Résistance aux attaques adversaires et contre-attaques adversaires

Introduction

On introduit la notion de *résistance*, qui quantifie la difficulté à tromper un réseau de neurones classificateur avec un exemple adversaire créé à partir d'une image donnée.

Ce concept offre trois apports intéressants :

On cherchera d'abord plusieurs expressions possibles de la résistance, et on essaiera d'utiliser ce concept comme méthode pour détecter les exemples adversaires et améliorer la performance d'un réseau classificateur.

1. Les attaques adversaires

1.1 Les exemples adversaires

Les réseaux de neurones sont notoirement vulnérables aux attaques par *exemples adversaires* [1, 2] : il s'agit d'entrées imperceptiblement perturbées pour induire en erreur un réseau classificateur.

Plus concrètement, en considérant $\boldsymbol{Pred}$ la fonction qui à une image associe la prédiction du réseau, et en considérant une image $\boldsymbol{img}$ de $[\mathbf{0},\mathbf{1}]^{\boldsymbol{n}}$ (c'est à dire à $\boldsymbol{n}$ pixels), on cherche une perturbation $\boldsymbol{r}$ de norme minimale telle que :

$$\begin{cases} \boldsymbol{img} + \boldsymbol{r} \in [\mathbf{0},\mathbf{1}]^{\boldsymbol{n}} \\ \boldsymbol{Pred}(\boldsymbol{img} + \boldsymbol{r}) \neq \boldsymbol{Pred}(\boldsymbol{img}) \end{cases}$$

1.2 Les attaques adversaires

On cherche un algorithme qui détermine un exemple adversaire à partir d'une image donnée. On dit qu'un tel algorithme réalise une *attaque adversaire*.

Une méthode d'attaque possible est la suivante. Introduisons $\boldsymbol{Conf_c}$ la fonction qui à une image associe la probabilité (selon le réseau) que l'image appartienne à la catégorie $\boldsymbol{c}$; et soit une image $\boldsymbol{img}$ de catégorie $\boldsymbol{c}$. On cherche alors à minimiser par descente de gradient la fonction $\boldsymbol{Loss_1}$ suivante :

$$\boldsymbol{Loss_1} = \begin{cases} \|\boldsymbol{r}\| & \textbf{si } \boldsymbol{Conf_c}(\boldsymbol{img}+\boldsymbol{r}) \leq \mathbf{0.2} \\ \boldsymbol{Conf_c}(\boldsymbol{img}+\boldsymbol{r}) + \|\boldsymbol{r}\| & \textbf{sinon.} \end{cases}$$

(*Note : on utilisera ici la norme euclidienne. D'autres normes sont évidemment possibles, mais sans amélioration sensible des résultats*)

Cette première fonction est expérimentalement peu satisfaisante, car l'attaque échoue souvent. La perturbation reste "bloquée" en $\mathbf{0}$, et n'évolue pas. Pour pallier cela, on oblige la perturbation à grossir en ajoutant un troisième cas de figure, quand $\boldsymbol{Conf_c}(\boldsymbol{img}+\boldsymbol{r}) > \mathbf{0.95}$, c'est à dire quand la perturbation n'est pas du tout satisfaisante :

$$Loss_2 = \begin{cases} \|r\| & \text{si } Conf_c(img+r) \leq 0.2 \\ Conf_c(img+r) + \|r\| & \text{si } Conf_c(img+r) \leq 0.9 \\ Conf_c(img+r) - \|r\| & \text{sinon.} \end{cases}$$

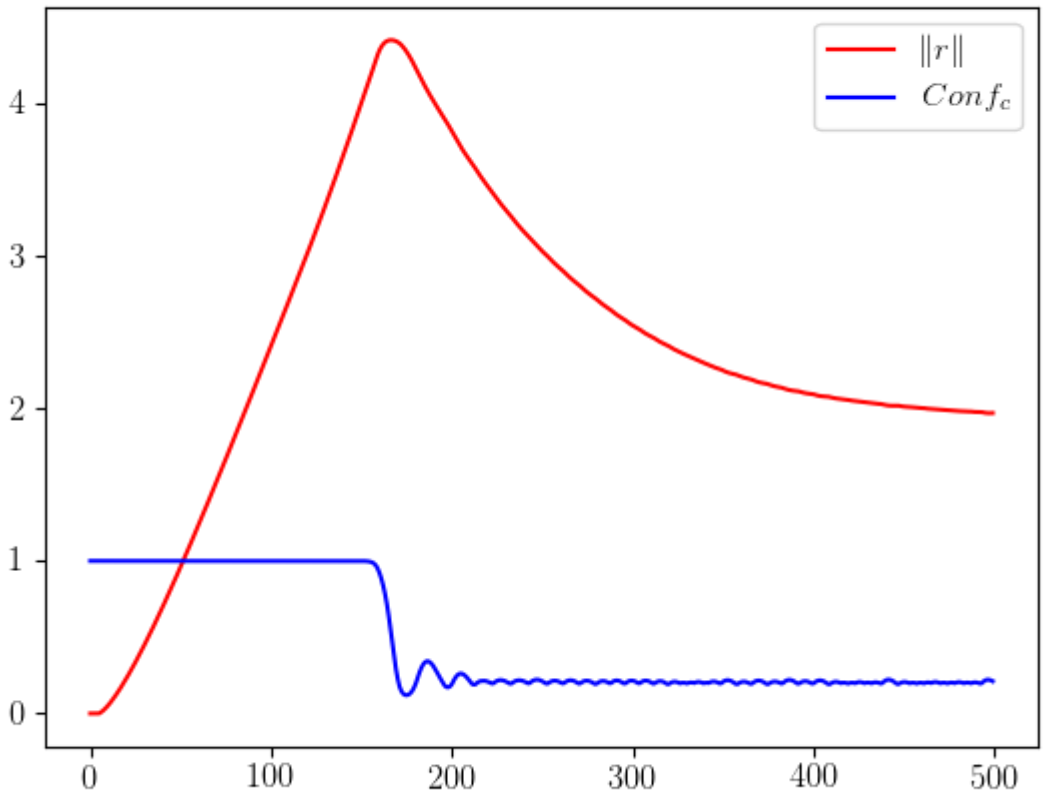
Cette deuxième fonction produit presque toujours un exemple adversaire pour un nombre d'étapes de descente de gradient suffisamment élevé (généralement 200 étapes suffisent), et c'est celle-ci qui sera utilisée par la suite.

On appellera $Pert_N$ la fonction qui à une image associe la perturbation obtenue après N étapes de descente de gradient (avec un taux d'apprentissage $\eta = 10^{-3}$).

1.3 Quelques résultats

On réalise l'attaque adversaire en effectuant 500 étapes de descente du gradient de $Loss_2$, avec un taux d'apprentissage $\eta = 10^{-3}$.

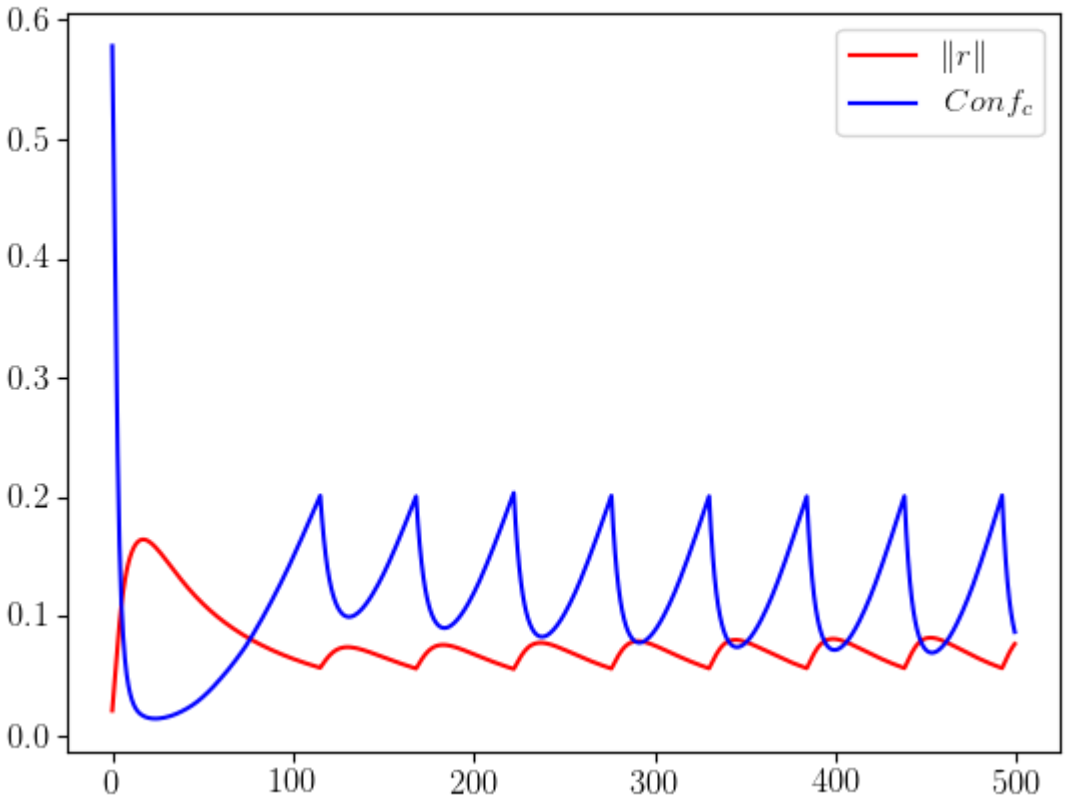
Les fonctions $Pred$ (en rouge) et $Conf_c$ (en bleu) évoluent alors de la manière suivante, en fonction du nombre d'étapes de descente de gradient effectuées :



Qualitativement, la norme de la perturbation augmente jusqu'à ce que $Conf_c$ passe en dessous de 0.9, à partir de quoi la norme diminue en gardant une valeur de $Conf_c$ stabilisée autour de 0.2.

Cette image peut être qualifiée de "difficile à attaquer" : il a été nécessaire d'augmenter très fortement la norme de la perturbation pour réussir à casser la prédiction du réseau, ce qui ne se produit qu'après un grand nombre d'étapes, et la norme finale de la perturbation est élevée.

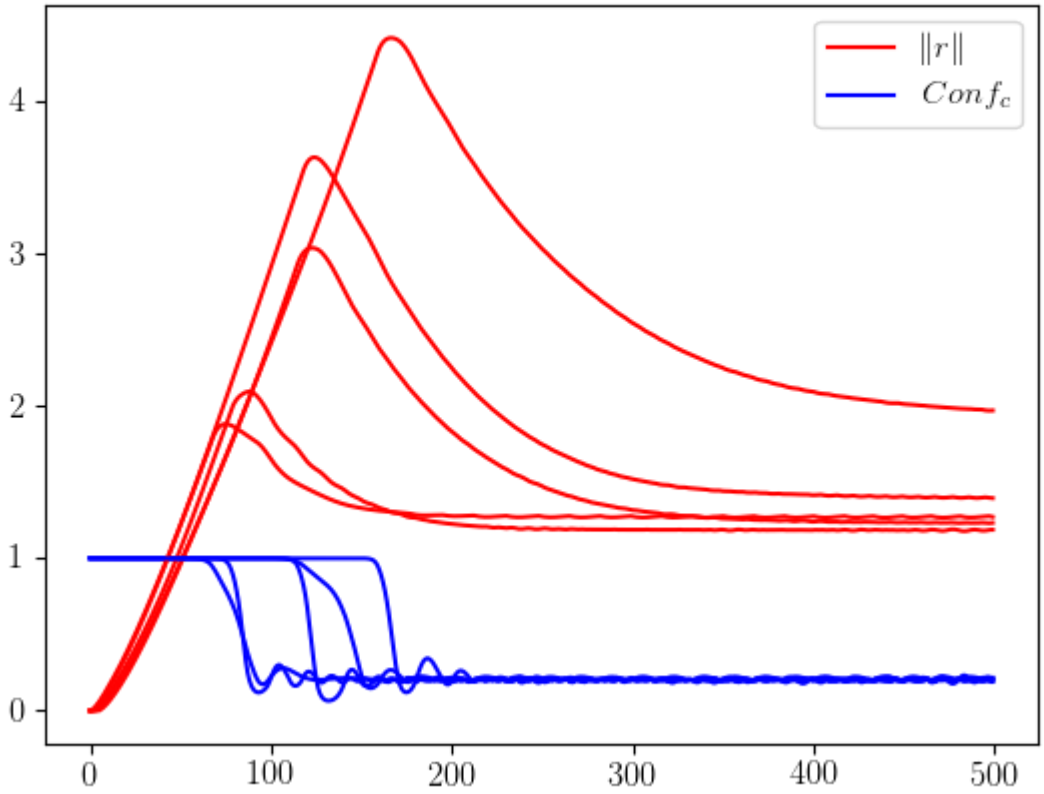
L'image suivante, Figure 2, peut au contraire être qualifiée de "facile à attaquer" : bien moins d'étapes ont été nécessaires pour casser la prédiction du réseau, la norme finale est très basse, et il n'y a pas eu de pic.



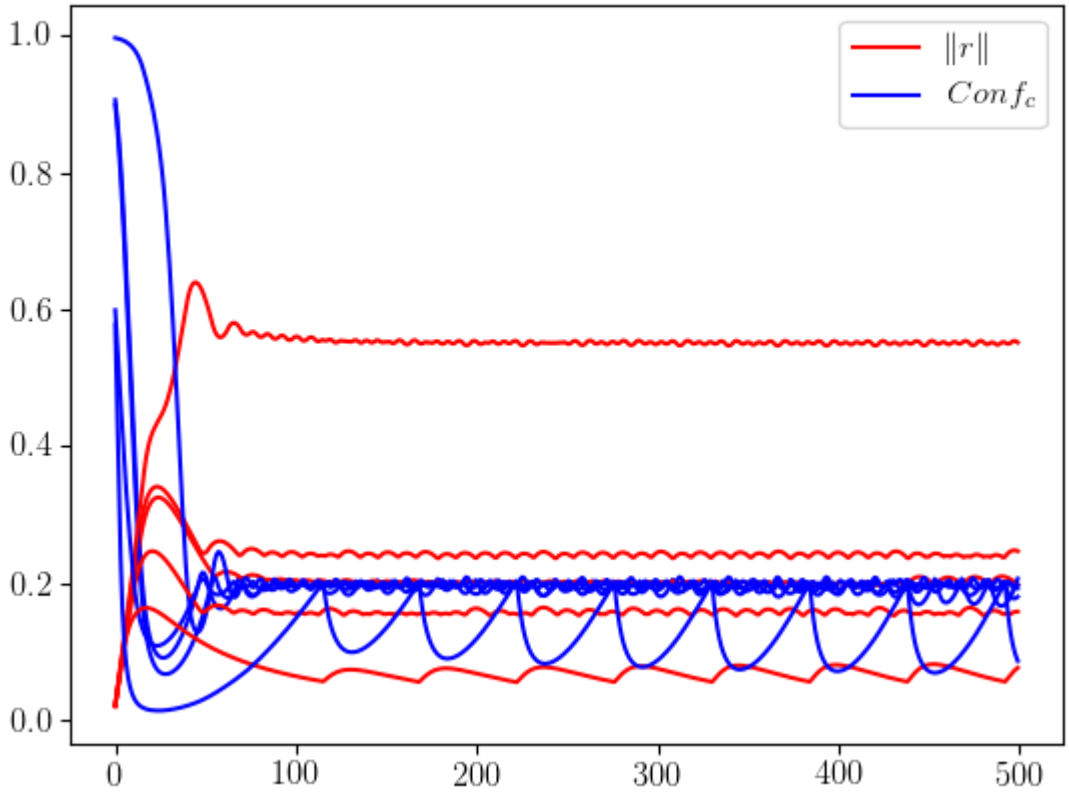
On voit nettement ici l'influence de la valeur du seuil à 0.2 dans la fonction $Loss$. Dès que $Conf_c$ est en dessous de 0.2, l'algorithme a pour seul objectif de réduire la norme de la perturbation, et fatalement $Conf_c$ repasse au dessus de 0.2. Il s'agit alors de réduire à la fois $\|r\|$ et $Conf_c$, jusqu'à ce que $Conf_c$ repasse en dessous de 0.2...

1.4 Un peu plus de résultats

La Figure 3 présente l'évolution de $\|r\|$ et de $Conf_c$ lors de l'attaque de 5 images "difficiles" à attaquer :



Puis Figure 4, avec 5 images "faciles" à attaquer.



Ici encore, on peut faire les mêmes observations :

	Images "faciles"	**images "difficiles"**
Pic	Absent ou faible	Haut
Étapes nécessaires	moins de $\mathbf{50}$	plus de $\mathbf{200}$
Norme de $\boldsymbol{r}$ finale	Faible	Élevée

Pour quantifier plus précisément cette difficulté à attaquer une image, introduisons le concept de *résistance*.

2. La résistance Res

2.1 La résistance à une attaque

Pour chaque image, on essaie de quantifier la *résistance* du réseau à une attaque adversaire. Plusieurs définitions sont possibles, par exemple la norme de la perturbation minimale mettant en échec le réseau :

$$Res_\infty(img) = min\{\|r\| \ ; \ Pred(img + r) \neq Pred(img)\}$$

(*Remarque : cette expression de la résistance n'est que d'un faible intérêt en pratique, car incalculable*)

On peut également utiliser comme valeur la norme finale obtenue après un certain nombre d'étapes dans l'attaque adversaire :

$Res_N(img) = \|Pert_N(img)\|$

Ou bien la hauteur du pic de la norme de la perturbation :

$Res_{max}(img) = max\{\|Pert_N(img)\| \ ; \ N \in \mathbb{N}\}$

Ou encore le nombre d'étapes qu'il a fallu pour abaisser $Conf_c$ à 0.2 :

$Res_{min}(img) = min\{N \in \mathbb{N} \ ; \ Conf_c(Pert_N(img)) < 0.2\}$

2.2 Une corrélation avec la fiabilité de la prédiction ?

Les images attaquées dans la partie **1.4** n'ont pas été choisies au hasard : celles Figure 3 sont les 5 premières de la base de donnée (classifiées correctement par le réseau) , et celles Figure 4 correspondent aux 5 premières erreurs de classification commises par le réseau.

Considérons un réseau de type **`AlexNet`** (CNN avec Dropout) appliqué au problème de la classification ds chiffres manuscrits de **`MNIST`**. Ce réseau est entraîné avec 60000 images, et sa performance évaluée sur 10000 images de validation. Sur ces dernières, toutes sauf 84 sont classifiées correctement par le réseau.

Étudions la répartition des valeurs des résistances $Res_{N=500}$ et Res_{max}, d'abord sur 200 images correctement classifiées (notées **V**), puis sur les 84 incorrectement classifiées (notées **F**).

Plage	**V -** Res_N	**F -** Res_N	**V -** Res_{max}	**F -** Res_{max}
$[0 \ , \ 0.2]$	0%	**41%**	0%	6%
$[0.2 \ , \ 0.6]$	3%	**48%**	1%	**68%**
$[0.6 \ , \ 1]$	**13%**	9%	3%	**19%**
$[1 \ , \ 2]$	**46%**	2%	**12%**	5%
$[2 \ , \ 5]$	**29%**	0%	**53%**	2%
$[5 \ , \ +\infty[$	9%	0%	**31%**	0%

Même étude sur la fonction Res_{min} :

Plage	**V -** Res_{min}	**F -** Res_{min}
$[0 \ , \ 10]$	0%	**12%**
$[10 \ , \ 50]$	4%	**77%**
$[50 \ , \ 100]$	**16%**	10%
$[100 \ , \ 200]$	**32%**	1%
$[200 \ , \ +\infty[$	**48%**	0%

Une corrélation se dessine nettement : les images correctement classifiées par le réseau sont très souvent de résistance bien plus élevée que les images sur lesquelles le réseau se trompe.

3. Les contre-attaques adversaires

On observe un autre phénomène : si une attaque adversaire cherche à tromper le réseau, une attaque adversaire sur une image incorrectement classifiée va, le plus souvent, produire une image qui sera correctement classifiée ! On parlera alors de *contre-exemple adversaire*.

Une contre attaque adversaire est donc une attaque adversaire sur une image incorrectement classifiée, dans l'espoir que la nouvelle catégorie soit la vraie.

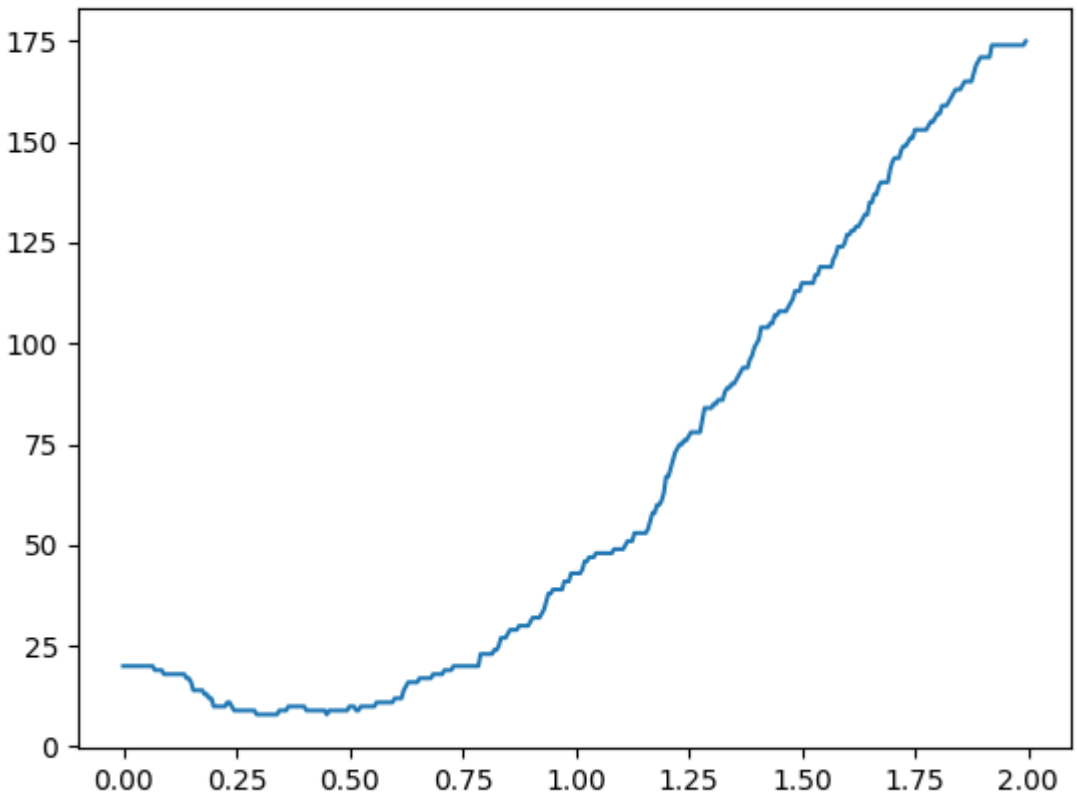
Toujours avec le même réseau, sur les 84 erreurs commises, 73 des contre-attaques adversaires donnent la bonne catégorie, soit dans 87% des cas !

4. Une méthode pour réduire l'erreur du réseau ?

4.1 Un premier résultat

Exploitons les deux phénomènes précédents pour tenter de réduire l'erreur commise par le réseau : On on détermine la résistance de chaque image du réseau. Si la résistance est supérieure à un certain critère, on considérera que la prédiction du réseau est correcte ; sinon on choisit comme prédiction le résultat de la contre-attaque adversaire.

Sur un lot de 270 images (250 justes, 20 erreurs), avec la fonction $Res_{N=500}$, on obtient le nombre d'erreurs commises en fonction du critère choisi.



Tout à gauche, un critère à 0 nous donne naturellement 20 erreurs, puisque l'on n'a rien modifié aux prédictions du réseau.

Plus intéressant, avec un critère à 0.45, le réseau ne commet plus que 8 erreurs !

4.2 Une méthode qui se généralise difficilement

En appliquant cette méthode à l'ensemble de 10000 images de validation de $\mathtt{MNIST}$, on ne réussit qu'à faire passer le nombre d'erreurs de 84 à 82 dans le meilleur des cas. Ceci s'explique simplement : le nombre d'erreurs est proportionnellement trop faible (moins de 1%), et donc les faux positifs, même si peu fréquents, vont annuler tout le gain obtenu.

Le choix arbitraire d'une fonction de résistance et d'un critère fixé n'est donc pas une méthode efficace dans ce cas.

4.3 Affinage de la méthode précédente

Plutôt que de s'arrêter à un critère fixé, on peut affiner nettement ce résultat par une régression linéaire sur des valeurs de $\|r\|$ au cours de l'attaque de chaque image.

(à compléter)

4.4 Un réseau de neurone pour calculer la résistance

(à compléter)

Bibliographie

[1] N. Srivastava, G. Hinton, A. Krizhevsky & al. JMLP, **Dropout: A Simple Way to Prevent Neural Networks from Overfitting.** Volume 15 (2014), Pages 1929-1958

[6] A. Krizhevsky, I. Sutskever & G. Hinton. NIPS'12 Proceedings, **ImageNet Classification with Deep Convolutional Neural Networks .** Volume 1 (2012), Pages 1097-1105