Détection d'objets urbains dans des nuages de points LiDAR

Y. Zegaoui[1], P. Borianne[2], M. Chaumont[3], G. Subsol[3] A. Seriai[1], M. Derras[1]

[1] Berger-Levrault
[2] CIRAD [3] LIRMM

{younes.zegaoui, abderrahmane.seriai, mustapha.derras}@berger-levrault.com, {marc.chaumont, gerard.subsol}@lirmm.fr, philippe.borianne@cirad.fr

Résumé

Les travaux en apprentissage profond sur les données 3D sont depuis peu très étudié. Leurs applications en milieu urbain permettraient de faciliter grandement la gestion des objets urbains par les agglomérations. Nous proposons une méthode originale pour la détection des objets urbains dans des nuages de points 3D issus d'acquisitions LiDAR.

Mots-clés

LiDAR, apprentissage profond, nuages de points 3D, convolution 3D, détection d'objets.

Abstract

Deep learning based research on 3D data has recently gained a lot of interest. Their application in urban areas would greatly facilitate the monitoring of urban objects by city managers. We propose an original method for the detection of urban objects in 3D point clouds from LiDAR acquisitions.

Keywords

LiDAR, deep-learning, 3D points cloud, 3D convolution, object detection.

1 Introduction

1.1 Gestion technique des équipements urbains

Le groupe Berger-Levrault, éditeur historique de logiciels administratifs pluriels, compte parmi ses clients un nombre important de collectivités territoriales avec des produits adaptés à leur diverses missions. Dans notre cas, nous nous intéressons particulièrement à la gestion technique des équipements en milieu urbain et notamment les objets urbains.

Nous désignons comme objet urbain tout objet disposé dans l'espace public et lié à un service offert par la collectivité territoriale. Cette définition inclut une grande diversité de cas, parmi lesquels des objets fixes tels que les lampadaires, des objets mobiles comme les poubelles et même des objets vivants comme les arbres urbains.

La mission des gestionnaires de ville est alors d'être en mesure de surveiller l'ensemble des objets urbains, de suivre l'évolution de leurs statuts au fil du temps ainsi que de prévenir les interactions problématiques entre objets urbains, comme par exemple une chute de branche d'arbres sur des câbles. La difficulté de cette tâche est d'autant plus grande dans les agglomérations urbaines où le nombre important d'objets urbains constitue le défi principal. Il serait alors intéressant d'automatiser en partie la chaîne de traitement et de suivi des objets urbains et en particulier leur recensement.

1.2 Reconnaissance d'objets dans des nuages de points LiDAR

Nos travaux s'inscrivent dans la continuité des recherches sur la télédétection en milieu urbain, réalisées en collaboration avec le laboratoire d'informatique de robotique et de micro-électronique de Montpellier [10, 11]. Les développements récents dans le domaine de la reconnaissance d'objets dans les nuages de points, notamment basés sur l'apprentissage profond, nous encourage à expérimenter la technologie LiDAR terrestre comme dispositif d'acquisition des objets urbains.

En effet, les nuages de points issus d'acquisition LiDAR comptent habituellement plusieurs millions de points et peuvent recouvrir plusieurs kilomètres. De plus la manipulation de données 3D n'étant pas une tâche triviale, un traitement automatique de ceux-ci afin d'en extraire les éléments d'intérêt, ici les objets urbains, s'avère une nécessité. Ainsi, nous comptons nous baser sur les avancées récentes afin de développer une méthode capable de réaliser une détection d'objets, c'est-à-dire prédire des boîtes englobantes autour des objets (voir figure 1 b), ce qui est différent de la segmentation sémantique, qui consiste à associer une classe à chaque point du nuage (voir figure 1 a).

Nous présentons en section 2 une revue des travaux connexes, en section 3 nous détaillons notre contribution et nous présentons les diverses expérimentations réalisées autour de celle-ci en section 4.

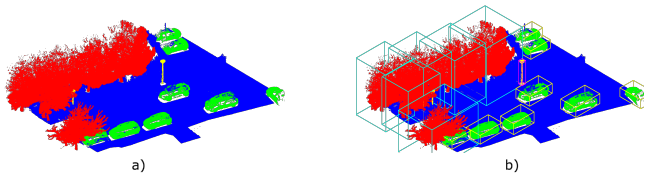

FIGURE 1 – Illustration de la tâche de segmentation sémantique (a) où une classe, représentée par une couleur, est associée à chaque point, et de la tâche de détection d'objets (b), où des boites englobantes sont définies autour des objets présents.

2 Travaux connexes

2.1 Apprentissage profond à partir de nuages de points 3D

2.1.1 Approches par structuration

Les premières tentatives de généralisation des réseaux *CNN* aux nuages de points 3D s'appuient sur une structuration préalable des nuages de points. Ainsi le réseau *VoxNet* [8] propose de discrétiser le nuage de points sous forme de voxels avant de le passer en entrée d'un *CNN*, dont l'architecture est légèrement modifiée afin de prendre en compte la dimension supplémentaire. Le réseau *MVCNN* [20] propose quant à lui de simuler des photos, appelées vues, prises tout autour d'un nuage de points et de passer celles-ci en entrée d'un *CNN* classique traitant des images 2D et de fusionner les différentes sorties entre elles.

Bien que ces méthodes par structuration aient permis d'adapter les méthodes de *deep-learning* aux nuages de points et d'obtenir les premiers résultats, elles comportent des limites importantes. Elles ont globalement pour conséquence de discrétiser la forme des nuages de points, causant ainsi une perte d'informations. De plus, les approches par multi-vues sont peu adaptées aux grands nuages de points avec beaucoup d'occultations et nécessitent souvent un maillage des nuages de points. Les approches volumiques souffrent quant à elles d'un coût en mémoire accrue causée par l'utilisation de convolutions volumiques. A noter sur ce point que les convolutions éparses (*sparse convolutions*) [24] permettent de réduire considérablement l'impact de ce défaut.

2.1.2 PointNet et ses successeurs

Le réseau *PointNet* [14] a introduit une architecture capable de réaliser l'apprentissage directement à partir du nuage de points sans transformation intermédiaire. Cette architecture repose en grande partie sur l'utilisation de perceptrons multi-couches (*multi-layer perceptrons*). Ceux-ci sont appliqués indépendamment sur chaque point du nuage en prenant uniquement les coordonnées (x, y, z) des points en entrée, permettant ainsi d'obtenir des vecteurs de *caractéristiques ponctuelles* pour chaque point, en rouge dans la figure 2. Une opération d'agrégation maximale est ensuite appliquée sur ces vecteurs afin d'obtenir le vecteur de caractéristique global, en vert dans la figure 2.

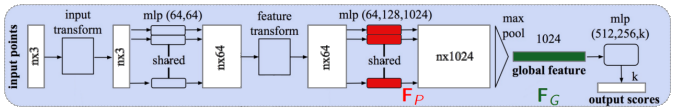

FIGURE 2 – Schéma de l'architecture du réseau *PointNet* [14], avec en rouge les vecteurs de *caractéristiques ponctuelles* et en vert le vecteur de *caractéristiques globales*.

La principale limite du réseau *PointNet* [14] est le manque de prise en compte du voisinage lors du calcul des *caractéristiques ponctuelles*, en effet celles-ci sont calculées pour chaque point indépendamment des autres or l'utilisation du voisinage est importante dans certaines applications comme la segmentation sémantique. Plusieurs méthodes ont alors proposé d'étendre *PointNet* [14] afin d'y ajouter des mécanismes permettant l'extraction de caractéristiques locales.

Le réseau *PointNet++* [15] propose d'appliquer le réseau *PointNet* [14] localement à plusieurs échelles en sous-échantillonnant à chaque étape le nuage de points. Le réseau *RandLaNet* [4] reprend cette idée en choisissant un sous-échantillonnage aléatoire plutôt que celui basé sur le *farthest point sampling (FPS)* et en le combinant avec un module d'attention. Le réseau *DGCNN* [22] propose d'utiliser un graphe dynamique afin de lier les *caractéristiques ponctuelles* entre elles. *SPG* [6] propose de partitionner le nuage de points en *super-points* liés entre eux dans une structure de graphe. *KP-Conv* [21] et *ConvPoint* [1] proposent d'étendre l'opération de convolution aux nuages de points 3D en définissant des noyaux de convolutions à partir de points.

2.2 Détection d'objets dans des nuages de points

Les méthodes post-*PointNet* [14] donnent de très bons résultats pour la tâche de segmentation sémantique, notamment dans le cas de nuages de points en extérieur. Néanmoins, ceux-ci ne permettent pas de réaliser la tâche de détection d'objets car elles ne comportent pas de notion de groupement de points : les traitements sont effectués uniquement au niveau des points (*caractéristiques ponctuelles*) et au niveau du nuage entier (*caractéristiques globales*). Ainsi, les méthodes de détection d'objets dans des nuages de points ont pour caractéristique d'ajouter un niveau intermédiaire représentant des groupements de points qui peuvent, ou non, contenir un objet. Dans la suite nous appelons ces groupements de points *régions* et nous distinguons plusieurs familles de méthodes selon leur manière de définir les *régions*.

2.2.1 Approches par vue d'oiseau

Les approches par vue d'oiseau s'appuient sur un concept simple : si nous observons un nuage de points 3D selon l'axe vertical, il est tentant de schématiser celui-ci sous la forme d'une image aérienne. Ainsi le nuage de points est partitionné selon une grille régulière horizontale et pour chaque cellule de cette grille, des *régions* sont définies en suivant le modèle des ancres (*anchor boxes*) utilisées dans

la détection d'objets dans des images [16]. Il est alors possible d'entraîner un réseau *CNN* à détecter des objets à partir de cette grille.
Le réseau *AVOD* [5] propose de combiner une grille horizontale (image aérienne) et une grille verticale (image frontale) et d'utiliser un premier *CNN* pour détecter les objets présents à partir de chacune des deux grilles, puis un second *CNN* afin de fusionner les prédictions. Les réseaux *VoxelNet* [26] et *PointPillars* [7] proposent d'utiliser *PointNet* [14] afin d'encoder la valeur des cellules de la grille horizontale en fonction des points qu'elles contiennent.

2.2.2 Projection d'images

Les approches par projection d'images utilisent, en plus du nuage de points, une photo rigoureusement étalonnée par rapport au nuage de points. Il est alors possible de faire le lien entre les pixels de cette photo et l'espace géométrique dans lequel est défini le nuage de points. La méthodologie est alors simple : un détecteur d'objets est appliqué sur la photo et permet d'obtenir des boîtes englobantes 2D. Ces boîtes sont ensuite projetées dans l'espace géométrique du nuage afin d'obtenir des boîtes englobantes 3D : il s'agit des *régions*. Enfin, les *régions* sont passés en entrée d'un réseau *PointNet* [14] afin de détecter les objets qui s'y trouvent.
Le réseau *FrustumNet* [13] est la première méthode introduisant cette méthodologie. Celui-ci a été plusieurs fois repris et amélioré, par exemple en appliquant le réseau *PointNet* plusieurs fois pour chaque *région* en "glissant" selon l'axe horizontal [23].

2.2.3 Génération de *régions* à partir des points

Les méthodes de génération de *régions* consistent à travailler directement sur les points à la manière de *PointNet*. Ainsi le réseau *PointRCNN* [19] propose dans un premier temps de segmenter le nuage de points entre les points de premier plan, appartenant à un objet, et les points d'arrière-plan, appartenant au fond. Dans un deuxième temps, des *régions* sont définies autour des points de premier plan et un réseau *PointNet* [14] est appliqué afin d'y détecter les objets présents. Le réseau *PointRGCN* [25] propose d'ajouter un mécanisme de *graph-convolution* afin de lier les *régions* entre elles et d'exploiter les relations pertinentes pour la détection d'objets. Le réseau *VoteNet* [12] propose de faire "voter" les points du premier plan, c'est à dire de prédire pour chacun d'entre eux un nouveau point censé être le centre de l'objet le plus proche. Les *régions* sont alors obtenues en groupant les *votes* les plus proches.

2.3 Bilan

Cette revue de l'état de l'art nous permet de présenter les familles de méthodes permettant d'obtenir de bons résultats sur les tâches de détection d'objets dans des nuages de points 3D en extérieur. Cependant aucune de ces méthodes n'a, à notre connaissance, été appliquée au problème de la détection d'objets urbains. De plus nous pouvons objecter que les méthodes par vue d'oiseau, bien que très performante dans le cas des voitures autonomes, ne sont pas bien adaptées aux problèmes des objets urbains où il est fréquent de retrouver des objets 'superposés' les uns sur les autres (par exemple des lampadaires et des arbres ou bien des poubelles et des abri-bus). Quant aux méthodes par projection d'images, la nécessité d'utiliser une caméra peut être un frein dans la mesure où celle-ci donne une moins grande résolution pour les objets les plus éloignés de la caméra comme les arbres ou les caténaires.
Nous nous donnons alors comme objectif d'étendre les méthodes de génération de *régions* en unifiant leur processus d'apprentissage au lieu d'utiliser deux sous réseau entraînés séquentiellement, ainsi qu'en définissant des *régions* de formes arbitraires au lieu d'utiliser des boîtes englobantes, afin d'être plus proche de la forme des objets urbains.

3 L'architecture *RPC*

Dans cette section nous présentons notre architecture générale pour la détection d'objets urbains dans des nuages de points 3D. Celle-ci se nomme *RPC* pour réseau avec proposition de *cluster*. En effet, notre architecture se base sur la définition de *régions* de formes arbitraires appelées *clusters*. Celle-ci peut être entraînée de bout-en-bout et prend en entrée directement les points du nuage.

Notation : Dans la suite nous désignons un point par $\mathbf{P} \in \mathbb{R}^{d+3}$, qui peut être décomposé en sa partie géométrique $\mathbf{P}_g \in \mathbb{R}^3 = (x, y, z)$ et sa partie caractéristique $\mathbf{P}_f \in \mathbb{R}^d$, qui peut être la couleur du point, son intensité ou bien tout autre vecteur de caractéristiques associé au point. Nous notons un nuage de N points comme $S = \{\mathbf{P}_i\}_1^N$.

3.1 Opération de *cluster-convolution*

Notre architecture se base sur la définition de l'opération originale de *cluster-convolution*. C'est cette opération qui permet d'extraire les informations nécessaires à la génération des *régions* pour la détection d'objets. Cette opération prend en entrée un nuage de points S_l et en extrait des caractéristiques pour aboutir à un nuage de points S_{l+1} plus petit mais plus "profond". Elle peut être formalisée comme ceci :

$$ClConv : \begin{vmatrix} \mathbb{R}^{N_l \times (3+d_l)} \rightarrow \mathbb{R}^{N_{l+1} \times (3+d_{l+1})} \\ S_l \rightarrow S_{l+1} \end{vmatrix} \quad (1)$$

où $d_{l+1} > d_l$ et $N_l > N_{l+1}$. L'opération de *cluster-convolution* est schématisée dans la figure 3. Cette opération peut être appliquée successivement en diminuant à chaque fois la taille du nuage et en augmentant la taille du vecteur de caractéristiques extraites pour chaque point.
Cette opération se décompose en 2 couches de calcul :

- la couche de *clustering*, c'est elle qui permet de diminuer la taille du nuage de points en créant des groupes de points appelés *clusters*
- la couche de *graph-convolution*, c'est elle qui permet d'extraire de nouvelles caractéristiques en exploitant les liens entre les points appartenant à un même *cluster*.

3.1.1 La couche de *clustering*

Le but de la couche de *clustering*, schématisée en figure 4, est de créer des groupements de points cohérents à partir

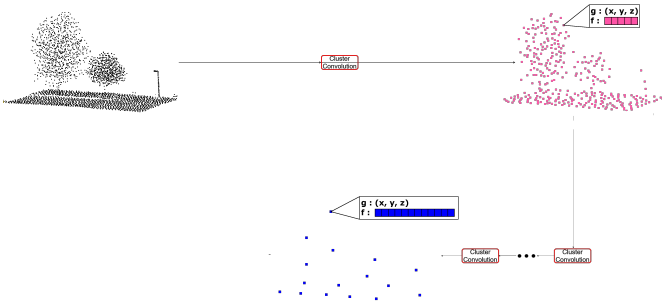

FIGURE 3 – Schéma de l'opération de *cluster-convolution* : un nuage de points est passé en entrée de celle-ci, obtenant en sortie un nuage de points de plus petite taille avec un nouveau de vecteur de caractéristiques pour chaque point. Cette opération peut être appliquée successivement impliquant à chaque fois un nuage de plus petite taille avec des vecteurs de caractéristiques de dimension supérieur.

d'un nuage de points, ces groupements se basent sur la définition d'un nombre fixe de *clusters*, notés K et formant une partition du nuage de points original :

$$\cup_i K_i = S \tag{2}$$

$$\cap_i K_i = \emptyset \tag{3}$$

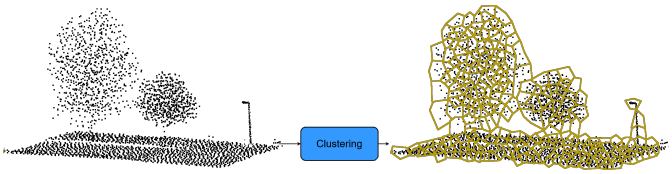

FIGURE 4 – Schéma de la couche de *clustering* avec le nuage de points en entrée qui est partitionnée en plusieurs *clusters* par l'application de la couche.

La cohérence du *clustering* est assuré par le fait que les points regroupés entre eux doivent être aussi similaires que possible aussi bien au niveau de leurs parties géométriques, que de leurs parties caractéristiques. Afin de vérifier au mieux cette propriété, nous nous sommes basés sur l'algorithme du *k-means* afin d'implémenter la couche de *clustering*. La différence principale avec une implémentation classique du *k-means* est la fonction de distance utilisée. En effet, pour pouvoir équilibrer les parties géométriques et caractéristiques dans le calcul des *clusters*, nous avons expérimenté avec plusieurs fonctions de distance.

Distance euclidienne Le choix naïf consiste à utiliser une distance euclidienne :

$$D_e(\mathbf{P}_1, \mathbf{P}_2) = \|\mathbf{P}_1, \mathbf{P}_2\|$$

où $\|\cdot\|$ représente la norme euclidienne. Ici aucune distinction entre partie géométrique et caractéristique n'est faite, ce qui a pour conséquence de déséquilibrer la distance. En effet, mis à part lors de la première étape de *cluster-convolution*, la partie caractéristique des points compte plus de composantes que la partie géométrique et aura donc tendance à s'imposer uniformément. C'est pourquoi nous avons expérimenté plusieurs autres fonctions de distance, permettant d'équilibrer les deux parties.

Distance normalisée Afin d'équilibrer le calcul de distance, une approche peut être de normaliser les parties géométriques et caractéristiques des points en les considérant comme indépendantes :

$$D_{norm}(\mathbf{P}_1, \mathbf{P}_2) = \frac{\|\mathbf{P}_{g_1} - \mathbf{P}_{g_2}\|}{\sigma_g} + \frac{\|\mathbf{P}_{f_1} - \mathbf{P}_{f_2}\|}{\sigma_f}$$

où σ est la variance des séries associées.

Distance pondérée Cette approche consiste à calculer séparément les distances des parties géométriques et caractéristiques, puis de faire la moyenne pondérée de celles-ci :

$$D_\lambda(\mathbf{P}_1, \mathbf{P}_2) = \lambda D_e(\mathbf{P}_{g_1}, \mathbf{P}_{g_2}) + (1-\lambda) D_e(\mathbf{P}_{f_1}, \mathbf{P}_{f_2})$$

où λ est le coefficient de pondération.

Distance maximale En s'inspirant du processus d'apprentissage de *PointNet* [14], nous avons défini la distance par composante maximale. Cette distance consiste à calculer une distance euclidienne sur les parties géométrique, augmentée du carré de la différence entre les composantes maximales des parties caractéristiques :

$$D_{max}(\mathbf{P}_1, \mathbf{P}_2) = \lambda D_e(\mathbf{P}_{g_1}, \mathbf{P}_{g_2}) + (1-\lambda) \left(\max_{\mathbf{P}_{f_1}} - \max_{\mathbf{P}_{f_2}} \right)^2$$

où $\max_{\mathbf{P}_f}$ est la composante maximale du vecteur caractéristique $\mathbf{P}_f$.

Expérimentalement, nous avons déduit que les distances maximales et normalisées permettent un meilleur équilibre entre les parties géométriques et caractéristiques, plus de détails est donné en section 4.

3.1.2 La couche de *graph-convolution*

L'objectif de la couche de *graph-convolution* est d'extraire des caractéristiques pertinentes pour la détection d'objets à partir des points du nuage. Cette opération est appliquée à la suite de la couche de *clustering* et prend en entrée les *clusters* de points. Le but est alors de représenter chaque *cluster* par un unique point dont la partie caractéristique prend en compte la répartition des points dans le *cluster* :

$$G_{conv} : K^l \rightarrow \mathbf{P}^{l+1}$$

où $\mathbf{P}^{l+1}$ est un nouveau point n'appartenant pas au nuage de point original S.

La définition de l'opération de *graph-convolution* diffère donc des opérations calculées par *PointNet* et *DGCNN* (voir comparatif dans la figure 5).

La fonction calculée par *PointNet* [14] est telle que :

$$PointNet : S \rightarrow \mathbf{F}$$

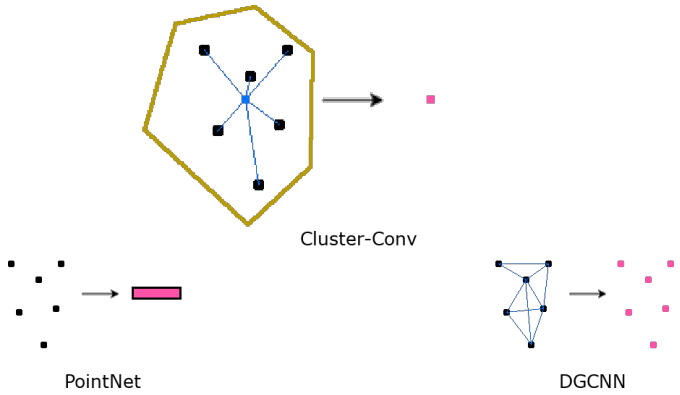

FIGURE 5 – Comparaison entre la couche de *graph-convolution* et les fonctions calculées par *PointNet* [14] et *DGCNN* [22]

où **F** est un vecteur de caractéristique non associé à une partie géométrique, ce qui empêche d'appliquer cette opération de manière successive.
La fonction de *convolution* calculée par *DGCNN* est telle que :

$$DGCNN : \{\mathbf{P}_i\}_1^N \in \mathbb{R}^{3+d_l} \rightarrow \{\mathbf{P}_i\}_1^N \in \mathbb{R}^{3+d_{l+1}}$$

Cette opération permet de calculer des nouvelles caractéristiques pour chaque point mais sans notion de groupes et donc ne permettant pas de restreindre le nombre de points à chaque application.
La procédure de calcul de l'opération de *graph-convolution* est décrite dans la figure 6, celle-ci est décomposée en plusieurs étapes :

- la couche prend en entrée les *clusters* de points précédemment définis
- pour chaque *cluster*, la moyenne des parties géométriques est calculée afin d'obtenir la partie géométrique du résultat $\mathbf{P}_g^{l+1}$
- le vecteur de caractéristique maximal **F** est calculé, puis est concaténé à chacune des parties caractéristiques du *cluster* et passé en entrée d'un perceptron multi-couches
- une opération d'agrégation maximale est appliquée sur la sortie du perceptron multi-couches afin d'obtenir la partie caractéristique du résultat $\mathbf{P}_f^{l+1}$

L'opération de *cluster-convolution* dépend ainsi de deux paramètres principaux :

1. le nombre de *clusters k* (décroissant au fil des applications successives)
2. le nombre de caractéristiques extraites (croissant au fil des applications)

3.2 Prédiction de boîtes englobantes

Après la dernière application de *graph-convolution*, nous obtenons un nuage de points S^l de taille bien inférieure au nuage de points en entrée S. Nous pouvons alors calculer une régression à partir des parties géométriques des points de S^l vers des vecteurs représentants des boîtes englobantes. Les boîtes englobantes sont ainsi définies :

$$\mathbf{B} = (x, y, z, h, w, d, \theta)$$

où *(x, y, z)* est le centre de la boîte englobantes, *h* sa hauteur, *w* sa largeur, *d* sa profondeur et θ son angle d'orientation par rapport à l'axe vertical.
La cible de cette opération de régression peut être déterminée facilement à partir des points $\mathbf{P}_i \in S^l$, en effet comme ceux-ci sont obtenus par application successives de *cluster-convolution*, il suffit de garder en mémoire l'affectation des *clusters* à chaque application de *cluster-convolution* pour pouvoir remonter au nuage de points original. Ainsi, nous pouvons à partir d'un point $\mathbf{P} \in S$ remonter à un sous-ensemble de S, c'est ce sous-ensemble que nous utilisons comme *région* pour la détection d'objets.
Nous considérons que chaque *région* peut au maximum contenir un seul objet que nous appelons l'objet majoritaire, noté O_{maj}. Cet objet est celui dont la proportion de points appartenant à la *région* par rapport à l'objet entier est maximal. La *région* est considérée comme positive si cette proportion est supérieure au seuil τ et négative dans le cas contraire.
La fonction de coût est alors définie en fonction de l'objet majoritaire pour chaque *région* :

$$\mathcal{L}\left(S^l, \mathbf{B}^*\right) = \frac{\alpha}{N_l}\sum_i^{N_l} \mathcal{L}_{cls}\left(\mathbf{B}_i, \mathbf{B}^*_{O_{maj}}\right) + \frac{\beta}{N_l}\sum_i^{N_l} \mathcal{L}_{reg}\left(\mathbf{B}_i, \mathbf{B}^*_{O_{maj}}\right)$$

c'est-à-dire une somme pondérée d'une fonction de classification, $\mathcal{L}_{cls}$, pour déterminer la classe des boîtes englobantes, et d'une fonction de régression, $\mathcal{L}_{reg}$, pour déterminer leurs géométries. Les boîtes englobantes vérité terrain sont notées $\mathbf{B}^*$.
La fonction $\mathcal{L}_{cls}$ est une fonction d'entropie croisée classique :

$$\mathcal{L}_{cls}(x, y) = -\sum_{C=1}^{N_C} y_C log(x_C)$$

où y_C vaut 1 quand la classe C est la classe attendue et 0 sinon.
La fonction $\mathcal{L}_{reg}$ est une fonction de Huber :

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

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

4 Résultats expérimentaux

Dans cette section, nous présentons les résultats que nous avons obtenus sur la tâche de détection d'objets urbains dans des nuages de points 3D à partir de notre architecture *RPC*.

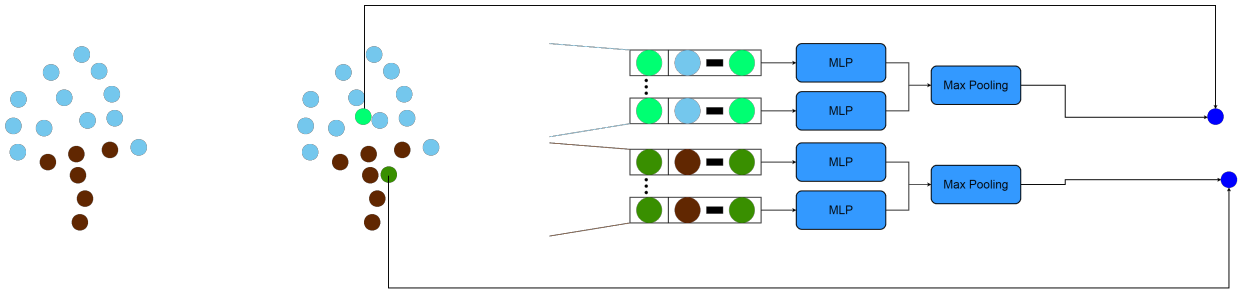

FIGURE 6 – Schéma du calcul de la couche de *graph-convolution*. Les *clusters* sont passés en entrée afin de calculer la partie géométrique du résultat, puis des perceptrons multi-couches sont suivi d'une agrégation maximale sont appliqués dans chaque *cluster* afin d'obtenir la partie caractéristique du résultat.

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

TABLE 1 – Bilan de la répartition des objets dans le jeu de données utilisé.

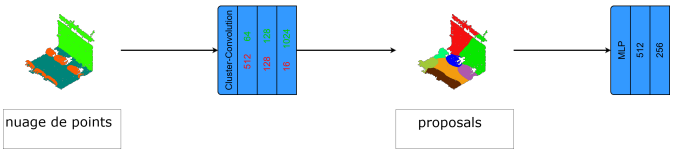

FIGURE 7 – Architecture du modèle utilisé pour nos expériences, avec en rouge le nombre de *clusters* et en vert le nombre de caractéristiques extraites pour chaque application de *cluster-convolution*.

	Moyenne	Signalisation/Poteau	Voiture	Arbre
Segmentation sémantique + *clustering* hiérarchique	**0.29**	**0.11**	**0.64**	**0.11**
RPC (notre architecture)	0.21	0	0.52	0.10
VoteNet	0.04	0	0.09	0.03

TABLE 2 – Résultats, en termes de précision moyenne (*average precision AP*), sur la scène de validation.

4.1 Jeu de données utilisé

La segmentation sémantique dans les nuages de points en extérieur est une tâche bien étudiée avec plusieurs jeux de données disponibles [18, 3, 9]. Cependant il n'en est pas de même pour la détection d'objets où il n'existe, à notre connaissance, pas de jeux de données publics, mis à part ceux orientés voitures autonomes qui ont leurs propres contraintes [2].

Nous avons alors décidé de nous baser sur le jeu de données *ParisLille* [17] car bien qu'initialement développé pour la tâche de segmentation sémantique, celui-ci possède des annotations au niveau instance. Nous utilisons alors une méthode semi-automatique afin de générer les boîtes englobantes associés aux objets et de les ajuster autour d'eux.

Le jeu de données qui en résulte est composé de 3 scènes issues d'acquisitions *LiDAR* différentes, dont 2 sont utilisés comme base d'entraînement, *Paris* et *Lille A*, et une pour la validation, *Lille B*. Nous comptons au total 170 objets répartis en 4 classes. Le détail de la répartition des objets est présenté dans le tableau 1.

Comme ces scènes comptent plusieurs millions de points chacune et s'étendent sur plusieurs dizaines de mètres, nous effectuons un pré-traitement afin de pouvoir les passer en entrée de notre architecture. Les nuages de points sont découpés en blocs de dimension égaux et sous-échantillonné à 2048 points.

4.2 Implémentation

Le modèle que nous utilisons est basé sur notre architecture *RPC*, celui-ci se compose de 3 opérations de *cluster-convolution* pour la génération des *régions*, suivi de perceptrons multi-couches pour la prédiction des boîtes englobantes. Les détails du modèle sont présentés dans la figure 7. Le modèle est implémenté à l'aide du *framework Tensorflow*. Afin d'être dérivable, le nombre d'itération est fixé à 10 lors du calcul du *k-means* de la couche de *clustering*. La valeur du seuil τ pour la définition de l'objet majoritaire est fixé à *0,5* et nous prenons $\beta = 10 \times \alpha$ dans le calcul de la fonction de coût. Les exemples de *régions* positives et négatives sont équilibrés lors de l'apprentissage et un algorithme de *non maximum suppression* (*NMS*) est appliqué afin de filtrer les boîtes prédites.

4.3 Résultats

Nous entraînons notre modèle pendant 200 epochs [1] sur notre base d'entraînement et nous évaluons celui-ci sur notre jeu de validation avec la métrique *AP*. Les résultats sont reportés dans le tableau 2 et des visualisations de prédictions du réseau sont illustrées dans les figures 8 à 11.

Comme il n'existe pas de *benchmark* pour la détection d'objets urbains, nous utilisons comme comparatif une méthode en deux étapes : segmentation sémantique suivi de *clustering* adaptatif. Cependant, comme le *clustering* adap-

1. Le temps d'entraînement est de 8 heures environ avec un GPU Titan Maxwell.

tatif est optimisé directement sur le jeu de validation, les résultats sont biaisés en la faveur de la méthode en deux étapes. En ce qui concerne l'évaluation de la méthode *VoteNet* [12], celle-ci étant optimisée pour des scènes en intérieur, nous avons remarqué que le réseau essayait d'apprendre une répartition spatiale des objets dans l'espace, or celle-ci est totalement aléatoire dû à la pré-segmentation en blocs, biaisant ainsi fortement les résultats en défaveur de *VoteNet*. Globalement, les résultats demeurent plutôt bas, témoignant ainsi de la difficulté de la tâche, mis à part pour la classe voiture. Les performances de notre architecture sont proches de celles de la méthode en deux étapes, à part pour la classe signalisation et nous pouvons nous attendre à ce qu'elles les dépassent en s'appuyant sur un jeu d'entraînement plus grand.

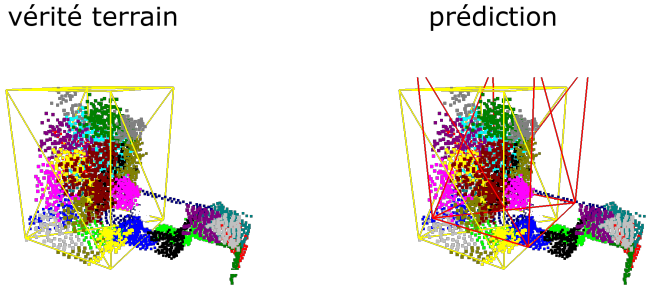

FIGURE 8 – Visualisation d'un exemple positif de prédiction d'un arbre par notre modèle. Chaque couleur correspond à une unique *région*, les boîtes englobantes jaunes sont la vérité terrain, les rouges sont les prédictions du réseau.

Dans la figure 8 nous pouvons voir que bien que l'arbre soit décomposé en plusieurs *régions* différentes, le réseau est tout de même capable de prédire une boîte englobante valide à partir d'une des *régions* située au centre de l'arbre.

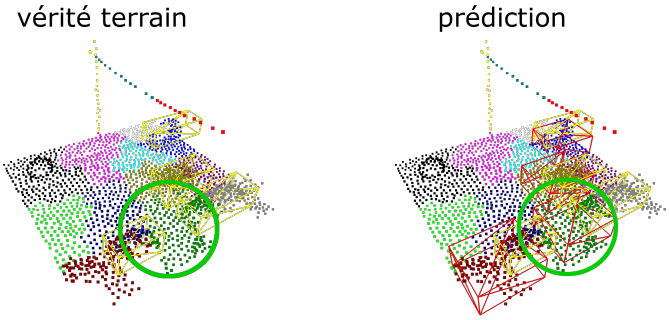

FIGURE 9 – Visualisation d'un exemple de prédiction négative du réseau dû à un chevauchement de 2 voitures par la *région*. Chaque couleur correspond à une unique *région*, les boîtes englobantes jaunes sont la vérité terrain, les rouges sont les prédictions du réseau.

Dans la figure 9, nous pouvons distinguer que la *région* est 'à cheval' sur deux voitures entraînant ainsi une erreur de

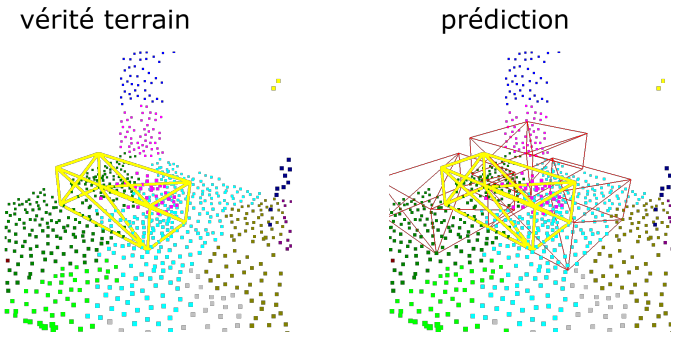

FIGURE 10 – Visualisation d'un exemple de prédiction négative du réseau dû à une voiture, chevauchant plusieurs *régions*. Chaque couleur correspond à une unique *région*, les boîtes englobantes jaunes sont la vérité terrain, les rouges sont les prédictions du réseau.

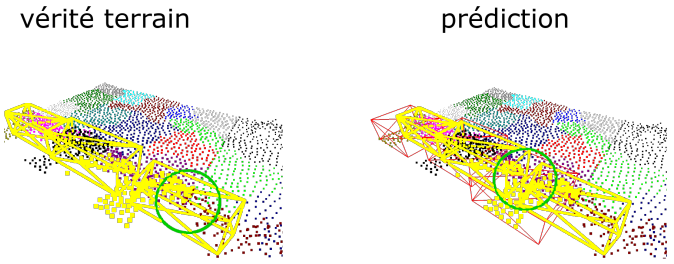

FIGURE 11 – Visualisation d'un exemple de prédiction négative du réseau dû à une voiture chevauchant deux blocs différents. Chaque couleur correspond à une unique *région*, les boîtes englobantes jaunes sont la vérité terrain, les rouges sont les prédictions du réseau.

prédiction du réseau avec une boîte englobante se situant entre les deux voitures. De même dans la figure 10, où cette fois un même objet, une voiture, est recoupé par plusieurs *régions* dont aucune n'est suffisamment proche de la forme de la voiture, impliquant ainsi une erreur de prédiction du réseau avec 3 boîtes englobantes prédites dont aucune ne recoupe correctement la voiture. Un autre type d'erreur est dû à la segmentation de la scène en blocs, ainsi dans la figure 11, un même objet est 'à cheval' sur deux blocs causant une erreur de prédiction avec une boîte englobante décalée par rapport à la voiture.

Nous pouvons alors formuler l'hypothèse suivante : les mauvaises prédictions du réseau sont en grande partie causées par des mauvaises répartitions de *régions* et inversement, qu'à partir d'une bonne répartition des *régions*, les prédictions du réseau ont tendance à être valide.

4.4 Expériences supplémentaires

Afin d'évaluer notre hypothèse nous proposons un indice de qualité des *régions* mesurant à quel point une *région* recoupe un unique objet. Cet indice est défini pour une *région*

R comme ceci :

$$qualite = \frac{Card(O_{maj} \cap R)}{Card(O_{maj} \cup R)}$$

c'est-à-dire la proportion de points appartenant à l'objet majoritaire par rapport au nombre total de points de l'objet majoritaire et de la *région*.
Nous calculons alors cet indicateur pour chaque *région* du réseau sur le jeu de validation et nous le mettons en lien avec la validité de la boîte englobante prédite à partir de cette même *région*. Cette prédiction est évaluée par l'intersection sur l'union (*intersection over union* ou *IoU*) entre la boîte prédite et la boîte vérité terrain de l'objet majoritaire. Les résultats sont reportés dans le diagramme de la figure 12.

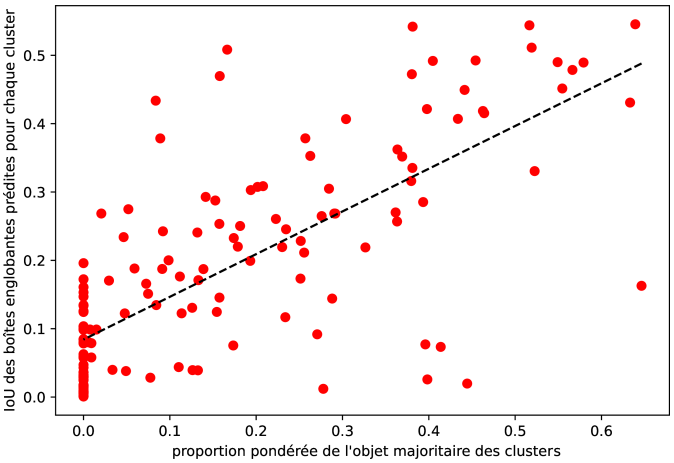

FIGURE 12 – Diagramme de l'intersection sur l'union des prédictions du réseau en fonction de la proportion d'objet majoritaire dans la *région* associée. Chaque point représente une unique *région*.

Nous remarquons alors qu'il existe bel et bien un lien entre ces deux grandeurs, qui peut par exemple être évalué avec un indice de corrélation de Pearson de 0.73, d'autant plus que les points *outliers* correspondent souvent à des erreurs de classification du réseau *i.e.* le réseau a reconnu la présence d'un objet mais pas sa classe.
A partir d'une répartition des *régions* correcte, le réseau a alors tendance à prédire des boîtes englobantes valides. Pour aller plus loin, nous proposons d'étudier plus en profondeur les paramètres qui régissent la répartition des *régions*.

4.4.1 Équilibrage de la fonction de distance

La répartition des *régions* dépend essentiellement de l'application de la couche de *clustering*. L'élément central de la couche de *clustering* est la fonction de distance calculée par l'algorithme du *k-means* (voir section 3 pour plus détails sur celui-ci). L'équilibrage entre partie géométrique et partie caractéristique joue ainsi un rôle crucial dans le calcul de cette distance.
Ce rôle peut être montré en fixant l'initialisation du *k-means* pour un même bloc et en faisant varier uniquement le coefficient λ (plus celui-ci est élevé plus la partie géométrique est importante et inversement). Une visualisation des répartitions ainsi obtenues pour plusieurs valeurs de λ est en figure 13 et l'évolution de l'indice de qualité des 2 *régions* qui recoupent le mieux les 2 voitures est présentée dans le diagramme 14.

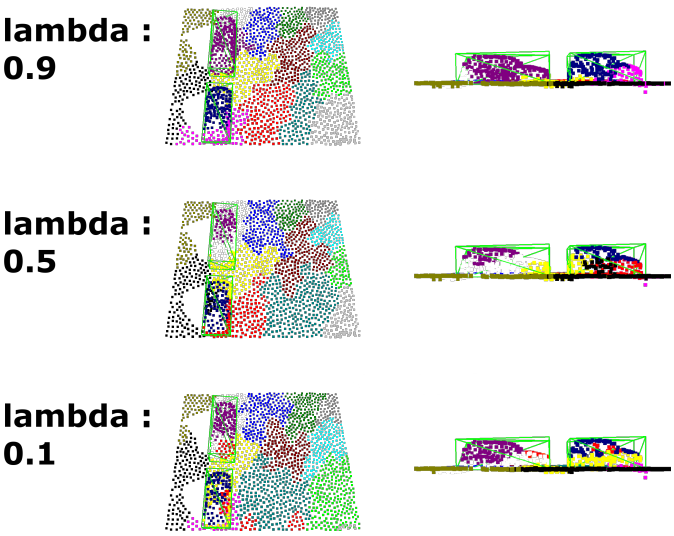

FIGURE 13 – Visualisation de la répartition des *régions* dans un même blocs, selon le coefficient λ de la fonction de distance.

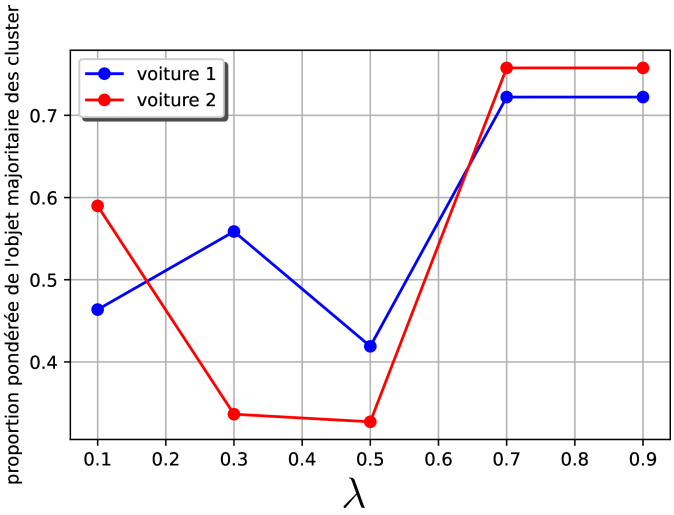

FIGURE 14 – Diagramme de la variation de l'indice de qualité de deux *régions* recoupant une voiture différente chacun, selon la valeur du coefficient λ de la fonction de distance.

Nous remarquons ainsi que cette répartition varie grandement selon la valeur de λ, avec des *régions* de bonne qualité pour des valeurs de λ supérieur à 0,7 et inférieur à 0,2 et de mauvaise qualité entre ces 2 valeurs. Néanmoins, il est difficile de tirer des conclusions globales, en effet les effets de la variation de λ sur la répartition des *régions* est clairement non uniforme, voir chaotique, et varient grandement d'un bloc à l'autre, ou selon la nature des objets présents.

4.4.2 Nombre de *régions*

L'autre paramètre régissant la répartition des *régions* est leur nombre, noté k, c'est-à-dire le nombre de *clusters* calculé lors de la dernière application de *cluster-convolution*. Nous fixons l'initialisation du *k-means* et faisons varier la valeur k. A chaque valeur de k nous calculons la valeur moyenne de qualité des *régions*, reportée dans le diagramme 16, et des exemples de visualisation sont présentés en figure 15.

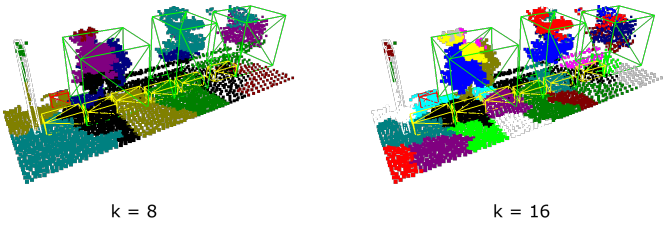

FIGURE 15 – Visualisation de la répartition des *régions* selon la valeur de k.

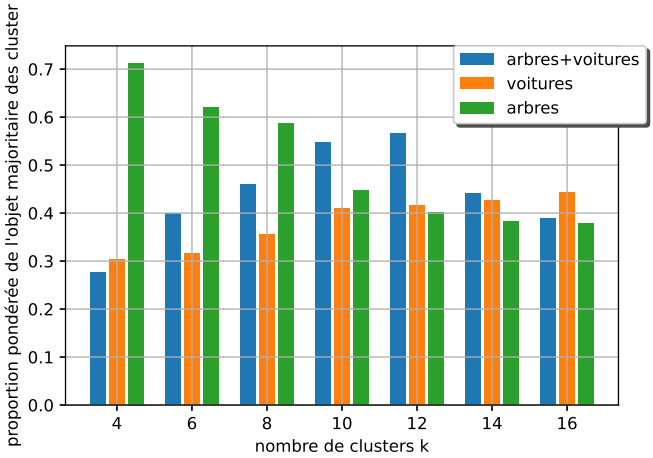

FIGURE 16 – Diagramme de la variation de l'indice de qualité des *régions* selon la valeur de k et la classe des objets présents en majorité dans les blocs.

Comme nous pouvons le constater dans le diagramme, la valeur de k exerce une influence directe sur la qualité des *régions*. Ainsi nous pouvons voir que celle-ci est liée à la taille des objets présents dans les nuages de points. Dans des blocs contenant en majorité des petits objets, comme des voitures, des grandes valeurs de k donnent les meilleurs résultats, et inversement pour les blocs contenant majoritairement des grands objets, comme des arbres, des petites valeurs de k donnent les meilleurs résultats. Il est alors difficile de choisir a priori, une valeur de k qui serait adapté à tous les blocs. C'est pourquoi nous fixons expérimentalement celle-ci à 8.

5 Conclusion

Nous avons présenté nos travaux portant sur la détection d'objets par apprentissage profond, ceux-ci ont abouti au développement d'une nouvelle architecture permettant la détection d'objets urbains dans des nuages de points *LiDAR*. Notre architecture se base sur l'opération originale de *cluster-convolution* permettant de simultanément extraire des caractéristiques à partir des nuages de points et de réduire leurs tailles. Nous avons montré que la répartition des *régions* générées par notre architecture exerce une influence majeure sur ses performances et comment celle-ci repose en partie sur les hyper-paramètres λ et k. L'importance de ces hyper-paramètres constitue une limite de notre architecture, dans la mesure où il serait préférable d'intégrer ceux-ci dans le réseau afin qu'ils soient optimisés lors du processus d'apprentissage. Une autre limite provient de la petite taille de notre jeu de données utilisées lors de nos évaluations expérimentales.

5.1 Perspectives

Afin de poursuivre nos travaux nous comptons utiliser des méthodes d'apprentissage semi-supervisé, comme le *deep-embeding clustering* (*DEC*) dans notre couche de *clustering* au lieu de l'algorithme du *k-means* dans le but de limiter l'influence des hyper-paramètres et d'accroître la robustesse de notre architecture. Nous comptons également adapter des jeux de données plus volumineux à la tâche de détection d'objets urbains voir d'annoter nos propres données *LiDAR* pour augmenter la portée de nos expériences et tester la généralité de notre approche. Enfin, les approches récentes par mécanismes d'attention pourraient être facilement adapté au problème de la détection d'objets urbains et ainsi être comparé à notre approche.

Références

[1] Alexandre Boulch. ConvPoint : Continuous convolutions for point cloud processing. *Computers and Graphics (Pergamon)*, 88 :24–34, 2020.

[2] 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.

[3] 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.

[4] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net : Efficient semantic segmentation of large-scale point clouds. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11105–11114, 2020.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Jerôme Pasquet. *Modélisation, détection et classification d'objets urbains à partir d'images photographiques aériennes*. PhD thesis, Université de Montpellier, 2016.

[11] Lionel Pibre. *Localisation d'objets urbains à partir de sources multiples dont des images aériennes*. PhD thesis, Université de Montpellier, 2018.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] Yan Yan, Yuxing Mao, and Bo Li. Second : Sparsely embedded convolutional detection. *Sensors (Switzerland)*, 18(10), 2018.

[25] Jesus Zarzar, Silvio Giancola, and Bernard Ghanem. PointRGCN : Graph Convolution Networks for 3D Vehicles Detection Refinement. *ArXiv*, abs/1911.1, 2019.

[26] 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.