République Algérienne Démocratique Et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

UNIVERSITE SAAD DAHLEB DE BLIDA

Faculté Des Sciences
Département d'Informatique

MEMOIRE DE MASTER EN INFORMATIQUE
Option : Système Informatique et Réseaux

APPRENTISSAGE DES REPRESENTATIONS DES FORMES 3D

Encadré par:
Mr.A.HICHAM KAMECHE

Réalisé par :
MEZHOUDA Hadjer
SALMI Farah

Devant le Jury:
CHERIF Zahar Amine (Président)
MEZZI Melyara (Examinatrice)

Session 2021-2022

Remerciement

Aujourd'hui, suite à la clôture de notre parcours universitaire, nous tenons à noter que cette année fut la plus marquante de toutes.

Nous remercierons en premier lieu le **Dieu**, le tout puissant de nous avoir donné le courage et la patience pour effectuer notre travail, en suite nos parents.

Nous adressons aussi notre sincère sentiment de gratitude à notre encadreur **Mr.KAMECHE Abdellah Hicham** pour ses conseils et encouragement.

Nous souhaitons exprimer nos remerciements à tous les enseignants du département informatique de Blida qui ont assuré notre formation Nous tenons également à remercier toute personne ayant contribué de près ou de loin à la concrétisation de ce mémoire.

Merci

Hadjer et Farah

Dédicace

J'ai le grand plaisir de dédier ce modeste travail

À mes chers **parents**, qui n'ont jamais cessé de prier pour moi, m'encouragents, me soutenir, surtout pour leurs amour et sacrifice.

À mon frère **Saad Eddine**, mes sœurs **Khawla** et **Meyssoune** pour leur soutien durant toute la période de travail.

À toute ma **famille**, **mes proches** et a ceux qui me donnent de l'amour et de la vivacité.

À mon binôme **Farah** qui a été présente avec moi durant toute cette période afin de faire du bon travail.

Et enfin à tous les **étudiants** de département d'informatique.

MEZHOUDA HADJER

Dédicace

J'ai le grand plaisir de dédier ce modeste travail

À mes chers **parents**, qui n'ont jamais cessé de prier pour moi, m'encourage, me soutenir, surtout pour leurs amour et sacrifice.

À mon frère **Akram** et ma sœur **Nahla** pour leur soutien durant toute la période de travail.

À mon **Mari** qui m'a toujours encouragé je lui souhaite plus de bonheur et de succès.

À toute ma **famille**, mes **proches** et a ceux qui me donne de l'amour et de la vivacité.

À mon binôme **Hadjer** qui a été présente avec moi durant toute cette période afin de faire du bon travail.

Et enfin à tous les **étudiants** de département d'informatique.

Une grande dédicace à ma fille **Djihane** que dieu te protège pour moi ma fille.

SALMI FARAH

Résumé

La génération des formes 3D est un domaine très difficile et qui intéressent beaucoup de chercheurs et qui nécessite une bonne représentation des formes 3D. Notre travail s'articule sur la représentation des formes 3D dans un contexte de génération à partir d'une description textuelle, en s'appuyant sur la puissance des réseaux de neurones antagonistes génératifs (GANs).

Nous avons proposé deux architectures différentes afin de représenter les formes 3D. la première se base sur le modèle Autoencoder. La seconde exploite les modèles des Transformers utilisés en TAL. Nous avons mené deux types d'expérimentations: intrinsèque où nous avons testé nos modèles et extrinsèque où nous avons mis en épreuve nos modèles dans une application bien précise.

Les WGANs sont de plus en plus connus comme une forme évoluée d'apprentissage automatique. Des chercheurs et des développeurs ont expérimenté l'utilisation de WGANs pour produire des copies, même imparfaites, d'œuvres célèbres telles que la Joconde et des portraits de personnes qui n'existent pas.

Mot clés : 3D embedding, Autoencodeur, Transformateurs, WGANs

---

Abstract

The generation of 3D shapes is a very difficult field which interests many researchers, and which requires a good representation of 3D shapes. Our work focuses on the representation of 3D shapes in a generation context from a textual description, relying on the power of generative antagonistic neural networks (GANs).

We proposed two different architectures to represent 3D shapes. the first is based on the Autoencoder model. The second exploits the Transformers models used in NLP. We conducted two types of experiments: intrinsic where we tested our models and extrinsic where we tested our models in a specific application.

WGAs are increasingly known as an advanced form of machine learning. Researchers and developers have experimented with using WGANs to produce even imperfect copies of famous works such as the Mona Lisa and portraits of people who don't exist

---

Keywords : 3D embedding, Autoencoder, Transformer, WGANs

Table Des Matières

Liste Des Figures

Liste Des Tableaux

Liste des Abréviations

Abréviation	Description complète
CNN	Convolutional neural network
	Réseau de neurone Convolutionnel
FNN	Feed-forward neural network
	Réseau de neurones à réaction
RNN	Recurrent neural network
	Réseau neuronal récurrent
GAN	Generative adversarial networks
	Réseaux antagonistes génératifs
WGAN	Wasserstein Generative Adversarial Networks
	Réseau Antagoniste Génératif Wasserstein
BERT	Bidirectional Encoder Representations from Transformers
	Représentations d'encodeurs bidirectionnels à partir de transformateurs
MVCNN	Multi-View Convolutional neural network
	Réseau neuronal convolutif multi-vues
VRN	Voxception Residual Network
	Réseau résiduel de Voxception
TAL	Natural Language Processing
	Traitement du langage naturel

Introduction Générale

L'utilisation des images et des scènes 3D est devenue monnaie courante de nos jours. On voit leur application dans beaucoup de domaines notamment avec l'émergence des nouvelles plateformes de réalités virtuelle (Meta , Amazon....). La 3D est également utilisée dans des domaines pratiques et utiles tels que l'imagerie médicale, le design immobilier, le design de véhicules ....

Les données sont tout aussi importantes que les données textuelles ou bien les images en 2D. Toutefois, la conception de scènes et de formes 3D nécessite certaines compétences et la maitrise de certains outils selon le domaine d'utilisation. De plus, leur traitement nécessite une grande puissance de calcul vu la taille imposante des formes 3D comparativement aux autres médias de données (image , texte , son). Un moyen efficace pour contourner ces problèmes est la compression et l'extraction des caractéristiques des formes 3D. Beaucoup d'algorithmes existent dans la littérature pour effectuer ces tâches, cependant, ces algorithmes ne permettent pas de recueillir et de collecter des informations sémantiques.

Notre but majeur est de concevoir une application qui permet de représenter la forme 3D et d'en extraire les caractéristiques les plus importantes pouvant être utilisées dans un domaine d'application donné.

Actuellement, l'intelligence artificielle influe sur différents domaines. Elle permet de réaliser des applications très performantes en adoptant diverses méthodes avec les différentes données notamment les formes 3D dont il existe une multitude d'études.

Les réseaux de neurones convolutifs ont montré leur efficacité dans la capture des caractéristiques profondes des images. De la même manière, les Transformers ont donné des résultats assez impressionnant pour la génération des words embeddings. Dans notre travail, on essaiera d'exploiter ces deux types d'architectures sur les données 3D. Les principales tâches sont :

- Proposer un modèle à base de réseaux de neurones pour apprendre les représentations des formes 3D.
- Valider le modèle en utilisant un vrai dataset.
- Exploiter le modèle dans un autre projet : la génération de formes 3D depuis une courte description textuelle

Notre mémoire est divisé en deux parties principales :la partie théorique est composé des deux premiers chapitres et la partie conception et réalisation pour le reste des chapitres.

- **Chapitre 1 :** ce chapitre est consacré à l'apprentissage automatique et les réseaux de neurones ou nous expliquerons ses différentes architectures.
- **Chapitre 2 :** dans ce chapitre nous présenterons les travaux connexes en relation avec la représentation des formes 3D.
- **Chapitre 3 :** on va détailler les méthodes utilisées ainsi que l'architecture de notre approche.
- **Chapitre 4 :** dans ce dernier chapitre on va présenter les résultats obtenus ainsi que les outils et l'environnement utilisé .

Chapitre 1

Apprentissage Automatique Et Apprentissage En Profondeur

1.1 Introduction

Ce chapitre sert de base d'informations pour ce projet. Il explique certains concepts de base de l'apprentissage automatique (ML) et de l'apprentissage en profondeur (DL), qui est une branche de ML basé sur un ensemble d'algorithmes qui tentent de modéliser les données.

La première section traite de ce qui est l'apprentissage automatique et d'une explication simplifiée de ses types. La deuxième section introduit le monde de l'apprentissage en profondeur, en commençant par l'identification et la structuration des réseaux de neurones jusqu'aux principales métriques qui seront utilisées dans l'évaluation du modèle.

1.2 Apprentissage Automatique

1.2.1 Définition

L'apprentissage automatique ou **apprentissage statistique** est un champ d'étude de l'intelligence artificielle qui se repose sur des approches mathématiques et statistiques et qui s'intéresse au type d'algorithmes qui donne la capacité à l'ordinateur à partir de données d'apprendre à résoudre un problème en faisant des essaies et en corrigeant ses erreurs.

L'algorithme d'apprentissage automatique procèdent comme suit :

- On lui fournit des donner d'entrainement D qui est l'ensemble des données entrées:

$$D = D(X_1, t_1) \dots (X_n, t_n). \tag{1.1}$$

  Où:

  - **X :** l'entrée (l'information sur le caractère entré).
  - **t :** la cible (la réponse à quelle classe correspond l'entrée).

- L'algorithme retourne un programme capable de généraliser de nouvelles données.
- Le programme générer est noter Y(X) qu'on va appeler modèle.

1.2.2 Les Types D'apprentissage Automatique

1.2.2.1 Apprentissage Supervisé

L'apprentissage supervisé est une approche d'apprentissage qui s'intéresse aux données étiquetées ou on trouve dans l'ensemble d'entrainement D(X) une cible à viser. L'objectif est de prédire l'étiquette (inconnue) Y associée à une nouvelle

observation X, à partir de la connaissance fournie par l'ensemble d'entrainement D(X). On distingue deux types de problème [1].

- **Classification:** Lorsque les étiquettes Yn prennent leurs valeurs dans un ensemble fini dont les éléments correspondent à des catégories (ou classes) à identifier, on parle de classification supervisée
  Exemple reconnaissance de caractère:
  - **X:** vecteur des intensités de tous les pixels d'une image
  - **t:** identité du caractère
- **Régression:** Lorsque les étiquettes Yn prennent des valeurs scalaires ou vectorielles, on parle de problème de régression Exemple prédiction de valeur d'une action a la bource :
  - **X:** vecteur contenant des informations sur l'activité économique de la journée
  - **t:** valeur d'une action à la bourse le lendemain

1.2.2.2 Apprentissage Non Supervisé

L'apprentissage non supervisé est l'approche qui traite des données non-étiquetées ou la cible est absente dans l'ensemble d'entrainement D(X) pas de cible t. L'objectif est d'identifier automatiquement des caractéristiques communes aux observations, les techniques d'apprentissage non supervisé peuvent être utilisées pour résoudre, entre autres, les problèmes suivants : [1]

- **Partitionnement de données(clustering)**
  Un algorithme qui regroupe les caractères qui se ressemble sans préciser les cibles.

Exemple: entrant un ensemble de caractères de deux nombres différents et qui nous donne en sortie deux ensemble contenant chaque un l'ensemble des caractères du même nombre.

- **Visualisation des données**

  Un algorithme qui permet de prendre les données du vecteur X entrant et les compresser en deux dimensions.

  Exemple: éxécuter un tel algorithme sur un ensemble de données qui a réussi à illustrer dans ces données on a le même visage mais vari sur un axe qui change l'orientation du visage.

- **Estimation de densité**

  Apprendre la loi de probabilité P(X) qui génère les données pour:

  – Générer de nouvelles données réalistes.

  – Distinguer les vraies données des fausses données.

1.2.2.3 Apprentissage Semi Supervisé

L'apprentissage semi-supervisé est une approche de l'apprentissage automatique qui combine une petite quantité de données étiquetées avec une grande quantité de données non étiquetées pendant la formation. Lorsque des données non étiquetées sont utilisées en combinaison avec une petite quantité de données étiquetées, la précision de la formation peut être considérablement améliorée. L'acquisition de données étiquetées pour les problèmes d'apprentissage nécessite souvent des agents humains qualifiés ou une expérience physique. Par conséquent, les coûts associés au processus d'étiquetage peuvent ne pas permettre de grands ensembles d'apprentissage entièrement étiquetés, tandis que l'obtention de données non étiquetées est relativement bon marché. Dans ce cas, l'apprentissage semi-supervisé peut être d'une grande

valeur pratique. [2]

1.2.2.4 Apprentissage Par Renforcement

L'apprentissage par renforcement consiste à entraîner des modèles d'intelligence artificielle d'une manière bien spécifique, laissez les agents autonomes apprendre les actions à entreprendre à partir de l'expérience afin d'optimiser les récompenses quantitatives au fil du temps. L'agent est immergé dans un environnement et prend des décisions en fonction de son état actuel. En retour, l'environnement fournit à l'agent une récompense, qui peut être positive ou négative. L'agent recherche un comportement décisionnel optimal (appelé politique ou politique, qui est une fonction qui associe l'état actuel à une action à effectuer) par le biais d'expériences itératives, c'est-à-dire qu'il maximise la somme des récompenses dans le temps. [2]

1.3 Apprentissage En Profondeur (Deep Learning)

1.3.1 Définition

L'apprentissage en profondeur (DL) est l'un des développements les plus importants de l'intelligence artificielle au cours des dernières années. Le développement de DL s'est produit en parallèle avec l'étude de l'intelligence artificielle, et en particulier avec l'étude des réseaux de neurones. C'était surtout dans les années 1980. Les algorithmes DL sont un ensemble de réseaux de neurones artificiels (ANN), qui peuvent faire de meilleures représentations d'ensembles de données à grande échelle, pour construire des modèles qui apprennent ces représentations de manière approfondie. Lan Goodfellow et d'autres [3] ont défini DL comme suit :

*"**L'apprentissage en profondeur** est un type particulier d'apprentissage automa-*

tique qui atteint une grande puissance et flexibilité en apprenant à représenter le monde comme une hiérarchie imbriquée de concepts, chaque concept étant défini sur des concepts plus simples et des représentations plus abstraites calculées en termes de représentations moins abstraites".

1.3.2 Architecture d'un Réseau Neuronal

1.3.2.1 Le Perceptron de Ronsenblatt

Pour bien comprendre le réseau de neurones, nous commencerons par la brique de base, le perceptron aussi nommé neurone artificiel. C'est un algorithme d'apprentissage automatique inspiré par des neurones biologiques, qui sont modélisés par une fonction linéaire, il prend une ou plusieurs entrées **(x1, x2 . . . xn)** et les combine pour générer une sortie ($\widehat{y}$). La première apparition du perceptron était dans les travaux de **Frank Rosenblatt** en 1962 [4].

Pour rendre le perceptron utilisable pour la solution des problèmes plus complexe, des poids **wi** et un biais **b** ont été ajoutés.

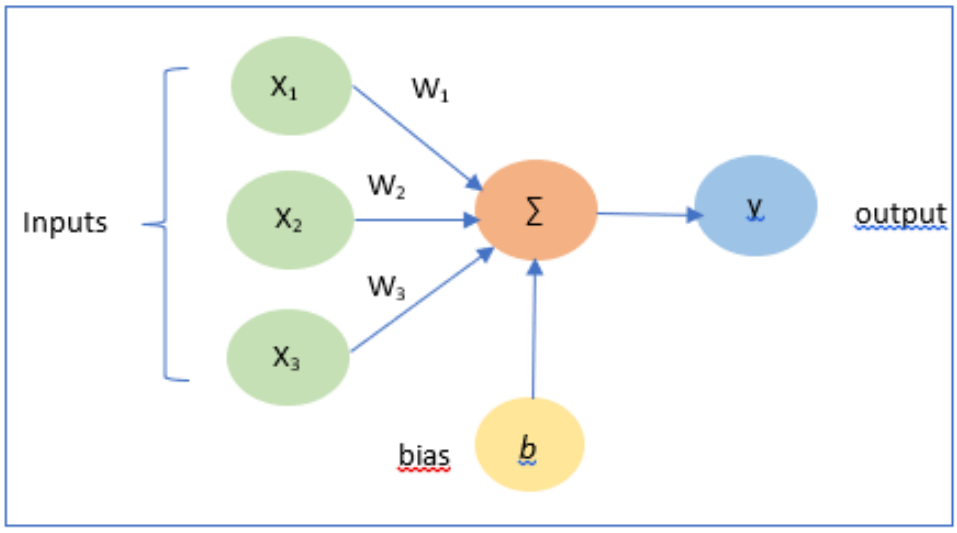

Figure 1.1: La structure d'un perceptron avec poids et biais [4]

La sortie est calculée comme suit :

$$\widehat{y} = \begin{cases} 0 & \sum wx + b < 0 \\ 1 & \sum wx + b \geqslant 0 \end{cases} \tag{1.2}$$

Où:

- **x:** la valeur d'entrée.
- **w:** la valeur du poids (Exprime l'importance de la valeur d'entrée qui lui est associée).
- **b:** c'est un paramètre qui est utilisé pour ajuster la sortie ainsi que la somme pondérée des entrées du neurone.

1.3.2.2 Les Fonctions d'activation

Pour l'instant, le neurone est une fonction linéaire. Afin que ce neurone produise des décisions non linéaires pour des cas plus complexes, on passe la sortie de sommation via une fonction non linéaire appelée **fonction d'activation** ou **fonction de transfert**. Il existe de nombreuses fonctions d'activation populaires et leur utilisation varie en fonction du problème à résoudre et de la topologie du réseau neuronal :

- **Fonction linéaire**

  Dans une **fonction d'activation linéaire**, la sortie de la fonction n'est limitée par aucune plage. Sa plage est spécifiée de l'infini à l'infini. Pour chaque neurone individuel, l'entrée est multipliée par le poids de chaque neurone correspondant, ce qui donne un signal de sortie proportionnel à l'entrée. Si toutes les couches d'entrée sont de nature linéaire, alors l'activation finale de la dernière couche sera en fait une fonction linéaire de l'entrée de la couche initiale. [5]

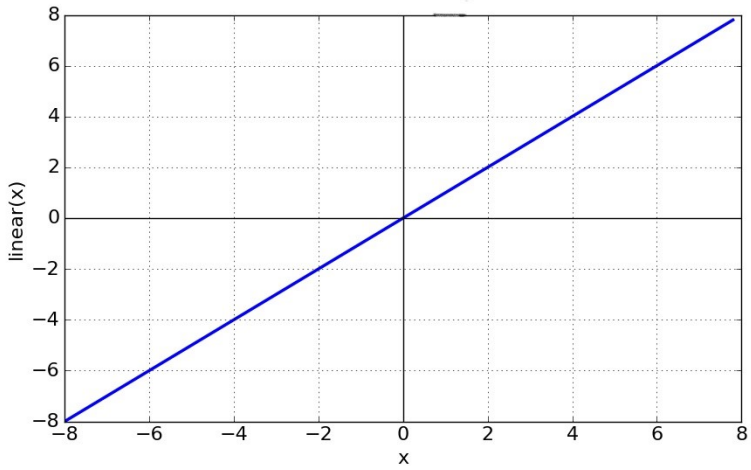

Figure 1.2: La fonction linéaire

- **Fonction non linéaire**

  C'est l'une des fonctions d'activation les plus couramment utilisées. Il aide les modèles à généraliser et à ajuster tout type de données afin d'effectuer une différenciation correcte entre les sorties. Il résout les problèmes suivants rencontrés par les fonctions d'activation linéaires, qui sont liés à la rétropropagation et à l'empilement de plusieurs couches de neurones. [5]

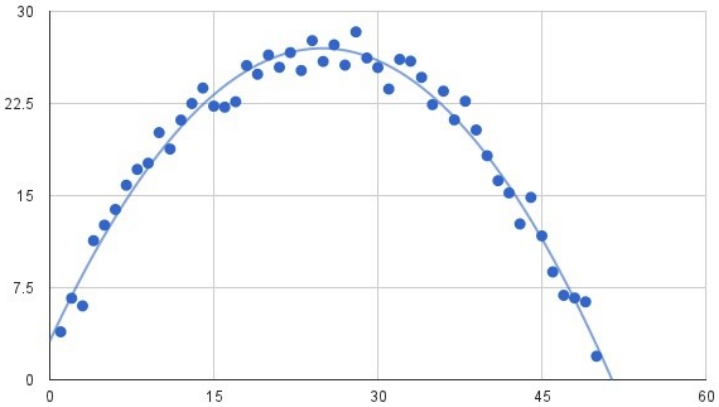

Figure 1.3: la fonction non linéaire
[5]

La fonction d'activation non linéaire est divisée en : fonction d'activation

sigmoïde ou **logistique**, fonction d'activation **Tanh** ou **tangente hyperbolique**, fonction d'activation **ReLU (unité linéaire rectifiée)** et fonction **Softmax**.

1. **La fonction sigmoïde**

   Donne une valeur entre 0 et 1, elle est donc très utilisée pour les classifications binaires, lorsqu'un modèle doit déterminer deux labels seulement. Ainsi, pour la classification des critiques de cinéma, plus la valeur retournée est proche de 1 plus la critique est considérée positive et plus elle est proche de 0, plus elle est considérée comme négative.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.3}$$

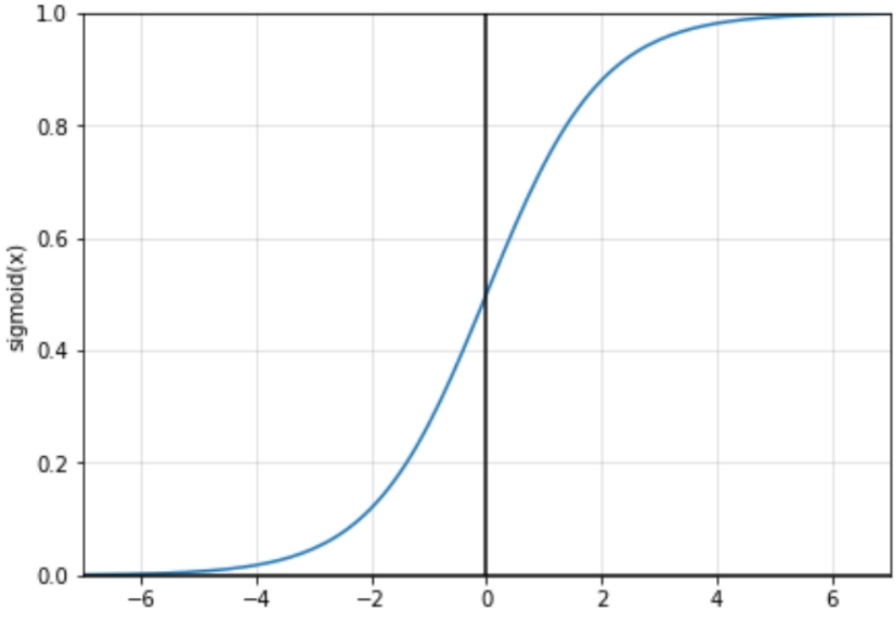

Figure 1.4: La fonction sigmoid
[5]

2. **La fonction Tanh**

La fonction de la tangente hyperbolique, elle s'agit d'une version mathématiquement décalée de la fonction sigmoïde. La fonction Tanh donne un résultat entre -1 et 1 son avantage est que les entrées négatives sont bien répertoriées comme négatives. Cette fonction est utilisée dans la classification binaire plus la valeur retourne par Tanh est proche de 1 plus le modèle considère la critique est positive, plus elle est proche de -1 plus elle est considérée comme négative.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)}$$
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.4}$$

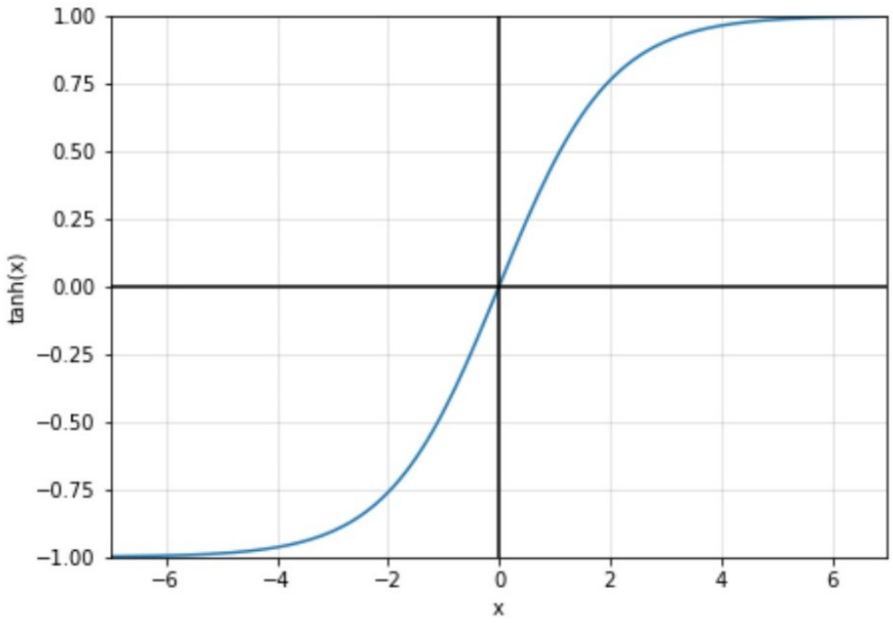

Figure 1.5: La fonction tanh
[5]

3. **La fonction ReLU (Rectified Linear Unit)**

La fonction d'activation la plus simple et la plus utilisée. Elle donne x si x est supérieur à 0, 0 sinon. Autrement dit, c'est le maximum entre x et 0:

$$ReLU(x) = \max(x, 0) \tag{1.5}$$

Cette fonction permet d'effectuer un filtre sur nos données. Elle laisse passer que les valeurs positives dans la couche suivante du réseau de neurones. Elle n'est utilisée que dans les couches intermédiaires mais surtout pas dans la couche finale.

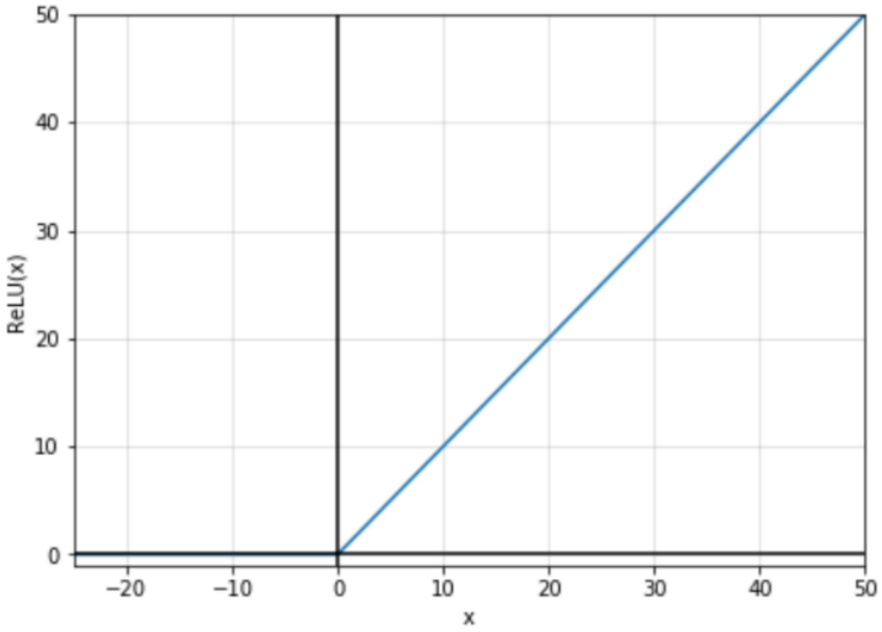

Figure 1.6: La fonction ReLU
[5]

4. **La fonction Softmax**

permet de transformer un vecteur réel en vecteur de probabilité. Elle est utilisée souvent dans la couche finale d'un modèle de classification, notamment pour les problèmes multiclasse. Dans cette fonction, chaque

vecteur est traité indépendamment. L'argument axis définit l'axe d'entrée sur lequel la fonction est appliquée.

$$softmax(x) = \frac{\exp(x)}{\sum \exp(x_i)} \tag{1.6}$$

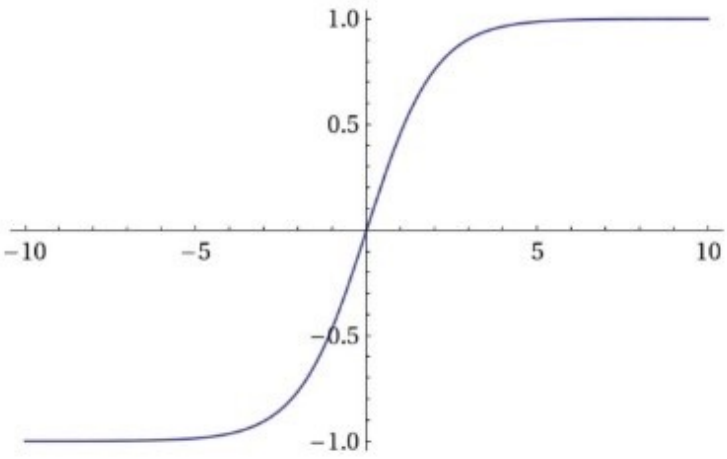

Figure 1.7: La fonction softmax
[5]

Avec la fonction d'activation l'équation de perceptron sera [4]:

$$\widehat{y} = \varphi(wx + b) \tag{1.7}$$

Où :

$$\varphi(.)$$

est une fonction d'activation

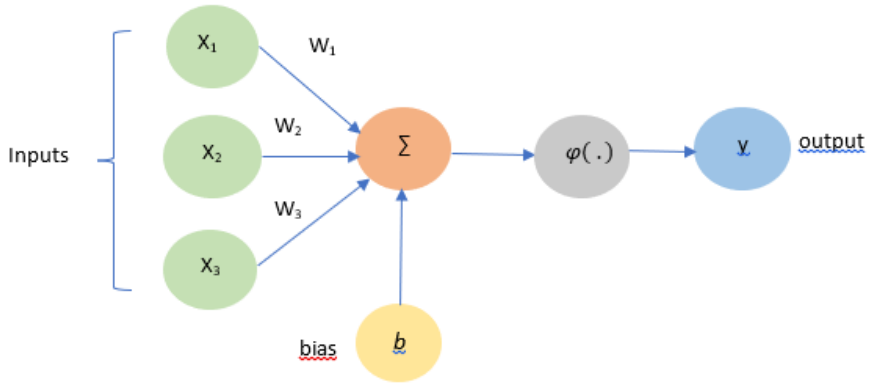

Figure 1.8: La structure d'un perceptron avec poids, biais et une fonction d'activation
[4]

1.3.2.3 Perceptron Multicouche MLP

Lorsque on connecte les perceptrons ou neurones artificiels ensemble de manière à ce que la sortie d'une couche devienne l'entrée de la couche suivante, jusqu'à ce que la sortie de la dernière couche devienne la sortie finale. Par cela, nous obtenons [4] des **Perceptrons Multicouches (MLP)** également appelés **réseaux de neurones profonds (DNN)** ou réseaux de neurones à **anticipation (FFNN)**.

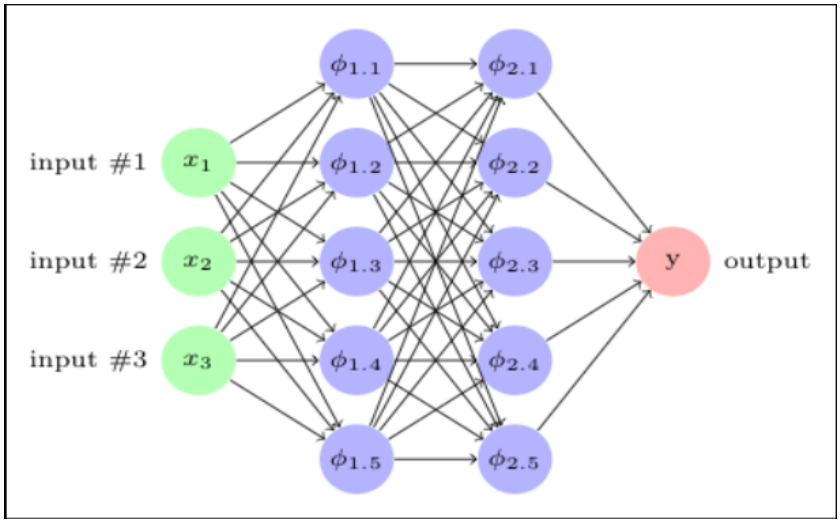

Figure 1.9: Un réseau neuronal entièrement connecté avec 3 entrées et deux couches cachées
[4]

1.3.3 Le prétraitement des données et entraînement du modèle

1.3.3.1 Le prétraitement des données

Le **prétraitement des données** est une étape cruciale qui permet d'améliorer la qualité des données afin de pouvoir l'extraction d'informations significatives à partir des données. Le prétraitement des données fait référence à la technique de préparation des données brutes pour les rendre adaptées à la construction et à la formation d'un modèle d'apprentissage en profondeur réussi. Le prétraitement des données se déroule en quatre étapes [6] , **nettoyage**, **réduction**, **intégration** et **transformation des données**.

Avant de passer les données au modèle pour commencer la phase d'entrainement, on les divise en trois groupes, des données **d'entraînement** (ce sont les données à partir desquelles le modèle apprend), de **validation** (les données qu'on va utiliser pour mesurer les hyperparamètres idéal du modèle) et de **test** (ces données sont utilisées lorsque le modèle termine son apprentissage pour mesurer leur performance). Ces trois ensembles de données distincts, tirés en tant qu'échantillons aléatoires de l'ensemble de données total [7].

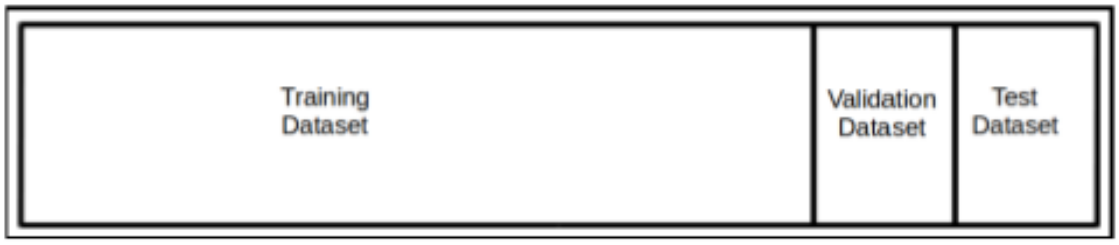

Figure 1.10: Les ensembles du jeu de données
[7]

1.3.3.2 Entrainement du modèle

L'entrainement d'un modèle [7] commence par une propagation vers l'avant où chaque neurone transmet leur valeur de sortie ($\widehat{y}$) au neurone suivant jusqu'à ce que nous atteignions la dernière couche. Après avoir obtenu la valeur ($\widehat{y}$) de chaque donnée, la fonction de perte est calculée :

$$L(\widehat{y}, y) = L(f(w, \widehat{y}), y) \tag{1.8}$$

Cette fonction mesure à quel point les valeurs attendues (y) et prédites ($\widehat{y}$) sont proches , étant donné les valeurs spécifiques des poids (des paramètres) w.
Pour que l'apprentissage se produise au sein du réseau, le signal d'erreur du réseau doit être **propagé vers l'arrière (rétropropagation)** [8] à travers les couches du réseau de la dernière couche à la première. L'objectif dans la rétropropagation est de propager ce signal d'erreur vers l'arrière pour mettre à jour les poids du réseau au fur et à mesure que le signal se propage. Mathématiquement, pour ce faire, nous devons minimiser la fonction de perte en poussant les poids vers des valeurs qui rendent la fonction de perte plus petite. Ce processus est appelé l'optimisation de réseau par la descente de gradient.
La descente de gradient [9]calcule la dérivée (gradient) de $L(\widehat{y}, y)$ par rapport à tous les poids du réseau. Le gradient indique le changement de $L(\widehat{y}, y)$ par rapport à chaque poids. Ensuite, l'algorithme utilise ces informations pour mettre à jour les poids de manière à minimiser la fonction de perte dans les itérations prochaine des mêmes paires valeur attendue/prédite. Le but est d'atteindre progressivement le minimum global de la fonction de perte.

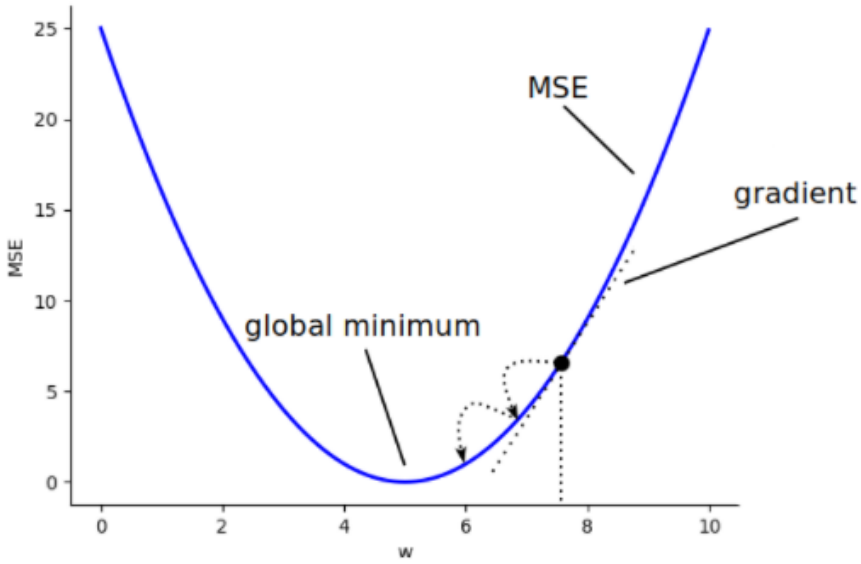

Figure 1.11: Visualisation de la descente de gradient de la fonction de perte MSE [9]

1.4 Les Types de Réseaux Neuronaux

1.4.1 Réseau neuronal récurrent RNN

Les réseaux de neurones feedforward FFNN ont fait leurs preuves dans de nombreuses applications, mais les chercheurs n'ont pas pu l'appliquer dans le domaine des données séquentielles car FFNNs ne prennent en compte que l'entrée actuelle et ne peuvent pas se souvenir de l'entrée précédente. Ici, les réseaux neuronaux récurrents RNNs sont apparus pour résoudre ces problèmes.

Les réseaux de neurones récurrents RNN [10] sont essentiellement des réseaux de neurones qui utilisent la récurrence, ils ont le concept de **mémoire** qui aide à stocker les informations provenant d'un précédent passage vers l'avant sur le réseau neuronal. Les RNNs sont adaptés et ont eu un succès incroyable lorsqu'ils sont appliqués à des problèmes dans lesquels les données d'entrée sur lesquelles les prédictions doivent être faites se présentent sous la forme d'une séquence (série

d'entités où l'ordre est important) par exemple dans le domaine de traitement des langues ou des sons.

1.4.1.1 La structure d'un réseau neuronal récurrent

On peut définir un RNN comme leur nom indique par une relation de récurrence [10]:

$$s_t = f(s_{t-1}, x_t) \tag{1.9}$$

Tel que :

- $s(t)$: c'est un vecteur de valeurs appelé l'état de la mémoire interne du réseau (au temps t).
- f: c'est une fonction différentiable.
- x_t : c'est l'entrée du réseau au temps t.

L'état de la mémoire interne au temps t est évalué en utilisant le même état de la mémoire au temps$t1$, celui au temps $t1$ avec la valeur au temps $t2$ et ainsi de suite, donc on s(t-1)considère comme le résumé du réseau de toutes les entrées précédentes. Le schéma suivant représente comment l'état de réseau évolue pas à pas au cours de la séquence via une boucle de rétroaction sur les états précédents :

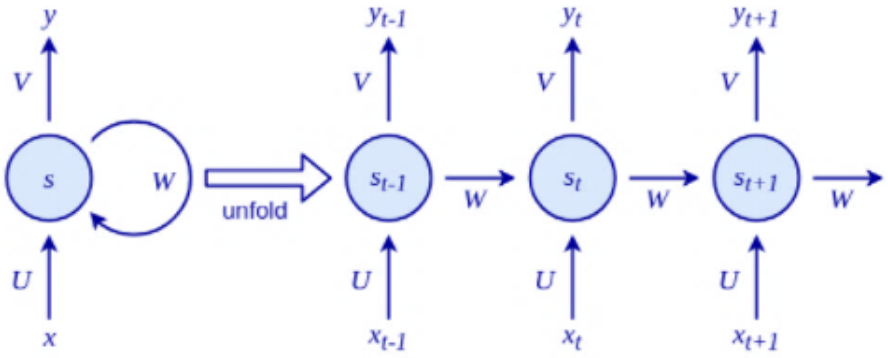

Figure 1.12: Illustration visuelle de la relation de récurrence dans RNN [9]

Les RNNs ont trois paramètres :

- U : transforme l'entrée x_t en l'état s_t.
- W : transforme l'état précédent s_{t-1} en l'état actuel s_t.
- V : mappe l'état interne récemment calculé $s(t)$ à la sortie, y_t.

U, V et W appliquent une transformation linéaire sur leurs entrées respectives comme le cas avec un perceptron simple. Maintenant on définit l'état interne et la sortie réseau comme suit [9]:

$$\begin{aligned} s_t &= f(Ws_{t-1}, Ux_t) \\ y_t &= Vs_t \end{aligned} \tag{1.10}$$

Où :

- f: Est une fonction d'activation non linéaire.
- y_t: Représente la sortie de RNN.

1.4.1.2 Les types de réseau neuronal récurrent

Il existe 4 types de réseau neuronal récurrent basant sur le nombre des entrées et des sorties et chacun a sa propre application [11]:

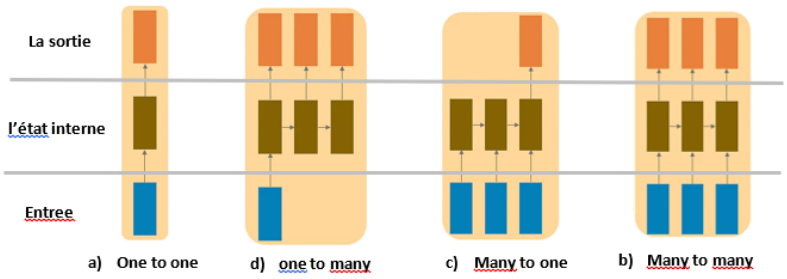

Figure 13: les types de RNN [11]

Figure 1.13: Les types de RNN
[11]

1.4.2 Réseau neuronal convolutif

Les réseaux de neurones convolutifs CNN désignent une sous-catégorie de réseaux de neurones, c'est une des techniques de classification et la reconnaissance d'images dans les réseaux de neurones. Il est conçu pour traiter les données via des tableaux multicouches.

Ce type de réseau de neurones est utilisé dans des applications comme la reconnaissance d'images ou la reconnaissance faciale. La principale différence entre CNN et les autres réseaux de neurones est que CNN accepte les entrées sous forme de tableau à deux dimensions. Il opère directement sur l'image au lieu de se concentrer sur l'extraction de caractéristiques ce que font les autres réseaux de neurones. Un réseau de neurones convolutifs (CNN ou ConvNet) est un réseau d'anticipation artificiel dont les schémas de connexion entre neurones s'inspirent de l'organisation du cortex visuel animal. CNN prend une image en entrée, qui est classés et traités dans une certaine catégorie telle que chien, chat, lion, tigre, etc. L'ordinateur voit une image comme un tableau de pixels et dépend de la résolution de l'image. Les réseaux de neurones convolutifs ont les 4 couches suivantes: [12]

1.4.2.1 La couche d'entrée

Contient les données des images qui sont représenter par une matrice tridimensionnelle qu'on doit la rassemble en une seule colonne

1.4.2.2 La couche de convolution

La couche de convolution est la composante clé des réseaux de neurones convolutifs, et constitue toujours au moins leur première couche.
Son but est de repérer la présence d'un ensemble de features dans les images reçues en entrée. Pour cela, on réalise un filtrage par convolution : le principe est de faire "glisser" une fenêtre représentant la feature sur l'image, et de calculer le produit de convolution entre la feature et chaque portion de l'image balayée. Une feature est alors vue comme un filtre.
La couche de convolution reçoit donc en entrée plusieurs images, et calcule la convolution de chacune d'entre elles avec chaque filtre. Les filtres correspondent exactement aux features que l'on souhaite retrouver dans les images.
On obtient pour chaque paire (image, filtre) une carte d'activation, ou feature map, qui nous indique où se situent les features dans l'image : plus la valeur est élevée, plus l'endroit correspondant dans l'image ressemble à la feature.

1.4.2.3 Couche de pooling

Ce type de couche est souvent placé entre deux couches de convolution : elle reçoit en entrée plusieurs feature maps, et applique à chacune d'entre elles l'opération de pooling qui consiste à réduire la taille des images, tout en préservant leurs caractéristiques importantes. Pour cela, on découpe l'image en cellules régulières, puis on garde au sein de chaque cellule la valeur maximale. En pratique, on utilise souvent des cellules carrées de petite taille pour ne pas perdre trop d'informations. Les

choix les plus communs sont des cellules adjacentes de taille 2×2 pixels qui ne se chevauchent pas, ou des cellules de taille 3×3 pixels, distantes les unes des autres d'un pas de 2 pixels (qui se chevauchent donc). On obtient en sortie le même nombre de feature maps qu'en entrée, mais celles-ci sont bien plus petites.

Les valeurs maximales sont repérées de manière moins exacte dans les feature maps obtenues après pooling que dans celles reçues en entrée. En effet, lorsqu'on veut reconnaître un chien par exemple, ses oreilles n'ont pas besoin d'être localisées le plus précisément possible : savoir qu'elles se situent à peu près à côté de la tête suffit !

Ainsi, la couche de pooling rend le réseau moins sensible à la position des features, le fait qu'une feature se situe un peu plus en haut ou en bas, ou même qu'elle ait une orientation légèrement différente ne devrait pas provoquer un changement radical dans la classification de l'image.

1.4.2.4 Couche de correction ReLu

Désigne la fonction réelle non-linéaire définir par $ReLU(x) = max(0, x)$, la couche de correction ReLU remplace donc toutes les valeurs négatives reçues en entrées par des zéros. Elle joue le rôle de fonction d'activation.

1.4.2.5 La couche fully-connected

La couche fully-connected constitue toujours la dernière couche d'un réseau de neurones, convolutif ou non – elle n'est donc pas caractéristique d'un CNN.

Ce type de couche reçoit un vecteur en entrée et produit un nouveau vecteur en sortie. Pour cela, elle applique une combinaison linéaire puis éventuellement une fonction d'activation aux valeurs reçues en entrée, elle permet de classifier l'image en entrée du réseau : elle renvoie un vecteur de taille N, où N est le nombre de classes dans notre problème de classification d'images. Chaque élément du vecteur

indique la probabilité pour l'image en entrée d'appartenir à une classe.

Chaque valeur du tableau en entrée "vote" en faveur d'une classe. Les votes n'ont pas tous la même importance : la couche leur accorde des poids qui dépendent de l'élément du tableau et de la classe. Pour calculer les probabilités, la couche fully-connected multiplie donc chaque élément en entrée par un poids, fait la somme, puis applique une fonction d'activation (logistique si N=2, softmax si N¿2) :

Ce traitement revient à multiplier le vecteur en entrée par la matrice contenant les poids. Le fait que chaque valeur en entrée soit connectée avec toutes les valeurs en sortie explique le terme fully-connected. [12]

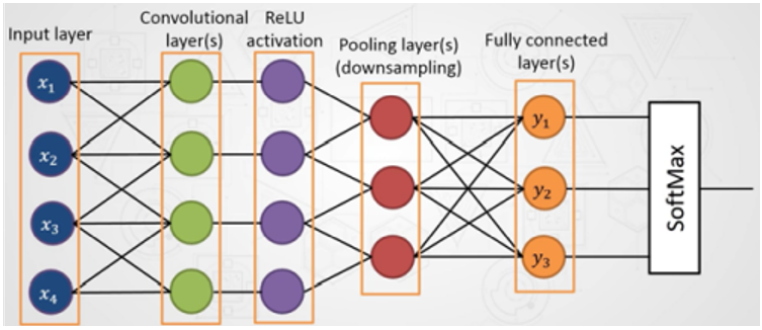

Figure 1.14: Architecture du CNN
[12]

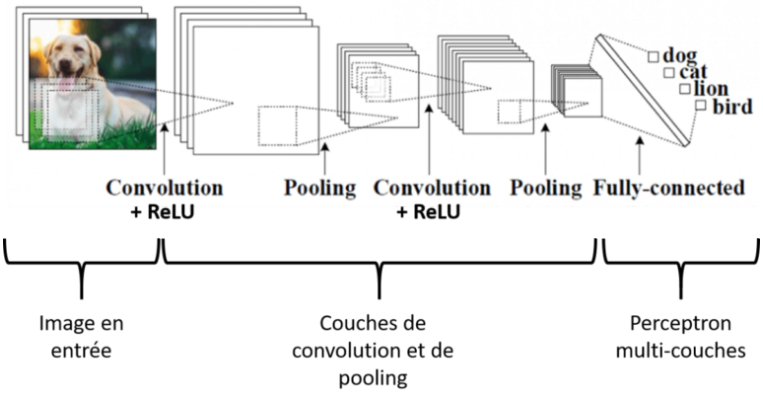

Figure 1.15: Les couches de CNN
[12]

1.4.3 Auto-Encodeur

Un auto-encodeur [13] est un réseau de neurones pour l'apprentissage non supervisé qui apprend à compresser et encoder efficacement les données x, puis apprend à reconstruire les données de la représentation codée réduite à une représentation x aussi proche que possible de l'entrée d'origine. Il convient de noter que l'idée est née dans les années 1980 et promue plus tard dans un article de Hinton et Salakhutdinov.

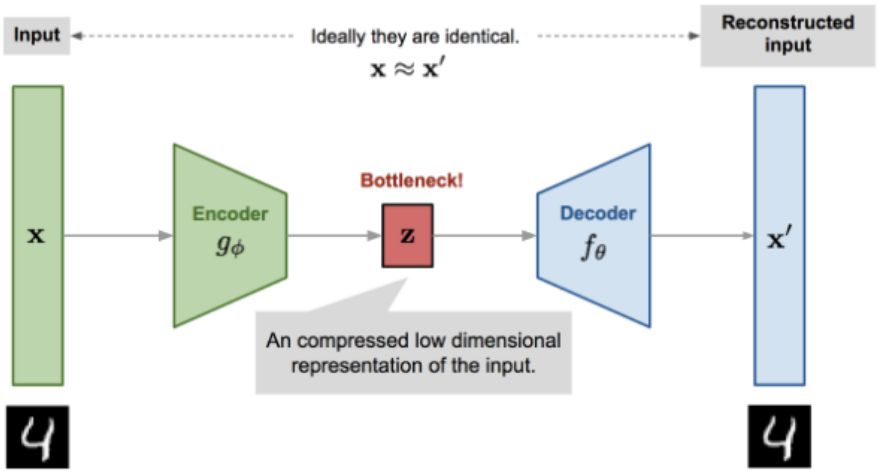

Figure 1.16: Une architecture d'un auto-encodeur [13]

L'auto-encodeur est constitué essentiellement de trois composants, **un encodeur** g_φ (il prend un vecteur de données d'entrée pour réduire leur dimension et le compresser dans une représentation codée), **espace latent** z (**bottleneck en anglais**, il contient les représentations codées réduites) et **le décodeur** f_θ (qui a la même architecture d'encodeur essaie de reconstruire cette entrée à partir de la représentation compressée).Leurs architecture varie entre un simple réseau FeedForward, un réseau RNN ou un réseau neuronal convolutif selon le cas d'utilisation.

1.4.4 Réseau antagoniste génératif

Le lundi 5 décembre 2016 [14] [15], Ian Goodfellow de Google Brain a présenté un tutoriel intitulé **"Generative Adversarial Networks GAN"** aux délégués de la conférence Neural Information Processing Systems (NIPS) à Barcelone. Les idées présentées dans le tutoriel sont considérées comme une des tournants clés de la modélisation générative et ont engendré une grande variété de variations sur son idée de base. Les GANs sont des auto-encodeurs qui ont été divisés au milieu et renversé. Goodfellow a poussé ce concept un peu plus loin en introduisant les concepts de **générateur** (fausseur d'art) et **discriminateur** (critique d'art).

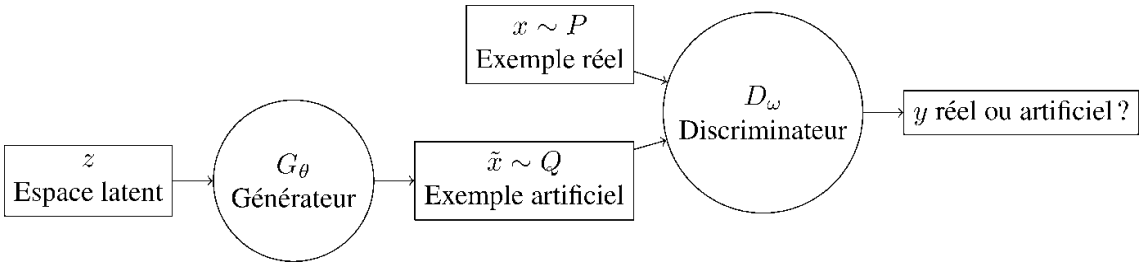

Figure 1.17: L'architecture du GAN
[14]

- **Le générateur** $G(.)$ apprend à créer de fausses données pour tromper le discriminateur. Il est formé pour minimiser l'erreur de classification finale (entre les données vraies et générées).

$$L^{(G)} = min[log(D(x)) + log(1 - D(G(z))] \tag{1.11}$$

- **Le discriminateur** $D(.)$ est simplement un classifieur. Il essaie de détecter les fausses données générées, de sorte que le réseau neuronal discriminatif est formé pour maximiser l'erreur de classification finale.

$$L^{(D)} = max[log(D(x)) + log(1 - D(G(z))] \tag{1.12}$$

En combinant les deux fonctions de perte d'entropie croisée, on obtient :

$$L = \max_{D} \min_{G}[log(D(x)) + log(1 - D(G(z))] \tag{1.13}$$

La fonction de perte précédente ne définit que la quantité de perte sur un seul pixel. Pour couvrir toute l'image ou un ensemble de données, nous devons appliquer la fonction objectif minimax suivante :

$$\max_{D} \min_{G} V(D, G) = \max_{D} \min_{G}(E_{xPdata(x)}[log(D(x)] + E_{zP(z)}[log(1 - D(G(z))]] \tag{1.14}$$

Où :

- $D(x)$: est la probabilité que les données x proviennent de l'ensemble des données d'entrainement.
- $G(z)$: représente les données générées.
- $E_{xPdata(x)}$: est l'espérance ou la distribution de données réelles.
- $E_{zP(z)}$: est la distribution attendue de fausses données.

Le générateur et le discriminateur forment deux réseaux contradictoires, ils essaient de se battre et, ce faisant, ils deviennent tous les deux de mieux en mieux. La compétition entre eux fait progresser ces deux réseaux par rapport à leurs objectifs. On considère ce cadre en théorie des jeux comme un jeu minimax à deux joueurs où l'état d'équilibre correspond à la situation où le générateur produit des données à partir de la distribution ciblée exacte et où le discriminateur prédit **"vrai"** ou **"généré"**.

1.4.5 Les Transformateurs

En décembre 2017, Google Brain et Google Research ont publié l'article fondateur de Vaswani et al. [16], **Attention is All You Need** . **Le transformateur** est né. Le transformateur a surpassé les modèles traitement du langage naturel PNL existante. Le transformateur s'est entraîné plus rapidement que les architectures précédentes et a obtenu des résultats d'évaluation plus élevés. En conséquence, les transformateurs sont devenus un élément clé de la NLP, plus récemment en vision par ordinateur.

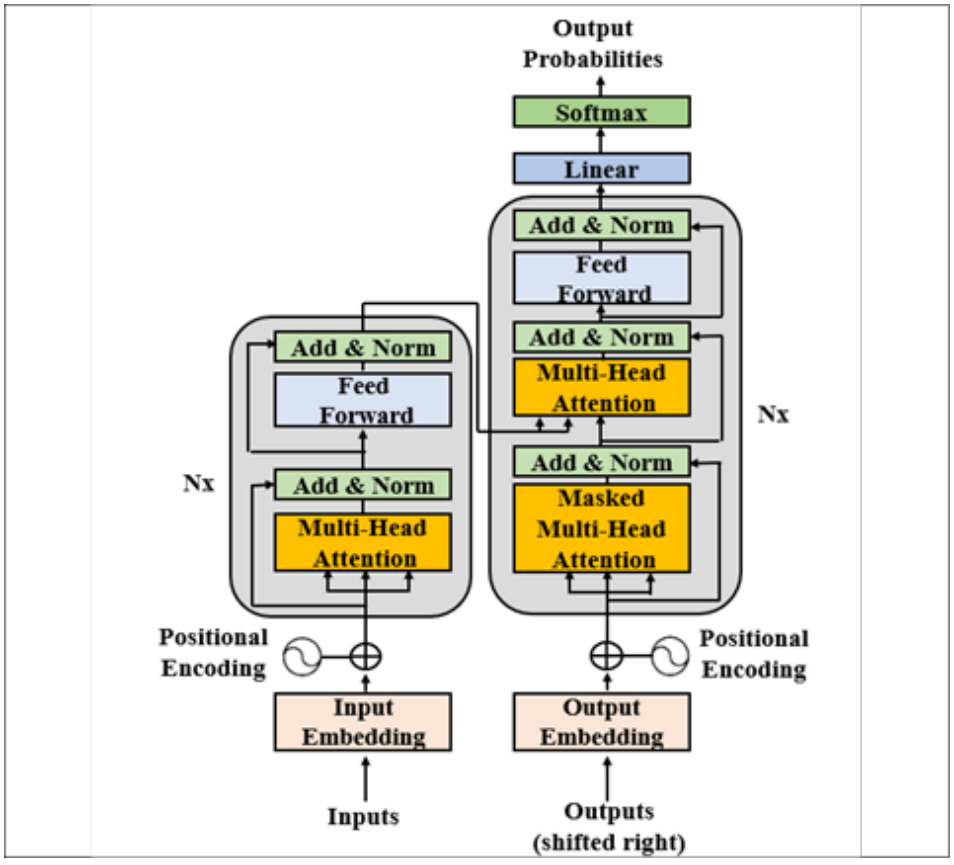

Figure 1.18: L'architecture du transformateur [16]

Les transformateurs basent essentiellement sur le concept d'attention. Pour in-

troduire rapidement ce terme, prenons un exemple simple de PNL en envoyant une phrase pour la traduire en un réseau de transformateurs. Dans ce cas, l'attention mesure la façon dont chaque mot de la phrase d'entrée est associé à chaque mot de la phrase traduite de sortie. De même, il y a aussi ce que nous appelons l'auto-attention qui pourrait être considérée comme une mesure de l'effet d'un mot spécifique sur tous les autres mots de la même phrase. Ce même processus peut être appliqué aux images en calculant l'attention des patchs des images et leurs relations les unes avec les autres.

Le transformateur est composé de **deux piles d'encodeur et décodeur**, **Embeddings** et **Softmax, Encodage positionnel**.

1.5 Mesures de performance et évaluation du modèle

L'évaluation des performances d'un modèle est à la fois une tâche critique et complexe. Il faut donc veiller à fiabiliser et à évaluer les résultats rapportés des modèles, ce qui rendent un modèle meilleur que l'autre. Plusieurs métriques ont été proposées pour évaluer les performances prédictives des problèmes de régression et de classification.

1.5.1 Évaluer la performance des algorithmes de classification

Évaluer les performances d'un algorithme de classification est souvent plus délicat que d'évaluer celles d'un algorithme de régression. Plusieurs méthodes permettent cependant de mesurer les performances de l'algorithme. La première concerne la matrice de confusion. L'idée derrière cette matrice est de faciliter la visualisation des performances et de mesurer le pourcentage des individus mal classés. Plus ce

pourcentage sera élevé, plus on peut remettre en question l'efficacité et les performances de l'algorithme. Sur les lignes de cette matrice se trouveront les classes prédites, tandis que chaque colonne représente une classe réelle. [17]

		Classe réelle	
		-	+
Classe prédite	-	True Negatives *(vrais négatifs)*	False Negatives *(faux négatifs)*
	+	False Positives *(faux positifs)*	True Positives *(vrais positifs)*

Figure 1.19: La matrice de confusion
[17]

1.5.1.1 Le taux de succès

Sur la diagonale principale de cette matrice se trouvent les cas correctement classés (TN et TP), toutes les autres cellules de la matrice contiennent des exemples mal classés. A partir de cette matrice, il nous est possible de calculer différentes mesures. La plus connue est l'Accuracy et mesure le nombre d'individus correctement classés sur le nombre total d'individus. Il s'agit donc de la somme des éléments de la diagonale de la matrice que l'on divise par le total de la matrice.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

1.5.1.2 Précision

Cette matrice donne beaucoup d'informations, mais parfois nous préférons une mesure plus brève. On introduit donc ici la Précision d'un classificateur qui se mesure comme suit :

$$Précision = \frac{TP}{TP + FP}$$

La Précision sera donc égale à 1, ce qui est un score parfait, si le classificateur prédit que des True Positives. Cette Précision est souvent utilisée en combinaison avec une autre mesure qui est Recall

1.5.1.3 Recall

Recall mesure le ratio des cas positifs qui ont été correctement détectés par l'algorithme . Dans certains cas, il peut être préférable de parvenir à détecter les cas indésirables que de détecter les cas souhaités.

$$Recall = \frac{TP}{TP + FN}$$

1.5.1.4 F1 score

Précision et Recall sont ainsi combinés dans une même mesure appelée F1 score, qui est la moyenne harmonique des deux outils de mesures. La moyenne harmonique, à l'inverse de la moyenne arithmétique, va accorder un poids plus élevé aux faibles valeurs. De ce fait, le F1 score d'un classificateur sera élevé si les mesures Précision et Recall sont toutes les deux élevées. Malheureusement, augmenter la Précision diminue le Recall, et vice-versa ce qui rend compliqué d'obtenir un F1 score élevé. Ce phénomène s'appelle le compromis Précision/Recall. [18]

$$F1Score = \frac{2(Recall * Precision)}{Recall + Precision}$$

1.5.2 Évaluer la performance des algorithmes de régression

Pour évaluer les modèles de régression et de les comparer, on peut calculer la distance entre valeurs prédites et vraies valeurs. Cela nous donne plusieurs critères :

1.5.2.1 L'erreur quadratique moyenne

L'erreur quadratique moyenne RMSE (Root mean squared error) est une formule populaire pour mesurer le taux d'erreur d'un modèle de régression. Cependant, il ne peut être comparé qu'entre des modèles dont les erreurs sont mesurées dans les mêmes unités.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(pi - ai)^2}{n}}$$

Où : p est la cible prévue et a est la cible réelle

1.5.2.2 L'erreur absolue moyenne

L'erreur absolue moyenne MAE (Mean Absolut error) a la même unité que les données d'origine et ne peut être comparé qu'entre des modèles dont les erreurs sont mesurées dans les mêmes unités. Son ampleur est généralement similaire à celle du RMSE, mais légèrement plus petite. t p sont défini dans l'erreur quadratique moyenne.

$$MAE = \frac{\sum_{i=1}^{n}|(pi - ai)|}{n}$$

1.5.2.3 Le coefficient de détermination R 2

Le coefficient de détermination R 2 (R squared error), résume le pouvoir explicatif du modèle de régression. R2 est calculé à partir des termes des sommes des carrés : SSE = erreur sur la somme des carrés SST = somme des carrés total R 2 décrit la proportion de variance de la variable dépendante expliquée par le modèle de régression. Si le modèle de régression est parfait , SSE vaut zéro et R2 vaut 1, Si

c'est un échec total, SSE vaut SST, aucune variance n'est expliquée par la régression et R2 vaut zéro [19].

$$R2 = 1 - \frac{SSE}{SST}$$

1.6 Conclusion

En résumé, notre premier chapitre commence par un exposé de l'intelligence artificielle, qui repose sur deux parties essentielles : l'apprentissage automatique avec trois types et les réseaux de neurones artificiels génératifs contradictoires. A partir de là, on peut saisir et comprendre : CNN est dédié au traitement (images, formes...), tandis que RNN est dédié au traitement de texte. En fin de compte, notre travail dépend des performances du système d'apprentissage automatique et de la spécification d'un bon moyen de tester ses performances. Pour le chapitre suivant, nous commencerons par présenter les travaux connexes liés à notre méthode.

Chapitre 2

Apprentissage Des Représentations Des Formes 3D

2.1 Inroduction

Avec la construction d39;ensembles de données de modèles 3D à grande échelle, de nombreux descripteurs profonds de formes 3D sont proposés. Basées sur différents types de données. Dans ce chapitre nous allons voir en premier l'extraction des features à partir de différentes représentations de formes 3D qui sont : la représentation volumétrique basée sur les voxels, la représentation multi-view, la représentation Mesh (maillage), la représentation Point cloud et en fin la représentation RGBD En deuxième partie on parle sur les architectures de l'apprentissage profond (comme CNN et Multi view CNN) et de la génération des forme 3D pour l'extraction des descripteurs des forme 3D.

2.2 Extraction des descripteurs 3D (Features extraction)

2.2.1 La représentation Volumétrique

Les données 3D offrent aux systèmes de vision par ordinateur une vision riche du monde, mais posent également un ensemble unique de défis, en particulier dans les applications où la compréhension de l'environnement est essentielle. En particulier, il n'est pas garanti que des données telles qu'un point cloud extrait d'une image RVB-D ou un maillage polygonal soient disposées dans une grille régulière, ce qui les rend inadaptées à une utilisation avec des algorithmes d'apprentissage automatique hautes performances tels que les réseaux de neurones convolutifs.

Les voxels stockent des informations physiques (couleur, densité, intensité, etc.) pour des points de volume sur une grille régulière. Ses coordonnées spatiales, voire temporelles, ainsi que sa taille ou d'autres informations (comme le matériau) sont parfois stockées avec sa valeur, parfois en parallèle. Alors que les espaces vectoriels sont bons pour cela, les voxels s'intègrent plus généralement dans les espaces matriciels. Ses coordonnées spatiales peuvent être des coordonnées polaires. Le format par voxel présente un certain nombre de difficultés. L'ajout d'une troisième dimension spatiale dans la grille régulière s'accompagne d'un coût de calcul correspondant, et la malédiction de la dimensionnalité est un problème central, limitant la résolution disponible de la grille de voxels. Les grilles à faible résolution rendent difficile la différenciation entre des formes similaires et rejettent certaines des informations de texture disponibles dans les rendus 2D de dimension équivalente. [20]

2.2.2 La représentation Multi View

Des méthodes récentes basées sur l'apprentissage ont tenté d'aborder la tâche d'inférence 3D à vue unique, mais cette capacité a un coût. Ces approches reposent sur une supervision 3D complète et nécessitent une forme 3D connue pour chaque image d'entraînement. Non seulement cette forme de supervision est écologiquement invraisemblable, mais elle est également pratiquement fastidieuse à acquérir et difficile à mettre à l'échelle. Au lieu de cela, en utilise la supervision multi-vues plus naturellement plausible. Le perceptron multi-view est un réseau qui prend une image de visage et un vecteur aléatoire en entrée et génère une vue aléatoire de ce visage avec le point de vue correspondant.

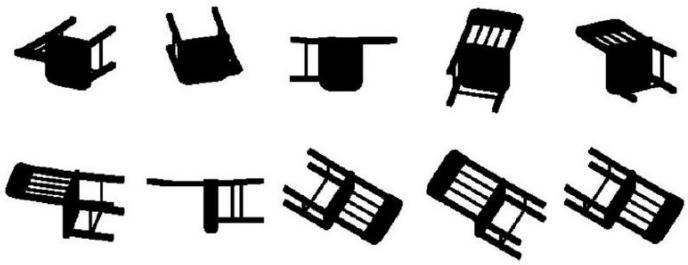

Figure 2.1: Le 3D modèle multi-view
[21]

Les humains sont des organismes en mouvement : notre surveillance écologique consiste à observer le monde et les objets qu'il contient sous différents angles, et ces multiples points de vue nous informent de la géométrie sous-jacente. Cette idée a été exploitée avec succès par une longue gamme de techniques de reconstruction basées sur la géométrie. Cependant, ces méthodes de structure à partir de mouvement ou de stéréo à vues multiples fonctionnent pour des instances spécifiques et ne se généralisent pas, contrairement aux humains, pour prédire la forme 3D d'une nouvelle instance à partir d'une seule vue.
Un réseau d'encodeur simple et élégant dépeint un modèle 3D d'un objet à par-

tir d'une seule image de cet objet, voir la figure 1, l'objet est présenté par ce que nous appelons modèle 3D multi-view l'ensemble de toutes ses vues et ses cartes de profondeur correspondantes. Compte tenu d'un point de vue arbitraire, le réseau génère une image RVB de l'objet et de la carte de profondeur. Cette représentation contient des informations riches sur la géométrie 3D de l'objet, mais permet une implémentation plus efficace que les modèles 3D basés sur Voxel.

En fusionnant plusieurs vues de notre représentation multi-view, nous obtenons un point cloud complet 3D de l'objet, y compris des pièces invisibles dans l'image d'entrée d'origine Alors que techniquement, la tâche s'accompagne de nombreuses ambiguïtés, les humains sont connus pour être bons dans l'utilisation de leurs connaissances antérieures sur des objets similaires pour deviner les informations manquantes. [21]

2.2.3 Mesh (maillage)

Le maillage est un type de données important et puissant pour les formes 3D et largement étudié dans le domaine de la vision par ordinateur et de l'infographie. Cependant, peu d'efforts ont été déployés pour utiliser les données de maillage ces dernières années, en raison de la complexité et de l'irrégularité des données de maillage.

Les données de maillage de formes 3D sont une collection de sommets, d'arêtes et de faces, dans lesquelles les sommets sont connectés avec des arêtes et des ensembles fermés d'arêtes forment des faces. Les données de maillage sont principalement utilisées pour stocker et rendre des modèles 3D en infographie, car elles fournissent une approximation des surfaces lisses des objets et simplifient le processus de rendu. De nombreuses études sur les formes 3D dans le domaine de l'infographie et de la modélisation géométrique sont prises à base de maillage.

Les données de maillage montrent une plus grande capacité à décrire des formes 3D. La grille volumétrique et les vues multiples sont des types de données définis pour éviter l'irrégularité des données natives telles que le maillage et le point cloud, tout en perdant certaines informations naturelles de l'objet d'origine. Pour le point cloud, il peut y avoir une ambiguïté causée par un échantillonnage aléatoire et l'ambiguïté est plus évidente avec moins de points. En revanche, le maillage est plus clair et perd moins d'informations naturelles. En outre, lors de la capture de structures locales, la plupart des méthodes basées sur un nuage de points collectent les voisins les plus proches pour construire approximativement une matrice de contiguïté pour un processus ultérieur, tandis que dans le maillage, il existe une connexion explicite pour montrer clairement la structure locale. Cependant, les données de maillage sont également plus irrégulières et complexes pour les compositions multiples et le nombre variable d'éléments.

Le problème de complexité est que le maillage est constitué de plusieurs éléments, et différents types de connexions peuvent être définis entre eux. L'irrégularité est un autre défi pour le traitement des données de maillage, ce qui indique que le nombre d'éléments dans le maillage peut varier considérablement entre les formes 3D et que leurs permutations sont arbitraires. Malgré les problèmes complexité et d'irrégularité, le maillage a une plus grande capacité de description de forme 3D que d'autres types de données. Dans de telles circonstances, comment représenter efficacement des formes 3D à l'aide de données de maillage est une tâche urgente et difficile. Pour tirer pleinement parti des avantages du maillage et résoudre le problème de son irrégularité et de sa complexité, Feng et al [22] proposent deux idées clés de conception :

- **Considérez le visage comme l'unité (Regard face as the unit) :**

  Les données de maillage sont constituées de plusieurs éléments et des connex-

ions peuvent être définies entre eux. Pour simplifier l'organisation des données, nous considérons la face comme la seule unité et définissons une connexion entre deux faces si elles partagent une arête commune. Il y a plusieurs avantages à cette simplification. Premièrement, une face triangulaire ne peut pas se connecter à plus de trois faces, ce qui rend la relation de connexion régulière et facile à utiliser. Plus important encore, nous pouvons résoudre le problème de désordre avec des processus par face et une fonction de symétrie, similaire à PointNet , avec des processus par face et une fonction de symétrie. Et intuitivement, la face contient également plus d'informations que le sommet et l'arête.

- **Fonction de fractionnement du visage (Split feature of face) :**
  Bien que la simplification ci-dessus nous permette de consommer des données de maillage similaires aux méthodes basées sur des points, il existe encore quelques différences entre l'unité de point et l'unité de face car la face contient plus d'informations que le point. Nous avons seulement besoin de savoir "où vous êtes" pour un point, alors que nous voulons également savoir "à quoi vous ressemblez" pour un visage. En conséquence, nous divisons la caractéristique des visages en caractéristique spatiale et caractéristique structurelle, ce qui nous aide à capturer les caractéristiques plus explicitement.

Les données de maillage sont transformées en une liste de faces. Pour chaque face, nous définissons les valeurs initiales de celle-ci, qui sont divisées en quatres parties :

1. Centre : coordonnée du point central
2. Coin : vecteurs du point central à trois sommets

3. Normal : le vecteur normal unitaire

4. Neighbor Index : index des faces connectées (rempli avec l'index de lui-même si le visage se connecte avec moins que trois visages)

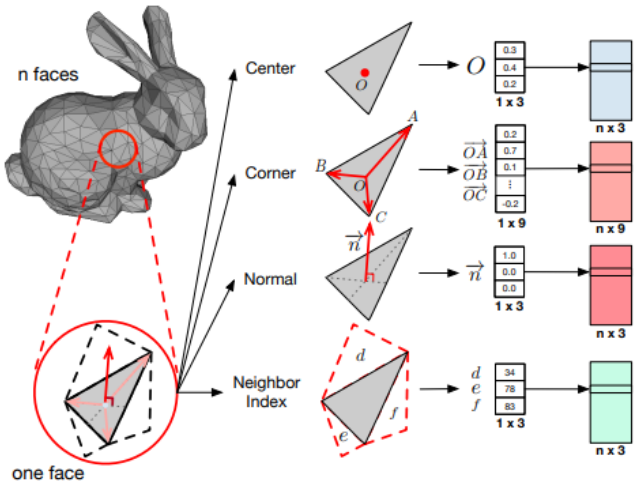

Figure 2.2: Valeurs initiales de chaque face

Descripteurs spatiaux et structurels: ils divisent les caractéristiques des visages en caractéristiques spatiales et en caractéristiques structurelles. La caractéristique spatiale est censée être pertinente pour la position spatiale des visages, et la caractéristique structurelle est pertinente pour les informations de forme et les structures locales.

- **Descripteur spatial:** La seule valeur d'entrée pertinente pour la position spatiale est la valeur centrale.

- **Descripteur structurel :** convolution de rotation de visage Nous proposons deux types de descripteurs structurels, et le premier est appelé convolution de rotation de visage, qui capture la structure interne des visages et se concentre sur les informations de forme des visages. L'entrée de ce bloc est la valeur du coin. [22]

2.2.4 Point cloud

La tâche de comprendre notre monde réel a atteint un grand bond ces dernières années. L'essor de puissantes ressources de calcul telles que les GPU et la disponibilité de données 3D provenant de capteurs de profondeur ont accéléré le domaine en pleine croissance de l'apprentissage en profondeur 3D. Ce dernier avec des données 3D a considérablement progressé depuis l'introduction des réseaux de neurones convolutifs capables de gérer l'ambiguïté de l'ordre des points dans les données de point cloud. Tout en étant capables d'obtenir de bonnes précisions dans diverses tâches de compréhension de scène, les méthodes précédentes ont souvent une faible vitesse d'apprentissage et une architecture de réseau complexe. Parmi les différentes représentations de données 3D, les point cloud sont largement utilisés en infographie et en vision par ordinateur grâce à leur simplicité. Des travaux récents ont montré de grandes promesses dans la résolution de problèmes classiques de compréhension de scène avec des nuages de points tels que des objets 3D classement et segmentation.

Cependant, les progrès actuels de la classification avec la 3D les nuages de points ont connu une tendance à la saturation des performances. Par exemple, de nombreuses méthodes récentes de classification d'objets ont rapporté des précisions très élevées en 2018, et la tendance à amener la précision vers la perfection est toujours en cours. Ce phénomène nous incite à nous demander si des problèmes tels que la classification d'objets 3D ont été totalement résolus, et de réfléchir à la manière d'aller de l'avant En analysant les résultats du benchmark, nous identifions trois questions ouvertes qui méritent d'être explorées plus avant pour de futures recherches. Premièrement, les modèles de classification entraînés sur des données synthétiques ne se généralisent souvent pas bien aux données du monde réel telles que les nuages de points reconstruits à partir de scans RVB-D, et vice versa. Deuxièmement, les observations

difficiles en contexte et partielles d'objets du monde réel sont courantes en raison d'occlusions et d'erreurs de reconstruction ; par exemple, ils peuvent être trouvés dans des détecteurs d'objets basés sur des fenêtres dans de nombreuses applications de robotique ou de véhicules autonomes. Enfin, comment gérer efficacement l'arrière-plan lorsqu'il apparaît avec des objets en raison de l'encombrement des scènes du monde réel. [23]

2.2.5 RGB-D

Avant le lancement de Microsoft Kinect en novembre 2010, la collecte d'images avec un canal de profondeur était une tâche lourde et coûteuse. Les chercheurs ont construit des configurations stéréo actives personnalisées et ont utilisé des scanners 3D coûtant des dizaines de milliers de dollars. Bon nombre de ces premiers ensembles de données ont capturé des images statiques d'objets isolés, car les capteurs utilisés ne se transportaient pas facilement comme dans cette figure

Passé: la plupart des ensembles de données de profondeur étaient petits et capturés en laboratoire.

Présent: ils bénéficent désormais de données RGBD provenant de scènes dynamiques et statiques du monde réel, avec une gamme de conditions d'étiquetage et de capture.

Avenir: ils peuvent anticiper les scans de scènes statiques et dynamiques sous forme de géométrie fusionnée, en exploitant les améliorations des algorithmes de reconstruction.

Les premiers ensembles de données Kinect se concentraient également sur des images statiques, souvent d'objets uniques ou de petites scènes. Au fur et à mesure que le

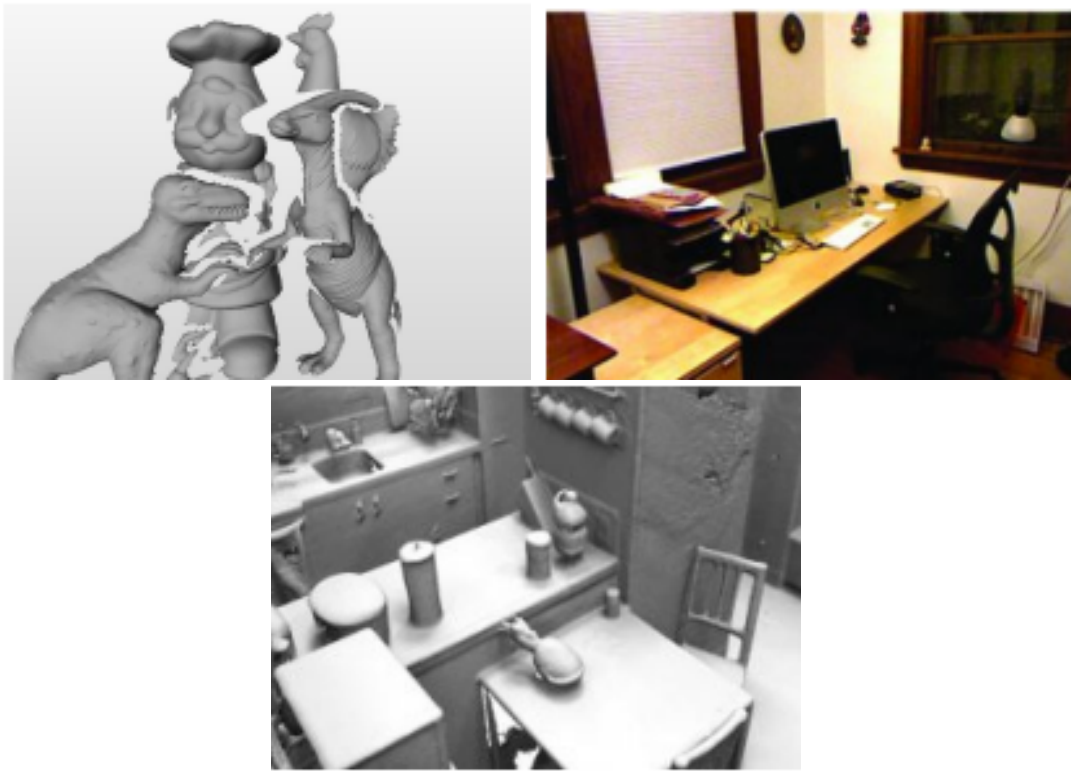

Figure 2.3: Le passé, le présent et l'avenir des ensembles de données RGB-D [24]

domaine mûrit, nous voyons des recherches mises en œuvre pour créer des ensembles de données RGBD plus vastes et plus ambitieux, et la quantité publiée chaque année ne montre aucun signe de diminution . Des étiquettes sémantiques ont été propagées à travers des vidéos, la reconstruction dense a été exploitée pour capturer les surfaces d'objets entiers et des algorithmes de scène générative ont été utilisés pour créer des données synthétiques plausibles. Nous voyons également de nouvelles étiquettes appliquées aux données existantes et les versions précédentes recompilées dans de nouvelles offres.

Malgré la disponibilité actuelle des capteurs, cependant, collecter des données RGBD n'est pas encore anodin. Les chercheurs utilisant le Kinect ont construit des dispositifs à batterie, écrit des pilotes et développé des formats de données personnalisés. Les ensembles de données RGBD accessibles au public peuvent, au

niveau le plus élémentaire, supprimer la nécessité de répéter la capture de données. Plus important encore, ils assurent la transparence dans la présentation des résultats et permettent de comparer les scores sur les mêmes données par différents chercheurs. Cela peut à son tour stimuler la concurrence pour des algorithmes plus performants. Enfin, un ensemble de données peut aider à orienter la recherche vers des directions. [24]

2.3 Descripteurs Profonds (Deep features)

2.3.1 3D CNN

VoxNet [25] a été le premier concept de convolution 3D et a été proposé par Maturana et Scherer pour l'apprentissage supervisé d'objets 3D sur différentes représentations de données 3D : données RVB-D, nuages de points LIDAR et modèles CAD 3D. La convolution dans VoxNet suivait la convolution 2D sauf pour le filtre où un filtre 3D a été utilisé à la place d'un filtre 2D. L'architecture du réseau est composée de (la couche d'entrée, deux couches Convolutives, une couche de pooling et deux couches FC). Les données d'entrée (ModelNet 10, ModelNet 40) ont été construites sous la forme d'une grille d'occupation volumétrique de 32x32x32 voxels et ont été transmises au réseau qui a été formé à l'aide de SGD avec optimiseur de moment (voir figure 2.4).

Brock et al [26] ont proposé le modèle Voxception-ResNet (VRN) très profond. Comme son nom l'indique, VRN s'appuyait sur les architectures Inception. Il est composé de 45 couches de profondeur, ce qui a nécessité une augmentation des données pour le processus de formation afin d'éviter le surajustement qui peut résulter de l'architecture profonde d'un petit ensemble de données. VRN est similaire à VoxNet en ce sens qu'ils adoptent tous deux ConvNet avec des filtres 3D,

mais VRN est très profond par rapport à VoxNet, qui a obtenu une amélioration significative de la tâche de classification (voir figure 2.5).

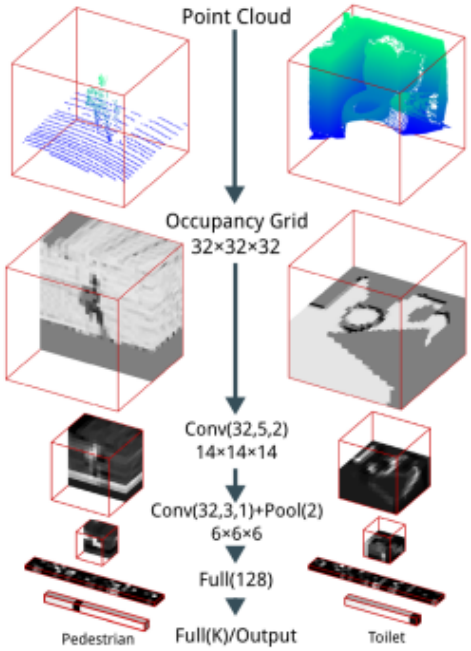

Figure 2.4: L'architecture du VoxNet
[25]

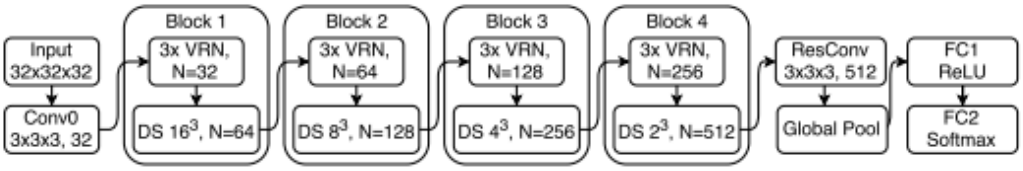

Figure 2.5: Architecture Voxception-ResNet 45 couches. Les DS sont des blocs Voxception-Downsample
[26]

2.3.2 Multi-View CNN

En raison de la puissance de calcul élevée nécessaire pour effectuer un CNN 3d, un nouveau "Multi-View CNN (MVCNN)" a été proposé par Su et al [27]. MVCNN a

traité plusieurs vues pour les objets 3D sans ordre spécifique à l'aide d'une couche de regroupement de vues. Deux configurations différentes pour capturer les objets 3D multi-vues ont été testées. Le premier a rendu 12 vues pour l'objet en plaçant 12 caméras virtuelles équidistantes entourant l'objet tandis que l'autre configuration comprenait 80 vues virtuelles. Le réseau proposé comporte deux parties, la première partie est l'endroit où les vues de l'objet sont traitées séparément et la deuxième partie est l'endroit où l'opération de mise en commun maximale a lieu sur toutes les vues traitées dans la couche de mise en commun des vues, ce qui donne une seule représentation compacte pour toute la forme 3D.

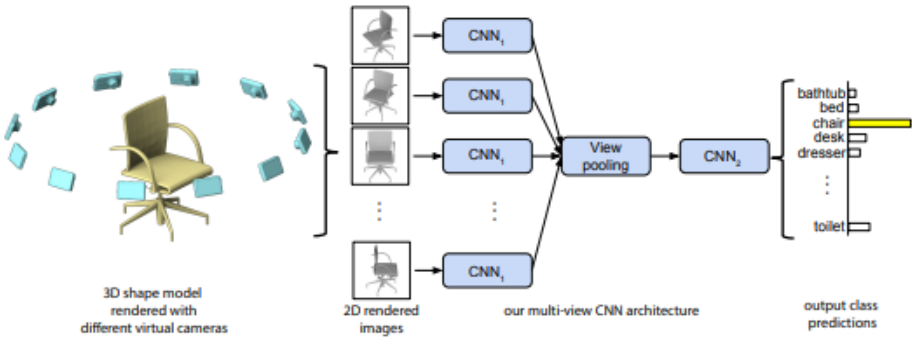

Figure 2.6: CNN multi-vues pour la reconnaissance de formes 3D [27]

2.4 La génération des formes 3D

De nombreuses recherches ont été publiées sur la génération d'images à partir de texte, mais très peu ont été faites sur la génération de formes 3D à partir de descriptions textuelles. keivin et al [28] présentent dans leur article Text2Shape : Generating Shapes from Natural Language by Learning Joint Embeddings le premier travail qui s'intéresse de génération de formes 3D colorées à partir d'une description en langage naturel. Ce travail s'est inspiré de la propriété communicative

de la cognition humaine pour permettre aux systèmes de l'imiter. Par exemple, si on décrive une forme de meuble avec une couleur et une texture par votre langage naturel à un ordinateur, il le concevra sans utilisation d'applications coûteuses en calcul (exigences matérielles et consommation de temps). Aussi, sans avoir besoin d'être un expert en design. Ce travail a été difficile pour les chercheurs en raison de:

- Il n'y a pas de correspondance directe entre le texte et les formes.
- Une forme a plus d'une description et vice versa.

Pour réaliser cette idée intéressante Kevin et al divisent le travail en deux tâches principales :

1. Récupération de texte en forme.
2. Génération de texte en forme.

Avant de plonger dans les détails des tâches, les auteurs ont utilisé nous le ShapeNet catégories d'objets table et chaise (avec 8 447 et 6 591 instances, respectivement). Ces formes 3D ont été créées par des concepteurs humains pour représenter avec précision la réalité objets. Ils choisissent les catégories table et chaise car elles contiennent de nombreuses instances avec des variations d'attributs fines dans la géométrie, la couleur et le matériau. Ils augmentent cet ensemble de données de formes avec 75 344 descriptions en langage naturel. Ils produisent ensuite une voxélisation des couleurs des maillages 3D à l'aide d'une méthode hybride d'échantillonnage basée sur la vue et sur la surface, suivie d'un sous-échantillonnage avec un filtre passe-bas dans l'espace voxel. La première tâche ils présentent d'abord une méthode pour apprendre un texte conjoint et un espace de représentation de forme directement à partir de descriptions en langage naturel d'instances de forme 3D, suivie de notre cadre de génération de texte à forme. Ils utilisent des encodeurs de texte et de forme pour calculer les plongements : les instances de chaque modalité

font des allers-retours vers des instances similaires de la même modalité, établissant des associations intermodales : les similitudes d'apprentissage des métriques sont calculées au sein et entre les modalités et les intégrations appartenant à la même classe étant rapprochées et les intégrations de différentes classes étant séparées. L'intégration conjointe regroupe des textes similaires et des formes similaires, maintienne les descriptions de texte proches de leur forme associée dans la position, et sépare le texte des formes qui ne sont pas similaires. Pour calculer les plongements, ils ont utilisé une structure CNN + RNN (GRU) pour l'encodeur de texte et un 3D-CNN pour l'encodeur de forme.

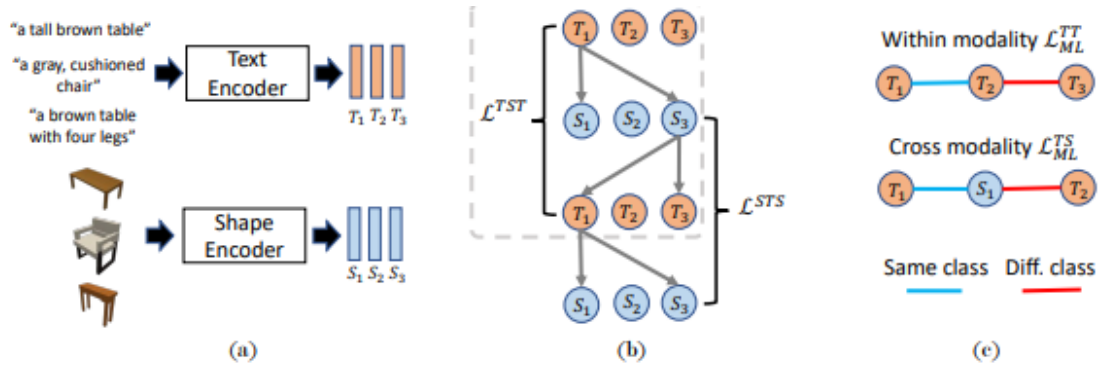

Figure 2.7: Approche d'apprentissage de la représentation conjointe [28]

La deuxième tâche, ils appliquent leur approche d'apprentissage de la représentation conjointe à la génération de texte en forme en utilisant des réseaux contradictoires génératifs. Le modèle est composé de trois composants principaux : l'encodeur de texte, le générateur et le critique. L'encodeur de texte mappe le texte en représentations latentes. Les vecteurs latents sont concaténés avec un vecteur de bruit, puis transmis au générateur, qui construit les formes de sortie. Enfin, le critique détermine à la fois le réalisme des sorties générées et leur degré de correspondance avec les descriptions textuelles correspondantes. Les GAN sont idéaux pour la génération de texte en forme car ils encouragent le modèle à générer des sor-

ties avec les propriétés correctes tout en évitant l'utilisation de contraintes par voxel. Ceci est crucial car la même description doit pouvoir générer différentes formes qui capturent toutes les attributs spécifiés. Les auteures utilisent une formulation base sur le GAN de Wasserstein qui améliore la diversité de sortie tout en évitant les problèmes d'effondrement de mode avec les GAN traditionnels.

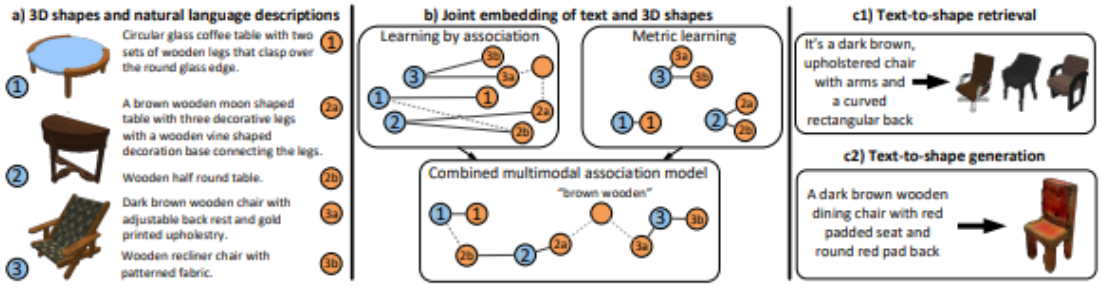

Figure 2.8: La génération des formes 3D
[28]

Article	Modèle profond	DataSet	Le type de représentations d'entrés
Maturana and Scherer	3D CNN	ModelNet	RGB-D,point clouds,3D CAD
Brock et al	3D CNN	ModelNet	RGB-D,point clouds, 3D CAD
Su et al	MVCNN	ImageNet1K	Les projections 2D du forme
Chen et al	CWGAN	ShapeNet	Voxles

Table 2.1: Architectures de l'apprentissage profond

2.5 Conclusion

Contrairement aux images 2D qui ont une représentation sous forme de matrices de pixels, les données 3D peuvent avoir différentes représentations où la structure et les propriétés géométriques varient d'une représentation à l'autre. Dans ce chapitre on a présenté dans la première partie les quatre majeures représentations des formes 3D. En deuxième partie on a abordé les précédents travaux des différents modèles des réseaux de neurones qui représentent et manipule les données 3D. Dans le prochain chapitre, on va passer à la partie conceptuelle, qui sera consacrée pour exprimer les architectures utilisées pour la réalisation de notre projet .

Chapitre 3

L‘architecture et La Conception

3.1 Introduction

Nous avons abordé dans le chapitre précédent un certain nombre de représentation 3D. Beaucoup de représentation restent utilisées, toutefois, dans un contexte de représentation automatique des objet 3D, il est primordial de trouver un moyen efficace qui permettrai de créer des représentations de forme 3D.Pour ce faire, il est nécessaire que les modèles utilisés assimilent de façon sémantique les représentation 3D.

Dans ce chapitre représente la partie conceptuelle de notre travail. Dans un premier lieu nous allons situer notre travail dans le cadre du projet. Nous par la suite les approches utilisées. Dans notre cas, il sera question de deux méthodes. La première est à base d'autoencoder et la seconde est à base de Transformer.

3.2 L'architecture globale

Notre approche s'insère dans le cadre du Self-Supervised Learning (Auto-apprentissage) où un modèle à base de réseau de neurones doit apprendre de lui-même à interpréter

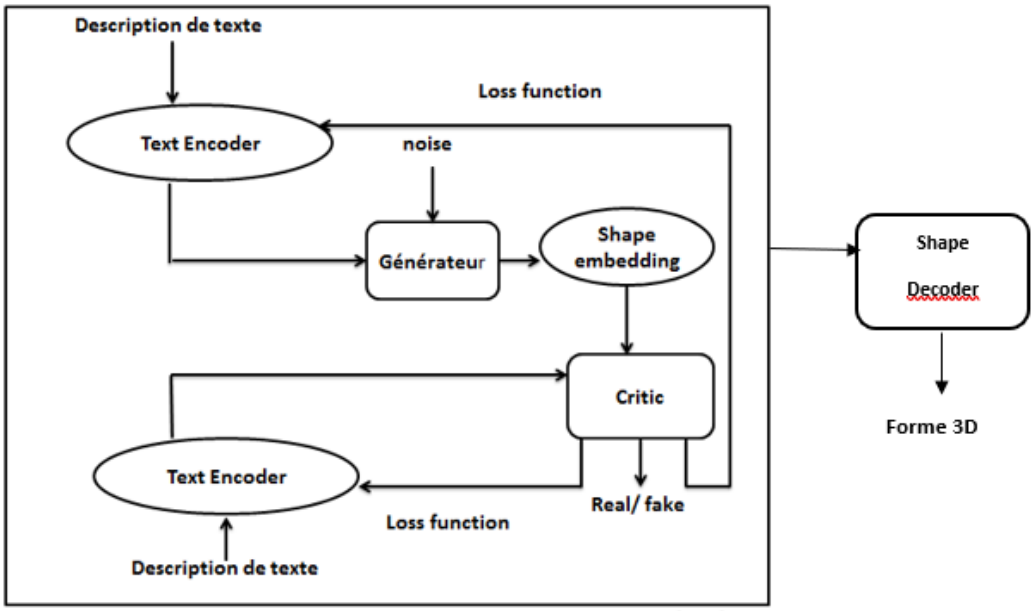

Figure 3.1: L'architecture générale du projet

de façon sémantique un dataset donné.

La solution proposée doit être insérée dans un autre projet : Génération de formes 3D à partir de courtes descriptions textuelles.

Il est illustré dans la figure suivante :

Le projet est constitué d'un GAN (Generative Adversial Network). Il est composé de deux blocs : le générateur et le discriminateur (Critic).

- Le rôle du générateur est de produire une forme 3D à partir d'une description textuelle. Il a comme entrée une description textuelle encodée avec le modèle BERT. Sa sortie est une forme 3D encodée également avec l'une de nos approches (shape embedding).

- Le rôle du discriminateur (critic) est d'évaluer la forme générée et de donner son avis au générateur pour qu'il s'améliore via la loss fonction (Mean Square Error).

A la fin, la forme 3D finale est obtenue avec le Shape decoder. Il va décoder la

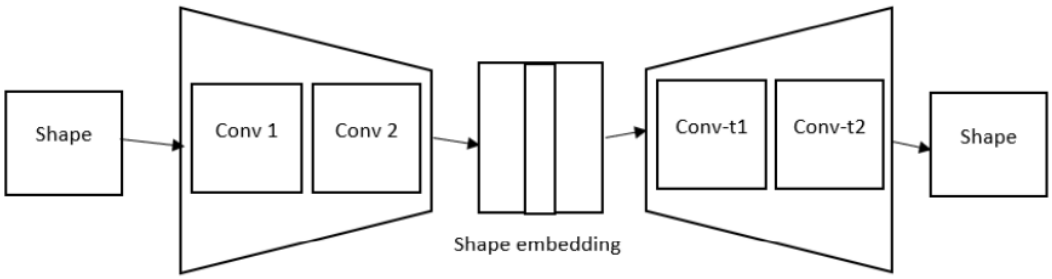

Figure 3.2: Architecture de l'autoencodeur

représentation de la forme 3D résultante en une vraie forme 3D.
Notre travail se focalisera sur la partie shape encoder / shape decoder . Nous proposerons pour ce faire deux solutions différentes.

3.3 Solution à base d'auto-encoder

Nous pouvons générer l'embedding capturé par les principales caractéristiques de nos formes 3D en se concentrant sur les propriétés les plus importants qui distinguent chaque forme d'une autre et en ignorant tout ce qui est répétitif et non schématique. On propose ici une architecture à base d'autoencodeur qui possède deux blocs ; un encodeur et un décodeur. L'architecture globale est définie dans la figure suivante

3.3.1 L'encodeur

L'encodeur a comme entrée la forme 3D sous forme de voxels. Il va encoder la forme et compresser les informations nécessaires afin d'avoir un shape embedding. Chaque forme a son propore shape embedding. Deux formes semblables auront des embeddings similaires également
L'architecture de l'encodeur est composée de deux couches conv3d (couches de con-

volution à 3 dimensions) suivies à la fois d'une fonction d'activation RELU (qui produit la valeur x si elle est positive, sinon elle produira 0). La première couche de convolution embarque un noyau de taille (32, 32, 32) et un stride de (2,1,1) avec un padding=(4, 2, 0) tandis que la seconde couche convolutive a un noyau de taille (5, 5, 1) sans stride et sans padding (tous deux définis sur 1).

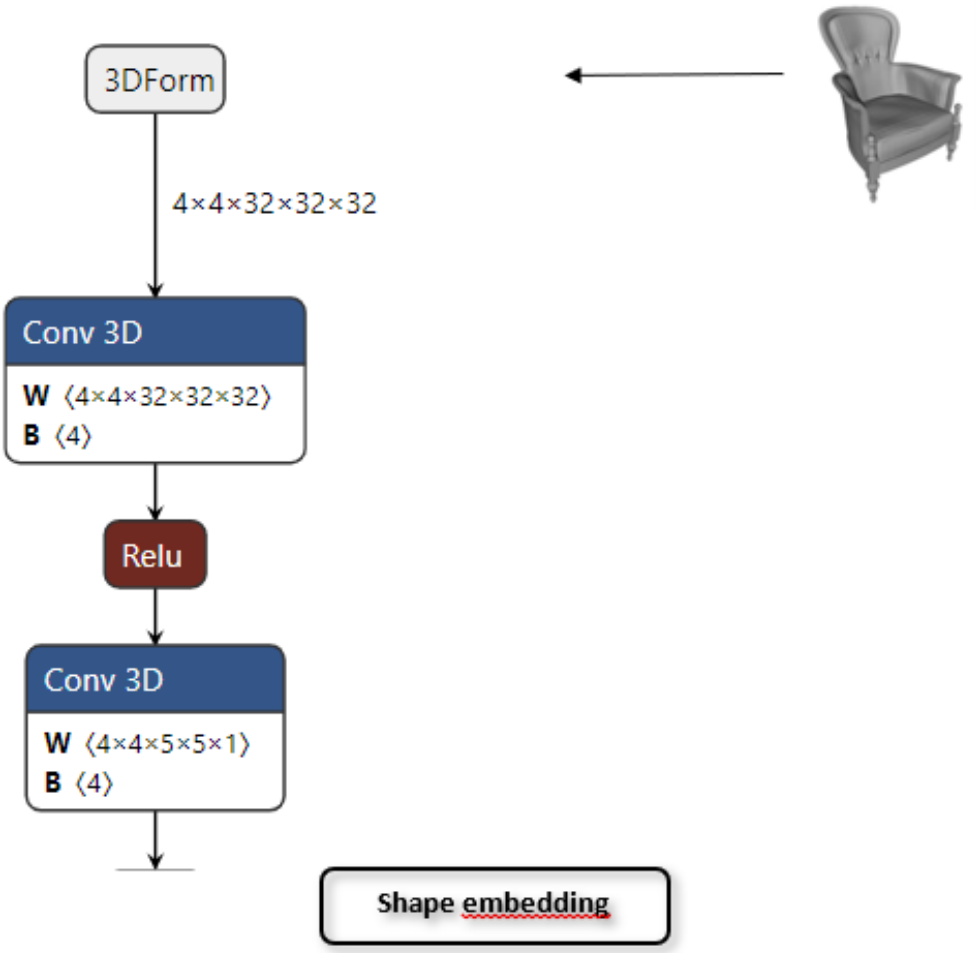

Figure 3.3: Architecture de l'encodeur

La taille du shape embedding est variable, nous réaliserons une étude expérimentale dessus pour étudier son impact sur la génération de la forme 3D.

3.3.2 Le décodeur

Le décodeur a un rôle inverse de l'encodeur. Il aura comme entrée le shape embedding et sa sortie sera la forme 3D mise en entrée de l'encodeur.
Vu qu'il fait le travail inverse, on utilisera une architecture inverse à celle de l'encodeur en utilisant des blocs de déconvolution.

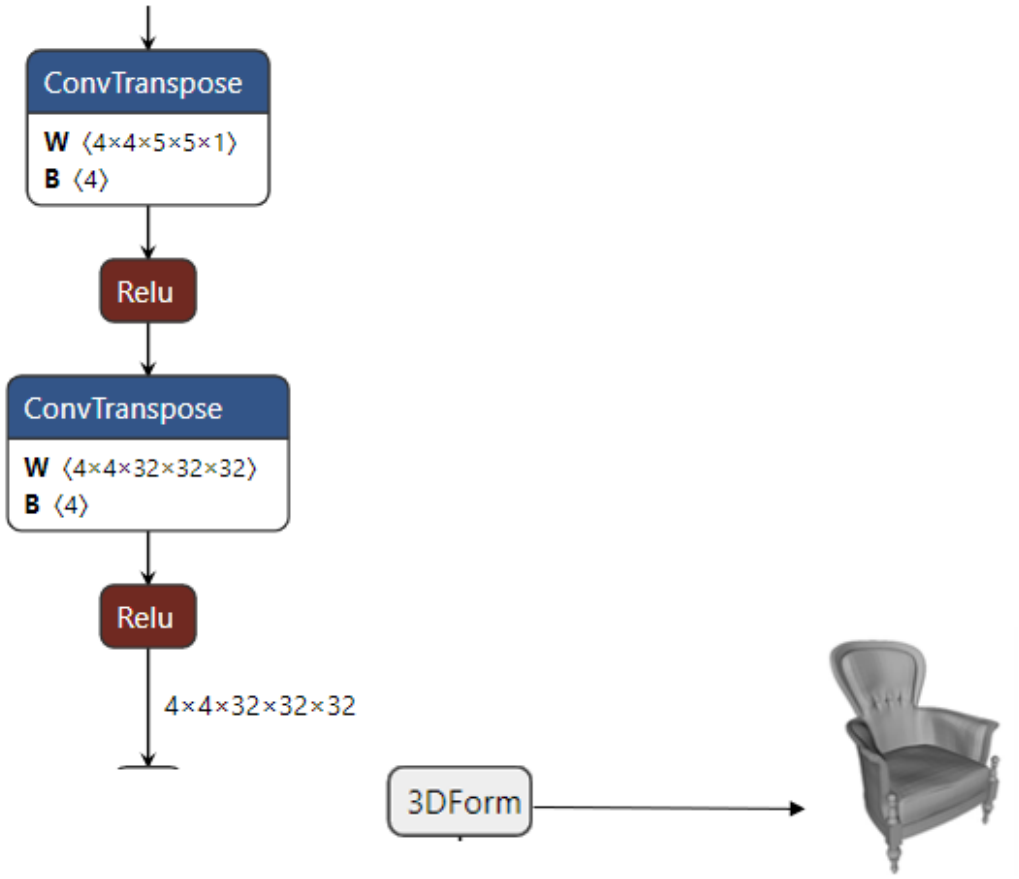

Figure 3.4: Architecture du décodeur

On applique mean squared error loss pour améliorer le modèle autoencodeur entre la forme générée par le décodeur et la forme en entrée dans l'encodeur et nous mettrons à jour les poids tout au long du processus de retro propagation.

Bien que cette solution soit efficace et rapide à intégrer, elle ne capture pas les

différences sémantiques. En effet, pour un auto-encodeur à base de CNN, de petites différences dans une même forme mènent à une interprétation différente. En effet, une même forme posée à l'envers sera vue différemment par le modèle. Les figure 5.a et 5.b mèneront à une interprétation différente même s'il s'agit de la même forme.

Figure 3.5: Différence sémantique

Afin de remédier à ce problème, nous présenterons une deuxième architecture à base de transformer.

3.4 Solution à base de transformer

Dans le domaine du traitement automatique du langage naturel, les méthodes à base d'auto-apprentissage supervisé sont bien développées. Le modèle BERT a gagné un succès énorme avec son approche d'apprentissage de texte. Bien que simple, elle s'est avérée efficace pour obtenir une représentation vectorielle sémantique des mots d'une langue donnée.

3.4.1 Méthode de self-learning dans BERT

Avant d'introduire des séquences de mots dans BERT, 15% des mots de chaque séquence sont remplacés par un jeton [MASK]. Le modèle tente ensuite de prédire

la valeur d'origine des mots masqués, en fonction du contexte fourni par les autres mots non masqués de la séquence. Techniquement, la prédiction des mots de sortie nécessite :

1. Ajout d'une couche de classification au-dessus de la sortie de l'encodeur.
2. Calculer la probabilité de chaque mot du vocabulaire avec softmax.

Dans l'exemple ci-dessous, c'est le mot w4 qui est masqué. Le modèle BERT tentera alors de prédire sa valeur en utilisant les mots du reste de la séquence :

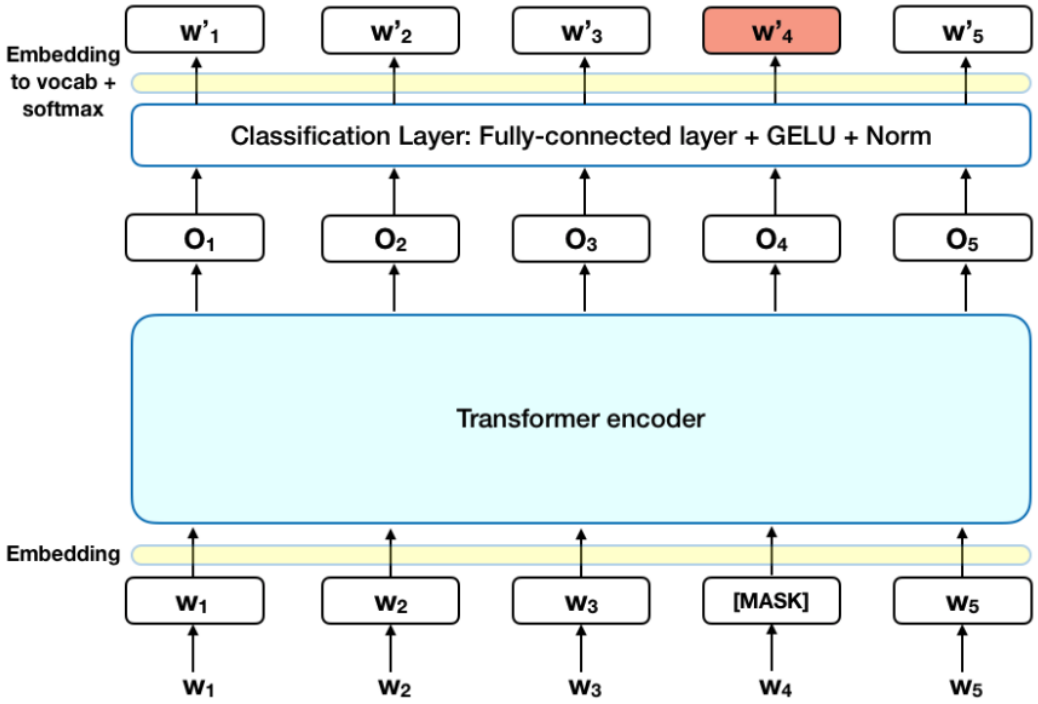

Figure 3.6: Modèle d'apprentissage dans BERT [29]

Lors de l'apprentissage, la fonction de perte dans BERT ne prend en considération que la prédiction des valeurs masquées et ignore la prédiction des mots non masqués. En conséquence, le modèle converge plus lentement que les modèles directionnels, une caractéristique qui est compensée par sa sensibilité accrue au contexte. [29]

3.4.2 Application du paradigme de self-learning en 3D

Dans notre approche, on essaiera d'appliquer le même paradigme utilisé dans le texte en masquant certaines régions des formes 3D.

Contrairement aux images en vision par ordinateur qui peuvent être naturellement divisées en morceaux d'images régulières, les formes 3D sont constituées de points non ordonnés dans l'espace.

En se basant sur cette propriété, nous traitons les formes 3Ds d'entrée en trois étapes :

- Génération de blocs de points
- Masquage des blocs.
- Obtention des shape Embedding

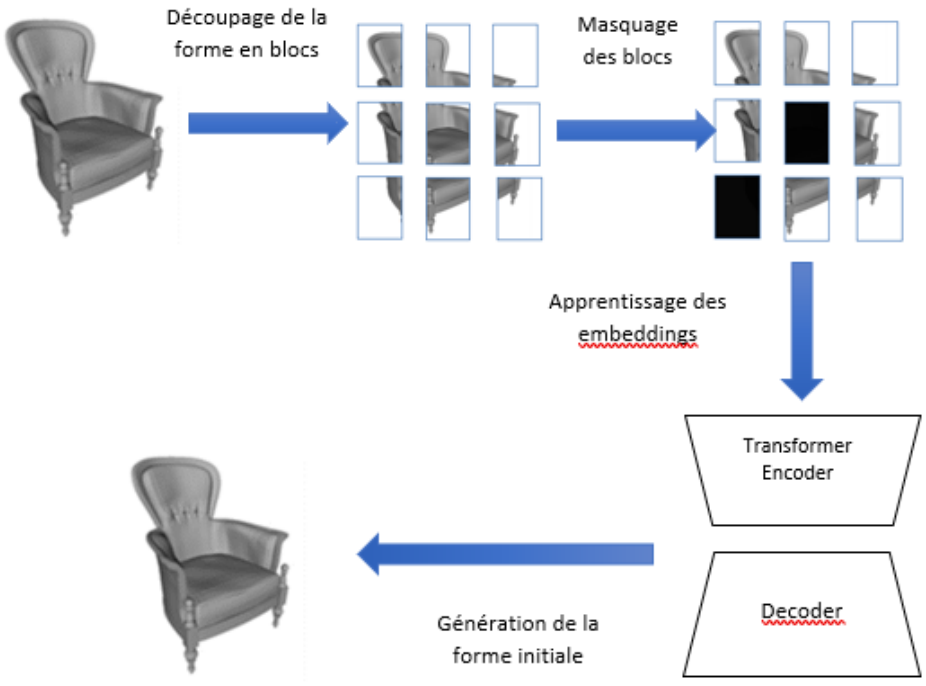

Figure 3.7: Paradigme d'apprentissage adopté

Nous allons détailler chacune de ces phases dans ce qui suit.

3.4.2.1 Génération des blocs de points

Lors de cette phase, la forme initiale va être subdivisée en blocs (patchs) . Si la taille de la forme est (h, w, d) .
où :

- h: et la hauteur de la forme
- w: et la largeur de la forme
- d: et la profondeur de la forme

Alors cette forme sera décomposée en une séquence de blocs dont le nombre M est défini comme suit :

$$M = \frac{hwd}{p^3} \tag{3.1}$$

Où: $p$est la taille d'un patch (p= hauteur = profondeur = largeur d'un patch) . Elle sera définie par l'utilisateur

3.4.2.2 Masquage des blocs.

Les blocs générés peuvent se chevaucher entre eux. Dans le processus du masquage, on essayera de séparer les blocs adjacents. On utilise un ration de masquage initiale $masque_{ratio}$. Les blocs seront masqués de façon aléatoire selon ce ratio. Si un bloc est masqué, ses blocs voisins auront un ratio de masquage égal à 0 . Il sera réinitialisé une fois qu'un bloc voisin est dépassé.

3.4.2.3 Obtention des shape Embeddings

Pour représenter chaque patch , on utilisera un autoenconder à base de Transformer similaire à celui utilisé dans BERT.

- **Partie d'encodage :**

  Notre encoder aura comme entrée les blocs non masqués seulement. Afin de mettre en évidence la position des blocs , on utilisera également une technique de positional embedding où la position d'un bloc dans une forme aura un certain poids p qui influera sur l'entrainement de l'encodeur. Cette information est nécessaire car elle permettra par la suite au décodeur de trouver la position du bloc lors de la reconstruction de la forme 3D.

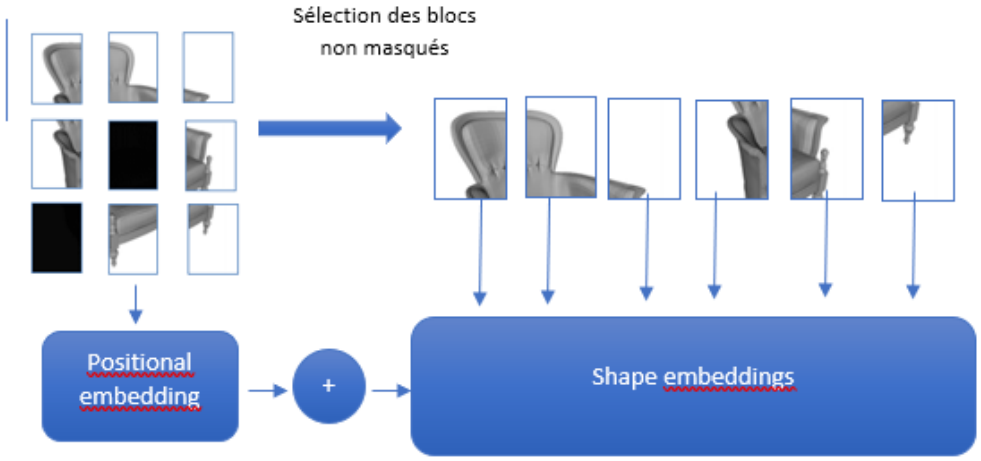

Figure 3.8: Encoder à base de transformer

- **b. Partie du décodage :**

  Le décodeur aura une architecture similaire à l'encodeur. Il sera constitué du même nombre de couche de transformers. Il aura comme entrée les représentations des blocs masqués et non masqués. On y introduira aussi la représentation des positions des blocs. Son rôle est de reproduire les blocs masqués.

  Une dernière couche à la sortie sera rajoutée (une couche complétement connectée) qui servira à retrouver la position des blocs masqués.

L'architecture finale est résumée dans le schéma de la figure suivante :

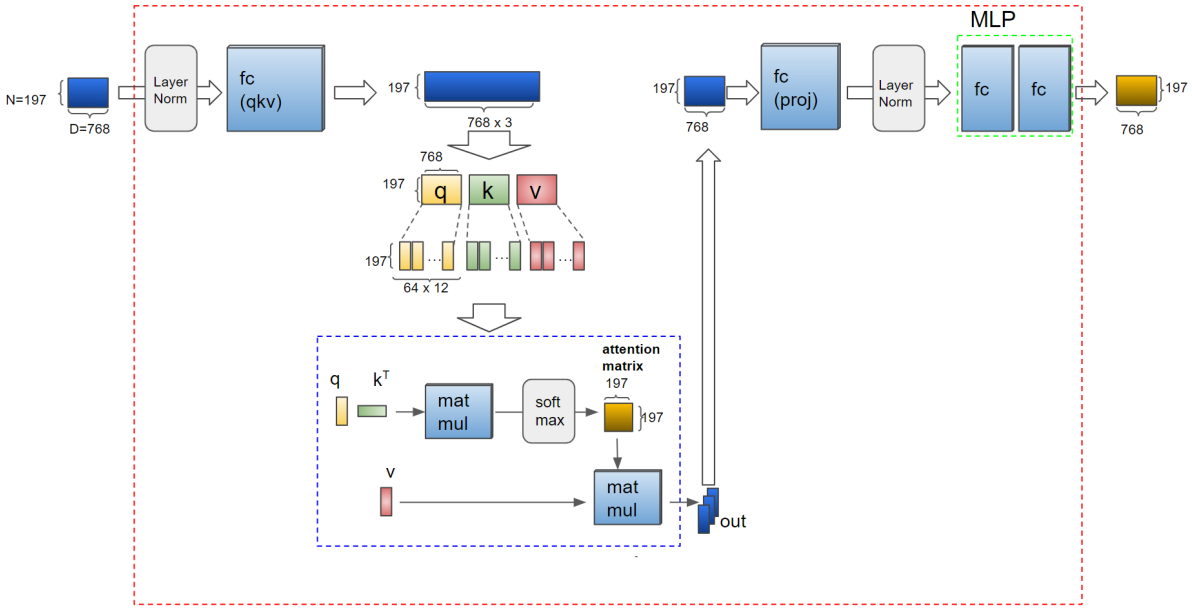

Figure 3.9: Architecture du bloc transformer

La taille de l'entrée encodée est 768 . 197 positions sont dédiées à l'apprentissage de la position des blocs , quant aux autres positions , elles serviront à encoder le bloc en lui-même.

L'architecture utilisée est la même que pour BERT. Les blocs sont répétés 12 fois jusqu'à obtention de la sortie.

La fonction de perte utilisée ici pour l'apprentissage de notre modèle est MSE également pour qu'on puisse comparer ce modèle au précédent. Toutefois, il est préférable d'utiliser d'autres fonctions de pertes plus pertinentes pour les transformers tel que la distance Chamfer.

3.5 Conclusion

Dans ce chapitre nous nous sommes penchés sur l'approche proposé. Nous avons présenté l'architecture du projet global. Nous avons par la suite expliqué en détails les deux approches de représentation utilisées dans notre travail.
La première approche est basée sur un autoencoder simple à base de CNN. La deuxième approche est basée sur les transformer pour le shape embadding.

Dans le prochain chapitre, nous présenterons les outils utilisés et nous mènerons une série d'expérimentations pour l'évaluation de notre travail.

Chapitre 4

L'implémentation Et La Discussion Des Résultats Obtenue

4.1 Introduction

Dans le chapitre précèdent nous avons expliqué l'architecture du modèle proposé. Dans ce qui suit, nous allons présenter les outils utilisés, nous décrirons le dataset utilisé, nous présenterons les métriques d'évaluations employées et nous finirons par une étude expérimentale sur nos modèles proposés.

4.2 Les outils utilisés

1. **Google collabotary**

   Également connu sous le nom de Colab est un environnement de bloc-notes Jupyter gratuit qui s'exécute dans le cloud et stocke ses blocs-notes sur Google Drive., il permet d'écrire et d'exécuter Python dans un navigateur, sans avoir recours à une configuration au préalable , accès gratuit aux GPU et partage facile, depuis qu'il utilise google drive vous pouvez facilement lire les données

du lecteur dans Colab et l'utiliser, ce qui est l'un des meilleurs fonctionnalités disponibles pour les développeurs du monde entier.

2. **Python**

   Python est un langage de programmation qui est devenu un incontournable de la science des données, permettant aux analystes de données et à d'autres professionnels d'utiliser le langage pour mener calculs statistiques complexes, créer des visualisations de données, créer des algorithmes d'apprentissage automatique, manipuler et analyser des données et effectuer d'autres opérations liées aux données Tâches. Python peut créer une large gamme de visualisations de données différentes, comme la ligne et des graphiques à barres, des camemberts, des histogrammes et des tracés 3D. Python a aussi un nombre de bibliothèques qui permettent aux codeurs d'écrire des programmes pour l'analyse de données et la machine apprendre plus rapidement et efficacement, comme TensorFlow, Pytorch et Keras.

3. **Outils de développement front-end**

   Pour notre interface nous avons utilisé HTML, CSS et JS, dans l'interface il y a un texte zone donc l'utilisateur peut mettre une petite description décrire la forme qu'il veut, et cliquez sur générer la description traitée dans notre backend et générer la forme.

4. **F3D**

   F3D est un programme de bureau créé par c++ et peut afficher des formes 3D dans NRRD format (le format principal que nous utilisons dans notre travail). Après que notre modèle a généré les formes nous glissons et déposons le fichier NRRD généré dans le programme et il l'affiche.

5. **Visual Studio Code**

C'est un éditeur de code simplifié prenant en charge le développement opérations telles que le débogage, l'exécution de tâches et le contrôle de version. Il vise à fournir juste les outils dont un développeur a besoin pour un cycle rapide de code-construction-débogage et laisse workflows plus complexes vers des IDE plus complets

6. **NumPy**

   NumPy est un projet open source visant à permettre le calcul numérique avec Python. Il a été créé en 2005, en s'appuyant sur les premiers travaux du Numeric and Bibliothèques Numarray.

7. **NRRD**

   Nrrd est une bibliothèque et un format de fichier pour la représentation et le traitement de données raster à n dimensions. Il a été développé par Gordon Kindlmann pour prendre en charge les applications de visualisation scientifique et de traitement d'images.it peut être utilisé, consulté et modifié via la bibliothèque Python Pynrrd.

8. **BERT**

   Les représentations d'encodeurs bidirectionnels à partir de transformateurs (BERT) sont une technique d'apprentissage automatique basée sur les transformateurs pour la préformation en traitement du langage naturel (TAL) développée par Google. il est conçu pour pré-entraîner en profondeur les représentations bidirectionnelles à partir de texte non étiqueté en conditionnant conjointement contexte à gauche et à droite. En conséquence, le modèle BERT préformé peut être affiné avec une seule couche de sortie supplémentaire pour créer des modèles à la pointe de la technologie. pour un large éventail de tâches .

9. **PyTorch**

   PyTorch est une bibliothèque d'apprentissage automatique open source basée sur la bibliothèque Torch, utilisé pour des applications telles que la vision par ordinateur et le traitement du langage naturel, principalement développé par le laboratoire de recherche sur l'IA de Facebook (FAIR). Bien que le Python l'interface est plus raffinée et l'objectif principal du développement, PyTorch également possède une interface C++.

4.3 Dataset utilisé

Pour mener à bien notre étude, nous avons utilisé le dataset ShapeNet. ShapeNet est un vaste référentiel riche en informations de modèles 3D. Il contient des modèles couvrant une multitude de catégories sémantiques. Il fournit de vastes ensembles d'annotations pour chaque modèle et les liens entre les modèles du référentiel et d'autres données multimédia hors du référentiel. [29]

ShapeNet fournit une vue des données contenues dans une catégorisation hiérarchique, contrairement à d'autres modèles référentiels. Il fournit également un riche ensemble d'annotations pour chaque forme. Les annotations comprennent les attributs géométriques tels que les vecteurs d'orientation, les normes, les symétries de forme et l'échelle de l'objet dans les unités du monde réel. Ces attributs fournissent des ressources précieuses pour traiter, comprendre et visualiser les formes 3D selon de la sémantique de la forme.

Dans notre étude, nous nous sommes focalisés sur deux catégories d'objets du dataset ShapeNet, à savoir les tables et les chaises (avec 8 447 et 6 591 cas, respectivement). Ces formes 3D ont été créées par des concepteurs pour représenter avec précision des objets réels.

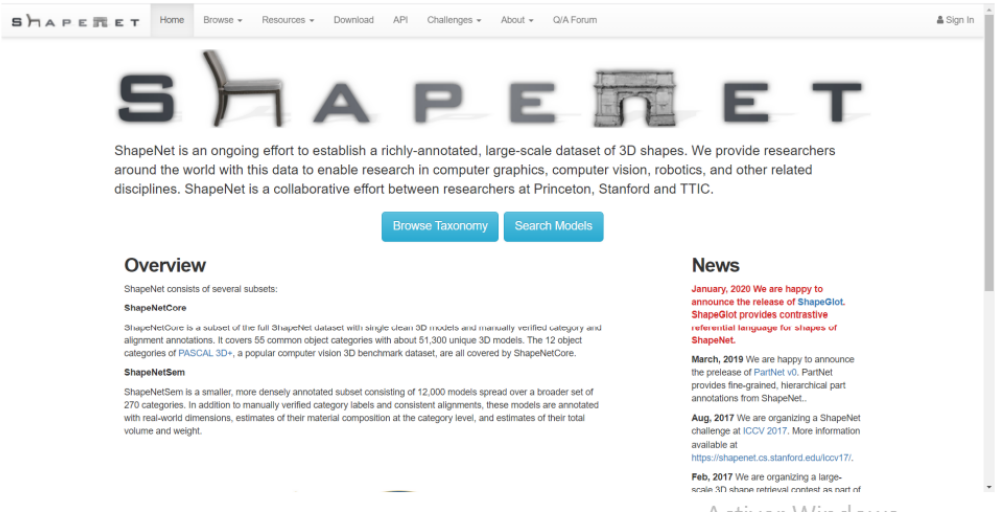

Figure 4.1: Page d'accueil Shapenet
[30]

Figure 4.2: Exemple de dataset shapenet table et chaise

Nous avons choisi les catégories de table et de chaise car elles contiennent de nombreuses positions avec des variations d'attributs fines dans la géométrie, la couleur et le matériau. Nous augmentons cet ensemble de données de formes avec 75 344 descriptions en langage naturel (5 descriptions en moyenne par forme) fournies par des personnes sur Amazon et un dataset contrôlé et généré de manière procédurale de primitives géométriques 3D. Cet ensemble de données à grande échelle fournit de nombreuses descriptions complexes en langage naturel associées à des formes 3D réalistes.

Figure 4.3: Les formes avec leurs descriptions

Pour permettre une évaluation quantitative systématique de nos modèles, nous utilisons un ensemble de données de primitives géométriques 3D avec les descriptions textuelles correspondantes. Ces données ont été générées en voxélisant 6 types de primitives (cuboïdes, ellipsoïdes, cylindres, cônes, pyramides et tores) en 14 variations de couleurs et 9 variations de taille,les variations de couleur et de taille sont soumises à des perturbations aléatoires générant 10 échantillons de chacune des 756 configurations primitives possibles, créant ainsi 7560 formes voxélisées.

4.4 Expérimentations

Dans cette partie, nous jetterons un coup d'œil sur les différentes architectures utilisés dans notre travail.

Notre expérimentation est subdivisée en deux parties ; une évaluation intrinsèque et une évaluation extrinsèque.

4.4.1 Evaluation intrinsèque :

Ici, nous allons nous focaliser sur le bon fonctionnement de nos modèles. Nous allons utiliser comme métrique la fonction de perte pour surveiller le comportement de nos modèles par rapport aux différents hyper paramètres.

4.4.1.1 Comportement de l'autoencoder par rapport au nombre d'épochs

Pour évaluer notre autoencoder, nous allons faire varier le nombre d'épochs et nous allons fixer les autres paramètres. Voici les valeurs prises :

- Batch size=8
- Le nombre de worker=4
- Optimizer=Adam
- Learning rate=e-3
- Taille du vecteur latent=384

La capture suivante montre l'évolution de la fonction de perte de l'autoencoder en augmentant le nombre d'épochs :

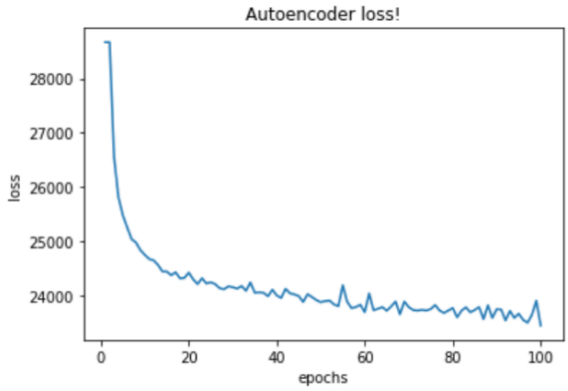

Figure 4.4: Effet du nombre d'epochs sur l'autoencodeur

Discussion :

On constate qu'au bout d'une centaine d'époch, la valeur de perte de l'autoencodeur reste inchangée. Cela montre que le modèle converge au bout de ce nombre d'itérations et qu'il serait inutile de faire d'avantage d'epochs.

4.4.1.2 Comportement de l'autoencoder par rapport à la taille du vecteur latent

Nous allons maintenant tester le comportement de l'autoencoder par rapport à la taille du vecteur latent en sortie (shape embeddings)
Nous allons utiliser l'autoencodeur avec les paramètres suivants :

- Nombre d'epochs :100
- Batch size :32
- Numbre de worker=4
- Learning rate=0.0002

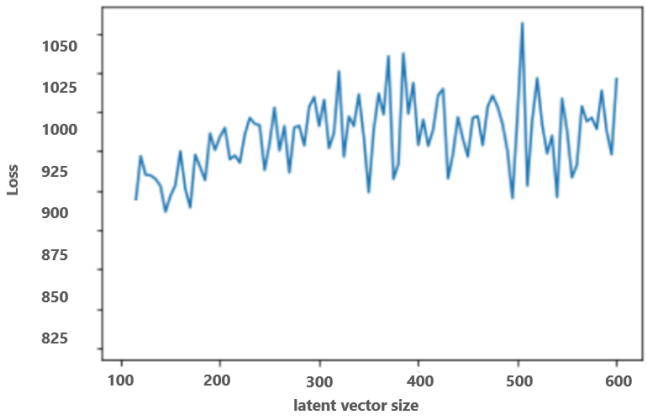

Figure 4.5: Effet de la taille du latent vector sur l'autoencodeur

Discussion :

Ici, on ne peut pas vraiment se prononcer. On voit qu'il n'y a pas d'incidence directe entre l'augmentation de la taille du latent vector et de la valeur loss. Il est toutefois préférable de choisir des valeurs assez élevées pour mieux capturer les relations sémantiques mais pas trop non plus car il y a consommation de l'espace mémoire. Ici, nous allons garder la valeur de référence 384.

4.4.1.3 Comportement du transformer par rapport au nombre d'époch

De la même façon, nous allons étudier l'impact du nombre d'époch sur notre modèle à base de transformers.
Voici les valeurs prises :

- Batch size=8
- Le nombre de worker=4
- Optimizer=Adam
- Learning rate=e-3
- Taille du vecteur latent=768

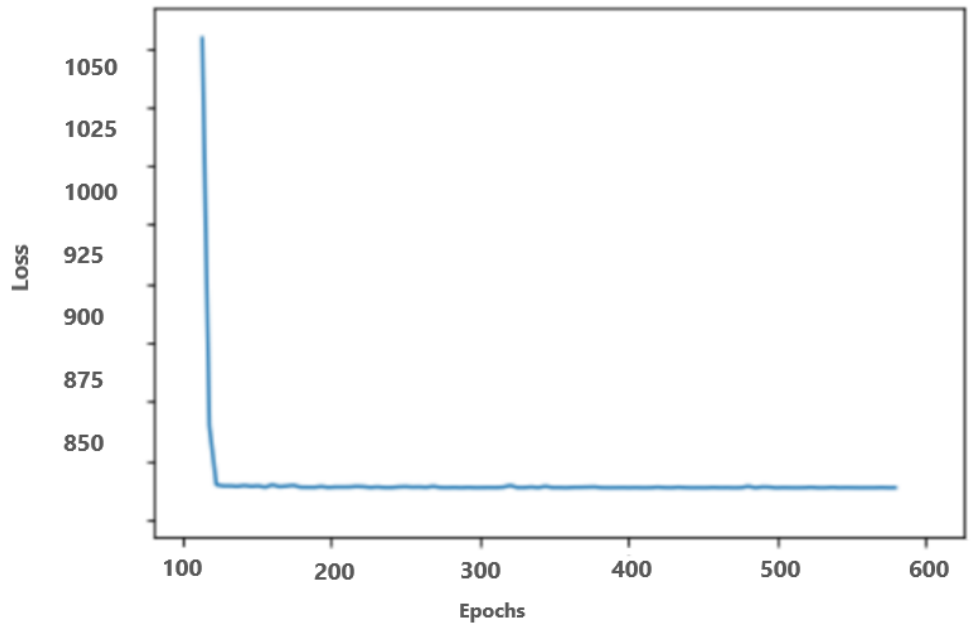

Figure 4.6: Effet du nombre d'epochs sur le transformer

Discussion :

Les modèles à base de transformers sont plus lents à converger contrairement aux modèles à base de CNN, toutefois , ici, on constate que la convergence s'est faite une fois la barre des 100 epochs franchie. Et qu'en général, la valeur de perte obtenue est réduite par rapport au modèle à base de CNN.

4.4.1.4 Comportement du transformer par rapport à la taille du vecteur latent

Nous allons maintenant tester le comportement du transformer par rapport à la taille du vecteur latent en sortie (shape embeddings)

Nous allons utiliser l'autoencodeur avec les paramètres suivants :

- Nombre d'epochs :200
- Batch size :64

- Numbre de worker=4
- Learning rate=0.0002

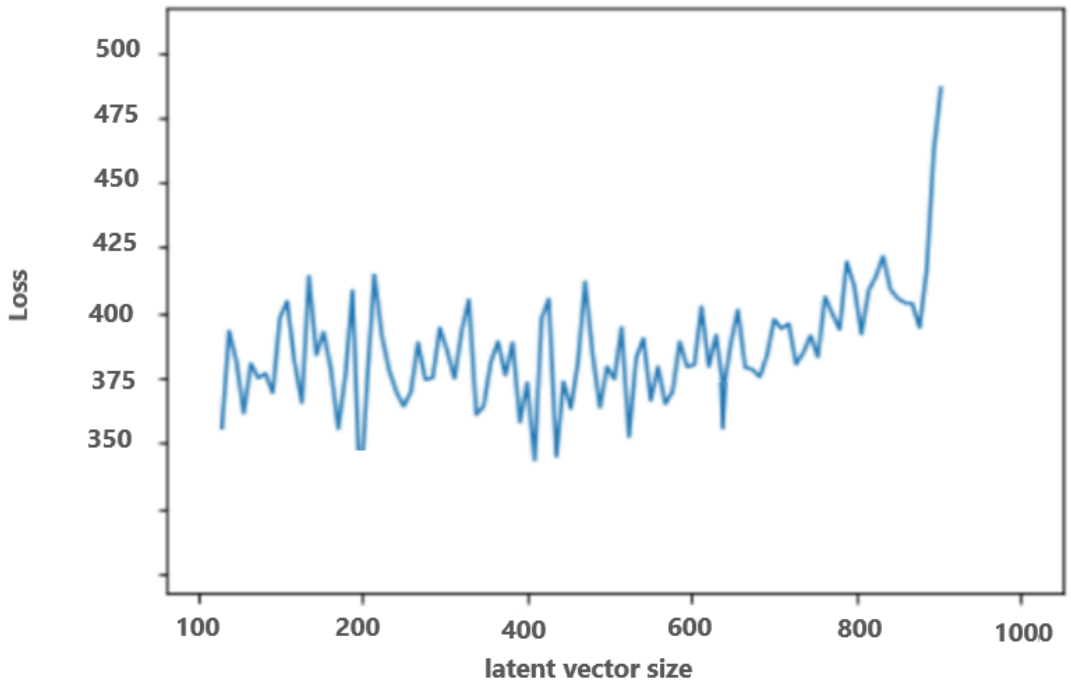

Figure 4.7: Effet de la taille du latent vector sur le transformer

Discussion :

La taille des embeddings joue un rôle important au niveau des transformers. Ici , on constate qu'au bout d'une taille dépassant 900, le modèle diverge. Ceci est expliqué par le manque de mémoire. Une fois ce seuil dépassé, nous n'avons pas pu entrainer notre modèle correctement, de ce fait , la valeur de loss a augmenté. Mais on a constaté plus ou moins des valeurs stables aux alentours de 700. Dans ce qui suit , nous allons travailler avec la valeur 768 qui est la valeur avec le plus bas dans l'intervalle 700-800.

4.4.1.5 Comportement du transformer par rapport à la taille des blocs

La taille des patchs ou blocs est tout aussi importante que la taille du vecteur latent. Toutefois, vu que les formes sur lesquelles nous avons travaillé sont de petites taille (32*32*32), il est difficile de trop découper ces formes-là. Nous avons opté pour deux découpages possibles : 8x8x8 et 16x16x16

Pour les deux cas, nous avons opté pour les mêmes valeurs des hyperparamètres :

- Nombre d'epochs :200
- Batch size :64
- Numbre de worker=4
- Learning rate=0.00001
- Vecteur latent = 768

Voici les résultats obtenus :

Modèle	Valeur de loss moyenne au bout de 200 epochs
Blocs 8x8x8	768
Blocs 16x16x16	696

Table 4.1: Modèle Valeur de loss moyenne au bout de 200 epochs par rapport au modeles

Discussion :

On ne peut rien dire ici car l'échantillon est faible et la différence n'est aps significative. Mais pour notre travail et pour accélérer les calculs, nous avons préféré utiliser des blocs de taille 16x16x16

4.4.2 Evaluation extrinsèque :

Dans ce type d'évaluation, nous allons tester les performances de notre modèle dans une autre tâche, à savoir la génération de formes 3D depuis une courte description textuelle.

Le modèle utilisé pour la génération de forme 3D et un WGAN. Il a comme entrée la description textuelle sous forme de text embedding obtenu via BERT et comme sortie, la forme 3D représentée avec l'un de nos modèles.

Nous allons comparer nos deux modèles avec le modèle WGAN sans représentation 3D. Ça sera notre modèle de référence.

Voici les paramètre choisis pour nos deux modèles :

Modèle à base de transformer	Modèle à base d'autoencoder
Nombre d'epochs :200	Nombre d'epochs :100
Batch size :64	Batch size :32
Numbre de worker=4	Numbre de worker=4
Learning rate =0.00001	Learning rate =0.0002
Vecteur latent = 768	Vecteur latent = 384

Table 4.2: Les performances de diffèrent modèles

Les résultats obtenus sont exprimés dans le tableau suivant :

Modèle	Valeur de précision (Acc)
WGAN Simple	93.0%
WGAN avec autoencoder	71.8%
WGAN avec Transformer	79.7%

Table 4.3: Valeurs de précision des modèles

Nous avons opté pour la mesure Accuracy ici pour estimer le rendu des formes 3D obtenues.

Discussion :

On constate ici que le meilleur modèle est celui du WGAN simple. Cela s'avère logique, car le modèle est entrainé directement sur les formes 3D d'origine et il n'y a pas de perte d'information des formes en entrées. Pour les deux autres modèles, ils ont été entrainés sur une donnée compressée. Il est tout à fait logique que ces modèles aient de moins bons résultats. Toutefois, il est à noter que le modèle à base de transformer à un meilleur score que le modèle à base d'autoencoder . Ceci s'explique par la qualité de la représentation de la 3D générée par le transformer que nous avons déjà pu constater dans les tests précédents.

4.5 Conclusion

Ce chapitre a été consacré à la présentation des différents outils de développement de l'approche ainsi que le dataset utilisé dans l'application pour la représentation des formes 3D nous avons aussi discuté des différents résultats retrouvés. Afin de bien évaluer nos modèles nous avons mené deux types d'évaluation : intrinsèque et extrinsèque. Les résultats s'avèrent concluants.

Conclusion et Perspectives

Le domaine de l'apprentissage automatique et en particulier l'auto apprentissage supervisé ont connu un grand engouement par les chercheurs ces dernières années. De nouvelles architectures sont apparues telles que les autoencoders, les transformers ou encore les constractive models.

Ces architectures ont trouvé de nombreux domaines d'application surtout dans le cadre de traitement d'image et de traitement de texte. Toutefois, peu de chercheurs se sont penchés sur le domaine de la 3D qui s'avère tout aussi important que les autres.

A travers ce travail nous avons proposé des modèles à base de réseaux de neurones permettant de représenter les formes 3D et d'en capturer les caractéristiques les plus importantes afin de les utiliser dans une application donnée.

Nous avons proposé un premier modèle autoencoder basé sur les réseaux de neurones convolutifs. Ce modèle est inspiré dans ce qui se fait dans le traitement d'images. Il permet de compresser les formes 3D de façon significative, toutefois, il reste très sensible aux changements dans une même forme.

Nous avons par la suite proposé un autre modèle à base de Transformer. Nous avons appliqué le paradigme d'apprentissage du modèle BERT dans le cadre de la 3D. A chaque fois, on essaie de cacher certaines parties de la forme 3D, et c'est à notre modèle d'essayer de la compléter. Ce type d'apprentissage permet au modèle d'être robuste aux différentes variations que peut subir une forme 3D.

Pour vérifier l'efficacité de nos modèles, nous avons mené deux types d'expérimentations. Le premier type est une expérimentation intrinsèque où nous avons testé nos modèles tels quels sur un jeu de données précis (Shapenet). Par la suite, nous avons mené une expérimentation extrinsèque où nous avions mis en épreuve nos modèles dans une application bien précise : la génération de formes 3D à partir de courtes descriptions textuelles.
Les résultats ont montré que le modèle à base de transformer est plus stable une fois bien entrainé.
Notre travail reste cependant qu'une ébauche à d'éventuels d'autre travaux. Afin de l'améliorer, nous proposons comme perspectives :

- Utiliser d'autres datasets mis à part Shapenet.
- Explorer d'autres types d'architectures neuronales (Diffusion models , Clip models . . .)
- Proposer un meilleur découpage des formes 3D en utilisant des techniques de clustering sur les nuages de points 3D.
- Utiliser de meilleures métriques afin d'évaluer le réalisme des résultats

Références

[1] *Réseaux de Neurones.* https://www-lisic.univ-littoral.fr/

[2] T. M. Mitchell (1997) *Machine Learning,*McGraw-Hill International Editions chap. 13 Reinforcement Learning, p. 367-390.

[3] G. Zaccone & K. Rezaul (2018) *Deep learning with tensorflow : Explore neural networks and build intelligent systems with python*, 2nd edition, Packt Publishing.

[4] A. Fandango (2020) *Mastering tensorflow 1. x : Advanced machine learning and deep learning concepts using tensorflow 1. x and keras,*Packt Publishing.

[5] F. q. lauzon (2012) *an introduction to deep learning,,*11th international conference on infor- mation science, signal processing and their applications (isspa), 2012, pp. 1438-1439, doi: 10.1109/isspa.2012.6310529, 2012.

[6] S. Anunaya, (10 08 2021) *Data Preprocessing in Data Mining -A Hands On Guide*, [En ligne]. Available: https://www.analyticsvidhya.com/blog/2021/08/data-preprocessing-in-data-mining-a-hands-on-guide/. [Accès le 08 03 2022].

[7] M. Bernico, (2018) *Deep Learning Quick Reference : Useful Hacks for Training and Optimizing Deep Neural Networks with TensorFlow and Keras.*

[8] U. Michelucci, (2022) *Applied Deep Learning with TensorFlow 2: Learn to Implement Advanced Deep Learning Techniques with Python.*New York: Apress.

[9] I. Vasilev, (2019) *Advanced Deep Learning with Python,* Packt Publishing.

[10] N. Ketka et J. Moolayil, (2021)*Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch,* New York: Apress.

[11] A. Biswal, (21 02 2022) *Recurrent Neural Network (RNN) Tutorial: Types, Examples, LSTM and More,* . [En ligne]. Available: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn. [Accès le 07 03 2022].

[12] https://openclassrooms.com/fr,[Enligne].Available:https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees\-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn.

[13] V. E. Irekponor, (28 05 2020.)*Mathematical Prerequisites For Understanding Autoencoders and Variational Autoencoders (VAEs),* [En ligne]. Available: https://medium.com/analytics-vidhya/mathematical-prerequisites-for\-understanding-autoencoders-and-variational-autoencoders-vaes. [Accès le 2022 03 12].

[14] I. Goodfellow, (05 12 2016) *Generative Adversarial Networks,* Google. [En ligne]. Available: https://developers.google.com/machine-learning/gan/gan_structure. [Accès le 27 12 2021].

[15] M. Lanham, (2021) *Generating a New Reality: From Autoencoders and Adversarial Networks to Deepfakes,* New York: Apress.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser et I. Polosukhin,(2017) *Attention is all you need,* Advances in neural information processing systems, vol. 30.

[17] A. Géron, (2019) *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems.*O'Reilly Media..

[18] B. Mohamed, *Deep Learning for Detecting and Identifying Blinding Retinal Diseases,* Thèse de master LMD en Informatique, sous la direction de Abdelouahab.

[19] S. Sayad, *Model Evaluation- Regression,* [En ligne]. Available: ://www.saedsayad.com/model_evaluation_r.html.

[20] Andrew Brock, Theodore Lim, J.M. Ritchie *Generative and Discriminative Voxel Modeling with Convolutional Neural Networks* School of Engineering and Physical Sciences Heriot-Watt University Edinburgh, UK.

[21] Maxim Tatarchenko, Alexey Dosovitskiy, Thomas Brox *Multi-view 3D Models from Single Images with a Convolutional Network* Department of Computer Science University of Freiburg.

[22] Yutong Feng, Yifan Feng, Haoxuan You, Xibin Zhao1 , Yue Gao1*MeshNet: Mesh Neural Network for 3D Shape Representation* 1BNRist, KLISS, School of Software, Tsinghua University, China. 2School of Information Science and Engineering, Xiamen University .

[23] *Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data.*

[24] *RGBD Datasets: Past, Present and Future Michael Firman University College London.*

[25] D. Maturana et S. Scherer, (2015) *Voxnet: A 3d convolutional neural network for real-time object,* IEEE/RSJ International Conference, p. 922–928.

[26] A. Brock, T. Lim, J. M Ritchie et N. Weston,(2016) *Generative and discriminative voxel modeling,* arXiv .

[27] H. Su, S. Maji, E. Kalogerakis et E. Learned-Miller,(2015) *Multi-view convolutional neural networks for 3d shape recognition,* In Proceedings of the IEEE international conference on computer vision, p. 945–953.

[28] K. Chen, C. B. Choy, M. Savva, A. X. Chang, T. Funkhouser et S. Savarese, (2018) *Text2shape: Generating shapes from natural language by learning joint embeddings,* springer.

[29] c. l. `https://towardsdatascience.com/bert-explained-state-\of-the-art-language-model-for-nlp-f8b21a9b6270.`

[30] (09 2022) [En ligne]. Available: https://shapenet.org/.