Abdelazer

DNNViz : Introspection de réseaux neuronaux profonds

12 février 2019

Table des matières

Bonjour à toutes et à tous,

Je publie pas très souvent des trucs par ici et du coup, pour une fois, pourquoi pas ? Je viens de finir mon master en *computer sciences* à l'université de Fribourg et si je vous dis ça c'est pas pour vous raconter ma vie, mais parce que c'est un peu l'objet de ce billet ;).

J'ai fait mon travail de master dans le domaine de la *data visualization* appliquée aux modèles de *deep learning* . Je vais probablement publier un papier sur le sujet dans les mois qui viennent, mais en attendant, je vais montrer un peu ce que j'ai fait, parce que je trouve ça cool et que ça me ferai plaisir d'avoir des retours d'utilisations.

Donc le projet consiste globalement à faire un outil de visualisation (un programme quoi) qui permet d'essayer de mieux comprendre ce qui se passe dans un réseau neuronal profond. C'est un sujet super vaste, d'autres outils existent, on sait pas trop si on peut vraiment expliquer des choses dans le domaine grâce à la visualisation. En bref, c'est assez passionant.

tl ;dr
L'outil est cool, vous pouvez tester par vous-même.

Pour ceux qui ne peuvent pas attendre :

Contenu masqué n°1

1. Concrètement, de quoi parle-t-on ?

Je vais pas faire un article sur qu'est-ce que c'est la *data visualization* et le *deep learning*. Ce serait trop long, trop chiant et il y a beaucoup de ressources sur internet.

> Si vous savez ce que veulent dire ”deep learning” et ”data visualization”, vous pouvez allègrement sauter cette section.

Donc brièvement, le **deep learning**. C'est un bon gros mot à la mode actuellement. Ça fait parti du *machine learning* et plus généralement de l'intelligence artificielle. En très (très) gros, c'est des techniques, qui permettent actuellement de résoudre des tâches très difficiles à traiter avec de l'informatique classique. Genre la reconnaissance d'image, le NLP (reconnaissance de texte) mais aussi la prise de décision dans des jeux (salut à toi, jeu de Go).

Le deep learning, c'est pleins de neurones (neurones informatiques, c'est un modèle pas très fidèle de nos bons vieux neurones) organisés en couches (on parlera plus volontiers de *layers*) qui, au terme d'un (relativement) long apprentissage, finissent par organiser l'information qu'ils contiennent et réussissent plutôt bien à ”inférer” des résultats. Le truc, c'est que si on commence à comprendre de mieux en mieux comment ça marche à l'intérieur, il faut bien le dire, on capte encore pas grand chose. La recherche est en partie basée sur des essais/erreurs et on peut souvent entendre le terme *black box* (boîte noire) pour définir l'intérieur du réseau neuronal.

Et la **data visualization** dans tout ça ! Hein ? Alors l'idée de la visualisation de données, c'est de prendre plein de données (mais genre vraiment beaucoup) et d'essayer de les représenter pour qu'on puisse, éventuellement, y voir des *patterns*. (De nouveau c'est une explication au bulldozer.)

Et du coup. Comme le *deep learning*, au final, c'est beaucoup de données. Et que la *data viz*, ça bosse avec beaucoup de données. Les deux devraient plutôt bien s'entendre.

Comme je l'ai dit dans l'introduction, il y a déjà pas mal de recherche dans ce domaine et plusieurs outils intéressants sont disponibles si vous voulez vous amuser. On peut citer bien sûr Tensorboard[1] ou ActiVis[2] qui sont des *dashboards* de visualisations. Mais voici également deux outils que je trouve très intéressants : ReVACNN[3] et DeepViz[4].

J'ai fait un état de l'art beaucoup plus approfondi dans mon travail et si quelqu'un est vraiment intéressé par le domaine, je pourrais lui en passer une copie (en anglais). Bref, voilà pour le contexte.

2. DNNViz : on rentre dans le dur !

Concrètement, mon projet est un outil qui va permettre de visualiser interactivement les **activations des neurones** pour des modèles de classification d'images (ça *devrait* pouvoir être générique et marcher pour n'importe quel type de réseau neuronal, mais c'est pas sûr). En très bref (à nouveau, et j'en suis désolé, mais le sujet est quand même cossu) quand on donne en

1. tensorflow.org/guide/summaries_and_tensorboard
2. Minsuk Kahng, Pierre Y. Andrews, Aditya Kalro, and Duen Horng Chau. ActiVis : Visual Exploration of Industry-Scale Deep Neural Network Models. arXiv :1704.01942 [cs, stat], April 2017. arXiv : 1704.01942.
3. Sunghyo Chung, Cheonbok Park, Sangho Suh, Kyeongpil Kang, Jaegul Choo, and Bum Chul Kwon. ReVACNN : Steering Convolutional Neural Network via Real-Time Visual Analytics. page 8.
4. Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding Neural Networks Through Deep Visualization. arXiv :1506.06579 [cs], June 2015. arXiv : 1506.06579.

entrée une image à un modèle (un réseau neuronal), certains neurones vont ”s'allumer” plus que d'autres. C'est ce qui fait qu'en fonction d'un *input*, un réseau va prédire un *output*.

Des outils de ce genre existent déjà, évidemment. Mon travail a porté sur différents points qui me semblaient prometteurs. L'outil permet notamment de visualiser :

— la **structure générale de réseaux neuronaux (très) profonds** (genre les derniers modèles de la littérature, type ResNet, GoogleNet, DenseNet par exemple), alors que la plupart des outils précédents se cantonnent à des réseaux plus ”classiques” et petits (la faute à la grande quantité d'info à afficher) ;
— l'**évolution de l'état interne du réseau** durant l'entraînement (là, il me semble, c'est totalement nouveau) ;
— les activations selon un *input* (une image en entrée) mais aussi selon des *inputs* **agrégés par classes** ou même agrégés sur l'ensemble du *dataset* ;
— une comparaison de différents *inputs* entre eux.

Plus spécifiquement, mon outil se décompose en deux parties :

— Le **backend**, une *task* DeepDIVA (j'explique juste après) qui permet, soit de capturer les activations d'un modèle entraîné d'après un *dataset* donné, soit de capturer les activations à intervalle régulier pendant l'entraînement d'un modèle d'après un *dataset* donné ;
— Le **frontend**, l'outil de visualisation proprement dit, qui consomme l'*output* produit par le *backend* (un dossier qui contient du `JSON`) et qui permet une exploration interactive de la visualisation.

Voilà (et c'est déjà pas mal).

2.1. Un mot sur les outils utilisés

Et oui ! Il y a pleins de développeurs ici

2.1.1. DeepDIVA

DeepDIVA [5] est un projet open-source de *framework* pour du *deep learning*. En fait, il est basé sur PyTorch (un framework de deep learning en python). C'est une sorte de *framework* pour un *framework*. Ça a été développé dans le même labo que mon projet Il a pas mal d'avantage, mais notamment, il propose : des modèles pré-écrits, un système de *tasks* modulaires et surtout, il a été conçu pour pouvoir reproduire des expériences de deep learning (basé sur l'usage de *seeds* explicites).

Le *backend*, vous l'aurez compris, est une *task* dans DeepDIVA.

2.1.2. ElectonJS + VueJS

Là-dessus, je fais vraiment court. La visualisation en elle-même est faite sur ElectronJS et VueJS. (Oui, c'est plutôt court.)

3. Ça ressemble à quoi ?

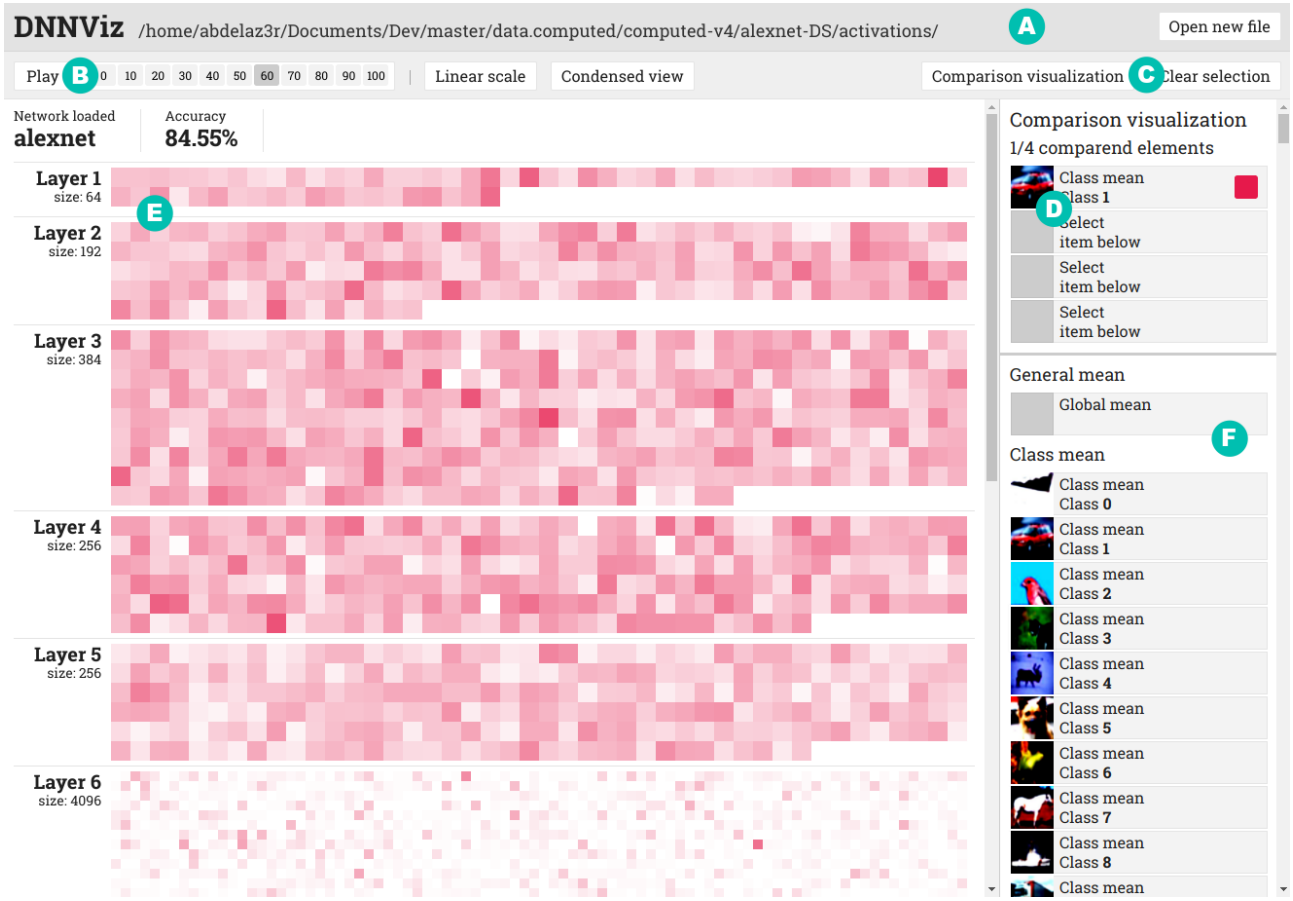


Figure : DNNViz. **A** Panneau des fichiers ouverts, **B** Panneau du contrôle de la visualisation, **C** Panneau de sélection du mode de visualisation, **D** Panneau de la sélection, **E** Visualisation, **F** Panneau de la liste des *inputs*.

Bon, laissez-moi vous expliquer un peu ce qu'on voit.

Ici, on visualise l'agrégation de tout les *inputs* de la classe 1 (automobile) d'un réseau neuronal AlexNet entraîné sur CIFAR-10 . Plus précisément, on visualise l'état des activations pour cet *input* à l'*epoch* 60 (l'entraînement va de l'*epoch* 0 à l'*epoch* 100). Sur la visualisation proprement dite, on peut voir la liste des *layers* (couches) du modèle. Chaque carré de couleur est, dans le cas d'une couche convolutionelle, la compression d'une *feature map*, dans le cas d'une couche classique, un neurone. Un carré presque blanc représente une activation moyenne (je rappelle qu'il s'agit de l'agrégation d'un ensemble d'*inputs*) presque nulle alors qu'un carré tout à fait rouge représente une activation moyenne très forte.

5. Michele Alberti, Vinaychandran Pondenkandath, Marcel Wrsch, Rolf Ingold, and Marcus Liwicki. DeepDIVA : A Highly-Functional Python Framework for Reproducible Experiments. arXiv :1805.00329 [cs], April 2018. arXiv : 1805.00329.

Bon, j'espère que vous avez bien tout suivit. Parce que maintenant, on rentre dans le dur !

4. On peut voir quoi au final ?

Comme ce billet commence déjà à être trop long, je ne vais pas présenter en détail les possibilités offertes par l'outil. Je vais juste vous montrer deux cas intéressants avec des résultats en images

4.1. Network pruning

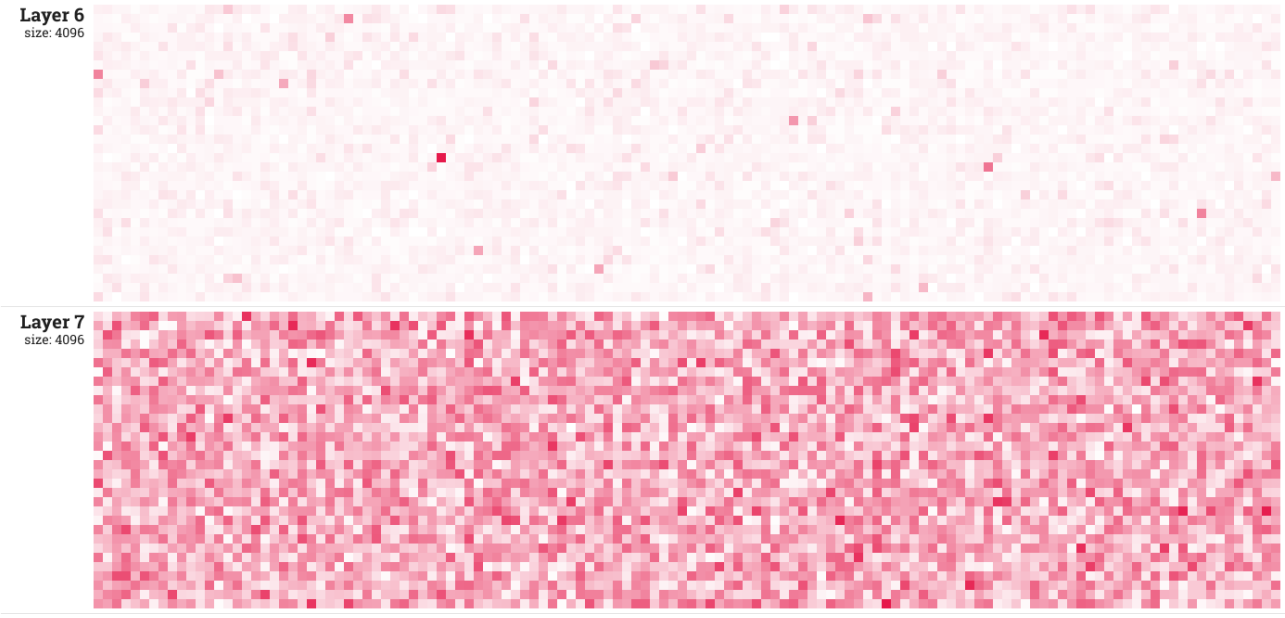
Donc. Le *network pruning*. Grosso modo, c'est le fait de "compresser" un réseau neuronal en enlevant les neurones qui ne servent à rien (après ou pendant l'entraînement). C'est une technique qui permet de réduire la taille (et donc la complexité) d'un modèle.

> Removing unimportant weights from a trained network to improve generalization, simplify networks, reduce hardware or storage requirements, increase the speed of further training, and in some cases enable rule extraction.[6]

C'est quelque chose d'assez connu. Juste en dessous, je vous montre la visualisation **générale** (c'est-à-dire que l'*input* affiché correspond à la moyenne de TOUTES les images du *dataset*). Ce genre de visualisation permet justement de voir, en quelques sortes, l'occupation du réseau. Dans un réseau dont l'information est bien répartie, presque tous les neurones d'une couche devraient (dans le cas de cette visualisation, générale donc) avoir une activation moyenne haute (visuellement ça voudrait dire des couches "très rouges"). Après bien évidemment ça va dépendre d'autres facteurs, de la nature du *dataset* notamment.

Le modèle proposé ici est AlexNet. Un réseau qui comprend 5 couches convolutionelles suivies de 2 couches *fully connected* (totalement connectées). Or, on sait que ces deux dernières couches contiennent, dans ce cas, trop de neurones, et alourdissent l'apprentissage.

L'image ci dessous montre les couches 6 et 7 de modèle, à la 200ème *epoch*.



4. On peut voir quoi au final ?

On peut voir que le *layer* 6 est totalement inutile. Quelques neurones sont hyper-activés et le reste est mort. Ci-dessous, sur un autre réseau, vous pouvez voir l'évolution sur 200 *epochs* des couches 4, 5 et 6.

https://asylamba.com/public/media/files/source

4.2. Adversarial Attack Detection

Alors. Tout d'abords, un peu de théorie.

Une *adversarial attack*, c'est la manipulation d'une image, par exemple, de telle sorte que cette manipulation ne soit pas détectable par un être humain, mais qu'elle change la classification donnée par le réseau neuronal.

Pour montrer vite fait les implications que ça pourrait avoir. Imaginons qu'on crée une image d'un signal routier qui, pour nous, ressemble comme 2 gouttes d'eau à un vrai signal routier (genre "limitation de vitesse à 20 km/h") mais qui, pour un algorithme d'IA embarqué dans, mettons, une voiture autonome, signifie "limitation de vitesse à 120 km/h". Je vous laisse imaginer le problème.

Si dessous, vous pouvez voir, à gauche, une image correctement classifiée (temple maya ou quelque chose dans ce genre), au centre, la différence entre les deux images et à droite, l'image truquée que le modèle classifie mal.

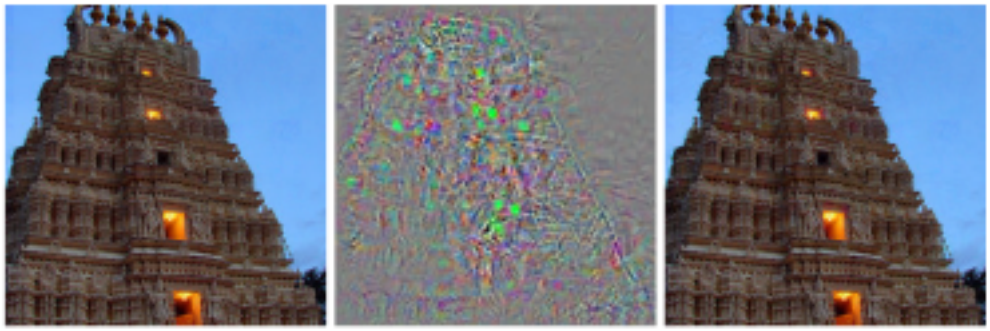


Figure : Exemple d'*adversarial images* générée pour un modèle AlexNet[7]

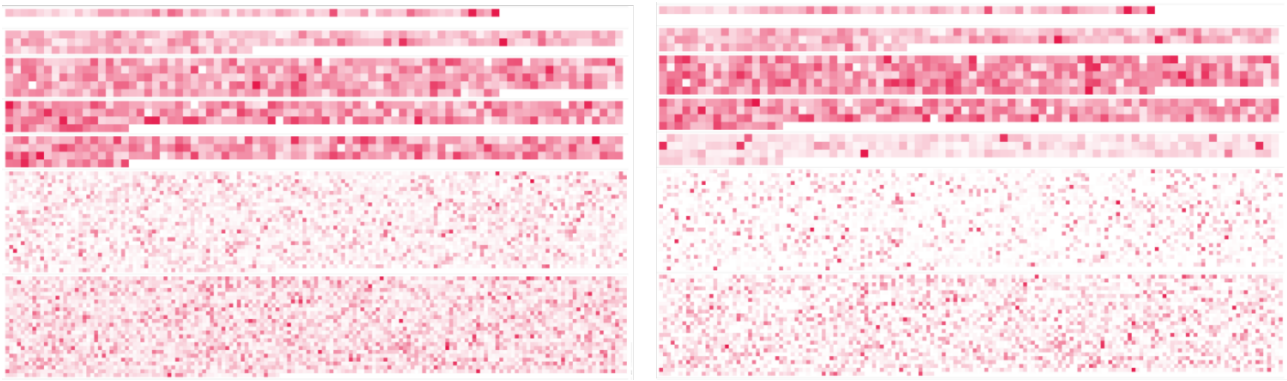
4.2.1. Data tempering

Mes collègues de labo ont bossé sur le *dataset tempering* (l'altération de *dataset*)[??]. Grosso modo, c'est une procédure pour faire une modification sur toute une classe du *dataset*. La modification n'est pas détectable avec les métriques classiques (accuracy, confusion matrices, ...) et permet de corrompre des *datasets* qui, par la suite, peuvent être laissé en libre accès sur internet.

4. On peut voir quoi au final ?

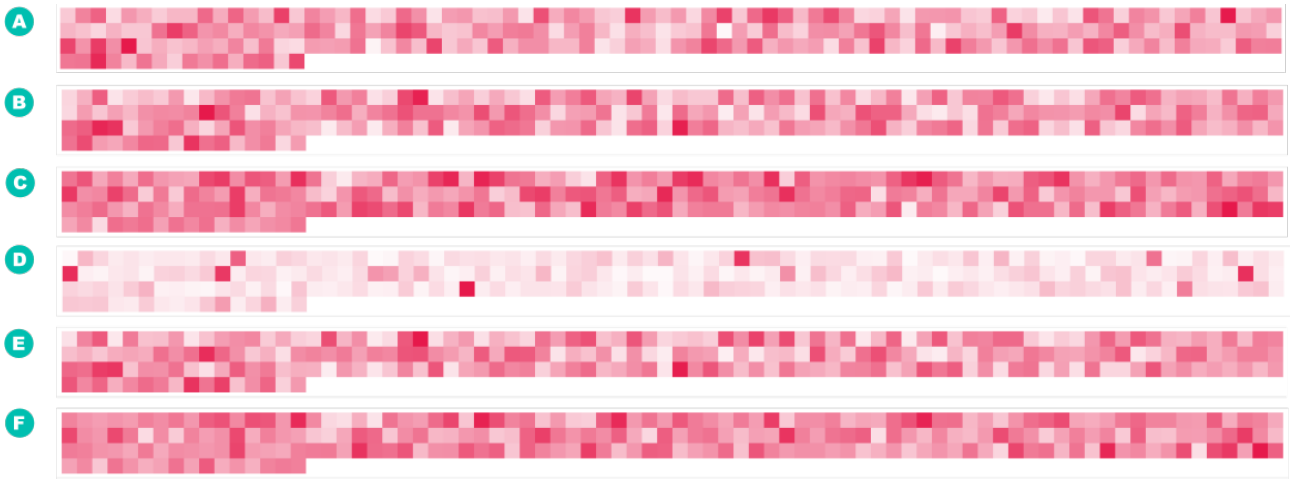
On a fait une petite expérience. Ils m'ont donné 2 *datasets* et m'ont dit d'entraîner 2 réseaux avec exactement les mêmes paramètres. Et puis ensuite de voir comment ça se voyait sur l'outil de visualisation.

Cette image montre la visualisation de la classe 1 sur l'ensemble du réseau neuronal, totalement entraîné. La moitié gauche montre le réseau entraîné sur le premier *dataset*, la partie droite, le réseau entraîné sur le deuxième *dataset*.



Vous n'avez rien vu ?

L'image suivante montre, suivant les mêmes conditions, le *layer* 5. **A**, **B**, **C** montrent respectivement la classe 0 (airplane), 1 (automobile) et 2 (bird) sur le premier dataset. **D**, **E**, **F** montrent la même chose, mais sur le deuxième *dataset*.



? Selon vous, quel est le *dataset* altéré et, plus précisément, quelle classe est altérée ?

6. B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal Brain Surgeon and general network pruning. In IEEE International Conference on Neural Networks, pages 293–299 vol.1, March 1993.

7. Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You ?" : Explaining the Predictions of Any Classifier. In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, pages 1135–1144, New York, NY, USA, 2016. ACM.

Bon, ce billet est finalement beaucoup trop long. Alors je m'arrête là. Je mettrais probablement le lien de l'article quand il sortira (s'il sort) si certains d'entre vous sont intéressés.

Pour ceux qui pratiquent le deep learning au quotidien (j'imagine qu'il y en a), vous pouvez trouver toutes les infos sur le site de DeepDIVA . Malheureusement, il n'y a pas encore de tutoriel pour utiliser la *task*. Sachez que cette dernière s'appelle `process_activation`. Je vous mets ici deux exemples de commande à lancer dans DeepDIVA pour capturer les activations et produire l'output utilisable.

```
# Train and process
python template/RunMe.py --runner-class process\_activation
    --output-folder \_log --dataset-folder _dataset/CIFAR10
    --model-name alexnet --ignoregit --train --epochs 200
    --process-size 200 --process-every 25 --save-images

# Process only
python template/RunMe.py --runner-class process\_activation
    --output-folder \_log --dataset-folder _dataset/CIFAR10
    --model-name CNN_basic --load-model _models/cnnbasic.pth.tar
    --ignoregit --process-size 500 --save-images
```

Les sources de l'outil de visualisation sont ici github.com/DIVA-DIA/DeepDIVA-DNNViz et des *packages* déjà prêt pour les différents OS sont aussi disponible.

Voilà, c'est encore un peu le chantier. Il y a encore des bugs. Mais c'est déjà marrant.

Bonne soirée à vous

Contenu masqué

Contenu masqué n°1

Contenu masqué

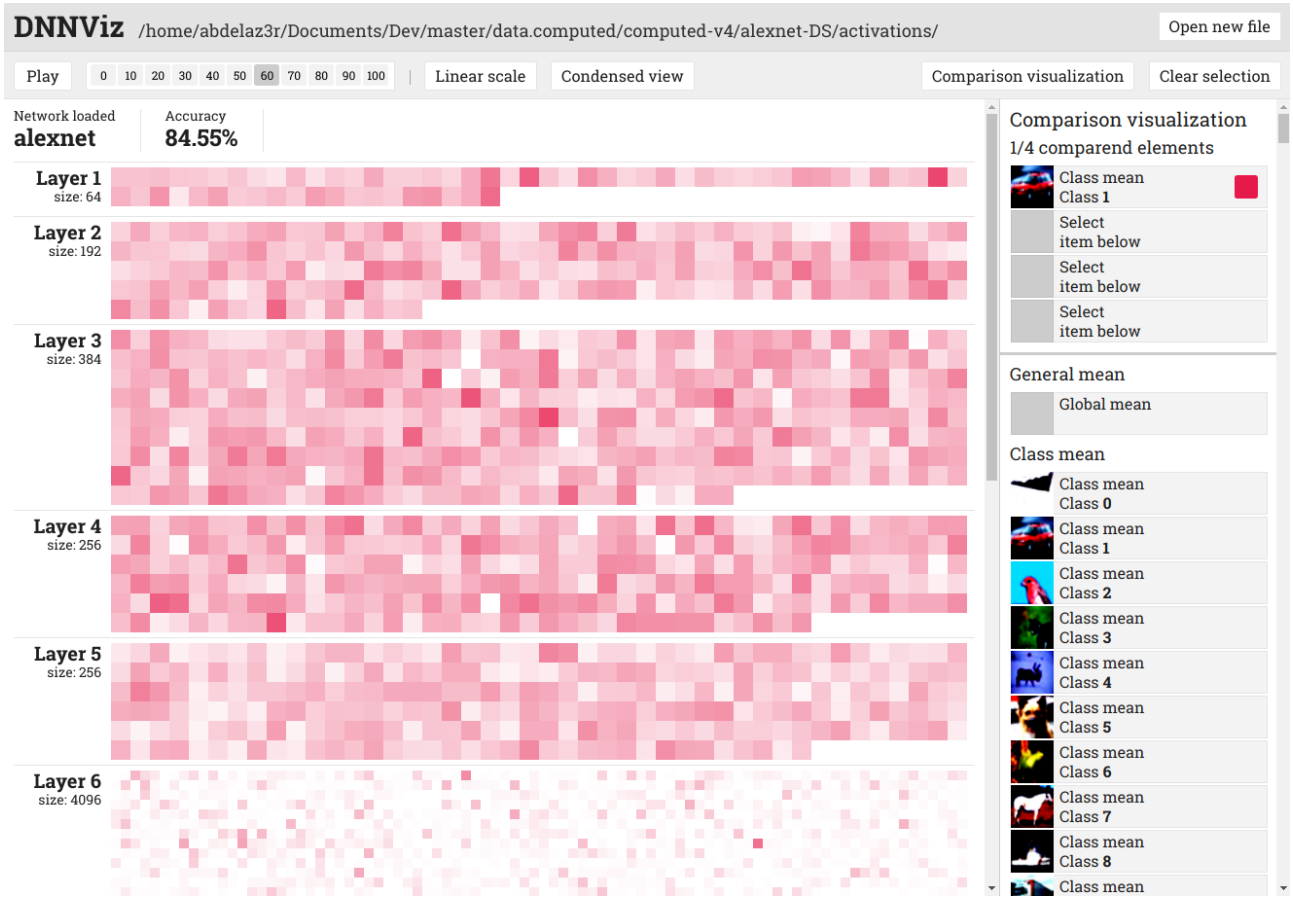


FIGURE 4. – DNNViz

Retourner au texte.