

Réseaux de neurones

IFT 603

Réseaux à convolution

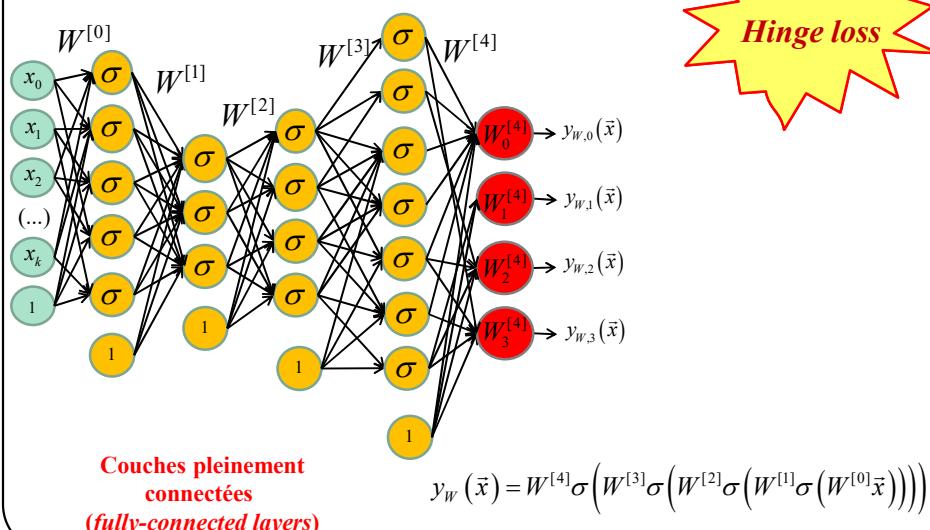
Par

Pierre-Marc Jodoin

1

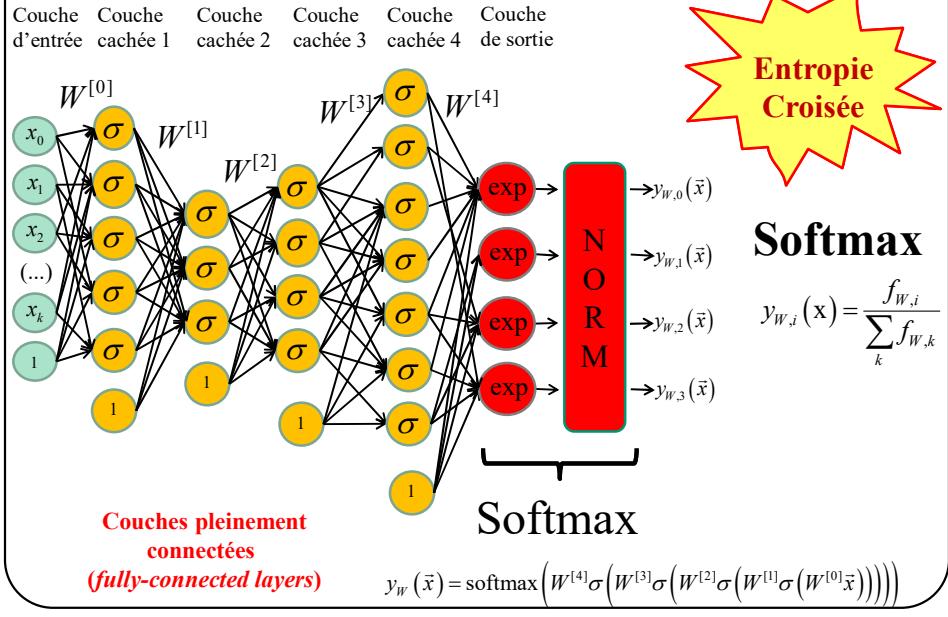
kD, 4 Classes, Réseau à 4 couches cachées

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



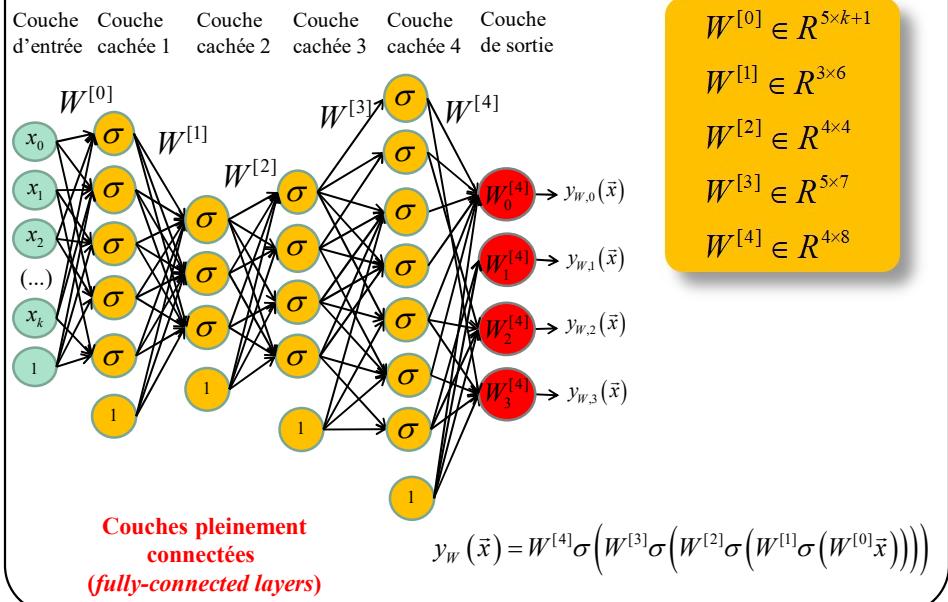
2

kD, 4 Classes, Réseau à 4 couches cachées



3

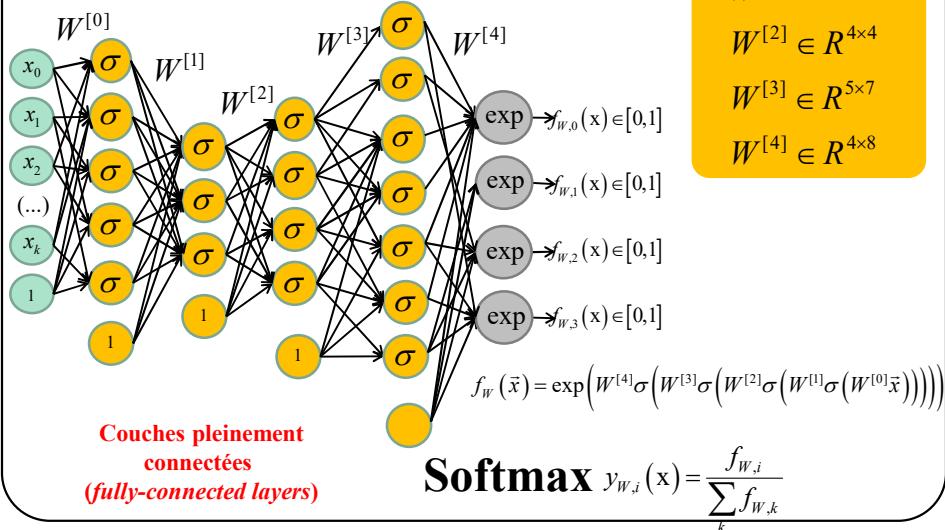
kD, 4 Classes, Réseau à 4 couches cachées



4

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



5

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".

Descente de gradient stochastique

```

Initialiser w
k=0
FAIRE k=k+1
    FOR n = 1 to N
        w = w - η[k] ∇E(̄xn)
    JUSQU'À ce que toutes les données
    soient bien classées ou k== MAX_ITER

```

Optimisation par Batch

```

Initialiser w
k=0
FAIRE k=k+1
    w = w - η[k] ∑i ∇E(̄xi)
    JUSQU'À ce que toutes les données
    soient bien classées ou k== MAX_ITER

```

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

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**/**Mini-batch size**/

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

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

} **Itération**

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* /**Mini-batch size**/

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

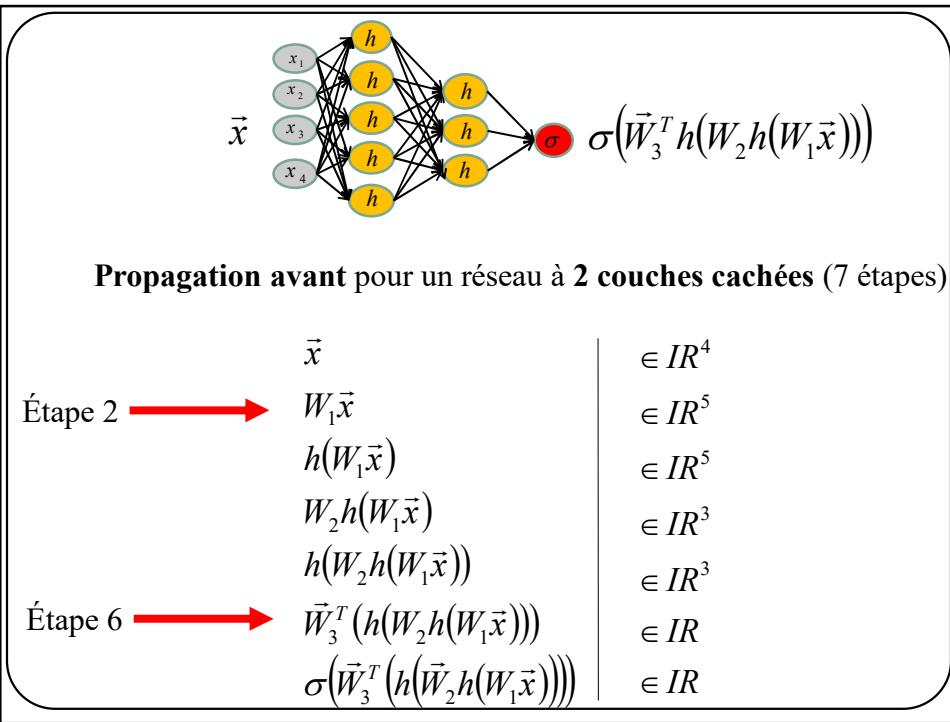
Epoch

JUSQU’À ce que toutes les données sont bien classées ou
 $k==\text{MAX_ITER}$

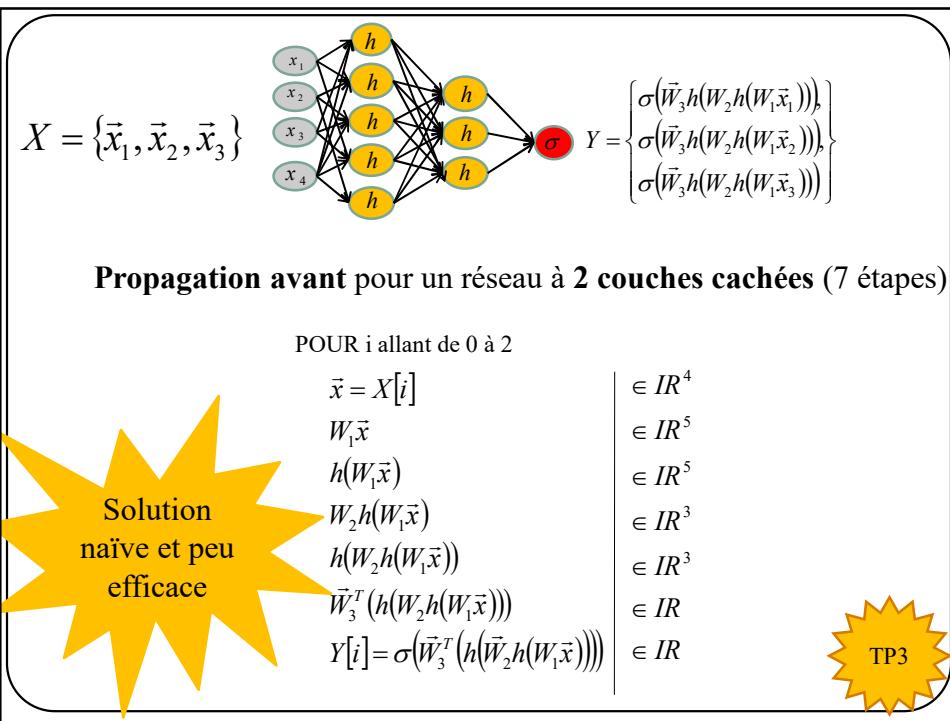
9

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

10



11



12

Solution

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

$$W_1 \vec{x}_1 = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{pmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix}$$

$$W_1 \vec{x}_2 = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{pmatrix} \begin{bmatrix} d \\ e \\ f \\ g \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix}$$

$$W_1 \vec{x}_3 = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \\ w_{51} & w_{52} & w_{53} & w_{54} \end{pmatrix} \begin{bmatrix} h \\ i \\ j \\ k \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix}$$

13

Solution

Il est plus efficace d'effectuer **UNE multiplication matricielle** que **PLUSIEURS matrice-vecteur** (exemple de la 2^e 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

$$\vec{W}_3^T(h(W_2 h(W_1 \vec{x})))$$

Solution

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

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

TROIS
produits
scalaires

15

Solution

Il est plus efficace d'effectuer **UNE multiplication matricielle** que **PLUSIEURS produits scalaires** (**exemple de la 6^e 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

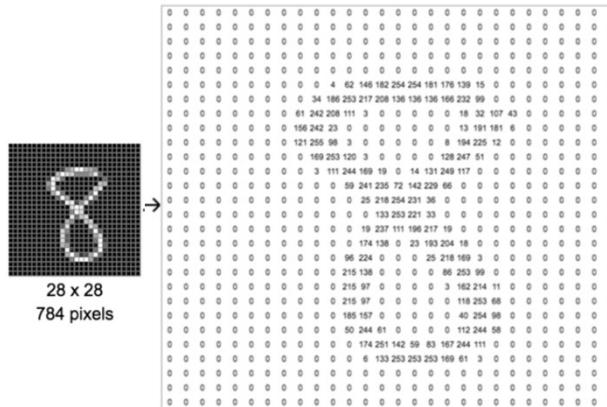
100% du temps, on combine ensemble les données
dans des **mini-batch** de 2 à 32 données.

17

Les réseaux à convolution

18

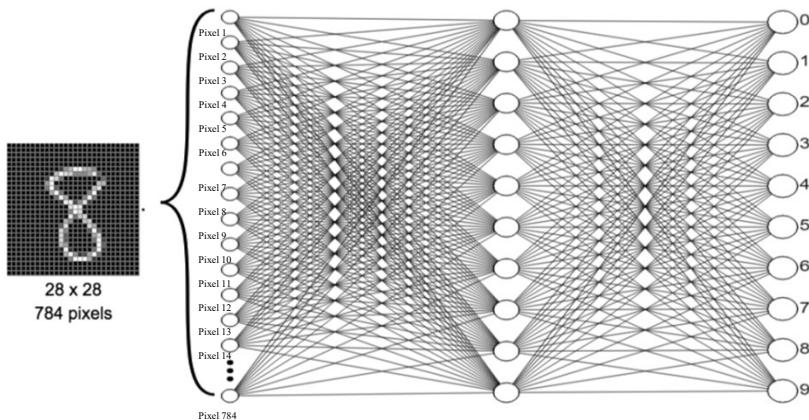
Comment classifier des images?



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

19

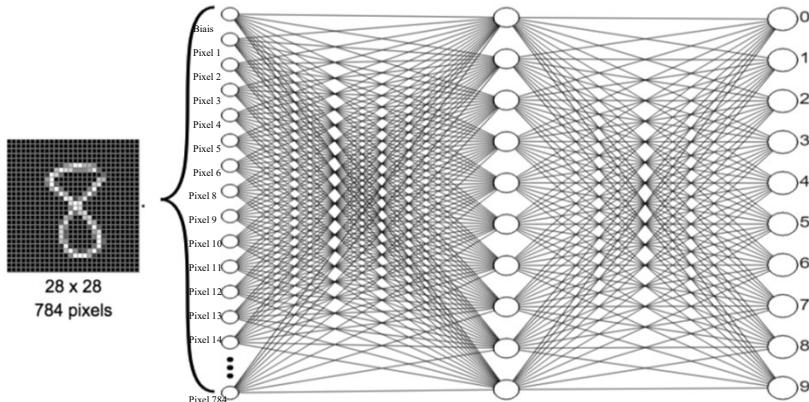
Comment classifier des images?



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

20

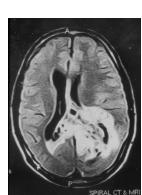
Beaucoup de paramètres (7850 dans la couche 1)



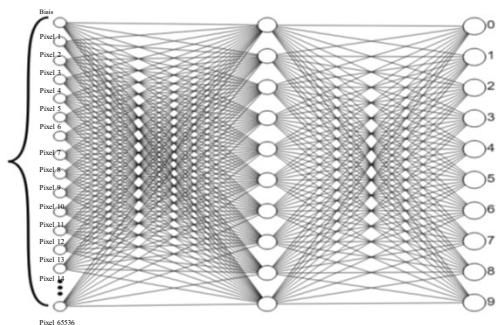
https://ml4a.github.io/ml4a/neural_networks/

21

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



256x256



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

Image médicale (IRM de cerveau)

22

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

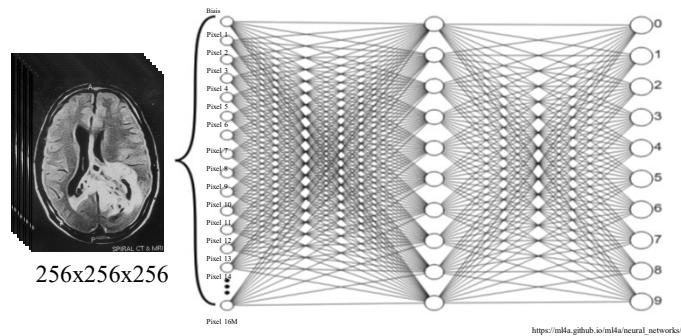


Image médicale 3D (IRM de cerveau)

23

Comment réduire le nombre de connections?

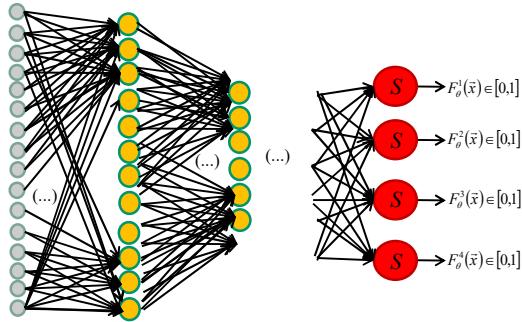


24

24

Comment réduire le nombre de connections?

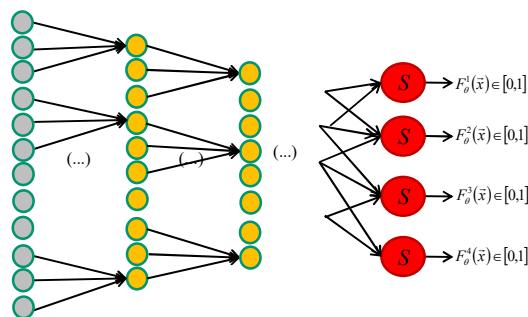
Les **couches pleinement connectées (fully-connected layers)** sont problématiques lorsque le **nombre de neurones** est élevé.



150-D en entrée avec 150 neurones dans la 1ère couche => 22,200 paramètres dans la couche d'entrée!!

25

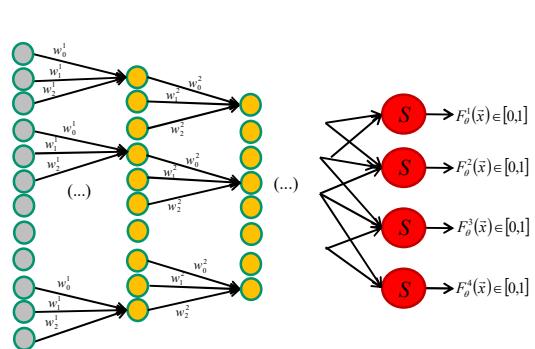
Solution : connexions partielles



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**



Convolution

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

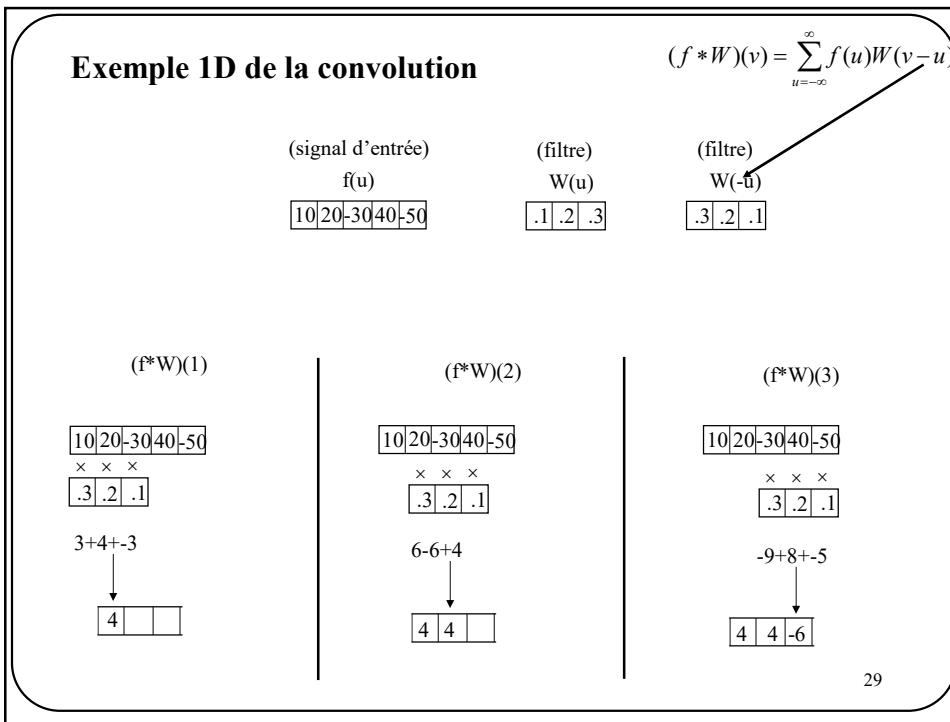
Faible nombre de paramètres = on peut augmenter la profondeur!

27

Convolution
et
couche convolutionnelle
1D

28

14



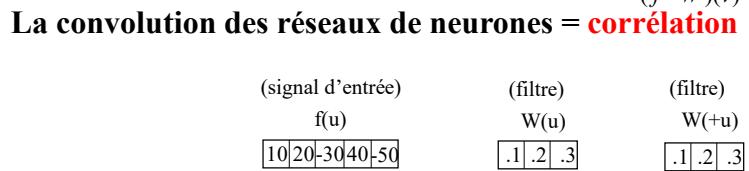
29

En gros

convolution = produit scalaire + translation

30

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

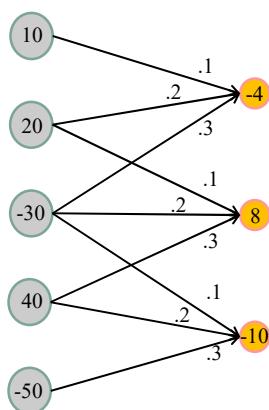


(f*W)(1)	(f*W)(2)	(f*W)(3)																																													
<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>10</td><td>20</td><td>-30</td><td>40</td><td>-50</td></tr><tr><td>x</td><td>x</td><td>x</td><td></td><td></td></tr><tr><td>.1</td><td>.2</td><td>.3</td><td></td><td></td></tr></table>	10	20	-30	40	-50	x	x	x			.1	.2	.3			<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>10</td><td>20</td><td>-30</td><td>40</td><td>-50</td></tr><tr><td>x</td><td>x</td><td>x</td><td></td><td></td></tr><tr><td>.1</td><td>.2</td><td>.3</td><td></td><td></td></tr></table>	10	20	-30	40	-50	x	x	x			.1	.2	.3			<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>10</td><td>20</td><td>-30</td><td>40</td><td>-50</td></tr><tr><td>x</td><td>x</td><td>x</td><td></td><td></td></tr><tr><td>.1</td><td>.2</td><td>.3</td><td></td><td></td></tr></table>	10	20	-30	40	-50	x	x	x			.1	.2	.3		
10	20	-30	40	-50																																											
x	x	x																																													
.1	.2	.3																																													
10	20	-30	40	-50																																											
x	x	x																																													
.1	.2	.3																																													
10	20	-30	40	-50																																											
x	x	x																																													
.1	.2	.3																																													
1+4-9 \downarrow <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>-4</td><td></td><td></td></tr></table>	-4			2-6+12 \downarrow <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>4</td><td>8</td><td></td></tr></table>	4	8		-3+8-15 \downarrow <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>-4</td><td>8</td><td>-10</td></tr></table>	-4	8	-10																																				
-4																																															
4	8																																														
-4	8	-10																																													

31

31

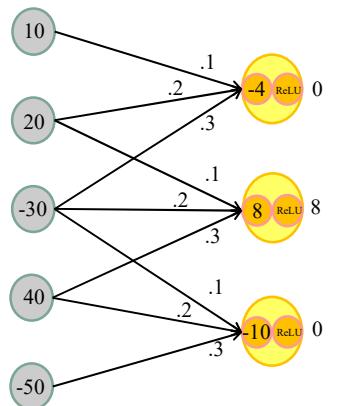
L'opération de la page précédente est équivalente à



32

32

L'opération de la page précédente est équivalente à

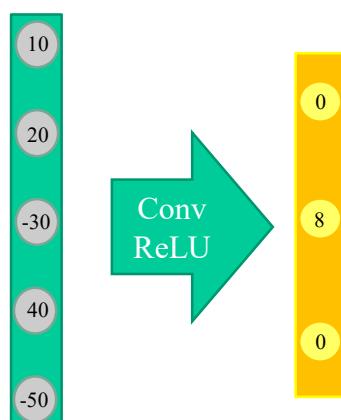


Fonction
d'activation
(ex. ReLU)

33

33

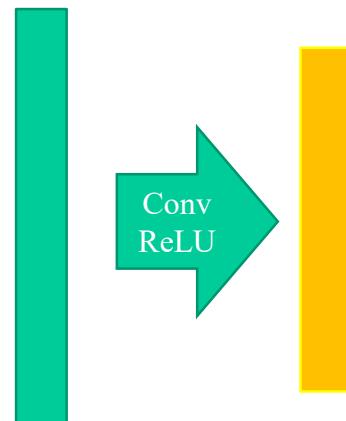
Représentation graphique courante (simple)



34

34

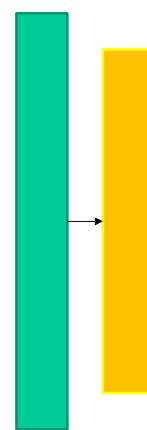
Représentation graphique courante (encore plus simple)



35

35

Représentation graphique courante (vraiment ultra simple)



36

36

Représentation graphique courante (eehhh...)

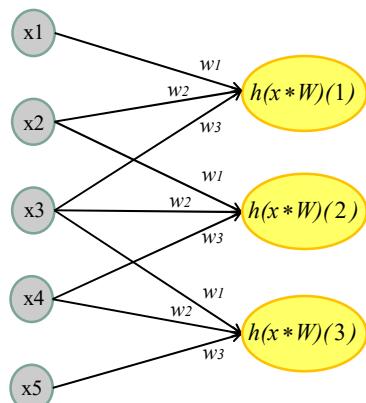
Vecteur d'entrée

Carte d'activation
(*feature map*)

37

37

Apprentissage = apprendre les **poids w_i** des filtres convolutifs

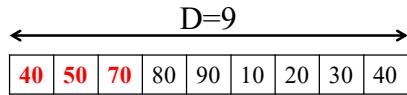


h : fonction d'activation
(tanh, ReLU, elu, etc.)

38

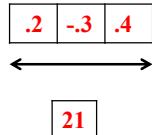
38

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



prod. scal.

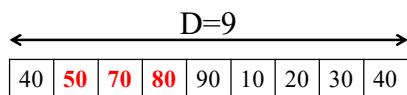
$F=3$



39

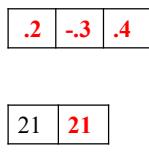
39

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



prod. scal.

$F=3$

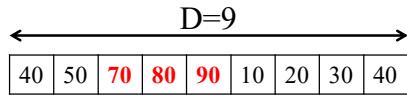


Stride = 1

40

40

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



prod. scal.

$F=3$

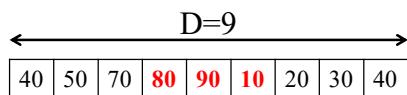
.2	-3	.4
-----------	-----------	-----------

21	21	26
----	----	-----------

41

41

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



prod. scal.

$F=3$

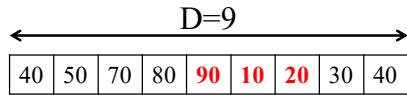
.2	-3	.4
-----------	-----------	-----------

21	21	26	-7
----	----	----	-----------

42

42

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



prod. scal.

$F=3$

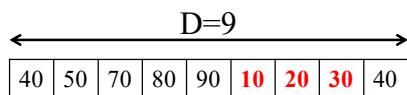
.2	-.3	.4
----	-----	----

21	21	26	-7	23
----	----	----	----	----

43

43

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



prod. scal.

$F=3$

.2	-.3	.4
----	-----	----

21	21	26	-7	23	8
----	----	----	----	----	---

44

44

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

D=9								
40	50	70	80	90	10	20	30	40

prod. scal.

F=3

.2	-.3	.4
----	-----	----

21	21	26	-7	23	8	11
----	----	----	----	----	---	----

Taille de la carte d'activation = 7

45

45

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

D=9									
40	50	70	80	90	10	20	30	40	

prod. scal.

F=5

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

35

46

46

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

D=9									
40	50	70	80	90	10	20	30	40	

prod. scal.

F=5

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

35	-18
----	-----

Stride = 1

47

47

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

D=9									
40	50	70	80	90	10	20	30	40	

prod. scal.

F=5

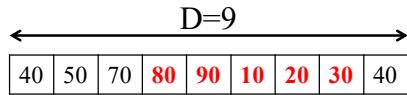
.2	-3	.4	-5	.6
----	----	----	----	----

35	-18	33
----	-----	----

48

48

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



prod. scal.

$F=5$

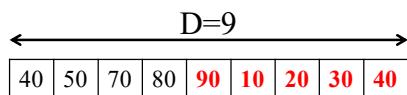
.2	-.3	.4	-.5	.6
----	-----	----	-----	----

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

49

49

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



prod. scal.

$F=5$

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

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

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

50

50

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

D=9								
40	50	70	80	90	10	20	30	40

prod. scal.

F=7

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

44

51

51

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

D=9								
40	50	70	80	90	10	20	30	40

prod. scal.

F=7

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

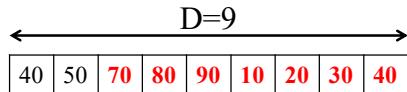
44 -8

Stride = 1

52

52

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



prod. scal.

$F=7$

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

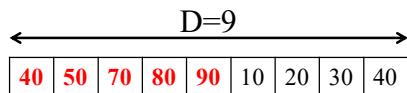
44	-8	44
----	----	----

Taille de la carte d'activation = **3**

53

53

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



prod. scal.

$F=5$

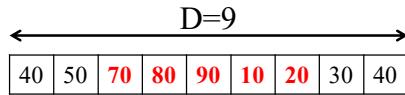
.2	-.3	.4	-.5	.6
----	-----	----	-----	----

35

54

54

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



prod. scal.

$F=5$

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

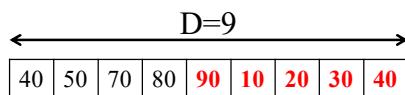
35	33
----	-----------

Stride = 2

55

55

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



prod. scal.

$F=5$

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

35	33	32
----	----	-----------

Taille de la carte d'activation = **3**

56

56

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

D=9									
40	50	70	80	90	10	20	30	40	

prod. scal.

F=5

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

35

57

57

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

D=9									
40	50	70	80	90	10	20	30	40	

prod. scal.

F=5

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

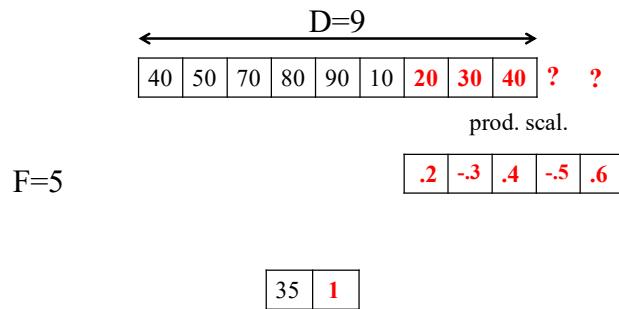
35 1

Stride = 3

58

58

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



ERREUR! Combinaison D-F-S invalide

59

59

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

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



60

60

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

$$? \boxed{10|20|30|40|50} \\ \times \quad \times \quad \times$$

Comment gérer les bords?

$$\boxed{.1|.2|.3}$$

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

$$\begin{array}{c} f(u) \\ \hline \boxed{0|10|20|30|40|50|0} \end{array}$$

$$\begin{array}{c} (f^*W)(u) \\ \hline \boxed{8|-4|8|10|-6} \end{array}$$

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

$$\begin{array}{c} f(u) \\ \hline \boxed{20|10|20|30|40|50|40} \end{array}$$

$$\begin{array}{c} (f^*W)(u) \\ \hline \boxed{10|-4|8|10|2} \end{array}$$

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

$$\begin{array}{c} f(u) \\ \hline \boxed{10|10|20|30|40|50|50} \end{array}$$

$$\begin{array}{c} (f^*W)(u) \\ \hline \boxed{9|-4|8|10|-21} \end{array}$$

61

61

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

$$? \boxed{10|20|30|40|50} \\ \times \quad \times \quad \times$$

Comment gérer les bords?

$$\boxed{.1|.2|.3}$$

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

$$\begin{array}{c} f(u) \\ \hline \boxed{0|10|20|30|40|50|0} \end{array}$$

$$\begin{array}{c} (f^*W)(u) \\ \hline \boxed{8|-4|8|10|-6} \end{array}$$

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

De loin l'option la plus utilisée

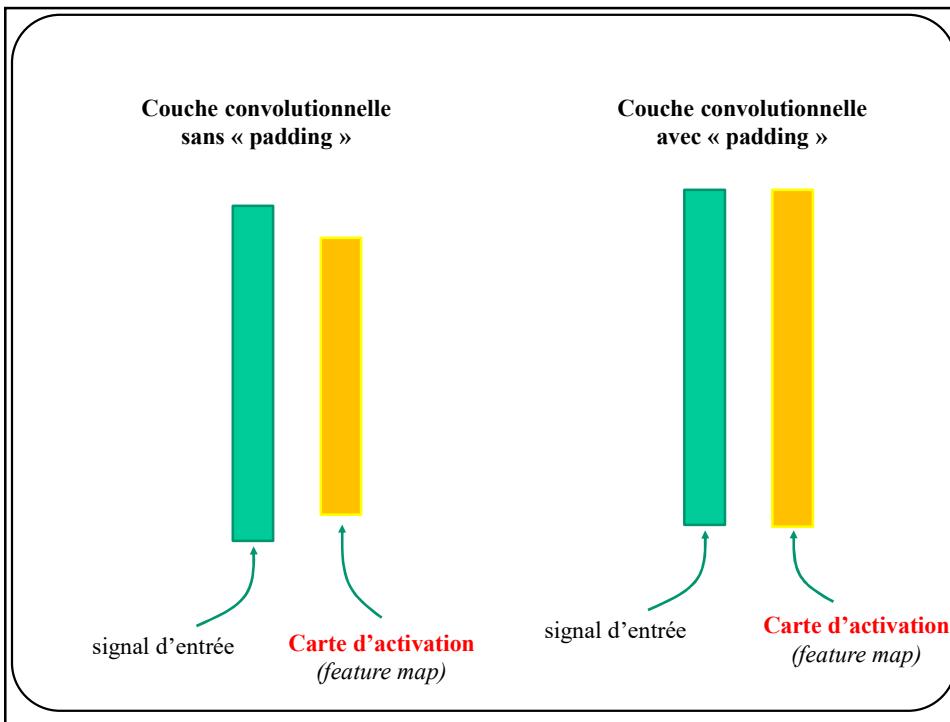
$$\begin{array}{c} (f^*W)(u) \\ \hline \boxed{-10|2} \end{array}$$

$$\begin{array}{c} f(u) \\ \hline \boxed{10|10|20|30|40|50|50} \end{array}$$

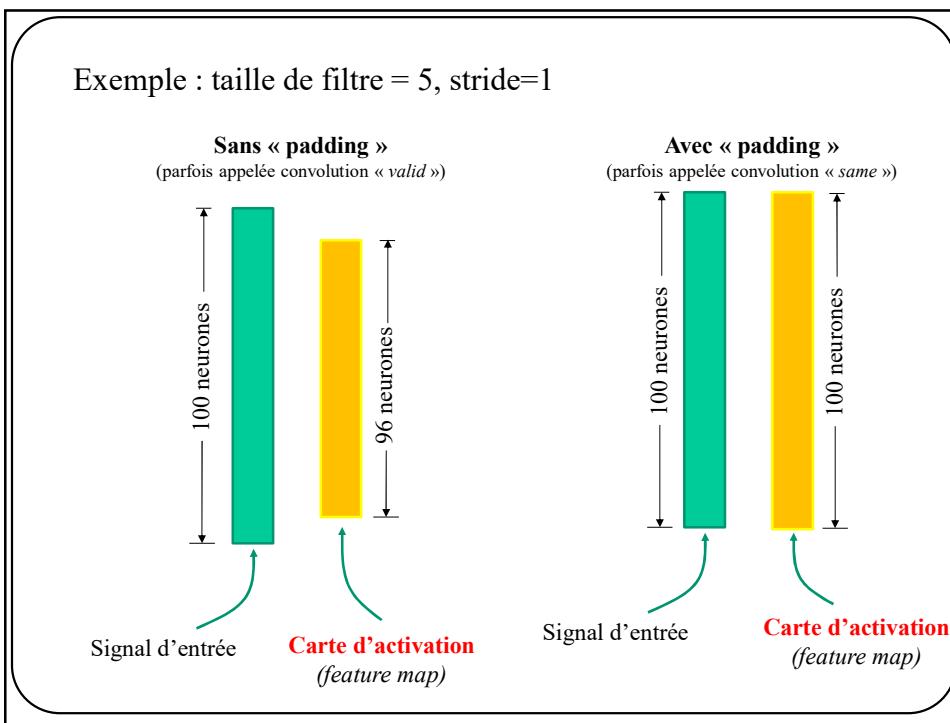
$$\begin{array}{c} (f^*W)(u) \\ \hline \boxed{9|-4|8|10|-21} \end{array}$$

62

62



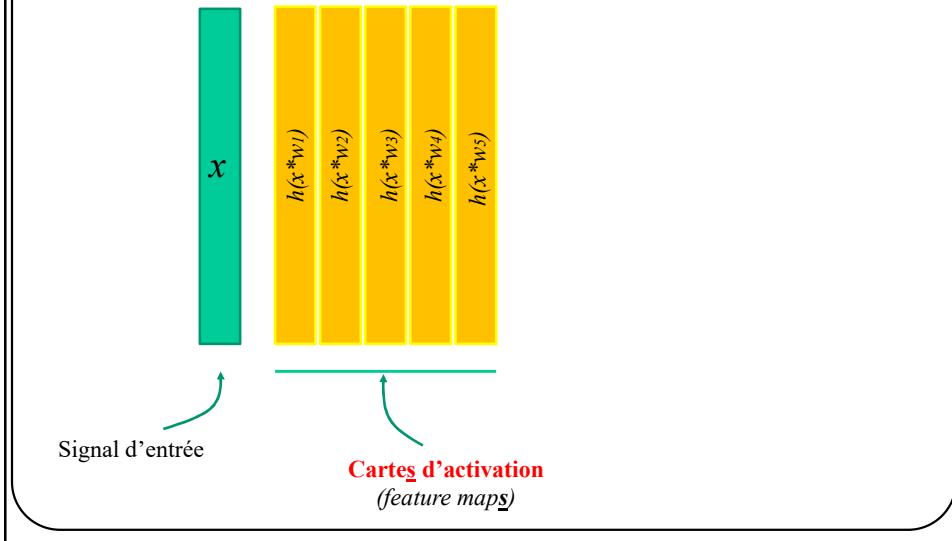
63



64

Il est possible d'apprendre **plusieurs filtres par couche**

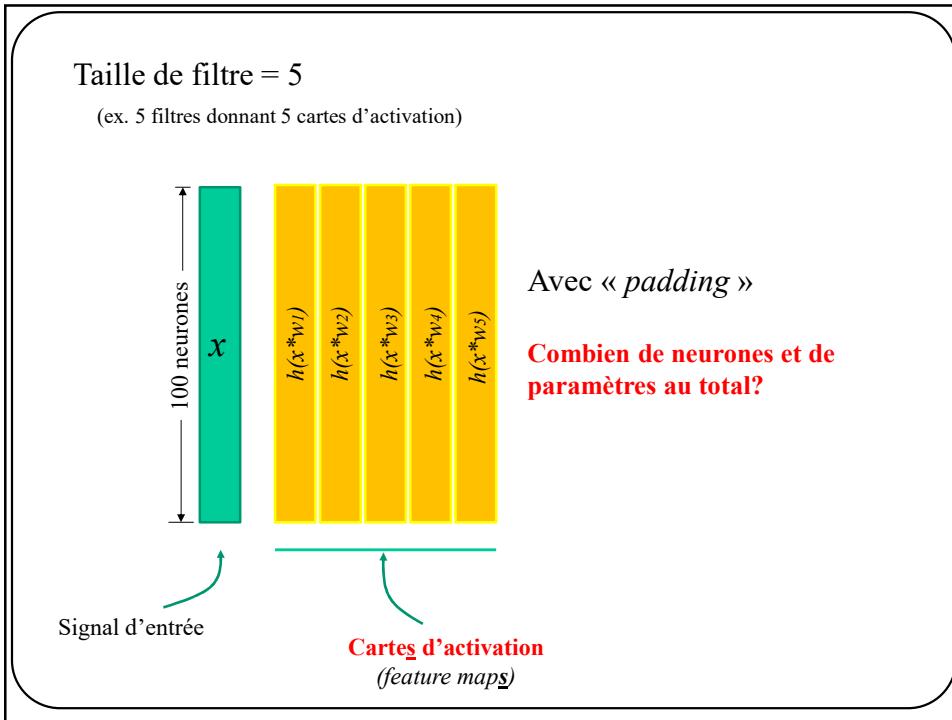
(ex. 5 filtres donnant 5 cartes d'activation)



65

Taille de filtre = 5

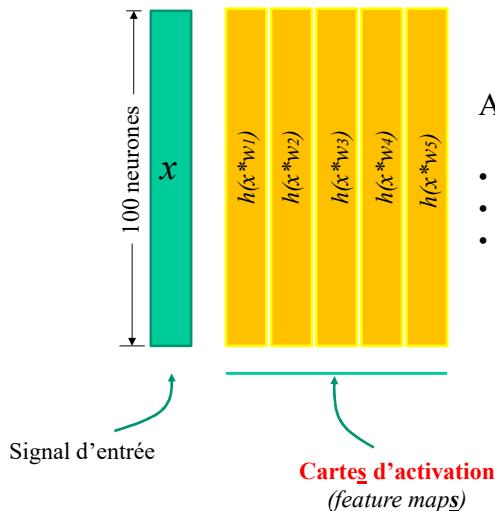
(ex. 5 filtres donnant 5 cartes d'activation)



66

Taille de filtre = 5

(ex. 5 filtres donnant 5 cartes d'activation)



Avec « *padding* »

- 100 neurones par carte d'activation
- 500 neurones au total
- 25 (5×5) paramètres au total

67

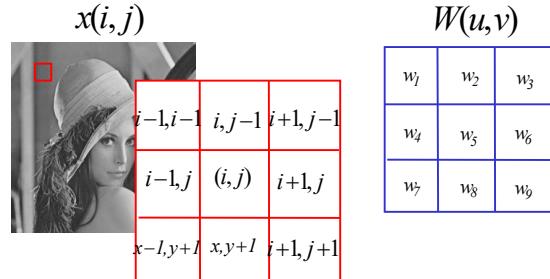
Convolution et couche convolutionnelle 2D

68

Filtage 2D

(sans flip de filtre)

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

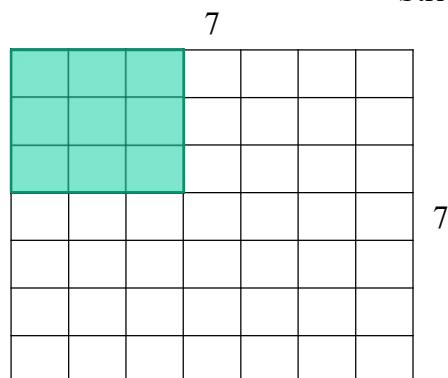


$$\begin{aligned} (x * W)(i, j) = & w_1 x(i-1, j-1) + w_2 x(i, j-1) + w_3 x(i+1, j-1) \\ & + w_4 x(i-1, j) + w_5 x(i, j) + w_6 x(i+1, j) \\ & + w_7 x(i-1, j+1) + w_8 x(i, j+1) + w_9 x(i+1, j+1) \end{aligned}$$

69

Convolution 2D

Filtre = 3x3
Stride = 1

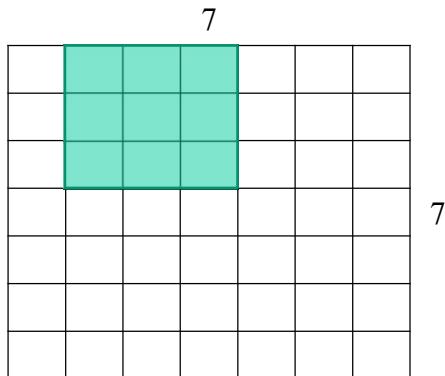


70

Convolution 2D

Filtre = 3x3

Stride = 1

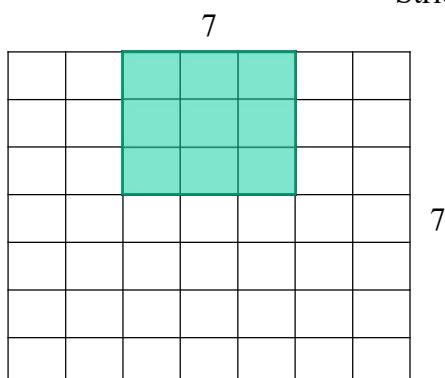


71

Convolution 2D

Filtre = 3x3

Stride = 1

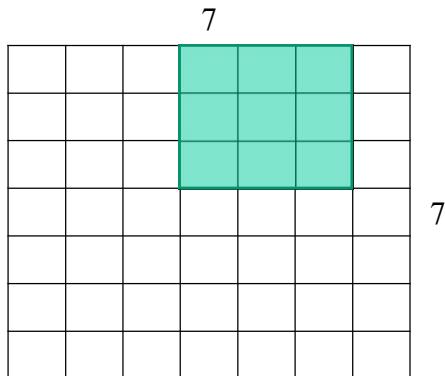


72

Convolution 2D

Filtre = 3x3

Stride = 1

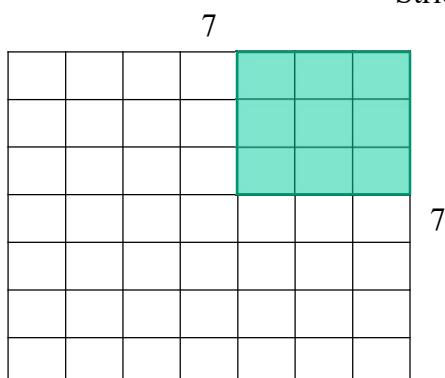


73

Convolution 2D

Filtre = 3x3

Stride = 1



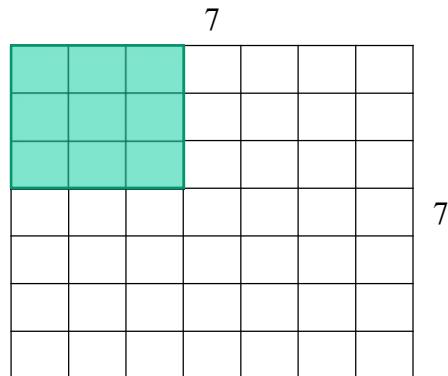
Taille de la carte d'activation (pour stride 1) = **5x5**

74

Convolution 2D

Filtre = 3x3

Stride = 2

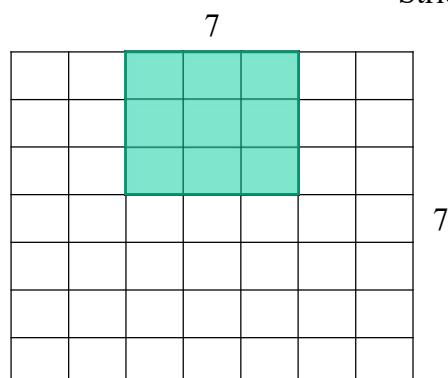


75

Convolution 2D

Filtre = 3x3

Stride = 2

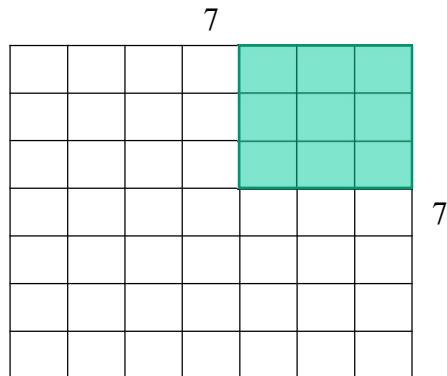


76

Convolution 2D

Filtre = 3x3

Stride = 2



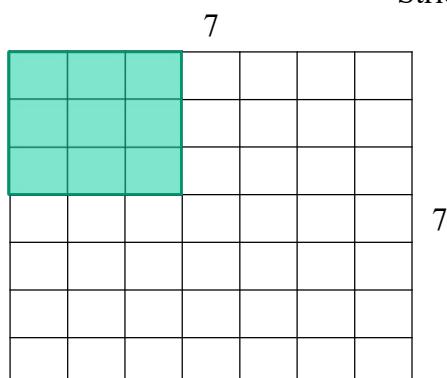
Taille de la carte d'activation (pour stride 2) = **3x3**

77

Convolution 2D

Filtre = 3x3

Stride = 3

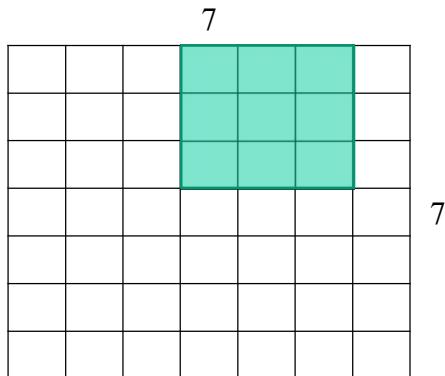


78

Convolution 2D

Filtre = 3x3

Stride = 3

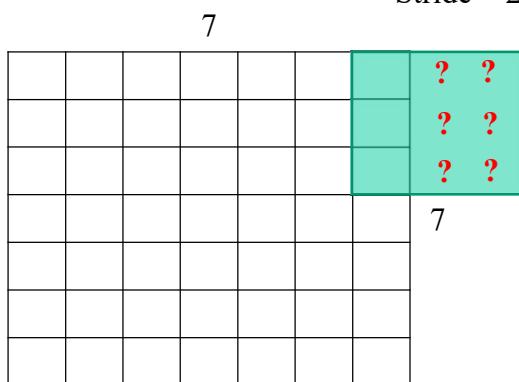


79

Convolution 2D

Filtre = 3x3

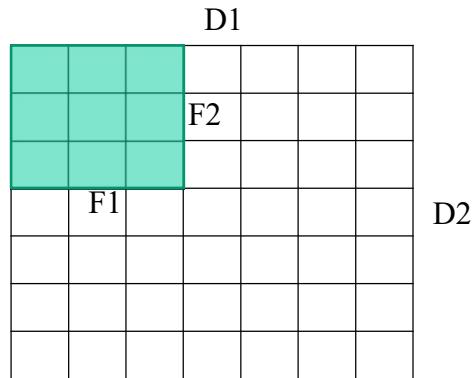
Stride = 2



Combinaison D-F-S invalide!

80

Convolution 2D

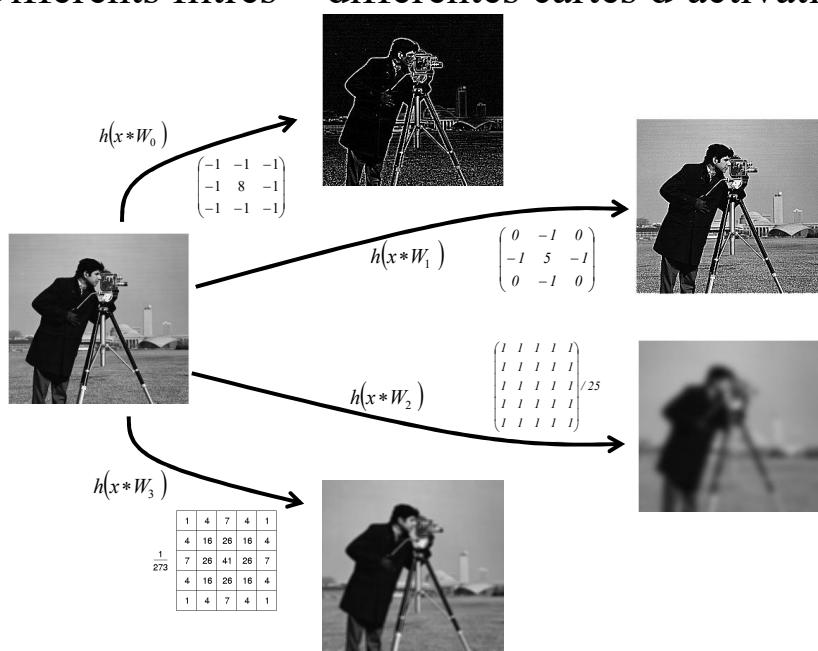


Taille de la carte d'activation :

$$(D1-F1)/S+1 \times (D2-F2)/S+1$$

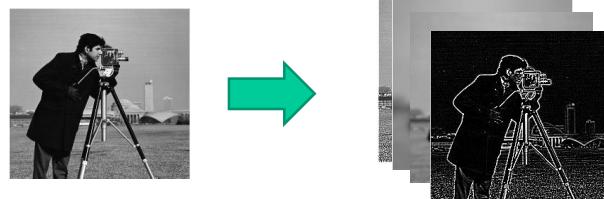
81

Différents filtres = différentes cartes d'activation



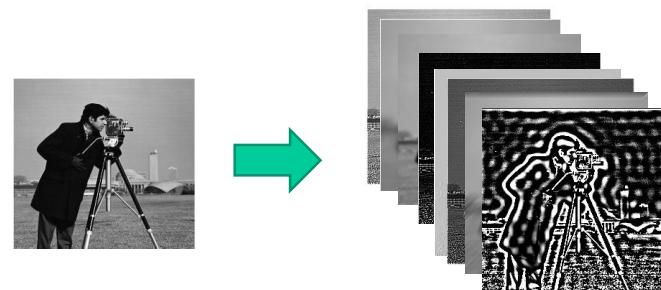
82

4 filtres = Couche convulsive avec 4 cartes d'activation



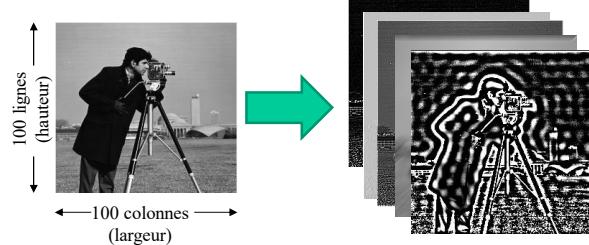
83

K filtres = Couche convulsive avec K cartes d'activation



84

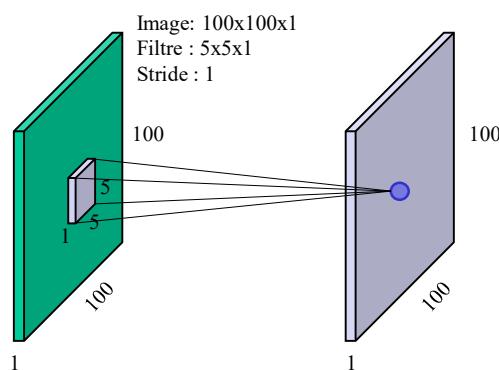
Ex.: taille de filtre : 5x5, 5 cartes d'activation, convolution « same »



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

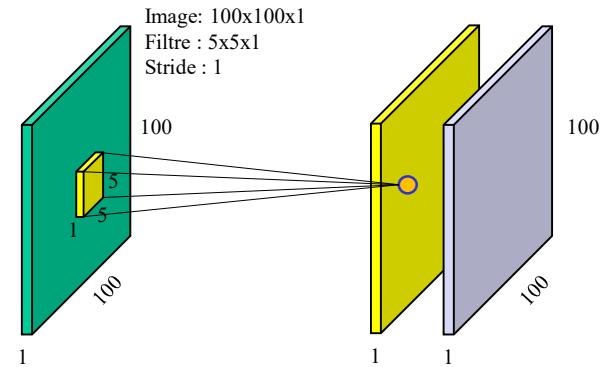
85

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



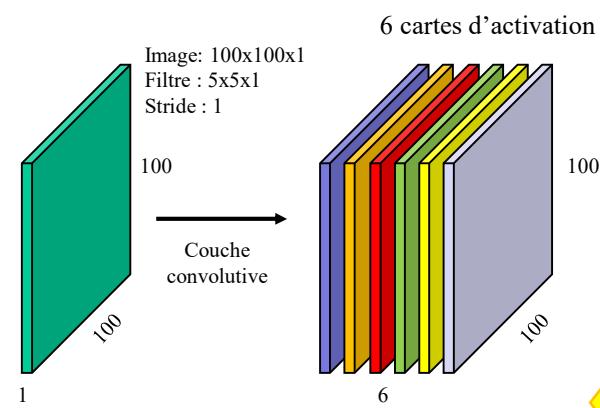
86

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



87

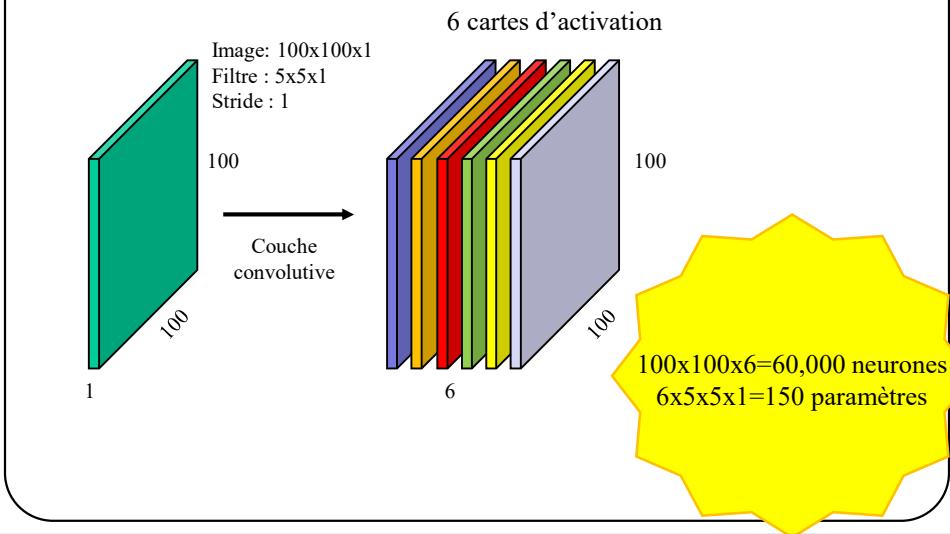
Représentation schématique
(6 filtres et 6 cartes d'activation, convolution « *same* »)



Combien de neurones
et de paramètres
au total?

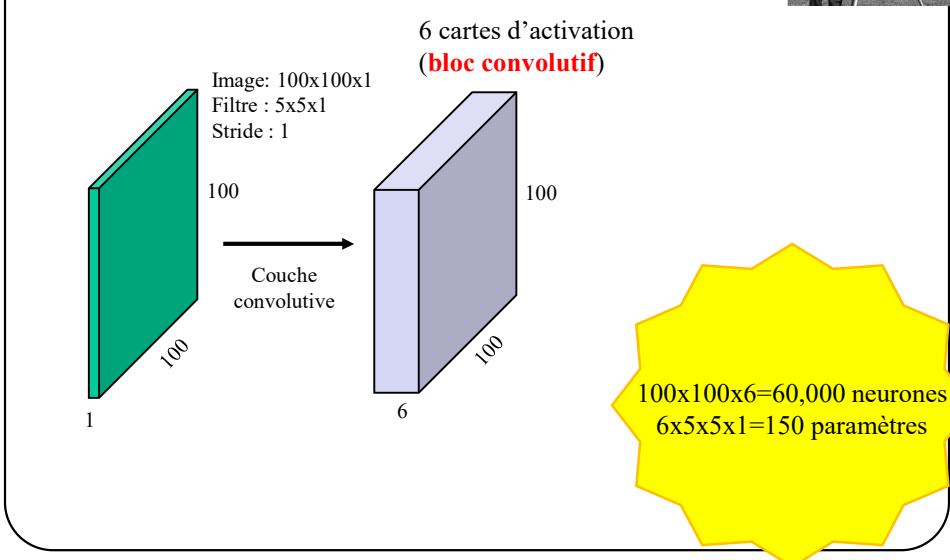
88

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



89

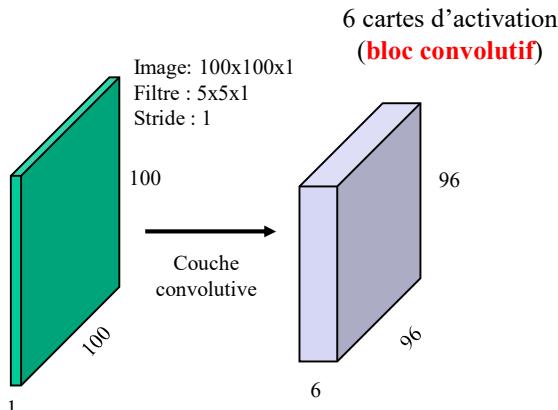
Représentation schématique simplifiée
(6 filtres et 6 cartes d'activation, convolution « *same* »)



90



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

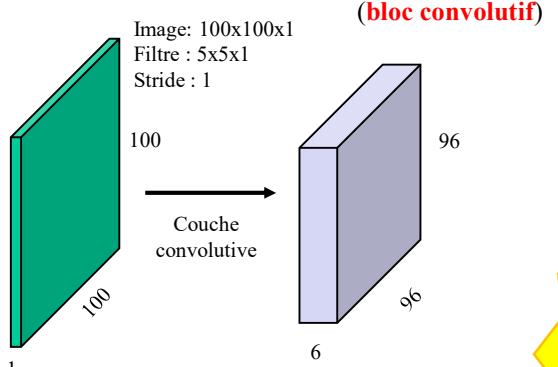


Combien de neurones
et de paramètres
au total?

91



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



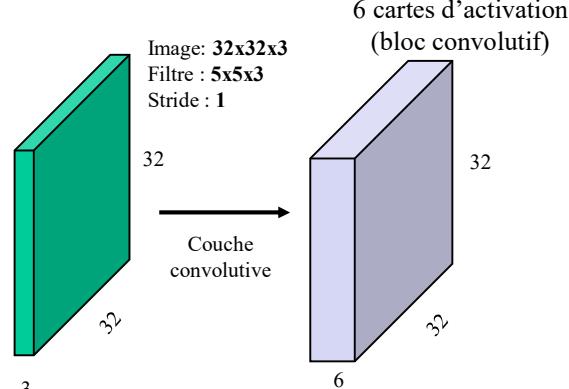
$$96 \times 96 \times 6 = 55,296 \text{ neurones}$$
$$6 \times 5 \times 5 \times 1 = 150 \text{ paramètres}$$

92

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



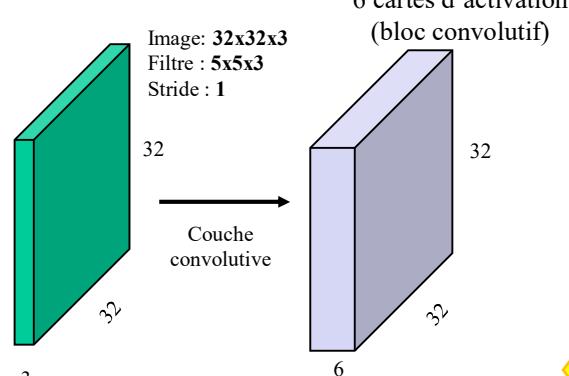
Exemples cifar10



Combien de neurones et de paramètres au total?

93

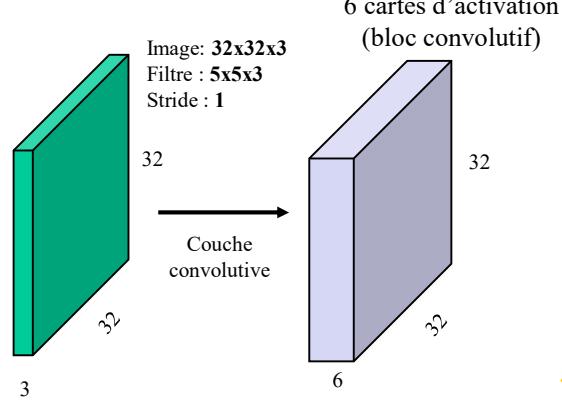
Représentation schématique **images couleur**
(ex.: images RGB de CIFAR10
convolution « same »)



$32 \times 32 \times 6 = 6,144$ neurones
 $6 \times 5 \times 5 \times 3 = 450$ paramètres

94

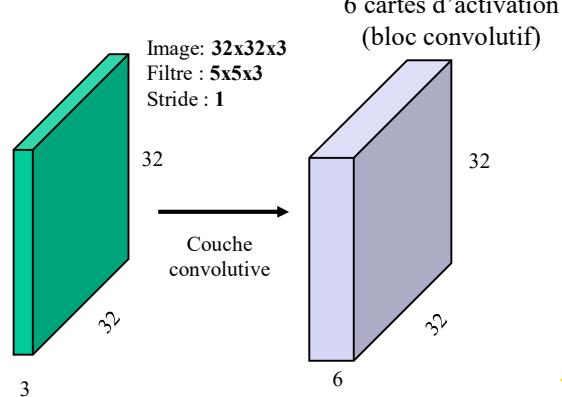
Représentation schématique **images couleur**
(ex.: images RGB de CIFAR10
convolution « same »)



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

95

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



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

96

Représentation schématique **images couleur**
(ex.: images RGB de CIFAR10
convolution « valid »)

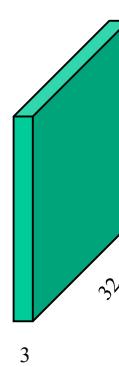
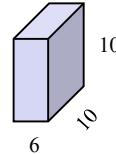


Image: $32 \times 32 \times 3$
Filtre : $5 \times 5 \times 3$
Stride : 3

32

Couche convective

6 cartes d'activation
(bloc convolutif)

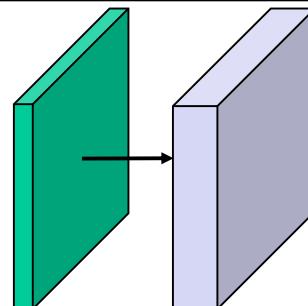


$10 \times 10 \times 6 = 600$ neurones
 $6 \times 5 \times 5 \times 3 = 450$ paramètres

97

Exemple

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



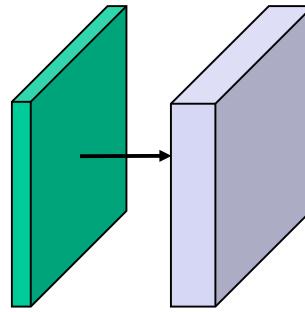
Combien y a-t-il de paramètres dans cette couche?

98

Exemple

Volume en entrée : $32 \times 32 \times 3$

10 filtres 5x5 avec stride = 1
et convolution « *same* »



Combien y a-t-il de paramètres dans cette couche?

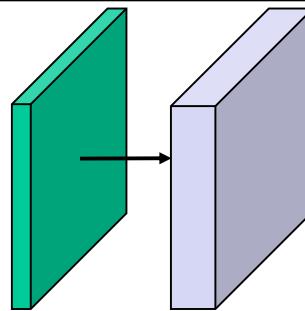
Chaque filtre a $5 \times 5 \times 3 = 75$ paramètres
Comme il y a **10 filtres** : **750** paramètres

99

Exemple

Volume en entrée : $32 \times 32 \times 3$

10 filtres 5x5 avec stride = 1
et convolution « *same* ».



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

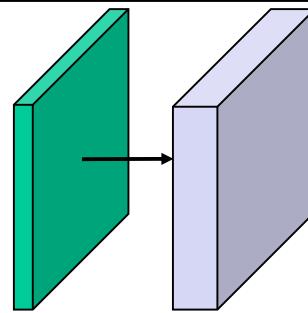
Chaque filtre a $5 \times 5 \times 3 + 1 = 76$ paramètres (+1 pour le biais)
Comme il y a **10 filtres** : **760** paramètres

100

Exemple

Volume en entrée : $32 \times 32 \times 3$

10 filtres 5x5 avec stride = 1
et convolution « *valid* »



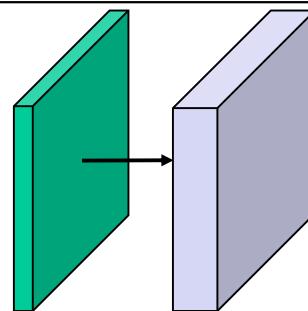
Combien de paramètres dans cette couche?

101

Exemple

Volume en entrée : $32 \times 32 \times 3$

10 filtres 5x5 avec stride = 1
et convolution « *valid* »



Combien de paramètres dans cette couche?

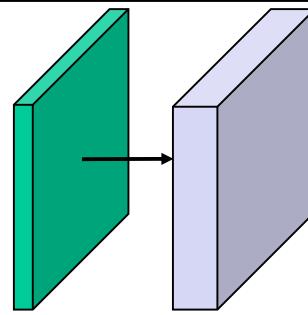
Même chose, cela ne change pas la conformité des filtres

102

Exemple

Volume en entrée : $32 \times 32 \times 3$

10 filtres 5x5 avec stride = 1
et convolution « *valid* »



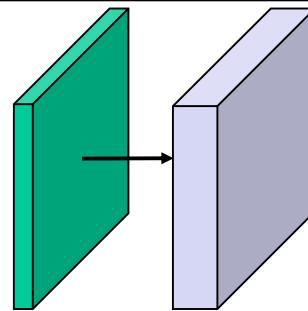
Combien de **neurones** dans les cartes d'activations?

103

Exemple

Volume en entrée : $32 \times 32 \times 3$

10 filtres 5x5 avec stride = 1
et convolution « *valid* »

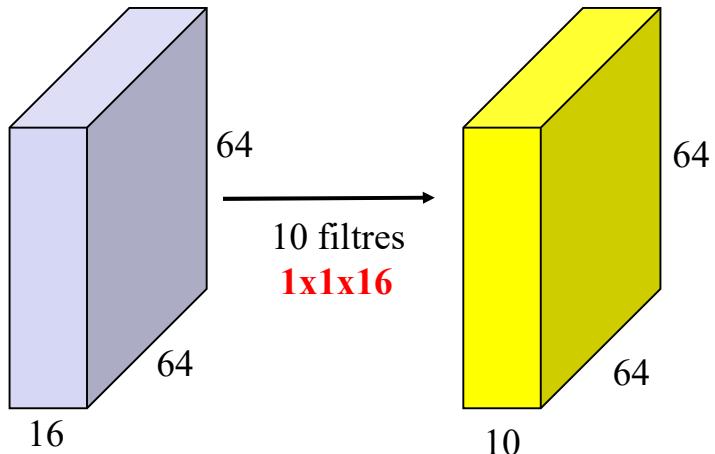


Combien de **neurones** dans les cartes d'activations?

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

104

Des filtres 1x1? Oui ça marche



105

Exemple simple d'un filtre 1x1



$$\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right]$$

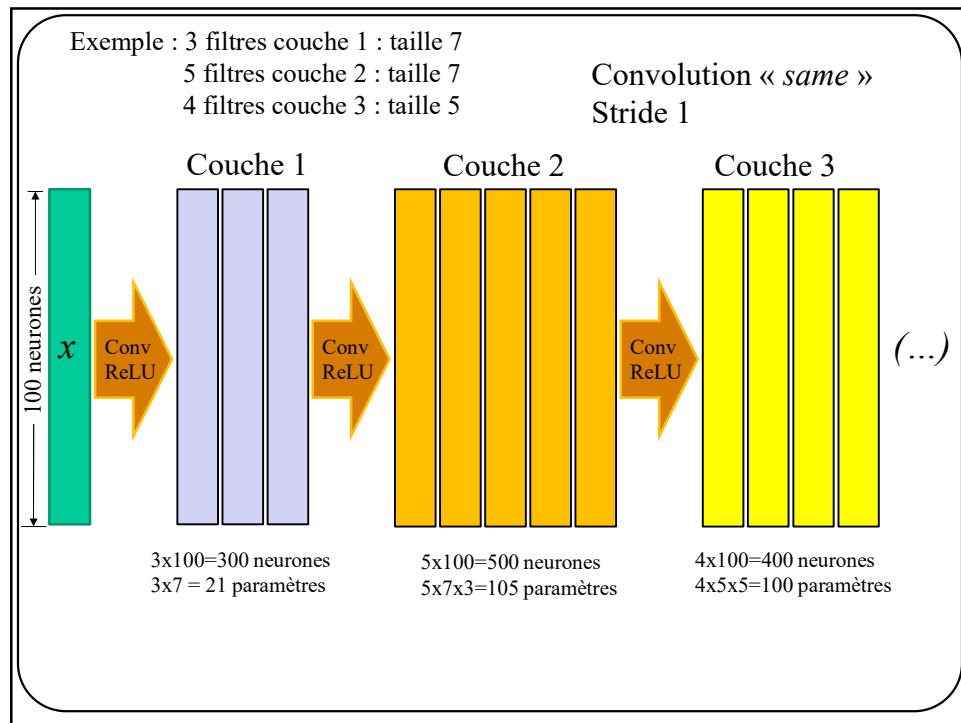


Filtre moyennant les canaux **rouge**, **vert**, **bleu** d'une image couleur.
Résultat, une image en **niveau de gris**.

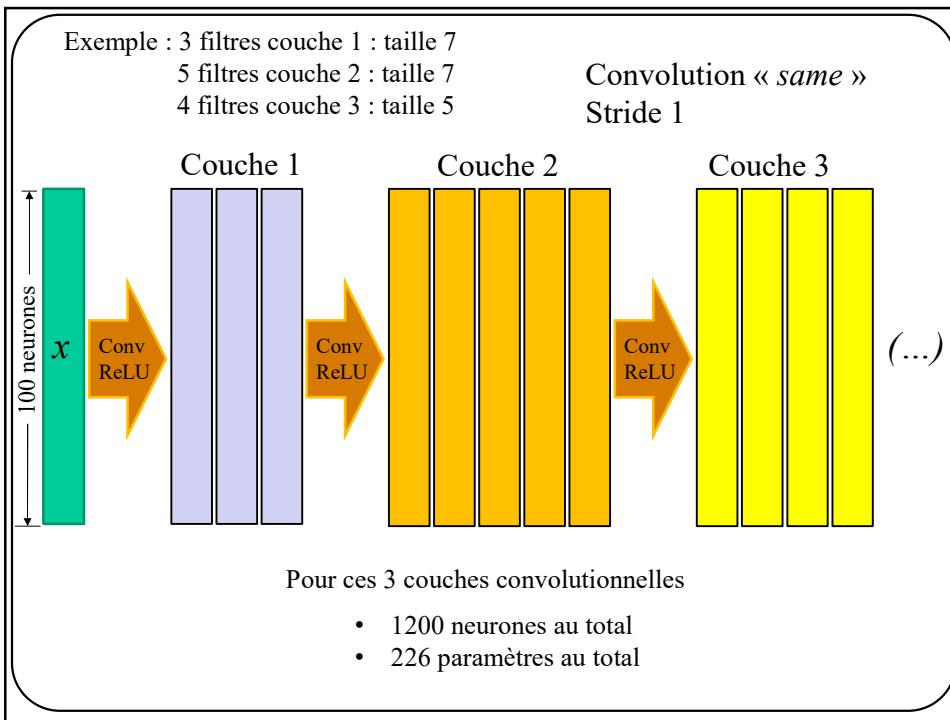
106

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

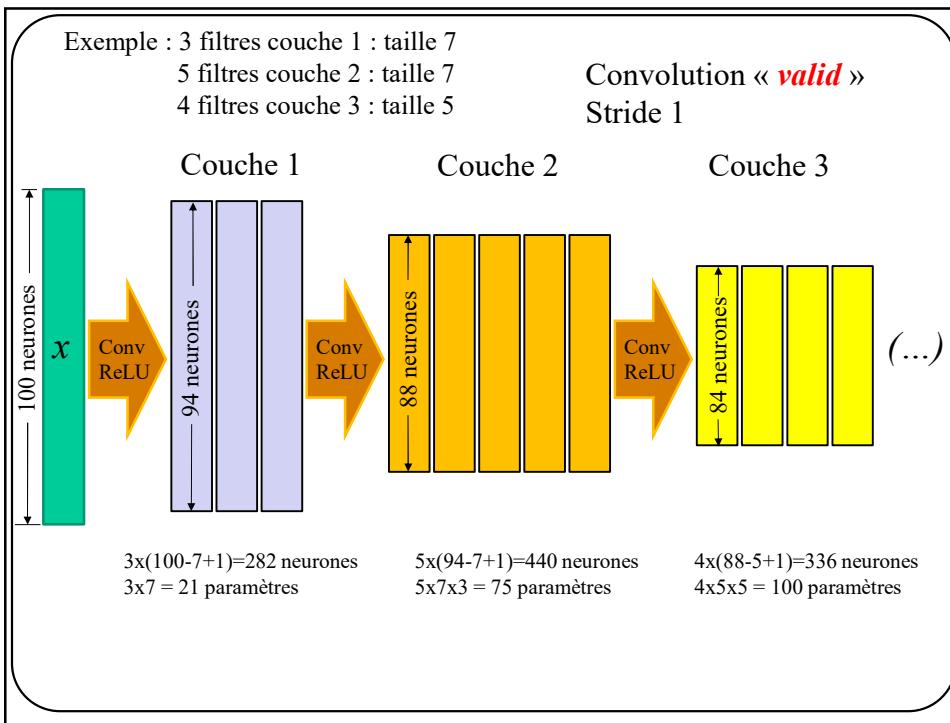
107



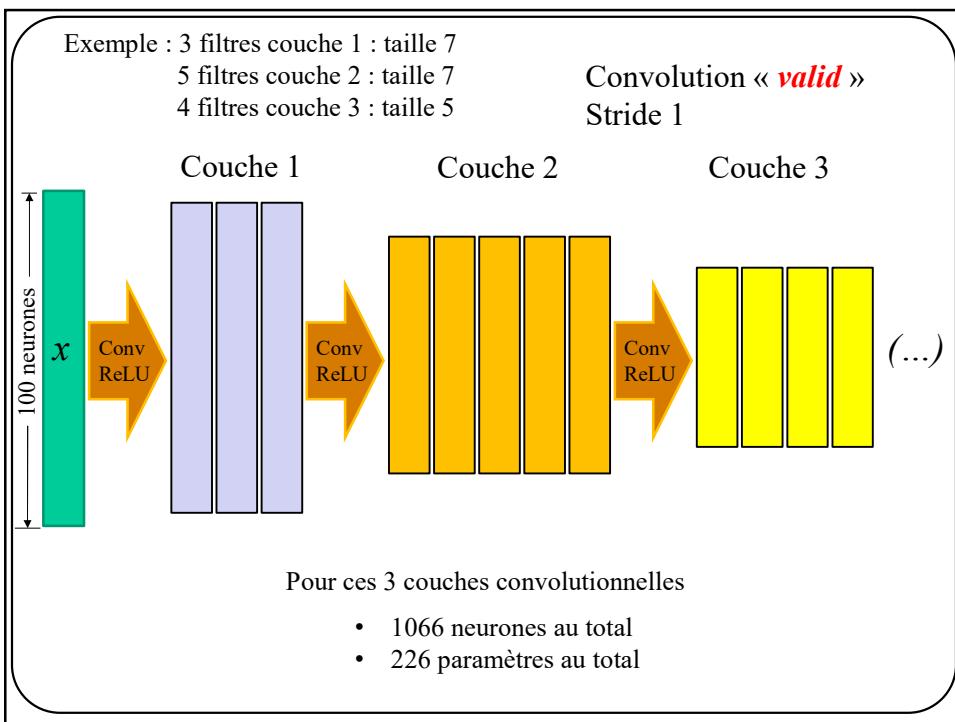
108



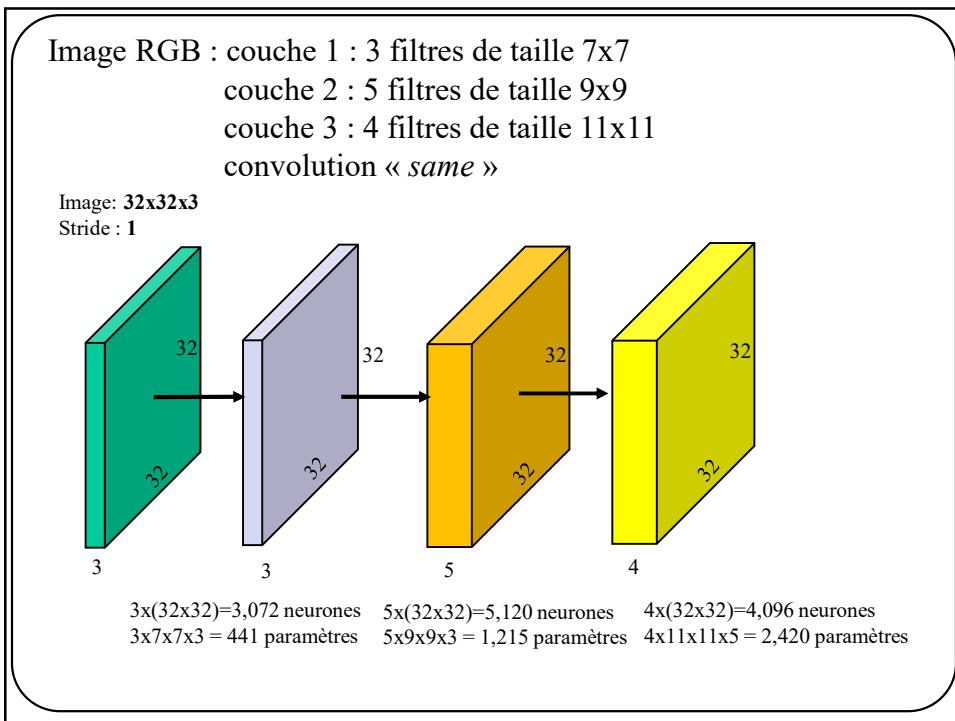
109



110



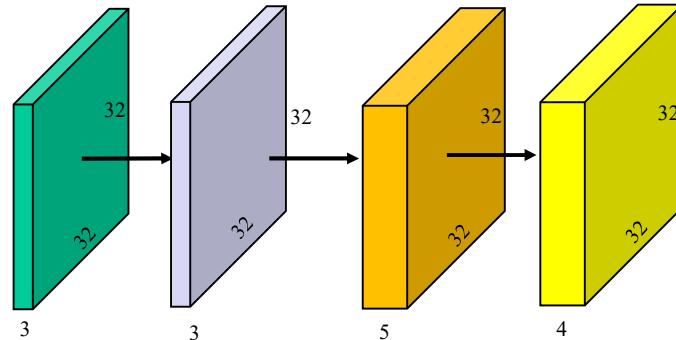
111



112

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

Image: 32x32x3
Stride : 1

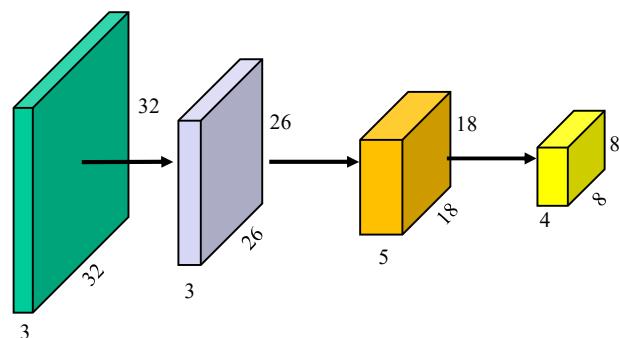


12,288 neurones au total
4,076 paramètres au total

113

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

Image: 32x32x3
Stride : 1

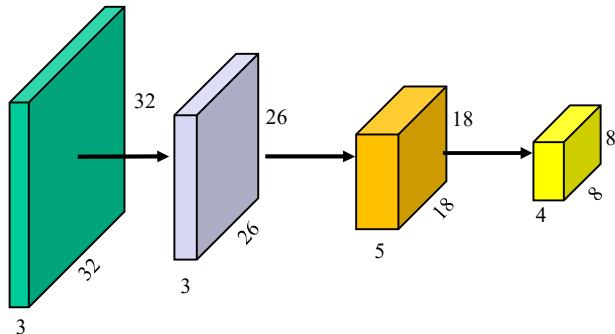


$3 \times (26 \times 26) = 2,028$ neurones $5 \times (18 \times 18) = 1,620$ neurones $4 \times (8 \times 8) = 256$ neurones
 $3 \times 7 \times 7 \times 3 = 441$ paramètres $5 \times 9 \times 9 \times 3 = 1,215$ paramètres $4 \times 11 \times 11 \times 5 = 2,420$ paramètres

114

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

Image: 32x32x3
Stride : 1

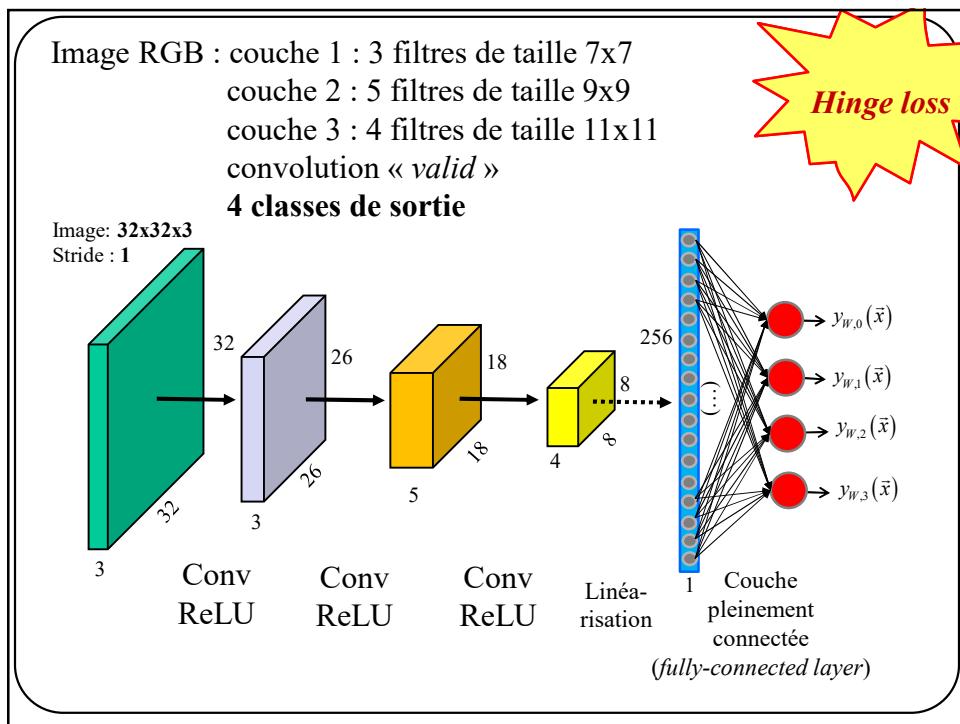


3,904 neurones au total
4,076 paramètres au total

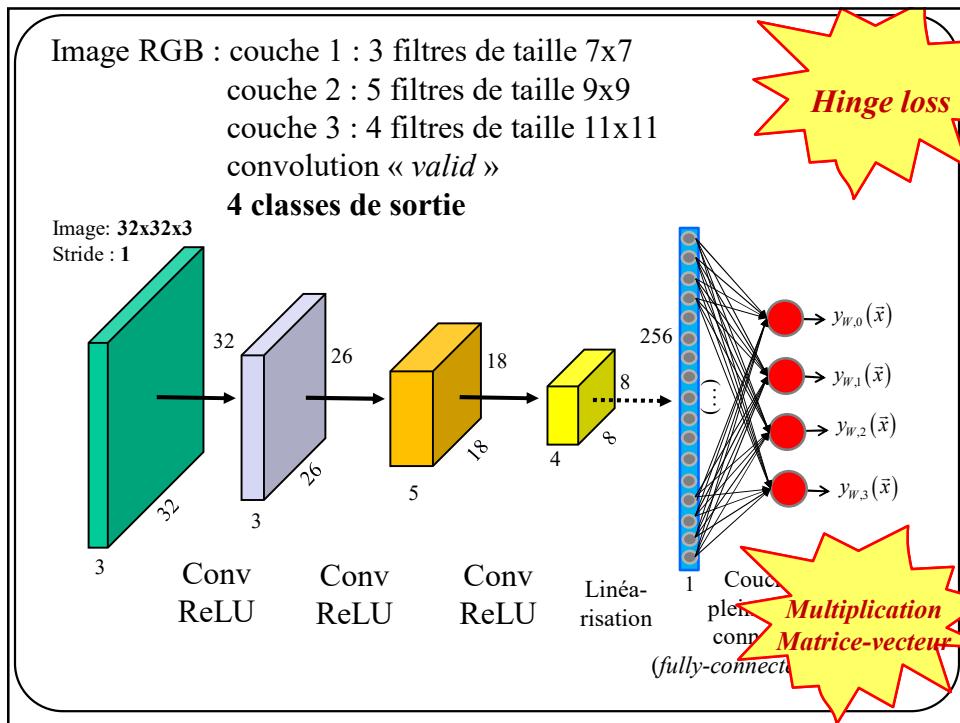
115

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

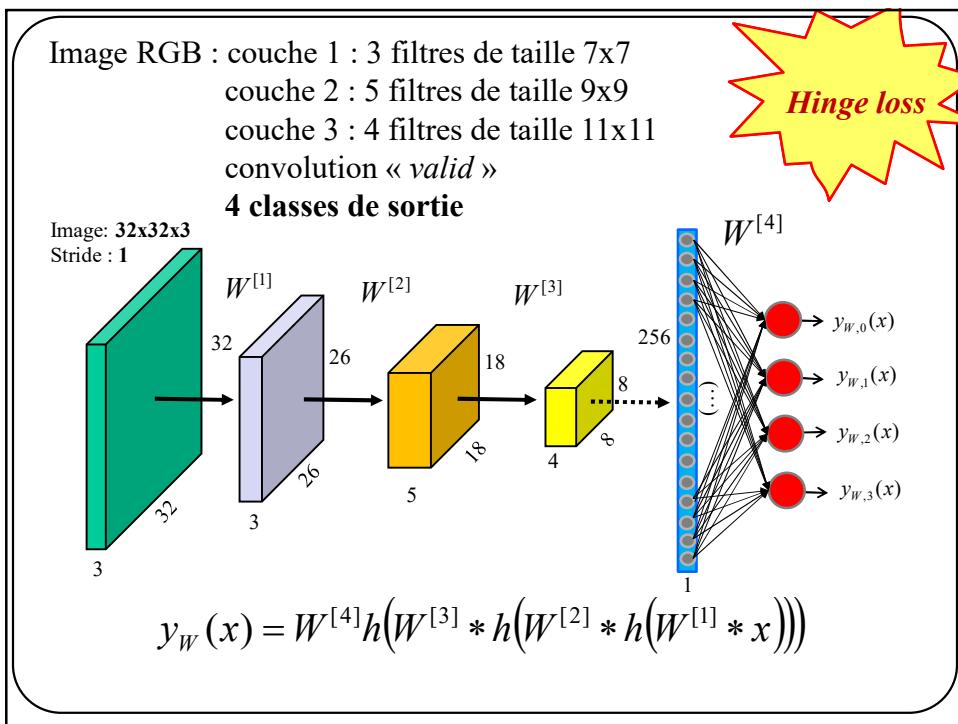
116



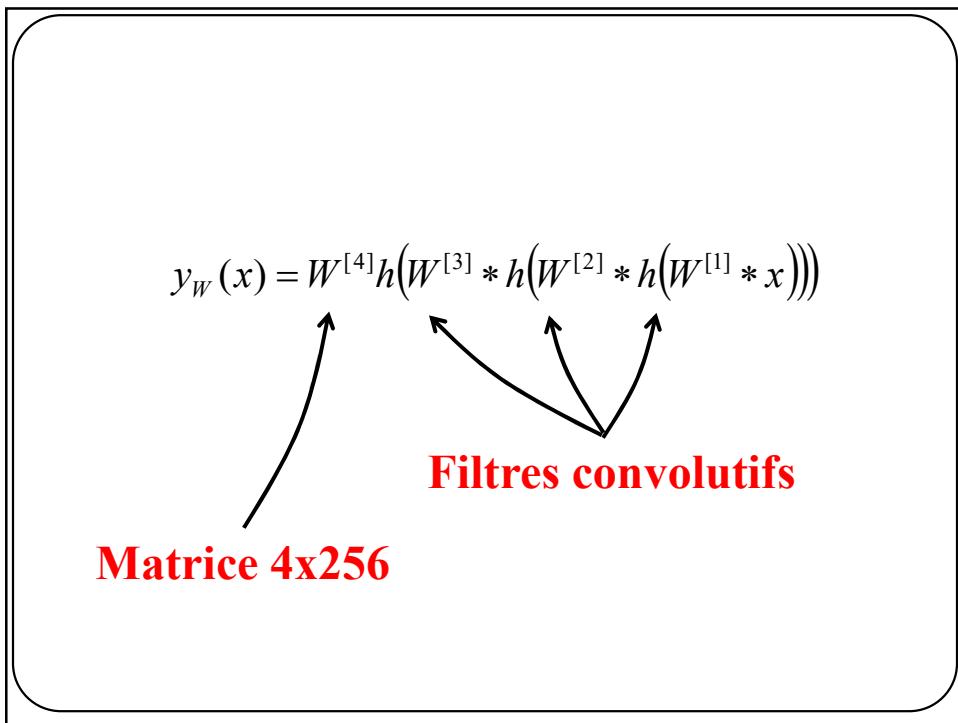
117



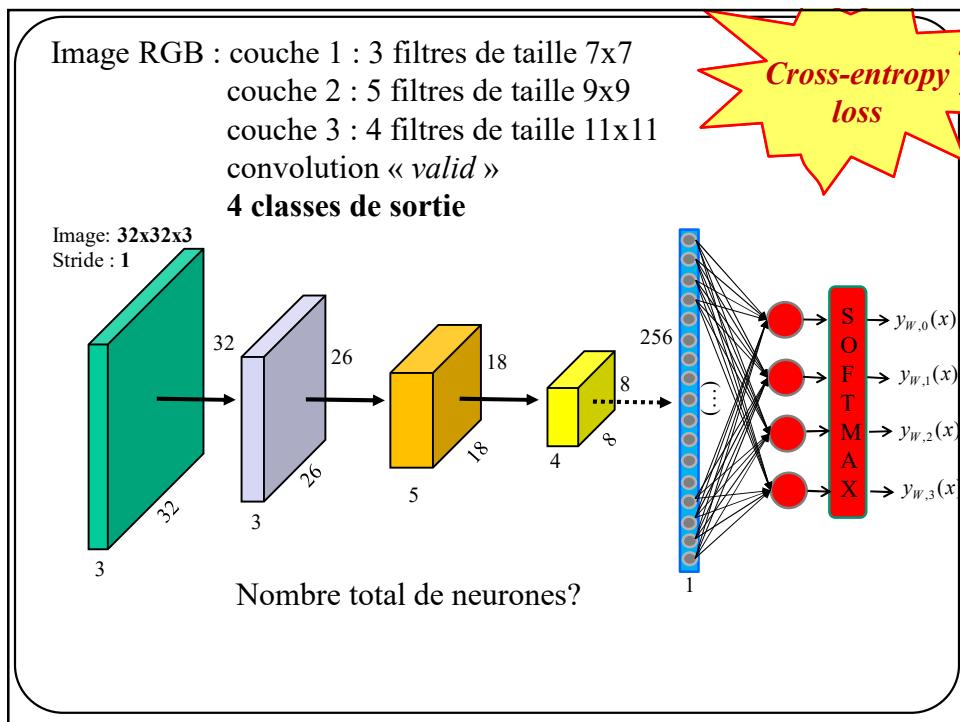
118



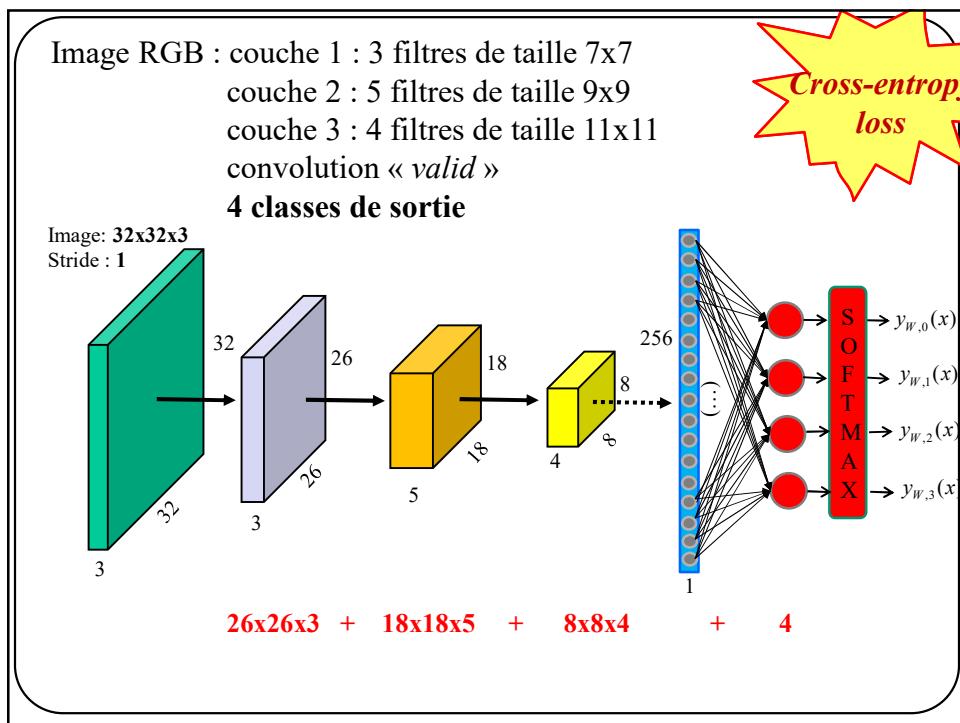
119



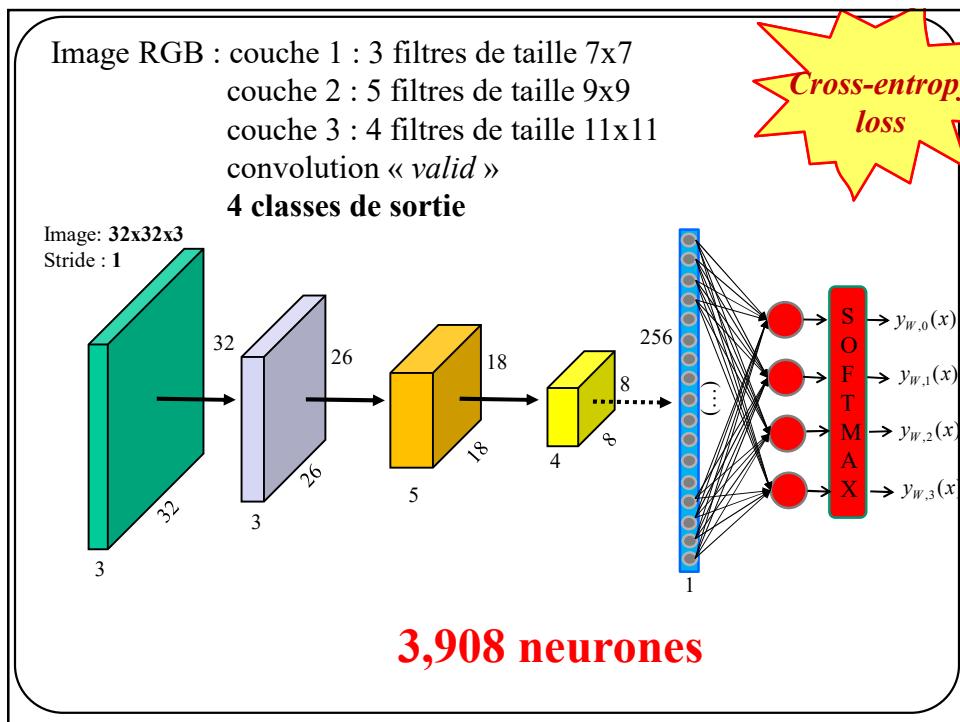
120



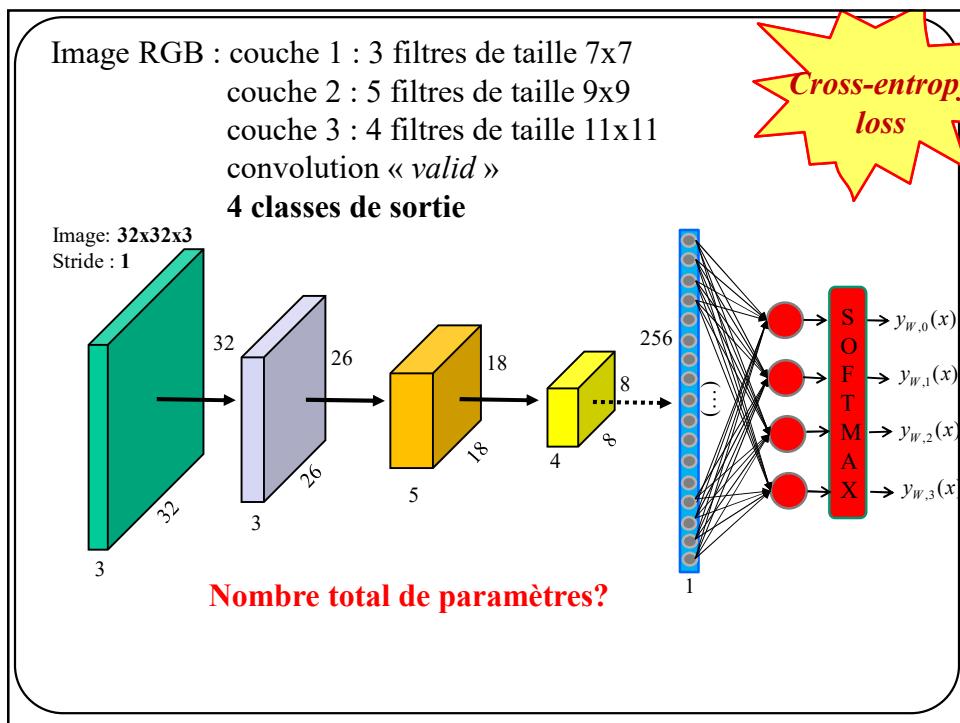
121



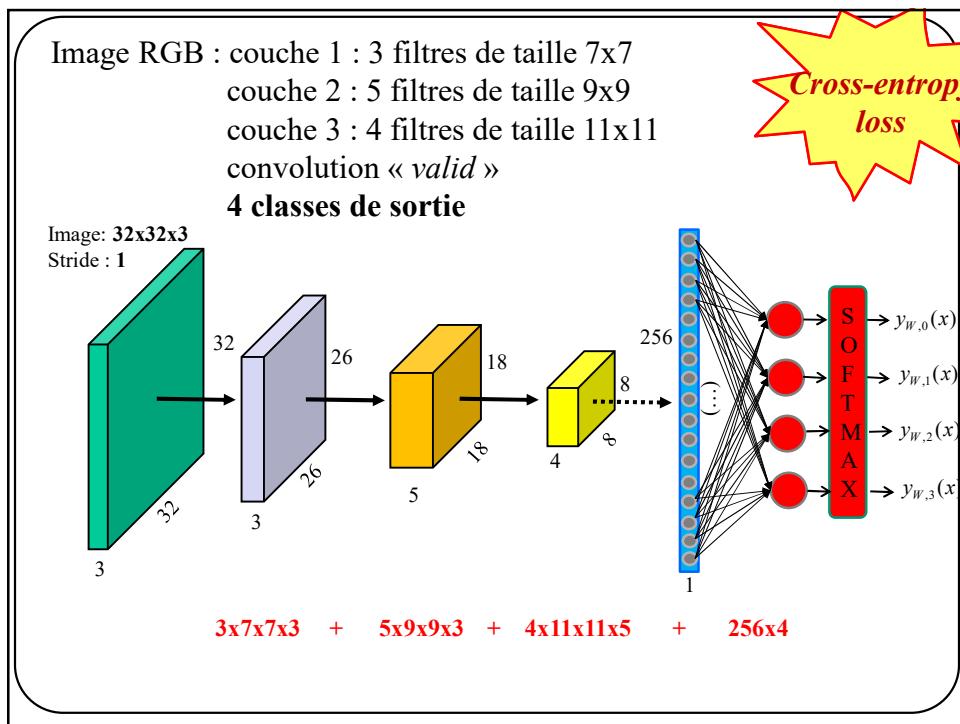
122



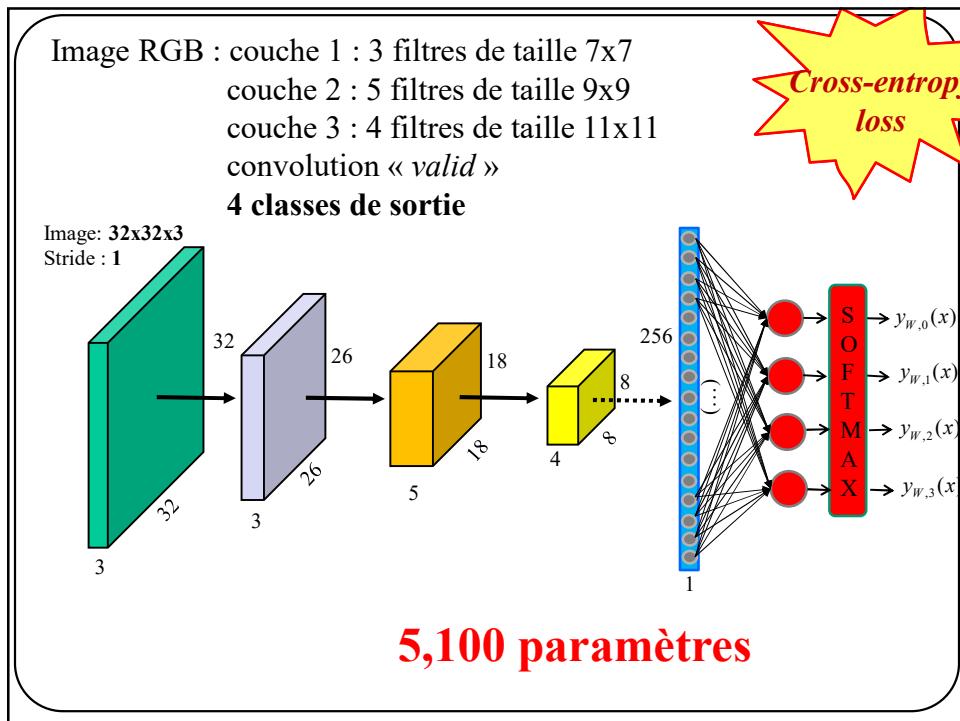
123



124



125



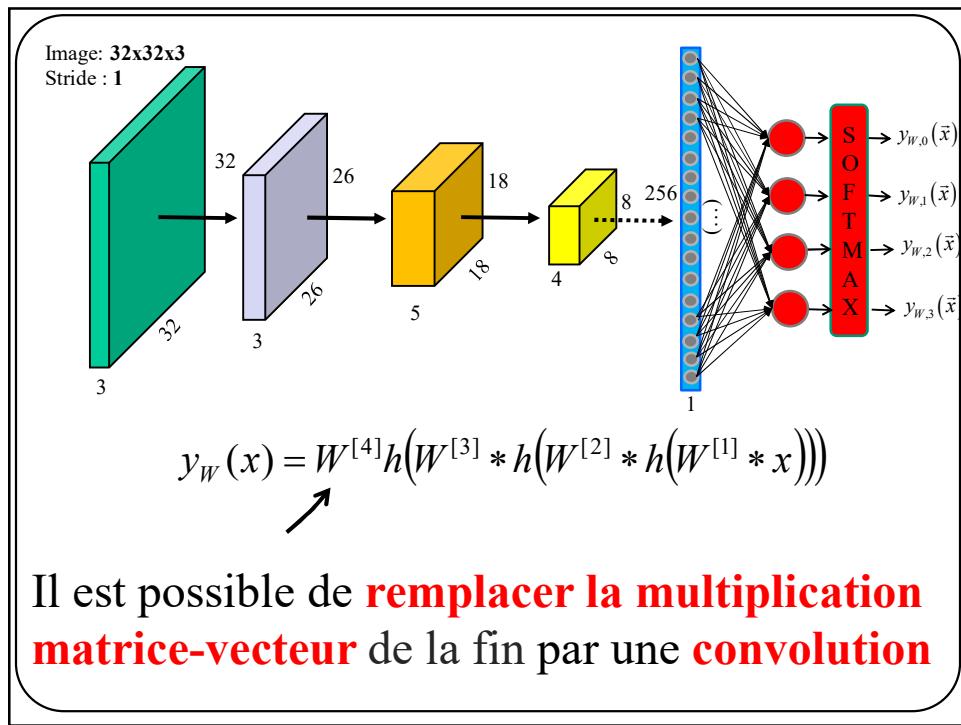
126

Réseaux à convolution

VS

Réseaux **pleinement** convolutifs

127



128

Exemple 1d (convolution « valid »)

$$\begin{array}{ccccccc} & \xrightarrow{9} & & \xrightarrow{3} & & \xrightarrow{7} & \\ \xleftarrow{\hspace{1cm}} & & & \xleftarrow{\hspace{1cm}} & & \xleftarrow{\hspace{1cm}} & \\ \boxed{40 \ 50 \ 70 \ 80 \ 90 \ 10 \ 20 \ 30 \ 40} & * & \boxed{.2 \ -3 \ .4} & = & \boxed{21 \ 21 \ 26 \ -7 \ 23 \ 8 \ 11} & & \end{array}$$

129

Exemple 1d (convolution « valid »)

$$\begin{array}{ccccccc} & \xrightarrow{9} & & \xrightarrow{5} & & \xrightarrow{5} & \\ \xleftarrow{\hspace{1cm}} & & & \xleftarrow{\hspace{1cm}} & & \xleftarrow{\hspace{1cm}} & \\ \boxed{40 \ 50 \ 70 \ 80 \ 90 \ 10 \ 20 \ 30 \ 40} & * & \boxed{.2 \ -3 \ .4 \ -.5 \ .6} & = & \boxed{35 \ -18 \ 33 \ 1 \ 32} & & \end{array}$$

130

Exemple 1d (convolution « valid »)

$$\begin{array}{ccccccc} & \xrightarrow{9} & & \xrightarrow{7} & & \xrightarrow{3} & \\ \hline 40 & 50 & 70 & 80 & 90 & 10 & 20 & 30 & 40 & * & [.2 & -.3 & .4 & -.5 & .6 & -.7 & .8] & = & [44 & -8 & 44] \end{array}$$

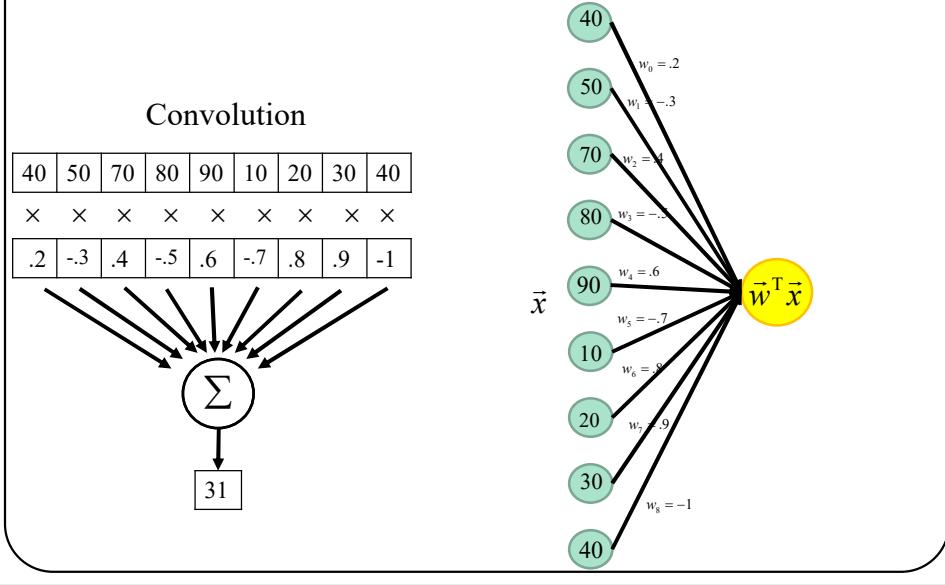
131

Taille filtre = nb de neurones couche précédente

$$\begin{array}{ccccccc} & \xrightarrow{9} & & \xrightarrow{9} & & \xrightarrow{1} & \\ \hline 40 & 50 & 70 & 80 & 90 & 10 & 20 & 30 & 40 & * & [.2 & -.3 & .4 & -.5 & .6 & -.7 & .8 & .9 & -.1] & = & [31] \end{array}$$

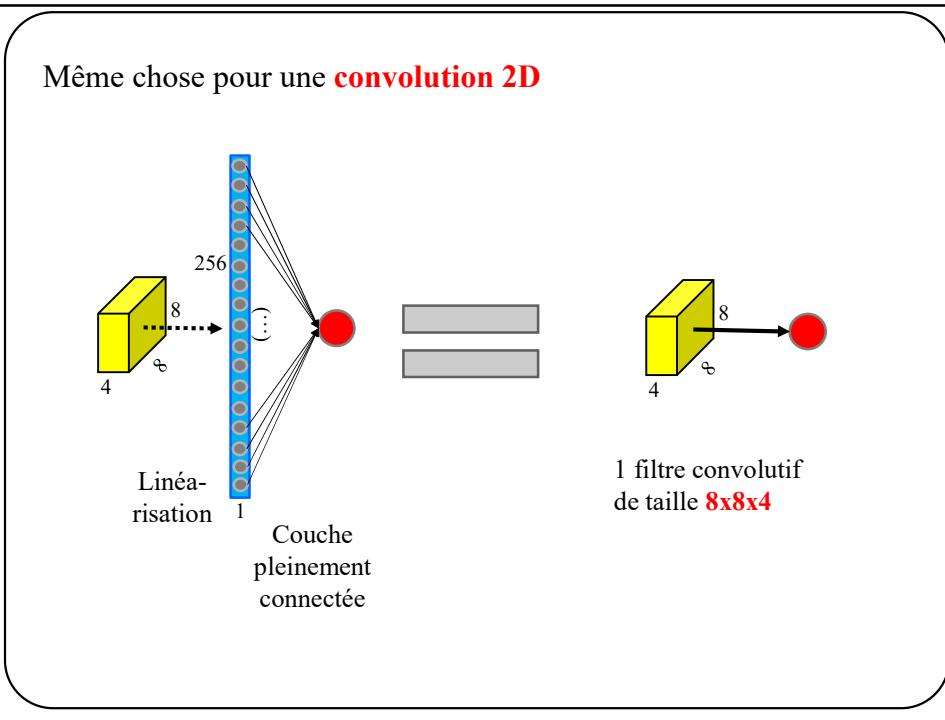
132

Signal d'entrée de **taille 9** convolué avec un filtre « same » de **taille 9** correspond à une **couche pleinement connectée**



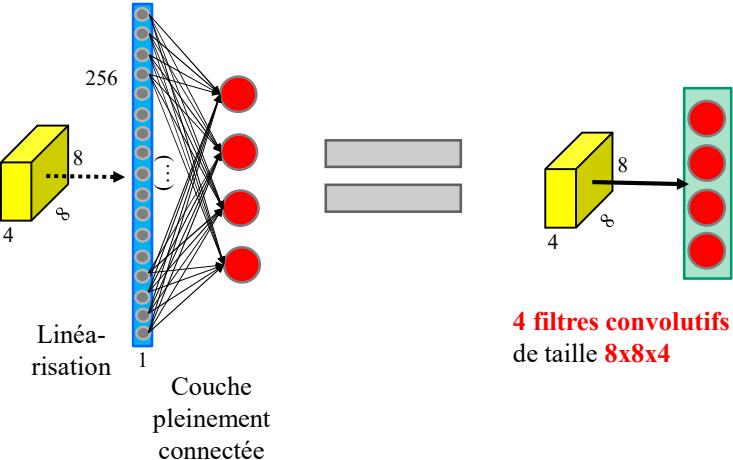
133

Même chose pour une **convolution 2D**

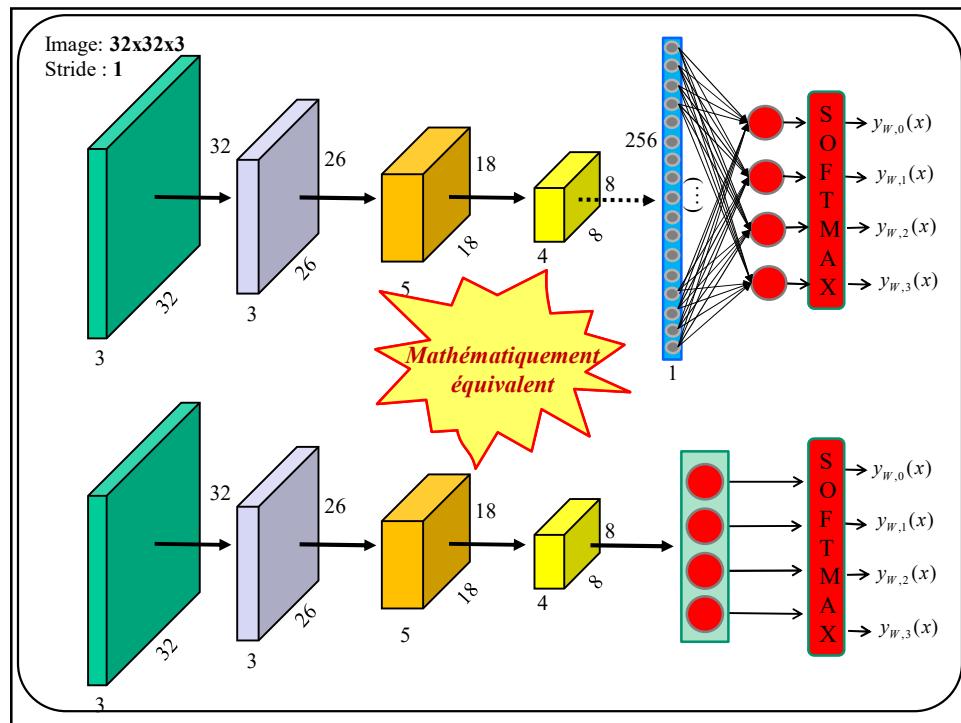


134

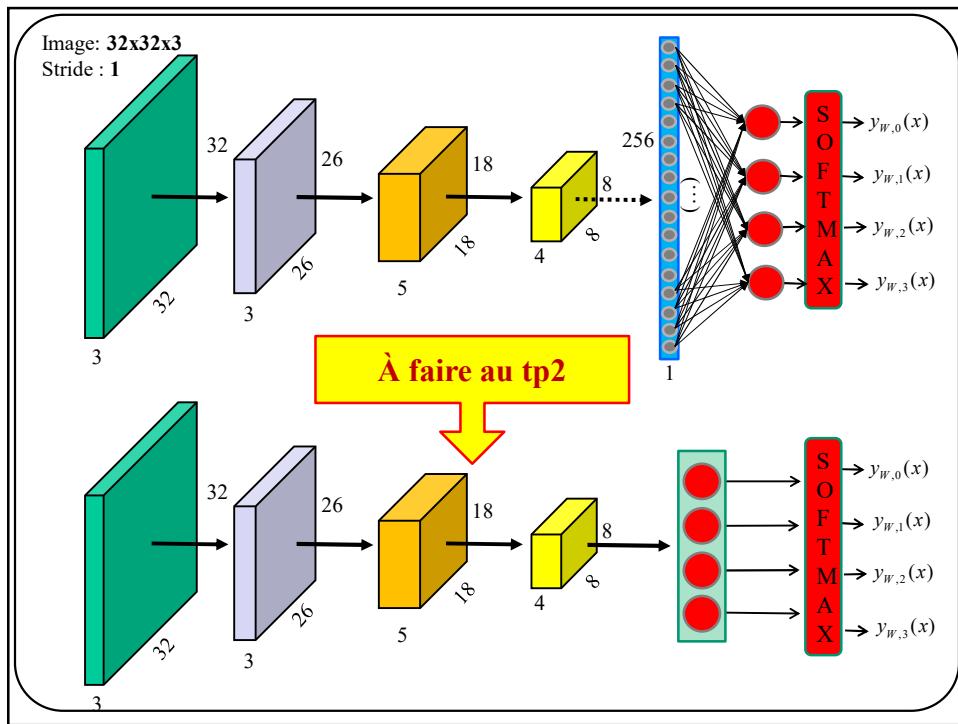
Même chose pour une **convolution 2D**



135



136



137

Configurations équivalentes

couche 1 : 3 filtres de taille 7x7
 couche 2 : 5 filtres de taille 9x9
 couche 3 : 4 filtres de taille 11x11
couche 4 pleinement connectée 256x4
 Softmax

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

En fait, presque équivalent ...

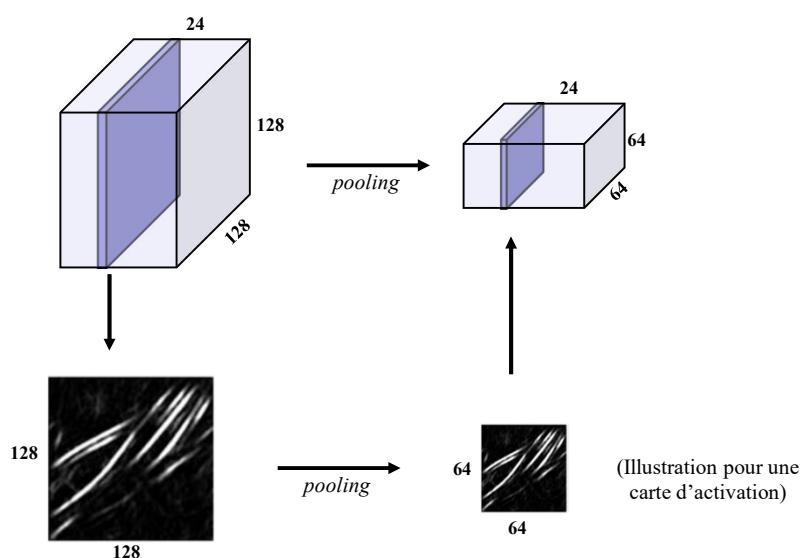
Question : qu'arrive-t-il si on remplace l'image 32x32x3 par une image 64x64x3?

138

Pooling

139

Réduction de la taille des cartes d'activation



140

Max pooling

Max pool par filtre « valid » 2x2 avec stride =2

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

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

141

Mean pooling

Moyenne par filtre « valid » 2x2 avec stride =2

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

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

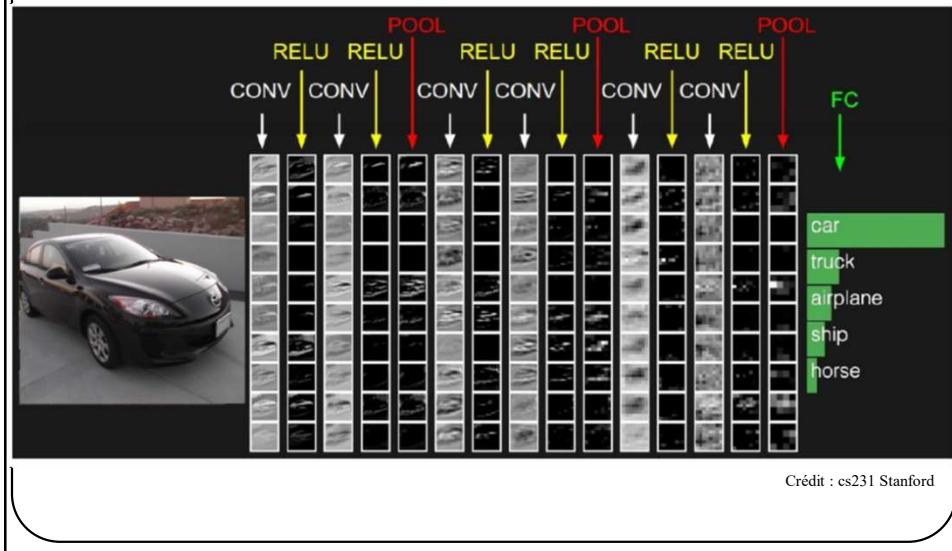
1	2	4	4	9	3	1	2
6	7	8	4	-3	-3	6	3
9	-9	8	-4	5	5	3	0
8	-8	9	-9	5	5	0	1
0	0	1	2	7	9	7	8
-1	-3	3	6	8	8	7	6
9	9	8	2	1	5	-1	-1
1	1	-2	8	3	7	4	-2

Max pooling 3x3
avec stride =2



144

Illustration d'un CNN complet



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

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

Filtre

W0	W1	W2
W3	W4	W5
W6	W7	W8

*

=

Y1	Y2
Y3	Y4

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

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

Filtre

W0	W1	W2
W3	W4	W5
W6	W7	W8

*

=

Y0	Y1
Y2	Y3

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

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

149

Rappel

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

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

Y0	Y1
Y2	Y3

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

150

Rappel

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

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

Y0	Y1
Y2	Y3

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

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
« matriçant » l'entrée

$$\begin{array}{ccccccccc} W_0 & W_1 & W_2 & W_3 & W_4 & W_5 & W_6 & W_7 & W_8 \end{array} \times \begin{array}{|c|c|c|c|} \hline X_0 & X_1 & X_4 & X_5 \\ \hline X_1 & X_2 & X_5 & X_6 \\ \hline X_2 & X_3 & X_6 & X_7 \\ \hline X_4 & X_5 & X_8 & X_9 \\ \hline X_5 & X_6 & X_9 & X_{10} \\ \hline X_6 & X_7 & X_{10} & X_{11} \\ \hline X_8 & X_9 & X_{11} & X_{13} \\ \hline X_9 & X_{10} & X_{12} & X_{14} \\ \hline X_{10} & X_{11} & X_{13} & X_{15} \\ \hline \end{array} = \begin{array}{|c|c|c|c|} \hline Y_0 & Y_1 & Y_2 & Y_3 \\ \hline \end{array}$$

158

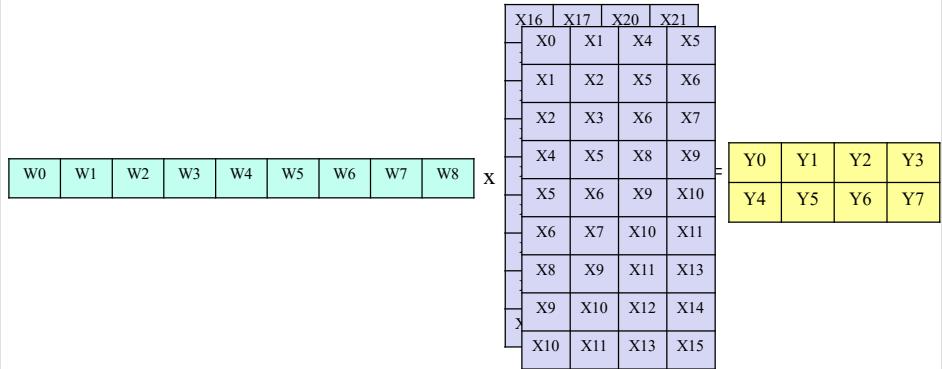
Autre exemple
conv « valid », mini-batch de 2 entrées

$$\begin{array}{cccc} \text{2 données en entrée} & & \text{Filtre} & \\ \begin{array}{|c|c|c|c|} \hline X_0 & X_1 & X_2 & X_3 \\ \hline X_4 & X_5 & X_6 & X_7 \\ \hline X_8 & X_9 & X_{10} & X_{11} \\ \hline X_{12} & X_{13} & X_{14} & X_{15} \\ \hline \end{array} & * & \begin{array}{|c|c|c|} \hline W_0 & W_1 & W_2 \\ \hline W_3 & W_4 & W_5 \\ \hline W_6 & W_7 & W_8 \\ \hline \end{array} & = \begin{array}{|c|c|} \hline Y_0 & Y_1 \\ \hline Y_2 & Y_3 \\ \hline Y_4 & Y_5 \\ \hline Y_6 & Y_7 \\ \hline \end{array} \end{array}$$

159

Autre exemple

conv « valid », mini-batch de 2 entrées



160

Autre exemple

conv « valid », une entrée, deux filtres

Entrée

X ₀	X ₁	X ₂	X ₃
X ₄	X ₅	X ₆	X ₇
X ₈	X ₉	X ₁₀	X ₁₁
X ₁₂	X ₁₃	X ₁₄	X ₁₅

Filtre

W ₀	W ₁	W ₂	Y ₀	Y ₁
W ₃	W ₄	W ₅	Y ₂	Y ₃
W ₆	W ₇	W ₈	Y ₄	Y ₅
W ₉	W ₁₀	W ₁₁	Y ₆	Y ₇
W ₁₂	W ₁₃	W ₁₄		
W ₁₅	W ₁₆	W ₁₇		

161

Autre exemple

conv « valid », une entrée, deux filtres

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|} \hline
 W_0 & W_1 & W_2 & W_3 & W_4 & W_5 & W_6 & W_7 & W_8 \\ \hline
 W_9 & W_{10} & W_{11} & W_{12} & W_{13} & W_{14} & W_{15} & W_{16} & W_{17} \\ \hline
 \end{array} \times \begin{array}{|c|c|c|c|} \hline
 X_0 & X_1 & X_4 & X_5 \\ \hline
 X_1 & X_2 & X_5 & X_6 \\ \hline
 X_2 & X_3 & X_6 & X_7 \\ \hline
 X_4 & X_5 & X_8 & X_9 \\ \hline
 X_5 & X_6 & X_9 & X_{10} \\ \hline
 X_6 & X_7 & X_{10} & X_{11} \\ \hline
 X_8 & X_9 & X_{11} & X_{13} \\ \hline
 X_9 & X_{10} & X_{12} & X_{14} \\ \hline
 X_{10} & X_{11} & X_{13} & X_{15} \\ \hline
 \end{array} = \begin{array}{|c|c|c|c|} \hline
 Y_0 & Y_1 & Y_2 & Y_3 \\ \hline
 Y_4 & Y_5 & Y_6 & Y_7 \\ \hline
 \end{array}$$

162

Autre exemple

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

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

$$\begin{array}{|c|c|c|} \hline
 W_0 & W_1 & W_2 \\ \hline
 W_3 & W_4 & W_5 \\ \hline
 W_6 & W_7 & W_8 \\ \hline
 W_9 & W_{10} & W_{11} \\ \hline
 W_{12} & W_{13} & W_{14} \\ \hline
 W_{15} & W_{16} & W_{17} \\ \hline
 \end{array} * = \begin{array}{|c|c|} \hline
 Y_0 & Y_1 \\ \hline
 Y_2 & Y_3 \\ \hline
 \end{array}$$

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
----	----	----	----	-------	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----

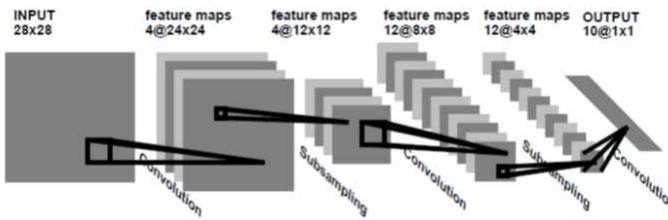
$$\mathbf{X} = \begin{array}{|c|c|c|c|} \hline X_0 & X_1 & X_4 & X_5 \\ \hline X_1 & X_2 & X_5 & X_6 \\ \hline X_2 & X_3 & X_6 & X_7 \\ \hline X_4 & X_5 & X_8 & X_9 \\ \hline (...) & (...) & (...) & (...) \\ \hline X_{22} & X_{23} & X_{26} & X_{27} \\ \hline X_{24} & X_{25} & X_{27} & X_{29} \\ \hline X_{25} & X_{26} & X_{28} & X_{30} \\ \hline X_{26} & X_{27} & X_{29} & X_{31} \\ \hline \end{array} \quad Y_0 \quad Y_1 \quad Y_2 \quad Y_3$$

164

Exemples d'architectures connues

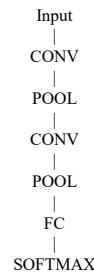
165

LeNet-1



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

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



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

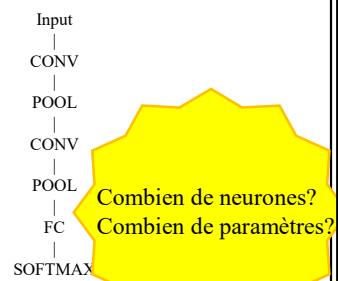
166

LeNet-1



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

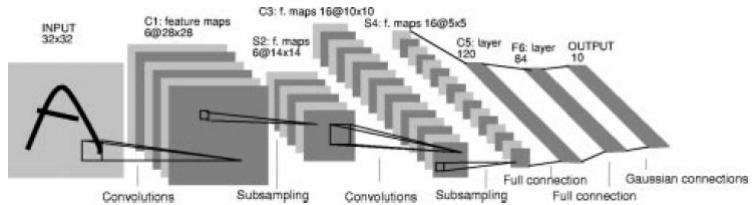
- Image d'entrée : $28 \times 28 \times 1$
- Filtres convolutionnels : 5×5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 4 et 12 filtres)
- 2 *average pooling*
- 1 couche pleinement connectée
- 10 classes



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

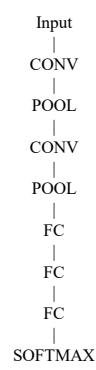
167

LeNet-5



Version améliorée:

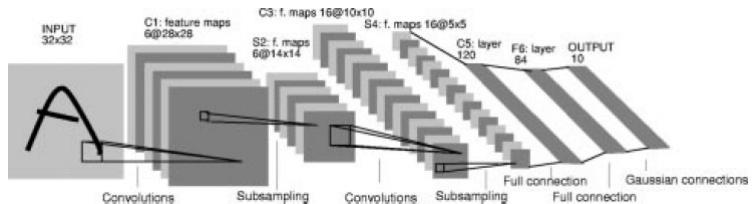
- Image d'entrée : $28 \times 28 \times 1$
- Filtres convolutionnels : 5×5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 6 et 16 filtres)
- 2 *average pooling*
- 3 couches pleinement connectées (120, 84 et 10 neurones)
- 10 classes



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

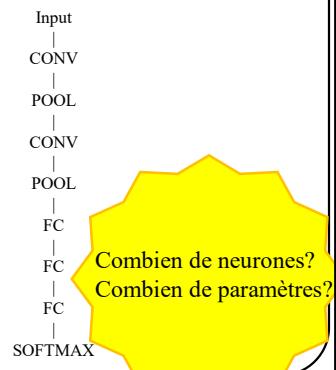
168

LeNet-5



Version améliorée:

- Image d'entrée : $28 \times 28 \times 1$
- Filtres convolutionnels : 5×5
- 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



169

Classification d'images

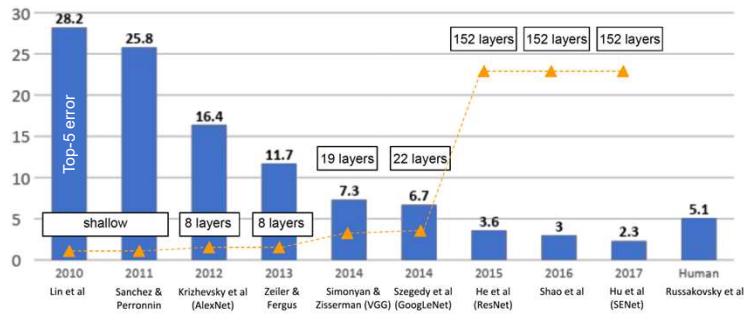
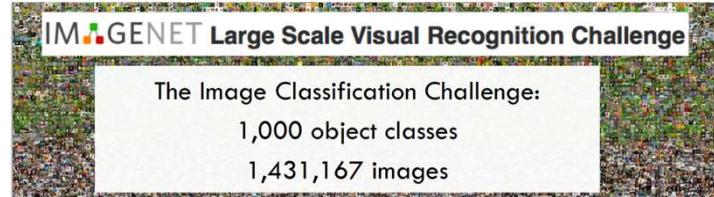


Image: <http://cs231n.stanford.edu/slides/2018>

170

Classification d'images

AlexNet [Krizhevsky et al, 2012]

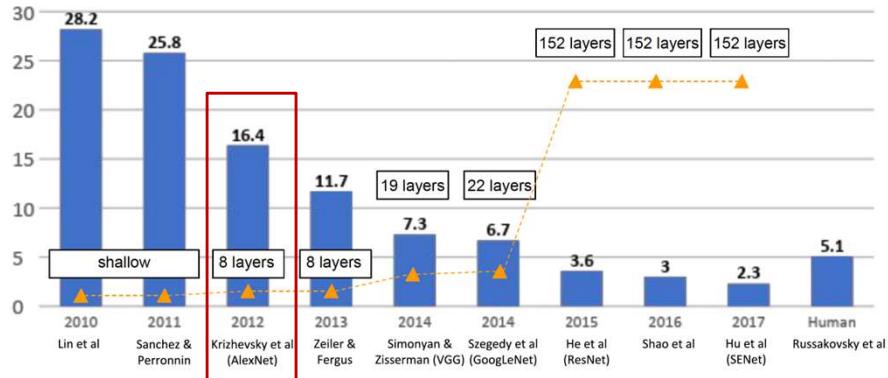
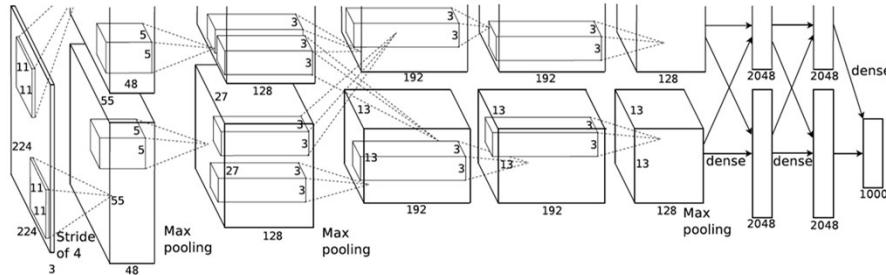


Image: <http://cs231n.stanford.edu/slides/2018>

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

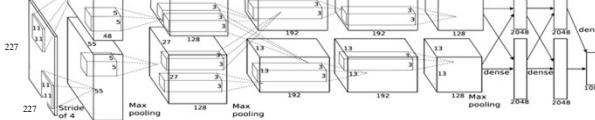
Image: Krizhevsky et al. "Imagenet classification with deep convolutional neural networks," NIPS 2012.

172

AlexNet

Architecture:

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



Entrée : image RGB: 227x227x3

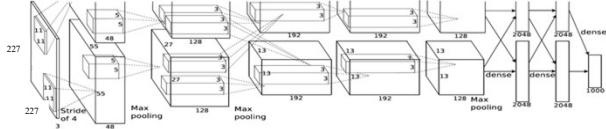
Couche 1 (CONV1): 96 filtres de taille 11x11 avec **stride de 4** et conv “**valid**”

Quelle est la taille des cartes d’activation?

Réponse: $(227-11)/4+1 = 55$

173

AlexNet



Architecture:

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

Entrée : image RGB: 227x227x3

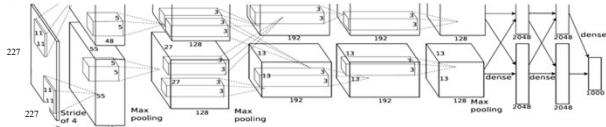
Couche 1 (CONV1): 96 filtres de taille 11x11 avec stride de 4 et conv “valid”

Cartes d’activation : 96 x 55 x 55

Q: Quel est le nombre de paramètres?

174

AlexNet



Architecture:

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

Entrée : image RGB: 227x227x3

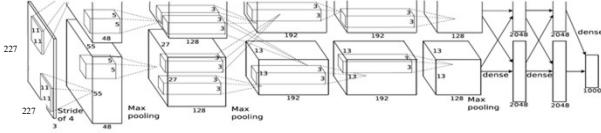
Couche 1 (CONV1): 96 filtres de taille 11x11 avec stride de 4 et conv “valid”

Cartes d’activation : 96 x 55 x 55

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

175

AlexNet



ENTRÉE : 227x227x3

CONV1: 96 x 55 x 55

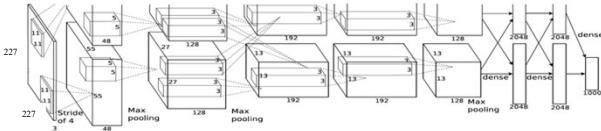
Couche 2 MaxPool : 3x3 stride stride 2

Quelle est la taille des cartes d'activation?

Réponse: $(55-3)/2+1 = 27$

176

AlexNet



ENTRÉE : 227x227x3

CONV1: 96 x 55 x 55

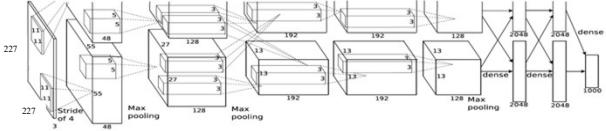
**Couche 2 MaxPool : 3x3 stride stride 2
27 x 27 x 96**

Combien y a-t-il de paramètres?

Réponse: 0!

177

AlexNet



ENTRÉE : 227x227x3

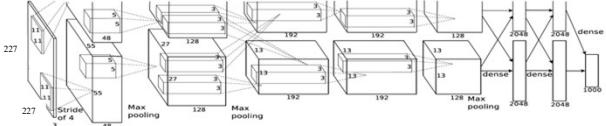
CONV1: 55 x 55 x 96

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

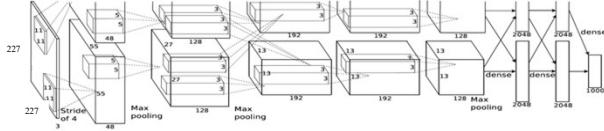
FC6 4096

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

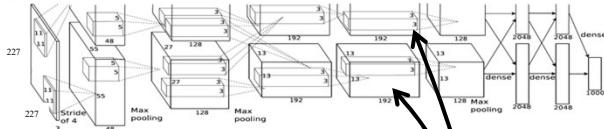
Taux d'apprentissage 0.01 avec

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**

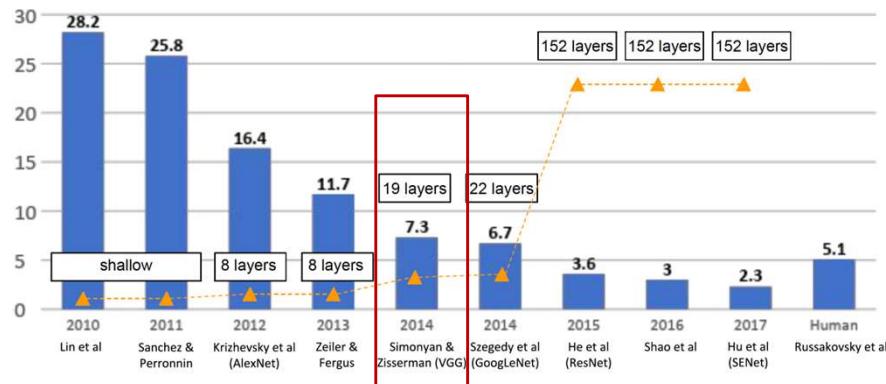
Fait voter un ensemble de 7 CNN:

18.2% -> 15.4

181

Classification d'images

VGGNet [Simonyan and Zisserman, 2014]



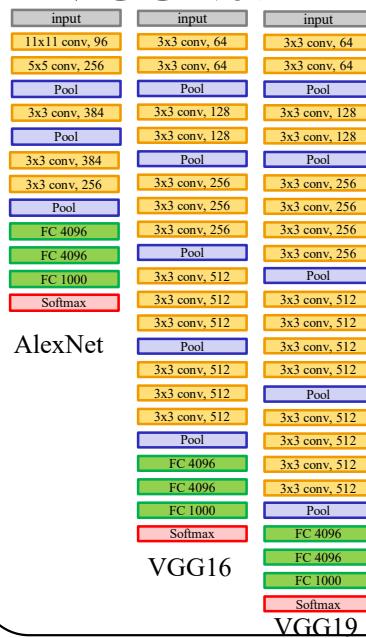
Karen Simonyan, Andrew Zisserman "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

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

182

182

VGGNet



Ce qui caractérise VGGNet par rapport à ses prédecesseurs:

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

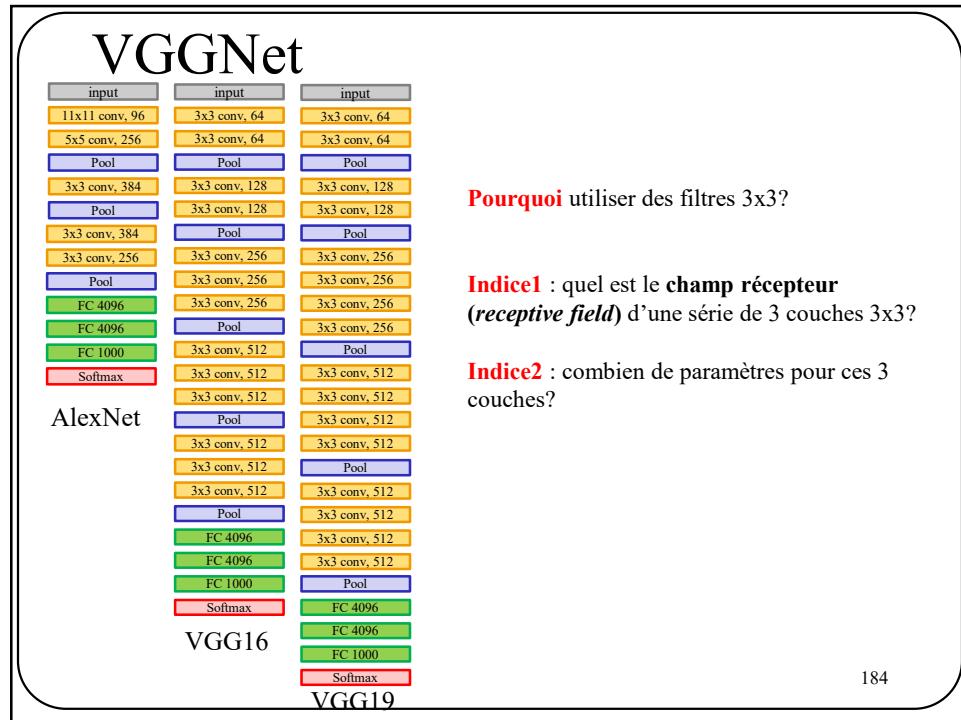
AlexNet : 8 couches

VGGNet : 16 ou 19 couches

7.3% d'erreur contre 11.7% pour ZFNet

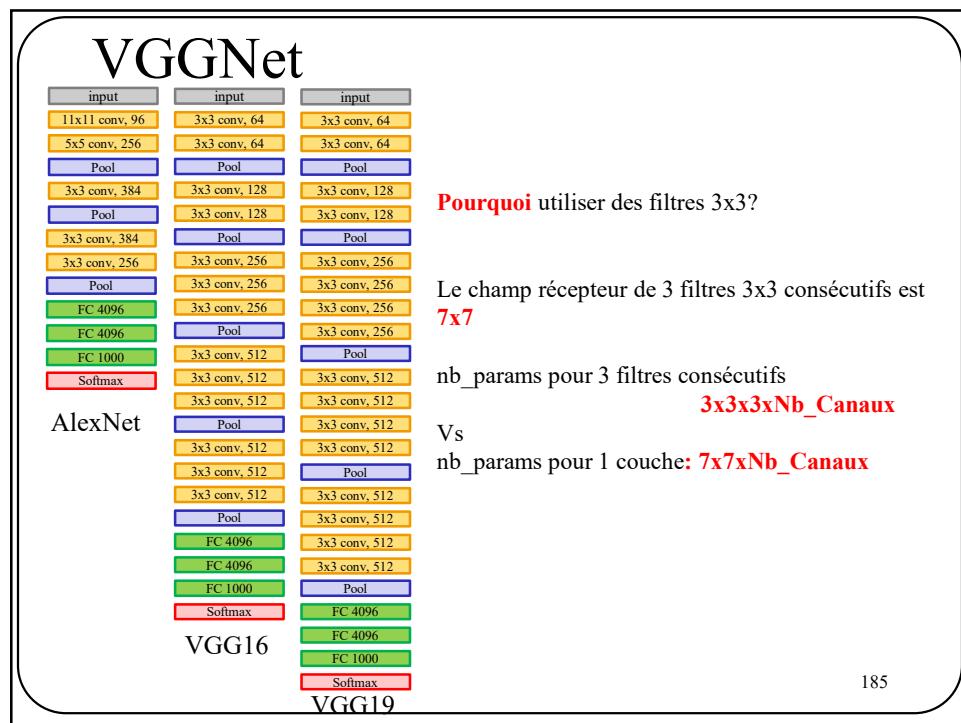
183

183



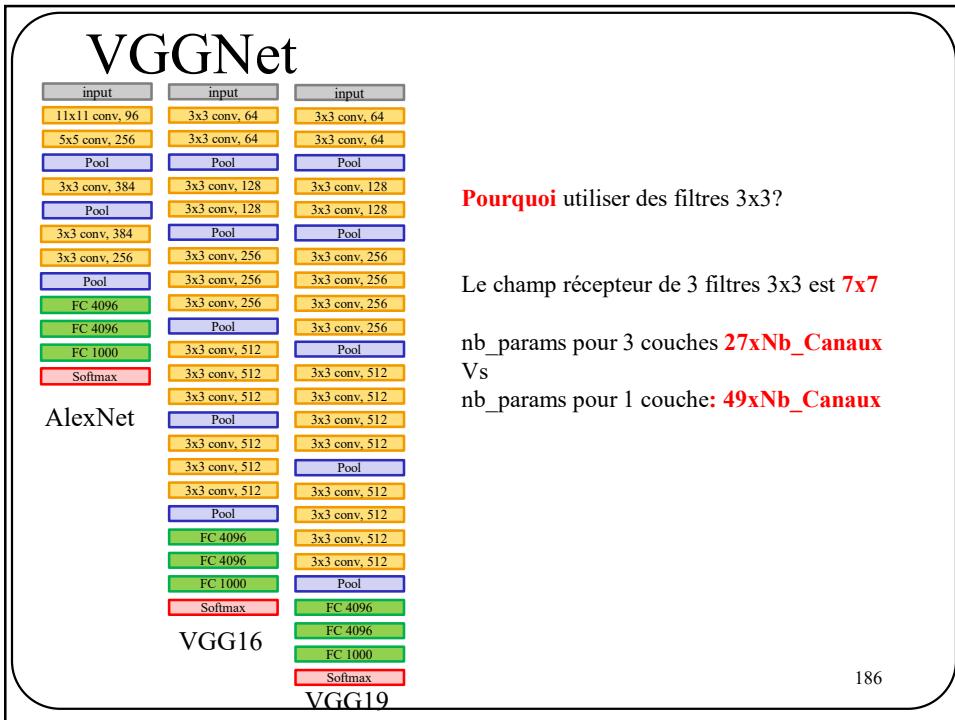
184

184



185

185



186

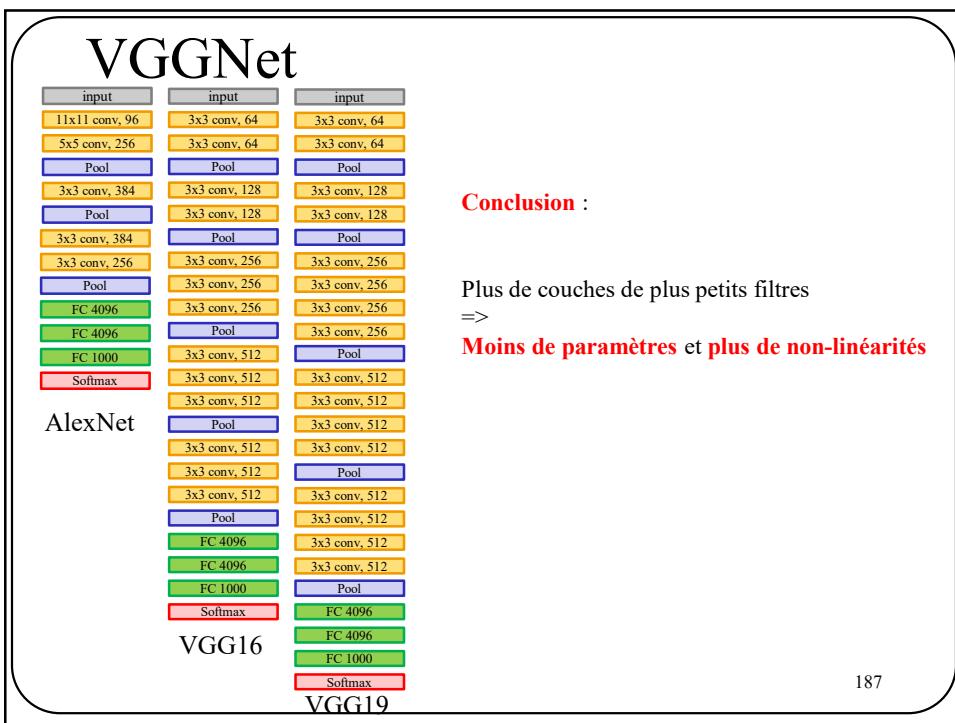
Pourquoi utiliser des filtres 3x3?

Le champ récepteur de 3 filtres 3x3 est **7x7**

nb_params pour 3 couches **27xNb_Canaux**

Vs

nb_params pour 1 couche: **49xNb_Canaux**



187

Conclusion :

Plus de couches de plus petits filtres

=>

Moins de paramètres et plus de non-linéarités

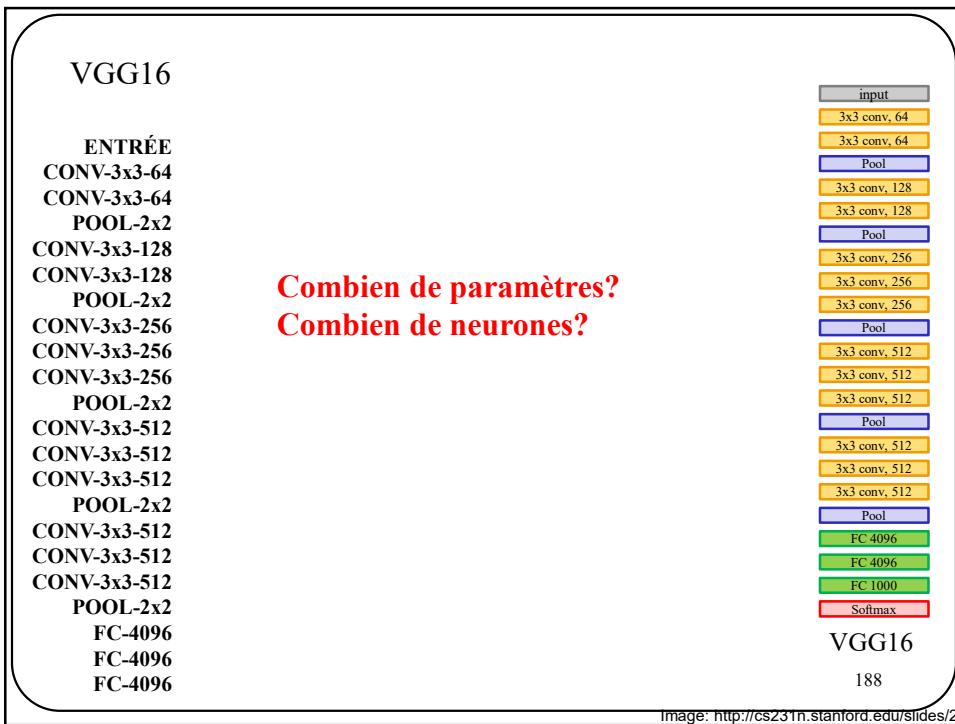


Image: <http://cs231n.stanford.edu/slides/2>

188

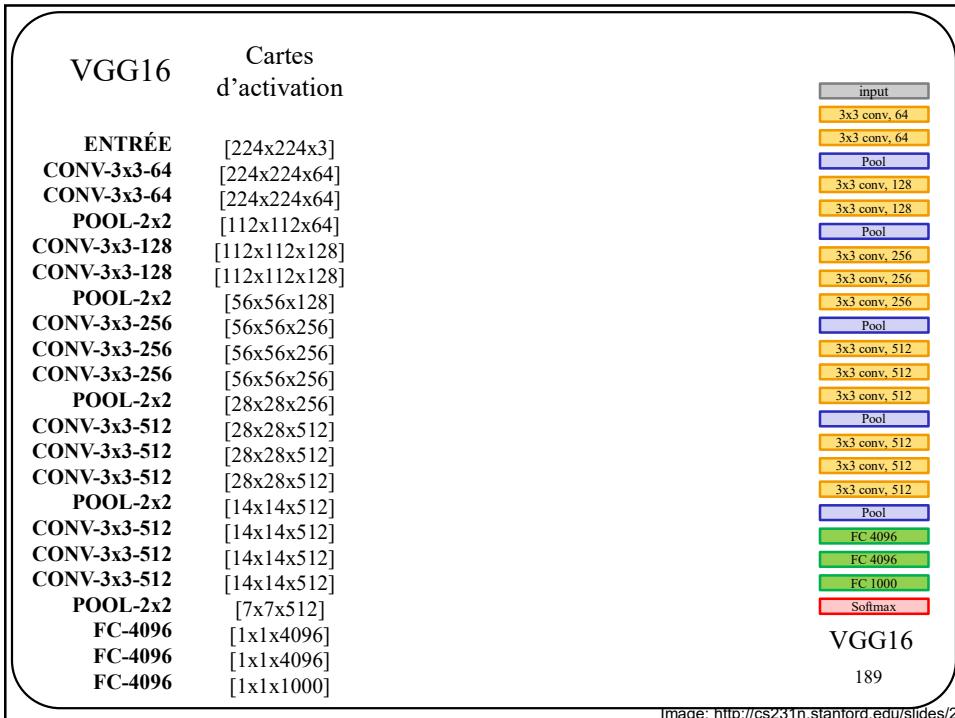


Image: <http://cs231n.stanford.edu/slides/2>

189

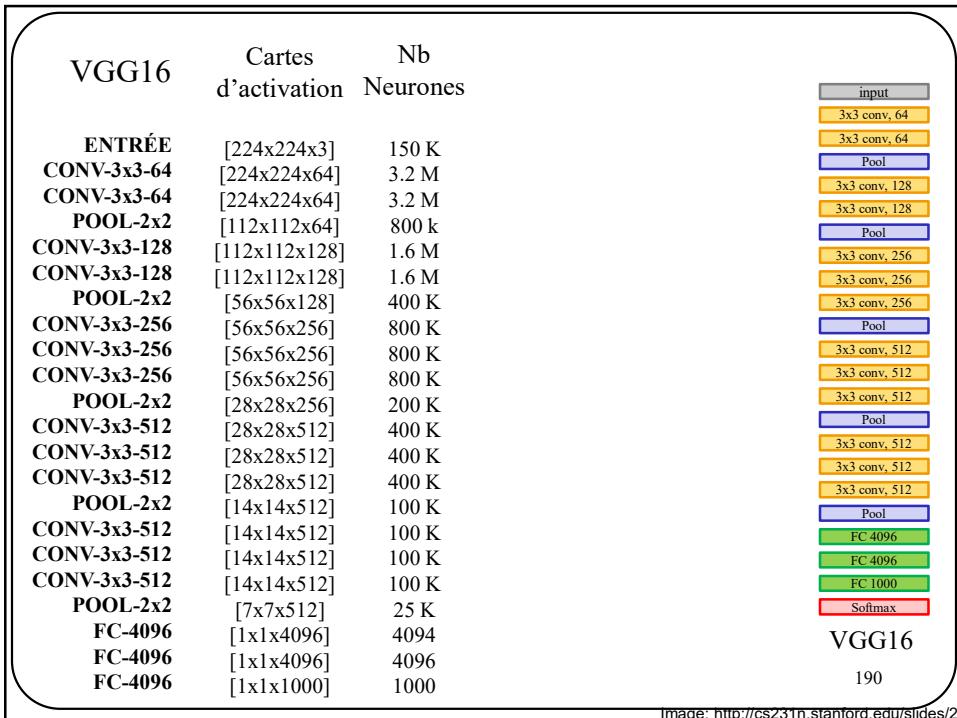


Image: <http://cs231n.stanford.edu/slides/2>

190

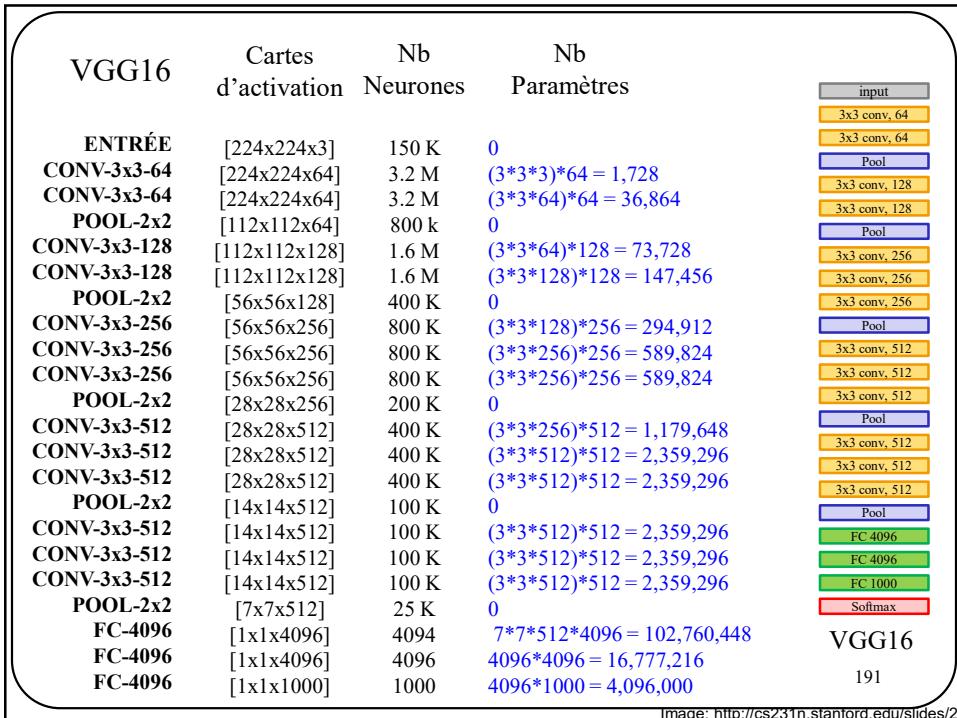


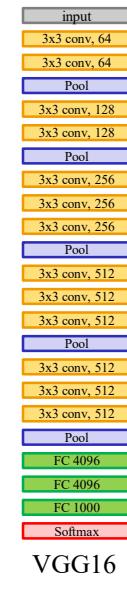
Image: <http://cs231n.stanford.edu/slides/2>

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



VGG16

192

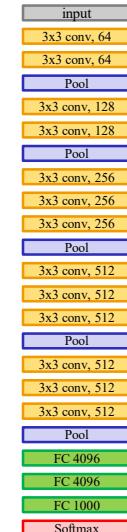
Image: <http://cs231n.stanford.edu/slides/2>

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



VGG16

193

Image: <http://cs231n.stanford.edu/slides/2>

193