



## MASTER DONNÉES, APPRENTISSAGE ET CONNAISSANCES-DAC

### RAPPORT TME 7-8-9-10 RDFIA

**Sujet :** TRANSFER LEARNING, VISUALIZATION ET GAN

REALISÉ PAR :

HANANE DJEDDAL  
LITICIA TOUZARI

# Table des matières

<b>1 TME 7 :</b>	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Partie 1 : Architecture VGG16 . . . . .	2
1.2.1 Réponse 1 : . . . . .	2
1.2.2 Réponse 2 : . . . . .	2
1.2.3 Réponse 3 : . . . . .	2
1.2.4 Réponse 4 : . . . . .	3
1.3 Partie 2 : Transfer Learning avec VGG16 sur 15 Scene . . . . .	3
1.3.1 Réponse 5 : . . . . .	3
1.3.2 Réponse 6 : . . . . .	3
1.3.3 Réponse 7 : . . . . .	4
1.3.4 Réponse 8 : . . . . .	4
1.3.5 Réponse 9 : . . . . .	4
1.3.6 Réponse 10 : . . . . .	4
1.3.7 Réponse 11 : . . . . .	4
<b>2 TME 8 : Visualisation des réseaux de neurones</b>	<b>5</b>
2.1 Introduction . . . . .	5
2.2 Partie 1 : Carte de saillance . . . . .	5
2.2.1 Réponse 1 : . . . . .	5
2.2.2 Réponse 2 : . . . . .	6
2.2.3 Réponse 3 : . . . . .	6
2.3 Partie 2 : Exemples adversaires . . . . .	6
2.3.1 Réponse 5 : . . . . .	6
2.3.2 Réponse 6 : . . . . .	7
2.4 Partie 3 : Visualisation de classes . . . . .	7
2.4.1 Réponse 8 : . . . . .	7
2.4.2 Réponse 9 : . . . . .	7
2.4.3 Réponse 10 : . . . . .	9
<b>3 TME 9 : Generative Adversarial Networks</b>	<b>9</b>
3.1 Introduction . . . . .	9
3.2 Partie 1 : Principe général . . . . .	9
3.2.1 Réponse 1 : . . . . .	9
3.2.2 Réponse 2 : . . . . .	10
3.2.3 Réponse 3 : . . . . .	10
3.3 Partie 2 : Architectures des réseaux . . . . .	10
3.3.1 Réponse 4 : . . . . .	10
3.3.2 Réponse 5 : . . . . .	10
<b>4 TME 10 : Conditional Generative Adversarial Networks</b>	<b>14</b>
4.1 Introduction . . . . .	14
4.1.1 Réponse 6 : . . . . .	14
4.1.2 Réponse 7 : . . . . .	14
4.1.3 Réponse 8 : . . . . .	14
4.1.4 Réponse 9 : . . . . .	14
4.1.5 Réponse 10 : . . . . .	14
4.1.6 Réponse 11 : . . . . .	16
4.1.7 Réponse 12 : . . . . .	16
4.1.8 Réponse 13 : . . . . .	17

# 1 TME 7 :

## 1.1 Introduction

Transfer Learning est une technique qui permet de transférer la connaissance acquise sur un jeu de données “source” pour mieux traiter un nouveau jeu de données dit “cible”. Dans ce TP on va travailler sur le réseau convolutif VGG16 et voir comment est ce qu’on pourrait le déployer pour accomplir une tâche de classification des scènes du jeu de données 15 Scene, avec un transfer Learning à partir de ImageNet.

## 1.2 Partie 1 : Architecture VGG16

VGG16 est une architecture du réseau de neurones convolutif proposée par Karen Simonyan and Andrew Zisserman pour la classification des images du dataset ImageNet. Le modèle prend en entrée une image en couleurs de taille  $224 \times 224$  px et la classifie dans une des 1000 classes. Dans cette partie, on va étudier cette architecture en utilisant le modèle pré-appris du pytorch.

### 1.2.1 Réponse 1 :

VGG-16 est constitué de plusieurs couches, dont 13 couches de convolution et 3 fully-connected. Il doit donc apprendre les poids de 16 couches. Chaque couche de convolution utilise des filtres en couleurs de taille  $3 \times 3$  px, déplacés avec un pas de 1 pixel. Le nombre de filtres varie selon le bloc dans lequel la couche se trouve. De plus, un paramètre de biais est introduit dans le produit de convolution pour chaque filtre.

Pour une couche de convolution prenant en entrée une feature map de taille  $l$  et qui donne en sortie une feature map de taille  $k$ , elle doit apprendre  $(3 \times 3 \times l) \times k$  paramètres, plus  $k$  paramètres de bias. Les deux premières couches fully-connected calculent chacune un vecteur de taille 4096. La dernière renvoie le vecteur de probabilités de taille 1000 (le nombre de classes) en appliquant la fonction softmax. De plus, ces trois couches utilisent aussi un paramètre de biais pour chaque élément du vecteur en sortie.

Bloc1 (2 convolution-64) :  $(3 \times 3 \times 3 + 1) \times 64 + (3 \times 3 \times 64 + 1) \times 64 = 38,720$   
Bloc2 (2 convolution-128) :  $(3 \times 3 \times 64 + 1) \times 128 + (3 \times 3 \times 128 + 1) \times 128 = 221,440$   
Bloc3 (3 convolution-256) :  $(3 \times 3 \times 128 + 1) \times 256 + (3 \times 3 \times 256 + 1) \times 256 \times 2 = 1,475,328$   
Bloc4 (3 convolution-512) :  $(3 \times 3 \times 256 + 1) \times 512 + (3 \times 3 \times 512 + 1) \times 512 \times 2 = 5,899,776$   
Bloc4 (3 convolution-512) :  $(3 \times 3 \times 512 + 1) \times 512 \times 3 = 7,079,424$   
fc1 :  $(512 \times 7 \times 7) \times 4,096 + 4,096 = 102,764,544$   
fc2 :  $4096 \times 4,096 + 4,096 = 16,781,312$   
fc3 :  $4096 \times 1,000 + 1,000 = 4,097,000$   
**Total** : 138,357,544

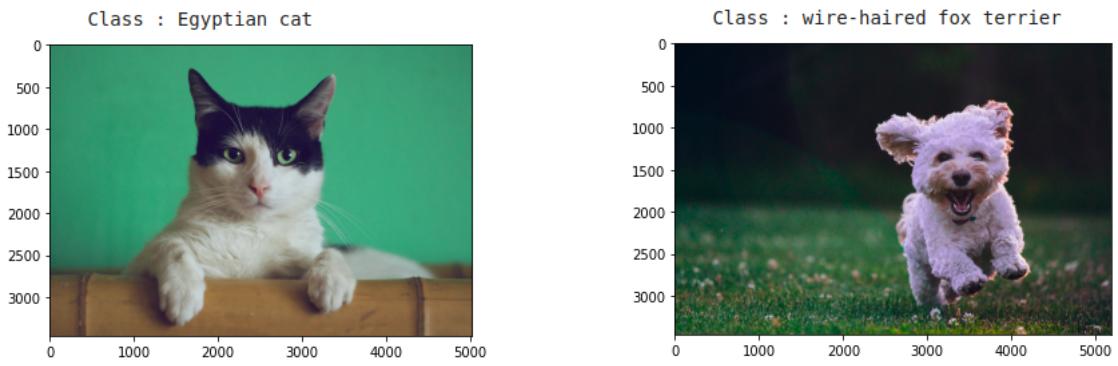
Donc le nombre de paramètres de VGG16 est 138 millions paramètres.

### 1.2.2 Réponse 2 :

La sortie de la dernière couche de VGG16 est un vecteur de taille 1000, qui contient les probabilités d’appartenance à chacune des classes.

### 1.2.3 Réponse 3 :

On applique le réseau VGG16 pré-appris du pytorch sur deux images : un chat et un chien. Le modèle retourne le type précis de chaque animal (Egyptian cat pour le chat et wire-haired fox terrier pour le chien) comme montré dans la figure 1.



(a) Image d'un chat classifiée comme Egyptien cat

(b) Image d'un chien classifiée comme wire-haired fox terrier

FIGURE 1 – Classification de deux images avec VGG16

#### 1.2.4 Réponse 4 :

La figure 2 montre les images correspondant à différentes cartes obtenues après la première convolution de VGG16. Les cartes sont le résultat de l'application d'un filtre sur l'image, ils représentent les caractéristiques les plus importantes que le filtre détecte.

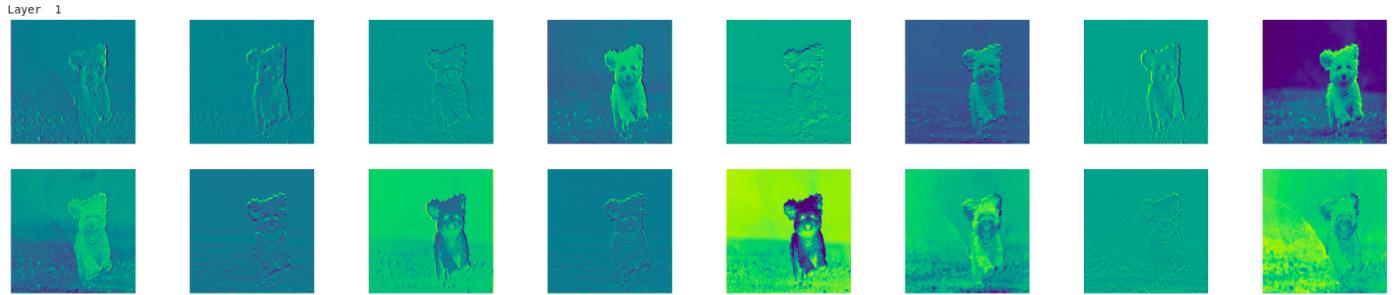


FIGURE 2 – Cartes de la première convolution appliquée sur l'image du chien

### 1.3 Partie 2 : Transfer Learning avec VGG16 sur 15 Scène

#### 1.3.1 Réponse 5 :

L'application du réseau VGG16 directement sur la base de données 15 scenes, implique que l'apprentissage des features se fait sur la base 15 scenes. Cette base, avec 15 classes de scenes et 210 à 410 images par classe, est d'une petite taille comparée à d'autres bases de données d'images, ce qui ne permet pas d'atteindre une bonne accuracy.

#### 1.3.2 Réponse 6 :

Le pré-apprentissage sur ImageNet va aider à extraire les régions d'importances d'une image (feature extraction). En passant une image de la base 15 Scenes par le réseau pré-appris, on obtiendra une représentation vectorielle de l'image décrivant son contenu. Comme ImageNet est beaucoup plus riche que 15 scenes, le réseau appris sur cette base donnera des meilleurs résultats.

### **1.3.3 Réponse 7 :**

Cette approche par feature extraction n'est utilisable que lorsque le jeu de données source et target sont plus ou moins similaires. Par exemple on ne peut pas utiliser un réseau appris sur des images des animaux de dessin animé pour classifier des images de fruits.

### **1.3.4 Réponse 8 :**

Le choix de la couche à laquelle les features sont extraites est très important. Les premières couches ne sont pas très utiles car le modèle n'a pas encore bien appris les features ; et dans les dernières couches, les features seront trop spécifiques aux données sources. Il faut donc trouver un compromis.

### **1.3.5 Réponse 9 :**

VGG16 attend des images RGB alors que les images de 15 Scene sont en noir et blanc. Pour cela, l'image est recopiée 3 fois, une pour chaque canal RGB.

### **1.3.6 Réponse 10 :**

C'est possible de n'utiliser que le réseau de neurones pour classifier les images plutôt que d'apprendre un classifieur indépendant. Pour cela, il faut charger le modèle pré-appris et puis réinitialiser la dernière couche FC (ou la couche où on veut extraire les features) ; on change la taille du vecteur de sortie (15 classes au lieu de 1000) et on lance l'apprentissage de cette dernière couche de classification sur le jeu de données 15 Scene.

### **1.3.7 Réponse 11 :**

On introduit quelques modifications sur le schéma de classification créé pour voir leurs effets sur les performances.

Dans un premier temps, on change la couche à laquelle sont extraites les features. La table 1 montre les accuracy obtenues pour chaque modification. La meilleure performance est obtenue avec une extraction à la 2ieme couche ReLu car à ce niveau les features sont plus générales. L'accuracy est stable à partir de la couche Relu7.

On garde le modèle avec l'extraction à la couche Relu7 et on varie les valeur du paramètre C de SVM. La figure 3 montre l'accuracy obtenue. La meilleure performance est pour C=1 ce qui implique que les données sont plus ou moins séparables.

On teste après, le modèle en utilisant AlexNet au lieu de VGG16 avec l'extraction à la couche conv 3 , l'accuracy obtenue est 0.86 qui est moins bonne que celle obtenue avec VGG16.

TABLE 1 – Comparaison des méthodes naïves

Modèle	Accuracy	
Modèle de base (VGG16 + SVM)	Extraction à la couche Relu2	0.906868
	Extraction à la couche Relu4	0.886097
	Extraction à la couche Relu7	0.888107
	Extraction à la couche Relu10	0.888107
AlexNet + SVM	0.868342	

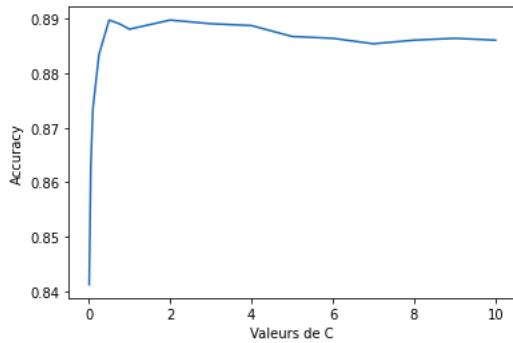


FIGURE 3 – Accuracy du modèle en variant le paramètre C de SVM

## 2 TME 8 : Visualisation des réseaux de neurones

### 2.1 Introduction

Dans ce TP, nous étudions diverses techniques 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.

On prend donc un réseau appris et on fige ses poids et on modifie l'image d'entrée afin d'étudier indirectement le comportement du réseau. On utilise dans un premier temps le réseau SqueezeNet appris sur ImageNet et nous verrons 3 façon d'étudier le réseau.

### 2.2 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.

#### 2.2.1 Réponse 1 :

Après exécution, on obtient les cartes de saillance de nos images, on remarque que les pixels les plus importants (en rouge) correspondent généralement aux pixels de l'objet mis en valeur dans l'image (marche bien pour l'image du chien mais moins bien pour quelque images comme stole).

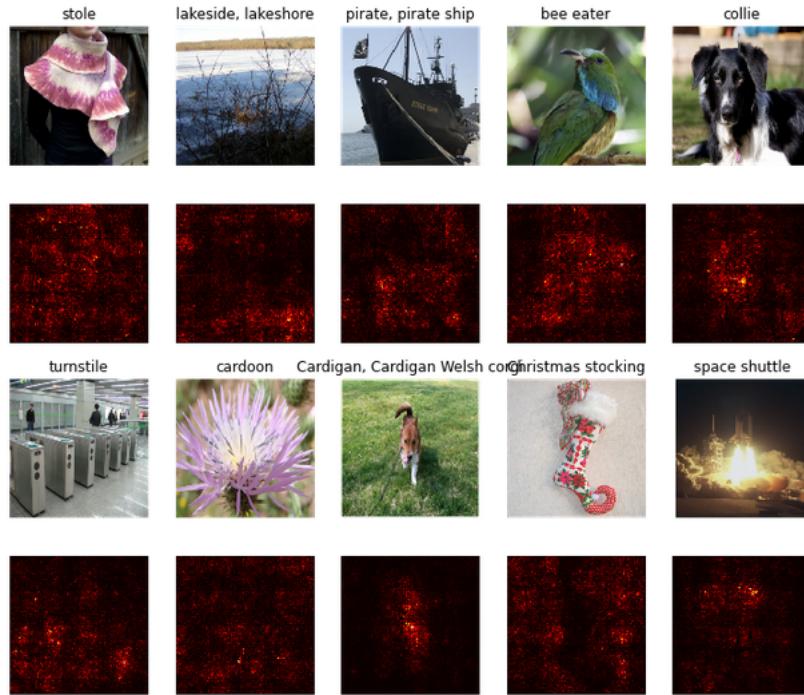


FIGURE 4 – Exemples de cartes de saillance

### 2.2.2 Réponse 2 :

L'une des limitations de cet algorithme est que la carte de saillance obtenue à partir de la rétropropagation du gradient est très bruyante en l'état.

### 2.2.3 Réponse 3 :

Une autre application envisageable pour carte de saillance est la segmentation d'images.

## 2.3 Partie 2 : Exemples adversaires

En 2014, Szegedy et al. ont découvert que de nombreux modèles d'apprentissage automatique, y compris la réalisation de réseaux de neurones performants de pointe, sont très vulnérables aux attaques adverses soigneusement élaborées, c'est-à-dire basées sur des modifications mineures de l'entrée. Dans la suite du tp, nous aborderons une méthode pour y parvenir.

### 2.3.1 Réponse 5 :

Goodfellow et coll. ont proposé d'ajouter un petit vecteur de bruit soigneusement conçu afin de tromper le réseau neuronal. Ci-dessous vous pouvez voir un exemple. Nous commençons par l'image correctement classée comme "quail". Après avoir ajouté un bruit imperceptible, le réseau neuronal a reconnu l'image comme "stingray" tout en gardant une image similaire à l'oeil nu, les figures de droites montrent les pixels modifiés.

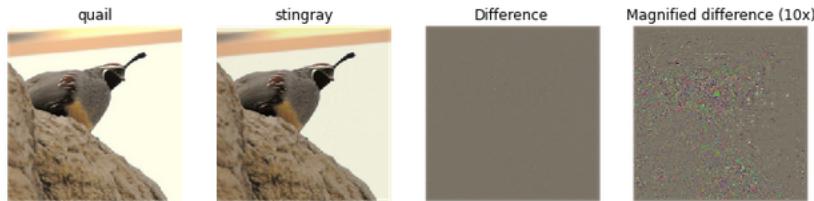


FIGURE 5 – Exemple d'une image adversaire

### 2.3.2 Réponse 6 :

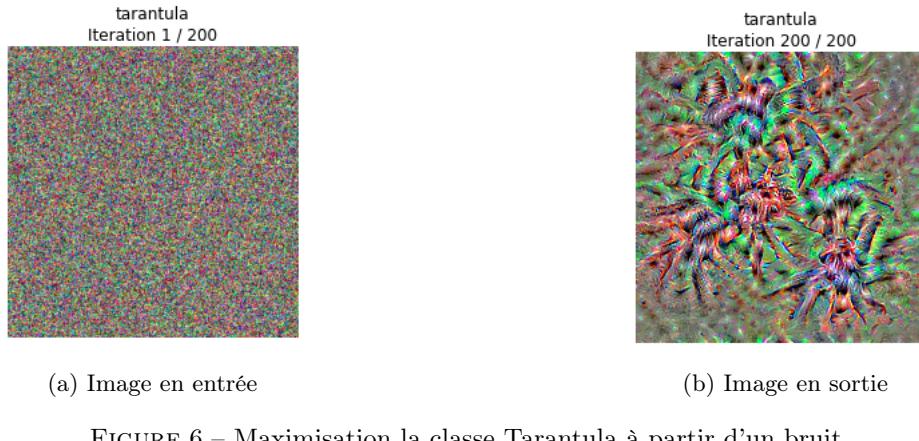
Il a été prouvé que les réseaux de neurones sont vulnérables aux attaques adverses. Cela remet en question leur applicabilité dans des scénarios du monde réel, en particulier dans les systèmes où assurer la sécurité est crucial. Il est donc très important de pouvoir se défendre contre de telles attaques.

## 2.4 Partie 3 : Visualisation de classes

Dans cette partie, on explore une approche simple afin de visualiser le type de patterns susceptible de produire une classe particulière en sortie du réseau.

### 2.4.1 Réponse 8 :

A partir d'une image contenant que du bruit aléatoire, on essaie de calculer une image maximisant le score d'une classe. Dans les exemples ci-dessous on essaie de maximiser le score de la classe Tarantula, on remarque qu'on obtient bien des formes de tarentules à la fin.



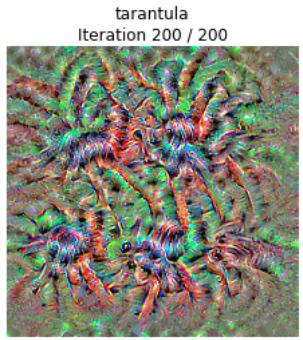
(a) Image en entrée

(b) Image en sortie

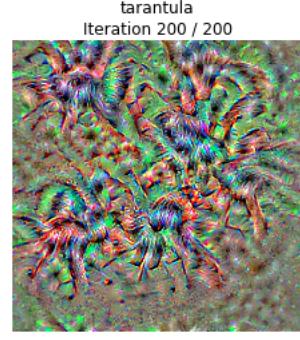
FIGURE 6 – Maximisation la classe Tarantula à partir d'un bruit

### 2.4.2 Réponse 9 :

On essaie de varier le nombre d'itérations, le learning rate et le poids de la régularisation, les images ci-dessous illustrent les résultats obtenus :



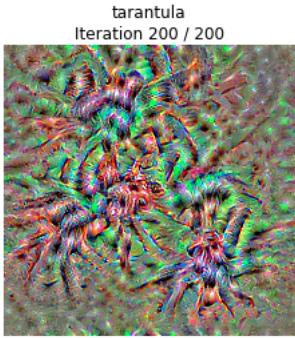
(a) Poids de la régularisation : 0.1



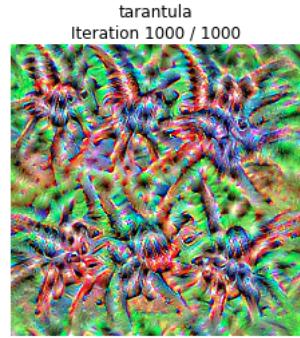
(b) Poids de la régularisation : 1e-10

FIGURE 7 – Variation du poids de la régularisation

La variation du poids de régularisation n'afflue pas beaucoup sur les résultats obtenus. On remarque tout de même que pour un poids très petit les contour de l'image ne sont pas modifiés, et les corps des araignées sont mieux délimités.



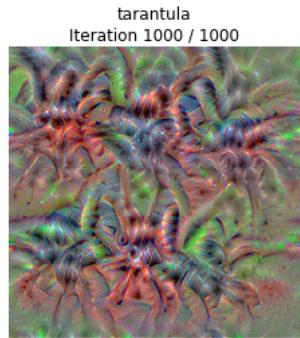
(a) Nombre d'itérations : 200



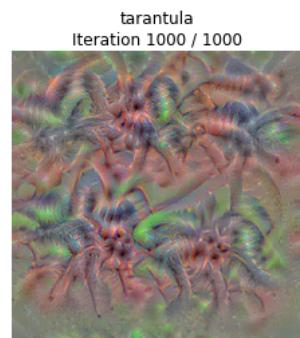
(b) Nombre d'itérations : 1000

FIGURE 8 – Variation du nombre d'itérations

On remarque que pour un nombre d'itérations très grand toute l'image est modifiée et on remarque plus de pattes et de corps d'araignées qu'avec un nombre d'itération plus petit.



(a) Learning rate : 1



(b) Learning rate : 0.5

FIGURE 9 – Variation du learning rate

Diminuer le learning rate permet d'affiner un peu plus l'image, on distingue mieux les corps des tarantules.

#### 2.4.3 Réponse 10 :

On refait les mêmes traitements que pour l'exemple précédent en remplaçant l'image de départ par une image d'ImageNet. On remarque que l'attention est portée sur les objets snail puisque c'est ce qui permet de classifier l'image. Maximiser le score de la classe de l'image permet d'améliorer la qualité des résultats et de reconnaître facilement la classe de l'image.

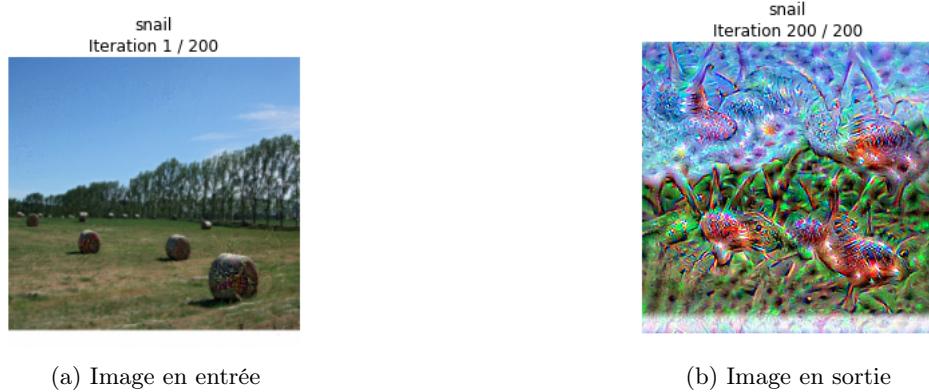


FIGURE 10 – Maximisation de la classe à partir d'une image de Snail

## 3 TME 9 : Generative Adversarial Networks

### 3.1 Introduction

Dans ce TP, nous allons nous intéresser à des modèles génératifs qui modélisent traditionnellement la probabilité jointe  $P(X, Y)$  et donc la vraisemblance des données  $X$ . En particulier, nous allons nous intéresser aux Generative Adversarial Networks (GAN).

### 3.2 Partie 1 : Principe général

#### 3.2.1 Réponse 1 :

Durant l'apprentissage d'un GAN on essaie d'optimiser le problème suivant :  $\min_G \max_D \mathbb{E}_{x^* \in Data} [\log(D(x^*))] + \mathbb{E}_{z \sim P(z)} [1 - \log(D(G(z)))]$

D'un point de vue pratique, on alterne entre l'apprentissage du générateur et du discriminateur, qui ont chacun une fonction objectif différente dérivée du problème ci-dessus :

1.  $\max_G \mathbb{E}_{z \sim P(z)} [\log(D(G(z)))]$
2.  $\max_D \mathbb{E}_{x^* \in Data} [\log(D(x^*))] + \mathbb{E}_{z \sim P(z)} [1 - \log(D(G(z)))]$

Pour l'équation du générateur on essaie de minimiser  $\min_G \mathbb{E}_{x^* \in Data} [\log(D(x^*))] + \mathbb{E}_{z \sim P(z)} [1 - \log(D(G(z)))]$  ce qui revient à minimiser  $\min_G \mathbb{E}_{z \sim P(z)} [1 - \log(D(G(z)))]$  et donc à maximiser  $\max_G \mathbb{E}_{z \sim P(z)} [\log(D(G(z)))]$  entre autre on voudrait que le générateur échoue à faire la différence entre les vraies images et les images générées par le générateur.

Pour l'équation du discriminateur on essaie de maximiser  $D(x^*)$  et minimiser  $D(G(z))$  entre autre on veut que le discriminateur réussisse à distinguer les vraies images des fausses.

### 3.2.2 Réponse 2 :

Le vecteur  $z$  est tiré d'une distribution de probabilité antérieure  $P(z)$  puis utilisé pour conditionner une autre distribution de probabilité  $P_G$  de sorte que les échantillons qui en sont tirés soient similaires aux images réelles. Donc la distribution de probabilité est choisie depuis le début et ne change jamais, et à chaque fois que nous alimentons une entrée au générateur, nous échantillonons un nouveau vecteur à partir de cette distribution, c'est-à-dire que nous alimentons un vecteur de nombre généré aléatoirement.

### 3.2.3 Réponse 3 :

La vraie équation du générateur est :  $\min_G \mathbb{E}_{x^* \in Data} [\log(D(x^*))] + \mathbb{E}_{z \sim P(z)} [1 - \log(D(G(z)))]$

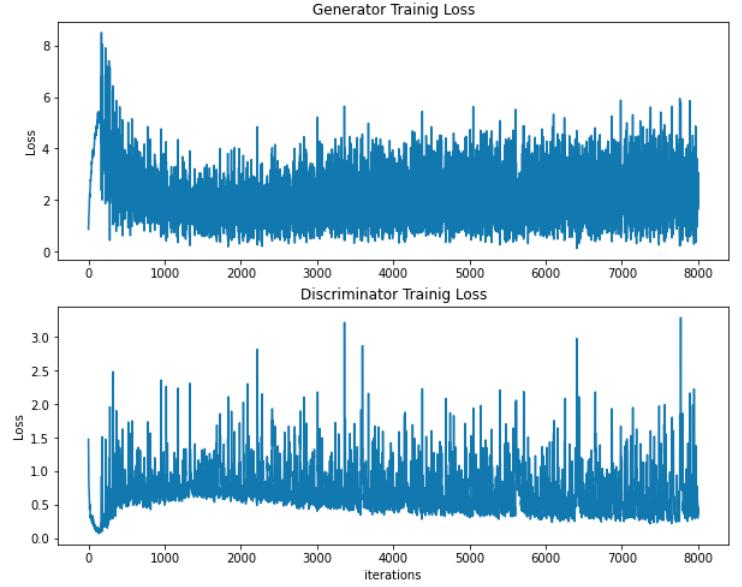
## 3.3 Partie 2 : Architectures des réseaux

### 3.3.1 Réponse 4 :

On entraîne notre réseau avec les paramètres donnés, on obtient des images plutôt bonnes et diverses. Les courbes de loss du générateur et du discriminateur ne sont pas stables puisque les deux réseaux sont adversaires, quand la loss du générateur diminue celle du discriminateur augmente mais puisque ce derniers essaye à son tour de diminuer sa loss celle du générateur augmente et vis-versa.



(a) Résultats du GAN



(b) Loss du générateur et du discriminateur

FIGURE 11 – Résultats obtenus avec les paramètres par défaut

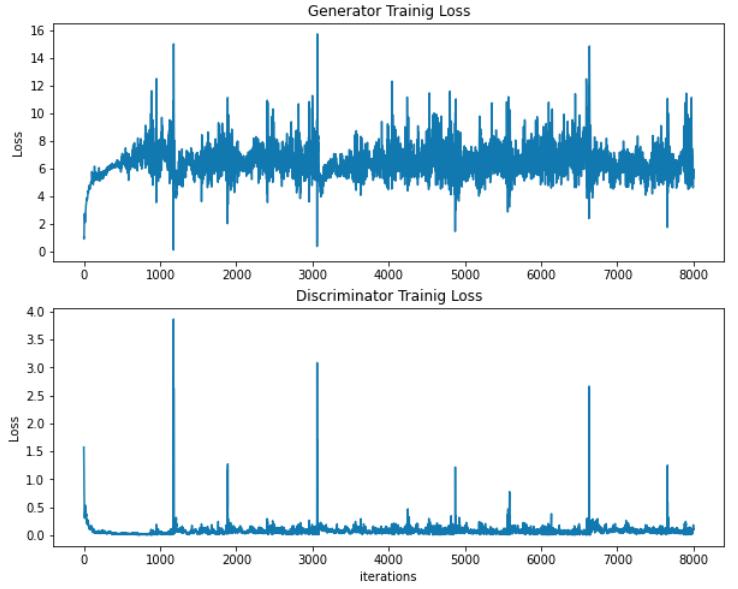
### 3.3.2 Réponse 5 :

On essaie d'explorer quelques hyperparamètres de notre réseau GAN pour voir leurs effets sur les résultats obtenus :

- **Momentum = 0.9** : Avec une valeur 0.9 pour le momentum du générateur et discriminateur, on obtient de mauvais résultats comparés à ceux obtenus avec une valeur 0.5. La loss du discriminateur est souvent presque nulle contrairement à celle du générateur qui reste élevée. Les images générées par le générateur sont mauvaises et de ce fait facilement reconnaissables par le discriminateur.



(a) Résultats du GAN



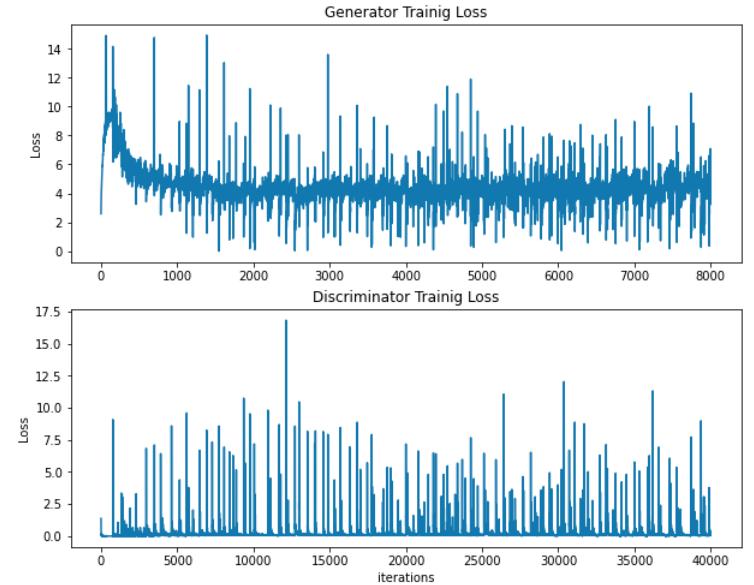
(b) Loss du générateur et du discriminateur

FIGURE 12 – Résultats obtenus pour momentum = 0.9

- **nb\_update\_D = 5 nb\_update\_G = 1 :** Pour un nombre de sous-étapes de l'optimizer à chaque étape égale à 1 pour le générateur et 5 pour le discriminateur les images obtenues sont de bonne qualité et très variées.



(a) Résultats du GAN

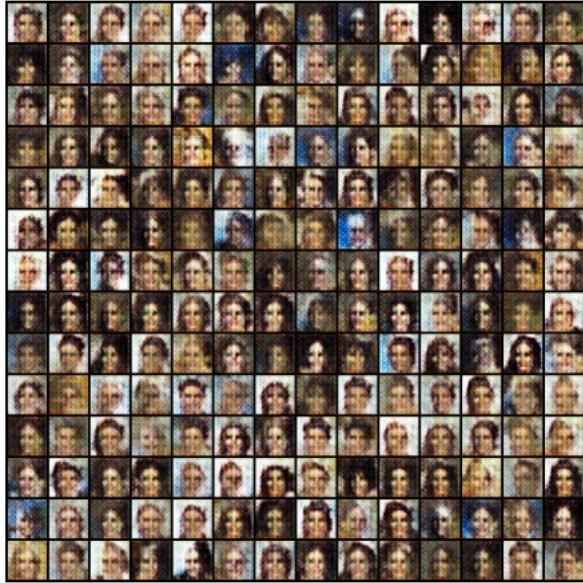


(b) Loss du générateur et du discriminateur

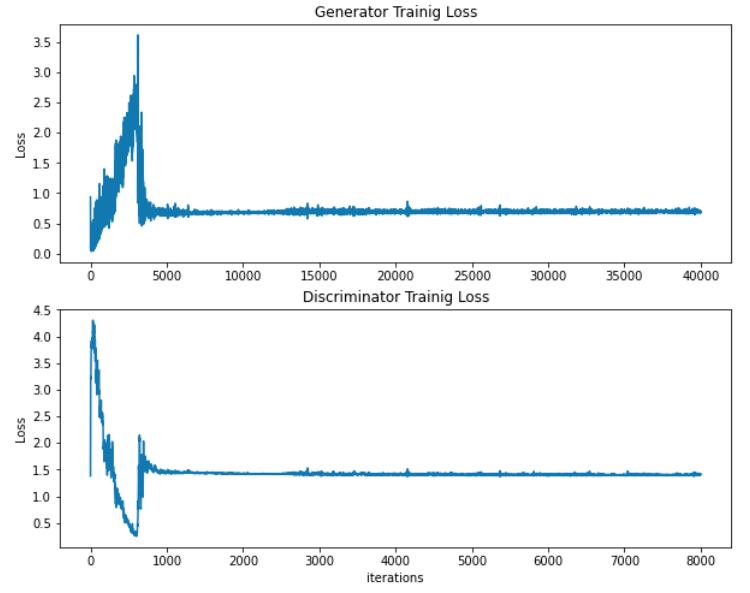
FIGURE 13 – Résultats obtenus pour nb\_update\_D = 5 nb\_update\_G = 1

- **nb\_update\_D = 1 nb\_update\_G = 5 :** Pour un nombre de sous-étapes de l'optimizer à chaque

étape égale à 5 pour le générateur et 1 pour le discriminateur on obtient des images plutôt de mauvaise qualité et pas très variées. Les deux loss ne sont pas stables, la loss du discriminateur est nulle sur plusieurs epochs puisqu'il fait 5 updates à chaque fois donc il apprends mieux à différencier les images avant que le générateur ne puisse corriger ses erreurs, la loss du générateur quant à elle est moins bonne puisqu'il a du mal à tromper le discriminateur qui a toujours une longueur d'avance sur lui.



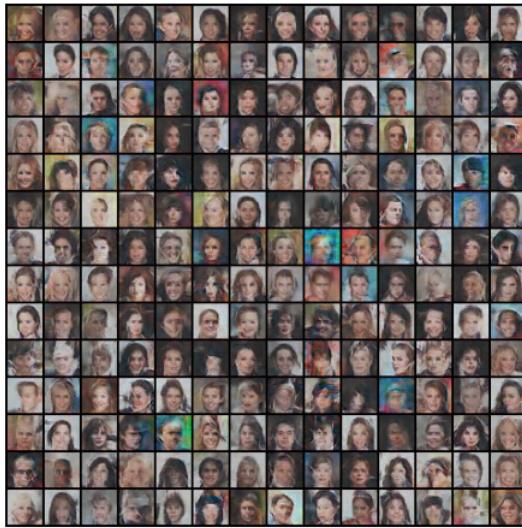
(a) Résultats du GAN



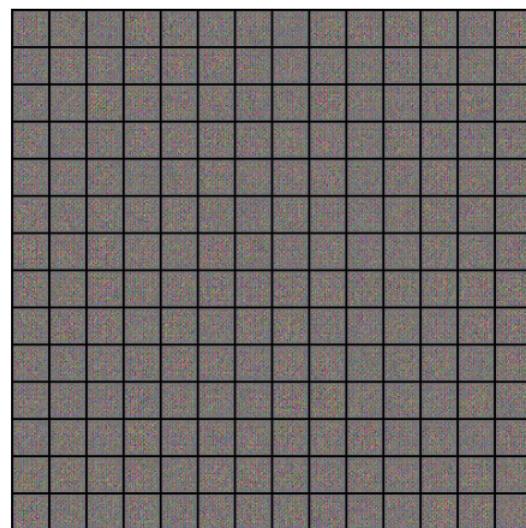
(b) Loss du générateur et du discriminateur

FIGURE 14 – Résultats obtenus pour nb\_update\_D = 1 nb\_update\_G = 5

-- lr = 0.1 et lr = 1e-10 On effectue des tests sur des valeurs de pas d'apprentissage différents les résultats obtenus sont représentés ci-dessous. Pour un pas d'apprentissage 0.1 les résultats sont plus au moins bon et diversifié, par contre pour un pas d'apprentissage très petit, on ne génère pas d'images.



(a) Learning rate = 0.1



(b) Learning rate = 1e-10

FIGURE 15 – Résultats obtenus pour des learning rate différents

-- **nz = 10 et nz = 1000** On effectue des tests sur des dimensions de couches cachées différentes (10, 1000). Les résultats obtenus avec des dimensions nz 10 et 1000 ont l'air bon dans les deux cas.



(a) nz = 10



(b) nz = 1000

FIGURE 16 – Résultats obtenus pour des valeurs de nz différents

## 4 TME 10 : Conditional Generative Adversarial Networks

### 4.1 Introduction

Le GAN conditionnel est un type de GAN où une contrainte est ajouté au problème d'optimisation. Dans le GAN, il n'y a aucun contrôle sur les données à générer. Le GAN conditionnel, par conséquent, ajoute un label  $y$  comme paramètre supplémentaire au générateur et ce label peut être modifié une fois l'apprentissage est terminée pour générer une diversité de caractéristiques. Dans ce TP, on implémente un modèle cDCGAN et un modèle cGAN pour générer les chiffres de MNIST.

#### 4.1.1 Réponse 6 :

Dans le cas de cGAN, On cherche donc à optimiser le problème suivant :

$$\min_G \max_D \mathbb{E}_{x* \in Data} [\log D(x*|y)] + \mathbb{E}_{z \sim P(z)} [1 - \log(D(G(z|y)))] \quad (1)$$

C'est à dire la fonction objectif du générateur et celle du discriminateur sont définie :

1.  $\max_G \mathbb{E}_{z \sim P(z)} [\log(D(G(z|y)))]$
2.  $\max_D \mathbb{E}_{x* \in Data} [\log(D(x*|y))] + \mathbb{E}_{z \sim P(z)} [1 - \log(D(G(z|y)))]$

#### 4.1.2 Réponse 7 :

Le générateur de la figure 6 est conditionné aux variables : âge (vieux / jeune) et le sexe (femme / homme).

#### 4.1.3 Réponse 8 :

Le générateur de la vidéo est conditionné à la saison (hiver/ été)

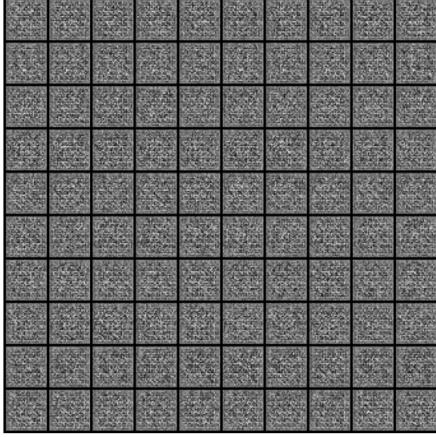
#### 4.1.4 Réponse 9 :

Le générateur de la vidéo est conditionné à des caractéristiques sémantiques qui décrivent une ville (voitures qui circulent et leurs positions, bâtiments etc). Ces descriptions sont introduites sous formes de cartes de segmentation.

#### 4.1.5 Réponse 10 :

La figure 17 montre la génération des chiffres de Mnist avec cDCGAN. Dans les premières étapes, le modèle ne génère que du bruit, puis il arrive à générer des chiffres bruités. La qualité de ces chiffres s'améliore au fur des époches, et à partir du step 5625, les chiffres générées sont de très bonne qualité, et la loss est stable (voir figure 18).

[ 0/ 7812] Loss\_D: 2.4141 Loss\_G: 0.0190 D(x): 0.1300 D(G(z)): 0.1306



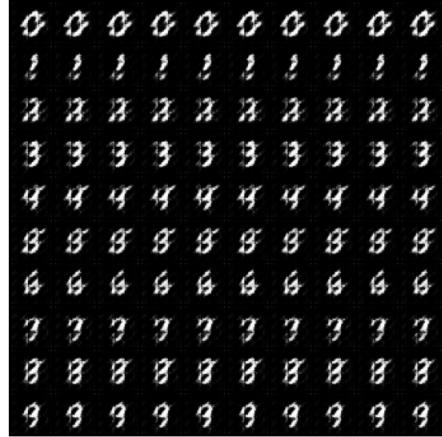
(a) Step 1

[ 5625/ 7812] Loss\_D: 1.1824 Loss\_G: 0.9234 D(x): 0.5365 D(G(z)): 0.4070  
 [ 5650/ 7812] Loss\_D: 1.1308 Loss\_G: 0.8725 D(x): 0.5810 D(G(z)): 0.4255  
 [ 5675/ 7812] Loss\_D: 1.1415 Loss\_G: 0.8734 D(x): 0.5889 D(G(z)): 0.4371  
 [ 5700/ 7812] Loss\_D: 1.1409 Loss\_G: 0.8802 D(x): 0.5840 D(G(z)): 0.4335



(c) Step 5625

[ 25/ 7812] Loss\_D: 1.6727 Loss\_G: 0.1309 D(x): 0.2505 D(G(z)): 0.0126  
 [ 50/ 7812] Loss\_D: 1.1510 Loss\_G: 4.3594 D(x): 0.8469 D(G(z)): 0.5838  
 [ 75/ 7812] Loss\_D: 1.2304 Loss\_G: 1.3424 D(x): 0.4641 D(G(z)): 0.1489  
 [ 100/ 7812] Loss\_D: 2.0913 Loss\_G: 0.8509 D(x): 0.2097 D(G(z)): 0.0440



(b) Step 25

[ 7725/ 7812] Loss\_D: 1.1761 Loss\_G: 0.8951 D(x): 0.5515 D(G(z)): 0.4241  
 [ 7750/ 7812] Loss\_D: 1.1310 Loss\_G: 0.8904 D(x): 0.5794 D(G(z)): 0.4242  
 [ 7775/ 7812] Loss\_D: 1.1083 Loss\_G: 0.8493 D(x): 0.5765 D(G(z)): 0.4073  
 [ 7800/ 7812] Loss\_D: 1.0880 Loss\_G: 0.8983 D(x): 0.5835 D(G(z)): 0.4052



(d) Step 7818

FIGURE 17 – Génération des chiffres de Mnist avec cDCGAN

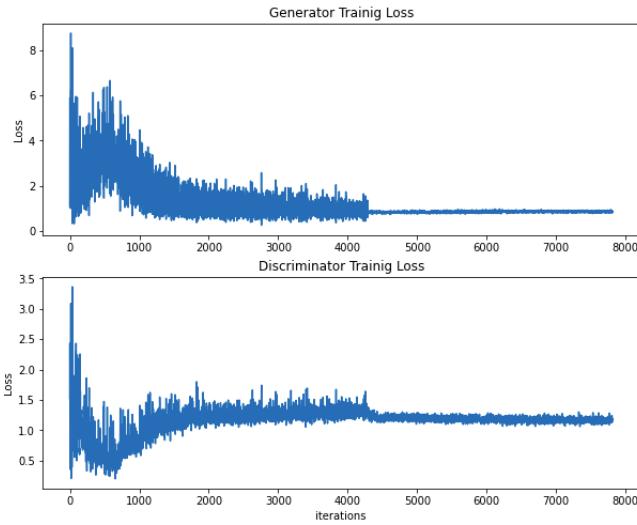


FIGURE 18 – Loss du Générateur et Discriminateur cDCGAN

#### 4.1.6 Réponse 11 :

Il est important que le discriminateur et le générateur ont accès au label  $y$  pour générer un chiffre désiré. Si on enlève le label  $y$  des entrées du discriminateur, le modèle ne va pas apprendre à générer sous une condition. Les chiffres générés seront aléatoires comme un GAN normal.

#### 4.1.7 Réponse 12 :

La figure 19 montre la génération des chiffres de MNIST avec cGAN. Le modèle prend beaucoup plus d'étapes pour générer des chiffres : il ne génère que du bruit jusqu'à l'étape 5000 où les images générées commencent à ressembler un peu à des chiffres. Après 11700 étapes, les chiffres générées restent de mauvaise qualité. La figure 20 montre la loss du générateur et discriminateur.

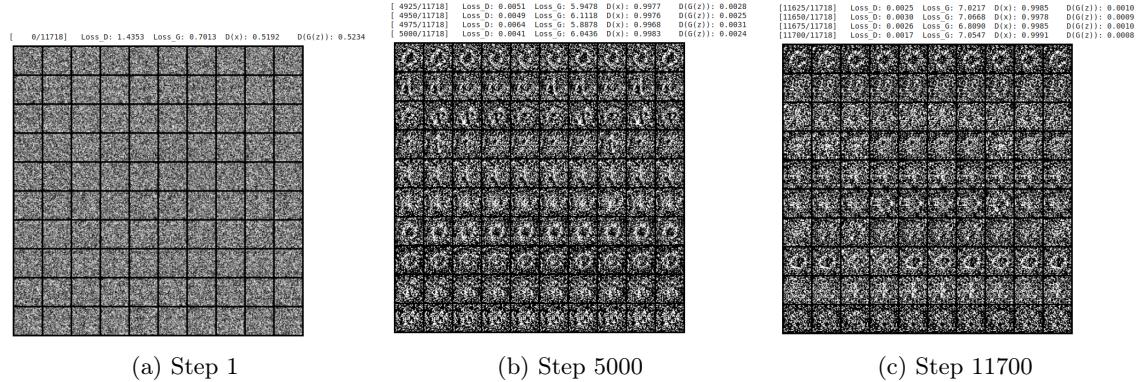


FIGURE 19 – Génération des chiffres de Mnist avec cGAN

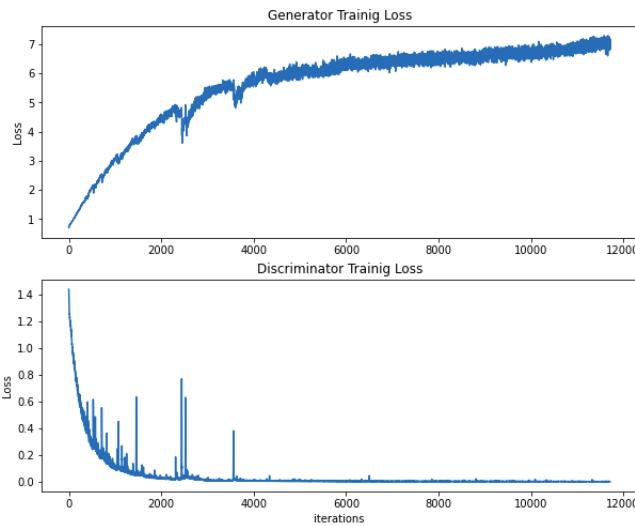


FIGURE 20 – Loss du Générateur et Discriminateur cGAN

#### 4.1.8 Réponse 13 :

Il est relativement plus difficile de générer des chiffres conditionnés avec un cGAN qu'avec un cDCGAN car avec un cGAN, on perd la notion de localité (patterns locaux), alors que les réseaux convolutionnels arrivent à mieux utiliser la structure de l'image. En plus, il y a plus de poids à apprendre avec les dense layers qu'avec la convolution.