

# Rapport

Benotsmane ismat

December 2020

## Table des matières

<b>1</b>	<b>Transfert learning</b>	<b>3</b>
1.1	Architecture VGG16 . . . . .	3
1.2	Tansfer learning avec VGG16 sur 15 Scene . . . . .	5
<b>2</b>	<b>Visualisation des réseaux de neurones</b>	<b>7</b>
2.1	Carte de saillance . . . . .	7
2.2	Exemples adversaires . . . . .	9
2.3	Visualisation de classes . . . . .	10
<b>3</b>	<b>Generative Adversarial Networks</b>	<b>13</b>
3.1	Principe général . . . . .	13
3.2	Architectures des réseaux . . . . .	14
<b>4</b>	<b>Conditional Generative Adversarial Networks</b>	<b>20</b>
4.1	Principe général . . . . .	20
4.2	Architectures cDCGAN pour MNIST . . . . .	21
4.3	Architectures cGAN pour MNIST . . . . .	22

# 1 Transfert learning

## 1.1 Architecture VGG16

(Q1) Rappelons que le nombre de poids à apprendre pour une couche fully-connected est le produit entre la dimension d'entrée et la dimension de sortie.

Le nombre de poids à apprendre dans les couches fully-connected du VGG16 est :

- FC1 :  $7 \times 7 \times 512 \times 4096 + 4096 = 102\,764\,544$
- FC2 :  $4096 \times 4096 + 4096 = 16\,781\,312$
- FC3 :  $4096 \times 1000 + 1000 = 4\,097\,000$

On estime donc qu'un réseau VGG16 doit apprendre au minimum **124M** de paramètres.

(Q2) La sortie de la dernière couche du réseau est un vecteur de dimension 1000, qui correspond à la probabilité d'appartenance à chacune des 1000 classes d'Imagenet sur lequel le réseau a été entraîné.

(Q3) Nous voulons observer la prédiction du réseau VGG16 sur quelques exemples d'image :



FIGURE 1 – Reflex camera

Pour la figure 1, on a **reflex camera** comme prédition de la classe la plus probable avec une probabilité de **0.263**.

Pour la figure 2, il prédit **dinning table** avec une proba **0.429** et **chair** avec une proba **0.401**.



FIGURE 2 – Table and chairs



FIGURE 3 – Skull

Quant à la figure 3, les trois classes les plus probables sont **golf ball**, **penny bank** et **conch**. On remarque qu'aucun des prédictions correspond à l'image. Ici, on remarque la limite de ce modèle qui fonctionne que sur des classes sur lesquelles il a été entraîné.

(Q4) La figure 4 illustre quelques exemples de filtres obtenus à la sortie de la première convolution. On peut observer par exemple la frontière qui sépare les couleurs sur la tête du chat, ou décomposition des différentes zones de couleurs sur l'image.

La première convolution capture des informations de bas niveau, comme des bords, des couleurs... c'est pourquoi il est difficile d'interpréter une carte.

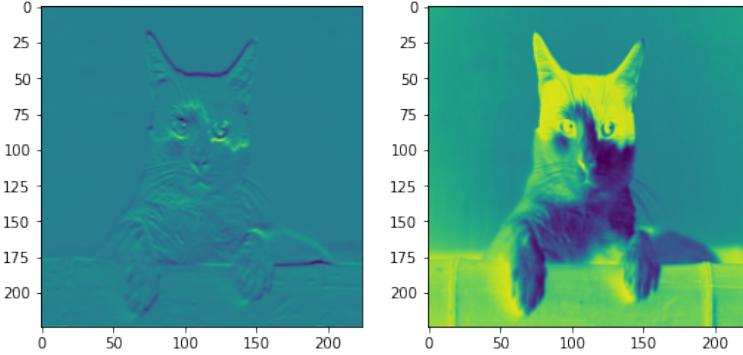


FIGURE 4 – Exemples de cartes obtenues après la première convolution

## 1.2 Transfer learning avec VGG16 sur 15 Scene

**(Q5)** Nous n'apprenons pas VGG16 directement sur 15 Scene car c'est un très petit dataset comparé à Imagenet et ce qui aura pour conséquence un sur-apprentissage avec ce réseau composé d'environ 140M de paramètres.

Une autre raison aussi, c'est le temps d'apprentissage conséquent et les ressources hardware nécessaires pour ce type d'architecture.

**(Q6)** Comme le dataset 15 Scene est très proche d'Imagenet, on peut se servir des très bonnes performances du VGG et de sa partie feature extraction pour extraire des caractéristiques générales des nouvelles images.

**(Q7)** Cette méthode peut poser problème car un VGG est appris sur un type d'image particulier, si on transfert ce réseau et qu'on essaye de lui fournir des images avec des caractéristiques différentes (par exemple très sombre) que celle sur lesquelles il a été pré-entraîné alors elles ne seront pas détectées.

**(Q8)** Les premières couches de convolutions extraient des patterns de très bas niveau et plus on avance dans le réseau plus on détecte des patterns de plus en plus complexes.

De ce fait, si on extrait les features à une couche en début de réseau on obtient des features plutôt générales. Si on sélectionne en fin de réseau, on obtient des features très spécifiques au dataset sur lequel le réseau a été pré-entraîné.

Le but est donc de trouver un compromis entre caractéristiques simples et caractéristiques spécifiques pour trouver la couche où couper notre réseau pré-entraîné, afin de généraliser sur notre problème.

**(Q9)** Les images de 15 Scene sont en noir et blanc, donc l'image est codé sur un seul channel. Or le réseau VGG prend en entrée une image codée sur 3 channels RGB.

Pour remédier à ce problème, on va dubliquer le channel de l'image 15 Scene 3 fois.

**(Q10)** On peut remplacer le classifieur SVM indépendant en remplaçant la dernière couche linéaire du réseau par notre propre couche linéaire qui a pour dimension de sortie 15.

Mais attention, lors de l'apprentissage l'erreur peut être propagée sur l'ensemble du réseau et donc corriger les couches empruntées au VGG16. Pour éviter cela, on peut utiliser l'option *freeze* proposé par Pytorch et ainsi on apprendra que sur la couche introduite.

On pourra aussi vouloir backpropagé l'erreur sur certaines couches du réseau dans le cas où le dataset de notre problème est très différent à celui de Imagenet et ainsi augmenté la performance du modèle.

**(Q11)** Nous avons tester notre modèle de base avec un SVM et l'hyperparamètres  $C = 1$ , à la sortie de la feature extraction. Nous avons obtenu une *accuracy* = **88.9%**.

Nous voulons maintenant effectuer plusieurs manipulations pour observer si on arrive à améliorer la performance de notre modèle.

- On a varié la valeur de l'hyperparamètre **C** avec les valeurs suivantes [0.001, 0.01, 0.1, 0.5, 1] et nous avons obtenu la même accuracy que précédemment **88.9%** avec **C=1**.
- On a ensuite essayé de couper le réseau VGG16 à différents endroits. On a coupé à dans le classifieur à la sortie de chacune des relus des couches linéaires et dans la partie convolution à la sortie de la dernière couche pooling, à la sortie de la dernière convolution et à la convolution qui la précède.

Le meilleur score d'accuracy **90.6%** a été obtenu à la sortie **relu 6**, c'est-à-dire à la sortie de la première couche linéaire du classifieur.

On peut noter également que plus on coupe en amont du réseau et plus l'espace de représentation d'une image est très grand, le temps d'apprentissage du classifieur augmentera en conséquence. De plus, avec un classifieur comme SVM il est moins performant sur des espaces de dimensions très grands.

- On a également remplacé le classifieur SVM par une couche linéaire qui a pour une dimension de sortie 15. On a **freeze** toutes les autres couches du réseau afin d'apprendre que les paramètres de cette couche. On a obtenu une accuracy de **89.6%** légèrement meilleure que le SVM.

L'inconvénient c'est le temps d'apprentissage beaucoup plus long.

## 2 Visualisation des réseaux de neurones

### 2.1 Carte de saillance

(Q1) Pour obtenir la carte de saillance, on affiche la norme de la dérivée de la sortie du réseau par rapport à l'image d'entrée.

La carte de saillance nous permet d'observer quels pixels ont participé à la classification de l'image.

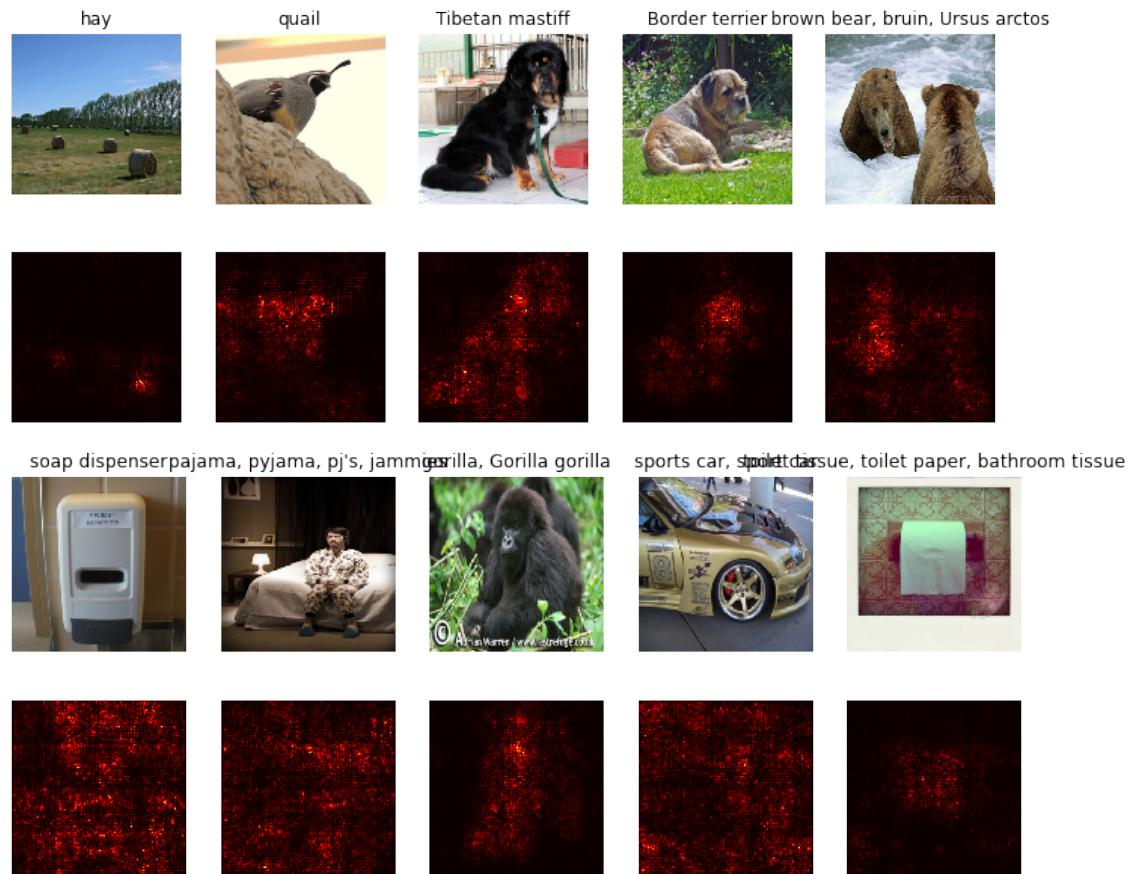


FIGURE 5 – Cartes de saillances avec SqueezeNet 1.1

À la figure 5, on peut voir que le réseau pour classifier l'image en botte de foin ou en chien il a activé pour les 2 images les pixels qui se trouvent sur ces derniers.

Pour l'image de la voiture, le réseau n'a pas utilisé les pixels auxquels un humain aurait pensé pour identifier cette image, c'est à dire seulement les pixels sur la voiture mais il a utilisé en plus des pixels qui concernent l'environnement. On

peut supposer que le dataset comporte un biais où toutes les images de voitures sont prises dans la rue.

Pour finir, on peut souligner l'intelligence du réseau dans la classification de l'image pyjama, où en s'appuyant sur les pixels de la chambre il a pu faire une inférence que le vêtement porté par la personne c'est un pyjama et pas un costume par exemple.

**(Q2)** La limite de cette technique, c'est qu'il arrive que certains pixels qui ont participé à la décision ne soient pas affichés. Ceci peut être expliqué par les couches de pooling qui compressent l'information.

On peut avoir aussi la participation de pixels qui à première vue n'ont aucune relation avec l'objet qu'on cherche à classifier, comme l'exemple de la voiture ou du pyjama vu précédemment.

**(Q3)** Cette technique peut servir par exemple pour la détection d'objet ou la segmentation. On peut s'en servir également pour tromper le réseau, en connaissant les pixels activés et les modifier. En s'appuyant sur le même principe, lors de l'apprentissage on peut brouter les pixels activés afin de rendre le modèle plus robuste.

**(Q4)** On a calculé les cartes de saillances sur les mêmes images mais avec le réseau **VGG16**. La première observation qui saute aux yeux, c'est l'impression que les pixels utilisés pour la classification sont plus précis qu'avec le réseau précédent.

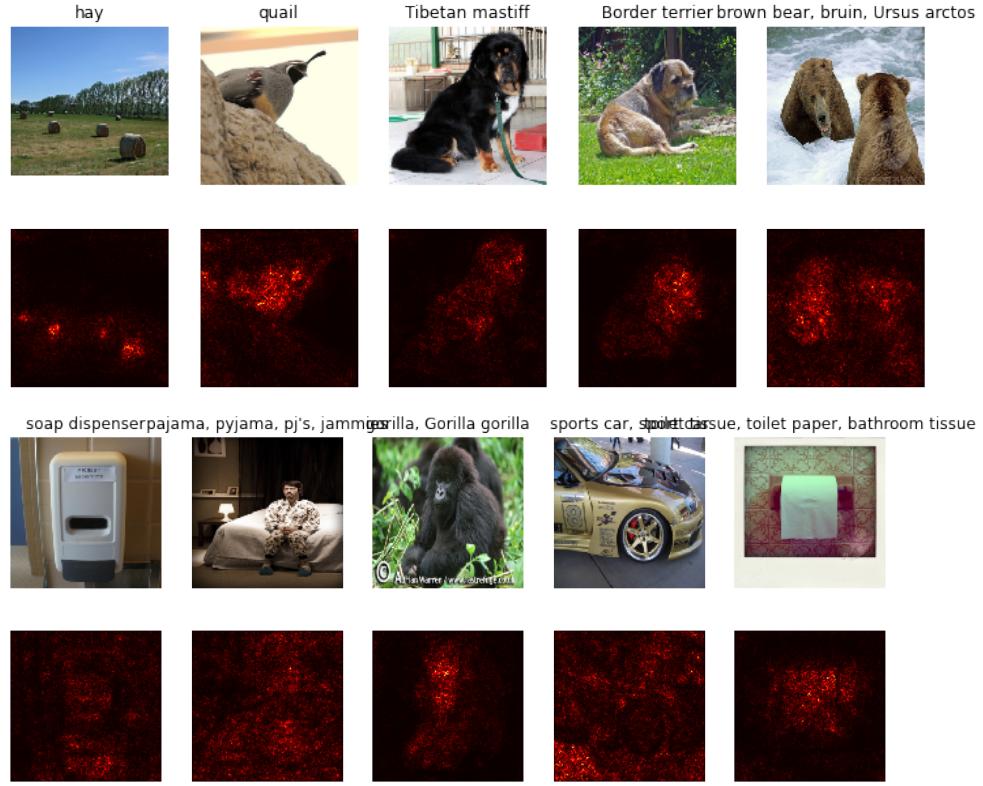


FIGURE 6 – Cartes de saillances avec VGG 16

## 2.2 Exemples adversaires

**(Q5)** Dans la figure 7 on a modifié l'image de *quail* jusqu'à qu'elle soit classifiée par le réseau comme un *stingray*. Comme on le voit dans la deuxième image, le résultat n'est pas du tout visible à l'oeil humain comme vient le confirmer la troisième carte qui représente le bruit ajouté à l'image d'origine. Il y'a que grâce à la dernière carte, où on a multiplier le bruit par 10 pour s'apercevoir des différences.

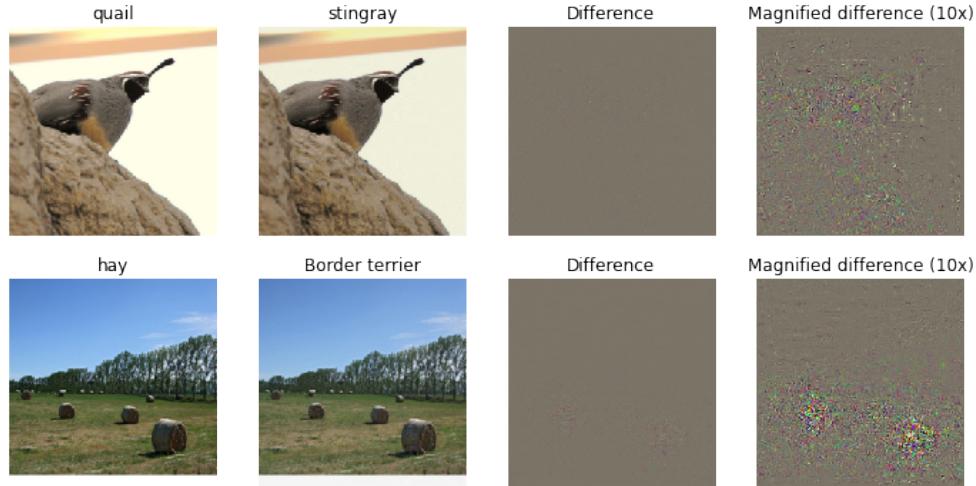


FIGURE 7 – Résultat d'exemple adversaire par montée de gradient

**(Q6)** On vient de montrer qu'on peut effectuer une modification sur une image sans qu'elle ne soit perceptible à l'oeil humain mais qui est perceptible par le réseau car il va la classer comme appartenant à une autre classe.  
Si on a accès au modèle, on peut apprendre à tromper le réseau pour une image donné.

**(Q7)** Cette technique présente des limites car elle suppose qu'on a accès au modèle. Mais dans le cadre où on a pas accès au modèle, on peut ajouter à l'image un motif qui caractérise une certaine classe afin de tromper le réseau.

### 2.3 Visualisation de classes

**(Q8)** En partant d'une image initial comme un bruit aléatoire, nous voulons modifier cette image pour la faire correspondre à une classe donnée.  
À la figure 8, on peut distinguer pour la tarentule ses pattes et son corps au centre. De même pour la caniche où on peut apercevoir ses pattes, sa tête ainsi qu'un œil.

On peut dire que cette technique permet de générer les patterns qui caractérise une classe donnée.

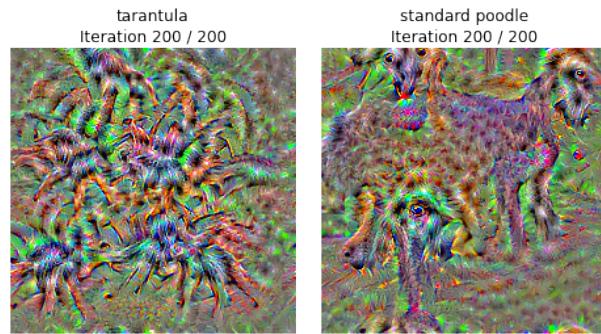


FIGURE 8 – D'image initialisée aleatoirement à une image de la classe souhaité

(Q9)

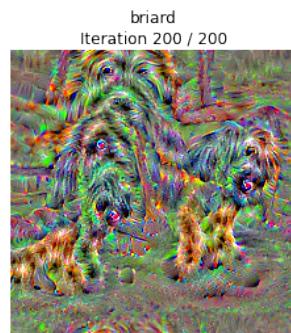


FIGURE 9 – Itérations=200, lr=5, l2=1e-3

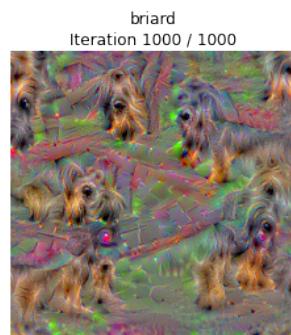


FIGURE 10 – Itérations=1000, lr=1, l2=1e-3

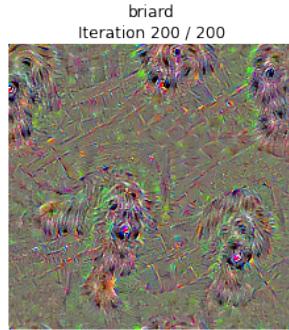


FIGURE 11 – Itérations=200, lr=5, l2=1

On remarque que plus la régularisation est faible plus l'image est psychédélique (**Fig. 9**). Et au contraire, si la régularisation est forte l'image est moins colorée (**Fig. 11**).

On a également augmenté le nombre d'itération et baisser le pas de gradient (**Fig. 10**) mais on observe pas d'amélioration.

**(Q10)** Au lieu de partir d'une image initialisée aléatoirement avec du bruit, on part maintenant d'une image du dataset ImageNet qui représente la classe qu'on veut illustrer.

Cette technique va amplifier les patterns déjà présent dans l'image, et donne un rendu bien meilleur car on converge mieux et rapidement qu'en partant d'une image aléatoire.

On peut distinguer dans la figure 12 des becs, des yeux, etc... des éléments qui caractérisent bien cette classe.

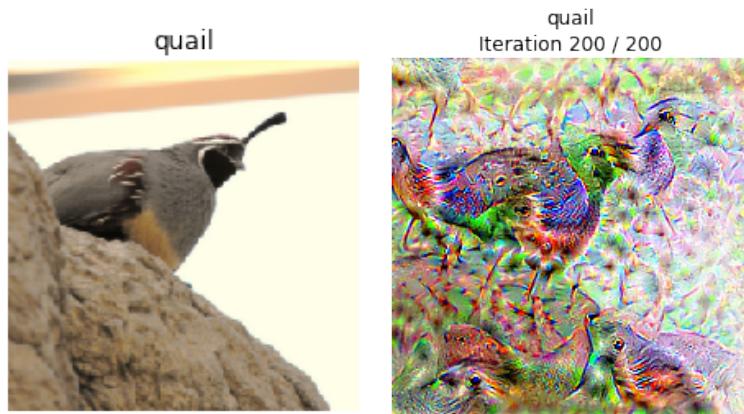


FIGURE 12 – Transformation en partant d'une image réelle

**(Q11)** On a effectué la même transformation que précédemment en utilisant les mêmes hyperparamètres, sauf que cette fois on a utilisé un **VGG16** à la place du **SqueezeNet1.1**.

On remarque qu'avec VGG on converge beaucoup mieux mais plus lentement qu'avec SqueezeNet. Mais le résultat est meilleur car on distingue mieux les traits caractéristiques.

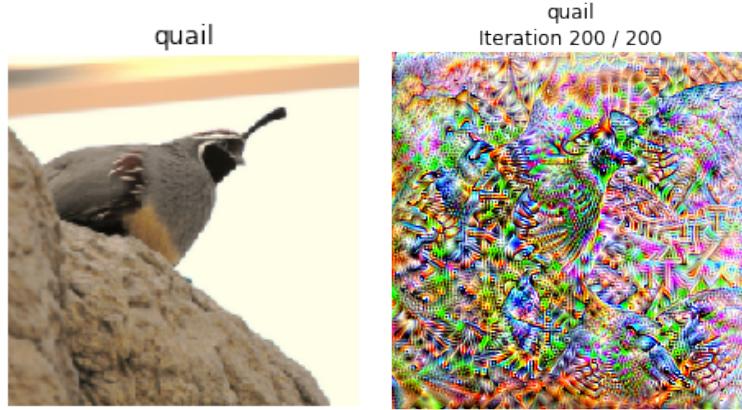


FIGURE 13 – Transformation en partant d'une image réelle avec VGG 16

### 3 Generative Adversarial Networks

#### 3.1 Principe général

**(Q1)** Les deux équations (6) et (7) correspondent aux objectifs des deux réseaux (descriminant et génératrice).

L'équation (6) correspond à l'objectif du réseau génératrice qui veut maximiser le fait de générer des images considérées réelles par le descritminant.

L'équation (7) correspond à l'objectif du descritminant qui veut maximiser d'une part la bonne classification des images réelles et d'autres part repérer les images générées.

Si on optimise uniquement le Descritminant, on pourra jamais générer d'image pouvant être considéré comme réelle. Et si on entraîne uniquement le Générateur, on apprend alors à générer une image avec un bruit adverse pour tromper le Descritminant mais cette image sera difficile à interpréter.

Les deux réseaux apprennent ensemble dans une relation gagnant-gagnant d'amélioration continue : le Générateur apprend à créer des images de plus en plus réalistes et le Discriminateur apprend à identifier de mieux en mieux les images réels de celle provenant du Générateur.

**(Q2)** Idéalement, le générateur G doit transformer la distribution  $P(z)$  en  $P(data)$ .

**(Q3)** La vraie équation pour G aurait été :

$$\min_G \mathbb{E}_{z \sim P(z)} \log(1 - D(G(z)))$$

### 3.2 Architectures des réseaux

**(Q4)**



FIGURE 14 – Après 500 steps



FIGURE 15 – Après 1000 steps

On peut remarquer que les images générées par le GAN sont bien des images de visages mais elles ne sont pas de très bonne qualité. On peut aussi noter que les images sont diversifiées et que ce n'est pas toujours les mêmes qui sont générées, ce qui signifie qu'on est pas entré en **mode collapse**.

On peut aussi remarquer qu'après 500 itérations on a distingué déjà des visages, et qu'entre 4000 et 8000 itérations il n'y a pas de différences notables.

Dans la figure 19 les loss sont difficilement interprétable, elles ne convergent pas vers 0 mais se trouvent dans un état d'équilibre dû aux objectifs opposés des 2 réseaux.

D'ailleurs il est important d'éviter que la loss du discriminateur converge à 0 car ça voudra dire que notre générateur n'apprend plus et ne s'améliore plus, car se dernier se corrige grâce aux images jugées *fake* par le Discriminant.



FIGURE 16 – Après 4000 steps



FIGURE 17 – Après 8000 steps

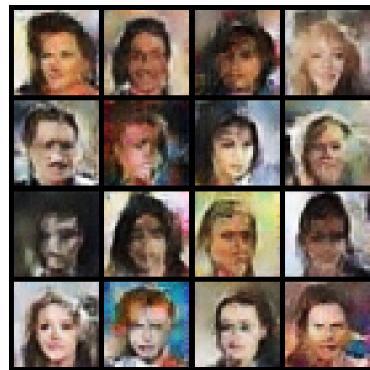


FIGURE 18 – Images générées par notre DCGAN

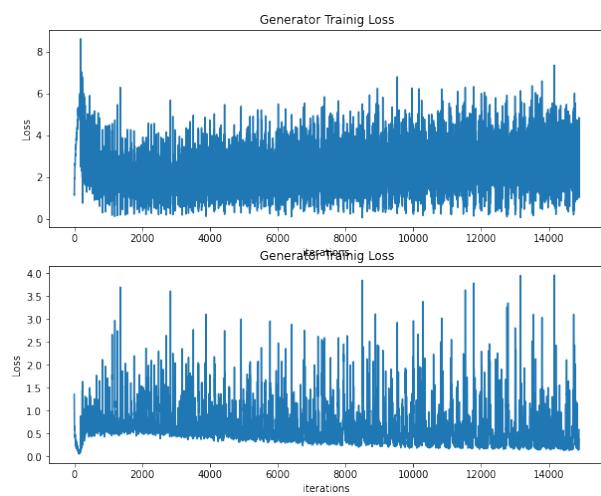


FIGURE 19 – Loss Générateur/Discriminant

**(Q5) Momentum :** En modifiant sa valeur de 0.5 à 0.9 on peut remarquer que la loss du réseau  $G$  lors des premières itérations augmente alors que logiquement elle devrait diminuer car les premières images générées sont plus facilement repérées par le réseau  $D$ .

Ceci est expliquer par le fait que le discriminant converge rapidement vers 0 ce qui signifie qu'il domine fortement le réseau  $G$  et ne lui permet pas de bien s'améliorer.

Un autre effet du réseau  $D$  très fort c'est les images générées à la figure 20 où on observe peu de diversité dans les images, on peut voir que c'est toujours une femme avec les mêmes traits de visage.

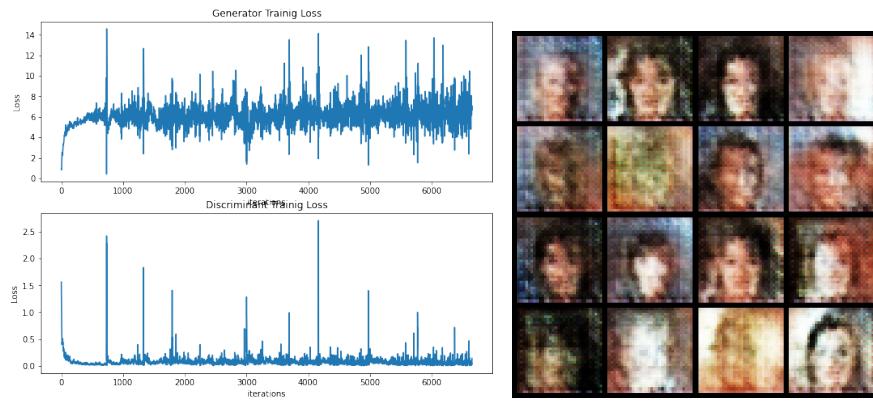


FIGURE 20 – DCGAN avec momentum à 0.9

**Implémentation de la fonction à la question (3) :** On remarque qu'après 4000 itérations on a toujours pas de génération satisfaisante. On peut donc en conclure que cette fonction converge beaucoup plus rapidement.

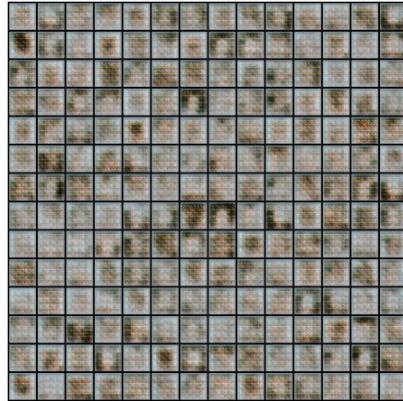


FIGURE 21 – Après 4000 steps



FIGURE 22 – Après 8000 steps

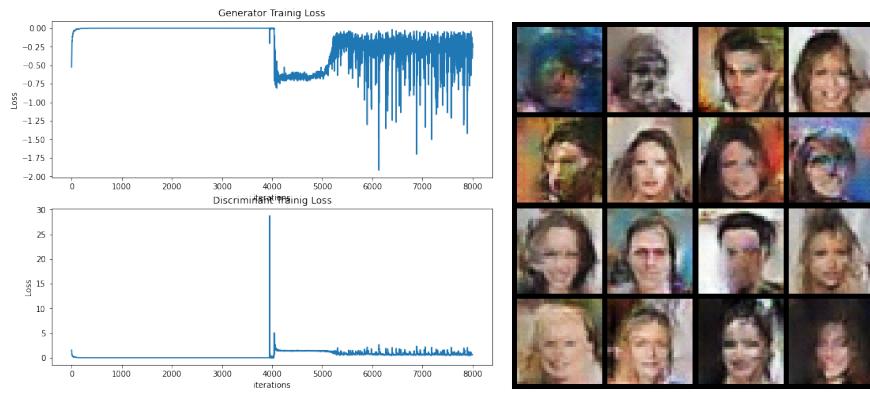


FIGURE 23 – DCGAN après changement de fonction loss

**Apprendre 5x plus le Discriminant :**

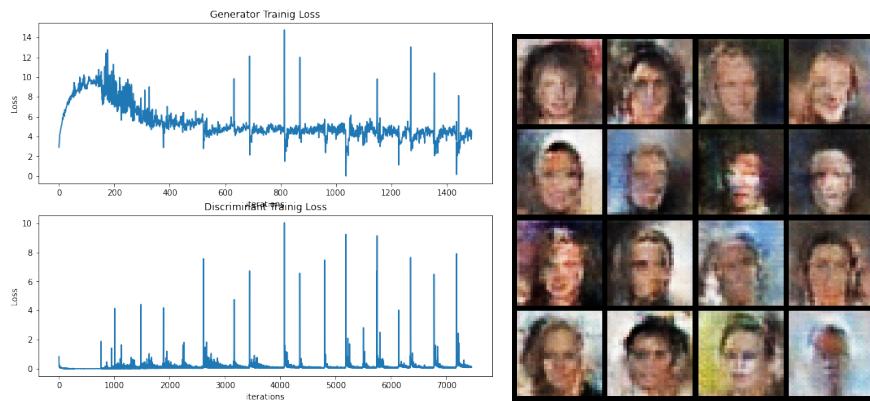


FIGURE 24 – MAJ du discriminant x5

### Apprendre 5x plus le Générateur :

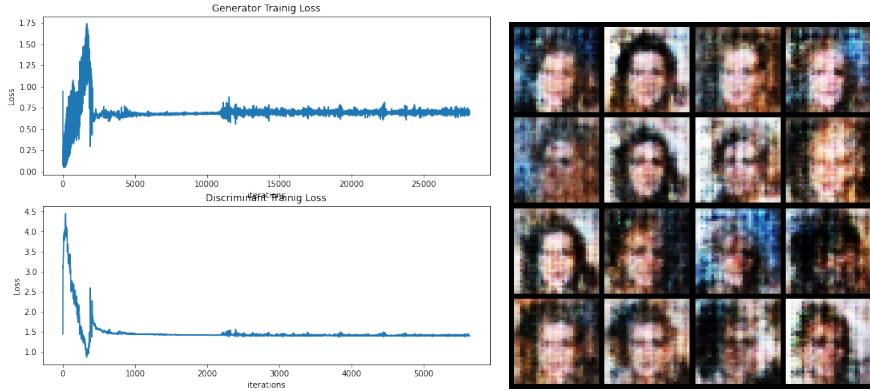


FIGURE 25 – MAJ du générateur x5

Dans les 2 variantes l'apprentissage est plus lent, avec un résultat final de moins bonne qualité qu'un apprentissage alterné.

### Variation de la taille du vecteur bruit Z :



FIGURE 26 – Vecteur Z de taille 10



FIGURE 27 – Vecteur Z de taille 1000

On note aucune différence avec les résultats qu'on avait précédemment, et on s'aperçoit également que l'augmentation de la taille du vecteur  $z$  ne signifie pas forcément amélioration des résultats.

**Dataset Celeba64** : on a modifié notre architecture pour prendre des images de dimension  $64 \times 64$ . Les images générées sont certes de hautes résolutions mais cela provoque plus d'erreur en génération contrairement à une

résolution plus faible, comme on le voit également la loss moyenne est plus élevé pour les 2 réseaux que sur le dataset *Celeba*.

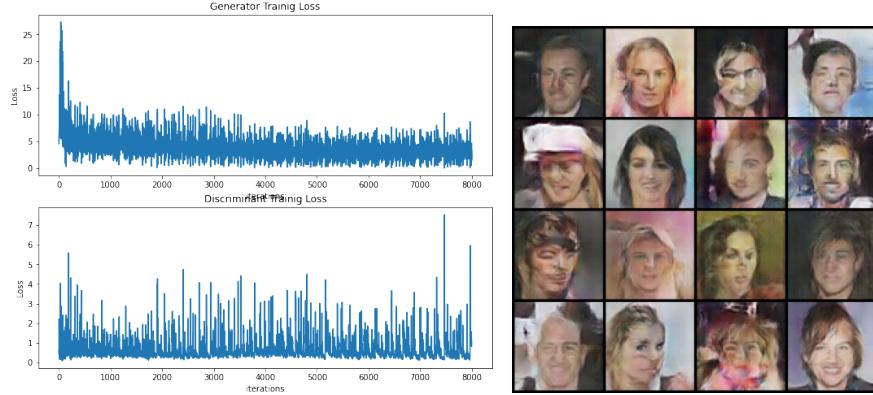


FIGURE 28 – DCGAN avec Celeba64

## 4 Conditional Generative Adversarial Networks

### 4.1 Principe général

**(Q6)** Dans le cas d'un cGAN on cherche à optimiser les équations suivantes :

$$\max_G \mathbb{E}_{z \sim P(z)} [\log(cD(cG(z, y), y))]$$

$$\max_D \mathbb{E}_{x^* \in Data} [\log(cD(x^*, y))] + \mathbb{E}_{z \sim P(z)} [\log(1 - cD(cG(z, y), y))]$$

**(Q7)** On peut conditioner ce modèle aux variables *age*(vieux, jeune) et *sex*(homme, femme).

**(Q8)** C'est un modèle qui génère une vidéo à partir d'une autre vidéo en changeant la saison d'hiver à été. Les variables de condition ici peuvent être neige, pluie, nuage, couleur du ciel...

**(Q9)** Les variables de conditions ici peuvent être tous les éléments relevés à la segmentation : piéton, voiture, arbre...

## 4.2 Architectures cDCGAN pour MNIST

**(Q10)** On remarque que l'apprentissage des deux réseaux est instable durant les 4000 premières itérations. Et à partir de 4000 itérations l'apprentissage se stabilise et on a de bien meilleurs résultats.

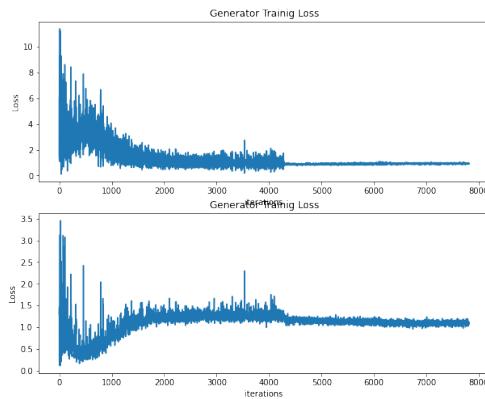


FIGURE 29 – cDCGAN loss

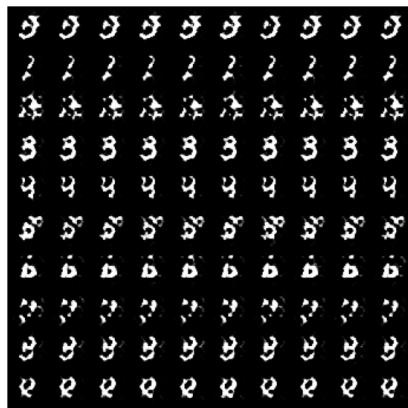


FIGURE 30 – cDCGAN - génération après 500 iter

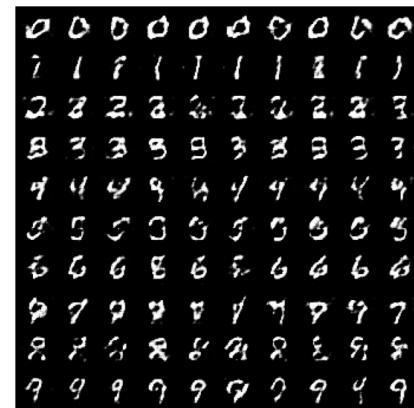


FIGURE 31 – cDCGAN - génération après 1000 iter

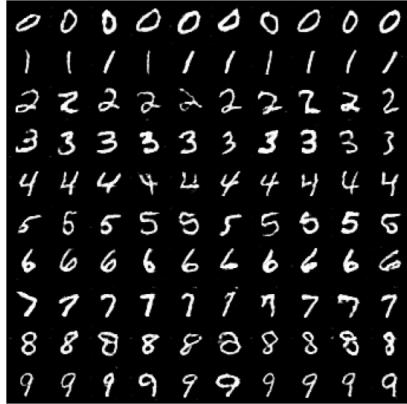


FIGURE 32 – cDCGAN - génération après 4000 iter

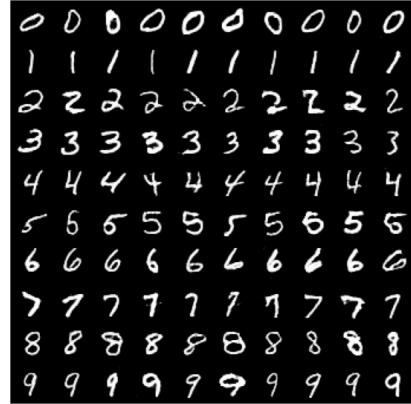


FIGURE 33 – cDCGAN - génération après 8000 iter

**(Q11)** Le Discriminant a besoin du vecteur  $y$  pour determiner si le Générateur a bien généré une image correspondant au  $y$ . Sans cette information, on a aucune garantie que les images générées correspondent à ce qui a été demandé au Générateur.

### 4.3 Architectures cGAN pour MNIST

**(Q12)** Lors de l'apprentissage, on a remarqué que la génération s'améliore jusqu'à l'itération 1000 puis se détériore jusqu'à l'itération 4000 et puis s'améliore de nouveau jusqu'à la fin.

Le rendu final du cGAN est de moins bonne qualité comparé au cDCGAN comme on le voit à la figure 37 certains chiffres sont illisibles.

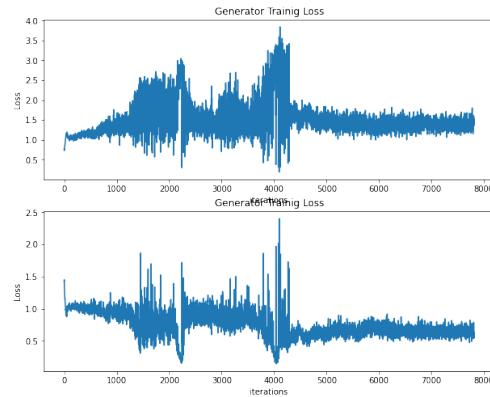


FIGURE 34 – cGAN loss



FIGURE 35 – cGAN - génération après 1000 iter

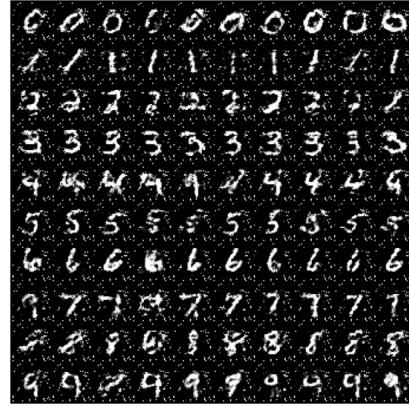


FIGURE 36 – cGAN - génération après 4000 iter



FIGURE 37 – cGAN - génération après 8000 iter

**(Q13)** Il est relativement plus difficile de générer des chiffres conditionnés avec un cGAN car il est composé uniquement de couche *fully-connected* ce qui entraîne une perte de la notion de spatialité d'une image. Au contraire d'un cDCGAN, qui lui utilise des couches de convolutions qui lui permettent d'avoir une compréhension globale de l'image.