

FACULTÉ DES SCIENCES ET INGÉNIERIE  
SORBONNE UNIVERSITÉ

RECONNAISSANCE DES FORMES POUR L'ANALYSE ET  
L'INTERPRÉTATION D'IMAGES

---

## Rapport de TMEs 8 à 11

---

*Auteur :*  
Ahmed Tidiane BALDÉ

*Encadrants :*  
Arthur DOUILLARD  
Yifu CHEN

6 janvier 2020



## TABLE DES MATIÈRES

---

<b>Transfer Learning par extraction de features dans un CNN</b>	<b>3</b>
<b>Partie 1 - Architecture VGG16</b>	<b>3</b>
Q1.1 . . . . .	3
Q1.2 . . . . .	3
Q1.3 . . . . .	3
<b>Partie 2 - Transfer Learning avec VGG16 sur 15 Scene</b>	<b>4</b>
2.1     Principe de la démarche . . . . .	4
Q2-1.5 . . . . .	4
Q2-1.6 . . . . .	4
Q2-1.7 . . . . .	4
2.2     Extraction des features de VGG16 . . . . .	5
Q2-2.8 . . . . .	5
Q2-2.9 . . . . .	5
2.3     Apprentissage de classifieurs SVM . . . . .	5
Q2-3.10 . . . . .	5
Q2-3.11 . . . . .	5
2.4     Aller plus loin . . . . .	5
Q2-4.12 . . . . .	5
<b>Visualisation des réseaux de neurones</b>	<b>6</b>
<b>Partie 1 - Carte de saillance</b>	<b>6</b>
Q1.1 . . . . .	6
Q1.2 . . . . .	6
Q1.3 . . . . .	7
Q1.4 . . . . .	7
<b>Partie 2 - Exemples adversaires</b>	<b>8</b>
Q2.5 . . . . .	8
Q2.6 . . . . .	8
Q2.7 . . . . .	8
<b>Partie 3 - Visualisation de classes</b>	<b>9</b>
Q3.8 . . . . .	9
Q3.9 . . . . .	9
Q3.10 . . . . .	10
Q3.11 . . . . .	10
<b>Generative Adversarial Networks</b>	<b>13</b>

<b>Partie 1 - Generative Adversarial Networks</b>	<b>13</b>
1.1    Principe général . . . . .	13
Q1-1.1 . . . . .	13
Q1-1.2 . . . . .	13
Q1-1.3 . . . . .	13
1.2    Architectures des réseaux . . . . .	13
Q1-2.4 . . . . .	13
Q1-2.5 . . . . .	15
<b>Partie 2 - Conditional Generative Adversarial Networks</b>	<b>15</b>
2.1    Principe général . . . . .	16
Q2-1.6 . . . . .	16
Q2-1.7 . . . . .	16
Q2-1.8 . . . . .	16
Q2-1.9 . . . . .	16
2.2    Architectures cDCGAN pour MNIST . . . . .	16
Q2-2.10 . . . . .	16
Q2-2.11 . . . . .	18
2.3    Architectures cGAN pour MNIST . . . . .	18
Q2-3.12 . . . . .	18
Q2-3.13 . . . . .	19
<b>Références</b>	<b>20</b>

# TRANSFER LEARNING PAR EXTRACTION DE FEATURES DANS UN CNN

Avec l'essor de la *Computer Vision*, les scientifiques tout comme les amateurs se sont vus confrontés à de nouveaux problèmes, qui sont d'une part le manque de ressources matérielles pour l'entraînement de ces réseaux de neurones profonds, et d'autre part la difficulté à récolter des *datasets* suffisamment larges et structurées pour atteindre des résultats significatifs. C'est alors que la méthode de **Transfer Learning** fait son entrée. Plutôt que de réentraîner des modèles à chaque nouvelle tâche, il s'agit ici d'utiliser un modèle précédemment appris mais dont les domaines d'application présentent des similitudes à celui que nous souhaitons traiter, puis, finalement, d'adapter le modèle en l'amputant à un certain niveau de son architecture, dans le but de pouvoir utiliser les caractéristiques de généralisation apprises au préalable ainsi que l'extraction de *features* plus complexes.

## PARTIE 1 - ARCHITECTURE VGG16

---

### Q1.1

Comme nous l'avions vu auparavant, l'une des différences les plus marquantes entre les convolutions et les *fully-connected* réside dans le nombre de paramètres à apprendre. Estimons ci-dessous le nombre de paramètres du **VGG16** :

- FC1 :  $7 \times 7 \times 512 \times 1 \times 1 \times 4096 + 4096 = 102.764.544$
- FC2 :  $1 \times 1 \times 4096 \times 1 \times 1 \times 4096 + 4096 = 16.781.312$
- FC3 :  $1 \times 1 \times 4096 \times 1 \times 1 \times 1000 + 1000 = 4.097.000$

Nous nous retrouvons alors avec un total de 123.642.856 de paramètres que pour les couches *fully-connected*.

### Q1.2

La taille de la dernière couche du réseau **VGG16** est de 1000, ce qui correspond naturellement au nombre de classes et plus particulièrement à la foi que le réseau attribue à chacune d'entre elles quant à la classification d'une nouvelle image.

### Q1.3

Quelques images classifiées par le réseau **VGG16**, ci-dessous.

Les deux dernières images sont bien classifiées. Quant à la première, nous avons une prédiction incorrecte.



pinwheel

'racer, race car, racing car'

'timber wolf, grey wolf, gray wolf, Canis lupus'

*learning\_rate : 1, num\_iterations : 500, l2\_reg=1e-1*

## PARTIE 2 - TRANSFER LEARNING AVEC VGG16 SUR 15 SCENE

### 2.1 PRINCIPE DE LA DÉMARCHE

#### Q2-1.5

D'une part, apprendre directement un si grand réseau sur un nouveau *dataset* demanderait non seulement trop de temps mais aussi des ressources matérielles conséquentes. Et d'autre part, rappelons que **VGG16** a été appris sur *ImageNet*. En l'occurrence, *15-Scene* est relativement trop petit, et le réseau s'exposerait très probablement à du sur-apprentissage.

#### Q2-1.6

Ci-haut, nous avons mentionné que pour adapter un réseau pré-appris à un nouveau problème, il faut qu'il y ait des similitudes. Or non seulement ces deux *datasets* ont des données similaires, mais le pré-apprentissage sur *ImageNet* est d'un atout considérable sur *15-Scene* car les premières convolutions apprennent des *features* plutôt générales notamment les contours et les formes des images, ces premiers filtres sont aussi connues sous le nom de *filtres de Gabor*. Plus on avance en profondeur dans les convolutions, plus elles deviennent spécifiques au problème. On pourrait alors amputer le réseau au niveau de ces premières convolutions, et ainsi nous servir des caractéristiques d'extraction générales apprises sur *ImageNet*.

#### Q2-1.7

Les limites se font sentir par l'impossibilité de *fine-tuner* les poids de notre réseau ici, en l'occurrence le SVM nous permet pas de garantir les optimisations par descente de gradient. Un second problème serait de trouver des domaines assez similaires pour justement assurer que le modèle transféré peut bien s'adapter au nouveau modèle.

## 2.2 EXTRACTION DES FEATURES DE VGG16

### Q2-2.8

Comme nous l'avions mentionné un peu plus haut, les premières couches de convolution extraient les contours, donc des caractéristiques générales et plus on avance en profondeur, plus les convolutions apprennent à détecter des objets et se spécialisent au problème. Pour choisir la couche à laquelle nous allons amputer le modèle, nous pouvons faire ce qu'on appelle une *ablation study*, quand bien même le principe reste de trouver un compromis de généralisation du modèle.

### Q2-2.9

Il suffirait de dupliquer le même pixel pour les trois canaux.

## 2.3 APPRENTISSAGE DE CLASSIFIEURS SVM

### Q2-3.10

Nous obtenons des résultats légèrement meilleurs que ceux obtenus avec l'approche *SIFT + BOW*.

### Q2-3.11

Il est tout à fait possible de n'utiliser que les réseaux de neurones au lieu de faire appel à un classifieur indépendant. On pourrait tout simplement remplacer la couche de classification par un réseau de neurones. Cela dit, il convient de prendre en compte le temps de calcul, mais aussi faire preuve d'attention afin de ne pas complètement oublier ce que les premières convolutions ont apprises et en même temps apprendre suffisamment sur le nouveau problème. Notons que cette pratique est utilisée quand les deux problèmes sont très distincts et qu'il est impératif de réapprendre les poids pour justifier de résultats performants.

## 2.4 ALLER PLUS LOIN

### Q2-4.12

Nous avons testé le modèle **VGG16** avec un simple classifieur linéaire **SVM** en variant le paramètre  $C$  de ce dernier et quelques fois, l'indice de coupure.

En terme de temps d'exécution, le **VGG16** n'est pas le meilleur modèle, d'autres modèles comme **Alex-Net** ou **SqueezeNet** aurait été nettement plus rapide. Le classifieur **SVM** est tout aussi long à apprendre. Quant à la variation des hyperparamètres, nous remarquons que le meilleur modèle est celui dont l'hyperparamètre  $C$  est le plus faible, 0.001. Finalement, nous remarquons également que supprimer quelques convolutions diminue l'accuracy de peu quand bien même cela accélère l'exécution tout naturellement.

Index of ablation	C	Accuracy
-	100	0.899
-	10	0.899
-	1	0.899
-	0.1	0.900
-	0.01	0.905
-	0.001	0.913
-2	0.001	0.8825
-4	0.001	0.8813

## VISUALISATION DES RÉSEAUX DE NEURONES

La puissance des réseaux de neurones, leur habileté à résoudre des problèmes complexes dans une infinité de domaines en arborant des performances notables, leur ont propulsé sur le devant de la scène depuis maintenant plusieurs années. Toutefois, leur usage dans un but industriel demeure encore être une réticence notamment pour la simple raison que nous ne sommes pas encore capables d'expliquer avec précision le pourquoi d'une de ces décisions. Leur explicabilité est d'ailleurs aujourd'hui un domaine d'étude en lui-même. Cette problématique est plus connu sous le nom de *boîte noire*.

### PARTIE 1 - CARTE DE SAILLANCE

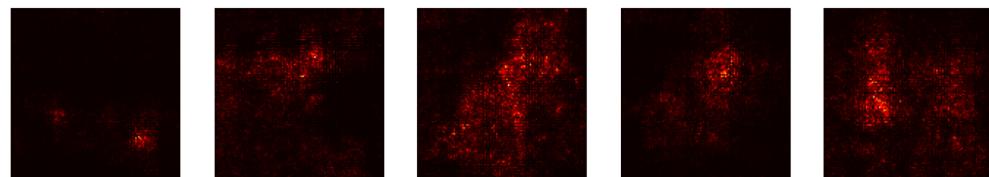
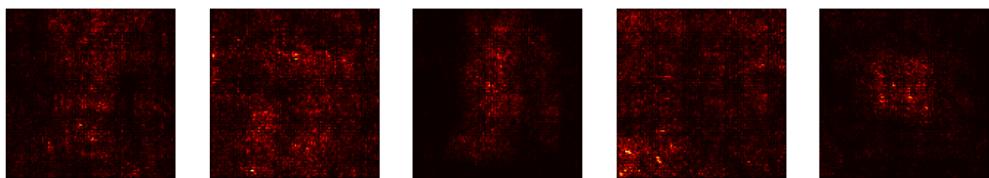
---

#### Q1.1

La carte de saillance est alors l'une des techniques nous permettant de visualiser les réseaux de neurones pour savoir quels sont les pixels qui ont été les plus impliqués dans la décision. Ci-dessous quelques cartes de saillance.

#### Q1.2

Une des limites des cartes de saillance serait le fait qu'elles ne représentent pas de manière juste tous les pixels qui ont été impliqués dans la décision. Ici, nous remarquons que certains pixels sont éteints alors qu'ils auraient dû être allumés et inversement. D'une part, nous justifions les pixels éteints par le *pooling*, rappelons qu'une fois le filtre de pooling appliqué, un seul pixel est retenu, ce qui donc expliquerait l'absence de plusieurs autres pixels quand bien même importants. D'autre part, en prenant par exemple la classe *pyjama*, ou *ours*, nous pouvons remarquer que plusieurs autres pixels sont allumés, le lit, la glace, respectivement, ce qui pourrait également s'expliquer par un biais d'apprentissage dans la base de données (les pyjamas étant le plus souvent portés dans la chambre, les ours et la glace, etc...)

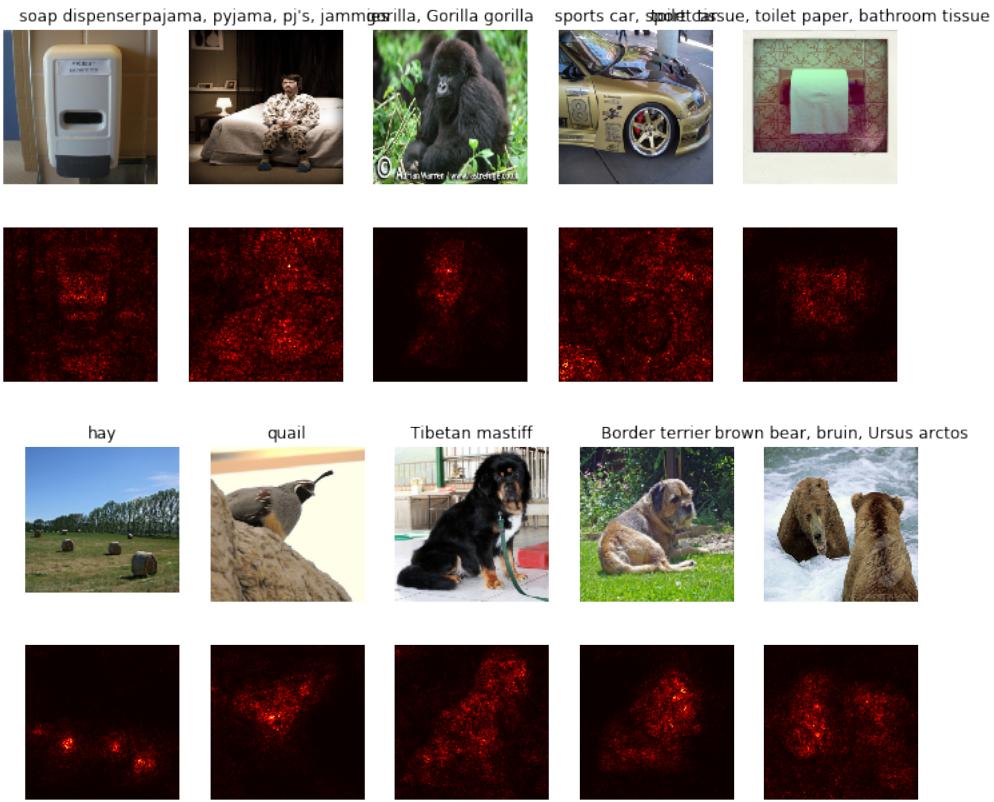


### Q1.3

Cette technique pourrait être utilisée à des fins de segmentation.

### Q1.4

Ci-dessous, nous avons généré les cartes de saillance en utilisant **VGG16**, notons que ce modèle est beaucoup plus précis dans ses contours ainsi que dans le choix des pixels les plus importants dans la décision. À titre d'exemple, comparé au **SqueezeNet**, les pixels des quatre (4) foin sont bien visibles.



## PARTIE 2 - EXEMPLES ADVERSAIRES

---

### Q2.5

La classe cible est atteinte avec succès au bout de seulement quelques itérations. Ces modifications trompent clairement notre réseau de neurones et sont complètement indiscernables à l'oeil nu.

### Q2.6

Les conséquences pourraient être considérables. Dans le cadre des voitures autonomes, son comportement pourrait donc théoriquement être piraté, considéré les feux de signalisation comme des arbres, les lignes blanches comme des trottoirs, etc... Nous extrapolons beaucoup certes, mais les conséquences de potentielles actions prises par une voiture autonome dont la perception est modifiée sont indénombrables.

### Q2.7

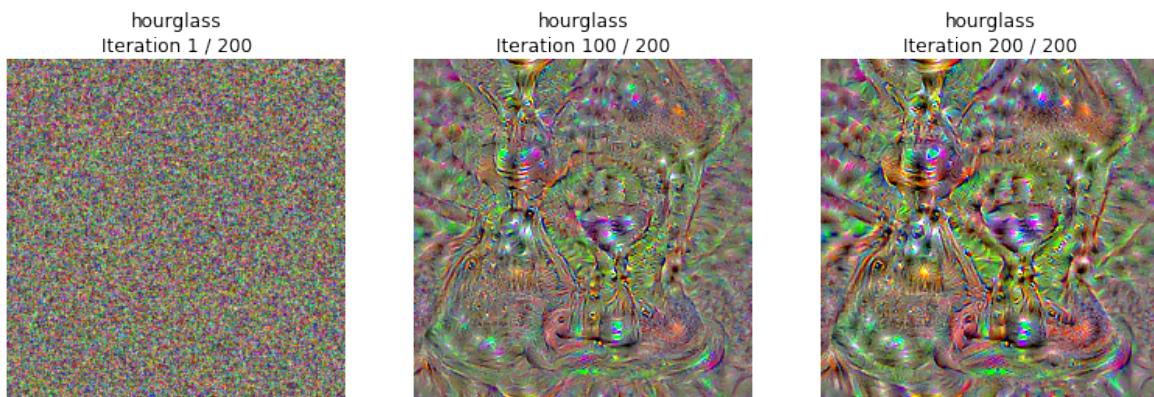
Sur des bases de données comme *MNIST*, les modifications effectuées par cette méthode pourraient être reconnu à l'oeil nu par un humain, car la méthode modifie un grand nombre de pixels. Des chercheurs de l'université de Berkeley ont proposé un papier Liu et al. (2016) sur la transférabilité des exemples adversaires créés pour un modèle à un autre modèle. Il s'agit donc ici de minimiser la quantité de modifications apportées pour tromper un modèle. L'attaque par *black box* en utilisant par exemple l'approche Fast Gradient Sign serait une proposition viable.



## PARTIE 3 - VISUALISATION DE CLASSES

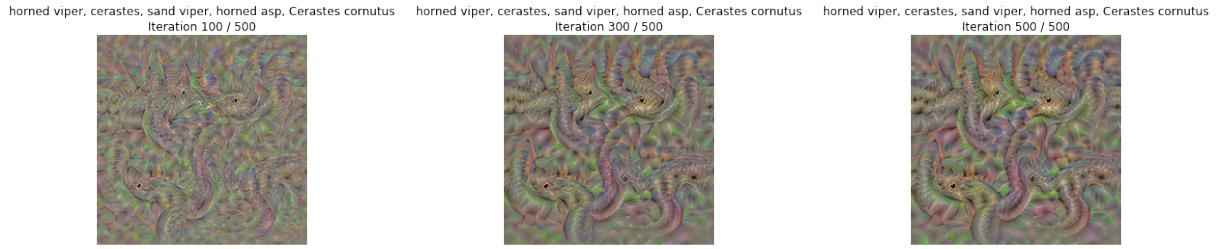
### Q3.8

Nous obtenons des résultats qui sont représentent de manière assez honnête les classes cibles. Ci-dessous en partant d'un bruit, au bout de 200 itérations, nous arrivons à distinguer les contours des objets que représentent la classe cible.



### Q3.9

En variant les paramètres, nous obtenons des résultats encore plus satisfaisants. Notons notamment, que l'augmentation du *learning rate* enrichie les images créées en couleur et amplifie par conséquent les contours. Quant au nombre d'itérations, nous remarquons qu'il ne fait pas varier les résultats tant que ça à partir de 200. Nous concluons que le modèle atteint donc convergence quand l'image est assez psychédélique.



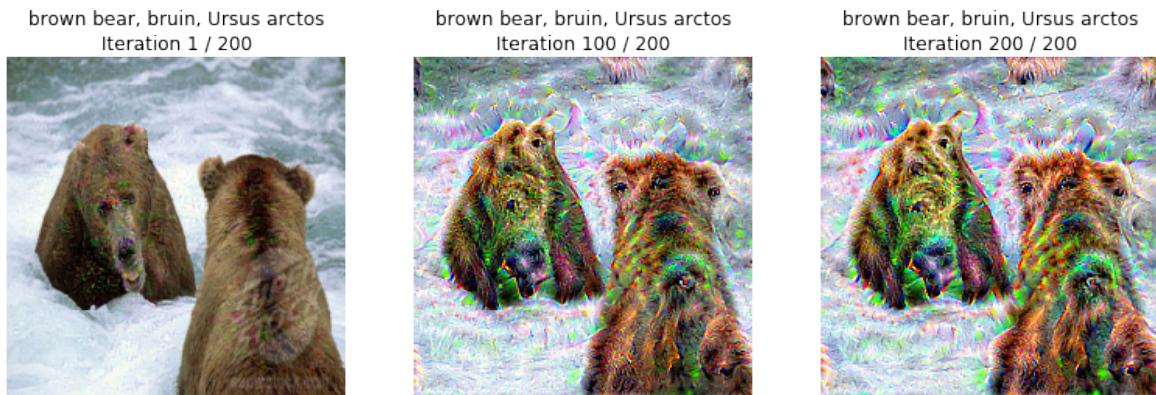
*learning\_rate : 1, num\_iterations : 500, l2\_reg=1e-1*



*learning\_rate : 10, num\_iterations : 500, l2\_reg=1e-1*

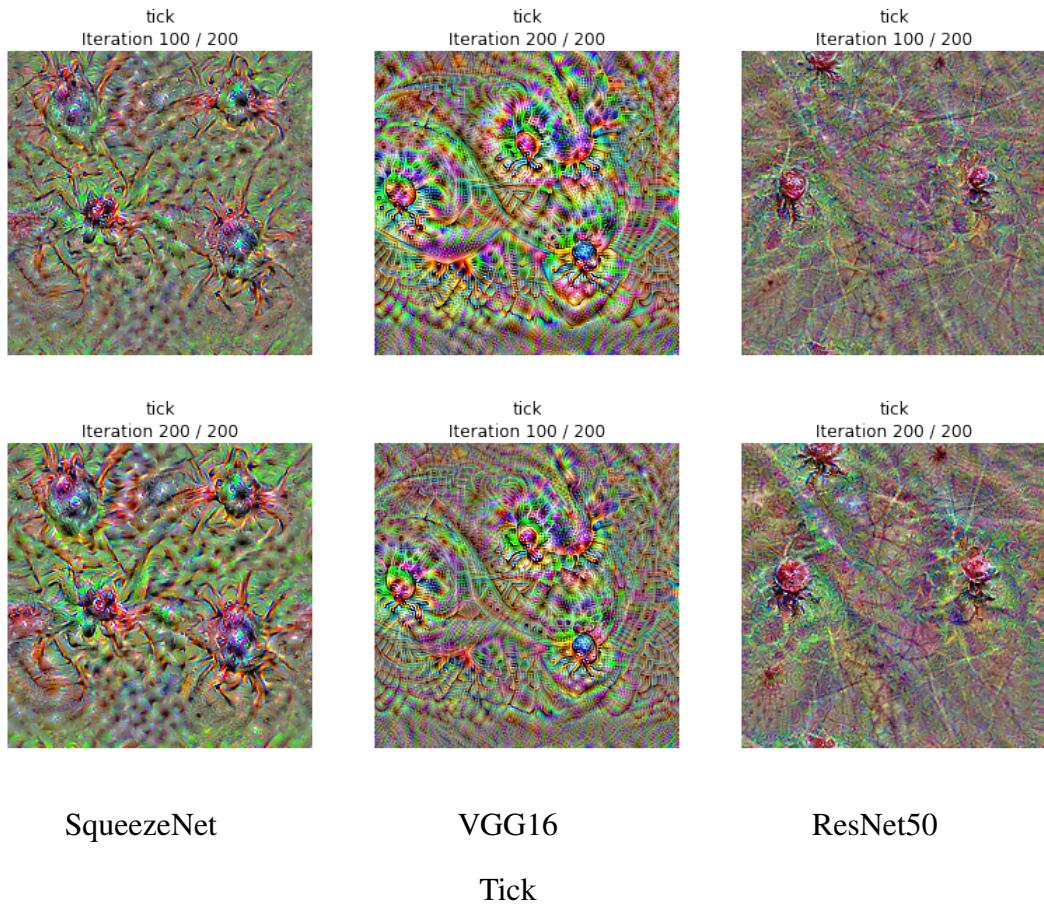
## Q3.10

Les *features* telles que les têtes d'ours, les yeux, les museaux, marquent leur présence au fur et à mesure des itérations. Tel est l'intérêt de cette méthode, comprendre quelles sont les features les plus influents dans la décision.

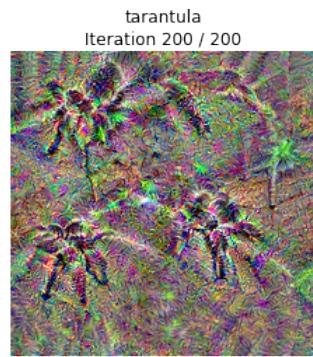
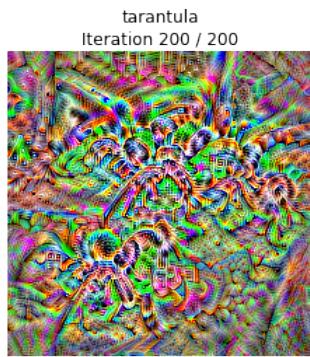
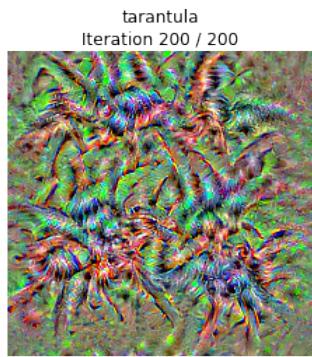
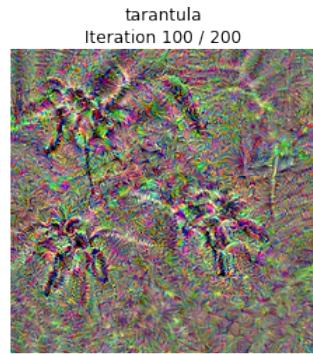
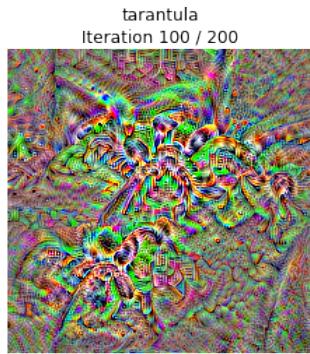
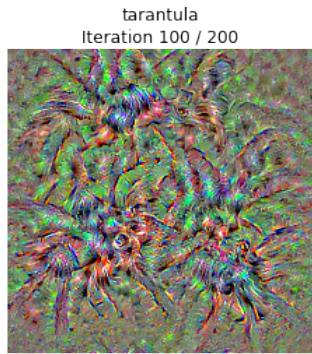


## Q3.11

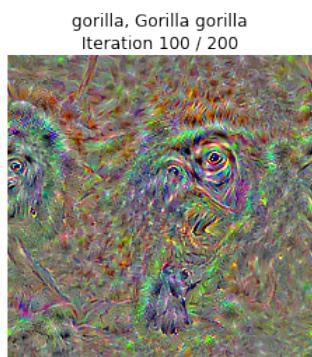
Nous avons testé différents modèles comme **VGG16** et **ResNet50** sur différents labels cibles notamment, *tick*, *tarantula* et enfin *gorilla*.



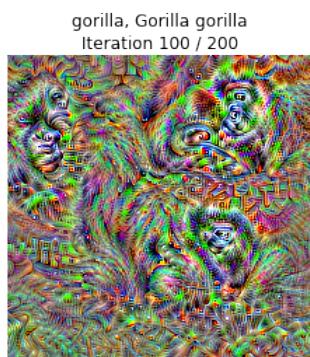
Les résultats obtenus par **VGG16** ne sont pas très différents de ceux de **SqueezeNet**. On peut toutefois souligner qu'ils sont beaucoup plus représentatifs des classes cibles. Quant à ResNet, il présente des résultats assez nets et non psychédéliques, nous arrivons à bien discerner les classes à l'oeil nu. Cela est peut être dû à son architecture très différentes des deux autres, en ce sens qu'il utilise les connections de raccourcis.



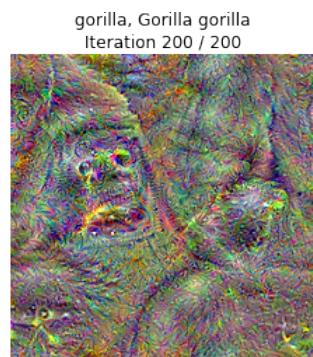
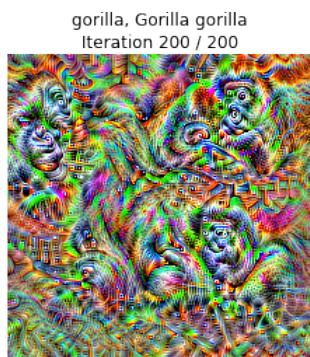
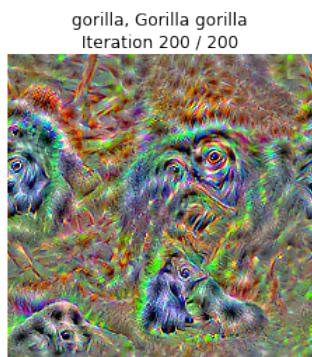
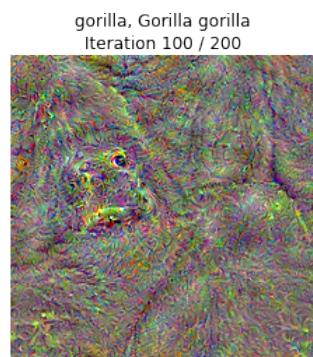
SqueezeNet



VGG16



ResNet50



SqueezeNet

VGG16

ResNet50

# GENERATIVE ADVERSARIAL NETWORKS

## PARTIE 1 - GENERATIVE ADVERSARIAL NETWORKS

---

### 1.1 PRINCIPE GÉNÉRAL

#### Q1-1.1

La première équation représente l'objectif de maximisation de l'erreur du discriminateur pour le générateur. En effet, nous voulons qu'il se trompe sur la classification des données générées. Et la seconde représente pour le discriminateur, dans un premier temps la maximisation de la bonne classification des données réelles, puis dans un second temps, la minimisation de l'erreur de classification sur les données générées. En utilisant seulement l'une des deux équations, un des classificateurs soit au niveau du discriminateur ou du générateur, apprendrait plus que l'autre, ce qui résulterait sur des images générées médiocres.

#### Q1-1.2

Le générateur cherche à approximer les données réelles. Dans un monde idéal, ce serait alors approximer la distribution  $P(z)$  en une distribution P(Data).

#### Q1-1.3

La vraie équation aurait été :

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

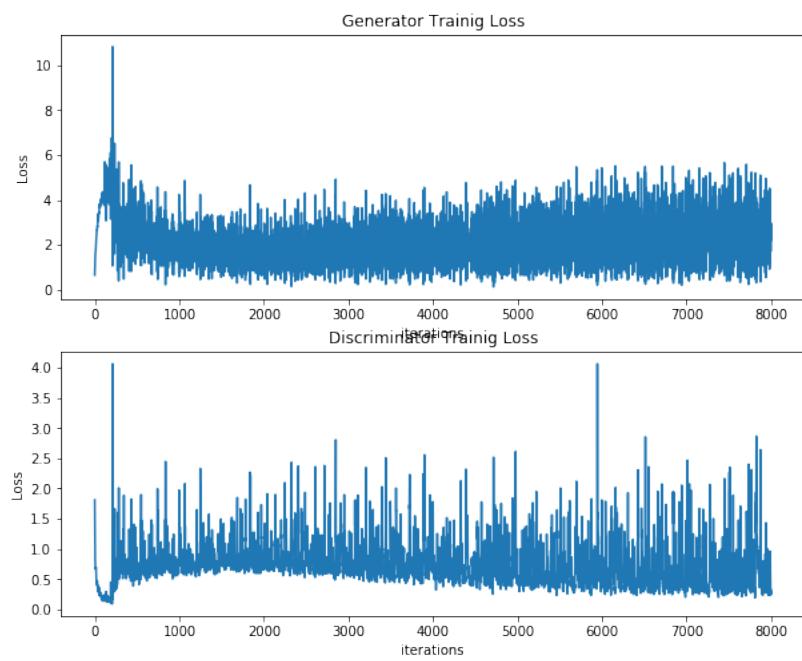
### 1.2 ARCHITECTURES DES RÉSEAUX

#### Q1-2.4

Les images générées par le modèle ont bel et bien la forme attendue, quand bien même la résolution est basse et les images floues. Il convient de signaler que nous avons obtenu des résultats probants vu de loin dès à partir des premières itérations. Les formes étaient là ! Il convient cependant de souligner que l'instabilité de la loss des deux classificateurs.



Génération d'images par le DCGAN



## Q1-2.5

En diminuant la taille de l'espace latent de manière conséquente ( $nz = 10$ ), nous ne remarquons pas une si grande différence avec les résultats obtenus précédemment, du moins, pas à l'oeil nu. Cela est sans doute du à la très faible résolution de nos images. Cela dit, nous avons très probablement des résultats bien plus simples, car théoriquement il serait difficile de modéliser nos données en un si petit vecteur. Inversement, si on prenait un vecteur de taille 1000 par exemple, cela prendrait sûrement trop de temps au modèle de mapper ces vecteurs aux données.



Génération d'images par le DCGAN avec un espace latent de taille 10

## PARTIE 2 - CONDITIONAL GENERATIVE ADVERSARIAL NETWORKS

Nous avons vu que l'un des principales problèmes que les GANs ont soulevé était le fait que de temps à autres notamment dans l'exemple de la base de données mangas, seul un genre était généré et cela pour toutes les images. Cela est probablement dû à un biais dans les données. C'est ainsi que rentre en jeu les GANs conditionnels qui permettent de générer les images conditionnées à une classe bien définie.

## 2.1 PRINCIPE GÉNÉRAL

### Q2-1.6

Dans le cadre des cGAN, nous cherchons à optimiser pratiquement la même équation à un détail près. Nous avons :

- Equation 6 :

$$\max_G \mathbb{E}_{z \sim P(z)} [\log(D(G(z|y), y)]$$

- Equation 7 :

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

### Q2-1.7

La variable  $y$  pourrait être conditionnée à des adjectifs ou classes comme, *homme, femme, jeune, vieux*, etc... De ce fait, on pourrait passer d'une image existante représentant la classe  $y = \text{homme}$  vers  $y = \text{femme}$ . Le principe étant de trouver un tirage  $z$  représentant la variable de départ et y appliquer la condition voulue, en l'occurrence *femme*.

### Q2-1.8

Aux variables, *été, hiver*.

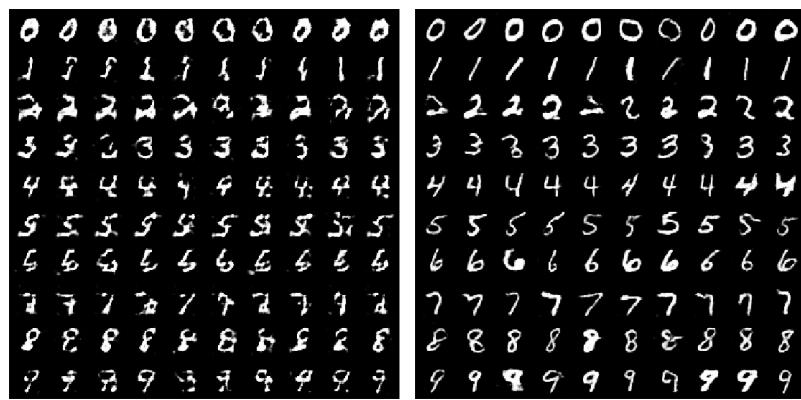
### Q2-1.9

Les variables dans cette vidéo sont conditionnées à la segmentation. Ce pourrait donc être, *arbre, voiture*, etc...

## 2.2 ARCHITECTURES cDCGAN POUR MNIST

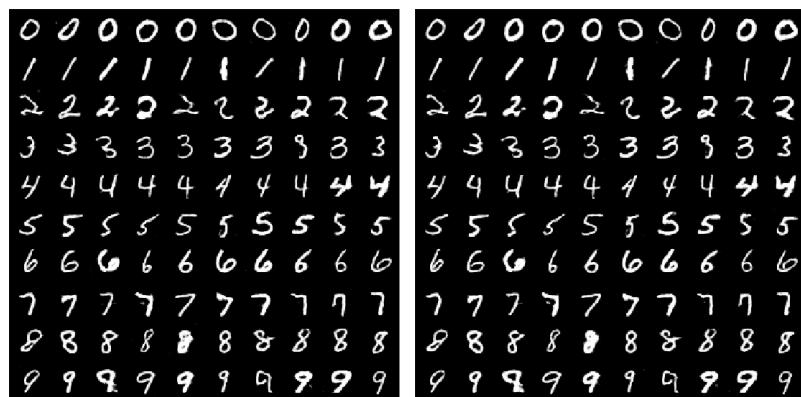
### Q2-2.10

Ici, nous avons appliqué le cDCGAN à la base de données MNIST en conditionnant chaque rangée à prédire le chiffre correspondant à son indice. Nous nous retrouvons avec des résultats très concluants. La loss des deux classifieurs convergent simultanément.



*iteration : 1000*

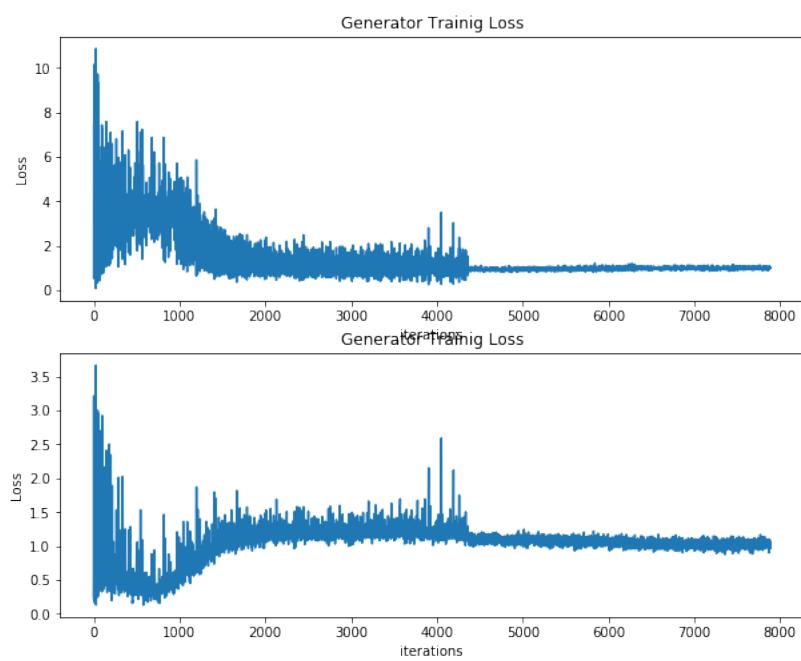
*iteration : 3000*



*iteration : 5000*

*iteration : 7800*

Génération d'images par le cDCGAN



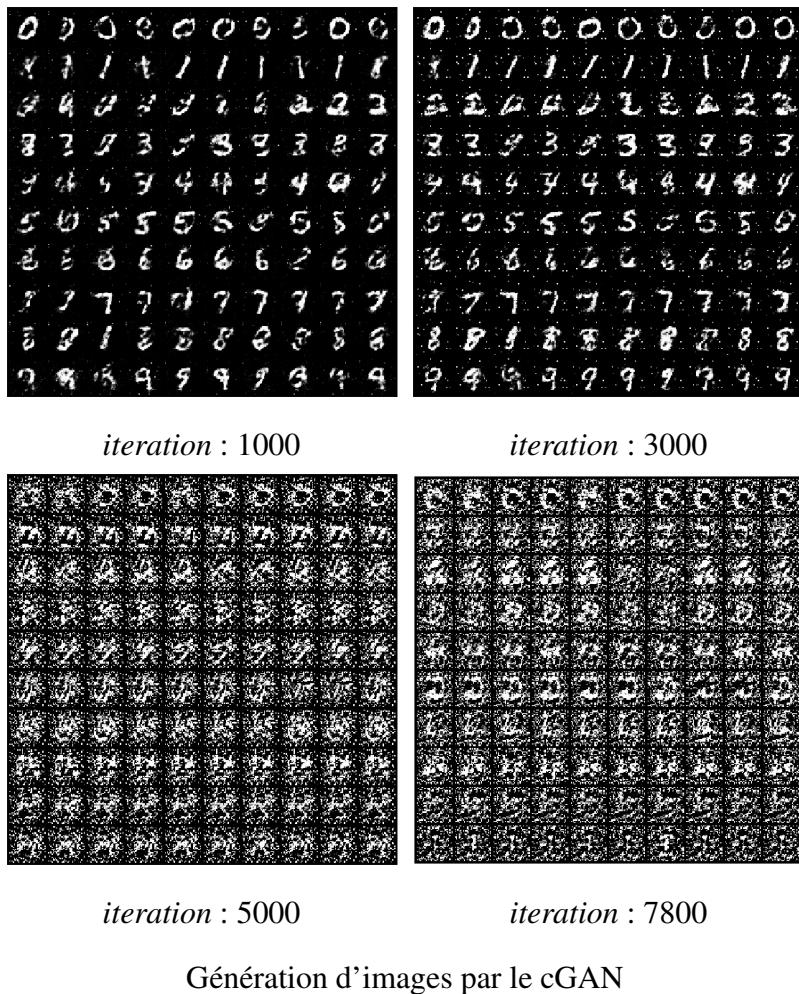
## Q2-2.11

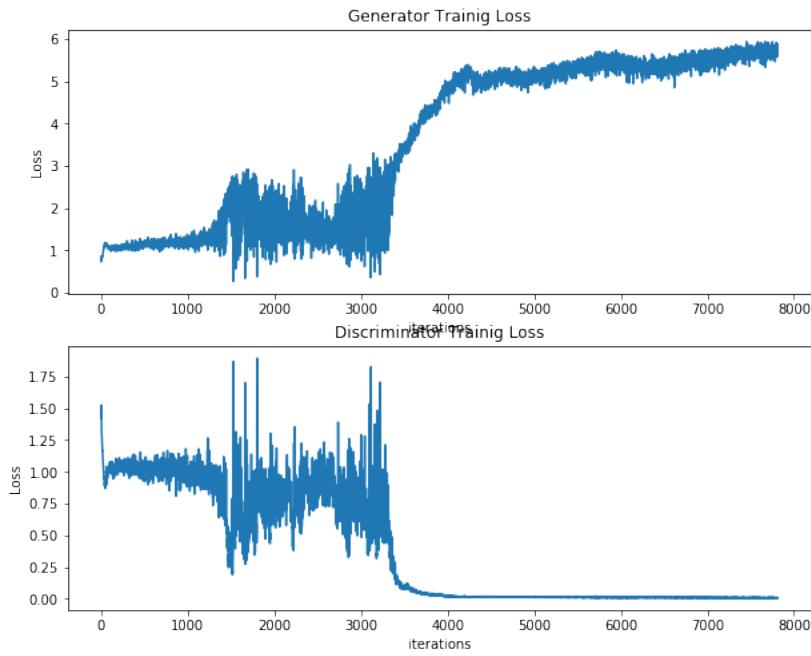
Si nous enlevons le vecteur  $y$  des entrées du discriminateur, cela conduirait alors tout simplement à générer des images autres que celles que nous attendons. Conditionnons par exemple notre modèle à prédire des images du chiffre 0, il s'agirait alors ici de prédire n'importe quoi et de ne pas être pénalisé par le discriminateur, ce dernier ne sachant pas quelle est la condition qui a été appliquée.

## 2.3 ARCHITECTURES CGAN POUR MNIST

### Q2-3.12

Le *cGAN* est relativement moins performant que son homologue, le *cDCGAN*. En effet, au bout d'un certain nombre d'itérations, le discriminateur devient très fort, si fort qu'il altère les performances du générateur. On remarque alors que ce dernier sous-apprend avec la loss du train qui augmente. C'est ce même phénomène que nous observons si nous entraînons un des deux, beaucoup plus que l'autre (*nb\_update\_G/D*).





## Q2-3.13

Il est en effet relativement plus difficile de générer des images avec un *cGAN* qu'avec un *cDCGAN*. Cela s'explique d'une part par le fait que le *cGAN* est exempt de convolutions, que des réseaux linéaires, ce qui entraîne donc la remise en question de tous les pixels à chaque fois que le discriminateur prend le dessus dans le bras de fer. C'est peut-être d'ailleurs ce qui explique aussi le bruit important sur les images générées.

## RÉFÉRENCES

---

Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *CoRR*, abs/1611.02770, 2016. URL <http://arxiv.org/abs/1611.02770>.