RDFIA TME 10-11
Visualisation des réseaux de neurones

Thomas Robert Taylor Mordan Remi Cadene Matthieu Cord

13 et 20 décembre 2017

Comptes rendus à rendre

Pour les TP sur les réseaux de neurones, vous devrez rendre :

— Un compte rendu de suivi dans la soirée après la séance (par mail à `taylor.mordan@lip6.fr` et `thomas.robert@lip6.fr`). Ce compte rendu contiendra en quelques lignes : (1) les réponses aux questions précédées par ★ ; (2) votre avancement dans le sujet, ce que vous avez réussi ou non ; (3) si vous voulez, des commentaires, remarques sur ce que vous avez compris ou non, etc.

— Un compte rendu global une semaine après la dernière séance (par mail), dont le but est de reprendre au propre le travail effectué pendant l'intégralité des séances de TP, en prenant du recul sur ce qui a été fait. Ce compte rendu représentera la majorité de la note de TP.

Les données et une version numérique du sujet sont accessibles
à l'adresse `http://webia.lip6.fr/~robert/teach-rdfia`

Introduction

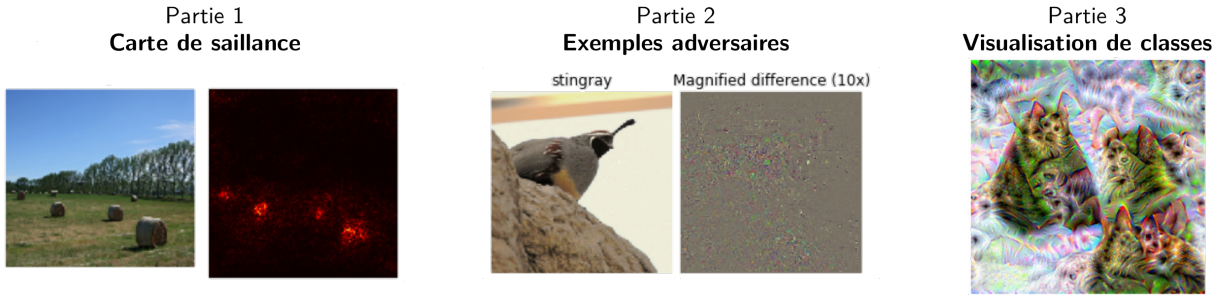

FIGURE 1 – Illustration des différentes parties

Dans ce TP, nous allons étudier diverses techniques publiées « récemment » dans le but d'étudier le comportement des réseaux de neurones convolutionnels. Le principe commun de ces approches est d'utiliser le gradient de l'image d'entrée par rapport à une classe de sortie.

Plus formellement, un réseau de neurones est une fonction mathématique f qui, à une image d'entrée, associe un **vecteur de scores** $\tilde{y}$ pour chaque classe :

$$\begin{array}{rccc} f: & \mathbb{R}^{h\times w\times c} & \to & \mathbb{R}^K \\ & x & \mapsto & \tilde{y} \end{array} \tag{1}$$

On applique généralement une activation softmax à ce vecteur de scores pour obtenir une distribution de **probabilité d'appartenance aux différentes classes** $\hat{y} = \mathrm{SoftMax}(\tilde{y})$.

Dans ce TP, le but n'est pas de modifier/apprendre le réseau mais de prendre un réseau appris, de figer ses poids et de modifier l'image d'entrée afin d'étudier indirectement le comportement du réseau. On utilisera dans un premier temps le réseau SqueezeNet de Iandola et al. (2016), appris sur ImageNet, conçu pour être très compact et efficace tout en permettant de bons résultats.

Nous verrons donc 3 façons d'étudier la fonction f en utilisant le gradient $\frac{\partial \tilde{\mathrm{y}}_i}{\partial \mathrm{x}}$ du score $\tilde{\mathrm{y}}_i$ en fonction de l'image d'entrée x :

— **Carte de saillance** *(saliency map)* : visualisation de l'importance de chaque pixel au score de la bonne classe, basé sur l'article de Simonyan et al. (2014).

— **Exemples adversaires** *(fooling images / adversarial samples)* : ajout de modifications mineures d'une image, imperceptibles pour l'humain, entrainant une mauvaise classification de l'image, basé sur l'article de Szegedy et al. (2014).

— **Visualisation de classes** *(Deep dream-like)* : "génération" d'une image corresponant à une catégorie, afin de visualiser le type de patterns detectés par le réseau, basé sur les articles de Simonyan et al. (2014) et Yosinski et al. (2015).

Instructions

Pour ce TP, un *notebook* Jupyter est fourni, c'est-à-dire un ensemble de blocs de texte et de code qui peuvent être édités et exécutés de façon interactive dans un navigateur. Pour le lancer en salle de TP, il suffit d'ouvrir un terminal dans le dossier dans lequel se trouve votre fichier `tme10-11.ipynb` et de faire `jupyter notebook`.

Sur votre machine, vous pouvez l'installer avec la commande `python3 -m pip install jupyter`.

Si vous n'êtes pas à l'aise avec le notebook, vous pouvez toujours utiliser les fichiers Python classique fournis.

Partie 1 – Carte de saillance

Une carte de saillance est une carte indiquant les zones importantes d'une image. Dans notre cas pour l'étude d'un CNN, il s'agit des pixels les plus importants pour prédire la bonne classe.

Pour produire ces cartes, Simonyan et al. (2014) proposent d'approximer le réseau de neurones f au voisinage d'une image x de classe i par une fonction linéaire : $\tilde{\mathrm{y}}_i = f_i(\mathrm{x}) \approx w_i^\top \mathrm{x} + b_i$ avec $w_i = \frac{\partial \tilde{\mathrm{y}}_i}{\partial \mathrm{x}}$.

Notez que les équations sont écrites pour un vecteur x mais en réalité x est bien sûr un tenseur 3D, tout comme w_i qui contient la dérivée du score de la classe de l'image d'entrée pour chaque pixel.

Pour produire la carte de saillance, Simonyan et al. (2014) proposent de prendre la valeur absolue du gradient w_i et de prendre pour chaque pixel la valeur maximale sur les 3 channels RGB pour produire une matrice 2D.

Questions

1. ★ Interprétez les résultats obtenus.
2. Discutez les limites de cette technique pour visualiser l'importance des différents pixels.
3. Cette technique pourrait-elle servir à autre chose qu'à interpréter le réseau ?
4. **Bonus :** Tester avec un autre réseau, par exemple VGG16.

Partie 2 – Exemples adversaires

Depuis quelques années, un aspect de l'étude des réseaux de neurones est l'étude des exemples adversaires *(fooling samples / adversarial samples)* ou encore attaques adversaires *(adversarial attacks)*. Il s'agit, compte tenu d'un réseau existant, et pour une image du jeu de test correctement classée par le réseau, de modifier le moins possible l'image mais de façon à ce que la nouvelle image soit désormais mal classée par le réseau.

En effet, Szegedy et al. (2014) montre pour la première fois l'existance de ce phénomène : il est possible, en modifiant très légèrement une image, de faire changer drastiquement la prédiction du réseau. Il a ensuite été davantage décrit par Goodfellow et al. (2014) puis par de nombreuses publications.

Note : Ce « domaine » d'étude des réseaux de neurones n'est pas à confondre avec les *Generative Adversarial Network (GAN)* dont le fonctionnement, s'il peut présenter quelques similarités, reste très distinct et différent du domaine des attaques adversaires.

Il existe de nombreuses façons de créer ces images, nous allons ici nous intéresser à une approche très simple et naïve (mais déjà assez efficace). Compte tenu d'une image x correctement classée dans la classe i, nous voulons désormais qu'elle soit classée dans la classe j en modifiant l'image et non le réseau.

Pour cela, nous allons réaliser une remontée de gradient sur le gradient $g = \frac{\partial \tilde{\mathrm{y}}_j}{\partial x}$ du score $\tilde{\mathrm{y}}_j$ de la classe j en fonction de l'entrée x.

On modifiera progressivement x selon la règle suivante :

$$\mathrm{x} \leftarrow \mathrm{x} + \eta \frac{g}{||g||_2} \qquad \text{avec } \eta \text{ le learning rate.} \tag{2}$$

Appliquer cette règle itérativement sur l'image voulue jusqu'à ce que la classe prédite soit désormais la classe j.

Questions

5. ★ Interpréter les résultats obtenus.
6. Quelles conséquences cela peut-il avoir pour l'utilisation de réseaux de convolution en pratique ?
7. **Bonus :** Discutez des limites de cette façon naïve de construire des images adversaires. Proposez éventuellement des alternatives / extensions. (Vous pouvez vous inspirer de la littérature)

Partie 3 – Visualisation de classes

Afin de visualiser le type de patterns succeptible de produire une classe particulière en sortie du réseau, et donc afin d'étudier indirectement ce que détecte le réseau, Simonyan et al. (2014) propose une approche simple, complétée par Yosinski et al. (2015).

Le principe est assez simple, considérant une image initiale x (par exemple simplememt un bruit aléatoire), on veut modifier cette image de façon à maximiser le score de la classe i. Pour améliorer la qualité des résultats, on propose d'utiliser un certain nombre de techniques de régularisation. Tout d'abord, on ajoutera la norme L2 de l'image comme régularisation lors du calcul du gradient. Par ailleurs, on appliquera régulièrement des régularisations implicites à l'image sur laquelle on est en train de travailler, par des translations et flous (déjà implémenté pour vous dans le notebook).

Formellement, pour une image x que l'on veut modifier pour la classer dans la classe i, on maximisera la loss :

$$\mathcal{L} = \tilde{\mathrm{y}}_i - \lambda ||\mathrm{x}||_2. \tag{3}$$

On modifiera progressivement x selon la règle suivante :

$$\mathrm{x} \leftarrow \mathrm{x} + \eta \nabla_{\mathrm{x}} \mathcal{L} \qquad \text{avec } \eta \text{ le learning rate.} \tag{4}$$

Questions

8. ★ Intepréter les résultats.
9. Essayer de varier le nombre d'itérations et le learning rate, le poids de la régularisation.
10. Essayer d'utiliser une image d'ImageNet comme image source au lieu d'une image aléatoire (paramètre `init_img`). Vous pouvez utiliser pour classe cible la classe réelle. Commenter l'intérêt.
11. **Bonus :** Essayer avec un autre réseau, par exemple VGG16.

Références

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv :1412.6572*, 2014.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet : Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv*, 2016.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks : Visualising image classification models and saliency maps. *ICLR Workshop*, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *ICLR*, 2014.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *ICML Workshop*, 2015.