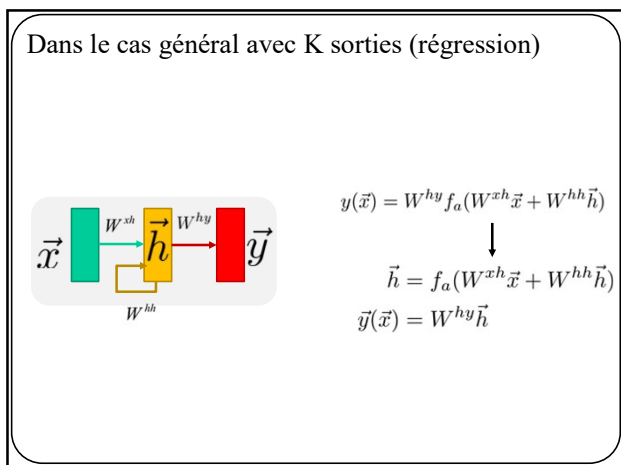
---

---

---

---

---



208

---

---

---

---

---

---

---

---

Dans le cas général avec K sorties (classification)

$$y(\vec{x}) = W^{hy} f_a(W^{xh} \vec{x} + W^{hh} \vec{h})$$

$$\vec{h} = f_a(W^{xh} \vec{x} + W^{hh} \vec{h})$$

$$\hat{y} = W^{hy} \vec{h}$$

$$\vec{y}(\vec{x}) = \text{softmax}(\hat{y})$$

209

---

---

---

---

---

---

---

---

Exemple pour N entrées et 1 sortie:  
Analyse grammaticale (classification) : (ê.t.r.e) => {verbe}

Puisque  $\vec{x}$ ,  $\vec{h}$  et  $y$  doivent être des **variables numériques**, on utilise souvent un encodage de type « one hot ».

'a' = [1, 0, ..., 0]	} $\in R^{256}$
'b' = [0, 1, ..., 0]	
'c' = [0, 0, 1, ..., 0]	
....	
'verbe' = [1, 0, 0, ..., 0]	} $\in R^M$
'nom' = [0, 1, 0, ..., 0]	
'adjectif' = [0, 0, 1, ..., 0]	

210

---

---

---

---

---

---

---

---

Exemple pour N entrées et 1 sortie:  
Analyse grammaticale (classification) : (ê.t.r.e) => {verbe}

$$\vec{h}_i = f_a(W^{xh} \vec{x}_i + W^{hh} \vec{h}_{i-1})$$

$$y = \text{softmax}(W^{hy} \vec{h}_{i-1})$$

211

---

---

---

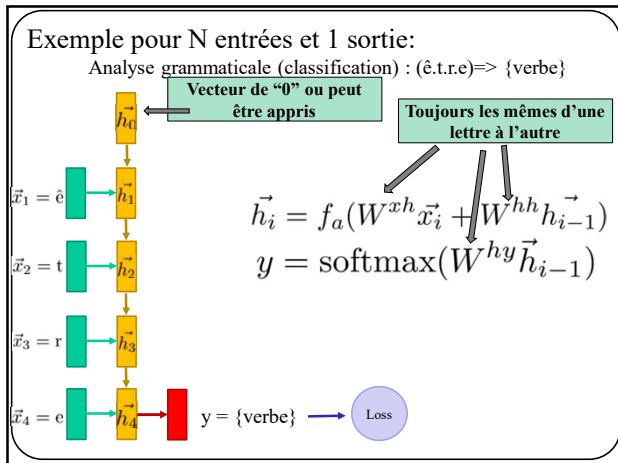
---

---

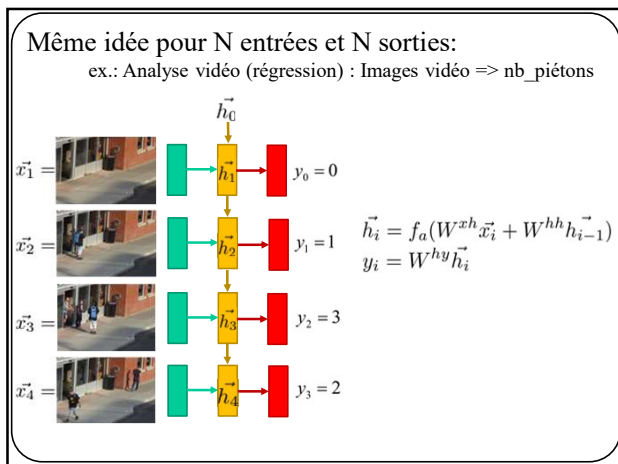
---

---

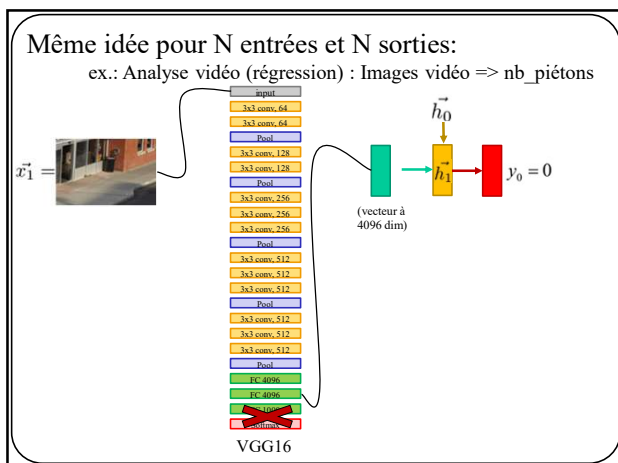
---



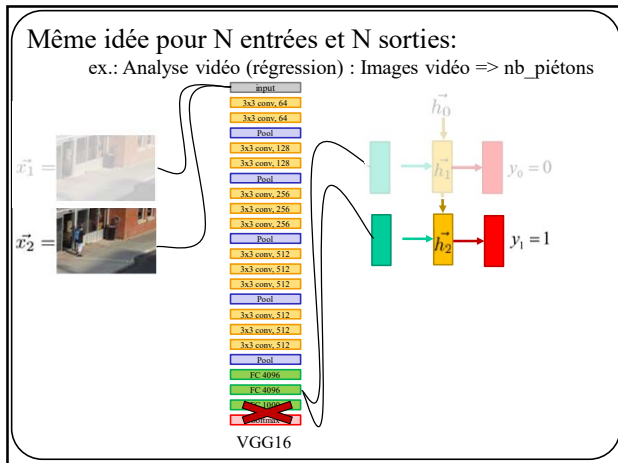
212



213



214



215

---

---

---

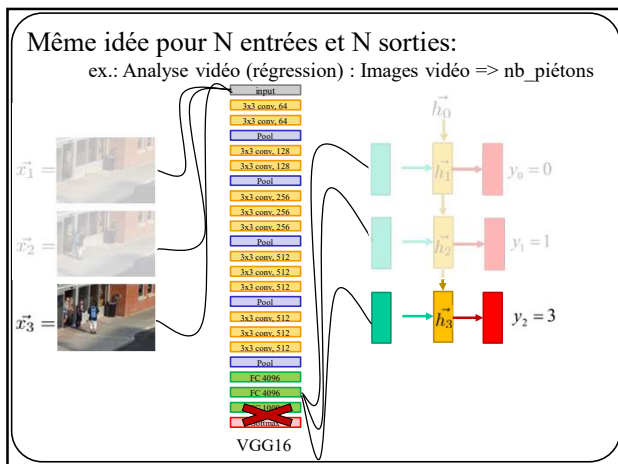
---

---

---

---

---



216

---

---

---

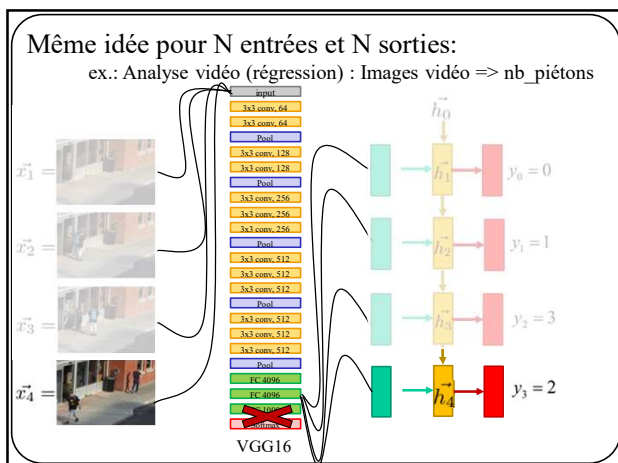
---

---

---

---

---



217

---

---

---

---

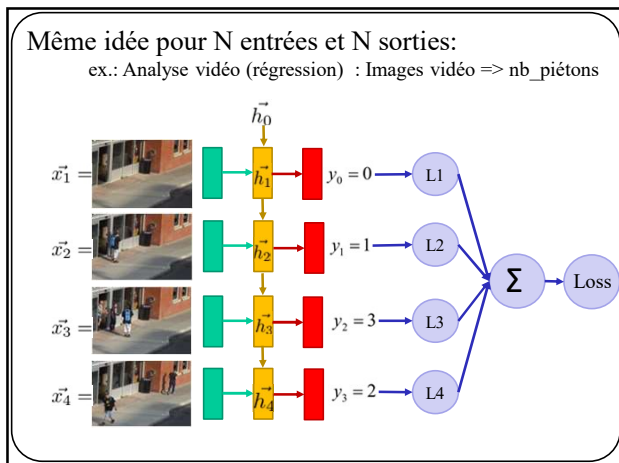
---

---

---

---





218

Autre exemple: **prédiction de caractères** (modèle de langue)

**Alphabet jouet** : {a,e,m,s}

**Représentation « one hot » jouet:**

'a' = [1, 0, 0, 0]  
 'e' = [0, 1, 0, 0]  
 'm' = [0, 0, 1, 0]  
 's' = [0, 0, 0, 1]

**But** : Entraîner un modèle à prédire les lettres du mot « masse ».

219

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet : {a,e,m,s}

**Entraîner** un modèle à prédire les lettres du mot « masse ».

$\vec{x}_1 = m$

220

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

Entraîner un modèle à prédire les lettres du mot « masse ».

$\vec{x}_1 = m$

$\vec{h}_i = \tanh(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1})$

221

---

---

---

---

---

---

---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

Entraîner un modèle à prédire les lettres du mot « masse ».

$\vec{x}_1 = m$

$\vec{h}_i = \tanh(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1})$

$\vec{y}(\vec{x}_i) = \text{softmax}(W^{hy}\vec{h}_i)$

222

---

---

---

---

---

---

---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

Entraîner un modèle à prédire les lettres du mot « masse ».

$\vec{x}_1 = m$

$\vec{x}_2 = a$

$\vec{h}_i = \tanh(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1})$

$\vec{y}(\vec{x}_i) = \text{softmax}(W^{hy}\vec{h}_i)$

223

---

---

---

---

---

---

---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

Entraîner un modèle à prédire les lettres du mot « masse ».

$\vec{x}_1 = m$

$\vec{x}_2 = a$

$\vec{x}_3 = s$

$$\vec{h}_i = \tanh(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1})$$

$$\vec{y}(\vec{x}_i) = \text{softmax}(W^{hy}\vec{h}_i)$$

224

---

---

---

---

---

---

---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

Entraîner un modèle à prédire les lettres du mot « masse ».

$\vec{x}_1 = m$

$\vec{x}_2 = a$

$\vec{x}_3 = s$

$\vec{x}_4 = s$

$$\vec{h}_i = \tanh(W^{xh}\vec{x}_i + W^{hh}\vec{h}_{i-1})$$

$$\vec{y}(\vec{x}_i) = \text{softmax}(W^{hy}\vec{h}_i)$$

225

---

---

---

---

---

---

---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

Entraîner un modèle à prédire les lettres du mot « masse ».

$\vec{x}_1 = m$

$\vec{x}_2 = a$

$\vec{x}_3 = s$

$\vec{x}_4 = s$

$\vec{t}_1 = a = [1000]$

$\vec{t}_2 = s = [0001]$

$\vec{t}_3 = s = [0001]$

$\vec{t}_4 = e = [0100]$

Cibles

226

---

---

---

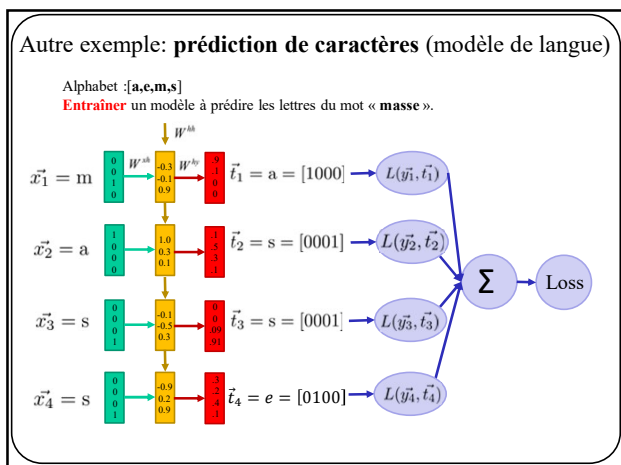
---

---

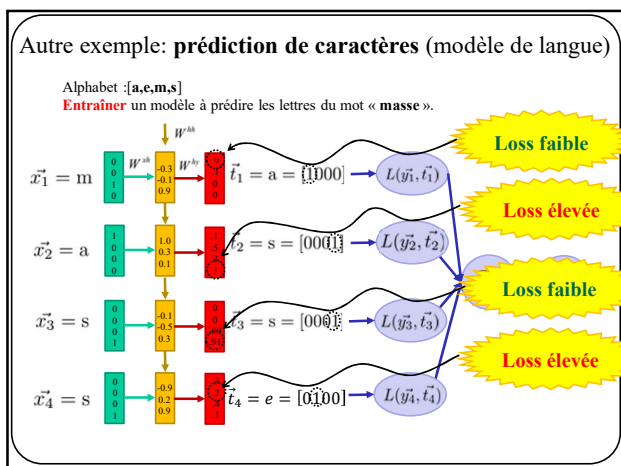
---

---

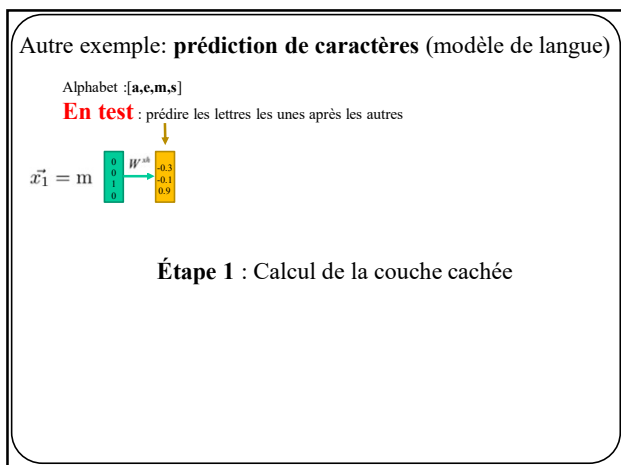
---



227



228



229

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[a,e,m,s]

**En test** : prédire les lettres les unes après les autres

$\vec{x}_1 = m$

**Étape 2** : Calcul de la sortie (softmax)

230

---

---

---

---

---

---

---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[a,e,m,s]

**En test** : prédire les lettres les unes après les autres

$\vec{x}_1 = m$

$y_1 = 'a'$

**Étape 3** : Sélectionner le caractère le plus probable

231

---

---

---

---

---

---

---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :[a,e,m,s]

**En test** : prédire les lettres les unes après les autres

$\vec{x}_1 = m$

$y_1 = 'a'$

**Étape 4** : Injecter le caractère prédit au début du réseau

232

---

---

---

---

---

---

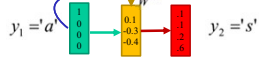
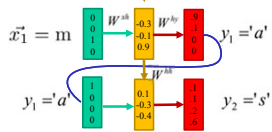
---

---

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

**En test** : prédire les lettres les unes après les autres



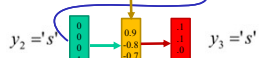
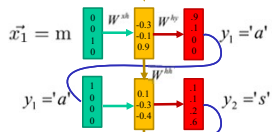
Et on recommence!

233

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

**En test** : prédire les lettres les unes après les autres

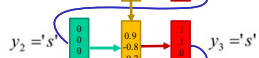
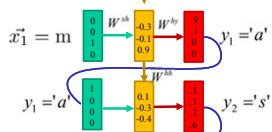


234

Autre exemple: **prédiction de caractères** (modèle de langue)

Alphabet :{a,e,m,s}

**En test** : prédire les lettres les unes après les autres



235

## Autre exemple: prédiction de caractères (modèle de langue)

Code python: “mini-char-RNN” de A. Karpathy

<https://gist.github.com/karpathy/d4dec566867f8291f086>

Un RNN en 112 lignes !

```
#!/usr/bin/env python3
# Mini-char-RNN model with a simple Recurrent Neural Network, in Python3
#
# This script implements a simple character-level Recurrent Neural Network (RNN)
# for predicting the next character in a sequence. It uses a simple RNN cell
# and a softmax output layer. The model is trained on a dataset of
# text files, and then used to generate new text.
#
# The model is trained using the following steps:
# 1. Load the dataset.
# 2. Preprocess the dataset.
# 3. Train the model.
# 4. Generate new text.
#
# The dataset is a list of text files. Each file is a sequence of
# characters. The dataset is preprocessed by converting the
# characters to integers. The model is trained using the
# following steps:
# 1. Initialize the model parameters.
# 2. Compute the loss.
# 3. Compute the gradients.
# 4. Update the model parameters.
#
# The model is then used to generate new text. The
# generated text is a sequence of characters. The
# generated text is then converted back to a string
# of characters.
```

$$\begin{aligned} 'a' &= [1, 0, 0, \dots, 0] \\ 'b' &= [0, 1, 0, \dots, 0] \\ 'c' &= [0, 0, 1, \dots, 0] \end{aligned} \left. \vphantom{\begin{aligned} 'a' &= [1, 0, 0, \dots, 0] \\ 'b' &= [0, 1, 0, \dots, 0] \\ 'c' &= [0, 0, 1, \dots, 0] \end{aligned}} \right\} \in \mathbb{R}^{256}$$

...

236

---

---

---

---

---

---

---

---

---

---

## Autre exemple: prédiction de caractères (modèle de langue)

Code python: “mini-char-RNN” de A. Karpathy

<https://gist.github.com/karpathy/d4dec566867f8291f086>

### THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,  
That fruitful breed may lengthen our days;  
But, since the sovereign love truly excites  
The soul to sweetest thoughts, and fancies,  
It under her soft poise brings on confusion,  
The shew of others' bodies doth unfix  
Thy eye, thy heart, thy love, and my desire;  
Thou wilt say, then, I'm foolish and unwise,  
To use that word which trespasses thine eyes;  
But thou dost wrong to that particular use,  
Whose look with my suppliance doth deceive,  
And well thou dost to share my rage, when I  
Proclaim that I will love thee till I die.

When forty winters shall besiege thy brow,  
And dig deep trenches in thy beauty's field,  
Thy youth's proud livery so fashed away,  
Will be a tattered weed of small worth held:  
Then to the steeps of old age I lead thee down,  
Where thou shalt behold how the blossom lies,  
Even as the bare fallow brings the seed to tillage,  
And the bare fallow brings the seed to tillage,  
Even as the bare fallow brings the seed to tillage,  
And the bare fallow brings the seed to tillage,  
Even as the bare fallow brings the seed to tillage,  
And the bare fallow brings the seed to tillage,

237

---

---

---

---

---

---

---

---

---

---

## Autre exemple: prédiction de caractères (modèle de langue)

tyntd-lafhatawlaohrdemot lytdws e ,ftti, astai f ogoe eoase rranbyne 'nhthnee e  
plia tkirpd t o idoe ns,smtt h ne etie h,regtrs nigtkie,soeans lng

train more

"Taont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, ammerenith ol sivh I lalterthend Bleipile shuw fil on aseterlome  
cooniogennc Phe lism thond hon at. Meidimorotoin in ther thize."

train more

Affair fall unsuch that the hall for Prince Velzonski's that me of  
her hoarly, and behs to so arwope fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and efter.

train more

"Why do what that day," replied Natasha, and wishing to himself the fact the  
princess, Princess Mary was easier, fed in had oftended him.  
Pierre aking his soul came to the packs and drove up his father-in-law women.

Crédit: A. Karpathy, CS231

238

---

---

---

---

---

---

---

---

---

---

## Texte généré une fois le modèle entraîné

PANDARUS:  
Alas, I think he shall be come approached and the day  
when little strain would be attain'd into being never fed,  
And who is but a chain and subjects of his death,  
I should not sleep.

Second Senator:  
They are away this miseries, produced upon my soul,  
Breaking and strongly should be buried, when I perish  
The earth and thoughts of many states.

DUKE VINCENTIO:  
Well, your wit is in the care of side and that.

Second Lord:  
They would be ruled after this chamber, and  
my fair news begun out of the fact, to be conveyed,  
Whose noble souls I'll have the heart of the wars.

Clown:  
Come, sir, I will make did behold your worship.

VIOLA:  
I'll drink it.

VIOLA:  
Why, Salisbury must find his flesh and thought  
That which I am not age, not a man and in fire,  
To show the raining of the reaven and the wars  
To grace my hand reproach within, and not a fair are hand.  
That Caesar and my goodly father's world;  
When I was heaven of presence and our finest,  
We spare with hours, but cut thy council I am great.  
Murdered and by thy master's ready there  
My power to give thou but so much as hell;  
Some service in the noble bondman here,  
Would show him to bar wise.

KING LEAR:  
O, if you were a fumble sight, the courtesy of your law,  
Your sight and several breath, will wear the gods  
With his hands, and my hands are wonder'd at the deeds,  
So drop upon your lordship's head, and your opinion  
shall be against your honour.

Crédit: A. Karpathy, CS231

239

## Entraînement sur le code source de Linux en C++

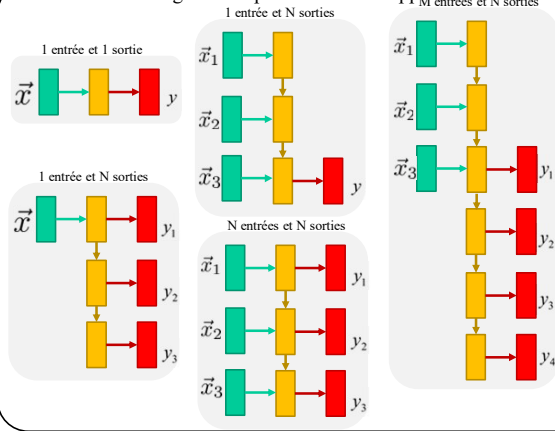
```
static void do_command(struct sig_file *s, void *)
{
    int column = 32 << (cmd[1] & 0x00);
    if (state)
        cmd = (int)(int_state * (int)(int_state_flags & CMD 7 2 * 1));
    else
        cmd = 1;
    for (i = 0; i < 16; i++) {
        if (i & 0x0001) {
            pipe = (int)(int_state & UNKTHREAD_UNCKA) +
                ((count & 0x00000000ffff) & 0x00000000) << 8;
            if (count == 0)
                subpid_ppc_md_kexec_handle, 0x00000000;
            pipe_set_bytes(i, 0);
        }
        /* From our user pages pointer to place names if all dash */
        subsystem_info = sub_changepage_PAGE_SIZE;
        sub_control(effect, id, subdata);
        /* Now we want to deliberately get it to derive */
        control_check_polarity(count, val, 0);
        for (i = 0; i < UNKTHREAD_LEN)
            sub_data(i, "policy");
    }
}
```

```
static void read_stat_HPC_read_mostly_offset(struct sig_argqueue, \
    ppc_t)
{
    static void
    we_protected(long arg)
    {
        struct control_argqueue
        PVE_PARAM_ARG2(i, val) = get_state_state();
        set_val_val(readmost_longstate, current_state_ptr(),
            readmost_longstate_ptr, full, low)
    }
}
```

Crédit: A. Karpathy, CS231

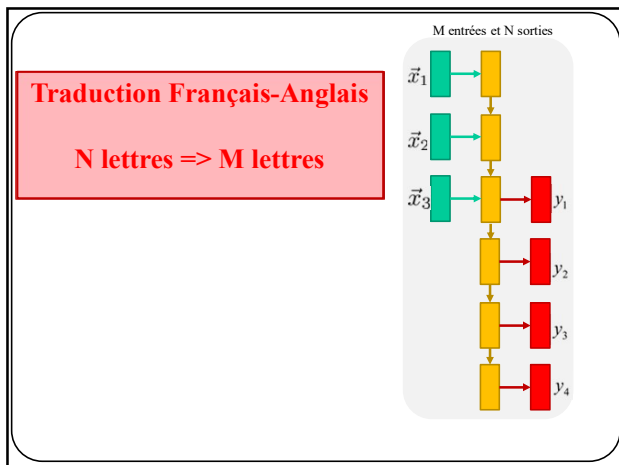
240

## Différentes configurations pour différentes applications



241





242

---

---

---

---

---

---

---

---

**Autre exemple: traduction**

Traduire 'assez' -> 'enough'  
 Alphabet fr: [**<BoS>**,a,e,s,z,<**EoS
 Alphabet en: [**<BoS>**,e,g,h,n,o,u,<**EoS

Pas le même nombre d'entrées  
que de sorties !  
 (BoS : Beginning of Sentence,  
 EoS:End of Sentence).

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.****

243

---

---

---

---

---

---

---

---

**Autre exemple: traduction**

Traduire 'assez' -> 'enough'  
 Alphabet fr: [**<BoS>**,a,e,s,z,<**EoS
 Alphabet en: [**<BoS>**,e,g,h,n,o,u,<**EoS

$x_1 = \text{<BoS>}$****

244

---

---

---

---

---

---

---

---

**Autre exemple: traduction**

Traduire 'assez' -> 'enough'  
 Alphabet fr : [**<BoS>**,a,e,s,z,<EoS>]  
 Alphabet en : [<BoS>,e,g,h,n,o,u,<EoS>]

$\vec{x}_1 = \text{<BoS>}$

$\vec{x}_2 = a$

245

---

---

---

---

---

---

---

---

**Autre exemple: traduction**

Traduire 'assez' -> 'enough'  
 Alphabet fr : [**<BoS>**,a,e,s,z,<EoS>]  
 Alphabet en : [<BoS>,e,g,h,n,o,u,<EoS>]

$\vec{x}_1 = \text{<BoS>}$

$\vec{x}_2 = a$

$\vec{x}_i = \dots$

$\vec{x}_7 = \text{<EoS>}$

246

---

---

---

---

---

---

---

---

**Autre exemple: traduction**

Traduire 'assez' -> 'enough'  
 Alphabet fr : [**<BoS>**,a,e,s,z,<EoS>]  
 Alphabet en : [<BoS>,e,g,h,n,o,u,<EoS>]

$\vec{x}_1 = \text{<BoS>}$

$\vec{x}_2 = a$

$\vec{x}_i = \dots$

$\vec{x}_7 = \text{<EoS>}$

$\vec{y}_1 = e$

$\vec{y}_i = \dots$

$\vec{y}_7 = h$

$\vec{y}_8 = \text{<EoS>}$

247

---

---

---

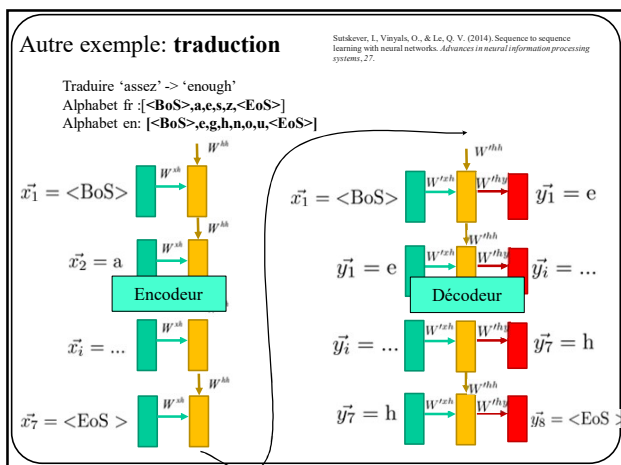
---

---

---

---

---



248

---

---

---

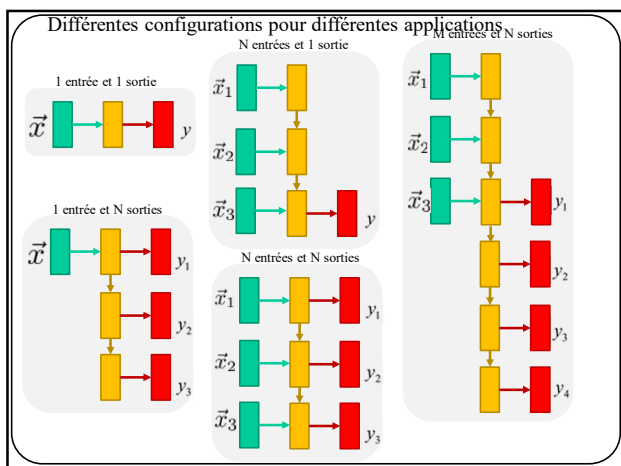
---

---

---

---

---



249

---

---

---

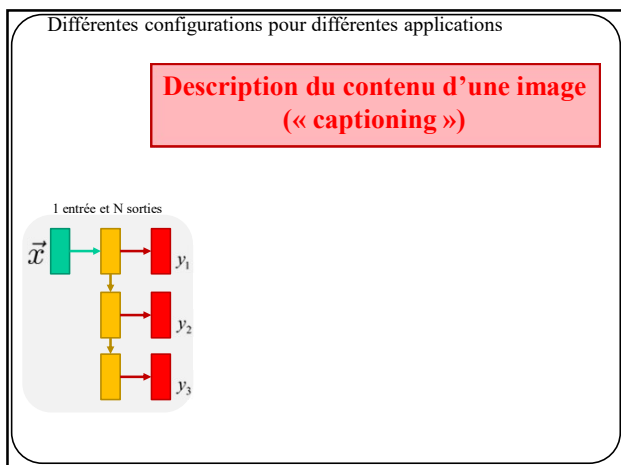
---

---

---

---

---



250

---

---

---

---


---

---

---

---

Captioning



input

3x3 conv, 64

3x3 conv, 64

Pool

3x3 conv, 128

3x3 conv, 128

Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

FC, 4096

FC, 4096

FC, 1000

Softmax

VGG16

251

251

---

---

---

---


---

---

---

---

Captioning



input

3x3 conv, 64

3x3 conv, 64

Pool

3x3 conv, 128

3x3 conv, 128

Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

FC, 4096

FC, 4096

FC, 1000

Softmax

VGG16

252

---

---

---

---


---

---

---

---

Captioning



input

3x3 conv, 64

3x3 conv, 64

Pool

3x3 conv, 128

3x3 conv, 128

Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

FC, 4096

FC, 4096

FC, 4096

$\vec{h}_0$

$x_1^* = \langle \text{BoS} \rangle$

$W^{ih}$

$W^{bh}$

Étape 1 : Init + Propagation avant

253

---

---

---

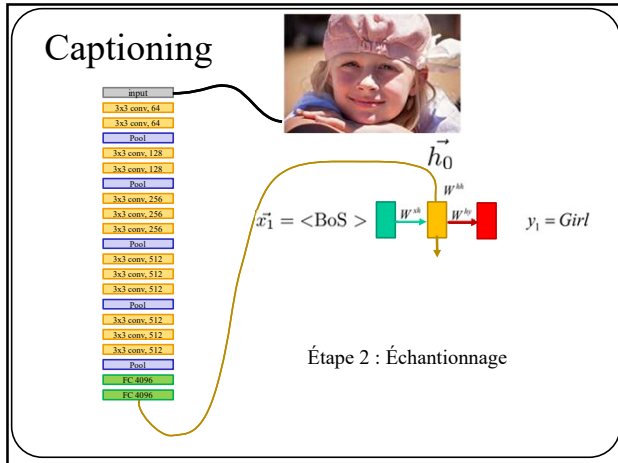
---

---

---

---

---



254

---

---

---

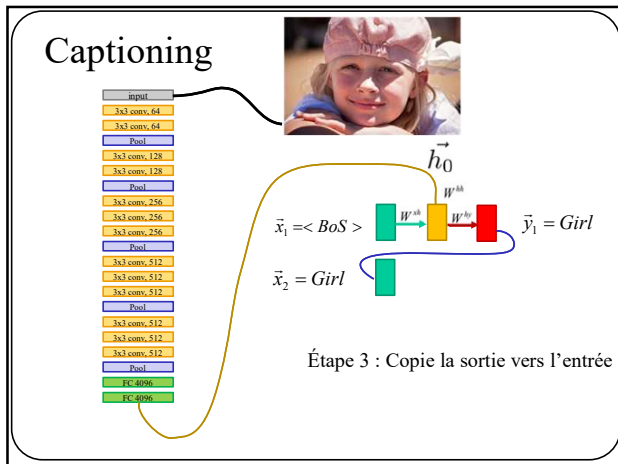
---

---

---

---

---



255

---

---

---

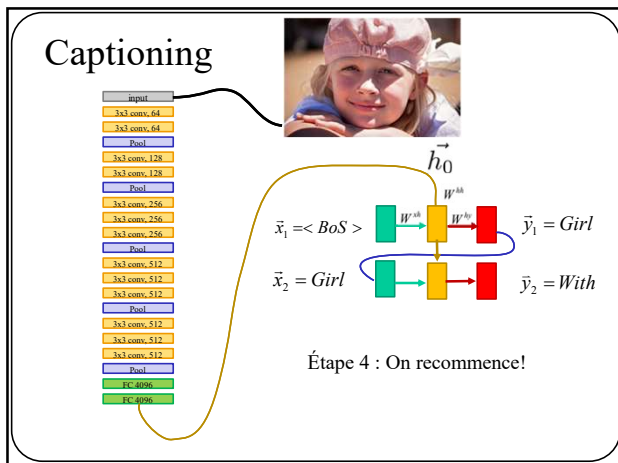
---

---

---

---

---



256

---

---

---

---

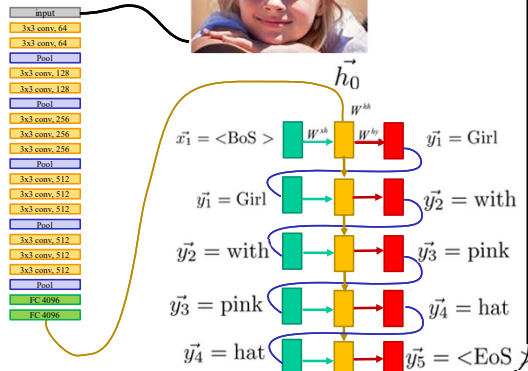
---

---

---

---

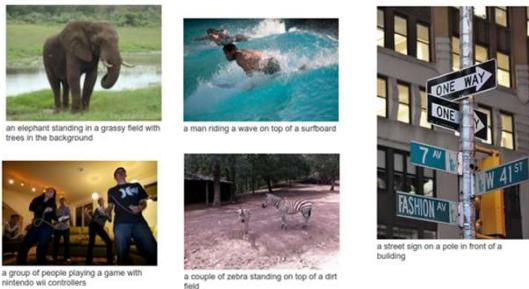
## Captioning



257

## Exemples de résultats

<https://github.com/karpathy/neuraltalk2>



258

## Exemples d'erreurs

<https://github.com/karpathy/neuraltalk2>



259

259