université de BORDEAUX

Projet de fin d'étude Master 2 Informatique pour l'image et le son

Génération d'images de documents synthétiques à l'aide de réseaux de types GAN

Hadrien Delengaigne
Victor Pivetaud
Charlie Schlick

Encadré par
Boris Mansencal
Nicholas Journet
Antoine Pirrone

24 mars 2019

Abstract

Today, algorithms for image processing or analysis need a lot of labeled data in order to be tested. DocCreator [1] is an open source software that can create synthetic document images which can be used as data to test these algorithms. The goal of our project was to explore the use of a new type of neural network called GAN (Generative Adversarial Network) in this problematic. After several researches and tests on different architectures of GAN we were able to adapt an existing model, MC-GAN [2] to accurately complete the fonts of DocCreator.

Résumé

Aujourd'hui, les algorithmes de traitements ou d'analyses d'images ont besoin de beaucoup de données labélisées afin de pouvoir être testés. DocCreator [1] est un logiciel open source capable de créer des images de documents synthétiques qui peuvent donc servir de données pour tester ces algorithmes. L'objectif de notre projet était d'explorer l'utilisation d'un nouveau type de réseaux de neurones appelé GAN (Generative Adversarial Network) dans cette problématique. Après plusieurs recherches et tests sur différentes architectures de GAN nous avons pu adapter un modèle existant, MC-GAN [2] pour compléter fidèlement les fontes de DocCreator.

Table des matières

1 Introduction

Ces dernières années, les réseaux de neurones convolutionnels (RNC ou CNN en anglais) ont révolutionné l'informatique graphique en surpassant les résultats obtenus par les hommes dans la classification d'images, la détection de visage ou encore la reconnaissance de parole. Cependant ce genre d'algorithme a besoin de nombreuses images pour fonctionner. Le groupe document du LaBRI a créé un outil pour répondre à ce besoin : DocCreator [1]. Cet outil permet de générer des images synthétiques de documents anciens en se basant sur des images de vrais documents. Les images produites sont automatiquement labélisées afin de pouvoir s'en servir dans des algorithmes d'apprentissage en tant que vérité terrain.

Afin de pouvoir générer des images de documents anciens, DocCreator a besoin de fontes anciennes. Une fonte est l'ensemble des caractères correspondant aux mêmes caractéristiques de corps, graisse et italique au sein d'une même fonte[1]. Actuellement, le seul moyen d'obtenir ces fontes est de les extraire interactivement à partir de vraies images d'anciens documents. Or, même si ce travail s'appuie sur les résultats d'un OCR (reconnaissance de caractères), il reste très long. En outre, dans la plupart des cas il ne permet pas d'avoir une fonte complète. En effet, si par exemple le document d'origine ne contient à aucun moment la lettre 'H' en majuscule, la fonte que l'on obtiendra en sera dépourvue. De plus, il est préférable d'avoir plusieurs exemplaires d'une même lettre afin d'obtenir des images plus réalistes dans le cas de document ancien (simulation de l'usure de l'appareil d'impression) et d'augmenter le nombre d'images que l'on peut produire. Mais là encore, cela dépend de la richesse du document d'origine. C'est donc dans le but de résoudre ces problématiques que nos clients ont proposés ce sujet. Pour ce faire, nous devions regarder si les réseaux de neurones de type GAN (Generative Adversarial Network) pouvaient servir de solution.

Les GAN sont des réseaux de neurones capables de générer des images. Dans notre cas, ils nous permettraient d'obtenir des caractères qu'on ne pourrait pas obtenir par extraction. Cependant, ces réseaux ont besoin d'un ensemble de données d'entraînement important ainsi que d'une forte puissance de calcul en parallèle afin de produire des images pouvant être confondues avec les images d'entraînement.

[1] `https://fr.wikipedia.org/wiki/Police_d'ecriture#Police_et_fonte`

2 Étude de l'existant

2.1 DocCreator

DocCreator est donc un logiciel open-source développé en C++ par Nicholas Journet et al. [1] qui permet de créer un grand nombre de documents synthétiques permettant d'enrichir des bases de données servant à l'entraînement d'algorithmes d'analyses et de reconnaissance d'images.

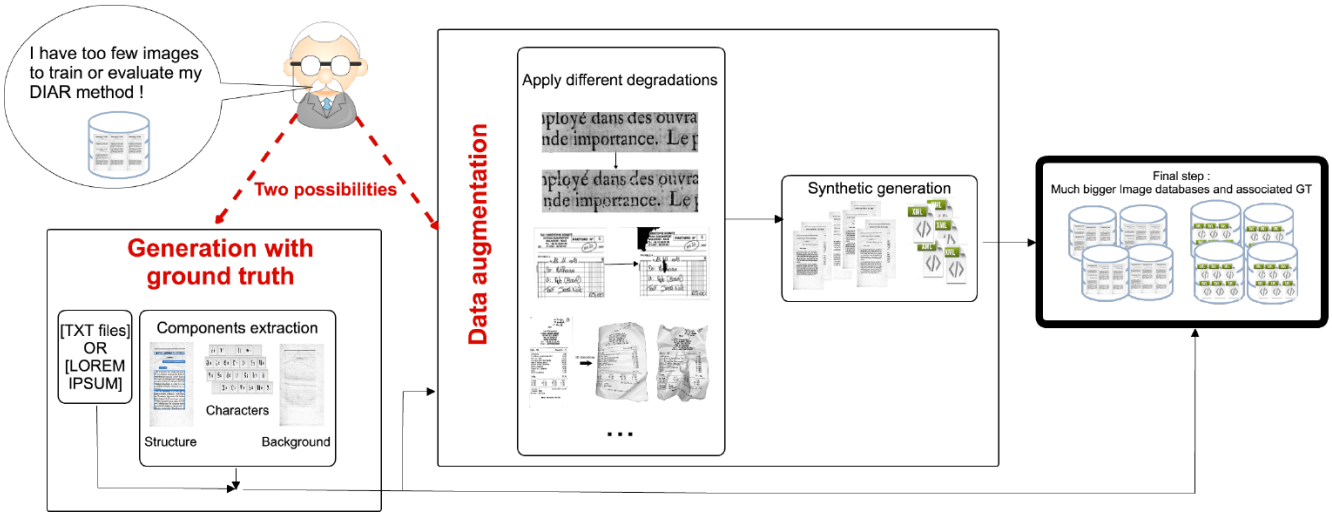

FIGURE 1 – Exemple d'utilisation de DocCreator[2]

Il peut par exemple générer de vrais documents en utilisant des fontes qui ont été préalablement extraites de documents réels, en structurant le texte et en utilisant un support (background) afin de rendre le document généré plus réaliste (figure 1). Il peut aussi servir à modifier des documents réels existant, en appliquant des dégradations, de telle sorte que l'image résultante soit perçue différemment par un algorithme de reconnaissance d'images mais puisse être labelisée automatiquement. Ce logiciel permet donc d'augmenter la taille d'une base de données existante, en rajoutant de la diversité dans les documents et en faisant correspondre les images générées avec leurs vérités-terrain.

[2] `http://doc-creator.labri.fr/` [1]

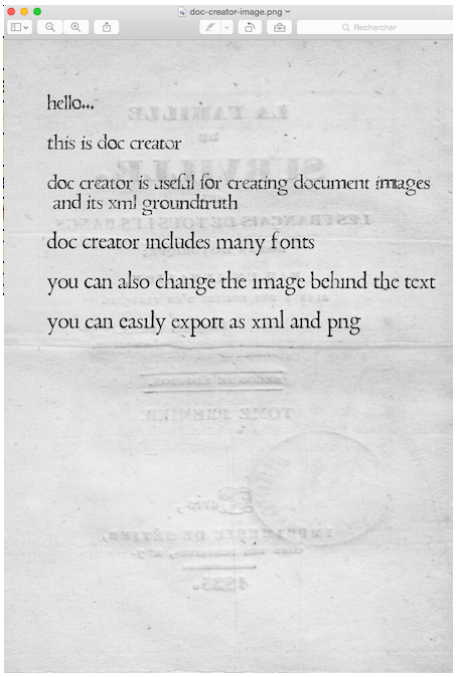

FIGURE 2 – Exemple de document créé sur DocCreator

C'est pourquoi dans une optique de génération de documents on peut envisager de se servir des GAN que ce soit pour essayer de générer directement des documents ou pour générer des éléments utiles à DocCreator tels que de nouvelles fontes. Cependant le format d'exportation des fontes de DocCreator étant en XML il faudra également pouvoir extraire les fontes de DocCreator pour s'en servir avec des GAN.

Les problématiques de DocCreator que nous allons tenter de résoudre sont donc :

- la complétion de fonte
- la création de nouvelle fonte

2.2 Réseau de neurones

Comme présenté dans l'introduction, les réseaux de neurones convolutionnels sont capables d'obtenir des meilleurs résultats que les êtres humains. Pour fonctionner, les CNN utilisent plusieurs couches de convolutions qui vont chercher différents motifs dans l'image que l'on donne au réseau. Ces couches sont souvent suivies par des couches de *pooling* ou relu afin que la prochaine couche de convolution puisse chercher des motifs différents de la précédente.

Chaque couche de ces réseaux possède des poids qui sont mis à jour à de nombreuses reprises pendant l'entraînement. Une fois que toutes ces données labélisées sont passées dans le réseau, ce dernier examine ensuite ses performances et modifie plus ou moins les poids de chaque couche en fonction de la qualité des résultats. C'est cet entraînement suivi de la modification des poids que l'on appelle une *epoch*. Pour être correctement entraîné, un RNC a besoin de réaliser de nombreuses *epochs*. Après quoi on récupère l'ensemble des poids pour l'appliquer sur les données que l'on veut traiter. Il n'est donc nécessaire d'entraîner le réseau qu'une seule fois.

Les réseaux antagonistes génératifs (RAG ou GAN en anglais) sont eux aussi des réseaux utilisant des couches de convolutions. Cependant, ils ne sont pas conçus pour analyser ou classer des données mais pour en générer. Ces réseaux ont été popularisés en 2014 par Ian J. Goodfellow et al. [3]

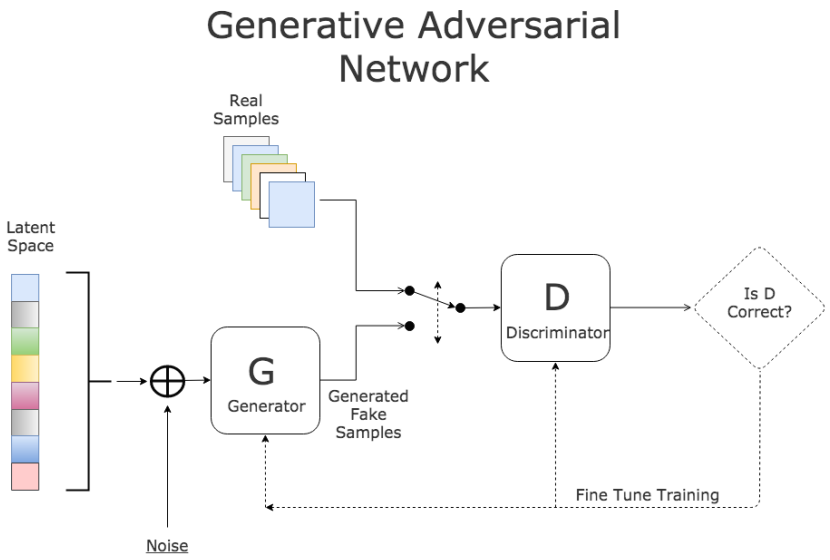

FIGURE 3 – Schéma d'un GAN[3]

Les GAN diffèrent également des CNN par leur fonctionnement. Comme on peut le voir sur la figure 3, ils sont composés de deux réseaux en compétition. Le réseau générateur "G" qui doit générer les données que l'on souhaite et un réseau discriminant "D" dont le rôle est de déterminer si la donnée qu'il reçoit est une vraie image ou si elle a été générée par le réseau précédent. Après quoi les poids des deux réseaux sont modifiés selon le résultat de la prédiction du réseau discriminant. Plus l'entraînement réalise d'*epoch* plus le réseau discriminant sera efficace pour discerner les vraies données de celles qui ont été générées, ce qui permet au réseau générateur de produire des images de plus en plus vraisemblables rapidement. Contrairement à un CNN, l'ensemble de données

[3] `https://mse238blog.stanford.edu/2017/08/teun/generative-adversarial-networks-the-future-of-deep-learning`

d'entraînement n'est pas utilisé en entrée du réseau mais il est fourni uniquement au réseau discriminant. Les données d'entraînement servent donc de facteur de comparaison pour évaluer les images que génèrent le réseau.

FIGURE 4 – Exemple d'images produites par NVIDIA[4]

Malgré le fait que les GAN soient très récents, ils sont déjà en mesure de produire des images capables de tromper un être humain, comme on peut le voir sur la figure 4. L'un des exemples les plus connus est celui proposé par les chercheurs de NVIDIA [4]. Leur réseau est conçu pour produire des images de visage humain de grande résolution à partir d'un nombre important de photographies de célébrité. Afin d'éviter de générer directement des images de grandes tailles et dans le but de réduire la durée d'entraînement, ils entraînent le réseau progressivement, en commençant par des images de petites tailles. Ils récupèrent le modèle de ce premier entraînement et s'en servent pour générer des images un peu plus grandes. Ils répètent ensuite ce procédé jusqu'à pouvoir générer des images de la taille souhaitée. Cependant, malgré cette optimisation il leur a quand même fallu entraîner le réseau pendant 4 jours sur 8 processeurs graphiques Tesla V100 pour produire ces résultats.

[4]Progressive growing of gans for improved quality, stability, and variation, Karras et al. 2017

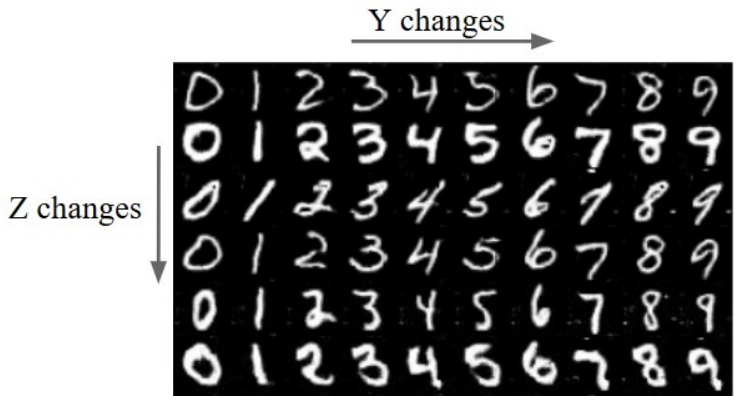

FIGURE 5 – Exemple d'images produites par un GAN sur le dataset MNIST [5]

Il y a également plusieurs travaux sur des GAN produisant des caractères [5], notamment à partir de l'ensemble de données MNIST qui contient des images de chiffres labélisées. On peut également voir sur la figure 5 qu'il est possible de générer des chiffres qui ont un style similaire. Pour ce faire, au lieu de donner du bruit aléatoire en entrée du réseau générateur, on lui donne des informations sur le style des caractères à générer (épaisseur, taille, rotation, etc). Le fonctionnement de cette variante de GAN est détaillé dans les travaux de Augustus Odena et al [6].

2.3 GAN dans la génération de document

Afin de résoudre les problématiques de notre client, nous avons exploré deux travaux utilisant des réseaux de type GAN appliqués sur des caractères dont le code était disponible en open-source.

2.3.1 Multi-Content GAN for Few-Shot Font Style Transfer

Dans leur article [2], Samaned Azadi et al. présentent un modèle capable de compléter des images de caractères en générant des lettres non présentes tout en gardant le style de la fonte apprise. Les auteurs ont rendu le code de leur méthode disponible librement sur GitHub[6].

FIGURE 6 – Exemple d'utilisation de MC-GAN [2]

[5] http://guimperarnau.com/blog/2017/03/Fantastic-GANs-and-where-to-find-them
[6] https://github.com/azadis/MC-GAN

Dans la figure 6, la ligne du haut correspond à la fonte originale, la ligne du bas est l'image que le réseau a complété à partir des caractères encadrés en rouge. On remarque qu'à partir de seulement 8 caractères ce réseau permet déjà de générer des caractères ayant un style identique à celui du style de base.

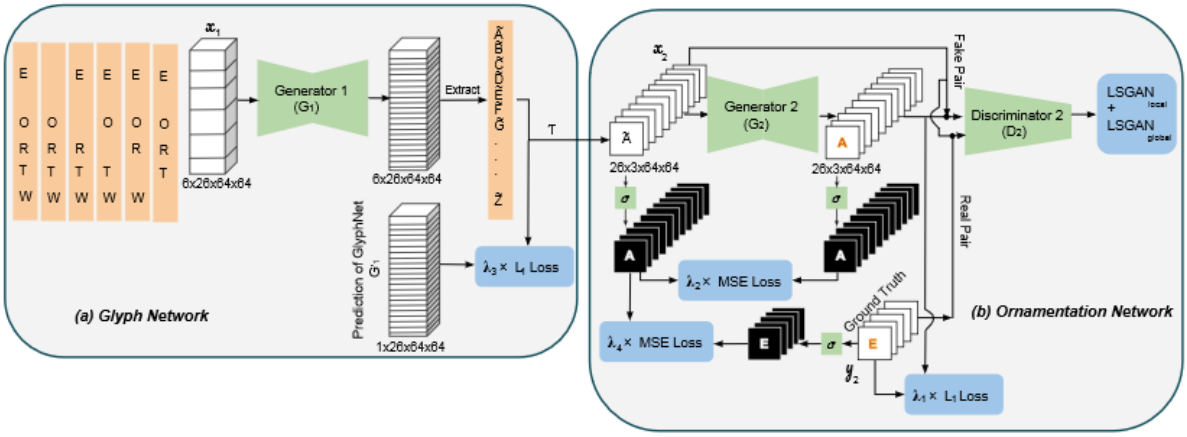

FIGURE 7 – Schéma de l'architecture du réseau de MC-GAN [2]

MC-GAN est divisé en deux réseaux GAN distincts. Un premier réseau appelé GlyphNet (A) qui a pour but de générer la forme des caractères et un deuxième réseau appelé OrnaNet (B) qui permet de rajouter de la couleur et certains ornements aux caractères créés par GlyphNet. Dans un premier temps l'utilisation de MC-GAN nécessite un pré-entraînement du réseau GlyphNet seul sur plusieurs milliers de fontes. Dans un deuxième temps, on utilise les deux réseaux joints (figure 7) en les entraînant spécifiquement sur la fonte que l'on veut compléter. Comme ils ne disposent pas de toutes les lettres puisque ils cherchent à les générer, ils ne peuvent pas comparer les caractères créés aux caractères originaux. Ils utilisent donc le réseau précédemment entraîné pour générer des images de comparaisons.

2.3.2 GlyphNet

Comme on peut le voir sur la figure 7, les 6 images en entrée du réseau sont de taille $26 * 64 * 64$ pixels, ce qui correspond aux 26 lettres de l'alphabet chacune ayant une taille de $64*64$ pixels. La construction des images fournies en entrée se fait de la façon suivante : dans la première seul un groupe de 5 lettres est conservé (tous les autres caractères sont effacés), les 5 images suivantes sont produites en retirant tour à tour une des 5 lettres initialement retenues. Chacune des 5 images ne contenant que 4 caractères va tenter de générer le 5^{eme}, la première image servira de référence pour faire des comparaisons (Voir exemple sur la figure 7 où sont fournies en entrée des images avec les caractères "E O R T W"). Les images fournies à GlyphNet sont en niveaux de gris, car

seule la forme des caractères est générée par ce réseau.

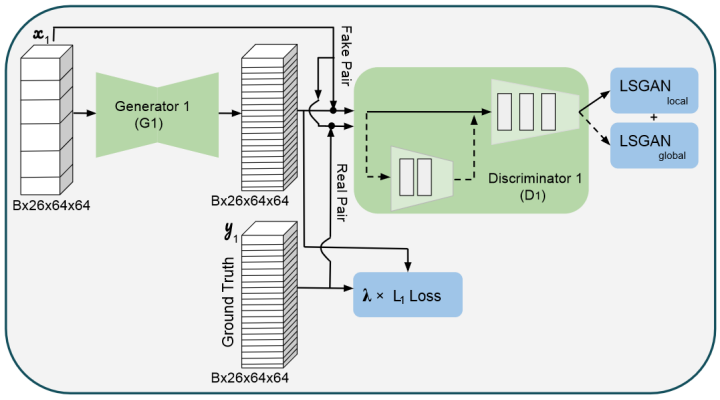

FIGURE 8 – Schéma de l'architecture du réseau de GlyphNet [2]

Le réseau générateur "G1" sur la figure 8 reçoit donc des images dans lesquelles certains caractères ont été effacés. La sortie du générateur est une image de même taille, contenant les caractères fournis au générateur, mais également les caractères manquants. Une fonction de perte LAD (*Least Absolute Deviation*) permet de pénaliser les différences entre les images générées et les images originales contenant tous les caractères.

Le réseau discriminant "D1" sur la figure 8 permet ensuite de différencier les images originales des images générées en utilisant, d'un côté un discriminant local inspiré du modèle PatchGAN [7] afin de vérifier l'authenticité des caractères, et d'un autre côté un discriminant global vérifiant le réalisme global de la fonte. À la sortie de ces deux réseaux discriminants une fonction de perte LSGAN [8] est appliquée pour améliorer les résultats.

Ce réseau discriminant "D1" n'est présent que lors du pré-entraînement de GlyphNet et est retiré par la suite lors de la jonction de GlyphNet et OrnaNet.

2.3.3 OrnaNet

La partie OrnaNet du réseau est utilisée uniquement lors de l'entraînement spécifique à la fonte que l'on veut générer. Cela s'explique par le fait que les ornements et les couleurs générés par ce réseau, sont des paramètres très spécifiques à chaque fonte, et ainsi il n'a pas besoin d'un entraînement préalable sur une base de données de fontes.

OrnaNet reçoit en entrée 26 images de taille $3 * 64 * 64$ (figure 7). Chaque image représente une lettre en niveaux de gris avec les 3 canaux RGB étant égaux à l'image en niveau de gris générée par GlyphNet.

Le réseau générateur "G2" sur la figure 8 reçoit donc les images générées par GlyphNet dans un format où les lettres ont été séparées les unes des autres.

Il applique ensuite des transformations pour modifier des détails spécifiques à cette fonte, tels que des motifs particuliers et la couleur.

Le réseau discriminant "D2" est le même réseau que "D1" et permet donc d'évaluer la qualité de la génération en comparant les caractères générés avec les caractères que l'on possèdent déjà.

FIGURE 9 – Ligne 1 : fonte originale. Ligne 2 : fonte en sortie de GlyphNet (niveau de gris). Ligne 3 : résultat final d'OrnaNet (Image extraite de MC-GAN) [2]

MC-GAN permet donc, à partir d'une image d'une fonte incomplète, de générer les lettres manquantes à partir de multiples réseaux GAN. Cette implémentation pourrait donc nous permettre de générer les caractères manquants des fontes de DocCreator. Nous disposons de nombreux caractères connus pouvant être fournis au réseau afin d'obtenir des résultats satisfaisants. C'est à dire, pouvant être confondus avec de vrais caractères extraits de documents anciens.

Néanmoins plusieurs difficultés devront être résolues afin de pouvoir utiliser ce réseau pour répondre aux problématiques rencontrées par nos clients telles que :

- Réussir à refaire l'apprentissage du réseau afin de pouvoir générer d'autres caractères, cela passe par l'utilisation d'un ensemble de données d'entraînement qu'il va falloir trouver ou créer nous même.
- Vérifier si ce réseau fonctionne pour la complétion d'un style particulier de fontes, les fontes anciennes.

2.3.4 zi2zi : Master Chinese Calligraphy with Conditional Adversarial Networks

Le but de ce projet [9], disponible sur GitHub[7], est d'effectuer un transfert de style entre des fontes chinoises. Ce réseau permet, à partir de deux caractères de styles différents, de créer un ensemble de styles de transitions entre ces caractères.

[7] `https://github.com/kaonashi-tyc/zi2zi`

FIGURE 10 – Exemple d'utilisation de zi2zi

Dans l'exemple ci-dessus, la ligne du haut et la ligne du bas contiennent des caractères écrits avec deux fontes existantes. Les 4 lignes centrales correspondent à des étapes intermédiaires de transitions d'une fonte à l'autre et ont été générées par le réseau.

Nous nous sommes intéressés à ce projet car il pourrait nous permettre de répondre à la problématique de nos clients sur le faible nombre de fontes disponibles dans le logiciel. En effet à partir de deux fontes extraites de documents anciens possédant des styles différents, on pourrait être capable de créer plusieurs fontes intermédiaires ayant des styles nouveaux.

Suite aux mauvais résultats obtenus lors des tests sur ce réseau, nous avons décidé de nous concentrer sur la partie MC-GAN et donc la complétion de fonte. C'est pourquoi nous n'aborderons pas zi2zi dans la suite du rapport. Des informations sur la reproduction des résultats de ce réseau sont présentes en annexe A

3 Cas d'utilisation/Scénario

Comme expliqué précédemment le but du réseau MC-GAN est de compléter des fontes, comme tous les autres réseaux de type GAN celui-ci va s'entraîner sur des images et en générer de nouvelles. Afin de faciliter l'utilisation de ce réseau notre but est de fournir des scripts permettant d'effectuer toutes les étapes nécessaires au ré-entraînement du réseau avec un nouvel ensemble de données et à la complétion de fontes.

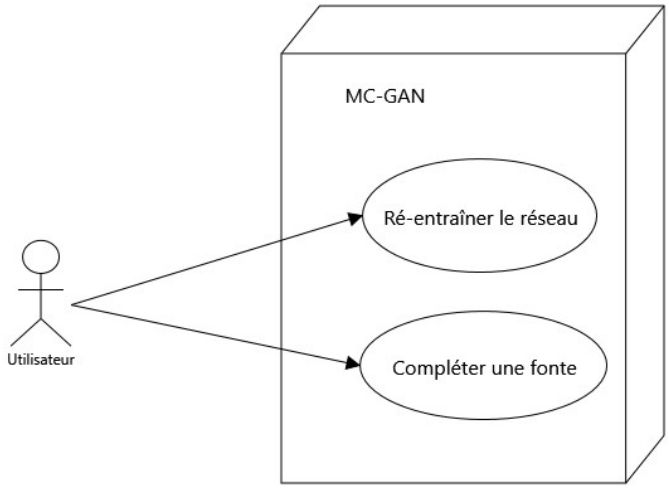

FIGURE 11 – Cas d'utilisation de notre version de MC-GAN

Ré-entraîner le réseau :

- Création d'images d'entraînements à partir de fichiers de fontes (otf/ttf)
- Création de l'arborescence nécessaire à l'entraînement du réseau
- Entraîner le réseau avec l'ensemble de données obtenues

Compléter une fonte :

- Extraction des fontes de DocCreator
- Création des images à donner au réseau
- Complétion des images par le réseau
- Création du fichier .of pour DocCreator à partir de l'image complétée

4 Besoins fonctionnels

4.1 Entraîner GlyphNet

Créer une base de données de fontes : Cette base de données de fontes devra être composée de fichiers .otf et .ttf, comportant tous les caractères latins (majuscules, minuscules, chiffres, ponctuations et caractères accentués).

Créer des images d'entraînement à partir des fichiers de fontes : Pour s'entraîner, le réseau a besoin d'images. Ces images doivent contenir tous les caractères que le réseau devra savoir générer : a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9 ! " # $ % & ' ( ) * + , - . / : ; < = > ? @ [ ] ^ _ ‘ { | } ~ à â ç è é ê î ô ù û À Â Ç È É Ê Î Ô Ù Û.
Les noms des images devront être `fontName.0.0.png`.

Créer l'arborescence nécessaire à l'entraînement de MC-GAN : Les images nécessaires à l'entraînement du réseau doivent suivre une arborescence précise. Il faudra donc créer cette arborescence à partir des images générées précédemment.

Lancer l'entraînement du réseau : Un script permet de lancer l'entraînement du réseau en spécifiant le nom du modèle que l'on veut créer. La modification des paramètres du réseau (nombres d'entrées, de sorties, *batchsize* et nombres *d'epochs*) pourra être faite dans ce script.

4.2 Entraîner Ornanet sur une fonte particulière pour la compléter

Extraire les caractères de DocCreator : Afin de compléter les fontes de DocCreator, il faut pouvoir extraire les caractères déjà présents dans le programme pour chacune des fontes. Les caractères doivent être extraits et enregistrés dans une image au format PNG ayant pour nom la lettre présente dans l'image. DocCreator possédant plusieurs versions d'un même caractère le nom doit aussi contenir un numéro permettant de les différencier. Par exemple : `B_-06.png` ou `!_02.png`.

Créer l'arborescence et les images pour entraîner la deuxième partie du réseau : Pour compléter une fonte, le réseau a besoin d'effectuer un deuxième entraînement dessus, pour ce faire on lui donne une arborescence contenant les images nécessaires.

Appliquer le modèle pour obtenir une image complétée : Un script permet de lancer l'application du modèle en précisant l'arborescence précédemment créée. Les paramètres du réseau doivent être facilement modifiables à

l'intérieur de ce script. A la fin de l'exécution de ce dernier, on doit obtenir l'image contenant les caractères connus avec tous les espaces blancs complétés par le réseau.

Créer le fichier `.of` de la fonte complétée : Une fois l'image complétée on veut pouvoir utiliser les caractères générés dans DocCreator. Ce programme doit prendre en entrée l'image générée par le modèle et retourner un fichier `.of`, étant reconnu par DocCreator.

5 Besoins non fonctionnels

Utilisable sur GPU : L'entraînement et l'application du réseau pour la complétion d'image doivent pouvoir s'effectuer sur des machines équipées d'une carte graphique suffisamment puissante (l'entraînement du réseau prend du temps et est accéléré proportionnellement à la puissance et la mémoire de la carte graphique). Pour nos tests nous allons utiliser la salle 201 du CREMI équipée de processeurs graphiques Nvidia GTX 1070 avec 8Go de mémoire. Nos clients peuvent eux effectuer un entraînement sur une machine du LaBRI possédant une carte graphique 2080Ti avec 11Go de mémoire.

Test d'entraînement du premier réseau inférieur à 24 heures : L'entraînement de la première partie du réseau (avec les images possédant tous les caractères que le réseau doit pouvoir générer) doit prendre moins de 24 heures sur les ordinateurs de la salle 201 du CREMI. Le client n'a pas de contrainte de temps d'entraînement (dans la limite du raisonnable) mais nous ne pouvons pas avoir accès à une machine du CREMI suffisamment puissante pendant plus d'une journée (utilisation par d'autres étudiants, redémarrage tous les matins).

Durée d'entraînement du deuxième réseau inférieur à quelques heures : L'entraînement de la deuxième partie du réseau (sur l'image à compléter) doit être inférieur à quelques heures sur les ordinateurs cités précédemment. Cet entraînement ne sera pas réalisé qu'une seule fois mais sur chaque fonte extraite. Il est donc préférable qu'il soit plus court que le précédent.

Possibilité de générer des caractères spéciaux : Notre implémentation doit pouvoir générer tous les caractères disponibles sur DocCreator : majuscules, minuscules, chiffres, ponctuation et caractères accentués.

Entraînable avec un nombre de données limité : En raison de la difficulté à trouver une base de données d'images ou de fontes possédant tous les caractères qui nous intéressent, le réseau doit être capable de s'entraîner avec un nombre d'images limité. L'auteur du réseau MC-GAN fournit son ensemble d'entraînement qui contient environ 10 000 fontes. Mais nous ne pouvons pas nous en servir car il ne contient que des majuscules. Il nous faudra donc ré-

cupérer ou créer un nouvel ensemble de données d'entraînement qui contienne d'avantage de caractères.

Avoir des résultats visuellement satisfaisants : Les caractères générés par notre réseau doivent pouvoir être confondus avec des caractères pouvant être trouvés dans des documents anciens et suivre le style donné lors de la génération de ces derniers. Cette qualité visuelle étant difficilement quantifiable, nous nous contenterons de vérifier si la qualité est acceptable pour le client.

6 GANTT

	Semaines								
Tâches	1	2	3	4	5	6	7	8	9
Recherche de l'existant									
Reproductions des résultats									
Application à DocCreator									
Création de notre de base de données de fontes									
Ré-entraînement du réseau									
Tests sur la génération									
Rédaction du rapport									

FIGURE 12 – Diagramme de Gantt du projet

La réalisation de ce projet a été très dépendante de l'état de nos recherches et de la conclusion de nos rendez-vous hebdomadaires avec le client. Pour cette raison nous n'avons pas pu créer de diagramme de Gantt au début du projet. La figure 12 a donc été réalisée à la fin du projet et représente les différentes tâches que nous avons effectuées semaine par semaine.

7 Architectures

Nous avons défini les scénarios dans la section 3 du rapport. Afin de les réaliser nous avons produit deux pipelines illustrés en figure 13 et 14 . DocCreator et MC-GAN étant dans deux langages différents, notre architecture se compose de plusieurs scripts permettant la transition d'un outil à l'autre (ils sont représentés par des ellipses oranges dans les schémas). "ExtractImageFromOF" de la figure 14 est représenté différemment car il s'agit d'un programme fourni par nos clients afin d'extraire les caractères de DocCreator. Les carrés verts représentent des états de notre pipeline et les losanges bleus les réseaux utilisés.

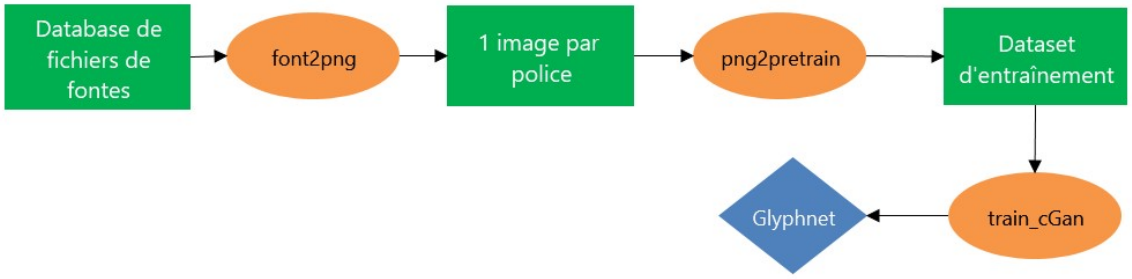

FIGURE 13 – Pipeline pour l'entraînement du réseau

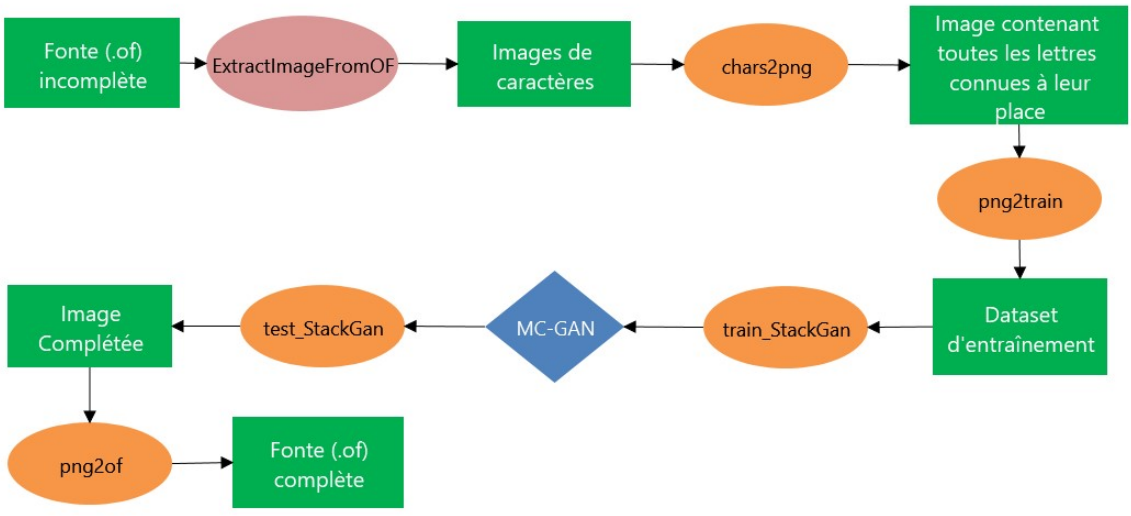

FIGURE 14 – Pipeline pour compléter une fonte

8 Implémentation

La réalisation de ce projet s'est faite en plusieurs étapes, définies par l'état de nos recherches en matière de GAN et des résultats obtenus. Ces différentes étapes vont être brièvement expliquées dans cette partie.

8.1 Reproduction des résultats :

Afin de vérifier si ces réseaux allaient être vraiment utilisables dans ce projet, nous avons tenté de reproduire les résultats obtenus dans la publication de MC-GAN. Pour cela, nous avons récupéré les modèles pré-entraînés utilisés par les créateurs du réseau. C'est aussi à ce moment-là que nous avons dû résoudre un problème auquel nous n'avions pas pensé : les nombreuses contraintes de versions liées à l'utilisation de bibliothèques d'apprentissages. En effet, ces bibliothèques possèdent d'importantes dépendances de versions entre elles et notamment avec CUDA[8]. Les machines auxquelles nous avons accès ayant une version de CUDA différente de celles des auteurs, nous avons donc dû résoudre les problèmes de compatibilités afin de pouvoir l'utiliser.

Après quoi nous avons obtenu nos premières images de résultat :

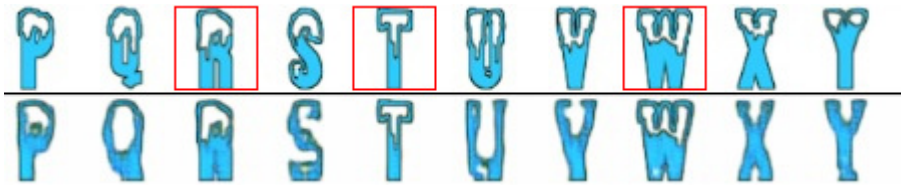

FIGURE 15 – Reproduction des résultats de MC-GAN

Sur cette image (figure 15) la ligne du haut correspond aux caractères de la fonte d'origine, avec encadrés en rouge, les caractères donnés au réseau pour apprentissage, et la ligne du bas est le résultat de la génération par le réseau. On voit que le réseau a bien réussi à générer les lettres manquantes en gardant le style des caractères qu'on lui a donné.

8.2 Application des réseaux sur des fontes extraites de DocCreator :

Nous avons ensuite voulu appliquer ces résultats sur les fontes extraites de DocCreator (par le programme fourni par les clients). Pour cela nous avons implémenté des scripts capables de transformer le résultat de l'extraction en des images compréhensibles et utilisables par le réseau (les fontes dans DocCreator sont stockées sous forme de fichier `.of` au format XML et nous les extrayons en plusieurs images PNG, une par caractères).

[8] technologie de NVIDIA permettant d'utiliser le processeur graphique (GPU) pour exécuter des calculs en parallèle à la place du processeur (CPU)

FIGURE 16 – Application de MC-GAN sur une fonte de DocCreator

Sur la figure 16, la ligne du haut représente les caractères de la fonte d'origine, et la ligne du bas est générée à partir des caractères encadrés en rouge. Le réseau MC-GAN tel qu'il est disponible sur le GitHub produit des résultats très convaincants même sur des fontes anciennes (les caractères générés peuvent être confondus avec des caractères extraits de documents anciens). Cependant à ce stade il ne nous était possible de générer que des lettres majuscules.

8.3 Ré-entraînement de MC-GAN :

Afin de pouvoir utiliser MC-GAN pour générer des caractères autres que majuscules (notre réseau doit aussi pouvoir créer des minuscules, des ponctuations, des chiffres et des caractères accentués) nous avons dû ré-entraîner le réseau. Cependant, afin d'avoir un entraînement assez efficace pour générer des caractères satisfaisants nous avons eu besoin de beaucoup d'images.

Nous avons donc cherché une base de données de fontes (`.ttf` et `.otf`) en contenant au moins 10 000, car c'est le nombre de fontes utilisées par l'auteur de MC-GAN. Bien que nous n'ayons pas trouvé de telle base de données nous sommes tombé sur un article utilisant un ensemble 50 000 images contenant des caractères majuscules, minuscules et des chiffres. Afin d'effectuer un premier test nous avons entraîné le réseau avec 10 000 images extraites de cette base de donnée afin de compléter des minuscules.

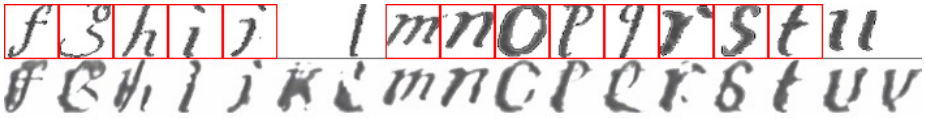

FIGURE 17 – Génération de minuscule par le MC-GAN ré-entraîner

On remarque que les caractères générés (figure 17) ne sont pas très lisibles et que le réseau a généré des caractères qui étaient connus à l'origine. En examinant les images de résultat nous avons conclu que notre base de données d'entraînement n'était pas assez triée, en effet certaines fontes utilisées dans la création des images que nous avons trouvées ne possédaient pas de caractères minuscule et ils étaient remplacés par des majuscules (ou encore des fontes ne contenant que des logos). C'est pourquoi le réseau génère mal certaines lettres (par exemple, on peut voir que le 'g' créé ressemble plus à un 'G' majuscule qu'à un minuscule).

Suite à ce test nous avons décidé de créer notre propre base de données de fichier de fontes, pour ce faire nous avons créé un script se rendant sur un site hébergeant des fichiers `.otf` et `.ttf` et qui va télécharger tous ceux qui nous intéressent (nous avons choisi dafont [9] qui permet de filtrer les polices et de ne garder que celles possédant des accents). Nous avons ensuite filtré ces fichiers afin de supprimer ceux ne possédant pas tous les caractères que nous voulons pouvoir gérer. À ces fichiers ont été rajoutées les polices provenant de Google[10] qui avaient pour avantage d'être très facilement filtrables par caractère.

A la fin de cette étape nous possédions environ 4200 fontes qui nous ont permis de créer le même nombre d'images afin d'entraîner notre réseau et d'obtenir nos derniers résultats présentés dans la section suivante.

[9] `https://www.dafont.com/fr/`
[10] `https://fonts.google.com/`

9 Résultats

FIGURE 18 – ensemble des caractères que nous pouvons générer sur la fonte Jules Vernes de DocCreator

Pour chaque bloc, la ligne du haut correspond aux caractères d'origines de DocCreator, les caractères encadrés en rouge sont ceux qui ont été fournis au réseau (le réseau n'ayant pas besoin de tous les caractères disponibles pour générer des images satisfaisantes, nous lui avons fourni ceux de meilleures qualités) et la ligne du bas correspondant à l'image générée par le réseau. DocCreator ne disposant pas de caractères accentués majuscules, la dernière ligne n'est composée que de la génération.

A l'heure actuelle, les résultats de la génération de 114 caractères sont moins satisfaisants que ceux sur les majuscules obtenus avec le modèle pré-entraîné disponible sur le GitHub. Néanmoins, en réalisant un entraînement du réseau plus long sur les machines du LaBRI (600 *epochs* à la place de 200) nous pensons obtenir des résultats bien plus satisfaisants.

Le deuxième problème de nos résultats est la gestion de la *baseline*, la *baseline* ou ligne de base en français, est la ligne sur laquelle la plupart des lettres reposent et au-dessous de laquelle s'étendent les jambages. On peut par exemple voir que le 'p' minuscule re-généré ressemble plus à une majuscule, ce problème est lié au fait que notre génération ne prend pas correctement en compte la *baseline*, la jambe du 'p' ne descend donc pas en dessous.

Enfin en raison d'une limitation liée aux cartes graphique disponibles, les images que nous avons générées mesurent 48 pixels de haut. Cependant, les cartes graphique disponibles au LaBRI disposant de plus de mémoire, il devrait être possible de générer des images de 64 pixels de haut (comme nos premières images) et donc d'améliorer la qualité.

10 Tests

Afin de tester le bon fonctionnement de notre projet, nous avons réalisé deux catégories de tests, les tests unitaires sur nos scripts de traitements de données et des tests de génération d'images par le réseau.

10.1 Tests unitaires

Nos tests unitaires ont été réalisés sous la forme de script bash, un pour chacun de nos traitements. Ces scripts lancent plusieurs fois chaque programme avec différents arguments, le but est d'être sûr que les arguments donnés aux programmes sont bien vérifiés au début de ceux-ci afin que leurs comportements soient bien ceux attendus.

```
(pfe_env) razen@ThinkPad:~/Documents/MC-GAN/test$ ./test_chars2png.sh
--- Test chars2png ---
Test 1: Error: number of argument != 2 (1)
python chars2png.py [dir with png file]


Test 2: Error: number of argument != 2 (3)
python chars2png.py [dir with png file]


Test 3: Error: image dir is not a directory
Need at least 2 png file in the image directory
python chars2png.py [dir with png file]


Test 4: Error: dataset name can't contain '_'
The dataset name can't contain '_'
python chars2png.py [dir with png file]


Test 5: Error: not enough png file in image dir
Need at least 2 png file in the image directory
python chars2png.py [dir with png file]


Test 6: Good: (without / at the end of dir)
UISPYAG


Test 7: Good, check the result image
UISPYAG
```

FIGURE 19 – Résultat de l'exécution d'un des scripts sur le terminal

La figure 19 montre le résultat d'un de ces scripts. On peut voir que le programme a été lancé 7 fois d'affilé, le résultat attendu (erreur avec le message d'erreur ou le bon fonctionnement) est indiqué à chaque exécution, suivi du retour du programme.

Ces scripts sont accompagnés d'un fichier PDF décrivant :

- l'objectif du test
- comment exécuter le script
- le résultat attendu sur le terminal
- le résultat attendu du programme (image ou architecture)

10.2 Test de génération d'images

Nous avons aussi effectué des tests sur la génération d'image par le réseau GAN afin d'étudier l'impact des caractères connus sur la génération des autres.

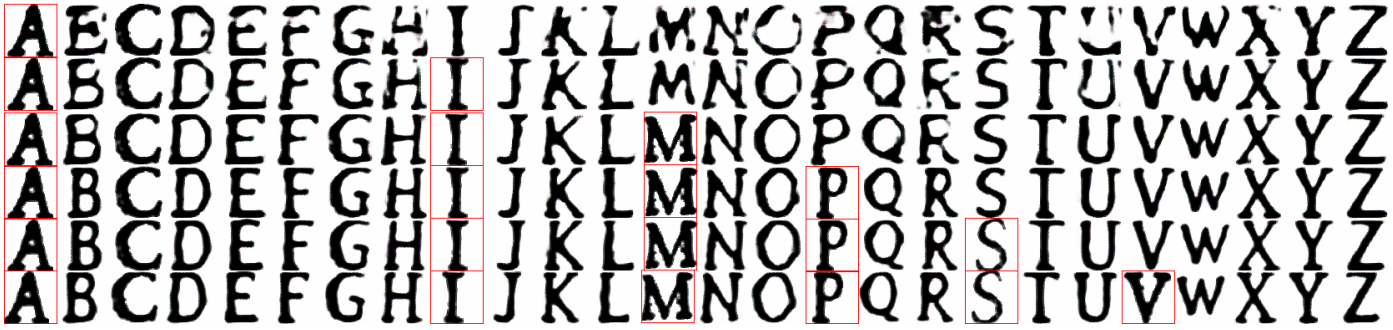

FIGURE 20 – Génération des lettres majuscules avec un nombre différent de lettres connues

La figure 20 montre le résultat d'un test sur l'impact du nombre de caractères connus lors de la complétion d'une fonte. Les lettres encadrées en rouge sont celles qui ont été utilisées pour réaliser l'apprentissage du réseau. À chaque génération (de bas en haut) on rajoute une lettre aux précédentes et on compare les résultats obtenus.

Ce test a permis de montrer que le réseau a besoin de peu de lettres pour générer des images de qualités satisfaisantes. En effet bien que les trois premières lignes possèdent des caractères pas complètement formés ou peu lisibles, dès la quatrième ligne avec cinq caractères connus, les résultats sont très satisfaisants. Le temps de génération d'une image augmentant avec le nombre de caractères utilisés lors de l'entraînement, nous en avons aussi déduit qu'il n'était pas nécessaire d'utiliser un grand nombre de caractères à chaque fois mais de filtrer seulement ceux qui ont été les mieux extraits des documents anciens.

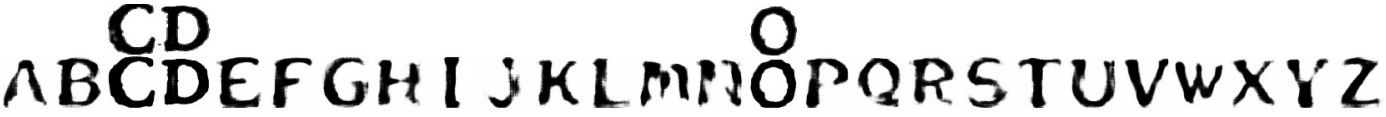

FIGURE 21 – Test de génération de lettres majuscules à partir de caractères arrondis

FIGURE 22 – Test de génération de lettres majuscules à partir de caractères droits

Les figures 21 et 22 montrent les images de résultat du test sur l'impact de la forme des lettres connues sur la génération des autres. Ainsi la figure 21 correspond à une génération à partir de caractères arrondis (C, D et O) et la figure 22 à partir de caractères droits (E, H et L).

La conclusion de ce test est que le réseau à plus de mal a générer des résultats satisfaisants si on lui donne seulement des caractères arrondis (on peut le voir avec le M et le N de la figure 21).

11 Conclusion et Perspectives

Nous avons réussi à générer la plupart des caractères manquants des fontes de DocCreator. Pour ce faire, nous avons dû adapter le réseau existant MC-GAN et créer notre propre ensemble de données d'entraînement. Les ordinateurs auxquels nous avions accès étant équipés de carte graphique GTX 1070 avec 8Gb de mémoire et étant difficilement accessibles pendant plus de 20 heures, nous avons dû nous restreindre à entraîner notre réseau sur seulement 200 *epochs* quand l'auteur du MC-GAN l'entraînait sur 600 *epochs*. Cependant, nos clients ont accès à un meilleur processeur graphique (2080Ti 11Gb). Il leur sera donc possible d'entraîner notre réseau d'avantage et d'obtenir par la suite de meilleurs résultats que ceux que nous produisons.

Si nous avions eu plus de temps pour faire le projet nous aurions aimé rajouter deux grandes fonctionnalités. Dans un premier temps l'intégration de notre complétion de fontes directement dans le logiciel DocCreator. Avec les dernières version d'openCV il est possible de charger un modèle de réseau déjà entraîné et de l'utiliser directement. Cependant certaines couches du réseau MC-GAN ne sont pas encore supportées par OpenCV et nous n'avons donc pas réussi à charger le réseau.

Enfin, maintenant qu'il nous est possible de compléter les fontes de DocCreator, nous aimerions appliquer le réseau de zi2zi afin de créer de nouvelles fontes. Il nous faudrait donc comprendre comment le ré-entraîner correctement sur des caractères latins.

Références

[1] Nicholas Journet, Muriel Visani, Boris Mansencal, Kieu Van-Cuong, and Antoine Billy. Doccreator : A new software for creating synthetic ground-truthed document images. *Journal of imaging*, 3(4) :62, 2017.

[2] Samaneh Azadi, Matthew Fisher, Vladimir G. Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. Multi-content gan for few-shot font style transfer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[4] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv :1710.10196*, 2017.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv :1511.06434*, 2015.

[6] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[7] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[8] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2794–2802, 2017.

[9] zi2zi. `https://kaonashi-tyc.github.io/2017/04/06/zi2zi.html`. Accessed : Mars 2019.

A Reproduction des résultats de zi2zi

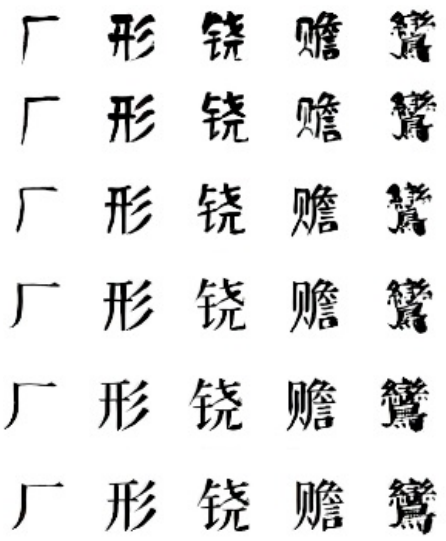

FIGURE 23 – Reproduction des résultat de zi2zi

Sur cette image (figure 23) la première et la dernière ligne correspondent à deux fontes différentes, les quatre lignes intermédiaires sont les étapes de l'interpolation générée par le réseau. On peut voir qu'en lui donnant une fonte très droite et une fonte plus arrondie il génère des caractères possédant un style entre les deux.

Nous avons ensuite voulu appliquer le réseau de zi2zi sur des caractères latins extraits de DocCreator mais nous n'avons pas réussi. Le réseau a été entraîné avec des caractères asiatiques et aucune explication n'était fournie sur le moyen de le ré-entraîner avec un nouveau dataset. C'est donc à ce moment-là que nous avons décidé de nous concentrer sur la complétion de fontes avec MC-GAN.