Rapport Technique LRE (Laboratoire de Recherche de l'EPITA) Hugo Saison

# Exploration des Obstacles à la Segmentation et à la Classification 3D : Un Parcours d'Apprentissage

*Abstract :*

Ce rapport traitait initialement de la détection, de l'identification et du suivi d'objets dans les nuages de points temporels à l'aide de l'intelligence artificielle. Cependant, de nombreux défis ont entravé notre progression, nous obligeant à réviser entièrement notre approche. Nous explorons les difficultés techniques rencontrées, évaluons les technologies existantes et expliquons pourquoi notre projet n'a pas avancé comme prévu. Enfin, nous proposons des orientations futures pour aborder ces problèmes et améliorer les méthodes actuelles.

*Résumé :*

Ce rapport explore les défis de la classification et de la segmentation des nuages de points 3D dynamiques, avec un focus sur la ré-implémentation de PointNet en utilisant Google Colab et Pytorch. La recherche de problématiques innovantes a été compliquée notamment par la domination de grands acteurs comme Google et Tesla. Cependant, ce travail a permis de mieux comprendre les applications actuelles et d'identifier des niches potentielles. Les pistes futures incluent l'étude des biais d'entraînement et l'optimisation des modèles pour une meilleure robustesse. Ce rapport n'apportera pas d'avancée technique au domaine.

## Introduction

### *Contexte général du travail*

La classification et la segmentation des nuages de points dynamiques en 3D sont essentielles pour des applications telles que la conduite autonome et l'aide à la vision. Les technologies d'acquisition comme les capteurs LiDAR et les caméras de profondeur sont largement utilisées, chacune ayant ses avantages et ses limitations. Le traitement de ces données implique un pipeline complexe [1] comprenant plusieurs étapes essentielles pour obtenir des résultats précis et fiables. Contrairement aux images 2D, les nuages de points en 3D présentent des défis uniques : la captation de données volumineuses, la taille des données, l'instabilité de l'espace, le désordre inhérent des nuages et le coût de calcul élevé. Ces obstacles rendent les techniques de traitement 2D inadaptées pour les nuages de points en 3D. Ce rapport détaille les

difficultés rencontrées dans ce domaine, évalue les technologies actuelles et propose des perspectives pour surmonter ces défis.

### *Description des problèmes rencontrés*

Ce domaine pose divers défis, notamment l'occlusion partielle des objets lors de la captation, la gestion des grandes quantités de données nécessaires, leur nature désordonnée et le coût élevé en termes de calcul. En tant qu'étudiant en première année d'école d'ingénieur en informatique, j'ai dû surmonter plusieurs obstacles majeurs. D'abord, la difficulté à identifier une problématique novatrice face aux efforts massifs de R&D des géants technologiques. Ensuite, l'accès limité à des données réelles m'a contraint à utiliser des ensembles de données disponibles en ligne, souvent insuffisants ou non publiés malgré les annonces initiales.

### *Contributions spécifiques*

Mon parcours a été marqué par une exploration intensive et un apprentissage des bases du domaine plutôt que par des contributions spécifiques. Mon intention initiale était d'analyser les biais induits par l'entraînement des modèles de conduite autonome sur des environnements diversifiés comme les routes indiennes et américaines, mais les défis techniques ont limité mes résultats à des essais exploratoires.

### *Plan du rapport*

Ce rapport est structuré comme suit :

**État de l'art :** Exploration détaillée de la classification et de la segmentation, avec une analyse des technologies existantes et de leurs avantages.
**Recherche de problématique :** Discussion sur les technologies actuelles, leur couverture des besoins existants, mes critères de sélection et les idées de problématique envisagées.
**Retour aux sources :** Approfondissement sur PointNet comme référence du domaine, avec une explication détaillée de son implémentation.
**Conclusion :** Réflexion sur les défis d'entrée dans le domaine et propositions pour les orientations futures.

### *Remerciements*

Je tiens à remercier particulièrement Thibault Lejemble et le reste de l’équipe qui m’a suivi à distance pendant ces 5 mois : Nicolas Boutry, Laurent Beaudouin et Loïca

Avanthey. Merci pour vos conseils, votre soutien et votre relecture, merci de m'avoir intéressé à ce domaine passionnant.

Je voudrais aussi remercier mes camarades de promotion dont la curiosité m'a poussé à en découvrir toujours plus, tout en trouvant des simplifications pour présenter le domaine.

Finalement, je dois remercier le Discord de PointNet from Pytorch pour leur conseils et travail de vulgarisation.

## Contexte et État de l'art

### *1. Cadre de travail*

La classification et la segmentation de nuages de points 3D dynamiques sont cruciales pour diverses applications, notamment la conduite autonome et l'aide à la vision pour les personnes malvoyantes. Le processus d'acquisition de données en 3D est complexe et nécessite l'utilisation de capteurs sophistiqués. Les capteurs LIDAR sont les plus précis et rapides, mais également les plus coûteux, tandis que les caméras de profondeur sont plus accessibles mais beaucoup plus lentes et moins précises. Pour les applications nécessitant une grande précision, comme la conduite autonome, le LIDAR est préféré.

Le traitement des nuages de points 3D suit généralement un pipeline structuré [1] comprenant plusieurs étapes essentielles : le pré-traitement des données, la segmentation, la classification, l'anticipation de trajectoire et le retour visuel. Chacune de ces étapes utilise des technologies et des méthodologies spécifiques, nous allons détailler ci-dessous quelques-unes.

### *2. Technologies et Méthodologies*

#### Acquisition des données

**LIDAR** : Fournit des données extrêmement précises et rapides, idéales pour la conduite autonome. Cependant, il est coûteux.

**Caméras de profondeur** : Moins chères et plus accessibles, mais leur précision et rapidité sont inférieures à celles des capteurs LIDAR. La Kinect utilise ces caméras de profondeur, elle rend accessible la création de nuage de points.

### Filtration [2]

**Statistique** : Une approche statistique de la répartition des points dans le nuage de point.

**Voisinage** : En comparant le point à ses voisins, il est décidé si le point est gardé ou non

**Projection** : Par une série de projection, on ajuste la position des points du nuage de points

### Segmentation [3] et Classification

**PointNet et PointNet++** [1]: PointNet est une méthode de segmentation qui traite directement les nuages de points en 3D, tandis que PointNet++ améliore cette approche en utilisant une hiérarchie de sous-ensembles de points pour capturer les structures locales. Ces méthodes sont puissantes pour traiter les données désordonnées inhérentes aux nuages de points 3D.

**Transformers [5]** : Utilisés pour capturer les relations globales entre les points, ces modèles sont efficaces pour des tâches nécessitant une compréhension globale de la scène.

**ASIS (Associatively Segmenting Instances and Semantics) [3] :** Une méthode qui segmente à la fois les instances et les sémantiques des objets dans un nuage de points, améliorant ainsi la précision des tâches de segmentation et de classification.

### Tracking

**Kalman Filter [6]** : Une méthode mathématique sans IA, utilisée pour le suivi des objets en temps réel. Elle est efficace mais peut être limitée par des hypothèses linéaires. Le filtre de Kalman est donc la méthode que les méthodes par IA veulent dépasser.

**MPBTrack [7]** et **Prediction Association [1]** : Utilisent des techniques d'IA pour associer les observations passées et prédire les mouvements futurs, améliorant la précision du suivi dans des scénarios complexes.

## *3. Problèmes rencontrés et solutions apportées*

### Taille et nature désordonnée des données

L'un des principaux défis de la manipulation des nuages de points 3D est leur nature désordonnée et la taille massive des données. Les nuages de points ne peuvent pas être traités de manière séquentielle comme les images 2D, ce qui nécessite le développement de nouvelles méthodes de traitement. Par ailleurs, la taille des fichiers rend difficile leur stockage et leur partage. Par exemple, une minute et quarante secondes de données capturées peut représenter 4,1 Go [11], ce qui pose des problèmes de stockage significatifs.

### Problèmes de stockage et de versions de packages

En raison des limitations de stockage sur ma machine personnelle, j'ai dû utiliser Google Collab, un service de notebook en ligne, pour héberger et traiter les données. Cette solution m'a permis d'utiliser le stockage en nuage pour gérer les grands ensembles de données et d'éviter les conflits de version de packages sur ma machine locale. Malgré cela, les problèmes de compatibilité de packages n'ont pas été entièrement résolus, même après avoir essayé plusieurs plateformes et gestionnaires de packages.

### *4. Recherche de problématique*

Ma recherche initiale portait sur l'analyse des biais induits par l'entraînement des modèles de conduite autonome dans différents environnements. Il est crucial de comprendre ces biais pour pouvoir adapter les datasets et améliorer la performance des modèles dans des situations variées. Cependant, trouver des datasets adaptés a été un défi en soi, car je ne pouvais pas créer mes propres données. Après une recherche approfondie, j'ai identifié plusieurs datasets pertinents disponibles en ligne, tels que NL-Drive, DHB, Kitti, SemanticPoss, NuScenes, SuperCaustic, Waymo, et ShapeNet. Ils sont tous soit dédié à la conduite autonome, soit à la détection d'objets du quotidien [12].

### Recherche de problématique et leurs problèmes

### *1. Contexte et recherche de problématique*

Dans le domaine de l'intelligence artificielle, trouver une problématique novatrice commence souvent par l'acquisition de données pertinentes. Les datasets publics disponibles sont majoritairement orientés vers deux domaines principaux : la conduite autonome et l'aide à la vision au quotidien pour les personnes malvoyantes. Ces domaines sont largement dominés par de grands acteurs technologiques tels que Tesla, Google, Microsoft, Apple, et Disney. Ces entreprises possèdent non seulement des ressources considérables mais aussi des initiatives avancées et protégées par le secret professionel dans ces domaines, rendant la recherche de nouvelles problématiques particulièrement difficile pour les chercheurs indépendants et les étudiants.

Malgré ces défis, cette recherche intensive a permis de comprendre en profondeur les diverses applications possibles et les solutions actuellement mises en œuvre par ces grandes entreprises. Cette exploration a été cruciale pour identifier les lacunes et les opportunités potentielles dans le domaine.

### *2. Exploration de problématiques envisageables*

Plusieurs problématiques ont été envisagées, mais beaucoup étaient déjà en cours de développement ou avaient été résolues par les grandes entreprises. Voici quelques exemples de ces problématiques :

**Association de Scan 3D et de caméra 2D pour une meilleure précision** : Cette approche est largement utilisée, formant la base des systèmes de vision avancée.
**Ajout de la fonctionnalité de tracking d'objets dans les Apple Vision Pro**: Cette technologie est en cours de développement par une collaboration entre Disney et Apple, avec des applications similaires déjà disponibles sur iPad et iPhone.
**Étude de la persistance de l'identification malgré des interactions inter-objets** : Par exemple, vérifier si les étiquettes d'objets restent correctes lorsqu'une main tient un mug. La méthode ASIS (Attention-based Supervised Image Segmentation) [3] aborde déjà cette problématique.
**Adaptation des voitures autonomes au passage de la conduite à droite à la conduite à gauche** : Tesla utilise une méthode d'entraînement sur les deux types de routes et ajuste le modèle en fonction de la position GPS.

Cependant, une nouvelle idée a émergé à partir de cette dernière problématique : **l'analyse des biais introduits par l'entraînement dans différents environnements routiers**.

### *3. Problématique sélectionnée : Analyse des biais dans la conduite autonome*

L'idée principale était de déterminer si une voiture autonome, entraînée dans un environnement spécifique, pouvait conduire efficacement dans des environnements différents. Cette étude visait à explorer les biais introduits par l'entraînement et leur impact sur la performance du modèle dans divers contextes routiers.

Pour ce faire, le plan consistait à :

Entraîner un modèle préexistant sur un dataset représentant une situation routière spécifique.
Tester ce modèle sur une situation complètement différente pour évaluer sa performance et identifier les biais.

Trois cas d'étude avaient été identifiés :

I. Une route allemande basique. [11]
II. Une route anglaise pour une conduite à gauche basique.

III. Une route indienne caractérisée par un flux routier chaotique et des véhicules spécifiques. [10]

L'objectif final était d'estimer la capacité de conduite autonome dans chaque situation et de quantifier les biais induits par l'entraînement. [8]

### *4. Problèmes rencontrés et limitations*

Malgré des efforts soutenus, plusieurs obstacles ont empêché la réalisation complète de cette étude :

**Disponibilité des datasets** : Le dataset indien annoncé publiquement était introuvable, ce qui a fortement limité la capacité à explorer tous les cas d'étude envisagés. [10]
**Taille des données** : Le volume des données nécessaires pour réentraîner les modèles était immense. Par exemple, une seule vidéo de moins de deux minutes pesait déjà 4 gigabits [11], remplissant rapidement la capacité de stockage disponible.
**Compatibilité des packages** : Les évolutions des packages nécessaires pour configurer l'environnement de travail ont entraîné des incompatibilités, notamment avec OpenCV, rendant impossible l'utilisation des modèles préentraînés.

Pour contourner les problèmes de stockage, le recours à Google Colab a été envisagé, permettant l'utilisation du cloud pour stocker les datasets. Cependant, malgré diverses tentatives sur différentes plateformes et systèmes d'exploitation, les incompatibilités des packages ont persisté, empêchant toute progression significative.

Ces défis ont mis en lumière les complexités techniques et logistiques de la recherche en conduite autonome et ont finalement conduit à un retour aux fondamentaux pour réimplémenter les modèles à partir de zéro.

## Réimplémentation de PointNet dans Google Colab utilisant uniquement Pytorch

### *1. Introduction à PointNet [4] [9]*

PointNet est un réseau de neurones révolutionnaire conçu pour traiter des nuages de points 3D. Contrairement aux approches traditionnelles qui utilisent des grilles régulières ou des voxels pour représenter des données 3D, PointNet traite directement les nuages de points bruts. Cette méthode permet une meilleure préservation des informations géométriques et topologiques des objets 3D, rendant PointNet

particulièrement efficace pour les tâches de segmentation et de classification dans des applications variées telles que la conduite autonome et la modélisation 3D.

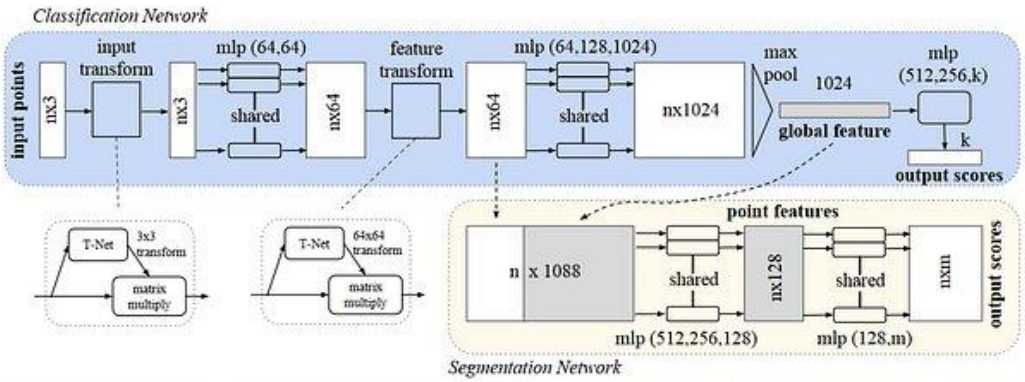



Figure 2. **PointNet Architecture.** The classification network takes $n$ points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for $k$ classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

### *2. Choix de l'environnement : Google Colab et Pytorch*

Pour surmonter les problèmes de stockage et de compatibilité des packages rencontrés précédemment, Google Colab a été choisi comme environnement de développement. Google Colab offre une infrastructure cloud qui permet l'utilisation de puissants GPU et un stockage étendu via Google Drive. De plus, il facilite l'installation et la gestion des packages nécessaires, réduisant ainsi les problèmes de compatibilité. De plus, nous avons préféré minimiser les packages nécessaires.

Pytorch a été sélectionné pour cette réimplémentation en raison de sa flexibilité et de son adoption croissante dans la communauté de recherche en deep learning. Pytorch permet une manipulation facile des tenseurs, une création intuitive des modèles et offre des performances optimisées sur GPU.

Enfin, il existe une communauté active autour de l'implémentation de PointNet à partir de Pytorch. Etant donné le temps disponible, pourvoir utiliser les ressources disponibles était d'une grande aide.

### *3. Étapes de ré-implémentation*

#### a. Préparation de l'environnement

La première étape a consisté à configurer l'environnement Google Colab pour utiliser Pytorch. Cela impliquait l'installation des packages nécessaires et la configuration de l'accès aux données stockées sur Google Drive.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
```

### b. T-nets

La classe T-nets permet à PointNet de traiter directement les nuages de points et non des voxels. [9]

```python
from typing_extensions import Required
class Tnet(nn.Module):
  ''' learns a transformation matrix with a specified dimension that will be apllied to any point in the pointcloud '''
  def __init__(self, dim, num_points = 2500):
    super(Tnet, self).__init__()

    #dimension of the transform matrix
    self.dim  = dim


    self.conv1 = nn.Conv1d(dim, 64, kernel_size = 1)  #premier doute ici sur quel sera sont utilite : Shared MLP implemented as 1D
    self.conv2 = nn.Conv1d(64, 128, kernel_size = 1)
    self.conv3 = nn.Conv1d(128, 1024, kernel_size = 1) #3conv 1D pour reqliser une conv 3D ?

    self.linear1 = nn.Linear(1024, 512) # que represente ces puissances de 2 ???
    self.linear2 = nn.Linear(512, 256)  # a quoi ca servira ? : permet de scale down ou up la dimension de la matrice
    self.linear3 = nn.Linear(256, dim**2)

    self.bn1 = nn.BatchNorm1d(64)
    self.bn2 = nn.BatchNorm1d(128)
    self.bn3 = nn.BatchNorm1d(1024) #ordre important ? pk 1024 directement pour redescendre ?
    self.bn4 = nn.BatchNorm1d(512)
    self.bn5 = nn.BatchNorm1d(256)

    self.max_pool = nn.MaxPool1d(kernel_size = num_points)

  def forward(self, x):
    bs = x.shape[0]

    # pass through shared MLP layers (conv1D)

    x = self.bn1(F.relu(self.conv1(x)))
    x = self.bn2(F.relu(self.conv2(x)))
    x = self.bn3(F.relu(self.conv3(x)))

    x  = self.max_pool(x).view(bs, -1)

    # pass through MLP
    x = self.bn4(F.relu(self.linear1(x)))
    x = self.bn5(F.relu(self.linear2(x)))
    x = self.linear3(x)

    #initialize id matrix :
    iden = torch.eye(self.dim,  requires_grad=True).repeat(bs, 1 ,1)
    if x.is_cuda :
      iden = iden.cuda()
    x = x.view(-1, self.dim, self.dim) + iden

    return x
```

### c. Le cœur de PointNet

Passons à l'implémentation de PointNet comme décrit sur l'image. On réutilise ici les T-nets et les Max Pooling Layers. [9]

```python
class PointNetBackbone(nn.Module):
    def __init__(self, num_points=2500, num_global_feats=1024, local_feat=True):
        ''' Initializers:
                num_points - number of points in point cloud
                num_global_feats - number of Global Features for the main
                                   Max Pooling layer
                local_feat - if True, forward() returns the concatenation
                             of the local and global features
            '''
        super(PointNetBackbone, self).__init__()
        self.max_pool = nn.MaxPool1d(kernel_size=num_points, return_indices=True)
        self.num_points = num_points
        self.num_global_feats = num_global_feats
        self.local_feat = local_feat
        # Spatial Transformer Networks (T-nets)
        self.tnet1 = Tnet(dim=3, num_points=num_points)
        self.tnet2 = Tnet(dim=64, num_points=num_points)
        # shared MLP 1
        self.conv1 = nn.Conv1d(3, 64, kernel_size=1)
        self.conv2 = nn.Conv1d(64, 64, kernel_size=1)
        # shared MLP 2
        self.conv3 = nn.Conv1d(64, 64, kernel_size=1)
        self.conv4 = nn.Conv1d(64, 128, kernel_size=1)
        self.conv5 = nn.Conv1d(128, self.num_global_feats, kernel_size=1)
        # batch norms for both shared MLPs
        self.bn1 = nn.BatchNorm1d(64)
        self.bn2 = nn.BatchNorm1d(64)
        self.bn3 = nn.BatchNorm1d(64)
        self.bn4 = nn.BatchNorm1d(128)
        self.bn5 = nn.BatchNorm1d(self.num_global_feats)

    def forward(self, x):

        bs = x.shape[0]
        # pass through first Tnet to get transform matrix
        A_input = self.tnet1(x)
        x = torch.bmm(x.transpose(2, 1), A_input).transpose(2, 1)
        # pass through first shared MLP
        x = self.bn1(F.relu(self.conv1(x)))
        x = self.bn2(F.relu(self.conv2(x)))
        # get feature transform
        A_feat = self.tnet2(x)
        # perform second transformation across each (64 dim) feature in the batch
        x = torch.bmm(x.transpose(2, 1), A_feat).transpose(2, 1)
        local_features = x.clone()
        # pass through second MLP
        x = self.bn3(F.relu(self.conv3(x)))
        x = self.bn4(F.relu(self.conv4(x)))
        x = self.bn5(F.relu(self.conv5(x)))
        global_features, critical_indexes = self.max_pool(x)
        global_features = global_features.view(bs, -1)
        critical_indexes = critical_indexes.view(bs, -1)
        if self.local_feat:
            features = torch.cat((local_features,
                                  global_features.unsqueeze(-1).repeat(1, 1, self.num_points)),
                                  dim=1)

            return features, critical_indexes, A_feat
        else:
            return global_features, critical_indexes, A_feat
```

### d. Classification

Nous pouvons finir la bande bleue de l'image avec la classe de Classification. Les outils sont les mêmes que pour les classes précédentes.

```python
class PointNetClassHead(nn.Module):
    ''' Classification Head '''
    def __init__(self, num_points=2500, num_global_feats=1024, k=2):
        super(PointNetClassHead, self).__init__()

        # get the backbone (only need global features for classification)
        self.backbone = PointNetBackbone(num_points, num_global_feats, local_feat=False)

        # MLP for classification
        self.linear1 = nn.Linear(num_global_feats, 512)
        self.linear2 = nn.Linear(512, 256)
        self.linear3 = nn.Linear(256, k)

        # batchnorm for the first 2 linear layers
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)

        # The paper states that batch norm was only added to the layer
        # before the classication layer, but another version adds dropout
        # to the first 2 layers
        self.dropout = nn.Dropout(p=0.3)


    def forward(self, x):
        # get global features
        x, crit_idxs, A_feat = self.backbone(x)

        x = self.bn1(F.relu(self.linear1(x)))
        x = self.bn2(F.relu(self.linear2(x)))
        x = self.dropout(x)
        x = self.linear3(x)

        # return logits
        return x, crit_idxs, A_feat
```

### e. Segmentation

Enfin, il reste la classe de segmentation reprenant de la Classification mais aussi de la sortie du cœur de PointNet.

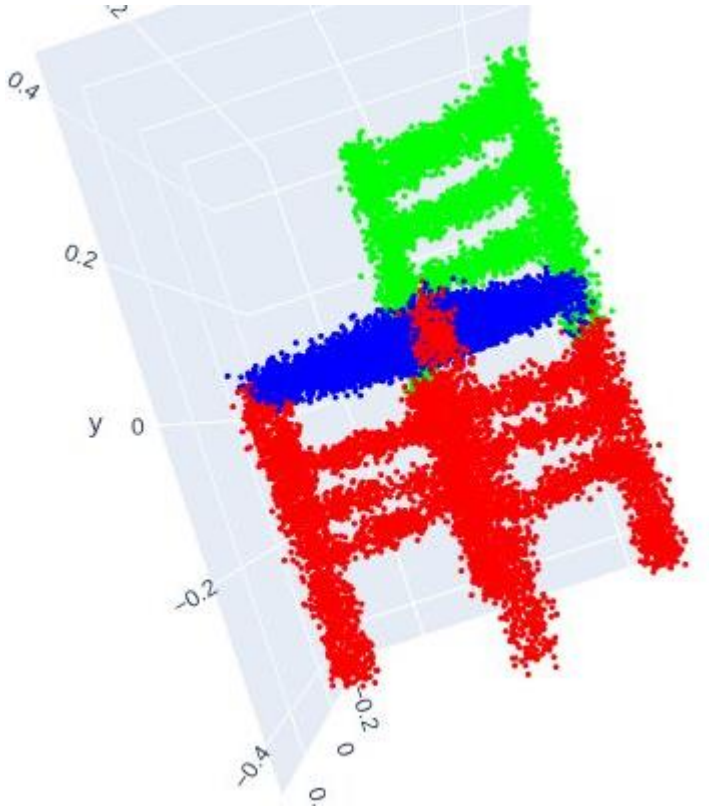
```python
class PointNetSegHead(nn.Module):
    ''' Segmentation Head '''
    def __init__(self, num_points=2500, num_global_feats=1024, m=2):
        super(PointNetSegHead, self).__init__()

        self.num_points = num_points
        self.m = m

        # get the backbone
        self.backbone = PointNetBackbone(num_points, num_global_feats, local_feat=True)

        # shared MLP
        num_features = num_global_feats + 64 # local and global features
        self.conv1 = nn.Conv1d(num_features, 512, kernel_size=1)
        self.conv2 = nn.Conv1d(512, 256, kernel_size=1)
        self.conv3 = nn.Conv1d(256, 128, kernel_size=1)
        self.conv4 = nn.Conv1d(128, m, kernel_size=1)

        # batch norms for shared MLP
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.bn3 = nn.BatchNorm1d(128)


    def forward(self, x):

        # get combined features
        x, crit_idxs, A_feat = self.backbone(x)

        # pass through shared MLP
        x = self.bn1(F.relu(self.conv1(x)))
        x = self.bn2(F.relu(self.conv2(x)))
        x = self.bn3(F.relu(self.conv3(x)))
        x = self.conv4(x)

        x = x.transpose(2, 1)

        return x, crit_idxs, A_feat
```

*4. Résultats et discussion*

La ré-implémentation de PointNet a permis de surmonter plusieurs défis techniques et logistiques rencontrés initialement. L'utilisation de Google Colab a fourni une solution viable pour les limitations de stockage et de calcul, tandis que Pytorch a offert une flexibilité et une efficacité dans le développement du modèle.

Les résultats obtenus ont démontré la capacité de PointNet à segmenter et classer efficacement des nuages de points 3D, confirmant ainsi la validité de l'approche. Cependant, l'évaluation complète des performances et des biais reste à réaliser

Instance Segmentation d'une chaise issue du dataset ShapeNet [13] par la ré-implémentation de PointNet.




Discussion :

Le travail réalisé reprend des ressources déjà disponibles en ligne, malgré la volonté d'apporter notre pierre au domaine, les problèmes multiples que nous avons rencontrés ne nous aurons pas permis en 5 mois de travail de réaliser une avancée dans le domaine de la classification et la segmentation de nuage de point 3D dynamique. Nous avons comme perspective de permettre à PointNet de prendre une suite de nuages de points pour analyser dynamiquement un objet. Pour ce qui est des nouveautés, l'étude des biais lors de l'entrainement attire toujours notre intérêt mais

une refonte complète du protocole devra être réalisé si nous voulons continuer dans cette direction. Notre travail aura surtout mis en lumière la complexité que d'innover à partir de zéro dans ce domaine tant au niveau technique qu'au niveau pratique. Développer comme l'on fait PointNet une introduction simple au domaine serait d'une grande aide pour les personnes intéressées par le domaine.

## Conclusion :

Ce travail a exploré les défis et les solutions liés à la recherche de problématiques novatrices et à la ré-implémentation de modèles avancés en vision par ordinateur 3D, en particulier PointNet. La première partie a mis en lumière les difficultés rencontrées pour trouver des datasets et des problématiques originales dans un domaine dominé par des géants technologiques comme Google, Microsoft et Tesla. Malgré ces obstacles, cette recherche a permis de mieux comprendre les applications actuelles de la vision par ordinateur et d'identifier des niches potentielles pour des contributions futures.

La seconde partie s'est concentrée sur la réimplémentation de PointNet, un modèle de référence pour le traitement des nuages de points 3D, utilisant l'environnement Google Colab et la bibliothèque Pytorch. Cette ré-implémentation a permis de surmonter les limitations matérielles et de compatibilité, tout en démontrant la capacité de PointNet à effectuer des tâches de segmentation et de classification avec une grande précision. L'utilisation de Google Colab a également souligné l'importance des infrastructures cloud dans la recherche moderne en intelligence artificielle.

Les principaux intérêts de ce travail résident dans l'approfondissement des connaissances sur les méthodologies actuelles en vision par ordinateur 3D et l'acquisition de compétences pratiques dans la ré-implémentation de modèles complexes. La compréhension des défis liés aux données, aux environnements de développement et aux modèles eux-mêmes a été grandement améliorée. De plus, ce travail a mis en évidence l'importance de choisir les bons outils et plateformes pour surmonter les contraintes techniques.

Cependant, plusieurs pistes restent encore à explorer. L'étude des biais induits par l'entraînement sur des environnements routiers spécifiques et leur impact sur la conduite autonome nécessite une investigation plus approfondie. De plus, l'optimisation des modèles pour qu'ils soient plus robustes et généralisables dans des contextes variés représente un défi majeur. Enfin, l'exploration de nouvelles applications de la vision par ordinateur 3D, telles que l'aide à la navigation pour les personnes malvoyantes, offre des opportunités pour des recherches futures.

En conclusion, ce travail a jeté les bases pour des explorations plus approfondies et a ouvert de nouvelles perspectives dans le domaine de la vision par ordinateur 3D. Les

défis rencontrés et les solutions mises en œuvre serviront de guide pour des recherches ultérieures, visant à améliorer et à étendre les applications de cette technologie prometteuse.

## Références :


1. Segmentation, Classification and Tracking of Objects in Lidar Point Cloud Data Using Deep Learninbg (ROBIN BERNSTA LE, HJALMAR LIND)
2. A review of algorithms for filtering the 3D point cloud (Xian-Feng Han , et al.)
3. Associatively Segmenting Instances and Semantics in Point Clouds (Xinlong Wang, et al.)
4. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation (Charles R. Qi, et al.)
5. Point Transformer (Hengshuang Zhao, et al.)
6. An Introduction to the Kalman Filter (Greg Welch, et al.)
7. MBPTrack: Improving 3D Point Cloud Tracking with Memory Networks and Box Priors (Tian-Xing Xu, et al. )
8. Is It Safe to Drive? An Overview of Factors, Metrics, and Datasets for Driveability Assessment in Autonomous Driving (Junyao Guo, et al.)
9. Point Net from Scratch (Isaac Berrios)
10. IDD-3D: Indian Driving Dataset for 3D Unstructured Road Scenes (Shubham Dokania, et al.)
11. The KITTI Vision Benchmark Suite (Karlsruhe Institute of Technology, Toyota Technological Institute at Chicago)
12. https://paperswithcode.com/datasets?mod=lidar
13. ShapeNet: An Information-Rich 3D Model Repository (Angel X. Chang, et al.)