

SORBONNE UNIVERSITÉ SCIENCES

RAPPORT RDFIA DES TMEs 8, 9, 10 ET 11

---

## Réseaux de Neurones

---

MASTER 2 DONNÉES, APPRENTISSAGE ET CONNAISSANCES

---



***OUARDA FENEK  
SARA YASMINE OUERK***

2019 - 2020

# 1 Réponses aux questions du TME 8

1. Estimation du nombre de paramètres du VGG :

$$Nb\_params = [7 * 7 * 512 * 4096 + 4096] + [4096 * 4096 + 4096] + [4096 * 1000 + 1000]$$

2. La taille de la sortie du VGG est 1000, elle correspond aux nombre de clusters différents. On aura pour chaque entrée sa probabilité d'appartenance à chacune de ces classes.

3. Application du réseau sur plusieurs images :

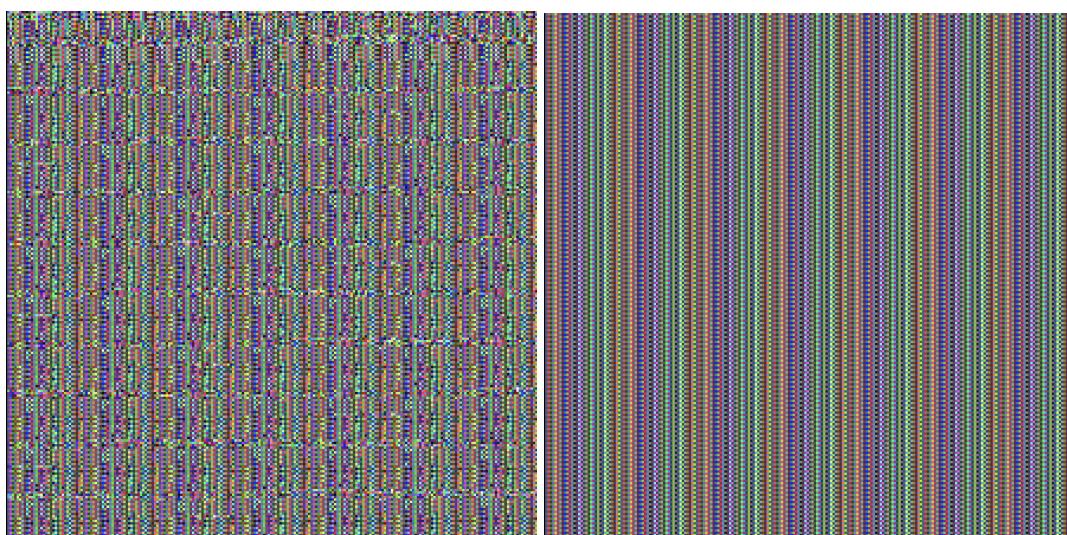
A priori les images sont bien classées.

Par exemple pour une image d'un chat et une autre d'un lion, on a eu respectivement, les classes **Egyptian Cat** et **Persian Cat**.



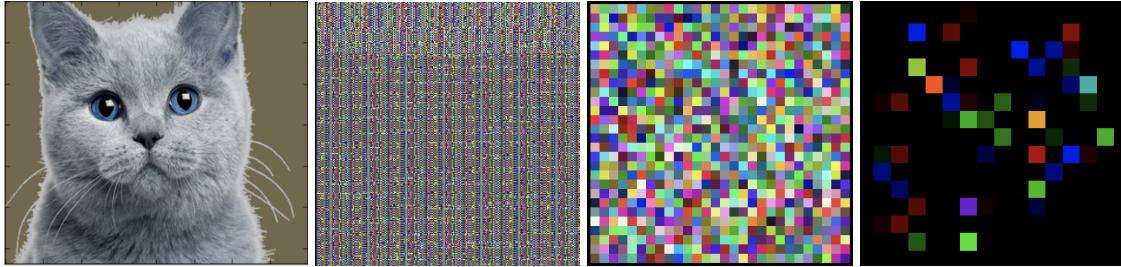
L'image du lion est certes en noir et blanc mais elle est en RGB. Sinon le format ne serait pas adapté vu que le vgg s'attend à avoir 3 channels en entrée.

4. En Appliquant seulement la première couche de convolution du VGG16 sur les images du chat et du lion montrées précédemment, les résultats étaient, respectivement, comme suit :



A cette étape, les images ne sont pas très interprétables, car lors des premières couches, le réseau extrait les concepts généraux.

Prenons une autre image et voyons comment cette extraction évolue au fur et à mesure des convolutions.



On passe l'image tout à gauche au réseau, l'image suivante est le résultat après la première convolution, celle d'après est le résultat après la 16ième convolution, et enfin, le résultat après la dernière convolution du VGG. Au début, on voit qu'il n'y a rien d'interprétables à l'oeil nu, mais à la dernière convolution, la tête du chat s'est bien formée, on pourrait reconnaître les oreilles et et les yeux par exemple, parce qu'à cette étape là, le réseau a extrait les concepts les plus abstraits qui permettent de reconnaître le chat.

5. On n'entraîne pas VGG directement sur 15 Scene parce que cette dernière a un nombre de données relativement petit par rapport à la taille du VGG qui est considéré comme étant un grand réseau.
6. Le pré-apprentissage sur ImageNet est considéré comme un bon savoir, on a déjà appris un bon nombre de features qui a été prouvé pertinent, ça serait bien de pouvoir exploiter ce savoir sur d'autres tâches.
7. Limites du Transfer Learning :

- Actuellement, l'une des plus grandes limites au transfert d'apprentissage est le problème du transfert négatif. L'apprentissage par transfert ne fonctionne que si les problèmes initiaux et cibles sont suffisamment similaires pour que le premier cycle d'apprentissage soit pertinent. Les développeurs peuvent tirer des conclusions raisonnables sur le type d'apprentissage considéré comme «suffisamment similaire» à la cible, mais l'algorithme n'a pas à être d'accord. Si le premier cycle d'entraînement est trop éloigné de la cible, le modèle peut en fait être moins performant que s'il n'avait jamais été entraîné du tout. À l'heure actuelle, il n'existe toujours pas de normes claires sur les types d'apprentissage qui sont suffisamment liés ou sur la manière de les mesurer.
- A l'utilisation du transfert d'apprentissage, les données d'entraînement doivent avoir deux options. Tout d'abord, la distribution des données de train que le modèle pré-entraîné a utilisé doit être similaire aux données auxquelles nous allons faire face pendant le test ou du moins ne pas trop varier. Deuxièmement, le nombre de données d'entraînement pour l'apprentissage par transfert doit être tel que le modèle ne soit pas surchargé.
- On ne peut pas supprimer des couches en toute confiance pour réduire le nombre de paramètres. Si on supprime des couches convolutives des premières couches, on n'aura pas un bon apprentissage en raison de la nature de l'architecture qui trouve des fonctionnalités de bas niveau. De plus, si on supprime les premières couches, on aura des problèmes pour les couches fully connected, car le nombre de paramètres entraînables change de cette façon. Les couches connectées et les couches convolutives profondes peuvent être de bons points pour une réduction, mais il peut prendre du temps pour trouver combien de couches et de neurones doivent être diminués afin de ne pas sur-apprendre.

8. L'influence des couches d'extraction :

Un réseau de neurones profond entraîné pour reconnaître les gens -par exemple- à partir d'un grand ensemble d'images, montrera un certain nombre de caractéristiques dans ses couches. Et dès lors les premières couches, ces caractéristiques deviennent de plus en plus complexes et abstraites. En vision par ordinateur, une telle complexité va des pixels, des taches, des yeux,

du nez, des visages, jusqu'aux vêtements et à des scènes entières. Bien sûr, des neurones spécifiques s'activeront pour chacun de ces concepts abstraits.

Un autre classificateur - par exemple des animaux, des fleurs ou des véhicules - utilisera le même encastrement en particulier dans les premières couches. Après tout, les taches de pixels, de segments et d'autres fonctionnalités de bas niveau sont généralement conservées dans tous les domaines.

Donc si on s'arrête à un transfert des premières couches seulement, on aura repris les concepts généraux. Ces conceptes deviendront de plus en plus complexes et abstraits en avançant dans les couches.

9. Les images de 15 Scene sont en noir et blanc, alors que VGG16 attend des images RGB, pour remédier à celà, on duplique le channel noir et blanc sur 3 channels pour avoir une représentation similaire au systèe RGB.

10. Résultat obtenu avec un SVM ( $batch\_size = 4$ ,  $C = 1.0$ ) :

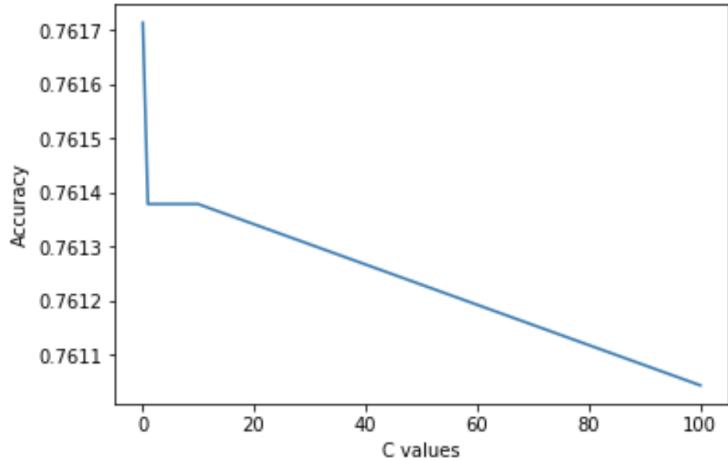
SVM accuraccy : **0.7771084337349398**

En reprenant les résultat qu'on a obtenu avec l'approche **BoW**, le meilleur résultat trouvé en test était de **0.58** pour un  $C = 7.0$ . On voir donc, qu'avec l'estraction de features on a remplacé toute l'étape de la construction du BoW sans avoir à ré-apprendre le modèle et on a eu des meilleurs résultats en classification.

11. Plutôt que d'apprendre un classifieur indépendant, il serait possible d'utiliser le réseau de neurones lui même et celà en faisant du **Fine Tuning**. Prendre le réseau pré-entraîné et l'entrainer un peu plus sur la tache en question. Ainsi les paramètres ne démareront pas de l'aléatoires et nous aurons une convergence plus vite. On pourrait également appliquer une autre méthode en utilisant le réseau en entien qui est **Knowledge Distillation**, mais cette dernière nécessite l'apprentissage d'un modèle en parallèle.

12. Pour aller plus loin :

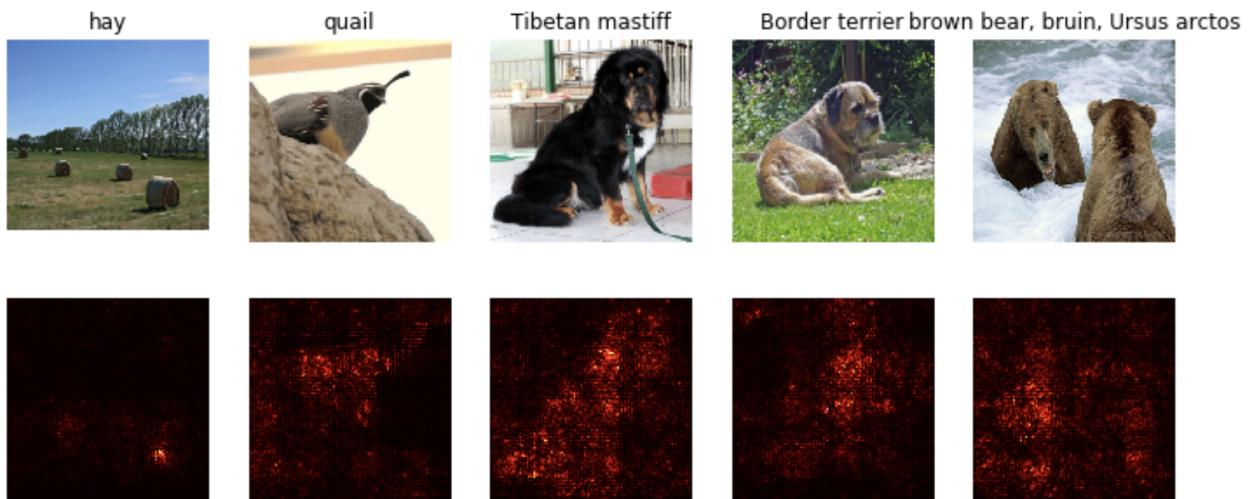
- On a encore retiré une autre couche du classifieur, et on a eu en résultat une accuracy de **0.7623828647925034**. On a perdu par rapport à l'extraction précédente. En fait, le plus on extrait de couches, le plus on extrait et on transfère des concepts plus complexes et plus abstraits. Le fait d'avoir retiré une couche nous a fait perdre certaines spécifications. Ce que l'on peut dire est que si on est sur deux bases dont les concepts de ressemblent, humains et animaux par exemple, il serait bien de garder une grande profondeur du réseaux car on a même concepts qui se répètent, voire, yeux, nez, oreilles, pieds ... Mais si on est sur deux bases totalement différentes, humains et objects par exemple, il serait mieux d'enlever une partie du réseau, car aux dernières couches, celui-ci aurait appris à caractériser très précisément l'humain, et il serait assez lourd pour le classifieur ajouté après de revenir sur les objects.
- On a appliqué une PCA sur les données d'entraînement avant de les passer au classifieur, celà a permis une exécution plus rapide, parce qu'on a moins de dimensions, les calculs matriciels se font plus rapidement. Mais on a quand même perdu en termes d'accuracy. Avec une **PCA à 2 dimensions**, on a eu une accuracy de **0.72**. Celà s'explique par le fait qu'en réduisant les dimensions spatiales, on a peut être perdu certaines caractéristiques clés qui auraient servi pour la classification des images. La PCA garde les axes les plus importants, mais des fois, l'information clé se trouve dans ce qui a été enlevé. C'est la malédiction de la dimension.
- On a fait varier les valeurs du paramètre **C** du SVM.



Le degré d'influence du paramètre  $C$  est de l'échelle du 1000ième. On avait l'habitude de voir qu'il avait une plus grande influence, mais là on dirait que les features extraits du VGG ont tellement servit, que le SVM n'est pas très sensible à la marge. L'idéal serait de trouver un moyen d'augmenter encore l'accuracy à un très haut degré et garder l'insensibilité du SVM au paramètre de la marge. On aurait ainsi une séparatrice parfaite.

## 2 Réponses aux questions du TME 9

- Après application de la carte de saillance sur quelques images, voilà le résultat obtenu en utilisant SqueezeNet :



On remarque bien que les pixels révélateurs sont bien trouvés, pour chaque image, on retrouve l'objet de la classe grâce à ces pixels, on voit bien la silhouette de l'objet, ça identifie ce dernier, et le reste des pixels de l'image est considéré comme environnement auxiliaire non pertinent pour la classification. Les pixels élluminés expliquent le label prédict. Ces cartes de saillance nous indiquent dans quelle mesure le score de prédiction pour la classe  $c$  changerait si les intensités de pixels RVB dans la zone en surbrillance de l'image étaient légèrement modifiées.

- Limites des cartes de saillance :

Les modèles de saillance ne peuvent toujours pas comprendre pleinement la sémantique de haut niveau dans les scènes sémantiquement riches (c.-à-d. «L'écart sémantique»). Pour continuer à approcher la performance au niveau humain, les modèles de saillance devront découvrir

des concepts de plus en plus élevés dans les images : texte, objets, lieux de mouvement et emplacements attendus des personnes dans les images, ainsi que déterminer la relation relative et l'importance des objets. Même les meilleurs prédicteurs de saillance existant aujourd'hui ont tendance à accorder une importance disproportionnée au texte et à l'humain (déttection de texte et de personnes), même si, ce ne sont pas nécessairement les parties les plus intéressantes sur le plan de l'image. Les modèles de saillance devront raisonner sur l'importance relative des régions d'image, comme se concentrer sur la personne la plus importante dans la pièce ou le panneau le plus informatif sur la route (qui dépend de l'image et du **contexte**)

Les études de l'attention cognitive peuvent aider à surmonter certaines de ces limitations.

### 3. L'estimation de saillance peut être considérée comme une instance de **segmentation d'image**.

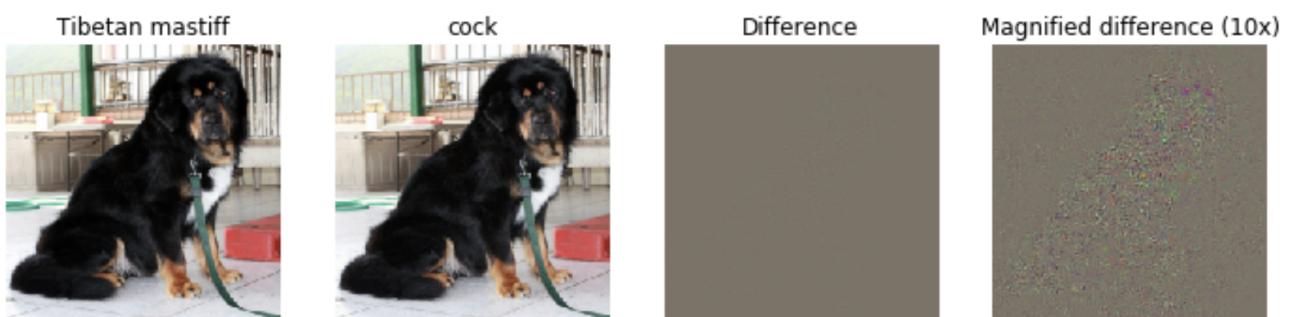
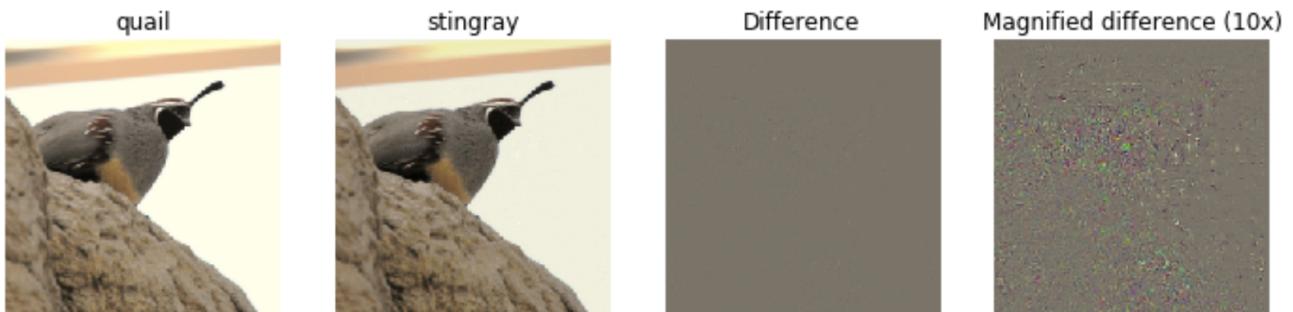
En computer vision, la segmentation d'image est le processus de partitionnement d'une image numérique en plusieurs segments (ensembles de pixels, également appelés superpixels). Le but de la segmentation est de simplifier et/ou de changer la représentation d'une image en quelque chose de plus significatif et plus facile à analyser. La segmentation d'image est généralement utilisée pour localiser des objets et des limites (lignes, courbes...) dans les images. Plus précisément, la segmentation d'image est le processus d'attribution d'une étiquette à chaque pixel d'une image de telle sorte que les pixels avec la même étiquette partagent certaines caractéristiques.

### 4. Résultats avec VGG16 :



En comparant aux cartes obtenues avec SqueezeNet, on voit certains pixels sont illuminés de façon plus intense, notamment le foins, SqueezeNet avait détecté un seul carré, avec VGG, on voit plus de carrés, même ceux qui sont relativement loins. Pour la caille, SqueezeNet avait détecté les pixels du rocher mais pas VGG. Et pour le chien, les deux le cernent bien, mais on voit que VGG se concentre plus sur le visage de ce dernier.

### 5. Résultats de la génération d'exemples adversaires (trompant) :

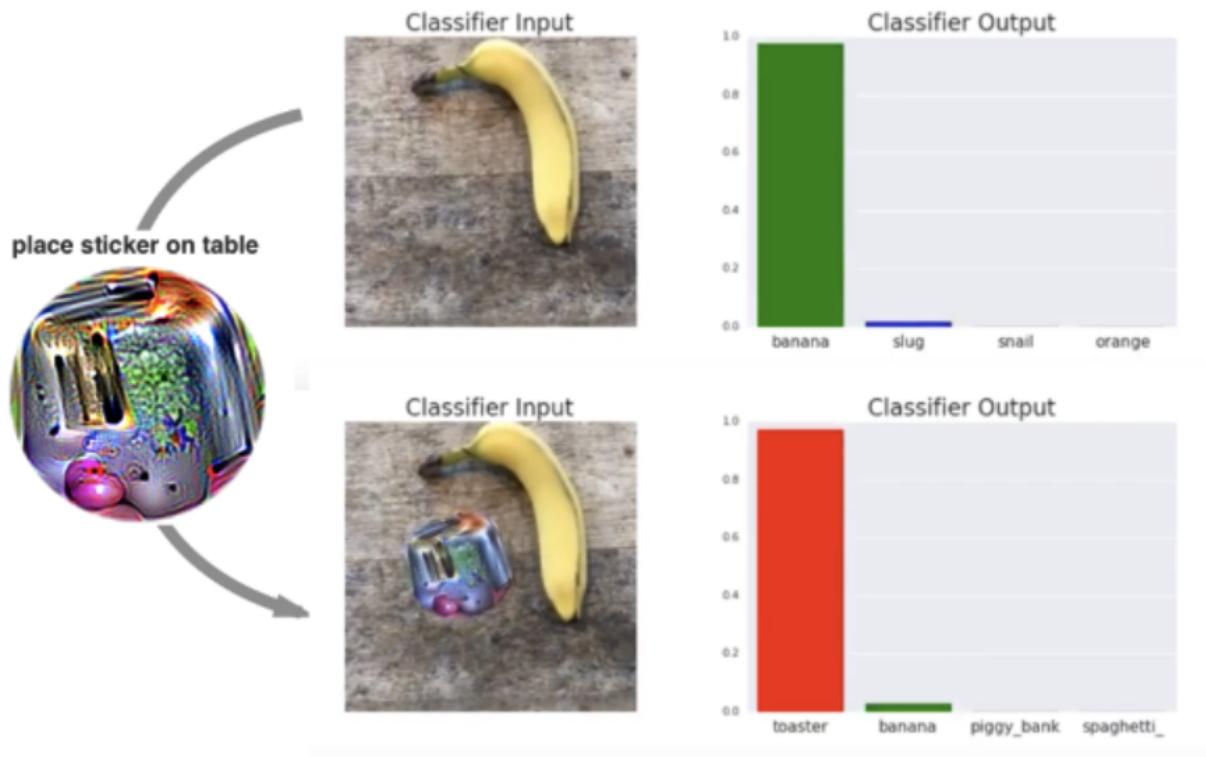


La première colonne représente l'image initiale et son label prédit correspondant. La deuxième représente l'image modifiée à partir de la première et son label prédit, visuellement, l'image n'a même pas l'air d'avoir changé et on voit qu'on a carrément une autre classe, qui est érronée. La différence ne se voit même pas à l'oeil, elle est très légère (d'après la troisième colonne), mais dès qu'on l'emplifie, on voit les pixels qui sont modifiés, et si on compare ça aux cartes de saillance obtenues précédemment pour les mêmes images, on pourrait en conclure que les pixels modifiés pour tromper le réseau sont ceux qui ont été jugés importants, sont les éléments clés de la classification. Une légère modification sur ces pixels là, d'une manière imperceptible pour les humains, peut tromper le réseau pré-entraîné.

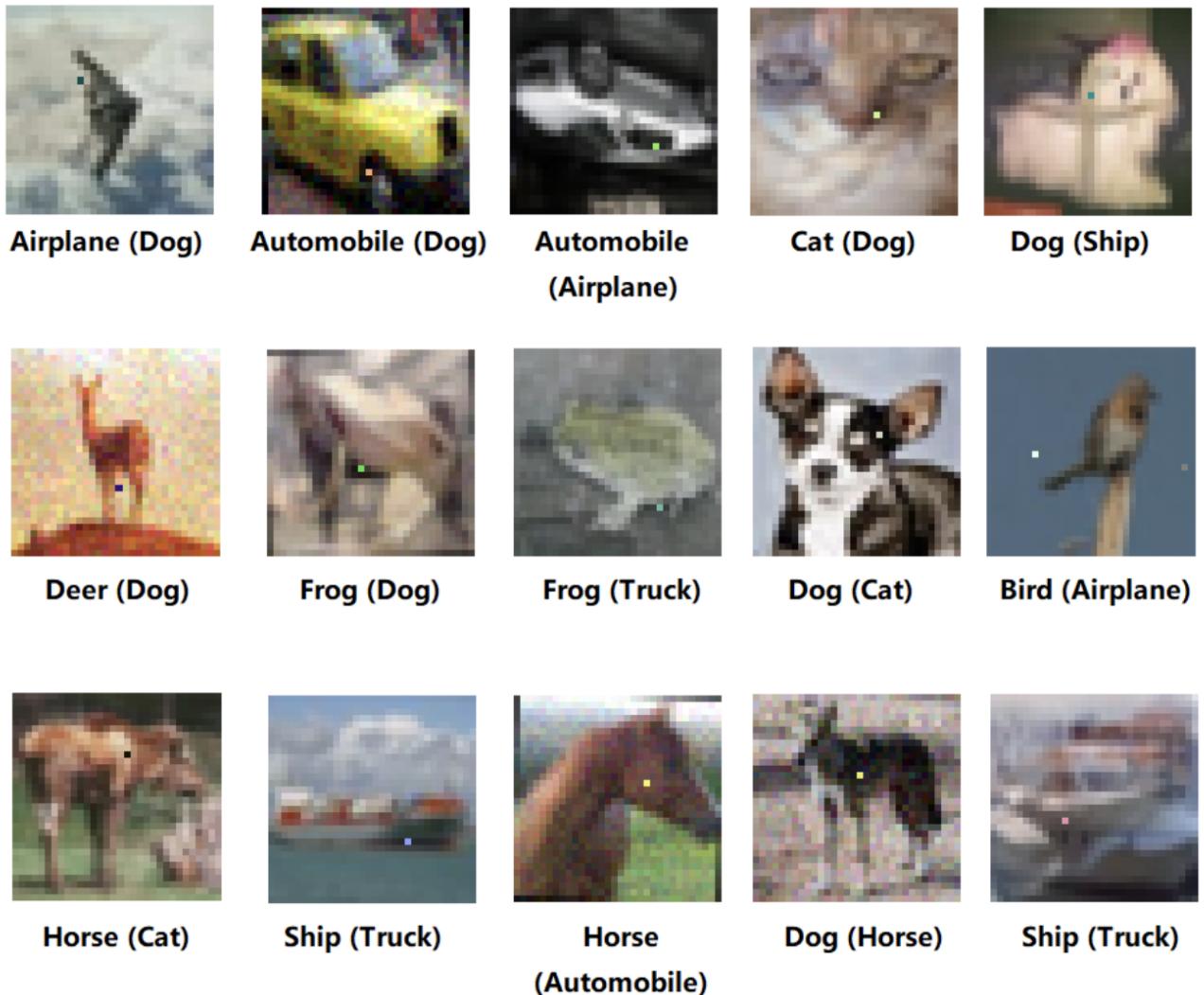
6. On vient de voir que changer une image (par exemple d'un lion) d'une manière imperceptible pour les humains peut amener un DNN à étiqueter l'image comme autre chose (par exemple, étiqueter un lion comme une bibliothèque). En pratique, il est donc facile de produire des images qui sont complètement méconnaissables aux humains, mais que les DNN croient être des objets reconnaissables avec une confiance de 99,99% (par exemple, étiqueter avec certitude que le bruit blanc statique est un Lion). Cela met en lumière des différences intéressantes entre la vision humaine et la computer vision, et soulève des questions sur la **généralité** des réseaux de neurones.

7. Cette approche qu'on vient d'utiliser est l'approche standard pour tromper un classificateur d'objets, elle consiste à créer des images en ajoutant de façon itérative du bruit d'une manière spécifique en essayant de maximiser l'erreur du modèle. L'image qui en résulte ressemble beaucoup aux humains, mais peut entraîner une énorme erreur sur le réseau. Cette façon produit bien des images adversaires mais elle nécessite un processus itératif une modification de toute l'image, de plus, l'image résultante dépend fortement de l'entrée.

Une autre façon pour faire cela serait d'ajouter un autocollant à l'image initiale près de l'objet que l'on cherche à détecter. Cette méthode a été introduite dans **Adversarial Patch**. Cela fonctionne à merveille dans un scénario réel lorsque, par exemple, le classificateur utilise une caméra pour effectuer la détection d'objets en direct.



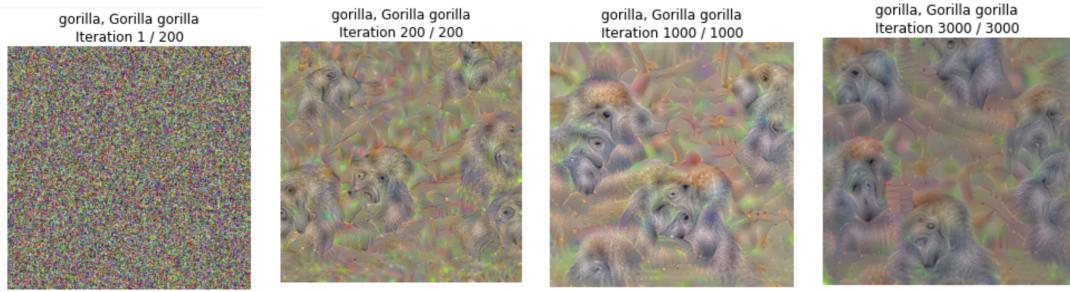
D'autres méthodes ont été présentées dans la littérature, notamment dans **One pixel attack for fooling deep neural networks**. L'article trouve de minuscules perturbations dans l'image entière pour rendre l'ensemble du réseau erroné. Les auteurs estiment que la modification de l'image entière n'est pas nécessaire. C'est une méthode qui ne donne aucune information sur les gradients. Trouver les bonnes valeurs pour l'entrée trompante peut être fait sans avoir aucune connaissance du fonctionnement du modèle (la méthode fonctionnera même si le modèle n'est pas différentiable). Voilà les résultats reportés par les auteurs de l'article :



8. Test avec diverses classes en partant d'un bruit aléatoire ( $l2\_reg = 1e - 3$ ,  $learning\_rate = 5$ ) :

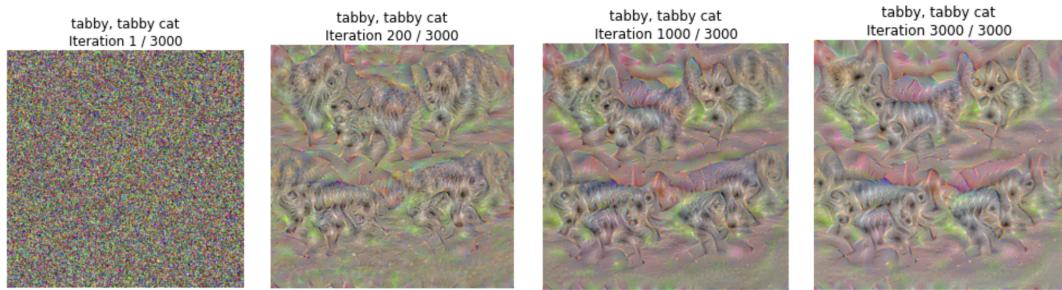


A partir du bruit initial, on voit que le modèle converge (vers ce qu'on obtient à partir de l'itération 1000). A cette étape, on est bien sûr que le modèle prédit la classe chat. En regardant bien les pixels, on pourrait reconnaître les yeux du chat, ses oreilles et même ses poils gris.

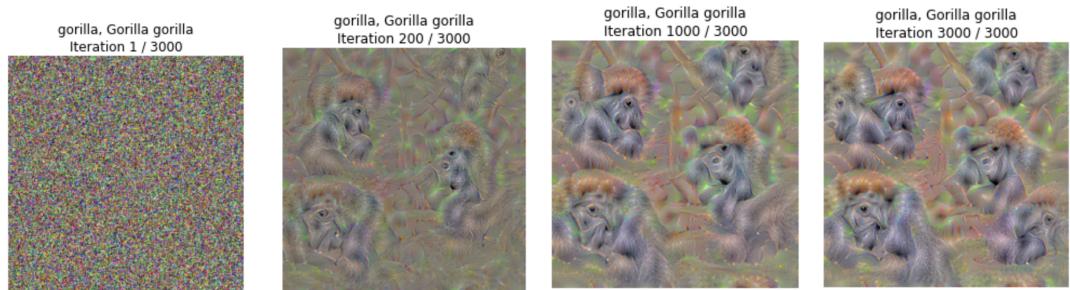


Pour le gorilla, on peut, vers l’itération 200 voir sa tête, on pourrait aussi distinguer de larges épaules noires après vers l’itération 1000.

9. Faisons varier le pas d’apprentissage et la régularisation :
- $learning\_rate = 10$  et  $l2\_reg = 1e - 3$



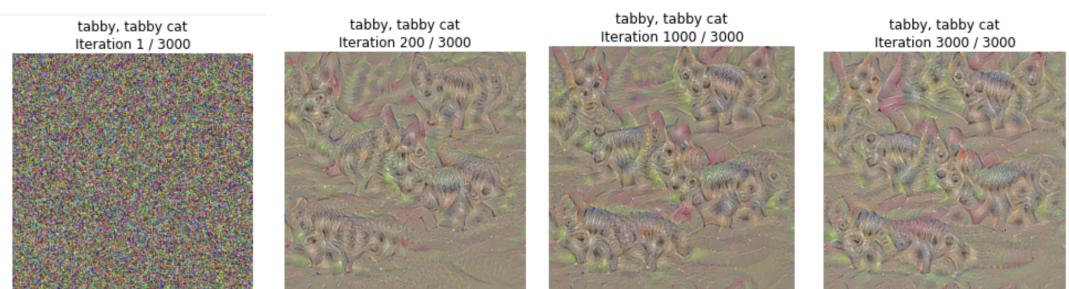
Pour le chat, on ne voit pas une grande différence par rapport au pas d’apprentissage précédent, mais la convergence est certainement plus rapide, si on compare les deux cas de figure à l’itération 200, l’image générée avec ce learning rate est plus intense en termes de formes et de couleurs.

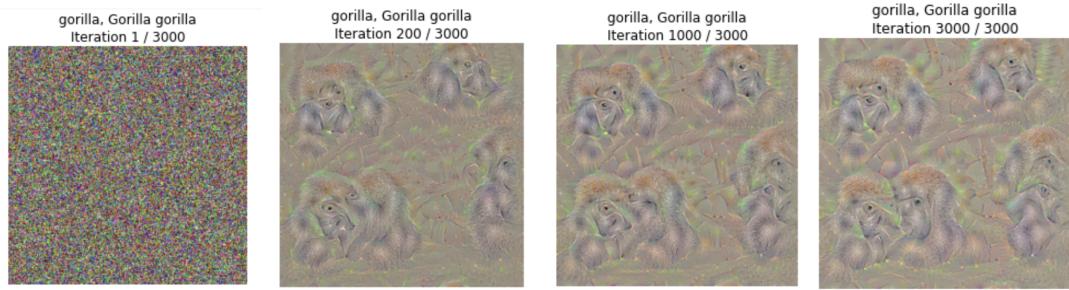


Si on compare l’image générée pour un gorilla à l’itération 3000 avec un pas d’apprentissage de 5 et celle-ci, on voit qu’on a presque les mêmes formes mais les couleurs sont plus intenses, à priori, ce pas est meilleur vu qu’on converge plus vers le noir qui est une caractéristique du gorilla.

Un pas d’apprentissage assez grand (10 dans notre cas) donne de bon résultat pour cette tâche.

$$learning\_rate = 10 \text{ et } l2\_reg = 1e - 2$$





Avec cette régularisation, on perdu en termes d'intensité par rapport aux ca précédents. Il vaut mieux donc garder une régularisation assez basse. Probablement parce que les modifications portées sur les valeurs des pixels sont légères.

10. Utilisation d'une image source venant de ImageNet (la classe cible est la classe réelle) :



Le fait de commencer d'une image, nous fait gagner nous permet déjà d'avoir moins de bruit, et idéalement, il serait possible que l'environnement des objets source et cibles se ressemblent, donc en plus que le fait d'avoir généré l'objet, on aura une image dans un contexte (avec de l'herbe par exemple).

Celà peut aussi nous aider à renforcer notre modèle ultérieurement, si on est en train d'apporter des modifications sur une image en étant sur de garder son label, donc on pourrait trouver un moyen d'apprendre différentes représentations pour un même objet. Et si par exemple on utilise une classe cible différente, on pourrait adapter ça à un algorithmes qui trouverait les similarités et les différences entre les objets, quelles sont les modifications à apporter sur quelque chose pour qu'elle devienne quelque chose d'autre.

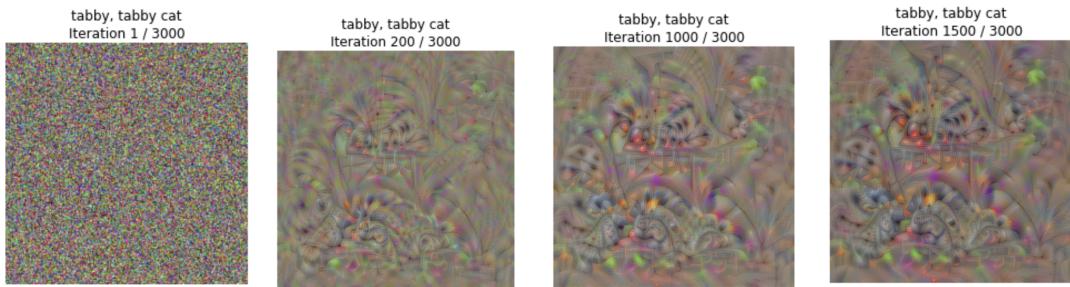
Prenons l'exemple suivant, on veut qu'à partir de "brown bear", obtenir "border terrier".



Voilà le résultat obtenu :

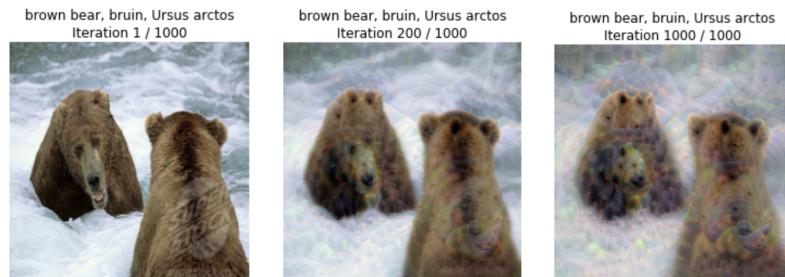


11. Tests avec VGG16 (l'exécution étant plus lente, on a utilisé un nombre d'itérations plus petit) :  
**A partir d'un bruit :**



De vu comme ça, SqueezeNet est mieux adapté pour cette tâche, avec VGG16, on arrive pas à distinguer le chat à l'oeil.

**A partir d'une image :**



Par contre pour cette image, on a un résultat plus stable, comme l'image cible est la source utilisée, le résultat ressemble toujours à l'image initiale.

### 3 Réponses aux questions des TMEs 10 et 11

1. Intérprétation des équations :

$G_{\theta_g}$  est entraîné pour tromper le discriminateur  $G$ . Il veut donc *maximiser* la sortie du Discriminateur pour les images que lui génère. Celà s'écrit donc :

$$\max_G E_{z \sim p_z} \log(D(G(z)))$$

$D_{\theta_d}$  quant à lui est entraîné pour reconnaître les images générées par  $G$ .

- Maximiser  $D(\cdot)$  sur  $p_{data}$ .
- Minimiser  $D(\cdot)$  sur  $p_z$ .

Comme la sortie de  $D$  est 0 pour *Fake* et 1 pour *Real*, minimiser  $D(\cdot)$  reviendrait à maximiser  $(1 - D(\cdot))$ . Le gain de  $D$  pourrait donc s'écrire de la façon suivante :

- Maximiser  $D(\cdot)$  sur  $p_{data}$ .
- Maximiser  $(1 - D(\cdot))$  sur  $p_z$ .

Le problème de façon probabilistique s'écrit donc avec l'équation :

$$\max_D [E_{x \sim p_{data}}(\log D(x)) + E_{z \sim p_z}(\log(1 - D(G(z)))]$$

2. Idéalement, le Générateur  $G$  transforme  $P(z)$  en  $P(data)$ .
3. L'équation initialement proposée est :  $\max_G - E_{z \sim p_z} \log(1 - D(G(z)))$ .

**Justification du papier :**

**Theorem 2.4 (Vanishing gradients on the generator).** Let  $g_\theta : \mathcal{Z} \rightarrow \mathcal{X}$  be a differentiable function that induces a distribution  $\mathbb{P}_g$ . Let  $\mathbb{P}_r$  be the real data distribution. Let  $D$  be a differentiable discriminator. If the conditions of Theorems 2.1 or 2.2 are satisfied,  $\|D - D^*\| < \epsilon$ , and  $\mathbb{E}_{z \sim p(z)} [\|J_\theta g_\theta(z)\|_2^2] \leq M^2$ , then

$$\|\nabla_\theta \mathbb{E}_{z \sim p(z)} [\log(1 - D(g_\theta(z)))]\|_2 < M \frac{\epsilon}{1 - \epsilon}$$

**Corollary 2.1.** Under the same assumptions of Theorem 2.4

$$\lim_{\|D - D^*\| \rightarrow 0} \nabla_\theta \mathbb{E}_{z \sim p(z)} [\log(1 - D(g_\theta(z)))] = 0$$

Dans lequel, le gradient disparaît lorsque le discriminateur devient optimal ( $D$  est proche de  $D^*$ ), i.e.,  $\Delta_{\theta_g} \log(1 - D(G(z))) \rightarrow 0$

**Theorem 2.5.** Let  $\mathbb{P}_r$  and  $\mathbb{P}_{g_\theta}$  be two continuous distributions, with densities  $P_r$  and  $P_{g_\theta}$  respectively. Let  $D^* = \frac{P_r}{P_{g_{\theta_0}} + P_r}$  be the optimal discriminator, fixed for a value  $\theta_0$ <sup>3</sup>. Therefore,

$$\mathbb{E}_{z \sim p(z)} [-\nabla_\theta \log D^*(g_\theta(z))|_{\theta=\theta_0}] = \nabla_\theta [KL(\mathbb{P}_{g_\theta} \| \mathbb{P}_r) - 2JSD(\mathbb{P}_{g_\theta} \| \mathbb{P}_r)]|_{\theta=\theta_0} \quad (3)$$

Le nouveau gradient se compose d'une divergence KL inverse et d'un terme de divergence JS. Le terme KL inversé attribue un coût élevé à la génération d'images non naturelles tandis que la suppression de mode est plus acceptable.

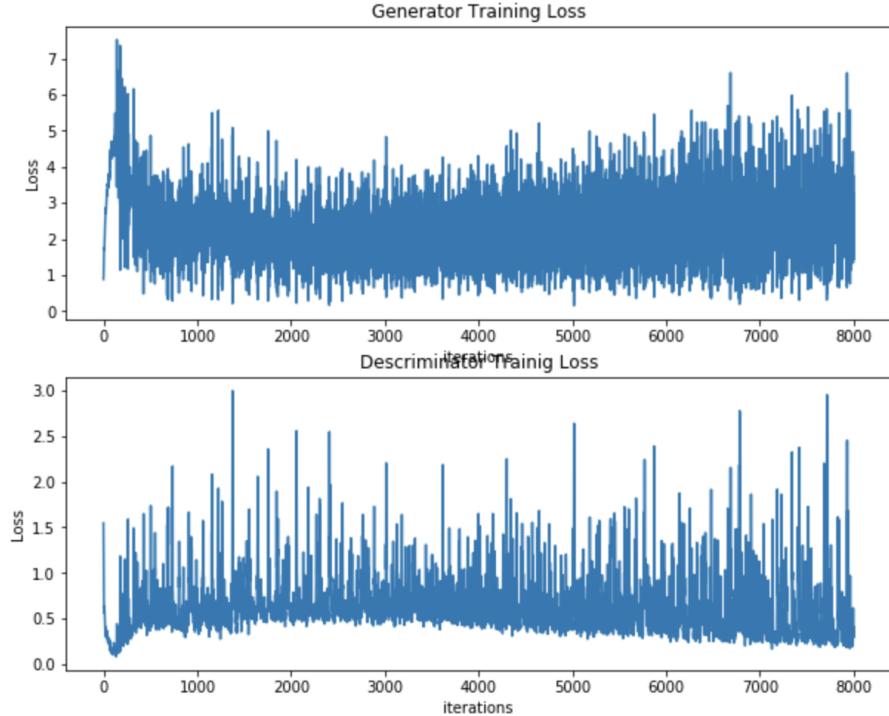
4. Résultats avec les hyper-paramètres proposés par défaut : Entre les 4000 et 8000 itérations on ne voit pas une grande différence. De loin, les images ont belle et bien la forme des visages, mais dès que l'on approche, on voit les déformations. On peut également remarquer leurs variétés (visages masculins et féminins, cheveux bruns et cheveux blancs).

Evolution de l'erreur :



(a) 4000 iterations

(b) 8000 iterations

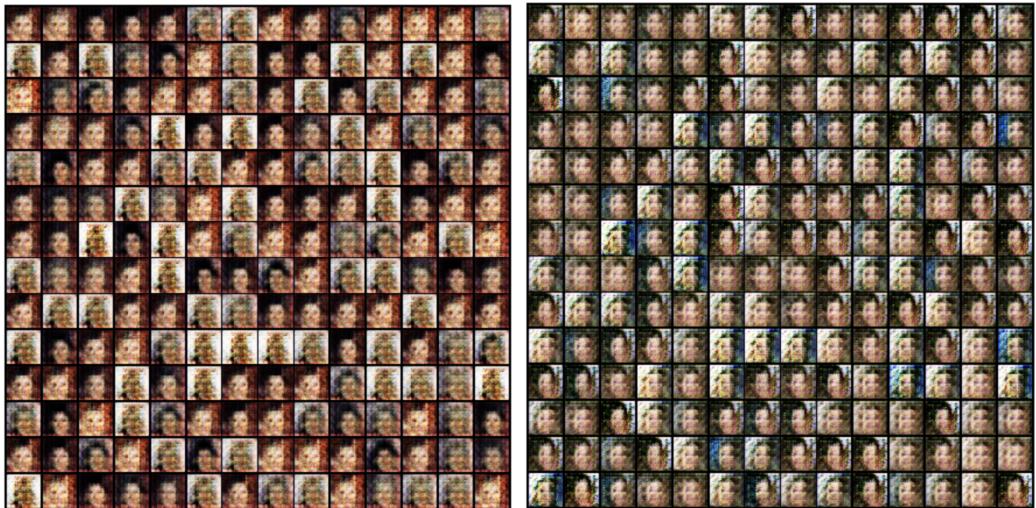


L'erreur semble stagner pour les deux modèles, aux alentours de 3 pour le générateur et de 0.5 pour le déiscriminateur. L'erreur du Descriminateur est toujours en dessous de celle du générateur, l'objectif n'est pas encore atteint dans ce cas.

##### 5. Différentes expériences :

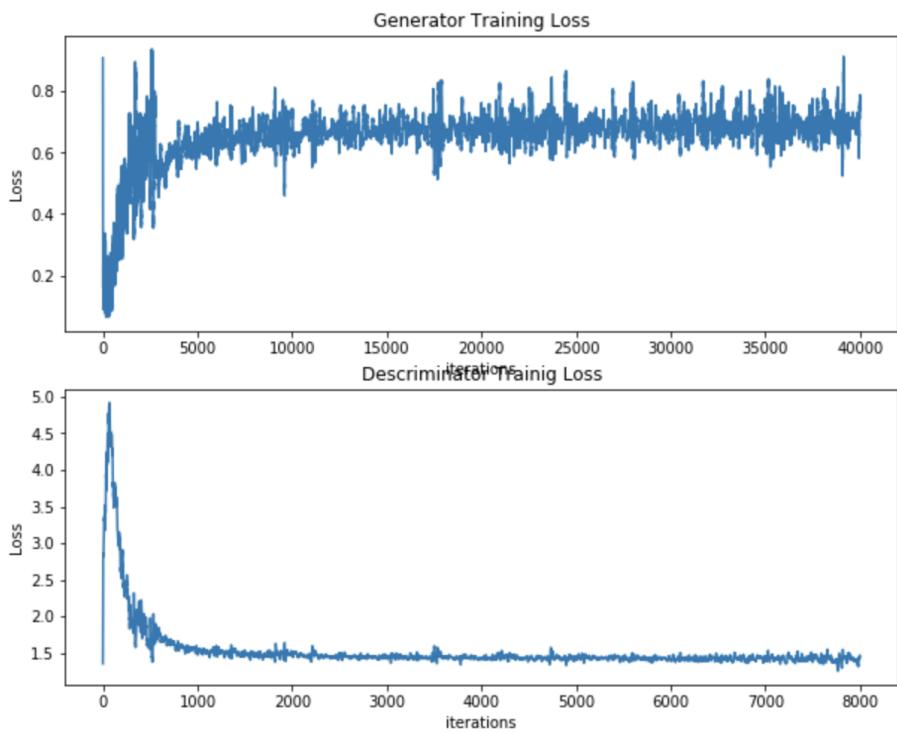
- Les premiers changements qu'on a apportés sur les paramètres sont sur le **momentum** qu'on a posé à **0.9** qui est la valeur par défaut d'Adam. On a également modifié l'équilibrage entre les deux modèles, de telle sorte ou on favorise l'apprentissage du Générateur et celà en mettant **nupdates\_G** à **5**.

Les images sont moins diversifiées (tout les cheveux sont noirs par exemple) et elles sont moins nettes.



(a) 4000 iterations

(b) 8000 iterations



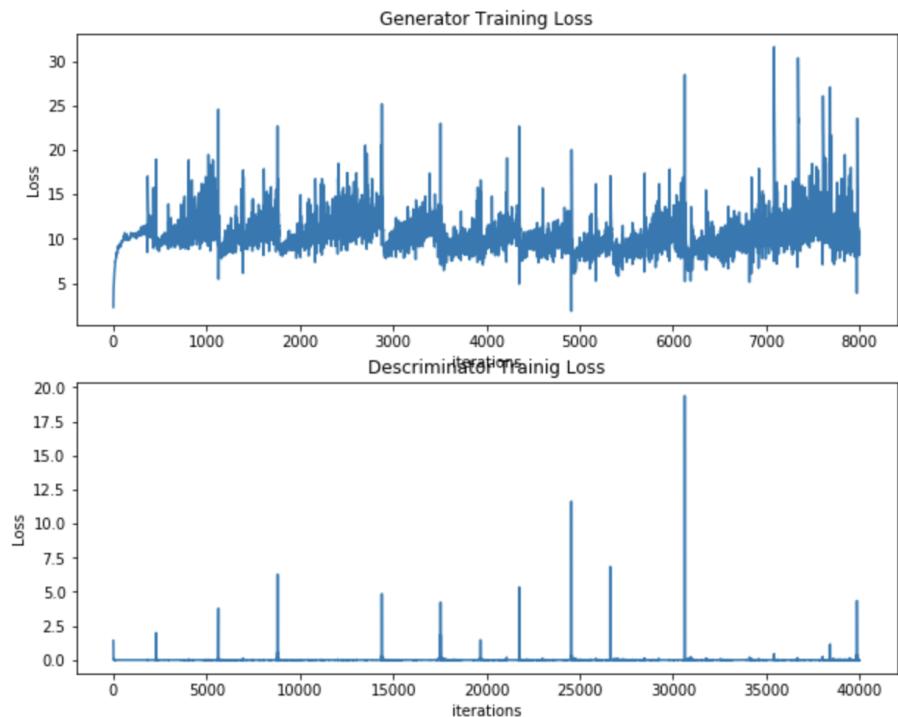
Même si les images ne semblent pas très claires, la loss évolue d'une bonne façon contre à elle, le fait d'avoir utilisé un **momentenum** assez élevé a permis de converger plus vite (le descriminateur sur les premières mille itérations et le générateur sur les premières 5000). Et le fait d'avoir favorisé l'apprentissage du générateur, a permis de le faire gagner, on voit d'ailleurs que vers la fin de l'entraînement, i.e, à la convergence, la loss du générateur est à **0.7** contre **1.5** pour le descriminateur.

- La modification suivante comporte à favoriser plutôt l'apprentissage du descriminateur, en mettant **nupdates\_D** à **5** et **nupdates\_G** à **1**, à priori celà ne va pas conduire à de bons résultats, vu que notre but principal est de faire gagner le générateur. Mais on va quand même faire celà pour pouvoir visualiser cette différence. Le **momentanum** est posé à **0.9**.



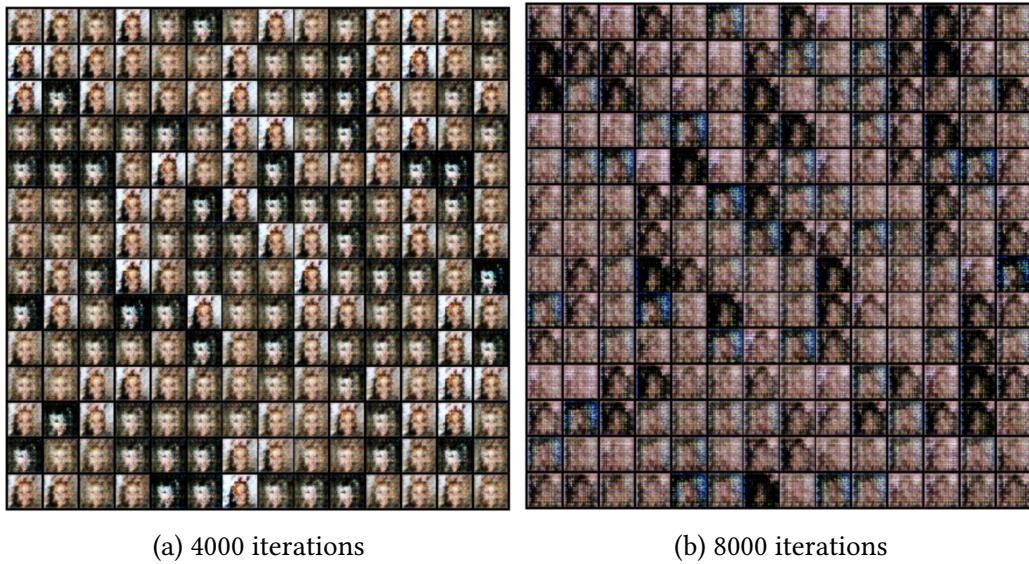
(a) 4000 iterations

(b) 8000 iterations



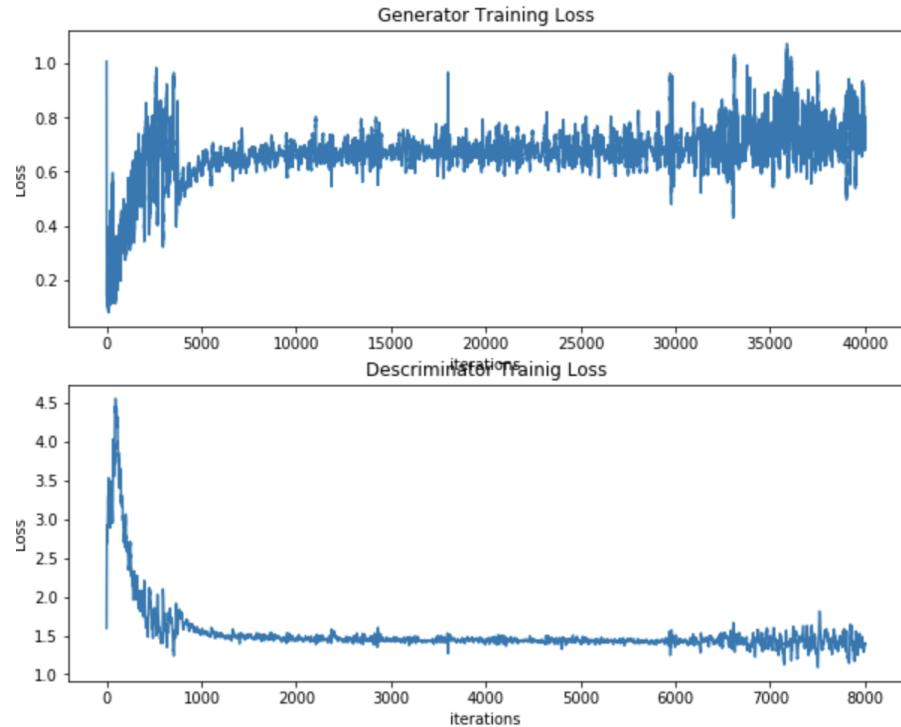
Le modèle ne converge pas, c'est le descriminateur qui est en train de gagner. Et c'est normal vu qu'on favoriser son apprentissage. Une règle à retenir est donc de faire apprendre le générateur plus que le descriminateur.

- On va maintenant s'intéresser à **n\_z** qu'on va fixer à **1000**. On garde le **momentenum** à **0.9**, et les valeurs de **nupdates\_G** et **nupdates\_D** sont respectivement **5** et **1** vu que c'était dans ce cas là où le générateur avait gagné.



(a) 4000 iterations

(b) 8000 iterations

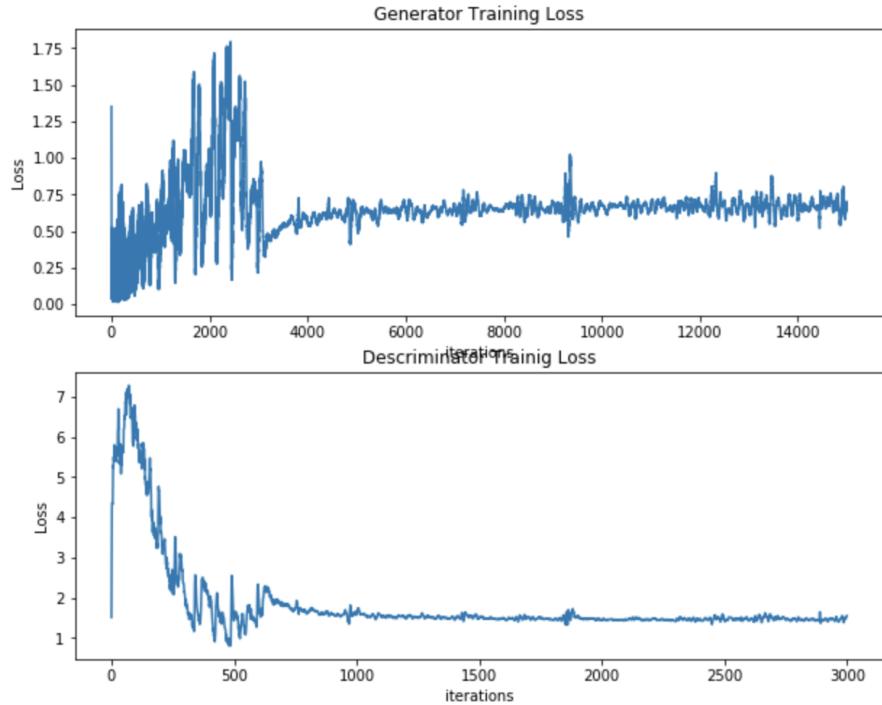


- Pour les essais suivants, on garde les hyper-paramètres qui ont donné les meilleurs résultats jusque là. Et on lance notre modèle sur la base **celeba64** dont les images sont 64x64. On diminue le nombre d’itérations pour assurer l’exécution, trop lente sinon.



(a) 4000 iterations

(b) 8000 iterations



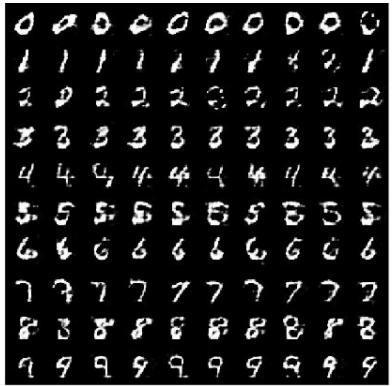
En termes de loss, le modèle agit bien, le générateur converge vers **0.70** et le discriminateur vers **1.5**, cependant, les images ne sont pas trop visibles, cette différence entre les deux pertes n'est pas assez significante pour obtenir des images vraisemblables. Celà pourrait s'améliorer en augmentant le nombre d'itérations, vu que les images sont plus grandes. On n'est pas allés au boüt de l'expérience car l'exécusion devenait trop lente.

## 6. Equations conditionnées :

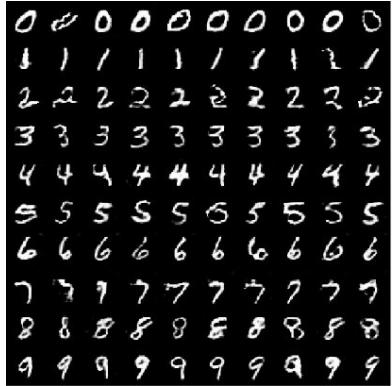
$$\max_{cG} E_{z \sim p_z} \log(D(cG(z, y))) \quad (6 \text{ bis})$$

$$\max_D [E_{x \sim p_{data}} (\log D(x, y)) + E_{z \sim p_z} (\log(1 - D(cG(z, y), y))] \quad (7 \text{ bis})$$

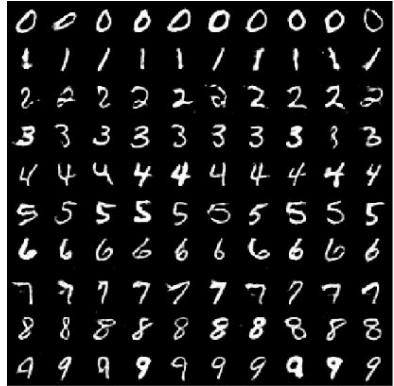
7. Il pourrait être conditionné aux variables : âge et sexe.
8. Il pourrait être conditionné aux variables : Saison (Couleur, forme des arbres...etc).
9. Il pourrait être conditionné aux variables : Type d'urbanisation, Couleur.
10. Résultats obtenus avec les hyper-paramètres proposés par défaut :



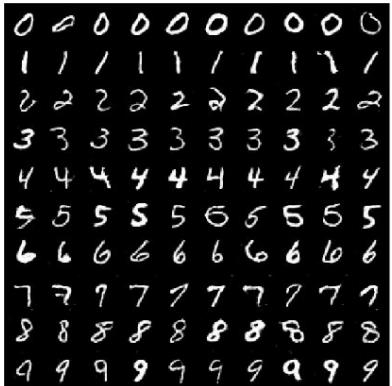
(a) 1000 iterations



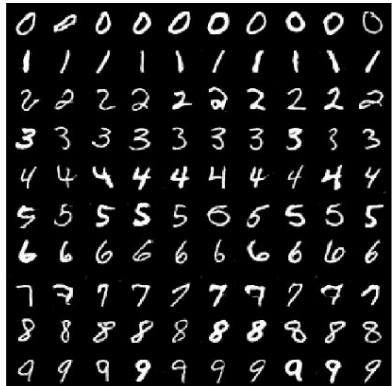
(b) 2000 iterations



(c) 3000 iterations



(a) 4000 iterations

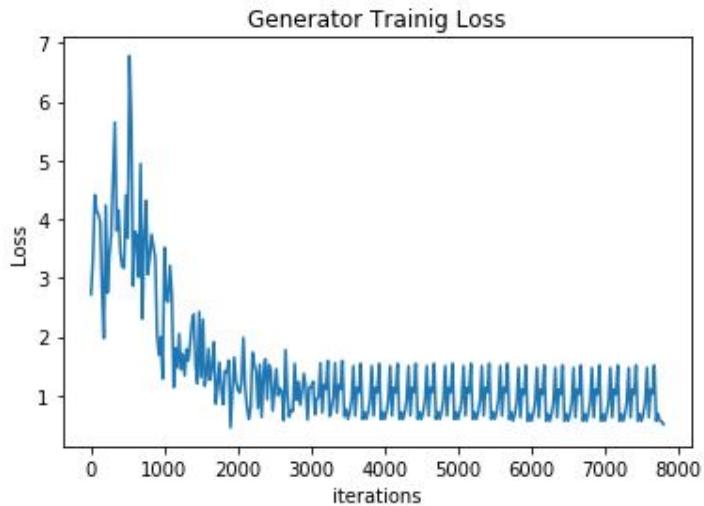


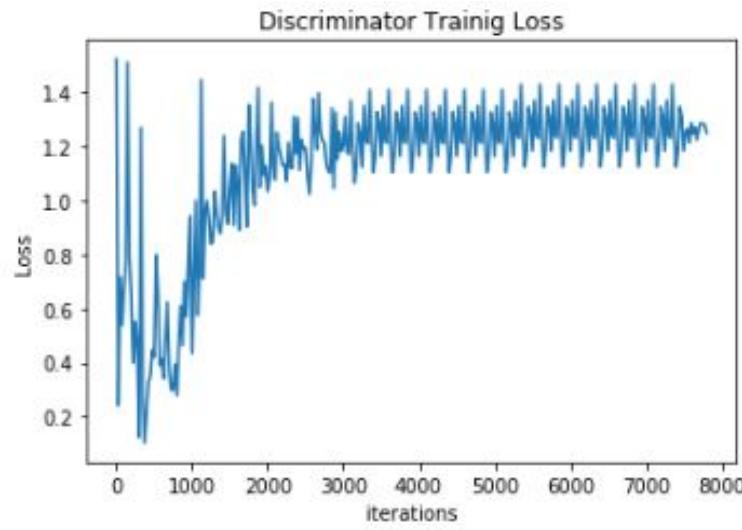
(b) 5000 iterations



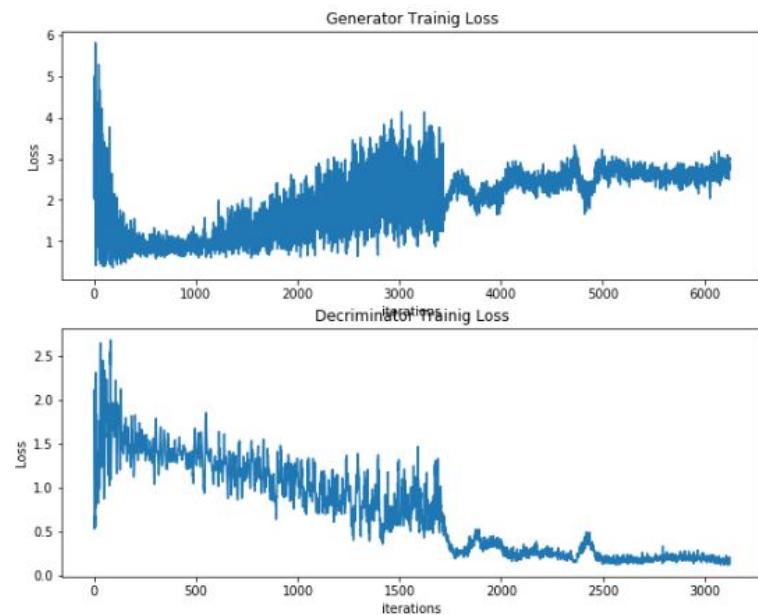
(c) 6000 iterations

En ce qui s'agit des images générées, on voit qu'au bout des 1000 itérations, le modèle commence déjà à générer des chiffres lisibles. Néanmoins, le but n'est pas encore tout à fait atteint à cette étape là. Au bout de 4000 itérations, on obtient des chiffres tout à fait lisibles et complets, à ce moment là on peut dire qu'en termes d'images notre but est atteint.





Les courbes ci-dessus représentent l'évaluation de l'erreur pour le générateur et le discriminateur (prises avec des pas de 25). On voit qu'après 3500 itérations environ, l'erreur se stabilise pour les deux (1 pour le générateur et 1.3 pour le discriminateur), on peut dire qu'on a à peu près atteint notre but de génération.



Les courbes ci-dessus représentent l'évaluation de l'erreur pour le générateur et le discriminateur (prises à chaque pas de temps) avec  $\text{nb\_update\_D}=1$  et  $\text{nb\_update\_G}=2$ . On n'a pris que 3125 itérations car l'exécution est très lente (manque de ressources). On voit que l'erreur du discriminateur est toujours en dessous de celle du générateur, l'objectif n'est pas encore atteint dans ce cas.

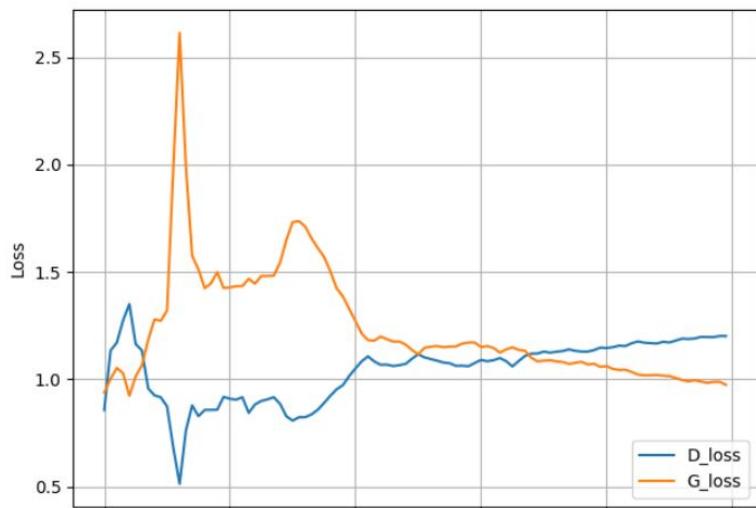
Les images obtenues sont :



On voit que les chiffres obtenus ne sont pas assez clairs et sont moins bons que ceux obtenus précédemment (On aurait pu obtenir de meilleurs résultats en augmentant le nombre d'itérations mais cela aurait encore pris un temps double (voir 24h ou plus)).

11. Oui, il est possible de ne conditionner que le générateur et de laisser le discriminateur sans contrainte.
12. La meilleure génération de chiffres conditionnés avec le cGAN :





Les courbes ci-dessus représentent la variation de la loss pour le générateur et le discriminateur en fonction du nombre d'itération. Au début de l'apprentissage (avant 3000 itérations environ), l'erreur en génération est plus grande que celle en discrimination, puis elle deviennent presques identiques, enfin, vers la fin de l'apprentissage, l'erreur du discriminateur va dépasser celle du générateur et le modèle va se stabiliser.

13. Les images que génère le cDCGAN sont meilleures que celles générées avec le cGAN, elles sont en fait plus claires et moins floues. En effet, Le DCGAN est proposé pour résoudre le problème d'instabilité du GAN. En comparant le cDCGAN avec un générateur conventionnel(cGAN) composé d'un réseau neuronal standard (NN), le cGAN génère des conceptions de sortie bruyantes et, par conséquent, leurs spectres de réflexion ne correspondent pas bien à la vérité terrain. Alors que cDCGAN fournit des images structurelles réaliste. Donc, le cDCGAN apprend efficacement la corrélation entre les spectres d'entrée et leurs images de conception correspondantes par rapport au cGAN conventionnel.