UNIVERSITÉ DU QUÉBEC À MONTRÉAL

RÉSEAU GÉNÉRATIF ANTAGONISTE POUR LA TRADUCTION DE SIGNAUX DE CAPTEURS AVEC APPLICATION À L'ECG

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

MOHAMED AMINE ABDELMADJID

JUILLET 2022

REMERCIEMENTS

Dans un premier lieu je tiens à adresser mes remerciements les plus sincères à monsieur Mounir Boukadoum, qui en tant que directeur de recherche, s'est toujours montré présent et à l'écoute, et ceci durant tous mon parcours de maîtrise, sans oublier les nombrables et judicieux conseils qui m'ont permis de m'améliorer et donner le meilleur de moi-même.

Je souhaite aussi adresser mes remerciements au corps enseignant et administratif de l'Université du Québec à Montréal pour la richesse et la qualité de leur enseignement, ainsi que les efforts déployés pour nous permettre de finaliser notre cursus malgré la crise difficile due au COVID.

Je remercie aussi la compagnie SigNum de Montréal de m'avoir fourni les enregistrements de signaux utilisés dans ce mémoire.

Enfin, je suis profondément reconnaissant à ma famille qui m'a toujours soutenu dans mes projets, ainsi que mes amis qui m'ont épaulé tout au long de mon cheminement

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

RÉSUMÉ

Ce travail décrit la conception d'un système d'apprentissage automatique pour la traduction de signaux chronologiques d'un format de représentation vers un autre, avec l'électrocardiogramme comme objectif d'application. Il est motivé par le fait que les signaux générés par les capteurs modernes, notamment les capteurs biomédicaux, peuvent différer des formats standards. Le problème de la transduction de séquences a été abordé durant les dernières décennies de diverses façons, dont les approches génératives et discriminantes. Cependant, les efforts des chercheurs ont porté principalement sur les séquences rencontrées lors du traitement automatique du langage naturel et les modèles discrets proposés sont adaptés aux données linguistiques. Il existe peu de travaux qui ont abordé la transduction d'autres types de données, dont les signaux chronologiques en provenance de capteurs biomédicaux. Ce travail présente un modèle de réseaux antagonistes génératifs (GAN en anglais) pour la transduction des signaux d'un réseau de capteurs de l'activité cardiaque chez l'humain. Il transforme les mesures brutes fournies par un tapis d'électrodes sans contact avec la peau en des signaux d'électrocardiogramme standard (appelé communément ECG ou EKG). Pour ce faire, le modèle CycleGAN, normalement employé pour la génération d'images synthétiques, est adapté pour l'application à l'ECG. Plusieurs itérations ont été nécessaires pour arriver au résultat. Dans un premier temps, un système de génération de symboles est étudié et validé sur la base MNIST, une base de données populaire de caractères manuscrits. Ensuite, un système de conversion de fonction en une autre fonction est étudié. Finalement, un système de traduction à canaux multiples est développé pour l'ECG. Ce modèle est évalué à l'aide de 31 enregistrements de l'activité cardiaque fournis par la compagnie SigNum de Montréal, effectués en parallèle avec leur tapis d'électrodes sans contact et des électrodes standards. Le calcul de l'erreur quadratique moyenne entre les signaux cibles et ceux prédits montre que la traduction de signaux chronologiques est possible en utilisant les réseaux CycleGAN, et que le modèle développé donne une meilleure précision que des modèles d'apprentissages classiques.

Mots-clés : Traduction de signaux, Électrocardiogramme, ECG, EKG, réseaux antagonistes génératifs, GAN, CycleGAN.

INTRODUCTION

Les prouesses récentes des réseaux de neurones artificiels (RNA) pour la solution des problèmes les ont rendus populaires dans un grand nombre de domaines et d'applications, allant de tâches simples à des tâches jugées trop compliquées à résoudre il y a quelques années. Ainsi, la place de l'intelligence artificielle dans un large spectre d'applications n'est plus discutable, notamment grâce à l'apprentissage profond. Ce dernier a montré son efficacité en offrant souvent une précision qui surpasse ce qu'on obtient avec les méthodes classiques (Mahapatra, 2018). On note aussi que cet outil ne cesse d'être amélioré à travers les années, avec un spectre d'applications de plus en plus vaste.

Parmi les nombreux champs d'application de l'apprentissage profond, on cite souvent la vision par ordinateur, un secteur où l'apprentissage profond permet la détection d'objets (Dasiopoulou *et al.*, 2005), le traitement et l'analyse d'images (Tyagi, 2018) et bien d'autres tâches qui facilitent l'extraction d'information visuelle ou sa classification.

Un autre domaine qui suscite l'intérêt est le traitement du langage naturel (TLN) ou du texte, une branche de l'intelligence artificielle qui est en constante progression. L'apprentissage profond y est utilisé pour la création de dialogues artificiels, la traduction de séquences et la détection d'émotions pour ne citer que ces applications. Les découvertes dans ce domaine alimentent aussi de nombreuses disciplines de recherche toutes aussi importantes les unes que les autres.

Les problématiques liées aux traductions de séquences, comme les phrases du langage naturel, existent depuis plusieurs décennies. Le milieu du XXe siècle a vu

naître les premières machines de traitement de texte. Cela a commencé par la création de dialogues à domaine fermé en utilisant des bases de réponses préétablies, jusqu'à réussir plus tard des tâches à domaine ouvert plus évoluées comme la traduction de texte (Poibeau, 2017). Parmi les approches récentes, on peut citer des modèles comme Seq2Seq [1], ou encore GPT [2] qui ont montré une efficacité et une capacité impressionnantes en traitement automatique du langage.

Cependant, la traduction de séquence ne se limite pas au langage naturel, et les dernières années ont montré un spectre d'application plus large. Par exemple, le traitement de séquence peut être utilisé dans la transformation de spectres sonores et bien à d'autres types donnés avec des structures complexes (Yu et Deng, 2010).

L'apprentissage profond offre un coffre à outils dans lequel on trouve des modèles pour résoudre des problématiques diverses. Parmi ses méthodes figurent les modèles génératifs qui suscitent un engouement grandissant. Ces modèles se distinguent de leurs compétiteurs, les modèles discriminants, par le fait qu'ils essayent de découvrir la distribution statistique des données au lieu de comparer leurs vraisemblances d'appartenir à une classe parmi d'autres. Une fois la distribution apprise, ils peuvent générer automatiquement des données qui la suivent. Les modèles génératifs ont déjà montré des résultats impressionnants dans plusieurs applications, que ce soit la transformation de la voix dans des enregistrements audios, ou bien dans la création de pastiches de peintures (Yu et Deng, 2010).

Pour notre projet, nous avons été attirés par la performance des modèles généra-

---

1. Seq2Seq est un modèle de transformation de séquence, basé sur un réseau récurent dont les états antérieurs servent de contexte de prédiction (Sutskever *et al.*, 2014)

2. Generative Pre-trained Transformer (GPT) est un modèle de transformation de séquence qui repose sur la notion d'attention. La version GPT-3 comprend 175 milliards de paramètres à régler (Heaven, 2020)

tifs à créer des motifs synthétiques, notamment les réseaux antagonistes génératifs (GAN en anglais) qui ont été utilisés pour résoudre des problèmes liés à la traduction de séquences linguistiques (Rashid *et al.*, 2019). Nous avons donc voulu utiliser un modèle génératif de type GAN comme outil de traduction d'un autre type de séquence, les signaux chronologiques que l'on rencontre fréquemment dans divers domaines scientifiques et techniques.

Problématique et motivation :

Comme déjà mentionné, le problème de la traduction de séquences n'est pas nouveau. On a déjà vu au début du XXe siècle des machines utilisées dans la traduction du langage humain, que ce soit d'une manière statistique ou mécanique, et les modèles basés sur l'apprentissage profond sont arrivés bien plus tard. Cependant, le langage humain est complexe, et certaines conceptions humaines demeurent ambiguës et difficiles à retranscrire et à implémenter dans une machine (Duarte *et al.*, 2018). On peut prendre en exemple la conservation du contexte lors d'une traduction ou bien la retranscription des émotions qui restent compliquées à établir sur une grande échelle. Souvent, cela est dû principalement à un manque de jeux de données de qualité pour l'apprentissage, à cause du coût important que peut avoir l'annotation ou le formatage de données brutes. (Zhou et Wang, 2017). Ce manque de qualité peut entraîner des biais lors de l'apprentissage, et donc des coûts supplémentaires pour avoir de bons résultats.(Saagie, 2020)

Le manque de jeux de données de qualité dans la traduction de séquences n'est pas limité au traitement du langage naturel. Dans ce mémoire, nous avons aussi rencontré ce problème pour traduire des signaux d'électrocardiogramme. Pour pallier ce problème, nous avons choisi une approche générative au problème de traduction, afin de prédire les réponses en se basant sur la distribution approximative

des données d'apprentissage et non pas leur jeu restreint de valeurs. À cet égard, le modèle CycleGAN offre une solution intéressante, en permettant d'établir une relation d'inférence mutuelle entre une séquence source et une séquence cible.

Des implémentations existent déjà, comme le CycleGAN-VC (Kaneko et Kameoka, 2018) qui permet de transformer les séquences audio, ou le modèle de base qui transforme un type d'image vers un autre (Zhu *et al.*, 2017). Les très bons résultats de ces implémentations sont encourageants et ont été des sources d'inspiration pour notre projet. Notre but est donc de pouvoir passer d'un signal A vers un signal B en utilisant un modèle génératif de type CycleGAN. Cependant, il est nécessaire d'expérimenter avec les modèles existants et de les adapter à notre situation pour atteindre l'objectif. Un des défis rencontrés lors du survol de l'état de l'art fut le manque de travaux sur la traduction de signaux. La solution a été de fusionner et modifier des implémentations réservées pour d'autres applications, et de les adapter à notre problématique.

Contribution :

L'objectif de notre projet est un modèle génératif de type CycleGAN capable de transformer la représentation temporelle d'un signal d'un format vers un autre. Plus spécifiquement, le modèle doit pouvoir traduire les signaux ECG fournis par une nouvelle méthode de récolte de données en signaux ECG standards. Afin de réaliser cet objectif, nous avons mis en place plusieurs prototypes de solutions, grâce à des modifications et fusions de modèles existants. Au final, nous pouvons identifier les contributions suivantes :

1. Création d'un modèle de réseaux antagonistes génératifs simple et adapté aux entrées considérées, donc qui prend en entrée un vecteur d'échantillons de signaux chronologiques.

2. Développement d'un modèle avec une grande précision de transduction de signal avec peu de données d'apprentissage.
3. Identification des obstacles pouvant dégrader la précision et la proposition de solutions pour les contourner dans des travaux futurs.

Structure du mémoire :

Dans un premier temps, nous introduisons l'apprentissage profond et les modèles de traduction de séquences existantes, plus spécifiquement les modèles de traitement du langage, pour ensuite expliquer pourquoi ils sont difficiles à adapter pour la transformation de signaux.

Dans un second temps, nous portons l'attention sur les réseaux antagonistes génératifs, en décrivant leurs différentes applications et leurs modes d'apprentissage ainsi que leurs structures et modes de fonctionnement. Nous introduisons aussi des variantes de GAN utilisées lors de la conception de notre projet.

Finalement, nous décrivons le modèle et les expériences menées pour traduire des signaux ECG acquis de deux manières différentes par la compagnie SigGNUM de Montréal. Nous identifions et discutons alors toutes les contraintes survenues lors du développement, et concluons avec un sommaire de nos résultats et les solutions ou améliorations futures à apporter au système.

CHAPITRE I

TRADUCTION ET TRANSFORMATION DE SÉQUENCES

1.1 Introduction à l'apprentissage profond

L'apprentissage profond se distingue des autres méthodes de l'apprentissage automatique par sa capacité à traiter des données brutes. C'est un concept élaboré qui permet à la machine d'apprendre des structures complexes à partir d'autres, plus simples (Goodfellow *et al.*, 2016). Son moteur principal est généralement un grand jeu de données, duquel l'apprentissage profond pourra déceler des informations ou en générer d'autres, et ceci avec ou sans supervision humaine. (Bengio, 2009)

L'apprentissage profond a connu ces dernières années une croissance impressionnante dans quasiment tous les domaines d'application, incluant le traitement de texte naturel, la vision d'ordinateur, ou la bio-informatique (Goodfellow *et al.*, 2016). Ces avancées ont aussi été transposées et implémentées dans des secteurs plus communs comme la médecine, l'économie ou même le multimédia.

Habituellement, quand on parle d'apprentissage profond, on pense aux architectures discriminantes classiques, comme les réseaux de neurones convolutifs ou les réseaux de neurones entièrement connectés, dont la forme est inspirée de la structure du cerveau humain. Leur objectif principal est de localiser des modèles dans la base de données grâce à des opérations mathématiques et probabilistes, puis

de généraliser et d'appliquer le processus appris à d'autres ensembles de données inconnus du réseau concerné. (Hansen et Salamon, 1990).

Le réseau de neurones artificiels est à la base de l'apprentissage profond et s'inspire de l'architecture cérébrale humaine. Les neurones qu'on y trouve forment des nœuds par lesquels transitent les données et les calculs, et qui sont reliés entre eux par des synapses pondérées (McCullum, 2020). Ces derniers sont appliqués aux entrées envoyées au neurone suivant, qui à son tour applique une fonction d'activation et transmet le tout au neurone d'après. (figure 1.1)

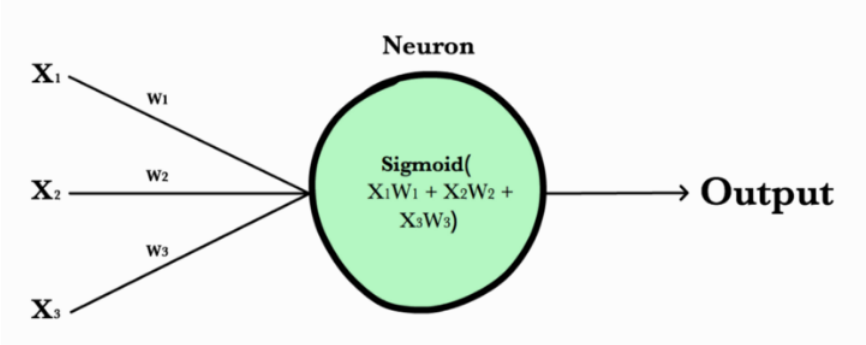

Figure 1.1: Structure d'un neurone

La fonction d'activation permet la transformation non linéaire des données lors de leur passage d'un neurone vers un autre, permettant ainsi d'établir des relations complexes entre les entrées.

Le réseau de neurones, dans sa forme générale, est constitué de plusieurs couches de neurones qui se succèdent toutes connectées grâce à des synapses. Le réseau commence par une couche d'entrée «Input layer» qui représente une unité distincte du jeu de données. Elle est suivie par une ou plusieurs couches cachées ou s'effectue l'application des poids et des fonctions d'activation (Pang *et al.*, 2020). La structure du réseau se termine par une couche de sortie qui renvoie le résultat final. Cette transformation non linéaire qui s'effectue d'une couche à une autre

s'appelle la propagation vers l'avant (Goodfellow *et al.*, 2016).

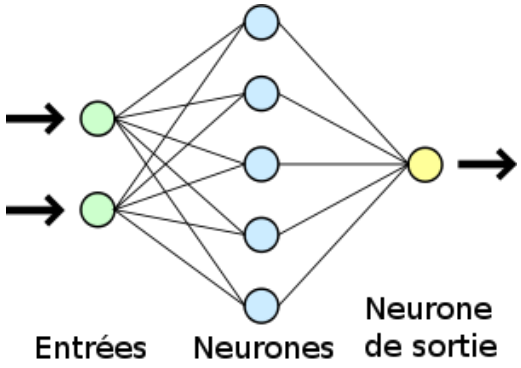

Figure 1.2: Exemple d'un réseau de neurones

Pour leur entraînement, les réseaux de neurones à propagation directe nécessitent aussi une propagation vers l'arrière qui prend en considération les résultats obtenus de la couche de sortie (valeur prédite) et la valeur visée fournie avec le jeu de base. Une fonction de perte prend alors en charge ces deux valeurs pour pouvoir mettre à jour les poids du réseau grâce à un algorithme d'optimisation comme la descente de gradient [1] (Ruder, 2016). Celui-ci vise à minimiser cette perte et par la même occasion à avoir de meilleurs résultats. (Goodfellow *et al.*, 2016).

La phase d'apprentissage des réseaux de neurones nécessite une puissance et un temps de calcul qui peuvent être énormes. Ceci est principalement dû au nombre important d'opérations mathématiques effectuées par le réseau. Une solution pour accélérer l'apprentissage est d'effectuer plusieurs calculs de façon parallèle à l'aide d'un GPU (Krizhevsky *et al.*, 2017). L'utilisation des GPUs a permis d'accélérer considérablement la phase d'apprentissage. rendant ainsi l'utilisation des réseaux

1. Cette métaheuristique est couramment utilisée pour déterminer le minimum global de la fonction de perte, par incréments ou décréments progressifs de signes opposés à la pente (gradient) des valeurs de la fonction de perte pour chaque valeur de (θ)

de neurones raisonnable en temps et coût de calcul.

Parmi les architectures neuronales proposées au fil des ans, les plus populaires incluent les réseaux de neurones convolutifs (CNN) qui ont fait leurs preuves en classification d'images, grâce entre autres à une structure flexible qui facilite l'ajustement des paramètres, dont la profondeur et la fonction de chaque couche (Kalchbrenner *et al.*, 2014). Les CNNs incluent à la fois des couches de convolution, des blocs de compression «pooling» et d'une couche finale de type perceptron. Ces couches ont pour but de prétraiter des données complexes en les redimensionnant et en les fragmentant, ce qui permet d'augmenter l'efficacité et la vitesse de traitement de ces réseaux (Ciregan *et al.*, 2012).

D'autres architectures puissantes sont les réseaux de neurones récurrents (RNN) qui sont populaires pour le traitement de séquences. À la distinction des modèles à propagation directe, les modèles récurrents possèdent des boucles internes qui leur permettent de mémoriser et de réutiliser une entrée reçue antérieurement dans la prédiction de sorties futures (colah, 2015). Cette structure permet une plus grande efficacité dans le domaine de la traduction, la création de texte, composition de musique et bien d'autres ou le contexte est important pour prédire la sortie (Hochreiter et Schmidhuber, 1997)

Cependant, un problème se pose lors de l'entraînement des réseaux récurrents pour les longs contextes. On rencontre alors des difficultés d'apprentissage, tel que des résultats incorrects, une disparition du gradient et un temps de calcul conséquent (Hochreiter et Schmidhuber, 1997). Pour régler cette problématique, le réseau LSTM «Long Short-Term Memory» a vu le jour. Ce modèle peut garder en mémoire des informations à plus long terme que le modèle RNN classique.

Les LSTM utilisent des portails, ou «gates», qui permettent de réguler le flux de données en ignorant certaines informations et en gardant uniquement les données

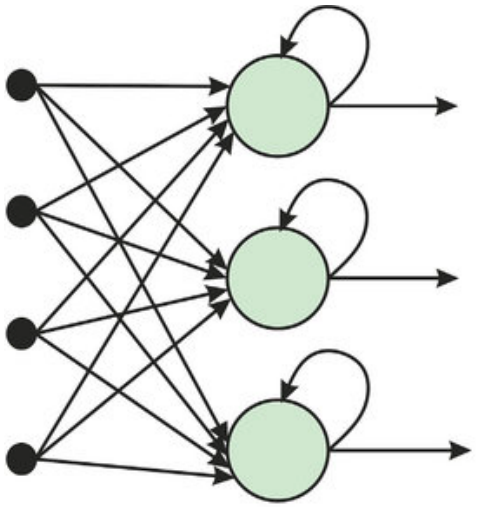

Figure 1.3: Exemple d'un réseau de neurones récurrents

pertinentes à la prédiction. Leur structure comprend des cellules qui implémentent les différents portails requis. Chaque cellule reçoit un état caché d'une étape précédente h_i, elle applique ensuite un portail d'oubli grâce à une fonction sigmoïde sur l'information actuelle et précédente $[h_i + x_i]$ pour garder ou retirer les informations non pertinentes ; ensuite une fonction tangente hyperbolique redimensionne le vecteur reçu dans une intervalle [-1,1] ce qui évite une explosion du gradient ou sa disparition 1.5 Finalement, les informations obtenues constituent l'état caché ou la mémoire actuelle, mais elles sont aussi utilisées pour trouver la sortie de la cellule suivante (colah, 2015).

Grâce à leurs propriétés de mémoire, les réseaux de neurones récurrents sont très populaires dans le domaine de la transformation et la traduction de séquences, et ils jouent un rôle important dans les techniques de traduction du langage moderne comme la traduction automatique neuronale « neural machine translation» ou NMT.

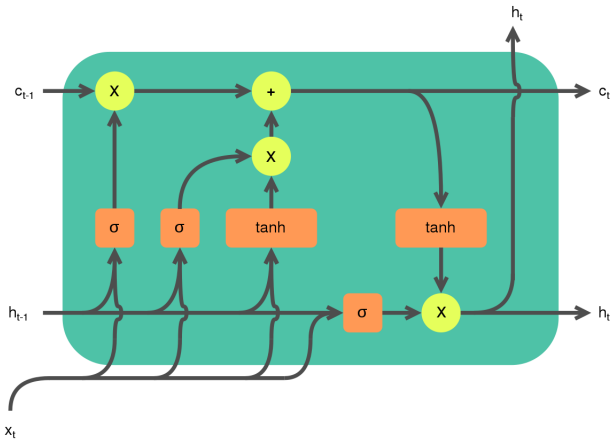

Figure 1.4: Exemple d'architecture LSTM (Shastri, 2020)

1.2 La traduction automatique

La traduction automatique peut être résumée en «la traduction d'une séquence de symboles vers une autre séquence de symboles». On essaye typiquement de traduire une phrase d'une langue vers une autre. Cependant, la complexité du langage humain rend la tâche difficile. En effet, il existe une ambiguïté et une conception humaine difficile à reproduire ou à transcrire dans une machine, faisant que le secteur de la traduction automatique du langage a présenté un grand défi aux chercheurs pendant des années. Ainsi, de nombreuses recherches ont été menées pour trouver une méthode de traduction précise et efficace, mais le manque de corpus d'entraînement et le manque de ressources et de puissance de calcul ont freiné pendant longtemps l'avancement de ce domaine. Néanmoins, on trouvait des approches basées sur des règles conçues par des linguistes, et il existait aussi des modèles statistiques utilisés par les grandes compagnies. On peut citer Google qui a longtemps utilisé un modèle statistique pour la traduction automatique dans sa plateforme Google traduction, avant de le remplacer par un modèle neuronal plus performant.

1.2.1 Traduction automatique statistique (SMT)

Avec l'évolution des technologies et du matériel de calcul, des méthodes statistiques conçues au milieu du XXe siècle ont commencé à refaire surface. L'utilisation des statistiques pour la traduction de texte est devenue une idée réalisable. Cette méthode était révolutionnaire, car elle avait juste besoin d'un corpus de donnée composé de la langue qu'on veut traduire et la langue qu'on veut obtenir, à l'encontre des méthodes qui existaient et qui utilisaient des règles prédéfinies par des linguistes, mais qui présentaient des problématiques telles que le coût et la difficulté de maintenance. (Brown *et al.*, 1990)

La traduction automatique statistique est devenue une référence dans le domaine de la traduction de langue, Google d'ailleurs a adopté cette technique pour ces modèles de traduction à l'époque (Och, 2006). La traduction automatique statistique cherche à maximiser la probabilité d'avoir une phrase visée (T) en ayant une phrase initiale (S), en d'autres mots le modèle cherche à maximiser la probabilité $argmax_y P(y|x)$ et, grâce au théorème de Bayes, on essaye alors de chercher le maximum de $argmax_y P(X|y) \times P(y)$. Comme évoqué plutôt, toutes ces probabilités sont apprises à partir d'un jeu de données constitué de séquences d'une langue et de son parallèle dans l'autre langue. (Brown *et al.*, 1990)

La traduction d'une langue vers une autre représente quand même un défi. Certains mots, par exemple n'ont pas une équivalence directe dans une autre langue, mais ils sont plutôt décrits par des phrases. Cependant, en prenant en considération l'alignement dans le calcul de probabilité on résout ce problème. On calcule alors $P(y, a|x)$, ainsi nous prenons en considération la probabilité qu'un mot soit dans une position précise, mais aussi qu'un mot ait une phrase qui le décrit. (Brown *et al.*, 1990)

Calculer la probabilité de toutes les hypothèses possibles reste très coûteux. Par conséquent, on applique une recherche heuristique, c'est-à-dire le modèle élimine tous les alignements avec une faible probabilité. Par exemple on émet une multitude d'hypothèses pour chaque mot de la phrase qu'on veut traduire, ensuite on constitue une sorte d'arbre pour les hypothèses avec les plus grosses probabilités et on obtient notre traduction. Cette méthode s'appelle le décodage. (Brown *et al.*, 1990)

Pendant longtemps, la traduction automatique statistique dominait, malgré cela ce modèle était compliqué à maintenir et nécessitait des coûts énormes pour les architectures les plus avancées. L'avènement des réseaux de neurones a éclipsé la structure statistique durant la dernière décennie.

1.2.2 Traduction automatique neuronale (NMT)

Ces dernières années, le domaine de la traduction automatique du langage n'a cessé de se développer. Des centaines d'articles et d'études scientifiques ont été menés pour trouver la meilleure méthode de traitement de texte. Dans le même temps, le domaine de l'apprentissage profond a également connu une croissance explosive, ce qui a donné naissance à des approches neuronales pour la traduction automatique de textes. Les nouvelles structures neuronales montrent une efficacité significative et de meilleures performances, éclipsant ainsi les modèles statistiques (Poliakow, 2017).

La traduction automatique neuronale est aussi une machine de langage conditionnelle, car elle cherche à maximiser la probabilité d'avoir une phrase (y) en partant d'une phrase (X) (Bahdanau *et al.*, 2014). En d'autres mots, l'entrée initiale joue un rôle très important dans la prédiction de la sortie. Comme on peut le voir dans l'équation (1.1), pour trouver la probabilité du prochain mot, nous la calculons en

prenant en considération les mots précédents. Cette méthode facilite grandement l'apprentissage par rapport au SMT.

$$P(y) = \prod_{t=1}^{T} p(y_t | y_1, \cdots, y_{t-1}, x) \tag{1.1}$$

Une approche populaire en NMT est le modèle encodeur-décodeur qui combine deux réseaux en séquence 1.5. Le premier, appelé encodeur, qui crée une représentation interne de la phrase source, et le deuxième, appelé décodeur, traduit la représentation créée en une phrase dans la langue cible. L'encodeur et le décodeur sont souvent des réseaux de neurones récurrents.

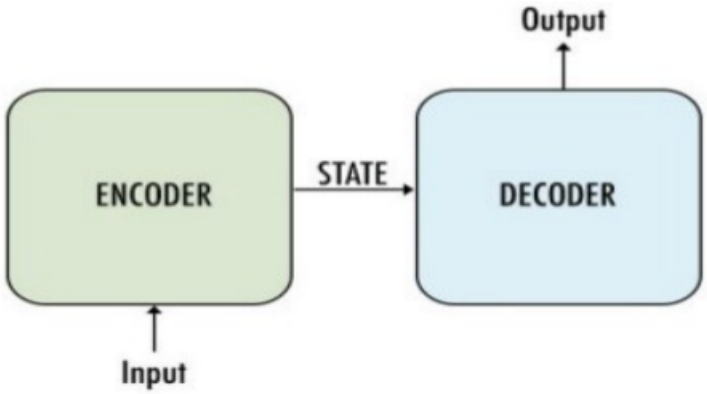

Figure 1.5: Architecture encodeur-décodeur (Bermudez Romero, 2021)

Lors de l'apprentissage du modèle encodeur-décodeur, deux jeux de données sont nécessaires. Le premier jeu concerne la langue source et est envoyé à l'encodeur, et le deuxième contient des phrases de la langue cible et est envoyé au décodeur. L'encodeur transforme une phrase du premier jeu en un vecteur de taille fixe appelé «vecteur de contexte» qui aide le décodeur à générer la traduction la plus probable. La fonction de perte J lors de l'apprentissage peut être calculée en additionnant l'entropie croisée de chaque jeton constituant notre phrase prédite $\hat{y}$ et celui qui lui correspond dans la phrase cible y :

$$j = \frac{1}{T} \sum_{t=1}^{T} j_t \tag{1.2}$$

La fonction de perte est minimisée par descente de gradient et est propagée dans le réseau d'un bout à un autre afin de régler les poids (Yang *et al.*, 2020). L'encodeur et le décodeur sont entraînés au même moment.

D'un point de vue pratique, le décodeur nous permet de traduire nos phrases sources, pour lesquelles il existe plusieurs méthodes de décodage. La recherche la plus élémentaire consiste à trouver des mots dans la langue cible à chaque étape de la traduction, avec la plus grande probabilité de correspondance. Il existe certaines limites à cette méthode, notamment l'impossibilité d'annuler la décision qui serait problématique lorsque le mauvais mot a été sélectionné dans la prédiction (Guillaume Genthial, 2019).

1.3 La traduction de séquences chronologiques

L'efficacité des réseaux de neurones pour la traduction automatique n'est plus discutable, au point que des instituts gouvernementaux et des géants de l'industrie informatique l'utilisent pour la traduction de texte. Que ce soit les LSTM, les réseaux à base d'attention, ou les modèles de traduction automatique neuronale en général, ils ont montré une puissance et une performance significatives. Cependant, ces modèles ont été conçus pour résoudre des problèmes liés à la langue, et non à des problèmes de traduction de signaux. Malgré l'adaptabilité de certaines de ces structures, on constate très peu de références ou de recherches qui traitent la traduction de séquences chronologiques avec des architectures neuronales.Il est aussi important de noter que les modèles cités précédemment montrent des faiblesses quand on à affaire â un petit jeu de données, ce qui est souvent le cas en sciences et en génie, dont le sujet de ce mémoire. Pour ce type de problème,

nous avons choisi une approche récente et novatrice, les modèles génératifs antagonistes, avec l'hypothèse qu'un des modèles peut être adapté nous donner de bons résultats de traduction de signaux chronologiques en présence de peu de données d'entraînement.

CHAPITRE II

RÉSEAUX ANTAGONISTES GÉNÉRATIFS

Les réseaux génératifs antagonistes, aussi appelés GANs, ont été introduits par Ian Goodfellow en 2014. Ils ont été conçus pour répondre à des contraintes rencontrées par les anciens modèles génératifs comme l'incapacité de traiter certaines distributions probabilistes (Goodfellow *et al.*, 2014). Depuis, ils ne cessent de gagner en popularité, grâce à des performances et résultats impressionnants. (Figure 2.1)

Figure 2.1: Image générée par le modèle hyper-réalistique GANs d'NVidia

Ce chapitre introduit les réseaux génératifs antagonistes en expliquant la façon par laquelle ils arrivent à apprendre et à reproduire une distribution, mais aussi en mettant en évidence les différentes problématiques qu'on peut rencontrer lors de l'implémentation. Il donne aussi des exemples d'implémentations tirés de l'état de l'art et les différentes variantes de GANs

2.1 Intuition des GANs

Le réseau GAN est l'un des modèles génératifs les plus intéressants de notre époque. La mise en œuvre de ces réseaux est plus simple et moins coûteuse que celles d'autres modèles génératifs comme l'autoencodeur variationnel et la machine de Boltzmann. De plus, Ils fournissent souvent des résultats qui dépassent les architectures génératives classiques.(Goodfellow *et al.*, 2014). Les architectures GAN reposent sur deux réseaux de neurones antagonistes qui essayent chacun de prendre le dessus sur l'autre lors de l'apprentissage. Cette idée est inspirée du concept de jeu à somme nulle en théories des jeux. Dans le cas des GAN, l'approche implique aussi la victoire d'un des deux agents à la fin de l'apprentissage, dans le cas où l'équilibre théorique n'est pas atteint. (Ge *et al.*, 2018).

Les réseaux antagonistes génératifs sont donc composés d'un réseau génératif (G) et d'un réseau discriminant(D). Le premier a pour rôle de créer de nouvelles données à partir de valeurs aléatoires, et la fonction du deuxième est de différencier les nouvelles données de celles faisant partie d'une distribution de données de référence. Plusieurs possibilités existent : soit le générateur l'emporte et arrive avec succès à reproduire la distribution initiale, soit le discriminateur arrive à prendre le dessus et les résultats ne sont pas alors satisfaisants, ou soit aucun des deux ne l'emporte, poussant les deux à plus d'efforts pour gagner.

La métaphore du faussaire est souvent utilisée pour expliquer le fonctionnement du GAN. Le générateur est un faussaire qui apprend de ses erreurs en tentant de créer des pièces contrefaites pour tromper le discriminateur qui est un marchand. Chaque fois que le commerçant distingue le vrai du faux, le contrefacteur se reprend en améliorant sa technologie de génération, afin de tromper la vigilance du commerçant.

2.2 Apprentissage des GANs

En raison de la structure spéciale du réseau antagoniste génératif, il nécessite une configuration spécifique pour son apprentissage. Des paramètres comme les fonctions d'activation, les fonctions de perte et l'optimisation du réseau ont été revus et adaptés pour permettre le bon fonctionnement du modèle.

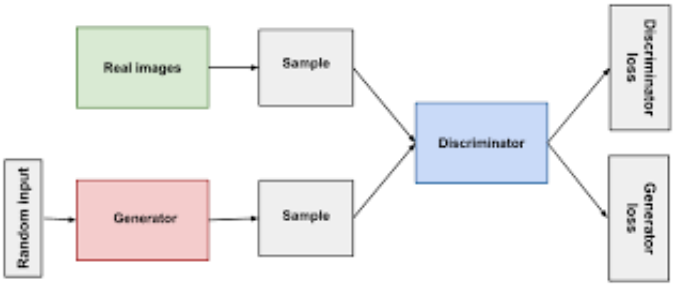

Figure 2.2: Architecture des GANs (Google Dev)

D'un point de vue pratique, le générateur G essaye de créer une distribution de données en appliquant une série de fonctions sur un vecteur ou scalaire composé de valeurs aléatoires Z. La sortie du générateur est ensuite envoyée au discriminateur qui prédit si la donnée reçue est vraie ou fausse. On optimise ensuite les poids du réseau grâce à une fonction de perte qui est optimisée par la méthode de descente du gradient, de façon à ce que le générateur apprenne à travers l'apprentissage et les décisions du réseau discriminant D (Goodfellow *et al.*, 2014). La Figure 2.2 donne un aperçu.

2.2.1 Fonction de perte des GANs

La fonction de perte est cruciale pour l'entraînement d'un GAN. Dans la dualité qui oppose les deux réseaux, le discriminateur essaye de maximiser la probabilité de faire la bonne prédiction en classifiant correctement les entrées réelles et

fausses ; de l'autre côté, le générateur essaye de minimiser la même probabilité pour tromper le discriminateur et ainsi générer une sortie proche de la distribution souhaitée.

Le discriminateur est un classifieur binaire, ce qui mène généralement à y utiliser l'entropie binaire croisée comme fonction de perte. Cette fonction repose sur une valeur d'information afin de déterminer la probabilité que l'entrée reçue est l'entrée réelle (marquée par 1), ou fausse (marquée par 0) (Godoy, 2018). La fonction est définie comme suit :

$$-\frac{1}{N}\sum_{i=1}^{n} y_i \times \log(p(y_i)) + (1 - y_i) \times \log(1 - p(y_i)) \tag{2.1}$$

Cette fonction de perte comprend deux parties : la première représente la probabilité que le discriminateur identifie une entrée x comme faisant partie d'une distribution de données réelles. Cette partie est décrite par l'équation :

$$\mathbb{E}_x \sim_{p_{data}(x)} [\log D(x)] \tag{2.2}$$

La deuxième partie de la fonction de perte représente la probabilité que le discriminateur identifie la valeur générée par G, comme étant une valeur qui n'appartient pas à la distribution des données réelles. Par conséquent, l'entrée $G(z)$ est fausse. En appliquant l'entropie croisée, la 2e partie de notre fonction devient alors la probabilité inverse que le discriminateur identifie l'entrée $G(z)$ soit vraie.

$$\mathbb{E}_x \sim_{p_z(z)} [\log(1 - D(G(z))] \tag{2.3}$$

De cette façon, on garde une seule fonction de perte (Équation 2.4) pour le discriminateur, qui la maximisera dans le but d'optimiser ses performances et différencier plus efficacement les distributions reçues.

$$\mathbb{E}_x \sim_{p_{data}(x)} [\log D(x)] + \mathbb{E}_z \sim_{p_z(z)} [\log(1 - D(G(z))] \tag{2.4}$$

Au final, on a affaire à une unique fonction de perte 2.5, qui est à la fois minimisée par le générateur et maximisée par le discriminateur. Ceci simplifie considérablement l'architecture des modèles génératifs antagonistes (Goodfellow *et al.*, 2014) :

$$\min_G \max_D (D, G) = \mathbb{E}_x \sim_{p_{data}(x)} [\log D(x)] + \mathbb{E}_z \sim_{p_{data}(z)} [1 - \log D(G(z))] \tag{2.5}$$

2.2.2 Optimisation des GANs :

Comme toutes les autres architectures neuronales, les réseaux génératifs antagonistes ont besoin d'ajuster leurs paramètres par apprentissage pour fournir les bons résultats. Les deux réseaux qui constituent un GAN utilisent alors la méthode de descente de gradient pour optimiser leurs paramètres (θ) et essayer de prendre chacun le dessus sur le réseau adverse.

L'article qui introduit les modelés génératifs antagonistes utilise la descente de gradient stochastique (Goodfellow *et al.*, 2014). Cette dernière se distingue par la mise à jour des paramètres θ en utilisant l'erreur instantanée au lieu de moyennes difficiles à déterminer. Par conséquent, la vitesse de traitement est augmentée et on obtient un algorithme plus rapide. (Ruder, 2016).

Cependant, puisque le générateur essaye de minimiser les paramètres $G(z, \theta)$, et le discriminateur essaye de maximiser $D(x, \theta)$, les GANs utilisent alors deux gradients différents :

— Une descente de gradient stochastique par le générateur afin de chercher le minimum global ;

— une ascendante de gradient par le discriminateur afin de chercher le maximum global de la courbe ;

D'un point de vue pratique, l'algorithme qui représente l'optimisation des modèles génératifs antagonistes est décrit et démontré dans le papier (Goodfellow *et al.*, 2014)

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Figure 2.3: Algorithme de la descente de gradient GANs (Goodfellow *et al.*, 2014)

La figure 2.3 montre que l'algorithme prend un échantillonnage à partir de données aléatoires z et un échantillon de taille m à partir d'un jeu de donnée réel x pour un nombre d'itérations prédéterminées k. Les poids du discriminateur ($D(x, \theta)$) sont ensuite optimisés à l'aide de la fonction de coût (équation 2.4) et d'une ascendante de gradient. Celle-ci vise à chercher le maximum global de notre gradient, et maximiser par la même occasion la probabilité que le discriminateur différencie une entrée réelle d'un bruit généré.

Le procédé inverse est appliqué pour l'optimisation du générateur ($G(z, \theta)$). Une descente de gradient sert à minimiser notre fonction de coût. Pour finir, l'algorithme est répété jusqu'à atteindre un équilibre.

En principe, si l'on possède le temps et la puissance nécessaires, il est possible d'atteindre un point de convergence et d'équilibre. Un équilibre représente une distance quasiment nulle entre la distribution générée et la distribution réelle x, rendant les deux distributions indissociables par le discriminateur ou par l'humain. Goodfellow a démontré que le point d'équilibre, dit de Nash, est égal à $V(G, D) = -\log(4)$ (Goodfellow *et al.*, 2014).

Cependant, cet équilibre n'est pas atteignable en pratique, car on ne dispose pas des conditions citées dans l'article (Capacité et temps d'entraînement optimal). Par conséquent, nous sommes confrontés à des difficultés lors de l'apprentissage et l'optimisation des GANs.

2.2.3 Difficulté d'apprentissage des GANs :

Des problèmes peuvent survenir lors de la phase d'entraînement des réseaux génératifs antagonistes, qui peuvent aboutir à une baisse de performance, des résultats non attendus et des coûts d'apprentissage élevés. Parmi ces problèmes on retrouve :

Problème de disparition du gradient : la disparition du gradient (Vanishing gradient) est un problème récurrent de l'apprentissage profond. En ce qui concerne les GANs, ce phénomène se produit dans le cas où le discriminateur développe une meilleure capacité d'apprentissage que celle de son adversaire. Ce dernier classifie alors tous les échantillons envoyés par le générateur comme étant des entrées fausses. Cela implique une disparition progressive du gradient et donc une qualité moindre des résultats fournis par le modèle (Arjovsky et Bottou, 2017).

Effondrement de l'apprentissage : Appelé « mode collapse » en anglais, c'est un autre problème important qui peut toucher l'apprentissage d'un réseau génératif antagoniste. L'effondrement de l'apprentissage est caractérisé par un manque de diversité des données générées par le GAN, qui se met à produire les mêmes sorties pour toutes les données d'apprentissage (Metz *et al.*, 2016).

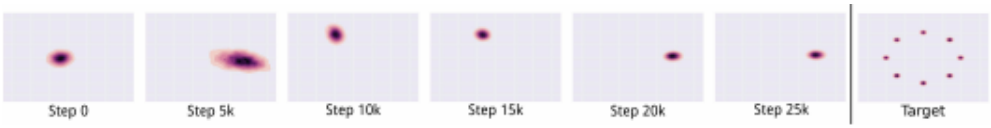

Figure 2.4: Illustration du mode Collapse (Metz *et al.*, 2016)

On observe dans la figure 2.4 que le générateur produit la même forme qui se voit rejetée par le discriminateur. On remarque aussi que les données produites sont très différentes de celle attendue.

2.2.4 Améliorations d'apprentissage des GANs

Pour remédier aux problèmes cités auparavant, il existe plusieurs solutions proposées par l'état de l'art qui sont généralement reliées à la fonction de perte. Certaines de ces solutions sont utilisées pour améliorer le modèle de notre projet de fin d'études.

Fonction de perte non saturée

Une des fonctions proposées dans le papier introduisant les GANs (Goodfellow *et al.*, 2014), est la fonction de perte non saturée. C'est un changement numérique qui est effectué sur la fonction de perte du générateur, pour que ce dernier puisse maximiser la fonction logarithmique au lieu de la minimiser. Le changement en question est appliqué de la sorte :

$$\log(D(G(z)) \longrightarrow \log(1 - D(G(z)))$$

En d'autres termes, le générateur cherche à maximiser la probabilité que l'image soit prédite comme vraie, plutôt que de minimiser la probabilité que l'image soit fausse.

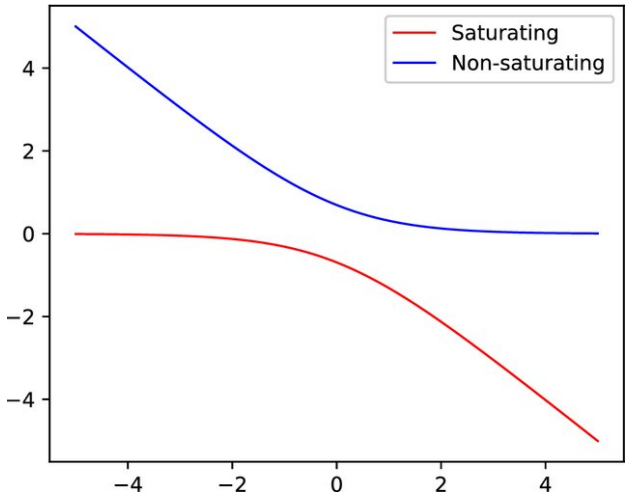

Figure 2.5: Gradient du générateur (ResearchGate)

Comme le montre la figure 2.5, afin de mettre à jour les paramètres du générateur, nous utilisons un gradient ascendant au lieu d'un gradient descendant. Néanmoins, cette méthode n'a pas été prouvée théoriquement, mais est plutôt le fruit d'une méthode heuristique. Cette transformation numérique implique qu'il n'y a plus de jeu « *minmax* » entre les deux réseaux.

Cette astuce permet d'avoir un gradient plus solide au début de l'entraînement, ainsi qu'une mise à jour des poids plus stables et plus fiables. Cela résout dans la majorité des cas le problème de la disparition du gradient (Goodfellow *et al.*, 2014)

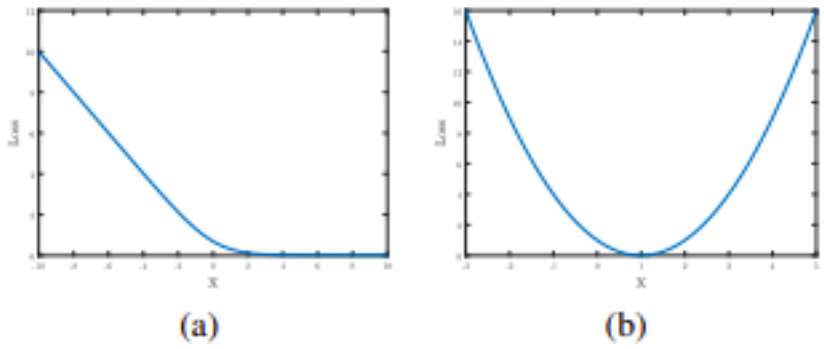

Figure 2.6: (a) Fonction Sigmoide (b) Fonction de moindres carrés (Mao *et al.*, 2017a)

Fonction de moindres carrés

Cette approche a été présentée dans (Mao *et al.*, 2017a) pour remédier à un problème susceptible de produire en utilisant l'entropie binaire croisée comme fonction de perte, plus précisément quand les données générées et les données réelles sont très différentes les unes par rapport aux autres. Cela provoque alors une disparition du gradient et donc un sous-entraînement du modèle. Cette technique met fin alors à la dualité «MinMax entre le générateur et le discriminateur. Dans ce cas, les deux réseaux visent à minimiser chacun leur propre fonction de perte.

La figure 2.6 montre la différence entre la structure sigmoïde de l'entropie croisée et la structure de la fonction du moindre carré.

Le discriminateur essaye de minimiser la somme des probabilités aux carrés de chaque type de distribution.

$$\min_{D} V_{LsGAN}(D) = \frac{1}{2} E_{x \sim p_{data}}[(D(x) - b)^2] + \frac{1}{2} E_{z \sim p_z(z)}[(D(G(z))) - a)^2] \quad (2.6)$$

Dans la formule (2.6) (a) et (b) représentent l'étiquetage «label» de chaque classe respectivement, 1 pour réel et 0 pour les entrées fausses.

La fonction du générateur devient alors :

$$\min_{G} V_{LSGAN}(G) = \frac{1}{2} E_{z \sim p_z(z)}[(D(G(z))) - c)^2] \tag{2.7}$$

Dans la fonction 2.6 la valeur «c» représente la valeur attendue. Dans notre cas le générateur essaye de créer des données réelles, donc on a $c = 1$

Wasserstein GAN

l'architecture Wasserstein GAN (WGAN) dérive une approche mathématique introduite par Arjovsky et al (Arjovsky *et al.*, 2017). Elle vise à remédier l'instabilité rencontrée lors de l'entraînement des réseaux génératifs antagonistes classiques.

Les WGANs évaluent la proximité de distribution des données générées et des données réelles en utilisant la fonction de Wasserstein au lieu de la divergence de Jensen–Shannon qui est implicite dans les GANs Standard, et qui calcule la distance séparant la distribution réelle p_r et celle générée p_t (Menéndez *et al.*, 1997) La distance de Wasserstein aussi appelée "Earth mover Distance" cherche plutôt à savoir le coût minimal pour transformer une distribution en une autre.(Weng, 2017).

La raison principale d'utiliser cette fonction est de simplifier l'entraînement des GANs. D'après Arjovsky, les modèles de WGAN sont moins sensibles à la structure du réseau de neurones et de ses hyperparamétres (Arjovsky *et al.*, 2017). Ceci facilite alors grandement la conception et l'apprentissage d'un modèle.

Dans l'implémentation des WGAN, le discriminateur ne cherche plus à savoir la probabilité que l'entrée soit réelle ou fausse, mais essaye plutôt de calculer un score de vraisemblance ou d'invraisemblance. Ce changement pousse théoriquement à obtenir un score très bas, et donc une distance minime entre la distribution des données réelles et des données générées.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Figure 2.7: Algorithme d'apprentissage des WGAN (Arjovsky *et al.*, 2017))

D'un point de vue pratique, l'implémentation des WGAN se fait en modifiant les fonctions de perte et d'optimisation du discriminateur. Ainsi, on remplace la fonction de sortie sigmoïde par une fonction linéaire afin d'obtenir un score de vraisemblance et non une probabilité, et on optimise le modèle grâce à RMSPROP au lieu d'ADAM, après initialisés les poids dans l'intervalle [-0.01,0.01] (Weng, 2017).

2.3 Conclusion :

De manière générale, les réseaux génératifs antagonistes ont une structure intéressante et efficace pour la génération d'images et de données, malgré les imperfections constatées lors de l'apprentissage de ces dernières. Le modèle ne cesse d'évoluer et d'être adapté pour des problématiques nouvelles. Ces améliorations à travers le temps ont permis de créer de puissantes variantes du réseau génératif antagoniste.

CHAPITRE III

VARIANTES GANS UTILISÉES

Les variantes des réseaux génératifs antagonistes sont les fruits de diverses adaptations du modèle original. Ces modifications ont été réalisées dans l'optique d'apporter des solutions à de nouvelles problématiques et de pallier les limitations rencontrées dans le processus d'apprentissage. Nous nous sommes inspirés de certaines de ces structures dans notre projet pour la conception de notre modèle et son évaluation.

Dans la suite de ce chapitre, nous introduisons les DCGANs, un modèle performant avec une architecture intéressante, dont la compréhension de la structure facilite celle d'autres variantes. Nous décrivons aussi le modèle PIX2PIX qui est utilisé dans la transformation et la traduction d'images, et qui a servi pour évaluer notre modèle. Finalement, nous décrivons le CycleGAN, l'architecture qui a servi d'inspiration pour ce projet.

3.1 Introduction des DCGANs

Dans le domaine de la génération et la transformation d'images, les GANs classiques donnent des sorties bruitées qui mènent à des résultats peu lisibles (Radford *et al.*, 2015). De nouvelles approches ont vu le jour pour remédier ce problème. C'est le cas du GAN convolutif profond DCGAN (Radford *et al.*, 2015) qui fu-

sionne la structure d'un CNN, dont l'efficacité en classification d'images est largement établie, avec le GAN original afin de répondre à l'instabilité des GANs à étudier des images. Le DCGAN est devenu un modèle de référence pour plusieurs architectures de type GAN, grâce à une certaine simplicité architecturale et la capacité de générer des images de haute qualité en peu d'itérations. Il a été appliqué avec succès à divers types de problèmes comme le montre l'exemple de la figure 3.1. Cette figure montre aussi la capacité du DCGAN d'appliquer des opérations et des fusions sur les données dans le but d'en générer de nouvelles.

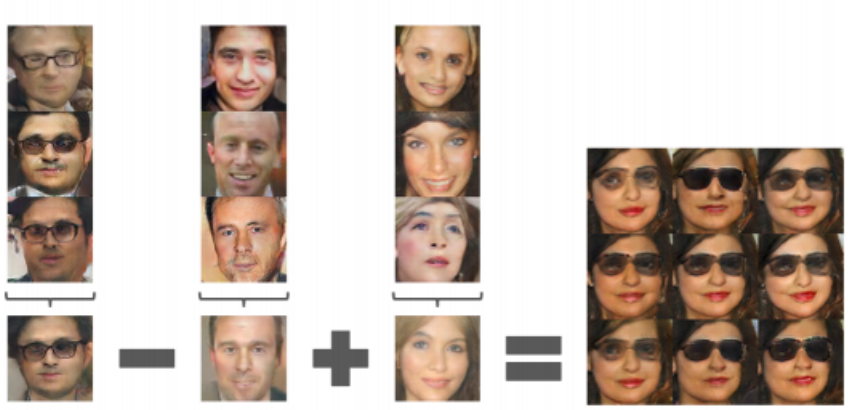

Figure 3.1: Comparaison d'images générées et réelles (Radford *et al.*, 2015)

Cependant, les résultats obtenus ne sont pas toujours concluants, certaines photos générées par le DCGANs ne sont pas lisibles. Cela peut être causé par plusieurs facteurs, dont un mauvais formatage des images, ou un jeu de données insuffisant.

3.1.1 Structure et apprentissage des DCGANs

L'architecture des DCGANs apporte des améliorations pour palier aux problèmes rencontrés lors de l'apprentissage du GAN classique ainsi que certaines autres variantes comme le LAPGAN (Denton *et al.*, 2015). On retrouve dans l'architecture des DCGANs (figure 3.2) des blocs résiduels, des blocs de convolutions et de

convolutions transposées, ainsi que d'autres configurations et modifications.

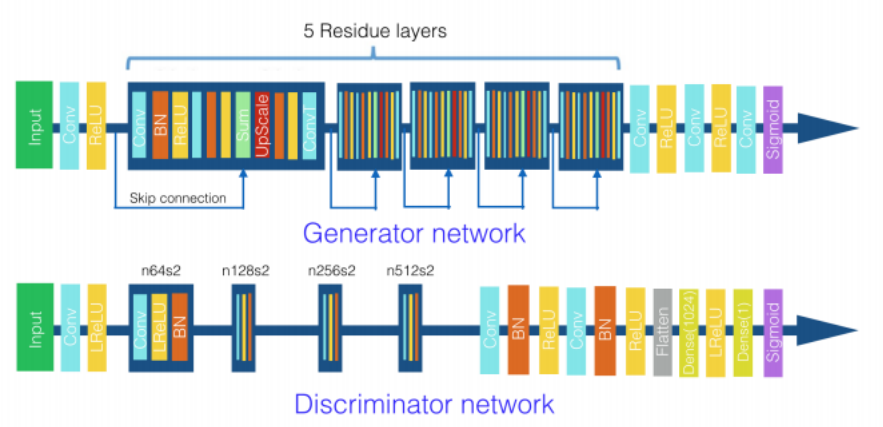

Figure 3.2: Architectures des réseaux composant le DCGAN

Blocs résiduels :

Les blocs résiduels sont une structure qui résout un problème d'apprentissage lié à une grande complexité ou profondeur d'un réseau de neurones. Un ajout conséquent de couches neuronales dans un réseau peut engendrer une saturation et une baisse de précision (He *et al.*, 2016). En d'autres termes, en cas de dégradation du réseau, les blocs résiduels peuvent ignorer certaines connexions et sauter de couche (Figure 3.3).

Réseau uniquement convolutif :

Une des principales modifications des DCGAN, est l'utilisation d'un concept de réseaux uniquement convolutif «All-convolutional network». Ce concept a été introduit dans (Springenberg *et al.*, 2014) pour réduire la dimensionnalité des entrées en se basant sur des pas de convolution «strides» au lieu de couches de compression «pooling». Les «strides» sont utilisées par le discriminateur D pour réduire la

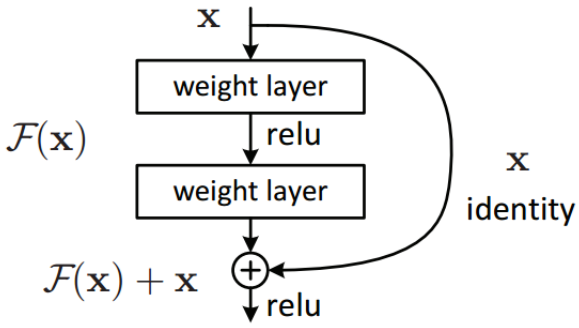

Figure 3.3: Fonctionnement d'un bloc résiduel (He *et al.*, 2016)

taille des entrées, mais elles sont aussi employées par le générateur pour appliquer les *transpositions de convolutions*, un autre concept important des DCGANs.

Connectivité :

La deuxième caractéristique de l'architecture des DCGANs est l'élimination des réseaux entièrement connectés. Cette approche vise à augmenter la stabilité du modèle en s'inspirant des algorithmes de classification qui utilisent la compression par moyenne «average pooling» . La structure des DCGANs est alors telle que plusieurs couches de convolutions ainsi faites se succèdent. Néanmoins, le générateur garde une couche entièrement connectée à son entrée. Le discriminateur quant à lui aplatit les résultats de convolutions pour donner une probabilité grâce à la fonction sigmoïde(Radford *et al.*, 2015)

Normalisation par lots «Batch-normalization» :

L'architecture des DCGANs est composée de plusieurs couches cachées, ce qui est une contrainte pour la mise à jour de poids, mais aussi pour le nombre d'itérations nécessaires pour l'entraînement. Les DCGANs utilisent la normalisation

par lots («Batch-normalization») pour aborder ce problème ; ils calculent les poids optimaux en utilisant des lots normalisés des données traitées, ce qui aide à prévenir l'effondrement de l'apprentissage «mode collapse» et avoir donc une meilleure stabilité (Radford *et al.*, 2015).

Transposition de convolution «Fractionally-strided» :

Les convolutions sont des opérations linéaires entre matrices, elles sont généralement appliquées pour créer une représentation creuse d'une entrée, en l'occurrence une image, en gardant l'essentiel de celle-ci. Ce procédé est surtout populaire dans le domaine de la vision d'ordinateur, car il permet un gain de temps considérable lors de la phase d'entraînement d'un réseau (Albawi *et al.*, 2017). Pour résumer le fonctionnement d'une convolution, on multiplie une matrice «stride» par une autre (l'image d'entrée) pour obtenir une troisième matrice qui représente la compression de l'image initiale.

Dans le fonctionnement des DCGANs, le réseau générateur (G) reçoit un vecteur de données aléatoires z d'une taille généralement fixe, cependant les données traitées par les discriminateurs des DCGANs, sont en forme d'image ou d'entrée à plusieurs dimensions. Si l'on prend par exemple des images de couleurs «RGB» de taille $(64 \times 64 \times 3)$, le générateur doit construire une autre image de la même dimensionnalité à partir du vecteur latent z reçu au départ (voir figure 3.4). On se retrouve alors avec une problématique de décompressions «Upsampling».

Pour résoudre cette problématique, la transposition de convolution est appliquée. Communément appelée «Fractionally-strided», elle est utilisée dans le papier de (Radford *et al.*, 2015) pour «upsampling». Ce concept est une façon de reconstruire la résolution spatiale d'une entrée. On utilise aussi pour ce procédé une matrice «kernel» comme pour les convolutions, cependant le passage du «kernel» et les

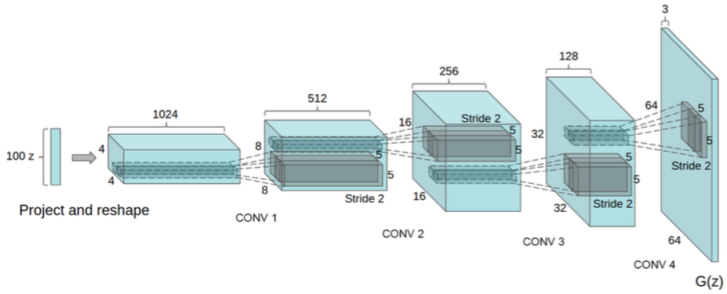

Figure 3.4: Calcul d'une sortie en utilisant les convolutions transposées (Radford *et al.*, 2015)

opérations appliquées sont inversés par rapport aux convolutions (Dumoulin et Visin, 2016).

Fonction d'activation DCGANs :

Le choix des fonctions d'activation est très important lors de l'entraînement d'un réseau de neurones. Dans le papier de (Radford *et al.*, 2015), les auteurs recommandent l'utilisation des fonctions ReLu (Unité linéaire rectifiée) pour le générateur, et sa version ajustée LeakyRelu pour le discriminateur.

Lors de l'utilisation du Relu, on peut tomber sur le problème des neurones morts. Ce problème survient quand la fonction d'activation reçoit des poids ou des entrées négatives ou égales à zéro, dans ce cas-là, la fonction ReLu renvoie la valeur zéro. Un neurone qui renvoie une valeur nulle à chaque fois est considéré comme inactif et donc mort. Dans le cas de figure où plusieurs valeurs du réseau sont négatives, plusieurs neurones sont alors considérés comme morts et donc inutiles (Nayak, 2018). L'apprentissage du générateur est corrélé au gradient du discriminateur. Le discriminateur utilise la fonction LeakyRelu qui résout le problème des neurones mort, et par la même occasion accélère l'entraînement.

La sortie des LeakyRelu sera positive si la valeur d'entrée est positive (même fonctionnement que le ReLu classique), cependant en cas d'entrée négative, LeakyRelu contrôle les sorties grâce à un paramètre alpha qui est une inclinaison de la courbe (voir Figure(3.5), ce qui augmente la tolérance du réseau. $f(x) = max(\alpha \times x, x)$. (Nayak, 2018)

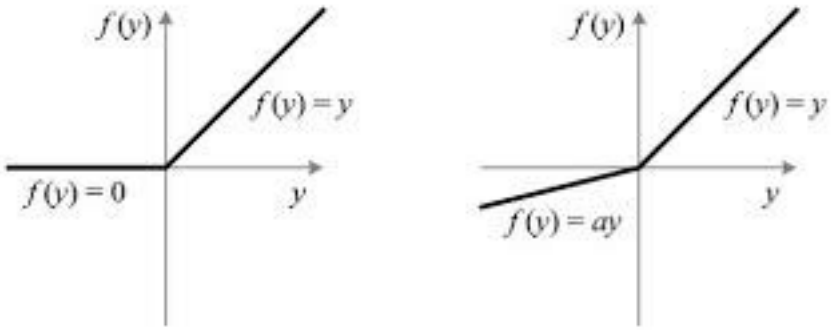

Figure 3.5: LeakyRelu v Relu (Nayak, 2018)

Puisque le discriminateur classifie les entrées en renvoyant la probabilité que l'image soit vraie ou fausse, une fonction sigmoïde est donc logique. Quant au générateur, les auteurs ont opté pour une fonction «tanh».

Les modifications apportées pour créer le DCGANs sont intéressantes et nous ont inspirés sur certains points lors de la conception de notre modèle. L'étude de l'architecture des DCGANs nous a aussi été utile pour mieux comprendre le fonctionnement et les structures des autres variantes de DCGAN et leur capacités d'apprentissage.

3.2 PIX2PIX

PIX2PIX, appelé aussi traduction d'image à image par réseau antagoniste conditionnel, est un modèle de traduction et de transformation d'images. Il reçoit en

entrée une donnée de type A et la transforme vers une autre de type B. Dans l'article orignal (Isola *et al.*, 2017), des images sont principalement utilisées pour valider et montrer l'efficacité du modèle. Le nom «réseau antagoniste conditionnel» fait référence au type d'entrée nécessaire à l'apprentissage du modèle. En effet, ce dernier nécessite un type de donnée spécifique et non un vecteur latent aléatoire comme c'est le cas pour les GANs classiques (Isola *et al.*, 2017).

Figure 3.6: Transformation d'une image satellite en une carte Google et inversement (Isola *et al.*, 2017)

Initialement, le modèle PIX2PIX utilisait l'erreur quadratique moyenne (MSE) comme fonction de perte, cependant, les sorties obtenues manquaient de précision et présentaient du bruit. Ceci impactait naturellement les résultats du modèle, sachant que son objectif premier est de générer des traductions quasi indistinguables des données visées. Pour remédier à cette imperfection, une fonction semblable à celle des GANs classiques a été implémentée (Isola *et al.*, 2017) :

$$L_{(D,G)} = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x,z}[\log(1 - D(G(x,z))] \tag{3.1}$$

On constate que cette fonction de perte utilise deux variables x et y pour effectuer

le calcul. La variable x représente la donnée initiale qu'on souhaite traduire, et la variable y représente la donnée visée. Un exemple commun de l'implémentation du modèle pix2pix est la transformation de croquis en images coloriées de ces mêmes dessins. 3.7).

Figure 3.7: Transformation d'un dessin vers une image colorée (Isola *et al.*, 2017)

Le modèle reçoit deux jeux de données, le premier contient des échantillons de croquis x, et le deuxième leur version colorée y. La norme L1 est appliquée pour calculer la distance entre la distribution initiale et la distribution visée. Cette approche donne de meilleurs résultats que la distance quadratique moyenne.

3.2.1 Structure du modèle PIX2PIX

Discriminateur :

L'architecture du discriminateur dans le modèle PIX2PIX, partage plusieurs similitudes avec celle du DCGAN. Effectivement, on trouve la fonction LeakyRelu ainsi que la normalisation par lots «Batch normalization». Le papier de PIXPIX introduit aussi le concept de «PatchGAN» pour l'apprentissage du discriminateur (Isola *et al.*, 2017). Le «PatchGAN» est un type de filtre qu'on applique à l'entrée

pour fragmenter cette dernière. On calque alors un bloc «Patch» de l'image, de taille fixe NxN, et l'on en extrait une partie qu'on passera au classifieur binaire. Le discriminateur renvoie une grille regroupant les différentes classifications des fragments de l'entrée au lieu de renvoyer une valeur binaire (0 pour donnée fausse et 1 pour vraie).

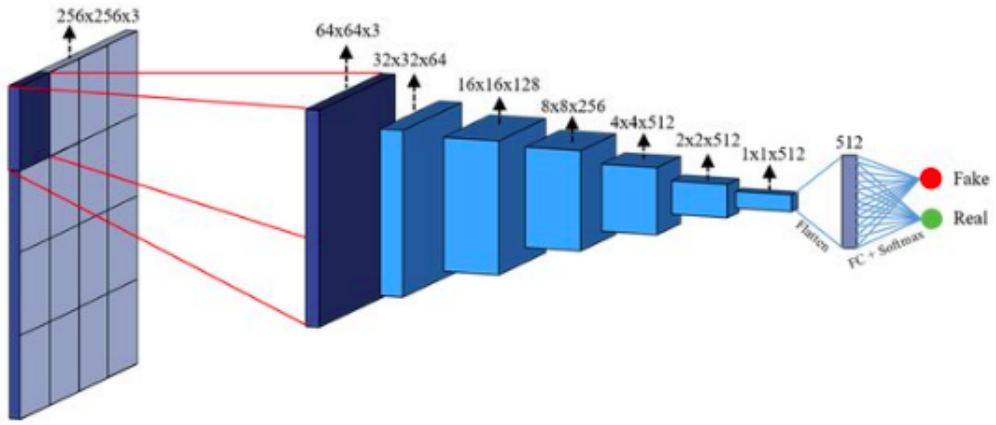

Figure 3.8: Structure du patchGAN dans le discriminateur (Ganokratanaa *et al.*, 2020)

Cette technique permet d'avoir un apprentissage plus simple et plus rapide, et elle épargne aussi la configuration de plusieurs paramètres. Le «patch» peut prendre plusieurs tailles et s'adapter aux entrées. Par exemple, dans l'article l'auteur choisit de prendre une taille ($70x70$) (Isola *et al.*, 2017). Pour la phase de l'optimisation, une descente de gradient est utilisée, cependant, une certaine modification a été apportée : au lieu de minimiser $1 - D(x, g(z, x)$, on minimise la fonction $D(x, g(z, x)$ (Isola *et al.*, 2017).

Générateur

La configuration du générateur est différente de celle des DCGANs, malgré l'utilisation de couches de convolutions. En effet le générateur a recourt à une structure

U-net (figure 3.9). Cette dernière prend une entrée et la sous-échantillonne «down-sampling» et ceci grâce à des couches de convolution et du «poolings». On répète ce procédé jusqu'à atteindre une carte de caractéristique «feature map» d'une taille égale à 1×1.

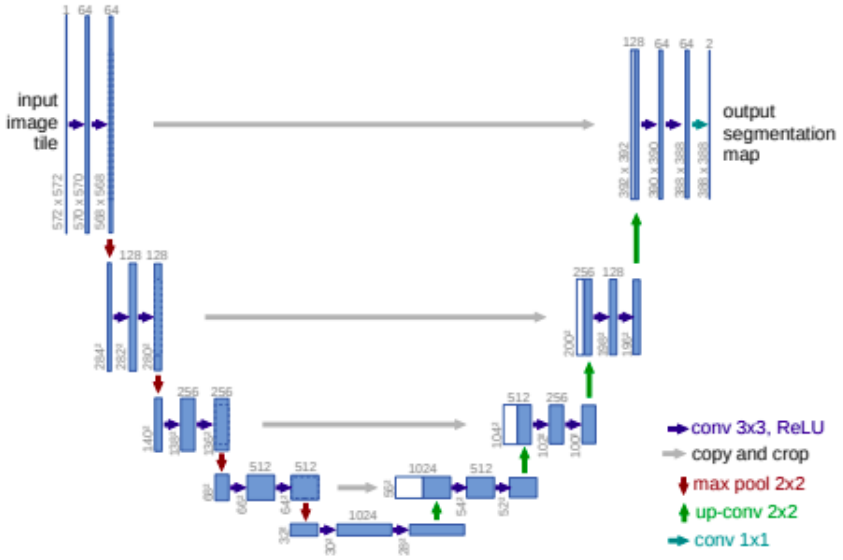

Figure 3.9: Architecture U-net utilisé par le générateur (Ronneberger *et al.*, 2015)

On applique ensuite le procédé inverse, c'est-à-dire on sur-échantillonne («Up-sample») la carte de caractéristiques pour atteindre la taille et la dimension voulue. La structure U-net utilise les convolutions et les convolutions transposées pour imiter un processus d'encodage et décodage des entrées. (Isola *et al.*, 2017).

3.2.2 Contrainte d'utilisation

PIX2PIX est un puissant modèle de traduction et de transformation d'image. Néanmoins, sa structure limite son utilisation, comme décris dans les sections précédentes, il est nécessaire pour l'apprentissage et le bon fonctionnement du

modèle, d'avoir des données de la même nature et du même type, il est aussi nécessaire d'avoir une certaine symétrie et une corrélation entre les données initiales et les données visées.

3.3 CycleGan :

Le CycleGAN est une variante des modèles génératifs antagonistes qui reprend en partie le principe de PIX2PIX, cependant la nature de ce réseau permet l'apprentissage de façon non supervisé d'un jeu de données non formaté (Zhu *et al.*, 2017). Cette technique de traduction impaire résout alors les problèmes rencontrés par le modèle PIX2PIX. En d'autres mots, il n'est plus nécessaire d'avoir une correspondance entre les images d'entrées et les images visées.

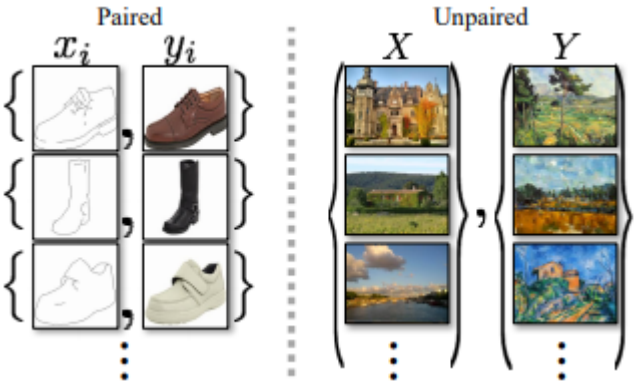

Figure 3.10: Difference entre la transformation paire et impaire (Zhu *et al.*, 2017)

L'objectif des CycleGANs ne s'arrête pas uniquement à transférer un style d'un domaine A à un autre domaine B, mais il vise aussi à mettre en place le processus inverse. L'exemple le plus fréquent est la transformation de photos de zèbres en chevaux 3.11 réalisé par (Zhu *et al.*, 2017). Dans cette implémentation, le CycleGAN est alors utilisé pour traduire un jeu de données constitué d'images de chevaux en un autre jeu de données composé de photos de zèbres. Le CycleGAN

ne reçoit pas de supervision directe, et essaye à partir uniquement de chaque photo de la base (A) afin d'obtenir une nouvelle image portant les caractéristiques du 2e jeu de données (B), il essaye ensuite de reconstruire l'image initiale à partir du résultat généré. Ce procédé, ressemblant à un cycle, représente l'essence même du modèle.

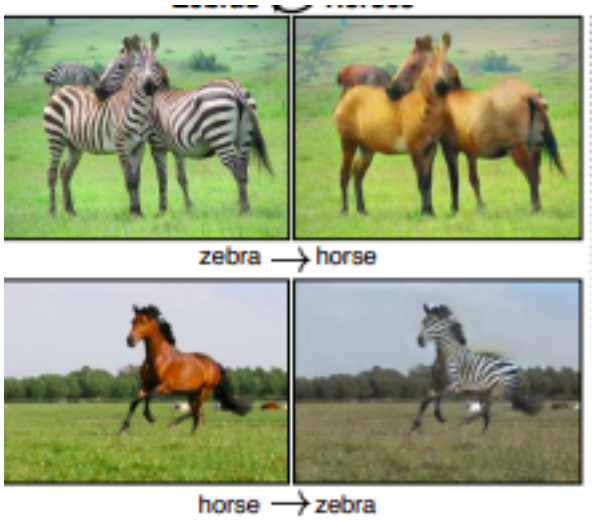

Figure 3.11: Transformation de photo de zèbre en cheval et inversement. (Zhu *et al.*, 2017)

3.3.1 Architecture et apprentissage du modèle CycleGAN

La structure du CycleGAN est une combinaison de deux GANs, le premier utilisé pour générer les nouvelles images, et le deuxième pour la reconstruction des entrées initiales.

Comme on peut l'observer dans la figure 3.12, nous avons un premier générateur (G) qui prend une entrée (X) et la transforme en la distribution visée (Y) ($G : X \rightarrow Y$). Le discriminateur D_y classifie alors la donnée générée (Y) en vrai ou faux. le processus inverse est ensuite appliqué pour reconstruire (Y) en une donnée

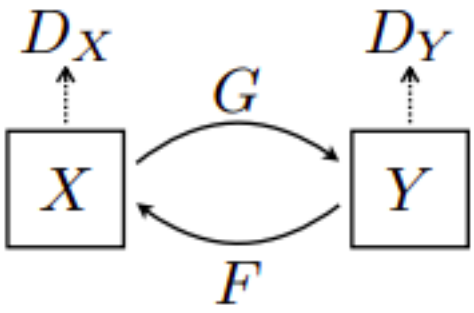

Figure 3.12: Architecture des CycleGAN (Zhu *et al.*, 2017)

initiale (X) (« $F : Y \rightarrow X$») (Zhu *et al.*, 2017).

Quant à la structure interne des générateurs, ces derniers utilisent trois couches de convolution résiduelles, ainsi qu'une instance de normalisation. On trouve aussi des convolutions transposées pour sur-échantillonner le tout. Le discriminateur possède une structure très similaire à celle vue dans le modèle PIX2PIX, il applique des Patch-GANs sur les entrées et renvoie si l'entité reçue est vraie ou fausse.(Zhu *et al.*, 2017). L'apprentissage est alors la répétition de ce processus sur plusieurs itérations.

Fonction de perte

Deux fonctions de pertes sont utilisées pour la mise à jour des paramètres du réseau. La première concerne la reconstruction des données et est intitulée *consistency loss*. Cette dernière calcule la norme L1 entre les images reconstituées et les images initiales :

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\|G(F(x)) - y\|_1] \quad (3.2)$$

on obtient de meilleurs résultats de reconstitution en la minimisant. La deuxième

fonction, est une fonction antagoniste similaire à celle du modèle PIX2PIX :

$$L_{GAN}(G, D_y, X, Y) = E_{y \sim p_{data}(y)}[\log D_y(y)] + E_{x \sim p_x(x)}[\log(1 - D_y(G(x)))] \quad (3.3)$$

Les deux fonctions sont combinées pour obtenir la fonction 3.4.

$$L(G, F, D_x, D_y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, X, Y) + \lambda L_{cyc}(G, F) \quad (3.4)$$

3.4 Conclusion

Les variantes introduites dans ce chapitre nous ont servi d'une manière ou d'une autre à la conception de notre architecture. Les modèles ont prouvé leurs efficacités dans la génération de nouvelles distributions, dont le transfert de caractéristiques sur des images, mais aussi de spectres audio (Kaneko et Kameoka, 2018). Cependant, malgré la puissance de ces modèles, ils souffrent encore de faiblesses dans leurs apprentissages, et aussi de lacunes pour la transformation d'entrées géométriques et certains types de données (Zhu *et al.*, 2017).

Dans notre projet nous nous sommes inspirés des points forts de ces architectures pour mettre en place notre modèle et l'adapter pour résoudre des problématiques de traductions de séquences ECG.

CHAPITRE IV

MÉTHODOLOGIE ET CONCEPTION DU CYCLEGAN SIMPLIFIÉ

4.1 Introduction

Le but de notre projet étant de développer un algorithme d'apprentissage profond pour transcrire les signaux chronologiques de type électrocardiogramme d'une forme vers une autre, nous nous sommes penchés sur les architectures génératives habituellement utilisées dans la conversion et la traduction des données. Parmi les modèles trouvés, le CycleGAN nous a paru être le plus prometteur. Il est facile à adapter et a déjà donné d'excellents résultats pour des problèmes similaires aux nôtres, mais s'appliquant à d'autres types de données.

Le design et la validation de notre modèle de transcription de signaux se sont déroulés en quatre étapes :

1. Conception d'un CycleGAN simplifié ;
2. Évaluation préliminaire à l'aide de deux jeux de données distincts ;
3. Réglage et Validation du CycleGAN simplifié pour deux types de signaux ECG ;
4. Comparaison de la performance du modèle avec d'autres modèles de l'état de l'art.

Nous décrivons dans ce chapitre les deux premières étapes.

4.2 Conception du CycleGAN simplifié

Les séquences chronologiques traitées sont représentées par des vecteurs dans un espace à une dimension, et notre modèle doit traiter des entrées de la même taille. Cependant, la structure du CycleGAN classique est orientée vers le traitement d'objets bidimensionnels. Par conséquent, la structure de ce modèle peut être inutilement complexe pour notre besoin. Ainsi, la version de base du CycleGAN comporte plusieurs couches de convolution et de convolution transposée sans utilité avérée pour le traitement d'entrées unidimensionnelles et qui augmentent la complexité des calculs sans améliorer la performance pour autant. Cela nous mène à développer un CycleGAN simplifié qui est exempt de ces couches et où les entrées du générateur et du discriminateur sont adaptées pour des entrées à une dimension.

4.2.1 Discriminateur du CycleGAN simplifié :

La structure du discriminateur est façonnée de manière à pouvoir gérer une entrée unidimensionnelle. Pour y parvenir, nous nous sommes inspirés du modèle GAN à une dimension décrit dans (Brownlee, 2019a). Sa structure est comme suit :

- Une couche d'entrée ajustée pour recevoir un vecteur d'une taille donnée ;
- Deux couches cachées, chacune couplée à une fonction d'activation Leaky-ReLu et de décrochage «dropout» ;
- Une couche de sortie avec une fonction de sortie sigmoïde pour déterminer si l'entrée initiale est fausse ou réelle ;

Dans notre modèle, nous utilisons l'erreur quadratique moyenne comme fonction de perte pour le discriminateur au lieu de l'entropie binaire croisée comme c'est

le cas dans l'implémentation des GANs classique. Ce choix permet de pénaliser encore plus les erreurs lors de la prise de décision par le modèle (Mao *et al.*, 2017b). Pour la fonction d'optimisation, nous avons sélectionné la metaheuristique des moments adaptatifs («adaptive moments ou Adam»), après qu'elle nous a fourni de meilleurs résultats comparés à d'autres optimiseurs tels que la descente de gradient stochastique et RMSPROP.

4.2.2 Générateur du CycleGAN simplfié

Nous avons simplifié le générateur en remplaçant tous les blocs résiduels et les couches de convolution de l'implémentation de base (Zhu *et al.*, 2017) par des couches cachées simples avec des fonctions d'activation de type ReLu et une couche de Dropout. Pour la couche de sortie, une fonction linéaire est utilisée. Le fonctionnement du générateur reste inchangé pour le reste, et il réussit à transformer une séquence chronologique d'un type A vers un type B. 4.2

Dans l'implémentation qui a inspiré notre conception, le générateur met à jour ses poids à travers un modèle composé.

4.2.3 Modèle composé du CycleGAN simplifié

Pour entraîner le CycleGAN, nous avons besoin d'un générateur qui transforme le domaine A vers le domaine B (AtoB), un deuxième générateur qui exécute le procédé inverse(BtoA) et deux discriminateurs qui se spécialisent chacun dans la classification d'un type de donnée différent (domaine A ou B).

L'implémentation qui inspire la conception de notre architecture (Brownlee, 2019b), met à jour les poids des générateurs à l'aide d'un modèle composé. Ce dernier est constitué d'un générateur(AtoB) entraînable, le discriminateur(B) lié au pre-

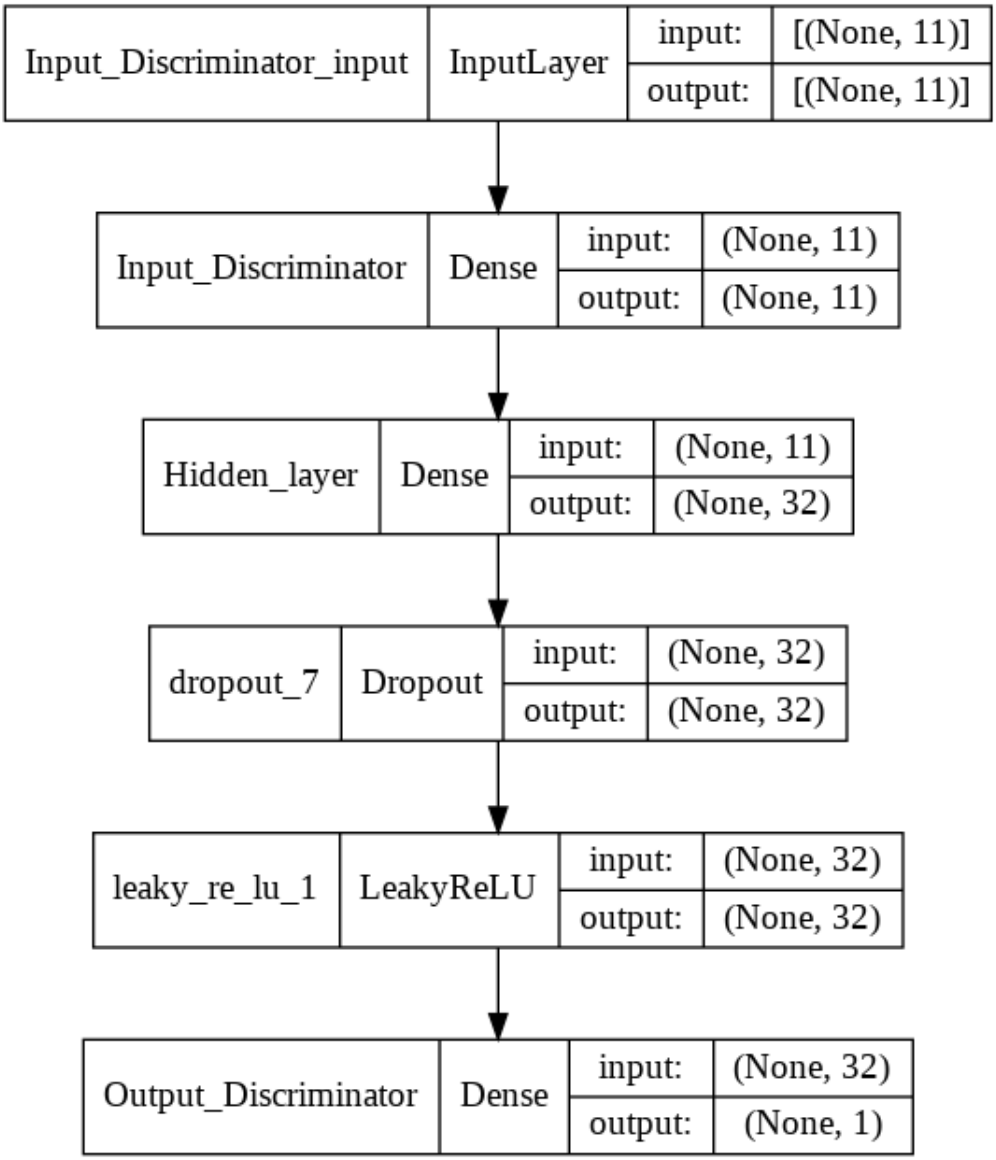

Figure 4.1: Discriminateur du CycleGAN simplifié

mier générateur et un deuxième générateur(BtoA) qui est utilisé uniquement pour calculer la perte (Figure 4.3). Pour mettre à jour les poids du deuxième générateur(BtoA), un deuxième modèle composé est nécessaire.

Le modèle composé calcule quatre fonctions de perte, chacune associée à un coeffi-

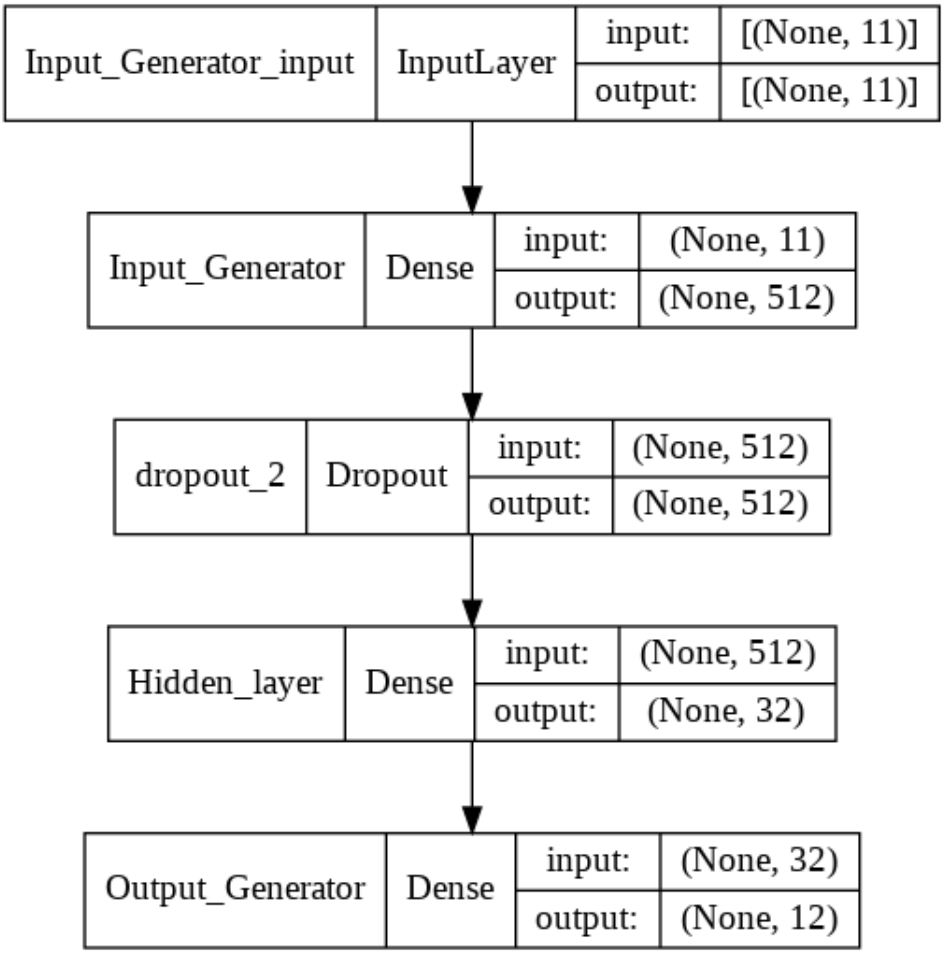

Figure 4.2: Structure du générateur(CycleGAN)

cient lambda «Loss weight», ces coefficients permettent de ralentir ou d'accélérer l'apprentissage d'un modèle par rapport aux autres. Par exemple dans l'implémentation (Brownlee, 2019b), l'auteur utilise les poids [1,5, 10,10] pour désigner que la perte du cycle vers l'avant et du cycle vers l'arrière a plus d'importance que la perte antagoniste et la fonction d'identité dans la mise à jour des poids du générateur. Le modèle calcule ensuite grâce aux lambda et aux résultats des fonctions de pertes, une moyenne pondérée qui servira l'optimisation du générateur.

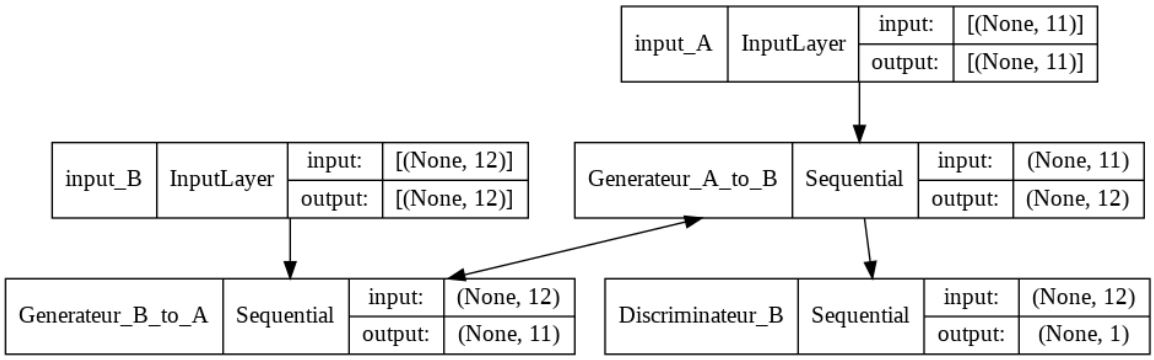

Figure 4.3: Modèle composite associé au générateur(AtoB)

Perte antagoniste : Le générateur(AtoB) reçoit une entrée du domaine A et la traduit vers le domaine B. Le résultat est ensuite envoyé au discriminateur (B) qui calcule si l'entrée reçue est générée ou réelle. On utilise une fonction d'entropie croisée binaire ou une fonction MSE pour calculer la perte.

Fonction d'identité : Le générateur(AtoB) reçoit cette fois une entrée du domaine B. Le modèle génère une sortie identique à l'entrée. On calcule alors l'erreur absolue moyenne à partir de la sortie générée et de l'entrée.

$$(\frac{1}{n}) \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

Cette fonction est utilisée dans l'implémentation du CycleGAN classique (Zhu *et al.*, 2017), pour conserver la texture et la couleur des images générées. Cette fonction est donc inutile pour traduire des séquences chronologiques et nous avons décidé de ne pas l'intégrer dans notre version du modèle.

Perte du cycle vers l'avant : La sortie du générateur(AtoB) est envoyée au deuxième générateur qui la transforme vers le domaine A. Le modèle calcule

ensuite l'erreur absolue moyenne entre la sortie obtenue (A') et l'entrée initiale du CycleGAN (entrée réelle A).

$$DomaineA \rightarrow Generator(AtoB) \rightarrow B' \rightarrow GeneratorBtoA \rightarrow A'$$

Perte du cycle vers l'arriéré : Le générateur(BtoA) reçoit une entrée du domaine B et renvoi (A'), sa sortie est ensuite envoyée au générateur(AtoB) qui générera à son tour (B'). Le modèle calcule ensuite l'erreur absolue moyenne entre l'entrée initiale B et la traduction générée B'.

$$DomaineB \rightarrow Generator(BtoA) \rightarrow A' \rightarrow GenerateurAtoB \rightarrow B'$$

4.3 Évaluation et réglage

Nous avons appliqué l'architecture conçue sur deux jeux de données de types et de tailles différentes, dans le but de déterminer le comportement du modèle pour des signaux distincts et d'ajuster ses paramètres pour une performance optimale.

4.3.1 Transformation des valeurs d'une fonction carrée en valeurs de fonction cubique

Pour la première évaluation, nous nous sommes inspirés d'une application qui transforme une valeur de variable numérique en l'élevant au carré (Brownlee, 2019a). Nous générons deux jeux de données, composés chacun de paires (x,f(x)), où x est une variable numérique définie dans l'intervalle [-0.5, 0.5], et où f(x) = x^2 pour le premier jeu et f(x) = x^3 pour le deuxième jeu. L'objectif du CycleGAN simplifié est d'apprendre à convertir chaque carrée en valeur cubique.

Pour l'entraînement du modèle, nous avons généré 1000 pairs (x,f(x)) pour chaque

type de séquences avec les deux fonctions définies au tableau 4.1). Nous avons réservé 200 autres paires pour la phase de test.

Tableau 4.1: Génération des jeux de données

```
def generate_testA(n):
  X1 = rand(n) - 0.5
  X2 = X1 * X1
  X1 = X1.reshape(n, 1)
  X2 = X2.reshape(n, 1)
  X = hstack((X1, X2))
  y = np.ones((n, 1))
  return X, y
```

```
def generate_testB(n):
  X1 = rand(n) - 0.5
  X2 = X1 * X1 * X1
  X1 = X1.reshape(n, 1)
  X2 = X2.reshape(n, 1)
  X = hstack((X1, X2))
  y = np.ones((n, 1))
  return X, y
```

Une fois les des deux séquences générées, nous ajustons les couches d'entrées et de sorties du modèle pour qu'ils puissent recevoir des vecteurs de taille (1×2), puis nous lançons une boucle d'apprentissage de 1000 itérations. La figure 4.4 affiche 100 vecteurs différents du signal source, et les valeurs correspondantes du signal cible et du signal obtenu après apprentissage. Le graphique bleu donne les valeurs carrées, tandis que le graphe orange donne les valeurs cubiques et le graphe vert, le résultat de la transformation des valeurs du premier type de données vers le 2e. Comme on peut le voir, les résultats obtenus sont prometteurs, mais la conversion est effectuée sur des valeurs individuelles de fonctions et non des séquences entières.

4.4 Transformation de séquences de valeurs

Pour cette évaluation, nous appliquons le modèle à des séquences numériques entières au lieu de valeurs extraites du jeu de donnée. Dans cette perspective, nous simulons des signaux unidimensionnels à l'aide d'images aplaties par concaténa-

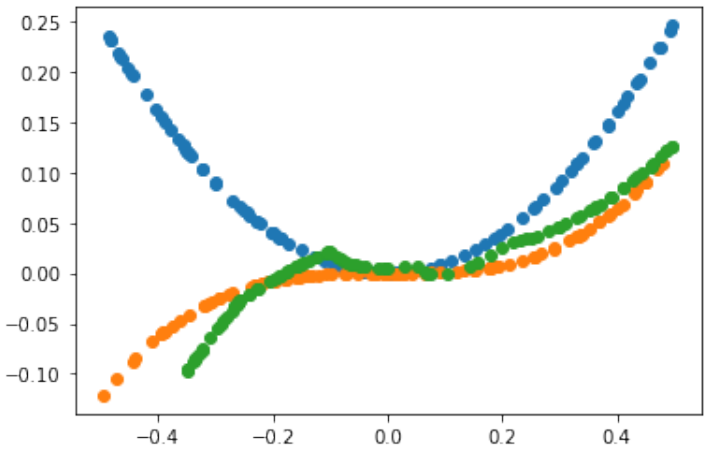

Figure 4.4: Résultats obtenus par notre modèle (vert) pour convertir une fonction carrée (bleu) en fonction cubique (orange)

tion de lignes. Ainsi, chaque image de dimension $n \times n$ devient un vecteur de dimension $1 \times n^2$.

MNSIT «Modified National Institute of Standards and Technology database» est un jeu d'images couramment utilisé pour l'évaluation des modèles d'apprentissage profond. Il est composé de 60000 images en noir et blanc de taille (28×28), chacune représentant un chiffre manuscrit ; 10000 autres images réservées à la phase de test. Après aplatissement, chaque image devient un vecteur de taille (1×784). La Figure 4.5 montre un exemple pour le chiffre 3.

Notre CycleGAN simplifié a pour but de transformer les images aplaties d'un chiffre vers un autre. Dans cette optique, nous extrayons 5600 images représentant le chiffre 3 et 5600 autres représentant le chiffre 5, ainsi que 800 échantillons supplémentaires consacrés à la phase de test. Une fois les données extraites, nous normalisons les valeurs des pixels en les projetant dans l'intervalle [-1,1] avec la formule $(x/127.5 - 1)$ (TensorflowDocs, 2021).

Après l'ajustement de la taille des couches d'entrées et de sorties des générateurs

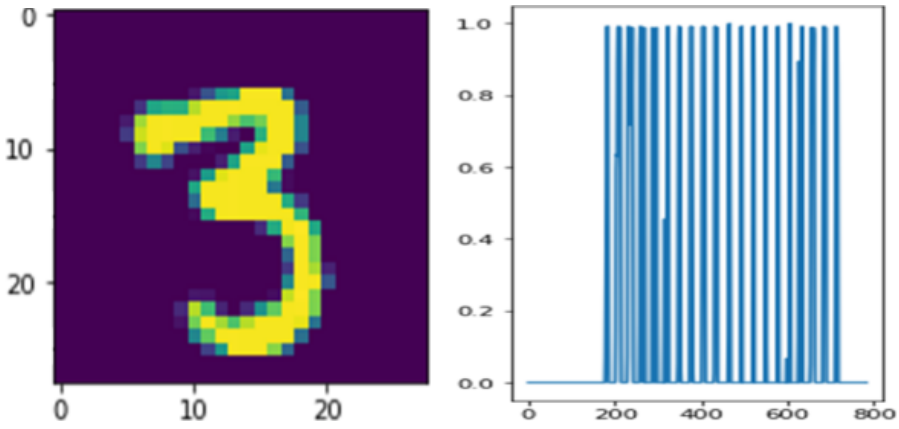

Figure 4.5: Exemple d'une image du chiffre «3» et sa version aplatie

et du discriminateur, nous lançons l'entraînement du modèle sur 1000 itérations avec un taux d'apprentissage «learning rate» de 0.02 et une taille d'échantillonnage «batch size» de 32. Une fois l'apprentissage terminé, nous remettons après chaque inférence les signaux sources et cibles au format 28x28 afin d'avoir une meilleure visualisation. La figure 4.6 montre un exemple concret.

4.5 Conclusion :

Ce chapitre décrit en premier lieu les étapes de conception du CycleGAN simplifié en couvrant les modifications apportées au discriminateur et au générateur, ainsi que les paramètres et les fonctions utilisées pour le bon fonctionnement du modèle. En second lieu, nous avons entamé un processus d'évaluation de l'architecture, et ceci en l'appliquant à deux jeux de données différents, le premier comportant des valeurs scalaires et un autre constitué d'images MNSIT. La retranscription des données est un succès dans l'ensemble, nous pouvons constater une bonne qualité de traduction et une réaction positive du modèle vis-à-vis de types de données distincts. Néanmoins, dans le cas de traduire des signaux physiologiques comme l'électrocardiogramme, plusieurs signaux sont enregistrés simultanément,

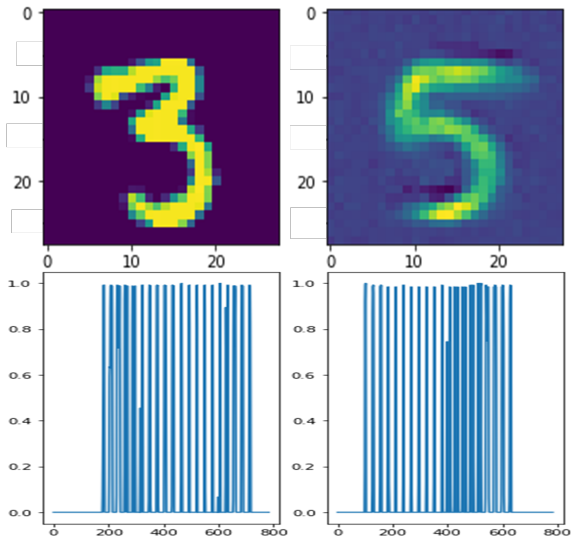

Figure 4.6: Exemple de traduction d'une image du chiffre 3 vers le chiffre 5 (en haut). Images aplaties pour simuler un signal (en bas)

cela implique d'une manière générale que la retranscription de ces mêmes signaux nécessite d'être effectuée en parallèle, en tenant compte des dépendances ou corrélations qui peuvent exister entre les canaux. Il devient important alors de formater les données convenablement pour obtenir la meilleure retranscription possible.

CHAPITRE V

APPLICATION DU CYCLEGAN SIMPLIFIÉ À L'ECG

Le domaine médical est en constante évolution en se renouvelant d'année en année. Plusieurs entreprises dans ce secteur étudient l'intégration d'algorithmes d'apprentissage profond pour résoudre des problèmes particuliers, et aussi simplifier certaines tâches liées au domaine (analyse, diagnostique, ...) (Suzuki, 2017). L'utilisation de modèles d'apprentissage ne se limite pas à des applications directes, mais peut également soutenir des projets plus généraux. Dans notre cas par exemple, le CycleGAN simplifié est conçu pour prendre en charge et accompagner une nouvelle méthode de collecte des signaux ECG.

Dans ce chapitre, nous présentons la méthode de collecte SIGNUM, ensuite nous décrivons l'application de notre CycleGAN simplifiée à la conversion des signaux ECG récoltés par SIGNUM en signaux standards. En conclusion, nous comparons la performance de notre modèle avec d'autres algorithmes d'apprentissage profond populaire, notamment, les LSTM, PIX2PIX et les perceptrons multicouches.

Pour quantifier la performance de notre architecture et avoir un point de comparaison avec les autres modèles cités ci-dessus, nous calculons la distance entre la sortie prédite $\hat{\mathbf{s}}$ et le résultat attendu $\mathbf{s}$ à l'aide de l'erreur quadratique moyenne. Pour un nombre n d'exemples traités, nous utilisons l'équation suivante :

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(s_i - \hat{s}_i)^2$$

Pour l'exécution et l'évaluation de notre modèle CycleGAN simplifié, nous utilisons la plate-forme virtuelle Colab de Google. Cet environnement permet d'entraîner plusieurs modèles en parallèle en ouvrant des sessions différentes tout en bénéficiant de la puissance de calcul du matériel fourni (voir la Figure 5.1 pour les spécifications et une comparaison avec un autre environnement populaire)

Parameter	Google Colab	Kaggle Kernel
GPU	Nvidia K80 / T4	Nvidia P100
GPU Memory	12GB / 16GB	16GB
GPU Memory Clock	0.82GHz / 1.59GHz	1.32GHz
Performance	4.1 TFLOPS / 8.1 TFLOPS	9.3 TFLOPS
Support Mixed Precision	No / Yes	No
GPU Release Year	2014 / 2018	2016
No. CPU Cores	2	2
Available RAM	12GB (upgradable to 26.75GB)	12GB
Disk Space	358GB	5GB

Figure 5.1: Configuration VM Google colab (kazemnejad, 2019)

5.1 La méthode SIGNUM pour acquérir l'ECG

L'électrocardiogramme capacitif (cECG) (Arcelus *et al.*, 2013) est une alternative prometteuse à la dérivation traditionnelle de l'ECG par électrodes humides. Le cECG s'obtient en allongeant le patient ou la patiente sur un matelas isolant qui

enveloppe une grille d'électrodes capacitives qui saisissent l'électrocardiogramme sans contact direct avec la peau, et qui évite d'enduire cette dernière avec un gel conducteur et d'autres désagréments que peut causer la méthode traditionnelle par électrodes à ventouses. Toutefois, cette méthode non conventionnelle de saisie de l'ECG génère des signaux différents des signaux standards. Cela rend difficile son interprétation par les professionnels de la santé qui sont généralement habitués aux signaux dérivés standards.

La méthode de récolte développée par SigNum Preemptive Healthcare, inc(Lessard-Tremblay *et al.*, 2020) fait face à la même problématique citée ci-dessus. Par conséquent, notre objectif est d'utiliser notre modèle CycleGAN simplifié pour convertir les données SIGNUM en données ECG standard (Figure 5.2), ce qui facilitera l'interprétation du CECG tout gardant les avantages de son mode de saisie sans contact.

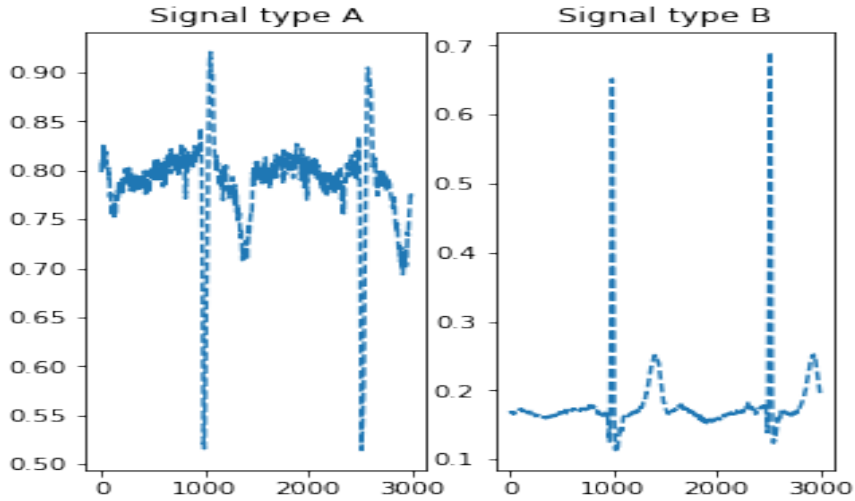

Figure 5.2: Exemple d'un signal SIGNUM(type A) et d'un signal ECG standard (type B)

5.2 Données

Dans le cadre du projet de transcription de séquences de SIGNUM, nous avions à notre disposition 32 enregistrements effectués sur des patients, chacun composé de 11 dérivations du CECG récoltées par SigNum en utilisant leurs électrodes sans contact et 12 dérivations d'ECG conventionnelles recueillies auprès des mêmes patients. Les deux types de signaux comprenaient 10000 échantillons pour chaque dérivation, ce qui donne un total de $32 \times 11 \times 10000$ données pour le CECG et $32 \times 12 \times 10000$ données pour l'ECG conventionnel.

De manière générale, nous avons besoin d'une quantité de données importantes pour entraîner efficacement un modèle non linéaire (Halevy *et al.*, 2009). Dans notre cas, nous avons uniquement les échantillons de 32 patients, une quantité normalement insuffisante pour de bons résultats. Pour remédier à ce manque, on entraîne le CyclGAN à traduire chaque échantillon de taille 1×11 à la fois au lieu de l'enregistrement CECG en entier, dont la taille aurait été 10000×11. Grâce à cette approche, nous augmentons la taille de l'ensemble d'apprentissage SIGNUM de (32×11) à (320000×11), et l'ensemble standard de (32×12) à (320000×12)

Les données sont ensuite normalisé avec la fonction MinMaxScale de Sktlearn pour ramener les valeurs à un intervalle [0,1], à l'exception de trois séquences ECG de patients qu'on a réservées pour la phase de test. Cela permet de normaliser les données d'entraînement à l'aide des valeurs minimales et maximales de leur ensemble, et d'utiliser ces mêmes valeurs pour normaliser les données de test. Cette méthode est employée pour réduire les erreurs de biais ou les écarts de valeurs qui peuvent affecter la performance du modèle.

5.3 CycleGAN simplifié sur SIGNUM

Pour cette expérimentation, nous gardons la même structure de générateur et de discriminateur qu'au chapitre précédent, à l'exception d'ajuster la taille des couches d'entrées et de sorties pour le jeu de données SIGNUM et Standard. Nous utilisons aussi le «Dropout» dans le discriminateur pour réduire le surapprentissage.

Nous avons renoncé à utiliser la fonction identité car inutile à la résolution de notre problème, et nous avons ajusté les coefficients des fonctions de perte dans le modèle composé en baissant les poids de perte («loss_weight») de [1,5,10,10] à [1,4,4]. Nous avons essayé plusieurs poids, en modifiant le ratio entre chaque fonction de perte pour garder ceux qui nous ont fourni le meilleur résultat.

Lors de la phase d'apprentissage, nous exécutons l'algorithme pendant 20000 itérations, traduisant ainsi chaque ligne à la fois (1×11). Une fois l'entraînement terminé, nous concaténons les signaux transformés pour obtenir la séquence ECG d'un patient. Les paramètres appris par le modèle sont appliqués au jeu de test extrait préalablement.

La bibliothèque Matplotlib est utilisée pour afficher la séquence électrocardiogramme. Toutefois, au lieu de dessiner chaque ligne séparément, nous affichons 3000 lignes de chaque colonne (capteur) indépendamment. Grâce à ce procédé, nous pouvons visualiser convenablement les signaux initiaux et les signaux résultants de la traduction (seulement 3000 ont été pris pour des soucis de détails et avoir un graphique plus lisible).

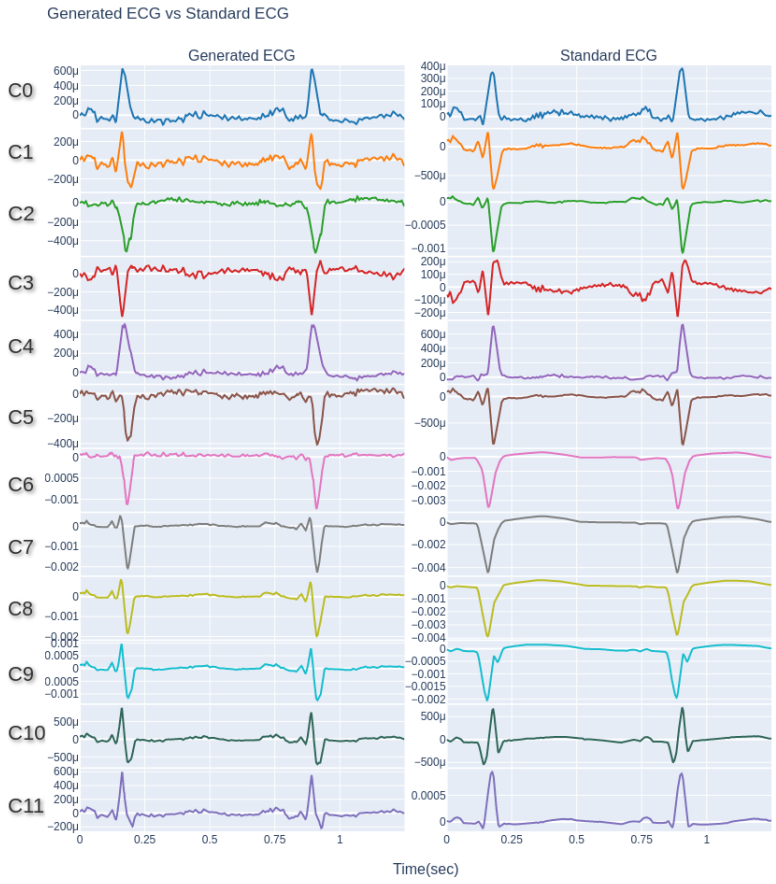

Figure 5.3: Signaux générés et cibles

5.3.1 Résultats :

Nous avons obtenu de bons résultats pour la plupart des canaux, nommément les six premiers et les deux derniers (voir figure 5.3 et l'appendice B). Nous pouvons aussi constater de mauvaises traductions pour certains. Cet effet pourrait être lié au manque d'exemples pour l'entraînement, et pourrait être corrigé avec un jeu de donnée plus conséquent et plus diversifié. Nous sommes arrivés à cette conclusion en appliquant le même modèle sur le jeu de donnée SIGNUM avec deux tailles différentes (29 patients pour la première implémentation et 13 pour la deuxième).

Une réduction de MSE à hauteur de 20% a été observée sur le jeu de donnée contenant plus d'exemples (Figure 5.4)

	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
MSE 13 enregistrement	0.0049	0.0147	0.0260	0.0017	0.0172	0.0218	0.0314	0.0426	0.0531	0.0270	0.0030	0.0056
MSE 29 enregistrement	0.0011	0.0014	0.0007	0.0022	0.0005	0.0010	0.0206	0.0291	0.0313	0.0179	0.0029	0.0019

Figure 5.4: Score MSE des résultats obtenus sur 29 et 13 patients

5.4 Expérimentation et discussion des résultats :

Afin de positionner notre modèle par rapport à l'état de l'art, nous avons implémenté trois autres modèles populaires et évalué leur performances en utilisant le même jeu de données. Il s'agit du modèle PIX2PIX, d'un réseau LSTM et d'un perceptron multicouche (MLP). Une fois l'entraînement de ces modèles effectué, nous calculons l'erreur quadratique moyenne entre la sortie prédite et la sortie cible des différentes architectures

5.4.1 PIX2PIX sur le jeu SIGNUM

Dans un premier temps, nous adaptons le modèle original pour qu'il puisse recevoir des entrées à une seule dimension et par la même occasion traduire nos signaux SIGNUM en signaux ECG standards. Cela est effectué en changeant principalement les types de convolutions utilisés en passant de convolution 2D à des convolutions 1D. Ensuite, nous changeons la taille des «stride» et le type de convolution transposée dans le but d'avoir la taille de sortie désirée. Finalement, nous lançons l'apprentissage du modèle avec les mêmes paramètres de l'implémentation de base sur 5000 itérations.

Cette méthode n'a pas abouti à de bons résultats, et nous avons pu observer une disparition ou saturation du gradient lors de l'apprentissage, car le modèle

n'arrivait plus à apprendre après un certain nombre d'itérations.

En deuxième recours, nous mettons en place le processus inverse. Au lieu de réajuster le modèle en fonction de la nature de nos données, nous redimensionnons les séquences ECG de sorte qu'ils conviennent à l'implémentation originelle de l'architecture PIX2PIX. Une fois l'apprentissage achevé, nous avons obtenu des résultats similaires à ceux de la figure 5.5. Cependant, même si les séquences obtenues sont relativement synchronisées avec les séquences cibles, un bruit important est observé dans la séquence générée. De plus le problème de la disparation de gradient persistait toujours. Les résultats obtenus avec PIX2PIX restent éloignés de ceux attendus.

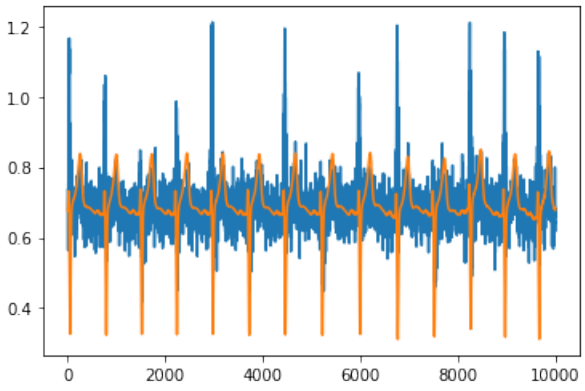

Figure 5.5: Résulat obtenu grâce aux modéles PIX2PIX adapté (Bleu : prédit, Orange : visé)

5.4.2 MLP sur le jeu SIGNUM

Pour l'implémentation du perceptron multicouche, nous utilisons le même procédé pour l'implémentation du CycleGAN simplifié. Le modèle est composé de 3 couches cachées de 50 neurones chacune, une couche d'entrée et une couche de sortie qui renvoie un vecteur (1×11) et un vecteur (1×12) . Nous lançons

l'algorithme pour 5000 itérations afin d'éviter le suapprentissage ou une baisse de précision. Les résultats obtenus sont bons dans l'ensemble, mais reste derrière en précision par rapport au CycleGAN simplifié sur la majorité des canaux. Le temps de l'apprentissage du réseau MLP est plus élevé que le temps d'exécution du CycleGAN simplifié.

5.4.3 LSTM sur le jeu SIGNUM

Le réseau LSTM est une référence parmi les modèles récurrents, spécialement pour le traitement et la transformation de séquences. Pour ce faire, nous avons pris une structure simple comprenant une couche d'entrée (1×11) et une couche de sortie qui reprennent la taille de (1×12) et d'une couche cachée composée de 50 neurones. [1]

Nous effectuons pour le LSTM, le même traitement de données que pour l'implémentation du CycleGAN simplifié. L'apprentissage se déroule sur 5000 itérations, avec une fonction de perte linéaire et un optimiseur de type «Adam». Le nombre d'itérations est plus bas que celui appliqué pour le CycleGAN simplifié car un plus grand nombre entraînerait une baisse de précision du réseau LSTM.

Les signaux générés pour certains canaux sont concluants, mais nous pouvons observé une baisse de précision sur le jeu de test indiquant du surapprentissage. La précision du LSTM peut être améliorée en ajoutant des échantillons supplémentaires, mais dans l'instant elle est moins concluante que le CycleGAN simplifié. De plus le temps d'apprentissage des LSTM est conséquent par rapport au CycleGAN simplifié ($\approx$ 8 heures pour le LSTM contre 48 minutes pour le CycleGAN

1. Nous avons aussi essayé un réseau LSTM empilé (Stacked LSTM ; (Brownlee, 2017)) constitué de trois couches de 35, 100 et 35 neurones, mais sans obtenir de meilleurs résultats.

simplifié).

5.5 Discussion

Après chaque expérimentation et mise en œuvre, nous appliquons le même processus pour comprendre les performances de chaque modèle. En d'autres termes, nous ne nous appuyons pas uniquement sur la visualisation des résultats, mais nous calculons également la distance entre la sortie prédite et la sortie cible. Nous avons pu regrouper les scores MSE de chaque structure sur chaque dérivation ECG dans le tableau 5.1, et ceci afin d'avoir une idée globale de la puissance du CycleGAN simplifié comparée aux autres.

Tableau 5.1: Précision des prédictions du modèle CycleGAN simplifié et des autres architectures

	MSE · 10^3			
Dérivations ECG	**Notre modèle**	**MLP**	**PIX2PIX**	**LSTM**
C0	1	1.2	73.2	**0.2**
C1	**1.1**	2	165.9	2.6
C2	**0.71**	0.77	401	1.3
C3	**2.2**	3	643.9	2.7
C4	**0.5**	1.1	151.8	0.7
C5	**1**	1.3	294	2.1
C6	19	**11**	347	16.2
C7	29	**14**	191.8	21.1
C8	31	**15**	124.4	38
C9	17	11	185	**9.2**
C10	**4**	7.7	587	11.8
C11	**1**	3	587	2.3

Le tableau 5.1 montre que le modèle CycleGAN simplifié a une meilleure précision sur quasiment la totalité des dérivations ECG, à l'exception des colonnes (C6 à C9). Le score des modèles MLP et PIX2PIX reflète les résultats mentionnés précédemment. Les architectures LSTM et MLP donnent de bons scores MSE, même s'ils demeurent insuffisants en général pour surpasser notre modèle, de plus le temps d'exécution de notre modèle est beaucoup plus faible par rapport aux autres architectures. Cela peut représenter un atout pour traiter des jeux de données plus conséquents et plus complexes. Le CycleGAN simplifié reste la meilleure implémentation effectuée pendant le projet, et les scores obtenus montrent que l'architecture a un grand potentiel sachant aussi qu'elle peut être encore améliorée en utilisant par exemple plus de données pour l'apprentissage.

CHAPITRE VI

CONCLUSION

Ce travail avait trait à traduire un signal chronologique d'une forme de représentation à une autre. Il a permis de mettre en exergue l'utilisation des modèles génératifs antagonistes et leurs différentes variantes dans la résolution du problème. Lors de la réalisation de ce projet, nous avons présenté des travaux liés au traitement de séquences ainsi qu'aux modèles génératifs antagonistes, et nous nous sommes servis de ces études antérieures pour concevoir un réseau antagoniste génératif simplifié de type CycleGAN, adapté aux entrées constituées de signaux chronologiques.

Pour réaliser ce projet, nous avions à notre disposition une application concrète sur laquelle nous nous sommes appuyés pour mener à bien nos expérimentations et le développement de notre modèle. En effet, malgré une faible quantité de données à notre disposition, les signaux ECG fournis par la société SigNum inc ont été d'une grande utilité dans l'évaluation et la validation de notre architecture.

Ce projet nous aura montré que l'architecture neuronale CycleGAN est un atout efficient pour traiter la transcription de différents signaux d'une représentation à une autre. Il aura permis également à travers nos expériences, effectuées avec 32 enregistrements ECG acquis avec deux modalités de détections différentes, de montrer l'excellente performance du modèle CycleGAN décrit par rapport aux

alternatives de réseaux neuronaux expérimentées.

Le déploiement de cette méthode sur notre problème spécifique fut l'occasion de mettre en exergue l'utilisation de modalités de détections multiples pour surveiller d'autres paramètres physiologiques. Étant donné que les signaux obtenus peuvent tous être transcrits en une représentation standard. Nous prévoyons par la suite d'acquérir davantage de données ECG pour une évaluation plus décisive des performances du modèle proposé. Nous appliquerons aussi notre modèle à d'autres signaux chronologiques, puisque l'utilisation du CycleGan développé n'est pas exclusive à l'ECG.

APPENDICE A

PUBLICATIONS

Liste des publications scientifiques qui rentrent dans le cadre des travaux de recherche de cette thèse :

M. A. Abdelmadjid and M. Boukadoum, "Neural Network-Based Signal Translation with Application to the ECG," 2022 20th IEEE International New Circuits and Systems Conference (NEWCAS), June 2022, Quebec City, Canada, pp. 1-4.

APPENDICE B

RÉSULTATS ANNEXES

La figure B.1 représente 1000 lignes de la première colonne obtenues après la transformation. Le graphe de couleur orange représente le résultat de la transformation, quant au graphe bleu ce dernier représente le résultat visé.

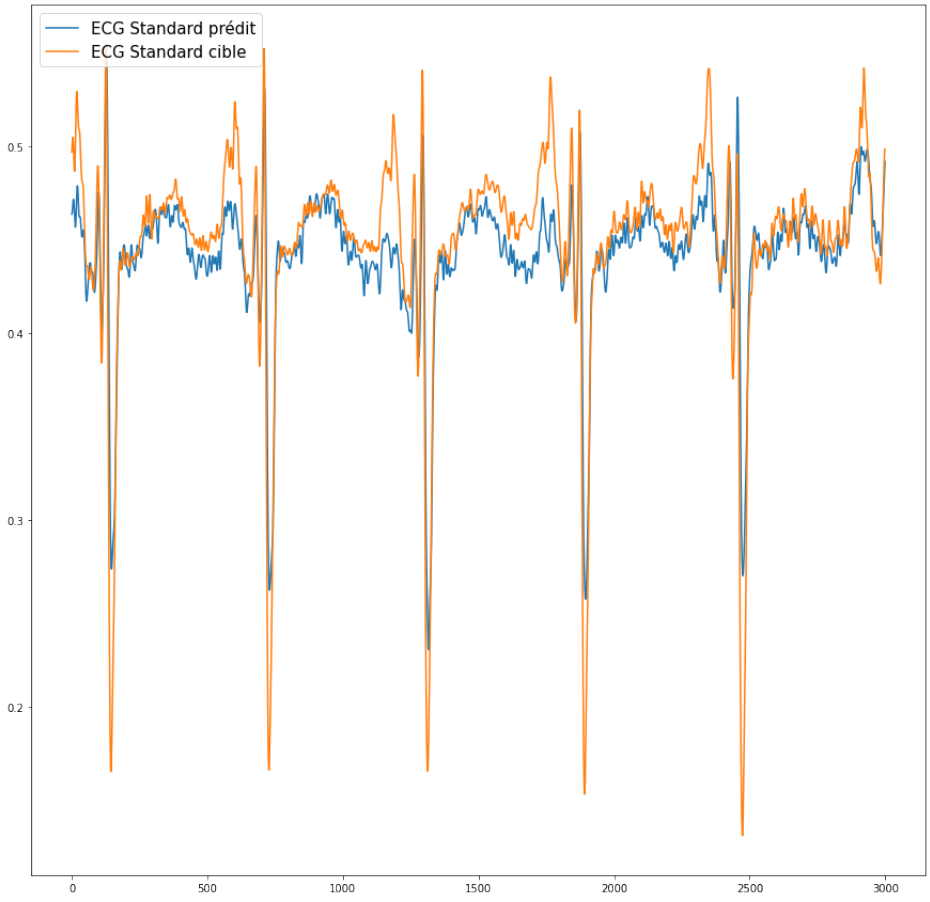

Figure B.1: Signaux du deuxième capteur (colonne 1)

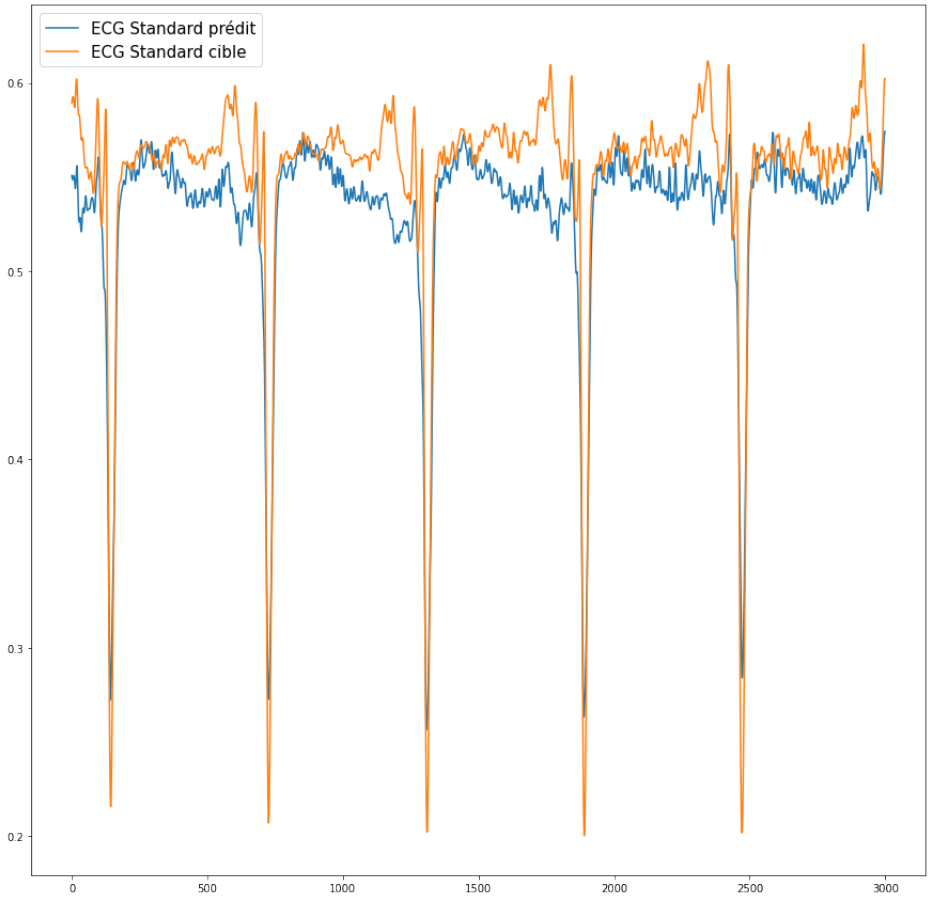

Figure B.2: Signaux du troisième capteur (colonne 2)

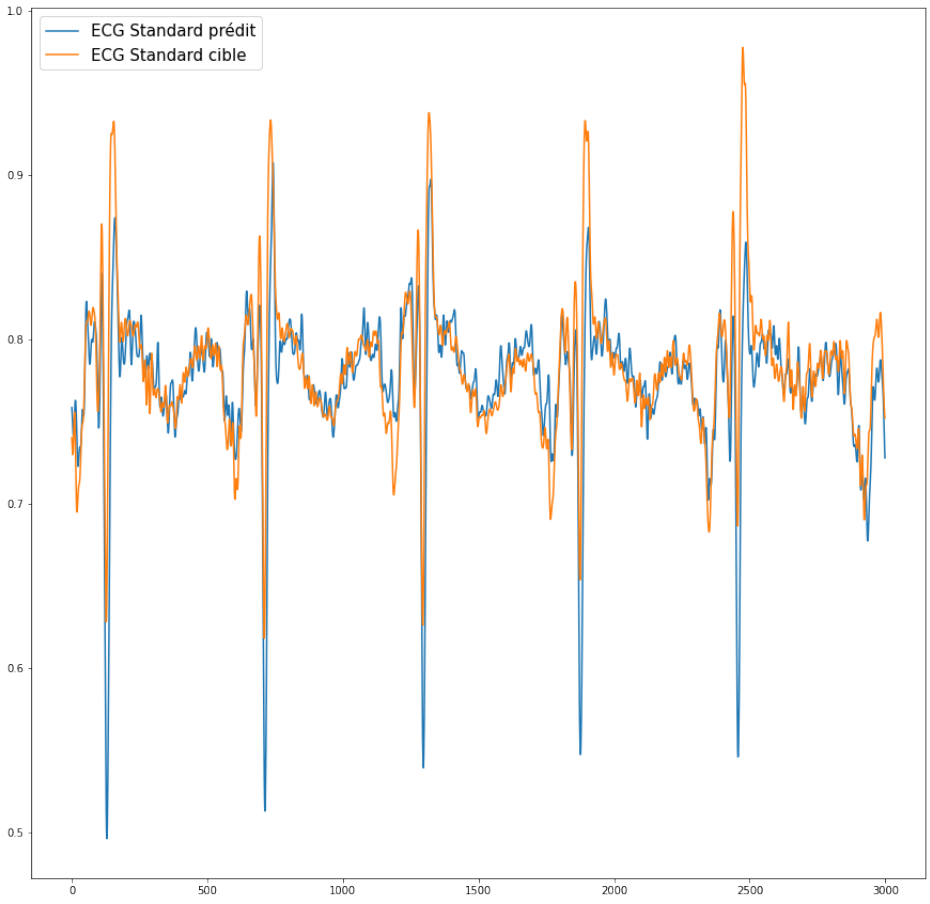

Figure B.3: Signaux du quatrième capteur (colonne 3)

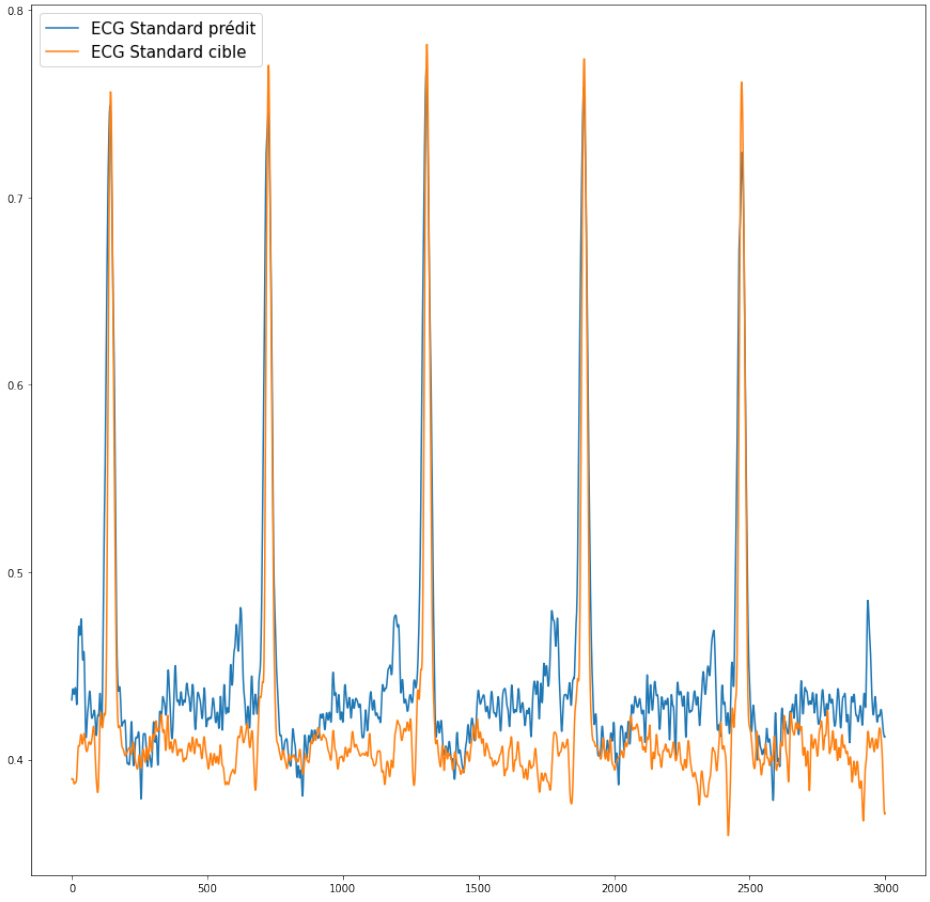

Figure B.4: Signaux du cinquième capteur (colonne 4)

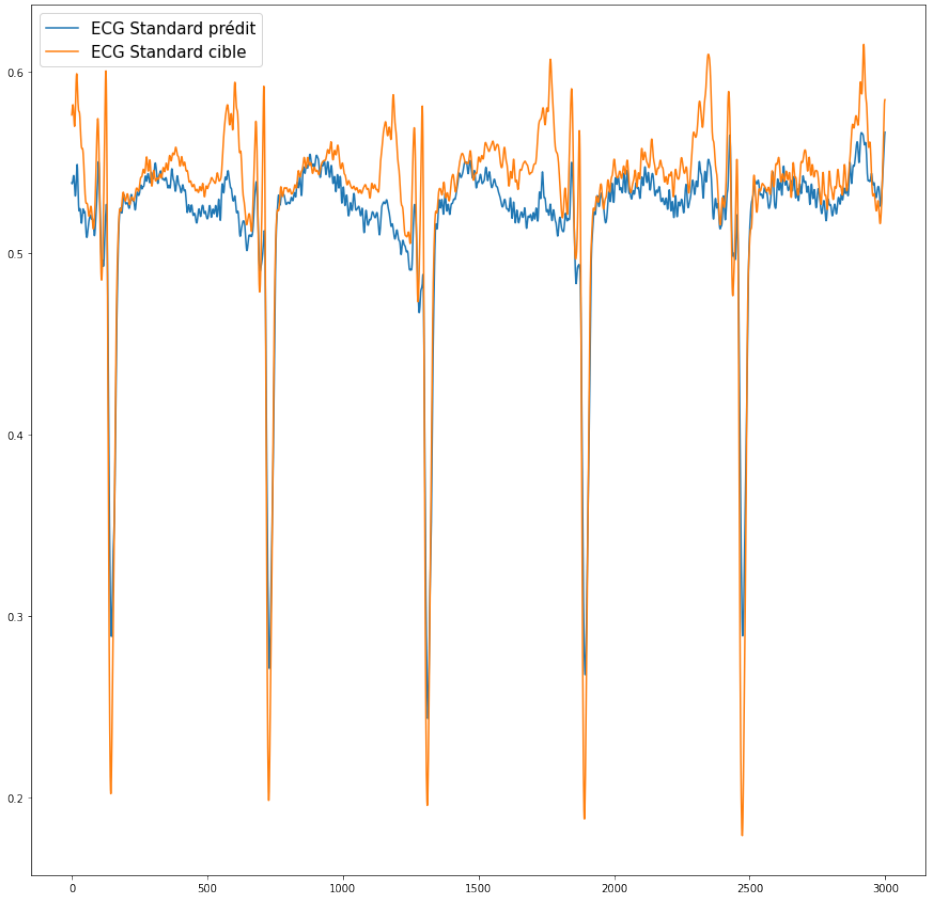

Figure B.5: Signaux sixième capteur (colonne 5)

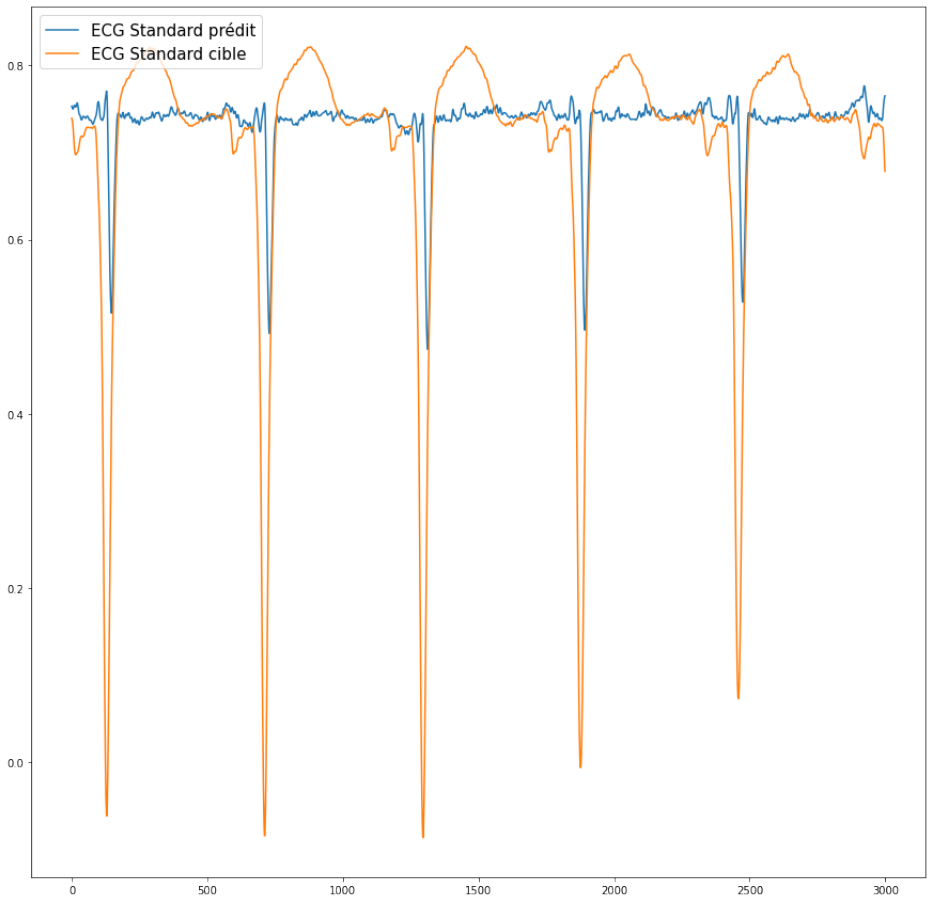

Figure B.6: Signaux septième capteur (colonne 6)

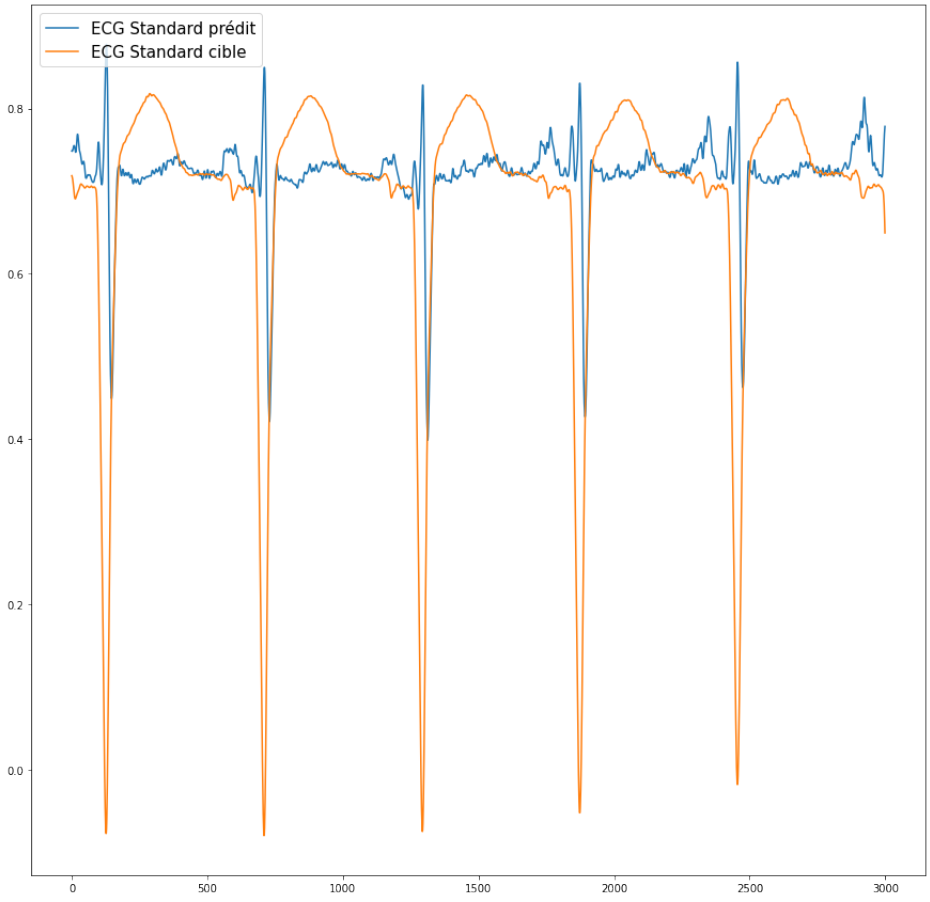

Figure B.7: Signaux huitième capteur (colonne 7)

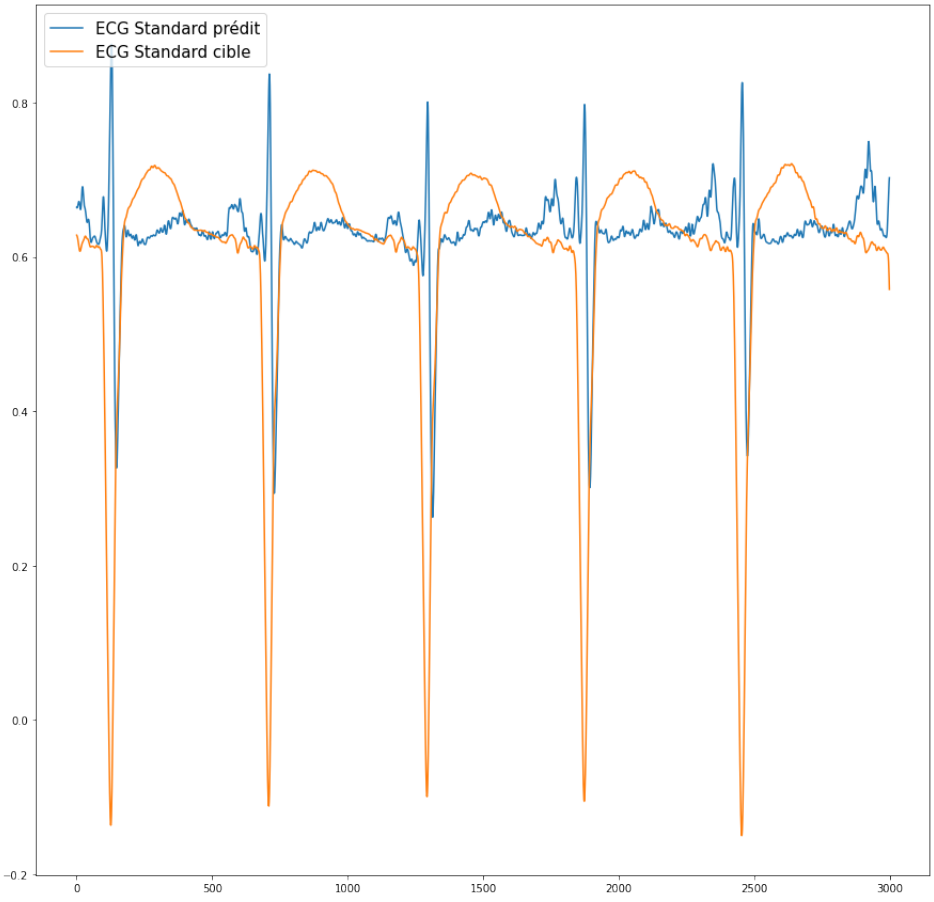

Figure B.8: Signaux neuvième capteur (colonne 8)

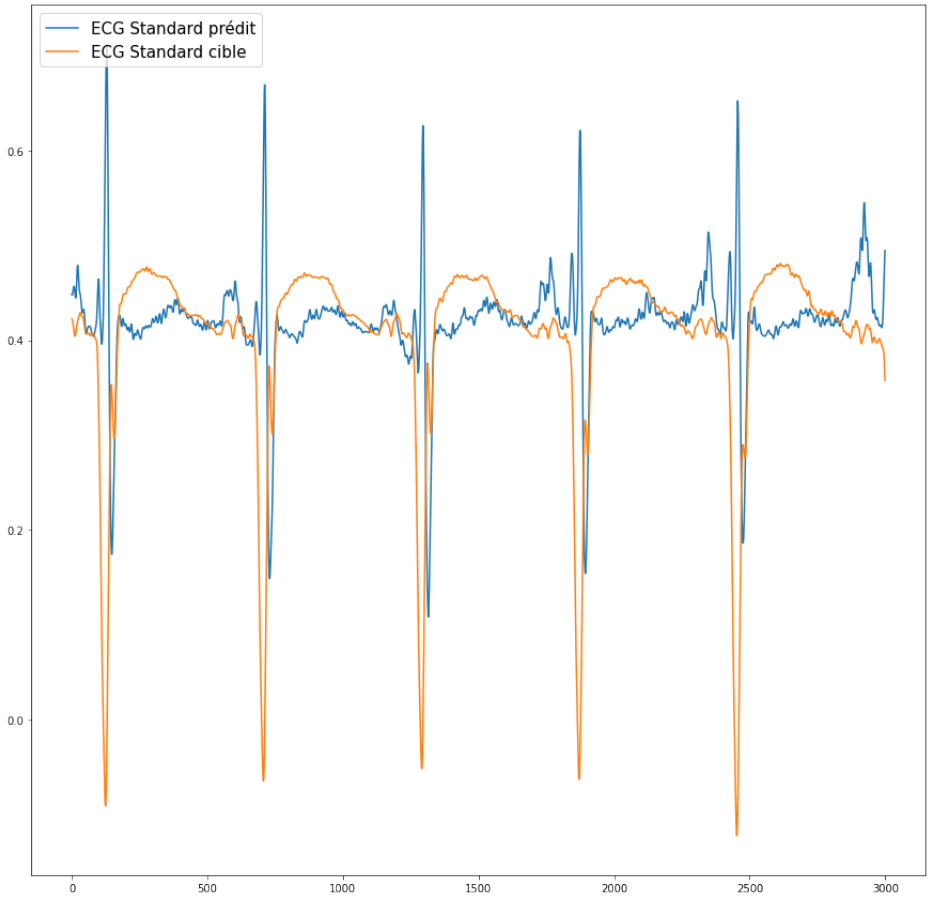

Figure B.9: Signaux dixième capteur (colonne 9)

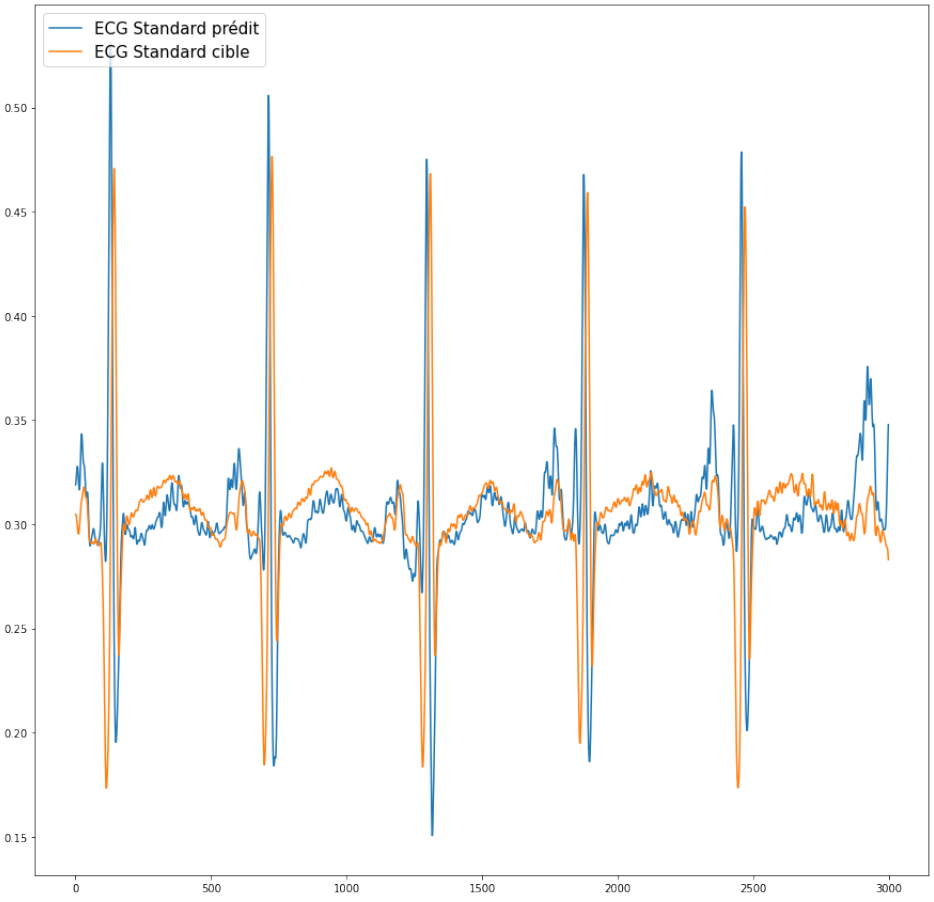

Figure B.10: Signaux onzième capteur (colonne 10)

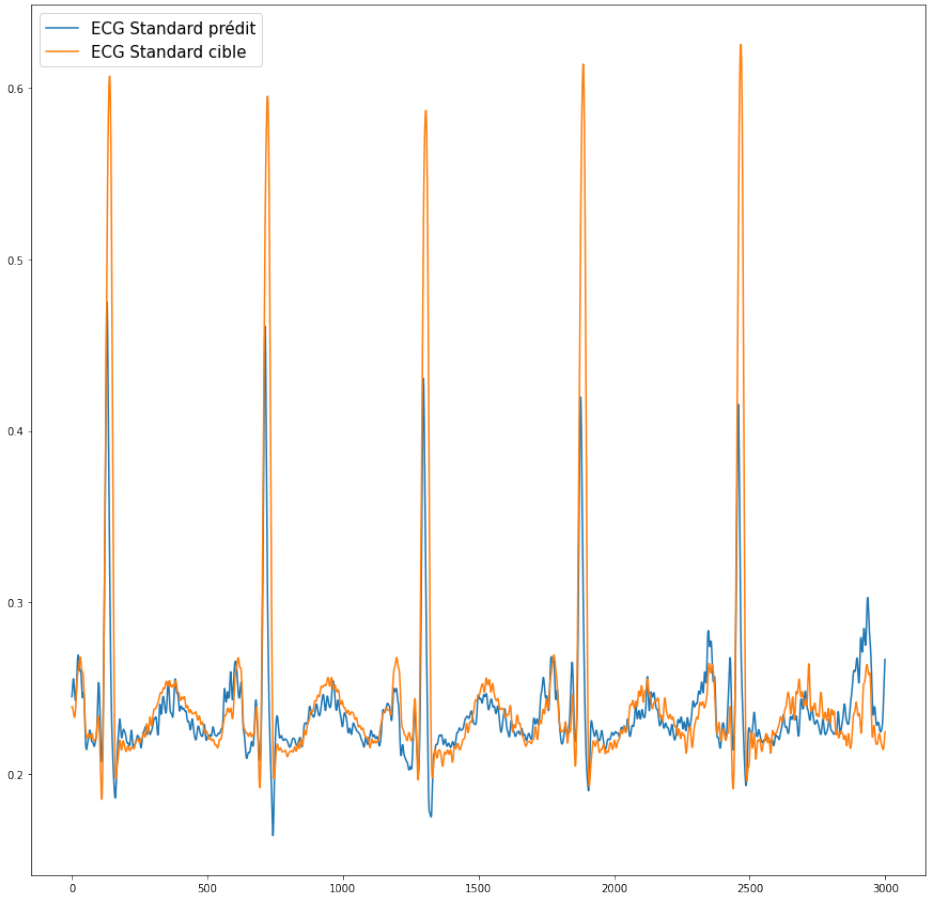

Figure B.11: Signaux douzième capteur (colonne 11)

RÉFÉRENCES

Albawi, S., Mohammed, T. A. et Al-Zawi, S. (2017). Understanding of a convolutional neural network. Dans *2017 International Conference on Engineering and Technology (ICET)*, 1–6. Ieee.

Arcelus, A., Sardar, M. et Mihailidis, A. (2013). Design of a capacitive ecg sensor for unobtrusive heart rate measurements. Dans *2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, 407–410. IEEE.

Arjovsky, M. et Bottou, L. (2017). Towards principled methods for training generative adversarial networks. *arXiv preprint arXiv :1701.04862*.

Arjovsky, M., Chintala, S. et Bottou, L. (2017). Wasserstein generative adversarial networks. Dans *International conference on machine learning*, 214–223. PMLR.

Bahdanau, D., Cho, K. et Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv :1409.0473*.

Bengio, Y. (2009). *Learning deep architectures for AI*. Now Publishers Inc.

Bermudez Romero, J. (2021). *Traduction automatique neuronale et littérature : Adaptation d'un système de traduction neuronal et analyse comparative de la traduction humaine et de la traduction automatique post-éditée*. (Thèse de doctorat). University of Geneva.

Brown, P. F., Cocke, J., Della Pietra, S. A., Della Pietra, V. J., Jelinek, F., Lafferty, J., Mercer, R. L. et Roossin, P. S. (1990). A statistical approach to machine translation. *Computational linguistics*, *16*(2), 79–85.

Brownlee, J. (2017). Stacked long short-term memory networks. Récupéré de `https://machinelearningmastery.com/`

Brownlee, J. (2019a). How to develop a 1d generative adversarial network from scratch in keras. Récupéré de `https://machinelearningmastery.com/`

Brownlee, J. (2019b). How to implement cyclegan models from scratch with keras. Récupéré de `https://machinelearningmastery.com/how-to-develop-cyclegan-models-from-scratch-with-keras/`

Ciregan, D., Meier, U. et Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. Dans *2012 IEEE conference on computer vision and pattern recognition*, 3642–3649. IEEE.

colah (2015). Understanding lstm networks. Récupéré de `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Dasiopoulou, S., Mezaris, V., Kompatsiaris, I., Papastathis, V.-K. et Strintzis, M. G. (2005). Knowledge-assisted semantic video object detection. *IEEE Transactions on Circuits and Systems for Video Technology*, *15*(10), 1210–1224.

Denton, E., Chintala, S., Szlam, A. et Fergus, R. (2015). Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv :1506.05751*.

Duarte, N., Llanso, E. et Loup, A. C. (2018). Mixed messages ? the limits of automated social media content analysis. Dans *FAT*, p. 106.

Dumoulin, V. et Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv :1603.07285*.

Ganokratanaa, T., Aramvith, S. et Sebe, N. (2020). Unsupervised anomaly detection and localization based on deep spatiotemporal translation network. *IEEE Access*, *8*, 50312–50329.

Ge, H., Xia, Y., Chen, X., Berry, R. et Wu, Y. (2018). Fictitious gan : Training gans with historical models. Dans *Proceedings of the European Conference on Computer Vision (ECCV)*, 119–134.

Godoy, D. (2018). Binary cross entropy.

Goodfellow, I., Bengio, Y., Courville, A. et Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. et Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv :1406.2661*.

Guillaume Genthial, Lucas Liu, B. O. K. R. (2019). Cs224n : Natural language processing with deep learning. *Stanford lecture*.

Halevy, A., Norvig, P. et Pereira, F. (2009). The unreasonable effectiveness of data. *IEEE Intelligent Systems*, *24*(2), 8–12.

Hansen, L. K. et Salamon, P. (1990). Neural network ensembles. *IEEE transactions on pattern analysis and machine intelligence*, *12*(10), 993–1001.

He, K., Zhang, X., Ren, S. et Sun, J. (2016). Deep residual learning for image recognition. Dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Heaven, W. D. (2020). Openai's new language generator gpt-3 is shockingly good—and completely mindless. *MIT Technology Review*.

Hochreiter, S. et Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Isola, P., Zhu, J.-Y., Zhou, T. et Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. Dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1125–1134.

Kalchbrenner, N., Grefenstette, E. et Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv :1404.2188*.

Kaneko, T. et Kameoka, H. (2018). Cyclegan-vc : Non-parallel voice conversion using cycle-consistent adversarial networks. Dans *2018 26th European Signal Processing Conference (EUSIPCO)*, 2100–2104. IEEE.

kazemnejad (2019). How to do deep learning research with absolutely no gpus. Récupéré de `https://kazemnejad.com/blog/`

Krizhevsky, A., Sutskever, I. et Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Lessard-Tremblay, M., Weeks, J., Morelli, L., Cowan, G., Gagnon, G. et Zednik, R. J. (2020). Contactless capacitive electrocardiography using hybrid flexible printed electrodes. *Sensors*, *20*(18), 5156.

Mahapatra, S. (2018). Why deep learning over traditional machine learning ? Récupéré de `https://towardsdatascience.com/`

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z. et Paul Smolley, S. (2017a). Least squares generative adversarial networks. Dans *Proceedings of the IEEE international conference on computer vision*, 2794–2802.

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z. et Paul Smolley, S. (2017b).

Least squares generative adversarial networks. Dans *Proceedings of the IEEE international conference on computer vision*, 2794–2802.

McCullum, N. (2020). Deep learning neural networks explained in plain english. Récupéré de `freecodecamp.org`

Menéndez, M., Pardo, J., Pardo, L. et Pardo, M. (1997). The jensen-shannon divergence. *Journal of the Franklin Institute*, *334*(2), 307–318.

Metz, L., Poole, B., Pfau, D. et Sohl-Dickstein, J. (2016). Unrolled generative adversarial networks. *arXiv preprint arXiv :1611.02163*.

Nayak, M. (2018). Deep Convolutional Generative Adversarial Networks(DCGANs) .

Och, F. (2006). Statistical machine translation live. Récupéré de `https://ai.googleblog.com/2006/04/statistical-machine-translation-live.html/`

Pang, X., Zhou, Y., Wang, P., Lin, W. et Chang, V. (2020). An innovative neural network approach for stock market prediction. *The Journal of Supercomputing*, *76*(3), 2098–2118.

Poibeau, T. (2017). *Machine translation*. MIT Press.

Poliakow, O. (2017). *Neural Machine Translation*. London : Machine Translation.

Radford, A., Metz, L. et Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv :1511.06434*.

Rashid, A., Do-Omri, A., Haidar, M., Liu, Q., Rezagholizadeh, M. *et al.* (2019). Bilingual-gan : A step towards parallel text generation. *arXiv preprint arXiv :1904.04742*.

Ronneberger, O., Fischer, P. et Brox, T. (2015). U-net : Convolutional networks for biomedical image segmentation. Dans *International Conference on Medical image computing and computer-assisted intervention*, 234–241. Springer.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv :1609.04747*.

Saagie (2020). Qu'est-ce que le natural language processing (nlp) ? Récupéré de `www.saagie.com`

Shastri, A. (2020). 3 neural network architectures you need to know for nlp ! Récupéré de `https://towardsdatascience.com/3-neural-network-architectures-you-need-to-know-for-nlp-5660f11281be`

Springenberg, J. T., Dosovitskiy, A., Brox, T. et Riedmiller, M. (2014). Striving for simplicity : The all convolutional net. *arXiv preprint arXiv :1412.6806*.

Sutskever, I., Vinyals, O. et Le, Q. V. (2014). Sequence to sequence learning with neural networks. Dans *Advances in neural information processing systems*, 3104–3112.

Suzuki, K. (2017). Overview of deep learning in medical imaging. *Radiological physics and technology*, *10*(3), 257–273.

TensorflowDocs (2021). Cyclegan tensorflow. Récupéré de `https://www.tensorflow.org/tutorials/generative/cyclegan/`

Tyagi, V. (2018). *Understanding digital image processing*. CRC Press.

Weng, L. (2017). From gan to wgan. *Lil'log GitHub*. Récupéré de `https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html`

Yang, S., Wang, Y. et Chu, X. (2020). A survey of deep learning techniques for neural machine translation. *arXiv preprint arXiv :2002.07526*.

Yu, D. et Deng, L. (2010). Deep learning and its applications to signal and information processing [exploratory dsp]. *IEEE Signal Processing Magazine*, *28*(1), 145–154.

Zhou, X. et Wang, W. Y. (2017). Mojitalk : Generating emotional responses at scale. *arXiv preprint arXiv :1711.04090*.

Zhu, J.-Y., Park, T., Isola, P. et Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. Dans *Proceedings of the IEEE international conference on computer vision*, 2223–2232.