

Réseaux de neurones IFT 780

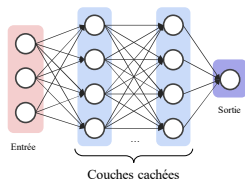
Réseaux à convolution avancés et architectures convolutives modernes

Par
Pierre-Marc Jodoin

1

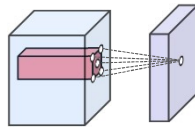
Réseaux de neurones à convolution

Réseaux de neurones multicouches



Beaucoup de paramètres
Moins de neurones

CNN



- Paramètres partagés
 - Connectivité locale
- Moins de paramètres
Beaucoup de neurones
Architectures plus profondes

2

2

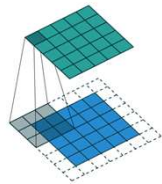
Réseaux de neurones à convolution

Convolution padding



Pas de padding
Stride = 1
Filtre 3x3

Carte d'activation plus petite
que le signal d'entrée



Zero padding
Stride = 1
Filtre 3x3

Carte d'activation de même taille
que le signal d'entrée

Images: <http://deeplearning.net/software/theano/tutorial>

3

3

Réseaux de neurones à convolution

Champ récepteur d'un neurone ("receptive field")

Région dans l'image d'entrée à l'intérieur de laquelle une variation pourrait avoir un effet sur la sortie d'un neurone.

Plus un neurone a un **champ récepteur large**, plus ce dernier encode du **contexe**.

4

4

Réseaux de neurones à convolution

Taille et convolution

K filtres $F_1 \times F_2$

- La **profondeur** du filter = le **nombre de canaux** en entrée (ici P)
- Le **nombre de cartes d'activation** = le **nombre de filtres** (ici K)
- Filter convolutif 1D est un tenseur 2D ($F \times P$)
- Filter convolutif 2D est un tenseur 3D ($F_1 \times F_2 \times P$)
- K filtres convolutifs 2D est un tenseur 4D ($K \times F_1 \times F_2 \times P$)

5

Réseaux de neurones à convolution

Taille et convolution

K filtres $F_1 \times F_2$

Taille du volume d'activation (2D) :

$D1' = (D1 - F1) / S + 1$

$D2' = (D2 - F2) / S + 1$

$K = \text{nb_filtres}$

6

6

Réseaux de neurones à convolution

Chaque couche convolutive contient une fonction d'activation

S'assure que tous les neurones sont activés

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

$\tanh(x)$



ReLU
 $\max(0, x)$

$\max(0, x)$



Réduit le problème de la
disparition du gradient

Leaky ReLU
 $\max(0.1x, x)$

$\max(0.1x, x)$



Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



7

7

Réseaux de neurones à convolution

Pooling

Max Pooling

29	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

2 x 2
pool size
(stride = 2)

100	184
12	45

Average Pooling

31	15	28	184
0	100	70	38
12	12	7	2
12	12	45	6

2 x 2
pool size
(stride = 2)

36	80
12	15

BUT

- Réduire la **résolution** des cartes d'activation
- Moins de **mémoire** et de **calculs**
- Augmente un peu la **robustess** à la rotation et au changement d'échelle

L'opération de *pooling* est généralement faite indépendamment sur chaque carte d'activation.

Images: <https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks/>

8

8

Réseaux de neurones à convolution

CNN = suite d'opérations convolutives (et autres, FC = *fully connected*)

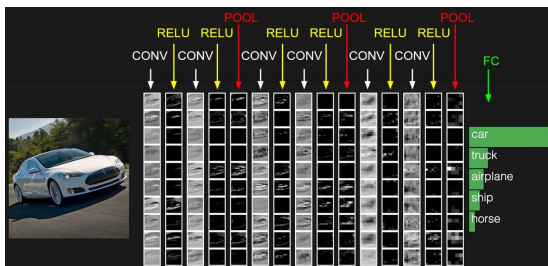


Image: <http://cs231n.github.io>

9

9

Autres types de couches convolutives

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

10

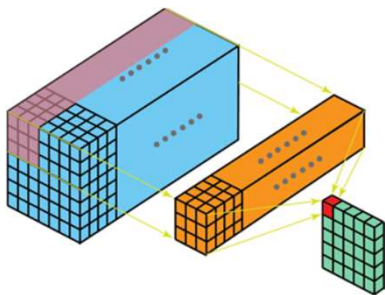
Convolution **séparée**

11

11

Convolution classique

Représentation classique d'une convolution 2D, ici un volume en entrée de taille **7x7xP** convolué par un **filtre 3x3xP** donne une **carte d'activation 5x5x1**

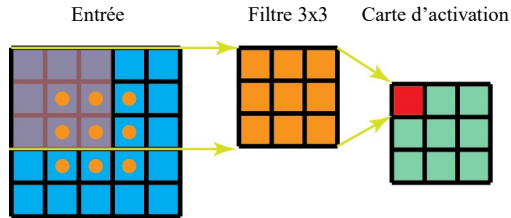


<https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215>

12

Convolution classique

Comme on le sait, une convolution « valid » avec **1 canal** en entrée et un filtre 2D s'opère ainsi:



<https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215> ¹³

13

Convolution séparée

Les filtres 2D et 3D peuvent être **séparés** en filtres plus élémentaires

3 filtres 2D
ayant
9 paramètres

$$h = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} / 9$$

$$h = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} / 16$$

$$h = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} / 3$$

14

Convolution séparée

Les filtres 2D et 3D peuvent être **séparés** en filtres plus élémentaires

3 filtres 2D
ayant
9 paramètres

$$h = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} / 9 \rightarrow h_x = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} / 3, h_y = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} / 3$$

$$h = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} / 16 \rightarrow h_x = \begin{pmatrix} 1 & 2 & 1 \end{pmatrix} / 4, h_y = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} / 4$$

$$h = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} / 3 \rightarrow h_x = \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}, h_y = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} / 3$$

Ces filtres 2D
peuvent être
décomposés
en 2 filtres de
3 paramètres
pour un total de
6 paramètres

15

Convolution séparée

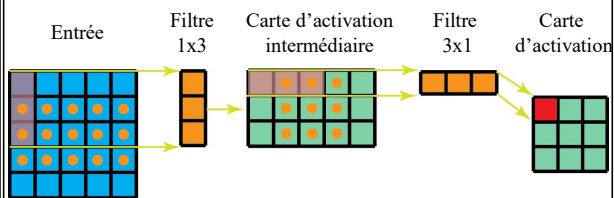
Les filtres 2D et 3D peuvent être **séparés** en filtres plus élémentaires

Bien que tous les filtres 2D (et 3D) ne soient pas tous mathématiquement séparables, on peut tout de même les **approximer** par des filtres 1D

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \approx \begin{pmatrix} k & l & m \end{pmatrix} * \begin{pmatrix} n \\ o \\ p \end{pmatrix}$$

16

Convolution séparée

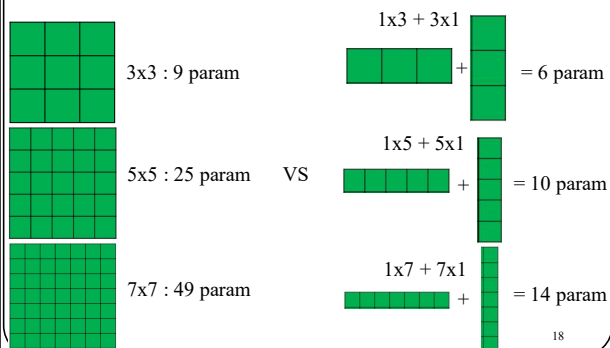


<https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281c58215>

17

Convolution séparée

Réduction du nombre de paramètres



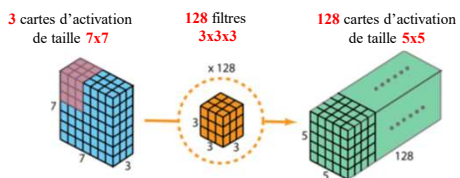
18

Convolution séparée en profondeur (Depthwise Separable Convolution)

19

19

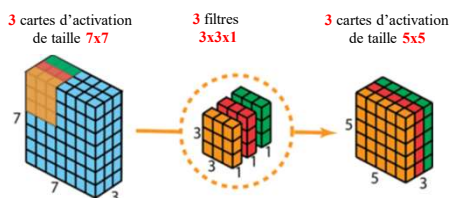
Convolution usuelle



$$3 \times 3 \times 3 \times 128 = 3456 \text{ params}$$

20

depth-wise convolution



21

Depthwise Separable Convolution

Depth-wise convolution + conv 1x1

3 cartes d'activation de taille 7x7 3 filtres 3x3x1 3 cartes d'activation de taille 5x5 1 filtre 1x1x3 1 carte d'activation de taille 5x5

22

<https://www.deeplibrary.com/course/convolutional-neural-networks-introduction-to-different-types-of-convolutions-in-deep-learning-660281c58216>

22

Depthwise Separable Convolution

Depth-wise convolution + conv 1x1

3 cartes d'activation de taille 7x7 3 filtres 3x3x1 3 cartes d'activation de taille 5x5 128 filtres 1x1x3 128 cartes d'activation de taille 5x5

$3 \times 3 \times 3 + 3 \times 128 = 411 \text{ params}$

23

<https://www.deeplibrary.com/course/convolutional-neural-networks-introduction-to-different-types-of-convolutions-in-deep-learning-660281c58216>

23

Depthwise Separable Convolution

Depth-wise convolution + conv 1x1

3 cartes d'activation de taille 7x7 1 filtre 3x1x1 1 filtre 1x3x1 1 filtre 1x1x3 1 carte d'activation de taille 5x5

24

<https://www.deeplibrary.com/course/convolutional-neural-networks-introduction-to-different-types-of-convolutions-in-deep-learning-660281c58216>

24

Depthwise Separable Convolution

Depth-wise convolution + conv 1x1

3 cartes d'activation de taille 7x7 1 filtre 3x1x1 1 filtre 1x3x1 1 filtre 1x1x3 128 cartes d'activation de taille 5x5

x 128

$(3 \times 1 \times 1 + 1 \times 3 \times 1 + 1 \times 1 \times 3) \times 128 = \mathbf{1158 \text{ params}}$

Jonghoon Jin, Aysegül Dundar, Eugenio Culurciello, Flattened convolutional neural networks for feedforward acceleration, ICCV2017

25

Convolution 3D

26

26

Convolution 3D

Très utile en imagerie médicale

Image « T2 »

256 256 52

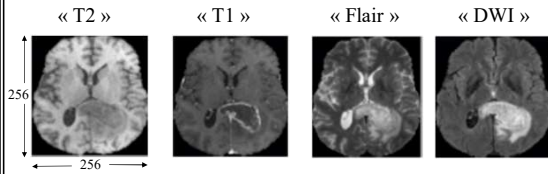
32 filtres de taille 5x5x1

convolution 2D usuelle

27

Convolution 3D

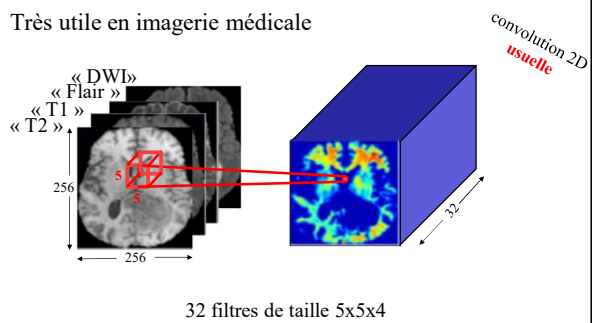
Souvent en imagerie médicale on utilise plusieurs « modalités »



28

Convolution 3D

Très utile en imagerie médicale

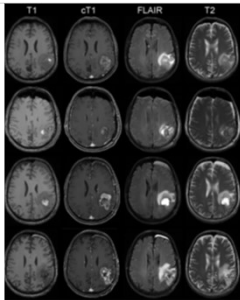


32 filtres de taille 5x5x4

29

Convolution 3D

Souvent en imagerie médicale on utilise plusieurs « modalités » **et des acquisitions en 3D** (ici un cerveau au complet)



30

Convolution 3D

Souvent en imagerie médicale on utilise plusieurs « modalités » **et des acquisitions en 3D**

31

Convolution 3D

Très utile en imagerie médicale

convolution 3D

1 filtres de taille 5x5x5x4

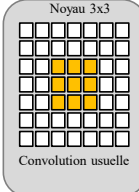
32

Convolution *à trous*

33

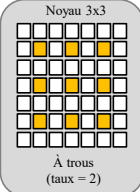
33

Convolution à trous (ou dilatée)



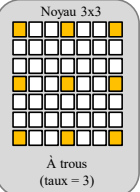
Noyau 3x3
Convolution usuelle

Champ récepteur=3x3



Noyau 3x3
À trous
(taux = 2)

Champ récepteur=5x5

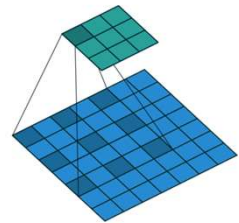


Noyau 3x3
À trous
(taux = 3)

Champ récepteur=7x7

34

Convolution à trous (ou dilatée)



- Augmentation “artificielle” de la taille du filtre en insérant des zéros entre les éléments non-nuls du noyau
- 1 hyper-paramètre *taux* (*taux* = 1 convolution ordinaire)

Images: <http://deeplearning.net/software/theano/tutorial>

35

Autres pratiques courantes

Prétraitement des données en entrée

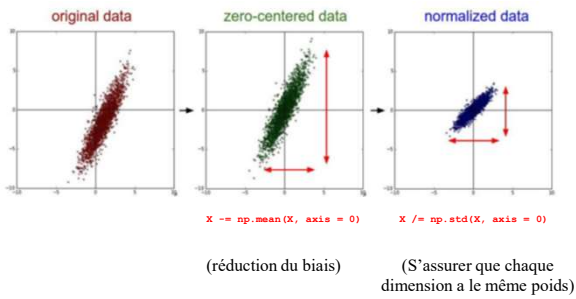
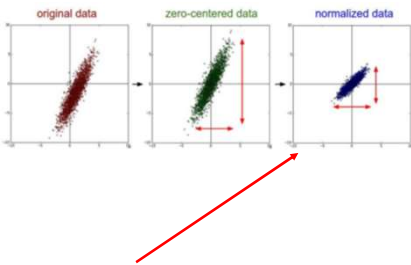


Image: <http://cs231n.github.io>

37

Prétraitement des données en entrée

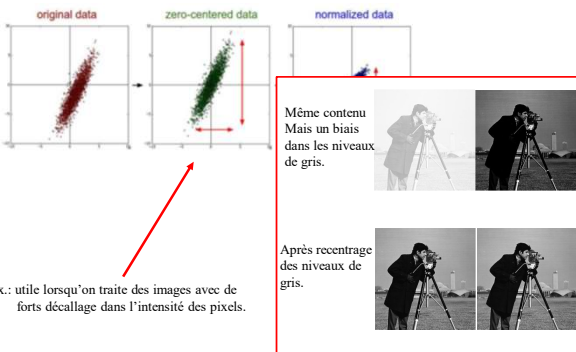


S'assurer que les dimensions ont les mêmes variances.

Exemple: âge maison (1 à 100) vs prix maison (100,000 à 2,000,000).

38

Prétraitement des données en entrée



39

Prétraitement des données en entrée

Pour des images RGB (ex. CIFAR10, CIFAR100, ImageNet, etc)

- Soustraire l'image moyenne des données d'entraînement (e.g AlexNet)
 - Soustraire une image 32x32x3 pour CIFAR10

$$x_{MOY} = \frac{1}{N} \sum_{i=1}^N x_i$$

40

40

Prétraitement des données en entrée

Pour des images RGB (ex. CIFAR10, CIFAR100, ImageNet, etc)

- Soustraire l'image moyenne des N images d'entraînement (e.g AlexNet)
 - Soustraire une image moyenne 32x32x3 pour CIFAR10

$$x_{MOY} = \frac{1}{N} \sum_{i=1}^N x_i$$

- Soustraire une moyenne par canal (e.g. VGGNet)
 - Soustraire **trois valeurs** : R, G, B

$$R = \frac{1}{N} \sum_{k=1}^N \sum_{i,j} x_k[i,j].R$$

$$G = \frac{1}{N} \sum_{k=1}^N \sum_{i,j} x_k[i,j].G$$

$$B = \frac{1}{N} \sum_{k=1}^N \sum_{i,j} x_k[i,j].B$$

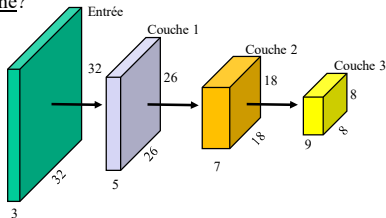
41

41

Normalisation

Observation : Normaliser les données en entrée est une bonne chose.

Question : pourquoi ne pas normaliser les données à l'entrée de chaque couche?



42

42

Normalisation : Plusieurs options

- Instance normalization**
- Layer normalization**
- Batch normalization**
- Group normalization**

Pour plus de détails
<https://medium.com/nerd-for-tech/overview-of-normalization-techniques-in-deep-learning-c12a79060daf>

43

Normalisation : Plusieurs options

- Instance normalization
- Layer normalization**
- Batch normalization**
- Group normalization

Pour plus de détails
<https://medium.com/nerd-for-tech/overview-of-normalization-techniques-in-deep-learning-c12a79060daf>

44

Normalisation par couche (*Layer norm*)

Entrée

Couche 1

Couche 2

Couche 3

3 32 32

5 26 26

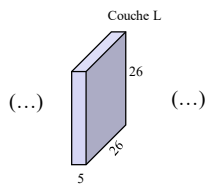
7 18 18

9 8 8

Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton "Layer normalization" arXiv:1607.06450v1

45

Normalisation par couche (*Layer norm*)

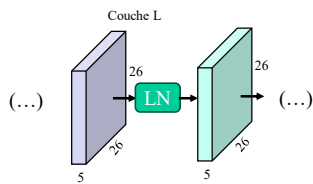


Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton "Layer normalization" arXiv:1607.06450v1

46

46

Normalisation par couche (*Layer norm*)

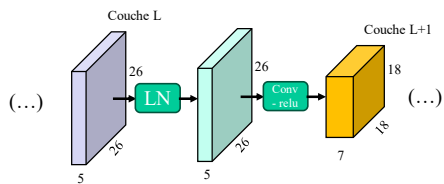


Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton "Layer normalization" arXiv:1607.06450v1

47

47

Normalisation par couche (*Layer norm*)

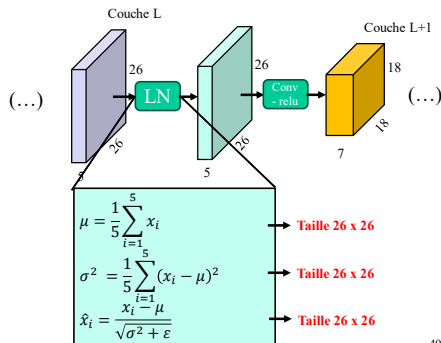


Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton "Layer normalization" arXiv:1607.06450v1

48

48

Normalisation par couche (*Layer norm*)



Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton "Layer normalization" arXiv:1607.06448v1

49

49

Normalisation par couche (*Layer norm*)

```
def layernorm_forward_pass(x, eps):
    #step 1 : calculer la moyenne et la variance
    mu = np.mean(x, -1)[:, :, np.newaxis]
    var = np.var(x, -1)[:, :, np.newaxis]

    #step 2 : normaliser les données
    x_norm = (x - mu) / np.sqrt(var + eps)

    return x_norm
```

Question: est-ce pertinent de normaliser tous les neurones de toutes les couches? **Pas toujours!**

50

50

Normalisation par couche (*Layer norm*)

Solution: permettre au réseau d'apprendre à *défaire* la normalisation

$$\begin{aligned} \mu &= \frac{1}{5} \sum_{i=1}^5 x_i && \rightarrow \text{Taille } 26 \times 26 \\ \sigma^2 &= \frac{1}{5} \sum_{i=1}^5 (x_i - \mu)^2 && \rightarrow \text{Taille } 26 \times 26 \\ \hat{x}_i &= \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} && \rightarrow \text{Taille } 26 \times 26 \\ \tilde{x}_i &= \gamma \circ \hat{x}_i + \beta && \rightarrow \gamma \text{ et } \beta \text{ de taille } 26 \times 26 \end{aligned}$$

Paramètres appris par le système. Ainsi, le réseau peut apprendre que $\gamma = \gamma$ et $\beta = \mu$ et ainsi annuler la normalisation au besoin.

51

51

Normalisation par couche (*Layer norm*)

NOTE: produit de Hadamar

$$\tilde{x}_i = \gamma \circ \hat{x}_i + \beta$$

$$\begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \gamma_4 \\ \gamma_5 & \gamma_6 & \gamma_7 & \gamma_8 \\ \gamma_9 & \gamma_{10} & \gamma_{11} & \gamma_{12} \\ \gamma_{13} & \gamma_{14} & \gamma_{15} & \gamma_{16} \end{bmatrix} \odot \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix} = \begin{bmatrix} \gamma_1 x_1 & \gamma_2 x_2 & \gamma_3 x_3 & \gamma_4 x_4 \\ \gamma_5 x_5 & \gamma_6 x_6 & \gamma_7 x_7 & \gamma_8 x_8 \\ \gamma_9 x_9 & \gamma_{10} x_{10} & \gamma_{11} x_{11} & \gamma_{12} x_{12} \\ \gamma_{13} x_{13} & \gamma_{14} x_{14} & \gamma_{15} x_{15} & \gamma_{16} x_{16} \end{bmatrix}$$

52

Normalisation par couche (*Layer norm*)

```
def layernorm_forward_pass(x, eps):

    #step 1 : calculer la moyenne et la variance
    mu = np.mean(x, -1)[:, :, np.newaxis]
    var = np.var(x, -1)[:, :, np.newaxis]

    #step 2 : normaliser les données
    x_norm = (x - mu) / np.sqrt(var + eps)

    #step 3 : "dénormaliser" les données
    x_norm = x_norm * self.gamma + self.beta

    return x_norm
```

53

Normalisation par lot (*Batch norm*)

[Ioffe and Szegedy, 2015]

Observation : Normaliser les données en entrée est une bonne chose.

Question : pourquoi ne pas normaliser les données à l'entrée de chaque couche?

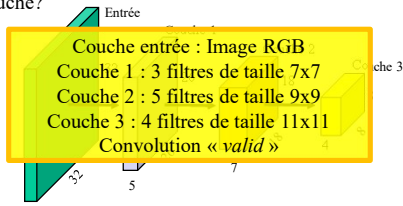
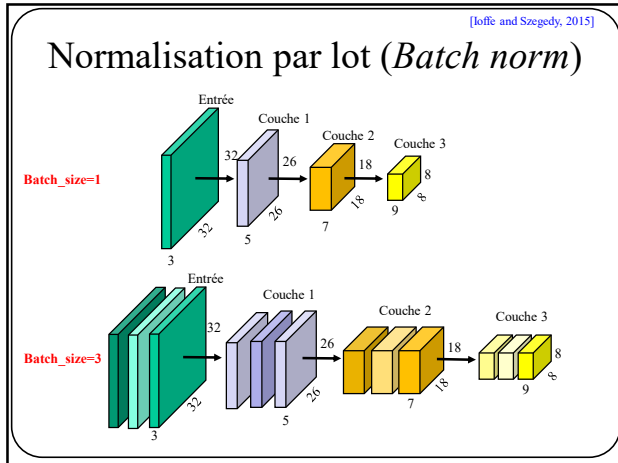
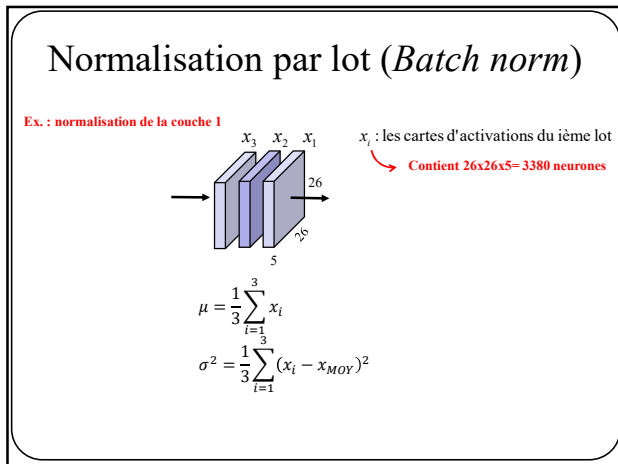


Image: Ioffe et al. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv, 2015.

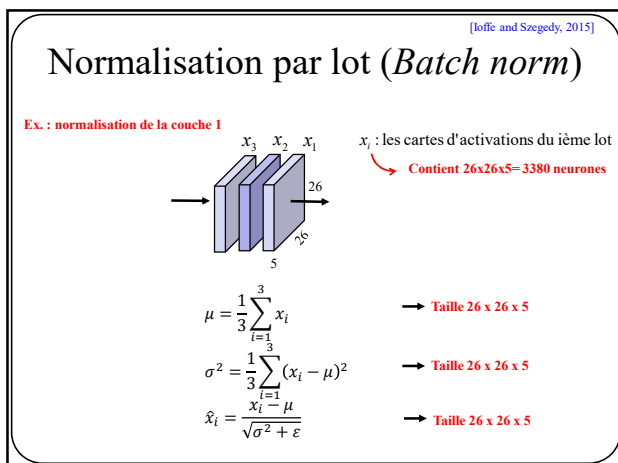
54



55



56



57

Normalisation par lot (*Batch norm*)

```
def batchnorm_forward_pass(x, eps):

    #step 1 : calculer la moyenne et la variance
    mu = np.mean(x, axis=0)
    var = np.var(x, axis=0)

    #step 2 : normaliser les données
    x_norm = (x - mu)/np.sqrt(var + eps)

    return x_norm
```

Question: est-ce pertinent de normaliser tous les neurones de toutes les couches? **Pas toujours!**

58

58

Normalisation par lot (*Batch norm*)

Solution: permettre au réseau d'**apprendre à défaire** la normalisation par lot

$$x_{MOY} = \frac{1}{3} \sum_{i=1}^3 x_i$$

$$x_{VAR} = \frac{1}{3} \sum_{i=1}^3 (x_i - x_{MOY})^2$$

$$\hat{x}_i = \frac{x_i - x_{MOY}}{\sqrt{x_{VAR} + \epsilon}}$$

$$\tilde{x}_i = \gamma \circ \hat{x}_i + \beta$$

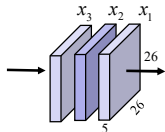
Paramètres appris par le système. Ainsi, le réseau peut apprendre que $\gamma = \sqrt{x_{VAR}}$ et $\beta = x_{MOY}$ et ainsi annuler la normalisation au besoin.

59

59

Normalisation par lot (*Batch norm*)

Ex. : normalisation de la couche 1



x_i : les cartes d'activations du ième lot
 Contient 26x26x5=3380 neurones

$$\mu = \frac{1}{3} \sum_{i=1}^3 x_i \quad \rightarrow \text{Taille } 26 \times 26 \times 5$$

$$\sigma^2 = \frac{1}{3} \sum_{i=1}^3 (x_i - \mu)^2 \quad \rightarrow \text{Taille } 26 \times 26 \times 5$$

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \rightarrow \text{Taille } 26 \times 26 \times 5$$

$$\tilde{x}_i = \gamma \circ \hat{x}_i + \beta \quad \rightarrow \gamma \text{ et } \beta \text{ de taille } 26 \times 26 \times 5$$

60

Normalisation par lot (*Batch norm*)

```
def batchnorm_forward_pass(x, eps):

    #step 1 : calculer la moyenne et la variance
    mu = np.mean(x, axis=0)
    var = np.var(x, axis=0)

    #step 2 : normaliser les données
    x_norm = (x - mu)/np.sqrt(var + eps)

    #step 3 : "dénormaliser" les données
    x_norm = x_norm*self.gamma + self.beta

    return x_norm
```

61

61

Normalisation par lot (*Batch norm*)

En généralisation, lorsqu'on souhaite traiter une seule donnée (donc une taille de lot de 1), on remplace et par des constantes précalculées

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

c'est-à-dire

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i$$

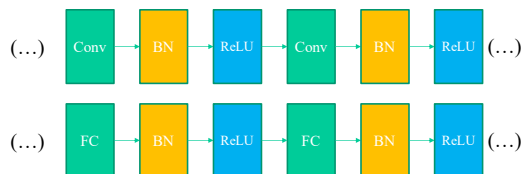
$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

Image: Ioffe et al. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv, 2015.

62

Normalisation (*Batch ou layer norm*)

Généralement **insérée entre la convolution** (ou la couche pleinement connectée) **et la non-linéarité**.



63

63

Normalisation

Avantages:

Accélère l'entraînement

Permet d'utiliser un plus gros Learning rate

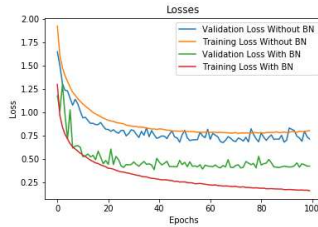


Image: Ioffe et al. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv, 2015.

64

Normalisation (*Batch et layer norm*)

Pour plus d'information:

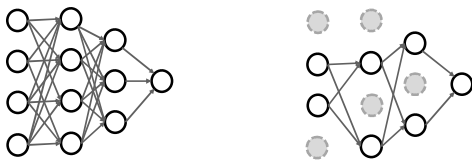
- <https://medium.com/@ilango100/batch-normalization-speed-up-neural-network-training-245e39a62f85>
- <https://deepnotes.io/batchnorm>
- Ioffe et al. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv, 2015.
- <https://medium.com/nerd-for-tech/overview-of-normalization-techniques-in-deep-learning-e12a79060daf>

65

65

Autre pratique courante : *Dropout*

Forcer à zéro certains neurones de façon aléatoire à chaque itération



Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

66

Autre bonne pratique : *Dropout*

```
p = 0.5 # probability of keeping a unit active, higher = less dropout

def train_step(X):
    """ X contains the data """

    # forward pass for example 3-layer neural network
    H1 = np.maximum(0, np.dot(W1, X) + b1)
    U1 = np.random.rand(*H1.shape) < p # first dropout mask
    H1 *= U1 # drop!
    H2 = np.maximum(0, np.dot(W2, H1) + b2)
    U2 = np.random.rand(*H2.shape) < p # second dropout mask
    H2 *= U2 # drop!
    out = np.dot(W3, H2) + b3

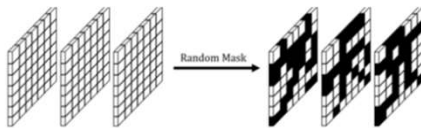
    # backward pass: compute gradients... (not shown)
    # perform parameter update... (not shown)
```

Crédit <http://cs231n.stanford.edu/>

67

Autre bonne pratique : *Dropout*

Dans le cas d'un réseau à convolution, dropout revient à appliquer un **masque binaire aléatoire** à chaque carte d'activation.

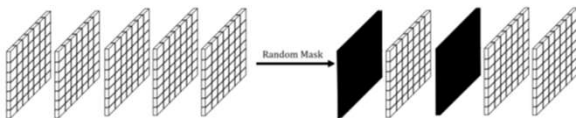


Jonathan Tompson, Ross Girshick, Arjan Jain, Yann LeCun, Christoph Bregler
"Efficient Object Localization Using Convolutional Networks", in proc of CVPR 2015

68

Spatial dropout, une variante du *dropout*

Au lieu de forcer à zéro des neurones,
on peut forcer à zéro des **cartes d'activation**.



Jonathan Tompson, Ross Girshick, Arjan Jain, Yann LeCun, Christoph Bregler
"Efficient Object Localization Using Convolutional Networks", in proc of CVPR 2015

69

69

Drop connect, autre variante de dropout

Dropout :
forcer des neurones à zéro

Dropconnect :
forcer des **poids** à zéro

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus Regularization of Neural Networks using DropConnect, ICML 2013

70

Drop connect, autre variante de dropout

Dans le cas d'une convolution

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

 $*$

W0	W1	W2
W3	W4	W5
W6	W7	W8

 $=$

Y0	Y1
Y2	Y3

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus Regularization of Neural Networks using DropConnect, ICML 2013

71

Drop connect, autre variante de dropout

Dans le cas d'une convolution

Dropout force les **neurones** à zéro

X0		X2	X3
	X5	X6	
X8	X9		X11
	X13	X14	

 $*$

W0	W1	W2
W3	W4	W5
W6	W7	W8

 $=$

Y0	Y1
Y2	Y3

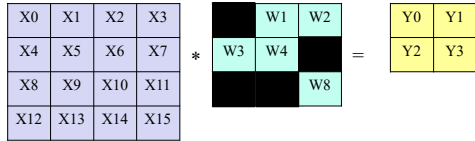
Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus Regularization of Neural Networks using DropConnect, ICML 2013

72

Drop connect, autre variante de dropout

Dans le cas d'une convolution

Dropconnect force les **paramètres** à zéro



Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus Regularization of Neural Networks using DropConnect, ICML 2013

73

Ensemble de modèles

- 1- Entraîner indépendamment différents modèles
- 2- En généralisation, **faire voter** ces modèles

Permet d'améliorer les performances de 2-3%

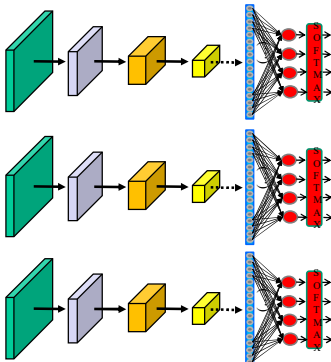
NOTE Même entraîner N-fois le même modèle fonctionne!

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus Regularization of Neural Networks using DropConnect, ICML 2013

74

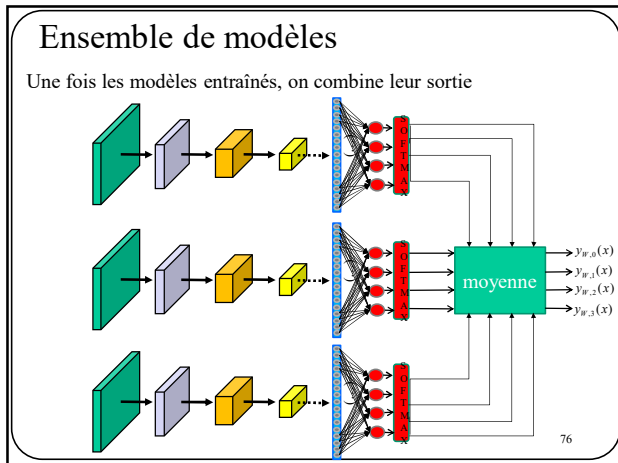
Ensemble de modèles

Entraîner plusieurs modèles (ou plusieurs fois le même modèle)

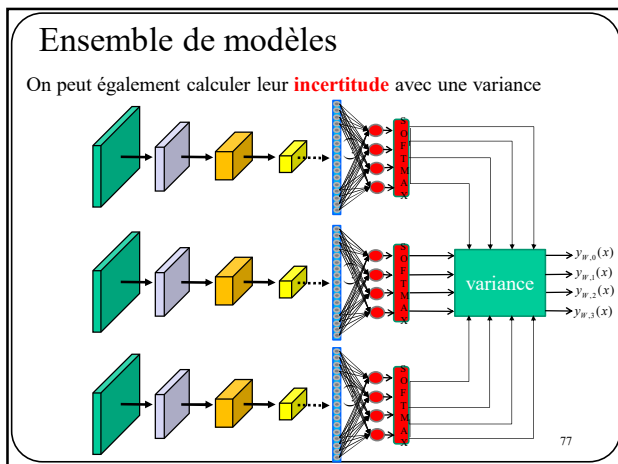


75

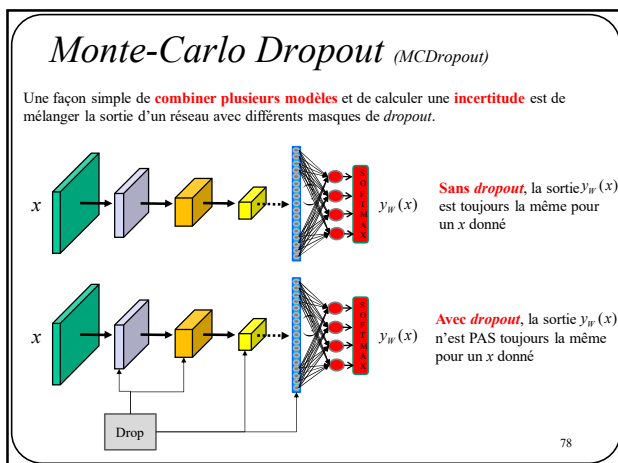
75



76



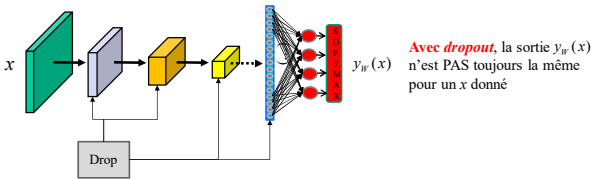
77



78

Monte-Carlo Dropout (MCDropout)

Une façon simple de **combiner plusieurs modèles** et de calculer une **incertitude** est de mélanger la sortie d'un réseau avec différents masques de *dropout*.



Avec dropout, la sortie $y_w(x)$ n'est PAS toujours la même pour un x donné

Si on fait M propagations avanti pour un x donné, on aura M résultats différents.

$$\begin{aligned}\hat{y}_w(x) &= \frac{1}{M} \sum_{m=1}^M y_w^m(x) && \longrightarrow \text{Prédictions moyennées} \\ &&& \text{(ensemble de modèles)} \\ \hat{y}_w^{VAR}(x) &= \frac{1}{M} \sum_{m=1}^M (y_w^m(x) - \hat{y}_w(x))^2 && \longrightarrow \text{Incertainitude}\end{aligned}$$

Prédictions moyennées
(ensemble de modèles)

Incertitude

79

Transfert d'apprentissage

(Transfer learning)

Question : il faut un très grand nombre de données annotées pour entraîner un réseaux de neurones profonds?

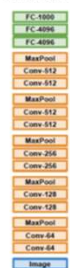
Réponse : Faux, si on dispose d'un modèle pré-entraîné sur une base de données similaire.

80

Transfert d'apprentissage

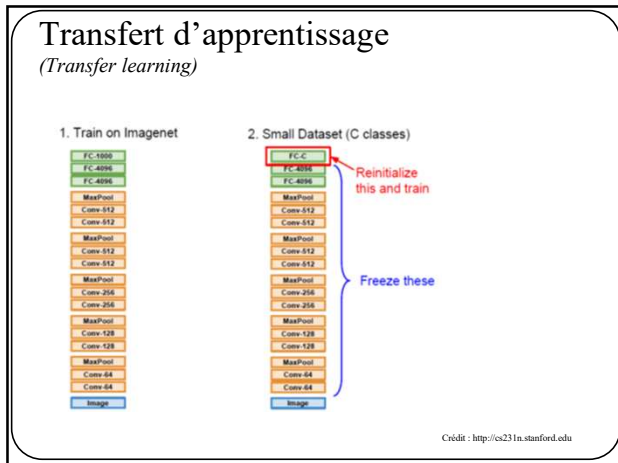
(Transfer learning)

1. Train on Imagenet

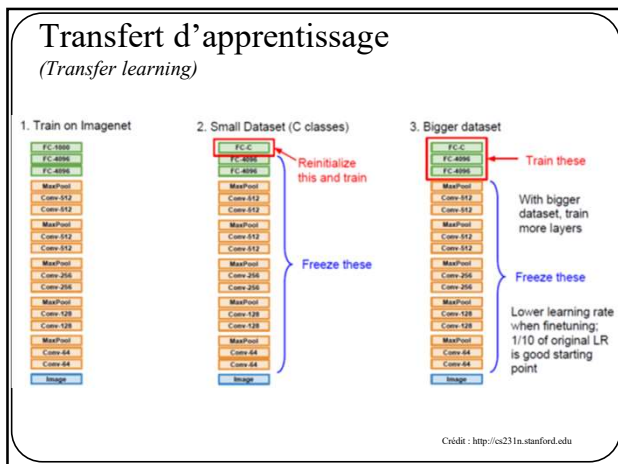


Crédit : <http://cs231n.stanford.edu>

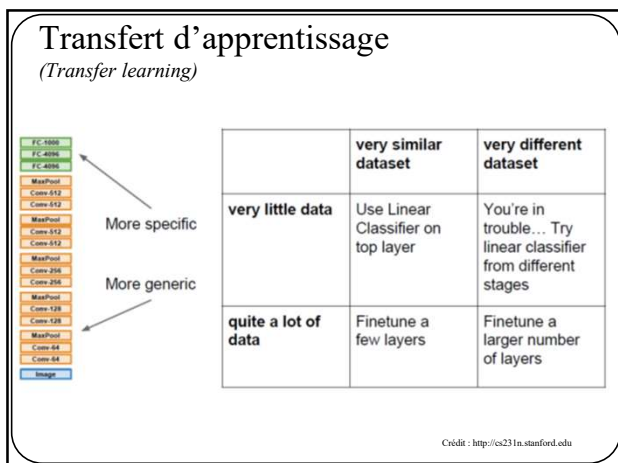
81



82



83



84

À retenir pour vos projets:

Vous avez une BD qui a un nombre limité de données annotées?

1. Trouvez une grosse BD contenant des données similaires
2. Entraînez un réseau de neurones
3. Transférez le modèle à votre projet
4. Réentraînez votre modèle (ou une partie de votre modèle)

Plusieurs bibliothèques ont un "Model Zoo" avec des modèles pré-entraînés

TensorFlow: <https://github.com/tensorflow/models>

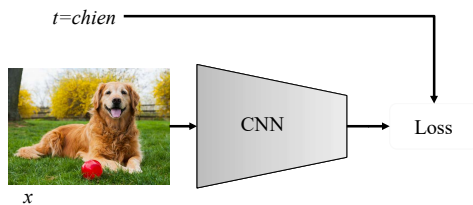
PyTorch: <https://github.com/pytorch/vision>

85

Augmentation de données

(Data augmentation)

On peut artificiellement augmenter le nombre de données en transformant les données sans pour autant affecter leur cible.

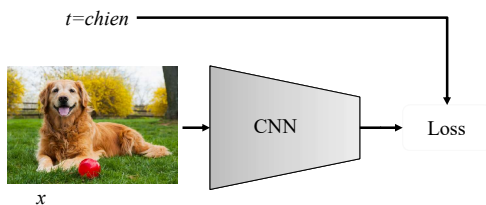


86

Augmentation de données

(Data augmentation)

Exemple de transformation : **flip**

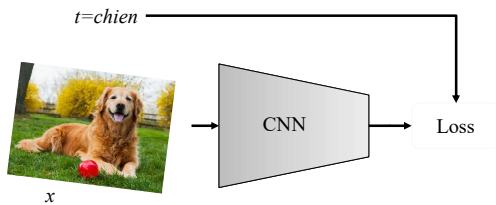


87

Augmentation de données

(Data augmentation)

Exemple de transformation : **rotation aléatoire de +/- 5 degrés**

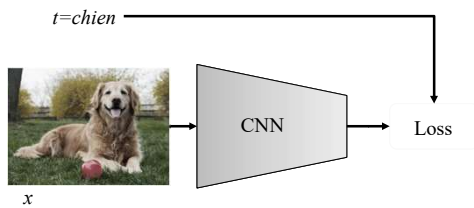


88

Augmentation de données

(Data augmentation)

Exemple de transformation : **changement aléatoire de la dynamique des couleurs**

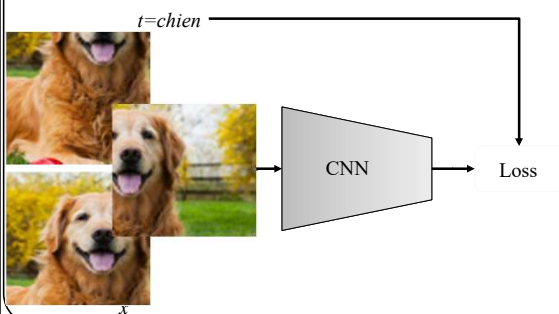


89

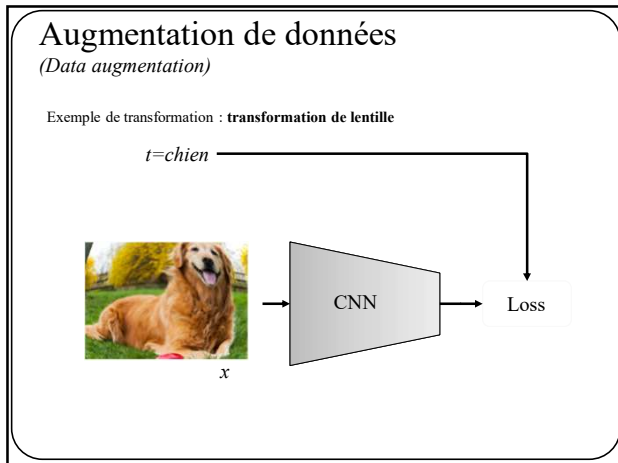
Augmentation de données

(Data augmentation)

Exemple de transformation : **crop aléatoire + redimension**



90



91

L'augmentation de données n'est pas une exception

c'est la norme

Il n'y a *a priori* **aucune raison pour ne pas l'utiliser dans vos projets.**

92

CLASSIFICATION D'IMAGE

93

Classification d'image



Chien: 85%
Chat: 10%
Cheval: 5%
Chaise: 0%

But: prédire une étiquette de classe (ou une distribution de probabilités) pour une image donnée.

94

94

Classification d'image

Défi



Ce que l'humain voit

17	49	43	43	43	45	51	54	45	39
47	64	88	37	48	56	37	47	81	47
56	87	84	39	88	64	31	48	49	31
28	49	32	75	48	49	71	25	47	27
51	82	33	44	48	45	35	48	38	48
34	77	26	36	46	27	42	76	79	45
48	36	40	37	34	42	72	41	69	46
35	48	82	38	67	43	54	77	72	72
42	85	65	79	72	62	64	54	73	81
39	52	35	44	45	79	34	76	68	57
55	68	25	56	62	77	78	89	79	25
35	47	68	43	42	55	55	57	59	39
48	41	72	75	75	39	63	58	75	48
35	28	33	48	71	88	26	48	71	76
22	83	31	88	58	67	79	67	69	38
48	49	31	75	46	66	67	32	89	38
42	58	35	49	42	64	32	29	89	38
59	56	36	47	58	51	54	89	30	81
75	79	43	38	49	63	63	82	81	35

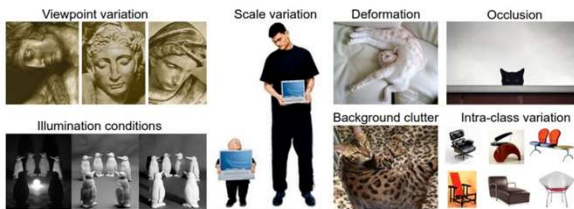
Ce que l'ordinateur voit

95

95

Classification d'image

Autres défis



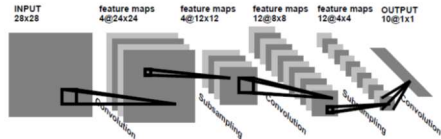
Facile pour les humains mais difficile pour les ordinateurs.

Image: <http://cs231n.github.io>

96

96

LeNet-1



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

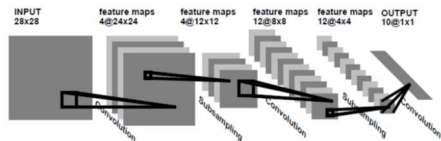
- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 4 et 12 filtres)
- 2 *average pooling*
- 1 couche pleinement connectée
- 10 classes

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

97

LeNet-1



Une des plus vieilles architectures faite 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

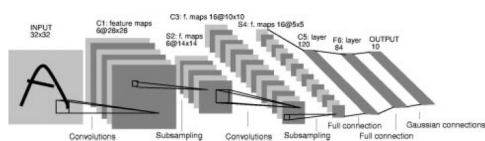
Input
CONV
POOL
CONV
POOL
FC
SOFTMAX

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

98

LeNet-5



Version améliorée:

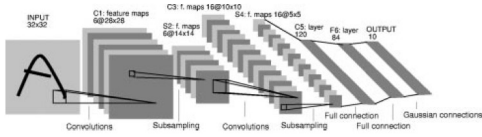
- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles (avec 6 et 16 filtres)
- 2 *average pooling*
- 3 couches pleinement connectées (120, 84 et 10 neurones)
- 10 classes

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

99

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?

100

Classification d'images

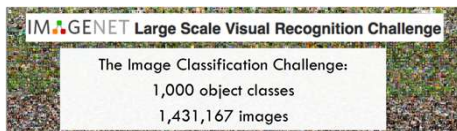


Image: <http://cs231n.stanford.edu/slides/2018>

101

101

Classification d'images

AlexNet [Krizhevsky et al, 2012]

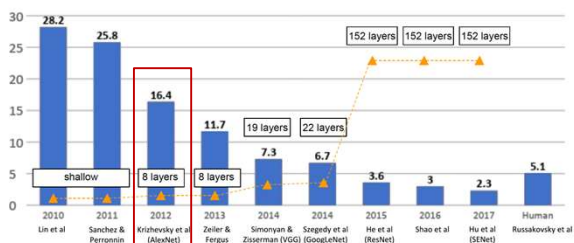
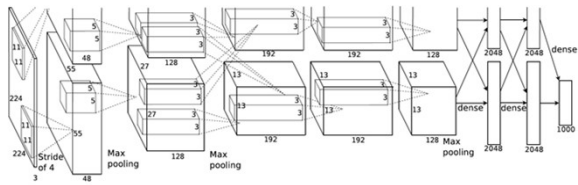


Image: <http://cs231n.stanford.edu/slides/2018>

102

102

AlexNet

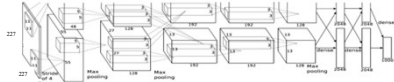


- Premier CNN à bien performer sur ImageNet (amélioration de 10% par rapport aux autres)
- Utilisation de techniques aujourd'hui fréquemment utilisées: **ReLU**, **data augmentation** and **dropout**
- Utilisation de **GPUs** (2 dans leur cas)
- Point de départ de la **révolution du "deep learning"**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

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

103

AlexNet



Architecture:

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

Entrée : image RGB: 227x227x3

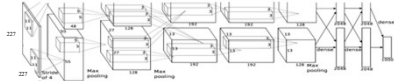
Couche 1 (CONV1): 96 filtres de taille 11x11 avec stride de 4 et conv "valid"

Quelle est la taille des cartes d'activation?

Réponse: $(227-11)/4+1 = 55$

104

AlexNet



Architecture:

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

Entrée : image RGB: 227x227x3

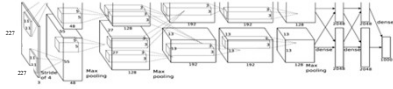
Couche 1 (CONV1): 96 filtres de taille 11x11 avec stride de 4 et conv "valid"

Cartes d'activation : 96 x 55 x 55

Q: Quel est le nombre de paramètres?

105

AlexNet



Architecture:

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

Entrée : image RGB: 227x227x3

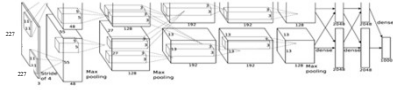
Couche 1 (CONV1): 96 filtres de taille 11x11 avec stride de 4 et conv "valid"

Cartes d'activation : 96 x 55 x 55

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

106

AlexNet



ENTRÉE : 227x227x3

CONV1: 96 x 55 x 55

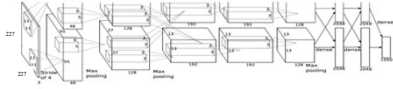
Couche 2 MaxPool : 3x3 stride stride 2

Quelle est la taille des cartes d'activation?

Réponse: $(55-3)/2+1 = 27$

107

AlexNet



ENTRÉE : 227x227x3

CONV1: 96 x 55 x 55

Couche 2 MaxPool : 3x3 stride stride 2

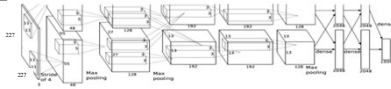
27 x 27 x 96

Combien y a-t-il de paramètres?

Réponse: 0!

108

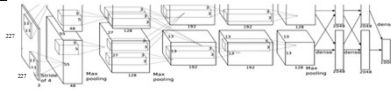
AlexNet



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

109

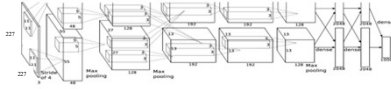
AlexNet



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

110

AlexNet



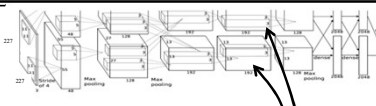
ENTRÉE 227 x 227 x 3
 CONV1 55 x 55 x 96
 MAX POOL1 27 x 27 x 96
 NORM1 27 x 27 x 96
 CONV2 27 x 27 x 256
 MAX POOL2 13 x 13 x 256
 NORM2 13 x 13 x 256
 CONV3 13 x 13 x 384
 CONV4 13 x 13 x 384
 CONV5 13 x 13 x 256
 MAX POOL3 6 x 6 x 256
 FC6 4096
 FC7 4096
 FC8 1000

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

111

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

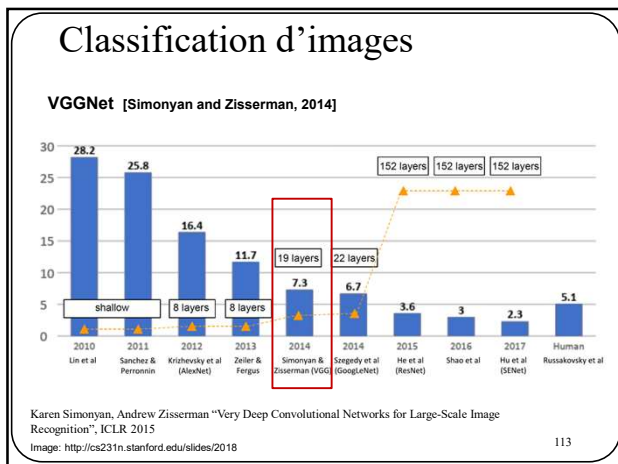
FC8 1000

Notes additionnelles:

Utilisation de **2 GPU**s

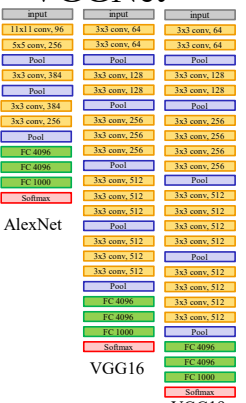
Ensemble de 7 CNN: 18.2% -> 15.4

112



113

VGGNet



AlexNet

VGG16

VGG19

Ce qui caractérise VGGNet par rapport à ses prédécesseurs:

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

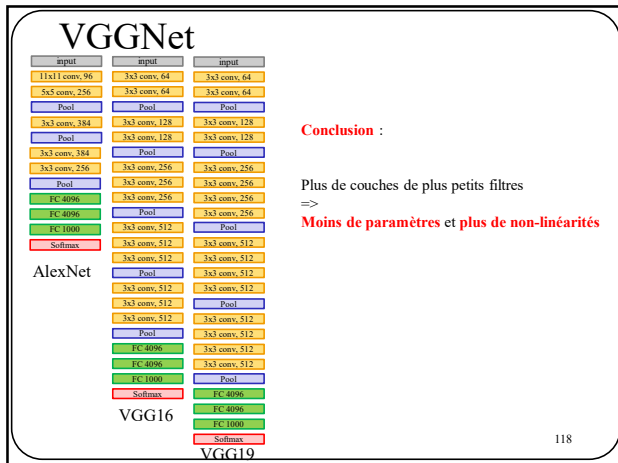
AlexNet : 8 couches

VGGNet : 16 ou 19 couches

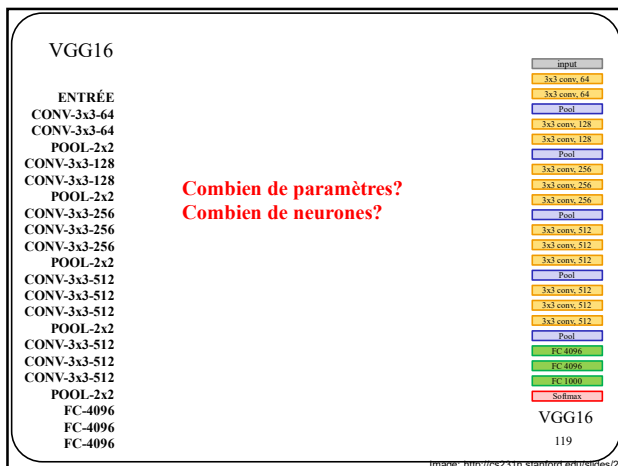
7.3% d'erreur contre 11.7% pour ZFNet

114

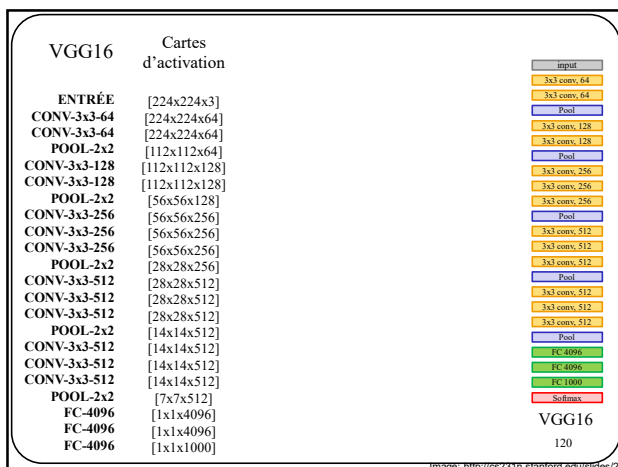
114



118



119



120

VGG16	Cartes d'activation	Nb Neurons	
ENTRÉE	[224x224x3]	150 K	input
CONV-3x3-64	[224x224x64]	3.2 M	3x3 conv, 64
CONV-3x3-64	[224x224x64]	3.2 M	3x3 conv, 64
POOL-2x2	[112x112x64]	800 k	Pool
CONV-3x3-128	[112x112x128]	1.6 M	3x3 conv, 128
CONV-3x3-128	[112x112x128]	1.6 M	3x3 conv, 128
POOL-2x2	[56x56x128]	400 K	Pool
CONV-3x3-256	[56x56x256]	800 K	3x3 conv, 256
CONV-3x3-256	[56x56x256]	800 K	3x3 conv, 256
POOL-2x2	[28x28x256]	200 K	Pool
CONV-3x3-512	[28x28x512]	400 K	3x3 conv, 512
CONV-3x3-512	[28x28x512]	400 K	3x3 conv, 512
POOL-2x2	[14x14x512]	100 K	Pool
CONV-3x3-512	[14x14x512]	100 K	3x3 conv, 512
CONV-3x3-512	[14x14x512]	100 K	3x3 conv, 512
POOL-2x2	[7x7x512]	25 K	Pool
FC-4096	[1x1x4096]	4094	FC, 4096
FC-4096	[1x1x4096]	4096	FC, 4096
FC-1000	[1x1x1000]	1000	FC, 1000
			Softmax
VGG16			121

121

VGG16	Cartes d'activation	Nb Neurons	Nb Paramètres	
ENTRÉE	[224x224x3]	150 K	0	input
CONV-3x3-64	[224x224x64]	3.2 M	$(3*3*3)*64 = 1,728$	3x3 conv, 64
CONV-3x3-64	[224x224x64]	3.2 M	$(3*3*64)*64 = 36,864$	3x3 conv, 64
POOL-2x2	[112x112x64]	800 k	0	Pool
CONV-3x3-128	[112x112x128]	1.6 M	$(3*3*64)*128 = 73,728$	3x3 conv, 128
CONV-3x3-128	[112x112x128]	1.6 M	$(3*3*128)*128 = 147,456$	3x3 conv, 128
POOL-2x2	[56x56x128]	400 K	0	Pool
CONV-3x3-256	[56x56x256]	800 K	$(3*3*128)*256 = 294,912$	3x3 conv, 256
CONV-3x3-256	[56x56x256]	800 K	$(3*3*256)*256 = 589,824$	3x3 conv, 256
POOL-2x2	[28x28x256]	200 K	0	Pool
CONV-3x3-512	[28x28x512]	400 K	$(3*3*256)*512 = 1,179,648$	3x3 conv, 512
CONV-3x3-512	[28x28x512]	400 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
POOL-2x2	[14x14x512]	100 K	0	Pool
CONV-3x3-512	[14x14x512]	100 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
CONV-3x3-512	[14x14x512]	100 K	$(3*3*512)*512 = 2,359,296$	3x3 conv, 512
POOL-2x2	[7x7x512]	25 K	0	Pool
FC-4096	[1x1x4096]	4094	$7*7*512*4096 = 102,760,448$	FC, 4096
FC-4096	[1x1x4096]	4096	$4096*4096 = 16,777,216$	FC, 4096
FC-1000	[1x1x1000]	1000	$4096*1000 = 4,096,000$	FC, 1000
				Softmax
VGG16			122	

122

VGGNet				
				input
				3x3 conv, 64
				3x3 conv, 64
				Pool
				3x3 conv, 128
				3x3 conv, 128
				Pool
				3x3 conv, 256
				3x3 conv, 256
				Pool
				3x3 conv, 512
				3x3 conv, 512
				Pool
				3x3 conv, 512
				3x3 conv, 512
				Pool
				FC, 4096
				FC, 4096
				FC, 1000
				Softmax
				VGG16
				123

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

123

VGGNet

Nb neurones totaux : ~15 M
 Mémoire totale neurones (4 octets par neurones) : ~60 Mo
 Nb paramètres totaux : ~138 M
 Mémoire total paramètres (4 octets par paramètres) : ~552 Mo

~612 Mo pour la propagation avant d'une image

>1.1 Go si on inclut la rétro-propagation

input
 3x3 conv, 64
 3x3 conv, 64
 Pool
 3x3 conv, 128
 3x3 conv, 128
 Pool
 3x3 conv, 256
 3x3 conv, 256
 3x3 conv, 256
 Pool
 3x3 conv, 512
 3x3 conv, 512
 3x3 conv, 512
 Pool
 FC, 4096
 FC, 4096
 FC, 1000
 Softmax

VGG16

124

124

GoogLeNet (Inception V1)

[Szegedy et al., 2014]

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovi, Goog Deeper with Convolutions, CVPR 2015

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

125

GoogLeNet (Inception v1)

CNN "Classique"

Entrée

Conv-ReLu

Conv-ReLu

POOL

Conv-ReLu

Conv-ReLu

POOL

Conv-ReLu

(...)

Conv-ReLu

POOL

FC

Softmax

Sortie

GoogLeNet – 21 couches

Entrée

Conv-ReLu

MaxPool

LRN

Conv-ReLu

LRN

MaxPool

INCEPTION

INCEPTION

MaxPool

INCEPTION

INCEPTION

INCEPTION

INCEPTION

MaxPool

INCEPTION

INCEPTION

AveragePool

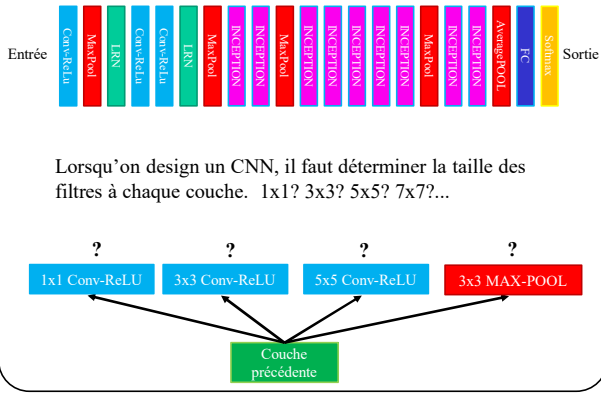
FC

Softmax

Sortie

126

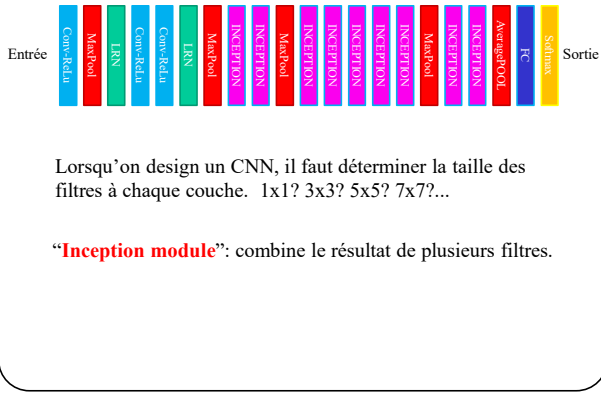
GoogLeNet (Inception v1)



127

[illegible]

GoogLeNet (Inception v1)

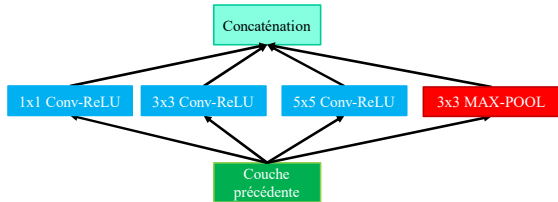


128

[illegible]

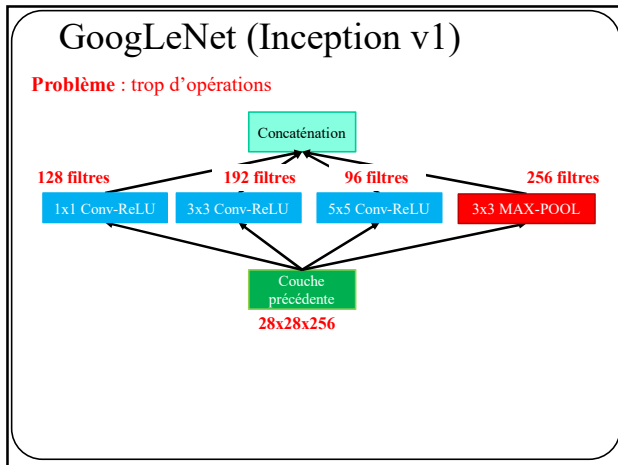
GoogLeNet (Inception v1)

Module d'inception « naïf »

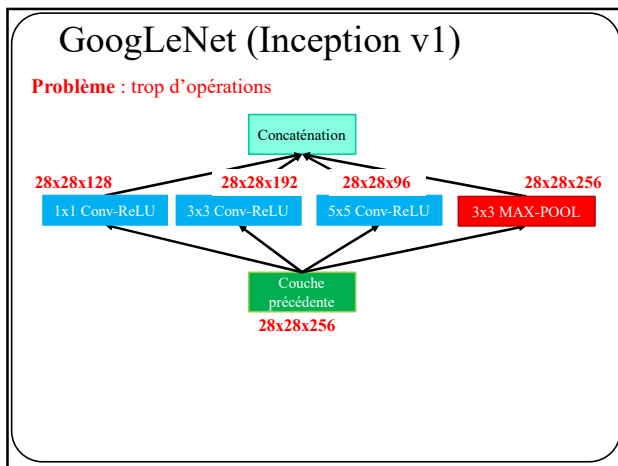


- Appliquer 4 filtres différents et « same »
- Concaténation des cartes d'activation

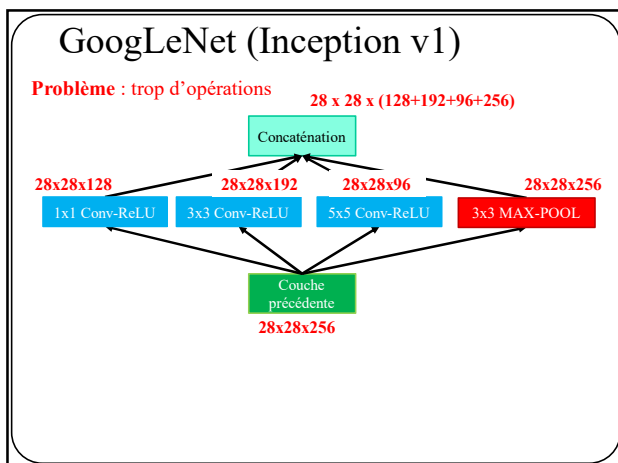
129



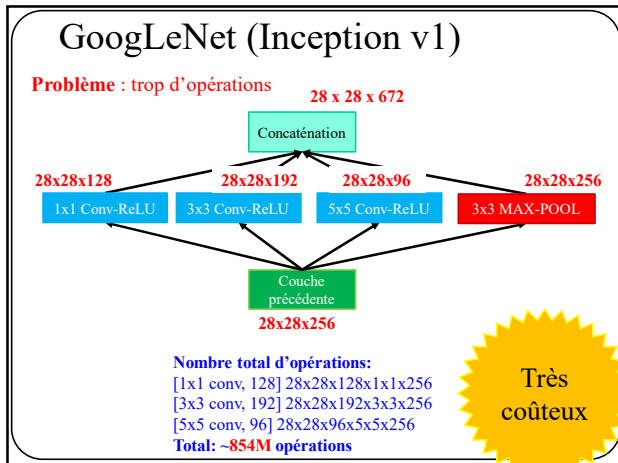
130



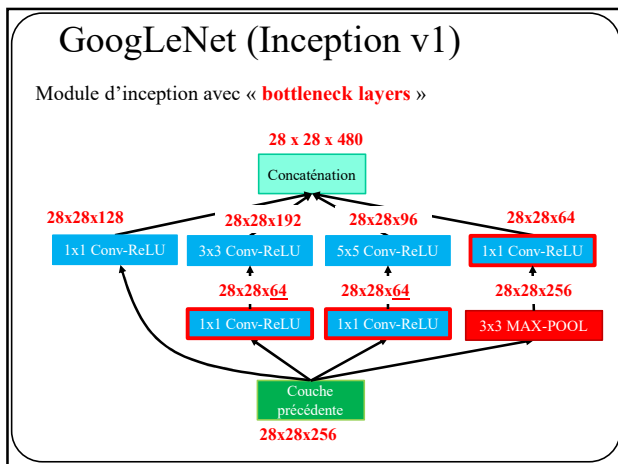
131



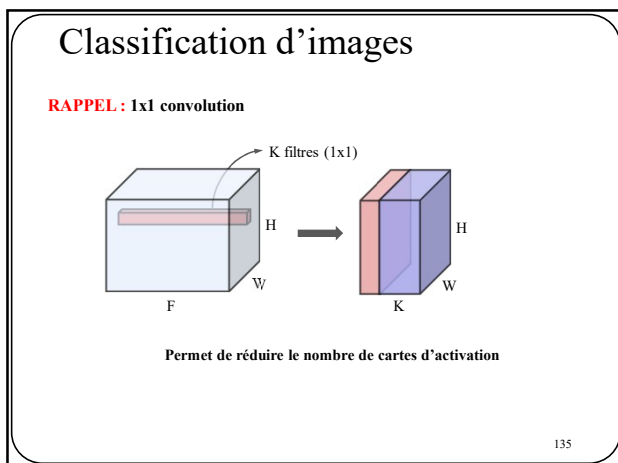
132



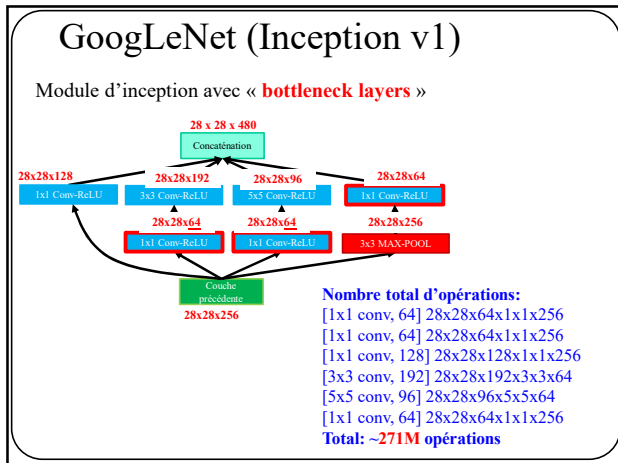
133



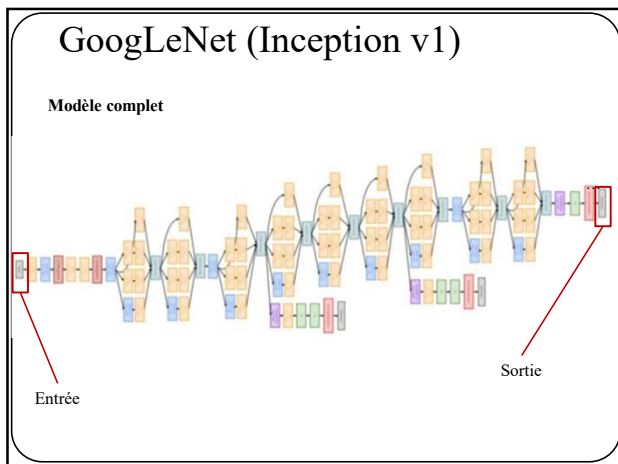
134



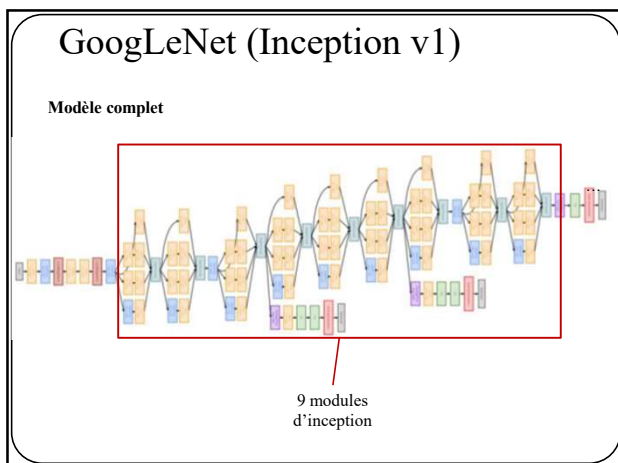
135



136



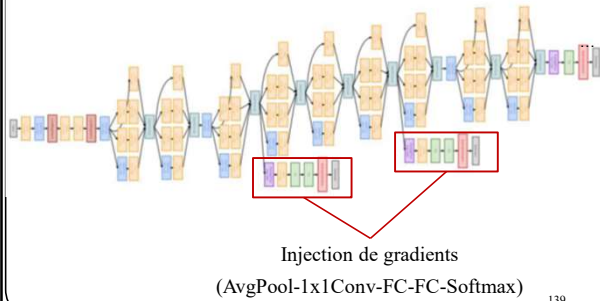
137



138

GoogLeNet (Inception v1)

Modèle complet

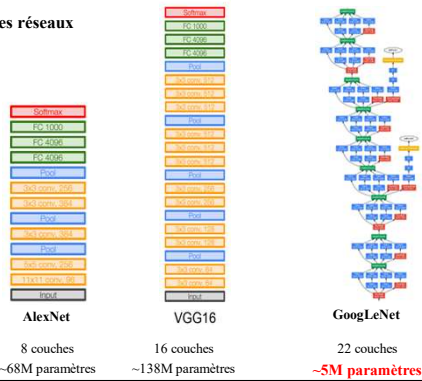


139

139

Classification d'images

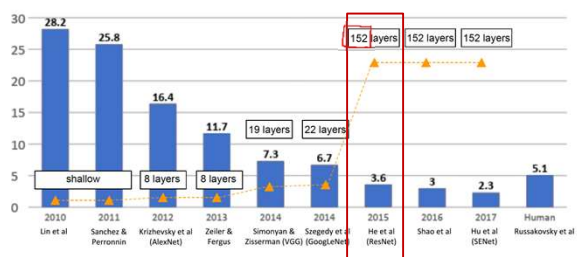
Taille des réseaux



140

Classification d'images

ResNet [He et al, 2016]

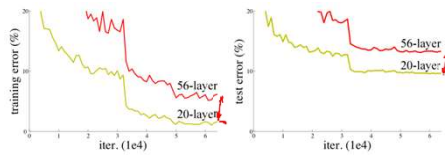


K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, CVPR 2016
Image: <http://cs231n.stanford.edu/slides/2018>

141

Classification d'images

Qu'arrive-t-il lorsqu'on rend très profond un CNN?



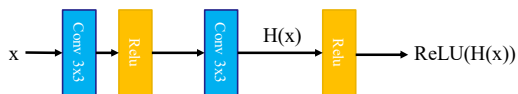
À la fois l'**erreur** de test et d'entraînement vont **augmenter**.

HYPOTHÈSE: Ceci n'est pas du sur-apprentissage (*overfitting*) c'est un **problème d'optimisation**.

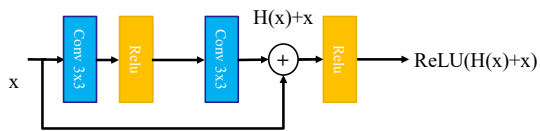
142

Classification d'image

Couches conventionnelles



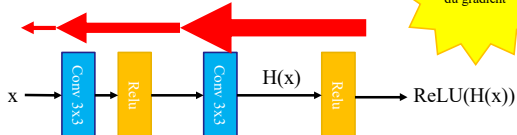
Bloc résiduel



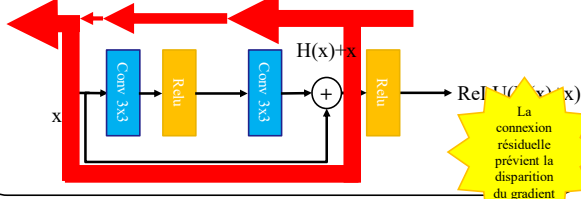
143

Rétro-propagation et gradients

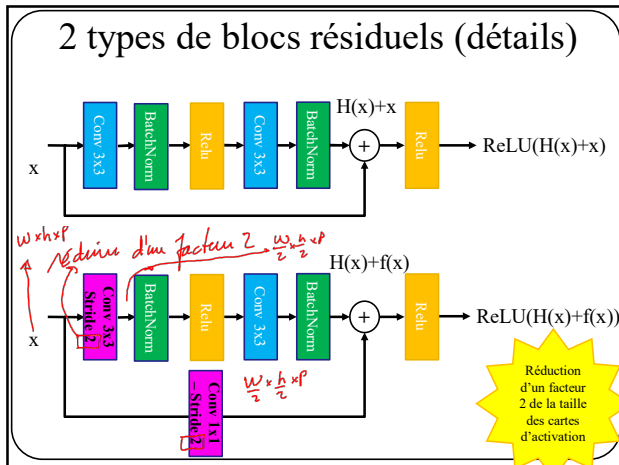
Couches conventionnelles



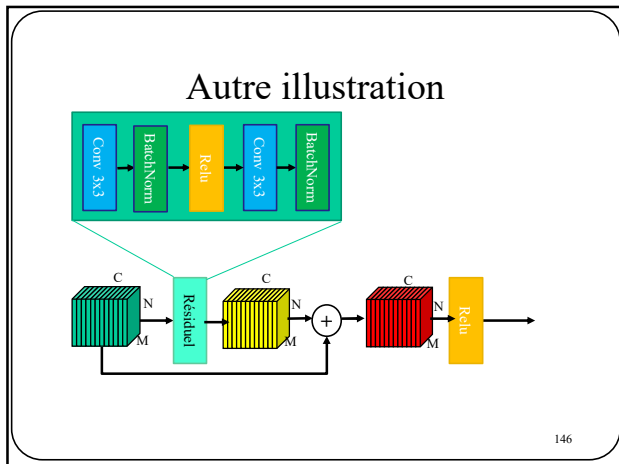
Bloc résiduel



144



145



146

Exemple de code simple

```
def forward(self, x):
    identity = x.clone()

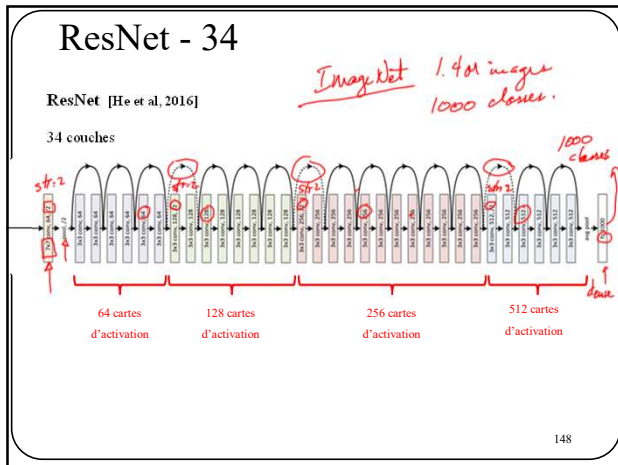
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.conv2(x)
    x = self.bn2(x)

    x += identity
    x = self.relu(x)

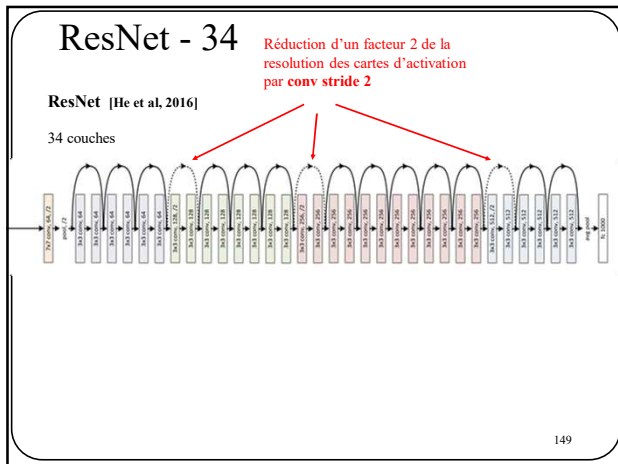
    return x
```

147

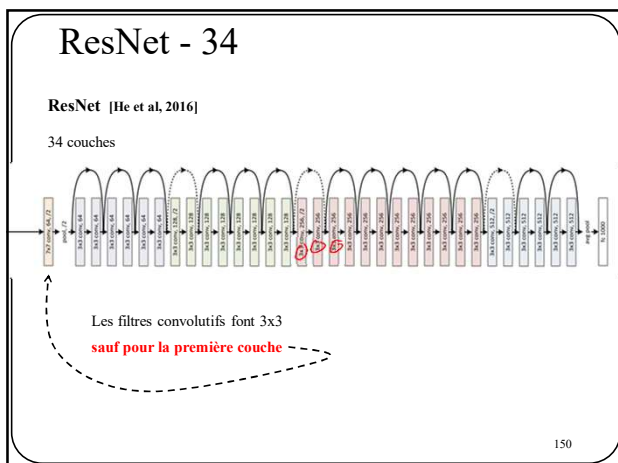
147



148



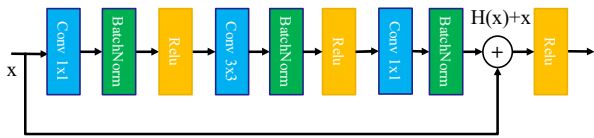
149



150

ResNet - 34

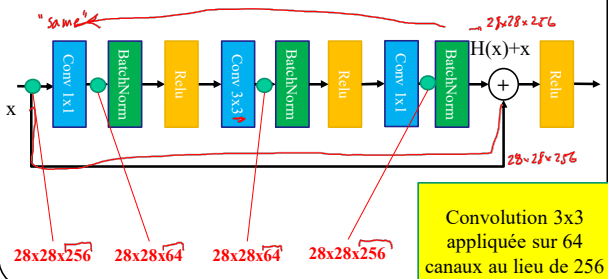
Pour les versions à 50, 101 et 152 couches, on utilise des « **bottleneck layers** »



151

ResNet – 50,101, 152

Pour les versions à plus de 50, 101 et 152 couches, utiliser des « **bottleneck layers** »

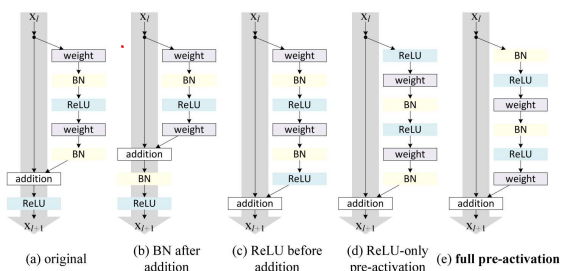


152

ResNet

"model zoo"

Autres types de connexions résiduelles

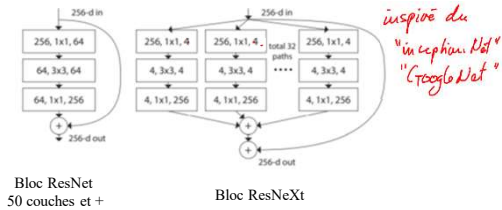


<https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>

153

ResNeXt [Xie et al. 2017]

De la part des mêmes auteurs, une « version améliorée »



S. Xie, R. Girshick, P. Dollar, Z. Tu, K. He
Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

154

ResNeXt

stage	output	ResNet-50	ResNeXt-50 (32×4d)
conv1	112×112	7×7, 64, stride 2	7×7, 64, stride 2
		3×3 max pool, stride 2	
conv2	56×56	1×1, 64 3×3, 64 1×1, 256	1×1, 128 3×3, 128, C=32 1×1, 256
		×3	×3
conv3	28×28	1×1, 128 3×3, 128 1×1, 512	1×1, 256 3×3, 256, C=32 1×1, 512
		×4	×4
conv4	14×14	1×1, 256 3×3, 256 1×1, 1024	1×1, 512 3×3, 512, C=32 1×1, 1024
		×6	×6
conv5	7×7	1×1, 512 3×3, 512 1×1, 2048	1×1, 1024 3×3, 1024, C=32 1×1, 2048
		×3	×3
	1×1	global average pool 1000-d fc, softmax	global average pool 1000-d fc, softmax
# params.		25.5×10 ⁹	25.0×10 ⁹
FLOPs		4.1×10 ⁹	4.2×10 ⁹

Bottleneck
Residual layers

S. Xie, R. Girshick, P. Dollar, Z. Tu, K. He
Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

155

ResNeXt

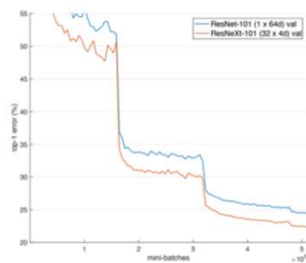


Figure 6. ImageNet-5K experiments. Models are trained on the 5K set and evaluated on the original 1K validation set, plotted as a 1K-way classification task. ResNeXt and its ResNet counterpart have similar complexity.

S. Xie, R. Girshick, P. Dollar, Z. Tu, K. He
Aggregated Residual Transformations for Deep Neural Networks, CVPR 2017

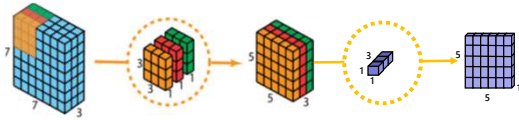
156

MobileNet

[Howard et al.'17]

Objectif : proposer un réseau efficace et sobre en calculs

Rappel *Depth-wise convolution* + conv 1x1



Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 157

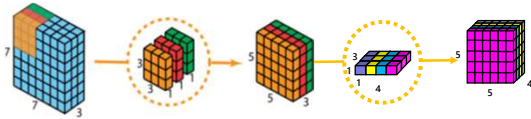
157

MobileNet

[Howard et al.'17]

Objectif : proposer un réseau efficace et sobre en calculs

Rappel *Depth-wise convolution* + N filtres conv 1x1

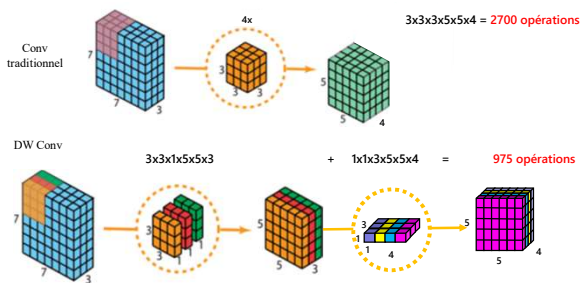


Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 158

158

MobileNet

[Howard et al.'17]



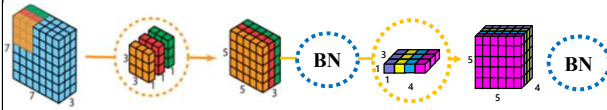
Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 159

159

MobileNet

[Howard et al.'17]

DW Conv de MobileNet inclut deux **batch norm**



Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 160

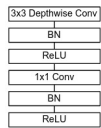
160

MobileNet

[Howard et al.'17]

Tirés de l'article

Conv dw



(autre illustration de la page précédente)

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32 \text{ dw}$	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64 \text{ dw}$	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128 \text{ dw}$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128 \text{ dw}$	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256 \text{ dw}$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256 \text{ dw}$	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
Conv dw / s1	$3 \times 3 \times 512 \text{ dw}$	$14 \times 14 \times 512$
5x Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512 \text{ dw}$	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024 \text{ dw}$	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 161

161

MobileNet

[Howard et al.'17]

Tiré de l'article

Meilleurs résultats
Moins de calculs
Moins de paramètres.

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 162

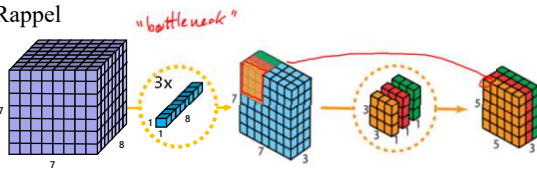
162

[Chollet’17]

Amélioration de ResNet et GoogleNet, similaire à MobileNet. Ici aussi on utilise des

« Depth-wise separable convolutions »

Rappel



F.Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, CVPR 2017 163

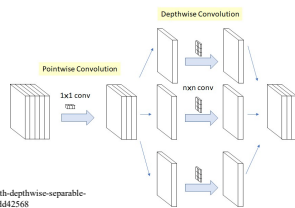
163

[Chollet’17]

Amélioration de ResNet et GoogleNet, similaire à MobileNet. Ici aussi on utilise des

« *Depth-wise separable convolutions* »

Autre illustration



<https://towardsdatascience.com/review-xception-with-depthwise-separable-convolution-better-than-inception-v3-image-dc967dd42568>

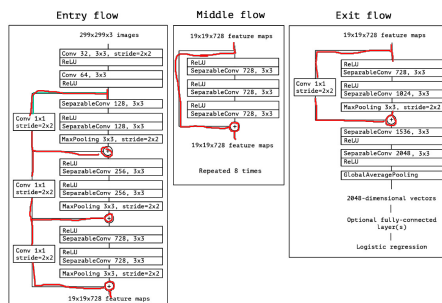
F.Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, CVPR 2017 164

164

XceptionNet

Image tirée de l'article

DWConv
+
Connexions
résiduelles



F.Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, CVPR 2017 165

165

XceptionNet

Image tirée de l'article

Table 1. Classification performance comparison on ImageNet (single crop, single model). VGG-16 and ResNet-152 numbers are only included as a reminder. The version of Inception V3 being benchmarked does not include the auxiliary tower.

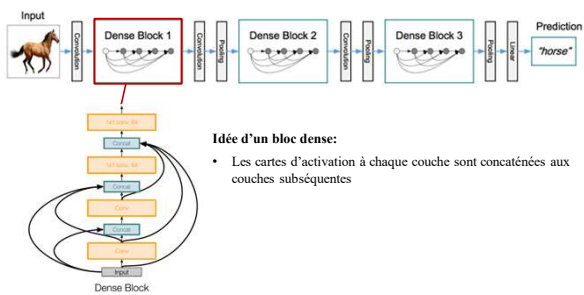
	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	0.790	0.945

F.Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, CVPR 2017 166

166

DenseNet

[Huang et al, 2017]



Idee d'un bloc dense:

- Les cartes d'activation à chaque couche sont concaténées aux couches subséquentes

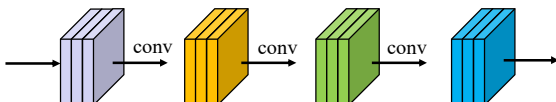
Image: <http://cs231n.stanford.edu/slides/2018>

Huang et al. "Densely connected convolutional networks" CVPR, 2017.

169

169

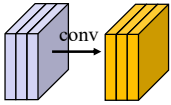
Bloc de 4 couches convolutives ordinaires avec 3 filtres chaque (fonctions d'activation omises pour alléger la figure)



170

170

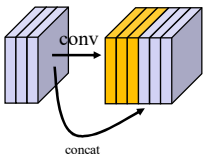
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



171

171

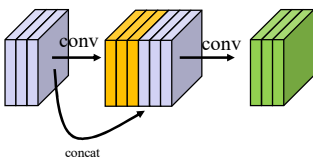
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



172

172

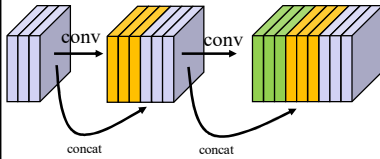
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



173

173

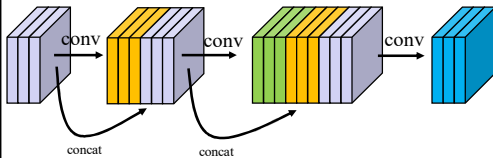
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



174

174

Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



175

175

Bloc dense

Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)

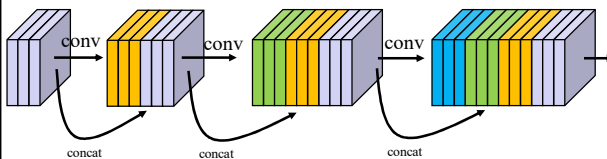


Image: <http://cs231n.stanford.edu/slides/2018>

176

176

Bloc dense

Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)

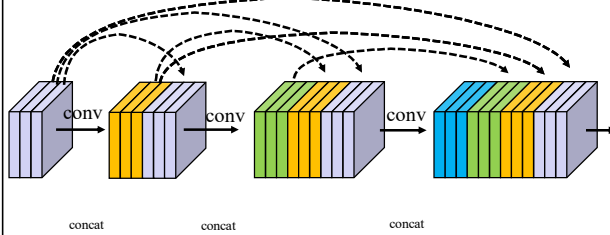
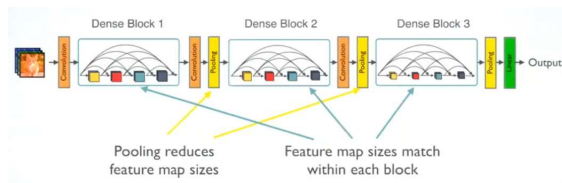


Image: <http://cs231n.stanford.edu/slides/2018>

177

177

DenseNet



<https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803>

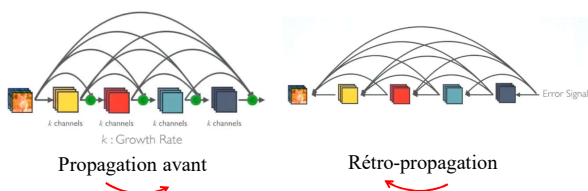
178

178

DenseNet

Avantages:

1. Les gradients circulent directement dans chaque couche lors de la rétro-propagation (moins de *vanishing gradient*).



<https://towardsdatascience.com/review-densenet-image-classification-b6631a8ef803>

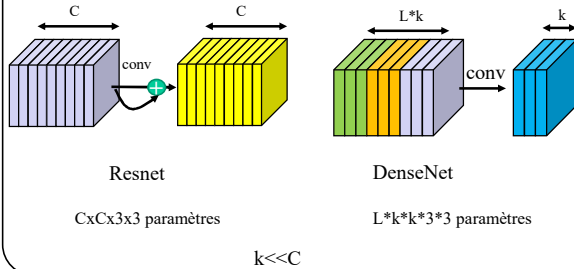
179

179

DenseNet

Avantages:

2. Peu de paramètres

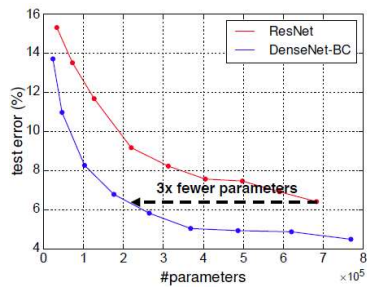


180

DenseNet

Avantages:

2. Peu de paramètres



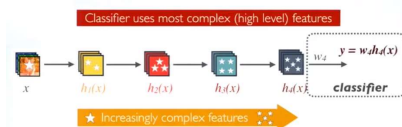
181

DenseNet

Avantages:

3. Le classificateur utilise des caractéristiques de bas et de haut niveau

Standard Connectivity:



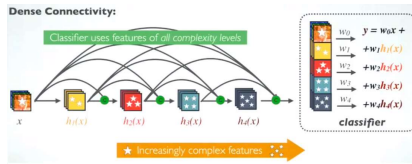
Avec un CNN conventionnel, le classificateur base sa prédiction sur les caractéristiques de la dernière couche, c-à-d des **caractéristiques de haut niveau**

182

DenseNet

Avantages:

- Le classificateur utilise des caractéristiques de bas et de haut niveau



Avec DenseNet, le classifieur utilise des **caractéristiques de bas et de haut niveau**

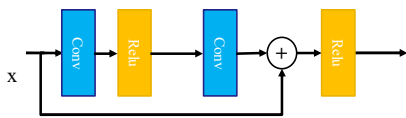
183

Squeeze-and-Excite Net (SENet)

[Hu et al. 2018]

Idée : améliorer les **blocs résiduels** en permettant au réseau de mettre **plus d'importance** sur les **cartes d'activation** les plus informatives.

Bloc résiduel



J. Hu, L. Shen, G. Sun Squeeze-and-Excitation Networks, CVPR 2018

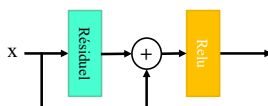
184

Squeeze-and-Excite Net (SENet)

[Hu et al. 2018]

Idée : améliorer les **blocs résiduels** en permettant au réseau de mettre **plus d'importance** sur les **cartes d'activation** les plus informatives.

Bloc résiduel



Par simplicité on appelle cette partie le « résiduel »

J. Hu, L. Shen, G. Sun Squeeze-and-Excitation Networks, CVPR 2018

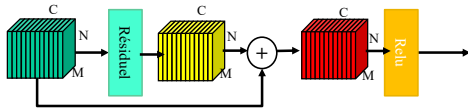
185

Squeeze-and-Excite Net (SENet)

[Hu et al. 2018]

Idee : améliorer les **blocs résiduels** en permettant au réseau de mettre **plus d'importance** sur les **cartes d'activation** les plus informatives.

Bloc résiduel
(avec illustration des cartes d'activation)



Avant et après le résiduel : cartes d'activation de taille $N \times M \times C$

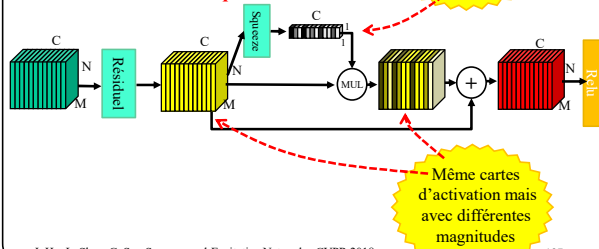
Après l'addition : cartes d'activation de taille $N \times M \times C$

186

186

Squeeze-and-Excite Net (SENet)

Bloc Squeeze-and-Excite



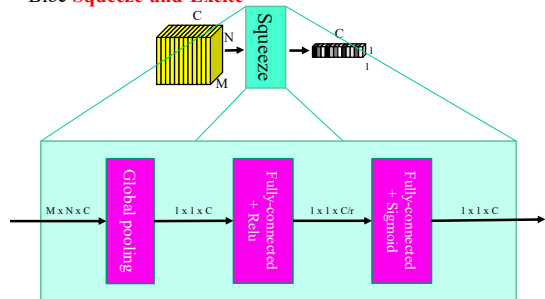
J. Hu, L. Shen, G. Sun Squeeze-and-Excitation Networks, CVPR 2018

187

187

Squeeze-and-Excite Net (SENet)

Bloc Squeeze-and-Excite



r: « squeeze rate » Sigmoid force les valeurs du vecteur de sortie entre 0-1

188

Global pooling

Rappel

Max ou Mean pooling « valid » avec un filtre de la taille des canaux

Résultat : un vecteur de la taille du nombre de canaux

189

Squeeze-and-Excite Net (SENet)

[Hu et al. 2018]

Autre illustration

J. Hu, L. Shen, G. Sun Squeeze-and-Excitation Networks, CVPR 2018

190

190

Squeeze-and-Excite Net (SENet)

[Hu et al. 2018]

Autre illustration

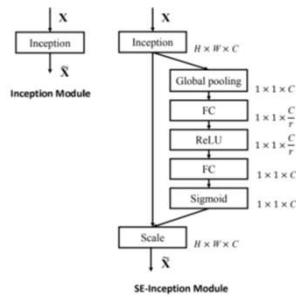
J. Hu, L. Shen, G. Sun Squeeze-and-Excitation Networks, CVPR 2018

191

191

Squeeze-and-Excite Net (SENet)

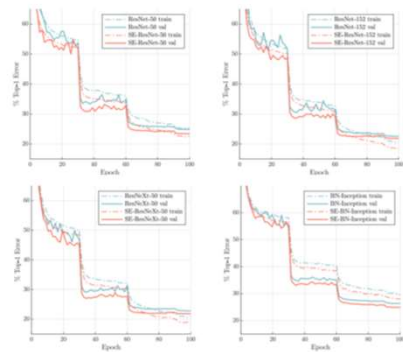
Fonctionne également avec les **modules d'inception**.



192

192

Meilleur que ResNet, ResNeXt et InceptionNet



193

193

Inclure les bonnes pratiques permet d'améliorer la performance des réseaux, même les plus vieux (ResNet – 2015)

Irwan Bello, William Fedus, Xianzhi Du, Ekin Dogus Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, Barret Zoph
Revisiting ResNets: Improved Training and Scaling Strategies, NeurIPS 2021
<https://arxiv.org/pdf/2103.07579.pdf>

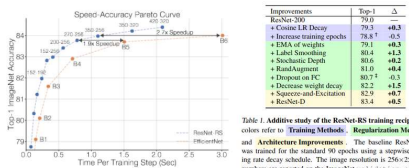


Figure 4: Speed-Accuracy Pareto curve comparing ResNets to EfficientNets. Properly scaled ResNets (ResNet-RS) are 1.7x - 2.7x faster than the popular EfficientNets when closely

Improvements	Top-1	Δ
ResNet-200	79.0	—
+ Constant LR Decay	79.5	+0.5
+ Increase training epochs	79.8	+0.3
+ EMA of weights	80.4	+0.6
+ Label Smoothing	80.6	+0.2
+ Stochastic Depth	81.0	+0.4
+ RandAugment	82.2	+1.2
+ Dropout on FC	82.5	+0.3
+ Decrease weight decay	82.5	+0.3
+ Squeeze-and-Excitation	83.4	+0.9
+ ResNet-D	83.4	+0.0

Table 1: Additive study of the ResNet-RS training recipe. The colors refer to Training Methods, Regularization Methods and Architecture Improvements. The baseline ResNet-200 was trained for the standard 90 epochs using a stepwise learning rate decay schedule. The image resolution is 224x224. All numbers are reported on the ImageNet validation set and averaged over 2 runs. Increasing training duration to 50 epochs only becomes useful once the regularization methods are used, otherwise the accuracy drops due to over-fitting. * dropout bars as we have not yet decreased the weight decay (See Table 2 for more details).

194

194

Improvements	Top-1	Δ
ResNet-200	79.0	—
+ Cosine LR Decay	79.3	+0.3
+ Increase training epochs	78.8 [†]	-0.5
+ EMA of weights	79.1	+0.3
+ Label Smoothing	80.4	+1.3
+ Stochastic Depth	80.6	+0.2
+ RandAugment	81.0	+0.4
+ Dropout on FC	80.7 [‡]	-0.3
+ Decrease weight decay	82.2	+1.5
+ Squeeze-and-Excitation	82.9	+0.7
+ ResNet-D	83.4	+0.5

Table 1. Additive study of the ResNet-RS training recipe. The colors refer to **Training Methods**, **Regularization Methods** and **Architecture Improvements**. The baseline ResNet-200 was trained for the standard 90 epochs using a stepwise learning rate decay schedule. The image resolution is 256×256. All numbers are reported on the ImageNet validation-set and averaged over 2 runs. [†] Increasing training duration to 350 epochs only becomes useful once the regularization methods are used, otherwise the accuracy drops due to over-fitting. [‡] dropout hurts as we have not yet decreased the weight decay (See Table 2 for more details).

Méthodes d'entraînement

Méthodes de régularisation

Améliorations de l'architecture de base

195

195

ConvNeXt : autre exemple d'un ResNet amélioré en fonction des bonnes pratiques

Z. Liu et al. **A ConvNet for the 2020s**, in proc of CVPR 2022

196

196

ConvNeXt : autre exemple d'un ResNet amélioré en fonction des bonnes pratiques

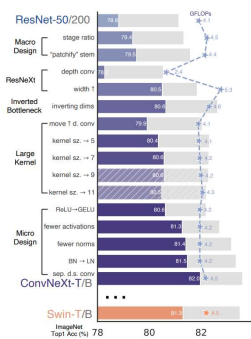
	ResNet-50	ConvNeXt-T
stem	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4
res2	1×1, 64 3×3, 64 × 3 1×1, 256	[d7×7, 96] 1×1, 384 × 3 1×1, 96
res3	1×1, 128 3×3, 128 × 4 1×1, 512	[d7×7, 192] 1×1, 768 × 3 1×1, 192
res4	1×1, 256 3×3, 256 × 6 1×1, 1024	[d7×7, 384] 1×1, 1536 × 9 1×1, 384
res5	1×1, 512 3×3, 512 × 3 1×1, 2048	[d7×7, 768] 1×1, 3072 × 3 1×1, 768
FLOPs	4.1 × 10 ⁹	4.5 × 10 ⁹
# params.	25.6 × 10 ⁶	28.6 × 10 ⁶

197

197

ConvNeXt : autre exemple d'un ResNet amélioré en fonction des bonnes pratiques

Autres améliorations



198