Réseaux de neurones

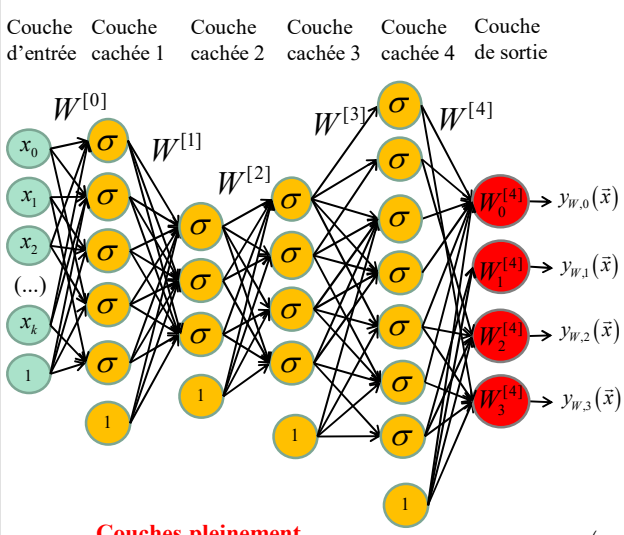
IFT 780

Réseaux à convolution

Par
Pierre-Marc Jodoin

1

kD, 4 Classes, Réseau à 4 couches cachées



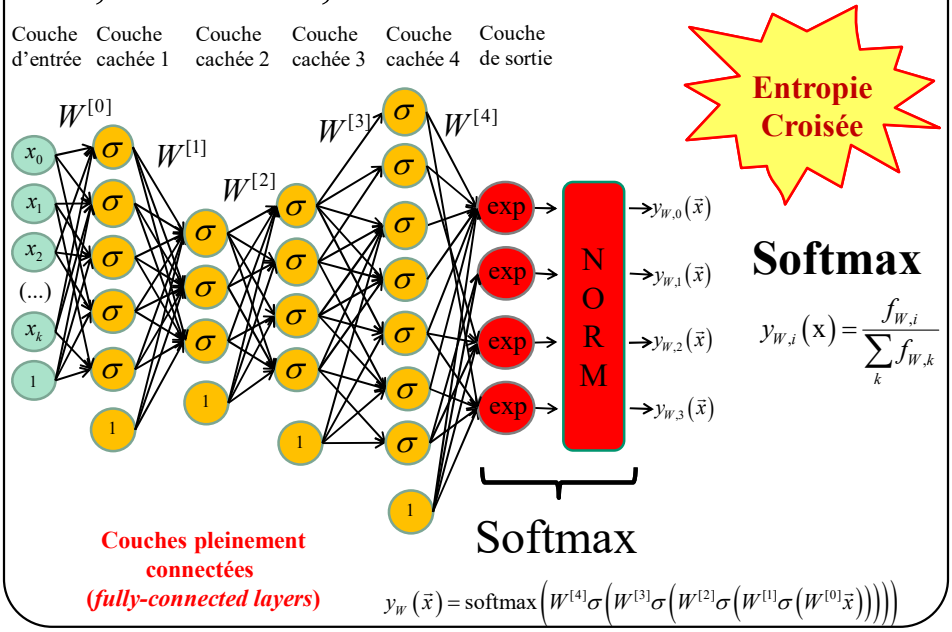
Hinge loss

Couches pleinement connectées
(fully-connected layers)

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

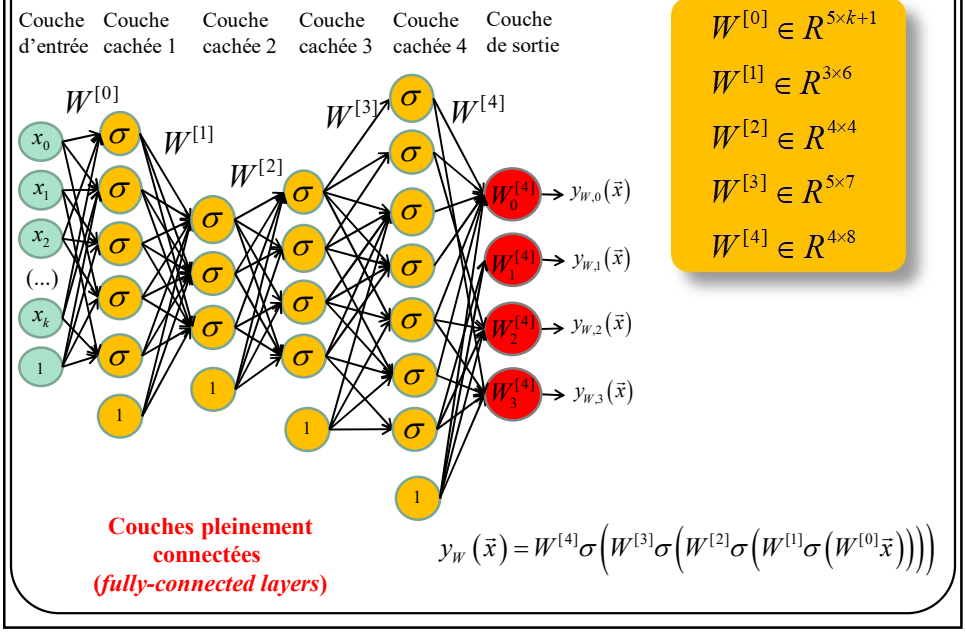
2

kD, 4 Classes, Réseau à 4 couches cachées



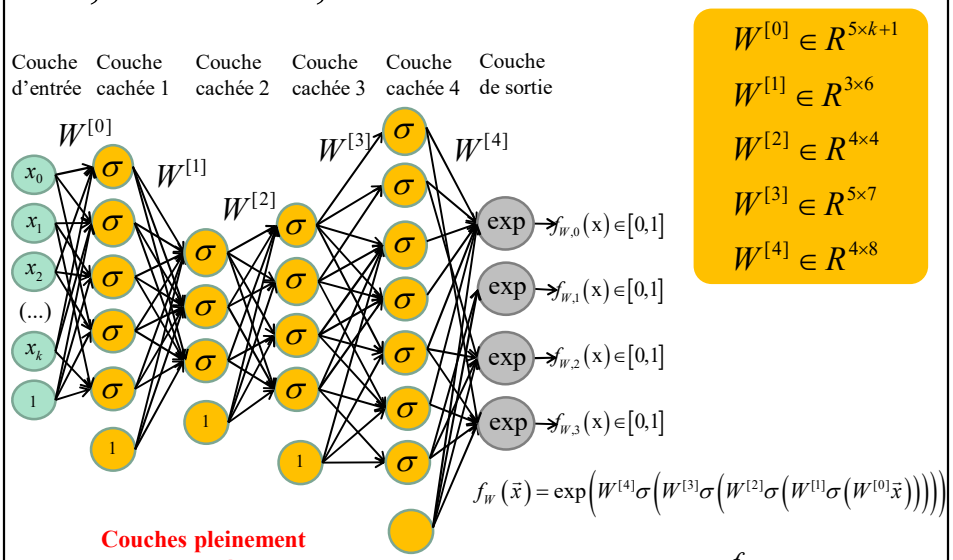
3

kD, 4 Classes, Réseau à 4 couches cachées



4

kD, 4 Classes, Réseau à 4 couches cachées



$W^{[0]} \in R^{5\times k+1}$

$W^{[1]} \in R^{3\times 6}$

$W^{[2]} \in R^{4\times 4}$

$W^{[3]} \in R^{5\times 7}$

$W^{[4]} \in R^{4\times 8}$

Couches pleinement connectées ***(fully-connected layers)***

Softmax $y_{W,i}(\mathrm{x}) = \dfrac{f_{W,i}}{\sum_k f_{W,k}}$

5

Optimisation

Descente de gradient

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

$\eta^{[k]}$: Taux d'apprentissage ou *"learning rate"*.

∇E : Gradient de la function de coût

Descente de gradient stochastique

Initialiser **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== MAX_ITER

Optimisation par *Batch*

Initialiser **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==MAX_ITER

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

6

Les bonnes pratiques

7

Optimisation

Essentiel au TP4

Descente de gradient

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

∇E → Gradient de la function de coût

$\eta^{[k]}$ → Taux d'apprentissage ou "*learning rate*".

Optimisation par **mini-batch**

Initialiser **w**
k=0
FAIRE k=k+1

FAIRE n=0 à N par sauts de ***MBS /*Mini-batch size*/***

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

} **Itération**

JUSQU'À ce que toutes les données soient bien classées ou
k==MAX_ITER

8

Optimisation

Descente de gradient

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

∇E → Gradient de la function de coût

$\eta^{[k]}$ → Taux d'apprentissage ou "*learning rate*".

Optimisation par ***mini-batch***

Initialiser **w**
k=0
FAIRE k=k+1

FAIRE n=0 à N par sauts de ***MBS /*Mini-batch size*/***

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

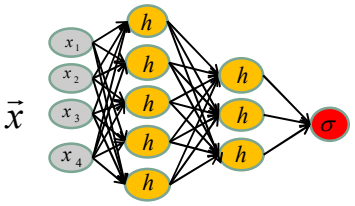
JUSQU'À ce que toutes les données sont bien classées ou
k==MAX_ITER

Epoch

9

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

10

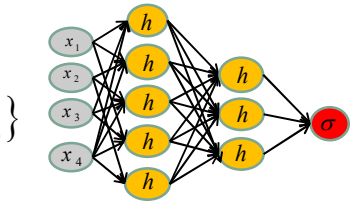


$\vec{x}$ $\sigma\left(\vec{W}_3^T h\left(W_2 h\left(W_1 \vec{x}\right)\right)\right)$

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

	$\vec{x}$	$\in IR^4$
Étape 2 →	$W_1\vec{x}$	$\in IR^5$
	$h\left(W_1\vec{x}\right)$	$\in IR^5$
	$W_2 h\left(W_1\vec{x}\right)$	$\in IR^3$
	$h\left(W_2 h\left(W_1\vec{x}\right)\right)$	$\in IR^3$
Étape 6 →	$\vec{W}_3^T\left(h\left(W_2 h\left(W_1\vec{x}\right)\right)\right)$	$\in IR$
	$\sigma\left(\vec{W}_3^T\left(h\left(\vec{W}_2 h\left(W_1\vec{x}\right)\right)\right)\right)$	$\in IR$

11



$X = \{\vec{x}_1, \vec{x}_2, \vec{x}_3\}$ $Y = \left\{\begin{array}{l}\sigma\left(\vec{W}_3 h\left(W_2 h\left(W_1\vec{x}_1\right)\right)\right),\\ \sigma\left(\vec{W}_3 h\left(W_2 h\left(W_1\vec{x}_2\right)\right)\right),\\ \sigma\left(\vec{W}_3 h\left(W_2 h\left(W_1\vec{x}_3\right)\right)\right)\end{array}\right\}$

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

POUR i allant de 0 à 2

$\vec{x} = X[i]$	$\in IR^4$
$W_1\vec{x}$	$\in IR^5$
$h\left(W_1\vec{x}\right)$	$\in IR^5$
$W_2 h\left(W_1\vec{x}\right)$	$\in IR^3$
$h\left(W_2 h\left(W_1\vec{x}\right)\right)$	$\in IR^3$
$\vec{W}_3^T\left(h\left(W_2 h\left(W_1\vec{x}\right)\right)\right)$	$\in IR$
$Y[i] = \sigma\left(\vec{W}_3^T\left(h\left(\vec{W}_2 h\left(W_1\vec{x}\right)\right)\right)\right)$	$\in IR$

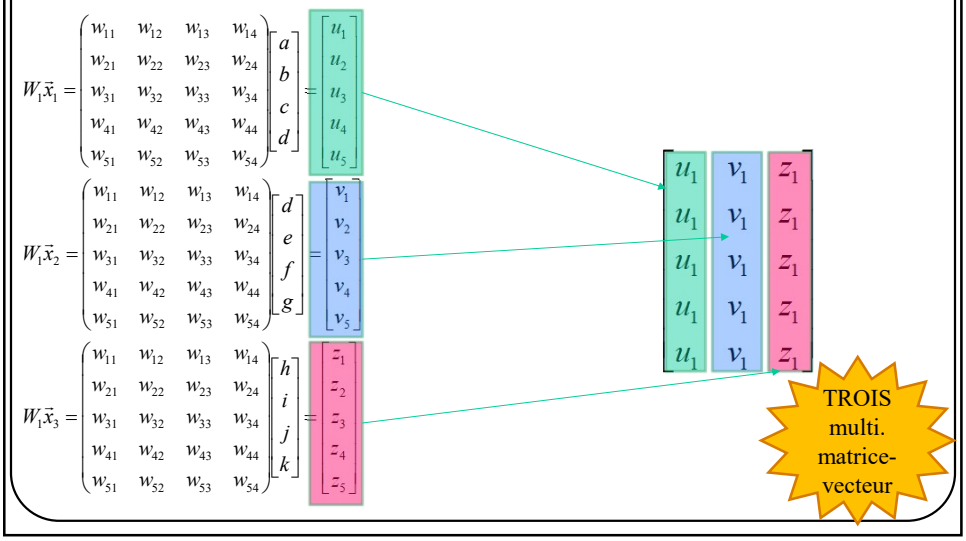
Solution naïve et peu efficace

TP3

12

Solution

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



13

Solution

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

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

Solution

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

$$\begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix}\begin{pmatrix} a \\ b \\ c \end{pmatrix} = \left(w_1 a + w_2 b + w_3 c\right)$$

$$\begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix}\begin{pmatrix} d \\ e \\ f \end{pmatrix} = \left(w_1 d + w_2 e + w_3 f\right)$$

$$\begin{pmatrix} w_1 & w_2 & w_3 \end{pmatrix}\begin{pmatrix} g \\ h \\ i \end{pmatrix} = \left(w_1 g + w_2 h + w_3 i\right)$$

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

TROIS produits scalaires

15

Solution

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

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

UNE multiplication matricielle

16

Conclusion

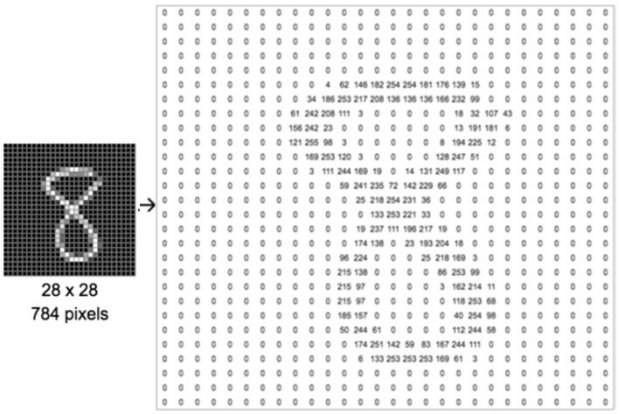
100% du temps, on combine ensemble les données dans des **mini-batch** de 2 à 32 données.

17

Les réseaux à convolution

18

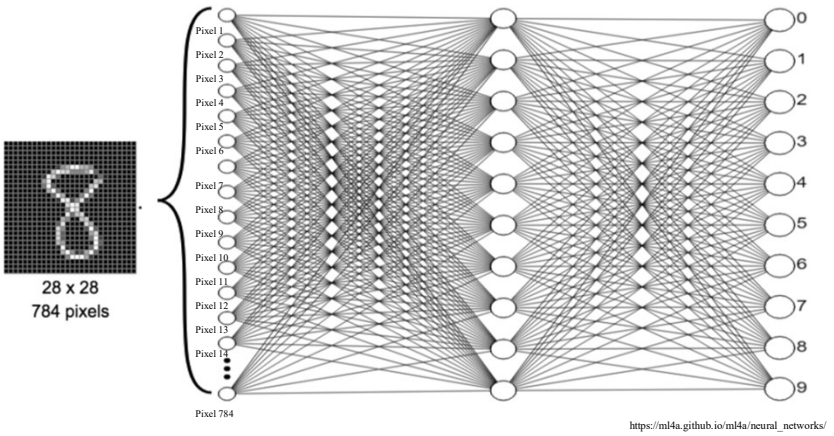
Comment classifier des images?



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

19

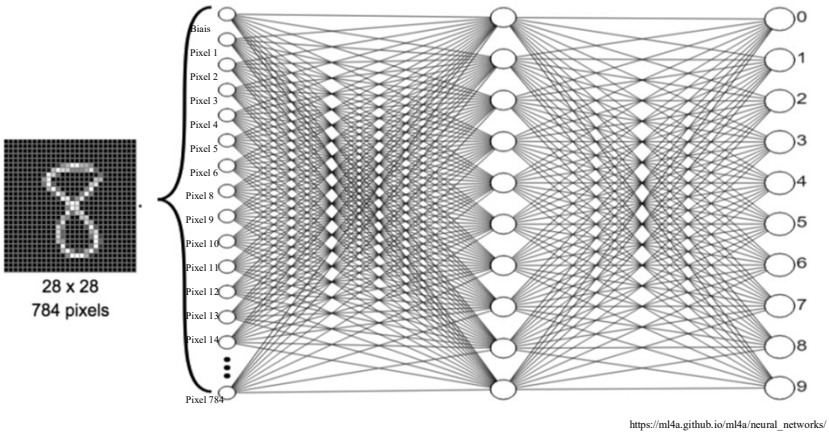
Comment classifier des images?



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

20

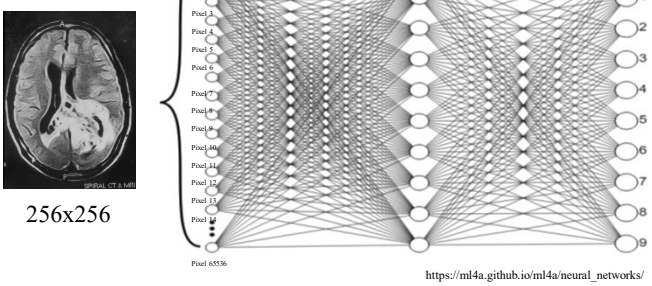
Beaucoup de paramètres (7850 dans la couche 1)



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

21

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



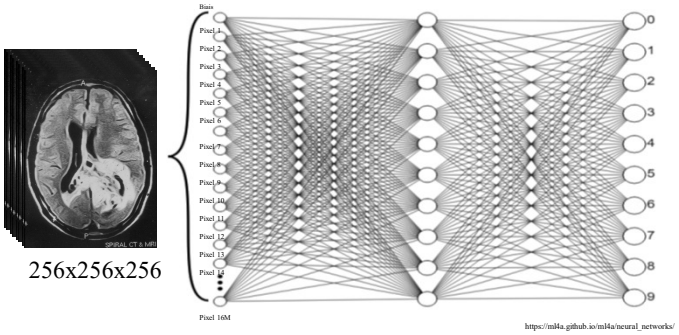
256x256

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

Image médicale (IRM de cerveau)

22

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



256x256x256

Image médicale 3D (IRM de cerveau)

23

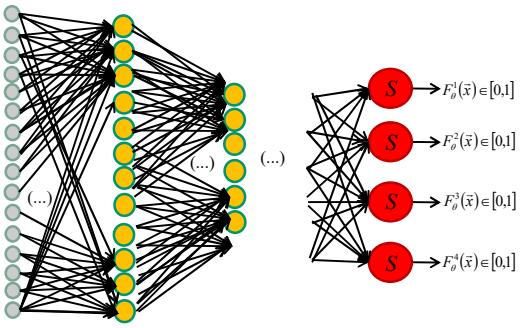
Comment réduire le nombre de connections?

24

24

Comment réduire le nombre de connections?

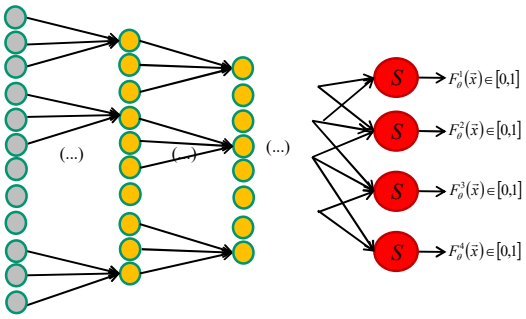
Les **couches pleinement connectées (*fully-connected layers*)** sont problématiques lorsque le **nombre de neurones est élevé.**



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

25

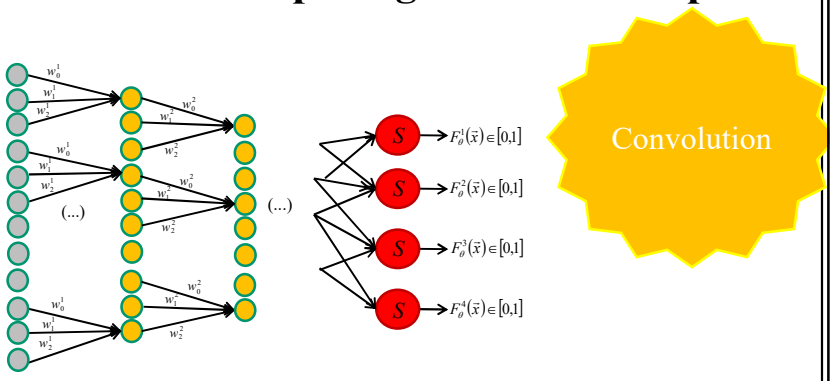
Solution : connexions partielles



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!

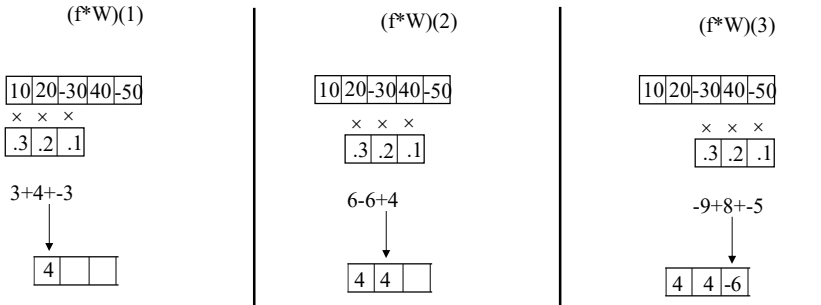
27

Convolution et couche convolutionnelle **1D**

28

Exemple 1D de la convolution

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



29

29

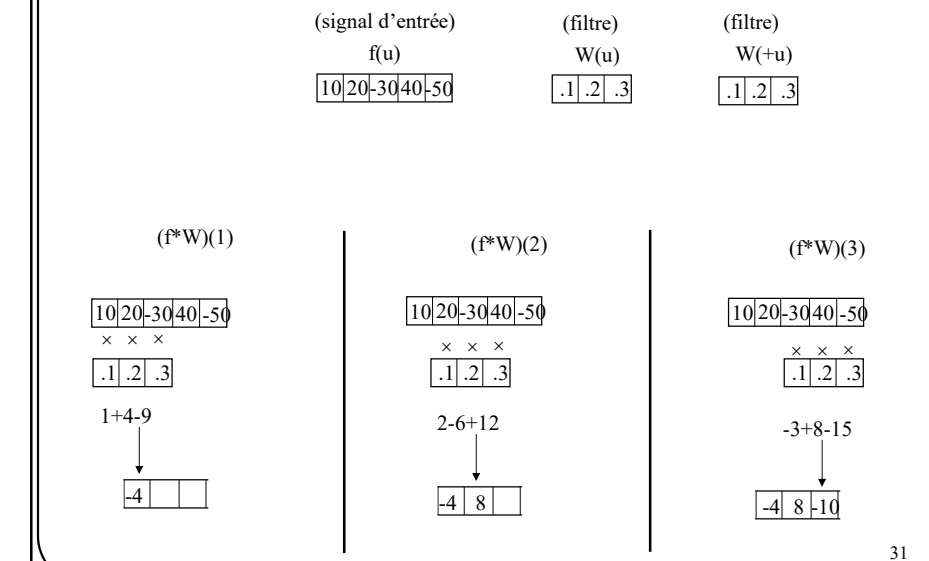
En gros

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

30

La convolution des réseaux de neurones = corrélation

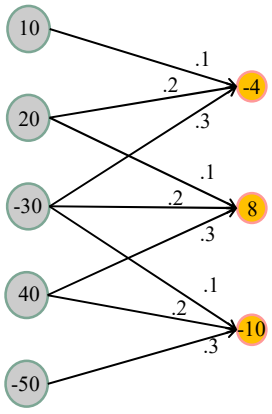
$$(f * W)(v) = \sum_{u=-\infty}^{\infty} f(u) W(v+u)$$



31

31

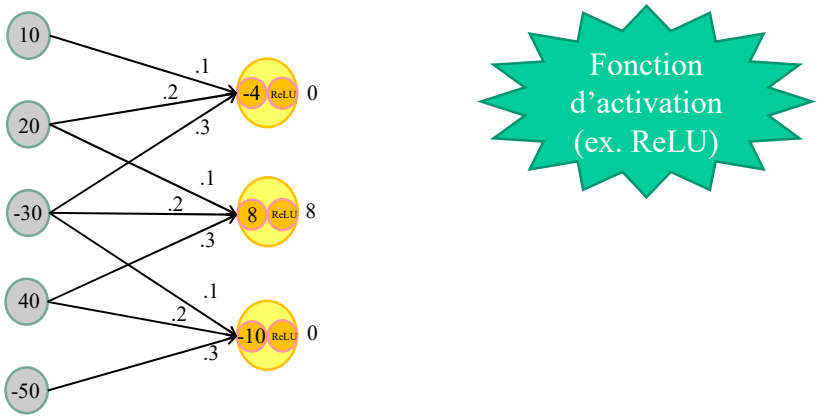
L'opération de la page précédente est équivalente à



32

32

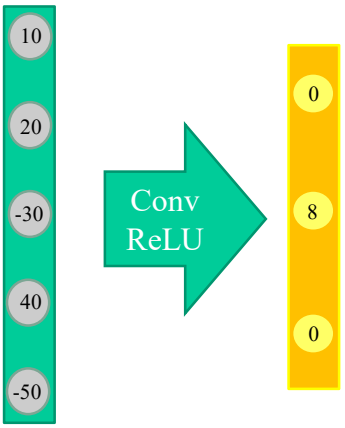
L'opération de la page précédente est équivalente à



33

33

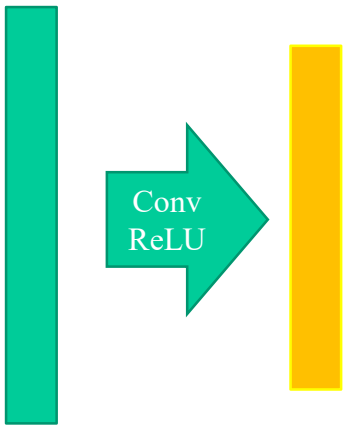
Représentation graphique courante (simple)



34

34

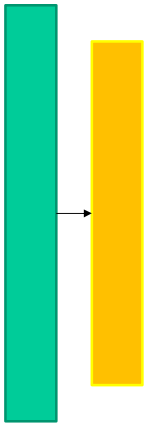
Représentation graphique courante (encore plus simple)



35

35

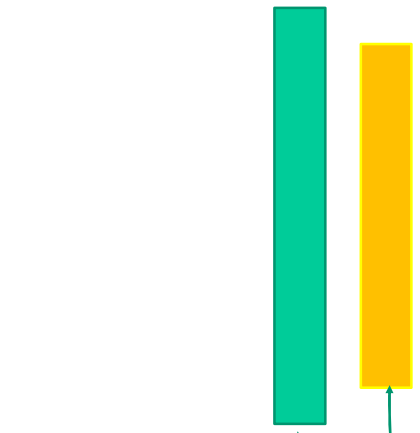
Représentation graphique courante (vraiment ultra simple)



36

36

Représentation graphique courante (eehhh…)



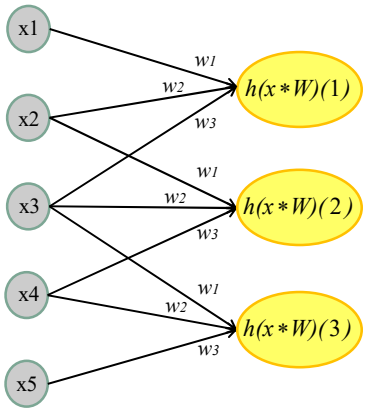
Vecteur d'entrée

Carte d'activation
(feature map)

37

37

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

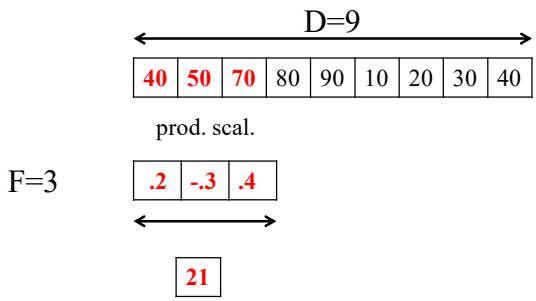


h: fonction d'activation
(tanh, ReLU, elu, etc.)

38

38

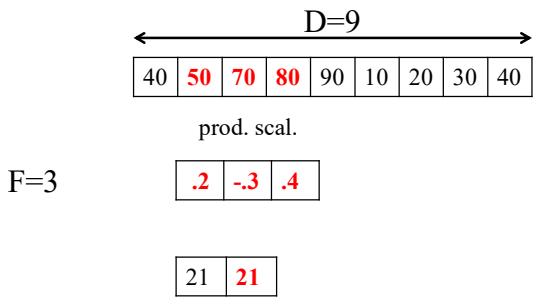
Stride et calcul de la taille de la carte d'activation



39

39

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

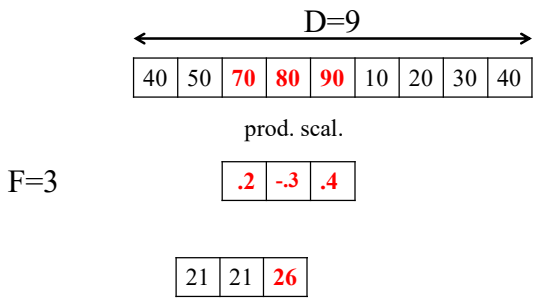


Stride = 1

40

40

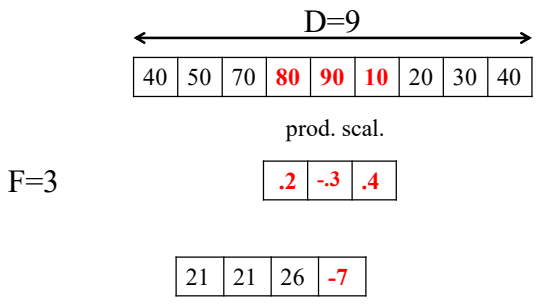
Stride et calcul de la taille de la carte d'activation



41

41

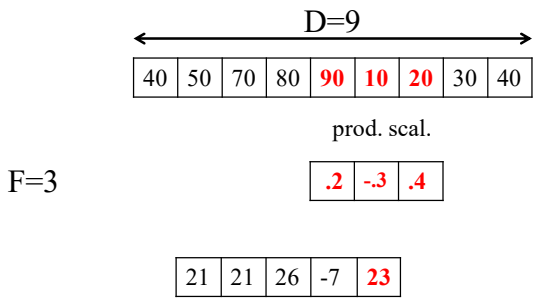
Stride et calcul de la taille de la carte d'activation



42

42

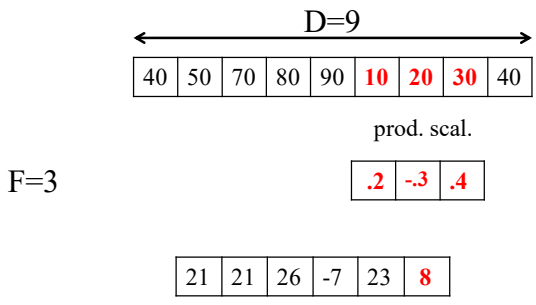
Stride et calcul de la taille de la carte d'activation



43

43

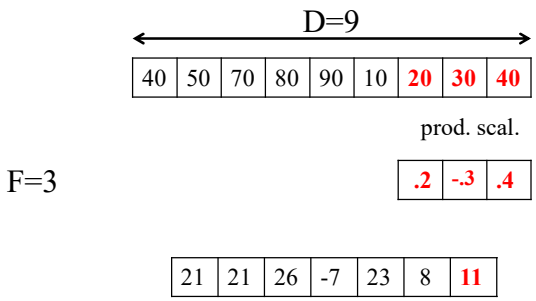
Stride et calcul de la taille de la carte d'activation



44

44

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

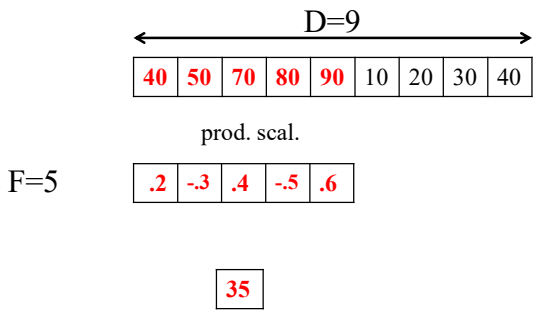


Taille de la carte d'activation = **7**

45

45

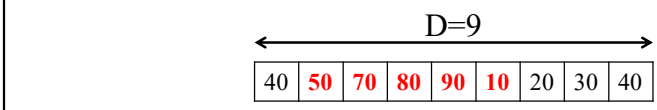
Stride et calcul de la taille de la carte d'activation



46

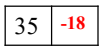
46

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



prod. scal.

F=5

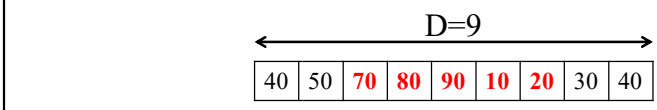


Stride = 1

47

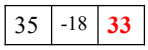
47

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



prod. scal.

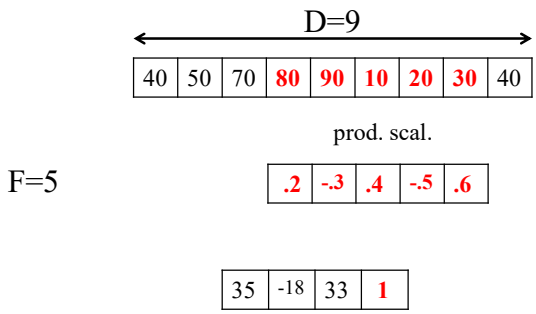
F=5



48

48

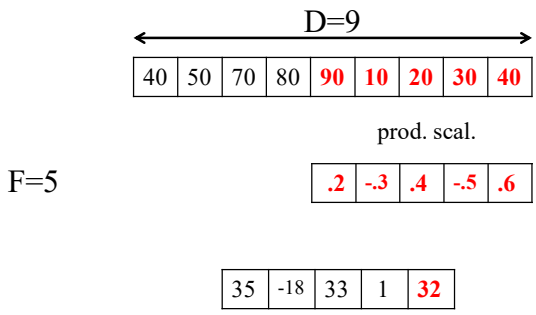
Stride et calcul de la taille de la carte d'activation



49

49

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

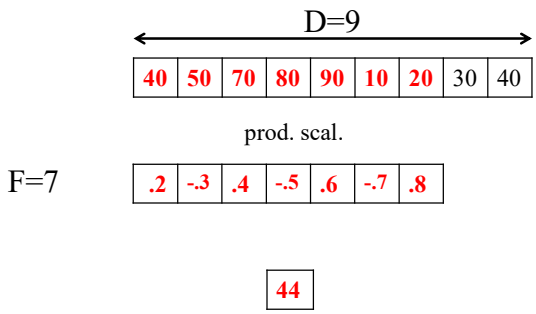


Taille de la carte d'activation = **5**

50

50

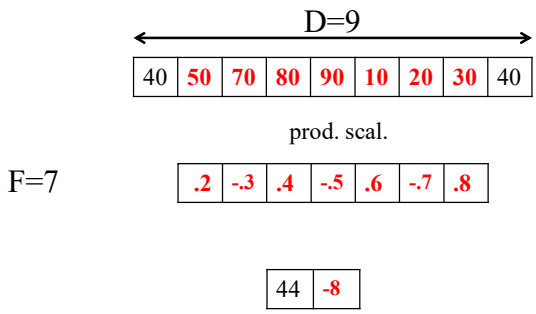
Stride et calcul de la taille de la carte d'activation



51

51

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

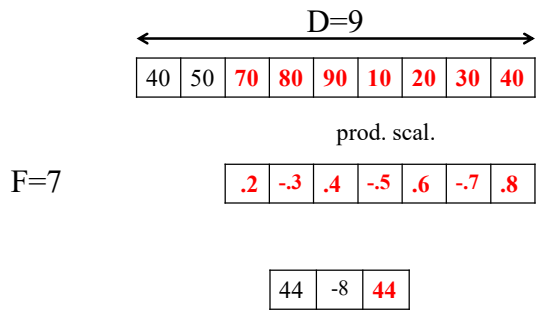


Stride = 1

52

52

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

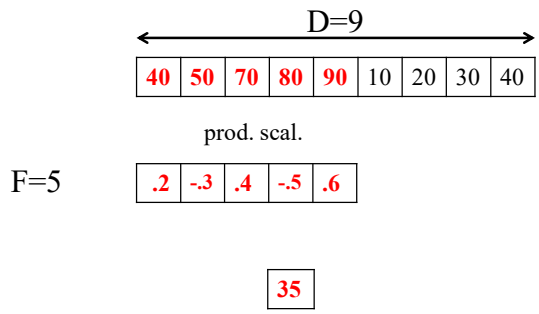


Taille de la carte d'activation = **3**

53

53

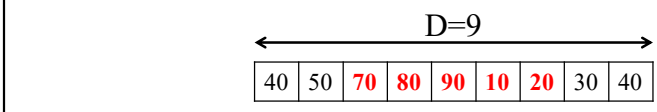
Stride et calcul de la taille de la carte d'activation



54

54

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



prod. scal.

F=5

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

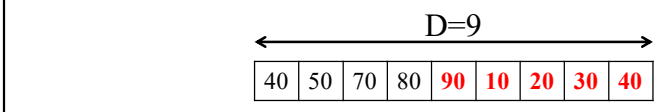
35	33

Stride = 2

55

55

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



prod. scal.

F=5

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

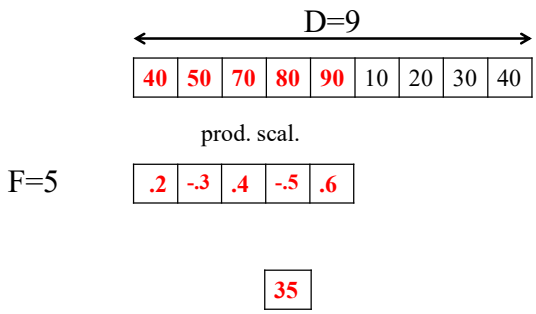
35	33	32

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

56

56

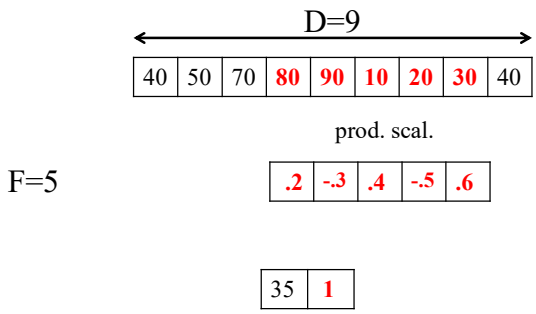
Stride et calcul de la taille de la carte d'activation



57

57

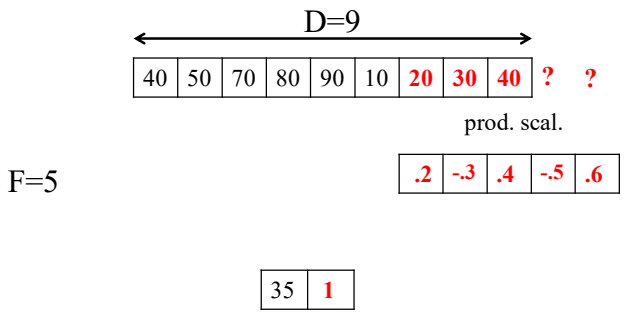
Stride et calcul de la taille de la carte d'activation



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

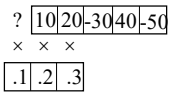
Taille de la carte d'activation = **(D-F)/S+1**

60

60

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

Comment gérer les bords?



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

f(u)

0	10	20	-30	40	-50	0

(f*W)(u)

8	-4	8	-10	-6

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

f(u)

20	10	20	-30	40	-50	40

(f*W)(u)

10	-4	8	-10	2

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

f(u)

10	10	20	-30	40	-50	-50

(f*W)(u)

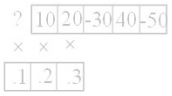
9	-4	8	-10	-21

61

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)

0	10	20	-30	40	-50	0

(f*W)(u)

8	-4	8	-10	-6

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

f(u)

10	10	20	-30	40	-50	-50

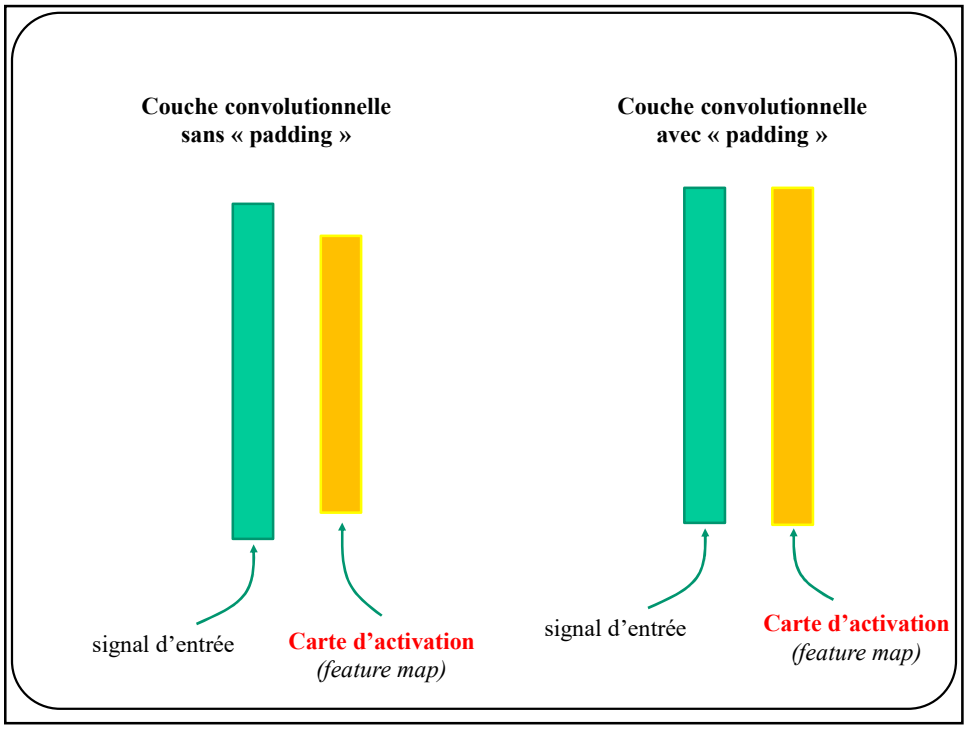
(f*W)(u)

9	-4	8	-10	-21

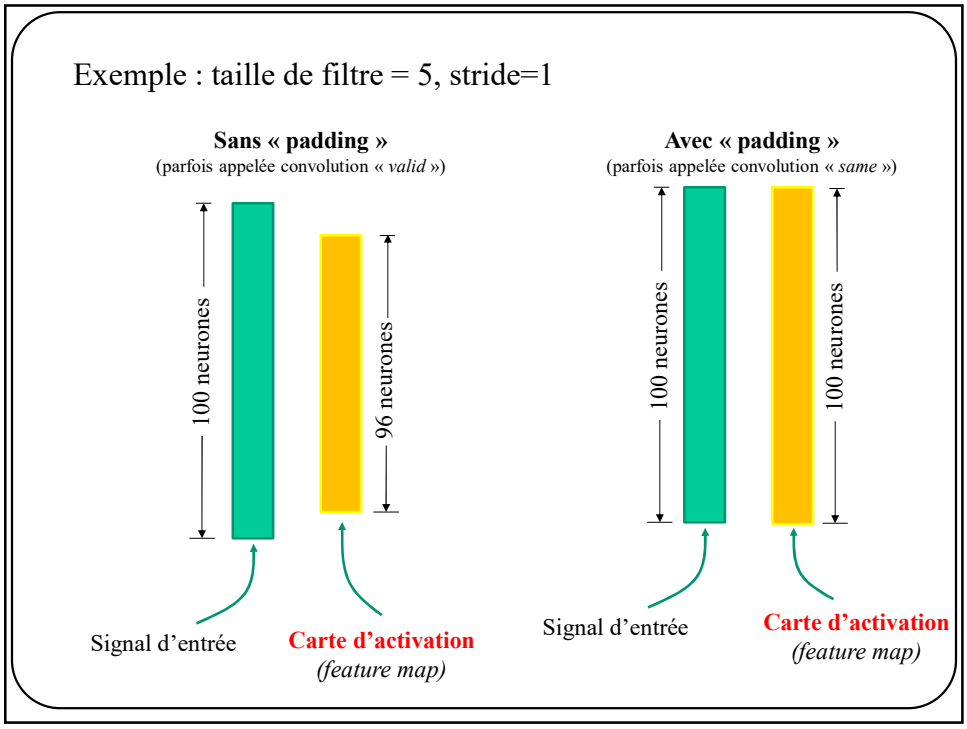
De loin l'option la plus utilisée

62

62


Couche convolutionnelle
sans « padding »
Couche convolutionnelle
avec « padding »
signal d'entrée
Carte d'activation
(feature map)
signal d'entrée
Carte d'activation
(feature map)

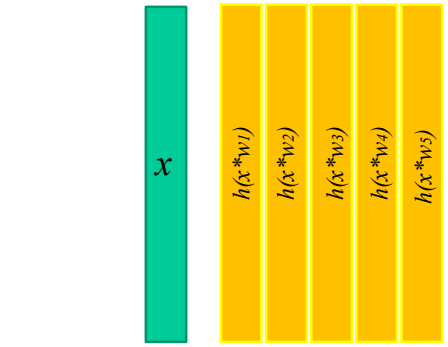
63


Exemple : taille de filtre = 5, stride=1
Sans « padding »
(parfois appelée convolution « valid »)
Avec « padding »
(parfois appelée convolution « same »)
100 neurones
96 neurones
100 neurones
100 neurones
Signal d'entrée
Carte d'activation
(feature map)
Signal d'entrée
Carte d'activation
(feature map)

64

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

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



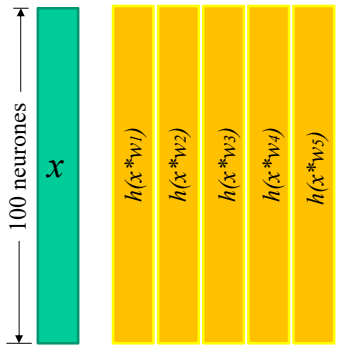
Signal d'entrée

Cartes d'activation
(feature maps)

65

Taille de filtre = 5

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



Avec « *padding* »

Combien de neurones et de paramètres au total?

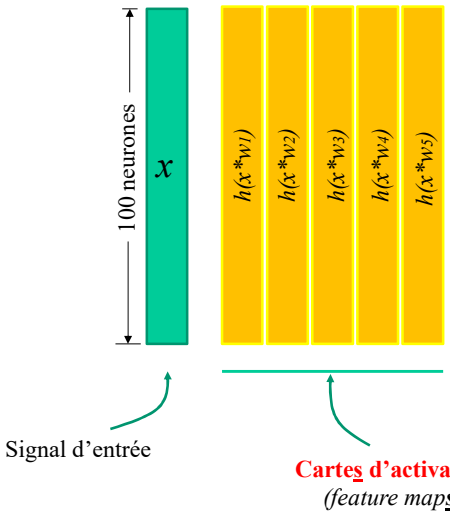
Signal d'entrée

Cartes d'activation
(feature maps)

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 (5x5) paramètres au total

67

Convolution et couche convolutionnelle **2D**

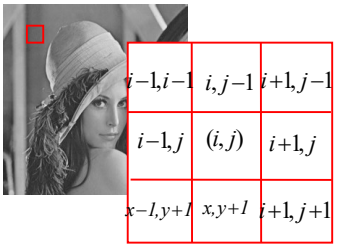
68

Filtage 2D

(sans flip de filtre)

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

$x(i,j)$



$W(u,v)$

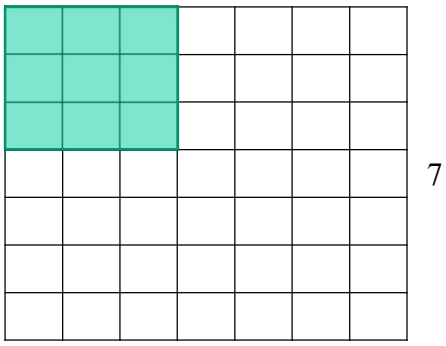
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

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

69

Convolution 2D

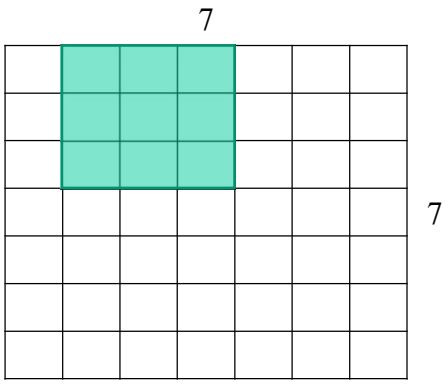
Filtre = 3x3
Stride = 1



70

Convolution 2D

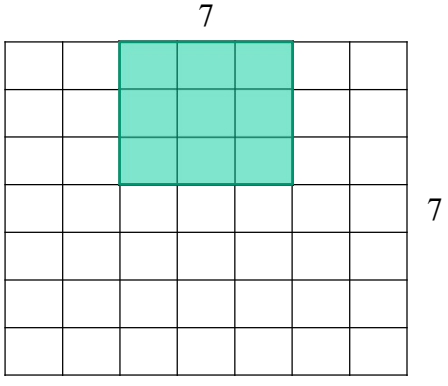
Filtre = 3x3
Stride = 1



71

Convolution 2D

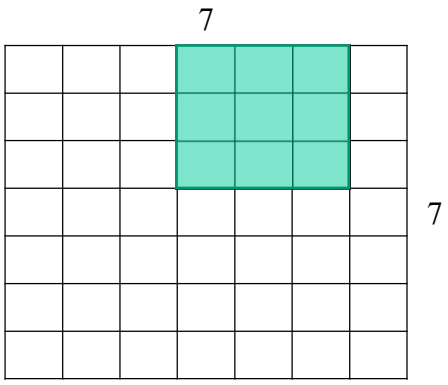
Filtre = 3x3
Stride = 1



72

Convolution 2D

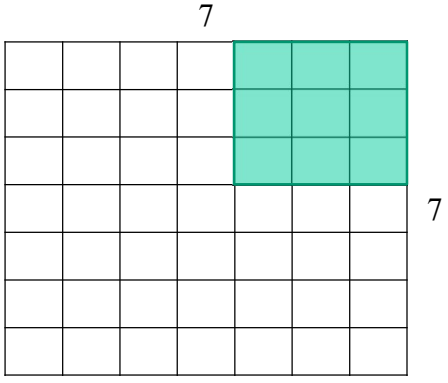
Filtre = 3x3
Stride = 1



73

Convolution 2D

Filtre = 3x3
Stride = 1

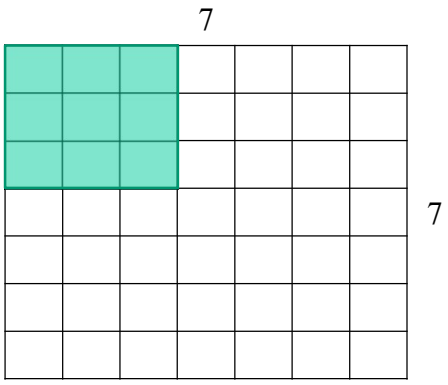


Taille de la carte d'activation (pour stride 1) = **5x5**

74

Convolution 2D

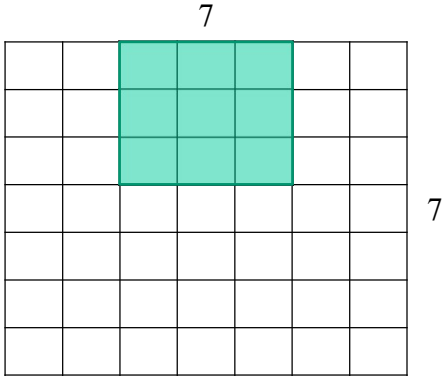
Filtre = 3x3
Stride = 2



75

Convolution 2D

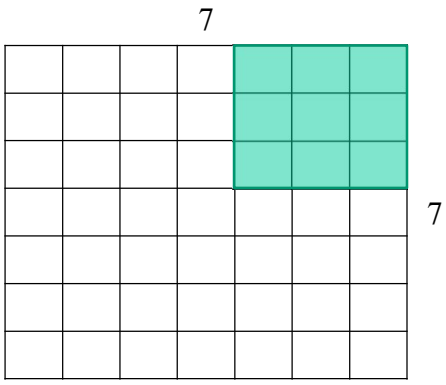
Filtre = 3x3
Stride = 2



76

Convolution 2D

Filtre = 3x3
Stride = 2

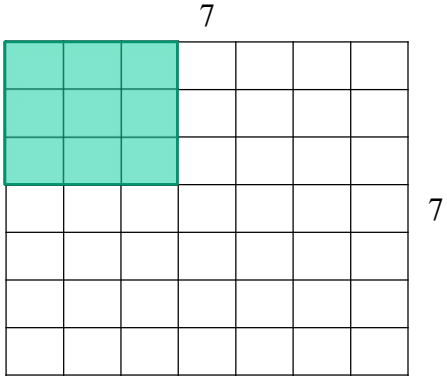


Taille de la carte d'activation (pour stride 2) = **3x3**

77

Convolution 2D

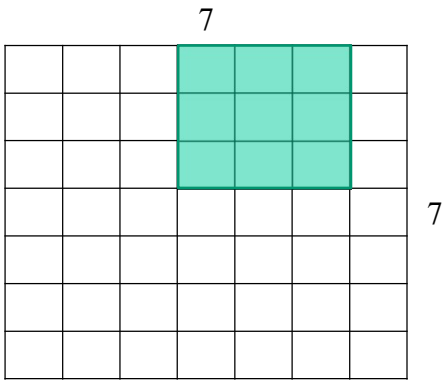
Filtre = 3x3
Stride = 3



78

Convolution 2D

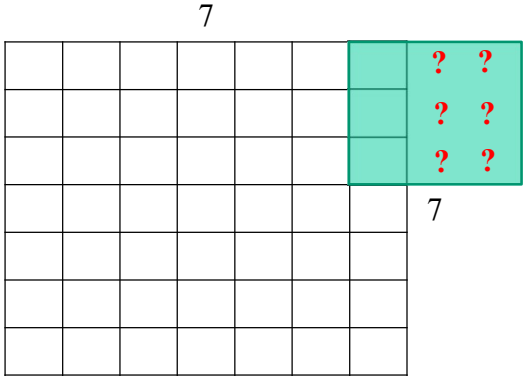
Filtre = 3x3
Stride = 3



79

Convolution 2D

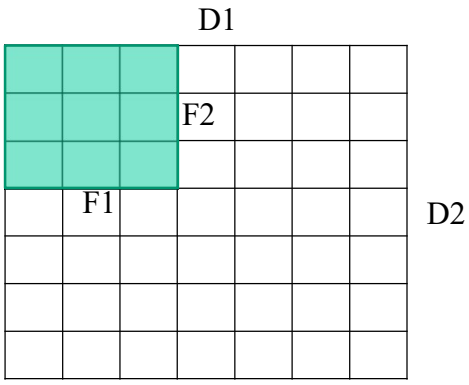
Filtre = 3x3
Stride = 2



Combinaison D-F-S invalide!

80

Convolution 2D

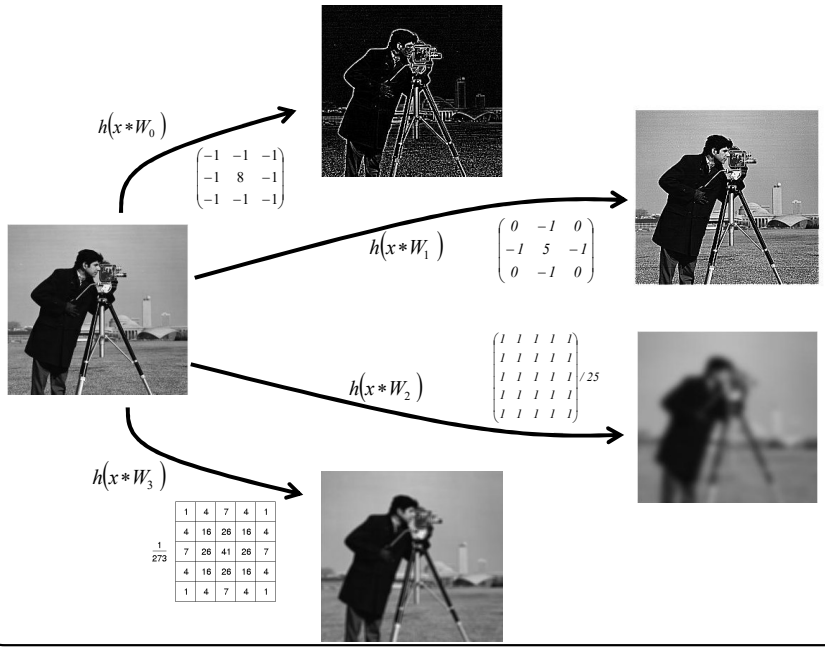


Taille de la carte d'activation :

(D1-F1)/S+1 x (D2-F2)/S+1

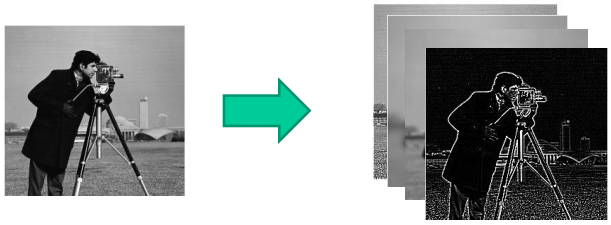
81

Différents filtres = différentes cartes d'activation



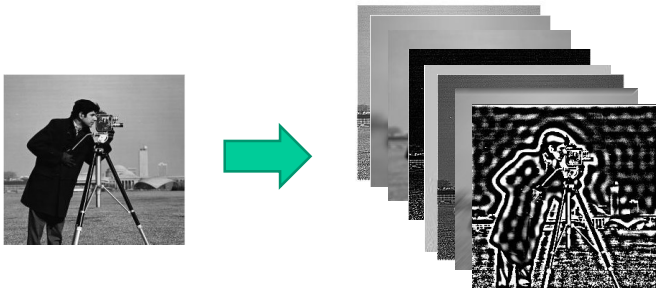
82

4 filtres = Couche convolutive avec **4 cartes d'activation**



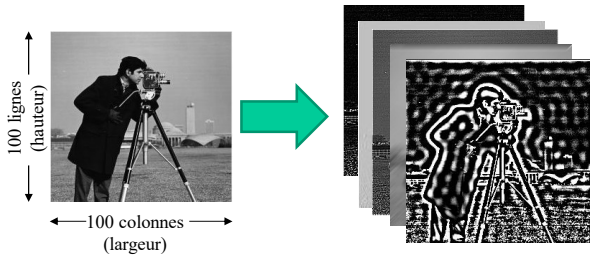
83

K filtres = Couche convolutive avec **K cartes d'activation**



84

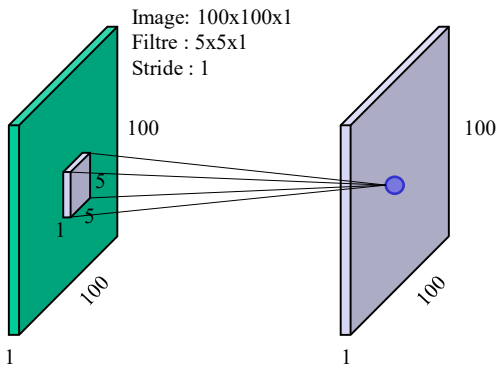
Ex.: taille de filtre : 5x5, 5 cartes d'activation, convolution « same »



- 10,000 neurones par carte d'activation
- 50,000 neurones au total
- 5x5x5 = 125 paramètres au total

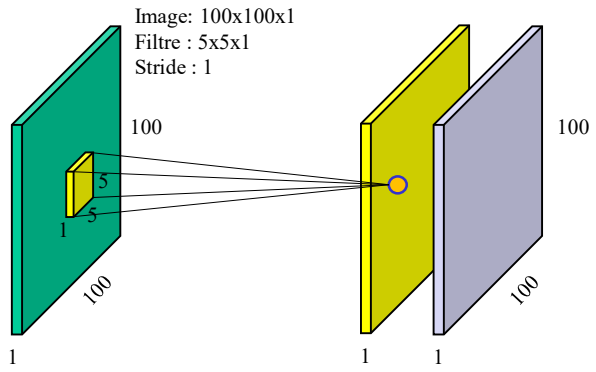
85

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



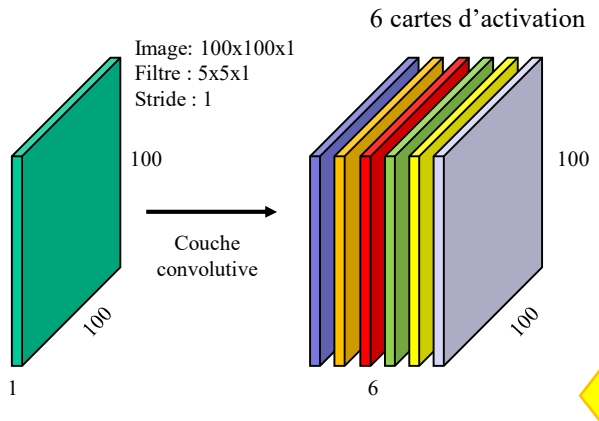
86

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



87

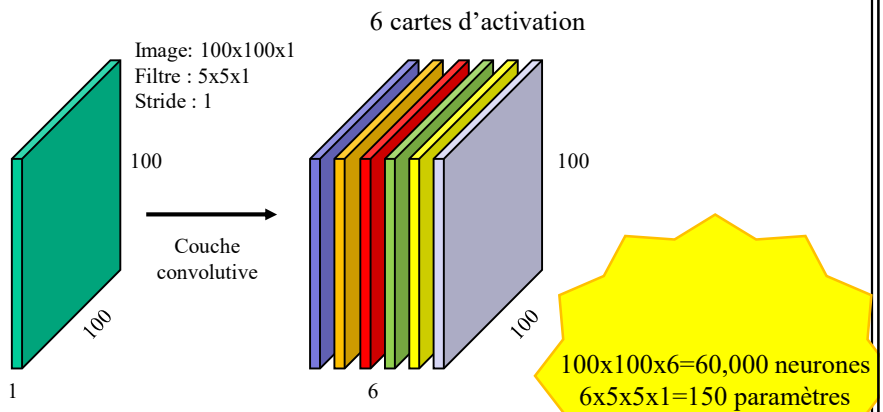
Représentation schématique
(6 filtres et 6 cartes d'activation, convolution « same »)



Combien de neurones et de paramètres au total?

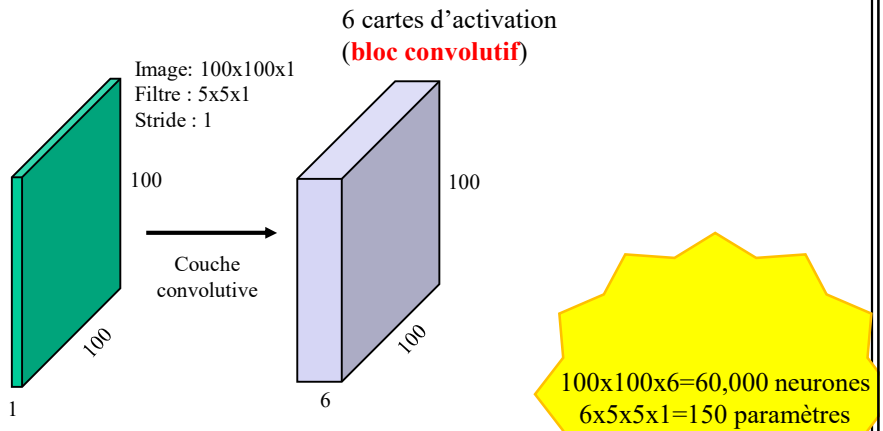
88

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



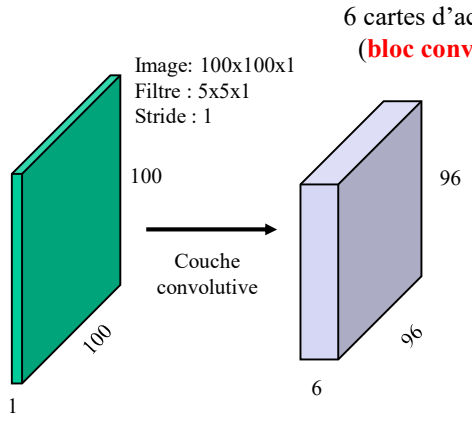
89

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



90

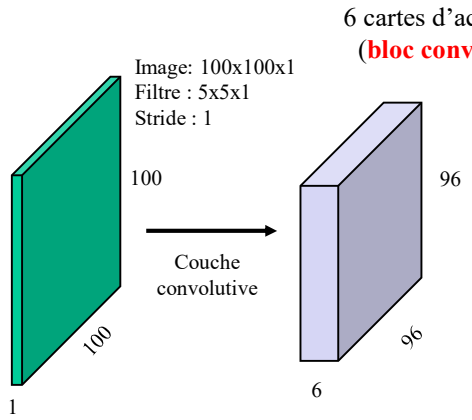
Représentation schématique simplifiée
(6 filtres et 6 cartes d'activation, convolution « ***valid*** »)



Combien de neurones
et de paramètres
au total?

91

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

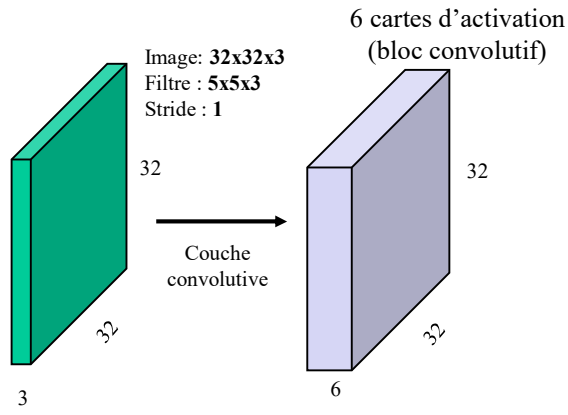


96x96x6=55,296 neurones
6x5x5x1=150 paramètres

92

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

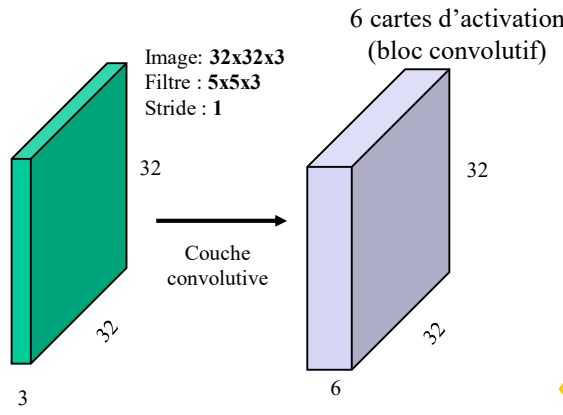
Exemples cifar10



Combien de neurones
et de paramètres
au total?

93

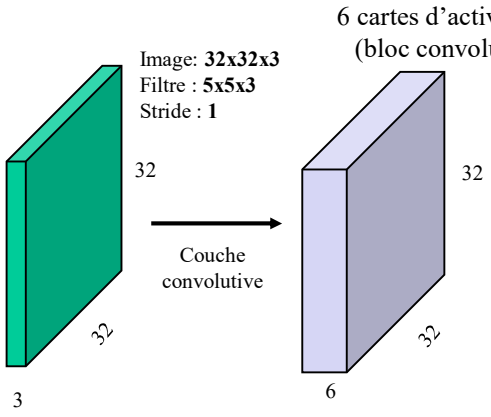
Représentation schématique **images couleur**
(ex.: images RGB de CIFAR10
convolution « *same* »)



32x32x6=6,144 neurones
6x5x5x**3**=**450** paramètres

94

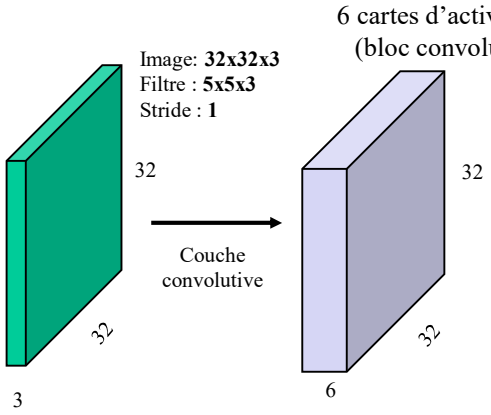
Représentation schématique **images couleur**
(ex.: images RGB de CIFAR10
convolution « *same* »)



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

95

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

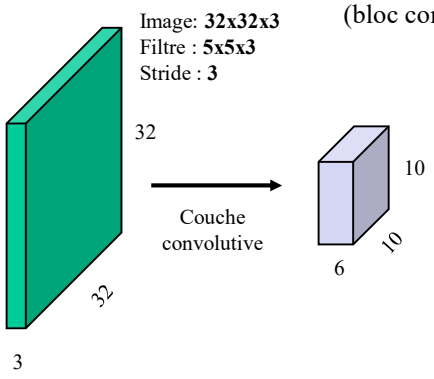


(D-F)/S+1
=
(32-5)/3+1=10

96

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

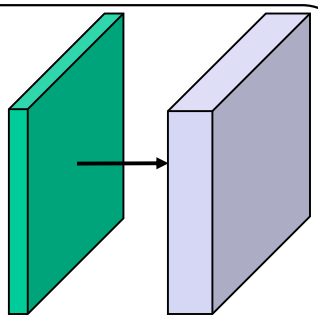
6 cartes d'activation
(bloc convolutif)



10x10x6=**600 neurones**
6x5x5x**3**=**450** paramètres

97

Exemple

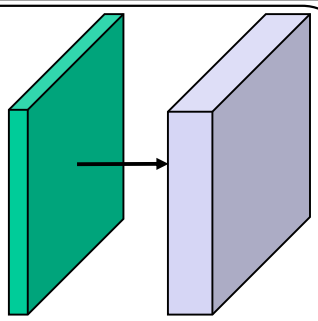


Volume en entrée : **32 x 32 x 3**
10 filtres 5x5 avec **stride = 1**
et convolution « ***same*** »

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

98

Exemple



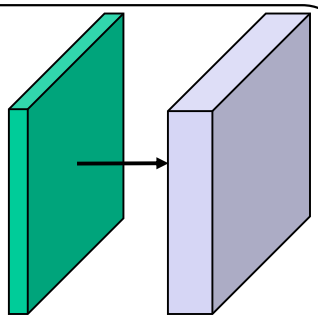
Volume en entrée : **32 x 32 x 3**
10 filtres 5x5 avec **stride = 1**
et convolution « ***same*** »

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

Chaque filtre a **5x5x3** = **75** paramètres
Comme il y a **10 filtres** : **750** paramètres

99

Exemple



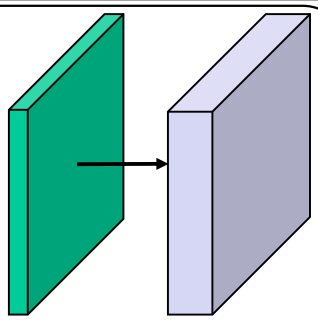
Volume en entrée : **32 x 32 x 3**
10 filtres 5x5 avec **stride = 1**
et convolution « ***same*** ».

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

Chaque filtre a **5x5x3+1** = **76** paramètres (+1 pour le biais)
Comme il y a **10 filtres** : **760** paramètres

100

Exemple

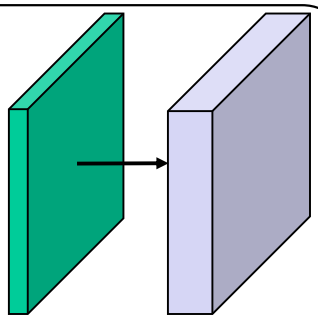


Volume en entrée : **32 x 32 x 3**
10 filtres 5x5 avec **stride = 1**
et convolution « ***valid*** »

Combien de paramètres dans cette couche?

101

Exemple



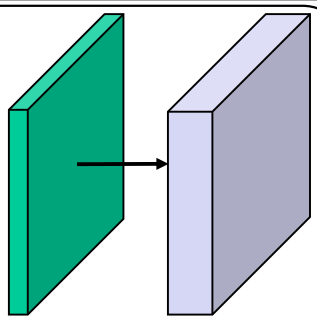
Volume en entrée : **32 x 32 x 3**
10 filtres 5x5 avec **stride = 1**
et convolution « ***valid*** »

Combien de paramètres dans cette couche?

Même chose, cela ne change pas la conformité des filtres

102

Exemple

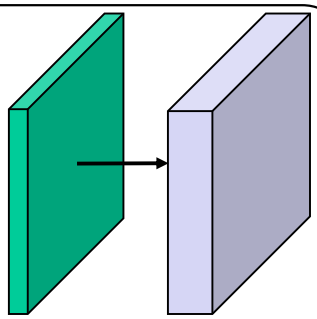


Volume en entrée : **32 x 32 x 3**
10 filtres 5x5 avec **stride = 1**
et convolution « ***valid*** »

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

103

Exemple



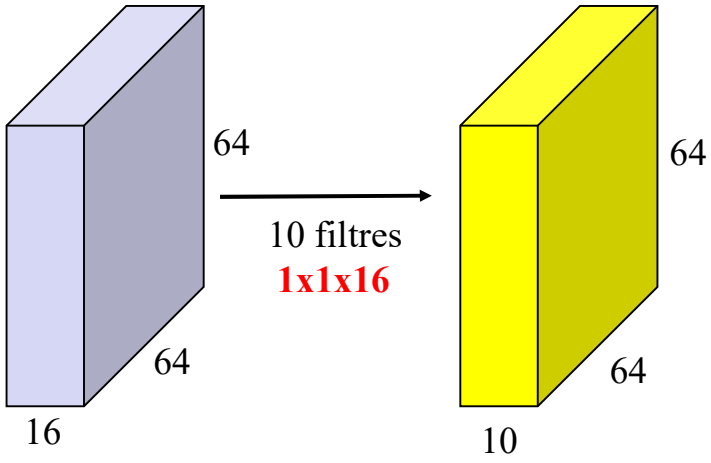
Volume en entrée : **32 x 32 x 3**
10 filtres 5x5 avec **stride = 1**
et convolution « ***valid*** »

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

(32-5+1) x (32-5+1) x 10 = 7,840

104

Des filtres 1x1? Oui ça marche



105

Exemple simple d'un filtre 1x1

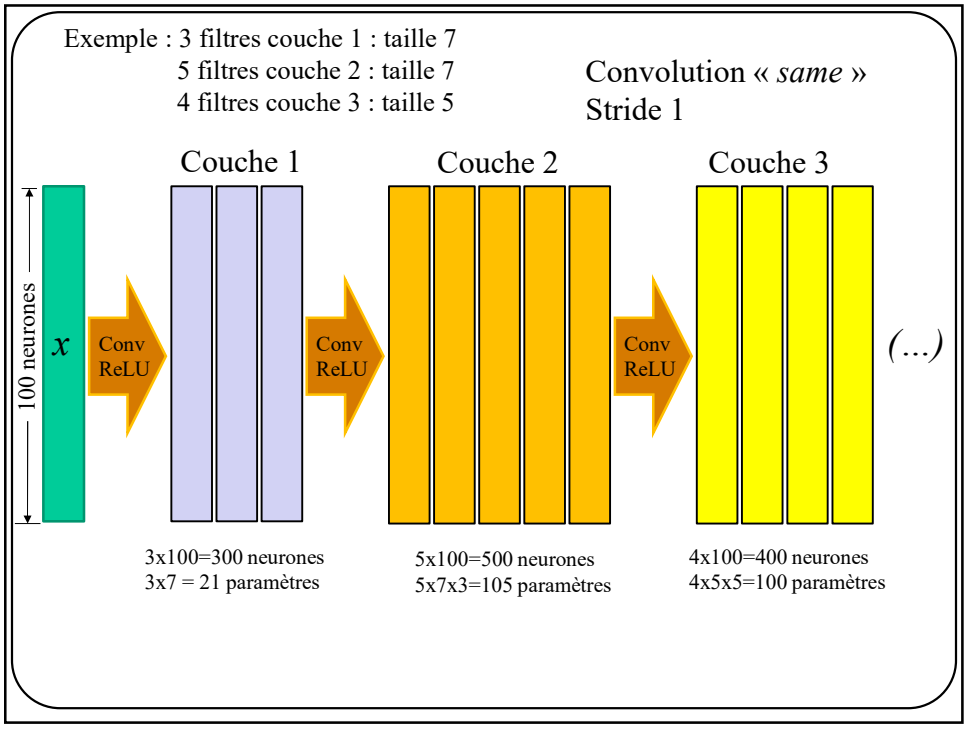
$$\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right]$$

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

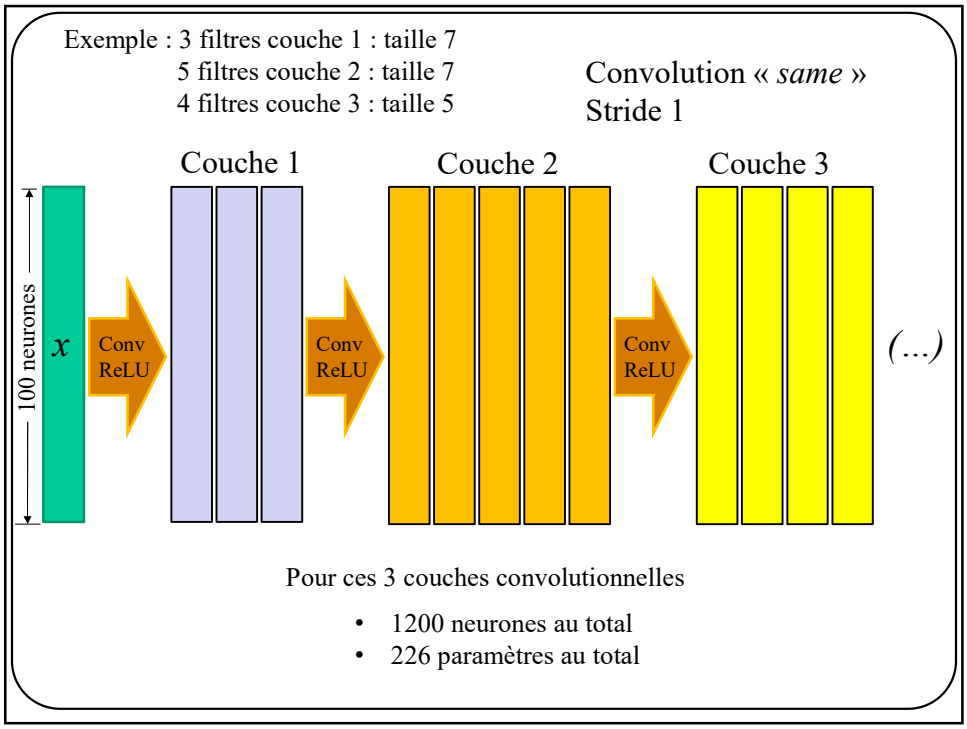
106

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

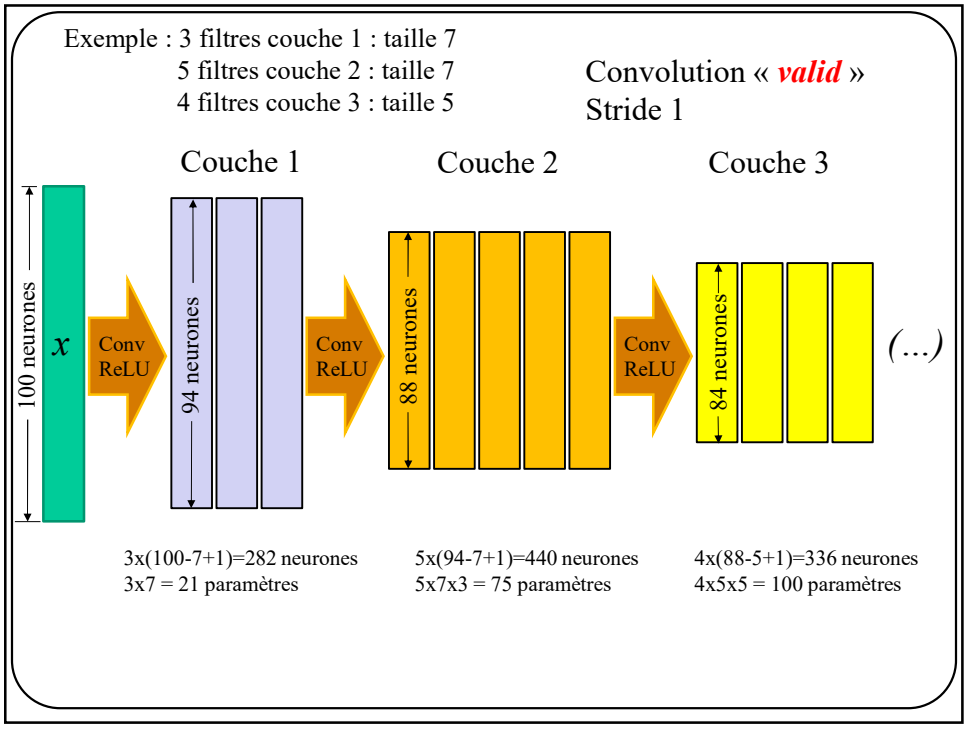
107



108


Exemple : 3 filtres couche 1 : taille 7
5 filtres couche 2 : taille 7
4 filtres couche 3 : taille 5
Convolution « same »
Stride 1
Couche 1
Couche 2
Couche 3
100 neurones
x
Conv
ReLU
Conv
ReLU
Conv
ReLU
(…)
Pour ces 3 couches convolutionnelles
• 1200 neurones au total
• 226 paramètres au total

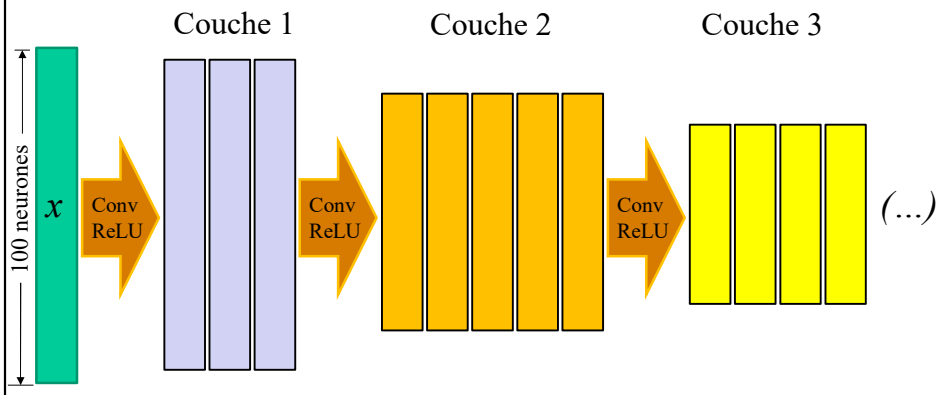
109


Exemple : 3 filtres couche 1 : taille 7
5 filtres couche 2 : taille 7
4 filtres couche 3 : taille 5
Convolution « valid »
Stride 1
Couche 1
Couche 2
Couche 3
100 neurones
x
Conv
ReLU
94 neurones
Conv
ReLU
88 neurones
Conv
ReLU
84 neurones
(…)
3x(100-7+1)=282 neurones
3x7 = 21 paramètres
5x(94-7+1)=440 neurones
5x7x3 = 75 paramètres
4x(88-5+1)=336 neurones
4x5x5 = 100 paramètres

110

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

Convolution « ***valid*** »
Stride 1



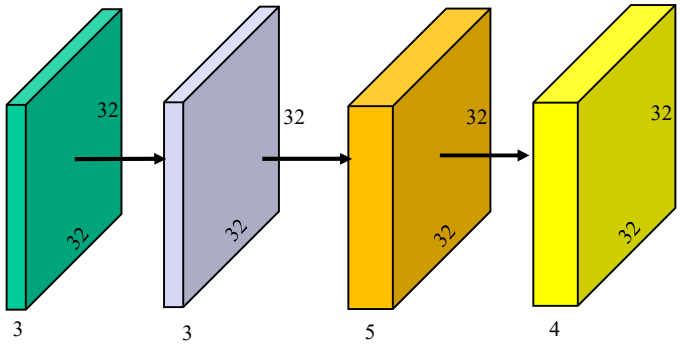
Pour ces 3 couches convolutionnelles

- 1066 neurones au total
- 226 paramètres au total

111

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**



3x(32x32)=3,072 neurones
3x7x7x3 = 441 paramètres

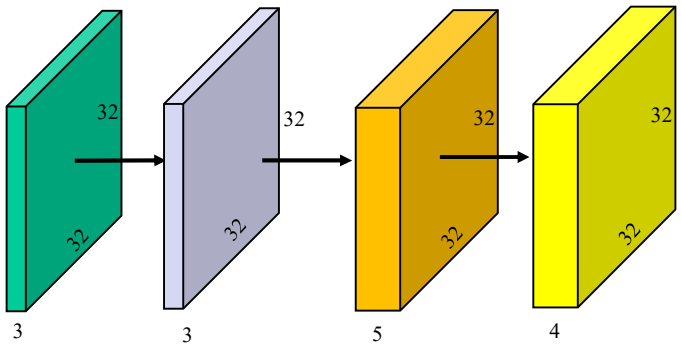
5x(32x32)=5,120 neurones
5x9x9x3 = 1,215 paramètres

4x(32x32)=4,096 neurones
4x11x11x5 = 2,420 paramètres

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**

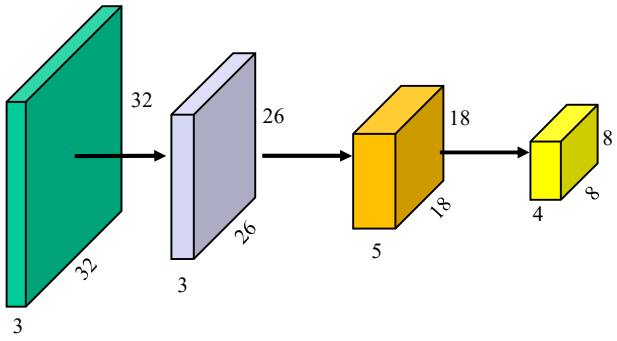


12,288 neurones au total
4,076 paramètres au total

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* »

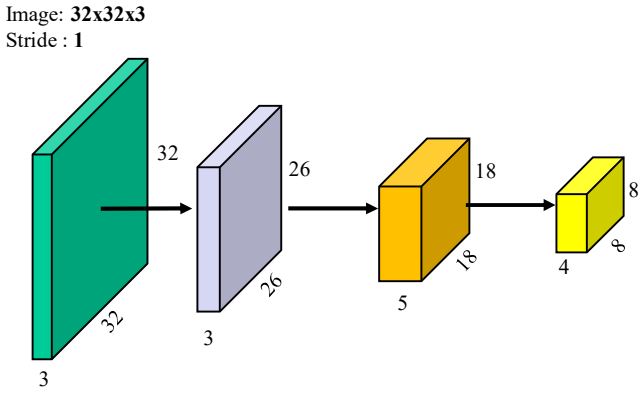
Image: **32x32x3**
Stride : **1**



3x(26x26)=2,028 neurones 5x(18x18)=1,620 neurones 4x(8x8)=256 neurones
3x7x7x3 = 441 paramètres 5x9x9x3 = 1,215 paramètres 4x11x11x5 = 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* »



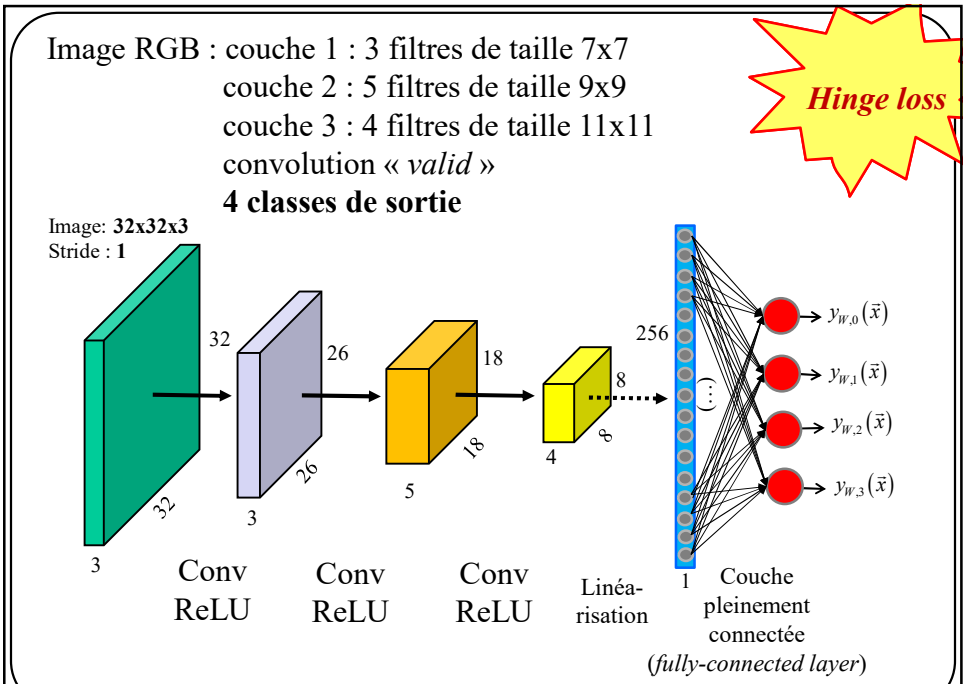
3,904 neurones au total
4,076 paramètres au total

115

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

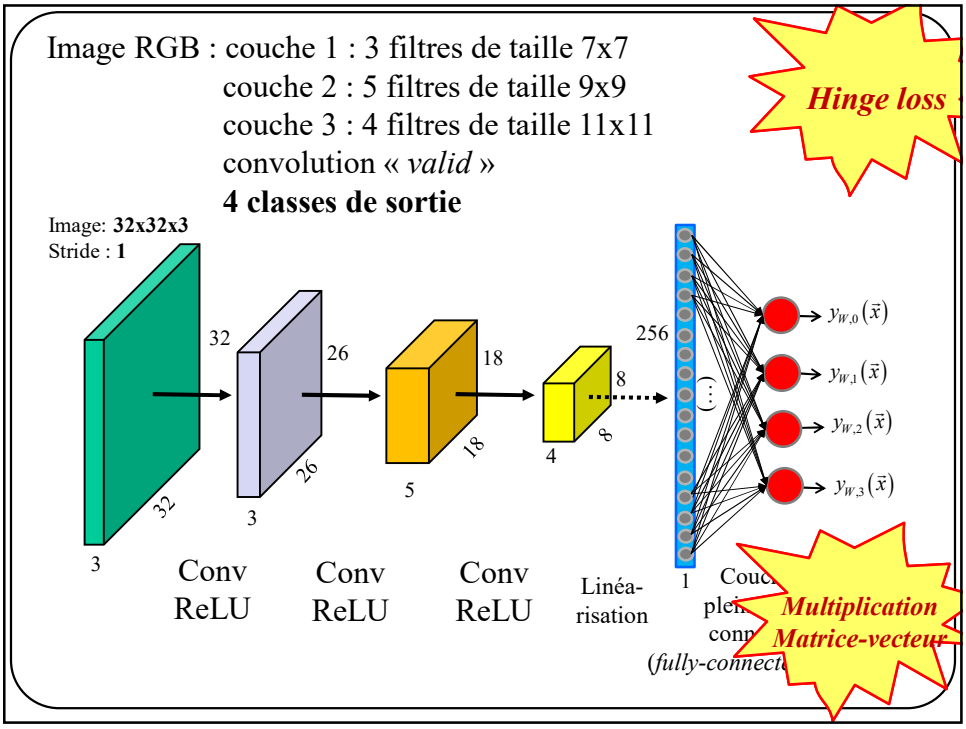
116

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



117

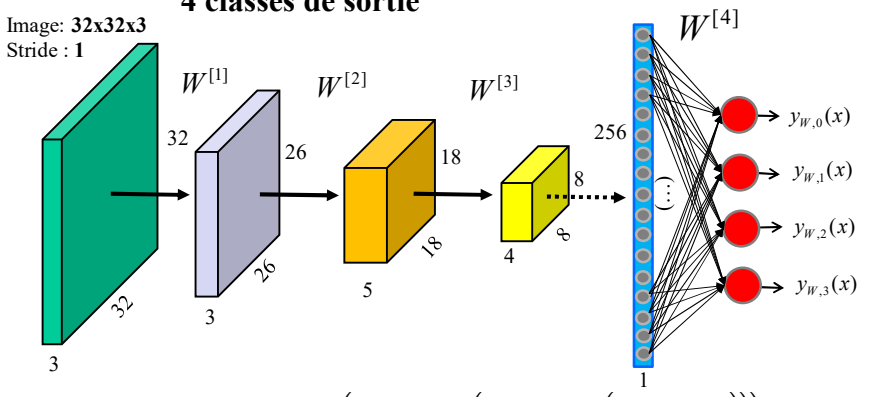
Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « *valid* »
4 classes de sortie



118

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

Hinge loss



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

119

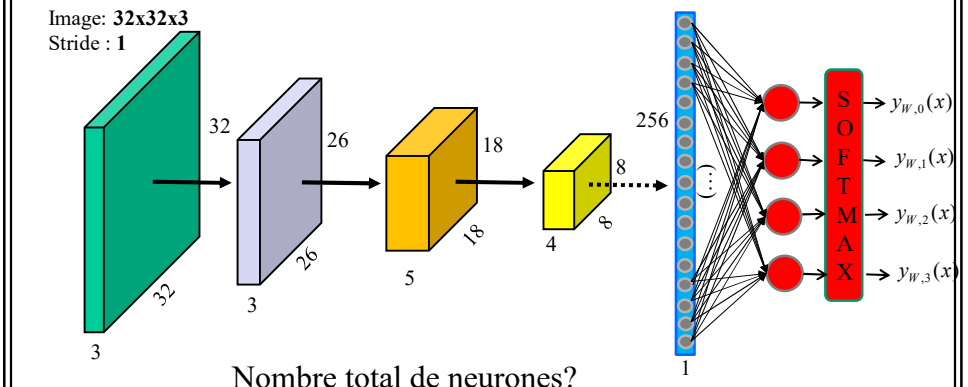
$$y_W(x) = W^{[4]} h\left(W^{[3]} * h\left(W^{[2]} * h\left(W^{[1]} * x\right)\right)\right)$$

Filtres convolutifs

Matrice 4x256

120

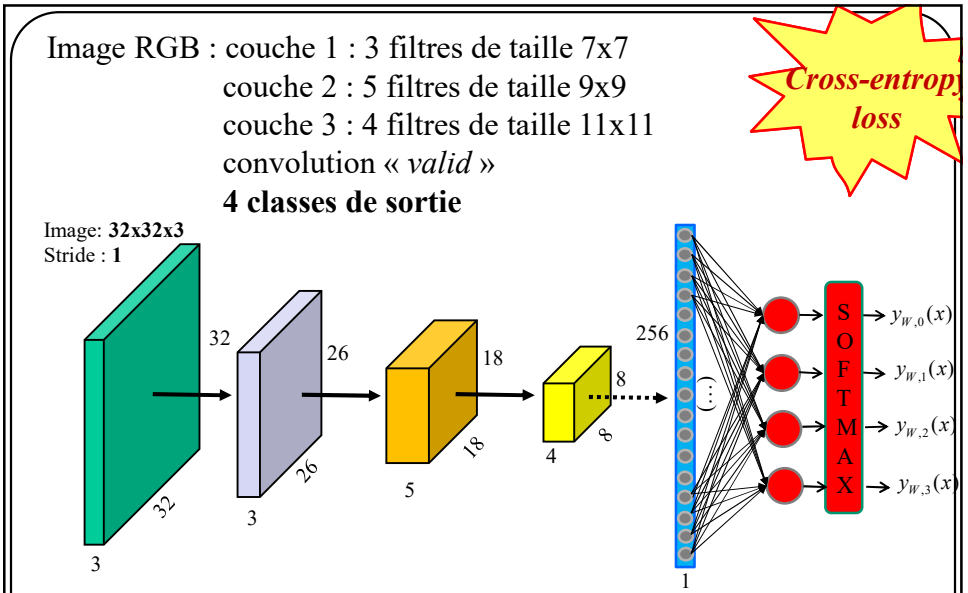
Image RGB : couche 1 : 3 filtres de taille 7x7
couche 2 : 5 filtres de taille 9x9
couche 3 : 4 filtres de taille 11x11
convolution « *valid* »
4 classes de sortie



Nombre total de neurones?

121

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

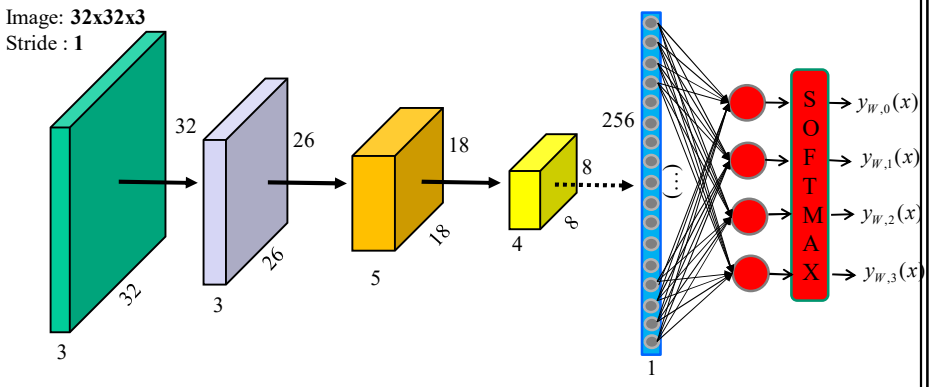


26x26x3 + 18x18x5 + 8x8x4 + 4

122

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

Cross-entropy loss

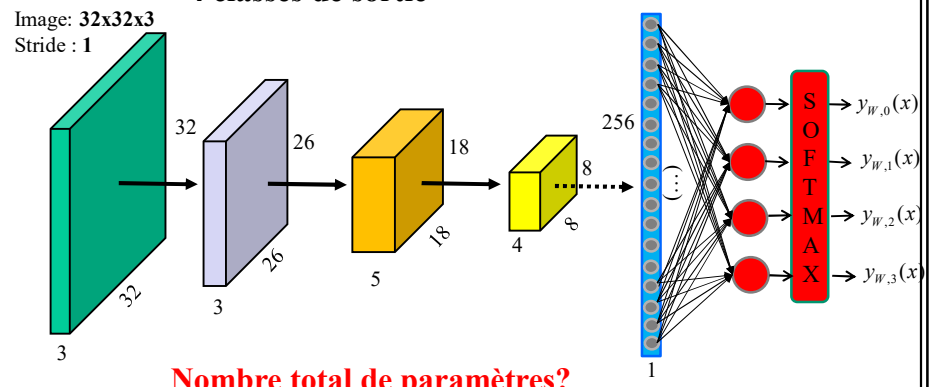


3,908 neurones

123

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

Cross-entropy loss

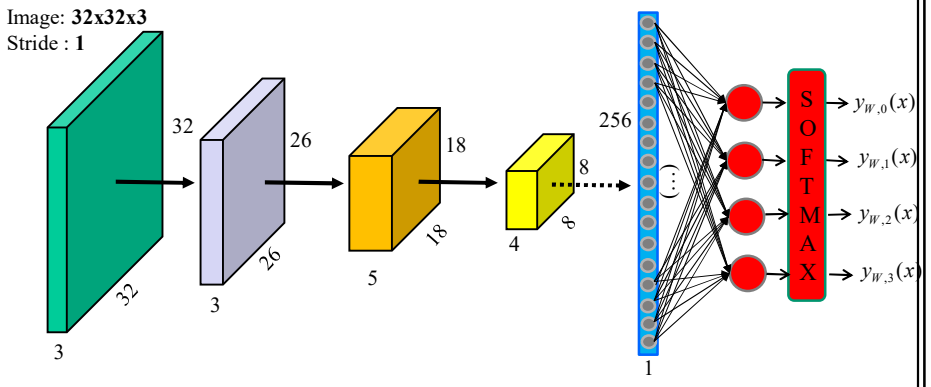


Nombre total de paramètres?

124

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

Cross-entropy loss

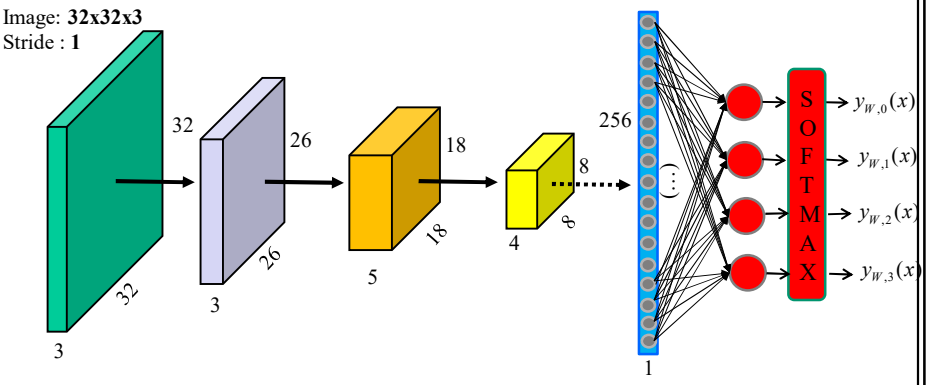


3x7x7x3 + 5x9x9x3 + 4x11x11x5 + 256x4

125

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

Cross-entropy loss



5,100 paramètres

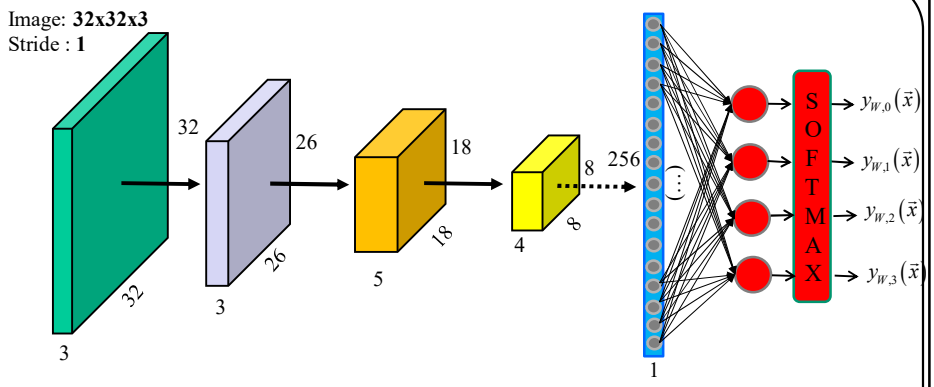
126

Réseaux à convolution

vs

Réseaux **pleinement** convolutifs

127

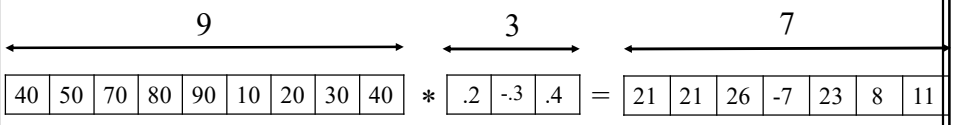


$$y_W(x) = W^{[4]}h\left(W^{[3]} * h\left(W^{[2]} * h\left(W^{[1]} * x\right)\right)\right)$$

Il est possible de **remplacer la multiplication matrice-vecteur** de la fin par une **convolution**

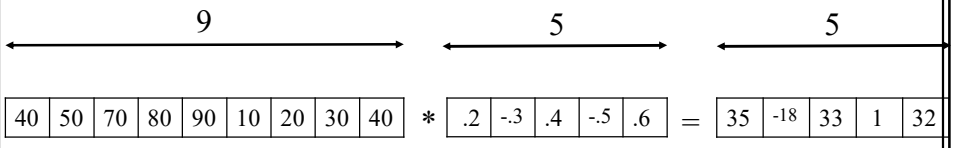
128

Exemple 1d
(convolution « valid »)



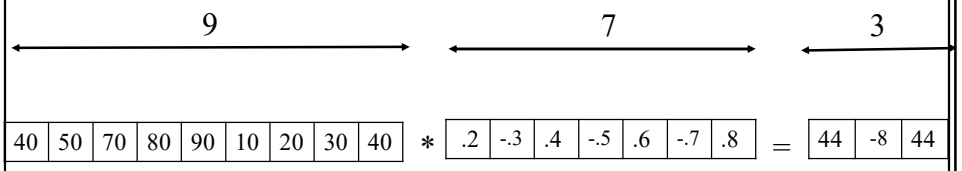
129

Exemple 1d
(convolution « valid »)



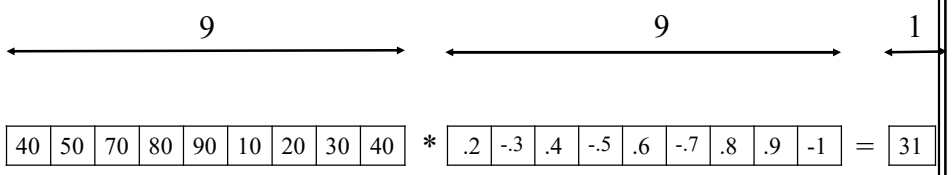
130

Exemple 1d
(convolution « valid »)



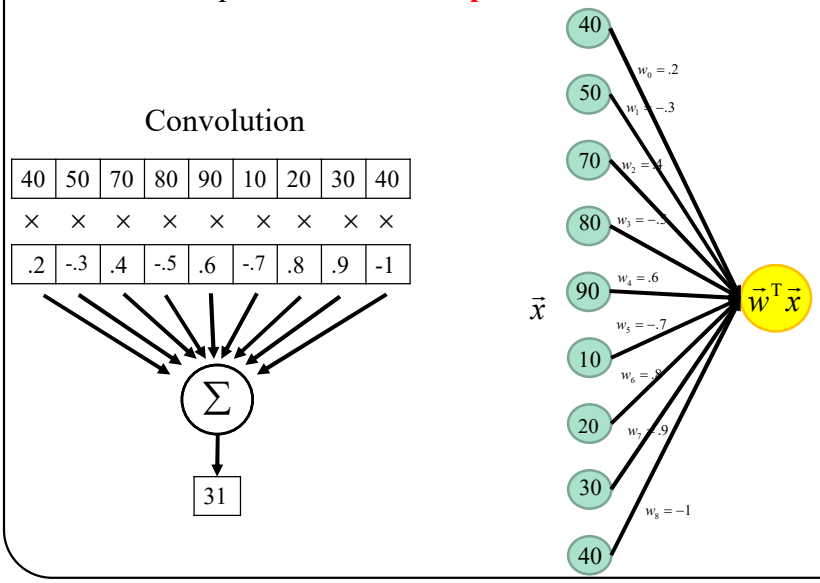
131

Taille filtre = nb de neurones couche précédente



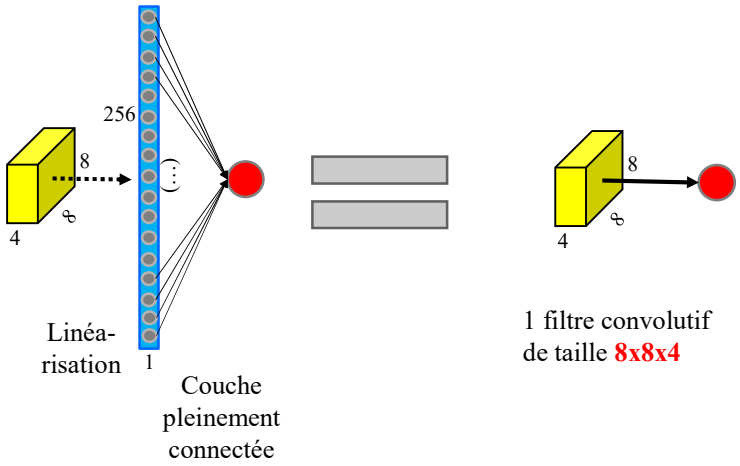
132

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

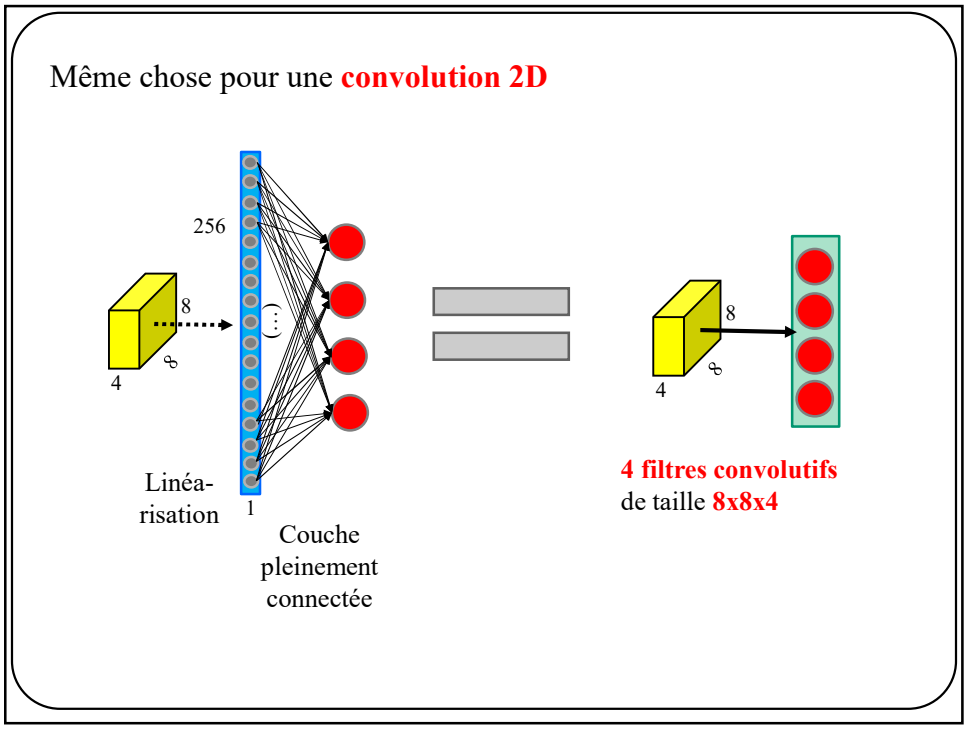


133

Même chose pour une **convolution 2D**



134


Même chose pour une convolution 2D
256
8
8
4
Linéa-
risation
1
Couche
pleinement
connectée
8
8
4
4 filtres convolutifs
de taille 8x8x4

135

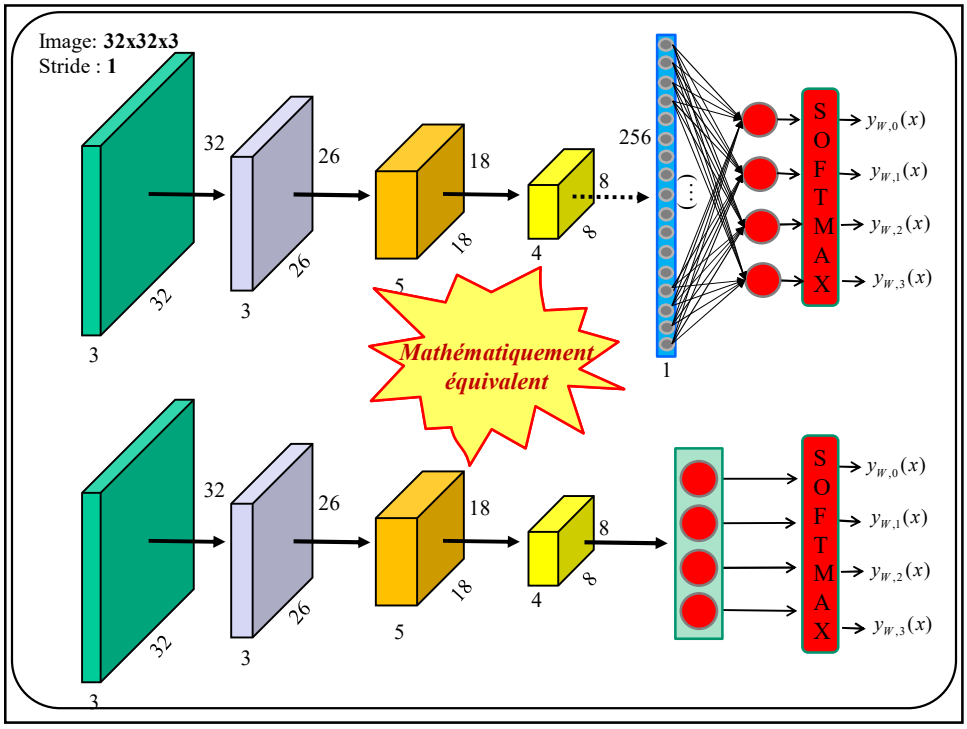
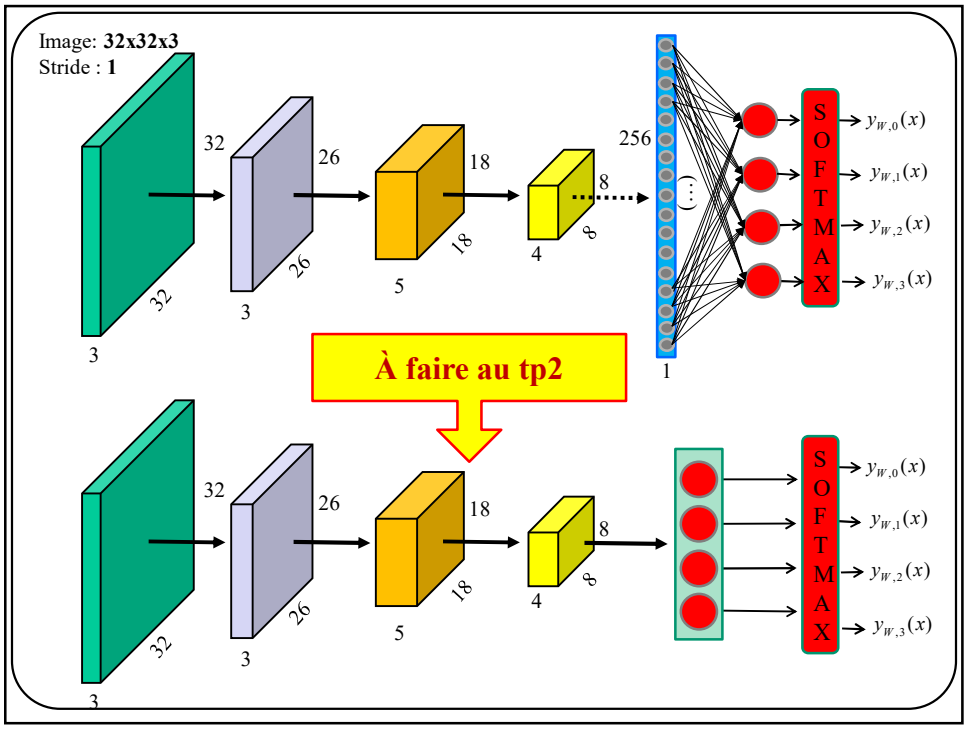


Image: 32x32x3
Stride : 1
32
32
3
26
26
3
18
18
5
8
8
4
256
1
Mathématiquement
équivalent
SOFTMAX
$y_{W,0}(x)$
$y_{W,1}(x)$
$y_{W,2}(x)$
$y_{W,3}(x)$

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 …

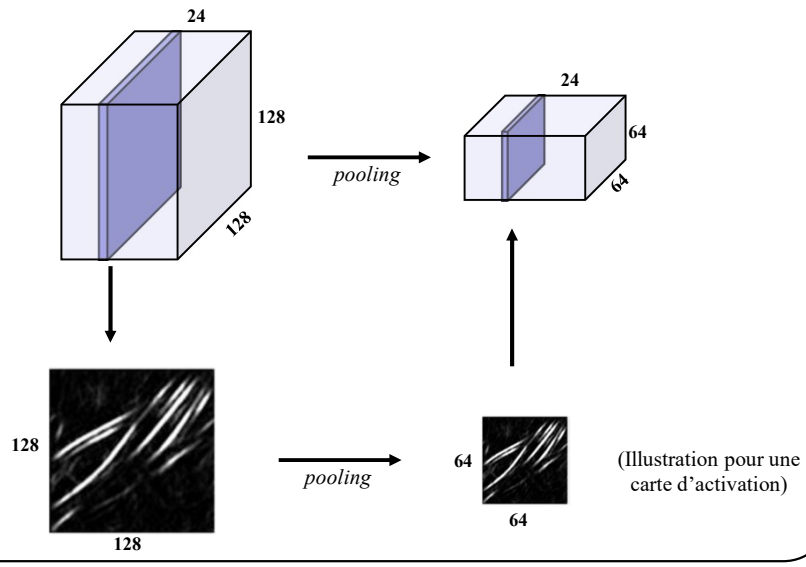
Question : qu'arrive-t-il si on remplace l'image 32x32x3 par une image 64x64x3?

138

Pooling

139

Réduction de la taille des cartes d'activation



140

Max pooling

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

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

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

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

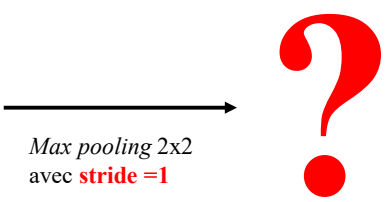
Moyenne par filtre « valid » 2x2 avec stride =2

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

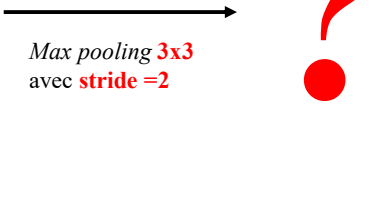


Max pooling 2x2
avec **stride =1**

143

Max pooling

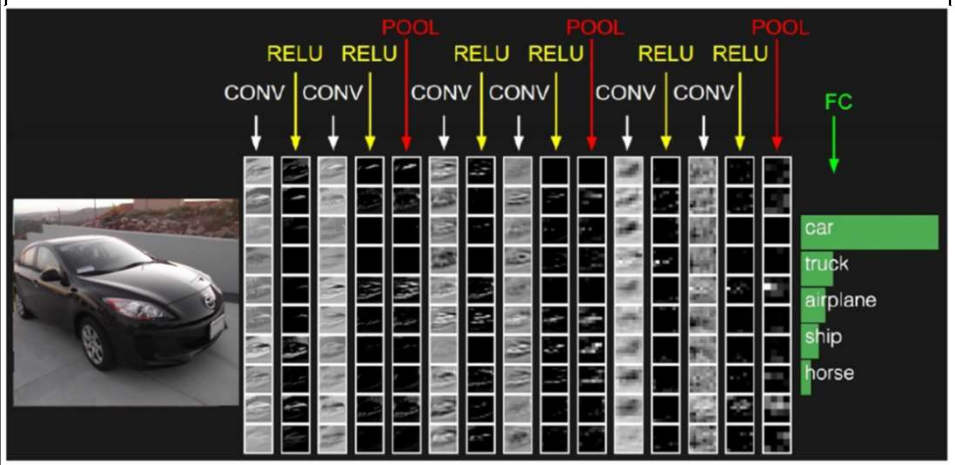
1	2	4	4	9	3	1	2
6	7	8	4	-3	-3	6	3
9	-9	8	-4	5	5	3	0
8	-8	9	-9	5	5	0	1
0	0	1	2	7	9	7	8
-1	-3	3	6	8	8	7	6
9	9	8	2	1	5	-1	-1
1	1	-2	8	3	7	4	-2



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

144

Illustration d'un CNN complet



Crédit : 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

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

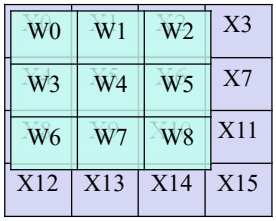
On peut **remplacer** une **convolution** par une **multiplication matrice-matrice** ou **matrice-vecteur**

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



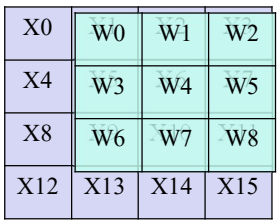
Y0	Y1
Y2	Y3

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



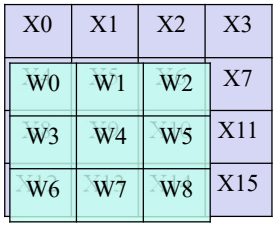
Y0	**Y1**
Y2	Y3

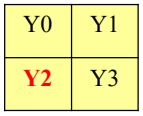
Y1=W0.X1+W1.X2+W2.X3+W3.X5+W4.X6+W5.X7+W6.X9+W7.X10+W8.X11

151

Rappel

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



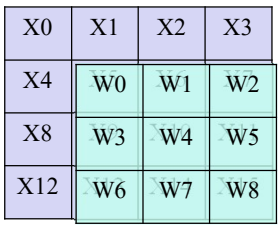


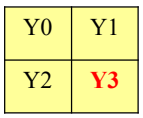
Y2=W0.X4+W1.X5+W2.X6+W3.X8+W4.X9+W5.X10+W6.X12+W7.X13+W8.X14

152

Rappel

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

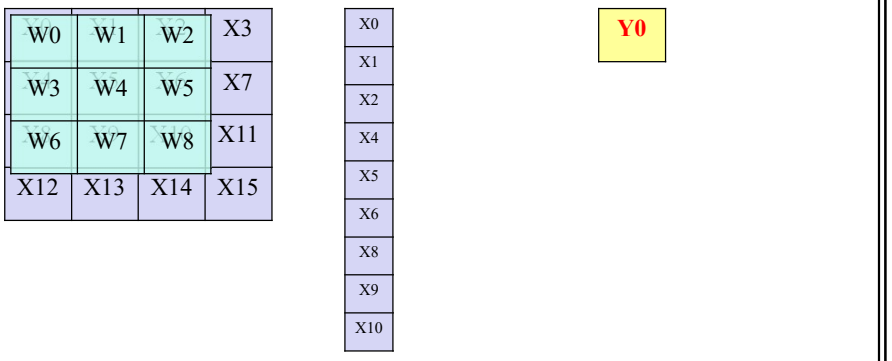




Y3=W0.X5+W1.X6+W2.X7+W3.X9+W4.X10+W5.X11+W6.X13+W7.X14+W8.X15

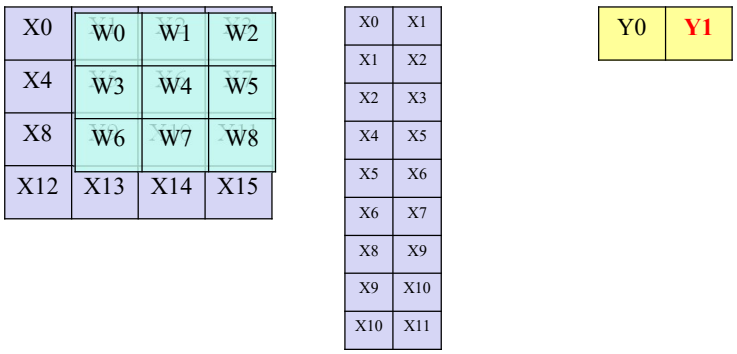
153

Autrement dit…



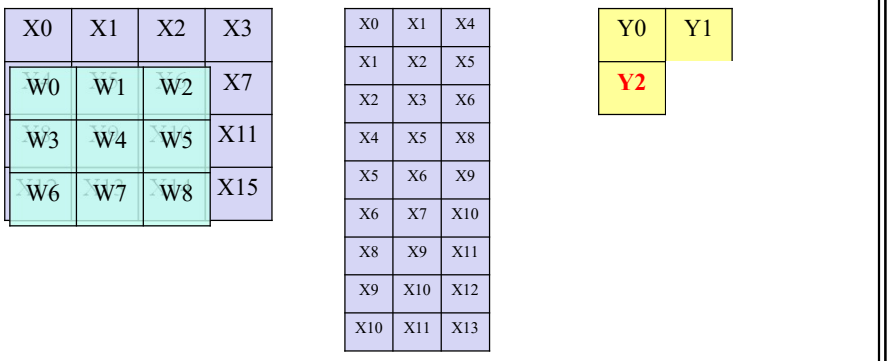
154

Autrement dit…



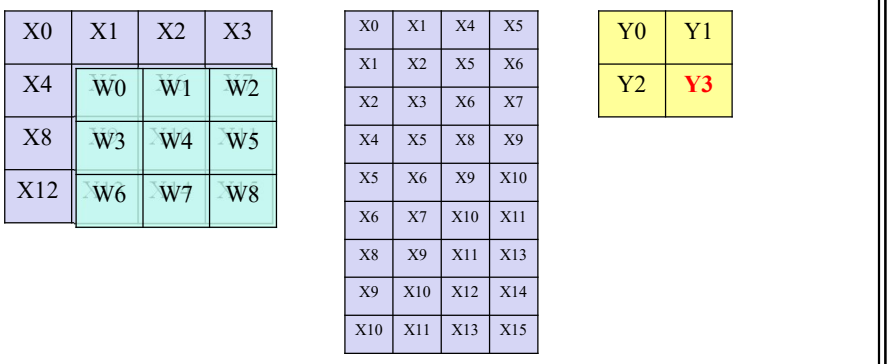
155

Autrement dit…



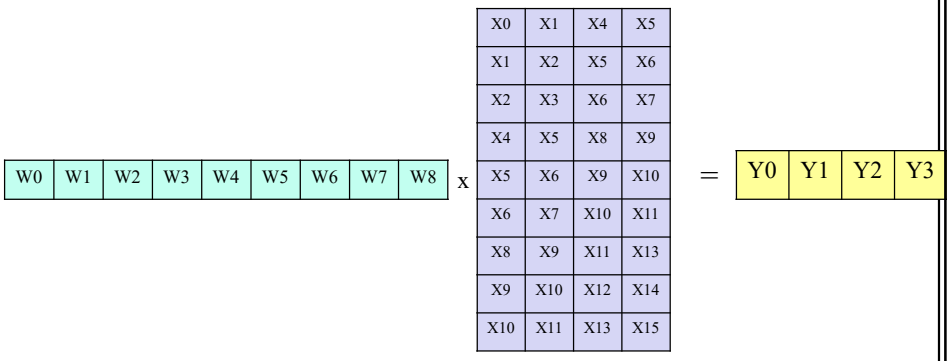
156

Autrement dit…



157

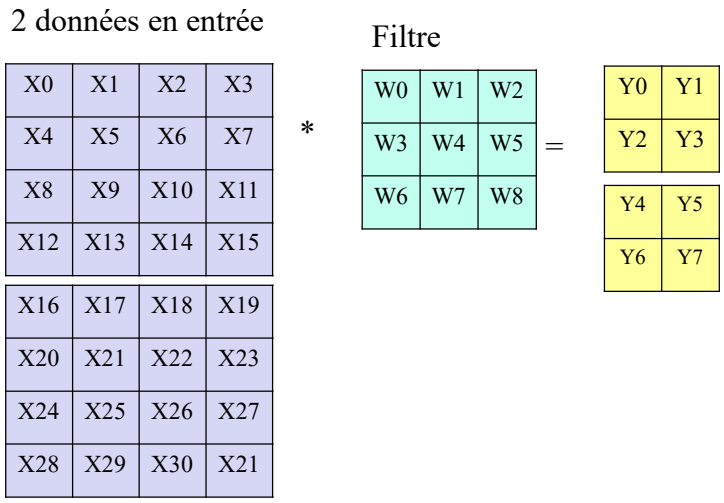
Convolution « valid » en **linéarisant le filtre** et en **« matriçant » l'entrée**



158

Autre exemple

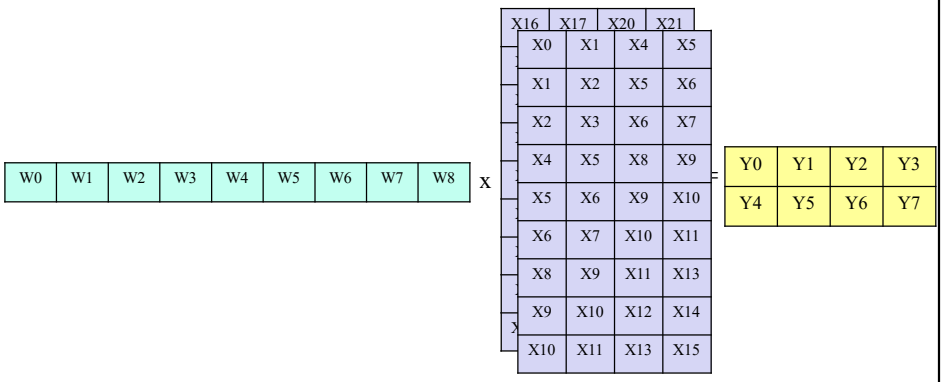
conv « valid », mini-batch de 2 entrées



159

Autre exemple

conv « valid », mini-batch de 2 entrées



160

Autre exemple

conv « valid », une entrée, deux filtres

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

W9	W10	W11
W12	W13	W14
W15	W16	W17

=

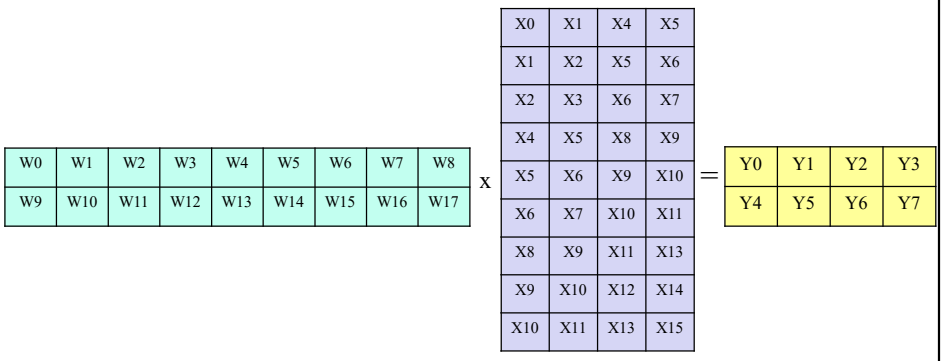
Y0	Y1
Y2	Y3

Y4	Y5
Y6	Y7

161

Autre exemple

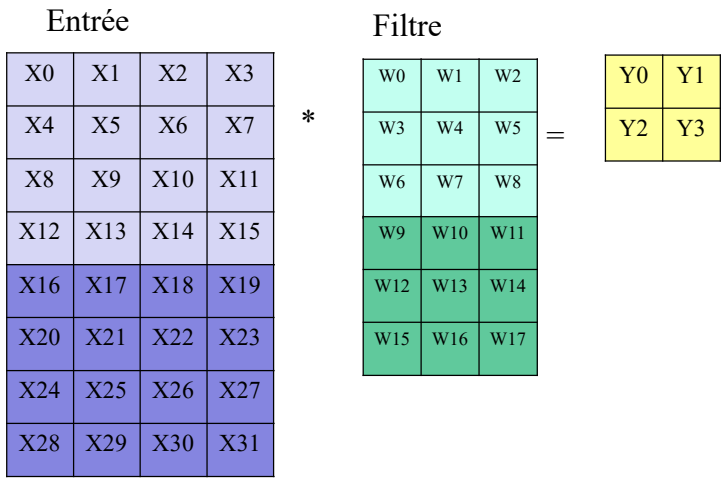
conv « valid », une entrée, deux filtres



162

Autre exemple

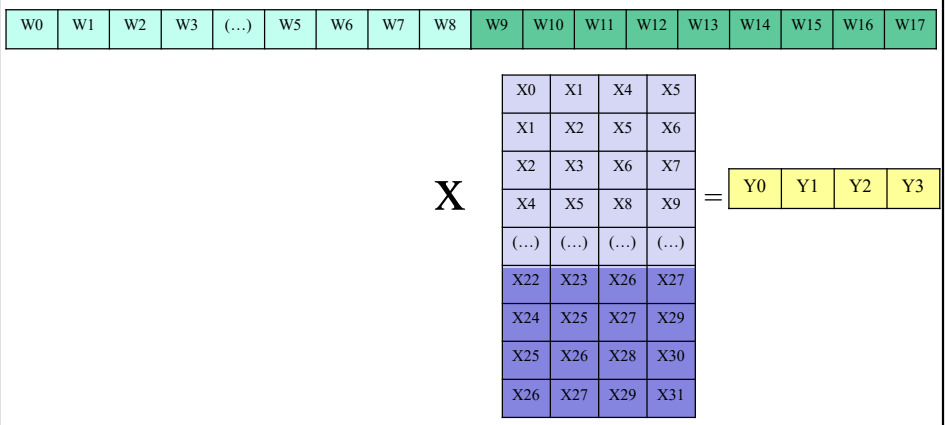
conv « valid », une entrée avec deux canaux, un filtre



163

Autre exemple

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

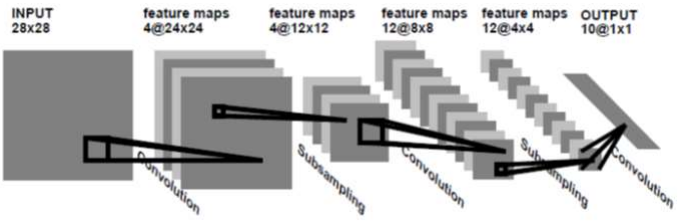


164

Exemples d'architectures connues

165

LeNet-1



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

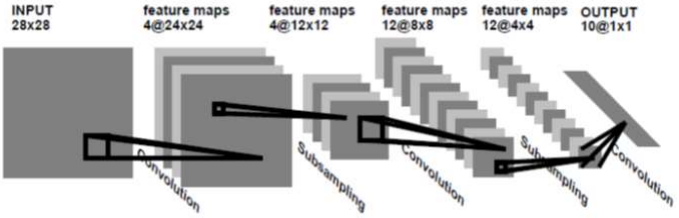
- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 4 et 12 filtres)
- 2 *average pooling*
- 1 couche pleinement connectée
- 10 classes

Input
CONV
POOL
CONV
POOL
FC
SOFTMAX

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

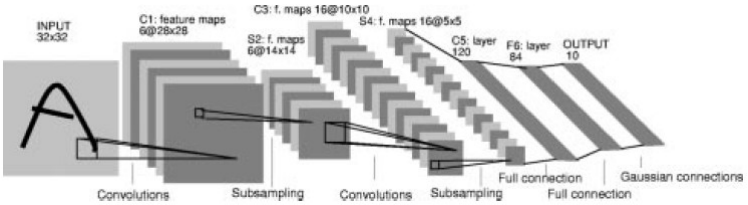
Input
CONV
POOL
CONV
POOL
FC
SOFTMAX

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:

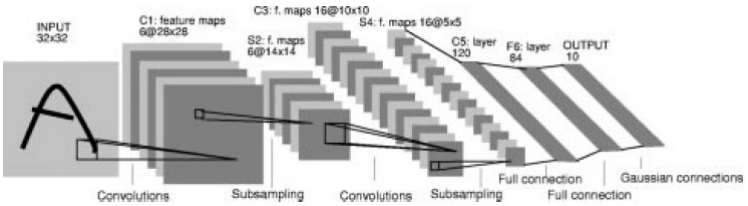
- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 6 et 16 filtres)
- 2 *average pooling*
- 3 couches pleinement connectées (120, 84 et 10 neurones)
- 10 classes

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

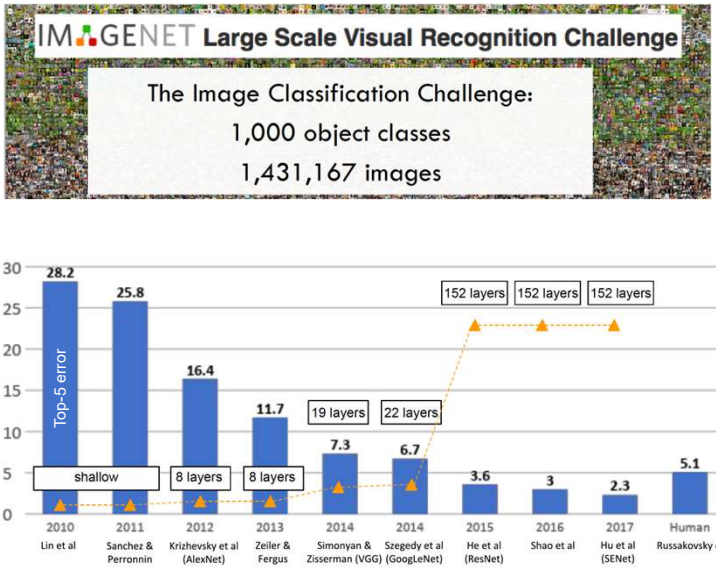


Image: http://cs231n.stanford.edu/slides/2018

170

170

Classification d'images

AlexNet [Krizhevsky et al, 2012]

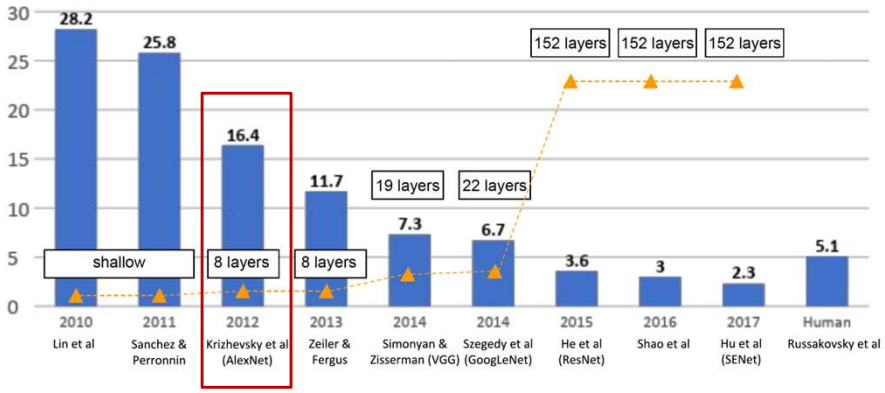
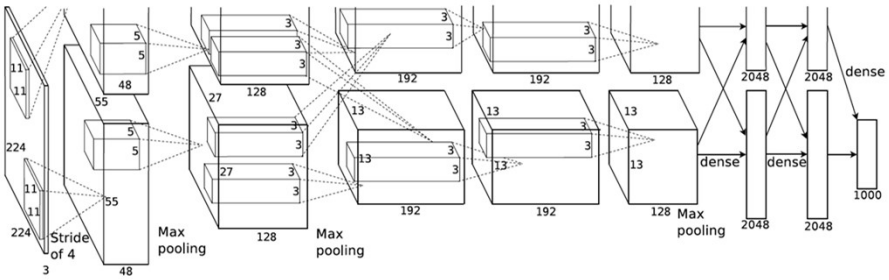


Image: http://cs231n.stanford.edu/slides/2018

171

171

AlexNet

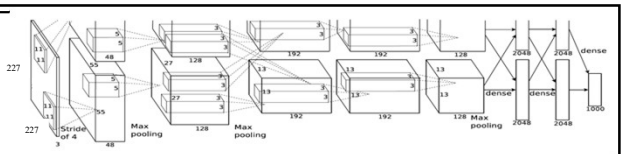


- Premier CNN à bien performer sur ImageNet (amélioration de 10% par rapport aux autres)
- Utilisation de techniques aujourd'hui fréquemment utilisées: **ReLU, data augmentation** and **dropout**
- Utilisation de **GPUs** (2 dans leur cas)
- Point de départ de la **révolution du "deep learning"** en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

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

172

AlexNet



Architecture:

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

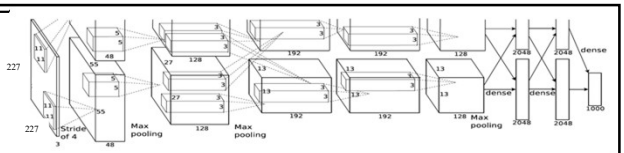
Entrée : image RGB: 227x227x3
Couche 1 (CONV1): 96 filtres de taille 11x11 avec **stride de 4** et conv **"valid"**

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

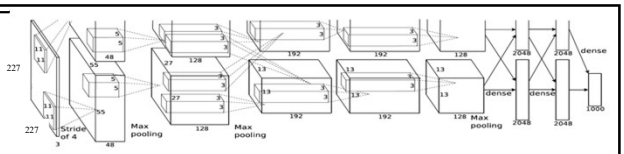
Entrée : image RGB: 227x227x3
Couche 1 (CONV1): 96 filtres de taille 11x11 avec **stride de 4** et conv **"valid"**

Cartes d'activation : 96 x 55 x 55

Q: Quel est le nombre de paramètres?

174

AlexNet



Architecture:

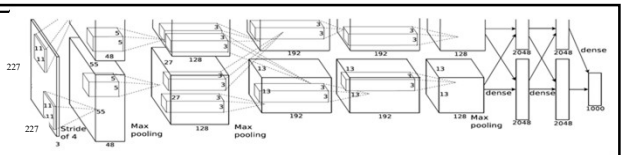
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

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 x 11 x 96 x 3 = 34,848

175

AlexNet



ENTRÉE : 227x227x3
CONV1: 96 x 55 x 55

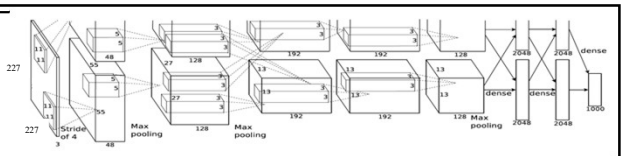
Couche 2 MaxPool : 3x3 stride stride 2

Quelle est la taille des cartes d'activation?

Réponse: (55-3)/2+1 = 27

176

AlexNet



ENTRÉE : 227x227x3
CONV1: 96 x 55 x 55

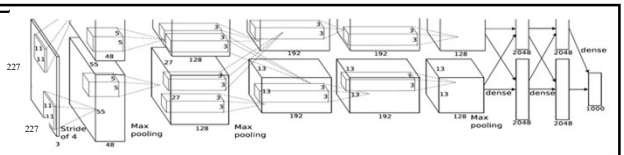
Couche 2 MaxPool : 3x3 stride stride 2
27 x 27 x 96

Combien y a-t-il de paramètres?

Réponse: 0!

177

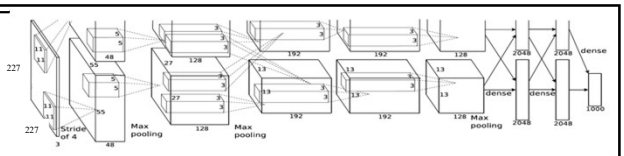
AlexNet



ENTRÉE : 227x227x3
CONV1: 55 x 55 x 96
MAX POOL1: 27 x 27 x 96
…

178

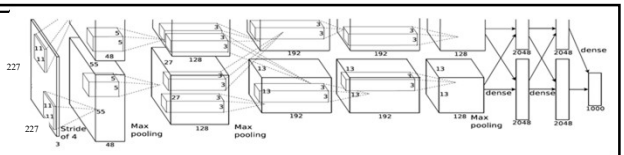
AlexNet



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



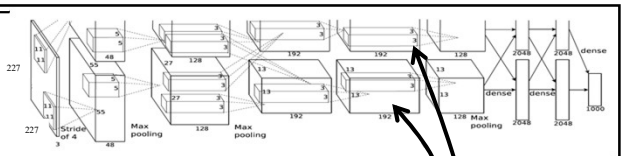
ENTRÉE	227 x 227 x 3
CONV1	55 x 55 x 96
MAX POOL1	27 x 27 x 96
NORM1	27 x 27 x 96
CONV2	27 x 27 x 256
MAX POOL2	13 x 13 x 256
NORM2	13 x 13 x 256
CONV3	13 x 13 x 384
CONV4	13 x 13 x 384
CONV5	13 x 13 x 256
MAX POOL3	6 x 6 x 256
FC6	4096
FC7	4096
FC8	1000

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

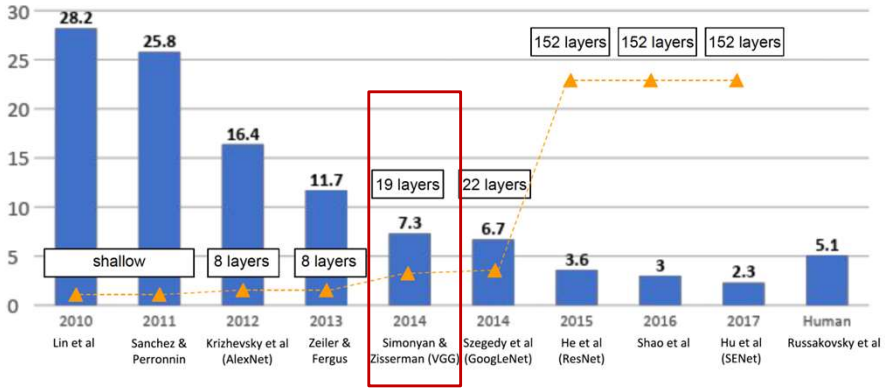
Notes additionnelles:

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

181

Classification d'images

VGGNet [Simonyan and Zisserman, 2014]



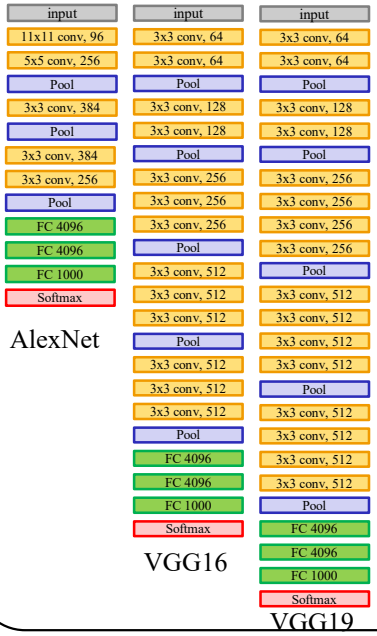
Karen Simonyan, Andrew Zisserman "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

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

182

182

VGGNet



Ce qui caractérise VGGNet par rapport à ses prédécesseurs:

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

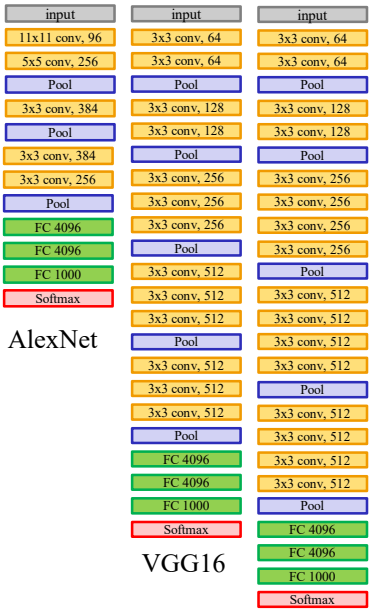
AlexNet : 8 couches
VGGNet : 16 ou 19 couches

7.3% d'erreur contre 11.7% pour ZFNet

183

183

VGGNet



Pourquoi utiliser des filtres 3x3?

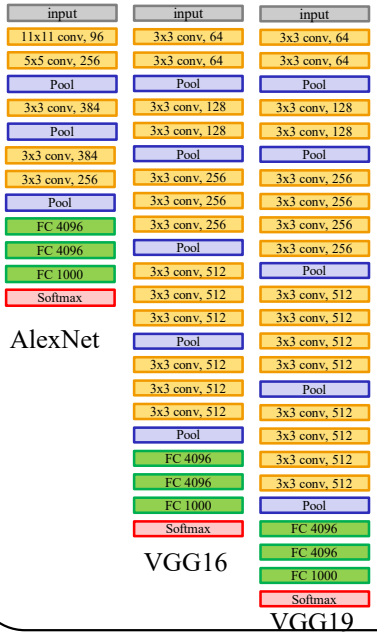
Indice1 : quel est le **champ récepteur (*receptive field*)** d'une série de 3 couches 3x3?

Indice2 : combien de paramètres pour ces 3 couches?

184

184

VGGNet



Pourquoi utiliser des filtres 3x3?

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

nb_params pour 3 filtres consécutifs **3x3x3xNb_Canaux**

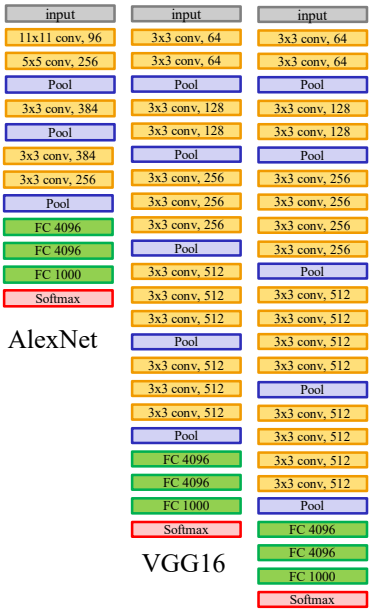
Vs

nb_params pour 1 couche**: 7x7xNb_Canaux**

185

185

VGGNet



Pourquoi utiliser des filtres 3x3?

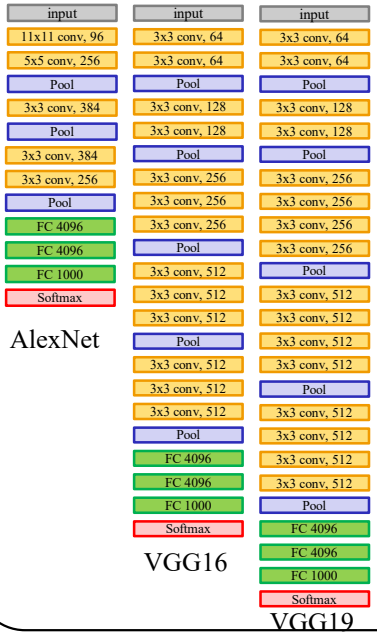
Le champ récepteur de 3 filtres 3x3 est **7x7**

nb_params pour 3 couches **27xNb_Canaux**
Vs
nb_params pour 1 couche**: 49xNb_Canaux**

186

186

VGGNet



Conclusion :

Plus de couches de plus petits filtres
=>
Moins de paramètres et **plus de non-linéarités**

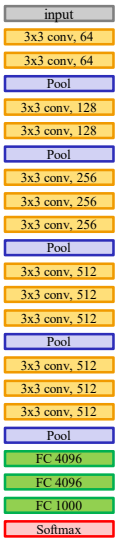
187

187

VGG16

ENTRÉE
CONV-3x3-64
CONV-3x3-64
POOL-2x2
CONV-3x3-128
CONV-3x3-128
POOL-2x2
CONV-3x3-256
CONV-3x3-256
CONV-3x3-256
POOL-2x2
CONV-3x3-512
CONV-3x3-512
CONV-3x3-512
POOL-2x2
CONV-3x3-512
CONV-3x3-512
CONV-3x3-512
POOL-2x2
FC-4096
FC-4096
FC-4096

Combien de paramètres?
Combien de neurones?



VGG16

188

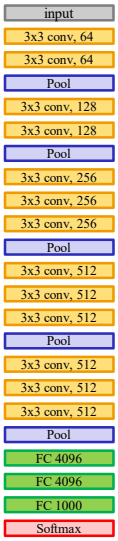
Image: http://cs231n.stanford.edu/slides/2

188

VGG16

Cartes d'activation

ENTRÉE	[224x224x3]
CONV-3x3-64	[224x224x64]
CONV-3x3-64	[224x224x64]
POOL-2x2	[112x112x64]
CONV-3x3-128	[112x112x128]
CONV-3x3-128	[112x112x128]
POOL-2x2	[56x56x128]
CONV-3x3-256	[56x56x256]
CONV-3x3-256	[56x56x256]
CONV-3x3-256	[56x56x256]
POOL-2x2	[28x28x256]
CONV-3x3-512	[28x28x512]
CONV-3x3-512	[28x28x512]
CONV-3x3-512	[28x28x512]
POOL-2x2	[14x14x512]
CONV-3x3-512	[14x14x512]
CONV-3x3-512	[14x14x512]
CONV-3x3-512	[14x14x512]
POOL-2x2	[7x7x512]
FC-4096	[1x1x4096]
FC-4096	[1x1x4096]
FC-4096	[1x1x1000]



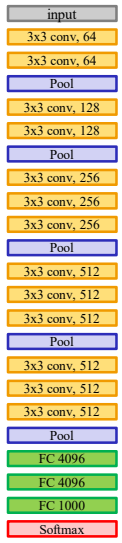
VGG16

189

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

189

VGG16	Cartes d'activation	Nb Neurones
ENTRÉE	[224x224x3]	150 K
CONV-3x3-64	[224x224x64]	3.2 M
CONV-3x3-64	[224x224x64]	3.2 M
POOL-2x2	[112x112x64]	800 k
CONV-3x3-128	[112x112x128]	1.6 M
CONV-3x3-128	[112x112x128]	1.6 M
POOL-2x2	[56x56x128]	400 K
CONV-3x3-256	[56x56x256]	800 K
CONV-3x3-256	[56x56x256]	800 K
CONV-3x3-256	[56x56x256]	800 K
POOL-2x2	[28x28x256]	200 K
CONV-3x3-512	[28x28x512]	400 K
CONV-3x3-512	[28x28x512]	400 K
CONV-3x3-512	[28x28x512]	400 K
POOL-2x2	[14x14x512]	100 K
CONV-3x3-512	[14x14x512]	100 K
CONV-3x3-512	[14x14x512]	100 K
CONV-3x3-512	[14x14x512]	100 K
POOL-2x2	[7x7x512]	25 K
FC-4096	[1x1x4096]	4094
FC-4096	[1x1x4096]	4096
FC-4096	[1x1x1000]	1000



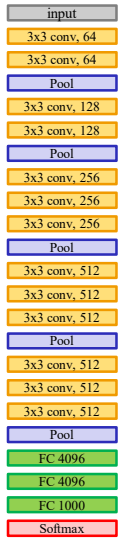
VGG16

190

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

190

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



VGG16

191

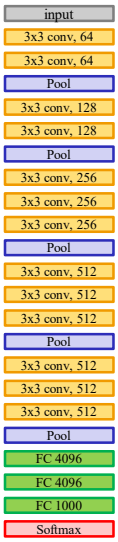
Image: http://cs231n.stanford.edu/slides/2

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

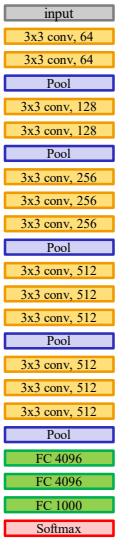
192

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



VGG16

193

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

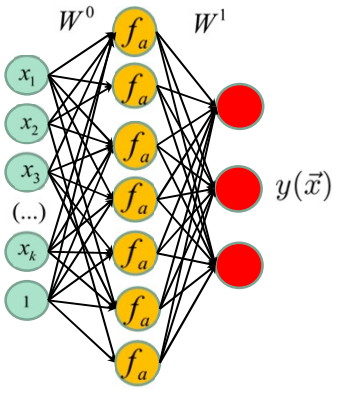
193

Réseaux récurrents

(si le temps le permet)

194

Réseau de neurones de base (régression)



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

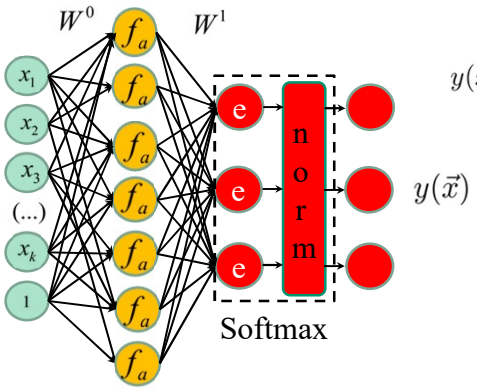
$$\vec{h} = f_a(W^0 \vec{x})$$

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

f_a : fonction d'activation

195

Réseau de neurones de base (classification)



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

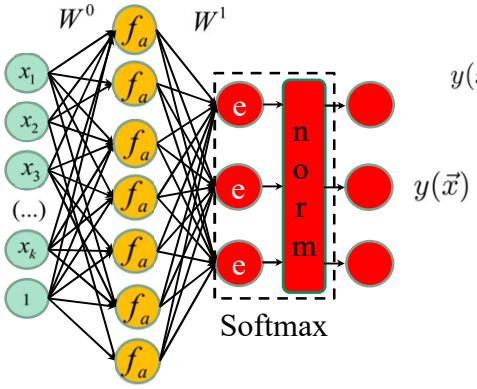
$$\vec{h} = f_a(W^0 \vec{x})$$

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

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

196

Réseau de neurones de base (classification)



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

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

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

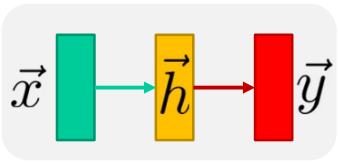
$$y(\vec{x}) = \text{softmax}(\hat{y})$$

Ne permet que des tâches “1 pour 1”

- Classification (1 image = 1 étiquette)
- Régression (1 donnée = 1 vecteur)
- Localisation (1 boîte = 1 classification + 1 régression)

197

Illustration simplifiée

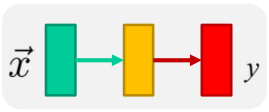


RN de base

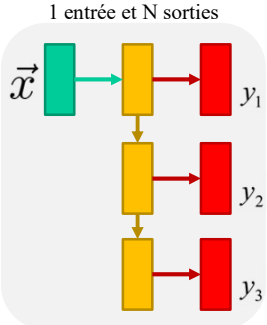
198

Différentes configurations pour différentes applications

1 entrée et 1 sortie



1 entrée et N sorties

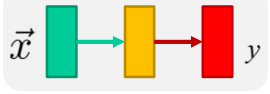


Ex.: description d'une image
1 image => N mots

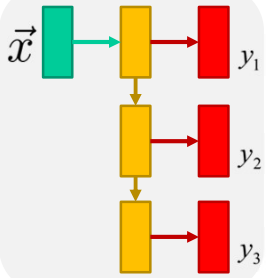
199

Différentes configurations pour différentes applications

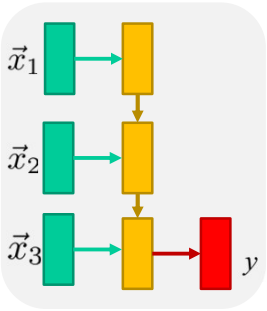
1 entrée et 1 sortie



1 entrée et N sorties



N entrées et 1 sortie

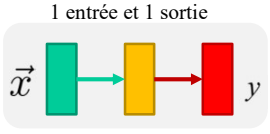


Ex.: Classification de texte
N mots => 1 classe

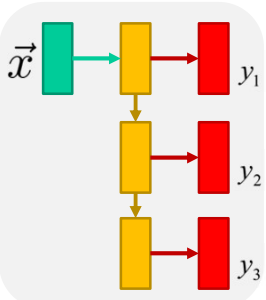
200

Différentes configurations pour différentes applications

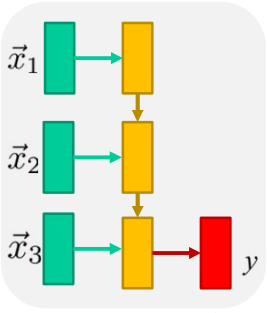
1 entrée et 1 sortie



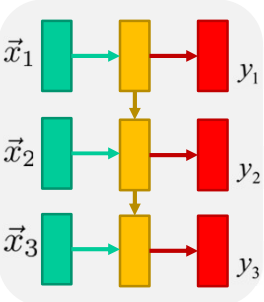
1 entrée et N sorties



N entrées et 1 sortie



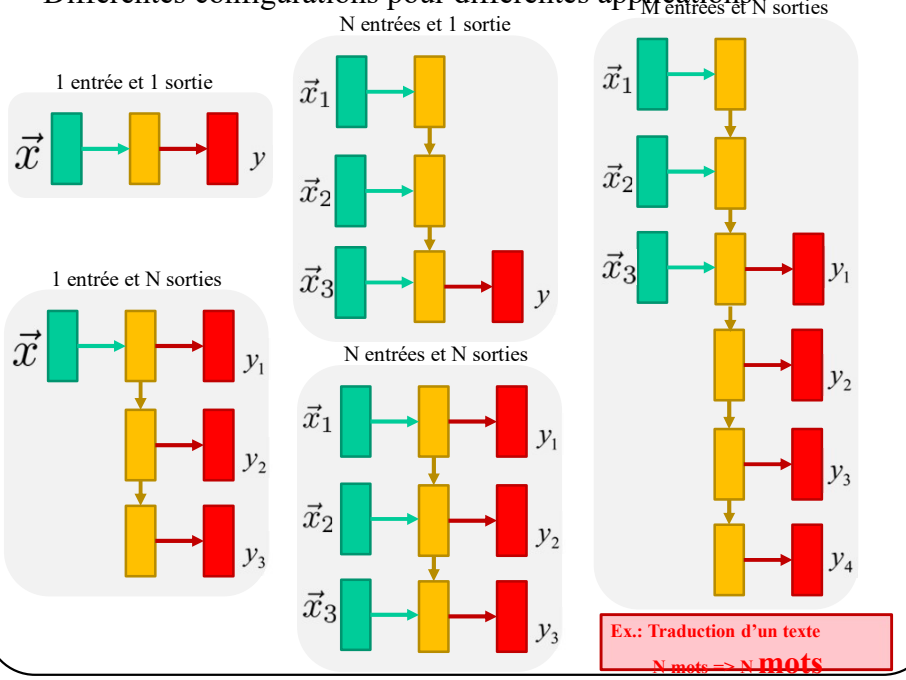
N entrées et N sorties



Ex.: Classification d'images vidéo
N images => N prédictions

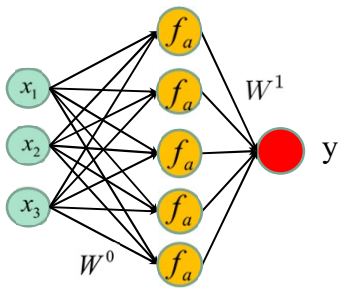
201

Différentes configurations pour différentes applications



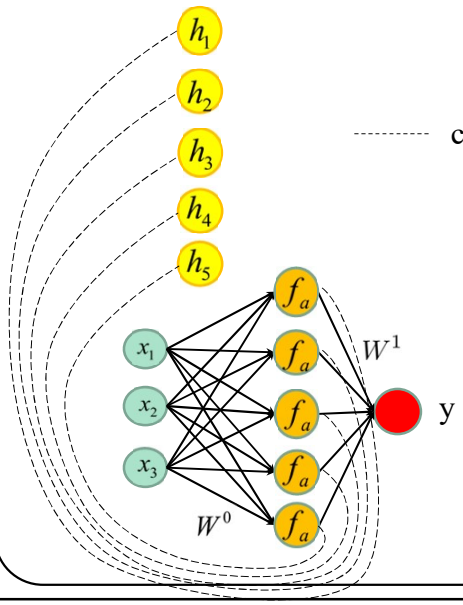
202

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



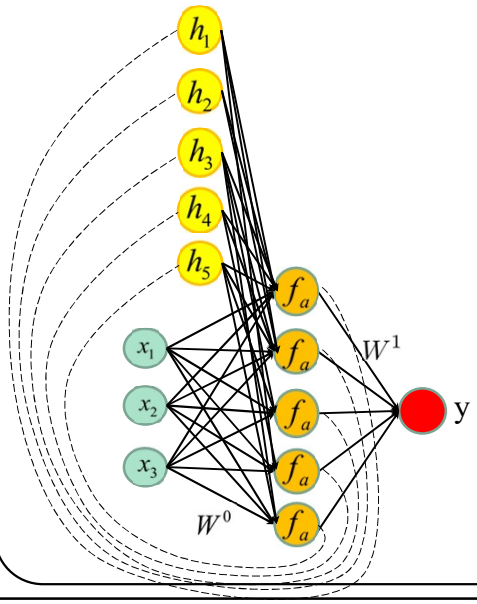
203

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



204

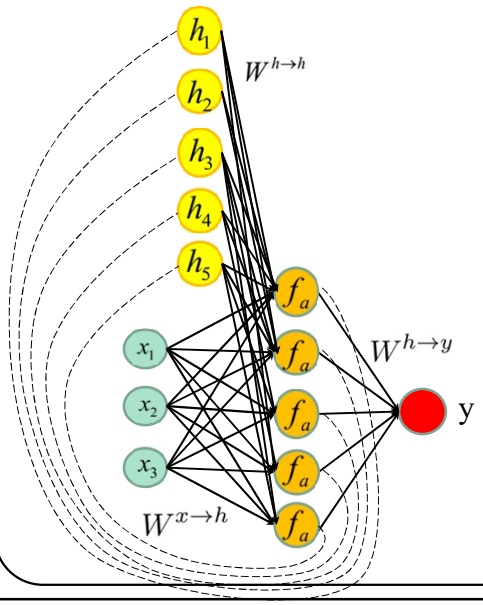
Réseau récurrent : la sortie des neurones est réinjectée dans leur entrée



Ici, au lieu d'avoir 3 entrées, chaque neurone caché a **3+5=8 entrées.**

205

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



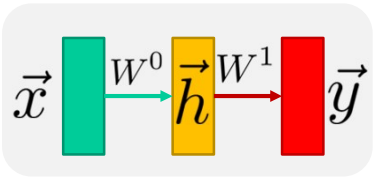
$$y(\vec{x}) = W^{h\to y} f_a(W^{x\to h}\vec{x} + W^{h\to h}\vec{h})$$

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

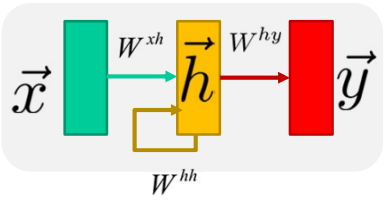
$$y(\vec{x}) = W^{h\to y}\vec{h}$$

206

Illustration simplifiée



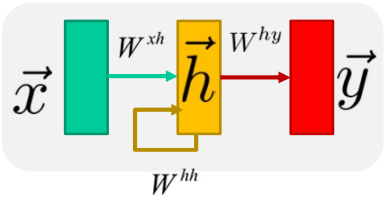
RN de base



RN récurrent

207

Dans le cas général avec K sorties (régression)



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

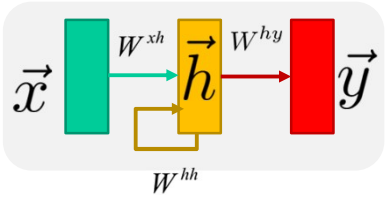
$$\downarrow$$

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

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

208

Dans le cas général avec K sorties (classification)



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

$$\downarrow$$

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

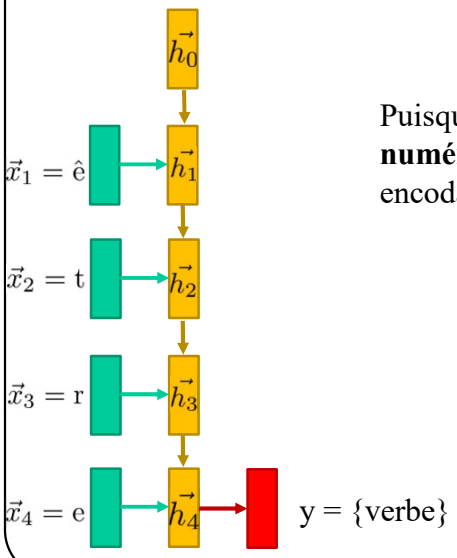
$$\hat{y} = W^{hy}\vec{h}$$

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

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* ».

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

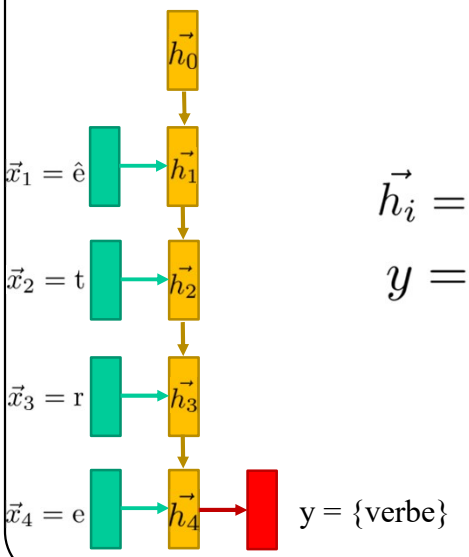
....

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

210

Exemple pour N entrées et 1 sortie:

Analyse grammaticale (classification) : (ê.t.r.e)=> {verbe}

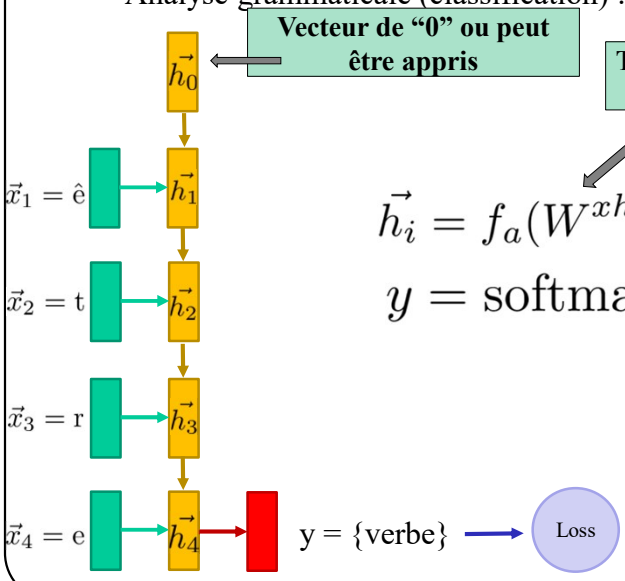


$$\vec{h_i} = f_a(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$y = \text{softmax}(W^{hy}\vec{h}_{i-1})$$

211

Exemple pour N entrées et 1 sortie:

Analyse grammaticale (classification) : (ê.t.r.e)=> {verbe}

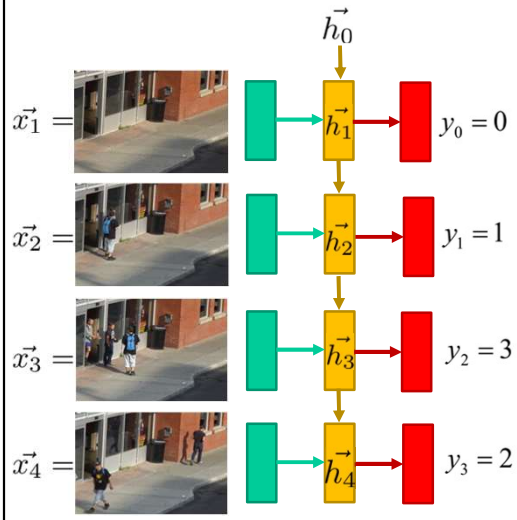


$$\vec{h_i} = f_a(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$y = \text{softmax}(W^{hy}\vec{h}_{i-1})$$

212

Même idée pour N entrées et N sorties:

ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons

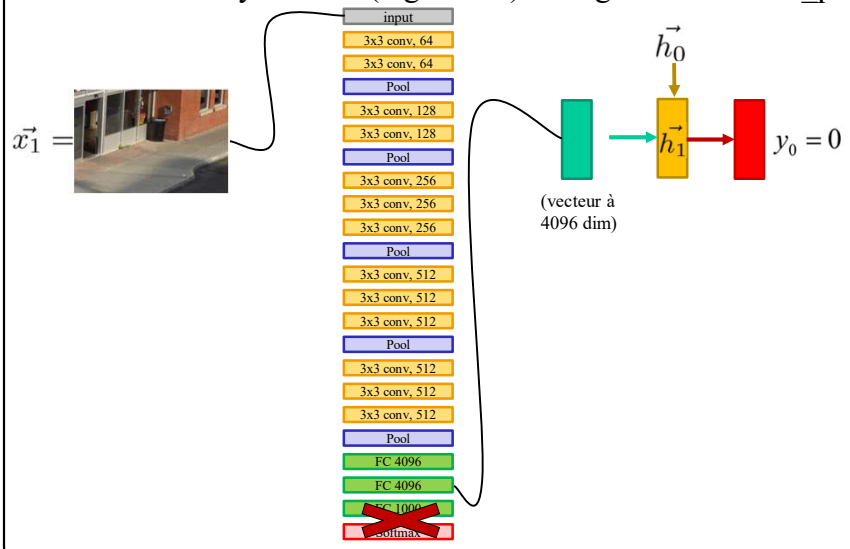


$$\vec{h_i} = f_a(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$y_i = W^{hy}\vec{h_i}$$

213

Même idée pour N entrées et N sorties:

ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons

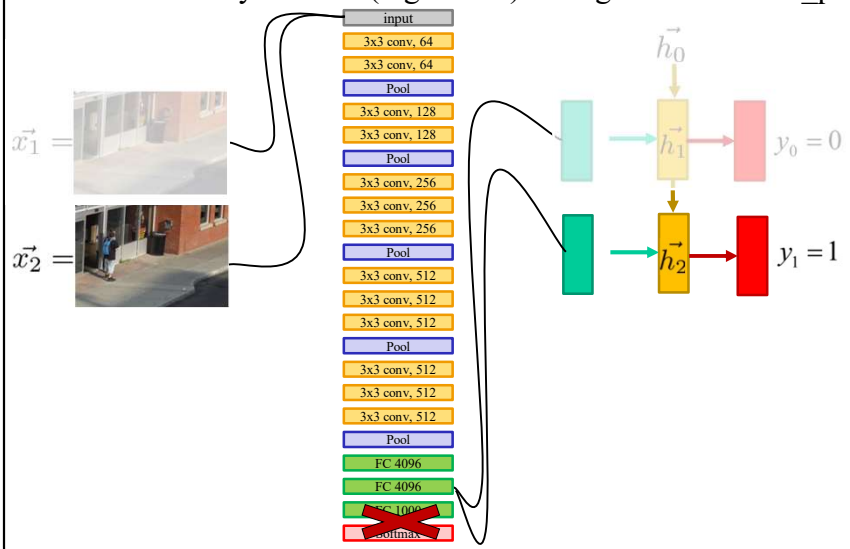


VGG16

214

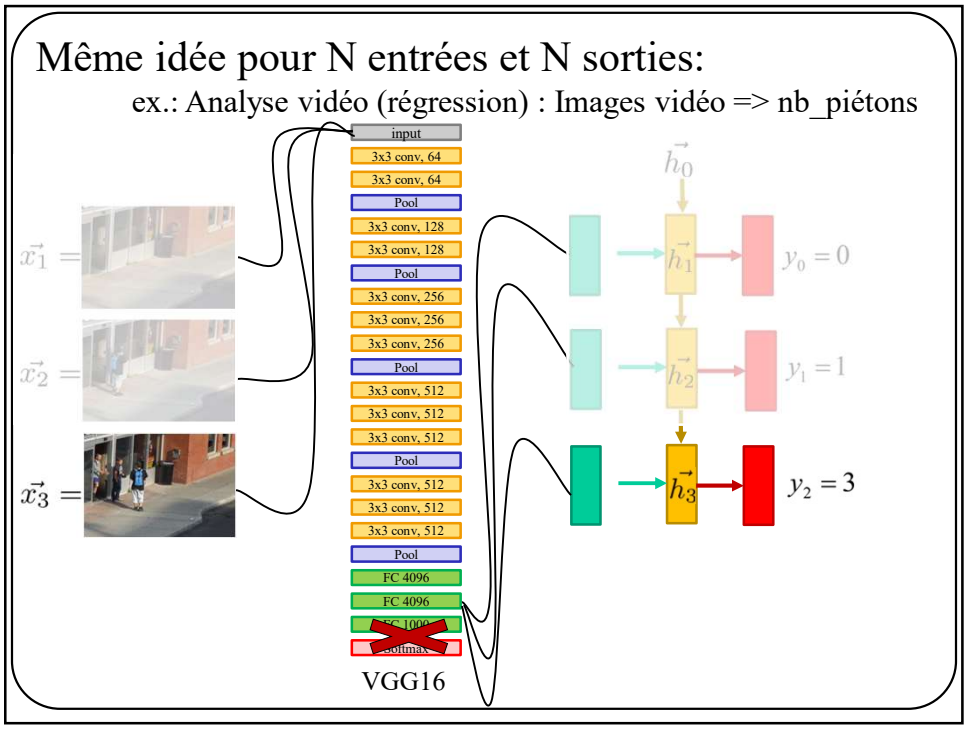
Même idée pour N entrées et N sorties:

ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons

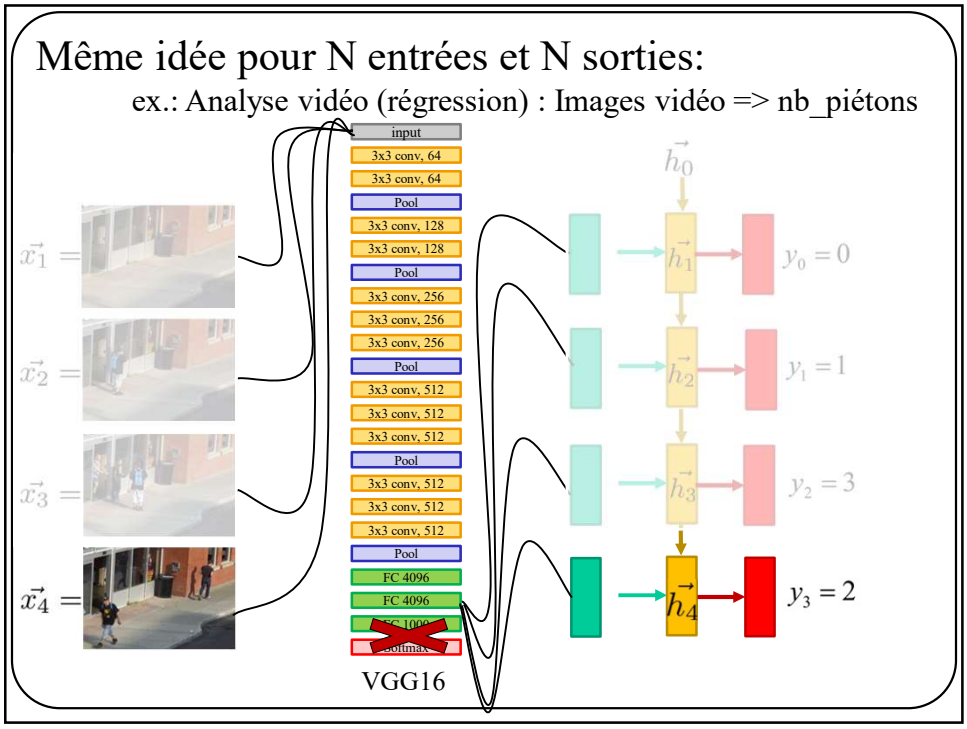


VGG16

215


Même idée pour N entrées et N sorties:
ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons
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
$\vec{x_1} =$
$\vec{x_2} =$
$\vec{x_3} =$
$\vec{h_0}$
$\vec{h_1}$
$\vec{h_2}$
$\vec{h_3}$
$y_0 = 0$
$y_1 = 1$
$y_2 = 3$
VGG16

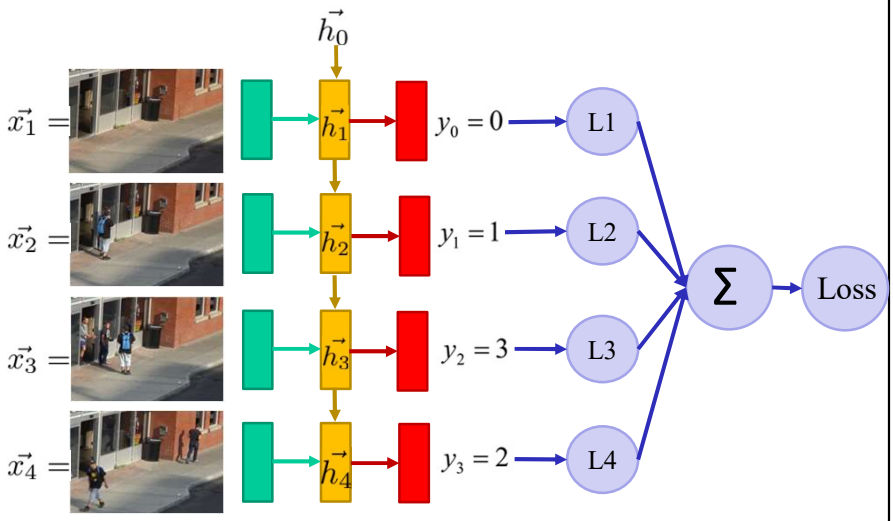
216


Même idée pour N entrées et N sorties:
ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons
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
$\vec{x_1} =$
$\vec{x_2} =$
$\vec{x_3} =$
$\vec{x_4} =$
$\vec{h_0}$
$\vec{h_1}$
$\vec{h_2}$
$\vec{h_3}$
$\vec{h_4}$
$y_0 = 0$
$y_1 = 1$
$y_2 = 3$
$y_3 = 2$
VGG16

217

Même idée pour N entrées et N sorties:

ex.: Analyse vidéo (régression) : Images vidéo => nb_piétons



218

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

Alphabet jouet :[**a,e,m,s**]

Représentation « one hot » jouet:

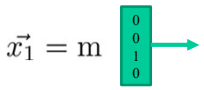
'a' = [1, 0, 0, 0]
'e' = [0, 1, 0, 0]
'm' = [0, 0, 1, 0]
's' = [0, 0, 0, 1]

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

219

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

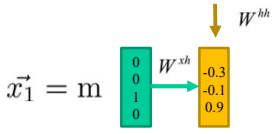
Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».



220

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

Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».

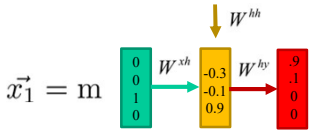


$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$

221

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

Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».

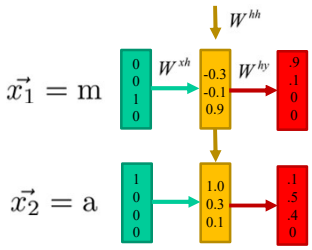


$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$\vec{y}(\vec{x_i}) = \mathrm{softmax}(W^{hy}\vec{h_i})$$

222

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

Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».

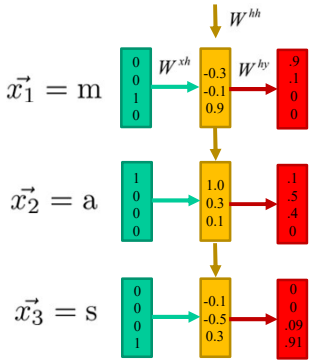


$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$\vec{y}(\vec{x_i}) = \mathrm{softmax}(W^{hy}\vec{h_i})$$

223

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

Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».

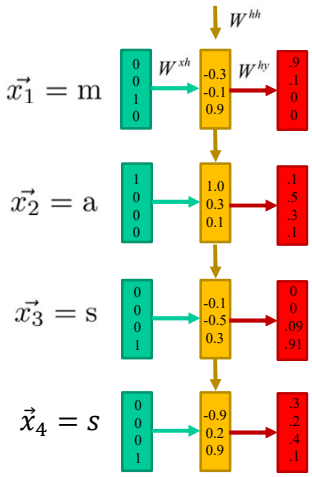


$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$\vec{y}(\vec{x_i}) = \text{softmax}(W^{hy}\vec{h_i})$$

224

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

Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».

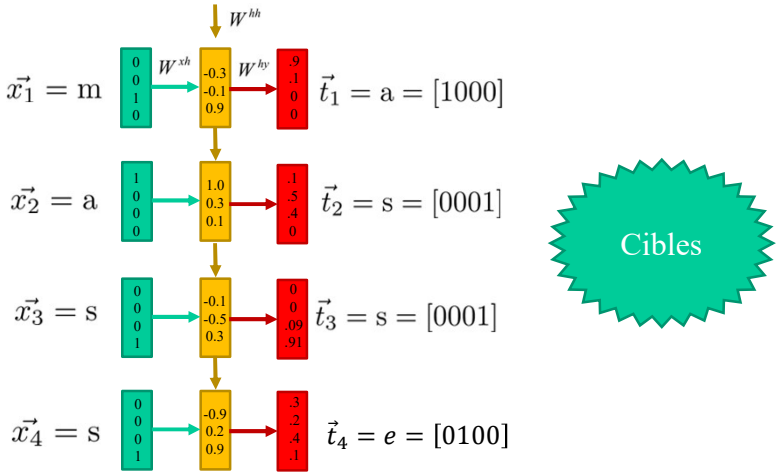


$$\vec{h_i} = \tanh(W^{xh}\vec{x_i} + W^{hh}\vec{h_{i-1}})$$
$$\vec{y}(\vec{x_i}) = \text{softmax}(W^{hy}\vec{h_i})$$

225

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

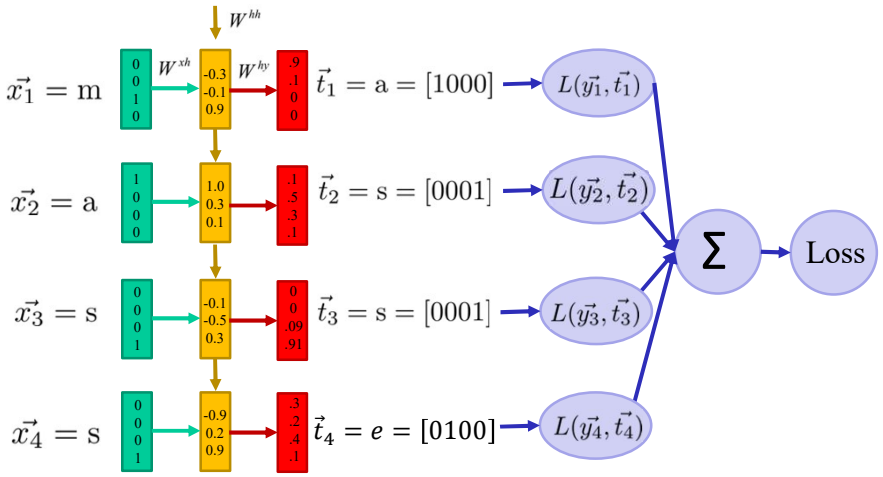
Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».



226

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

Alphabet :[**a,e,m,s**]
Entraîner un modèle à prédire les lettres du mot « **masse** ».

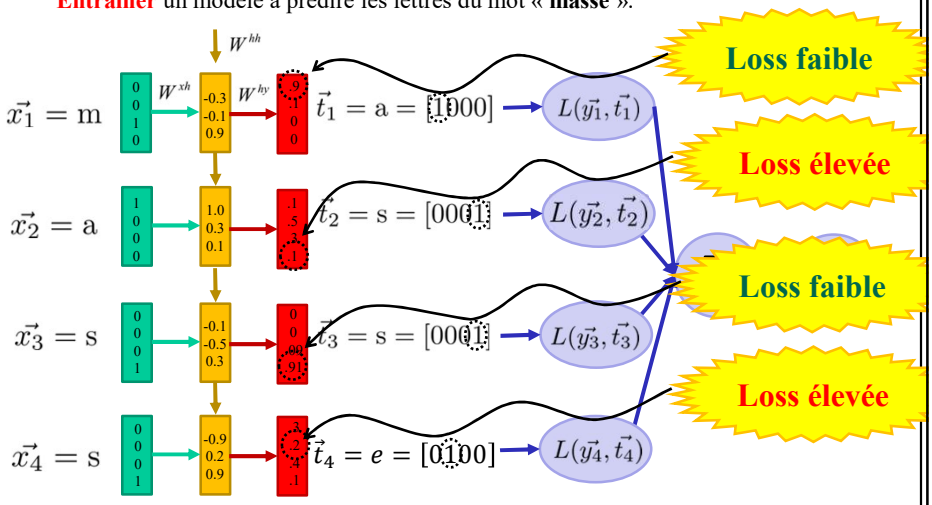


227

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

Alphabet :[**a,e,m,s**]

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

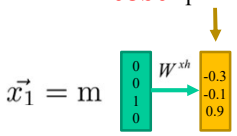


228

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

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres



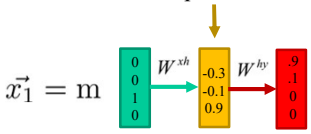
Étape 1 : Calcul de la couche cachée

229

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

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres



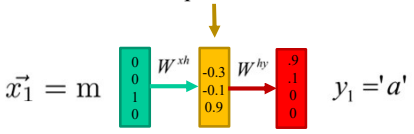
Étape 2 : Calcul de la sortie (softmax)

230

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

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres



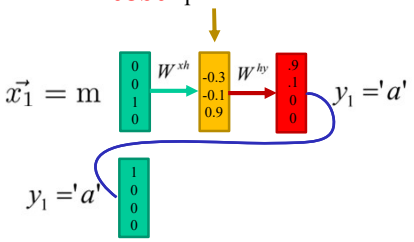
Étape 3 : Sélectionner le caractère le plus probable

231

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

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres



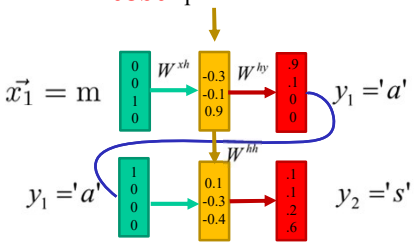
Étape 4 : Injecter le caractère prédit au début du réseau

232

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

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres



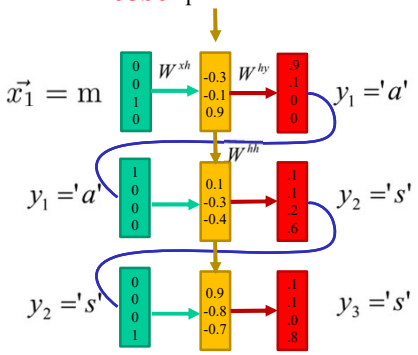
Et on recommence!

233

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

Alphabet :[**a,e,m,s**]

En test : prédire les lettres les unes après les autres

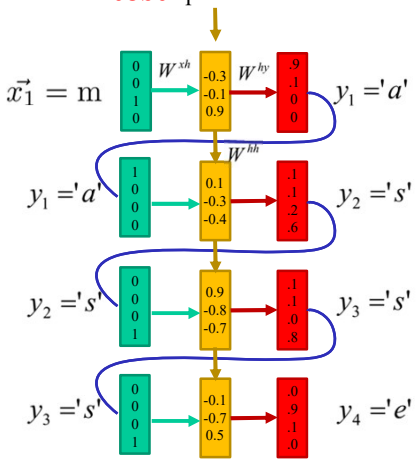


234

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/d4dee566867f8291f086
Un RNN en 112 lignes !

Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy

min-char-rnn.py

```
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
```

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

236

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

Code python: **"mini-char-RNN" de A. Karpathy**
https://gist.github.com/karpathy/d4dee566867f8291f086

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
  Pity the world, or else this glutton be,
  To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
  This were to be new made when thou art old,
  And see thy blood warm when thou feel'st it cold.

237

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

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened 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 srain 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 nues 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 aps, not a man and in fire,
To show the reining of the raven 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 fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, 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++

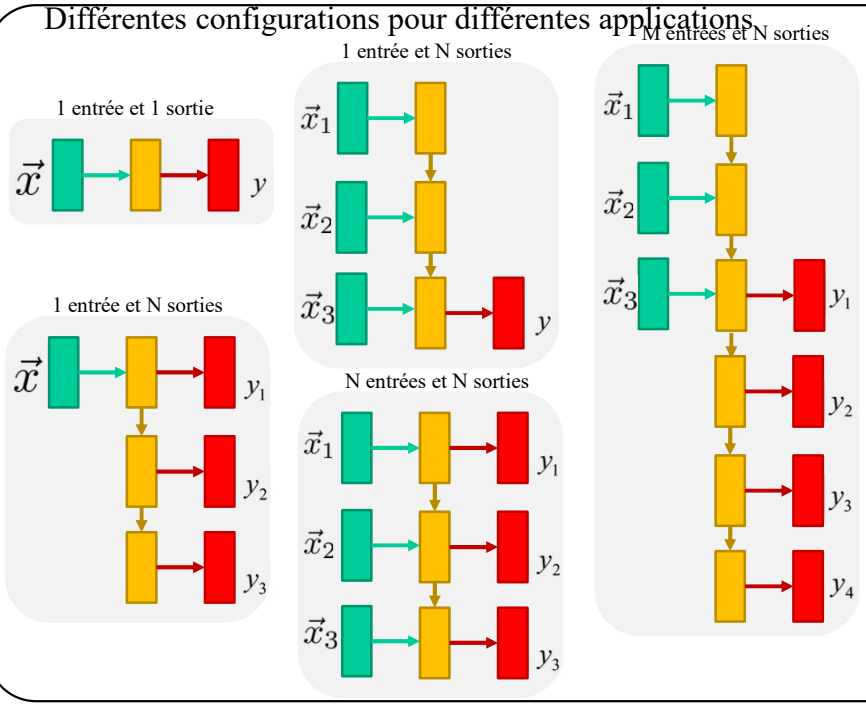
```
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

Crédit: A. Karpathy, CS231

240

Différentes configurations pour différentes applications

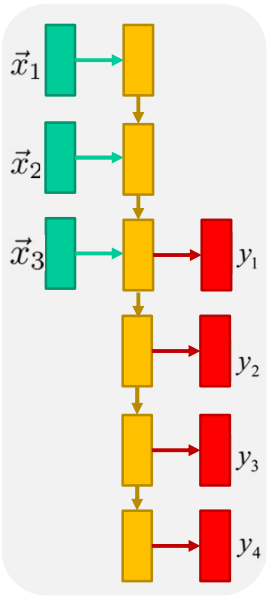


241

Traduction Français-Anglais

N lettres => M lettres

M entrées et N sorties



242

Autre exemple: **traduction**

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

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 : Begining of Sentence, EoS:End of Sentence).

243

Autre exemple: **traduction**

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

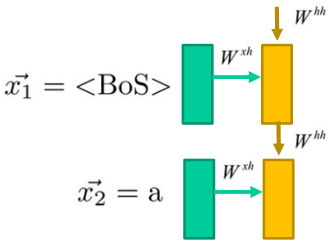
Traduire ‘assez’ -> ‘enough’
Alphabet fr :[**<BoS>,a,e,s,z,<EoS>**]
Alphabet en: **[<BoS>,e,g,h,n,o,u,<EoS>]**

244

Autre exemple: **traduction**

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

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

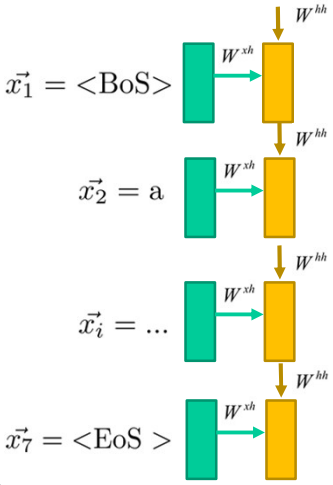


245

Autre exemple: **traduction**

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

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

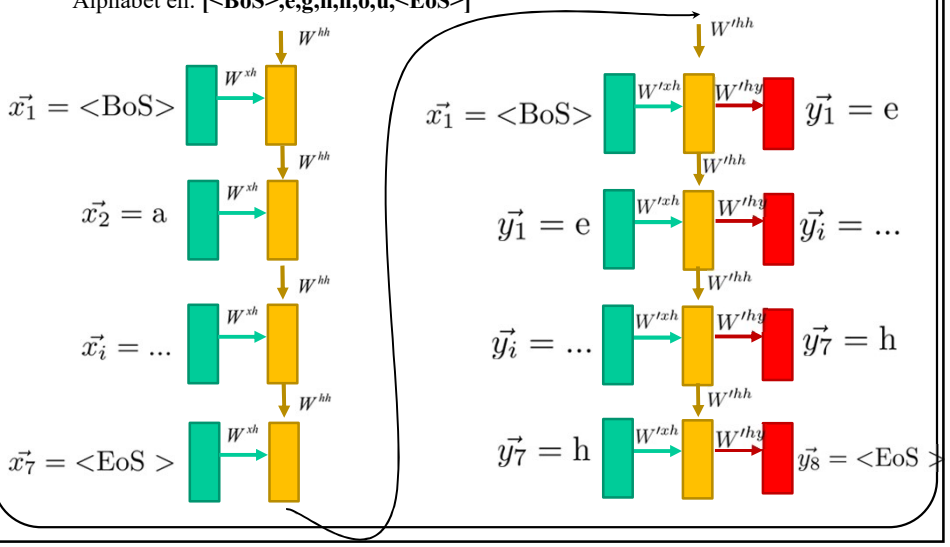


246

Autre exemple: **traduction**

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

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

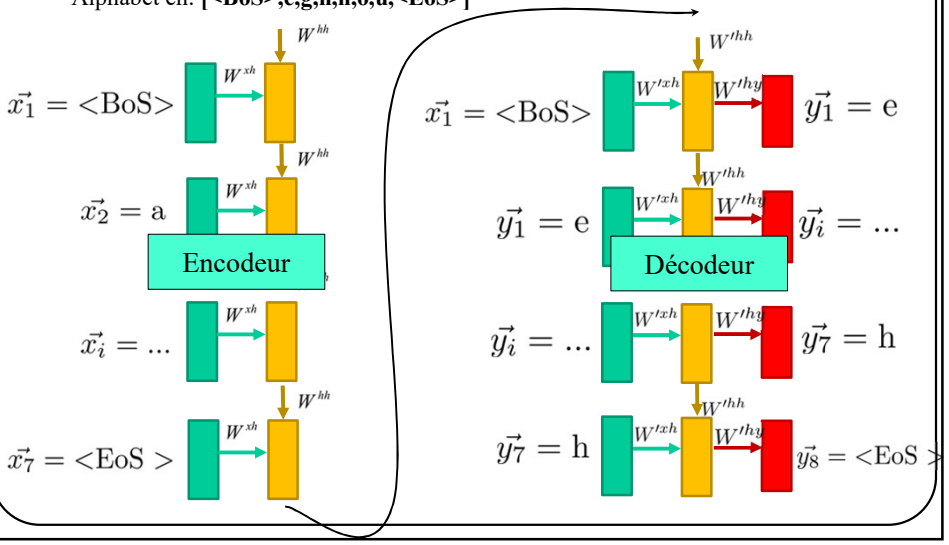


247

Autre exemple: **traduction**

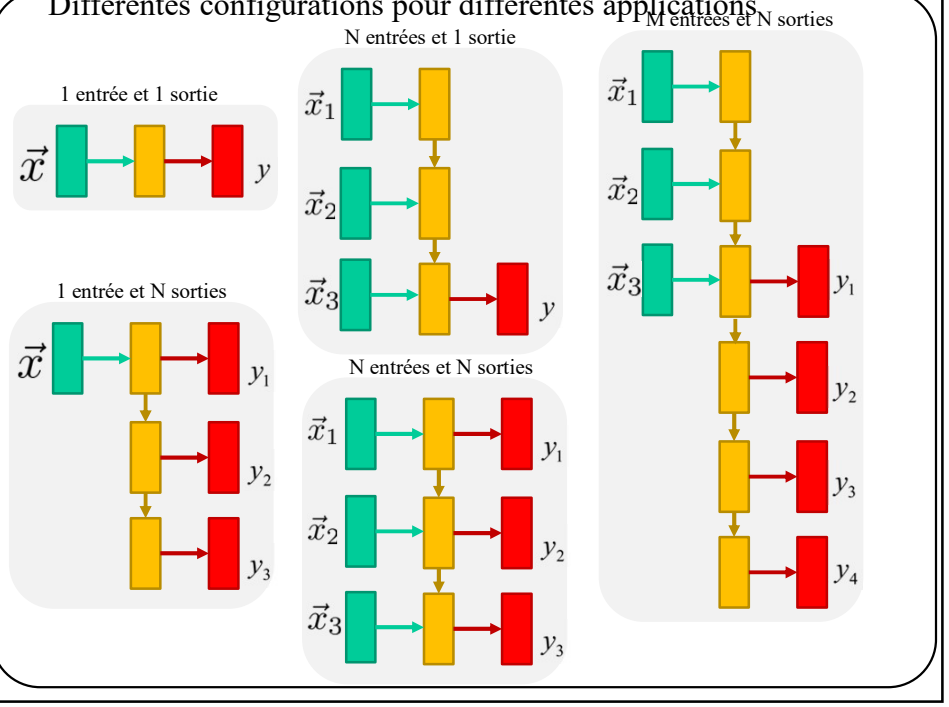
Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

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



248

Différentes configurations pour différentes applications

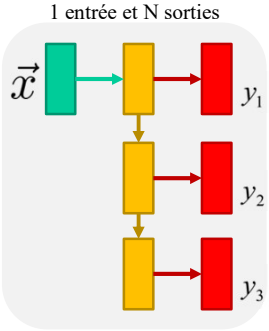


249

Différentes configurations pour différentes applications

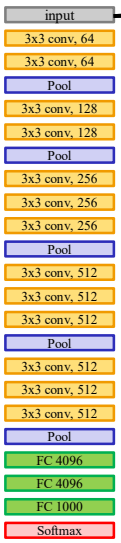
Description du contenu d'une image (« captioning »)

1 entrée et N sorties



250

Captioning



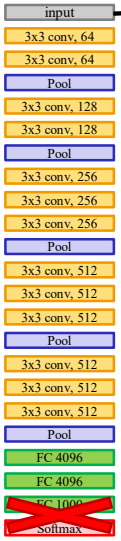
VGG16

Réseau VGG pré-entraîné sur *ImageNet*

251

251

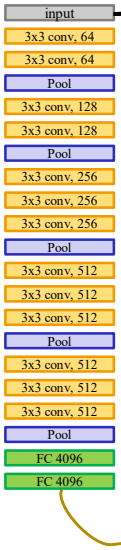
Captioning

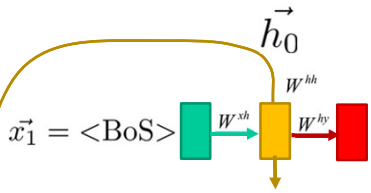


VGG16

252

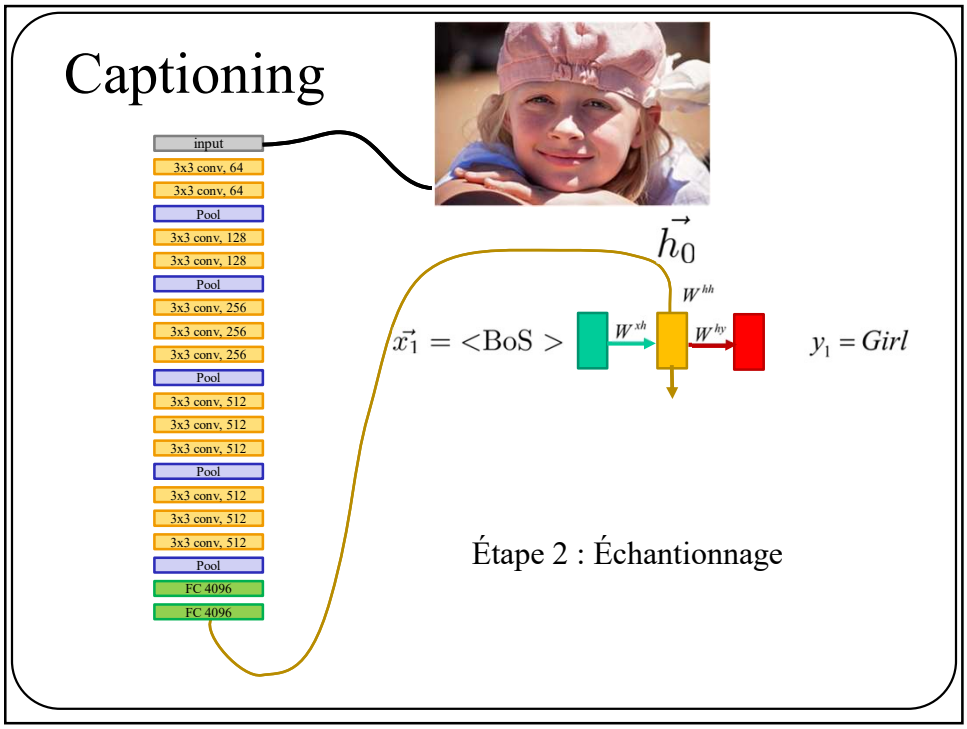
Captioning



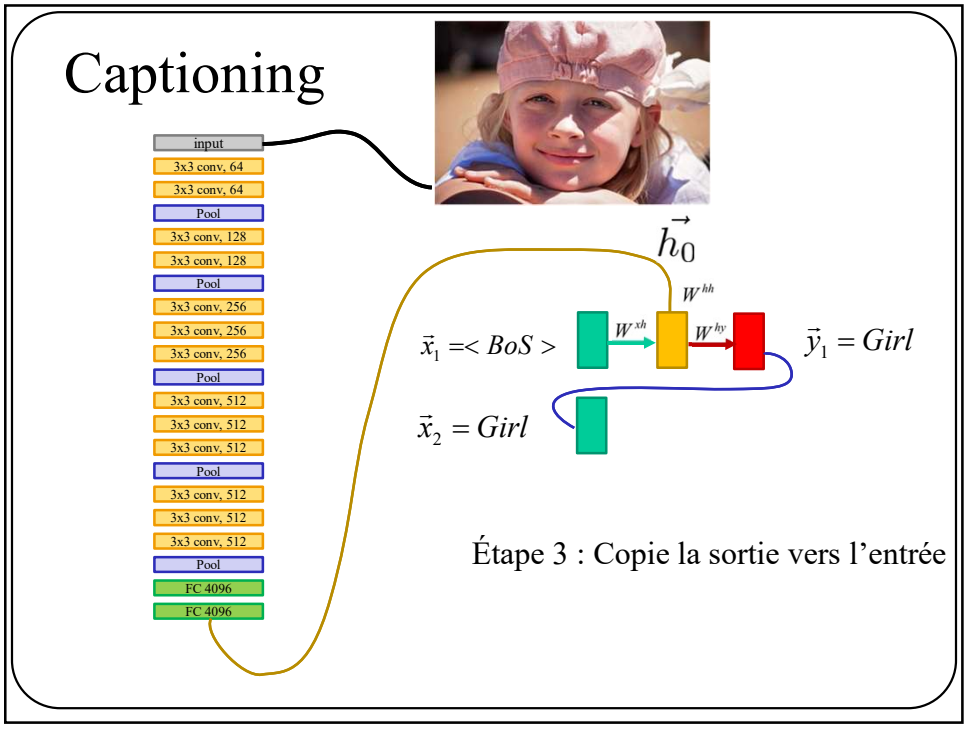


Étape 1 : Init + Propagation avant

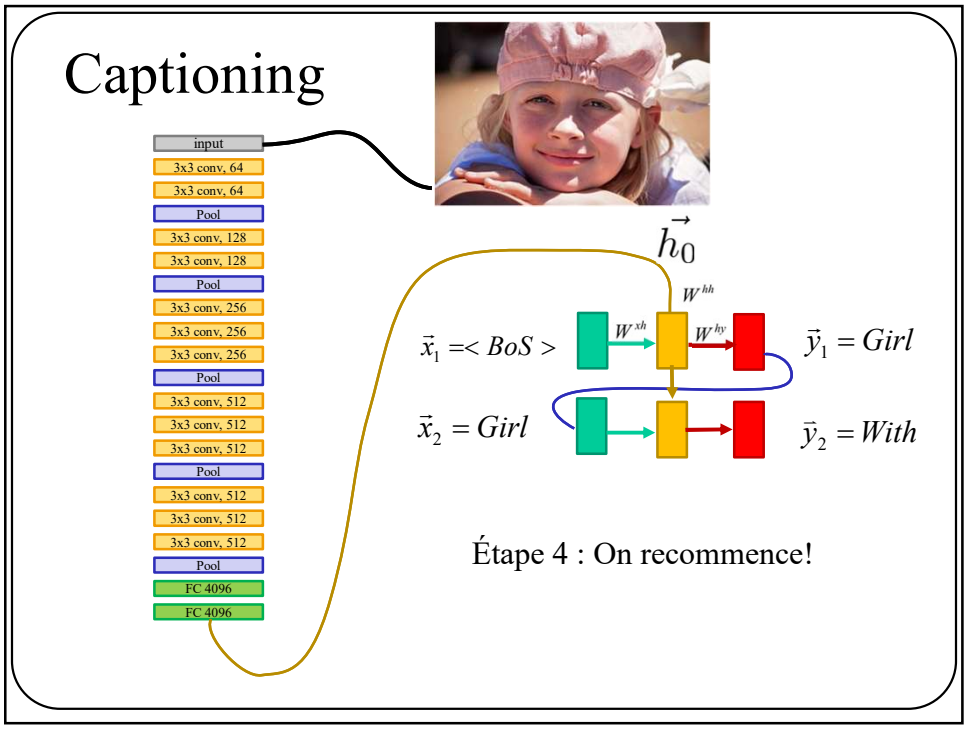
253


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
$\vec{h}_0$
W^{hh}
$\vec{x_1}$ = <BoS >
W^{xh}
W^{hy}
y_1 = Girl
Étape 2 : Échantionnage

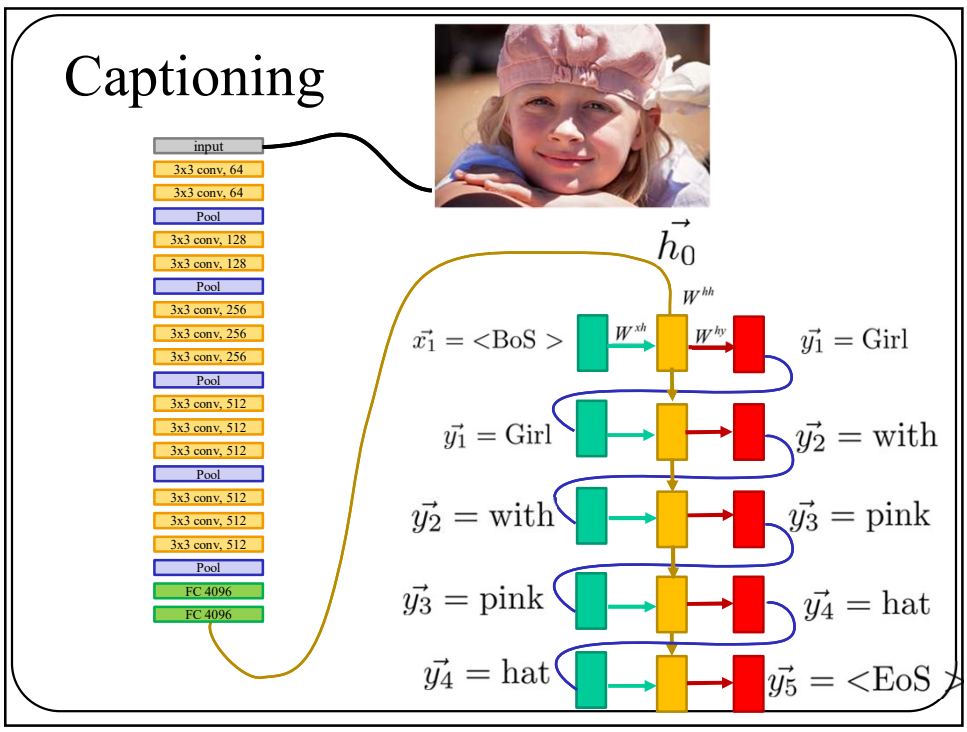
254


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
$\vec{h}_0$
W^{hh}
$\vec{x}_1$ =< BoS >
W^{xh}
W^{hy}
$\vec{y}_1$ = Girl
$\vec{x}_2$ = Girl
Étape 3 : Copie la sortie vers l'entrée

255


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
$\vec{h}_0$
W^{hh}
W^{xh}
W^{hy}
$\vec{x}_1 =< BoS >$
$\vec{y}_1 = Girl$
$\vec{x}_2 = Girl$
$\vec{y}_2 = With$
Étape 4 : On recommence!

256


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
$\vec{h}_0$
W^{hh}
W^{xh}
W^{hy}
$\vec{x}_1 = <\text{BoS}>$
$\vec{y}_1 = \text{Girl}$
$\vec{y}_1 = \text{Girl}$
$\vec{y}_2 = \text{with}$
$\vec{y}_2 = \text{with}$
$\vec{y}_3 = \text{pink}$
$\vec{y}_3 = \text{pink}$
$\vec{y}_4 = \text{hat}$
$\vec{y}_4 = \text{hat}$
$\vec{y}_5 = <\text{EoS}>$

257

Exemples de résultats

https://github.com/karpathy/neuraltalk2

an elephant standing in a grassy field with trees in the background

a man riding a wave on top of a surfboard

a street sign on a pole in front of a building

a group of people playing a game with nintendo wii controllers

a couple of zebra standing on top of a dirt field

258

258

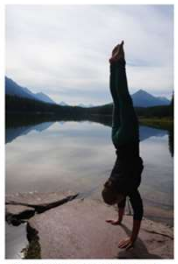
Exemples d'erreurs https://github.com/karpathy/neuraltalk2

a man is throwing a frisbee in a park

a man riding a skateboard down a street

a laptop computer sitting on top of a wooden desk

A woman is holding a cat in her hand



A woman standing on a beach holding a surfboard

259

259