Réseaux de neurones

IFT 780

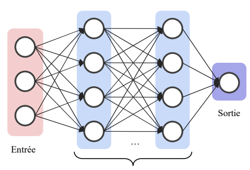
Réseaux à convolution avancés et architectures convolutives modernes

Par
Pierre-Marc Jodoin

1

Réseaux de neurones à convolution

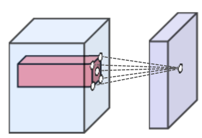
Réseaux de neurones multicouches



Couches cachées

Beaucoup de paramètres
Moins de neurones

CNN



- Paramètres partagés
- Connectivité locale

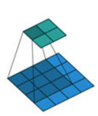
Moins de paramètres
Beaucoup de neurones
Architectures plus profondes

2

2

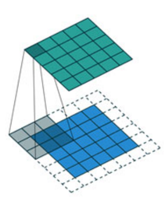
Réseaux de neurones à convolution

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

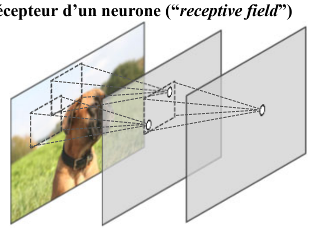
Images: http://deeplearning.net/software/theano/tutorial

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**.

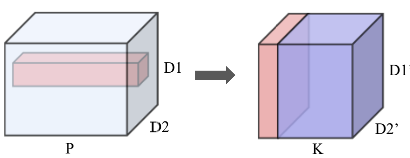
4

4

Réseaux de neurones à convolution

Taille et convolution

K filtres F1xF2



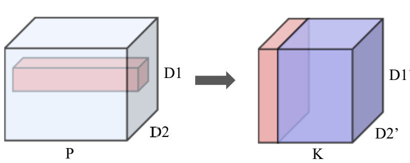
- La **profondeur** du filter = le **nombre de canaux** en entrée (ici P)
- Le **nombre de cartes d'activation** = le **nombre de filtres** (ici K)
- Filtre convolutif 1D est un tenseur 2D (**F * P**)
- Filtre convolutif 2D est un tenseur 3D (**F1 * F2 * P**)
- K filtres convolutifs 2D est un tenseur 4D (**K * F1 * F2 * P**)

5

Réseaux de neurones à convolution

Taille et convolution

K filtres F1xF2



Taille du volume d'activation (2D) :

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

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

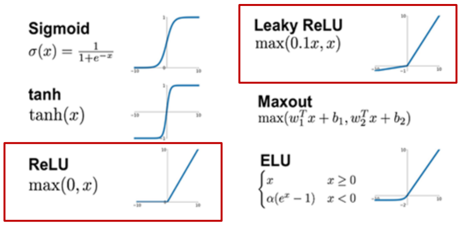
K = nb_filtres

6

Réseaux de neurones à convolution

Chaque couche convolutive contient une function d'activation

S'assure que tous les neurones sont activés



Réduit le problème de la disparition du gradient

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*)

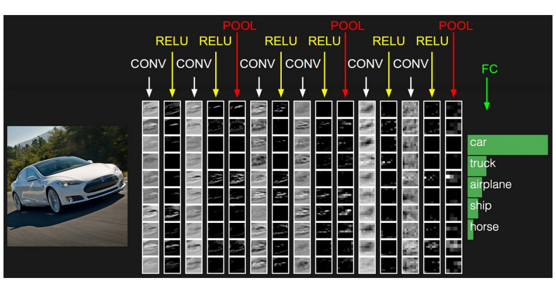


Image: http://cs231n.github.io

9

9

Autres types de couches convolutives

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

10

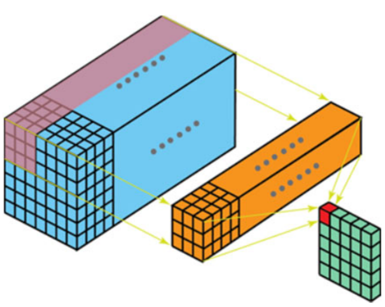
Convolution **séparée**

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**



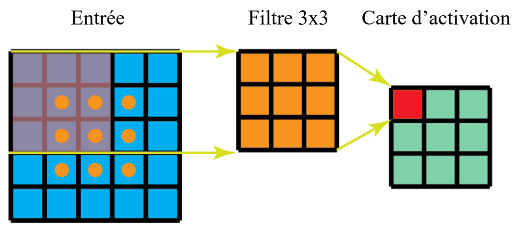
12

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

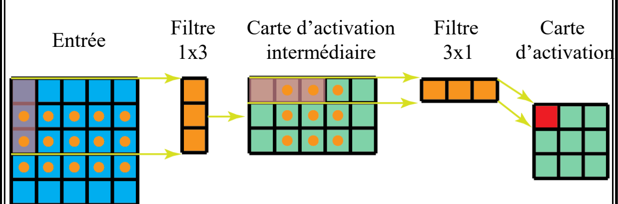
Les filtres 2D et 3D peuvent être **séparés** en filtres plus élémentaires

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



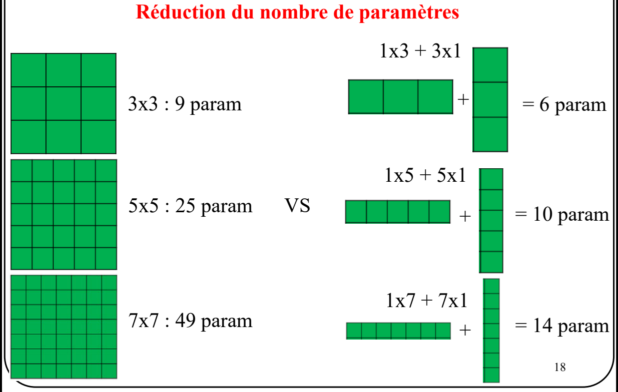
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

17

17

Convolution séparée

Réduction du nombre de paramètres



18

18

Convolution séparée en profondeur

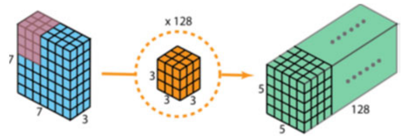
(Depthwise Separable Convolution)

19

19

Convolution usuelle

3 cartes d'activation de taille **7x7** — **128** filtres **3x3x3** — **128** cartes d'activation de taille **5x5**



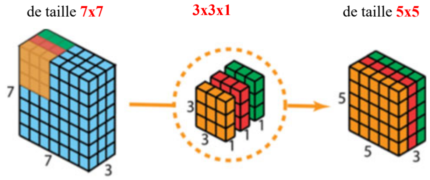
3x3x3 x128 = **3456 params**

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

20

depth-wise convolution

3 cartes d'activation de taille **7x7** — **3** filtres **3x3x1** — **3** cartes d'activation de taille **5x5**



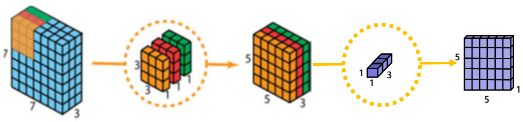
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

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

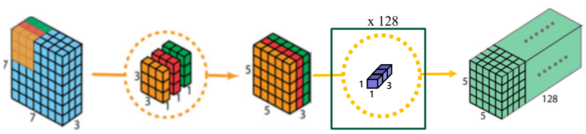
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

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**



3x3x3+3x128 = **411 params**

23

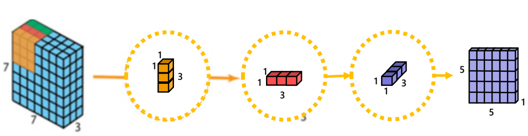
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

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**



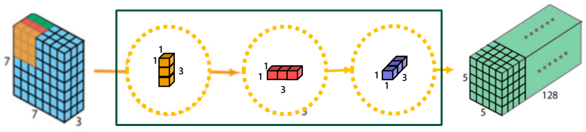
24

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

24

Depthwise Separable Convolution

Depth-wise convolution + conv 1x1



(3x1x1 + 1x3x1 + 1x1x3)x128 = **1158 params**

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

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

25

25

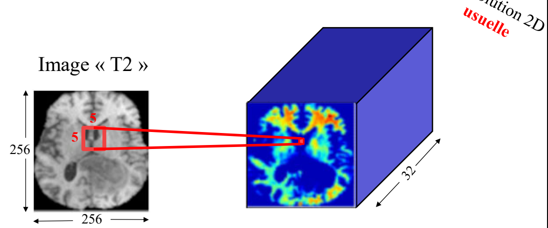
Convolution 3D

26

26

Convolution 3D

Très utile en imagerie médicale

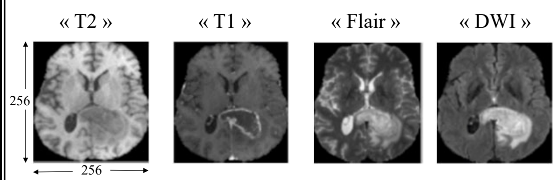


32 filtres de taille 5x5x1

27

Convolution 3D

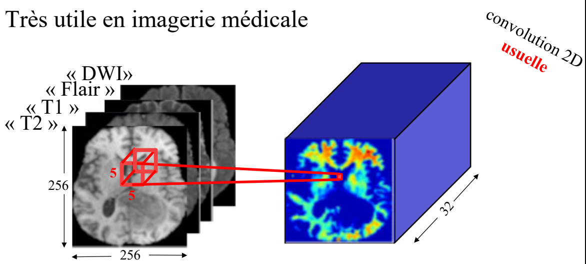
Souvent en imagerie médicale on utilise plusieurs « modalités »



28

Convolution 3D

Très utile en imagerie médicale

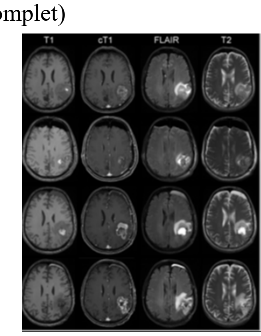


32 filtres de taille 5x5x4

29

Convolution 3D

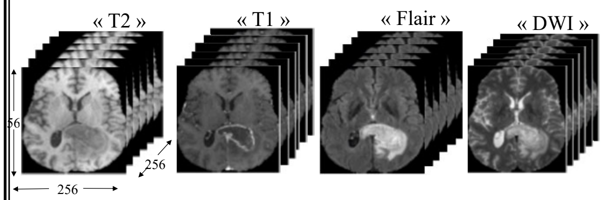
Souvent en imagerie médicale on utilise plusieurs « modalités » **et des acquisitions en 3D** (ici un cerveau au complet)



30

Convolution 3D

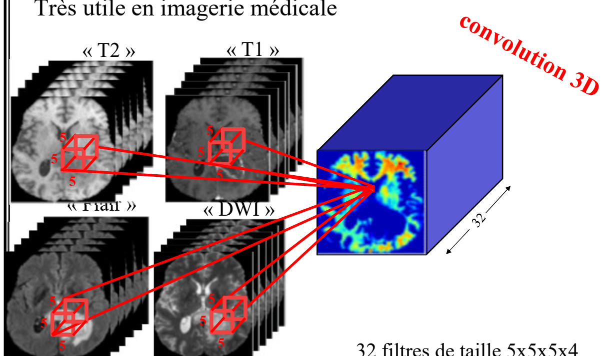
Souvent en imagerie médicale on utilise plusieurs « modalités » **et des acquisitions en 3D**



31

Convolution 3D

Très utile en imagerie médicale



32 filtres de taille 5x5x5x4

32

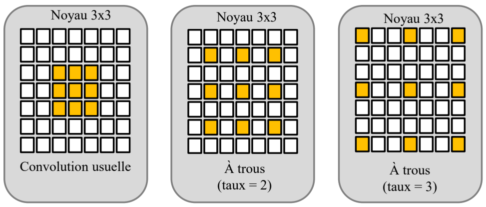
Convolution *à trous*

33

33

Convolution *à trous*

*(*ou dilatée*)*

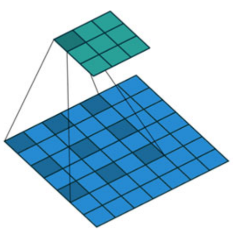


Champ récepteur=3x3 **Champ récepteur=5x5** **Champ récepteur=7x7**

34

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

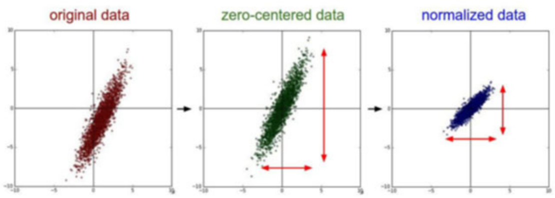
35

35

Autres pratiques courantes

36

Prétraitement des données en entrée



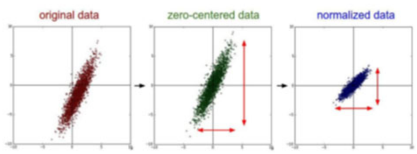
`X -= np.mean(X, axis = 0)` `X /= np.std(X, axis = 0)`

(réduction du biais) (S'assurer que chaque dimension a le même poids)

Image: http://cs231n.github.io

37

Prétraitement des données en entrée

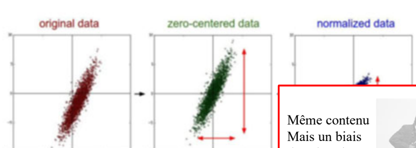


S'assurer que les dimensions ont les mêmes variances.

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

38

Prétraitement des données en entrée



Même contenu Mais un biais dans les niveaux de gris.

Après recentrage des niveaux de gris.

Ex.: utile lorsqu'on traite des images avec de forts décallage dans l'intensité des pixels.

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)
  - Soustraite 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)
  - Soustraite 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)
  - Soustraite **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

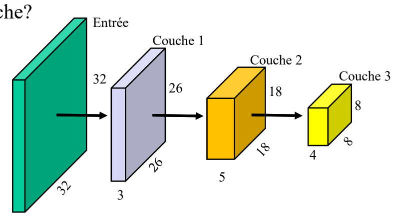
41

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

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

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



42

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

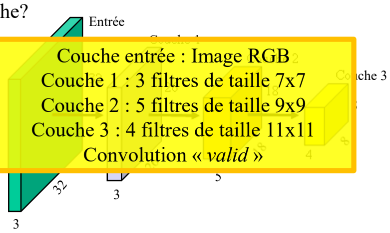
42

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

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

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



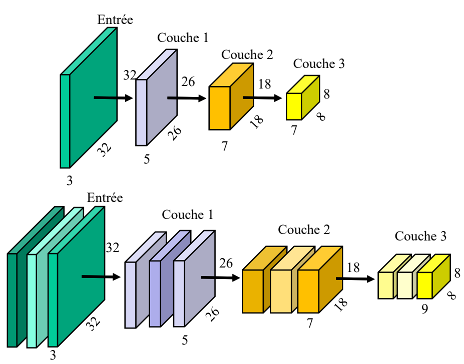
Couche entrée : Image RGB
Couche 1 : 3 filtres de taille 7x7
Couche 2 : 5 filtres de taille 9x9
Couche 3 : 4 filtres de taille 11x11
Convolution « *valid* »

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

43

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

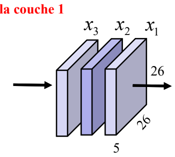


44

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

Ex. : normalisation de la couche 1



x_i : les cartes d'activations du ième lot

Contient 26x26x5= 3380 neurones

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

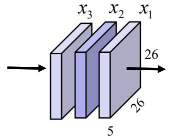
$$x_{VAR} = \frac{1}{3}\sum_{i=1}^{3} (x_i - x_{MOY})^2$$

45

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

Ex. : normalisation de la couche 1



x_i : les cartes d'activations du ième lot

Contient 26x26x5= 3380 neurones

$$x_{MOY} = \frac{1}{3}\sum_{i=1}^{3} x_i$$ → **Taille 26 x 26 x 5**

$$x_{VAR} = \frac{1}{3}\sum_{i=1}^{3}(x_i - x_{MOY})^2$$ → **Taille 26 x 26 x 5**

$$\hat{x}_i = \frac{x_i - x_{MOY}}{\sqrt{x_{VAR} + \varepsilon}}$$ → **Taille 26 x 26 x 5**

46

[Ioffe and Szegedy, 2015]

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!

47

47

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

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

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

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

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

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

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.

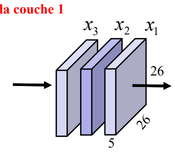
48

48

Normalisation par lot (*Batch norm*)

[Ioffe and Szegedy, 2015]

Ex. : normalisation de la couche 1



x_i : les cartes d'activations du ième lot

Contient 26x26x5= 3380 neurones

$$x_{MOY} = \frac{1}{3}\sum_{i=1}^{3} x_i$$ → **Taille 26 x 26 x 5**

$$x_{VAR} = \frac{1}{3}\sum_{i=1}^{3} (x_i - x_{MOY})^2$$ → **Taille 26 x 26 x 5**

$$\hat{x}_i = \frac{x_i - x_{MOY}}{\sqrt{x_{VAR} + \varepsilon}}$$ → **Taille 26 x 26 x 5**

$$\tilde{x}_i = \gamma \circ \hat{x}_i + \beta$$ → **γ et β de taille 26 x 26 x 5**

49

Normalisation par lot (*Batch norm*)

NOTE: produit de Hadamar

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

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

50

50

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

```
def batchnorm_forward_pass(x, gamma, beta, 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*gamma + beta

return x_norm
```

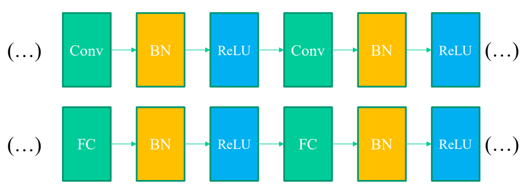
51

51

[Ioffe and Szegedy, 2015]

Normalisation par lot (*Batch norm*)

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



52

52

Normalisation par lot (*Batch norm*)

Avantages:

Accélère l'entraînement

Permet d'utiliser un plus gros *Learning rate*

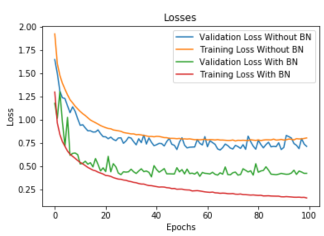


Image: Ioffe et al. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv, 2015.

53

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

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

c'est-à-dire

Nb_training_data

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

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

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

54

Normalisation par lot (*Batch norm*)

Pour plus d'information:

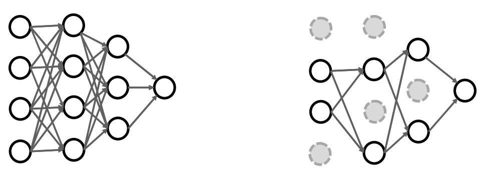
- https://medium.com/@ilango100/batch-normalization-speed-up-neural-network-training-245e39a62f85
- https://deepnotes.io/batchnorm
- Image: Ioffe et al. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv, 2015.

55

55

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

56

Autre bonne pratique : *Dropout*

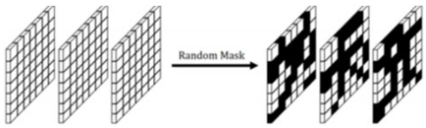
```python
p = 0.5 # probability of keeping a unit active. higher = less dropout

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/

57

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.

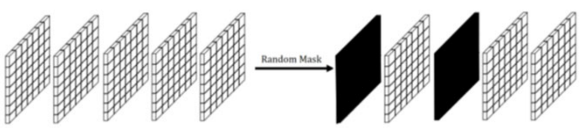


Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, Christoph Bregler
"Efficient Object Localization Using Convolutional Networks", in proc of CVPR 2015

58

Spatial dropout, une variante du *dropout*

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

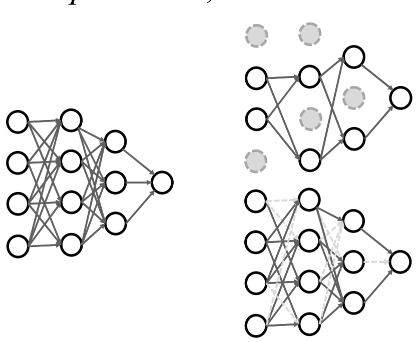


Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, Christoph Bregler
"Efficient Object Localization Using Convolutional Networks", in proc of CVPR 2015

59

59

Drop connect, autre variante de *dropout*



Dropout :
forcer des neurones à zéro

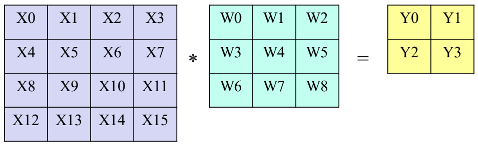
Dropconnect :
forcer des **poids** à zéro

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

60

Drop connect, autre variante de *dropout*

Dans le cas d'une convolution



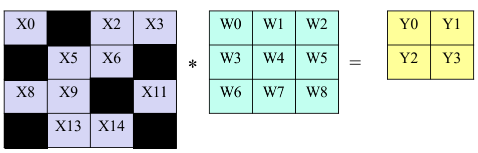
Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus Regularization of Neural Networks using DropConnect, ICML 2013

61

Drop connect, autre variante de *dropout*

Dans le cas d'une convolution

Dropout force les **neurones** à zéro



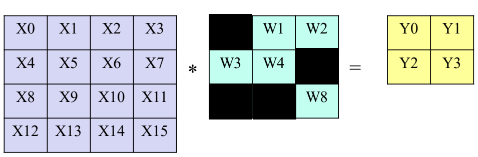
Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, Rob Fergus Regularization of Neural Networks using DropConnect, ICML 2013

62

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

63

Ensemble de modèles

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

Permet d'améliorer les performances de 2-3%

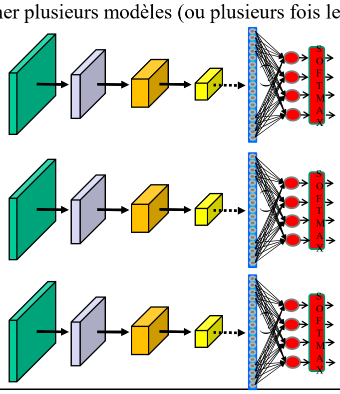
NOTE Même entraîner N-fois le même modèle fonctionne!

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

64

Ensemble de modèles

Entraîner plusieurs modèles (ou plusieurs fois le même modèle)

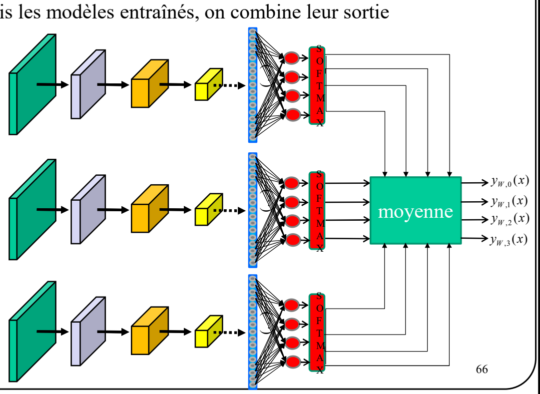


65

65

Ensemble de modèles

Une fois les modèles entraînés, on combine leur sortie

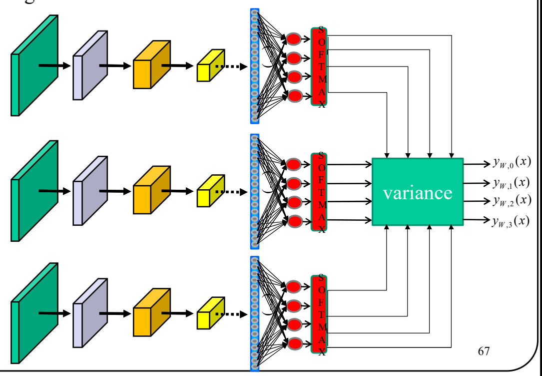


66

66

Ensemble de modèles

On peut également calculer leur **incertitude** avec une variance

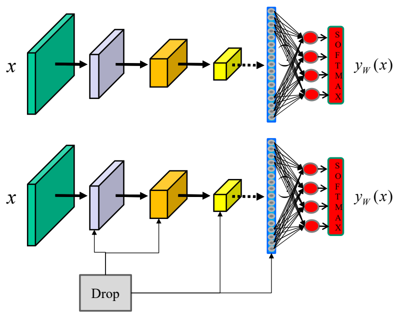


67

67

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*.



Sans *dropout*, la sortie $y_W(x)$ est toujours la même pour un x donné

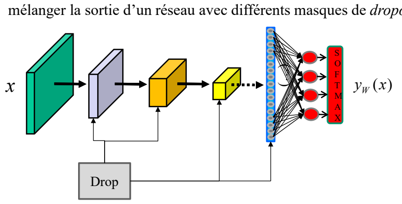
Avec *dropout*, la sortie $y_W(x)$ n'est PAS toujours la même pour un x donné

68

68

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 avants* pour un x donné, on aura M résultats différents.

$$\hat{y}_W(x) = \frac{1}{M}\sum_{m=1}^{M} y_W^m(x) \longrightarrow$$ **Prédictions moyennées** (ensemble de modèles)

$$\hat{y}_W^{VAR}(x) = \frac{1}{M}\sum_{m=1}^{M}\left(y_W^m(x) - \hat{y}_W(x)\right)^2 \longrightarrow$$ **Incertitude**

69

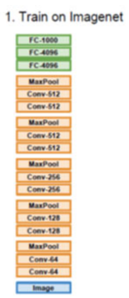
Transfert d'apprentissage
(Transfer learning)

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.

70

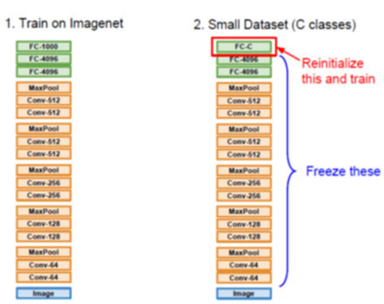
Transfert d'apprentissage
(Transfer learning)



Crédit : http://cs231n.stanford.edu

71

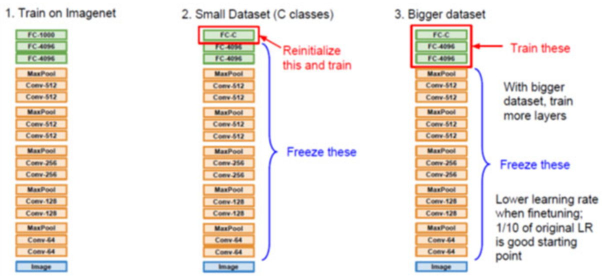
Transfert d'apprentissage
(Transfer learning)



Crédit : http://cs231n.stanford.edu

72

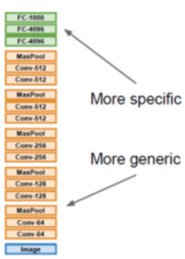
Transfert d'apprentissage
(Transfer learning)



Crédit : http://cs231n.stanford.edu

73

Transfert d'apprentissage
(Transfer learning)



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Crédit : http://cs231n.stanford.edu

74

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)

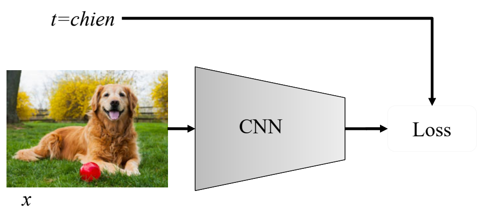
Plusieurs bibliothèques ont un "Model Zoo" avec des modèles pré-entraînés

TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision

75

Augmentation de données
(Data augmentation)

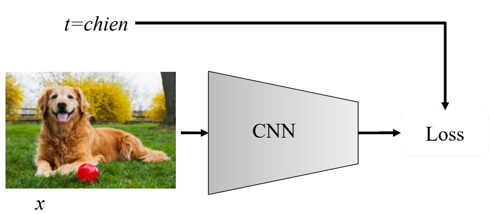
On peut artificiellement augmenter le nombre de données en transformant les données sans pour autant affecter leur cible.



76

Augmentation de données
(Data augmentation)

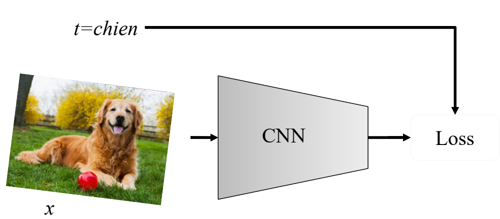
Exemple de transformation : **flip**



77

Augmentation de données
(Data augmentation)

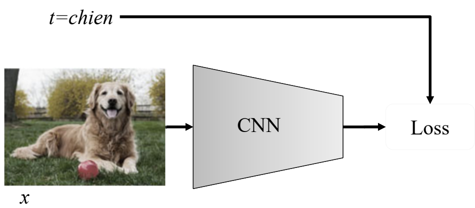
Exemple de transformation : **rotation aléatoire de +/- 5 degrés**



78

Augmentation de données
(Data augmentation)

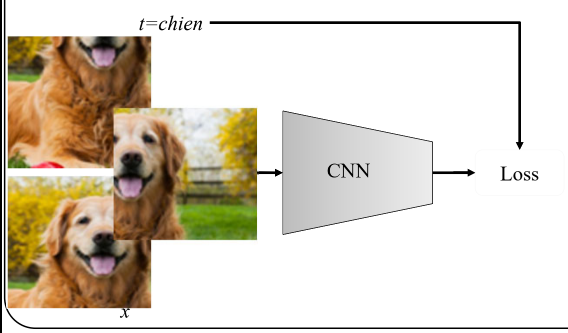
Exemple de transformation : **changement aléatoire de la dynamique des couleurs**



79

Augmentation de données
(Data augmentation)

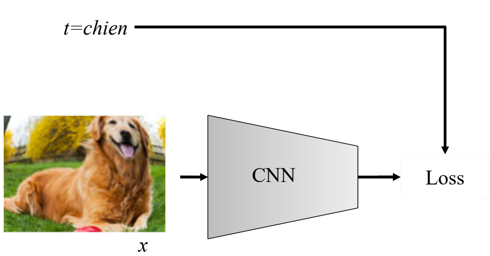
Exemple de transformation : **crop aléatoire + redimension**



80

Augmentation de données
(Data augmentation)

Exemple de transformation : **transformation de lentille**



81

L’augmentation de données n’est pas une exception

c’est la norme

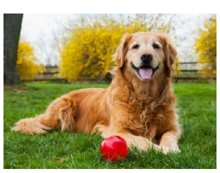
Il n’y a *a priori* **aucune raison pour ne pas l’utiliser dans vos projets.**

82

CLASSIFICATION D’IMAGE

83

Classification d’image



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

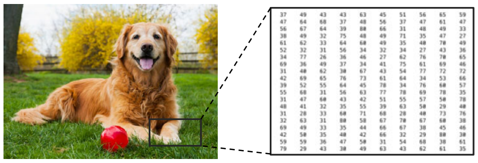
But: prédire une étiquette de classe (ou une distribution de probabilités) pour une image donnée.

84

84

Classification d’image

Défi



Ce que l’humain voit

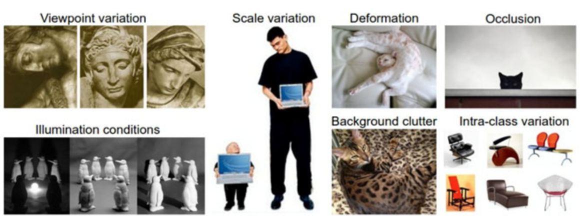
Ce que l’ordinateur voit

85

85

Classification d’image

Autres défis



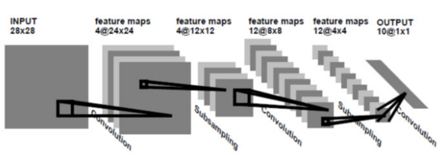
Facile pour les humains mais difficile pour les ordinateurs.

Image: http://cs231n.github.io

86

86

LeNet-1



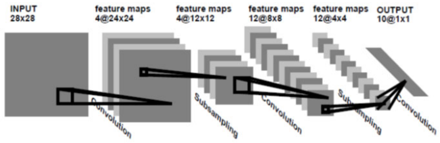
Une des plus vieilles architectures faite pour la reconnaissance de caractères.

- 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

87

LeNet-1



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

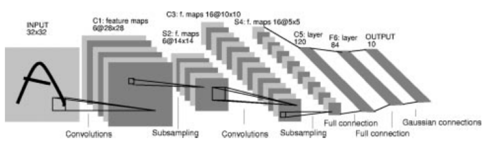
- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles
  (avec 4 et 12 filtres)
- 2 *average pooling*
- 1 couche pleinement connectée
- 10 classes

Input
|
CONV
|
POOL
|
CONV
|
POOL
|
FC
|
SOFTMAX

Combien de neurones?
Combien de paramètres?

88

LeNet-5



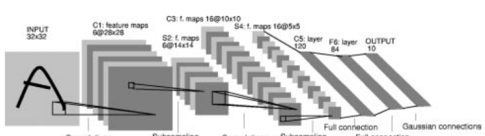
Version améliorée:

- 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

89

LeNet-5



Version améliorée:

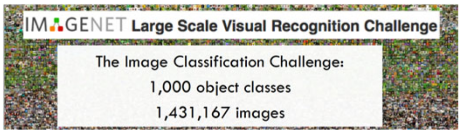
- Image d'entrée : 28x28x1
- Filtres convolutionnels : 5x5
- Conv « valid » + tanh
- 2 couches convolutionnelles
  (avec 6 et 16 filtres)
- 2 *average pooling*
- 3 couches pleinement connectées
  (120, 84 et 10 neurones)
- 10 classes

Input
|
CONV
|
POOL
|
CONV
|
POOL
|
FC
|
FC
|
FC
|
SOFTMAX

Combien de neurones?
Combien de paramètres?

90

Classification d’images



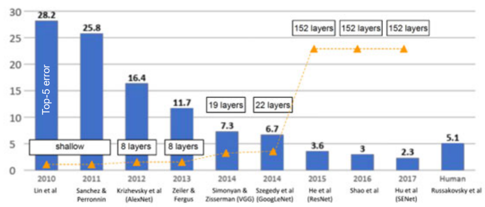


Image: http://cs231n.stanford.edu/slides/2018

91

91

Classification d’images

AlexNet **[Krizhevsky et al, 2012]**

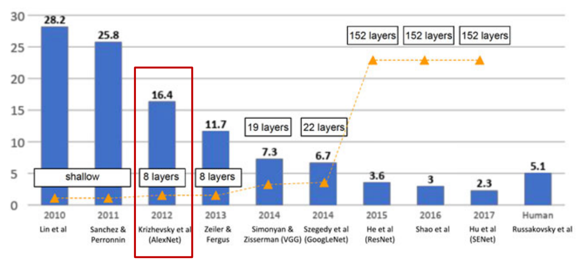
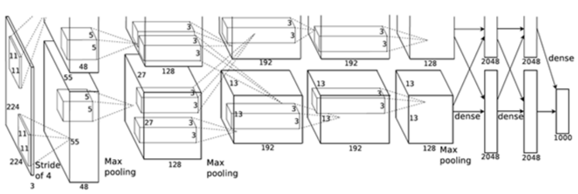


Image: http://cs231n.stanford.edu/slides/2018

92

92

AlexNet

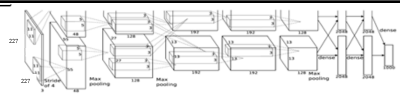


- Premier CNN à bien performer sur ImageNet (amélioration de 10% par rapport aux autres)
- Utilisation de techniques aujourd’hui fréquemment utilisées: **ReLU, data augmentation** and **dropout**
- Utilisation de **GPUs** (2 dans leur cas)
- Point de départ de la **révolution du “deep learning”** en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

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

93

AlexNet



Architecture:

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

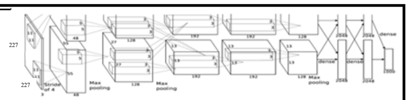
Entrée : image RGB: 227x227x3
Couche 1 (CONV1): 96 filtres de taille 11x11 avec **stride de 4** et conv **"valid"**

Quelle est la taille des cartes d'activation?

Réponse: (227-11)/4+1 = 55

94

AlexNet



Architecture:

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

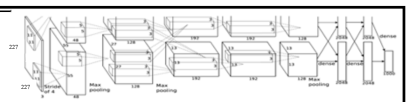
Entrée : image RGB: 227x227x3
Couche 1 (CONV1): 96 filtres de taille 11x11 avec **stride de 4** et conv **"valid"**

Cartes d'activation : 96 x 55 x 55

Q: Quel est le nombre de paramètres?

95

AlexNet



Architecture:

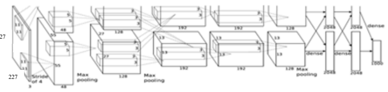
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

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

96

AlexNet



ENTRÉE : 227x227x3
CONV1: 96 x 55 x 55

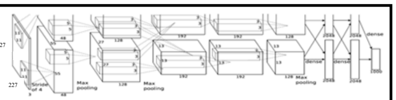
Couche 2 MaxPool : 3x3 stride stride 2

Quelle est la taille des cartes d'activation?

Réponse: (55-3)/2+1 = 27

97

AlexNet



ENTRÉE : 227x227x3
CONV1: 96 x 55 x 55

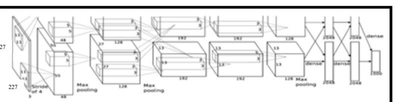
Couche 2 MaxPool : 3x3 stride stride 2
27 x 27 x 96

Combien y a-t-il de paramètres?

Réponse: 0!

98

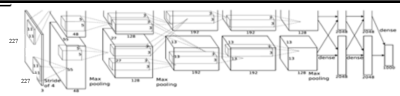
AlexNet



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

99

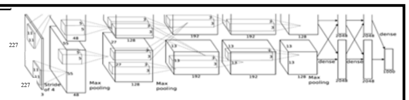
AlexNet



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

100

AlexNet



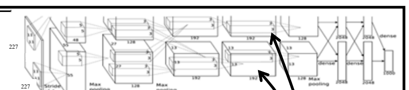
ENTRÉE	227 x 227 x 3
CONV1	55 x 55 x 96
MAX POOL1	27 x 27 x 96
NORM1	27 x 27 x 96
CONV2	27 x 27 x 256
MAX POOL2	13 x 13 x 256
NORM2	13 x 13 x 256
CONV3	13 x 13 x 384
CONV4	13 x 13 x 384
CONV5	13 x 13 x 256
MAX POOL3	6 x 6 x 256
FC6	4096
FC7	4096
FC8	1000

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

101

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

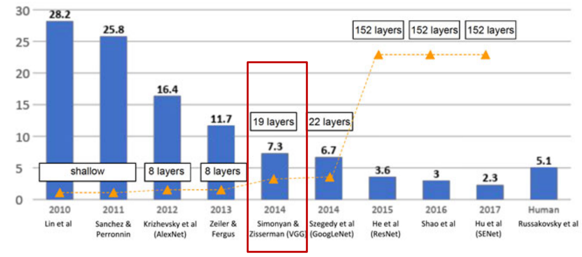
Notes additionnelles:

Utilisation de **2 GPUs**
Ensemble de 7 CNN: 18.2% -> 15.4

102

Classification d'images

VGGNet **[Simonyan and Zisserman, 2014]**



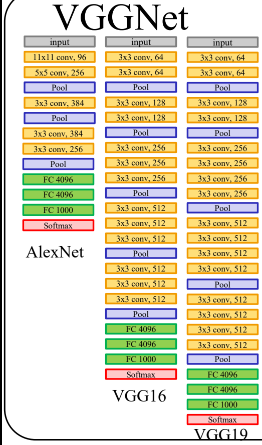
Karen Simonyan, Andrew Zisserman "Very Deep Convolutional Networks for Large-Scale Image Recognition", ICLR 2015

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

103

103

VGGNet



Ce qui caractérise VGGNet par rapport à ses prédécesseurs:

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

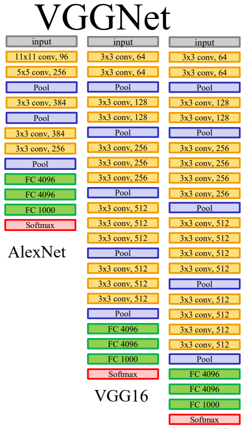
AlexNet : 8 couches
VGGNet : 16 ou 19 couches

7.3% d'erreur contre 11.7% pour ZFNet

104

104

VGGNet



Pourquoi utiliser des filtres 3x3?

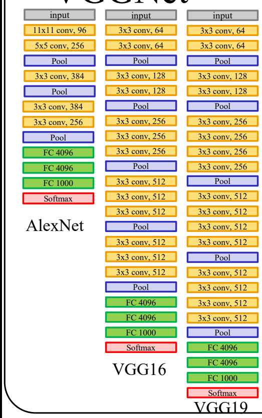
Indice1 : quel est le **champ récepteur** (***receptive field***) d'une série de 3 couches 3x3?

Indice2 : combien de paramètres pour ces 3 couches?

105

105

VGGNet



Pourquoi utiliser des filtres 3x3?

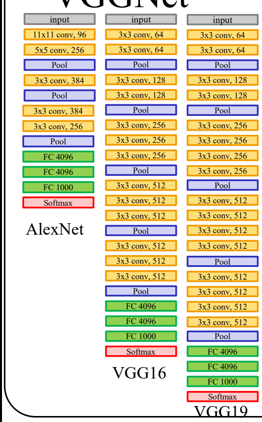
Le champ récepteur de 3 filtres 3x3 consécutifs est **7x7**

nb_params pour 3 filtres consécutifs **3x3x3xNb_Canaux**
Vs
nb_params pour 1 couche: **7x7xNb_Canaux**

106

106

VGGNet



Pourquoi utiliser des filtres 3x3?

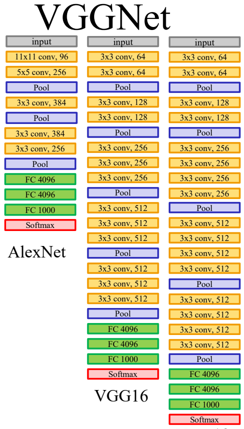
Le champ récepteur de 3 filtres 3x3 est **7x7**

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

107

107

VGGNet



Conclusion :

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

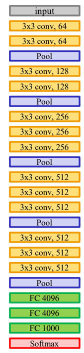
108

108

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

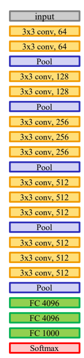
109

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

109

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

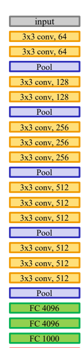
110

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

110

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



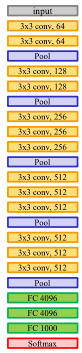
VGG16

111

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

111

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

112

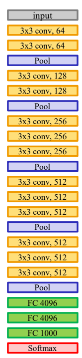
Image: http://cs231n.stanford.edu/slides/2

112

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

113

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

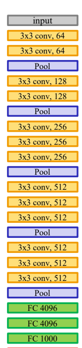
113

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

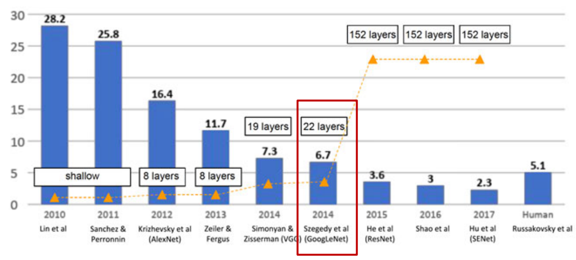
114

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

114

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, Going Deeper with Convolutions, CVPR 2015

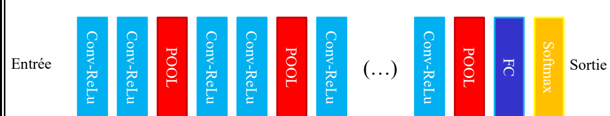
Image: http://cs231n.stanford.edu/slides/2018

115

115

GoogLeNet (Inception v1)

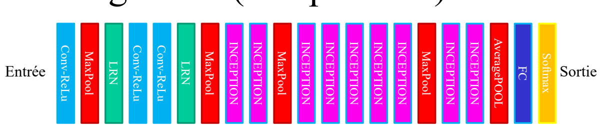
CNN “Classique”



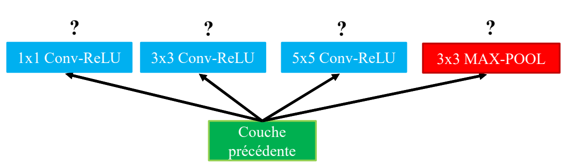
GoogLeNet – 21 couches

116

GoogLeNet (Inception v1)

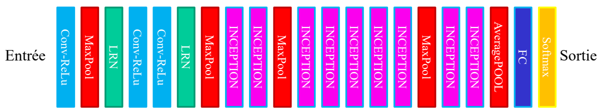


Lorsqu’on design un CNN, il faut déterminer la taille des filtres à chaque couche. 1x1? 3x3? 5x5? 7x7?...



117

GoogLeNet (Inception v1)



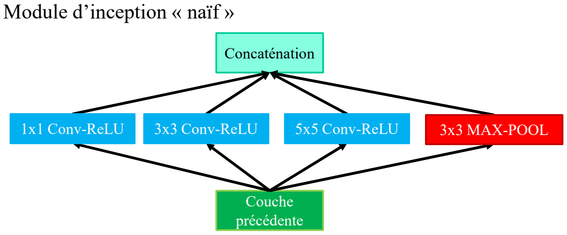
Lorsqu'on design un CNN, il faut déterminer la taille des filtres à chaque couche. 1x1? 3x3? 5x5? 7x7?...

"**Inception module**": combine le résultat de plusieurs filtres.

118

GoogLeNet (Inception v1)

Module d'inception « naïf »

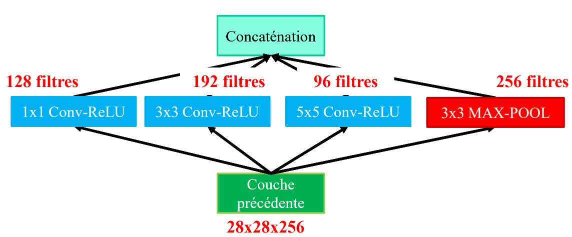


- Appliquer 4 filtres différents et « same »
- Concaténation des cartes d'activation

119

GoogLeNet (Inception v1)

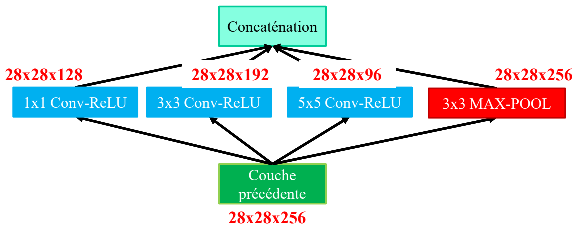
Problème : trop d'opérations



120

GoogLeNet (Inception v1)

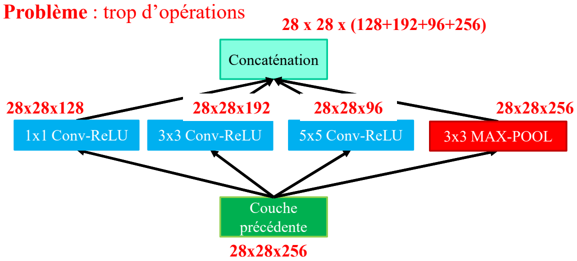
Problème : trop d'opérations



121

GoogLeNet (Inception v1)

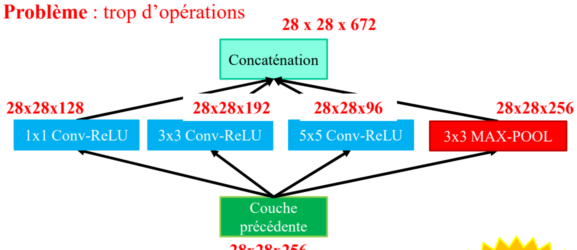
Problème : trop d'opérations



122

GoogLeNet (Inception v1)

Problème : trop d'opérations

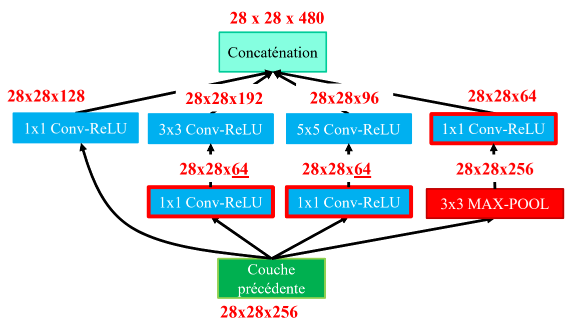


Nombre total d'opérations:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: ~854M opérations

123

GoogLeNet (Inception v1)

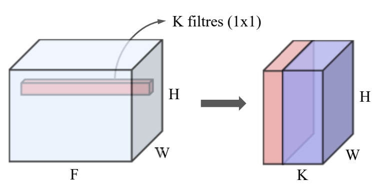
Module d'inception avec « **bottleneck layers** »



124

Classification d'images

RAPPEL : 1x1 convolution



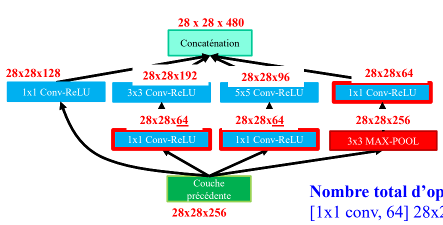
Permet de réduire le nombre de cartes d'activation

125

125

GoogLeNet (Inception v1)

Module d'inception avec « **bottleneck layers** »

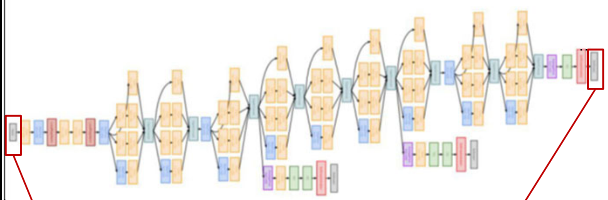


Nombre total d'opérations:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: ~271M opérations

126

GoogLeNet (Inception v1)

Modèle complet



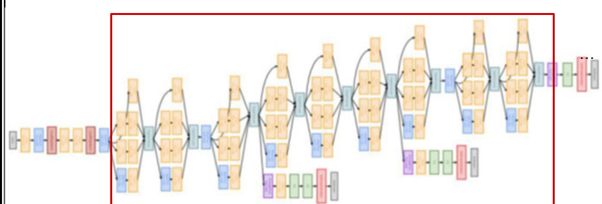
Sortie

Entrée

127

GoogLeNet (Inception v1)

Modèle complet

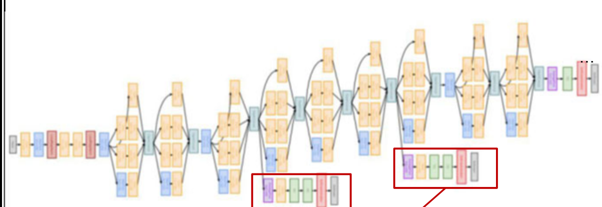


9 modules
d'inception

128

GoogLeNet (Inception v1)

Modèle complet



Injection de gradients
(AvgPool-1x1Conv-FC-FC-Softmax)

129

129

Classification d'images

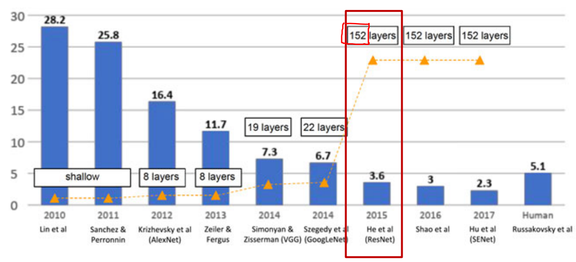
Taille des réseaux

AlexNet	VGG16	GoogLeNet
8 couches	16 couches	22 couches
~68M paramètres	~138M paramètres	**~5M paramètres**

130

Classification d'images

ResNet **[He et al, 2016]**



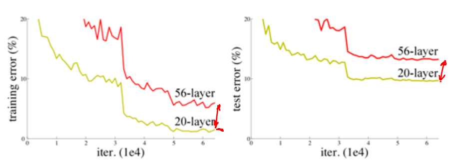
K.He, X.Zhang, S. Ren, J.Sun, Deep Residual Learning for Image Recognition, CVPR 2016

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

131

Classification d'images

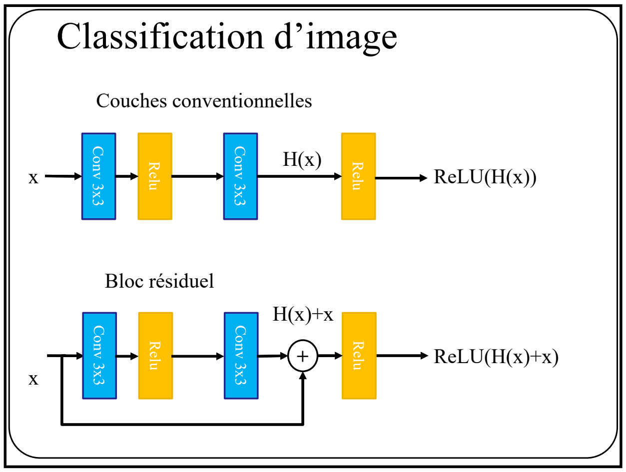
Qu'arrive-t-il lorsqu'on rend très profond un CNN?



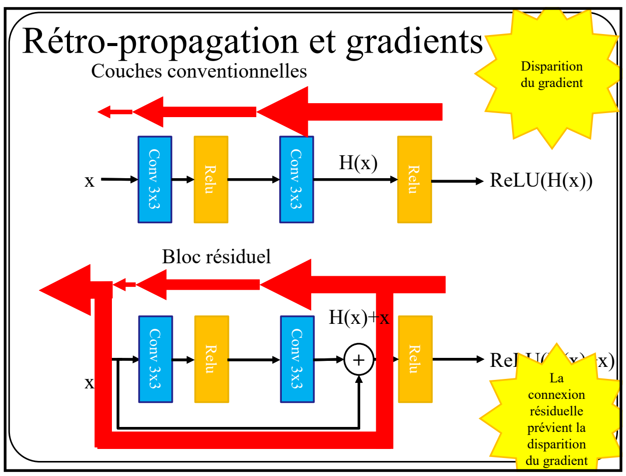
- À la fois **l'erreur** de test et d'entraînement vont **augmenter**.
- HYPOTHÈSE : Ceci n'est pas du sur-apprentissage (*overfitting*) c'est un **problème d'optimisation**.

132

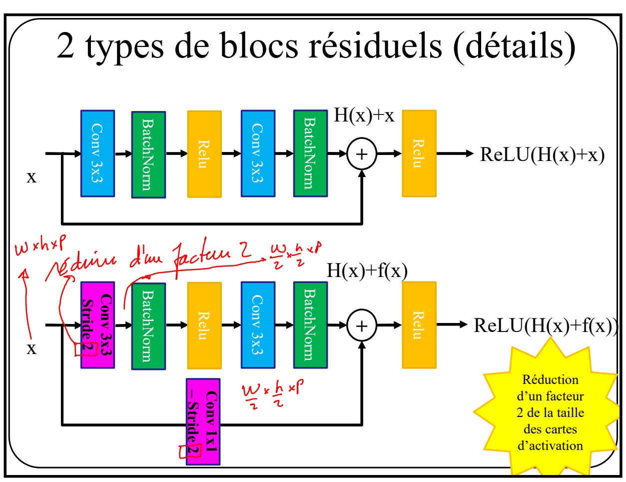
132


Classification d'image
Couches conventionnelles
x
Conv 3x3
Relu
Conv 3x3
H(x)
Relu
ReLU(H(x))
Bloc résiduel
x
Conv 3x3
Relu
Conv 3x3
H(x)+x
+
Relu
ReLU(H(x)+x)

133

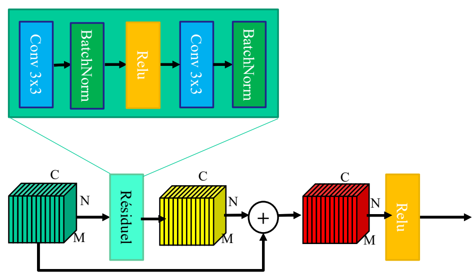

Rétro-propagation et gradients
Disparition du gradient
Couches conventionnelles
x
Conv 3x3
Relu
Conv 3x3
H(x)
Relu
ReLU(H(x))
Bloc résiduel
x
Conv 3x3
Relu
Conv 3x3
+
Relu
La connexion résiduelle prévient la disparition du gradient

134


2 types de blocs résiduels (détails)
x
Conv 3x3
BatchNorm
Relu
Conv 3x3
BatchNorm
H(x)+x
+
Relu
ReLU(H(x)+x)
w×h×p
réduire d'un facteur 2
w/2 × h/2 × p
x
Conv 3x3 Stride 2
BatchNorm
Relu
Conv 3x3
BatchNorm
H(x)+f(x)
+
Relu
ReLU(H(x)+f(x))
Conv 1x1 – Stride 2
w/2 × h/2 × p
Réduction d'un facteur 2 de la taille des cartes d'activation

135

Autre illustration



136

136

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
```

137

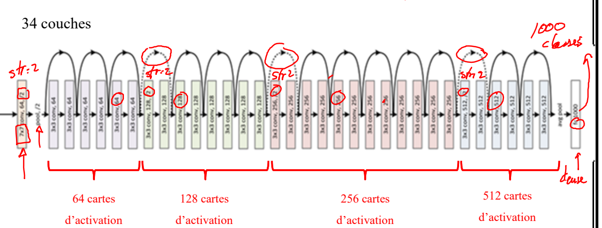
137

ResNet - 34

ImageNet 1.4M images
1000 classes.

ResNet [He et al, 2016]

34 couches



138

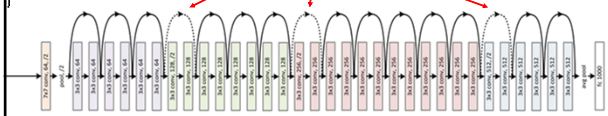
138

ResNet - 34

Réduction d'un facteur 2 de la resolution des cartes d'activation par **conv stride 2**

ResNet **[He et al, 2016]**

34 couches



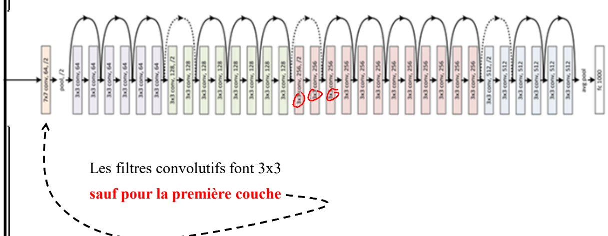
139

139

ResNet - 34

ResNet **[He et al, 2016]**

34 couches



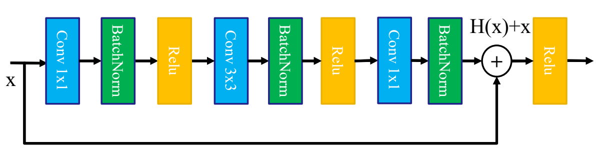
Les filtres convolutifs font 3x3
sauf pour la première couche

140

140

ResNet - 34

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

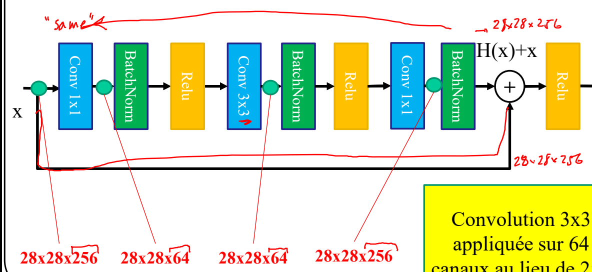


141

141

ResNet – 50,101, 152

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



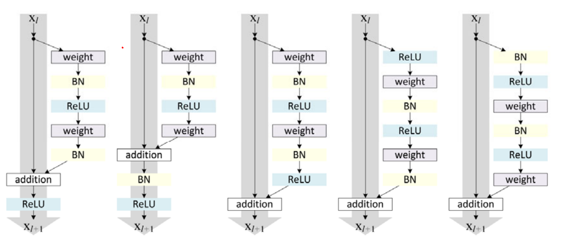
Convolution 3x3 appliquée sur 64 canaux au lieu de 256

142

ResNet

" model zoo "

Autres types de connexions résiduelles



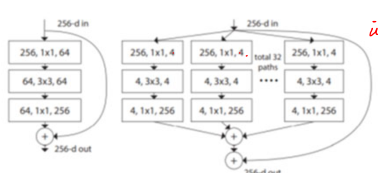
(a) original (b) BN after addition (c) ReLU before addition (d) ReLU-only pre-activation (e) **full pre-activation**

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

143

ResNeXt [Xie et al. 2017]

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



inspiré du "inception Net" "GoogleNet"

Bloc ResNet
50 couches et +

Bloc ResNeXt

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

144

144

ResNeXt

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

Bottleneck
Residual layers

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

145

145

ResNeXt

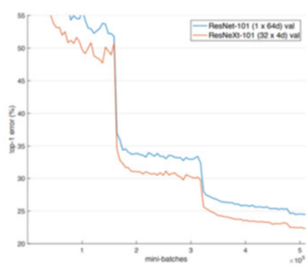


Figure 6. **ImageNet-5K** experiments. Models are trained on the 5K set and evaluated on the original 1K validation set, plotted as a 1K-way classification task. ResNeXt and its ResNet counterpart have similar complexity.

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

146

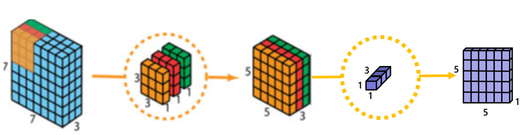
146

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

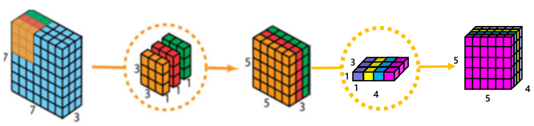
147

147

MobileNet
[Howard et al.'17]

Objectif : proposer un réseau efficace et sobre en calculs

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

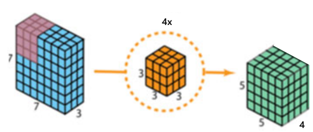


Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 148

148

MobileNet
[Howard et al.'17]

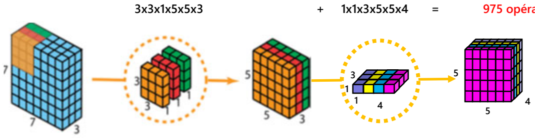
Conv traditionnel



3x3x3x5x5x4 = 2700 opérations

DW Conv

3x3x1x5x5x3 + 1x1x3x5x5x4 = 975 opérations

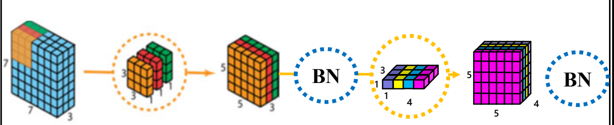


Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arxiv 1704.04861, 2017 149

149

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 150

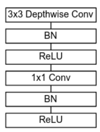
150

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	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ 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 151

151

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 152

152

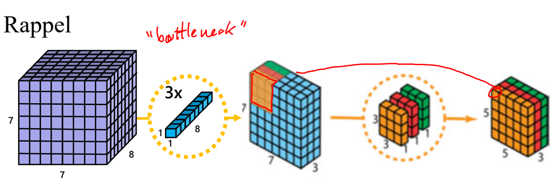
XceptionNet

[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 153

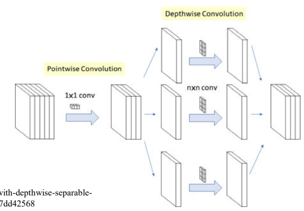
153

XceptionNet

[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

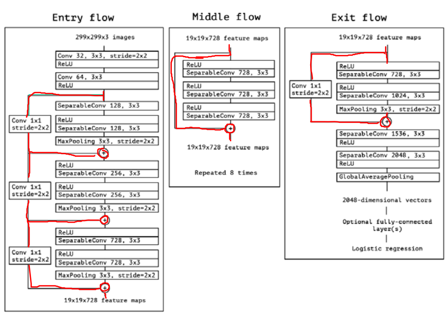
F.Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, CVPR 2017 154

154

XceptionNet

Image tirée de l'article

DWConv + Connexions résiduelles



F.Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, CVPR 2017 155

155

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.

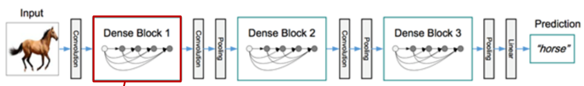
	Top-1 accuracy	Top-5 accuracy
VGG-16	0.715	0.901
ResNet-152	0.770	0.933
Inception V3	0.782	0.941
Xception	**0.790**	**0.945**

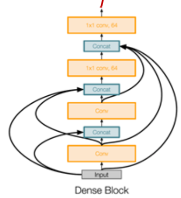
F.Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, CVPR 2017 156

156

DenseNet

[Huang et al, 2017]





Idée d'un bloc dense:

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

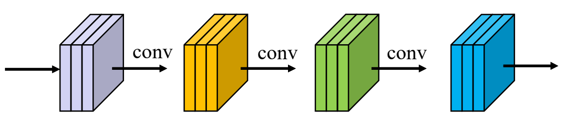
Image: http://cs231n.stanford.edu/slides/2018

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

159

159

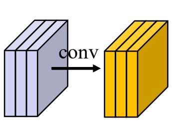
Bloc de 4 couches convolutives ordinaires avec 3 filtres chaque (fonctions d'activation omises pour alléger la figure)



160

160

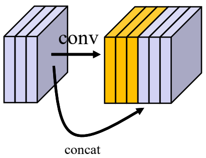
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque (fonctions d'activation omises pour alléger la figure)



161

161

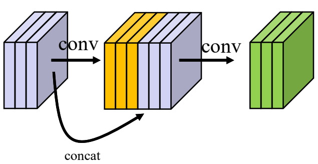
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



162

162

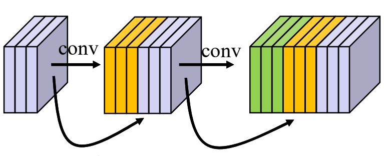
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



163

163

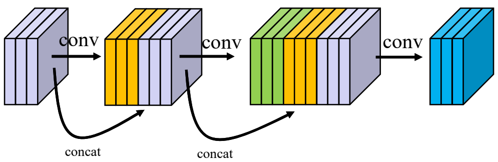
Bloc de 4 couches convolutives **denses** avec 3 filtres chaque
(fonctions d'activation omises pour alléger la figure)



164

164

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



165

165

Bloc dense

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

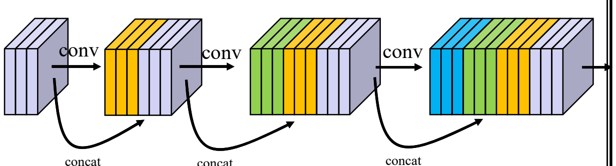


Image: http://cs231n.stanford.edu/slides/2018

166

166

Bloc dense

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

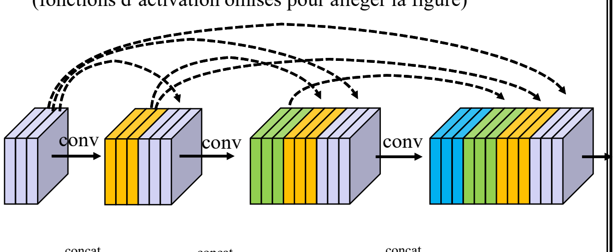
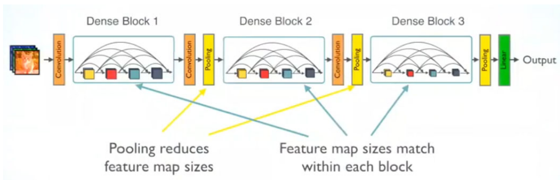


Image: http://cs231n.stanford.edu/slides/2018

167

167

DenseNet



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

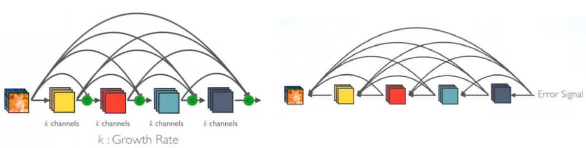
168

168

DenseNet

Avantages:

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



Propagation avant

Rétro-propagation

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

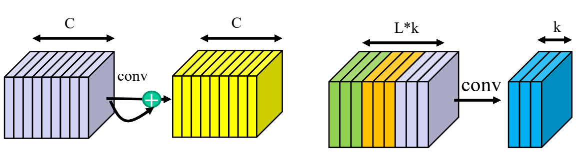
169

169

DenseNet

Avantages:

2. Peu de paramètres



Resnet

CxCx3x3 paramètres

DenseNet

L*k*k*3*3 paramètres

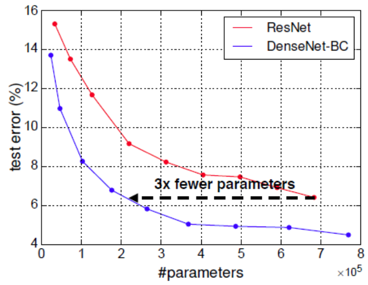
k<<C

170

DenseNet

Avantages:

2. Peu de paramètres

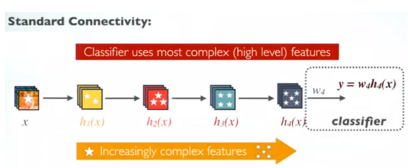


171

DenseNet

Avantages:

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



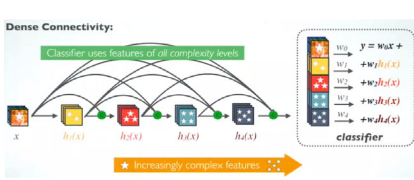
Avec un CNN conventionnel, le classificateur base sa prédiction sur les caractéristiques de la dernière couche, c-à-d des **caractéristiques de haut niveau**

172

DenseNet

Avantages:

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



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

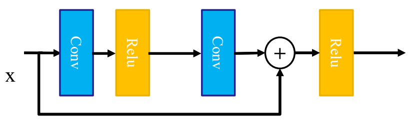
173

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

174

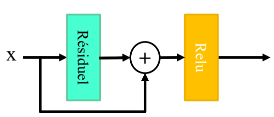
174

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

175

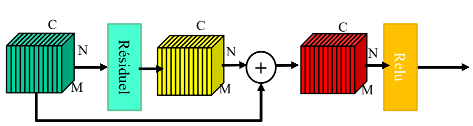
Squeeze-and-Excite Net (SENet)

[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

(avec illustration des cartes d'activation)



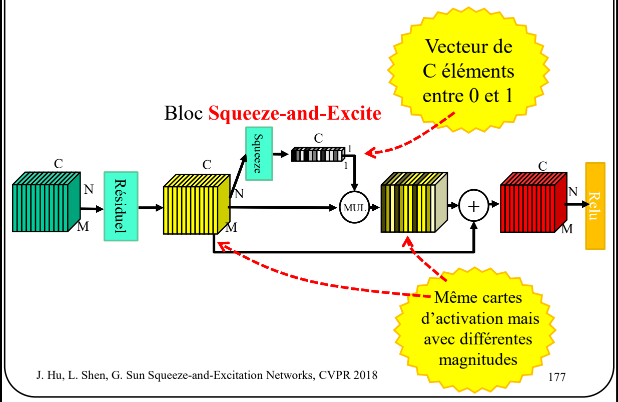
Avant et après le résiduel : cartes d'activation de taille N*M*C

Après l'addition : cartes d'activation de taille N*M*C

176

176

Squeeze-and-Excite Net (SENet)

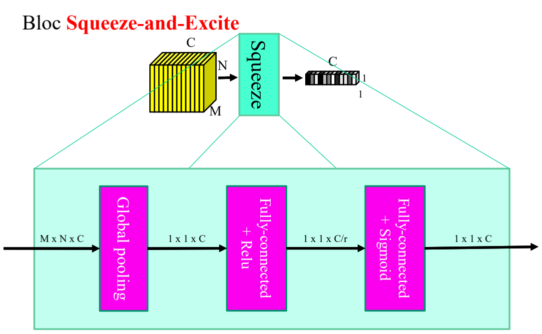


J. Hu, L. Shen, G. Sun Squeeze-and-Excitation Networks, CVPR 2018

177

177

Squeeze-and-Excite Net (SENet)



r: « squeeze rate »

Sigmoid force les valeurs du vecteur de sortie entre 0-1

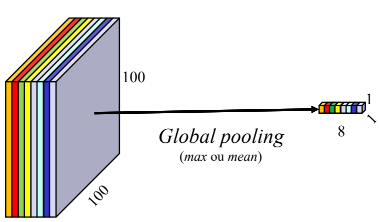
178

Rappel

Global pooling

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

Résultat : un **vecteur** de la taille du nombre de canaux

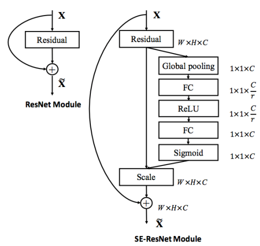


179

Squeeze-and-Excite Net (SENet)

[Hu et al. 2018]

Autre illustration



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

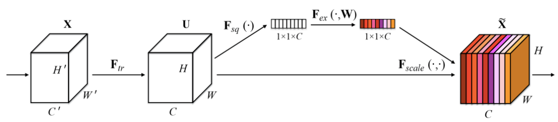
180

180

Squeeze-and-Excite Net (SENet)

[Hu et al. 2018]

Autre illustration



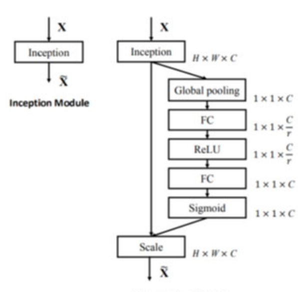
J. Hu, L. Shen, G. Sun Squeeze-and-Excitation Networks, CVPR 2018

181

181

Squeeze-and-Excite Net (SENet)

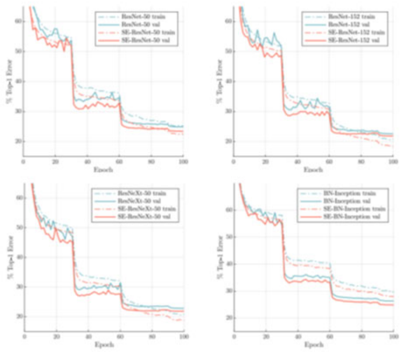
Fonctionne également avec les **modules d'inception**.



182

182

Meilleur que ResNet, ResNeXt et InceptionNet



183

183

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

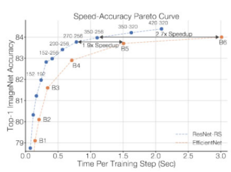


Figure 4. **Speed-Accuracy Pareto curve comparing ResNets-RS to EfficientNet.** Properly scaled ResNets (ResNet-RS) are **1.7x - 2.7x** faster than the popular EfficientNets when closely

Improvements	Top-1	Δ
ResNet-200	79.0	—
+ 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).

184

184

Improvements	Top-1	Δ	
ResNet-200	79.0	—	
+ Cosine LR Decay	79.3	**+0.3**	Méthodes d'entraînement
+ Increase training epochs	78.8 †	-0.5	
+ EMA of weights	79.1	**+0.3**	Méthodes de régularisation
+ 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**	Améliorations de l'architecture de base
+ 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).

185

185