Université Claude Bernard Lyon 1

RAPPORT DE PROJET

Deep Learning – Master 2 informatique

2023 – 2024

BOURBON ELODIE

p1906918

EL KAOUT MOHAMED AMINE

p2121433

MERCIER LORIS

p1906860

SIMOUD ACHOUR

p2312223

Code source : https://github.com/LorisMercier/Projet_Deep_CycleGan

Référent projet : Hana SEBIA

Table des matières

Introduction

Apparu pour la première fois en 2015 dans l'article *A Neural Algorithm of Artistic Style* [1], le transfert de style est un concept permettant aux intelligences artificielles d'appliquer un style particulier à une image donnée. Utilisé d'abord à travers les architectures de type GAN (Generative Adversarial Network) basées sur un générateur et un discriminateur, c'est ensuite aux CycleGAN introduit en 2019 par Zhu et al [2] de faire évoluer le domaine en proposant une consistance entre chaque génération. Intéressé par la dimension artistique d'une telle technologie, nous décidons dans le cadre de l'UE deep learning de réaliser un projet de transfert de style entre l'univers de l'incontournable saga *Pirate des Caraïbes* et l'un des plus célèbres mangas au monde : *One Piece*. Notre ambition affichée est de transformer une courte vidéo de Pirate des Caraïbes en style One Piece.

Après une brève introduction théorique sur les CycleGan, nous reviendrons dans ce rapport sur notre avancée et nos nombreuses difficultés quant à la réalisation de notre transfert de style initial. Nous évoquerons ensuite les étapes nous ayant amené à changer de projet pour avoir un rendu portant finalement sur un transfert plus classique entre des chevaux et des zèbres.

1. Etude théoriques des CycleGAN

Les réseaux génératifs adverses (GAN) ont marqué une révolution majeure dans le domaine du deep learning, offrant une capacité inédite de générer rapidement des données d'apparence réaliste. Les CycleGAN apparus en 2019 [2] se démarquent en tant que variantes spécialisées des GAN, conçues spécifiquement pour accomplir le transfert de style entre deux domaines visuels distincts, sans nécessiter une correspondance directe entre les images de ces deux domaines.

1.1 Principe de base des GAN

Les GAN sont des modèles composés de deux réseaux de neurones dits antagonistes, un générateur qui est spécialisé dans la génération de données synthétiques et un discriminateur entraîné pour différencier les données réelles des données synthétiques. Cette dualité conduit le générateur à produire des données pratiquement indiscernables des données réelles afin de tromper le discriminateur.

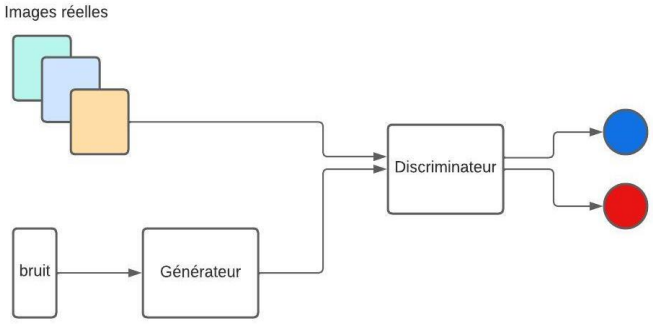

Figure 1: Architecture d'un GAN

1.2 Architecture des CyclesGAN :

L'architecture typique d'un CycleGAN comprend deux générateurs, G_{AB} qui transforme des données du domaine A vers le domaine B, et G_{BA} qui effectue l'inverse. Chacun de ces générateurs est associé à un discriminateur, respectivement D_A et D_B chargés d'évaluer la qualité des données générées dans leur domaine cible.

Au-delà des composants classiques que sont les générateurs et les discriminateurs, les CycleGAN intègrent une fonction de perte particulière appelée la "cycle-loss". Cette innovation garantit que les différences entre l'image d'origine et celle générée après un cycle complet soient minimisées. Ces modèles ouvrent ainsi la voie à des transferts de style plus cohérents garantissant pour une même image, une sortie quasiment similaire à chaque exécution.

C'est d'ailleurs pour cette raison que nous choisissons d'implémenter un CycleGan dans le cadre de notre projet. Transformant des frames successifs afin de reformer une vidéo, la cohérence des images générées est obligatoire afin de pouvoir suivre correctement l'évolution d'une scène.

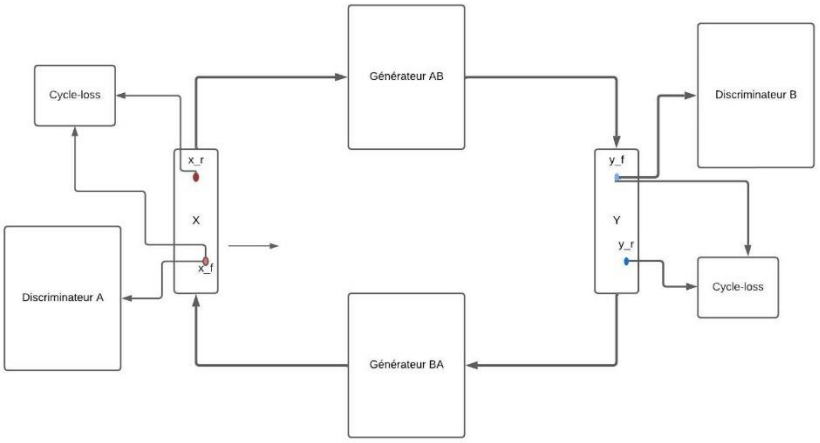

Figure 2: Architecture d'un CycleGan avec cycle-loss

2. Transfert de style Pirate des caraïbes – One Piece

Après avoir étudié la partie théorique des CycleGan, nous décidons de nous lancer dans un transfert de style entre *Pirates des Caraïbes* et *One Piece.* Notre objectif final est de reconstruire une mini-séquence vidéo de pirate des caraïbes en style One Piece. Mais avant cette ambition finale, débutons par la construction de notre jeu de données.

2.1 Construction du jeu de données

2.1.1 Obtention des images

Étape capitale de tout algorithme de machine learning, la construction d'un jeu de données doit attirer toute notre attention afin d'obtenir de bons résultats. Alors que de nombreuses bases spécifiques existent pour les modèles d'apprentissage, nous ne trouvons rien en rapport avec nos deux films.

Nous décidons alors de construire nous même notre database à partir d'extraits vidéo des deux films. Nous utilisons pour cela le logiciel *VLC* qui permet d'extraire des frames à partir d'une vidéo. Néanmoins nous devons veiller à obtenir des images différentes afin d'avoir un jeu de données variés. Prendre l'ensemble des frames d'un extrait n'est donc pas pertinent, la différence d'une image à l'autre étant minime. Pour pallier ce problème, nous choisissons plusieurs extraits des deux films pour lesquels nous exploitons qu'une frame par seconde. Ainsi, nous nous retrouvons avec une base d'image un peu plus riche qu'avec un frame par frame classique.

Toutefois, le manque d'extrait vidéo gratuit en ligne compromet légèrement cette construction de base. Aussi, de nombreux extraits comportent des sous-titres, ou logos altérant les images et réduisant encore plus le nombre de vidéos disponibles. A noter également que toute cette procédure soulève quelques questions au niveau des droits d'auteurs. Bien que le projet soit dans un cadre scolaire, le téléchargement d'extraits protégés pour une exploitation en projet n'est en théorie pas autorisé. Nous décidons donc de ne chercher d'autres extraits que ceux disponibles sur la plateforme *Youtube*.

Finalement, nous nous retrouvons avec un jeu de données de 150 images par style, soit 300 en tout. Ce nombre est relativement faible mais devrait malgré tout permettre d'obtenir des premiers résultats. Si ceux-ci sont concluants, nous enrichirons notre base pour gagner en performance.

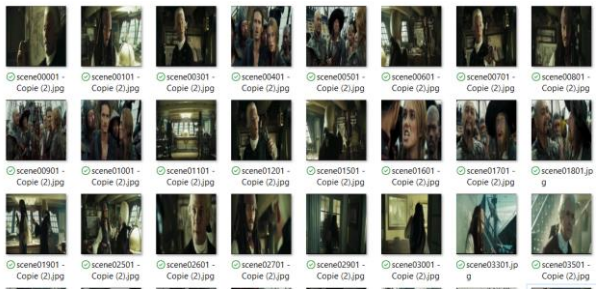

Figure 3: Jeu de données Pirates des Caraïbes

Figure 4: Jeu de données One Piece

2.1.2 Traitement des images

Une fois nos images répertoriées, nous devons les prétraités afin de les passer au modèle dans un format cohérent. Cette étape peut se réaliser directement dans la construction du modèle avec des fonctions de transformation des inputs lors du chargement des jeux de données. Nous préférons le faire en amont afin de réduire le poids de nos archives.

Pour cela, nous réalisons un script python réalisant les deux opérations suivantes :

- Suppression des barres noirs
- Réduction en taille 256*256

Ce script est à retrouver dans le notebook *traitement_image.ipynb.*

2.2 Mise en place du modèle

Pour réaliser la phase d'entraînement, nous décidons de créer notre propre modèle de CycleGan en s'inspirant des nombreux articles scientifiques et/ou notebook kaggle disponible. Nous suivons donc les indications de cette architecture en construisant :

- Un générateur AB (input : Pirates des caraïbes, output : One Piece)
- Un générateur BA (input : One Piece, output : Pirates des caraïbes)
- Un discriminateur B (Discrimine les vrais ou fausses images Pirates des caraïbes)
- Un discriminateur A (Discrimine les vrais ou fausses images One Piece)

Les générateurs partagent la même architecture et sont composés de 3 parties :

- Un premier bloc de convolution avec une couche convolutive.
- Un bloc de sous-échantillonnage avec deux couches convolutives (128 et 64).
- Un bloc résiduel avec 2 couches convolutives et des padding pour garder la taille inchangée.
- Un bloc sur-échantillonnage avec 2 couches de convolution et une interpolation bilinéaire (128,256)
- Une couche de sortie pour reconstituer les canaux de couleurs

Les discriminateurs sont construits selon l'architecture suivante :

- Un bloc de convolution de taille 64
- Un bloc de sous-échantillonnage avec deux couches convolutives (128 et 256)
- Une couche de sortie (0 ou 1)

Pour les Loss, nous reprenons celles évoquées dans l'article d'origine [2] :

- Une L1-Loss pour la *cycle-loss* entre la donnée réel A en entrée et sa reconstitution.
- Deux MSE pour l'*adversial-loss* mesurant à quel point les générateurs trompent leur discriminateur

Enfin, nous utilisons l'optimiseurs Adam pour nos 4 modèles.

2.3 Résultats

Confiant jusqu'à cette étape, les résultats obtenus sont en revanche très inquiétants. Notre modèle ne semble pas réellement apprendre et après une cinquantaine d'epoch, nous obtenons les résultats suivants :

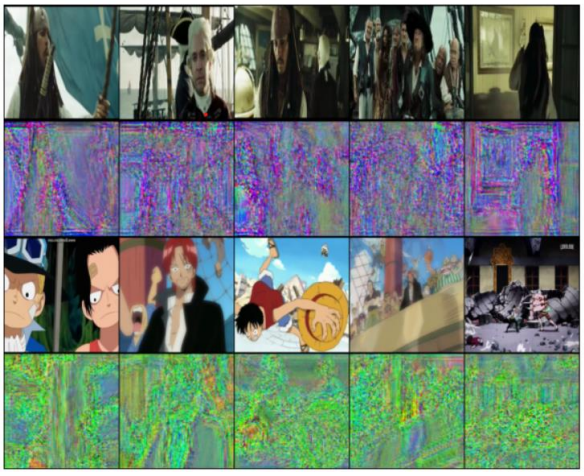

Figure 5: Test du modèle après 47 epochs

Seul du bruit semble être généré s'expliquant d'ailleurs très bien par l'ensemble de nos loss qui ne progressent pas. 47 epochs est certes très faibles, mais nous devrions au minimum constater une progression. Plusieurs hypothèses peuvent expliquer cet échec :

1. L'architecture du modèle

   Notre architecture n'est surement pas la plus optimale et peut comporter des erreurs majeures. Nous avons essayé de nous rapprocher petit à petit du modèle initial décrit dans l'article de *Zhu et al. [2]* en refaisant des phases de train mais les résultats n'ont pas été plus concluant

2. Notre base de train

   Petite et limitée en richesse due à la difficulté de trouver des extraits en libre accès, notre jeu de données est peut-être l'épine qui nous coince dans l'avancée de ce projet. Nous pensons que les styles ne sont pas assez parlants dans nos images pour que le modèle puisse en détecter ses originalités. Si nous regardons les exemples classiques de CycleGan tels que *Horse2Zebra* ou *Apple2Orange*, on remarque que l'ensemble des images sont spécifiques à un élément bien précis (les chevaux, ou les pommes). Le modèle peut ainsi assimiler les styles. Dans notre cas, toutes les images sont différentes sans réel point commun. En regardant les images de notre base de test, il est par exemple difficile de décrire un style précis de la dernière image de One Piece (voir ci-dessus). On pourrait presque assimiler cette image à un autre style que One piece rendant ainsi l'apprentissage très compliqué.

3. Les ressources disponibles

   Enfin, une dernière complication indépendante de ce transfert de style précis se présente à nous : le temps et les ressources de calculs.

   Aucun membre de notre groupe possédant une carte graphique performante, nous devons utiliser google collab. Or, en version gratuite, collab ne propose qu'une dizaine d'heures de calcul par jour. Cela semble bien suffisant pour notre projet si nous nous y mettons sur plusieurs jours mais collab gratuit ajoute également une détection de présence toutes les 20/30 minutes à travers un captcha. Il nous est alors très compliqué d'entraîner le modèle sur des longues durées sans interruption de cette fameuse détection de présence.

   En prenant en compte les nombreux changements d'architectures de notre modèle, nous avons effectué au global environ 300 runs pour une durée approximative de 27h.

3. Transfert de style Cheval - Zebre

N'arrivant pas à trouver de solutions optimales pour notre premier transfert de style à quelques jours du rendu, un dilemme se présente alors à notre groupe : poursuivre nos efforts quitte à récupérer une architecture toute faite, ou changer de projet pour un transfert plus classique avec un jeu de données beaucoup plus fiable que le nôtre.

Le temps pressant et les consignes insistant sur la reproductibilité du code, nous décidons de partir sur un transfert de style Cheval/Zèbre largement documenté sur le web et pour lequel nous pouvons nous appuyer sur des éléments existants.

Nous gardons en revanche en tête notre idée d'appliquer ce transfert de style sur une vidéo afin de valoriser le travail effectué sur la première partie du projet.

3.1 Recherche du jeu de données et du modèle

3.1.1 Le jeu de données

Afin de ne pas retomber dans les mêmes difficultés qu'auparavant, nous partons d'un jeu de données existant et standard pour ce transfert de style. Nous le trouvons sur kaggle à l'adresse suivante : https://www.kaggle.com/datasets/balraj98/horse2zebra-dataset

Ce jeu de données à l'avantage d'avoir été pré-traité et d'avoir été utilisé à de nombreuses reprises dans des problèmes de transfert de style. Nous l'analysons malgré tout avec un script python (*voir traitement_image.ipynb*) pour détecter d'éventuels problèmes de format et nous trouvons 4 images non RGB. Nous décidons de les supprimer du jeu d'entraînement.

3.1.2 Le modèle

Concernant le modèle, nous décidons de reprendre une architecture existante pour laquelle nous adapterons certaines parties. Nous essayerons aussi de jouer sur les hyper-paramètres. Le code du modèle initial se retrouve ici : https://www.kaggle.com/code/lmyybh/pytorch-cyclegan/notebook

Ce modèle a été entraîné originellement sur un transfert entre une photo et le style de peinture de Monet. Dans notre cas, nous repartons uniquement du code (et non du modèle) pour réeffectuer la tâche d'apprentissage avec notre jeu de données.

3.2 Adaptation et entraînement du modèle

La première adaptation du code a été de modifier le chargement des données en séparant le jeu de test et de train. Nous prenons également le temps d'inspecter notre jeu de données et de supprimer certaines images que nous jugeons non pertinentes. Nous enlevons par exemple les images non RGB.

Au niveau des hyperparamètres, nous jouons et effectuons quelques tests sur les paramètres bêtas de l'optimizer Adam. Ces deux paramètres sont des taux de décroissance influant dans le calcul du gradient à chaque moment. Nous les fixons à 0,9 et 0,999 comme recommandé dans de nombreux exemples que nous trouvons. Nous ajustons aussi le learning rate à 0.0001. Néanmoins, notre architecture utilise des *lr_scheduler()* ajustement dynamiquement ce learning rate pendant le processus d'entraînement.

Autre ajout indispensable pour notre projet, nous créons des fonctions de sauvegarde et lecture pour effectuer des checkpoints à chaque epoch. Collab risquant à tout moment de nous bloquer faute de ressource ou d'inactivité, il est essentiel d'avoir des sauvegardes fréquentes afin de ne jamais perdre l'entraînement effectué.

3.3 Résultats

Malgré un modèle et un jeu de données fiable, nos résultats nous laissent quelque peu dubitatifs. Après 100 runs, nous obtenons les résultats suivant sur notre base de tests :

```
[Epoch 97/100]
[G loss: 2.175433874130249 | identity: 0.1376338005065918 GAN: 0.5842071771621704 cycle: 0.09030577540397644]
[D loss: 0.09991714358329773 | D_A: 0.11929457634687424 D_B: 0.08053971081972122]
Nb epoch :  97
[Epoch 98/100]
[G loss: 2.1727867126464844 | identity: 0.13737079501152039 GAN: 0.5878854990005493 cycle: 0.08980470895767212]
[D loss: 0.09768021106719971 | D_A: 0.12661972641944885 D_B: 0.06874069571495056]
Nb epoch :  98
[Epoch 99/100]
[G loss: 2027.5877685546875 | identity: 0.6628081798553467 GAN: 2017.12255859375 cycle: 0.7151128053665161]
[D loss: 1518.744873046875 | D_A: 1915.7060546875 D_B: 1121.7835693359375]
Nb epoch :  99
```

Figure 6: Loss d'entrainement de notre modèle horse2zebra

Figure 7: Test du modèle à 90 epoch

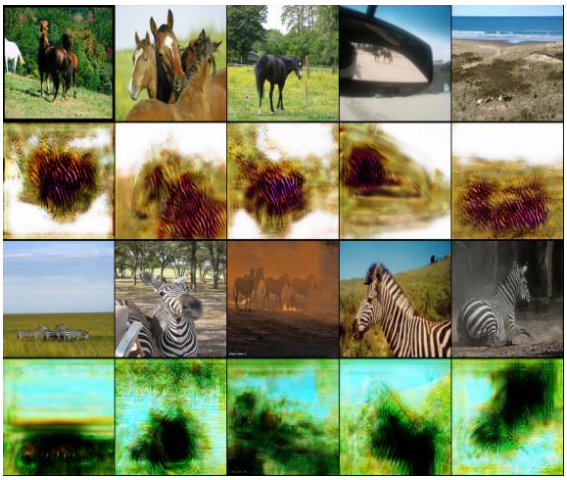

Figure 8: Test du modèle à 100 epoch

Figure 9: Transfert de style sur la vidéo

On constate que le modèle a totalement dégénéré sur la dernière epoch. Sauf si nous avions souhaité faire de l'art moderne, ce résultat est extrêmement décevant après 15h d'entraînement consécutif. Le résultat après 89 epoch est en revanche plus encourageant. Toutefois, le modèle ne semblait plus progresser depuis une trentaine de Loss comme s'il était bloqué sur un minimum local.

La vidéo est à retrouver dans le répertoire video/ de notre archive.

Trois hypothèses nous viennent en tête pour expliquer ces résultats déroutants :

1. Le temps

   N'ayant aucun repère dans l'architecture deep, il est difficile pour nous de savoir si 100 epoch est suffisant pour obtenir de bons résultats. Bien que notre modèle stagne d'epoch en epoch (sans parler de la dernière epoch), il est peut-être logique d'avoir une phase plus calme où le générateur et discriminateur s'affronte avant de donner de vraies améliorations plus tard.

2. La sauvegarde

   Notre système de sauvegarde et de chargement est peut-être défaillant. Tout l'apprentissage n'est peut-être pas correctement enregistré entraînant des pertes d'informations acquises par le modèle.

3. L'architecture du modèle

   Nous avons volontairement fait le choix de ne pas choisir un modèle pré-conçue et pré-entraîné sur le transfert de style cheval > zebre afin que notre travail soit valorisable sur le fine tuning et l'entraînement du modèle. Cependant, certains choix de design du code dépendent peut-être de son application d'origine (les peintures de Monet) rendant son exploitation plus délicate pour notre transfert de style actuel.

Conclusion

En conclusion, ce projet de deep learning nous a permis d'approfondir les techniques de GAN découvert en cours en appliquant ces principes sur un concept légèrement plus complexe : les transferts de style via CycleGan. Notre première idée ambitieuse de créer une courte vidéo de Pirate des Caraïbes en One Piece l'était peut-être un peu trop et nous ne sommes pas parvenus à obtenir ne serait-ce qu'un début de résultat. Dos au mur et non aidé par nos machines ne permettant pas de faire du deep learning dans de bonnes conditions, nous avons opté en dernier recours dans un projet plus simple en entraînant et fine-tunant un modèle pré-existant. Néanmoins, force est de constater que ce pari ne fut pas gagnant et nos résultats, bien qu'encourageant, ne sont pas réellement exploitables.

Sur le plan technique, ce projet n'est donc pas satisfaisant pour nous. Cependant, c'est à travers ce genre de difficultés que nous apprenons. A travers notre travail de recherche initiale pour comprendre plus en profondeur l'univers des GAN/CycleGan et notre première implémentation de modèle à partir de l'article scientifique original, nous avons acquis des connaissances nouvelles sur cette branche du deep learning.

Références

[1] Gatys, L. A., Ecker, A., & Bethge, M. (2015). A neural algorithm of artistic style. *arXiv (Cornell University)*. https://arxiv.org/pdf/1508.06576v1

[2] Zhu, J., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1703.10593