HAL
open science

Détection/reconnaissance d'objets urbains à partir de données 3D multicapteurs prises au niveau du sol, en continu

Younes Zegaoui

To cite this version:

Younes Zegaoui. Détection/reconnaissance d'objets urbains à partir de données 3D multicapteurs prises au niveau du sol, en continu. Electronique. Université Montpellier, 2021. Français. NNT : 2021MONTS070 . tel-03589031

HAL Id: tel-03589031

https://theses.hal.science/tel-03589031

Submitted on 25 Feb 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR DE L'UNIVERSITÉ DE MONTPELLIER

En informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

Détection/reconnaissance d'objets urbains à partir de données 3D multi-capteurs prises au niveau du sol en continu

Présentée par Younes ZEGAOUI
Le 12 octobre 2021

Sous la direction de Marc CHAUMONT
Philippe BORIANNE et Gérard SUBSOL

Devant le jury composé de

Géraldine MORIN, Pr, INP Toulouse/IRIT	Présidente/examinatrice
François GOULETTE, Pr, MINES ParisTECH	Rapporteur
Sébastien LEFEVRE, Pr, Univ.de Bretagne Sud/IRISA IUT Vannes	Rapporteur
Christophe FIORIO, Pr, Université de Montpellier	Examinateur
Gérard SUBSOL, CR, CNRS/LIRMM	Co-encadrant
Marc CHAUMONT, MCF HDR, Univ. de Nîmes/LIRMM	Directeur
Philippe BORIANNE, Chercheur-Ingénieur, AMAP	Invité Co-encadrant
Laurent DERUELLE, MCF, DRI – Berger-Levrault	Invité

UNIVERSITÉ DE MONTPELLIER

Table des matières

Table des figures

Liste des tableaux

Liste des symboles

$\boldsymbol{\mu}$	Centroïde associé à un *cluster* dans notre architecture de réseau. Chaque centroïde est défini par une partie géométrique $\mathbf{g} \in \mathbb{R}^3$ et une partie caractéristique $\mathbf{f} \in \mathbb{R}^d$
$\hat{\mathbf{B}}_{2D}$	Boîte modèle, c'est à partir des dimensions des boîtes modèles que sont définies les dimensions des boîtes prédites par les RPN
$\mathscr{K}(\boldsymbol{\mu})$	Région de points appartenant au nuage de points original S et associée au *centroïde* $\boldsymbol{\mu} \in S_l$
$\mathscr{L}$	Fonction de coût d'un réseau de neurones
C	Classe d'un objet, limitée par le nombre de classes possibles N_C
D	Fonction de distance définie entre 2 points $\mathbf{P}$ ou 2 *centroïdes* $\boldsymbol{\mu}$. Si aucune spécification sur la nature de cette fonction n'est donnée, il s'agit d'une distance euclidienne
I	Image, de dimension $H \times L$. $I_{x,y}$ représente la valeur du pixel de l'image I à la position (x, y)
O	Identifiant unique d'un objet. Deux objets différents peuvent avoir la même classe
O_{maj}	Identifiant relatif à l'objet majoritaire présent dans un *cluster*. C'est autour de cet objet que le réseau va essayer de prédire une boîte englobante
S	Nuage de points, équivalent à S_0, avec $S = \mathbb{R}^{3\times N}$ où N est le nombre de points
S_l	Nuage de points en sortie de la $l$ième couche de $cluster - convolution$. Les points $\boldsymbol{\mu} \in S_l$ correspondent à des *centroïdes* calculés par le réseau. On note $S_l \in \mathbb{R}^{d_l \times N_l}$, où N_l est le nombre de *centroïdes* et d_l la dimension de leurs parties caractéristiques f
$\mathbf{B}$	Boîte englobante, définie par un centre (x, y, z) et une dimension (hauteur h, largeur l, profondeur d) ainsi que la classe C de l'objet qu'elle contient

P	Point appartenant à un nuage de points, il s'agit d'un triplet (x, y, z)
R	Région d'un nuage de points : sous-ensemble de points de ce nuage
R_{2D}	Région d'une image, sous-ensemble d'une image compris dans le rectangle de centre (x, y) et de dimension (l, h)

Préambule : présentation de l'entreprise

Les travaux de recherche effectués lors de cette thèse sont issus d'une collaboration scientifique entre le laboratoire d'Informatique de Robotique et de Micro-électronique de Montpellier (LIRMM) et l'entreprise Berger-Levrault. Ceux-ci s'inscrivent dans le cadre du dispositif CIFRE mis en oeuvre par l'Agence Nationale de la Recherche et de la Technologie (ANRT).

Officiellement fondé en 1676, le groupe Berger-Levrault est initialement une imprimerie, une maison d'édition et une librairie. Celui-ci s'occupe principalement de la publication et de l'impression de documents officiels et administratifs de l'Etat français. A partir de 1821, l'entreprise croît fortement grâce aux commandes de documents imprimés par les pouvoirs publics. Cette croissance permet une modernisation de l'entreprise en 1850 grâce aux avancées technologiques et sociales. En 1981, le groupe prend le virage du numérique en passant de l'édition de documents à l'édition de logiciels informatiques. En 2013, le groupe Berger-Levrault s'ouvre à l'international grâce à une implantation au Canada, qui sera suivie d'une implantation en Espagne et au Maroc. Même si les clients principaux de Berger-Levrault sont les collectivités territoriales ainsi que les administrations publiques, son secteur d'activité est plus vaste, comportant notamment : le médical, le social, l'enseignement, les transports, la gestion d'entreprises publiques ou privées, la gestion d'équipements... Aujourd'hui, le groupe emploie plus de 1700 personnes, compte plus de 50 000 clients repartis autour du monde et 90% de son activité est consacrée au numérique.

Chapitre 1

Introduction

1.1 Contexte applicatif de la thèse

Parmi toutes les activités de Berger-Levrault, les travaux présentés dans ce manuscrit s'insèrent dans le cadre des problématiques rencontrées par les gestionnaires de villes.

Par gestion des objets des villes, nous désignons l'ensemble des actions ou pratiques qui ont pour but d'administrer de manière raisonnée et durable l'environnement urbain ainsi que de maîtriser les interactions entre les différents objets urbains. Dans la suite nous nous concentrons uniquement sur l'aspect objet urbain de la gestion des villes.

1.1.1 Gestion des objets urbains

Qu'est-ce qu'un objet urbain ?

Un objet urbain peut être défini par "un objet ou un dispositif public ou privé installé dans l'espace public et lié à une fonction ou à un service offert par la collectivité [1]". Cette définition permet de prendre en compte la diversité des objets urbains et d'englober une grande variété d'objets, tels que les arbres, les panneaux de signalisation ou encore les bancs dans les espaces verts.

Les deux acteurs de la gestion

Les actions de gestion des objets urbains peuvent être réparties en 2 catégories : celles effectuées par les gestionnaires de villes eux-mêmes, dans ce cas il s'agit d'action *métier* et celles effectuées par les fournisseurs d'outils, comme l'édition de logiciels par Berger-Levrault, dans ce cas il s'agit d'action de *gestion de données*.

1. Annie Boyer, Elizabeth Rojat-Lefebvre, *Aménager les espaces publics : le mobilier urbain*, Le Moniteur, 2001

Action métier : Les actions métier concernent différents aspects de la ville visant chacun un objectif, dont certains se recoupent :

- Actions économiques : l'objectif est de minimiser les coûts d'entretien des objets urbains. Parmi ces actions : le recensement des objets urbains tels que les arbres, la maîtrise du cycle de vie des différents équipements de la ville comme les panneaux et feux de signalisation...
- Actions risque : le but est d'assurer que les objets urbains, du fait de leur présence ou de leurs fonction, n'introduisent pas de nouveaux risques. Il s'agit de préserver l'intégrité des biens et personnes au sein de l'environnement urbain. Par exemple, éviter que la chute d'un arbre provoque des dommages aux individus circulant dans la ville ou à leurs biens.
- Actions résilience : l'objectif est d'assurer la continuité des services de la ville en cas de crise. Par exemple assurer une circulation routière minimale en cas d'intempéries ou bien alimenter en priorité les hôpitaux et feux de signalisation en cas de panne d'électricité, au détriment par exemple des lampadaires.
- Action traçabilité : l'objectif est de suivre précisément le cycle de vie des objets urbains. Pour chacun d'entre eux, il s'agit de recueillir les informations nécessaires au dépistage de situations problématiques qui peuvent être causées, soit par un imprévu dans le changement d'état d'un objet, comme par exemple un lampadaire hors service, soit par l'interaction entre les objets urbains, comme par exemple la croissance d'un arbre qui viendrait sectionner des câbles électriques.

Action gestion de données : Les actions gestion de données sont proposées par les produits à visée des gestionnaires de ville, comme par exemple *Atal II*, développé par Berger-Levrault, qui est une application donnant accès aux gestionnaires de villes à une base de données des objets urbains. La mise à disposition d'outils doit faciliter les actions des gestionnaires de ville ainsi que la centralisation des données recueillies par les gestionnaires de ville en vue de leur accessibilité.

Un des prolongements des actions gestions de données fournies par *Atal II* est le protocole de suivi d'arbres urbains proposé par l'application mobile *Treetech*, destinée à être intégrée à *Atal II*. Cette application est un outil de terrain développé dans le but d'uniformiser les observations d'arbres et censé équiper le personnel de la ville mandaté pour ces observations.

La limite principale de ce protocole est qu'il nécessite de dépêcher régulièrement des personnes sur le terrain afin qu'elles réalisent les observations pour chaque arbre. Or pour prendre l'exemple de l'agglomération de Montpellier, celle-ci compte : plus de 50 000 arbres, plus de 8000 panneaux de signalisation, 60 km de ligne de tramway (arrêt de tramway, caténaire) [2]. Un suivi régulier du même type que *TreeTech* ne serait alors pas adapté à une telle quantité

2. https://data.montpellier3m.fr/

d'objets, car cela nécessiterait des ressources humaines considérables, dont ne disposent pas les collectivités aujourd'hui.

1.1.2 Automatisation de la gestion des objets urbains

Un moyen de répondre aux impératifs soulevés par l'application *TreeTech*, en termes de moyen à disposition, est la mise en place d'outils qui permettent d'automatiser une partie des actions nécessaires à la gestion des objets urbains. Dans le cas des arbres urbains, cela se traduirait par un suivi de l'état des arbres qui ne nécessiterait pas de déplacer des personnes mandatées sur le terrain.

Nous distinguons deux courants de recherches spécifiques au problème de l'automatisation de la gestion des objets urbains, chacun proposant une approche différente :

- La première est d'installer des capteurs directement sur les objets urbains et d'opérer selon la méthodologie de l'Internet des objets (*Internet of things* ou *IoT*).
- La deuxième consiste à utiliser des capteurs à distance pour surveiller les objets urbains.

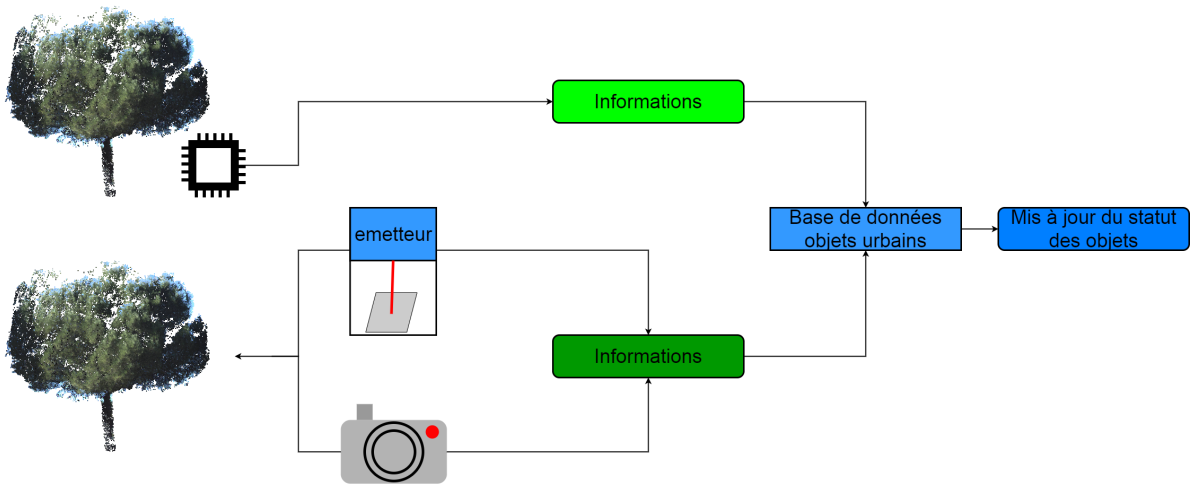

FIGURE 1 – Illustration du principe de relevés réguliers pour la gestion des objets urbains. En haut, l'approche classique des objets connectés, où un capteur et un processeur sont intégrés directement sur l'objet afin d'envoyer des informations sur le statut de l'objet à un serveur distant, en bas l'approche qui consiste à utiliser un capteur distant aux objets afin d'en extraire des informations.

a) Des objets urbains connectés

L'approche par objet connecté consiste à équiper chaque objet urbain de capteurs pour mesurer plusieurs des caractéristiques de l'objet et de son environnement telle la localisation, la température, la luminosité ou l'humidité [VD10]. Ces informations sont alors soit envoyées à une base de données afin de mettre à jour le statut de l'objet ou bien elles permettent de modifier le service rendu par l'objet grâce à l'installation d'actionneurs adaptés. C'est le cas par exemple de

certains lampadaires équipés de détecteur de mouvements qui s'allument uniquement lorsqu'une personne est à proximité et reste éteints le reste du temps [HSA18].

Puisque les objets peuvent en permanence être reliés à la base de données, il est possible de suivre leurs évolutions en temps réel et ainsi de planifier les différentes opérations de maintenance dans le but d'éviter des futures pannes [VD10].

La limite principale de cette approche est qu'elle nécessite une mise en place extrêmement lourde : il faut équiper chaque objet urbain de capteurs et/ou d'actionneurs alors que le nombre de ces objets est conséquent. De plus cette approche ne peut être appliquée que sur un sous-ensemble restreint d'objets urbains compatibles avec l'ajout de capteurs adapté, comme par exemple les lampadaires ou les bancs [HSA18].

b) Suivre les objets à distance : la télédétection

La télédétection consiste à utiliser des capteurs distants afin de récolter des informations sur les objets urbains puis à analyser ces informations. Cette approche a déjà été explorée dans le cadre du recensement automatique d'objets urbains spécifiques dans les travaux de recherches précédents menés conjointement par le LIRMM et le groupe Berger-Levrault. Ceux-ci proposent par exemple de se baser sur l'imagerie aérienne afin de localiser précisément les pierres tombales dans les cimetières [Pas16][PCSD15][PDB+16], les bouches d'égout [CENP+17] ou bien les arbres en ville [Pib18][PCS+18]. D'autres travaux proposent d'utiliser des types de capteurs différents, tels que l'imagerie *SAAR* pour la reconstruction 3D en milieu urbain [BJS19] ou l'imagerie par satellite pour la cartographie automatique de bidonvilles [SWTZ19].

Notre but est d'approfondir ces travaux pour, d'une part prendre en compte plusieurs catégories d'objets urbains et non une seule, et d'autre part localiser précisément ces objets urbains afin de mettre en lien les résultats de l'analyse des informations et la base de données d'objets urbains. Pour cela, les observations par images ne sont pas suffisantes car le géo-référencement y est limité. C'est pourquoi nous avons décidé d'explorer les possibilités offertes par la technologie *LiDAR*.

Ainsi une campagne d'acquisition de données à l'échelle d'une ville et utilisant un capteur *LiDAR* géo-référencé, permettrait alors de recenser automatiquement les objets urbains présents et ainsi mettre à jour, ou créer, les bases de données d'objets urbains. Cette application est illustrée dans la figure 1. De plus, des campagnes d'acquisition réalisées à intervalle de temps régulier, par exemple à partir de bennes à ordures équipées de capteurs *LiDAR*, permettraient le suivi des informations collectées aussi bien dans l'espace, grâce à l'utilisation de système d'information géographique, comme par exemple celui proposé par *Atal II*, que dans le temps grâce à une historisation des données. Ce suivi régulier permettrait alors un *monitoring* des objets urbains et ainsi dépister les anomalies présentes et le cas échéant alerter les gestionnaires.

1.2 Des données 3D à l'échelle d'une ville : le *LiDAR*

1.2.1 Principes du *LiDAR*

Utilisé pour la première fois en 1961, le *LiDAR*, pour *LIght Detection and Ranging*, est une technologie basée sur le principe de l'écho-localisation permettant d'évaluer des distances précises entre deux points. Contrairement au *radar* qui utilise des ondes radioélectriques, le *LiDAR* se base sur des lasers avec une longueur d'onde comprise le plus souvent entre *600 nm* et *1000 nm*.

De la mesure de distance...

Un dispositif *LiDAR* est composé de deux parties, un émetteur et un récepteur. L'émetteur est un laser impulsionnel et le récepteur est un photodétecteur couplée à une optique. Afin de mesurer la distance d entre le *LiDAR* et une surface A, l'émetteur envoie tout d'abord une impulsion laser, celle-ci va rebondir sur la surface A et une partie du faisceau lumineux va être captée par le récepteur. Le signal lumineux capté par le récepteur est ensuite filtré spectralement afin d'y retirer le bruit provenant d'autres signaux lumineux. L'intervalle de temps t entre l'émission du laser et sa réception est alors calculé, permettant d'en déduire la distance d au regard de la vitesse de la lumière c dans l'air :

$$d = \frac{c \times t}{2} \tag{1.1}$$

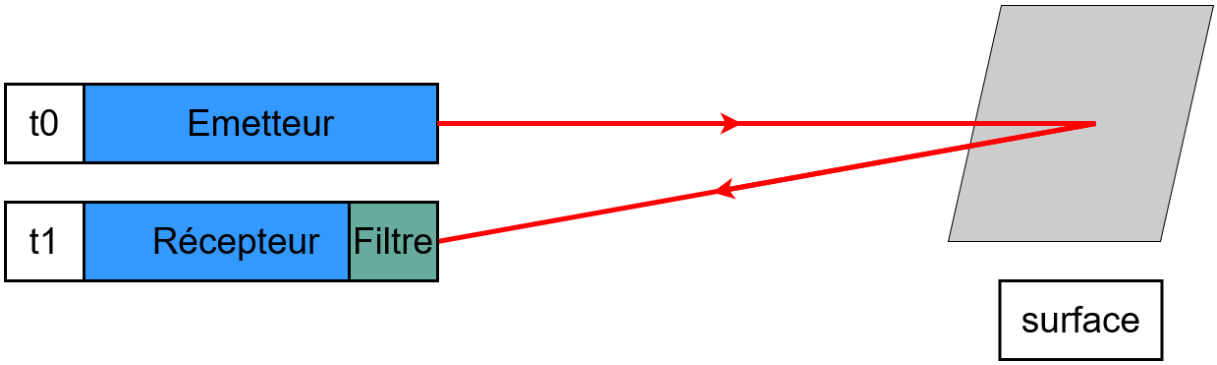

FIGURE 2 – Schéma du fonctionnement du calcul de distance grâce à l'utilisation d'un *LiDAR*. L'intervalle de temps t correspond à $t_1 - t_0$.

Ce procédé est illustré dans la figure 2. Cette méthodologie a des applications directes en télémétrie où elle permet d'évaluer ou de vérifier précisément des distances, par exemple en architecture. Une version modifiée du *LiDAR*, appelé *LiDAR* fibré, incorporant un modulateur accousto-optique, permet en plus de la distance de mesurer la vélocité d'une surface en mouvement, comme une voiture, ou bien d'identifier son mode de vibration, qui peut être analysé spectralement. Par exemple, pour évaluer l'état de santé de bâtiments ou de falaises rocheuses.

...à la numérisation de scènes

En plus du dispositif émetteur-récepteur, il est possible d'ajouter un système de miroir rotatif permettant de dévier l'angle d'émission et de réception du *LiDAR*. Ce type de *LiDAR* est alors appelé *LiDAR* par balayage en référence aux mouvements rotatifs des rayons lasers permis grâce au miroir.

Le miroir rotatif permet de dévier la direction de l'impulsion laser d'un angle radial θ et d'un angle zénithal ϕ. Pour chaque angle $\theta \in [0, 2\pi]$ et $\phi \in [-\pi, \pi]$, la distance ρ entre le LiDAR et la première surface rencontrée par le laser dévié est calculée (Eq 1.1) en prenant t l'intervalle de temps entre l'émission du laser et la réception du signal retour filtré dont l'amplitude est la plus grande. Ainsi il est possible d'enregistrer un triplet (ρ, θ, ϕ) pour chaque émission de laser et de considérer ces triplets comme un ensemble de points dans le repère sphérique qui a comme origine la position du *LiDAR*. Ces points sont ensuite transformés en coordonnées cartésiennes et enregistrés. Chaque point **P** correspond à un rebond d'une impulsion laser et donc à une surface scannée par le *LiDAR*. Le point **P** peut alors être défini par rapport à ses coordonnées dans un repère cartésien arbitraire :

$$\mathbf{P} = (x, y, z) \tag{1.2}$$

Nous pouvons alors exprimer le nuage de points S, résultat de l'acquisition *LiDAR* par balayage, comme un ensemble de N points :

$$S = \{\mathbf{P}_i\}_{i=1}^{i=N} \in \mathbb{R}^{N\times 3} \tag{1.3}$$

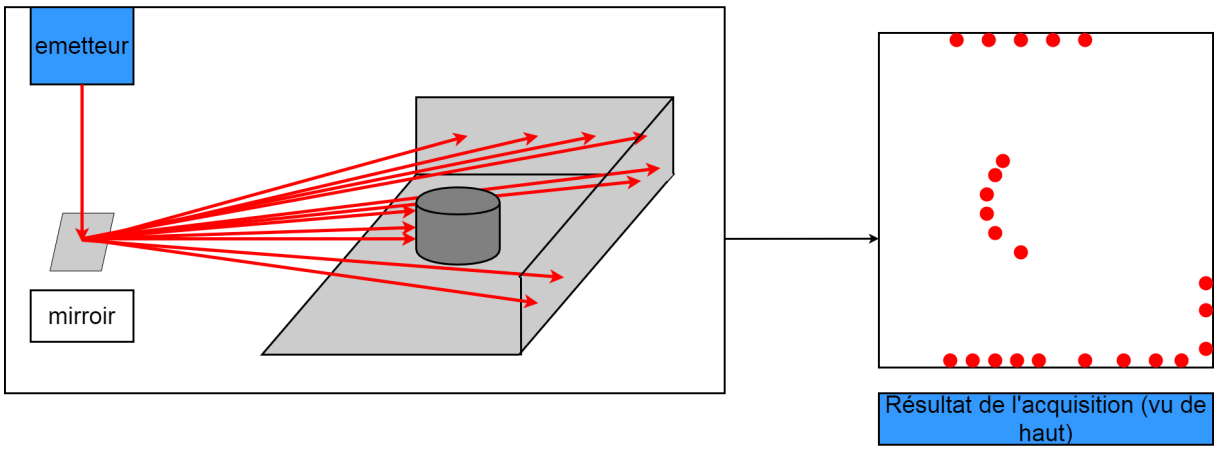

FIGURE 3 – Schéma représentant à gauche un *LiDAR* à balayage grâce à un miroir rotatif. Plusieurs impulsions lasers, en rouge, sont représentées pour des valeurs de θ différentes et à droite une représentation du résultat obtenu par cette acquisition.

Nous représentons le principe de fonctionnement d'un *LiDAR* par balayage dans la figure 3. La densité de ce nuage de points dépend de deux paramètres : la résolution angulaire, fonction de l'angle minimal entre deux émissions de laser, et la portée du *LiDAR*, c'est à dire la distance

maximale pouvant être calculée à partir de la puissance du laser utilisé. Plus ces deux valeurs sont faibles, plus la densité du nuage de points est élevée. Il est possible d'augmenter encore plus cette densité et de limiter les occultations en positionnant le *LiDAR* a différentes positions puis en combinant les nuages de points obtenus à chaque acquisition. Cependant cela nécessite d'aligner les différents nuages de points dans un repère commun et de les fusionner. Cela nécessite souvent le positionnement de marqueurs physiques lors des différentes acquisitions.

1.2.2 *LiDAR* en milieu urbain.

De par leur capacité à numériser des scènes de grandes tailles en reproduisant leur géométrie, les dispositifs *LiDAR* sont aujourd'hui souvent utilisés en milieu urbain. Dans ce cadre, deux solutions de transports des dispositifs *LiDAR* ont été imaginées.

Deux solutions de transport de l'appareil *LiDAR* :

Les dispositifs d'acquisitions statiques ou *TLS* , pour *Terrestrial Laser Scanning*, sont les premiers à utiliser les dispositifs *LiDAR*. Les acquisitions *TLS* nécessitent que le *LiDAR* soit immobile pendant toute la durée de l'acquisition. Elles ont plusieurs applications en milieu urbain, telle que l'estimation du niveau du sol dans des chantiers de constructions ou bien la numérisation du patrimoine architectural. D'autres applications existent comme l'estimation de la couverture végétale d'une forêt [PSDH16] ou en géologie où elles permettent de réaliser des relevés précis sur la structure du sol. Néanmoins, celles-ci ont pour désavantage de nécessiter un temps d'installation du dispositif à chaque nouvelle acquisition. Les acquisitions *TLS* couvrent alors un champ plutôt réduit, de l'ordre de quelques dizaines de mètres [CJO19]. De plus seuls les objets proches du capteur *LiDAR* sont acquis avec une précision maximale et celle-ci se dégrade rapidement pour les objets plus éloignés.

Les dispositifs d'acquisition mobiles, *MLS* et *ALS* , pour *Mobile Laser Scanning* et *Aerial Laser Scanning*, étendent les acquisitions *LiDAR* grâce à l'ajout de capteurs complémentaires. Ainsi, un *LiDAR*, une antenne *GNSS* (*Géolocalisation* et *Navigation* par un *Système* de *Satellites*) et une centrale inertielle peuvent être embarqués sur le même dispositif mobile afin de réaliser une acquisition *LiDAR* dynamique. Cette acquisition est appelée *MLS*, dans le cas où le *LiDAR* est installé sur un dispositif terrestre comme un sac à dos ou une voiture et *ALS* dans le cas où le *LiDAR* est installé sur un véhicule aérien comme un drone ou un avion. Le principe des acquisitions dynamiques, *ALS* ou *MLS*, consiste à utiliser le capteur *GNSS* ainsi que la centrale inertielle pour combiner automatiquement plusieurs nuages de points *LiDAR* (Eq 1.3), issus d'acquisitions par balayage, en un grand nuage de points couvrant la totalité du territoire parcouru.

Les acquisitions *ALS* sont celles qui permettent de couvrir le champ le plus grand en survolant une ville entière. Cependant celles-ci sont moins performantes au niveau de la résolution,

notamment à cause de l'altitude du drone ou de l'avion, qui implique une plus grande distance entre les objets urbains et le capteur *LiDAR*. De leurs côtés, les acquisitions *MLS* permettent une précision bien plus importante en étant plus proches des objets urbains [CJO19]. Néanmoins celles-ci recouvrent un champ bien inférieur aux acquisitions *ALS*, car les véhicules terrestres sont contraints de naviguer selon les axes routiers existants.

Limites et difficultés des acquisitions *LiDAR*

Quelle que soit la solution de transport adoptée, les acquisitions *LiDAR* partagent un certain nombre de limites. Ainsi, des occultations sont toujours présentes dans les scènes numérisées même si elles ne sont pas de même nature. Dans le cas d'acquisition terrestre *TLS*, les occultations sont plutôt horizontales : seule une partie des objets est scannée, selon l'orientation du capteur *LiDAR*, alors que l'autre partie est cachée. Il arrive aussi que des objets soient occultés par d'autres objets plus proches du capteur *LiDAR*. Dans le cas d'acquisitions aériennes *ALS*, les occultations sont plutôt verticales : seules les parties hautes, comme les toits de maison ou de voiture, sont scannées. De même, un certain bruit au niveau de la géolocalisation des données est très souvent constaté pour les 2 solutions de transport.

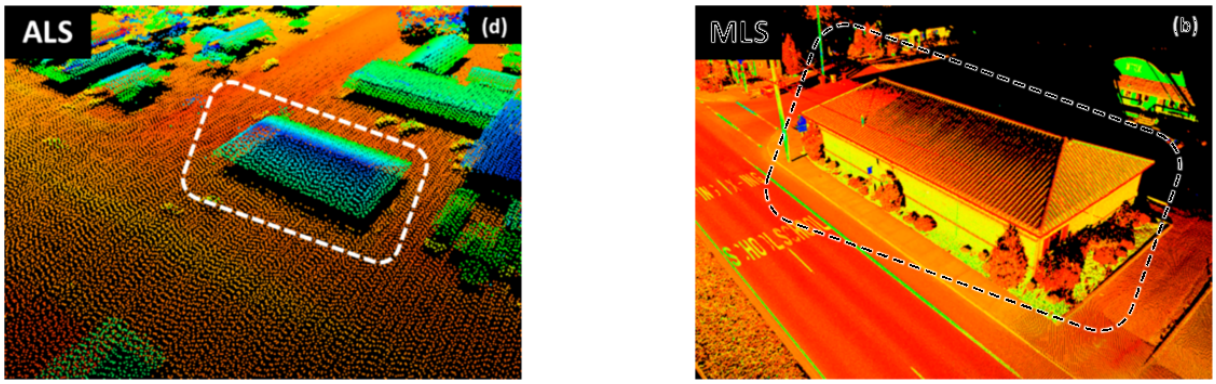

FIGURE 4 – Comparaison entre une acquisition *ALS*, à gauche, et une acquisition *MLS*, à droite, réalisées sur la même zone. Nous pouvons remarquer que dans le nuage de points *ALS* ; à cause des occultations verticales, seul le toit de la maison est reconnaissable et que les arbustes aux abords de la façade ne sont pas visibles. Alors que dans le cas du nuage de points *MLS*, les arbustes et les murs de la maison sont visibles malgré les occultations horizontales qui cachent une façade de la maison. Figure tirée de [CJO19].

D'autres limites sont spécifiques au vecteur de transport choisi, ainsi le champ réduit recouvert par les acquisitions *TLS* les rend peu adaptées pour couvrir une ville. C'est aussi le cas des acquisitions *MLS* où la présence de trafic routier ainsi que la vitesse du véhicule impacte la précision de l'acquisition : plus la vitesse du dispositif augmente moins la précision est importante et plus le trafic routier est important plus il y'a d'occultations. Dans le cas des acquisitions *ALS*, les réglementations juridiques existantes sur le survol de certaines zones peuvent également limiter leurs précision, en imposant par exemple une altitude trop élevée, ou alors tout simplement les rendre impossibles en interdisant le survol.

Dans notre cas, nous considérons que les acquisitions *MLS* sont les plus adaptées, car elles permettent de recouvrir une plus large gamme d'objets urbains que les autres. Ainsi, dans la figure 4, nous constatons que certains objets urbains, tel que les lampadaires ou les arbustes, ne sont pas relevés par l'acquisition *ALS* à cause des occultations verticales, contrairement à l'acquisition *MLS*.

1.2.3 Des données 3D enrichies

Le résultat d'une acquisition *MLS* est un large nuage de points (Eq1.3) couvrant tout le territoire parcouru. Il est alors possible de combiner d'autres capteurs, rigoureusement étalonnés par rapport au dispositif *LiDAR*, afin d'ajouter d'autres informations au nuage de points comme par exemple un appareil photo numérique pour obtenir un nuage de points colorés. Les points sont ainsi caractérisés par d'autres valeurs que leurs positions géométriques (x, y, z), comme par exemple la couleur RGB de ceux-ci. De même, d'autres informations peuvent être ajoutées pour chaque point comme la réflectance du matériau de la surface percutée par le laser et l'angle d'incidence entre le laser et la surface. Dans la suite du manuscrit nous exprimons alors un nuage de points S comme :

$$S = \{\mathbf{P}_i\}_{i=1}^{i=N} \in \mathbb{R}^{N\times d} \text{ avec } d \geq 3 \tag{1.4}$$

avec d la dimension de chaque point.

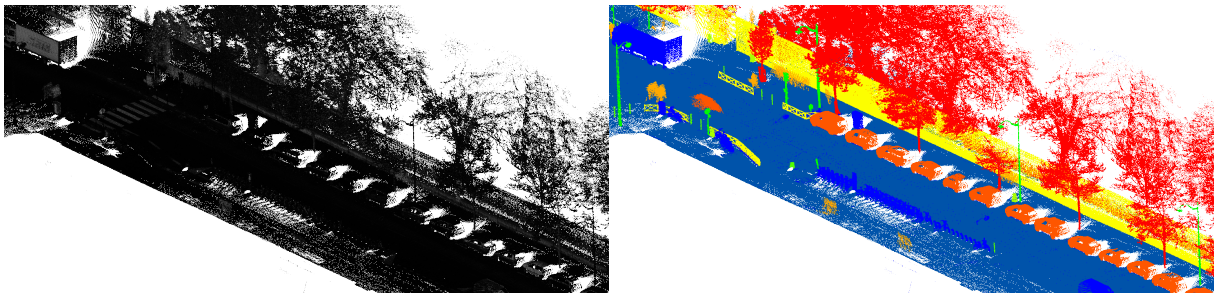

FIGURE 5 – Capture d'écran de l'affichage d'un nuage de points issu d'une acquisition *MLS* en milieu urbain. A gauche les points sont représentés par rapport à la valeur de réflectance du *LiDAR*, plus celle-ci est faible plus le point est sombre et inversement, et à droite chacun des objets a été isolé manuellement et colorié selon la catégorie d'objet à laquelle il appartient.

Observer directement les objets urbains à partir d'un nuage de points *MLS* n'est pour autant pas un problème simple, d'une part à cause de la taille des scènes 3D, autant en nombre de points qu'en nombre d'objets urbains présents, et d'autre part à cause des limites des interfaces de manipulation de scènes 3D. Nous illustrons ce problème dans la figure 5, où il est bien difficile d'identifier tous les objets présents dans le nuage de points. Une telle opération d'identification et d'isolement des objets représente alors un coût important, qui l'est d'autant plus selon la taille du nuage de points, si elle est réalisée manuellement.

1.3 Objectifs de la thèse

Les travaux accomplis dans le cadre de cette thèse se situent au croisement de plusieurs problématiques liés à des champs de recherche différents : le traitement de données 3D, la télédétection et l'apprentissage automatique.

1.3.1 Gestion de grands nuages de points 3D

Les nuages de points sont des *ensembles* de points ; il n'y a donc pas d'ordre entre les points $\mathbf{P}$ appartenant à un même nuage S, ce qui signifie qu'il n'existe pas de corrélation *a priori* entre deux points $\mathbf{P}_i$ et $\mathbf{P}_{i+1}$ ni de lien entre l'indice i d'un point $\mathbf{P}_i$ et sa position dans son repère. Le traitement des nuages de points 3D est plus compliqué que celui des images 2D par exemple, où il existe une topologie implicite de voisinage, la grille de *pixels*, et une corrélation entre *pixels* voisins. Il est donc possible de partitionner une image qui serait trop grande sans en connaître le contenu. La difficulté est d'autant plus accrue dans le cas des nuages de points 3D issus d'acquisitions *MLS* car ceux-ci peuvent contenir plusieurs centaines de millions de points représentant des trajets de plusieurs kilomètres. La complexité de calcul et de stockage ajoute encore plus de difficultés.

Afin d'extraire des informations pertinentes à partir d'un nuage de points, il est alors essentiel de lui appliquer certains pré-traitements. Les plus importants sont les opérations de sous-échantillonnage qui permettent de réduire l'échelle du problème adressé. Par exemple dans le cas d'une acquisition *MLS*, il est courant de séparer celle-ci, pendant l'acquisition, en plusieurs sous-acquisitions donnant chacune un nuage de points. Une autre opération classique est la réduction du nombre de points d'un nuage de points afin de réduire le coût de traitement en termes de temps et de mémoire. Le schéma classique du traitement d'un nuage de points *MLS* S est alors le suivant : une transformation est d'abord appliquée sur S afin d'obtenir une version sous-échantillonnée $\hat{S}$; les informations utiles sont ensuite extraites à partir de $\hat{S}$, par exemple le nombre d'objets urbains présents, puis ces informations sont enfin projetées dans le nuage de points initial S.

1.3.2 Détection d'objets urbains dans des nuages de points 3D

Afin de mettre en pratique une approche automatique de gestion des objets urbains à partir de nuages de points, une série d'opérations successives doit être appliquée : acquisition *MLS*, stockage des nuages de points, extraction de sous-nuages de points correspondants aux objets isolés présents dans chaque nuage de points, mise en relation de ces objets isolés avec les objets présents dans la base de données grâce à leurs positions *GNSS*. Parmi ces opérations, nos travaux s'insèrent dans l'avant-dernière étape, c'est-à-dire la localisation et l'identification automatique d'objets urbains présents dans un nuage de points *MLS*.

Les travaux existants en reconnaissance de formes et en vision par ordinateur catégorisent les

approches visant à la localisation et l'identification automatique d'objets en différentes *tâches*. Nous allons passer en revue les différentes tâches qui correspondent le plus à notre cas. Nous avons choisi de présenter celles-ci par ordre croissant de difficulté.

A. Classification d'objets. Cette tâche consiste à associer un label à un nuage de points ne contenant qu'un seul objet. Dans ce cas, les nuages de points sont constitués uniquement de points qui appartiennent à un même objet : il n'y a qu'un seul objet par nuage et aucun point du nuage n'appartient au fond. Le label, ou classe notée C, assigné au nuage de points est choisi parmi une liste de N_C noms préalablement établie. Nous pouvons alors définir la classification d'un nuage de points S comme ceci :

$$f^{Cls} : \begin{cases} \mathbb{R}^{N\times d} \rightarrow \{1, ...N_C\} \\ S \rightarrow C = f^{Cls}(S) \end{cases} \tag{1.5}$$

où d est la dimension des points $\mathbf{P}$, avec $d \geq 3$, et N est le nombre de points du nuage S.

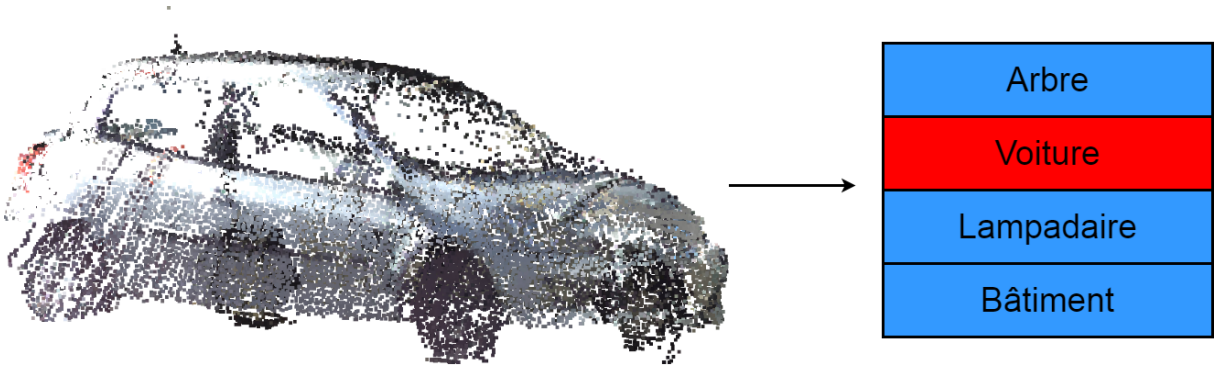

FIGURE 6 – Illustration de la tâche de classification d'objets urbains. Un nuage de points, à gauche, est associé à un label, en rouge, parmi une liste de labels en bleu.

Nous illustrons cette tâche dans la figure 6 où un nuage de points représentant une voiture est associé au label correspondant.

B. Segmentation sémantique. Aussi appelée *classification de points*, cette tâche consiste à assigner un label à chaque point d'un nuage. Le nuage de points représente une scène 3d où sont présents plusieurs objets ainsi que des points appartenant au fond (sol, bâtiment...). Il arrive parfois que le nuage de points représente un unique objet comme dans la classification d'objets : dans ce cas nous parlons de *segmentation de parties*, illustrée dans la figure 8. Les labels, ou classes notées C, assignés au nuage de points sont choisis parmi une liste de N_C noms préalablement établie. Nous définissons alors la segmentation sémantique d'un nuage de points S comme ceci :

$$f^{Seg} : \begin{cases} \mathbb{R}^{N\times d} \rightarrow \{1, ...N_C\}^N \\ \{\mathbf{P}_i\}_{i=1}^{i=N} \rightarrow \{C_i\}_{i=1}^{i=N} = f^{Seg}(S) \end{cases} \tag{1.6}$$

où d est la dimension des points $\mathbf{P}$, avec $d \geq 3$, C_i est le label du point $\mathbf{P}_i$ pour chaque $i \in \{1, ...N_C\}$.

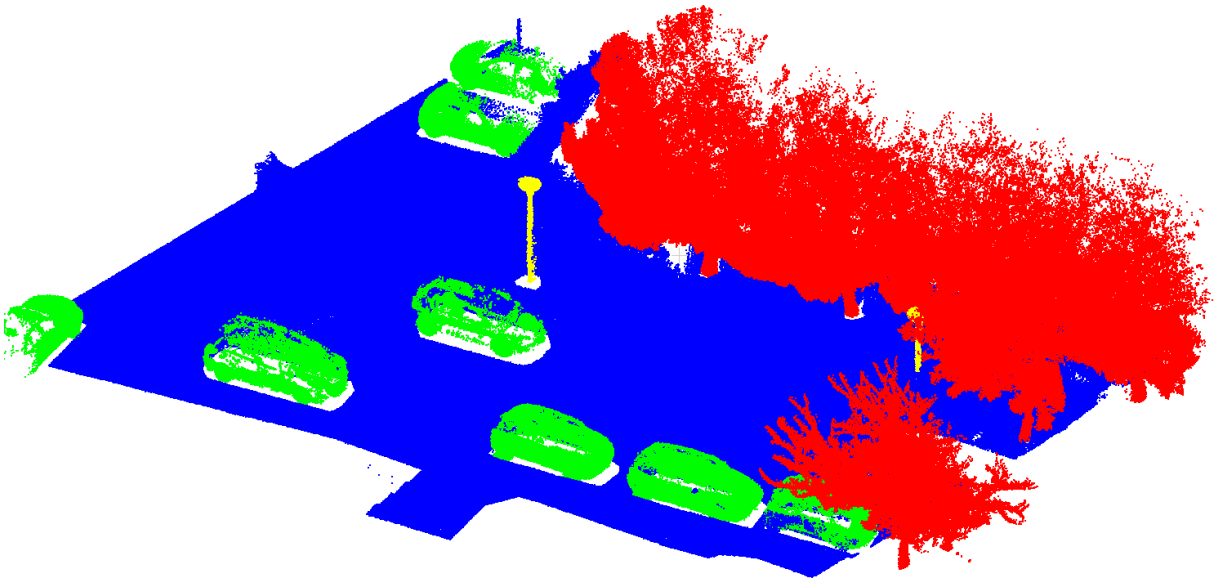

FIGURE 7 – Illustration de la tâche de segmentation sémantique où un label, représenté par une couleur, est associé à chaque point selon l'objet auquel il appartient. Les points bleus correspondent au fond, les rouges aux arbres, les verts aux voitures et les jaunes aux lampadaires.

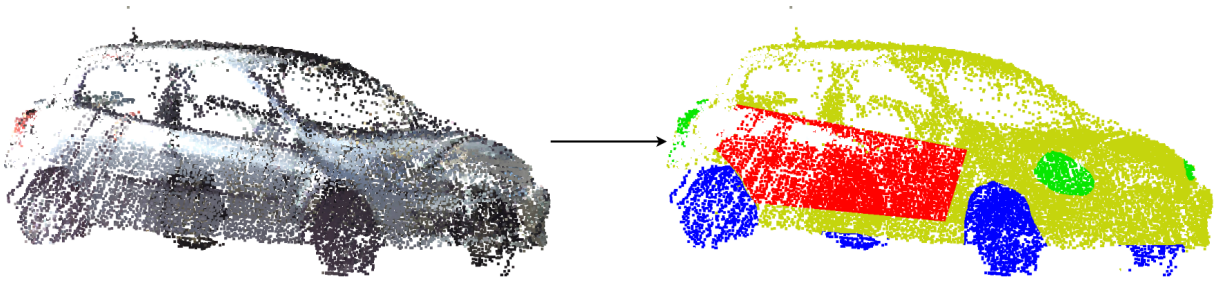

FIGURE 8 – Illustration d'une variante de la segmentation sémantique où le nuage en entrée, à gauche, est un unique objet. Il s'agit alors de *segmentation de parties*, où un label est associé à chaque point selon la partie de l'objet à laquelle il appartient. Ainsi les points appartenant aux portières sont en rouge, les pneus en bleu, les feux en vert et le reste de la carrosserie en jaune.

Un exemple de segmentation sémantique appliqué à un nuage de points issu d'une acquisition *MLS* est présentée dans la figure 7. Chaque couleur correspond à une classe différente d'objets.

C. Détection d'objets. Cette tâche consiste à localiser tous les objets présents dans un nuage de points. Ainsi pour chaque objet présent dans le nuage de points, il faut lui assigner un label et déterminer sa position ainsi que sa forme approximative. Le plus souvent la forme est déterminée sous la forme d'une boîte englobante, c'est-à-dire un parallélépipède rectangle orienté selon les axes du référentiel d'acquisition. Nous définissons alors la détection d'objets dans un nuage de points S comme ceci :

$$f^{Detec} : \begin{cases} \mathbb{R}^{N\times d} \rightarrow \left[\mathbb{R}^6 \times \{1, ...N_C\}\right]^* \\ S \rightarrow \{\mathbf{B}_i, C_i\}_{i=1}^* = f^{Detec}(S) \end{cases} \tag{1.7}$$

où chaque boîte englobante $\mathbf{B}_i$ est définie par deux points *(x, y, z)*, ce qui donne $\mathbf{B}_i \in \mathbb{R}^6$, et C_i est le label attribué à cette boîte. On notera N_b le nombre de boîtes détectées.

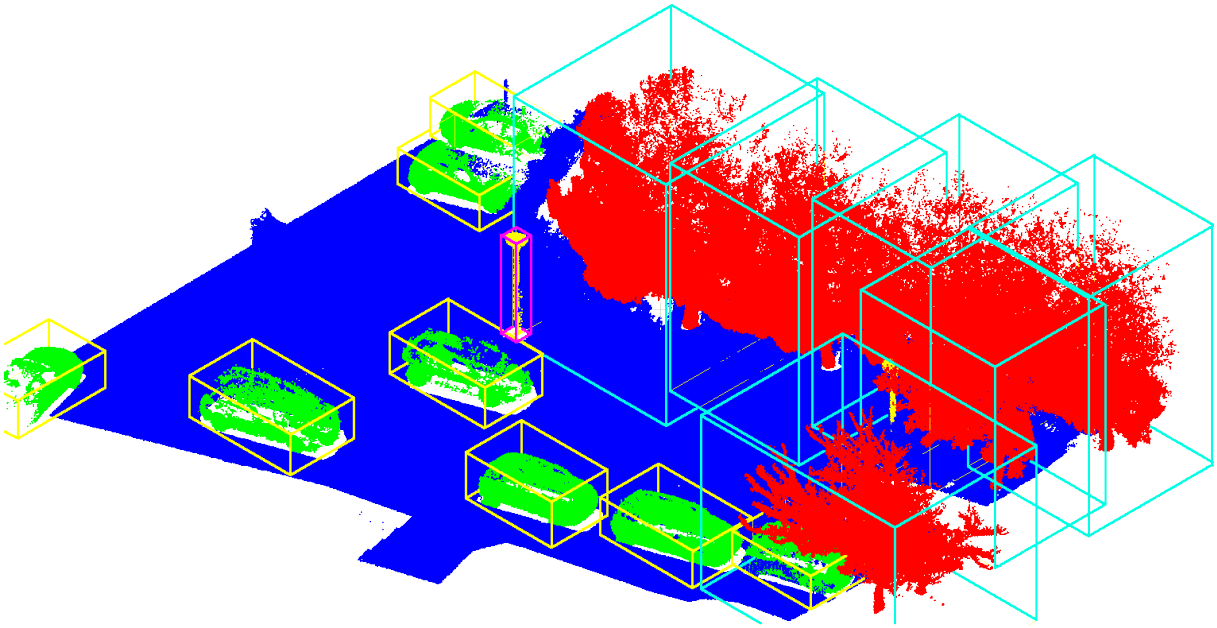

FIGURE 9 – Illustration de la tâche de détection d'objets : pour chaque objet, une boîte englobante délimite sa forme approximative et un label lui est associé. Le label de l'objet est représenté par la couleur de la boîte englobante : bleu clair pour les arbres, jaune pour les voitures et rouge pour les lampadaires. Chaque point est colorié selon la classe de l'objet auquel il appartient.

Un exemple de détection d'objets appliquée à un nuage de points issu d'une acquisition *MLS* est présenté dans la figure 9. Chaque couleur de boîte englobante correspond à une classe différente.

D. Segmentation d'instance. Il s'agit de la tâche la plus difficile ; celle-ci consiste à isoler chacun des objets présents dans un nuage de points et à lui assigner un label. Ainsi le nuage de points est décomposé en plusieurs *sous-nuages* : qui contiennent soit uniquement les points appartenant au fond, soit uniquement les points d'un même objet. Nous définissons alors la segmentation d'instance dans un nuage de points S comme ceci :

$$f^{Inst} : \begin{cases} \mathbb{R}^{N\times d} \rightarrow \left(\mathbb{R}^{N\times d}\right)^* \\ S \rightarrow P(S) = f^{Inst}(S) \end{cases} \tag{1.8}$$

avec $P(S)$ la partition du nuage S tel que $P(S) = \{S_i\}_{i=1}^*$ et $S_i \cap S_j = \emptyset \ \forall_{i\neq j}$, où les S_i sont les *sous-nuages* de S composés uniquement de points appartenant au même objet. On notera N_B le

nombre d'objets présents dans le nuage de points S et $N^* < N$ le nombre de points, différent pour chaque S_i.

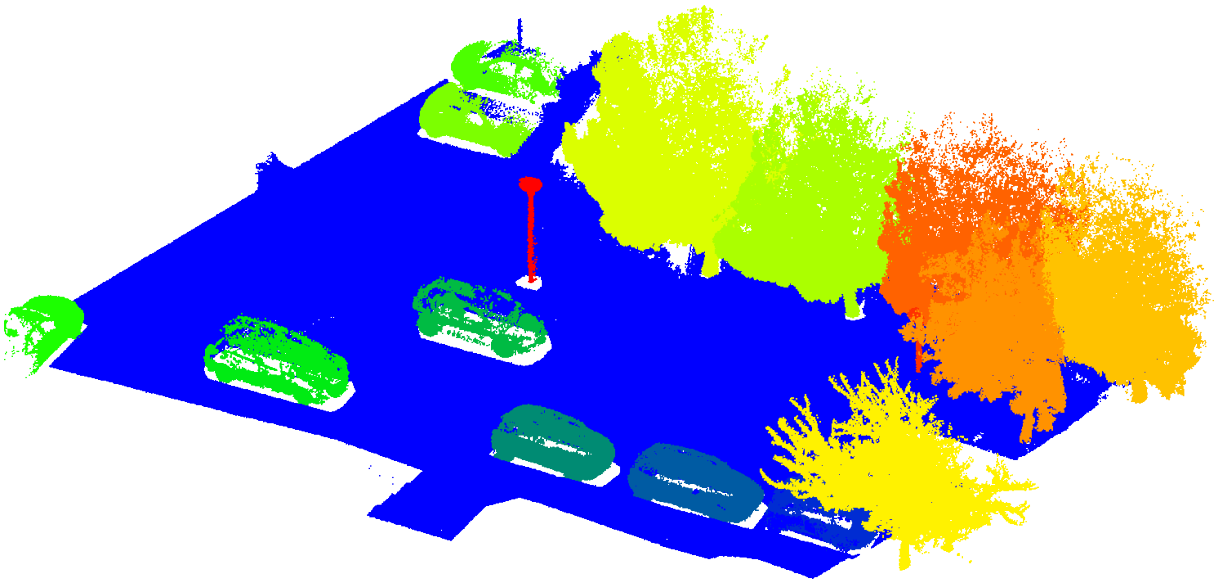

FIGURE 10 – Illustration de la tâche de segmentation d'instances : à chaque point est associé un label selon l'objet auquel il appartient. Ainsi tous les points appartenant au fond sont assignés au même label et tous les points appartenant à un même objet sont assignés à un unique label correspondant à cet objet.

Un exemple de segmentation d'instances appliquée à un nuage de points issu d'une acquisition *MLS* est présenté dans la figure 9. Chaque couleur de points correspond à un label différent et donc à un objet différent.

La *classification* (**A**) ainsi que la *segmentation sémantique* (**B**) ne sont pas adaptées à notre cas car elles ne permettent pas de distinguer les objets dans un nuage de points mais uniquement leurs classes. La *détection d'objets* (**C**) est alors la tâche la plus adaptée à notre cas. En effet, déterminer la position de chaque objet, dans un nuage de points géo-référencé, permet de faire le lien avec la base de données des objets urbains. La *segmentation d'instance* (**D**) pourrait aussi être adaptée à notre cas, en effet il est immédiat d'obtenir une boîte englobante à partir des contours des instances des objets. Néanmoins, il suffit d'une erreur de segmentation au niveau de quelque points pour que les boîtes englobantes qui en découlent soient complètement erronées. Alors que dans le cas de la détection d'objets, la marge d'erreur sur la définition des boîtes englobantes (tailles et dimensions) peuvent être plus tolérables, notamment dans le cas d'objets occultés.

1.3.3 Plan de la thèse

Les travaux accomplis lors de cette thèse sont répartie en 5 chapitres.

Dans le deuxième chapitre, nous exposons l'état de l'art des méthodes de *détection automatique* d'objets urbains dans des nuages de points 3D. Nous y présentons dans un premier temps,

les méthodes de partitionnement et de classification de nuages de points 3D et dans un second temps les principes des réseaux de neurones profonds ainsi que les approches permettant de les généraliser aux nuages de points 3D.

Dans le troisième chapitre, nous présentons notre contribution principale, plus particulièrement l'architecture neuronale que nous proposons pour la détection d'objets urbains dans des nuages de points 3D. Cette architecture s'appuie sur la définition d'une nouvelle couche de calcul, que nous appelons *cluster-convolution*.

Dans le chapitre 4, nous présentons les applications de nos travaux sur la détection d'objets. Ce chapitre est divisé en deux parties, une première sur les expériences que nous avons menées sur la *classification d'objets* (cf. **A**) urbains, et une deuxième sur l'application de notre architecture neuronale au problème de *détection d'objets* (cf. **C**) urbains dans des nuages de points MLS.

Ce manuscrit se conclut dans le chapitre 5 par un bilan de nos travaux de thèse et par une ouverture sur des perspectives de nos travaux ainsi que de mises en relation avec des travaux connexes dans le chapitre 6.

Chapitre 2

Etat de l'art

Dans la suite nous distinguons quatre échelles auxquelles peuvent être réalisés les différents traitements sur les nuages de points.

L'échelle la plus bas niveau se situe au niveau des points. Ce niveau correspond aux opérations visant à obtenir un résultat sur un unique point, comme par exemple la prédiction d'un label pour chaque point dans la tâche de segmentation sémantique définie dans le chapitre 1 (Eq 1.6).

Au contraire, l'échelle la plus haute est celle des nuages de points en entier, c'est-à-dire les opérations qui visent à obtenir un résultat unique pour l'ensemble des points du nuage. C'est le cas par exemple de la tâche de classification de nuages de points présentée dans le chapitre 1 (Eq 1.5).

Entre ces deux niveaux extrêmes, il existe deux niveaux intermédiaires : le niveau région et le niveau objet. Une région, notée R_{3D}, d'un nuage de points S, est composé d'un sous-ensemble des points de ce nuage :

$$R_{3D} = \{\mathbf{P}_1, ..., \mathbf{P}_N | \forall_{1 \leq i \leq N} \mathbf{P}_i \in S\}$$

et les points de ce sous-ensemble R_{3D} caractérisent un espace délimité du nuage qui représente un certain intérêt sémantique, comme par exemple la présence d'un objet. Le niveau objet est alors une région composée uniquement de points appartenant au même objet.

Notons que l'on fait ici la distinction entre une région notée R_{3D}, qui est de forme arbitraire, et la notion de boîte englobante, notée $\mathbf{B}$ (Eq 1.7), et qui correspond à un parallélépipède rectangle.

2.1 Premières méthodes de détection d'objets dans un nuage de points 3D

Les premières méthodes de détection d'objets urbains proposent chacune de se concentrer sur une seule classe d'objets bien spécifique à la fois, comme par exemple les arbres [YLG+14]. Malgré le fait que ces méthodes divergent énormément selon la classe d'objet pour laquelle elles

sont définies, elle suivent toutes un schéma similaire :

1. Le nuage de points S est d'abord *partitionné* en plusieurs régions $\{R_{3D} \in S\}$.
2. Chaque région R_{3D} est ensuite *classifiée* comme étant un objet ou non.

2.1.1 Quelques méthodes de partitionnement de nuage de points 3D

Nous pouvons regrouper les méthodes de partitionnement d'un nuage de points en deux familles distinctes :

- les approches de partitionnement global,
- les approches de partitionnement hiérarchique.

Partitionnement global

Parmi les méthodes de partitionnement global, les plus connues sont appelées méthodes de *clustering global*. Les méthodes de *clustering global* ne sont pas exclusives aux nuages de points, celles-ci proviennent du domaine de l'analyse statistique. Elles reposent sur le calcul d'un nombre fixe N_S de *centroïdes* $c \in \mathbb{R}^3$, autour desquels vont se répartir les points du nuage pour former des régions. Les différentes régions, aussi appelées *clusters*, sont alors composées des points les plus proches, selon une distance D, de leur *centroïde* respectif et sont donc au nombre de N_S. Formellement l'opération de *clustering* peut être définie comme ceci :

$$Clustering : \left| \begin{array}{l} \mathbb{R}^{N\times 3} \rightarrow \mathbb{R}^{N_S\times 3} \\ S \rightarrow \{\mathbf{c}_1, ...\mathbf{c}_{N_S}\} \end{array} \right.$$

$$\forall_{1\leq i\leq N_S} R_i = \{\mathbf{P} \in S | D(\mathbf{P}, \mathbf{c}_i) < D(\mathbf{P}, \mathbf{c}) \forall \mathbf{c} \neq \mathbf{c}_i\}$$

$$\bigcup_{i=1}^{N_S} R_i = S$$

$$\bigcap_{i=1}^{N_S} R_i = \emptyset$$

où D est la mesure de *distance euclidienne* entre deux points.

L'approche la plus répandue pour calculer les *centroïdes* est l'algorithme itératif du *k-means* [Llo82]. Celui-ci consiste à tirer aléatoirement un nombre N_S de points appartenant au nuage de points S comme première estimation des *centroïdes*. Puis les régions sont formées en prenant pour chaque point le centroïde dont il est le plus proche. Une nouvelle estimation des *centroïdes* est ensuite calculée en prenant pour chaque région la moyenne arithmétique des points qui la

composent. Ce processus est répété pendant un certain nombre d'itérations ou bien jusqu'à ce que la répartition des points dans les *clusters* ne change plus d'une itération à l'autre.

Les méthodes de partitionnement global comportent deux limites dans le cadre des nuages de points :

- la définition du nombre de *centroïdes* N_S nécessite une étude préalable des scènes 3D afin d'avoir une première approximation du nombre d'objets.
- l'utilisation de la *distance euclidienne* pour la construction des régions qui a pour effet d'aboutir à la formation de régions de formes convexes uniquement, là ou des régions de formes arbitraires seraient plus adaptées à la variété de forme des objets urbains.

Partitionnement hiérarchique

Ces méthodes reposent sur la décomposition récursive du nuage de points entre points similaires et points différents, ainsi que sur la définition d'une condition d'arrêt. La décomposition peut être effectuée : (a) de manière *divisive*, '*top-down*', dans ce cas là on parlera de *clustering hiérarchique* ou bien (b) de manière *agglomérative*, ou '*bottom-up*', dans ce cas là on parlera plutôt de *croissance de régions*. La différence majeure avec les méthodes de *clustering global* est alors l'absence de paramètre relatif au nombre de régions : celui-ci est variable d'un nuage de points à l'autre.

a) Clustering par densité . Ces méthodes proposent de récursivement extraire du nuage les points qui partagent des caractéristiques similaires afin de former des régions. Les points qui n'appartiennent à aucune région à la fin de la phase récursive sont alors considérés comme du bruit. Parmi celles-ci la méthode la plus utilisée est l'algorithme *DBSCAN* [EKSX96] qui revient à regrouper les points en utilisant pour seul critère la densité des régions.

L'algorithme du *DBSCAN* [EKSX96] consiste à parcourir l'ensemble des points du nuage : chaque fois qu'un point satisfait la contrainte pour être un *point centre*, c'est-à-dire si le nombre de points dans un rayon ϵ autour de lui est supérieur à un certain seuil M, alors il est colorié et l'ensemble des points dans le rayon ϵ est ajouté à la liste des points à traiter. Une fois que la liste des points à traiter est vide, un nouveau point est choisi au hasard parmi ceux non traités. A la fin de l'algorithme, les points de même couleur sont assignés à la même région et les points qui n'ont pas de couleur sont considérés comme appartenant au fond.

Cet algorithme permet d'obtenir des *clusters* de formes arbitraires et non uniquement convexes, ce qui est plus intéressant dans le cas des objets urbains qui ont des formes très variables. Cependant le résultat de l'algorithme est très sensible au choix des paramètres, en particulier le rayon ϵ, qui définit le voisinage de chaque point.

Des méthodes d'estimation automatique de la valeur du paramètre ϵ [SEKX98][SSE+17] permettent de s'adapter aux nuages de points en entrée. L'algorithme *OPTICS* [ABKS99] réarrange l'ordre des points afin d'appliquer l'algorithme *DBSCAN* [EKSX96] de manière *hiérarchique*,

avec une valeur croissante pour ϵ, ce qui permet d'extraire des *clusters* de densités différentes. Plusieurs autres versions améliorées ont depuis été proposées : [WJW+19][CMZS15].

b) Croissance de régions . Ces méthodes proposent de récursivement agglomérer les points suffisamment proches les uns des autres afin de créer des régions qui contiennent de plus en plus de points et qui sont de moins en moins nombreuses. Ces méthodes suivent un schéma en deux étapes :

1. Un sous-ensemble du nuage original $S_G \in S$ est d'abord sélectionné. Les points du sous-ensemble S_G sont appelés graines.

2. Un taux de ressemblance entre chaque point graine $\mathbf{P}_G$ et chacun des points $\mathbf{P}$ dont il est le plus proche est calculé. Si ce taux de ressemblance est inférieur à un certain seuil prédéfini τ, alors le point $\mathbf{P}$ est ajouté à la région associée au point $\mathbf{P}_G$. L'algorithme va ainsi permettre d'agglomérer récursivement les points qui partagent des caractéristiques similaires dans des régions qui augmentent en taille au fur et à mesure de l'exécution de l'algorithme.

L'avantage principal de ces méthodes est leur facilité d'implémentation : mis à part quelques améliorations, seule le calcul du taux de ressemblance diffère entre les méthodes. Néanmoins, ces méthodes sont peu robustes à cause de l'importance du choix des paramètres, comme par exemple le seuil τ. De plus, la plupart de ces méthodes nécessitent de *trianguler* préalablement le nuage de points.

En 1988 [BJ88], la première méthode par croissance de régions est développée. Elle repose sur la segmentation des points en fonction de la valeur et du signe de leurs *courbures moyennes et gaussiennes* [BJ86] comme graines. L'étape de croissance est ensuite calculée en essayant d'ajuster un nombre fixe (4) de surfaces paramétriques, sur les graines et leurs voisins. Cette méthode peut être généralisée aux nuages de points en opérant au préalable une triangulation de ces derniers [Gor02].

Plusieurs améliorations de cet algorithme ont été proposées. [TP05] reprend cet algorithme, en définissant des plans pour chaque graine à partir de la définition d'un voisinage local basé sur une approche des k plus proches voisins dans un espace cylindrique prédéfini. [VS05] montre expérimentalement que les points d'intérieur, c'est-à-dire les points qui ne sont pas situés sur des bordures d'objets, donnent de meilleures régions. [Des10] propose de calculer pour chaque point un *score de planéité* locale afin de conserver comme régions graines uniquement les points avec le plus haut score.

2.1.2 Quelques méthodes de classification de nuages de points

Les approches que nous présentons ici s'appliquent toutes sur des nuage de points. Cependant dans notre cas d'utilisation, celles-ci sont appliquées sur des régions extraites grâce à une des méthodes présentées dans la section précédente. Ainsi dans cette section nous utilisons le terme

de nuage de points pour décrire les données sur lesquelles s'appliquent ces méthodes, sans perte de généralité.

Approches par modèles

L'approche par modèle consiste à associer à chaque nuage de points un modèle mathématique dont les paramètres sont ajustés afin de correspondre le plus possible au nuage.

La *transformée de Hough* [Hou60] est une opération initialement introduite dans le domaine du traitement des images. Celle-ci consiste, dans sa forme la plus basique, à projeter une image dans un espace défini par l'ensemble des paramètres des droites contenues dans cette image [DH72]. Elle peut être généralisée à d'autres types de primitives comme les cercles ou bien sur des formes géométriques arbitraires [Bal81]. Plus intéressant dans notre cas, elle peut être étendue aux nuages de points 3D [DH72].

Tout comme sur les images, la *transformée de Hough* permet de détecter des primitives dans un nuage de points ou bien, dans sa forme généralisée, des formes arbitraires. Le cas le plus simple est la détection de plan dans un nuage de points [DH72].

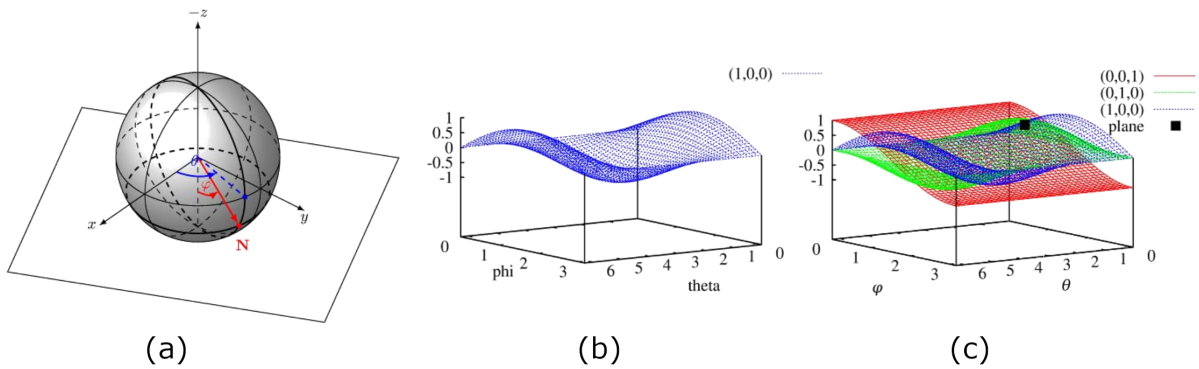

FIGURE 11 – Illustration de la transformée de Hough dans le cas de la détection de plans. En *(a)* le système de coordonnées utilisé, en *(b)* la représentation d'un point dans l'espace des plans qui correspond à l'ensemble des plans de l'espace *(ρ, θ, ϕ)* qui intersectent ce point et en *(c)* l'intersection de plusieurs surfaces en un point, qui correspond à plan de l'espace *(ρ, θ, ϕ)*. Figure tirée de [DH72]

Dans ce cas la transformation s'appuie sur la définition des plans sous leur *forme normale de Hesse*. Ainsi chaque plan est défini par un vecteur **n** ainsi qu'une distance ρ, de telle sorte que chaque point **P** du plan vérifie la relation : $\mathbf{P}\cdot\mathbf{n}=\rho$. Le vecteur **n** peut être défini par les coordonnées polaires (θ, ϕ), comme dans la figure 11 (*a*). Chaque plan peut alors être défini simplement par un triplet (θ, ϕ, ρ). *La transformée de Hough* consiste à projeter les nuages de points dans l'espace $\mathbb{R}^3$ des plans (θ, ϕ, ρ), comme dans la figure 11 (*b*). Ainsi une fois cet espace des plans discrétisé en cellules, il est possible de calculer pour chaque point 3D les plans auxquels il appartient et d'incrémenter la valeur des cellules associées à ces plans. Enfin il suffit de conserver les cellules avec les valeurs maximales pour obtenir les plans les plus susceptibles

de se trouver dans le nuage de points, en effet ces cellules correspondent alors à l'intersection de plusieurs points dans l'espace des plans, comme dans la figure 11 (*c*). Il existe plusieurs améliorations [IK88][KHXO95][BELN11] de cette approche visant soit à minimiser le temps de calcul de la projection du nuage de points dans l'espace des plans [KEB91][YK94][GMK99], soit à augmenter la fidélité de la discrétisation de cet espace [CC09][ZP02].

L'algorithme *RANSAC* [FB81] est une méthode qui permet, à partir d'un modèle préexistant et d'un ensemble d'observations, de séparer les observations conformes au modèle (appelées *inliers*) de celles qui n'y sont pas conformes (appelées *outliers*). Dans notre cas l'ensemble d'observations est un nuage de points S et on cherche à savoir quels sont les points $\mathbf{P} \in S$ qui correspondent à un modèle. Les paramètres de ce modèle sont déterminés par l'algorithme afin de trouver ceux qui sont les plus adaptés au nuage de points S.

Par exemple, dans le cas où le modèle choisi est un plan, seuls 3 paramètres sont à déterminer, il s'agit des trois points du nuage de points S qui définissent le plan le plus adapté. L'algorithme va alors procéder de manière itérative :

1. tout d'abord 3 points sont sélectionnés aléatoirement parmi le nuage de points S
2. le plan $\mathscr{P}$ défini par ces 3 points est calculé
3. pour chaque autre point $\mathbf{P} \in S$, l'algorithme calcule la distance entre le point $\mathbf{P}$ et le plan $\mathscr{P}$ et si celle-ci est inférieure à un seuil τ, alors le *score* associé au plan $\mathscr{P}$ est incrémenté
4. les étapes **1** à **3** sont répétées un nombre N de fois
5. le résultat est le plan dont le *score* est le plus haut.

Cet algorithme simple est particulièrement efficace dans le cas où la forme approximative des objets est connue et peut être représentée par un modèle paramétrique, par exemple des objets sphériques. Cependant, les performances de l'algorithme dépendent beaucoup du choix des paramètres N et τ.

Classification fondée sur des *descripteurs*

Ces approches consistent à construire un modèle à partir des données dont nous disposons. Dans le cas des acquisitions *LiDAR*, certains nuages de points 3D comptent plusieurs millions de points et contiennent plusieurs centaines d'objets. La quantité de données caractérisant chaque nuage de points est alors trop importante pour pouvoir les traiter en utilisant directement les méthodes classiques de l'apprentissage automatique, tels que les *classifieurs SVM* [BGV92] ou les algorithmes de *forêts aléatoires* [BMS+11]. Afin de contourner cette difficulté et en prenant exemple sur l'apprentissage automatique dans le cas du traitement des images, il est possible de passer par une représentation intermédiaire des nuages de points appelée *vecteur descripteur*.

Il existe un grand nombre de méthodes pour représenter un nuage de points sous la forme d'un *vecteur descripteur* [HSSX18]. Ces méthodes se répartissent en deux familles : les *descripteurs* globaux et les *descripteurs* locaux.

***Descripteur* global** Le but d'un *descripteur* global est de représenter le nuage de points dans son entièreté par un unique vecteur. Par exemple, [PRM+00] propose de construire un *descripteur* à partir des moments statistiques M du nuage de points S. Chaque moment M est défini par :
‘

$$M_{q,r,s}(S) = \sum_{i=1}^{N} (x_i - x_B)^q (y_i - y_B)^r (z_i - z_B)^s$$

où (x_B, y_B, z_B) est le barycentre du nuage de points S et (x_i, y_i, z_i) les coordonnées du point $\mathbf{P}_i$ du nuage S. Le descripteur est alors un vecteur composé de quatre valeurs : la valeur de $M_{q,r,s}$ et les trois valeurs du triplet (q, r, s). Les valeurs du triplet (q, r, s) définissent l'ordre du moment statistique calculé ; plus celui-ci est grand, plus le descripteur sera concentré sur les détails de la forme du nuage de points et inversement, plus celui-ci est petit, plus le descripteur sera concentré sur la forme générale du nuage de points.

***Descripteur* local** Contrairement aux *descripteurs* globaux, plusieurs *descripteurs locaux* sont calculés pour un même nuage de points. Ceux-ci correspondent au calcul de caractéristiques dans plusieurs régions du nuage de points ou bien à partir d'un sous-ensemble de points.

En prenant en entrée un *maillage*, [JH99] propose de calculer une image 2D, appelée *spin-image* en chaque point du maillage. Pour cela, un vecteur normal $\mathbf{n}$ est associé à chaque point en interpolant un plan à partir de ses voisins. Chaque point $\mathbf{P}$ est ainsi défini par : $\mathbf{P} = \{x, y, z, \mathbf{n}\}$ où $\mathbf{n}$ est le vecteur normal associé au point $\mathbf{P}$. Le processus de calcul des *spin-images* est illustrée dans la figure 12 pour 3 points différents. Pour chacun de ces points $\mathbf{P}$, une fonction de projection des points 3D vers la *spin-image* est définie. Le support de cette fonction est définie par 3 paramètres : la largeur de la *spin-image* l, la hauteur de la *spin-image* h et l'angle a. Les paramètres l et h sont définis par rapport à un repère cylindrique d'origine le point $\mathbf{P}$ et d'axe vertical le vecteur normal $\mathbf{n}$, représenté par une flèche dans la figure 12. Ainsi l est la distance radiale α maximale par rapport à l'axe $\mathbf{n}$ et h l'élévation β maximale selon $\mathbf{n}$. L'angle a est un seuil de planéité entre le point $\mathbf{P}$ et les autres points $\mathbf{P}_i$ du nuage de points : le point $\mathbf{P}_i$ est dans le support de la *spin-image* associé au point $\mathbf{P}$ si et seulement si : $\arccos(\mathbf{n} \cdot \mathbf{n}_i) < a$.

Ainsi chaque pixel (x, y) de la *spin-image* a pour valeur le nombre de points appartenant à son support et qui ont pour coordonnées cylindriques : $\alpha = x$ et $\beta = y$. Enfin une mesure de similarité entre objets et scènes 3D peut être déterminée en calculant la distance ou le coefficient de corrélation entre les *spin-images* respectives.

Même si ils sont moins sensibles aux occultations que les *descripteurs globaux*, la plupart des *descripteurs locaux* [RBB09][STD14][GSB+13] nécessitent tout de même de trianguler au

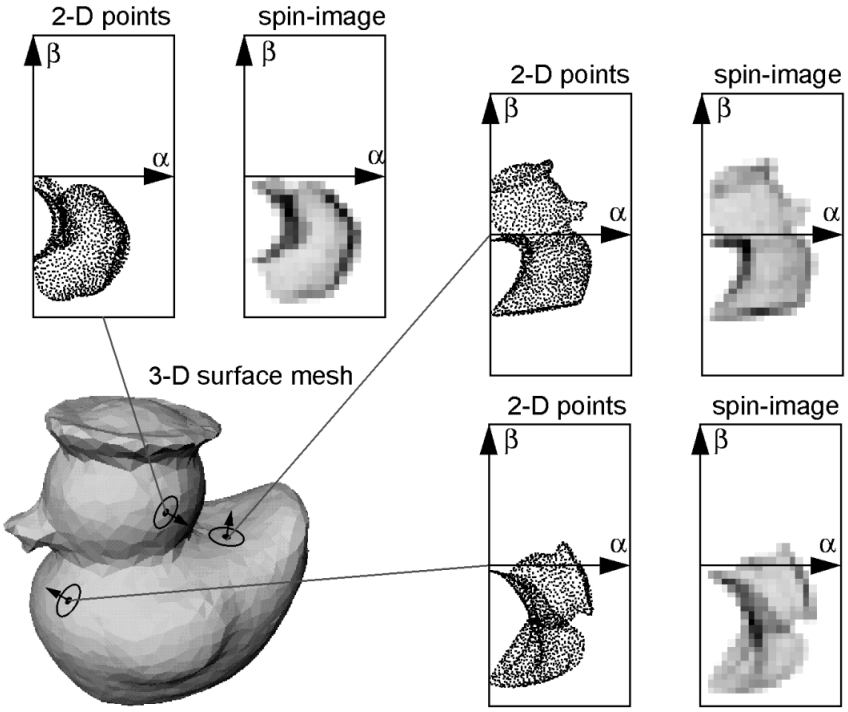

FIGURE 12 – Sur chacun des trois points ici sélectionnés, la flèche représente l'axe **n** dont dépend l'élévation β et le cercle représente la distance radiale α. A partir de ce repère cylindrique, les points appartenant au support de chaque spin-image sont représentés à gauche de celle-ci. Figure tirée de [JH99].

préalable les nuages de points, ce qui peut à la fois être coûteux en termes de calcul et source d'erreurs selon la résolution du nuage de points.

Méthodes de classification Après avoir calculé un *descripteur* par nuage de points, il reste à classifier ceux-ci afin de déterminer la classe des nuages de points. Pour cela il existe plusieurs approches dont nous proposons de présenter rapidement les principales :

- Seuillage : il s'agit de l'approche la plus simple, elle consiste à déterminer un seuil τ sur les valeurs du *descripteur* à partir duquel le nuage de points peut être considéré comme un objet. La valeur de ce seuil τ peut être définie de manière uniquement empirique, ou bien dépendre des valeurs du *descripteur*, dans ce cas on parle de *seuillage adaptatif*.

- Apprentissage : ces méthodes plus récentes [Bre01][CV95] consistent à apprendre les paramètres d'un modèle, la plupart du temps non linéaire, à partir de l'ensemble des *descripteurs* à disposition. Par exemple, les forêts d'arbre décisionnels [Ho95], sont un moyen de classifier un vecteur *descripteur* à partir d'un ensemble d'*arbres décisionnels*. Chacun de ces arbres est entraîné sur un échantillon de l'ensemble des *descripteurs* et cherche, à chaque itération, à diviser cet échantillon en deux dans le but de minimiser l'entropie de chacun des nouveaux échantillons qui en résultent.

2.1.3 Applications à la détection d'objets urbains dans un nuage de points 3D

Ici nous allons présenter les approches de détection d'objets urbains dans des nuages de points 3D. Ces approches s'appuient sur les méthodes présentées en 2.1.1 et 2.1.2. Les spécificités de ces méthodes varient grandement selon la classe des objets urbains que l'on cherche à détecter [WPC18][CJO19]. Néanmoins ces méthodes suivent globalement un schéma en 3 étapes :

1. Tous les points appartenant au sol sont automatiquement détectés et extraits du nuage de points.
2. Le nuage de points est ensuite partitionné en régions grâce à une des méthodes présentées dans la section 2.1.1.
3. Une des méthodes présentées dans la section 2.1.2 est ensuite appliquée afin de distinguer les régions qui contiennent un objet.

Extraction du sol Les méthodes les plus simples d'extraction des points appartenant au sol, dans le cas des nuages de points orientés avec Z la coordonnée représentant l'altitude, reposent sur un *clustering hiérarchique* des points d'altitude minimale. Par exemple dans [HV17], le nuage de points est d'abord partitionné selon une grille horizontale. Pour chaque cellule de cette grille, deux *clusters* sont définis : le premier pour les points appartenant au sol et le second pour les autres. Initialement, tous les points sont ajoutés au second *cluster*. Les points sont successivement déplacés du second au premier *cluster* par ordre croissant d'altitude tant que l'écart-type de l'altitude de ces points de ne dépasse pas un seuil fixé. D'autres méthodes [YSL17], proposent d'utiliser une *distribution gaussienne* au lieu de l'écart-type comme condition d'arrêt. Ces approches sont illustrées dans la figure 13, avec la grille horizontale représentée en *(a)* et le calcul sur la répartition des points selon leurs altitudes en *(b)*.

Des méthodes plus complexes proposent de se baser sur une approche par croissance de régions. C'est le cas de l'algorithme *PTD* [Axe00][LZ15] où les points avec les plus basses altitudes sont choisis comme les graines. Les critères de similitude sont alors la planéité et l'altitude moyenne des régions formées.

Une fois les points appartenant au sol retirés, il est plus facile de *partitionner* les objets restants dans des régions grâce à une méthode de *clustering* ou bien de *croissance de régions*. Il est alors possible d'isoler les objets et de les classifier en partant des objets les plus volumineux comme les immeubles.

Façades d'immeubles Lors d'une acquisition *LiDAR* terrestre, seules les façades des immeubles sont présentes (contrairement aux acquisitions aériennes où le toit est présent). L'essentiel des méthodes de détection des bâtiments consiste alors à détecter des plans verticaux.

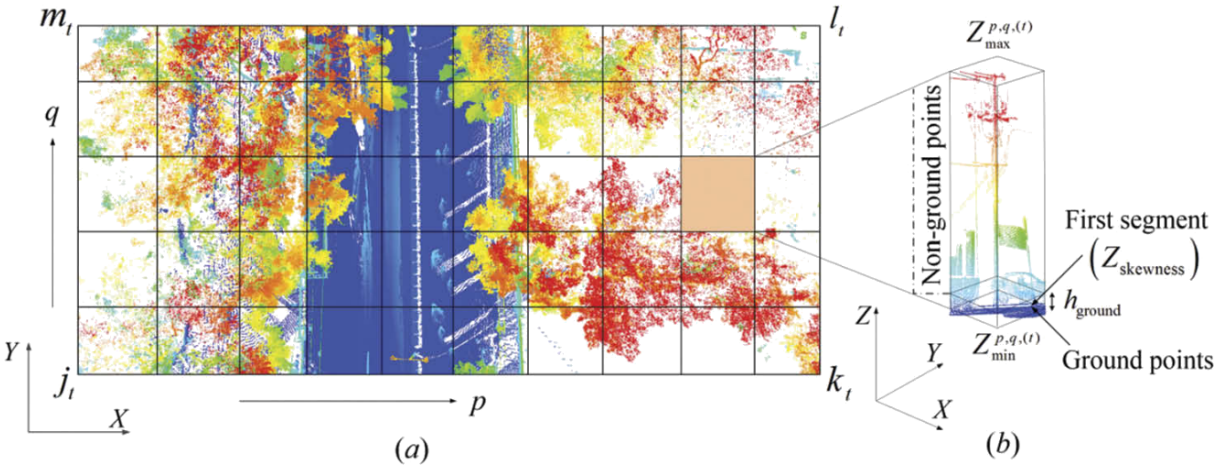

FIGURE 13 – Présentation d'une approche de segmentation automatique des points appartenant au sol dans un nuage de points. En *(a)*, le nuage de points est découpé selon une grille horizontale et en *(b)*, dans chaque région la répartition des points selon leur altitude, coordonnée **Z** dans la figure, est calculée pour déterminer les points qui appartiennent au sol. La couleur représente l'altitude des points : rouge pour les plus hauts, bleu pour les plus bas. Figure tirée de [YSL17].

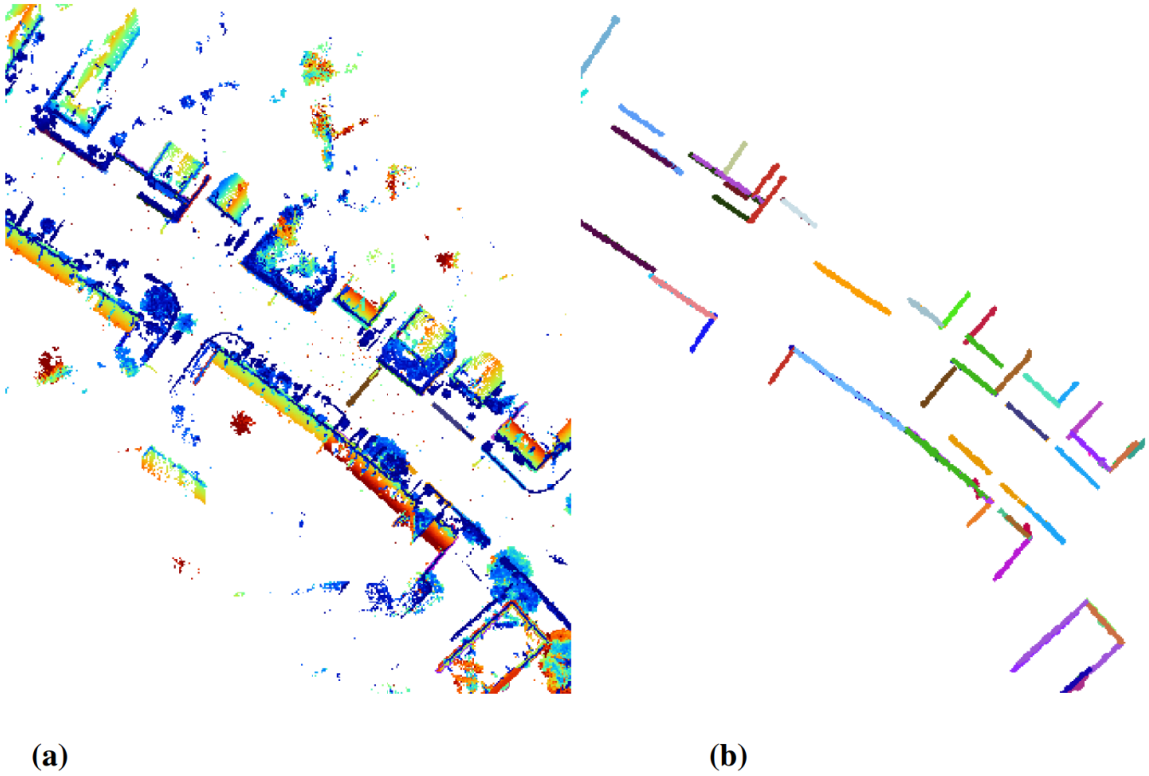

FIGURE 14 – Illustration de la détection de façades d'immeubles dans un nuage de points : en *(a)* le nuage de points original colorié selon l'altitude des points, avec en rouge les points les plus hauts, et en *(b)* les plans verticaux détectés correspondant aux façades d'immeubles. Figure tirée de [JHR11].

Les régions qui contiennent les points dont l'altitude est maximale sont sélectionnées, avec par exemple des méthodes de croissance de régions dont les points graines sont les points les

plus hauts. Ensuite ces régions vont être classifiées afin de conserver uniquement celles qui ressemblent le plus à un plan, comme dans la figure 14 où une méthode de croissance de région est appliquée pour détecter les plans verticaux dans un nuage de points *(b)*. Cette opération peut être effectuée de deux manières différentes selon les méthodes :

- Par l'application d'une méthode de croissance de régions avec une contrainte sur la planéité, puis du calcul d'un seuil de planéité sur la plus grande région obtenue [JHR11].
- Directement par l'application de l'algorithme *RANSAC* [WBF11].

Détection d'objets urbains verticaux Ces objets regroupent aussi bien les arbres que les lampadaires et autres feux de signalisation. En effet du fait de leur forme verticale, les approches visant à détecter ces objets proposent dans un premier temps de détecter tous les objets verticaux puis, dans un deuxième temps, de ne conserver uniquement que les objets qui appartiennent à la classe voulue.

Les méthodes les plus simples de détection d'objets verticaux [YMST16] consistent à classifier chacune des régions en gardant les objets dont l'axe principal forme un angle avec le sol, (la pente) proche de 90°. Ainsi les régions dont la pente est supérieure à un certain seuil sont classées comme appartenant à un objet vertical.

Des méthodes plus robustes proposent une version modifiée des approches par croissance de régions. Parmi celles-ci, [GYLL16] consiste à choisir comme points graines les points d'altitude minimale et à définir les voisins de chaque point **P** comme les points les plus proches qui ont une altitude supérieure au point **P**.

D'autres méthodes [TC15][EHL11] proposent d'utiliser une *analyse en composante principale* [Pea01] puis, à partir des résultats de celle-ci, d'ajuster un cylindre aux points grâce à une *transformée de Hough* ou d'un algorithme *RANSAC*, voir les cylindres en bleu dans la figure 15. Ce qui permet de ne pas être sensible aux choix des points graines.

A partir de la détection d'objets verticaux, il existe plusieurs méthodes permettant de classifier plus finement chacun de ces objets. Parmi ces méthodes, les plus utilisées sont celles qui permettent de catégoriser les objets verticaux en deux classes : arbres et infrastructures routières (feux de signalisation, lampadaires...).

Le critère le plus important de ces méthodes est la différence au niveau de la forme horizontale des arbres par rapport à celles des infrastructures routières. Ainsi certaines méthodes proposent simplement d'appliquer un seuil sur la largeur de la couronne des objets verticaux [WYY+13][LLZL16] : au dessus de celui-ci, l'objet est catégorisé comme un arbre et en dessous l'objet est catégorisé comme une infrastructure routière. D'autres seuils peuvent être définis, comme par exemple sur l'écart-type des composantes x et y des points [ZCX+17].

Véhicules Le cas des véhicules est le plus compliqué car leur forme ne peut pas être approchée par une primitive tel qu'un cylindre ou un plan. Les méthodes de détection de véhicules [XVSP16]

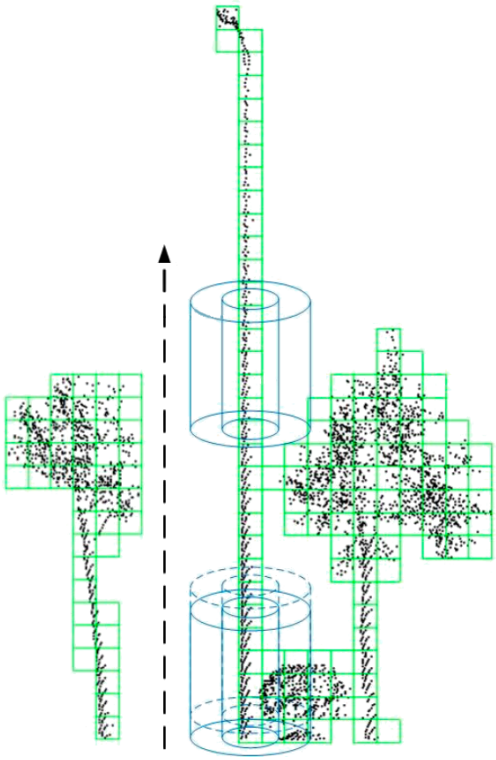

FIGURE 15 – Illustration de la détection d'objets verticaux dans un nuage de points. L'utilisation d'une analyse par composante principale permet d'extraire toutes les régions dont la composante principale est verticale. Ensuite la modélisation par un cylindre (en bleu) permet de séparer les lampadaires et poteaux des arbres, pour lesquels un cylindre n'est pas un modèle adéquat. Figure tirée de [EHL11].

s'appuient alors sur des calculs de distance ou de recalage avec des modèles de voitures existants ou bien par le calcul de descripteurs locaux suivi d'un classifieur *SVM* ou d'un algorithme de *forêt aléatoire*.

2.1.4 Bilan

Le point commun à toutes ces méthodes est leur incapacité à détecter plusieurs classes d'objets à la fois. En effet chaque méthode ne permet de détecter qu'une seule classe d'objets bien précise. Ce qui implique qu'il faut appliquer plusieurs méthodes différentes pour être en mesure de détecter tous les objets présents dans un nuage de points 3D. Se pose alors le problème de la gestion des collisions, c'est-à-dire de cas d'objets détectés par des méthodes différentes (correspondant chacune à une classe d'objet différente) pour une même région du nuage de points.

Une autre limite majeure de ces méthodes est leur manque d'adaptabilité par rapport à la diversité géométrique des objets urbains, aussi bien entre les différentes classes d'objets, qu'au sein même de chaque classe. En effet ces méthodes reposent en grande partie sur la définition de seuils ou de caractéristiques estimés soit expérimentalement soit à partir d'un petit nombre d'exemples. Le plus souvent ceux-ci ne sont pas représentatifs de la diversité des objets 3D,

d'autant plus que cette diversité est accrue par les nombreuses occultations présentes dans les nuages de points *MLS*.

Dans le but de développer une méthode robuste permettant de détecter plusieurs classes d'objets, nous avons décidé d'explorer les opportunités offertes par l'apport de l'apprentissage profond.

2.2 Apprentissage profond

2.2.1 Brève introduction aux réseaux de neurones

L'architecture de réseau de neurones la plus simple est celle appelée *perceptron* [Ros58]. Le *perceptron* est un classifieur binaire [1], qui va chercher à déterminer si un vecteur en entrée $\mathbf{X}$ appartient ou non à une classe C. En effet la sortie du *perceptron*, notée Y, vaut 1 si le vecteur $\mathbf{X}$ appartient à la classe C et 0 sinon. Cette sortie Y est calculée par la somme des composantes du vecteur $\mathbf{X}$ en entrée, pondérées par les poids W du *perceptron* :

$$Y = \sum_i W_i X_i \tag{2.1}$$

où les X_i sont les composantes du vecteur X en entrée et W_i les poids du *perceptron*.

Afin d'assurer la tâche de classification binaire, une fonction d'activation $\mathscr{H}$ est ensuite appliquée sur la sortie Y :

$$\mathscr{H} : \mathbb{R} \to \{0, 1\}$$
$$\mathscr{H}(Y) = \begin{cases} 1 \text{ si } Y \geq 0 \\ 0 \text{ si } Y < 0 \end{cases}$$

Il est alors possible d'entraîner de manière supervisée ce *perceptron*. En prenant le *résultat attendu* pour l'entrée $\mathbf{X}$, aussi appelée *label* et noté $\hat{Y}$, il est possible de mettre à jour les poids du perceptron pour que celui-ci renvoie une valeur plus proche de $\hat{Y}$ lors de sa prochaine application sur le vecteur $\mathbf{X}$. Nous définissons la mise à jour des poids W_i ainsi :

$$W_{i,t+1} \leftarrow W_{i,t} - \eta \times (\hat{Y} - Y) \times X_i \tag{2.2}$$

où $W_{i,t}$ est le poids W_i lors de la $t$ième application et $\eta \in [0, 1]$ est le taux d'apprentissage choisi.

1. Il est possible d'entraîner un perceptron sur des tâches plus compliquées, néanmoins, par souci de clarté, nous évoquerons ici uniquement le cas de la classification binaire.

2.2.2 Réseau de neurones profond

L'idée derrière les réseaux de neurones profonds est d'ajouter plusieurs couches de calcul entre l'entrée et la sortie du réseau, contrairement au *perceptron* où il n'y en a qu'une seule.

Couches cachées

Ces couches intermédiaires sont appelées les *couches cachées*. Ainsi, la fonction calculée par un réseau de neurones profond, notée $\mathscr{F}$, est la composée des fonctions calculées par chacune des *couches cachées* du réseau. Ainsi nous notons $\mathscr{F}^l$ la fonction calculée par la $l$ième couche du réseau. La dernière couche du réseau a comme particularité de calculer la *fonction de coût* du réseau, notée $\mathscr{L}$. Plusieurs types de fonctions de coût existent et leur utilisation dépend du problème adressé par le réseau de neurones. La sortie Y du réseau de neurones avec un vecteur en entrée $\mathbf{X}$ se définit alors comme ceci :

$$Y = \mathscr{L}(\mathscr{F}^{l-1}(...\mathscr{F}^1(\mathbf{X})...) \tag{2.3}$$

Comme dans le cas du *perceptron*, il est possible de mettre à jour les poids du réseau de neurones en fonction de la *sortie attendue*, notée $\hat{Y}$, pour le vecteur en entrée $\mathbf{X}$. Cependant, comme les réseaux de neurones profonds comptent beaucoup plus de poids W que les perceptrons, à cause des *couches cachées*, une grande quantité de données est nécessaire pour entraîner un réseau de neurones profond.

Descente de gradient stochastique

La méthode d'apprentissage des réseaux de neurones profonds est la descente de gradient. Celle-ci consiste à estimer, à chaque étape de l'entraînement, le gradient de la fonction de coût appliquée à l'ensemble des données en entrée $\mathbf{X}$, noté $\nabla\mathscr{L}(\mathbf{X}, \hat{\mathbf{Y}})$ où $\hat{\mathbf{Y}}$ sont les labels associés à l'entrée $\mathbf{X}$. Les poids W sont ensuite mis à jour par rapport à la valeur de ce gradient. Ce qui donne, dans sa forme la plus simple :

$$W_{t+1} \leftarrow W_t - \eta\nabla\mathscr{L}(\mathbf{X}, \hat{\mathbf{Y}}) \tag{2.4}$$

où η est le taux d'apprentissage et W_t représente les poids du réseau à la $t$ième étape d'entraînement. Cependant du fait de la grande taille des jeux données, le calcul du gradient sur l'ensemble des données devient trop coûteux. Une méthode itérative lui est préférée, appelée *descente stochastique du gradient (stochastic gradient descent ou SGD).*

Celle-ci consiste, à chaque itération, à calculer le gradient de la fonction de coût $\mathscr{L}$ appliquée uniquement sur un échantillon $\mathbf{X}^*$ du jeu de données initial. La taille de ce sous-échantillon (*batch size*) notée N_B est un hyper-paramètre de la méthode. Cette méthode permet ainsi de mettre à

jour les poids W du réseau seulement à partir de ce gradient partiel, défini ainsi :

$$W_{t+1} \leftarrow W_t - \frac{\eta}{N_B} \sum_{\mathbf{X}_i \in \mathbf{X}^*} \nabla \mathscr{L}(\mathbf{X}_i, \hat{Y}_i) \tag{2.5}$$

où η est le taux d'apprentissage, les $\mathbf{X}_i$ sont les vecteurs qui appartiennent au sous-ensemble $\mathbf{X}^*$ de taille N_B et W_t représente les poids du réseau à la $t$ième étape d'entraînement. Bien que cette approche résout le problème de la taille du jeu de données en entrée, un autre problème demeure : le calcul du gradient d'une fonction qui est la composée de plusieurs autres fonctions (Eq 2.3) avec chacune ses propres poids à apprendre.

Rétropropagation du gradient

Afin de mettre à jour les poids d'une couche spécifique l du réseau, il faut alors calculer le gradient (voir équation 2.4) de la fonction de coût $\mathscr{L}$ par rapport aux poids de cette couche l uniquement. Ce gradient est noté :

$$\nabla_{W^l} \mathscr{L} = \frac{\partial \mathscr{L}}{\partial W^l} \tag{2.6}$$

Comme la fonction de coût $\mathscr{L}$ (Eq 2.3) est une composée de fonctions, nous pouvons réécrire l'équation précédente :

$$\frac{\partial \mathscr{L}}{\partial W^l} = \frac{\partial \mathscr{L}}{\partial X_S^l} \frac{\partial X_S^l}{\partial W^l} \tag{2.7}$$

où X_S^l est la sortie de la $l$ième couche. Nous observons que le second terme dépend uniquement de la fonction calculée par la $l$ième couche, notée $\mathscr{F}^l$. Dans le cas où $\mathscr{F}^l$ est une fonction dérivable, on peut alors réécrire le second terme de l'équation 2.7 : $\mathscr{F}'(X_E^l)$, où $\mathscr{F}'$ est la dérivée de la fonction $\mathscr{F}$ et X_E^l est l'entrée de la $l$ième couche.

Le premier terme de l'équation 2.7 est plus compliqué à déterminer. Dans le cas où l est la dernière couche du réseau, alors il s'agit simplement de la dérivée de la fonction de coût $\mathscr{L}$ (Eq 2.3) appliquée à la sortie du réseau. Dans le cas contraire, il faut prendre en compte toutes les couches qui succèdent à la $l$ième couche. Ainsi il est possible d'arriver à une définition récursive de ce terme :

$$\frac{\partial \mathscr{L}}{\partial X_S^l} = \sum_{i>l} \left(\frac{\partial \mathscr{L}}{\partial X_S^i} \frac{\partial X_S^i}{\partial X_E^l} W^i \right) \tag{2.8}$$

Nous notons ce terme, E^l, pour l'erreur à la couche l et en l'injectant, ainsi que le second terme de l'équation 2.7, dans l'équation 2.6, nous obtenons la définition suivante :

$$\nabla_{W^l}\mathscr{L} = E^l \mathscr{F}'(X_E^l) \tag{2.9}$$

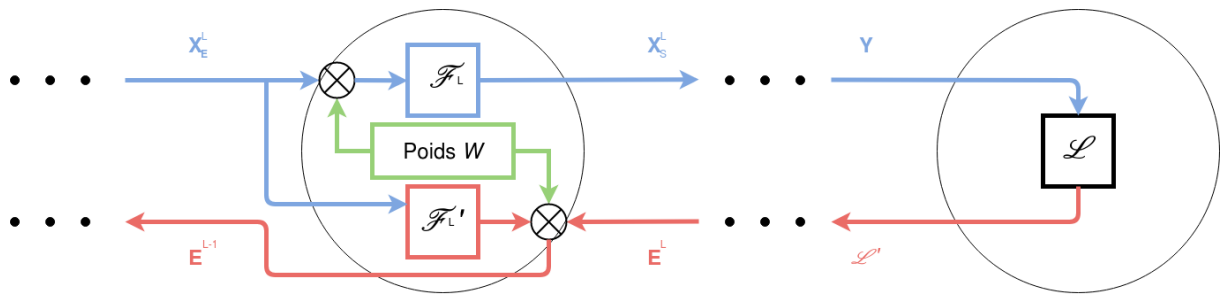

FIGURE 16 – Illustration de l'apprentissage des poids d'un réseau par rétro-propagation des gradients. Le sens usuel du réseau (*forward*) est représenté en bleu, avec à la fin la sortie du réseau notée Y. Le sens de rétro-propagation (*backward*) est représentée en rouge et débute lors du calcul de la fonction de coût, notée $\mathscr{L}$. Pour chaque couche L, les poids W et sa fonction $\mathscr{F}_{\mathscr{L}}$ sont appliquées sur l'entrée X_E^L afin de calculer la sortie X_S^L. Lors du calcul des gradients, la couche prend en entrée l'erreur E^L pour calculer la multiplication avec ses poids W et la dérivée de sa fonction, notée $\mathscr{F}_{\mathscr{L}}'$. Ce calcul sert d'une part à mettre à jour la valeur des poids W et d'autre part à calculer E^{L-1}, nécessaire à la mise à jour des poids des couches précédentes.

Il est alors possible de calculer efficacement les gradients de chaque couche en partant de la dernière sans avoir à effectuer des calculs de dérivées redondants [KEL60][RHW86]. Ce calcul revient à effectuer le parcours du réseau en sens inverse et à propager à chaque couche l'erreur E^L vers les couches précédentes. Ce sens de parcours du réseau est représenté en rouge dans la figure 16. De plus ces calculs peuvent être représentés facilement par des opérations matricielles, augmentant ainsi l'efficacité de ces derniers. A noter que cet algorithme nécessite de calculer au préalable la dérivée des fonctions $\mathscr{F}$ calculées par chacune des couches du réseau.

Afin d'améliorer la mise à jour des poids, vitesse et convergence de l'apprentissage, d'autres méthodes de calcul des gradients ont été proposées. Celles-ci sont appelées des optimisateurs de la descente de gradient, comme par exemple l'optimisateur ADAM [KB15] qui est le plus utilisé en vision par ordinateur.

2.2.3 Réseau de neurones convolutifs

Chaque couche d'un réseau de neurones est chargée de calculer une fonction bien spécifique. L'introduction de couches calculant une opération de convolution entre le signal en entrée et les poids du réseaux de neurones a permis de changer la donne. En effet, ces couches de calcul, appelées *couches de convolution*, permettent de s'affranchir de la définition de *vecteurs descripteurs* et d'extraire automatiquement des caractéristiques à partir des données brutes.

Les couches de convolution comme extracteur de caractéristiques

Néanmoins cette amélioration est possible uniquement dans le cas où les données brutes peuvent être écrites sous la forme de matrices nD (généralisation des matrices $2D$). De plus, l'application de convolutions suppose l'existence d'une corrélation entre les éléments voisins dans les matrices *n-d*. Si aucune corrélation de la sorte n'est présente dans les données brutes, les caractéristiques extraites par les couches de convolution ne peuvent pas être généralisées à des données absentes du jeu d'entraînement. Par exemple il est possible d'appliquer une couche de convolutions sur des images car celles-ci sont représentables sous la forme de matrices 2D : la continuité spatiale des informations contenues dans une image assure un certain niveau de corrélation entre chaque pixel et ses voisins. L'application d'une convolution de noyau W sur une image I se définit alors comme ceci :

$$(W * I)(x, y) = \sum_{i=-w_1}^{w_1} \sum_{j=-w_2}^{w_2} W(i + w_1, j + w_2) I(x + i, y + j) \tag{2.10}$$

où W est le noyau ou filtre de convolution utilisé qui est de dimension $(2w_1 + 1, 2w_2 + 1)$. On obtient alors la *carte de caractéristiques* F, de dimension identique [2] à I. Chaque élément $F_{i,j}$ de F correspond à l'application de W à I, à la même position (i, j) :

$$\mathbf{F}_{i,j} = (W * I)(i, j)$$

Le nombre de poids appris par chaque couches de convolution dépend alors uniquement de la taille des noyaux W et de leur nombre, ce qui permet de réduire considérablement le nombre de poids à apprendre comparativement au nombre de neurones classiques. Ainsi celui-ci est de $((2w_1+1) \times (2w_2+1))$ avec dans la plupart des cas $w_1 = w_2 = 1$, alors que dans le cas de neurones classiques, le nombre de poids à apprendre est fonction du nombre de pixels dans l'image.

L'opération d'agrégation de caractéristiques

Une autre opération importante dans le cas des images 2D est *l'agrégation*. Le rôle de cette opération est d'agréger localement les vecteurs de la *carte de caractéristiques* F en chaque endroit afin de permettre aux couches de convolution suivantes d'extraire des caractéristiques de plus haut niveau. Le plus souvent cette agrégation consiste à extraire les caractéristiques maximales mais il est aussi possible de calculer la moyenne des caractéristiques à la place. L'application de ces couches *d'agrégation* sur une *carte de caractéristiques* F se définit par rapport à deux paramètres : la taille $(h \times l)$ et le pas p. La sortie d'une couche d'agrégation, notée MP, est de dimension inférieure à l'image en entrée : $(H_{MP} = \frac{H-h}{p} + 1, L_{MP} = \frac{L-l}{p} + 1)$.

2. Pour les pixels en bordure de l'image, des voisins de valeur 0 sont extrapolés, on parle de *padding*

L'application d'une opération *d'agrégation maximale MP* sur une *carte de caractéristiques F* à la position (x, y) est définie comme ceci :

$$MP_{(x,y)}(F) = \max_{i \in [x - \lfloor \frac{l}{2} \rfloor, x + \lfloor \frac{l}{2} \rfloor], j \in [y - \lfloor \frac{h}{2} \rfloor, y + \lfloor \frac{h}{2} \rfloor]} F(i, j) \tag{2.11}$$

Application des *réseaux de neurones convolutifs* à la classification d'images

A partir des deux opérations précédemment exposées, *l'agrégation MP* (Eq 2.11) et la *convolution* (Eq 2.10), l'architecture *AlexNet* [KSH17] est développée en 2012 et remporte le plus grand challenge de classification d'images *ILSVRC* [RDS+15] avec une marge conséquente. Cette architecture est composée de 8 couches en tout : 5 *couches de convolutions* (Eq 2.10), les 4 premières étant suivies chacune par une couche *d'agrégation MP* (Eq 2.11), et 3 *couches entièrement connectées* (*fully connected ou FC*) très similaires aux *perceptrons* (Eq 2.1). La dernière couche sert de *classifieur*. Le réseau est entraîné grâce à une fonction de coût appelée *entropie croisée* et définie comme ceci :

$$\mathscr{L}_{Cls}(\mathbf{Y}, \hat{Y}) = -\sum_{C=1}^{NC} y_C \log(\mathbf{Y}_C) \tag{2.12}$$

où $\mathbf{Y}$ est le vecteur en sortie de la dernière *couche entièrement connectée*, $\hat{Y}$ correspond au label, N_C est le nombre de classe possible et y_C est un indicateur binaire qui vaut 1 si la classe $C = \hat{Y}$ et 0 sinon.

Suite au succès de l'architecture *AlexNet* [KSH17], de nombreux *réseaux de neurones convolutifs* ont été proposés parmi lesquels :

- *ZF-Net*, [ZF14] montre que réduire la taille des filtres utilisés dans les couches de convolutions permet d'obtenir de meilleurs résultats sur la tâche de classification d'images.

- Les architectures *VGGs* [SZ15] montrent l'importance de la profondeur des *réseaux de neurones convolutifs* en utilisant 3 architectures comptant 11, 16 et 19 couches obtenant des scores croissant selon le nombre de couches. Elles mettent aussi en évidence les limites de l'augmentation du nombre de couches à partir d'une certaine profondeur dû au fait que *l'erreur résiduelle de chaque couche* (Eq 2.8) devient alors négligeable, problème dit du *'vanishing gradient'* [KK10].

- Le réseau *NiN(network in network)* [LCY14] montre que l'utilisation de couches de convolution avec une *taille de filtre de 1×1*, intercalées entre les couches de convolution *classiques* permet d'augmenter la non linéarité du réseau et permet d'ajouter plus de couches de convolution *classiques* sans rencontrer de problème de gradient.

– L'architecture *GoogLeNet* [SLJ+15] introduit les *modules inception* permettant d'augmenter non pas la profondeur du réseau mais sa largeur. En effet ces modules appliquent plusieurs opérations sur la même carte de caractéristiques puis les différents résultats obtenus sont concaténés ensemble.

– Enfin *ResNet* [HZRS16] et *DenseNet* [HLVW17] contournent le problème du *'vanishing gradient'* en combinant les caractéristiques extraites par chaque *couche de convolution* à celles extraites par les couches précédentes, soit par une somme, soit directement par une concaténation.

Application des *réseaux de neurones convolutifs* à la segmentation sémantique d'images

Une autre tâche sur laquelle peuvent être entraînés les *réseaux de neurones convolutifs* est la segmentation sémantique d'image. Celle-ci consiste à prédire une classe pour chaque pixel (x, y) d'une image I. Une première manière naïve de parvenir à ce résultat est de partir d'une architecture de classification d'images, par exemple *AlexNet* [KSH17], d'en retirer les *couches d'agrégation* et de remplacer les *couches entièrement connectées* par des *couches de convolution* avec des filtres de taille (1×1). C'est l'idée de l'architecture *FCN (fully convolutional network)* [SLD17]. Ainsi au lieu de s'entraîner sur une fonction d'entropie croisée classique (Eq 2.12), la fonction de coût utilisée est une fonction d'entropie croisée appliquée à chacun des pixels de l'image :

$$\mathscr{L}_{SemSeg}(\mathbf{Y}, \hat{Y}) = \sum_{i}^{H \times L} \frac{1}{H \times L} \sum_{C}^{N_C} y_{i,C} \log(\mathbf{Y}_{i,C}) \tag{2.13}$$

où $\mathbf{Y}_i$ est la sortie du réseau pour le $i$ème pixel de l'image I, $\hat{Y}_i$ son label associé et $y_{i,C}$ est un indicateur binaire qui vaut 1 si la classe $C = \hat{Y}_i$ et 0 sinon.

Cette idée est reprise dans les réseaux *U-Net* [WZ21] et *SegNet* [BKC17] qui introduisent les premières *architectures en sablier* pour la segmentation sémantique d'images. Ceux-ci introduisent l'opération de *déconvolution* permettant à partir d'une *carte de caractéristiques* de prédire une nouvelle *carte de caractéristiques* de dimension supérieure. L'architecture qui en résulte est alors similaire aux *auto-encoders* [LCLL14] avec en plus une concaténation des caractéristiques de la partie encodeur du réseau avec celles de la partie décodeur.

Au delà de la *classification* et de la *segmentation sémantique*, les *réseaux de neurones convolutifs* permettent de s'attaquer à une tâche beaucoup plus proche de notre problème : la détection d'objets dans des images.

2.3 Détection d'objets dans des images 2D

2.3.1 Les approches par classification de régions

La détection d'objets 2D est la tâche qui à partir d'une image I, de dimension $L \times H$, va chercher à prédire des boîtes englobantes autour des objets contenus dans l'image I. Ces boîtes englobantes, notées $\mathbf{B}_{2D}$, sont définies par leur taille $l \times h$, la position de leur centre (x, y) dans l'image I et par la classe de l'objet qu'elles contiennent C :

$$f^{Detec} : \begin{cases} \mathbb{R}^{H \times L} \rightarrow \left[\mathbb{R}^4 \times \{1, ...N_C\}\right]^* \\ I \rightarrow \{\mathbf{B}_{2D_i}, C_i\}_{i=1}^* = f^{Detec}(I) \end{cases} \tag{2.14}$$

avec $\mathbf{B}_{2D_i} \in \mathbb{R}^4$ les coordonnées définissant la position de la $i$ème boîte dans le repère 2D et C_i le label attribué à cette boîte. On notera N_b le nombre de boîtes détectées.

Comme les *réseaux de neurones convolutifs* ne sont capables de reconnaître qu'un seul objet à la fois, les premières méthodes de détection d'objets reposent sur une approche en deux étapes :

- une première étape consiste à extraire des régions R de l'image I, celles-ci sont définies comme l'ensemble des pixels de l'image I compris dans la boîte englobante $\mathbf{B}_{2D}$ de centre (i, j) et de dimension $l \times h$:

$$R_{2D_B} = \left\{ I(x, y) \middle| x \in \left[i - \frac{h}{2}; i + \frac{h}{2}\right], y \in \left[j - \frac{l}{2}; j + \frac{l}{2}\right] \right\} \tag{2.15}$$

- La deuxième étape consiste à utiliser un *réseau de neurones convolutifs* qui prend en entrée chaque région R_{2D} et leur associe une classe C qui peut être soit la classe de l'objet contenu dans la région R_{2D} : $C \in \{1, ..., N_C\}$, où N_C est le nombre de classes possibles, soit 0 si la région R_{2D} ne contient pas d'objet.

Le résultat de la détection d'objets est alors l'ensemble des boîtes englobantes $\mathbf{B}_{2D}$ (Eq 2.14) qui délimitent les régions R_{2D}, ainsi que les classes C associées à ces régions : $\{(R_1, C_1), ...(R_b, C_b)\}$.

Fenêtres glissantes

Historiquement, les premières méthodes suivant l'approche par classification de régions sont les algorithmes de fenêtre glissante. L'idée est d'extraire d'une image I toutes les régions R_{2D} (Eq 2.15) possibles en termes de position (x, y) et de dimension $l \times h$, puis de les classifier une par une.

Cependant, du fait du nombre de régions R_{2D} possibles dans une image, d'un facteur $2^{L \times H}$, la complexité algorithmique d'un tel calcul est bien trop grande. A la place, nous prenons une fenêtre glissante qui se déplace dans l'image, d'un pas p et selon le sens indiqué par les flèches dans la figure 17. Pour chaque emplacement (x, y) de cette fenêtre glissante, un nombre prédéfini

N_R de régions R_{2D} de dimensions différentes les unes des autres sont extraites. Le nombre total de régions extraites est alors le nombre N_R de régions extraites à chaque emplacement multiplié par le nombre d'emplacements de la fenêtre glissante $\frac{H \times L}{p}$.

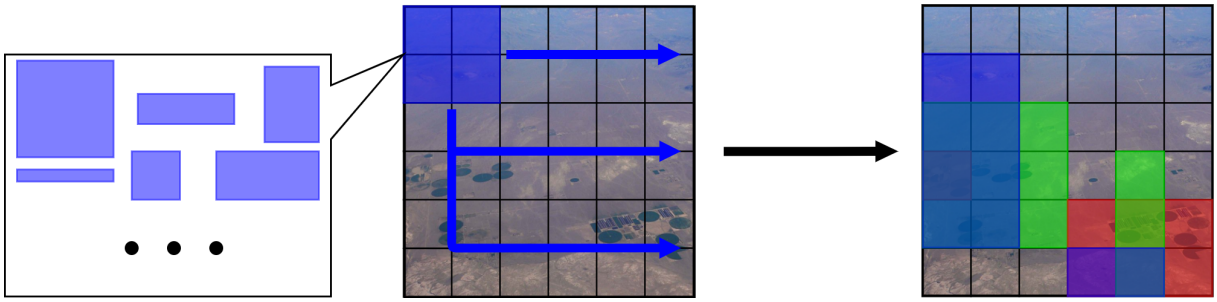

FIGURE 17 – Application d'une méthode de détection par fenêtre glissante. A gauche, les différentes formes de régions traitées et à droite une illustration du problème de recouvrement des régions détectées : plusieurs régions différentes recouvrent le même objet, comment faire pour ne conserver uniquement celle qui propose la meilleure approximation de la forme de l'objet ?

Un *réseau de neurones convolutif* est ensuite appliqué sur chacune des régions R_{2D} pour prédire sa classe et ainsi obtenir l'ensemble des boîtes englobantes $\mathbf{B}_{2D}$ (Eq 2.14). Cette approche peut être généralisée en 3D par les 'boîtes glissantes' [QSMG17].

Le principal problème des algorithmes de fenêtre glissante provient du fait qu'un réseau de neurones doit être appliqué pour chaque région R_{2D}, ce qui est très couteux en termes de calcul. De plus, selon la valeur du pas p, le recouvrement des boîtes englobantes $\mathbf{B}_{2D}$, c'est-à-dire le nombre de boîtes englobantes définies autour du même objet, sera plus ou moins important : plus le pas p est grand, moins le recouvrement est important et inversement. Nous illustrons ce problème dans la figure 17 où nous pouvons noter que, à droite de la figure, pour chaque objet plusieurs boîtes englobantes sont proposées : une rouge, une verte et une bleue. Pour résoudre le problème du recouvrement il faut conserver uniquement la boîte qui décrit le mieux l'objet aussi bien au niveau de sa classe que de sa forme. Par exemple, il est possible, pour plusieurs boîtes $\mathbf{B}_{2D}$ englobant le même objet O de garder uniquement celle dont le *score* prédit par le classifieur du *réseau de neurones convolutif* utilisé est le plus important. Néanmoins, cela ne permet pas de résoudre le cas où les boîtes $\mathbf{B}_{2D}$ n'englobent chacune qu'une partie de l'objet O et où calculer l'union des boîtes $\mathbf{B}_{2D}$ serait plus intéressant. Ainsi d'autres méthodes existent afin de fusionner les boîtes englobantes $\mathbf{B}_{2D}$ [SEZ+14].

Sélection de régions d'intérêt

C'est dans l'optique de résoudre ces problématiques qu'est développé le premier *réseau de neurones convolutifs* avec approche par régions (*region-based convolutional Neural Network*, noté *RCNN* dans la suite) en 2013 [GDDM14]. Celui-ci a pour nouveauté de ne plus chercher à extraire toutes les régions R_{2D} possibles mais uniquement celles, appelées régions d'intérêt (*re-*

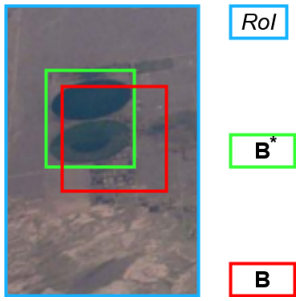

FIGURE 18 – Détails de la prédiction des boîtes englobantes par le réseau *RCNN* : en bleu la région d'intérêt, *RoI* en entrée du second module, en vert la boîte vérité terrain. $\mathbf{B}^*$ et en rouge la boîte **B** prédite par le réseau. La fonction d'apprentissage doit alors permettre au réseau de détecter les objets indépendamment de la taille de la région ; c'est pourquoi les dimensions de la taille prédite par le réseau doivent être normalisées par la dimension de la *RoI* lors du calcul de la fonction de coût.

gion of interest, noté RoI dans la suite), qui sont susceptibles de contenir un objet. L'architecture du $RCNN$ est alors divisé en deux modules :

- Un premier module pour l'extraction des RoI à partir de l'image en entrée I.
- Un second module pour la prédiction de boîtes englobantes $\mathbf{B}_{2D}$ à partir de ces RoI.

Premier module Le schéma du premier module est présenté dans la figure 19 (a). Ce module prend en entrée une image I et lui applique un algorithme de recherche sélective (selective research [UVGS13]), en vert dans le schéma. Cet algorithme, similaire aux approches par croissance de régions, cherche à combiner récursivement les pixels voisins dans des *régions* en fonction de leurs similarités en termes de couleur et de texture. Il permet ainsi d'extraire un nombre fixe N_R de RoI. Ces RoI sont ensuite passées en entrée d'un *réseau de neurones convolutifs*, en rose dans le schéma, qui va les classifier en deux catégories : les 'vraies' RoI, c'est-à-dire celles qui contiennent un objet, en orange dans le schéma, et les régions ne contenant pas d'objet, en jaune dans le schéma. Ainsi le *réseau de neurones convolutif* va prédire pour chaque RoI un label o qui vaut 1 si elle contient un objet et 0 sinon.

Second module Le schéma du second module est présenté dans la figure 19 (b). Ce module prend en entrée uniquement les RoI qui contiennent un objet, c'est-à-dire les RoI avec un *label* associé $o = 1$. Chaque *RoI* est passée en entrée d'un second *réseau de neurones convolutif* en bleu dans le schéma. Ce *réseau de neurones convolutif* calcule 2 sorties pour chaque *RoI* : **1** une *classification*, en rose, pour déterminer la classe C de l'objet contenu dans la *RoI* et **2** une *régression*, en rouge, afin d'ajuster une boîte englobante $\mathbf{B}_{2D}$ autour de l'objet contenu dans la *RoI*. La sortie de l'opération de *régression* est un vecteur caractérisant la boîte englobante $\mathbf{B}_{2D} = (x, y, l, h)$ prédite par le réseau autour de l'objet O.

L'opération de régression est entraînée par une fonction de coût $\mathscr{L}_{reg}$ définie par rapport à la taille de la boite englobante prédite par le réseau, notée $\mathbf{B}_{2D}$, la boîte englobante qui correspond

à la vérité terrain [3] autour de l'objet O, notée $\mathbf{B}^*_{2D} = (x^*, y^*, l^*, h^*)$, et par rapport à la boîte englobante qui définit les dimensions de la *RoI* en entrée, notée $\hat{\mathbf{B}}_{2D} = (\hat{x}, \hat{y}, \hat{l}, \hat{h})$, comme détaillé dans la figure 18. Cette fonction de coût correspond à la différence entre les dimensions de la boîte englobante prédite par le réseau $\mathbf{B}_{2D}$ et celles de la boîte englobante vérité terrain $\mathbf{B}^*_{2D}$, normalisée par les dimensions de la *RoI*. Elle est définie comme ceci :

$$\mathscr{L}_{reg}(\hat{\mathbf{B}}_{2D}, \mathbf{B}^*_{2D}, \mathbf{B}_{2D}) = \left(\frac{x^* - \hat{x}}{\hat{l}} - x\right) + \left(\frac{y^* - \hat{y}}{\hat{h}} - y\right) + \left(\log\left(\frac{l^*}{\hat{l}}\right) - l\right) + \left(\log\left(\frac{h^*}{\hat{h}}\right) - h\right) \tag{2.16}$$

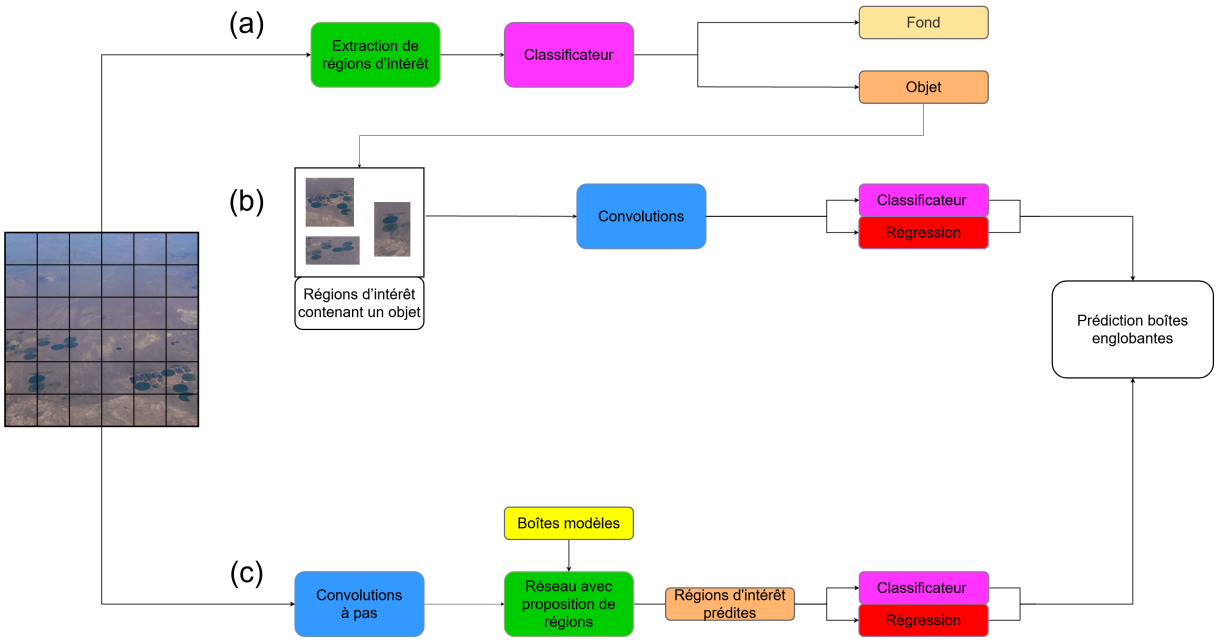

FIGURE 19 – Schéma des deux architectures de réseaux de neurones par régions. En haut une illustration de l'apprentissage en deux temps d'un *RCNN* et en bas l'apprentissage direct d'un *RPN*.

Contrairement aux méthodes par fenêtres glissantes, le *RCNN* permet de se concentrer uniquement sur les *RoI* au lieu de parcourir l'image en entier. Néanmoins, comme le second module a besoin des prédictions du premier module, les deux modules doivent être entraînés l'un après l'autre. De plus la prédiction de boîtes englobantes dépend entièrement de l'algorithme de recherche sélective du premier module. Si celui-ci manque un objet dans l'image, alors il n'y a pas de possibilité pour le second module de rattraper l'erreur. Or, l'algorithme de recherche sélective agit indépendamment des fonctions de coût des *réseaux de neurones convolutifs* du premier et du second module. Ainsi, les *RoI* demeurent identiques pendant tout le long de l'entraînement, ce qui réduit la robustesse du *RCNN*.

3. Le terme de *vérité terrain* est employé pour décrire indistinctement des annotations réalisées par une personne sur une image ou un nuage de points

2.3.2 Apprentissage automatique des *RoI*

En 2015, le premier réseau à proposition de régions (*region proposal network*, noté RPN dans la suite) [RHGS17] est développé. Il permet de s'affranchir de la contrainte de l'entraînement séparé en deux temps et d'optimiser, en même temps, la prédiction des *RoI* et des boîtes englobantes. Cette amélioration est rendue possible par l'utilisation de *convolution à pas* (*strided convolutions*) à la place des convolutions standards.

Le principe des *convolutions à pas*

Dans le cas des *convolutions* classiques (Eq 2.10) la *carte de caractéristiques* obtenue, notée F, est de dimension identique à I. Contrairement à celles-ci, les *convolutions à pas* reviennent à utiliser un pas p supérieur à 1, comme si le filtre W enjambait certaines positions de l'image I. L'une des conséquences immédiates est que la *carte de caractéristiques* F en sortie est de dimensions inférieures à celles de l'image I. C'est dans le contexte d'apprentissage profond que l'utilisation des *convolutions à pas* a reçu un gain de popularité, car elles sont utilisées à la place des couches *d'agrégation*.

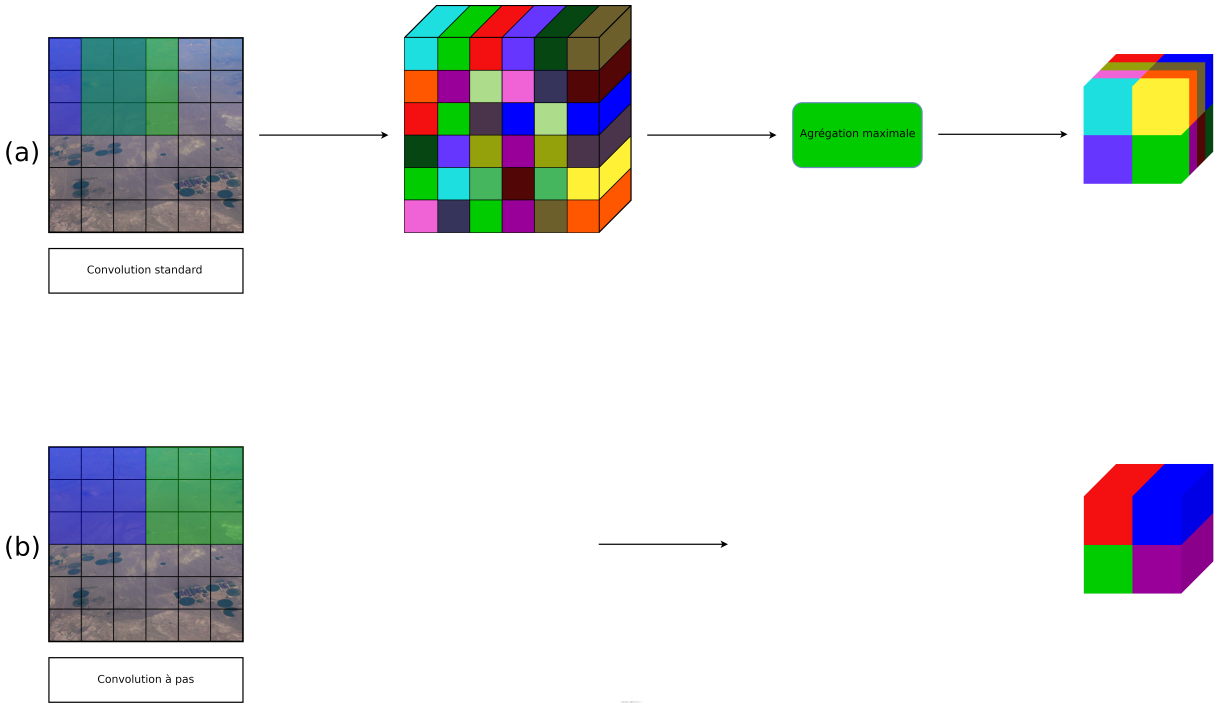

FIGURE 20 – Différence entre convolution classique, en haut, et *convolution avec pas*, en bas, appliquées à une image 2D. Dans le cas des convolutions standards, des caractéristiques sont extraites pour chaque pixel de l'image originale. Il y a donc besoin de l'opération d'*agrégation maximale* (*max pooling*) pour compacter les informations extraites. Dans le cas des *convolutions à pas*, le noyau est appliqué seulement dans quelques positions de l'image, ce qui permet d'obtenir une caractéristique par imagettes, en bleu et vert dans la figure, contrairement aux *convolutions classiques* où uniquement les caractéristiques maximales sont conservées.

En effet, l'utilisation de *convolution classique* suivi d'une couche *d'agrégation*, représentée

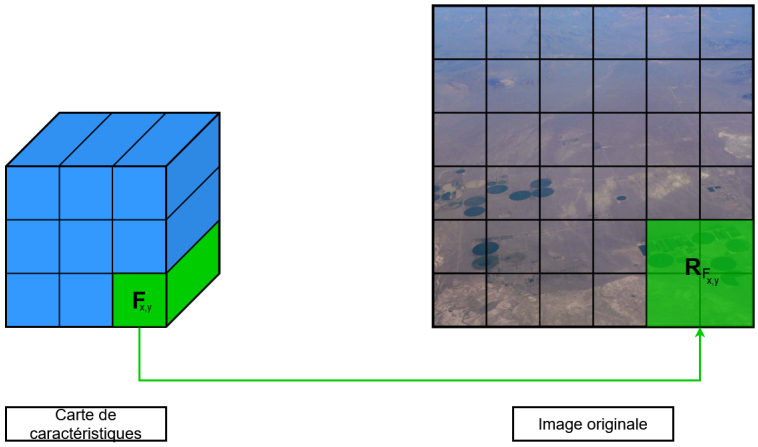

FIGURE 21 – Illustration de la correspondance entre *carte de caractéristiques* et image originale rendue possible grâce à l'utilisation des *convolutions à pas*. C'est par rapport à ces régions correspondantes R_F, appelées aussi *champs de vision* (*receptive field*) des convolutions, que les boîtes modèles $\hat{\mathbf{B}}_{2D}$ sont définies.

dans la figure 20 (*a*), permet de calculer un *vecteur de caractéristiques* pour chaque pixel de l'image puis d'extraire uniquement les composantes maximales de ces vecteurs pour chaque région. Ainsi les éléments de la *carte de caractéristiques* en sortie comportent uniquement les composantes maximales des régions de l'image en entrée. Alors que dans le cas des *convolutions avec pas*, représentées dans la figure 20 (*b*), un *vecteur de caractéristiques* est directement extrait de chaque région et donc chaque élément de la *carte de caractéristiques* correspond au *vecteur de caractéristiques* d'une région de l'image en entrée.

La correspondance entre éléments de la *carte de caractéristiques* F et régions R_{2D} de l'image I, dans le cas des *convolutions à pas* est illustrée dans la figure 21. Ainsi, pour chaque emplacement (x, y) de la *carte de caractéristiques* F, nous notons $\mathbf{F}_{x,y}$ le *vecteur de caractéristiques* défini à cet emplacement, en vert, en bas à gauche de la *carte de caractéristiques* sur la figure 21. Or, comme la *carte de caractéristiques* est obtenue par l'application d'une *convolution à pas* sur l'image I, il existe une région R de l'image I, telle que les caractéristiques extraites de cette région par l'opération de *convolution à pas*, sont regroupées dans le vecteur $\mathbf{F}_{x,y}$. Nous notons alors cette région $R_{F_{x,y}}$, en vert transparent et en bas à gauche de l'image originale dans la figure 21. Nous exprimons cette région $R_{F_{x,y}}$ de l'image originale en fonction du vecteur $\mathbf{F}_{x,y}$ associé en partant de la définition des convolutions 2D (Eq 2.10) :

$$\forall F(x, y), \exists R_{F_{x,y}} \in I \mid \mathbf{F}_{x,y} = \sum_{i,j=1}^{w_1 w_2} W_{i,j} R_{i,j} \tag{2.17}$$

où W est le noyau de convolution de dimension $(w1, w2)$, les $W_{i,j}$ dénombrent les éléments de ce noyau dans l'ordre, $R_{F_{x,y}}$ est la région de l'image I de centre (x, y) et de dimension identique au filtre W et les $R_{i,j}$ dénombrent les éléments de cette région dans l'ordre. Nous pouvons

alors exprimer l'opérateur de *convolution à pas* comme combinaison de deux opérateurs : une convolution classique, extraction automatique de *caractéristiques*, ainsi qu'un lien spatial entre les *caractéristiques* extraites et les régions de l'image I.

Proposition de *RoI*

Le schéma global de fonctionnement des RPN est présenté dans la figure 19 (c). Le RPN prend en entrée l'image I et lui applique une série de *convolutions à pas*, en bleue, afin d'obtenir la *carte de caractéristiques* F.

A partir de cette *carte de caractéristiques* F, le RPN va déterminer des régions R_F (Eq 2.17), en analysant la *carte de caractéristiques* F. L'objectif est de trouver les R_F qui contiennent un objet, appelées *RoI*. Afin de ne pas se limiter à une seule taille pour chaque région R_F, [RHGS17] introduit l'utilisation des *boîtes modèles*. Ces boîtes modèles, notées $\hat{\mathbf{B}}_{2D}$, correspondent à un ensemble de boîte de dimensions et de taille prédéfinies. Ainsi pour chaque élément $\mathbf{F}_{x,y}$ de la *carte de caractéristiques* F, représenté par un carré vert dans la figure 22, k boîtes modèles $\hat{\mathbf{B}}_{2D}$, de dimension respectives $\hat{\mathbf{B}}_{2D} = (\hat{l}, \hat{h}, \hat{x}, \hat{y})$, sont définies. Celles-ci sont définies par rapport aux dimensions de la région $R_{F_{x,y}}$, connectée à l'élément de F dans la figure 22. Le centre $(\hat{x}, \hat{y})$ est égal au centre de la région $R_{F_{x,y}}$, la hauteur $\hat{h}$ et la largeur $\hat{l}$ sont différentes pour chacune des k boîtes modèles. Dans [RHGS17], $k = 9$, boîtes modèles, $\hat{\mathbf{B}}_{2D}$, sont définies et 3 tailles différentes, petite, moyenne et grande, combinées avec les trois $ratio$ (aspect ratio) qui sont : 1 : 1, 1 : 2 et 2 : 1 sont utilisés pour calculer les dimensions $\hat{l} \times \hat{h}$. La figure 22 illustre le fonctionnement des boîtes modèles avec $k = 3$ et où sont donc représentés 3 boîtes modèles de taille différentes, toutes centrées autour de la même région R_F.

Comme une région R_F est associée à chaque élément de la *carte de caractéristiques* F et que pour chaque région, k boîtes modèles sont définies, le nombre total de boîtes modèles $\hat{\mathbf{B}}_{2D}$ est $L \times H \times k$. Comme dans l'architecture du $RCNN$, le RPN ne va pas directement prédire des boîtes englobantes B_{2D} à partir des régions définies par les boîtes modèles $\hat{\mathbf{B}}_{2D}$, notées $\hat{R}_{2D}$, mais va d'abord les filtrer afin d'utiliser uniquement celles qui contiennent un objet. Cette opération est réalisée par le *réseau de neurones convolutif glissant*, représenté en vert dans la figure 19 (c), qui prend en entrée la *carte de caractéristiques* F et prédit pour chaque région $\hat{R}_{2D}$ une classe C, qui peut être soit la classe de l'objet présent dans cette région soit 0 si la région ne contient pas d'objet. Pour cela, au lieu d'un label o comme dans le $RCNN$, un score s est prédit. Ce score s correspond à la probabilité que la région $\hat{R}_{2D}$ contienne un objet. De plus, le *réseau de neurones convolutif glissant* calcule aussi une régression 2D (Eq 2.16) afin d'affiner les dimensions des boîtes modèles $\hat{\mathbf{B}}_{2D}$ autour des objets qu'elles sont susceptibles de contenir. Un nombre fixe, 2000 dans [RHGS17], de régions $\hat{R}_{2D}$ est ensuite extrait parmi celles qui ont les score s les plus élevées : celles-ci sont les RoI prédites par le RPN.

Le RPN va finalement prédire pour chacune de ces RoI une boîte englobante 2D grâce à une dernière couche de *convolution* suivie d'une *régression 2D* (Eq 2.16) et d'un $classifieur$. Comme les RoI sont directement prédites par le réseau, celui-ci peut être entraîné conjointement

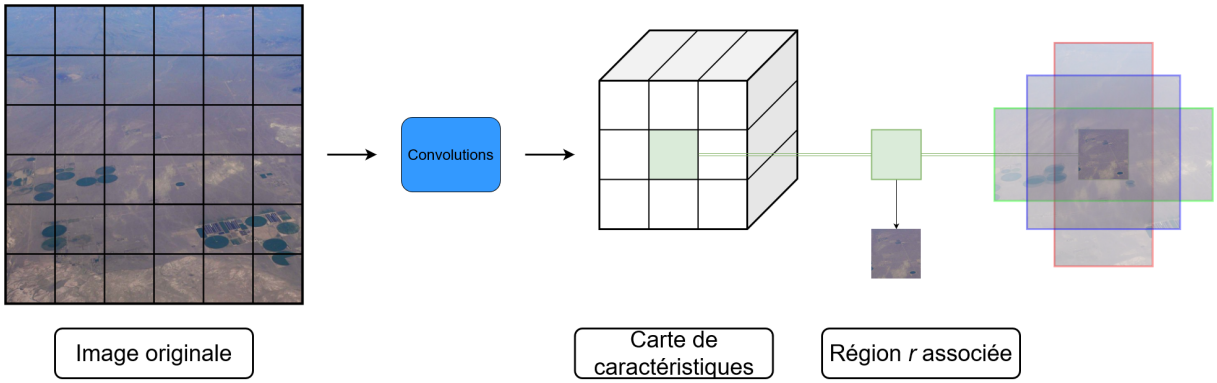

FIGURE 22 – Exemple illustrant le principe des boîtes modèles dans un *RPN*. Le réseau associe à chaque élément $E_{i,j}$, colorié en vert, de la *carte de caractéristiques* un nombre fixe, ici 3, de boîtes modèles. Ces boîtes sont toutes de dimensions différentes et sont centrées sur la région r associée à l'élément $E_{i,j}$

avec la partie détection de boîtes englobantes (*end-to-end learning*), comme représenté dans la figure 19 (*c*). Contrairement au *RCNN*, les performances du *RPN* ne dépendent alors pas de l'application d'un module déconnecté du réseau, ce qui le rend plus robuste. Ainsi, les architectures basées sur ce *RPN* occupent aujourd'hui les premières places des compétitions de détection d'objets dans des images 2D [CFL+15]

Réseau avec une grille de cellules

Les réseaux avec propositions de régions précédemment présentés opèrent en 2 temps :

1. proposition de régions à l'aide des boîtes modèles
2. prédiction de boîtes englobantes à partir de ces régions

Comme ces deux étapes comportent chacune des couches de *convolutions*, leurs applications successives sont coûteuses en termes de temps. Le temps de calcul lors de la phase de test (ou de prédiction) comporte alors un frein pour les applications en temps réel, comme par exemple la détection d'objets dans des flux vidéos [ZZXW19]. Ainsi des réseaux ont alors été proposés afin d'être conformes aux contraintes de la détection d'objets en temps réel.

Le réseau *YOLO* [RDGF16] permet d'unifier l'étape de proposition de régions avec l'étape de prédiction de boîtes englobantes. Pour cela, l'image est divisée selon une grille régulière en un nombre fixe de *cellules*. Chacune de ces cellules est alors *responsable* des objets présents dans celle-ci, c'est-à-dire les objets dont le centre de la boîte englobante appartient à cette cellule. Le schéma d'application est alors le suivant :

1. plusieurs couches de *convolutions à pas* sont appliquées afin d'obtenir une *carte de caractéristiques* de même dimension que la grille régulière

2. pour chaque cellule, un score de *classe* est prédit ainsi qu'un nombre N_B (hyper-paramètre) de boîtes, avec pour chacune de ces boîtes un score représentant l'intersection sur l'union (*IoU*) avec la boîte vérité terrain

3. lors de la phase de test, un score est associé à chaque boîte englobante en combinant leur score *IoU* avec le score de *classe* de la cellule à laquelle la boîte appartient.

Cette approche permet des gains conséquents au niveau du temps de calcul, ce qui rend possible le traitement d'images en temps réel. Cependant la division de l'image en une grille de taille fixe induit des difficultés pour la détection des objets trop petits (ou trop grands) par rapport à la résolution de la grille de cellules.

Le réseau *SSD* [LAE+16] propose de résoudre ce problème en utilisant une grille multi-échelles. L'architecture se base sur le réseau *YOLO* [RDGF16] mais en ajoutant une prédiction de boîtes englobantes (avec un score *IoU* et un score de *classe* pour chaque boîte englobante) à chaque étape de *convolution à pas*. Ainsi le réseau se comporte comme si l'image en entrée était divisée en une grille de cellules de plus en plus grandes au fur et à mesure de l'application des couches de *convolutions à pas*.

Plusieurs améliorations de ces approches ont été proposées, afin d'inclure les boîtes modèles [RF17] ou bien des mécanismes d'extraction de *caractéristiques* multi-échelles [SLL+17, AOH+20].

Bilan

Les *réseaux de neurones convolutifs* ont permis d'apporter un nouveau souffle aux problèmes de reconnaissance de forme à partir de données structurées, notamment sur les problèmes plus compliqués comme la détection d'objets dans des images. De plus, dès 2015, certaines approches ont tenté d'étendre le champ d'application des réseaux de neurones convolutifs aux nuages de points [MS15][CFG+15][SMKLM15] et offrent de fortes potentialités. C'est pourquoi nous avons cherché dans nos travaux à étendre le domaine d'application des réseaux de neurones convolutifs aux nuages de points.

Cependant, comme les avancées permises par les réseaux de neurones convolutifs découlent du calcul de convolution sur les données en entrée, il est impossible des les appliquer en l'état sur des nuages de points. En effet, aucune structure n'est définie sur les nuages de points : il n'y a pas d'ordre sur les points et il n'existe pas de voisinage régulier pour ces points. Un travail de généralisation doit alors être accompli afin de rendre les nuages de points compatibles avec les réseaux de neurones convolutifs ou inversement.

2.4 Comment généraliser les *réseaux de neurones convolutifs* aux nuages de points ?

La généralisation des *réseaux de neurones convolutifs* peut s'effectuer de 2 manières différentes :

- Structurer les nuages de points pour pouvoir leurs appliquer des couches de convolutions.
- Définir une opération qui permet d'accomplir un des rôles des couches de convolutions (l'extraction automatique de caractéristiques) et qui soit applicable aux nuages de points.

2.4.1 Structuration des nuages de points 3D

Les premiers *réseaux de neurones convolutifs* appliquées aux nuages de points 3D [MS15] [SMKLM15] reposent sur une idée simple qui est l'utilisation d'une transformation appliquée aux nuages de points. Cette transformation a pour objectif de prendre en entrée un nuage de points S et d'en sortir une matrice nD, qui a pour propriété d'être une *donnée structurée*, c'est-à-dire une donnée sur laquelle il est possible d'appliquer une opération de convolution (Eq 2.10).

Il est alors possible de classer les différentes méthodes de *structuration* existantes selon la nature de la transformation qu'elles appliquent aux nuages de points.

Discrétiser l'espace géométrique des nuages de points : la *voxelisation*

L'opération de *voxelisation* permet de discrétiser un nuage de points afin de le représenter sous forme d'une grille volumique où une valeur est associée à chaque cellule, appelé *voxel*, en fonction du nombre de points qu'elle contient. Les *voxels* sont la plupart du temps de dimension $(h,\ l,\ p)$ identique et cette dimension, appelée aussi *résolution* est le paramètre principal de l'opération de *voxelisation*.

Par exemple, le réseau *VoxNet* [MS15] propose de *voxeliser* un nuage de points, voir figure 23 (a), avec une résolution de *voxel* de $h{=}l{=}p{=}0.1$ et pour chaque *voxel* $\mathscr{V}_{i,j,k}$ de calculer la densité des points de l'ensemble $S_{i,j,k}$ [Thr03][TA10], où $S_{i,j,k}$ est l'ensemble des points du nuage de points S contenus dans le *voxel* $\mathscr{V}_{i,j,k}$:

$$S_{i,j,k} = \{\mathbf{P} \in S | x \in [i \times h, (i+1) \times h], y \in [j \times l, (j+1) \times l], z \in [k \times p, (k+1) \times p]\}$$

où (x, y, z) sont les coordonnées du point $\mathbf{P}$ et (h, l, p) la résolution du *voxel*. Deux exemples de cartes volumiques ainsi obtenues, aussi appelées *carte d'occupation (occupancy grid)*, sont présentées en figure 23 (b) où un niveau de gris plus sombre représente une densité plus élevée pour chaque *voxel*.

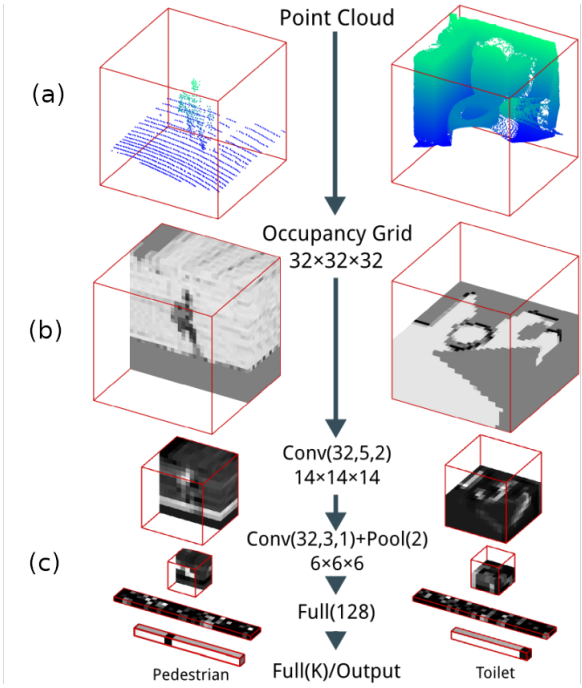

FIGURE 23 – Architecture VoxNet avec de haut en bas : le nuage de points original S, la carte d'occupation, 2 couches de convolutions 3D volumétriques et le classifieur. Figure tirée de [MS15].

Il est alors possible d'adapter des architectures de *réseaux de neurones convolutifs* pour la classification d'images afin d'extraire automatiquement des caractéristiques à partir des voxels de cette image volumique. Il suffit en effet d'utiliser des filtres 3D, noté W_{3D}, pour les *couches de convolution* (Eq 2.10) :

$$(W_{3D} * V)(x, y, z) = \sum_{i=-w_1}^{w_1} \sum_{j=-w_2}^{w_2} \sum_{k=-w_3}^{w_3} W_{(i+w_1, j+w_2, k+w_3)} I_{(x+i, y+j, z+k)} \tag{2.18}$$

où W_{3D} est le noyau de convolution 3D de dimension $(2w_1 + 1, 2w_2 + 1, 2w_3 + 1)$. Il en est de même pour les couches *d'agrégation maximale* (Eq 2.11) qui peuvent être étendues aux *cartes volumiques* en utilisant une dimension en 3D (h, l, p) :

$$MP_{(x,y,z)}(I_V) = \max_{i \in [x - \lfloor \frac{l}{2} \rfloor, x + \lfloor \frac{l}{2} \rfloor], j \in [y - \lfloor \frac{h}{2} \rfloor, y + \lfloor \frac{h}{2} \rfloor], k \in [z - \lfloor \frac{p}{2} \rfloor, z + \lfloor \frac{p}{2} \rfloor]} \mathscr{V}(i, j, k) \tag{2.19}$$

Dans la figure 23 (c) nous présentons un exemple d'une telle architecture qui compte 2 couches de *convolution volumique* (Eq 2.18) et une couche *d'agrégation volumique* (Eq 2.19).

Un problème des *convolutions volumiques* est que, comparées aux *convolutions classiques*, le nombre de paramètres passe de $(2w_1 \times 2w_2)$ à $(2w_1 \times 2w_2 \times 2w_3)$ à cause de la dimension supplémentaire. Ceci explique pourquoi les architectures par *voxelisation* sont en général moins

profondes que les *réseaux de neurones convolutifs classiques*. Ceci est d'autant plus vrai que la plupart des calculs réalisés lors d'une convolution volumique sont inutiles à cause des *voxels* vides. Pour résoudre ce problème, des opérateurs de *convolutions clairsemés (sparse convolution)* ont été développés. Ceux-ci ont pour particularité d'appliquer les *convolutions volumiques* (Eq 2.18) uniquement sur les voxels non vides.

Simuler des séries de photos dans une scène 3D

Une autre transformation pouvant être appliquée aux nuages de points est une projection des points 3D sur une multitude de plans. Ce qui revient à utiliser une série de photos d'un nuage de points 3D au lieu des points qui le composent. L'idée des méthodes de multi-vues est d'utiliser plusieurs images 2D d'un nuage de points, générées par une caméra virtuelle, correspondant chacune à un angle de vue différent, pour compenser la perte d'information causée par le passage d'une représentation 3D à une représentation 2D. Les images obtenues peuvent ensuite être passées en entrée de *réseaux de neurones convolutifs* existants.

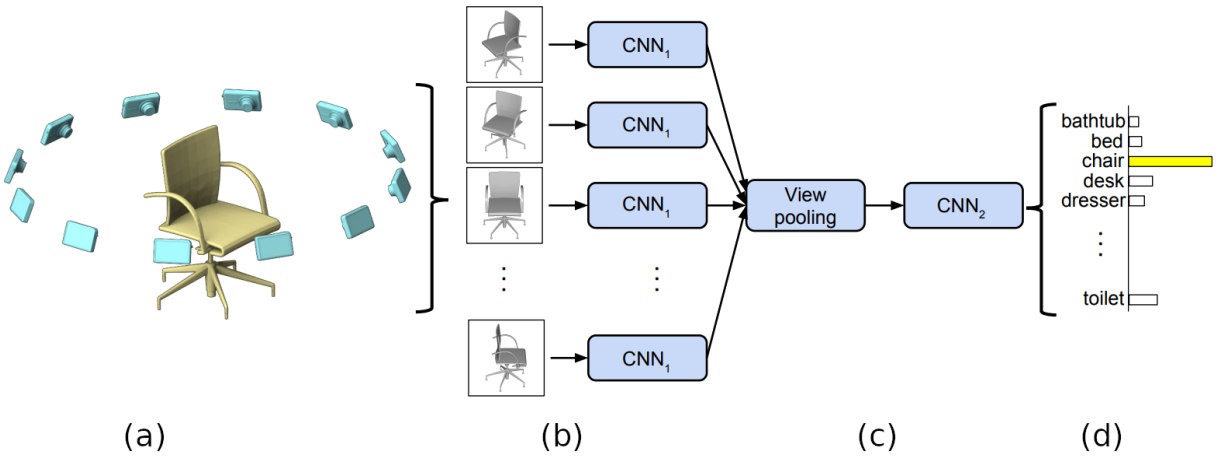

FIGURE 24 – Architecture *MVCNN* avec : en (*a*) le maillage surfacique original (ou un nuage de points triangulé), en jaune, et les différents plans de projection, en bleu, ainsi que les images associées à ces plans. En (*b*) chacune des images est passée en entrée d'un *réseau de neurones convolutif*, 'CNN$_1$' en bleu, pour obtenir une carte de caractéristiques. En (*c*) et (*d*) les cartes de caractéristiques sont *fusionnées* à l'aide d'un second *réseau de neurones convolutif*, 'CNN$_2$' en bleu, afin de prédire une classe pour le nuage de points original. Figure tirée de [SMKLM15].

Les approches multi-vues permettent ainsi de bénéficier des architectures de *CNN 2D* et donc de s'appuyer sur des recherches menées depuis 2012. En effet, les différentes méthodes multi-vues [SMKLM15][SBZB15][BGLA18][LDT+17] suivent toutes un schéma similaire :

1. Plusieurs plans de projection sont définis pour calculer une liste d'images $\{I_i\}_1^{N_V}$ à partir du nuage de points S. Dans la figure 24 (*a*) nous représentons les plans de projection, en bleu, ainsi que les images qui en découlent.

2. Chaque image est ensuite passée en entrée à un *réseau de neurones convolutif* afin

d'obtenir une *carte de caractéristiques*. Cette étape est illustrée dans la figure 24 (*b*) où chaque image est passée en entrée d'un réseau intitulé 'CNN$_1$'.

3. Toutes les *cartes de caractéristiques* sont *fusionnées* entre elles afin d'obtenir une *carte de caractéristiques globale*. Il existe plusieurs méthodes pour fusionner les *cartes de caractéristiques*, par exemple [SMKLM15] propose de concaténer celles-ci puis de les passer en entrée d'un second *réseau de neurones convolutif*. Cette méthode est illustrée dans la figure 24 (*c*) où le *réseau de neurones convolutif* utilisé est intitulé 'CNN$_2$'.

4. Il est possible de calculer une *fonction de coût* sur la *carte de caractéristiques globale* afin d'apprendre une tâche précise. Par exemple avec une fonction *d'entropie croisée* (Eq 2.12), il est possible d'apprendre une classification, représentée dans la figure 24 (*d*).

Afin d'éviter d'obtenir une image 'à trous' pour chaque plan de projection à cause des espaces vides entre les points du nuage de points S, le nuage de points peut être *triangulé* puis *rasterisé*. Néanmoins comme nous l'avons précédemment indiqué, l'opération de triangulation d'un nuage de points peut donner comme résultat un modèle surfacique très grossier selon la densité du nuage de points et ce d'autant plus en présence d'occultations.

Autres transformation existantes

Il existe d'autres manières de structurer des nuages de points dans le but de leur appliquer des couches de convolution. Par exemple le réseau *KdNet* [KL17] propose de représenter chaque nuage de points par un espace de recherche *kd*. Le réseau *SOnet* [LCL18] utilise quant à lui une représentation sous forme de *carte auto-adaptative* pour chaque nuage de points.

2.4.2 Limites des approches par structuration

Une des limites les plus évidentes des approches par structuration est le fait que les discrétisations et/ou échantillonnages causés par les transformations appliquées sur les nuages de points, comme les voxelisations ou les images de projections, peuvent causer une perte d'informations. Un autre problème des approches par structuration est leurs dépendances par rapport aux *réseaux de neurones convolutifs* existants, développés initialement pour l'apprentissage à partir de données structurées et non de nuages de points.

A ces limites générales s'ajoutent des problèmes spécifiques à notre problématique. Ainsi, d'une part les nuages de points sur lesquels nous voulons travailler sont issus d'acquisition *LiDAR MLS*, ce qui fait qu'il s'agit de nuages de points de grande taille. Or une taille importante accentue les problèmes de consommation de mémoire des approches par voxelisation. D'autre part, les nombreuses occultations, inhérentes aux acquisitions *MLS*, complexifient l'utilisation d'approche par multi-vues.

2.4.3 Définition d'une fonction de *quasi-convolution*

A partir de 2017, une nouvelle famille de méthodes est apparue avec le réseau *PointNet* [QSMG17]. Cette famille de méthodes consiste à définir une fonction $\mathscr{F}_{Point}$ capable d'extraire des caractéristiques à partir d'un nuage de points S sans passer par une transformation de celui-ci. La fonction $\mathscr{F}_{Point}$ est définie comme ceci :

$$\mathscr{F}_{Point} : \left| \begin{array}{l} \mathbb{R}^{N\times 3} \rightarrow \mathbb{R}^d \\ S \rightarrow \mathbf{F}_S \end{array} \right. \tag{2.20}$$

où $\mathbf{F}_S$ est un *vecteur de caractéristiques* de dimension d et N la taille du nuage de points S.

Une fonction symétrique pour l'extraction de caractéristiques

Comme les nuages des points ne sont pas ordonnés, la fonction $\mathscr{F}_{Point}$ doit être invariante à l'ordre des points et être symétrique : $\mathscr{F}_{Point}(S) = \mathscr{F}_{Point}(\sigma(S))$, pour toute permutation σ.

Afin de garantir la symétrie de la fonction $\mathscr{F}_{Point}$, les auteurs de [QSMG17] proposent de définir cette dernière sous cette forme :

$$\forall S = \{\mathbf{P}_i\}_1^N, \mathscr{F}_{Point}(S) = \square(g(\mathbf{P}_1), \ldots g(\mathbf{P}_N)) \tag{2.21}$$

où la fonction g est une fonction appliquée directement sur chaque point $\mathbf{P}_i$ et $\square$ la fonction permettant d'agglomérer les $g(\mathbf{P}_i)$. La fonction $\square \circ g$ est alors symétrique si et seulement si la fonction $\square$ est symétrique. La fonction choisie pour $\square$ est la fonction *d'agrégation maximale* afin de satisfaire la contrainte de symétrie. En ce qui concerne la fonction g, celle-ci est implémentée par une *couche de perceptron-multicouches (multi-layer perceptron ou MLP)*. Concrètement il s'agit de plusieurs couches de *perceptron* (Eq 2.1) calculées successivement. Ces couches sont représentées par des rectangles bleus étiquetés '*mlp*' dans la figure 25. La fonction g est donc définie comme ceci :

$$g : \left| \begin{array}{l} \mathbb{R}^3 \rightarrow \mathbb{R}^d \\ \mathbf{P} \rightarrow \mathbf{F}_P \end{array} \right. \tag{2.22}$$

où $\mathbf{F}_P$ est le *vecteur de caractéristiques ponctuelles* associé au point $\mathbf{P}$, représenté en rouge dans la figure 25. Comme la fonction $\square$ (Eq 2.21) choisie est l'agrégation maximale, le *vecteur de caractéristiques globales* $\mathbf{F}_S$, représenté en vert dans la figure 25, est donc défini comme :

$$\mathbf{F}_S | \forall j \in \{1 \ldots d\}, \mathbf{F}_{S,j} = \max_{i \in \{1 \ldots N\}} \mathbf{F}_{P_i,j} \tag{2.23}$$

où $\mathbf{F}_{S,j}$ est la $j$ième composante du vecteur $\mathbf{F}_S$ et $\mathbf{F}_{P_i,j}$ est la $j$ème composante du *vecteur de caractéristiques ponctuelles* associé au point $\mathbf{P}_i$.

Les caractéristiques extraites par le réseau doivent aussi être invariantes aux systèmes de coordonnées utilisés par les différents nuages de points. Ainsi, avant de calculer les *vecteurs de caractéristiques ponctuels* $\mathbf{F}_P$, les coordonnées des points sont d'abord normalisées en $[0, 1]$ et une transformation géométrique leur est appliquée. Les coefficients de la matrice de transformation sont appris par une version miniature du réseau *PointNet*, appelé *T-Net*. Le réseau *T-Net* est représenté dans la figure 25 encadré en rose.

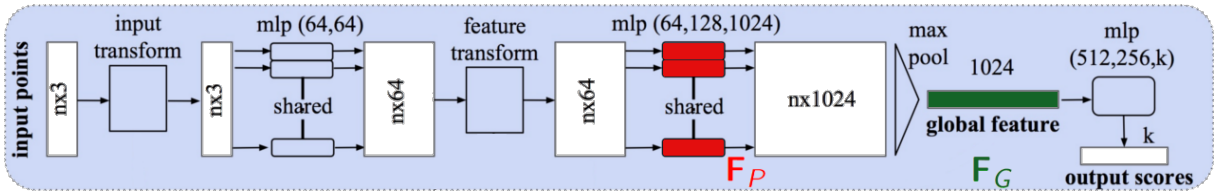

FIGURE 25 – Schéma de l'architecture *PointNet* [QSMG17] : les points sont passés en entrée des *couches de perceptron-multicouches* afin de calculer un *vecteur de caractéristiques ponctuelles*, en rouge. Une opération *d'agrégation maximale* est ensuite appliquée sur ces *vecteurs de caractéristiques ponctuels* afin d'obtenir un *vecteur de caractéristiques globales*, en vert. Figure tirée de [QSMG17]

Détails sur le processus d'apprentissage de *PointNet*

L'utilisation de *couches de perceptron-multicouches* à poids partagés implique que les *vecteurs de caractéristiques ponctuels* $\mathbf{F}_P$ sont calculés pour chaque point indépendamment des autres. Or puisque le *vecteur de caractéristiques globales* $\mathbf{F}_S$ n'est composé que des composantes maximales des *vecteurs de caractéristiques ponctuels* $\mathbf{F}_P$, certains points $\mathbf{P}$ n'ont alors aucun impact sur le *vecteur de caractéristiques globales* $\mathbf{F}_S$. Ainsi on peut noter S^{Max} l'ensemble des points qui ont un impact sur le vecteur $\mathbf{F}_S$:

$$S^{Max} = \{\mathbf{P} \in S | \exists j, \mathbf{F}_{P,j} = \mathbf{F}_{S,j}\} \tag{2.24}$$

où $\mathbf{F}_{S,j}$ est la $j$ième composante du *vecteur de caractéristiques globales* $\mathbf{F}_S$ et $\mathbf{F}_{P,j}$ la $j$ième composante du *vecteur de caractéristiques ponctuelles* du point $\mathbf{P}$. S^{Max} est alors appelé l'ensemble des *points critiques* du nuage de points S. De plus les auteurs de *PointNet* produisent la preuve de la continuité de la fonction g autrement dit : deux points $\mathbf{P}_i$ et $\mathbf{P}_j$ suffisamment proches géométriquement, donnent des *vecteurs de caractéristiques ponctuels* $\mathbf{F}_{P_i}$ et $\mathbf{F}_{P_j}$ avec des valeurs proches. Ce qui leur permet de définir l'ensemble $\hat{S}$ comme ceci :

$$\hat{S} = \{\mathbf{P} \in \mathbb{R}^3 | \forall j, \exists \mathbf{P}_i \in S^{Max} : F_{P,j} \leq F_{P_i,j}\} \tag{2.25}$$

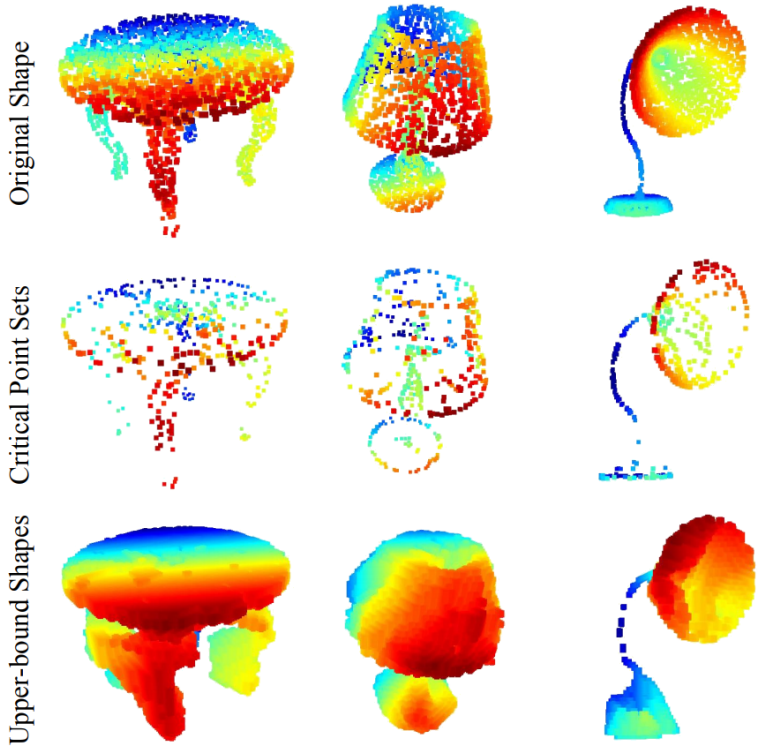

FIGURE 26 – De haut en bas : le nuage de points original S, l'ensemble des *points critiques* S^{Max} (Eq 26) et *l'enveloppe* du nuage de points $\hat{S}$ (Eq 2.25). La couleur correspond à la profondeur du nuage dans l'espace (axe Y). Figure tirée de [QSMG17].

L'ensemble $\hat{S}$ est appelé *l'enveloppe du nuage de points* S, il s'agit de l'ensemble des points de $\mathbb{R}^3$ qu'il est possible d'ajouter au nuage de points S sans que cela modifie son *vecteur de caractéristiques globales* $\mathbf{F}_S$. Des exemples d'ensembles de points critiques S^{Max} et d'enveloppes $\hat{S}$ sont illustrés en figure 26. La fonction d'apprentissage de *PointNet* peut alors être définie comme celle cherchant à délimiter l'espace géométrique compris entre l'ensemble des points critiques S^{Max} et l'enveloppe $\hat{S}$.

Les deux paramètres les plus importants sont alors le nombre de points N par nuage de points et la dimension d des *vecteurs de caractéristiques* $\mathbf{F}_P$ et $\mathbf{F}_S$, puisque par définition (Eq 2.24) le nombre de points critiques est borné par la dimension d. Ainsi une plus grande valeur de d permet, entre autres, d'augmenter potentiellement le nombre de points critiques par nuage de points, tandis qu'une plus grande valeur de N permet d'avoir un choix plus large lors de l'apprentissage des points critiques.

2.5 *PointNet* : un cadre adapté à la généralisation des *réseaux de neurones convolutifs*

Les avancées en termes d'extraction de caractéristiques des nuages de points rendues possibles par l'utilisation de *réseaux de neurones profonds* basés sur *PointNet* [QSMG17] ont permis

de repenser les approches de traitement automatique de nuages de points. L'utilisation de ces réseaux de neurones pour le traitement de données 3D a permis d'augmenter significativement les performances de nombreux cas d'application, voire de s'attaquer à de nouveaux problèmes jusqu'alors insolubles.

Plusieurs réseaux de neurones 3D ont ainsi été utilisés pour des problèmes aussi divers que le recalage et le repositionnement de nuages de points [VBV18][DBI18], la génération aléatoire de nuages de points à partir d'un modèle [SLK19][SWL+20][LZZ+19][YHCOZ18][ZT18a][YHH+19], la définition de 'points-clés' [FLCP+20][YSGG17], la reconstruction de modèles 3D à partir d'un nuage de points [YLF+18a][ZBDT19], la subdivision de modèles 3D et de nuages de points [YLF+18b][YKH+18][YWH+19][LLF+19], la traduction de scènes 3D en langage naturel [RWS+18], l'estimation de positions 6D [CDB+20], la reconstruction 3D à partir d'une image unique [ZKG18], le débruitage de nuages de points [RLG+20][CRR19][DCK19], l'estimation de normales dans un nuage de points [LFXP19], la prédiction de champs de mouvement (*flow*) d'une scène 3D [LQG19][WLHJ+20][GWW+19].

Dans la suite de cette section nous allons examiner plus en détails les apports des *réseaux de neurones profonds* basés sur *PointNet* [QSMG17] aux problèmes définis dans le chapitre 1.

2.5.1 Classification d'objets 3D

PointNet classique

Le réseau *PointNet* [QSMG17] peut facilement être utilisé pour classifier un nuage de points S. En effet il suffit de faire passer *le vecteur de caractéristiques globales* $\mathbf{F}_S$ (Eq 2.23) en entrée d'un classifieur afin de prédire une classe C. Néanmoins, comme la fonction $\mathscr{F}_{Point}$ (Eq 2.21) calculée par PointNet [QSMG17] ne prend pas en compte les relations locales entre les points, celui-ci ne peut pas prendre en compte les détails de la structure d'un nuage de points mais uniquement sa forme globale. Deux approches ont ainsi été proposées afin de résoudre ce problème. La première consiste à fusionner *PointNet* à des méthodes par *structuration* et la deuxième consiste à prendre en compte le contexte local des points lors du calcul des *vecteurs de caractéristiques ponctuels*. Cette dernière étant plus utilisée pour la tâche de *segmentation sémantique*, nous la présentons dans la section suivante.

PointNet combiné aux méthodes par *structuration*

Le but de ces méthodes est d'utiliser le réseau *PointNet* [QSMG17] afin d'augmenter les performances de méthodes par structuration.

Le réseau décrit dans [RROG18] propose de modifier la sortie du réseau *PointNet* [QSMG17] afin de prédire une série de plans de projections à partir du *vecteur de caractéristiques globales*. Comme dans les approches multi-vues, le nuage est projeté dans une image suivant chacun de ces

plans. Ces images sont synthétisées sous forme de *carte de profondeur*. Les *cartes de profondeur* sont ensuite passées en entrée d'un *réseau de neurones convolutif* pour en déduire des *cartes de caractéristiques*. Les *cartes de caractéristiques* sont ensuite fusionnées et le résultat est passé en entrée de couches *entièrement connectées* afin de prédire une classe C.

Le réseau *PointGrid* [LD18] propose de *voxeliser* le nuage de points en entrée comme les méthodes par *voxelisation*. Cependant, au lieu de calculer pour chaque *voxel* la densité des points qu'il contient, comme dans *VoxNet* [MS15] par exemple, *PointGrid* [LD18] propose, pour chaque *voxel* $\mathscr{V}_{i,j,k}$, de faire passer en entrée d'un réseau *PoinNet* les points qu'il contient pour calculer leur *vecteur de caractéristiques globales* et associer ce dernier au *voxel* $\mathscr{V}_{i,j,k}$. L'image volumique qui en résulte est finalement passée en entrée d'un *réseau de neurones convolutifs volumique* pour prédire une classe C.

2.5.2 Segmentation sémantique

Limites de l'approche *PointNet*

Le réseau PointNet [QSMG17] peut aussi être entraîné sur la tâche de segmentation sémantique. Pour cela, le *vecteur de caractéristiques globales* $\mathbf{F}_S$ (Eq 2.23) est concaténé à chacun des *vecteurs de caractéristiques ponctuels* $\mathbf{F}_P$ (Eq 2.22) pour obtenir des *vecteurs de caractéristiques mixtes*. Ces *vecteurs de caractéristiques mixtes* sont ensuite passés en entrée d'un classifieur pour prédire une classe C pour chaque point du nuage. Cette opération est représentée dans la figure 25 par un encadré jaune. Cependant, comme nous l'avons précédemment indiqué, le calcul des *vecteurs de caractéristiques ponctuels* ne prend pas en compte le contexte local des points mais uniquement leurs répartitions dans la forme globale du nuage. Ce problème est plus handicapant dans le cas de la *segmentation sémantique* où le réseau a besoin de prédire des classes différentes pour des points d'un même nuage.

Afin de résoudre ce problème, plusieurs approches basées sur *PointNet* [QYSG17] et visant à introduire le contexte local des points ont vu le jour. Nous proposons de répartir ces approches *post-Pointnet* en deux familles :

(**a**) les approches par *PointNet* localisé

(**b**) les approches par *convolutions de points*

Avant de présenter ces approches, nous introduisons d'abord un certain nombre d'opérateurs classiques du traitement des données 3D qui nous serviront à décrire précisément le fonctionnement de ces réseaux de neurones.

Des opérateurs 3D particulièrement utiles

Certains opérateurs classiques du traitement des données 3D peuvent être utilisés selon leurs capacité à être intégrés dans un réseau de neurones. Parmi les plus utilisés figure le voisinage

relatif d'un point $\mathbf{P}$, noté $\mathcal{N}_{Rel}(\mathbf{P})$, et défini comme ceci :

$$\mathcal{N}_{Rel} : \left| \begin{array}{l} \mathbb{R}^3 \times \mathbb{R}^{N\times 3} \rightarrow \mathbb{R}^{k\times 3} \\ \mathbf{P} \times S \rightarrow \mathcal{N}_{Rel}(\mathbf{P}) = \{\mathbf{P}_i \in S\}_{i=1}^{i=k} \end{array} \right. \tag{2.26}$$

où l'ensemble $\mathcal{N}_{Rel}(\mathbf{P})$ est formé des k points du nuage S les plus proches de $\mathbf{P}$.

Afin de borner la distance maximale entre points les plus proches, l'opérateur de *voisinage sphérique* ('*ball query*'), noté $\mathcal{N}_S$ est parfois utilisé. Celui-ci est défini comme un calcul des plus proches voisins $\mathcal{N}_{Rel}$ dont la distance est bornée par un seuil ρ :

$$\mathcal{N}_S(\mathbf{P}) = \{\mathbf{P}_i \in \mathcal{N}_{Rel}(\mathbf{P}) | D(\mathbf{P}, \mathbf{P}_i) \leq \rho\} \tag{2.27}$$

Un autre opérateur très utilisé est l'opérateur de *sous-échantillonnage*, noté ξ, qui échantillonne un nombre fixe, N_E, parmi les N points d'un nuage S :

$$\xi : \left| \begin{array}{l} \mathbb{R}^{N\times 3} \rightarrow \mathbb{R}^{N_E\times 3} \\ \{\mathbf{P}_i\}_1^N \rightarrow \{\mathbf{P}_i\}_1^{N_E} \end{array} \right. \tag{2.28}$$

Les points peuvent être sélectionnés de manière purement aléatoire ou bien en utilisant une approche plus complexe comme l'*algorithme des points les plus éloignés* (*farthest point sampling ou FPS*) [Ove00].

a) *PointNet* localisé : graphe et voisinages

Les approches par *PointNet* localisé consistent essentiellement en la définition de structures locales pour représenter le contexte de chaque point. Parmi les plus importantes nous pouvons citer : *PointNet*++[QYSG17], les approches par graphes dynamiques [WSL+19] et [ZHW+19] et l'approche par graphe statique [LS18].

Le réseau *PointNet*++ [QYSG17] propose d'améliorer le réseau *PointNet* [QSMG17] en y ajoutant deux nouvelles couches : la couche SA et la couche FP.

La couche SA ('*set abstraction layer*') est la couche de regroupement. Elle est représentée dans la figure 27 à gauche. Elle prend en entrée un nuage de points S et lui applique d'abord une opération de *sous-échantillonnage* ξ (Eq 2.28). Pour chaque point $\mathbf{P}$ du nuage de points sous-échantillonné, noté $\xi(S)$, un contexte local est défini par *voisinage sphérique* $\mathcal{N}_S$ (Eq 2.27), représenté par les cercles en pointillé dans la figure 27. Enfin, chaque ensemble de points $\mathcal{N}_S(\mathbf{P})$ est passé en entrée d'un réseau *PointNet* [QSMG17] afin d'en prédire un *vecteur de caractéristiques globales* $\mathbf{F}_S$. La couche SA peut être appliquée de manière successive, c'est-à-dire que les points $\mathbf{P}$ en entrée peuvent être définis par un *vecteur de caractéristiques ponctuelles* $\mathbf{F}_P$, de dimension d. Ainsi l'application de la $l$ième couche SA est définie comme ceci :

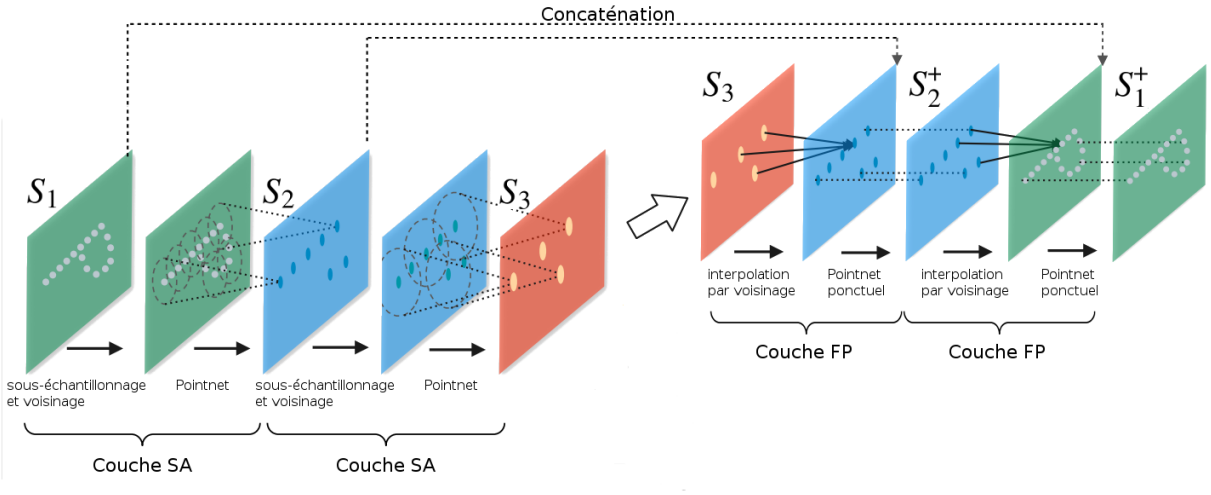

FIGURE 27 – Schéma de l'architecture de *PointNet++* : à gauche un empilement de 2 couches SA, à droite en haut un empilement de deux 2 FP pour la segmentation sémantique. figure tirée de [QYSG17].

$$SA : \left| \begin{array}{l} \mathbb{R}^{N_l \times (d_l)} \longrightarrow \mathbb{R}^{N_{l+1} \times (d_{l+1})} \\ S_l \longrightarrow S_{l+1} \end{array} \right. \tag{2.29}$$

où S_l est le nuage de points en entrée de la $l$ième couche SA, N_l sa taille, avec $N_l > N_{l+1}$ et d_l la dimension des *vecteurs de caractéristiques ponctuels* $\mathbf{F}_P$ associés à chaque point du nuage S_l. Dans le cas de la première couche SA, on a $d_l = 3$ quand les points ne sont définis que par leurs coordonnées (x, y, z). La sortie S_{l+1} de la $l$ième couche SA est définie comme ceci :

$$S_{l+1} = \{\mathbf{F}_P = \mathscr{F}_{Point}\left(\mathscr{N}_S(\mathbf{P}_i \in \xi(S))\right)\}_{i=1}^{i=N_{l+1}} \tag{2.30}$$

où $\mathscr{F}_{Point}(\mathbf{P})$ est la fonction calculée par le réseau *PointNet* [QSMG17] (Eq 2.23), c'est à dire le *vecteur de caractéristiques globales* de l'ensemble des points du voisinage sphérique des points $\mathbf{P}_i \in \xi(S)$.

Le rôle des couches FP est de mettre à jour chaque nuage S_l à partir de S_{l+1}, avec $S_{l+1} = SA(S)$. Ainsi, en notant S_L le résultat de la dernière couche SA, la première couche FP va mettre à jour le nuage S_{L-1}, puis les couches FP suivantes vont s'appliquer successivement jusqu'à arriver à S_0. Notons que les couches FP s'appliquent de manière successives et que, par conséquent, celles-ci s'opèrent à partir des versions mises à jour des nuages S_l, notées S_l^+, avec $0 \leq l \leq L$. La fonction calculée par les couches FP est alors définie comme ceci

$$FP : \left| \begin{array}{l} \mathbb{R}^{N_l \times d_l} \times \mathbb{R}^{N_{l+1} \times d_{l+1}^+} \longrightarrow \mathbb{R}^{N_l \times d_l^+} \\ S_l \times S_{l+1}^+ \longrightarrow S_l^+ \end{array} \right. \tag{2.31}$$

où S_l^+ est le nuage S_l après mise à jour, de même d_l^+ est la dimension des points du nuage S_l après mise à jour. Ainsi dans le cas où $l = L : S_l^+ = S_l$ et dans le cas où $l < L : S_l^+ = FP(S_l)$.

Le fonctionnement des couches FP est présenté dans la figure 27 à droite. Pour chaque point $\mathbf{P}$ du nuage S_l, avec $0 \leq l < L$, un *vecteur de caractéristiques* est tout d'abord calculé par la somme pondérée des voisins relatifs de $\mathbf{P}$ dans le nuage S_{l+1}^+. Ce *vecteur de caractéristiques* est concaténé au point $\mathbf{P}$ et le résultat est passé en entrée de la fonction g du réseau *PointNet* (Eq 2.22) afin de calculer un nouveau *vecteur de caractéristiques ponctuels*. Ainsi la sortie de la couche FP appliquée au nuage S_l est définie comme ceci :

$$S_l^+ = \left\{ \mathbf{F}_P^+ = g\left(\mathbf{F}_P \oplus \sum_{\mathbf{P}_i \in \mathcal{N}_{rel}(\mathbf{P})} w_i \mathbf{F}_{P_i}^+ \right) \right\} \tag{2.32}$$

où $\oplus$ est l'opération de concaténation et les poids w_i sont calculés comme l'inverse de la distance euclidienne D entre le point $\mathbf{P}$ et son voisin $\mathbf{P}_i$: $w_i = \frac{1}{D(\mathbf{P},\mathbf{P}_i)}$. Nous pouvons alors reconnaître dans l'architecture du réseau *PointNet++* [QYSG17], présentée en figure 27, une architecture similaire au réseau *U-Net* [WZ21] où les couches de *convolution* sont remplacées par les couches *SA* (Eq 2.30) et les couches de *déconvolution* sont remplacées par les couches *FP* (Eq 2.32). Après la dernière couche FP, un score est prédit pour chaque *vecteur de caractéristiques ponctuelles* $\mathbf{F}_P$.

Les réseaux *DGCNN* [WSL+19] et *LDGCNN* [ZHW+19] s'appuient uniquement sur le calcul des voisins les plus proches $\mathcal{N}_{Rel}$ (Eq 2.26), pour chaque point $\mathbf{P}$ du nuage de points en entrée S, pour calculer une fonction g_{DGCNN} qui permet d'extraire des *caractéristiques locales* pour chaque point $\mathbf{P}$. La fonction g_{DGCNN} est alors définie comme ceci :

$$g_{DGCNN} : \left| \begin{array}{l} \mathbb{R}^3 \times \mathbb{R}^{K\times 3} \rightarrow \mathbb{R}^d \\ \mathbf{P} \times \mathcal{N}_{Rel}(\mathbf{P}) \rightarrow \mathbf{F}_P \end{array} \right. \tag{2.33}$$

où K est le nombre de voisins utilisés dans la fonction $\mathcal{N}_{Rel}$ (Eq 2.26). Le résultat $\mathbf{F}_P$ de l'application de la fonction g_{DGCNN} sur un point $\mathbf{P}$ est définie ainsi :

$$\mathbf{F}_P = \max_{\mathbf{P}_i \in \mathcal{N}_{Rel}(\mathbf{P})} g(\mathbf{P} \oplus (\mathbf{P}_i - \mathbf{P})) \tag{2.34}$$

où $\oplus$ représente l'opération de concaténation et max est un opérateur d'agrégation maximale composante par composante.

Le réseau *SPG* [LS18] permet de prendre en entrée de très grands nuages de points en calculant au préalable un graphe $(s, \mathscr{A})$ par un algorithme de partition de graphe [LO16], tel qu'un sommet représente un ensemble de points du nuage et une arête un lien de proximité entre deux ensembles de points. Pour chaque arête $\mathscr{A}$, un vecteur *descripteur* $\mathbf{A}$ est calculé. Pour chaque sommet s, un *vecteur de caractéristiques* $\mathbf{F}_s$ est calculé par un *réseau de neurones récurent* (*RNN*) [CVG+14]. Celui-ci a pour état initial $\mathbf{F}_{s,0}$:

$$F_{s,0} = \mathscr{F}_{Point}(\{\mathbf{P} \in s\}) \tag{2.35}$$

où $\{\mathbf{P} \in s\}$ est l'ensemble des points du nuage S associé au sommet s par l'algorithme de partition de graphe [LO16] et $\mathscr{F}_{Point}$ (Eq 2.23) est la fonction calculée par le réseau *PointNet*, c'est-à-dire le *vecteur de caractéristiques globales* de l'ensemble des points $\{\mathbf{P} \in s\}$.

Le $l$ième état du *vecteur de caractéristiques* $\mathbf{F}_{s,l}$ est alors calculé de manière récursive :

$$\mathbf{F}_{s,l} = \sum_{A=(s_i,s)\in\mathscr{A}} \Gamma(\mathbf{A}\mathbf{F}_{s_i,l-1}, \mathbf{F}_{s,l-1}) \tag{2.36}$$

où $\mathbf{A}$ est le vecteur descripteur associé à l'arête A et Γ la fonction d'actualisation du *réseau de neurones récurrent* [CVG+14]. Ainsi chaque sommet s du graphe calcule un *vecteur de caractéristiques* associé à un sous-ensemble du nuage de points S, puis ces *vecteurs de caractéristiques* sont affinés par rapport à ceux des sommets voisins. Cependant, le réseau de neurones *SPG* [LS18] nécessite un temps non négligeable pour construire le graphe $(s, \mathscr{A})$ et la construction de celui-ci n'est pas supervisée par le réseau. Ainsi, comme les *vecteurs de caractéristiques* sont calculés au niveau des sommets, et non au niveau des points, tous les points appartenant à un même sommet seront classifiés de manière identique par le réseau. Ce qui veut dire que si des points appartenant à des classes C différentes sont regroupés dans un même sommet, le réseau ne pourra pas 'rattraper' cette erreur. Une version améliorée du réseau *SPG* [LB19] permet d'être moins sensible à ce problème.

b) Les approches par *convolution* de points

Les approches par convolution de points propose de s'appuyer sur le réseau *PointNet* [QSMG17] afin de généraliser les couches de *convolutions 2D* (Eq 2.10) aux nuages de points 3D. En effet, dans les approches spatiales présentées dans la section précédente n'apparaît pas au sein du mécanisme de rétropropagation les poids des sommes pondérées entre points voisins. Dans tous les cas, les poids ne sont pas appris par le réseau. Soit les poids sont identiques pour tous les voisins, ou soit les poids sont fixes, comme par exemple la distance (Eq 2.32) dans le réseau *PointNet*++ [QYSG17] ou les descripteurs dans le réseau *SPG* [LS18]. Contrairement à ces méthodes, les approches par *convolution de points* proposent de faire apprendre au réseau ces poids, comme c'est le cas dans les couches de *convolution 2D* (Eq 2.10). Dans un premier temps, l'opération de *convolution 2D* (Eq 2.10) d'une image I par un noyau W sur le pixel P est

redéfinie comme ceci :

$$(W * I)(P) = \sum_{P_i \in \mathcal{N}(P)} \sum_{w_j \in W} \Phi_{i,j} P_i w_j \tag{2.37}$$

où $\mathcal{N}(P)$ dénombre, dans l'ordre, les pixels voisins du pixel P et les w_j représentent, dans l'ordre, les poids du noyau W. L'application Φ a pour but d'associer chaque poids W au pixel P_i correspondant. Ainsi dans le cas des *convolutions 2D*, celle-ci est définie comme ceci :

$$\Phi_{i,j} = \begin{cases} 1 \text{ si } i = j \\ 0 \text{ sinon} \end{cases} \tag{2.38}$$

Ainsi l'application Φ est ici implicite de par la nature structurée des images 2D. Nous l'illustrons dans la figure 28 (*a*) où le pixel P est représenté en rouge, l'ensemble $\mathcal{N}(P)$ est représenté en vert et les poids W sont représentés en bleu. L'application Φ, en rose, est alors simplement une correspondance entre les pixels $\mathbf{P}_i \in \mathcal{N}(P)$ et les poids W à la même position i.

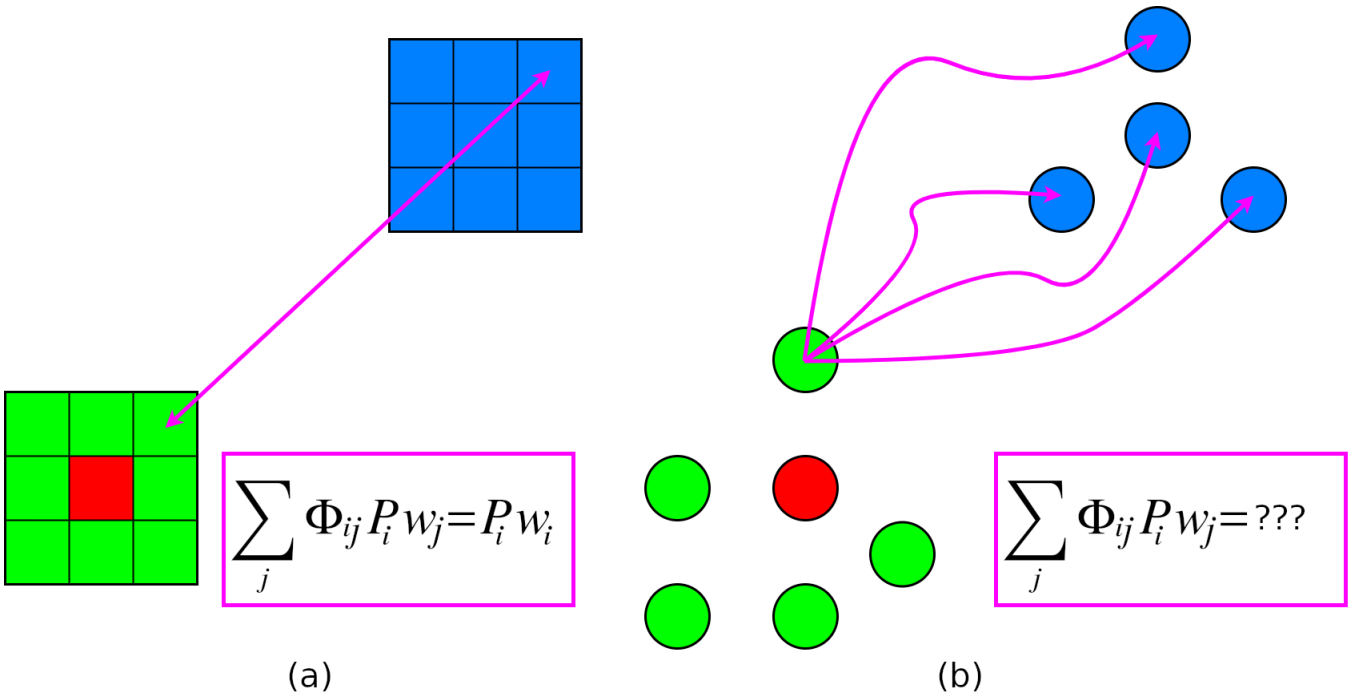

FIGURE 28 – Illustration du problème soulevé par les approches de *convolution par points*. En rouge sont représentés le pixel P et le point $\mathbf{P}$, en vert leur voisinage $\mathcal{N}(P)$ et en bleu les poids W. L'application Φ, représentée par les flèches roses, est triviale sur une grille 2D structurée, dans le cas des *convolutions standards* (Eq 2.10) mais l'application Φ doit être définie pour un nuage 3D non régulier et non ordonné.

Dans le cas des nuages de points 3D où aucun ordre n'est défini entre les points, la définition d'une fonction Φ permettant de généraliser les *convolutions* n'est alors pas un problème trivial

comme en 2D. Nous illustrons ce problème dans la figure 28 (*b*) où le point $\mathbf{P}$ est représenté en rouge, l'ensemble de ses voisins $\mathcal{N}(P)$ en vert et les poids W en bleu. L'application Φ, représentée par les flèches, doit alors définir pour chaque point $\mathbf{P}_i \in \mathcal{N}(P)$ comment il est *impacté* par chacun des poids $w_j \in W$. L'idée derrière les *convolutions de points* est de déterminer une application Φ capable d'assurer cette tâche.

La *convolution par noyau de points* [TQD+19] introduit l'utilisation de points comme *noyau de convolutions*. Ainsi au lieu d'être défini simplement par une liste de scalaires, chaque élément $\mathbf{w} = (\mathbf{w}^g, \mathbf{w}^f)$ d'un noyau W est défini par un point $\mathbf{w}^g \in \mathbb{R}^3$ et un poids $\mathbf{w}^f \in \mathbb{R}^d$. De même, chaque point $\mathbf{P}$ est composé d'une partie géométrique $\mathbf{P}^g \in \mathbb{R}^3$ et d'une partie caractéristique $\mathbf{P}^f \in \mathbb{R}^d$. Pour chaque noyau la valeur initiale des $\mathbf{w}^g$ est définie comme celle maximisant la distance entre chaque point $\mathbf{w}^g$ au sein d'une sphère unitaire. L'opération de convolution d'un point $\mathbf{P}$ par le noyau W est alors calculée comme ceci :

$$Conv_{KP}(\mathbf{P}, W) = \sum_{\mathbf{P}_i \in \mathcal{N}_S(\mathbf{P})} \sum_{\mathbf{w}_j \in W} \Phi_{i,j} \mathbf{P}_i^f \mathbf{w}_j^f \tag{2.39}$$

où $\mathcal{N}_S$ est le voisinage sphérique défini en (Eq 2.27), w_j^f est le poids w^f associé au $j$ième élément du noyau W et l'application Φ est définie comme ceci :

$$\Phi_{i,j} = \max\left(0.1, \frac{D(\mathbf{P}_i^g, \mathbf{w}_j^g)}{\sigma}\right) \tag{2.40}$$

où D est la distance euclidienne entre deux points, w_j^g est le point associé au $j$ième élément du noyau W et σ est un hyperparamètre du réseau qui est choisi selon la densité du nuage de points S.

L'opérateur de convolution défini dans [Bou20] reprend la définition (Eq 2.39) à la différence que les points w_g sont initialisés de manière aléatoire et en introduisant une nouvelle application Φ :

$$\Phi_{i,j} = MLP(\mathbf{P}_i^g - \mathbf{w}_{j,g}) \tag{2.41}$$

où MLP est la fonction calculée par un *perceptron multi-couches* comme pour la fonction g (Eq 2.22) du réseau *PointNet* [QSMG17]. Un schéma de l'application de cet opérateur de *convolution par points* est présenté en figure 29, avec le calcul de l'application Φ en vert. Ainsi à la différence de [TQD+19], cet opérateur de convolution est compatible avec d'autres type de voisinage que le voisinage sphérique (Eq 2.27).

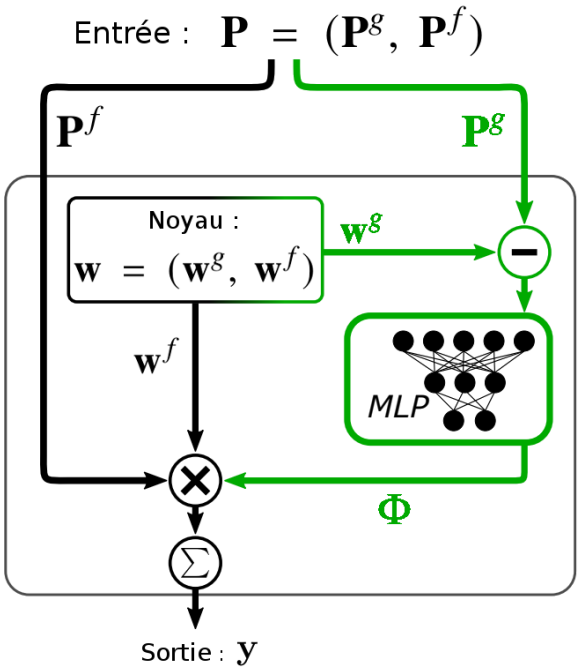

FIGURE 29 – Schéma de l'application d'une convolution de [Bou20] : en haut le point en entrée de caractéristiques $\mathbf{P}^f$ et de coordonnées $\mathbf{P}^g$, au centre le poids défini par sa partie géométrique $\mathbf{w}^g$ et sa partie caractéristique $\mathbf{w}^f$ et à droite la fonction ϕ implémentée par une couche de *perceptrons multi-couches* (*MLP*). Figure tirée de [Bou20].

2.5.3 Difficultés à passer à la détection d'objets

Comme nous l'avons décrit dans la section 1.3.2, le réseau *PointNet* n'est capable d'opérer qu'à deux échelles uniquement : au niveau point avec le calcul de *vecteurs de caractéristiques ponctuelles* $\mathbf{F}_P$ (Eq 2.22) pour chaque point $\mathbf{P}$ ainsi qu'au niveau nuage avec le calcul d'un *vecteur de caractéristiques globales* $\mathbf{F}_S$ (Eq 2.23). Or comme nous l'avons défini dans le chapitre 1, la détection d'objets nécessite d'obtenir des résultats au niveau *objet* et non pas seulement au niveau ponctuel. Cependant aucun mécanisme ne permet de réaliser des opérations au niveau objet ni même au niveau *région* dans le réseau *PointNet* [QSMG17].

Ainsi, la seule option pour réaliser la tâche de détection d'objets avec le réseau *PointNet* est de passer par une approche par fenêtre glissante [QSMG17], et ce même si celle-ci est très limitée à cause des dimensions fixes imposées par la fenêtre. Afin de pouvoir réaliser la détection d'objets il faut alors ajouter au réseau *PointNet* un module capable de réaliser des calculs au niveau *objet* ou du moins au niveau *région*.

2.6 Approches existantes de détection d'objets basées sur PointNet

Dans cette section nous allons présenter les premières tentatives d'incorporation d'un module effectuant des traitements au niveau *région* au sein d'une architecture basée sur *PointNet*

[QSMG17]. Nous regroupons ces approches en 3 familles différentes.

2.6.1 Projection de régions dans un nuage de points

L'idée derrière cette famille de méthodes est d'exploiter les avancées dans le domaine de la détection d'objets dans les images 2D. En effet, il existe plusieurs architectures de *réseaux de neurones convolutifs* capables de prédire des boîtes englobantes autour des objets présents dans une image 2D [RHGS17][LAE+16], voir section 2.3 pour plus de détails. Ces réseaux permettent d'implémenter une fonction que nous appelons $\mathscr{F}_{ROI}$ qui prend en entrée une image 2D et retourne une liste de boîte englobantes 2D $\{\mathbf{B}_{2D}\}$ correspondant aux objets présent dans cette image.

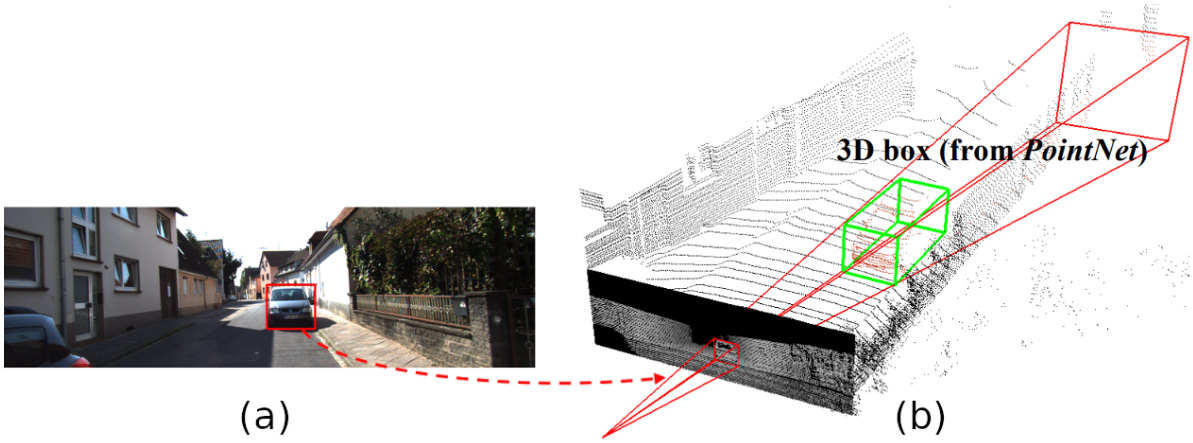

FIGURE 30 – Schéma de l'approche de [QLW+18] : (*a*) un *réseau de neurones convolutif* est appliqué sur une image pour obtenir une boîte englobante 2D B_{2D}, représenté par le rectangle rouge, puis celle-ci est projetée dans le nuage de points pour définir la boîte 3D B_{3D} (*b*), représentée par la pyramide rouge ; *PointNet* [QSMG17] est alors appliqué sur les points dans B_{3D} pour obtenir la boîte finale B, en vert à droite. figure tirée de [QLW+18].

Afin de tirer parti de la fonction $\mathscr{F}_{ROI}$, les méthodes de cette famille prennent en entrée, en plus d'un nuage de points S, une image 2D I étalonnée par rapport au capteur *LiDAR* utilisé. C'est-à-dire qu'il existe une fonction de transformation $T_{I\to S}$ définie pour chaque pixel (u, v) de l'image I :

$$T_{I\to S} : \left| \begin{array}{l} \mathbb{R}^2 \to \mathbb{R}^3 \\ (u, v) \to \mathbf{P} = (x, y, z) \end{array} \right. \tag{2.42}$$

ainsi que la transformée inverse $T_{S\to I}$ définie pour chaque point $\mathbf{P} \in S$:

$$T_{S\to I} : \left| \begin{array}{l} \mathbb{R}^3 \to \mathbb{R}^2 \\ \mathbf{P} = (x, y, z) \to (u, v) \end{array} \right. \tag{2.43}$$

Le module de passage au niveau *région* des méthodes de cette famille suit alors le schéma

suivant [QLW+18][ZLHH19][XAJ18][SKT19][WJ19] :

1. Le module prend en entrée l'image I étalonnée par rapport au nuage de points S. L'image I est représentée par une photo en couleur dans la figure 30 (a).

2. La fonction $\mathscr{F}_{ROI}$ est appliquée sur l'image I afin d'obtenir une liste de boîtes englobantes 2D $\{\mathbf{B}_{2D}\}$. Un exemple de boîte $\mathbf{B}_{2D}$ est représenté par un rectangle rouge autour d'une voiture dans la figure 30 (b).

3. Pour chaque boîte englobante 2D $\mathbf{B}_{2D}$, une première estimation de boîte englobante $3D$ notée $\mathbf{B}'$ est calculée en appliquant la transformation $T_{I\rightarrow S}$ (Eq 2.42) sur chaque pixel appartenant à la boîte englobante 2D $\mathbf{B}_{2D}$. Un exemple de boîte 3D **B**' est représenté par un cône rouge dans la figure 30 (b).

4. Pour chaque première estimation de boîte 3D $\mathbf{B}'$, la boîte englobante finale $\mathbf{B}$ est calculée en appliquant la fonction g (Eq 2.23) du réseau *PointNet* [QSMG17] sur l'ensemble des points $\mathbf{P}$ du nuage de points S qui appartiennent à la boîte $\mathbf{B}'$ afin d'obtenir son *vecteur de caractéristiques globale* $\mathbf{F}_S$. Dans la suite nous notons cet ensemble de points $S_{B'}$. Une régression est apprise à partir de ce *vecteur de caractéristiques globale* afin de prédire la boîte finale $\mathbf{B}$, représentée par un pavé vert dans la figure 30 (b).

La principale limite de cette famille de méthodes est la nécessité d'avoir à disposition des photos étalonnées pour chaque nuage de points, ce qui n'est pas toujours le cas lors d'acquisition *LiDAR* et augmente énormément le volume de données à stocker dans le cas d'acquisition *MLS*. De plus le fait que ces méthodes soient dépendantes d'un *réseau de neurones convolutifs* pour le calcul de la fonction $\mathscr{F}_{ROI}$ les rend très sensibles aux occultations.

Ces deux limites majeures sont dues au fait que le cas natif d'application de cette famille de méthodes est le développement d'assistant de conduite automatique pour des véhicules dits autonomes. Dans ce cas, le volume de données importe peu puisque celles-ci sont traitées en temps réel et les occultations sont moins embêtantes car l'information principale est l'emplacement des objets aux alentours du véhicule. De même il existe d'autres méthodes basées essentiellement sur une prédiction de régions à partir de structuration du nuage de points qui ont le même cadre d'application (voiture autonomes) :

- Les approches par *vues d'oiseau* qui projettent le nuage de points sur une grille horizontale : [CMW+17][YLU18][BGM+18][ZHL+18][YLU20][KML+18][LYWU18]

- Les approches par *voxelisation* qui généralisent la fonction $\mathscr{F}_{ROI}$ aux images volumiques : [ERW+17][LGKX19]

- Les approches mixtes qui utilisent *PointNet* [QSMG17] afin d'extraire des caractéristiques du nuage de points, puis une approche *vue d'oiseau* ou *voxelisation* pour passer au niveau région : [ZT18b][YML18][LVC+19][ZGF+20]

2.6.2 Extrapoler une région par point : généralisation des *RCNN* aux nuages de points

Contrairement aux méthodes précédentes qui s'appuient sur des images 2D pour la prédiction des boîtes englobantes, les méthodes de cette famille proposent de prédire les boîtes englobantes **B** directement à partir d'un nuage de points S. Ainsi le module de passage au niveau objet des méthodes de cette famille peut être défini comme la généralisation de la fonction $\mathscr{F}_{ROI}$ aux nuages de points 3D. Néanmoins, bien que ce module soit appliqué directement sur les points, cela n'empêche pas d'utiliser des images 2D afin d'enrichir l'information contenue dans le nuage de points.

Le réseau *PointRCNN* [SWL19] propose une méthode en 2 temps qui repose sur l'utilisation de 2 réseaux de neurones différents. D'autres méthodes suivent un schéma similaire [YSL+18][VLHB20][ZGG19] :

1. Le nuage de points S est passé en entrée d'un réseau *PointNet++* [QYSG17], afin de prédire une classe C, pour chaque point **P**, qui vaut 1 si le point appartient à un objet et 0 si le point appartient au fond, ainsi qu'une première estimation de boîte englobante **B**'. Cette étape est illustrée dans la figure 31 (*a*), où les boîtes **B**' sont représentées par des pavés oranges.

2. Pour chaque point **P** dont la classe $C = 1$, l'ensemble des points du nuage S qui appartiennent à la boîte **B'** est passé en entrée d'un réseau *PointNet* [QSMG17] afin d'en prédire une boîte finale **B**. Cette deuxième étape est illustrée dans la figure 31 (*b*), où les boîtes finales **B** sont représentées par des pavés verts.

Une des limites de cette approche est la nécessité d'entraîner deux réseaux de manière séquentielle, la sortie du premier étant nécessaire pour entraîner le second. En plus d'augmenter le nombre de paramètres à apprendre, ceci a aussi pour conséquence de ralentir la convergence de la méthode. Néanmoins, il existe une sous-famille récente de ces méthodes qui propose de combiner les deux réseaux utilisés. La méthode en une seule étape qui en résulte est une généralisation de la *transformée de Hough* [Hou60] grâce à un réseau de neurones. Elle suit le schéma suivant :

1. Un nuage de points S de taille N est passé en entrée d'un réseau *PointNet++* [QYSG17], par exemple un nuage de points représentant une table dans la figure 32 (*a*). La dernière couche FP (Eq 2.32) du réseau renvoie un ensemble de N_l points, avec $N_l < N$, chacun associé à un *vecteur de caractéristiques ponctuelles* $\mathbf{F}_i$. Cet ensemble de points est représenté en bleu dans la figure 32 (*b*).

2. Chaque *vecteur de caractéristiques ponctuelles* **F** est ensuite passé en entrée à une *couche entièrement connectée* afin de prédire un point $\mathbf{P}_C = (x, y, z)$. Il s'agit de l'étape de *vote*.

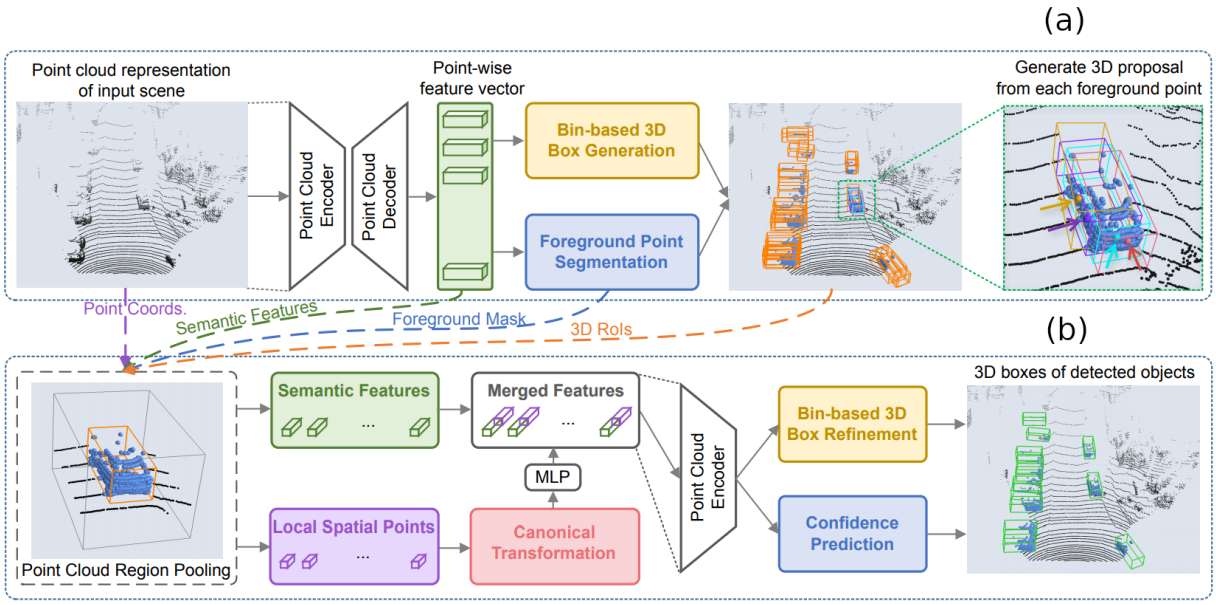

FIGURE 31 – Architecture globale de *PointRCNN* avec : (*a*) le premier sous-réseau chargé de prédire les premières estimations de boîtes englobantes B_{3D} et (*b*) le deuxième sous réseau qui prédit les boîtes englobantes finales B à partir des estimations B_{3D}. Figure tirée de [SWL19].

Les *votes* sont représentés par des points rouges dans la figure 32 (*c*). En effet les points $\mathbf{P}_C$ sont censés être les centres des objets auxquels appartiennent les points en entrée de la *couche entièrement connectée*.

3. Finalement les points $\mathbf{P}_C$ sont regroupés par l'application d'une couche *SA* (Eq 2.30), chaque groupe est représenté par une sphère dans la figure 32 (*d*), et chaque groupe est passé en entrée d'un réseau *PointNet* [QSMG17] afin de prédire une boîte englobante $\mathbf{B}$ (Eq 2.23) qui correspond à une boîte englobante. Ces boîtes englobantes sont représentées par des pavés verts et bleus dans la figure 32 (*e*).

La famille des méthodes par régions, a pour principal avantage de s'appliquer directement sur les nuages de points, même s'il est possible d'améliorer ces méthodes en utilisant des images étalonnées. Cependant, le choix de prédire des boîtes englobantes pour chaque point peut être critiqué. En effet, dans le cas des images 2D, ce choix se justifie par le fait que les images 2D sont, en dernière analyse, des ensembles de boîtes englobantes. Alors que dans le cas des nuages de points 3D, le passage par des boîtes englobantes revient à une approximation, là où un groupement de points semblerait plus pertinent [WYHN18].

La sous-famille des méthodes par vote s'avère alors être un travail pionnier dans le cadre de la détection d'objets 3D. En effet, le groupement des *votes* est une nouvelle manière de passer du niveau point au niveau région sans perdre en précision. Celle-ci est tout de même limitée par le fait que l'algorithme de regroupement des *votes* n'est pas supervisé par le réseau de neurones et est contraint à des groupes de formes sphériques. Cette limite pourrait être d'autant plus handicapante dans le cas de scènes en extérieur où il y a proportionnellement plus de points

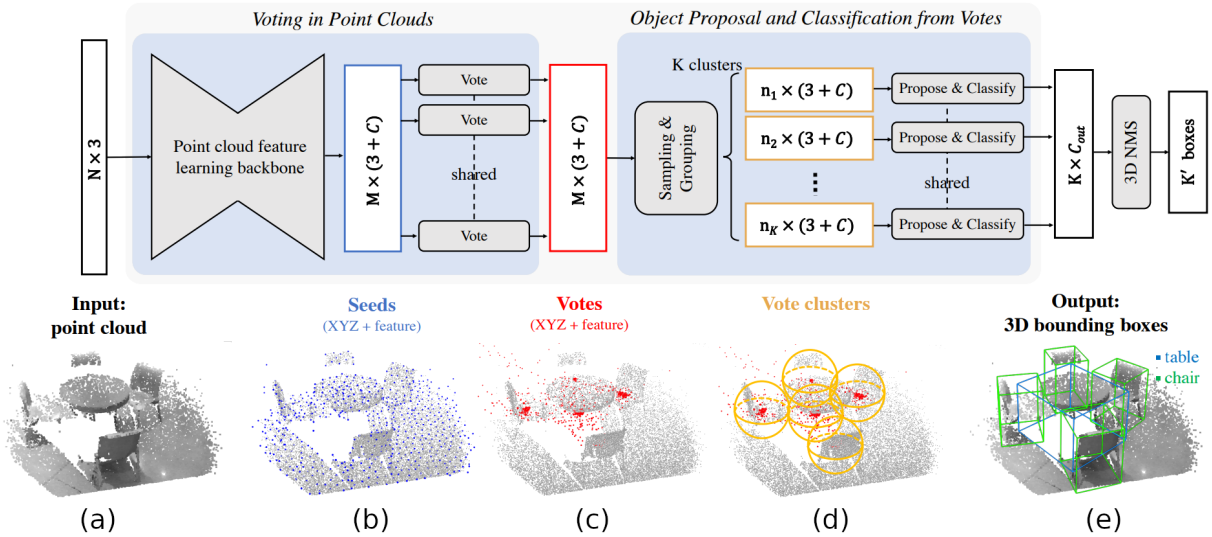

FIGURE 32 – Architecture globale de VoteNet : en haut les couches associées aux différentes étapes : calcul des points graines, vote, regroupement des votes, prédiction de boîtes englobantes et en bas un exemple d'application pour chacune de ces étapes. Figure tirée de [QLHG19].

appartenant au fond que dans les scènes en intérieur.

2.6.3 Segmentation d'instance directe à partir des caractéristiques des points

Cette famille de méthodes ne se base pas sur la prédiction de boîtes englobantes mais prédit directement des sous-ensembles de S correspondant aux objets présents. En effet ces méthodes sont entraînées sur la tâche de *segmentation d'instance* telle que nous l'avons définie au chapitre 1.

Ces méthodes suivent alors un schéma similaire introduit pour la première fois par le réseau *SGPN* [WYHN18]. D'autres méthodes [WLS+19][PNH+19][LF19][YWC+19] ont par la suite repris ce schéma que nous résumons ici :

1. Les points du nuage de points S de taille N, représenté dans la figure 33 (a), sont passés en entrée d'un réseau *PointNet++* [QYSG17] afin de prédire un *vecteur de caractéristiques ponctuelles* $\mathbf{F}_P$ pour chacun d'entre eux.

2. A partir des caractéristiques $\mathbf{F}_P$, le réseau apprend les poids d'une fonction de similarité D_{SIM} définie comme ceci :

$$D_{SIM} : \left| \begin{array}{l} \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R} \\ \mathbf{P}_i \times \mathbf{P}_j \rightarrow D_{SIM}(P_i, P_j) \end{array} \right. \tag{2.44}$$

Cette fonction de similarité est définie selon 3 cas différents :

$$D_{SIM}(\mathbf{P}_1, \mathbf{P}_2) = \begin{cases} D_0 \text{ si } \mathbf{P}_1 \text{ et } \mathbf{P}_2 \text{ n'appartiennent pas à la même classe} \\ D_C \text{ si } \mathbf{P}_1 \text{ et } \mathbf{P}_2 \text{ appartiennent à la même classe mais pas au même objet} \\ D_{Obj} \text{ si } \mathbf{P}_1 \text{ et } \mathbf{P}_2 \text{ appartiennent au même objet} \end{cases} \tag{2.45}$$

et la fonction de similarité doit vérifier : $D_0 > D_C > D_{Obj} > 0$.

3. Pour chaque point $\mathbf{P}$ la fonction D_{SIM} est calculée par rapport à tous les autres points du nuage de points S afin d'obtenir une matrice de différence M_D de taille $N \times N$, représentée en rouge dans la figure 33 (b).

4. Un algorithme de *clustering* ou du *MeanShift* est appliqué sur la matrice M_D afin d'obtenir le partitionnement du nuage de points S en groupes où chaque groupe représente une instance d'objet. Un exemple de partitionnement est représenté dans la figure 33 (c).

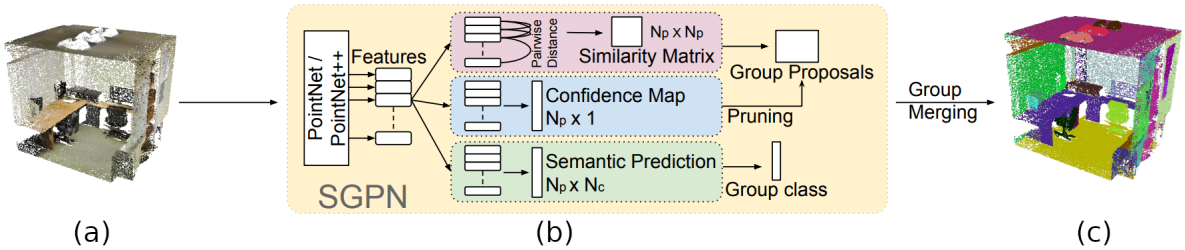

FIGURE 33 – Architecture globale de *SGPN* : à gauche le nuage de points en entrée et à droite le résultat de la segmentation d'instances. Au milieu les différentes opérations calculées, de haut en bas : la matrice distance entre chaque point, un score de confiance pour chaque point, une segmentation sémantique. Figure tirée de [WYHN18].

Les méthodes de cette famille ne sont pas limitées par l'approximation causée par la définition de boîtes englobantes et permettent ainsi un niveau de segmentation plus fin que les précédentes. Néanmoins, elles ont une limite majeure du fait qu'une estimation du nombre maximal d'objets par scène doit être possible, ce qui est n'est pas une hypothèse très restrictive dans le cas de scènes en intérieur mais qui le devient dans le cas de scènes en extérieur.

2.7 Bilan de l'état de l'art

Les premières approches de détection d'objets dans un nuage de points adaptées du réseau *PointNet* [QSMG17], que nous présentons en section 2.6, ne correspondent pas à notre problème. Bien que la définition de la détection d'objets est conforme avec celle que nous donnons au chapitre 1, nous constatons plusieurs différences avec la nature des scènes pour lesquelles ces méthodes sont développées.

Pour rappel, les scènes sur lesquelles nous travaillons sont des nuages de points issus d'acquisition *LiDAR*, dites *MLS*, en milieu urbain. Ainsi les nuages de points ont plusieurs caractéristiques en commun :

- Taille : les nuages de points sont plutôt de grande taille, représentant plusieurs centaines de mètres linéaires d'acquisition voir quelques kilomètres. Cela se traduit en terme de nombre de points par des nuages de plusieurs millions de points.
- Densité : c'est le paramètre qui varie le plus selon le dispositif utilisé. Cependant les nuages sont toujours de haute densité, c'est-à-dire que même les objets les moins volumineux, comme par exemple les panneaux de signalisation, sont représentés par au moins quelques centaines de points.
- Objets : des objets de plusieurs classes différentes sont présents dans chaque scène, voir la diversité des objets urbains présentée en chapitre 1. Quant au nombre de ces objets, celui-ci peut varier d'une scène à l'autre mais certaines catégories d'objets sont toujours présentes en quantité comme par exemple les voitures garées, les lampadaires ou bien les arbres d'alignement le long des routes.

En fait, nous trouvons peu de travaux sur la détection d'objets urbains avec ce type de scène. Ces derniers concernent exclusivement la segmentation sémantique [HSL+17][RDG18]. Comme discuté précédemment, les travaux en détection d'objets dans des nuages de points 3D se concentrent alors sur d'autres types d'application : les scènes en intérieur ainsi que les voitures autonomes. Chacune de ces deux applications présente des défis et des caractéristiques qui lui sont propres. Dans le cas des scènes en intérieur, celles-ci sont de petites taille, quelques mètres linéaires tout au plus, de densité comparable à nos scènes et comptent un nombre d'objets limité qui sont repartis entre plusieurs classes (table, chaise...). Un des défis principaux de ces scènes est la petite taille des objets qui peuvent s'y trouver tels que des écrans d'ordinateur ou bien des vidéo-projecteurs. Dans le cas des voitures autonomes, les scènes utilisées sont de grandes taille mais de densité très faible. Ce qui s'explique par la contrainte du traitement en temps réel dans ce cas d'application. En effet, les nuages de points correspondent à une acquisition *LiDAR* à un instant précis [GLU12] (pour être en mesure de réagir en temps réel à l'environnement) alors que dans notre cas la détection d'objets s'opère *a posteriori*, une fois que l'acquisition a été réalisée en entier. De plus ces travaux [GLU12] ne s'intéressent qu'a 3 classes d'objets : voitures, piétons et cyclistes.

C'est pourquoi nous avons décidé de travailler sur la conception d'une méthode de détection d'objets dans des nuages de points, basée sur le réseau *PointNet* [QSMG17], qui soit adaptée aux scènes sur lesquelles nous travaillons. En d'autres termes, nos travaux consistent à développer un module qui permet de passer d'opérations au niveau point à des opérations au niveau région et qui vérifie les propriétés suivantes :

1. Utilise uniquement le nuage de points en entrée contrairement aux méthodes du 1.5.1.

2. Est adapté aux grandes scènes (contrairement aux scènes d'intérieur des méthodes du 1.5.2 et 1.5.3) qui comportent plusieurs classes d'objets en grande quantité (contrairement aux scènes adaptées aux voiture autonomes des méthodes du 1.5.1 et 1.5.2).

3. Utilise la même définition de régions 3D que nous (un sous ensemble de points du nuage) : contrairement aux sous-boîtes englobantes 3D définies pour chaque point des méthodes du 1.5.2.

Dans le chapitre suivant nous décrivons notre proposition d'architecture permettant de vérifier toutes ces propriétés.

Chapitre 3

Architecture par clustering

Dans ce chapitre nous présentons l'architecture du réseau neuronal que nous avons développé dans le cadre de nos recherches sur la détection d'objets dans des nuages de points. La structure globale du réseau de neurones ainsi que ses fondements conceptuels sont explicités dans la section 3.1. Dans les sections 3.2 et 3.3 nous détaillons les deux briques les plus importantes de notre opération clé. Dans la section 3.4 nous décrivons le fonctionnement de la prédiction de boîtes englobantes à partir de notre architecture. Une liste des symboles () utilisés dans ce chapitre est disponible à la fin de celui-ci.

3.1 Description de notre architecture générale de *RPN 3D*

Nous rappelons que nous avons formalisé le problème de détection d'objets dans un nuage de points S, par la définition d'une fonction de prédiction de boîtes englobantes appelée $Detec$:

$$Detec : \left| \begin{array}{l} \mathbb{R}^{3\times N} \rightarrow \left(\mathbb{R}^6 \times \mathbb{N}\right)^{N_B} \\ S \rightarrow \{\mathbf{B}_i, C_i\}_1^{N_B} \end{array} \right. \tag{3.1}$$

où les $\mathbf{B}_i$ représentent les boîtes englobantes et N_B est leur nombre total. Il s'agit d'une formulation simplifiée de l'équation 1.7.

Dans cette section nous présentons nos travaux sur la prédiction des boîtes englobantes. Afin de réaliser cette tâche nous nous sommes basés sur la prédiction de boîtes englobantes 2D par les RPN [RHGS17].

3.1.1 Une opération manquante pour la généralisation du *RPN*

Dans la figure 34 en haut, nous schématisons le fonctionnement des RPN par l'application de 3 opérations successives. Les deux premières sont assurées par les *convolutions à pas* qui permettent d'extraire automatiquement des *caractéristiques* (a) des images ainsi que *d'agglomérer*

ces *caractéristiques* (b). La 3ème opération est la prédiction de boîtes englobantes calculée par l'action combinée du $classifieur$ et de la *régression 2D* (c).

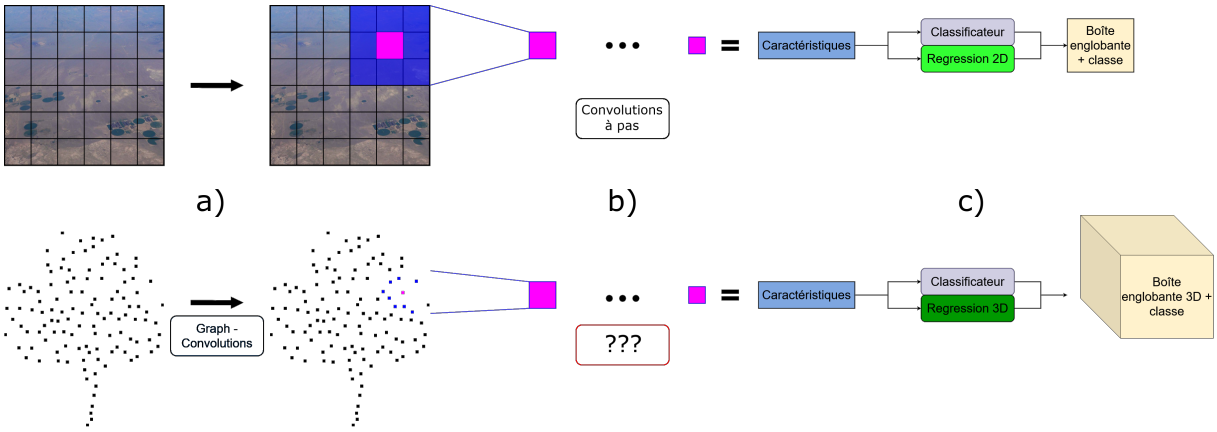

FIGURE 34 – Présentation des opérations nécessaires pour la généralisation des architectures *RPN* aux nuages de points 3D. Le passage d'une représentation par pixels, à une représentation par régions est permis par l'opération de convolution à pas. Nous n'avons pas trouvé dans la littérature une opération de *convolution à pas* définie dans le cas des nuages de points.

Des convolutions à pas adaptées aux nuages de points ?

Dans notre cas, seule la première et la dernière opération (a) (c) sont clairement définies sur les nuages de points. Ainsi, comme nous le montrons dans les sections 2.5.1 et 2.6, la régression et la classification de boîtes englobantes peuvent facilement être étendues aux boîtes englobantes 3D. Dans la section 2.5.2, nous montrons que des méthodes existantes permettent d'extraire automatiquement des caractéristiques d'un point du nuage en prenant en compte son contexte local. Cependant ces méthodes ne permettent pas d'agréger les caractéristiques extraites, ou, quand elles le font, n'assurent pas le même lien entre *vecteurs de caractéristiques* agglomérés et nuage de points initial que les *convolutions à pas*. Il nous manque alors une *opération*, que nous représentons en rouge dans la figure 34 (b), qui permettrait d'assurer cette *liaison* entre points et régions, en même temps que l'extraction automatique des caractéristiques. Nous devons alors passer par la définition de cette *opération* pour pouvoir généraliser les RPN aux nuages de points 3D.

3.1.2 Une brique centrale de notre architecture : la *cluster-convolution*

Nous proposons l'opération de *cluster-convolution*, notée $ClConv$, pour généraliser les RPN aux nuages de points 3D. Cette opération peut s'appliquer directement sur un nuage de points et permet de généraliser les *convolutions à pas*.

Dans la suite nous notons S_l un ensemble de N_l vecteurs $\boldsymbol{\mu}^l$. Chaque vecteur $\boldsymbol{\mu}^l$ est de dimension $3 + d_l$. Dans le cas où $l = 0$: l'ensemble S_0 est alors le nuage de points initial avec

$\boldsymbol{\mu}_i^0 = (x_i, y_i, z_i)$, $\forall_{0<i<N_0}$, et $N_0 = N$. Chaque ensemble S_l suivant correspond à l'application d'une opération $ClConv$ définie comme ceci :

$$ClConv : \left| \begin{array}{l} \mathbb{R}^{N_l \times (3+d_l)} \longrightarrow \mathbb{R}^{N_{l+1} \times (3+d_{l+1})} \\ S_l \longrightarrow S_{l+1} \end{array} \right. \tag{3.2}$$

où $S_l = \{\boldsymbol{\mu}_i^l\}_{i=1}^{i=N_l}$ avec $\boldsymbol{\mu}_i^l \in \mathbb{R}^{3+d_l}$. Ainsi S_l est la sortie de la $l$ème opération de $ClConv$ et S_{l+1} est la sortie de la $l+1$ème opération de $ClConv$. C'est autour de cet enchaînement d'opérations $ClConv$, appliquées séquentiellement, que va s'articuler l'architecture de notre réseau. Celles-ci ont pour effet de réduire progressivement la taille de l'entrée tout en augmentant le nombre de caractéristiques extraites.

Nous présentons le schéma de notre architecture dans la figure 35. Un nuage de points S_0 (*Fig 35(a)*) est passé en entrée du réseau. Des opérations $ClConv$, encadré rouge dans la figure 35, sont appliquées sur le nuage de points. Chaque opération *ClConv*, qui prend en entrée un nuage S_l, est composée de 3 couches différentes :

(a) une couche de *clustering* qui va regrouper les éléments de S_l en N_{l+1} groupes, notés K_i^{l+1}, ce qui implique $N_l > N_{l+1}$.

(b) une couche de *graph-convolution* qui va convoluer dans chaque groupe K_i^{l+1} et ainsi en extraire des caractéristiques.

(c) une couche *d'agrégation maximale* afin d'obtenir pour chaque groupe K_i^{l+1} un vecteur $\boldsymbol{\mu}_i^{l+1}$ de dimension $3 + d_{l+1}$ avec $d_{l+1} \geq d_l$. A noter que chaque vecteur $\boldsymbol{\mu}_i^{l+1}$ est associé à un groupe K_i^{l+1}.

La sortie de la dernière opération $ClConv$ (*Fig(e)*), est l'ensemble $S_L = \{\boldsymbol{\mu}_i^L\}_{i=1}^{i=N_L}$ et grâce aux associations entre $\boldsymbol{\mu}_i$ et K_i, il est possible, à partir de chaque $\boldsymbol{\mu}_i^L$, de remonter d'étape en étape jusque la région R du nuage de points original concernée par $\boldsymbol{\mu}_i^L$, voir ligne en pointillé bleu dans la figure 35.

Ces vecteurs $\boldsymbol{\mu}_i^L$ sont ensuite passés en entrée d'un $classifieur$ permettant de prédire, pour chaque $\boldsymbol{\mu}_i^L$, la classe et la *boîte englobante* 3D autour de l'objet présent dans la région R du nuage de points original S_0 associé. L'architecture est ainsi séparée en deux modules calqués sur ceux du *RPN* et dont le fonctionnement doit être adapté aux nuages de points 3D.

1

Le premier module est séparé en deux couches d'opérations que nous présentons dans les sections 3.2 et 3.3. Le second module est présenté dans la section 3.4 et a pour but la prédiction des *boîtes englobantes*.

Le premier module de notre architecture a deux objectifs :

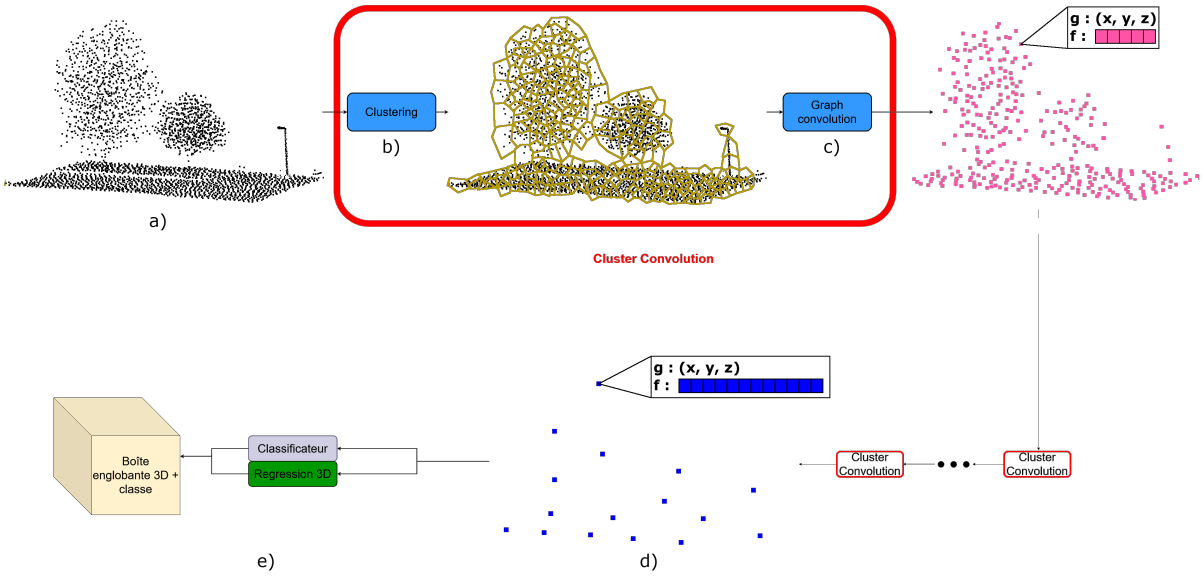

FIGURE 35 – Schéma global de notre architecture. Sont illustrées les opérations appliquées sur le nuage de points en entrée (*a*) : (clustering (*b*), graph-convolution (*c*), ainsi que la prédiction de boites englobantes par *classification* et *régression* (*e*).

- partitionner le nuage de points en différents groupes disjoints (voir figure 35 (*b*) ; section 3.2)
- extraire des caractéristiques apprises par le réseau pour chacun de ces groupes (voir figure 35 (*c*) ; section 3.3)

Ces opérations doivent être implémentées par des fonctions dérivables afin d'être intégrées à un réseau de neurones et d'apprendre les caractéristiques extraites des nuages de points.

3.2 La couche de *clustering*

Cette couche a pour but de regrouper les points afin de pallier le manque de structuration dans les nuages de points. Nous exposons ici les questions soulevées par l'élaboration d'une telle couche ainsi que les choix auxquels nous nous sommes confrontés lors de son développement et de son implémentation.

3.2.1 Spécification de l'opération

L'opération de *clustering* des points assure la diminution de la taille des nuages de centroïdes en entrée et permet ainsi le passage d'une analyse centrée sur les points à une analyse au niveau région. Dans les réseaux de neurones appliqués aux images 2D, typiquement les *RPN*, cette opération est quasi inhérente à la nature structurée des images : le voisinage de chaque pixel peut être considéré comme une région. Il suffit alors de décider des paramètres de ce groupement implicite : taille des noyaux de *convolutions* et pas des *convolutions à pas*. Dans le cadre des

nuages de points où ni voisinage implicite ni ordre ne sont définis sur les points, la définition d'une fonction de *clustering* est alors nécessaire.

Cette fonction doit satisfaire deux contraintes :

1. Lier entre eux les points suffisamment proches, géométriquement ou sémantiquement.
2. Contrôler le taux de recouvrement de ces groupes de points, ie. le nombre de groupes doit être inférieur strictement au nombre de points.

Une dernière contrainte est que la fonction doit pouvoir être intégrée au sein de l'architecture d'un réseau de neurones, c'est-à-dire être décomposable en fonctions dérivables.

Définition du *clustering*

Nous définissons le regroupement, ou *clustering*, d'un nuage S_l par une fonction $f_{Clustering}$ comme il suit :

$$\mathbf{g}f_{Clustering} : \left| \begin{array}{l} \mathbb{R}^{N_l \times d_l} \rightarrow \mathbb{R}^{n \times d_l} \\ S_l \rightarrow \{K_1, ...K_n\} \end{array} \right.$$

$$\bigcup_{i=1}^{N_{l+1}} K_i = S_l$$

$$\bigcap_{i=1}^{N_{l+1}} K_i = \emptyset$$

c'est-à-dire par la définition d'un nombre n de groupes, appelés *clusters* et notés K_i, d'éléments de S_l. Ces clusters doivent former une partition de S_l. Remarque : $n = Card(S_{l+1})$.

3.2.2 Choix de la méthode de clustering

Afin de réaliser le *clustering* des points, nous proposons d'utiliser l'algorithme du *k-means* [Llo82]. Parmi tous les algorithmes de clustering existants [Jai10], nous avons choisi le *k-means* car il s'agit de la méthode la plus simple et la plus utilisée dans le cas d'intégration dans un réseau de neurones profond [MGL+18].

L'algorithme du *k-means* est un algorithme glouton et itératif, qui cherche à partager un espace, ici un nuage de points, en un nombre fixe de *clusters*. Se reporter à l'algorithme 1 pour une présentation d'une version simplifiée et appliquée aux nuages de points. Du fait de sa relative simplicité, l'intégration du *k-means* à l'intérieur d'un réseau de neurones est possible. Nous détaillons celle-ci dans la section 3.2.4.

Algorithm 1 : K-means appliqué aux nuages de points

Data : $S_l = \{\mathbf{P}_1 \ldots \mathbf{P}_{N_l}\}$; N_{l+1} nombre de *clusters* ; $iMax$ nombre d'itérations
Result : $\mu = \{\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_{Nl+1}\}$ *centroïdes* de chaque *cluster*, assignation des *clusters*
initialisation ;
for $1 \leq i \leq N_{l+1}$ **do**
  $\boldsymbol{\mu}_i \leftarrow \mathbf{P}_j$, avec j tiré au hasard sans redondance ;
$iter \leftarrow 0$;
while $iter \leq iMax$ **do**
  for $1 \leq i \leq N_l$ **do**
    $A \leftarrow argmin_j(Distance(\mathbf{P}_i, \boldsymbol{\mu}_j))$;
    for $1 \leq j \leq N_{l+1}$ **do**
      $K_j \leftarrow gatherPoints(A, S_l)$;
    for $1 \leq j \leq N_{l+1}$ **do**
      $\boldsymbol{\mu}_j \leftarrow moyenne(K_j)$;
  $iter \leftarrow iter + 1$

Où A est la *matrice d'assignation*, la fonction *gatherPoints* permet de récupérer tous les points associés à un même *centroïde* et la fonction *moyenne* permet de calculer la valeur géométrique et la valeur caractéristique d'un *centroïde* en fonction des points qui lui sont associés.

Cet algorithme comporte tout de même certains points négatifs par rapport à notre objectif. Le nombre de régions ou *clusters* étant fixe, se pose le problème de l'adéquation entre le nombre de régions choisies et le nombre d'objets présents dans les scènes à analyser. En effet, si le nombre de *clusters* est moins grand que le nombre d'objets présents dans le nuage de poins, alors il y aura forcément des *clusters* composés de plusieurs objets différents, ce qui complique la phase de détection (plus de détails à ce sujet en section 3.3). Un autre paramètre du *k-means* est la fonction de distance utilisée afin d'assigner à chaque points un *cluster*.

3.2.3 Fonctions de distance

Dans le cas d'un algorithme du *k-means* appliqué au nuage de points, une fonction de distance euclidienne est habituellement utilisée. Cependant dans notre cas, puisque l'opération de *clustering* est appliquée successivement sur le nuage de points, comme les couches de convolutions dans les *réseaux de neurones convolutif*, les points donnés en entrée de l'algorithme ne sont pas simplement des coordonnées géométriques, mise à part lors de la première couche, mais des *vecteurs*. Ces *centroïdes* (*vecteurs*) sont calculés dans la couche de *clustering* précédente et sont composés d'une partie *caractéristiques*, notée **f**, et de coordonnées 3D, notée **g** :

$$\forall \boldsymbol{\mu} \in S_l \text{ avec } l \geq 1, \boldsymbol{\mu} = (\mathbf{g}, \mathbf{f}) \tag{3.3}$$

avec $\boldsymbol{\mu} \in S_l$, le résultat de la $l$ième couche de *cluster-convolution* comme défini en (Eq 3.17), $\mathbf{g} \in \mathbb{R}^3$ et $\mathbf{f} \in \mathbb{R}^{d_l}$.

Ainsi, les *centroïdes* sont représentés par des vecteurs composés de deux parties : une partie géométrique, les coordonnées du *centroïde* de la région, ainsi qu'une partie définie dans un espace en haute dimension, les *caractéristiques* extraites pour cette région. Il est donc nécessaire de définir une distance entre ces *centroïdes* qui puisse prendre en compte ces deux parties. Une première idée est la distance euclidienne définie comme ceci :

$$D_e(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \left\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right\| \tag{3.4}$$

où $\|\mathbf{V}\|$ représente la norme du vecteur $\mathbf{V}$.

Cependant, l'utilisation d'une distance euclidienne D_e calculée sur chaque composante rendrait négligeable les parties géométriques des *clusters* du fait du déséquilibre entre les deux parties $\mathbf{g}$ et $\mathbf{f}$ au niveau du nombre de composantes. En effet, la partie géométrique compte tout au plus trois composantes, x, y et z, alors que la partie caractéristique peut en compter plusieurs milliers selon le paramétrage de l'opération de *cluster-convolution*. Étant donné que chaque composante des vecteurs $\mathbf{g}$ et $\mathbf{f}$ sont normalisées dans un intervalle $[0, 1]$, du fait de la normalisation dans une sphère $(0, 1)$ des nuages de points pour le vecteur $\mathbf{g}$ et du fait de la *normalisation par lot* (*batch normalization*) pour le vecteur $\mathbf{f}$, une simple distance euclidienne n'est pas adaptée au calcul de distance entre deux vecteurs $\boldsymbol{\mu}_1$ et $\boldsymbol{\mu}_2$.

Toute la problématique du choix d'une fonction de distance réside alors dans la recherche d'un compromis entre d'une part la similarité des points au niveau des caractéristiques et d'autre part leurs proximités dans l'espace géométrique. A cela il faut ajouter le fait que la définition d'une distance dans un espace vectoriel de grande dimension est en soi un problème difficile. Nous avons exploré deux manières, ne s'excluant pas mutuellement, d'aborder ces problèmes lors de la définition d'une fonction de distance.

Distance normalisée

Comme son nom l'indique, l'approche par normalisation revient à normaliser les valeurs des *centroïdes* entre elles afin d'effacer la prépondérance de la partie caractéristique par rapport à la partie géométrique, ainsi que la prépondérance au sein même de la partie caractéristique de certaines composantes qui peuvent éclipser les autres. Deux approches par normalisations ont été explorées. La première est une normalisation classique, utilisée par exemple pour les *analyses en composantes principales (ACP)*, qui consiste pour chaque série de données, ici les points, à soustraire la moyenne m et diviser par la variance σ^2 :

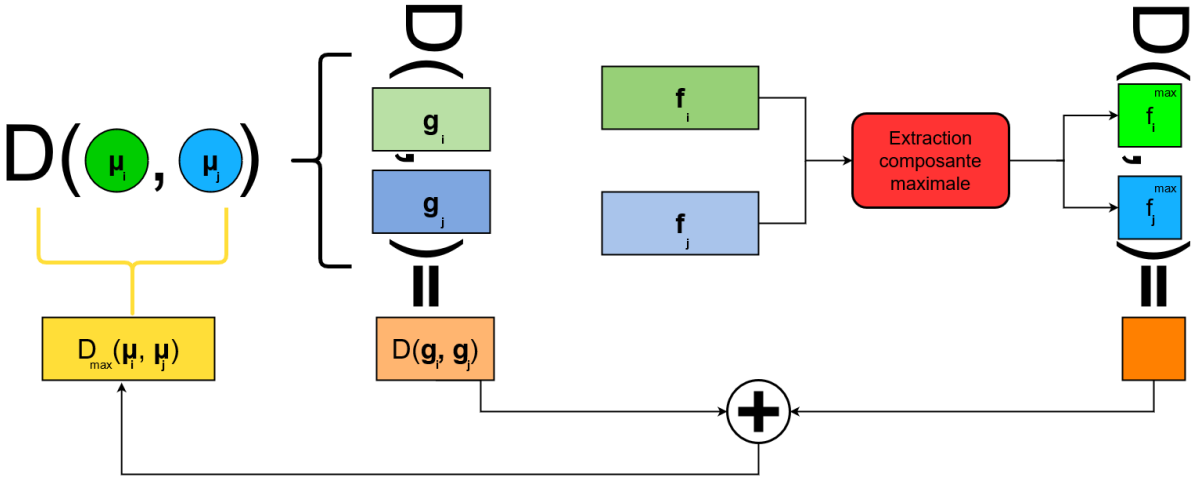

FIGURE 36 – Schéma illustrant le calcul de la fonction de distance par caractéristique maximale entre un point, en vert, et un *centroïde*, en bleu. Tout d'abord, la distance euclidienne des deux parties géométriques **g**, en vert et bleu clair, est calculée, orange clair. Puis, les composantes maximales des parties caractéristiques **f**, carrés vert et bleu à droite de la figure, sont extraites. La différence entre les composantes maximales est ensuite calculée, carré orange, puis additionné avec la distance des parties géométriques, en bas de la figure, pour donner la distance finale, en jaune.

$$D_{norm}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \frac{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|}{\sigma^2} - m \tag{3.5}$$

où m et σ^2 désignent respectivement la moyenne et la variance de la série formée par les caractéristiques des points du nuage. La seconde consiste à calculer la *distance de Mahalonobis* entre chacun des points (Eq 3.6), ce qui revient à calculer les covariances pour chacune des composantes des *centroïdes* [Mah36]. La *distance de Mahalanobis* est définie comme ceci :

$$D_M(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \sqrt{(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T \mathscr{M}^{-1} (\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)} \tag{3.6}$$

où $\mathscr{M}$ désigne la matrice de covariance de la série associée, dans notre cas il s'agit du nuage de points.

Calcul disjoint

Le calcul disjoint de la fonction de distance revient à séparer les deux parties des *centroïdes* et effectuer le calcul de la distance indépendamment pour chaque partie. Une façon simple de le mettre en oeuvre est de calculer une *distance euclidienne* pour chacune des deux parties caractéristiques et géométriques, **f** et **g**, puis de calculer une somme pondérée de ces deux distances :

$$D_\lambda(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \lambda \times D_e(\mathbf{g}_i, \mathbf{g}_j) + (1 - \lambda)(D_e(\mathbf{f}_i, \mathbf{f}_j)) \tag{3.7}$$

où λ est le coefficient choisi, $\mathbf{g}_i$ et $\mathbf{f}_i$ sont respectivement les parties géométriques et caractéristiques des points.

Nous avons proposé une autre approche appelée *distance par caractéristique maximale*. L'idée est de s'appuyer sur le fait que le processus d'apprentissage du réseau *PointNet* [QSMG17], dont nous nous inspirons, repose sur la définition de *points critiques*. Ces *points critiques* sont pour la plupart situés à des endroits stratégiques, tels que les bordures d'un objet ou bien les extremums, et permettent au réseau de reconnaître des formes particulières. Ainsi nous faisons l'hypothèse que deux *points critiques* proches au niveau spatial ont plus de chance d'appartenir à un même objet et nous nous basons sur le calcul de la composante maximale des vecteurs de *caractéristiques* $\mathbf{f}$ afin d'estimer la probabilité pour qu'un vecteur $\boldsymbol{\mu}$ soit un *point critique*.

Nous illustrons cette approche dans le schéma de la figure 36. Celle-ci consiste, dans un premier temps, à calculer la différence entre les parties géométriques $\mathbf{g}$, puis dans un deuxième temps à extraire les composantes maximales des parties *caractéristiques* $\mathbf{f}$ et de calculer leurs distances euclidiennes respectives. Finalement les deux différences sont ensuite additionnées. Nous résumons ce calcul dans l'équation 3.8 :

$$D_{\max}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \lambda - D_e(\mathbf{g}_i, \mathbf{g}_j) + (1 - \lambda) D_e\left(\max_{f_i^h} \mathbf{f}_i, \max_{f_j^h} \mathbf{f}_j\right) \tag{3.8}$$

où f_i^h représente la $h$ème composante du vecteur de *caractéristiques* $\mathbf{f}_i$.

3.2.4 Implémentation de la couche de clustering

Cette couche ne doit pas perturber la rétropropagation des gradients lors de l'apprentissage du réseau. La contrainte principale est donc la décomposition de l'algorithme du *k-means*, appliqué à des points ou des points-régions, en succession d'opérations dérivables, s'assurant ainsi du bon fonctionnement de l'apprentissage.

Tout d'abord, N_l points appartenant au nuage de points S_{l-1} sont sélectionnés. Ces points correspondent à l'étape d'initialisation des *centroïdes*. La sélection des points est typiquement réalisée par un tirage aléatoire mais il est possible de sélectionner simplement les N_l premiers points si les nuages subissent déjà des permutations aléatoires comme pré-traitements.

Ensuite la partie itérative de l'algorithme est amorcée ; celle-ci débute par le calcul des distances de chaque point à chaque *centroïde*. Un *vecteur d'assignation* est alors construit en sélectionnant pour chaque point le *centroïde* le plus proche. A partir de ce vecteur, les points qui partagent la même assignation sont regroupés dans les *clusters* associés à celle-

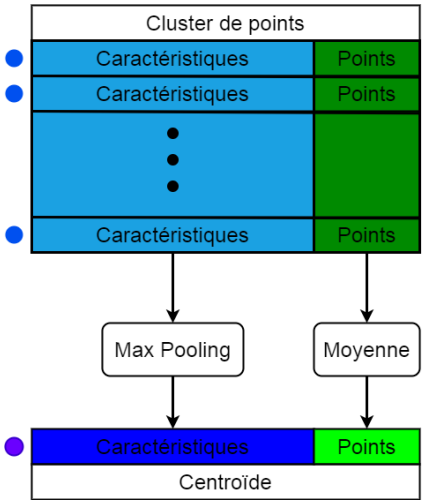

FIGURE 37 – Schéma de notre méthode de calcul du *centroïde* pour un *cluster*. Le *cluster* est représenté sous la forme d'une liste de points, en haut, avec les parties caractéristiques f_i des points en bleu et les parties géométriques g_i des points en vert. Le *centroïde* du *cluster* est un vecteur, en bas, avec pour partie caractéristique, en bleu foncé, les composantes maximales des f_i et pour partie géométrique, en vert clair, la moyenne des g_i.

ci. Ce regroupement est réalisé à l'aide d'un procédé algorithmique inspiré de [VBV18]. A partir de ces *clusters* il est alors possible de mettre à jour la valeur des *centroïdes*. Le calcul des nouvelles valeurs des *centroïdes* est illustré dans la figure 37. Pour chaque *cluster* K, la moyenne des parties géométriques $\mathbf{g}$, en vert foncé dans la figure 37, des points appartenant au *cluster* K est calculée. Le résultat du calcul de cette moyenne est la nouvelle valeur de la partie géométrique $\mathbf{g}$ du nouveau *centroïde* associé au *cluster* K, en vert clair dans la figure 37. Une opération d'agrégation maximale, composante par composante, est ensuite calculée sur les partie caractéristiques $\mathbf{f}$, en bleu clair dans la figure 37, des points appartenant au *cluster* K. Le *vecteur de caractéristiques* maximal qui en découle est la nouvelle valeur de la partie *caractéristique* $\mathbf{f}$ du nouveau *centroïde* associé au *cluster* K, en bleu foncé en bas dans la figure 37.

Lors de la dernière itération, les dernières valeurs des *centroïdes* sont retournées ainsi que les dernières assignations afin de pouvoir faire le lien entre le nuage de points en entrée et les *clusters* déterminés.

Afin de pouvoir être intégré à un réseau de neurones, cet algorithme doit être décomposable en une succession d'opérations dérivables, tels que des additions ou des soustractions par exemple. Ici, seule la fonction *gatherPoints* de l'algorithme 1 pose problème. En effet, puisque nous ne connaissons pas à l'avance le nombre d'éléments de chaque *cluster*, construire itérativement ceux-ci semble être l'option à privilégier. Cependant cette opération, triviale d'un point de vue algorithmique, n'est pas décomposable en opérations dérivables. Une astuce consiste alors à fixer un nombre maximal d'élément pour tous les *clusters*, puis d'utiliser des masques binaires afin de gérer les éléments *vides* [VBV18].

3.3 La couche de *graph-convolution*

La seconde moitié de l'opération de *cluster-convolution*, c'est-à-dire la couche de *convolutions sur graphe* (*graph convolution*), prend en charge toute la partie *extraction de caractéristiques* du réseau de neurones. C'est dans celle-ci que se trouve la majeure partie des variables apprises par le réseau afin d'extraire les *caractéristiques* qui serviront à reconnaître des formes dans les nuages de points.

Ici nous présentons les difficultés spécifiques à l'intégration de la prise en compte du contexte spatial des points dans la conception d'une couche de convolutions en 3D.

3.3.1 Définition

Nous résumons la sortie de la couche de *clustering* dans la figure 38. A partir d'un nuage en entrée (a) nous obtenons un nuage de points partitionnés en N_l *clusters* noté K^l, voir figure 38 (b). Chacun des éléments $\boldsymbol{\mu} \in S_l$ est un vecteur avec une partie *géométrique* et une partie *caractéristique* (Eq 3.3). De plus pour chacun de ces *clusters* est calculé un centroïde, voir figure 38 (c). Dans la suite, nous détaillons comment apprendre à extraire des *caractéristiques* à partir de ces vecteurs.

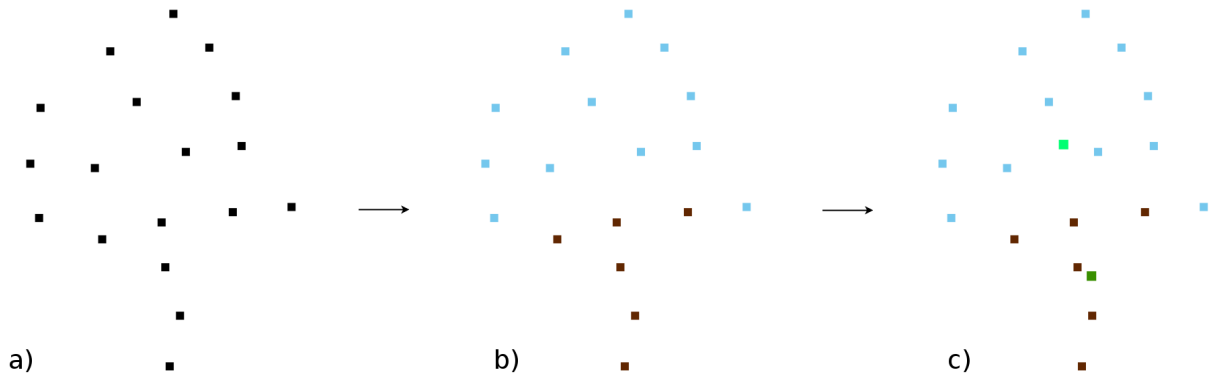

FIGURE 38 – Schéma récapitulatif de la couche de *clustering*. Un nuage de points (ou de centroïdes) est passé en entrée (a), le nuage est partitionné en un nombre fixe de *clusters* (b) et un *centroïde* est calculé pour chaque *clusters* (c).

L'approche la plus évidente pour extraire ces *caractéristiques* est d'appliquer directement le réseau *Pointnet* [QSMG17] sur chacun des *clusters*. Cependant, comme les *caractéristiques* extraites par ce réseau ne prennent pas en compte le contexte local de chaque point, nous ne pouvons pas nous appuyer directement sur le mécanisme utilisé dans le réseau *PointNet* [QSMG17]. Pour plus de détails à ce sujet, se reporter à la section 2.5.2.

Une manière plus adaptée est alors de s'inspirer des approches *post PointNet*, voir section 2.5.2, afin de définir un opérateur de *graph-convolution*, que nous noterons par la suite $GConv$. Nous présentons la différence entre ces deux approches dans la figure 39. Dans notre cas, l'opérateur doit prendre en entrée un nuage de points correspondant à un *cluster* et lui associer

un unique point (X, Y, Z) ainsi qu'un vecteur de *caractéristiques*. Contrairement à *PointNet* [QSMG17] où le réseau calcule un vecteur de caractéristiques à partir du nuage de points en entrée (voir figure 39), l'opérateur défini par *DGCNN* [WSL+19] calcule un vecteur de caractéristique pour chaque point en prenant en compte son contexte local, voir figure 39. Nous définissons alors l'opérateur *GConv* comme ceci :

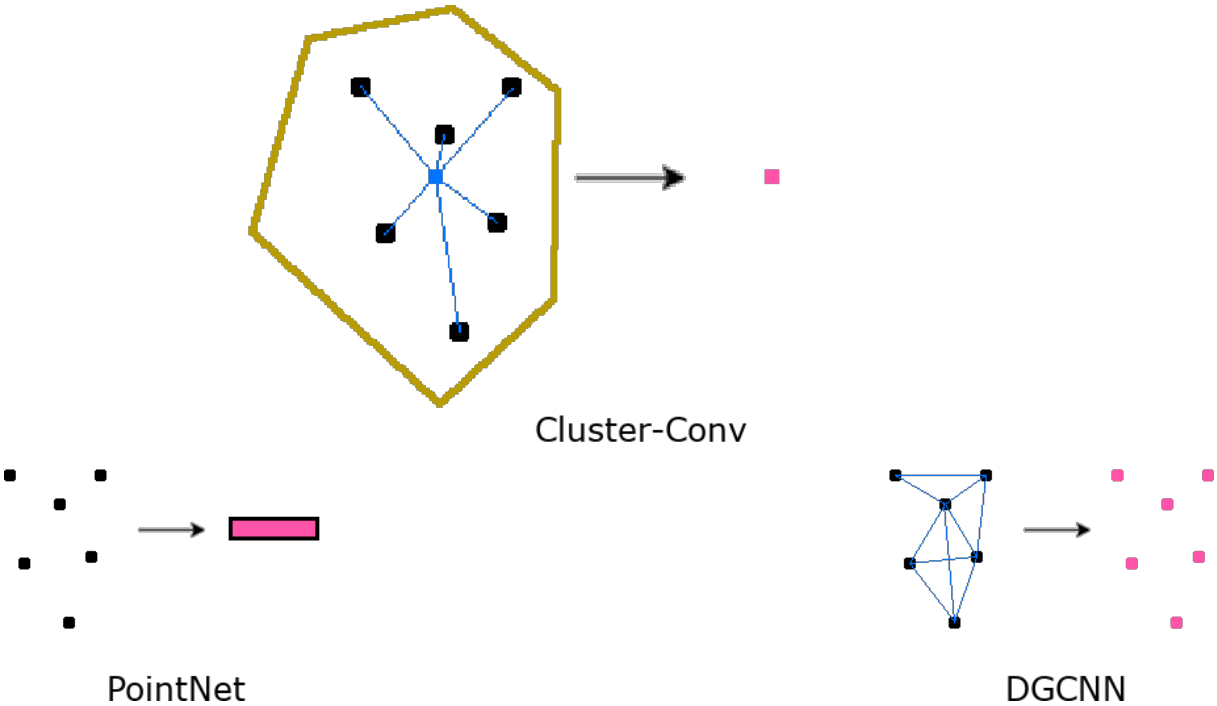

FIGURE 39 – Différence d'approches pour l'extraction de caractéristiques des nuages de points : notre opération a pour but d'extraire un nouveau point à partir d'un *cluster* avec une partie géométrique et une partie *caractéristique* qui prend en compte la répartition des points dans le *cluster*, tandis que l'opération définie par [QSMG17] calcule uniquement un vecteur de caractéristique à partir du nuage de points et que l'opération définie par *DGCNN* [WSL+19] permet d'extraire des caractéristiques pour chaque point en fonction de ses points les plus proches.

$$GConv : \left| \begin{array}{l} \mathbb{R}^{d_l} \times \mathbb{R}^{N_N \times d_l} \rightarrow \mathbb{R}^{d_{l+1}} \\ \mathbf{P} \times \mathcal{N}(\mathbf{P}) \rightarrow \mathbf{F} \end{array} \right. \tag{3.9}$$

où $\mathcal{N}(\mathbf{P})$ est le contexte local du point $\mathbf{P}$, voir exemple de voisinage dans la section 2.5.2, N_N est le nombre de voisins du point $\mathbf{P}$ et $\mathbf{F}$ le *vecteur de caractéristiques* extrait par l'opération *GConv* pour le point $\mathbf{P}$. Nous introduisons la fonction Ψ qui permet d'extraire des *caractéristiques* d'un point $\mathbf{P}$ en fonction d'un autre point $\mathbf{P}_j$:

$$\Psi : \left| \begin{array}{l} \mathbb{R}^{d_l} \times \mathbb{R}^{d_l} \rightarrow \mathbb{R}^{d_{l+1}} \\ \mathbf{P} \times \mathbf{P}_j \rightarrow \mathbf{F} \end{array} \right. \tag{3.10}$$

où $\mathbf{F}$ est le vecteur de *caractéristiques* extrait par la fonction Ψ. Par exemple Ψ peut consister à concaténer le point $\mathbf{P}$ et la différence entre $\mathbf{P}$ et $\mathbf{P}_j$ puis appliquer un *MLP* [WSL+19], voir figure 39. Nous introduisons également l'opérateur d'agglomération générique, noté $\square_i$:

$$\square : \left| \begin{array}{l} \mathbb{R}^{N_l \times d_{l+1}} \rightarrow \mathbb{R}^{d_{l+1}} \\ \{\mathbf{F}_i^l\}_{i=1}^{i=N_l} \rightarrow \mathbf{F}^{l+1} \end{array} \right. \tag{3.11}$$

En pratique, cet opérateur peut être implémenté par exemple, par une somme ou une agrégation maximale, voir figure 39. A partir des fonctions Ψ et $\square$, nous définissons l'application de l'opération $GConv$ sur un point $\mathbf{P}$ et son contexte local, noté $\mathcal{N}(\mathbf{P})$ comme ceci :

$$GConv(\mathbf{P}) = \square_i \{\Psi(\mathbf{P}, \mathbf{P}_i) | \mathbf{P}_i \in \mathcal{N}(\mathbf{P})\} \tag{3.12}$$

Ainsi, les *caractéristiques* extraites par l'opérateur *GConv* dépendent non seulement du point $\mathbf{P}$ mais aussi de son contexte local, c'est-à-dire les points $\mathbf{P}_j \in \mathcal{N}(\mathbf{P})$. Le réseau *PointNet* peut alors être considéré comme un cas particulier de l'opérateur $GConv$ où $\Psi_{PointNet}(\mathbf{P}, \mathbf{P}_i \in \mathcal{N}(\mathbf{P})) = \Psi_{PointNet}(\mathbf{P})$.

3.3.2 Discussion autour des approches existantes

A partir de la définition générique de l'opérateur de graph-convolution (Eq 3.12), nous proposons de passer en revue les propositions existantes de fonctions de *graph-convolution* appliquées à des nuages de points. Les méthodes utilisant un opérateur $GConv$ que nous présentons sont entraînées sur la tâche de segmentation sémantique évoquée en 2.5.2. Nous proposons de les répartir par type de voisinage utilisé et de détailler pour chacune la fonction Ψ utilisée pour extraire les caractéristiques.

Voisinage par distance absolue

Les méthodes basées sur un voisinage par distance absolue proposent d'intégrer dans le calcul des caractéristiques des points, la répartition des autres points du nuage dans un certain rayon ρ.

Les convolutions à noyau sphériques proposent de partitionner l'espace en plusieurs cellules, noté Ω. Pour chaque point $\mathbf{P}$, les points présents dans le voisinage $\mathcal{N}_{Abs}(\mathbf{P})$ sont répartis dans

des cellules selon leurs positions par rapport à **P**. Ces cellules peuvent être définies sous la forme d'une section de sphère ou bien plus simplement de grille cubique comme dans [HTY18]. La fonction Ψ est calculée avec un *poids* w_j pour la cellule Ω_j. L'application de la fonction Ψ est ainsi définie par :

$$\Psi_{cellules}(\mathbf{P}, \mathbf{P}_j) = w_j \frac{1}{Card(\Omega_j)^2} \sum \mathbf{P}_i \in \Omega_j \tag{3.13}$$

où Ω_j est la cellule à laquelle appartient le point $\mathbf{P}_j$, w_j le *poids* associé à la cellule Ω_j et $Card(\Omega_j)$ le nombre de points dans cette cellule. Le principal défaut de ce type d'approche, est que le voisinage $\mathcal{N}_{Abs}(\mathbf{P})$ est discrétisé. Cette discrétisation a pour effet de ne pas prendre en compte la répartition des points dans le voisinage de **P** mais uniquement leurs distances par rapport à **P**. De plus ces distances sont des distances discrètes entre cellules et donc susceptibles de causer des effets de seuil.

Les opérateurs de convolution par noyau de points, comme [TQD+19] ou [Bou20], proposent de s'affranchir de la discrétisation du voisinage grâce à l'utilisation de points dans les noyaux de convolution. Au lieu d'être simplement défini par une liste de poids, chaque noyau de convolution est ainsi défini par une liste de points $\{\mathbf{w}_i^g\} \in \mathbb{R}^3$ avec les poids associés $\{\mathbf{w}_i^f\}$. De même chaque point **P** est composé d'une partie géométrique et d'une partie caractéristique : $\mathbf{P} = (\mathbf{P}^g, \mathbf{P}^f)$, avec $\mathbf{P}^g \in \mathbb{R}^3$ et $\mathbf{P}^f \in \mathbb{R}^{d_l}$. Pour plus de détail sur les convolutions par noyau de points, se reporter à la section 2.5.2 du manuscrit. Ainsi l'application de la fonction Ψ est définie par :

$$\Psi_{KernelPoint}(\mathbf{P}, \mathbf{P}_j) = \frac{1}{Card(W)} \sum_i \mathbf{w}_i^f \Phi(\mathbf{P}_j^g, \mathbf{w}_i^g) \mathbf{P}_j^f \tag{3.14}$$

où $Card(W)$ est le nombre de poids **w** et la fonction Φ permet de faire le lien entre les poids **w** et le contexte local de **P**. Le contexte local de chaque point peut alors être pris en compte sans discrétisation. Cependant le domaine d'application de la fonction Φ, qui est une sphère, n'est pas adapté avec nos *clusters* de points qui ont vocation à être de forme arbitraire et non sphérique.

Voisinage par distance relative

Les méthodes basées sur le voisinage par distance relative proposent de prendre en compte les points qui sont les plus proches, $\mathcal{N}_{Rel}(\mathbf{P})$. Le réseau *FeaStNet* [VBV18], propose d'apprendre une fonction q permettant d'associer à chaque paire $(\mathbf{P}, \mathbf{P}_j \in \mathcal{N}_{Rel}(\mathbf{P}))$, un coefficient associé aux poids **w** de la fonction Ψ. L'application de la fonction Ψ est définie par :

$$\Psi_{FSN}(\mathbf{P}, \mathbf{P}_j) = \mathbf{w} \times q(\mathbf{P}, \mathbf{P}_j) \mathbf{P}_j \tag{3.15}$$

où $\mathbf{w}$ sont les poids de la fonction Ψ, identiques pour tous les points $\mathbf{P}_j$.

Une autre approche plus simple est l'opération proposée par le réseau *DGCNN*. Celle-ci consiste à calculer pour chaque point $\mathbf{P}$, la différence avec tous ses points voisins $\mathbf{P}_j \in \mathcal{N}_{Rel}(\mathbf{P})$ et à appliquer les poids de $\mathbf{w}$ sur ces différences. L'application de la fonction Ψ est ainsi définie par :

$$\Psi_{DGCNN}(\mathbf{P}, \mathbf{P}_j) = \mathbf{w} \times \mathbf{P} \oplus (\mathbf{P}_j - \mathbf{P}) \tag{3.16}$$

où $\oplus$ représente l'opération de concaténation. Nous illustrons un exemple d'application de cette opération dans la figure 39 ; avec en haut une application d'un réseau $PointNet$ où les caractéristiques de chaque point sont calculées indépendamment les unes des autres par des couches *MLP*, alors que dans le cas du réseau $DGCNN$, en bas, le calcul des caractéristiques du point vert prend en compte ses points voisins, en rouge.

Dans notre cas, comme les *clusters* sont construits par rapport à la distance relative entre points et centroïdes, les approches par voisinage relatif sont plus adaptées. Nous avons donc décidé de nous inspirer de l'opération proposée par le réseau $DGCNN$ pour implémenter notre couche de *graph-convolution*, car c'est celle que nous trouvons la plus simple à adapter à notre cas.

3.3.3 Implémentation conjointe avec la couche de *clustering*

Pour rappel, nous définissons l'opération de *cluster-convolution*, notée *ClConv* dans l'équation 3.17. Nous pouvons réécrire cette équation en y injectant la définition générique de l'opérateur *GConv* (Eq 3.12) :

$$ClConv : \left| \begin{array}{l} \mathbb{R}^{N_l \times d_l} \rightarrow \mathbb{R}^{N_{l+1} \times d_{l+1}} \\ S_l \rightarrow \left\{ \underset{j}{\square} \Psi_{ClConv}(\boldsymbol{\mu}_i^{l+1}, \boldsymbol{\mu}_j^l \in K_i^l) \right\}_{i=1}^{i=N_{l+1}} \end{array} \right. \tag{3.17}$$

où K_i^l est le *cluster* dont $\boldsymbol{\mu}_i^{l+1}$ est le centroïde et Ψ_{ClConv} est l'opération de *graph-convolution* calculée par l'opérateur *ClConv*. Nous avons choisi d'implémenter l'opérateur d'agrégation $\square$ par une agrégation maximale et l'opérateur Ψ_{ClConv} comme ceci (voir figure 40) :

$$\Psi_{ClConv} : \left| \begin{array}{l} \mathbb{R}^{d_l} \times \mathbb{R}^{d_l} \rightarrow \mathbb{R}^{d_{l+1}} \\ \boldsymbol{\mu}^{l+1} \times \boldsymbol{\mu}^l \rightarrow MLP\left(\boldsymbol{\mu}_i^{l+1} \oplus (\boldsymbol{\mu}_i^l - \boldsymbol{\mu}_i^{l+1})\right) \end{array} \right. \tag{3.18}$$

où $\boldsymbol{\mu}_i^l$ appartient au *cluster* dont $\boldsymbol{\mu}_i^{l+1}$ est le centroïde et $\oplus$ est l'opérateur de concaténation.

Nous pouvons alors décrire étape par étape le fonctionnement de la couche de *graph-convolution* telle que nous l'avons définie. Chacune de ces étapes est représentée dans la figure 40 :

(a) La couche prend en entrée un nuage décomposé en plusieurs *clusters* avec pour chacun d'entre eux un *centroïde*.

(b) Pour chaque *centroïde*, un nombre de voisins les plus proches du *centroïde* est sélectionné. En pratique, nous choisissons un nombre de voisins supérieur au nombre maximal de points par *clusters*. Cette étape permet simplement de s'assurer que notre opération est dérivable (se reporter à la section 2.2.2 pour plus de détails sur l'importance de la dérivabilité dans un réseau de neurones).

(c) Pour chaque *centroïde*, un vecteur de répartition est construit concaténant le *centroïde* et la différence entre le *centroïde* et chacun de ses voisins.

(d) Ces vecteurs de répartition sont ensuite passés en entrée d'une couche de *perceptron multi-couches* pour en extraire des nouvelles *caractéristiques*.

(e) Finalement, une opération *d'agrégation maximale*, composante par composante, est appliquée afin de conserver uniquement les *caractéristiques* maximales.

(f) Le nouveau nuage S_{l+1} en sortie est construit avec pour chaque $\boldsymbol{\mu}_i^{l+1} \in S_{l+1}$:

- sa partie géométrique est la même que celle du centroïde du *cluster* K_i^l.
- sa partie caractéristique est la sortie de l'opération *d'agrégation maximale*.

Le nuage de points en entrée, en noir, est en premier lieu décomposé en plusieurs *clusters*, deux *cluster* dans la figure 40 : un bleu et un marron. Pour chacun de ces *clusters* un *centroïde* est calculé par le procédé illustré dans la figure 37, les *centroïdes* des deux *clusters* sont représentés par deux points respectivement vert clair et vert foncé dans la figure 40.

Notre opération $ClConv$ dépend ainsi de quatre paramètres : le nombre de *clusters*, le nombre d'itérations de l'algorithme du *k-means*, le nombre de noeuds et la taille du *vecteur de caractéristiques* en sortie. La couche de *clustering* est calculé en première et permet ainsi d'appliquer ensuite une opération de *graph-convolution* directement sur les *clusters*.

3.4 Prédiction de boîtes englobantes à partir des *clusters*

Nous présentons ici le module placé à la fin du réseau de neurones. Celui-ci effectue les prédictions sur la localisation des objets dans le nuage de points ainsi que sur leurs classes. Autrement dit, il s'agit de la couche du réseau qui se charge de la prédiction des *boîtes englobantes* à partir des caractéristiques calculées par l'application successive de *cluster-convolution*.

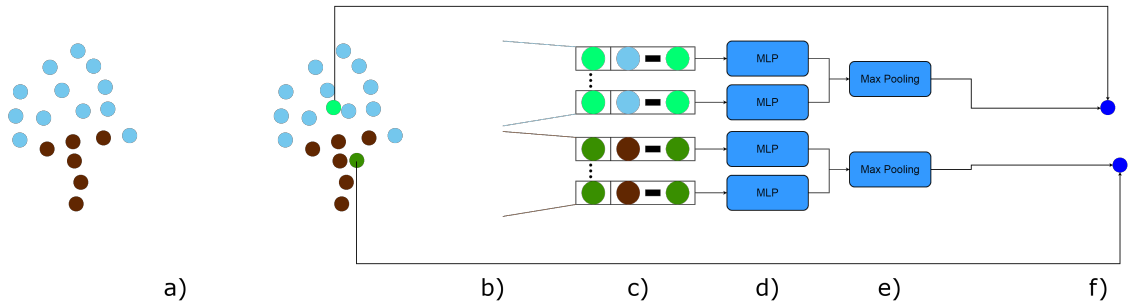

FIGURE 40 – Schéma présentant le fonctionnement de la couche de *graph-convolution*. Un nuage de point partitionné en 2 *clusters* est passé en entrée (*a*), pour chacun de ces *clusters* un *vecteur de caractéristiques* est extrait par l'application de perceptron multi-couches (MLP) (*d*), à poids partagés, sur des vecteurs composés de la concaténation du centroïde ainsi que de la différence, composante par composante, entre le *centroïde* et chacun des points du *cluster* (*c*). Finalement, le nuage de point en sortie est composé de deux points uniquement (*f*), dont les coordonnées géométriques sont celles des *centroïdes* et les caractéristiques celles qui ont été calculées pour chaque *clusters*.

3.4.1 Boîtes englobantes 3D

En sortie de la couche de *graph-convolution*, nous obtenons un nuage de *centroïdes* dont les parties caractéristiques correspondent aux caractéristiques extraites pour chacun des clusters en entrée de la couche, voir figure 40. En notant L le nombre d'opérations *ClConv* appliquées au nuage de points en entrée, S_L est la sortie de la dernière couche de *graph-convolution*. Notre but est alors de prédire pour chacun des $\boldsymbol{\mu} \in S_L$, une boîte englobante $\mathbf{B}$ autour de l'objet présent dans ce *cluster*, si il y en a, ainsi que sa classe. Cette boîte englobante, positionnée dans le nuage initial S_0, peut déborder de l'enveloppe définie par le *cluster*.

Boîtes modèles en 3D

De la même manière que dans les *RPN* [Gir15], nous avons choisi de définir les boîtes $\mathbf{B}$ prédites par le réseau par rapport à un ensemble de boîtes modèles $\hat{\mathbf{B}}$. Cependant, comme les boîtes modèles utilisées par [Gir15] sont en 2 dimensions uniquement, nous devons adapter le calcul des dimensions de celles-ci. Dans le cas d'images 2D, il n'y a pas de lien entre la taille réelle (en mètres) des objets et la taille de leurs projections dans l'image (en pixels). Ainsi les dimensions des boîtes modèles sont déterminées de manière générique pour tous les objets. Tandis que dans le cas de nuage de points 3D, la taille réelle des objets en mètres est restituée telle quelle. Ainsi il est possible de définir une boîte modèle pour chaque catégorie d'objet que nous souhaitons détecter.

Nous suivons alors l'approche utilisée par [SMAG18] et définissons une boîte modèle par classe. Pour chaque classe d'objet, par exemple voiture, arbre et lampadaire, nous calculons la moyenne des dimensions des objets de la base d'entraînement appartenant à cette classe. A partir de la moyenne des dimensions nous calculons une boîte modèle pour chaque classe, (voir

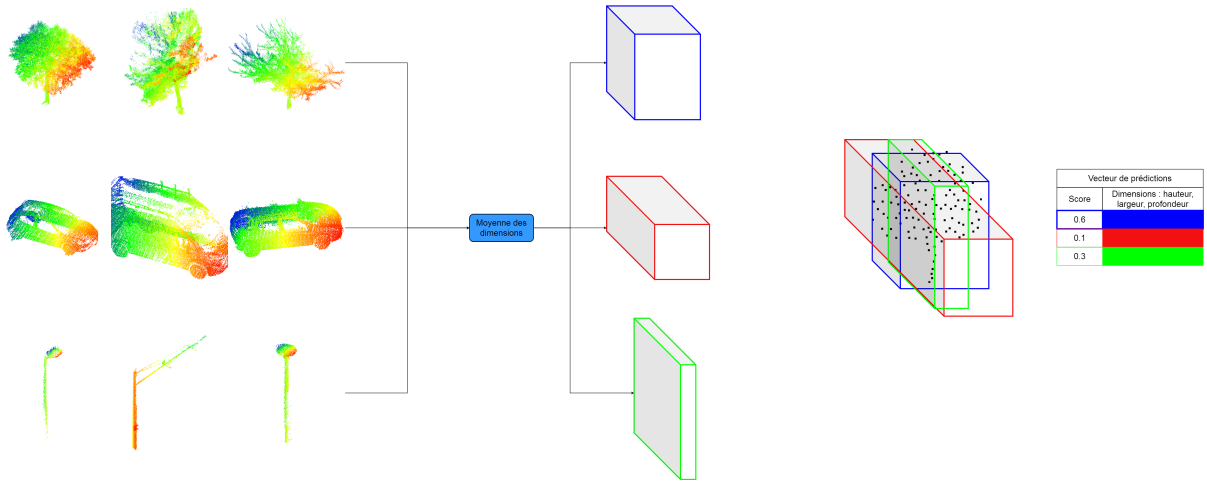

FIGURE 41 – Schéma représentant le calcul et le rôle des boîtes modèles 3D dans le réseau de neurones. La coloration des nuages de points représentent la profondeur du repère géométrique. Au centre le calcul des boîtes modèles par moyenne de la taille des objets pour chaque classe , à droite la prédiction du réseau par rapport à chacune de ces boîtes modèles.

par exemple dans la figure 41), 3 boîtes modèles sont calculées, une cubique pour la classe arbre, une 'horizontale' pour la classe voiture et une 'verticale' pour la classe lampadaire. La fiabilité de ces boîtes modèles peut être augmentée grâce à un niveau d'annotation plus fin, par exemple en définissant les classes camion, automobile, et bus, pour mieux prendre en compte les différences de dimension entre ces classes plutôt que de simplement les regrouper dans la classe voiture.

Représentation des boîtes englobantes prédites par le réseau

Nous introduisons les vecteurs de prédictions, notés $\mathbf{V}$, qui ont pour rôle d'encoder les boîtes englobantes $\mathbf{B}$ prédites par le réseau. Nous définissons les vecteurs de prédictions comme ceci :

$$\mathbf{V} = (\mathbf{M}, \mathbf{T}, \mathbf{\Theta}) \tag{3.19}$$

où $\mathbf{M}$ est le centre des boîtes englobantes, $\mathbf{T}$ est leurs dimensions et $\mathbf{\Theta}$ est l'angle d'orientation selon l'axe vertical. Comme dans les *RPNs* [RHGS17], le vecteur $\mathbf{T}$ est en fait la différence entre les dimensions de la boîte modèle et celles de la boîte prédite (*offset*).

Nous présentons en détail chacune des composantes des vecteurs de prédiction $\mathbf{V}$ dans la figure 42. Le centre $\mathbf{M}$, en rouge dans la figure 42, est encodé par un simple triplet *(x,y,z)* correspondant aux coordonnées du centre des boîtes englobantes $\mathbf{B}$:

$$\mathbf{M} = (x, y, z) \tag{3.20}$$

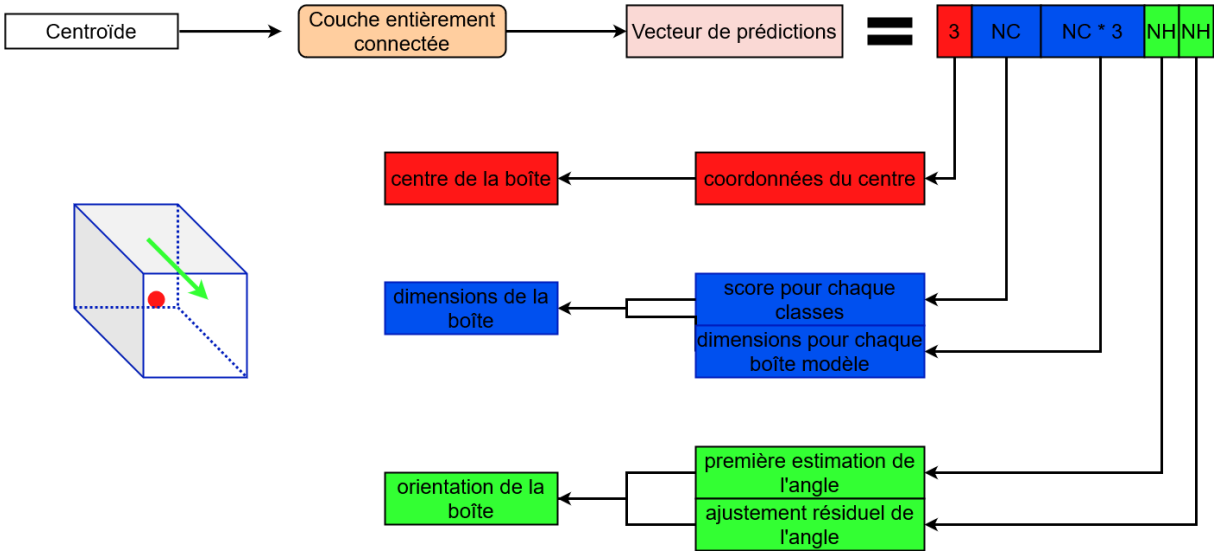

FIGURE 42 – Illustration du vecteur de prédiction pour un *centroïde*. Avec en rouge la composante **M** chargée du centre des boîtes englobantes, en bleu la composante **T** chargée de leurs dimensions et en vert la composante **Θ** chargée de leur orientation.

Ainsi la composante **M** est représentée par 3 scalaires : $Card(\mathbf{M}) = 3$.

Les dimensions des boîtes englobantes **T**, en bleu foncé dans la figure 42, sont encodées chacune par leurs hauteur h, largeur l et profondeur d par rapport à leurs boîtes modèles associées ainsi que par un score s d'appartenance à la classe de leurs boîtes modèles. Le réseau ne va donc pas prédire une unique boîte mais une boîte pour chacune des classes que nous souhaitons détecter. Le réseau va ensuite sélectionner la boîte **B** qui est la plus susceptible de correspondre à la classe de l'objet détecté. Nous illustrons ce procédé dans le schéma à droite dans la figure 41 : à partir d'un *cluster* où se trouve un arbre, représenté par les points noirs dans le schéma, le réseau va prédire 3 boîtes englobantes, une pour chacune des classes arbre, voiture et lampadaire, ainsi qu'un score représentant la probabilité que l'objet détecté appartienne à chacune de ces classes. :

$$\mathbf{T} = (s_1, h_1, l_1, d_1, ...s_{N_C}, h_{N_C}, l_{N_C}, d_{N_C}) \tag{3.21}$$

où s_i est le score d'appartenance à la classe i, N_C le nombre de classes et (h_i, l_i, d_i) l'*offset* de dimension prédit tel que $h_i = \hat{h}_i$ - h'_i, $l_i = \hat{l}_i$ - l'_i et $d_i = \hat{d}_i$ - d'_i, avec $(\hat{h}_i, \hat{l}_i, \hat{d}_i)$ les dimensions de la boîte modèle $\hat{\mathbf{B}}_i$ et (h_i, l_i, d_i) les dimensions de la boîte englobante. Comme il y a autant de boîtes modèles que de nombre de classes N_C, la composante **T** est représentée par $N_C \times 4$ scalaires : $Card(\mathbf{T}) = N_C \times 4$.

Afin de faciliter la convergence du réseau, l'orientation de la boîte englobante, représentée par le vecteur **Θ**, en vert dans la figure 42, est encodée en deux parties :

– une première estimation correspondant à un secteur angulaire : $A \in \left\{\frac{2\pi}{N_H}, ...2\pi\right\}$, où N_H

est le nombre de secteurs angulaires.

– un affinement de la première estimation pour chaque secteur angulaire : $\delta_A \in [-1, 1]$.

La première estimation est encodée par un score A_i, représentant la probabilité pour que l'orientation de la boîte englobante **B** appartient au $i$ème secteur angulaire. L'affinement est encodé par un scalaire pour chaque secteur angulaire représentant la différence entre le secteur angulaire i et l'angle d'orientation de la boîte prédite. L'angle $\boldsymbol{\theta}$ est ainsi encodé comme ceci :

$$\boldsymbol{\Theta} = (A_1, \delta_1, ...A_{N_H}, \delta_{N_H}) \tag{3.22}$$

où A_i est le score d'appartenance au secteur angulaire $\left[\frac{2\pi\times(i-1)}{N_H}; \frac{2\pi\times(i)}{N_H}\right]$ et δ_i est la différence entre l'angle $\boldsymbol{\Theta}$ et la borne supérieure du $i$ème secteur angulaire. Comme il y a en tout N_H secteurs angulaires, $\boldsymbol{\Theta}$ est encodé par $2 \times N_H$ scalaires : $Card(\boldsymbol{\Theta}) = 2 \times NH$.

Ce qui nous donne au total $3 + N_C \times 4 + N_H \times 2$ scalaires par vecteur de prédiction **V**.

Chacune des composantes des *vecteurs de prédictions* **V** sont calculées par une *couche entièrement connectée* (*fully connected layer*). Cette couche de calcul va permettre au réseau de prédire pour chaque *centroïde* $\boldsymbol{\mu}_i^L$, un *vecteur de prédiction* $\mathbf{V}_i$.

Il est possible d'ajouter un certain nombre de composantes supplémentaires aux vecteurs de prédiction **V**, comme par exemple le taux d'encombrement, qui définit à quel point l'objet est occulté, si cette information est présente dans la vérité terrain. Le rôle de ces attributs peut être d'apporter des informations supplémentaires lors de la détection d'objets ou bien de faciliter la phase d'apprentissage [GLSU13, SWL19].

3.4.2 Répartition des points à l'intérieur des *clusters*

Les vecteurs de prédiction **V** étant définis à partir des *clusters*, leurs valeurs dépendent fortement de la répartition de ces derniers. En effet, le nombre d'objets contenus dans un *cluster* ou le taux d'occultation de ces objets, a un impact sur la boîte englobante prédite à partir de ce *cluster*. Nous avons alors besoin de déterminer si notre architecture est capable de produire des *clusters* compatibles avec les deux objectifs de la prédiction de boîte englobante qui sont :

– la définition d'une boîte **B** la plus proche possible des dimensions de l'objet. Objectif qui est plus simple à réaliser quand tous les points du *cluster* qui ne sont pas du fond (*background*) appartiennent au *même objet*.

– la prédiction de la classe à laquelle appartient l'objet détecté. Objectif qui est plus simple à réaliser quand tous les points du *cluster* appartiennent à la *même classe* (mais pas forcément au même objet).

En faisant la synthèse de ces deux objectifs, la répartition de *cluster* idéale est alors celle où chaque objet est contenu entièrement dans un unique *cluster*, voir figure 43 (*a*). Or obtenir une

telle répartition de *cluster* reviendrait à accomplir la tâche de segmentation d'instance (voir section 1.3.2 pour plus de détails) ce qui rendrait inutile la définition de boîtes englobantes.

Un autre cas particulier de répartition des *clusters* est celui où tous les objets de la même classe sont répartis dans un unique *cluster*, voir figure 43 (*b*). Bien que ce cas favorise la prédiction de classe, car tous les points d'un même *cluster* ont la même classe, il est très défavorable à la prédiction de boîtes englobantes car il nécessite de prédire plusieurs objets pour un même *cluster*.

Bien que toutes deux intéressantes, nous trouvons que ces répartitions théoriques ne peuvent pas être atteintes en utilisant une simple fonction de distance, telle que celles discutées dans la section 3.2.3. La répartition des *clusters* dépend alors principalement de l'équilibre entre les parties géométriques $\mathbf{g}$ et parties caractéristiques $\mathbf{f}$ des *clusters* (Eq 3.3), décrite en section 3.2.3, dans le calcul de la distance D.

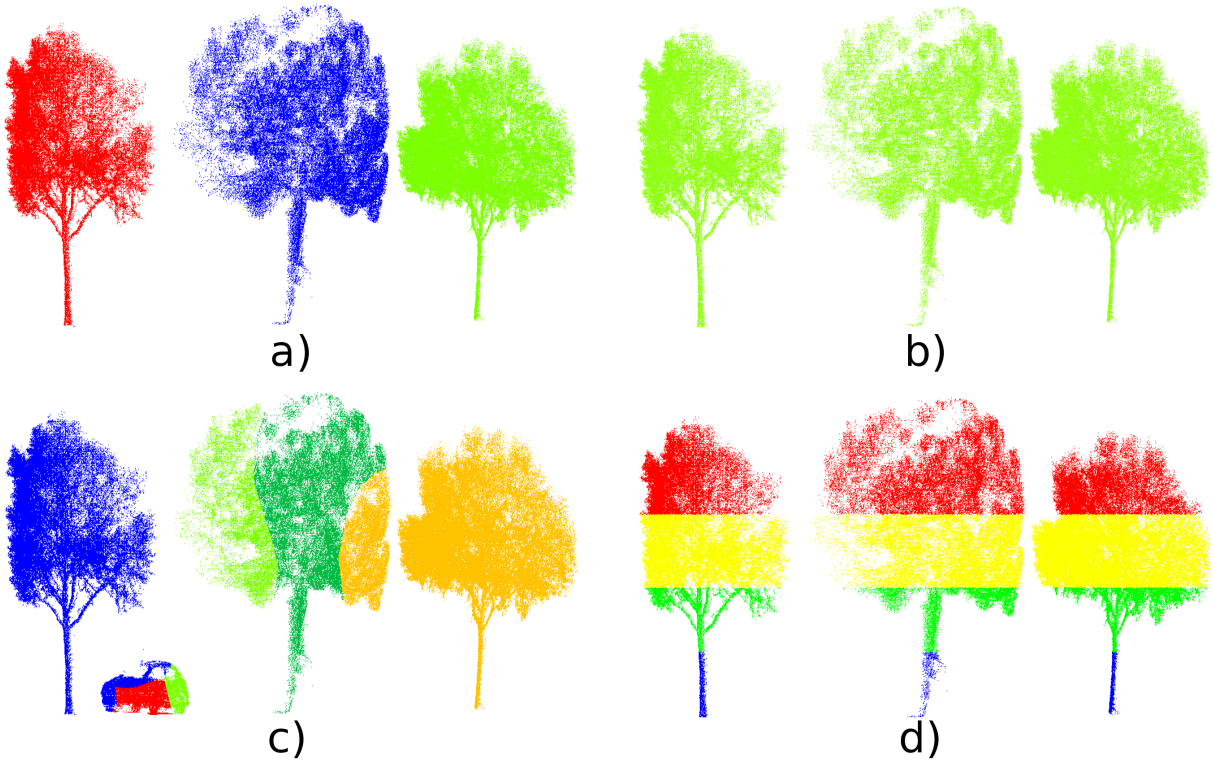

FIGURE 43 – Illustrations de différentes répartitions des *clusters* : en (*a*) le cas optimal avec chaque objet réparti dans un unique *cluster*, en (*b*) le cas où tous les objets de la même classe sont dans le même *cluster*, en (*c*) le cas où la fonction de distance utilisée est calculée uniquement sur la partie géométrique et en (*d*) le cas où la fonction de distance utilisée est calculée uniquement sur la partie caractéristique.

Fonctions de distance naïves

Dans la figure 43 (*c*) nous illustrons le type de répartition des *clusters* obtenue par l'utilisation d'une fonction de distance euclidienne définie uniquement sur les parties géométriques $\mathbf{g}$ des centroïdes $\boldsymbol{\mu}$: $D_{geo}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \left\|\mathbf{g}_i - \mathbf{g}_j\right\|$. Cela a pour effet d'obtenir des *clusters* en forme de

sphère. A part dans certains cas où deux objets sont très proches l'un par rapport à l'autre, comme par exemple les *clusters* bleu foncé et orange dans la figure 43 (*c*), la contrainte d'un seul objet par *cluster* est respectée, ce qui rend plus facile la prédiction de boîte englobante autour de cet objet. Dans cette configuration, les *clusters* sont pour la plupart composés soit uniquement de points appartenant au fond (*background*), soit d'une partie d'objet plus ou moins accompagnée de fond, comme dans les *clusters* rouge et vert foncé dans la figure 43 (*c*). Or la présence de points n'appartenant pas à l'objet dans le *cluster* est susceptible de perturber le vecteur caractéristique de celui-ci et donc compliquer la tâche de classification de l'objet.

Dans la figure 43 (*d*), nous illustrons le type de répartition des *clusters* obtenue par l'utilisation d'une fonction de distance euclidienne définie uniquement sur les parties caractéristiques $\mathbf{f}$ des centroïdes $\boldsymbol{\mu}$: $D_{feat}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j) = \|\mathbf{f}_i - \mathbf{f}_j\|$. Dans ce cas, les *clusters* ont tendance à être composés de points appartenant à des régions qui partagent des caractéristiques communes, sans tenir compte de leurs proximité spatiale. Il est alors possible d'obtenir des *clusters* qui contiennent plusieurs parties d'objets différents appartenant tous à la même classe, comme par exemple le *cluster* bleu dans la figure 43 (*d*) qui contient 3 troncs d'arbre différents. Déduire d'une telle répartition une boîte englobant un seul objet s'avère alors impossible. Néanmoins, une telle répartition a pour conséquence de simplifier la classification, puisque les points d'un même *cluster* sont proches au niveau de leurs caractéristiques il est hautement probable qu'ils appartiennent à des objets de même classe.

Compromis entre distance géométrique et caractéristique

En analysant la répartition de *clusters* obtenues grâce à l'utilisation des fonctions de distance D_{geo} et D_{feat}, nous en sommes venus à la conclusion que la détection au niveau objet est influencée par la distance entre les parties géométriques des points alors que la détection au niveau classe est lié à la distance entre les parties caractéristiques des points.

En recherchant un compromis entre ces deux distances présentées ci-dessus, nous avons identifié les fonctions de distance qui sont à privilégier, parmi celles présentées en section 3.2.3.

Malgré une approche prometteuse, la *distance de Mahalanobis*, définie en 3.6 ne peut être utilisée dans notre architecture à cause du problème connu sous le nom de fléau de la dimension (*curse of dimensionnality*). En effet, le calcul de la matrice de covariance nécessite un échantillon de taille significativement supérieur à la dimension des parties caractéristiques des points ; dans notre cas le nombre de *clusters* se compte en dizaine dans les dernières couches de convolution or le nombre de caractéristiques extraites dépasse habituellement le millier.

Les fonctions qui offrent le meilleur compromis sont :

- la normalisation par moyenne et variance à condition de rééquilibrer la partie géométrique grâce à un coefficient supérieur (Eq 3.5)
- la fonction de distance par agrégation maximale de la partie caractéristique (Eq 3.8)

3.4.3 Processus d'apprentissage

Dans cette section, nous allons présenter le mécanisme d'apprentissage de notre réseau de neurones. Comme dans tout réseau de neurones profond, ce mécanisme passe par la minimisation d'une fonction appelée *fonction de coût* (loss function).

Cette fonction de coût est définie à partir des vecteurs de prédictions $\mathbf{V}_i$, pour chaque *centroïde* $\boldsymbol{\mu}_i \in S_L$, ainsi que d'une vérité terrain, notée $\mathbf{Y}$. Cette fonction de coût $\mathscr{L}$ se décompose en deux termes :

- Le premier terme $\mathscr{L}_{cls}$ sert à superviser la prédiction de la classe de l'objet prédit.
- Le deuxième terme $\mathscr{L}_{box}$ sert à superviser la prédiction de boîte englobante. Le but est de mesurer à quel point la boîte prédite B_i est proche d'un objet de la vérité terrain.

La fonction de coût $\mathscr{L}$ est alors définie comme ceci :

$$\mathscr{L}(\{\mathbf{V}_i\}_1^{N_L}, \mathbf{Y}) = \frac{\alpha}{N_L} \sum_i^{N_L} \mathscr{L}_{cls}(\mathbf{V}_i, \mathbf{Y}) + \frac{\beta}{N_L} \mathscr{L}_{box}(\mathbf{V}_i, \mathbf{Y}) \tag{3.23}$$

où N_L est le nombre de vecteurs de prédiction $\mathbf{V}$, α et β sont des coefficients. La vérité terrain $\mathbf{Y}$ correspond au résultat attendu pour chaque vecteur de prédiction. Celle-ci peut être définie à partir des annotations du nuage de points en entrée, noté S^*. Ces annotations sont effectuées à deux niveaux :

- Au niveau point, où chaque point $\mathbf{P} \in S^*$ est associé à une classe C et à un objet O.
- Au niveau objet, où pour chaque objet O, une boîte englobante $\mathbf{B}_O^*$ est définie.

Nous définissons ainsi l'annotation d'un point $\mathbf{P} \in S^0$ comme ceci :

$$label : \left| \begin{array}{l} \mathbb{R}^3 \rightarrow \{0, ..., N_C\} \times \mathbb{N}^* \\ \mathbf{P} \rightarrow (C, O) \end{array} \right. \tag{3.24}$$

où C est la classe du point $\mathbf{P}$ et O l'indice de l'objet auquel il appartient. L'indice de classe 0 est réservé pour les points n'appartenant à aucun objet. Et les annotations au niveau objet par rapport à l'indice d'objet O :

$$\mathbf{B}_O^* = (\mathbf{M}_O^*, \mathbf{T}_O^*, \boldsymbol{\Theta}_O^*) \tag{3.25}$$

où $\mathbf{M}_O^*$ est le centre de la boîte englobante $\mathbf{B}_O^*$ autour de l'objet O comme définie en (Eq 3.20), $\boldsymbol{\Theta}_O^*$ est l'angle d'orientation de la boîte $\mathbf{B}_O^*$ comme définie dans (Eq 3.22) et $\mathbf{T}_O^*$ est la dimension

de la boîte $\mathbf{B}^*_O$. $\mathbf{T}^*_O$ est définie comme dans (Eq 3.21), à la différence près qu'au lieu d'être définies par rapport à toutes les boîtes modèles possibles, les dimensions de la boîte $\mathbf{B}^*_O$ sont définies uniquement par rapport à la boîte modèle associé à la classe C à laquelle appartient l'objet O.

Néanmoins, avant de définir formellement notre fonction de coût, il nous reste à résoudre une question importante. En effet, le résultat attendu est évident pour un vecteur de prédiction correspondant à un *cluster* ne contenant aucun objet ou bien au contraire pour un *cluster* contient un unique objet en entier. Cependant ce résultat est bien moins évident dans certains cas, comme par exemple quand le *cluster* contient uniquement une partie d'un objet, ou bien plusieurs objets à la fois. Nous devons alors définir un lien entre les *clusters* calculés par le réseau de neurones et les annotations S^* dont nous disposons.

Définition de l'objet majoritaire

Nous définissons l'application $\mathscr{K}$ qui permet de calculer pour chaque *centroïde* $\boldsymbol{\mu} \in S_l$, les points du nuage original S_0 qui lui sont associés. En effet, pour chaque point du nuage de points S_0, l'opération *ClConv* associe un *centroïde* $\boldsymbol{\mu} \in S_1$. Puis à partir de ce *centroïde*, un autre centroïde $\boldsymbol{\mu} \in S_2$ lui est associé par une application récursive de l'opération *ClConv* et ainsi de suite jusqu'à un dernier *centroïde* $\boldsymbol{\mu} \in S_l$. L'application $\mathscr{K}$ permet alors de calculer cette opération dans le sens inverse, c'est-à-dire pour chaque *centroïde* $\boldsymbol{\mu} \in S_l$, de remonter les assignations de *centroïde* calculées successivement par les opérations *ClConv*, jusqu'à arriver aux points du nuage intial S_0. Cette application est alors définie comme ceci :

$$\mathscr{K} : \left| \begin{array}{l} \mathbb{R}^{d_l} \rightarrow \mathbb{R}^{N_i \times 3} \\ \boldsymbol{\mu} \in S_l \rightarrow \{\mathbf{P}_i\}_1^{N_i} \subset S_0 \end{array} \right. \tag{3.26}$$

où les points $\mathbf{P}_i$ sont les points de S_0 et N_i est le nombre de points qui sont associés au *centroïde* $\boldsymbol{\mu}$. Nous appelons cet ensemble de points la *région* $\mathscr{K}(\boldsymbol{\mu})$. En pratique, l'application $\mathscr{K}$ peut être implémentée simplement par la lecture de la *matrice d'assignation*, voir l'algorithme 1, de chaque opération $ClConv$.

Lors de la phase d'entraînement, les points du nuage de points original S_0 sont les mêmes que les points du nuage de points annotés S^*, ainsi l'application de $\mathscr{K}$ sur un *centroïde* $\boldsymbol{\mu} \in S_l$ désigne dans la suite la région du nuage annoté S^* qui correspond au *cluster* dont $\boldsymbol{\mu}$ est le *centroïde*.

Nous voulons alors définir pour chaque *centroïde* $\boldsymbol{\mu}_i \in S_l$, l'objet majoritaire, O_{maj}, que doit englober la boîte encodée par le vecteur de prédiction $\mathbf{V}_i$. En d'autres termes, l'objet majoritaire O_{maj} est *l'objet majoritaire* de l'ensemble des points de la *région* $\mathscr{K}(\boldsymbol{\mu}_i)$. Puisque la définition d'un objet majoritaire O_{maj} nécessite d'avoir accès au nuage de point annoté S^*, celle-ci est effectuée uniquement lors de l'entraînement du réseau.

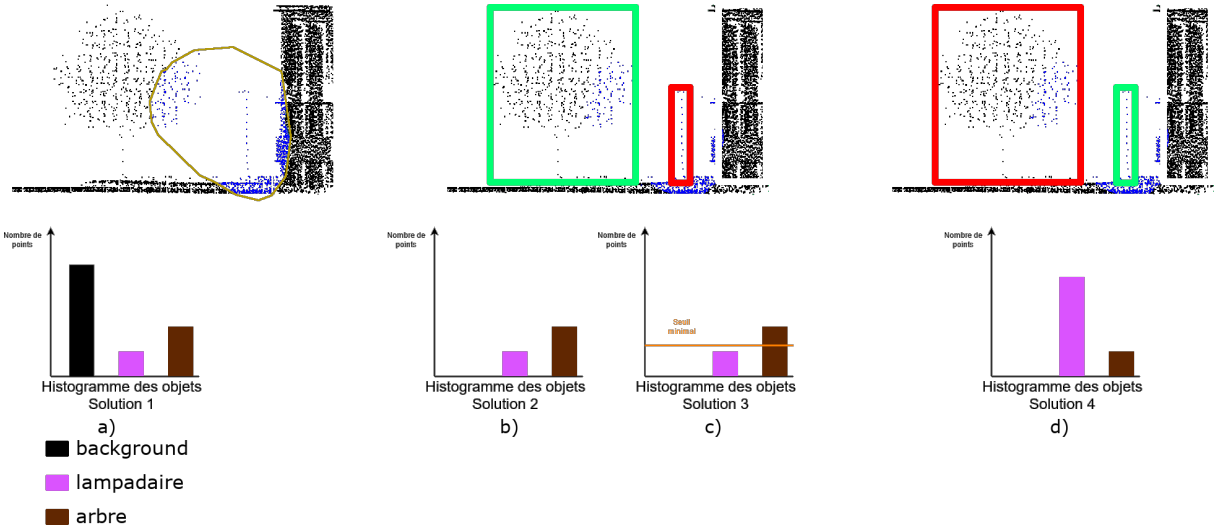

FIGURE 44 – Illustration du problème de détermination de l'objet majoritaire : nous cherchons à déterminer l'objet majoritaire pour le *cluster* central, en bleu. Les histogrammes représentent le nombre de point dans ce *cluster* avec en noir le fond (*background*), en marron l'arbre et en rose le lampadaire. Cet objet majoritaire est différent selon la définition que nous utilisons : (*a*) l'objet majoritaire est défini comme le maximal en termes de nombre de points, on obtient alors un cluster sans objet (*background*) ; (*b, c*) l'objet majoritaire est défini comme le maximal en termes de nombre de points avec un seuil minimal τ, sans compter les points appartenant au fond, on obtient alors l'arbre ; (*d*) l'objet majoritaire est défini selon le nombre de points de l'objet appartenant au *cluster* par rapport au nombre de points de l'objet en entier, on obtient alors le lampadaire.

Une première solution consiste à définir l'objet majoritaire O_{maj} comme l'objet étant le plus représenté en termes de nombre de points dans le *cluster* $\mathscr{K}(\boldsymbol{\mu})$. L'indice O^i_{maj} associé au *centroïde* $\boldsymbol{\mu}_i$ est alors défini comme ceci :

$$O^i_{maj} = \underset{O \in 0,\ldots,N_O}{\arg\max} \sum_{\mathbf{P}_j \in \mathscr{K}(\boldsymbol{\mu}_i)} Card(\{\mathbf{P}_j | O_j = O\}) \tag{3.27}$$

où O_j est l'indice qui correspond à l'objet auquel appartient le point $\mathbf{P}_j$ et N_O est le nombre d'objets.

Cette méthode de calcul est illustrée dans la figure 44 (*a*). Un défaut de cette méthode de calcul est qu'il y a plus de points appartenant au fond que de points appartenant à des objets, dû au fait que les objets urbains sont moins volumineux que les façades de bâtiments ou les routes. Ainsi, la majeure partie des *clusters* ont pour objet majoritaire un objet du fond. Par exemple dans la figure 44, le *cluster* rouge contient des points appartenant à une façade d'immeuble, or le nombre de points appartenant à cette façade est supérieur au nombre de points appartenant à l'arbre et au poteau, voir l'histogramme en dessous 44 (*a*). L'objet majoritaire du *cluster* est donc la façade et celui-ci est donc considéré comme un *cluster de fond* alors même que celui-ci contient un objet.

Une deuxième solution visant à résoudre ce problème consiste à ne pas prendre en compte lors du calcul de l'objet majoritaire O_{maj} les points appartenant au fond. Ainsi l'indice O_{maj} associé au *centroïde* $\boldsymbol{\mu}_i$ est alors défini comme ceci :

$$O_{maj} = \begin{cases} \underset{O \in 1,\ldots,N_O}{\arg\max} \sum\limits_{\mathbf{P}_j \in \mathscr{K}(\boldsymbol{\mu}_i)} Card\left(\{\mathbf{P}_j | O_j = O\}\right) \\ \text{si } \exists \mathbf{P}_j \in \mathscr{K}(\boldsymbol{\mu}_i) | O_j \neq 0 \\ 0 \text{ sinon} \end{cases} \tag{3.28}$$

où O_j est l'objet auquel appartient le point $\mathbf{P}_j$ et $O_j \neq 0$ signifie que le point $\mathbf{P}_j$ n'appartient pas au fond.

Le problème majeur de cette approche est la définition des *clusters* ne contenant pas d'objets ; il suffit qu'un seul point n'appartenant pas au fond se trouve dans un *cluster* pour que l'objet associé à ce point soit défini comme objet majoritaire. Nous illustrons cette méthode de calcul dans la figure 44 (*b*) où le nombre de points appartenant au fond n'est pas pris en compte. Ainsi dans le cas d'un *cluster* contenant uniquement des points appartenant au fond sauf un, le *cluster* est considéré comme contenant un objet alors même que cet objet n'est représenté par un unique point dans la région $\mathscr{K}(\boldsymbol{\mu}_i)$. Prédire une boîte englobante à partir d'un unique point s'avère alors une tâche impossible.

Une troisième solution consiste alors à définir un taux d'occupation minimale τ représentant un rapport entre nombre de points appartenant à un objet et nombre de points appartenant à la région $\mathscr{K}(\boldsymbol{\mu}_i)$. Ainsi l'indice O_{maj} associé au *centroïde* $\boldsymbol{\mu}_i$ est alors défini comme ceci :

$$O_{maj} = \begin{cases} \text{si } \exists \mathbf{P}_j \in \mathscr{K}(\boldsymbol{\mu}_i) | O_j \neq 0 \text{ et } Card(O_j) \geq \tau \times Card\left(\mathscr{K}(\boldsymbol{\mu}_i)\right) : \\ \underset{O \in 1,\ldots,N_O}{\arg\max} \sum\limits_{\mathbf{P}_j \in \mathscr{K}(\boldsymbol{\mu}_i)} Card\left(\{\mathbf{P}_j | O_j = O\}\right) \\ 0 \text{ sinon} \end{cases} \tag{3.29}$$

où $Card\left(\mathscr{K}(\boldsymbol{\mu}_i)\right)$ est le nombre de points de la région $\mathscr{K}(\boldsymbol{\mu}_i)$ et $Card(O_j)$ est le nombre de point de la région $\mathscr{K}(\boldsymbol{\mu}_i)$ appartenant à l'objet O_j. La valeur de τ peut être déterminée expérimentalement, par exemple ce taux est de 70% dans notre cas.

Nous illustrons cette méthode de calcul dans la figure 44 (*c*). Cette approche comporte un défaut dans le cas des *clusters* regroupant des points appartenant à des objets différents, comme le *cluster* en rouge dans la figure 44 qui regroupe un lampadaire en entier ainsi que des points appartenant au feuillage d'un arbre. Ainsi, dans ce cas, malgré le fait que le lampadaire est présent en entier dans le *cluster*, l'objet majoritaire calculé est l'arbre, car il y a plus de points dans la partie du feuillage de l'arbre que dans le lampadaire en entier. Le problème est donc

causé par la différence de volume entre les différents objets urbains. Modifier la valeur de τ serait alors insuffisant pour régler ce problème. En effet, modifier cette valeur, représentée par une ligne horizontale orange dans l'histogramme 44 (*b*), ne modifierait pas le ratio entre les deux valeurs.

Une dernière approche consiste alors à définir un taux d'occupation τ_O pour chaque objet O. Ainsi l'indice O_{maj} associé au *centroïde* $\boldsymbol{\mu}_i$ est alors défini comme ceci :

$$O_{maj} = \begin{cases} \text{si } \exists \mathbf{P}_j \in \mathscr{K}(\boldsymbol{\mu}_i) | O_j \neq 0 \text{ et } Card(O_j) \geq \tau_{O_j} \times Card\left(\mathscr{K}(\boldsymbol{\mu}_i)\right) : \\ \underset{O \in 1,\ldots,N_O}{\arg\max} \dfrac{\tau_O}{Card\left(\mathscr{K}(\boldsymbol{\mu}_i)\right)} \displaystyle\sum_{\mathbf{P}_j \in \mathscr{K}(\boldsymbol{\mu}_i)} Card\left(\{\mathbf{P}_j | O_j = O\}\right) \\ 0 \text{ sinon} \end{cases} \tag{3.30}$$

Nous illustrons cette méthode de calcul dans la figure 44 (*d*). Ici le nombre de points par objet, représenté par l'histogramme dans la figure 44 (*d*), est normalisé par le taux τ_O correspondant à chaque objet. Cette approche comporte tout de même des limites, ainsi il est nécessaire de déterminer une valeur pour chaque taux d'occupation τ_O. En pratique il est possible de déterminer uniquement une valeur de τ de manière empirique puis de calculer chaque τ_O comme le produit entre le taux d'occupation τ et le rapport entre le nombre de points du nuage de points initial S_0 appartenant à l'objet O et le nombre total de points du nuage S_0.

$$\tau_O = \tau \times \frac{Card\left(\{\mathbf{P}_i \in S^* | O_i = O\}\right)}{Card(S_0)} \tag{3.31}$$

où $Card(S_0)$ est le nombre de points appartenant au nuage de points S_0. Dans ce cas, le nombre, ainsi normalisé, de points appartenant au lampadaire dans la figure 44 est supérieur au nombre de points normalisé appartenant à l'arbre, parce que le seuil spécifique à la classe lampadaire est inférieur à celui de la classe arbre. Cette différence au niveau des seuils est dû au fait que le deuxième terme de (Eq 3.31) est plus petit dans le cas des lampadaires car ceux-ci occupent moins d'espace dans le nuage de points initial.

Définition de la fonction de coût

Rappelons que pour chaque *cluster* $\boldsymbol{\mu}_i$ nous déterminons :

- un vecteur de prédiction $\mathbf{V}_i = (\mathbf{M}, \mathbf{T}, \boldsymbol{\Theta})$ (Eq 3.19)
- un objet majoritaire O^i_{maj}

Nous pouvons alors définir l'application de *la fonction de coût* de notre réseau sur chaque cluster $\boldsymbol{\mu}_i$. Cette fonction de coût $\mathscr{L}$ se décompose en deux termes :

– Le premier terme $\mathscr{L}_{cls}$ sert à superviser la prédiction de la classe de l'objet majoritaire.

– Le deuxième terme $\mathscr{L}_{box}$ sert à superviser la prédiction de boîte englobante. Le but est de mesurer à quel point la boîte prédite par le réseau B_i est proche de la boîte englobante annotée $\mathbf{B}^*_{i,O_{maj}}$.

La fonction de coût $\mathscr{L}$ est donc définie comme ceci :

$$\mathscr{L}(\{\mathbf{V}\}_1^{N_l}, S^*) = \frac{1}{N_l}\sum_i^{N_l} \mathscr{L}_{cls}\left(\mathbf{V}_i, \mathbf{B}^*_{i,O_{maj}}\right) + \frac{1}{N_l}\mathscr{L}_{box}\left(\mathbf{V}_i, \mathbf{B}^*_{i,O_{maj}}\right) \tag{3.32}$$

où : S^* est le nuage de points original annoté, $\{\mathbf{V}\}_1^{N_l}$ l'ensemble des vecteurs de prédiction prédits par le réseau et $\mathbf{B}^*_{i,O_{maj}}$ la boîte englobante correspondant à l'objet majoritaire du *cluster* associé au *centroïde* $\boldsymbol{\mu}_i \in S_l$.

La fonction $\mathscr{L}_{cls}$ est une fonction d'entropie croisée avec en entrée les scores prédits pour chacune des classes possibles. Ces scores sont encodés dans les vecteurs de prédictions $\mathbf{V}$ (Eq 3.19) par la composante $\mathbf{T}$ (Eq 3.21). Ainsi la fonction $\mathscr{L}_{cls}$ est définie comme ceci :

$$\mathscr{L}_{cls}(\mathbf{V}_i, \mathbf{B}^*_{i,O_{maj}}) = -\sum_{C=1}^{N_C} y_C \log(s_C) \tag{3.33}$$

où N_C est le nombre de classes, y_C est un indicateur binaire qui vaut 1 si C est égale à la classe de l'objet majoritaire O_{maj} et 0 sinon, s_C est le Cième score de la composante dimension $\mathbf{T}$ (Eq 3.21) du vecteur de prédiction $\mathbf{V}_i$.

La fonction $\mathscr{L}_{box}$ est divisée en 3 termes, chacun chargé de superviser l'apprentissage d'une des 3 composantes des vecteurs de prédictions $\mathbf{V}$:

$$\begin{aligned}\mathscr{L}_{box}(\mathbf{V}_i, \mathbf{B}^*_{i,O_{maj}}) = \lambda_1 \times \mathscr{L}_{centre}(\mathbf{M}_i, \mathbf{M}^*_{O_{maj}}) \\ +\lambda_2 \times \mathscr{L}_{offset}(\mathbf{T}_i, \mathbf{T}^*_{O_{maj}}) \\ +\lambda_3 \times \mathscr{L}_{angle}(\boldsymbol{\Theta}_i, \boldsymbol{\Theta}^*_{O_{maj}})\end{aligned} \tag{3.34}$$

où $\mathbf{M}_i$ (Eq 3.20), $\mathbf{T}_i$ (Eq 3.21) et $\boldsymbol{\Theta}$ (Eq 3.22) sont les 3 composantes du vecteur de prédiction $\mathbf{V}_i$. $\mathbf{M}^*_{O_{maj}}$, $\mathbf{T}^*_{O_{maj}}$ et $\boldsymbol{\Theta}^*_{O_{maj}}$ sont les 3 composantes de la boîte englobante associée (Eq 3.25) à l'objet majoritaire O_{maj} du $i$ème *cluster*.

La fonction $\mathscr{L}_{centre}$ supervise l'apprentissage de la prédiction du centre de la boîte englobante. Afin de faciliter la convergence lors de l'apprentissage, nous implémentons la fonction $\mathscr{L}_{centre}$ par une fonction d'Huber, notée $\mathscr{L}_{Huber}$ [Hub64]. En effet, comme celle-ci est monotone décroissante, elle permet une meilleur convergence comparée à une distance euclidienne. Ainsi la fonction $\mathscr{L}_{centre}$ est définie ainsi :

$$\mathscr{L}_{centre}(\mathbf{M}_i, \mathbf{M}^*_{O_{maj}}) = \mathscr{L}_{Huber}((x_i, y_i, z_i), (x^*, y^*, z^*)) \tag{3.35}$$

où (x_i, y_i, z_i) sont les coordonnées du centre prédit par le réseau (Eq 3.20) et (x^*, y^*, z^*) les coordonnées du centre de la boîte englobante associée à l'objet majoritaire O_{maj}.

La fonction $\mathscr{L}_{offset}$ supervise l'apprentissage de la prédiction des dimensions de la boîte englobante. Encore une fois nous préférons utiliser la fonction d'Huber pour calculer la différence entre les dimensions de la boîte prédite et celles de la boîte correspondant à l'objet majoritaire. Seules les dimensions calculées par rapport à la boîte modèle associée à la classe de l'objet majoritaire O_{maj} sont prises en compte lors du calcul de $\mathscr{L}_{offset}$. Ainsi la fonction $\mathscr{L}_{offset}$ est définie comme ceci :

$$\mathscr{L}_{offset}(\mathbf{T}_i, \mathbf{T}^*_{O_{maj}}) = \sum_{C=1}^{N_C} y_C \times \mathscr{L}_{Huber}((h_C, w_C, d_C), (h^*, w^*, d^*)) \tag{3.36}$$

où N_C est le nombre de classes, y_C un indicateur binaire qui vaut 1 si C est la classe de l'objet majoritaire O_{maj} et 0 sinon, (h_C, w_C, d_C) sont les dimensions de la boîte prédite par le réseau par rapport à la boîte modèle associée à la classe C et (h^*, w^*, d^*) sont les dimensions de la boîte englobant l'objet majoritaire $\mathbf{B}_{O_{maj}}$, qui sont définies par rapport aux dimensions de la boîte modèle $\hat{\mathbf{B}}$ associée à la classe de l'objet majoritaire O_{maj}.

La fonction $\mathscr{L}_{angle}$ supervise l'apprentissage de la prédiction de l'angle d'orientation de la boîte englobante par rapport à l'axe vertical. Cette fonction est définie en deux termes :

- *Une fonction d'entropie croisée* pour apprendre à déterminer le bon secteur angulaire.
- *Une fonction d'Huber* pour calculer la distance entre l'affinement δ_i entre le bon secteur angulaire et l'angle d'orientation de la boîte englobante de l'objet majoritaire $\mathbf{B}_{O_{maj}}$.

Ainsi la fonction $\mathscr{L}_{angle}$ est définie ainsi :

$$\mathscr{L}_{angle}(\mathbf{\Theta}_i, \mathbf{\Theta}^*_{O_{maj}}) = -\sum_{H=1}^{N_H} y_H \log(A_H) + \sum_{H=1}^{N_H} y_H \times \mathscr{L}_{Huber}(\delta_H, \delta^*_{H^*}) \tag{3.37}$$

où N_H est le nombre de secteur angulaire, y_H un indicateur binaire qui vaut 1 si l'angle d'orientation de la boîte englobante de l'objet majoritaire $\mathbf{B}_{O_{maj}}$ est compris dans le secteur angulaire $\left[\frac{(H-1)\times 2\pi}{N_H}; \frac{H\times 2\pi}{N_H}\right]$, A_H est le score pour ce secteur angulaire et δ_H l'affinement prédit par rapport à ce secteur angulaire (Eq 3.22).

La fonction d'Huber que nous utilisons est définie ainsi :

$$\mathscr{L}_{huber}(x, y) = \begin{cases} \frac{1}{2}(x-y)^2 \text{ si } (|x-y|) \leq \delta \\ \delta(|x-y|) - \frac{1}{2}\delta \text{ sinon} \end{cases} \tag{3.38}$$

où $|x - y|$ est la valeur absolue de $x - y$.

Bilan

Nous proposons une nouvelle architecture de réseaux de neurones pouvant être entraînée sur la tâche de détection d'objets. Notre architecture prend en entrée uniquement les coordonnées (*x, y ,z*) des nuages de points 3D. La détection d'objets est permise grâce à une nouvelle opération, *cluster-convolution*, que nous proposons afin de généraliser les *convolutions à pas* aux domaines des nuages de points.

Dans le chapitre suivant, nous présentons les expériences que nous avons menées sur la détection d'objets urbains à partir de notre architecture.

Chapitre 4

Application à la détection d'objets en milieu urbain

4.1 Introduction

4.1.1 Difficultés spécifiques aux applications en milieu urbain

Rareté des travaux existants

Comme nous l'évoquons dans la section 2.7, à notre connaissance aucun travail de recherche sur la détection d'objets urbains par *deep-learning 3D* n'a été mené avant celui que nous présentons. Les bases de données publiques utilisées pour la tâche de détection d'objets dans des nuages de points [GWH+20], le sont ainsi uniquement dans le cas de détection d'objets dans des scènes intérieures ou bien dans le cadre de développement de véhicules autonomes. Ces deux cas d'applications comportent chacun leurs spécificités, qui s'éloignent de celles des objets urbains (voir section 2.7 pour plus de détails).

Néanmoins, la particularité des objets urbains, en *deep-learning 3D*, est que ceux-ci sont la cible d'études approfondies dans le cadre de projets de cartographie urbaine. En effet, il existe un certain nombre de travaux portant sur la *segmentation sémantique* dans des acquisitions *LiDAR* en milieu urbain. Ces travaux s'accompagnent de la constitution de plusieurs bases de données publiques de nuages de points en milieu urbain. Bien que la tâche de *segmentation sémantique* soit plus simple que la détection d'objets, notamment à cause de l'absence d'individualisation des objets traités (voir section 1.3.2 pour plus de détails), ces bases de données peuvent tout de même nous être utiles.

Revue des bases de données disponibles

Nous passons ici en revue les différentes bases de données existantes, celles-ci sont utilisées pour la *segmentation sémantique* en milieu urbain, afin de déterminer lesquelles sont exploitables et dans quelle mesure. Nous résumons notre revue des bases de données existantes dans le tableau

4.1. Nous utilisons les critères suivants afin d'évaluer la pertinence de ces bases de données :

- protocole d'acquisition : comme nous le spécifions dans le chapitre 1, le type d'acquisition à privilégier dans notre cas est l'acquisition dynamique terrestre (*MLS*). En cas de protocole différent, *LiDAR* statique ou bien monté à l'avant du véhicule comme dans le cas des véhicules autonomes, seuls les objets proches de la source d'acquisition sont détectables,

- classes : il s'agit des différentes classes d'objets présents dans le nuage de points. Même si une base de données est correctement annotée, elle est utile dans notre cas uniquement si elle contient au moins une classe d'objets urbains (voir section 1.1.1 pour plus de détails sur la définition des objets urbains),

- annotations : il s'agit du critère le plus important. En effet, dans le cadre de travaux sur la *segmentation sémantique*, les nuages de points sont annotés au niveau *classe* uniquement ; à chaque point (x, y, z) est associé un label C correspondant à la classe de l'objet auquel il appartient. Ce niveau d'annotation est insuffisant pour la détection d'objets. Cependant, certains nuages de points sont annotés avec un niveau de détails plus important : à chaque point (x, y, z) est associé non seulement un label C mais aussi un label O correspondant à un identifiant de cet objet (voir section 1.3.2 pour plus de détails). Il est alors possible d'extraire chaque objet présent dans la base de données en filtrant le nuage de points sur la valeur O.

Le premier constat de cette revue des bases de données existantes, est qu'un certain nombre d'entre elles sont utilisables afin de mener des expériences sur la détection d'objets : le *Kevin Lai dataset* [LF10], la base *Paris rue Madame* [SMGD14] et le *Sydney urban dataset* [QUD12]. Néanmoins le nombre d'objets disponibles est largement inférieur à celui utilisé dans d'autres applications. Par exemple, la plus grande base de données d'objets urbains [RDG18] compte 2479 objets, alors que dans le cas des scènes d'intérieur la base *SunRGB-D* [SLX15] en compte plus de 65 000. Le faible nombre d'objets disponibles apporte une contrainte supplémentaire à nos travaux.

Limites des méthodes utilisées

Nous montrons dans la section 2.4.3 que les méthodes basées sur *PointNet* [QSMG17], dont la notre, comportent une contrainte sur le nombre de points maximal des nuages de points. En effet au delà d'un certain seuil, noté N_{max}, la consommation de mémoire est trop importante. Ce seuil varie énormément selon le détail des architectures de réseau et du matériel utilisé, et se situe entre *4096* et *8192* [1] dans la plupart des cas.

Par ailleurs nous montrons également dans la section 2.4.3, que ces mêmes méthodes comportent aussi une limite sur la taille minimale des nuages de points. En effet, en dessous

1. Avec une carte graphique Nvidia Titan X disposant de 12Go de mémoire.

Nom	Protocole d'acquisition	Taille des scènes	Niveau d'annotations	Nombre d'objets	Classes
Malaga urban dataset [BCMDGJ14]	acquisition *MLS* : caméra stéréo 5 capteurs *LiDAR*	38.5 km découpé en 15 scènes	aucun	-	-
Oakland 3D point cloud dataset [MBVH09]	acquisition *MLS* : *LiDAR* monté sur plateforme mouvante	1.6 km	classe	-	bruit bâtiment poteau arbre
Kevin Lai dataset [LF10]	acquisition *MLS* *LiDAR* monté sur voiture	~1 km	objet	1014	bruit bâtiment voiture signalisation poteau arbre
Sydney urban object [QUD12] *dataset*	acquisition *TLS* uniquement les objets isolés	-	objet	588	bâtiment voiture signalisation poteau arbre personne
Paris rue Madame [SMGD14] *database*	acquisition *MLS* *LiDAR* monté sur voiture	160 m	objet	642	bruit bâtiment voiture signalisation poteau arbre personne
Large-scale point cloud classification benchmark [HSL+17]	acquisition *TLS*	-	classe	-	bruit bâtiment voiture arbre
Paris-Lille dataset [RDG18]	acquisition *MLS* *LiDAR* monté sur voiture	2 km	objet	2479	bâtiment voiture signalisation poteau arbre personne

TABLE 4.1 – Tableau récapitulatif des jeux de données publics contenant des objets urbains

d'un certain nombre N_{min}, globalement situé aux alentours de 100, l'identification des objets par le réseau est fortement entravée.

4.1.2 Portée des expériences réalisées

Nous avons voulu aborder la tâche de détection d'objets sous 2 angles différents :

- *La classification* d'objets, c'est-à-dire être en mesure, à partir d'un objet isolé, de prédire la classe à laquelle il appartient (voir section 1.3.2 pour plus de détails).
- *Le partitionnement*, c'est-à-dire identifier à l'intérieur d'un nuage de points entier, des *régions* susceptibles de comporter un objet.

a) Classification

Dans notre cas le problème de classification des objets urbains constitue une étape intermédiaire vers la détection d'objets. Comme il n'existe pas, à notre connaissance, de travaux de

classification d'objets urbains à l'aide du *deep-learning 3D*, nous devons évaluer son efficacité dans ce cas. En effet, si il est difficile de classifier des objets isolés alors il sera d'autant plus difficile de détecter ces objets dans une scène réelle. Nous avons alors proposé des travaux sur la tâche de *classification* d'objets urbains [ZCS+19][ZCS+18]. Nous détaillons ces travaux et leurs résultats dans la section 4.2.

b) Partitionnement

Le problème de partitionnement d'un nuage de points en régions pertinentes est le problème principal posé par la tâche de détection d'objets. C'est à travers la résolution de celui-ci que nous pouvons évaluer l'efficacité de notre proposition. Cette évaluation du partitionnement n'est pas sans difficulté, car contrairement aux premières méthodes de détection d'objets (voir section 2.1), dans notre architecture le partitionnement et le calcul des *caractéristiques* nécessaires à l'identification, sont effectués simultanément (voir sections 3.2 et 3.3). Ainsi à travers plusieurs expériences de détection d'objets nous proposons une analyse quantitative et qualitative du partitionnement de notre approche par *cluster-convolution*. Nous décrivons ces expériences et leurs résultats dans la section 4.3.3.

4.2 Classification d'objets urbains isolés

4.2.1 Protocole expérimental

Jeu de données utilisé

A partir de notre comparatif des jeux de données existants (voir tableau 4.1) nous avons sélectionné 3 bases qui correspondent à nos critères : *Kevin Lai dataset* [LF10], *Paris rue Madame database* [SMGD14] et *Sydney urban object dataset* [QUD12]. Cette dernière a l'avantage d'être une base initialement prévue pour la classification d'objets et où chaque nuage de points correspond à un objet isolé. Pour les deux autres bases, nous illustrons notre procédé d'individualisation des objets dans la figure 46 : celui-ci consiste à isoler les objets en filtrant les points des scènes 3D selon leurs label d'identifiant d'objet, noté O. Les objets contenant plus de 100 points sont alors ajoutés à notre base de données.

En ce qui concerne les classes d'objets, nous avons choisi les classes présentes dans au moins 2 des jeux de données sélectionnés et en adoptant pour chaque classe le niveau de taxonomie le plus large défini par ces 3 jeux de données. Par exemple le jeu de données [QUD12] fait la distinction entre la classe *tronc* et la classe *arbre*, or cette distinction n'est pas faite dans les 2 autres jeux de données, nous avons donc regroupé les objets de classe *tronc* dans la classe *arbre*. Il en est de même pour les objets de classe *feux de signalisation* et ceux de classe *panneaux de signalisation*. Nous avons dû écarter d'autres catégories comme les abri-bus et les poubelles, car trop peu d'exemples sont disponibles. Pour améliorer le caractère distinctif de notre jeu de données, nous avons ajouté d'autres classes qui ne correspondent pas à des objets urbains. Ces objets sont ajoutés parce qu'ils sont très souvent présents en milieu urbain et permettent donc

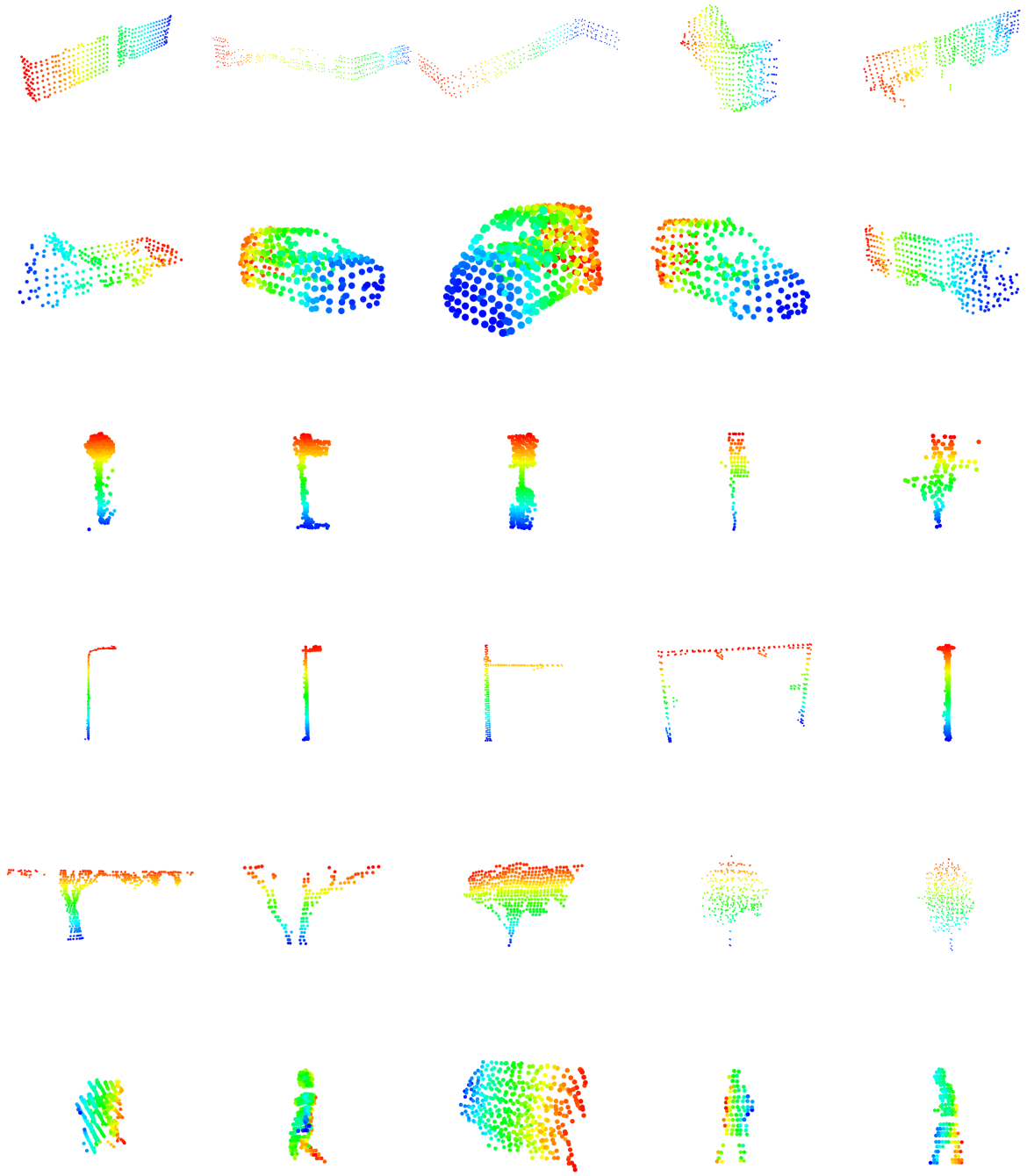

FIGURE 45 – Visualisation d'exemples pour chacune des classes que nous avons utilisées, mis à part la classe '**bruit**'. La couleur correspond à la profondeur du nuage de points dans l'espace (axe Y). De haut en bas : la classe '**bâtiment**' avec plusieurs exemples de façades d'immeuble ou de maisons, la classe '**voiture**' où sont regroupés des voitures et des camions, la classe '**signalisation**' avec des feux et des panneaux de signalisation, la classe '**poteau**' où sont regroupés des lampadaires et des poteaux de caténaire, la classe '**arbre**' avec plusieurs essences d'arbres avec ou sans feuille et enfin la classe '**personne**' où on peut remarquer l'effet de traînée causé par les personnes en mouvement.

Classe	voiture	signalisation	poteau	arbre	personne	bruit	bâtiment
Nombre d'exemples	88	141	22	149	164	57	94
Total	**724**						

TABLE 4.2 – Tableau récapitulatif de la répartition de la *BaseA*.

de mieux appréhender la variabilité d'un environnement urbain. C'est le cas des classes *bruit*, où les divers artefacts d'acquisition sont regroupés, et *bâtiments* où sont regroupés des façades d'immeubles et de maison. Des exemples pour chacune des classes sont présentées dans la figure 45.

En tout nous utilisons 724 objets urbains avec la répartition suivante (voir tableau 4.2.1) : 88 objets de classe *voiture*, 141 objets de classe *signalisation*, 22 objets de classe *poteau*, 149 objets de classe *arbre* et 164 objets de classe *personne*, 57 objets de classe *bruit*, 94 objets de classe *bâtiment*. Nous appelons le jeu de données issu de cet assemblage des bases *Kevin Lai dataset* [LF10], *Paris rue Madame database* [SMGD14] et *Sydney urban object dataset* [QUD12] la *Base A* [2].

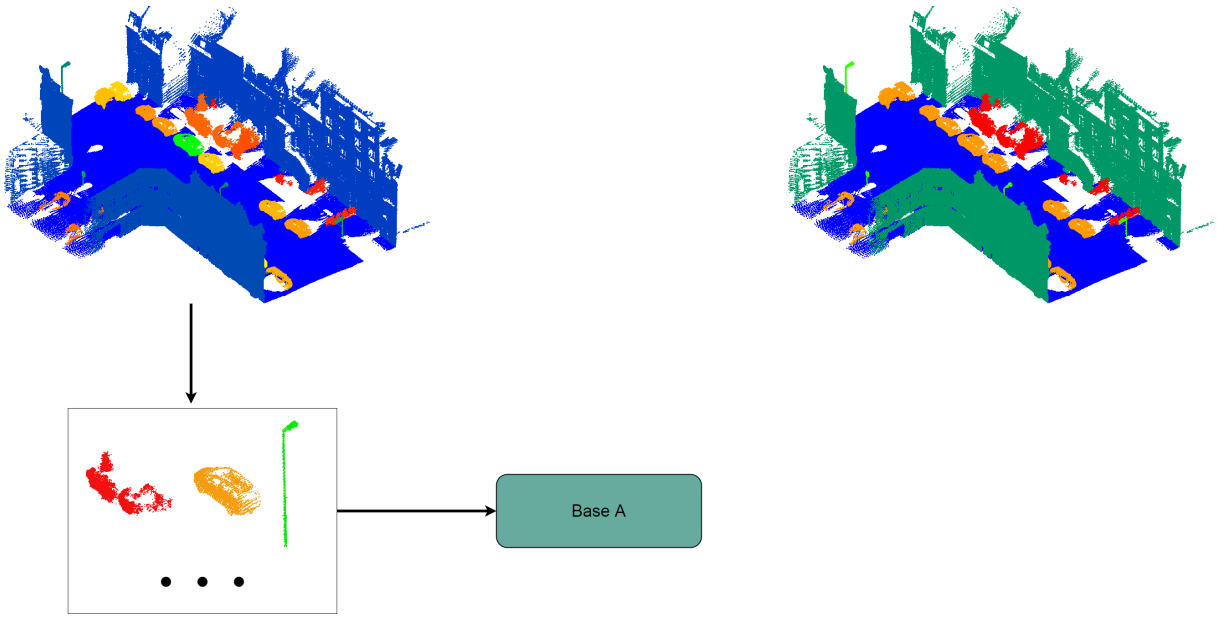

FIGURE 46 – Présentation de la différence entre annotations au niveau classe, à droite, où chaque objet de classe identique est de même couleur et annotations au niveau objet, à gauche, où chaque objet a une couleur différente des autres. C'est à partir de ces dernières qu'il est possible d'extraire des objets, ici un arbuste, une voiture et un lampadaire, et de les ajouter à notre *Base A*

Pré-traitement des nuages de points

Afin de contrôler les expériences de classification d'objets urbains, nous avons effectué des pré-traitements sur nos données. En effet, du fait de l'hétérogénéité des nuages de points

2. jeu de données disponibles à l'adresse suivante : www.lirmm.fr/~zegaoui

provenant initialement de jeux de données différents, nous avons décidé d'homogénéiser nos données. Il s'agit d'un problème classique d'apprentissage automatique : la variabilité du jeu d'apprentissage. Si celle-ci est trop grande le modèle peut avoir du mal à converger lors de l'apprentissage et si celle-ci est trop faible, le modèle peut avoir du mal à être généralisé à d'autres jeux de données.

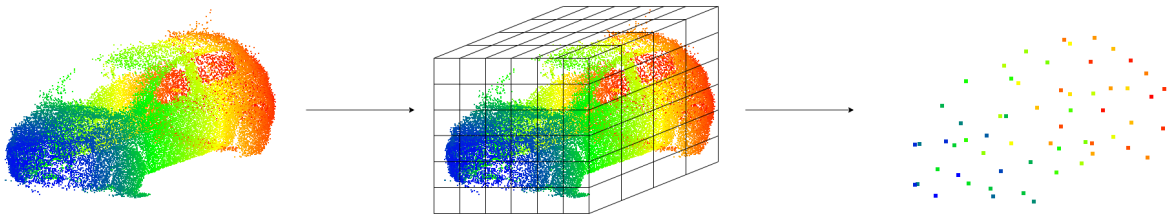

FIGURE 47 – Illustration de l'algorithme de sous-échantillonnage. Une carte volumique (grille de voxels), au centre, est d'abord ajusté sur le nuage de points original, à gauche. Pour chaque élément volumique (voxel), la moyenne des coordonnées des points qu'il contient est calculée. Le nuage sous-échantilloné, à droite, est alors composé uniquement de ces moyennes de points. Le procédé est répété avec une résolution différente pour la carte volumique, jusqu'à arriver à un nuage de points qui contient le nombre de points souhaité.

Le paramètre qui varie le plus selon les objets est le nombre de points par objet. Par exemple, certains arbres du jeu de données *Paris rue Madame* [SMGD14] sont composés de dizaines de milliers de points, alors que d'autres dans le jeu de données *Sydney urban dataset* [QUD12], en compte une centaine tout au plus. Il en va de même pour la densité des nuages de points, lié à la diversité des appareils *LiDAR* utilisés.

Afin d'homogénéiser ces paramètres, nous appliquons sur tous les objets de la *Base A* un filtre volumique (*voxel-grid filter*) pour fixer le nombre de points à 512. Ce filtre consiste à *voxeliser* le nuage de points et à conserver pour chaque *voxel* la moyenne des coordonnées des points. Nous illustrons l'application de cet algorithme sur un nuage de points dans la figure 47. La particularité de ce filtre est qu'il permet d'obtenir en même temps un nombre de point et une densité identique pour tous les objets. Dans le cas des objets qui contiennent moins de 512 points, leurs coordonnées sont répétées. Une fois les nuages pré-traités, nous les centrons en 0 et normalisons chacune de leurs coordonnées entre 0 et 1. Nous passons ensuite ces nuages normalisés en entrée d'un réseau *PointNet* [QSMG17] afin de prédire une classe pour chacun d'entre eux. Ces tests sont réalisés en suivant le protocole d'entraînement par défaut du réseau *PointNet* [QSMG17]. Nous modifions uniquement la sortie du classifieur, pour que celle-ci corresponde au nombre de classes de la *Base A* : 7.

	dispositif d'acqusition	distance parcourue	nombre de points	annotations
acquisition a	sac Leica pegasus backpack	200 m A/R	27 millions	oui
acquisition b	3D laser mapping Robin monté sur sac à dos	400 m A/R	1 milliard	non
acquisition c	3D laser mapping Robin monté sur voiture	3 km	300 millions	non

TABLE 4.3 – Tableau récapitulatif des acquisitions LiDAR que nous avons réalisées

Acquisitions *LiDAR* réalisées

Afin de tester la généricité des *caractéristiques* apprises par le réseau entraîné sur la *Base A*, nous avons décidé de le tester sur un jeu de données issu d'une autre acquisition. Nous avons alors réalisé notre propre acquisition *LiDAR* en milieu urbain. Cette acquisition n'a pas vocation à remplacer la *Base A*, ni même à l'enrichir, mais à permettre de valider nos résultats sur des données prises en conditions réelles.

Nous avons préalablement identifié différents itinéraires où se trouvent des objets urbains pour s'assurer de leur présence dans le nuage de points. Chacun de ces itinéraires est représenté dans la figure 48. Chaque acquisition a été réalisée dans des conditions différentes les unes des autres, par exemple en changeant le mode de locomotion (piéton ou voiture) ainsi que le dispositif *LiDAR* utilisé permettant ainsi de collecter des données avec des caractéristiques différentes.

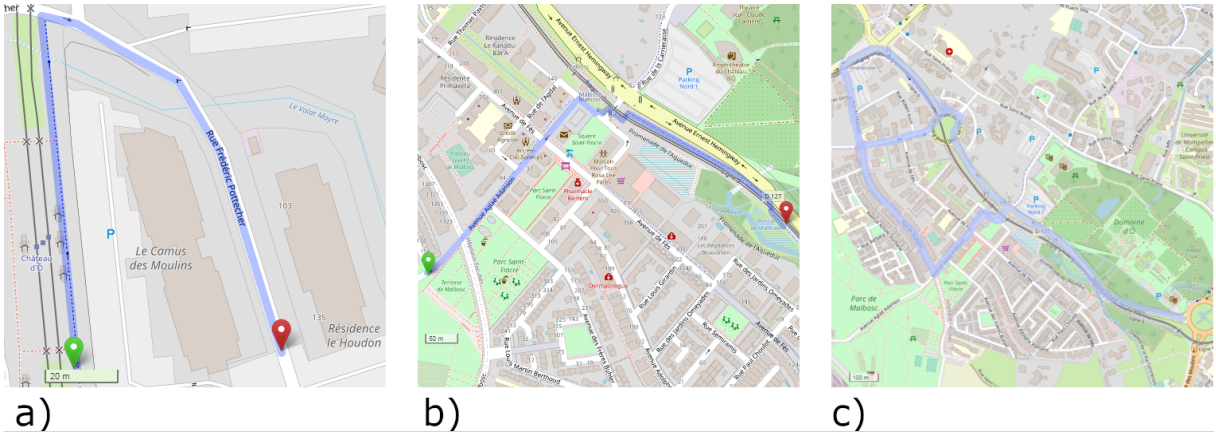

FIGURE 48 – Plan des différents itinéraires des acquisitions *LiDAR* que nous avons réalisé dans la ville de Montpellier (voir tableau 4.3). En (*a*), l'acquisition à l'aide du sac Leica, en (*b*) l'acquisition à l'aide de la solution LaserMapping embarqué en sac à dos et en (*c*) l'acquisition réalisée avec le même dispositif *LiDAR* embarqué sur voiture. Les trajets des acquisitions (*b*) et (*c*) se recoupent partiellement afin d'évaluer la différence entre acquisitions sac à dos et voiture pour un même dispositif *LiDAR*.

Au total, nous avons réalisé 3 acquisitions *LiDAR*, que nous détaillons dans le tableau 4.3. Nous présentons aussi des photos du dispositif utilisé pour l'acquisition (*b*) dans la figure 49, où nous pouvons apercevoir le sac à dos *LaserMapping* embarquant le dispositif *LiDAR*.

FIGURE 49 – Photos de l'appareil 3D laser mapping Robin monté sur un sac à dos, pendant l'acquisition à gauche et isolé à droite.

Classe	voiture	signalisation	poteau	arbre	personne
Nombre d'exemples	39	8	23	75	29
Total	**174**				

TABLE 4.4 – Tableau récapitulatif de la répartition de la *BaseB*.

Nous avons extraits manuellement les objets urbains contenus dans le nuage de points issu de l'acquisition (*a*) à l'aide du logiciel CloudCompare [3]. En tout nous avons isolé 174 objets (voir tableau 4.2.1) dont : 39 objets de classe *voiture*, 8 objets de classe *signalisation*, 23 objets de classe *poteau*, 75 objets de classe *arbre* et 29 objets de classe *personne*. Nous appelons le jeu de données ainsi obtenu *Base B* et nous lui appliquons les mêmes pré-traitements que les nuages de la *Base A*. Comme les classes *bâtiment* et *bruit* ne sont pas des objets à proprement parler, celles-ci ne sont pas représentées dans la *Base B*.

4.2.2 Résultats obtenus et discussion

Nous cherchons à savoir si il est possible de classifier des objets urbains provenant d'acquisitions effectuées dans des conditions réelles à partir d'apprentissage issu de jeux de données publiques. Nous cherchons ainsi à évaluer la généricité des caractéristiques apprises par le réseau et ainsi s'assurer que le réseau ne s'appuie pas seulement sur des caractéristiques spécifiques à certains types d'acquisitions *LiDAR* ou bien aux formes d'objets urbains de certaines villes au dépend des autres.

3. CloudCompare (version 2.9) [GPL software]. (2020). Retrieved from http ://www.cloudcompare.org/

Mesure des performances

La mesure que nous avons privilégiée afin de quantifier les performances du réseau est la *F-mesure* (ou score *F1*) [Chi92]. Il s'agit d'une moyenne harmonique de la *précision*, taux d'objets correctement classifiés par le réseau, et du *rappel*, mesure de la fiabilité des prédictions du réseau. De plus nous présentons aussi la matrice de confusion pour analyser plus en détails le comportement du réseau.

Nous utilisons 80% de la *Base A* comme ensemble d'entraînement (20% de cet ensemble d'entraînement étant utilisé comme ensemble de validation, ceci est également le cas pour les expériences de classification suivantes), les 20% constituant alors l'ensemble de validation, et la *Base B* comme ensemble de test. Les résultats sont reportés dans le tableau 4.5. Dans la suite nous appelons cette expérience l'expérience témoin.

		Vérité terrain (annotations)				
		Voiture (39)	***Signalisation (8)***	***Poteau (23)***	***Arbre (74)***	***Personne (29)***
Prédictions	***"voiture"***	29	0	0	1	0
	"signalisation"	0	6	13	1	2
	"poteau"	0	0	3	0	0
	"arbre"	2	0	1	62	0
	"personne"	0	2	0	0	25
	"bruit"	8	0	1	9	2
	"bâtiment"	0	0	5	1	0
	F-mesure	0.841	0.400	0.231	0.892	0.893

TABLE 4.5 – Matrice de confusion pour l'expérience témoin. *F-mesure* globale : 0.744

Nous remarquons des scores élevés, pour les classes *voiture*, *arbre* et *personne*, avec des *F-mesures* de respectivement 0.841, 0.892 et 0.893. Cependant ces scores sont bien moins élevés pour les classes *signalisation*, 0.4, et *poteau*, 0.231. De plus nous remarquons que 13 des objets de classe *poteau* de la *Base B* sont étiquetés par le réseau comme des objet de classe *signalisation*. Il est alors fortement probable que le réseau opère une confusion entre ces deux classes distinctes.

Malgré la petite taille de notre jeu d'entraînement, son hétérogénéité et le fait que notre ensemble de test provienne d'une acquisition différente de celles utilisées pour le jeu d'entraînement, nous trouvons que les résultats obtenus sont encourageants. Il en reste néanmoins que les faiblesses du réseau pour les classes *signalisation* et *poteau* nous semblent être un frein sérieux pour le passage à la détection d'objets.

Dans la section suivante, nous cherchons alors à expérimenter des méthodes pour augmenter simplement les performances sans toucher au réseau de neurones utilisé.

4.2.3 Expériences supplémentaires

Enrichissement de la base de données

Lors de cette expérience nous avons ajouté à la *Base A* le jeu de données *ParisLille* [RDG18]. Nous regroupons alors les objets qui correspondent à nos classes et nous leurs appliquons les

mêmes pré-traitements que les autres nuages de la *Base A*. En tout 924 objets supplémentaires sont ajoutés, pour un total de 1668 objets pour la nouvelle *Base A*. Nous entraînons alors le réseau sur cette nouvelle *Base A* et nous utilisons la *Base B* comme ensemble de test. Les résultats sont reportés dans le tableau 4.6.

		Vérité terrain				
		Voiture (39)	*Signalisation (8)*	*Poteau (23)*	*Arbre (74)*	*Personne (29)*
Prédictions	*"Voiture"*	39	0	0	2	1
	Signalisation	0	7	13	2	2
	Poteau	0	1	11	0	0
	Arbre	0	0	2	70	1
	Personne	0	0	1	0	25
	Bruit	0	0	0	0	0
	Bâtiment	0	0	0	1	0
	F1 score	0.963	0.467	0.629	0.946	0.893

TABLE 4.6 – Matrice de confusion pour l'expérience de classification d'objets avec enrichissement de la base d'entraînement. *F-mesure* globale : 0.857.

Nous remarquons une nette amélioration des résultats par rapport à notre expérience témoin et ce pour chacune des classes (la classe *personne* mise à part), avec en particulier le score de la classe *signalisation* qui passe de 0.4 à 0.467 et le score de la classe *poteau* qui passe de 0.231 à 0.629.

Ces gains de performance ne sont pas surprenants dans la mesure où la taille de notre jeu de d'entraînement a plus que doublé avec l'ajout de la nouvelle base.

Fusion de classes

Dans les résultats de l'expérience témoin, nous avons remarqué que le réseau était nettement moins performant pour les classes *signalisation* et *poteau*, avec plus de la moitié des exemples de poteaux étiqueté *signalisation* par le réseau. Nous avons alors émis l'hypothèse que le réseau arrive bien à les distinguer des autres classes mais sans faire la différence entre les deux. Afin de tester cette hypothèse nous avons regroupé ces deux classes en une seule, baptisé *signalisation et poteau* et noté *TSLP*. Tous les autres paramètres sont identiques à l'expérience témoin. Les résultats sont reportés dans le tableau 4.7.

		Vérité terrain (annotations)			
		Voiture (39)	*TSLP(31)*	*Arbre (75)*	*Personne (29)*
Classification	*"voiture"*	27	0	1	0
	"TSLP"	0	28	2	5
	"arbre"	3	3	69	0
	"piéton"	0	0	0	24
	"bruit"	9	0	2	1
	"bâtiment"	0	0	1	0
	F-mesure	0.806	0.849	0.920	0.750

TABLE 4.7 – Matrice de confusion pour l'expérience de classification d'objets avec fusion des classes *signalisation* et *poteau*. *F-mesure* globale : 0.831

Nous remarquons que le score de la classe *TSLP* est plutôt élevé avec une *F-mesure* de 0.849,

c'est-à-dire à peu près au même niveau que les classes *voiture*, *arbre* et *piéton* dans l'expérience témoin. La fusion des classes s'accompagne également d'une légère baisse de performance pour ces mêmes classes, avec notamment la classe *personne* qui passe de 0.893 à 0.75.

Nous considérons que notre hypothèse sur la confusion entretenue par le réseau entre les classes *signalisation* et *poteau* est vérifiée. En effet le score de la classe fusionnée *TLSP*, 0.849, est nettement supérieur à ceux des classes *poteau* et *signalisation*, respectivement 0.4 et 0.231. Néanmoins et comme nous pouvions nous y attendre, la réduction du nombre de classes entraîne un espace de caractéristiques moins contraint et donc des caractéristiques moins discriminantes apprises par le réseau. Cet effet se traduit par une baisse des performances pour les autres classes.

Influence du nombre de points

Les auteurs de *PointNet* [QSMG17] montrent qu'au delà d'un certain seuil, le réseau est peu sensible au nombre de points des nuages. Nous avons voulu tester cette hypothèse dans le cas des objets urbains. Ainsi nous avons relancé l'expérience témoin plusieurs fois avec à chaque fois un nombre de points différent. Nous présentons les résultats que nous avons obtenus sous forme de courbe dans la figure 50.

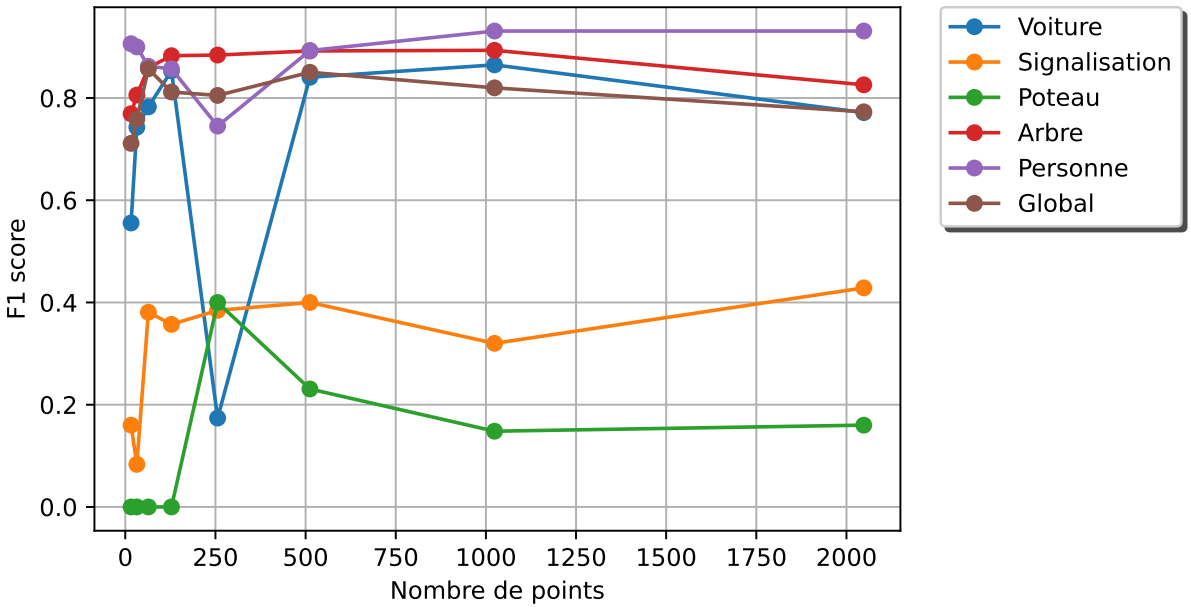

FIGURE 50 – Evolution des performances sur la tâche de classification d'objets, en termes de *F-mesure*, selon le nombre de points des nuages.

Nous remarquons que les résultats obtenus, en termes de *F-mesure*, demeurent relativement stable en fonction du nombre de points, avec tout de même une chute remarquable en dessous de 64 points. Ces résultats sont conformes avec ceux présentés dans [QSMG17]. Nous notons que la baisse des performances induite par la diminution du nombre de points est bien moins prononcée dans notre cas. Nous expliquons cette différence par le fait que nous avons utilisé un sous-échantillonnage par carte volumique qui permet de garder une idée grossière de la forme

des objets malgré le faible nombre de points, alors que dans [QSMG17] les points sont choisis aléatoirement.

Augmentation générique de la base de données

Afin de prendre en compte l'impact des occultations sur la forme des objets, nous avons mis au point deux méthodes *d'augmentation virtuelle* (*data augmentation*) de notre ensemble d'entraînement. Le but de ces augmentations est double :

- Faire apprendre au réseau des caractéristiques à partir de nuages occultés afin qu'il ne soit pas pénalisé par les occultations présentes lors de la phase de test.
- Augmenter la taille de l'ensemble d'entraînement en prenant acte de l'augmentation des performances constatée lors de notre expérience d'enrichissement de la *Base A*.

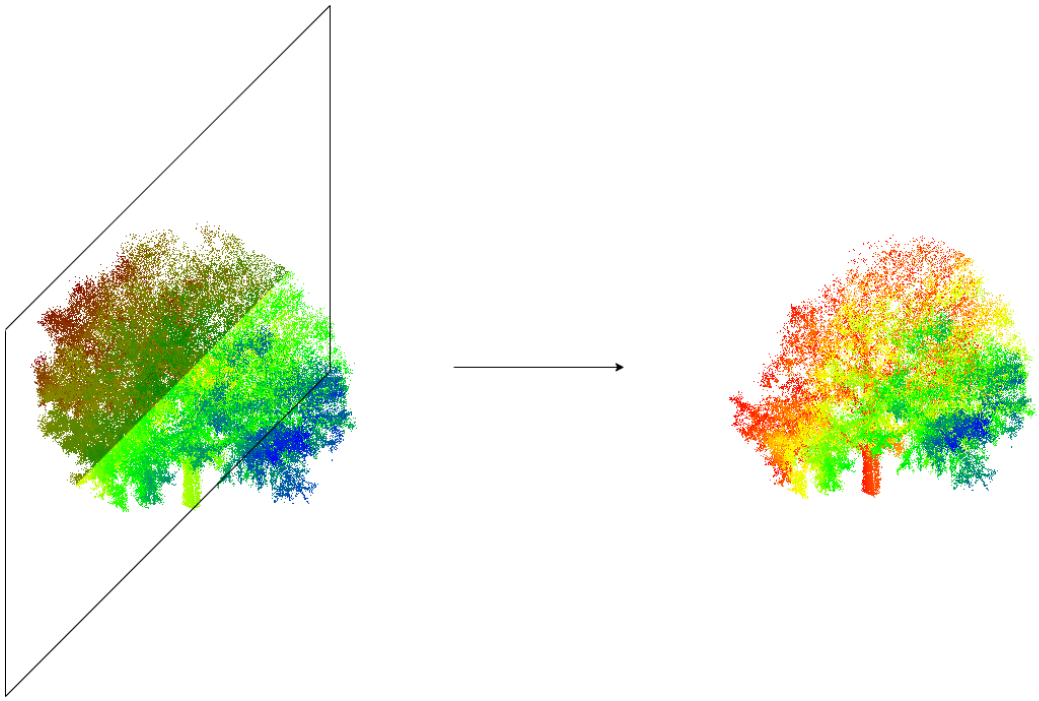

FIGURE 51 – Afin de simuler une occultation sur un nuage de points, un plan aléatoire recoupant l'axe vertical est généré, à gauche. Celui-ci coupe le nuage de points en deux parties, seule la partie la plus grande, en nombre de points est conservée, à droite dans la figure.

La première méthode consiste en la génération aléatoire de plusieurs plans de coupe autour d'un axe vertical. Nous illustrons ce procédé dans la figure 51 où un nuage de points est coupé en deux selon un plan vertical généré aléatoirement. La seconde, plus sophistiquée, consiste dans un premier temps à *voxeliser* le nuage de points, puis à simuler une acquisition *LiDAR* en ne conservant qu'une seule *face* de l'objet. Dans cette seconde méthode (illustrée dans la figure 52) le nuage de points est d'abord *voxelisé* puis des rayons horizontaux sont simulés. Ces rayons ne s'arrêtent que lorsqu'ils rencontre un *voxel* contenant un nombre suffisant de points. Seuls ces points seront présents dans la version de l'objet ainsi occulté artificiellement.

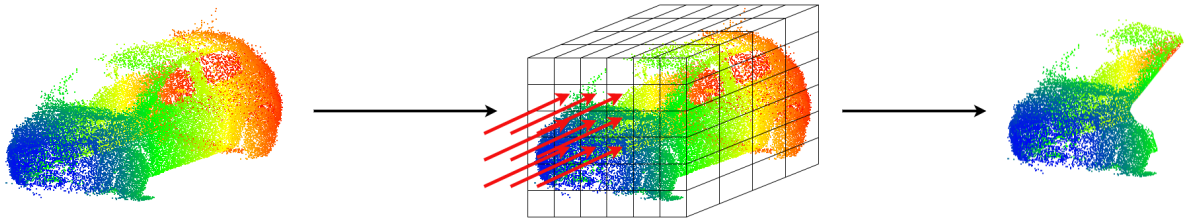

FIGURE 52 – Le nuage de points original, à gauche, subit d'abord une rotation d'angle aléatoire sur l'axe vertical. Ensuite, une *carte volumique*, au centre dans la figure, est calculée autour de ce nuage de points. Des rayons, représentés par des flèches rouges, parcourant la *carte volumique* dans le sens de la profondeur (axe Y représenté par la couleur), sont ensuite simulés. Ceux-ci s'arrêtent uniquement lorsqu'ils 'traversent' un élément volumique contenant des points, en mettant de côté ces derniers. Le nuage de points obtenu, à droite dans la figure, est alors composé uniquement des points précédemment mis de côté.

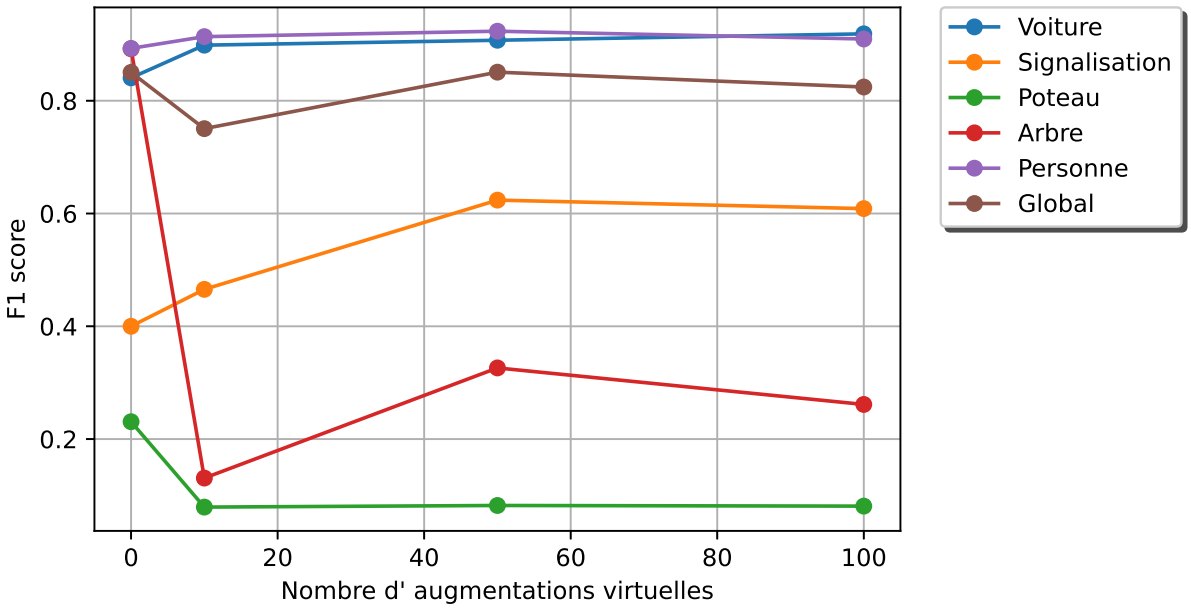

FIGURE 53 – Evolution des performances sur la tâche de classification d'objets, en termes de *F-mesure*, selon le nombre d'augmentations virtuelles par objet ajoutées à la base d'entraînement.

Nous appliquons alors ces méthodes d'augmentation de données sur chaque objet de la *Base A*, en faisant varier le nombre d'augmentations par objet. Nous entraînons ensuite le réseau à partir de la *Base A* augmentée, et ce pour chaque nombre d'augmentations testé. Nous reportons les résultats dans la figure 53. Comme la première méthode engendre des performances nettement inférieures à celles obtenues dans l'expérience témoin, seules les performances de la deuxième méthode sont présentées.

Nous remarquons alors dans le graphique 53 que cette méthode d'augmentation de la base virtuelle ne permet pas d'augmenter globalement les performances du réseau en termes de

F-mesure. Celle-ci engendre une forte baisse du score pour la classe *arbre*, qui passe de 0.892 à moins de 0.4. Ainsi même si cette baisse est contrebalancée par une augmentation pour les classes *voiture*, *personne* et *signalisation*, les résultats ne sont globalement pas satisfaisants. Nous considérons que cela est du au fait que les *ensembles de points critiques* obtenues à partir des objets occultés sont redondants par rapport à ceux obtenues à partir des objets entiers (voir section 2.4.3).

4.3 Evaluation de notre architecture sur la tâche de détection d'objets en milieu urbain

4.3.1 Protocole expérimental

Jeu de données utilisé

La tâche de détection d'objets dans des scènes 3D nécessite d'utiliser des bases de données, qui en plus d'être annotées au niveau *objet*, doivent aussi incorporer des boîtes englobantes pour chaque objet, ce qui n'est le cas d'aucune base de données parmi celles que nous avons relevées (voir tableau 4.1). Il est tout de même possible de générer ces boîtes englobantes en partant des nuages de points de chaque objet isolé :

- le centre de la boîte englobante est la moyenne des coordonnées des points du nuage isolé.
- les dimensions de la boîte englobante sont définies par rapport aux *extrema* du nuage isolé sur chacune des composantes x, y et z.

Néanmoins cette approche nécessite de vérifier après coup chaque boîte englobante ainsi générée afin de vérifier que celle-ci soit au plus proche de l'objet et de l'ajuster dans le cas contraire.

Nous avons choisi de nous concentrer sur le jeu de données [RDG18], car parmi les jeux de données identifiés, il s'agit de celui qui contient le plus grand nombre d'objets et offre le plus de diversité, avec des scènes 3D provenant d'endroits différents. Nous appliquons la procédure de génération des boîtes englobantes précédemment décrite pour compléter ce jeu de données. Afin de minimiser le temps de traitement de la génération de boîtes englobantes ainsi que le temps de vérification de celles-ci, nous utilisons la version réduite de ce jeu de données. Cette version réduite conserve la représentativité réelle des données originales.

Notre jeu de données correspond alors à trois scènes 3D de petite taille, intitulées *Paris*, *Lille A* et *Lille B*. Ce jeu de données compte 170 objets en tout (voir tableau 4.8). Cette base a pour avantage de répartir les objets dans des classes que nous avons précédemment définies pour les objets urbains, c'est-à-dire : les classes *sol*, *bâtiment*, *signalisation* (où sont regroupés également les poteaux), *personne*, *voiture* et *arbre*.

Classe	signalisation	personne	voiture	arbre
Paris	11	17	10	28
Lille A	12	0	24	15
Lille B	7	1	29	16
Total	**30**	**18**	**63**	**59**

TABLE 4.8 – Tableau récapitulatif du jeu de données utilisé pour la détection d'objets.

Nous utilisons alors les scènes *Paris* et *Lille A* comme jeu d'entraînement et la scène *Lille B* comme jeu de validation. Nous avons choisi cette répartition pour avoir un jeu d'entraînement le plus divers possible.

Partitionnement des scènes 3D en blocs

Pour être en mesure de détecter l'ensemble des objets urbains dans une scène 3D, cette scène doit être traitée par le réseau de neurones dans son entièreté. Cependant, les scènes de notre jeu de données comptent plusieurs millions de points chacune, largement au dessus du seuil N_{max}. Nous avons pu contourner ce problème lors de nos expériences de classification grâce aux sous-échantillonnage des nuages de points mais cette solution ne peut pas être applicable en l'état ici. En effet, sous-échantillonner directement une scène 3D composée de plusieurs millions de points vers un nombre inférieur à N_{max}, aurait pour effet d'obtenir un nuage de points où les objets sont constitués par moins d'une dizaine de points chacun. Nous illustrons cette problématique dans la figure 54. En (a) le nuage de points original où le nombre de points N y est trop important pour être passé en entrée de notre réseau, $N > N_{max}$. En (b) le nuage de points sous-échantillonné, le nombre de points est inférieur au seuil N_{max} mais il n'y a qu'une dizaine de points par objet urbain, ce qui est inférieur au seuil N_{min} à partir duquel le réseau peut apprendre à les identifier.

Nous nous inspirons alors des méthodes de segmentation sémantique de scène 3D pour mettre au point une approche par blocs [QSMG17]. Celle-ci consiste à découper au préalable la scène 3D en plusieurs blocs de taille égale recouvrant la totalité de la scène et espacés entre eux par une distance appelée *pas*. Dans le cas de l'ensemble d'entraînement les blocs sont calculés "avec chevauchements", c'est-à-dire avec un *pas* inférieur à la taille des blocs, alors que dans l'ensemble de test il n'y a pas de chevauchement, la valeur du *pas* est égal à la taille des blocs. Nous illustrons dans la figure 55 le découpage par blocs "sans chevauchement" et en figure 56 "avec chevauchements". Cette différence provient du traitement des objets frontières, c'est-à-dire des objets situés entre plusieurs blocs. Ainsi lors de l'entraînement, un chevauchement des blocs permet d'apprendre au réseau plusieurs configurations d'objets frontières afin que le réseau puisse correctement les identifier lors de la phase de test.

Au niveau des coordonnées des points dans chaque bloc, deux approches existent : la normalisation par scène et la normalisation par blocs [ASZ+16]. La normalisation par scène

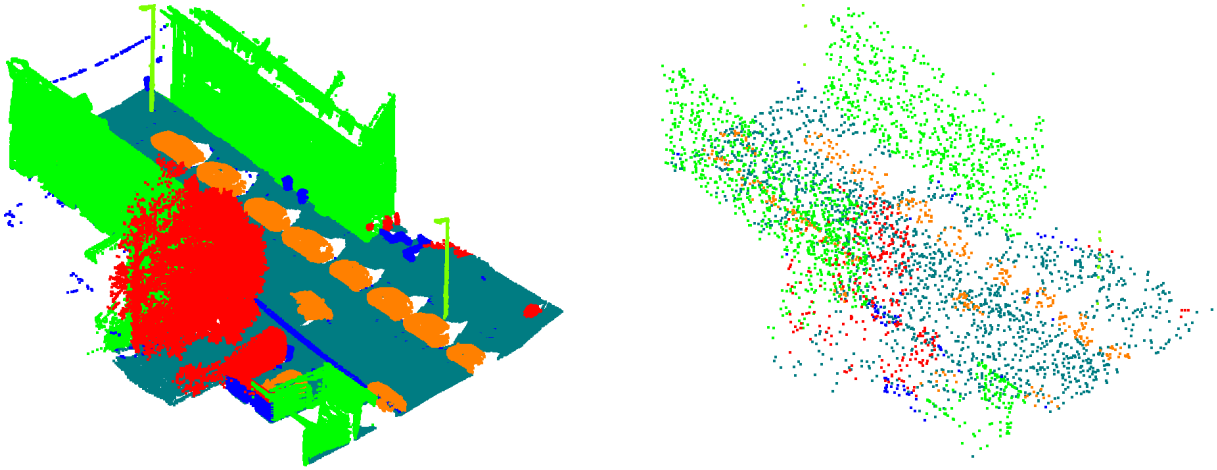

FIGURE 54 – A gauche nous avons la scène 3D originale, coloriée par classes d'appartenance de chaque point pour mieux distinguer les objets. A droite cette même scène 3D sous-échantillonnée afin de contenir un nombre de points inférieur à N_{max}, ici 4096. Nous remarquons alors que les objets, par exemples les voitures, ne sont plus représentés que par quelques dizaines de points, ce qui est bien inférieur à la valeur de N_{min} que nous avons identifiée en section 4.2.3.

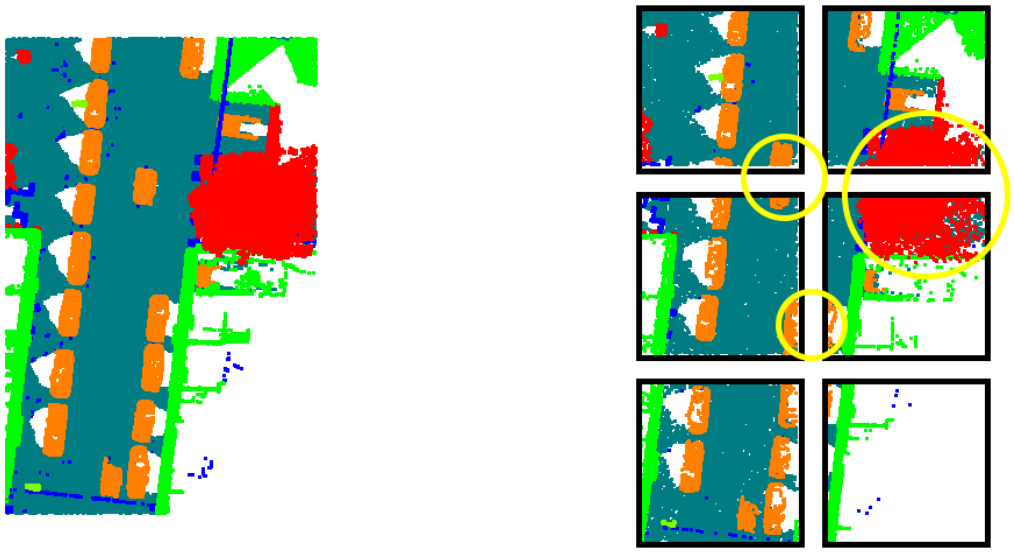

FIGURE 55 – Découpage de la scène 3D originale, à gauche, en plusieurs blocs, à droite. Nous pouvons voir dans les blocs, à l'intérieur des cercles jaunes, les objets frontières. Les points sont coloriés par rapport à leurs classes d'appartenance afin de mieux distinguer les objets.

consiste à exprimer les coordonnées de chaque point **P** du bloc en fonction des coordonnées du nuage original S :

$$\mathbf{P} = \left(x, y, z, \frac{x - X_{min}}{X_{max} - X_{min}}, \frac{y - Y_{min}}{Y_{max} - Y_{min}}, \frac{z - Z_{min}}{Z_{max} - Z_{min}}\right) \tag{4.1}$$

où X_{min} (respectivement Y_{min} et Z_{min}) représente la coordonnée minimale en x (respectivement en y et z) parmi les points présents dans le nuage S, X_{max} (respectivement Y_{max} et Z_{max}) représente la coordonnée maximale en x (respectivement en y et z) parmi les points présents

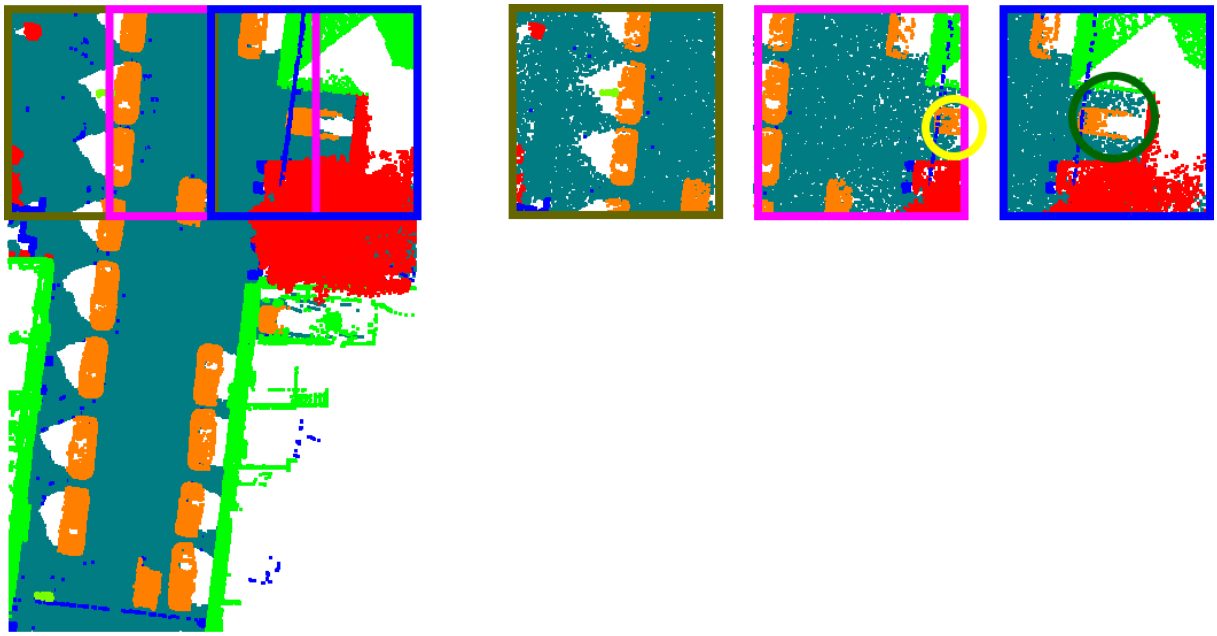

FIGURE 56 – Découpage de la scène 3D originale, à gauche, en plusieurs blocs se chevauchant, à droite. Nous remarquons qu'un objet frontière dans un bloc, dans le cercle jaune, peut être contenu dans son entièreté dans un bloc adjacent. Les points sont coloriés par rapport à leurs classes d'appartenance afin de mieux distinguer les objets.

dans le nuage S et (x, y, z) sont les coordonnées originales du point $\mathbf{P}$.

Cette approche suppose une certaine régularité dans la répartition des objets urbains dans le nuage de points initial. Elle est par exemple utilisée dans le cadre de scènes en intérieur [QSMG17].

La normalisation par blocs revient simplement à exprimer les coordonnées de chaque point par la taille du bloc dans lequel il se trouve. Cela correspond à l'équation 4.1 mais où le minimum et le maximum des coordonnées de chaque composante, sont calculés uniquement par rapport aux points du même bloc et non par rapport aux points du nuage original.

Nous avons choisi l'approche de normalisation par blocs car dans notre cas il n'existe pas de relations à priori entre la scène 3D et la répartition des objets. Cette répartition dépend de l'itinéraire parcourue lors de l'acquisition, de l'endroit numérisé (par exemple parc, quartier résidentiel...) ainsi que de la ville (ou du pays) où celle-ci a été réalisée.

Une fois les scènes 3D découpées en blocs, nous sous-échantillonnons chacun de ces blocs de sorte que le nombre de points de ceux-ci soit inférieur au seuil N_{max}. Il est alors possible d'obtenir des objets qui sont constitués d'un nombre de points supérieur à N_{min} selon la taille des blocs utilisée. Expérimentalement, cette condition est vérifiée à partir d'une taille de bloc inférieure à 15×15 mètres. Le sous-échantillonnage est appliqué sur chaque bloc indépendamment des autres, de la même manière que lors du pré-traitement des objets lors des expériences de classification de la section 4.2.1.

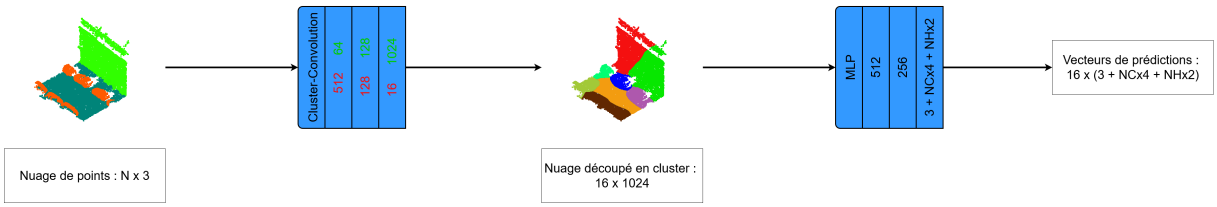

FIGURE 57 – Détail de l'architecture de notre modèle utilisé pour la détection d'objets. Le nuage de points est premièrement passé en entrée de 3 couches de *cluster-convolution* successives, les paramètres de celles-ci sont indiqués sur la figure : en rouge le nombre de clusters et en vert le nombre de caractéristiques extraites. Puis le module de prédiction de boîte englobante implémenté par un perceptron multi-couches (MLP), où N_C désigne le nombre de classes et N_H le nombre de secteurs angulaires pour prédire l'orientation des boîtes englobantes.

Détails sur notre architecture

Nous pouvons alors entraîner notre méthode par *clusters* présentée dans le chapitre 3 à partir de ces blocs afin de détecter les objets qu'ils contiennent. Pour implémenter notre méthode, nous nous inspirons des modèles présentés dans [WSL+19], [LBS+18] et [VBV18], eux-mêmes basés sur l'architecture 2D *Alex-Net* [KSH17]. Dans la suite nous appelons l'architecture qui en découle *RPC* pour *réseau avec proposition de cluster*.

Notre modèle *RPC* (voir figure 57) est ainsi constitué de trois couches de *cluster-convolutions* successives. Celles-ci sont appliquées de manière hiérarchique avec un nombre de *clusters* décroissant et un nombre de *caractéristiques* extraites croissants. La prédiction de boîte englobante est ensuite appliquée sur la sortie de la troisième couche de *cluster-convolution* pour calculer les *vecteurs de prédiction* pour chaque *cluster*. Nous avons implémenté cette étape par un *perceptron multi-couches* dont la sortie correspond aux *vecteurs de prédiction*. A partir de ces vecteurs nous calculons la fonction de coût que nous décrivons en section 3.4.3. Le détail du paramétrage pour chacune des couches est donné dans la figure 57 représentant l'architecture de notre modèle. Celui-ci utilise la normalisation par lot (*batch normalization*) sur chaque calcul de *cluster-convolution* ainsi que l'opération de *point dropout* définie par [QSMG17] lors de l'entraînement.

Choix de la mesure d'évaluation

Afin d'évaluer les performances de notre architecture, nous avons choisi de nous baser sur la *précision moyenne* (*average precision* ou *AP*) afin de correspondre aux standards en termes de détection d'objets 2D [EEV+15] ainsi qu'aux premiers travaux de détection d'objets 3D [GLU12][SLX15].

Les prédictions de notre réseau se présentent sous la forme d'une liste de couples boîte englobante/*score*. La valeur de *score* correspond à un indice de confiance, compris entre 1 et 0, de la boîte englobante associée. Nous évaluons d'abord, pour chaque classe, le nombre de prédictions exactes, fausses et manquées en calculant *l'intersection sur l'union* (*intersection*

over union ou *IoU*) entre les boîtes prédites et les boîtes vérité terrain. Nous considérons qu'une prédiction est valide si sa valeur de *IoU* est supérieure à 0.25 [DCS+17].

Nous pouvons alors calculer la *précision*, proportions de prédictions valides, et le rappel, proportions d'objets détectés parmi l'ensemble des objets. Ces valeurs dépendent du seuil que nous avons déterminé, c'est-à-dire de la valeur minimale de *score* d'une prédiction pour que celle-ci soit prise en compte dans le calcul de la précision et du rappel. Nous pouvons alors tracer une courbe *ROC*, pour chaque classe, en faisant varier la valeur de ce seuil entre 1 et 0 et nous définissons la valeur de l'indicateur *AP* comme l'aire sous cette courbe.

Cependant, comme les valeurs de *score* sont prédite par le réseau, il est possible que des boîtes englobantes avec un *score* moins élevé soient plus valides que des boîtes avec un *score* plus élevé, ce qui donne alors à la courbe *ROC* une allure en 'zigzag'. Afin d'atténuer ces variations, l'indicateur *AP* est calculé à partir d'une version lissée de la courbe *ROC*. Cette méthode permet ainsi de restituer plus fidèlement les performances dans le cas de la détection d'objets par rapport au calcul de la *F-mesure*. Nous adoptons la modalité de calcul définie par le concours *VOC* 2011 [EEV+15], qui consiste à sommer la valeur de l'aire sous la courbe en chaque point où la valeur du rappel est différente :

$$AP = \sum (r_{i+1} - r_i)\, \rho(r_{i+1})$$

où les r_i sont les niveaux de *rappel* correspondant aux prédictions du réseau et $\rho(r)$ la valeur de *précision* interpolée pour une valeur de *rappel* r.

4.3.2 Études comparatives

Comme nous le rappelons dans la section 4.1, nous n'avons pas trouvé dans la littérature d'autres architectures de *deep-learning 3D* appliqués à la détection d'objets urbains. Ainsi il est particulièrement compliqué de comparer notre approche à l'état de l'art en termes de performances obtenues. Cette comparaison est d'autant plus compliquée que nous n'avons pas connaissance d'un de jeu de données de référence en ce qui concerne la détection d'objets urbains.

Une expérimentation possible pour donner un ordre de comparaison pourrait être d'entraîner et de tester les réseaux existants en détection d'objets intérieur (ou orienté véhicule autonome) sur notre jeu de données. Cependant cette comparaison ne serait pas équitable et en défaveur de ces méthodes car elles se basent, entre autres, sur une hypothèse différente de la notre en termes de répartition des objets. Ainsi les tests que nous avons réalisés à l'aide de l'architecture *VoteNet* [QLHG19] ont révélé que celle-ci cherchait à apprendre des répartitions caractéristiques d'objets urbains dans des blocs. Or comme nous le montrons dans la section 4.2.1, cette hypothèse n'est pas valide dans notre cas.

Nous avons alors décidé de comparer notre réseau à une approche consistant à entraîner

	Moyenne	Signalisation/Poteau	Personne	Voiture	Arbre
Segmentation sémantique + *clustering* hiérarchique	0.22	**0.11**	0	**0.64**	**0.11**
RPC (notre architecture)	**0.38**	0	**1**	0.52	0.10

TABLE 4.9 – Résultat de notre expérience comparative. La mesure utilisée est la précision moyenne (*AP*). Nous comparons les résultats de notre architecture aux résultats obtenus grâce à un réseau de segmentation sémantique suivi d'une méthode de *clustering* hiérarchique.

un réseau avec notre jeu de données, sur la tâche de *segmentation sémantique* uniquement, puis d'appliquer des méthodes de regroupement en *post-traitements* afin de calculer des boîtes englobantes à partir de la segmentation prédite par le réseau. Ces expériences n'ont pas pour objet de prouver une quelconque supériorité au niveau des méthodes d'extraction de caractéristiques que nous utilisons mais d'appuyer notre argument selon lequel la recherche en détection d'objets urbains est bien une tâche à part dans le champ du *deep-learning 3D* et que cette tâche ne saurait être inférée à partir d'applications sur la segmentation sémantique.

Pour l'approche par segmentation sémantique, nous avons choisi d'appliquer le réseau *PointNet++* [QYSG17] sur notre jeu de données, puis d'appliquer une méthode de *clustering* adaptatif sur les prédictions du réseau afin d'en déduire des boîtes englobantes. Nous avons fixé les paramètres de cette méthode manuellement, en procédant par essai et erreur directement sur l'ensemble de test. Ainsi l'approche par segmentation sémantique est avantagée du fait que le choix de ces paramètres est la limite principale des méthodes sans apprentissage (voir section 2.1.1 pour plus de détails).

En tout, les deux réseaux, *PointNet++* et *RPC* sont entraînés sur 53 blocs provenant du jeu d'entraînement (*Paris* + *Lille A*) et testés sur 16 blocs provenant du jeu de validation (*Lille B*). Les résultats en termes d'*AP* sont reportés dans le tableau 4.9. Lors de l'entraînement, les deux réseaux apprennent à reconnaître les classes appartenant au fond, c'est-à-dire le sol, les façades de bâtiment et les artefacts de scan. Cependant seules les classes correspondant à des objets sont indiquées dans les résultats.

Nous constatons alors une légère avance de l'approche par segmentation sémantique sur les classes *arbre*, *signalisation* et *voiture*. Bien que *RPC* (notre architecture) obtienne un score global plus élevé, $mAP = 0.38$, celui-ci est dû à la configuration de la scène test qui ne contient qu'un seul objet de classe *personne* (voir tableau 4.8), manqué par l'approche par segmentation sémantique.

Toutefois, nous interprétons ces résultats comme un argument envers le développement et l'amélioration des réseaux orientés détection d'objets. En effet, bien que le protocole de test soit nettement à l'avantage de l'approche par segmentation sémantique, et que les performances sur la tâche de segmentation sémantique sont très hautes, le résultat en termes de détection d'objets est décevant. Cette expérience montre ainsi, d'après nous, que se risquer à inférer une prédiction de boîtes englobantes à partir d'un réseau orienté segmentation sémantique ne peut pas être une solution à moyen/long terme. Nous pensons au contraire que seul le développement des réseaux

orientés détection d'objets peut permettre de s'atteler à cette tâche dont nous espérons avoir démontré la difficulté dans le cadre des objets urbains.

4.3.3 Détails sur le processus de *clustering*

Comme détaillé dans le chapitre 3, les prédictions de notre architecture sont calculées à partir de la dernière couche de *clustering*. Ainsi, nous proposons d'analyser en détails les *clusters* calculés par notre architecture par le biais des couches de *clusters-convolution*, afin d'évaluer le comportement de notre architecture sur le problème de *partitionnement*.

Influence de la qualité des *clusters*

Le partitionnement que nous proposons dans notre architecture dépend, en dernière analyse, des *clusters* formés lors du calcul de la dernière couche de *cluster-convolution*. En effet, c'est à partir de ces *clusters* que sont définis les *vecteurs de prédiction* qui servent ensuite à encoder les boîtes englobantes prédites par le réseau. Comme nous avons discuté dans la section 3.4.2, certaines répartitions peuvent compliquer ou bien faciliter la prédiction de ces boîtes. Nous cherchons alors à analyser plus finement quel est l'impact de la répartition des *clusters* sur les performances du réseau.

Nous définissons un critère de *qualité* des *clusters* visant à quantifier à quel degré chaque *cluster* englobe un objet unique. Cet indice vaut 1 quand le *cluster* contient uniquement tous les points d'un même objet et 0 quand il ne contient aucun point appartenant à un objet. Cet indice est calculé pour chaque *cluster* K en fonction de son *objet majoritaire* O_{maj} :

$$\text{qualité} = \frac{Card\left(O_{maj} \cap K\right)}{Card\left(O_{maj} \cup K\right)} \tag{4.2}$$

où l'opérateur *Card* est le cardinal.

Pour avoir une idée précise du lien entre prédictions du réseau et *qualité* des *clusters* (Eq 4.2), nous récupérons les *clusters* associés à chacune des prédictions, positives comme négatives, du réseau. Nous sélectionnons ensuite les *clusters* qui contiennent un nombre de points qui appartient à un objet supérieur au seuil N_{min}, afin de prendre en compte les limites de l'identification. Pour chacun des *clusters* sélectionnés, nous calculons l'intersection sur l'union (IoU) entre la prédiction qui lui est associée et la vérité terrain. Nous représentons les résultats pour chaque *cluster* dans le graphique 58 selon leurs indices de *qualité* (Eq 4.2).

Nous remarquons alors dans le graphique 58 que la *qualité* des *clusters* est positivement corrélée à la validité des prédictions, évaluée par l'intersection sur l'union (IoU) avec la boîte englobante vérité terrain. Nous évaluons cette corrélation à 0.73 à l'aide d'un taux de corrélation de Pearson [FPP07]. Ainsi, les *outliers* sont en grande partie dus aux erreurs *d'identification*, telles que nous les évoquons dans la section 4.2.

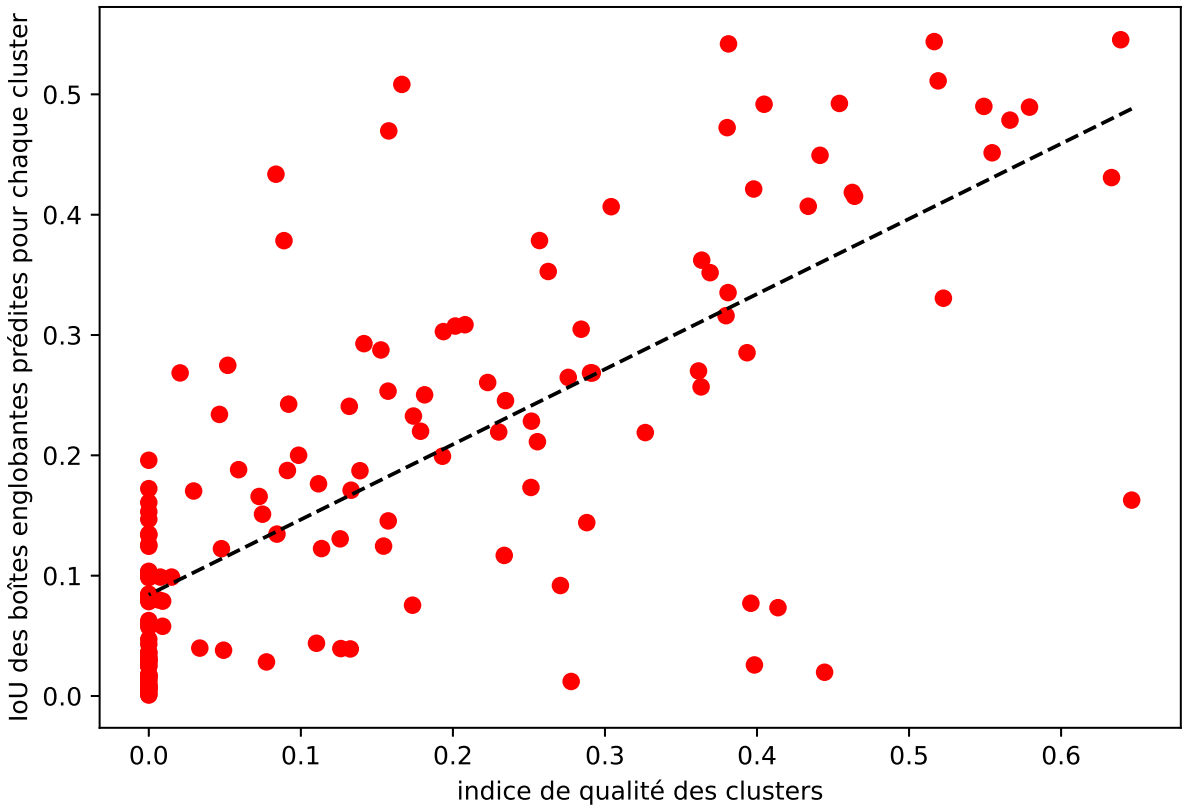

FIGURE 58 – Evaluation des boîtes englobantes prédites par chaque *cluster* en fonction de la qualité de ces *clusters*. Chaque point représente une boîte englobante prédite par le réseau ainsi que le *cluster* à partir duquel cette prédiction est calculée. Le coefficient de corrélation de *Pearson* entre la qualité des *clusters* et la validité des boîtes englobantes prédites, mesurée par l'*IoU* avec les boîtes vérité terrain, est de 0.73.

Nous en tirons alors comme conclusion qu'une bonne répartition des *clusters* implique des prédictions de boîtes englobantes valides par le réseau.

Fonction de distance et *partitionnement*

Comme nous avons mis en évidence que la validité des prédictions du réseau est fortement liée à la qualité des *clusters* calculés par notre architecture, nous souhaitons analyser les déterminants de cette répartition. Nous cherchons alors à tester l'hypothèse, émise en section 3.4.2, selon laquelle cette répartition dépend fortement de l'équilibre entre partie *géométrique* et partie *caractéristique* dans le calcul de la fonction de distance.

Nous détaillons le protocole que nous avons mis au point afin d'évaluer l'influence de la fonction de distance sur la formation des *clusters* ainsi que les résultats obtenus dans la figure 59. En fixant l'initialisation de l'algorithme de *clustering*, nous faisons varier, pour un même bloc, le coefficient, noté λ, affecté à la partie *géométrique* dans le calcul de la fonction de distance. Une valeur de λ à 1 implique une distance uniquement géométrique et une valeur de λ à 0 implique une distance calculée uniquement sur les *caractéristiques*. Nous calculons ensuite

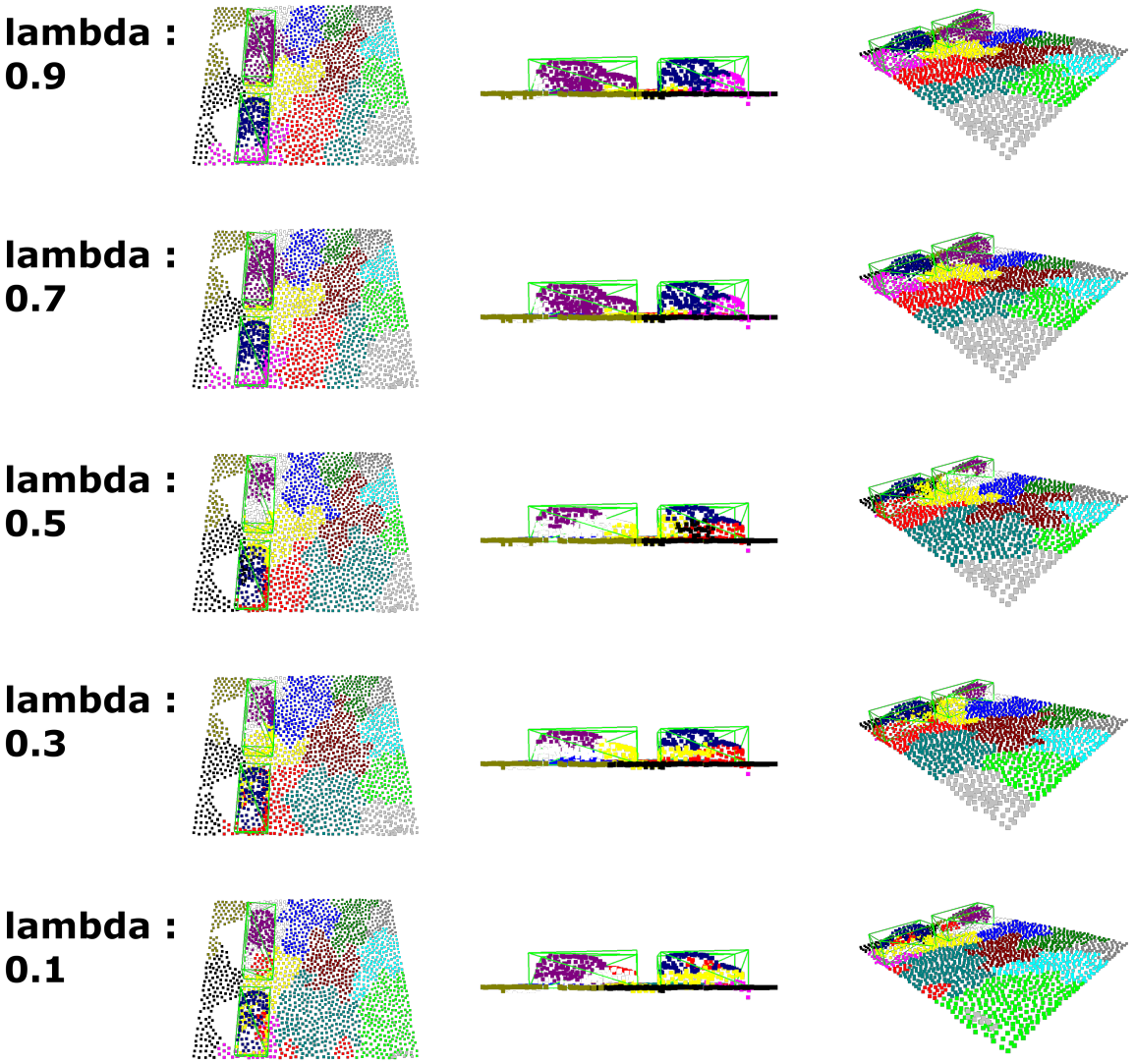

FIGURE 59 – Illustration de l'influence du coefficient λ dans le calcul de la fonction de distance sur la répartition des *clusters*. Nous faisons varier le coefficient λ entre 0.1 et 0.9 pour un même bloc et nous représentons la répartition des *clusters* qui en résulte. Les points de même couleur appartiennent à un même *cluster*. Nous représentons plusieurs angles de vue sur le même bloc afin de mieux distinguer l'évolution de chaque *cluster*.

l'indice de *qualité* pour 2 *clusters* témoins à chaque valeur de λ, et nous reportons ces valeurs dans le graphique 60. Ainsi pour chaque niveau du coefficient λ, la répartition des *clusters* se trouve modifiée. Notons que la relation entre l'indicateur de *qualité* des *clusters* témoins et le coefficient λ n'est pas linéaire. Dans le cas des *clusters* utilisés pour notre expérimentation, une valeur de λ supérieure à 0.7 donne les meilleurs résultats, car plus adaptée aux objets les plus compacts, comme les voitures, où la proximité dans l'espace géométrique est plus importante.

En choisissant expérimentalement une valeur de λ de 0.8, nous obtenons globalement des résultats satisfaisants. Néanmoins, nous notons que selon les cas, des valeurs de λ différentes

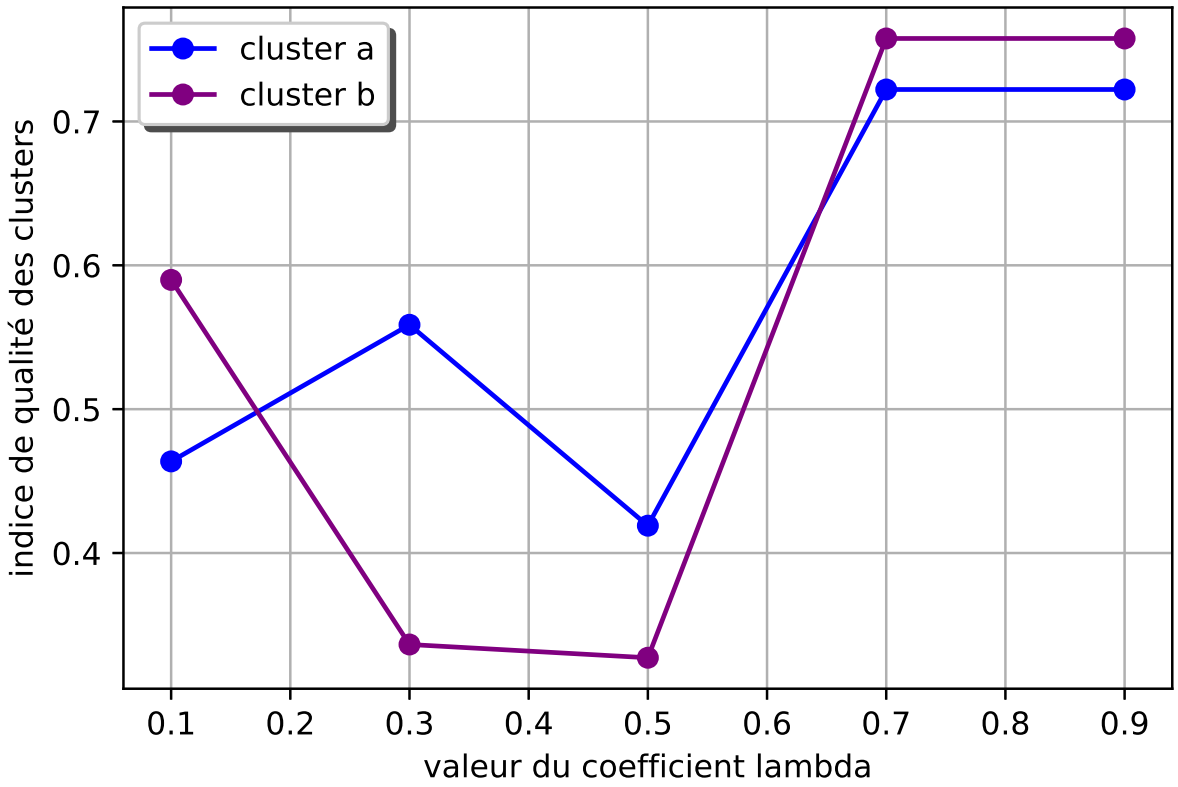

FIGURE 60 – Evolution de la *qualité* de 2 *clusters* en fonction du coefficient λ. Nous partons de 2 *clusters* avec un bon indicateur de *qualité*, obtenus avec $\lambda = 0.9$. Ces 2 *clusters* correspondent à ceux qui recouvrent initialement le mieux les 2 voitures du bloc représenté dans la figure 59.

peuvent donner de meilleurs résultats pour certains *clusters* ou certains *blocs*.

Influence du nombre de clusters

Un autre paramètre important impactant la répartition des *clusters* est le nombre k de *clusters* calculés par le réseau lors de la dernière couche de *cluster-convolution*. Afin d'évaluer ce paramètre, nous faisons varier le nombre k et calculons, pour 3 séries de blocs adjacents (voir figure 61 pour un exemple d'une série de 3 blocs), la qualité moyenne des *clusters* pour chaque valeur de k.

Nous présentons dans la figure 62 les résultats obtenus et dans la figure 61 la répartition des *clusters* pour $k = 8$ et $k = 16$. Nous remarquons alors que l'évolution de la *qualité* moyenne des *clusters* en fonction de k est non linéaire. En effet, comme nous pouvons le noter dans la figure 61, une valeur de k plus petite implique des *clusters* plus grands, en termes de nombre de points, et donc mieux adaptée aux objets volumineux comme les arbres. Inversement, une valeur de k plus grande implique des *clusters* moins grands, en termes de nombre de points, et donc mieux adaptée aux objets moins volumineux comme les personnes ou les voitures. Dans le cas des 3 blocs adjacents présentés en figure 61, la valeur $k = 12$ est la meilleure. En effet comme ces blocs contiennent quasiment autant d'arbres que de voitures, cette valeur permet un compromis

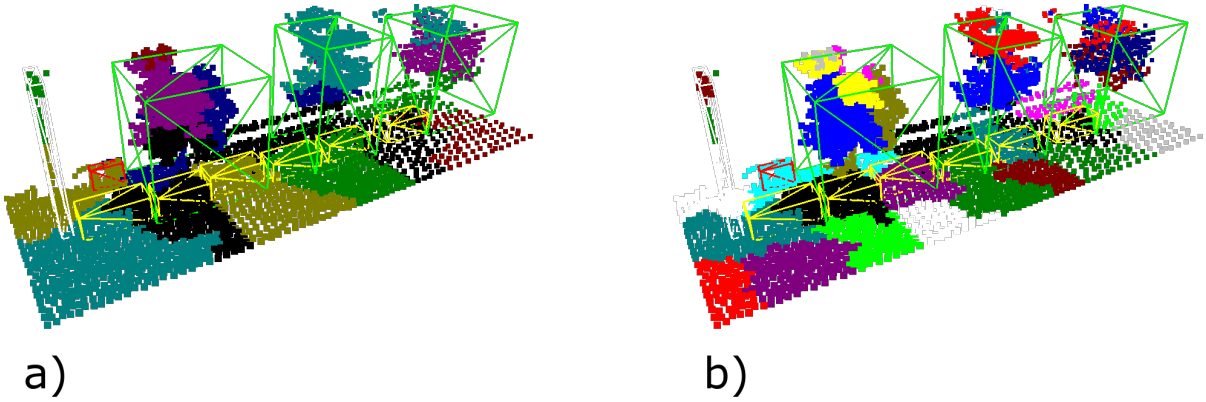

FIGURE 61 – Illustration de l'influence du nombre de *clusters* sur la répartition des *clusters* dans 3 blocs adjacents. Pour des raisons de lisibilité, seuls les cas avec 8 *clusters*, en (*a*), et avec 16 *clusters*, en (*b*), sont représentés. Les points de même couleurs appartiennent à un même *cluster* et les boîtes englobantes de vérité terrain permettent de mieux distinguer les objets présents.

entre les *clusters* adaptés aux arbres et ceux adaptés aux voitures. Cependant, pour un bloc avec une répartition différente, par exemple qui contiendrait plus d'arbres que de voitures, une autre valeur de k serait préférable. Ainsi comme pour le coefficient λ, il n'existe pas de valeur à priori optimale pour k.

Une solution serait alors de prédire des boîtes englobantes après chaque opération de *cluster-convolution*, avec un nombre décroissant de *clusters*. Cette idée s'inspire du réseau 2D *SSD* [LAE+16] et revient à prédire d'abord les objets les plus petits, puis les objets plus volumineux au fur et à mesure que les *clusters* s'agrandissent.

Illustrations / exemples

Ici nous passons en revue un certain nombre de prédictions faites par le réseau, aussi bien des valides que des erreurs, que nous essayons d'interpréter à la lumière des expériences décrites précédemment. Le but étant avant tout de mieux comprendre le comportement du réseau dans les cas problématiques, nous analysons majoritairement des exemples où le réseau commet une erreur de prédiction.

Dans la figure 63 en (*a*) nous présentons une répartition des *clusters* dans un arbre. Bien que plusieurs *clusters* soient présents dans le feuillage de l'arbre, le réseau arrive tout de même à prédire une boîte englobante valide (*e*) autour de l'arbre. En effet, les prédictions provenant des autres *clusters* ont un score attribué par le réseau inférieur et sont donc mises de côté par l'algorithme de fusion des boîtes englobantes appelé *non maximum suppression* ou *NMS*. Celui-ci consiste [4] à supprimer, pour chaque boîte englobante, les boîtes qui l'intersectent et qui ont un score de prédiction inférieure.

En (*b*) nous présentons un problème de partitionnement où un *cluster*, dans le cercle vert,

4. L'algorithme décrit ici est la forme basique du *NMS*, plusieurs formes plus complexes existent comme par exemple [BSCD17]

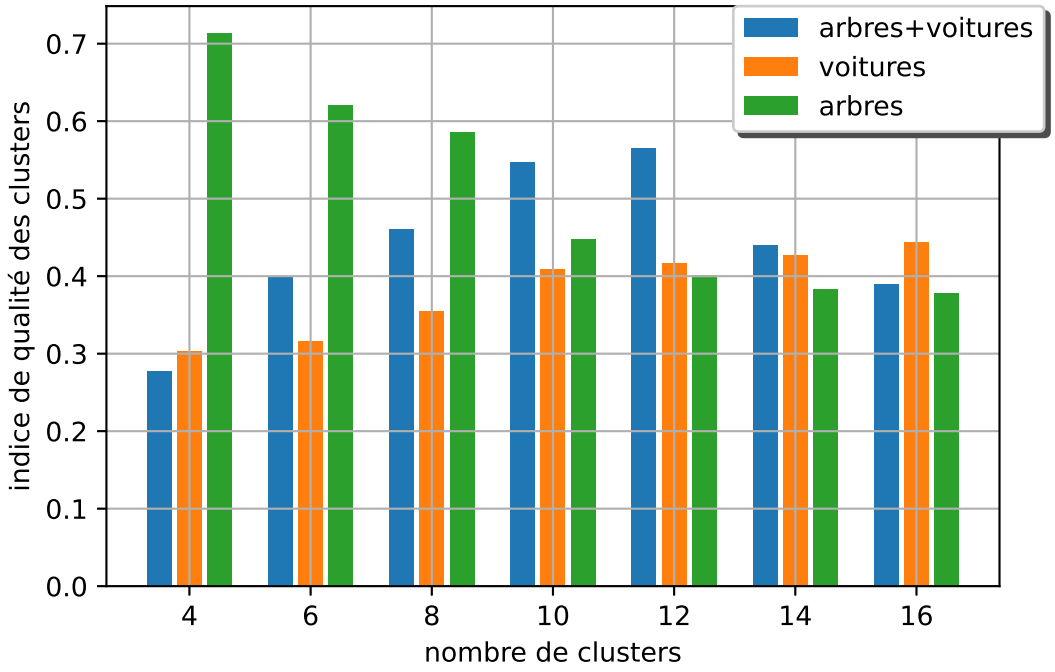

FIGURE 62 – Evolution de la *qualité* des *clusters* présents dans 3 séries différentes de blocs adjacents selon le nombre k de *clusters*. En vert une série de blocs où sont présents uniquement des arbres, en orange une série de blocs où sont présents uniquement des voitures et en bleu une série de blocs où sont présents à la fois des voitures et des arbres. Cette dernière série de blocs est représentée en figure 61.

se retrouve à cheval sur deux objets. Le réseau prédit alors bien une voiture (f) mais la boîte englobante prédite se retrouve alors entre les deux boîtes vérité terrain. Dans ce cas l'erreur provient probablement du choix des paramètres λ et k. En effet, même si nous avons fixé les valeurs de λ et k à partir d'expériences globales, il existe toujours des configurations où des valeurs différentes peuvent donner des meilleurs résultats. Par exemple, une valeur plus grande de k pourrait éviter la formation de *cluster* assez grand pour être "à cheval" sur deux voitures et une valeur plus petite de λ pourrait éviter de regrouper dans un même *cluster* des points appartenant à des voitures et autant de points appartenant au sol.

En (c) nous exposons un autre problème de partitionnement lié cette fois au découpage en blocs. En effet nous remarquons la présence d'un objet frontière (une voiture à cheval entre deux blocs), indiqué dans le cercle vert. Le réseau prédit une boîte englobante à partir du *cluster* qui recoupe la partie de la voiture qui est dans chaque bloc (g), ce qui donne une boîte mal centrée sur la voiture. Nous pouvons alors faire l'hypothèse que le réseau a interprété ce *cluster* comme étant l'arrière d'une voiture qui serait présente en entier dans le bloc. Ce type d'erreur peut être évité en incorporant une stratégie de traitement spécifique des objets frontières. Ainsi une méthode simple consiste à mettre de côté les boîtes englobantes prédites par le réseau qui sont trop proches des frontières du bloc, puis de relancer le réseau avec des nouveaux blocs centrés sur ces boîtes englobantes.

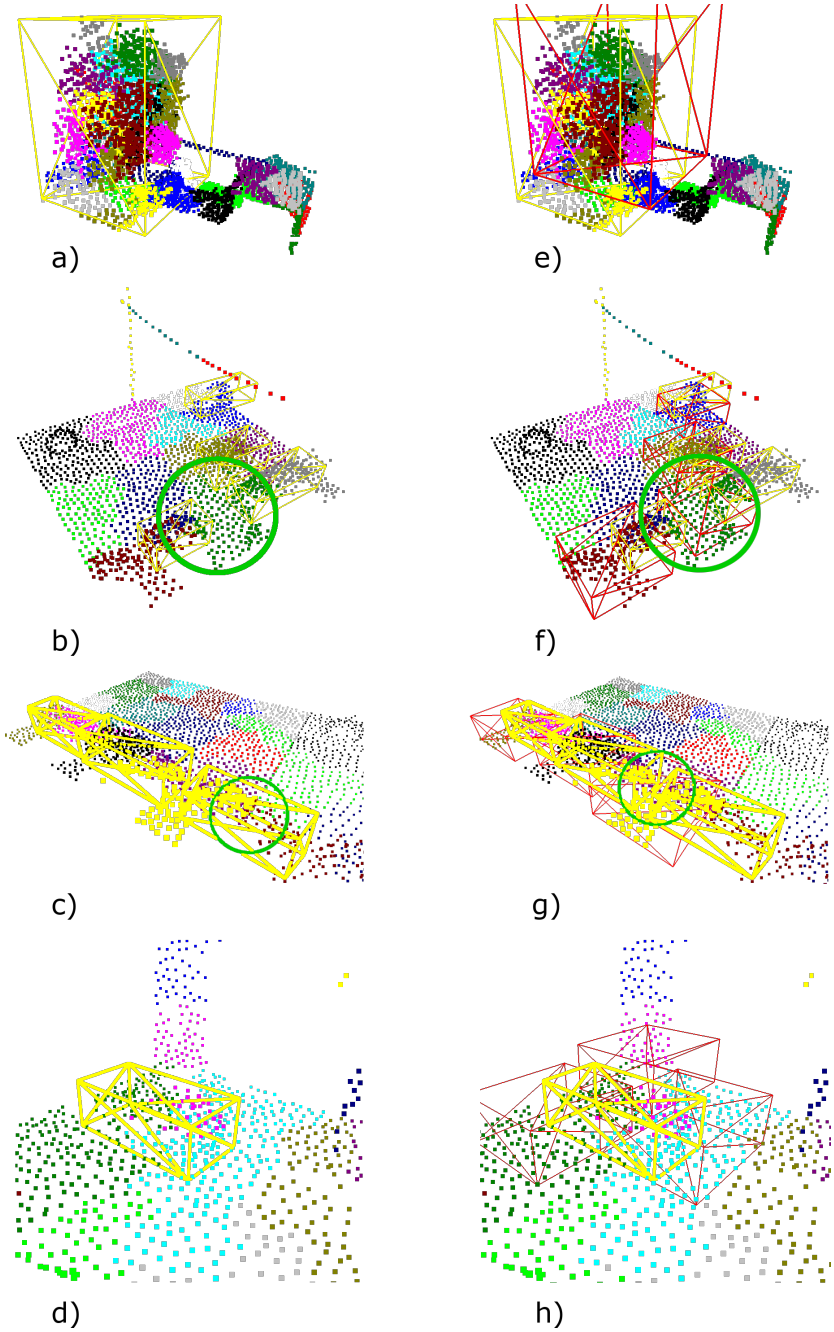

FIGURE 63 – Illustrations d'exemples de prédictions sur la tâche de détection d'objets à partir de notre architecture. Dans la colonne de gauche seules les boîtes vérité terrain sont représentées, en jaune, et dans la colonne de droite les boîtes prédites sont aussi représentées, en rouge. Les points de même couleur appartiennent à un même *cluster*.

En (*d*) nous présentons un dernier type de problème de partitionnement, où un même objet est présent (avec une quantité de points suffisante) dans plusieurs *clusters* différents. Ici une même voiture est présente dans 3 *clusters*, qui essayent chacun de prédire une boîte englobante à partir du fragment de voiture qu'il contient. Le résultat qui en découle est alors erroné puisque 3 boîtes englobantes différentes sont prédites pour un même objet (*h*).

4.3.4 Tests d'ablation

Dans cette section nous proposons une série d'expériences réalisées afin de justifier nos choix d'implémentation. Nous fixons le paramètre λ et utilisons 3 valeurs différentes de k (4, 8 et 16) afin de se concentrer sur le choix des fonctions utilisées. La mesure du *mAP* obtenu (voir figure 64) est utilisée pour comparer entre elles les différentes possibilités d'implémentation.

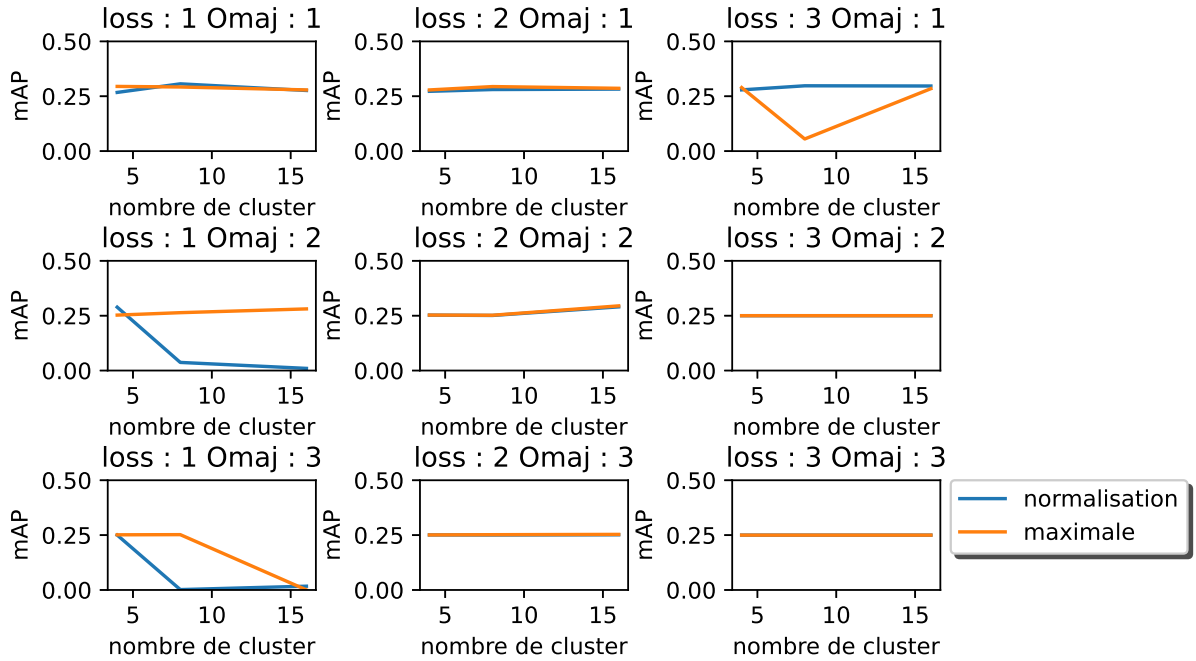

FIGURE 64 – Evolution des performances de notre architecture selon le nombre de *clusters* de la dernière couche, la fonction de distance adoptée, le calcul de la fonction de coût (*loss*), ainsi que la détermination de l'objet maximal (*Omaj*).

Choix de *l'objet majoritaire*

Nous évaluons ici les différentes approches de définition de *l'objet majoritaire*, c'est-à-dire l'objet que l'on cherche à détecter pour chaque *cluster* (voir section 3.4.3 pour plus de détails). Nous comparons alors 3 manières différentes de définir *l'objet majoritaire* de chaque *cluster* :

1. l'objet dont la distance avec le centroïde du *cluster* est la plus petite
2. l'objet qui compte les plus grand nombre de points dans le *cluster* proportionnellement au nombre de points de cet objet présent dans le bloc
3. l'objet qui compte les plus grand nombre de points dans le *cluster* proportionnellement au nombre de points total de cet objet

A partir des résultats obtenus en termes de *mAP* (voir figure 64), nous montrons que le choix 3 pour la définition de l'objet majoritaire est le moins bon parmi les 3. En effet, l'utilisation du

nombre total de points des objets implique des écarts trop importants selon les objets, ce qui entraîne des problèmes de convergence du réseau. Sans prendre en compte les autres paramètres, les choix 1 et 2 semblent au même niveau.

Équilibrage de la fonction de coût

Comme les *clusters négatifs*, c'est-à-dire ne contenant pas d'objets, sont en surnombre lors de l'apprentissage, nous devons équilibrer la fonction de coût afin qu'elle ne tienne pas compte de ceux-ci. Il y a alors plusieurs méthodes possibles afin d'équilibrer la fonction de coût, selon la définition des *clusters vides* que nous choisissons. Ainsi, la fonction de coût ne tient pas compte des *clusters* tels que :

1. la distance entre le centroïde du *cluster* et *l'objet majoritaire* est supérieure à un seuil τ_0
2. le nombre de points de *l'objet majoritaire*, proportionnellement au nombre de points de cet objet dans le bloc, est inférieur à un seuil τ_1,
3. la fonction de coût n'est pas équilibrée

Les résultats en termes de *mAP* (voir figure 64) nous montrent que le choix 3, ainsi que le 1 quand la distance par normalisation est utilisée, sont les moins bons.

Choix de la fonction de distance

Nous évaluons ici les fonctions de distance (voir section 3.2.3 pour plus de détails) proposées afin d'équilibrer les parties *caractéristiques* et géométriques des centroïdes. Nous avons proposé 2 fonctions différentes pour atteindre ce but :

- une distance par *normalisation* (courbes bleues dans la figure 64), qui consiste à normaliser les distances euclidiennes de chaque partie en les divisant par leurs variances et en leurs soustrayant leurs moyennes,
- une distance par *composante maximale* (courbes oranges dans la figure 64), qui consiste à calculer une distance euclidienne pour les parties géométriques et une soustraction entre les composantes maximales pour les parties caractéristiques

Les résultats obtenus en termes de *mAP* (voir figure 64), nous montrent que la distance par *normalisation* donne de meilleurs résultats quand l'*objet majoritaire* choisi est *le plus proche* (*Omaj* : 1), tandis que la distance par *composante maximale* donne de meilleurs résultats quand l'*objet majoritaire* choisi est celui dont la proportion est *maximale* (Omaj : 2).

En prenons en compte tous les paramètres, nous arrivons à la conclusion que le meilleur couple *fonction de coût* / *objet maximal* est le 2 / *2* avec utilisation de distance par composante maximale. Celui-ci a une avance de 5% en termes de *mAP* par rapport au second candidat qui est le couple *1* / *1* avec utilisation de distance par normalisation.

C'est en utilisant ces fonctions que nous obtenons les résultats présentés dans le tableau 4.9.

4.3.5 Bilan

Nous avons montré comment l'architecture *RPC* que nous proposons, basée sur l'opération originale de *cluster-convolution*, permet de détecter des objets urbains dans une scène 3D. Malgré la taille restreinte du jeu de données utilisé, nous obtenons des résultats encourageants en termes de *mAP*. A partir de l'analyse fine des paramètres les plus importants de l'opération de *cluster-convolution*, nous avons mis en lumière les axes d'amélioration de notre méthode qui sont : l'influence des paramètres k et λ ainsi que les objets frontières dans certains blocs.

Nous avons également comparé notre architecture avec une méthode *en deux temps* basée sur la segmentation sémantique. Il en ressort que le développement de réseaux de neurones orientés détection objet serait la solution plus adaptée afin de réaliser la tâche de détection d'objets urbains. Ce développement des réseaux détecteurs d'objets peut passer par plusieurs pistes :

- poursuivre les recherches sur notre architecture afin de dépasser les limitations que nous avons évoquées : la nécessité de découper les scènes en blocs, la fonction de distance quasi-indépendante de l'apprentissage par le réseau et le choix du nombre k de *clusters* à fixer.

- adapter aux problèmes des objets urbains les premières méthodes qui donnent de bons résultats dans le cadre de la détection d'objets en intérieur, comme [QLHG19] ou des véhicules autonomes, comme [YSL+19].

- proposer de nouvelles manières d'appréhender cette tâche en s'appuyant sur les recherches de plus en plus nombreuses en *DeepLearning 3D*.

Chapitre 5

Conclusion

5.1 Bilan de la thèse

Dans le premier chapitre nous avons identifié les difficultés de la gestion d'objets urbains et les défis que celle-ci soulève, notamment au niveau du nombre d'objets à suivre. A partir de cette analyse nous avons proposé une approche pour la gestion et le suivi des objets urbains, basée essentiellement sur des acquisitions *LiDAR MLS* ainsi que sur un traitement automatisé des nuages de points obtenus par le *LiDAR MLS*.

Dans le chapitre 2 nous avons passé en revue les différentes approches permettant de réaliser automatiquement la *détection des objets* présents dans un nuage de points 3D, en insistant sur les approches du domaine de l'apprentissage profond. Parmi celles-ci, nous avons identifié les méthodes basées sur le réseau *PointNet* [QSMG17] comme étant les plus prometteuses. Néanmoins, comme ces méthodes ne sont pas complètement adaptées à la tâche de détection d'objets et à cause du manque de travaux dans l'état de l'art sur les objets urbains, nous avons proposé une méthode originale pour traiter ces problèmes.

Dans le chapitre 3 nous détaillons les caractéristiques de notre architecture de réseau de neurones. Celle-ci s'insère dans la famille des méthodes basées sur *PointNet* et s'inspire des *réseaux avec propositions de régions (RPN)* [Gir15]. Notre architecture s'articule autour d'une opération clé appelée *cluster-convolution* composée de deux couches :

- La couche de *clustering* qui permet de regrouper les points dans des *clusters*. Le regroupement est calculé selon une distance combinant la position géométrique des points et leurs *caractéristiques* calculées par le réseau.

- La couche de *graph convolution* qui permet d'extraire des *caractéristiques* à partir de chacun des *clusters*, en fonction de la répartition respective de ces derniers.

Cette opération est appliquée de manière hiérarchique aux nuages et permet de détecter les objets présents dans le nuage de points à partir des derniers *clusters* calculés par le réseau.

Dans le chapitre 4 nous décrivons dans un premier temps les expériences que nous avons menées sur la classification d'objets urbains à partir de nuage de points. Nous montrons alors que

les *caractéristiques* extraites par le réseau *PointNet* sont suffisantes pour apprendre à reconnaître des objets urbains à l'aide d'un classifieur. Nous avons pu aussi tirer d'autres résultats de ces expériences tels que :

- Enrichir la base d'entraînement par des nuages de points provenant d'acquisitions variées permet d'améliorer significativement les performances de la classification. Les différences entre les acquisitions portent sur : la ville d'acquisition, le matériel utilisé, les catégories d'objets annotés, etc...
- Le réseau a du mal à correctement identifier les objets dans le cas des catégories d'objets géométriquement semblables, comme par exemple les feux de signalisation et les lampadaires.
- Le nombre de points par nuage n'a pas énormément d'influence sur les performances de classification tant que celui-ci est supérieur à un certain seuil, que nous évaluons expérimentalement à 32 (avec utilisation d'un filtre volumique).

Dans un second temps, nous décrivons les expériences que nous avons menées sur la détection d'objets urbains. Nous y montrons qu'en suivant une méthodologie de segmentation du nuage en blocs réguliers, il est possible d'apprendre à notre réseau à prédire des boîtes englobantes autour des objets urbains présents. Nous mettons en évidence les paramètres principaux sur lesquels reposent les performances de notre méthode :

1. le découpage préalable des scènes en blocs et le problème des objets frontières,
2. le choix du nombre de *clusters* de la dernière couche,
3. le choix de la fonction de distance adoptée dans la couche de *clustering*.

5.2 Résultats

Dans le premier chapitre du manuscrit, nous avons présenté les objectifs que nous nous sommes fixés. Pour la détection d'objets urbains, nous avons obtenus les résultats suivants :

1. nous avons montré, à partir de la réalisation d'expériences de classification de nuage de points, que le choix des approches d'apprentissage profond basées sur les points est adapté à la détection d'objets urbains.
2. le développement de l'opération de *cluster-convolution* nous a permis de proposer une architecture de réseau adaptée au problème de la détection d'objets urbains dans des grands nuages de points.
3. nous avons illustré la difficulté d'adaptation des méthodes existantes à notre cas d'application à travers la comparaison de notre architecture à des architectures réalisant la même tâche.

Discussion autour des approches récentes

Pendant la période où les travaux présentés dans ce manuscrit ont été réalisés, d'autres méthodes de détection d'objets basées sur *PointNet* [QSMG17] ont été proposées. Ces méthodes récentes font l'objet d'analyse dans la section 2.6.2, aussi nous détaillons ici les liens et différences entre ces méthodes et la nôtre.

Comparaison expérimentale

Les résultats que nous avons obtenus sur la tâche de détection d'objets urbain, environ 0.4 en termes de *mAP*, sont en dessous des résultats obtenus dans d'autres cas d'application telle que la détection d'objets en intérieur ou bien la détection pour véhicule autonome, qui donnent des résultats respectivement autour de 0.5 et 0.8 en termes de *mAP*.

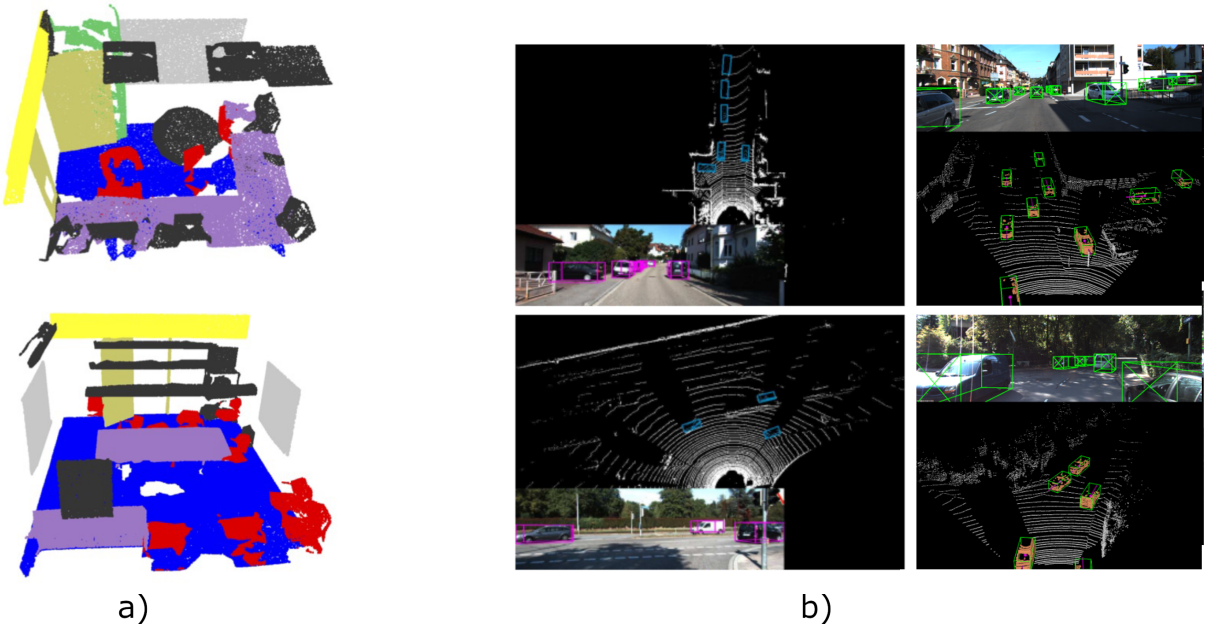

FIGURE 65 – Exemple de nuages de points provenant d'autres cas d'applications. En (*a*), des scènes 3D issues de la base *S3DIS* [ASZ+16] et coloriées selon la classe des objets auxquels appartiennent les points. En (*b*), des scènes issues de la base *KiTTi* [GLU12]. Figures tirées de [QSMG17], [CMW+17] et [SWL19].

Cet écart s'explique principalement par la spécificité des nuages de points utilisés dans chaque cas d'application.

(**a**) Dans le cas des scènes en intérieur, comme par exemple les jeux de données *S3DIS* [ASZ+16] et *SUN RGB-D* [SLX15], la taille des nuages de points dépend des dimensions des salles scannées, qui sont souvent inférieures à 5×5 mètres. De plus, il y a une régularité dans la distribution des points dans chaque salle. Par exemple chaque salle est toujours délimitée par au moins 4 murs placées aux extrémités, un plafond en haut et un sol en bas. Il existe ainsi des emplacements particuliers où sont situés les objets, par

exemple les tableaux (gris) sont tous adossés à un mur et les poutres (jaune) sont placées dans un coin supérieur (voir figure 65 (*a*)).

(**b**) Dans le cas des voitures autonomes, comme par exemple dans le jeu de données *KiTTi* [GLU12], le dispositif *LiDAR* monté horizontalement (et non verticalement comme dans la plupart des cas) a plusieurs conséquences sur les nuages de points obtenus (voir figure 65 (*b*)) :

- la taille des scènes est réduite à quelques dizaines de mètres seulement
- les points situés à plus de 2 mètres de hauteur sont quasiment tous absents
- à part les objets les plus proches du *LiDAR*, tous les autres comptent seulement quelques dizaines de points

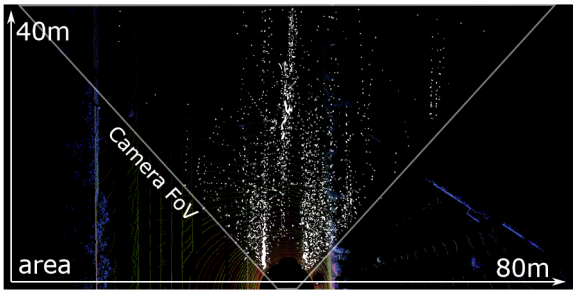

FIGURE 66 – Distribution des objets dans les nuages de points issus de la base *KiTTi* [GLU12]. Chaque point blanc représente un objet. Figure tirée de [SMAG19].

De plus, il semble qu'il existe là aussi une certaine régularité sur la distribution des points dans les scènes 3D. En effet, [SMAG19] note que tous les objets sont présents dans la même région du nuage de points correspondant au *champ de vision* (*field of view* ou *FOV*) du *LiDAR* (voir figure 66). Nous notons aussi que les objets ne sont pas répartis équitablement à l'intérieur même du *champ de vision*. Ainsi, nous distinguons une sur-représentativité des objets (représentée par des 'lignes' dans la figure 66) directement à gauche et à droite du *LiDAR*, qui correspond sans doute aux voitures garées en ville, ainsi qu'aux véhicules roulant devant le *LiDAR*.

L'écart de performance que nous obtenons pour la détection d'objets urbains peut alors s'expliquer d'une part par les contraintes associées aux nuages de points sur lesquels nous travaillons : des nuages de points de grande taille (environ 200 mètres en linéaire) et sans régularité spatiale en termes de distribution des objets à cause du découpage en blocs.

D'autre part, l'écart de performance peut aussi s'expliquer par le faible volume de données à notre disposition par rapport aux autres applications. En effet les jeux de données *SunRGB-D* [SLX15] et *KiTTi* [GLU12] comptent plusieurs dizaines de milliers d'objets, alors que la base que nous avons utilisée en compte quelques centaines tout au plus. Nous pouvons alors nous attendre à une hausse conséquente des performances de notre architecture au fur et à mesure

que le nombre de données augmentera, tel que nous l'avons montré dans les expériences de classification d'objets.

Comparaison méthodologique

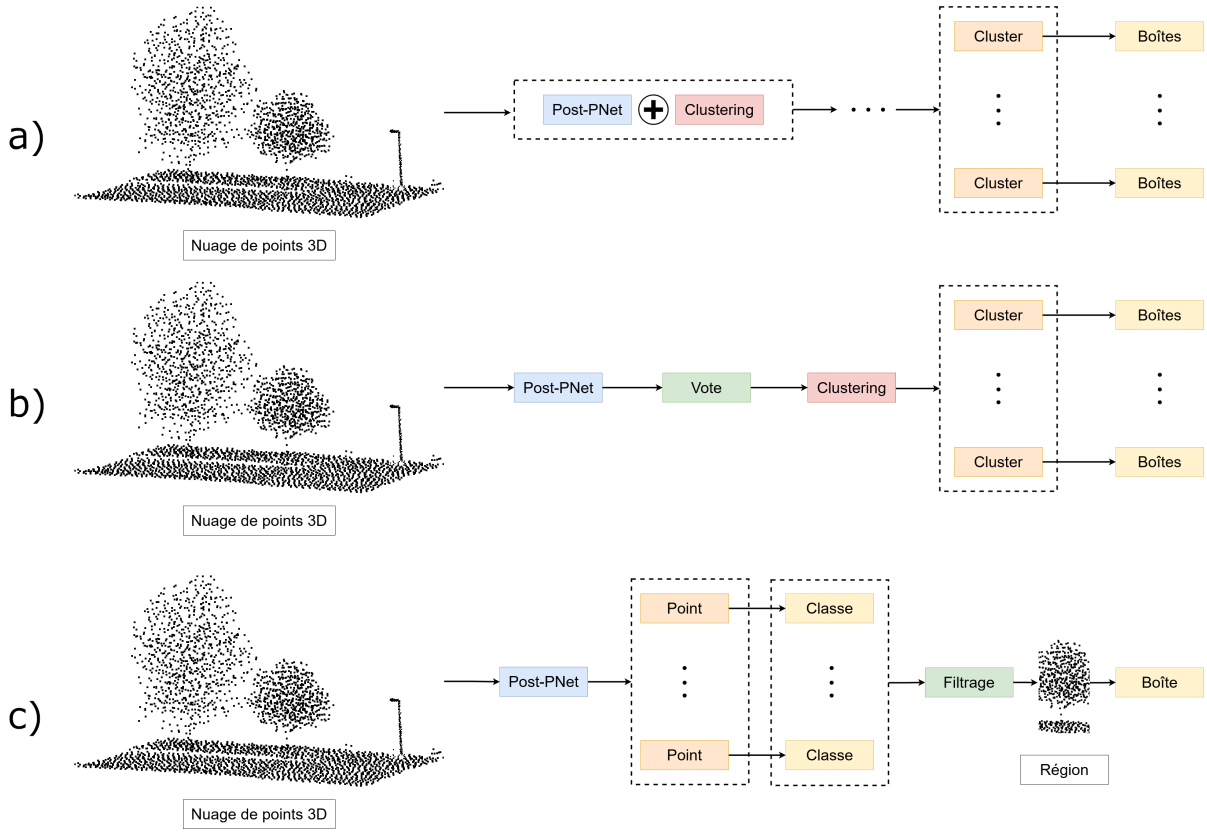

FIGURE 67 – Schéma comparatif de la méthodologie avec 3 types de méthodes de détection d'objet : (*a*) notre architecture par *cluster*, (*b*) les méthodes basées sur *VoteNet* [QLHG19] et (*c*) les méthodes basées sur *PointRCNN* [SWL19].

Nous comparons notre architecture avec 2 autres réseaux de détection d'objets inspirés de *PointNet* : le réseau *PointRCNN* [SWL19] appliqué au jeu de données *KiTTi* [GLU12] et le réseau *VoteNet* [QLHG19] appliqué au jeu de données d'intérieur *SunRGB-D* [SLX15]. Nous notons que ces 3 architectures s'appuient sur les mêmes opérations modulaires que sont :

- l'extraction pour chaque point de caractéristiques ponctuelles par une approche héritée de *PointNet* (plus de détails en section 2.5.2), notée *PostPNet* dans la figure 67, qui est implémentée par notre opération de *cluster-convolution* dans notre cas et par *PointNet++* [QYSG17] dans le cas des 2 autres approches

- le *clustering* des points, qui peut être soit accompli par une réelle méthode de *clustering* comme dans notre cas ou bien simplement un voisinage sphérique à partir de points tirés au sort, représenté par la brique *Clustering* dans la figure 67

– la prédiction de boîte englobantes par *couches entièrement connectées* à partir d'un vecteur de *caractéristiques*, représenté par la brique *Boîtes* dans la figure 67. Les prédictions sont calculées soit sur chaque *cluster* dans le cas de notre architecture (voir figure 67 (*a*)) et de *VoteNet* [QLHG19] (voir figure 67 (*b*)) soit sur chaque point filtré dans le cas des méthodes basées sur *PointRCNN* [SWL19] (voir figure 67 (*c*))

Alors que notre architecture (voir figure 67 (*a*)) repose sur une combinaison d'opérations *PostPNet* et *Clustering* appliquées de manière hiérarchique, *VoteNet* (voir figure 67 (*b*)) propose de d'abord appliquer les opérations *PostPNet*, puis l'opération originale de *Vote* et enfin le *Clustering*. L'opération de *Vote* joue un rôle crucial, car sans elle le réseau correspondrait à une approche naïve de segmentation sémantique à cela près qu'une boîte englobante et non une classe est prédite pour chaque point. L'opération de *Vote* consiste à prédire pour chaque point le centre de l'objet le plus proche. Elle permet ainsi de rapprocher les points situés sur un même objet, ce qui la rapproche ainsi du rôle joué par notre fonction de distance appliquée aux caractéristiques. En effet, deux points qui ont des valeurs de *Vote* proches auront plus de chance de se trouver dans le même voisinage et dans notre cas deux points avec des caractéristiques similaires auront plus de chance de se trouver dans le même *cluster*.

Les méthodes basées sur *PointRCNN* (voir figure 67 (*c*)) reviennent alors à suivre la méthode naïve décrite ci-dessus mais en effectuant au préalable un filtrage des points qui n'appartiennent à aucun objet grâce à un sous réseau entraîné sur la tâche de segmentation sémantique. Cette approche a alors pour avantage de se concentrer uniquement sur les points susceptibles d'aboutir à une prédiction de boîte englobante valide, car proche d'un objet.

5.3 Bilan industriel

Durant ma thèse, j'ai pu profiter du cadre industriel de celle-ci afin d'échanger avec des professionnels de la gestion d'objets urbains tels que l'équipe de développement de *ATAL II*, qui est l'application de *Berger-Levrault* destinée aux gestionnaires de villes. Ces échanges ont permis de détailler le potentiel applicatif de nos travaux. En effet, l'inventaire et la gestion des objets urbains étant une des fonctionnalités de l'outil *ATAL II* (voir chapitre 1), l'automatisation partielle de cette tâche grâce à l'utilisation de notre méthode s'avère très intéressante. Nous avons identifié 2 cas d'utilisation que pourrait offrir notre méthode : la visualisation augmentée de scènes urbaines avec informations détaillées des objets urbains détectés et la mise à jour des informations contenues dans la base de données des gestionnaires de ville à partir des informations recueillies par notre méthode.

Nous avons développé un démonstrateur pour le premier cas d'utilisation. Celui-ci est basé sur les technologies *Unity* et *PCX* [1]. Une capture d'écran de ce démonstrateur est présentée en figure 68. Le démonstrateur permet d'importer et de visualiser des scènes *LiDAR* sous formes

1. https://github.com/keijiro/Pcx

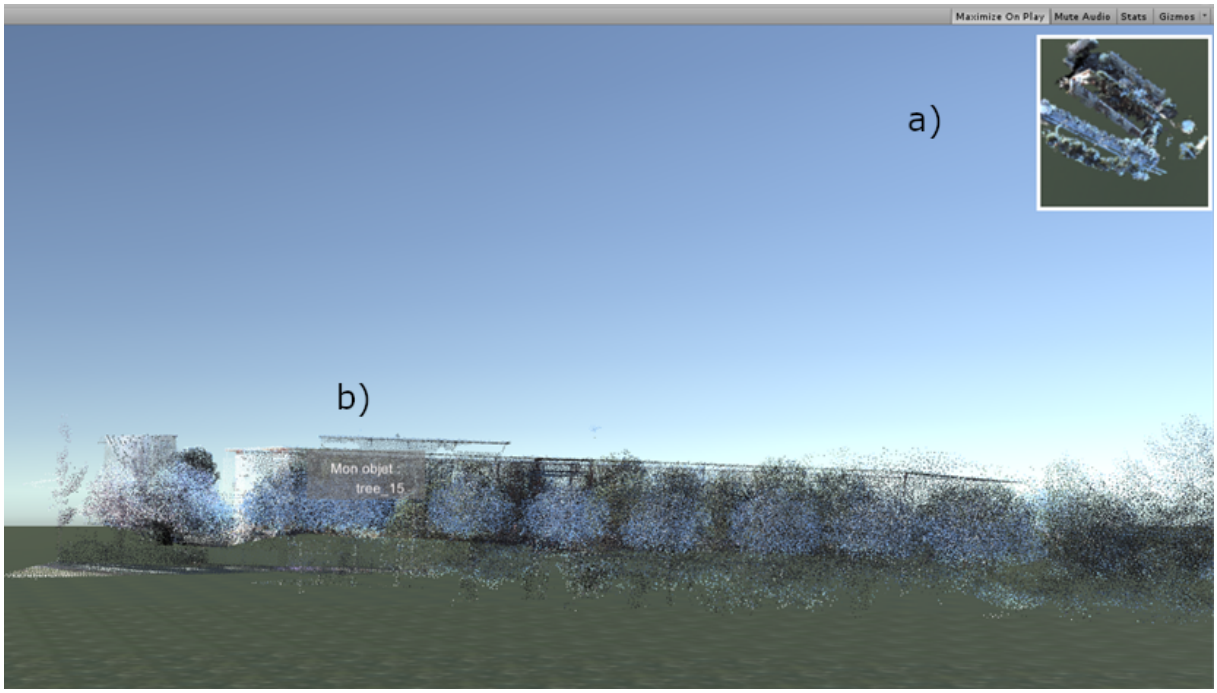

FIGURE 68 – Capture d'écran du démonstrateur développé avec : en (*a*) une vue aérienne de la scène globale et un indicateur de la position de l'utilisateur dans celle-ci et en (*b*) la vue à la première personne de la scène par rapport à la position de l'utilisateur. La fenêtre de dialogue correspond aux informations que l'utilisateur peut obtenir en cliquant sur un objet.

de nuages de points 3D. La scène 3D affichée est alors interactive : l'utilisateur peut se déplacer à l'aide des touches directionnelles, tout en observant la scène dans son entièreté sur une fenêtre secondaire (*mini-map*). Les boîtes englobantes prédites par le réseau sont aussi importées dans la scène : il est alors possible d'obtenir des informations complémentaires, comme la classe, sur chaque objet urbain présent dans la scène en cliquant dessus, grâce à l'utilisation du *raycasting*. A terme les informations affichées auraient pour finalité de refléter les informations disponibles dans les bases de données de suivi des objets urbains en communiquant avec l'application *ATAL II*.

Chapitre 6

Perspectives

6.1 Apprentissage des *clusters*

Rappelons que notre architecture consiste à décomposer le nuage de points en entrée en plusieurs *clusters* à partir desquels le réseau prédit ensuite des boîtes englobantes. Les paramètres de l'opération de *clustering*, basé sur l'algorithme du *k-means*, tels que le nombre de *clusters* et la fonction de distance utilisée ne sont pas appris par le réseau. L'influence de l'apprentissage du réseau sur la répartition des *clusters* ne se fait alors que de manière indirecte (voir figure 69).

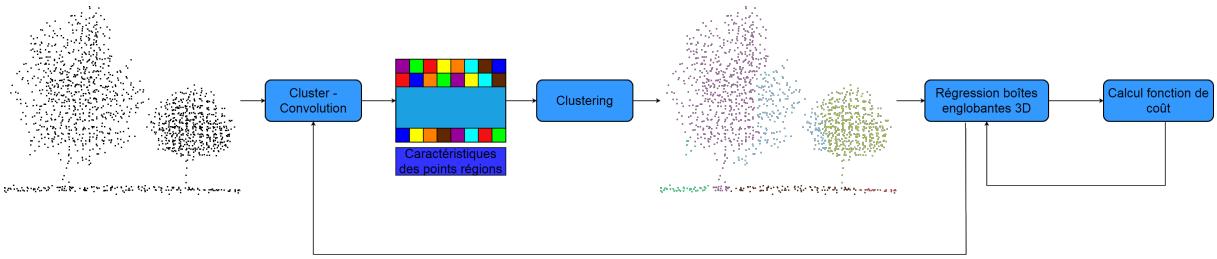

FIGURE 69 – Récapitulatif de la fonction d'apprentissage de notre méthode : les paramètres de la couche de *clustering* ne sont pas appris par le réseau. Ainsi la fonction de coût ne permet pas d'optimiser la répartition des *clusters* mais simplement d'essayer d'obtenir les meilleures boîtes englobantes possibles quelque soit la répartition obtenue par la couche de *clustering*.

Ainsi les principales limites de notre approche sont liées au choix expérimental des paramètres de la couche de *clustering*. En effet comme nous le montrons dans les expériences détaillées en section 4.3.3, le réseau ne garantit pas de corriger les mauvaises répartitions de *clusters* et celles-ci entraînent souvent des erreurs de prédictions de boîtes englobantes. Une solution à ce problème serait de faire apprendre au réseau à obtenir les répartitions de *clusters* les mieux adaptées à la prédiction de boîtes englobantes. Nous détaillons ci-dessous les premières approches que nous avons expérimentées ainsi que des pistes éventuelles pour aller plus loin.

6.1.1 Premières tentatives d'apprentissage automatique des *clusters*

Une fonction de coût directement impactée par les *clusters*

Nous avons expérimenté une approche qui consiste à modifier la fonction de coût de notre réseau afin que celle-ci soit directement liée à la répartition des *clusters* (voir figure 70). Pour rappel la fonction de coût de notre réseau est définie comme ceci :

$$\mathscr{L}(\{\mathbf{V}\}_1^{N_l}, S^*) = \frac{\lambda_{cls}}{N_l} \sum_i^{N_l} \left[\mathscr{L}_{cls}\left(\mathbf{V}_i, \mathbf{B}^*_{i,O_{maj}}\right) + \frac{\lambda_{box}}{N_l} \mathscr{L}_{box}\left(\mathbf{V}_i, \mathbf{B}^*_{i,O_{maj}}\right) \right] \tag{6.1}$$

où O_{maj} est l'objet majoritaire, c'est-à-dire l'objet qui est le plus présent dans le *cluster*, et les $\mathbf{V}_i$ sont les vecteurs de prédictions calculés à partir des derniers *clusters*. Le premier terme sert à superviser la prédiction des classes des objets détectés, le second terme sert à superviser les dimensions et la position de chaque boîte englobante, voir section 3.4.3 pour plus de détails. L'idée est de lier directement les distributions de chaque *cluster* à cette fonction de coût afin d'obtenir des *caractéristiques* plus compatibles avec l'algorithme du *k-means*. Contrairement à l'approche par *clustering semi-adaptatif* (voir figure 72), qui reste à être expérimenté dans notre cas, la couche de *clustering* demeure inchangée par rapport à notre architecture de base. Nous ajoutons alors un terme dans la fonction de coût, qui est spécifique à la répartition des *clusters* :

$$\frac{\lambda_{prop}}{N_l} \sum_i^{N_l} \mathscr{L}_{Prop}(\boldsymbol{\mu}_i) \tag{6.2}$$

où $\boldsymbol{\mu}_i$ est un centroïde et $\mathscr{L}_{Prop}$ une fonction que nous définissons comme ceci :

$$\mathscr{L}_{Prop} : \left| \begin{array}{l} \mathbb{R}^d \rightarrow \mathbb{R}^+ \\ \boldsymbol{\mu} \rightarrow \mathscr{L}_{Prop}(\boldsymbol{\mu}) \end{array} \right. \tag{6.3}$$

et l'application de la fonction $\mathscr{L}_{Prop}$ sur le centroïde $\boldsymbol{\mu}$ est définie en fonction de l'*objet majoritaire* du *cluster* associé :

$$\mathscr{L}_{Prop}(\boldsymbol{\mu}) = \sinh\left(\frac{Card\left(O_{maj}\right)}{Card\left(\mathscr{K}(\boldsymbol{\mu})\right)}\right) \tag{6.4}$$

où $\mathscr{K}(\boldsymbol{\mu})$ est la région du nuage de points initial S_0 associé au *centroïde* $\boldsymbol{\mu}$.

Nous suivons alors le même protocole d'entraînement que celui détaillé dans la section 4.3.1 en utilisant cette nouvelle fonction de coût. Nos premiers résultats montrent que le réseau n'arrive pas à converger. L'utilisation d'une fonction de sinus hyperbolique ne parvient pas à

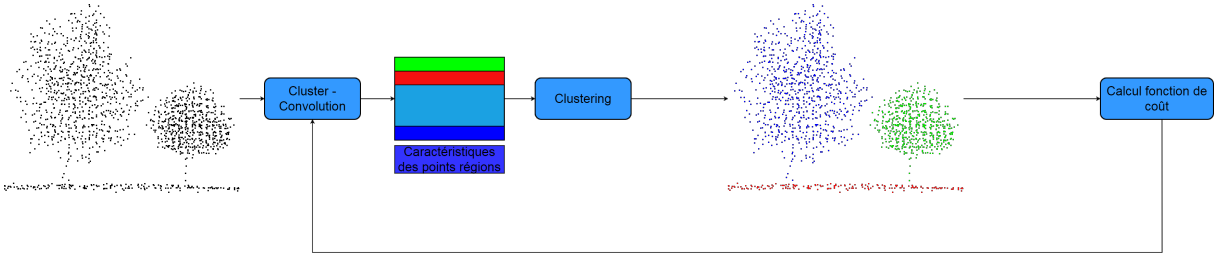

FIGURE 70 – L'approche que nous proposons avec une nouvelle fonction de coût qui cherche à optimiser la répartition des *clusters* tout en conservant la même couche de *clustering*. Le but est d'obtenir des caractéristiques qui facilitent une répartition optimale des *clusters* (un objet par *cluster*).

résoudre ce problème de convergence.

Nous avons identifié comme raison principale de cette absence de convergence la fluctuation permanente de la répartition des *clusters* d'une scène à l'autre. En effet, comme nous l'avons montré dans la section 4.3.3, la répartition des *clusters* dépend de la distribution des objets dans le bloc. Or celle-ci varie beaucoup selon les blocs. Ainsi, les variations de distribution d'objets dans le bloc affectent directement le calcul de la fonction de coût, empêchant le réseau de calculer des *caractéristiques* pertinentes pour la détection des objets.

Apprentissage en deux temps

Dans le but de minimiser l'impact des fluctuations de la répartition des *clusters* sur le réseau, nous avons développé une approche d'apprentissage en deux temps. L'idée est de réaliser l'entraînement, en alternant deux étapes (voir figure 71) :

1. le réseau est entraîné normalement, en calculant des nouveaux *clusters* qui sont sauvegardés pour chaque nuage de points en entrée

2. le réseau est entraîné en utilisant les *clusters* sauvegardés les plus récents. Cette étape est répétée un certain nombre de fois avant de revenir à l'étape (**a**)

Nous relançons alors l'entraînement sur notre base de données avec la nouvelle fonction de coût (Eq 6.3) et notre stratégie d'apprentissage en deux temps. Nous remarquons que cette modification permet une convergence du réseau par à-coups. En effet, lorsque le réseau est dans l'étape 2 (voir figure 71 (*b*)), il commence à converger. Néanmoins, dès que le réseau repasse dans l'étape 1, on remarque une discontinuité au niveau du résultat de la fonction de coût. Ainsi, le réseau n'arrive pas à converger de manière globale vers une valeur assez basse.

Bilan des premières expériences

Les premières approches que nous avons expérimentées ne permettent pas d'aboutir à une méthode d'apprentissage des *clusters* par le réseau. Ainsi, nos deux approches ne permettent pas de surmonter l'obstacle de la corrélation entre les deux objectifs fixés. En effet, nous cherchons

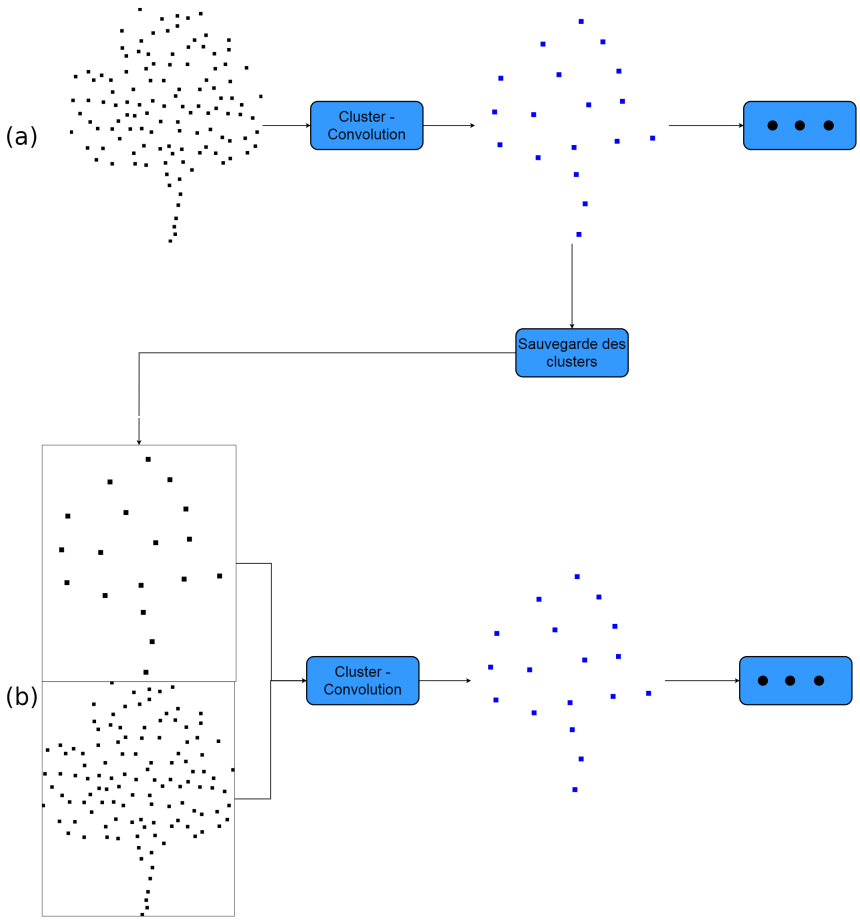

FIGURE 71 – Illustration de notre approche d'apprentissage en 2 étapes. En (*a*) l'étape d'apprentissage avec sauvegarde des *clusters* calculs et en (*b*) l'étape d'apprentissage à partir des *clusters* enregistrés.

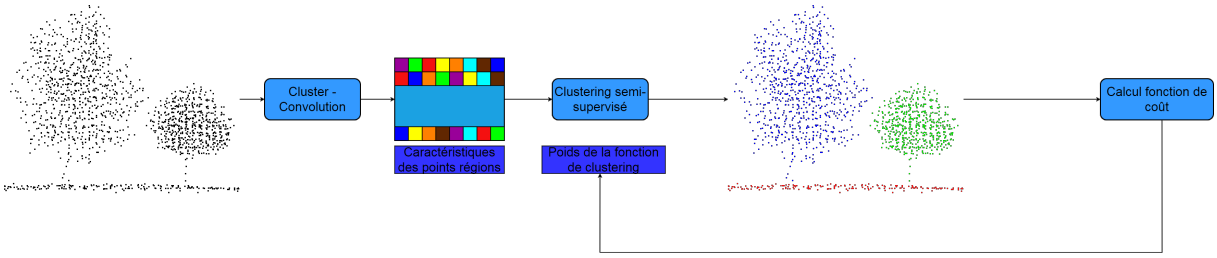

FIGURE 72 – Approche possible pour l'optimisation de la répartition des *clusters* par le réseau : remplacer la couche de *clustering* basée sur l'algorithme du *k-means* par une nouvelle couche basée sur un algorithme semi-supervisé comme par exemple le *fuzzy c-mean*. L'idée est de faire apprendre les paramètres de cette couche par le réseau grâce à une fonction de coût et un algorithme de *clustering* adaptée.

à faire apprendre au réseau à obtenir une répartition de *clusters* satisfaisante et en même temps à prédire des boîtes englobantes à partir de cette répartition de *clusters*. Or ces deux objectifs sont réalisés à partir des mêmes *caractéristiques* calculées par le réseau, aboutissant ainsi à un

résultat qui ne satisfait aucun des deux objectifs.

Nous en déduisons que pour mener à bien l'apprentissage des *clusters*, ces deux objectifs doivent être déconnectés lors de la phase d'entraînement.

6.1.2 Pistes pour aller plus loin

Afin de poursuivre les travaux sur l'apprentissage de la répartition des *clusters* par la fonction de coût du réseau de neurones, une solution pourrait être de se baser sur les méthodes de *clustering par encodage profonds ('deep embedding clustering' ou DEC* [GGLY17]).

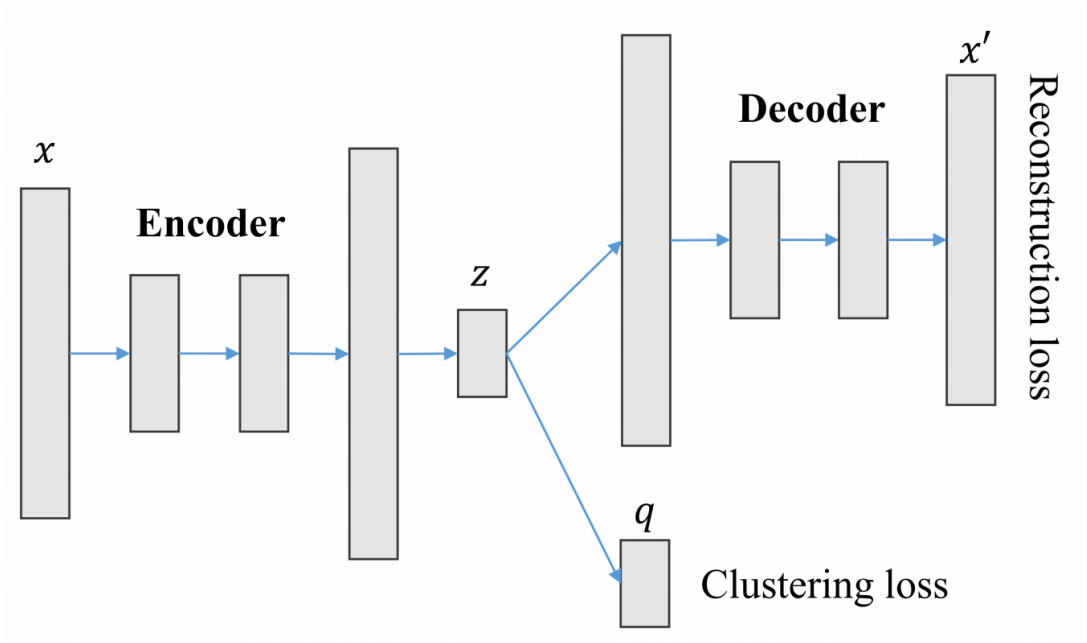

FIGURE 73 – Schéma général d'un réseau de neurones orienté *deep embedding clustering*. Figure tirée de [GGLY17].

Les méthodes de *DEC* (voir figure 73) consistent à utiliser un réseau de neurones, la plupart du temps un *auto-encodeur*, afin de projeter un ensemble de données X dans un espace de dimension supérieure $\mathscr{Z}$. Le réseau calcule ensuite un résultat X' à partir du projeté de X dans $\mathscr{Z}$. Parallèlement, un algorithme de *clustering* est appliqué dans $\mathscr{Z}$. L'entraînement du réseau est alors réalisé de manière non supervisée par 2 fonctions de coûts :

(**a**) Une fonction de *reconstruction* correspondant à la différence entre les éléments de x et les éléments de x'.

(**b**) Une fonction de la répartition des *clusters*, très souvent une divergence *KL*, qui permet de mesurer la similarité de la distribution de chaque *cluster*.

Une manière de s'inspirer de ces méthodes de *DEC* serait alors de calculer des caractéristiques spécifiques avant chaque opération de *cluster-convolution* (voir figure 74). Ces caractéristiques

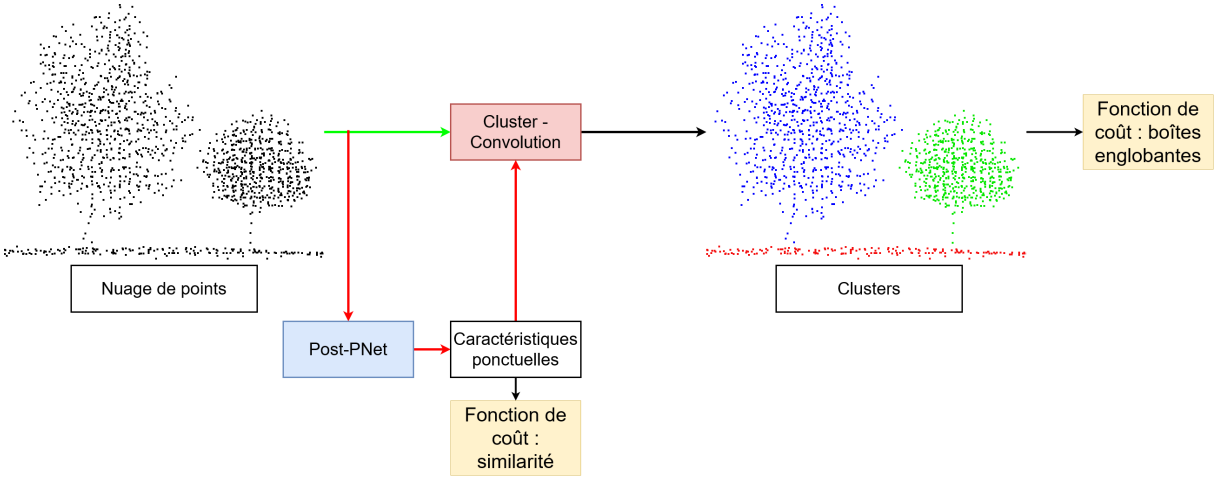

FIGURE 74 – Schéma de l'approche inspirée des réseaux *DEC* ainsi que des méthodes de segmentation d'instance de nuages de points [WYHN18]. La flèche verte représente la partie géométrique et les flèches rouges représentent la partie *caractéristique*.

seraient supervisées par une fonction de similarité comme dans le réseau *SGPN* [WYHN18]. Chaque centroïde serait alors associé à 2 parties caractéristiques **f** différentes. L'une servirait uniquement à prédire la classe et la boîte englobante, tandis que l'autre servirait uniquement au *clustering*. Ainsi, ces *caractéristiques* seraient entraînées par 2 fonctions de coût différentes :

1. notre première fonction de coût (Eq 6.1) pour superviser la prédiction des classes et des boîtes englobantes,

2. une fonction de similarité pour entraîner les *caractéristiques* spécifiques au *clustering*.

6.2 Améliorations de l'architecture neuronale

Au niveau de la prédiction de boîtes englobantes et de la classification des objets par notre réseau, un moyen d'améliorer les performances est de réduire les erreurs de prédiction, c'est-à-dire les erreurs du réseau à partir de *cluster* qui ont un bon score de *qualité*. Pour cela une solution simple pourrait être d'intégrer certaines propositions récentes de la discipline du *deep-learning 3D* afin d'affiner les *caractéristiques* calculées par le réseau. Parmi ces propositions nous pouvons citer :

– Ajouter un mécanisme d'attention pour apprendre au réseau à se concentrer sur les points les plus importants, comme par exemple le mécanisme décrit dans [ZJFJ19]. En effet, ce réseau qui se base sur les progrès réalisés par les mécanismes d'attention dans le domaine du langage naturel [BCB15], permet de mettre en lien chaque point **P** avec ses points les plus proches afin de déterminer lesquels ont le plus d'influence sur **P**. Cet ajout semble tout à fait compatible avec notre réseau puisque il se base également sur un voisinage relatif.

– Utiliser la *focal loss* [LGG+17] afin de rendre le réseau moins sensible au déséquilibre qui existe dans notre jeu d'entraînement entre les différentes classes en termes de nombres d'exemples disponibles.

6.3 Nuages de points synthétisés pour l'enrichissement des bases de données

Comme nous le montrons dans le chapitre 4, l'un des points les plus cruciaux dans une méthode de reconnaissance d'objets par apprentissage profond est la quantité de données disponibles dans la base d'entraînement. Or comme il est plus coûteux de créer une base de données de nuage de points qu'une base de données d'images, aussi bien au niveau du coût d'acquisition que du coût d'annotations Il s'agit d'une des limites majeures de l'apprentissage profond sur nuage de points.

Une manière de contourner ce problème serait d'utiliser la synthèse de nuage de points afin d'enrichir nos bases de données. En effet cela permettrait d'augmenter significativement la taille des ensembles d'entraînement tout en s'affranchissant du coût d'acquisition et d'annotation de nuages de points.

Pour les objets urbains avec les formes les plus simples, de simples modèles paramétriques pourraient suffire, comme par exemple des cylindres pour modéliser des poteaux. Dans le cas des arbres il serait possible d'utiliser un des nombreux simulateurs de modèles 3D d'arbres [1]. Pour les objets plus complexes, nous pouvons envisager d'utiliser des réseaux de neurones génératifs [CHCP19][SPK19]. Une approche plus globale consiste à combiner plusieurs méthodes semi-automatiques afin de générer directement des scènes 3D en milieu urbain [GB19][RSM+16][DRC+17].

Limites

L'approche par nuages de points synthétiques comporte 2 limites majeures qui doivent être maîtrisées afin d'être applicable.

La première concerne la représentativité des objets synthétiques. Ce critère est crucial dans le cas des objets urbains où nous remarquons une grande variabilité intra-classe, notamment pour les arbres ou les voitures. C'est notamment le cas si l'on souhaite enrichir notre jeu d'entraînement avec des objets provenant de base d'objets synthétiques comme [WSK+15].

La deuxième limite est liée à la configuration des objets urbains dans la scène 3D. Même si nous sommes en mesure de synthétiser avec un degré satisfaisant de fidélité des nuages de points représentatifs de la diversité des objets urbains, il reste à les positionner dans une scène d'une manière représentative de la réalité. En effet le réseau risque de ne pas reconnaître certaines

1. Voir par exemple : http://andrewmarsh.com/software/tree3d-web ou https://github.com/dan-c-underwood/tree-generator. Il existe également des logiciels sous licence professionnelle

configurations réelles d'objets, telles que des lampadaires entre des branches d'arbre, si celles-ci sont absentes des données d'entraînement. Les méthodes utilisées pour la synthèse des nuages de points doivent alors aussi être en mesure de simuler la complexité des interactions entre objets urbains. Par exemple lors de l'utilisation de données synthétique provenant d'acquisitions *LiDAR* simulées dans un jeu vidéo, les auteurs de [WWYK18][WZZ+19] ont relevé que le réseau a du mal à être appliqué sur des données réelles à cause des problèmes spécifiques aux acquisitions *LiDAR* en conditions réelles, comme par exemple la portée réduite du laser, les artefacts de scans, etc...

6.4 Agrandir la gamme d'objets d'urbains : ajout de capteurs

Dans les travaux que nous avons présentés, nous considérons les nuages de points comme des listes de triplets (x, y, z) uniquement. Cependant il est possible d'étendre leurs définition en ajoutant d'autres informations que la position des points, comme par exemple leurs couleurs. C'est par exemple le cas si le dispositif *LiDAR* est combiné à un appareil photo numérique afin d'obtenir comme nuage de points une liste de 6-uplets : (x, y, z, r, g, b). Néanmoins, l'information couleur n'est pas toujours utile dans le cas d'application en milieu urbain à cause des variations en termes d'exposition lumineuse, par rapport aux applications en intérieur où la luminosité peut être contrôlée. D'autre part la synchronisation entre l'appareil photo et le capteur *LiDAR* comporte toujours quelques erreurs sur certains points.

Un moyen de résoudre ce problème est d'observer également le domaine du non-visible grâce aux capteurs multi-spectraux [STMA12]. Les informations recueillies par ces capteurs additionnels peuvent ainsi permettre de mieux distinguer les objets urbains, par exemple l'utilisation de l'index *NDVI* [Tuc79] permet de faciliter l'estimation de la couverture végétale grâce à un capteur proche infrarouge. Ainsi, intégrer ces informations additionnelles aux nuages de points en entrée du réseau pourrait permettre d'extraire des caractéristiques plus discriminantes et donc faciliter la détection d'objets.

De plus, l'utilisation de capteurs additionnels peut également fournir des informations utiles et exploitables pour des applications post-détection. Parmi ces applications nous pouvons citer :

(**a**) caractériser les données physiologiques des objets urbains, par exemple analyser l'état de santé d'un arbre à partir des données multi-spectrales [AP20],

(**b**) collecter des informations supplémentaires sur la biodiversité dans la ville, comme par exemple l'évolution de la biomasse des arbres ou bien la présence d'espèces animales protégées grâce à une caméra thermique,

(**c**) détecter des objets qui n'apparaissent pas dans le nuage de points *LiDAR*, comme par exemple les objets urbains enfouis (canalisations, racines d'arbres) grâce à un *radar terrestre* (*GPR*) [TGY20]

6.5 Nuages de points 4D (3D + temps) pour l'automatisation du monitoring des objets urbains

Pour aller plus loin dans la tâche de monitoring des objets urbains, il serait pratique de prendre en compte l'évolution des objets urbains au cours du temps. Dans ce cas, au lieu d'un unique nuage de points, le réseau prendrait en entrée une série de nuage de points, correspondant chacun à une acquisition *LiDAR* réalisée au même endroit mais à une date différente des autres. Le but serait alors d'apprendre au réseau à détecter les modifications de l'environnement urbain, comme par exemple la croissance des arbres, les objets déplacés ou endommagés, ainsi que les nouveaux objets [Lid05].

Cette idée rejoint les travaux existants sur la détection d'objets dans des vidéos [BWR+20] [ZWL+20]. Ces approches consistent à réaliser la détection d'objets dans une série d'images en exploitant le lien de temporalité entre elles au lieu de simplement les traiter une par une. De plus il existe également des méthodes de détection de changement dans un flux vidéo [VGRB19]. Celles-ci consistent à détecter les modifications d'ordre sémantique, par exemple des objets manquants ou ajoutés, entre deux images. Il faudrait alors généraliser ces approches applicables uniquement à des séries temporelles structurées sous forme d'image, à des séries temporelles non structurées [dGLC21].

Une autre inspiration pourrait être les travaux de segmentation sémantique sur des nuages de points 3D+temps [CGS19]. Ici, l'aspect temporel sert uniquement à affiner les prédictions du réseau en termes de segmentation sémantique. Cependant il pourrait être possible d'entraîner ce réseau à reconnaître les modifications des objets présents dans le nuage au cours du temps.

6.6 Perspectives industrielles

L'application des travaux présentés dans ce manuscrit nécessiterait des moyens techniques et logistiques massifs, comme l'installation de capteurs *LiDAR* sur des véhicules circulant régulièrement en ville (par exemple des camions poubelles), la mise en place d'un pipeline informatique afin de stocker, traiter et archiver les nuages de points, ainsi que de transmettre les boîtes englobantes calculées par le réseau. Cependant, ce type d'application constitue aujourd'hui un des axes de recherche stratégiques de l'entreprise *Berger-Levrault*.

Actuellement, *Berger-Levrault* travaille sur la gestion d'équipements en intérieur. Nos recherches constituent alors un socle pour la mise en place d'outils à court-terme. Ces travaux s'inscrivent dans la thématique du *BIM* (*Building Information Modeling*) qui est aujourd'hui un axe de recherche prioritaire au sein de l'activité *CARL* de l'entreprise dans le but de réaliser automatiquement des inventaires d'équipements. La problématique générale étant la même que dans le cas des objets urbains, la difficulté provient de la différence significative au niveau des scènes 3D (voir section 2.7 pour plus de détails).

Une approche naïve consisterait alors à simplement appliquer les méthodes existantes à nos données, cependant elle ne serait pas adaptée à la diversité des bâtiments dans lesquels nous voulons l'appliquer, d'autant plus que nous constatons un manque de travaux existants concernant les objets relatifs aux normes de sécurité tels que les alarmes incendies ou bien les extincteurs, tous deux absents des bases de données habituelles.

Chapitre 7

Bibliographie

[ABKS99] Mihael Ankerst, Markus M. Breunig, Hans Peter Kriegel, and Jörg Sander. OPTICS : Ordering Points to Identify the Clustering Structure. In *SIGMOD Record (ACM Special Interest Group on Management of Data)*, volume 28, pages 49–60, 1999. 20

[AOH+20] Ryosuke Araki, Takeshi Onishi, Tsubasa Hirakawa, Takayoshi Yamashita, and Hironobu Fujiyoshi. Mt-dssd : Deconvolutional single shot detector using multi task learning for object detection, segmentation, and grasping detection. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10487–10493, 2020. 45

[AP20] Azadeh Abdollahnejad and Dimitrios Panagiotidis. Tree species classification and health status assessment for a mixed broadleaf-conifer forest with uas multispectral imaging. *Remote Sensing*, 12(22) :1–21, 2020. 145

[ASZ+16] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1534–1543, 2016. 115, 133

[Axe00] P. Axelsson. DEM Generation from Laser Scanner Data Using adaptive TIN Models. *International Archives of Photogrammetry and Remote Sensing*, 23(B4) :110–117, 2000. 26

[Bal81] D. H. Ballard. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2) :111–122, 1981. 22

[BCB15] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 1409, 2015. 143

[BCMDGJ14] José Luis Blanco-Claraco, Francisco Ángel Moreno-Dueñas, and Javier González-Jiménez. The Málaga urban dataset : High-rate stereo and LiDAR in a realistic urban scenario. *International Journal of Robotics Research*, 33(2) :207–214, 2014. 102

[BELN11] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nüchter. The 3D Hough Transform for plane detection in point clouds : A review and a new accumulator design. *3D Research*, 2(2) :1–13, 2011. 23

[BGLA18] Alexandre Boulch, Joris Guerry, Bertrand Le Saux, and Nicolas Audebert. SnapNet : 3D point cloud semantic labeling with 2D deep segmentation networks. *Computers and Graphics (Pergamon)*, 71 :189–198, 2018. 48

[BGM+18] Jorge Beltrán, Carlos Guindel, Francisco Miguel Moreno, Daniel Cruzado, Fernando García, and Arturo De La Escalera. BirdNet : A 3D Object Detection Framework from LiDAR Information. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018-November :3517–3523, 2018. 63

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. Training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 144–152, 1992. 23

[BJ86] Paul J. Besl and Ramesh C. Jain. Invariant surface characteristics for 3D object recognition in range images. In *Computer Vision, Graphics and Image Processing*, volume 33, pages 33–80, 1986. 21

[BJ88] Paul J. Besl and Ramesh C. Jain. Segmentation Through Variable-Order Surface Fitting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(2) :167–192, 1988. 21

[BJS19] Alessandra Budillon, Angel C. Johnsy, and Gilda Schirinzi. Contextual information based SAR tomography of urban areas. *2019 Joint Urban Remote Sensing Event, JURSE 2019*, pages 1–4, 2019. 6

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12) :2481–2495, 2017. 36

[BMS+11] M. D. Buhmann, Prem Melville, Vikas Sindhwani, Novi Quadrianto, Wray L. Buntine, Luís Torgo, Xinhua Zhang, Peter Stone, Jan Struyf, Hendrik Blockeel, Kurt Driessens, Risto Miikkulainen, Eric Wiewiora, Jan Peters, Russ Tedrake, Nicholas Roy, Jun Morimoto, Peter A. Flach, and Johannes Fürnkranz. Random

Decision Forests. In *Encyclopedia of Machine Learning*, pages 827–827, 2011. 23

[Bou20] Alexandre Boulch. ConvPoint : Continuous convolutions for point cloud processing. *Computers and Graphics (Pergamon)*, 88 :24–34, 2020. 6, 60, 61, 83

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1) :5–32, 2001. 25

[BSCD17] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-NMS - Improving Object Detection with One Line of Code. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-October, pages 5562–5570, 2017. 125

[BWR+20] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. Context R-CNN : Long term temporal context for per-camera object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 13072–13082, 2020. 146

[CC09] Andrea Censi and Stefano Carpin. HSM3D : Feature-less global 6DOF scan-matching in the hough/radon domain. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3899–3906, 2009. 23

[CDB+20] Wei Chen, Jinming Duan, Hector Basevi, Hyung Jin Chang, and Ales Leonardis. PointPoseNet : Point pose network for robust 6D object pose estimation. *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*, pages 2813–2822, 2020. 53

[CENP+17] B. Commandre, D. En-Nejjary, L. Pibre, M. Chaumont, C. Delenne, and N. Chahinian. Manhole cover localization in aerial images with a deep learning approach. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, volume 42, pages 333–338, Hannover, Germany, 2017. 6

[CFG+15] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet : An Information-Rich 3D Model Repository. *ArXiv*, abs/1512.0, 2015. 45

[CFL+15] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollar, and C. Lawrence Zitnick. Microsoft COCO Captions : Data Collection and Evaluation Server. *ArXiv*, abs/1504.0, 2015. 44

[CGS19] Christopher Choy, Junyoung Gwak, and Silvio Savarese. 4D spatio-temporal convnets : Minkowski convolutional neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :3070–3079, 2019. 146

[CHCP19] Lucas Caccia, Herke Van Hoof, Aaron Courville, and Joelle Pineau. Deep Generative Modeling of LiDAR Data. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5034–5040, 2019. 144

[Chi92] Nancy Chinchor. MUC-4 evaluation metrics. In *Proceedings of the 4th Conference on Message Understanding*, MUC4 '92, page 22, USA, 1992. Association for Computational Linguistics. 109

[CJO19] Erzhuo Che, Jaehoon Jung, and Michael J. Olsen. Object recognition, segmentation, and classification of mobile laser scanning point clouds : A state of the art review. *Sensors (Switzerland)*, 19(4), 2019. 9, 10, 26

[CMW+17] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3D object detection network for autonomous driving. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January :6526–6534, 2017. 63, 133

[CMZS15] Ricardo J.G.B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *ACM Transactions on Knowledge Discovery from Data*, 10(1) :5 :1–5 :51, 2015. 21

[CRR19] Pedro Hermosilla Casajus, Tobias Ritschel, and Timo Ropinski. Total denoising : Unsupervised learning of 3D point cloud cleaning. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October :52–60, 2019. 53

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3) :273–297, 1995. 25

[CVG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1724–1734, 2014. 58

[DBI18] Haowen Deng, Tolga Birdal, and Slobodan Ilic. PPFNet : Global Context Aware Local Features for Robust 3D Point Matching. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 195–205, 2018. 53

[DCK19] Chaojing Duan, Siheng Chen, and Jelena Kovacevic. 3D Point Cloud Denoising via Deep Neural Network Based Local Surface Estimation. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2019-May :8553–8557, 2019. 53

[DCS+17] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet : Richly-annotated 3D reconstructions of indoor scenes. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January :2432–2443, 2017. 119

[Des10] Jean-emmanuel Deschaud. A Fast and Accurate Plane Detection Algorithm for Large Noisy Point Clouds Using Filtered Normals and Voxel Growing. In *Symposium A Quarterly Journal In Modern Foreign Literatures*, 2010. 21

[dGLC21] Iris de Gélis, Sébastien Lefèvre, and Thomas Corpetti. Change Detection in Urban Point Clouds : An Experimental Comparison with Simulated 3D Datasets. *Remote Sensing*, 13(13) :2629, July 2021. 146

[DH72] Richard O. Duda and Peter E. Hart. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Communications of the ACM*, 15(1) :11–15, 1972. 22

[DRC+17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA : An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 144

[EEV+15] Mark Everingham, S. M.Ali Eslami, Luc Van Gool, Christopher K.I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes Challenge : A Retrospective. *International Journal of Computer Vision*, 111(1) :98–136, 2015. 118, 119

[EHL11] Sherif Ibrahim El-Halawany and Derek D. Lichti. Detection of road poles from mobile terrestrial laser scanner point cloud. *2011 International Workshop on Multi-Platform/Multi-Sensor Remote Sensing and Mapping, M2RSM 2011*, pages 1–6, 2011. 28, 29

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996. 20

[ERW+17] Martin Engelcke, Dushyant Rao, Dominic Zeng Wang, Chi Hay Tong, and Ingmar Posner. Vote3Deep : Fast object detection in 3D point clouds using efficient

convolutional neural networks. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1355–1361, 2017. 63

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus : A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6) :381–395, 1981. 23

[FLCP+20] Clara Fernandez-Labrador, Ajad Chhatkuli, Danda Pani Paudel, Jose J. Guerrero, Cédric Demonceaux, and Luc Van Gool. Unsupervised Learning of Category-Specific Symmetric 3D Keypoints from Point Sets. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12370 LNCS :546–563, 2020. 53

[FPP07] David Freedman, Robert Pisani, and Roger Purves. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007. 121

[GB19] David Griffiths and Jan Boehm. SynthCity : A large scale synthetic point cloud. In *ArXiv preprint*, 2019. 144

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. 38

[GGLY17] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 0, pages 1753–1759, 2017. 8, 142

[Gir15] Ross B Girshick. Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 86, 131

[GLSU13] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics : The KITTI dataset. *International Journal of Robotics Research*, 32(11) :1231–1237, 2013. 89

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving ? the KITTI vision benchmark suite. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. 68, 118, 133, 134, 135

[GMK99] C. Galambos, J. Matas, and J. Kittler. Progressive Probabilistic Hough Transform for line detection. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 554–560, 1999. 23

[Gor02] Ben Gorte. Segmentation of Tin-structured Surface Models. In *Symposium A Quarterly Journal In Modern Foreign Literatures*, volume 34, pages 465–469, 2002. 21

[GSB+13] Yulan Guo, Ferdous Sohel, Mohammed Bennamoun, Min Lu, and Jianwei Wan. Rotational projection statistics for 3D local surface description and object recognition. *International Journal of Computer Vision*, 105(1) :63–86, 2013. 24

[GWH+20] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep Learning for 3D Point Clouds : A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP :1–1, 2020. 100

[GWW+19] Xiuye Gu, Yijie Wang, Chongruo Wu, Yong Jae Lee, and Panqu Wang. HPLFLOWNET : Hierarchical permutohedral lattice flownet for scene flow estimation on large-scale point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :3249–3258, 2019. 53

[GYLL16] Haiyan Guan, Yongtao Yu, Jonathan Li, and Pengfei Liu. Pole-Like Road Object Detection in Mobile LiDAR Data via Supervoxel and Bag-of-Contextual-Visual-Words Representation. *IEEE Geoscience and Remote Sensing Letters*, 13(4) :520–524, 2016. 28

[HLVW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January :2261–2269, 2017. 36

[Ho95] Tin Kam Ho. Random decision forests. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, volume 1, pages 278–282, 1995. 25

[Hou60] Paul Hough. Method and Means for Recognizing Complex Patterns, 1960. 22, 64

[HSA18] Marvin Hubl, Philipp Skowron, and Michael Aleithe. Towards a Supportive City with Smart Urban Objects in the Internet of Things : The Case of Adaptive Park Bench and Adaptive Light. In *Position Papers of the 2018 Federated Conference on Computer Science and Information Systems*, volume 16, pages 51–58, 2018. 6

[HSL+17] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys. Semantic3D.Net : a New Large-Scale Point Cloud Classification Benchmark. In

ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, volume 4, pages 91–98, 2017. 68, 102

[HSSX18] Xian-Feng Han, Shi-Jie Sun, Xiang-Yu Song, and Guo-Qiang Xiao. 3D Point Cloud Descriptors in Hand-crafted and Deep Learning Age : State-of-the-Art. *arXiv*, abs/1802.0, 2018. 24

[HTY18] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. Pointwise convolutional neural networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 984–993, 2018. 83

[Hub64] Peter J. Huber. Robust Estimation of a Location Parameter. In *The Annals of Mathematical Statistics*, volume 35, pages 73–101, 1964. 97

[HV17] Arshad Husain and R. C. Vaishya. A time efficient algorithm for ground point filtering from mobile LiDAR data. *ICCCCM 2016 - 2nd IEEE International Conference on Control Computing Communication and Materials*, pages 1–5, 2017. 26

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-December :770–778, 2016. 36

[IK88] J. Illingworth and J. Kittler. A survey of the hough transform. *Computer Vision, Graphics and Image Processing*, 44(1) :87–116, 1988. 23

[Jai10] Anil K. Jain. Data clustering : 50 years beyond K-means. *Pattern Recognition Letters*, 31(8) :651–666, 2010. 74

[JH99] Andrew E. Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5) :433–449, 1999. 24, 25

[JHR11] Andreas Jochem, Bernhard Höfle, and Martin Rutzinger. Extraction of vertical walls from mobile laser scanning data for solar potential assessment. *Remote Sensing*, 3(4) :650–667, 2011. 27, 28

[KB15] Diederik P. Kingma and Jimmy Lei Ba. Adam : A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, abs/1412.6, 2015. 33

[KEB91] N. Kiryati, Y. Eldar, and A. M. Bruckstein. A probabilistic Hough transform. *Pattern Recognition*, 24(4) :303–316, 1991. 23

[KEL60] HENRY J. KELLEY. Gradient Theory of Optimal Flight Paths. *ARS Journal*, 30(10) :947–954, 1960. 33

[KHXO95] Heikki Kälviäinen, Petri Hirvonen, Lei Xu, and Erkki Oja. Probabilistic and non-probabilistic Hough transforms : overview and comparisons. *Image and Vision Computing*, 13(4) :239–252, 1995. 23

[KK10] John F. Kolen and Stefan C. Kremer. Gradient Flow in Recurrent Nets : The Difficulty of Learning LongTerm Dependencies. In *A Field Guide to Dynamical Recurrent Networks*, 2010. 35

[KL17] Roman Klokov and Victor Lempitsky. Escape from Cells : Deep Kd-Networks for the Recognition of 3D Point Cloud Models. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October :863–872, 2017. 49

[KML+18] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L. Waslander. Joint 3D Proposal Generation and Object Detection from View Aggregation. *IEEE International Conference on Intelligent Robots and Systems*, pages 5750–5757, 2018. 63

[KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In *Communications of the ACM*, volume 60, pages 84–90, 2017. 35, 36, 118

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng Yang Fu, and Alexander C. Berg. SSD : Single shot multibox detector. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9905 LNCS, pages 21–37, 2016. 45, 62, 125

[LB19] Loic Landrieu and Mohamed Boussaha. Point cloud oversegmentation with graph-structured deep metric learning. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :7432–7441, 2019. 58

[LBS+18] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN : Convolution on -transformed points. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 828838, Red Hook, NY, USA, 2018. Curran Associates Inc. 118

[LCL18] Jiaxin Li, Ben M. Chen, and Gim Hee Lee. So-net : Self-organizing network for point cloud analysis. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9397–9406, 2018. 49

[LCLL14] Cheng Yuan Liou, Wei Chen Cheng, Jiun Wei Liou, and Daw Ran Liou. Autoencoder for words. *Neurocomputing*, 139 :84–96, 2014. 36

[LCY14] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, abs/1312.4, 2014. 35

[LD18] Truc Le and Ye Duan. PointGrid : A Deep Network for 3D Shape Understanding. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 9204–9214, 2018. 54

[LDT+17] Felix Järemo Lawin, Martin Danelljan, Patrik Tosteberg, Goutam Bhat, Fahad Shahbaz Khan, and Michael Felsberg. Deep projective 3D semantic segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10424 LNCS, pages 95–107, 2017. 48

[LF10] Kevin Lai and Dieter Fox. Object recognition in 3D point clouds using web data and domain adaptation. *International Journal of Robotics Research*, 29(8) :1019–1037, 2010. 101, 102, 103, 105

[LF19] Chen Liu and Yasutaka Furukawa. MASC : Multi-scale Affinity with Sparse Convolution for 3D Instance Segmentation. *ArXiv*, abs/1902.0, 2019. 66

[LFXP19] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :8887–8896, 2019. 53

[LGG+17] Tsung Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal Loss for Dense Object Detection. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2017-October, pages 2999–3007, 2017. 144

[LGKX19] Xuesong Li, Jose E Guivant, Ngaiming Kwok, and Yongzhi Xu. 3D Backbone Network for 3D Object Detection. *ArXiv*, abs/1901.0 :1–10, 2019. 63

[Lid05] Ground Lidar. Change Detection on Points Cloud Data Acquired With a Ground Laser Scanner, 2005. 146

[LLF+19] Ruihui Li, Xianzhi Li, Chi Wing Fu, Daniel Cohen-Or, and Pheng Ann Heng. PU-GAN : A point cloud upsampling adversarial network. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October :7202–7211, 2019. 53

[Llo82] Stuart P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2) :129–137, 1982. 19, 74

[LLZL16] Lin Li, Dalin Li, Haihong Zhu, and You Li. A dual growing method for the automatic extraction of individual trees from mobile laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 120 :37–52, 2016. 28

[LO16] Loic Landrieu and Guillaume Obozinski. Cut pursuit : Fast algorithms to learn piecewise constant functions. *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016*, 10 :1384–1393, 2016. 58

[LQG19] Xingyu Liu, Charles R. Qi, and Leonidas J. Guibas. Flownet3d : Learning scene flow in 3D point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :529–537, 2019. 53

[LS18] Loic Landrieu and Martin Simonovsky. Large-Scale Point Cloud Semantic Segmentation with Superpoint Graphs. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018. 55, 58

[LVC+19] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars : Fast encoders for object detection from point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :12689–12697, 2019. 63

[LYWU18] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep Continuous Fusion for Multi-sensor 3D Object Detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11220 LNCS, pages 663–678, 2018. 63

[LZ15] Xiangguo Lin and Jixian Zhanga. Segmentation-based ground points detection from Mobile Laser Scanning point cloud. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 40(7W4) :99–102, 2015. 26

[LZZ+19] Chun Liang Li, Manzil Zaheer, Yang Zhang, Barnabás Póczos, and Ruslan Salakhutdinov. Point cloud gan. *Deep Generative Models for Highly Structured Data, DGS@ICLR 2019 Workshop*, abs/1810.0, 2019. 53

[Mah36] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2 :49–55, 1936. 77

[MBVH09] Daniel Munoz, J. Andrew Bagnell, Nicolas Vandapel, and Martial Hebert. Contextual classification with functional max-margin markov networks. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, volume 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 975–982, 2009. 102

[MGL+18] Erxue Min, Xifeng Guo, Qiang Liu, Gen Zhang, Jianjing Cui, and Jun Long. A Survey of Clustering with Deep Learning : From the Perspective of Network Architecture. *IEEE Access*, 6 :39501–39514, 2018. 74

[MS15] Daniel Maturana and Sebastian Scherer. VoxNet : A 3D Convolutional Neural Network for real-time object recognition. *IEEE International Conference on Intelligent Robots and Systems*, 2015-December :922–928, 2015. 6, 45, 46, 47, 54

[Ove00] Gary D Overturf. Fast Marching farthest point sampling. In *University of Cambridge Computer*, volume 106, page 367, 2000. 55

[Pas16] Jerôme Pasquet. *Modélisation, détection et classification d'objets urbains à partir d'images photographiques aériennes*. PhD thesis, Université de Montpellier, 2016. 6

[PCS+18] Lionel Pibre, Marc Chaumon, Gerard Subsol, Dino Lenco, and Mustapha Derras. How to deal with multi-source data for tree detection based on deep learning. *2017 IEEE Global Conference on Signal and Information Processing, GlobalSIP 2017 - Proceedings*, 2018-January :1150–1154, 2018. 6

[PCSD15] J. Pasquet, M. Chaumont, G. Subsol, and M. Derras. An efficient multi-resolution SVM network approach for object detection in aerial images. *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, 2015-November :1–6, 2015. 6

[PDB+16] Jérôme Pasquet, Thibault Desert, Olivier Bartoli, Marc Chaumont, Carole Delenne, Gérard Subsol, Mustapha Derras, and Nanée Chahinian. Detection of Manhole Covers in High-Resolution Aerial Images of Urban Areas by Combining Two Methods. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(5) :1802–1807, 2016. 6

[Pea01] Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11) :559–572, 1901. 28

[Pib18] Lionel Pibre. *Localisation d'objets urbains à partir de sources multiples dont des images aériennes*. PhD thesis, Université de Montpellier, 2018. 6

[PNH+19] Quang Hieu Pham, Thanh Nguyen, Binh Son Hua, Gemma Roig, and Sai Kit Yeung. JSIS3D : Joint semantic-instance segmentation of 3D point clouds with multi-task pointwise networks and multi-value conditional random fields. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :8819–8828, 2019. 66

[PRM+00] Eric Paquet, Marc Rioux, Anil Murching, Thumpudi Naveen, and Ali Tabatabai. Description of shape information for 2-D and 3-D objects. *Signal Processing : Image Communication*, 16(1) :103–122, 2000. 24

[PSDH16] Michael Palace, Franklin B. Sullivan, Mark Ducey, and Christina Herrick. Estimating tropical forest structure using a terrestrial lidar. *PLoS ONE*, 11(4), 2016. 9

[QLHG19] Charles R. Qi, Or Litany, Kaiming He, and Leonidas Guibas. Deep hough voting for 3D object detection in point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-October, pages 9276–9285, 2019. 66, 119, 130, 135, 136

[QLW+18] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018. 62, 63

[QSMG17] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet : Deep learning on point sets for 3D classification and segmentation. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January, pages 77–85, 2017. 38, 50, 51, 52, 53, 54, 55, 56, 58, 60, 61, 62, 63, 64, 65, 67, 68, 78, 80, 81, 101, 106, 111, 112, 115, 117, 118, 131, 133

[QUD12] A. Quadros, J. P. Underwood, and B. Douillard. An occlusion-aware feature for range images. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 4428–4435. IEEE, 2012. 101, 102, 103, 105, 106

[QYSG17] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++ : Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, volume 2017-December, pages 5100–5109, 2017. 54, 55, 56, 57, 58, 64, 66, 120, 135

[RBB09] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D registration. *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009. 24

[RDG18] Xavier Roynard, Jean Emmanuel Deschaud, and François Goulette. Paris-Lille-3D : A large and high-quality ground-truth urban point cloud dataset for automatic segmentation and classification. *International Journal of Robotics Research*, 37(6) :545–557, 2018. 68, 101, 102, 109, 114

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016. 44, 45

[RDS+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3) :211–252, 2015. 35

[RF17] Joseph Redmon and Ali Farhadi. Yolo9000 : Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525, 2017. 45

[RHGS17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN : Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6) :1137–1149, 2017. 41, 43, 62, 70, 87

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088) :533–536, 1986. 33

[RLG+20] Marie Julie Rakotosaona, Vittorio La Barbera, Paul Guerrero, Niloy J. Mitra, and Maks Ovsjanikov. PointCleanNet : Learning to Denoise and Remove Outliers from Dense Point Clouds. *Computer Graphics Forum*, 39(1) :185–203, 2020. 53

[Ros58] F. Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6) :386–408, 1958. 30

[RROG18] Riccardo Roveri, Lukas Rahmann, A. Cengiz Oztireli, and Markus Gross. A Network Architecture for Point Cloud Classification via Automatic Depth Images

Generation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4176–4184, 2018. 53

[RSM+16] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The SYNTHIA Dataset : A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, pages 3234–3243, 2016. 144

[RWS+18] Dario Rethage, Johanna Wald, Jürgen Sturm, Nassir Navab, and Federico Tombari. Fully-Convolutional Point Networks for Large-Scale Point Clouds. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11208 LNCS, pages 625–640, 2018. 53

[SBZB15] Baoguang Shi, Song Bai, Zhichao Zhou, and Xiang Bai. DeepPano : Deep Panoramic Representation for 3-D Shape Recognition. *IEEE Signal Processing Letters*, 22(12) :2339–2343, 2015. 48

[SEKX98] Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases : The algorithm GDBSCAN and its applications. *Data Mining and Knowledge Discovery*, 2(2) :169–194, 1998. 20

[SEZ+14] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat : Integrated recognition, localization and detection using convolutional networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, abs/1312.6, 2014. 38

[SKT19] Kiwoo Shin, Youngwook Paul Kwon, and Masayoshi Tomizuka. RoarNet : A Robust 3D object detection based on region approximation refinement. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2019-June :2510–2515, 2019. 63

[SLD17] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4) :640–651, 2017. 36

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June-2015 :1–9, 2015. 36

[SLK19] Muhammad Sarmad, Hyunjoo Jenny Lee, and Young Min Kim. RL-GAN-net : A reinforcement learning agent controlled gan network for real-time point cloud shape completion. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :5891–5900, 2019. 53

[SLL+17] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod : Learning deeply supervised object detectors from scratch. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1937–1945, 2017. 45

[SLX15] Shuran Song, Samuel P. Lichtenberg, and Jianxiong Xiao. SUN RGB-D : A RGB-D scene understanding benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June-2015 :567–576, 2015. 101, 118, 133, 134, 135

[SMAG18] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-YOLO : Real-time 3D Object Detection on Point Clouds. *ArXiv*, abs/1803.0, 2018. 86

[SMAG19] Martin Simon, Stefan Milz, Karl Amende, and Horst Michael Gross. Complex-YOLO : An euler-region-proposal for real-time 3D object detection on point clouds. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11129 LNCS, pages 197–209, 2019. 134

[SMGD14] Andrés Serna, Beatriz Marcotegui, François Goulette, and Jean Emmanuel Deschaud. Paris-rue-madame database : A 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods. In *ICPRAM 2014 - Proceedings of the 3rd International Conference on Pattern Recognition Applications and Methods*, pages 819–824, 2014. 101, 102, 103, 105, 106

[SMKLM15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. *Proceedings of the IEEE International Conference on Computer Vision*, 2015 International Conference on Computer Vision, ICCV 2015 :945–953, 2015. 45, 46, 48, 49

[SPK19] Dongwook Shu, Sung Woo Park, and Junseok Kwon. 3D point cloud generative adversarial network based on tree structured graph convolutions. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-October, pages 3858–3867, 2019. 144

[SSE+17] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. DBSCAN revisited, revisited : Why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems*, 42(3) :19 :1–19 :21, 2017. 20

[STD14] Samuele Salti, Federico Tombari, and Luigi Di Stefano. SHOT : Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125 :251–264, 2014. 24

[STMA12] Helmi Z.M. Shafri, Ebrahim Taherzadeh, Shattri Mansor, and Ravshan Ashurov. Hyperspectral remote sensing of urban areas : An overview of techniques and applications. *Research Journal of Applied Sciences, Engineering and Technology*, 4(11) :1557–1565, 2012. 145

[SWL19] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN : 3D object proposal generation and detection from point cloud. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :770–779, 2019. 64, 65, 89, 133, 135, 136

[SWL+20] Yongbin Sun, Yue Wang, Ziwei Liu, Joshua E. Siegel, and Sanjay E. Sarma. PointGrow : Autoregressively learned point cloud generation with self-attention. *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*, pages 61–70, 2020. 53

[SWTZ19] Thomas Stark, Michael Wurm, Hannes Taubenbock, and Xiao Xiang Zhu. Slum mapping in imbalanced remote sensing datasets using transfer learned deep features. *2019 Joint Urban Remote Sensing Event, JURSE 2019*, pages 1–4, 2019. 6

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, abs/1409.1, 2015. 35

[TA10] Gian Diego Tipaldi and Kai O. Arras. FLIRT - Interest regions for 2D range data. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3616–3622, 2010. 46

[TC15] Tee Ann Teo and Chi Min Chiu. Pole-Like Road Object Detection from Mobile Lidar System Using a Coarse-to-Fine Approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(10) :4805–4818, 2015. 28

[TGY20] Zheng Tong, Jie Gao, and Dongdong Yuan. Advances of deep learning applications in ground-penetrating radar : A survey. *Construction and Building Materials*, 258 :120371, 2020. 145

[Thr03] Sebastian Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous Robots*, 15(2) :111–127, 2003. 46

[TP05] D. Tóvári and N. Pfeifer. Segmentation based robust interpolation - A newapproach to laser data filtering. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, volume 36, 2005. 21

[TQD+19] Hugues Thomas, Charles R. Qi, Jean Emmanuel Deschaud, Beatriz Marcotegui, Francois Goulette, and Leonidas Guibas. KPConv : Flexible and deformable convolution for point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October :6410–6419, 2019. 60, 83

[Tuc79] Compton J. Tucker. Red and photographic infrared linear combinations for monitoring vegetation. *Remote Sensing of Environment*, 8(2) :127–150, 1979. 145

[UVGS13] J. R.R. Uijlings, K. E.A. Van De Sande, T. Gevers, and A. W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2) :154–171, 2013. 39

[VBV18] Nitika Verma, Edmond Boyer, and Jakob Verbeek. FeaStNet : Feature-Steered Graph Convolutions for 3D Shape Analysis. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2598–2606, 2018. 53, 79, 83, 118

[VD10] Jean Philippe Vasseur and Adam Dunkels. *Interconnecting Smart Objects with IP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. 5, 6

[VGRB19] Ashley Varghese, Jayavardhana Gubbi, Akshaya Ramaswamy, and P. Balamuralidhar. ChangeNet : A deep learning architecture for visual change detection. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11130 LNCS, pages 129–145, 2019. 146

[VLHB20] Sourabh Vora, Alex H. Lang, Bassam Helou, and Oscar Beijbom. Pointpainting : Sequential fusion for 3D object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, abs/1911.1 :4603–4611, 2020. 64

[VS05] Miguel Vieira and Kenji Shimada. Surface mesh segmentation and smooth surface extraction through region growing. *Computer Aided Geometric Design*, 22(8) :771–792, 2005. 21

[WBF11] Ruisheng Wang, Jeff Bach, and Frank P. Ferrie. Window detection from mobile LiDAR data. *2011 IEEE Workshop on Applications of Computer Vision, WACV 2011*, pages 58–65, 2011. 28

[WJ19] Zhixin Wang and Kui Jia. Frustum ConvNet : Sliding Frustums to Aggregate Local Point-Wise Features for Amodal. *IEEE International Conference on Intelligent Robots and Systems*, pages 1742–1749, 2019. 63

[WJW+19] Chunxiao Wang, Min Ji, Jian Wang, Wei Wen, Ting Li, and Yong Sun. An improved DBSCAN method for LiDAR data segmentation with automatic Eps estimation. *Sensors (Switzerland)*, 19(1), 2019. 21

[WLHJ+20] Zirui Wang, Shuda Li, Henry Howard-Jenkins, Victor Adrian Prisacariu, and Min Chen. FlowNet3D++ : Geometric losses for deep scene flow estimation. *Proceedings - 2020 IEEE Winter Conference on Applications of Computer Vision, WACV 2020*, pages 91–98, 2020. 53

[WLS+19] Xinlong Wang, Shu Liu, Xiaoyong Shen, Chunhua Shen, and Jiaya Jia. Associatively segmenting instances and semantics in point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :4091–4100, 2019. 66

[WPC18] Ruisheng Wang, Jiju Peethambaran, and Dong Chen. LiDAR Point Clouds to 3-D Urban Models : A Review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(2) :606–627, 2018. 26

[WSK+15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets : A deep representation for volumetric shapes. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June-2015, pages 1912–1920, 2015. 144

[WSL+19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph Cnn for learning on point clouds. *ACM Transactions on Graphics*, 38(5), 2019. 55, 57, 81, 82, 118

[WWYK18] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. SqueezeSeg : Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1887–1893, 2018. 145

[WYHN18] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. SGPN : Similarity Group Proposal Network for 3D Point Cloud Instance Segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2569–2578, 2018. 65, 66, 67, 143

[WYY+13] Bin Wu, Bailang Yu, Wenhui Yue, Song Shu, Wenqi Tan, Chunling Hu, Yan Huang, Jianping Wu, and Hongxing Liu. A voxel-based method for automated identification and morphological parameters estimation of individual street trees from mobile laser scanning data. *Remote Sensing*, 5(2) :584–611, 2013. 28

[WZ21] Weihao Weng and Xin Zhu. INet : Convolutional Networks for Biomedical Image Segmentation. *IEEE Access*, 9 :16591–16603, 2021. 36, 57

[WZZ+19] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. SqueezeSegV2 : Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2019-May, pages 4376–4382, 2019. 145

[XAJ18] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. PointFusion : Deep Sensor Fusion for 3D Bounding Box Estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 244–253, 2018. 63

[XVSP16] Wen Xiao, Bruno Vallet, Konrad Schindler, and Nicolas Paparoditis. Street-side vehicle detection, classification and change detection using mobile laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 114 :166–178, 2016. 28

[YHCOZ18] Kangxue Yin, Hui Huang, Daniel Cohen-Or, and Hao Zhang. P2P-NET : Bi-directional point displacement net for shape transform. *ACM Transactions on Graphics*, 37(4) :1–13, 2018. 53

[YHH+19] Guandao Yang, Xun Huang, Zekun Hao, Ming Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow : 3D point cloud generation with continuous normalizing flows. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October :4540–4549, 2019. 53

[YK94] Antti YlaJaaski and Nahum Kiryati. Adaptive Termination of Voting in the Probabilistic Circular Hough Transform. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9) :911–915, 1994. 23

[YKH+18] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. PCN : Point completion network. *Proceedings - 2018 International Conference on 3D Vision, 3DV 2018*, pages 728–737, 2018. 53

[YLF+18a] Lequan Yu, Xianzhi Li, Chi Wing Fu, Daniel Cohen-Or, and Pheng Ann Heng. EC-Net : An edge-aware point set consolidation network. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11211 LNCS, pages 398–414, 2018. 53

[YLF+18b] Lequan Yu, Xianzhi Li, Chi Wing Fu, Daniel Cohen-Or, and Pheng Ann Heng. PU-Net : Point Cloud Upsampling Network. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2018. 53

[YLG+14] Y. Yu, J. Li, H. Guan, D. Zai, and C. Wang. Automated extraction of 3D trees from mobile LiDAR point clouds. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 40(5) :629–632, 2014. 18

[YLU18] Bin Yang, Wenjie Luo, and Raquel Urtasun. PIXOR : Real-time 3D Object Detection from Point Clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018. 63

[YLU20] Bin Yang, Ming Liang, and Raquel Urtasun. HDNET : Exploiting HD Maps for 3D Object Detection. In *CoRL*, 2020. 63

[YML18] Yan Yan, Yuxing Mao, and Bo Li. Second : Sparsely embedded convolutional detection. *Sensors (Switzerland)*, 18(10), 2018. 63

[YMST16] Wai Yeung Yan, Salem Morsy, Ahmed Shaker, and Mark Tulloch. Automatic extraction of highway light poles and towers from mobile LiDAR data. *Optics and Laser Technology*, 77 :162–168, 2016. 28

[YSGG17] Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. SyncSpecCNN : Synchronized spectral CNN for 3D shape segmentation. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-January :6584–6592, 2017. 53

[YSL17] Manohar Yadav, Ajai Kumar Singh, and Bharat Lohani. Extraction of road surface from mobile LiDAR data of complex road environment. *International Journal of Remote Sensing*, 38(16) :4645–4672, 2017. 26, 27

[YSL+18] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. IPOD : Intensive Point-based Object Detector for Point Cloud. *ArXiv*, abs/1812.0, 2018. 64

[YSL+19] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. STD : Sparse-to-dense 3D object detector for point cloud. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2019-October, pages 1951–1960, 2019. 130

[YWC+19] Bo Yang, Jianan Wang, Ronald Clark, Qingyong Hu, Sen Wang, Andrew Markham, and Niki Trigoni. Learning object bounding boxes for 3D instance segmentation on point clouds. *Advances in Neural Information Processing Systems*, 32, 2019. 66

[YWH+19] Wang Yifan, Shihao Wu, Hui Huang, Daniel Cohen-Or, and Olga Sorkine-Hornung. Patch-based progressive 3D point set upsampling. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :5951–5960, 2019. 53

[ZBDT19] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3D point capsule networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :1009–1018, 2019. 53

[ZCS+18] Younes Zegaoui, Marc Chaumont, Gérard Subsol, Philippe Borianne, and Mustapha Derras. First Experiments of Deep Learning on LiDAR Point Clouds for Classification of Urban Objects First Ex-periments of Deep Learning on LiDAR Point Clouds for Classification of Urban Objects. CFPT : Conférence Française de Photogrammétrie et de Télédétection. *CFPT*, 2018. 103

[ZCS+19] Younes Zegaoui, Marc Chaumont, Gerard Subsol, Philippe Borianne, and Mustapha Derras. Urban object classification with 3D deep-learning. In *2019 Joint Urban Remote Sensing Event, JURSE 2019*, Vannes, France, 2019. 103

[ZCX+17] Lishan Zhong, Liang Cheng, Hao Xu, Yang Wu, Yanming Chen, and Manchun Li. Segmentation of Individual Trees from TLS and MLS Data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 10(2) :774–787, 2017. 28

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8689 LNCS, pages 818–833, 2014. 35

[ZGF+20] Feihu Zhang, Chenye Guan, Jin Fang, Song Bai, Ruigang Yang, Philip H.S. Torr, and Victor Prisacariu. Instance Segmentation of LiDAR Point Clouds. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 9448–9455, 2020. 63

[ZGG19] Jesus Zarzar, Silvio Giancola, and Bernard Ghanem. PointRGCN : Graph Convolution Networks for 3D Vehicles Detection Refinement. *ArXiv*, abs/1911.1, 2019. 64

[ZHL+18] Yiming Zeng, Yu Hu, Shice Liu, Jing Ye, Yinhe Han, Xiaowei Li, and Ninghui Sun. RT3D : Real-time 3-D vehicle detection in LiDAR point cloud for autonomous driving. *IEEE Robotics and Automation Letters*, 3(4) :3434–3440, 2018. 63

[ZHW+19] Kuangen Zhang, Ming Hao, Jing Wang, Clarence W. de Silva, and Chenglong Fu. Linked Dynamic Graph CNN : Learning on Point Cloud via Linking Hierarchical Features. *ArXiv*, abs/1904.1, 2019. 55, 57

[ZJFJ19] Hengshuang Zhao, Li Jiang, Chi Wing Fu, and Jiaya Jia. Pointweb : Enhancing local neighborhood features for point cloud processing. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June :5560–5568, 2019. 143

[ZKG18] Wei Zeng, Sezer Karaoglu, and Theo Gevers. Inferring Point Clouds from Single Monocular Images by Depth Intermediation. *ArXiv*, abs/1812.0, 2018. 53

[ZLHH19] Xin Zhao, Zhe Liu, Ruolan Hu, and Kaiqi Huang. 3D object detection using scale invariant and feature reweighting networks. In *33rd AAAI Conference on Artificial Intelligence, AAAI 2019, 31st Innovative Applications of Artificial Intelligence Conference, IAAI 2019 and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019*, pages 9267–9274, 2019. 63

[ZP02] Titus Zaharia and Françoise Prêteux. Shape-based retrieval of 3D mesh models. In *Proceedings - 2002 IEEE International Conference on Multimedia and Expo, ICME 2002*, volume 1, pages 437–440, 2002. 23

[ZT18a] Maciej Zamorski and Tomasz Trzcinski. Adversarial Autoencoders for Compact Representations of 3D Point Clouds Adversarial Autoencoders for Compact Representations of 3D Point Clouds. *ArXiv*, abs/1811.0(November), 2018. 53

[ZT18b] Yin Zhou and Oncel Tuzel. VoxelNet : End-to-End Learning for Point Cloud Based 3D Object Detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018. 63

[ZWL+20] Haidi Zhu, Haoran Wei, Baoqing Li, Xiaobing Yuan, and Nasser Kehtarnavaz. A review of video object detection : Datasets, metrics and methods. *Applied Sciences (Switzerland)*, 10(21) :1–24, 2020. 146

[ZZXW19] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu. Object detection with deep learning : A review. *IEEE transactions on neural networks and learning systems*, 30(11) :32123232, November 2019. 44