MÉMOIRE DE MASTER

L'INTELLIGENCE ARTIFICIELLE ET LA CONCEPTION ARCHITECTURALE

Les réseaux de neuronnes antagonistes et génératifs conditionnés (cGAN) au service de la réalisation de modèles d'implantation de bâtiment dans un site

Etudiant:
POUHE Fahé Yacoub Abdallah

Encadré par:

GUENA François
SILVESTRE Joaquim
TÜSCHER Anne

Réalisé en Janvier 2023

Domaine d'étudue:
Concevoir et Construire l'Architecture (CCA)

Séminaire:
Savoirs des Activités de Projet Instrumentés (SAPI)

ensa paris la villette

REMERCIEMENTS

Je tiens à exprimer ma profonde gratitude envers mes parents, dont le soutien indéfectible est une source de motivation durant ces études d'architecture et plus précisément durant la réalisation de ce mémoire.

Je souhaite également adresser mes remerciements sincères à M. François GUENA pour son expertise et ses conseils précieux tout au long de l'élaboration de ce mémoire. Ses conseils et analyses avisés dans mon apprentissage ont grandement contribué à l'enrichissement de ce travail.

Mes remerciements vont également à M. Joaquim SILVESTRE, dont les discussions stimulantes ont éclairé ma réflexion et apporté des perspectives nouvelles sur le sujet. Son intérêt pour l'intelligence artificielle a été une source d'inspiration constante.

Enfin, je tiens à exprimer ma reconnaissance envers Mme Anne TÜSCHER pour sa guidance éclairée et son soutien constant. Ses conseils avisés ont été d'une grande importance dans la structuration et la rédaction de ce mémoire.

TABLES DES MATIERES

AVANT-PROPOS

A travers les pages de ce mémoire, je partage le cheminement de ma quête au croisement de l'architecture et de l'intelligence artificielle. Ce mémoire de master représente un témoignage honnête des explorations, des obstacles rencontrés, et des découvertes qui ont émaillé cette entreprise intellectuelle.

L'étude plonge dans l'univers complexe des réseaux de neurones et des algorithmes génératifs, se focalisant particulièrement sur les GAN.

Au fil de ces pages, j'invite le lecteur à suivre mon parcours, à travers des lignes qui dévoilent les différentes étapes que j'ai dû suivre dans cette recherche académique. Il s'agit également d'une aventure personnelle dans un domaine qui est totalement inconnu par la majorité des architectes mais qui est tout autant passionnant que l'exercice « classique » de l'architecture. L'intention ici n'est pas d'émerveiller, mais de partager les réflexions et les apprentissages issus de cette exploration, dans l'espoir que ces modestes contributions puissent éclairer d'autres esprits curieux.

INTRODUCTION

« Si on arrive à automatiser la réalisation des plans d'architecture, ça permettrait de gagner beaucoup de temps. Aurons-nous toujours besoin des architectes ? ». C'est la question que je me suis posé lors d'un stage dans un bureau d'étude technique (BET) lorsqu'un ingénieur en informatique qui m'observait dessiner le plan d'une maison, m'a posé la question suivante « Est-ce qu'on peut programmer le dessin du plan que tu es en train de faire pour qu'il se fasse tout seul ? ». Suite à cette question très pertinente, il s'en est suivis une discussion très enrichissante autour de la programmation informatique, le métier d'architecte, le temps de travail mais aussi les répercussions économiques que cela pourrait générer pour les architectes.
Ce moment d'échange très enrichissant qui nous a mené sur des sujets très larges tels que la politique, l'économie, l'efficacité au travail, la satisfaction émotionnelle après un travail acharné et bien fait m'a fait remettre beaucoup de choses en question et m'a donné envie d'explorer en profondeur cette thématique qu'est l'automatisation ou la programmation informatique appliquée à la conception en architecture.

Mon sujet de mémoire consiste donc à explorer cette thématique en mettant en concurrence deux algorithmes d'intelligence artificielle afin d'identifier laquelle est la mieux adaptée à la génération de propositions pertinentes qui feront office d'aide à la conception pour les architectes. Les deux types d'algorithme que nous mettrons en application sont un réseau de neurones antagonistes génératifs (GAN) et un cGAN (réseau de neurones antagonistes génératifs conditionnés) qui ont chacun des avantages et des inconvénients qu'il s'agira de tester pour en tirer le meilleur. Le but sera donc, entre autres, pour les architectes de pouvoir identifier à la suite de l'étude, vers quelle technologie se tourner s'ils en ont besoin.
Par ailleurs, je choisis d'utiliser ces deux types d'intelligence artificielle car ils sont de type génératif c'est-à-dire qu'ils sont capables de générer des propositions d'images à partir de d'autres images d'entrainement (Le mode de fonctionnement de ces algorithmes sera expliqué plus en détail dans la partie « état de l'art »). En effet, les images sont plus parlantes et intéressantes car elles se rapprochent de l'architecture à travers son moyen d'expression qu'est la représentation imagée à travers le dessin. Néanmoins, il convient de préciser que les expériences seront menées, dans la mesure du possible, dans le temps d'un mémoire de master. Je veillerai donc à restreindre le nombre de tests et de système d'évaluation des résultats de la recherche.

La présentation de cette étude débutera dans un premier temps, par l'état de l'art que je vais développer sous deux axes. D'une part, je présenterai le contexte historique de l'automatisation en architecture et les différents types d'intelligences artificielles notamment des CNN, des GAN, des cGAN, entre autres, leur mode de fonctionnement et leur utilité en Architecture. Et d'autre part, nous verrons l'état de la recherche c'est-à-dire l'ensemble des recherches menées sur mon thème de recherche afin d'en tirer le meilleur pour la mise en place de mon étude. Cela permettra de mettre en évidence les méthodes, les enjeux et les résultats des études menées préalablement sur cette thématique de recherche. Ainsi, il s'agira de présenter chacune des études menées en Architecture sur l'utilisation des intelligences artificielles notamment les GAN et les cGAN dans la génération de plans.

Dans un deuxième temps, nous verrons la problématique que je présenterai en deux sous-thèmes que sont les « questionnements » et la « méthode de recherche ». Les questionnements me permettront d'expliciter comment depuis mon intention de thème de départ et l'état de l'art, j'arrive à ma question de recherche. J'en profiterai donc pour expliciter pourquoi je compte réaliser l'étude avec précisément un GAN et un cGAN et comment elles s'influencent dans la réalisation de l'étude.
Enfin, la méthode de recherche sera l'occasion d'expliquer plus en détail la démarche scientifique que j'utiliserai afin de traiter le sujet de recherche, les différents critères d'évaluation des algorithmes et comment je les comparerai.

Dans une troisième phase, j'aborderai l'expérimentation de recherche de manière concrète. Cette phase débutera par la présentation des outils nécessaires à la bonne réalisation de cette étude et leur fonction, le processus de réalisation des expériences, la mise en place de la base de données d'entrainement, l'élaboration du GAN, du cGAN et enfin l'analyse des résultats de l'expérience.

En conclusion, il s'agira de montrer les conclusions générales que l'on peut tirer de l'étude, les limites de l'expérience, les pistes d'amélioration et des ouvertures vers d'autres sujets qui pourraient intéresser les futurs lecteurs de ce mémoire.

Partie 1 :
ETAT DE L'ART

I. Connaissances en intelligences artificielles

Avant de faire la présentation des différentes études menées sur l'intelligence artificielle appliquée à l'architecture, il me parait important de présenter l'évolution de l'automatisation ou la systématisation de l'architecture en passant par les différents algorithmes d'intelligence artificielles.

1. Evolution des techniques d'automatisation/systématisation

L'Architecture, domaine faisant intervenir différentes disciplines connexes, tend à s'automatiser au cours des cent (100) dernières années notamment depuis les années 1940 à nos jours. Dans l'ensemble, ce phénomène a débuté par la notion de la **modularité** consistant à créer un modèle d'architecture idéal et à le multiplier afin de reduire les temps de conception, de construction mais aussi afin de reduire les couts de construction. La modularité prend de l'ampleur avec le Bauhaus notamment grace à l'architecte Walter Gropius avec le concept de « Baukasten » et les projets de Le Corbusier à travers le principe du Modulor en 1945.

Suite à cette phase de modularité, nous avons eu le **DAO (Dessin Assisté par Ordinateur) en 1977** avec l'avènement du logiciel autoCAD. La révolution des logiciels CAD a permis aux architectes d'accélérer et simplifier le processus de dessin et de représentation. Grâce à ces outils numériques, notamment grâce à Catia, les possibilités de conception et créativité sont décuplées. L'architecte star Franck Gehry, a pu développer et mettre en évidence ses capacités architecturales grâce à ces outils de conception numérique.

Dans un troisième temps, nous avons **le paramétrisme depuis les années 1990** qui permet l'automatisation et la systématisation de la conception. Comme le dit Stanislas Chaillou lors d'une conférence chez Autodesk au sujet de l'architecture et les intelligences artificielles : « Le paramétrisme est un principe de synthétisation et de décomposition des différentes étapes nécessaires à la réalisation d'un design, d'une forme, d'un principe de conception. » Par ailleurs, des paramètres, des variables d'ajustement pourront être appliquées à chacune des étapes de ce processus afin d'influencer la forme, les caractéristiques compositionnelles du design final.

Et enfin, **l'essor de l'intelligence artificielle (IA) depuis les années 2010** avec la recherche sur les GAN en 2014 de I. Goodfellow ainsi que l'étude de S. Chaillou en 2020 sur la conception en architecture avec les GAN Pix2Pix.
Depuis, l'IA notamment les GAN intègrent de plus en plus l'architecture aussi bien lors de la conception que dans ses aspects techniques et réglementaires. Il me parait donc important de **s'initier à cette nouvelle technologie** afin d'en tirer le meilleur pour l'architecture.
Il convient notamment de préciser que ces grands mouvements de l'évolution de qui s'inter-influencent et qui ne sont donc pas en opposition.

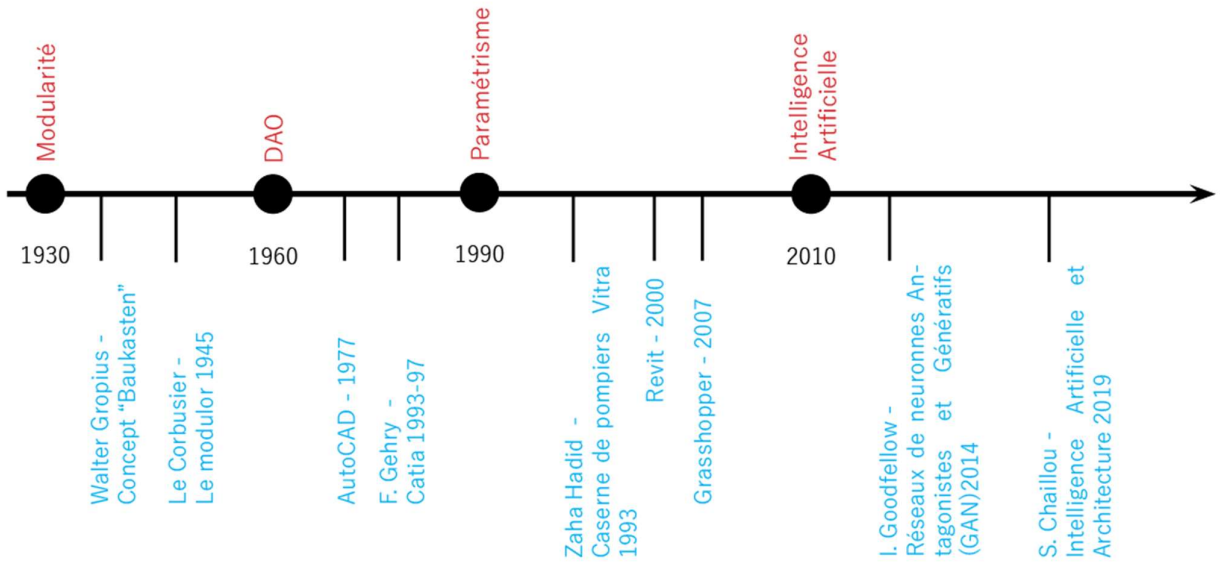


Figure 1: Grandes étapes des phénomènes d'automatisation ou de systématisation

Source : POUHE Fahé, Auteur

2. Algorithmes d'intelligence artificielle

a. Qu'est-ce qu'une intelligence artificielle ?

Le terme « intelligence artificielle », créé par John McCarthy, est souvent abrégé par le sigle « IA » (ou « AI » en anglais, pour artificial intelligence). McCarthy définit l'IA ainsi : « C'est la science et l'ingénierie de la fabrication de machines intelligentes, en particulier de programmes informatiques intelligents. Elle est liée à la tâche similaire qui consiste à utiliser des ordinateurs pour comprendre l'intelligence humaine, mais l'IA ne doit pas se limiter aux méthodes qui sont biologiquement observables. »

Elle est également définie par l'un de ses créateurs, Marvin Lee Minsky, comme « la construction de programmes informatiques qui s'adonnent à des tâches qui sont, pour l'instant, accomplies de façon plus satisfaisante par des êtres humains car elles demandent des processus mentaux de haut niveau tels que : l'apprentissage perceptuel, l'organisation de la mémoire et le raisonnement critique ». On y trouve donc le côté « artificiel » atteint par l'usage des ordinateurs ou de processus électroniques élaborés et le côté « intelligence » associé à son but d'imiter le comportement. Cette imitation peut se faire dans le raisonnement, par exemple dans les jeux ou la pratique des mathématiques, dans la compréhension des langues naturelles, dans la perception : visuelle (interprétation des images et des scènes), auditive (compréhension du langage parlé) ou par d'autres capteurs, dans la commande d'un robot dans un milieu inconnu ou hostile.

Même si elles respectent globalement la définition de Minsky, certaines définitions de l'IA varient sur deux points fondamentaux :

- les définitions qui lient l'IA à un aspect humain de l'intelligence, et celles qui la lient à un modèle idéal d'intelligence, non forcément humaine, nommée rationalité ;
- les définitions qui insistent sur le fait que l'IA a pour but d'avoir toutes les apparences de l'intelligence (humaine ou rationnelle), et celles qui insistent sur le fait que le fonctionnement interne du système d'IA doit ressembler également à celui de l'être humain et être au moins aussi rationnel.

Par ailleurs, les IA se divisent en famille notamment le machine learning et le deep-learning. Cette dernière faisant partie intégrante du machine learning. Nous nous concentrerons dans le cadre de ce mémoire sur le deep-learning. Le schéma ci-dessous permet de mieux cerner les familles.

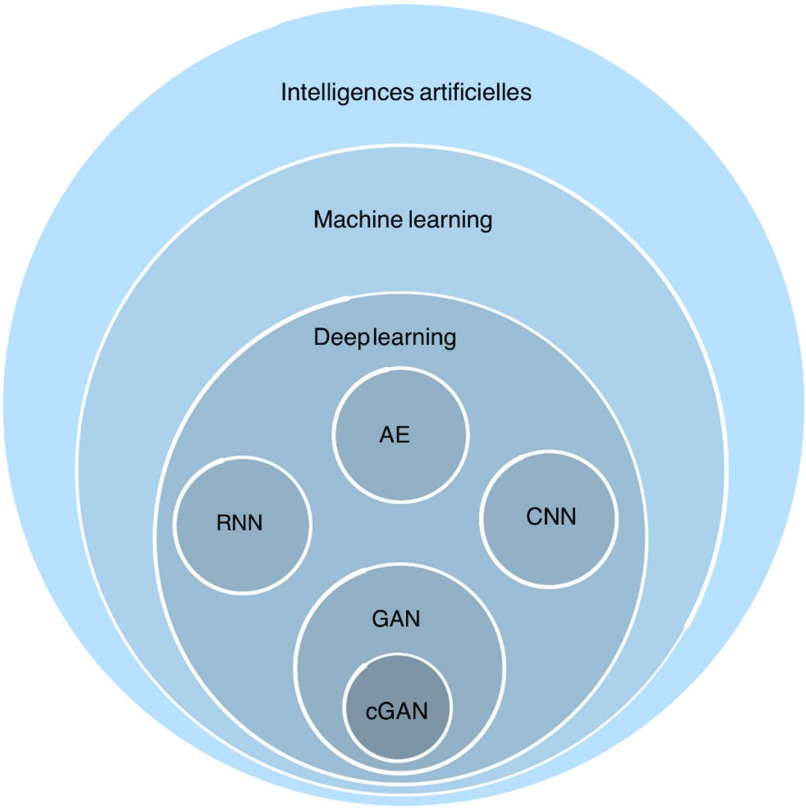


Figure 2: Familles et sous-familles des intelligences artificielles

Source : Auteur, POUHE Fahé

Le deep learning, ou apprentissage profond, est une branche de l'intelligence artificielle (IA) qui utilise des réseaux neuronaux artificiels pour résoudre des problèmes complexes. Les réseaux neuronaux artificiels sont des modèles mathématiques qui sont inspirés du fonctionnement du cerveau humain. Ils sont constitués d'une série de neurones artificiels, qui sont reliés entre eux par des synapses.

Cette branche de l'IA est une technique d'apprentissage automatique qui permet aux machines d'apprendre à partir de données. Les données sont utilisées pour entraîner les réseaux neuronaux artificiels, qui apprennent à reconnaître les modèles dans les données. Au fur et à mesure que les réseaux neuronaux artificiels s'entraînent, ils deviennent de plus en plus capables de résoudre les problèmes pour lesquels ils ont été entraînés.

Le deep learning est une technologie en plein développement qui a de nombreuses applications potentielles. Il est déjà utilisé dans des domaines tels que la reconnaissance d'images, la reconnaissance vocale, la traduction automatique, la détection des fraudes et la médecine.

❖ Principe de fonctionnement

Un réseau neuronal artificiel est un modèle mathématique qui est inspiré du fonctionnement du cerveau humain. Il est constitué d'une série de neurones artificiels, qui sont reliés entre eux par des synapses. Chaque neurone artificiel reçoit des entrées de ses neurones voisins, et calcule une sortie en fonction de ces entrées. La sortie du neurone est ensuite transmise à ses neurones voisins. Le réseau neuronal artificiel apprend à reconnaître des modèles dans les données en ajustant les poids des connexions entre ses neurones.

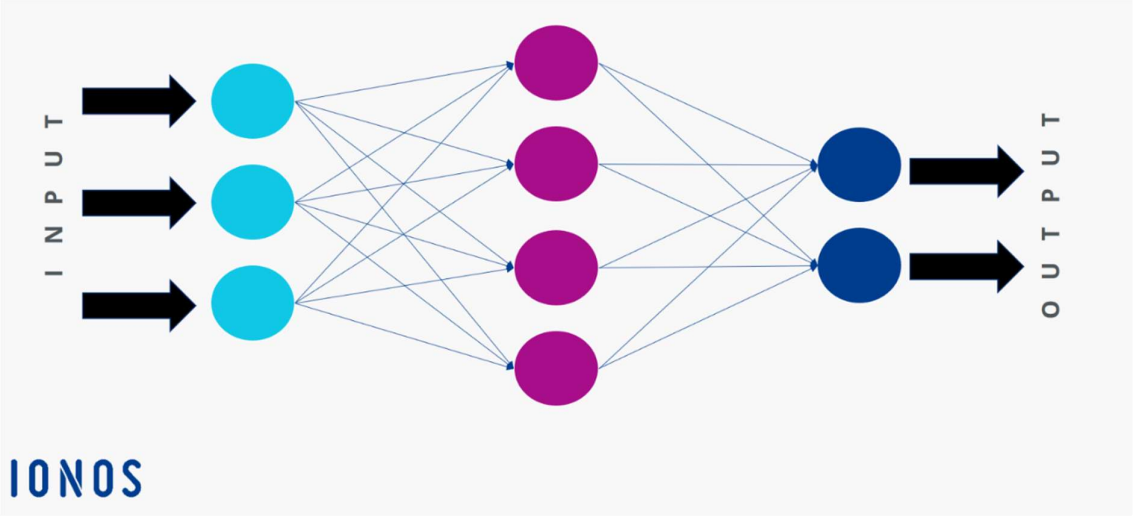


Figure 3: Structure d'un réseau de neurones

Sachant désormais quel est le mode de fonctionnement des intelligences artificielles du deep learning. Nous allons maintenant nous attarder sur les différents algorithmes que l'on retrouve dans cette famille de deep-learning.

b. Algorithmes de deep learning :

Il existe de nombreuses architectures de réseaux de neurones parmi lesquels les plus utilisés sont les CNN, les RNN, Les AE, les GAN et les cGAN.

❖ CNN soit Réseaux de neurones convolutifs :

Les réseaux de neurones convolutifs (CNN) sont un type de réseau de neurones artificiels qui est couramment utilisé pour la reconnaissance d'images et la vision par ordinateur. Ils sont efficaces pour reconnaître des patterns dans les images, tels que les bords et les formes.
Les CNN fonctionnent en appliquant une série de filtres aux images. Ces filtres sont conçus pour détecter des patterns spécifiques, tels que les bords horizontaux ou verticaux. Les sorties des filtres sont ensuite combinées pour générer une représentation de l'image.
Les CNN sont composés de plusieurs couches, chacune avec un ensemble de filtres. Les filtres de la première couche détectent les patterns les plus simples, tandis que les filtres des couches suivantes détectent des patterns plus complexes.

Exemples d'utilisation des CNN :

Les CNN sont utilisés dans de nombreuses applications de reconnaissance d'images, notamment :

- La reconnaissance faciale : les CNN sont utilisés pour identifier les personnes à partir de leurs visages.
- La détection d'objets : les CNN sont utilisés pour détecter des objets dans les images, tels que des voitures ou des personnes.
- La classification d'images : les CNN sont utilisés pour classer les images en catégories, telles que "chien" ou "chat".

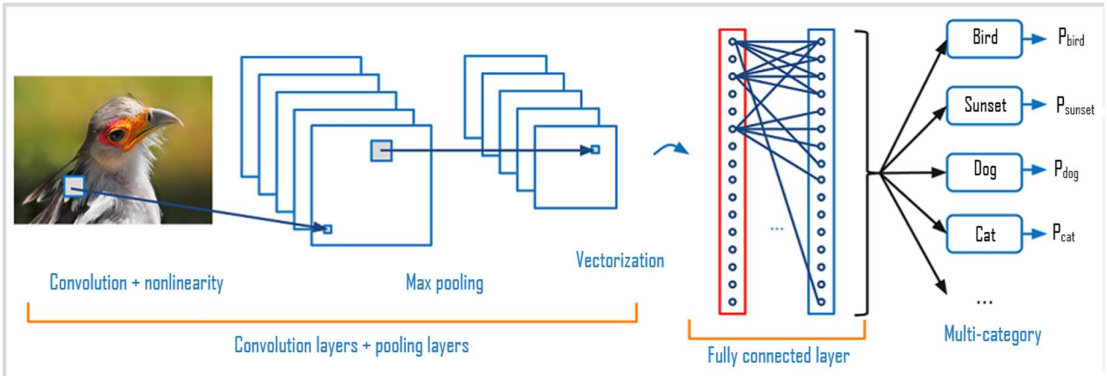


Figure 4: Principe de fonctionnement d'un CNN, ici pour la reconnaissance d'un animal

Source : Nada Belaidi, BLENT.AI

❖ Les RNN : Réseaux de neurones récurrents

Les réseaux de neurones récurrents (RNN) sont un type de réseau de neurones artificiels qui est couramment utilisé pour traiter des données séquentielles, telles que les séquences de mots ou de sons. Ils sont efficaces pour prendre en compte l'historique des données lorsqu'ils traitent une nouvelle donnée.

Les RNN ont des connexions entre les neurones qui s'étendent sur plusieurs couches. Cela leur permet de prendre en compte l'historique des données lorsqu'ils traitent une nouvelle donnée.

Par exemple, un RNN peut être utilisé pour traduire une phrase d'une langue à l'autre. Le RNN peut prendre en compte les mots précédents dans la phrase lorsqu'il essaie de traduire le mot actuel.

Les RNN sont utilisés dans de nombreuses applications qui nécessitent le traitement de données séquentielles, notamment :

- La reconnaissance vocale : les RNN sont utilisés pour convertir la parole en texte.
- La traduction automatique : les RNN sont utilisés pour traduire des langues.
- La génération de texte : les RNN sont utilisés pour générer du texte, comme des poèmes ou des histoires.

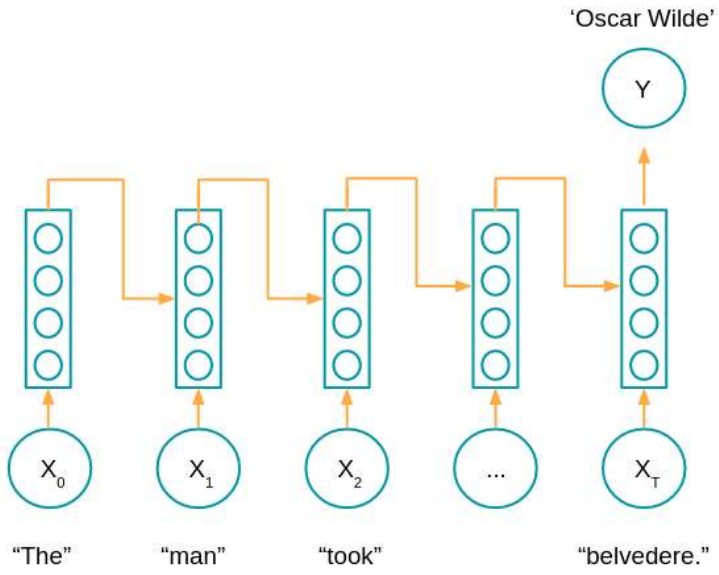


Figure 5: Principe de fonctionnement d'un RNN, ici appliqué à la reconnaissance de l'auteur de la phrase « The man took … belvedere ».

Source : Gaël Bonnardot, DATAKEEN.co

❖ AE, soit les Auto-Encodeurs

Les auto-encodeurs sont des modèles de machine learning qui apprennent à représenter les données en les compressant dans un format compact, puis en les décompressant pour les reconstruire. Le processus d'apprentissage consiste à entraîner le modèle à reproduire l'entrée aussi fidèlement que possible. L'auto-encodeur se compose de deux parties principales : l'encodeur, qui convertit les données d'entrée en une représentation comprimée, et le décodeur, qui reconstruit les données à partir de cette représentation.

Les auto-encodeurs sont utilisés dans de nombreuses applications, notamment :

- La réduction de dimension : les auto-encodeurs peuvent être utilisés pour réduire la taille des données sans perdre trop d'informations.
- La dénoising : les auto-encodeurs peuvent être utilisés pour supprimer le bruit des données.
- La génération de données : les auto-encodeurs peuvent être utilisés pour générer de nouvelles données similaires aux données d'origine.

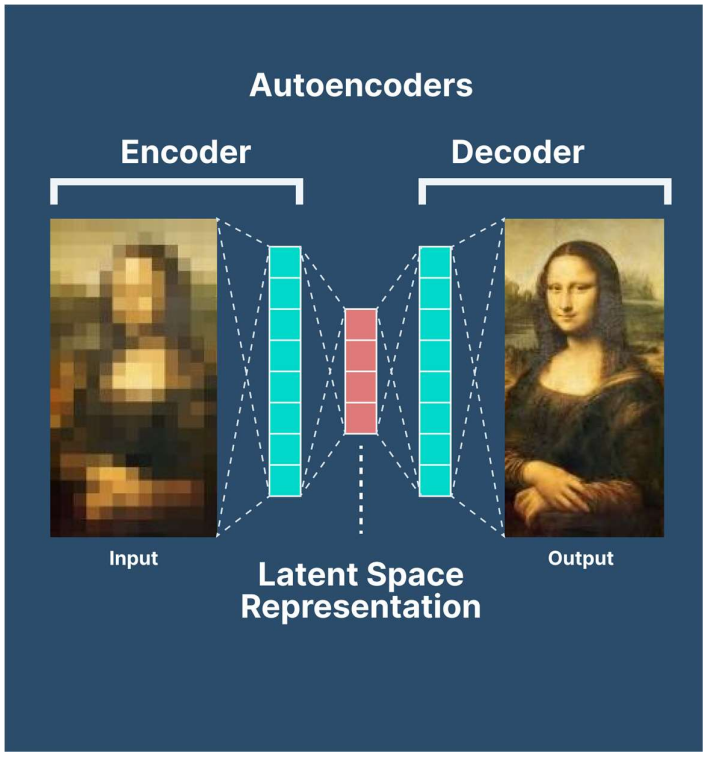


Figure 6: Principe de fonctionnement d'un AE, ici pour la reconstruction plus nette d'une image de la Joconde

Source : Hmrishav Bandyopadhyay, v7labs.com

Par ailleurs, Les différents types d'algorithmes étant décrit, nous allons maintenant nous interesser aux algorithmes qui sont l'objet de ce mémoire. Il s'agit des GAN mais aussi des cGANs, catégorie de GAN, soit des Conditionals GAN (GAN conditionnés).

c. GAN, réseau de neurones antagonistes génératifs

Les réseaux adversaires génératifs (GAN) sont une classe de cadres d'apprentissage automatique proposés à l'origine par Ian J. Goodfellow en 2014. Un GAN se compose de deux réseaux neuronaux en concurrence l'un avec l'autre, dans le but de créer de faux artefacts impossibles à distinguer des artefacts réels.

À partir d'un ensemble d'apprentissage, un GAN apprend à générer de nouvelles données présentant les mêmes caractéristiques sous-jacentes que l'ensemble d'apprentissage.

L'idée centrale d'un GAN est l'utilisation d'un "générateur" et d'un "discriminateur" pour faciliter la formation indirecte. Le générateur apprend à créer des échantillons de données plus réalistes, tandis que le discriminateur apprend à distinguer les vrais échantillons des faux créés par le générateur. Après des milliers de cycles de formation, le système devient très efficace pour générer de nouveaux échantillons très réalistes ressemblant à l'ensemble de données d'origine.

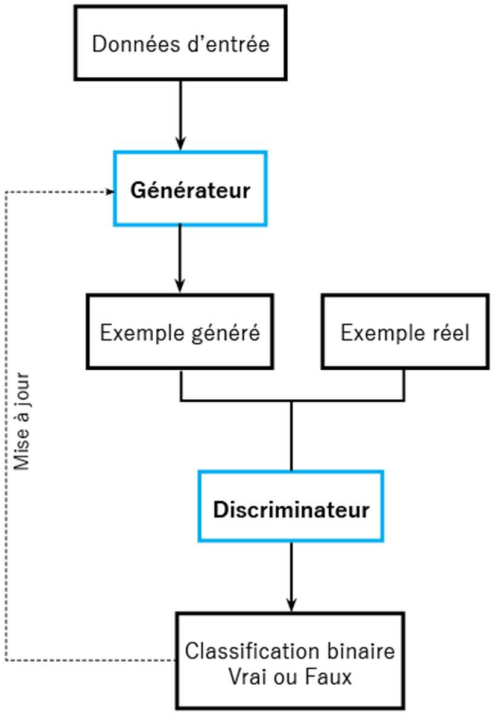


Figure 7: Mode de fonctionnement d'un GAN

Source : Auteur, POUHE Fahé

❖ À quoi peuvent servir les GAN ?

Les GAN peuvent être utilisés pour créer ou étendre des artefacts visuels dans une variété de cas d'utilisation. Les principales applications des GAN sont les suivantes :

- Augmentation des données - Le GAN peut être entraîné à générer de nouveaux échantillons d'images à partir de données existantes afin d'élargir un ensemble de données. Lorsque le GAN est arrivé à maturité, ses images de sortie peuvent être utilisées pour entraîner d'autres modèles de vision par ordinateur.
- Du texte à l'image - Le GAN est utilisé pour créer des bandes dessinées et des séquences vidéo en générant automatiquement des séquences d'images à partir d'un texte.
- Génération de visages – A l'aide d'une base de données à grande échelle de visages, les GAN permettent de créer des visages réalistes de personnes qui n'existent pas vraiment.
- Traduction d'image à image-Les GAN peuvent apprendre à mapper des motifs d'une image d'entrée à une image de sortie. Par exemple, ils peuvent être utilisés pour transformer une image dans un style artistique spécifique (transfert de style), pour vieillir l'image d'une personne ou pour de nombreuses autres transformations d'images.
- Conception industrielle et architecture - Les GAN peuvent être utilisés pour créer de nouvelles conceptions de produits en 3D à partir de produits existants. Par exemple, un GAN peut être entraîné à créer de nouveaux meubles ou à proposer de nouveaux styles architecturaux.

Figure 8: Visages générés par des GAN

Source : Jason Browlee, Machine Learning Mastery

Figure 9: Exemples de Text-to-Image d'images d'oiseaux générées à partir de textes

Source : Jason Browlee, Machine Learning Mastery

❖ cGAN

Nous allons enfin nous interesser aux cGAN qui sont des GAN mais plus spécifiques car ils permettent d'avoir un peu plus de contrôle sur les résultats générés par le GAN. Par exemple, supposons que nous avons utilisé un large éventail d'images de fleurs pour entraîner un GAN capable de produire de fausses images de fleurs. Avec un GAN classique, si nous voulons générer l'image d'une fleur au hasard, nous ne pouvons pas demander à l'algorithme de créer l'image d'une tulipe ou d'un tournesol, par exemple.

Le GAN conditionnel (cGAN) nous permet de conditionner le réseau avec des informations supplémentaires telles que les étiquettes de classe. Cela signifie qu'au cours de l'apprentissage, nous transmettons au réseau des images avec leurs étiquettes réelles (rose, tulipe, tournesol, etc.) pour qu'il apprenne à faire la différence entre elles. De cette manière, nous pouvons demander à notre modèle de générer des images de fleurs spécifiques.

Par ailleurs, tout comme le GAN, un cGAN est composé d'un générateur et d'un discriminateur qui jouent le même rôle que dans un GAN. A la différence qu'ils sont entraînés sur un ensemble de données d'images et de conditions. Le générateur essaie de créer des images qui correspondent à la condition, tandis que le discriminateur essaie de distinguer les images réelles des images générées.

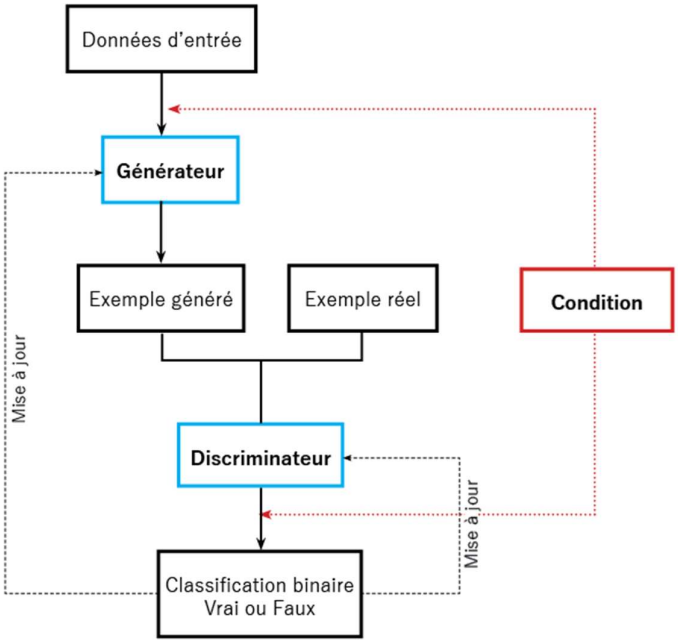


Figure 10: Mode de fonctionnement d'un cGAN

Source : Auteur, POUHE Fahé

En terme d'exemples, les cGAN ont des applications similaires aux GAN à la différence que les propositions générées sont modifiables selon les conditions insérées dans le programme.

II. Etat de l'art des recherches menées sur l'intelligence artificielle appliquées ou non à l'architecture

Il existe aujourd'hui plusieurs études qui traitent de la relation entre les IA et l'architecture. Bon nombre d'entre elles sont axées sur l'optimisation de techniques de construction, la recherche de forme, d'optimisation énergétique de façade mais aussi en faible proportion, l'optimisation de la conception de l'architecte grâce aux GAN. En effet, cette dernière branche est celle que je compte le plus explorer pour mon mémoire de master.

Même si certaines études existent au sujet de l'application de l'IA à l'architecture, de nombreuses autres existent au sujet des IA génératives appliquées à d'autres domaines tels que la médecine, le trafic routier, la génération de visages etc.

1. Recherches sur l'intelligence artificielle et l'architecture

A ce sujet, cinq études ont été réalisée successivement depuis les années 2016 jusqu'en 2020. En l'occurrence :

a. Stanislas Chaillou (2020) aborde l'IA et la conception en Architecture

En 2019, Stanislas Chaillou fait sa thèse sur la génération de plans d'appartements aménagés grâce aux GAN, à Harvard GSD, dans laquelle grâce aux GAN Pix2Pix, il entraine un réseau de neurones qui génère un plan d'étage d'immeuble d'habitation avec ses appartements aménagés et orientés selon la position des entrées et des ouvertures qu'il renseigne manuellement.
Ainsi, son objectif est de démontrer comment les réseaux neuronaux antagonistes génératifs (GAN) peuvent être utilisés pour concevoir des plans d'étage et des bâtiments entiers. L'étude propose une approche statistique de la conception architecturale, moins déterministe et plus holistique, où les machines extraient des qualités significatives et les miment tout au long du processus de conception. Cette approche statistique permettra donc de créer une pile de plans d'appartements en utilisant les modèles GAN Pix2Pix pour concevoir la disposition des bâtiments, la répartition des pièces et l'ameublement des espaces, tout en permettant une interaction constante entre l'utilisateur et les modèles d'IA. Cette dernière est passible grâce à une interface utilisateur intuitive qui facilite ce processus itératif, permettant aux utilisateurs de spécifier des contraintes et des paramètres pour générer des plans d'étage personnalisés.

Afin de mener à bien son étude, Chaillou met en place une méthode qui repose sur l'utilisation de modèles GAN Pix2Pix pour chaque étape du processus de conception. Le premier modèle génère des empreintes de bâtiments typiques à partir de données SIG de la ville de Boston. Le deuxième modèle gère la répartition des pièces de vie et la fenestration en fonction de l'empreinte de chaque unité de logement. Le troisième modèle traite l'ameublement des espaces en fonction de la répartition des pièces définie par le modèle précédent. Chaque modèle est formé indépendamment, permettant aux architectes d'intervenir et de perfectionner les résultats entre chaque étape.

Enfin, en mettant en œuvre cette pile de plans d'appartements à l'échelle d'un bâtiment entier, l'étude ouvre la voie à des applications potentielles à grande échelle dans la conception architecturale. Cependant, des limitations subsistent, telles que la gestion des murs porteurs dans les bâtiments à plusieurs étages et la transformation des sorties des modèles GAN en formats compatibles avec les outils et pratiques architecturaux courants. Malgré ces défis, l'étude montre le potentiel de l'IA pour transformer le processus de conception architecturale, offrant aux architectes de nouvelles possibilités pour repenser la manière dont les bâtiments sont conçus et construits dans le futur.

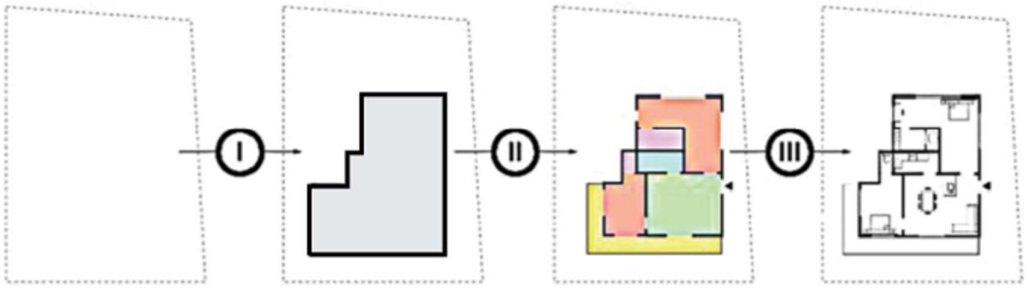


Figure 11: Grandes étapes de la génération de plans d'appartements aménagés à partir de l'empreinte du batis

Source : Stanislas Chaillou, ArchiGAN

Par ailleurs, sa thèse est la mise en relation de trois études précédentes faites dans la même école qui permettent de créer une chaine de conception depuis l'implantation jusqu'au plan d'appartement aménagé.
Les trois études de référence à celle de Chaillou sont les suivantes :

b. Zheng et Huang (2018) aborde la reconnaissance et la génération de plans

En 2018, Zheng et Huang proposent l'utilisation de l'IA pour reconnaître et générer des plans d'appartements via un GAN Pix2PixHD. Leur modèle de GAN traduit les images de plan d'étage en patchs colorés, puis ces couleurs en pièces dessinées. L'étude se focalise sur la reconnaissance et la génération de dessins architecturaux via l'apprentissage automatique.

L'objectif est d'appliquer les GAN, notamment le pix2pixHD, afin d'identifier les caractéristiques des plans et en générer de nouveaux, tout en comprenant leur fonctionnement et en visualisant leurs apprentissages.

Pour atteindre ces objectifs, les chercheurs mettent en place une méthode de recherche consistant à entraîner les réseaux de neurones à partir d'un ensemble de données composé de plans d'appartements en couleur annotés, puis à évaluer les performances des réseaux de neurones en reconnaissant et en générant de nouveaux plans. La première étape consiste à établir des principes d'étiquetage, attribuant différentes couleurs à des zones fonctionnelles spécifiques dans les plans architecturaux. Ensuite, deux réseaux de neurones sont entraînés : l'un pour reconnaître les plans en générant des cartes colorées annotées, et l'autre pour générer des dessins architecturaux à partir de cartes colorées annotées en entrée.

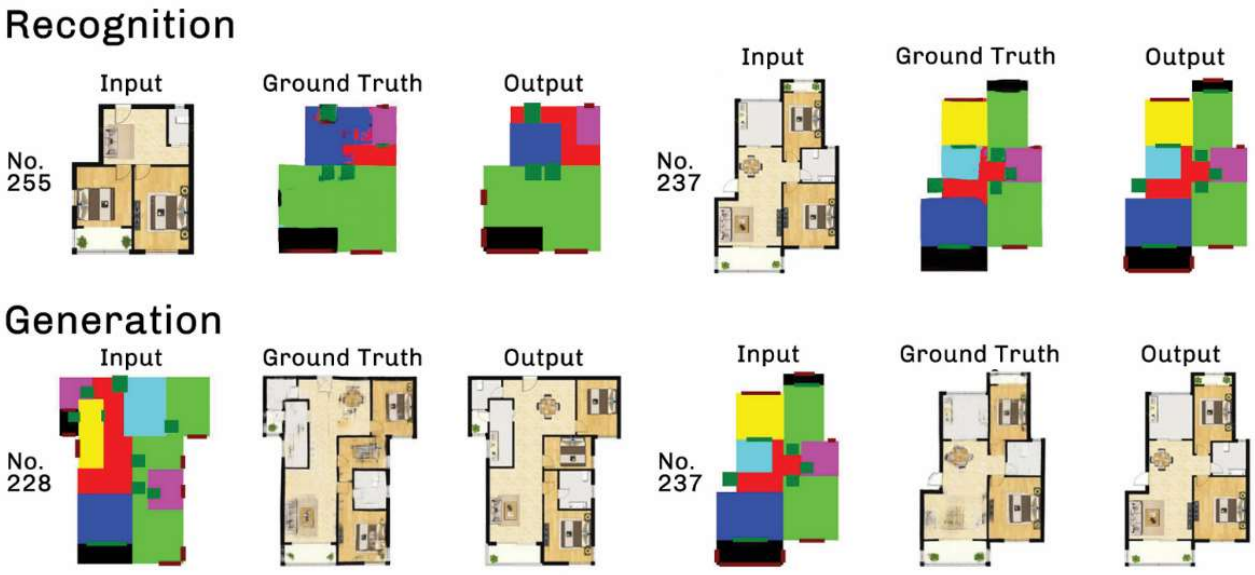


Figure 12: Résultats de reconnaissance (recognition) et de génération (generation) de plans d'appartements aménagés

Source : Auteurs de l'étude, Zheng et Huang (2018)

c. Nathan Peters (2017) aborde l'aménagement de pièce de vie

En 2017, Nathan Peters transforme une empreinte vide en taches de couleurs programmatiques sans l'indication d'une contrainte telle que la position des fenêtres. L'enjeux de son étude réside dans la création d'un système de conception participatif, cherchant à permettre aux utilisateurs de définir leurs propres espaces de vie. Inspiré par le travail de l'architecte Yona Friedman, Nathan Peters explore la possibilité de développer un cadre permettant à quiconque d'adapter son propre logement via Internet. Afin d'atteindre son objectif qui est de rendre la conception architecturale plus accessible et démocratique en permettant aux utilisateurs de participer activement à la conception de leurs espaces de vie. Pour se faire, il développe un algorithme d'apprentissage automatique pour générer des plans d'étage. En utilisant un réseau génératif antagoniste conditionnel Pix2Pix (Cgan – Pix2Pix), nommé YONA, Peters cherche à produire des images de plans d'étage classées à partir du contour d'un plan d'étage donné.

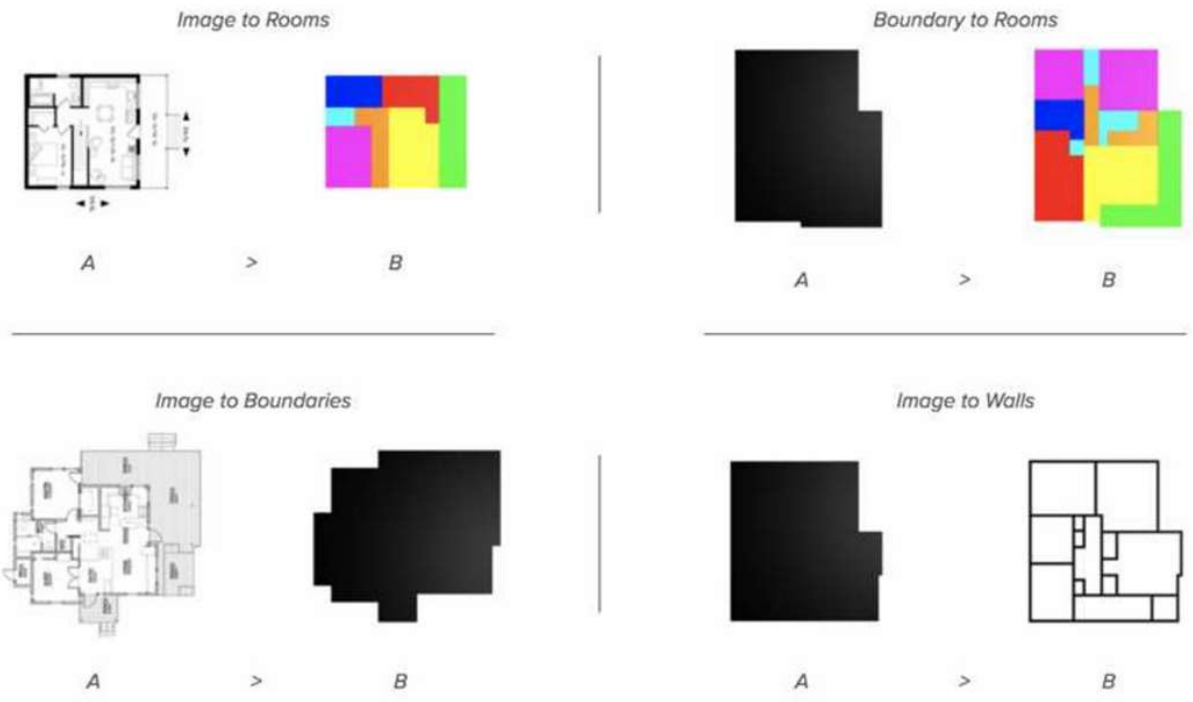


Figure 13: Exemples de résultats de l'étude

Source : Auteur de l'étude, Nathan Peters

d. Nono Martinez (2016) aborde le travail cumulé entre l'humain et la machine

Et enfin, en 2016, Nono Martinez faisait sa thèse sur l'idée d'une boucle de conception entre la machine et le concepteur lors du processus de conception, en particulier à travers le dessin suggestif. Alors que les logiciels de conception offrent des fonctionnalités avancées, l'introduction de l'intelligence artificielle dans le processus créatif ouvre de nouvelles perspectives, mais pose également des défis quant à la manière dont les machines peuvent interpréter et participer à la représentation du monde. L'idée est d'utiliser l'apprentissage automatique pour enseigner aux machines comment dessiner en observant des images réelles, afin qu'elles puissent participer activement au processus de création artistique, offrant des suggestions pour transformer, continuer, analyser ou rationaliser un dessin.

Afin de réaliser cette collaboration suggestive, grâce à un GAN Pix2Pix, des bots sont entraînés à reconnaître et à suggérer les éléments de dessin en fonction des entrées des utilisateurs.

Figure 14: Exemple de génération d'image (à droite) à partir d'un dessin (à gauche)

Source : Auteur de l'étude, Nono Martinez

e. Jean Raphael Piquard (2020) au sujet de l'intelligence artificielle et la génération de forme architecturale

En 2020, Jean-Raphaël Piquard réalise son mémoire de master sur l'étude de l'enrichissement d'un espace de solution paramétrique grâce aux GAN, à l'ENSAPLV. Et donc sur l'utilisation de l'apprentissage machine dans la conception architecturale. L'idée est de tirer parti de la capacité des GAN à produire des résultats inattendus et créatifs.

Ainsi, pour mener à bien cette quête, l'étude propose une approche expérimentale basée sur la génération de données paramétriques, leur alimentation dans des réseaux neuronaux antagonistes pour l'apprentissage, et l'analyse des résultats obtenus pour évaluer l'enrichissement de l'espace de solution. Pour se faire, l'on y distingue trois parties distinctes : une expérience témoin pour valider le processus, une exploration des formes paramétriques et une étude des dispositions spatiales. Chaque partie est conçue pour tester des critères spécifiques et évaluer l'impact des réseaux antagonistes génératifs sur l'enrichissement de l'espace de solution paramétrique.
A la fin de l'expérimentation, l'analyse critique des résultats permet de tirer des conclusions sur l'efficacité et le potentiel créatif de cette approche dans le domaine de la conception architecturale.

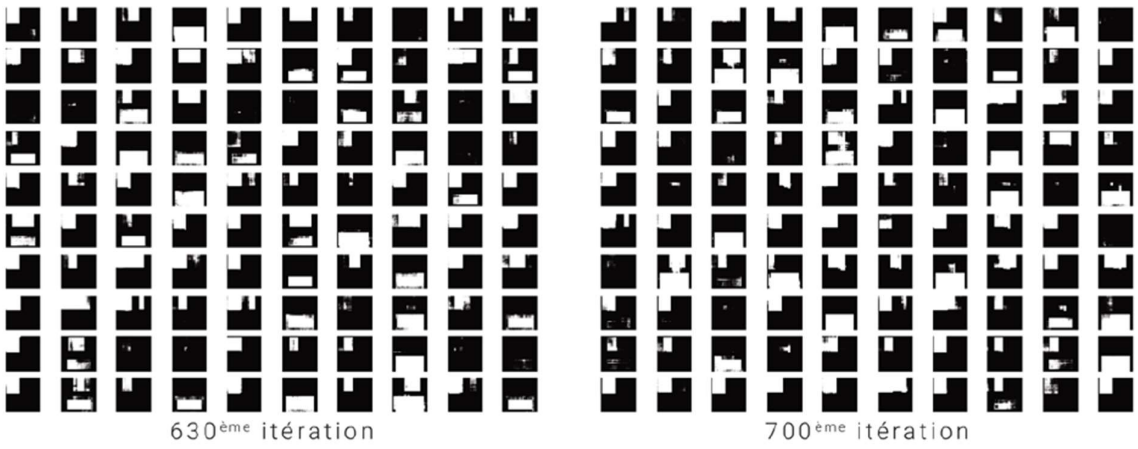


Figure 15: Itération du GAN de génération de génération de formes

Source : Jean-Raphaël Piquard, Mémoire de Master

2. Recherches menées sur l'intelligence artificielle en général

En dehors des études menées sur les intelligences artificielles appliquées en architecture, il convient de signifier que plusieurs autres études ont été réalisées au sujet de la mise en application des IA génératives telles que les GAN au service de différents domaines. Ces domaines sont très variés parmi lesquels nous avons la génération d'objets 3D, la médecine, les pandémies, le traitement d'images, le contrôle de trafic entre autres.

En effet, une étude ([1]) réalisée en avril 2021 par les chercheurs Alankrita AGGARWAL, Mamta MITTAL et Gopi BATTINENI permet de faire un récapitulatif des différentes études récentes réalisées au sujet des intelligences artificielles de type GAN et leurs applications dans les différents domaines cités précédemment. Le choix de ces études respecte un protocole scientifique afin d'identifier lesquels sont les plus pertinentes.

a. Méthode de sélection des études

❖ Stratégie de recherche

Afin de mener à bien cette étude, les auteurs mettent en œuvre une stratégie de recherche pour collecter les articles pertinents au sujet de l'application des GAN dans divers domaines. Ils ont suivi une approche de revue systématique de la littérature basée sur des études antérieures ([2]) datant de 2017 et 2020 aux sujets du « Rôle des attributs du comportement humain dans la détection des foules mobiles : une revue systématique de la littérature » et « Examen des cadres d'évaluation de la performance des projets d'administration en ligne ».

Trois bases de données (PubMed, EMBASE et Web of Science) ont été utilisées pour extraire les travaux pertinents, et cela, sur une période allant de 2016 à 2020. Les auteurs ont développé des stratégies de recherche pour identifier la littérature clé parmi les applications et les fonctionnalités des GAN. Les critères de sélection ont été appliqués pour choisir les articles appropriés, en se concentrant sur les applications récentes des GAN dans des domaines tels que la génération d'objets 3D, la médecine, le traitement d'images, la détection de visages, etc.

---

[1] Aggarwal, Alankrita, Mittal, Mamta et Battineni, Gopi. (2021). Generative adversarial network: An overview of theory and applications. *International Journal of Information Management Data Insights*, *1*(1), 100004. https://doi.org/10.1016/j.jjimei.2020.100004

[2] Agarwal, Neetima, Chauhan, Sumedha, Kar, Arpan Kumar et Goyal, Sandeep. (2017). Role of human behaviour attributes in mobile crowd sensing: a systematic literature review. *Digital Policy, Regulation and Governance*, *19*(2), 168-185. https://doi.org/10.1108/DPRG-05-2016-0023
Singh, Harjit, Grover, Purva, Kar, Arpan Kumar et Ilavarasan, P. Vigneswara. (2020). Review of performance assessment frameworks of e-government projects. *Transforming Government: People, Process and Policy*, *14*(1), 31-64. https://doi.org/10.1108/TG-02-2019-0011

Par la suite, les chercheurs évaluent l'abécédaire en parcourant les résumés et en rédigeant un récapitulatif des articles qu'ils considéraient comme qualifiés. À ce moment-là, les mots-clés de recherche sont examinés pour détecter les disparités avant de faire la sélection finale des dits mots-clés. Lorsque les auteurs ont choisi les articles qui devaient être retenus pour l'enquête, ils ont lu tous les articles pour rassembler des informations utiles à l'objectif final de sélection. Enfin, une fois les articles sélectionnés, les auteurs ont analysé indépendamment les données et ont comparé leurs opinions pour parvenir à un accord.

❖ Critères de sélection

Les résultats de la recherche documentaire ont permis d'obtenir 2084 articles classés par publications associées à l'objectif de cette étude. Plus précisément, dans les différentes bases de données ont été sélectionnées, 1141 articles de Scopus (Embase), 537 de WoS et 406 de PubMed.

Les trois types d'articles suivants ont été pris en compte : les articles originaux, les revues et les études analytiques.

Les articles sélectionnés ont été choisis selon les critères d'inclusion suivants et dans l'ordre :

- L'article sélectionné présente-t-il l'examen basé sur l'application des travaux du GAN ?
- La publication traite-t-elle des progrès du GAN dans les industries en temps réel ?
- L'article répond-il fidèlement à la question et aux objectifs de la recherche ?

Quant aux critères d'exclusion, les articles exclus ont été ceux qui respectaient les conditions suivantes :

- Les enregistrements de différentes caractéristiques gérés avec des critères d'inclusion,
- La langue (pas en anglais),
- Les études sans objectifs précis du GAN.

❖ Résultats

Suite à la première sélection des articles, 1783 articles qui n'étaient pas appropriés aux objectifs de cette enquête ont été éliminés pour les raisons connexes : 1757 articles ont été identifiés comme des doublons et 26 articles ne sont pas en anglais. Au cours de la deuxième phase de sélection, les 301 articles restants ont été distribués à parts égales aux auteurs pour une évaluation indépendante parmi lesquels, seulement 61 articles ont été sélectionnés.

Parmi ceux-ci, neuf (09) ont été rejetés parce qu'ils ne respectaient pas toutes les conditions énoncées précédemment. En fin de compte, **52 articles** sont pris en compte pour une analyse plus approfondie.

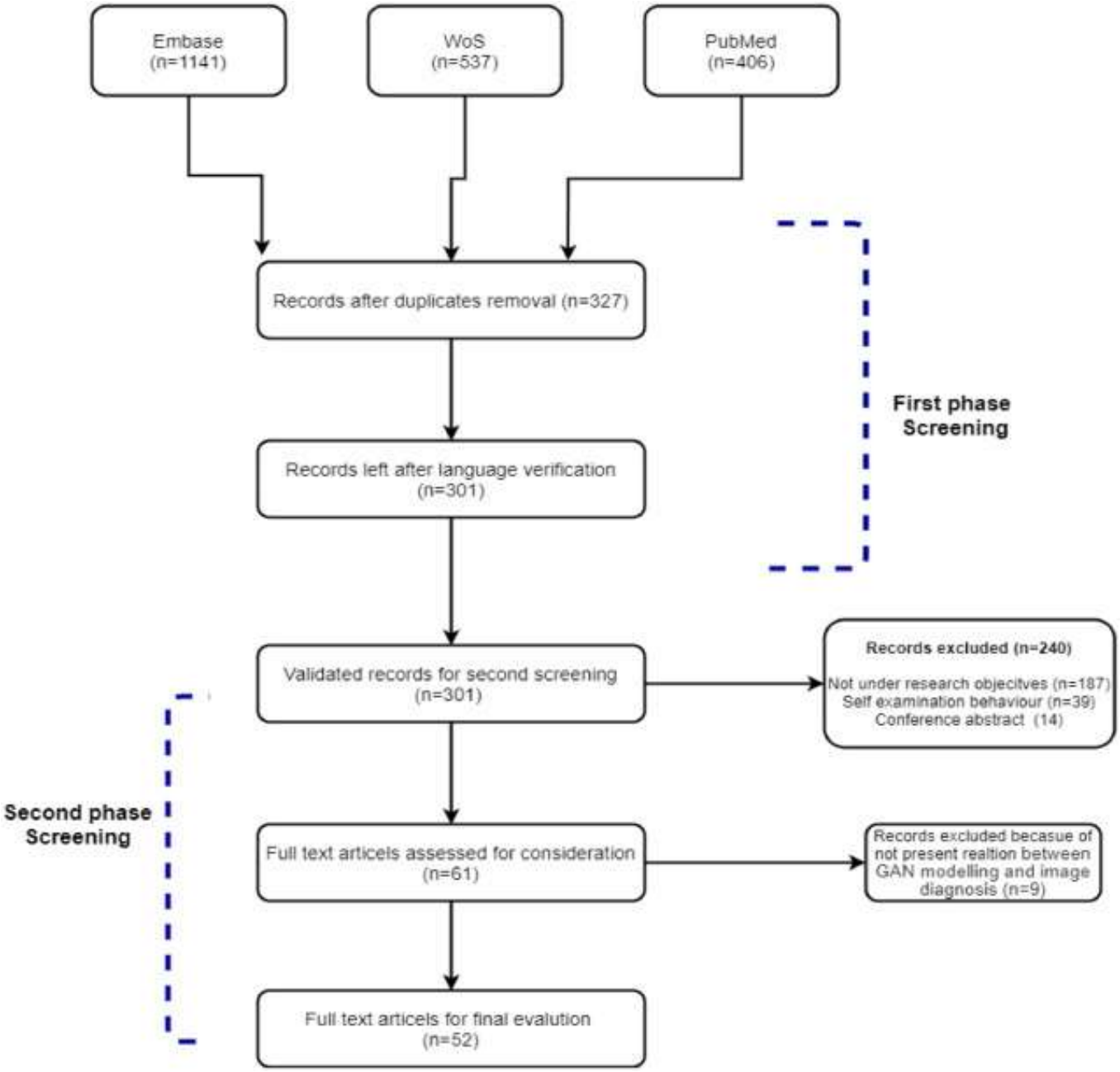


Figure 16: Protocole de sélection des études

Source : Auteurs de l'étude : Alankrita AGGARWAL, Mamta MITTAL et Gopi BATTINENI

b. Etudes sélectionnées par les chercheurs

Comme précisé dans la méthode, cinquante-deux (52) études ont été sélectionnées à la suite de l'étude. Chacune d'entre elles est catégorisée selon le domaine et la technologie utilisée. L'ensemble de ces études est à retrouver dans le tableau suivant, dans lequel est précisé l'application de chacune des technologies d'IA génératives.

Ainsi, afin d'approfondir cet état de l'art, au sujet des études menées sur l'IA générative durant les dernières années, je présenterai une étude (en gras dans le tableau) dans chacune des catégories mais l'ensemble des références de toutes les études sont à retrouver dans la bibliographie.

DOMAINES	*AUTEURS*	*ANNEE*	*MODELE*	*APPLICATION*
GENERATION D'OBJETS 3D	Yu Y.et autres	2020	Encodeur de point GAN	Traite les données non structurées sans étiquetage
	Y. Chen et autres	2018	3D-CNN	Créez des images nettes et de bonne qualité
	G. Ye et autres	2020	GAN basé sur le deep learning	Amélioration des images monochromatiques 2D
	Q. Ma et autres	**2020**	**Modèles 3D génératifs**	**Capture de mouvement humain**
	Y. Jin et autres	2020	Modèle GAN avec principe antagoniste à trois niveaux	Production d'objets 3D de haute qualité
MEDECINE	S. Baek et autres	2020	Modèle GAN et maillé	Production d'images IRM dans des pixels scellés
	Jain D. K. et autres	2020	GAN poser	Détection de mouvements humains
	A. Teramoto et autres	2020	Réseau de neurones convolutifs profonds (DCCN) avec GAN	Classer les images cytologiques
	M. D. Cirillo et autres	2020	Vox2Vox: 3D-GAN	Segmentation des tumeurs cérébrales
	H. C. Shin et autres	2018	GAN conventionnel	Identifier les images médicales

	J. Islam et Y. Zhang	**2020**	**GAN conventionnel**	**Génération d'images cérébrales**
	H. Lan et autres	2020	SC-GAN	Synthèse de neuroimages
	G. Zhaoa	2020	GAN conditionnel bayésien	Synthèse d'images cérébrales par IRM
	R. Oulbacha et autres	2020	Pseudo-3D Cycle GAN	Synthèse de la colonne lombaire par IRM et tomodensitométrie
	X. Zhang et autres	2020	Deform-GAN	Réduction du bruit dans les images médicales 3D
	D. Yang et autres	2019	Réseaux antagonistes d'image à image	Synthèse d'images médicales et segmentation sémantique
PANDEMIES	**Loey M. et autres**	**2020**	**GAN et apprentissage par transfert profond**	**Détection de la COVID-19 à l'aide d'images thoraciques**
	S. Albahli	2020	GAN avec le modèle de réseau de neurones profonds	Diagnostiquer la pneumonie de la maladie à coronavirus
TRAITEMENT D'IMAGES	C. Li et autres	2016	Markovian GAN	Générer une image 3D à partir d'une image 2D
	H. Zhou et autres	2020	Dual GAN	Récupération d'images haute résolution
	T. Go et autres	2020	GAN basé sur un réseau de neurones profonds	Effectuer une transformation d'image
	S. Zhang et autres	2020	GAN conventionnel	Débruitage d'image
	H. Tang et autres	**2020**	**GAN conventionnel**	**Génération de scènes guidées sémantiques**
DETECTION DE VISAGES	F. Mokhayeri et autres	2020	Un nouveau GAN contrôlable (C-GAN)	Synthèse de faces inter-domaines

	J. Zhao et autres	2019	Réseau antagoniste génératif à double agent (DA-GAN)	Reconnaissance faciale sans contrainte
	M. Kowalski et autres	**2020**	**GAN basé sur le deep learning**	**Génération d'images de visage**
	D. P. Jaiswal et autres	2020	GAN conventionnel	Animation de visages
TRANSFERT DE TEXTURE	**L. Sixt et autres**	**2019**	**GAN conventionnel**	**Génération de données étiquetées réalistes**
	R. Spick et autres	2020	3D-GAN	Générez une texture de haute qualité en ajoutant de la couleur
CONTROLE DE LA CIRCULATION	**D. Xu et autres**	**2020**	**GE-GAN**	**Estimation du trafic routier**
	Fathi-Kazerooni S. et autres	2020	GAN Tunnel	Détection d'images de trafic

Figure 17: Vue d'ensemble des études au sujet des GAN appliqués à différents domaines

Source : Auteurs de l'étude : Alankrita AGGARWAL, Mamta MITTAL et Gopi BATTINENI

❖ Etude au sujet de la génération d'objets 3D

Il s'agit d'une étude[3] qui met en œuvre un modèle 3D génératif. Le but de l'étude a été d'entrainer un algorithme d'IA générative capable d'habiller des corps humains scannés en 3D. Cela permet donc à l'issue de la recherche d'habiller des corps de différentes formes et dans différentes postures. En effet, les modèles de corps humain en trois dimensions sont largement utilisés dans l'analyse de la posture et du mouvement humain. Cependant, les modèles existants s'entrainent à partir de scans 3D de personnes peu vêtues. De plus, les modèles actuels manquent de puissance expressive nécessaire pour représenter la géométrie non linéaire complexe des formes de vêtements dépendant de la posture.

[3] Ma, Qianli, Yang, Jinlong, Ranjan, Anurag, Pujades, Sergi, Pons-Moll, Gerard, Tang, Siyu et Black, Michael J. (2020). Learning to Dress 3D People in Generative Clothing. Dans *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 6468-6477). IEEE. https://doi.org/10.1109/CVPR42600.2020.00650

Pour remédier à cela, lors de l'étude, ils entrainent un modèle de maillage 3D génératif de personnes habillées à partir de scans 3D avec différentes postures et vêtements. L'algorithme d'IA entrainé est un VAE-GAN de maillage conditionnel capable d'apprendre la déformation des vêtements à partir du modèle de corps SMPL. L'algorithme est conditionné à la fois par la posture et le type de vêtement, ce qui donne la capacité de dessiner des échantillons de vêtements pour habiller différentes formes de corps dans une variété de styles et de postures. Pour préserver les détails des plis, le VAE-GAN étend les discriminateurs par patchs aux maillages 3D. Ainsi, l'algorithme nommé CAPE, représente la forme globale et la fine structure locale, étendant efficacement le modèle de corps SMPL aux vêtements. Selon l'étude, il s'agit du premier modèle d'IA générative qui habille directement des maillages de corps humain en 3D et qui est aussi capable d'habiller différents corps dans différentes postures.

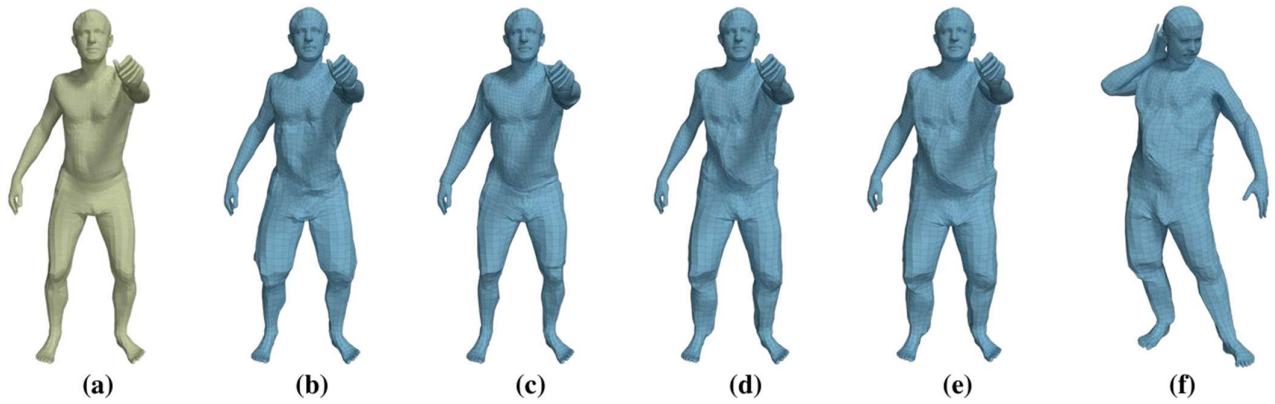


Figure 18: Modèle CAPE pour des humains habillés : (a) CAPE « habille » aléatoirement le maillage 3D, (b,c,d) Peut générer différents types de vêtements, les vêtements générés peuvent être généralisés à différentes formes de corps (e) et à différentes poses (f).

Source : Auteurs de l'étude : Ma Q, Yang J, Ranjan A, Pujades S, Pons-Moll G, Tang S, Black MJ.

❖ IA générative appliquée à la médecine

Dans cette catégorie, j'ai sélectionné l'étude [4]de J. Islam, en Y. Zhang réalisée en 2020 au sujet de la génération d'images cérébrales afin de faciliter la détection de la maladie d'Alzheimer chez des patients. En effet, cette étude aborde les enjeux majeurs liés à l'analyse d'images médicales, en mettant en lumière la difficulté d'obtenir des ensembles de données annotées et de qualité pour entraîner des modèles d'IA de diagnostic automatique des maladies. Avec un accent particulier sur la maladie d'Alzheimer, la recherche souligne les défis associés à la disponibilité limitée et à la qualité des données médicales, ainsi qu'aux coûts élevés et aux préoccupations liées à la confidentialité des données des patients. Ces obstacles entravent le développement de systèmes de diagnostic assistés par l'IA, ce qui rend crucial l'exploration de nouvelles approches pour générer des données synthétiques et surmonter ces difficultés.

De ce fait, le but de l'étude est de proposer une solution novatrice en utilisant les GANs afin de générer des images médicales synthétiques, en particulier des images tomographiques par émission de positons (PET), pour les différents stades de la maladie d'Alzheimer. Cette approche vise à pallier le manque de données annotées en produisant un ensemble de données synthétique, équilibré et représentatif pour l'entraînement des modèles d'IA. En utilisant les capacités de généralisation des GANs, l'étude cherche à créer des images réalistes et diversifiées, capturant les variations de la maladie à différents stades, ce qui pourrait améliorer la précision des diagnostics et permettre un traitement précoce et efficace.

Enfin, la méthode de recherche utilisée se compose de trois (03) étapes principales. Tout d'abord, une sélection minutieuse des données est effectuée, avec l'utilisation de 411 scans PET provenant de 479 patients collectés à partir de la base de données de l'Initiative de Neuroimagerie de la Maladie d'Alzheimer (ADNI). Ensuite, les GANs sont introduits pour générer des images synthétiques, avec une attention particulière portée aux Deep Convolutional Generative Adversarial Networks (DCGANs) pour leur capacité à produire des images de haute qualité et leur stabilité d'entraînement. En dernière étape, le modèle proposé pour la production d'images PET synthétiques repose sur l'utilisation des DCGANs, avec une mise en œuvre et un entraînement conformes aux directives de l'étude antérieure [5] de Radford A., Metz L. et Chintala S. réalisée en 2015.

---

[4] Islam, Jyoti et Zhang, Yanqing. (2020). GAN-based synthetic brain PET image generation. *Brain Informatics*, *7*(1), 3. https://doi.org/10.1186/s40708-020-00104-2

[5] Radford, Alec, Metz, Luke et Chintala, Soumith. (2016, 7 janvier). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks* (arXiv:1511.06434). arXiv. https://doi.org/10.48550/arXiv.1511.06434

Les résultats suivants ont été obtenus à la suite de l'étude :

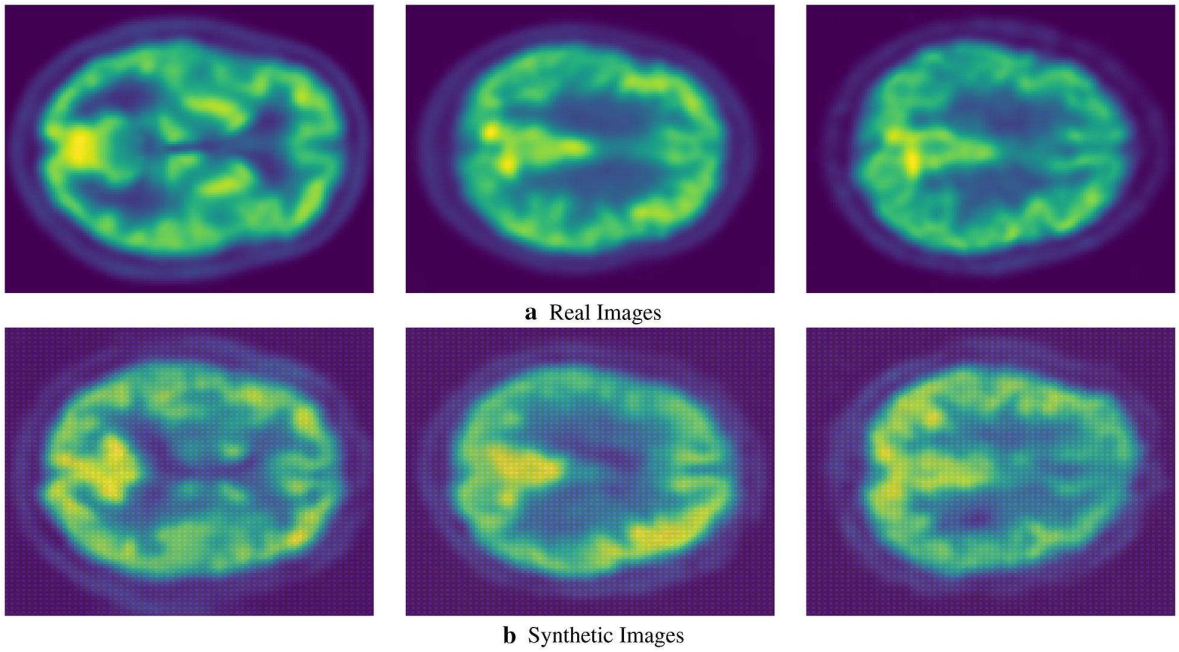


Figure 19: Images TEP cérébrales réelles et synthétiques d'un patient normal : **a** réel **b** synthétique

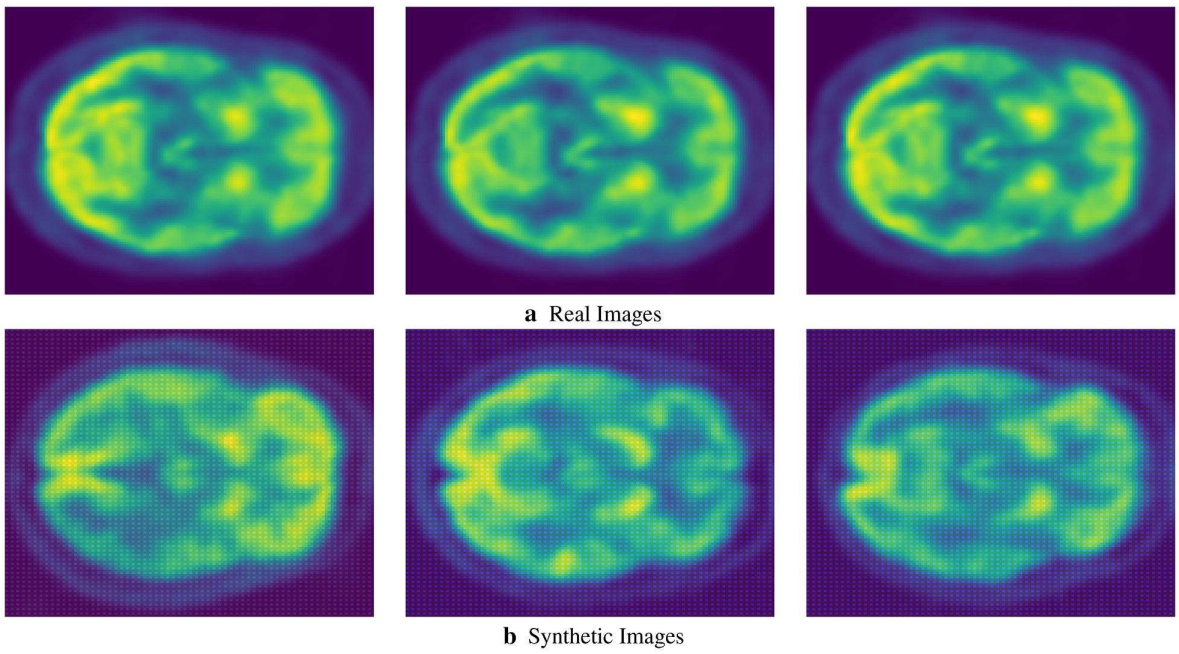


Figure 20: Images réelles et synthétiques de la TEP cérébrale d'un patient en déficience cognitive légère (DCL) : **a** réel **b** synthétique

Sources : Auteurs de l'étude : J. Islam, en Y. Zhang (2020)

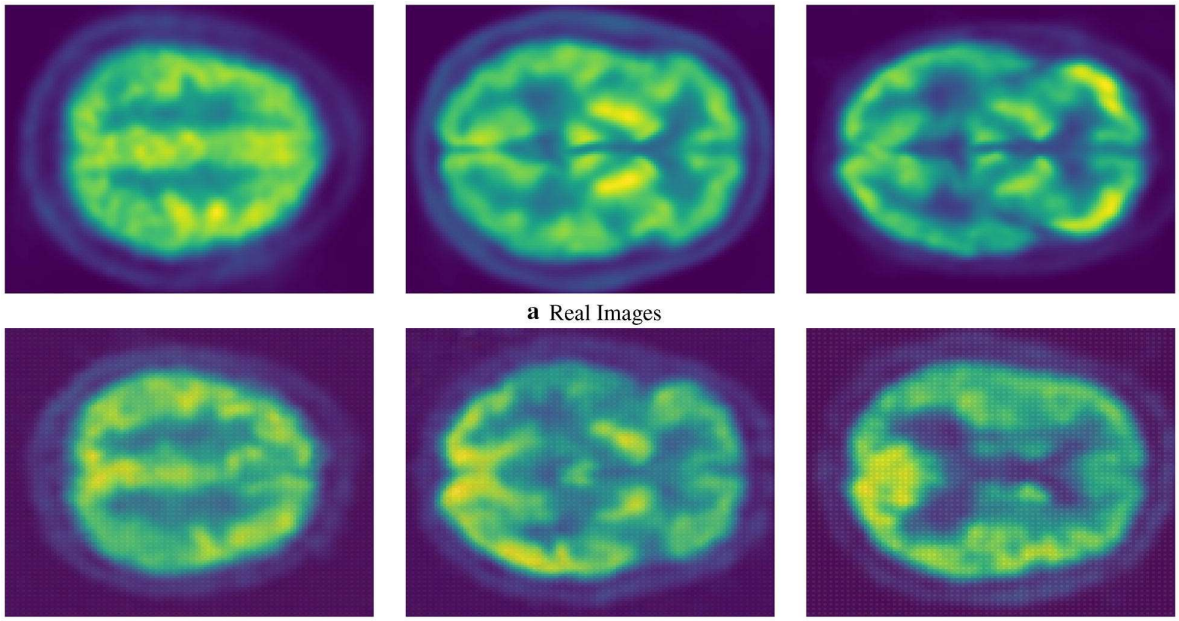


Figure 21: Images TEP cérébrales réelles et synthétiques d'un patient atteint de la maladie d'Alzheimer (AD) : **a** réel **b** synthétique

Source : Auteurs de l'étude : J. Islam, en Y. Zhang (2020)

❖ GAN au service de la détection de la maladie du COVID-19

L'étude[6] sélectionnée est celle de M. Loey, F. Smarandache, et N.E.M. Khalifa réalisée en 2020 au sujet de la détection du COVID-19, en l'absence d'un ensemble de données sur les radiographies pulmonaires de la COVID-19 grâce à un nouveau modèle de détection basé sur le GAN et l'apprentissage par transfert profond.

Les chercheurs proposent un modèle novateur basé sur les GAN et l'apprentissage profond pour la détection du coronavirus (COVID-19) dans les images radiographiques du thorax. Le manque de données sur le COVID-19, en particulier dans les images radiographiques thoraciques, constitue la principale motivation de cette étude scientifique. L'idée principale est donc de collecter toutes les images possibles du COVID-19 existant jusqu'à la rédaction de cette recherche et d'utiliser un algorithme de type GAN pour générer davantage d'images afin de faciliter la détection de ce virus. Cette détection est faite à partir des images radiographiques disponibles avec la plus grande précision possible. Ainsi, afin de mener à bien la recherche, trois (03) modèles d'IA de transfert profond sont sélectionnés pour l'enquête, à savoir l'Alexnet, le Googlenet et le Restnet18, en raison de leur faible complexité et de leur temps d'exécution réduit.

Pour se faire, l'étude vise à collecter toutes les images disponibles du COVID-19 et à utiliser le réseau GAN pour générer davantage d'images, afin d'enrichir le jeu de données et d'améliorer la précision de la détection. L'objectif est de développer un modèle robuste capable de distinguer le COVID-19 des autres affections pulmonaires et de poumons normaux avec la plus grande précision possible. Ces affections pulmonaires peuvent être des pneumonies bactériennes et des pneumonies virales.

Enfin, la méthode de recherche repose sur une collecte minutieuse de données à partir de différentes sources, suivie de l'entraînement et de l'évaluation de modèles d'IA de transfert profond sur le jeu de données constitué. L'étude explore également plusieurs scénarios de classification en utilisant différentes combinaisons de classes de maladies, afin d'évaluer les performances des modèles dans des contextes variés.

---

[6] Loey, Mohamed, Smarandache, Florentin et M. Khalifa, Nour Eldeen. (2020). Within the Lack of Chest COVID-19 X-ray Dataset: A Novel Detection Model Based on GAN and Deep Transfer Learning. *Symmetry*, *12*(4), 651. https://doi.org/10.3390/sym12040651

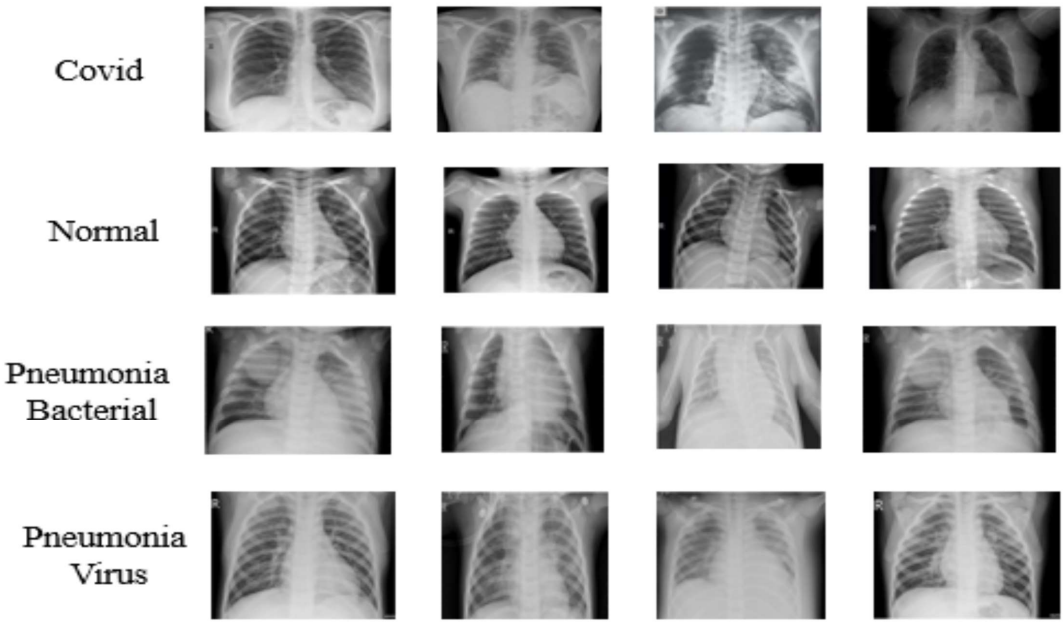


Figure 22: Exemples d'images utilisées dans cette recherche

Figure 23: Exemples d'images générées à l'aide de la structure GAN proposée

Sources : Auteurs de l'étude

❖ Etude au sujet de la création d'images de visages

Cette étude [7] CONFIG, intitulée "Controllable Neural Face Image Generation" réalisée par Kowalski et autres en 2020 vise à résoudre le défi de contrôler finement le processus de génération d'images de visages par des réseaux neuronaux. Bien que la capacité à générer des images réalistes ait progressé, la capacité à contrôler précisément des aspects spécifiques du processus générateur est en retard par rapport aux techniques de rendu graphique traditionnelles.

Son objectif à partir de cette technologie émergente est d'atteindre un niveau de contrôle similaire à celui des animations par ordinateur, sans sacrifier le réalisme inhérent aux images générées. L'étude présente ConfigNet, un modèle neuronal pour la génération d'images de visages, formé à la fois sur des images réelles et synthétiques. La méthode novatrice utilise des données synthétiques pour factoriser l'espace latent en éléments correspondant aux entrées d'un pipeline de rendu graphique traditionnel, permettant ainsi le contrôle indépendant de divers aspects du visage, tels que la pose de la tête, l'expression faciale, le style capillaire, l'éclairage, etc. Une évaluation combinant un réseau de détection d'attributs et une étude utilisateur démontre un contrôle individuel de pointe sur les attributs des images générées.

Dans cette optique de génération de visages modifiables, Kowalski et les autres chercheurs détaillent une méthode en trois étapes pour l'entraînement de ConfigNet. La première étape entraîne les sous-réseaux, à l'exception de l'encodeur des données réelles, avec une perte spécifique. Dans la deuxième étape, l'encodeur des données réelles est ajouté, et l'entraînement se poursuit pour améliorer la contrôlabilité des images générées. Et enfin, ConfigNet est soumis à des évaluations afin d'évaluer le niveau de réalisme et de contrôle que l'on obtient des résultats. Les résultats de l'étude permettent d'effectuer des avancées technologiques qui pourraient trouver des applications dans divers domaines, notamment la production d'animations et de rendus réalistes pour les personnages virtuels.

[7] Kowalski, Marek, Garbin, Stephan J., Estellers, Virginia, Baltrušaitis, Tadas, Johnson, Matthew et Shotton, Jamie. (2020). CONFIG: Controllable Neural Face Image Generation. Dans Andrea Vedaldi, Horst Bischof, Thomas Brox et Jan-Michael Frahm (dir.), *Computer Vision – ECCV 2020* (p. 299-315). Springer International Publishing. https://doi.org/10.1007/978-3-030-58621-8_18

Figure 24: Exemples de données d'entrainement synthétiques (à gauche) et données d'entrainement réels (à droite)

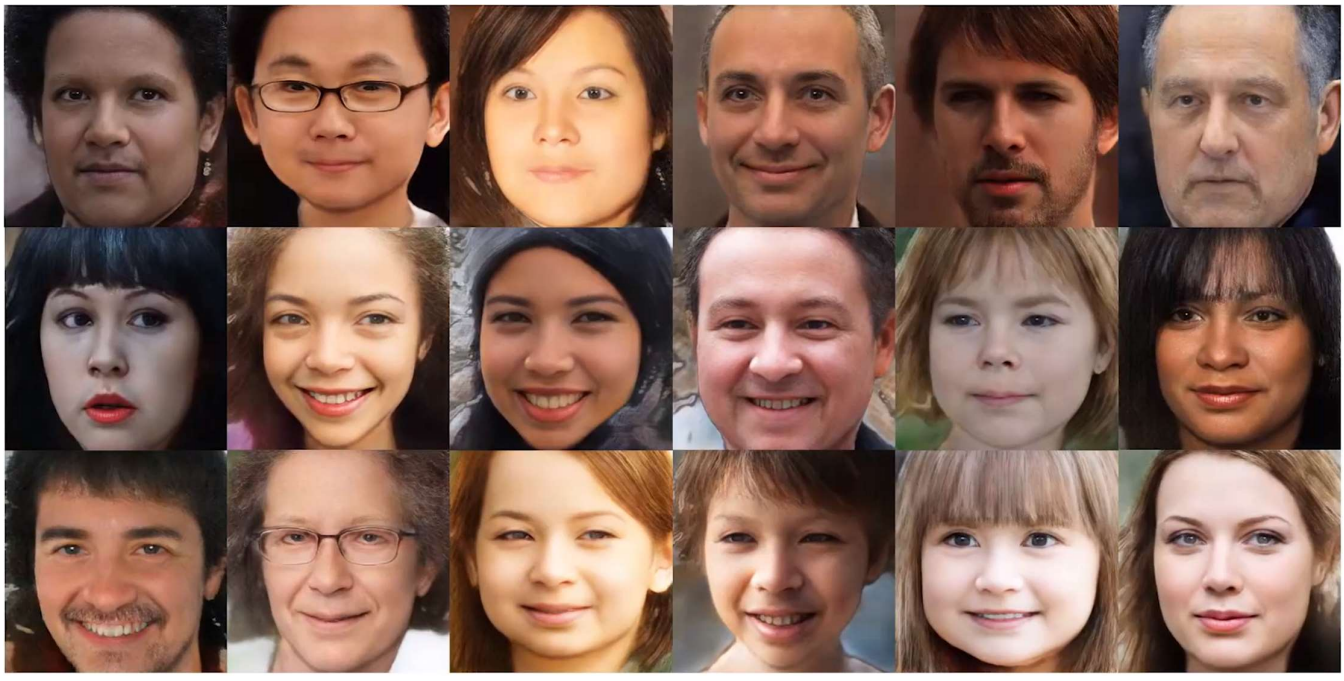


Figure 25: Exemples de visages générés

Sources : Auteurs de l'étude

❖ IA pour la génération d'images

L'étude[8] que j'ai sélectionnée au sujet de la génération d'image est celle de Hao Tang, Dan Xu, Yan Yan, Philip H. S. Torr et Nicu Sebe réalisée en 2020 conjointement dans les universités de Trento, Oxford, Texas State et Huawei Research en Ireland. L'étude examine le domaine de la génération de scènes guidée par la sémantique et cible la difficulté courante des méthodes de génération d'images globales à capter les petits objets et les textures locales détaillées.

Elle vise donc à élaborer un cadre de génération d'images qui combine à la fois la génération d'images globale et locale, en exploitant les avantages de chacune. Pour ce faire, les chercheurs conçoivent un réseau de génération conjointe avec un module de fusion d'attention et une structure de double discriminateur intégrée.

L'objectif est de générer des scènes plus réalistes en capturant à la fois la structure globale de l'image et les détails locaux, en utilisant les cartes sémantiques pour guider la génération. Cette captation de données permettra par la suite de générer une image en perspective relatant ce que montre en 2D l'image de départ.

Pour cela, la méthode de recherche de l'étude comprend une série d'expériences exhaustives menées sur deux tâches de génération d'images de scènes, à savoir la traduction d'images entre différentes perspectives et la synthèse d'images sémantiques. Ensuite, les performances du modèle proposé sont évaluées à l'aide de métriques quantitatives telles que l'indice d'inception, la précision et la divergence notée KL, ainsi que des évaluations qualitatives comparatives avec d'autres méthodes de recherche.

[8] Tang, Hao, Xu, Dan, Yan, Yan, Torr, Philip H.S. et Sebe, Nicu. (2020). Local Class-Specific and Global Image-Level Generative Adversarial Networks for Semantic-Guided Scene Generation. Dans *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 7867-7876). IEEE. https://doi.org/10.1109/CVPR42600.2020.00789

Figure 26: Exemples de résultats de traduction d'images transversales sur Dayton avec différents paramètres de notre LGGAN

Source : Auteurs de l'étude

❖ Etude au sujet du transfert de texture

L'étude[9] intitulée "RenderGAN: Generating Realistic Labeled Data", au sujet de la génération de données étiquetées réalistes réalisée par Leon Sixt, Benjamin Wild et Tim Landgraf en 2019 se situe au croisement de plusieurs enjeux majeurs en « computer vision ». Tout d'abord, elle aborde le défi crucial de l'annotation manuelle des données, qui peut être coûteuse et fastidieuse, limitant ainsi le déploiement de réseaux de neurones convolutionnels profonds (DCNN) dans des applications réelles. Ensuite, elle se penche sur la génération de données annotées à grande échelle, nécessaire pour entraîner efficacement ces réseaux de neurones, en particulier dans des domaines où les données sont rares ou difficiles à obtenir. Enfin, l'étude explore la nécessité de créer des données d'entraînement réalistes pour que les modèles d'IA puissent correctement généraliser les résultats à de nouvelles situations, en tenant compte des variations telles que l'éclairage, le fond et le bruit de l'image.

Le but principal de l'étude est de présenter un nouvel algorithme appelé RenderGAN, qui vise à surmonter les limitations liées à l'annotation manuelle des données en générant des images annotées de manière réaliste à partir d'un modèle 3D et de GANs. L'objectif est donc de fournir une solution efficace pour obtenir des données annotées à grande échelle, en minimisant les coûts et le temps nécessaires aux annotations manuelles.

Afin de mener à bien l'étude, la méthode de recherche repose sur une combinaison de modèles 3D et de GANs pour générer des images annotées de manière réaliste. Ensuite, est faite une description détaillée de l'architecture du générateur et du discriminateur utilisés pour le RenderGAN, ainsi que les étapes d'augmentation des images nécessaires pour le rendu réaliste des données générées. Enfin, l'étude évalue empiriquement l'efficacité du RenderGAN en comparant les performances des modèles DCNN entraînés sur des données générées par RenderGAN avec différentes balises, démontrant ainsi la supériorité de l'approche proposée.

[9] Sixt, Leon, Wild, Benjamin et Landgraf, Tim. (2018). RenderGAN: Generating Realistic Labeled Data. *Frontiers in Robotics and AI*, *5*. https://www.frontiersin.org/articles/10.3389/frobt.2018.00066

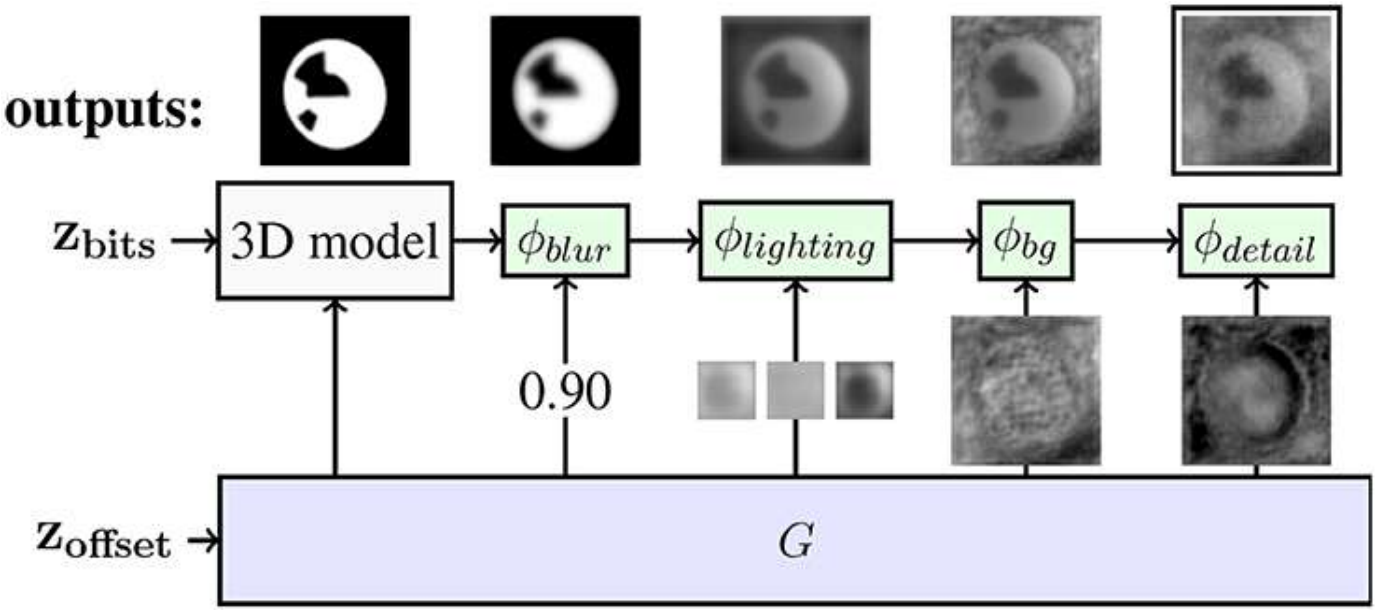


Figure 27: Augmentations du RenderGAN appliquées au projet BeesBook. Les flèches de G vers les augmentations φ représentent les entrées vers les augmentations. Le générateur fournit la position et les orientations du modèle 3D, tandis que les bits sont échantillonnés uniformément. En haut, la sortie de chaque étape est affichée. La sortie de $\varphi_{détail}$ est transmise au discriminateur.

Source : Auteurs de l'étude

❖ Etude au sujet du contrôle du trafic routier

Quant à la dernière étude[10] sélectionnée, il s'agit de celle intitulée "GE-GAN: A novel deep learning framework for road traffic state estimation" concernant l'estimation de l'état de trafic routier à l'aide de l'algorithme de deep-learning GE-GAN. L'étude réalisée par Dongwei Xu, Chenchen Wei, Peng Peng , Qi Xuan et Haifeng Guo en 2020 aborde plusieurs enjeux clés dans le domaine des systèmes de transport intelligents (ITS). L'un de ces enjeux concerne l'estimation précise de l'état du trafic routier, essentielle pour la gestion efficace des réseaux de transport et la prévention des embouteillages. Cependant, les données d'état du trafic collectées dans le monde réel sont souvent incomplètes, ce qui rend difficile leur utilisation pour des estimations précises. C'est dans ce contexte que s'inscrit cette étude, qui propose un nouveau cadre d'apprentissage profond pour estimer l'état du trafic en utilisant des informations provenant de liens adjacents. Ainsi, le but principal de l'étude est de proposer une solution novatrice pour surmonter les limitations des méthodes traditionnelles d'estimation de l'état du trafic en utilisant à la fois l'intégration de graphe (GE) et les GANs. En combinant ces deux techniques, l'étude vise à générer des données d'état du trafic routier en temps réel. L'enjeu est donc d'améliorer la précision de l'estimation de l'état du trafic tout en minimisant les contraintes liées à la disponibilité des données et à la complexité des modèles.

Tout comme les études précédentes, une méthode de recherche claire a été élaborée afin de réaliser l'étude dans de bonnes conditions. Cette dernière se divise en plusieurs étapes clés, notamment la représentation du réseau routier à l'aide de l'intégration de graphe, la sélection des données adjacentes pertinentes et l'application des GANs pour générer les données d'état du trafic. En outre, l'étude présente une évaluation approfondie de la méthode proposée en utilisant des jeux de données réels provenant de deux réseaux routiers différents, ce qui permet de valider empiriquement l'efficacité de l'approche proposée.

[10] Xu, Dongwei, Wei, Chenchen, Peng, Peng, Xuan, Qi et Guo, Haifeng. (2020). GE-GAN: A novel deep learning framework for road traffic state estimation. *Transportation Research Part C: Emerging Technologies*, *117*, 102635. https://doi.org/10.1016/j.trc.2020.102635

Partie 2 :
PROBLEMATIQUE

I. Questionnements

1. Ce que je tire de l'état de l'art Etude à mener dans ce mémoire

Suite à la lecture des études de l'état de l'art, je constate que les chercheurs se sont, pour beaucoup, concentrés sur l'utilisation des GAN pour la conception des plans de logements et de leur aménagement avec des contraintes telles que la position des fenêtres, des portes, etc. Jean-Raphaël Piquard, quant à lui, a suivis cette même idée mais, dans le cadre d'un mémoire de master, il s'est limité à la génération de formes à l'aide de GAN.

Néanmoins, l'utilisation des cGAN n'a pas été envisagée afin de pouvoir générer des formes (qui pourraient être considérée comme des emprises de bâtiments), ni de pouvoir les ajuster en fonction de certaines contraintes de construction que sont par exemple le POS (Pourcentage d'Occupation de Sol), les reculs de servitudes etc.

2. Etude à mener dans ce mémoire

Dans le but de poursuivre l'étude menée par Jean-Raphaël Piquard mais d'explorer les questions qui n'ont pas été abordées, mon étude a donc pour objectif sur la mise en place de réseaux de neurones antagonistes et génératifs conditionnés (cGAN) au service de la réalisation de modèle d'implantation de bâtiment dans un site. C'est-à-dire qu'il s'agira de mettre en place cet algorithme afin de générer des formes qui seront modifiables selon les contraintes que l'on y appliquera. In finé, l'enjeu serait à long terme de pouvoir se rapprocher le plus possible d'un algorithme cGAN capable de générer des empreintes de bâtiments que l'on pourra modifier selon les contraintes d'urbanisme.

II. Méthode de recherche

1. Démarche scientifique

Avant de présenter en détail, la méthode de recherche qui pourrait être difficile à cerner pour des personnes qui ne s'y connaissent pas très bien sur les sujets liés à la création d'algorithme d'intelligence artificielle, faisons une analogie avec le système d'enseignement à l'école.

Pour l'exemple, prenons deux élèves A et B qui suivent des cours à l'école auprès d'un enseignant et qui seront soumis à des partiels à la fin de l'année. Ces élèves suivent les cours, apprennent des enseignants, font des expériences, des recherches afin de mieux assimiler les différents cours suivis pour pouvoir par la suite les restituer. Sauf que les deux étudiants ont des particularités. L'étudiant A est assez bon pour restituer textuellement ce qu'il apprend alors que l'étudiant B est, quant à lui, capable de restituer ce qu'il apprend mais peut améliorer ses résultats quand on lui fait une correction. Ainsi, afin de pouvoir challenger l'étudiant B, l'enseignant veut tester ce dernier afin qu'il puisse avoir des résultats meilleurs que l'étudiant A, en lui faisant des corrections supplémentaires.

La fin de l'année arrive, les partiels se déroulent durant lesquels les élèves proposent des projets que l'on appellera Pa(i) pour l'étudiant A et Pb(i) pour l'étudiant B. « i » étant le nombre de propositions. Soit, par exemple, Pa3, le troisième résultat de l'étudiant A.

Enfin, à la fin du semestre, afin d'observer si l'étudiant B est meilleur que l'étudiant A, l'enseignant va comparer leurs propositions. Ainsi, si les deux étudiants ont les mêmes notes, il constatera que B est aussi bon que l'autre étudiant. Si B a une meilleure note que l'autre étudiant, l'enseignant sera rassuré sur le fait que l'étudiant B a bien pris en compte ses corrections.

J'en convient que ce n'est pas un système d'enseignement équitable mais il s'agit simplement d'une analogie afin de simplifier les concepts.

L'analogie étant faite, ma méthode de recherche suivra, dans l'ensemble, le même processus de notation. De ce fait, dans ma recherche, l'étudiant A correspond à un GAN et l'étudiant B correspond à un cGAN. Le cGAN étant l'algorithme qui est sujet de mon étude et donc qu'il convient de tester. Les partiels correspondent au processus de génération d'image du GAN et du cGAN tandis que les propositions Pa(i) et Pb(i) correspondent aux propositions générées respectivement par chacun des deux algorithmes. L'enseignant, c'est moi, c'est-à-dire que c'est moi qui réalise les catégorisations des résultats, leur analyse et les conclusions qu'il faut en tirer. Et enfin, les critères d'évaluation sont le respect d'un ratio de surface et le respect d'une fidélité de forme des propositions par rapport aux formes d'entrainement.

2. Comment caractérise-t-on chacun des critères d'évaluation ?

a. Le ratio de surface :

Sachant que les propositions de forme devront être générées dans un cadre rectangulaire de dimension 20mX30m soit un rapport de 1 x 1,5, il s'agira du pourcentage d'occupation que prend la forme générée dans le cadre. Par ailleurs, je veillerai à ce que les formes d'entrainement aient un ratio de surface de 0,6 ou 60% afin de le comparer à ceux des propositions. Par exemple, si la forme générée mesure 150m2 de surface, cela signifiera qu'elle a un ration de surface de (150/600)*100= 25%, soit 0,25. Ou comme le montre le schéma ci-dessous, les propositions peuvent respecter le même ratio de surface même si la forme proposée est différente.

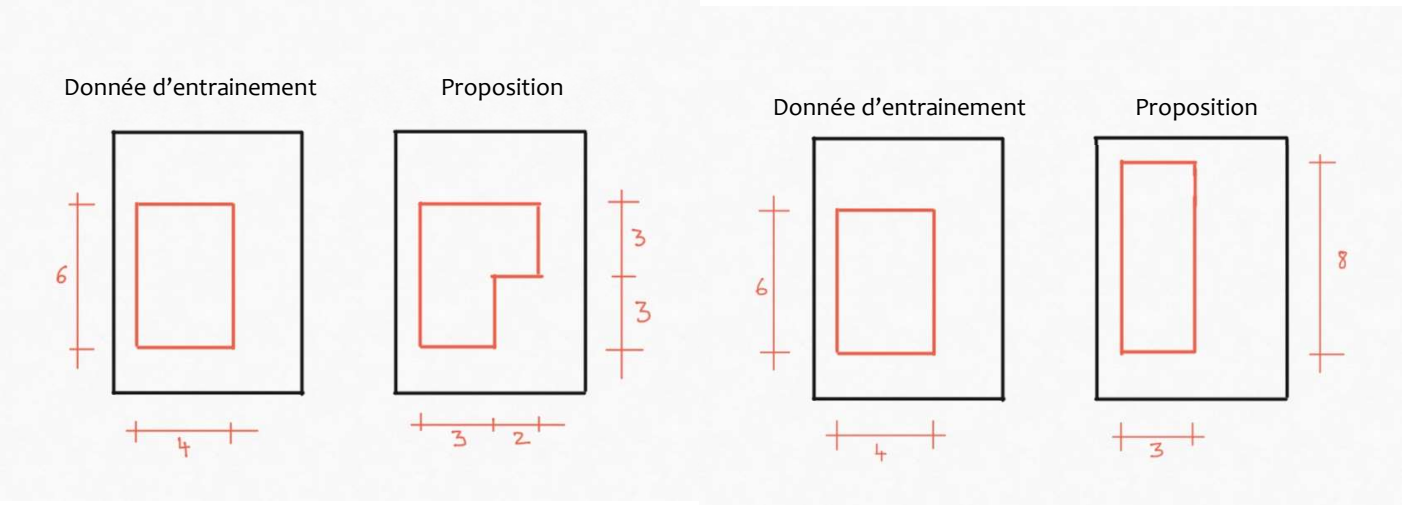


Figure 28: Illustration du ratio de surface

Source : Auteur, POUHE Fahé

b. Le respect de fidélité de forme :

Quant à lui consiste à vérifier que les propositions générées correspondent uniquement aux formes des prototypes d'entrainement. Cela permet donc que des propositions de taille réduite ou plus grande mais qui respectent les formes d'entrainement puissent être valide. Par exemple, le schéma ci-dessous permet de montrer que l'on peut respecter une forme précise mais pas sa taille.

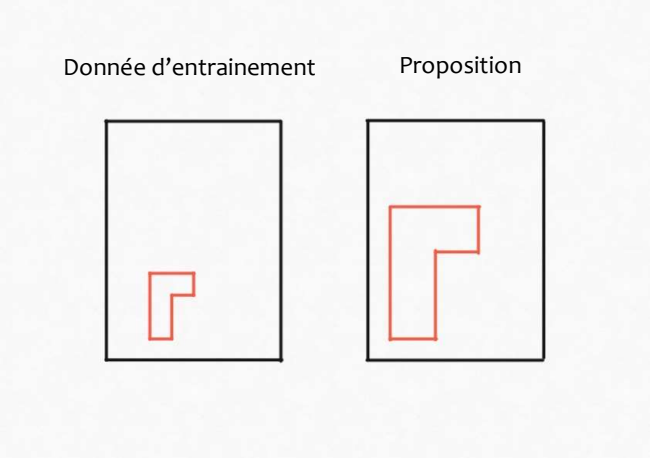


Figure 29: Illustration de la fidélité de forme

Source : Auteur, POUHE Fahé

c. Processus de recherche

Pour ainsi dire, la méthode de recherche que j'utilise pour cette recherche est la suivante :

Cette méthode s'articule autour de 3 grandes étapes que sont la mise en place d'un GAN qui servira de point de départ de l'étude mais aussi de balise de comparaison, ensuite nous avons la mise en place d'un cGAN qui servira à évaluer l'efficacité de notre algorithme d'IA et enfin une étape d'évaluation des propositions obtenues par les deux algorithmes. Cette dernière a pour but d'analyser et de comparer les résultats du GAN à ceux du cGAN et d'en tirer des conclusions.

L'élaboration du GAN se déroule en quatre (04) étapes principales :

- Génération de modèle d'implantation d'apprentissage grâce à un outil paramétrique. Ces modèles seront créés afin de respecter au moins l'un des futurs critères d'évaluation
- Créer un réseau de neurones (GAN) qui apprend de ces modèles tout comme l'étudiant A apprend ses cours
- Générer de nouveaux modèles d'implantation à partir de ce GAN
- Analyser (manuellement) des propositions obtenues : Cette étape permettra d'identifier les modèles exploitables ou non. Ces derniers seront identifiés selon des critères d'évaluation que sont :

    - Fidélité d'apprentissage : le but est de savoir si le résultat obtenu est fidèle aux modèles d'apprentissage. S'il l'est, on saura que l'algorithme est capable d'imiter. Par analogie, ça signifie que si l'étudiant A est capable de restituer à l'identique ce qu'il a appris, ça signifie qu'il connait au moins ses cours.
    - Respect d'un ratio de surface : il s'agit de savoir si les modèles générés respectent une surface précise. Cela voudrait dire que l'étudiant A a bien appris et est capable d'appliquer une compétence précise acquise en cours. Pour un étudiant en Architecture, cela pourrait être, par exemple, le fait d'être capable d'utiliser une fonction précise d'un logiciel appris en cours mais pour son projet de semestre.

Cette analyse consistera donc à classifier, catégoriser les propositions obtenues par le GAN et ensuite faire des ratios de performances qui permettront d'identifier le ratio de fidélité respecté par rapport au nombre total de propositions obtenues, le ratio de surface d'occupation respecté par rapport au nombre total de propositions obtenues et enfin, comparer ces deux ratios afin d'identifier quel est le critère d'évaluation le plus respecté par le GAN.

Tableau d'évaluation statistique

Evaluation du GAN	Critères d'évaluation	
	Fidélité	Surface d'occupation
Proposition Pa1	√	√
Proposition Pa2	-	√
Proposition Pa (i)	-	√
Nbre de résultats par critère	NaF	NaS
Nbre total de résultats	NaTotal	

Ratio fidélité = $\frac{NaF}{NaTotal}$, il s'agit de la proportion de propositions respectant la fidélité de forme par rapport à l'ensemble des propositions

Ratio surface d'occupation = $\frac{NaS}{NaTotal}$, il s'agit de la proportion de propositions respectant le critère de surface par rapport à l'ensemble des résultats.

Ensuite, nous passons à la mise en place du cGAN qui se déroule en trois (03) étapes :

- Créer un réseau de neurones conditionné (cGAN) à partir du GAN mais le conditionner afin qu'il puisse respecter les critères d'évaluation tels que la surface d'occupation et la fidélité de la forme.
- Insérer les propositions générées par le GAN dans le cGAN créé afin de les conditionner et générer d'autres propositions. Ici, contrairement à l'analogie, l'on utilisera les bons résultats de l'étudiant A afin de faire apprendre les cours à l'étudiant B pour savoir s'il est capable de faire de même ou mieux.
- Analyser les résultats obtenus selon les mêmes critères d'évaluation et le même procédé que ceux du GAN :

Tableau d'évaluation statistique

Evaluation du cGAN	Critères d'évaluation	
	Fidélité	Surface d'occupation
Proposition Pb1	√	√
Proposition Pb2	-	√
Proposition Pb (i)	-	√
Nbre de résultats par critère	NbF	NbS
Nbre total de résultats	NbTotal	

Ratio fidélité = $\frac{NbF}{NbTotal}$

Ratio surface d'occupation = $\frac{NbS}{NbTotal}$

Chacune des étapes de l'analyse des résultats ou propositions obtenues équivaut donc à l'enseignant qui va noter les performances de l'étudiant A et ensuite noter celles de l'étudiant B.

Enfin, suite à l'élaboration de nos récapitulatifs statistiques respectifs du GAN et du cGAN, nous avons l'étape d'analyse globale qui consiste à comparer les données statistiques obtenues des deux technologies afin d'identifier laquelle est plus performante selon quel critère et laquelle est plus performante dans l'ensemble. Tout comme l'enseignant fera une comparaison entre ses deux étudiants.

Par conséquent, si le GAN a un ratio de fidélité moins élevé que celui du cGAN, l'on pourra affirmer que le cGAN est meilleur dans la restitution de données apprises que le GAN. Par ailleurs, si le GAN a un ratio de surface d'occupation plus grand que celui du cGAN, l'on pourra affirmer que le GAN est meilleur sur ce sujet précis. Et enfin, en moyenne, si le cGAN fournit le plus de résultats conformes aux critères d'évaluation, l'on pourra affirmer qu'il est meilleur dans l'ensemble.

Partie 3 :
EXPERIMENTATION

I. Outils nécessaires à la réalisation de l'expérience

Réaliser une expérience liée aux intelligences artificielles nécessite certains **outils spécifiques de programmation**, des **outils de visualisation 2D ou 3D** vu qu'en Architecture nous travaillant souvent avec des représentations graphiques. Mais avant tout, générer des IA nécessitant des connaissances spécifiques, avant de commencer l'expérience, j'ai tout d'abord dû réapprendre les bases de la programmation informatique. Même si j'en avais déjà certaines connaissances, j'étais habitué à travailler avec le langage de programmation Java. Dans le cadre de cette étude, **Python est le langage de programmation le plus accessible** et qui dispose le plus d'information et de documentations en ligne.

1. Apprentissage de la programmation

Dans un premier temps, j'ai donc commencé par apprendre à programmer en Python, notamment quelle en est la syntaxe, la définition de variables, de matrices, de conditions et de boucles, entre autres. La plateforme la plus accessible que j'ai trouvé pour cela est **Mimo**.

Il s'agit d'un site internet et d'une application mobile permettant d'apprendre à coder en HTML, JavaScript, CSS et plus. C'est une application que j'ai principalement utilisée afin de réapprendre à coder plus rapidement et simplement à coder en python grâce à des challenges et des énigmes que l'outil met en place.

2. Logiciels de création de la base de données

Dans un deuxième temps, j'ai dû utiliser différents outils pour la génération de données. Il s'agit principalement de deux outils dont **Rhinoceros et Grasshopper**. Les autres outils sont des pulgins téléchargeables ou intégrés dans Grasshopper.

Rhinoceros 3D est un logiciel de modélisation 3D utilisé principalement dans les domaines de l'architecture, du design industriel et de l'ingénierie. Il offre des outils puissants pour la création et la manipulation de formes géométriques complexes. Notamment, l'ensemble des dessins nécessaires à générer la base de donnée est réalisé et affiché dans l'interface de Rhinoceros.

Grasshopper : C'est un plug-in pour Rhinoceros 3D qui permet de créer des algorithmes visuels pour la conception paramétrique et générative. Il m'a permis de créer des modèles 2D en manipulant des composants visuels et en connectant des flux de données. C'est notamment grâce à ce outil que je crée la base de données du GAN et cGAN.

TT Toolbox (plugin) : Il s'agit d'un plug-in pour Grasshopper qui fournit une collection d'outils et de fonctionnalités supplémentaires pour la conception paramétrique et la modélisation 3D. En l'occurrence, c'est grâce à ce plugin que les formes générées par Grasshopper sont enregistrées et exportées en images. Ce sont ces images qui constituent la base de données finale.

GhPython : Il s'agit d'un composant de Grasshopper qui permet d'écrire des scripts Python directement à l'intérieur de l'environnement Grasshopper. C'est dans cet outil que je crée des scripts de génération de formes.

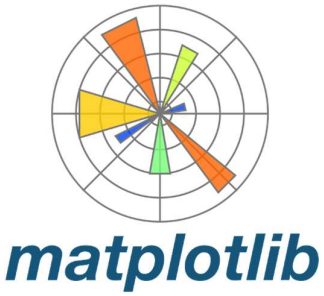
Rhinoscriptsyntax : C'est un module Python qui fournit une interface simple et efficace pour interagir avec Rhinoceros 3D depuis Python. Il permet d'automatiser des tâches de modélisation et de scripter des opérations dans Rhinoceros 3D. Ainsi, à travers ce module, le script python créé dans GhPython peut interagir avec Grasshopper et avec Rhinoceros 3D par ricochet.

3. Outils de mise en place de l'algorithme

Enfin, comme indiqué précédemment, les outils spécifiques de mise en place de l'algorithme sont **Python et Google Colab**. Les autres outils sont des bibliothèques à implémenter dans Python.

Python est un langage de programmation permettant de créer et modifier le script de l'algorithme GAN mais aussi permettant de créer le code nécessaire à la réalisation des formes dans GhPython. Sinon il s'agit d'un langage de programmation polyvalent et populaire, largement utilisé dans divers domaines, y compris le développement web, l'analyse de données, l'apprentissage automatique et la modélisation 3D. Il est apprécié pour sa simplicité et sa lisibilité.

Google Colab est une plateforme de notebook basée sur le cloud qui permet d'exécuter du code Python, notamment pour l'apprentissage machine, l'analyse de données et d'autres tâches de programmation, directement dans un navigateur web. J'en ai eu l'utilité afin de mettre en place l'algorithme du GAN. Avant d'utiliser cet outil, j'avais commencé par travailler sur un autre notebook du nom de **Anaconda**. J'ai arrêté d'utiliser cette dernière parce qu'il était assez fastidieux de mettre en place des environnements de travail avant même de commencer à programmer l'algorithme.



Matplotlib est une bibliothèque de visualisation de données en Python, largement utilisée pour créer des graphiques 2D et 3D de données de haute qualité.

Keras est une bibliothèque open source de réseaux de neurones de haut niveau, écrite en Python et fonctionnant sur **TensorFlow** (Bibliothèque de Machine learning de Google). Elle permet de construire, de former et de déployer rapidement des modèles d'apprentissage en profondeur.

NumPy est une bibliothèque Python qui permet de créer des tableaux multidimensionnels et les fonctions mathématiques pour travailler efficacement avec des données numériques.

MLxtend est une bibliothèque Python d'extensions pour l'apprentissage machine, offrant une large gamme d'outils et d'utilitaires pour la préparation des données, la validation des modèles et l'interprétation des résultats.

II. Mise en place de l'algorithme

Cette étape est délicate à réaliser car elle nécessite d'avoir de bonnes connaissances en programmation en Python, connaitre les différentes bibliothèques de commande nécessaire à la bonne réalisation de l'algorithme mais aussi être bien renseigné sur les algorithmes de réseaux de neurones et les matrices de données.

Ayant des connaissances en programmation en Java, j'ai pu appréhender de manière général, le mode de fonctionnement des prototypes d'algorithmes de réseaux de neurones que j'ai pu observer sur internet mais il m'a fallu acquérir plusieurs connaissances en programmation en Python. N'étant néanmoins, pas un expert, j'ai rencontré plusieurs difficultés dans la correction d'erreur de code qui m'ont beaucoup ralenti lors de l'étude.

Néanmoins, j'ai pu me procurer le script du GAN de génération utilisé par Jean-Raphaël PIQUARD pour son mémoire de Master à l'ENSAPLV, permettant de générer des écritures manuscrites mais qui peut être utilisé pour d'autres fonctions. Il s'agit d'un exemple de GAN couramment utilisé et facilement retrouvable sur internet pour les personnes qui souhaitent débuter dans la création de GAN.

Ainsi, avant de pouvoir le modifier et l'utiliser à ma guise, il m'a fallu comprendre son fonctionnement.

1. Principe de fonctionnement d'un GAN :

Importation de bibliothèques et modules :

Importation de fonctions et modules de NumPy pour la manipulation des données et de divers modules de Keras, tels que **Sequential**, **Dense**, **Conv2D**, **LeakyReLU**, **BatchNormalization**, et d'autres, nécessaires pour définir et entraîner les modèles du GAN.

```
# example of training a stable gan for generating a handwritten digit
from os import makedirs

# Vérifier la présence de Keras
import keras
print(keras.__version__)

# Vérifier la présence de NumPy
import numpy as np
print(np.__version__)

# Vérifier la présence de Matplotlib
import matplotlib
print(matplotlib.__version__)

# Vérifier la présence de mlxtend
import mlxtend
print(mlxtend.__version__)

from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
from keras.initializers import RandomNormal
from matplotlib import pyplot
```

Figure 30: Importation de bibliothèques et modules

Source : POUHE Fahé

Définition du Discriminateur :

1. La fonction **define_discriminator** définit le modèle du discriminateur.
2. Le modèle utilise des couches convolutionnelles pour réduire progressivement la résolution spatiale des images.
3. Le modèle prend en entrée une image de forme (28, 28, 1), typique des images MNIST.
4. La dernière couche est une couche dense avec une activation sigmoïde pour la classification binaire (réelle ou générée).

```
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # define model
  model = Sequential()
  # downsample to 14x14
  model.add(Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init, input_shape=in_shape))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # downsample to 7x7
  model.add(Conv2D(64, (4,4), strides=(2,2), padding='same', kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model
```

Figure 31: Script de définition du Discriminateur

Source : POUHE Fahé

Définition du Générateur :

1. La fonction **define_generator** définit le modèle du générateur.
2. Le modèle utilise des couches denses pour générer une représentation latente de forme (128 * 7 * 7), qui est ensuite remodelée en une image 3D.
3. Des couches de transposition de convolution sont utilisées pour augmenter progressivement la résolution spatiale de l'image générée jusqu'à (28, 28, 1).
4. La dernière couche utilise une activation tanh pour assurer que les valeurs générées sont dans la plage [-1, 1].

```python
# define the standalone generator model
def define_generator(latent_dim):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # define model
  model = Sequential()
  # foundation for 7x7 image
  n_nodes = 128 * 7 * 7
  model.add(Dense(n_nodes, kernel_initializer=init, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((7, 7, 128)))
  # upsample to 14x14
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 28x28
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same', kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # output 28x28x1
  model.add(Conv2D(1, (7,7), activation='tanh', padding='same', kernel_initializer=init))
  return model
```

Figure 32: Script du Générateur

Source : POUHE Fahé

Définition du GAN :

1. La fonction **define_gan** combine le générateur et le discriminateur en un modèle.
2. Les poids du discriminateur ne sont pas entraînés lors de la mise à jour du GAN.

```python

# define the combined generator and discriminator model, for updating the generator
def define_gan(generator, discriminator):
  # make weights in the discriminator not trainable
  discriminator.trainable = False
  # connect them
  model = Sequential()
  # add generator
  model.add(generator)
  # add the discriminator
  model.add(discriminator)
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt)
  return model
```

Figure 33: Script de définition du GAN

Source : POUHE Fahé

Chargement des données :

1. Les données MNIST sont chargées à l'aide de la fonction **loadlocal_mnist** de la bibliothèque **mlxtend**.
2. Les images sont mises à l'échelle de la plage [-1, 1].

```
102 import mlxtend
103 import numpy as np
104
105 from mlxtend.data import loadlocal_mnist
106
107 trainX, trainy = loadlocal_mnist(
108         images_path=r'Z:\Fahé POUHE\Mémoire - Fahé\test-images-idx3-ubyte',
109         labels_path=r'Z:\Fahé POUHE\Mémoire - Fahé\test-labels-idx1-ubyte')
110
111 print(trainX.shape[0], trainX.shape[1])
112 trainX=np.reshape(trainX,(trainX.shape[0],28,-1))
113 print(trainX.shape[0], trainX.shape[1], trainX.shape[2])
114
115 # load mnist images
116 def load_real_samples():
117   # load dataset
118   # expand to 3d, e.g. add channels
119
120   X = expand_dims(trainX, axis=-1)
121   # select all of the examples for a given class
122   selected_ix = trainy == 8
123   X = X[selected_ix]
124   # convert from ints to floats
125   X = X.astype('float32')
126   # scale from [0,255] to [-1,1]
127   X = (X - 127.5) / 127.5
128   return X
129
```

Figure 34: Script de données

Source : POUHE Fahé

Définition des fonctions pour la génération de données réelles et fausses :

1. La fonction **generate_real_samples** sélectionne un échantillon aléatoire d'images réelles à partir du jeu de données.
2. La fonction **generate_latent_points** génère des points dans l'espace latent pour alimenter le générateur.
3. La fonction **generate_fake_samples** utilise le générateur pour créer un échantillon d'images générées.

```
# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n_samples):
  # generate points in latent space
  x_input = generate_latent_points(latent_dim, n_samples)
  # predict outputs
  X = generator.predict(x_input)
  # create class labels
  y = zeros((n_samples, 1))
  return X, y
```

Figure 35: Script de données réelles et fausses

Source : POUHE Fahé

Fonctions pour la sauvegarde de la performance du modèle :

1. La fonction **summarize_performance** génère des échantillons générés par le générateur, les sauvegarde sous forme d'une grille, et sauvegarde le modèle du générateur.
2. La fonction **plot_history** crée un graphique de l'historique des pertes et de l'exactitude du discriminateur.

```
158 # generate samples and save as a plot and save the model
159 def summarize_performance(step, g_model, latent_dim, n_samples=100):
160   # prepare fake examples
161   X, _ = generate_fake_samples(g_model, latent_dim, n_samples)
162   # scale from [-1,1] to [0,1]
163   X = (X + 1) / 2.0
164   # plot images
165   for i in range(10 * 10):
166     # define subplot
167     pyplot.subplot(10, 10, 1 + i)
168     # turn off axis
169     pyplot.axis('off')
170     # plot raw pixel data
171     pyplot.imshow(X[i, :, :, 0], cmap='gray_r')
172   # save plot to file
173   pyplot.savefig('results_baseline/generated_plot_%03d.png' % (step+1))
174   pyplot.close()
175   # save the generator model
176   g_model.save('results_baseline/model_%03d.h5' % (step+1))
```

Figure 36: Script de sauvegarde de performance du modèle

Source : POUHE Fahé

Boucle d'entraînement :

1. La fonction **train** effectue l'entraînement du GAN.
2. Elle utilise des lots d'images réelles et générées pour mettre à jour les poids du discriminateur et du générateur.
3. Les performances du modèle sont résumées périodiquement, et l'historique des pertes est tracé à la fin de l'entraînement.

```
# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=10, n_batch=50):
    # calculate the number of batches per epoch
    bat_per_epo = 10
    # calculate the total iterations based on batch and epoch
    n_steps = bat_per_epo * n_epochs
    # calculate the number of samples in half a batch
    half_batch = int(n_batch / 2)
    # prepare lists for storing stats each iteration
    d1_hist, d2_hist, g_hist, a1_hist, a2_hist = list(), list(), list(), list(), list()
    # manually enumerate epochs
    for i in range(n_steps):
        # get randomly selected 'real' samples
        X_real, y_real = generate_real_samples(dataset, half_batch)
        # update discriminator model weights
        d_loss1, d_acc1 = d_model.train_on_batch(X_real, y_real)
        # generate 'fake' examples
        X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
        # update discriminator model weights
        d_loss2, d_acc2 = d_model.train_on_batch(X_fake, y_fake)
        # prepare points in latent space as input for the generator
        X_gan = generate_latent_points(latent_dim, n_batch)
        # create inverted labels for the fake samples
        y_gan = ones((n_batch, 1))
        # update the generator via the discriminator's error
        g_loss = gan_model.train_on_batch(X_gan, y_gan)
        # summarize loss on this batch
        print('>%d, d1=%.3f, d2=%.3f g=%.3f, a1=%d, a2=%d' %
            (i+1, d_loss1, d_loss2, g_loss, int(100*d_acc1), int(100*d_acc2)))
        # record history
        d1_hist.append(d_loss1)
        d2_hist.append(d_loss2)
        g_hist.append(g_loss)
        a1_hist.append(d_acc1)
        a2_hist.append(d_acc2)
        # evaluate the model performance every 'epoch'
        if (i+1) % bat_per_epo == 0:
            summarize_performance(i, g_model, latent_dim)
    plot_history(d1_hist, d2_hist, g_hist, a1_hist, a2_hist)
```

Figure 37: Script d'entrainement du générateur et du discriminateur

Source : POUHE Fahé

Création d'un dossier pour les résultats :

1. Le script crée un dossier nommé 'results_baseline' pour sauvegarder les résultats générés et les modèles.

Entraînement du GAN :

1. La taille de l'espace latent est définie à 50.
2. Le discriminateur, le générateur, et le GAN sont créés.
3. Les données MNIST sont chargées.
4. L'entraînement du GAN est effectué en utilisant les fonctions définies précédemment.

```
235 # make folder for results
236 makedirs('results_baseline', exist_ok=True)
237 # size of the latent space
238 latent_dim = 50
239 # create the discriminator
240 discriminator = define_discriminator()
241 # create the generator
242 generator = define_generator(latent_dim)
243 # create the gan
244 gan_model = define_gan(generator, discriminator)
245 # load image data
246 dataset = load_real_samples()
247 print(dataset.shape)
248 # train model
249 train(generator, discriminator, gan_model, dataset, latent_dim)
```

Figure 38: Script de sauvegarde des données générées et d'entrainement du GAN

Source : POUHE Fahé

Le script complet du GAN est à retrouver en annexe.

III. Mise en place de la base de données :

Création de la base de données : Génération de prototypes d'implantation dans Grasshopper

La base de données peut être réalisée de plusieurs manières mais dans le cadre de ce mémoire, j'en ai testé deux et gardé une manière afin de faciliter le déroulement de l'expérience.

1. Base de données à l'aide de code Python

Ce procédé consiste en la création de la base de données à l'aide de trois (03) scripts en GhPython ("Formes", "Terrains" et "Insertion") qui permettent d'automatiser la génération des modèles d'entrainement du GAN. Ces scripts sont mis en relation à travers l'outil de paramétrisme Grasshopper. Dans ce premier procédé, le modèle de génération de données d'entrainement repose sur une séquence logique afin de créer des compositions harmonieuses dans un modèle 2D. Tout d'abord, dans le script "Terrains", des rectangles sont générés pour représenter le contour des terrains. Les dimensions des contours sont de 20 unités x 30 unités afin de singer un terrain réel de 20m sur 30m, soit 600m2. Mais l'on a la possibilité de modifier les dimensions des contours directement dans Grasshopper à l'aide du composant « Rectangle » et des paramètres qui lui sont attribués. Ces délimitations de terrains sont donc définies manuellement et servent de toile de fond.

En parallèle, le script "Formes" est responsable de la génération de formes, également définies manuellement, mais avec une superficie correspondant à 60% des terrains de 600m2 précédemment créés. Dans le script aux lignes 20 et 21, le critère d'évaluation « ratio de surface » est définis à 60% mais toujours modifiable si besoin.

```
19
20      # Calcule la superficie cible des formes (60% de la superficie de x)
21      target_shape_area = 0.6 * x_area
22
```

Figure 39: Entrée du critères d'évaluation dans le script « Formes »

Source : Auteur, POUHE Fahé

Le script "Insertion" s'en charge car il orchestre la fusion harmonieuse des formes générées dans les terrains correspondants. À travers une itération sur les paires de terrains et de formes, le script vérifie d'abord que ces éléments sont des courbes valides, puis positionne les formes à l'intérieur des terrains.

Ce processus séquentiel assure que chaque forme trouve sa place dans le contexte du terrain associé, aboutissant à un modèle 2D cohérent.

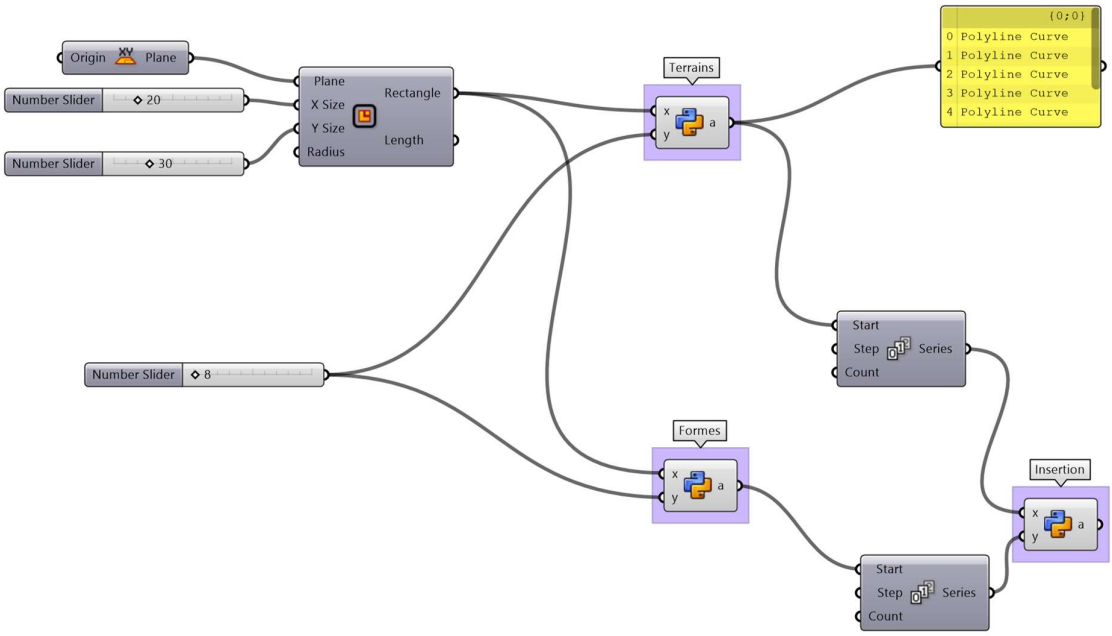


Figure 40: Script Grasshopper de génération de formes

Source : Auteur, POUHE Fahé

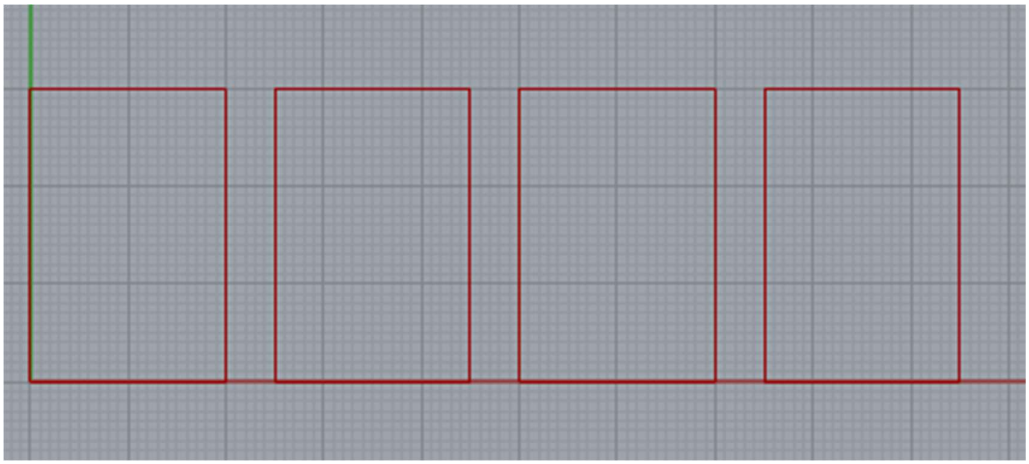


Figure 41: Visualisation des contours de terrains de 20 x 30

Source : Auteur, POUHE Fahé

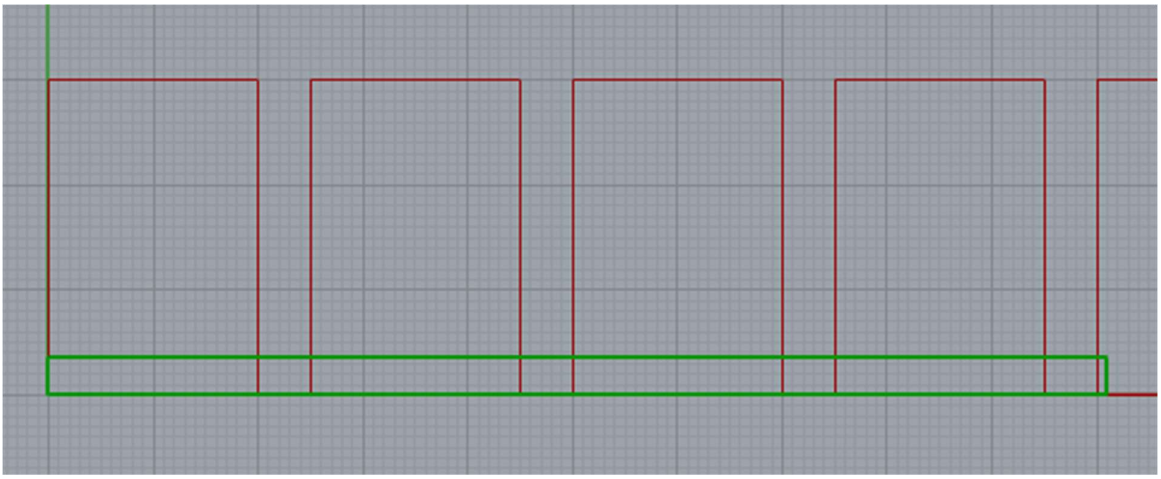


Figure 42: Visualisation des formes générées (en vert)

Source : Auteur, POUHE Fahé

Les formes étant générées sont des rectangles, elles doivent être par la suite être intégrées dans les terrains. Néanmoins, j'ai rencontré un problème de précision dans les dimensions des formes générées car elles sont trop grandes en proportion comparé aux terrains. Ayant rencontré des difficultés lors de la première manière que j'aurais pu résoudre mais qui me demanderait plus de temps et de connaissance, j'ai décidé de remédier à cela en générant les formes paramétriquement et directement avec les commandes de Grasshopper.

2. Base de données à l'aide du paramétrisme

C'est cette démarche que je vais finalement utiliser pour créer la base de données. Ainsi, afin de simplifier le plus possible et optimiser les possibilités de génération de différentes formes, je me suis finalement servis du modèle Grasshopper créé par Jean Raphaël PIQUARD lors de son mémoire sur les GAN et l'architecture. Dans son modèle Grasshopper, l'on retrouve différentes formes telles que des carrés, des rectangles, des blobs, etc mais ces formes ne respectent pas forcément un critère spécifique de taille ou superficie comme je veux le faire dans mon mémoire. J'ai donc modifié son modèle afin que les formes générées respectent le critère de ratio de surface qui est de 60% de 600m2, soit 360 m2 au maximum. Afin d'obtenir des formes viables et ainsi éviter d'obtenir des formes de 1m2 (par exemple et non viable), je renseigne un ensemble de variables permettant d'obtenir des formes de superficie comprise entre 150 m2 et 360 m2.

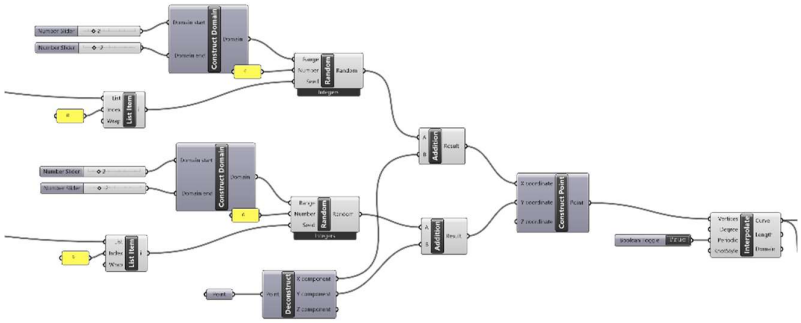


Figure 43: Modèle Grasshopper de génération de formes

Source : Jean Raphael PIQUARD, Mémoire de master 2020

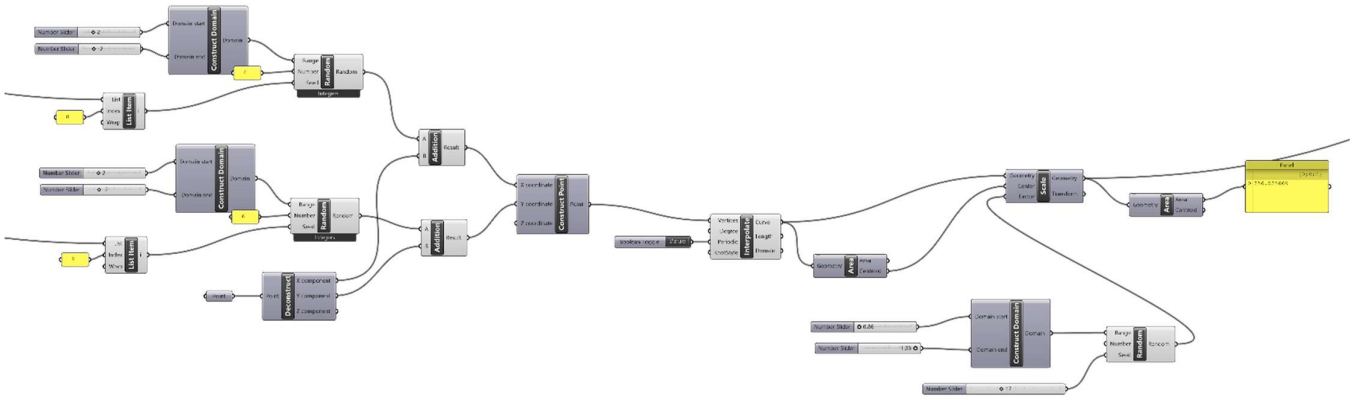


Figure 44: Modèle de génération de formes modifié respectant le critère ratio de surface

Source : Auteur, POUHE Fahé, mémoire de Master

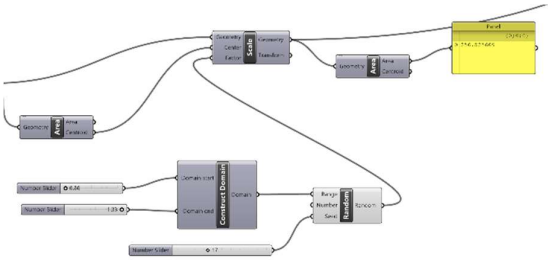


Figure 45: Modèle Grasshopper rajouté permettant de respecter le ratio de surface

Source : Auteur, POUHE Fahé, mémoire de Master

Le rajout de code Grasshopper permet d'établir un changement d'échelle de la forme initiale générée afin qu'elle puisse respecter le critère « ratio de surface ».

Suite à la modification complète du modèle Grasshopper et le dessin du « terrain » de dimension 20 m x 30 m directement dans Rhino, l'on obtient les images constitutives de la base données des formes inscrites dans le terrain. Chacune des images de la base de données sont donc respectueuses des critères d'évaluation et peuvent de ce fait être utilisée comme données d'entrainement du GAN.

Par ailleurs, la figure ci-dessous permet d'observer des exemples de formes de blobs inscris dans le terrain. La plupart des formes sont très différentes les unes des autres mais restent toutes des blobs. Néanmoins, l'on observe que certaines formes débordent de la limite du terrain même si elles respectent le ratio de surface. Il conviendrait donc dans certains cas, de rectifier cela en rajoutant des contraintes de limite dans le modèle Grasshopper.

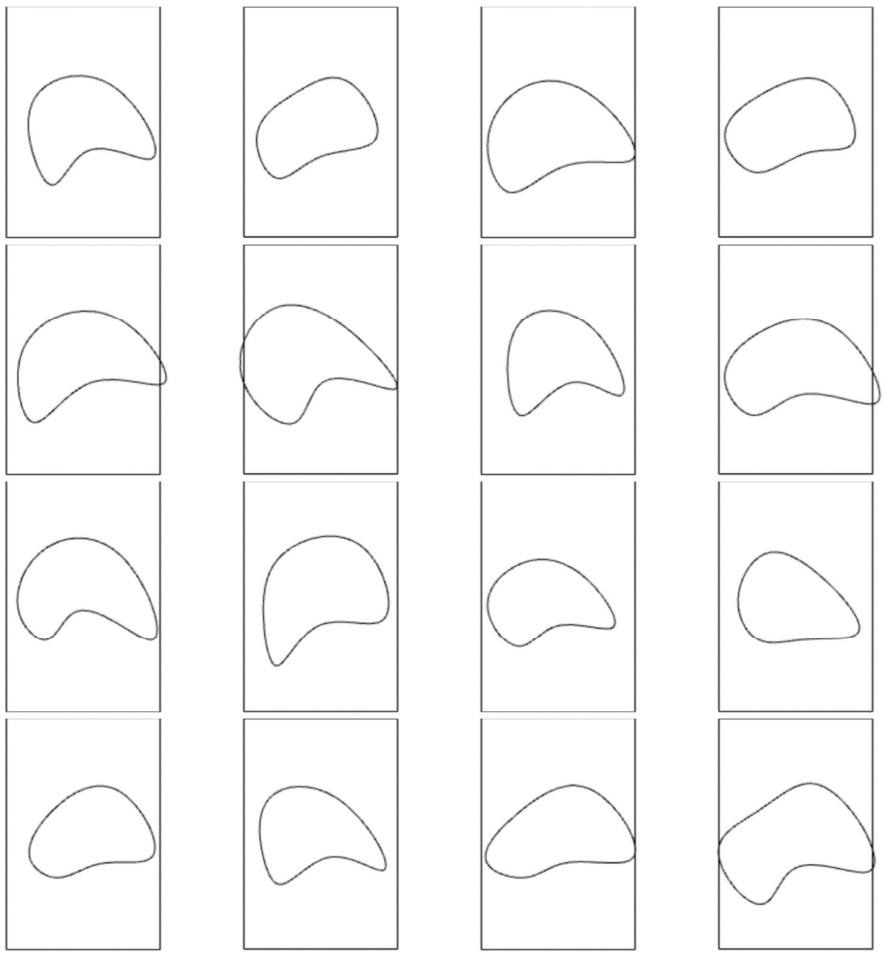


Figure 46: Fragments des formes de blobs générées par le modèle paramétrique

Source : Auteur, POUHE Fahé

IV. Entraînement de l'algorithme

L'entrainement du GAN ainsi que de la base de données n'a pas pu aboutir car plusieurs blocages ont eu lieu aussi bien la création de la base de données d'entrainement mais aussi lors de la connexion entre la base de données et le script du GAN.

1. Concernant la base de données :

Dans les meilleures conditions, les étapes à suivre seraient d'écrire les scripts de génération de terrains, de forme et d'insertion des formes dans les terrains. Ensuite, je devrais exporter les exemples d'entrainement en format image. Ces images devraient par la suite être connectées à l'algorithme GAN afin de procéder à l'entrainement.

Je me suis donc arrêté à l'étape de la génération des exemples d'entrainement. Plus précisément, il conviendrait de résoudre le problème de proportion des formes générées. Ce qui revient à modifier le script « Formes » et par la suite, rectifier le script « Insertion » afin qu'il puisse vérifier, avant tout, que les formes peuvent entrer dans les limites du terrain. Après avoir fait cette vérification, il faudrait le modifier afin qu'il puisse bien insérer la forme dans le terrain.

2. Au sujet du script du GAN :

Lors de la mise en place de ce script, j'ai rencontré plusieurs difficultés notamment pour l'implémentation de l'environnement virtuel de programmation qui consiste entre autres, à implémenter tous les modules, fonctions et bibliothèques nécessaires au bon fonctionnement de l'algorithme mais j'ai remédié en partie à ce problème en faisant basculer tout le code dans l'environnement de travail de Google Colab qui de manière standard intègre déjà une bonne partie des bibliothèques nécessaires au bon fonctionnement du programme.

Par ailleurs, des problèmes se sont posés lors de la mise en relation entre le script du GAN et la base de données d'écriture manuscrite de J-R Piquard. J'avais besoin de faire cette connexion afin d'entrainer son modèle dans le but d'observer en temps réel le processus d'entrainement d'un GAN. Néanmoins, j'ai pu faire cette observation en regardant des exemples d'entrainement de modèle GAN dans des vidéos tutoriels sur internet.

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-5-aae3004d6e9d> in <cell line: 107>()
    105 from mlxtend.data import loadlocal_mnist
    106
--> 107 trainX, trainy = loadlocal_mnist(
    108         images_path=r'D:\train-images-idx3-ubyte',
    109         labels_path=r'D:\train-labels-idx1-ubyte')

/usr/local/lib/python3.10/dist-packages/mlxtend/data/local_mnist.py in loadlocal_mnist(images_path,
labels_path)
     35
     36     """
---> 37     with open(labels_path, "rb") as lbpath:
     38         magic, n = struct.unpack(">II", lbpath.read(8))
     39         labels = np.fromfile(lbpath, dtype=np.uint8)

FileNotFoundError: [Errno 2] No such file or directory: 'D:\\train-labels-idx1-ubyte'
```

Figure 47: Message d'erreur lors de la connexion entre le script du GAN et un exemple Base de données d'entrainement

Source : Auteur, POUHE Fahé

Après plusieurs recherches faites sur internet et tentatives de correction de bug, par ailleurs, mes connaissances en programmation d'intelligence artificielle étant limitée, je n'ai pas pu le résoudre ce qui limite la poursuite de mon expérience. Ainsi, l'entrainement du GAN n'ayant pas eu lieu dans les meilleures conditions, l'étape de la mise en place du cGAN et de son entrainement n'a pas pu, non plus, avoir lieu. Sachant que le cGAN serait basé sur l'algorithme du GAN auquel j'aurais intégré les conditions ou critères d'évaluation définis dans la méthode de recherche.

Néanmoins, la base de données que j'ai générée peut toujours être utilisée afin de poursuivre la recherche ultérieurement.

V. Résultats attendus :

L'étude que je comptais mener lors de ce mémoire n'a pas pu aboutir aux résultats que j'espérais mais je m'attendais à obtenir certains résultats spécifiques tels que :

- Une observation de fortes similarités dans les propositions générées par les GAN et cGAN car les deux modèles devraient être entrainés sur la même base de données. Ces similarités devraient être, entres autres, dues au fait que l'ensemble des données d'entrainement doit être assez élevé et varié afin d'obtenir des propositions différentes les unes des autres. Mais ne disposant pas de machine de calcul à la hauteur de celles d'entreprise comme NVIDIA, les ressources matérielles limitent la pluralité des propositions

- Le respect du critère d'évaluation « ratio de surface » pour une majorité des propositions obtenues car dans la création de la base de données, je renseignais déjà ce critère. Ainsi, si les données d'entrainement respectent déjà ce critère-là, les propositions générées ont plus de chances de le respecter.

- Le cGAN soit plus performant et génère des propositions singulières grâce aux contraintes qui lui seraient appliquées.

- Un bon respect de la fidélité de formes mais pas toujours des proportions comme indiqué dans la méthode de recherche.

- L'obtention de propositions inattendues qui ne respectent ni la fidélité de forme, ni le ratio de surface qui pourraient être enrichissantes pour l'étude.

CONCLUSION

Ce que je tire de ce mémoire :

Au cours de cette étude, j'ai eu l'opportunité captivante d'explorer le fonctionnement des intelligences artificielles génératives de type GAN, plongeant dans le monde complexe des réseaux de neurones. La compréhension approfondie de ces mécanismes a constitué une étape cruciale dans l'accroissement de mes compétences en programmation de réseaux de neurones. La création d'une base de données d'entraînement a également été un volet essentiel de cette démarche, me permettant de saisir pleinement le rôle crucial de données de qualité dans le développement et la formation des modèles.

Cette étude a également mis en lumière les enjeux profonds liés à l'utilisation de l'intelligence artificielle en architecture. J'ai pris conscience des implications tant créatives que éthiques de l'intégration de l'IA dans le domaine architectural. Mon objectif personnel, résultant de cette exploration, est de me familiariser davantage avec les IA génératives appliquées à l'architecture. Ainsi cette connaissance approfondie deviendra une ressource inestimable pour mes expérimentations futures, avec l'ambition ultime de les intégrer de manière fluide et innovante dans mon processus de conception en architecture. De ce fait, cette étude marque non seulement une étape importante dans mon parcours académique, mais également le point de départ d'une exploration continue et créative des possibilités offertes par l'IA dans le domaine de l'architecture.

Pistes d'amélioration :

Plusieurs pistes d'amélioration sont envisageables notamment la plus importante serait de continuer ce sujet en veillant à faire les corrections des différents bugs d'algorithme, de base de données et de liaison des environnements que j'ai pu mentionner précédemment. Il s'agit de l'amélioration est la plus simple à réaliser sachant que la démarche scientifique de l'étude, l'état de l'art, les enjeux de cette étude et la base de données d'entrainement du GAN et cGAN ont été déjà réalisés dans ce mémoire.

Enfin, lors de la réalisation de ce mémoire, j'ai pu me rendre compte qu'il était possible d'aboutir quasiment aux mêmes résultats de génération de propositions de modèle d'implantation soumis à des contraintes sans passer par les GAN ou cGAN. En effet, il est possible de réaliser uniquement cette étude grâce au paramétrisme à travers Grasshopper, GhPython et Rhino. Cela permettrait donc aux personnes qui sont familières à ces outils de paramétrisme de l'expérience.

BIBLIOGRAPHIE

❖ Travaux de recherche

PIQUARD, Jean-Raphael. (2020). L'apprentissage machine au service de la conception architecturale: Comment enrichir un espace de solution paramétrique grâce aux réseaux antagonistes génératifs ?

Chaillou, Stanislas. (2020). ArchiGAN: Artificial Intelligence x Architecture. Dans Philip F. Yuan, Mike Xie, Neil Leach, Jiawei Yao et Xiang Wang (dir.), Architectural Intelligence: Selected Papers from the 1st International Conference on Computational Design and Robotic Fabrication **(CDRF** 2019) @. 117-127). Springer Nature. https://doi.org/10.1007/978-981- 15-6568-7_8

Huang, W., **& Zheng, H. (2018).** Architectural drawings recognition and generation through machine learning (p 156-165) 2018 ACADIA PAPER Proceedings Final.indd (cumincad.org)

Peters, N. (2017). Master thesis: "Enabling alternative architectures: Collaborative frameworks for participatory design."

Martinez, N. (2016). Suggestive drawing among human and artificial intelligences.

Islam, Jyoti et Zhang, Yanqing. (2020). GAN-based synthetic brain PET image generation. *Brain Informatics*, *7*(1), 3. https://doi.org/10.1186/s40708-020-00104-2

Kowalski, Marek, Garbin, Stephan J., Estellers, Virginia, Baltrušaitis, Tadas, Johnson, Matthew et Shotton, Jamie. (2020). CONFIG: Controllable Neural Face Image Generation. Dans Andrea Vedaldi, Horst Bischof, Thomas Brox et Jan-Michael Frahm (dir.), *Computer Vision – ECCV 2020* (p. 299-315). Springer International Publishing. https://doi.org/10.1007/978-3-030-58621-8_18

Loey, Mohamed, Smarandache, Florentin et M. Khalifa, Nour Eldeen. (2020). Within the Lack of Chest COVID-19 X-ray Dataset: A Novel Detection Model Based on GAN and Deep Transfer Learning. *Symmetry*, *12*(4), 651. https://doi.org/10.3390/sym12040651

Tang, Hao, Xu, Dan, Yan, Yan, Torr, Philip H.S. et Sebe, Nicu. (2020). Local Class-Specific and Global Image-Level Generative Adversarial Networks for Semantic-Guided Scene Generation. Dans *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (p. 7867-7876). IEEE. https://doi.org/10.1109/CVPR42600.2020.00789

Xu, Dongwei, Wei, Chenchen, Peng, Peng, Xuan, Qi et Guo, Haifeng. (2020). GE-GAN: A novel deep learning framework for road traffic state estimation. *Transportation Research Part C: Emerging Technologies*, *117*, 102635. https://doi.org/10.1016/j.trc.2020.102635

Agarwal, Neetima, Chauhan, Sumedha, Kar, Arpan Kumar et Goyal, Sandeep. (2017). Role of human behaviour attributes in mobile crowd sensing: a systematic literature review. Digital Policy, Regulation and Governance, 19(2), 168-185. https://doi.org/10.1108/DPRG-05-2016-0023

Aggarwal, Alankrita, Mittal, Mamta et Battineni, Gopi. (2021). Generative adversarial network: An overview of theory and applications. International Journal of Information Management Data Insights, 1(1), 100004. https://doi.org/10.1016/j.jjimei.2020.100004

Gaël. (2018, 25 février). 3 Algorithmes de DeepLearning expliqués en Langage Humain. Datakeen. https://www.datakeen.co/3-deep-learning-architectures-explained-in-human-language-2/

Karras, Tero, Laine, Samuli et Aila, Timo. (2019, 29 mars). A Style-Based Generator Architecture for Generative Adversarial Networks (arXiv:1812.04948). arXiv. Récupéré le 8 janvier 2024 de http://arxiv.org/abs/1812.04948

Ma, Qianli, Yang, Jinlong, Ranjan, Anurag, Pujades, Sergi, Pons-Moll, Gerard, Tang, Siyu et Black, Michael J. (2020). Learning to Dress 3D People in Generative Clothing. Dans 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (p. 6468-6477). IEEE. https://doi.org/10.1109/CVPR42600.2020.00650

Mirza, Mehdi et Osindero, Simon. (2014, 6 novembre). Conditional Generative Adversarial Nets (arXiv:1411.1784). arXiv. Récupéré le 21 mai 2023 de http://arxiv.org/abs/1411.1784

Radford, Alec, Metz, Luke et Chintala, Soumith. (2016, 7 janvier). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks (arXiv:1511.06434). arXiv. https://doi.org/10.48550/arXiv.1511.06434

Singh, Harjit, Grover, Purva, Kar, Arpan Kumar et Ilavarasan, P. Vigneswara. (2020). Review of performance assessment frameworks of e-government projects. Transforming Government: People, Process and Policy, 14(1), 31-64. https://doi.org/10.1108/TG-02-2019-0011

Sixt, Leon, Wild, Benjamin et Landgraf, Tim. (2018). RenderGAN: Generating Realistic Labeled Data. Frontiers in Robotics and AI, 5. https://www.frontiersin.org/articles/10.3389/frobt.2018.00066

❖ Ressources en ligne

Team, Keras. (consulté en Décembre 2022). Keras documentation: Conditional GAN. https://keras.io/examples/generative/conditional_gan/

Generative Adversarial Networks: Basics & 4 Popular Extensions. (consulté en Décembre 2022**.).** Datagen. https://datagen.tech/guides/computer-vision/generative-adversarial-networks/

Maynard-Reid, Margaret. (2022, 13 septembre). Intro to Generative Adversarial Networks (GANs). PyImageSearch. https://pyimagesearch.com/2021/09/13/intro-to-generative-adversarial-networks-gans/

Brownlee, Jason. (2023, 12 mars). How to Setup Your Python Environment for Machine Learning with Anaconda. MachineLearningMastery.com. https://machinelearningmastery.com/setup-python-environment-machine-learning-deep-learning-anaconda/

Autoencoders in Deep Learning: Tutorial & Use Cases [Décembre 2022]. (s. d.). https://www.v7labs.com/blog/autoencoders-guide, https://www.v7labs.com/blog/autoencoders-guide

Réseaux convolutifs (CNN) : comment ça marche ? (consulté en Décembre 2022) Formation Tech et Data en ligne | Blent.ai. https://blent.ai/blog/a/cnn-comment-ca-marche

❖ Livres et conferences

Stanislas Chaillou, Intélligence et Architecture, consulté en Décembre 2022, https://www.youtube.com/watch?v=xNW_UnSIrqk&pp=ygUbaWEgZXQgYXJjaGloZWN0dXJlIGNoYWlsbG91

GLOSSAIRE

Intelligence Artificielle (IA) : La science et l'ingénierie de la fabrication de machines intelligentes capables de réaliser des tâches qui nécessitent des processus mentaux de haut niveau, généralement effectuées par des êtres humains.

Algorithme : Un algorithme est une séquence finie et non ambiguë d'instructions permettant de résoudre un problème ou d'effectuer une tâche donnée.

Deep Learning : Une branche de l'intelligence artificielle qui utilise des réseaux neuronaux artificiels pour résoudre des problèmes complexes en apprenant à partir de données.

Modularité : La notion de créer un modèle d'architecture idéal qui est ensuite multiplié pour former un ensemble d'architecture, réduisant ainsi les temps et coûts de conception et de construction.

DAO (Dessin Assisté par Ordinateur) : L'utilisation de logiciels informatiques pour simplifier et accélérer le processus de dessin et de représentation en architecture.

Paramétrisme : Un principe de synthétisation et de décomposition des différentes étapes nécessaires à la réalisation d'un design, permettant l'application de paramètres ajustables à chaque étape du processus de conception.

Grasshopper : Un environnement de modélisation algorithmique utilisé dans la conception architecturale, souvent associé au logiciel Rhino.

Python : Un langage de programmation largement utilisé, particulièrement dans le domaine de l'apprentissage machine et de l'IA.

Bibliothèque : Une bibliothèque, dans le contexte de la programmation informatique, est un ensemble de fonctions pré-écrites qui peuvent être utilisées pour effectuer des tâches spécifiques. Les bibliothèques peuvent inclure des fonctions pour divers domaines tels que la manipulation de fichiers, les opérations mathématiques, la manipulation de chaînes de caractères, la visualisation de données, l'interaction avec des bases de données, etc.

Auto-encodeurs (AE) : Des modèles de machine learning qui apprennent à représenter les données en les compressant dans un format compact, puis en les décompressant pour les reconstruire.

Réseaux de Neurones Convolutifs (CNN) : Des réseaux de neurones artificiels utilisés pour la reconnaissance d'images et la vision par ordinateur, efficaces pour détecter des patterns dans les images.

Réseaux de Neurones Récurrents (RNN) : Des réseaux de neurones adaptés au traitement de données séquentielles, tels que les séquences de mots ou de sons, capables de prendre en compte l'historique des données.

Réseaux Adversaires Génératifs (GAN) : Une classe de cadres d'apprentissage automatique où deux réseaux neuronaux, un générateur et un discriminateur, sont en concurrence pour créer des artefacts réalistes indiscernables des artefacts réels.

Conditionals GAN (cGAN) : Une catégorie de GAN qui permet d'avoir plus de contrôle sur les résultats générés en introduisant des conditions supplémentaires, tels que des étiquettes de classe, dans le processus d'apprentissage.

GAN Pix2Pix : Une technique de GAN utilisée pour la conversion d'images d'un domaine à un autre, par exemple, transformer des dessins en images réalistes.

VAE-GAN (Variational Autoencoder Generative Adversarial Network) est une architecture de réseau de neurones qui combine les techniques de l'autoencodeur variationnel (VAE) et des réseaux génératifs adversaires (GAN) pour générer des données réalistes à partir d'un espace latent. Cette méthode permet d'apprendre une représentation dense des données d'entrée et de générer de nouvelles données tout en conservant leurs caractéristiques essentielles.

SMPL (Simplified Human Body Model) est un modèle de corps humain simplifié largement utilisé dans les domaines de la vision par ordinateur, de la réalité virtuelle et de la synthèse d'images pour représenter la forme et la pose du corps humain de manière paramétrique. Il offre une représentation compacte et expressive de la forme du corps, permettant une variété d'applications dans la modélisation et l'animation 3D.

Nvidia : Nvidia est une société spécialisée dans la conception de cartes graphiques, de processeurs graphiques et de systèmes intégrés utilisés dans les domaines du jeu vidéo, de la visualisation professionnelle, de l'intelligence artificielle et du calcul haute performance.

ANNEXES

Code informatique

Script du GAN :

```
# example of training a stable gan for generating a handwritten digit
from os import makedirs

# Vérifier la présence de Keras
import keras
print(keras.__version__)

# Vérifier la présence de NumPy
import numpy as np
print(np.__version__)

# Vérifier la présence de Matplotlib
import matplotlib
print(matplotlib.__version__)

# Vérifier la présence de mlxtend
import mlxtend
print(mlxtend.__version__)

from numpy import expand_dims
from numpy import zeros
from numpy import ones
from numpy.random import randn
from numpy.random import randint
from keras.datasets.mnist import load_data
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Reshape
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import Conv2DTranspose
from keras.layers import LeakyReLU
from keras.layers import BatchNormalization
from keras.initializers import RandomNormal
from matplotlib import pyplot
```

```
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # define model
  model = Sequential()
  # downsample to 14x14
  model.add(Conv2D(64, (4,4), strides=(2,2), padding='same',
kernel_initializer=init, input_shape=in_shape))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # downsample to 7x7
  model.add(Conv2D(64, (4,4), strides=(2,2), padding='same',
kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
  return model

# define the standalone generator model
def define_generator(latent_dim):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # define model
  model = Sequential()
  # foundation for 7x7 image
  n_nodes = 128 * 7 * 7
  model.add(Dense(n_nodes, kernel_initializer=init, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((7, 7, 128)))
  # upsample to 14x14
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same',
kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 28x28
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same',
kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # output 28x28x1
```

```
  model.add(Conv2D(1, (7,7), activation='tanh', padding='same',
kernel_initializer=init))
  return model

# define the combined generator and discriminator model, for updating the
generator
def define_gan(generator, discriminator):
  # make weights in the discriminator not trainable
  discriminator.trainable = False
  # connect them
  model = Sequential()
  # add generator
  model.add(generator)
  # add the discriminator
  model.add(discriminator)
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt)
  return model


import mlxtend
import numpy as np

from mlxtend.data import loadlocal_mnist

trainX, trainy = loadlocal_mnist(
        images_path=r'Z:\Fahé POUHE\Mémoire - Fahé\test-images-idx3-ubyte',
        labels_path=r'Z:\Fahé POUHE\Mémoire - Fahé\test-labels-idx1-ubyte')

print(trainX.shape[0], trainX.shape[1])
trainX=np.reshape(trainX,(trainX.shape[0],28,-1))
print(trainX.shape[0], trainX.shape[1], trainX.shape[2])

# load mnist images
def load_real_samples():
  # load dataset
  # expand to 3d, e.g. add channels

  X = expand_dims(trainX, axis=-1)
  # select all of the examples for a given class
  selected_ix = trainy == 8
  X = X[selected_ix]
  # convert from ints to floats
  X = X.astype('float32')
  # scale from [0,255] to [-1,1]
  X = (X - 127.5) / 127.5
  return X
```

```
# select real samples
def generate_real_samples(dataset, n_samples):
  # choose random instances
  ix = randint(0, dataset.shape[0], n_samples)
  # select images
  X = dataset[ix]
  # generate class labels
  y = ones((n_samples, 1))
  return X, y

# generate points in latent space as input for the generator
def generate_latent_points(latent_dim, n_samples):
  # generate points in the latent space
  x_input = randn(latent_dim * n_samples)
  # reshape into a batch of inputs for the network
  x_input = x_input.reshape(n_samples, latent_dim)
  return x_input

# use the generator to generate n fake examples, with class labels
def generate_fake_samples(generator, latent_dim, n_samples):
  # generate points in latent space
  x_input = generate_latent_points(latent_dim, n_samples)
  # predict outputs
  X = generator.predict(x_input)
  # create class labels
  y = zeros((n_samples, 1))
  return X, y

# generate samples and save as a plot and save the model
def summarize_performance(step, g_model, latent_dim, n_samples=100):
  # prepare fake examples
  X, _ = generate_fake_samples(g_model, latent_dim, n_samples)
  # scale from [-1,1] to [0,1]
  X = (X + 1) / 2.0
  # plot images
  for i in range(10 * 10):
    # define subplot
    pyplot.subplot(10, 10, 1 + i)
    # turn off axis
    pyplot.axis('off')
    # plot raw pixel data
    pyplot.imshow(X[i, :, :, 0], cmap='gray_r')
  # save plot to file
  pyplot.savefig('results_baseline/generated_plot_%03d.png' % (step+1))
  pyplot.close()
  # save the generator model
  g_model.save('results_baseline/model_%03d.h5' % (step+1))
```

```
# create a line plot of loss for the gan and save to file
def plot_history(d1_hist, d2_hist, g_hist, a1_hist, a2_hist):
  # plot loss
  pyplot.subplot(2, 1, 1)
  pyplot.plot(d1_hist, label='d-real')
  pyplot.plot(d2_hist, label='d-fake')
  pyplot.plot(g_hist, label='gen')
  pyplot.legend()
  # plot discriminator accuracy
  pyplot.subplot(2, 1, 2)
  pyplot.plot(a1_hist, label='acc-real')
  pyplot.plot(a2_hist, label='acc-fake')
  pyplot.legend()
  # save plot to file
  pyplot.savefig('results_baseline/plot_line_plot_loss.png')
  pyplot.close()

# train the generator and discriminator
def train(g_model, d_model, gan_model, dataset, latent_dim, n_epochs=10,
n_batch=50):
  # calculate the number of batches per epoch
  bat_per_epo = 10
  # calculate the total iterations based on batch and epoch
  n_steps = bat_per_epo * n_epochs
  # calculate the number of samples in half a batch
  half_batch = int(n_batch / 2)
  # prepare lists for storing stats each iteration
  d1_hist, d2_hist, g_hist, a1_hist, a2_hist = list(), list(), list(),
list(), list()
  # manually enumerate epochs
  for i in range(n_steps):
    # get randomly selected 'real' samples
    X_real, y_real = generate_real_samples(dataset, half_batch)
    # update discriminator model weights
    d_loss1, d_acc1 = d_model.train_on_batch(X_real, y_real)
    # generate 'fake' examples
    X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
    # update discriminator model weights
    d_loss2, d_acc2 = d_model.train_on_batch(X_fake, y_fake)
    # prepare points in latent space as input for the generator
    X_gan = generate_latent_points(latent_dim, n_batch)
    # create inverted labels for the fake samples
    y_gan = ones((n_batch, 1))
    # update the generator via the discriminator's error
    g_loss = gan_model.train_on_batch(X_gan, y_gan)
    # summarize loss on this batch
    print('>%d, d1=%.3f, d2=%.3f g=%.3f, a1=%d, a2=%d' %
      (i+1, d_loss1, d_loss2, g_loss, int(100*d_acc1), int(100*d_acc2)))
```

```
    # record history
    d1_hist.append(d_loss1)
    d2_hist.append(d_loss2)
    g_hist.append(g_loss)
    a1_hist.append(d_acc1)
    a2_hist.append(d_acc2)
    # evaluate the model performance every 'epoch'
    if (i+1) % bat_per_epo == 0:
      summarize_performance(i, g_model, latent_dim)
  plot_history(d1_hist, d2_hist, g_hist, a1_hist, a2_hist)

# make folder for results
makedirs('results_baseline', exist_ok=True)
# size of the latent space
latent_dim = 50
# create the discriminator
discriminator = define_discriminator()
# create the generator
generator = define_generator(latent_dim)
# create the gan
gan_model = define_gan(generator, discriminator)
# load image data
dataset = load_real_samples()
print(dataset.shape)
# train model
train(generator, discriminator, gan_model, dataset, latent_dim)
```

Programme de Grasshopper

Script GhPython :

Script "Formes"

```
# SCRIPT FORMES

import rhinoscriptsyntax as rs

# Entrée manuelle depuis Grasshopper pour le rectangle x
x = x  # Utilise la variable x stockée dans le contexte sc

# Entrée manuelle depuis Grasshopper pour le nombre de rectangles y
y = int(y)  # Convertit y en un nombre entier

# Initialise la liste des formes générées
generated_shapes = []

# Vérifie si x est un rectangle
if rs.IsCurve(x) and rs.IsCurvePlanar(x):
    if rs.IsCurveClosed(x):
        # Calcule la superficie du rectangle x
        x_area = rs.Area(x)

        # Calcule la superficie cible des formes (60% de la superficie de x)
        target_shape_area = 0.6 * x_area

        # Boucle pour générer les formes
        for i in range(y):
            # Calcule la longueur et la largeur du rectangle pour atteindre la
superficie cible
            current_rectangle_area = 0
            current_length = rs.CurveLength(x)  # Longueur initiale

            while current_rectangle_area < target_shape_area:
                current_length += 1
                current_width = target_shape_area / current_length
                current_rectangle_area = current_length * current_width

            # Crée le rectangle
            shape = rs.AddRectangle([0,0,0], current_length, current_width)

            if shape:
                # Ajoute le rectangle à la liste des rectangles générés
                generated_shapes.append(shape)
            else:
                print("Impossible de créer un rectangle.")
    else:
        print("L'objet sélectionné n'est pas un rectangle fermé.")
```

```
else:
    print("Veuillez sélectionner un objet de type courbe (rectangle) en entrée.")

# Sortie "a" contenant les rectangles générés
a = generated_shapes
```

Script "Terrains"

```
import rhinoscriptsyntax as rs

# Entrée manuelle depuis Grasshopper pour le rectangle x
x = x  # Utilise la variable x stockée dans le contexte sc

# Entrée manuelle depuis Grasshopper pour le nombre de rectangles y
y = int(y)  # Convertit y en un nombre entier

# Initialise la liste des objets dupliqués
duplicated_rectangles = []

# Vérifie si x est un rectangle
if rs.IsCurve(x) and rs.IsCurvePlanar(x):
    if rs.IsCurveClosed(x):
        # Boucle pour dupliquer le rectangle x
        for i in range(y):
            # Duplique le rectangle x
            duplicated_rectangle = rs.CopyObject(x, [i * 25, 0, 0])  # Ajuste la translation selon les besoins
            duplicated_rectangles.append(duplicated_rectangle)
    else:
        print("L'objet sélectionné n'est pas un rectangle fermé.")
else:
    print("Veuillez sélectionner un objet de type courbe (rectangle) en entrée.")

# Sortie "a" contenant les objets dupliqués
a = duplicated_rectangles
```

Script "Insertion"

```
# SCRIPT INSERTIONS

import rhinoscriptsyntax as rs

# Entrée manuelle depuis Grasshopper pour les terrains (rectangles)
terrains = x  # Utilise la variable terrains stockée dans le contexte sc

# Entrée manuelle depuis Grasshopper pour les formes générées
formes_generees = y  # Utilise la variable formes_generees stockée dans le
contexte sc

# Ajoutez ces sorties pour imprimer les types et le contenu des variables
print(type(terrains))
print(type(formes_generees))

# Vérifie si les terrains et les formes générées sont des listes
if isinstance(terrains, list) and isinstance(formes_generees, list):
    # Vérifie si les terrains et les formes générées sont valides
    if terrains and formes_generees:
        # Boucle pour positionner les formes à l'intérieur des terrains
        for terrain, forme_generee in zip(terrains, formes_generees):
            # Vérifie si les terrains et les formes sont des courbes
            if rs.IsCurve(terrain) and rs.IsCurve(forme_generee):
                # Copie la forme générée à l'intérieur du terrain
                rs.CopyObject(forme_generee, translation=[0,0,0], copy=True)
            else:
                print("Les objets sélectionnés ne sont pas des courbes.")
    else:
        print("Veuillez fournir les terrains et les formes générées en
entrée.")
else:
    print("Les entrées ne sont pas valides. Assurez-vous que terrains et
formes_generees sont des listes.")
```

```
# define the standalone discriminator model
def define_discriminator(in_shape=(28,28,1)):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # define model
  model = Sequential()
  # downsample to 14x14
  model.add(Conv2D(64, (4,4), strides=(2,2), padding='same',
kernel_initializer=init, input_shape=in_shape))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # downsample to 7x7
  model.add(Conv2D(64, (4,4), strides=(2,2), padding='same',
kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # classifier
  model.add(Flatten())
  model.add(Dense(1, activation='sigmoid'))
  # compile model
  opt = Adam(lr=0.0002, beta_1=0.5)
  model.compile(loss='binary_crossentropy', optimizer=opt,
metrics=['accuracy'])
  return model

# define the standalone generator model
def define_generator(latent_dim):
  # weight initialization
  init = RandomNormal(stddev=0.02)
  # define model
  model = Sequential()
  # foundation for 7x7 image
  n_nodes = 128 * 7 * 7
  model.add(Dense(n_nodes, kernel_initializer=init, input_dim=latent_dim))
  model.add(LeakyReLU(alpha=0.2))
  model.add(Reshape((7, 7, 128)))
  # upsample to 14x14
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same',
kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # upsample to 28x28
  model.add(Conv2DTranspose(128, (4,4), strides=(2,2), padding='same',
kernel_initializer=init))
  model.add(BatchNormalization())
  model.add(LeakyReLU(alpha=0.2))
  # output 28x28x1
  model.add(Conv2D(1, (7,7), activation='tanh', padding='same',
kernel_initializer=init))
  return model
```