

kaggle

Rapport compétition Kaggle
« I'm Something of a Painter Myself »
Projet 8

Table des matières

<u>1.</u>	<u>INTRODUCTION.....</u>	<u>1</u>
<u>2.</u>	<u>PRESENTATION DU DATASET</u>	<u>1</u>
<u>3.</u>	<u>PRESENTATION DU NOTEBOOK REUTILISE</u>	<u>2</u>
<u>3.1.</u>	<u>CONCEPTION GENERALE DU CYCLEGAN</u>	<u>3</u>
<u>3.2.</u>	<u>TWO-OBJECTIVE DUALHEAD.....</u>	<u>5</u>
<u>3.3.</u>	<u>DATA-AUGMENTATION</u>	<u>8</u>
<u>4.</u>	<u>COMPARAISON CPU/GPU/TPU</u>	<u>8</u>
<u>5.</u>	<u>AMELIORATION ET UTILISATION DU NOTEBOOK.....</u>	<u>10</u>
<u>5.1.</u>	<u>SCENARIO 1.....</u>	<u>10</u>
<u>5.2.</u>	<u>SCENARIO 2.....</u>	<u>11</u>
<u>5.3.</u>	<u>SCENARIO 3.....</u>	<u>12</u>
<u>5.4.</u>	<u>SCENARIO 4.....</u>	<u>13</u>
<u>6.</u>	<u>CONCLUSIONS</u>	<u>14</u>
<u>7.</u>	<u>ANNEXES.....</u>	<u>15</u>

1. Introduction

Dans le cadre du projet 8, j'ai choisi la compétition « I'm Something of a Painter Myself ». Cette compétition porte sur le thème des réseaux antagonistes génératifs ou en anglais generative adversarial networks (GAN). Les GAN sont des algorithmes d'apprentissage non supervisés qui permettent de générer des images. Ils se composent d'un réseau générateur qui crée l'image (dans notre cas de figure) et d'un réseau adversaire qui doit distinguer s'il s'agit d'une image réelle ou générée. Dans le contexte de la compétition Kaggle choisie, il s'agit de générer des images ressemblant à des tableaux du peintre Monet, cette tâche est appelée « Style transfer » ou encore « image to image translation ».

2. Présentation du dataset

Le dataset mis à disposition se compose de 300 images de peintures de Monet et 7028 photos qui peuvent être utilisées comme base pour générer des images de tableaux avec le style de Monet.

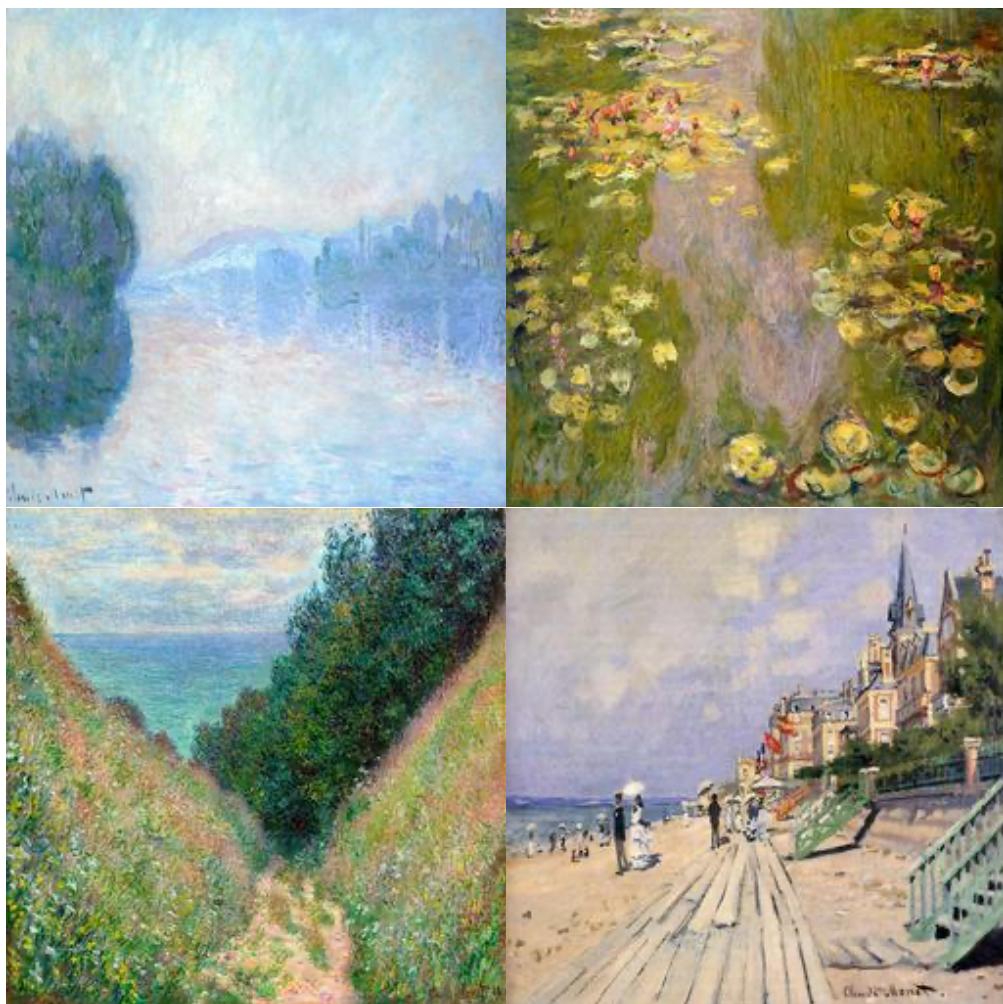


Figure 1-Exemples de peintures de Monet

Deux jeux de données peuvent être utilisés, un avec des images en format JPEG et un autre avec ces images converties au format TFRecords. Ce dernier format est plus efficace en termes

de volume de stockage, de temps d'accès et est optimisé pour les opérations de calcul en parallèles. En raison de ces avantages, je choisis de travailler les données au format TFRecords.

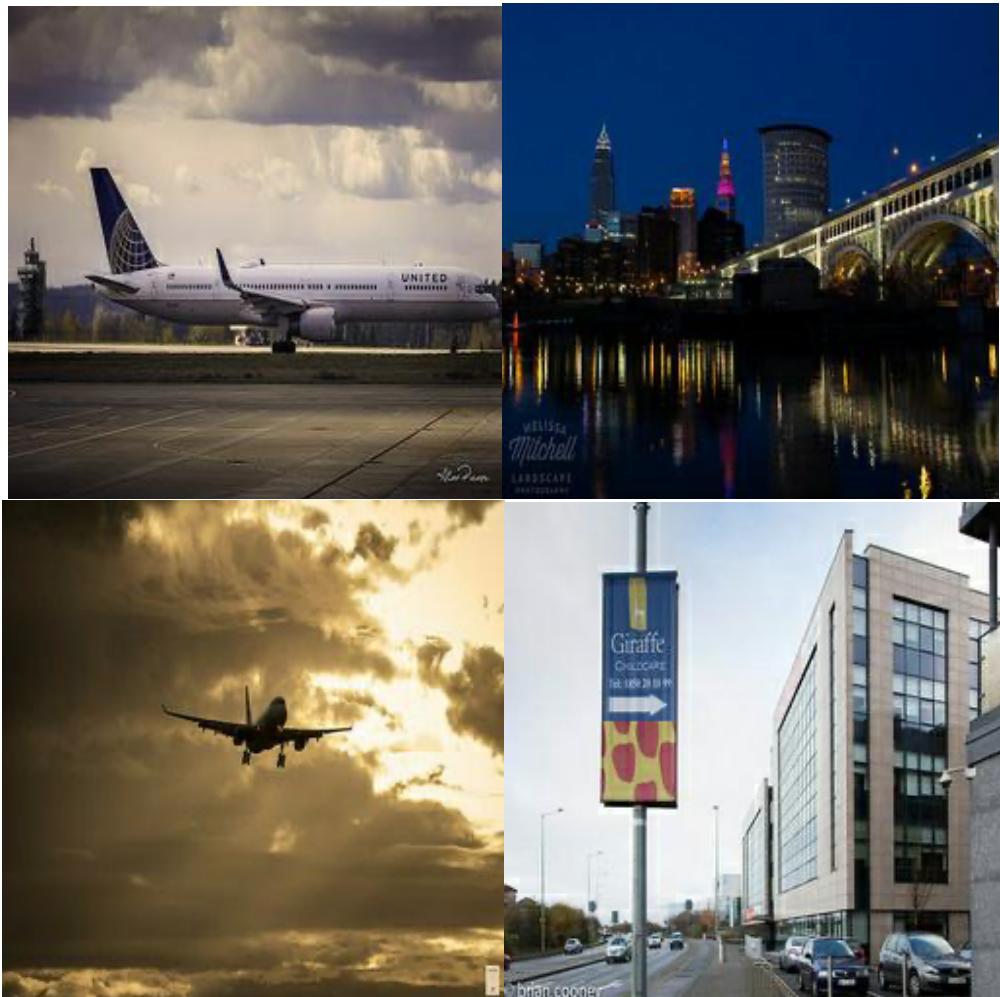


Figure 2-Exemples d'images pouvant servir de base à la génération

En parcourant les images pouvant servir de base à la génération, j'observe que certaines font apparaître des éléments anachroniques (moteur à réaction, voitures récentes, buildings de verres) par rapport à l'époque de Monet (1840-1926). Si l'objectif devait être de tromper des amateurs de peinture, il conviendrait de retirer ces photos qui seraient facilement disqualifiées, même par un néophyte.

3. Présentation du notebook réutilisé

EN recherchant sur page de la compétition, j'ai trouvé un notebook disponible qui intégrait un CycleGAN de base avec des fonctions de Data-Augmentation et de Discriminateur Dual à cette adresse :

<https://www.kaggle.com/code/unfriendlyai/two-objective-discriminator/notebook>

Avant de commencer à l'utiliser, je l'ai étudié. Son fonctionnement est présenté dans les chapitres qui suivent.

3.1. Conception générale du CycleGAN

Il existe de nombreuses tâches de computer vision (Image Generation, Image Classification, Object Detection, etc..) parmi lesquelles nous trouvons le Style Transfer. Cette tâche consiste à faire évoluer le style d'une image vers celui d'une image d'un autre domaine. Dans le cadre de ce projet, il s'agit de donner à des photos le style d'une peinture réalisée par Monet.

Certains modèles de GAN requièrent des datasets d'images appariées, c'est-à-dire qu'à une image dans le domaine de départ a été associée l'exacte image correspondante dans le domaine d'arrivée. Ce type de dataset est fastidieux à créer et souvent limité en termes de taille selon les domaines de départ ou d'arrivée. Par exemple il est possible d'une part de trouver des millions de photos de paysage, cependant Monet n'a peint « que » quelques centaines de tableaux. C'est pourquoi des modèles ne requérant pas de datasets appariés ont été développés, car il est facile de constituer deux datasets d'images, chacun spécifique au domaine de départ ou d'arrivée mais sans avoir à relier chaque image de départ à une image d'arrivée.

Un CycleGAN entraîne 2 générateurs d'images et 2 discriminateurs dont les architectures sont exposées ci-dessous.

Un générateur se compose d'un :

- Encoder, constitué de layers de convolution, qui produit des features map à partir de l'image d'entrée.
- Transformer, qui contient dans ses filtres, appris lors de l'entraînement, le style de Monet
- Decoder, qui génère à partir des features maps « stylisées » une image re-stylisée

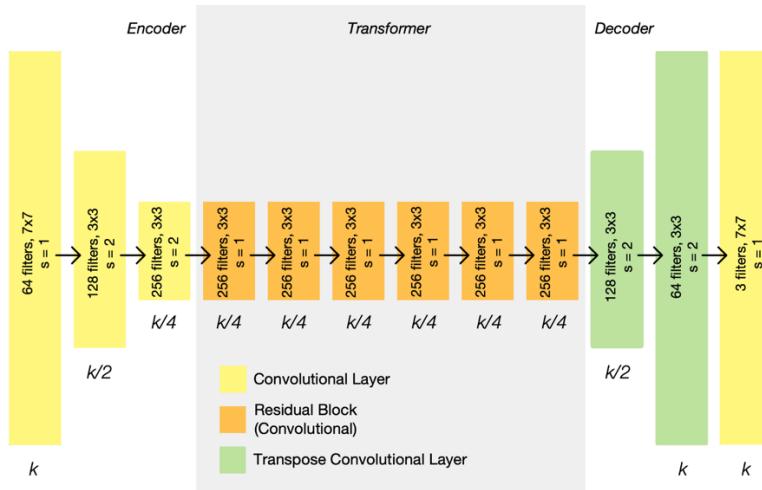


Figure 3 - Architecture générateur (Source : CycleGAN: Learning to Translate Images)

Un discriminateur est un réseau de layers de convolution qui se termine avec une fonction d'activation de type sigmoïde pour former un classifier binaire.

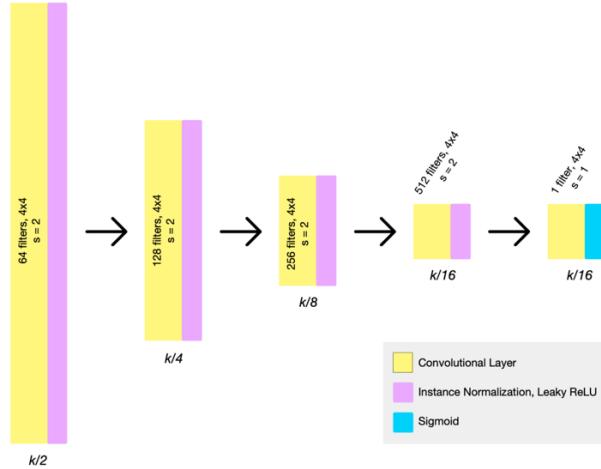


Figure 4 - Architecture discriminateur (Source : CycleGAN: Learning to Translate Images)

Le modèle CycleGAN est conçu pour utiliser des datasets d’images non appariées et fonctionne pendant son entraînement des 2 manières suivantes :

Démarche 1 (forward pass) :

- Une photo X passe dans le générateur de peintures de Monet G
- Il en ressort une peinture générée \hat{Y} ,
- La peinture générée \hat{Y} passe dans le générateur de photos F
- Il en ressort une photo générée \hat{X}

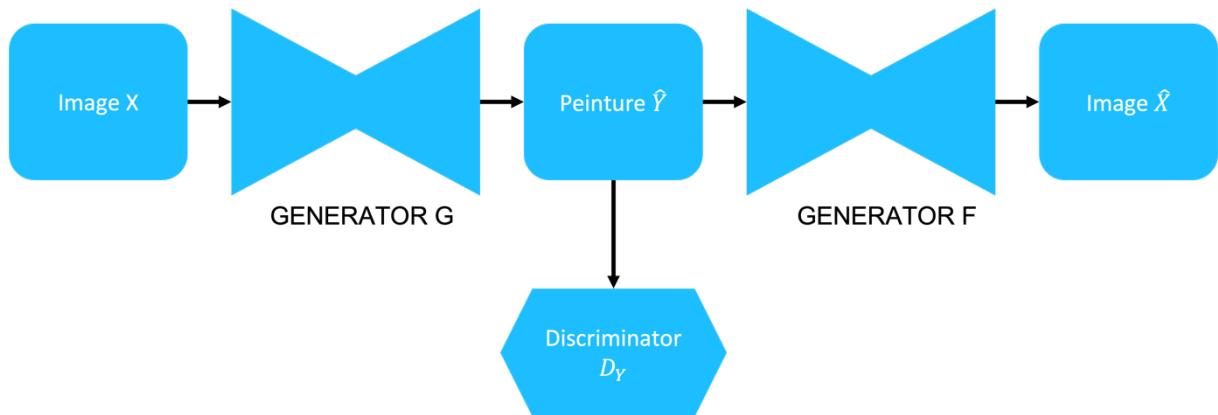


Figure 5 - CycleGAN forward pass

Démarche 2 (backward pass) :

- Une peinture Y passe dans le générateur de photo F
- Il en ressort une photo générée \hat{X} ,
- La photo générée \hat{X} passe dans de peintures de Monet G
- Il en ressort une peinture générée \hat{Y}

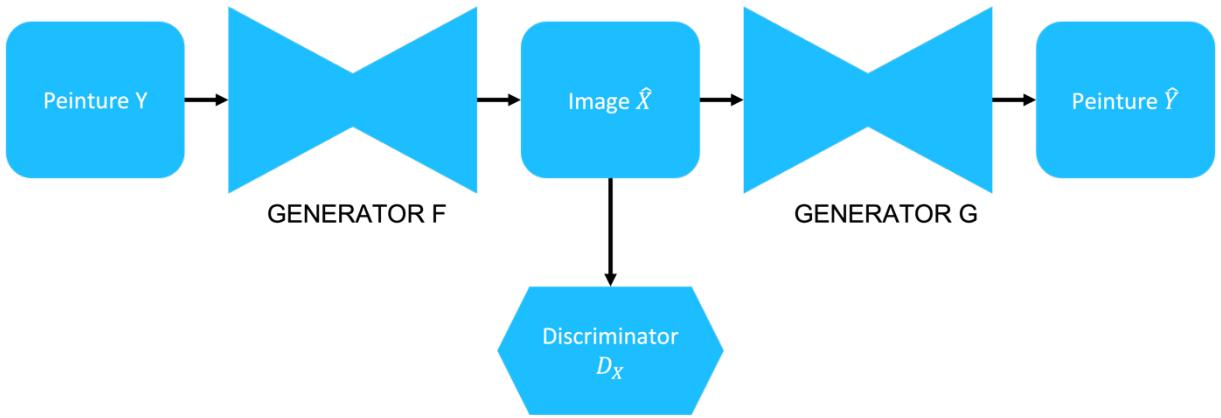


Figure 6 - CycleGAN backward pass

A chaque Epoch une fonction de perte est calculée et une descente de gradient stochastique avec rétro-propagation est utilisée pour calculer les gradients et mettre à jour les poids des différents réseaux de neurones.

Cette fonction de perte peut être dissociée en 3 parties :

- Adversarial Loss, elle mesure l'erreur du discriminateur D_x à identifier une fausse peinture d'une vrai et aussi l'erreur du discriminateur D_y à identifier une fausse photo d'une vraie.
- Cycle Consistency Loss, si nous n'utilisions que la première fonction de perte le modèle tendrait à produire toujours la même peinture générée ou la même photo générée. Mais il ne s'assurerait pas que l'image d'input et d'output soient similaires dans les formes. C'est pourquoi la Cycle Consistency Loss est introduite, elle mesure l'erreur entre l'image initiale et celle reconstituée ($X \sim \hat{X}$, $Y \sim \hat{Y}$) pour forcer les générateurs à conserver les formes des images.
- Identity loss, elle force le modèle à changer une partie d'une image que lorsque cela est nécessaire, ($X \sim G(X)$, $Y \sim F(Y)$). Ainsi une peinture de Monet passée dans le générateur de peinture doit ressortir quasiment inchangée, idem pour une photo dans le générateur de photo.

3.2. Two-objective dualhead

Le problème du dataset utilisé est le faible nombre d'images de peintures de Monet, ceci tend à mener à un overfitting du CycleGAN, c'est-à-dire que le modèle ne capture qu'une partie de la distribution possible des données réelles. Ce problème est aussi connu sous le nom de « mode collapse ». Pour éviter cela le notebook met en œuvre la démarche suivante pour la partie discrimination d'images de peintures générées :

- Crédit à un discriminateur D1 qui attribue un score élevé pour les peintures réelles et faible pour les peintures générées ;
- Crédit à un discriminateur D2 qui attribue un score élevé pour les peintures générées et faible pour les peintures réelles ;

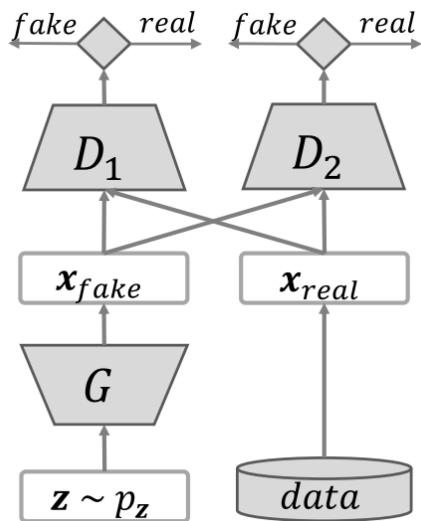


Figure 7 - Architecture d'un double discriminateur
(Source : Dual Discriminator Generative Adversarial Network)

Pour bien comprendre comment l'ajout du 2ème discriminateur lutte contre le « mode collapse », nous allons regarder séparément l'effet de chaque discriminateur puis faire la synthèse.

Effet du discriminateur 1 :

Celui-ci calcule la probabilité que l'image soumise soit une peinture réelle de Monet. Il va avoir tendance à forcer le générateur de peinture à apprendre une répartition des variables de telle sorte cette répartition soit comprise dans la répartition réelle. Le générateur maximise ainsi la probabilité que l'image générée soit identifiée comme une vraie peinture, mais avec le risque de ne pas être capable de générer des images dans l'ensemble du domaine possible.

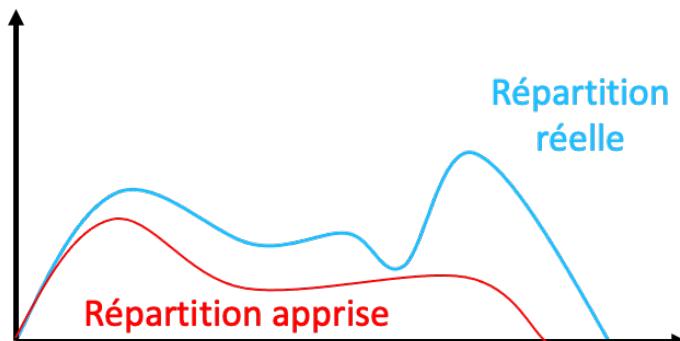


Figure 8 – Répartition tendant à être apprise avec le discriminateur 1

Effet du discriminateur 2 :

Celui-ci calcule la probabilité que l'image soumise soit une peinture générée. Il va avoir tendance à forcer le générateur de peinture à apprendre une répartition des variables de telle sorte cette répartition inclut complètement la répartition réelle. Le générateur maximise ainsi la probabilité qu'une peinture de Monet soit identifiée comme une image générée. Dit autrement le générateur est ainsi sûr de couvrir tout le domaine de génération de la répartition réelle, mais avec le risque de générer des images ne correspondant pas du tout à ce domaine.

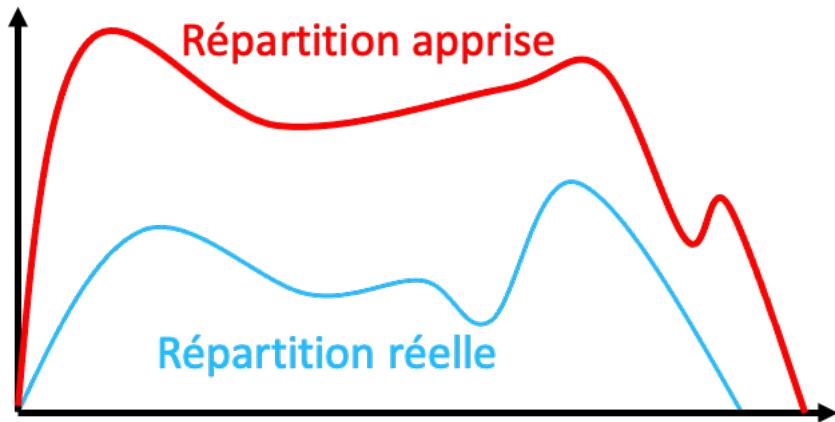
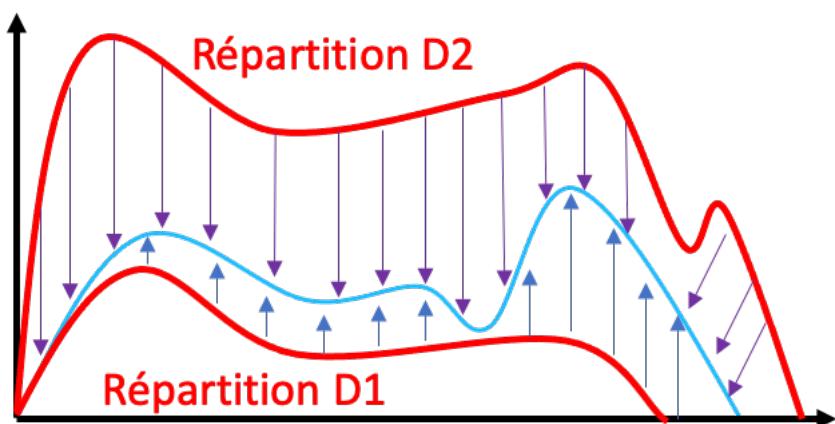


Figure 9 - Répartition tendant à être apprise avec le discriminateur 2

Effet de l'association des 2 discriminateurs :

En combinant les 2 discriminateurs, le générateur d'image est forcé par le discriminateur 1 à être inclus dans la répartition réelle des variables et est forcé par le discriminateur 2 à englober la répartition réelle des variables.



Le discriminateur 1 tend à ramener la répartition D2 vers la répartition réelle et le discriminateur 2 tend à ramener la répartition D1 vers la répartition réelle. Cette démarche est aussi connue sous le nom de « minimal enclosing ball ».

D'un point de vue technique, cette démarche est mise en œuvre de la manière suivante :

- Les discriminateurs 1 et 2 vont partager les mêmes layers de convolution (`m_disc`), seul le dernier layer (`Dhead`) sera appris séparément pour chacun des discriminateurs.

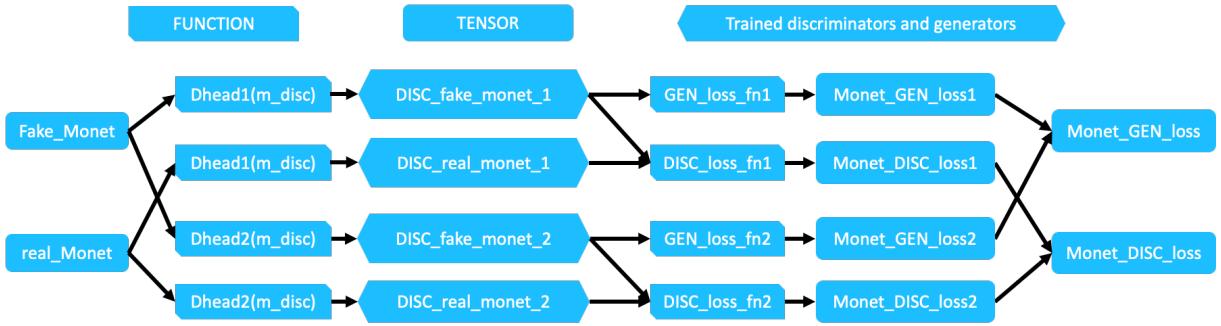


Figure 10 – Processus de calcul pour Monet_Generator_Loss et Monet_Discriminator_Loss

- Les fonctions GEN_loss_fn1 et DISC_loss_fn1 mettent en œuvre la fonction Hinge Loss : $\max(0,1-F(x))$
- Les fonction GEN_loss_fn2 et DISC_loss_fn2 mettent en œuvre la fonction Binary Cross Entropy (BCE) : $L_{cross-entropy}(\hat{y}, y) = -\sum_i y_i \times \log(\hat{y}_i)$

3.3. Data-augmentation

Une opération de data-augmentation est menée à chaque epoch_step sur les données, celle-ci comprend les opérations suivantes :

- Variation aléatoire de la luminosité des images
- Variation aléatoire de la saturation
- Variation aléatoire du contraste
- Décalage aléatoire de l'image selon la verticale ou l'horizontale
- Rognage aléatoire des images

4. Comparaison CPU/GPU/TPU

Dans le cadre de ce projet j'ai pu noter l'accélération appréciable qu'apportait l'utilisation d'un TPU par rapport à un GPU. C'est pourquoi il me paraît important d'expliquer brièvement de quelle façon il se distingue d'un GPU.

Un GPU (Graphics Processing Unit) est initialement un processeur graphique utilisé pour le rendu des images sur ordinateur. En raison du grand nombre de coeurs, des GPU ont été développés pour servir d'unité de calcul supervisée par un CPU, car ils permettent une grande parallélisation des calculs et donc l'accélération des traitements notamment en machine learning.

Les CPU restent plus performant sur l'exécution séquentielle de tâches car ils ont des coeurs optimisés pour le traitement en série. Ce ne sont donc que les tâches requérant beaucoup de calcul parallélisable qui sont confiées au GPU (ex : produit de matrices).

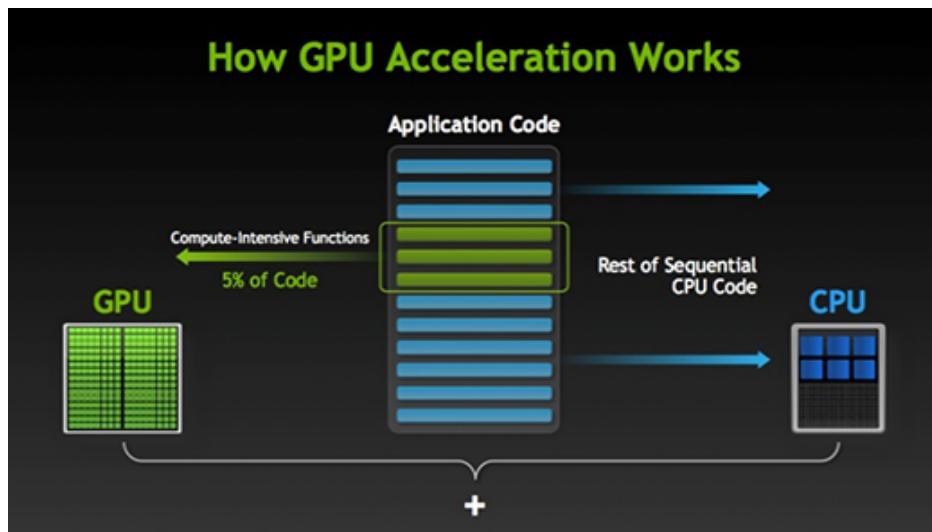


Figure 11 - How GPU Acceleration Works (Source: NVIDIA)

Les TPU (Tensor Processing Unit) sont des processeurs spécialement développés par Google pour faire des calculs matriciels qui sont très utilisés dans les modèles de réseaux de neurones. Ces processeurs dont la structure est dédiée à ce type de calcul réduisent considérable le temps de calcul ci-dessous le tableau indique le temps de calcul d'une epoch selon les configurations CPU, GPU et TPU.

Tableau 1 - Comparaison temps de calcul CPU/GPU/TPU

Configuration	Durée calcul Epoch
CPU-Batch size 16	4h05m11s
CPU-Batch size 32	4h14m07
GPU-Batch size 16	9m35
GPU-Batch size 32	8m21s
TPU-Batch size 16x8	35s
TPU-Batch size 32x8	30s

Sur Kaggle, des TPUv3-8 sont accessibles, avec ce modèle il est possible de faire des batchs de 8192 images (1024x8), chaque époque dure alors moins d'une seconde.

5. Amélioration et utilisation du notebook

Pour tester et rechercher la meilleure séquence d'entraînement, j'ai commencé par adapter dans le notebook la taille du batch aux capacités du TPU, ceci m'a permis d'accélérer considérablement le temps de calcul et de pouvoir tester plus rapidement différentes combinaisons. Cet aspect est important dans la mesure où le temps d'accès à un TPU que ce soit sur Kaggle ou Google Colab est limité, par exemple sur Kaggle, le temps d'utilisation est limité à 20h par semaine. Par ailleurs pour qu'un notebook soit compiler et noter par Kaggle. Il faut que le notebook soit capable de tourner en moins de 5h.

Ensuite j'ai créé une fonction qui permet de lancer en boucle différentes phases d'entraînement selon différent learning rate et nombres de batchs. Cette fonction prend comme argument 2 tableaux, le premier avec la séquence de learning rate et le second avec le nombre d'epoch de l'entraînement pour chaque learning rate.

5.1. Scénario 1

Le tableau ci-dessous présente les différents learning rate et le nombre d'epoch d'entraînement utilisé :

Tableau 2 - Paramètres scénario 1

learning_rate_array	number_epoch_array
2e-4	300
1e-4	200
1e-5	100

Score MiFID obtenu sur Kaggle : 45.80296

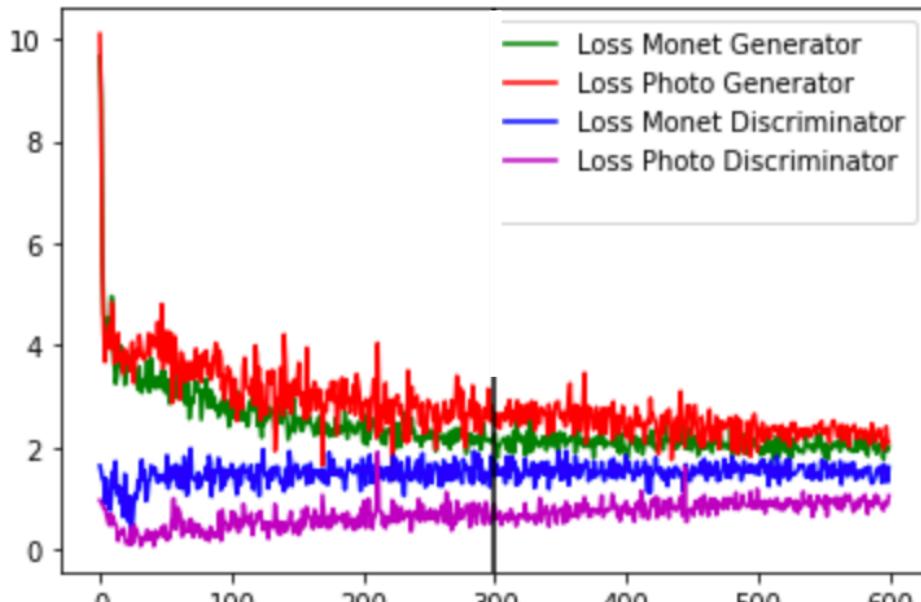


Figure 12 - Evolution de monet_gen_loss – scenario 2



Figure 13 - Photo réelle



Figure 14 - Peinture générée

5.2. Scénario 2

Le tableau ci-dessous présente les différents learning rate et le nombre d'epoch d'entraînement utilisé :

Tableau 3 – Paramètres scénario 2

learning_rate_array	number_epoch_array
2e-4	400
1e-4	100
1e-5	100

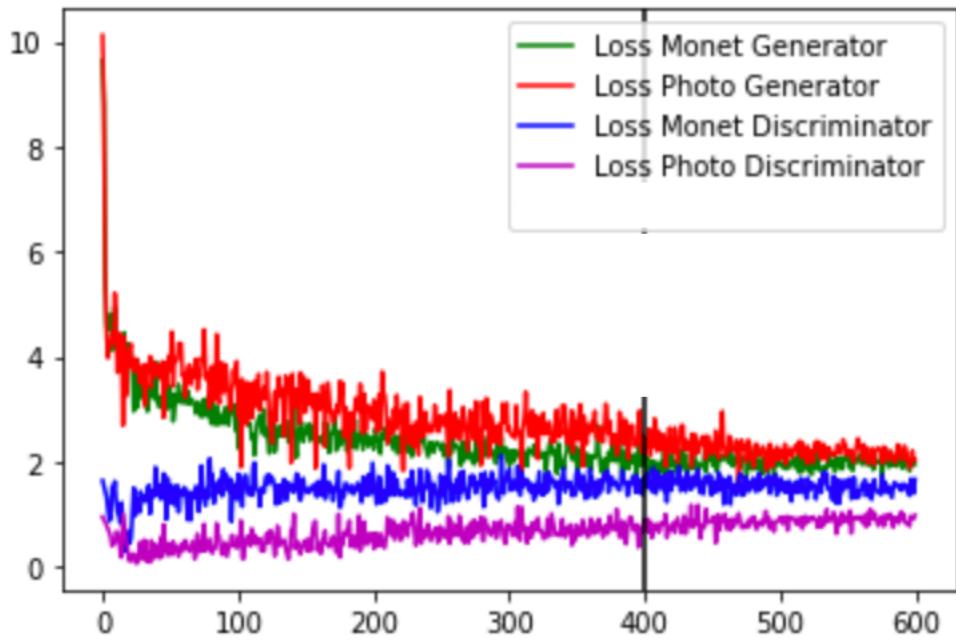


Figure 15 - Evolution de monet_gen_loss – scenario 2

Score MiFID obtenu sur Kaggle : 44.02150



Figure 16 - Photo réelle



Figure 17 - Peinture générée

5.3. Scénario 3

Le tableau ci-dessous présente les différents learning rate et le nombre d'epoch d'entraînement utilisé :

Tableau 4 - Paramètres scénario 3

learning_rate_array	number_epoch_array
2e-4	200
1e-4	200
5e-5	200
2.5e-5	200
1e-5	200

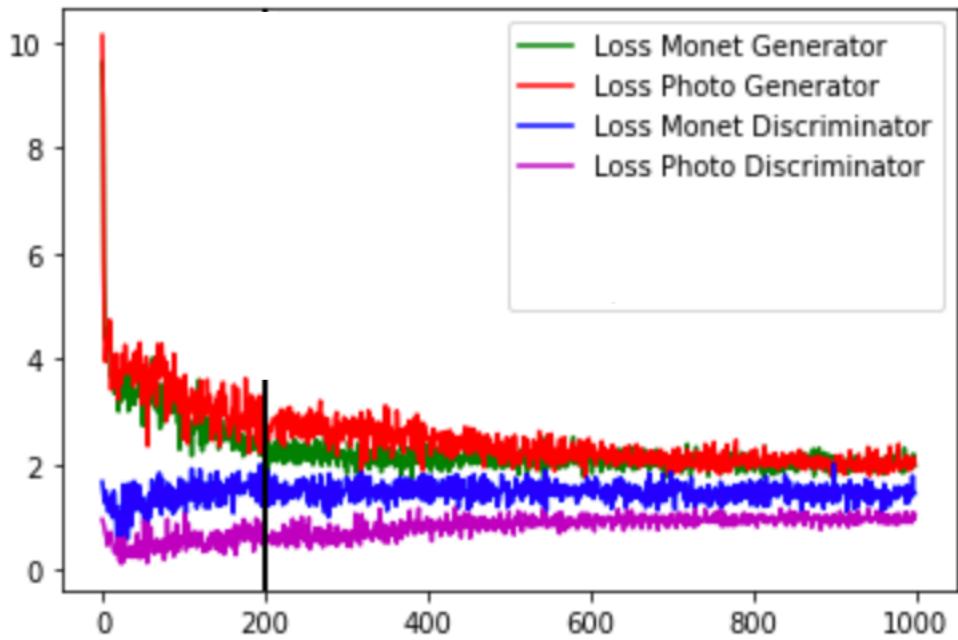


Figure 18 - Evolution de monet_gen_loss – scenario 3

Score MiFID obtenu sur Kaggle : 43.47180



Figure 19 - Photo réelle

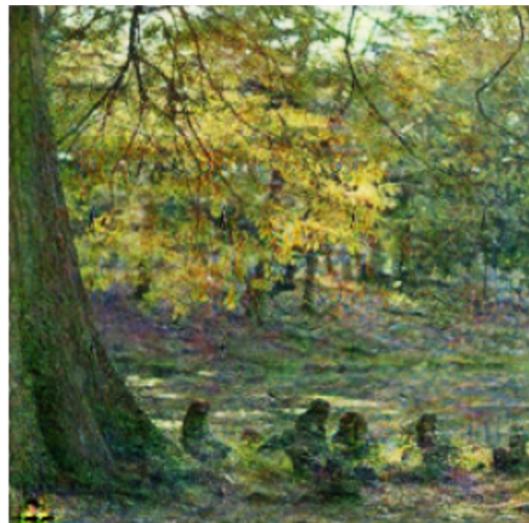


Figure 20 - Peinture générée

5.4. Scénario 4

Le tableau ci-dessous présente les différents learning rate et le nombre d'epoch d'entraînement utilisé :

Tableau 5 - Paramètres scénario 3

learning_rate_array	number_epoch_array
2e-4	400
1e-4	300
5e-5	200
2.5e-5	200
1e-5	200

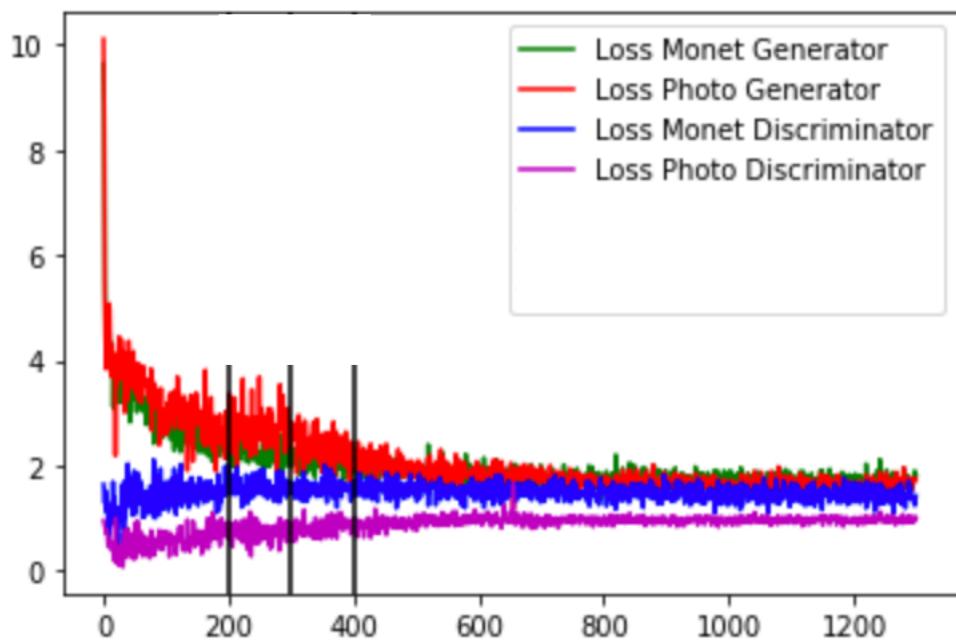


Figure 21 - Evolution de monet_gen_loss – scenario 4

Score MiFID obtenu sur Kaggle : 37.43675

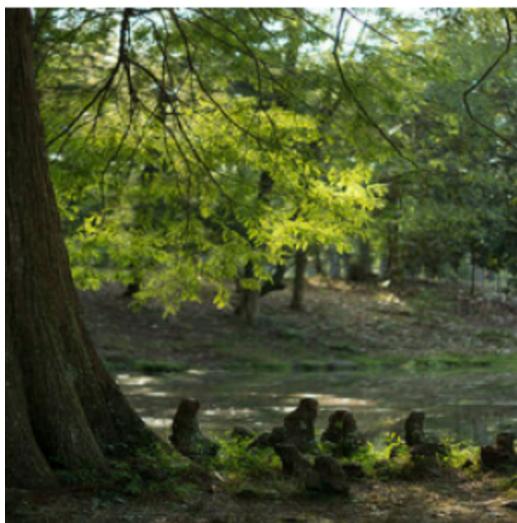


Figure 22 - Photo réelle



Figure 23 - Peinture générée

6. Conclusions

Pour avoir accès à de la ressource TPU sans délais d'attente, j'ai principalement travaillé depuis Google Collab. De même l'optimisation de la taille du batch au TPU disponible a considérablement diminué le temps de calcul et donc les différents essais.

Concernant les résultats, c'est en faisant varier le learning rate au cours de l'entraînement du modèle que j'ai pu améliorer mon score. D'après les différents scénarios, il apparaît que le modèle s'améliore énormément au cours des 20-30 premières epochs avec le learning rate le plus élevé. C'est ce learning rate qui permet d'améliorer le plus de score. Ensuite l'utilisation de learning rate de plus en plus fin permet d'améliorer lentement la performance.

Le kernel du scénario le plus performant peut être consulté à cette adresse :

<https://www.kaggle.com/code/jeff86/p8-01-cyclegan-v1?scriptVersionId=91487304>



Figure 24-Exemples de peintures générées avec le style de Monet

7. Annexes

Bibliographie/sources:

- <http://www-igm.univ-mly.fr/~dr/XPOSE2013/GPGPU/generalites.html#1>
- https://www.youtube.com/channel/UCytP08ynIym_J3Jr8zlmk3A/videos
- <https://www.kaggle.com/code/unfriendlyai/diffaugment-is-all-you-need/notebook>
- <https://livebook.manning.com/book/gans-in-action/chapter-9/40>
- <https://towardsdatascience.com/cyclegan-learning-to-translate-images-without-paired-training-data-5b4e93862c8d>
- <https://arxiv.org/pdf/1709.03831.pdf>