

Rapport de projet :

Application des Techniques de Deep Learning
pour la Conversion d'Illustrations en Pixel Art
(Transfert de Style, CycleGAN et Diffusion Stable)

AUMASSON Yanis
HANAMPATRA Gernido
BOHELAY Corentin

I. Introduction	3
1. Contexte et motivation	3
2. État de l'art	3
II. Description de l'Architecture des Modèles	4
1. Transfert de Style avec VGG-19	4
a. Principes de base	4
b. Architecture de Réseau utilisée	4
c. Fonction de perte	5
2. CycleGan	5
a. Principes de base	5
b. Architecture de Réseau utilisée	6
c. Fonction de perte	6
3. Stable Diffusion with Fine-Tuning	7
a. Présentation de Stable Diffusion	7
b. Principes de bases	7
c. Architecture	8
III. Détails de l'Entraînement	8
1. Plateforme et Infrastructure Utilisée	9
2. Stratégies d'Optimisation	9
3. Prétraitement des Données	9
IV. Analyse des Résultats Obtenus	10
1. Résultats Qualitatifs	10
a. Style Transfert avec VGG-19	10
b. CycleGAN	11
c. SD Fine-Tuning	13
2. Résultats Quantitatifs	14
a. Style Transfert	14
b. CycleGAN	14
c. SD Fine-tuning	16
3. Discussion critique sur les résultats	16
a. Style Transfert	16
b. CycleGAN	16
c. SD Fine-Tuning	16
V. Conclusion	17
1. Problèmes rencontrés et solutions	17
2. Répartition du travail	17
3. Synthèse	17
VI. Bibliographie	18

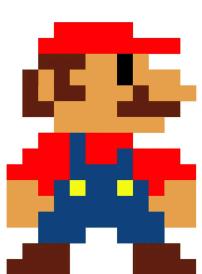
I. Introduction

Dans le cadre de l'unité d'enseignement Deep Learning, nous avons entrepris un projet visant à explorer les techniques de transfert de style appliquées à la conversion d'images dessinées en pixel art, et inversement. Le transfert de style est une branche du deep learning qui permet de fusionner le contenu d'une image avec le style d'une autre, ouvrant ainsi la voie à des créations artistiques innovantes.

1. Contexte et motivation

Le pixel art, avec son esthétique rétro et ses contraintes artistiques, occupe une place particulière dans le domaine des jeux vidéo et des médias numériques. Son caractère minimaliste et ses limites techniques en font un style apprécié pour sa capacité à transmettre des émotions et des histoires de manière concise.

Voici 2 exemples de Pixel Arts issus du monde du jeu vidéo :



Super Mario Bros (Nintendo NES)



Épée en fer (Minecraft)

La création manuelle de pixel art peut être laborieuse et chronophage, surtout lorsqu'il s'agit de convertir des illustrations détaillées en ce format spécifique. C'est dans ce contexte que le transfert de style émerge comme une solution prometteuse pour automatiser et faciliter ce processus de conversion.

2. État de l'art

Le transfert de style et la génération d'images par deep learning ont considérablement évolué grâce à des techniques telles que les Generative Adversarial Networks (GAN), CycleGAN, le transfert de style classique utilisant des réseaux comme VGG-16 ou VGG-19, et les modèles de diffusion stable. Les GAN permettent de créer des images réalistes en opposant un générateur et un discriminateur, tandis que CycleGAN

facilite la traduction d'images entre différents domaines sans nécessiter de paires correspondantes, ce qui est particulièrement utile pour la conversion entre dessins et pixel art. Le transfert de style classique, souvent basé sur VGG-19, combine le contenu d'une image avec le style d'une autre pour produire des œuvres artistiques uniques. Plus récemment, les modèles de diffusion stable ont émergé, générant des images de haute qualité en affinant progressivement des détails à partir de bruit aléatoire. Ces avancées offrent une flexibilité et une qualité accrues, constituant la base de notre projet visant à optimiser la conversion d'images dessinées en pixel art en exploitant et en comparant ces différentes approches.

II. Description de l'Architecture des Modèles

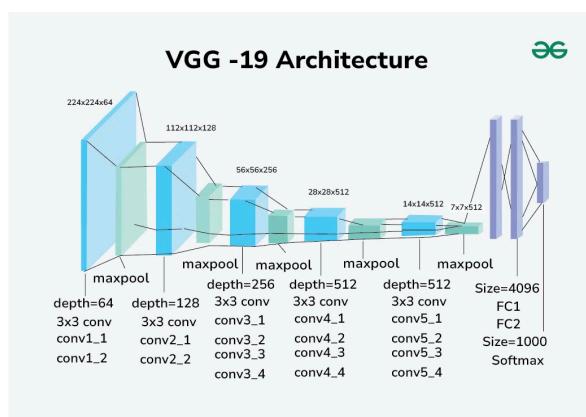
1. Transfert de Style avec VGG-19

a. Principes de base

Le transfert de style (Style Transfert) est une technique de deep learning qui permet de combiner le contenu d'une image avec le style visuel d'une autre, créant ainsi une nouvelle image qui conserve les éléments structurels de la première tout en adoptant les caractéristiques esthétiques de la seconde. Cette méthode a gagné en popularité grâce à son application dans la création artistique numérique, permettant de transformer des photographies ou des illustrations en œuvres d'art aux styles variés, tels que ceux de peintres célèbres comme Van Gogh ou Picasso. L'objectif est de transférer le style d'une image source (style) vers une image cible (contenu) sans altérer la structure sous-jacente de cette dernière.

b. Architecture de Réseau utilisé

Pour réaliser cette séparation, le modèle de réseau de neurones convolutifs VGG-19 est couramment utilisé en raison de sa capacité à extraire efficacement des représentations hiérarchiques des images. VGG19, développé par Simonyan et Zisserman inclut des couches convolutives et des couches de pooling, pré-entraîné sur le dataset ImageNet (une base de données de plus de 14 millions d'images).



VGG-19 comporte 16 couches de convolution regroupées en 5 blocs. Après chaque bloc, il y a une couche Maxpool qui réduit la taille de l'image d'entrée par 2 et augmente également le nombre de filtres de la couche de convolution par 2. Les dimensions des trois dernières couches denses sont respectivement de 4096, 4096 et 1000. VGG classe les images d'entrée en 1000 catégories différentes.

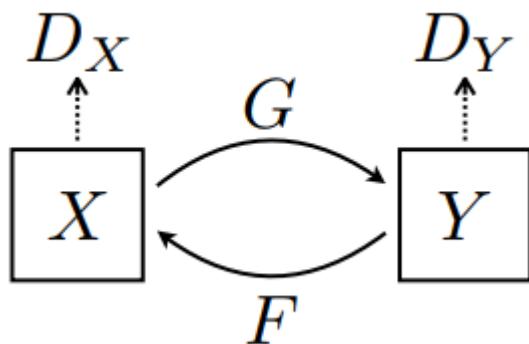
c. Fonction de perte

La fonction de perte utilisée dans ce projet se compose de deux parties principales, la perte de style et la perte de contenu. Ces deux pertes sont combinées pour obtenir la perte totale, qui est ensuite minimisée lors de l'entraînement du modèle. La perte de style mesure la différence entre le style de l'image générée et celui de l'image de référence. Elle est calculée en comparant les caractéristiques stylistiques extraites par le réseau VGG-19. La perte de contenu évalue la similarité entre le contenu de l'image générée et celui de l'image source. Elle assure que la structure et les éléments principaux de l'image source sont préservés.

2. CycleGan

a. Principes de base

Un Cycle-Consistent Adversarial Networks est une approche permettant d'apprendre un mapping d'un domaine X (source) à un autre domaine Y (target) sachant que les données ne sont pas appariées. Il est donc souvent utilisé pour faire du transfert de style. Le principe repose sur 2 générateurs (G, F) et 2 discriminateurs (Dx, Dy) :



G apprend à transformer une image X en une image Y (F l'inverse).

Dx apprend à différencier le vrai X du X généré (respectivement Y pour Dy).

Pour s'assurer que la transformation d'un domaine à l'autre conserve les caractéristiques des données d'origine, une contrainte de cycle-consistance est imposée. Concrètement, si on reconstruit une image transformée, on doit retomber sur l'image du domaine d'origine :

- $F(G(x)) \approx x$ et $G(F(y)) \approx y$

b. Architecture de Réseau utilisée

Nous avons utilisé une architecture U-Net pour les générateurs, et PatchGAN pour les discriminateurs.

Structure des générateurs :

- Encodage (Downsampling)
 - 8 couches d'encodage de (128, 128, 64) à (1, 1, 512) composées de :
 - Une convolution 2D (4*4) avec stride = 2, initialisée par une loi normale ($\mu = 0$, $\sigma = 0.02$)
 - Une normalisation InstanceNorm
 - Une activation LeakyReLU
- Décodage (Upsampling)
 - 7 couches de décodage qui réintroduisent des détails spatiaux en utilisant des skip-connections de type U-Net. (2, 2, 1024) à (128, 128, 128).
 - Une convolution transposée 2D (4*4)
 - Une normalisation InstanceNorm
 - Une activation LeakyReLU
 - Un dropout de 50% appliqué au trois premières couches
- La dernière couche effectue une convolution transposée pour avoir une sortie à 3 canaux, avec la fonction d'activation tanh (valeur entre -1 et 1).

Structure des discriminateurs (PatchGAN) :

- 3 couches d'encodage de (32, 32, 256) à (128, 128, 64) composées de :
 - Une convolution 2D (4*4)
 - Une normalisation InstanceNorm
 - Activation LeakyReLU
- 1 couche de convolution sans stride, suivie d'une InstanceNorm et d'un LeakyReLU.
- 1 couche finale de convolution (4*4)

c. Fonction de perte

Le CycleGAN utilisé considère plusieurs fonctions de pertes pour combiner les aspects adversariaux des GAN et la consistance cyclique décrite plus haut. L'entraînement se fait donc en calculant les loss suivantes :

- Perte du discriminateur : le discriminateur est pénalisé quand il échoue à distinguer les images réelles de celles générées, elle est calculée en utilisant la Binary Cross Entropy (1 pour les images réelles, 0 sinon).
- Perte du générateur : pénalisé en fonction de sa capacité à tromper le générateur.

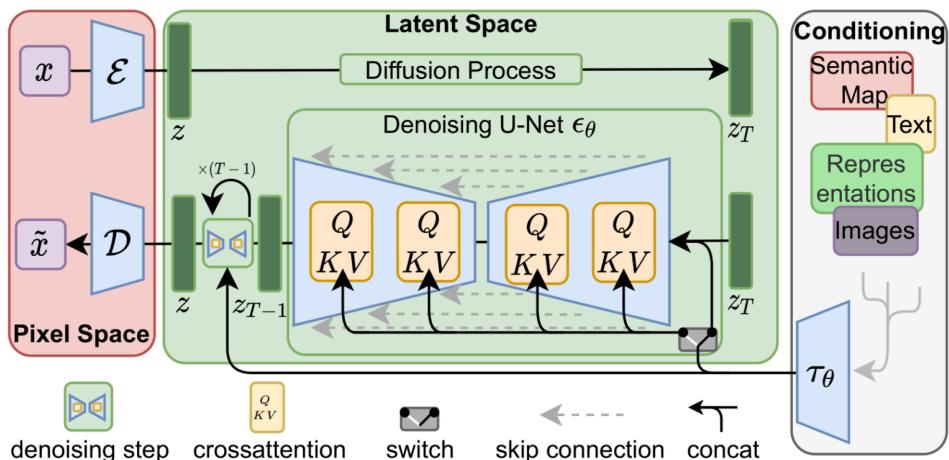
- Perte cyclique : Distance moyenne absolue après une transformation successive de la même image (après un cycle complet).
- Perte d'identité : garantit qu'une image déjà dans le domaine cible ne change pas si on la passe dans le générateur inverse. Utilise la MAE également.

3. Stable Diffusion with Fine-Tuning

a. Présentation de Stable Diffusion

Stable Diffusion (SD) est un modèle génératif développé par Stability AI. Il est utilisé principalement en text -> image. SD est un modèle de diffusion, il opère donc en apprenant à débruiter des images réelles (entraînement), puis lorsqu'on lui donne du bruit pur il va chercher à reconstruire une image cohérente. Ce modèle de diffusion est augmenté de CLIP permettant une interaction textuelle, qui donne un conditionnement au décodage. Il utilise également U-Net lors du processus de denoising. Toutes ces technologies s'appliquent et permettent de travailler dans un espace latent. Bien que plus complexes conceptuellement, ces technologies donnent beaucoup de flexibilité à l'espace.

b. Principes de bases

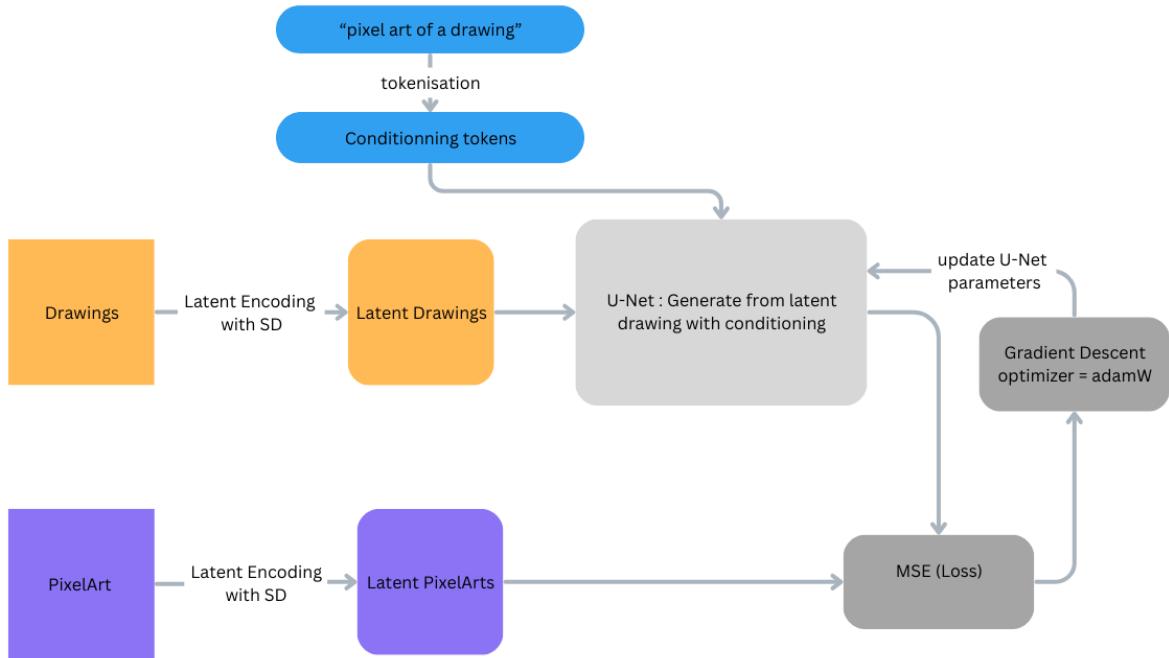


Le fonctionnement de SD est Bruit->VAE->Conditionnement->VAE->Image.

L'intégralité de l'entraînement original du latent space (VAE) ne nous intéresse pas ici, on cherche juste à entraîner une version de SD dont le VAE va être ajusté/fit au conditionnement de pixel art, et avec des inputs non pas bruités mais des vrais dessins. Avec une simple fonction de loss sur les VAE attendus (pixelart encodés avec VAE), et ceux générés (en ajoutant le conditionnement et débruitant une image dessinée). L'idée nous est venue en cherchant des combinaisons de dessins/pixel art pour un apprentissage classique. Celles-ci sont très compliquées à trouver, mais très faciles à générer avec DALL-E ou SD. Cependant utiliser un tel dataset artificiel serait simplement apprendre à un modèle à

reproduire un diffuser existant. Ceux-ci travaillent dans un espace latent et pourrait potentiellement contourner le problème d'input/target non corrélés en apprenant le concept de "traduire en pixel art"

c. Architecture



L'architecture est assez simple, on essaye de reproduire un workflow similaire à un modèle d'apprentissage classique. La complexité résidait surtout dans l'absence de combinaison pixel art/drawing pour l'entraînement.

Le passage par le latent space de SD nous permet d'apprendre à l'U-Net le concept de pixel art pour pouvoir le reproduire. Cet apprentissage se fait comme tout apprentissage avec un dataset input, output, une loss (MSE) et une descente de gradient avec optimiseur (AdamW). L'intérêt réside dans l'ajout de token "pixel art of a drawing" à l'U-Net, pour le conditionner à nous fournir un espace latent qui traduit des dessins en pixel art.

III. Détails de l'Entraînement

1. Plateforme et Infrastructure Utilisée

Dans un premier temps, le CycleGAN a été entraîné sans GPU, en utilisant Tensorflow. L'entraînement a duré 10h au total pour faire que 9 epochs. Ensuite, le code a été repris mais en configurant Tensorflow Metal pour entraîner le modèle sur une puce M3. Cette fois-ci, l'entraînement a duré 10h pour faire 47 epochs, avant que le notebook ne plante à l'epoch 48. L'entraînement a été fait sur des batch de taille 32.

Pour ce qui est du Style Transfert, l'entraînement a été réalisé également sans GPU et avec TensorFlow, et il dure environ 30 minutes sur un macbook doté d'une puce M3. Il comprend 10 epochs de 100 étapes chacunes.

Pour Stable Diffusion nous avons utilisé une session Google Colab doté d'un A100 avec 40 Gb de Ram. 10 epoch on pris environ 4h et 40 unités de compute Colab, soit 4 eur. Les manipulations se sont fait principalement en PyTorch et avec la librairie diffusers de HuggingFace, qui inclus une pipeline pour SD.

2. Stratégies d'Optimisation

Pour optimiser l'entraînement de nos modèles, nous avons utilisé l'optimiseur Adam pour le transfert de style et CycleGAN, en raison de son efficacité éprouvée en deep learning. Pour le modèle SD, nous avons opté pour AdamW, reconnu comme une amélioration d'Adam basée sur des observations empiriques et particulièrement populaire avec les architectures de type diffusers.

3. Prétraitement des Données

Nous disposons d'un dataset de 7000 images Pixel Art. (respectivement 1000 de chaque taille 1*1, 2*2, 3*3, 4*4, 5*5, 6*6, 7*7). Plus la taille des cellules est élevée, plus l'effet "Pixel-Art" est marqué. Nous avons également un dataset de 4235 dessins, avec un style "cartoon" en majorité.

- Les images sont normalisées entre -1 et 1.
- Les images sont redimensionnées en 286*286*3 en utilisant la méthode du plus proche voisin de Tensorflow.
- Les images sont découpées aléatoirement (256*256*3) pour simuler les variations de positions de l'objet (pour limiter l'overfitting).
- Symétrie horizontale aléatoire pour avoir des variations d'orientation (limite l'overfitting).

Pour entraîner le CycleGAN, nous n'avons conservé que les pixel arts avec une taille de cellule importante (>3*3), car nous craignions que ceux avec un style pixelisé trop léger

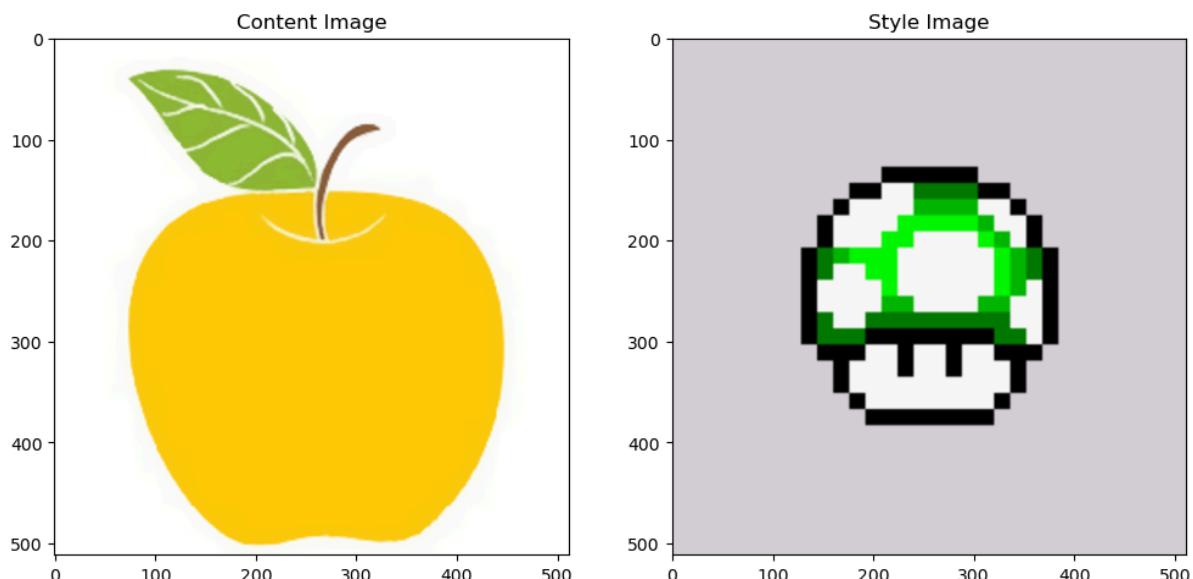
pourraient fausser l'entraînement. Cependant, cela a grandement réduit le nombre de données utilisables.

IV. Analyse des Résultats Obtenus

1. Résultats Qualitatifs

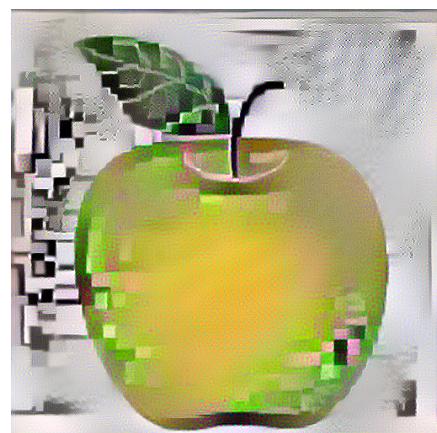
a. Style Transfert avec VGG-19

Dans le cadre du transfert de style utilisant VGG-19, nous avons fourni deux images distinctes : une image de contenu et une image de style, présentées ci-dessous.



L'image de contenu sert à définir la structure et les éléments principaux de l'image générée, tandis que l'image de style apporte les caractéristiques artistiques et esthétiques souhaitées.

Résultat obtenu avec la paire d'image :



Le transfert de style semble avoir bien compris l'objectif de créer un effet pixelisé caractéristique du pixel art. Cependant, certains problèmes sont visibles dans le résultat final. En effet, la partie gauche de la pomme ne devrait pas être pixelisée, de plus, les contours ainsi que certaines parties de la pomme apparaissent trop lisses, ce qui va à l'encontre de l'aspect granuleux typique du pixel art. En revanche, la feuille de la pomme est relativement convaincante et respecte mieux les attentes stylistiques. Par ailleurs, les couleurs de l'image de contenu semblent avoir influencé le style pixel art, entraînant l'apparition de teintes vertes dans l'image finale. Cette influence colorimétrique suggère que le modèle a intégré des éléments de l'image source au-delà des simples caractéristiques stylistiques, impactant ainsi la fidélité au style souhaité.

b. CycleGAN

Pour le CycleGAN, notre objectif était de pouvoir transformer une image avec un style Pixel-Art en une image avec un style d'illustration moins pixelisé, plus cartoon, et inversement.

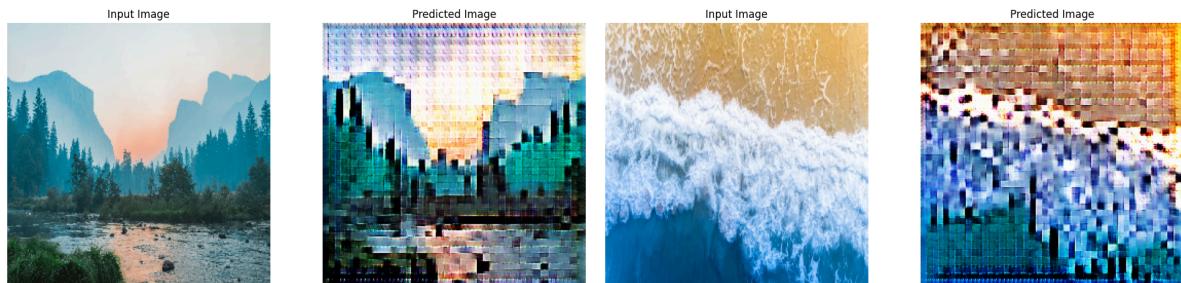
Notre modèle, entraîné sur 47 epochs, génère les images suivantes pour la tâche “dessin (à gauche) vers pixel (à droite)” :



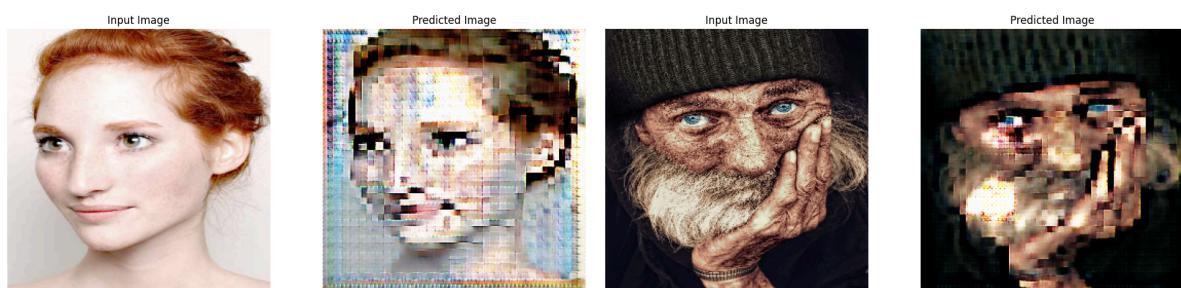
On observe d'abord que les images générées sont quadrillées avec des cellules de mêmes tailles pour chaque image, ce qui donne cet aspect pixelisé. Le modèle a donc bien compris le style à reproduire. Mais pour certaines images, la couleur des cellules n'est pas uniforme, il y a un mélange de blanc et de bleu par exemple, ou bien elles sont d'une couleur différente de l'image d'origine. Cet effet est moins visible sur les données d'entraînement (images du haut), ce qui peut laisser penser que le modèle soit overfit. Ce qui est possible puisque l'on a entraîné le modèle sur 3200 Pixel-Arts seulement, car on n'a gardé que celles avec une taille de cellule importante pour avoir un style plus marqué. Les

cellules de couleur non uniforme apparaissent aussi surtout sur les images avec un fond blanc ou gris.

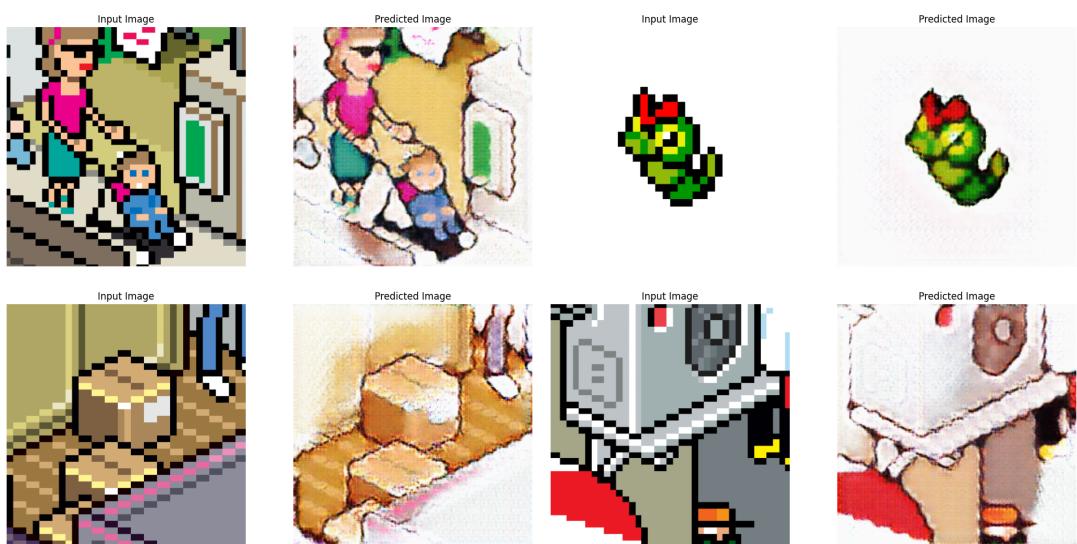
On remarque également que le modèle arrive plus ou moins à transformer une image de paysage réelle, non illustrée, en une image pixelisée, sachant que le modèle n'a pas été entraîné sur ce type d'image :



Nous pensons que ce qui explique ces résultats est que le style Pixel-Art soit si marqué et distinctif, et que les paysages sont si similaires à des dessins ou peintures très réalistes. Car lorsqu'il s'agit de vrais visages humains, les résultats ne sont pas aussi satisfaisants :



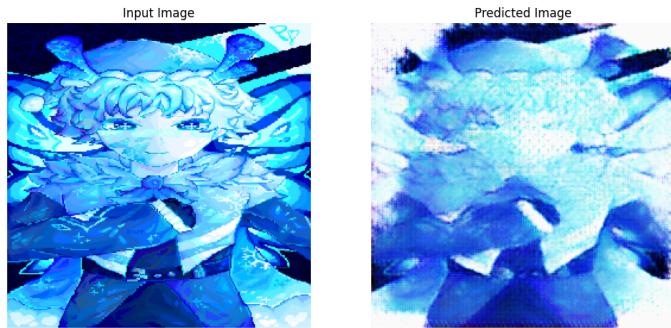
Cette fois-ci, nous nous intéressons à l'opération inverse : Pixel-Art (à gauche) vers Dessin (droite).



Le modèle est plus convaincant sur cette tâche. Globalement, on observe que le modèle essaie de lisser au mieux les bordures, et les angles vifs caractéristiques du pixel-art sont atténuées. Même si l'effet "escalier" est transformé en effet "tremblement" ou flou autour des bords.

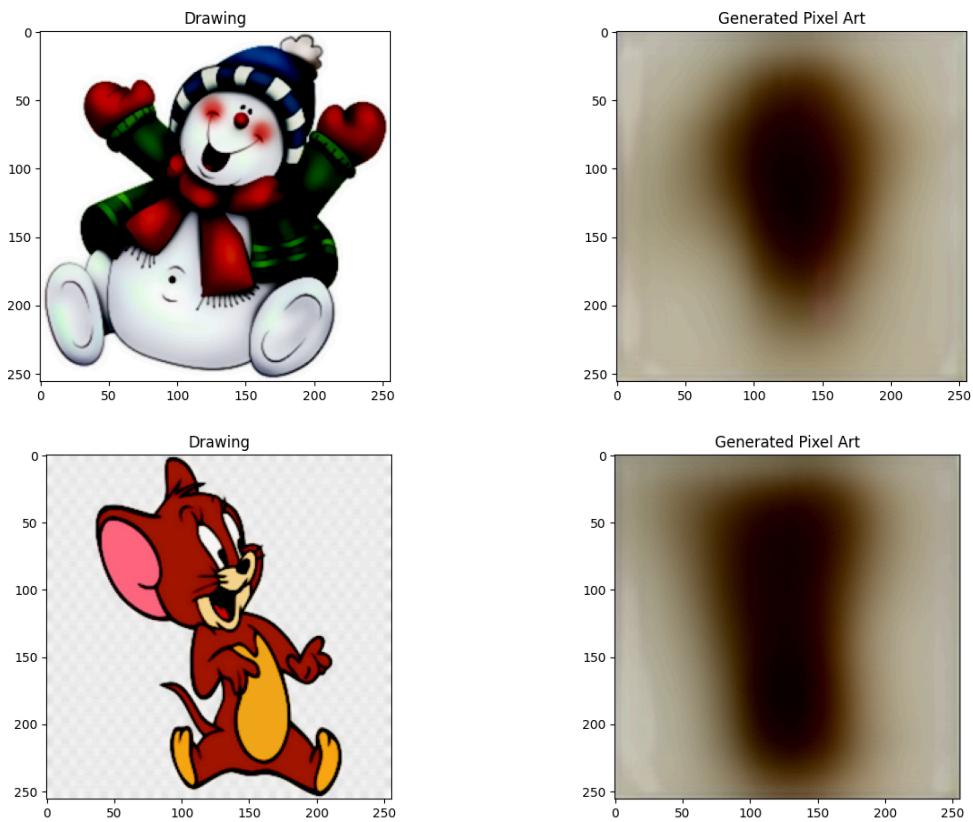
En effet, les couleurs sont ici plus étalées voir effacées, ce qui donne cette impression de flou, voire même de zone sans couleur ou trop blanche sur les images ci-dessus.

C'est encore pire lorsque l'on a un pixel art avec des contours peu marquées, et des couleurs adjacentes similaires comme celle-ci :



c. SD Fine-Tuning

Les résultats ne sont vraiment pas bons. Les formes sont à peine reconnaissables et les couleurs sont toujours les mêmes, comme une photo brûlée ou surexposée.



2. Résultats Quantitatifs

a. Style Transfert

Pour évaluer la qualité des images générées par le transfert de style, nous avons calculé la perte de variation totale (Total Variation Loss) de l'image résultante. La fonction `total_variation_loss` mesure la quantité de variation entre les pixels adjacents, favorisant ainsi la douceur et la cohérence spatiale de l'image. Dans notre cas, la perte de variation totale obtenue est de 122 819,805. Cela suggère la présence de certaines irrégularités ou de variations de couleur indésirables dans l'image générée.

b. CycleGAN

Pour quantifier la qualité de notre CycleGAN, nous avons décidé de comparer la reconstruction loss sur le dernier epoch et l'un des premiers epochs de notre modèle à deux étapes différentes. L'idée était que si après transformation de pixel à dessin, puis de dessin à pixel (ou l'inverse) on retombe sur une image similaire à celle en entrée, et donc que la distance absolue entre les deux est moindre, alors le modèle n'est pas mauvais.



Après 47 epochs - Dessin vers Dessin (Exemple) - Cycle Loss moyen = 0.3208



Après 9 epochs - Dessin vers Dessin (Exemple) - Cycle Loss moyen = 0.1887



Après 47 epochs - Pixel vers Pixel (Exemple) - Cycle Loss moyen = 0.3208



Après 9 epochs - Pixel vers Pixel (Exemple) - Cycle Loss moyen = 0.2072

Cependant, on voit les limites si on considère uniquement cette métrique. Puisque le modèle peu entraîné (9 epochs) ne transformant quasiment pas, ou peu, l'image d'entrée, il est plus probable de retomber sur une image similaire à l'original, et donc que cette loss soit moindre. On aurait eu des résultats plus pertinents en considérant la loss du discriminateur en plus, mais aussi en comparant deux structures différentes (U-Net et ResNet par exemple), entraîné sur un nombre d'epoch équivalent. Le papier “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks” considère la FCN score.

c. SD Fine-tuning

Intéressant on obtient pas de meilleure loss après le premier epoch. Au terme de celui-ci on a environ 20% de loss, les epoch suivants gravitent autour de cette valeur. A savoir si la loss a une signification.

3. Discussion critique sur les résultats

a. Style Transfert

Pour ce qui est du transfert de style, chaque paire d'images (image de contenu et image de style) nécessite un ré entraînement du modèle, ce qui rend cette approche peu scalable pour des applications à grande échelle. De plus, les résultats obtenus ne sont pas satisfaisants. Ces limitations font du transfert de style une méthode peu pratique et inefficace pour des projets nécessitant la transformation simultanée de nombreuses images.

b. CycleGAN

Globalement, les résultats sont satisfaisants, malgré les problèmes soulignés plus tôt, à savoir l'étalement (ou l'effacement) des couleurs dans les dessins, l'apparition de cellules de couleur non uniforme (pixellisation). Afin de limiter ces problèmes, nous proposons les pistes d'explorations suivantes :

- Modifier la valeur des padding dans les filtres de convolution, car on remarque que dans les images générées (Drawing to Pixel), les cellules de couleurs différentes (bleus, jaunes) se trouvent souvent sur les bords de l'image.
- Augmenter davantage le nombre d'images pour éviter les problèmes d'overfitting soulignés plus haut. Également ajouter ou augmenter la valeur des couches de dropout à cette fin.

c. SD Fine-Tuning

Les résultats sont très décevants, surtout en partant d'un modèle aboutit comme Stable Diffusion... Celui-ci utilise un certain nombre de techniques assez avancées, comme les U-Net, CLIP, et des interactions entre différents espaces latents. Les résultats sont très compliqués à interpréter. Il était assez ambitieux de vouloir manipuler un outil abouti et state of the art comme Stable Diffusion sans entraînement ou éducation préalable sur toutes ses composantes.

Utiliser un diffuseur pour générer des images n'est pas en soi une mauvaise idée, mais ceux-ci ne sont pas idéaux pour la transformation. Je pense que c'est possible d'apprendre à l'espace latent de "traduire" entre 2 styles et de l'extraire du modèle. Je pense donc que l'erreur dans notre projet est technique et pas conceptuelle.

Les modèles et encodeurs sont *extrêmement* gros, ils demandent beaucoup de VRAM à manipuler. Pour avoir des temps d'entraînement tolérables on a recours à Google Colab Pro. Même avec 40 GB de VRAM on a 4h (4 eur) pour 10 epoch. Et 1h40 (2 eur) pour 1 epoch. Le processus itératif est également très compliqué, on ne peut pas juste entraîner un modèle, puis corriger les erreurs a posteriori.

V. Conclusion

1. Problèmes rencontrés et solutions

Le principal problème rencontré est le temps nécessaire pour entraîner les modèles utilisés. Par exemple, pour CycleGAN, même en configurant une version optimisée de Tensorflow pour les puces Mac, un epoch prenait toujours plus de 20 minutes. Et comme il fallait au moins une trentaine d'epoch pour avoir un transfert de style acceptable sur notre dataset, il était difficile de tester des hyperparamètres différents ou même légèrement modifier l'architecture de nos U-Net et PatchGAN.

2. Répartition du travail

Au cours du projet, Yanis s'est concentré sur le transfert de style, Gernido a pris en charge la mise en œuvre de CycleGAN, et Corentin a travaillé sur Stable Diffusion. Avec cette répartition des tâches, nous avons maintenu une communication constante au sein des séances mais également en distanciel ce qui nous a permis de nous entraider les uns les autres et de connaître l'avancement de chacun.

3. Synthèse

Dans ce projet, nous avons exploré trois techniques de deep learning pour la conversion d'images dessinées en pixel art : le transfert de style, CycleGAN et Stable Diffusion. Parmi ces méthodes, CycleGAN s'est révélé être le modèle le plus adapté, offrant des résultats plus cohérents et de meilleure qualité pour notre tâche spécifique. Bien que nous soyons globalement satisfaits des résultats, nous avons été déçus par les performances du transfert de style, qui n'a pas atteint les résultats escomptés, ainsi que par les difficultés rencontrées avec Stable Diffusion, qui n'a pas fonctionné comme prévu. Une

piste d'amélioration aurait pu être d'entraîner un modèle personnalisé spécifiquement conçu avec notre dataset (et non le VGG-19) pour le transfert de style et explorer d'autres méthodes de deep learning liées au transfert de style,

VI. Bibliographie

- Lien du GitHub : <https://github.com/Atatra/draw2pix>
- https://www.tensorflow.org/tutorials/generative/style_transfer?hl=fr
- <https://www.tensorflow.org/tutorials/generative/cyclegan?hl=fr>
- <https://github.com/huggingface/diffusers>
- Dataset : <https://github.com/WuZongWei6/Pixelization>