# Optimisation des réseaux convolutifs profonds

RCP 209, 2026 – 2027

Davide Carbone

le cnam

# Au-delà de la descente de gradient stochastique (SGD)

## Limites de la descente de gradient

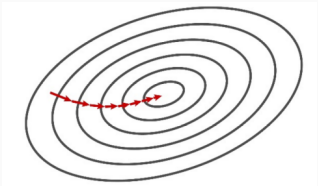
Optimisation par descente de gradient:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial f}{\partial \mathbf{w}} \left(\mathbf{w}^t\right) = \mathbf{w}^t - \eta \nabla f\left(\mathbf{w}^t\right)$$

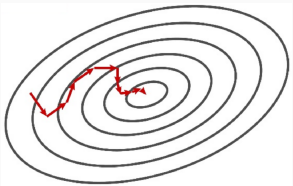
Deux problèmes:

(i) la fonction objectif $f$ peut changer rapidement selon une direction et lentement selon une autre

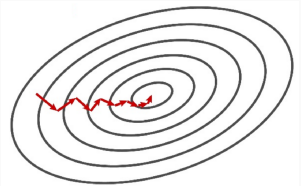
(ii) la descente de gradient *stochastique* utilise une approximation du gradient réel



Descente de gradient



Descente de gradient en ligne



Descente de gradient par batch

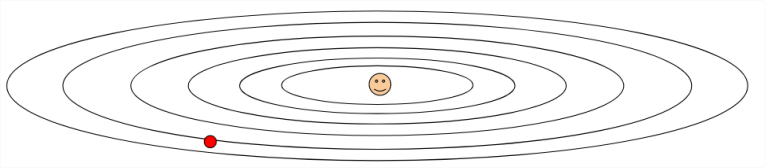
# Pourquoi la convergence peut-elle être lente?

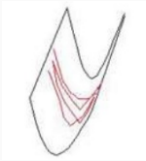
**Matrice Hessienne:**

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{bmatrix}$$

- Mauvais conditionnement de la matrice Hessienne $\implies$ valeur élevée du nombre de condition (rapport plus grande/plus petite valeur singulière)

$\implies$ "Vallée": décroissance rapide selon une direction, lente selon une autre.





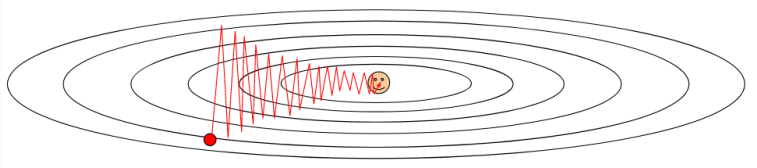
# Pourquoi la convergence peut-elle être lente?

**Matrice Hessienne:**

$$H(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{bmatrix}$$

- Mauvais conditionnement de la matrice Hessienne $\Longrightarrow$ valeur élevée du nombre de condition (rapport plus grande/plus petite valeur singulière)

$\Longrightarrow$ "Vallée": décroissance rapide selon une direction, lente selon une autre.

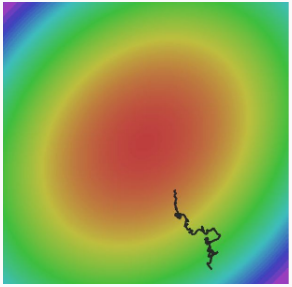


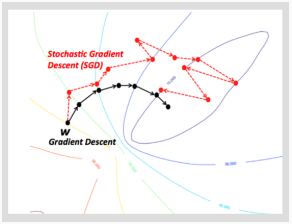
$\rightarrow$ **Descente de gradient:** progrès lent selon la direction "étroite", oscillations selon la direction "raide".

# Approximation du gradient réel

- Fonction objectif $f(\mathbf{w}) = \sum_{i=1}^{N} f_i(\mathbf{w})$, par exemple $f_i(\mathbf{w}) = \ell_{CE}(\hat{\mathbf{y}}_i, \mathbf{y}_i^*)$
- **Descente de gradient *stochastique*:** approximation du véritable gradient, par exemple sur un mini-batch:

$$\nabla f\left(\mathbf{w}^t\right) \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial f_i(\mathbf{w})}{\partial \mathbf{w}} \left(\mathbf{w}^{(t)}\right)$$





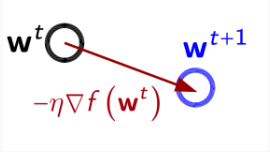

⇒ *l'utilisation d'un gradient approximatif introduit du bruit et une convergence difficile*

# Solution 1: méthode de l'inertie (*momentum*)

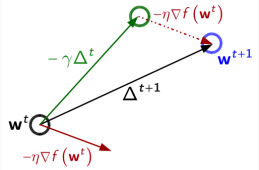
## Principe

- Dans l'algorithme de descente de gradient: $\boxed{\mathbf{w}^{t+1} = \mathbf{w}^t - \Delta^{t+1}}$
  - Ici, $\Delta^{t+1}$ représente le vecteur de mise à jour des paramètres $\mathbf{w}^t \rightarrow \mathbf{w}^{t+1}$
  - Dans l'algorithme classique: $\Delta^{t+1} = \eta \nabla f\left(\mathbf{w}^t\right)$ avec $\eta$ le pas d'apprentissage.
- **Moment**: conserver une moyenne glissante des mises à jour passées pour exploiter "l'historique" du gradient:

$$\Delta^{t+1} = \eta \nabla f\left(\mathbf{w}^t\right) + \gamma \Delta^t \quad \text{avec} \quad \gamma \in [0; 1[$$




*Mise à jour à l'itération $t$*




*Mise à jour à l'itération $t+1$*

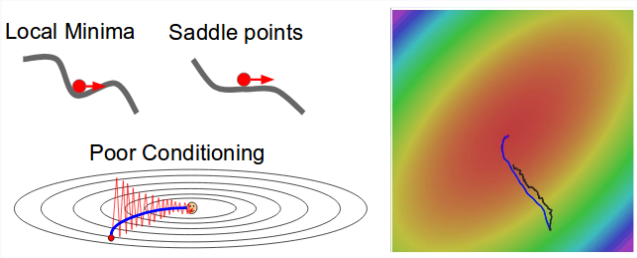
La technique du moment s'applique à tous les algorithmes "basés gradient".

# Avantages de l'inertie

- $\Delta^t$: $\mathbf{v}^t$ est la vélocité, avec une inertie: $\mathbf{w}^{t+1} = \mathbf{w}^t - \mathbf{v}^{t+1}$
  - Dans les directions où le gradient oscille, la vélocité se compense.
  - Dans les directions où le gradient est petit mais la direction constante, la vélocité s'accumule (et croît).

$\Longrightarrow$ convergence plus robuste lors de la traversée des plateaux, des minima locaux et des points selles

$\Longrightarrow$ estimation du gradient bruité lissée sur plusieurs itérations

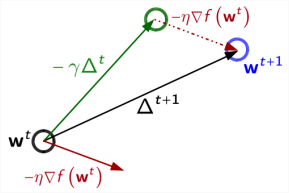


# Solution 2: gradient accéléré de Nesterov (NAG)

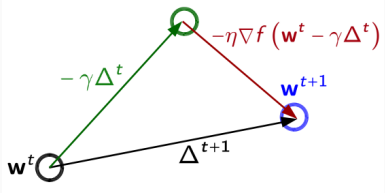
## *Nesterov Accelerated Gradient* [Sutskever et al. 2013]

≈ Gradient +*momentum*, mais le calcul du gradient se fait à la position *future* (le déplacement par l'inertie $\Delta^t$ est apliquée avant la mise à jour)

$$\Delta^{t+1} = \eta \nabla f\left(\mathbf{w}^t - \gamma \Delta^t\right) + \gamma \Delta^t \qquad \text{généralement } \gamma \sim 0.9$$
$$\mathbf{w}^{t+1} = \mathbf{w}^t - \Delta^{t+1}$$




*Gradient + momentum*




*NAG*

# Avantages du gradient accéléré de Nesterov

$$\Delta^{t+1} = \eta \nabla f\left(\mathbf{w}^t - \gamma \Delta^t\right) + \gamma \Delta^t \quad \text{généralement } \gamma \sim 0.9$$

En réécrivant $\mathbf{x}^t = \mathbf{w}^t - \gamma \Delta^t$, la règle de mise à jour peut être simplifiée:

$$\Delta^{t+1} = \eta \nabla f(x^t) + \gamma \Delta^t$$
$$\mathbf{x}^{t+1} = \mathbf{x}^t + \gamma \Delta^t - (\gamma + 1)\Delta^{t+1}$$

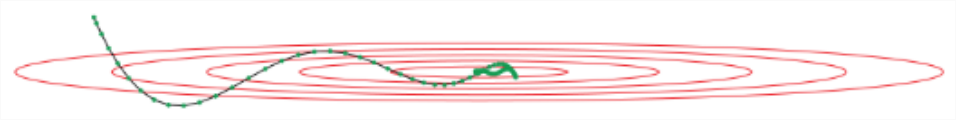
**Avantages**

La mise à jour "anticipée" permet:

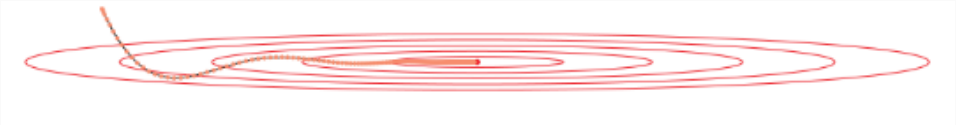
- ⊕ d'éviter de mises à jour de trop grande amplitude (overshooting)
- ⊕ d'accroître la réactivité de l'optimiseur à la surface de la fonction objectif

# Avantages du gradient accéléré de Nesterov

Momentum



NAG



# Pas d'apprentissage variable

**Objectif:** adapter le pas d'apprentissage pour chaque paramètre, en fonction de la surface de la fonction de coût.

Règle de mise à jour **pour chaque paramètre**: $\boxed{w_i^{t+1} = w_i^t - \Delta_i^{\mathbf{t+1}}}$

## Adaptive Gradient (Adagrad) [Duchi, Hazan, and Singer 2011]

- On note $g_i^t = \frac{\partial f}{\partial w_i}(w_i^t)$ le gradient selon une seule composante $i$.
- $\mathbf{G_i^t} = \sum_{\mathbf{i=1}}^{\mathbf{t}} (\mathbf{g_i^t})^{\mathbf{2}}$ (historique des gradients au carré)
- $\sqrt{G_i^t}$: norme $\ell_2$ du gradient pour $t \in \{1, t\}$

**Adagrad**: ($\varepsilon \sim 10^{-8}$, $\eta \sim 10^{-2}$): $\quad \Delta_i^{t+1} = \frac{\eta}{\sqrt{G_i^t + \varepsilon}} g_i^t = \eta' g_i^t$

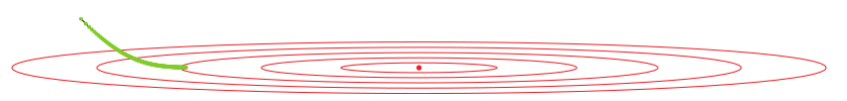
$\rightarrow$ **Intuition:** si $G_i^t$ est grand $\Rightarrow \downarrow \eta'$, si $G_i^t$ est petit $\Rightarrow \uparrow \eta'$

$\Rightarrow$ encourager des mises à jour pour des paramètres qui contribuent beaucoup à la décision mais qui sont relativement rarement modifiés

# RMSProp

Inconvénient d'Adagrad: $(g_i^t)^2 \geq 0 \Rightarrow 1/G_i^t$ décroît de façon monotone

$\Longrightarrow$ le pas d'apprentissage décroît fortement avec le nombre d'itérations, ce qui peut stopper les mises à jour…



**Root Mean Square Propagation (RMSProp) [Tielman and G. Hinton 2012]**

Plutôt que de calculer $G_i^t$ sur tout l'historique, conserver seulement une moyenne glissante:

$$\tilde{G}_i^t = \rho G_i^{t-1} + (1-\rho)(g_i^t)^2 \quad \text{avec } \rho \sim 0.9$$

$$\Delta_i^{t+1} = \frac{\eta}{\sqrt{\tilde{G}_i^t + \varepsilon}} g_i^t = \eta' g_i^t$$

$\rightarrow$ $\tilde{G}_i^t$ n'est **pas** une fonction croissante monotone

$\rightarrow$ décroissance moins agressive du pas d'apprentissage

# Adaptive Moment Estimation (Adam)

## Adam

Adam $\sim$ RMSProp + momentum

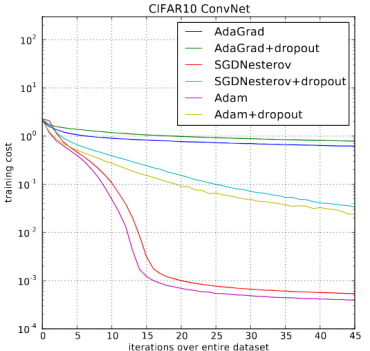
- Deux paramètres: $\beta_1, \beta_2$
- Définitions: moment de 1er et 2ème ordre du gradient

$$m_i^t = \beta_1 m_i^{t-1} + (1-\beta_1)(g_i^t)$$
$$v_i^t = \beta_2 v_i^{t-1} + (1-\beta_2)(g_i^t)^2$$

- Mise à jour: $w_i^{t+1} = w_i^t - \eta \frac{\hat{m}_i^t}{\sqrt{\hat{v}_i^t + \varepsilon}}$

# Pas d'apprentissage adaptatif

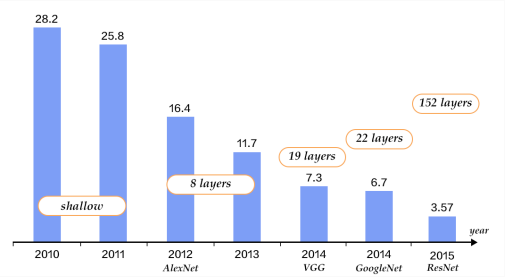
- Mise à jour pour chaque dimension: utile dans le cas des données*sparse*
- Adagrad, RMSProp, (Adadelta) et Adam: estimation du moment de 2ème ordre du gradient au dénominateur
- (Adadelta), Adam: ajout d'une inertie au numérateur
- Adam est un bon choix par défaut dans la plupart des cas



# Apprentissage profond depuis 2012

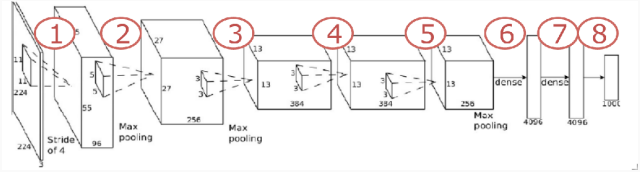
## Compétition ImageNet (ILSVRC)

- Jeu de données de 1 000 000 images réparties dans 1 000 classes
- **2011**: première implémentation rapide d'un CNN sur GPU, victoire dans deux compétitions (reconnaissance de caractères chinois, panneaux routiers).
- **2012**: première édition remportée par un CNN.
- **2013, 2014, 2015**: ↗ des performances, ↗ de la profondeur des CNN.



# AlexNet (ImageNet 2012)

- [Krizhevsky, Sutskever, and G. E. Hinton 2012]
- 5 couches convolutives, 3 couches entièrement connectées (60M paramètres)
  → Couche convolutive: convolution + non-linéarité (ReLU) + pooling
- Entraînement sur 2 GPUs pendant une semaine

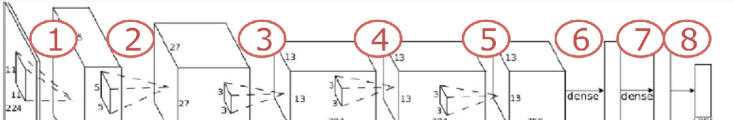



## Détails des couches

- **Entrée:** images $227 \times 227 \times 3$
- **Couches convolutives:** $11 \times 11$ (stride = 4) $\times 1$, $5 \times 5$ (stride = 2) $\times 1$, $3 \times 3$ (stride = 2) $\times 3$
- **Couches connectées:** $4096$ + Dropout, $4096$ + Dropout, $1000$

# AlexNet (ILSVRC)

- Architecture globalement identique aux modèles convolutifs historiques, comme LeNet
  - → Optimisation par rétropropagation et descente de gradient stochastique
- Modèle plus profond (et plus large): 60M paramètres (LeNet: seulement 60K)
  - → Nécessite beaucoup plus de données ($10^6$ pour ImageNet contre $10^4$ pour MNIST)
  - → Implémentation rapide sur GPU pour compenser les temps de calcul
- Quelques améliorations architecturales par rapport à MNIST:
  - → Non-linéarité ReLU plutôt que sigmoïde
  - → Régularisation par Dropout
  - → Augmentation de données

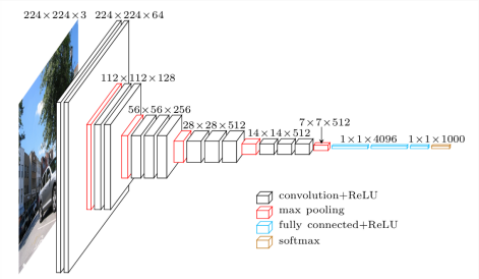


# ImageNet 2014: VGG-16 et VGG-19

Tentative d'adopter une approche "standardisée" de la construction de modèle

## Bloc convolutif VGG [Simonyan and Zisserman 2015]

- Deux ou trois couches convolutives $3 \times 3$, pas de 1
- Même nombre de filtres au sein d'un bloc, largeur $\times 2$ au bloc suivant
- Non-linéarité ReLU après chaque couche
- Maxpooling à la fin du bloc



# ImageNet 2014: VGG-16 et VGG-19

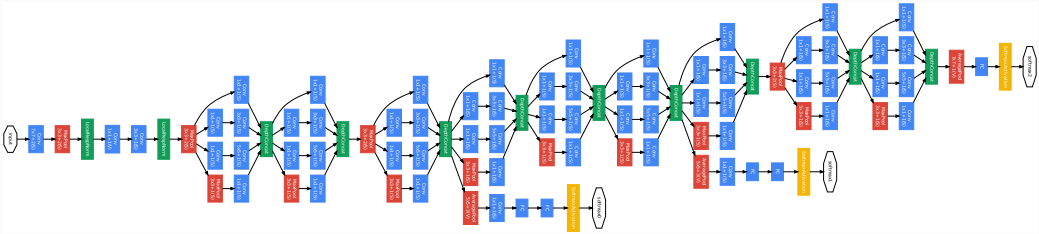
Tentative d'adopter une approche "standardisée" de la construction de modèle

## Bloc convolutif VGG [Simonyan and Zisserman 2015]

- Deux ou trois couches convolutives $3 \times 3$, pas de 1
- Même nombre de filtres au sein d'un bloc, largeur $\times 2$ au bloc suivant
- Non-linéarité ReLU après chaque couche
- Maxpooling à la fin du bloc



# ImageNet 2014: Inception



## Architecture “Inception” (GoogLeNet) [Szegedy et al. 2015]

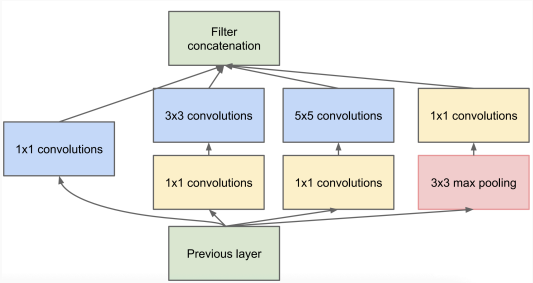
Architecture avec trois modules:

(i) “Base”: modèle convolutif (convolution + pooling)

(ii) Modules Inception: plusieurs convolutions en parallèle

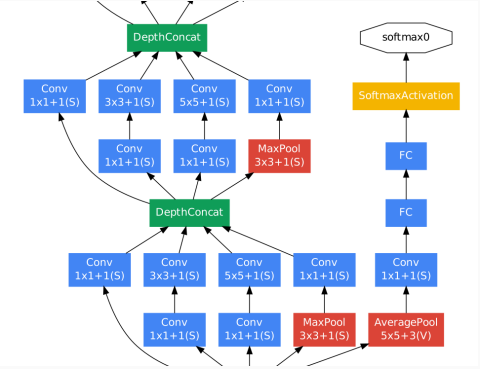
(iii) Classifieurs auxiliaires

# Module Inception




- **Intuition:** pourquoi choisir une seule taille de filtre plutôt que toutes les tester?
  - $\rightarrow$ Le bloc suivant pourra décider d'utiliser les activations $3 \times 3$, les activations $5 \times 5$, les activations après pooling, etc.
- Permet d'extraire des activations à plusieurs échelles
- Convolution $1$: réduction de dimension ($\searrow$ canaux)
  - $\rightarrow$ permet de maintenir raisonnable le nombre de paramètres du modèle

# Classifieurs auxiliaires




- Ajouter une branche annexe de classification sur une couche intermédiaire du modèle:
  - → Permet de rétropropager du gradient même aux couches inférieurs (limite les gradients évanescents)
  - → Ajoute une régularisation pendant l'apprentissage (même les activations des couches inférieures doivent être utiles à la classification)

Fonction de coût = somme pondérée de l'entropie croisée finale et des entropie croisées "auxiliaires"

$$\mathcal{L} = \ell_{\text{CE}} + 0.3\ell_{\text{CE}}^{(4a)} + 0.3\ell_{\text{CE}}^{(4d)}$$

# Inception: limiter le nombre de paramètres

**Comment limiter l'explosion du nombre de paramètres à optimiser dans un réseau très profond?**

AlexNet: $\approx$ 62 millions de paramètres

- 3M de paramètres pour les couches convolutives
- 59M de paramètres pour les couches entièrement connectées !

**Solution:** limiter le nombre de couches connectées.

Couches finales de Inception v1:

- Average pooling adaptatif pour réduire les dimensions spatiales
- Une seule couche connectée $1024 \rightarrow K$

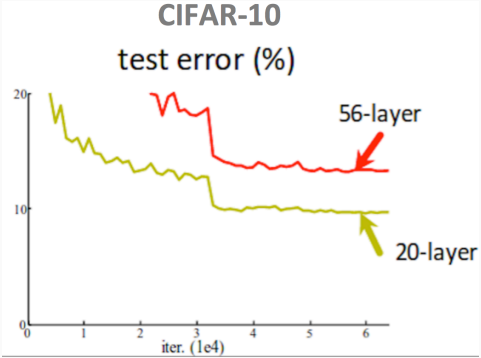
$\Longrightarrow$ **seulement 30M de paramètres pour Inception v1**

$\Longrightarrow$ le pooling adaptatif permet de traiter des images de dimensions arbitraires!

# Peut-on aller plus profond?

- Après une amélioration, les modèles plus profonds semblent avoir de moins bonnes performances.
- **Illustration:** modèles type VGG (empilement de convolutions $3 \times 3$)

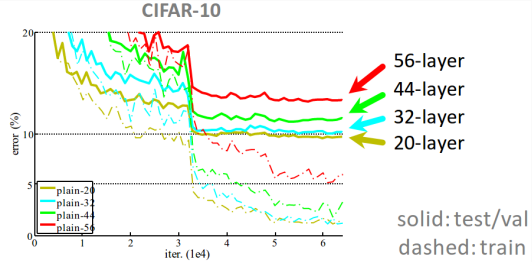
$\Longrightarrow$ le modèle VGG-56 obtient des performances inférieures au modèle VGG-20

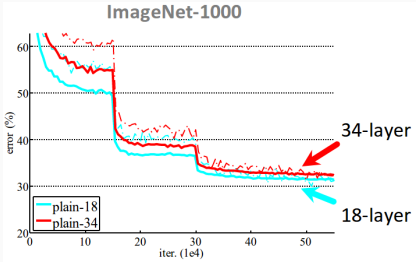


# Les réseaux profonds généralisent moins bien?

- Ce n'est **pas** un problème de généralisation/sur-apprentissage: l'erreur d'apprentissage est elle-aussi plus élevée avec un modèle plus profond!
- Phénomène général qui s'observe sur plusieurs jeux de données et des tâches différentes

$\Longrightarrow$ **problème d'optimisation, on ne parvient pas à minimiser la fonction de coût...**




$\rightarrow$ Deux solutions: normalisation (la suite !) et connexion résiduelle (cours 7).

# Batch Normalization (BN)

- Entraînement des réseaux profonds: descente de gradient $\Longrightarrow$ problème non-convexe donc sensible à l'initialisation.
  - Importance de l'initialisation pour obtenir des gradients de norme équivalente dans toutes les couches (par exemple, initialisation de Glorot).
- La distribution des activations des couches cachées est inconnue et change au cours de l'apprentissage.

**Batch Normalization [Ioffe and Szegedy 2015]**

Normaliser les **activations** en sortie de chaque couche.

$\Longrightarrow \searrow$ l'importance de l'initialisation, $\searrow$ les variations de distribution au cours de l'apprentissage.

# Implémentation de la Batch Normalization (BN)

**Principe:** centrer et réduire la distribution des activations pour chaque couche.

- Pour les activations $x_i$ en sortie d'une couche donnée, estimer la moyenne et la variance sur un mini-batch:

$$\mu_B = \frac{1}{N}\sum_{i=1}^{N} x_i, \qquad \sigma_B^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_B)^2$$

- Centrer et réduire les activations:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B + \epsilon}$$

avec $\epsilon$ faible ($\approx 10^{-6}$) pour la stabilité numérique.

- Opération différentiable (permet la rétropropagation)
- S'applique après la couche (convolutive ou entièrement connectée) mais avant l'activation

# BatchNorm et transformation affine

Centrer-réduire pour obtenir une distribution des activations $\sim \mathcal{N}(0, 1)$ n'est pas toujours une bonne idée:

- Certaines fonctions d'activation peuvent ne jamais saturer (*e.g.*, sigmoïde ou $\tanh$)
- Rester proche de 0 bloque les activations dans le régime linéaire $\approx$ modèle linéaire indépendamment de la profondeur

**Batch Norm en pratique: centrer-réduire + transformation affine**

(i) Normaliser les activations:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B + \epsilon}$$

(ii) Application d'une transformation affine (scale + shift):

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

avec $\gamma$ et $\beta$ des paramètres optimisables.

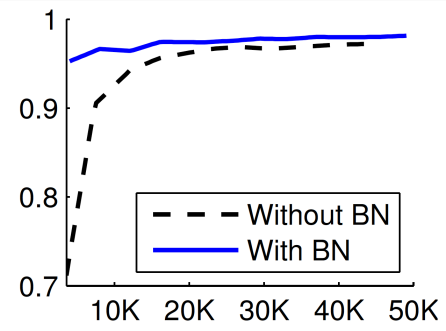
# Batch Normalization à l'inférence

**Comment appliquer la Batch Norm en test?**

- Option 1: normaliser les activations sur le batch (même opération qu'en entraînement).
  - ⊖ les résultats ne sont plus déterministes en fonction de la taille du batch, ou des données passées en entrée
- Option 2: normaliser à partir des statistiques (activation moyenne et variance) estimées sur le jeu d'apprentissage
  - → lors de l'apprentissage, on stocke les valeurs de $\mu_B$ et $\sigma_B$ avec une moyenne glissante pour les réutiliser (fixées) en test.

# Avantages de la Batch Normalization

- **Réduction des problèmes de gradients évanescents ou explosifs:** une mise à jour importante des poids ne risque plus de produire des changements importants des activations, puisqu'elles sont normalisées.
- **Convergence plus rapide** du modèle (permet d'utiliser un *learning rate* plus élevé)
- Effet régularisateur souvent positif sur les performances en **généralisation**

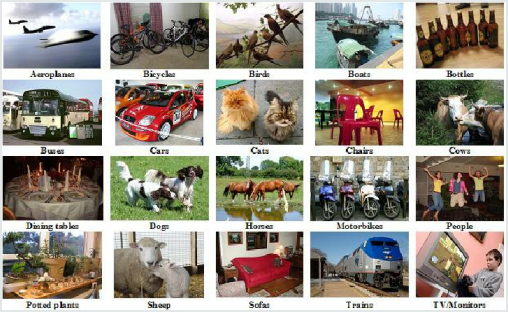


# Apprentissage profond sur des petits jeux de données

Les modèles profonds ont beaucoup de paramètres.

**Que se passe-t-il sur des jeux d'entraînement de petite taille ?**

## Exemple: Pascal VOC 2007




- 20 catégories, 5 000 exemples d'apprentissage.
- Apprentissage d'un modèle VGG-16: $\approx$ 40% mAP
- Modèle "sac-de-mots visuels" avec des caractéristiques expertes: $\approx$ 70% mAP!

# Comparaison avec ImageNet





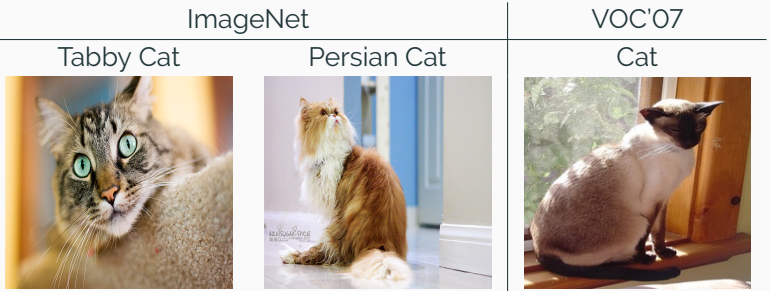

- ImageNet: deep >> hancrafted
- VOC'07: deep << hancrafted
  - 200 fois moins d'exemples
  - Image et tâche plus complexe (multi-classes)

# Principe de l'apprentissage par transfert

**Intuition**: exporter les connaissances apprises sur une tâche source vers une tâche cible.

- Source: bonnes performances de généralisation, beaucoup d'exemples.
- Cible: tâche plus difficile, peu d'exemples.

**Hypothèse**: les tâches source et cibles sont différentes mais présentent des similarités.



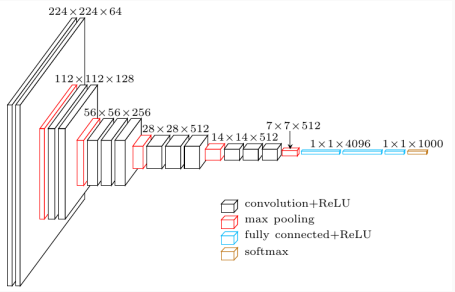
# Approche naïve

(1) On part d'un modèle (par exemple, VGG-16) "pré-entraîné" pour la classification sur ImageNet.

(2) Appliquer le modèle sur chaque exemple du jeu de données cible (par exemple, VOC 2007).

(3) Extraire les activations d'une couche au choix (par exemple, `fc7`) $\Longrightarrow$ caractéristiques profondes (*deep features*)

(4) Utiliser les vecteurs obtenus comme descripteurs pour un modèle de ML classique (random forest, SVM, etc.)



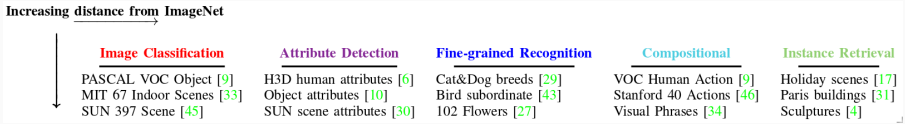
# Caractéristiques profondes

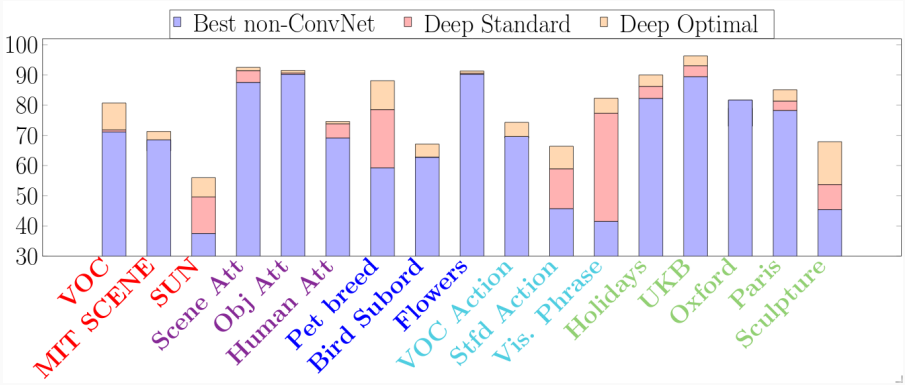
- L'usage des *deep features* a permis aux CNNs de s'étendre bien au-delà d'ImageNet.

$\Rightarrow$ un grand jeu de données ou réentraîner un modèle profond n'est **pas** indispensable!

- Les modèles pré-entraînés sur ImageNet généralisent même à des domaines éloignés
  - $\rightarrow$ imagerie satellitaire, imagerie médicale (échographie, radios, IRM...)
  - $\rightarrow$ 1000 classes $\Longrightarrow$ large ensemble de concepts visuels génériques
- Les caractéristiques profondes s'obtiennent "sur étagère" et presque toujours équivalentes ou meilleures que les plus descripteurs experts les plus performants.

# Caractéristiques profondes

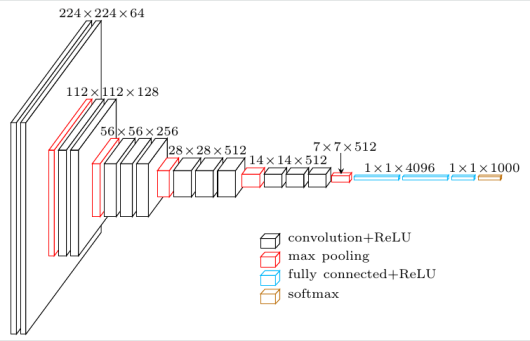


# Fine-tuning: "spécialisation"

**Intuition:** remplacer la dernière couche du modèle par une nouvelle, qui sera "spécialisée" sur le nouveau jeu de données.

## Exemple sur VGG-16 pour la classification

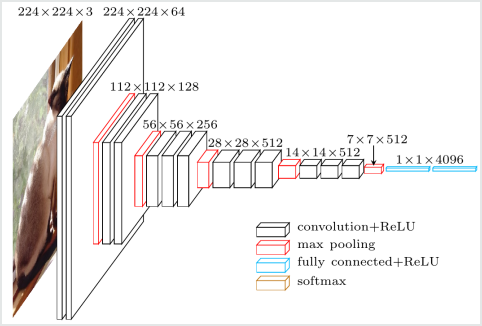



→ VGG-16 pour ImageNet: dernière couche entièrement connectée = 1000 classes

# Fine-tuning: “spécialisation”

**Intuition:** remplacer la dernière couche du modèle par une nouvelle, qui sera “spécialisée” sur le nouveau jeu de données.

## Exemple sur VGG-16 pour la classification

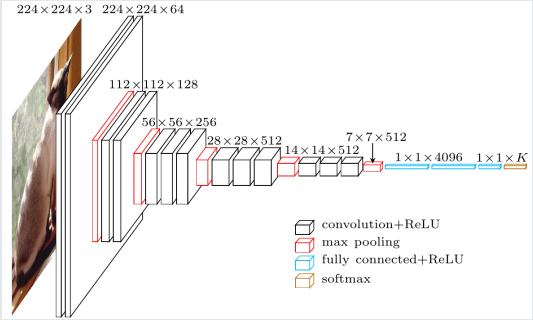



→ Retrait de la dernière couche

# Fine-tuning: “spécialisation”

**Intuition:** remplacer la dernière couche du modèle par une nouvelle, qui sera “spécialisée” sur le nouveau jeu de données.

## Exemple sur VGG-16 pour la classification




→ Remplacement par une nouvelle couche à $K$ classes ($K = 20$ pour VOC 2007)

# Stratégies de fine-tuning

**3 stratégies possibles:**

## Pur transfert

Les poids de toutes les couches sont "gelés", sauf la dernière couche.

$\Longrightarrow$ **pas de mise à jour sur les couches sauf la dernière**

## Fine-tuning

Les poids de **toutes les couches sont mis à jour** par la descente de gradient, mais le **pas d'apprentissage est diminué** pour toutes les couches, sauf la dernière.

Typiquement, $\eta$ est 10 fois plus faible pour les couches inférieures par rapport à la couche *fine-tuned*.

## Transfert par initialisation

On utilise les poids du modèle pré-entraîné comme initialisation pour un apprentissage classiques. Tous les paramètres sont mis à jour normalement.

# Quelques exemples: Pascal VOC2007

- Petit jeu de données (1000 à 10000 exemples), sémantiquement et visuellement proche de ImageNet.

| Modèle | mAP (%) |
|---|---|
| VGG (*from scratch*) | $\approx 40$ |
| Descripteurs experts | $\approx 70$ |
| VGG (transfert pur) | $\approx 83$ |
| VGG (fine-tuning) | $\approx 85$ |

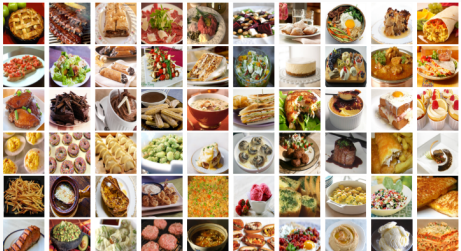
- Fine-tuning > transfert >> approche experte
- Modèle appris *from scratch*: faibles performances car **sur-apprentissage** (pas assez de données)



# Quelques exemples: UPMC-food-101

- Jeu de données de taille moyenne ($\geq$ 10000 exemples), visuellement et sémantiquement proche d'ImageNet.
- 100 classes, 1000000 exemples.
- Modèle VGG *from scratch*: performances correctes
- Fine-tuning >> *from scratch* >> transfert >> approche experte

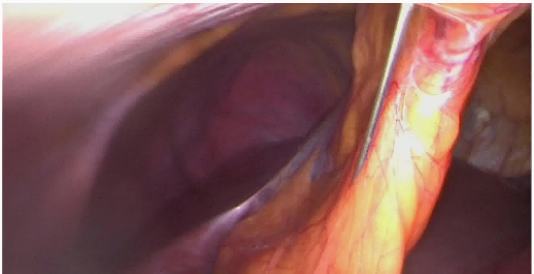
$\rightarrow$ Combiner les connaissances apprises sur ImageNet + les connaissances spécifiques au jeu de données permet de dépasser le modèle entraîné sur UPMC-food-101 seul!



| Modèle | mAP (%) |
|---|---|
| Descripteurs experts | $\approx 25\%$ |
| VGG (transfert) | $\approx 40\%$ |
| VGG (from scratch) | $\approx 53\%$ |
| VGG (fine-tuning) | $\approx 65\%$ |

# Quelques exemples: M2CAI 2016

- Jeu de données médical de taille moyenne ($\approx$ 10 000 images), éloigné d'ImageNet
- 22 vidéos, 8 classes.
- Fine-tuning >> from scratch >> transfert
- Le transfert obtient des performances relativement élevées sans aucun ré-apprentissage, en dépit de l'écart important sur le contenu visuel des jeux de données considérés.



| Modèle | *Accuracy* (%) |
|---|---|
| VGG (transfert) | $\approx 60\%$ |
| VGG (from scratch) | $\approx 70\%$ |
| VGG (fine-tuning) | $\approx 80\%$ |

# Conclusion

**Règle du pouce**

Plus l'écart entre le jeu de données source et le jeu de données cible est grand, plus il sera nécessaire d'effectuer le fine-tuning sur de nombreuses couches.

| | **Écart visuel faible** | **Écart visuel important** |
|---|---|---|
| **Peu de données** | Transfert | Fine-tuning (couches basses) |
| **Beaucoup de données** | Fine-tuning (couches hautes) | Fine-tuning (tous)/initialisation |

**À retenir**

De manière générale, le transfert direct depuis un modèle appris sur une grande quantité de données permet généralement d'obtenir des performances élevées **sans** ré-entraînement du réseau.

$\Longrightarrow$ *baseline* à envisager presque systématiquement.

# Normalisation des images

**Quels prétraitements appliquer aux images en entrée d'un CNN?**

→ **Apprentissage profond:** laisser le modèle apprendre les représentations adaptées
$\implies$ **éviter d'inclure trop d'a priori** avec les prétraitements

Une normalisation minimaliste est suffisante en pratique:

$(i)$ Normaliser les valeurs d'intensité des pixels entre 0 et 1 si ce n'est pas déjà le cas
→ Pour du RGB sur 8 bits, diviser par 256.

$(ii)$ Soustraire à tous les pixels le pixel moyen calculé canal par canal:

$$\hat{\mathcal{I}} = \mathcal{I} - [r_{\text{mean}}, g_{\text{mean}}, b_{\text{mean}}]$$

# Augmentation de données

- La construction de modèles avec des architectures profondes introduit de plus en plus de paramètres et des modèles de capacité ↗
- Risque de sur-apprentrissage $\Longrightarrow$ régularisation
- Idéalement, on aimerait disposer de plus de données $\Longrightarrow$ *data augmentation*

**Principe**

Générer des variantes artificielles des données existantes.

Exemple pour des images:

- Symétrie verticale
- Ajout de bruit blanc faible
- Changements aléatoires de contraste
- etc.



# Bibliographie i


Duchi, John, Elad Hazan, and Yoram Singer [July 2011]. **"Adaptive Subgradient Methods for Online Learning and Stochastic Optimization".** In: The Journal of Machine Learning Research 12, pp. 2121–2159. ISSN: 1532-4435 [cit. on p. 11].


Ioffe, Sergey and Christian Szegedy [2015]. **"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".** In: Proceedings of the 32nd International Conference on Machine Learning. International Conference on Machine Learning (ICML), pp. 448–456. URL: `http://jmlr.org/proceedings/papers/v37/ioffe15.html` [visited on 05/27/2016] [cit. on p. 26].

# Bibliographie ii




Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton [2012]. **"ImageNet Classification with Deep Convolutional Neural Networks".** In: Proceedings of the Neural Information Processing Systems (NIPS). NeurIPS, pp. 1097–1105. URL: `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf` [cit. on p. 16].



Razavian, Ali Sharif et al. [June 2014]. **"CNN Features Off-the-Shelf: An Astounding Baseline for Recognition".** In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPR pp. 512–519. DOI: `10.1109/CVPRW.2014.131` [cit. on p. 36].

# Bibliographie iii


- Simonyan, Karen and Andrew Zisserman [May 2015]. **"Very Deep Convolutional Networks for Large-Scale Image Recognition".** In: Proceedings of the International Conference on Learning Representations (ICLR). URL: http://arxiv.org/abs/1409.1556 [cit. on pp. 18, 19].
- Sutskever, Ilya et al. [2013]. **"On the importance of initialization and momentum in deep learning".** In: International conference on machine learning. PMLR, pp. 1139–1147 [cit. on p. 8].
- Szegedy, Christian et al. [June 2015]. **"Going Deeper with Convolutions".** In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9. DOI: 10.1109/CVPR.2015.7298594. URL: http://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html [visited on 05/26/2016] [cit. on p. 20].

# Bibliographie iv

 Tielman, Tijmen and Geoffrey Hinton [2012].
**Lecture 6.5—RmsProp: Divide the Gradient by a Running Average of Its Recent Magnitu**
[Cit. on p. 12].