Reconstruction 3D et Géometrie dans l'espace

Juin, 2019

Auteurs **Keyvan Beroukhim** & **Sabrina Chaouche**

Encadrants **Nicolas Baskiotis** & **Vincent Guigue**

Résumé

Le but de ce projet est de travailler sur la reconstruction de structures 3D à partir d'image 2D, avec comme supervision des nuages de points.
Il s'agit de mettre en place une architecture capable d'analyser des images et d'estimer la forme des objets en sortie. Il faudra donc prendre en main des architectures de réseaux de neurones pour traiter les images puis bâtir une architecture au dessus. L'idée est de partir de travaux effectués par Thibault Groueix et une équipe de chercheurs lors d'un stage d'été chez Adobe Research.

Table des matières

1 Introduction

De nos jours, de nombreuses applications avancées nécessitent des informations en trois dimensions (3D). La troisième dimension joue un rôle décisif dans l'analyse des environnements dynamiques comme des environnements statiques.
Des champs d'application de la troisième dimension comprennent les domaines de la surveillance, la robotique, les voitures autonomes, ... qui exploitent les informations de la profondeur afin d'obtenir une meilleure analyse de l'environnement mais également les domaines du traitement d'images 3D, de la photographie numérique, des jeux, du multimédia, la visualisation 3D et la réalité augmentée.
Dans le domaine médical, la 3D est exploitée pour l'impression bio des tissus et organoïdes, la préparation à la chirurgie en répliquant un organe artificiel qui correspond à celui du patient pour que le chirurgien pratique dessus, ...

2 Etat de l'art en reconstruction 3D :

2.1 ImageNet Challenge :

Vu l'importance qu'a l'image dans différents domaines, plusieurs challenges ont été lancés dont le plus connu the ImageNet Large Scale Visual Recognition Challenge qui se base sur la base de données ImageNet.
La base de données ImageNet est le fruit d'une collaboration entre l'Université de Stanford et l'Université de Princeton. Elle est devenue une référence dans le domaine de la vision par ordinateur.
Elle contient à l'origine environ 14 millions d'images étiquetées avec Synsets (Synonym set : pour chaque mot, on a un set de synonymes) de l'arborescence lexique de WordNet (une base de données pour répertorier, classifier et mettre en relation de diverses manières le contenu sémantique et lexical de la langue anglaise) .
Le challenge initial consistait en une tâche de classification simple, chaque image appartenant à une seule catégorie parmi les 20000. Bien que le défi initial soit toujours d'actualité, il a évolué pour devenir une tâche de classification multiple avec des cadres de sélection autour de chaque objet individuel.
Le challenge d'ImageNet a été traditionnellement traité avec des algorithmes d'analyse d'image tels que SIFT avec des résultats mitigés jusqu'à la fin des années 90.
Cependant, un écart assez remarquable de performance a été crée en utilisant des réseaux de neurones. Inspiré par Y. Lecun et al. (1998), le premier modèle d'apprentissage en profondeur publié par A. Krizhevsky et al. (2012) a attiré l'attention du grand public comme celle des spécialistes de l'image en obtenant un score dépassant celui du meilleur modèle déjà existant avec une précision de 26,2%.
Ce fameux modèle, appelé "AlexNet", est ce qu'on peut aujourd'hui considérer comme une architecture simple avec cinq filtres de convolution consécutifs, des couches de pool maximal et trois couches entièrement connectées.
Depuis, plusieurs architectures ont été proposées pour les différentes tâches de vision par ordinateur dans le but d'améliorer les performances.

2.2 PointNet :

Motivé par les performances apportées par l'utilisation des réseaux de neurones dans le traitement d'image depuis 2012 et le manque de travaux se

basant sur les nuages de points en entrée, PointNet est une nouvelle approche qui consiste à utiliser un réseau de neurones et des données sous forme de nuages de points en entrée pour exécuter différentes tâches du domaine de traitement de l'image.
Sa principale caractéristique est la robustesse du réseau face aux données non-ordonnées en entrée en apprenant à résumer la forme à l'aide d'un ensemble de points clairs et d'une seule fonction symétrique : le max pooling.
Il s'agit d'un "framework" unifié pour diverses tâches : classification, segmentation(Part Segmentation) et segmentation sémantique.
De ce fait, son architecture est un peu compliquée :

1. Pour la tâche de classification, on a un réseau qui prend en entrée n points, leur applique les différentes transformations et agrège par la suite les différents features en utilisant le max pooling. Le résultat est un score de classification pour les "m" classes.
2. Pour la segmentation, le réseau de classification est prolongé par d'autres MLP afin de construire le résultat final en agrégeant les global features retournés par le réseau de classification et les points features induites par les MLP ajoutés.

Les résultats empiriques de PointNet sont de l'ordre ou dépassent l'état de l'art.
Les travaux de Groueix sont basés sur l'utilisation de PointNet comme auto-encodeur pour la génération des vecteurs latents.

2.3 Les autres méthodes :

2.3.1 3D reconstruction from multiple images / Stéréovision :

Elle est de loin la méthode la plus populaire dans les logiciels de reconstruction 3D.
Il s'agit d'une technique qui permet d'obtenir une représentation 3D d'un objet ou d'une scène à partir d'un ensemble d'images 2D de l'objet/scène prises sous différents angles.
Basée sur le principe de triangulation : Les points visibles sur une image sont les projections des points réels qu'on peut situer sur des droites et la position dans l'espace réel 3D peut alors être obtenue par intersection de ces droites.
Cette technique pose différents problèmes :

1. La calibration qui consiste à connaître comment se projettent les points de l'objet/la scène sur l'image.

2. Le matching qui correspondant à la capacité de reconnaître et d'associer les points qui apparaissent sur plusieurs images.
3. L'opacité : les objets observés ne doivent pas posséder des surfaces semi-transparentes pour que à chaque (x,y) ne correspond qu'un seul point (x,y,z).
4. La cohérence photométrique : Idéalement , les objets doivent renvoyer la même intensité lumineuse dans toutes les directions.

2.3.2 Shape from focus (defocus) :

Technique basée sur l'utilisation de l'effet de flou de dé-focalisation pour estimer la distance ou la forme des objets.
L'idée est que si la profondeur du champ est finie, le flou dépend de la distance entre l'objet et la caméra.
Dans ce cas là, les propriétés du système optique d'acquisition des images doivent être connues (ouverture de l'objectif, profondeur de champ) pour pouvoir faire des estimations assez prises pour la reconstruction.

2.3.3 Shape from shading :

Une technique qui permet de reconstruire la forme 3D d'un objet à partir d'une seule image en utilisant la variation d'intensité dans l'image pour estimer la distance ou la forme des objets.
L'idée est que l'intensité de la lumière réfléchie dépend de son incidence : la réflexion de la lumière R est le cosinus de l'angle entre la vecteur lumière L(x) et le vecteur normal n(x) à la surface [6] :

$$R = cos(L, n) = \frac{L}{|L|}.\frac{n}{|n|}$$

avec L et n qui dépendent de (x1,x2) les coordonnées 2D du point x.
Contraintes : Les conditions d'éclairage doivent être connues et il est démontré que la solution peut ne pas être unique.

2.3.4 Reconstruction par space carving :

On reconstruit la forme 3D en éliminant des voxels qui ne se projettent pas sur la silhouette de l'objet.

Pour améliorer le rendement, il est possible d'associer aux voxels des propriétés comme la couleur.
L'inconvénient principal est que la qualité de la forme 3D reconstituée est liée directement au pas de discrétisation.

3 Prise en main des données :

3.1 Outils utilisés :

Pour manipuler les images, on se sert de CNN, les CNN (Convolutional Neural Network) sont des réseaux de neurones spécialement conçus pour travailler sur des images, ils utilisent des convolutions au lieu de multiplications matricielles classiques.
On a choisi le framework de deep learning PyTorch plutôt que TensorFlow ou Keras.
Parmi les différents CNN existants, on a choisi VGG (développé par Visual Geometry Group) pour ses performances et sa simplicité. Via PyTorch, on a accès à différentes versions de **réseaux VGG pré-entraînés sur ImageNet** pour des tâches de classifications.

3.1.1 Les réseaux de neurones convolutifs :

"Les réseaux de neurones convolutifs sont simplement des réseaux de neurones qui utilisent la convolution dans à la place de la multiplication matricielle classique au sein d'au moins une de leurs couches." [4]
Cette définition assez concise est la plus claire qu'on puisse donner pour un CNN.
On distingue les CNN entre eux par leurs architectures (couches, ...), leurs niveaux de complexité (le nombre de neurones, présence ou non de boucles de rétroaction dans le réseau), par le type des neurones (leurs fonctions d'activation) et enfin par l'objectif visé : apprentissage supervisé ou non-supervisé, optimisation, ...

3.1.2 PyTorch :

Il s'agit d'un Framework Python open source développé par Facebook et conçu pour le Machine Learning.
Il permet d'effectuer des calculs tensoriels nécessaires à l'apprentissage de notre réseau de neurones mais également de calculer des gradients pour appliquer facilement des algorithmes d'optimisation par descente de gradient.

3.1.3 VGG -Visual Geometry Group- :

Il s'agit d'un modèle de réseau de neurones pré-entraîné composé de 16 à 19 couches convolutifs avec des filtres ou noyaux de convolution de petite taille généralement $3 * 3$, avec une architecture assez uniforme.
Les premières couches résument l'entrée en un plus petit vecteur, c'est la partie « feature extraction ». Les couches suivantes réalisent l'étape de classification à proprement parler.
Il est actuellement le choix le plus populaire dans la communauté pour l'extraction des features à partir des images d'une part pour la simplicité de son architecture et d'autre part pour le fait qu'on a accès à différentes versions de réseaux VGG pré-entraînés sur ImageNet pour des tâches de classifications.

3.1.4 OpenGl :

Pour la reconstruction des faces et l'affichage du résultat, on s'est servi du package PyOpenGL qui contient l'API OpenGL.

3.2 Utilisation de VGG :

3.2.1 Visualisation tSNE sur VGG :

La partie classification de VGG ne nous intéressant pas, nous nous servons du réseau comme extracteur de caractéristique en stoppant l'exécution du réseau de neurone à une couche intermédiaire (on récupère un vecteur latent).
VGG prend en entrée des images RGB 224*224, la dimension de l'entrée est donc $224 * 224 * 3 = 150528$. Le vecteur latent récupéré est de dimension 25088.
Afin de visualiser les caractéristiques extraites par VGG, on utilise l'algorithme T-SNE. T-SNE (pour 'T-distributed Stochastic Neighbor Embedding') est un algorithme servant à visualiser des données de grande dimension. Il prend en entrée des points dans un espace de grande dimension et les ramène dans un espace à 2 ou 3 dimensions tout en tentant de préserver les distances entre les points.
La dimension des vecteurs latents étant trop élevée, nous commençons par réduire le nombre de dimensions par PCA (Principal Component Analysis) afin de réduire le bruit (comme demandé dans la documentation de sklearn.tSNE).

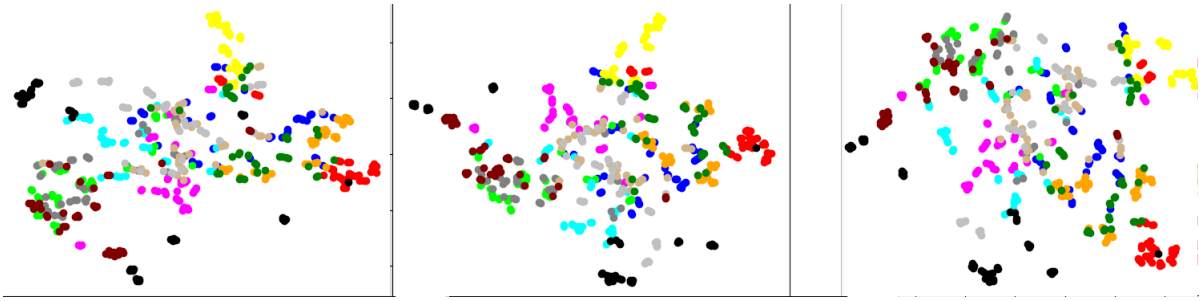


FIGURE 1 – Résultat du tSNE avec les 13 classes (pour 3 différents learning rate)

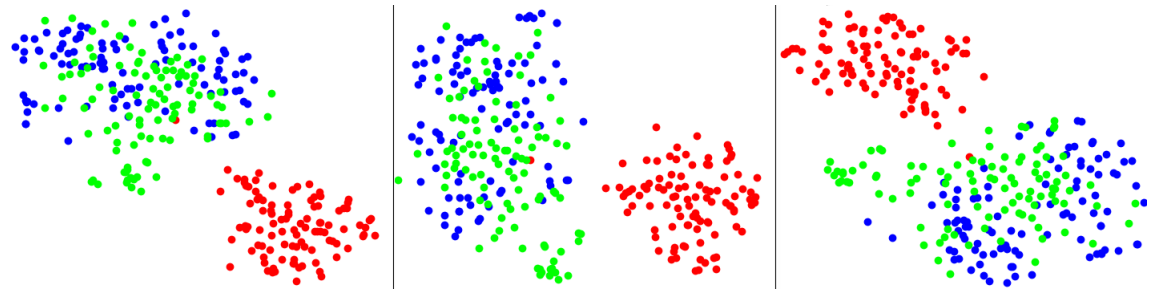


FIGURE 2 – Exhibition de deux classes confondues et une distincte (pour 3 différents learning rate) rouge : 'avions' – bleu : 'meubles' – vert : 'enceintes'

Les différentes classes d'objet se regroupent bien en **cluster**, cela démontre la capacité du réseau de neurone trouver des **caractéristiques propres à chaque classe**.

3.2.2 Classifieur :

En se basant sur la figure précédente, nous avons entraîné deux réseaux simples prenant en entrée les vecteurs latents et classifiant en deux classes. Les performances obtenues sont alors :

	train	test
meuble – avion	100%	100%
meuble – enceintes	78%	66%

Les performances sont en accord avec notre intuition (et la visualisation précédente) : un meuble et une enceinte sont plus difficiles à distinguer qu'un meuble et un avion.

3.3 Auto encodeur de nuages de points 3D :

Un auto-encodeur est un réseau de neurones utilisé pour l'apprentissage de manière non-supervisée de représentation des données en plus faible dimension, il est constitué d'un encodeur et d'un décodeur.
L'encodeur est formé d'un ensemble de couches qui traitent les données en entrée dans pour construire de nouvelles représentations que les couches du décodeur vont recevoir et traiter dans le but d'essayer de reconstruire la représentation des données en entrée.
Les différences entre l'objet initial et sa version encodée puis décodée permettent de mesurer une erreur et d'appliquer l'algorithme de back-propagation.

3.3.1 Architecture :

Nous avons entraîné un auto encodeur de nuages de points 3D. L'auto encodeur implémenté est un réseau de neurones défini par :

- Une entrée "3N" : un ensemble de n points dans $[-1; 1]^3$
- Un encodeur : 3N -> FC 512 -> ReLU -> FC 128 -> ReLU -> FC a (on récupère un vecteur latent de taille a)
- Un décodeur : a -> FC 128 -> ReLU -> FC 512 -> ReLU -> FC 3N -> tanh (une sortie de même dimension que l'entrée)

3.3.2 Données d'apprentissage :

Pour entraîner le réseau, on a généré nos propres nuages de points. Certains nuages sont constitués de points tirés selon une distribution gaussienne autour du centre du nuage, d'autres sont des sphères.
Afin de complexifier les nuages, on a aussi créé des nuages contenant plusieurs gaussiennes ou sphères.
La taille de l'entrée du réseau étant fixe, tous nos nuages contiennent exactement le même nombre de point. Le nombre de points est fixé arbitrairement à la main (en général entre 50 et 360) de telle sorte que les formes soient bien reconnaissables à vue d'œil.
Pour chaque point du nuage initial, l'auto encodeur génère un unique point correspondant. On peut donc définir une fonction de coût définie comme la somme des écarts entre chaque couple de point. En élevant les écarts au carré, on obtient le coût des moindres carrés.
La fonction de coût utilisée pour entraîner le réseau est inspirée de la distance de Chamfer (cf partie suivante). Cette version simplifiée correspond au

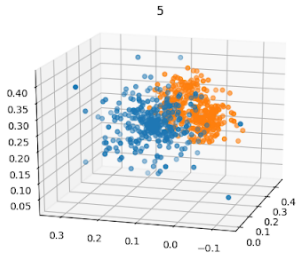
coût des moindres carrés additionné à la plus petite distance entre un point du nuage original et un point du nuage reconstruit. Par la suite, nous avons remarqué que la distance minimale devient rapidement négligeable comparée aux moindres carrés.
Pour entraîner le réseau on a utilisé 5 types de nuages différents. Un nuage de point est un ensemble de points échantillonnés aléatoirement sur :

1. une gaussienne
2. deux gaussiennes (avec exactement autant de point sur chacune des deux gaussiennes, et au total autant que dans le type précédent)
3. trois gaussiennes (de même)
4. une sphère
5. deux sphères (de même)

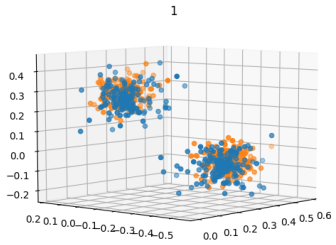
Environ 10 nuages de chaque type. Chaque nuage n'est généré qu'une fois : une fois généré, ses points sont constants pour toutes les itérations de l'apprentissage.
De l'ordre de 60 points par nuages.
Un vecteur latent de taille environ 10.

3.3.3 Résultats :

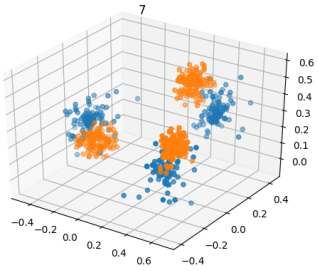
- La valeur du coût est bien représentative de ce que l'on voit. Sur la figure 3 (b) , les nuages sont confondus et on a un coût de 3,5 alors que sur la figure 3 (c) les nuages sont légèrement séparés et le coût est de 16,2.
- En utilisant un vecteur latent de taille 10, le réseau de neurones a parfois du mal à reconstruire les nuages qu'il a appris.
- Plus la taille du vecteur latent est grande, meilleurs sont les performances. Le réseau de neurones a toujours de meilleurs performances sur l'ensemble d'apprentissage (cf. Figure 4).
- Le réseau de neurones peut toujours réduire la loss en train mais cela devient du **sur-apprentissage** au bout de 200 epochs environ (cf. Figure 5).



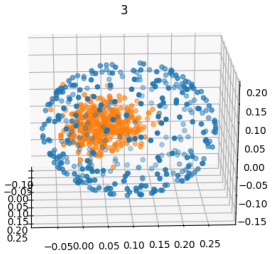
(a) 1 Gaussienne



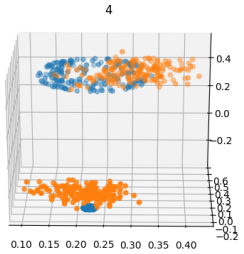
(b) 2 Gaussiennes



(c) 3 Gaussiennes



(d) 1 sphère



(e) 2 sphères

FIGURE 3 – Résultat de la reconstruction : nuages de point initiaux en bleu et nuages reconstruits en orange

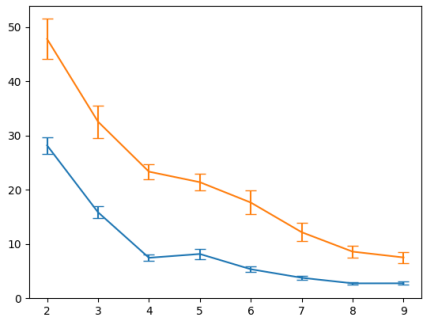


FIGURE 4 – Erreur en fonction de la taille du vecteur latent : train en bleu, test en orange avec des nuages contenant une ou deux gaussiennes (barres d'erreur à deux écarts-types sur une cross validation à 3 fold) 60 nuages, 288 points par nuage

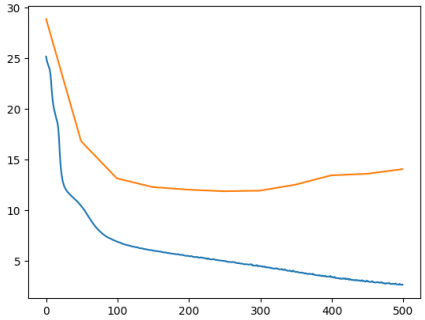


FIGURE 5 – Loss train et test en fonction du nombre d'epochs (60nuages, 50 pts, vecteur latent de taille 2)

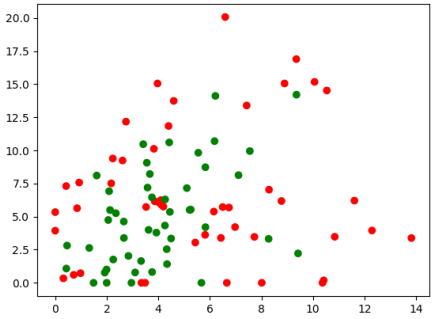


FIGURE 6 – Les points verts sont des 1-sphère et les points rouges des 2-sphères

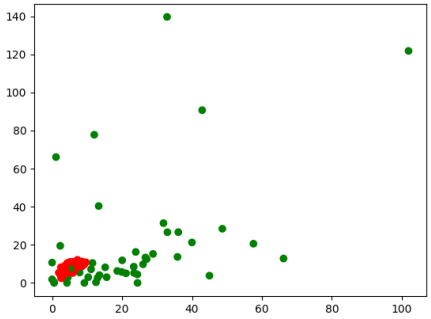


FIGURE 7 – Les gaussiennes sont toutes regroupées au même endroit alors que les sphères ne le sont pas

— En utilisant un vecteur latent de taille 2, on peut représenter de façon exacte les vecteurs latents sans devoir appliquer une technique de réduction de dimension comme tSNE ou PCA.

— Les points **ne sont pas répartis en 2 clusters comme on l'aurait souhaité**. On a donc recommencé l'expérience avec des sphères contre des gaussiennes.
— Avec un vecteur latent de taille 2, **la séparation en cluster n'est pas aussi nette** que dans la visualisation des vecteurs latents de VGG (cf. Figure 7).

4 La reconstruction 3D :

4.1 La méthode de Groueix :

4.1.1 Architecture :

Le réseau de neurones conçu par Groueix est défini de la manière suivante :
- Il prend en entrée une représentation d'une image 2D (Groueix utilise le vecteur latent de ResNet).
- Un MLP est constitué de 4 couches linéaires de taille 512 – 256 – 128 – 3 totalement connectées séparées par des ReLu puis un tanh afin de générer des points 3D dont les coordonnées sont entre -1 et 1.
- Pour reconstruire un objet morceau par morceau, on entraîne plusieurs MLP. Afin de générer plusieurs points par MLP, on ajoute en entrée du réseau les coordonnées d'un point échantillonné dans un carré unitaire. Le caractère continu de la reconstruction fait que les points reconstruits par un MLP forment une surface couvrant une petite zone de l'objet de la supervision.

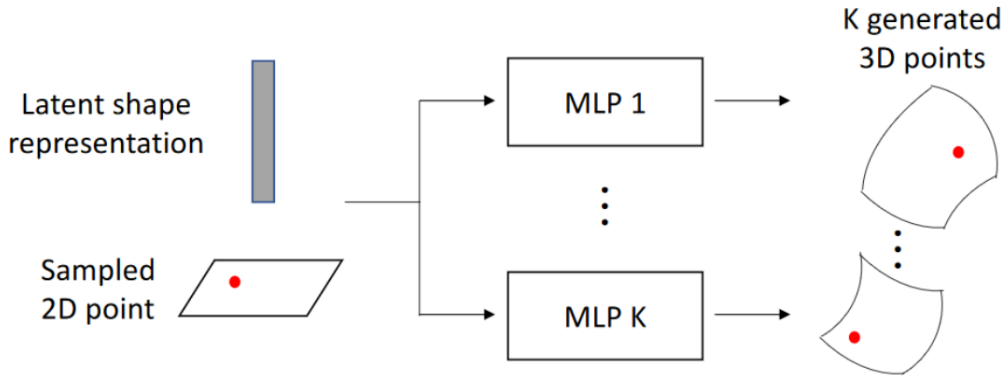


FIGURE 8 – Reconstruction des nuages (source : article de Groueix)

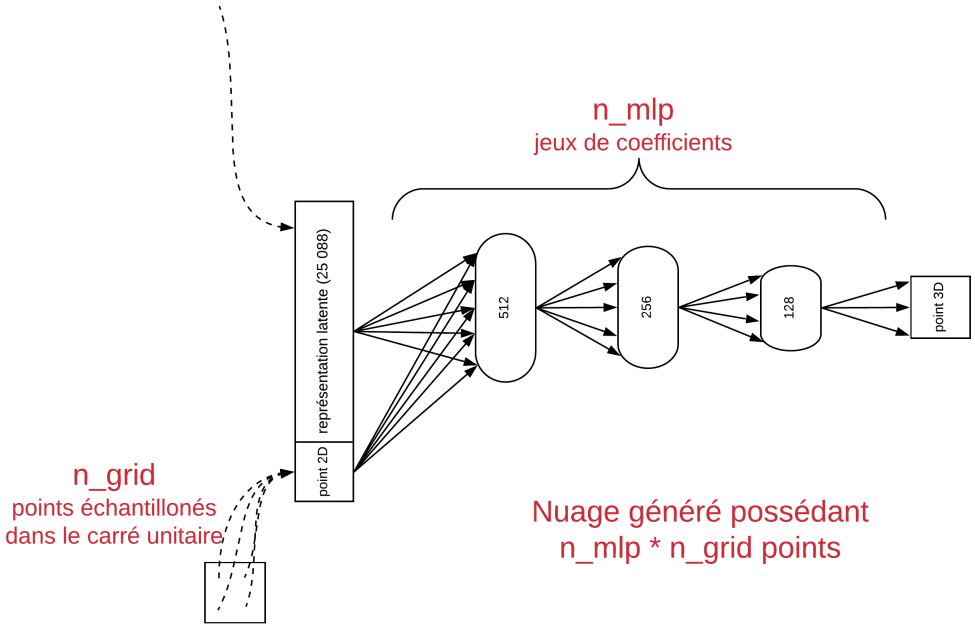


FIGURE 9 – Architecture du réseau de neurones

Afin de ne pas sur-apprendre les nuages de supervision, Groueix les ré-échantillonne à chaque epoch. Il rééchantillonne aussi les points du carré unitaire.

4.1.2 La fonction de coût :

La fonction de coût utilisée est la distance de Chamfer, elle est définie par :

$$CD(Y, S^*) = \sum min_{s \in S^*} \|\mathbf{y} - \mathbf{s}\|^2 + \sum min_{y \in Y} \|\mathbf{y} - \mathbf{s}\|^2$$

Le premier terme de la loss minimise pour chaque point de S*, la distance au point généré le plus proche, cela implique que les points de S* doivent bien être atteints par la forme générée. Le deuxième terme de la loss minimise pour chaque point généré, la distance au point de S* le plus proche, cela implique que les points générés doivent bien atteindre un point de S*, la reconstruction 'ne doit pas dépasser' la forme originale.
Une interprétation visuelle de l'effet du deuxième terme est qu'il fait se **déplacer les MLP vers la zone de S* la plus proche**. Avec une résolution

de sampling moyenne (6x6), **le deuxième terme empêche les MLP d'atteindre des zones utiles** de la figure. Il **bloque la convergence dans des optimums locaux**.
Sans le deuxième terme, si aucun des points d'un MLP n'est le plus proche d'un des points de S*, le MLP n'intervient dans aucune dérivée, il ne va donc pas bouger et le **MLP** va être « **oublié** » par le réseau de neurone. De plus, générer des **points loin de l'objectif** est visuellement une erreur mais, sans le deuxième terme, cela ne serait **pas pénalisé**.

4.2 Notre implémentation de la méthode de Groueix :

4.2.1 Architecture :

Nous nous servons de la représentation latente des images obtenue par VGG. L'entrée de notre réseau est donc un vecteur de taille $25088 + 2 \approx 25.10^3$.
Le premier problème de performances auquel nous faisons face est que la première couche du réseau possède donc $25090 * 512 \approx 13.10^6$ coefficients. De plus, on utilise plusieurs réseaux, Groueix en utilise 25.
Avec ce nombre on aurait environ 3.10^8 coefficients. Avec des coefficients stockés sur 64 bits, cela prend 2,6 Go de RAM. Les coefficients **passent heureusement en RAM** et laissent suffisamment de place pour stocker les images et les nuages de points (le nombre de coefficients dans les autres couches est négligeable).

4.2.2 La passe forward :

Afin d'obtenir le vecteur latent à partir d'une image, il faut la faire passer à travers les 30 couches du réseau VGG c'est très coûteux. En précaculant les représentations des images, on sauve une grande partie du temps de calcul.
Naïvement, générer un nuage de n points nécessite de **lire n fois** les 3.10^8 poids, cela est très coûteux. La première couche étant un simple calcul linéaire et les vecteurs en entrée possédant beaucoup de valeurs en commun, il est possible d'améliorer considérablement les performances.
Pour un MLP, on applique une seule fois la première couche $25088 * 512$. On duplique le vecteur obtenu (qui est de taille 512 seulement) autant de fois que l'on échantillonne de points. On fait passer chaque point échantillonné dans une couche $2 * 512$. Il suffit alors ensuite d'ajouter les deux vecteurs de taille

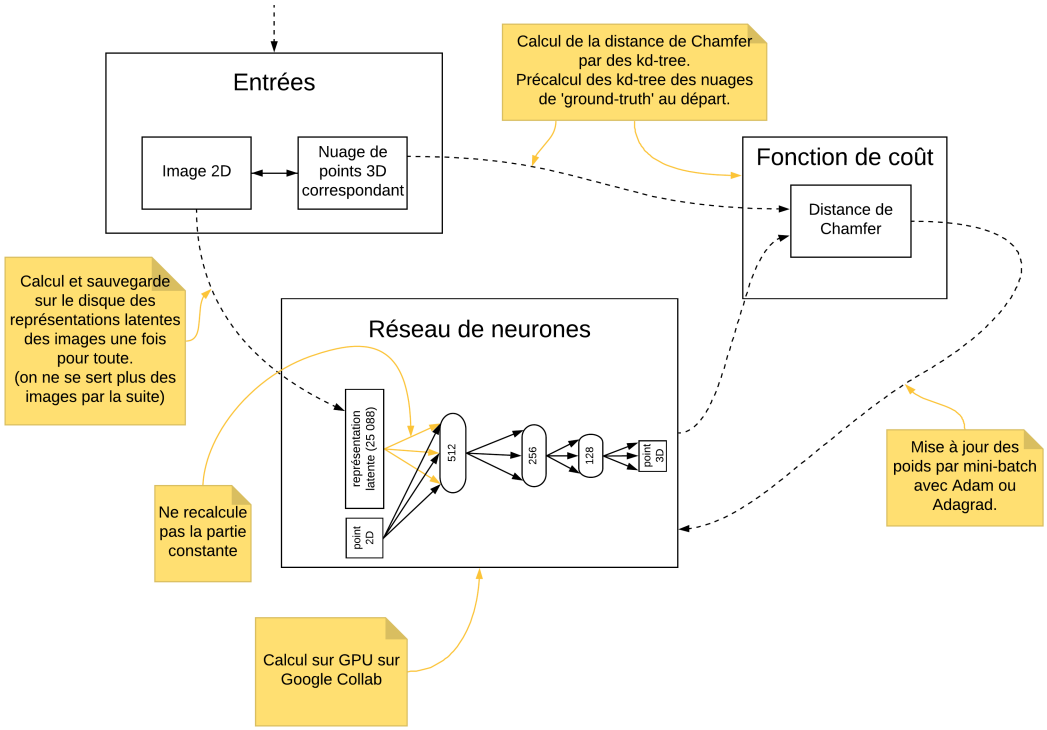


FIGURE 10 – Notre implémentation

512 pour chacun des points. En notant nmlp le nombre de MLP utilisés, la complexité est alors :

$$nmlp * 3.10^8 + n * 1024 * nmlp * 3.10^8$$

On se ramène donc à **nmlp lectures** uniquement. Il faudrait une résolution d'échantillonnage de l'ordre de 105 pour que le deuxième terme soit significatif, mais en pratique la résolution ne dépasse pas 100 points par MLP.
Notons que grâce à autograd, la passe backward est aussi améliorée de la même manière. De plus pour la passe backward, nous implémentons une descente de gradient mini-batch de telle sorte que le nombre de passes backward est négligeable par rapport au nombre de passes forward.

4.2.3 Calcul de la loss et optimisation :

Notons n le nombre de points générés et m le nombre de points des nuages de ground truth. La complexité de l'algorithme naïf consistant à calculer la matrice des distances entre chacun des points générés et de supervision pour ensuite trouver le minimum est $O(n * m)$.
Une première proposition a été de segmenter l'espace 3D en cases, on cherche le point le plus proche en parcourant les cases par rayon croissant. Une implémentation peut être trouvé dans le code source.
Les performances sont expérimentalement meilleures qu'avec l'algorithme naïf.
Une meilleure approche a été d'utiliser des kd-tree. On construit un kd-tree en $O(nlog(n))$, par la suite étant donné des coordonnées, on peut trouver le plus proche voisin en moyenne en temps logarithmique. Au final on obtient une complexité en $O(nlog(n) + nlog(m) + mlog(n))$, avec n et m du même ordre de grandeur, on obtient une complexité en $O(nlog(n))$.
Afin de ne pas sur-apprendre les nuages de supervision, Groueix les rééchantillonne à chaque epoch.
Notre implémentation revient à **rééchantillonner les nuages de supervision pour le premier terme de la loss** ('est-ce que le nuage est couvert ?') *mais pas pour le deuxième* ('est-ce qu'on dépasse ?').
Cela semble être une meilleur idée que celle de Groueix car on rééchantillonne ce qu'il faut atteindre, mais on ne pénalise pas les points reconstruits correspondant à un point non échantillonné à cette itération.
Pour ne pas avoir à recalculer les kd-tree de supervision à chaque itération, on calcule initialement les kd-tree des nuages de supervision sans sous-échantillonnage. Pour le calcul du premier terme de la loss, on itère simplement sur un sous-échantillon de la liste des points, pour le deuxième, on itère sur la liste des points générés et on cherche leur plus proche voisin dans le kd-tree entier.
Pour les réseaux précédents, nous utilisions l'algorithme de descente de gradient '**Adam**'. Cependant, la loss mesurée en apprentissage pour ce réseau n'arrive plus à décroître au bout d'une dizaine d'itération. Adam ne mettant pas à disposition un **facteur de décroissance** de taux d'apprentissage, nous avons opté pour l'algorithme '**Adagrad**'.

4.2.4 GPU, Google Collab et sauvegarde des poids :

Afin d'accélérer les temps de calcul, on exécute le code sur carte graphique sur les serveur de Google (cf code source). Cependant, le temps de calcul n'est que **4 fois meilleur environ**.
Après entraînement d'un réseau, on peut le sauvegarder facilement sur le disque afin de ne pas avoir à le ré-entraîner. Cela permet par ailleurs de télécharger un réseau de neurones entraîné sur le cloud.

5 Démarche et Résultats :

On a commencé par entraîner notre réseau sur une seule catégorie d'objet. On a alors obtenu la courbe suivante :

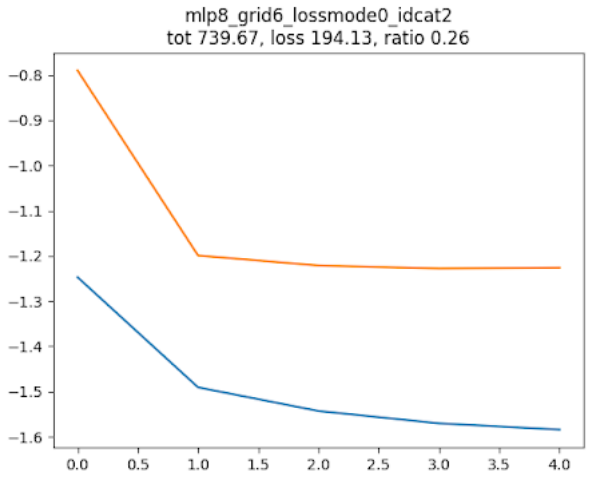


FIGURE 11 – Erreur en train (bleu) et test (orange) en fonction de l'epoch sur 50 nuages représentant des lampes

La loss en test diminue au départ car les points se répartissent dans l'espace mais ne converge pas par la suite. En réitérant l'expérience sur des formes plus simples comme des meubles, on obtient le même résultat : **l'erreur sur les nuages d'apprentissage converge mais sur le jeu de test, l'erreur ne converge pas et on ne peut pas reconnaître les objets reconstruits.** Nous avons alors étudié les convergences visuellement, voir le répertoire 'animations' :

— "0" : Les patchs sont facilement reconnaissables à l'œil nu. Ils se répartissent relativement bien sur la zone à couvrir.

— "1" : Si les patchs sont peu nombreux (ex 6) et trop gros (ex 10x10) ils restent coincés dans des optimums locaux à cause du deuxième terme de la loss qui leur impose de ne pas dépasser de la zone à reconstruire. Avec plus de patchs (ex 10), le problème est moins prononcé.

— "2" : En mettant un poids nul au deuxième terme de la loss, les patchs se répartissent mieux sur la zone à couvrir mais certains patchs peuvent dépasser de la zone à couvrir ou être inutiles initialement ils n'apparaissent alors dans aucune dérivée et sont inutilisés. On a alors tenté de changer la loss après plusieurs epochs en modifiant le poids du deuxième terme.

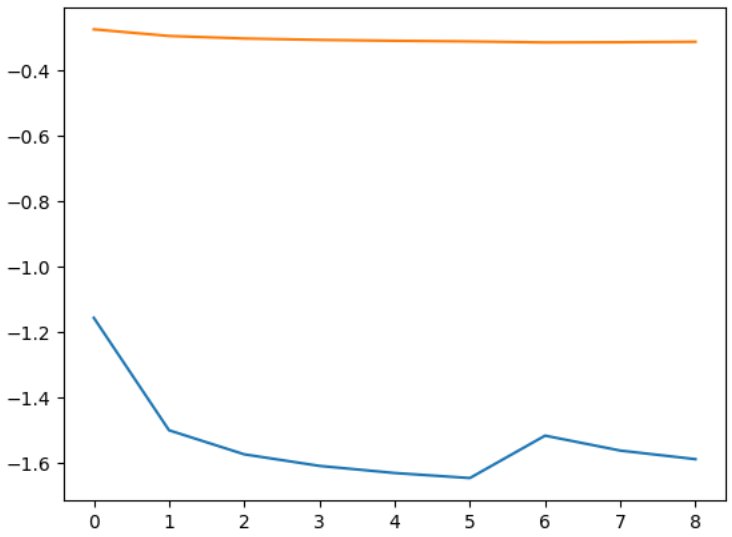


FIGURE 12 – Dans une tentative de faire converger l'erreur en test, on change la loss utilisée en train en cours de route (à l'epoch 6) en augmentant le poids du second terme dans la loss. Cela n'a eu aucun effet sur le test.

— "3" : Par la suite, plutôt que d'initialiser tous les patchs vers le centre de la figure (initialisation aléatoire par défaut) on utilise une initialisation de 'Xavier' afin de répartir les patchs aléatoirement dans la figure.

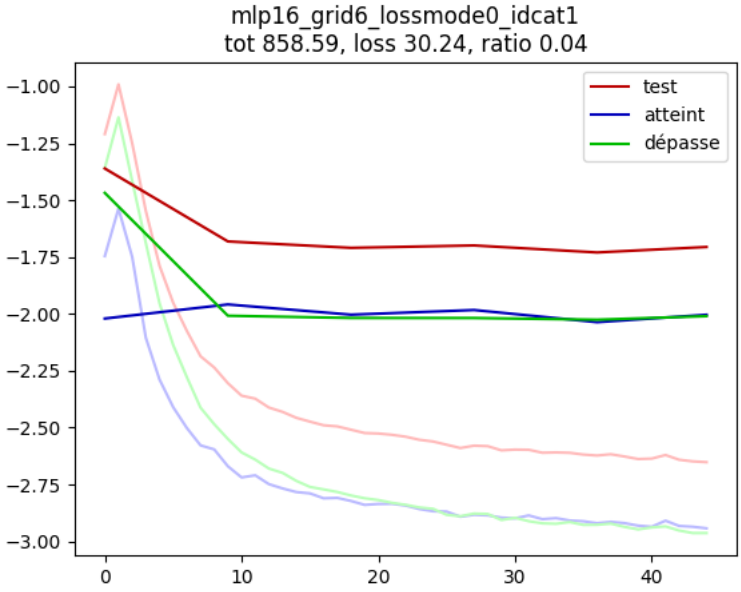


FIGURE 13 – Les courbes indiquent les loss en train (clair) et test(foncé) en détail : 1er terme de la loss (bleu) – 2ème terme (vert) – total (rouge)

— "4" : Un problème que nous avons rencontré est qu'en recommençant l'entraînement plusieurs fois, le réseau peut converger en train vers différentes reconstructions de coût proche mais seulement l'une reconstruction est visuellement bonne. Le problème surgit par exemple sur les meubles dès qu'il y a plus d'un exemple en apprentissage (voir animations).

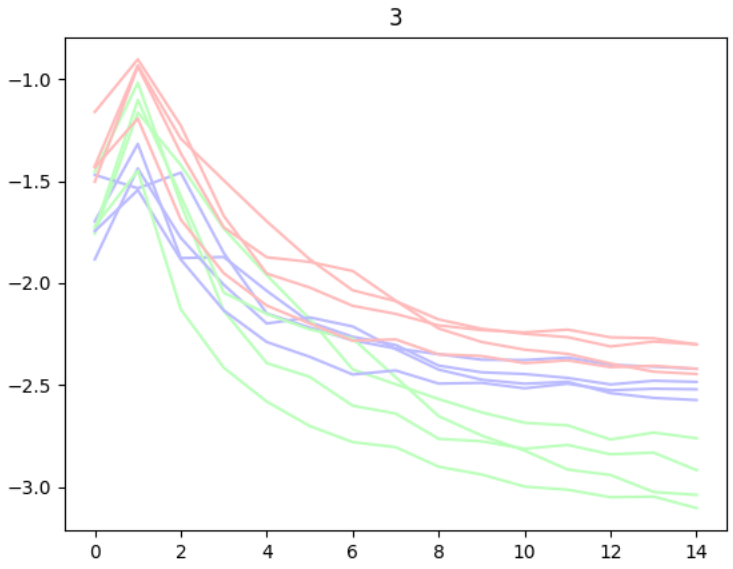


FIGURE 14 – Erreurs détaillées lors d'une convergence avec 4 nuages menant à une reconstruction visuellement incorrecte. Les erreurs sont toutes similaires.

Plusieurs tentatives ont été réalisées afin de résoudre les problèmes : nous avons changé chacun des paramètres de mini-batch, rééchantillonnage des nuages, rééchantillonnage de la grille, taille des nuages, loss définie par une moyenne ou une somme, poids des termes de la loss. Cependant, ces tentatives n'ont pas abouti.

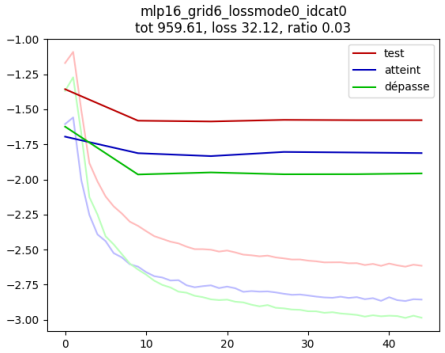


FIGURE 15 – Nuages générés de taille 16 * 6*6 = 576 et des nuages de supervision de taille 300, le premier terme de la loss est légèrement plus fort que le deuxième.

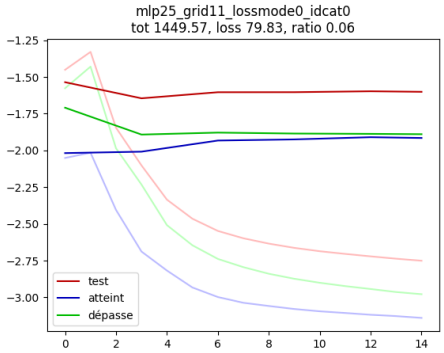


FIGURE 16 – Nuages générés de taille 25 * 11*11 = 3025 et une supervision de taille 3000, c'est le deuxième terme qui est le plus grand.

Les erreurs en test ne convergent pas dans les deux cas.

6 Reconstruction des surfaces :

Notre implémentation rend un nuage de points en 3D qu'on a exploité pour reconstruire les faces de l'objet.
A partir de la grille qui nous a permis de générer les nuages, on peut peut facilement déduire quels sommets il faut relier afin de former les faces de l'objet 3D. Les surfaces viennent se superposer vu les distances insignifiantes entre leurs bords ce qui nous a permis de ne pas avoir à les connecter.
Une fois, les surfaces construites, on a utilisé PyOpenGL pour afficher l'objet et sauvegarder l'image correspondante.

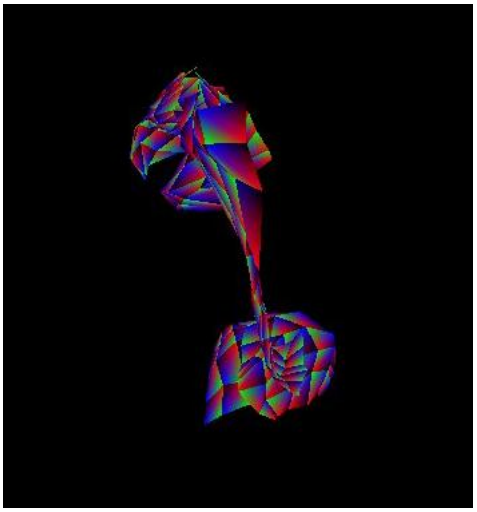


FIGURE 17 – Résulat de la reconstruction d'une lampe

7 Conclusion :

Durant ce projet, nous avons étudié un article de recherche en détail et découvert différentes techniques de reconstruction 3D.
Via PyTorch, nous avons appris à nous servir de réseaux complexes pré-entraînés et à définir et entraîner nos propres réseaux. Par la même occasion, nous avons défini différentes loss et nous sommes servis de plusieurs algorithmes de descente de gradient.
Nous avons résolu des problèmes de complexité calculatoire de manière logicielle via l'utilisation d'algorithmes et de structures de données appropriés et de manière matérielle en exécutant notre code sur carte graphique sur les serveurs de Google.
Pour la reconstruction, plusieurs tentatives ont été réalisées afin de résoudre les problèmes de mauvaise convergence du train et de non convergence du test, mais malheureusement, ces essais se sont révélés infructueux. Le fonctionnement en 'boîte noire' des réseaux de neurones rend la résolution des problèmes auxquels nous faisons face plus difficile.

Bibliographie :

[1] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "AtlasNet : A Papier-Mâché Approach to Learning 3D Surface Generation," arXiv :1802.05384 [cs], Feb. 2018.

[2] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet : Deep Learning on Point Sets for 3D Classification and Segmentation," arXiv :1612.00593 [cs], Dec. 2016.

[3]S. Ruder, "An overview of gradient descent optimization algorithms," arXiv :1609.04747 [cs], Sep. 2016.

[4] "Deep Learning." [Online]. Available : https ://www.deeplearningbook.org/. [Accessed : 15-Mar-2019].

[5] M. Siudak and P. Rokita, "A Survey of Passive 3D Reconstruction Methods on the Basis of More than One Image," Machine Graphics and Vision, vol. Vol. 23, No. 3/4, pp. 57–117, 2014.

[6] Perception.inrialpes.fr. (2019). [online] Available at : http ://perception.inrialpes.fr/Publications/2006/PF06a/chapter-prados-faugeras.pdf

[7] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum, "Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling," arXiv :1610.07584 [cs], Oct. 2016.

[8] "Open3D – A Modern Library for 3D Data Processing." [Online]. Available : http ://www.open3d.org/. [Accessed : 15-Mar-2019].